Automotive System Design
System and Software Architecture

Hochschule Esslingen
University of Applied Sciences

Chapter 1 Overview
Chapter 2 Automotive Electrical & Electronic (E/E) Architectures
Chapter 3 Networks: Bus Systems and Communication Protocols
Chapter 4 System Safety and Reliability

Student Presentations will cover further topics

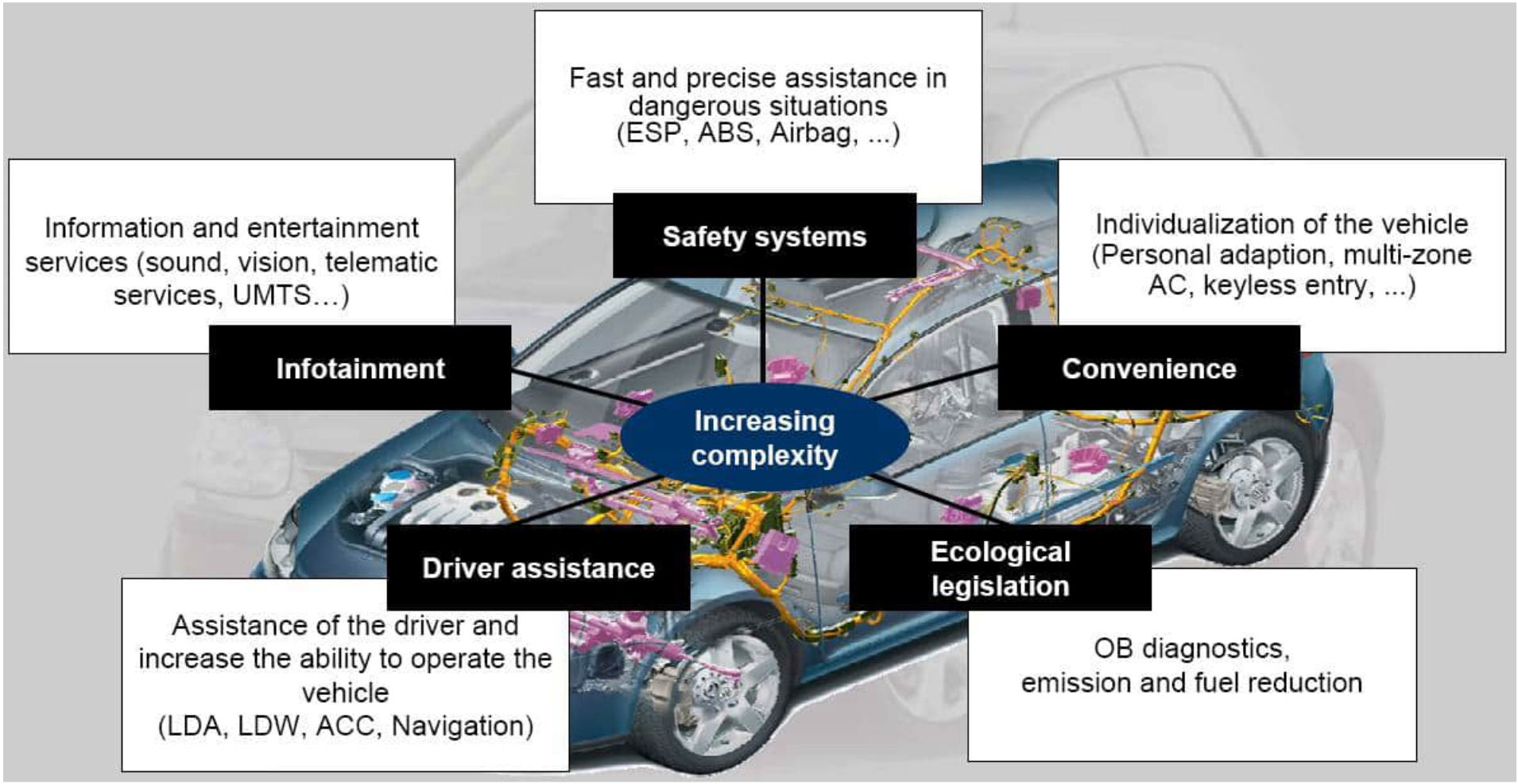
Additional Information

- Diagnostics and Calibration
- Electronic Control Units – Hardware and Software
- Advanced Mobility
- Literature recommendations at the end of each chapter
- Documentation package

Prof. Dr.-Ing. Werner Zimmermann, Hochschule Esslingen – Faculty of Information Technology
Web page: https://www.hs-esslingen.de/en/staff/werner-zimmermann/
http://WernerZimmermann.name

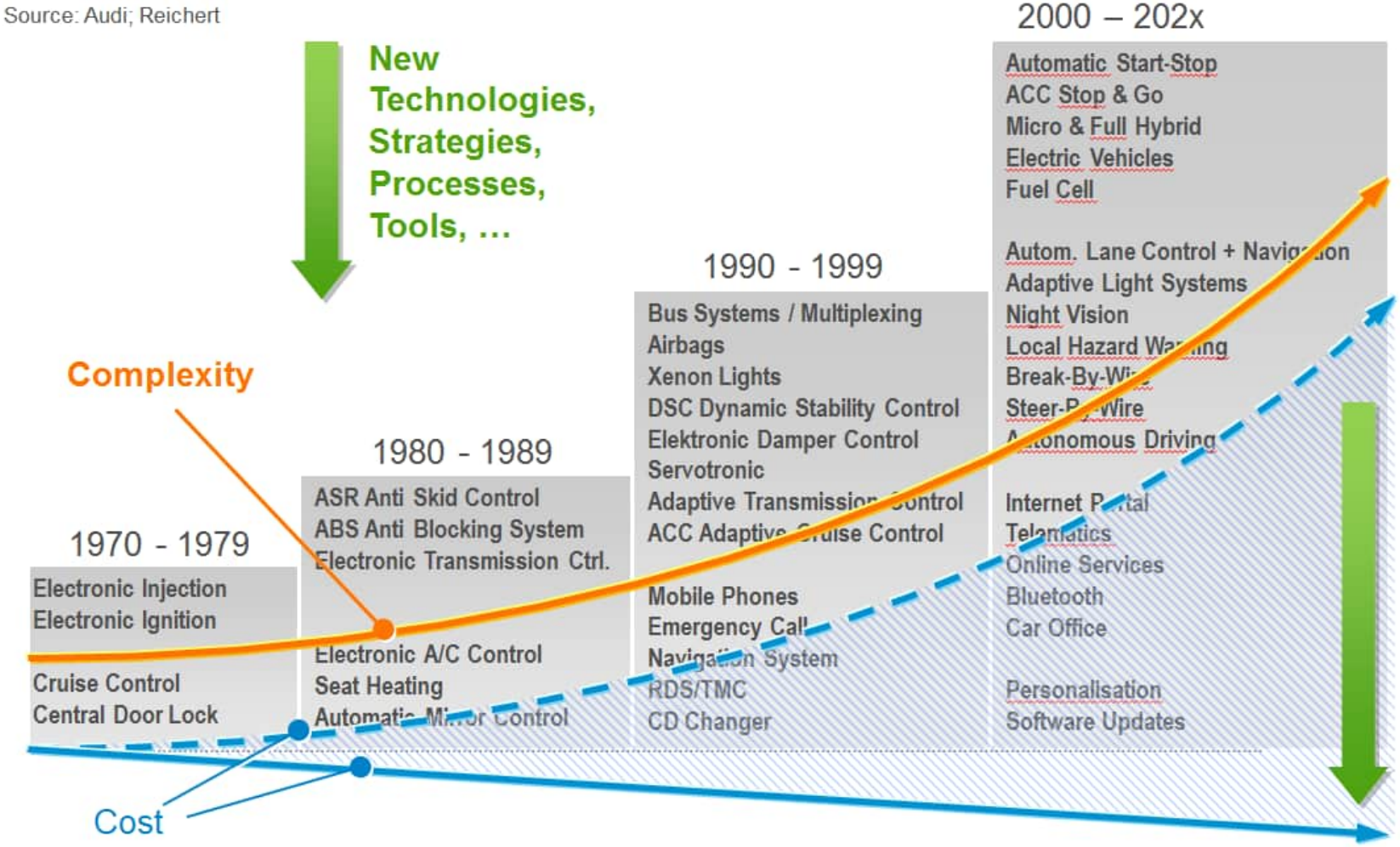
Material for this lecture: Moodle course ASD-ASM-AIM
https://moodle.hs-esslingen.de/moodle/course/view.php?id=29313

Automotive Innovations driven by an intelligent combination of Mechanics, Electronics Hardware and Software = Mechatronics



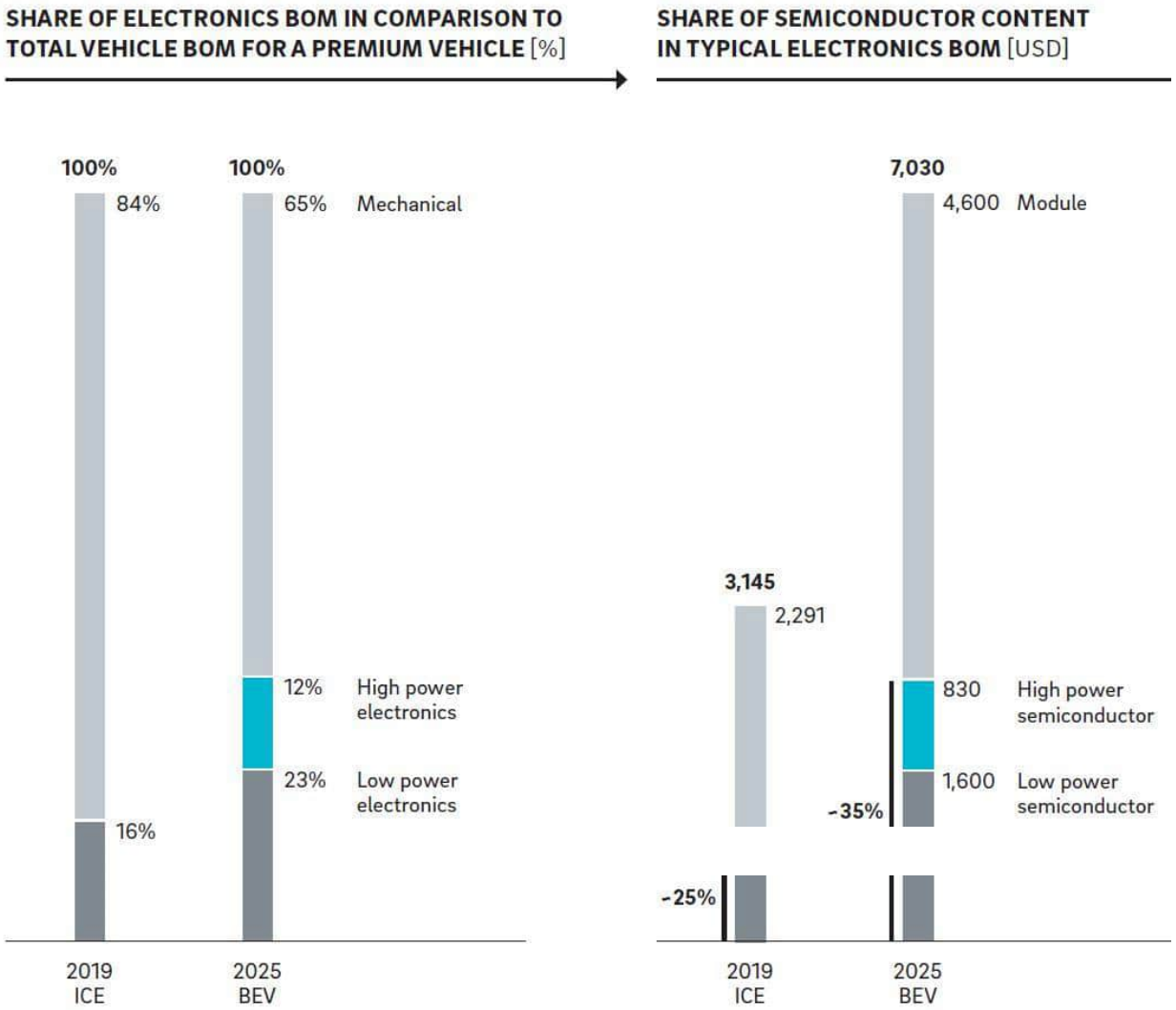
Source: K. Lange, Volkswagen

Innovations spread from luxury segment over midsize cars down to compact class
Complexity goes up, what about cost and reliability?



Source: Dr. Supke, Emotive

Innovations are driven by electric/electronic components and do change the cost structure of cars



BOM … Bill of Materials (Parts Cost)

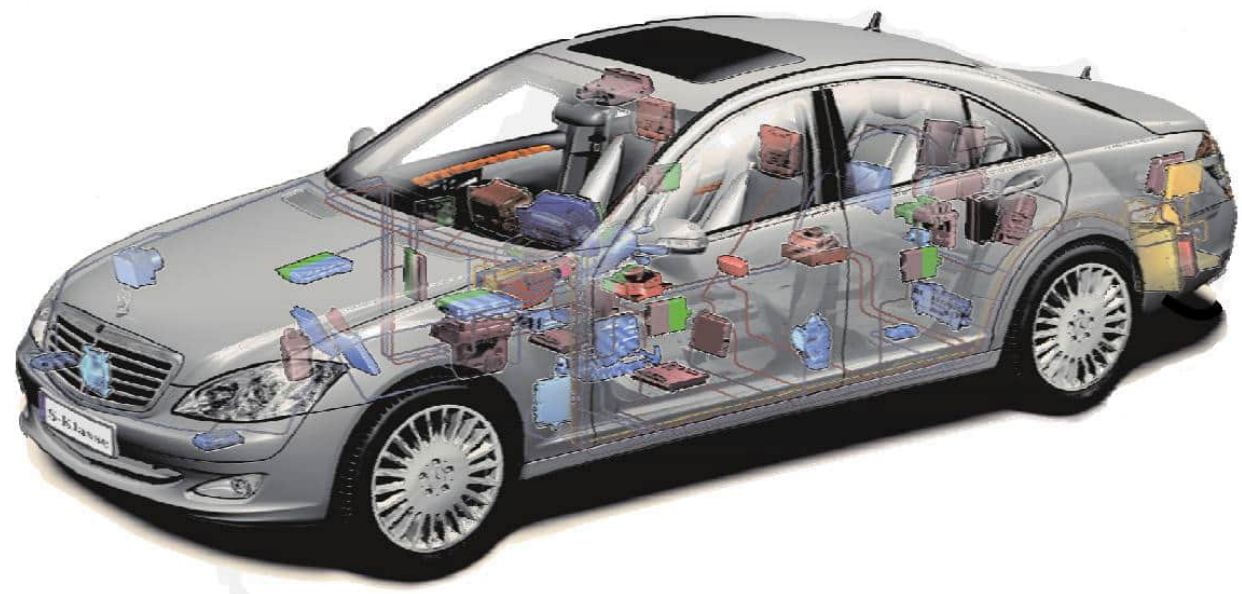
ICE … Internal Combustion Engine

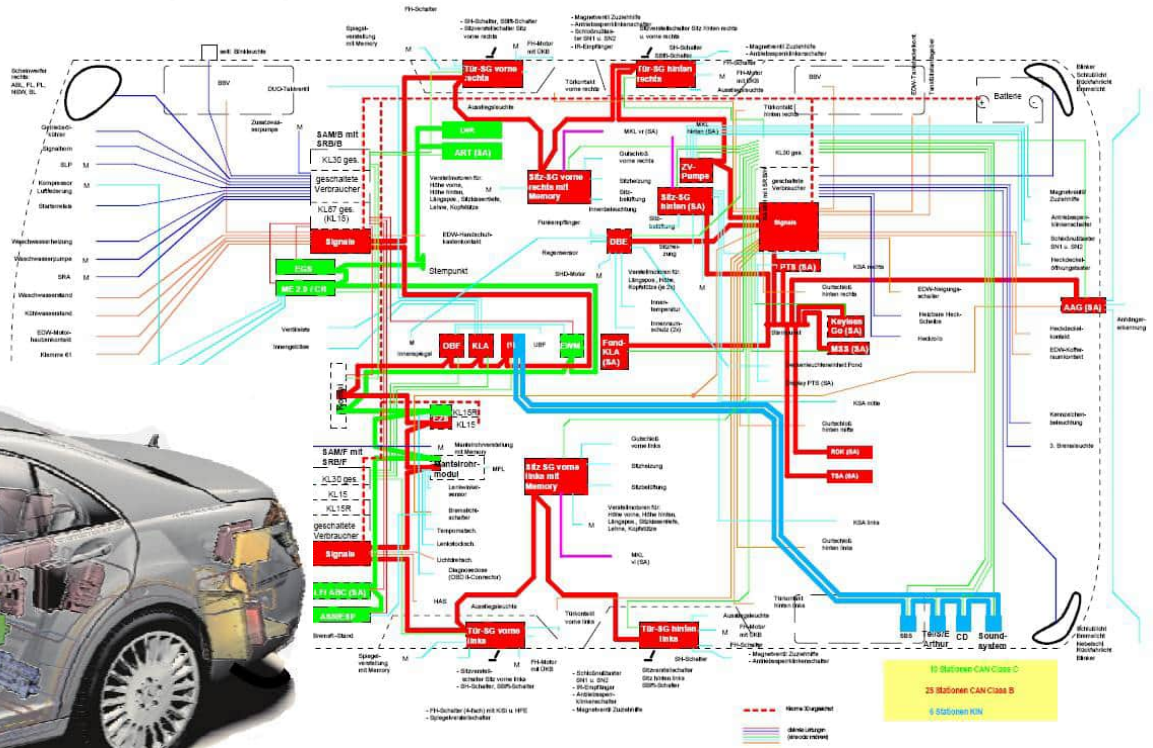
BEV … Battery Electric Vehicle

Source:
RolandBerger.com
Computers On Wheels. 2020

Automotive E/E systems are a combination of

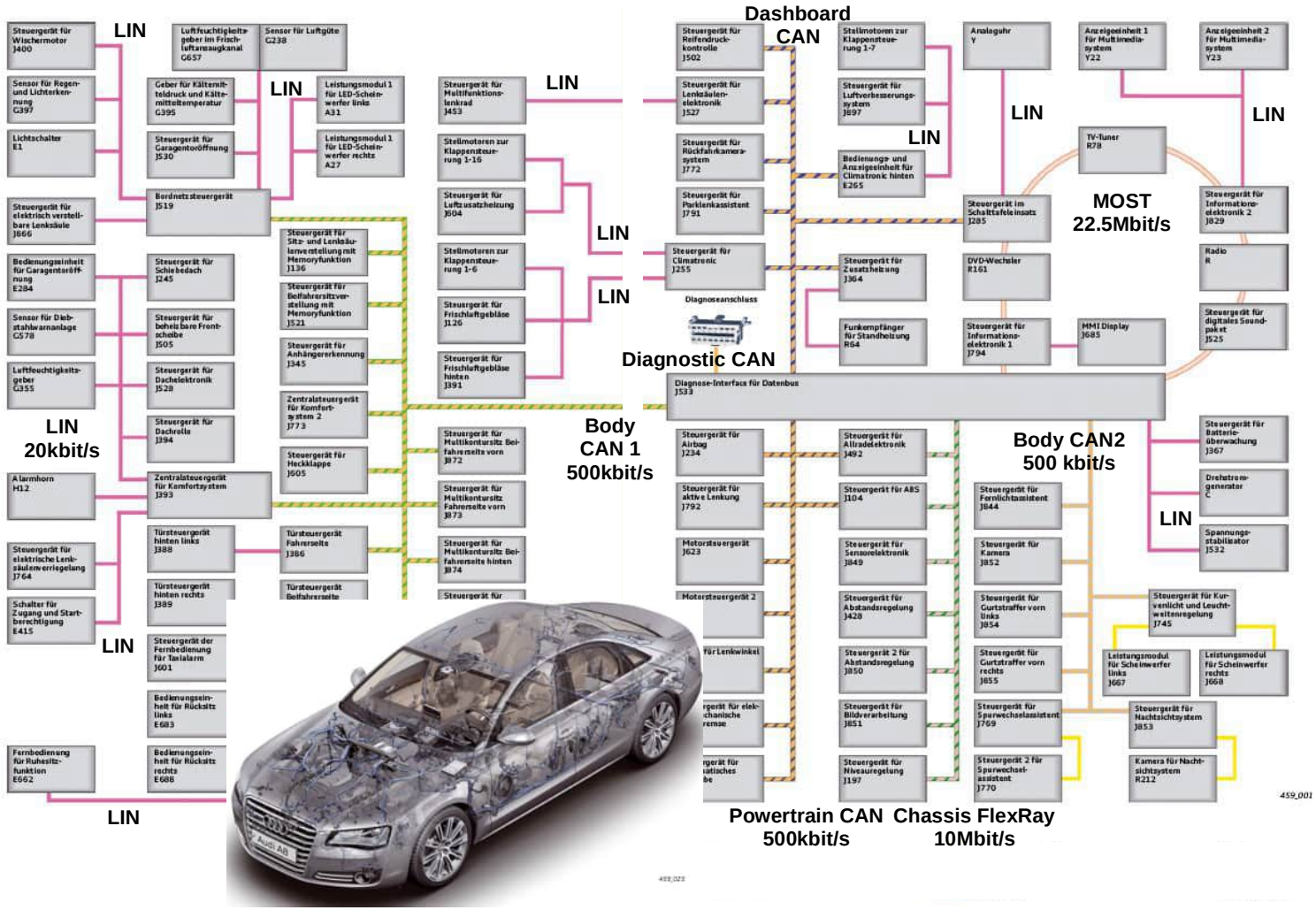
- intelligent **sensors**
- smart **actuators** and a
- **network** of **electronic control units** (ECUs)
  - 20 ... 100+ ECUs
  - 2 ... 9+ busses





Mercedes-Benz S-Class (Luxury Class)
Source: Dr. Hofmann, Daimler

. . . with an uncountable number of variants



Audi A8.
(Luxury Class)
ECU and Bus
Structure
Source: Audi

- 60 standard and 40+ optional ECUs, options available in (nearly) any combination
- 8 bus systems plus some subbusses in several different technologies

Today's Big Automotive Trends

Electrification: From ICE (Internal Combustion Engine) via HEV (Hybrid) to EV (Electric Vehicle)



Source: Google image search



Source: Toyota



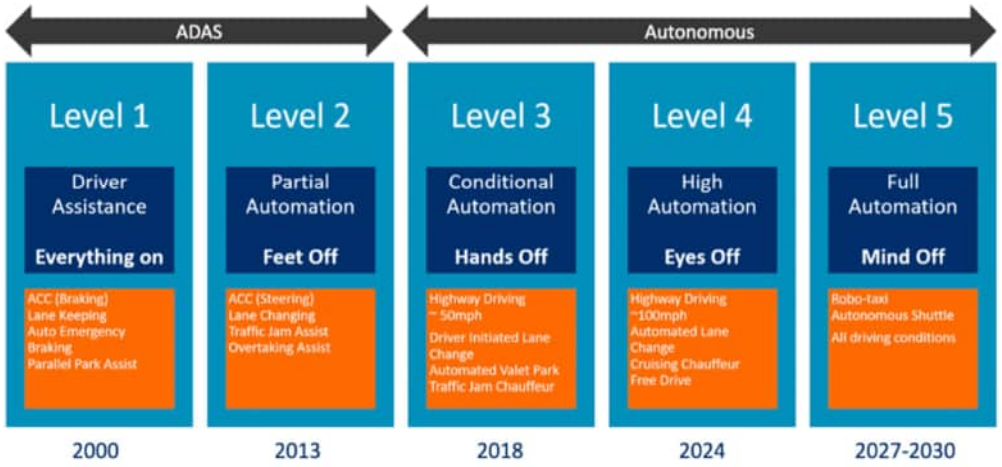
Source: Tesla

Unsolved Problems:

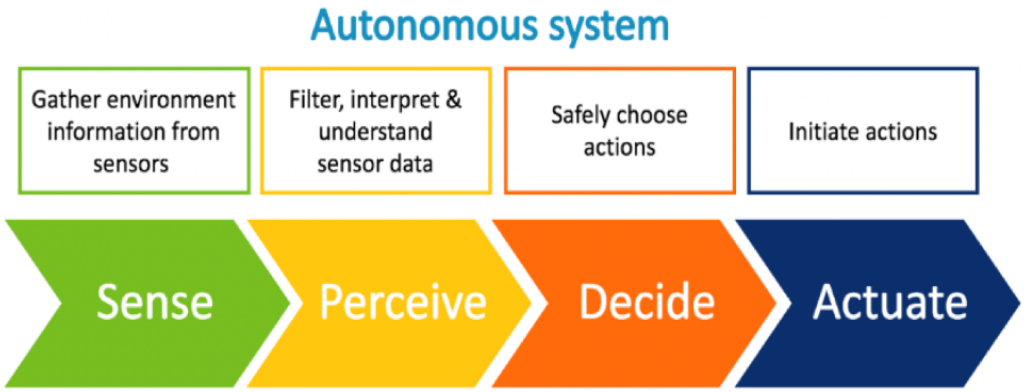
- Batteries: energy density/weight - chemistry – endurance – recycling, similar for Fuel Cells
- Clean electric energy generation – energy distribution – charger infrastructure
- Competitive charging times (hours vs. minutes)

Today's Big Automotive Trends (continued)

Automated Driving: From Advanced Driver Assistance (ADAS) to Autonomous Driving



Source: SAE J3016 Levels of Driving Automation



Source: ARM

Unsolved Problems:

- Sensing:
  Mother nature's sensors (eyes, ears) vs. cameras, radar and ultrasonic sensors
- Perception and Decision Making:
  Human's slow but smart brains vs. computer algos (Artifical Intelligence)
- Safety:
  Human errors ("blame the driver") vs. producer liability ("blame the OEM")

Problems of E/E automotive systems

- **Simultaneous engineering** in different technical disciplines: **Mechanical engineering** (chassis, body, engine, …), **electrical engineering** (ECU hardware), **computer science** (ECU software) and **physicists/chemical engineers** (sensors, batteries,…)
- Long **automotive product lifecycles**

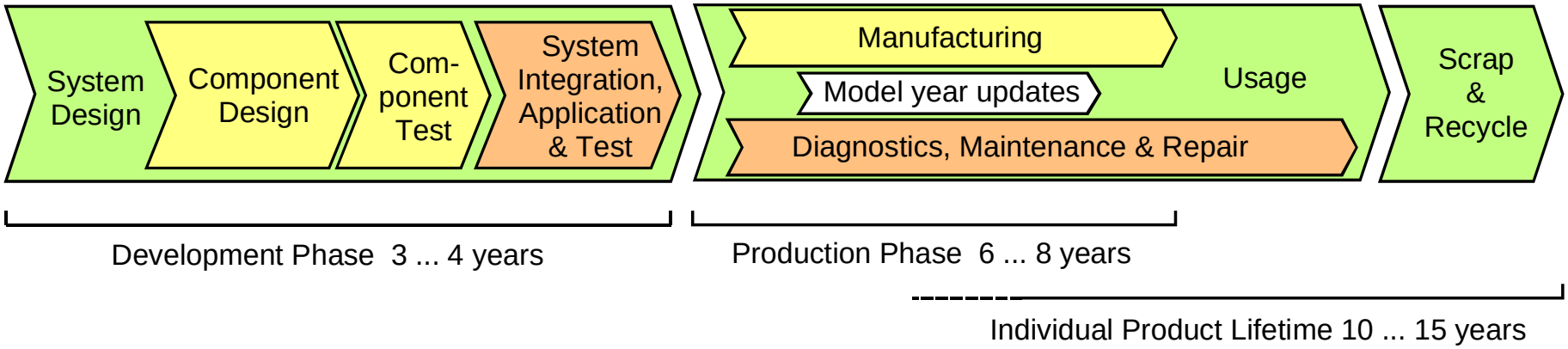
Start of development	–	End of production	**9 … 12 years**
. . .	–	End of use	20 … 25 years

- Use of **consumer electronic devices** and tools due to cost reasons, however typical lifecycle of consumer electronic products

Development phase	0.5 … 2 years
Production phase	**1 … 5 years**

→ *How to ensure long-term quality & deliverability with increasing HW & SW complexity?*

Automotive Product Lifecycle



Prerequisites for this lecture

I do assume, that students have a solid foundation in mathematics and physics and have a basic knowledge of the

- **fundamentals of automotive engineering**, especially combustion engines (working principle, emissions) and vehicle dynamics (kinetics, steering system, brakes), and of the
- **fundamentals of computer hardware and software** (bits and bytes, microprocessors and memory, programming languages, operating systems)

As a refresher and reference the following 2 books are recommended (aka required!) reading:

Konrad Reif (Ed.): Fundamentals of Automotive and Engine Technology. Springer Vieweg, 2014
https://link.springer.com/book/10.1007%2F978-3-658-03972-1

Robert Bosch (Ed.): Automotive Electrics and Automotive Electronics. Springer Vieweg, 2014
https://link.springer.com/book/10.1007%2F978-3-658-01784-2

Both books are available as E-Books to download from the HE library:
https://bsz.ibs-bw.de/aDISWeb/app?service=direct/0/Home/$DirectLink&sp=SOPAC07

If the direct links above don't work for you, use the library catalog to search for the book titles, click on the search results and then on the link "Zum Online Dokument" at the bottom of the respective page.

To download the books, you must **login** into the university network directly or via **via OpenVPN**.

Chapter 2
Automotive Systems E/E-Architecture

Hochschule Esslingen
University of Applied Sciences

2.1 E/E Architectures Overview 2
2.2 Application Domains 5
2.3 State of the Art: Distributed E/E-Architecture 6
2.4 E/E-Architecture Development Trends 9
2.5 Advanced E/E: Domain-centralized Controllers, Zonal Architecture 10
2.6 Electronic Control Units ECUs 14
2.7 Typical Electronic Controls Systems 17

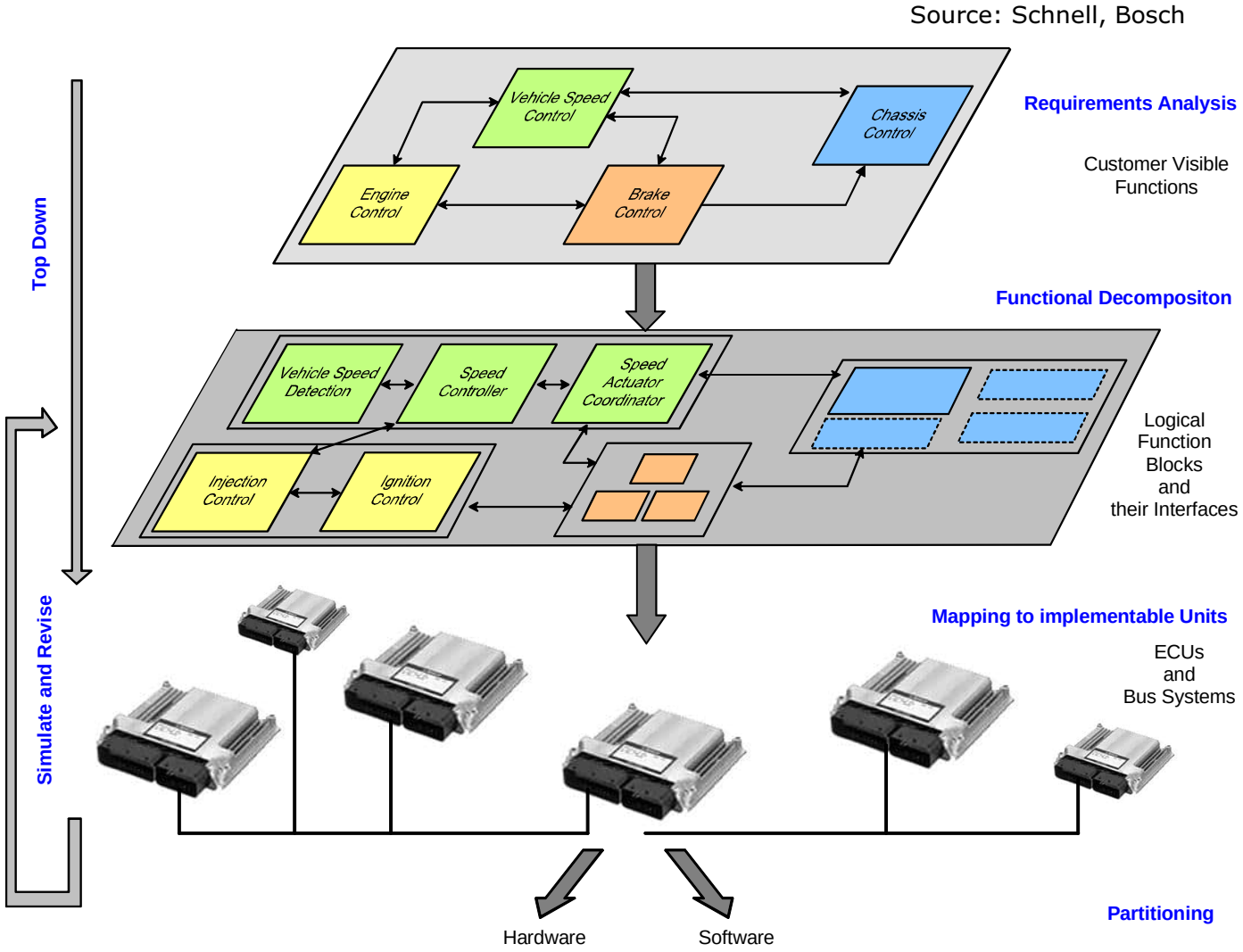
Appendix: Old NEFZ Driving Cycle for Emission Measurement 24
Appendix: New WLTP and RDE Tests for Emission Measurement 25
Further Reading 26

2.1 E/E Architectures Overview

Development

of an E/E architecture

- Identify **customer visible features**: What does the customer want/need?
- Analyse and engineer the **required functions**
- Break them into implementable **hardware and software components**



E/E architecture design shall transform user requirements into a structured, cost-effective **hardware and software implementation**

Electrical and Electronic (E/E) Architectures define

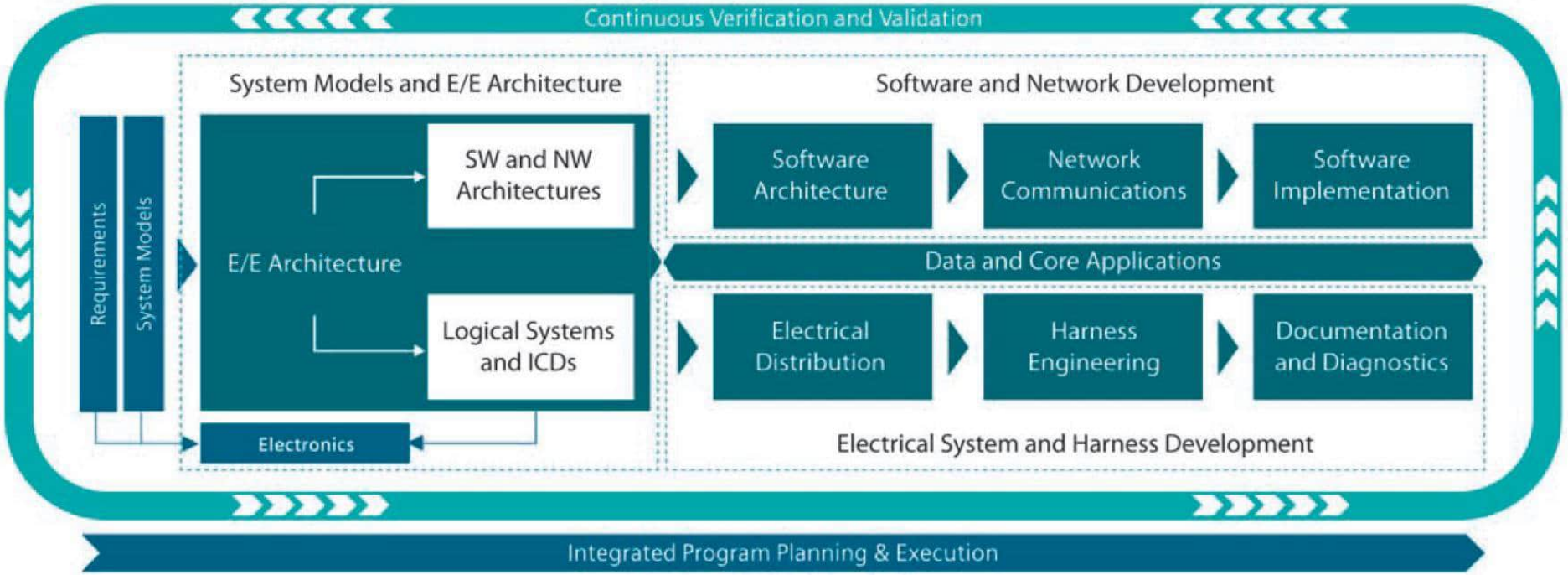
- how many Electronic Control Units (ECUs), sensors and actuators to use
- where to place these components in the vehicle
- which functions will be assigned to these ECUs
- how these functions will be partitioned into hardware and into software modules
- how the ECUs do communicate with each other: **Communication Network** (bus systems)
- how electrical power is distributed and managed: **Electrical Power Network**

. . . under cost, reliability and functional aspects

- **simplify wiring** harness: Major source of weight, manufacturing cost and reliability issues
- **integrate** as many **functions** as possible into one ECU ←→ reduce bus communication
- **mechanically integrate** ECU with sensors or actuators ←→ harsh environment conditions
- **implement** as much as possible **in software** ←→ software quality hard to control
- **customize** functionality: Functions and ECUs must be usable in various combinations (**Composability**) and on various vehicle platforms (**Reuse**) because of customer selectable options and country or model specific requirements

.

Once the E/E architecture has been designed, software and hardware components can be developed in parallel:



Source: Siemens - The Criticality of the automotive E/E architecture. www.siemens.com/ee-systems

2.2 Application Domains

Powertrain

- Engine Management
  Injection, Ignition, EGR, …
- Transmission Control
  Automatic Shifting, Clutch, …
- . . .

Chassis

Active Safety

- ABS Anti-Lock Brake System
- ASR Traction Control
- ESP Stability / Anti-Skid Control
- Suspension / Damper Control
- . . .

Driver Assistance Systems (Cross Domain)

- Brake Assist
- Obstacle Warning / Park Assistant
- ACC Adaptive Cruise Control
- Adaptive Light, Night Vision
- Lane-Departure Warning
- Automated Driving
- . . .

Passive Safety

- Front and Side Airbags

Comfort

- Automatic Climate Control
- Central Lock System
- Xenon Head Lights
- Window Lift, Mirror, Seat Control
- . . .

Body

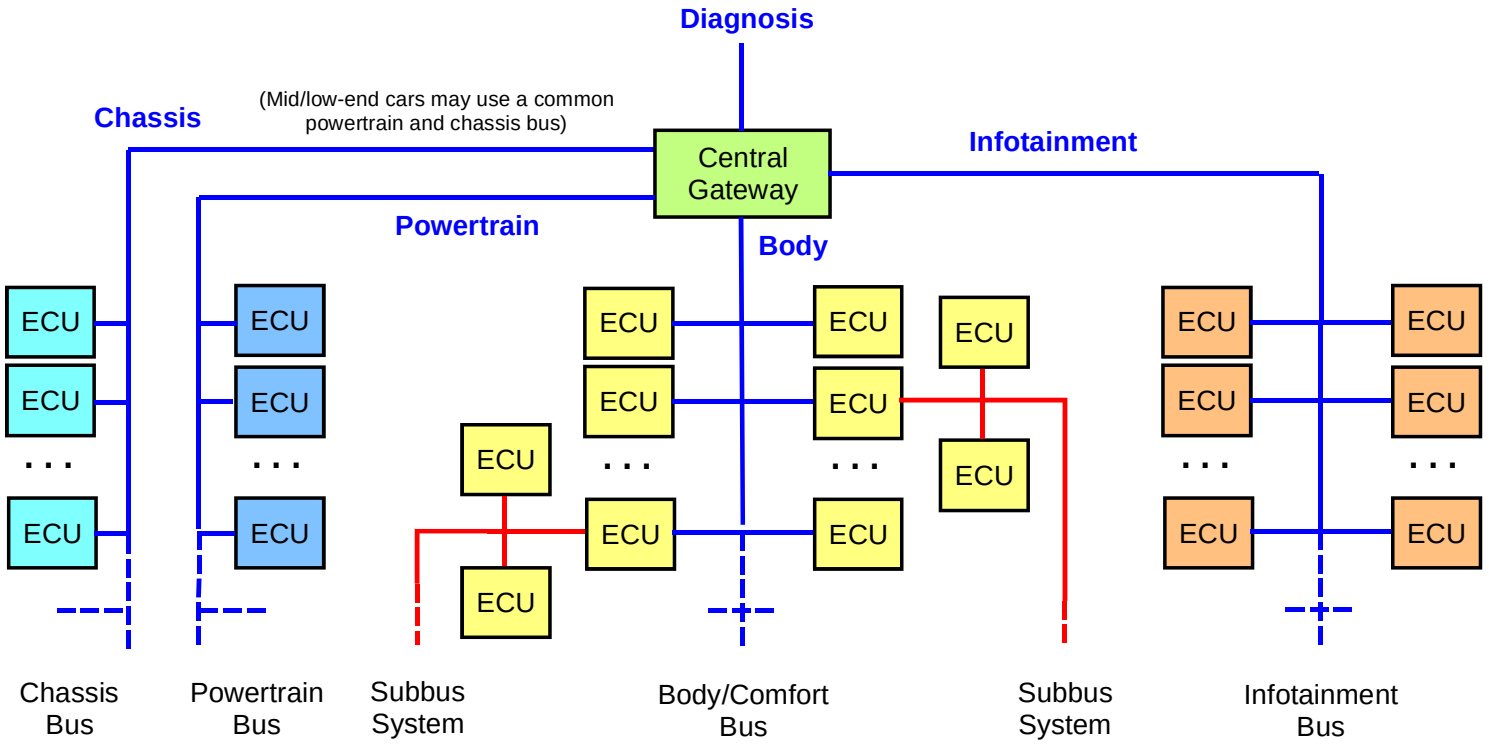
- Radio
- RDS Radio Traffic Information
- Car Telephone
- Navigation
- Television
- Rear and Front Live Video
- . . .

Infotainment

2.3 State of the Art: Distributed E/E-Architecture

Application-domain specific ECU clusters with central gateway ECU

→ Basic idea: Combine ECUs based on functional aspects



Central gateway, 3 … 4 major ECU clusters/busses with several sub-busses:

- Powertrain, Chassis, Body, Infotainment and Diagnosis network
- Reduced wiring with intelligent sensors and smart actuators with integrated bus interface
- Central gateway is a performance bottleneck, high busload within and across clusters

Real World Example





Typical Executive Class Car

Dashboard CAN
Diagnostic CAN 500 kbit/s
Diagnostic IP 100 Mbit/s
MOST Ring 25 Mbit/s
Gateway
Powertrain / Chassis CAN 500 kbit/s
LIN 20 kbit/s
Body CAN 100 kbit/s
LIN 20 kbit/s
LIN 20kbit/s
LIN 20 kbit/s
LIN
LIN
Proprietray
Analog

Engine Mgmt. ECU
Transmission Mgmt. ECU
ABS/ESP
Suspension Control
Airbags
Adaptive Cruise Ctrl.
Headlamp Level Ctrl.
Adaptive Front Light
Electronic Park Brake
Auxiliary Heating
Steering Angle
Steering Wheel
Door Front Left
Door Front Right
Door Rear Left
Door Rear Right
Rain Sensor
Panel left
Panel right
Fan
Window Heater
Air Condition Climate Ctrl.
Park Pilot
Interior Light Rear
Tilt Sensor
Interior Light Front Left
Interior Light Front Right
Garage Door Ctrl.
Energy Management
Rear Climate Ctrl.
Starter Key Door Locks
Keyless Entry
Tire Pressure Monitor
Roof Module
Sun Roof
An-tenna
Head Unit
Radio Tuner
Standard Sound Sys.
Single CD
CD Changer
Bose Sound
Navigation
Cell Phone
Telematics
TV Tuner
Voice Command
MMI Display
Loud-speakers
Microphone
Headset

red: standard - green: optional equipment

Source: Schleuter, Audi (simplified!)

One ECU per subsystem plus a bundle of sensors and actuators

- **bottom-up development** over 30 years with more ECUs added when systems were added
- allows **independent development** and **supplier selection for each system**
- **OEM as systems integrator**, ECU HW/SW **development outsourced to ECU suppliers**

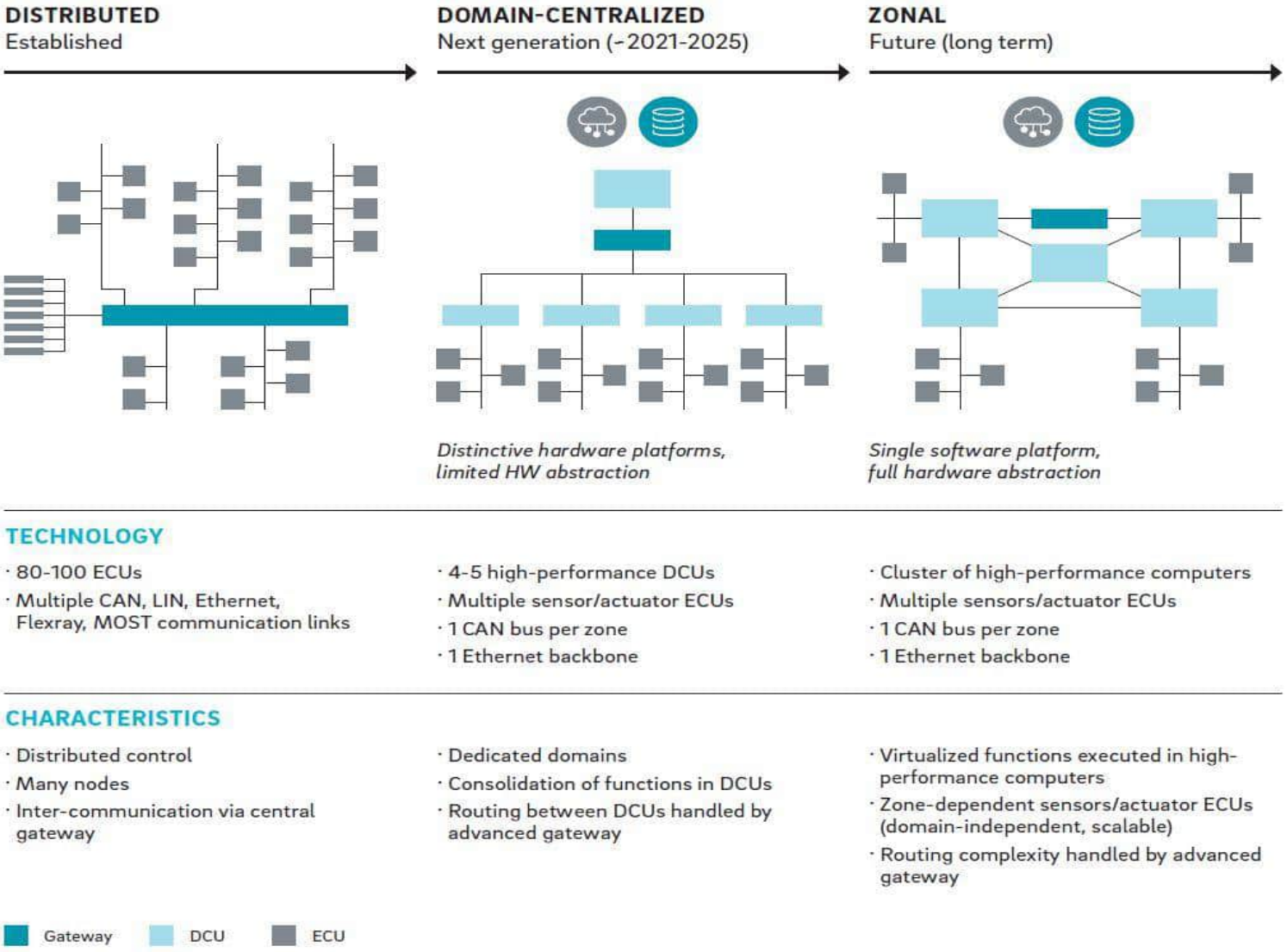
Several **bus systems of different types and bit rate** used, because

- none of today's available automotive bus systems can achieve the overall bus bandwidth and/or latency requirements and cost targets of a modern car with a single bus technology
- electrical specifications (max. length of bus line, max. electrical load) do not allow to connect all ECUs to a single bus system

Several aspects define the structure

- **performance and safety critical ECUs**, which need tight interaction, connected to the same bus, e.g. powertrain bus – chassis bus, compromises due to **wiring restrictions**
- **external accessible diagnosis bus decoupled** from the rest of the bus systems via the gateway ECU due to security reasons
- **low-cost bus systems** (Low-Speed-CAN, LIN) used to reduce cost, where performance is sufficient (hardware cost more important than complexity reduction)
- **special bus systems where required by performance**, e.g. MOST for infotainment, proprietary or analog point to point connections for MMI display or speaker connections
- bus systems coupled via **special gateways or normal ECUs including gateway functionality** (coupling/decoupling of bus systems of different types or bit rates including message filtering, buffering and forwarding, protocol conversion

2.4 E/E-Architecture Development Trends

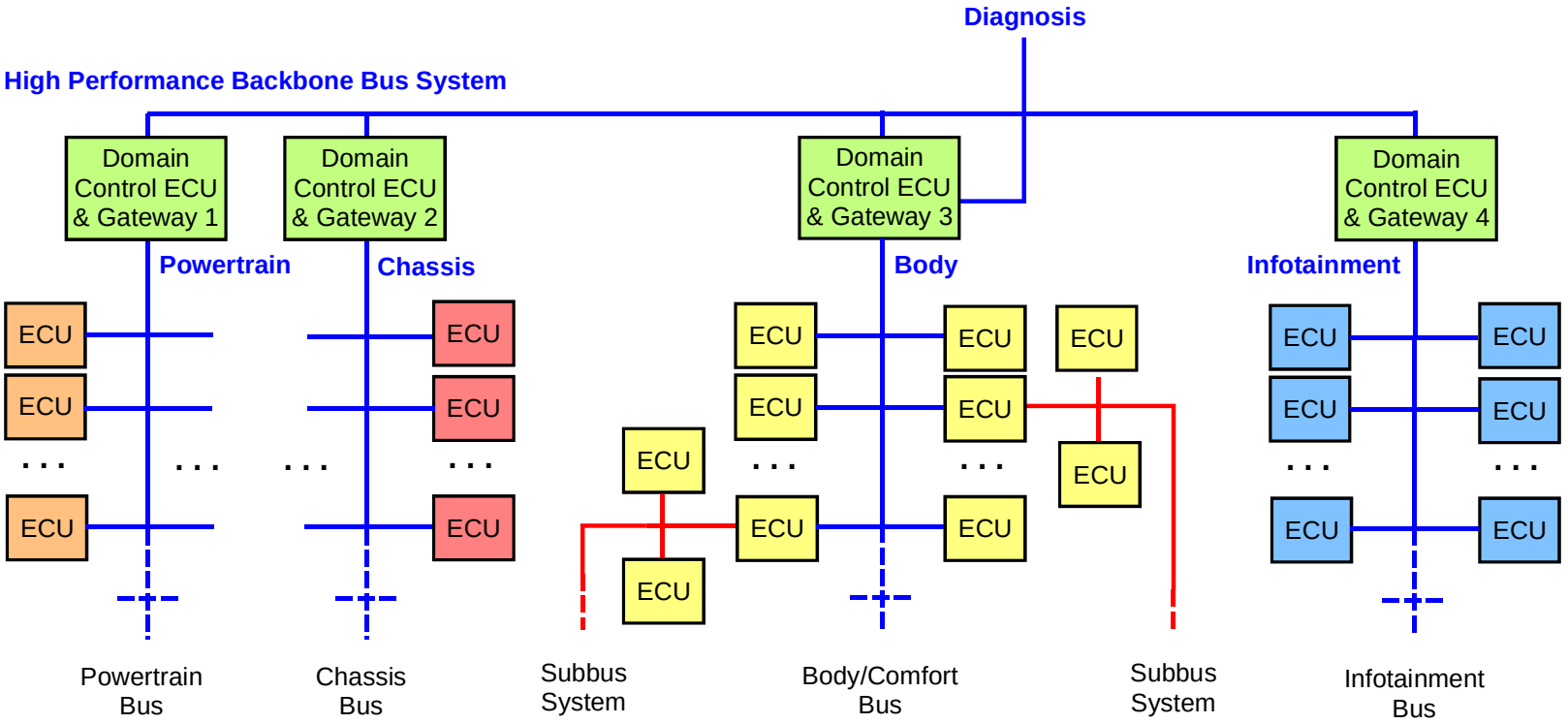


- Ongoing centralization: Less but more powerful ECUs
- Reduce network communication
- Adopt mainstream computer HW/SW technology rather than automotive specific solutions
- Large OEMs found their own software companies to evolve from systems integrator to systems implementer: Software Defined Car

Source:
RolandBerger.com
Computer on Wheels. 2020

2.5.1 Advanced E/E: Domain-centralized Controllers with Backbone Bus

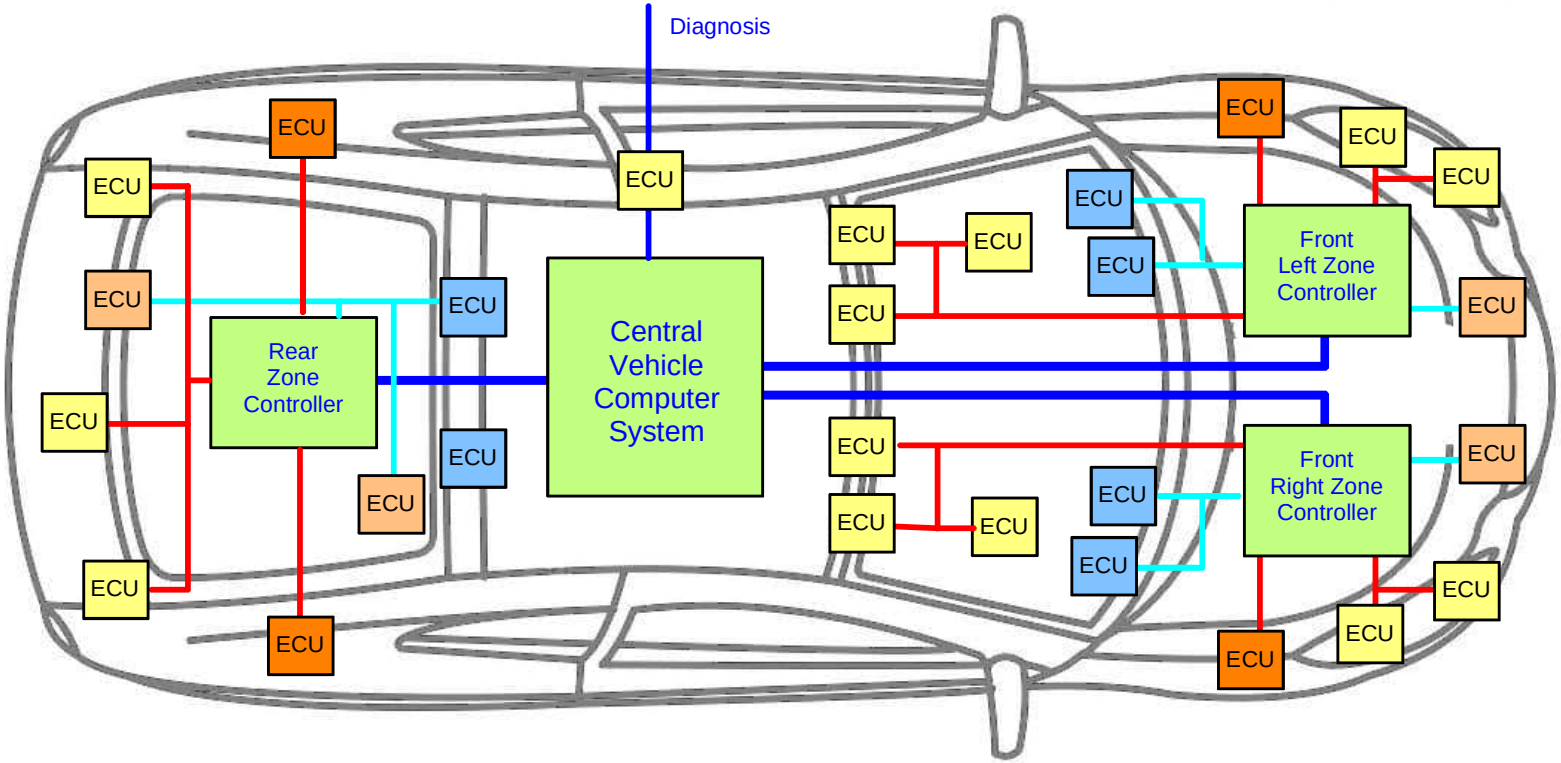
Basic idea: Keep domain-oriented clusters, but move SW functions into powerful domain Control ECUs, many small ECUs as sensor/actuator frontends



- Backbone bus: Higher performance for information distribution between ECU clusters
- Unified technologies: Ethernet for backbone and infotainment, CAN-FD for real-time control
- Advantage: Gradual upgrade from existing architecture
- Disadvantage: Most clusters distributed across the complete vehicle, i.e. complex wiring

2.5.2 Advanced E/E: Zonal Architecture

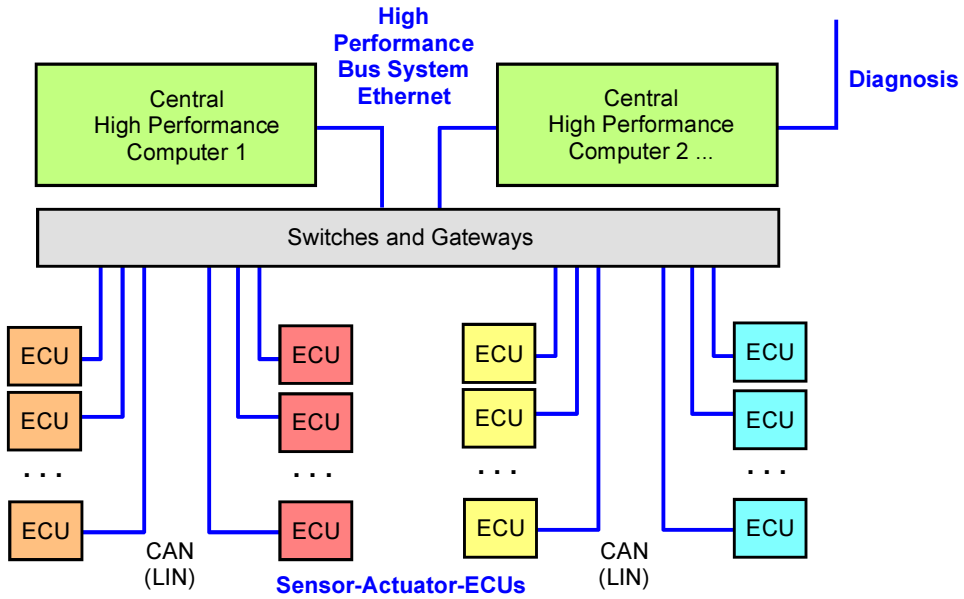
Basic idea: Organize clusters based on geometry, i.e. “zones”, e.g. Front–Central–Rear.
Move most of the SW functions to central vehicle computer (“server”).



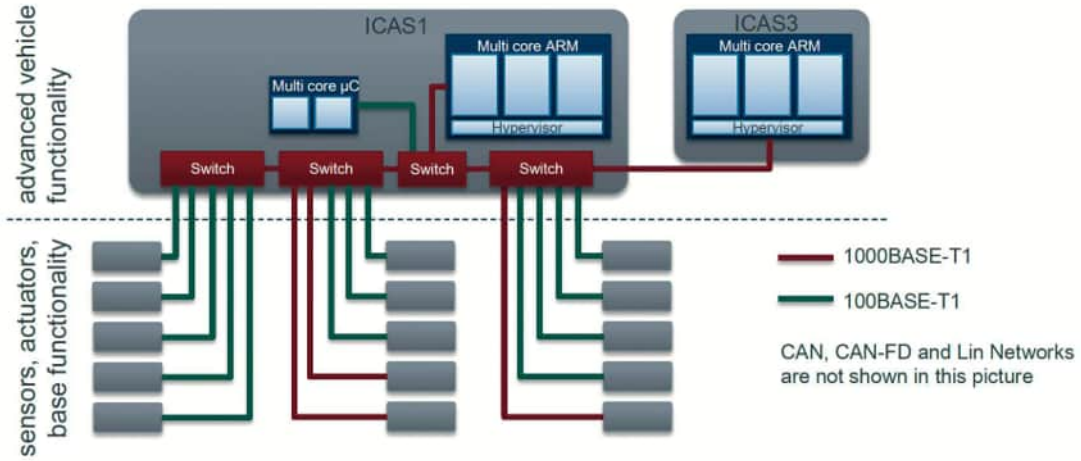
- **Zone controllers** act as concentrators for local **sensor/actuator ECUs** and perform real-time closed loop control tasks. All high level task like automated driving, vehicle dynamics coordination etc. move to a **central high performance computer system**
- Advantage: Shorter wiring, simpler network structure with Ethernet and CAN
  Reduced network communication, simpler interaction of SW functions.

Example: "Tesla-inspired" **In Car Application Server (ICAS)** architecture for VW's new EV platform MEB

- Powerful central **compute server**(s) based on multicore ARM CPUs, extended via Cloud connection. Other OEMs also think about using x86 or NVIDIA CPUs as well. I.e. **PC-like technology** running several general purpose and realtime operating systems, e.g. Posix-based like Linux, Adaptive AUTOSAR, .. in virtual machines VM/Hypervisors, i.e. **isolated software containers**.
- **Microcontroller-based** ECUs with Classic AUTOSAR act as **sensor-actor frontends**
- Automotive **Ethernet** as **high-speed networks** between ICAS computers and to video cameras. Standard automotive networks like **CAN to sensor/actuator ECUs**.



"Tesla-like" architecture

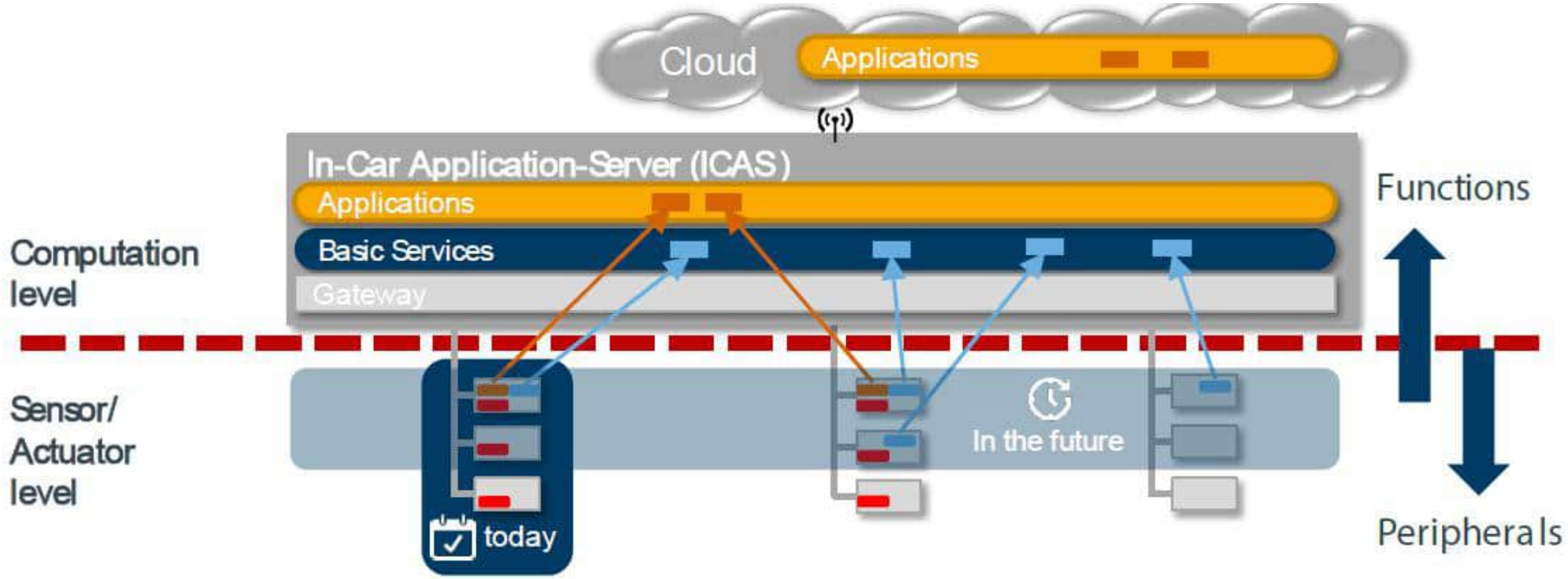


VW ICAS architecture

Source: M. Wille, O. Krieger: Ethernet and Adaptive Autosar. Key elements of the new VW E/E architecture

Example continued:

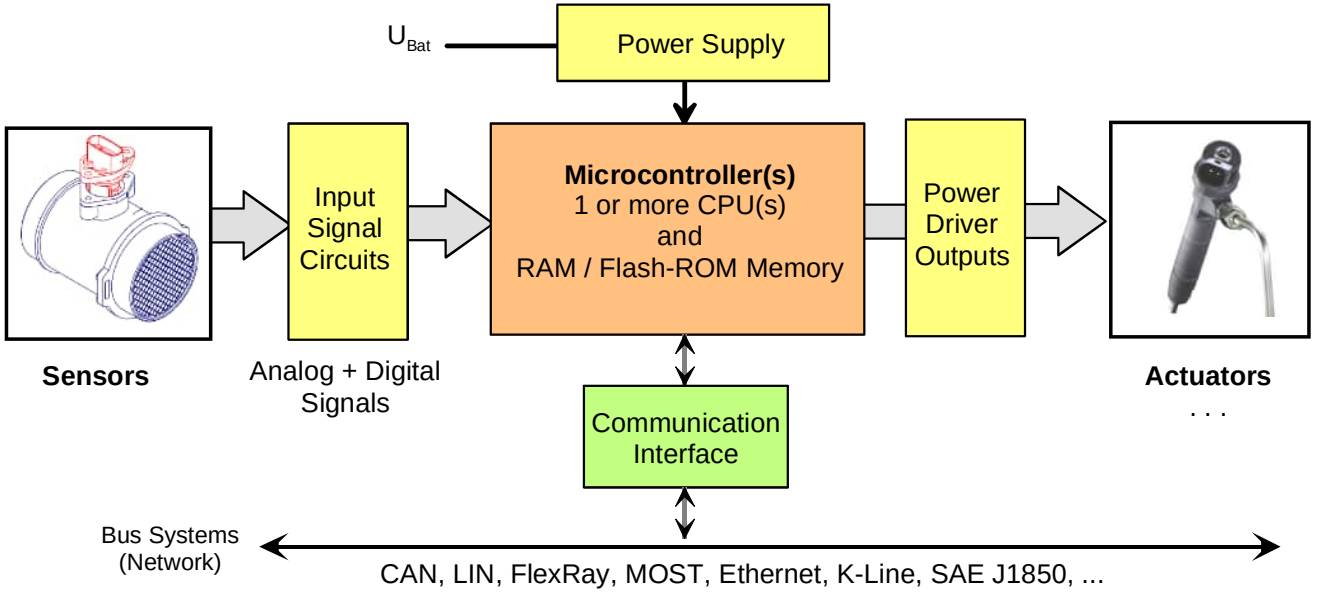
Layered architecture with sensor/actor level, computation level and cloud-based services



Source: M. Wille, O. Krieger: Ethernet and Adaptive Autosar. Key elements of the new VW E/E architecture

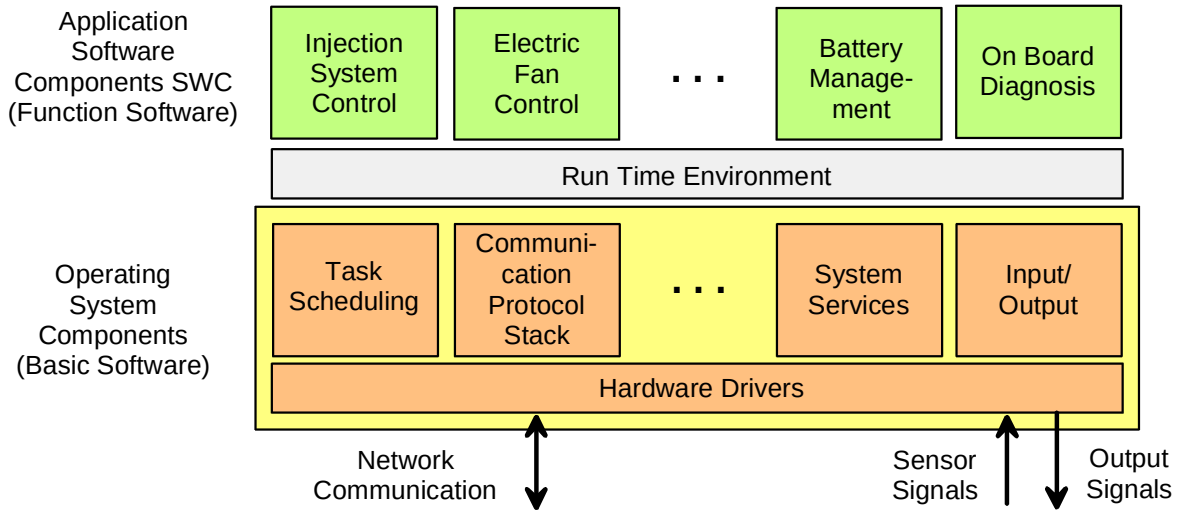
2.6 Electronic Control Units ECUs

ECU Hardware Structure (for more details see appendix Electronic Control Units ECUs)



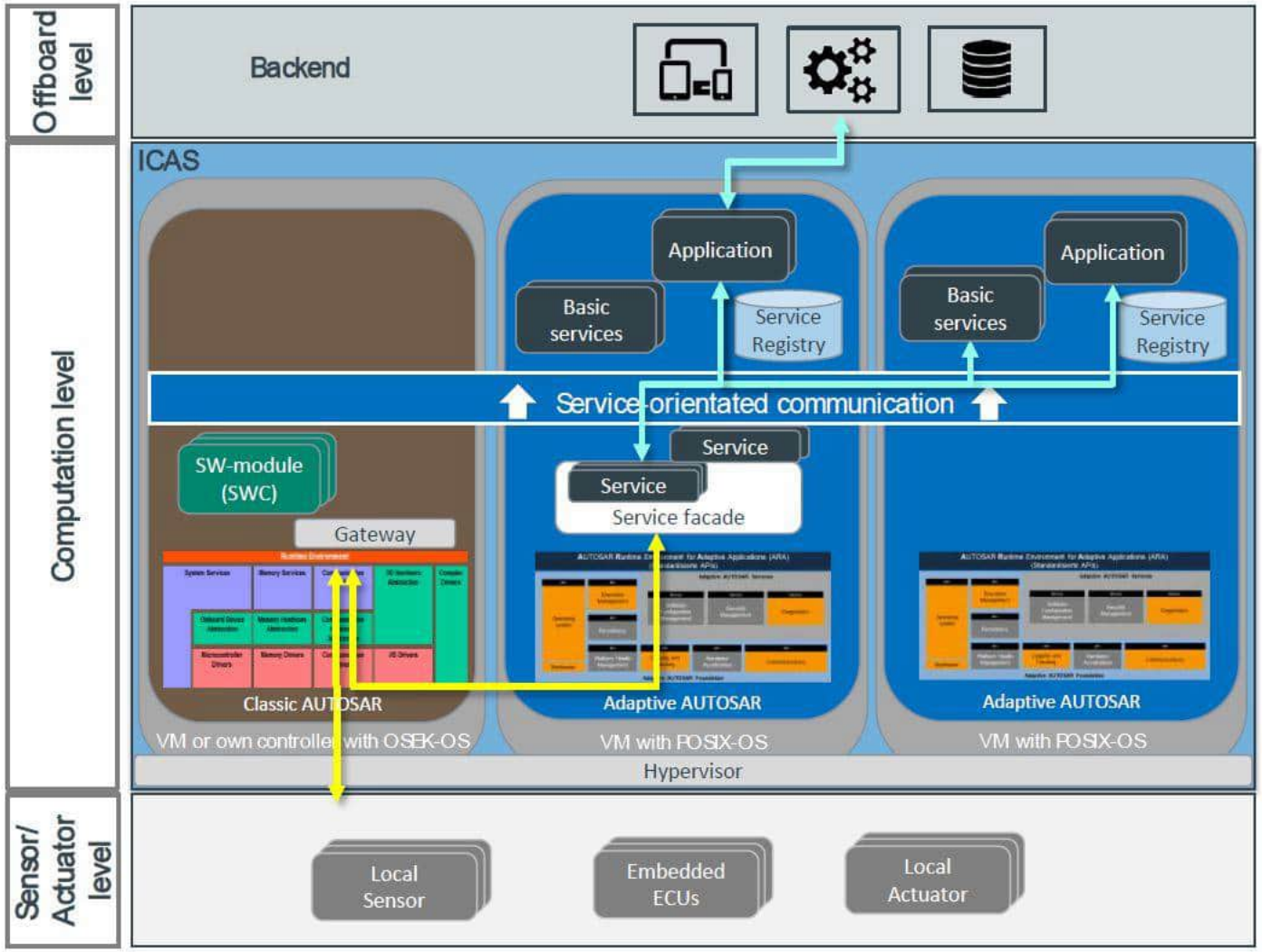
- ECU variants differ in **number of inputs/outputs**, network interfaces, **CPU performance and memory** size **from small microcontroller systems** (e.g. body electronics ECUs < 64kB) **to full-size computers** (e.g. navigation and infotainment headunits > 4GB)
- Standard consumer electronic devices cannot be used due to **harsh environment requirements**: Temperature range -40°C … +85°C … + 125°C, vibration and shocks, sealing dependent on mounting location. Electromagnetic compatibility EMC and large supply voltage range, e.g. passenger car ECUs 5V … 18V … 24V … 32V
- ECUs are **OEM specific developments**, not standard off-the-shelf products

ECU Software Structure (for more details see appendix Electronic Control Units ECUs)



- **Classic business model**: Proprietary **Software** implemented by ECU supplier, **bundled with hardware**. **Larger OEMs** needed to **transfer** their **system know how to ECU suppliers** for software implementation. **ECU suppliers acted as** system developers for smaller OEMs, thus being **enablers for new OEMs in emerging countries**.
- **New** ECU **trend: Standardized AUTOSAR** "operating system" with well-defined application programming interfaces API allows (future) off-the-shelf hardware and "installing" application software components SWC from OEM or several vendors.
  - **AUTOSAR CLASSIC for real time control** ECUs (engine management, ABS/ESC, …) established since mid of the 2000s, **based on** former **OSEK/OS** standard
  - **AUTOSAR ADAPTIVE for** infotainment and **advanced driver assistant systems ADAS** under development since mid of the 2010s, **based on POSIX** standard

Example: SW-architecture of VW's new EV platform

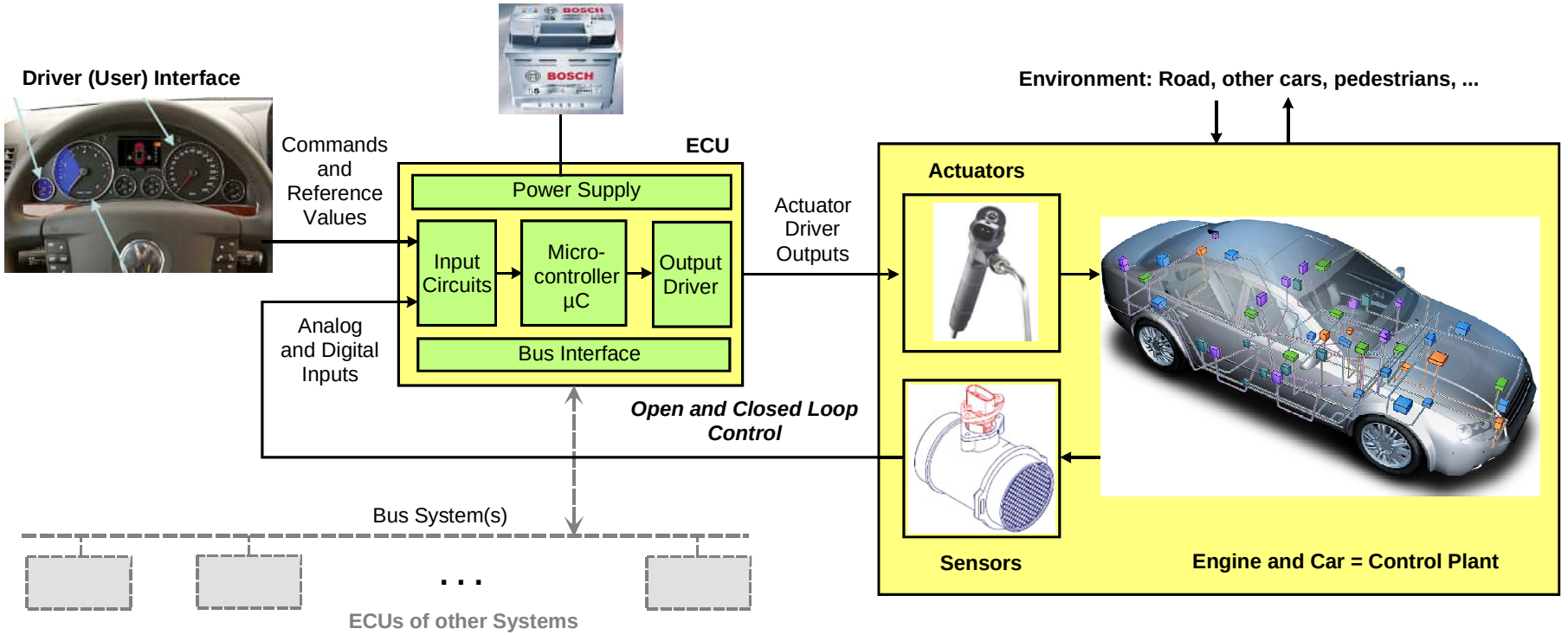


Source: M. Wille, O. Krieger: Ethernet and Adaptive Autosar. Key elements of the new VW E/E architecture

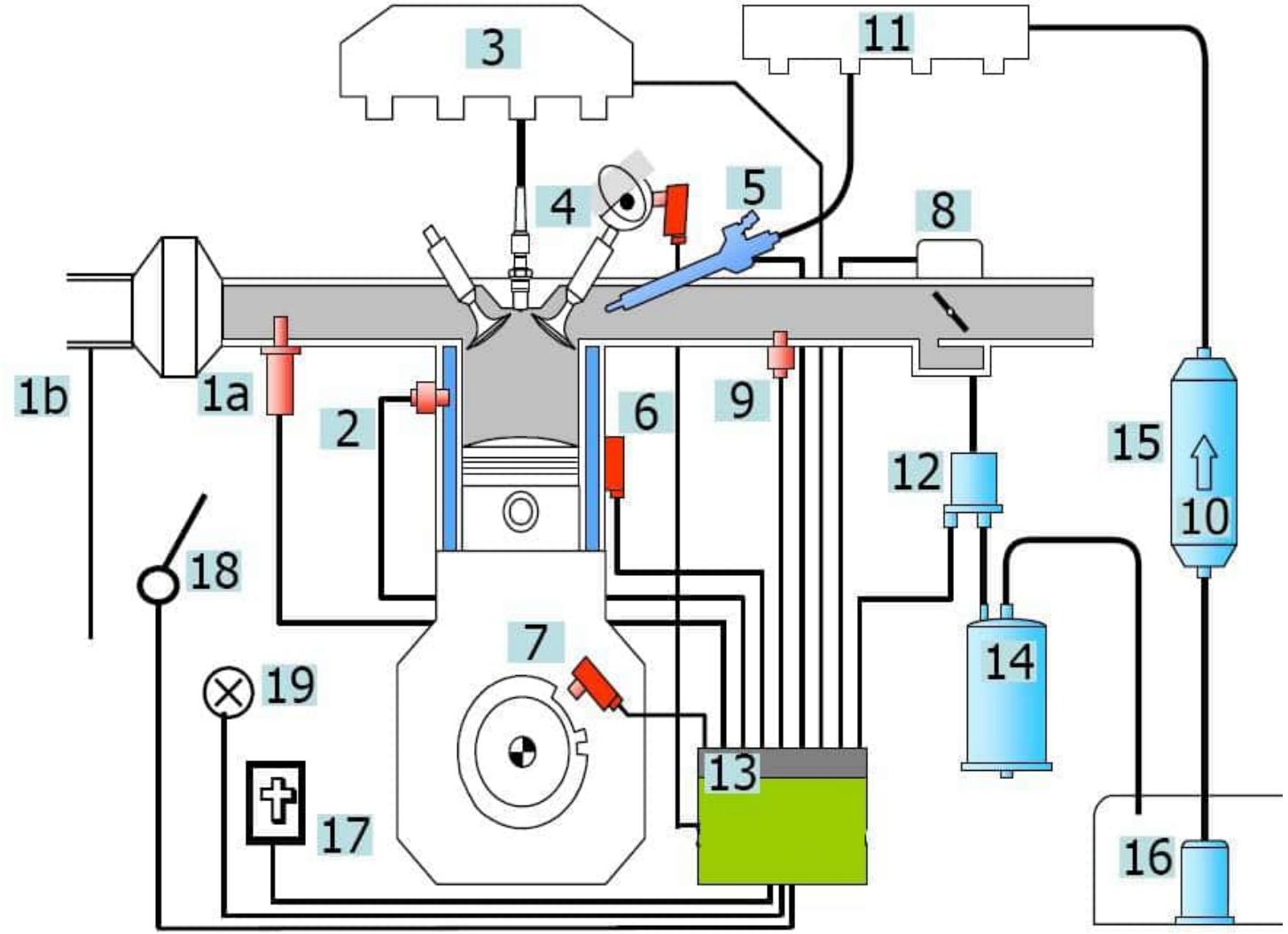
2.7 Typical Electronic Controls Systems

E/E-subsystems implement open and closed-loop control systems (see e.g. [2.14] pg. 10-12)

- One subsystem – one ECU, e.g. engine management
- Several sensors and actuators directly connected to the ECU
- Inter-domain and intra-domain interaction between ECUs via bus-systems, e.g.
  - transmission control coordinates gear shift with engine management (intra-domain)
  - ABS/ASR control commands engine torque reduction when active (inter-domain)



Typical Powertrain System: Compact Car Gasoline Engine Management (see [2.13] pg. 52-77, 158-171 and [2.14] pg. 13-23)



Source: Form, Fahrzeugelektronik, Uni Braunschweig

Sensors

1 Pre- and post Catalyst Lambda-Sensors
2 Water Temperature
6 Knock Sensor
7 Crankshaft Speed Sensor
9 Intake Air Pressure and Air Temperature
18 Accelerator Pedal Sensor
20 Camshaft Speed Sensor (not shown)

Actuators

3 Ignition Coils
4 Spark Plug
5 Fuel Injectors
8 E-Throttle
17 Malfunction Indicator Lamp (MIL)
19 ECU Error Lamp
12 Active Carbon Filter Valve

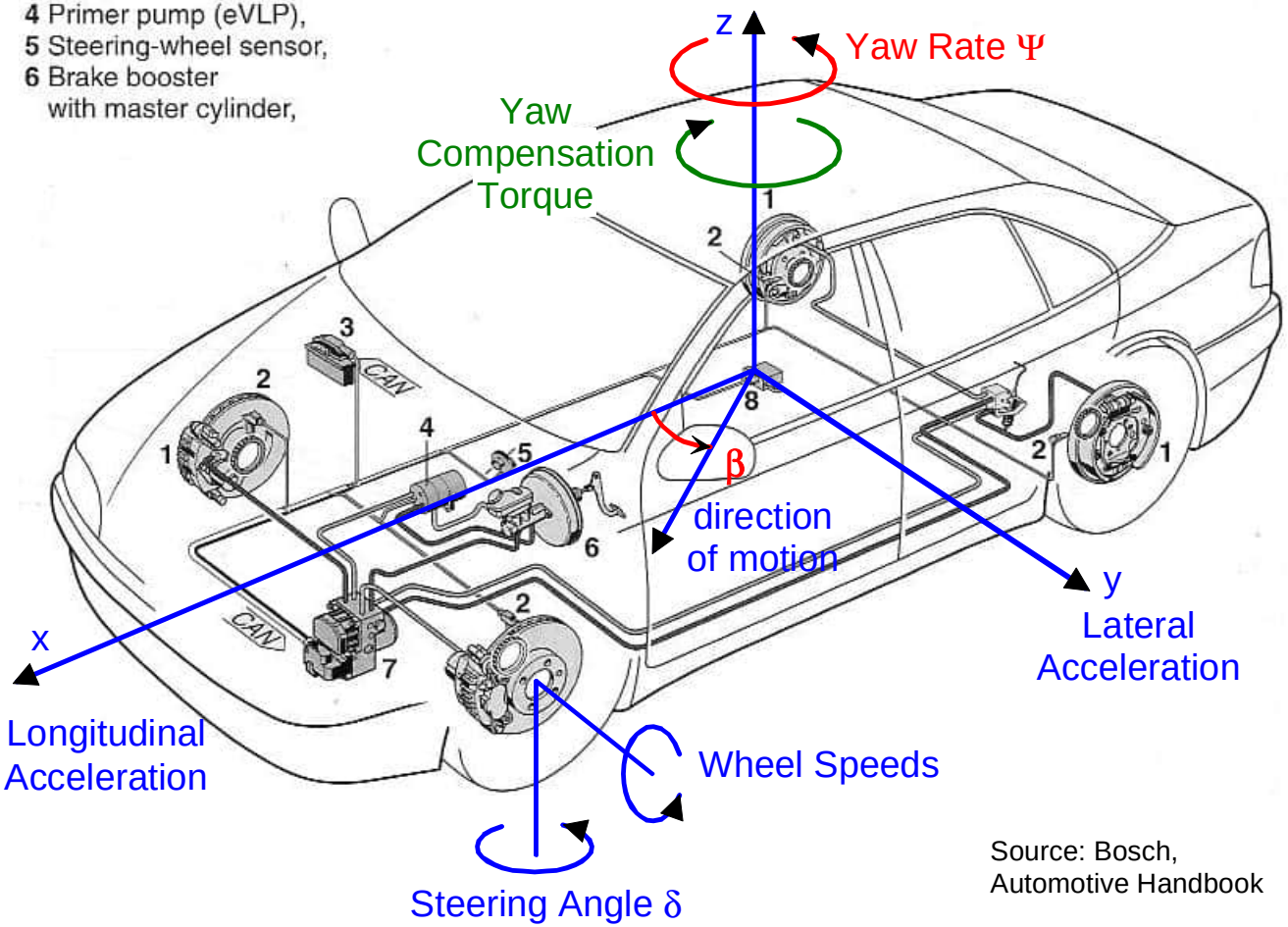
Miscellaneous

10 Fuel Pressure Control Valve
11 Fuel Rail
13 ECU
14 Active Carbon Filter
15 Fuel Filter
16 Fuel Pump and Tank

Typical Chassis System: Electronic Stability Control ESP (see e.g. [2.13] pg. 194-201 and [2.14] pg. 46-61)

ESP – Complete closed-loop control system (component locations)

1 Wheel brakes,
2 Wheel-speed sensors,
3 ECU,
4 Primer pump (eVLP),
5 Steering-wheel sensor,
6 Brake booster with master cylinder,
7 Hydraulic modulator with primary-pressure sensor,
8 Yaw sensor with lateral-acceleration sensor.



Source: Bosch, Automotive Handbook

Measure and observe the driver's intention and the vehicle's actual motion.

Control the stability of the car (yaw rate Ψ and float angle β)

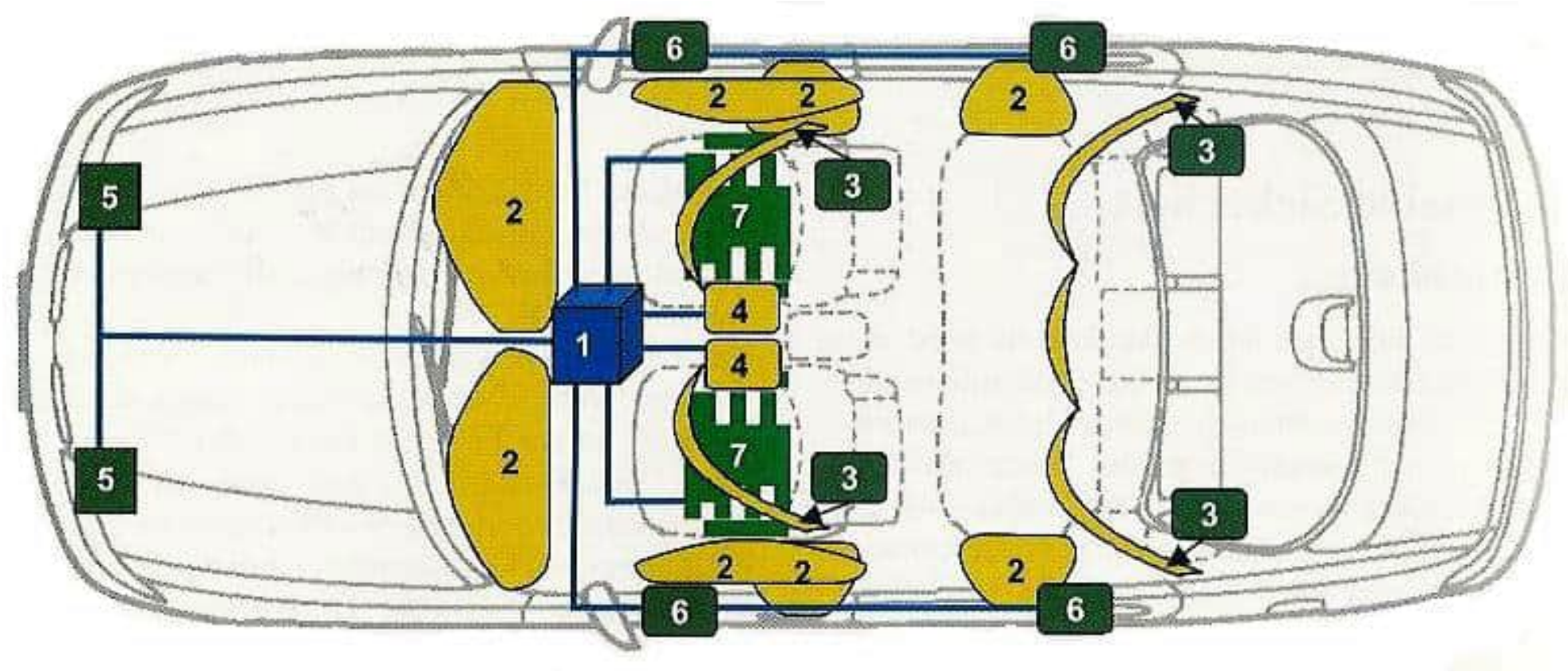
- **by braking** the appropriate wheels and/or
- **by increasing** the **engine torque** at the driven axle

to create the necessary yaw compensation torque.

