Vehicular Communications

Prof. Gianluigi Ferrari, Prof. Luca Davoli

Internet of Things (IoT) Lab

Department of Engineering and Architecture

University of Parma

https://iotlab.unipr.it

Lecture 3

Intra-vehicle Communications

Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

Outline

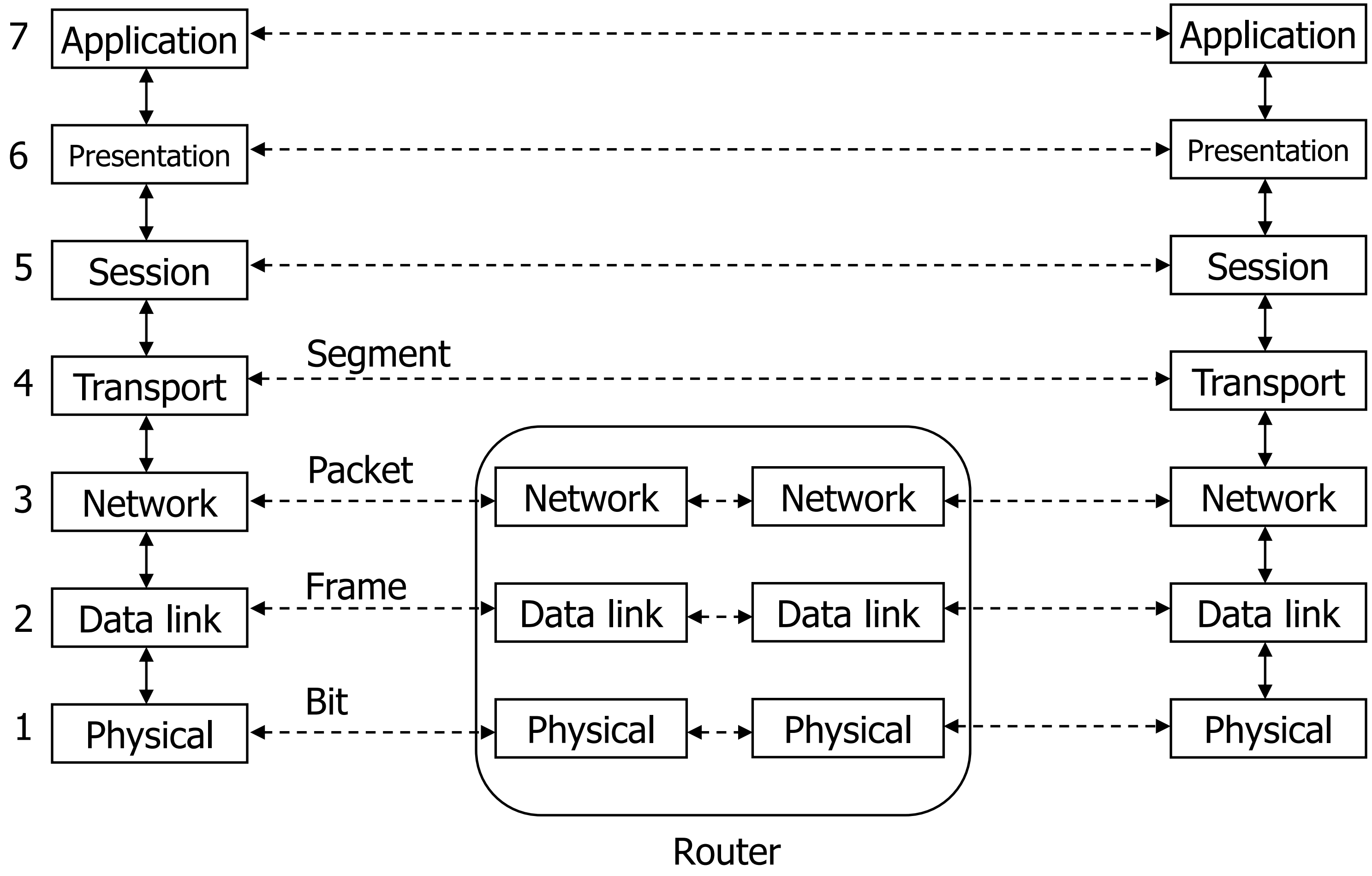
- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

ISO/OSI Layers

- Layered communication architecture
  - One layer ⇔ one function ⇔ one protocol
  - Layer interacts only with immediate base layer
  - Interfaces follow rigid specification
    - commonly by standards body

- ISO/OSI layered communication model
  - Defines 7 layers
    - see next slide
  - Common architectures relax rigid guidelines
    - cf. TCP/IP

ISO/OSI Layers: Router



ISO/OSI Layers: Functions in Detail

- Physical Layer
  - Specifies mechanical, electrical properties to transmit bits
  - Time synchronization, coding, modulation, ...
- Data Link Layer
  - Checked transmission of frames
  - Frame synchronisation, error checking, flow control, ...
- Network Layer
  - Transmission of datagrams / packets
  - Connection setup, routing, resource management, …
- Transport Layer
  - Reliable end to end transport of segments

ISO/OSI Layers: Functions in Detail

- Session Layer
  - Establish and tear down sessions
- Presentation Layer
  - Define Syntax and Semantics of information
- Application Layer
  - Communication between applications

- Our focus in this lecture:
  - Physical Layer
  - Data Link Layer

Why bus systems?

- Lower cost
  - Material
  - Weight
  - Volume

- Higher modularity
  - customizability
    of vehicles
  - cooperation with Original Equipment Manufacturers (OEMs)

- Shorter development cycles
  - Re-usability of components
  - Standard protocols and testing plans ⇨ less errors

History

- First micro processors in vehicles in 1980s
- Communication via point to point connections
- Simple control lines, little real data transmission
- True data transmission for connection external diagnosis equipment
- Birth of standard for character transmission
  - via K-Line (ISO 9141)
- Finally: introduction of data busses for in-vehicle communication
- Later standardized as CAN (ISO 11898)
- Use in series production models starts 1991

Overview and Use Cases

- State of the art
  - K-Line and CAN are part of On Board Diagnosis (OBD) connector
  - Enables, e.g., reading engine parameters, catcon, oxygen (lambda) sensor
  - Mandatory for newly registered vehicles in both EU und U.S.

Use Cases

- Driveline
  - Engine and transmission control
- Active Safety
  - Electronic Stability Programme (ESP)
- Passive Safety
  - Air bag, belt tensioners
- Comfort
  - Interior lighting, A/C automation
- Multimedia and Telematics
  - Navigation system, CD changer

Classification: On board communication

- Complex control and monitoring tasks
  - Data transmissions between ECUs / to MMI
  - E.g., engine control, ext. sensors, X-by-Wire
- Simplification of wiring
  - Replaces dedicated copper wiring
  - E.g., central power locks, power windows, turn signal lights
- Multimedia bus systems
  - Transmission of large volumes of data
  - E.g., Navigation unit, Radio/CD, Internet

Classification: Off board communication

- Diagnosis
  - Readout of ca. 3000 kinds of errors
  - Garage, exhaust emission testing
- Flashing
  - Initial installation of firmware on ECUs
  - Adaptation of ECU to make, model, extras, ...
- Debugging
  - Detailed diagnosis of internal status
  - During development

Classification by use case

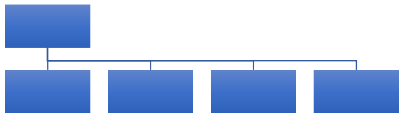
Application	Message length	Message rate	Data rate	Latency	Robustness	Cost
Control and monitoring		★★	★★	★★★	★★★	★★
Simplified Wiring				★	★★	★
Multimedia	★	★★	★★★	★	★	★★★
Diagnosis						★
Flashing	★★		★★		★	
Debugging		★	★	★★		

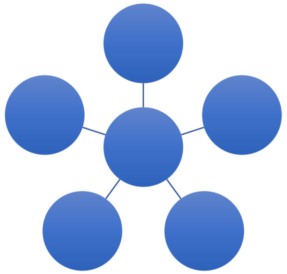
Classification by Society of Automotive Engineers (SAE)

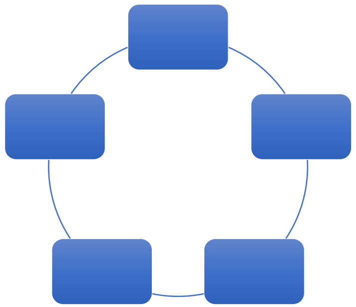
Class		
Class A		
Data rate ~ 16 kBit/s	Cheap	Diagnosis, Sensor-Actor
Class B		
Data rate ~ 64 kBit/s	Error correcting	Networking ECUs
Class C		
Data rate ~ 1 MBit/s	Real time requirements	Drive train
Class D		
Data rate ~ 10 MBit/s	Low latency	X-By-Wire, Multimedia

Network Topologies

- Line
  - ✔ Cost
  - ✔ Complexity
  - ☐ Robustness
- Star
  - ☐ Cost
  - ✔ Complexity
  - (✔) Robustness
- Ring
  - ✔ Cost
  - ☐ Complexity
  - ✔ Robustness







Network Topologies

- Coupling of bus elements
  - Repeater
    - Signal amplification
    - Signal refreshing
  - Bridge
    - Medium / timing adaptation
    - Unfiltered forwarding
  - Router
    - Filtered forwarding
  - Gateway
    - Address adaptation
    - Speed adaptation
    - Protocol adaptation

1 – Phy	
Bus 1	**Bus 2**

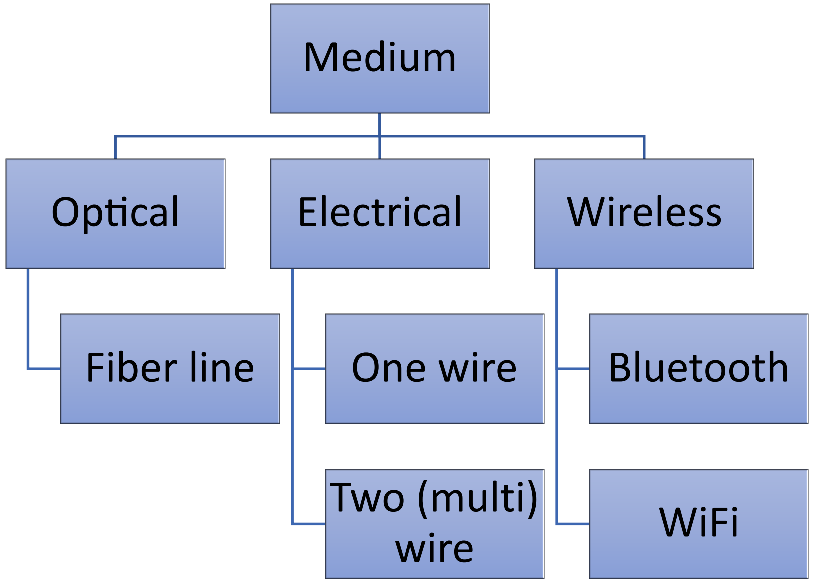
2 - Lnk	
1 - Phy	1 - Phy
Bus 1	**Bus 2**

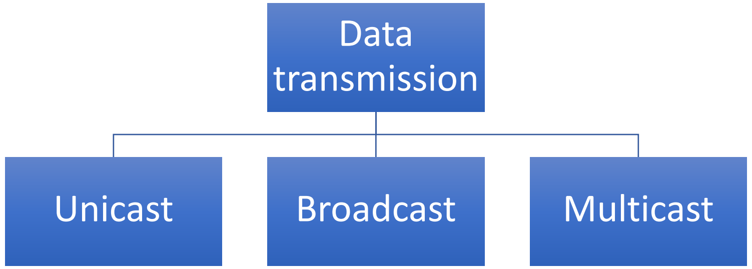
3 - Net	
2 - Lnk	2 – Lnk
1 - Phy	1 - Phy
Bus 1	**Bus 2**

