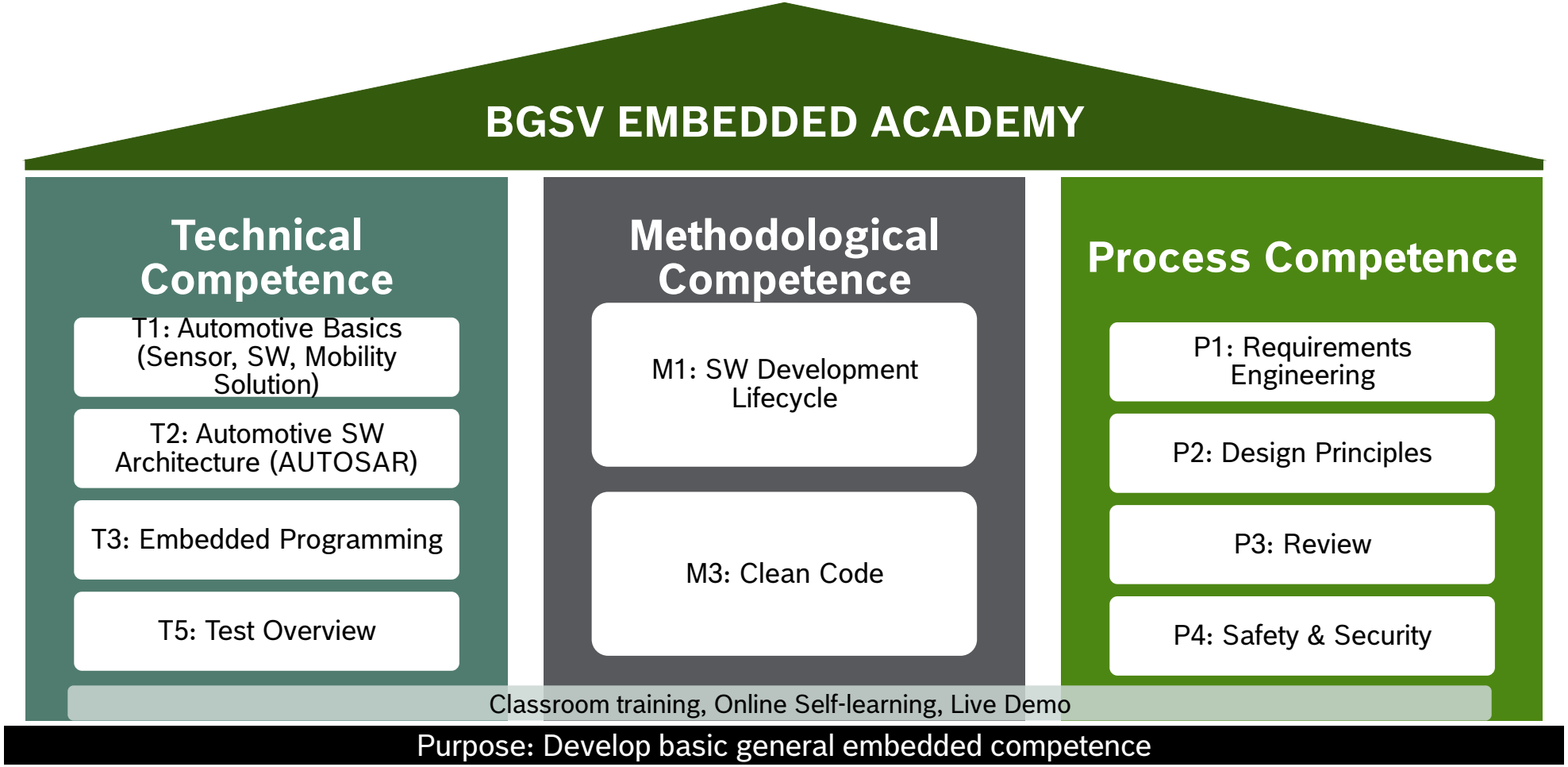
EMBEDDED ACADEMY

PEDAL TO THE MEDAL

BOSCH

BGSV Embedded Academy (BEA)

Focused Program to Develop Embedded Competence



Bosch Global Software Technologies
alt_future

Disclaimer

- This slide is a part of BGSV Embedded Academy (BEA) program and only used for BEA training purposes.
-
- This slide has some copyright images and text, which belong to the respective organizations.

T2
AUTOMOTIVE SOFTWARE ARCHITECTURE

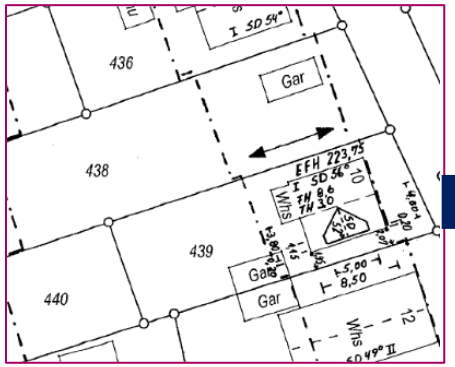
WHAT IS SOFTWARE ARCHITECTURE?

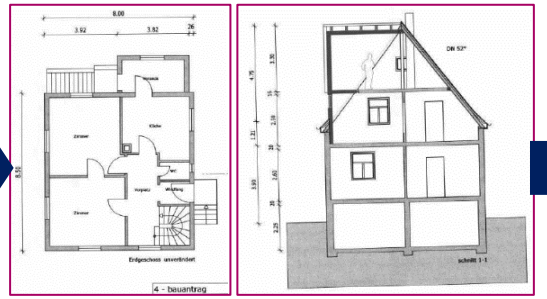
What is Software Architecture

How could we be so wrong?

What is Software Architecture
Build a house

- Surveyor plan
- Floor plan
- 2D or 3D views
- Implementation





Plot environment (adjacent to neighbors‘ plots), permissible and used building footprint

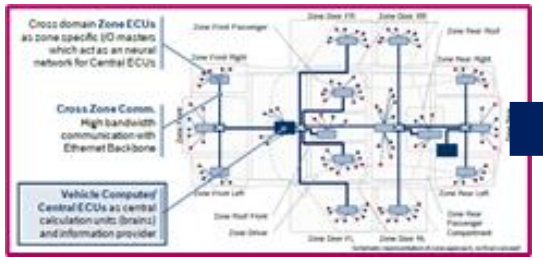
Position and size of walls, wall openings, floors and ceilings

3D representation of the building or parts of it. Location and views of furniture and fixtures. Virtual tour inside the object.

Physical realization: built home (ready to move in).

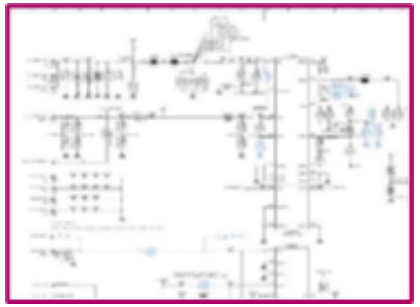
What is Software Architecture
How to build an ECU

- E/E architecture



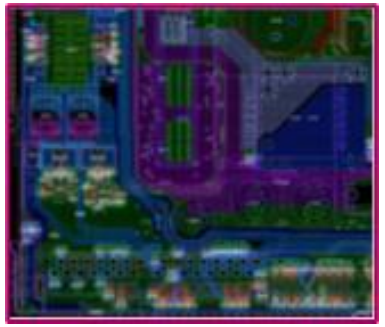
Network and electrical environment of the ECU

- Circuit



Logical wiring of components describing electrical functions

- PCB Layout



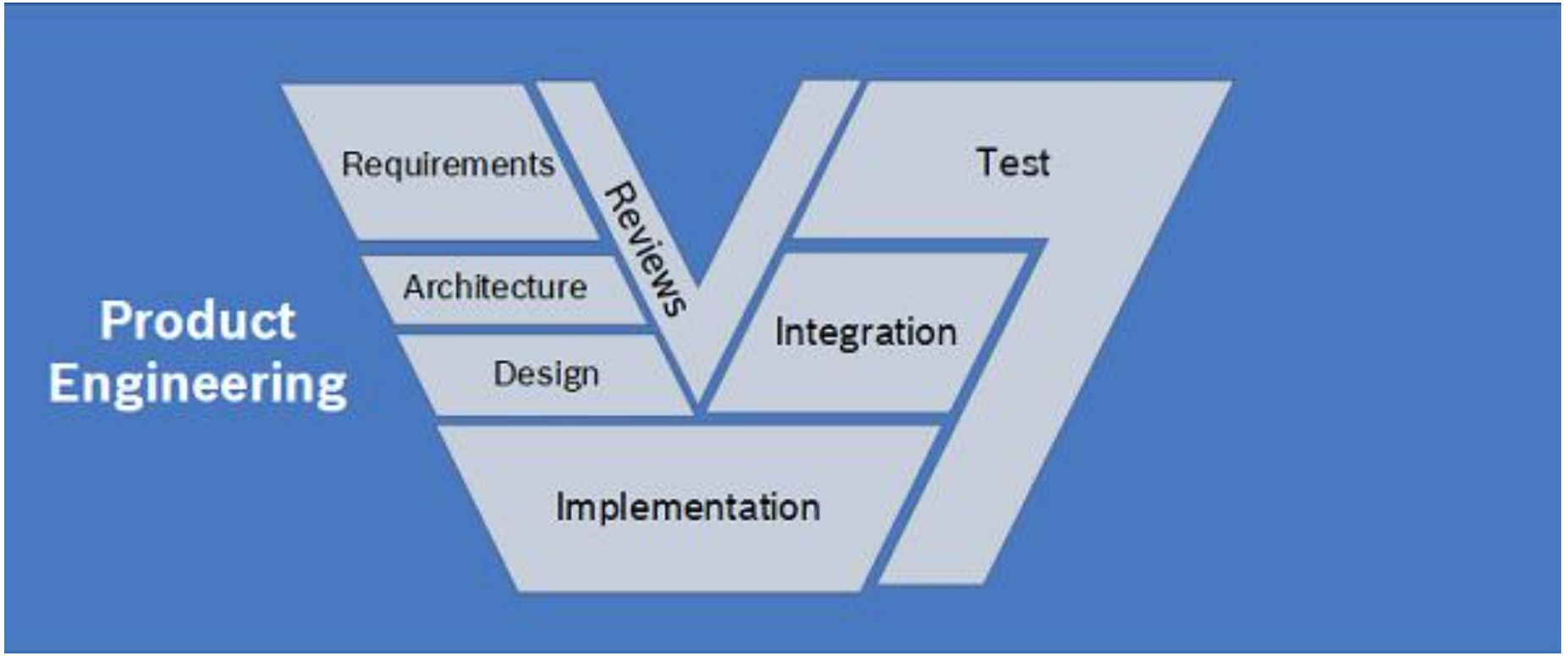
Physical location and wiring of components (considering of non-functional requirements like EMV, heat dissipation, physical interfaces to housing)

- Implementation

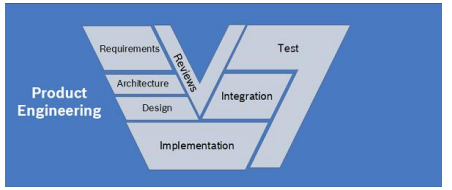
Physical realization: anufactured Electronic Control Unit to be installed into the vehicle.

What is Software Architecture

How to build a software product – Oversimplify

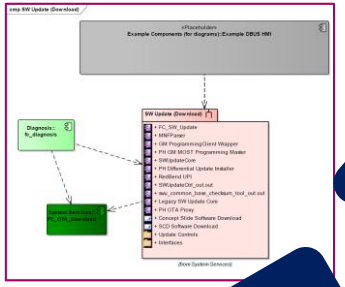


What is Software Architecture
How to build a software product – Architectural Views



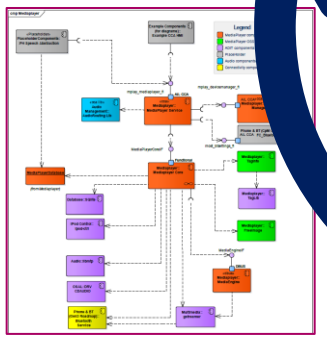
▶ Context view

Embedding of SW systems (as black box) in its environment, interfaces to neighboring systems (e.g. through different communication channels) (Relation with new E/E architectures).

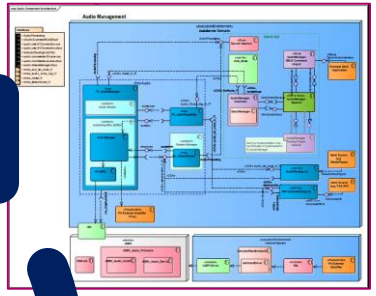


▶ Component view (+interfaces)

Static (hierarchical) composition of the SW system consisting of architectural building blocks, subsystems, SW components and their interfaces.

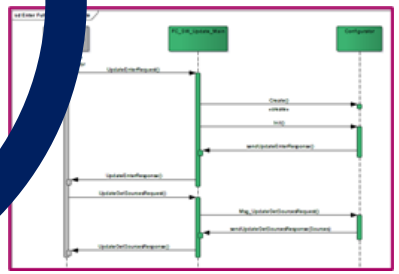


+ change
Implementation



▶ Deployment view

Environment in which the SW is running: HW components running the SW, processors, network topologies and protocols, as well as further physical components of the system environment. The component view within the environment is optional.



▶ Dynamic runtime view

Description of run time behavior of existing SW elements and their concurrence. Dynamical structures.

What is Software Architecture
What is an architecture

Definition of Architecture (IEEE, 2011-12-01):

> ***architecture*** *<system> fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*

According to this definition architectures describe:

- fundamental **concepts** on which corresponding systems are built,
- the **environment** where the system under design need to be integrated into,
- **components** which the system consists of, and
- **relations** between the components and the environment.

What is Software Architecture
Quiz time

▶ Select the **three most often used** architectural views:

(a) Physical database view

(b) Context view

(c) Building Block/Component view

(d) Test-driven view

(e) Configuration view

(f) Runtime view

What is Software Architecture
Quiz time

▶ Select the **three most often used** architectural views:

(a) Physical database view

(b) Context view

(c) Building Block/Component view

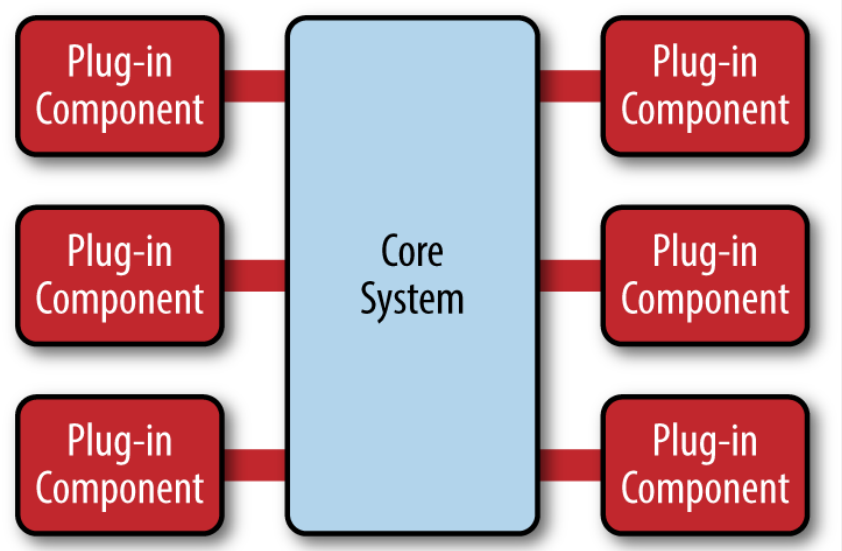
(d) Test-driven view

(e) Configuration view

(f) Runtime view

SOFTWARE ARCHITECTURAL PATTERN

Software Architectural Pattern
Microkernel Architecture



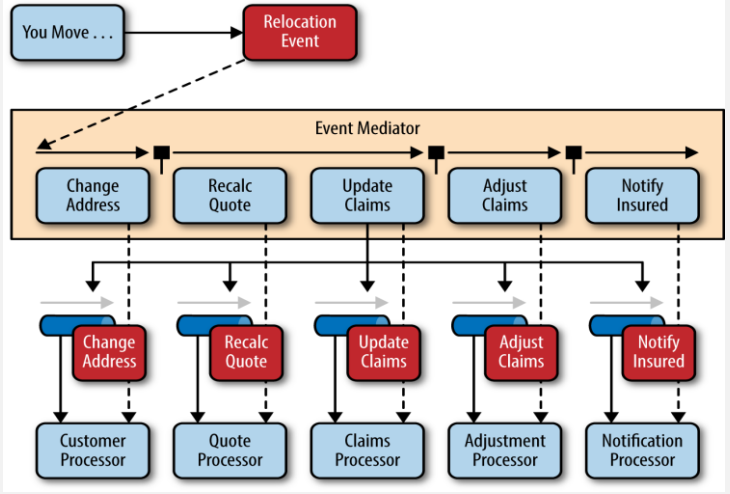
Overall agility	High	Performance	High
Ease of deployment	High	Scalability	Low
Testability	High	Ease of development	Low

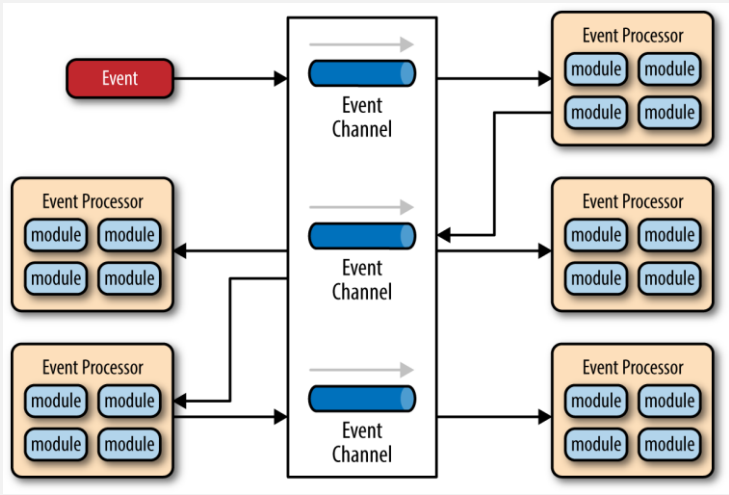
The rating for each characteristic is based on the natural tendency for that characteristic as a capability based on a typical implementation of the pattern, as well as what the pattern is generally known for.

Key concept:

- A core system and plug-in modules
- Plug-in registry
- Minimal interface between plug-in modules
- Toward evolutionary design and incremental development
- “Product” nature

Software Architectural Pattern
Event-Driven Architecture





Overall agility	High	Performance	High
Ease of deployment	High	Scalability	High
Testability	Low	Ease of development	Low

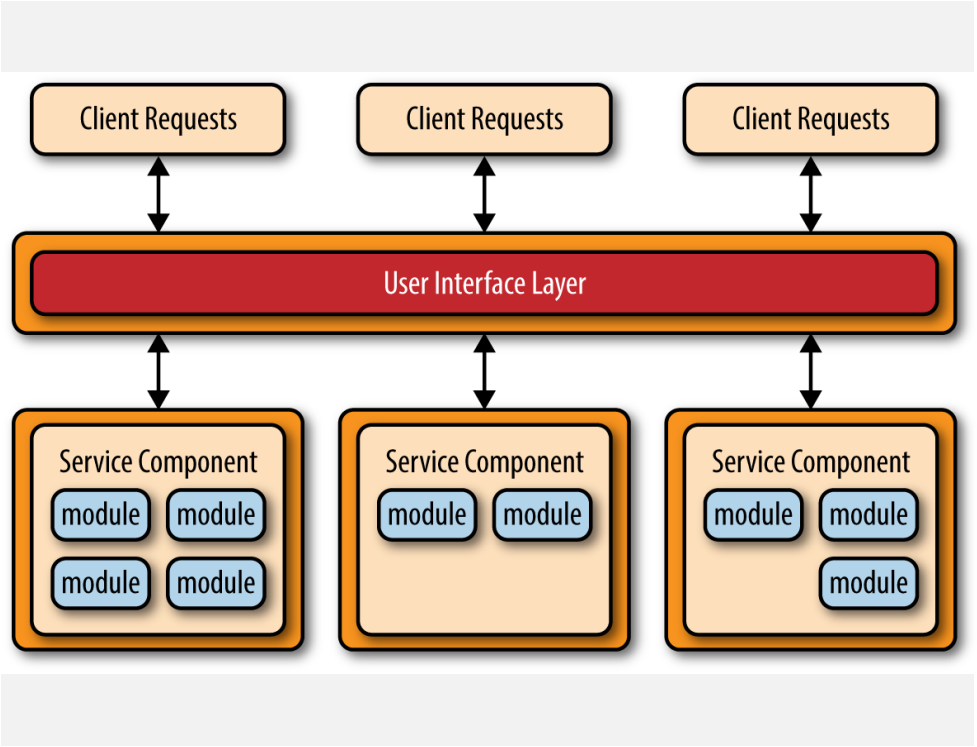
The rating for each characteristic is based on the natural tendency for that characteristic as a capability based on a typical implementation of the pattern, as well as what the pattern is generally known for.

Key concept:

- Highly decoupled, single-purpose event processing components
- Mediator or Broker topology
- “Asynchronous” nature

Bosch Global Software Technologies
alt_future

Software Architectural Pattern
Microservices Architecture



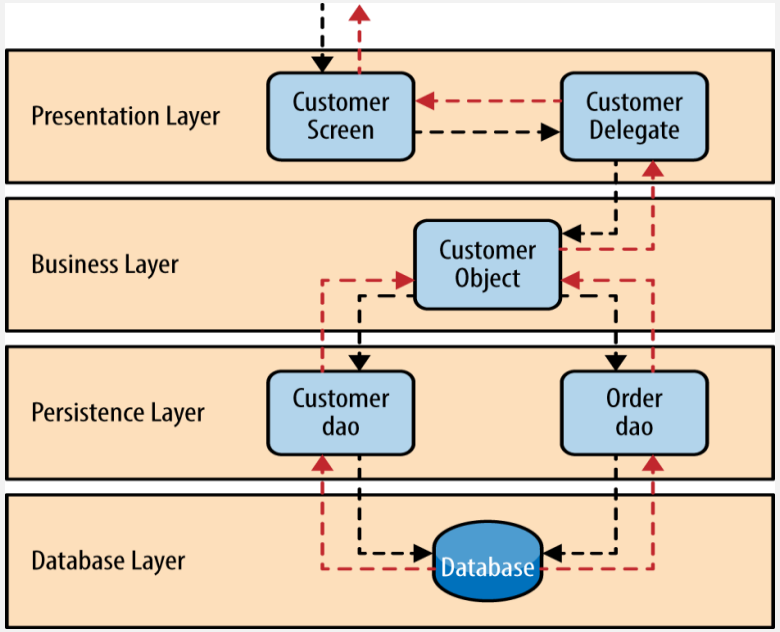
Overall agility	High	Performance	Low
Ease of deployment	High	Scalability	High
Testability	High	Ease of development	High

The rating for each characteristic is based on the natural tendency for that characteristic as a capability based on a typical implementation of the pattern, as well as what the pattern is generally known for.

Key concept:

- Separately deployable units i.e., service components
- Granularity of service component
- Dependencies and orchestration
- “Distributed” nature

Software Architectural Pattern
Layered Architecture



Overall agility	Low	Performance	Low
Ease of deployment	Low	Scalability	Low
Testability	High	Ease of development	High

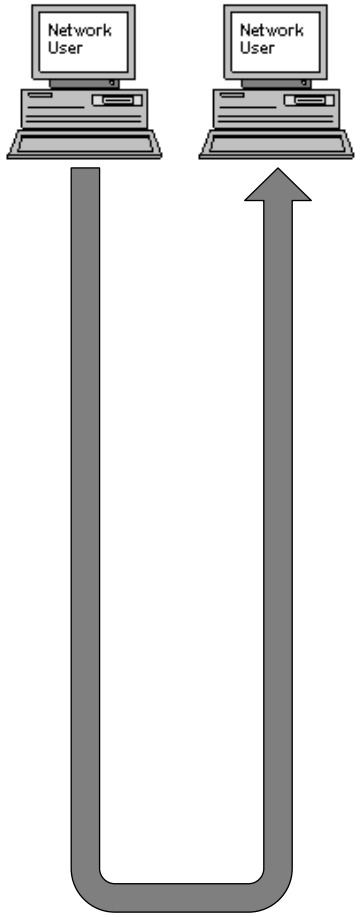
The rating for each characteristic is based on the natural tendency for that characteristic as a capability based on a typical implementation of the pattern, as well as what the pattern is generally known for.

Key concept:

- Horizontal layers
- Separation of concerns
- Layers are “closed”
- “Monolith” nature

Software Architectural Pattern
OSI – Open Systems Interconnection

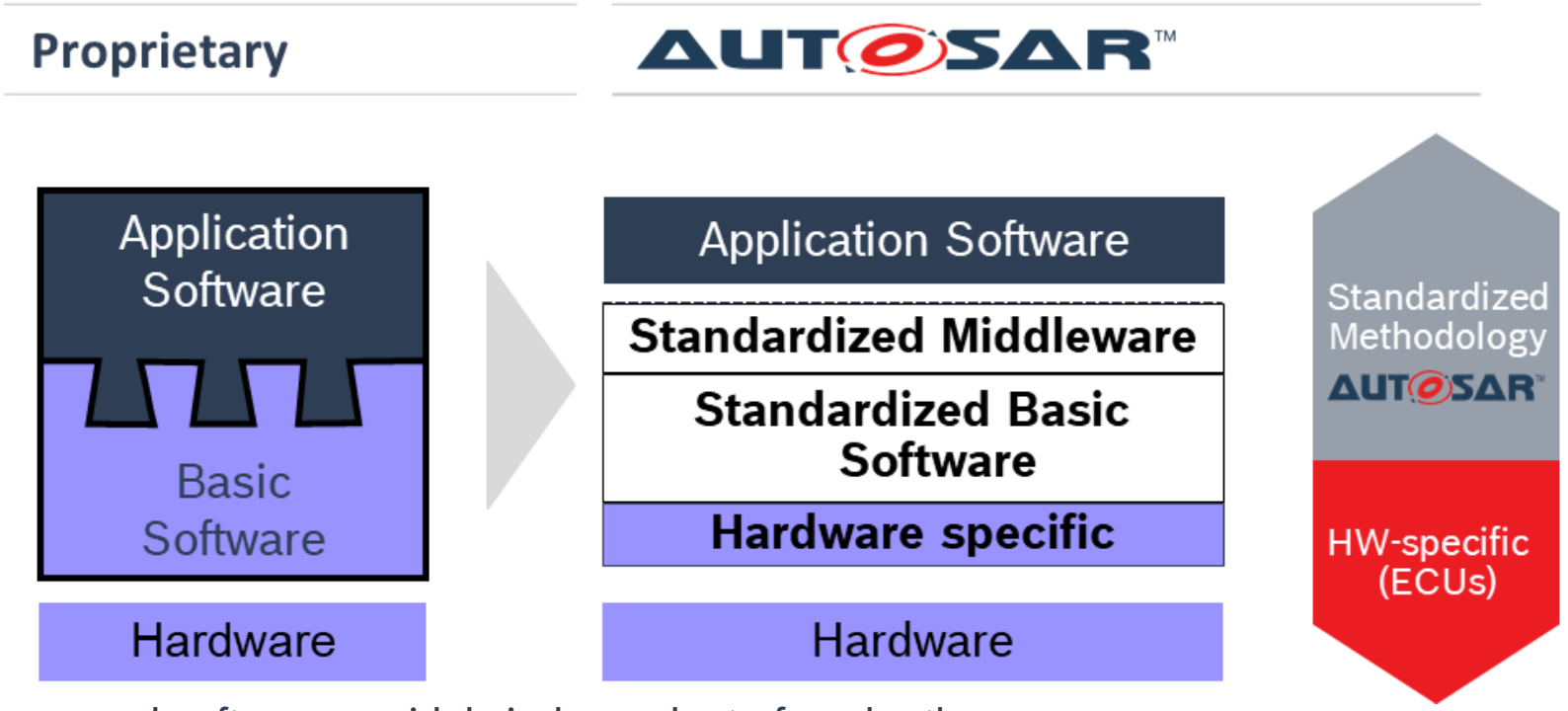
Host layers	7	**Application**	High-level protocols such as for resource sharing or remote file access, e.g. HTTP.
	6	**Presentation**	Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption.
	5	**Session**	Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes.
	4	**Transport**	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing.
Media layers	3	**Network**	Structuring and managing a multi-node network, including addressing, routing and traffic control.
	2	**Data Link**	Transmission of data frames between two nodes connected by a physical layer.
	1	**Physical**	Transmission and reception of raw bit streams over a physical medium.



ISO/IEC 7498

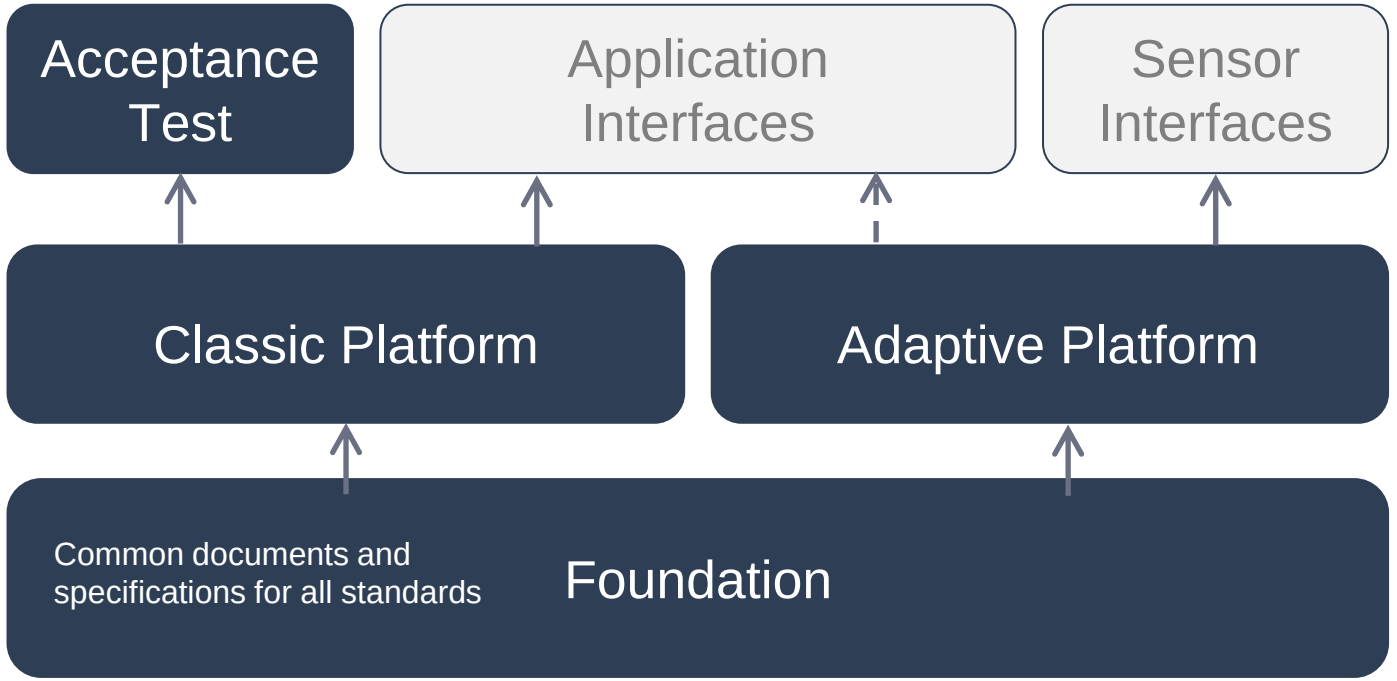
WHAT IS AUTOSAR?