Hierarchical control system:

- Top level
  - ESP control
- Lower Level
  - ABS control: wheel slip during active braking
  - MSR control: engine drag torque during passive deacceleration
  - TCS (traction) control: wheel slip during acceleration

Typical Body System: Airbag Control (see e.g. [2.13] pg.210-217 and [2.14] pg. 62-69)



Quelle: Wallentowitz/Reif, Handbuch Fahrzeugelektronik, Vieweg

	Sensors
5	Front Impact Sensors
6	Side Impact Sensors
7	Occupant Sensors

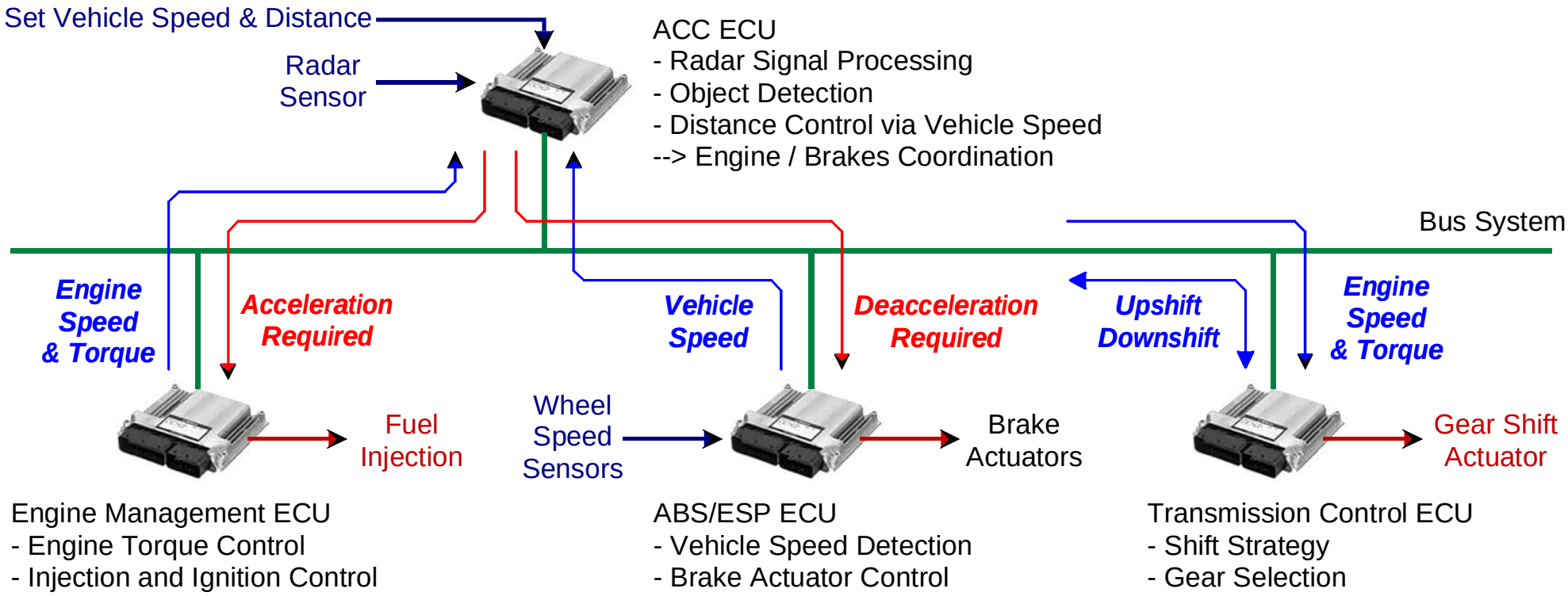
	Actuators
2	Front, Side and Rear Airbags
3	Safety Belt Fasteners
4	Safety Belt Force Limiters

	Miscellaneous
1	ECU

Typical Driver Assistance System: Adaptive Cruise Control ACC

(see [2.13] pg. 202-209 and [2.14] pg. 54-61)

- Keep the vehicle speed at a constant value, set by the driver
- Decelerate, if the distance to the car ahead is too small
- Accelerate up to the set speed, if the distance to the car ahead is big enough
- Implemented as a **distributed system** with various ECUs working together



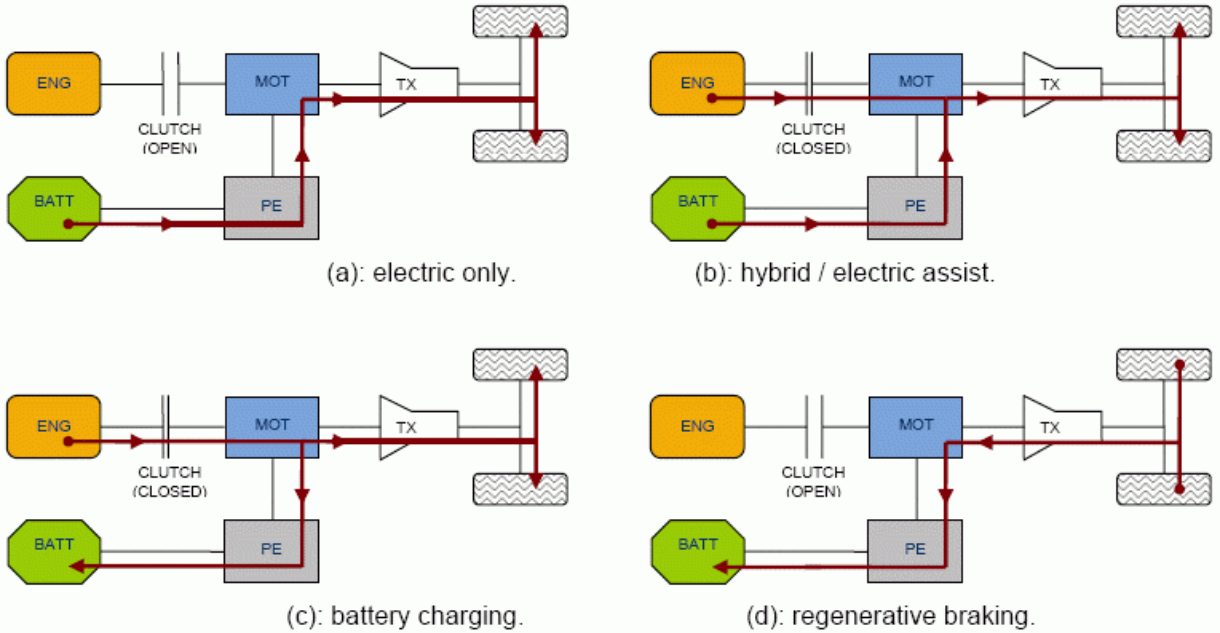
Typical Hybrid Vehicle (HEV) System (see e.g. [2.13] pg. 218-242 and [2.14] pg. 366-383)

- Propel the car via combustion engine and/or electric motor

  Various architectures with series and/or parallel structure allow pure electric (EV) mode with electric motor only, power boost with motor and engine, …

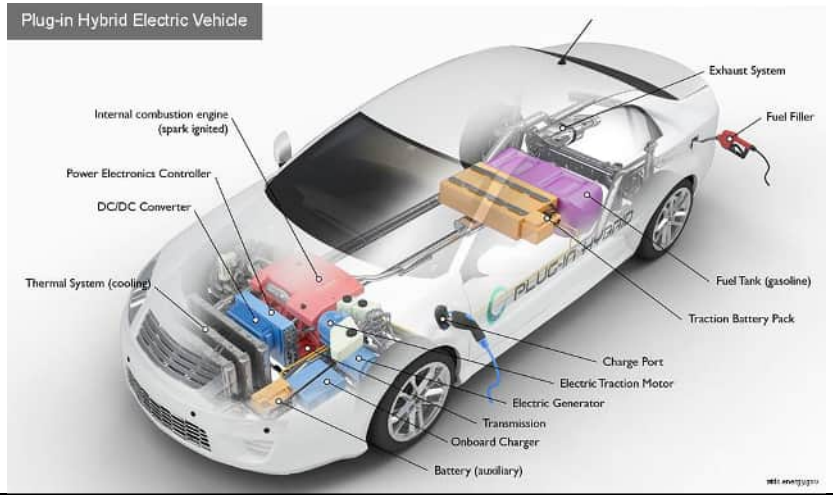
- Recharge battery when braking (regenerative braking, recuperation) by operating motor in generator mode → reduce fuel/energy consumption

  Additionally charge battery from mains (**Plugin-Hybrid PHEV**) when parked → use non-gasoline (cleaner ?) energy sources



ENG … combustion engine
BATT … battery,
PE … power electronics,
MOT … electric motor
TX … transmission

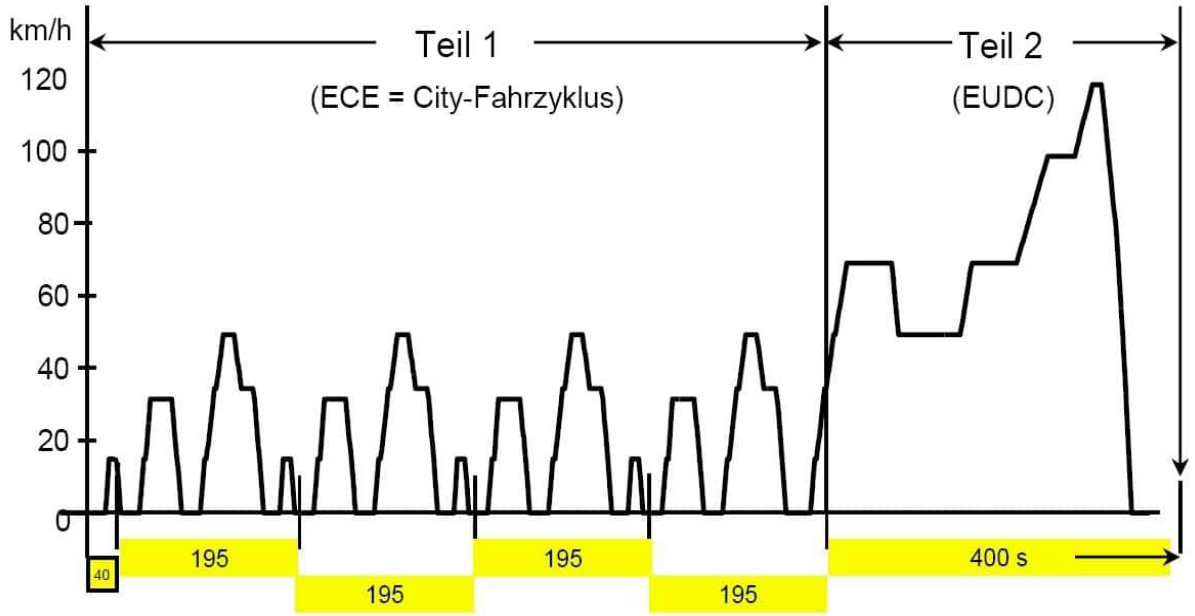
ECUs not shown



Hybrid and Electric Vehicle Features:

Category / *Feature*	*Hybrid Vehicle Internal Combustion Engine ICE + Electric Motor EM*				*Battery Electric Vehicle* ***BEV***	*Fuel Cell Electric* Vehicle **FEV**
	Micro	***Mild***	***Full HEV***	***Plug-In PHEV***		
Start-Stop-System Shutdown combustion engine, when car is stopped	Yes	Yes	Yes	Yes	No ICE	Electric energy dyn. generated via H_2 Fuel Cell
Recuperation Recharge battery when deaccelerating/braking	(Yes)	Yes	Yes	Yes	Yes	
Power Boost Additional torque from E-motor when accelerating	No	Yes	Yes	Yes		
Electrical Driving Driving without combustion engine	No	No	Yes (~ 2 km)	Yes (~ 50 km)	Yes (> 150 km)	Yes (> 300km)
Typ. Power E-Motor	2 … 3 kW	10 … 15 kW	> 15 kW	> 15 kW	> 50 kW	
Typ. Battery Capacity		< 2 kWh	< 2 kWh	5 … 15 kWh	> 15 kWh	
Power Battery Voltage	12V	< 60V	> 60V	> 100V	> 300V	
Power Battery Design	Optimized for Peak Power	→			Optimized for Max. Capacity	Small battery + liquid/high press. H_2 tank
Optional Source: M.Vogt, S.Link: Herausforderung Elektromobilität. Elektronik Automotive Sonderausgabe Elektromobilität				Recharge battery from public power grid ("Plug-In")	Range Extender **REX**: comb. engine to recharge battery	

Appendix: Old NEFZ Driving Cycle for Emission Measurement

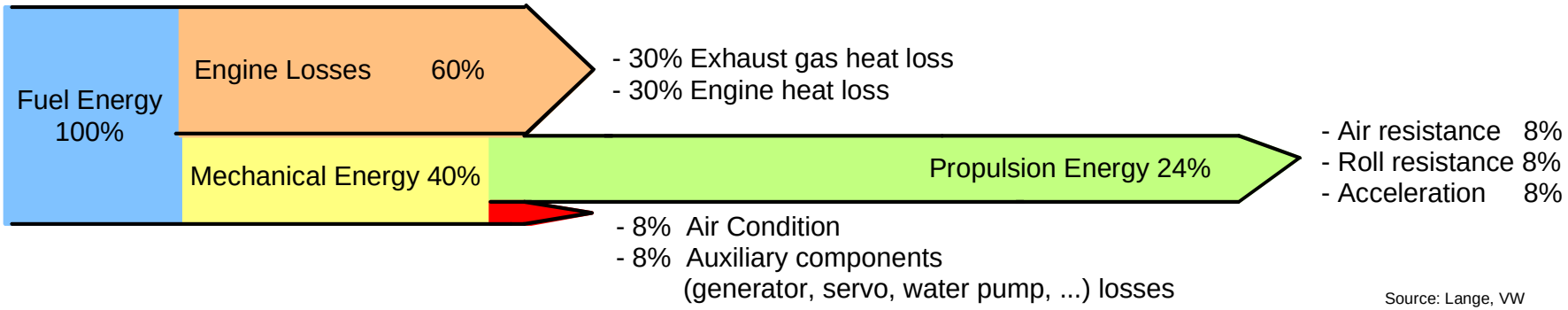


ECE European City Cycle
- Duration: 195 s
- Length: 4 km
- Idling: 31%
- Avg. Speed: 19 km/h
- Max. Speed: 50 km/h
- Repeated 4 x

EUDC Extra Urban Driving Cycle
- Duration: 400 s
- Length: 11 km
- Max. Speed: 120 km/h

These cycles partly also used for fuel consumption measurement.

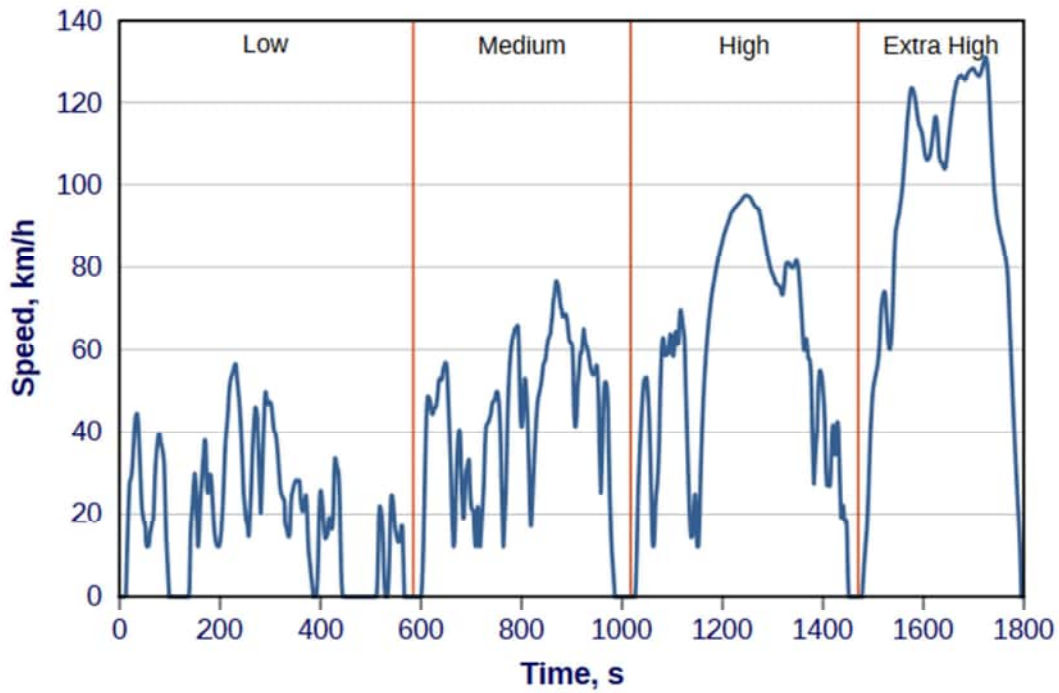
ECE/EUDC Energy Flow in NEFZ cycle



Source: Lange, VW

Appendix: New WLTP and RDE Tests for Emission Measurement

The NEFZ was substituted by the **Worldwide Harmonized Light Vehicle Test Procedure WLTP** in 2017/2018, which includes more realistic driving cycles:



(Sources: http://www.dieselnet.com/standards/cycles/, http://wltpfacts.eu/what-is-wltp-how-will-it-work/)

Additionally, **Real Driving Emission RDE** tests will be conducted on roads, rather than on test stands only (http://www.caremissionstestingfacts.eu/rde-real-driving-emissions-test/#), which should lead to more realistic emission and fuel consumption measurements.

Further Reading

[2.1] Robert Bosch GmbH (Editors): Automotive Handbook, Wiley & Sons, 10th edition, 2018

[2.2] Robert Bosch GmbH (Editors): Automotive Electrics – Automotive Electronics. Springer Vieweg, 5th ed., 2013

[2.3] Robert Bosch GmbH (Editors): Diesel-Engine Management. Springer Vieweg, 2014

[2.4] Robert Bosch GmbH (Editors): Gasoline-Engine Management. Springer Vieweg, 2014

[2.5] Robert Bosch GmbH (Editors):Brakes, Brake Control and Driver Assistance. Springer Vieweg, 2014

[2.6] Short manuals about various automotive systems (Bosch Yellow Jackets Series), see http://www.bosch-fachinformation-automobil.viewegteubner.de/

[2.7] Robert Bosch GmbH (Editors): Fachwörterbuch Kraftfahrzeugtechnik German – English – French – Spanish. Vieweg-Teubner, 3rd edition, 2005 (Automotive Vocabulary Dictionary)

[2.8] R.K. Jurgens: Automotive Electronics Handbook. McGraw-Hill, 1999 (partially outdated)

[2.9] H. Wallentowitz, K. Reif: Handbuch Kraftfahrzeugelektronik, Vieweg, 2nd edition, 2011 (in German)

[2.10] U. Kienecke, L. Nielsen: Automotive Control Systems. SAE, 1st edition, 2000
For other SAE Books and Papers see http://store.sae.org/catalog

[2.11] T. Form, T.: Fahrzeugelektronik. Lecture Manuscript (in German), TU Braunschweig, 2006, www.ifr.ing.tu-bs.de/lehre/vorlesungen.php (now requires TU Braunschweig account)

[2.12] McKinsey: Automotive software and electrical/electronic architecture: Implications for OEMs. 2019

[2.13] K. Reif (Ed.): Fundamentals of Automotive and Engine Technology. Springer Vieweg, 2014
https://link.springer.com/book/10.1007%2F978-3-658-03972-1

[2.14] Robert Bosch GmbH (Ed.): Automotive Electrics and Electronics. Springer Vieweg, 2014
https://link.springer.com/book/10.1007%2F978-3-658-01784-2

Most recent info about automotive technology and industry trends:

Web-Portal „ATZ online“ and the periodicals “ATZ Automobiltechnische Zeitschrift”, “MTZ Motortechnische Zeitschrift”, „ATZ autotechnology“ and „ATZ elektronik“. Printed publications in German, the online version of the peridocals is also available in English. www.atzonline.de

Web-Portal SAE Automotive Engineering International Online. www.sae.org/mags/AEI

Note: [2.1] – [2.6] are also available in German from Springer Verlag. German editions in many cases are more up-to-date than English editions.

Videos

There is a nice mix of short Youtube videos on various automotive electronic systems topics by Vector Informatik:
https://www.youtube.com/results?search_query=%23engineeringthejigsaw

Some interesting videos are:

- Episode F1: E/E systems: A 50k Jigsaw puzzle? https://www.youtube.com/watch?v=4-Myr2foivE
- Episode F2: What is an ECU? https://www.youtube.com/watch?v=xHfInsMEYdQ
- Episode F3: Networks in Vehicles https://www.youtube.com/watch?v=DUNWZnhneiI
- Episode F4: What is Vehicle Diagnostics? https://www.youtube.com/watch?v=E2OP4jLEV50
- Episode F5: Calibration https://www.youtube.com/watch?v=pp4woRbJWUE
- Episode F6: What is AUTOSAR? https://www.youtube.com/watch?v=W5F8nQuwuWY
- Episode F7: How to update ECU Software https://www.youtube.com/watch?v=REA-rw2WQ44
- Episode F8: High Performance Computing Platforms https://www.youtube.com/watch?v=_uDG-ENHqds
- Episode F9: What is AUTOSAR Adaptive? https://www.youtube.com/watch?v=o9VMp2lNwF4
- Episode F10: How have E/E Systems Evolved? https://www.youtube.com/watch?v=OjnQ52f1SEE
- Episode F11: How are Vehicle E/E Systems Evolving? https://www.youtube.com/watch?v=XpvTOuOgGWY

and many more …

Selected E/E-System and ECU Suppliers

Bosch	www.bosch.de
Denso	http://www.denso-am.de/
Delphi	https://www.delphi.com/
Conti	www.conti-online.com/www/automotive_de_en
Visteon	https://www.visteon.com/
Hella	www.hella.com/hella-com/52.html?rdeLocale=en
Automotive Lightning	www.al-lighting.com/home
Autoliv	https://www.autoliv.com/
Leoni	https://www.leoni.com/en/
Valeo	www.valeo.de/en/
Magneti Marelli	www.magnetimarelli.com
. . .	

Selected Development Tools and Engineering Service Companies

Vector Informatik	vector.com/vi_index_en.html
Elektrobit	www.elektrobit.com
Bosch Engineering	www.bosch-engineering.com/
Porsche Engineering	www.porscheengineering.com/peg/en/
Akka (former MBTech, Gigatronik)	www.akka-technologies.com/?lang=de
Bertrandt	www.bertrandt.com/en/
Softing	automotive.softing.com/en/startpage.html
DSA	www.dsa.de/en
Star Corporation	https://www.star-cooperation.com/en/solutions/fahrzeugelektronik/
EDAG	https://www.edag-engineering.de/en/
. . .	

Note: Incomplete list. Chinese, Indian and many other European, Asian and American suppliers not included

Appendix: Car Classification Glossary

US English (British English)	*German*	*Example*
Mini Compact Car	Stadtwagen	Smart ForTwo
Sub Compact Car	Kleinwagen	VW Polo, Up BMW Mini
Compact Car (Small Family Car)	Untere Mittelklasse (*Golf-Klasse*)	VW Golf Mercedes Benz A-Class BMW 1…
Midsize Car (Family Car)	Mittelklasse	VW Passat
Compact Executive		Audi A4 BMW 3… Mercedes-Benz C-Class
Large Car (Large Family Car)	Obere Mittelklasse	Ford Mondeo
Executive		Audi A6 BMW 5… Mercedes-Benz E-Class
Luxury Car	Oberklasse	Audi A8 BMW 7… Mercedes-Benz S-Class
Super Luxury Car	Luxusklasse	Bentley Rolls Royce
Sports Car (Performance Car, Roadster)	Sportwagen	BMW Z3 Mercedes-Benz SLK Porsche 911
Super Sports Car	Supersportwagen	Ferrari …
Sports Utility Vehicle (SUV)	Pseudo-Geländewagen	BMW X3 Mercedes-Benz M-Class Porsche Cayenne / VW Touareg
Multi Purpose Vehicle (MPV)	Familien-Van	VW Sharan

US English	*German*	*Example*
Light Duty Truck	Transporter	< 2.8 t
Medium Duty Truck	Leichter Lastwagen	< 7.5 t
Heavy Duty Truck	Schwerer Lastwagen	< 40 t

Body types

US English (British English)	*German*	*Example*
Sedan (Saloon)	Limousine mit Stufenheck	Mercedes-Benz S-Class
Hatchback, Fastback	Limousine mit Heckklappe	VW Golf
Station Wagon (Estate Car)	Kombi, Variant	Audi A4 Avant
Convertible	Cabriolet	VW EOS

Online and Offline Dictionaries

[1.1] Lingopad. Offline tool with dictionaries for various languages to German and vice versa. Freeware. www.ego4u.com/en/lingopad

[1.2] GE Trans. Offline tool with dictionary English-German and vice versa. Shareware. www.theabsolute.net/sware/getrans.html

[1.3] LEO. Online tool with dictionaries for various languages to German and vice versa. www.leo.org

[1.4] Babylon. Online tool. Translates between various languages. http://translation.babylon-software.com/

[1.5] Quick Dic. Online and offline dictionary. www.quickdic.org

[1.6] Google Translator. https://translate.google.de/

[1.7] Bosch Fachwörterbuch Kraftfahrzeugtechnik. Vieweg+Teubner-Verlag. http://www.amazon.de/dp/3322803341

[1.8] Vector Informatik Online Dictionary for Electronic Networking in the Automobile. https://vector.com/vi_dictionary_en.html

Automotive System Design

System and Software Architecture

Hochschule Esslingen
University of Applied Sciences

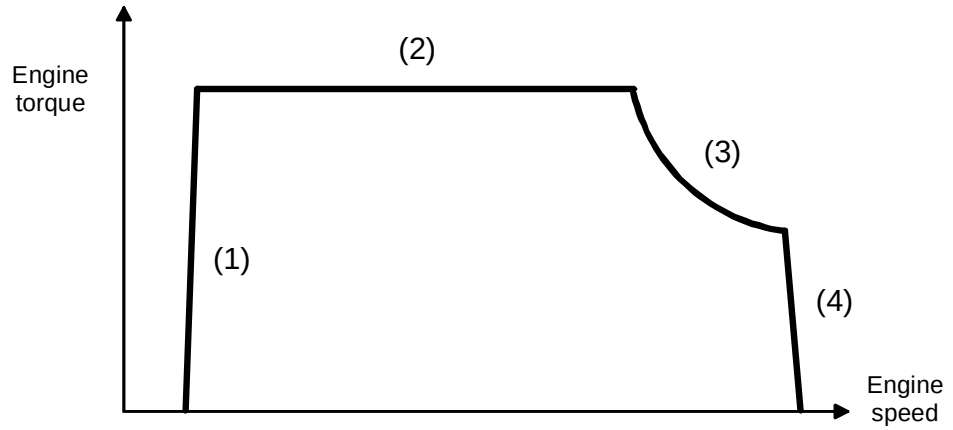
Chapter 2 Quick Test

Try to answer the following questions to find out, how good your knowledge about typical automotive systems is. These questions cover topics each automotive engineer must be familiar with. You should be able to answer most of them by heart without searching the manuscript or Google!

If you have problems, you definitely should read literature references [2.13] and [2.14] and the lecture manuscript's appendices ***Advanced Mobility*** and ***ECU Electronic Control Units***.

1. Combustion Engines and Emissions

1.1 The following diagram shows the typical speed – torque limits of an ICE (simplified). What does cause these limits (1) to (4) ?



Limit	Root cause
(1)	Idle speed limit, for lower speeds ICE do not run smoothly
(2)	Max. torque limit due to mechanical constraints (forces)
(3)	Max. power limit, due to limited cooling
(4)	Max. speed limit, due to mechanical constraints (forces)

1.2 The crankshaft of a combustion engine is running at 2000 rpm. What is the associated camshaft speed? Does it make a difference, whether it is a 2-stroke or a 4-stroke engine?

1000 rpm for a 4-stroke engine

A 2 stroke engine normally does not have a camshaft. If it has one, the camshaft speed is the same as the crankshaft speed, i.e. here 2000 rpm.

1.3 What is the major actuator variable to control the engine torque of . . .

. . . a gasoline engine	Air quantity (throttle position)
. . . a diesel engine	Fuel quantity (injection valve opening)

1.4 Name at least 4 emission components of gasoline and diesel ICE and mark the engine operating range, in which these emission components are most critical:

Emission Component	Most critical , when the ICE is operating …	
	… under full load conditions	… under part/low load conditions
CO2 / CO	X	
HC	X	
NO_X		X
PM	X	

1.5 What does happen, if the engine management tries to run a gasoline ICE at lambda $\lambda = 8$?

Fuel – air mixture too lean to burn, no combustion, engine stalls.

1.6 The output power of a combustion engine is $P \sim n \cdot m_{air} / \lambda$ with engine speed n, air mass m_{air} and air to fuel ratio λ. Which of these values is used as the primary control variable and what is the primary actuator, when the driver wants to accelerate in a car with …

… a gasoline engine	m_{air} via the throttle (quantity control)
… a diesel engine	λ via the fuel injection system (quality control)

1.7 Why do gasoline engines achieve higher maximum engine speeds than diesel engines?

Gasoline engine combustion operates at lower cylinder pressures and thus creates less mechanical stress at the same speed or vice versa can reach higher speeds at the same level of mechanical stress vs. a diesel engine.

1.8 Why do diesel engines have a big NO_X emission problem while NO_X is less critical for gasoline engines?

Because diesel engines works with lean combustion ($\lambda >> 1$), e.g. excess oxygen at high temperatures and high pressures, which does generate NO_X.

1.9 Which combustion emission component is addressed via Exhaust Gas Recirculation EGR?

NO_x, by reducing the amount of excess O_2 in diesel engines.

1.10 Which type of gasoline engine may need a particulate filter additionally to the classical 3-way catalyst?

Gasoline engines with direct injection (DI).

1.11 What is the advantage of RDE vs. FTP-75 emission testing?

Real Driving Emission tests measure emissions under real traffic operating conditions, while FTP-75 tests under artificial (= less critical) conditions on a test-stand.

1.12 A 6 cylinder 4-stroke gasoline engine is running at 6000 rpm crankshaft speed. Calculate the following times (formula and numerical result in milliseconds):

Time between two adjacent ignitions in the same cylinder:

$$T_{ign} = \frac{2}{n} = \frac{2}{6000\ 1/min} = \frac{2 \cdot 60}{6000}\ secs = 20\ msec$$

Time between ignitions in adjacent cylinders:

$$T_{cyl/cyl} = \frac{T_{ign}}{6} = \frac{20\ msec}{6} = 3{,}3\ msec$$

2. HEVs and EVs

2.1 What are the advantages and disadvantages of serial hybrid vs. parallel hybrid designs?

Serial hybrids require 2 electrical engines (motor and generator, i.e. more weight, higher electrical losses), but can always operate the combustion engine at the optimal engine speed independent of vehicle speed.

Parallel hybrids require a more complex transmission, but increases the output power, because the combustion engine's and the electric motor's torque do add.

2.2. State-of-the-art Formula 1 race cars do use a KERS system. What is KERS?

KERS = Kinetic Energy Recovery System, i.e. regenerative braking system

2.3 Why do combustion engines require a multi-level gearbox and a clutch, while electrical drives do not?

Electric motors can operate down to standstill (0 rpm), while combustion engines can't → clutch required.
Combustion engines can generate a considerable amount of torque only over a very limited engine speed range (idle speed ~800 rpm .. max. speed 4000 rpm (diesel) … 6000 rpm (gasoline)), while electrical engines can generate constant torque from 0 rpm to nominal speed and constant power above nominal speed to max speed (with max speed 3..10 x nominal speed) → no multi-level gearbox required.

2.4 A typical gasoline engine car with a fuel tank capacity of 40 ltr ≈ 30 kg has a driving range of approx. 500 km (8 ltr / 100 km). What is the estimated mass of the battery of an electrical vehicle with the same driving range?

Relative efficiency of gasoline engine vs. electrical engine: $\eta_{Gasoline} / \eta_{Electric} = 0.3$

Energy density of gasoline: $c_{Gasoline} = 11.0$ kWh / kg

Energy density of Li-Ion battery: $c_{Li\text{-}Ion} = 0.2$ kWh / kg

Required gasoline energy $E_{Gasoline} = m_{Gasoline} \cdot c_{Gasoline} = 30 \text{ kg} \cdot 11 \text{ kWh/kg} = 330 \text{ kWh}$

Required electric energy $E_{Battery} = E_{Gasoline} \cdot \eta_{Gasoline} / \eta_{Electric} = 330 \text{ kWh} \cdot 0.3 \approx 100 \text{ kWh}$

Required battery mass $m_{Battery} = E_{Battery} / c_{Li\text{-}Ion} = 100 \text{ kWh} / 0.2 \text{ kWh} \approx 500 \text{ kg}$

2.5 How does a non plug-in hybrid drive reduce fuel consumption?

Because of regenerative braking. That is, when the car is braking, part of the kinetic energy is converted and stored as electric energy, rather than dissipated as heat. The stored energy then is reused, when the car accelerates again.

2.6 What are the main differences between a HEV, a PHEV and a BEV with Range Extender?

PHEV and HEV use a **combustion engine and** an **electrical motor** to drive the car. In a **BEV w/ REX only the electrical motor drives the car**, while the combustion engine is only used to reload the battery, when it is (nearly) unloaded.

In all types, when braking the electrical motor operates in generator mode to recuperate the battery.

In **PHEVs** the battery can, in **BEVs** the **battery** must be **loaded from** the **public mains**. In HEVs the battery is loaded via the combustion engine only.

BEVs use very **large batteries**, **PHEV medium size batteries**, HEVs small batteries.

2.7 A BMW i3 EV consumes approx. 13 kWh electrical energy per 100 km, a BMW 116i approx. 6 l gasoline per 100 km (official VO (EU) 715/2007 test procedure data from BMW's web page as of 12/2018).

With its 52 l gasoline tank, the BMW 116i achieves a nominal driving distance of approx. 850 km.

Which battery mass would be needed, if the BMW i3 EV was to reach the same driving distance (before recharging the battery)? The batteries have an energy density of ~ 56 Wh / kg.

850 km x 13 kWh / 100 km = 110,5 kWh → 110 500 Wh / 56 Wh/kg = 1973 kg

2.8 What are the major temperature related problems of LiIon batteries?

High temperatures:

Overheating → thermal runaway. Battery may start to burn.

Low temperatures:

Internal resistance increases → Battery drastically looses capacity.

2.9 What are SOC and SOH of a battery? Explain.

SOC:

State of charge = current energy contents, which indicates the remaining driving range

SOH:

State of Health = current battery capacity (affected by aging) related to original capacity, which indicates remaining battery lifetime due to aging

2.10 How long does it take at least to fully charge an empty battery of a Tesla model S (90 kWh) with a standard EV home charger with 7.5 kW?

90 kWh / 7.5 kW = 12 h

3. Vehicle Dynamics (ESP/ESC, ABS, EPS), Safety and Driver Assistance Systems

3.1 Is the car in the picture below oversteering or understeering? How shall the ESP system react?



Oversteering. Reduce engine torque, brake outer wheels

3.2 ESP systems need an information about the direction of the vehicle intended by the driver. Which sensor(s) is/are used for this purpose?

Steering angle sensor

3.3 What is the major difference between Active Safety Systems and Passive Safety Systems? Name an example for each of these systems.

Active Safety Systems try to avoid accidents by actively influencing the driving behavior. Example: Electronic Stability Control ESC.

Passive Safety Systems try to mitigate consequences in case of an accident. Example: Airbag

3.4 ABS, ESP and TCS help drivers to control their car in various driving situations. Please check the following statements and mark the respective system(s).

System …	ABS	ESP	TCS
… reduces brake distance on dry road.			
… improves steerability of car when braking.	X		
… allows to drive faster through curves.		X	
… allows to accelerate faster on wet roads.			X

3.5
The radar sensor of an ACC system has a measurement range of approx. 150 m and provides distance measurement values with a sample rate of 10 Hz. Imagine a scenario, where a car is driving with a speed of 180 km/h, when a non-moving obstacle enters the measurement range. Assume, that the brakes are activated immediately after the next measurement sample. Can the car be stopped before hitting the obstacle, when the brakes provide a constant de-acceleration of 8 m/sec².

(4 points)

180 km/h = 50 m/s

Remaining distance, when brakes are activated: 150 m – 50 m/s * 0.1 s = 145m

$v = a \cdot t$ → Time required for braking $t = v / a$

Braking distance: $s = 1 / 2\ a\ t^2 = 1 / 2\ v^2 / a = 1 / 2 \cdot (50\ m/s)^2 / 8\ m/sec^2 = 156\ m > 145\ m$ → Accident cannot be avoided.

3.6 Why do today's Electronic Active Steering systems still have a mechanical steering column additionally to the electric steering motor?

Redundancy in case the electric/electronic system fails

3.7 Into which SAE level for assistance and automation systems does an Electronic Park Assistant system fit, if it interacts with the Electric Active Steering system when parking?

Class 2 (Partial Automation)

3.8 Which car is more likely to oversteer? Check all that apply.

[ ]	Front-wheel driven	[X]	Real-wheel driven	[ ]	All-wheel drive

Which sensor(s) will detect the oversteering situation?

Yaw sensor, wheel speed sensors (and steering angle sensor)

3.9 What does the acronym ADAS stand for?

Advanced Driver Assistance System

3.10 Cars use a combination of sensors to detect their environment. Which advantages and limits/disadvantages do the following sensors have?

Sensor	Advantage	Disadvantage/limitation
Ultrasonic	Cheap	Short distances <5..10m only
Radar	Works even under bad visibility conditions Precise distance / speed measurment.	Medium to long distances only Costly signal processing
Video camera	Can detect type of object 3D view	Costly signal processing Night view, heavy rain, fog

3.11 ABS and TCS systems shall detect spinning or blocking wheels. How can the system find out, when …

… the left front wheel is blocking?

Speed of blocking wheel (measured by sensor) << speed of other wheels

… when the driven axle is spinning?

Speed of wheels on driven axle >> speed of wheels on undriven axle

3.12 What is “yawing”? Which system tries to control it?

Rotation around the vertical axis. → Measured and controlled by ESP

4.1 ECU Hardware and Software

4.1 What are the major components of an electronic control system? Name at least 3.

ECU with control algorithm
Actuator(s)
Sensor(s)

4.2 What are the major circuit blocks of an electronic control unit ECU? Draw a block diagram.

Microcontroller (CPU) with memory and clock generator
Signal inputs
Driver outputs
Network interface
Power supply
Block diagram see e.g. manuscript chapter 2.6

4.3 Which ECU hardware is cheaper: Processing the signal from a switch or from a potentiometer? Why?

Digital inputs (switch) are cheaper than analog inputs (potentiometer), because analog inputs require a costly ADC.

4.4 What is EMC? Explain the EMC requirements "***Interference***" and "***Susceptibility***". Keywords only.

EMC: Electromagnetic compatibility

Interference: Don't disturb other electronic systems

Susceptibility: Don't be disturbed by other electronic systems.

4.5 What are the 3 major differences in design requirements/contraints for automotive software, when compared with personal computer software?

Safety & Reliability requirements

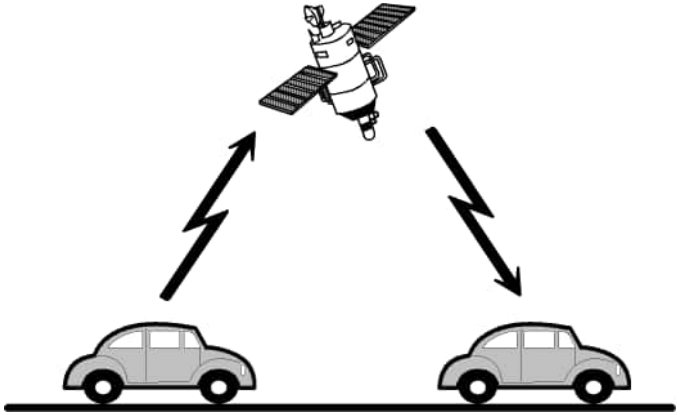
Hard realtime requirements

Ressource (memory, CPU performance) contraints

4.6 In which operating scenario must a passenger car ECU withstand a power supply voltage of 24V for several minutes, even if its nominal supply voltage is only 12V?

Jump start when the passenger car is connected to a truck battery.

4.7
If a Car 2 Car –communication system used geostationary satellites, what would be the minimum transmission latency due to the limited signal propagation speed? The one-way distance earth - satellite is approx. 36 000 km, electromagnetic waves travel with light speed, i.e. 300 000 km/sec.



$T_{Latency}$ = 2 · 36000 km / 300000 km/sec = 240 msec

4.8 What are the main ideas behind the AUTOSAR software initiative?

Reduce cost and improve reliability:

Make automotive software hardware-independent and standardize SW interfaces to allow reuse of software components.

Try to establish an off-the-shelf component scenario, where hardware and software can be sourced and combined from different suppliers.

Allow OEMs to provide mission-critical software components in-house to get a competitive advantage and reduce the dependency on major suppliers.

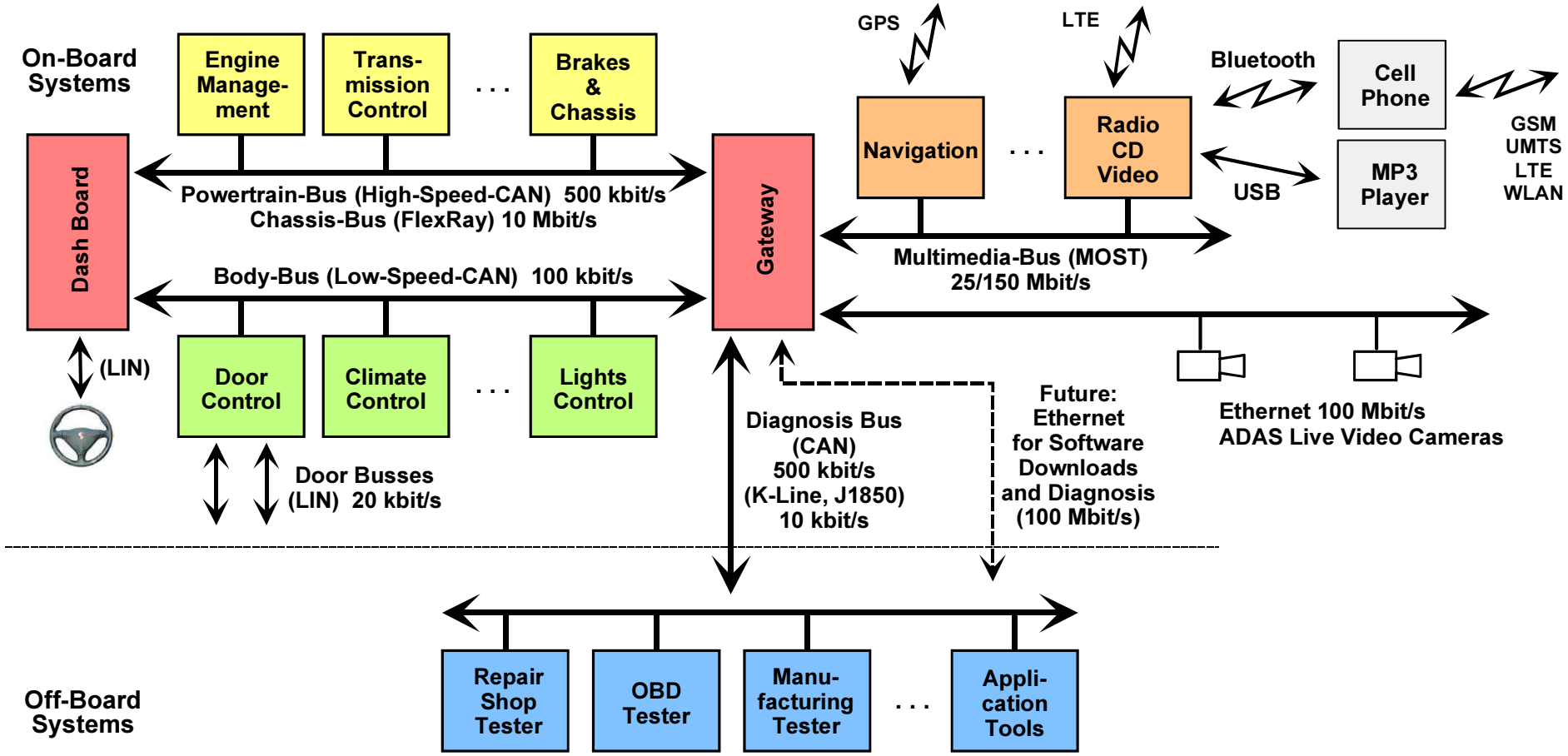
Chapter 3
Networks
Bus Systems and Communication Protocols

Hochschule Esslingen
University of Applied Sciences

3.1 Overview 2
3.2 Basics 5
3.3 Controller Area Network CAN Classic, CAN-FD and CAN-XL 16
3.4 Local Interconnect Network LIN 28
3.5 FlexRay 36
3.6 Media Oriented Systems Transport MOST 48
3.7 Automotive Ethernet 52
3.8 Outdated Busses: K-Line, SAE J1850 and SAE J1587/1708 57
3.9 Transport Protocols 58
3.10 Diagnostic Communication Protocols OBD and UDS 64
3.11 Network Communication Design 74

Further Reading 81

3.1 Overview



Multiplexing Busses, e.g. CAN, FlexRay, LIN, … :

How to transfer bits and bytes over wires?

Higher-Layer Protocols, e.g. KWP, UDS, OBD, … :

What do these bits and bytes mean?

SAE Classification of bus systems by bit rate

< 25 kbit/s	Class A	K-Line, J1850, LIN	≤ 10 Mbit/s	Class D	FlexRay/CAN FD
< 125 kbit/s	Class B	Low-Speed CAN	> 10 Mbit/s	Class E ADAS	Ethernet
< 1 Mbit/s	Class C	High-Speed CAN	> 10 Mbit/s	Media	MOST

Classification of bus systems by application

Drive	ECU to ECU	**On-Board Communication** • Open-loop and closed-loop control tasks • High real-time and safety requirements • Advanced Driver Assistance ADAS (Video, Radar,…)
Entertain	ECU to driver/passenger	**Infotainment** • High data rates, not safety related
Maintain	Workshop Authorities (TÜV/Dekra)	**Off-Board Communication „Diagnosis"** • Read diagnostic trouble codes • Change vehicle parameters, check sensors and actuators • Flash/EOL (End of Line) /OTA (Over the Air) programming Various levels of access protection (security) needed
Manufacture	Vehicle manufacturer (OEM) ECU manufacturer	
Calibrate	Development engineers	**Special case of Off-Board Communication** • Unprotected, fast access to all ECU internals

Requirements and Solutions

Application		*Message Length*	*Data Rate*	*Preferred Bus*
On-Board Communication				
Very-Low-Speed-Control (Multiplexing) e.g. Window Lifter	Slow Switch Signals	Short (several bits)	Low	Class A Bus **LIN** (20 kbit/s)
Low-Speed-Control e.g. Air Condition	Slow Measure-ment & Control Signals	Short (several bytes)	Medium	Class B Bus **Low-Speed-CAN** (125 kbit/s)
High-Speed-Control e.g. Engine Management	Fast Measure-ment & Control Signals	Short (several bytes)	High	Class C Bus **High-Speed-CAN** (500 kbit/s)
Very-High-Speed-Control e.g. Suspension Control	Fast Measure-ment & Control	Short (several bytes)	Very high Low Jitter	Class D Bus **FlexRay** 10 Mbit/s
Advanced Driver Assis-tance ADAS	Real Time Video & Radar	Streaming	Extremely high	**Automotive Ethernet** 100 Mbit/s … 1Gbit/s
Infotainment				
e.g. Telephone, Radio, TV, Navigation Maps	Audio- & Video Signals	Medium	High to Ex-tremely high	Media Bus **MOST** 25 (150) Mbit/s
Off-Board Communication				
e.g. Workshop Diagnosis End of Line Test	On-Board Dia-gnosis OBD Flash / EOL	Short to Very long (KB … GB)	Low Medium to high	**K-Line** (outdated) **SAE J1850** (outdated) **CAN, Ethernet**

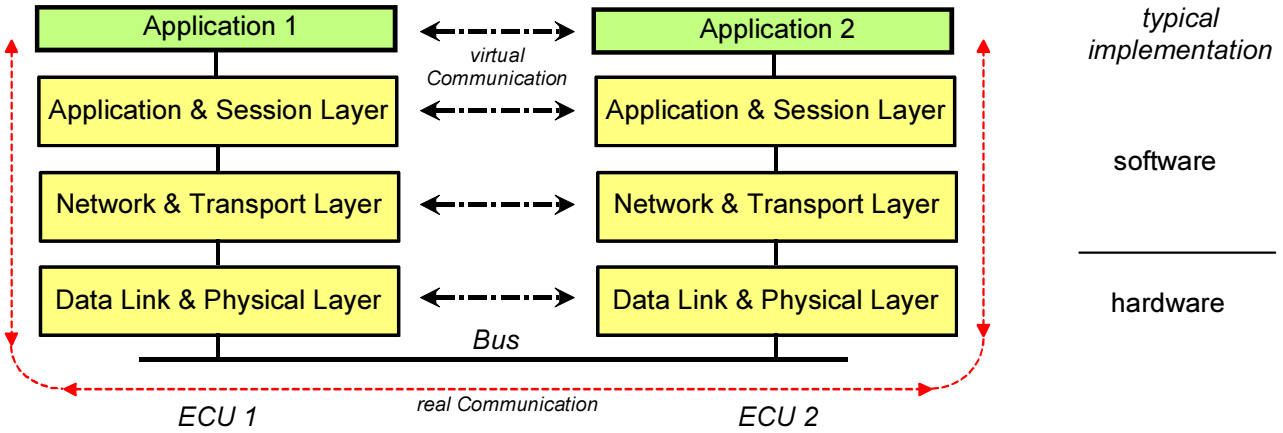
3.2 Basics

Open Systems Interconnect (OSI) Communication Model (ISO 1978) - Protocol Stack

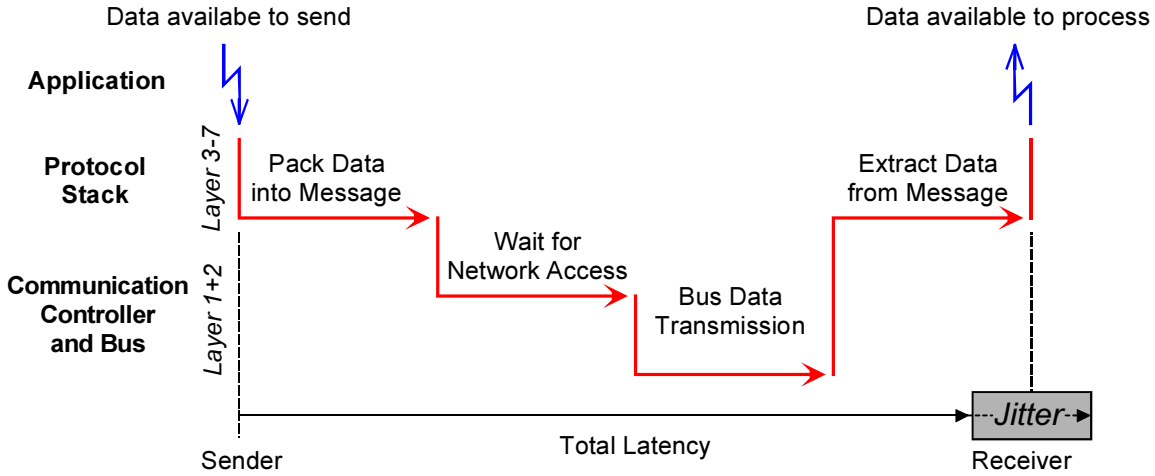
User application, e.g. engine management or user interface of a diagnosis tester			
Layer	*Name*	*Purpose*	*Example*
7	**Application Layer**	Interface to applications, e.g. diagnostic service *Read Diagnostic Trouble Codes*	Diagnosis protocol UDS on CAN ISO 15765-3
6	Presentation Layer		
5	Session Layer	Authentification, login	
4	**Transport Layer**	Segmentation of data on several mes-sages, flow control	Transport protocol ISO 15765-2
3	Network Layer	Routing, addressing, node detection and management	
2	**Data Link Layer**	Frame format, network access policy error detection	CAN ISO 11898-1
1	**Physical Layer**	Bit coding, signal levels	High-Speed-CAN ISO 11898-2/5
Bus wires and connectors (mechanics)			

- Bus systems like CAN, LIN or FlexRay typically define layer 1 and 2
- Upper layers are handled by software according to ISO, SAE or IEEE protocol standards
- In automotive protocols layer 3, 5 and 6 tasks typically are handled by layers 2, 4 or 7.

- In the ideal case, applications seem to interact directly without having to care about the underlying protocol layers and bus system. The different layers take care about communication details. Each layer virtually interacts with the associated layer in the other ECU.



- The more layers are used, the higher the transmission latency will be:



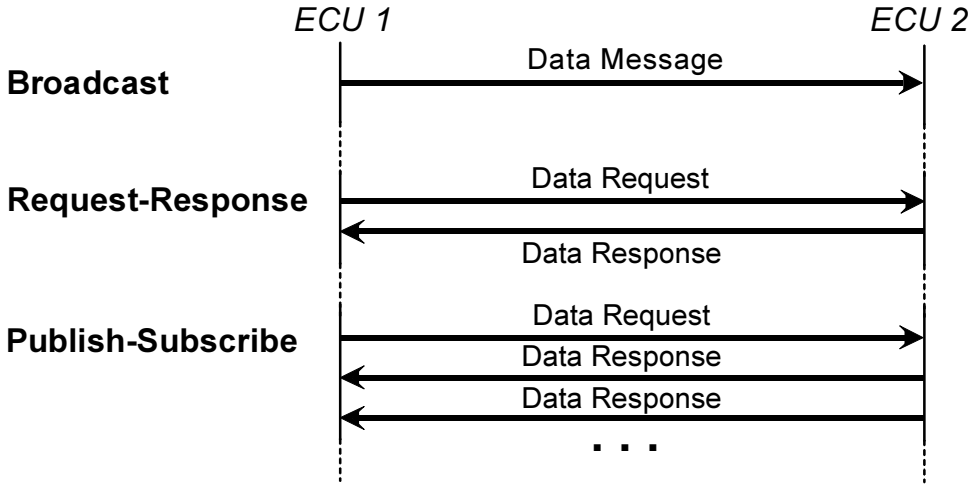
- On-board communication uses layer 1+2 only for minimum latency of real-time applications
- Off-board communication uses full protocol stack for ease of communication.

Communication flow

Broadcast
ECU sends periodically (**time-triggered**) or spontaneously (**event-triggered**) without being requested
→ used in on-board communication

Request – Response (Client – Server)
ECU sends only, when requested
(one request – one response)
→ used in off-board communication

Publish – Subscribe
ECU starts broadcasting after receiving a single request → used in calibration



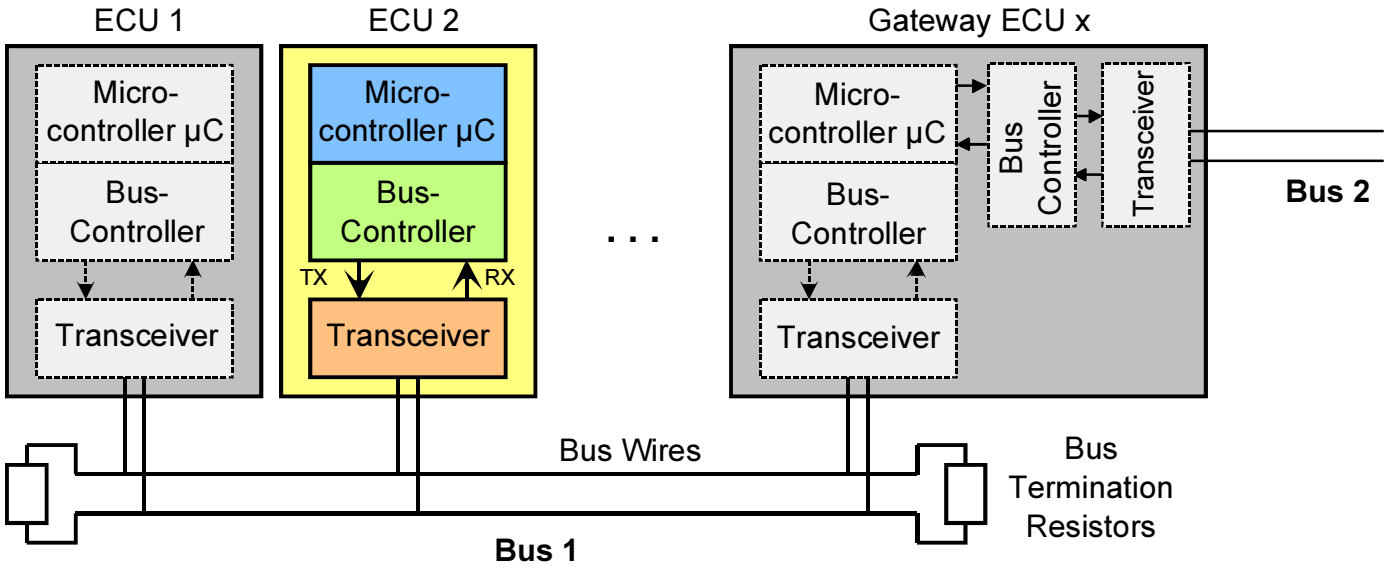
Receiver feedback

- Unacknowledged: No feedback
  → used in on-board communication (at least, if no errors occur)
- Acknowledged: Positive or negative feed-back from receiver to allow error handling. ACK/NAK can be included in frame or in response to reduce bus load.

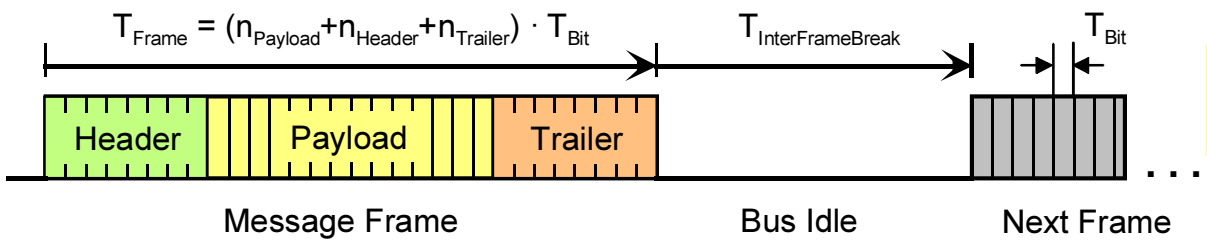


- **Error detection** by frame format, check sum and time out monitoring
- **Error handling** by ignoring faulty messages or automatic retransmission (reliable transmission)

Major Network Components



- **Microcontrollers** send data, process received data and implement protocol layers 4 to 7
- **Bus-Controllers** (Communication Controller CC, Media Access Unit MAC) implement layers 1 & 2
  - add message header (addressing, protocol control information PCI) and trailer (error detection) to payload (user data)
  - parallel – serial conversion of messages → bit stream



Bit Rate

$$f_{Bit} = \frac{1}{T_{Bit}}$$

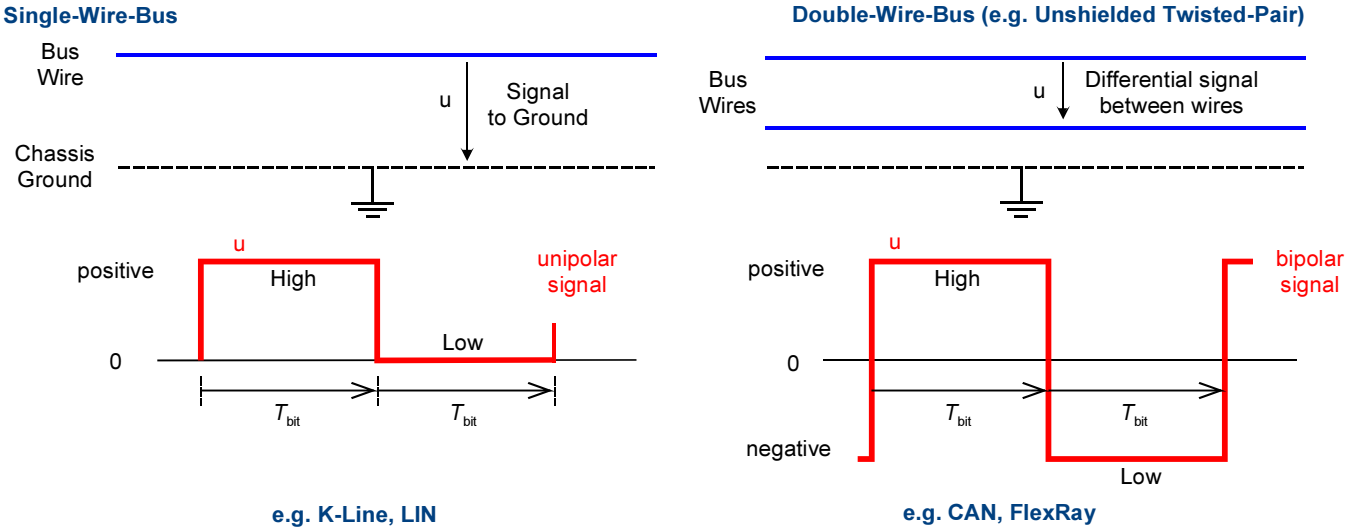
Unit: bit/s

Net Data Rate

$$f_{Data} = f_{Bit} \cdot \frac{n_{Payload}}{n_{Payload}+n_{Trailer}+n_{Header}}$$

with n_x ... number of payload, header or trailer bits

- **Transceivers** (Bidirectional Signal Driver with Transmit TX – Receive RX, Physical Bus Interface PHY) convert between signal levels on the bus wires and ECU internal signals
- **Single-wire busses** with unipolar signal to ground (low-cost) or **double-wire busses** with bipolar differential signal (better noise immunity → higher bit rates)



Wiring is critical for high-speed networks: Termination resistors required

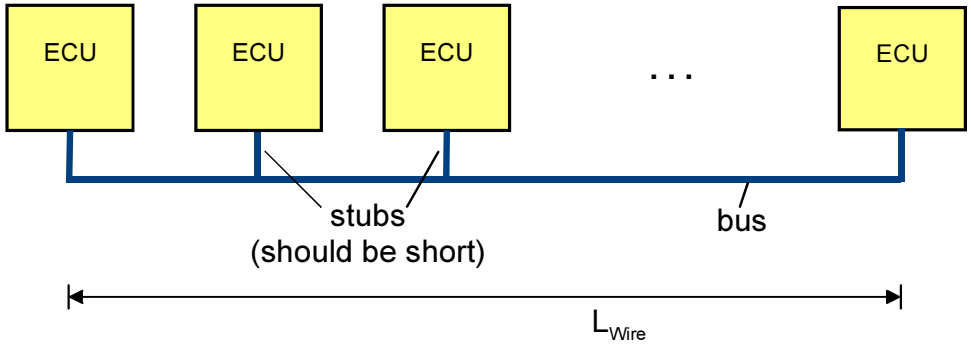
Signal propagation delay $T_{Prop} = T_{Transceiver,\ Sender} + T_{Transceiver,\ Receiver} + T_{Wire} << T_{bit}$

with $T_{wire} = \frac{L_{Wire}}{c_0/\sqrt{\varepsilon_R}} = \frac{L_{Wire}}{10 \ldots 15\ cm/ns}$, where L_{Wire} … length of wire between sender and receiver

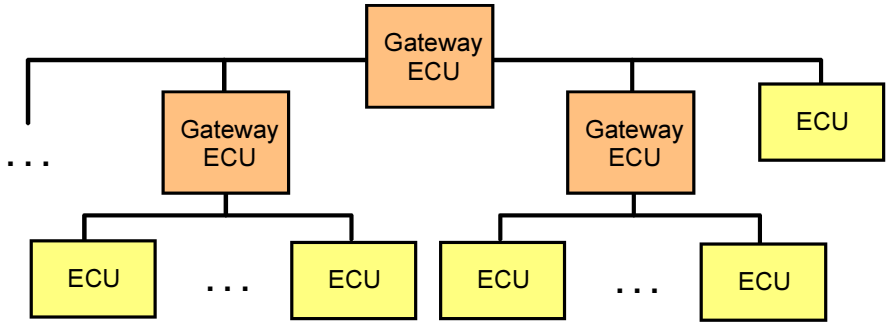
- **Gateways** decouple bus systems of different technology (protocol conversion, e.g. CAN - LIN) and/or different bit rate (bit rate conversion with message buffering, e.g. between high-speed and low-speed CAN). Presently routers, switches and hubs not used in automotive systems.

Network Topology

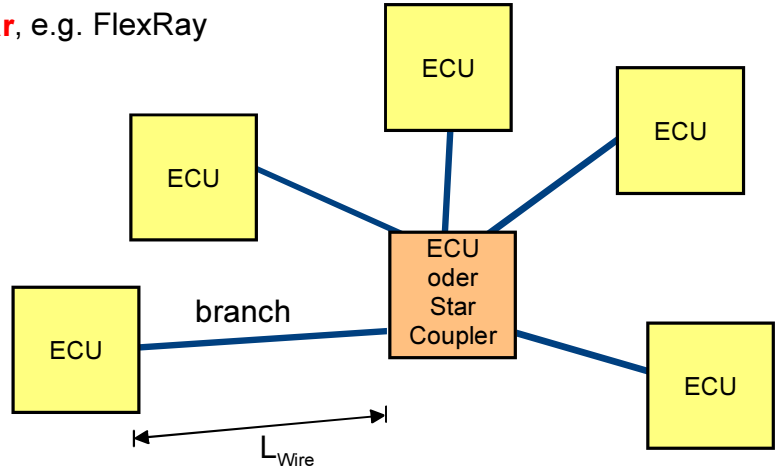
Linear Bus, e.g. CAN, optional for FlexRay



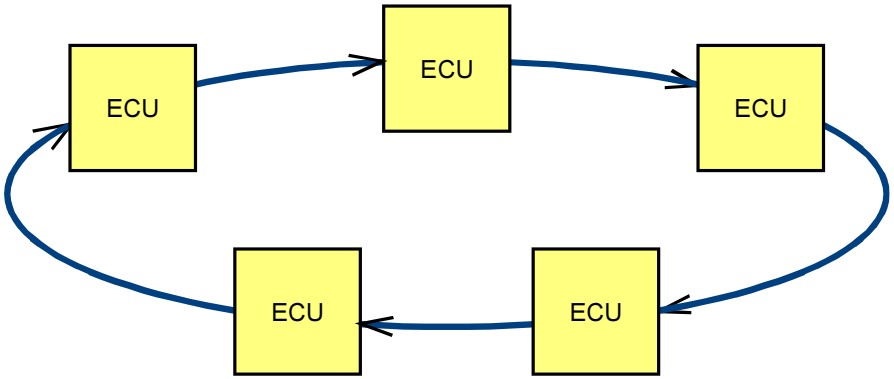
Tree, e.g. complete networks with subbusses



Star, e.g. FlexRay



Ring, e.g. MOST (unidirectional optical ring)



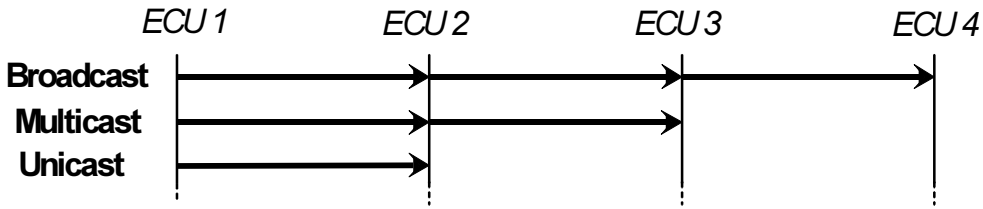
Message Addressing

In a bus system, technically any node can receive all messages. A node decides, which messages need to be processed, based on addresses, i.e. unique numbers contained in the message header. Two major addressing modes are used in automotive networks:

- **Node (station) based addressing**: Each node has a unique number, called **address**. The sender includes the address of the receiving node in the message header. Additionally the sender's address is included in most cases. Typical e.g. for Ethernet

  Used in higher-layer protocols, e.g. diagnostic off-board communication OBD, when one sender communicates with exactly one receiver (**1:1 or unicast communication**).

  Special addresses are used in case that a message shall be send to more than one node (**1:n or multicast communication**) or to all nodes (**1:all or broadcast communication**).



  In OBD or UDS diagnostic protocols a unique node address is called **Physical Address** and a multicast address, i.e. an address for a certain group of ECUs, is called a **Functional Address**.

- **Content based addressing**: Each message has a unique number, called **message or frame identifier**, identifying the message content. Any node interested in the content, can process the message. As each message identifier has exclusively one sender, a sender address is not required. Transmission with content based addressing is a special case of broadcast.

  Used in automotive bus systems like **CAN, LIN and FlexRay**.

Network Access

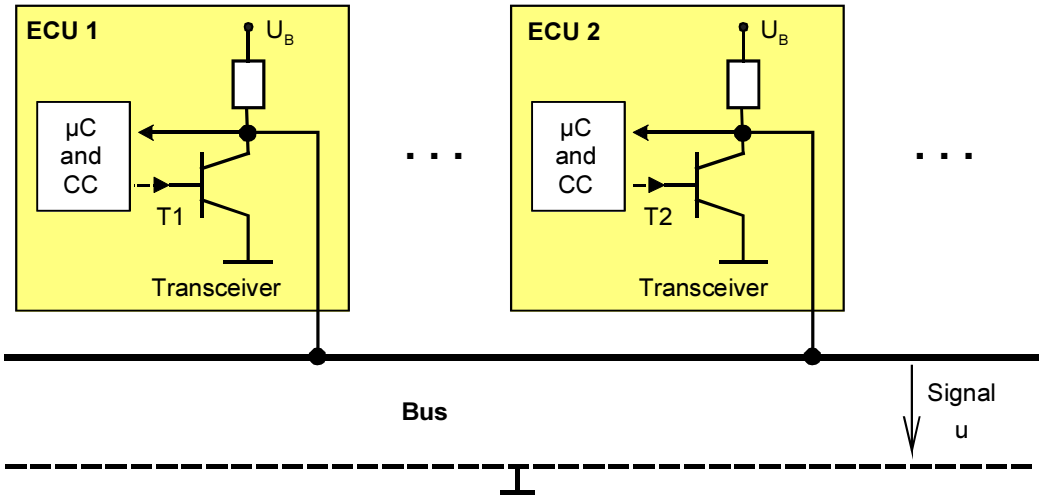
The network is a shared media, which may only be accessed by one sender at a time:

Managed Network Access		Arbitrary Network Access	
Synchronous		Asynchronous	
Centralized: **Master-Slave**	Decentralized: **Time-triggered Multi-Master**	**Event-triggered Multi-Master** Sender waits, till bus is free and sends …	
		with Collision Detection **CSMA/CD** Carrier Sense - Multiple Access / Collision Detect	with Collision Resolution **CSMA/CR** Carrier Sense - Multiple Access / Collision Resolution
A single Master-ECU requests all other ECUs (Slaves) to send one at a time according to a predefined schedule	Exclusively reserved time slots (Schedule) of fixed (**TDMA** Time Division Multiple Access) or variable length (**FTDMA** Flexible TDMA) for each sender	In case of a collision all senders stop sending and retry later	In case of a collision, one sender dominates (**Priority-based Collision Resolution**), other senders stop sending and retry later
e.g. **LIN**	e.g. **FlexRay**	e.g. **Ethernet**	e.g. **CAN**
Problems: • Communication breakdown, if master fails	Problems: • Common net-wide time base required • Free time slots cannot be used by other senders • Slots for new messages be reserved to avoid schedule modification in all ECUs	Problems: • Bus-access non-deterministic for all messages • Communication breakdown at high bus loads due to permanent collisions → Problem for real-time communication	Problems: • Low-priority messages are non-deterministic • Max. bit rate limited by time required for collision resolution

Collision Detection and Resolution:

- Typical Bus-Interface:
  Logic level "H" → u=U_B
  Logic level "L" → u=0

  e.g. K-Line, LIN:
  U_B … battery voltage



- Bus Idle or ECUs sending "H": T1:off and T2:off → Signal u = U_B → "H" OK
  ECU 1 sending „L", ECU 2 idle: T1:on and T2:off → Signal u = 0 → "L" OK
  ECU 1 sending „L", ECU 2 trying to send „H"
  T1:on and T2:off → Signal u = 0 → "L" Collision
  ⇒ „L" *wins* → **dominant** signal level „H" *looses* → **recessive** signal level
- **Collision Detection**
  All senders read back the bus signal and compare it with their desired signal level.
- **Collision Resolution**
  The ECU, which detects a collision, stops sending. The other ECU will continue its transmission without failure.

Automotive Networks

On the following pages, the most important automotive networks will be discussed. The description will be short with respect to …

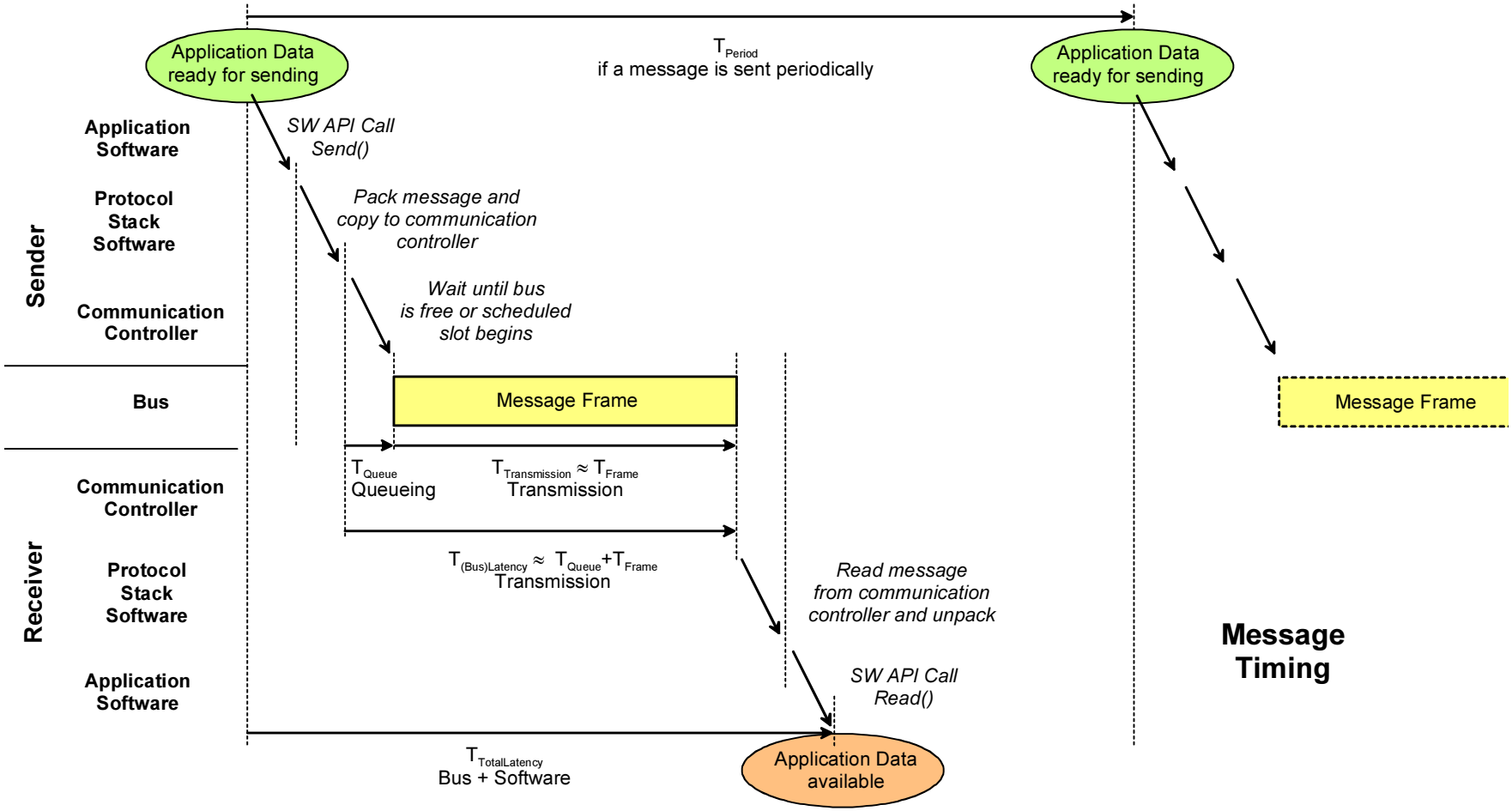
… Hardware Aspects:

- Bitrate (bus bandwidth)
- Bus topology and wiring requirements, bus length limitations
- Transceiver circuits
- Electromagnetic compatibility

We will look more detailed into …

… System aspects:

- Net data rate (“effective bus bandwidth”)
- Frame format → addressing mechanisms, data length
- Message latency and jitter ← consequences of the bus access methods
- Error detection on the sender and receiver side → bit monitoring, frame format checks, checksums, timeouts
- Error signaling and handling to other bus stations and to the application software layer → send error frames, ignore faulty data, repeat transmission, …



- When calculating message latency here, we only consider the communication bus and neglect the delay caused by software (which often is much greater than the bus latency).
- Jitter = statistical tolerance of $T_{Latency}$ (mostly) due to variable queuing times, i.e. $T_{Jitter} = T_{Latency,max} - T_{Latency,min}$
- If a message is sent periodically, T_{Period} is the time between two sending attempts, sometimes also called *sending interval* instead of *sending period*. For non-period messages, T_{Period} is the minimum time between sending events.

3.3 Controller Area Network CAN Classic, CAN-FD and CAN-XL

Bit rate	f_{Bit} = 10 kbit/s … 1 Mbit/s (CAN Classic) … 5 … 10 Mbit/s (CAN FD/XL)
Bus Topology	Linear double-wire bus (CAN_H, CAN_L), single-wire option for low bit rates only
Bus Length	Max. bus length L_{max} depends on bit rate f_{Bit}: $L_{max} \leq 40 \ldots 50\text{m} \cdot \frac{1\ \text{Mbit/s}}{f_{Bit}}$
History	First Class C automotive network. Developed by Bosch, licensed to many microcontroller manufacturers. First use in Mercedes-Benz S-Class 1991, since mid 1990s most frequently used automotive network.
Usage	Passenger cars • Europe: since mid 1990s for on-board, since 2000 for off-board communication • US: since 2000 for on-board, since 2008 for off-board communication (OBD) mandatory Trucks: • since mid 1990s worldwide truck standard **SAE J1939** (layer 1 … 7)
Public standard	**ISO 11898** (layer 1 and 2) [S2] • Bosch's Data Link Layer in two variants **CAN 2.0A** and **2.0B** [S1] • Various incompatible physical layers **High-speed CAN**: typ. 500 kbit/s, max. 1 Mbit/s **Low-speed CAN**: typ. 100 kbit/s, max. 125 kbit/s Single-wire CAN: 33 kbit/s (GM), 83 kbit/s (Chrysler), outdated

Physical Layer

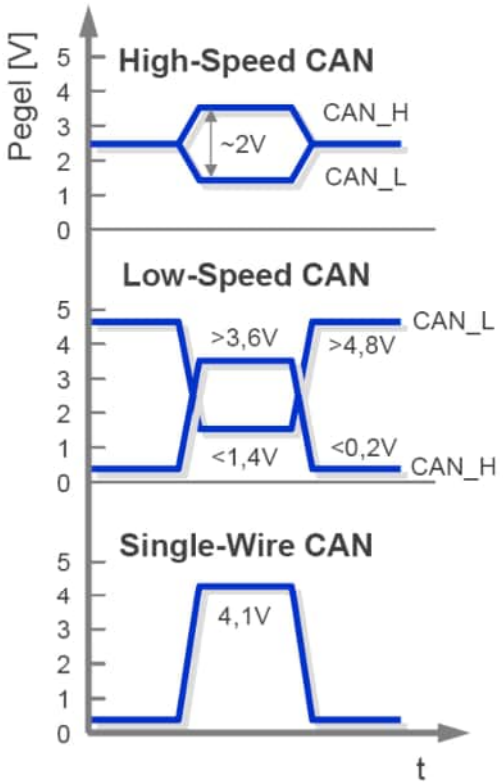
High-speed CAN

- requires termination resistors (120Ω) on both ends of the bus (often integrated in ECU)
- stubs should be short (< 30cm @ 1Mbit/s)

Low-speed CAN

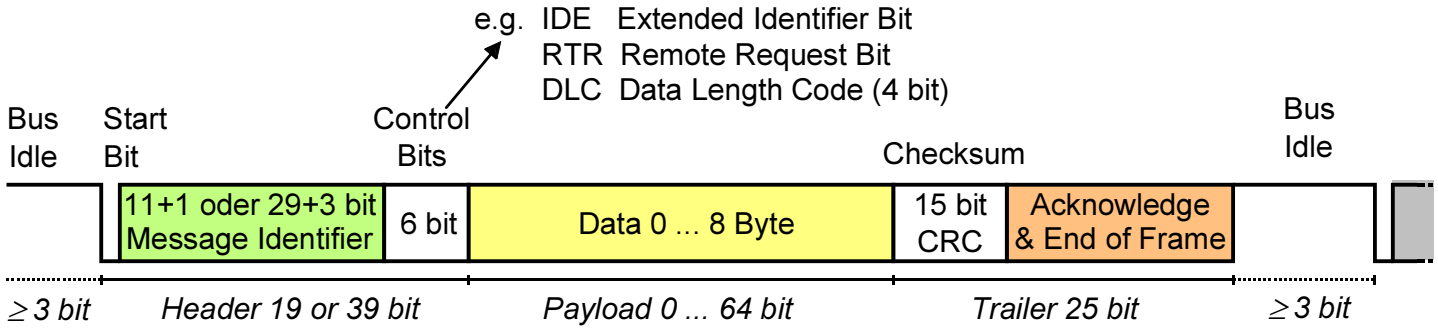
- fault-tolerant to a single broken wire and to short circuit between wires or from one wire to ground
- tolerates long stubs, i.e. star and tree like topologies

Signal levels



CAN Classic Frame Format

(see also note *1 and *2 below)



Max. 8 data bytes per frame, data length indicated by DLC field in header.

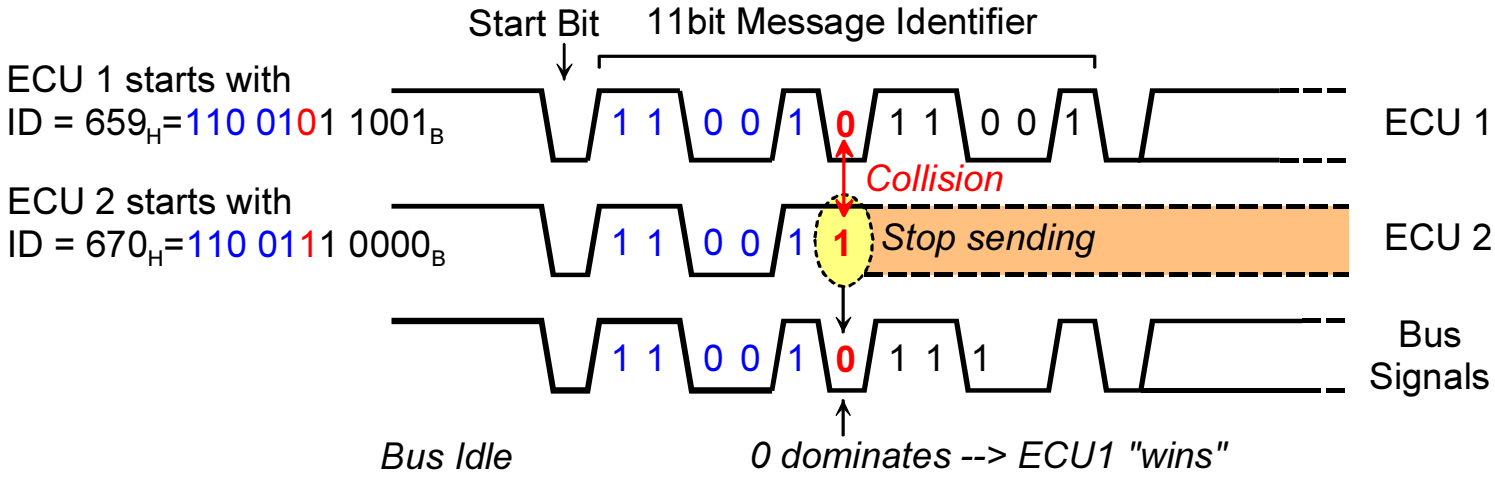
Addressing

Content-based with **11bit** (CAN 2.0A Standard Identifier) **or 29bit** (CAN 2.0B Extended Identifier) **message identifier** (message ID, CAN ID). Both identifier lengths can be mixed in the same network.

Network Access

Event-triggered communication with **CSMA/CR** bus access, **message ID defines message priority** in case of collisions (lower ID = higher priority)

Example



The higher priority message "wins" the arbitration, the lower priority message will be retransmitted automatically, when the bus is idle again.

Message ID selection for on-board-communication is done by the OEM. For OBD diagnosis IDs are standardized by ISO 15031 [S11], for truck on- and off-board applications by SAE J1939 [S12].

Error Detection and Handling

Receivers (and senders) check frame format and CRC checksum. Receivers confirm fault-free reception by sending an *ACK* bit (included in the trailer field). In case of a failure, the receiver sends an *Error Frame* (six dominating 0 bits) instead of the *ACK* bit and the following *End of Frame* bits.

In case of an *Error Frame,* **all** receivers do ignore the faulty message.

If the **sender** does not get an *ACK* bit or does receive an *Error Frame*, the sender **automatically retransmits** the faulty message.

To avoid continuous retries or permanently sending false *Error Frames*, each communication controller has several error counters. It automatically stops communication after a predefined number of faults (States: Error active = Node does check messages and send *ACK* bits and *Error Frames*, Error passive = Node does no longer send ACK bits and Error Frames in order not to disturb others – Bus Off = Node does not send or receive messages any more).

Errors are indicated to the application via communication controller status bits or interrupts. Timeout monitoring on application layer.

Note *1

Bit Stuffing: The communication controller adds/removes additional bits on the physical layer to ensure bit clock synchronization between sender and receiver. Theoretically, for each 5 bits in a frame one stuff bit could be added (max. 24 (29) stuff bits for 11 (29) bit CAN ID messages with 8 data bytes, i.e. about 25% worst-case stuffing overhead) to avoid long sequences of 0s or 1s on the bus wires, i.e. no signal edges to identify bit timing. Statistical investigations showed that the average message length does only increase by about 15% [T2].

Note *2

For data blocks > 8 byte Classic CAN needs a higher-layer **transport protocol**, which splits the data block into 8 byte segments (**segmentation**)

Frame Length

$$T_{Frame} \le \frac{1.25\ (n_{Data} + n_{Header} + n_{Trailer})}{f_{Bit}}$$

with n_{Data} data length (in bits)

$n_{Header} + n_{Trailer}$ length of header+trailer in bits
11bit CAN ID: (19+25=44) bit
29bit CAN ID: (39+25=64) bit

Net Data Rate (Bus bandwidth)

$$f_{Data} = \frac{n_{Data}}{T_{Frame}} = \frac{n_{Data}}{1.25\ (n_{Data} + n_{Header} + n_{Trailer})} \cdot f_{Bit}$$

Note: Factor 1.25 to take into account worst-case bit stuffing overhead of about 25%.