7 - App	
3 - Net	3 - Net
2 - Lnk	2 – Lnk
1 - Phy	1 - Phy
Bus 1	**Bus 2**

Network Topologies

- Medium and Data transmission



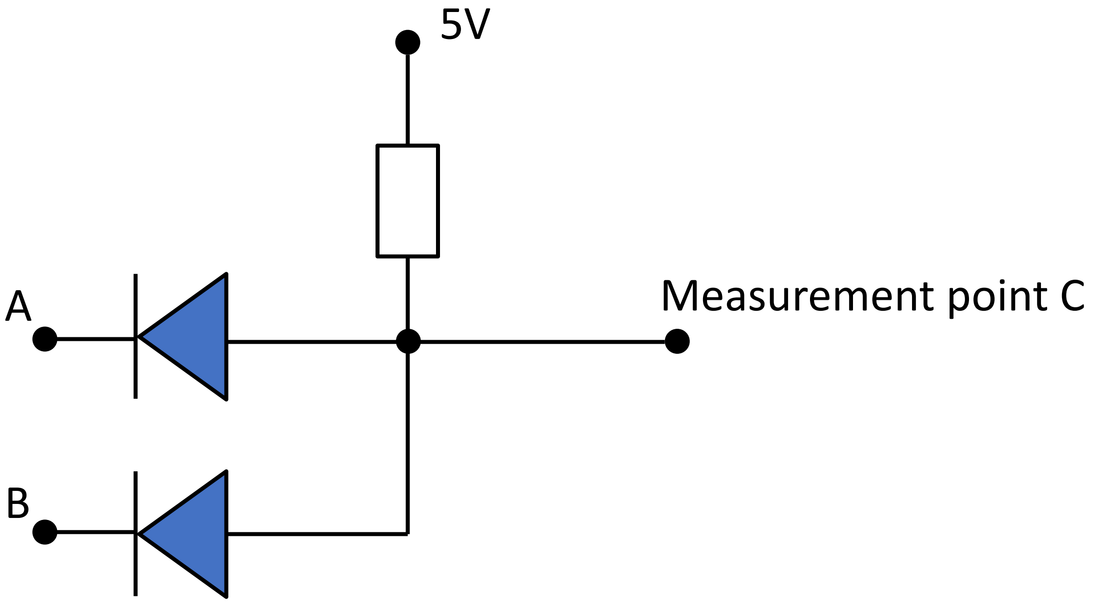


Network Topologies

- Concurrent bus access for typical wiring
  - Shared data line connected to pull-up resistors
  - Transistors can pull data line to GND (signal ground)
  - Base state
    - transistors non-conductive
    - pull up resistors raise bus level to *high*
  - One or more ECUs turn transistor conductive
    - This connects bus to signal ground
    - Bus level is *low* independent of other ECUs (⇨ dominant state)
  - Wired OR (if *low* ≙ 1) / Wired AND (if *low* ≙ 0)

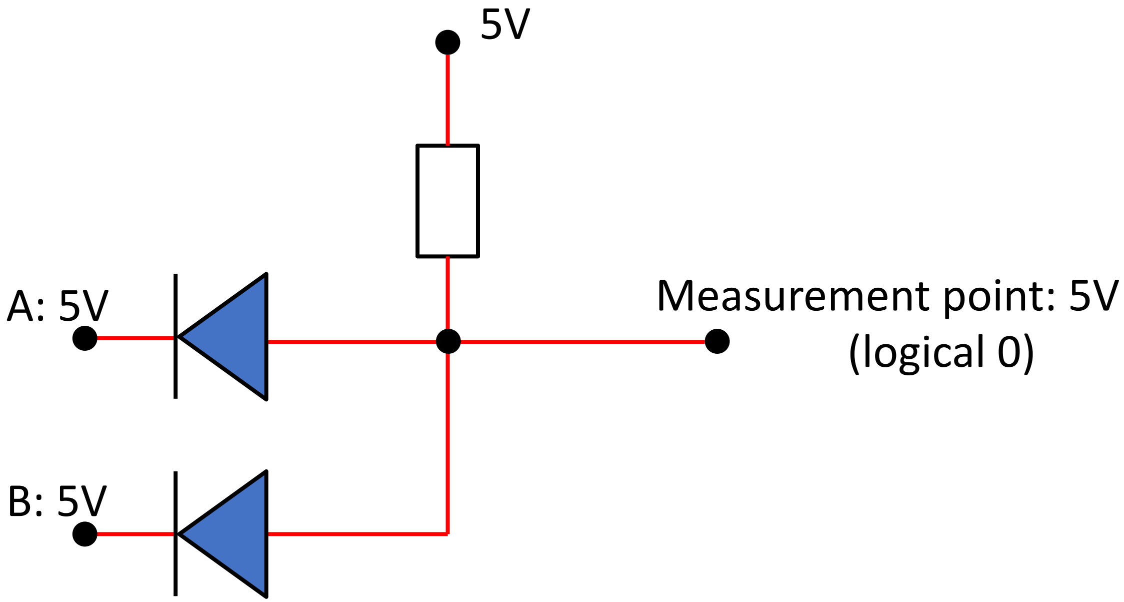
Network Topologies

- Wired OR
    - Example (assuming negative logic)
        - 5V = logical 0
        - 0V = logical 1



Network Topologies

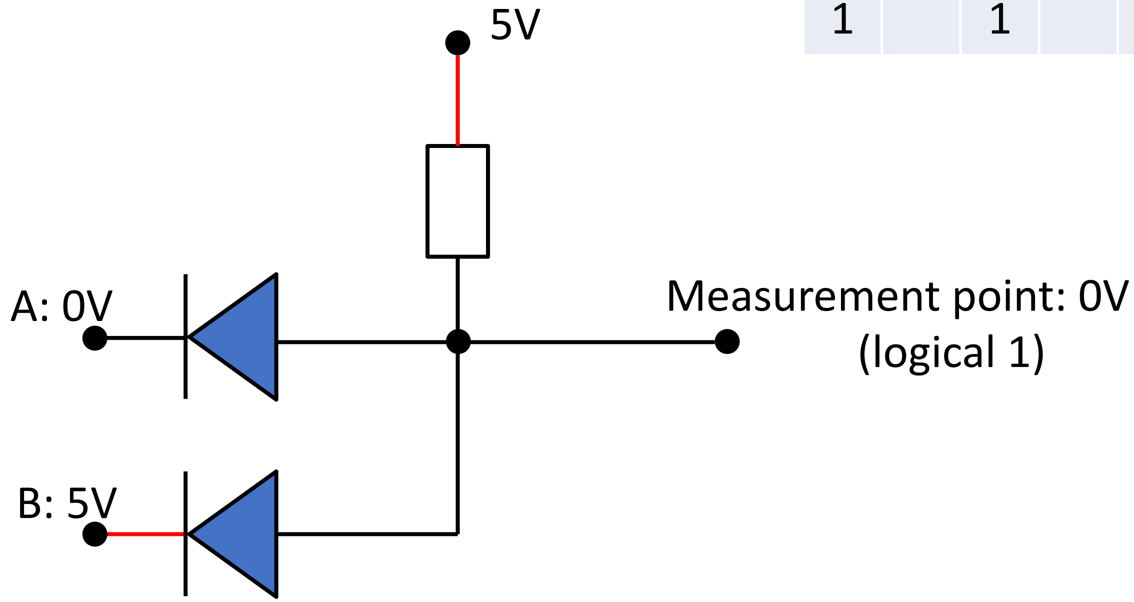
- Wired OR
  - Example (assuming negative logic)
    - 5V = logical 0
    - 0V = logical 1



Network Topologies

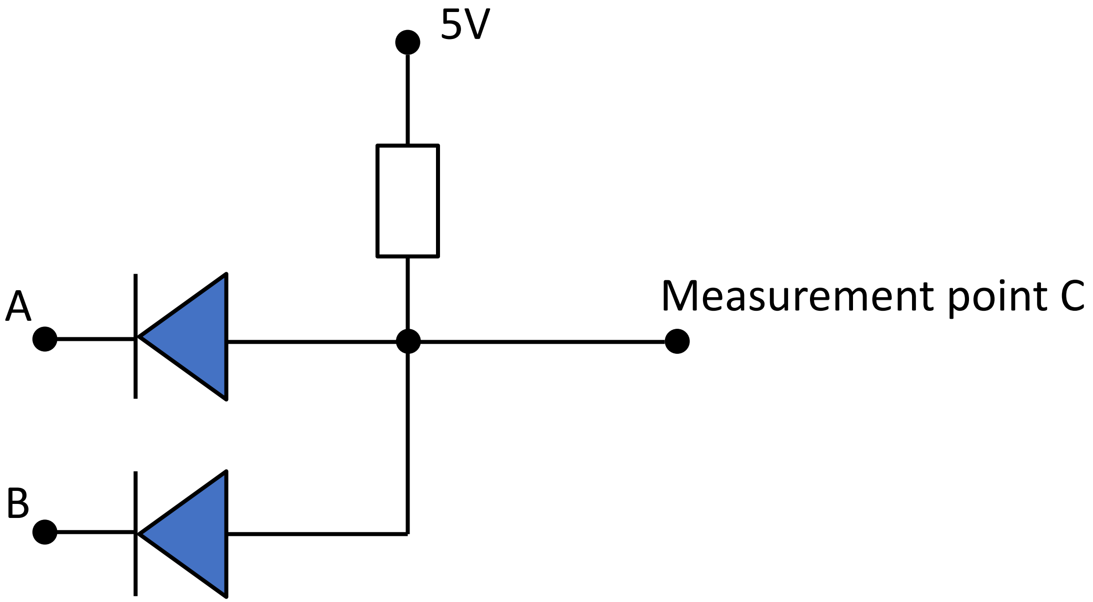
- Wired OR
  - Example (assuming negative logic)
    - 5V = logical 0
    - 0V = logical 1

A	+	B	=	C
0		0		0
0		1		1
1		0		1
1		1		1



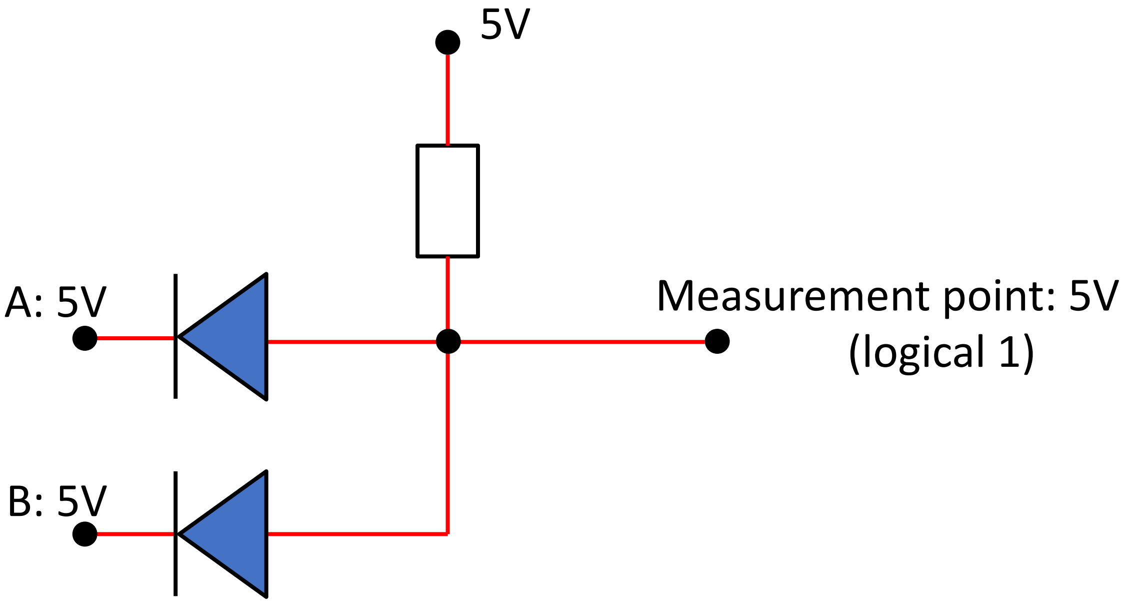
Network Topologies

- Wired AND
  - Example (assuming positive logic)
    - 5V = logical 1
    - 0V = logical 0