What is AUTOSAR
Aims and benefits of using AUTOSAR



- Hardware and software – widely independent of each other.
- Development can be de-coupled (through abstraction) by horizontal layers, reducing development time and costs.
- Reuse of software, enhances quality and efficiency

What is AUTOSAR
AUTOSAR Deliverables

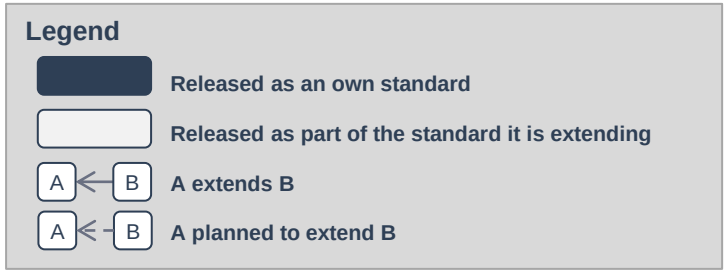


AUTOSAR SVN copy @Bosch:
file:////si8256.de.bosch.com/AUTOSAR$/SVN3-COPY/26_Standards/02_Releases/

AUTOSAR Docupedia @Bosch:
https://inside-docupedia.bosch.com/confluence/display/AUT

Most common type of deliverables

- ATS: Acceptance Test Specification
- CONC: Concept document
- EXP: Explanation document
- MMOD: Meta-model files (M2)
- MOD: Model files (M1)
- PRS: Protocol Specification
- RS/SRS: Requirement Specification
- SWS: Software Specification
- TPS: Template Specification
- TR: Technical Report



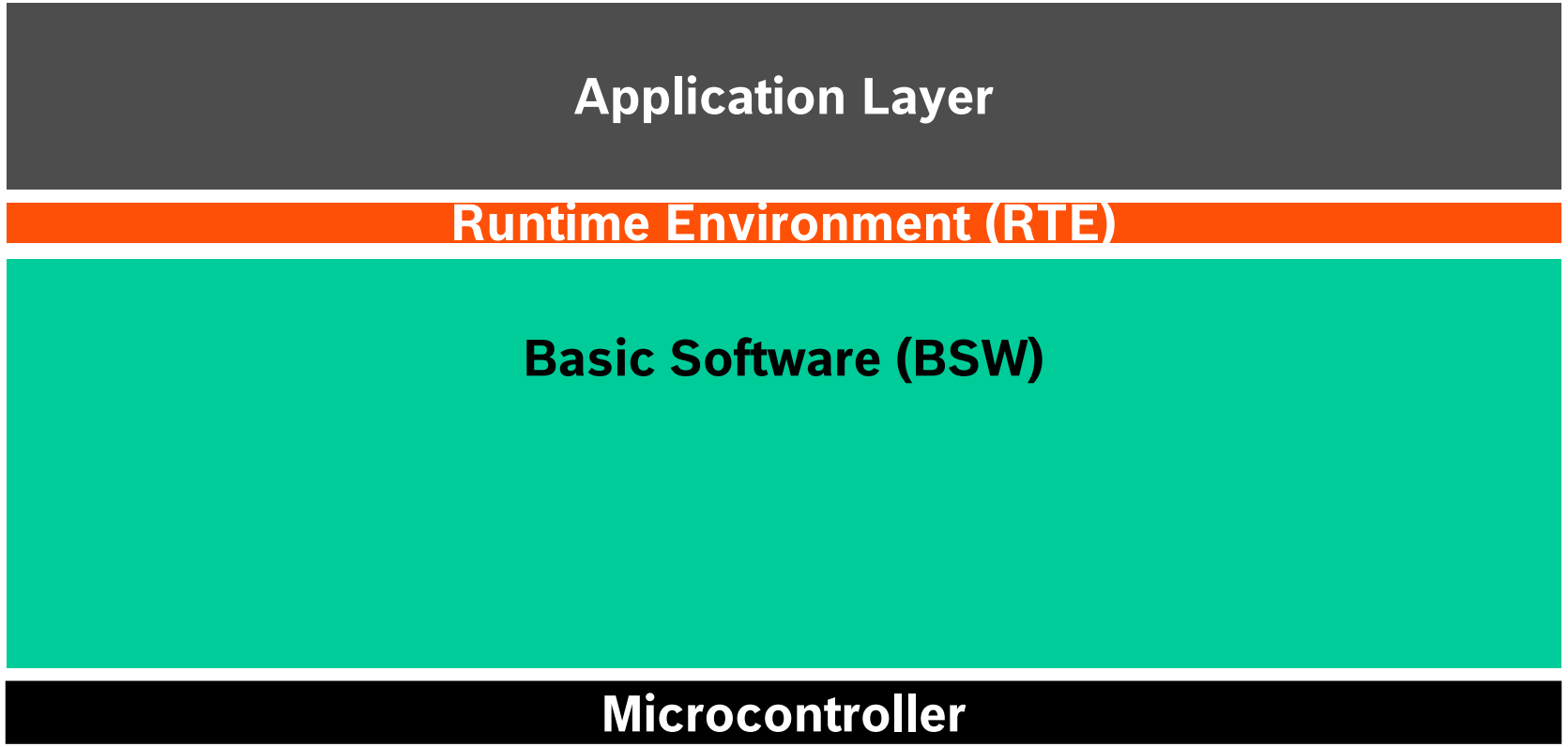
Bosch Global Software Technologies
alt_future

AUTOSAR LAYERED Architecture

AUTOSAR Layered Architecture

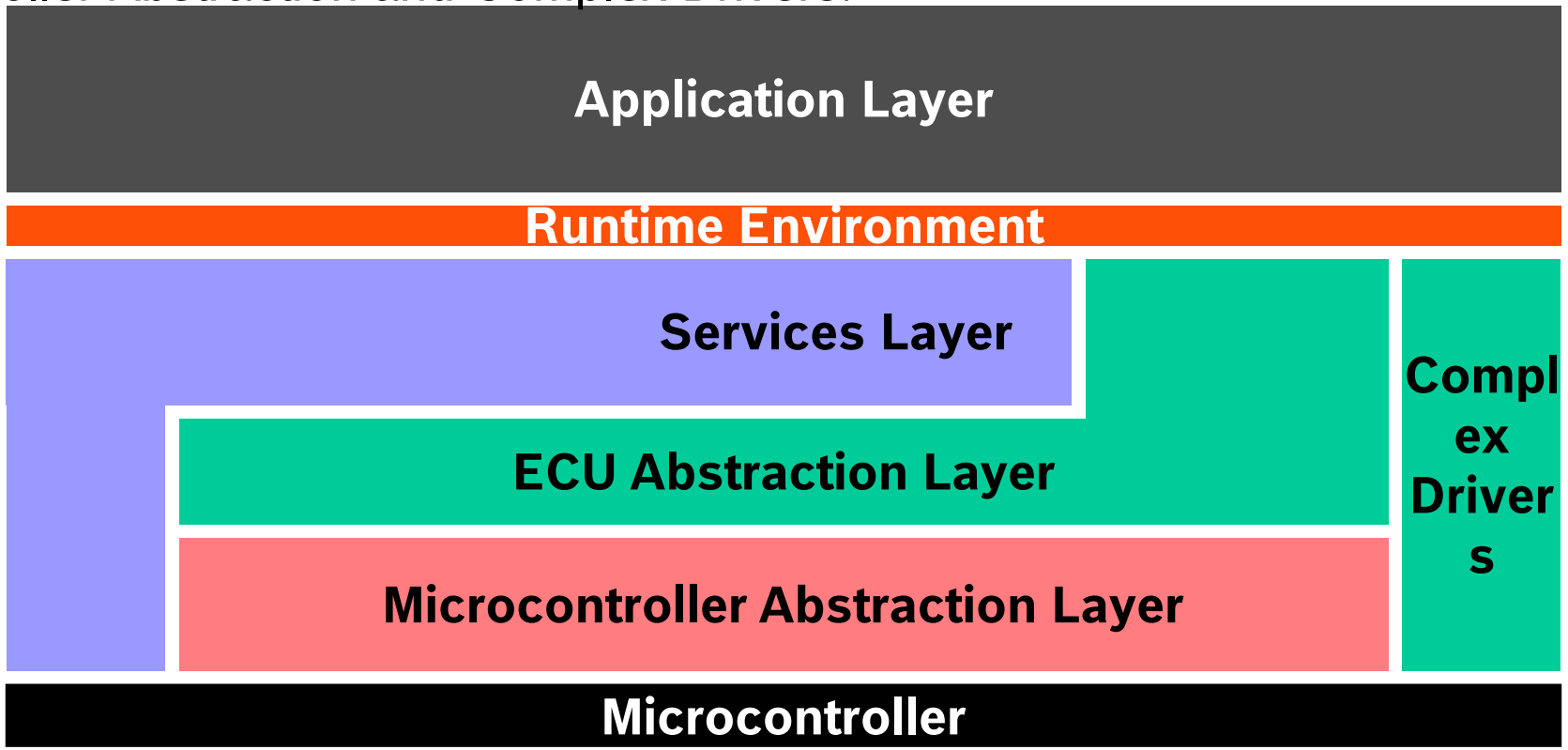
Top View

The AUTOSAR Architecture distinguishes on the highest abstraction level between three software layers: Application, Runtime Environment and Basic Software which run on a Microcontroller.



AUTOSAR Layered Architecture
Coarse view

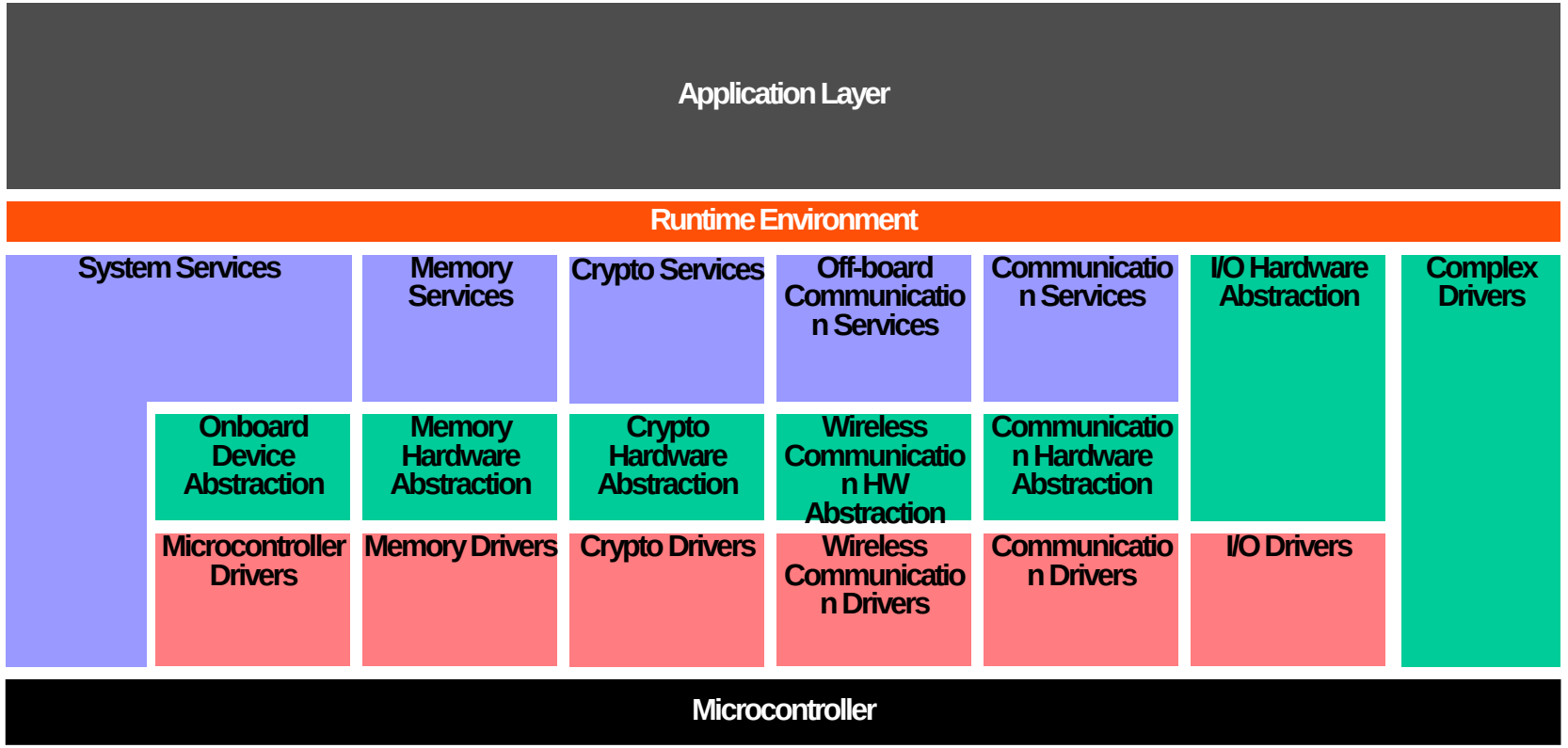
The AUTOSAR Basic Software is further divided in the layers: Services, ECU Abstraction, Microcontroller Abstraction and Complex Drivers.



AUTOSAR Layered Architecture

Detailed view

The Basic Software Layers are further divided into functional groups. Examples of Services are System, Memory and Communication Services.



AUTOSAR Layered Architecture
Microcontroller Abstraction Layer

The **Microcontroller Abstraction Layer** is the lowest software layer of the Basic Software.

It contains internal drivers, which are software modules with direct access to the µC and internal peripherals.

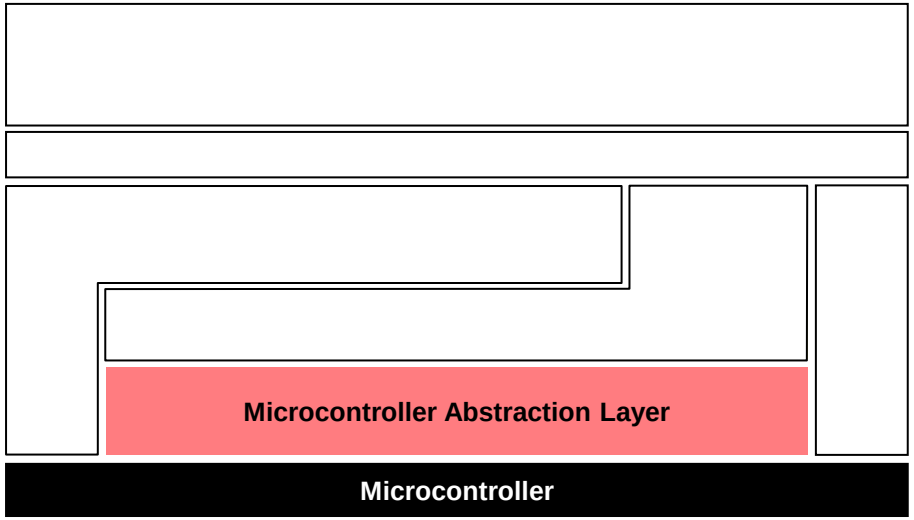
Task

Make higher software layers independent of µC

Properties

Implementation: µC dependent

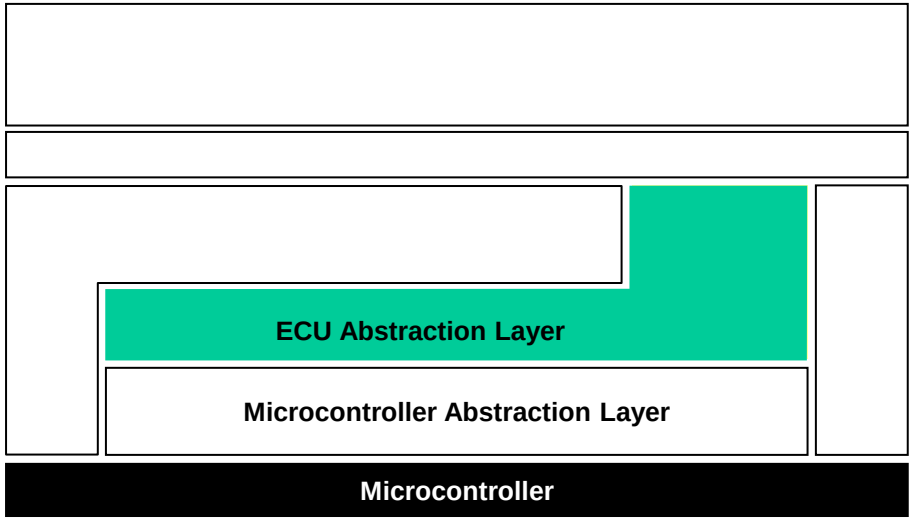
Upper Interface: standardized and µC independent



AUTOSAR Layered Architecture
ECU Abstraction Layer

The **ECU Abstraction Layer** interfaces the drivers of the Microcontroller Abstraction Layer. It also contains drivers for external devices.

It offers an API for access to peripherals and devices regardless of their location (µC internal/external) and their connection to the µC (port pins, type of interface)



Task

Make higher software layers independent of ECU hardware layout

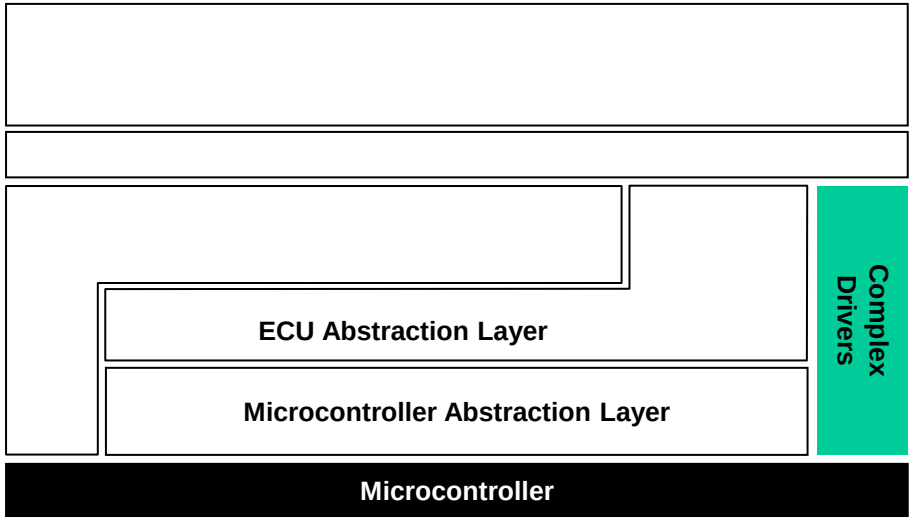
Properties

Implementation: µC independent, ECU hardware dependent

Upper Interface: µC and ECU hardware independent

AUTOSAR Layered Architecture
Complex Drivers

The **Complex Drivers Layer** spans from the hardware to the RTE.



Task

Provide the possibility to integrate special purpose functionality, e.g. drivers for devices:

- which are not specified within AUTOSAR,
- with very high timing constrains or
- for migration purposes etc.

Properties

Implementation: might be application, µC and ECU hardware dependent

Upper Interface: might be application, µC and ECU hardware dependent

AUTOSAR Layered Architecture

Services Layer

The **Services Layer** is the highest layer of the Basic Software which also applies for its relevance for the application software: while access to I/O signals is covered by the ECU Abstraction Layer, the Services Layer offers:

- Operating system functionality
- Vehicle network communication and management services
- Memory services (NVRAM management)
- Diagnostic Services (including UDS communication, error memory and fault treatment)
- ECU state management, mode management
- Logical and temporal program flow monitoring (Wdg manager)

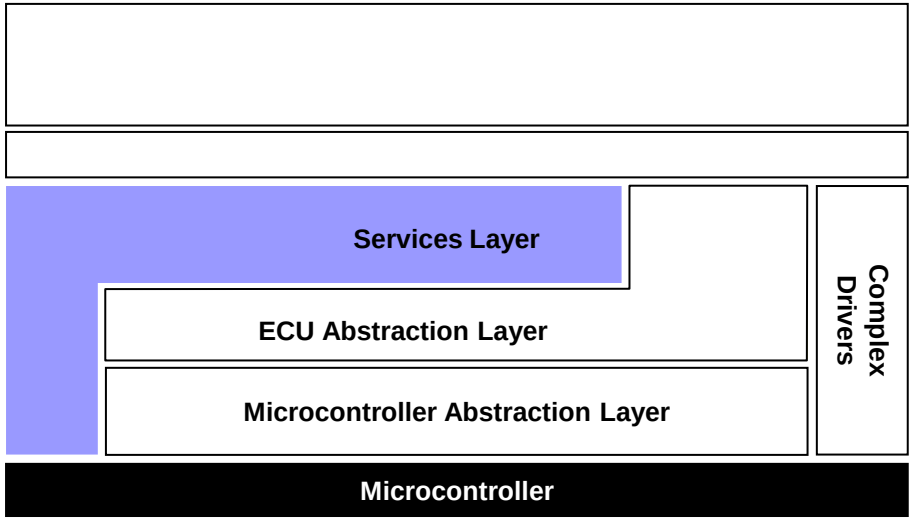
Task

Provide basic services for applications, RTE and basic software modules.

Properties

Implementation: mostly µC and ECU hardware independent

Upper Interface: µC and ECU hardware independent



AUTOSAR Layered Architecture
AUTOSAR Runtime Environment (RTE)

The **RTE** is a layer providing communication services to the application software (AUTOSAR Software Components and/or AUTOSAR Sensor/Actuator components).

Above the RTE the software architecture style changes from “layered“ to “component style“.

The AUTOSAR Software Components communicate with other components (inter and/or intra ECU) and/or services via the RTE.

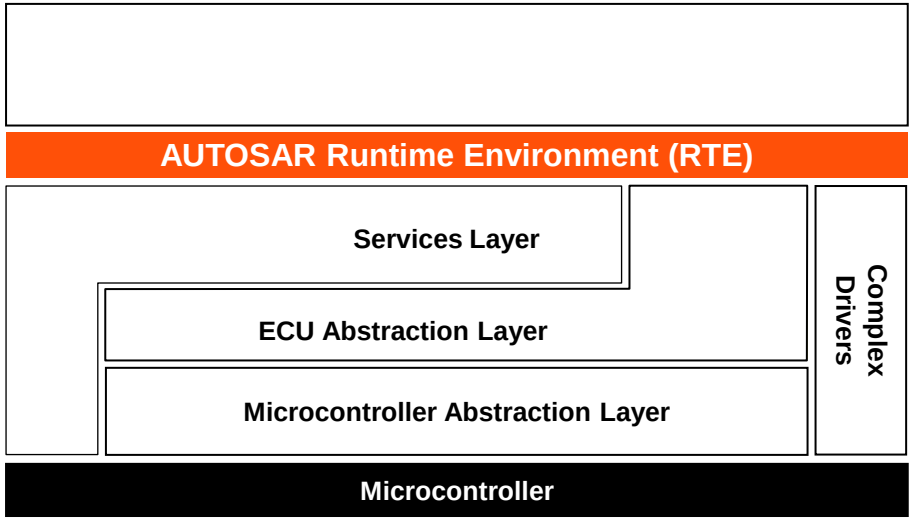
Task

Make AUTOSAR Software Components independent from the mapping to a specific ECU.

Properties

Implementation: ECU and application specific (generated individually for each ECU)

Upper Interface: completely ECU independent



AUTOSAR Basic Software
Quiz time

- **From the top to bottom, how many software layers of the highest abstraction level of AUTOSAR architecture?**

(a) 3 layers: Application, RTE, BSW.

(b) 5 layers: Application, RTE, Service layer, ECU Abstraction layer, MCAL.

(c) 3 layers: Services layer, Abstraction layer, MCAL.

(d) 4 layers: Application, RTE, BSW, MCAL.

RTE = Runtime Environment

BSW = Basic Software

MCAL = Microcontroller Abstraction Layer

AUTOSAR Basic Software
Quiz time

- **From the top to bottom, how many software layers of the highest abstraction level of AUTOSAR architecture?**

(a) 3 layers: Application, RTE, BSW.

(b) 5 layers: Application, RTE, Service layer, ECU Abstraction layer, MCAL.

(c) 3 layers: Services layer, Abstraction layer, MCAL.

(d) 4 layers: Application, RTE, BSW, MCAL.

RTE = Runtime Environment

BSW = Basic Software

MCAL = Microcontroller Abstraction Layer

AUTOSAR Layered Architecture
Quiz time

- **Which of the following qualities can most likely be improved by using a layered architecture?**

(a) Runtime efficiency (performance).

(b) Flexibility in modifying or changing the system.

(c) Flexibility at runtime (configurability).

(d) Non-repudiability.

AUTOSAR Layered Architecture
Quiz time

- **Which of the following qualities can most likely be improved by using a layered architecture?**

(a) Runtime efficiency (performance).

(b) Flexibility in modifying or changing the system.

(c) Flexibility at runtime (configurability).

(d) Non-repudiability.

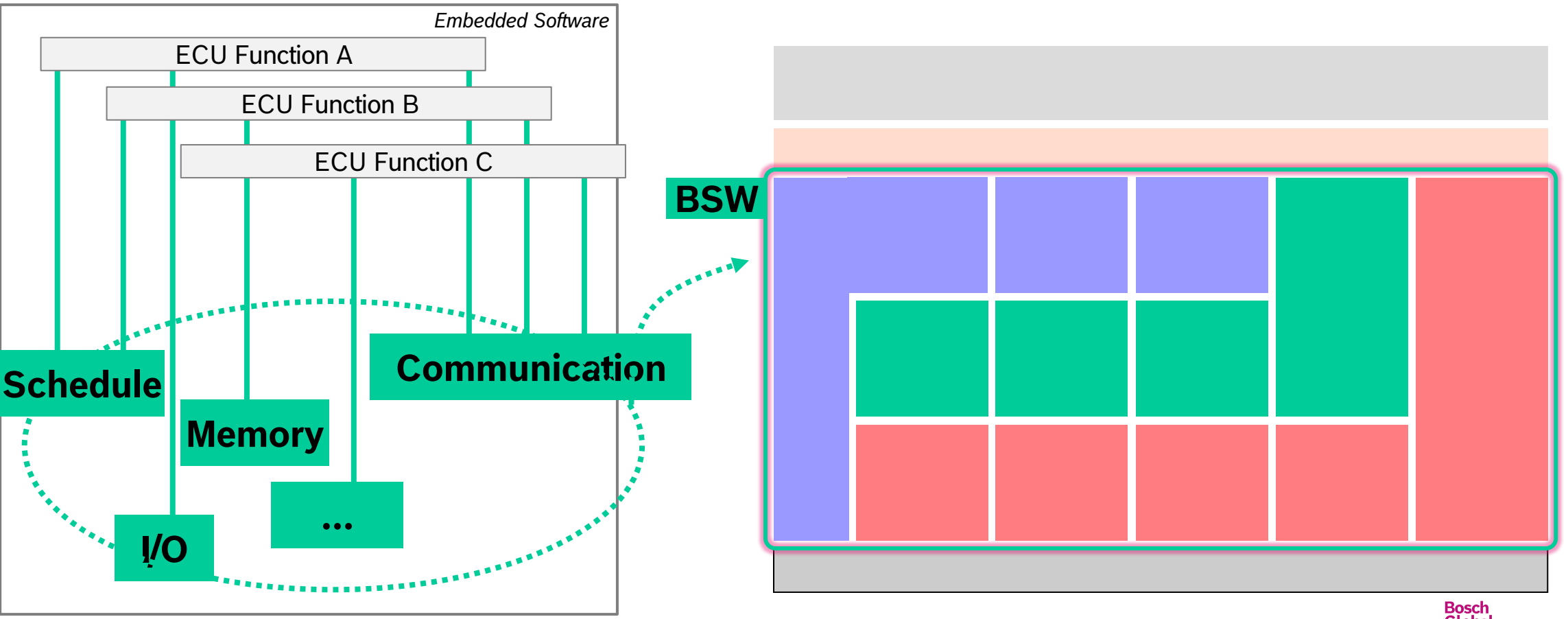
AUTOSAR BASIC SOFTWARE

AUTOSAR Basic Software
What is “Basic Software”?

Basic /ˈbeɪ.sɪk/, adj
simple and not complicated, so able to provide the base or starting point from which something can develop
[cambridge.org]

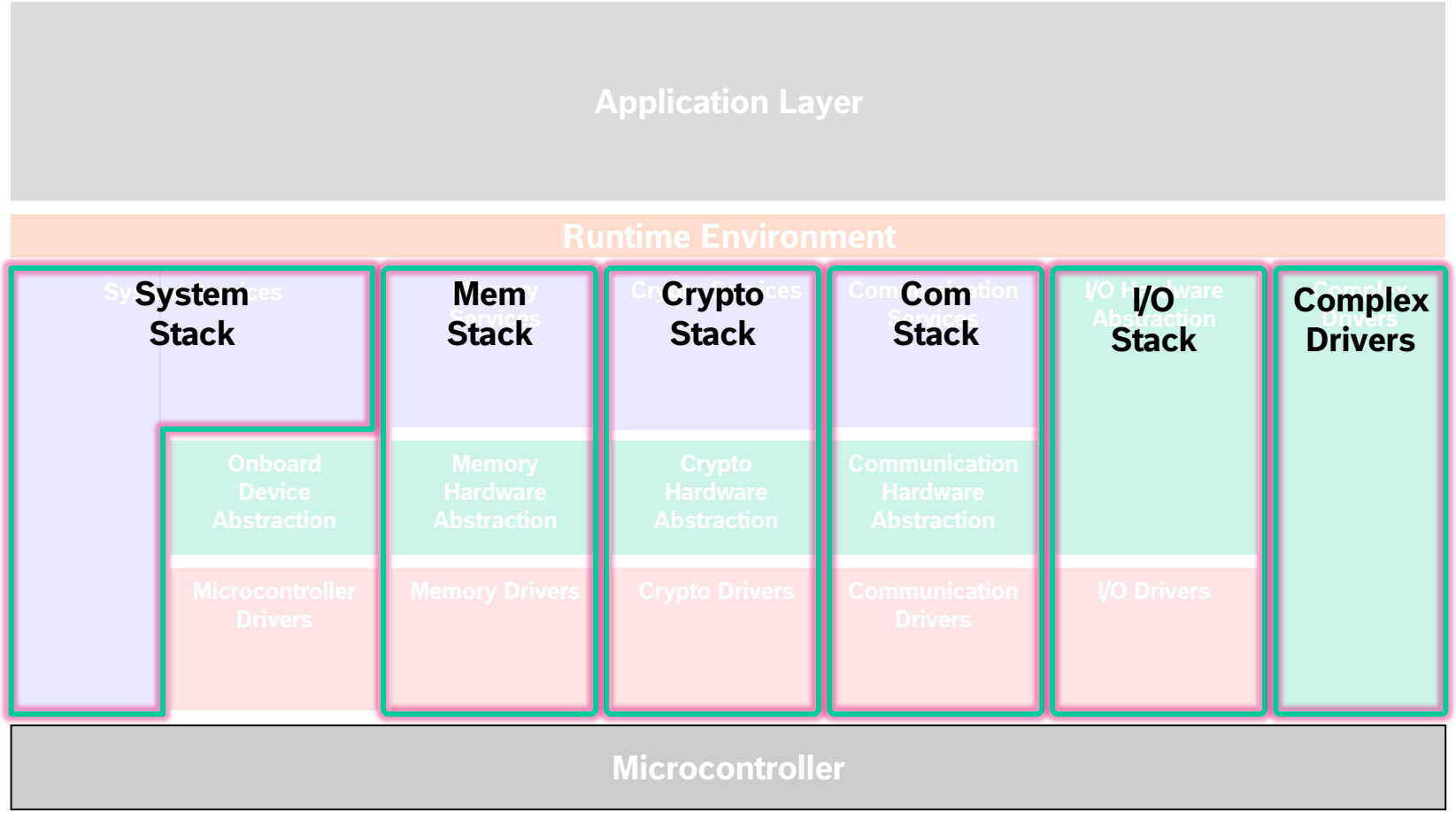
Software /ˈsɑːft.wer/, noun
the instructions that control what a computer does; computer programs
[cambridge.org]

AUTOSAR Basic Software
What is “basic” to Embedded Software?