Example: High-Speed CAN f_{bit} = 500 kbit/s, 11 bit CAN ID (29 bit CAN ID)

a) 1 data byte → n_{Data} = 8 bit → $T_{Frame,\ min} = \frac{1.25\ (8+44)\ bit}{500\ kbit/s} \approx 130\mu s$

$$f_{Data,\ min} = \frac{8\ bit}{1.25\ (8+44)\ bit} \cdot f_{Bit} = 0.12 \cdot f_{Bit} \approx 62\ kbit/s \approx 8\ KB/s$$

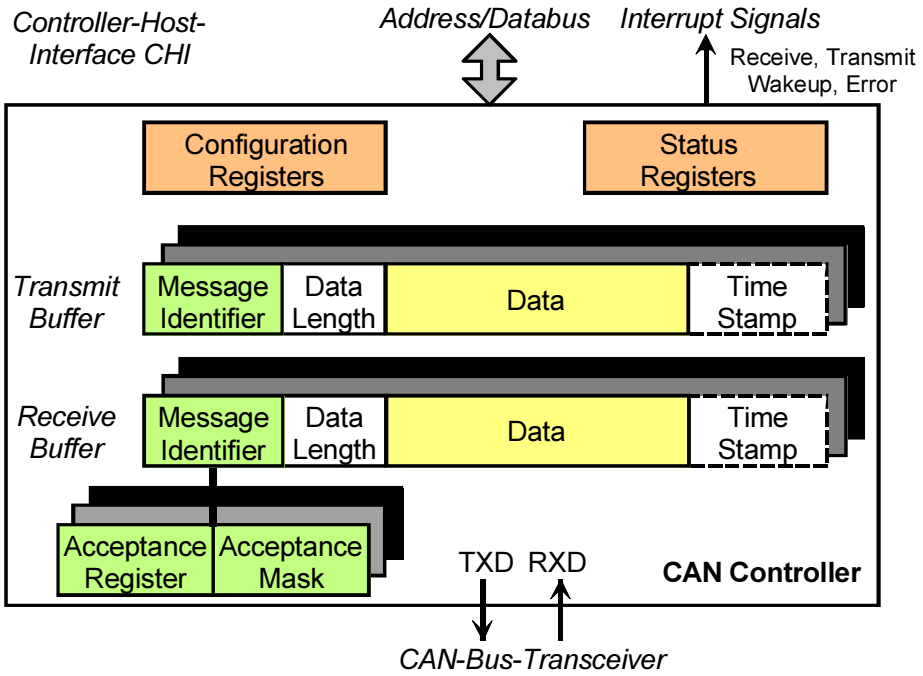
b) 8 data byte → n_{Data} = 64 bit → $T_{Frame,max} = \frac{1.25\ (64+44)\ bit}{500\ kbit/s} \approx 270\ \mu s$ (≈ 320µs)

$$f_{Data,\ max} = \frac{64\ bit}{1.25\ (64+44)\ bit} \cdot f_{Bit} = 0.5 \cdot f_{Bit} \approx 240\ kbit/s \approx 29\ KB/s$$

($0.4 \cdot f_{Bit} \approx 200$ kbit/s ≈ 24 KB/s)

Best case protocol efficiency 50% (40%) - Protocol overhead 50% (60%)

Message Reception and Acceptance Filters



Typical CAN controllers

- can send or receive messages with 11bit and/or 29 bit identifiers
- have several transmit and receive buffers, to/from which messages are written/read
- receive message detection via polling by host µC or by interrupt

To reduce µC load at heavy bus traffic, most CAN controllers can be configured to receive only messages with predefined CAN IDs (**Acceptance Filtering**). Typically for each receive buffer a must match / don't care filter can be set up.

If a new message with the same CAN ID arrives, the old message is overwritten.

Example: A CAN buffer shall accept all CAN messages in the range CAN ID_{11bit} = $7D0_H$ … $7DF_H$

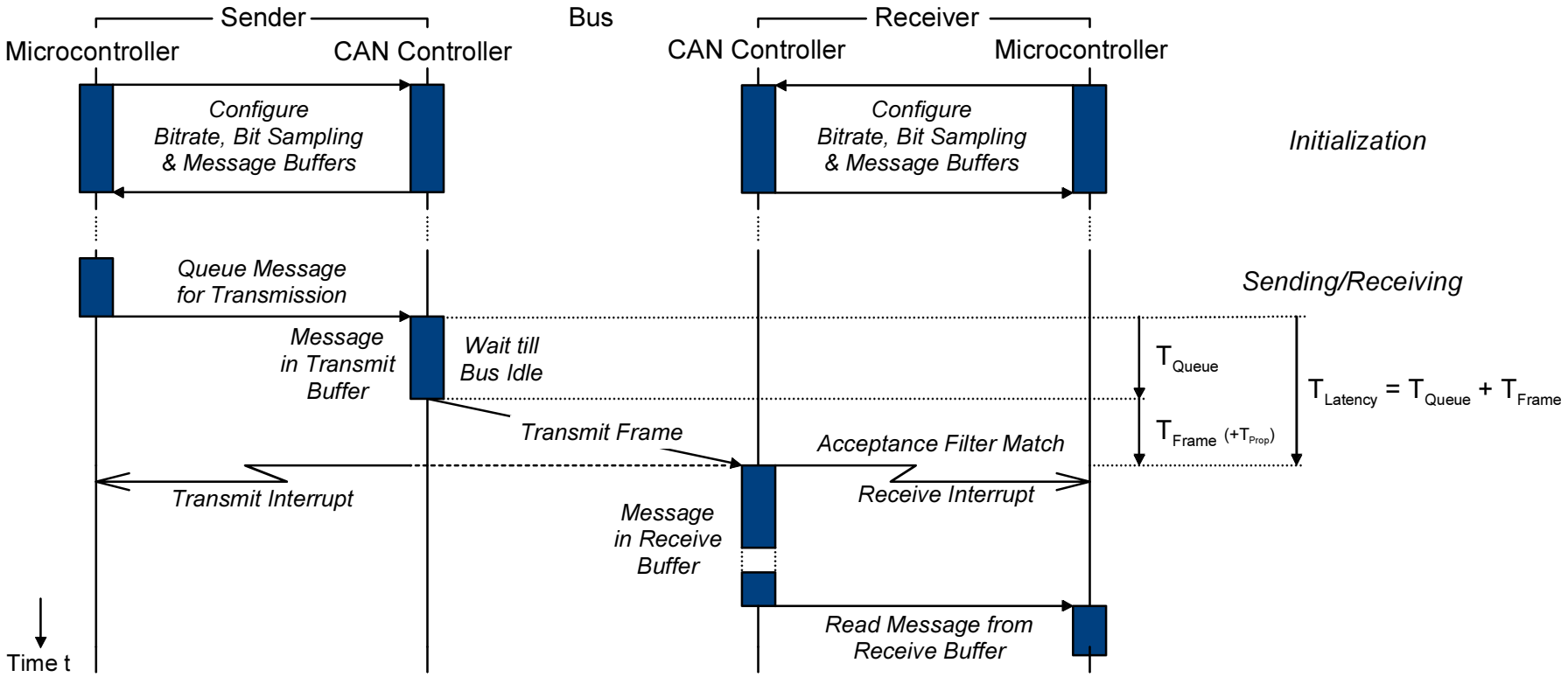
$7D0_H$ = $111\ 1101\ 0000_B$

$7DF_H$ = $111\ 1101\ 1111_B$

Filter: $111\ 1101\ XXXX_B$

Controllers without hardware acceptance filters are called ***Basic CAN*** controllers. All newer CAN controllers have hardware acceptance filters: ***Full CAN*** controllers.

CAN Message Transfer Sequence



Bus Load (Usage)

- Assume: i=1…N frames of length $T_{Frame,i}$ to be sent at intervals ≥ T_i , no bus errors

$$\Rightarrow \quad \text{Bus Load BL} = \sum_{i=1}^{N} \frac{T_{Frame,i}}{T_i} \qquad \text{should be} \ll 100\% \text{ in a production system} \quad (1)$$

Message Latency

$$T_{Latency} = T_{Queue} + T_{Frame} \quad (2)$$

- Assume: i=1…N frames of length $T_{Frame,\,i}$ to be sent at intervals $\geq T_i$
  no queuing jitter, no bus errors
- Best case: Message m sent immediately: $T_{Queue,m} = 0$ → $T_{Latency,m} = T_{Frame,\,m}$
- Worst case for highest priority message:
  Message sent, after current message l on the bus has been transmitted:
  → $T_{Latency,m} = T_{Frame,m} + \max(T_{Frame,\,i})$
- Worst case for other messages
  Message sent, after current message on the bus AND all higher priority messages have been transmitted

During $T_{Queue,m}$, while message m is waiting for the bus, in worst case a higher priority message k will be transmitted several times, if its sending interval is $T_k < T_{Queue,m}$, thus

$$T_{Queue,m} \leq \max(T_{Frame\ of\ other\ messages}) + \Sigma\, T_{Transmission\ of\ higher\ priority\ messages}$$

$$= \max_{i \neq m}(T_{Frame,i}) + \sum_{k=\text{all higher priority messages}} \left\lceil \frac{T_{Queue,m}}{T_k} \right\rceil T_{Frame,k} \quad (3)$$

Unfortunately, as $T_{Queue,m}$ occurs on both sides, this equation can only be solved iteratively. Starting with $T_{Queue,m}$= $\max(T_{Frame})$, the old value of $T_{Queue,m}$ is inserted on the right side and the new result is calculated, until the value does not change significantly any more. If the calculation does not converge, the latency is unbound, i.e. the bus is overloaded. $\lceil T_{Queue,m}/T_k \rceil$ must be rounded up to the next larger integer (in C: ceil() function).

Example:

Message m	CAN ID	Priority	T_{Frame}	Sending Interval T	Worst Case $T_{Latency} = T_{Queue} + T_{Frame}$
A	115	Highest	1 ms	2.5 ms	1 ms + 1 ms = 2 ms
B	347		1 ms	5 ms	2 ms + 1 ms = 3 ms
C	572	Lowest	1 ms	10 ms	4 ms + 1 ms = 5 ms

Bus Load: $BL = \frac{1\text{ ms}}{10\text{ ms}} + \frac{1\text{ ms}}{5\text{ ms}} + \frac{1\text{ ms}}{2.5\text{ ms}} = 70\%$ ok, but high!

Message m	$T_{Queue,m} = \max_{i \neq m}(T_{Frame,i}) + \sum_{k=\text{all higher priority messages}} \lceil \frac{T_{Queue,m}}{T_k} \rceil T_{Frame,k}$			
	Initial	*1st iteration*	*2nd iteration*	*3rd iteration*
	[all values in ms]			
A	1 ms	-	-	-
B	1 ms	$1 + \lceil \frac{1}{2.5} \rceil \cdot 1 \rightarrow 2$	$1 + \lceil 2/2.5 \rceil \cdot 1 \rightarrow 2$	-
C	1 ms	$1+\lceil 1/2.5 \rceil \cdot 1 + \lceil 1/5 \rceil \cdot 1 \rightarrow 3$	$1+\lceil 3/2.5 \rceil \cdot 1 + \lceil 3/5 \rceil \cdot 1 \rightarrow 4$	$1+\lceil 4/2.5 \rceil \cdot 1 + \lceil 4/5 \rceil \cdot 1 \rightarrow 4$

Message Priority Selection

- Message priorities have a major influence on worst case message latencies
- In any case, message latency should be smaller than the minimum required send interval of the message, i.e. $T_{Latency,max} < T_{Period,min}$, otherwise, the message could be overwritten in the transmit buffer before being sent.
- Simple priority selection strategy

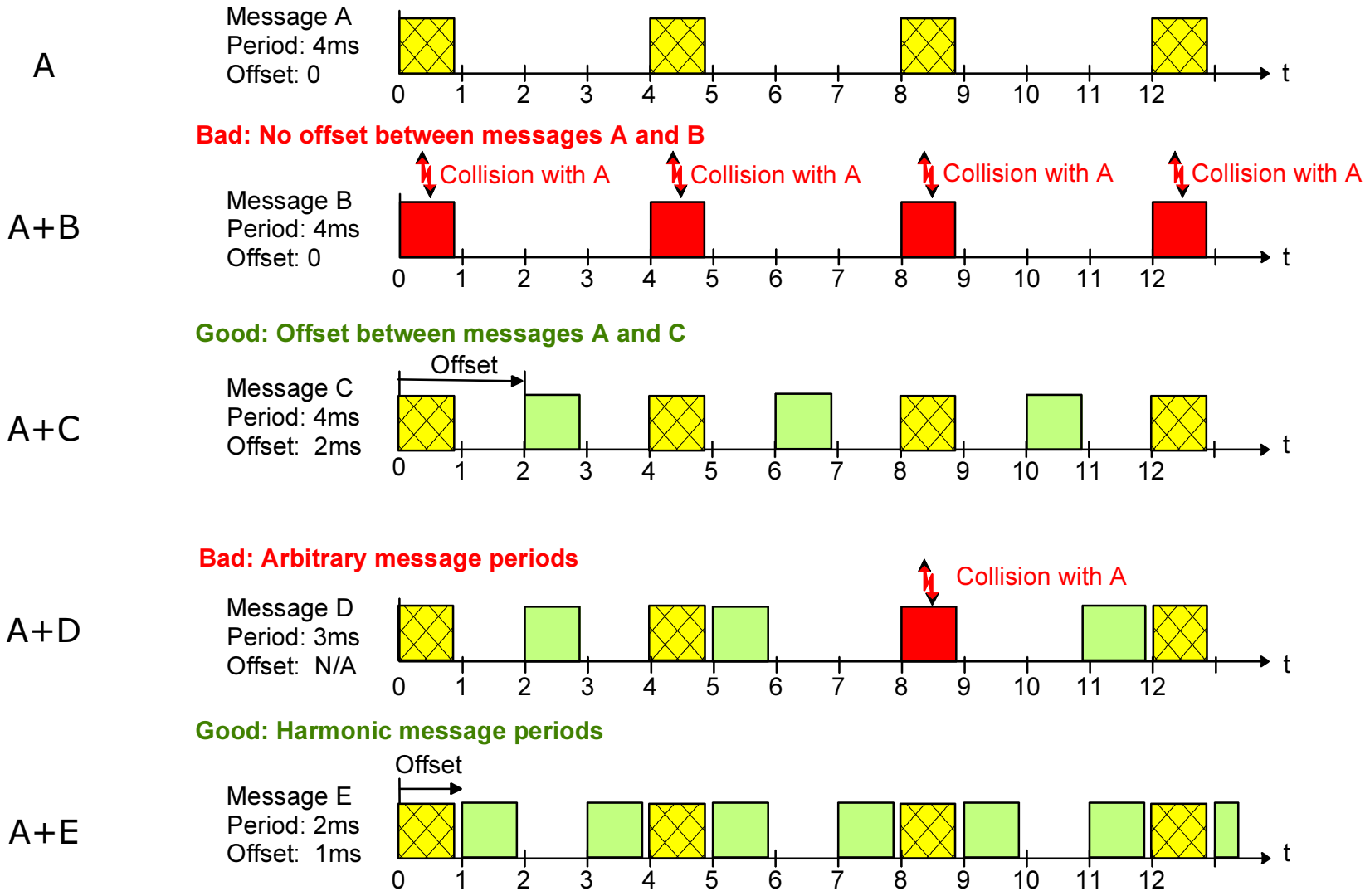
  The smaller the required latency or message period is, the higher the message priority shall be (**Deadline Monotonic Rate (DMR) Scheduling**)

Jitter of Periodic Messages (see following page)

- Periods of periodic messages should be harmonic multiples, e.g. 1 ms, 5 ms, 10 ms, ...
- Messages should be sent with appropriate offsets, so that queuing several messages at the same instant will be avoided and thus jitter be reduced.

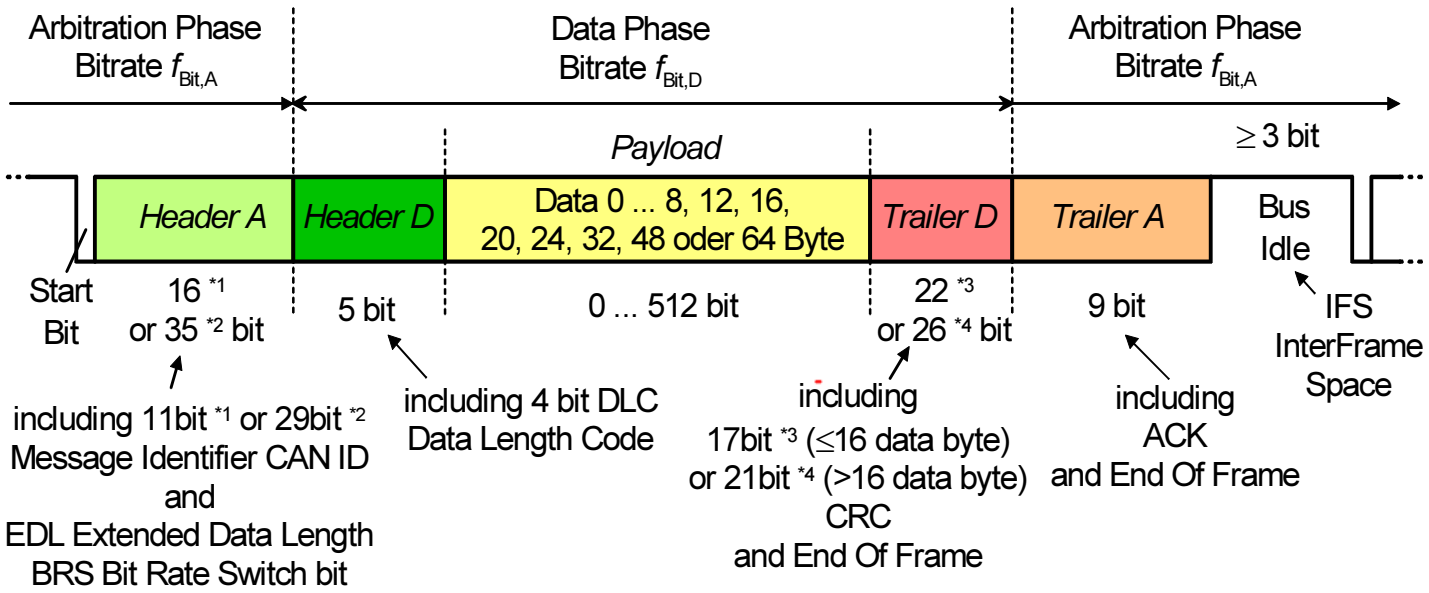
The discussion of message latency and priority selection presented here uses several simplifying assumptions. For a more detailed analysis including bus errors, variable message lengths, bit stuffing effects etc. and a more advanced priority selection algorithm proposed by Audsley see ref. [T1, T2].

Example of a message schedule with and without collisions



CAN Flexible Data-rate CAN-FD

CAN FD
Frame Format



To increase CAN's bandwidth limit, two modifications were made:

- **Longer message length** up to 64 bytes per message (indicated with new EDL header bit and requires modified CRC). Data length is 0…8, 12, 16, 20, 24, 32, 48 or 64 bytes.
- **Higher bitrate for** the major **Data Phase of the CAN message** (indicated with new BRS header bit). Due to the bitwise arbitration of the CAN message identifier in the header and the message acknowledgement in the trailer the involved bits must have a minimum length and thus the bitrate is limited (see equation on page 16). For the rest of the message this limitation does not apply, so CAN-FD may dynamically switch between the arbitration bit-

rate, e.g. $f_{Bit,A}$ = 500 kbit/s, and a higher rate for the data phase, e.g. $f_{Bit,D}$ = 4 Mbit/s. The higher bitrate includes the second part of the trailer and the first part of the header section.

CAN-FD frames can be mixed with CAN Classic on the same bus, but all ECUs must use a CAN-FD compatible communication controllers. CAN Classic communication controllers would send an Error Frame when receiving a CAN FD frame. Bit rates $f_{bit,D}$ up to approx. 2 Mbit/s can be reached with standard CAN transceivers. For bitrates up to 8 Mbit/s all ECUs must use new SIC (Signal Improvement Capability) bus transceivers. In combination with future CAN XL (see below) and modified bus signal levels 12 Mbit/s shall be achieved. In all cases, the maximum bitrate depends on the bus wiring harness (length of stubs, quality of cable).

Frame Length

$$T_{Frame,FD} \le 1.25 \left[\frac{n_{Header,A}+n_{Trailer,A}}{f_{Bit,A}} + \frac{n_{Data} + n_{Header,D}+n_{Trailer,D}}{f_{Bit,D}} \right] \quad *1$$

with n_{Data} data length (in bits)

$n_{Header,A}+ n_{Trailer,A}$ length of header+trailer in bits @ arbitration phase bitrate $f_{bit,A}$

26 bit for 11bit CAN IDs
45 bit for 29 bit CAN ID

$n_{Header,D}+ n_{Trailer,D}$ length of header+trailer in bits @ data phase bitrate $f_{bit,D}$

27 bit with 17 bit CRC for ≤ 16 data byte,
31 bit with 21 bit CRC for > 16 data byte

Stuffing in the CRC field in Trailer D has been slightly modified vs. CAN Classic, but the factor 1.25 for worst-case bit stuffing can still be used.

The first CAN FD specification issued by Bosch in 2012 [S1] contained a bug which made CRC error detection fail under some rare conditions. Thus the ISO 11898 standardized CAN FD was modified and is not fully compatible with the original Bosch version, which you find in older publications.

T_{Frame} calculation example:

	CAN Classic 11 bit CAN ID @ f_{Bit} = 500 kbit/s	**CAN FD** 11 bit CAN ID @ $f_{Bit,A}$ = 500 kbit/s, $f_{Bit,D}$ = 4 Mbit/s
8 data bytes	$1.25\ [\frac{(64+44)\text{bit}}{500\text{kbit/s}}] \approx 270\ \mu s$ (see pg. 3.20)	$1.25\ [\frac{26\text{bit}}{500\text{kbit/s}}+\frac{(64+27)\text{bit}}{4\text{Mbit/s}}] \approx 95\mu s$
64 data bytes	Not possible, requires transport protocol and multiple frames (see section 3.9)	$1.25\ [\frac{26\text{bit}}{500\text{kbit/s}}+\frac{(512+31)\text{bit}}{4\text{Mbit/s}}] \approx 235\mu s$

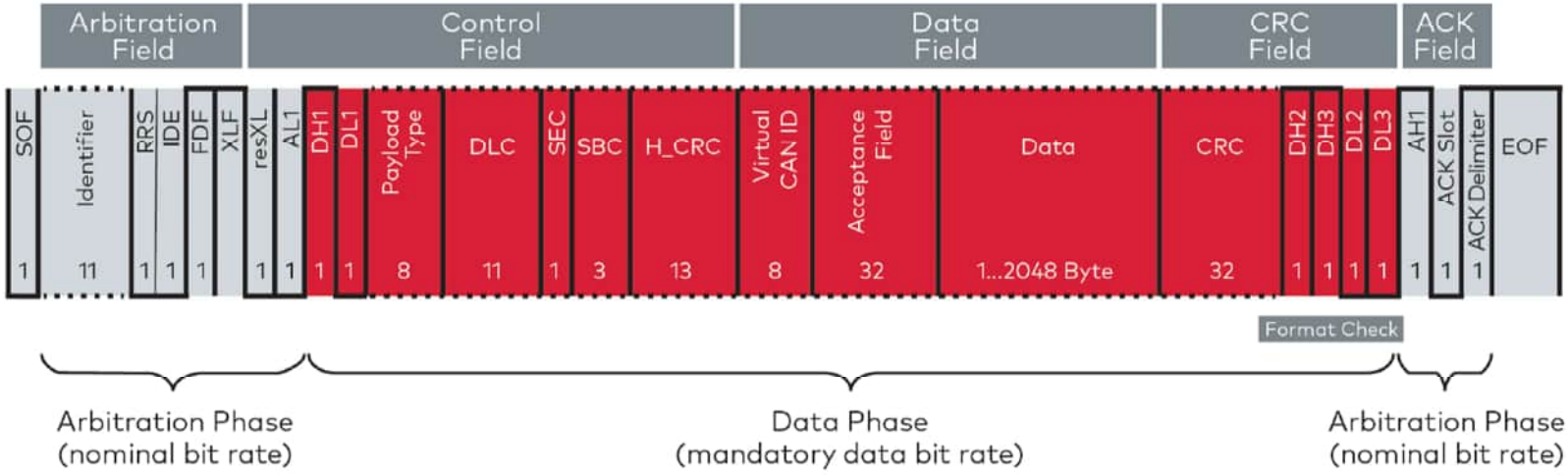
CAN FD Light

As an alternative to LIN (see section 3.4) STMicroelectronics has proposed a low-cost version of CAN FD. The lower cost is achieved by operating with constant bitrate (no bitrate switching) and a Master-Slave concept (no arbitration), so that Slave ECUs can be implemented in hardware only, i.e. without microcontroller, and without crystal quartz.

Status: Today, it is unclear if there is a market for such a solution and if standardization will be successful.

CAN-XL extension for tunneling Ethernet/IP via CAN

- Proposed extension for CAN FD with 1 … 2048 (!) byte payload data as alternative to Ethernet 10-Base-T1S (see section 3.7). Up to 12 Mbit/s during data phase (Note: Estimated frame length 1.4 ms (!) @ 12 Mbit/s with 2048 byte data)
- 11 bit CAN ID used for arbitration priority only. 32 bit Message ID in new Acceptance Field. Up to 256 virtual CAN networks possible.
- Modified header to include Ethernet-like info like protocol type, intended to tunnel Ethernet/IP frames via CAN XL. Additional CRC for header field for better error protection.

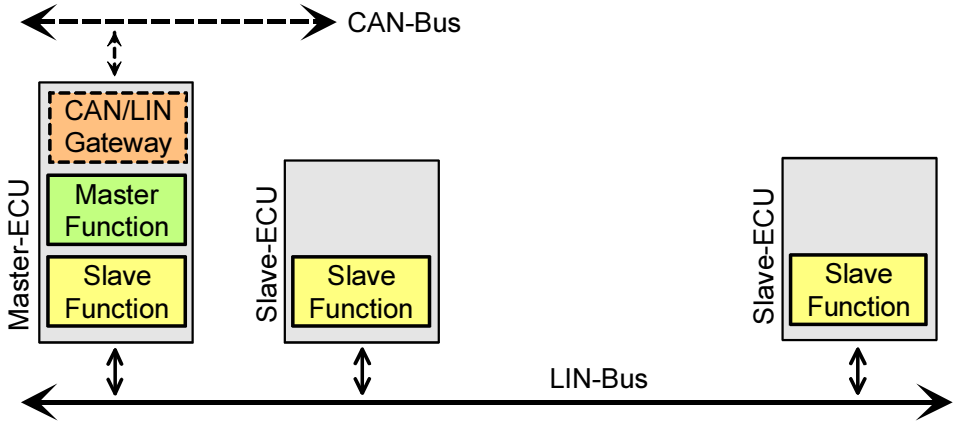


Source: O. Garnatz, P. Decker (Vector Informatik)

- Status: Proposal by VW, formal specification work in progress.

3.4 Local Interconnect Network LIN

Bit rate	$f_{Bit} \leq 20$ kbit/s
Topology	Single-wire bus, linear or star topology Master-ECU typically acts as gateway to CAN
Physical Layer	Uses the old ISO 9141 K-Line physical layer (with minor modifications) with "H"=U_{Bat} and "L"=GND signal levels, see page 13



Master-Function: Send message header to trigger data transfer
Slave-Function: Send/Receive Data

History and Usage	Class A automotive network. Developed by Freescale and several OEMs as a low-cost bus around 2000. Used as a CAN-subbus system for simple body applications like window lifters, mirror or light control.
Public standard	Industry standard by LIN Consortium (www.lin-subbus.org). Versions 1.3, 2.0 and 2.2 in use, further standardization handed over to **ISO 17987** [S3].

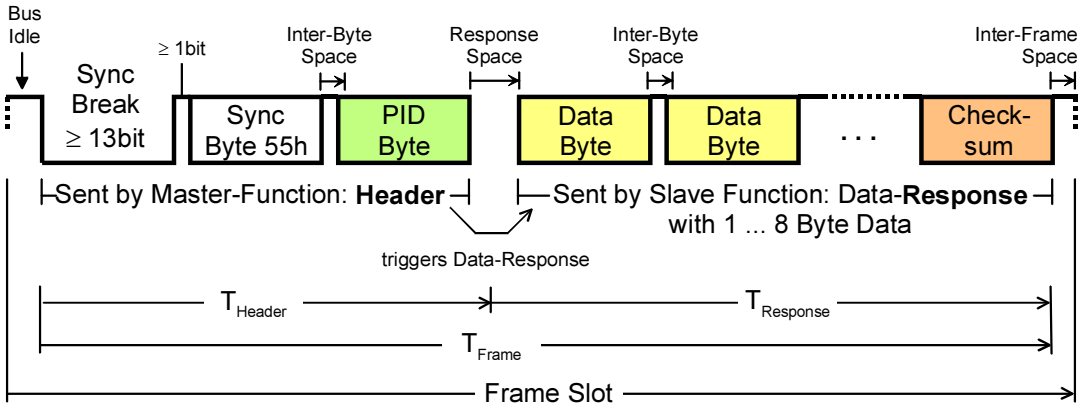
Network Access

Master-Slave System

All communication is initiated by the **Master**-function, which **sends** out the message **Header** according to a predefined schedule (see below). Similar to CAN, the header contains a **message identifier** byte **PID** defining the content of the following data. At the begin of the header a variable length Sync Break/Sync Byte sequence is sent to signal the start of the message.

Exactly one **Slave**-function is responsible to **send** the **response for** the requested **PID**. The response has 1 to 8 byte data plus a checksum byte. The data length is not sent, but has been fixed for the associated PID during the development phase of the network by the OEM.

Frame Format



LIN is **implemented by standard serial µC interfaces (UART/SCI)** plus software. On the physical layer, messages are sent as UART characters rather than a continuous bit stream. To each frame byte 1 start and 1 stop bit (i.e. 10 bits per byte) plus an inter-byte-space is added, which depends on how fast the sender software provides the message bytes.

Addressing

Content-based with LIN **Protected Identifier PID** = 0 … $3F_H$ (6 bit plus 2 parity bits, not shown). PIDs 0 … $3B_H$ can be assigned freely by the OEM. PID $3C_H$ (Master) and $3D_H$ (Slave) are reserved for the LIN transport protocol for data blocks larger than 8 byte (similar to the ISO TP for CAN, see chap. 3.9). For each PID the format of the message data is defined (so called *Unconditional Frame*).

Error Detection and Handling

Test checksum and timeouts between header and response and/or between messages. A node, which detects a fault, will ignore the message.

Further error handling and feedback about the fault from the receiver to the sender or the other nodes is not defined by LIN. The failed message will not be retransmitted, a message update however will be sent in the next cycle.

If the predefined Master-ECU fails, bus communication stops. LIN does not provide a concept for Slave-ECUs to automatically take over the Master role.

Frame Length

Header length $T_{Header,nominal} = 34 \cdot T_{bit} = 1.8$ ms

with (13+1) bit sync break + 2 bytes · 10 bit/byte = 34 bit

$T_{Header,max} = 1.4 \cdot T_{Header,nominal} = 2.5$ ms

Response length $T_{Response,\ nominal} = 10$ bit/byte $\cdot (N_{Data}+1$ byte$) \cdot T_{bit} = 4.7$ ms

with N_{Data} … Number of data **bytes**

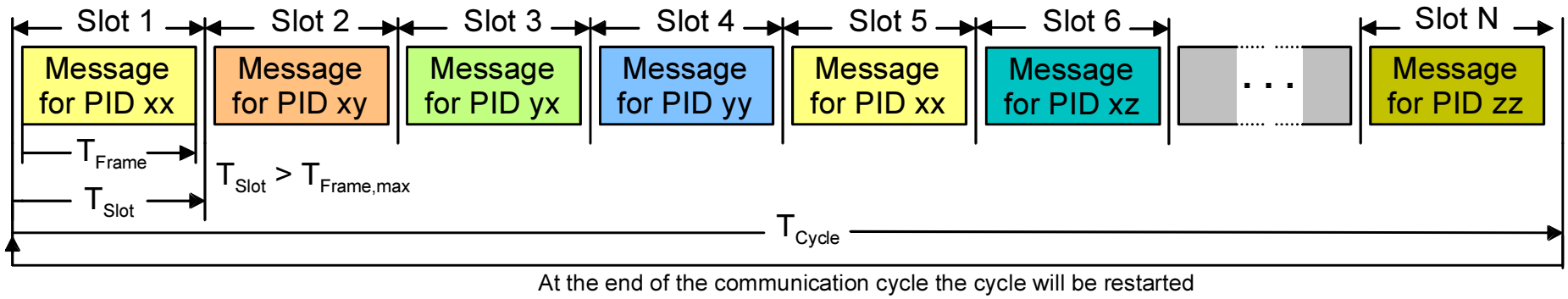
$T_{Response,max} = 1.4 \cdot T_{Response,nominal} = 6.6$ ms

Frame length $T_{Frame} = T_{Header} + T_{Response} = 6.5$ ms … 9.1 ms

Example: $N_{Data} = 8$ Byte, $f_{Bit} = 19.2$ kbit/s (“20kbit/s”) → $T_{Bit} = 52$ µs

Schedule

Typically, the communication scheme, called **schedule table**, is defined during the development phase of the system. A sequence of messages, which is periodically repeated, is called a **cycle**. Each message (identified by its PID and its associated sender ECU) is assigned to a certain **time slot**.



The length of these slots must be choosen so, that the associated LIN message can be transmitted safely within these slot (including a safety margin). To simplify the software implementation, one of two approaches can be used

- constant slot length: All slots of equal length, where

  $T_{Slot} > \max(T_{Frame,\,i})$, e.g. T_{Slot} = 10ms at f_{Bit}=20 kbit/s

- variable slot length: Slot length is a multiple of a basic time tick where

  $T_{Slot,\,i} > T_{Frame,i}$, e.g. T_{Slot} = 5ms, 7.5ms, 10ms

If not all messages need to be send with the same period, the cycle length is chosen as the least common multiple of all message periods. Some of the messages then are sent more than once during one cycle.

The sender must know the complete schedule. Each slave must only know the PIDs, which it has to receive or respond to. If required, the Master can switch between different schedule tables.

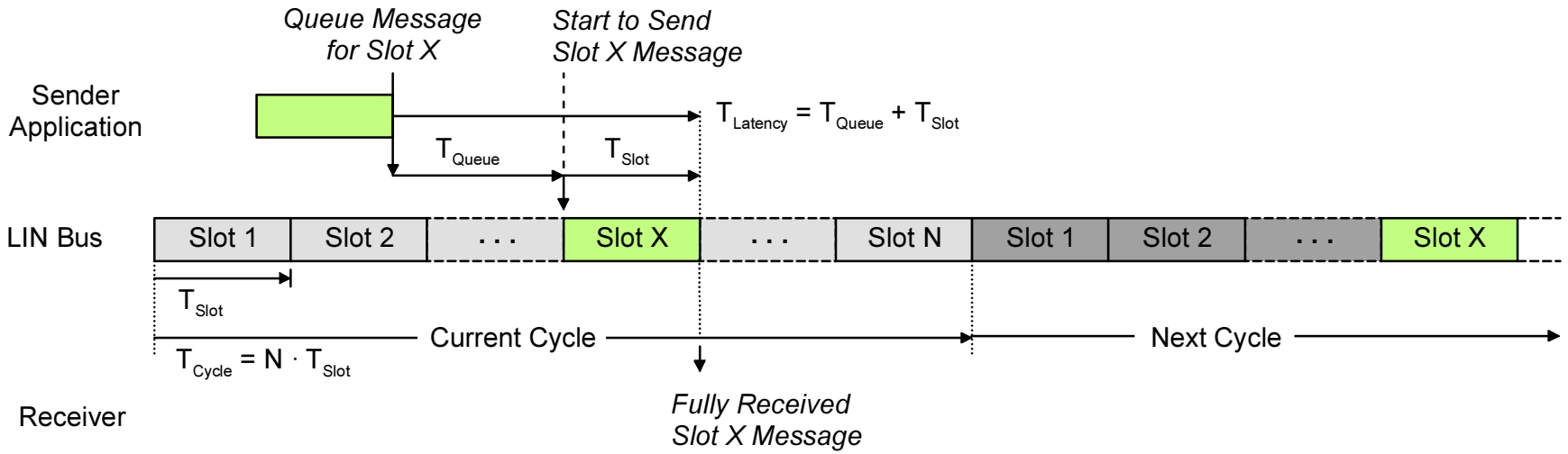
LIN allows for special cases, where a slot can be left free, if no communication is required (*Sporadic Frame*), or where more than one ECU can respond during the same slot (*Event-triggered Frame*), possibly causing a collision. Both are rarely used, for details see the LIN specification.

Net Data Rate
(for constant slot length schedules)

$$f_{Data} = \frac{N_{Data}}{T_{Slot}} = \frac{8\ byte}{10\ ms} = 0.8\ KB/s = 6.4\ kbit/s$$

Protocol Efficiency $\frac{6.4\ kbit/s}{19.2\ kbit/s} = 33\%$

Message Latency



- Best case: Application queues message immediately before begin of its slot

  Message sent immediately: $T_{Queue} = 0$ → $T_{Latency} = T_{Frame} \approx T_{Slot}$

- Worst case: Application queues message immediately after begin of its slot

  Message must wait one full cycle: $T_{Queue} = T_{Cycle}$ → $T_{Latency} = T_{Cycle} + T_{Frame}$

  for a constant slot length system → $\approx (N+1) \cdot T_{Slot}$

  N … number of slots in one cycle

→ Time-critical software applications should be synchronized with LIN cycles

LIN Sleep and Wake-up

- Some ECUs need to remain partly operational, when the car is parked and ignition (terminal 15) switched off.
- E.g. the central door locking remote control receiver needs to still work. To save battery power, the local door ECUs and the bus can go to **sleep mode** (stand-by mode), where all bus communication is stopped. To switch ECUs to sleep mode, the Master ECU sends a **Sleep Message** with LIN PID=$3C_H$ with first data byte = 00_H.
- If the doors shall be opened, the central door locking ECU wakes up the local door ECUs on the bus via a special **wake-up sequence**: Bus low for 0.25 … 5 ms (LIN V2.0). This sequence is detected by the LIN bus transceiver and UART and thus the ECU does switch to normal operation. Any ECU may send the wake-up sequence. The Master-ECU and all Slave-ECUs have to resume the normal message schedule within 100 ms.
- If the Master does not wake-up immediately, the sequence can be repeated every 150 … 250 ms up to 3 times, then a pause of min. 1.5 s is required.

CAN Sleep and Wake-up

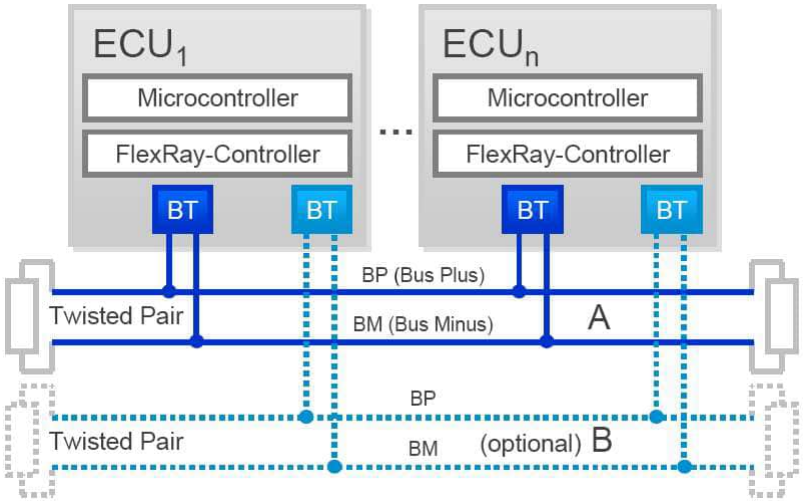
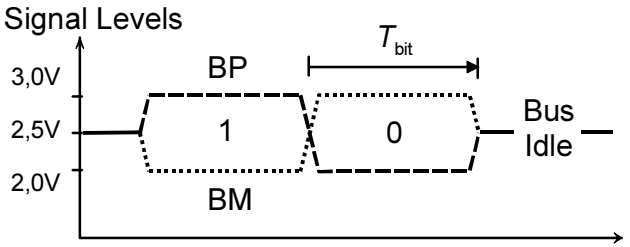
- CAN does not define special sleep and wake-up messages.
- Sleep mode is commanded via a normal application message.
- Most CAN controllers do wake-up from sleep mode, when receiving any message.
- How fast ECUs resume normal operation and sending/receiving messages, must be defined application-specific by the OEM.

Remarks

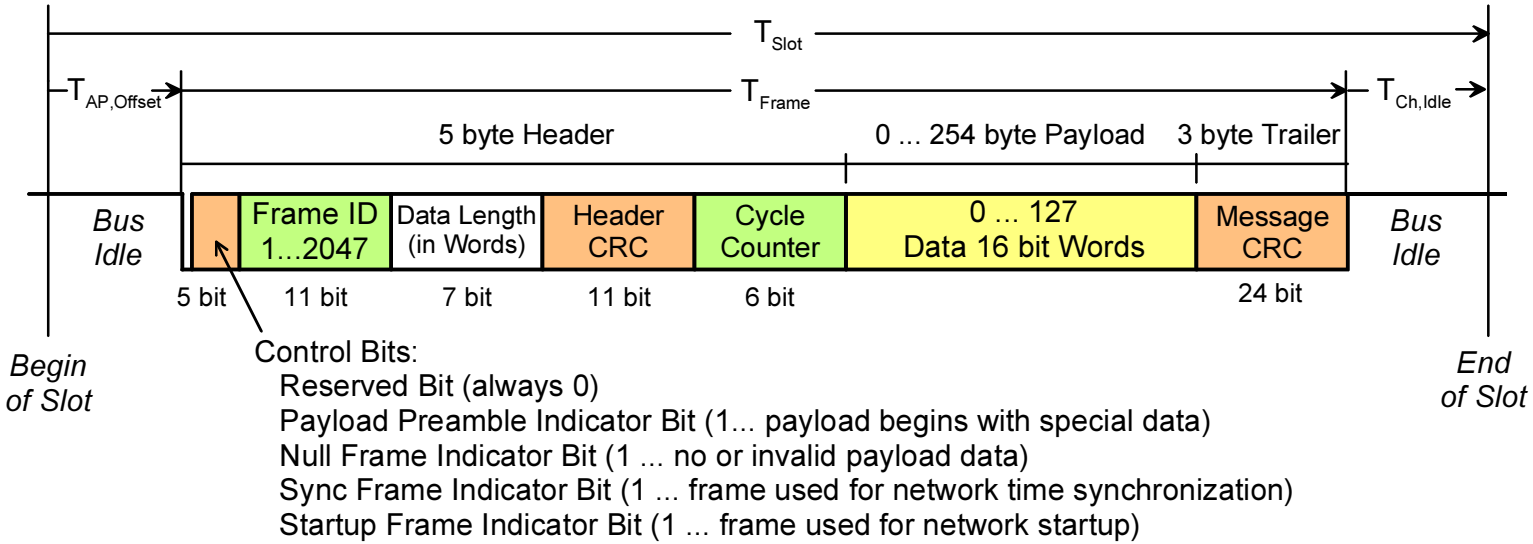
- The LIN Transport Protocol, used to transmit data blocks of > 8 bytes, is similar to the CAN Transport Protocol (ISO TP) defined in ISO 15765-2 (see chapter 3.9).
- Development tools may use a formal textual description language (LIN Configuration Language for Node Capability NCF and LIN Description LDF files), to describe LIN messages and schedules.
- For in-vehicle configuration of out-of-the-box ECUs special node configuration and identification messages are defined. Support for these messages is optional.

LIN protocol version 2.x is not completely upward compatible to version 1.3 with respect to checksum calculation, wake-up sequence and LIN configuration.

3.5 FlexRay

Bit rate	f_{Bit} = 10 Mbit/s (per channel)
Topology and bus length	Double-wire bus (per channel), Single or Dual Channel operation (each ECU may be connect to channel A or channel B or both; coldstart nodes connected to both) Linear bus or star with passive star max. 24 m end-to-end Active star with transceivers at the star point max. 24 m per branch. Optional hybrid topology with max. 2 stars cascaded or one branch as linear bus.
Physical Layer	 Channels … A, B BT … Bus Transceiver
	
History and Usage	Class D automotive network. Developed by an industry consortium as a high-speed successor for CAN, first series application in BMW's X5 2006.
Public standard	Industry standard by FlexRay Consortium (www.flexray.com), now standardized as **ISO 10681** [S4].

Frame Format



0, 2, 4, 6, … , 254 data bytes per frame. Frame ID and Cycle Counter are used for the TDMA/FTDMA network access (see below)

Frame Length

On the physical layer to each frame byte 2 additional bits (BSS Byte Start Sequence) and at the beginning and end of each frame 6 … 18 bits (TSS + FSS Transmission + Frame Start Sequence, FES Frame End Sequence) are added.

$$T_{Frame} = \frac{10}{8} \cdot \frac{n_{Data} + n_{Header} + n_{Trailer}}{f_{Bit}}$$

with n_{Data} data length (in bits)

$n_{Header} + n_{Trailer} = 64$ bit

Note: TSS, FSS and FES neglected.

Slot Length

FlexRay frames must fit into TDMA slots (similar to LIN) with some safety margins $T_{AP,offset}$ (Action Point Offset) and $T_{Ch,idle}$ (Channel Idle) at both sides of the slot.

The required safety margin depends on several parameters of the physical layer (transceiver and bus wire propagation delays, sender and receiver clock tolerances, …). A detailed calculation of the required values is given in the appendix of the FlexRay protocol specification. As a rule of thumb, we will use $T_{AP,offset} = T_{ch,idle} = 5\ \mu s$ in this lecture.

$$T_{Slot} = T_{Frame} + T_{AP,offset} + T_{ch,idle}$$

Net data rate

$$f_{Data} < \frac{n_{Data}}{T_{Slot}}$$

Example:

- n_{Data}= 8 byte data → $T_{Frame} = \frac{10}{8} \cdot \frac{(64+64)\ bit}{10\ Mbit/s} = 16\ \mu s$

  → $T_{Slot} = 16\ \mu s + 10\ \mu s = 26\ \mu s$ → $f_{Data} = \frac{8\ byte}{26\ \mu s} = 300\ KB/s$

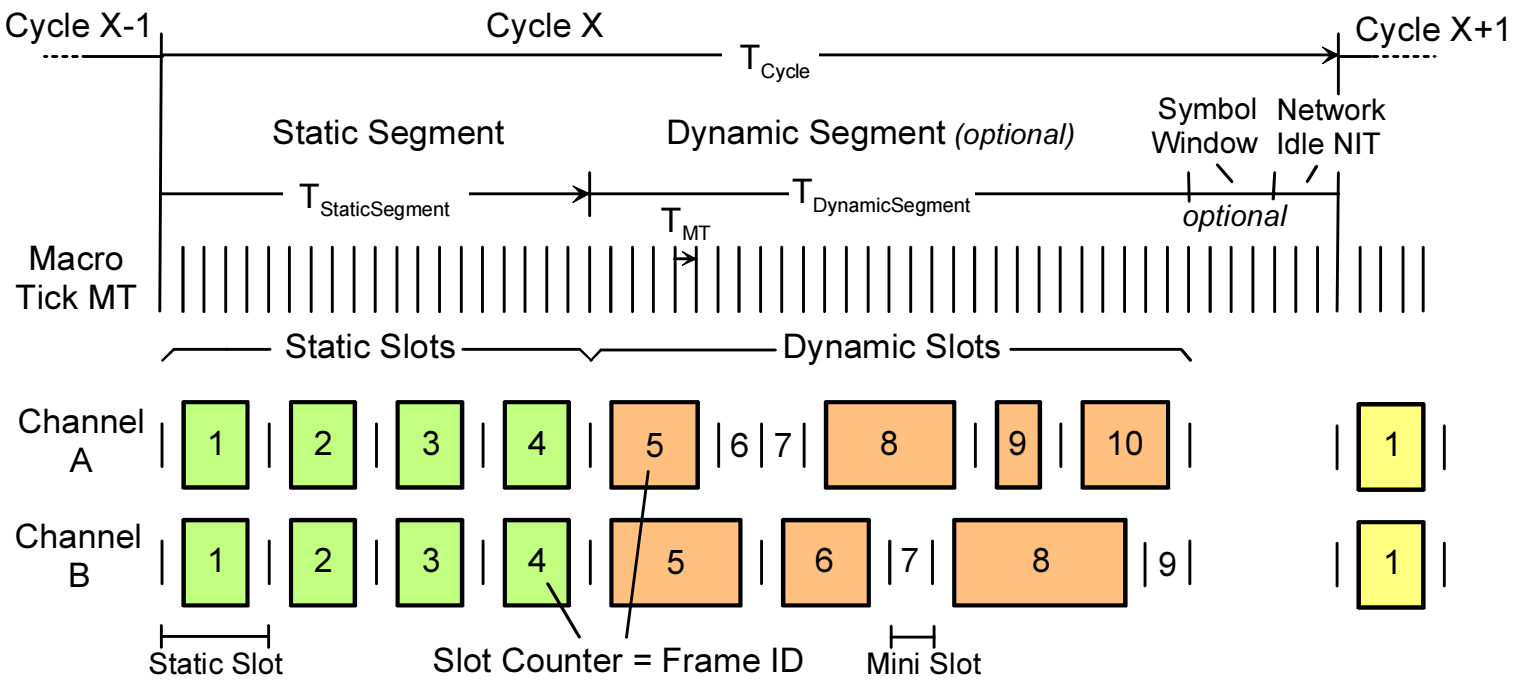
- n_{Data}= 16 byte data → $T_{Frame} = \frac{10}{8} \cdot \frac{(128+64)\ bit}{10\ Mbit/s} = 24\ \mu s$

  → $T_{Slot} = 24\ \mu s + 10\ \mu s = 34\ \mu s$ → $f_{Data} = \frac{16\ byte}{34\ \mu s} = 460\ KB/s$

- n_{Data}=254 byte data → $T_{Frame} = \frac{10}{8} \cdot \frac{(2032+64)\ bit}{10\ Mbit/s} = 262\ \mu s$

  → $T_{Slot} = 262\ \mu s + 10\ \mu s = 272\ \mu s$ → $f_{Data} = \frac{254\ byte}{272\ \mu s} = 910\ KB/s$

Network Access Schedule



- Schedule repeated periodically, one period is called a **Cycle** (e.g. 5 ms).
- Cycle divided into a **Static Segment** (e.g. 3 ms), an optional **Dynamic Segment** (e.g. < 2 ms), an optional (short) **Symbol Window** and the (short) **Network Idle Time NIT**. All have a predefined fixed length.
- Both Segments are divided into Static or Dynamic **Slots** (max. 2047), which are counted by a **Slot Counter** (one per channel) in the communication controller, starting from 1 at the begin of the cycle.
  - A message is sent, when its **Frame ID** matches the current value of the Slot Counter (of the associated channel and cycle, see below).

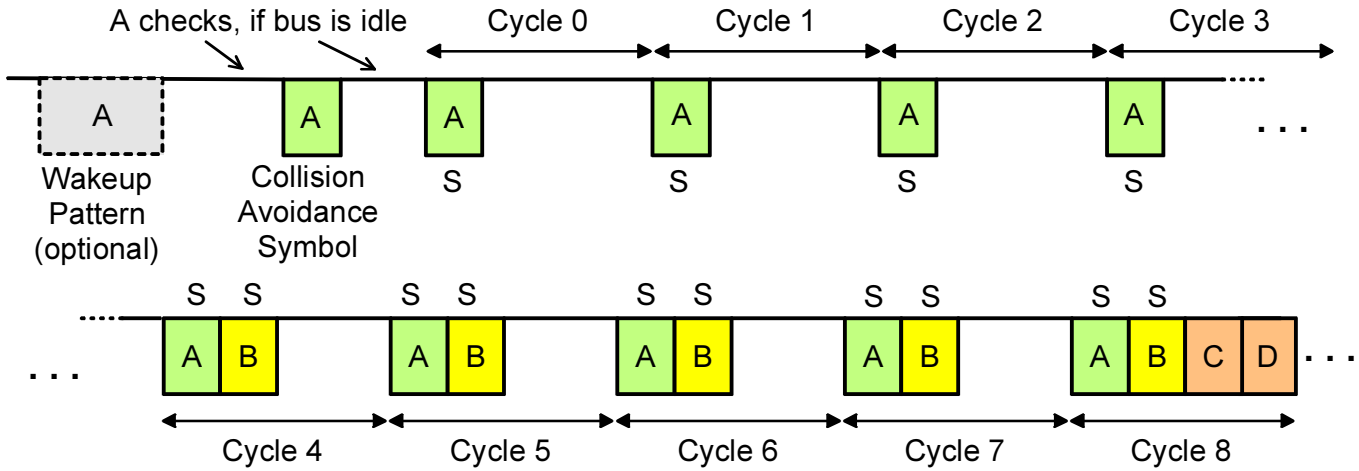
- In each Slot exactly one predefined ECU per channel may start to send a frame. If an ECU needs to send more than one frame per cycle, several Slots may be assigned to the same ECU and/or the ECU may send on both channels in the same or in different slots.
- Only ECUs connected to the same channel can communicate.

- A **Static Segment** contains (at least 2, max. 1023) **Static Slots**:
  - All frames in Static Slots must have the same length T_{Slot} (**TDMA**), which must be long enough to transmit a complete frame (plus safety margin, see Frame Format).
  - Slot Counters of both channels are incremented synchronously in the Static Segment.
- A **Dynamic Segment** contains fixed length **Mini Slots** $<< T_{Frame}$, numbered in the same way by the Slot Counters as in the Static Segment.
  - If no frame needs to be sent, the Mini Slot remains unused and the Slot Counter is incremented immediately at the next Mini Slot.
  - If a frame is send, it typically requires several Mini Slots. The Slot Counter is incremented at the next Mini Slot after the frame has been fully transmitted, i.e. the effective Dynamic Slot length defined by the Slot Counter increments varies (**FTDMA**).
  - Frames in the Dynamic Segment may have different lengths.
  - In the Dynamic Segment, the Slot Counter may differ in both channels.
  - A frame must not be sent, if it cannot be sent completely within the Dynamic Segment, and has to wait at least until the next cycle.

- As previous frames in the dynamic segment do delay the transmission of later frames, the slot number (=**Frame ID**) in the dynamic segment effectively **defines** the **priority** of a message (similar to CAN).

- **Symbol Window** reserved for testing future bus monitoring circuits (Bus Guardians), length similar to a Static Slot, i.e. T_{Slot}, currently not used.

Network Startup and Clock Sync



A, B … Cold Start Nodes, C, D … normal ECUs. S … Frame with Startup and Sync Bit set

While CAN nodes may start sending messages any time after booting up and LIN or MOST networks start sending, when the Master-ECU finished booting and initiates communication. FlexRay networks require a net wide startup sequence, until the Macro Ticks, Slots and Cycles are synchronized. If the network is down, only predefined **Coldstart Nodes** may initiate communication, by sending a special bit pattern (**Collision Avoidance**

Symbol CAS) once and then beginning to send normal frames in Static Slots with the **Startup Frame** (and Sync Frame) Indicator Bits in the header set. Network startup takes at least 8 full communication cycles. Other ECUs can start sending, after having received at least 4 Startup/Sync Frames from two Coldstart Nodes.

Startup Frames (and selected other frames from 2 … 12 different ECUs, so called Sync Nodes, have the Sync Frame Indicator Bit set) are also used for clock synchronization between ECUs. All nodes synchronize their local clock (**Micro Ticks µT**) to the virtual network wide clock (**Macro Ticks MT**, typ. 1 µs). This synchronization is done during the **Network Idle Time NIT** by varying its length in the local nodes (e.g. 100 µs) to compensate for clock tolerances.

Error Detection and Handling

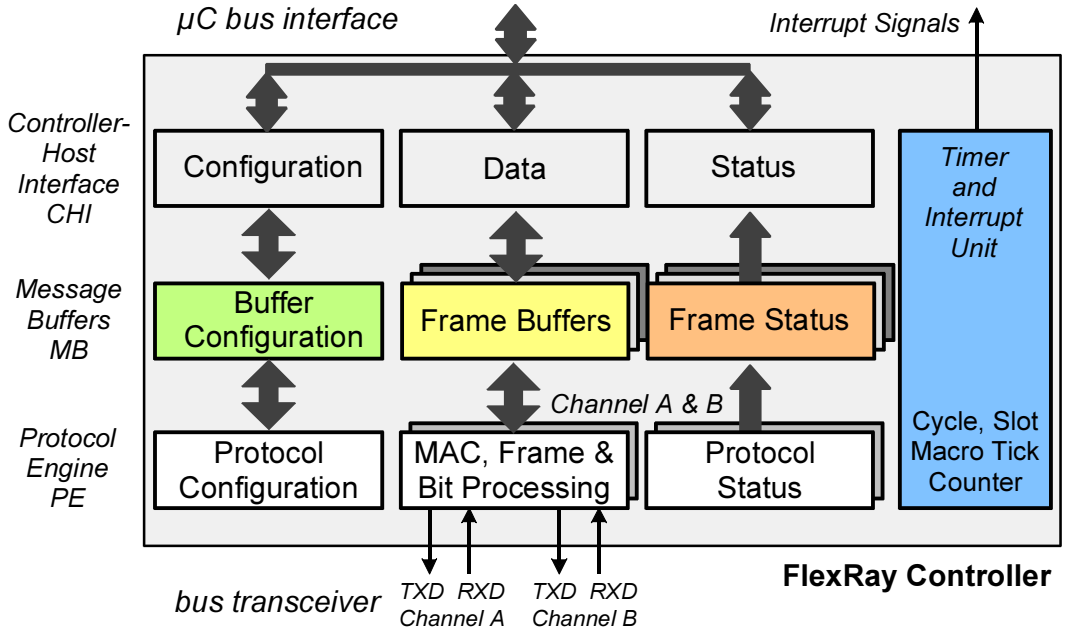
Receivers (and senders) check frame format, frame ID – slot counter match and CRC checksums and indicate errors to the local application via communication controller status bits or interrupts. However, there is no ACK/NACK signaling to the sender or to other nodes and no automatic retransmission.

Depending on the kind and number of errors, a communication controller may stop sending and/or receiving and try to resync to the schedule.

For safety critical applications, the same message can be send over both channels. However, neither sending nor comparison of redundant messages is done automatically by existing communication controllers, but must be handled by application software.

TDMA systems can be safeguarded by only enabling bus transmitters in those slots, in which a node is allowed to send (so called Bus Guardians). Details for this concept are specified by FlexRay, but existing silicon does not yet support this. Currently, FlexRay is only used in applications with the same security requirements, which are already fulfilled by CAN.

Typical FlexRay-Controller



- Limited selection of comm controllers: Freescale Core and Bosch E-Ray Core (licensed to several IC suppliers)
- Complex configuration of protocol parameters with many dependencies
- Acceptance filtering by channel, cycle and slot counter/frame ID
- Complex frame buffer handling
  - Read/write of buffers must be synchronized between µC (Host) and bus
  - Large buffer space required, because of long and/or many messages

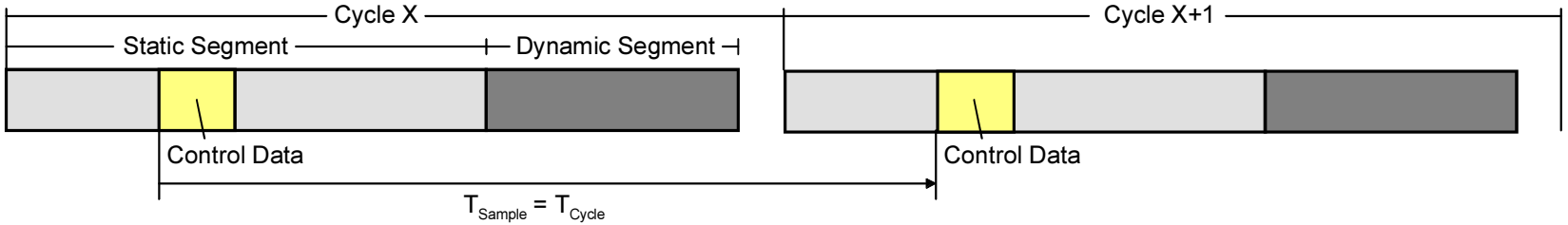
Defining a FlexRay Schedule

- Net wide Parameters

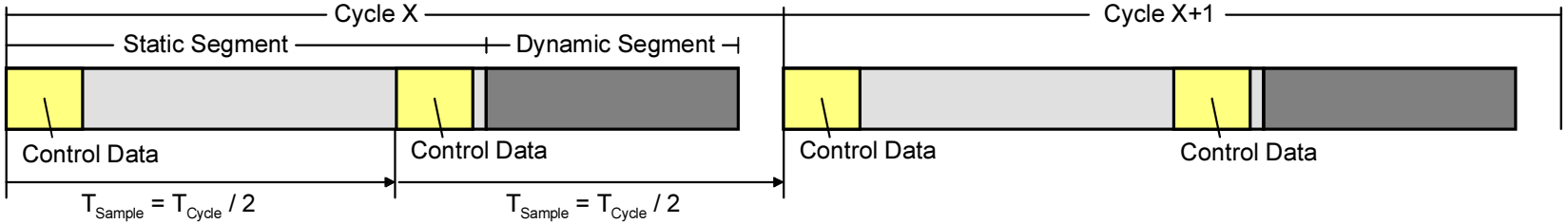
Note: This is a simplified description, values used are sample values, the FlexRay specification has an exhaustive appendix about how these values should be selected or calculated.

Macro Tick Length	T_{MT} = 1 µs	Base for all other timings
Cycle Length	T_{Cycle} = 5 ms = 5000 T_{MT}	
Static Segment Frame Data Size	$N_{Data,StaticSlot}$ = 16 byte If too small, bad net data rate. If too big, reduce number of slots and – if not all ECUs need long messages – waste bandwidth.	
→ Length of Static Slots	$T_{StaticSlot}$ = 34 T_{MT} 24 µs frame length plus 10 µs safety margin, see frame and slot length calculation.	
Number of Static Slots	$N_{StaticSlots}$ = 90	min. 2, defines length of static segment
→ Length of Static Segment	$T_{StaticSegment} = N_{StaticSlots} \cdot T_{StaticSlots} \approx 3$ ms	must be < T_{Cycle}
Length of Symbol Window & NIT	$T_{SW} + T_{NIT}$ = 100 T_{MT}	
→ Length of Dynamic Seg.	$T_{DynamicSegment} = T_{Cycle} - T_{StaticSegment} - T_{SW} - T_{NIT} \approx 1.9$ ms	
Length of Mini Slots	$T_{MiniSlot}$ = 5 T_{MT} Because in a Mini Slot a frame does either start or end, but not both, a length of about half the safety margin of a static slot is sufficient.	
→ Number of Mini Slots	$N_{MiniSlots} = T_{DynamicSegment} / T_{MiniSlot} \approx 368$	

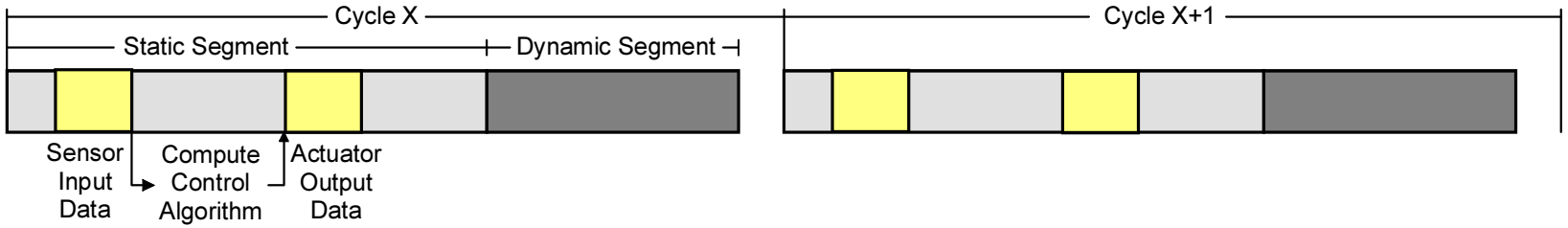
- Static Segment frames often used for periodical control algorithms with $T_{Sample} = T_{Cycle}$



- Increasing the sampling rate without reducing the cycle time: 2 samples per cycle (requires asymmetric segments with $T_{StaticSegment} > T_{Cycle} / 2$)



- In a distributed control system, where sensor signal sampling, control algorithm and actuator output is distributed over 3 different ECUs, latency can be reduced, if the sensor input data is transmitted early in a cycle and the actuator output data later in the same cycle:
  In Cycle Response



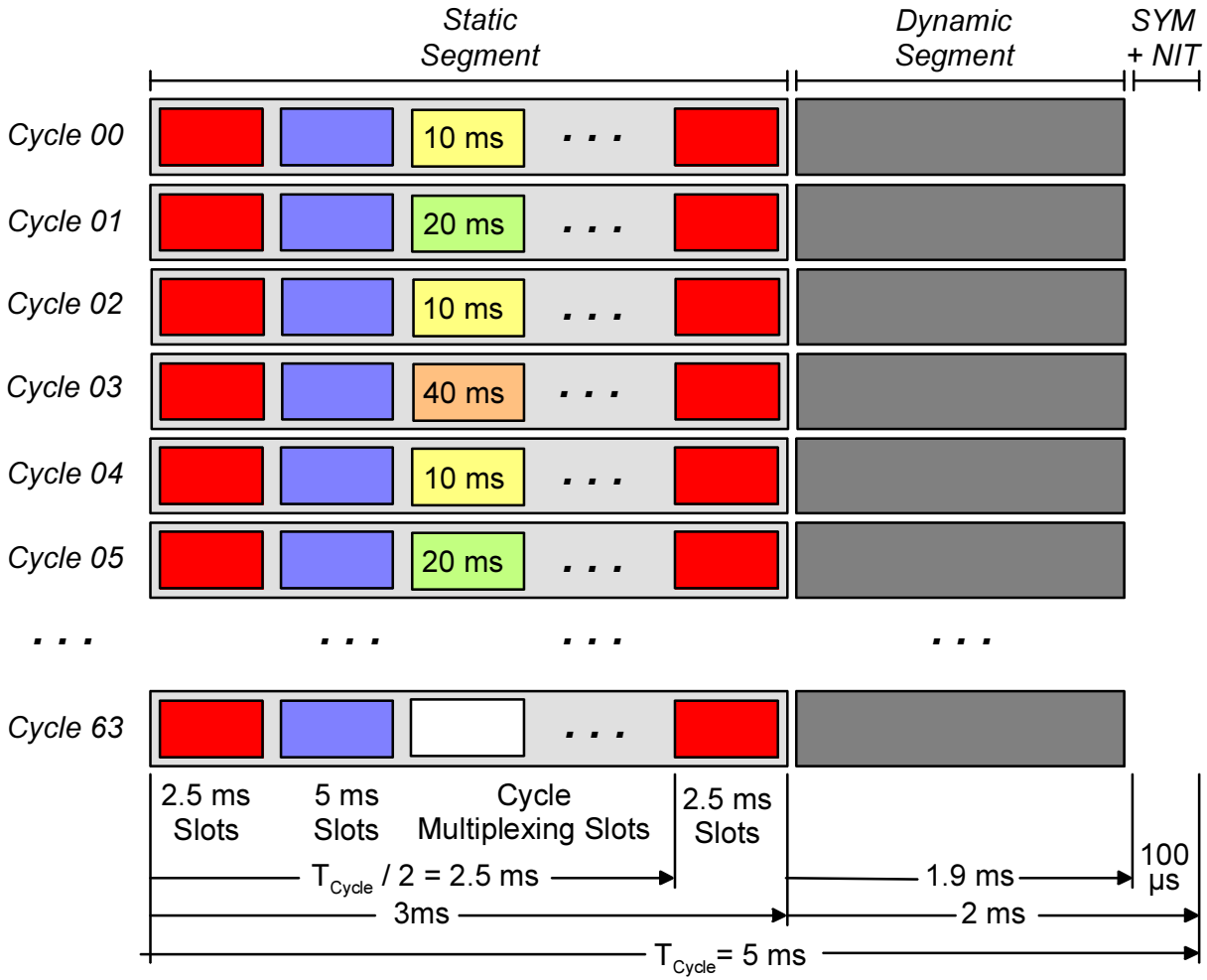
- **Cycle Multiplexing**
  Sampling rates $T_{Sample} = k \cdot T_{Cycle}$ with $k = 2, 4, 8, \ldots$ can easily be implemented by sending frames in the same slot, but with different content every 2^{nd}, 4^{th}, 8^{th}, … cycle.

Cycle Multiplexing is supported by a hardware cycle counter in the communication controller and by sending the current cycle number in the Cycle Counter field of each frame.

Message buffers can be configured in such a way, that messages are sent or received in certain slots and cycles only.

Cycle counters are initialized with 0 and increment up to 63 automatically at the end of each cycle, i.e. max. 64 cycles.

Example:



Message Latency

- Time-triggered Frames (Static Segment) $T_{Frame} < T_{Latency} < T_{Cycle} + T_{Frame}$

  To minimize latency, application software sending and receiving FlexRay messages, should run synchronized with the FlexRay schedule, see LIN Message Latency discussion.

- Event-triggered Frames (Dynamic Segment) $T_{Frame} < T_{Latency} < T_{Queue} + T_{Frame}$

  Messages in the dynamic segment have to wait till the end of the static segment and may be further delayed by messages with lower Frame ID (=higher priority), similar to CAN. If there is not enough time in the dynamic segment, the total delay T_{Queue} may increase to several cycles. Calculation of the worst-case queuing time T_{Queue} is difficult. A similar approach as with CAN could be used, considering the Static Segment as one large, high priority message, for details see [3.3].

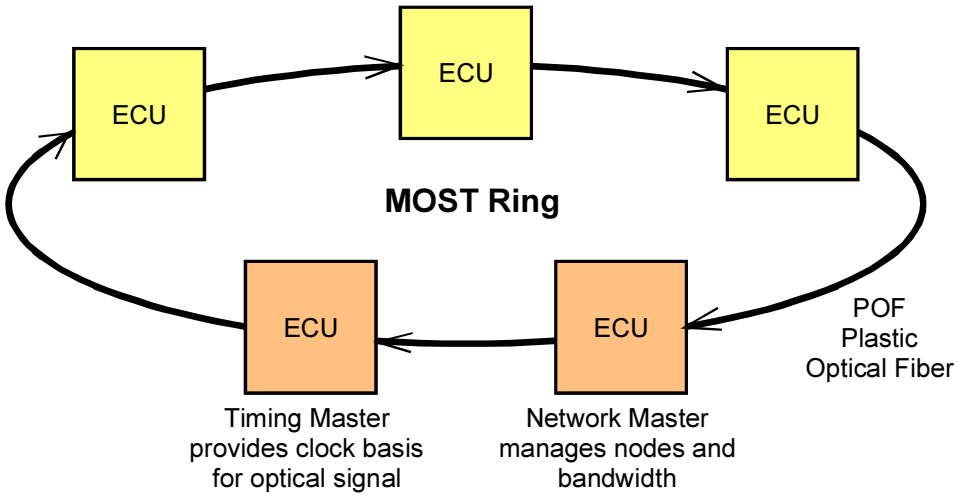
Composability and Maintainability

- In an automotive system during the development cycle and in model year updates new functionality and thus new bus messages will be added.
- In event-triggered communication, e.g. CAN or FlexRay Dynamic Segment, new messages may change worst-case latencies of existing messages. However, as long as there is enough bus bandwidth available, no other system parameters are affected, if new messages are added, and thus existing ECUs need not be changed.
- In time-triggered communication, e.g. LIN or FlexRay Static Segment, new messages change the communication schedule for all ECUs, if no free slots have been reserved. To avoid later changes in all ECUs, schedules must leave enough free slots for future updates.

Info only

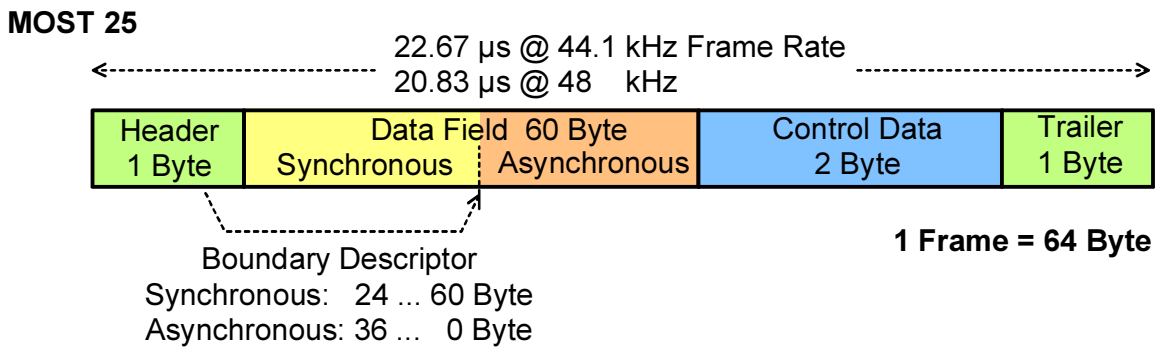
3.6 Media Oriented Systems Transport MOST

Bit rate	$f_{Bit} \leq 25$ Mbit/s, variant with 50 Mbit/s, newest version with 150 Mbit/s
Topology Physical Layer	Plastic-optical-fiber (POF) unidirectional ring, electro-optical transceivers in all ECUs (MOST50 with electrical physical layer rarely used)



History and Usage	Developed by OASIS (then SMSC, now owned by Microchip) and Harman Becker supported by several OEMs. Used in executive and luxury class cars in Europe since 2002 for infotainment applications (audio, video, navigation). At present SMSC still single supplier for MOST communication controllers.
Public standard	Industry standard by MOST Cooperation (www.mostcooperation.com). Only part of the standard freely available, data link layer details not published and covered be company patents [S5]. To avoid being made obsolete by Automotive Ethernet, MOST now is being publicly standardized as **ISO 21806**.

Frame Format



- MOST frames are sent with the standard audio frame rates (44.1 or 48 kHz). Same for MOST25, 50 and 150, the frame rate does not depend on the bit rate!
- MOST25 frames are 64 bytes long, out of which 60+2 bytes can be used for data transfer. Up to 60 bytes can be used for streaming data (synchronous data), the rest of the 60 byte data field is used for packet data (asynchronous data), 2 bytes are used for control messages. The size of the synchronous data field can be changed by restarting the network, but must be the same for all frames in a MOST ring during normal operation
  (MOST50 frame have 128 bytes with 117+4 byte data, MOST150 frames have 384 bytes with 372+4 byte data)

Message Types
Net Data Rate

- MOST provides 3 different message mechanisms:
  - **Synchronous Channels**: Transmitting raw audio or video data at a fixed rate. Synchronous channels are exclusively reserved for one sender by the network management, i.e. TDMA bus access. No addressing or other communication overhead is required. E.g. to stream a stereo audio signal with 16 bit CD quality from the CD player to the

amplifier, a 4 byte wide synchronous channel is required. Data rate for synchronous channels is $f_{Sync} = N \cdot f_{Frame}$ with $24 < N < 60$ byte ...number of bytes in the synchronous channels, i.e. max. $f_{Sync,\,max}$ = 2.9 MB/s (MOST25 at 48 kHz frame rate). Worst case latency for synchronous data is

$$T_{Latency,max} \approx 1/f_{Frame} = 21\ \mu s,$$

if the sending application is not synchronized with the MOST frame rate (see LIN and FlexRay for a discussion).

- **Asynchronous Messages**: The rest of the data field, which is not required for synchronous channels, can be used for asynchronous data, which is transferred via MOST's packet data message format. Such messages may contain up to 48 (optional up to 1014 byte data) plus unique source and target addresses. If an asynchronous message is longer than the asynchronous data field within a single frame, the packet is automatically segmented into multiple frames. Asynchronous channels use the remaining bandwidth $f_{Async} = f_{Sync,max} - f_{Sync}$, which is not used by the synchronous channels. Protocol overhead for asynchronous messages is approx. 20% (@ 48 byte data per message).
- **Control Messages** are used for short data blocks and especially for MOST's network management. Control messages may have up to 17 byte data plus source and target addresses and are automatically segmented into 16 MOST frames. Control channel net data rate is max. 51 KB/s (MOST25 at 48 kHz frame rate). Best case latency is

$$T_{Latency,min} = 16 \cdot 1/f_{Frame} = 333\ \mu s.$$

Despite the high data rates, **Asynchronous and Control Messages** are **not suited for real-time communication**, because neither typical nor maximum message latencies are specified. Both use a proprietary CSMA-like bus access method, which is probably non-deterministic, without revealing further details.

Network Management

MOST specifies, how nodes on the network must be managed and which application layer functionality is required for typical applications, e.g. CD/DVD players or audio amplifiers (so called **MOST Function Blocks**).

Higher-Layer Protocols

On top of MOST's asynchronous messages other protocols like TCP/IP can be implemented.

MOST will be discussed in more detail in the lecture "Automotive Communications".

3.7 Automotive Ethernet

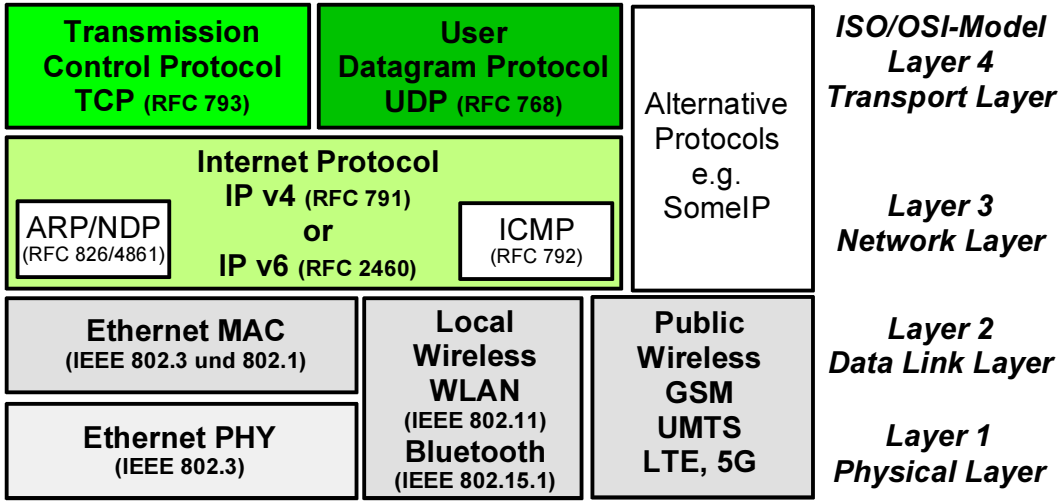
Purpose:

- Provide **Internet connectivity** for the car for navigation, infotainment and diagnosis
- **Integrate consumer devices** (notebook and tablet computers, smartphones)
- **Increase network bandwidth** and **reduce cost** and development time for onboard-communication technology

Solution:

- Use **standard "Internet" protocol stack**
- Use **Ethernet**, WLAN or Bluetooth network
- Avoid special automotive solutions, use low cost **Ethernet** networking technology

Standard "Internet" protocol stack



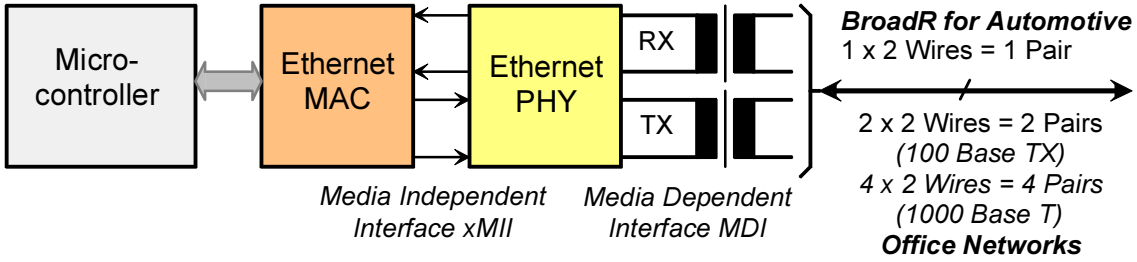
TCP, UDP and IP protocols independent from bus system, but

- Large overhead, especially TCP
- TCP not suited for realtime communication
- TCP, UDP and IP are resource intensive: CPU performance, memory size

→ ok for infotainment ECUs, a challenge for engine, chassis and body ECUs

→ Automotive specific Ethernet/IP modification **IEEE 802.1 Time-Sensitive Networking (TSN)** (evolving standard family [S19])

Ethernet Physical Layer



Ethernet as of IEEE 802.3 fully **separates** Medium Access Control **MAC** (=Data Link Layer) and Physical Layer **PHY**

- **MAC** layer is (nearly) **stable** across generations
- **PHY** layer has **many variants** depending on the required bit rate reaching from 10 Mbit/s … >10 Gbit/s

---

Standard office physical layers (10Base-T/100Base-TX):

- **Full-duplex** communication over 2 pairs of UTP CAT3/5 cable and RJ45-connectors (1000Base-T with 4 pairs UTP CAT6)

→ Not suited for on-board communication due to **cost**, **EMC** and **reliability issues** (vibration, corrosion)

Used

- for **off-board diagnostic communication** and flash programming (software download) → **DoIP** Diagnostic over IP
- long-term substitute for OBD CAN diagnostic interface

Automotive physical layer (BroadR-Reach) **100/1000-Base-T1**

- **Full-duplex** communication over 1 pair of "FlexRay quality" cable and standard automotive connectors, max. 15m
- **100 Mbit/s** in production, **1 Gbit/s** in prototype projects

(**In-vehicle Ethernet ISO 21111** [S18] which refers to the various IEEE standards)

→ **Limited field experience** in automotive environment, another **single-supplier** (Broadcom) situation possible

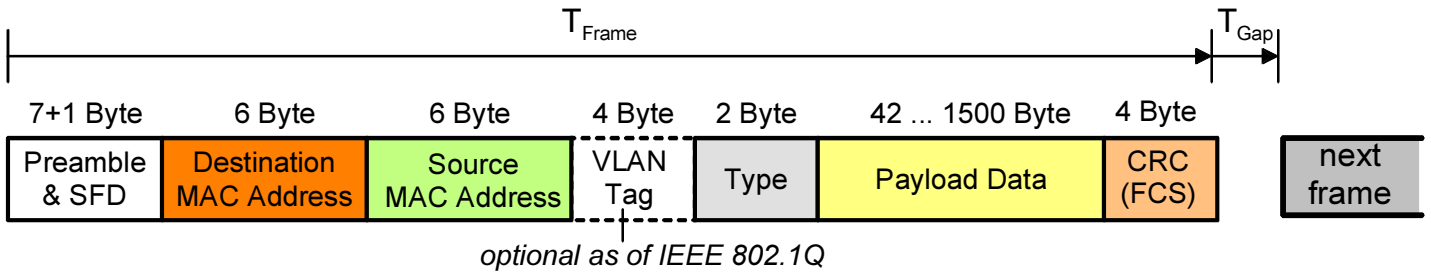
(however several licenses granted, see www.opensig.org for more information)

Used

- as **substitute for MOST** infotainment network and new driver assistant systems with distributed video cameras
- **mid- to long-term substitute for FlexRay** on-board communication, if realtime issues can be solved → TSN

Ethernet Data Link Layer (Ethernet Frames)

- **Device address scheme** with 6 byte source and 6 byte destination addresses
- **Type field** to indicate contents type (higher layer protocol), e.g. IP, TCP, UDP, …
- **VLAN tag** (optional, but will always be used in automotive applications) to signal message priority (and other purposes)
- n_{Data} = **42 … 1500 byte data** per message
- Messages with checksum errors will be discarded, no automatic retries
- Minimum gap of $n_{Gap.min}$ = 12 byte between frames



$$T_{Frame} = \frac{8\text{bit/byte} \cdot (n_{Data} + n_{Header} + n_{Trailer})}{f_{Bit}}$$

with n_{Data} data length (in byte)

$n_{Header} + n_{Trailer}$ 30 byte (including Preamble and VLAN Tag)

$$T_{Gap,min} = \frac{8\text{bit/byte} \cdot n_{Gap,min}}{f_{Bit}} = \frac{96\text{ bit}}{f_{Bit}}$$

with $n_{gap,min}$ 12 byte minimum inter-frame gap

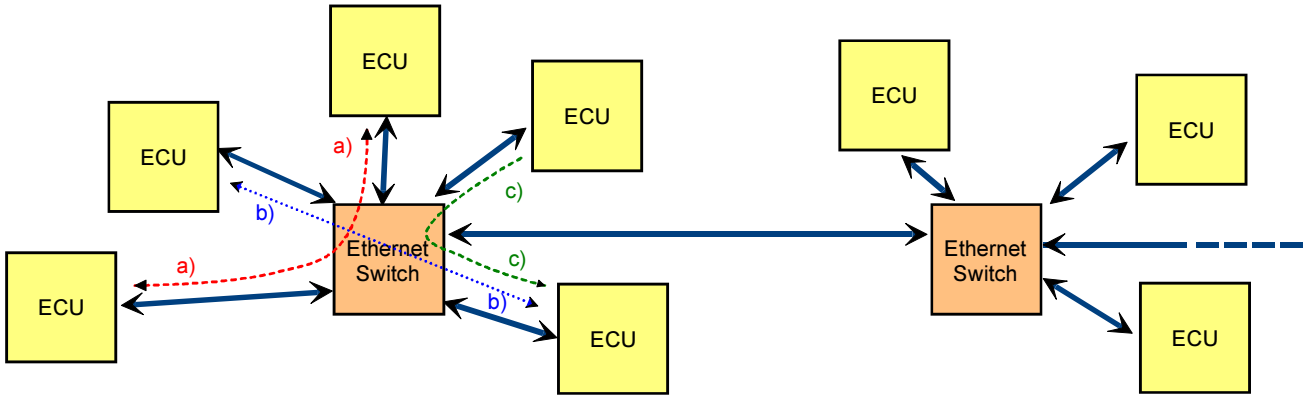
Example: f_{Bit} = 100 Mbit/s → $T_{Gap,min}$ = 0.96 µs

n_{Data} = 42 byte → $T_{Frame,min}$ = 5.8 µs

n_{Data} = 1500 byte → $T_{Frame,max}$ = 122.4 µs

Ethernet Topology and Network latency

Star topology with one (or more) Ethernet **switches** as star points (**Switched Ethernet**)



- A **Switch buffers** each **incoming frame and forwards it to the destination ECU** only (**Store-and-forward** operation), i.e. Ethernet is **not a real bus system** (unlike CAN, LIN or FlexRay)

  $T_{Latency,min} = (N_{switch} + 1) \cdot T_{Frame}$ with N_{Switch} number of switches between ECUs, internal buffer handling and processing time neglected

  Note: A technology named Cut-Through-Switching allows to forward frames after receiving the header already rather than storing the complete frame, thus reducing the latency. In reality, most real-world switches use Store-and-Forward, because Cut-Through does not allow CRC checks in the switch.

- ECUs can send any time, i.e. **event-based communication** like CAN
- **Frames from different source ECUs to different destination** ECUs, e.g. (a) + (b), can be **transmitted simultaneously** (**collision free**), **if they don't share a common connection** to the destination
- **If several frames**, e.g. (b) + (c) are **sent to the same destination** (interference) or over a common link between two switches simultaneously, the **frames** will be **buffered in the switch** and forwarded sequentially (queuing), i.e. **the latency will increase** by at least T_{Frame} for the second frame, 2 T_{Frame} for the third frame etc. (still collision free, but delayed transmission).

- **If** too many messages for the same destination need to be stored simultaneously, **the switch's buffer overflows** and **frames will be dropped**, i.e. frames will get lost without notice. The buffering strategy may be managed via higher TSN layer protocols (Precision Timer Protocol, Time Synchronization and Scheduling, Traffic Shaper, Stream Reservation, … see below)

Ethernet in Real Time and Safety Related Applications [S19, S20]

- Because standard (office) Ethernet does not define minimum buffer sizes, processing order for frames based on priorities and maximum processing time, $T_{Latency,max}$ cannot be specified → **standard Ethernet IEEE 802.3** is **not** suited **for real time applications**
- **Some** of the issues have been resolved by **proprietary solutions**, especially in the automation industry (Ethercat, Profinet, …), which are incompatible to standard Ethernet (and to each other).
- **Some** of the issues **have been resolved by adding higher-layer protocols**, e.g.
  - modifying the VLAN tag to implement a (very limited) priority scheme
  - reserving minimum buffer sizes and sequential sending schemes for groups of frames (Packet Scheduling), i.e. reserving bandwidth with guaranteed maximum latencies
  - implementing a Time-Division-Multiplex scheme (think FlexRay) via software, which requires time synchronization across the network (again via software)

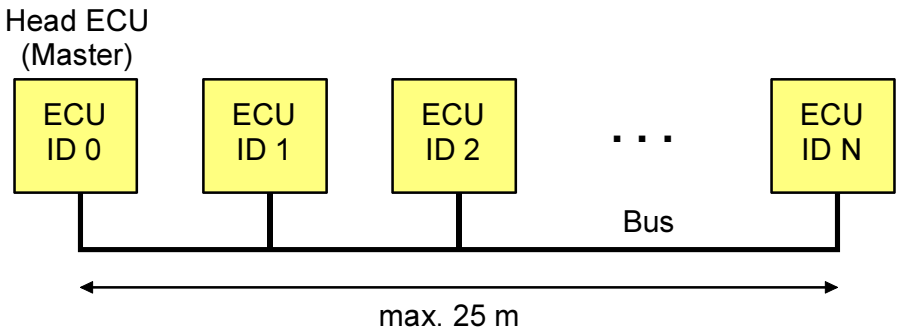
  These concepts are defined in **IEEE 802.1Ax**, **IEEE 802.1Qx, IEEE 1722** etc. → **Time Sensitive Networking TSN** (former name AVB Audio Video Broadcasting [S19]). As there are many interested parties with conflicting goals (computer industry, network providers, automation industry, automotive industry), this is still **work in progress** and some solutions seem to be overly complicated.