Network Topologies

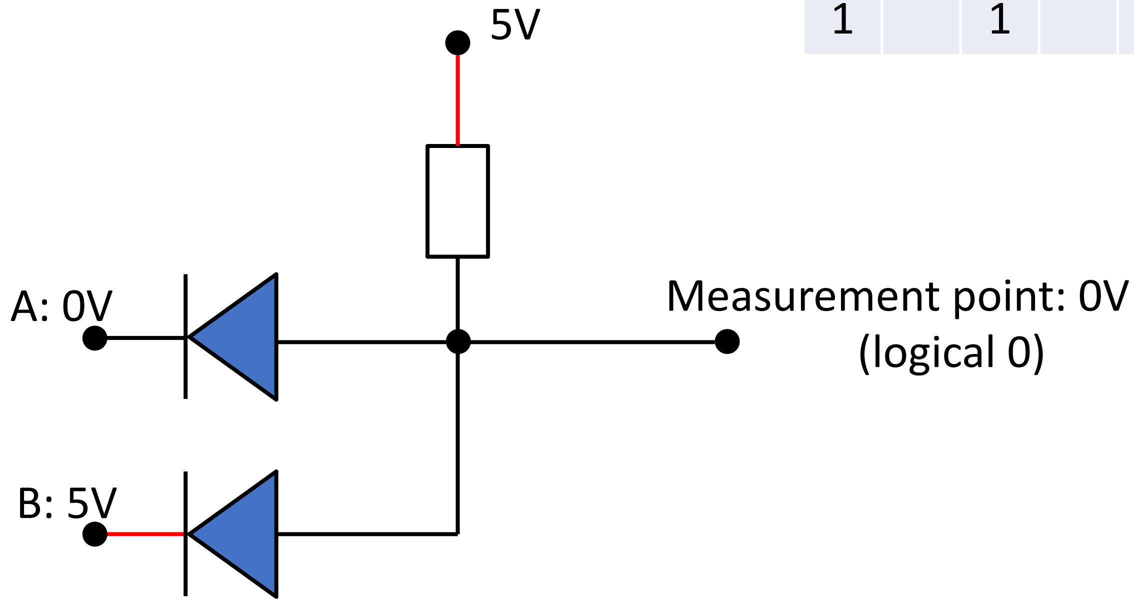
- Wired AND
  - Example (assuming positive logic)
    - 5V = logical 1
    - 0V = logical 0



Network Topologies

- Wired AND
  - Example (assuming positive logic)
    - 5V = logical 1
    - 0V = logical 0

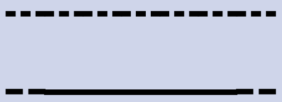
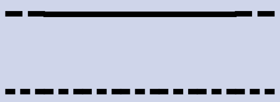
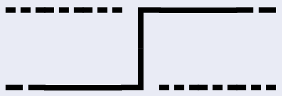
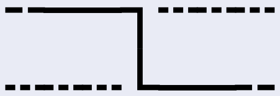
A	·	B	=	C
0		0		0
0		1		0
1		0		0
1		1		1



Network Topologies: Wave Effects

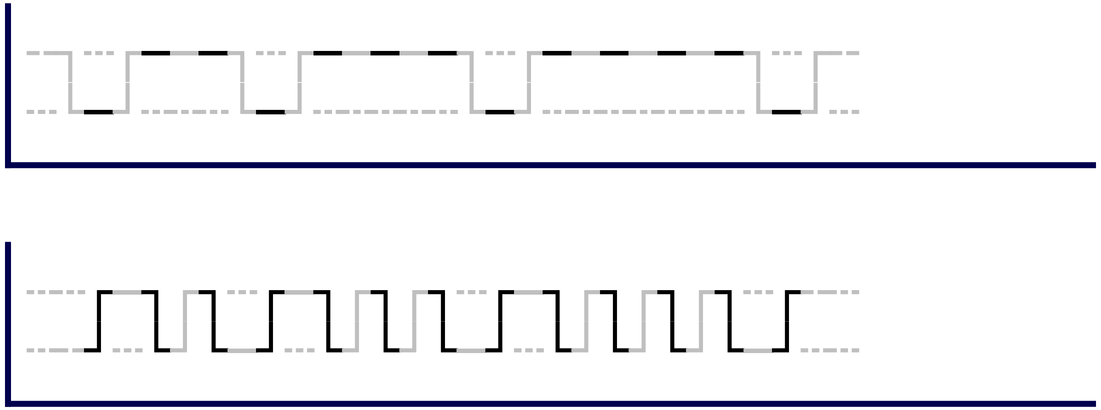
- Wave effects: Reflections and ends of wire or connectors
- Non negligible at high data rates, i.e., short bit lengths
- Propagation velocity of a signal on in-vehicle bus:
  - $c \approx \frac{1}{3} c_0$
- Signal delay on typical in-vehicle bus:
  - $t = \frac{l}{c} \approx 200\text{ns}$
- Wave effects problematic if:
  - $t_{bit} < 10t$
- Countermeasures
  - Add terminator plugs (resistor)
  - Minimize use of connectors

Bit coding

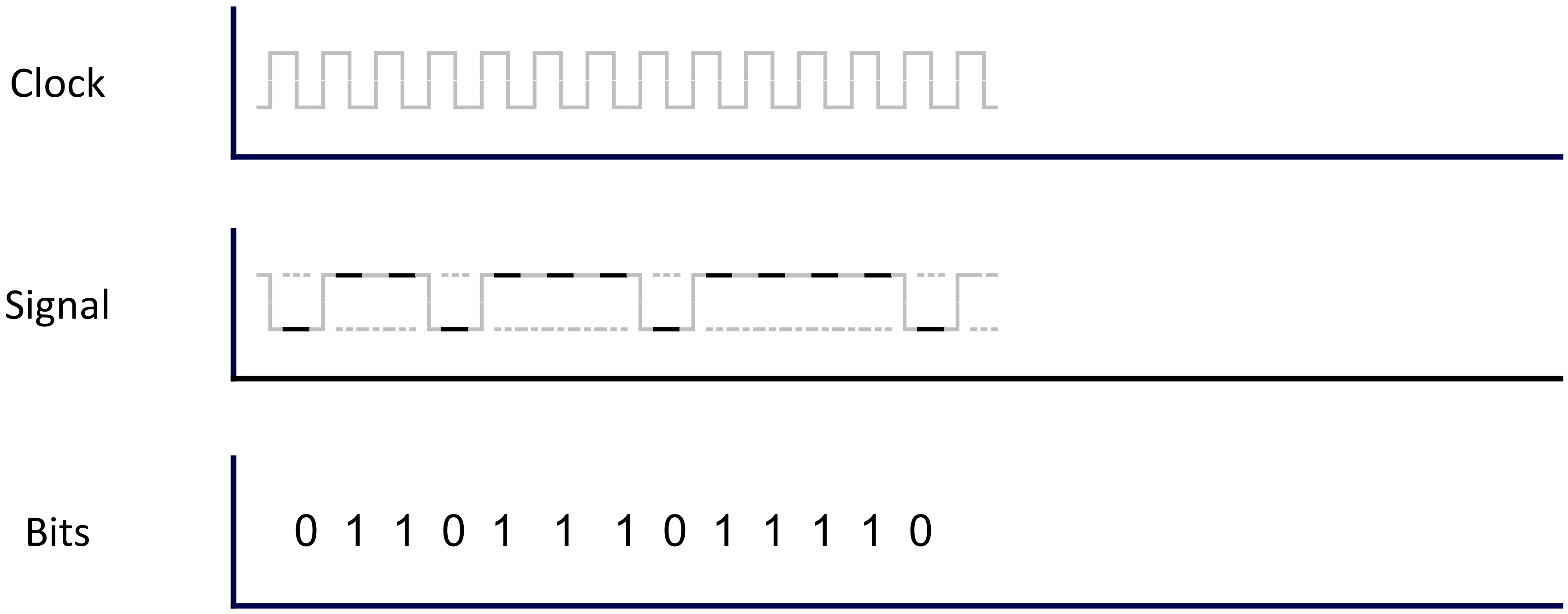
	logical 0	logical 1
Non return to Zero (NRZ)		
Manchester (original variant)		

NRZ

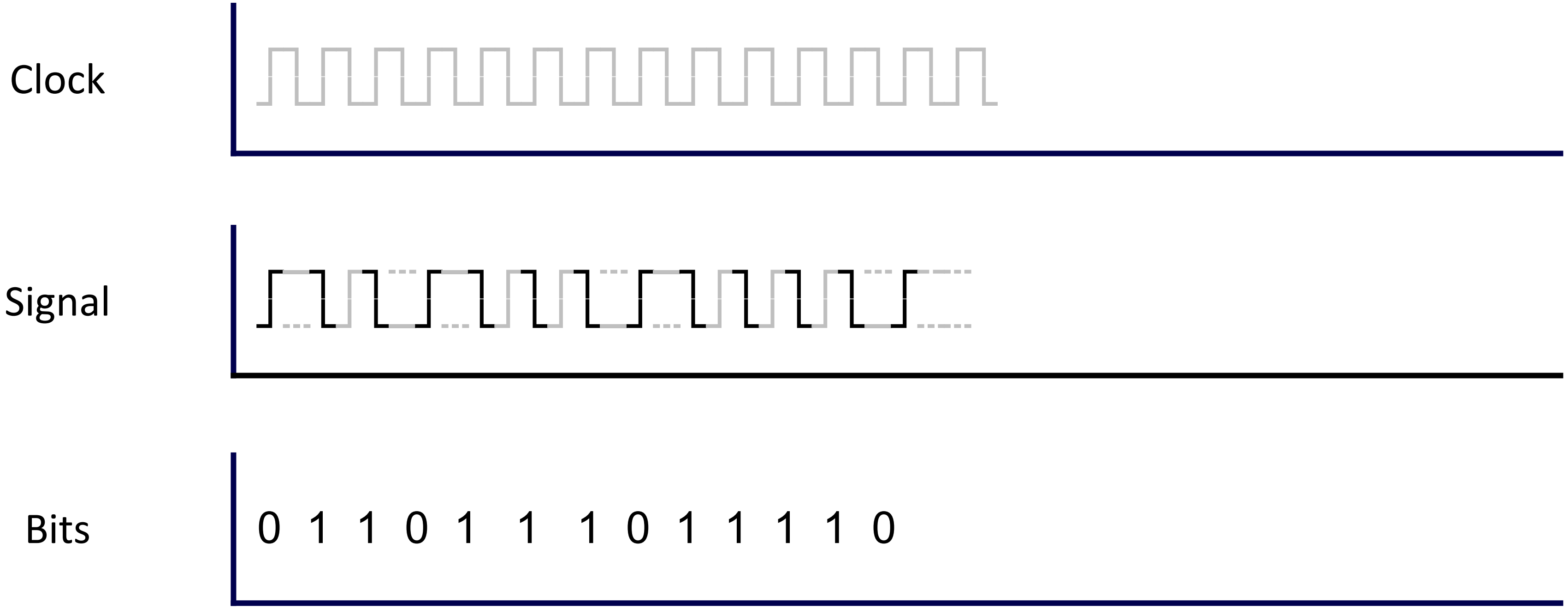
Manchester



Non Return to Zero (NRZ)