Bosch Global Software Technologies
alt_future

AUTOSAR Basic Software
BSW Vertical view / Stack view

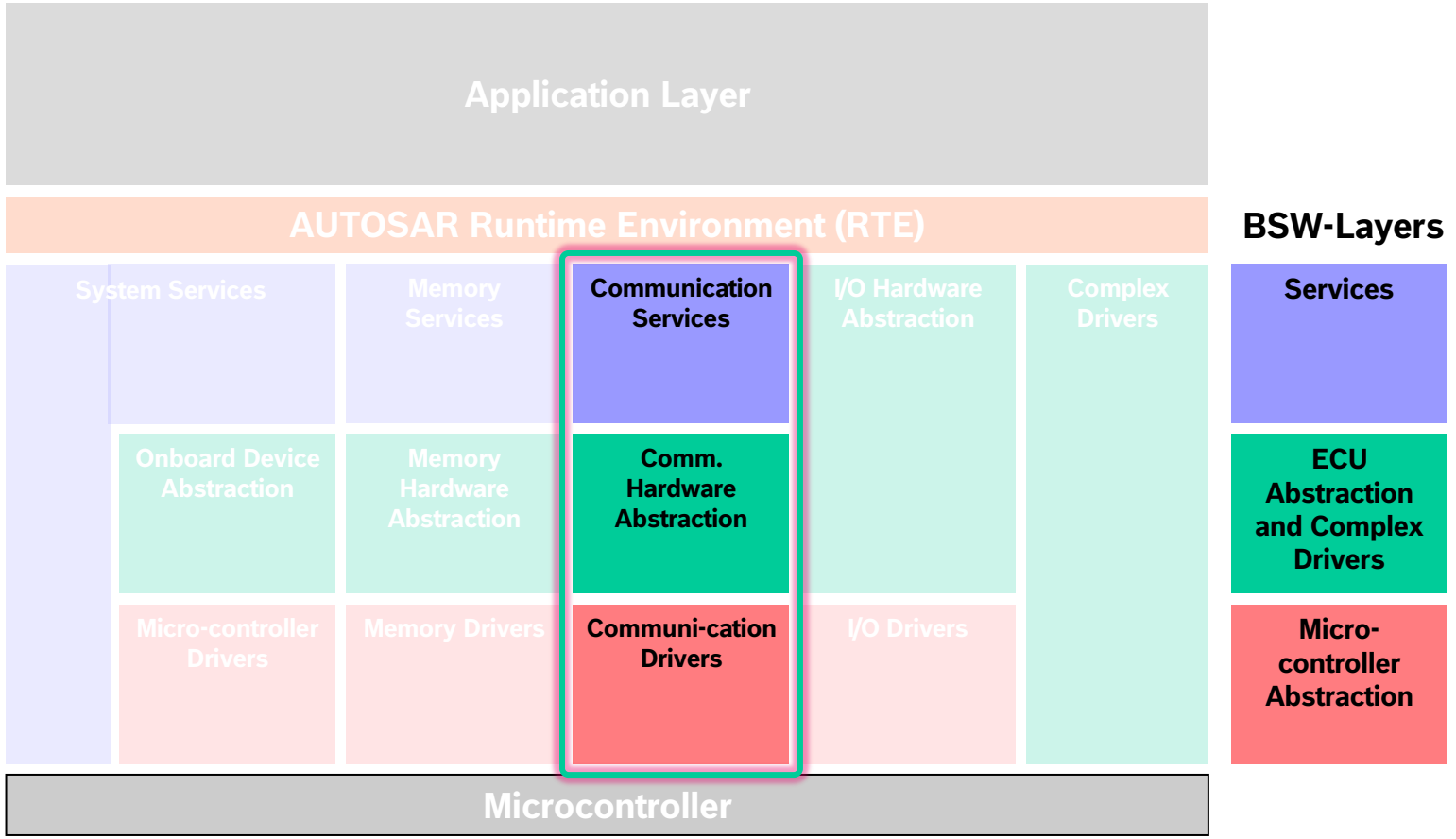


BSW Layers

- Services
- ECU Abstraction and Complex Drivers
- Micro-controller Abstraction

Bosch Global Software Technologies
alt_future

AUTOSAR Basic Software
Communication Stack – Building Blocks



ComStack facilitates vehicle network communication and provides communication services to other BSW components and Application Layer.

ComStack is consisting of Bus-dependent and Bus-independent components. The following bus system are covered by AUTOSAR:

- CAN(TT)
- LIN
- FlexRay
- Ethernet

Different bus system follows the same layered architecture but defines their components distinctly. *CAN is chosen to represent ComStack in this document.*

AUTOSAR Basic Software
Communication Stack – Building Blocks (cont.)

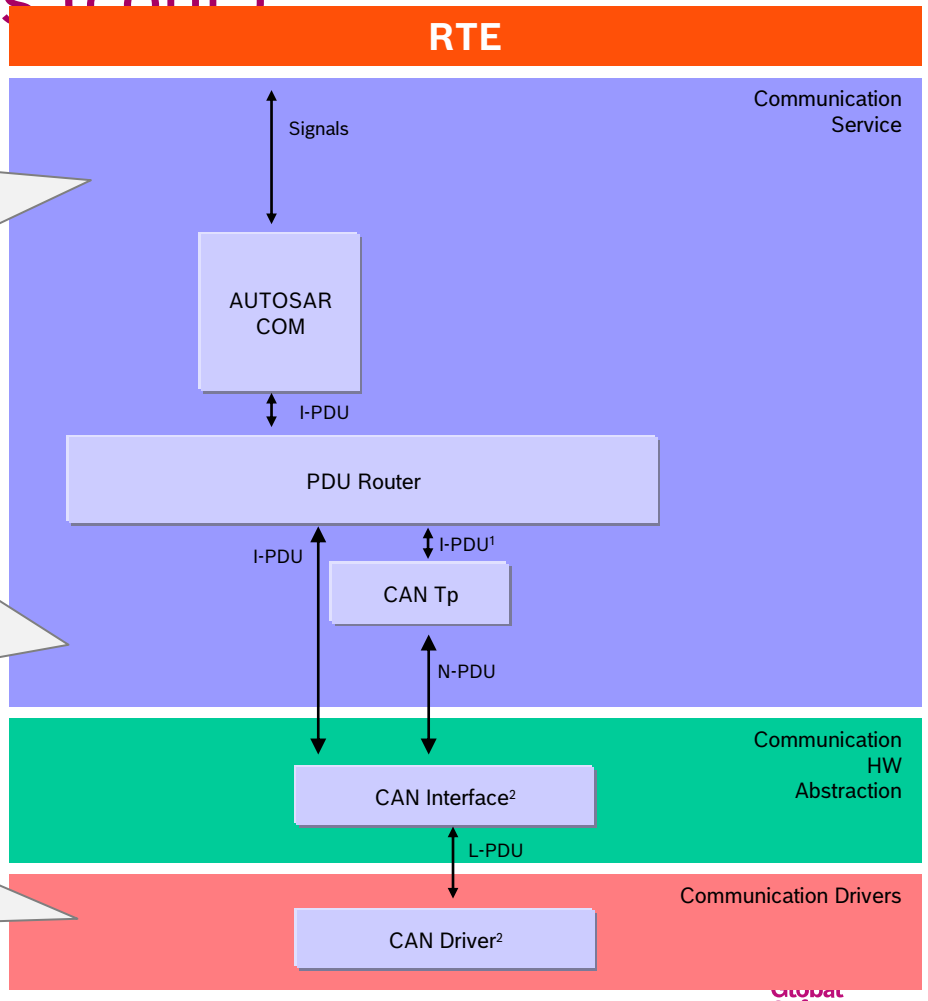
REF

Groups

Communication Services
is a group of modules for vehicle network communication with the communication system CAN.
Task:
- Provide a uniform interface to the CAN network
- Hide protocol and message properties from the application.

Communication Hardware Abstraction
is a group of modules which abstracts from the location of communication controllers and the ECU hardware layout.
- Provide equal mechanisms to access CAN bus channel regardless of it's location (on-chip / on-board)
- µC independent, ECU hardware dependent and external device dependent

Communication Driver
contains internal drivers, which are software modules with direct access to the µC internal peripherals and memory mapped µC external devices.
- Make higher software layers independent of µC



Global Software Technologies alt_future

AUTOSAR Basic Software
Communication Stack – Building Blocks (cont.)

REF

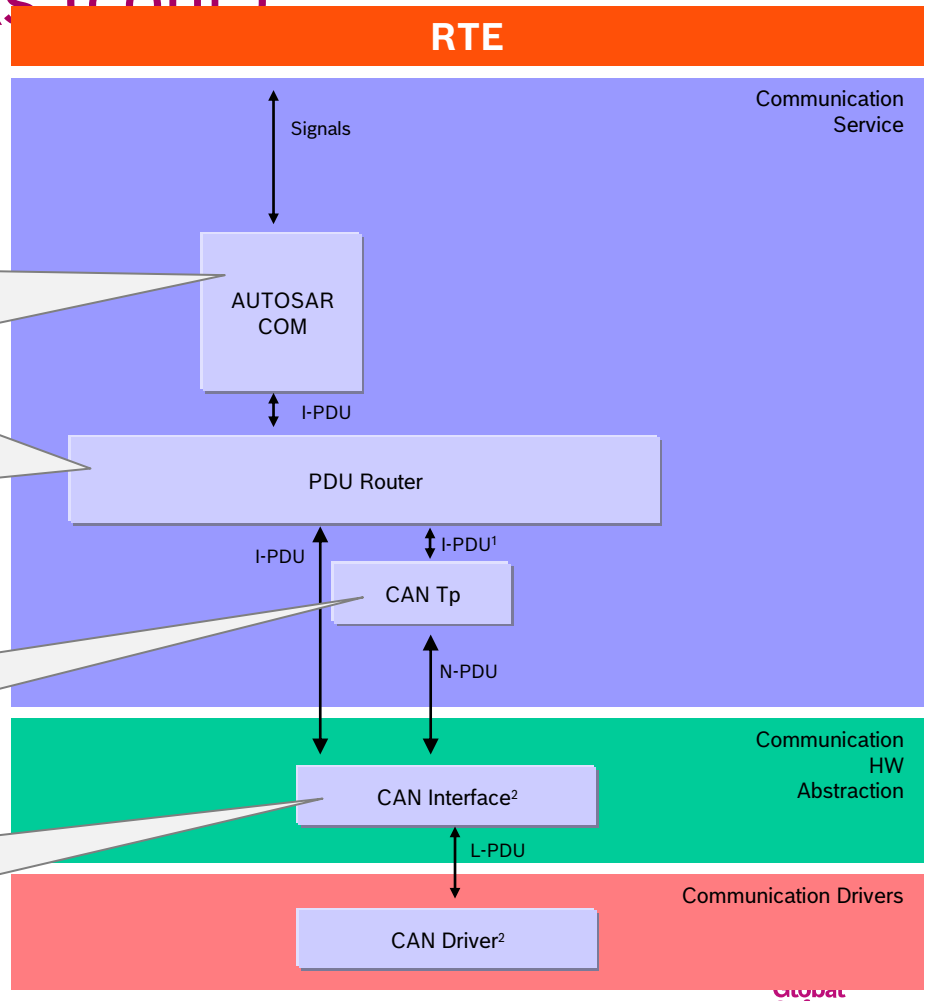
Components

Com
- Provides signal-oriented data interface to the RTE
- Packing/unpacking of AUTOSAR signals to I-PDUs
- Provides routing of individual signals or groups of signals between different I-PDUs

PduR
- Provides routing of PDUs between different abstract communication controllers and upper layers
- Provides TP routing on-the-fly. Transfer of TP data is started before full TP data is buffered

<BusType>Tp
- The main purpose of the TP module is to segment and reassemble (CAN) I-PDUs longer than 8 bytes

<BusType>If
- Provides a unique interface to manage different hardware device types e.g., CAN controllers and CAN transceivers used by the defined ECU hardware layout



PDU = Protocol Data Unit

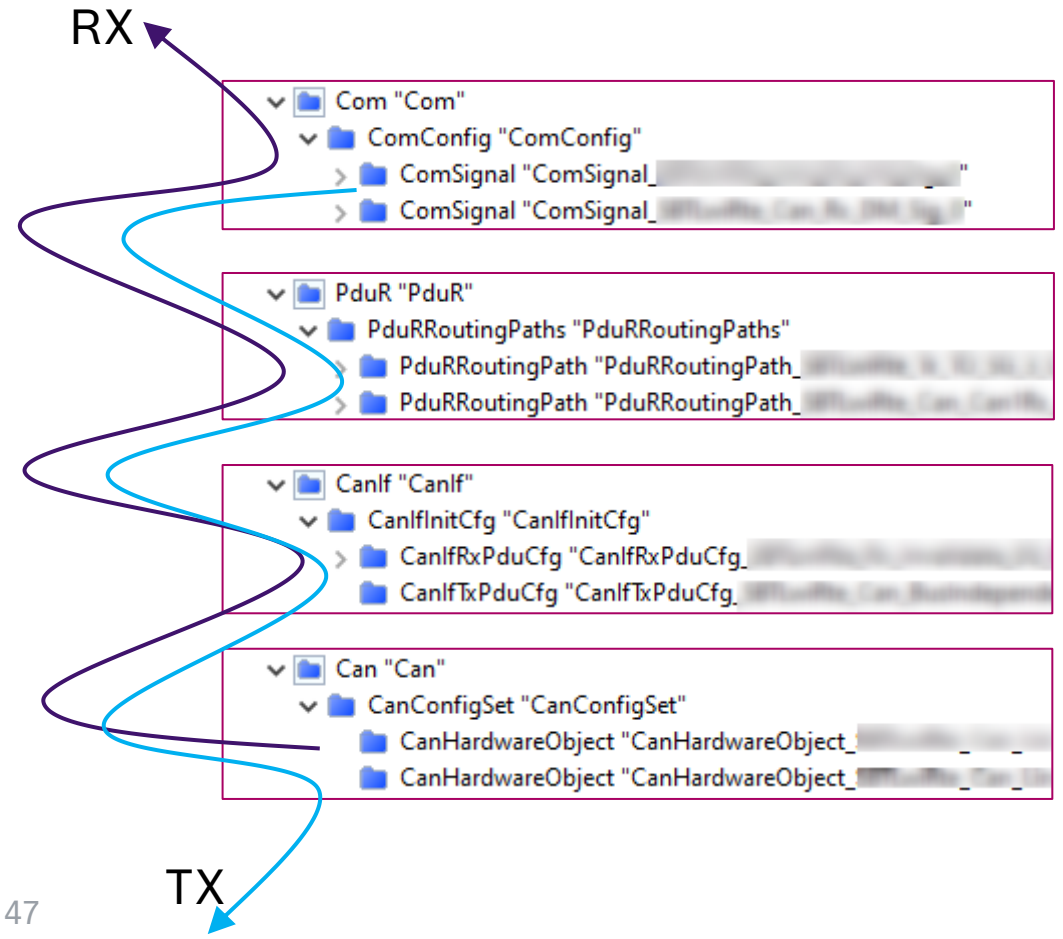
Global Software Technologies alt_future

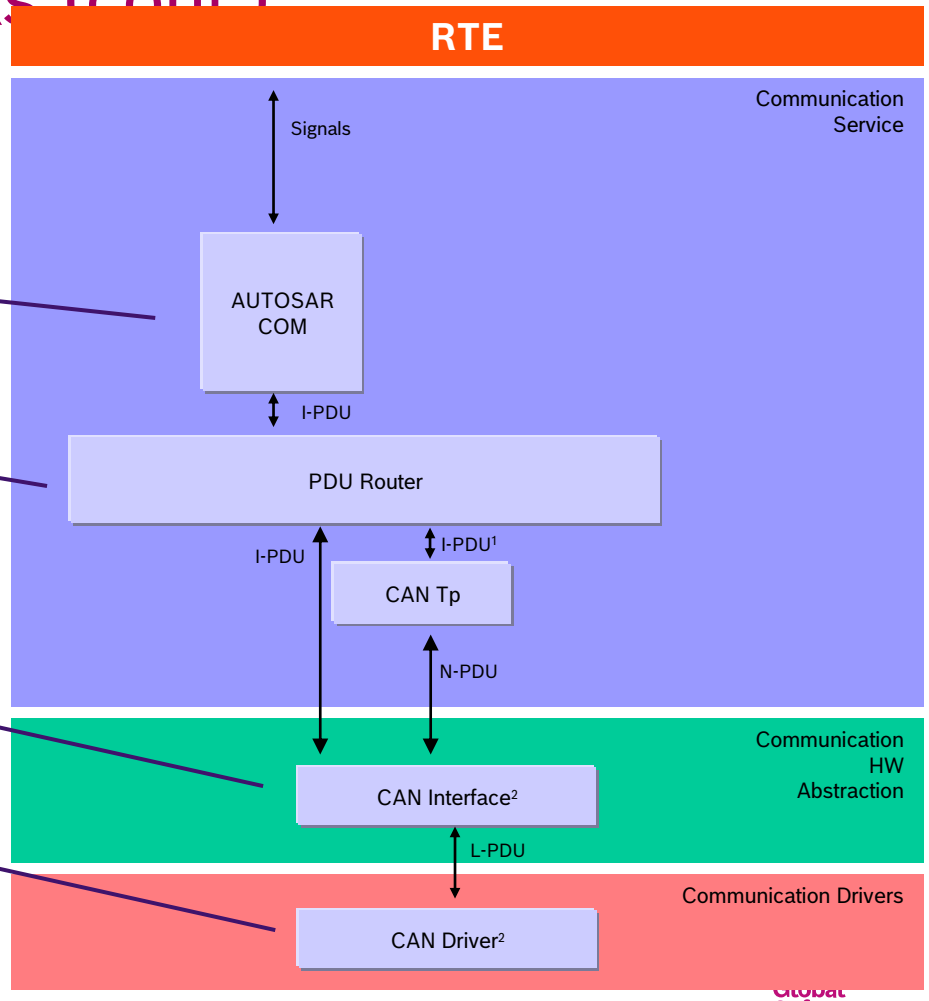
AUTOSAR Basic Software

Communication Stack – Building Blocks (cont.)

REF

AUTOSAR model – ECU Configuration Values





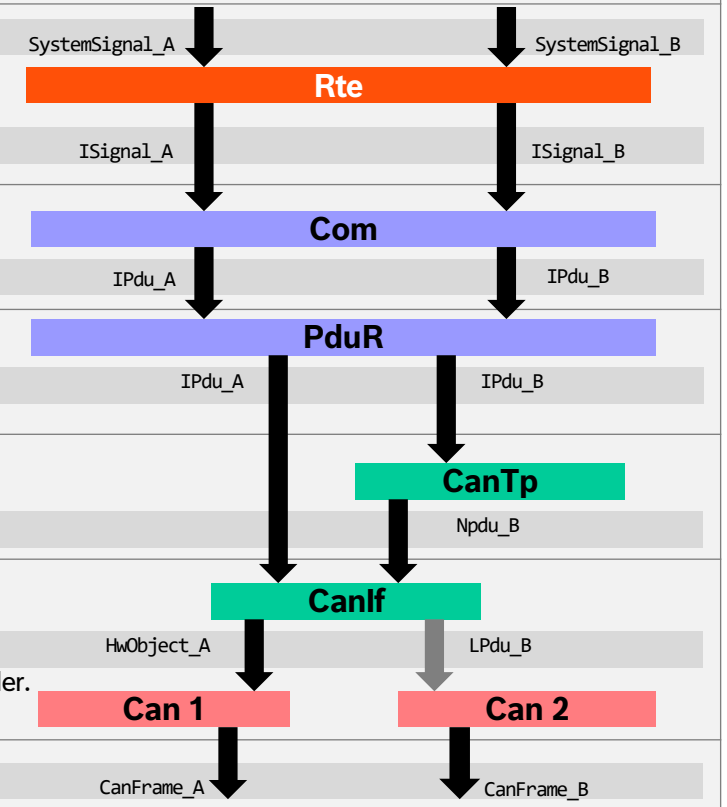
Global Software Technologies alt_future

AUTOSAR Basic Software

REF

Communication Stack – Runtime: transmission and reception

OSI Layer	Layer Prefix	AUTOSAR Modules	PDU Name	
7: Application	Not in scope of BSW			Generic interface at VFB. Data exchanged as System Signals. Generic interface. Data exchanged as I-Signals.
6: Presentation (Interaction)	I	COM, DCM	I-PDU	Generic interface. Data exchanged as I-PDUs.
	I	PDU router, PDU multiplexer	I-PDU	Bus Type dependent interface. Data exchanged as I-PDUs.
3: Network	N	TP Layer	N-PDU	Can Transport Protocol dependent interface. Data exchanged as N-PDUs.
2: Data Link	L	Driver, Interface	L-PDU	Can Driver Instance dependent interface e.g. rba_Can_MCan_Write. Data exchanged as L-PDU and HW Object handler.
1: Physical	Microcontroller, Transceiver, Bus medium			Can Frames on physical network



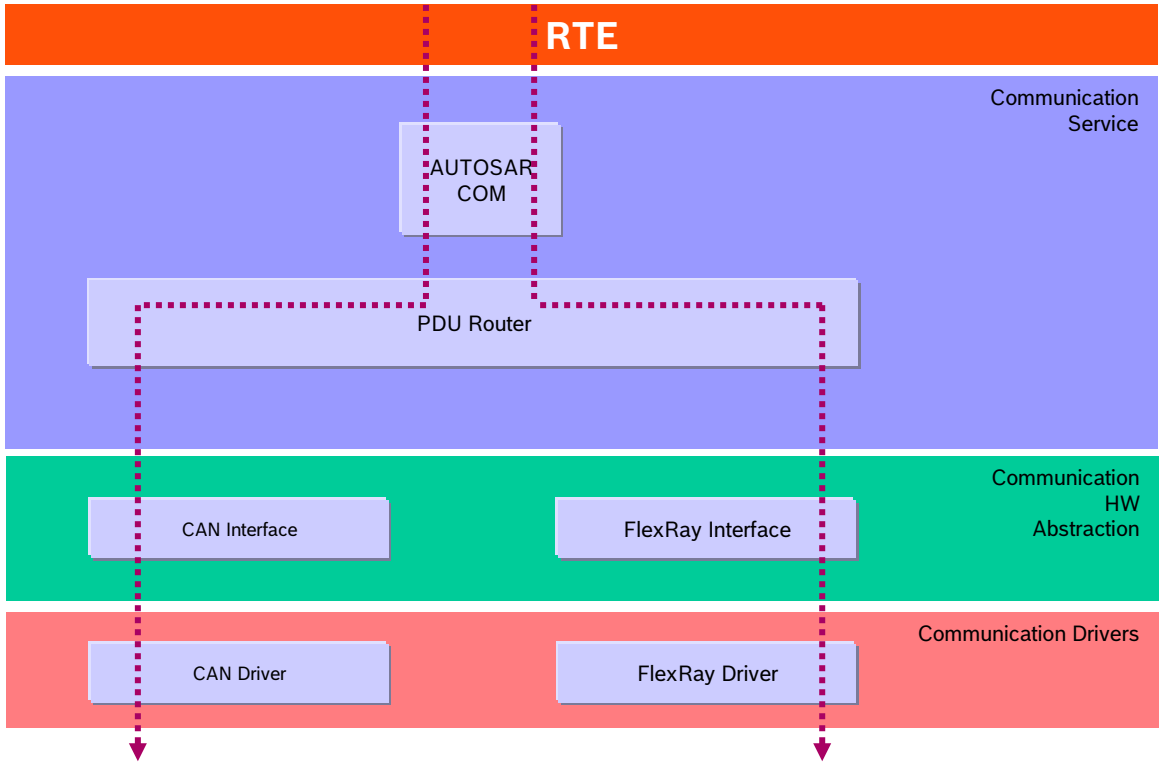
Global Software Technologies alt_future

AUTOSAR Basic Software

Communication Stack – Building Blocks (cont.)

REF

Transmission



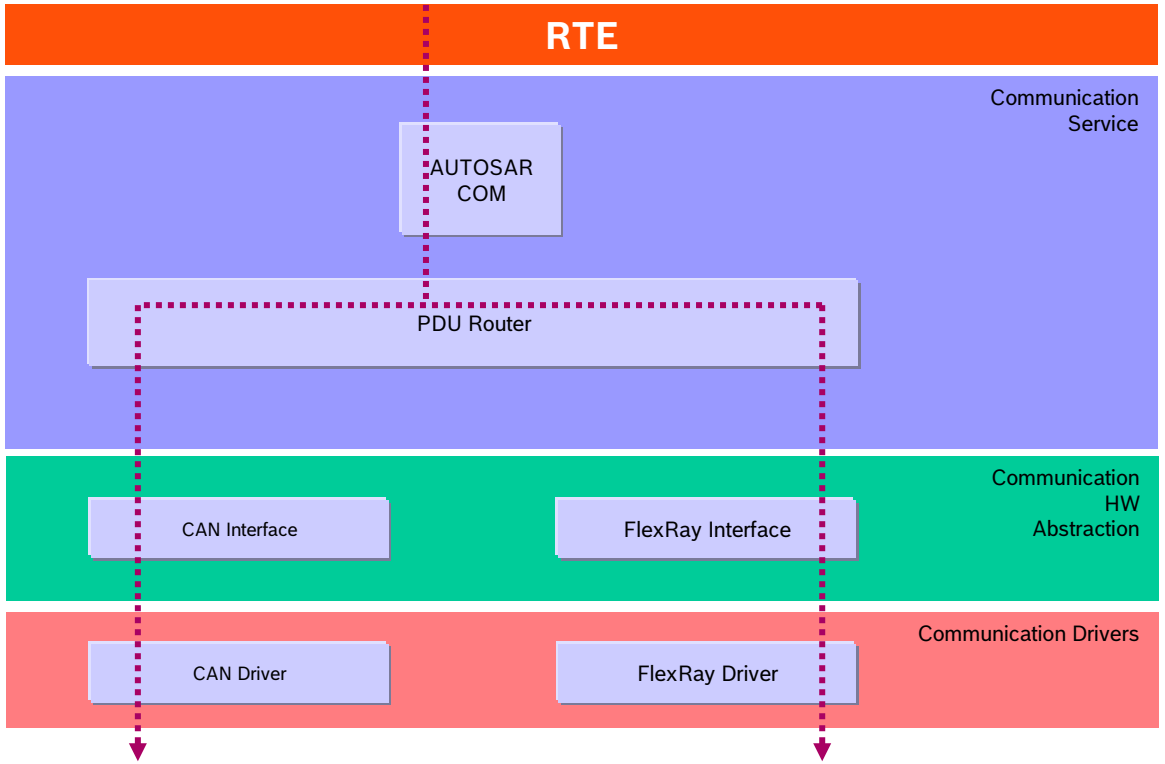
Bosch Global Software Technologies
alt_future

AUTOSAR Basic Software

Communication Stack – Building Blocks (cont.)

REF

Transmission (fan-out)



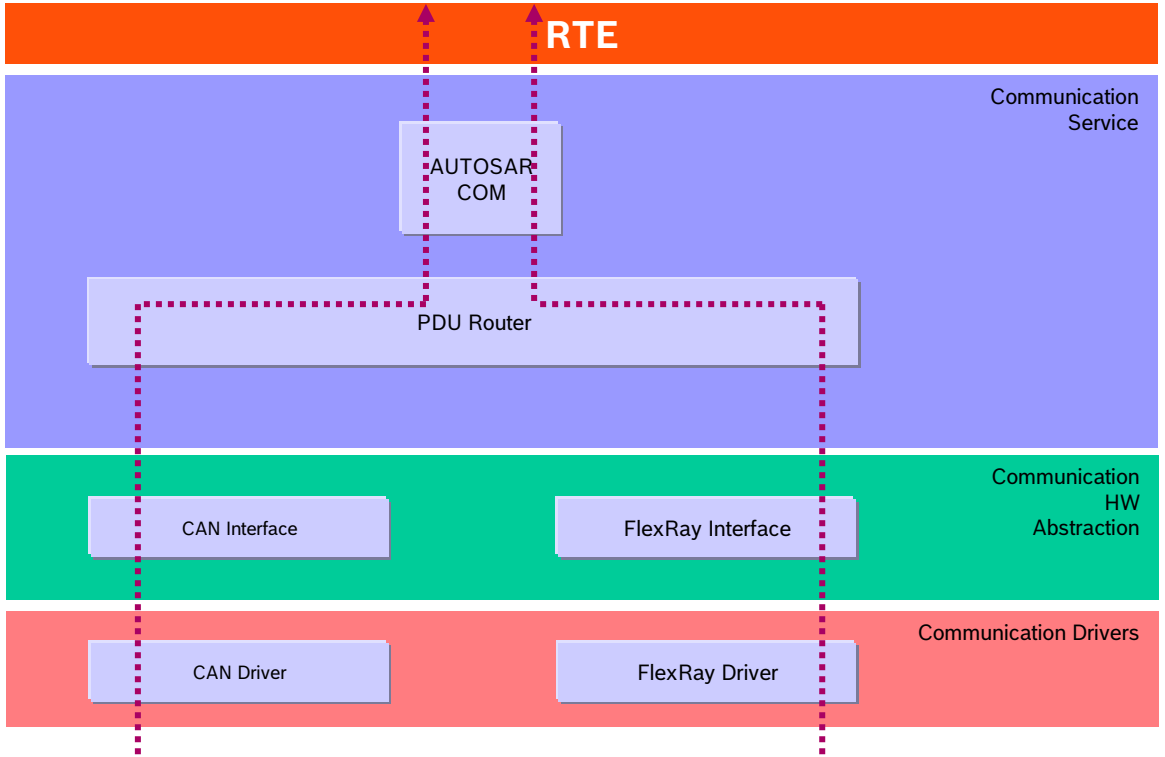
Bosch Global Software Technologies
alt_future

AUTOSAR Basic Software

Communication Stack – Building Blocks (cont.)