  TSN does primarily deal with Ethernet, but not with IP, UDP and TCP and application layer protocols, which have their own real time problems. This is (partly) addressed via the automotive specific **SOME/IP** Scalable Service Oriented Middleware over IP [S20] concept, which has been integrated into AUTOSAR.

Info only

New Low-Cost Ethernet Version 10-Base-T1S

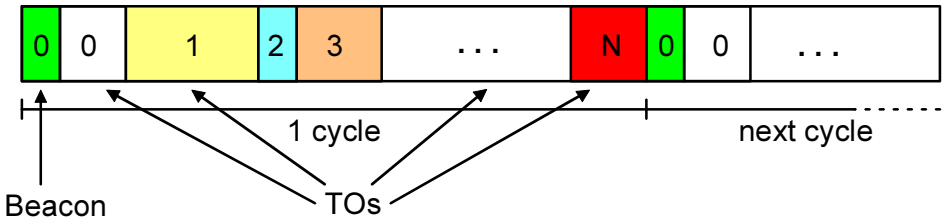
- Fill the bandwidth gap between Ethernet 100-Base-T1 and FlexRay / CAN FD, alternative to CAN XL
- 10 Mbit/sec, Single Pair Unshielded Twisted Pair UTP
- **Bus** topology like CAN/LIN, **no switch (!)**
  (back to the roots: this was Ethernet's topology in the pre 1990 era)



- **Physical Layer Collision Avoidance** (**PLCA**) (to avoid the CSMA/CD related problems of classical Ethernet bus systems) using a schedule like LIN but with variable slots like the dynamic segment of FlexRay:

- ECUs are numbered with IDs 0 … N. ECU ID 0 is the **Head ECU** (Master), which **starts** each communication **cycle with** a unique Ethernet frame (**Beacon**).
- Each ECU / ID has one slot (**Transmit Opportunity TO**) per cycle, where this ECU has the exclusive right to send an Ethernet frame → no collisions.
- The **slot length** is **variable** according to the length of the respective Ethernet frames. If no data is available, the slot may be left empty (idle). Empty slots are very short, ~ 20 T_{bit}

PLCA schedule



Info only

Overhead of Internet Protocols TCP/IP and UDP/IP

IP IP frames are transmitted as payload data of Ethernet frames

IPv4 20+byte header (incl. 4 byte source IP and 4 byte destination IP address) + ~64kB data

IPv6 40 byte header (incl. 8 byte source IP and 8 byte destination IP address) + ~64kB data

If the IP frame is >1500 byte, i.e. longer than the max. payload of an Ethernet frame, the IP layer will split (fragment) the IP frame into several Ethernet frames.

World-wide unique IP addresses are required to route an IP frame over different networks, while Ethernet MAC addresses need only be unique within a single Ethernet network

UDP UDP frames are transmitted as payload data of IP frames

8+byte header (incl. 2 byte source port and 2 byte destination port number) + ~64kB data

Connection-less, i.e. sender can send event-based, no feedback from receiver and no flow control. Port numbers identify a unique application (software module) on the sender and on the receiver side.

TCP TCP frames are transmitted as payload data of IP frames

20+byte header (2 x 2 port number like UDP) + (nearly unlimited) stream of data

Connection must be established (via 3 messages) and cancelled. Dynamic flow control and error signalling via ACK messages from receiver, automatic retries in case of errors → not suited for real-time

Info only

3.8 Outdated Busses: K-Line, SAE J1850 and SAE J1587/1708

K-Line

- Used for off-board diagnostic communication in European passenger cars and trucks; substituted by CAN
- Standardized in ISO 9141 [S7] and ISO 14230 (KWP 2000) [S8]
- Physical layer similar to LIN: Single-wire, bit rates up to 10.4 kbit/s, implemented with standard micro-controller UARTs plus bus transceiver
- Connection-oriented communication between the off-board diagnostic tester and one of the ECUs in the vehicle. Not a real bus system, i.e. no communication between ECUs.

SAE J1850

- Used for on- and off-board communication in US passenger cars, also used by Ford Europe; substituted by CAN
- Two incompatible physical and data link layer defined by SAE in the same J1850 standard [S6]:
  - Ford: Double-wire bus, Pulse-Width-Modulation (PWM) for bit-encoding, 41.6 kbit/s
  - General Motors & Chrysler: Single-wire bus, Variable Pulse-Width-Modulation (VPWM) for bit-encoding, 10.4 kbit/s,

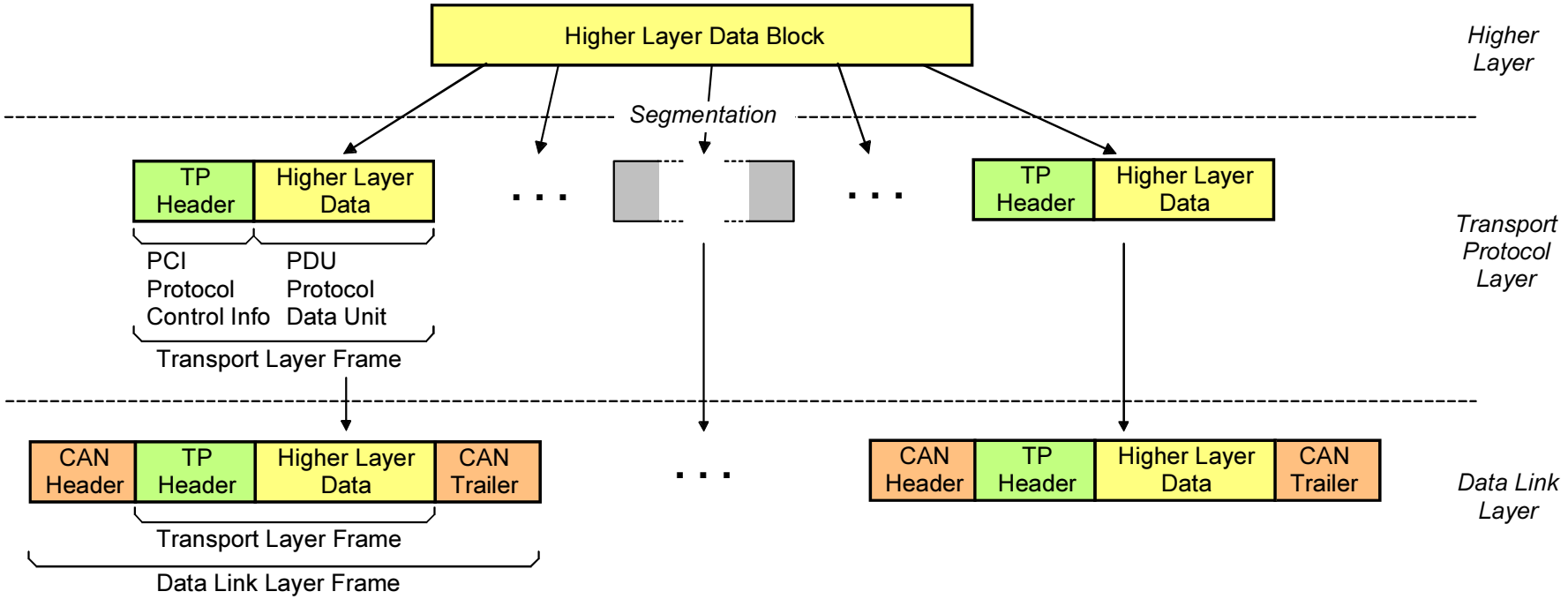
  Data-link-layer implementation with normal microcontroller peripherals difficult

SAE J1587/1708

- Used for on-board and off-board communication in US trucks; substituted by CAN and SAE J1939
- Double-wire bus with 9.6 kbit/s, implemented with standard µC UART, standardized SAE J1708
- Associated SAE J1708 standard defined content of messages for on-board-communication

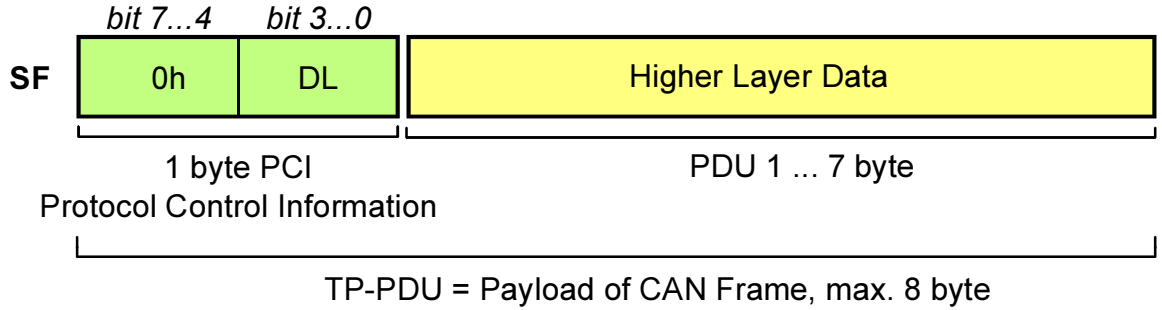
3.9 Transport Protocols

- CAN and LIN may transmit max. 8 data bytes, FlexRay max. 254 data bytes with one bus message (data link layer frame)
- Diagnostic applications may require much larger, variable length data blocks, e.g. reading the vehicle identification number (in Germany 17 ASCII characters) or End-of-Line-Flash-Programming (some KB up to several MB)
- Transport Protocols (TP) define, how to split large, variable length data blocks into multiple bus messages (**segmentation**) on the sender side and reassemble on the receiver side and how to synchronize the sender in order not to overflow the receiver (**flow control**).



CAN-Transport Protocol ISO 15765-2 (ISO TP) uses 4 different message types [S10] :

- If data block size < frame data size → Unsegmented **Single Frame** (SF) message

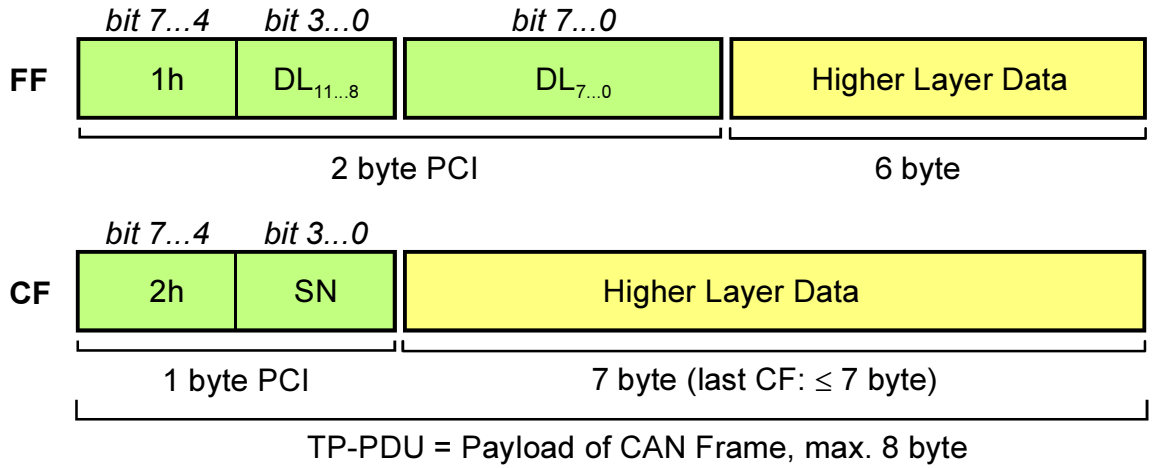


PCI = 1 byte header

Bit 7...4 = 0 marks **SF**

Bit 3..0 = DL Data Length
Number of data bytes

- If data block size > frame data size → **Segmentation** with **First Frame** (FF) and **Consecutive Frames** (CF)



First Frame with 2 byte PCI

1h ... marks **FF**

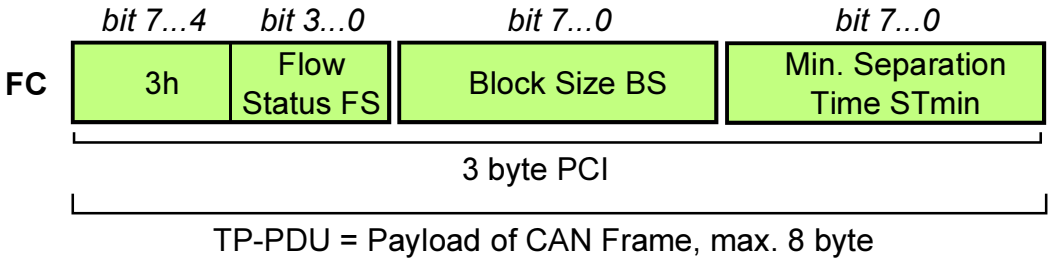
DL ... Number of bytes in complete data block, max. 4095 byte

Consecutive Frame with 1 Byte PCI

2h ... marks **CF**

SN ... Sequence Number: Serial number of CF frames, starts at SN=1 with first CF in block, counts modulo 16

- If segmentation is required, the receiver informs the sender, how many and how fast it can receive frames → **Flow Control** Frame (FC)



3h …	marks **FC**
FS …	Flow Status 0 … Clear to send CTS 1 … Wait, 2 … Overflow
BS …	Block Size: Allowed number of sequential CF frames, max. 255 BS=0 is interpreted as "No limit"
ST …	Min. Separation Time: Minimum interval between CF frames in ms

FF – FC – CF is sent in a well-defined sequence:

- Sender sends *First Frame* FF → Receiver responds with *Flow Control Frame* FC
- Sender sends BS *Consecutive Frames* with minimum time intervals ST_{min}
- After BS *Consecutive Frames*, the sender stops
- When the receiver has processed the first BS messages, it sends a new *Flow Control Frame* FC to request the sender to continue
- . . .
- The sequence repeats until the complete data block was transmitted.

Example: Diagnostic message with $35=23_H$ byte, receiver can buffer 24 byte only

Sender

FF: 10h, 23h + 6 byte data

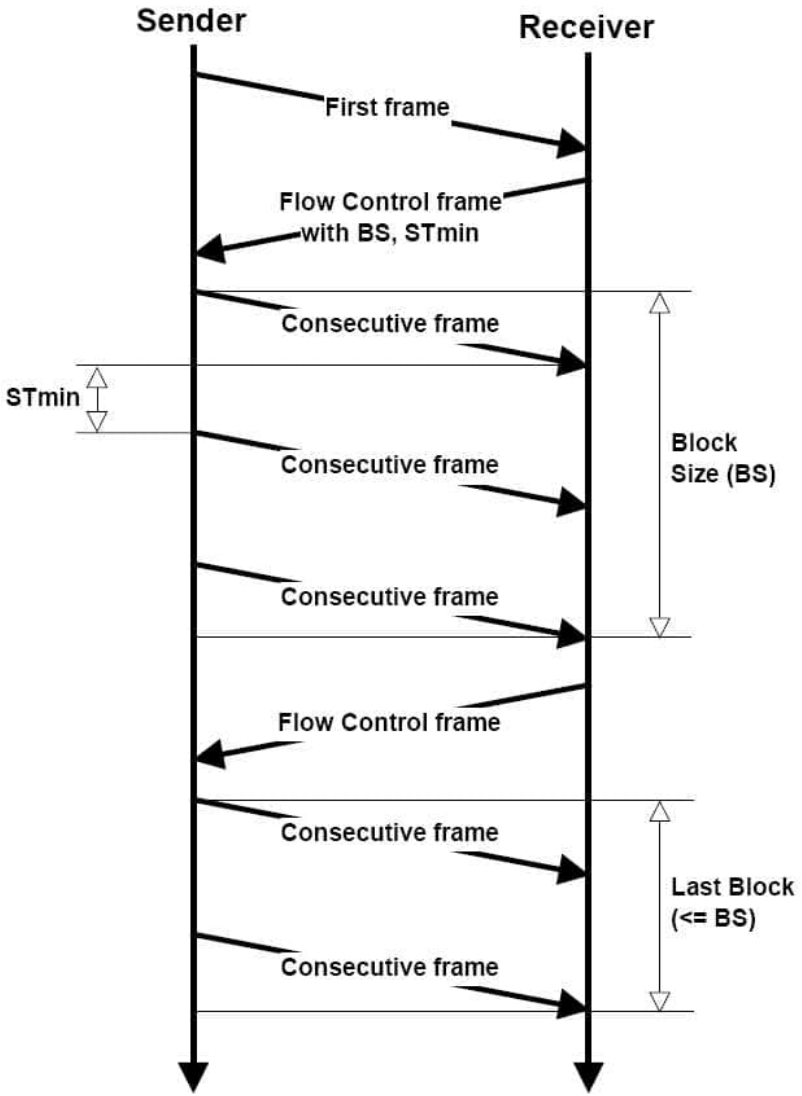
CF: 21h + 7 byte data
(SN=1)

CF: 22h + 7 byte data
(SN=2)

CF: 23h + 7 byte data
(SN=3)

CF: 24h + 7 byte data
(SN=4)

CF: 25h + 1 byte data
(SN=5, CF with < 8 byte data indicates end of transmission)



Receiver

30h, 03h, 0Ah :FC
(CTS, BS=3, STmin=10ms)

30h, 03h, 0Ah :FC

(Source: ISO 15765-2)

Timeouts

- Sender side: Time between sending FF and receiving FC max. 1 s
- Receiver side: Time between CF and next CF max. 1 s

If the receiver needs more time for processing, it may send a *Flow Control Frame* with *Flow Status* FS = Wait to request a pause. After max. 1 s the receiver can request the sender to wait again or to continue to send by another FC frame.

ECUs may use any value for BS and STmin within the specified limits, while diagnostic testers are expected to use BS=0, STmin=0.

Error Detection

- Sender and receiver do check timeouts. Receivers do additionally check sequence numbers.
- If an error is detected, the received message is ignored and the application on the sender or receiver side must handle the problem. The transport layer does not define any error handling like automatic retransmission or error messages between senders and receivers other than FC with FS=Overflow.

Transport Protocol Limits

- If ISO TP were used for on-board communication rather than raw CAN frames, net data rate would be reduced by at least 1/8 = 12.5%, because of the 1 byte PCI overhead in SF.
- On-board communication messages typically are received and processed by more than one ECU. However, ISO TP does only allow segmented transfers to one receiver only, because it does not define how to handle FC frames from multiple receivers, i.e. ISO TP is intended for unicast 1:1 communication.

ISO TP for CAN FD

- As CAN FD can send up to 64 byte in one frame, the transport protocol is modified as follows:
  - In SF frames, the 4 bit DL field in the first byte is set to 0 and an 8 bit DL is sent in the following byte, i.e. the PCI header is 2 byte rather than 1 byte.
  - In FF frames, the 12 bit DL field set to 0 and an 32 bit DL is sent in the following 4 bytes, i.e. the PCI header is 6 bytes rather than 2 bytes.

Other CAN Transport Protocols

- For OBD diagnostics (*legalized communication*) all OEMs must use ISO TP accoring to ISO 15765-4. OEMs typically also use it for UDS repair shop diagnosis. Only VW and Audi historically used a proprietary protocol named **VW TP 1.6** and **VW TP 2.0**.
- In medium and heavy duty trucks the standard family **SAE J1939** [S12] is used for on-board communication, which has its own transport protocol (using similar principles like ISO TP, but differing in all details). For off-board diagnostics, ISO TP may be used additionally.
- Note: On the same CAN network, various protocols can be used in parallel by assigning different CAN IDs.

LIN Transport Protocol

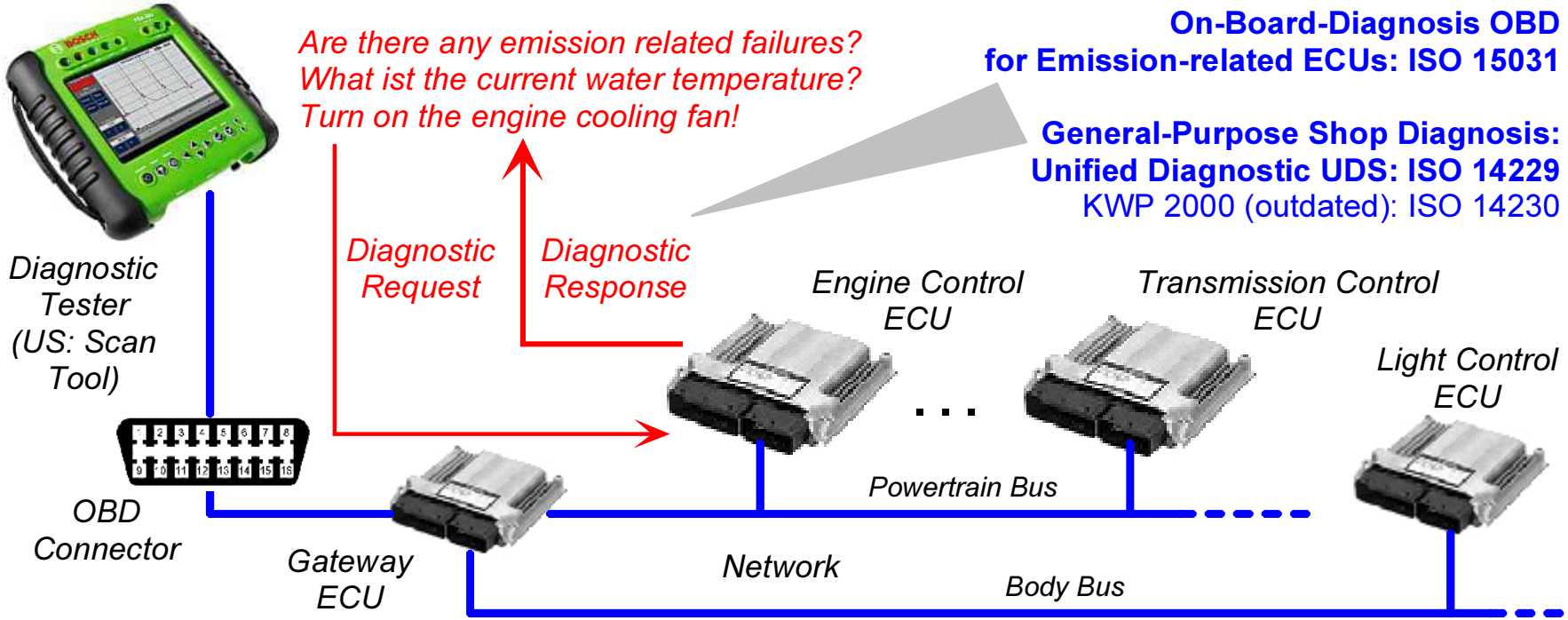
LIN uses a modification of ISO TP:

- All LIN TP frames use LIN PID=3CH for data sent from the Master- to any Slave-ECU and PID=3Dh for data sent from any Slave- to the Master-ECU, i.e. PID cannot be used to identify the slave. Therefore …
- LIN TP frames use the first payload byte for node addressing (NAD) to uniquely identify the sender or receiver node. And thus …
- SF can only contain 6 higher-layer data bytes. FC frames are not required, because the normal LIN slot schedule is used for TP messages too.

FlexRay Transport Protocol

- Extended version of ISO TP to accommodate for larger frame and block sizes, published as ISO 10681-2

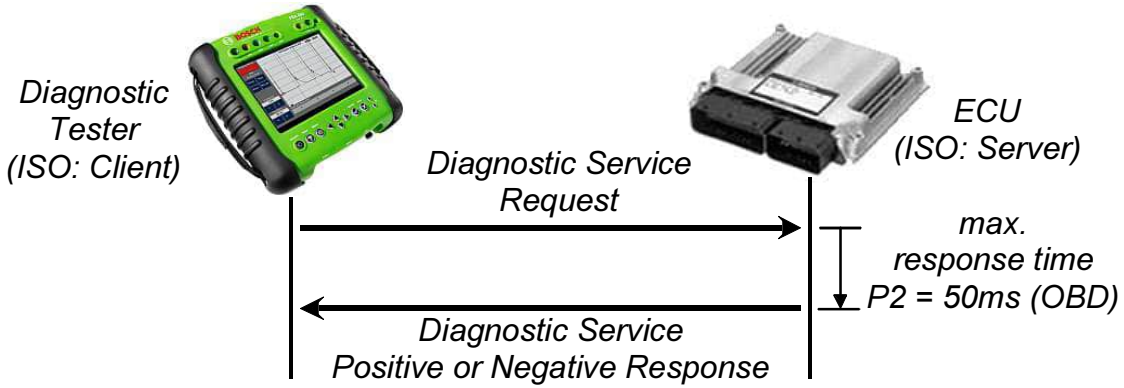
3.10 Diagnostic Communication Protocols OBD and UDS

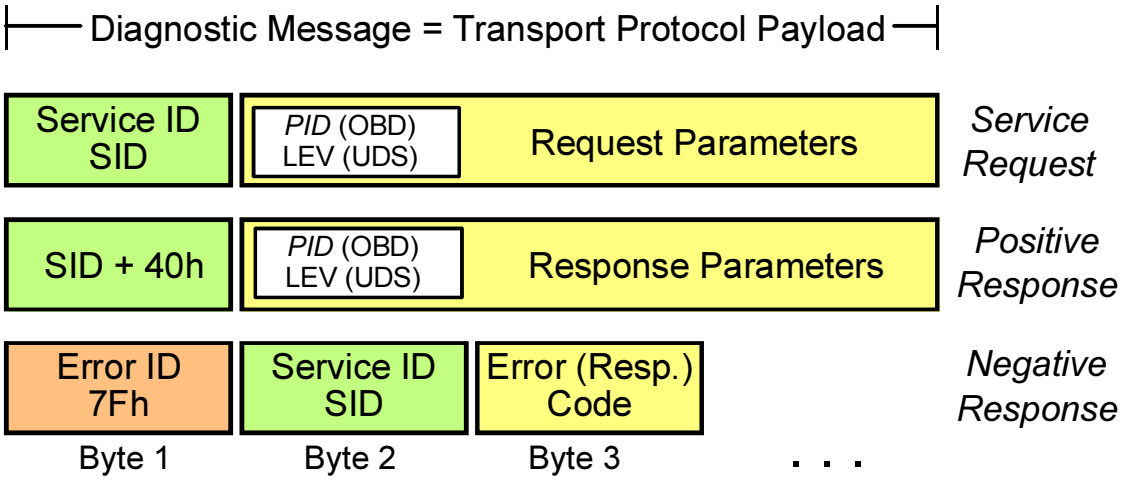


Communication model:

- Off-board communication on-demand only
- **Diagnostic Tester** (***Client***) sends a **Request** – one or more **ECUs** (***Servers***) send a **Response**
- Diagnostic Tester does not know in advance, which ECUs are installed in a car
- Critical ECU data must be protected against unauthorized access and manipulation

Common Principles of OBD and UDS on CAN





Example: Requests uses SID = 01_H ...
Positive Response is 41_H ...
Negative Response is $7F_H$ 01_H ...

- Tester sends Service Request to one (or more) ECUs
- Each ECU sends one positive or negative Response
- Timeout between Request and Response (SF or FF) of all ECUs $P2_{max}$=50ms, minimum time Response and next Request P3=0

- **SID** ... **Service Identifier** marks contents of Request (*Diagnostic **Service***, *Test Mode*), other bytes contain request parameters or data
- **Positive Response** uses **SID+40h** as the first byte, other bytes contain response data
- **Negative Response** uses **Error ID=7Fh**, **SID** and an **Error Code**

Standardized SID-Ranges

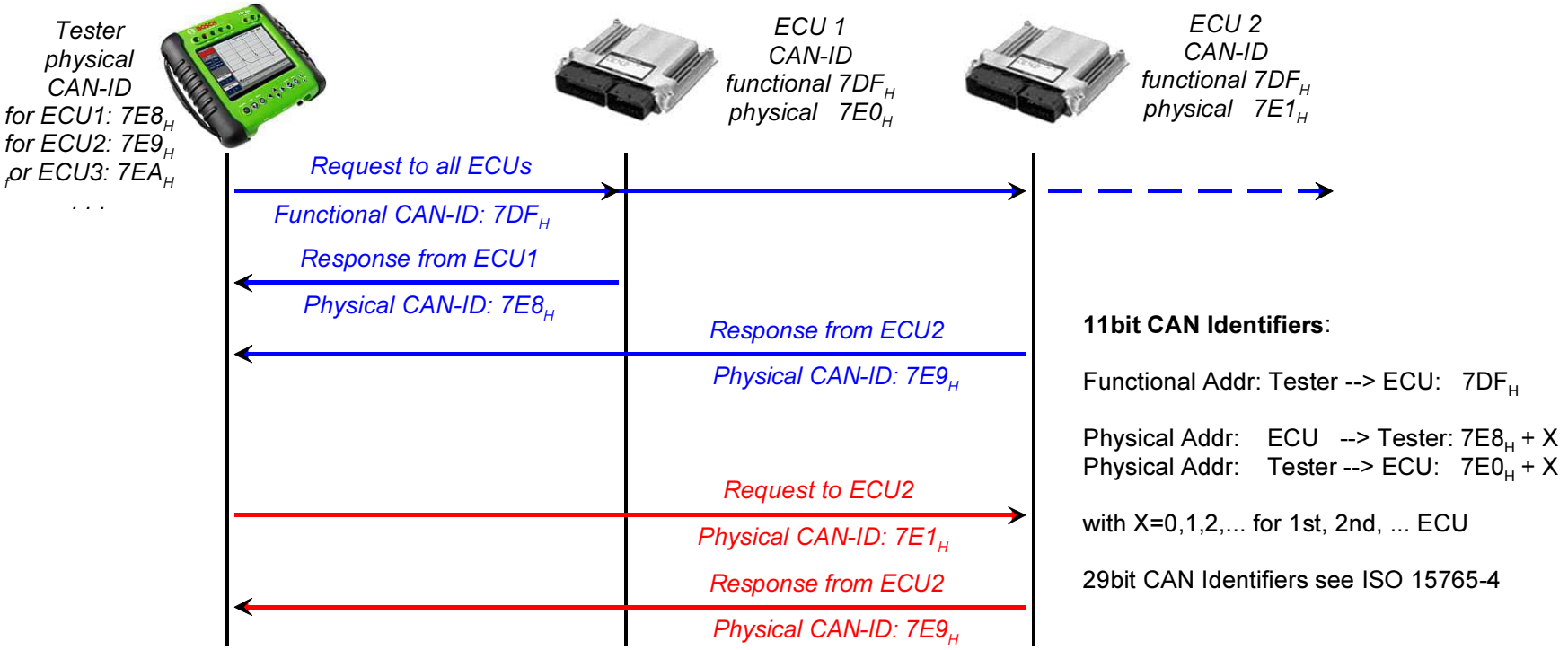
Request	*Positive Response*	
00_H ... $0F_H$	40_H ... $4F_H$	**OBD** as of ISO 15031-5 Diagnosis of emissions-related components
10_H ... $3E_H$ 83_H ... 87_H	50_H ... $7E_H$ $C3_H$ ... $C7_H$	**UDS** as of ISO 14229 or **KWP 2000** as of ISO 14230 General-purpose workshop diagnosis UDS and KWP 2000 use the same SIDs, but with different functionality and/or parameters.
81_H ... 82_H	$C1_H$ ... $C2_H$	KWP 2000 on K-Line as of ISO 14230
$A0_H$ ... $B9_H$	$E0_H$ ... $F9_H$	Reserved for OEM use
BA_H ... BE_H	FA_H ... FE_H	Reserved for ECU manufacturer use
Other values		Reserved
	Negative Response	
	$7F_H$	Error message

Some Error Response Codes (Note: These are not Diagnostic Trouble Codes !)

10_H	General reject
11_H, 12_H, $7E_H$, $7F_H$	Service/Sub-function not supported (in active session)
13_H, 31_H	Message length or format incorrect, Out of range
21_H, 78_H	Busy – repeat request, Busy - response pending
22_H, 24_H	Conditions not correct, Request sequence error
33_H, 35_H, 36_H	Security access denied, Invalid key, Exceed attempts

Functional and Physical Addressing

- **Physical Addressing**: Every communication link between the tester and each ECU uses a unique pair of CAN IDs: one request CAN ID and one response CAN ID per ECU
- **Functional Addressing**: Because the diagnostic tester does not know, which ECUs are built-in, it will send its requests to a broadcast CAN ID, which is valid for all ECUs of a certain group, e.g. emissions-related ECUs (only used in tester requests, not used in ECU responses)
- On a request with functional addressing, several ECUs may send responses. Sender ECUs can be identified by the tester, because their responses use unique physical addresses.



OBD on CAN as of ISO 15031-5 / SAE J1979 (OBD) and ISO 15765-4 (CAN for OBD)

- **Transport Protocol ISO TP and CAN Requirements**
  - Networks may use CAN bit rates 250 kbit/s or 500 kbit/s, 11bit or 29bit CAN IDs, must be detected automatically by diagnostic tester. Tester effectively must not use flow control for reception, i.e. must send Flow Control Frames with BS=0, STmin=0 only.
  - Diagnostic requests must use functional addressing, responses use physical addresses.
  - ECUs, which do not support a certain OBD request, shall not send a response to reduce busload, i.e. tester always has to wait for $P2_{max}$ timeout. Reason: Unknown how many ECUs will respond
  - Requests must use SF messages, responses may use SF or FF/CF

- **ODB Diagnostic Services** (called *Test Modes* in SAE standards)

	SID	
Measurement Data and Vehicle Info	01_H	Read measurement values
	09_H	Read vehicle information
Failure Storage	03_H	Read confirmed DTCs (Diagnostic Trouble Codes)
	02_H	Read freeze frame data
	07_H	Read confirmed and preliminary DTCs
	04_H	Clear DTCs, freeze frame data and MIL
Test of emissions-related components	06_H	Read on-board monitoring test results
	08_H	Trigger a component test, e.g. tank ventilation test

Example: Read measurement value service SID=01_H (1^{st} byte of request)

- Selection of value by Parameter Identifier PID (2^{nd} byte of request):

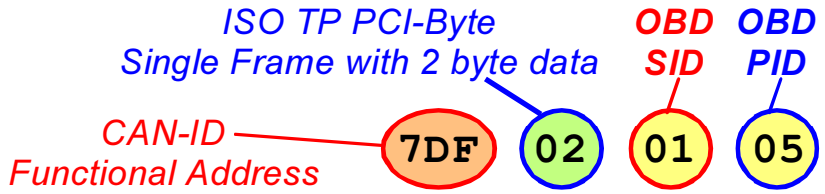
PID	*Value*	*Length*	*Range (min...max)*
01h	Number of stored DTCs	8 bit	
04h	Engine load	8 bit	0 … 100 %
05h	Coolant water temperature	8 bit	-40 … +215 °C
0Bh	Intake manifold pressure	8 bit	0 … 255 kPa
0Ch	Engine speed	16 bit	0 … 16383.75 1/min
0Dh	Vehicle speed	8 bit	0 … 255 km/h
0Fh	Intake manifold temperature	8 bit	-40 … +215 °C
11h	Throttle position (accelerator pedal)	8 bit	0 … 100 %
31h	Driving distance since DTCs cleared	16 bit	0 … 65535 km
21h	Driving distance since MIL turned on	16 bit	0 … 65535 km
. . .	. . .	. . .	. . .

16 bit values are transmitted in Big Endian format, e.g. 8192 km → 20 00_H

Each SID has its own list of PIDs. The above list is for SID=01_H and 02_H. A detailed list for all services can be found in ISO 15031-5 Annex A – G.

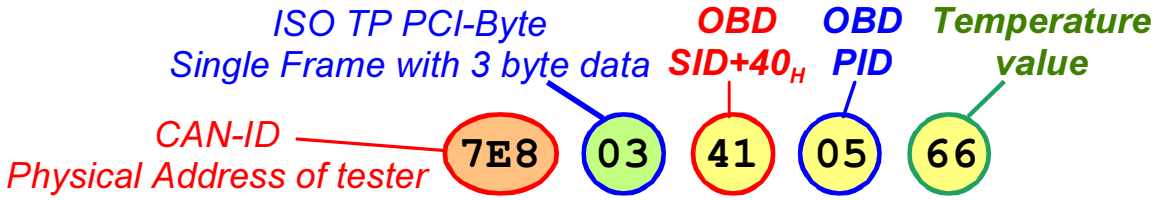
An ECU need not support all PIDs. The tester can send a query for each SID using PID=0, which will return in a bit field, which PIDs the ECU does support (see below).

- Tester request to read the coolant water temperature



Tester can request several values by sending a list of PIDs rather than a single PID

- Engine management ECU response



(see page 3.72 for conversion details)

Extract from ISO 15031-5

Section 6: Diagnostic Service Definition for ISO 15765-4

6.1.2.3 Request current powertrain diagnostic data request message definition (read PID values)

Table 110 — Request current powertrain diagnostic data request message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	Request current powertrain diagnostic data request SID	M	01	SIDRQ
#2	PID#1 (see Annex B)	M/C	xx	PID
#3	PID#2 (see Annex B)	U/C	xx	PID
#4	PID#3 (see Annex B)	U/C	xx	PID
#5	PID#4 (see Annex B)	U/C	xx	PID
#6	PID#5 (see Annex B)	U/C	xx	PID
#7	PID#6 (see Annex B)	U/C	xx	PID

U = User Optional — the parameter may be present or not

C = Conditional — if a PID is not supported but requested then there shall be no response for that PID

continued

1 … 4 data bytes as required

6.1.2.4 Request current powertrain diagnostic data response message definition (report PID values)

Table 111 — Request current powertrain diagnostic data response message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	Request current powertrain diagnostic data response SID	M	41	SIDPR
	data record of 1st supported PID = [			PIDREC_
#2	PID#1	M	xx	PID
#3	data A,	M	xx	DATA_A
#4	data B,	C1	xx	DATA_B
#5	data C,	C1	xx	DATA_C
#6	data D]	C1	xx	DATA_D
:	:	:	:	:
	data record of mth supported PID = [			PIDREC_
#n-4	PID#m	C2	xx	PID
#n-3	data A,	C2	xx	DATA_A
#n-2	data B,	C3	xx	DATA_B
#n-1	data C,	C3	xx	DATA_C
#n	data D]	C3	xx	DATA_D

C1 = Conditional — "data B - D" depend on selected PID value

C2 = Conditional — parameter is only present if supported by the ECU

C3 = Conditional — parameters and values for "data B - D" depend on selected PID number and are only included if PID is supported by the ECU

Annex B: PIDs for services $01 and $02 scaling & definition

Table B.6 — PID $04 - $05 definition

PID (hex)	Description	Data byte	Min. value	Max. value	Scaling/bit	External test equipment SI (Metric) / English display
04	Calculated LOAD Value!	A	0 %	100 %	100/255 %	LOAD_PCT: xxx.x %
05	Engine Coolant Temperature	A	-40 °C	+215 °C	1 °C with -40 °C offset	ECT: xxx °C (xxx °F)
	ECT shall display engine coolant temperature derived from an engine coolant temperature sensor or a cylinder head temperature sensor. Many diesels do not use either sensor and may substitute Engine Oil Temperature instead.					

Note: Conversion between physical and hexadecimal coded values

Physical value	***Hexadecimal coded value*** *(Dual Offset Code)*				
			n = 8bit	n = 16bit	n = 32bit
PhysMax	HexMax	$2^n - 1$	255_D = FF_H	$65\ 536_D$ = $FFFF_H$	$4\ 294\ 967\ 295_D$ = $FFFF\ FFFF_H$
PhysValue	HexValue				
PhysMin	HexMin	0	0	0	0

Phys → Hex: $$HexValue = \frac{PhysValue - PhysMin}{PhysMax - PhysMin} \cdot (2^n - 1)$$

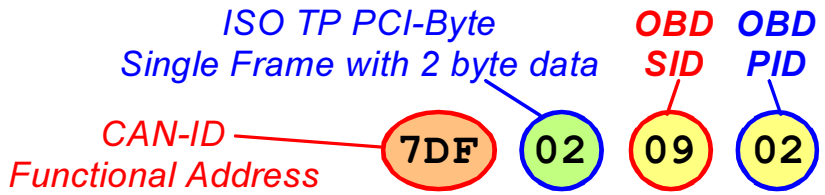
Hex → Phys: $$PhysValue = \frac{HexValue}{2^n - 1} \cdot (PhysMax - PhysMin) + PhysMin$$

e.g. Coolant Temperature Hex 66H

$$\mathbf{PhysValue = \frac{66_H = 102_D}{2^8 - 1} \cdot (215°C - (-40°C)) + (-40°C) = 62°C}$$

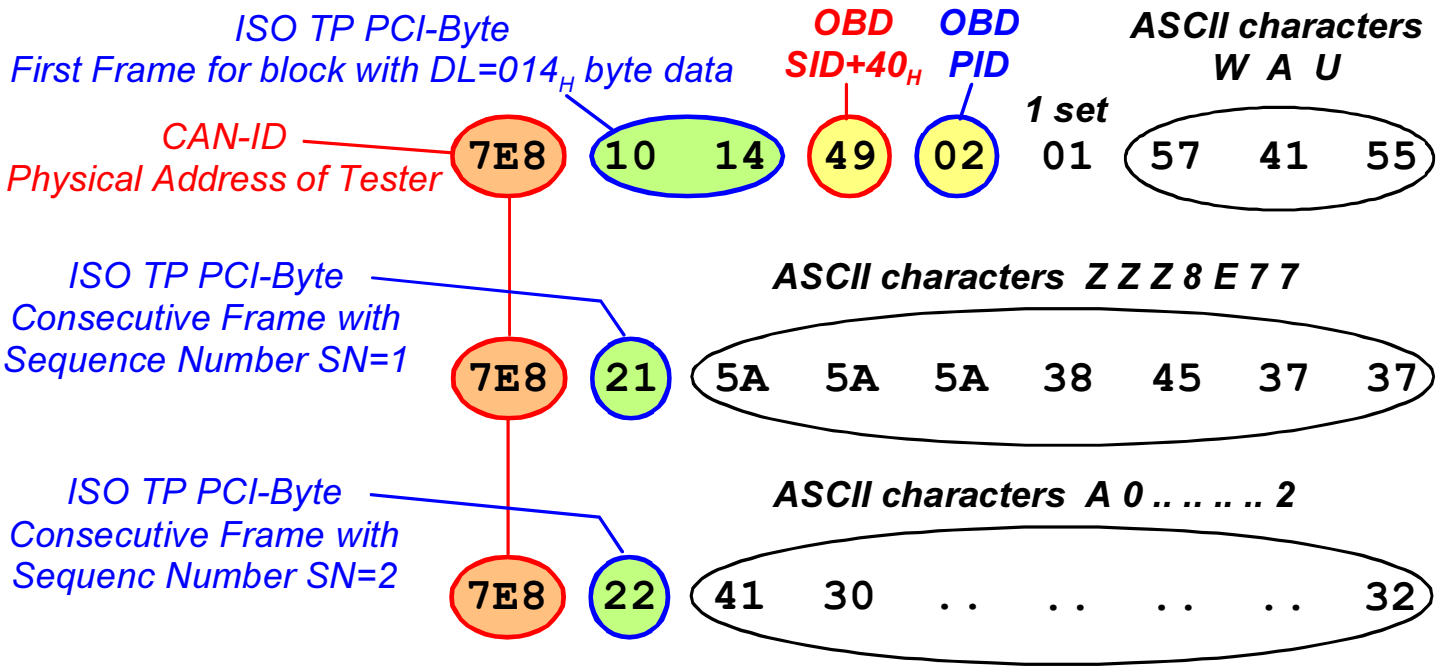
Example: Read Vehicle Identification Number Service SID=09_H, PID=02_H

- Tester request to read VIN (Vehicle Identification Number)



Request format as above
PID=02 … VIN

- ECU response



VIN

=

data set count

here 01_H, shows how many VINs will follow

+

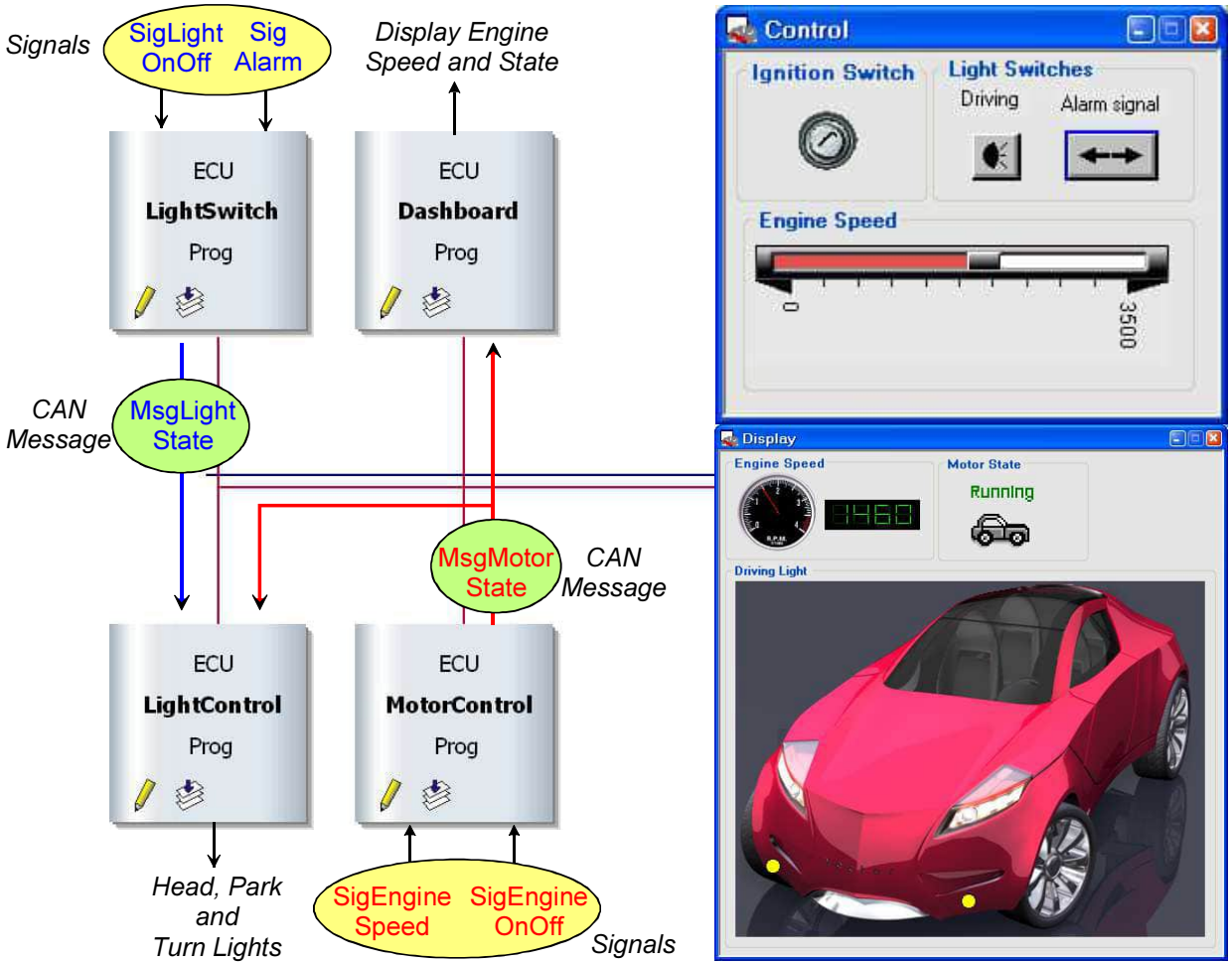
17 ASCII characters

Note: Flow control frame not shown here

3.11 Network Communication Design

The typical steps when designing network communication shall be demonstrated with an example. Our simple system has 4 ECUs connected via a CAN bus with the following functions:

- **Engine**: The current engine speed and state of the ignition switch shall be visualized by dashboard instruments.
- **Light**: The driver can turn on the car's head lights and let blink the turn lights in emergency situations (called Alarm Signal here) by switches, connected to the Light Switch ECU. This ECU also generates the blink frequency. The lights are connected to the Light Control ECU. The head light can only be turned on, when the engine is on, otherwise only the park light will be on.
- Note: The design is simulated with an Evaluation version of **Vector Informatik's Network Simulation Tool CANoe**, Project HE_Easy.



- Step 0: **E/E Architecture**

Which ECUs (network nodes) are included in the system, which functions do they have and over which bus systems are they connected?

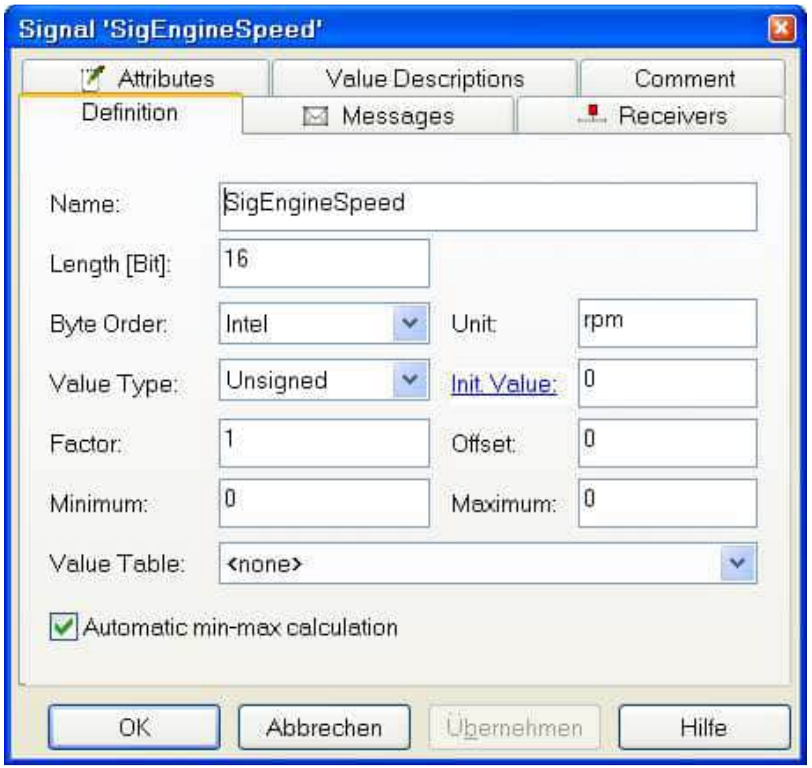
This step is not really part of the network design, but result of the previous definition of the E/E architecture, which already had to take basic communication aspects into account.

Note: Normally, an ECU is identical with one network node. Only when an ECU is connected to more than one bus system, e.g. a gateway, an ECU contains several network nodes, one for each bus.

- Step 1: **Defining Signals**

Signals are interface parameters of those software functions, which need to communicate between two or more ECUs. Signals have the well known characteristics of typical C/C++ variables:

- Signal name
- Encoding: Length n in bit and type (signed or unsigned)
  Binary value range unsigned: $0 \ldots 2^n - 1$
  signed: $-2^{n-1} \ldots +2^{n-1}-1$
- Byte order for multi-byte values
  Intel = Little Endian = Least Significant Byte first or
  Motorola = Big Endian = Most Significant Byte first.
- Optional: Physical unit and relation between raw computer representation and physical value, e.g.
  phys_value = factor * raw_value + offset
- Event-triggered or time-triggered (periodic, cyclic) message. Period or minimal inter-arrival time.



Often several signals must be handled in combination, to ensure data consistency. Such signals are handled in a *Signal Group* (same as a structure in C or a record in Pascal).

Signals in the Example system:

Name	*Encoding*	*Physical*	*Attributes*
SigLightOnOff	1 bit	1=on 0=off	Periodic ≤ 100 ms
SigAlarm	1 bit	1=on 0=off	Periodic ≤ 100 ms
SigEngineOnOff	1 bit	1=on 0=off	Periodic ≤ 10 ms
SigEngineSpeed	16 bit unsigned	1 bit per rpm	Periodic ≤ 10 ms

- Step 2: **Communication Matrix** - Assigning Signals to Sender and Receiver ECUs

  Note: Each signal has exactly **one** sender and one or more receiver ECUs.

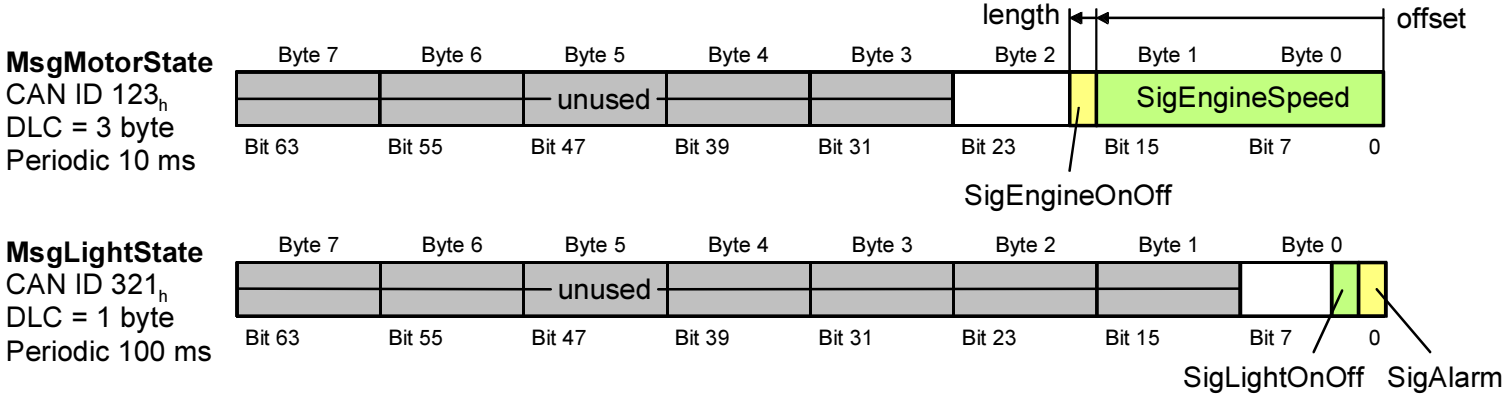
  Communication Matrix of the example system (Sender = Transmitter TX, Receiver = RX):

Signal / *Node*	*Light Switch ECU*	*Dashboard ECU*	*Motor Control ECU*	*Light control ECU*
SigLightOnOff	TX			RX
SigAlarm	TX			RX
SigEngineOnOff		RX	TX	RX
SigEngineSpeed		RX	TX	

- Step 3: Combining Signals into **Messages**

  Due to protocol overhead, sending each signal in a separate message would not be efficient, thus several signals are combined into one message according to the following criteria:

  - Only signals of the same sender ECU can be combined.
  - Event-triggered messages, which are triggered by the same event should be combined.
  - Time-triggered messages, which shall or may have the same period, should be combined.
  - Messages which have the same receiver(s) should be combined (to simplify acceptance filtering on the receiver side).
  - All signals of a signal group should go into a single message.
  - Integer values should begin on byte borders (byte offset) to simplify software processing, binary values (bits and bitfield) may start at any bit position (bit offset).



Now the communication matrix can be rewritten message-based rather than signal-based

Communication Matrix: All networks

Signals/Node	LightSwitch	Dashboard	MotorControl	LightControl
SigLightOnOff	<Tx> MsgLightState (0x321)			MsgLightState (0x321)
SigAlarm	<Tx> MsgLightState (0x321)			MsgLightState (0x321)
SigEngineOnOff		MsgMotorState (0x123)	<Tx> MsgMotorState (0x123)	MsgMotorState (0x123)
SigEngineSpeed		MsgMotorState (0x123)	<Tx> MsgMotorState (0x123)	

- Step 4: Assign **Message Priority** (CAN) or plan **Schedule** (LIN, FlexRay)

  CAN: For each message assign CAN ID = message priority, see CAN chapter
  LIN: For each message assign LIN PID and slot in the schedule, see LIN chapter
  FleyRay: For each message assign Frame ID = slot in the schedule, see FlexRay chapter

- Step 5: Compute bus load & worst-case latencies and **verify** by simulation

CANoe Simulation: Message trace

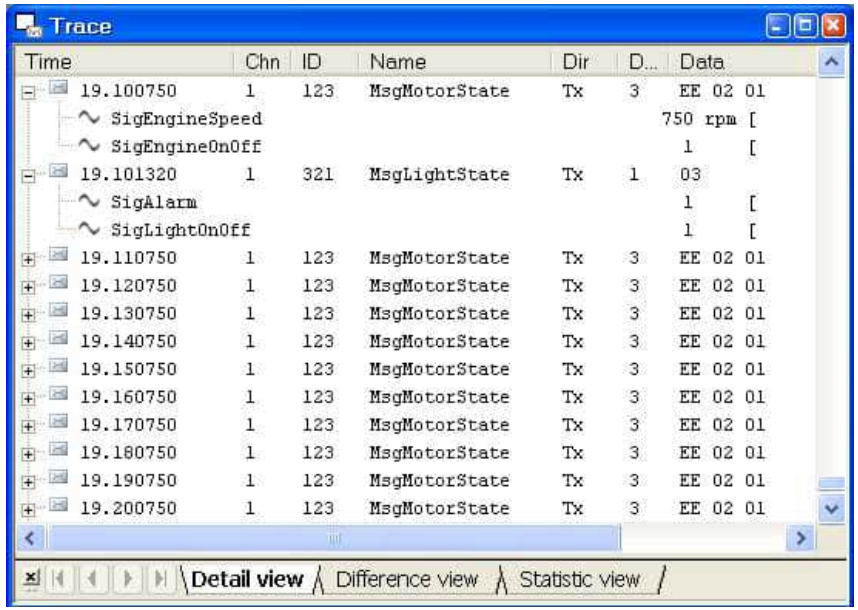
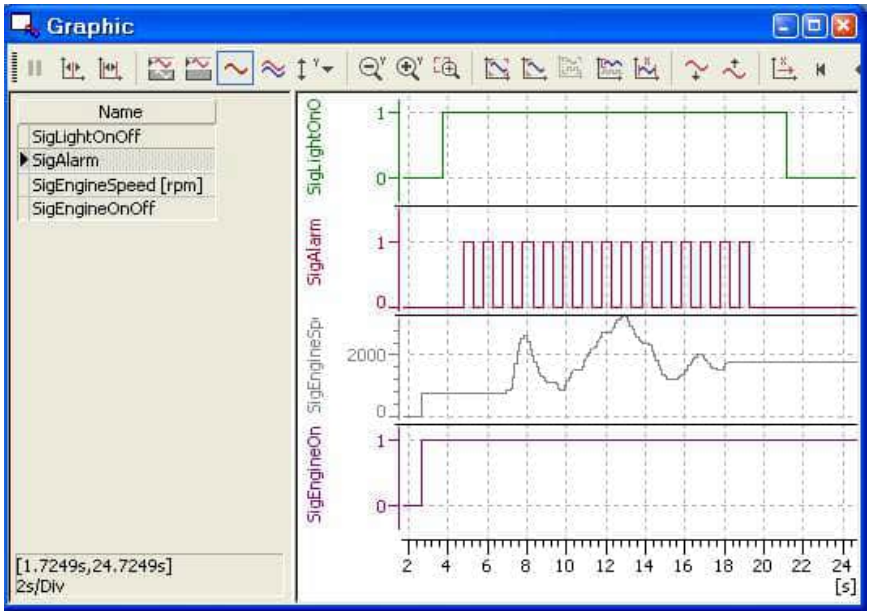
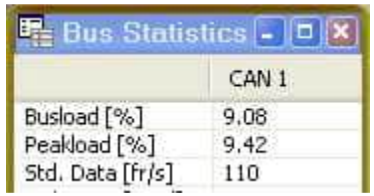


Diagram of received signals





Bus statistics

f_{Bit}=100 kbit/s

Calculated average bus load

$$BL = 1.25 \cdot \left[\frac{(24+44)\text{bit} / 100\text{kbit/s}}{10\text{ms}} + \frac{(8+44)\text{bit} / 100\text{kbit/s}}{100\text{ms}}\right]$$

$$= 1.25 \cdot 7.3\% = 9.2\%$$ (25% worst case stuffing overhead)

- Repeat steps 3 and 4, if requirements are not met.

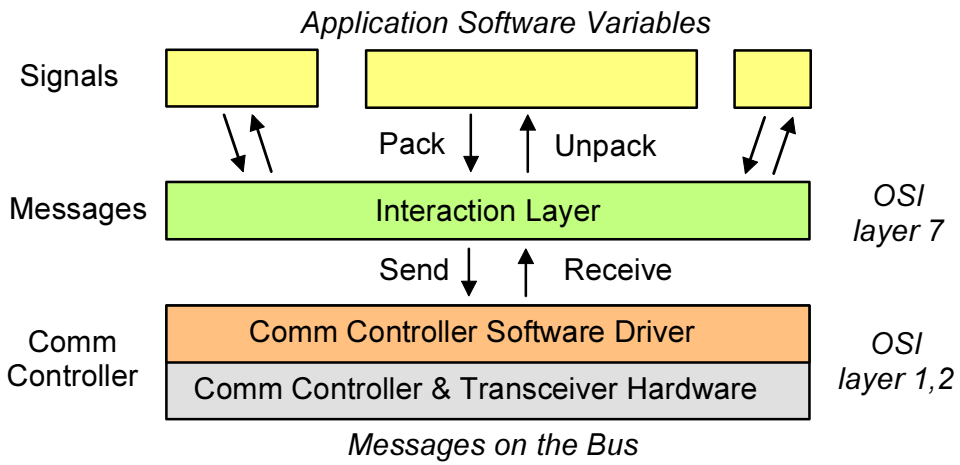
Network Database

- Due to a lack of tools, when CAN networking was introduced in the early 1990s, signals, message definitions and communication matrices were described using plain text in text processors like Word or spreadsheets like Excel.
- With the introduction of network analysis and simulation tools like Vector Informatik's CANalyzer and CANoe (www.vector-informatik.de), database-like descriptions were introduced:

CANdb	"CAN Database"	Industry standard format for CAN (ASCII-text format, by Vector Informatik, proprietary, but also used by many other tool vendors [T3])
LDF	"LIN Description Format"	Open standard for LIN (documented ASCII-text format by LIN Consortium [S3])
FIBEX	"Field Bus Exchange"	Open standard for CAN, LIN, FlexRay, MOST (documented XML format by ASAM [S14])
ARXML	"AUTOSAR XML"	Open standard for Automotive Systems (communication subset, XML format by AUTOSAR [S17])

- Tool chains typically allow to
  - Define the network database
  - Simulate the network (*Full Simulation* or *Rest-Bus Simulation* see below)
  - Generate C code for the implementation of the *Interaction Layer* (see below) and configure *Communication Controller Software Driver*
  - Analyze communication on working networks in real-time

Interaction Layer

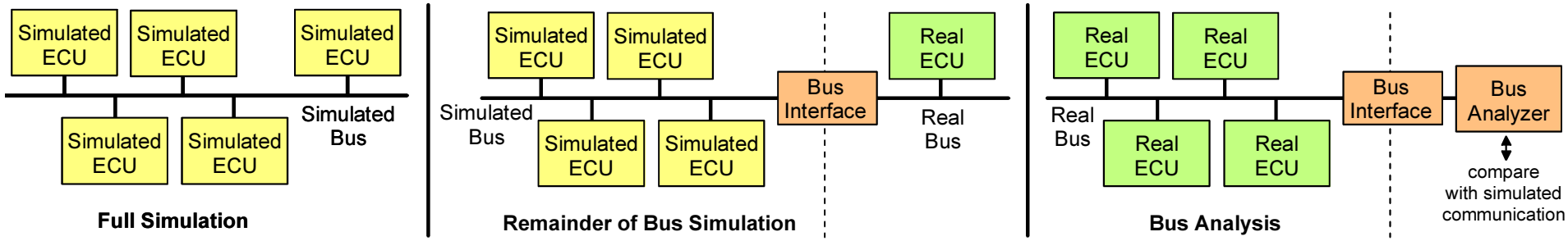


The ECU software, which packs / unpacks signals to / from messages and triggers the transmission is called the **Interaction Layer**, e.g. **OSEK or AUTOSAR COM** [S13]

In a bus simulation system, the interaction layer must be simulated too to generate the event-triggered or time-triggered messages.

Interaction layer is a low-overhead substitute for the application layer 7 of ISO/OSI protocol stacks for on-board communication, where layers 3 to 6 are not used.

Remainder of Bus Simulation (in German: Restbus Simulation)



- Results of the network design phase can be used during ECU development, when the real ECU is simulated against a model of the remainder of the system and for verification of the real system.

Only ECU communication is simulated, not the other ECU functions.

Further Reading

Books or Lectures

[3.1] Most books about automotive electronics (see chapter 2) have a section on networking and bus systems.

[3.2] D. Paret: Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe by Wire. Wiley & Sons, 1st edition, 2007

[3.3] W. Zimmermann, R. Schmidgall: Bussysteme in der Fahrzeugtechnik. Springer Vieweg, 5th edition, 2014 (in German), http://www.wernerzimmermann.name/automotive-networks-book.html

[3.4] Rausch, M.: FlexRay. Grundlagen, Funktionsweise, Anwendung, Hanser, 1st edition 2007 (in German)

[3.5] Grzemba, A.; von der Wense, H.C.: LIN-Bus, Franzis Verlag, 1st edition 2005 (in German)

[3.6] Etschberger, K.: Controller Area Network, Hanser Verlag, 3rd edition 2006 (in German)

[3.7] Lawrenz, K.: Controller Area Network, VDE Verlag, 5th edition 2011 (in German)

[3.8] Grzemba, A.: MOST – The Automotive Multimedia Network, free download from http://www.mostcooperation.com/publications/mostbook/index.html

[3.9] T. Form: Datenbussysteme in Straßenfahrzeugen. Lecture Manuscript, TU Braunschweig, 2006, www.ifr.ing.tu-bs.de/lehre/vorlesungen.php (in German)

[3.10] P. Koopmann: Distributed Embedded Systems. Lecture 18-649 Carnegie Mellon University, Spring 2008, www.ece.cmu.edu/~koopman

[3.11] Vector Informatik: E-Learning Modules for CAN, LIN, FlexRay and AUTOSAR. https://elearning.vector.com/vl_index_en.html

[3.16] C. Marscholik, P. Subke: Road Vehicles – Diagnostic Communication. Huethig, 1st edition, 2008

[3.17] World-Wide-Harmonized OBD diagnostics ISO 27145

Further Reading

Standards

[S1] CAN and CAN-FD original Bosch specifications, www.can.bosch.com

[S2] CAN standard family ISO 11898: Road Vehicles – Controller area network. www.iso.org

[S3] LIN specification. www.lin-subbus.org, now standard family ISO 17987 1-7, www.iso.org

[S4] FlexRay specification. www.flexray.com, now ISO standard family ISO 17458, www.iso.org

[S5] MOST specification. www.mostcooperation.com

[S6] SAE J1850: Class B Data Communications Network Interface. www.sae.org

[S7] K-Line ISO 9141: Road vehicles -- Diagnostic systems -- Requirements for interchange of digital information

[S8] KWP 2000 diagnostics ISO 14230: Road-Vehicles Diagnostic Systems – Keyword Protocol 2000

[S9] UDS diagnostics ISO 14229: Road vehicles – Unified diagnostic systems (UDS)

[S10] UDS on CAN ISO 15765: Road vehicles – Diagnostics on Controller Area Networks (CAN)

[S11] OBD diagnostics ISO 15031: Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics

[S12] SAE J1939: Recommended Practice for a Serial Control and Communications Vehicle Network for On-Highway Equipment

[S13] OSEK/VDX Communication (OSEK COM Interaction Layer), www.osek-vdx.org and ISO 17356-4

[S14] ASAM FIBEX Field Bus Exchange Format, www.asam.net

All ISO (International Organization for Standardization) standards available from www.iso.org

All SAE (Society of Automotive Engineers) standards available from www.sae.org

ISO and SAE standards typically consist of several documents (*standard family*)

[S15] World-Wide-Harmonized OBD diagnostics ISO 27145

[S16] Diagnostics on IP (DoIP) ISO 13400

Further Reading

[S17] AUTOSAR, www.autosar.org

[S18] ISO 21111 Road vehicles – In-vehicle Ethernet, www.iso.org

[S19] TSN Time Sensitive Networking, https://1.ieee802.org/tsn

[S20] SOME/IP Scalable Service Oriented Middleware over IP, www.some-ip.com

Special Topics

[T1] A. Burns et.alt.: Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. University of Cork Report 408, 2006, www.cs.york.ac.uk/ftpdir/reports/YCS-2006-408.pdf
Based on previous work by K. Tindell et. alt. The paper has a comprehensive list of references.

[T2] T. Nolte, H. Hansson, C. Norström, S. Punnekkat: Using bit-stuffing distributions in CAN analysis. IEEE Real-Time Embedded Workshop, 2001

[T3] Vector Informatik GmbH: Network design and simulation tools: CANoe, DaVinci, CANdb, PreeVision see www.vector.com

[T4] R. De Andrade et.alt.: Analytical and Experimental Performance Evaluations of CAN FD Bus. IEEE https://ieeexplore.ieee.org/document/8338047

Free E-Learning Courses by Vector Informatik

[V1] Topics: CAN, LIN, FlexRay, Ethernet, AUTOSAR, …
https://elearning.vector.com/?lang=en

Chapter 4
System Safety and Reliability

Hochschule Esslingen
University of Applied Sciences

4.1 Basics 3
4.2 Reliability Calculus 8
4.3 Reliability Prediction for Electronic 12
4.4 Analyzing System Safety 1: Fault Tree Analysis FTA 15
4.5 System Monitoring, Redundancy 19
4.6 Analyzing System Safety 2: Failure Modes and Effects Analysis FMEA 28
4.7 Safety Integrity Levels and Functional Safety 29
4.8 Cybersecurity 37

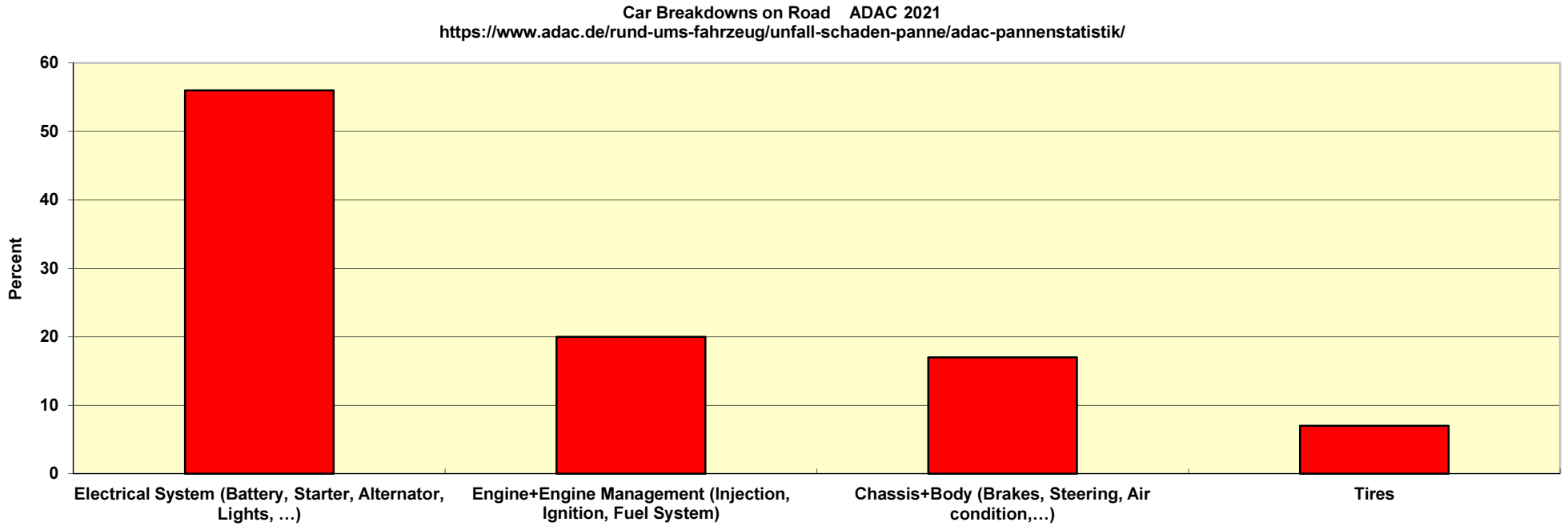
Further Reading 38
Appendix 40
Glossary 53

If anything can go wrong, it will … (Murphy's Law: Edward Murphy, US Air Force, 1949)



(Source: www.handelsblatt.com)

4.1 Basics



Breakdowns per 1000 units for 5 year old cars:

- Compact cars (VW Golf, MercedesBenz A, BMW 1, …) Best in class: 8 Worst in class: 38
- Lower Luxury class (Audi A6, Mercedes Benz E, BMW 5, …) Best in class: 11 Worst in class: 21

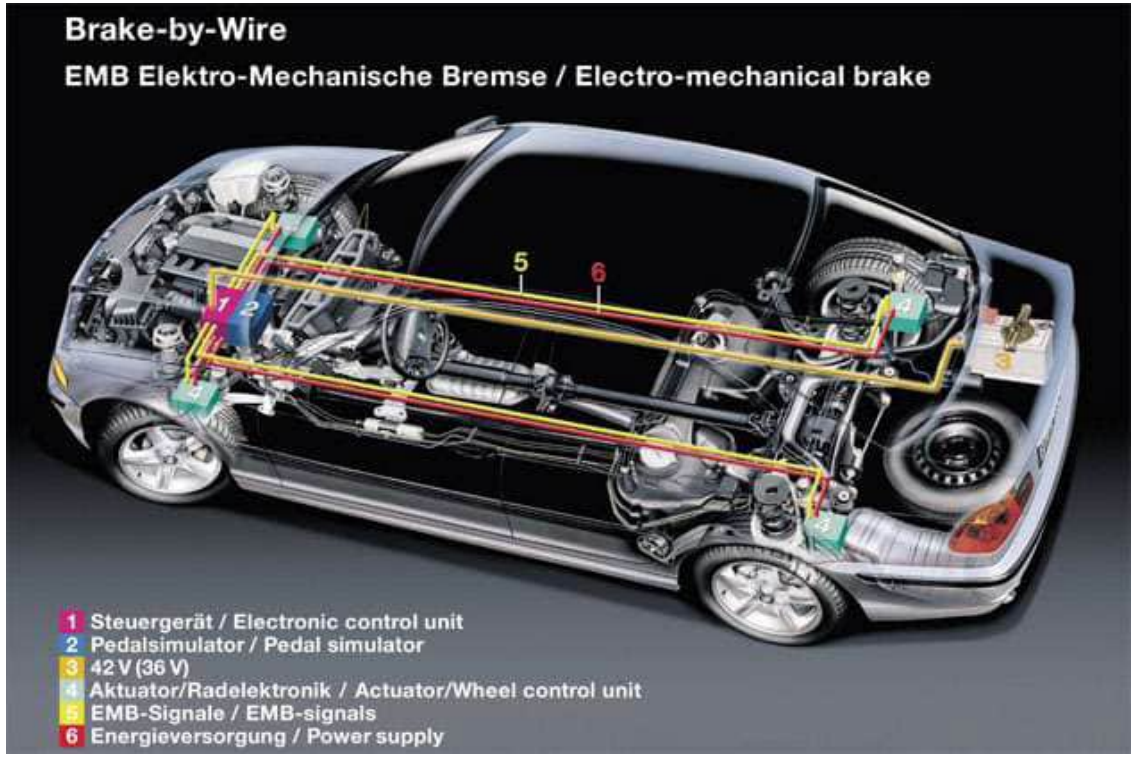
→ Trivial electrical failures (battery, generator, wiring harness, spark plugs, connectors, …) cause > 50% of all problems

→ The more E/E functionality a car has, the more likely E/E failures will occur

→ **Reliability** as part of product quality is an important issue

Example: Concept study for a brake-by-wire system:

- Actuation of the brakes by electromagnetic actuators controlled by an ECU
- Advantage: Avoids the hydraulic system
- Problem: High electrical peak power necessary



Source: Article "by-wire - …", www.heise.de/autos

How safe is it compared to the proven hydraulical dual-circuit braking system with mechanical parking brake?

- What if the battery is discharged?
- What if one of the wires is broken?
- What if the software in the ECU crashes?

→ **Safety** analysis necessary

- Customer expectations (reliability) and legal requirements (safety)
- Trade off between cost and technical perfection required

Some Technical Terms

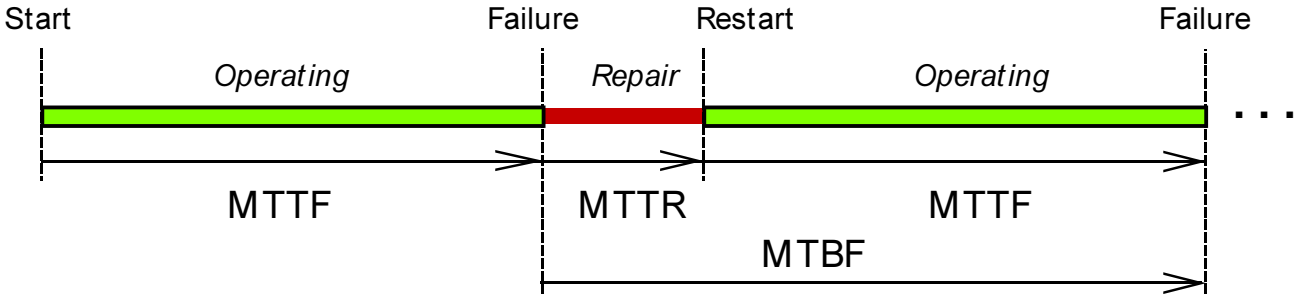
- **Reliability** and **Dependability** (in German: Zuverlässigkeit und Verlässlichkeit)
  Capability to perform its function as desired, i.e. without failure in any of its components. In newer publications *Reliability* is used for the quantitative aspect, whereas *Dependability* is used for the qualitative aspect.
- **Safety** (in German: Sicherheit)
  Capability to operate without hurting people or hardware. When a failure occurs, a system may switch into a safe state, e.g. with no or limited functionality (**Fail Safe / Fail Stop**), or may continue to work using redundant components (**Failure Tolerant / Fail Operational**).
- **Security** → Not discussed in this module (in German: Zugangsschutz)
  Capability to allow access only to authorized users, keep information confidential, ...

- **Fault** e.g. leakage in brake hydraulics (in German: Fehler, Störung, Defekt)
  A hardware component **defect** or a software **bug**.
- **Failure** e.g. car does not brake (in German: Ausfall, Fehlverhalten)
  A fault's **system level effect**, may occur immediately or delayed until the faulty component or the buggy function is used.
- **Hazard** (in German: Gefährdung)
  The potential of a failure to injure or kill men or damage or destroy goods or data.
- **Risk** (in German: Risiko)
  Assessment of failures based on a combination of hazard potential and failure probability.

Classification of Failures

- **Failure types/modes** *= What kind of failure?*
  - **Parameter** failure – **Partial** failure – **Total** failure (blackout)
- **Failure duration** *= How long does the failure occur?*
  - **Permanent** failure – **Temporary** failure
- **Failure probability** *= How often does the failure occur?*
  - **Systematic** failure: All devices do contain the same failure, the failure is reproducible if the same operating conditions are applied
  - **Stochastic** (random) failure: Failure, which does not occur in all devices due to tolerances in manufacturing processes.
- **Failure occurrence** *= When does the failure occur?*
  - **Early** failures (in the automotive industry: 0 km failures)
  - **Mid of life** failure (in the automotive industry: field failures)
  - **End of life** failures: Failure due to natural wear out
- **Failure cause** *=Why does the failure occur?*
  - **Design** error (=systematic) → Avoid by **proven design methods and processes**
  - **Manufacturing** error (=stochastic (in most cases)) → Detect by **factory testing**
  - Operator error, overstress, wear out, EMC → Can't avoid, handle by **Functional Safety**
    - **Monitor** (detect failure) - **Act** (switch to safe state) - **Notify** the user (or log error)

Measuring reliability



- **Mean Time to Failure MTTF**: Average time of operation, until a failure will occur

  If n out of N devices fail at times $t_1, t_2, \ldots, t_n$ within test time T:

$$\textbf{MTTF} = \frac{t_1 + t_2 + \ldots + t_n + (N-n) \cdot T}{N}$$

  if n=0: MTTF = T

- **Mean Time to Recovery (Repair) MTTR**: Average time to repair a failed system. Time for preventive maintenance is treated like repair time.

- **Mean Time Between Failures MTBF** = MTTR + MTTF

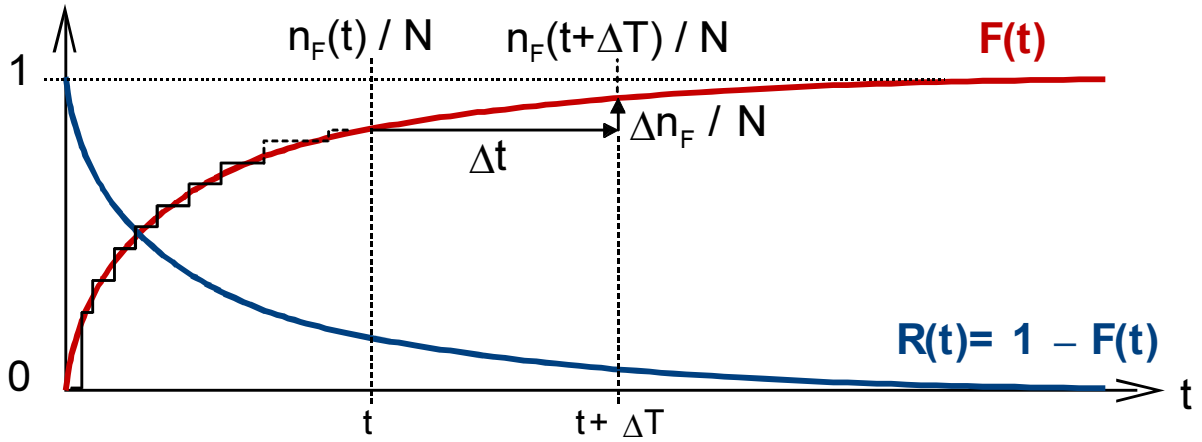
- **Availability**

$$AV = \frac{MTTF}{MTBF} = 1 - \frac{MTTR}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

Example: A system has AV = 99% → within 1 year the system is not available for 0.01 · 8760 hrs = 87.6 hrs = 3.7 days

4.2 Reliability Calculus

■ **Failure Probability F**: Relative failures over time



with	N	total number of units
	n_F (t)	number of failed units at time
	n_R (t) = N – n_F(t)	number of remaining units at time t.

If N is big enough and failures are statistically independent, probabilities can be calculated:

$$F(t) = \frac{n_F(t)}{N}$$ Failure probability with $F(t=0) = 0$ and $0 \le F(t) \le 1$ (1)

$$R(t) = \frac{n_R(t)}{N} = 1 - F(t)$$ Reliability = probability, that a unit is still good at time t (2)

■ **Failure Density**

$$f(t) = \lim_{\Delta T \to 0} \frac{n_F(t+\Delta T) - n_F(t)}{\Delta T \cdot N} = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt} \qquad (3)$$

■ **Failure Rate**

$$\lambda(t) = \frac{-1}{R(t)} \cdot \frac{dR(t)}{dt} = \frac{1}{1-F(t)} \cdot \frac{dF(t)}{dt} \qquad (4)$$

Probability, that one of the remaining devices fails in time interval t … t+ΔT (PFH … Probability of Failure per Hour)

Note: Relation between F(t), R(t) and λ(t):

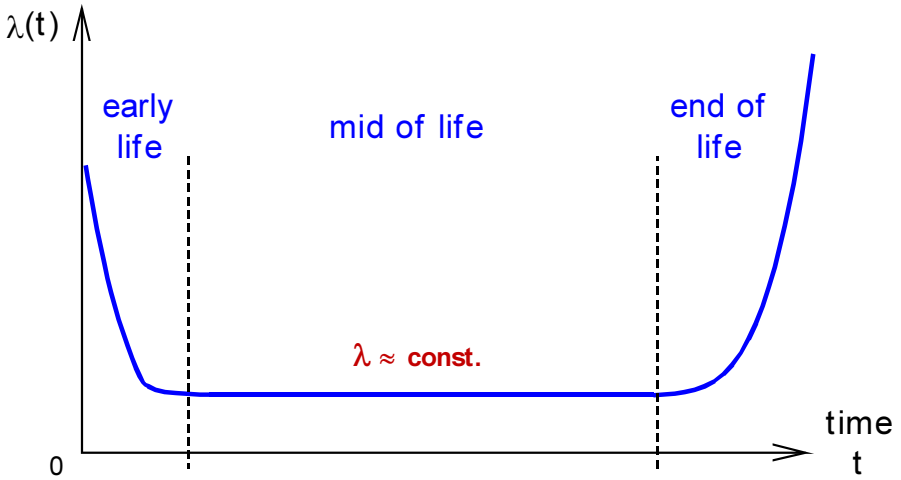
From eq. (4) $\lambda(t) \cdot R(t) = -\frac{dR(t)}{dt} \rightarrow \lambda(t) \cdot R(t) + \frac{dR(t)}{dt} = 0$ with $R(t=0) = 1$

$$\rightarrow \quad R(t) = e^{-\int_0^t \lambda(\tau)\, d\tau} \qquad (5) \qquad \text{and} \qquad F(t) = 1 - R(t) \qquad (6)$$

$$\rightarrow \quad MTTF = \int_0^\infty t \cdot f(t)\, dt = \int_0^\infty R(t)\, dt \qquad (7)$$

Typical failure rate
over product lifetime

(bathtub curve)

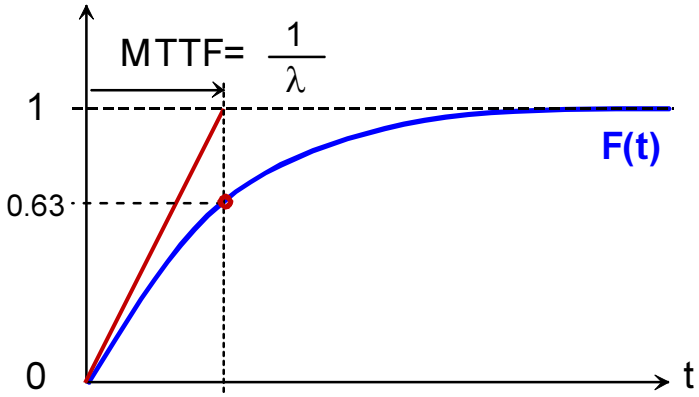


Exponential Distribution

During the **mid of life phase** often the failure rate can be considered to be constant

- if λ = const.: $R(t) = e^{-\lambda \cdot t}$ (8a) $F(t) = 1 - e^{-\lambda \cdot t}$ (8b) $MTTF = \frac{1}{\lambda}$ (8c)
- if $\lambda \cdot t \ll 1$ (and λ = const.) this can be simplified to

$$R(t) \approx 1 - \lambda \cdot t \qquad F(t) \approx \lambda \cdot t$$



If you use the linear approximation, please be careful to ensure

$$0 \le R(t), F(t) \le 1.$$

The approximation error for F(t) is less than 10% for $t < 0.1$ MTTF

Typical units . . .

. . . for **failure probability F**:	**1 ppm**	**= 10^{-6} = 1 part per million**
	10000 ppm	= 1%
. . . for **failure rate**:	**1 fit**	**= 10^{-9}/h** = 1 ppm/1000h =1 failure per 10^9 h

Example:

An electronic control unit (ECU) has a failure rate of $\lambda = 500$ fit?

How many ECUs out of a production batch of 5 Mio. ECUs will fail in the first year of continuous operation?

$$F = \lambda \cdot T = 500 \cdot 10^{-9}/\text{h} \cdot 365 \text{ days} \cdot 24 \text{ h} \approx 4400 \text{ ppm} = 0.44\%$$

$$n_F = F \cdot N = 4380 \text{ ppm} \cdot 5 \text{ Mio.} \approx 22000$$

How long will each ECU survive with 95% likelihood?

$$R = e^{-\lambda \cdot T} = 0.95 \quad \rightarrow \quad T = -\frac{1}{\lambda} \cdot \ln 0.95 = 100\,000 \text{ h} \approx 12 \text{ a}$$

4.3 Reliability Prediction for Electronic Hardware

Assumptions

- Only **random errors** which occur during normal operation of the device are considered, **no systematic errors** in design or manufacturing.
- Only **mid of life failures** are considered, assuming $\lambda \approx$ const. (no early/end of life failures)

Failure Rate Data for Components

- Failure rates **cannot be precalculated** from geometrical and material properties.
- Failure rates of electronic devices are **too small** (1 … 1000 fit) **to be measured by testing** individual devices, but can be statistically collected for classes of devices.
- Actual **failure rates are company secrets**. Publicly available data comes from US military (**MIL Handbook 217**), the Society of Automotive Engineers (SAE 870050), Bellcore (TR/SR-332) or the International Engineering Consortium (IEC 61709).
- MIL and others do publish **basic device failure rates** λ_B for nominal operating conditions. Different operating conditions (within the specified min – max range) can be taken into account by correction factors (stress factors): $\lambda = \lambda_B \cdot c_\vartheta \cdot c_M \cdot \ldots$

  temperature stress factor $c_\vartheta = 2^{\Delta\vartheta / 10K}$ with $\Delta\vartheta$ … difference between actual and nominal temperature (i.e. a 10K temperature increase doubles the failure rate (Arrhenius law)

  mechanical stress factor c_M = 0.5 … stationary, 1 … ground mobile, 2 … in flight

  voltage or current stress indirectly included via their temperature effect

  additional "stress factors" can be used to assess new technologies or other types of risk

Base failure rates (Estimates, based on MIL HDBK 217E)

Type	*Base rate* λ_B		*Type*	*Base rate* λ_B	
Semiconductors (active components):			*Passive components:*		
CMOS microcontroller	200 fit		Metal file resistor	0.3 fit	
EPROM, RAM	100 fit		Film capacitor	0.5 fit	
CMOS logic IC	20 fit		Ceramic capacitor	0.3 fit	
Operational amplifier (OP)	50 fit	+ 0.5 fit per pin *1	Aluminum capacitor	10 fit	+ 0.5 fit per pin *1
Small signal transistor/diode	0.5 fit		Inductor (coil, transformer)	5 fit	
Power transistor/diode	50 fit		Quartz crystal	200 fit	
LED	25 fit				
Optocoupler	100 fit				
Electromechanical components (λ not temperature dependent)					
Switch	5 fit				
Relay	30 fit				
Connector per pin	5 fit				

Operating conditions: Temperature 45°C ambient, 85°C junction; ground mobile

*1 Per pin failure rate and failure rate of mechanical components not temperature dependent.