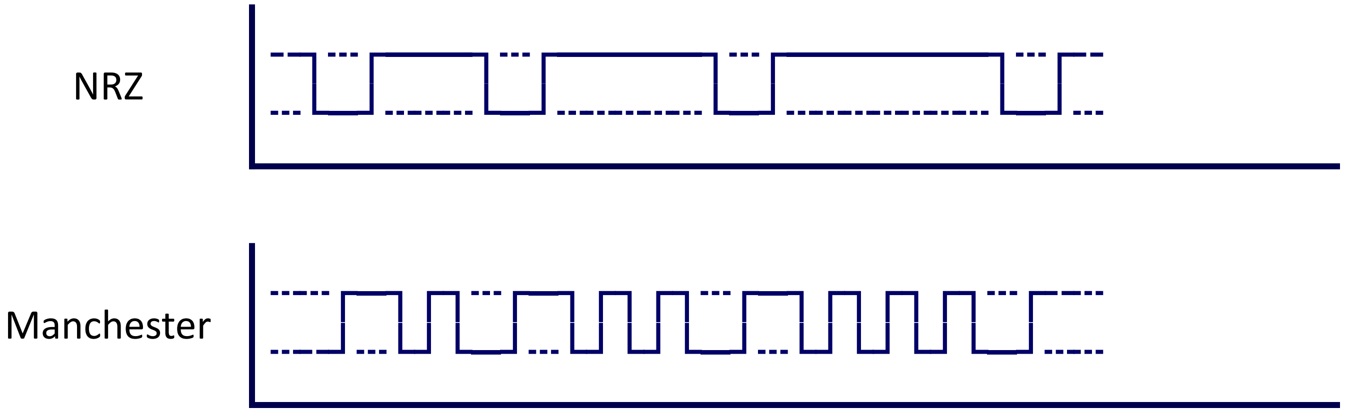


Manchester Code



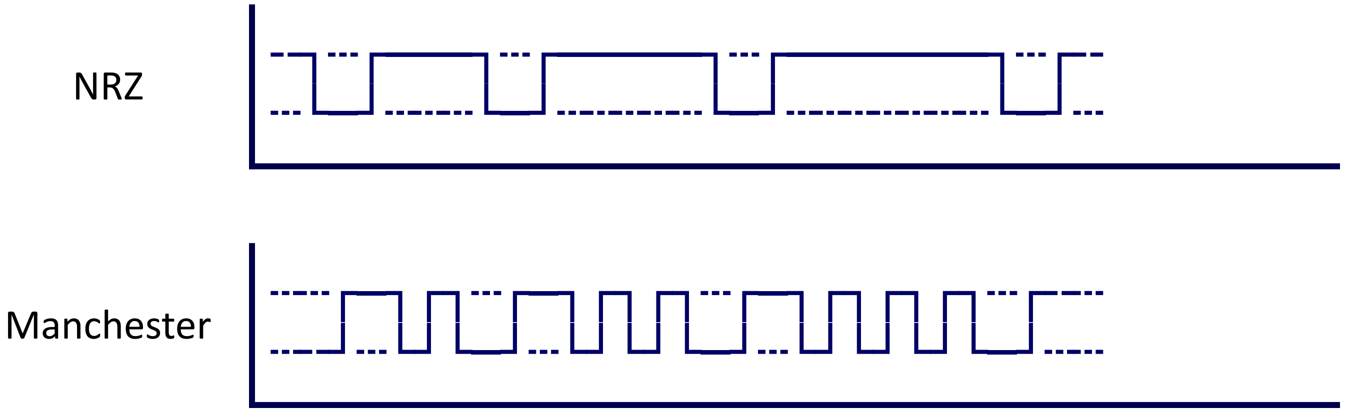
Reducing ElectroMagnetic Interference (EMI)

- Add shielding to wires
- Use twisted pair wiring
- Reduce steepness of signal slope
- Use coding with few rising/falling signal edges (NRZ)



Clock drift

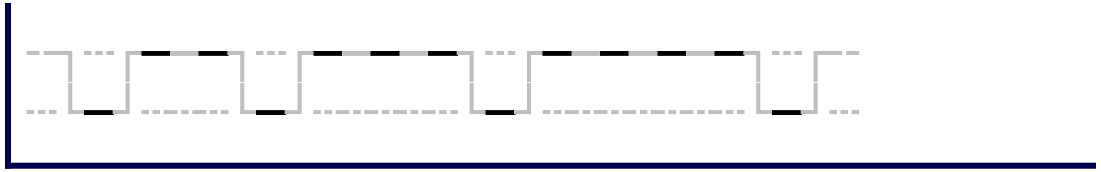
- Caused by natural variations of quartz, environment
- Receiver must sample signal at right time instant
- Clock drift leads to de-synchronization
- Bit timing has to be re-adjusted continually
- Commonly used: rising/falling signal edges



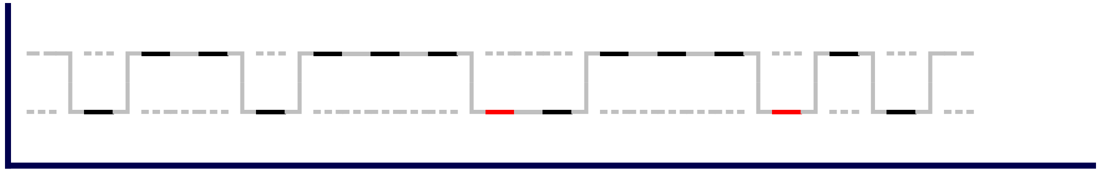
Bit stuffing

- Problem
  - When using NRZ coding, sending many identical bits leaves no signal edges that could be used to compensate for clock drift
- Solution
  - Insertion of extra bits after n consecutive identical bits
- Example (stuffing width: 3)

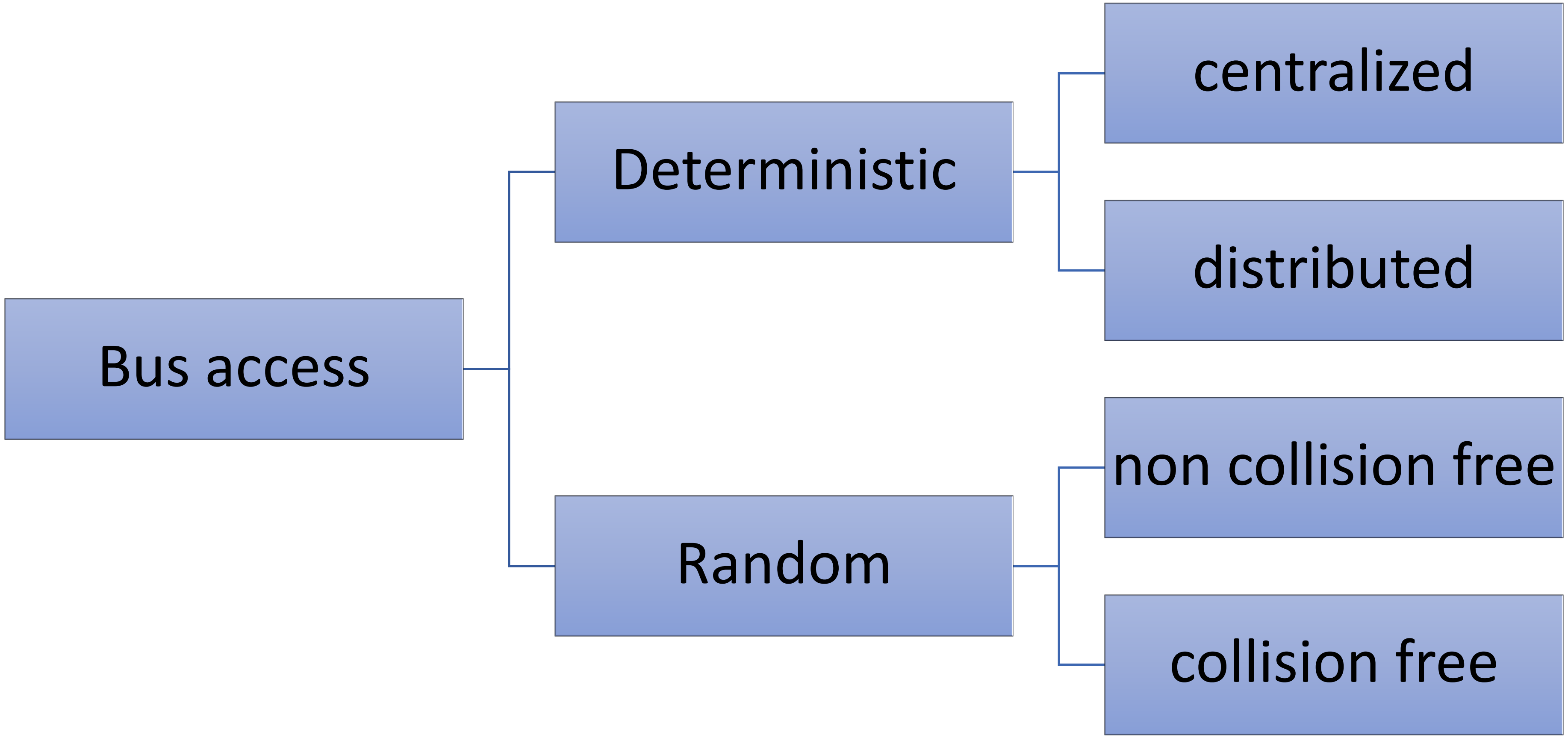
NRZ
plain



NRZ
w/ bit stuffing

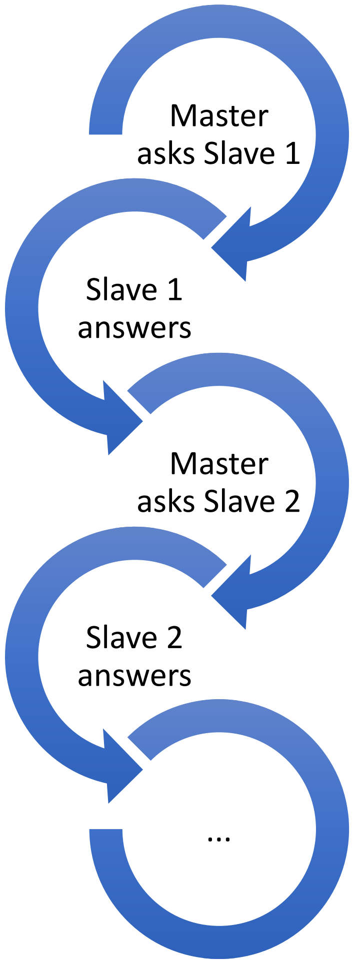


Classification according to bus access



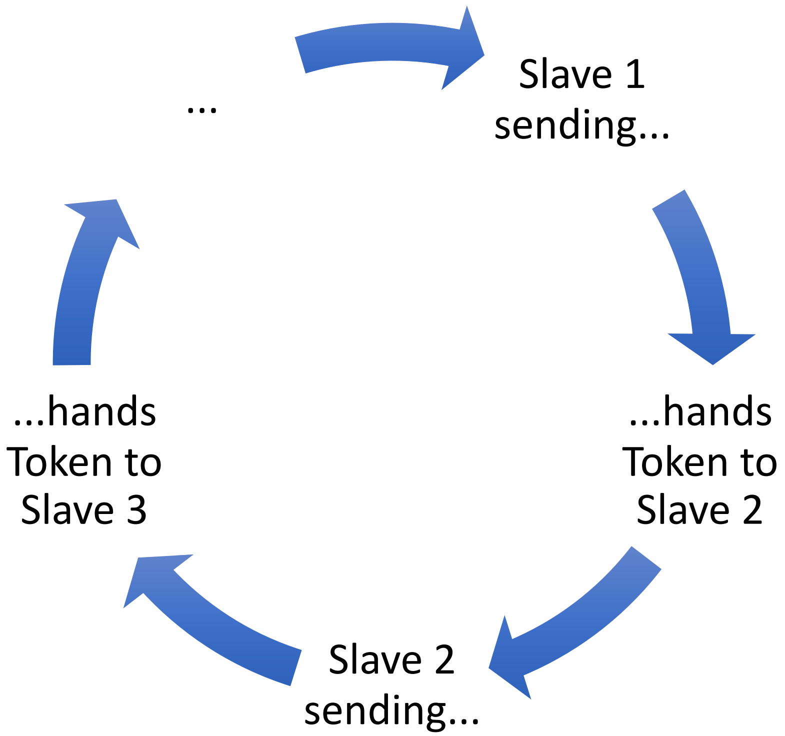
Bus Access: deterministic, centralized

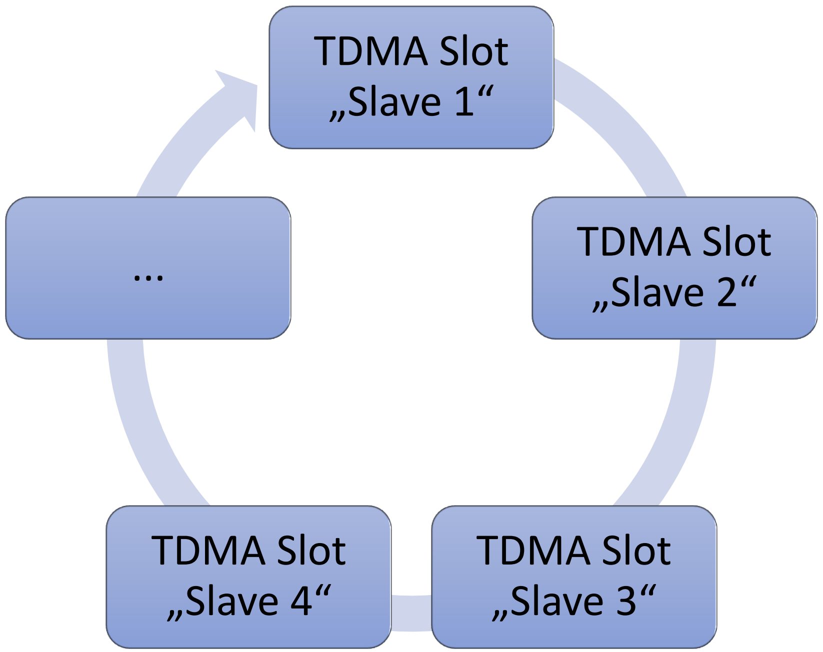
- Master-Slave protocols
- Simple request/response pattern