REF

Reception



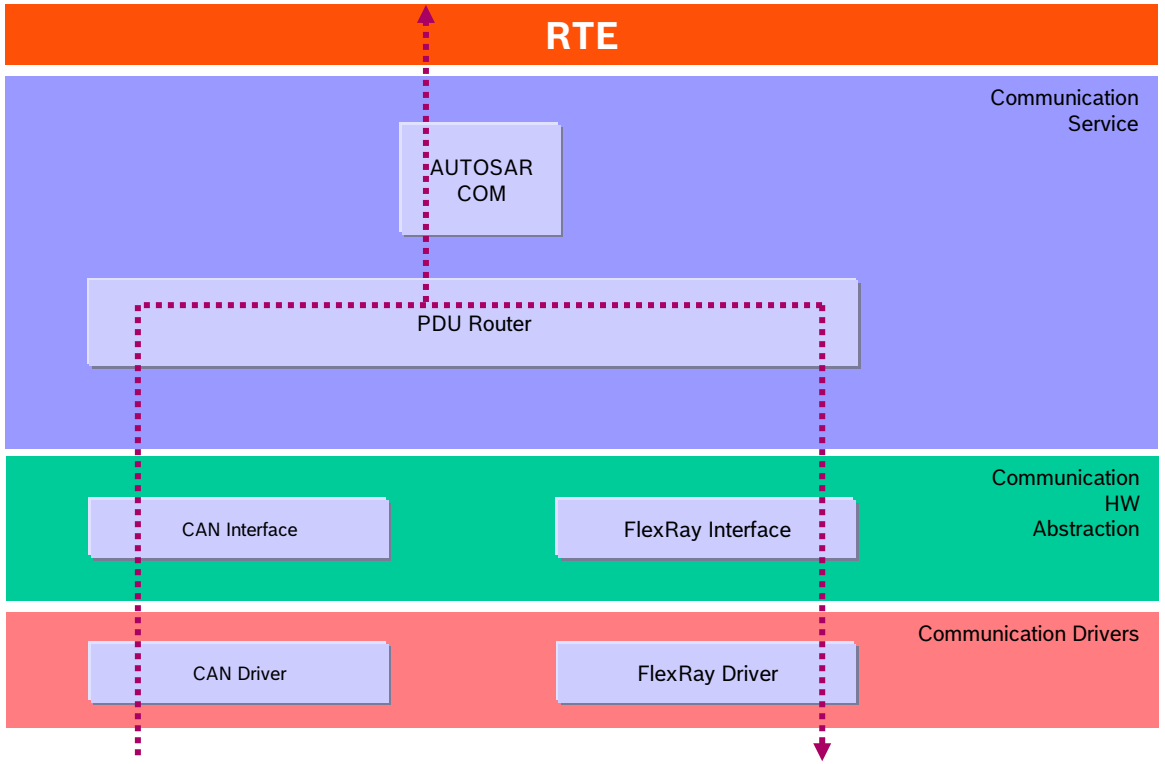
Bosch Global Software Technologies
alt_future

AUTOSAR Basic Software

Communication Stack – Building Blocks (cont.)

REF

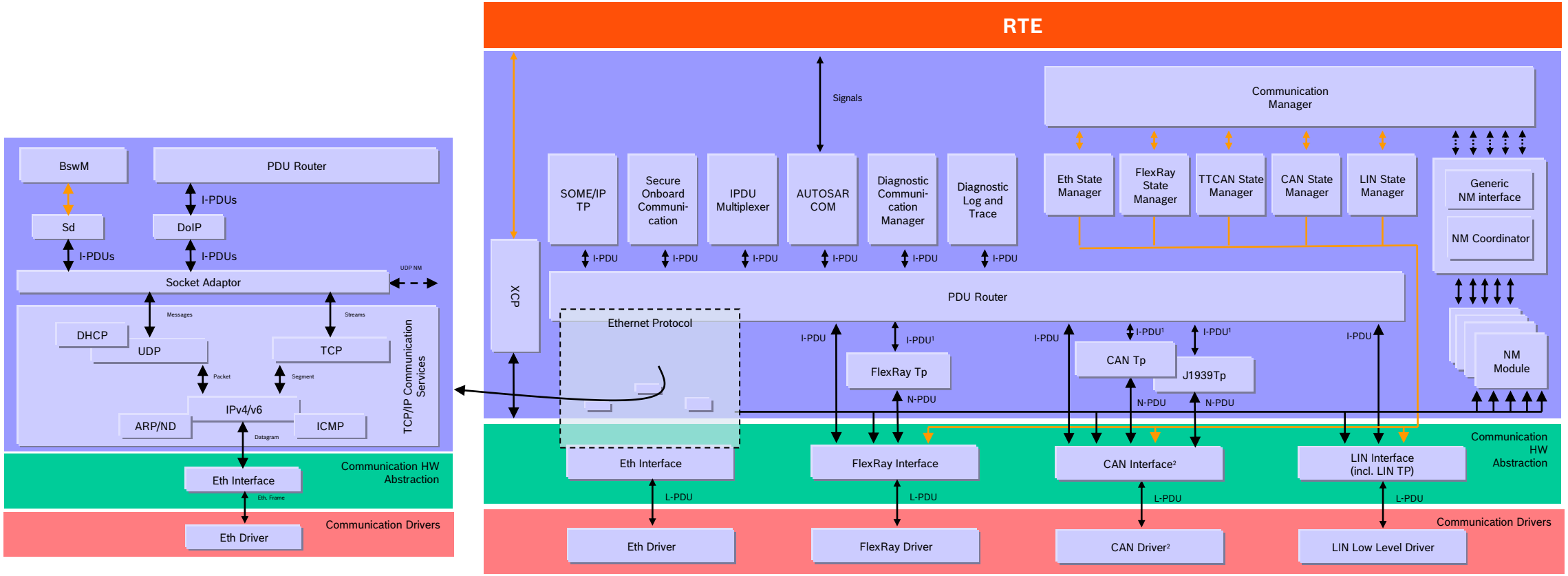
Reception + Gateway



Bosch Global Software Technologies
alt_future

AUTOSAR Basic Software
Communication Stack – Runtime (cont.)

REF



I-PDU: Interaction Layer PDU
N-PDU: Network Layer PDU
L-PDU: Data Link Layer PDU

Note: This image is not complete with respect to all internal communication paths.

[1] The Interface between PduR and Tp differs significantly compared to the interface between PduR and the Ifs.
In case of TP involvement a handshake mechanism is implemented allowing the transmission of I-Pdus > Frame size.
[2] CanIf with TTCAN serves both CanDrv with or without TTCAN. CanIf without TTCAN cannot serve CanDrv with TTCAN.

AUTOSAR Basic Software
Communication Stack – OSEK COM Layer Model

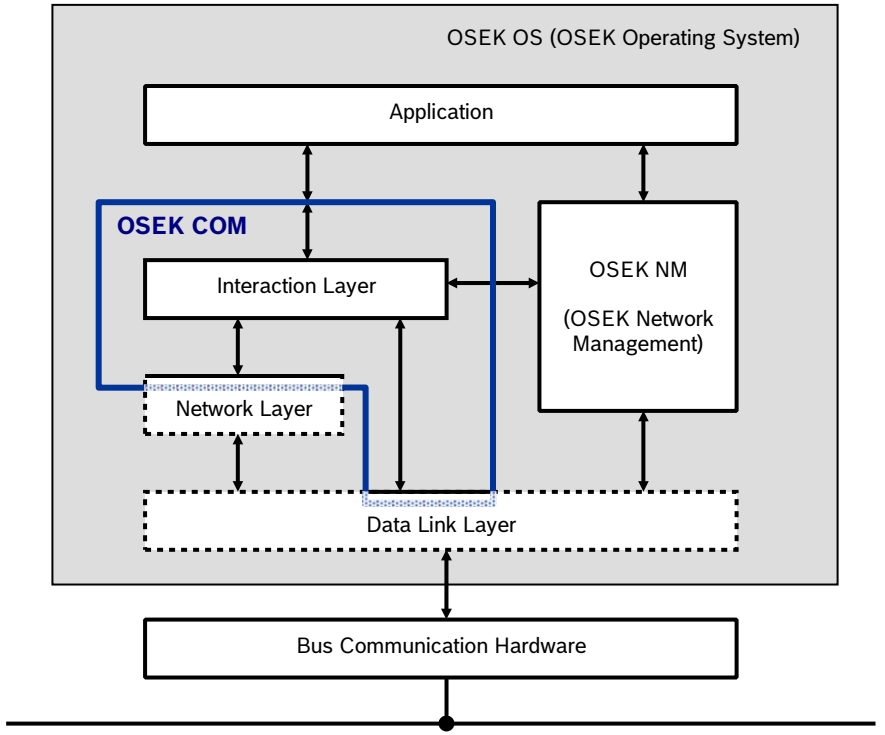
REF

It's the foundation of AUTOSAR ComStack!

Interaction Layer (IL)
- Provides the OSEK COM API which contains services for the transfer (send and receive operations) of messages.
- For external communication it uses services provided by the lower layers, whereas internal communication is handled entirely by the IL.

Network Layer
- Handles – depending on the communication protocol used – message segmentation/recombination and acknowledgement.
- Provides flow control mechanisms to enable the interfacing of communication peers featuring different levels of performance and capabilities.
- The Network Layer uses services provided by the Data Link Layer.
- OSEK COM does not specify the Network Layer; it merely defines minimum requirements for the Network Layer to support all features of the IL.

Data Link Layer
- Provides the upper layers with services for the unacknowledged transfer of individual data packets (frames) over a network.
- Provides services for the NM.
- OSEK COM does not specify the Data Link Layer; it merely defines minimum requirements for the Data Link Layer to support all features of the IL.



ISO/OSI Model

Application
Presentation
Session
Transport
Network
Data Link Layer (DLL)
Physical

OSEK: Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (DE) or Open Systems and the Corresponding Interfaces for Automotive Electronics (EN).

Bosch Global Software Technologies
alt_future

AUTOSAR Layered Architecture
Quiz time

- **Which module is responsible for distribution of Pdu for inter- and intra-ECU communication?**

(a) Com.

(b) PduR.

(c) ComM.

(d) Rte.

AUTOSAR Layered Architecture
Quiz time

- **Which module is responsible for distribution of Pdu for inter- and intra-ECU communication?**

(a) Com.

(b) PduR.

(c) ComM.

(d) Rte.

AUTOSAR Layered Architecture
Quiz time

- **Which statement below is correct when talking about inter-ECU data exchange in AUTOSAR?**

(a) Com Transmission can happen in two fashion: top-down or bottom-up.

(b) Com knows the bus-type of the Pdu it transmit.

(c) Com knows the bus-type of the Pdu it receives.

(d) Transmission is top-down. Reception is bottom-up.

AUTOSAR Layered Architecture
Quiz time

- **Which statement below is correct when talking about inter-ECU data exchange in AUTOSAR?**

(a) Com Transmission can happen in two fashion: top-down or bottom-up.

(b) Com knows the bus-type of the Pdu it transmit.

(c) Com knows the bus-type of the Pdu it receives.

(d) Transmission is top-down. Reception is bottom-up.

AUTOSAR Layered Architecture
Quiz time

▶ **What are the drawbacks of AUTOSAR?**

(a) Standardization by aggregation.

(b) It is huge.

(c) Moving target.

(d) All of above.

AUTOSAR Layered Architecture
Quiz time

▶ **What are the drawbacks of AUTOSAR?**

(a) Standardization by aggregation.

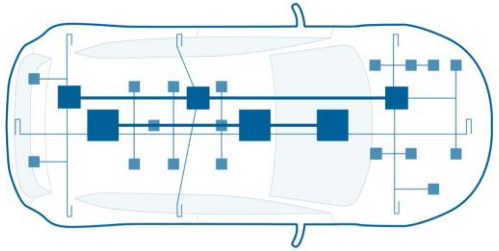
(b) It is huge.

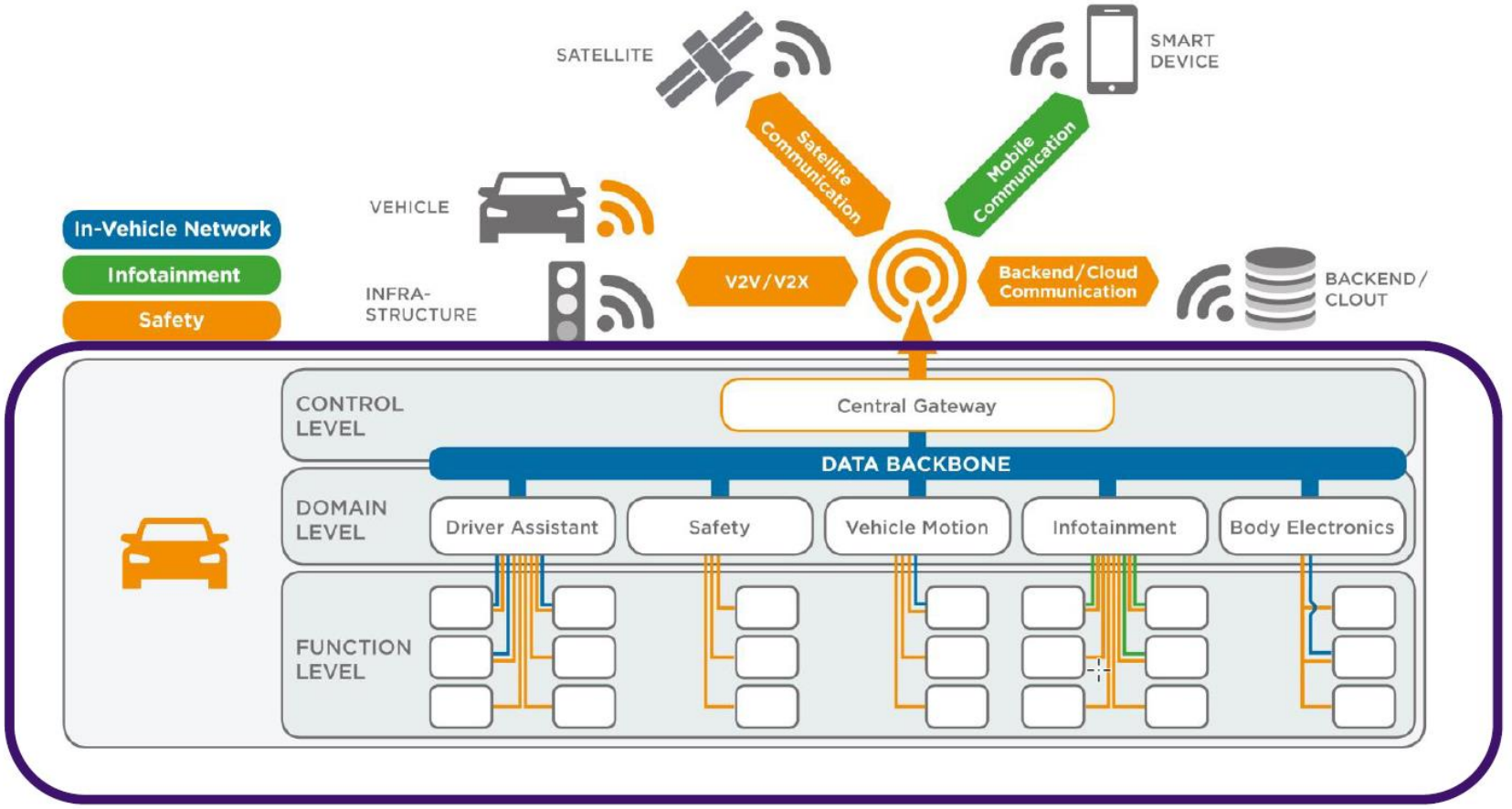
(c) Moving target.

(d) All of above.

CAN
PROTOCOL

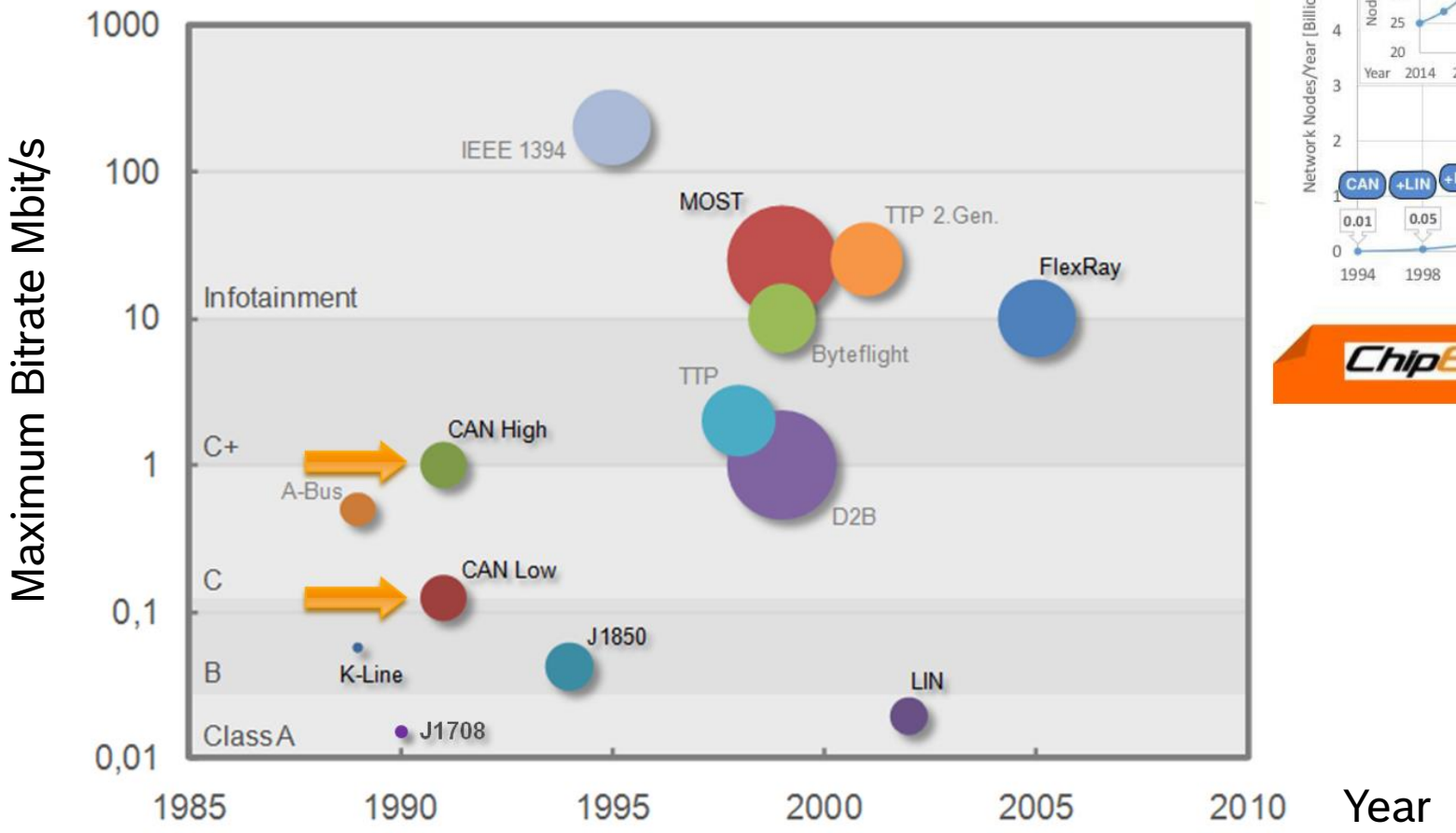
Introduction
Vehicle network

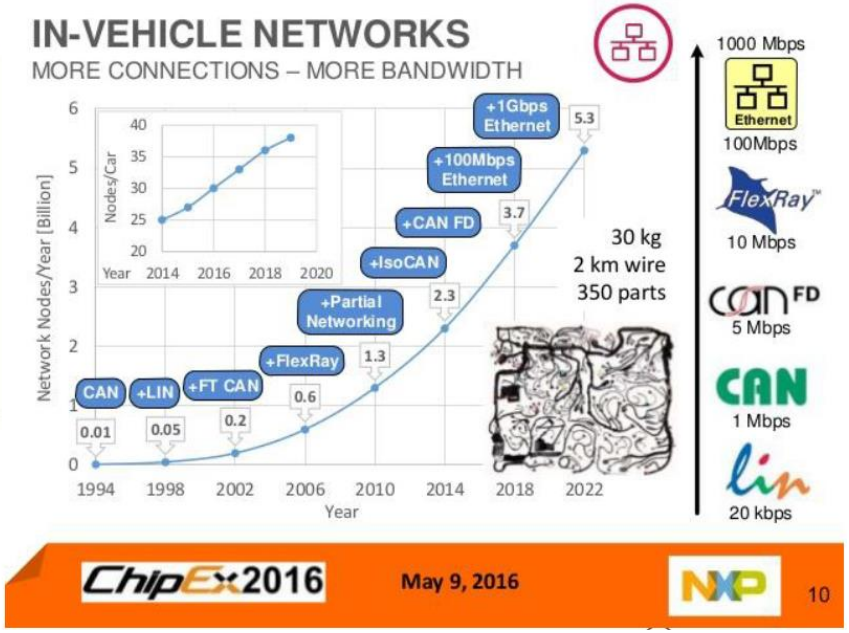




Bosch Global Software Technologies
alt_future

Introduction
History



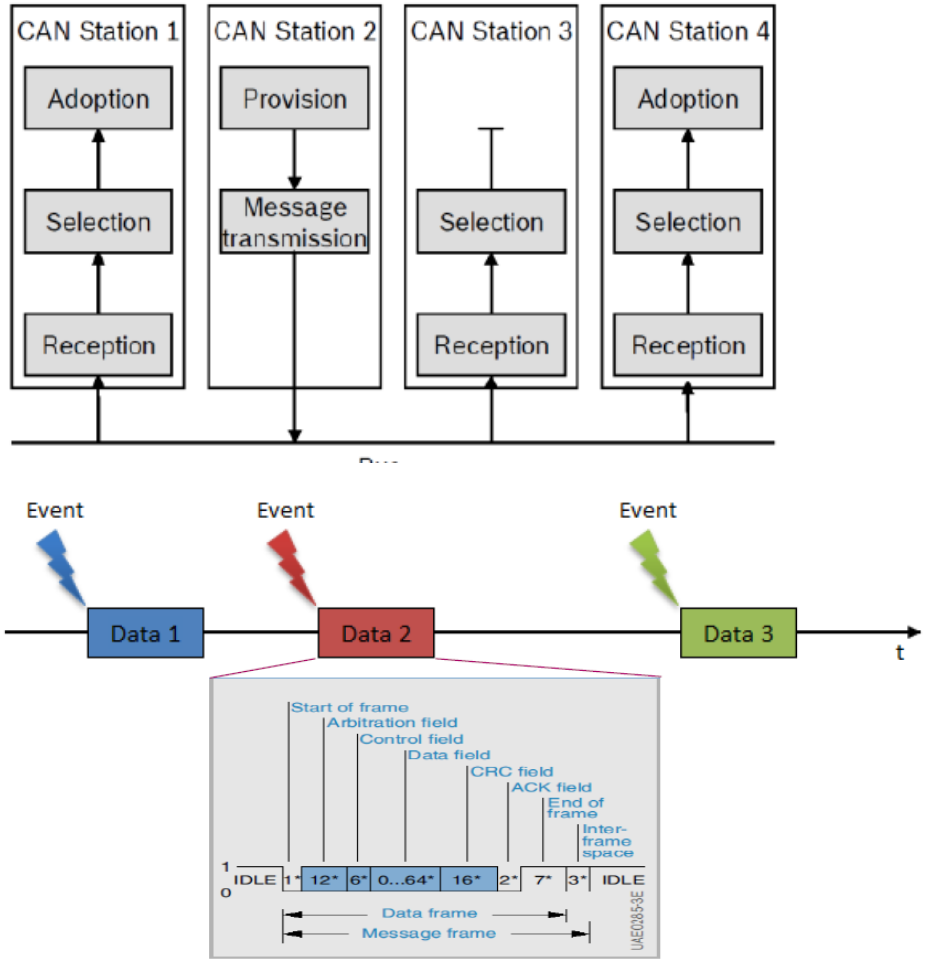


Bosch Global Software Technologies
alt_future

Introduction

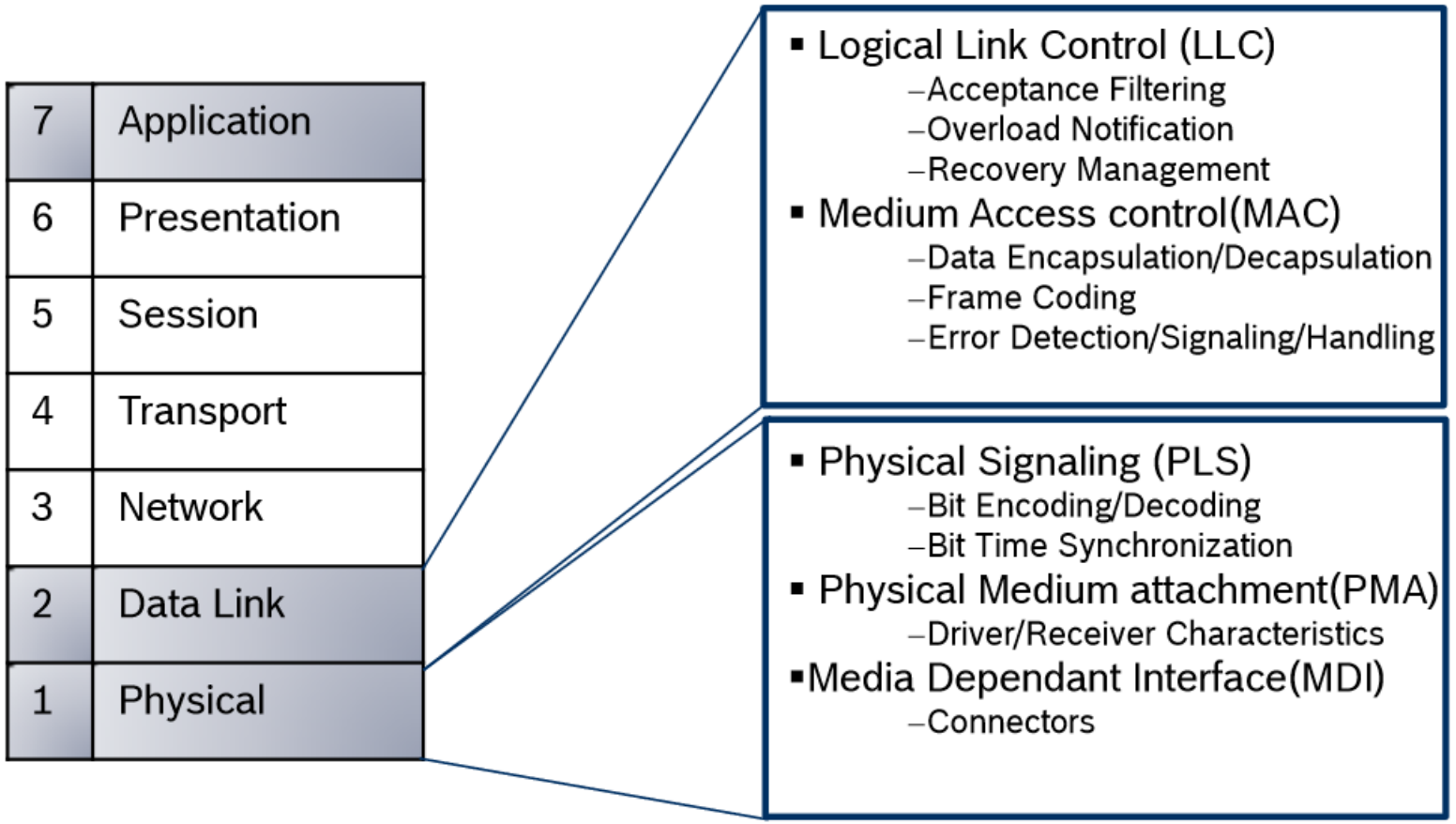
Characteristics of ‘CAN’

- CAN is a multi-master Bus
- Theoretically No limitation on the number of nodes
- Configuration flexibility - No node addressing
- Prioritization of messages through “Identifiers”
- Multicast reception with the time synchronization
- System wide data consistency
- Guarantee of latency times
- Error detection and error signaling
- Automatic retransmission of corrupted messages
- Temporary errors - permanent failures of nodes and autonomous switching off defect nodes



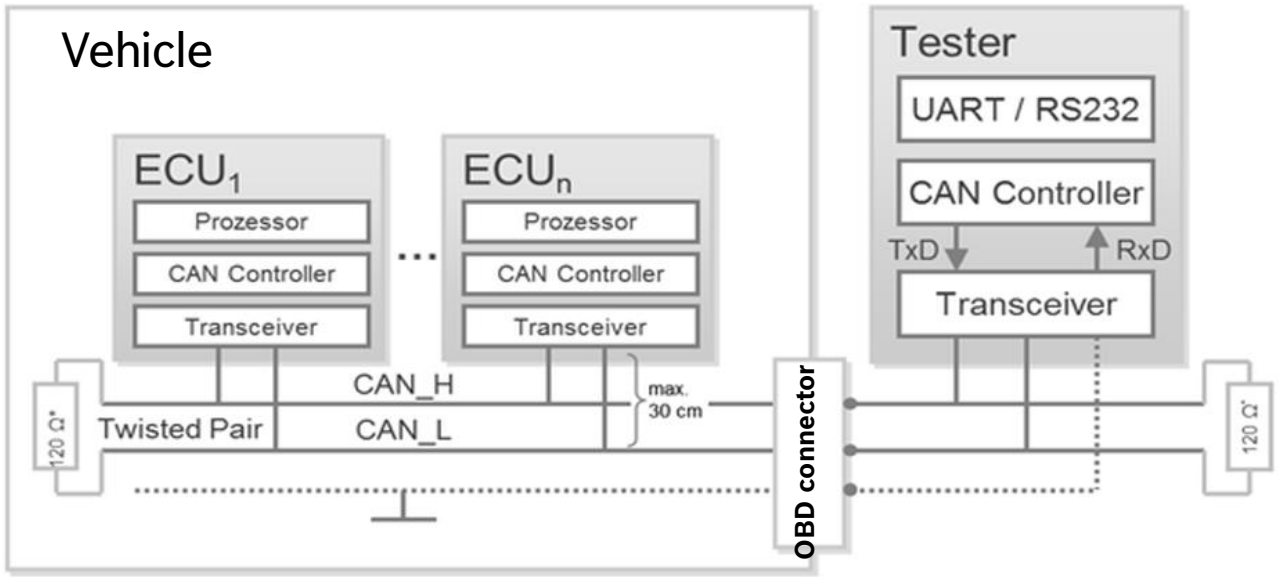
Bosch Global Software Technologies
alt_future

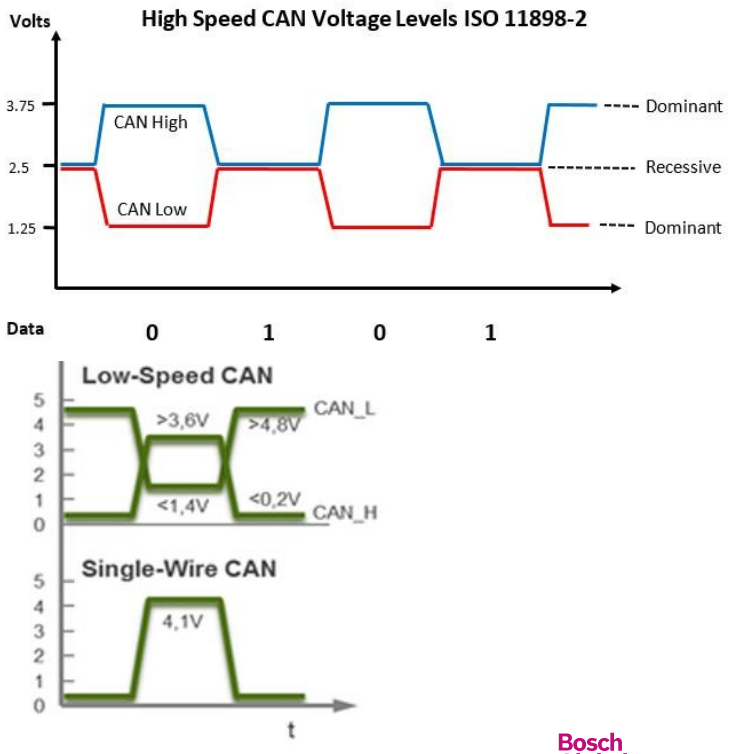
Introduction
CAN in the OSI model



CAN Protocol
Physical Layer

- Bit rate: up to 1Mbit/s
- Bidirectional Dual-wire bus with 40-50m maximum in length
- Multi-Master