Typical Operating Times

- Passenger cars 300h/year
- Trucks 2000h/year
- TV sets 1500h/year
- Telephones, fax 8760h/year
- Automation equipment 1 working shift = 2000h/year 3 working shifts = 6000h/year
- If operating time is not known, use calendar time (1d=24h, 1w=168h, 1y = 8760h)

MTTF estimation for circuits: Parts Count Method

- Sum of the failure rates λ_i of all i=1…n components: $\lambda_{circuit} = \sum_{i=0}^{n} \lambda_i$ $MTTF_{circuit} = \frac{1}{\lambda_{circuit}}$
- This method does not take into account, how a component fails (parameter drift, open or short circuit, …), which effect this failure has on the circuits overall function and how this failure propagates to the system level.

How to address Early Life Failures?

- Can be reduced by **burn in**/**run in** or by **better production testing**

How to address End of Life Failures (Wear out)?

- Predictive maintenance → change components before they fail

4.4 Analyzing System Safety 1: Fault Tree Analysis FTA

- Not all failures are safety related, thus the parts count approach is too pessimistic with respect to safety.
- To get a more reasonable estimate, we should distinguish between "safe" (uncritical) and dangerous failures:

$$\lambda_{total} = \lambda_S + \lambda_D = \lambda_S + \lambda_{DD} + \lambda_{DU}$$

Dangerous failures (λ_D) can be further divided into detectable (λ_{DD}) and undetectable (λ_{DU}) ones. Assuming, that for detectable failures appropriate countermeasures can be taken before a safety critical situation occurs, the remaining safety risk is given by λ_{DU}.

If no better estimate is available, IEC 61508 recommends to assume $\lambda_D = 0.5\ \lambda_{total}$. This standard also defines the **Safe Failure Fraction SFF** = $(\lambda_S + \lambda_{DD}) / \lambda_{total}$. (SFF is similar to SPFM and LFM in ISO 26262, see chapter 4.7).

To find out, which failures are safety related, a cause and effects analysis is needed:

- **Fault Tree analysis FTA**

  Top down ("deductive") approach: Identify system level safety critical events and track them down to component failures

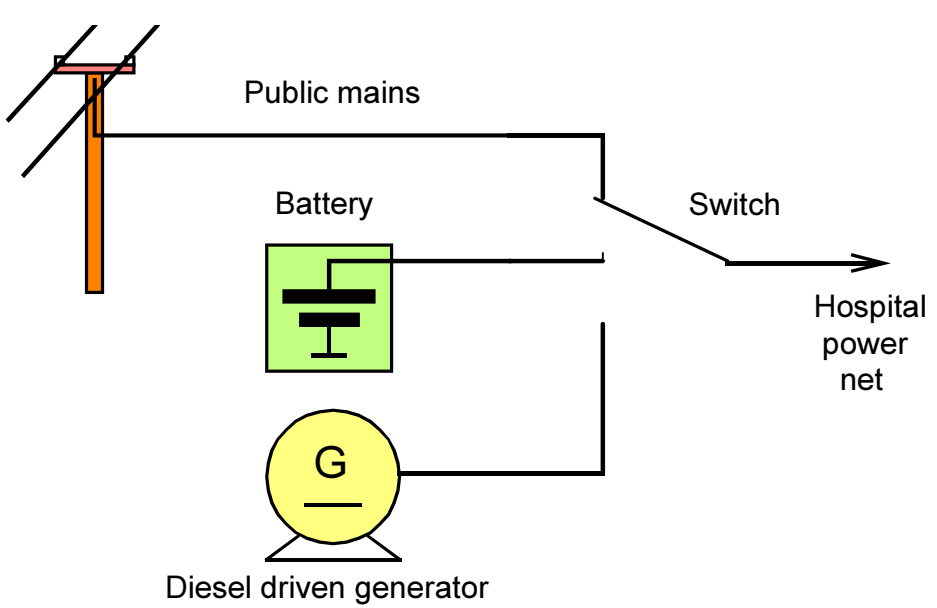
  Standardized by IEC 61025, DIN 25424 and various industry standards, e.g. SAE ARP 4761

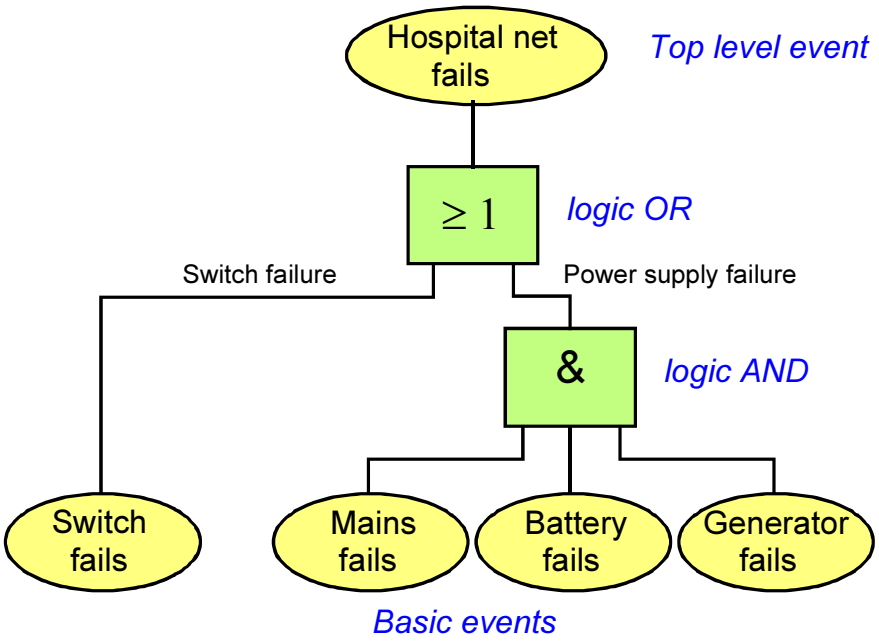
- **Failure Modes and Effects Analysis FMEA**

  Bottom up ("inductive") approach: Identify component failures, track effect up to system level

  Standardized by IEC 60812, DIN 25448 and various industry standards, e.g. SAE J1739

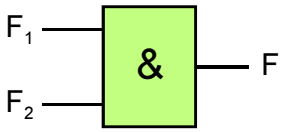
FTA example: Electrical power supply for a hospital





- **Fault trees describe**, which **system failure (=top level event)** does have which **root cause(s) (=basic event)**. Basic events typically are component failures.
- **Basic events must be independent** on each other
- If a failure has more then one cause,
  - **causes** are **logically ANDed**, if the system fails only, when all causes occur,
  - **causes** are **logically ORed**, if the system fails already, when one of the causes occurs.
- A typical fault tree has several top level events and a multi-level hierarchy of ANDs and ORs

- **System level failure probability** as a function of component failure probabilities:



Logical AND

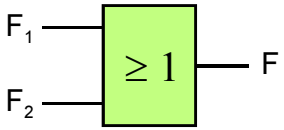
Failure event F occurs, if failure F_1 AND failure F_2 do occur

$F = F_1 \cdot F_2$

$R = 1 - F_1 \cdot F_2 = 1 - (1-R_1) \cdot (1-R_2)$

$= R_1 + R_2 - R_1 \cdot R_2$

Special case $F_1 \approx 1$, $F_2 << F_1$ → $F \approx F_2$



Logical OR

Failure event F occurs, if failure F_1 OR failure F_2 or both do occur

$F = 1 - R_1 \cdot R_2 = 1 - (1-F_1) \cdot (1-F_2)$

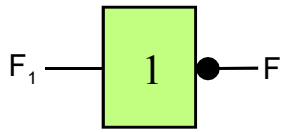
$= F_1 + F_2 - F_1 \cdot F_2$

$R = R_1 \cdot R_2$

Special case F_1 or $F_2 \approx 1$ → $F \approx 1$

Special case F_1 , $F_2 << 1$ → $F \approx F_1 + F_2$

Special case $F_1 >> F_2$ → $F \approx F_1$



Logical NOT

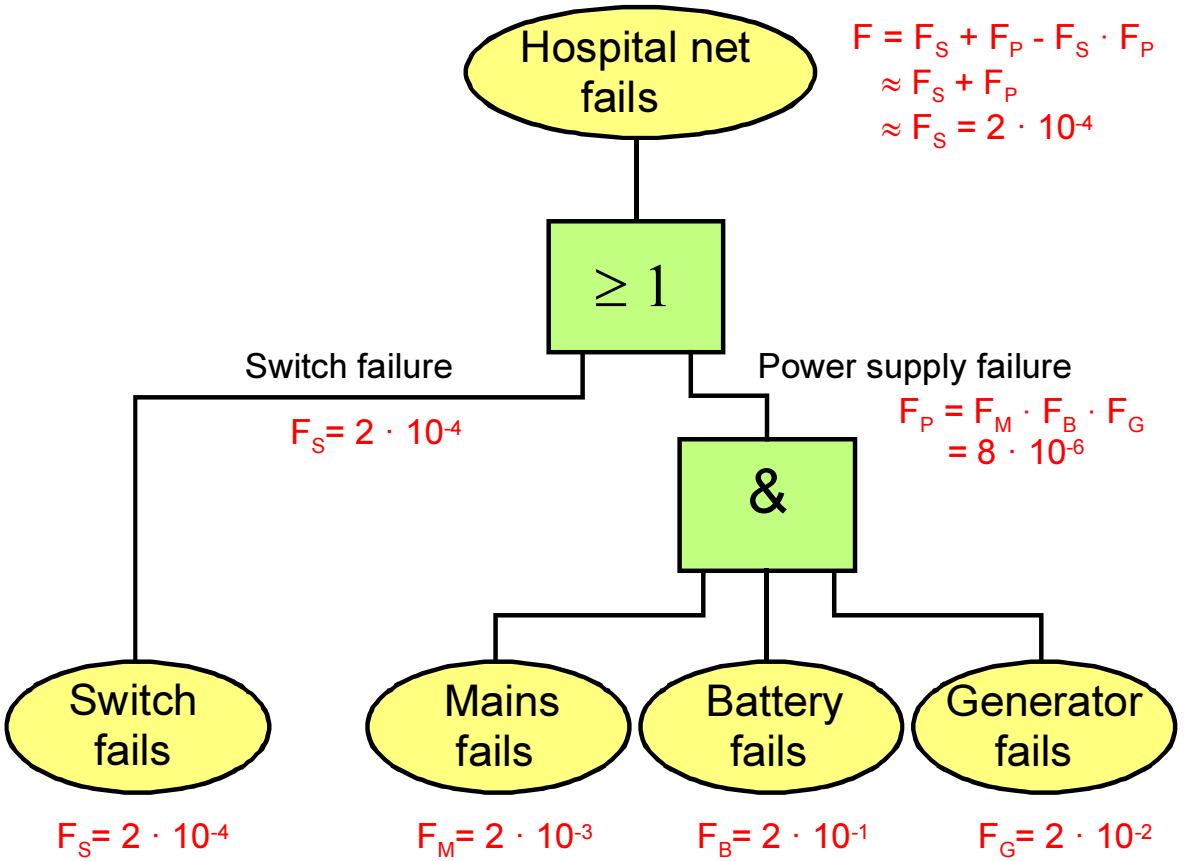
Event F occurs, if event F_1 does not occur

(NOT should not be used in FTAs, if possible)

$F = 1 - F_1$

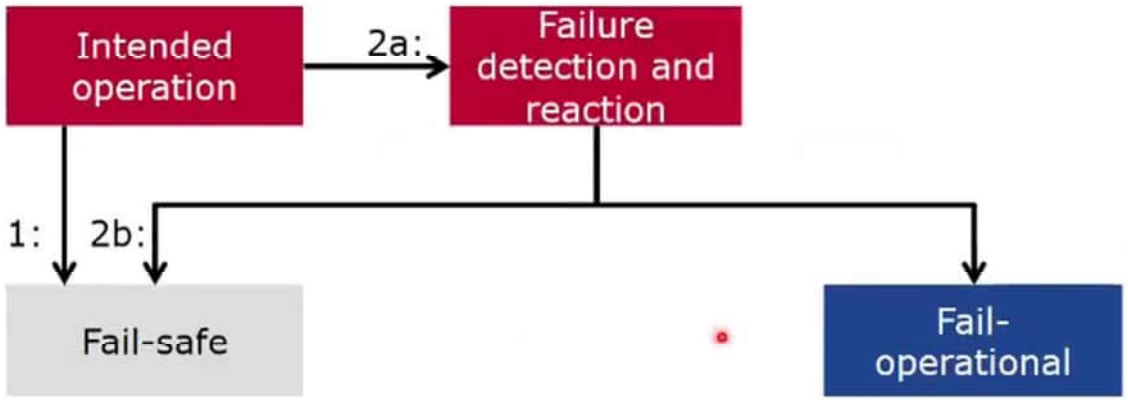
$R = 1 - R_1$

Example continued:



Example Data	λ	$F=1-e^{-\lambda \cdot T}$ for T=168h (1w)
Switch	10^{-6}/h = 1 kfit	200ppm = $2 \cdot 10^{-4}$
Mains	10^{-5}/h = 10 kfit	2000ppm = $2 \cdot 10^{-3}$
Generator	10^{-4}/h =100 kfit	20000 ppm = $2 \cdot 10^{-2}$
Battery	10^{-3}/h = 1 Mfit	200000ppm = $2 \cdot 10^{-1}$

4.5 System Monitoring, Redundancy and Diversity



- Bring the system into the fail-safe state to avoid any hazard.
- Two approaches:
  1. Fail-safe by design (default)
  2. Failure mitigation and transition to fail-safe state
- Sufficient for most "classic" automotive systems, often with mechanical back-up

- System remains operational
- E.g. degraded - but safe - operation mode.
- Availability of elements assuring the required safety
- Diverse / redundant architecture
- Required for continuous and automated safe operation

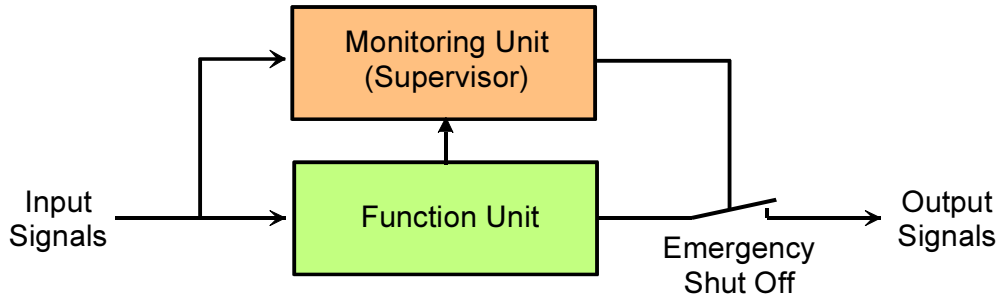
Source:
Vector Informatik

If failures cannot be avoided, apply a **Functional Safety** concept

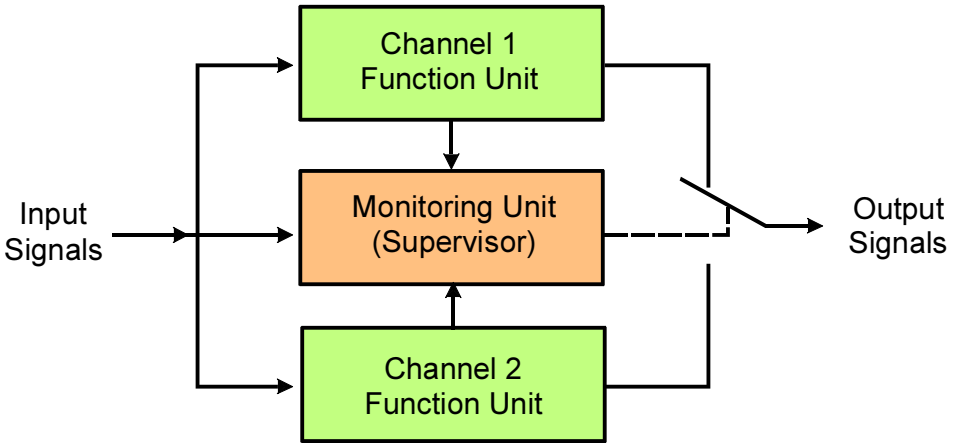
→ Detect failures by **monitoring**

→ React by switching the system into a **fail-safe** state (e.g. shut off)
OR use **redundancy** to keep system **fail-operational**

Structure of Safety Critical Systems



1 out of 1 system (1oo1): 1 functional unit



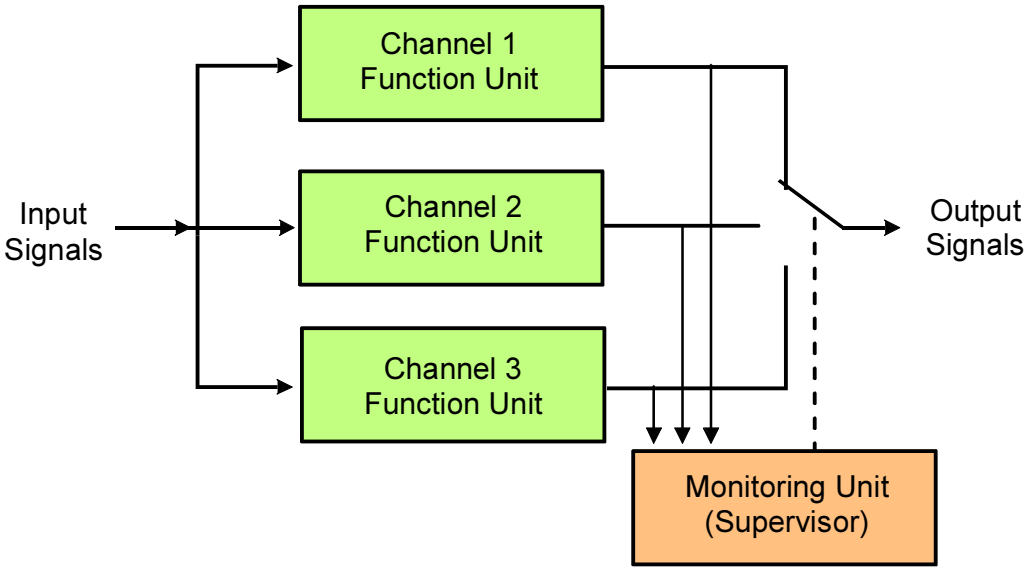
1 out of 2 system (1oo2): HFT=1
2 functional units, 1 required for operation
(DMR Dual Modular Redundancy)

Independent monitoring unit:

- Detect failures
- Act in case of failures, e.g. shut down (**fail safe**), reduce functionality (**graceful degradation**, **limp home**) or switch to redundant unit (**fault tolerant – fail operational**)

Redundant channels operation:

- **Cold stand-by**
  - Redundant channel 2 normally is off, will be turned on only in case of failure of channel 1
- **Hot stand-by**
  - Redundant channel 2 is permanently operating in parallel to channel 2
  - Improved failure detection, because channel 1 and channel 2 outputs can be compared. However: Comparison alone does not allow to find out, which channel failed



In hot stand-by operation, **monitoring by majority voting** possible:

- Channel outputs are compared
- If one channel's output differs from the two others, this channel is considered to be faulty

1 out of 3 system (1oo3):
3 functional units, 1 required for operation
(TMR Triple Modular Redundancy)

HFT = 2 (Hardware Failure Tolerance)

2 out of 3 system (2oo3):
3 functional units, 2 required for operation

HFT = 1

(Hardware) Failure Tolerance HFT: Number of (hardware) failures a system can tolerate

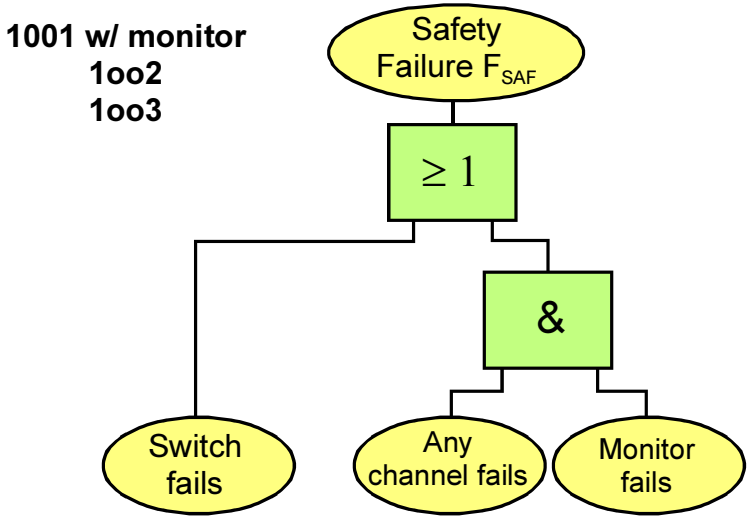
Analysis of these structures: see next page

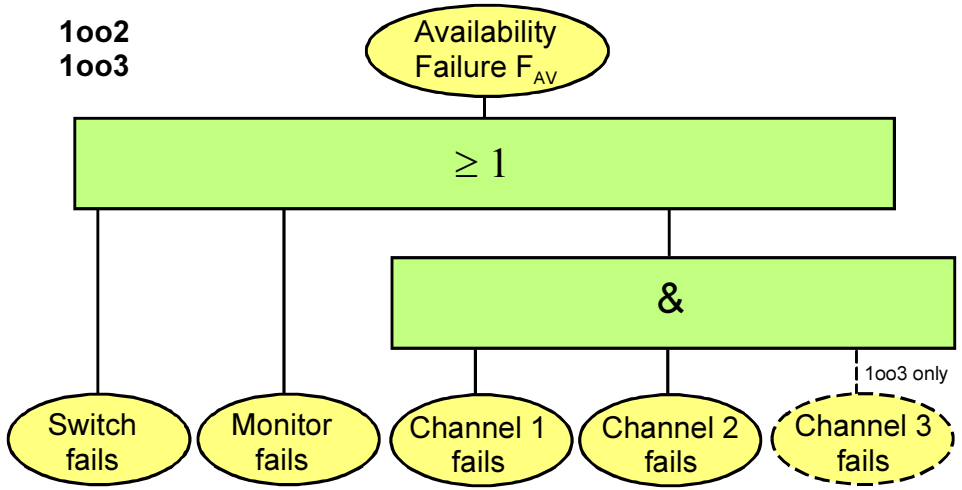
- Switch causes a single point failure
- Monitoring does improve safety
- Redundancy does improve availability, increases cost, reduces reliability

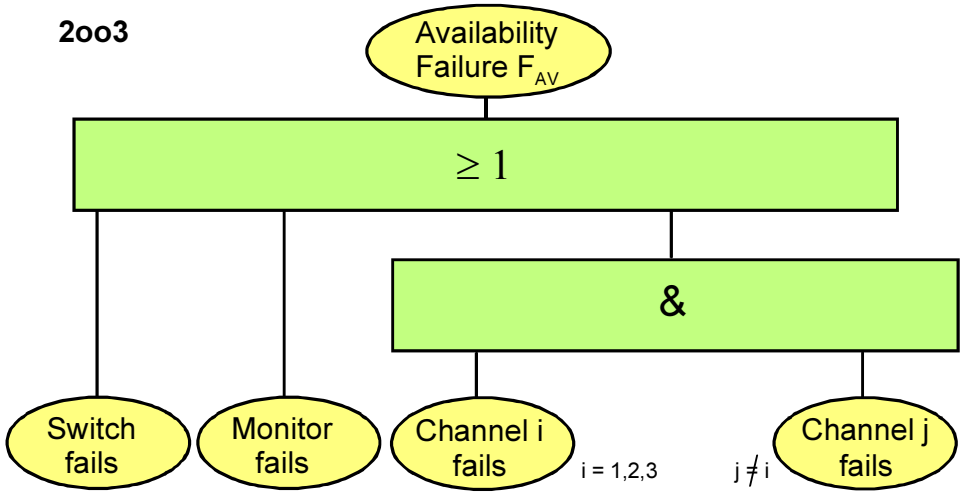
Failure Probabilities of Redundant Structures

- Assumptions: Identical channels, no common mode failures, $F_{monitor}$, $F_{switch} << F_{channel} << 1$
- F_{REL} Probability, that some failure occurs, no matter how severe (Parts count)
- F_{AV} Probability, that the system cannot be used, i.e. is not available
- F_{SAF} Probability of an undetected safety critical failure (assume: system off is safe)

1oo1 without monitoring and switch	$F_{REL} = F_{AV} = F_{SAF} = F_{channel}$
1oo1 with monitoring and switch	$F_{REL} \approx F_{AV} = F_{channel} + F_{monitor} + F_{switch}$ $F_{SAF} \approx F_{channel} \cdot F_{monitor} + F_{switch}$
1oo2	$F_{REL} \approx 2\ F_{channel} + F_{monitor} + F_{switch}$ $F_{AV} \approx F_{channel}^2 + F_{monitor} + F_{switch}$ … no common cause failures $F_{SAF} \approx F_{channel} \cdot F_{monitor} + F_{switch}$
1oo3	$F_{REL} \approx 3\ F_{channel} + F_{monitor} + F_{switch}$ $F_{AV} \approx F_{channel}^3 + F_{monitor} + F_{switch}$ $F_{SAF} \approx F_{channel} \cdot F_{monitor} + F_{switch}$
2oo3	$F_{REL} \approx 3\ F_{channel} + F_{monitor} + F_{switch}$ $F_{AV} \approx F_{channel}^2 + F_{monitor} + F_{switch}$ (same as 1oo2) $F_{SAF} \approx F_{channel} \cdot F_{monitor} + F_{switch}$


1001 w/ monitor
1oo2
1oo3
Safety Failure F_{SAF}
≥ 1
&
Switch fails
Any channel fails
Monitor fails

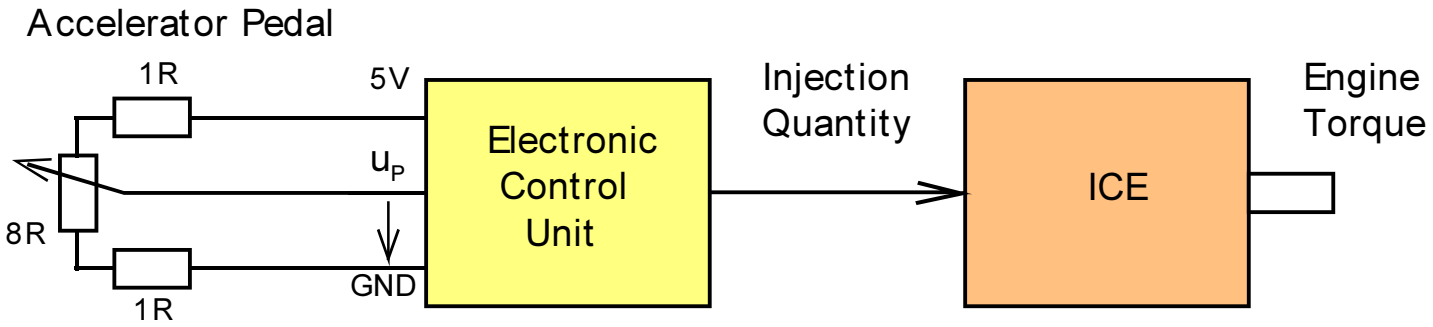

1oo2
1oo3
Availability Failure F_{AV}
≥ 1
&
1oo3 only
Switch fails
Monitor fails
Channel 1 fails
Channel 2 fails
Channel 3 fails


2oo3
Availability Failure F_{AV}
≥ 1
&
Switch fails
Monitor fails
Channel i fails
i = 1,2,3
j ≠ i
Channel j fails

Monitoring Methods

- **Signal range check** SRC: Check if a signal is within its physical limits (normal operating range) ← feasible for all signals with know signal range

  Example: Engine control system



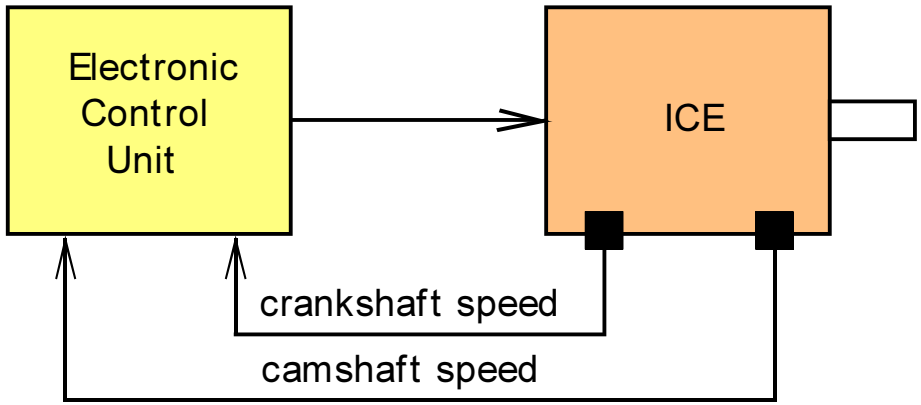
Normal operation: 0V < **0.5V** ≤ u_p ≤ **4.5V** < 5V

min idling max. torque max

|← normal operating range →|

Possible Failure	u_p	Comment
5V wire broken	0	<0.5V failure can be detected
u_p wire broken	0	<0.5V failure can be detected
GND wire broken	5V	>4.5V failure can be detected
. . .	. . .	. . .

- **Static plausibility check**: Compare a signal with another signal, which has the same or a similar information content, e.g.



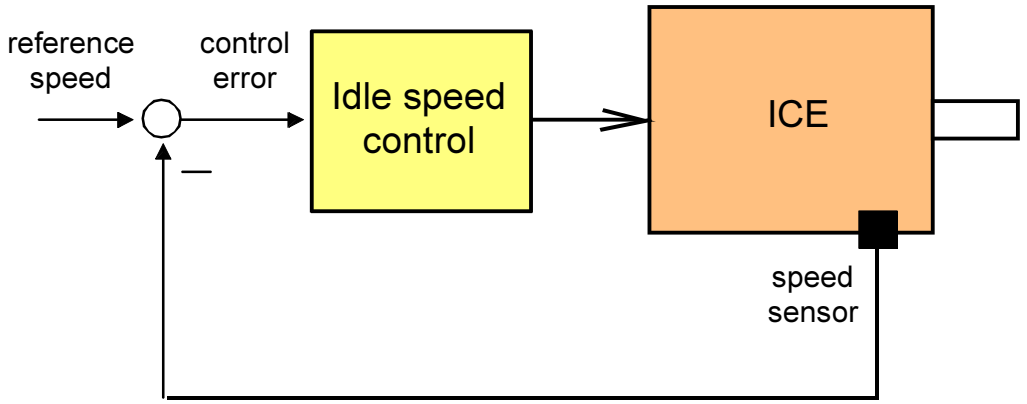
In a 4 stroke engine the camshaft speed is ½ of the (average) crankshaft speed.

If |camshaft speed – crankshaft speed/2| > 10% → failure of one of the speed sensors

- **Dynamic plausibility check**: Compare a signal's rate of change (slew rate) with the physical limits of its slew rate, e.g.
  - Typically the water temperature is monitored with 1 sample/sec
  - Typically the engine heats up or cools down with < 10°C / min
  - If the measured water temperature changes faster → sensor or wire failure

- **Short circuit and broken wire detection** for ECU output drivers and signal inputs

- **Steady state error check**: Check if steady state control error in closed loop systems is within tolerance, e.g.



- **Event sequence check**: Check the sequence of events and/or operator actions, e.g. to start a car with automatic transmission, the driver must
  - put the transmission in park position
  - start engine by turning the ignition key
  - put foot on the brake pedal
  - engage gear into D(rive) position

  . . .

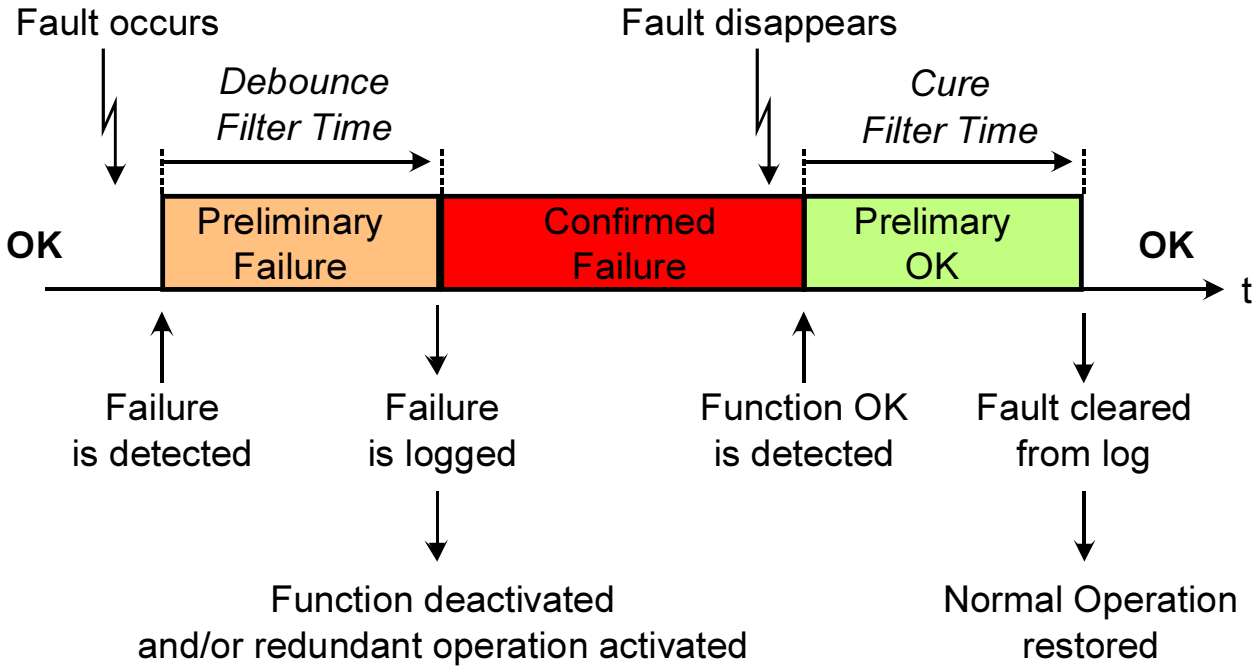
- **Runtime check**, e.g. watchdog
  - The monitored item, e.g. a microprocessor or a human operator, must periodically trigger the monitoring system ("watchdog").
  - If it fails to do so, the "watchdog" switches the system into a safe state and/or re-sets/restarts the monitored device.

- **Message timeouts and information redundancy** in data communication, e.g.
  - Parity and CRC checksums for digital data
  - Timeout monitoring for bus messages

  . . .

Typically, failures are detected and localized by a combination of several monitoring methods.

Because wrong detection is possible, debounce time filters and error counters are used:



Filter times for Function activation/deactivation and Error logging may be different.

Monitoring methods can be active

- during system start (power-on self-test)
- periodically (cyclic test, background test)
- continuously during normal operation
- on demand (test mode)

Quantitative description of monitoring quality:

Diagnostic Coverage for dangerous (safety critical) failures $DC = \frac{\lambda_{DD}}{\lambda_D} = \frac{\lambda_{DD}}{\lambda_{DD} + \lambda_{DU}}$

Diversity

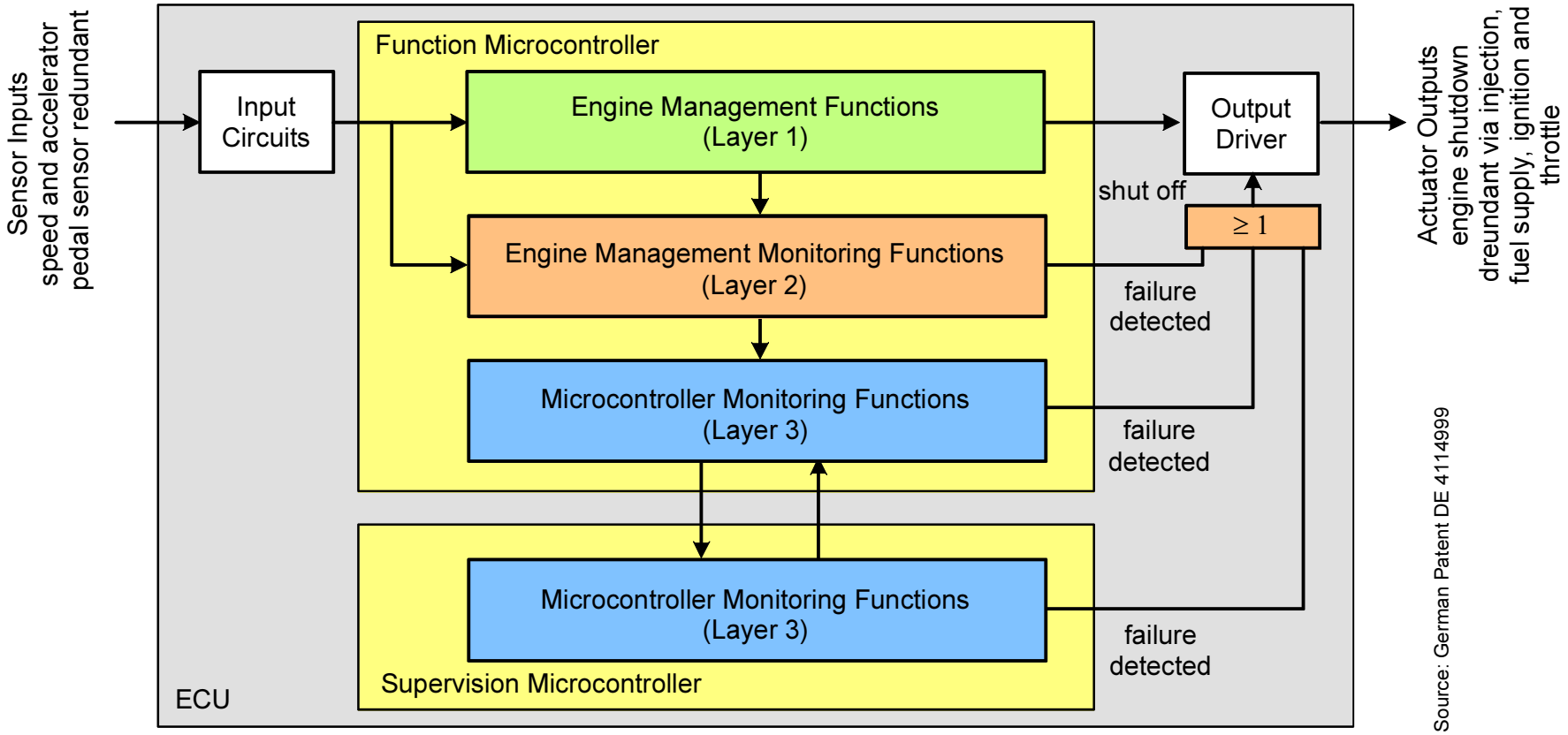
- Common cause failures can be avoided/reduced, if the redundant channels are built with
  - different technologies (e.g. electronic control with mechanical backup system)
  - different hardware and/or software and are
  - developed with different development tools and by different people.

e.g. elevators use an electrical drive but have a mechanical emergency brake

US Space Shuttle uses 4 different computer systems in main flight control system

- Disadvantage:
  very expensive: development effort for multiple systems, reduced economy of scale
- Remaining risk:
  Even diverse systems will be developed according to a common subset of requirement specifications

Redundant structure of engine management ECUs (Standardized EGAS safety concept [S15])



- Supervision of engine management function via function software monitoring (signal range check, static and dynamic plausibility, timeouts, …)
- Basic self monitoring of microcontroller operation via software (watchdog, RAM check, …)
- Diverse, redundant monitoring of microcontroller hardware via supervisor microcontroller

4.6 Analyzing System Safety 2: Failure Modes and Effects Analysis FMEA

FTA is a **top-down** ("deductive") approach, which

- is of limited value, when failure probabilities are not know
- can't guarantee, that all component failures are investigated
- does not systematically look for failure detection and avoidance methods

FMEA is a **bottom-up** ("inductive") approach, which

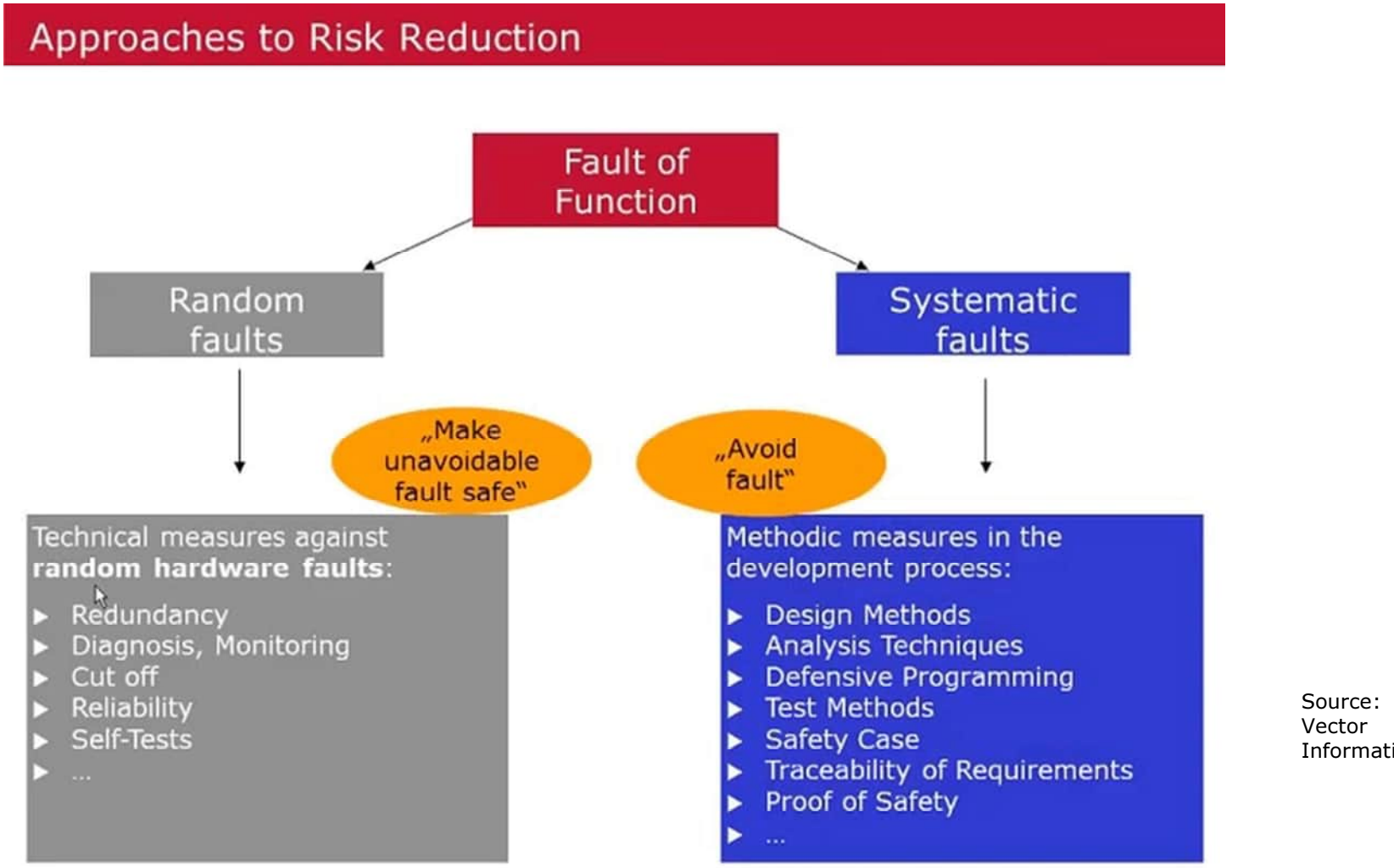
- systematically **tracks all component failures** up to their system level effect
- inherently **requires to develop failure detection** and avoidance methods
- bridges the gap between quantitative analysis and qualitative engineering know how
- **assesses failure risk** as a combination of probability and criticality on a relative scale based on engineering judgement even if quantitative data is not yet available
- The FMEA process **enforces team discussions** and leads to improved solutions.

Major drawback

- For a complex system FMEA is a lengthy process or one must restrict the level of details

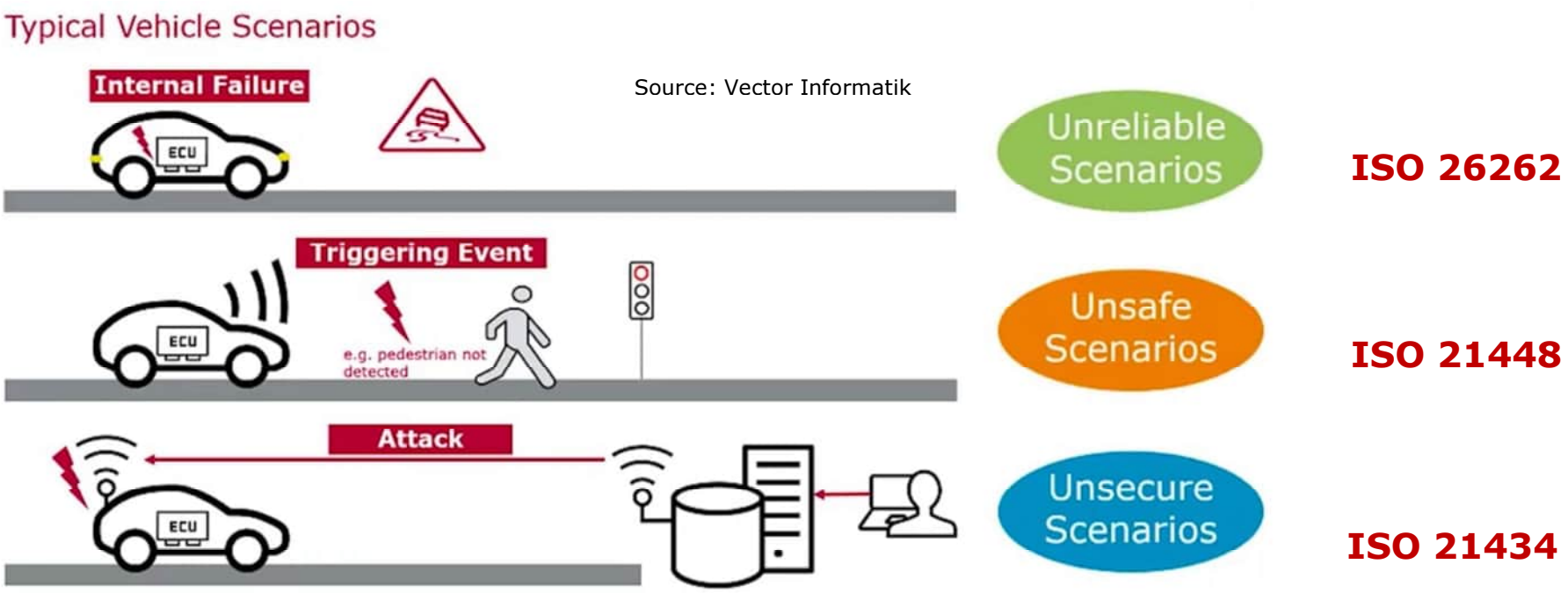
→ See the appendix for more info about FMEAs

4.7 Safety Integrity Levels and Functional Safety – Standardization Framework



Source: Vector Informatik

How much safety (and security) do we need and how do we achieve it?



- ISO 26262 **Functional Safety** (ISO 26262 is the automotive version of the generic IEC 61508)
  → Technical measures for the product and recommended development methods to **make sure, that a system complies with its safety requirements**
- ISO 21448 Safety of the Intended Functionality (SOTIF) – **new, work in progress**
  → Proposes measures to develop a complete safety requirements **including foreseeable misuse of the system** (!) by the user, **extend ISO 26262 to automated driving**.
- ISO/SAE 21434 **Cyber-Security** Engineering (based on SAE J3061) – **new, work in progress**
  → Proposes measures to deal with external attacks, i.e. **network-based threats**

How much safety do we need? → Acceptable risk

- Technical risks should be as low as reasonably possible (**ALARP**). But: Insurance companies do calculate with "cost per life", typical 1 … 10 Mio $ per life (source: US Department of Transport report DOT HS 809 835, 2004). Shall we really use cost to assess, which safety is worthwhile ???
- A new technical system (e.g. electrical brakes) must **not** have a **higher risk than** an **existing solution** (e.g. mechanical/hydraulical brakes).
- Technical systems should **not increase the human risk of life**, i.e. < 10^{-5} /a for general purpose technical systems. More risk may be accepted by individuals, who are free to decide, whether to expose to the risk or not, e.g. gliding with $2 \cdot 10^{-3}$ /a.

Risk	Percentage of death persons per year = accepted (?) failure rate
Total mid of lifetime death risk (10…15 year old men)	10^{-3} / a
- death by all types of accidents	$0.5 \cdot 10^{-3}$ / a
- accidents at home	$0.4 \cdot 10^{-3}$ / a
- traffic accidents	$0.06 \cdot 10^{-3}$ / a
- natural disasters	$0.002 \cdot 10^{-3}$ / a

Source: David J. Smith & Kenneth G. L. Simpson, Functional Safety, 2nd Edition, Elsevier, 2004

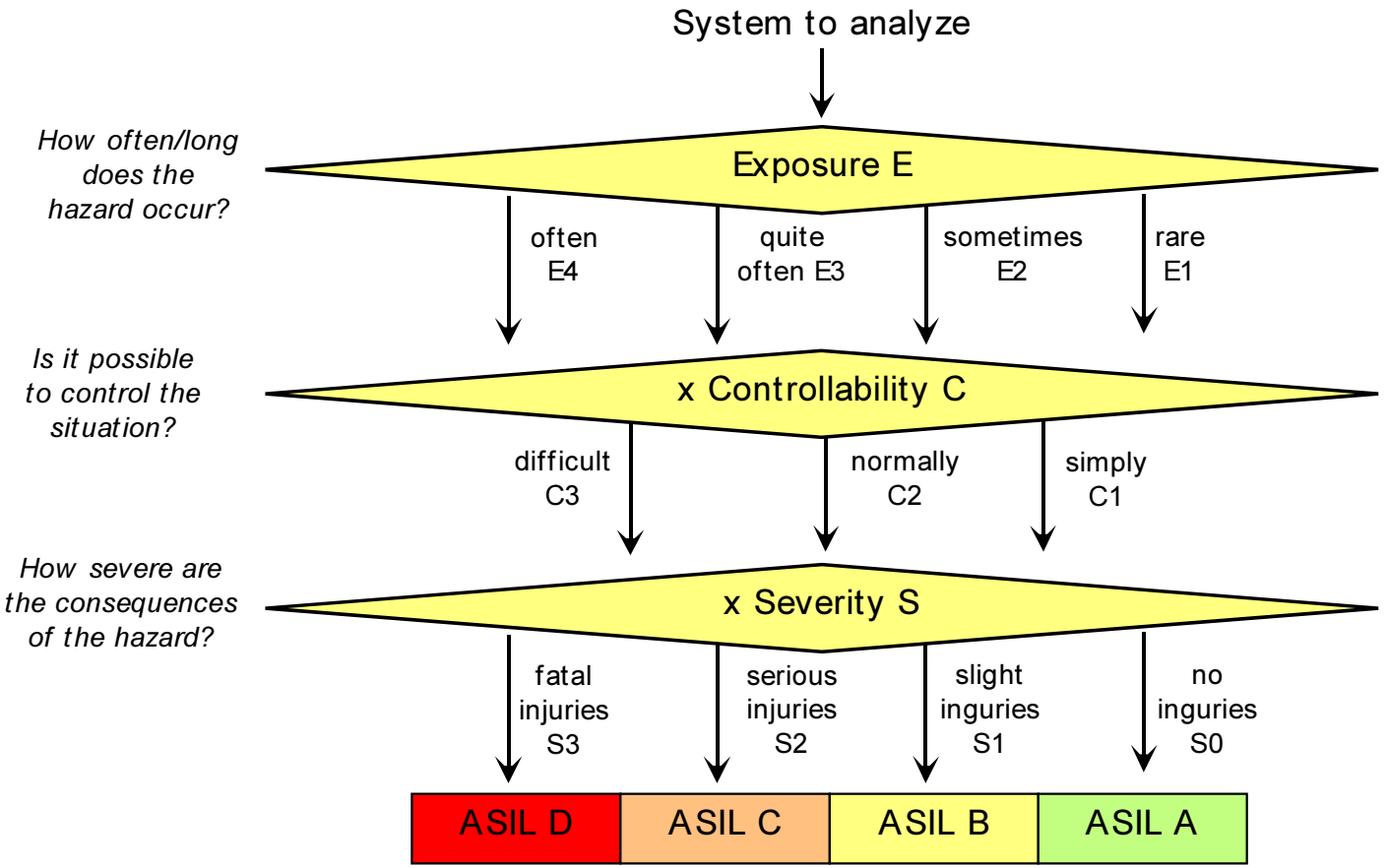
- **ISO 26262** *Road Vehicles – Functional Safety* (www.iso.org) defines four **Automotive Safety Integrity Levels ASIL** D to A depending on the riskiness of a system.

  Note: ASILs are similar, but not identical to the SIL levels of the generic IEC 61508 *Functional safety of electrical/electronic/programmable safety-related systems* (www.iec.ch)

  For each ASIL failure probability limits and required development methods are specified.

ISO 26262 (IEC 61508) Risk Graph: Mapping risk to ASIL (SIL)

- ISO 26262, the automotive version of IEC 61508, defines the required **Automotive Safety Integrity Level ASIL** via the **Severity S**, **Exposure E** and **Controllability C** of failures



IEC 61508 and ISO 26262 are not completely compliant with respect to ASIL vs. SIL and some other items.

ISO 26262 Exposure Time Classification:

•Class	E1	E2	E3	E4
Description	Rare events	Sometimes	Quite often	Often
•Informative examples	Accident situation that requires release of the airbag Stop at railway crossing, which requires start of engine Towing, jump start.	Pulling a trailer, driving with roof rack Driving on a mountain pass with unsecured steep slope Driving situation with deviation from desired path	Fuelling, passing, tunnels, hill hold, car wash Night driving on roads without streetlights, wet roads, snow and ice, congestion	Starting, shifting gears, accelerating, braking, steering, using indicators, parking, driving backwards Driving on highways, driving on secondary roads, city driving
Value E	**0.001**	**0.01**	**0.1**	**1**

Source: ISO 26262

ISO 26262 Controllability Classification:

Class	C1	C2	C3
Description	Simply controllable	Normally controllable	Difficult to control or uncontrollable
Definition	Less than 1% of average drivers or other traffic participants are usually unable to control the damage.	Less than 10% of average drivers or other traffic participants are usually unable to control the damage.	The average driver or other traffic participant is usually unable, or barely able, to control the damage.
Value C	**0.01**	**0.1**	**1**
Informative examples	When starting the vehicle with blocked steering column, the car can be brought to stop by almost all drivers early enough to avoid harm to persons nearby. Faulty adjustment of seats while driving can be controlled by almost all drivers through adjustment of seats and bringing the vehicle to a stop.	Driver can normally avoid departing from the lane: • in case of a failure of ABS during emergency braking. • on snow or ice in a curve in case of a failure of ABS during emergency braking. • in case of a motor failure at high lateral acceleration (motorway exit). Driver is normally able to bring the vehicle to a stop in case of a total light failure at medium or high speed on an unlighted country road without departing from the lane in an uncontrolled manner.	Self-steering with high angular speed at medium or high vehicle speed can hardly be controlled by the driver. Driver cannot bring the vehicle to a stop if a total loss of braking performance occurs. In case of faulty airbag release at high or moderate vehicle speed, driver usually cannot prevent vehicle from departing from the lane.

Quelle: ISO/WD 26262-3

ISO 26262 Severity and ASIL level assignment:

Source: ISO 26262

		E (exposure time) * C (controllability)					
		1	0.1	0.01	0.001	0.0001	0.00001
Severity	S0 - no injuries	QM	QM	QM	QM	QM	QM
	S1 - slight and moderate injuries	ASIL B	ASIL A	QM	QM	QM	QM
	S2 - serious, including life-threatening, injuries, survival probable	ASIL C	ASIL B	ASIL A	QM	QM	QM
	S3 - life-threatening injuries (survival uncertain) or fatal injuries	ASIL D	ASIL C	ASIL B	ASIL A	QM	QM

QM … Non-safety related systems, only normal quality management required. E and C classification see above

Example: Steering system of a car

Severity: S3 Exposure time: E4 → E = 1 Controllability: C3 → C = 1

$\Rightarrow$ E · C = 1 + S3 $\Rightarrow$ ASIL D



Possible Classification (no industry standard!!!)

Source: Freescale

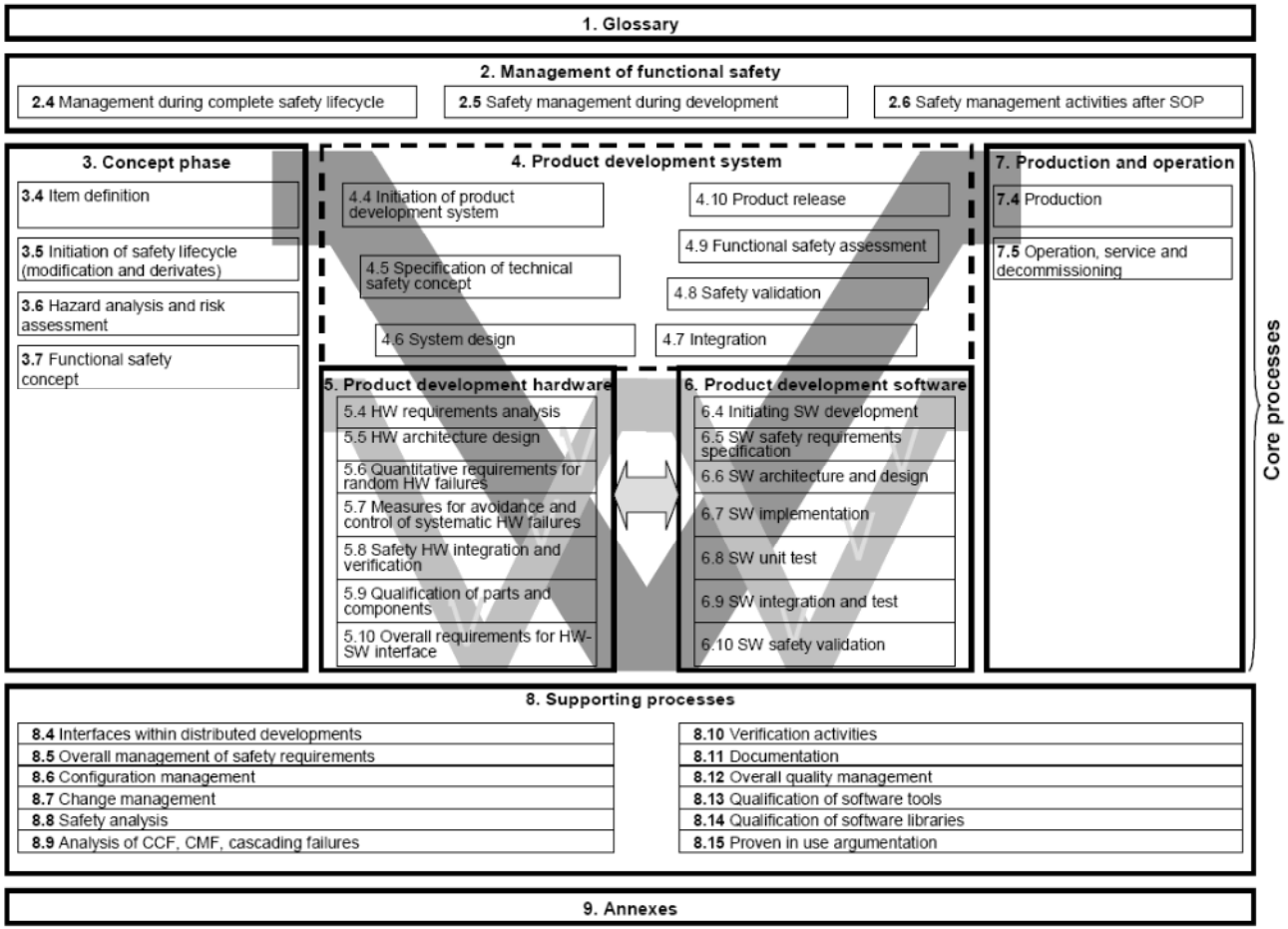
ISO 26262 Required Reliability depends on ASIL (IEC 61508 SIL)

ISO 26262 Level (IEC 61508 Level)	Acceptable failure rate λ_D for dangerous failures (PFH Failures per Hour)	Acceptable failure probability F_D for dangerous failures (PFD Failure on Demand)
ASIL D (SIL 4) Highest level	$< 10^{-8}$ / h	$< 10^{-4}$
ASIL C (SIL 3)	$< 10^{-7}$ / h	$< 10^{-3}$
ASIL B (SIL 2)	$< 10^{-7}$ / h ($< 10^{-6}$ / h)	$< 10^{-2}$
ASIL A (SIL 1) Lowest level	None ($< 10^{-5}$ / h)	$< 10^{-1}$
Applies to	High demand rate systems, i.e. systems, which are frequently used, e.g. brakes of a car, so that faults will show up immediately as failures.	Low demand rate systems, i.e. systems, which are rarely used, e.g. airbags, so that faults can be dormant. In case that the system has a built in self test, the test period T is used when calculating F.

Required Safe Failure Fraction, i.e. failures, which must not lead to a safety critical situation

	ASIL A	ASIL B	ASIL C	ASIL D
Single-Point Fault Metric SPFM	No requirements	≥ 90%	≥ 97%	≥ 99%
Latent Fault Metric LFM dormant failure which will lead to a safety-critical situation in combination with a second failure	No requirements	≥ 60%	≥ 80%	≥ 90%

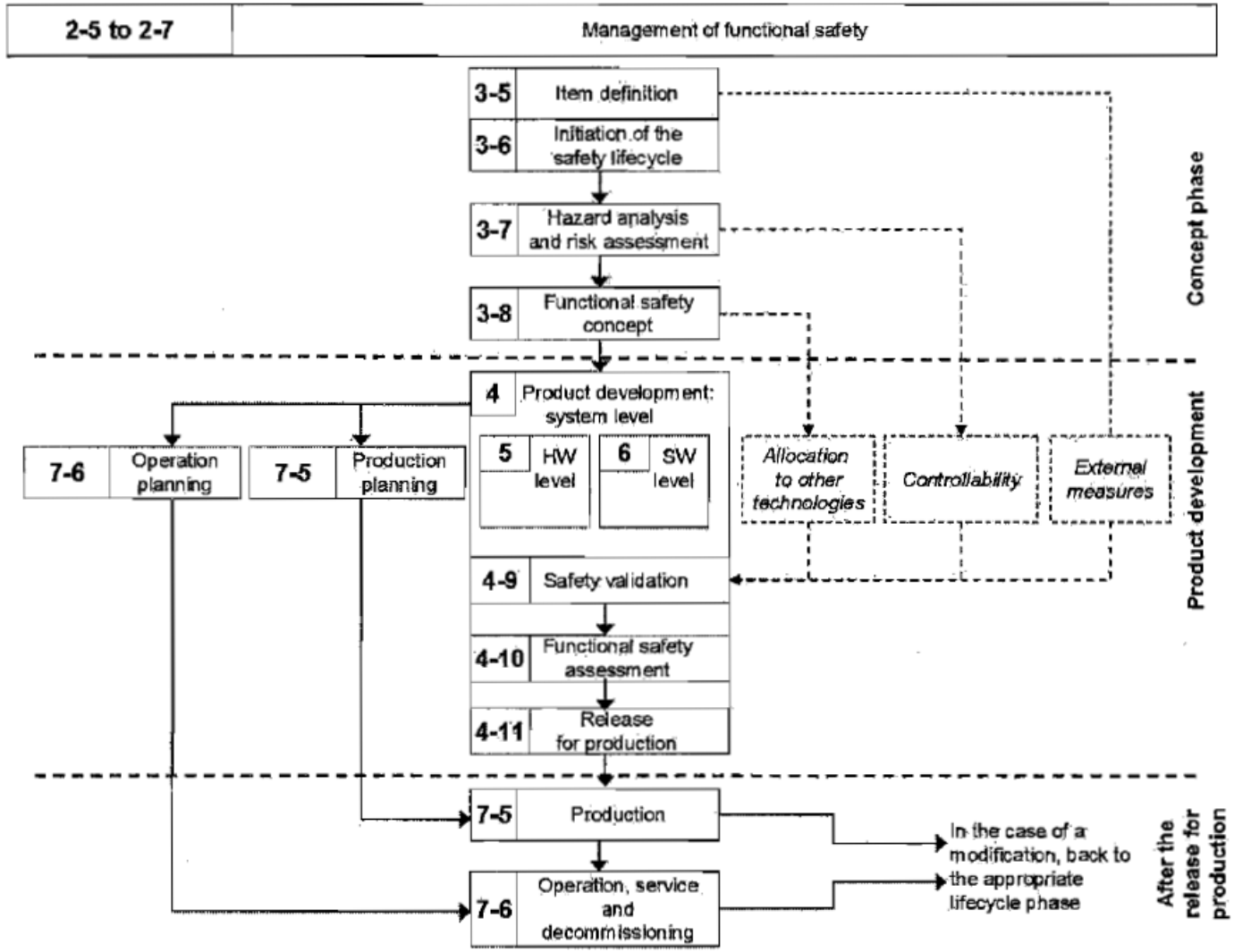
IEC 61508 and ISO 26262 define **requirements for the complete life cycle** of a product, e.g. a development process according to the V-model is required:



Source: ISO 26262

In many points IEC 61508 uses a very generic approach and often gives recommendations, what to do, but not how to do it. Thus many additional or competing industry and/or company standards do exist, which go much more into detail, but are not fully compatible.

ISO 26262 Safety Lifecycle



Source: ISO 26262

Major Steps:

- Hazard Analysis and Risk Assessment

  Which hazards may occur and how risky are they?

- Functional Safety Concept

  Design appropriate failure detection and handling functions for risky hazards?

- Safety Validation

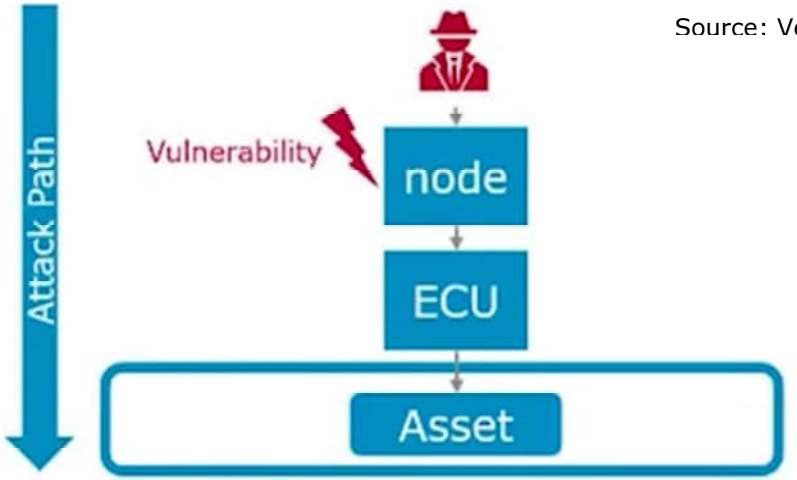
  Validate by theoretical proofs and practical tests that the safety functions do manage risky hazards correctly.

- Functional Safety Assessment

  Formally assess your safety concept using FMEA, FTA etc.

4.8 Cybersecurity

Source: Vector Informatik



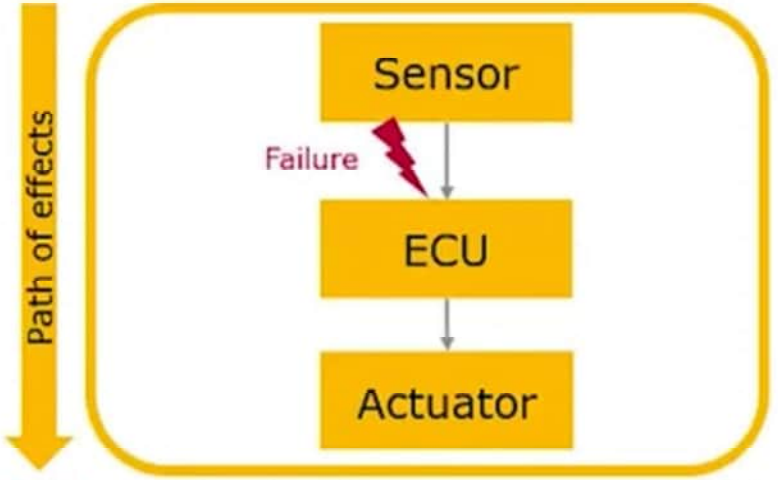
Vulnerability: Weakness (**unknown**), which can be exploited by an **attacker**.

Security Control: **Reduces probability** of unauthorized access (**known**) by the **attacker**.

... Vulnerability detected during **testing**.

... a **how-to** to create the exploit is published.

... exploit can be **downloaded** from a black market.

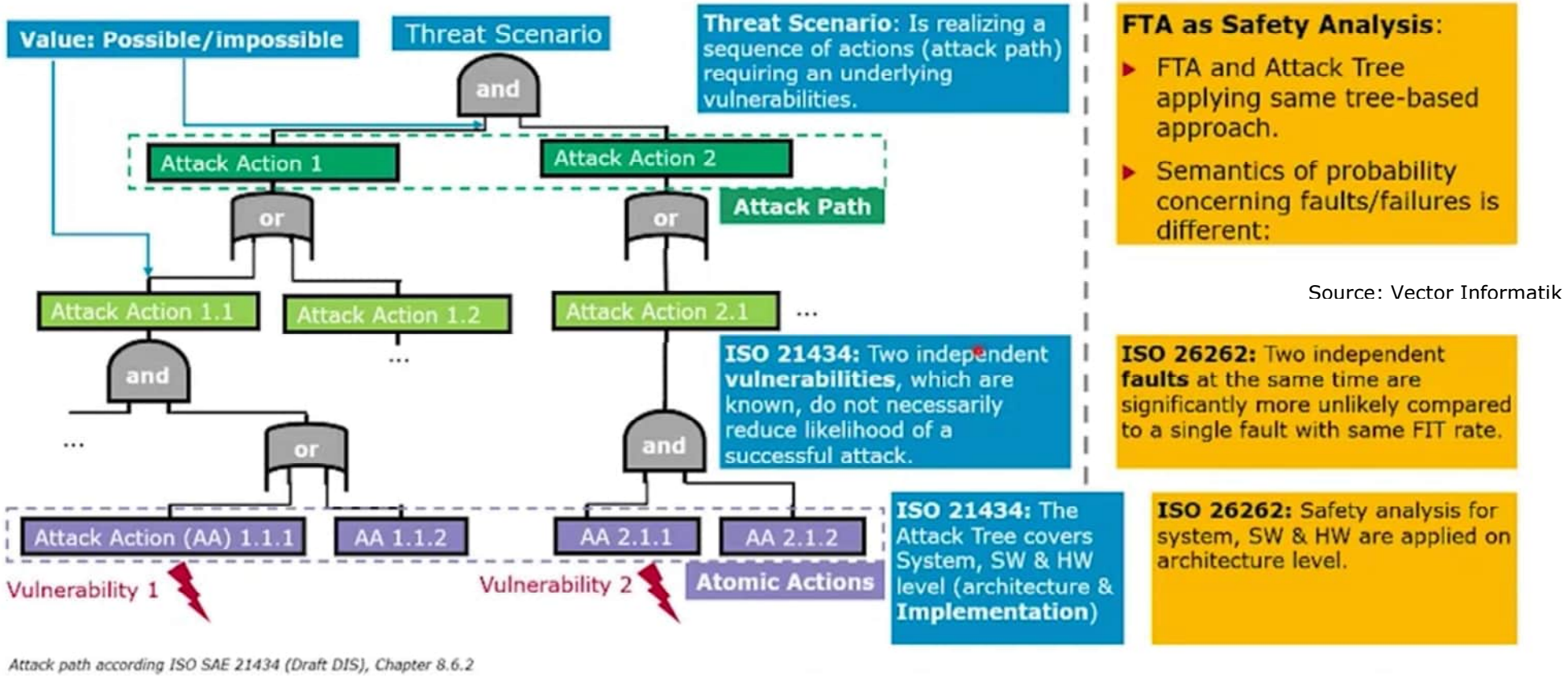


Failure: Random/systematic fault which leads with a certain **probability** to a violation of a **Safety Goal**.

Safety Mechanism: **Reduces probability** of safety goal violation.

Vulnerability (Cybersecurity) versus **Failure (Safety)**

ISO 21434 Cybersecurity analysis similar to functional safety analysis



Attack path according ISO SAE 21434 (Draft DIS), Chapter 8.6.2

Attack Tree versus **Fault Tree**

Further Reading

- Books and Papers

[4.1] A. Birolini: Reliability Engineering. Springer, 8th Edition, 2017

[4.2] J. Börcsök: Electronic Safety Systems. Laxmi, 2009

[4.3] J.W. Evans, J.Y. Evans: Product Integrity and Reliability in Design. Springer, 2012

[4.4] W. Denson, Mary Priore: Automotive Electronic Reliability Prediction, SAE Paper 870050, www.sae.org

[4.5] D. Hermann: Software Safety and Reliability. IEEE Computer Society, 3rd Edition, 2000

[4.6] N. Leveson: Safeware, System Safety and Computers. Addison-Wesley, 1995

[4.7] M.R. Lyu (editor): Handbook of Software Reliability Engineering. McGraw-Hill, 1998

[4.8] C. Jones: Software estimating rules of thumb, IEEE Computer 3/1996, page 116

[4.9] L. Rosenberg, T. Hammer: Software Metrics and Reliability. 1998, satc.gsfc.nasa.gov

[4.10] W. Vesely: NASA Fault Tree Handbook with Aerospace Applications, 2002, http://www.hq.nasa.gov

[4.11] D. Smith, K. Simpson: Safety Critical Systems Handbook. Elsevier, 4rd Edition, 2016

[4.12] J. Börcsök: Functional Safety. Hüthig, 1st Edition, 2007

[4.13] Exida: IEC 61508 Overview Report. 2006. www.exida.com

[4.14] D. Smith: Reliability, Maintainability and Risk. Elsevier, 8th edition, 2011

- Standards and Guidelines

[S1] MIL Handbook (HDBK) 217F Reliability Prediction of Electronic Equipment, US Department of Defense

[S2] MIL Handbook 338B Electronic Reliability Design Handbook

Further Reading

[S3] IEC 61709 Electronic Component Reliability. www.iec.ch

[S4] Bellcore TR-332: Reliability Prediction for Electronic Equipment

[S5] IEC 61709: Electronic Components Reliability. www.iec.ch

[S6] DO 178B Software Considerations in Airborne Systems and Equipment Certification. www.rtca.org

[S7] NASA Technical Standard 8719.13A: Software Safety. www.nasa.gov

[S8] MISRA Motor Industry Software Reliability Association, www.misra.org.uk:
Guidelines for the use of the C language in critical systems.
Guidelines for the use of the C++ language in critical systems.
Development guidelines for vehicle based software.

[S9] IEC 61508 Functional Safety of Electrical, Electronic and Programmable Safety-Related Systems. www.iec.ch/functionalsafety

[S10] ISO 26262 Road Vehicles – Functional Safety, www.iso.org

[S11] DIN V 19250: Grundlegende Sicherheitsbetrachtungen für MSR-Schutzeinrichtungen (withdrawn standard in favor of IEC 61508). www.din.de

[S12] SAE ARP 4754: Certification Considerations for Highly-Integrated Or Complex Aircraft Systems, www.sae.org

[S13] SAE ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment www.sae.org

[S14] VDA FAKRA (Society of German Automobile Manufacturers): Automotive SPICE Prozessassessment. 2007, www.vda-qmc.de

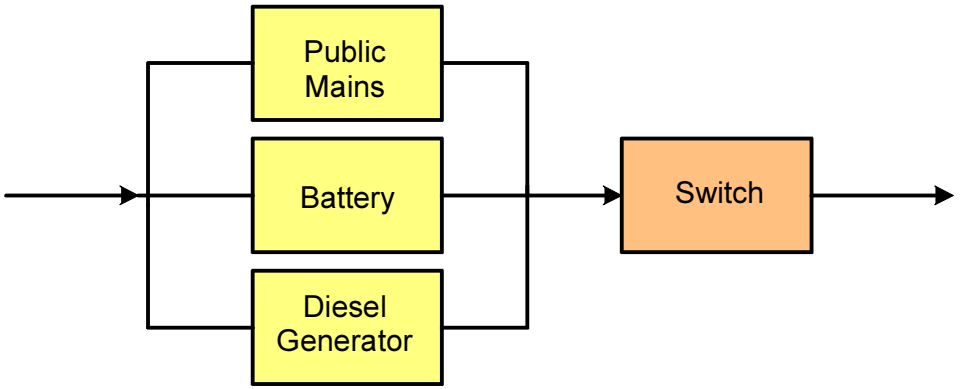
[S15] AK EGAS: Standardized EGAS Safety Concept for Gasoline and Diesel Engine Management Systems.

Appendix: Methods not discussed in detail and additional information

Reliability Block Diagrams

- Describes which blocks of a system are involved in providing a certain functionality. Required blocks are connected in series, alternative blocks are connected in parallel.

  E.g.: Reliability block diagram for the electrical power supply of a hospital



Note: In a reliability block diagram the same block can occur several times if the functional logic requires it.

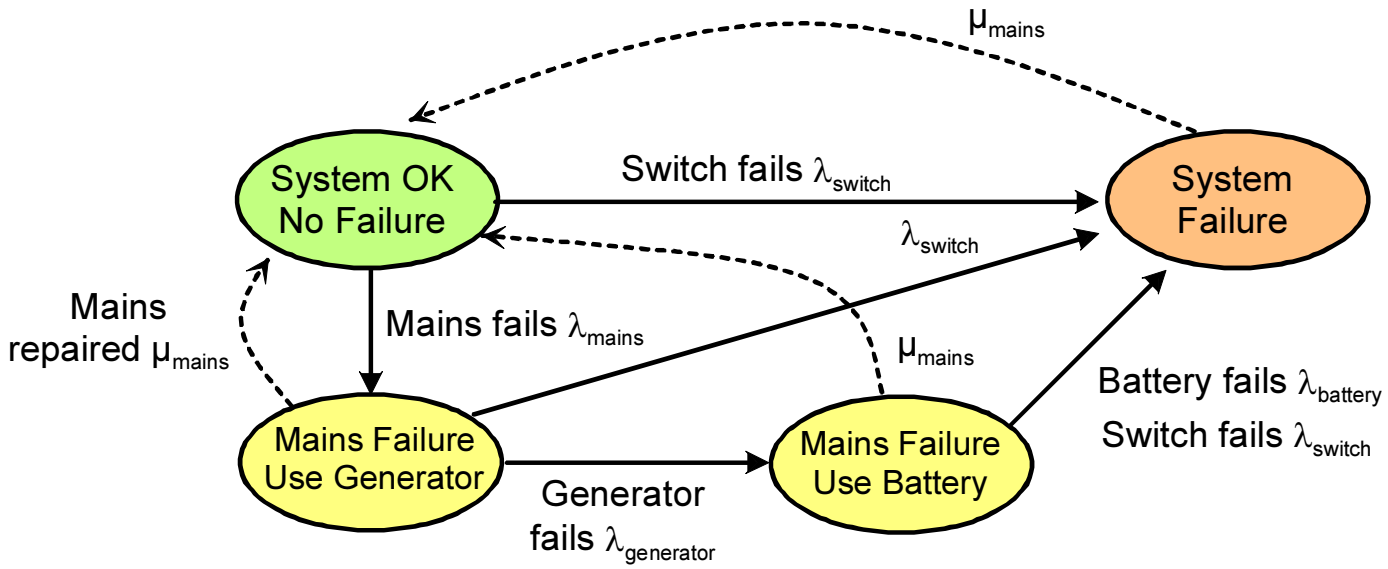
- Used as an alternative to fault trees FTA
- The same mathematics applies:

	Fault Tree	Reliability Block Diagram
System fails, if any of the blocks fails	OR gate	Series connection
System fails, if all of the blocks fail	AND gate	Parallel connection

Markov Models

- Similar to state charts, where working and failed components are described by states and failure events by state transitions. Failure rates λ describe the probability of the transitions.

  E.g.: Markov model for the electrical power supply of a hospital (incomplete)



- Can describe the dynamic behaviour of a system when failure events do occur in sequence, compared to FTA, which can only describe static behaviour.
- Allows to analyse systems, where all or some of the component failures are repaired, described by their respective repair rate μ (similar to failure rate λ)

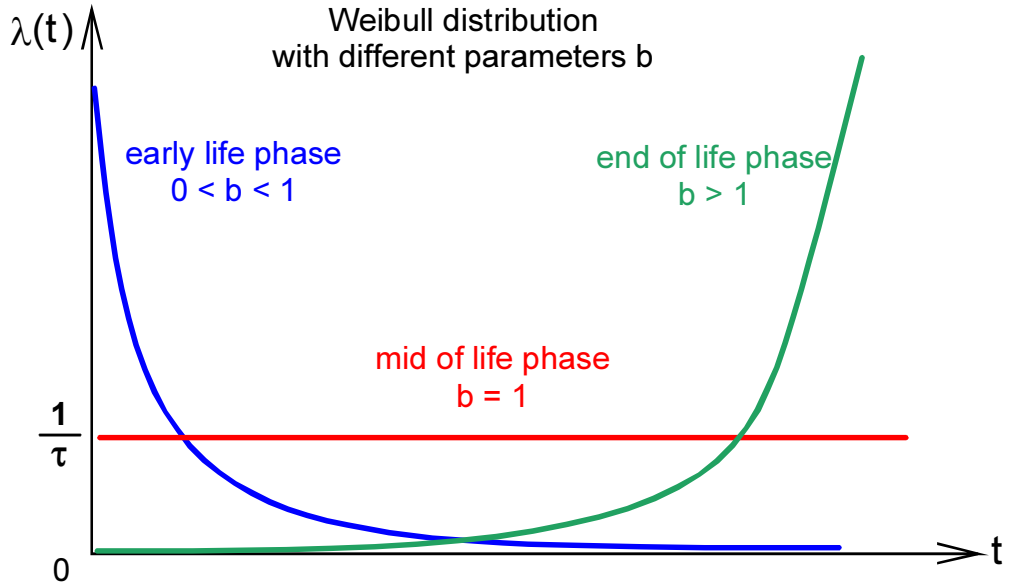
Weibull-Distribution
(used if λ not constant)

The 3 segments of the bathtub curve can be approximated by

$$\lambda(t) = \frac{b}{\tau} \cdot \left(\frac{t}{\tau}\right)^{b-1}$$

with parameters b and τ.

Note:
Each of the 3 segments of the bathtub curve has its own set of parameters b, τ.



For an arbitrary Weibull distribution:

$$R(t) = e^{-(t/\tau)^b}$$

$$MTTF = \tau \cdot \Gamma\left(\frac{b+1}{b}\right)$$

with the Gamma-Funktion

b	0.5	1	2	5
$\Gamma\left(\frac{b+1}{b}\right)$	2	1	0.89	0.92

For b=1 the Weibull distribution is equal to the exponential distribution with λ=const. described by eq. (8a-c).

MTTF for 1ooN redundant structures

$F = (F_{channel})^N \rightarrow R = 1 - (1 - R_{channel})^N = 1 - (1 - e^{-\lambda \cdot t})^N$

$$\rightarrow \quad MTTF = \int_0^\infty R(t)\,dt = \{ 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N} \} \cdot \frac{1}{\lambda}$$

E.g.:

N=1: $MTTF = \frac{1}{\lambda}$

N=2: $MTTF = \frac{3}{2} \cdot \frac{1}{\lambda}$

50% improvement
100% cost increase

N=3: $MTTF = \frac{11}{6} \cdot \frac{1}{\lambda}$

83% improvement
200% cost increase

Reliability Prediction for Software

- Software bugs have a systematic nature, i.e. they are included in each copy of the software. However, often they occur under very special, rare operating conditions only. Thus, they may also be described by statistical methods.
- **Development phase software reliability prediction** is difficult or impossible, as all software bugs are development errors. Development quality is highly dependent on application area, software complexity, developers' experience and available time. Not much freely available statistical data does exist. The following data was published by CMU [*1]:

Quality status	*Bugs per 1000 LOC*	*CMMI Level*
Untested software	250	-
Unusable software	> 10	-
Faulty software	< 10	0
Unstable software	< 6	1
Mature software	< 3	2, 3
Stable software	< 1	4, 5

LOC … programming language source code statement, empty lines and comments not counted, e.g. Windows XP has 45 Mio. LOC

A typical software test (Code review, inspection or test step does find 30% of all bugs, i.e. 70% of all bugs survive each test step).

CMMI … Capability Maturity Model Integration is a formal method to assess the quality of software development processes, proposed by [*1] Carnegie Mellon University, Pittsburg, USA. A similar assessment exists in Europe as ISO 15504 Automotive Software Process Improvement and Capability Determination (SPICE).

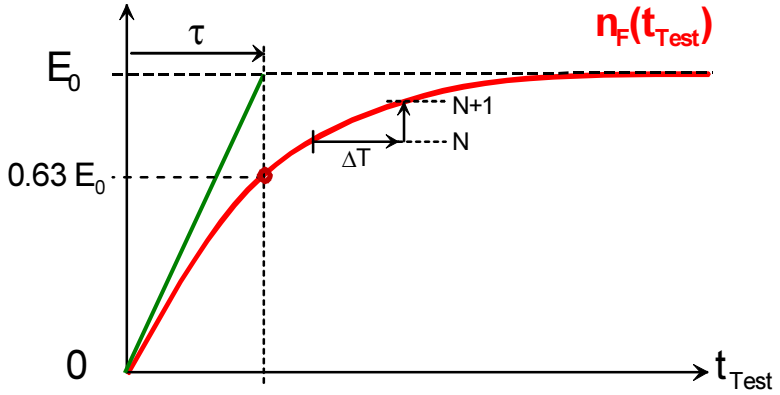
Musa's Software Reliability Model: Prediction based on software test statistics

… assumes, that

- testing is done under conditions similar to the later use of the software
- at test begin, the software has a finite, but unknown number of bugs E_0
- bugs are found and removed (without introducing new bugs) according to an exponential function

$$n_F(t_{Test}) = E_0 \left(1 - e^{-t_{Test}/\tau} \right)$$

where τ is an unknown parameter describing the intensity of the tests



- The more bugs were removed, the more time is necessary to find the next one, i.e. MTTF does increase exponentially with test time. After bug N has been found and fixed, it takes time ΔT to find the next bug N+1. The rate, at which bugs are found, is

$$\frac{N+1-N}{\Delta T} = \frac{1}{\Delta T} \approx \frac{dn_F}{dt_{Test}} = E_0 \cdot \frac{1}{\tau} \cdot e^{-t_{Test}/\tau}$$

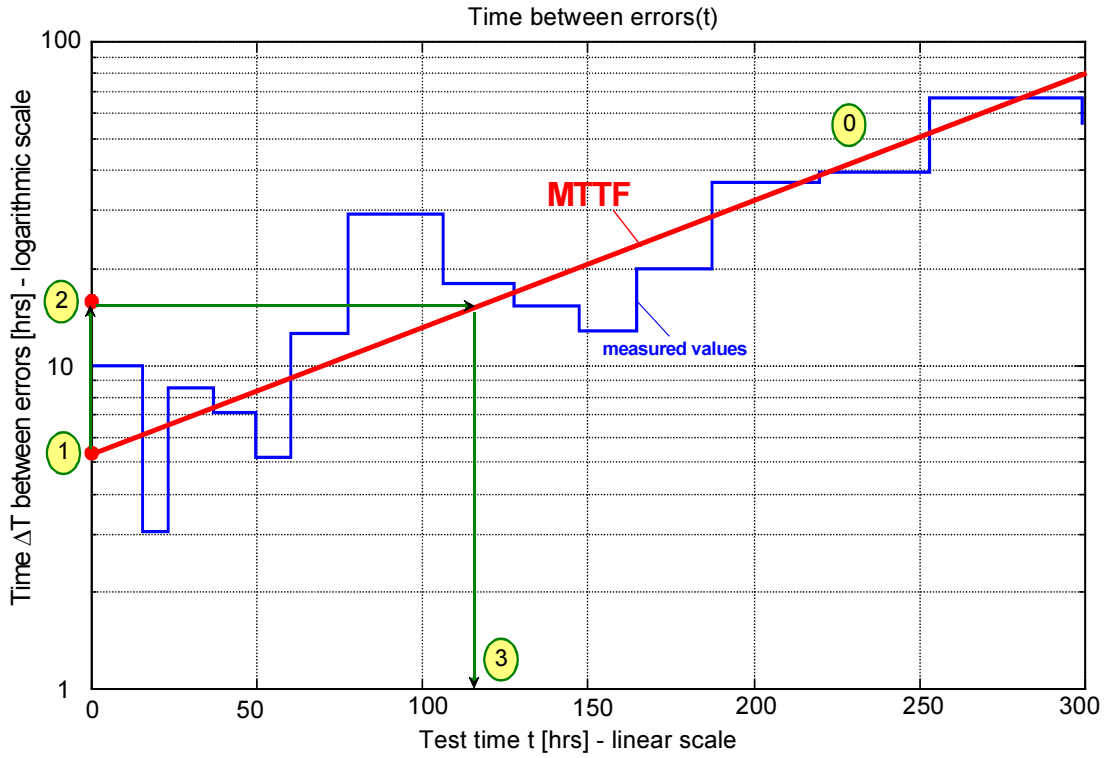
From this equation, we can calculate: $MTTF(t_{Test}) \approx \Delta T = \frac{\tau}{E_0} e^{t_{Test}/\tau}$

i.e. when testing and fixing bugs, the MTTF increases exponentially with test time.