Bus Access: deterministic, distributed

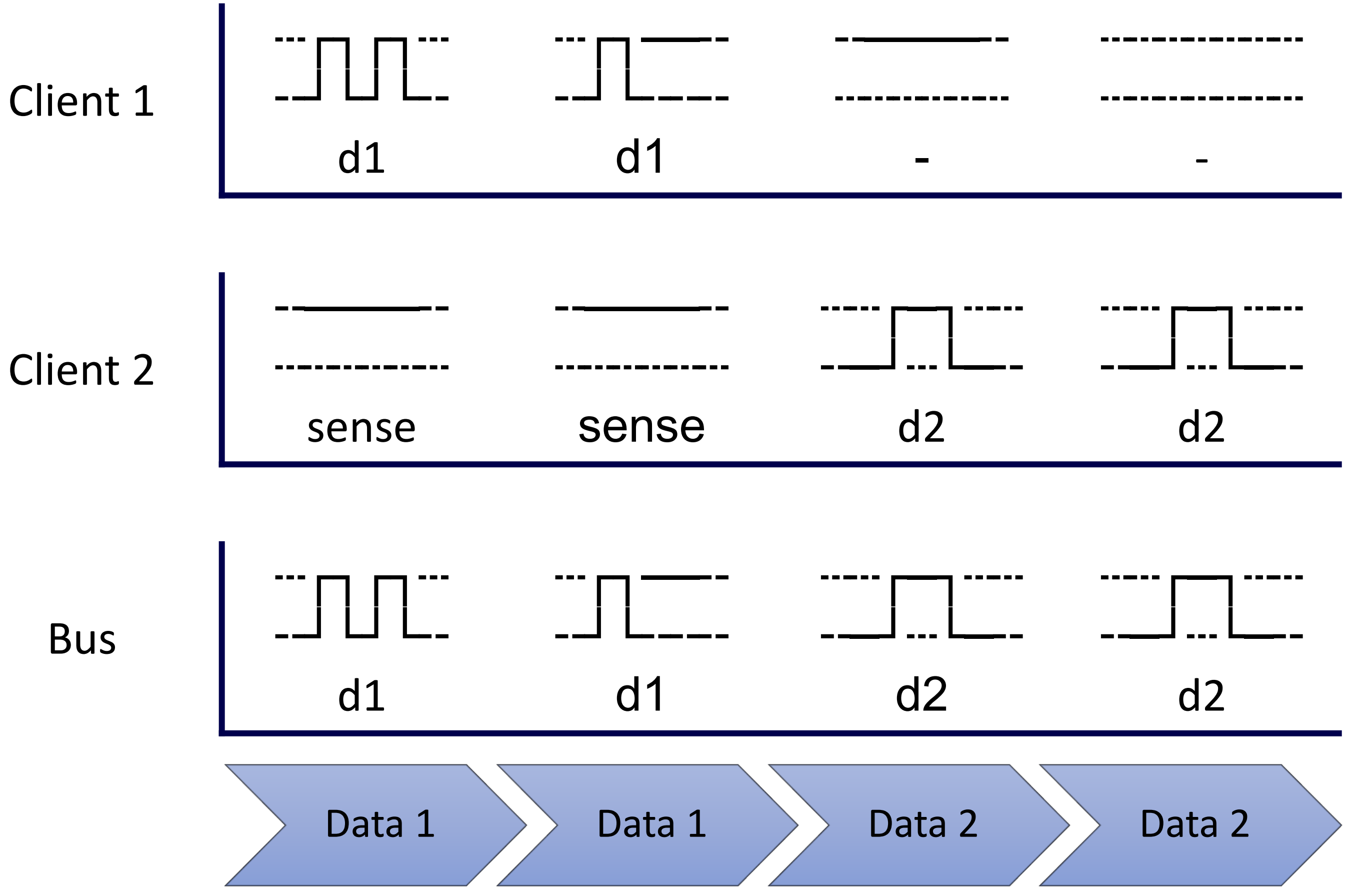
- Token based protocols, TDMA protocols





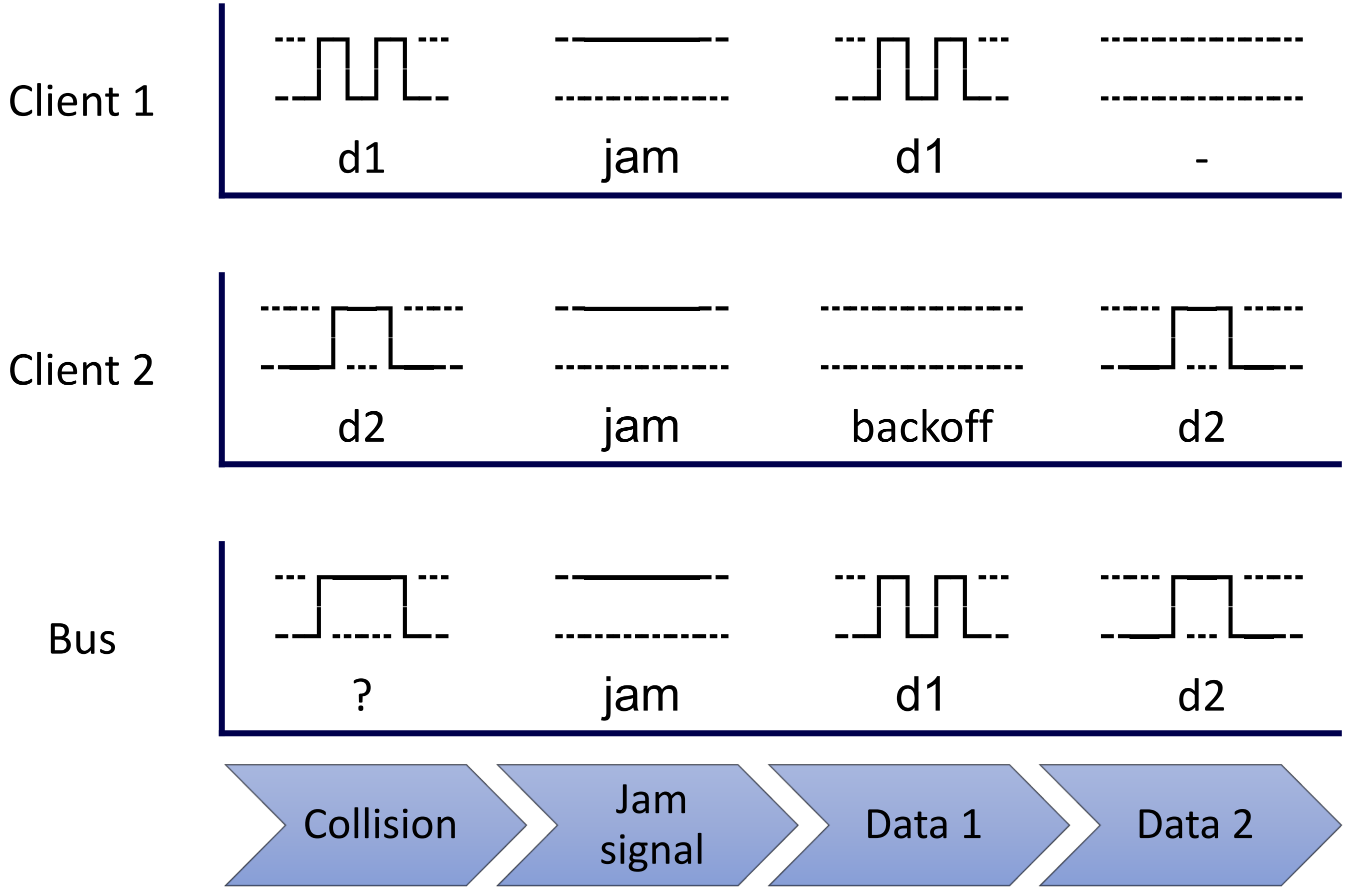
Bus Access: Random access, non collision free

- CSMA/CA (Collision Avoidance)



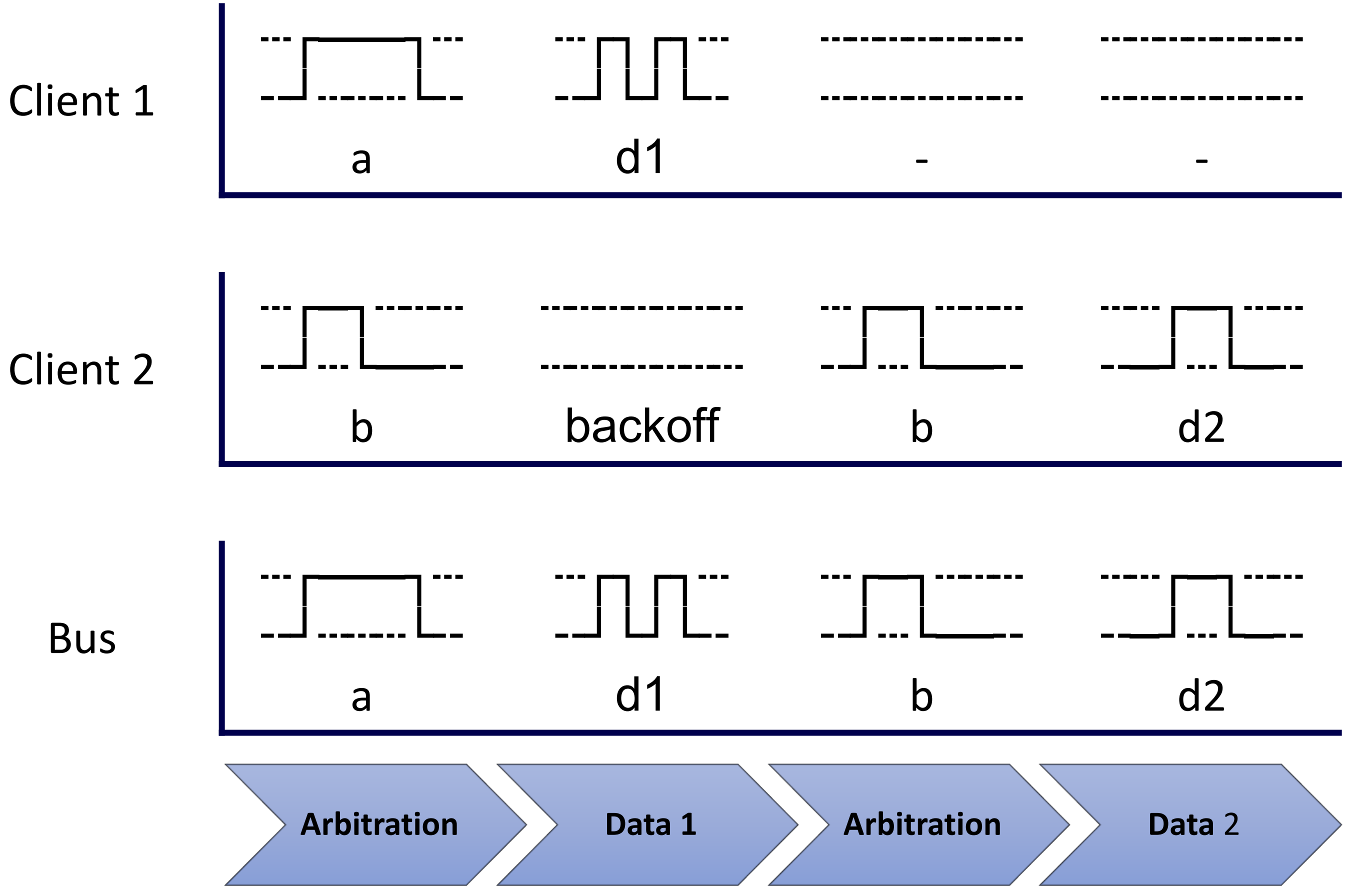
Bus Access: Random access, non collision free

- CSMA/CD (Collision Detection)