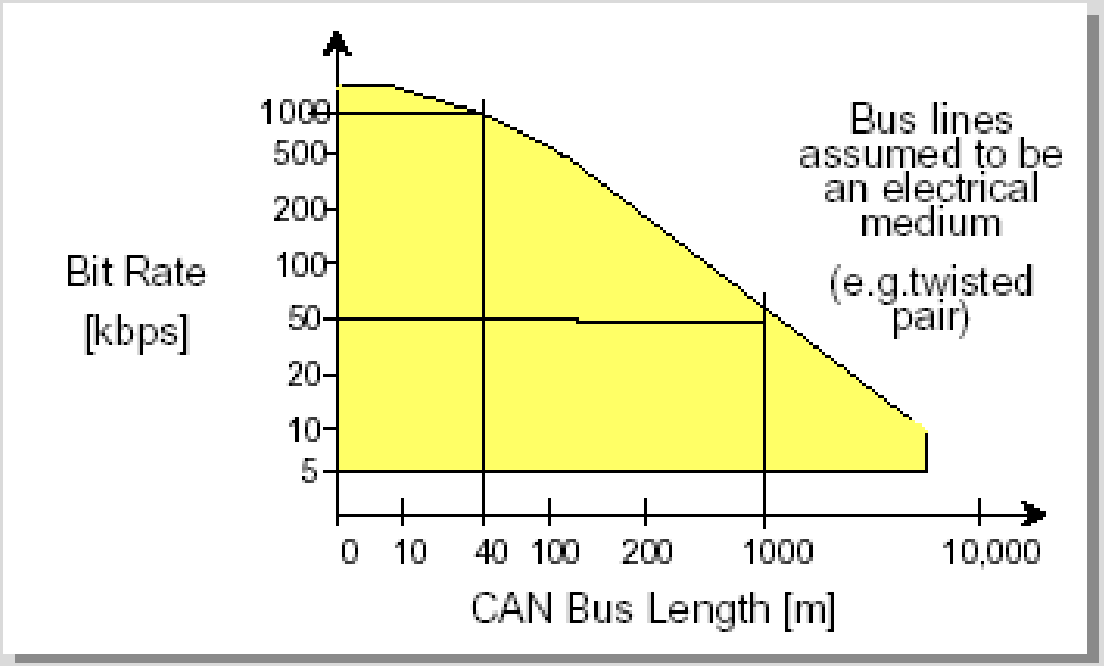


Bosch Global Software Technologies
alt_future

CAN Protocol

Relation between Baud Rate and Bus Length

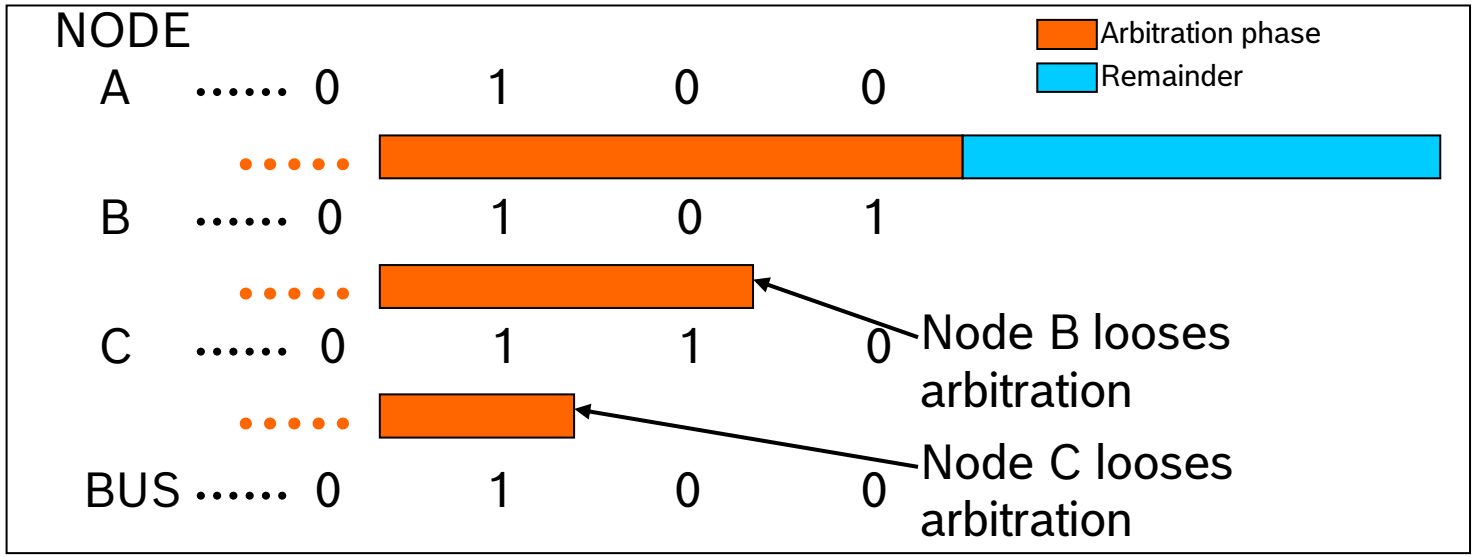
- Up to 1Mbit / sec @40m bus length (130 feet)



CAN Protocol

Bus Access and Arbitration

- Bus access through CSMA with AMP



- Advantages
  - No Collision
  - Transmission of highest priority message within the latency time

CAN Protocol
Message Transfer

Frame Formats

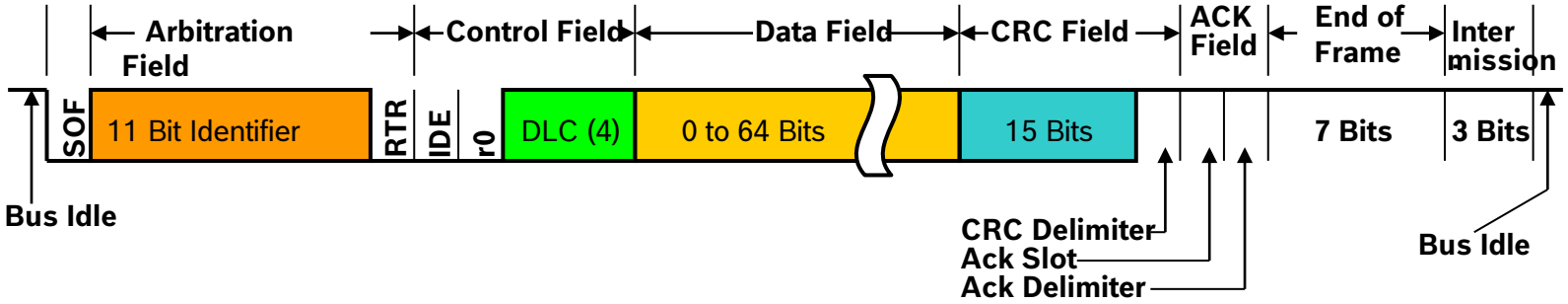
- Standard Frame - 11bit Identifier
- Extended Frame - 29 bit Identifier

Frame Types

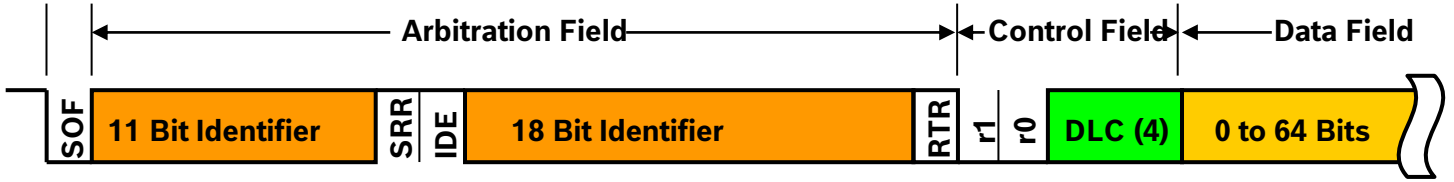
- Data Frame
- Remote Frame (not useful)
- Error Frame
- Overload Frame (not useful)

CAN Protocol
Data Frame

Standard Data Frame Format



Extended Data Frame Format

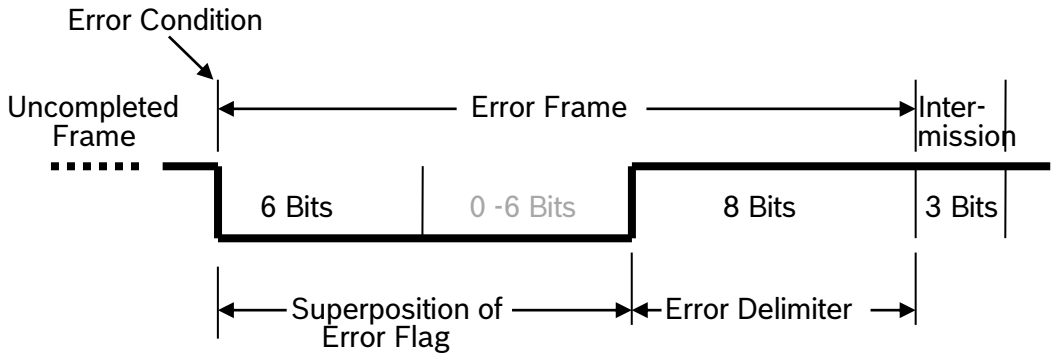


Difference between Standard Frame and Extended Frame

- **Differs only in Arbitration field and Control field**

CAN Protocol
Error Frame

Error Frame Format (Active Error Frame)

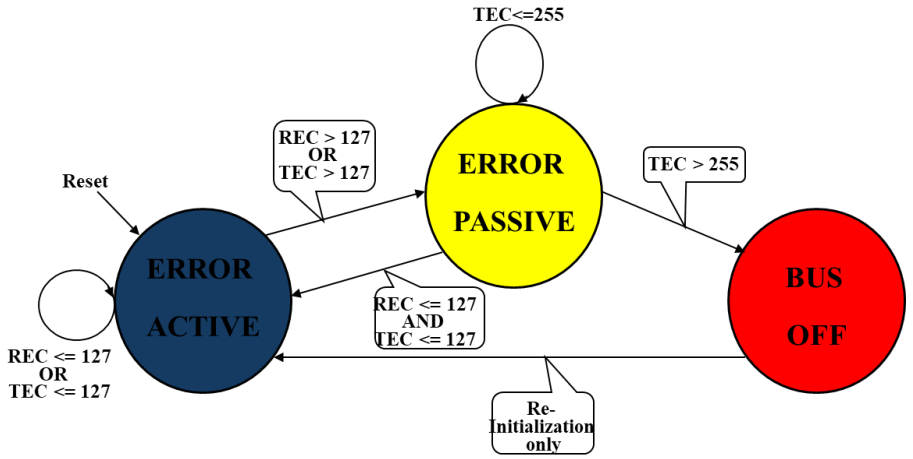


- Error flag can start within the frame that is currently being transmitted

Types of Error flags

- Active Error flag - consists of 6 consecutive ‘dominant’ bit
- Passive Error flag - consists of 6 consecutive ‘recessive’ bit

CAN Protocol
Error Handling



The mode of the controller is controlled by two error counters - the transmit error counter (tx_count) and the receive error counter (rx_count).

The following rules apply:

- The CAN controller is in **error active mode** if:

tx_count <= 127 AND rx_count <= 127.

- Passive mode is used if :

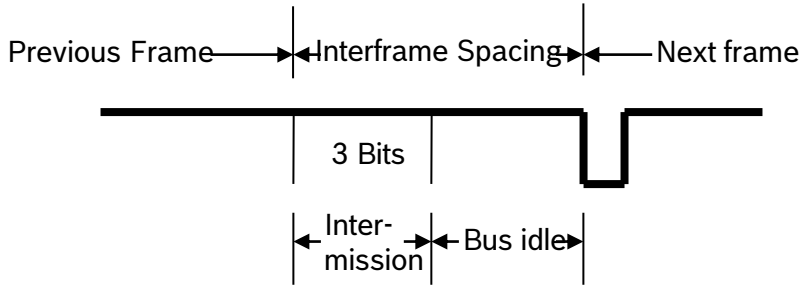
tx_count > 127 or rx_count>127 AND tx_count <= 255.

- Bus off is entered if:

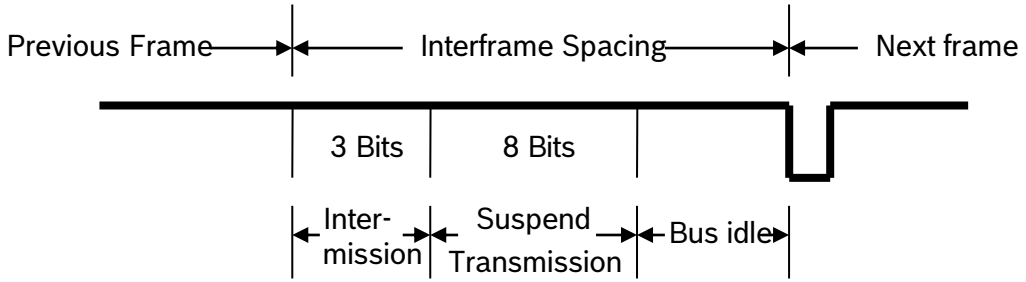
tx_count > 255.

CAN Protocol
Interframe Spacing

After the transmission of a frame by an Error Active node



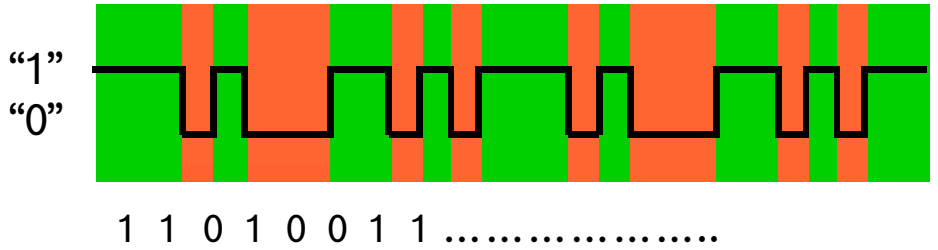
After the transmission of a frame by an Error Passive node



CAN Protocol
Message Coding

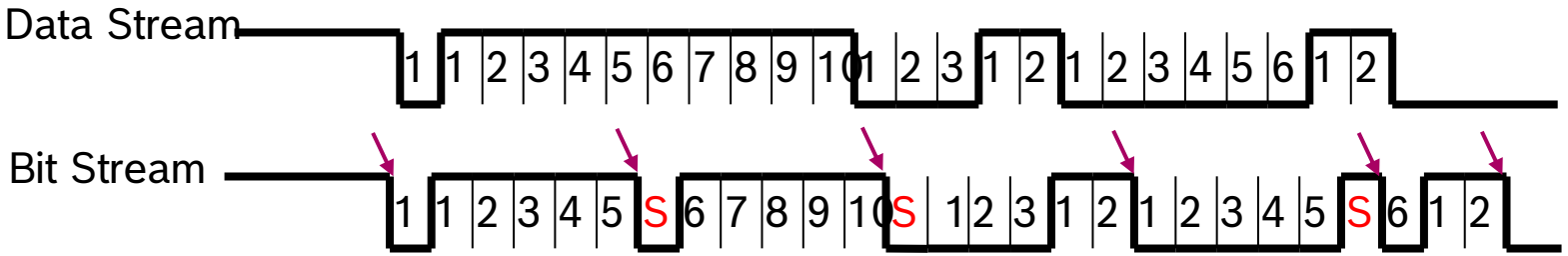
Non-Return-to-Zero coding

- Keeps the frequency of the signal on the bus to minimum.

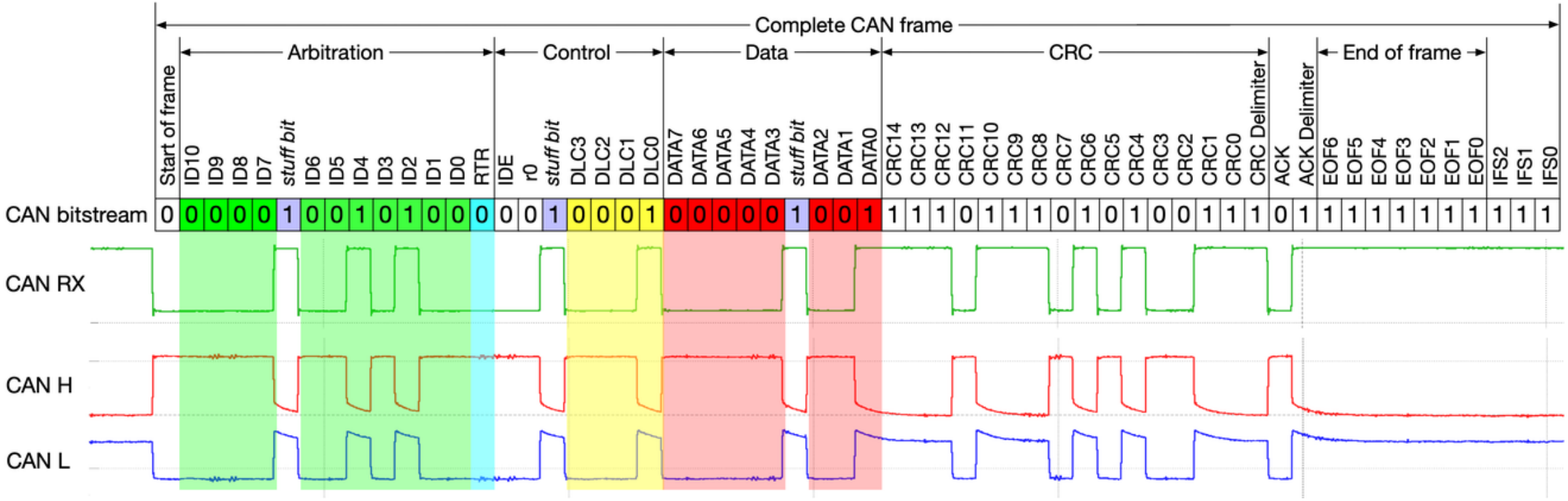


Bit-Stuffing

- Ensures sufficient Recessive and Dominant edges for Re-Synchronization.



A complete CAN bus frame



Source: Wikipedia

CAN Protocol
Types of Error Detected in CAN Bus

CRC Error:

- Every node receive the message, Calculate CRC and compare it with Received CRC.

Acknowledge Error:

- Transmitting node send a ACK slot bit as a recessive bit and check for dominant bit to verify reception.

Form Error:

- Generated when any of following bit is detected as a dominant bit where One should not be.

e.g. CRC delimiter, ACK delimiter, End of Frame, Inter Frame Space.

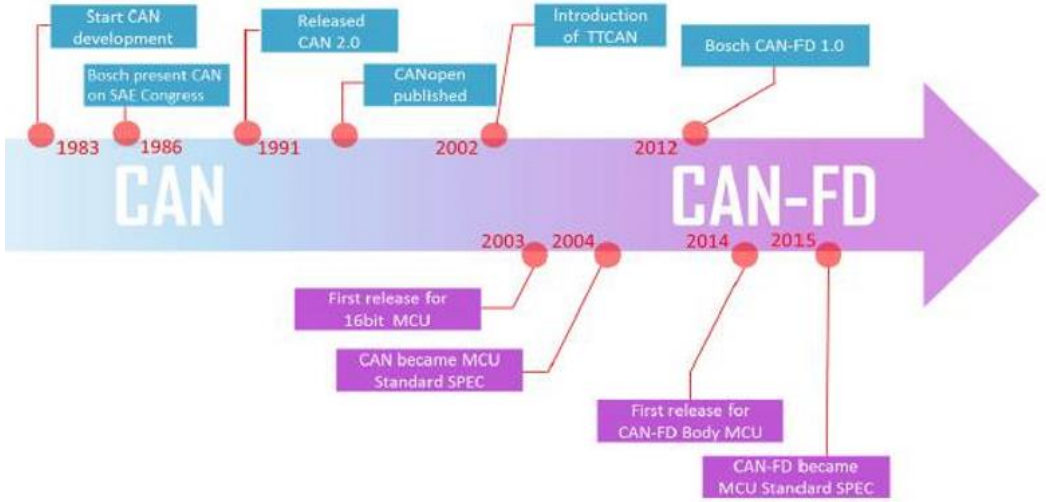
Bit Error:

- Node detect the signal that is opposite of what it send on Bus.

Stuff Error:

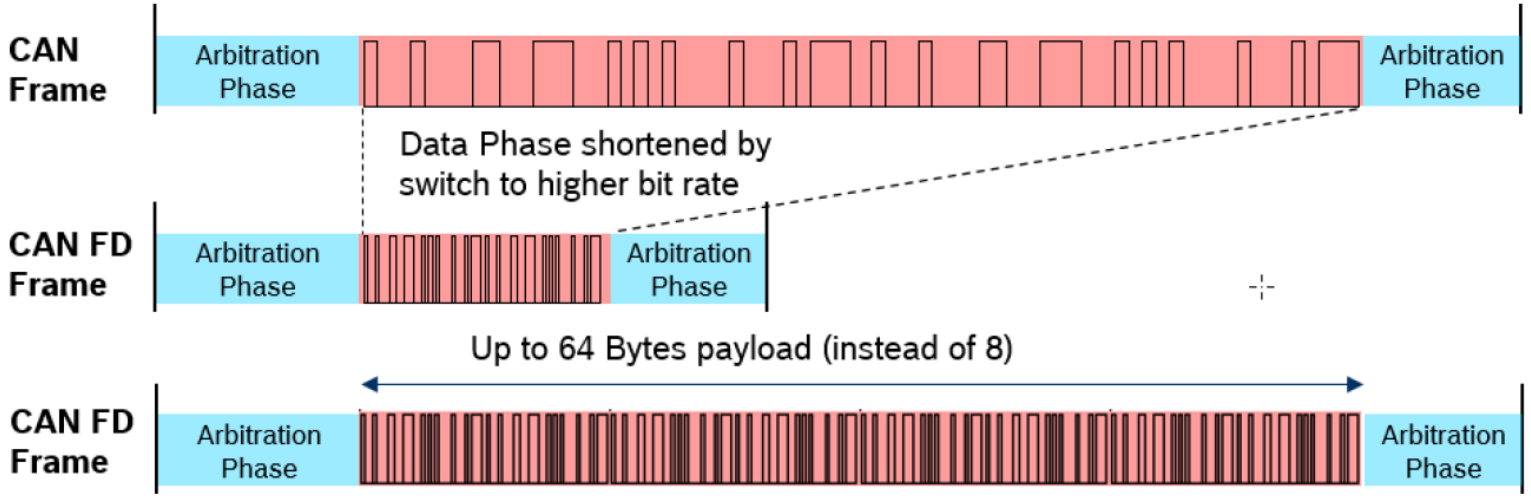
- Bit stuffing rule is violated when 6-consecutive bits with the same polarity are detected.

Introduction about CAN FD



Main improvement:

- Increase bit rate (2,4 … up to 8 Mbit/s)
- Increase payload up to 64 bytes



Bosch Global Software Technologies
alt_future

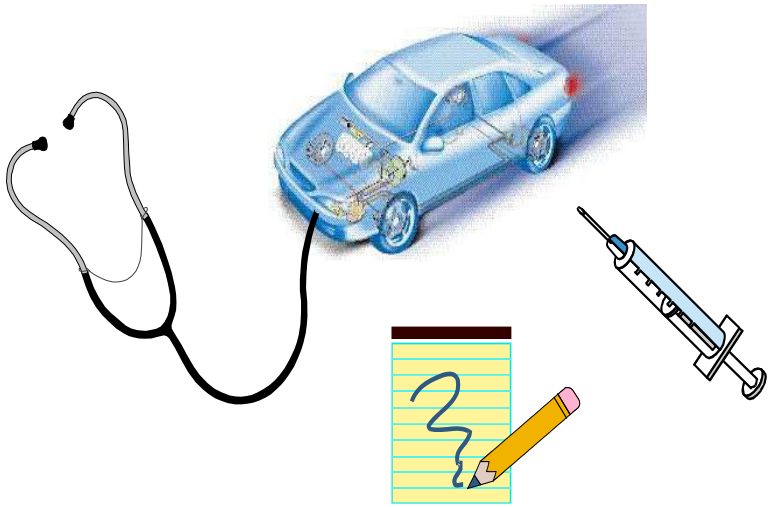
Reference

- CAN Specification 2.0 – Bosch
- ISO 11898-2 – High speed CAN
- ISO 11898-2 2015 – CAN FD
- Bit timing calculation: http://www.bittiming.can-wiki.info/
- Return to zero: https://en.wikipedia.org/wiki/Return-to-zero
- None return to zero : https://en.wikipedia.org/wiki/Non-return-to-zero
- CSMA/CD : https://en.wikipedia.org/wiki/Carrier-sense_multiple_access_with_collision_detection

Q & A

DIAGNOSTIC COMMUNICATION OVERVIEW

Diagnosis In Automotive
Definitions



"In automotive engineering, **Diagnosis** is typically used to determine the causes of symptoms and solutions to issues."

- symptom(s) – what the user/operator/repairer of the system (vehicle or whatever) notices;
- fault(s) – the error(s) in the system that result in the symptom(s);
- root cause(s) – the cause(s) of the fault.

Source: Advanced Automotive Fault Diagnosis- Automotive Technology: Vehicle Maintenance and Repair

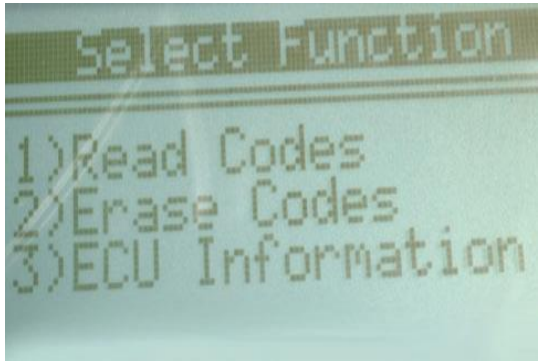
TOM DENTON

Diagnosis In Automotive
Methods

To do Diagnostic, Technician have to know how to use Diagnostic Tools and Equipment.

Tool and Equipment could be classified into:

- Basic Equipment: such as Multi-meter
- Tracing Tool: like Oscilloscope
- Scanner/Fault Code Readers and Analyzers



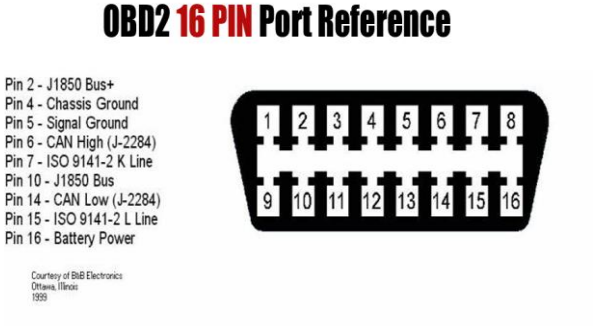
Bosch Global Software Technologies
alt_future

Diagnosis In Automotive
Methods (cont.)

- The Equipment shall help technician indicate where is fault occurs in systems.
- In the other word, In Vehicle, Systems should have ability to provide information in case request.
- This is the motivation of On-board diagnostics (OBD).

- On-board diagnostics (OBD) is a generic term referring to a vehicle's self-diagnostic and reporting system. OBD systems give the vehicle owner or a technician access to information for various vehicle systems.
- OBD system illuminates a warning lamp known as the malfunction indicator lamp (MIL) or malfunction indicator (MI) on the instrument cluster.

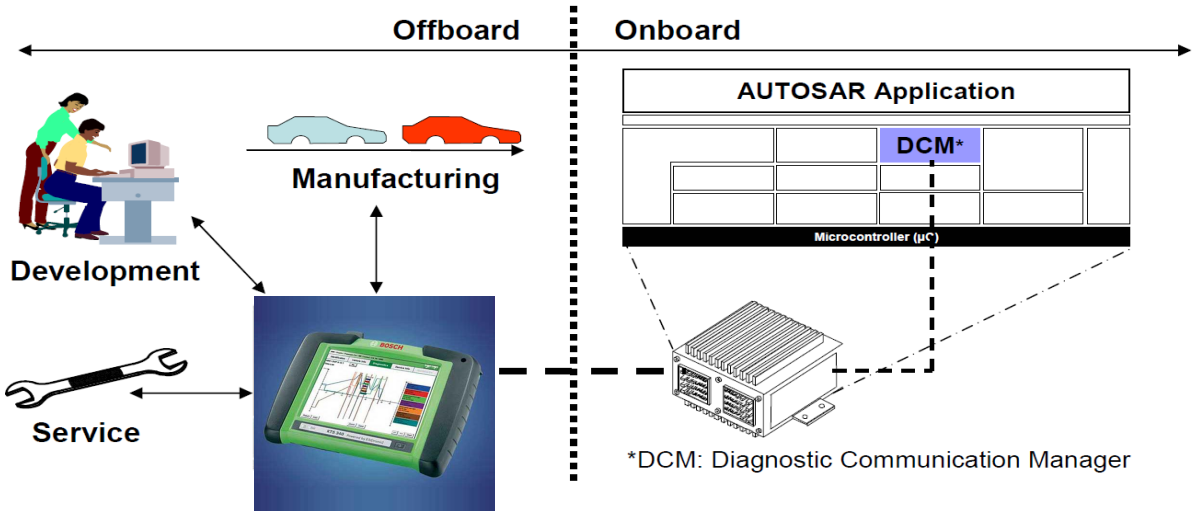
Diagnosis In Automotive
Methods (cont.)