- Once the software is delivered, the MTTF = $1/\lambda$ does not change anymore.

Practical application: When testing, the times $\Delta T \approx$ MTTF between bugs can be recorded and plotted. When using a diagram with linear axis, the plot should show a step approximation of the above exponential function $MTTF(t_{Test}) \approx \Delta T = \frac{\tau}{E_0} e^{t_{Test}/\tau} = MTTF(t_{Test}=0) \cdot e^{t_{Test}/\tau}$

By computer-aided curve fitting, the unknown parameters E_0 and τ can be found. For manual analysis, it is better to use a semi-logarithmic plot. In such a semi-logarithmic plot, an exponential function turns into a straight line, which simplifies curve-fitting drastically:



Step 0:

- Draw the straight line approximation for MTTF. Note: The areas above and below the red line should be balanced.

Step 1:

- Get the value $MTTF(t_{Test}=0) = \tau / E_0$ from the diagram.

Step 2:

- Calculate $MTTF(t_{Test}= \tau) = MTTF(t_{Test}=0) \cdot e$

Step 3:

- Get the value τ from the diagram.

Step 4:

- Calculate $E_0 = \frac{\tau}{MTTF(t_{Test}=0)}$

Failure Modes and Effects Analysis FMEA

FTA is a **top-down** approach, which

- is of limited value, when failure probabilities are not know
- can't guarantee, that all component failures are investigated
- does not systematically look for failure detection and avoidance methods

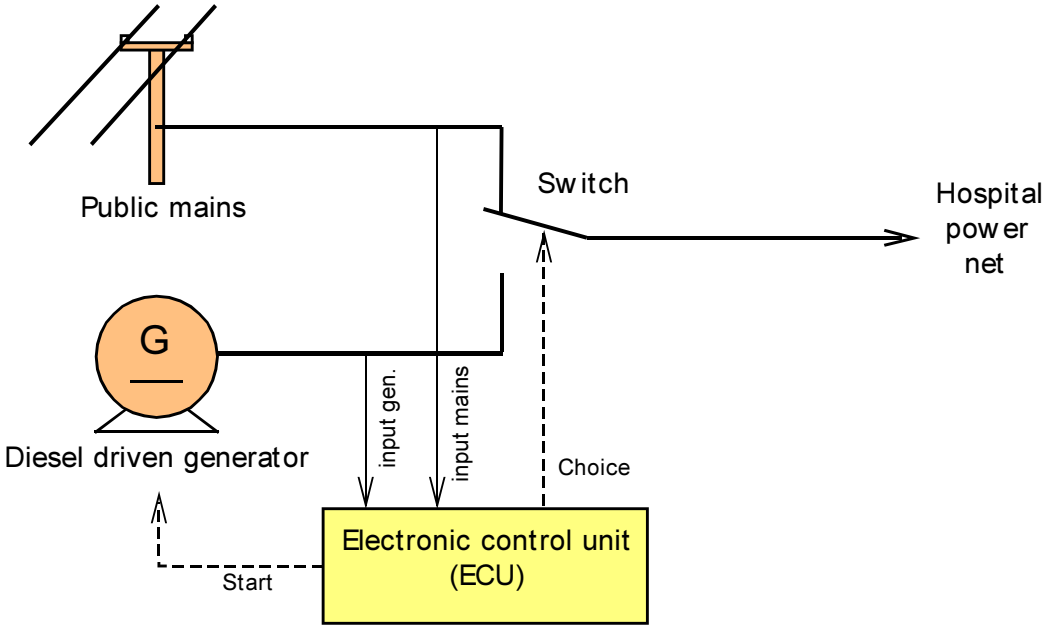
FMEA is a **bottom-up** approach, which

- systematically **tracks all component failures** up to their system level effect
- inherently **requires to develop failure detection** and avoidance methods
- bridges the gap between quantitative analysis and qualitative engineering know how
- **assesses failure risk** as a combination of probability and criticality on a relative scale based on engineering judgement even if quantitative data is not yet available
- The FMEA process **enforces team discussions** and leads to improved solutions.

FMEAs can be used on various levels:

- **System Design FMEA**: Used when defining the system architecture, investigates subsystem and/or component interaction, discusses interface failures, but treats subsystems and/or components as black box
- **Component Design FMEA**: Used when designing a component, tracks internal failures up to the component interface
- **Process FMEA**: Used to analyze manufacturing (and other) processes

FMEA example: Electrical power supply for a hospital



System function

- Normally the hospital net is powered by the mains.
- An ECU monitors the mains voltage. If the voltage is too high or too low, the ECU starts up a diesel driven generator and switches the hospital net to the generator.
- The generator's output voltage is also monitored. If this voltage is too high, the generator is stopped.

Step 1: Which **components** do we have?
Step 2: Which component failures can occur? (**Failure Modes**)
Step 3: What may **cause** these failures?
Step 4: What is the system level **effect** of these failures?
Step 5: How can failures be **detected** and which **countermeasures** can be taken?
Step 6: What is the **rest effect** of these failures, when detected & countermeasures taken?
Step 7: Assess **criticality C**, **probability P** and **detectability D** of these failures on a scale from 1 … 10 and compute the **risk priority R = C · P · D**
Step 8: **Address** all **failures with high risk prority R**

#	Component [1]	Failure [2]	Possible cause [3]	Effect when not detected [4]	Detection / Countermeasure [5]	Effect when detected [6]	Risk priority [7]			
							C	P	D	R
1.1	Mains	voltage too high	bad control quality by mains operator	destroys electrical equipment connected	by voltage measurement in ECU → switch to generator	redundant power supply by generator	3	6	3	54
1.2		voltage too low	same as 1.1 broken wire	power loss in hospital net	same as 1.1	same as 1.1	3	10	3	90
2.1	Generator	voltage too high	failure in generator voltage controller	same as 1.1 (but only when generator is needed)	by voltage measurement in ECU → switch off generator, call service personnel	power loss in hospital net	10	5	3	150
2.2		voltage too low	same as 2.2	same as 1.2 (but only when generator is needed)	by voltage measurement in ECU → call service personnel	malfunction of electrical equipment, in worst case: power loss	9	5	3	135

#	Component [1]	Failure [2]	Possible cause [3]	Effect when not detected [4]	Detection / Countermeasure [5]	Effect when detected [6]	Risk priority assignment [7]			
							C	P	D	R
3	Switch	stick in pos. 1	wear out	cannot switch to generator (only when generator is needed)	None → ECU must measure switch output voltage too	same as 2.1	10	3	10	300
4.1	Diesel	does not start	• no fuel • starter battery not loaded • . . .	same as 2.2	same as 2.1	same as 2.1	10	7	3	210
4.2		runs too fast or too slow	failure in diesel engine speed control	hospital net frequency too high or too low → malfunction of electrical equipment	monitor frequency by ECU → call service personnel	same as 2.2	9	3	3	81
5.1	ECU mains in	open circuit	corrosion vandalism	ECU assumes mains voltage too low and switches to generator	signal range check in ECU → call service personnel	same as 1.1	3	5	3	45

Appendix

# [2a]	Component [1]	Failure [2]	Possible cause [3]	Effect when not detected [4]	Detection / Countermeasure [5]	Effect when detected [6]	Risk priority assignment [7]			
							C	P	D	R
5.2	ECU gen. in	open circuit	same as 5.1	generator voltage and frequency cannot be monitored (only when generator is needed)	same as 5.1	none	2	5	3	30
5.3	start out	open circuit	same as 5.1	same as 4.1	same as 4.1	same as 4.1	10	5	3	150
5.4	choice out	open circuit	same as 5.1	same as 3	same as 3	same as 3	10	5	10	500

Top 3 problems (Pareto chart)	0	...	100	200	300	400	500
5.4 ECU signal 'choice out'							
3 Switch sticky							
4.1 Diesel does not start							

Please note: To introduce the basic FMEA principles, only selected components and failure modes where analyzed here. In a real world scenario more components (e.g. wires, connectors, …) and more failure modes (open circuit, short circuit, intermittent contacts, …) need to be analyzed.

Assessing criticality, probability and detectability

A scale of 1 lowest risk / probability, best detectability . . . 10 highest risk / probability, worst detectability can be used. C, P and D should be assessed independently on a project specific basis, e.g.

Criticality *when available countermeasure has been taken*

1, 2	No effect on system function	user will not notice the failure
3, 4	Only unimportant functions are affected	only slightly annoying
5, 6	Systems works with limited function or performance	annoying, user will be unhappy
7, 8	System does not work any more (not safety critical)	user will be severely annoyed
9, 10	Safety critical	may damage life or goods

Probability *for automotive systems*

1, 2	Failure will never occur	Experience from similar systems	<10 ppm
3, 4	Small		<100 ppm
5, 6	Moderate		<1000 ppm
7, 8	High	In similar systems this failure showed up often	>10 000 ppm
9, 10	Failure will definitely occur		>100 000 ppm

Detectability

1, 2	automatically by the system's monitoring function, before the failure's effect shows up.
3, 4	automatically at the same time or shortly after the failure's effect shows up
5, 6	automatically, but long after the failure's effect shows up
7, 8	not automatically detectable, only detectable by human operator
9, 10	same as 7 or 8, but operator cannot take any countermeasures

English – German Glossary (in German: Fachbegriffe)

Englisch	*Deutsch*	*Englisch*	*Deutsch*
Availability AV	Verfügbarkeit, Funktionsfähigkeit	Failure Rate λ (Probability of Failure per Hour PFH)	Ausfallrate, Fehlerrate
Bug	Softwarefehler	Fault	Störung, Fehler
Component	Bauteil	Fault Tolerant	Ausfalltolerantes System, das auch bei Ausfall/Störung einer Komponente funktionsfähig bleibt
Defect	Mangel, Defekt	Fault Tree	Fehlerbaum
Dependability	Verläßlichkeit, Überbegriff für Zuverlässigkeit	Functional Safety	Funktionale Sicherheit
Device	Gerät, Bauteil	Hardware Fault/Failure Tolerance HFT	Hardware-Fehlertoleranz
Dual Modular Redundancy DMR	Zwei-kanalige Redundanz	Harm	Schaden
Electronic Control Unit ECU	Steuergerät	Hazard	Gefährdung, gefährlicher Fehler
Error	Fehler im Sinn von Abweichung	Injury	Verletzung
Event	Ereignis	Line of Code LOC	Programmzeile
Fail Safe	System geht bei Ausfall einer oder mehrere Komponenten in einen sicheren Zustand	Maintenance	Wartung
Failure	Ausfall, Fehlverhalten	Mean Time Between Failure MTBF = MTTF + MTTR	Mittlere Zeit zwischen zwei Ausfällen (Betriebdauer plus Reparaturdauer)

Englisch	*Deutsch*	*Englisch*	*Deutsch*
Failure Cause	Ausfallursache	Risk	Risiko = Kombination von Ausfallschwere und Ausfall-häufigkeit
Failure Duration	Ausfalldauer	Root Cause	Grundursache, auslösende Ursache
Failure Frequency	Ausfallhäufigkeit	Safe Failure Fraction SFF	Prozentualer Anteil der Aus-fälle, die nicht sicher-heitskritisch sind.
Failure Mode	Ausfallart	Safe State	Sicherer Zustand
Failure Occurrence	Auftretenszeitpunkt des Aus-falls	Safety	Sicherheit im Sinne von Ge-fahrlosigkeit
Failure Probability F = 1 – R (Probability of Failure PF)	Ausfallwahrscheinlichkeit	Safety Critical	Sicherheitskritisch, gefähr-dend
Mean Time To Failure MTTF	Mittlere Betriebszeit bis zum Auftreten eines Ausfalls (oh-ne Reparaturdauer)	Safety Integrity Level	Sicherheitsstufe
Mean Time To Repair MTTR	Mittlere Reparaturdauer	Security	Sicherheit im Sinne von Zugriffs- und Datenschutz
Module	Baugruppe	Signal Range Check SRC	Signalbereichsüberprüfung
Monitoring	Überwachung	Stochastic	Zufällig
Notification	Benachrichtigung	Systematic	Systematisch
Overload, Overstress	Überlastung	Triple Modular Redundancy TMR	Drei-kanalige Redundanz
Plausibility	Plausibilität	Wear Out	Verschleiß
Redundancy	Redundanz		
Reliability R = 1 – F	Zuverlässigkeit, Überlebens-wahrscheinlichkeit		

Additional Information:

Electronic Control Units ECUs

Hardware and Software

Hochschule Esslingen
University of Applied Sciences

E.1 Overview 2

E.2 Hardware 3

E.3 Software 8

E.4 Model-based Development 14

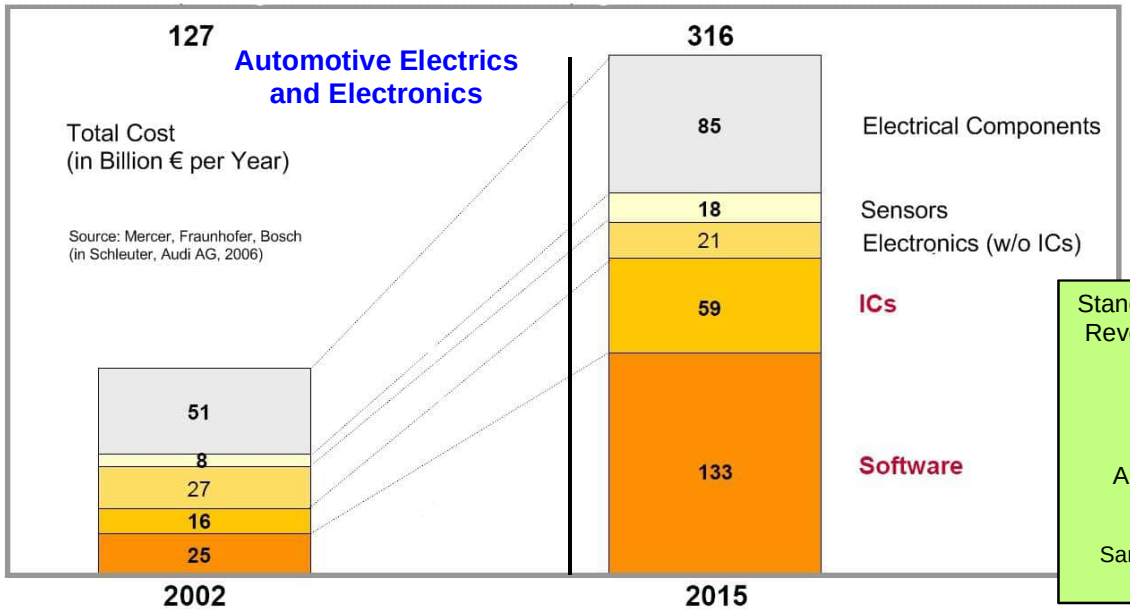
E.5 Electrical Power Distribution and Management 16

Further Reading 20

Appendix: Modelling and Description Methods 22

Appendix: AUTOSAR Classic Standards 30

ECU.1 Overview



Standard Software Industry Revenues 2013 in Billion € of biggest companies	
Microsoft	109
IBM (HW+SW)	92
Apple (HW+SW)	89
Oracle	63
SAP	23
Samsung (HW+SW)	17
Total	393

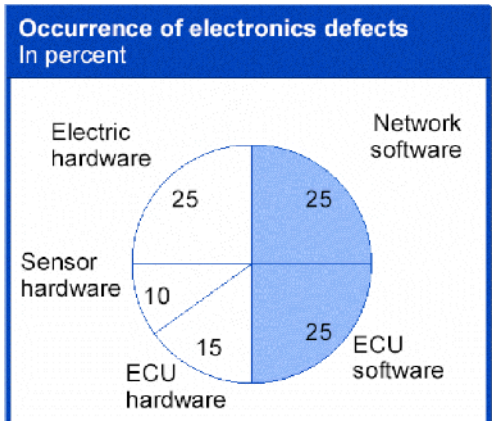
E/E Hardware costs are rising fast

Software costs are rising faster

Trend may accelerate with further electrifi-cation and autono-mous driving

Automotive hardware and software is a major

- competitive factor
- driver of cost **or** revenue

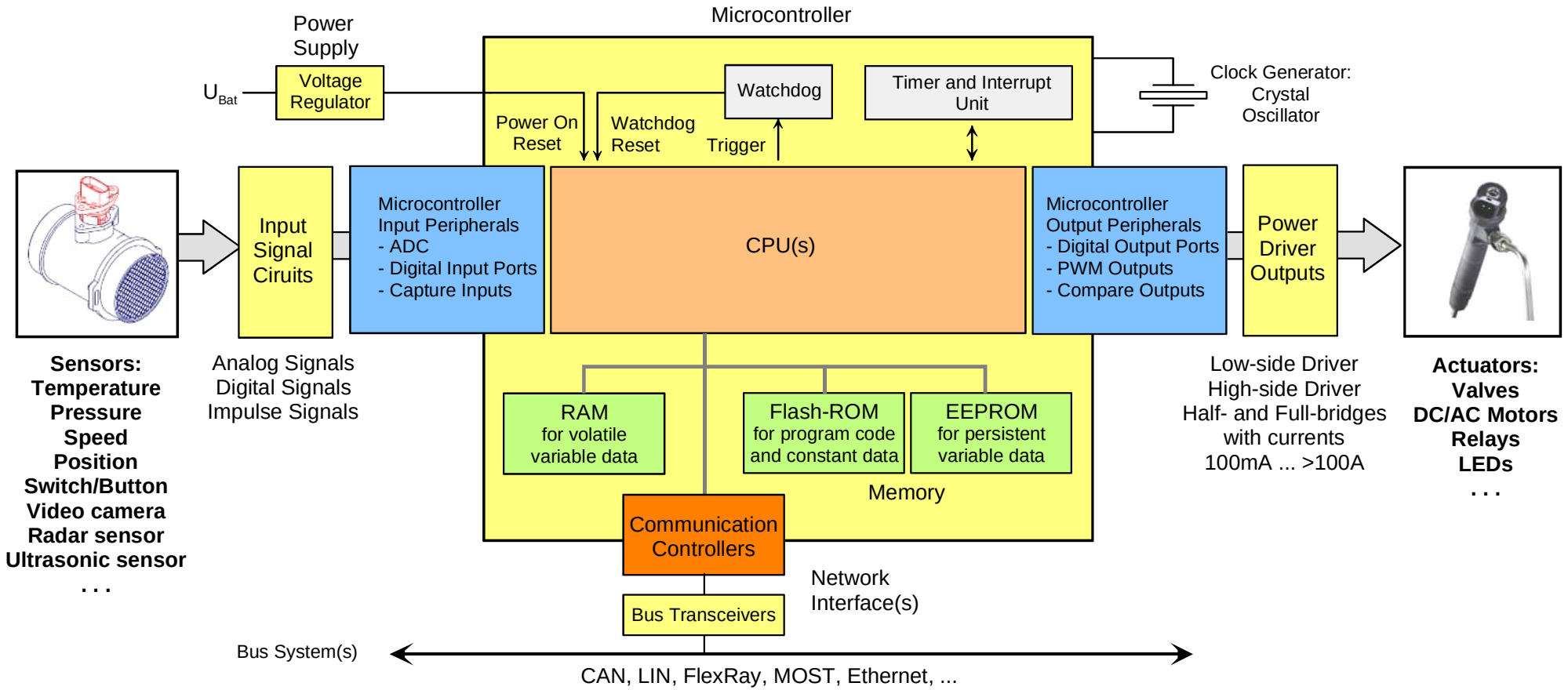


Software complexity **and** failure cost are increasing

Sources:
Left: Schleuter, Audi
Right: McKinsey, Form

ECU.2 Hardware

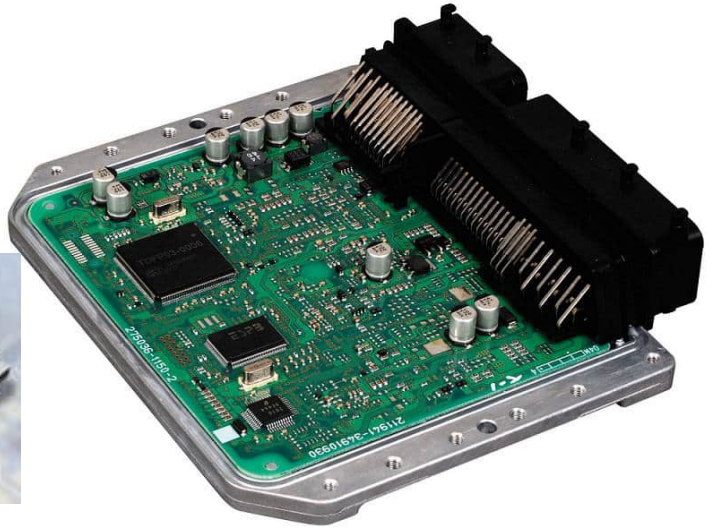
Structure of a typical ECU



Bosch Motronic





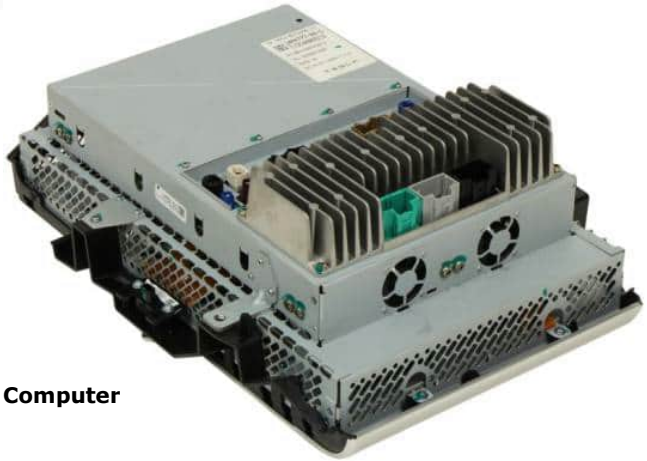


Conti Light Control





Tesla Vehicle Computer



- Metal or metalized plastic box
- Multiple connectors with >100 pins

Inside:

- Printed circuit board (PCB) with integrated circuits and passive components

or

- Hybrid: Ceramic substrate with unhoused chips and "printed" connections/resistors

Design Requirements

Electrical

- Operate at variable supply voltage from 5 … 6 V up to 18V (32V for trucks)
- Inputs and outputs must withstand short circuits to ground and to battery
- Tolerate short voltage impulses (load dump, electrostatic discharge ESD) at all pins
- Electromagnetic compatibility EMC= radio frequency RF interference: Don't disturb other components (interference) – don't be disturbed by other components (susceptibility)
- Failures (hardware/software) must be detected and functional safety must be ensured
- Low operating power, especially in standby operation
- Ensure component availability over car's full production cycles, which is much longer than in consumer electronics
- High reliability at low cost

Mechanical/Environment

- Robust against vibration and shock, level depending on mounting location (engine or chassis – engine compartment – passenger compartment - dashboard)
- Tolerate humidity, splash water or completely sealed
- Tolerate low and high temperatures (-40°C … +85°C … +125°C), temperature cycles and temperature shocks depending on mounting location
- Small size and weight

Trend to higher integration

1975 – today: **Monolithic systems** Each new system added another ECU
Luxury vehicles with 100+ ECUs

1985 – today: **Functional integration** Motronic combined injection and ignition control, later added electronic throttle control

1995 – today: **Mechanical integration** Integrate ECU with sensors and actuators

e.g. ESP system

1st generation: Hydraulic unit, yaw rate sensor, lateral acceleration sensor, ECU
→ 4 separate components *1

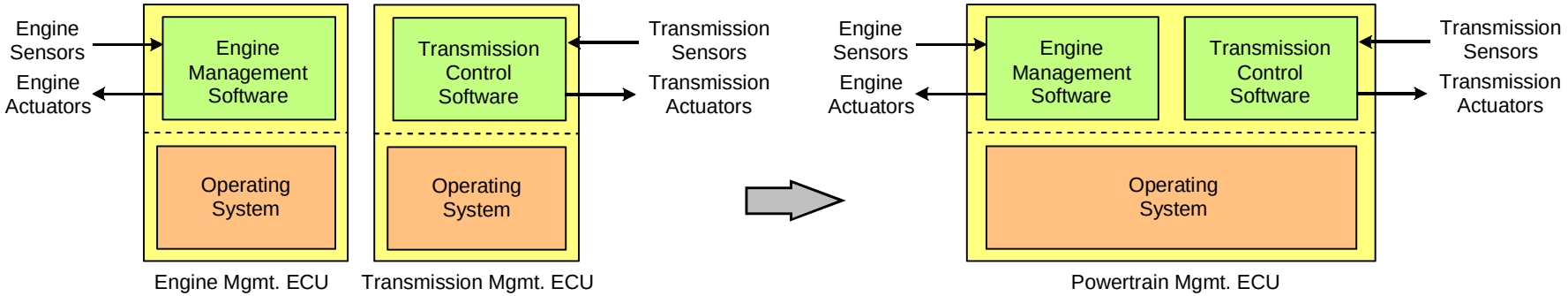
2nd generation: Hydraulic unit + ECU, yaw rate + lateral acceleration sensor
→ 2 separate components *1



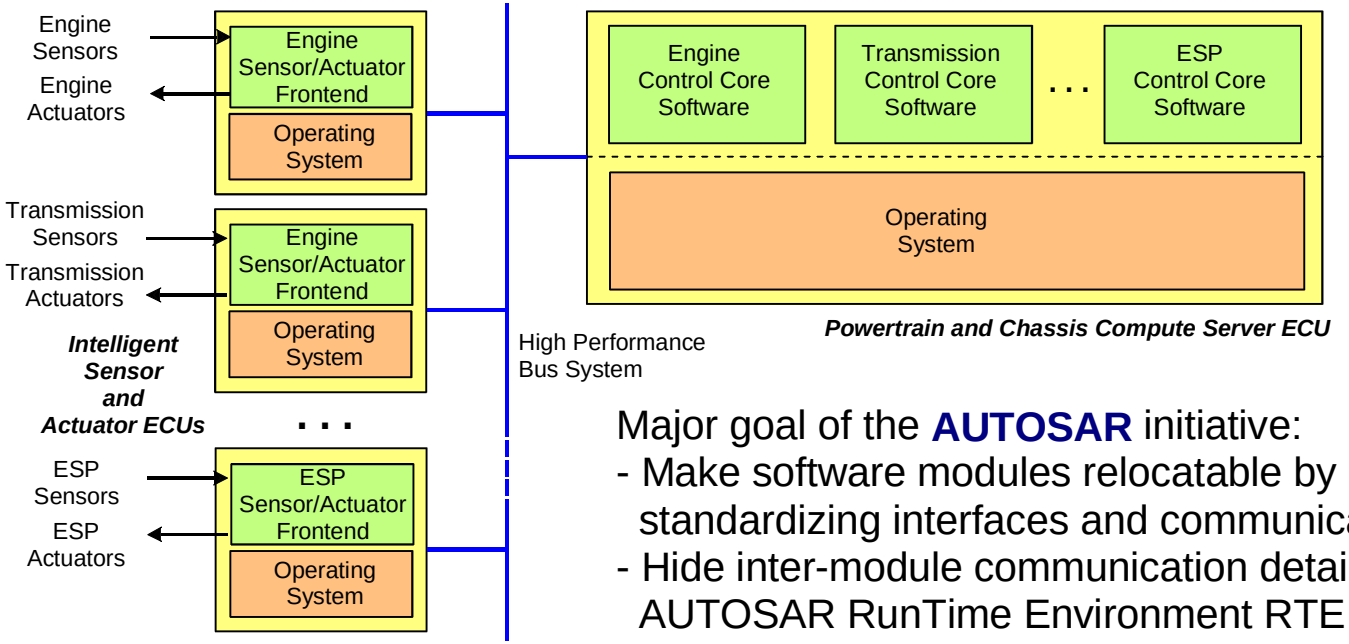


Now: Full integration → 1 single component plus steering angle and wheel speed sensors

Reduce number of ECUs by combining software of different systems in a single **Shared ECU**

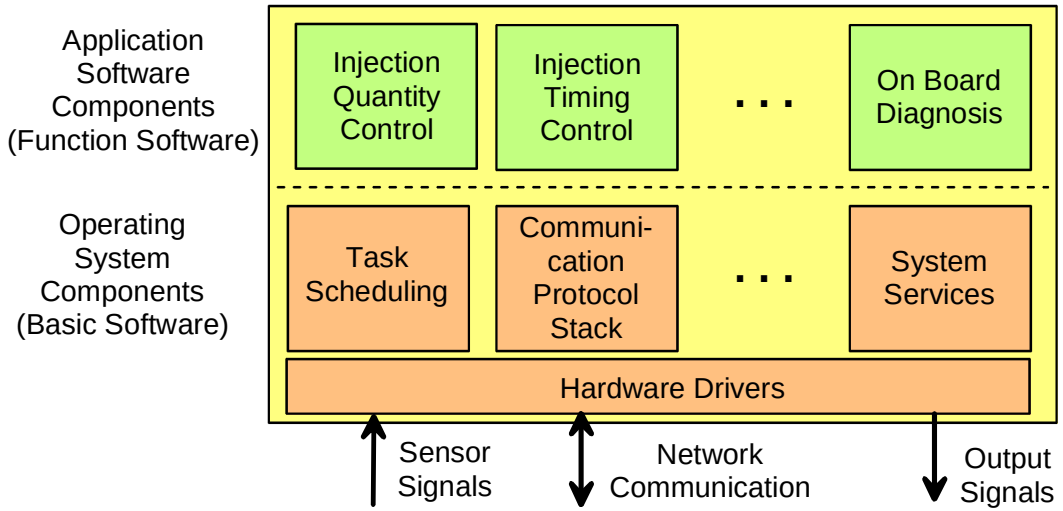


Future: Relocate computing intensive software to high performance **Compute Servers**



Major goal of the **AUTOSAR** initiative:
- Make software modules relocatable by standardizing interfaces and communication
- Hide inter-module communication details by AUTOSAR RunTime Environment RTE

ECU.3 Software



Example: Engine management

SW components consist of

- Program Code
- Data Set
  - Configuration Data
  - Application Data
  - Field Adjustable Data

- **Programming in C/C++** due to memory size and CPU performance constraints, resource-hungry programming languages (C#, Java, Python, …) only in infotainment.
- Increasing use of automatic code generation from UML-, Simulink-or ASCET-models (**model-driven development**)
- **Real-time operating system** AUTOSAR OS (suppliers e.g. Vector Informatik, Elektrobit, …), but also various proprietary operating systems in use, e.g. QNX (a UNIX variant) for infotainment systems
- **Communication protocols** partly standardized AUTOSAR COM and various ISO and SAE standards, but also proprietary communication protocols in use
- **Application-level software** hard to standardize, but AUTOSAR tries to standardize at least the software module interfaces, communication and synchronization mechanisms

Why is ECU software special?

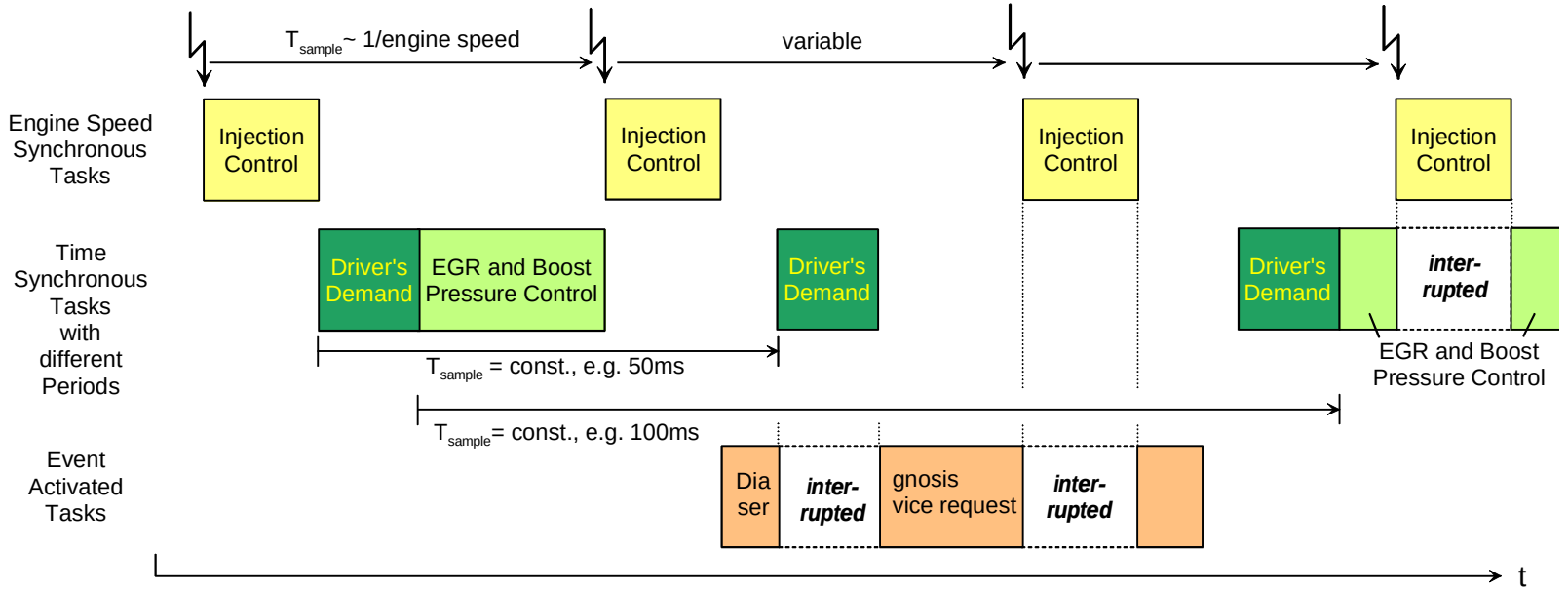
Reliability requirements: **Unreliable software may kill people**!

- Restarting a software module or rebooting the ECU in case of failure is not an option.
- Best-case design does not work: Input signals are noisy, intermittent contacts, broken wires or EMC disturbances do occur.

Real-time requirements

- Software **results** must not only be correct, but must also be **provided in-time**, e.g. ignition control must be synchronized to the rotation of the engine.
- Due to **synchronization to external events**, software **execution sequence** is **variable** and can't be perfectly predetermined and reproduced in lab tests.

Task Scheduling

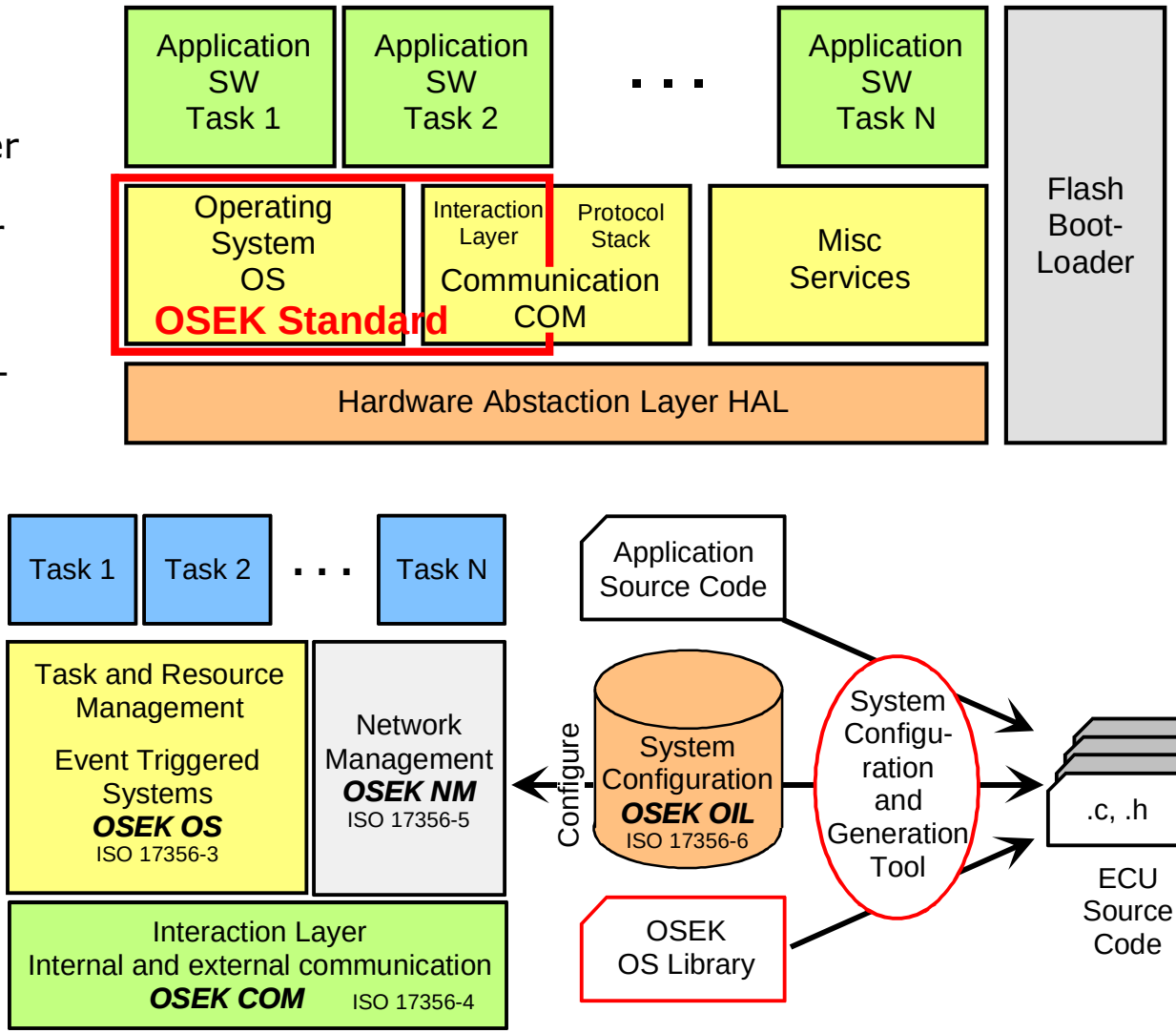


Since mid 1990s ECUs use the

Structured SW-Architecture

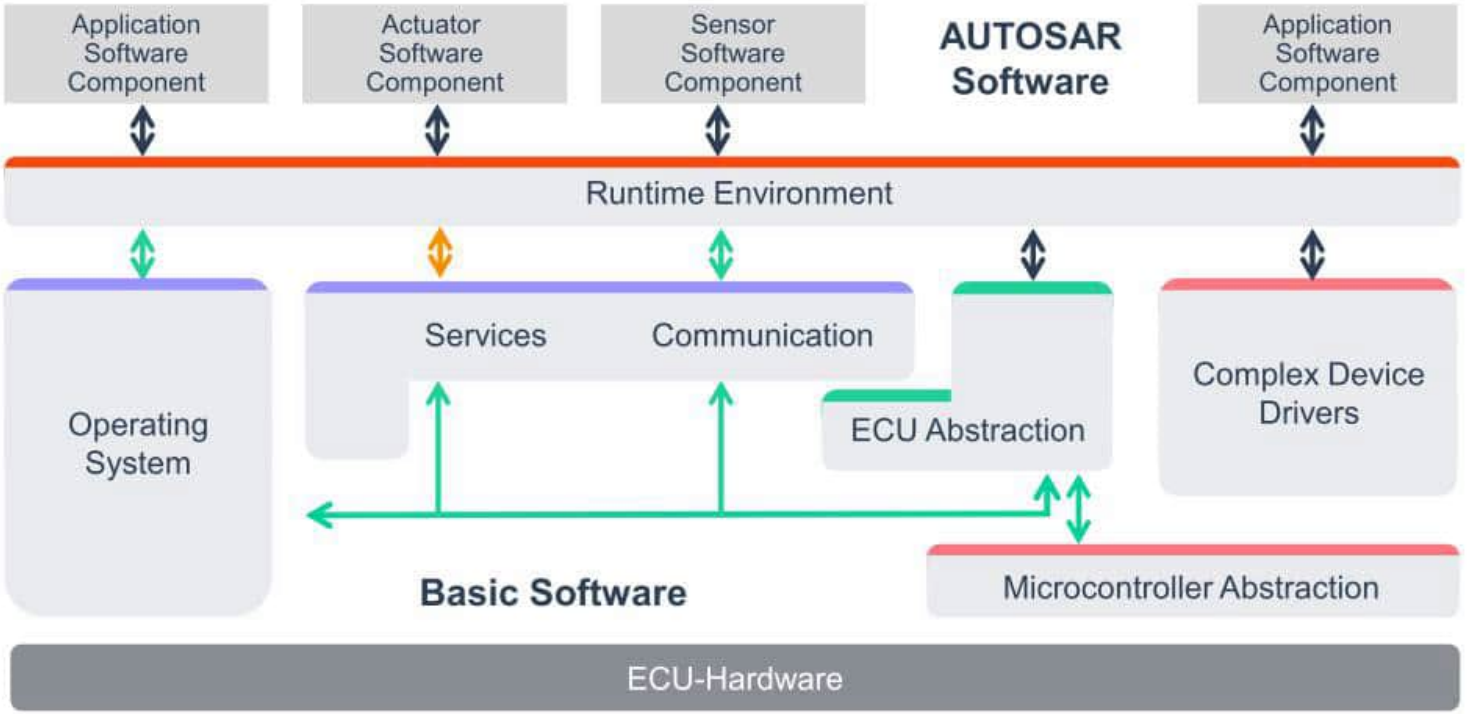
- Application SW organized in tasks which logically run in parallel
- Standardized multi-tasking scheduler **OSEK OS** (mini operating system) and interaction layer **OSEK COM** for on-board communication [S1], now migrated to **AUTOSAR OS/COM**.
- Other modules proprietary with non-standardized interfaces
- Programmed in C
- Application programming interface **API** for task handling, synchronization, inter-task and network communication
- OSEK is predefined (**static configuration**) during the development phase via **OSEK OIL** (OSEK Implementation Language).
- A tool generates C header and OSEK library files, which are linked with the application code

OSEK … Offene Systeme und Schnittstellen für die Elektronik im Kraftfahrzeug – Open Systems and Interfaces for Automotive Electronics [S1]



Since 2010+ ECUs use the **AUTOSAR CLASSIC SW-Architecture** Automotive Open Systems Architecture [S3]

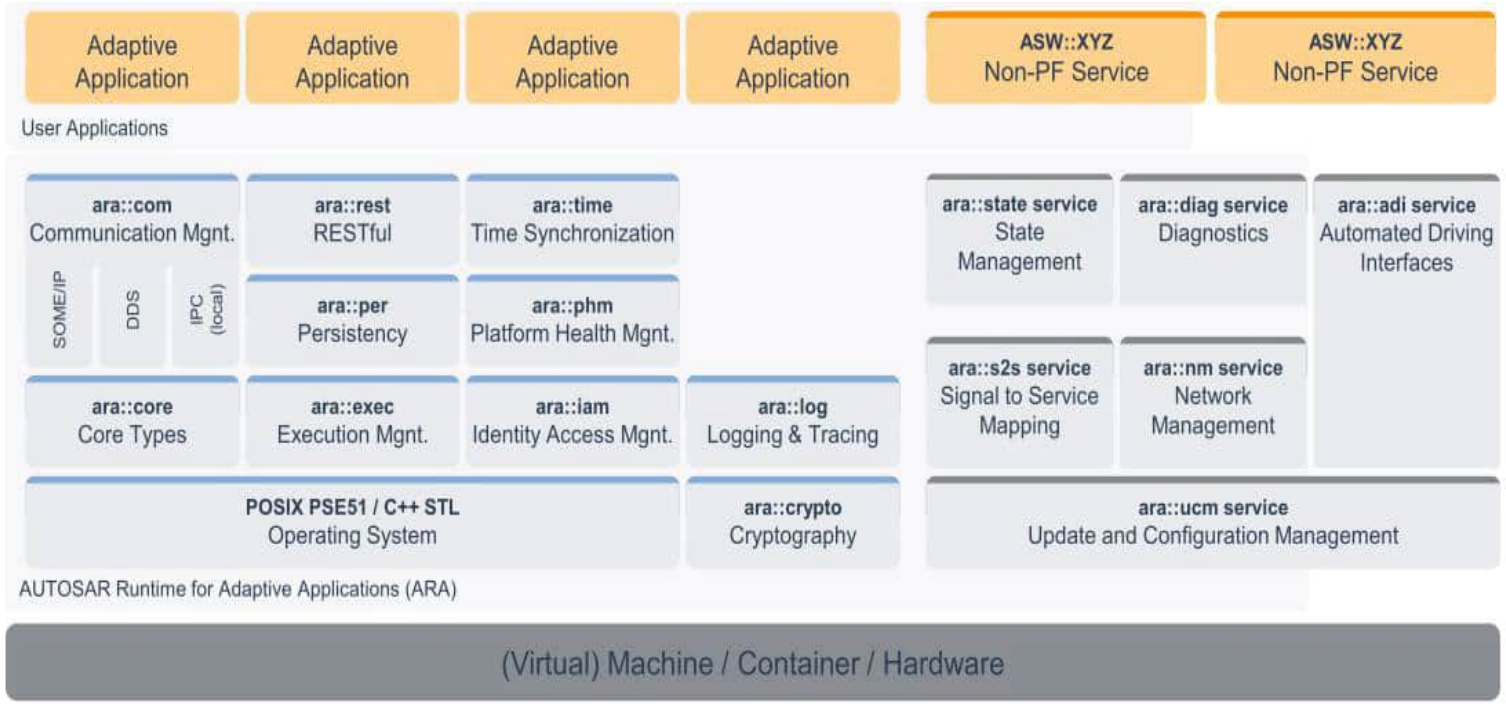
- Based on OSEK OS / COM
- Standardized full-fledged "operating system", called **Basic Software**
- Standardized abstract interface for application SW components, called **Runtime Environment RTE** (Virtual Function Bus VFB)



- Runtime Environment **RTE** interface **generated by tool chain** (rather than being programmed)
- **Basic SW** shall become **off-the-shelf software**, configured by OEM and ECU supplier
- **Application SW components** (not part of the standard!) programmed in C or C++
- In the long run many SW components shall become **configurable standard components**, which can be interchanged between various ECUs and combined from various suppliers

→ PC-like SW concept: Want PC-like prices, but also accept PC-like SW quality?

The AUTOSAR CLASSIC standard focuses on real-time control ECUs with limited resources like engine management, ABS/ESC and body control ECUs. **For infotainment, ADAS and autonomous driving** with a lot of image/video processing, artificial intelligence and internet connectivity in the mid 2010s a new platform **AUTOSAR ADAPTIVE** was defined:

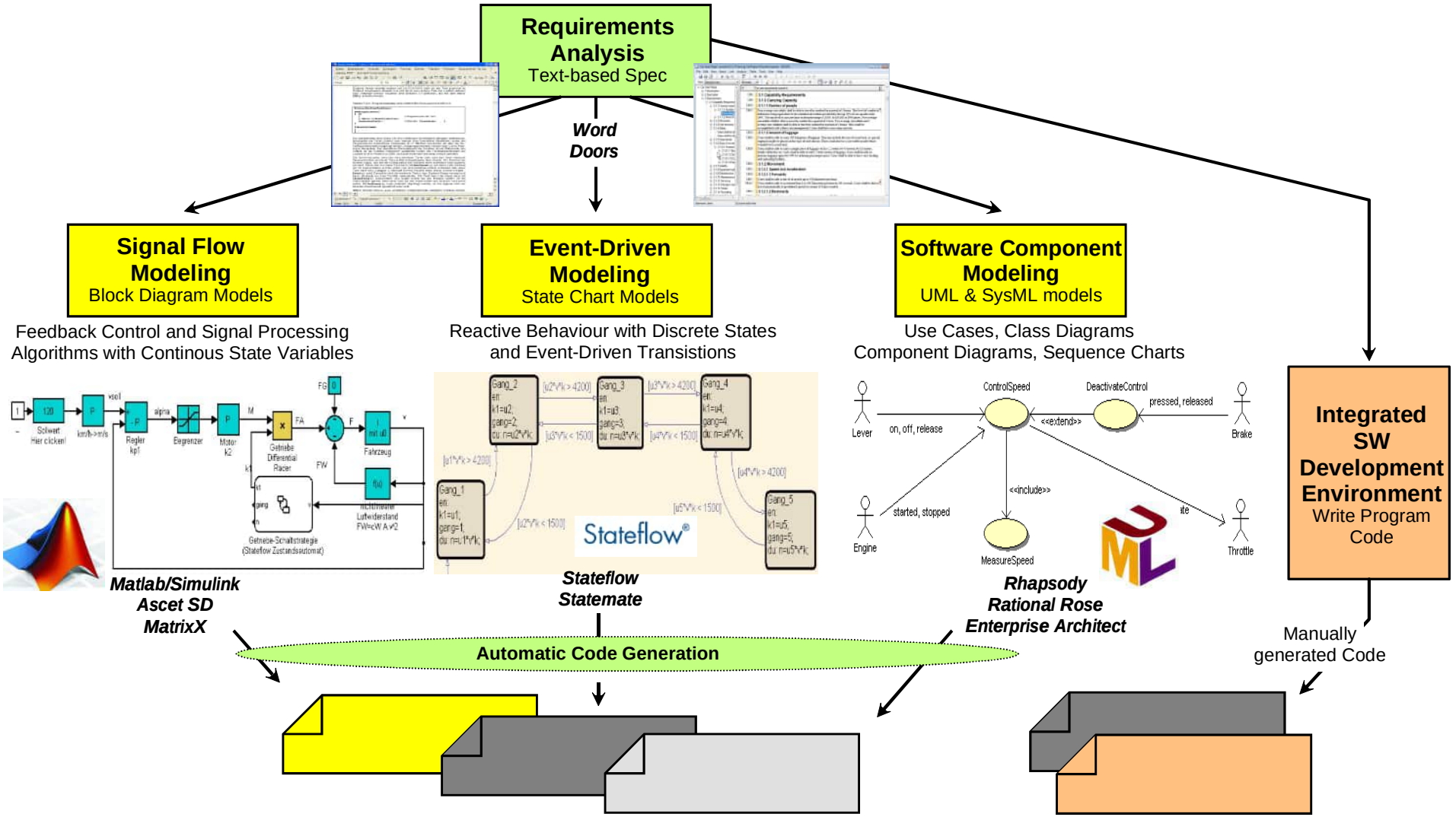


- Dynamically configured RAM based software ←→ rather than ROM based static configuration
- Operating system based on POSIX (a UNIX subset) ←→ rather than OSEK/OS
- Service oriented communication SOME/IP ←→ rather than signal/message based
- Low-level communication services compatible with AUTOSAR CLASSIC

ECU Business Model

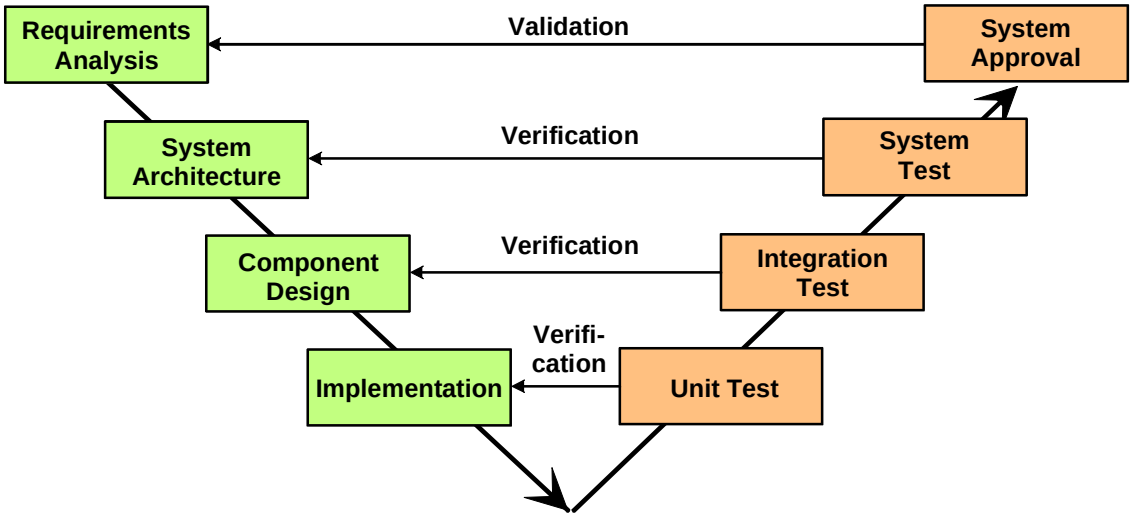
- **1980/1990s**: Role model similar to model for many mechanical components
  - OEMs develop system level requirements and select ECU supplier(s).
  - ECUs suppliers refine the requirements, develop the hardware and software and deliver both as a packaged product. OEMs integrate all ECUs into their cars.
  - Some large ECU suppliers (Bosch, …) establish themselves as systems rather than ECU suppliers only, especially for smaller OEMs. Niche suppliers merge (VDO, Siemens → Continental, …)
- **1990/2000s**: OEMs try to enter the ECU business
  - Some large OEMs (General Motors, Ford, VW, Daimler, Fiat, …) fear growing dependency on ECU suppliers and knowledge leaks and start in-house software and/or hardware development. Daughter companies for ECU production (GM's Delphi, Ford's Visteon, Daimler's Temic, Fiat's Marelli, …) are established or existing suppliers purchased.
  - As pure in-house suppliers, due to lack of scale most of these efforts failed commercially. OEMs sold them to independent investors or closed them. Independent suppliers strengthened.
- **2010/2020s**: OEMs establish AUTOSAR to decouple ECU software and hardware
  - ECU suppliers only to provide hardware and eventually a small part of the basic software
  - OEMs try to develop parts or the complete software in-house, fear growing competition with more software oriented companies like Tesla, maybe Apple or Google
  - Large OEMs establish daughter companies for software development and try to develop their own software platforms (VW's CARIAD, Mercedes MB.OS, …). Results to be seen …
  - Independent suppliers still needed for hardware and remain main SW providers for other OEMs

ECU.4 Model-based Development



Software-based systems have two major drawbacks:

- Converting design into program code ("**programming**") is an **error-prone** process **with limited productivity** (100 … 1000 lines of source code per programmer & month including test & documentation)
- Requirement validation only possible at the end of the development → **long redesign loop**

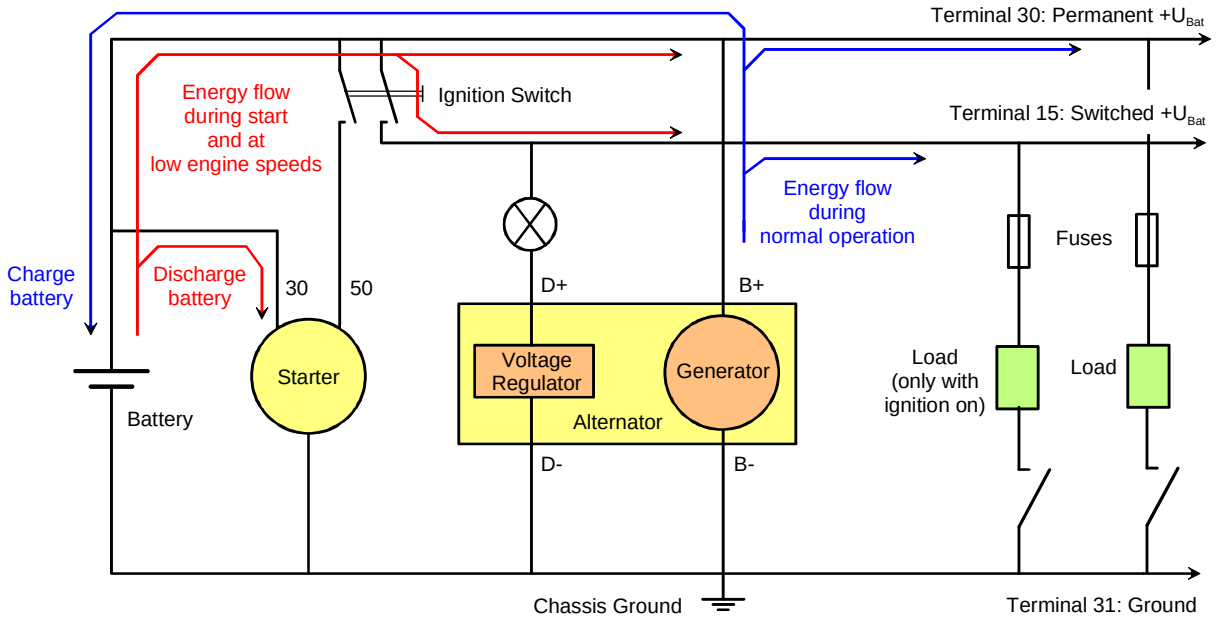


Possible solution: **Model-based Development**

- Describe requirements, architecture and design by **formal models** (block diagram, state charts, UML, …)
- Use **simulation** and rapid prototyping to validate the formal models → **Matlab/Simulink/dSpace**
- Convert the models into production program code by **automatic code generators**
- **Automate the test procedure** by testing program code against the models using real-time **Hardware-in-the-Loop (HIL) simulators**

ECU.5 Electrical Power Distribution and Management

Basic structure



- Permanent and switched supply voltage
- Battery charged from generator, when the combustion engine is running
- Some vehicles use recuperation of mechanical energy during deacceleration
- Battery and generator voltage variable over a large range depending on operating conditions, large transient voltage peaks (e.g. load dump or jump start) possible

		Nominal	*Typical*	*Operating Range*
Passenger Cars Trucks	Europe/US US	12V	14V	6 … 16V
Trucks	Europe	24V	28V	6 … 32V

Typical Electrical Power Consumption (car with classical combustion engine)

Compact Car	*Midsize Car*	*Executive Class*	*Luxury Class*
500 … 1000W	1000 … 2000W	1500 … 3000W	> 3000 W

- Source and further details: Bosch, Automotive Electrics and Electronics, Wiley & Sons

- Electrical power generation contributes to fuel consumption with approx. **1 l per 1 kW·h**

  1 l = 25 g CO_2

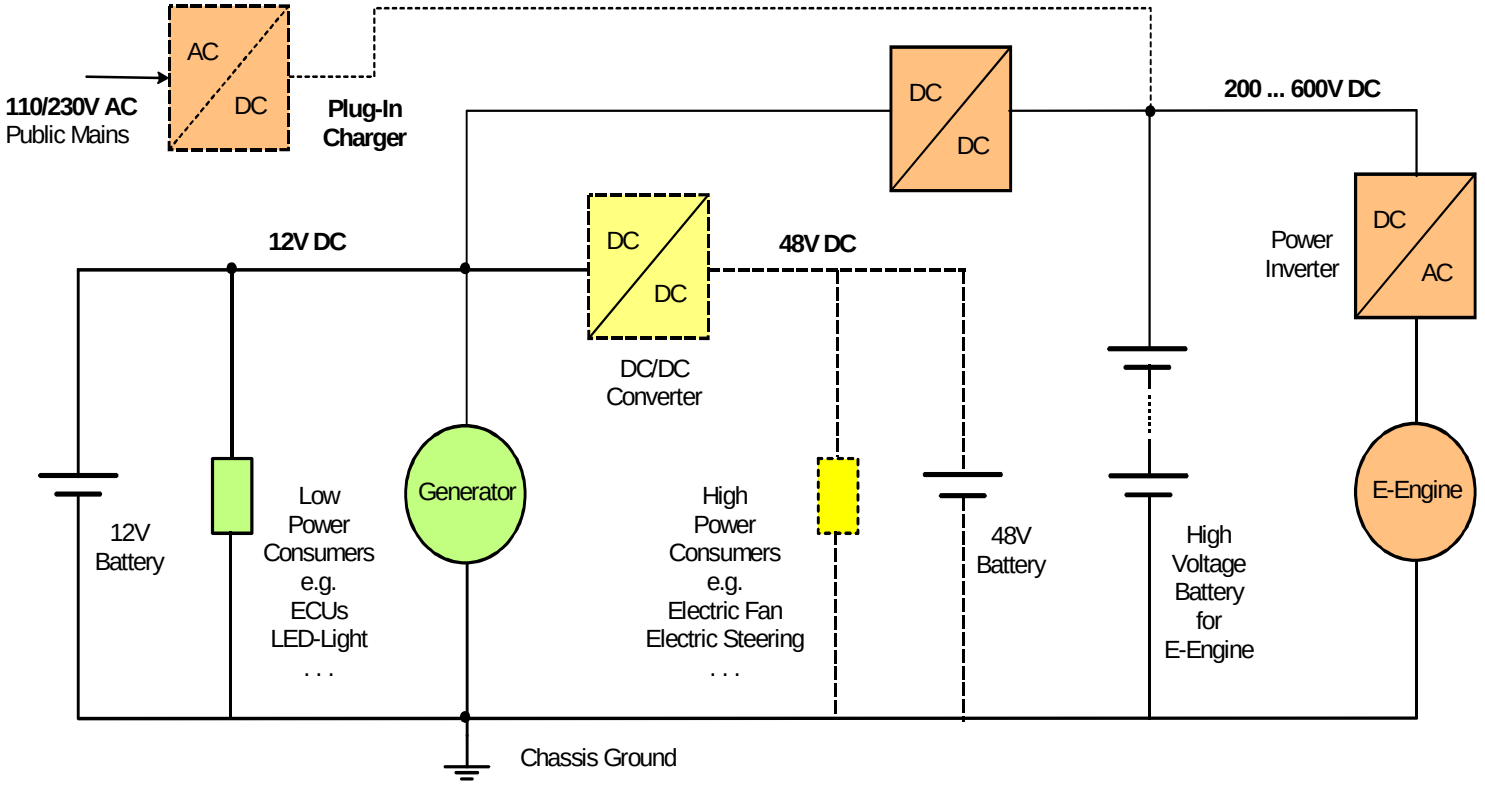
- **Some systems active** (e.g. remote door opener) **or in standby**, **when engine** is **turned off** → battery will be discharged (generator off)
- Some consumers with **high power consumption**, e.g. windshield heater or start-stop operation of combustion engines, **at low engine speeds** → battery will be discharged (generator power too low at low speeds)
- **Electric power generation inefficient** due to cost contraints and low voltage level, e.g.

  $I = P / U = 500W / 12V \approx 40A$ → big cable cross-sections on wires and contacts

Electrical Power Management

- **Estimate battery state of charge and state of health**
- Control generator to **avoid overcharged or deep discharged battery states**
- Ensure battery energy for next engine start by **turning-off various electrical loads**, e.g. seat-heating, door-lock control, when battery state gets critical

- Ongoing "**electrification**" of mechanical systems (electric power steering, electric turbo charger, ...) **exceeds** the **capacity of standard 12V systems**
- Hybrid cars like Toyota Prius use **voltage levels of > 140V ... 800V** → high voltages **and big Li-Ion-batteries** are **safety critical**.
- **Ongoing introduction of an intermediate level of 48V** (operating range with full (limited) functionality 36…52V (24…54V)), especially for "mild" HEVs. Requires a bidirectional DC/DC converter between 12V and 48V level.



Power/Battery Management for HEV/PHEV and EV

- **LiIon** batteries are **temperature sensitive**. At **low temperatures** < 0°C they lose capacity. **At high temperatures** when discharged or charged with high currents they heat up even more and thus may destroy themselves or even start to burn (thermal runaway).
- Fast charging, i.e. charging with high currents is only possible at medium temperatures. At very low and very high temperatures charging currents must be drastically reduced. Thus larger EV **batteries** always need thermal management via a cooling and heating system, i.e. even EVs need sort of a water cooling system!
- Battery lifetime can be improved, when the state of charge (SOC) is kept in a range of approx. 20 … 80%, i.e. batteries should not rarely be fully charged or discharged.

More about EVs and LiIon batteries can be found in the lecture manuscript section "Advanced Mobility".

Further Reading

Books

[6.1] J. Schaeuffele, T. Zurawka: Automotive Software Engineering. SAE International, 1st edition, 2005, http://www.sae.org/technical/books/R-361
Note: There is a more recent 6th edition 2016 in German published by Springer Vieweg.

[6.2] W. Zimmermann, R. Schmidgall: Bussysteme in der Fahrzeugtechnik. Springer Vieweg, 5th edition, 2014 (in German, has an extensive section about OSEK, HIS and AUTOSAR standard-based software)

[6.3] J. Lemieux: Programming in the OSEK/VDX environment. McGraw-Hill, 2001

[6.4] M. Homann: OSEK. MTP, 2004 (in German)

Most books about embedded systems or real-time-systems intensively discuss the special problems encountered in resource-limited software-hardware-systems, e.g.

[6.5] J. Liu: Real-time systems, Prentice Hall, 2000

[6.6] Robert Bosch GmbH (Ed.): Automotive Electrics and Electronics. Springer Vieweg, 2014
(especially pages 152 - 207), https://link.springer.com/book/10.1007%2F978-3-658-01784-2

Standards

[S1] OSEK (Offene Systeme und Schnittstellen für die Elektronik im Kraftfahrzeug) Consortium
Specifications for Operating System (OSEK OS), Communication (OSEK COM) and Network Management (OSEK NM), formerly www.osek-vdx.org → now moved to public standard ISO 17356, www.iso.org

[S2] HIS (Hersteller-Initiative Software) Consortium, www.automotive-his.de → moved to www.autosar.org
Specifications for Flash-Programming & Flash-Loaders and I/O-Drivers.

[S3] AUTOSAR (Automotive Open System Architecture) Consortium, www.autosar.org
Specifications for all levels of automotive software components.

Literature

[S4] MISRA (Motor Industry Software Reliability Association) Consortium, www.misra.org.uk
Guidelines for the use of the C language in critical systems and other aspects of safety-critical software.

Special topics

[T1] A. Coombes: Deadline Timing and OSEK. Design article www.embedded.com, Nov. 2002
http://www.embedded.com/9900832?_requestid=316568

[T2] K. Tindell: Deadline Monotonic Analysis. Embedded Systems Programming, Jun. 2000,
http://www.embedded.com/2000/0006/0006feat1.htm

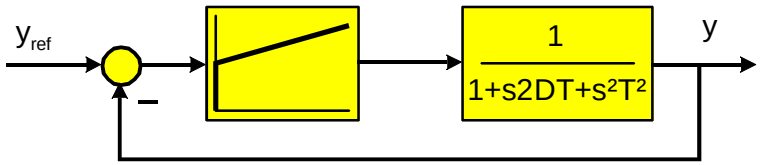
[T3] K. Richter, R. Ernst et.alt.: System Level Performance Analysis – the SymTA/S Approach.
IEE Proceedings Computers and Digital Techniques, March 2005. www.symta.org

[T4] W. Lei et.alt.: Worst-Case Response Time Analysis for OSEK/VDX Compliant Real-time
Distributed Control Systems. IEEE International Computer Software and Applications Conference, 2004

Appendix: Modelling and Description Methods

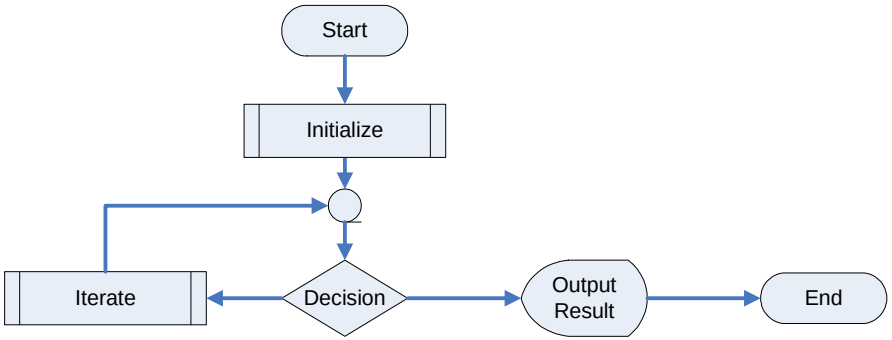
Typical describing elements for models are:

- **Natural language text**. Easy to use, but leaves room for interpretation and cannot be validated formally and/or converted automatically into implementation.
- **Signal flow block diagrams** using linear (transfer functions, step responses) and non-linear blocks (characteristic curves and maps), well-suited to describe feedback control and signal-processing systems

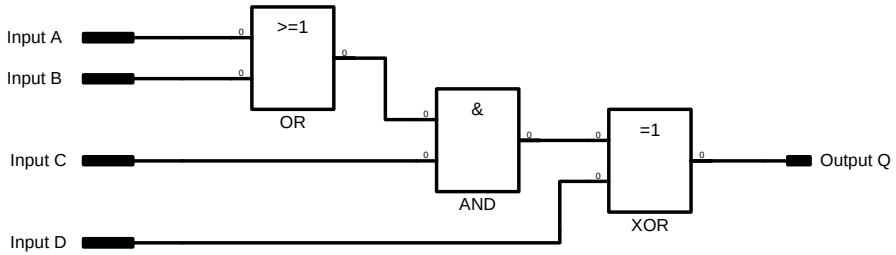


→ Matlab/Simulink, ASCET

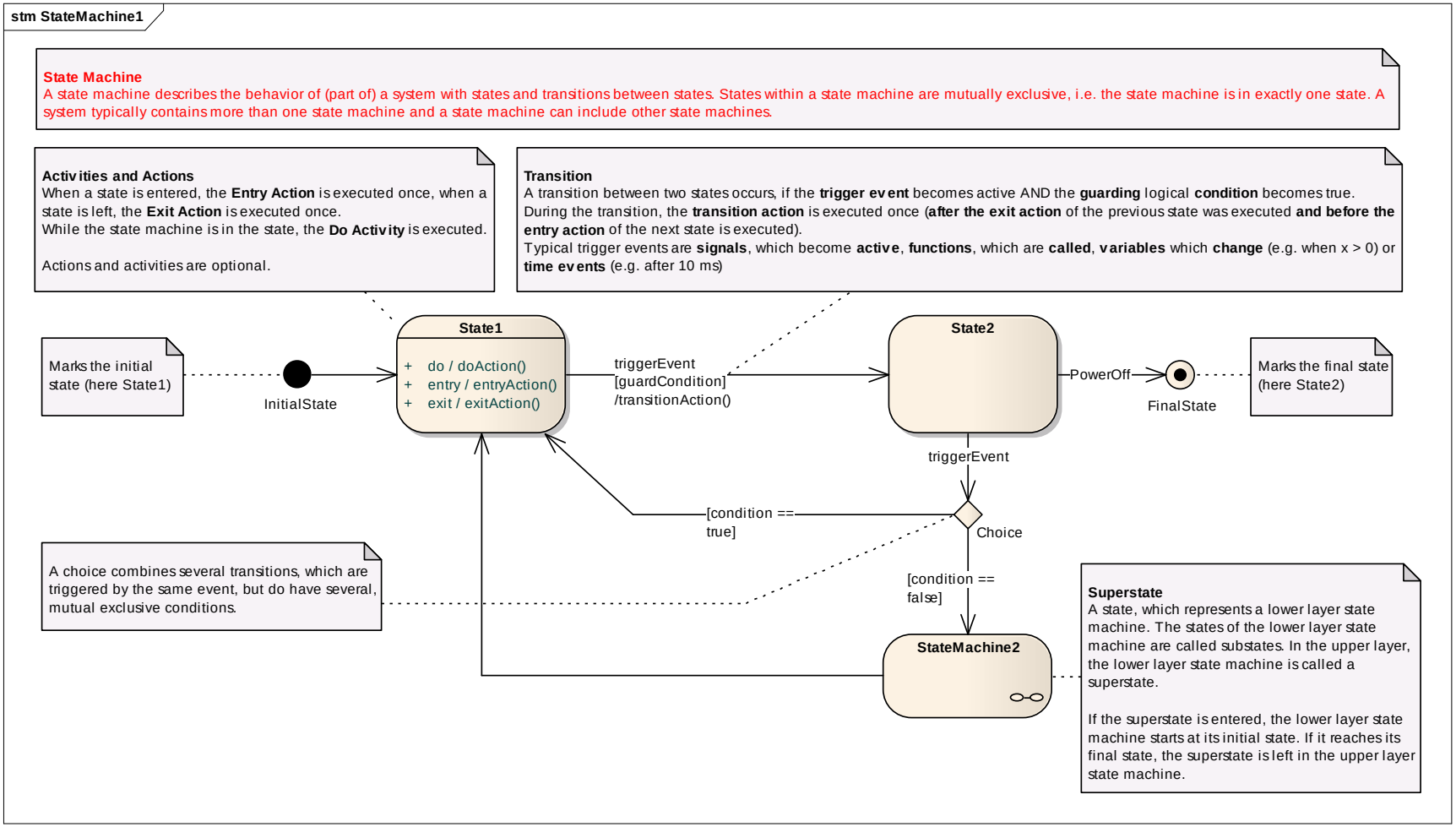
- Program **flow charts** or structure charts, well-suited to describe mathematical algorithms



- **Logic diagrams** and/or truth tables, well-suited to describe stateless logic conditions

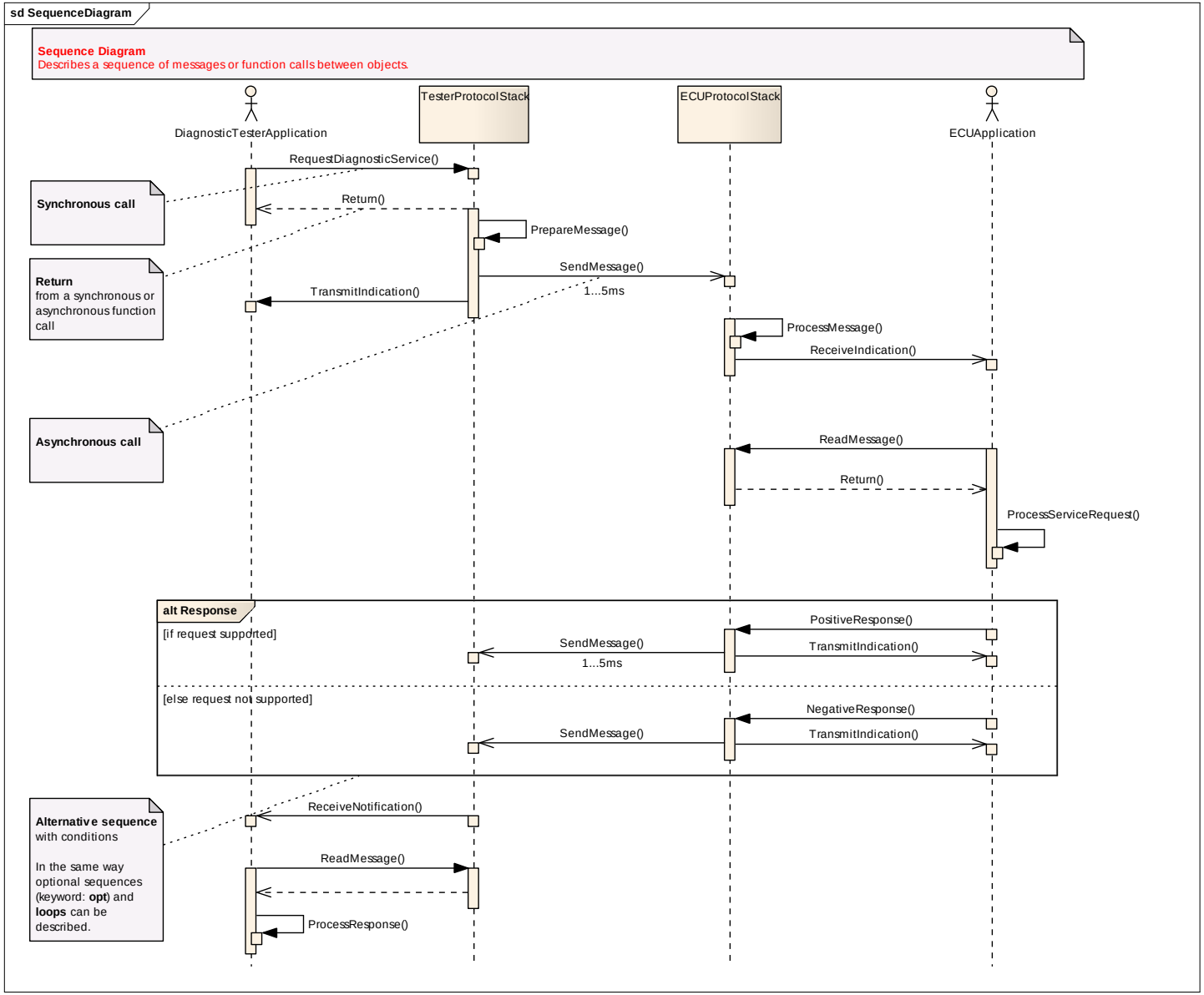


- **State charts** [*1] or state tables, well-suited to describe discrete states and state transitions of systems

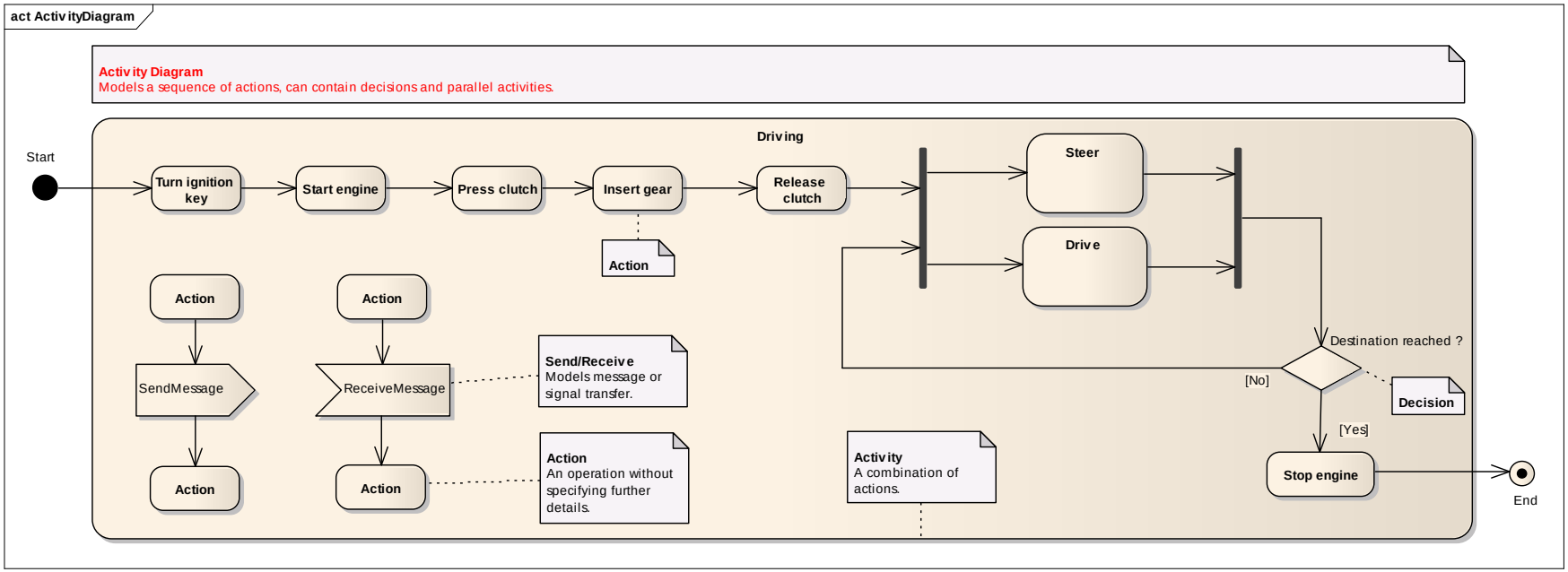


[*1] **Unified Modeling Language** UML description. UML defines diagrams to describe **structure**, e.g. class diagrams or component diagrams and **behavior**, e.g. state charts, sequence diagrams, activity diagrams and use cases.

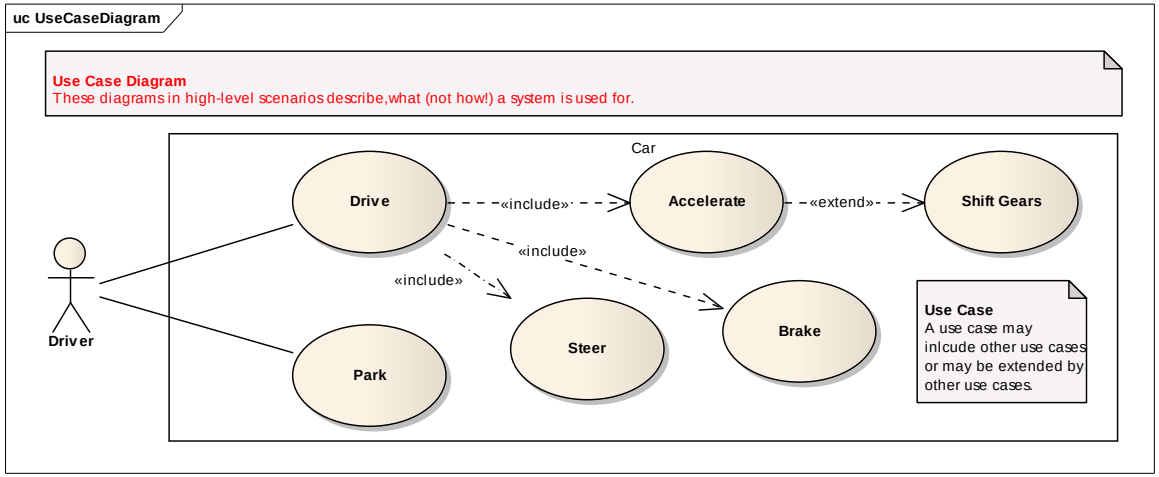
- **Sequence diagrams** *1 and message sequence charts, well suited to describe protocols or the dynamic behavior of systems, e.g. flow of actions and activities.



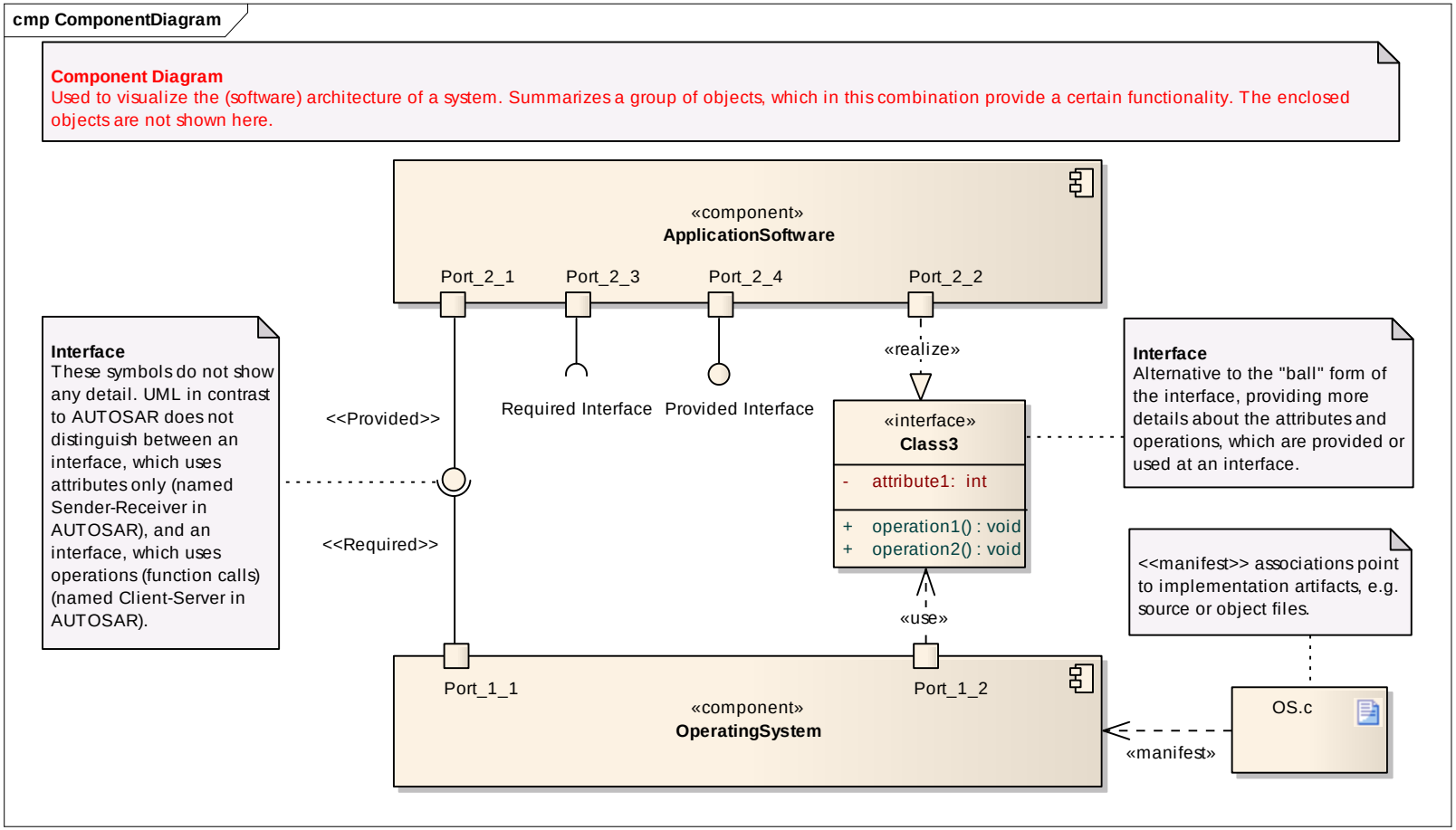
- **Activity diagram**, a combination of state charts and sequence diagrams



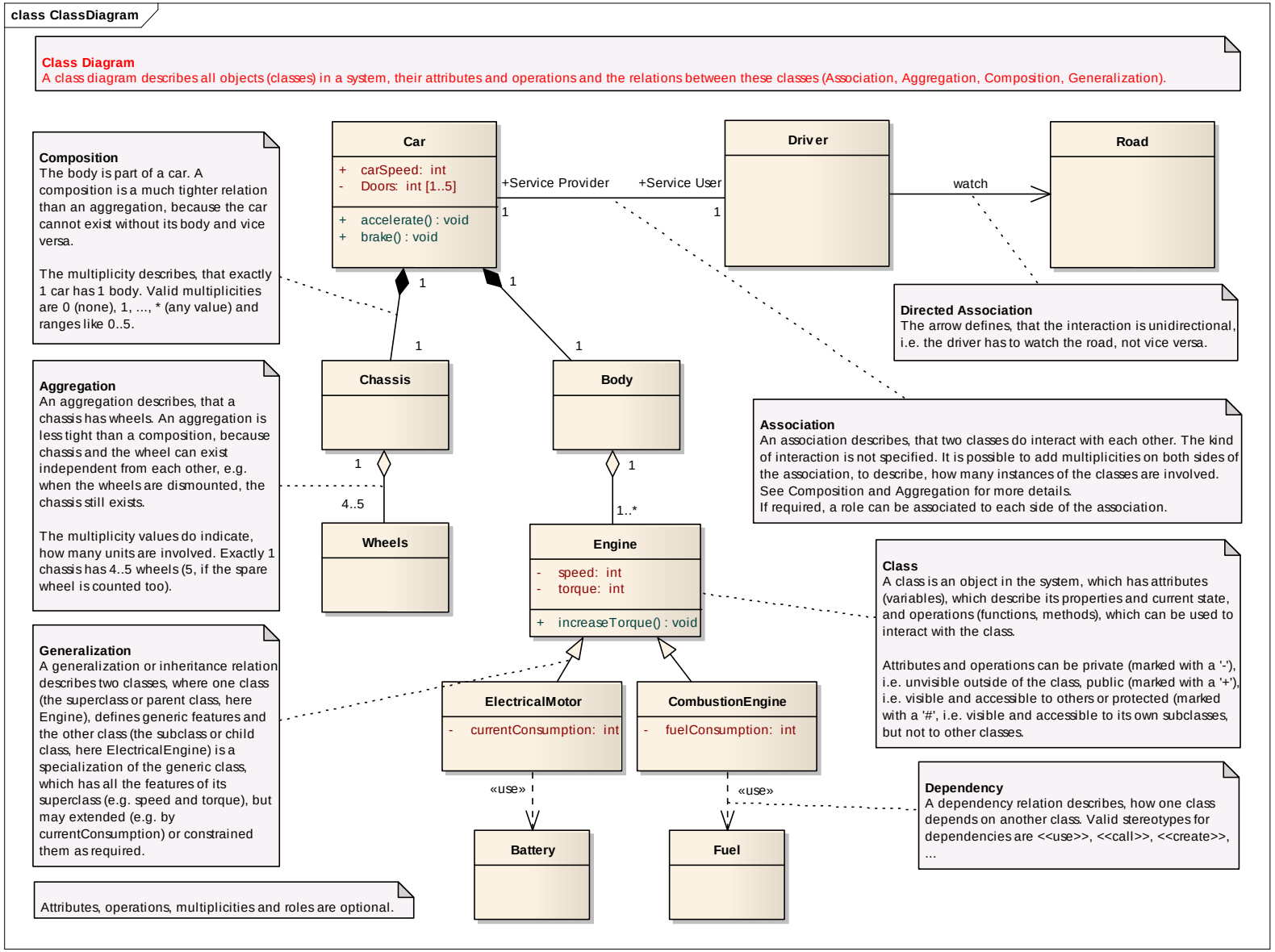
- **Use case diagram**, describing typical high-level usage scenarios of a system



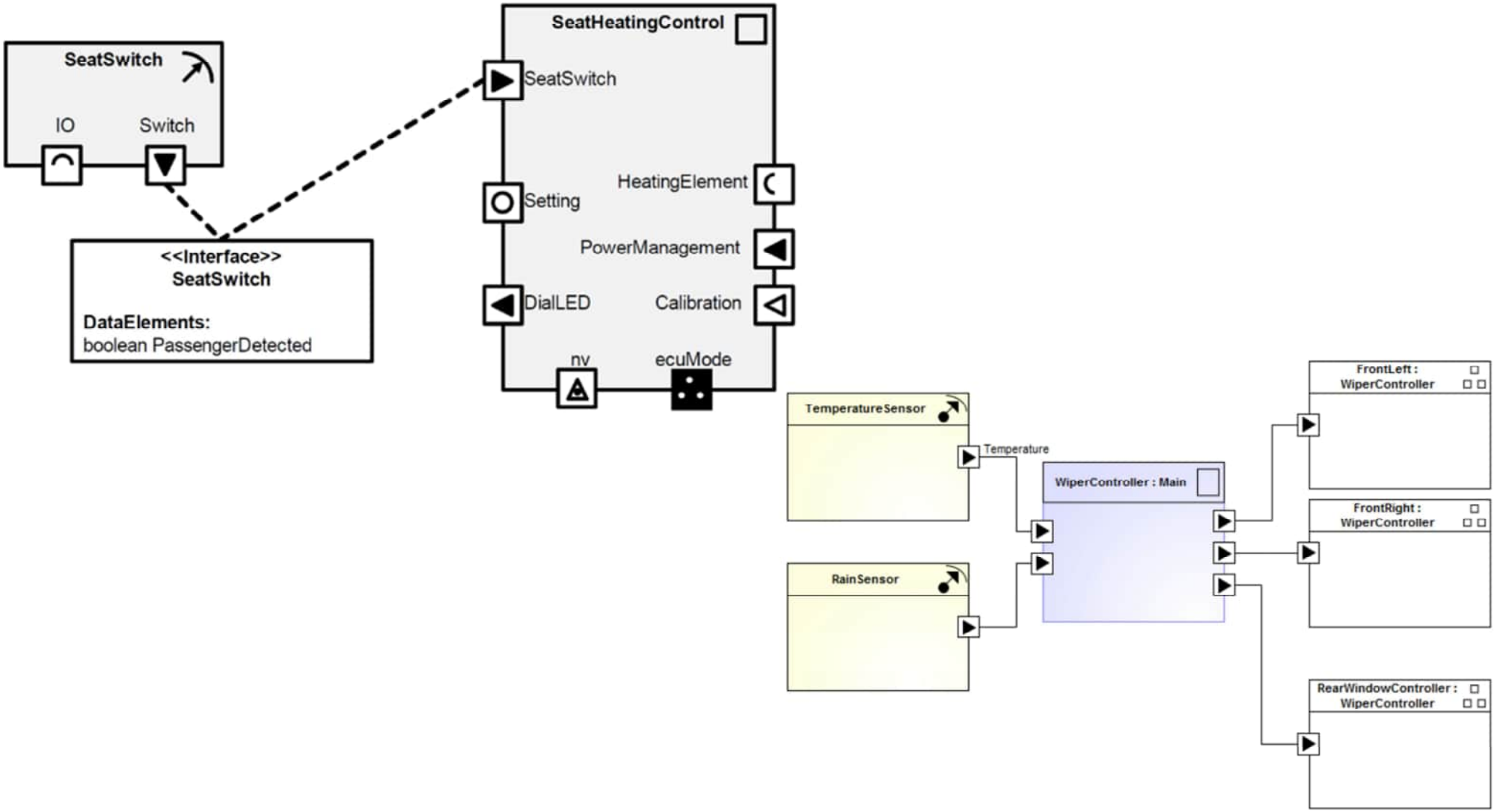
- **Component diagrams**, well suited to describe the architecture of a system by components and their interfaces.