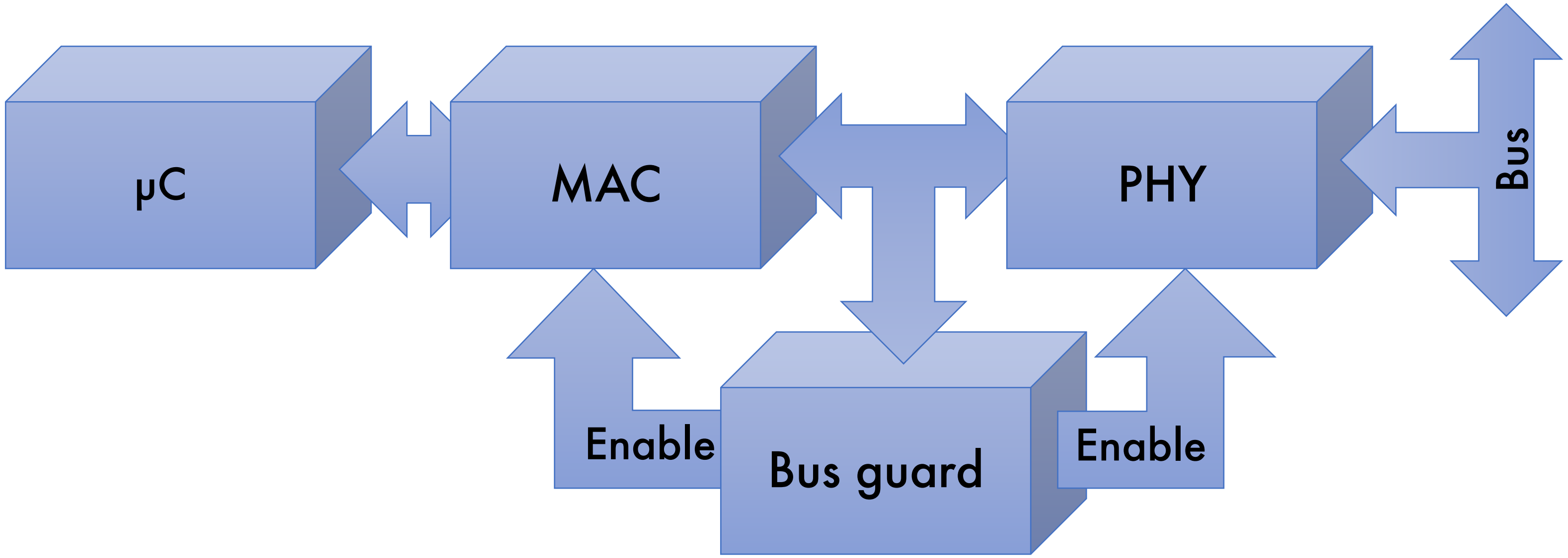
Bus Access: Random access, collision free

- CSMA/CR (Collision Resolution)



Typical structure of an ECU

- Separation by Layers
- Physical Layer: Transceiver / Bus driver
- Bus access: Communication controller
- Application layer: Microprocessor
- Commonly with bus guard for *emergency shutdown*



Main Takeaways

- Network Topologies
  - Single wire, two wire
  - Wired OR, wired AND
  - Non Return to Zero (NRZ) vs. Manchester coding
  - Clock drift, synchronization, bit stuffing

- Bus access
  - Deterministic, non-deterministic access
  - CSMA/CA, CSMA/CD, CSMA/CR
  - Bus guard

Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

The K-Line Bus

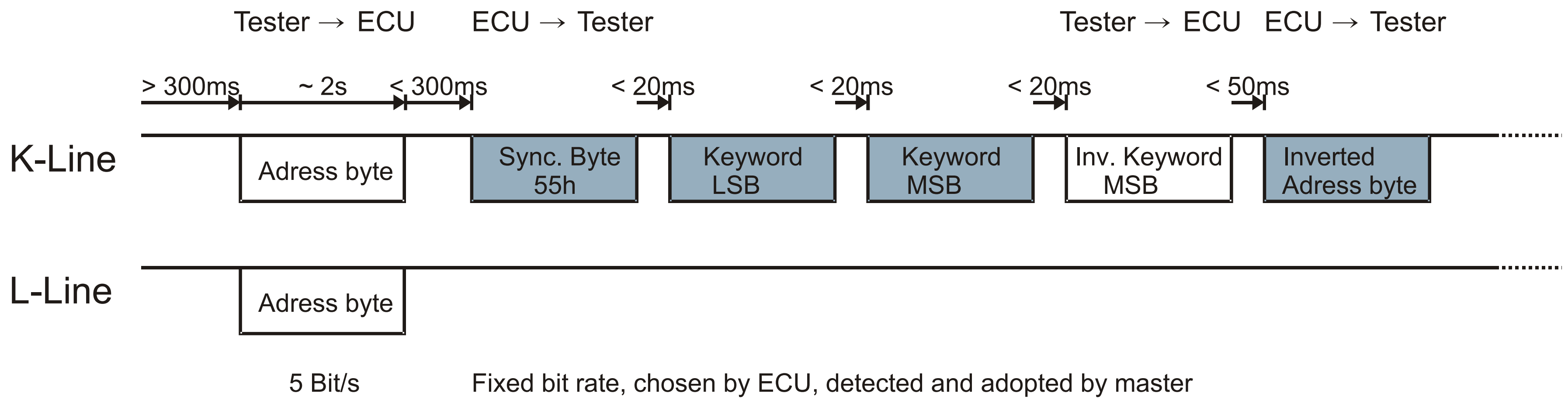
- Industry standard of the 80s, much later standardized as ISO 9141
- Numerous variants exist (esp. upwards of Link Layer)
- Focus on ISO 14230: the KWP 2000 (Keyword Protocol)
- Specifies Physical and Link layers
- Bidirectional bus, communicating over 1 wire (the **K Line**)

The K-Line Bus

- Optional: additional unidirectional **L Line**
  - Allows mixed networks (using only K Line / using both K+L Line)
- Mostly used for connecting ECU ⇔ Tester, seldom ECU ⇔ ECU
- Logic levels are relative to on board voltage (< 20% and > 80%)
- Bit transmission compatible to UART (Universal Asynchronous Receiver Transmitter): 1 start bit, 8 data bits, 1 stop bit, optional parity bit
- Bit rate 1.2 kBit/s ... 10.4 kBit/s
  - Dependent on ECU, not Bus
  - Master must be able to handle multiple bit rates

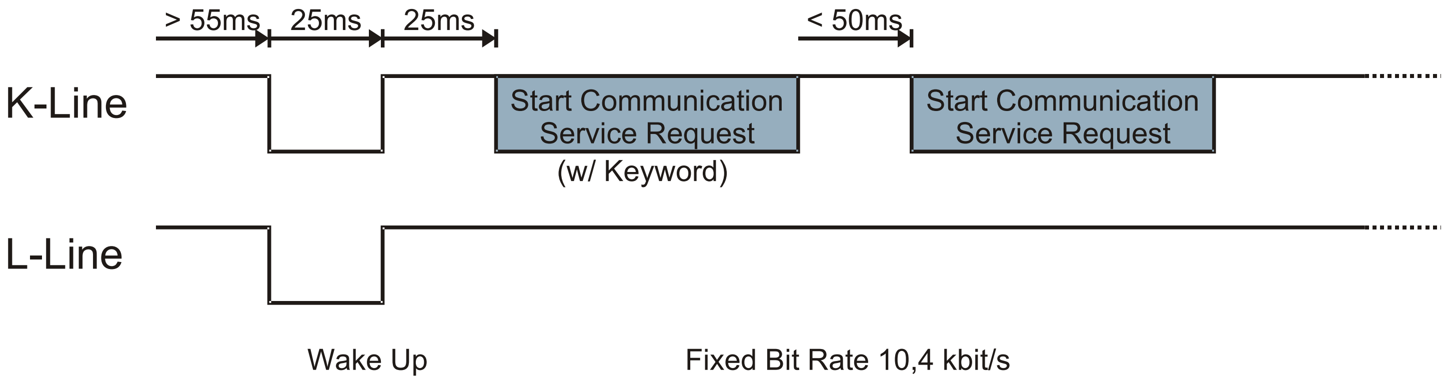
The K-Line Bus: Protocol

- Connection establishment (2 variants: 5 Baud or Fast init)
  - 5 Baud init
    - Master sends destination address (using 5 Bit/s)
    - ECU answers: 0x55 (01010101), keyword low Byte, keyword high Byte (with desired data rate)
    - Master derives bit rate from pattern, sends Echo (inv. High Byte)
    - ECU sends Echo (inv. Destination address)



The K-Line Bus: Protocol

- Connection establishment (2 variants)
  - Fast init (100 ms, Bitrate always 10,4 kBit/s)
    - Master sends *Wake Up* pattern (25 ms low, 25 ms pause)
    - Master sends *Start Communication Request,* includes dest address
    - ECU answers with keyword, after max. 50 ms
    - Keyword encodes supported protocol variants takes values from 2000 .. 2031 (KWP 2000)



The K-Line Bus: Protocol

- Communication always initiated by master
  - Master sends Request, ECU sends Response
- Addressing
  - Address length is 1 Byte
  - Either: physical addressing (identifies specific ECU)
  - Or: functional addressing (identifies class of ECU) e.g., engine, transmission, ...
  - Differentiated via format byte
- Duration of single transmission at 10.4 kBit/s
  - best case: 250 ms, worst case 5.5s
  - i.e., application layer data rate < 1 KB/s

The K-Line Bus: Protocol Header

- Format Byte
  - Encodes presence and meaning of address bytes
  - Short packet length can be encoded in format byte; length byte then omitted
- Destination address
- Source address
- Length
- Payload
  - Up to 255 Byte
  - First Byte: Service Identifier (SID)
- Checksum
  - Sum of all Bytes (mod 256)

0 .. 7	8 .. 15
Format byte	Destination
Source	Length
Payload...	
...	Checksum

The K-Line Bus: Service IDentifiers (SIDs)

- Standard Service Identifiers
  - Session Initialization and teardown
    - 0x81h Start Communication Service Request
    - 0x82h Stop Communication Service Request
  - Configuring protocol timeouts
    - 0x83h Access Timing Parameter Request (optional)

- Other SIDs are vendor defined
  - Passed on (unmodified) to application layer
  - Typical use: two SIDs per message type
    - First SID: Positive reply
    - Second: Negative reply

The K-Line Bus: Error Handling

- If erroneous signal arrives
  - ECU ignores message
  - Master detects missing acknowledgement
  - Master repeats message

- If invalid data is being sent
  - Application layer sends negative reply
  - Master / ECU can react accordingly

Use in On Board Diagnostics (OBD)

- Pin 7 of OBD connector is K-Line
- OBD uses stricter protocol variant
- Bit rate fixed to 10.4 kBit/s
- No changes in timing
- Header no longer variable
  - Length byte never included
  - Address always included
- Max. Message length is 7 Byte
- Shall use
  - logical addressing by tester,
  - physical addressing by ECUs

Main Takeaways

- K-Line
  - Mainly for diagnostics
  - Transmission uses UART signaling
  - Communication using Request-Response pattern

Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN: Controller Area Network
  - LIN
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

The CAN Bus

- “Controller Area Network”
- 1986
- Network topology: Bus
- Many (many) physical layers

- Common:
  - Up to 110 nodes
  - At 125 kBit/s: max. 500m

- Always: Two signal levels
  - low (dominant)
  - high (recessive)

The CAN Bus

- In the following: ISO 11898
  - Low Speed CAN (up to 125 kBit/s)
  - High Speed CAN (up to 1 MBit/s)
- Specifies OSI layers 1 and 2
  - Higher layers not standardized by CAN, covered by additional standards and conventions
  - e.g., CANopen
- Random access, collision free
  - CSMA/CR with Bus arbitration
  - (sometimes called CSMA/BA – bitwise arbitration)
- Message oriented
- Does not use destination addresses
  - Implicit Broadcast/Multicast

Physical layer (typical)

- High Speed CAN
  - 500 kBit/s
  - Twisted pair wiring
  - Branch lines max. 30 cm
  - Terminating resistor mandated (120 Ω)
  - Signal swing 2 V
  - Error detection must happen within one Bit's time

  ⇨ bus length is limited to $l \leq 50\mathrm{m} \times \dfrac{1\ \mathrm{MBit/s}}{\text{data rate}}$