- When the fault occurs, the system stores a diagnostic trouble code (DTC), also store important information of the vehicle when the fault was set.
- A service technician is able to connect a diagnostic scan tool or a code reader that will communicate with the system and retrieve this information.
- As vehicles and their systems become more complex, the functionality of OBD is being extended to cover vehicle systems and components that do not have anything to do with vehicle emissions control: Vehicle body, chassis and accessories
- OBD systems use a standardized communications port to provide data
- The Communication between Diagnostic Equipment and ECUs through Vehicle Special Interface for Diagnosis purpose is called **Diagnostic Communication.**

Diagnostic Communication
Definitions

- The Communication between Diagnostic Equipment and ECUs through Vehicle Special Interface for Diagnosis purpose is called **Diagnostic Communication**



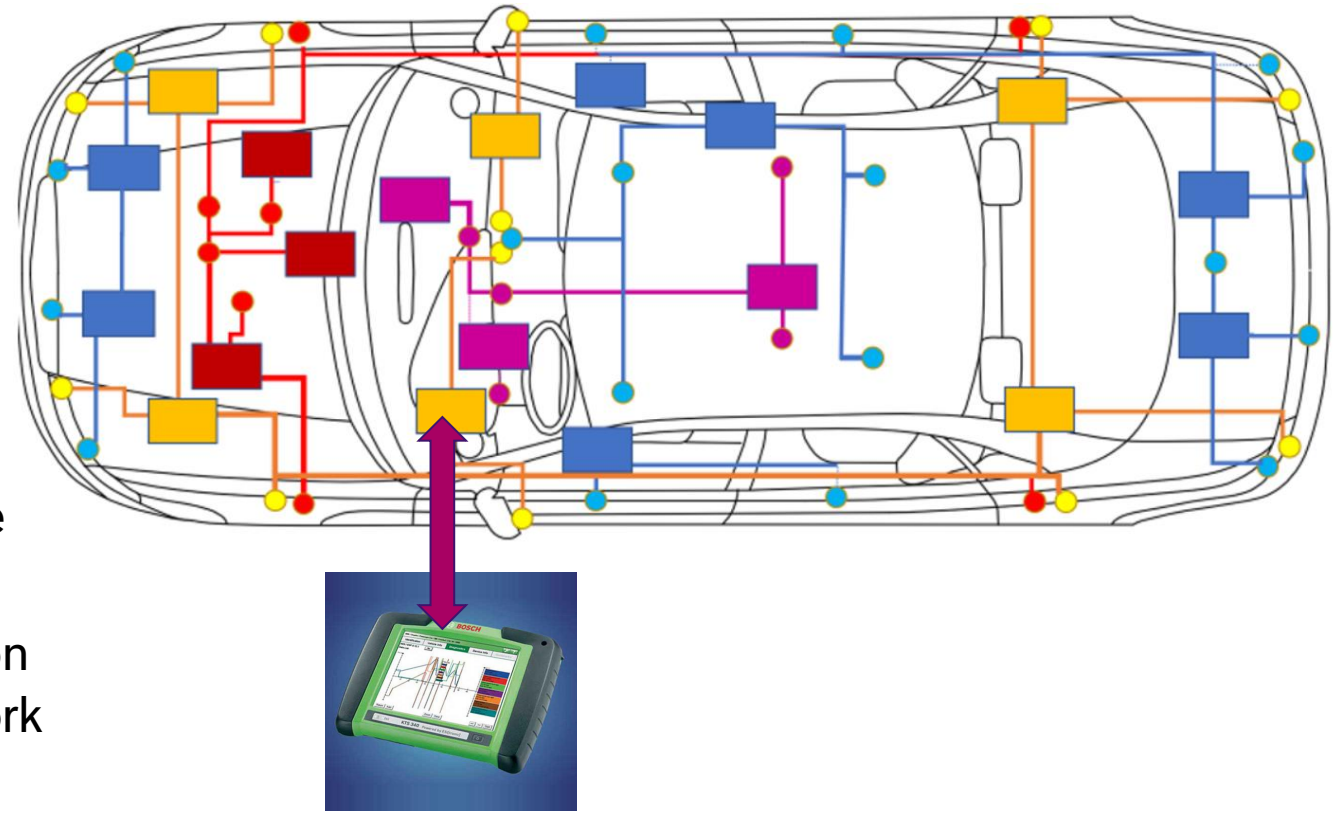
- **Onboard:** ECUs that are part of vehicle network with SW that can perform Diagnostic Communication
- **Offboard:** Diagnostic Equipment also with SW that can perform Diagnostic Communication
- **Users:**
  - Developers, technicians at manufacture lines, technicians at repair/service workshop…
  - In most cases, drivers – or end users, are not target users of Diagnostic Communication

Diagnostic Communication
Use Cases

- Diagnosis is used to detect the fault in the system.
  - → The only Use Case that is actually Diagnosis-related
- Use to read the parameters like SW version info, SW parameters,....
- Used for calibration/configuration of SW
- Use to run EOL(End of Line) routines.
- Used for reprogramming.
  - → Motivation for the rest?

Diagnostic Communication
Challenges

- Vehicle network includes dozens of ECUs
  - How to make connection with all ECUs?
  - How many Equipment do we need?



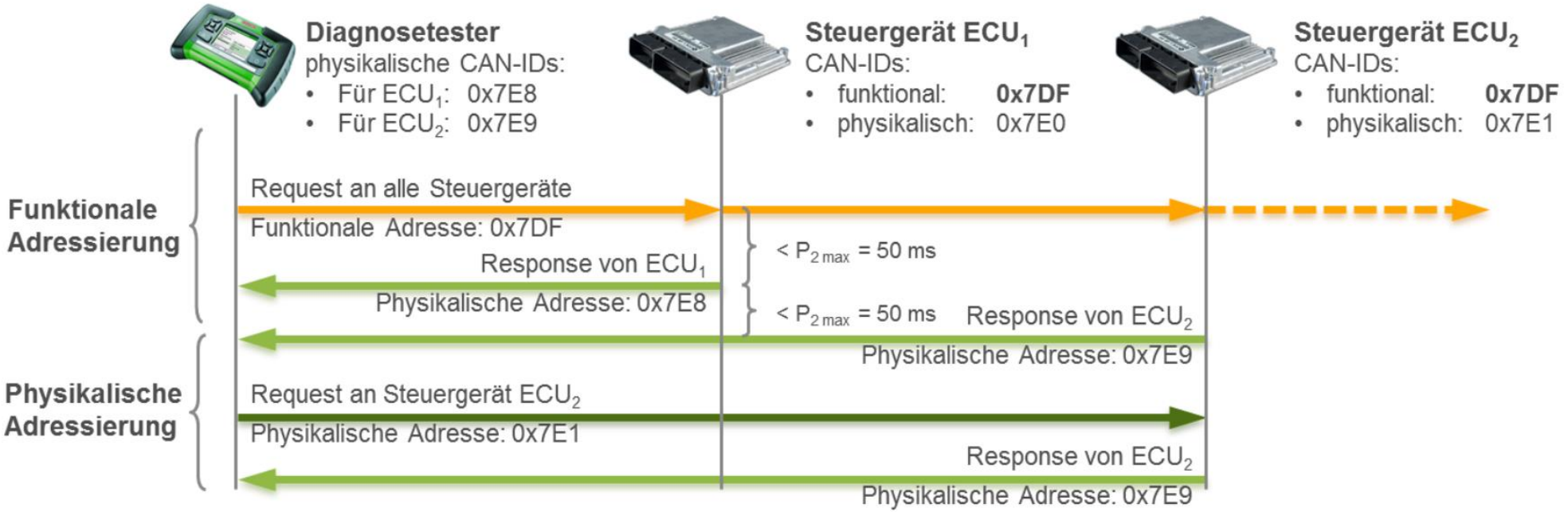
- Physical connection is only between the Equipment and one/few ECU(s)
- Messages for Diagnostic Communication will then be forwarded via vehicle network

- Communication between Equipment and ECUs need to be standardized

=> Diagnostic Communication Protocols

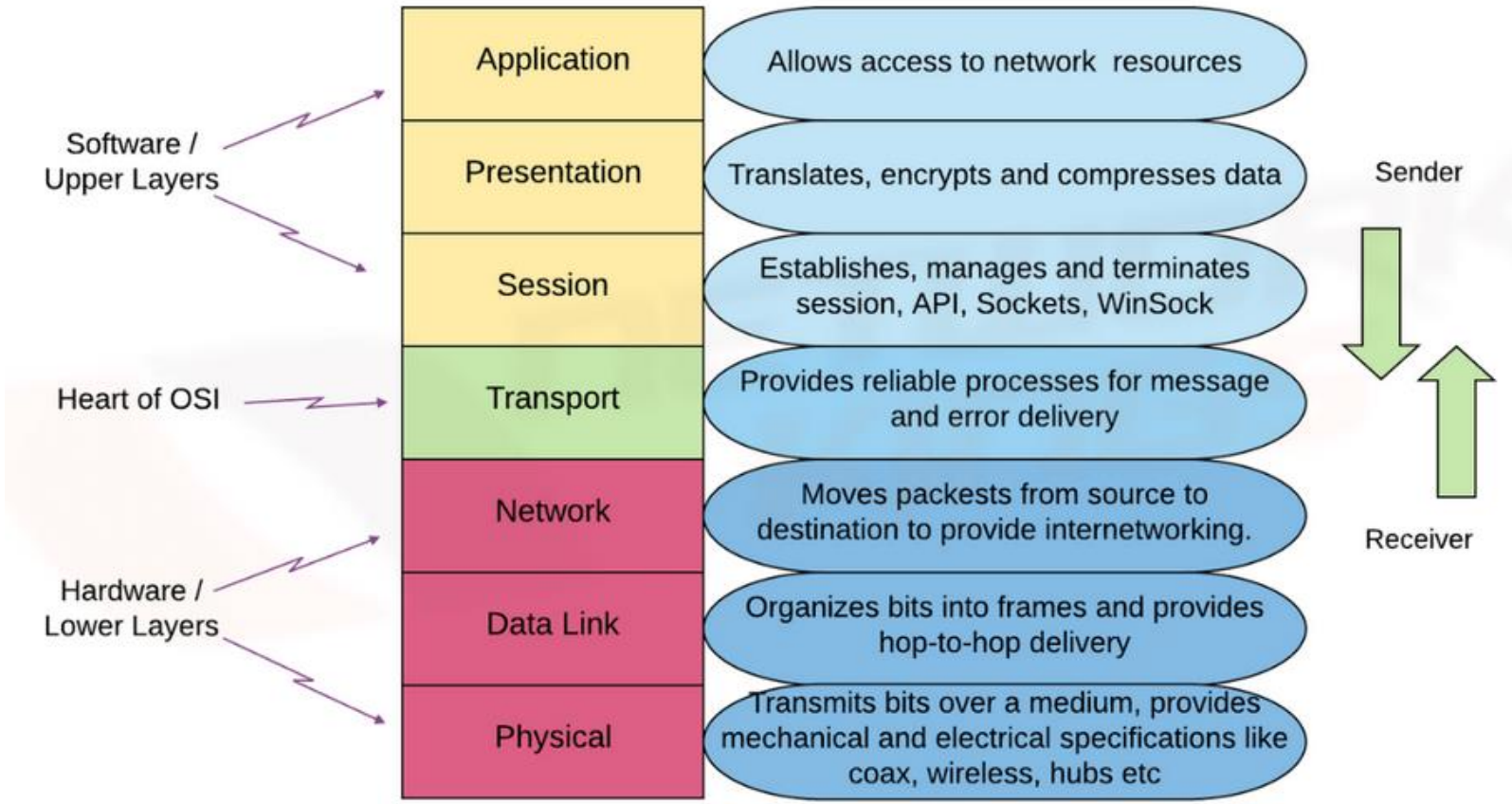
Diagnostic Communication
Addressing Mode

- Functional Addressing: from Tester to many/all ECUs in the network
- Physical Addressing: from Tester to a single identified ECU in the network

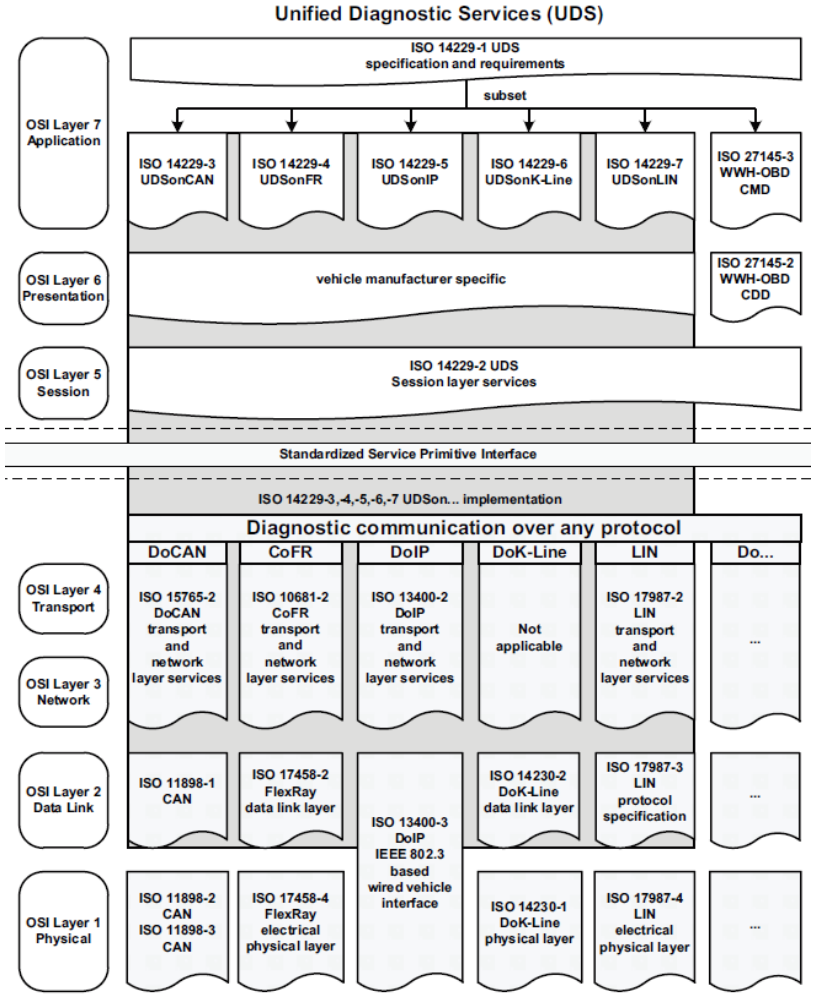


Diagnostic Communication Protocols
OSI 7 Layers Model

- Diagnostic Communication Protocols
- Diagnostic over certain network (for e.g. CAN)
- Vehicle network



Diagnostic Communication Protocols
Unified Diagnostic Services (UDS)



Bosch
Global
Software
Technologies
alt_future

Diagnostic Communication Protocols
Emissions-related diagnostics (emissions-related OBD)

<table>
<tr><th>Applicability</th><th>OSI 7 layers</th><th colspan="7">Emissions-related OBD communication requirements</th><th colspan="3">Emissions-related WWH-OBD communication requirements</th></tr>
<tr><td rowspan="9">Seven layer according to ISO/IEC 7498-1 and ISO/IEC 10731</td><td>Application (layer 7)</td><td colspan="7">ISO 15031-5</td><td colspan="3">ISO 27145-3</td></tr>
<tr><td rowspan="2">Presentation (layer 6)</td><td colspan="7">ISO 15031-2, -5, -6
SAE J1930-DA/SAE J1979-DA</td><td colspan="3">ISO 27145-2
SAE J1930-DA/SAE J1979-DA</td></tr>
<tr><td colspan="7">SAE J2012-DA (OBD)</td><td colspan="3">SAE J2012-DA (WWH-OBD)</td></tr>
<tr><td>Session (layer 5)</td><td colspan="2">Not applicable</td><td colspan="6">ISO 14229-2</td></tr>
<tr><td>Transport (layer 4)</td><td colspan="2" rowspan="2">ISO 15031-5</td><td rowspan="2">ISO 14230-4</td><td rowspan="2">ISO 15765-2</td><td rowspan="3">ISO 15765-4</td><td>ISO 15765-2</td><td rowspan="3">ISO 27145-4</td><td rowspan="2">ISO 13400-2</td></tr>
<tr><td>Network (layer 3)</td></tr>
<tr><td>Data link (layer 2)</td><td rowspan="2">SAE J1850</td><td rowspan="2">ISO 9141-2</td><td>ISO 14230-2</td><td rowspan="2">ISO 11898-1, ISO 11898-2</td><td rowspan="2">ISO 11898-1, ISO 11898-2</td><td rowspan="2">ISO 13400-3</td></tr>
<tr><td>Physical (layer 1)</td><td>ISO 14230-1</td></tr>
</table>

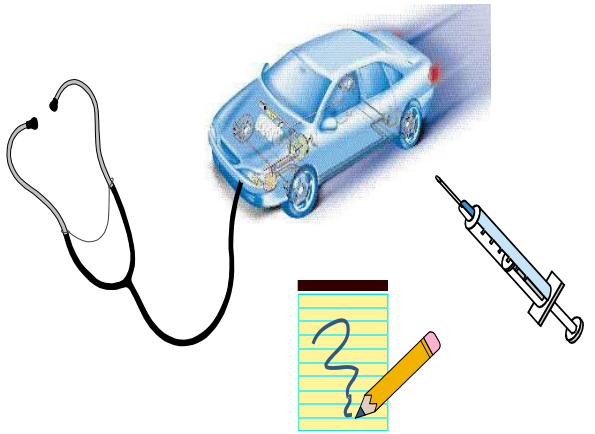
UNIFIED DIAGNOSTIC SERVICES

Agenda

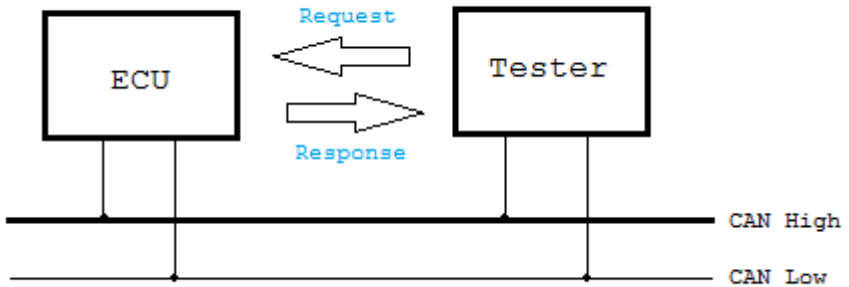
1. Overview
    1. UDS
    2. Diagnostic Communication Protocol
    3. Related ISO Standards
2. UDS In General
    1. Application Layer
    2. Session Layer
3. UDS In CAN Implementation

Overview
UDS

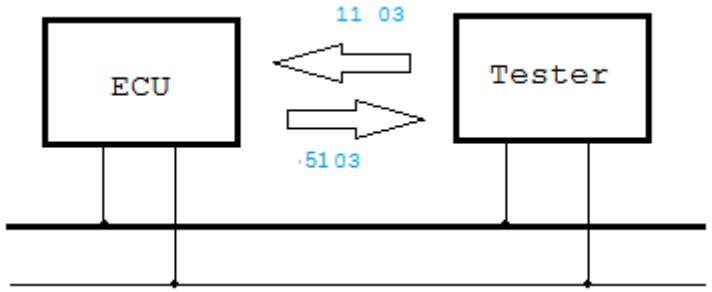
- (U)nified (D)iagnostic (S)ervices is a diagnostic communication protocol
- UDS is a non-OBD protocol
- Standardized across manufacturers and standards
- Use cases:
  - Detect faults in the system
  - Read vehicle parameters
  - Configuration/Calibration ECUs
  - End Of Line Routine
  - Reprogramming



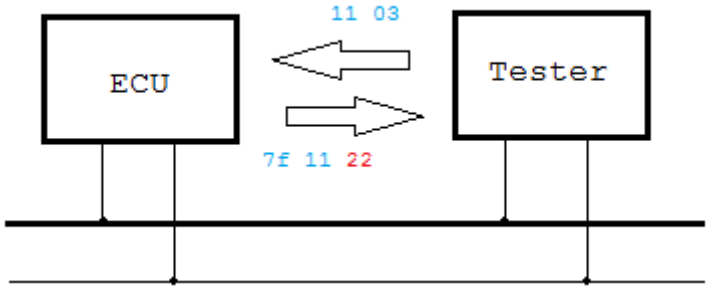
Overview
Protocol



Protocol: server ⇔ client



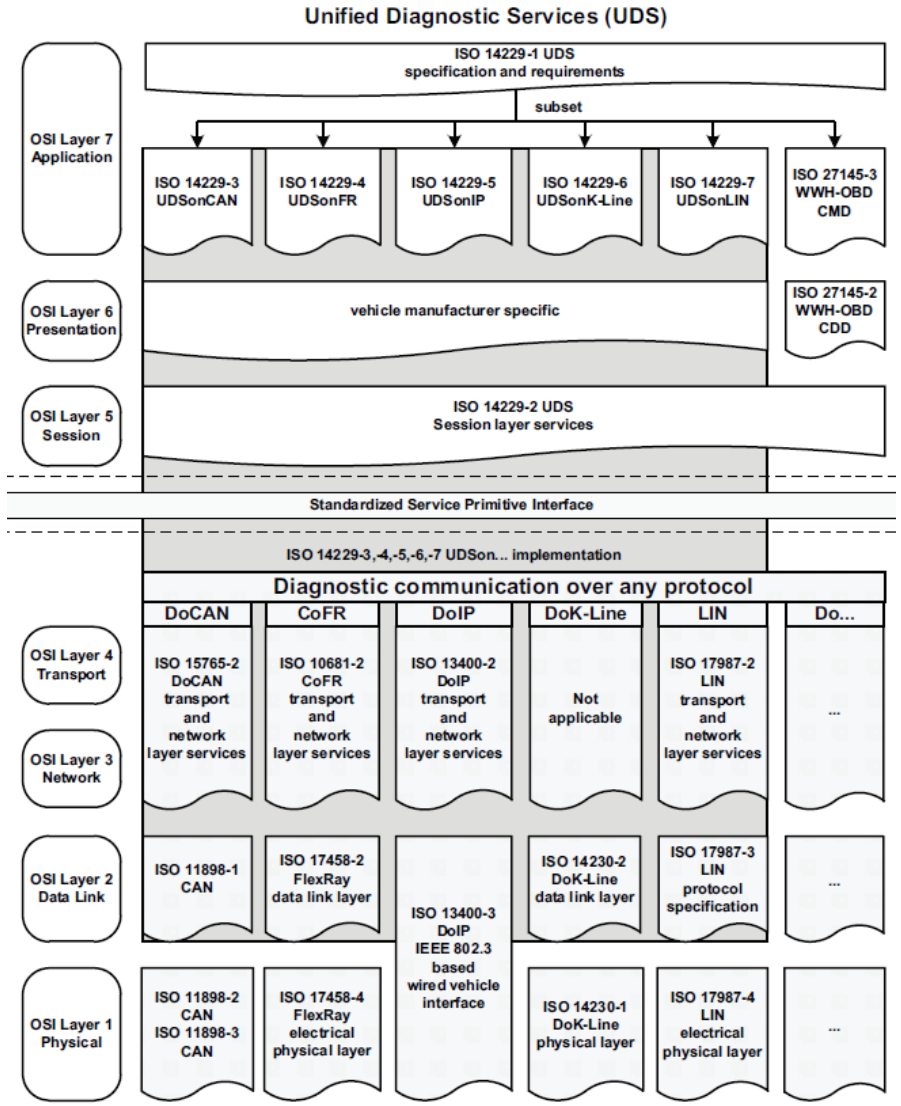
Positive response



Negative response

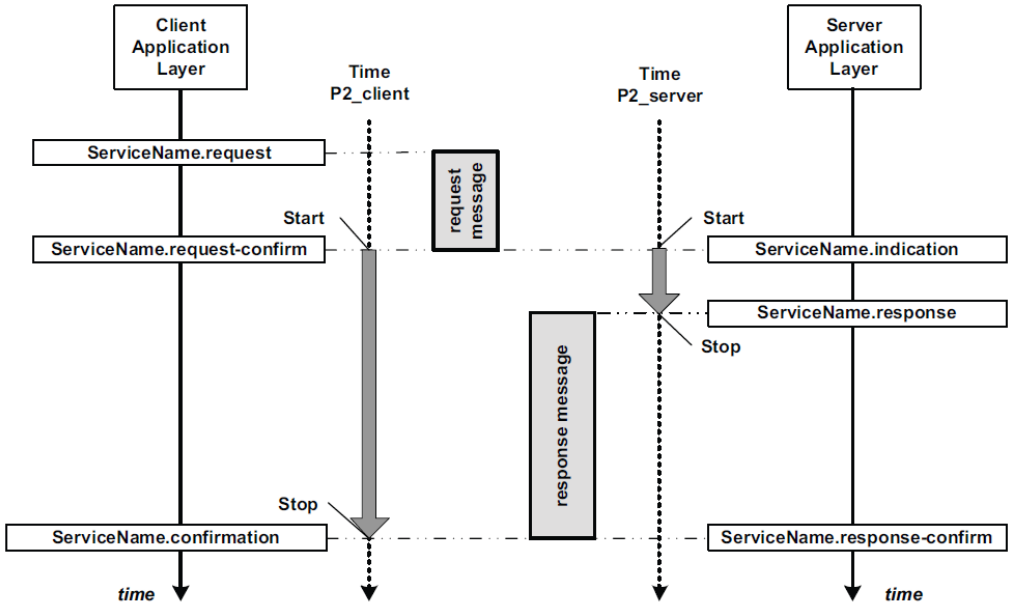
Overview
Related ISO Standards

- Derived and distributed based on OSI model
- UDS specification covers layer 5~7
- Service ID and associated parameters are contained in the payload of message frame
- Communication protocol covers layer 1~2



UDS in General
Application Layer – Services

- Also referred to as Diagnostic Services
- At client side: used to request certain diagnostic functions
- At server side: used to send response data back to client
- 6 service primitives are specified
  - Service request primitive
  - Service request-confirmation primitive
  - Service indication primitive
  - Service response primitive
  - Service response-confirmation primitive
  - Service confirmation primitive



Bosch Global Software Technologies
alt_future

UDS in General
Application Layer – Request Format

	Application Layer Protocol Data Unit (A_PDU)	
	Protocol Control Information (A_PCI)	Service Data Unit (A_SDU)
Request with only Service ID (SID)	SID	Opt. Data
Request with Service ID (SID) and Sub-function (SF)	SID	SF Opt. Data
Request with Service ID (SID) and Data ID (DID)	SID	DID DID Opt. Data
Request with Service ID (SID), Sub-function (SF) and Data ID (DID)	SID	SF DID DID Opt. Data

UDS in General
Application Layer – Response Format

	Application Layer Protocol Data Unit (A_PDU)	
	Protocol Control Information (A_PCI)	Service Data Unit (A_SDU)
Positive Response with only Service ID (SID)	SID + 0x40	Opt. Data
Positive Response with Service ID (SID) and Sub-function (SF)	SID + 0x40	SF Opt. Data
Positive Response with Service ID (SID) and Data ID (DID)	SID + 0x40	DID DID Opt. Data
Positive Response with Service ID (SID), Sub-function (SF) and Data ID (DID)	SID + 0x40	SF DID DID Opt. Data
Negative Response	0x7F SID	NRC

UDS in General
Application Layer – Service ID List

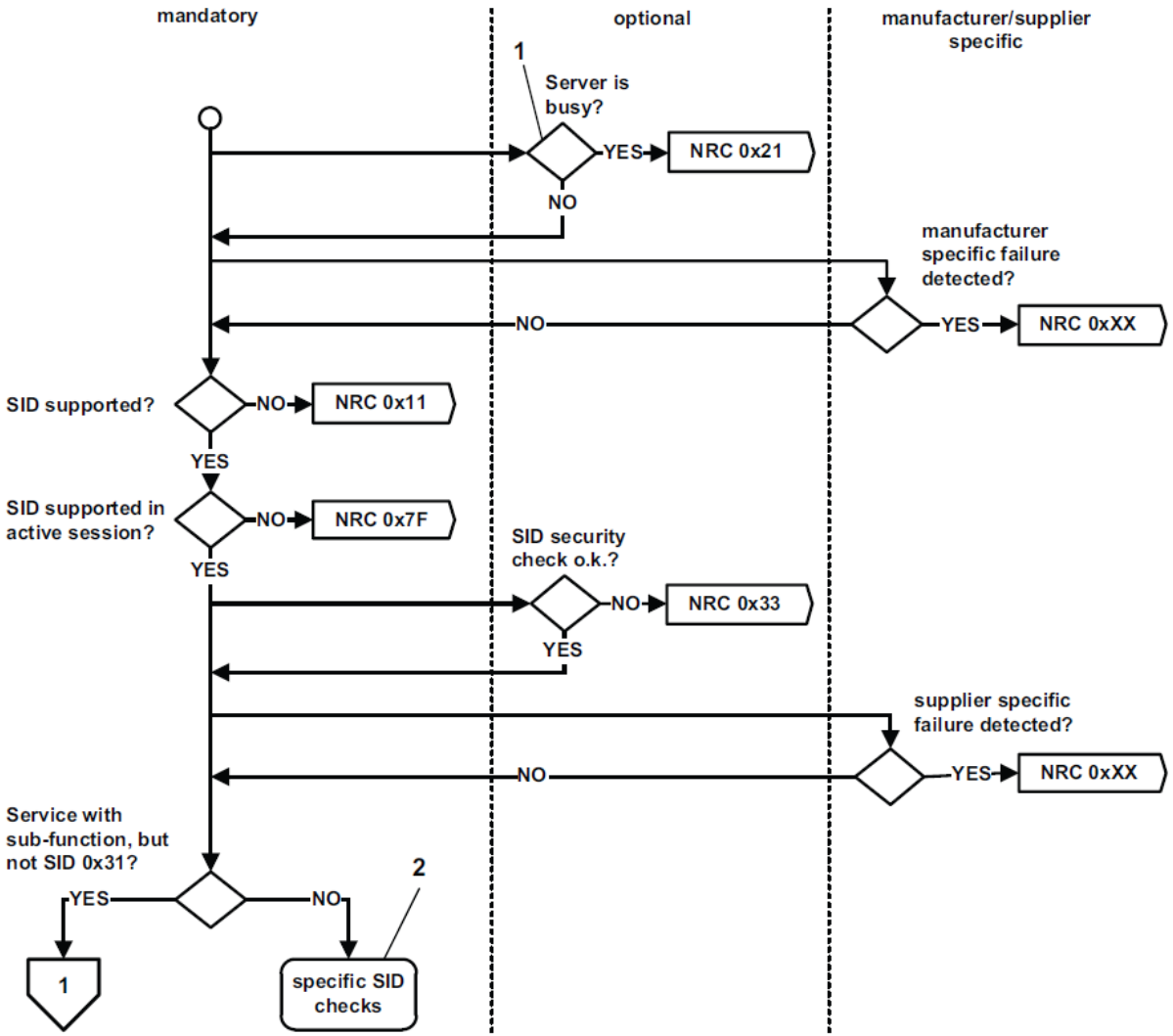
Service identifier (SI)	Service type (bit 6)	Where defined
0x10 – 0x3E	ISO 14229-1 service requests	ISO 14229-1
0x3F	Not applicable	Reserved by document
0x50 – 0x7E	ISO 14229-1 positive service responses	ISO 14229-1
0x7F	Negative response service identifier	ISO 14229-1
0x80 – 0x82	Not applicable	Reserved by ISO 14229-1
0x83 – 0x88	ISO 14229-1 service requests	ISO 14229-1
0x89 – 0xB9	Not applicable	Reserved by ISO 14229-1
0xBA – 0xBE	Service requests	Defined by system supplier
0xBF – 0xC2	Not applicable	Reserved by ISO 14229-1
0xC3 – 0xC8	ISO 14229-1 positive service responses	ISO 14229-1
0xC9 – 0xF9	Not applicable	Reserved by ISO 14229-1
0xFA – 0xFE	Positive service responses	Defined by system supplier
0xFF	Not applicable	Reserved by document

UDS in General
Application Layer – Common NRC List

Value	Response Code	Value	Response Code
0x00	positiveResponse	0x10	generalReject
0x11	serviceNotSupported	0x12	subFunctionNotSupported
0x13	incorrectMessageLengthOrInvalidFormat	0x21	busyRepeatRequest
0x22	conditionNotCorrect	0x24	requestSequenceError
0x31	requestOutOfRange	0x33	securityAccessDenied
0x35	invalidKey	0x36	exceedNumberOfAttempts
0x37	requiredTimeDelayNotExpired	0x72	generalProgrammingFailure
0x78	responsePending	0x7E	subFunctionNotSupportedInActiveSession
0x7F	serviceNotSupportedInActiveSession	0x83	engineIsRunning
0x88	vehicleSpeedTooHight	0x93	voltageTooLow