- **Class diagrams** well-suited for a detailed description of the static structure of a system



SysML extends the UML diagrams to describe complete systems



- **Extended Markup Language XML** is a data description language, e.g. for diagnostic services (ODX) or network structures and messages (FIBEX).

  The data elements (e.g. PowertrainCAN) are included between opening tags (e.g. <ECU…>) and closing tags (e.g. </ECU…>), which can be predefined for a certain application. Elements can have a value (data) and attributes, e.g. an element ID or an element name. They can be nested and other elements may be referenced via their ID or name.

  The grammar (tags, required and optional elements) of a data description is defined via an **XML Schema**, so that the format of the data can be validated automatically.

```
<Vehicle>
   <name> AudiVolksPorscheBenz T520 </name>
   <Busses . . . >
      <Gateway-Ref ID="ecu1" />
      <Bus ID="bus1" type="CAN" . . . >
         <name> PowertrainCAN </name>
               . . .
      </Bus>
      <Bus ID="bus2" type="LIN" . . . >
         <name> LeftDoorLIN </name>
         <Gateway-Ref ID="ecu5" />   ⟵ reference by ID
               . . .
      </Bus>
   </Busses>
   . . .
   <ECUs . . . >
      <ECU ID="ecu1" . . . >          ⟵ opening tag
                  <name> EngineECU </name>  ⟵ data
                  <Bus-Ref ID-REF="bus1" />
               . . .
      </ECU>                          ⟵ closing tag
          . . .
      <ECU ID="ecu5" . . . >
                  <name> DashboardECU </name>
                  <Bus-Ref ID-REF="bus1" />
                  <Bus-Ref ID-REF="bus2" />
      </ECU>
   </ECUs>
</Vehicle>
```

Appendix: AUTOSAR Classic Standards [S1 – S3]

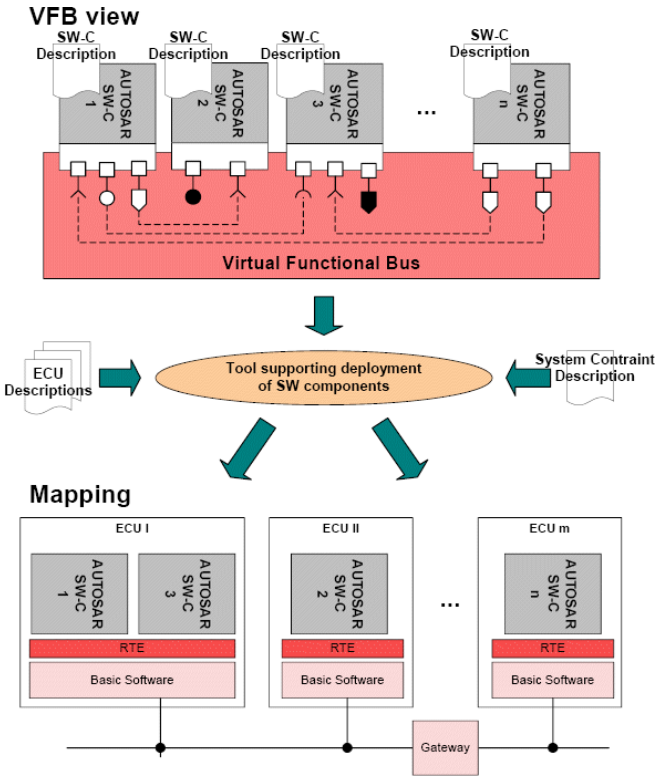
AUTOSAR Classic tries to cover two major areas:

- A complete operating system (AUTOSAR **Basic SW**), provided as a standard component by various suppliers and configured application specific, rather than re-coded.
- **Application SW components** shall be mapped on any ECU as required with well-defined interfaces, which decouples them from hardware and makes them independent from whether their communication partners are internal or external ("Virtual Function Bus").

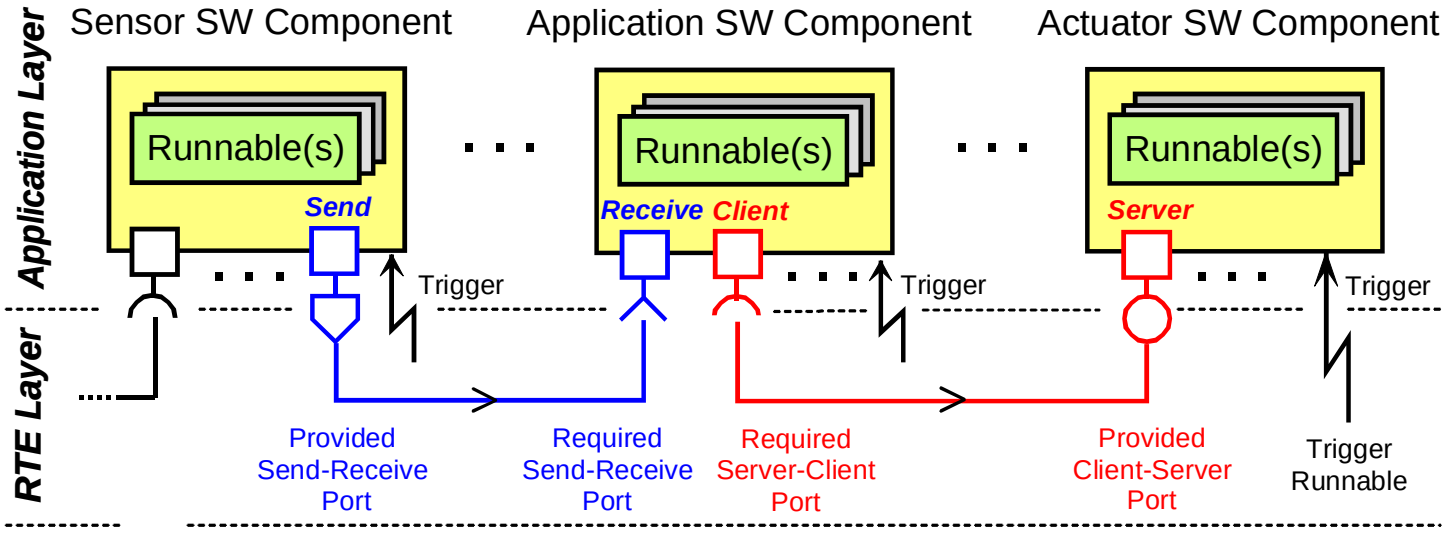
Hardware independent, component based application software development

Automatic configuration of the system based on standardized formats

Automatic configuration/generation of environmental standardized software



- Software components never call Basic SW functions directly (not even the task management AUTOSAR OS or the communication AUTOSAR COM!), but only use the **AUTOSAR Run Time Environment RTE API** (the ECU internal implementation of the Virtual Function Bus concept), which is **automatically generated by** a tool chain from **XML-based SW component description files**.



- Each SW component has well-defined Ports (normally called *Interface* in computer science) of data or functions, which it provides to others (**Provided Port PPort**) or which is uses/calls from others (**Required Port RPort**).

  Data ports are called **Send-Receive-Ports** (marked with round connectors, see figure). Senders can send signals to several receivers. Like OSEK COM, the RTE can either provide an internal data buffer (last-is-best, called Data Distribution semantics in AUTOSAR) or a data queue (FIFO, called Event Distribution semantics). Instead of always accessing the data buffer directly in each read or write operation

(explicit read/write), tasks may request a local copy of the data at task start, which will be written back to the buffer at the end of the task, if required (implicit read).

Generated API functions	**`Rte_{I}Read_<port name>_<name of data>()`**	Explicit/Implicit read with buffer
	`Rte_{I}Write_<port name>_<name of data>()`	Explicit/Implicit write with buffer
	`Rte_{I}Receive_<port name>_<name of data>()`	Explicit/Implicit read with queue
	`Rte_{I}Send_<port name>_<name of data>()`	Explicit/Implicit write with queue

Function ports are called **Client-Server-Ports** (marked with arrow shaped connectors). Server functions can be called by several clients either synchronous (client waits, until the function returns with results) or asynchronous (client continues immediately and is notified later, when results are available).

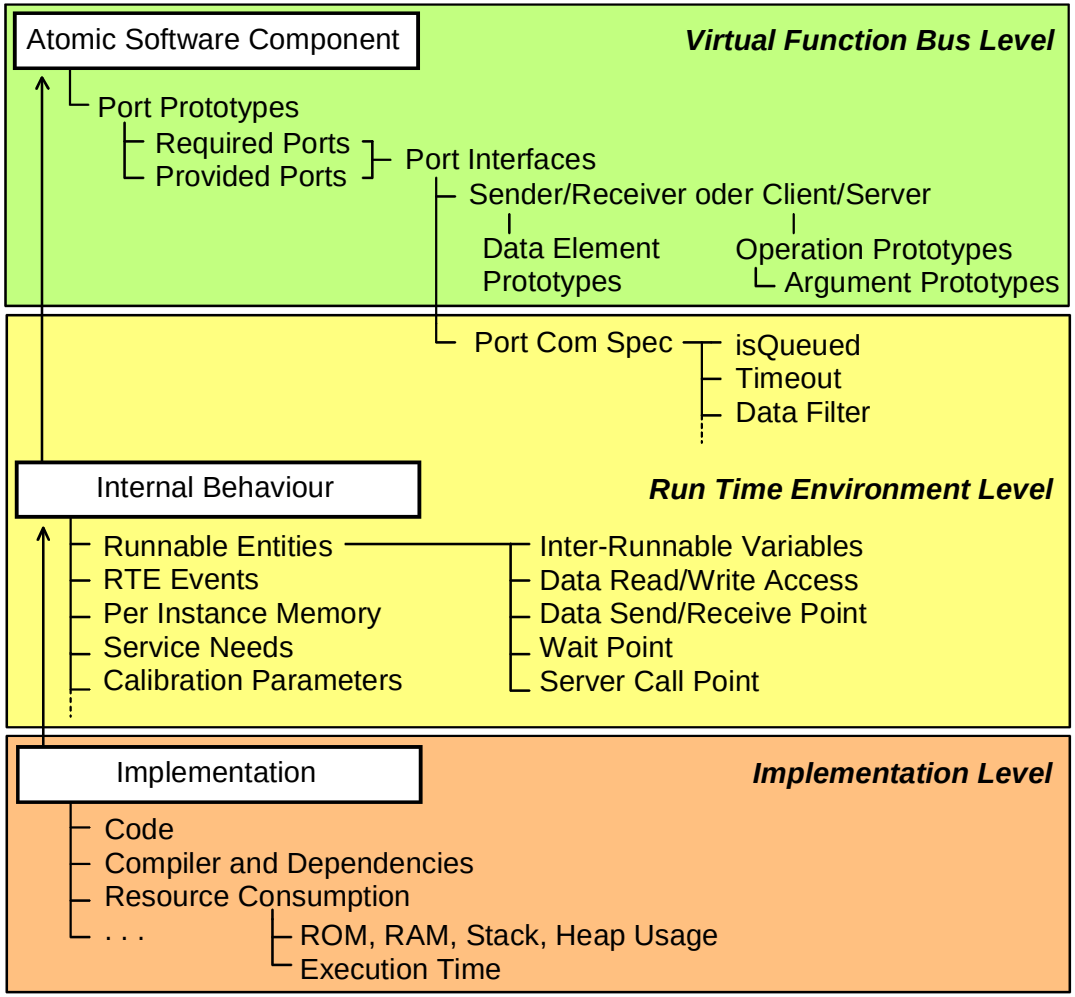
Generated API functions	**`Rte_Call_<port name>_<name of operation>()`**	Client side call of a server function
	`Rte_Result_<port name>_<name of operation>()`	Get results of an asynchronous call

- The program code of a SW component is called a **Runnable Entity** (from the operating system's point of view, a Runnable Entity is sort of a primitive OS task). Runnable Entities are **triggered by** so-called **RTE Events** (an abstraction of OS events). Typical RTE Events are:
  - Timing events: Used to periodically trigger a runnable entity (on the OS level, this is not an OS event, but a task activation).
  - Data received events: Used to signal, when a receiver port has new data.
  - Operation invoked event: Used to signal, that a client has called a server function
  - Asynchronous server call returns event: Used to signal availability of function results

  Events can either start a runnable entity or resume a runnable entity, waiting for the event.

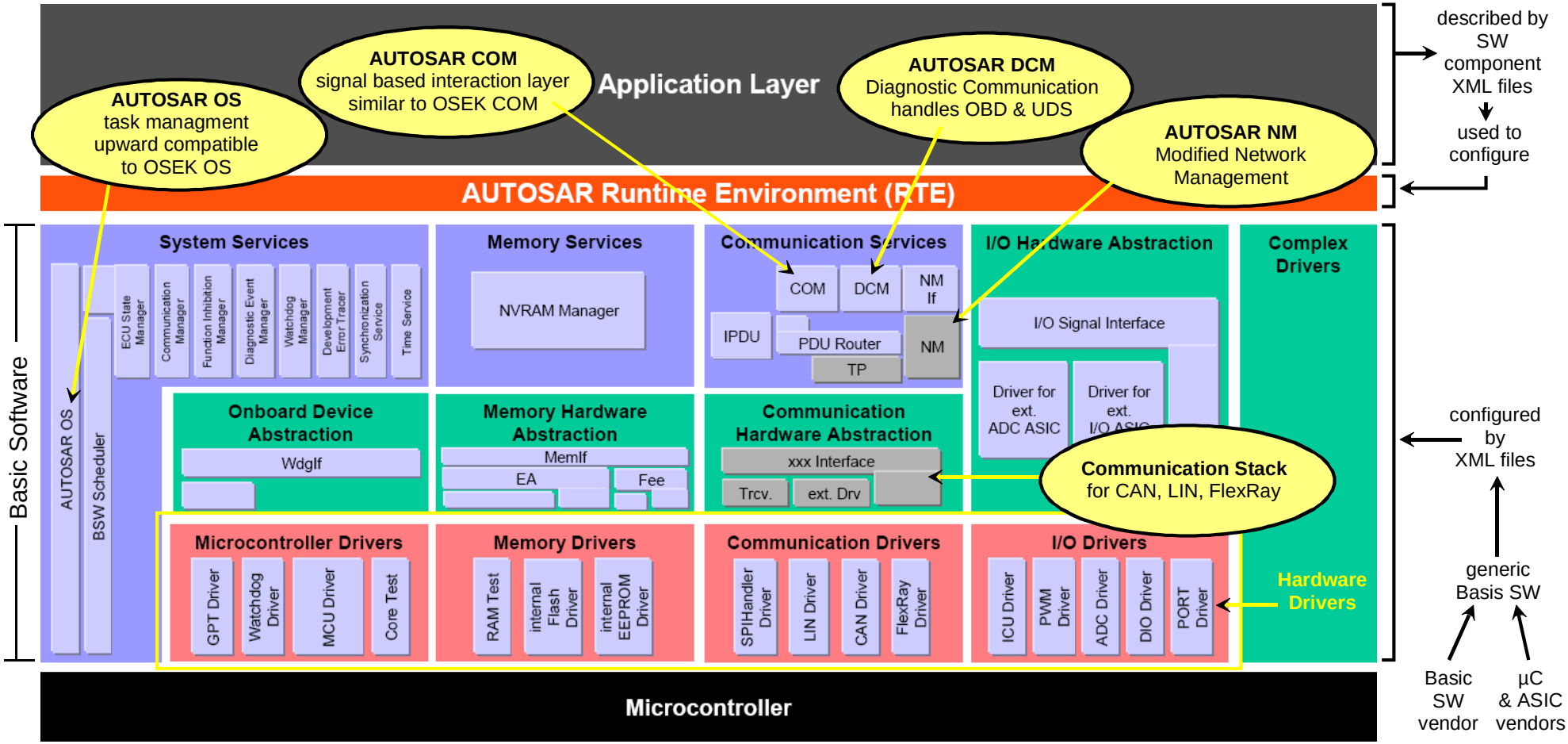
  Runnable Entities and RTE Events are mapped to AUTOSAR OS tasks and events during ECU configuration. So-called Category 1 Runnables not waiting for events internally are mapped to OS basic tasks. Category 2 Runnables using events are mapped to OS extended tasks (see OSEK OS).

- **SW Component Template**:
  A supplier of AUTOSAR SW component has to provide the component's program code (normally .c and .h files) and a XML description file, which contains detailed info about the component's interface (ports) and usage conditions including configurable items, memory requirements, execution time, …



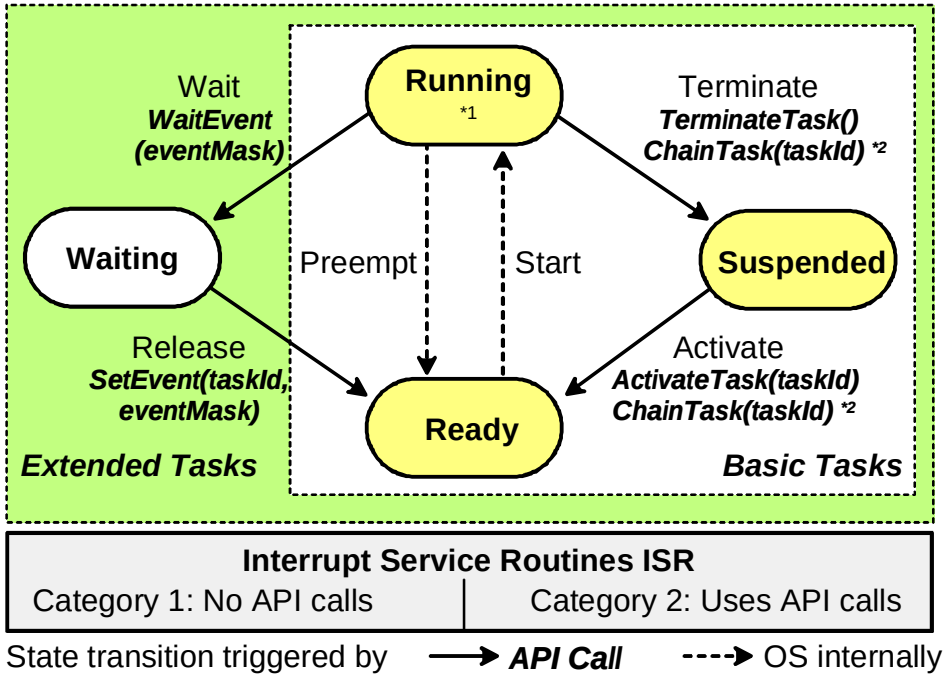
Converting automotive software development to fully use AUTOSAR concepts is an ongoing process, which will still take some more years and requires complex configuration tools, which are available e.g. from Vector Informatik or Continental/Elektrobit.

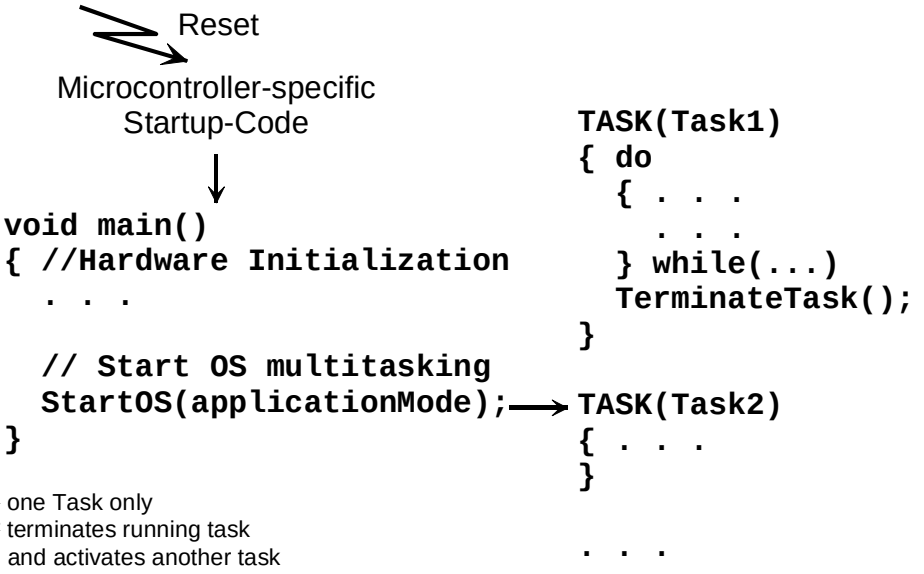
AUTOSAR Classic Basic Software



The Basic SW has more than 80 modules, for each the specification defines the API interface, the internal behavior and configurable items and XML configuration file format

AUTOSAR OS (former OSEK OS) task management uses the following **Task State Model**:





*1 one Task only
*2 terminates running task and activates another task

- Tasks, which want to run, are in state **Ready**. Each task has a fixed priority (0, 1, 2, … with 0 being the lowest priority). Amongst all tasks in the Ready state, the **scheduler selects** the **highest priority task** to run. As several tasks may have the same priority, for each priority level a queue is maintained. Amongst these, always the first entry is selected.
- Exactly one task is **Running**, i.e. its program code is actually executed by the CPU. When a new task is started, the previously running task is **preempted** by the OS (if the task did not terminate itself or switch to Waiting), i.e. the task context is saved and the task is switched to state Ready as the first entry of its priority queue (context switch).

- A task, which terminates itself (API call `TerminateTask()`), transits to state **Suspended**. It can be activated again by another task (API call `ActivateTask()`). When terminating, the suspending task can "chain" another task, i.e. switch it into state Ready (API call `ChainTask()`). Activated tasks are inserted at the end of their priority's queue.
- A task, which needs to wait for an external event, before it can continue processing, switches to state **Waiting** (API call `WaitEvent()`), until the event is set (API call `SetEvent()`).
  Tasks, which never use Waiting, are called **Basic Tasks**; tasks using all states are called **Extended Tasks**.
- For hardware events, **Interrupt Service Routines** (ISR) rather than tasks are still required. ISRs shall only handle hardware-related operations directly and defer other processing to normal tasks. If required, an ISR (so called Category 2 ISR) may use a limited set of API calls, e.g. activating tasks or setting events.

Task Scheduling Policy

If and when a task is preempted (switched from Running to Ready) depends on the scheduling policy, which is **assigned on a task-by-task basis** during system configuration:

- Task with **Non-preemptive Scheduling**: Task will only be preempted, when it switches to Waiting (API call `WaitEvent()`), actively calls the scheduler (API call `Schedule()`) or terminates itself (API calls `TerminateTask(), ChainTask()`), i.e. the task is not preempted even if a higher priority task is activated while the non-preemptive task is running.
  Non-preemptive scheduling often is used for short tasks, when task run-time is in the same order of magnitude as task switching time.
- Task with **Full-preemptive Scheduling**: Task may be preempted any time, i.e. additionally when an event occurs, a resource is released or a higher priority task is activated (by the running task itself, by an alarm or by a Category 2 ISR).

Full-preemptive scheduling often is used for long tasks to give other tasks a chance to run.

Full-preemptive tasks can occasionally block the scheduler, by locking the scheduler as a resource.

- Both types of tasks will be interrupted by ISRs, but non-preemptive tasks will always continue to run, when the ISR returns, while full-preemptive tasks may be preempted, when a Category 2 ISR returns.

OSEK Conformance Classes BCC1, BCC2, ECC1 and ECC2 and AUTOSAR Scalability Classes

OSEK allowed vendors to provide four different implementation levels with differing memory requirements for systems which only use basic tasks with unique priorities (BCC1, lowest class) up to systems with extended (and basic) tasks where several tasks may have the same priority (ECC2, highest class). AUTOSAR uses a similar concept named scalability classes. AUTOSAR's minimal class 1 systems use OSEK OS extended by schedule tables only (see below). Execution time monitoring and memory protection are available in AUTOSAR OS scalability classes 2 (time monitoring only), 3 (memory protection only) and 4 (both).

AUTOSAR OS provides the following **Task Synchronization Mechanisms**:

- **Events**: Tasks can wait for events set by other tasks or interrupt service routines.
- **Alarms** and their associated **Counters** (similar to timers in other operating systems) are used to implement delays, timeouts or periodical activities. Alarms can set events, activate tasks or call functions. Typically, an alarm is coupled with a microcontroller timer, i.e. really causes "time events". However, alarms can also be triggered by any other hardware counter, e.g. increment sensor pulse counters, or even software counters.
- **Resources** (similar to semaphores, mutexes or critical sections in other operating systems) allow exclusive access to shared data, memory or hardware peripherals used by several tasks. (Note: Priority inversion problems are avoided, as OS uses priority ceiling for resources).

AUTOSAR OS extended OSEK OS by

- **Schedule Tables**: Preconfigured list of tasks activated at predefined times. Schedule tables help synchronize tasks with time-triggered communication schedules (FlexRay, LIN), while other tasks still can be event-triggered (classical OSEK OS tasks).

  Time-triggered tasks could be implemented in OSEK OS by using one alarm per task plus a counter. Schedule tables do not need extra alarms and are easier to configure, if many time-triggered tasks are needed.

- **Execution Time Monitoring**: If a task runs longer than a preconfigured maximum time, a hook function can be called to handle the overload situation.

- **Memory Access Protection**: Runtime checking memory address ranges, which may only be accessed by certain tasks. This principle is commonly used in "big" multitasking operating systems (Unix, Windows, …) to protect programs from corrupting each other, but requires hardware support by the microcontroller (**Memory Protection Unit**), which most automotive microcontrollers do not yet have, however.

- Configuration does no longer use OSEK OIL but AUTOSAR XML description files.

AUTOSAR OS API function calls (selection)

API call		*ISR*	*Sched*
Starting and Stopping Multitasking			
StartOS(applicationMode)	Start multitasking; called once outside of any task; only returns, when a task calls **ShutdownOS()**.	-	-
ShutdownOS()	Stop multitasking; terminates all tasks and returns to the function, which originally called **StartOS()**	YES	-
GetActiveApplicationMode()	Returns current application mode	YES	-
Task Management			
ActivateTask(taskId)	Switches a task to state *Ready*.	YES	YES*1
TerminateTask()	Switches the currently running task to *Suspended* and calls the scheduler. Tasks may never return, but may only end with calls to **TerminateTask()** or **ChainTask()**.	-	YES
ChainTask(taskId)	Switches the currently running task to *Suspended*, switches another task to *Ready* and calls the scheduler.	-	YES
GetTaskID(*taskId)	Returns the task ID of the currently running task.	YES	-
GetTaskState(taskId, *state)	Returns the state (one of *Running*, *Ready*, …) of a task.	YES	-
Schedule()	Explicit call of the scheduler. If a higher priority task is waiting, the calling task will be preempted, otherwise it will continue. Used in non-preemptable tasks only.	-	YES*2
Events			
SetEvent(taskId, eventMask)	Set an event. Each task may be assigned a unique set of events, when configuring the operating system. Events typically are implemented as an array of bits in an integer or character variable, which can be set (but not cleared!) via the eventMask. Events cannot be set for suspended tasks.	YES	YES*1
ClearEvent(eventMask)	Used to clear an event by the task which called **WaitEvent()** for the event before; cannot clear events of other tasks!	-	-
GetEvent(taskId, *eventMask)	Get the status of all events for a certain task.	YES	-

WaitEvent(eventMask)	Wait for events specified in **eventMask**, i.e. switches the task to *Waiting*; returns, when at least one of the events specified by **eventMask** has been set by another task or ISR.	-	YES
Resources			
GetResource(resourceId)	Lock a resouce for exclusive use; used for exclusive access (critical section) to shared memory variables or shared hardware components between several tasks. Resource assignement to tasks must be defined when configuring the operating system. When properly configured, due to a feature called Priority Ceiling implemented in the OS, **GetResouce()** will never fail. Resource locking should be as short as possible. If a normally preemptable task does not want to be preempted by other tasks, it can lock the scheduler using **RES_SCHEDULER** as **resourceId**.	YES	-
ReleaseResource(resourceId)	Unlock a resource.	YES	YES[*1]
Alarms			
	Alarms are a timer/counter mechanism, to activate a task, set an event periodically or after a certain 'timeout' or call a function (alarm callback). Alarms, activated tasks and events must be defined during OS configuration. Alarms count in 'ticks', which can be an arbitrary physical unit. The tick counter can be any hardware or software counter, including timers, incremental sensors etc. OSEK does not define an API to configure the counter's hardware side, OS vendors typically provide proprietary function calls like **IncrementCounter(counterId)**.	YES	
SetAbsAlarm(alarmId, absStartTick, cycleTick) **SetRelAlarm(alarmId, relStartTick, cycleTick)**	Set the absolute or relative (to the time when the function is called) start time of an alarm and the cycle period, if periodic (cyclic) alarms are required (0 otherwise). Started alarms must be cancelled, before new values can be set.		

CancelAlarm(alarmId)	Stop an alarm.	YES	
GetAlarm(alarmId, *tick)	Query, how long (in ticks) it takes, until the alarm expires, i.e. when the alarm counter reaches its predefined value.	YES	
Interrupt Control			
Disable/EnableAllInterrupts()	Enable or disable interrupts; used to implement critical sections, i.e. code, which may not be interrupted; if API calls are required within a critical section, use **Get/ReleaseResource()** instead.	YES	-
Suspend/ResumeAllInterrupts()	Same as above, but **Suspend/Resume** calls may be nested.		

ISR … API-function may be called in Category 2 ISR.

Sched … API-function internally calls the scheduler, i.e. the calling task may be preempted.

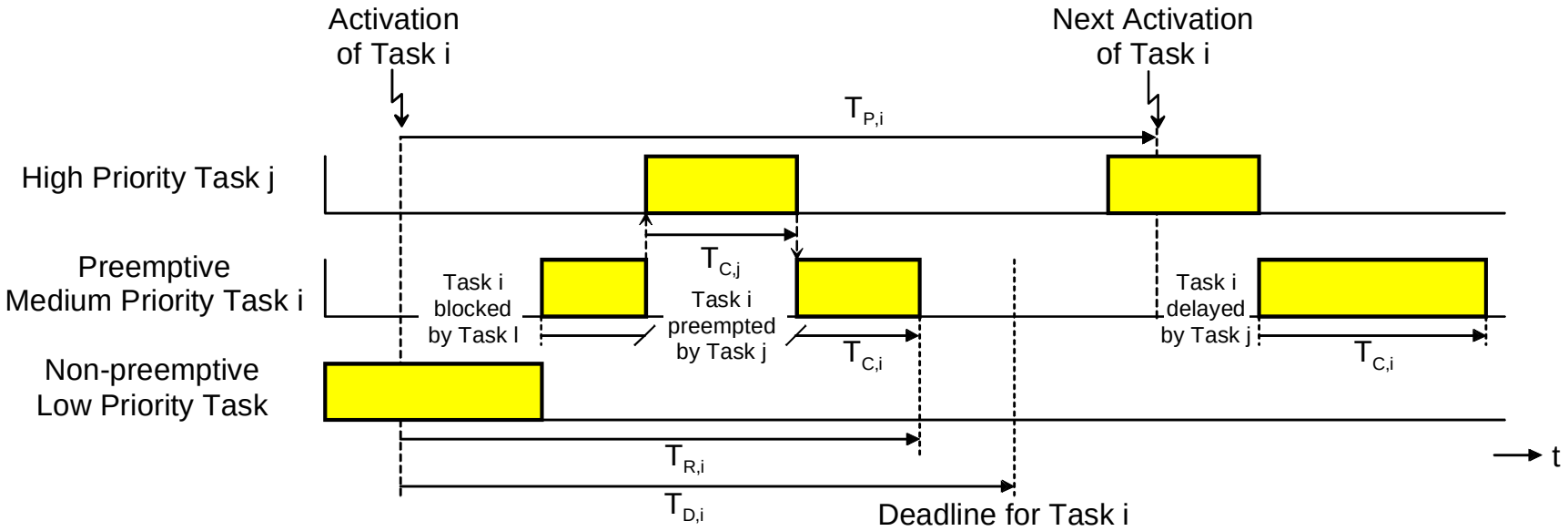
*1 Scheduler called in non-preemptable tasks only *2 Scheduler called for full-preemptable tasks only

Most API functions return **E_OK** in case of success and an error code otherwise.

Hook routines

OS lets the user provide several callback functions (called *Hook* routines), which the OS calls in certain situations, e.g. if an error occurs, when starting or terminating the OS or a task. This allows to "hook" in application defined initialization code and extension of the OS.

AUTOSAR OS Timing



$T_{R,i}$ … Response time = latency from activation to completion of task i $T_{C,i}$ … Execution time for task i

$T_{D,i}$ … Deadline = maximum tolerable response time of task i for correct system operation

$T_{P,i}$ … Period for time-triggered (cyclic) tasks or minimum inter-arrival time for event-triggered tasks

- **CPU Load (Usage)**

  Assume k=1…N tasks with worst case execution times $T_{C,k}$ which are activated with fixed periods (cyclic tasks) or defined minimum inter-arrival times (event-triggered tasks) $T_{P,k}$

  CPU Load $$CL = \sum_{k=1}^{N} \frac{T_{C,k}}{T_{P,k}}$$ (should be << 100% in a production system) (1)

- **Assigning task priorities: Deadline monotonic scheduling**

  Priorities should be selected in deadline order, i.e. the shorter the deadline, the higher the task's priority.

  If the deadline value is not known, use $T_{D,j} = T_{P,j}$, i.e. the task period (for time-triggered tasks) or the minimum inter-arrival time (for event-triggered tasks) → Rate Monotonic Scheduling.

  It can be shown under certain simplifying assumptions that this priority selection leads to optimal task response times for static priority pre-emptive systems, see [6.5, T1, T2] for details.

- **Task response times** depend on the priorities and scheduling properties of all tasks:
  - **Best case response time** for task i: $T_{R,i} \geq T_{C,i}$ (2)
  - **Worst case response time** for task i: The calculation of the worst case response time follows the same principles as the calculation of message latencies for CAN messages in chapter 3.3, eq. (1 – 3):

$$T_{R,i} \leq \sum_{k \in EP(i)} T_{C,k} + \max_{k \in LP(i)} (T_{C,k,\text{ non-preemptive}}, T_{B,k}) + \sum_{k \in HP(i)} \left\lceil \frac{T_{R,i}}{T_{P,k}} \right\rceil T_{C,k} \quad (3)$$

$\lceil \ldots \rceil$ Quotient rounded up to the nearest integer

The first term takes into account the **execution time of task i** plus the **execution times of all tasks of equal priority** as task i (noted as $k \in EP(i)$). These tasks **block** the start of task i, if they are in the scheduler's queue already, when i is activated.

The last term takes into account the execution of all **higher priority tasks** (noted as $k \in HP(i)$), which may **preempt** task i, while i is blocked or running.

The middle term describes the longest time, task i may be **blocked by a lower priority task** (noted as $k \in LP(i)$), which is running, when task i is activated. Such blocking may oc-

cur, if the lower priority task is configured as *non-preemptive* or if it is in a critical section (e.g. interrupts disabled or use of a resource, which it shares with a higher priority task)

Interrupts service routines are considered as having a higher priority than all tasks.

Unfortunately, eq. (3) cannot be solved for $T_{R,i}$ directly, because it occurs on both sides. Therefore, **eq. (3) must be solved recursively**, by inserting a value on the right side, calculating the result on the left side and reinserting the new value, until the value does not change significantly any more. As long as CL < 100%, the result should converge.

Some remarks:

Scheduler execution time is neglected in eq. (3). Task switching times can be taken into account by adding them to the task execution time (one task switch into and one out of each task). For details see [T1 – T4].

Eq. (3) assumes arbitrary phases between tasks, which is the absolute worst case, for event-triggered tasks and cyclic tasks, whose period is not an integer multiple of the base period. By selecting periods as harmonic multiples of the base period (e.g. 1 ms, 5 ms, 10 ms, …) and carefully choosing constant phase offsets between tasks, the actual worst case response times can be drastically reduced (see [T3] for an example).

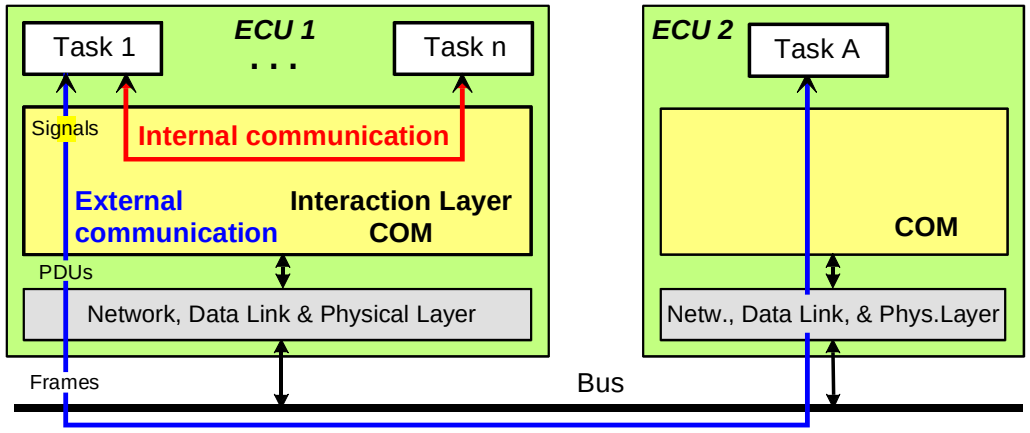
Note:

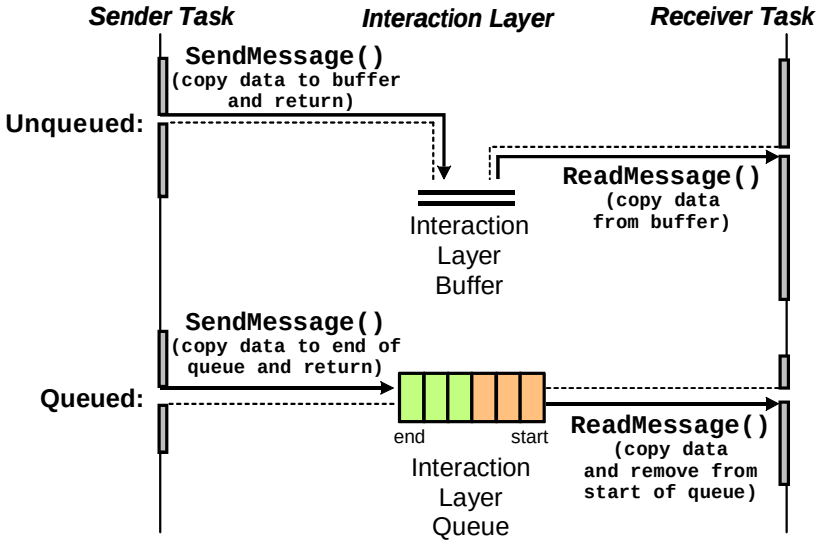
The scheduling analysis presented may lead to very **pessimistic results**, because it assumes, that all tasks are activated at the same instant (worst case) and worst case execution times for all tasks shall be used. Very often, measured results are much smaller, because the worst case execution times typically are much bigger than real-world average times and activation of all tasks at the same instant occurs rarely. For time-triggered tasks, known task phasing should be taken into account, to reduce the amount of pessimism.

On the other hand, the calculation uses the assumption, that event-triggered task activation occurs at a certain minimum inter-arrival time. If events occur more frequently, the calculation leads to **optimistic results** and real task response times may even be unbound. In any case, minimum inter-arrival time should be monitored and appropriate measure be taken, e.g. not allowing task activation, if events occur too frequently (graceful degradation).

AUTOSAR COM Communication (based on OSEK COM)

- AUTOSAR provides a message-based interface named **Diagnostic Communication Manager DCM**, which by default supports OBD and UDS, but can be extended for other protocols. DCM uses transport layer ISO TP for CAN, LIN and FlexRay.
- For real-time control **AUTOSAR COM** implements a unified **signal-based interface** (similar to OSEK COM, see below) to all supported bus systems (CAN, LIN, FlexRay) plus limited routing functionality between bus systems via the PDU router module. COM does not use a transport layer to reduce overhead and latency for time-critical signals.
- COM allows **asynchronous communication** between tasks on the same CPU (**internal**) or via the network between different CPUs (**external**).





- COM provides a **signal-based API** (in the specification unfortunately *signals* are called *messages*) and hides the details of packing the signals into PDUs and bus frames completely. So for the application, internal and external communication look the same.
- All messages must be predefined (data length, sending and receiving tasks) during system configuration. For external communication, several signals can be combined into one PDU. If a PDU does not fit into a single bus frame, a transport protocol, e.g. ISO TP, will be used.
- A message can use one of two **buffering strategies**:
  - **Unqueued**: The interaction layer provides a single buffer for the message. If a new message is sent, the old content is overwritten. The buffer can be read any time, reading will always return the last message written (last-is-best semantics).
  - **Queued**: The interaction layer provides a queue (FIFO) of predefined length. A new message is inserted at the end of the queue. If the queue is full, the new message is lost. When reading, the first, i.e. oldest message in the queue, is read and removed from the buffer, i.e. each message can only be read once. If the queue is empty, reading fails. If a message has more than one receiver, the interaction layer provides separate queues.
- **Notifications**: The interaction layer can notify the sender or receiver task by one of the following mechanisms, when a message has been sent or received:
  - **Flag**s: The interaction layer sets a flag, which must be polled by the task.
  - **Event**: The interaction layer sets an OS event. The task can wait for this event. By this mechanism, messages can also be used to synchronize tasks via sending messages.
  - **Task activation**: The interaction layer can activate an OS task.
  - **Callback routine**: The interaction layer can call an application provided function.

- **Deadline monitoring** for external communication: COM can detect, when a message cannot be transmitted or is not periodically received within a specified maximum time.
- **Transmission strategy** for external communication: Bus messages will either be sent as soon as possible (*Direct Transmission*) after **`SendMessage()`** or with a fixed period (*Periodical Transmission*) independent of the API calls.
- **Message filtering**: Messages can be filtered, i.e. a message will only be sent or received, if its value did or did not change, lies inside or outside of specified limits, ...

COM API function calls (selection)

API call	
Starting and Stopping Communication	
`StartCom(comApplicationMode)`	Start communication, called once. Like OS, COM allows to configure various sets of messages for different operating conditions, called communication application modes.
`StopCom(…)`	Stop communication.
Sending and Receiving Data	
`SendMessage(msgId, *data)` **`SendDynamicMessage(msgId, *data, length)`**	Send a message (with predefined fixed length or with variable length up to a fixed maximum). Sending occurs asynchronously, i.e. the function copies data into an interaction layer buffer and returns immediately. The calling task does not have direct control, when sending actually occurs (but can be notified by one of the notificaion mechanisms). Function may fail with *queued* messages, if the queue is full.
`ReceiveMessage(msgId, *data)` **`ReceiveDynamicMessage(msgId, *data, length)`**	Read a message (with predefined fixed length or with variable length up to a fixed maximum). Reading is non-blocking, i.e. the function returns immediately. If the message is *unqueued*, the latest received data is copied into a buffer provided by the calling tasks. If the message is *queued*, the first entry of the receiver queue is copied into a buffer provided by the calling task and the removed from the queue. Function may fail with *queued* messages, if the queue is empty.

Message Status	
`ReadFlag...(), ResetFlag...()`	Poll the status of a message (message has been transmitted, message has been received, error occurred), if Flag has been configured as notification mechanism for a message.
`GetMessageStatus(msgId)`	Returns status of a message queue (full, empty).

AUTOSAR COM extends the original OSEK COM concepts considerably and encapsulates COM via the AUTOSAR RTE (Run Time environment) layer, so that the application layer software can communicate between SWCs inside an ECU and (via bus systems) between different ECUs using the same RTE functions.

Additional Information:
Advanced Mobility

Hochschule Esslingen
University of Applied Sciences

Propulsion Concepts

AM.1 Internal Combustion Engines (ICE) 2
AM.2 Alternative Fuels for ICE 6
AM.3 Battery Electric Vehicles (BEV) 7
AM.4 Fuel Cell Electric Vehicles (FCEV) 12
AM.5 Hybrid (HEV) and Plug-In Hybrid (PHEV) Vehicles 14
AM.6 Minimum Required Energy for Driving 18

Automated Driving

AM.7 Overview 22
AM.8 Advanced Driver Assistance Systems (ADAS) 27

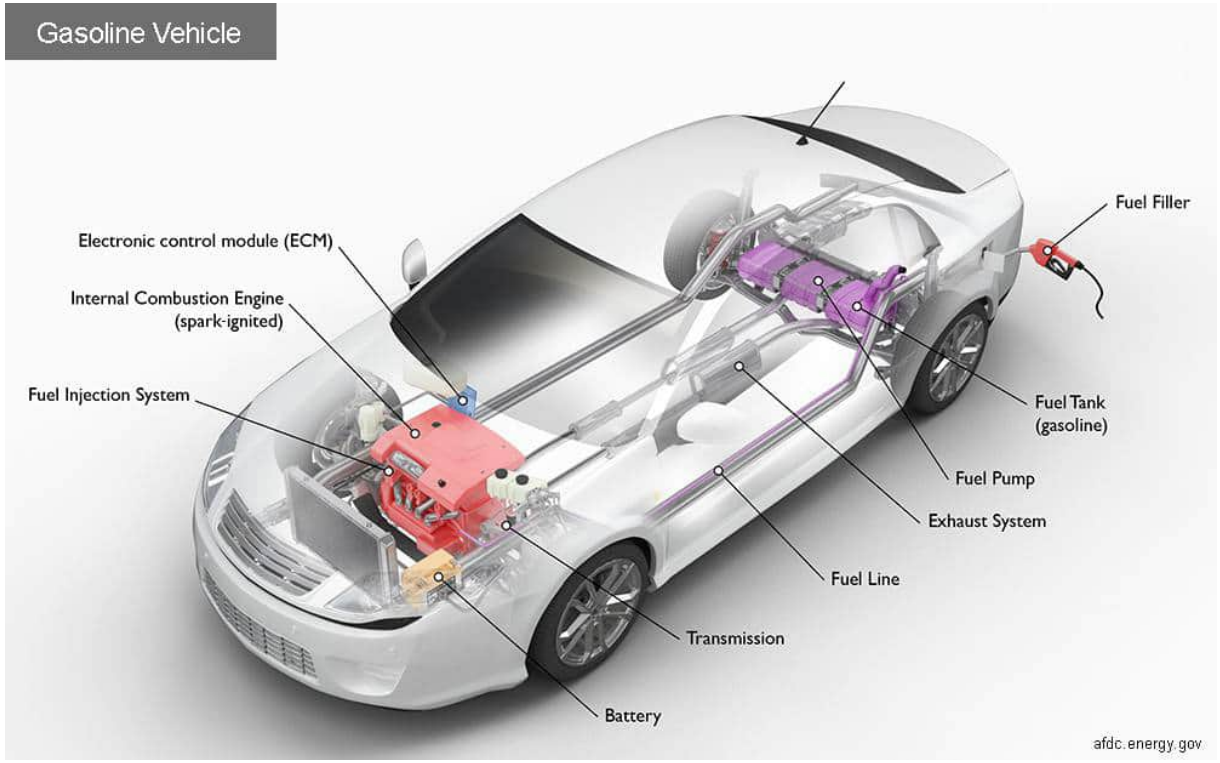
Note: This chapter is more a collection of keywords and basic concepts than a full manuscript with detailed explanations. Descriptions concentrate on first principles and may be overly simplified.

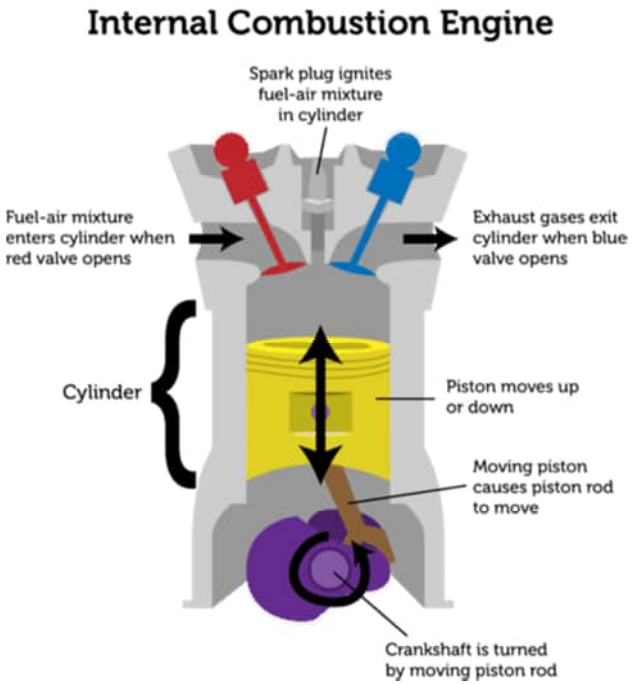
Propulsion Concepts

AM.1 Internal Combustion Engines (ICE)

- Gasoline spark ignition] [Indirect Injection (IDI) (homogenous air fuel mixture, most Gasoline engines)
- Diesel self ignition] [Direct Injection (DI) (non-homogenous air fuel mixture, most Diesel engines)

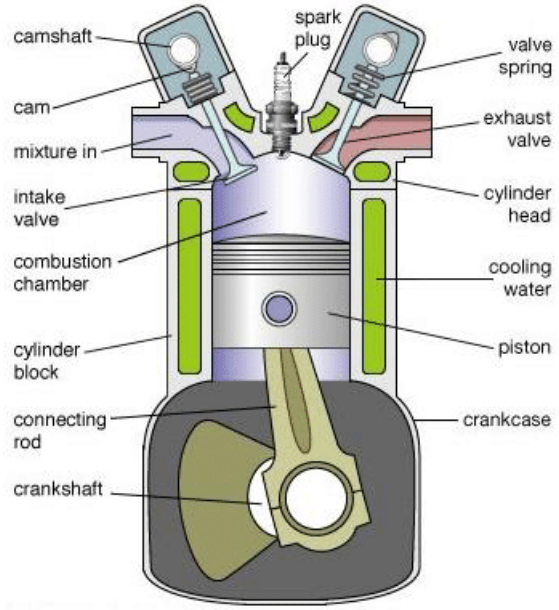
Diesel combustion more efficient (power vs. fuel consumption) than gasoline, DI more efficient than IDI.





Operating Principle

Indirect Injection (IDI) Gasoline Internal Combustion Engines (ICE) use a **4 stroke principle**:



- **Intake stroke**: Piston moves down with intake valve open and sucks in a homogenous air-gasoline mixture, optionally supported by a turbo charger to increase air (boost) pressure.
- **Compression stroke**: Piston moves up with all valves closed, thus air is compressed.
- **Combustion stroke**: Close to the top dead center (TDC) position of the piston the air-fuel mixture is ignited via a spark plug. The burning air fuel mixture expands and drives down the piston generating torque.
- **Exhaust stroke**: The piston moves up again with outlet valve open feeding the exhaust gas to the catalyst/exhaust gas pipe (not shown).

➔ Only 1 out of 4 strokes creates torque. To smooth the crankshaft motion, engines have multiple cylinders to shorten time between ignitions.

The crankshaft transmits the torque, while the camshaft, which rotates at half the camshaft speed, controls valve opening.

Because compression does require some minimum speed (= idle speed) and creates high forces limiting max speed, ICEs require a clutch and a transmission with several gears to operate over a large vehicle speed range down to vehicle stand still.

Direct Injection (**DI**) engines inject the fuel under high pressure (>100 bar @ Gasoline DI, >1000 bar @ Diesel DI) directly into the combustion chamber at the end of the compression stroke and during the combustion, while IDI engines inject with low pressure (~ 5 … 10 bar @ gasoline IDI) into the intake manifold.

Diesel engines are self-ignited (no spark plug/ignition coil). Due to higher compression, diesel fuel can be injected into the compressed hot air and ignites directly, but creates bigger forces → lower max speed.

DI leads to ~10 … 20% better fuel efficiency than IDI and Diesel is another ~10 …30% more efficient than Gasoline, but because of non-homogenous air-fuel mixture worse emissions (especially soot/particulate matters PM and NO_x).

Common Characteristics

- 4 stroke principle: intake – compression – combustion – exhaust
  only one stroke generates torque → engine speed oscillates, smoothing via flywheel
  → minimum ("idle") speed required (800 … 1000 rpm)
  → multiple phase shifted cylinders, clutch + gears
  no recuperation of kinetic energy, moderate efficiency ($\eta \leq 40\%$ fuel to kinetic energy)
- Engine power $P = M \cdot n$ with n … engine speed
  Gasoline engines allow higher engine speeds than same size diesel engines
- Engine torque $M \sim m_{Fuel} \sim \frac{m_{Air}}{\lambda} \sim \frac{V_{cyl} \cdot p_{boost} \cdot \alpha}{\lambda}$

with

m_{Fuel}	…	fuel mass
m_{Air}	…	air mass ("quantity")
$\lambda = \frac{m_{Air}}{m_{Fuel}}$	…	air-fuel ratio ("quality") $\lambda \approx 1$ const for gasoline engines, $\lambda > 1$ variable for diesel engines
V_{cyl}	…	cylinder volume
p_{boost}	…	air/boost pressure — variable with turbo charger
α	…	throttle opening $0 \leq \alpha \leq 1$ variable for gasoline engines, $\alpha \approx 1$ const for diesel engines

- Torque control
  - Spark induced Gasoline engine combustion only works in a range 0.9 (rich) < λ < 1.1 (lean mixture) → torque M controlled via air mass with (nearly) constant λ ≈ 1

    Driver controls air mass via throttle. Injection system measures air mass and injects fuel quantity proportionally to keep λ ≈ 1.
  - Diesel engine combustion works over a wide range 1.2 < λ < 30 → torque M controlled via fuel mass with variable λ with (nearly) constant air mass (except for variable boost pressure).

    Driver controls fuel mass via injection system.
- Major emission components:

CO_2	carbon dioxide	→	proportional to fuel consumption Gasoline engine worse than diesel due to lower efficiency, DI better than IDI
CO	carbon monoxide	→	lack of oxygen during combustion under full load or excess EGR
HC	hydrocarbons	→	unburned or evaporated fuel components espec. gasoline
NO_x	nitrogen oxide	→	due to excess oxygen during combustion DI engines, especially Diesel engines, due to excess oxygen under low/medium load
PM	particulate matter	→	lack of oxygen during combustion under full load or excess EGR DI engines, especially Diesel, due to inhomogenous air fuel mixture

- Limitations

  Minimum engine speed see above, maximum engine speed due to mechanical forces

  Maximum torque due to mechanical forces **and** emissions → limited m_{Air} due to given cylinder volume and maximum boost pressure

AM.2 Alternative Fuels for ICE

- Liquid Natural Gas LNG (methane = normal natural gas)
- Compressed Natural Gas CNG (butane, propane = by-products of oil production)
- Bio-Fuels, e.g. Ethanol → convert plants, e.g. corn, into Ethanol
- Synthetic Fuels, e.g. convert "natural" CO_2 into fuel in a chemical production

Pros

- Storage and transport (tanks, pipelines) similar to classic fuels
- Classic ICE needs only minor redesign. Bi-fuel engines possible → allows gradual transition from gasoline/diesel to alternative fuels

Cons

- Combustion of these fuels still generates local emissions, but may have a smaller or even no emission footprint from well-to-well.

For more info see e.g. https://afdc.energy.gov .

AM.3 Battery Electric Vehicles (BEV)

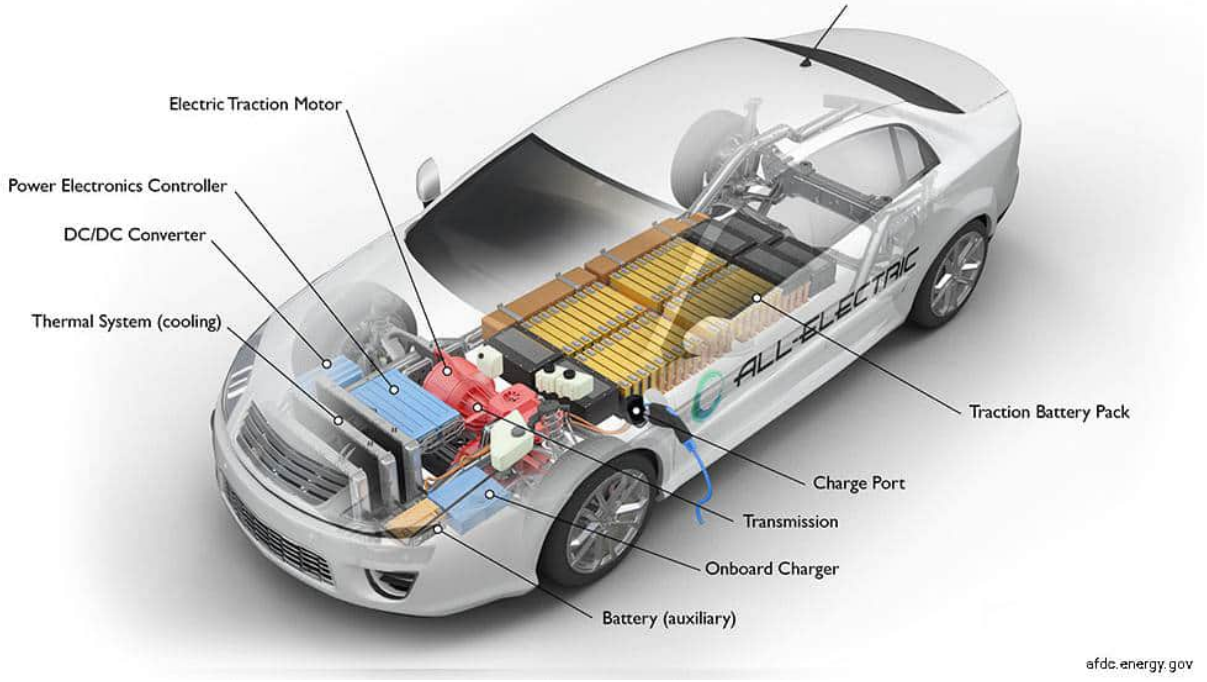
A major transition

- **Onboard**

  ICE → Electric motor + power electronics
  Fuel tank → Battery

- **Infrastructure**

  Gas pump → Battery charger
  Oil producers & refiners → Decarbonized energy production (nuclear, solar, wind, …)



Characteristics

- Electric motor: Brushless DC Motor (BDM) or Permanent Magnet Synchronous Motor (PSM) with rare earth magnets, e.g. Neodymium.

  Alternatives without PM: Asynchronous Motors (ASM) or Reluctance Motors (RM), cheaper, but less efficient.

- Power converter: DC to 3-phase AC power converter with variable voltage & frequency

  $u, f \sim n$ with n … motor speed, u … voltage, f … AC frequency

  $-u_{max} \leq u \leq +u_{max}$ with $u_{max} = U_{battery}$ limited by battery

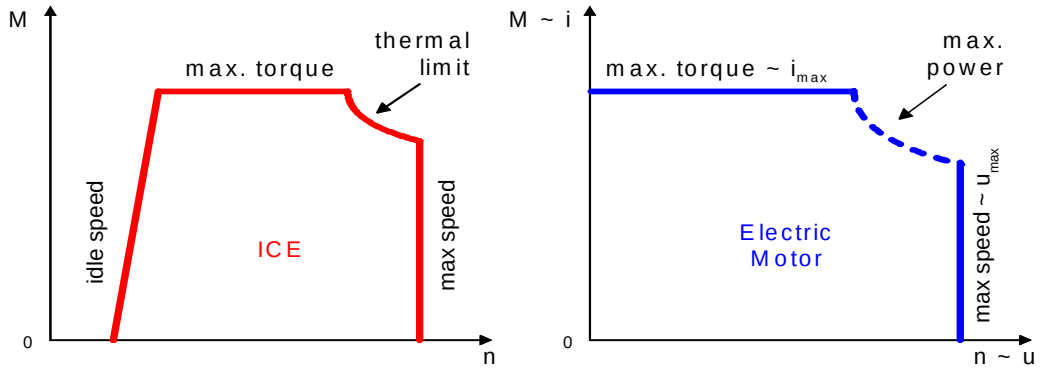
- Motor power $P = M \cdot n \sim u \cdot i$

- Motor torque $M \sim i$ with i … motor current

  $-i_{max} \leq i \leq +i_{max}$ → constant torque over (nearly) full speed range down to 0, i.e. no clutch required, fixed gear transmission

  motor can operate as generator (i<0), i.e. recuperation/ electrical braking

Speed-torque diagrams (simplified)



Battery technology

- **Killer factor for EVs:** Operating range, weight/size, cost, lifetime, reliability/safety, …
- High voltage battery/converter system required:

  Example: P = 100 kW @ $U_{battery}$ = 12V → $I = \frac{P}{U_{battery}}$ = 8333 A not feasible

  @ $U_{battery}$ = 400V → I = 250A ok

  Series (and parallel) connection of battery cells required → battery stack plus **Battery Management BEM**. BEM must load-balance all cells, monitor state of charge (SOC), state of health (aging) and battery temperature (**thermal management**, see below).

  High voltages (> 60V DC) safety critical

- Heavy batteries: Nickel Metal Hydride (NiMH) 60 … 120 Wh/kg → outdated

  Lithium Ion (LiIon) 100 … 250 Wh/kg → vs. gasoline ~10 kWh/kg

  Example: Tesla Model 3 Long Range Rear Wheel Drive: 15 kWh/100km
  → target operating range 600 km
  → required energy 90 kWh @ 200 Wh/kg
  → required battery size 450 kg (cells only) plus housing and BEM

  Worldwide Lithium resources limited, often in politically instable regions (think: oil reserves!). Environmental impact of production and recycling unclear.

Safety: LiIon is inflammable (high temperatures → cooling!), short circuits critical, e.g. in case of an accident. Difficult to extinguish by water or powder extinguisher.

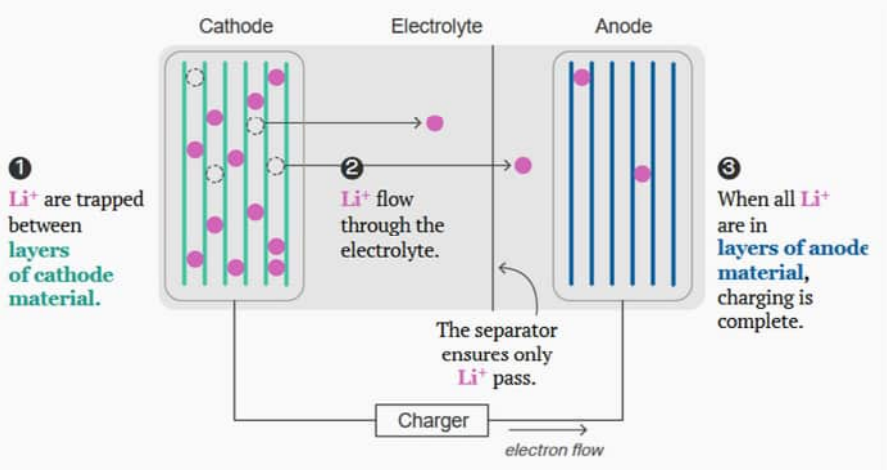
Battery capacity decreases with temperature (winter! – but typical energy consumption in winter = 1,5 x summer to heat the passenger cabin, driving lights etc.) and over lifetime ("aging" = calendar time and charge/discharge cycle limits, typical warranty periods 8 years, 160000 km with battery capacity > 60% of initial capacity)

Charging takes much longer than filling a fuel tank:

Example: 90 kWh battery → charge with 11 kW charger @ 90% efficiency → 9h

There are chargers available with up to 250 kW ("Tesla Supercharger"), but charging stresses the battery. The number of charging cycles limits the lifetime of a battery. The higher the charging current, the higher the battery stress → reduced battery lifetime.

Thermal management (heating/cooling) required, as cold or hot batteries tolerate only small charging currents.



Charging of a LiIon battery (Discharging works inverse)

typ. 3.6 V/cell (LFP 3,3V/cell)
Minus Pole: Graphite
Plus Pole:

- LCO Lithium Cobalt Oxide
  (200Wh/kg, 1000 cycles, consumer electronics)
- NMC Lithium Nickel Manganese Cobalt Oxide
  (220Wh/kg, 1500…2000 cycles, EVs)
- NCA Lithium Nickel Cobalt Aluminum
  (250Wh/kg, 500 cycles, Tesla)
- LFP Lithium Iron (Fe) Phosphate
  (120Wh/kg, >2000 cycles, no thermal runaway)

Source: https://qz.com/1588236, Wikipedia.de

New battery technologies

- Improved battery technologies (higher capacity, non-flammable, faster charging, …) to overcome the problems of LiIon batteries, e.g. nanowire, solid state, graphene, … still in research state.
- For more info see e.g. https://www.technologyreview.com/climate-change/battery-technology/

Charger Infrastructure

- Building up a charger infrastructure is critical for the success of EVs:
  - AC or DC charging
  - Connector type/cable between charger and car
  - Accounting system
  - Peak power requirements may exceed limits of mains power distribution network

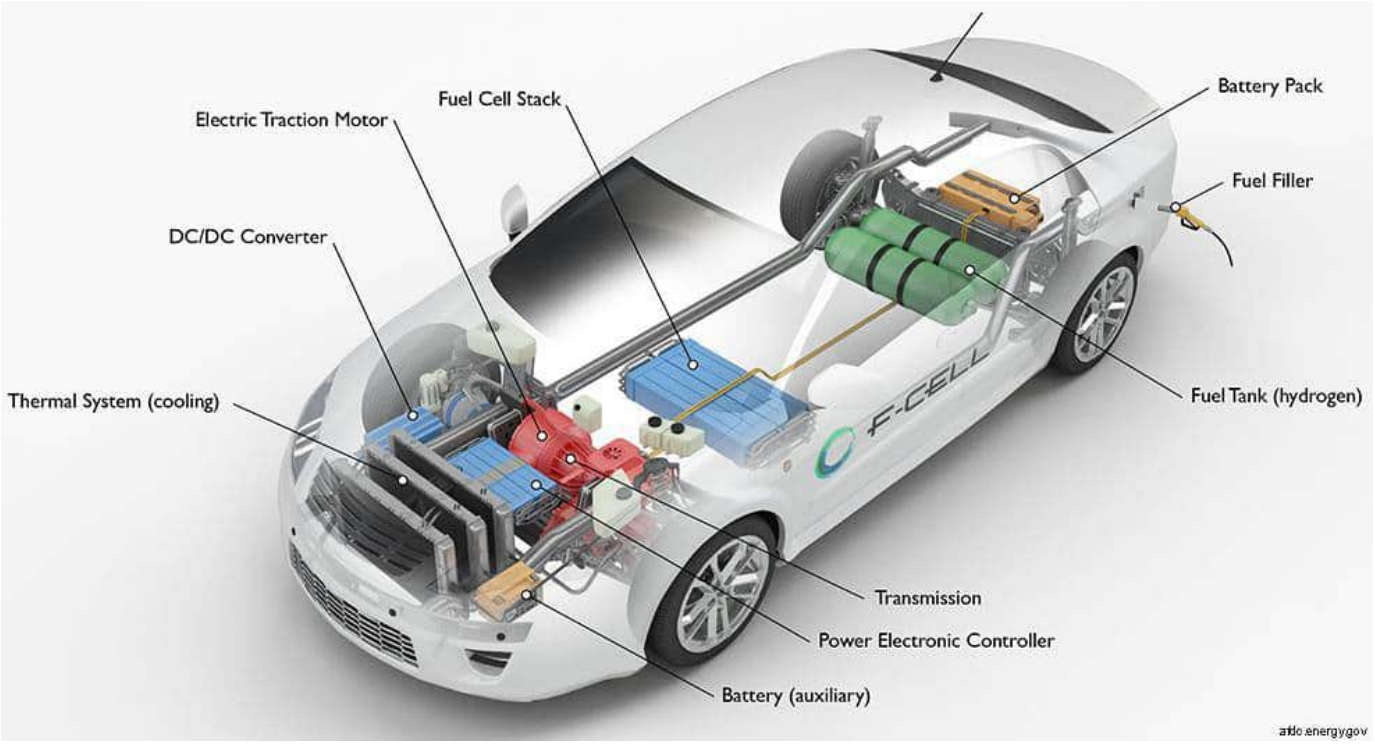
Clean electric energy generation?

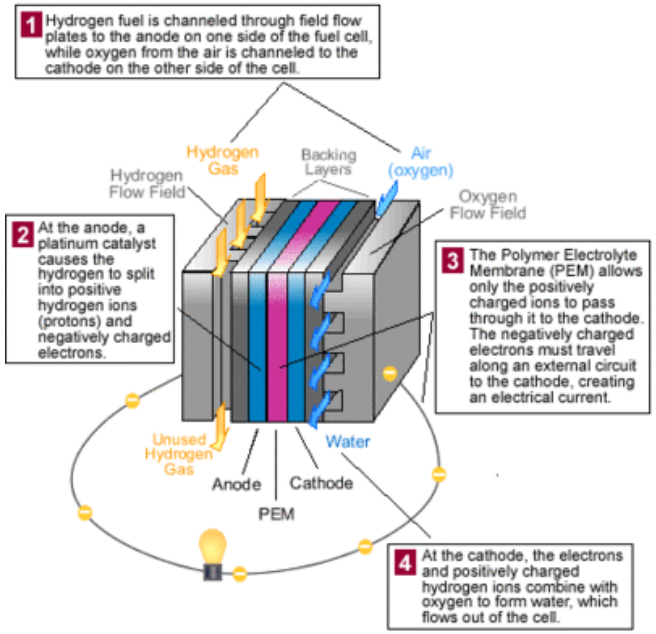
- Conversion from gas/oil to electric powered vehicles requires investments in electric power plants
  - Nuclear power
  - Solar power
  - Wind power
  - Coal, oil, natural gas …

  How many new power plants and how many new power links do we have to build?

AM.4 Fuel Cell Electric Vehicles (FCEV)

Rather than storing electrical energy in a large, heavy battery, generate it on the fly by a stack of **Fuel Cells**, which converts hydrogen H_2 (very high energy density 34kWh/kg!) into electrical power.



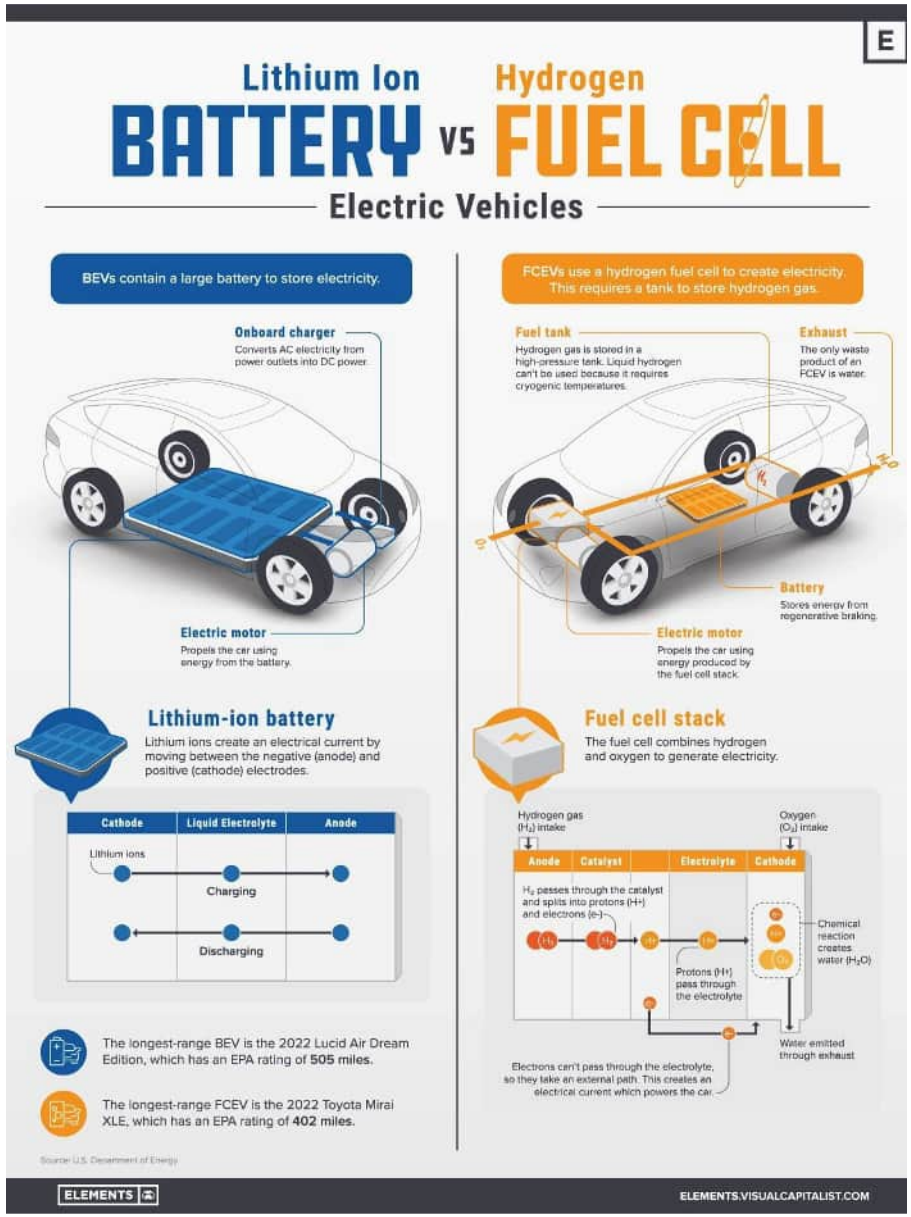


Source: https://en.wikipedia.org/wiki/Fuel_cell

For more info see e.g. http://www.fchea.org/fuelcells

Major problems are:

- Cost and reliability of the fuel cell stack
- Storage of hydrogen: Hydrogen storage systems leak, hydrogen destroys tank material.
- Hydrogen production (requires a lot of primary energy!) and H_2 distribution network



Sample FCEV vehicles:

- Toyota Mirai (in production):
  https://www.toyota.com/mirai/
- Hyundai Nexo (in production):
  https://www.hyundaimotorgroup.com/tech/821
- BMW iX5 Hydrogen (prototype):
  https://www.bmwblog.com/2022/03/10/bmw-ix5-hydrogen-review/
- Daimler Truck Fuel Cells (prototypes):
  https://media.daimlertruck.com/marsMediaSite/en/instance/ko.xhtml?oid=9265782

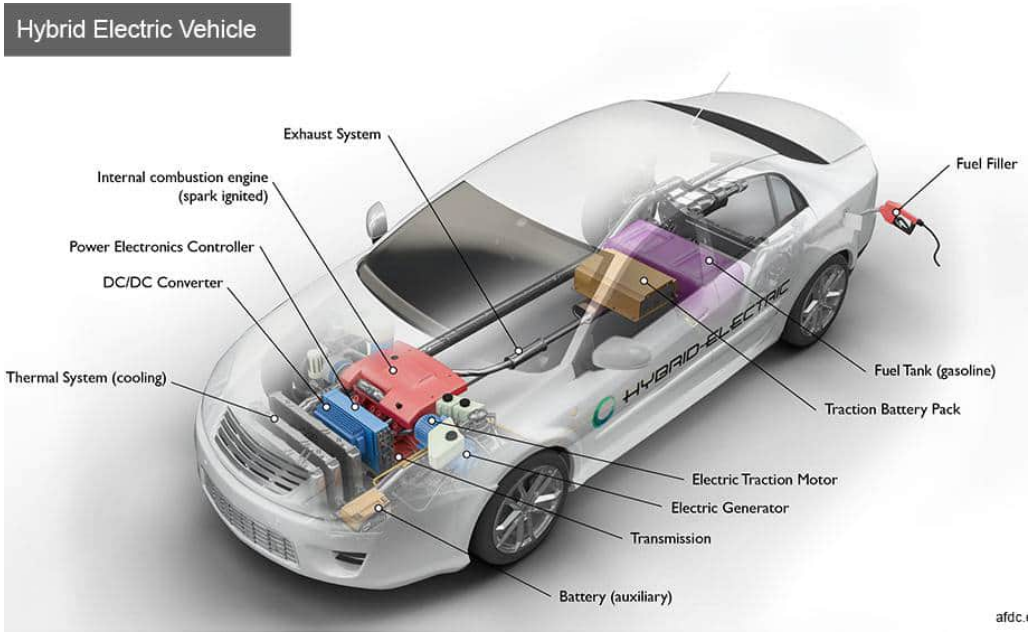
Source:
https://www.visualcapitalist.com/visualized-battery-vs-hydrogen-fuel-cell/

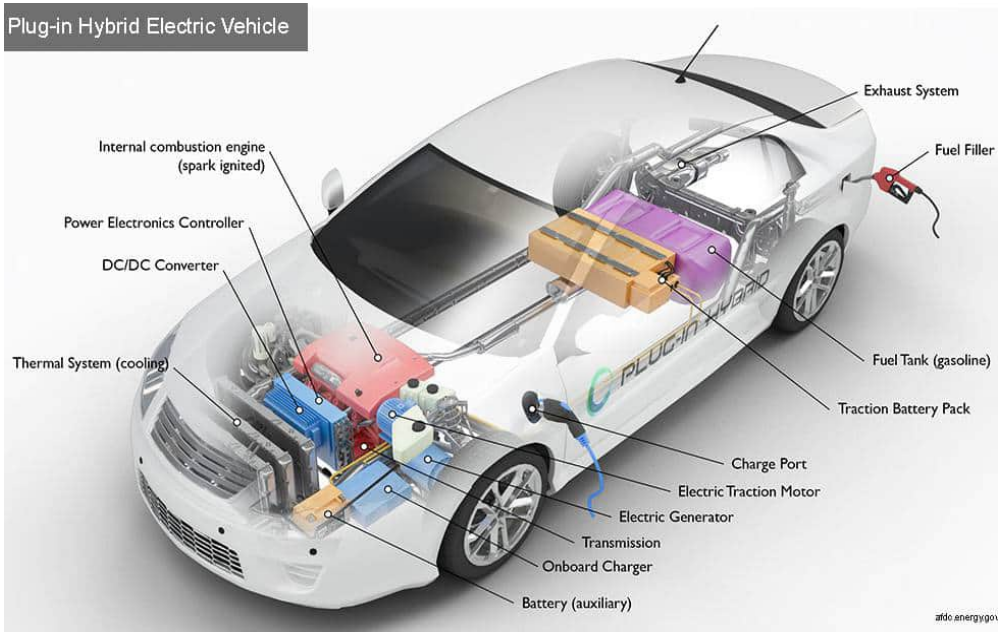
AM.5 Hybrid (HEV) and Plug-In Hybrid (PHEV) Vehicles

Hybrid: Combining a classical ICE with an electric motor (EM) and a medium sized battery

- HEV are self-sufficient, i.e. the battery is charged via the ICE and by recuperation.
- PHEVs additionally charge the battery by an on/off-board charger via the public mains.

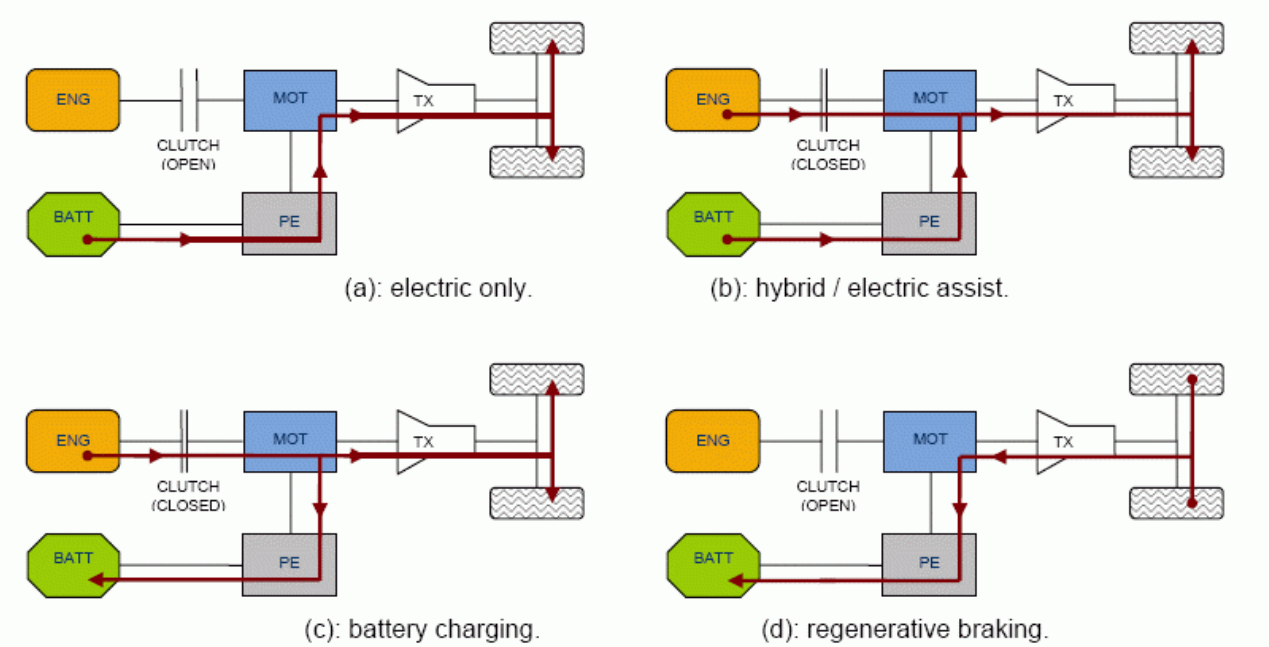
Hybrids are a bridge technology between ICE powered cars and pure EVs.





Typical operating modes of HEVs and PHEVs are

- ICE mode: Drive with the ICE alone
- Boost (hybrid/assist) mode: Drive with a combination of ICE and EM
- EV mode: Drive with the EM alone (and re-charge battery)
- Recuperation: Brake with the EM in generator mode



The transition between the various modes occurs automatically.

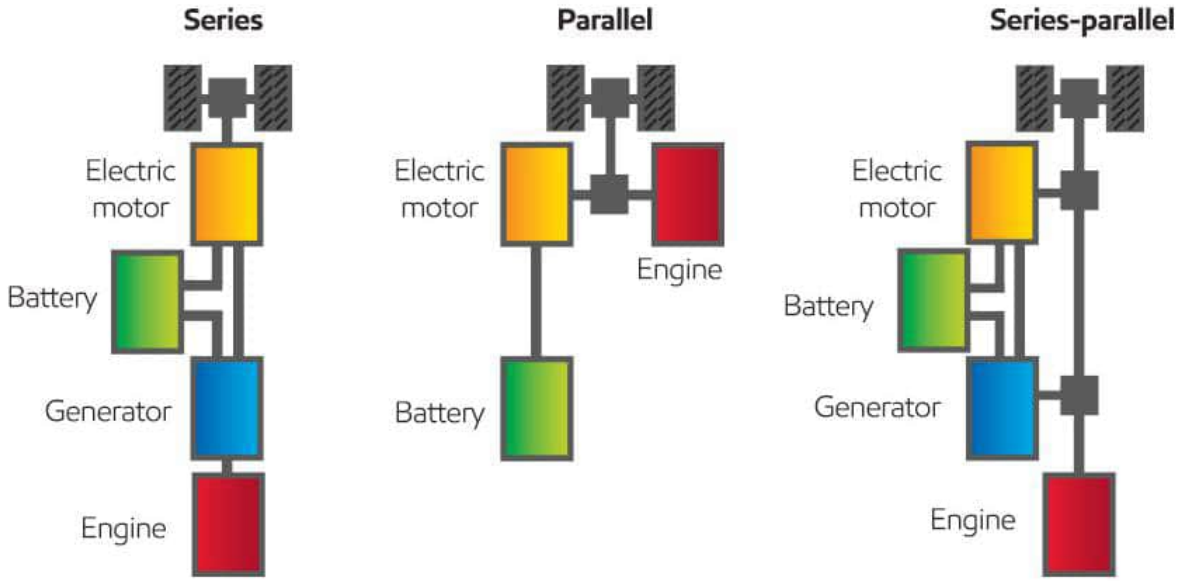
ENG: combustion engine ICE
MOT: electric motor EM
TX: transmission
BATT: battery
PE: power electronics

The picture shows a parallel ICE-EM architecture.

Limits:

- HEV typically have a small EM (≤ 50 kW) and battery (≤ 50 kg). Pure EV operation is restricted to short distances (≤ 2 … 5 km) and small speeds (≤ 120 km/h).
- PHEVs have a larger EM (≤ 100 kW) and battery (≤ 150 kg). Pure EV distances are 40 … 60 km.

Architecture



Note: Transmission and clutches not shown

- The **parallel architecture** allows a pure ICE and a pure EV mode, but requires a mechanical coupling (and synchronization) of both motors for boost mode. The EM is often used as flywheel combined with the clutch and transmission.
- The **series architecture** uses two electrical motors. The ICE drives the first EM in generator mode. The generator loads the battery and/or provides electrical energy for the second EM, which actually drives the car. In the series architecture, the ICE can work in its most efficient operating range independent from the cars driving conditions.

  Note: Some BEV use a series architecture with a small ICE to charge the battery as a Range Extender (REX).

Advantages

- Proven gasoline/diesel technology and infrastructure
- Fuel-savings by recuperation and limited EV operation
- Cheaper than BEV/FCEV (as long as batteries/fuel cells are very expensive)

Disadvantages

- Local emissions, when ICE is running
- Additional cost and weight vs. classical ICE powered vehicle.

  Mild Hybrid: To reduce costs, some OEMs use a small, low voltage battery (e.g. 48 V) and a small EM, which allows boost and recuperation mode, but no pure EV mode.