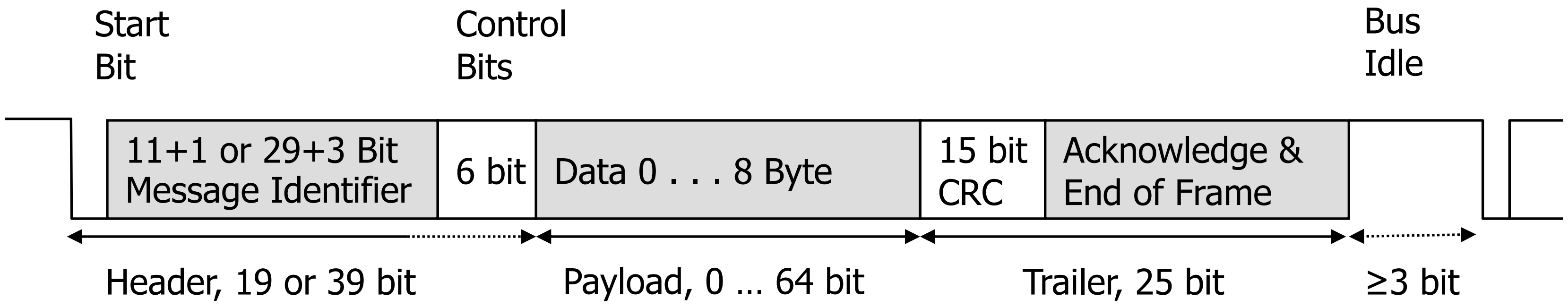
Physical layer (typical)

- Low Speed CAN
  - Up to 125 kBit/s
  - Standard two wire line suffices
  - No restriction on branch lines
  - Terminating resistors optional
  - Signal swing 5 V

- Single Wire CAN
  - 83 kBit/s
  - One line vs. ground
  - Signal swing 5 V

CAN in Vehicular Networks

- Address-less communication
  - Messages carry 11 Bit (CAN 2.0A) or 29 Bit (CAN 2.0B) message identifier
  - Stations do not have an address, frames do not contain one
  - Stations use message identifier to decide whether a message is meant for them
  - Medium access using CSMA/CR with bitwise arbitration
  - Link layer uses 4 frame formats
    Data, Remote (request), Error, Overload (flow control)
  - Data frame format:



CAN in Vehicular Networks

- CSMA/CR with bitwise arbitration
  - Avoids collisions by priority-controlled bus access
  - Each message contains identifier corresponding to its priority
  - Identifier encodes “0” dominant and “1” recessive: concurrent transmission of “0” and “1” results in a “0”
  - Bit stuffing: after 5 identical Bits one inverted Stuff-Bit is inserted (ignored by receiver)
  - When no station is sending the bus reads “1” (recessive state)
  - Synchronization happens on bit level, by detecting start bit of sending station

CAN in Vehicular Networks

- CSMA/CA with bitwise arbitration
  - Wait for end of current transmission
    - wait for 6 consecutive recessive Bits
  - Send identifier (while listening to bus)
  - Watch for mismatch between transmitted/detected signal level
    - Means that a collision with a higher priority message has occurred
    - Back off from bus access, retry later

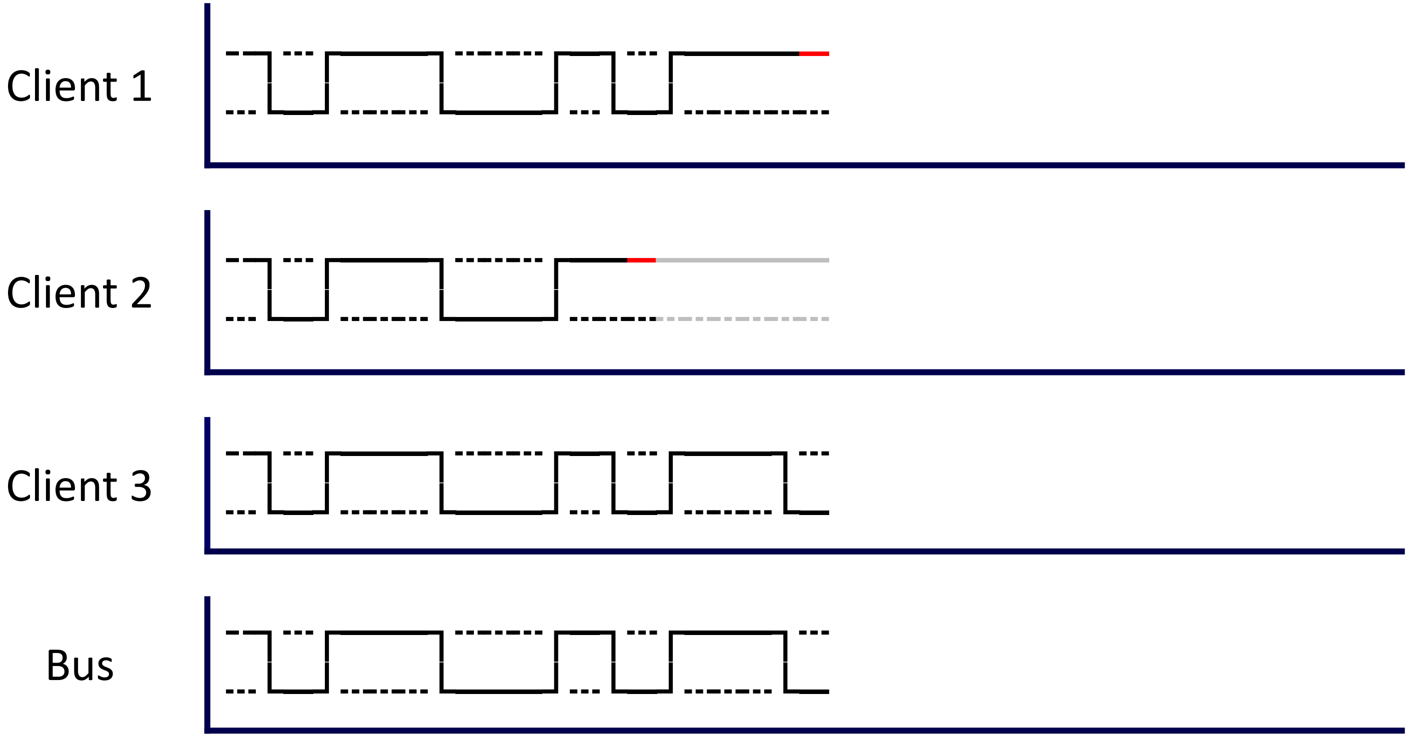
  - Realization of non-preemptive priority scheme
  - Real time guarantees for message with highest priority
    - i.e., message with longest “0”-prefix

The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 2 recognizes bus level mismatch, backs off from access

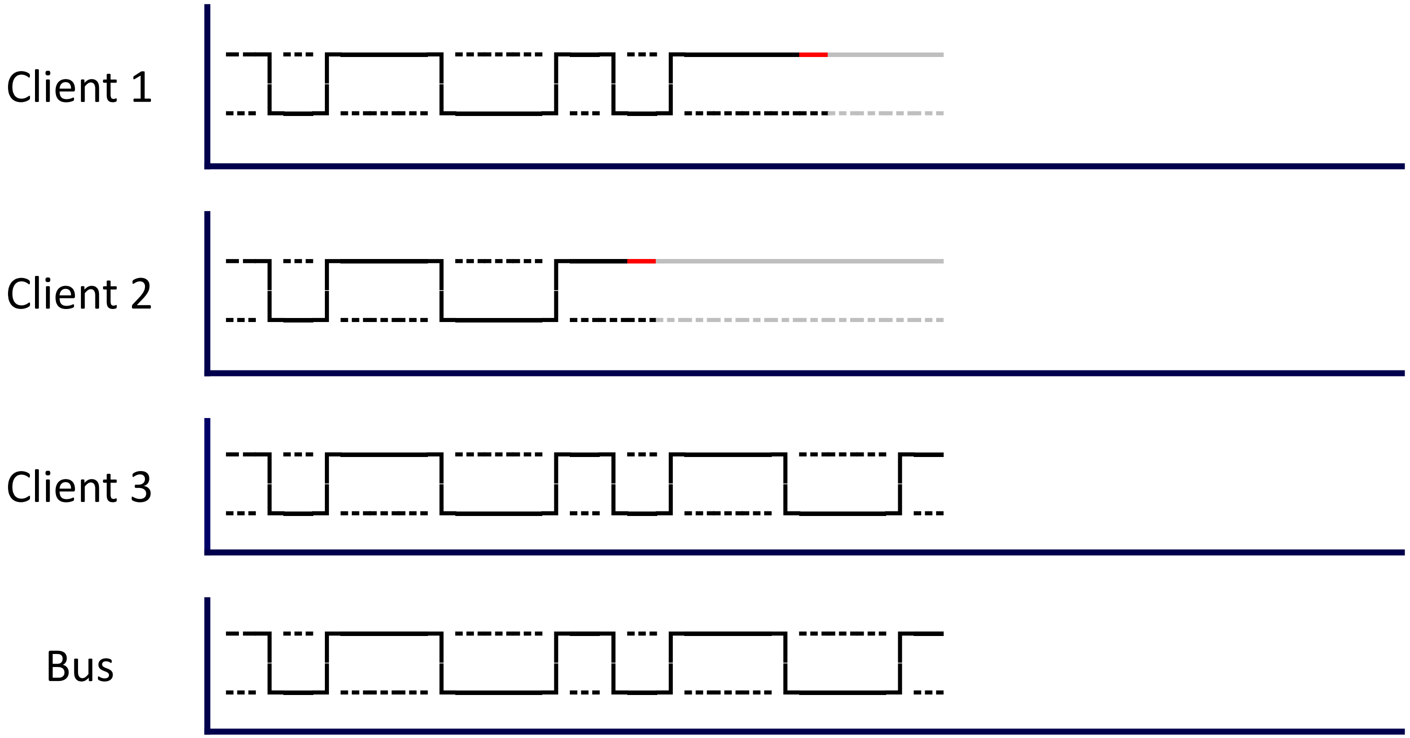
The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 1 recognizes bus level mismatch, backs off from access



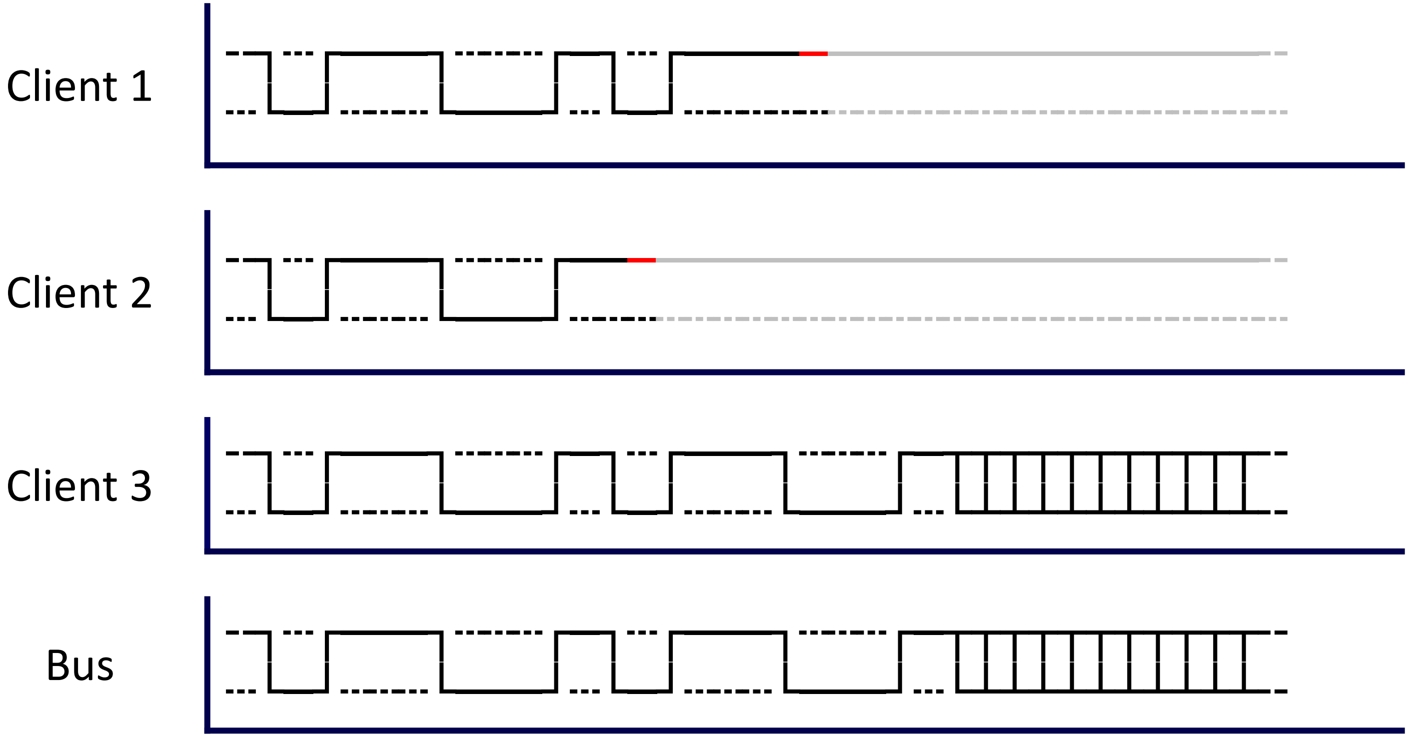
The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 3 wins arbitration