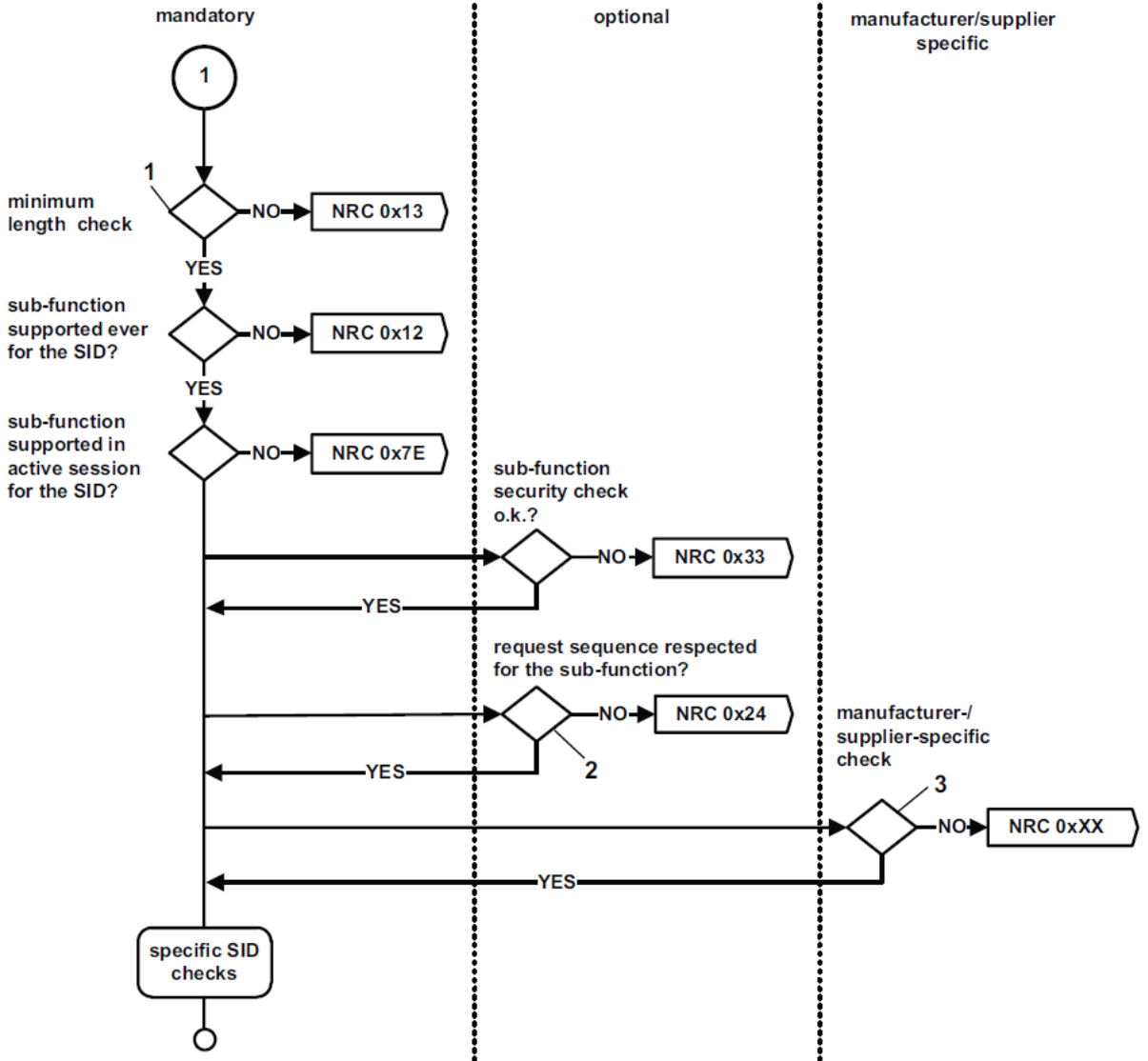
UDS in General
Application Layer – Server Response Implementation Rules

- General server response behavior upon receiving a request from client
- Services with sub-function and without sub-function are handled differently in next steps
- Service 0x31 (only service with both sub-function and data ID) are also handled differently



UDS in General
Application Layer – Server Response Implementation Rules

- General server response behavior upon receiving a request with sub-function is given in the picture
- There is no general server response behavior for requests without sub-function
- Server response behavior also depends on addressing mode (physical/functional) and on whether “suppress-positive-response” is requested



Software Technologies alt_future

UDS in General
Application Layer – List Of Standardized Services

SID	Service Name	SID	Service Name
Diagnostic & Communication Management			
0x10	Diagnostic Session Control	0x11	ECU Reset
0x27	Security Access	0x28	Communication Control
0x3E	Tester Present	0x83	Access Timing Parameter
0x84	Secured Data Transmission	0x85	DTC Control Setting
0x86	Response On Event	0x87	Link Control
Data Transmission			
0x22	Read Data By Identifier	0x2E	Write Data By Identifier
0x23	Read Memory By Address	0x3D	Write Memory By Address
0x23	Read Scaling Data By Identifier	0x2C	Dynamically Defined Data Identifier
0x2A	Read Data By Periodic Identifier		

UDS in General
Application Layer – List Of Standardized Services

SID	Service Name	SID	Service Name
Stored Data Transmission			
0x14	Clear Diagnostic Information	0x19	Read DTC Information
Input Output Control			
0x2F	Input Output Control By Identifier		
Routine			
0x31	Routine Control		
Upload Download			
0x34	Request Download	0x35	Request Upload
0x36	Transfer Data	0x37	Exit Transfer

UDS in General
Application Layer – Service $10 – Overview

- Service $10 – Diagnostic Session Control
- A diagnostic session enables a set of services/functions
- Only 1 active diagnostic session at a time
- System starts with default session at power up. Other sessions must be requested with service $10 to start

UDS in General
Application Layer – Service $10 – Request Format

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DiagnosticSessionControl Request SID	M	0x10	DSC
#2	sub-function = [diagnosticSessionType]	M	0x00 – 0xFF	LEV_DS_

Diagnostic Session Type:

- 0x00: reserved
- 0x01: default session
- 0x02: programming session
- 0x03: extended session
- 0x04: safety system diagnostic session
- 0x05-0x3F: reserved
- 0x40-0x5F: vehicle manufacture specific
- 0x60-0x7E: supplier specific
- 0x7F: reserved

UDS in General
Application Layer – Service $10 – Positive Response Format

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DiagnosticSessionControl Response SID	M	0x50	DSCPR
#2	sub-function = [diagnosticSessionType]	M	0x00 – 0xFF	LEV_DS_
#3 : #6	sessionParameterRecord[]#1 = [data#1 : data#4]	M : M	0x00 – 0xFF : 0x00 – 0xFF	SPREC_ DATA_1 : DATA_m

- Diagnostic Session Type

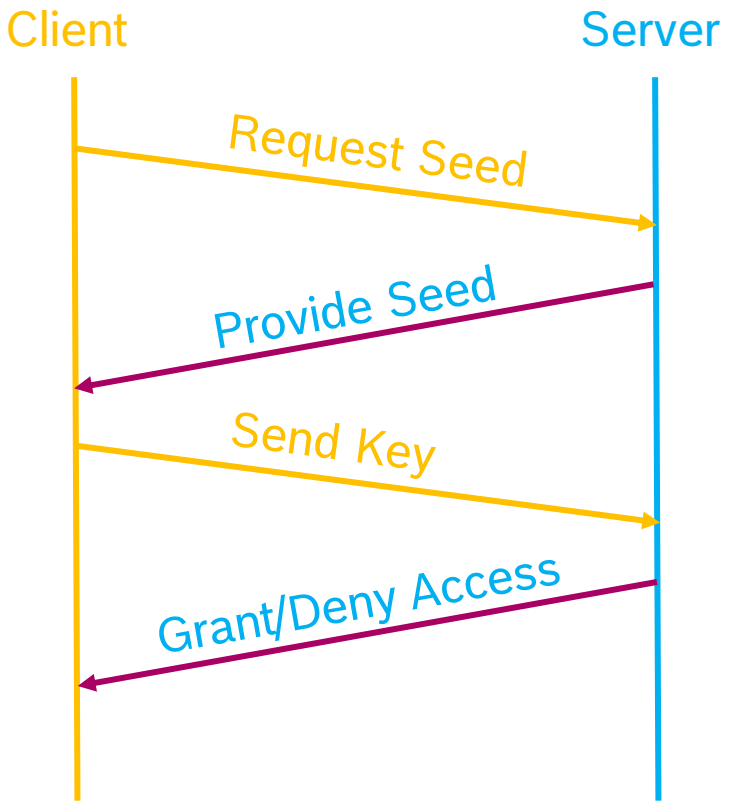
(same as in request format)

- Session Parameter Record

Byte pos. in record	Description	Cvt	Byte Value	Mnemonic
	sessionParameterRecord[] = [			SPREC_
#1	$P2_{Server\_max}$ (high byte)	M	0x00 – 0xFF	P2SMH
#2	$P2_{Server\_max}$ (low byte)	M	0x00 – 0xFF	P2SML
#3	$P2^*_{Server\_max}$ (high byte)	M	0x00 – 0xFF	P2ESMH
#4	$P2^*_{Server\_max}$ (low byte)]	M	0x00 – 0xFF	P2ESML

UDS in General
Application Layer – Service $27 - Overview

- Service $27 – Security Access
- Provide a mean to access data/services
- Implement using seed/key relationship
- Formula to calculate key from seed is pre-determined and known by both client and server
- Access is granted when key calculated by client matches with that by server



UDS in General
Application Layer – Service $27 - Overview

- Security Level

Security Level	1	2	3	...	n	...
Seed-related sub-function	1	3	5	...	2n-1	...
Key-related sub-function	2	4	6	...	2n	...

- Security levels are independent and non-arbitration from each other
- At any moment, only 1 security level can be in unlock state
- Data/services can be assigned to more than 1 security level. Access is granted when any of those is in unlock state

UDS in General
Application Layer – Service $27 – Request Format

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	SecurityAcces Request SID	M	0x27	SA
#2	sub-function = [securityAccessType = requestSeed]	M	0x01, 0x03, 0x05, 0x07 – 0x7D	LEV_ SAT_RSD
#3 : #n	securityAccessDataRecord[] = [parameter#1 : parameter#m]	U : U	0x00 – 0xFF : 0x00 – 0xFF	SECACCDR_ PARA1 : PARAm

- Security Access Data Record (optional): information about the seed

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	SecurityAcces Request SID	M	0x27	SA
#2	sub-function = [securityAccessType = sendKey]	M	0x02, 0x04, 0x06, 0x08 – 0x7E	LEV_SAT_SK
#3 : #n	securityKey[] = [key#1 (high byte) : key#m (low byte)]	M : U	0x00 – 0xFF : 0x00 – 0xFF	SECKEY_ KEY1HB : KEYmLB

UDS in General
Application Layer – Service $27 – Response Format

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	SecurityAccess Response SID	M	67	SAPR
#2	sub-function = [securityAccessType]	M	00-7F	LEV_SAT_SK
#3 : #n	securitySeed[] = [seed#1 (high byte) : seed#m (low byte)]	C : C	0x00 – 0xFF : 0x00 – 0xFF	SECSEED_ SEED1HB : SEEDmLB
C: The presence of this parameter depends on the securityAccessType parameter. It is mandatory to be present if the securityAccessType parameter indicates that the client wants to retrieve the seed from the server.				

- Security Access Type in the response is echoed from the request
- Seed length and algorithm is defined by user (usually a random generation algorithm)
- Key length and algorithm is defined by user.
- Seed length and key length are not necessarily be same

UDS in General
Application Layer – Service $22/$2E – Overview

- Service $22 – Read Data by Identifier
- Service $2E – Write Data by Identifier

- Data Identifier (DID): 2 byte – mark for an internal location of a data record

- DID along with referred data length and format are pre-defined by user

- Service $2E might involve NVM-related services to store data

- Service $2E might involve certain security check methods

UDS in General
Application Layer – Service $22/$2E – Request Format

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteDataByIdentifier Request SID	M	0x2E	WDBI
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	byte#2]	M	0x00 – 0xFF	LB
	dataRecord[] = [			DREC_
#4	data#1	M	0x00 – 0xFF	DATA_1
:	:	:	:	:
#m+3	data#m]	U	0x00 – 0xFF	DATA_m

- Data from multiple DIDs can be read in 1 single request
- Maximum number of DIDs must be pre-determined

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteDataByIdentifier Request SID	M	0x2E	WDBI
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	byte#2]	M	0x00 – 0xFF	LB
	dataRecord[] = [			DREC_
#4	data#1	M	0x00 – 0xFF	DATA_1
:	:	:	:	:
#m+3	data#m]	U	0x00 – 0xFF	DATA_m

UDS in General
Application Layer – Service $22/$2E – Response Format

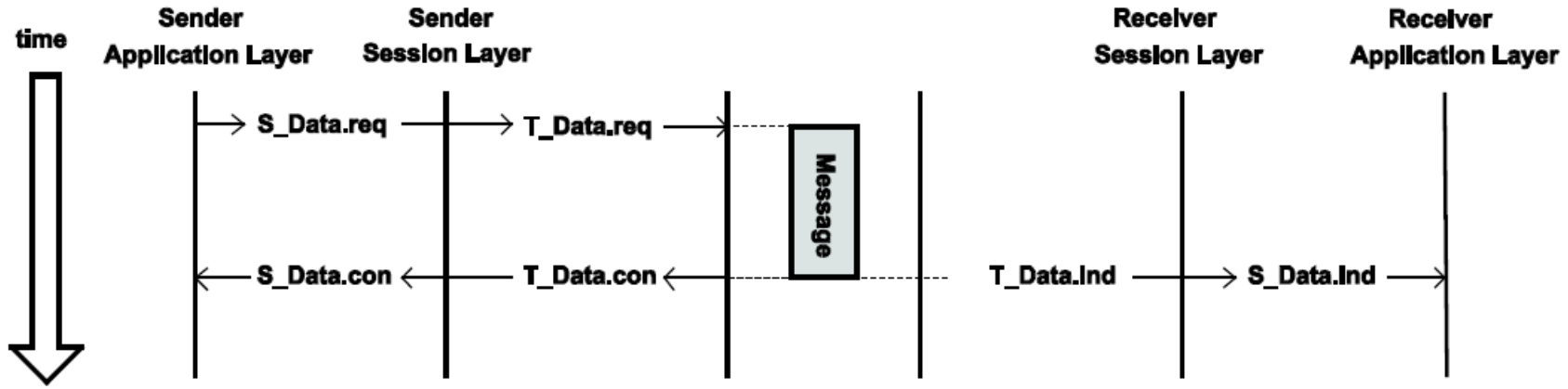
A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDataByIdentifier Response SID	M	0x62	RDBIPR
#2 #3	dataIdentifier[]#1 = [byte#1 (MSB) byte#2]	M M	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB
#4 : #(k-1)+4	dataRecord[]#1 = [data#1 : data#k]	M : U	0x00 – 0xFF : 0x00 – 0xFF	DREC_ DATA_1 : DATA_m
:	:	:	:	:
#n-(o-1)-2 #n-(o-1)-1	dataIdentifier[]#m = [byte#1 (MSB) byte#2]	U U	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB
#n-(o-1) : #n	dataRecord[]#m = [data#1 : data#o]	U : U	0x00 – 0xFF : 0x00 – 0xFF	DREC_ DATA_1 : DATA_k

- For multiple DIDs read request, data is provided in the response along with the respective DID

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteDataByIdentifier Response SID	M	0x6E	WDBIPR
#2 #3	dataIdentifier[] = [byte#1 (MSB) byte#2]	M M	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB

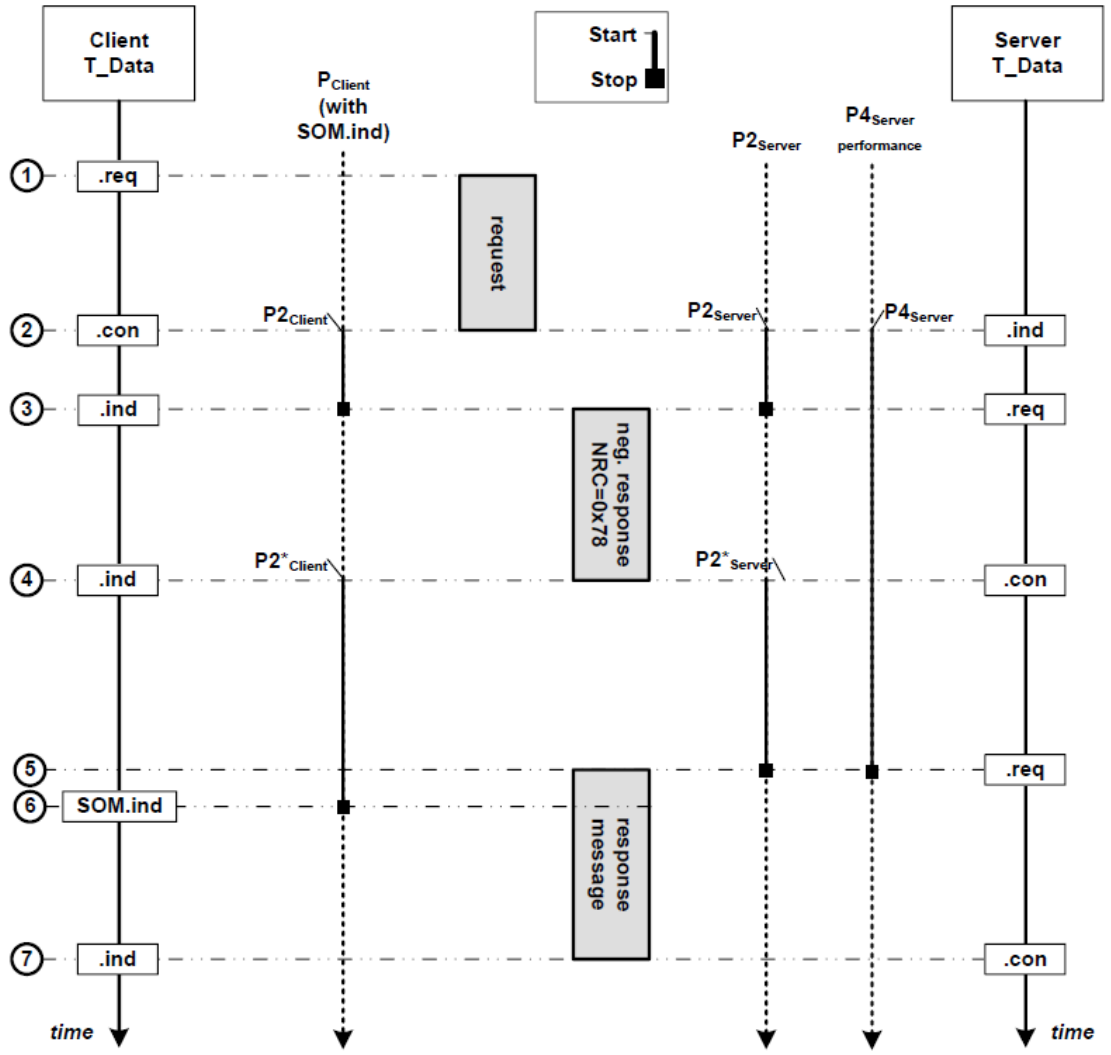
UDS in General
Session Layer - Services

- E.g.: transmission/reception of data, setting of protocol parameters
- Primitives: request, indication, confirm



UDS in General
Session Layer – Timing Parameter – P2, P2*, P4

- P2: from receiving of request to start of 1st response (can be NRC 0x78)
- P2*: from receiving NRC 0x78 to start of next response (can also be NRC 0x78)
- P4: from receiving of request to start of final response (cannot be 0x78)
- Note: delay between layers and transmission medium should be accounted for
- Note: from server side, a *max* threshold is also defined for each parameter



Bosch Global Software Technologies
alt_future

UDS in CAN Implementation
Application Layer

SID	Service Name	SID	Service Name
Diagnostic & Communication Management			
0x10	Diagnostic Session Control	0x11	ECU Reset
0x27	Security Access	0x28	Communication Control
0x3E	Tester Present	0x83	Access Timing Parameter
0x84	Secured Data Transmission	0x85	DTC Control Setting
0x86	Response On Event	0x87	Link Control
Data Transmission			
0x22	Read Data By Identifier	0x2E	Write Data By Identifier
0x23	Read Memory By Address	0x3D	Write Memory By Address
0x23	Read Scaling Data By Identifier	0x2C	Dynamically Defined Data Identifier
0x2A	Read Data By Periodic Identifier		

UDS in CAN Implementation
Application Layer

SID	Service Name	SID	Service Name
Stored Data Transmission			
0x14	Clear Diagnostic Information	0x19	Read DTC Information
Input Output Control			
0x2F	Input Output Control By Identifier		
Routine			
0x31	Routine Control		
Upload Download			
0x34	Request Download	0x35	Request Upload
0x36	Transfer Data	0x37	Exit Transfer

Q & A

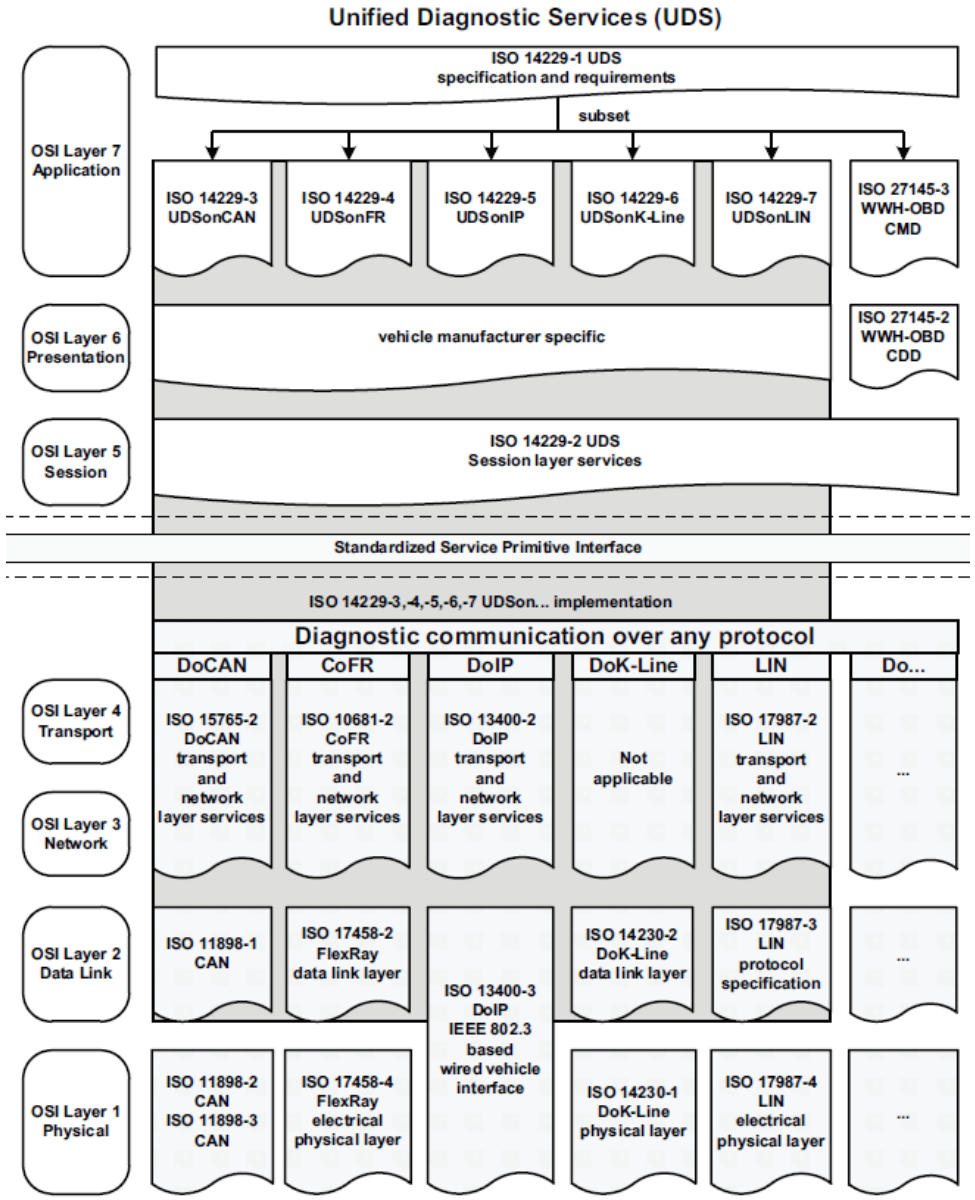
DIAGNOSTIC COMMUNICATION OVER CAN

Agenda

1. Overview
    1. Diagnostic over any protocol
    2. OSI encapsulation model
2. Diagnostic over CAN
    1. Problem
    2. Single-frame transmission
    3. Multi-frame transmission
    4. N_PDU definition
    5. Timing parameters

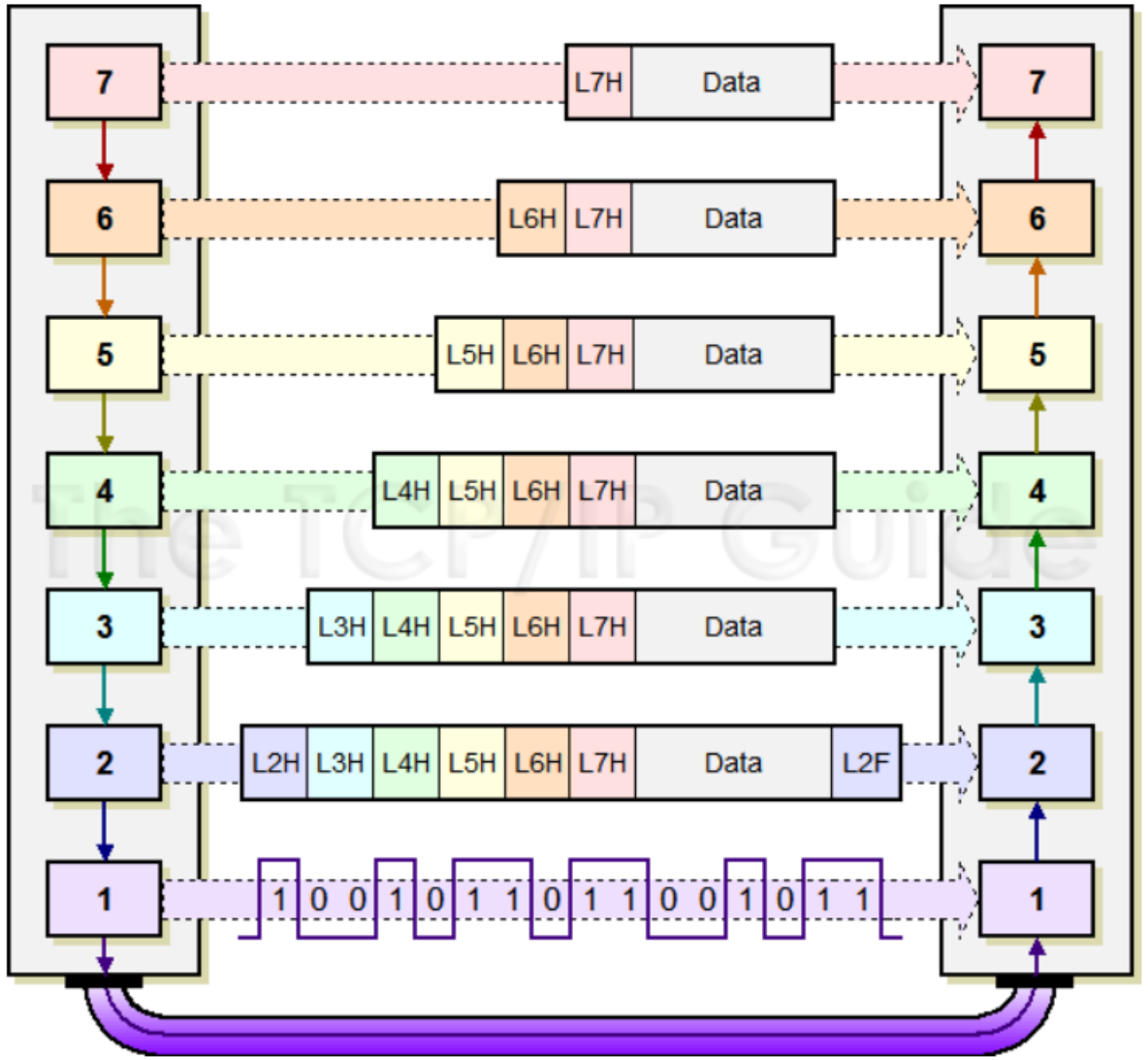
Overview
Diagnostic Over Any Protocol

- Diagnostic communication protocols (like UDS) cover the upper layers (5, 6, 7)
- Communication protocols (like CAN) cover the lower layers (1, 2)
- Diagnostic over <protocol> covers Network layer (3) and Transport layer (4)
- Diagnostic over <protocol> ensure diagnostic data from upper layers can be transmitted/received using <protocol> communication

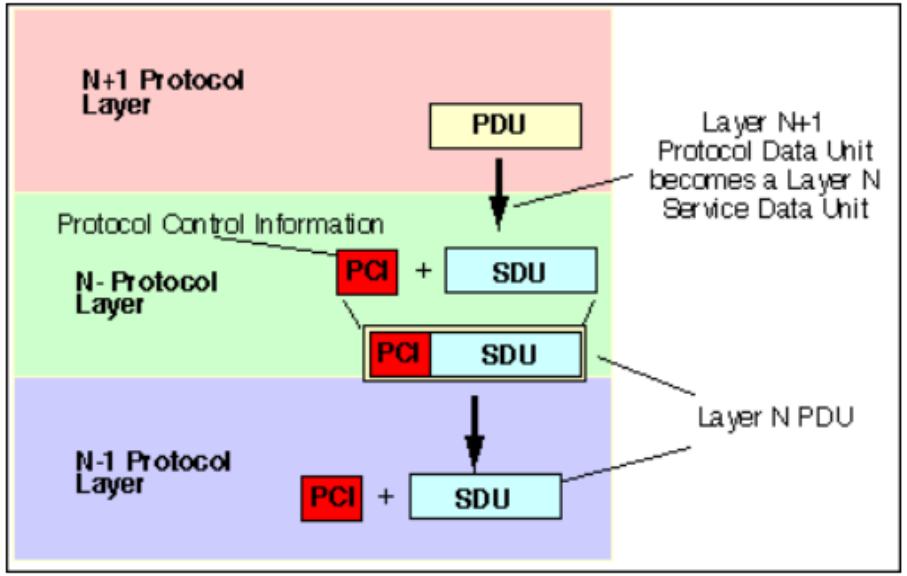


Overview
OSI Encapsulation Model

- Layer header is included in the data before passing to the adjacent lower layer
- Content of the header is related to the protocol/format used at that level
- Data and header of N^{th}-layer will become data of $N\text{-}1^{th}$-layer
- Header included in the N-layer at sender side will later be used by same N-layer at receiver side to process the data



Overview
OSI Encapsulation Model



- Terminology:
  - SDU – Service Data Unit (payload)
  - PCI – Protocol Control Information (header)
  - PDU – Protocol Data Unit

- Note:
  - N^{th}-PCU is added by N^{th}-layer of sender, then removed by N^{th}-layer of receiver
  - Not always all 7 layers are present, processing data or adding header
  - For the rest of this document, PDU provided by network layer are denoted as N_PDU, similarly for N_PCI and N_SDU

Diagnostic Over CAN
Problem - Overview

- Requirement: ECU must be able to provide Vehicle Identification Number (VIN) via a diagnostic read service