Selected cars with alternative drives

HEV / PHEV	Toyota Prius (series/parallel architecture) https://www.toyota.com/prius/ Hyundai Ioniq, Kia Niro (parallel architecture) https://www.hyundaiusa.com/ioniq-hybrid/ Honda CR-V (series architecture) https://www.honda.co.uk/cars/new/cr-v/overview.html
BEV	Tesla S, Tesla Model 3 https://www.tesla.com BMW i3 und i8 https://www.bmw.co.uk/en/all-models/bmw-i/i3/2019/at-a-glance.html
FCEV	Toyota Mirai https://www.toyota-europe.com/world-of-toyota/feel/environment/better-air/fuel-cell-vehicle Hyundai Nexo https://www.hyundai.co.uk/new-cars/nexo

AM.6 Minimum Required Energy for Driving

No matter which propulsion technology we use, all cars need a minimum amount of energy. We basically have 4 physical root causes:

- Friction caused by **rolling resistance of wheels**

  $$F_{roll} = \mu_{roll} \cdot G = \mu_{roll} \cdot m\ g$$

  with $G = m \cdot g$ … weight of the vehicle, m … vehicle mass, g = 9.81 m/s²

  μ_{roll} … friction coefficient, for passenger car tires on asphalt road $\mu_{roll} \approx 0.01 \ldots 0.015$

  (Source: https://en.wikipedia.org/wiki/Rolling_resistance)

  Assuming constant speed v, the required energy E_{roll} to drive distance $\Delta x = v \cdot t$ is

  $$E_{roll} = F_{roll} \cdot v \cdot t = F_{roll} \cdot \Delta x = \mu_{roll} \cdot G \cdot \Delta x = \mu_{roll} \cdot m\ g \cdot \Delta x \qquad (1)$$

  This is a best-case estimate. Axle, differential gear and transmission friction add on top.

- **Air resistance (drag)** $F_{air} = \frac{1}{2}\, c_w\, \rho\, A\, v^2$

  with $\rho = 1.2041$ kg/m³ … air density @ 20°C / 100 kPa,

  A … vehicle front cross section area

  c_w … air drag coefficient (Source: https://en.wikipedia.org/wiki/Drag_(physics)

  Assuming constant speed v, the required energy E_{air} to drive distance $\Delta x = v \cdot t$ is

  $$E_{air} = F_{air} \cdot v \cdot t = \frac{1}{2}\, c_w\, \rho\, A\, v^2 \cdot \Delta x \qquad (2)$$

- Kinetic Energy for **acceleration** and de-acceleration

  When a car accelerates from speed v0 to speed v, the required energy is

  $$E_{acc} = ½\ m\ (v^2 - v_0^2) \qquad (3)$$

  Cars with recuperation can partly recover kinetic energy when braking from speed v to speed v_1 $\quad E_{rec} = \eta_{rec} \cdot ½\ m\ (v^2 - v_1^2)$

  Where η_{rec} is the efficiency of the recuperation.

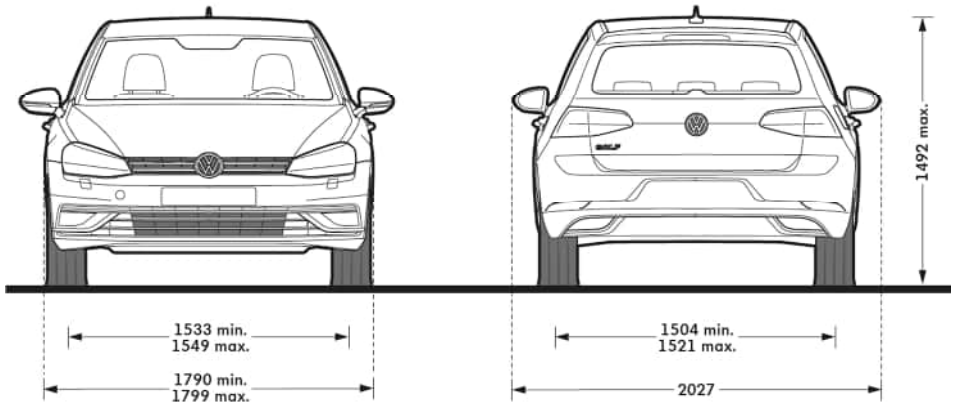
- When **driving uphill**, the required energy to overcome a height difference Δh is

  $$E_{uphill} = G \cdot \Delta h = m\ g \cdot \Delta h \qquad (4)$$

  When driving downhill, similar to recuperation, part of this energy can be recovered.

In this chapter we do neglect, that in a real car the energy for air condition/heating and supplying all electric equipment adds on top of these basic requirements.

Example: VW Golf model year 2020, 1.5 TSI, 110 kW, DCT transmission, m = 1340 kg



Front cross section	$A = 2.5\ m^2$
Air drag coefficient	$c_w = 0.27$

Driving distance $\Delta x = 100$ km with v = 100 km/h requires approx. 13 kWh:

(eq. 1): $\mathbf{E_{roll}} = \mu_{roll} \cdot m\ g \cdot \Delta x = 0.012 \cdot 1340\ kg \cdot 9.81\ m/s^2 \cdot 10^5\ m = 15.8 \cdot 10^6\ Ws =$ **4.4 kWh**

(eq. 2): $\mathbf{E_{air}} = ½\ c_w\ \rho\ A\ v^2 \cdot \Delta x = ½ \cdot 0.27 \cdot 1.204\ kg/m^3 \cdot 2.5\ m^2 \cdot (10^5\ m/3600\ s)^2 \cdot 10^5\ m$
$=$ **8.7 kWh**

(same distance @150km/h → 19.6 kWh , at 50 km/h → 2.2 kWh)

(eq. 3): Acceleration 0 to 100 km/h

$E_{acc} = ½\ m\ v^2 = ½ \cdot 1340\ kg \cdot (10^5\ m / 3600\ s)^2 = 0.14$ kWh Important in city driving only partly compensated by recuperation.

(eq. 4): Uphill driving $\Delta h = 100$m

$E_{uphill} = G \cdot \Delta h = m\ g \cdot \Delta h = 1340\ kg \cdot 9.81\ m/s^2 \cdot 100m = 0.37$ kWh (recuperation see above)

Conversion of required energy into equivalent fuel consumption and emissions:

1 ltr gasoline: **specific energy 8.5 kWh/ltr** **gasoline ICE efficiency ~ 0.4**

CO_2 equivalent: 2.39 kg CO_2/ltr

(Sources: en.wikipedia.org/wiki/Energy_density, ecoscore.be/en/info/ecoscore/co2)

Example: 13 kWh / (0.4 · 8.5 kWh/l) = 3.8 ltr gasoline → 91 g CO_2/km

(VW Golf model year 2015 1.5 TSI DCT spec: 6.0 ltr/100 km city, 4.0 ltr/100km / 95 g CO_2/km extra-urban)

Conversion of required energy into equivalent Li-Ion battery size:

LiIon battery: **specific energy 0.25 kWh/kg** **charging & EM efficiency 0.85**

Example: 13 kWh / (0.85 · 0.25 kWh/kg) = 61 kg

(Tesla Model 3 Long Range RWD spec: 16 kW/100 km, range 635 km with 480 kg battery
→16 kW · 635 km/100km / 480 kg = 0,21 kWh/kg = 0,85 · 0,25 kWh/kg)

VW Golf: Driving 100 km at constant speed v (no hills, no acceleration/de-acceleration)

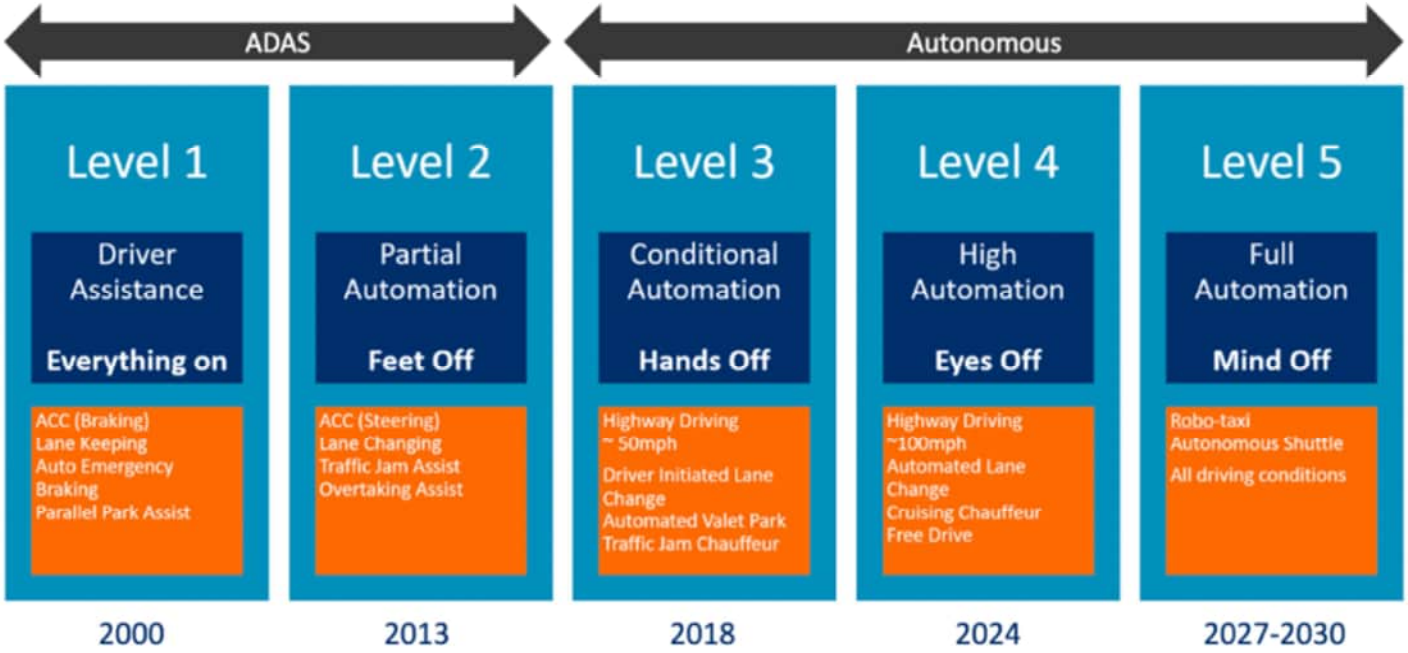
Speed v	Required energy $E_{roll} + E_{air}$	Required Gasoline consumption	Equivalent CO_2 emissions	Equivalent Li-Ion battery *1
100 km/h	13 kWh	3.8 ltr	91 g/km	61 kg
150 km/h	24 kWh	7.1 ltr	170 g/km	113 kg
50 km/h	7 kWh	2.1 ltr	50 g/km	33 kg

*1: Assuming, that a Golf with electric motor and battery is not heavier than with ICE and/or that higher weight can be compensated by tires with lower rolling resistance!

Automated Driving

AM.7 Overview

To reduce the impact of human errors and the number of accidents, **Advanced Driver Assistance Systems ADAS** gradually take responsibility from the driver. The steps, which finally will lead to fully **automated** (=**autonomous**) **driving**, according to SAE J3016 are:



Source: www.sae.org/autodrive

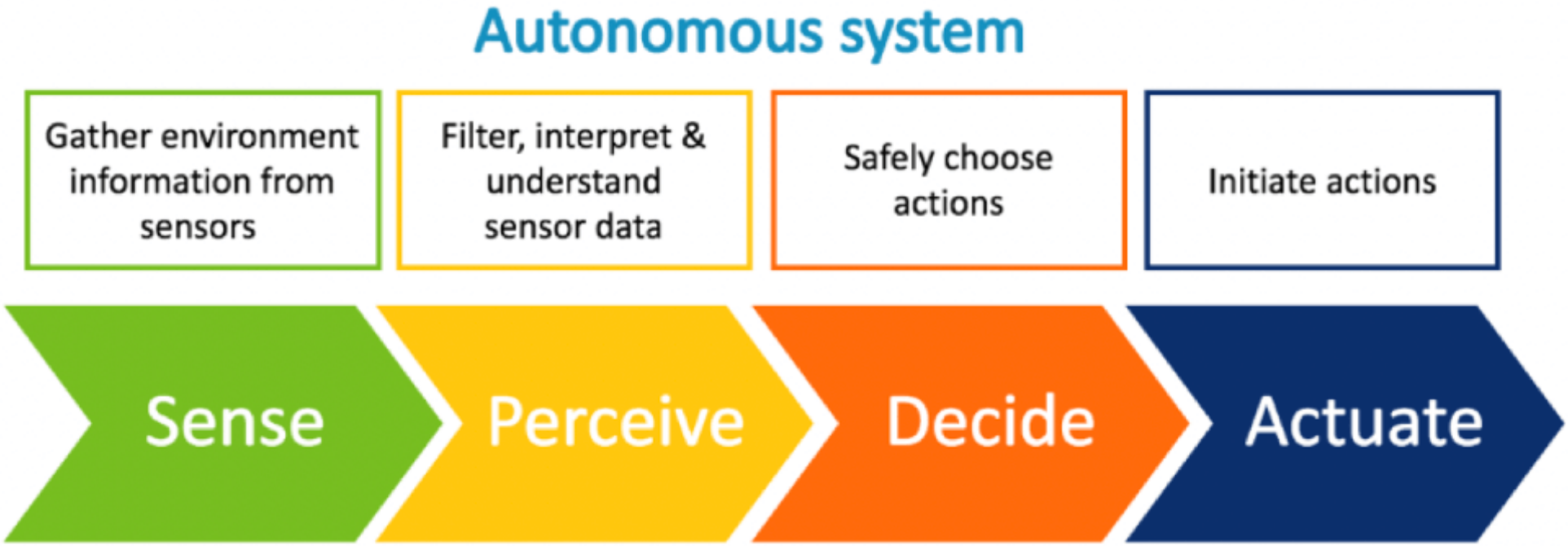
Level 1...3 is typically implemented with one channel with full functionality plus a monitoring channel with graceful degradation till the driver takes over (**Fail Safe/Fail Stop**).

Level 4...5 must be implemented as full dual redundant system (DMR) with both channels providing the full functionality (**Fail Operational**)

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

SAE J3016

All tasks, which are performed by the driver, must be implemented by hard- and software:



Source: www.arm.com

Today, the most evolved stage is the **Actuator** stage:

- **Acceleration** is controlled electronically via the electronic throttle and injection system (for ICE) or power electronics (for EV) → **Engine Management**
- **Deacceleration** is controlled with an electric pump generating hydraulic pressure to brake even without driver intervention → **ESP** (and recuperation) **system**
- **Steering** is possible via an electric motor, which originally was only a substitute for the pneumatic steering support → **Electric Power Steering**

Sensing the environment around the vehicle is evolving, but it is still difficult to fully substitute the eyes (and ears) of human driver:

- Front and rear **Radar Sensors** (→ Adaptive Cruise Control, Rear Traffic Detection) can detect hard objects in front or in the rear of a car. Radar sensors can measure the distance (≈ 5 … 250 m) and relative speed of the object. However, object classification despite a lot of signal processing does not work well. Radar sensors have problems with soft object (humans, animals) and cannot detect road lanes and curves.
- **Ultrasonic sensors** (→ park assistance) work over small distances (≤ 5m) only. Other problems are similar to radar sensors.
- Multiple **Video Cameras** can provide a surround view of the car. Object detection and classification possible with costly AI (Artificial Intelligence) signal processing. Problems to measure the distance and speed of objects.

  Major problems under bad visibility conditions (night, fog, rain, snow, …), similar/worse than human eyes. To overcome some of these problems, special technologies needed: Infrared cameras (→ Night View), **Laser Detection and Ranging** (**LIDAR**) scanners.
- **Microphones** with appropriate signal processing would allow to detect sound, e.g. the siren of a police car, the horn of another car or the voices of children, but don't seem to find much use in today's autonomous driving research projects. Microphones however are important for the coming **natural speech interfaces** between drivers and cars.
- Car/Vehicle to Car, Car to Infrastructure **C2X/V2X communication**, e.g. via WiFi or LTE 5G+, to share information about road and traffic conditions is technically promising, but requires infrastructure investment and rises security issues. Today only limited traffic info and location data (**GPS**) for medium/long range navigation is available.

Perception

- Typically, a static road map combined with GPS info is used to get a coarse **Localization** of the car.
- All the objects detected via combining the mass of raw sensor data (**Sensor Fusion**) are used to refine that map to get a clear picture of what is going on around the car (**Mapping**)
- However, despite a lot of **Artificial Intelligence** (Machine Learning, Deep Neural Networks, …) perception is still inferior to what the human brain does.

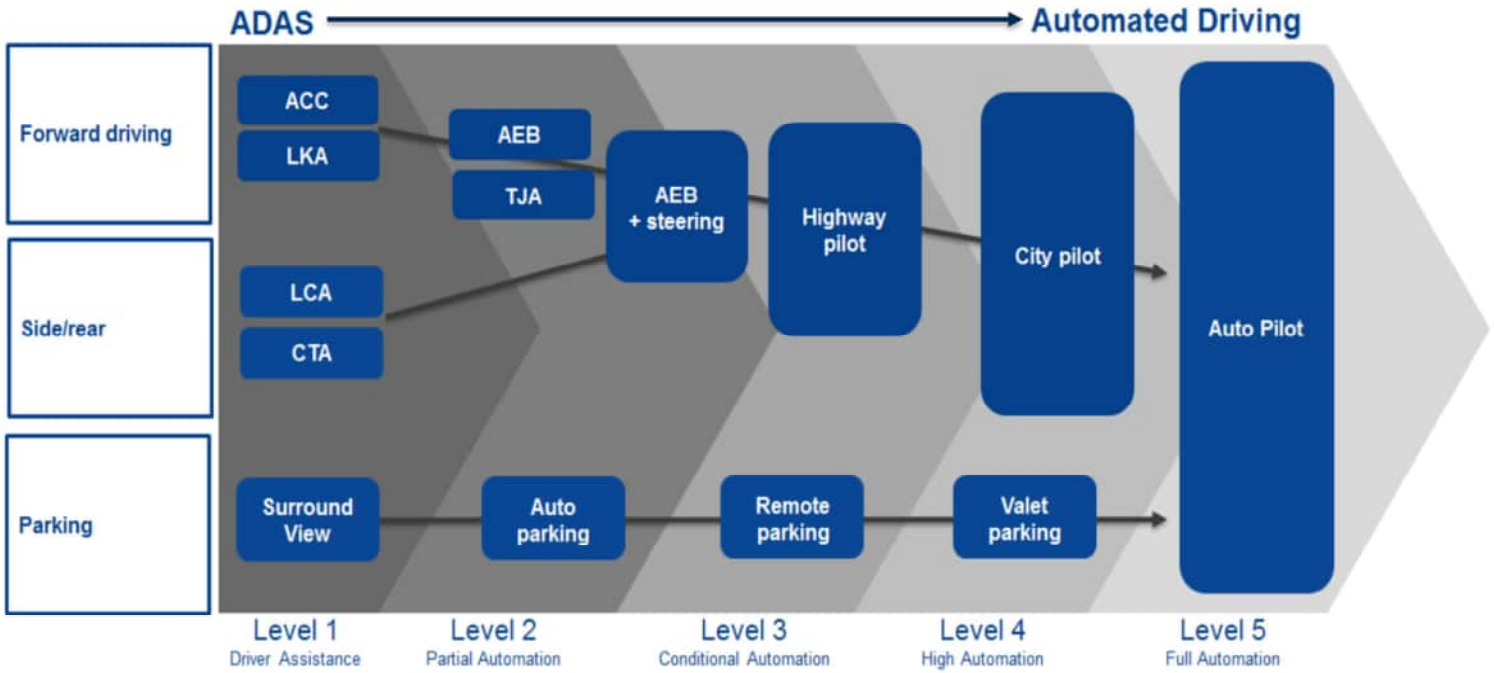
Decision Making

- While the static road map together with traffic information ensures state-of-the-art medium/long-range navigation, near-range navigation, i.e. **path/trajectory planning** taking into account the estimated trajectories of the surrounding objects, is an unresolved issue.

  Especially if the situation conflicts with or has no exact traffic rules, e.g. four vehicles nearing a crossing where no direction has a clear right of way.
- Again, this is a field, where advanced algorithms, eventually based on artificial intelligence need to be applied.

AM.8 Advanced Driver Assistance Systems (ADAS)

On the path towards full automation, many advanced driver assistance systems were introduced already:



Source: www.renasas.com

- **ACC Adaptive Cruise Control** controls vehicle speed and distance to the car in front. The newest **TJA Traffic Jam Assistants** even work in **stop and go** traffic situations.
- **AEB/EBS Automatic Emergency Brake** stops the car, when it approaches an obstacle and the driver does not react.
- **Traffic sign and signal detection systems** detect speed limits or red traffic lights.

- **LDW Lane Departure Warning** and **LKA Lane Keep Assistant** systems monitor traffic on the lanes next to the car and warn the driver if the car reaches the limits of its lane or event automatically corrects the car's direction via the **electric power steering EPS**.
- **CTA/RTA/BSD Cross / Rear Traffic / Blind Spot Detection Assistants** warn when an object is approaching the car's lane, e.g. when trying to reverse from a parking spot.
- **Highway pilot systems**, e.g. Tesla's Autopilot, combine ACC and LKA and allow full automation under well-defined (simple) driving scenarios, but the driver is still expected to take over immediately when the system runs into a problem.
- **Parking assistance systems**, which originally only warned about obstacles, now detect parking lots and issue hints how to steer, accelerate, brake or reverse. The newest systems can park the car automatically, some even when the driver is outside the car.
- . . .

A web site with some cool videos demonstrating these ADAS systems is mycardoeswhat.org

Legal issues with commercial impact - the largest obstacle to automated driving

While all technical issues will be resolved by scientific invention and engineering in the medium-term future, the most important question is: What if something goes wrong?

Even with the most advanced driver assistance systems today, **in case of a traffic accident**, **the driver,** a pedestrian or the owner of the car, i.e. a single or small number of human beings **will be blamed** → **Human error**. This is an accepted risk for everybody being part of our traffic system.

When we have an accident with automated driving, **the OEM** and/or his suppliers, i.e. an anonymous company (or the managers or engineering teams?) **will have to take over** this **responsibility**. Because faults in these systems must be considered to be systematically rather than bad luck, imagine the billion dollar claims of large law firms against the OEMs …

And imagine you being the victim of such an accident and trying to go to court against Daimler, VW, Toyota, General Motors, Ford, Hyundai, BYD, …

Additional Information:
Diagnostics and Calibration

Hochschule Esslingen
University of Applied Sciences

D.1 Overview 2

D.2 On Board Diagnosis OBD – Typical Monitoring Functions 3

D.3 Diagnostic Communication Protocols 7

D.4 Additional Info for OBD on CAN 8

D.5 UDS on CAN 10

D.6 World-Wide-Harmonized OBD 16

D.7 Diagnostics on IP 17

D.8 Development of Diagnostic Systems: ASAM D-Servers and ODX 18

D.9 Calibration Support: ASAM ASAP2, CCP and XCP 22

Further Reading 26

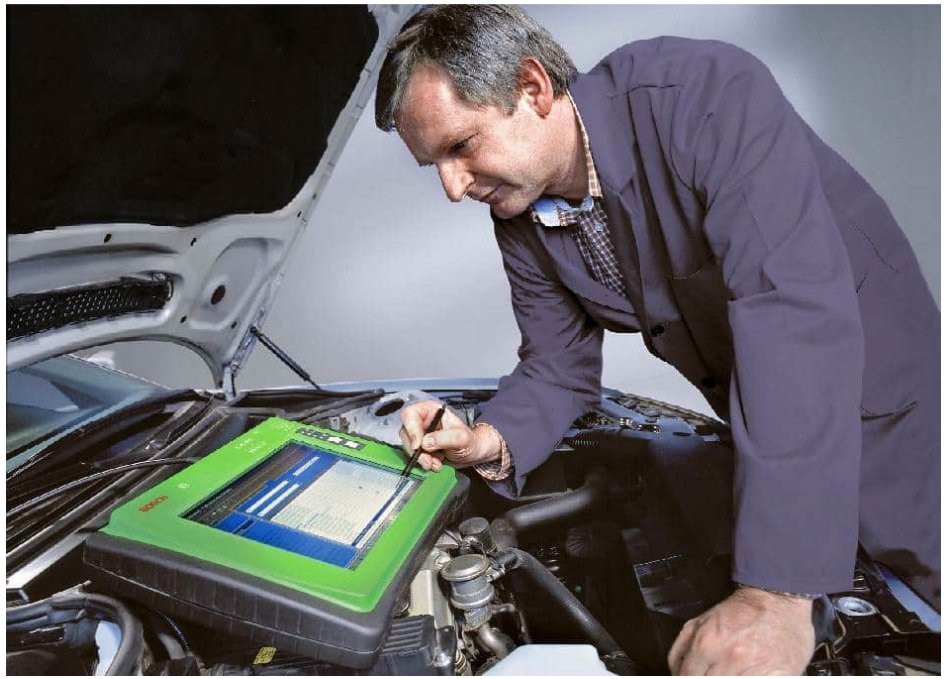
Dia.1 Overview

Purpose of Diagnostic Systems

- **Safety**: Detect failures to avoid safety critical situations (Functional Safety, see chap. 5), requirements defined by IEC 65108, ISO 26262 and country specific rules.
- **Emissions**: Detect failures to avoid excessive emissions, requirements defined by US, ECE or other country specific regulations. Typically, emission standards like EURO6/7 and diagnostic requirements for emission-related components like **OBD (On-Board-Diagnosis)** or EOBD (European variant of OBD) are issued in parallel.
- **Availability**: Detect failures and activate redundant functions to keep system available.
- **Maintainability**: Store failure information and help workshop staff to repair the system.
- **Manufacturability**: Similar to workshop use, diagnostic functions are used in production testing. Diagnostic interface also used for End-of-Line-programming and variant coding.

Diagnostic systems typically consist of several modules

ECU / Vehicle Components	Workshop Components
- Failure detection during normal operation and special test modes - Failure storage with **Diagnostic Trouble Codes (DTC)** and information about operating conditions (**Freeze Frame**) - Redundant functions in case of failures - Bus interface and communication protocol	- **Diagnostic tester**: Computer and software application to access **and** interpret DTCs, Freeze Frames and other ECU data - Trouble-shooting and repair procedure manuals to guide workshop staff - Access to the spare parts database and accounting systems



Source: Bosch

Diagnosis of emissions-related components defined very detailed by government regulations and public standard **OBD**:

- diagnosis functions
- communication interface, protocol and diagnosis data

For **Workshop diagnosis** only

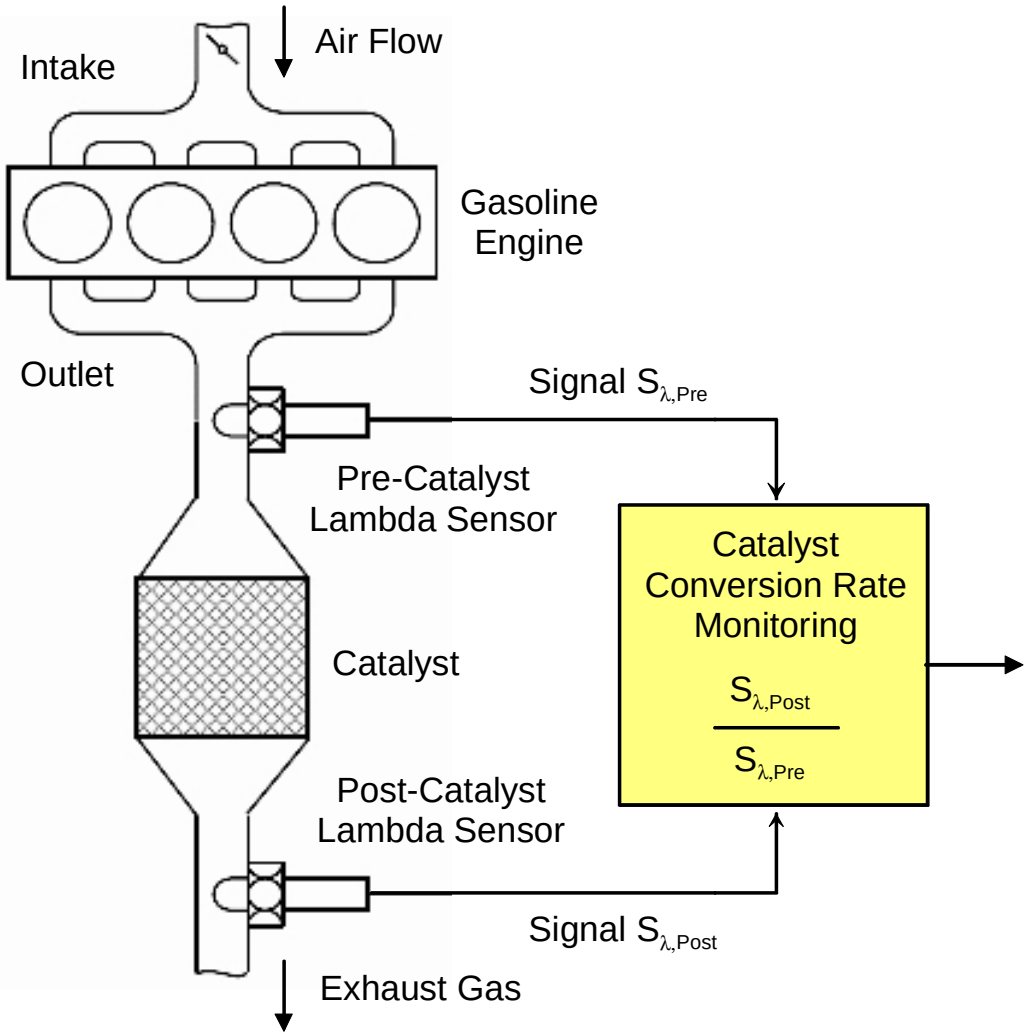
- communication interface and communication protocols

are covered by public standard **UDS**, but not the diagnostic functions and data.

D.2 On Board Diagnosis OBD – Typical Monitoring Functions

- All component failures, which cause an increase of the exhaust-gas emissions above a defined limit, must be detected.
- Continuous monitoring during normal operation is required, however, not all monitoring functions can work in all operating situations of a car (e.g. catalyst monitoring does not work during engine startup). Whether operating conditions did allow certain tests or not, is stored in so-called **Readiness** codes in the ECU memory.
- In case of a malfunction, failure information has to be stored in an OBD defined format and the driver has to be informed via the **Malfunction Indicator Lamp MIL** in the dashboard.

- **Example OBD-Function: Catalyst Monitoring – Passive Test**

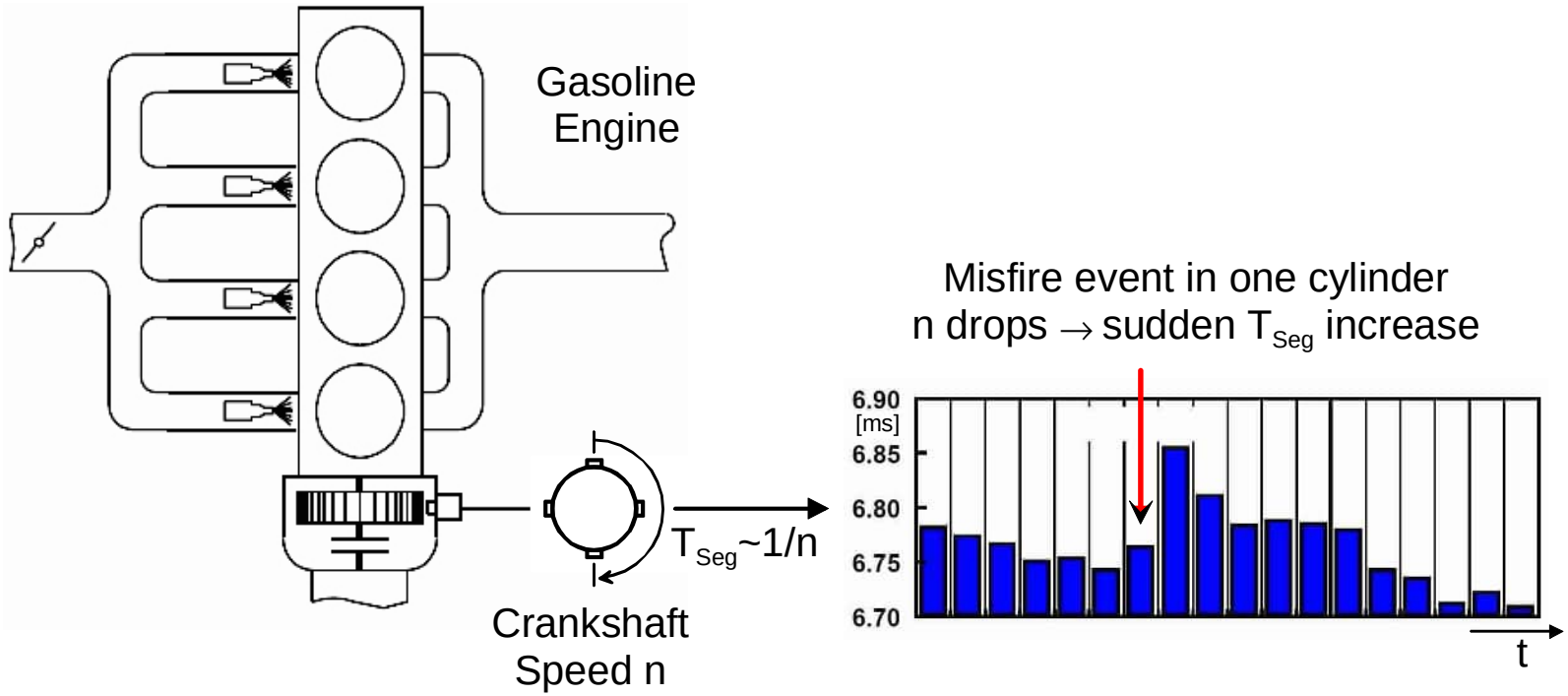


Passive test = monitoring of sensor data during normal engine operation

Active test = Test activated by ECU or tester on-demand in special situations, e.g. in workshop

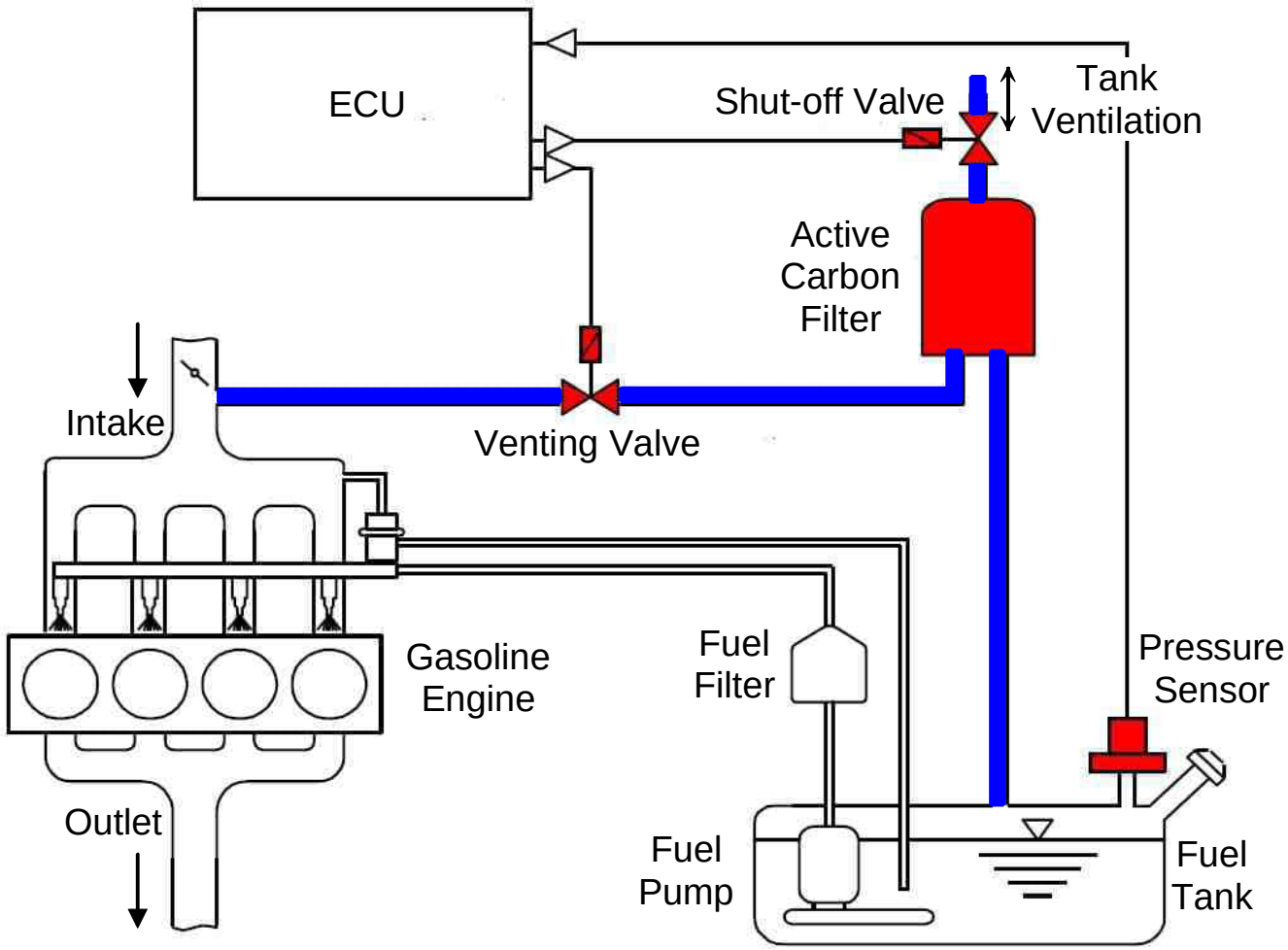
OBD requires that the catalyst be monitored by continuously testing its conversion rate.

- **Example OBD-Function: Misfire Detection – Passive Test**



Catalysts are destroyed rapidly, if unburned fuel gets into the catalyst due to ignition problems. Thus, OBD requires that misfire events must be detected and injection turned off as fast as possible.

- **Example OBD-Function: Tank Ventilation Monitoring – Active Test**



To avoid leakage of hydrocarbons (HC), the tank ventilation uses an active carbon filter.

The HC components in the filter are discharged from time to time via the intake manifold of the engine.

The ECU controls the system via two valves.

OBD requires to test the ventilation system for leakage by actuating the two valves and monitoring pressure increase and decrease in the fuel tank.

Test triggered by diagnostic tester.

Dia.3 Diagnostic Communication Protocols Standards

<table>
<tr><th></th><th>General-Purpose Workshop Diagnosis</th><th>On Board Diagnosis for Emissions-related ECUs</th></tr>
<tr><td rowspan="2">Application Layer</td><td>UDS
ISO 14229-1</td><td rowspan="2">OBD

ISO 15031-5
(content identical with
US standard SAE J1979)</td></tr>
<tr><td>ISO 15765-3
UDS on CAN</td></tr>
<tr><td>Transport Layer</td><td colspan="2">ISO 15765-2 ISO TP for CAN</td></tr>
<tr><td rowspan="2">Data Link Layer
Physical Layer
(Bus)</td><td colspan="2">ISO 11898 CAN (parameter limitations in ISO 15765-4)</td></tr>
<tr><td colspan="2">Outdated alternatives:
ISO 9141-2 K-Line,
SAE J1850 PWM or SAE J1850 VPWM</td></tr>
</table>

History of diagnostic communication in Europe

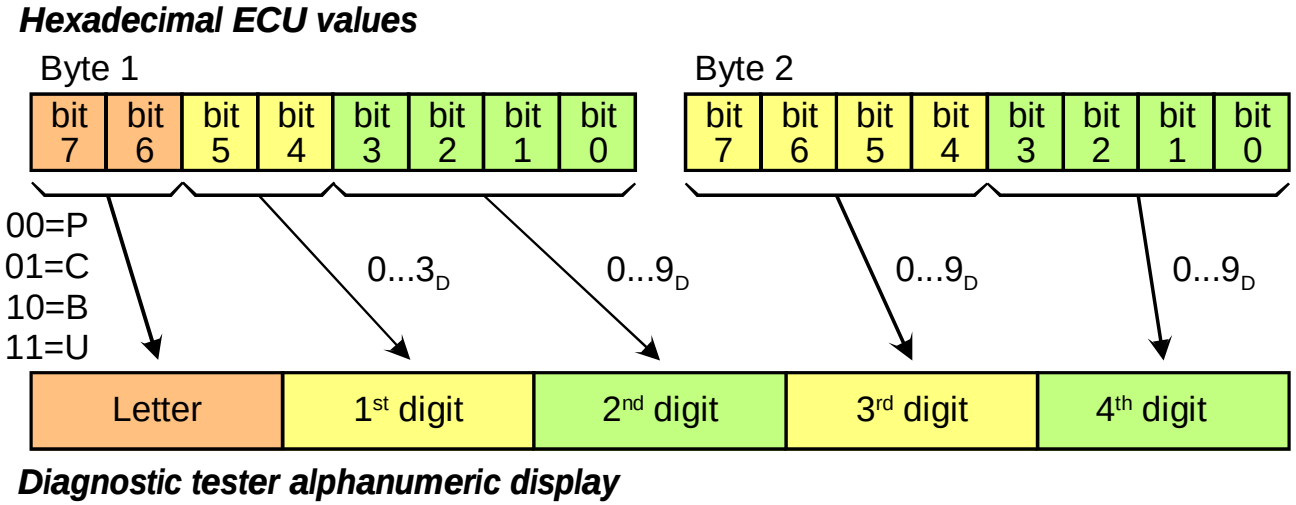
- **K-Line** physical layer (12V signal levels, UART, bit rates 300 … 9600 bit/s) standardized as **ISO 9141** by end of the 1980s, proprietary higher-layer protocols like Bosch KWP 71 or VW KWP 1281
- OBD diagnostics standardized in US beginning of 1990s with **SAE J1850** bus for US OEMs, K-Line interface **ISO 9141-2** CARB accepted for European OEMs. OBD required in Europe since 2001 (Diesel cars 2004). Usage of CAN obligatory in US since 2008 (passenger cars, light duty trucks), heavy-duty trucks from 2010 on.
- Keyword Protocol 2000 (**KWP 2000**) standard by mid of 1990s as **ISO 14230** based on K-Line physical layer.
- A port of KWP 2000 to CAN („KWP 2000 Diagnostics on CAN") was published as draft international standard ISO/DIS 15765-3 and used by many OEMs, but withdrawn in favor of UDS on CAN, current official standard ISO 15765-3, which is similar in concept, but not fully upward compatible to KWP 2000!

Dia.4 Additional Info for OBD on CAN as of ISO 15031-5 / SAE J1979 (OBD) and ISO 15765-4

- Which **PIDs** OBD diagnostic services SID 01_H, 02_H, 06_H, 08_H, 09_H do support, can be queried for each SID with PID=00_H, e.g. SID=01_H – PID=00_H. The response contains a bit field, where for each PID starting with PID=01_H a bit indicates, if the PID is supported, e.g.
  - Request: 7DF 02 01 00
  - Response: 7E8 06 41 00 98 3B 80 19

    98 3B … $_H$ = 1001 1000 0011 1011 … $_B$ → PIDs 1, 4, 5, 11, ...

- **Diagnostic Trouble Codes DTC** defined by ISO 15031-6



- Letter
  P … Powertrain
  C … Chassis
  B … Body
  U … Network
- 1st digit
  0 … DTC defined by OBD
  1,2 … DTC defined by OEM
- 2nd digit
  1,2 … Injection system
  3 … Ignition system
  4 … Exhaust gas & air system
- 3rd and 4th digit
  Component & kind of failure

Example: DTC = 0119_H → P 0 1 1 9

Powertrain, OBD DTC, Injection System,
Engine coolant temperature sensor, intermittent contact

- **Diagnostic Sequence**: Often a sequence of multiple requests is needed to accomplish a certain operation, e.g. typical trouble shooting process

Request	*Response*
SID=01_H – PID=01_H Query, if and how many DTCs are stored	1^{st} byte, bit 7 = MIL Status 1^{st} byte, bit 6…0 = DTC count Other bytes: Information about active monitoring functions, e.g. misfire monitoring, fuel system monitoring, …
SID=03_H – no PID Query of DTCs	1^{st} byte: Number of DTCs 2^{nd}+3^{rd} byte: DTC for first failure 4^{th}+5^{th} byte: DTC of second failure etc.
SID=02_H – list of PIDs Read stored measurement values for stored DTCs (Freeze Frames)	Freeze Frame data (uses same PIDs and format as SID=01_H)
Repeated requests of monitoring results with SID=06_H, 07_H, 08_H and reading of measurement values with SID=01_H during trouble shooting and repair	
SID=04_H – no PID Clear all DTCs and Freeze Frames	Confirmation

→ For practical use, tools are required, which hide the request / response details and convert data into a human-readable format (**OBD Scan Tools**)

Dia.5 UDS on CAN

ISO 14229-1 (UDS), ISO 15765-3 (UDS on CAN),
Based on former ISO 14230 (KWP 2000 on K-Line), ISO/DIS 15765-3 (KWP 2000 on CAN)

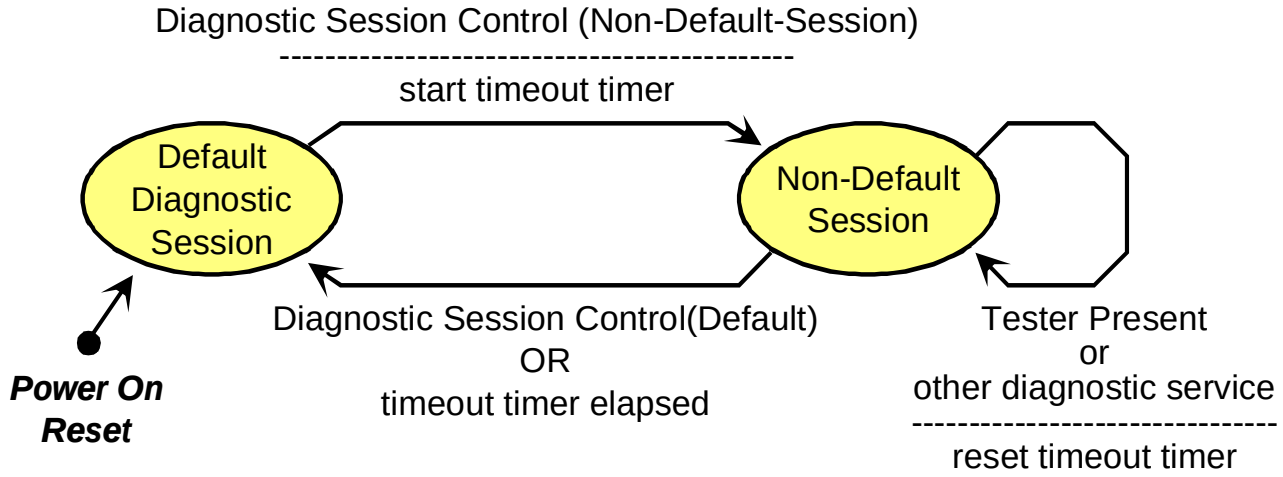
- UDS Workshop Diagnosis vs. OBD Emissions-related Diagnosis
  - CAN IDs not standardized, defined by OEM. Bit rate defined indirectly, if the same bus is used for OBD
  - UDS only specifies SID and PID (in UDS called LEV, when selecting sub-functions, called Identifier ID, when selecting data values), but not the data contents of the messages, e.g. DTCs and data (P)IDs are freely defined by the OEM
  - Some services, e.g. reading DTCs, have equivalent OBD services, but use other SIDs and parameters
  - Provides **read and write access** to nearly all ECU internals with access protection **via Diagnostic Session** concept (see below)
  - Data can either be accessed via (P)IDs, which are predefined by the OEM (read/write data by identifier), or by memory addresses (read/write memory by address).
- Same message format as OBD messages (see lecture chapter 3.10)
  - Selection of **Service by SID** (1st byte)
  - Selection of a sub-function or parameter by **Sub-function Level Byte LEV$_{\text{Bit 6...0}}$** (2nd byte, same as PID Byte in OBD messages)
  - If in the tester request the LEV$_{\text{Bit 7}}$= 1 option is set, the ECU need not send a positive response, e.g. used with Tester Present service (see Diagnostic Session concept below). Negative responses must always be sent.

UDS Diagnostic Services (extract from ISO 14229 and ISO 15765-3)

Functional Group	*SID*	*Default*		
Diagnostic and Communication Management	10_H	X	Diagnostic Session Control	*Management of Diagnostic Sessions*
	$3E_H$	X	Tester Present	
	27_H		Security Access	
	11_H	X	ECU Reset	
	28_H		Communication Control	*Control of Bus Communication*
	85_H		Control DTC Setting	
	83_H		Access Timing Parameters	
	87_H		Link Control	
	86_H	(X)	Response on Event Service	
Data Transmission	22_H	(X)	Read Data by Identifier	*Read and Write ECU Data*
	23_H	(X)	Read Memory by Address	
	$2E_H$	(X)	Write Data by Identifier	
	$3D_H$	(X)	Write Memory by Address	
	$2A_H$		Read Data by Periodic Identifier	
	$2C_H$	X	Dynamically Define Data Identifier	
Stored Data Transmission	14_H	X	Clear Diagnostic Information	*Read and Write Diagnostic Trouble Codes (Fault Memory)*
	19_H	X	Read DTC Information	
Input/Output Control	$2F_H$		Input Output Control by Identifier	*Execute ECU-Routines, e.g. Actuator Tests, Flash-Programming Routines*
Remote Activation of Routine	31_H	(X)	Routine Control	
Upload / Download	34_H		Request Download	*Read and Write large Blocks of Data Main Purpose: Flash-Programming*
	35_H		Request Upload	
	36_H		Transfer Data	
	37_H		Request Transfer Exit	

Diagnostic Sessions

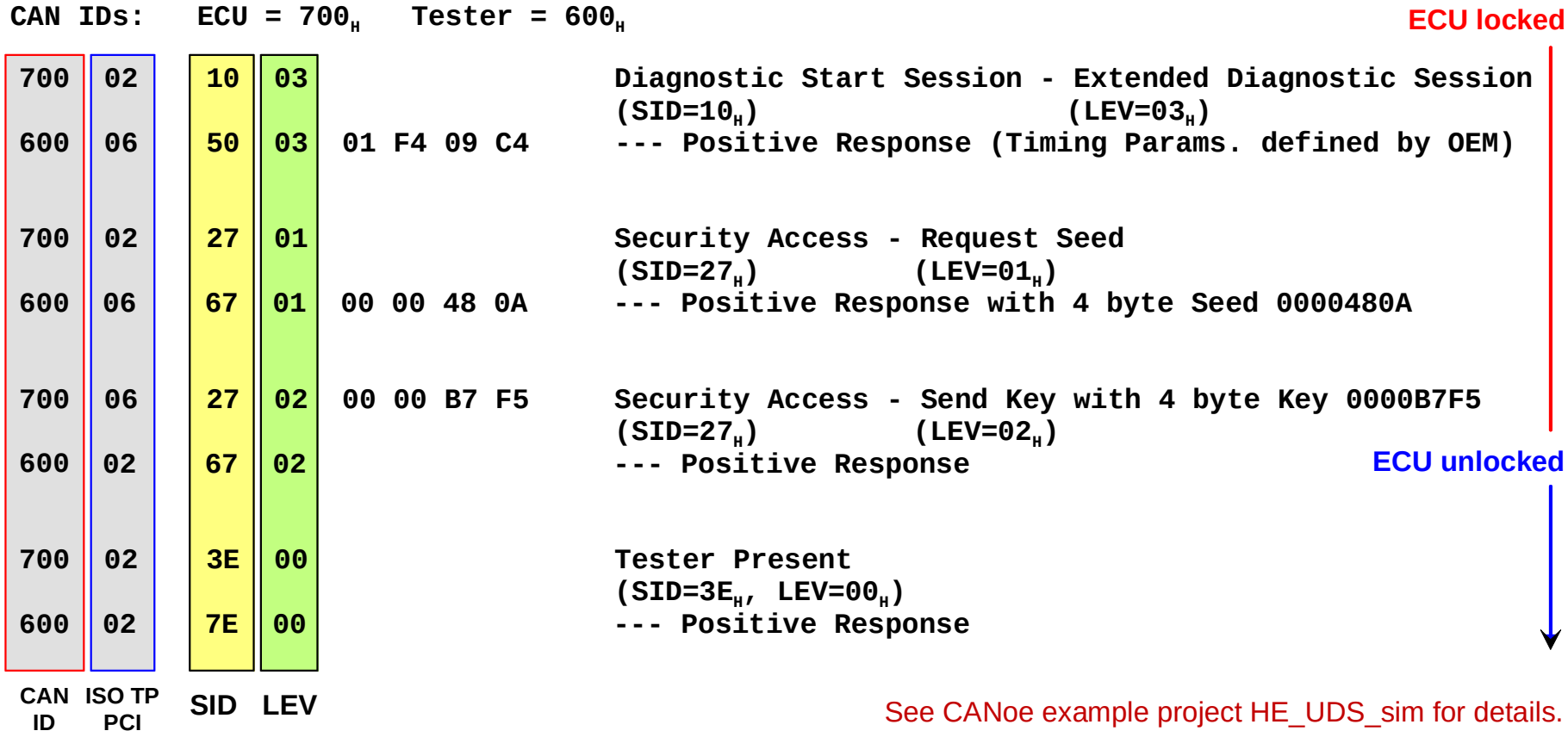
- In normal operation (***Default Diagnostic Session***) only a subset of diagnostic services is available (all OBD services plus those UDS services marked with X in the UDS service table) and only restricted access to ECU data is possible (services with restricted access marked with (X))
- Critical services require switching to a special diagnostic session via service ***Diagnostic Session Control***: *Programming Session*: used for Flash-Programming
  Extended Diagnostic Session: Adjustment of engine idle speed etc.
  Additional OEM- or manufacturer-specific sessions can be defined
- A non-default-session will be terminated automatically, if the tester does not periodically send ***Tester Present*** or any other diagnostic request (timeout $P3_{max}$= 5s, typ. 4s).



- A non-default session typically requires a login procedure, where the tester sends a ***Security Access*** request and the ECU responds by sending an initial value (***Seed***) for the au-

thentification algorithm. The tester computes a "password" (***Key***) based on the *Seed* and sends it back to the ECU. The ECU checks the *Key* and confirms login to the non-default session by a positive response or denies access sending a negative response. The secret algorithm to compute the *Key* is OEM-specific and must be known by both the ECU and by the tester.

Example: Switch to another diagnostic session with security access



See CANoe example project HE_UDS_sim for details.

Example: Flash-Programming (simplified procedure according to HIS [T1])

Read Data By Identifier $(22)_H$		*Read ECU identification to check hardware and software compatibility*
Diagnostic Session Control – Extended Diagnostic Session $(10\ 03)_H$		*Switch to extended diagnostic session (Seed & Key login not shown)*
Routine Control – Check Progr.Precond. $(31\ 01\ 02\ 03)_H$		*Check preconditions, e.g. engine must be off*
Control DTC Setting – DTC Setting = Off $(85\ 02)_H$		*Turn off fault storage in other ECUs*
Comm. Control – Disable Non-Diagnostic Communication $(28\ 01\ 01)_H$		*Switch off normal communication in other ECUs (to increase available bus bandwidth)*
Diagnostic Session Control – Progr. Session $(10\ 02)_H$		*Switch to programming session*
Security Access – Request Seed $(27\ 11)_H$ Security Access – Transfer Key $(27\ 12)_H$		*Login procedure: Request seed – send Key to ECU*
Write Data By Identifier – Write Fingerprint $(2E)_H$		*Write tester identification to ECU, to document programming operation*
Routine Control – Erase Memory $(31\ 01\ FF\ 00)_H$		*Erase Flash memory*
Loop with	Request Download $(34)_H$ Transfer Data $(36)_H$ Request Transfer Exit $(37)_H$	*Transfer data to ECU*
Routine Control – Check Memory $(31\ 01\ 02\ 02)_H$ Routine Control – Check Reprog. Dependencies		*Verify programming and check consistency*
ECU Reset $(11\ 01)_H$		*Reset ECU*
Comm. Control – Enable Non-Diag.Comm. $(28\ 00\ 00)_H$		*Turn on normal communication again*
Control DTC Setting – DTC Setting = On $(85\ 01)_H$		*Turn on failure storage again*
Diagnostic Session Control – Default Session $(10\ 01)_H$		*End of programming session → default session*

Communication Parameters

- When switching to another diagnostic session, timing parameters of the transport protocol or data link layer, e.g. timeout values, can be changed via ***Access Timing Parameters*** and ***Link Control*** services.
- To increase the available bus-bandwidth for Flash-programming, ***Normal Communication***, i.e. sending of on-board messages, can be turned off in all ECUs via service ***Communication Control***.
- When testing actuators or Flash-programming, error monitoring and failure storage can be turned off via service ***Control DTC Setting***.

Special UDS Features: Services to log measurement data on test stands or vehicle calibration

- ECU can send **measurement data periodically**, when configured once by a ***Read Data By Periodic Identifier*** request (instead of polling, which must be used with OBD or KWP 2000). The sender can use one of 3 different periods predefined by the OEM. This service allows optional response formats without PCI and SID to reduce latency and busload.
- **Event-triggered transfer of ECU data**, when configured once by a ***Response on Event Service*** request. Events must be predefined by the OEM, e.g. timer interrupts, failure events, values changing or reaching certain limits of ECU variables

Practical Implementation

- UDS is a complex protocol. Some of its services (e.g. *Response on Event*, *Read Scaling Data*) require a lot of ECU resources and some of the services are redundant (e.g. *Write Memory by Address*, *Transfer Data*) → typically an OEM-specific subset is used

Dia.6 World-Wide-Harmonized OBD

Current situation: 2+ independent diagnostic protocols

- for **emissions-related** diagnosis **OBD** on CAN standardized in ISO 15031
- for **repair-shop diagnostics** in Europe **UDS** on CAN standardized in ISO 14229
- for repair-shop diagnostics of commercial vehicles in US SAE J1939/73

Problem:

- Functionally, OBD-services are a subset of existing UDS services, but the protocol implementation is different.
- **Additional implementation effort for the OBD protocol** both in the ECUs and in the diagnostic testers

Solution: **World-Wide-Harmonized OBD standardized in ISO 27145**

- **Map ODB services to UDS services**, i.e. only implement UDS protocol.
- E.g. UDS-service SID=22_H Read Data by Identifier instead of OBD-services SID=01_H Read Measurement Values and SID=09_H Read Vehicle Information
- Or UDS-service SID=14H Clear Diagnostic Information rather than OBD-service SID=04H Clear DTCs

Current status since first standard release in 2012

- **First use cases in some commercial vehicles** (trucks and busses) as a mid-term substitute for existing standards ISO 15031 and SAE J1939/73
- Use in passenger cars intended, to be expected long-term

Dia.7 Diagnostics on IP

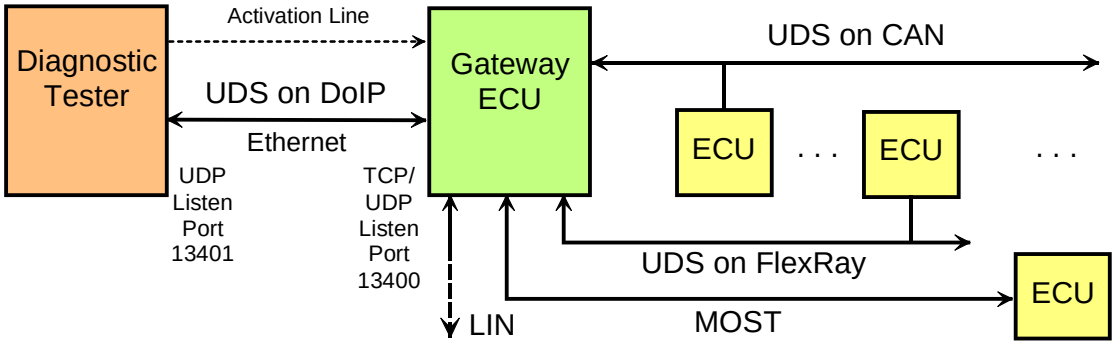
Current situation:

- OBD (and repair-shop) diagnostics use **CAN interface on the vehicle-side**
- **Diagnostic testers** are **implemented on normal PC/notebook computer hardware** (and operating systems). Standard computer interfaces are Ethernet, USB, WLAN, …

Problem:

- Costly CAN **bus interface hardware on tester-side** required
- **CAN bus too slow for large ECU software updates**, especially for infotainment ECUs.

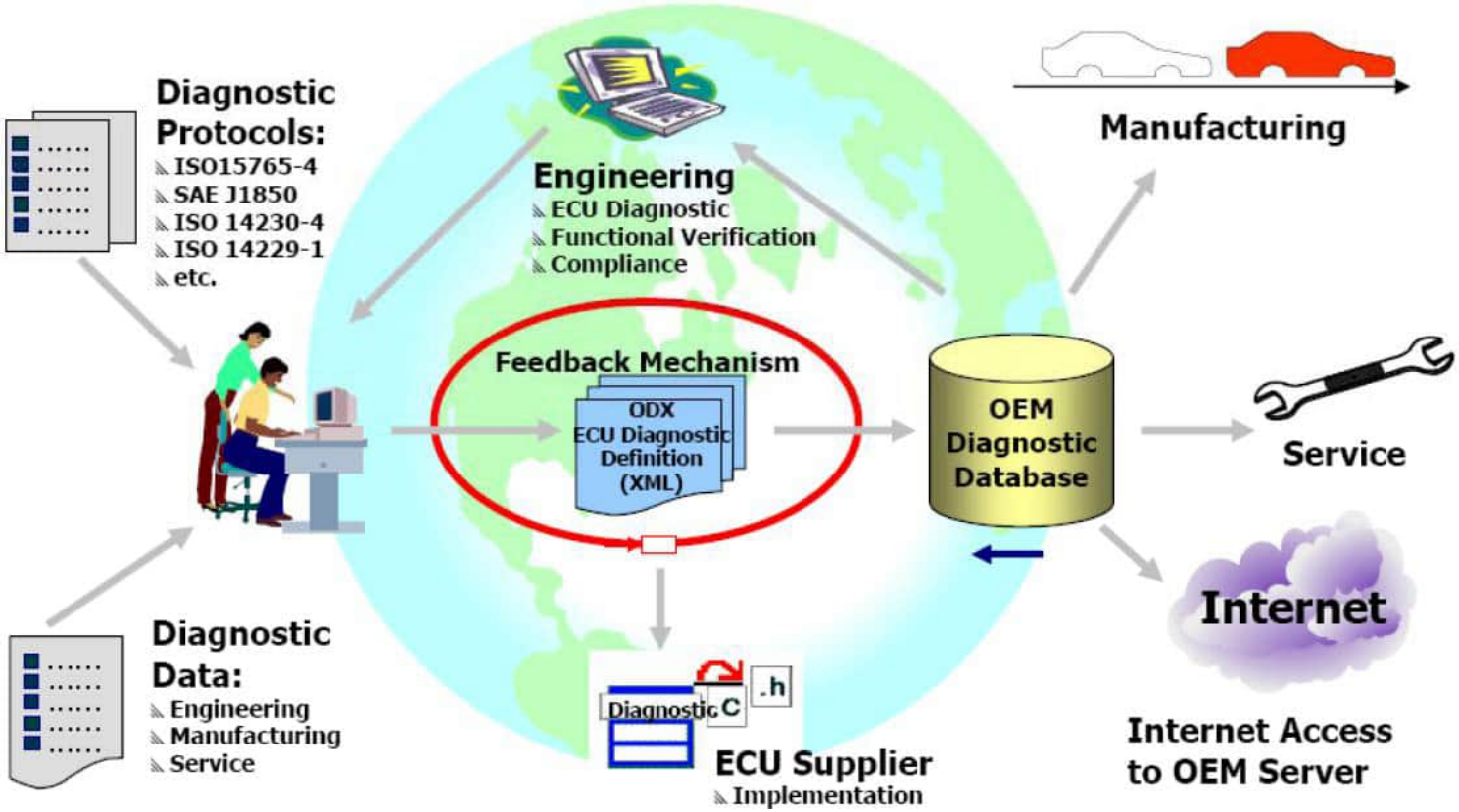
Solution: **Diagnostics over IP (Internet Protocols) in ISO 13400**



- Use standard **Ethernet** interface **as** the **vehicle diagnostics connection**, short-term in parallel to CAN, long-term as substitute (for off-board communication)
- **Tunnel UDS/OBD-messages via UDP+TCP/IP** using a modified version of ISO TP to allow easy forwarding of messages to/from the internal CAN, FlexRay and LIN busses

Dia.8 Development of Diagnostic Systems: ASAM D-Servers and ODX

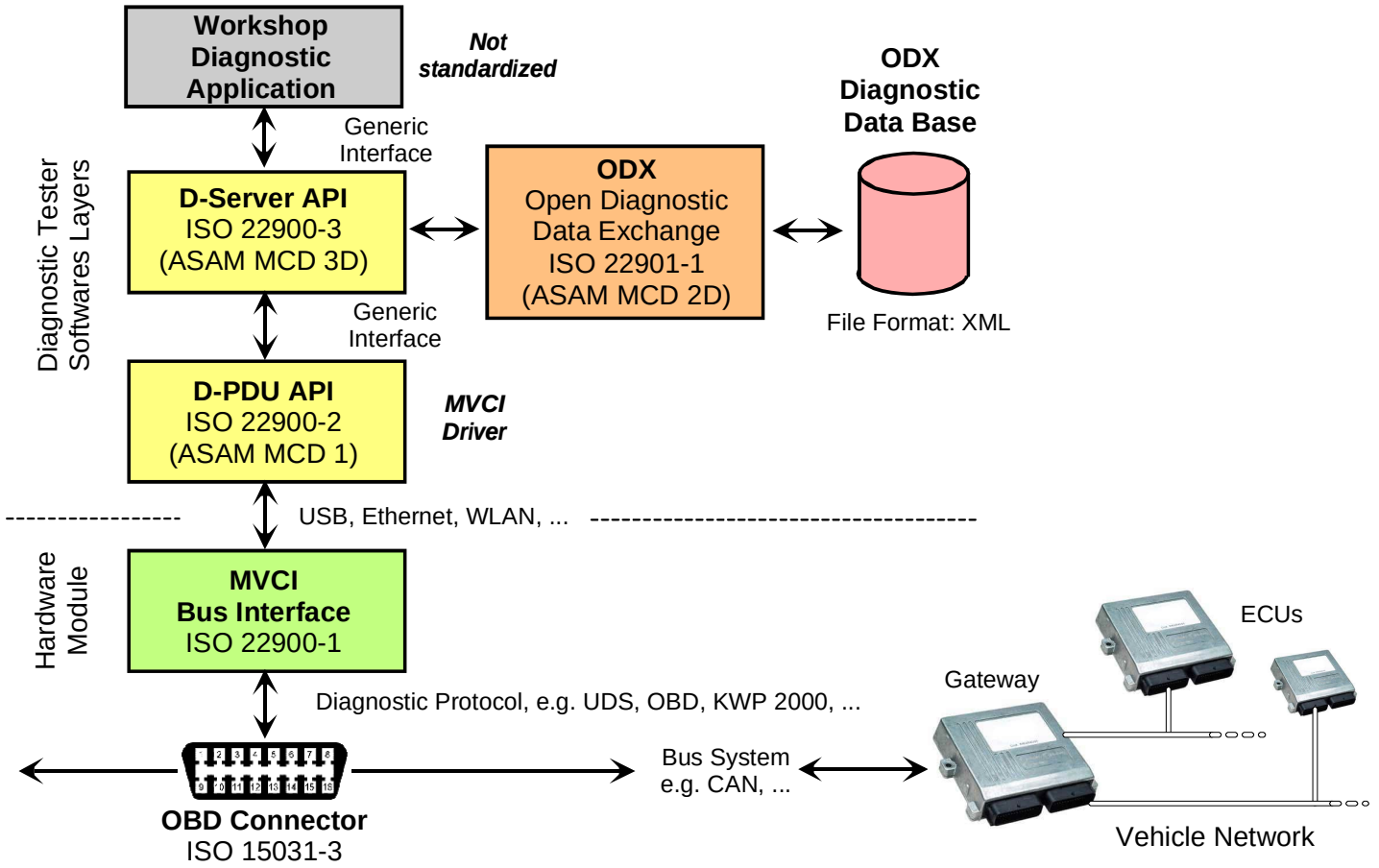
Diagnosis must be handled over the complete lifecycle of a vehicle and should be compatible between all car platforms of an OEM



Source: ASAM … Association for Standardization of Automation and Measuring Systems (former name: ASAP)

To reduce the effort of **developing diagnostic testers** and **simplify data exchange** between ECU suppliers, OEMs and service organizations, the **ASAM** initiative was founded.

ASAM has proposed a **standard structure for diagnostic systems**



AE … Automotive Electronics
M … Measurement, C … Calibration, D … Diagnosis
PDU … Protocol Data Unit

Modular Vehicle Communication Interface **MVCI** handles the various diagnostic protocols OBD, UDS, … including real-time aspects, e.g. timeouts.

D-PDU API (Application Programming Interface) provides a bus system independent software interface between tester and ECUs via the MVCI.

ODX (Open Diagnostic Exchange) holds the data describing all diagnostic functionality available in the vehicle.

D-Server API provides a run-time environment for the OEM-specific diagnostic application, hides database and communication details.

ODX, an XML based data format, plays a major role in describing, exchanging and merging diagnostic-relevant data between OEMs and suppliers. The most important ODX description categories are:

Vehicle Info & ECU Config Spec	Comm Param Spec	Diag Layer Containers	Function Dictionary, Flash & ECU Jobs
• How to identify the vehicle • How to access the ECUs in the vehicle • How to configure the ECUs	• Diagnostic communication protocols, bus systems and their parameters	• Diagnostic services • Interpretation of ECU internal data, e.g. conversion between raw data and physical values	• Library combining various services to achieve a certain workshop-level functionality • Script automation for diagnostic sequences, e.g. Flash- and End-of-Line programming

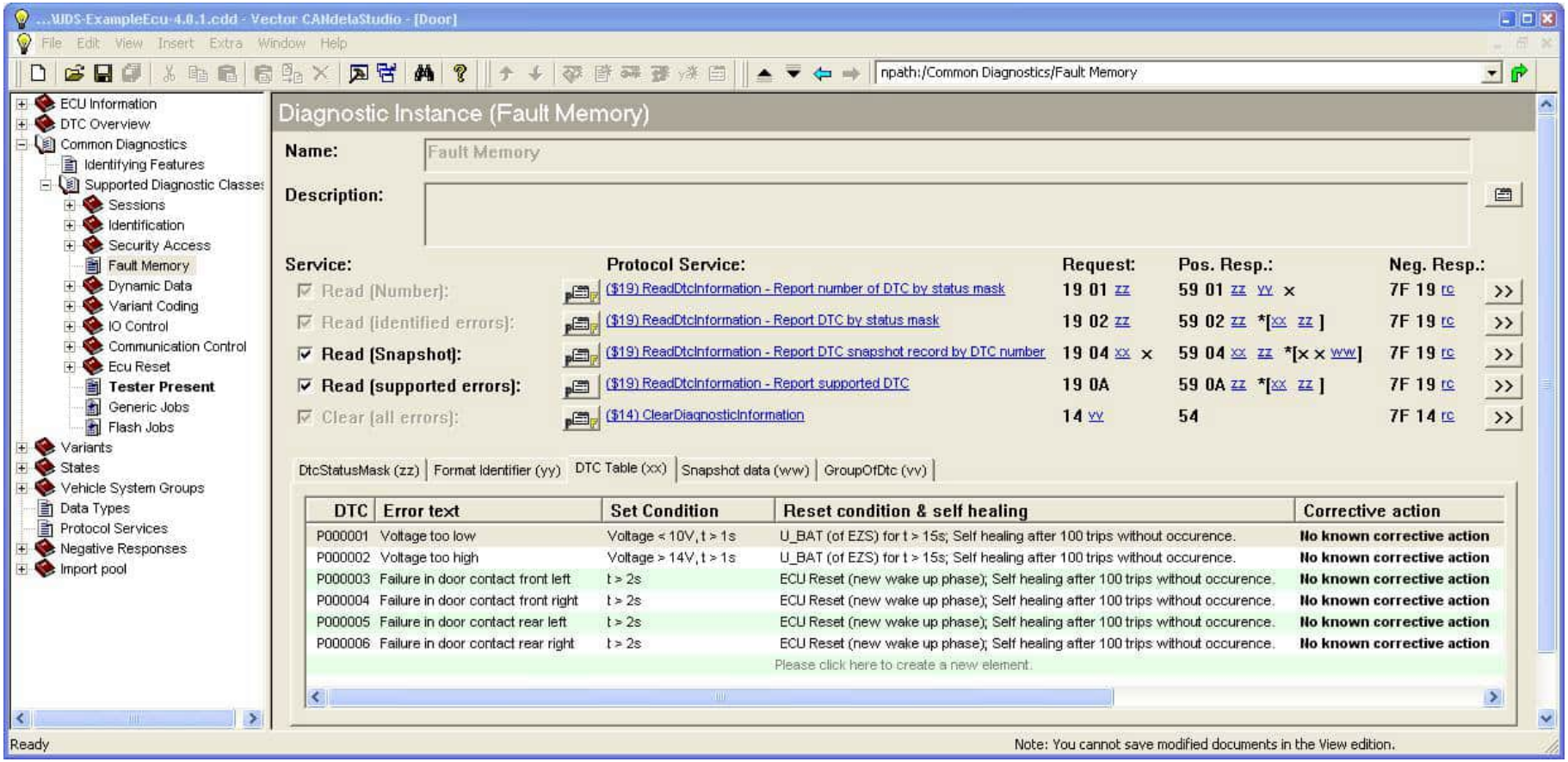
Key ideas of ODX are

- **single-source principle**, i.e. data is only described once, e.g. one generic description for UDS protocol for all vehicles of an OEM
- **variant management** by configuring generic descriptions, restricting or extending functionality in a hierarchical way, so that families of products can easily be handled, e.g. door control ECU base variant – variant for left front door – German and US variants

Info for software engineers:
These features are implemented by an object-oriented XML description with inheritance, association and aggregation of objects.

Tools to create and manage ODX databases and develop diagnostic testers according to the ASAM structure are available by various vendors, e.g.

- Samtec Diagnostic Framework samDia, ODXExplorer, samMCD3server, bus interfaces, …
- Softing DTS Venice, DTS Monaco, ...
- Vector CANdela Studio, CANdito, ...



Example: Fault Memory definition for CANoe project HE_UDS_sim in CANdela viewer

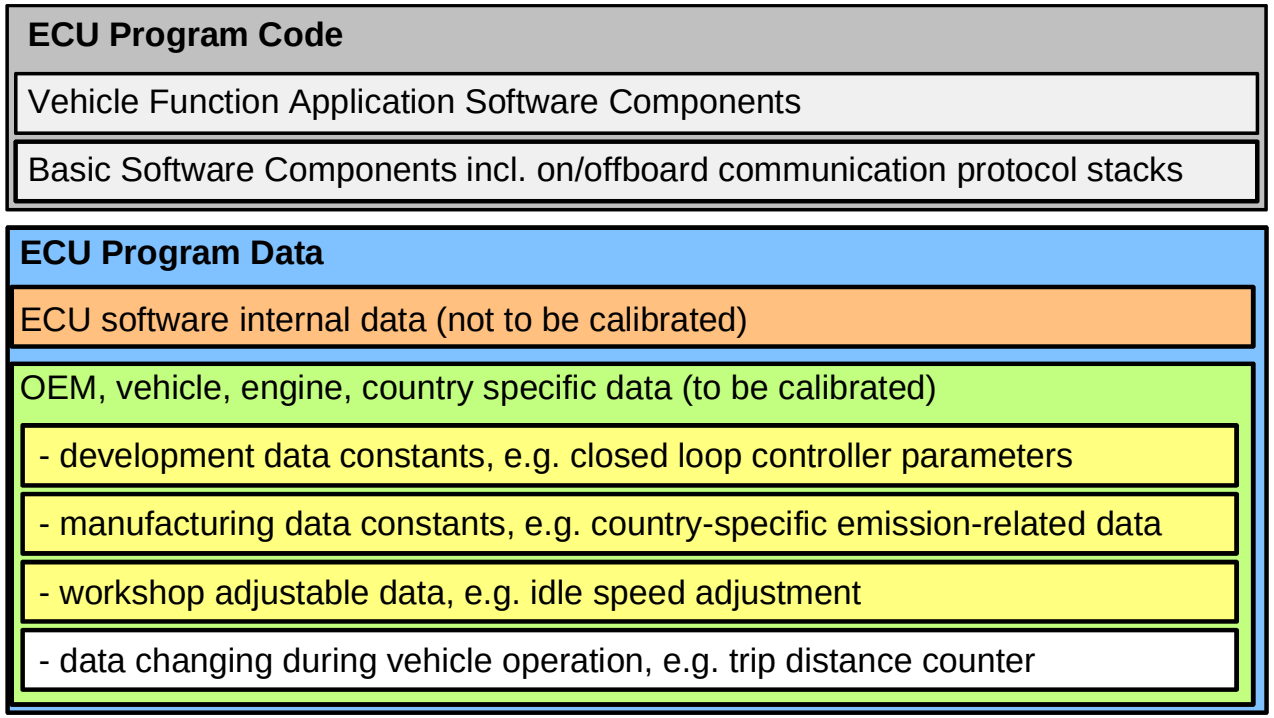
Dia.9 Calibration Support: ASAM ASAP2, CCP and XCP

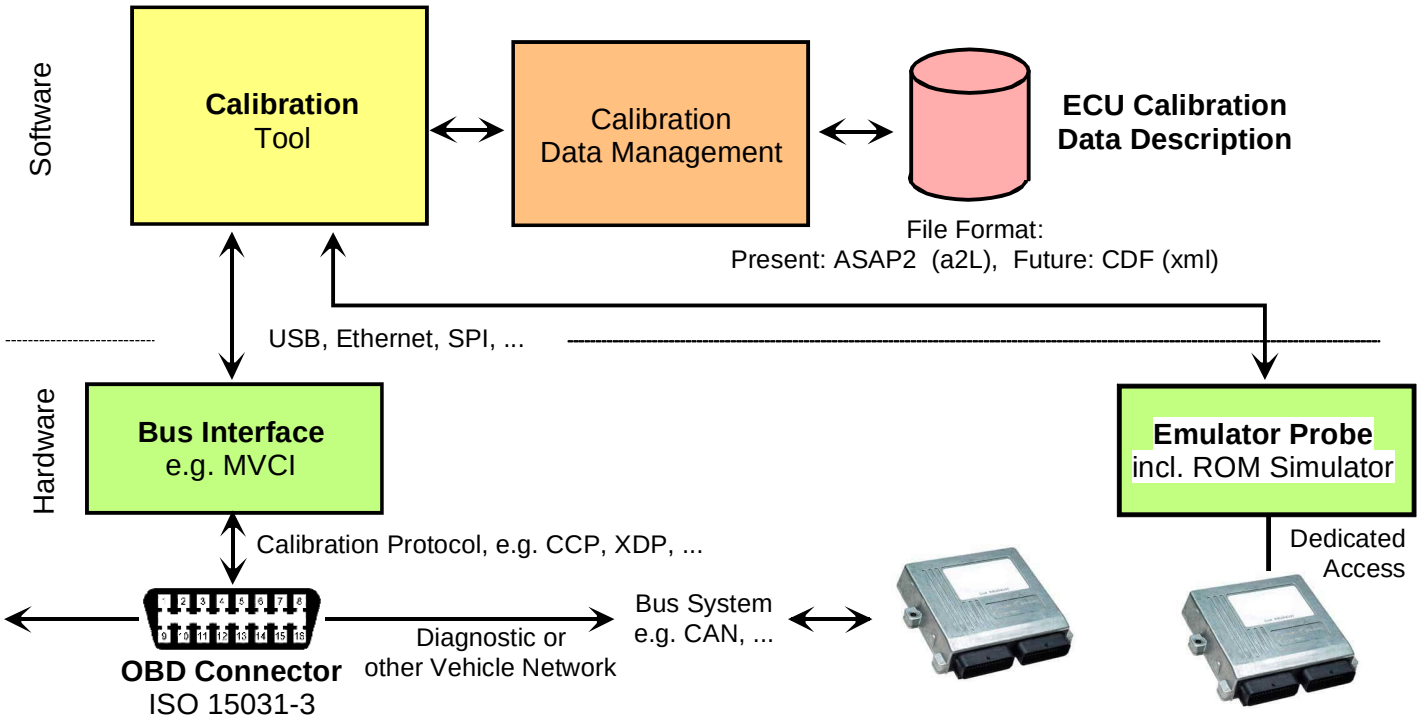
Most systems are developed in a generic way (base system or platform system) and later adapted to OEM and vehicle model specific conditions. To reduce the development effort, most of the adaptation shall be done via changing parameters in the ECU software.

The adaptation typically is done by **measuring** data on test stands, test tracks, … and **calibrating** the ECU parameters accordingly, e.g. closed-loop controller settings, open-loop characteristic curves and maps, ….

Software structure to allow application-specific adaptation of generic ECU software:

- Keep code and data separate
- Partition data to control access separately for
  - ECU development
  - OEM development
  - ECU manufacturing
  - OEM manufacturing
  - OEM-certified workshops
  - Others





Theoretically, measurement and calibration could use standard UDS diagnostic services like Read/Write Memory over the ECU's bus interface and ODX as data base format.

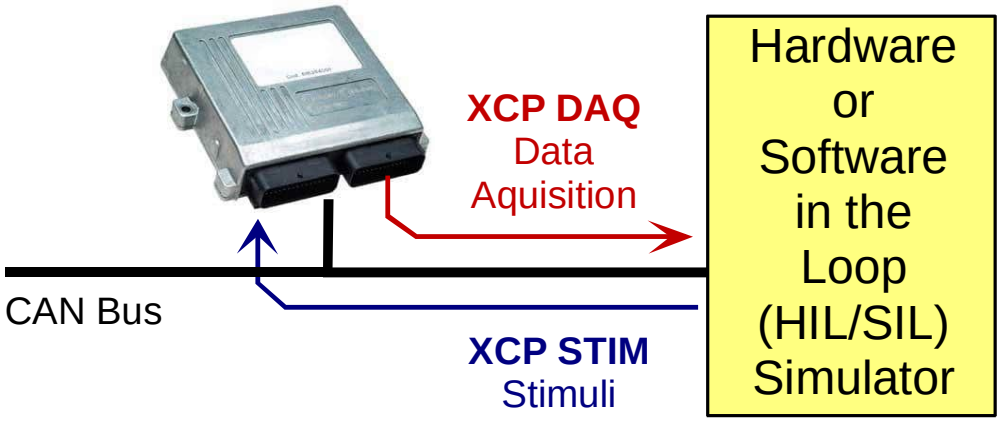
Due to historical and technical reasons however, **Measurement and Calibration** (ASAM MC) does use own protocols and data formats (all standardized by ASAM):

- **CAN Calibration Protocol CCP** (CAN only) and its successor **Extended Calibration Protocol XCP** (CAN, FlexRay, SPI, Ethernet, …)
- **ASAP2 text format** to describe ECU data (memory addresses, encoding, conversion between raw and physical values, …) or its successor **Calibration Data Format CDF** or **AUTOSAR XML**.

Reasons for using different solutions are:

- MC communication during calibration could influence normal network and/or ECU operation
- Overhead of diagnostic protocols could reduce measurement sampling rate
- Unlimited access to all ECU internals, which is required for calibration, could create back-doors for hackers and tuners
- Access via non-standard methods, e.g. emulator probes required for ECUs without (fast) bus-interface, is easier to implement.
- MC communication and data management already was used in the K-line epoch in the pre-diagnostics on CAN age and long before ODX or UDS were standardized.

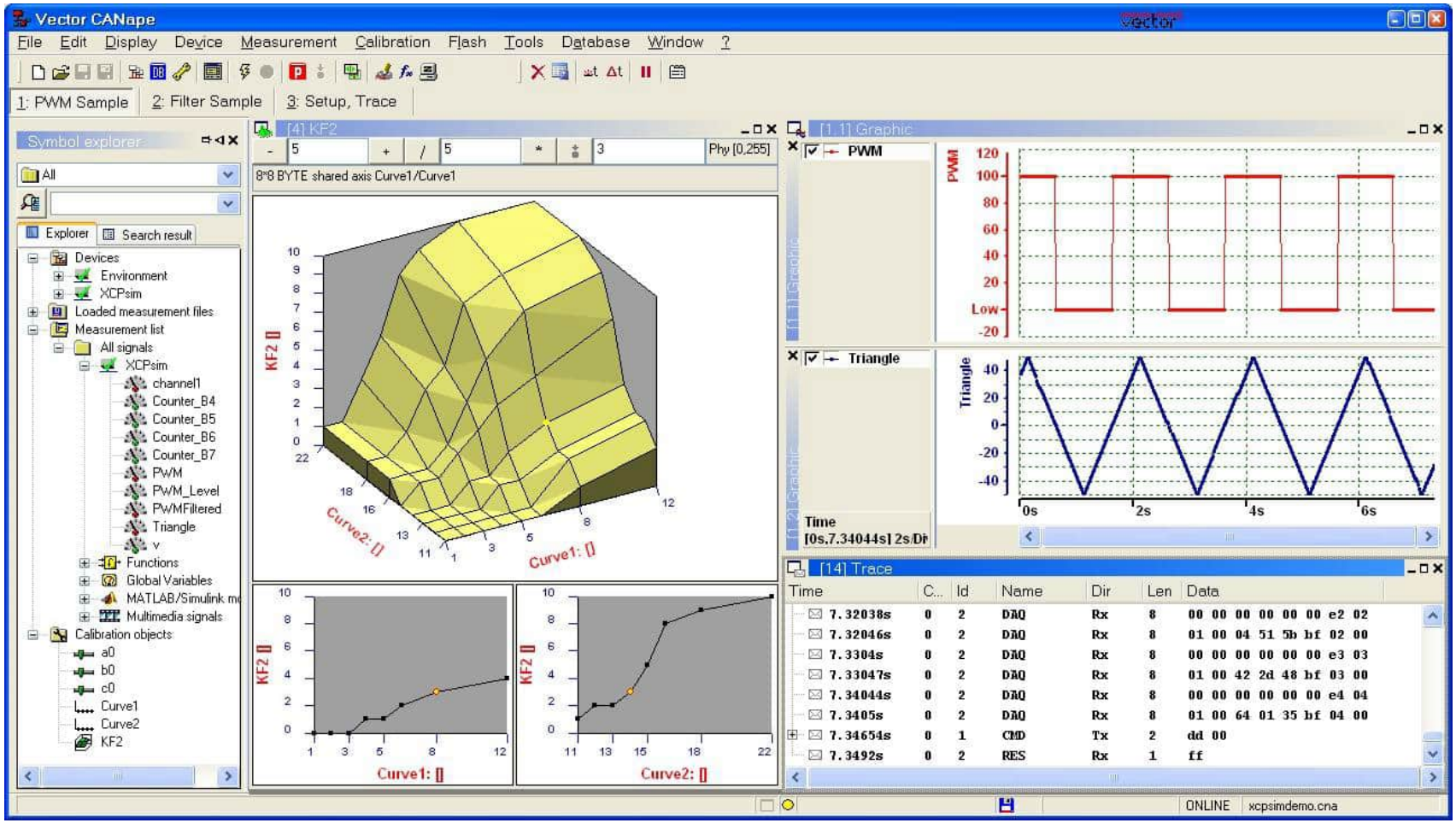
ASAM is trying to integrate and streamline MC definitions into their D-related standards, but have not yet been very successful with respect to available real-world tools.



XCP communication can also be used in HIL- and Bypass-simulation and rapid-prototyping systems, e.g.

- ETAS ASCET , INTECRIO, LabCAR
- dSpace Prototyping Systems
- Mathworks Matlab/Simulink

Tools to create calibration data descriptions and handle measurement and calibration tasks like data visualization are available by various vendors, e.g. ETAS INCA or Vector CANape



Example: Vector CANape XCPsim demo

Further Reading

- Standards

[D.1] OBD diagnostics ISO 15031: Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics.

[D.2] UDS diagnostics ISO 14229: Road vehicles – Unified diagnostic systems (UDS)

[D.3] UDS on CAN ISO 15765 (DoCAN): Road vehicles – Diagnostics on Controller Area Networks (CAN)

[D.4] KWP 2000 diagnostics ISO 14230 (DoK): Road-Vehicles Diagnostic Systems – Keyword Protocol 2000

[D.5] KWP 2000 on CAN: ISO/DIS 15765-3: Road vehicles – Diagnostics on Controller Area Networks (CAN)

This draft international standard (DIS), called *KWP 2000 on CAN* was withdrawn in favor of *UDS on CAN.*

[D.6] ODX: ISO/DIS 22901 Road vehicles - Open diagnostic data exchange (ODX).

Based on the ASAM specification for ASAM MCD 2D

[D.7] D-Server, D-PDU and MVCI: ISO 22900 Road vehicles - Modular vehicle communication interface

Based on the ASAM specification for ASAM MCD 3D (Diagnostics)

[D.8] CCP, XCP, ASAP2/A2L: ASAM specifications for ASAM MCD 2MC (Measurement and Calibration)

All ISO (International Organization for Standardization) standards available from www.iso.org

All SAE (Society of Automotive Engineers) standards available from www.sae.org

All ASAM (Association for Standardization of Automation and Measuring Systems, former name ASAP) standards available from www.asam.net

[D.9] World-Wide-Harmonized OBD diagnostics ISO 27145

[D.10] Diagnostics on IP (DoIP) ISO 13400

Most books mentioned in chapter 2 and 3 also have a section about diagnosis.

Literature

- Special Topics

[T1] Herstellerinitiative Software (HIS): HIS-conformant programming of ECUs based on UDS. Version 1.0, 2006 (in German). www.automotive-his.de, now moved and integrated into www.autosar.org

- Books

[D.10] W. Zimmermann, R. Schmidgall: Bussysteme in der Fahrzeugtechnik. Vieweg, 5th edition, 2014 (in German, has an extensive section about diagnostic and calibration support protocols)

[D.11] C. Marscholik, P. Subke: Road Vehicles – Diagnostic Communication. Huethig, 1st edition, 2008 (newer 2011 version in German available)