Example

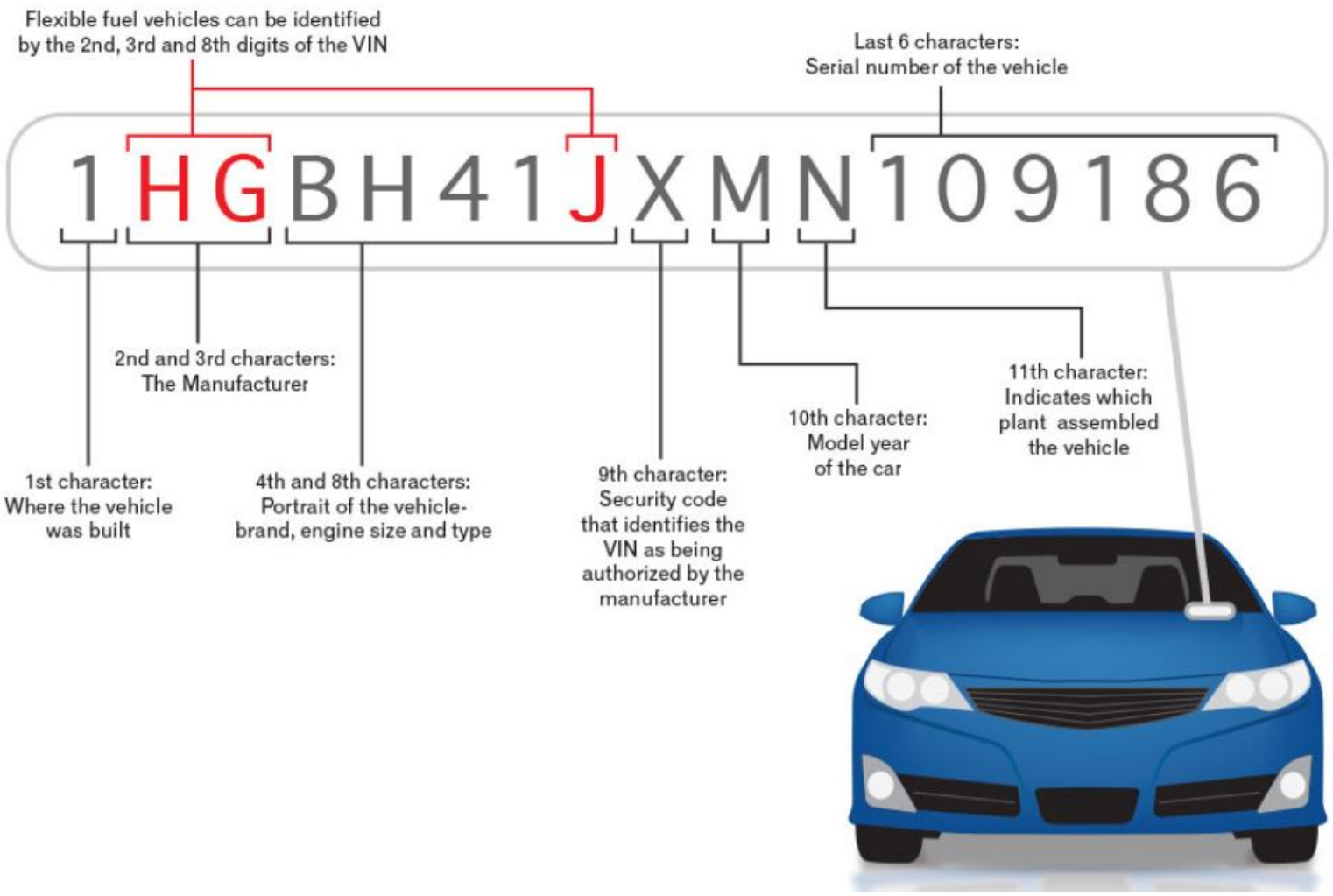
Tester: 22 F1 90

ECU: 62 F1 90 31 48 47 42 48 34 31 58 4A 4D 4E 31 30 39 31 38 36

(ASCII translation: 1HGBH41XJMN109186)

- Problem:
  - VIN according to automotive standard are 17-byte long
  - UDS protocol requires 3 more bytes to specify for the service
  - CAN protocol only allows 8 bytes in the data field at max

Diagnostic Over CAN
Problem – VIN Format



Bosch Global Software Technologies
alt_future

Diagnostic Over CAN
Problem – CAN Frame Format

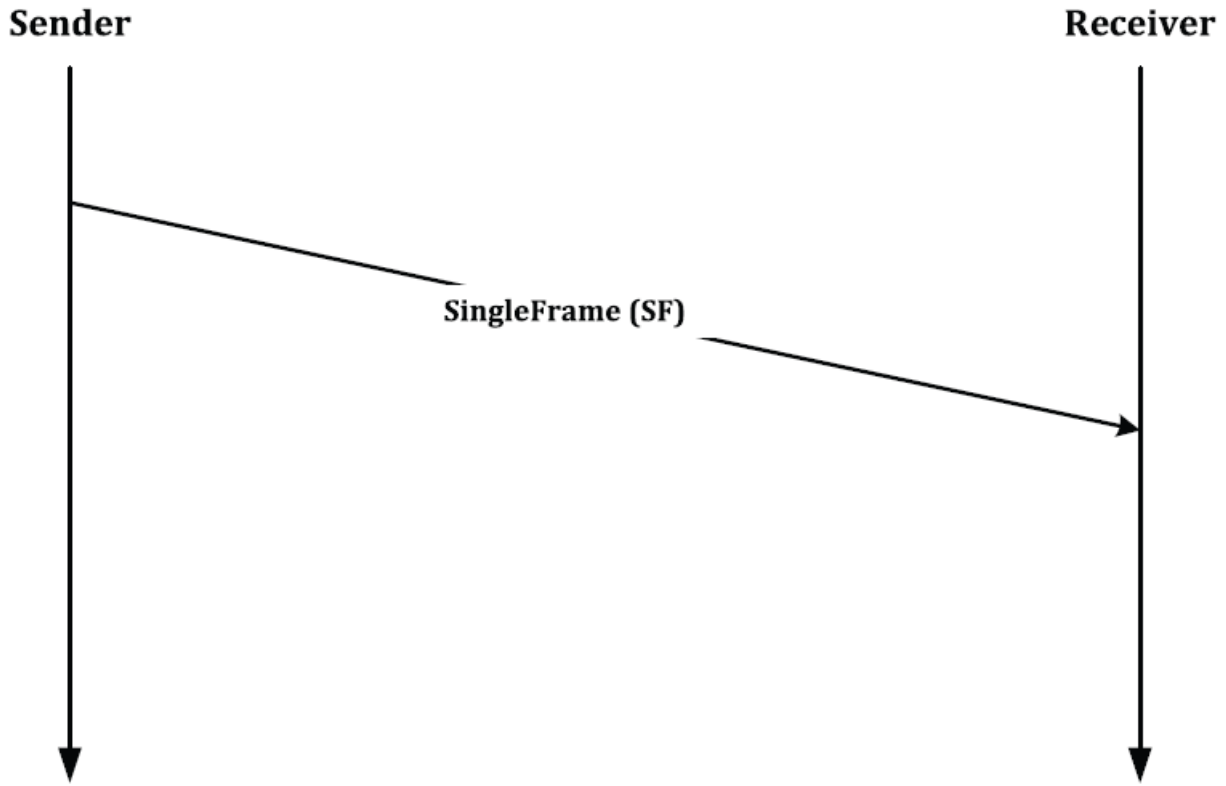
SOF	Arbitration Field												Control Field						Data Field								CRC Field		ACK Field		EOF						
	Identifier											RTR	Res		DLC				Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #1	Deli	Ack	Deli							
0	10	9	8	7	6	5	4	3	2	1	0	0	5	4	3	2	1	0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	0	1	0	6	5	4	3	2	1	0

CANNOT FIT IN???

UDS Protocol			VIN Data															
Byte #1	Byte #2	Byte #3	Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #9	Byte #10	Byte #11	Byte #12	Byte #13	Byte #14	Byte #15	Byte #16	Byte #17
7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
62	F1	90	31	48	47	42	48	34	31	58	4A	4D	4E	31	30	39	31	38	36

Diagnostic Over CAN
Single-frame Transmission

- Data can be fitted in single CAN frame
- No segmentation is needed
- Network layer data packet for this case is call SingleFrame (SF)

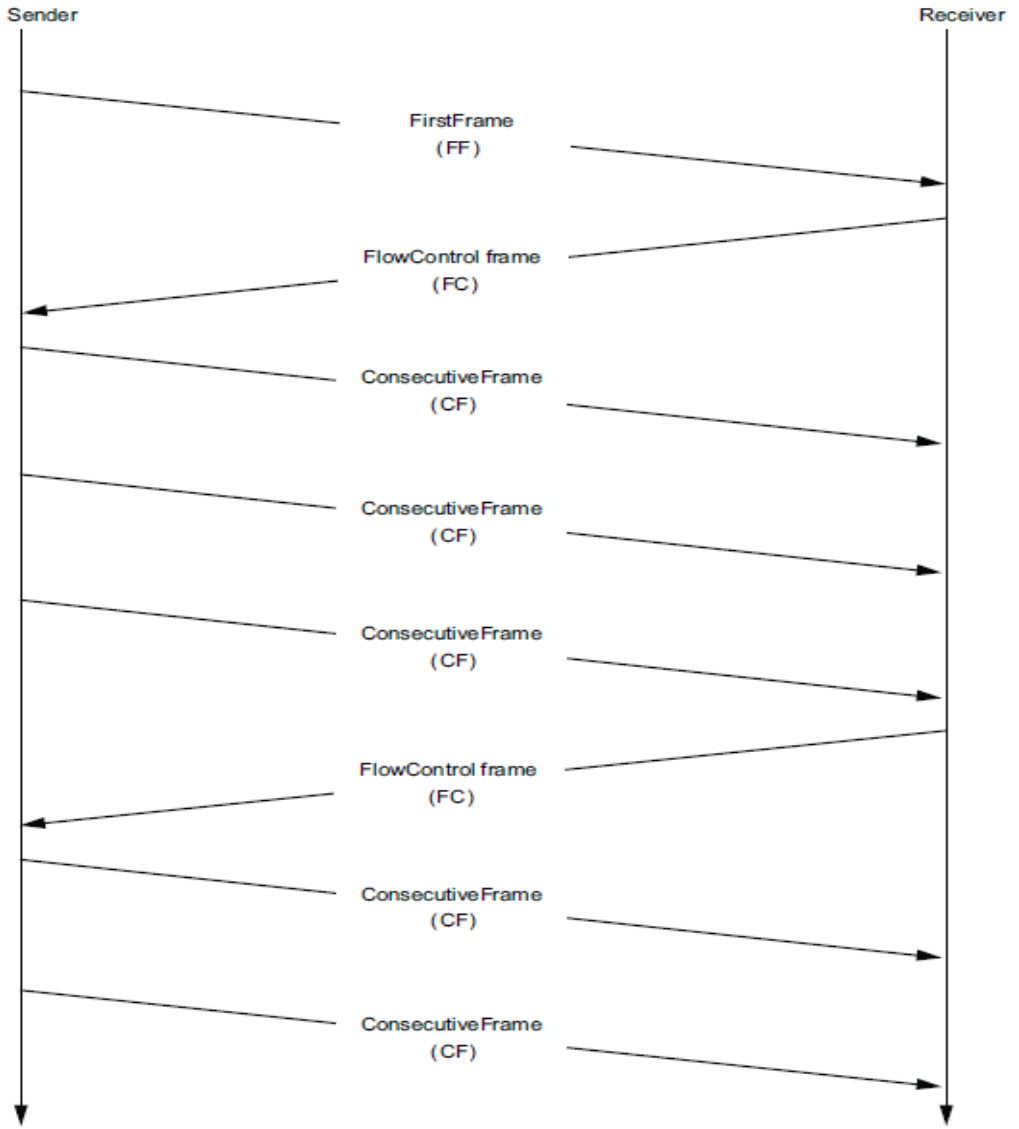


Diagnostic Over CAN
Multi-frame Transmission

- Data doesn't fit a single CAN frame

- Network layer segments the data into multiple CAN frames (FF and CF)
  - FirstFrame (FF) contains the total length
  - ConsecutiveFrame (CF) contains respective sequence number

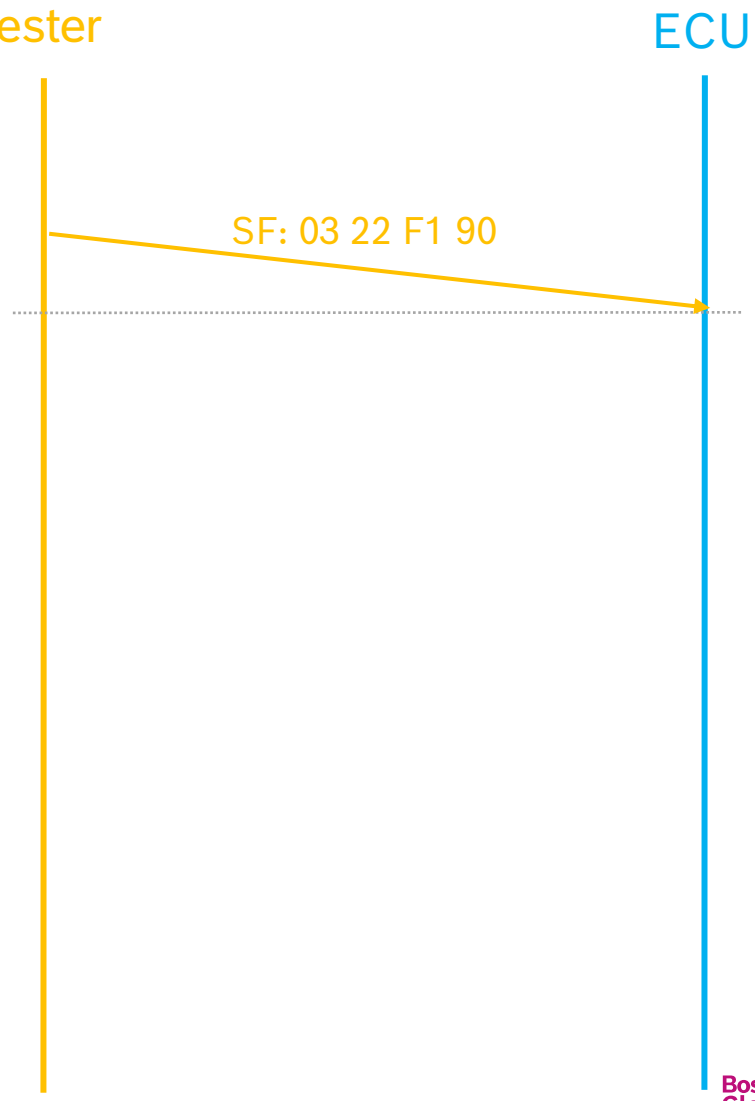
- FlowControl is provided by receiver, to adjust the sender to the network layer capacities of receiver



Diagnostic Over CAN
N_PDU Definition – Single Frame

SOF	Arbitration Field											Control Field					Data Field								CRC Field		ACK Field		EOF
	Identifier											RTR	Res		DLC		Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #1	Deli	Ack	Deli	
0	10	9	8	7	6	5	4	3	2	1	0	0	5	4	3	2	1	0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	0	1	0	6 5 4 3 2 1 0

N_PDU								
N_PCI		N_SDU						
Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	DL							
0	3	22	F1	90	x	x	x	x

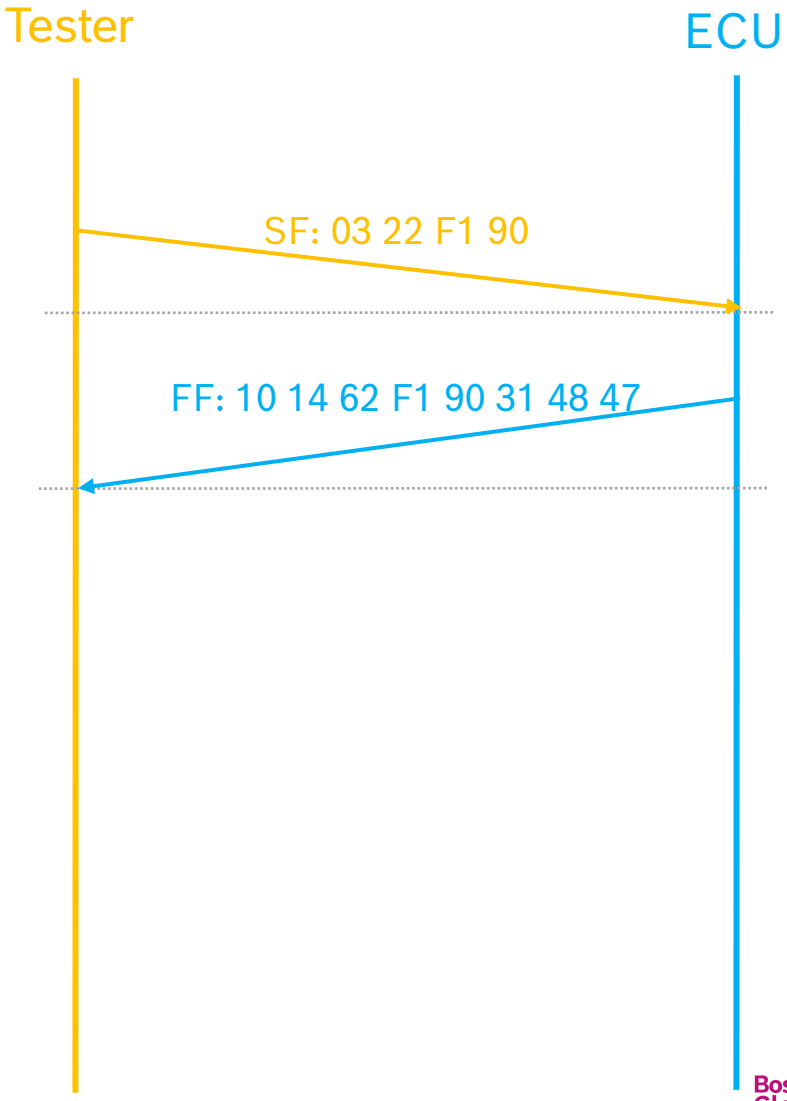


- FT: Frame Type: 0x0 for Single Frame
- DL: Data Length, maximum 7 for Single Frame

Diagnostic Over CAN
N_PDU Definition – First Frame

SOF	Arbitration Field											Control Field					Data Field								CRC Field		ACK Field		EOF
	Identifier										RTR	Res		DLC			Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #1	Deli	Ack	Deli	
0	10	9	8	7	6	5	4	3	2	1	0	0	5	4	3	2	1	0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	0	1	0	6 5 4 3 2 1 0

N_PDU											
N_PCI				N_SDU							
Byte #1		Byte #2		Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8		
7-4	3-0	7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0		
FT	DL										
1	0	1	4	62	F1	90	31	48	47		



- FT: Frame Type: 0x1 for First Frame
- DL: total Data Length, maximum 4096 for First Frame
- First Frame contains 1st 6 bytes of total payload

Diagnostic Over CAN
N_PDU Definition – Flow Control

SOF	Arbitration Field											Control Field					Data Field								CRC Field		ACK Field		EOF
	Identifier										RTR	Res		DLC			Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #1	Deli	Ack	Deli	
0	10	9	8	7	6	5	4	3	2	1	0	0	5	4	3	2	1	0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	0	1	0	6 5 4 3 2 1 0

N_PDU								
N_PCI				N_SDU				
Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	FS	BS	ST min	N/A	N/A	N/A	N/A	N/A
3	0	00	00	x	x	x	x	x

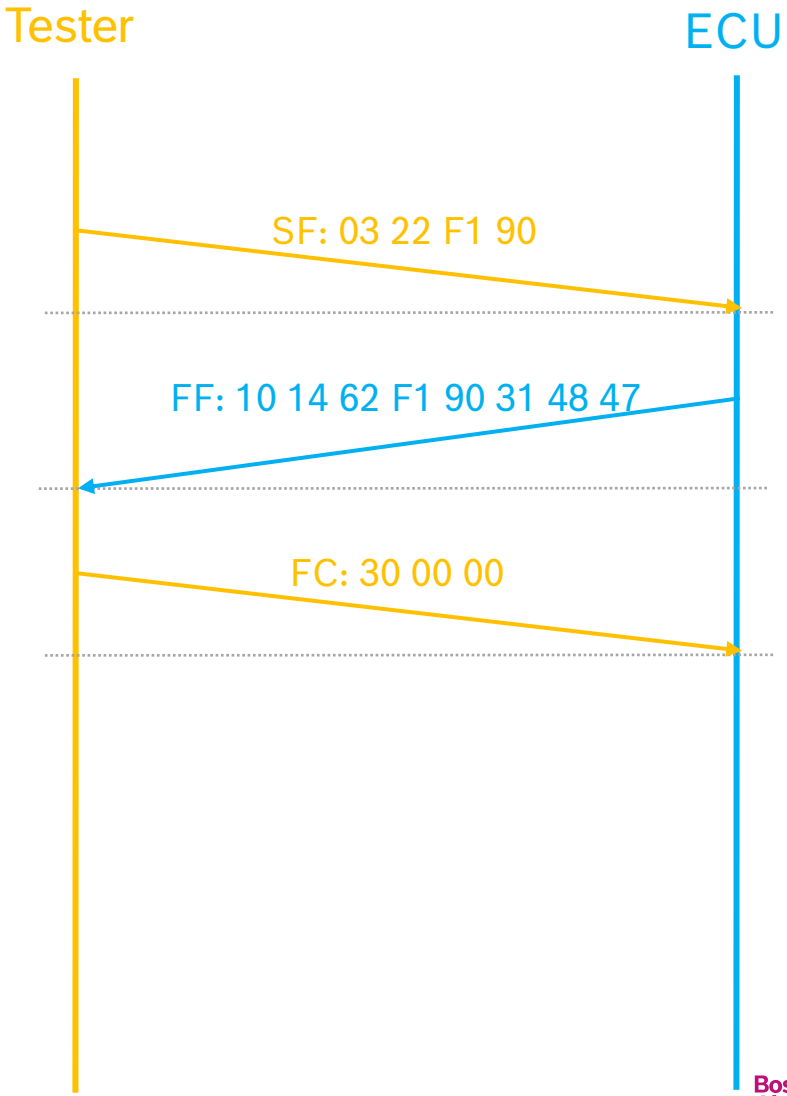
- Note: no data for FC

- FT: Frame Type: 0x3 for Flow Control
- FS: Flow State
  - 0x0: CTS – continue to send
  - 0x1: WT – wait
  - 0x2: OVFLW – overflow
- BS: Block Size
  - 0x00: no more FC needed
  - 0x01..0xFF: max number of CFs can be sent until the next FC
- STmin: Separation Time Minimum
  - 0x00..0x7F: 0..127 ms
  - 0xF1..0xF9: 100..900 us

Diagnostic Over CAN
N_PDU Definition – Flow Control

SOF	Arbitration Field												Control Field					Data Field								CRC Field		ACK Field		EOF						
	Identifier											RTR	Res		DLC			Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #1	Deli	Ack	Deli							
0	10	9	8	7	6	5	4	3	2	1	0	0	5	4	3	2	1	0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	0	1	0	6	5	4	3	2	1	0

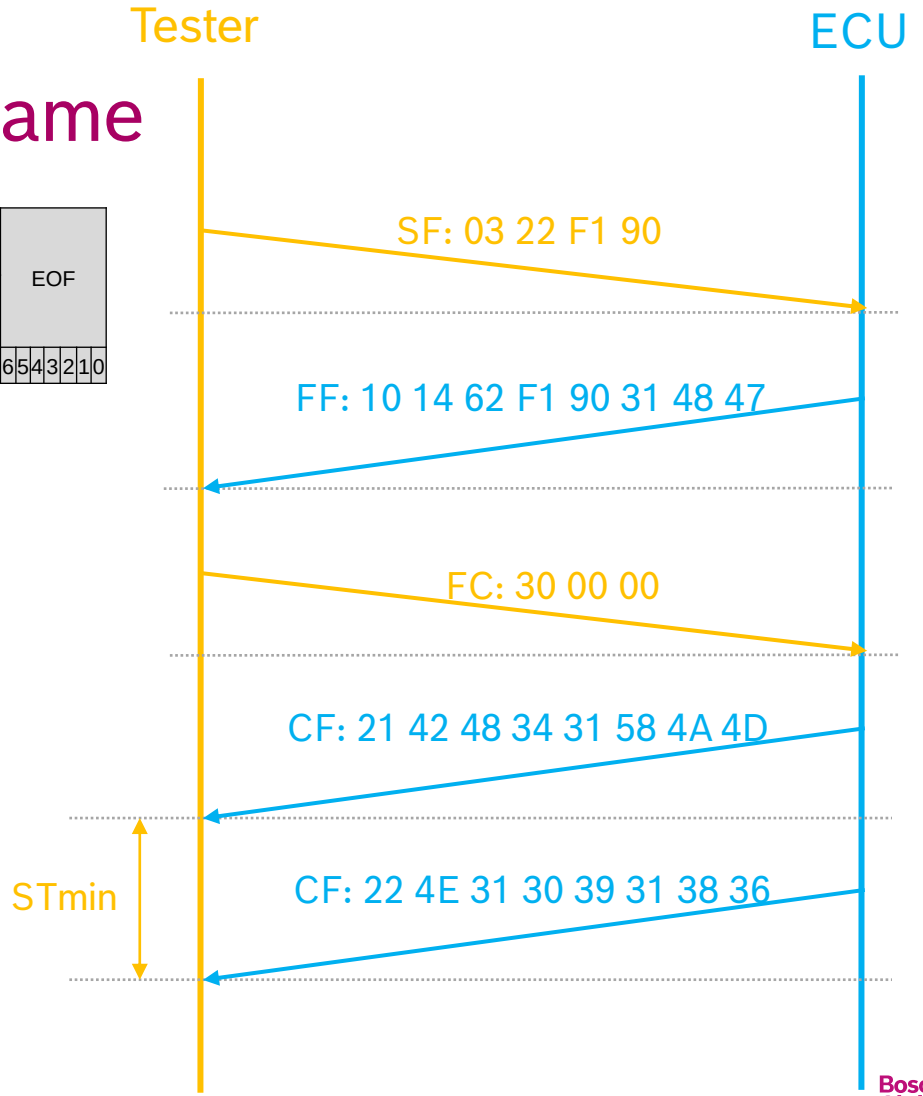
N_PDU								
N_PCI				N_SDU				
Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	FS	BS	ST min	N/A	N/A	N/A	N/A	N/A
3	0	00	00	x	x	x	x	x



Diagnostic Over CAN
N_PDU Definition – Consecutive Frame

SOF	Arbitration Field											Control Field					Data Field								CRC Field		ACK Field		EOF						
	Identifier											RTR	Res		DLC		Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #1	Deli	Ack	Deli								
0	10	9	8	7	6	5	4	3	2	1	0	0	5	4	3	2	1	0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	0	1	0	6	5	4	3	2	1	0

N_PDU								
N_PCI		N_SDU						
Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	SN							
2	1	42	48	34	31	58	4A	4D
2	2	4E	31	30	39	31	38	36



- FT: Frame Type: 0x2 for Consecutive Frame
- SN: Sequence Number, initial 0x1, range 0x0..0xF

Diagnostic Over CAN
N_PDU Definition - Summary

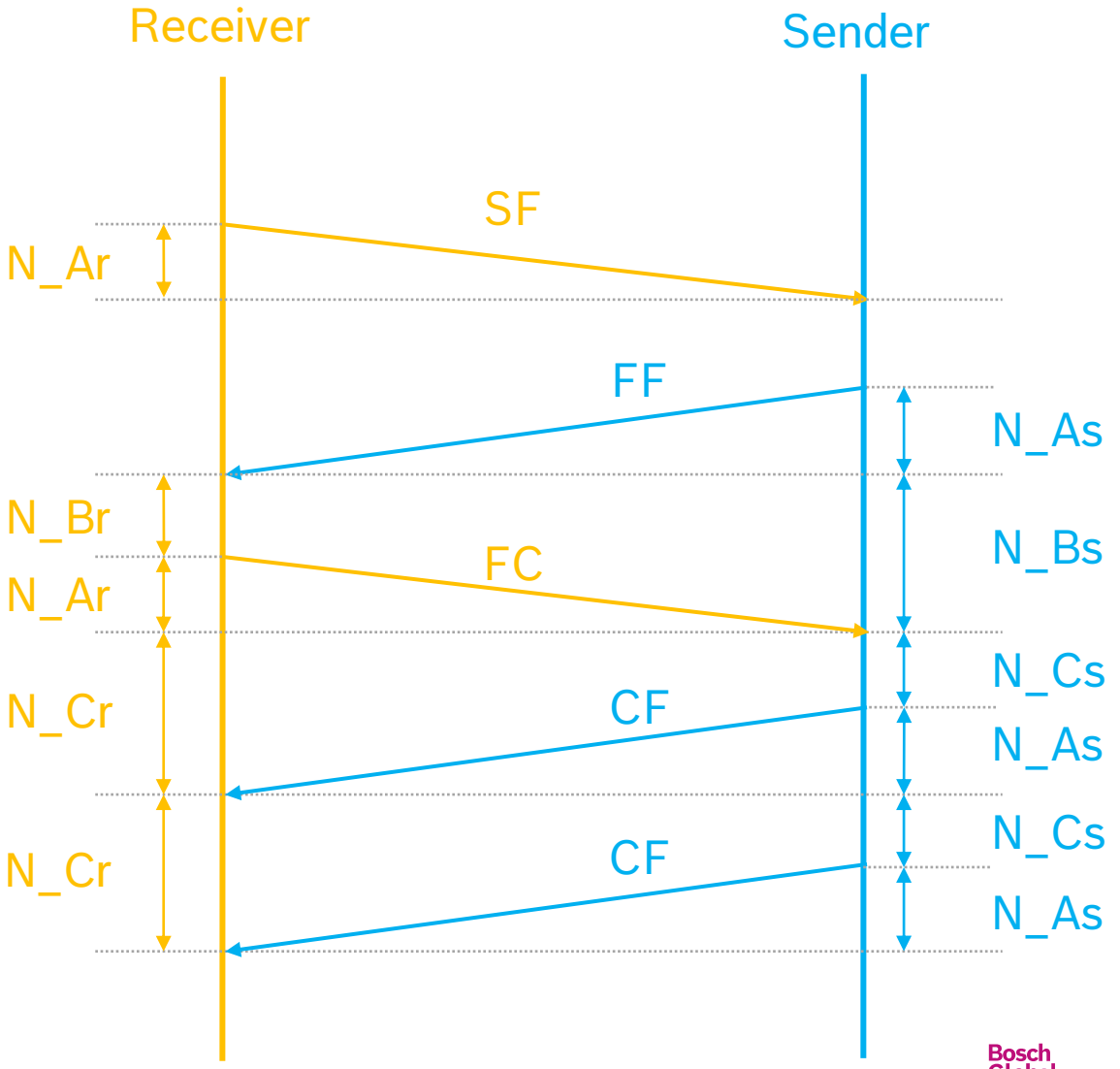
	N_PDU								
	Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
	7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
Single Frame	0	DL	Data						
First Frame	1	DL		Data					
Consecutive Frame	2	SN	Data						
Flow Control	3	FS	BS	Stmin	N/A				

N_PCI

N_SDU

Diagnostic Over CAN
Timing Parameters

Timing Parameter	Description	Timeout (ms)	Performance (ms)
N_As	Time for transmission of the CAN frame (any N_PDU) on the sender side	1000	N/A
N_Ar	Time for transmission of the CAN frame (any N_PDU) on the receiver side	1000	N/A
N_Bs	Time until reception of the next FC N_PDU	1000	N/A
N_Br	Time until transmission of the next FC N_PDU	N/A	(N_Br + N_Ar) < (0.9 * N_Cr)
N_Cs	Time until transmission of the next CF N_PDU	N/A	(N_Br + N_Ar) < (0.9 * N_Cr)
N_Cr	Time until reception of the next CF N_PDU	1000	N/A



Bosch Global Software Technologies
alt_future

Thank you!

Bosch
Global
Software
Technologies
alt_future