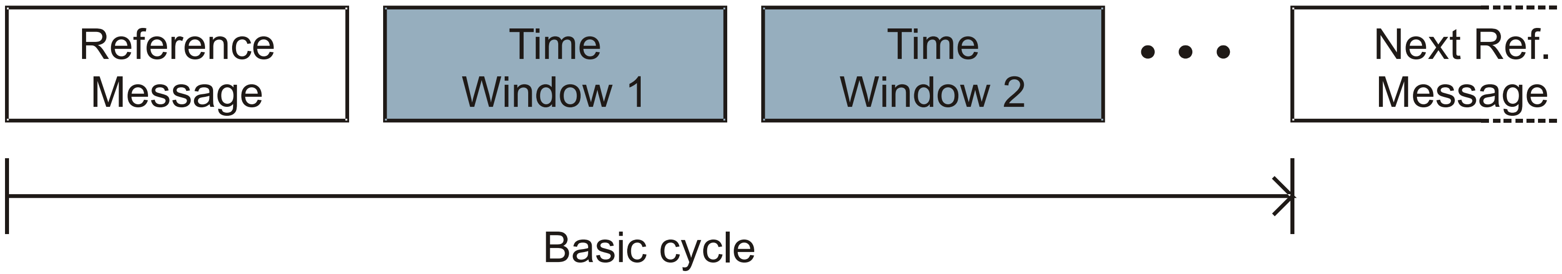
The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 3 starts transmitting data



The CAN Bus: Time-Triggered CAN (TTCAN)

- ISO 11898-4 extends CAN by TDMA functionality
- Solves non-determinism of regular CAN
  - Improves on mere “smart” way of choosing message priorities
- One node is dedicated “time master” node
- Periodically sends reference messages starting “basic cycles”
- Even if time master fails, TTCAN keeps working
  - Up to 7 fallback nodes
  - Nodes compete for transmission of reference messages
  - Chosen by arbitration



The CAN Bus: TTCAN Basic Cycle

- Basic cycle consists of time slots
  - Exclusive time slot
    - Reserved for dedicated client
  - Arbitration time slot
    - Regular CAN CSMA/CR with bus arbitration
- Structure of a basic cycle arbitrary, but static
- CAN protocol used unmodified

  ➔ Throughput unchanged

- TTCAN cannot be seen replacing CAN for real time applications
  - Instead, new protocols are being used altogether (e.g., FlexRay)

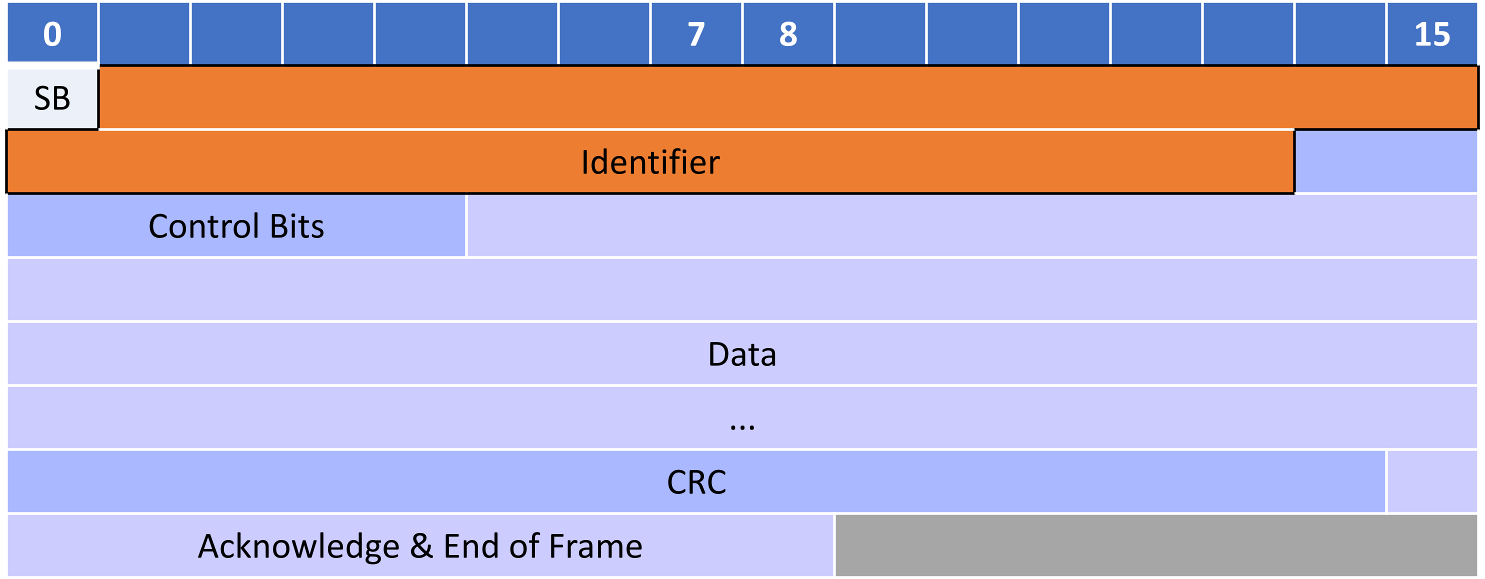
The CAN Bus: Message Filtering

- Message filtering
  - Acceptance of messages determined by message identifier
  - Uses two registers
    - Acceptance Code (bit pattern to filter on)
    - Acceptance Mask ("1" marks relevant bits in acceptance code)

Bit	10	9	8	7	6	5	4	3	2	1	0
Acceptance Code Reg.	0	1	1	0	1	1	1	0	0	0	0
Acceptance Mask Reg.	1	1	1	1	1	1	1	0	0	0	0
Resulting Filter Pattern	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**X**	**X**	**X**	**X**

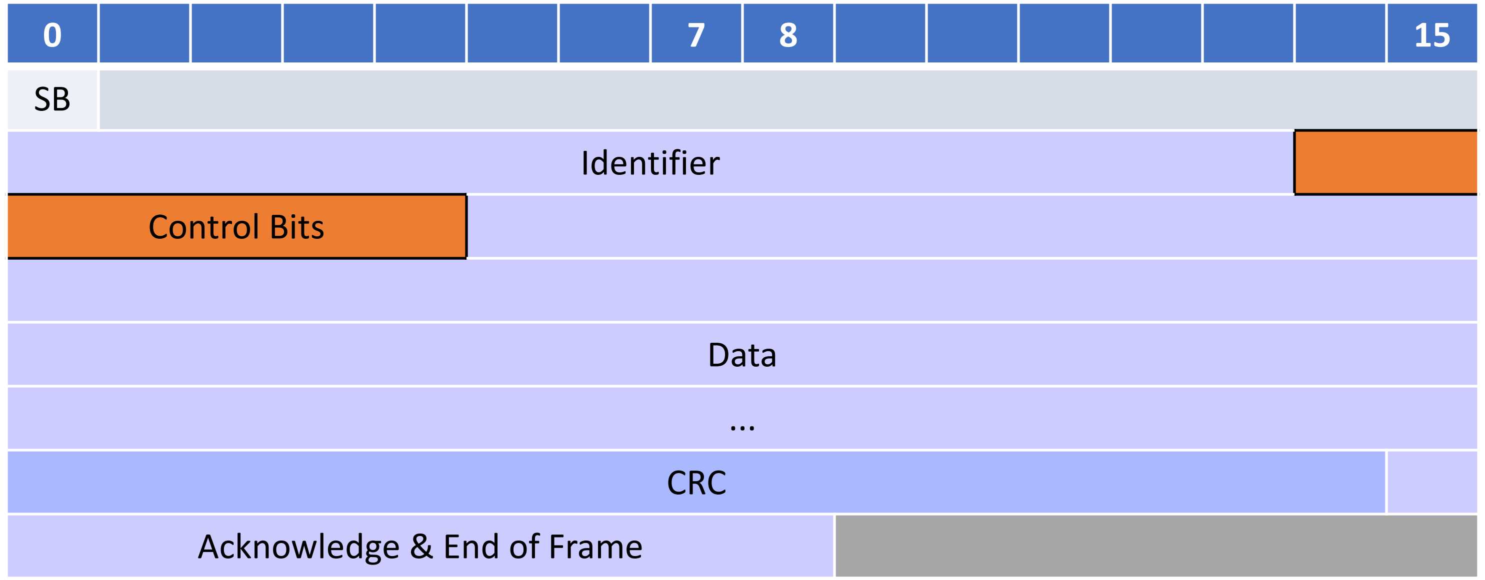
The CAN Bus: Data Format

- NRZ
- Time synchronization using start bit and stuff bits (stuff width 5)
- Frame begins with start bit
- Message identifier 11 Bit (CAN 2.0A), now 29 Bit (CAN 2.0B)



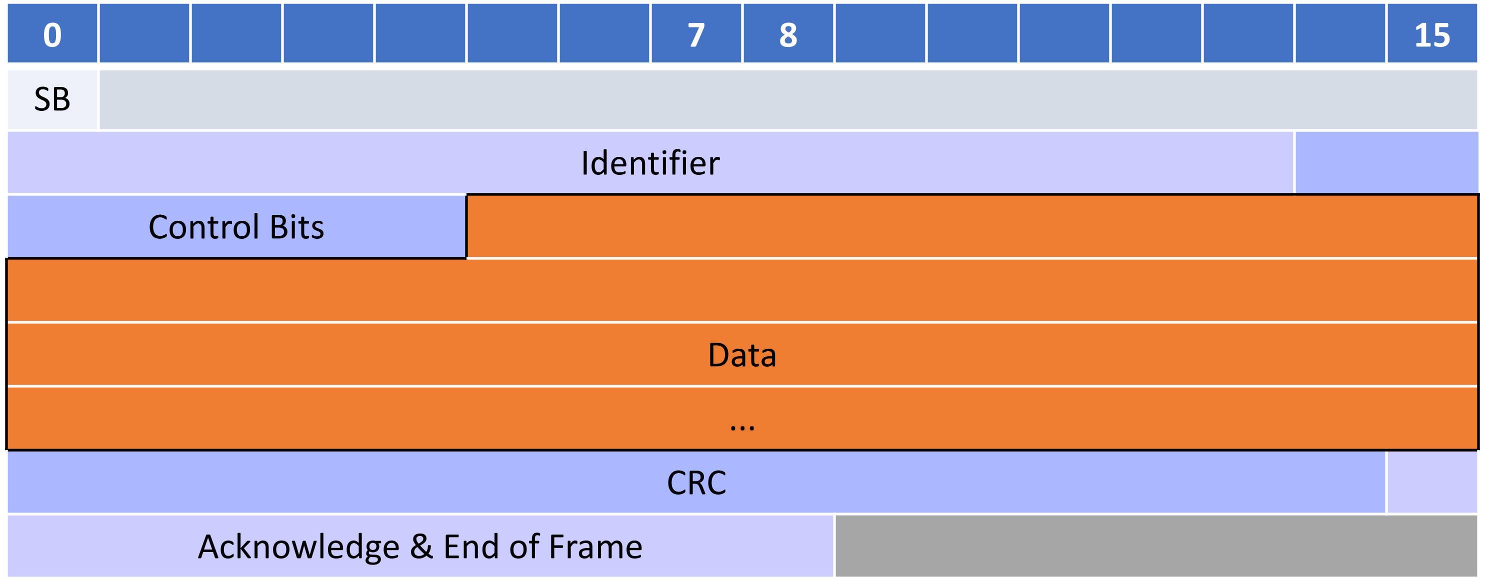
The CAN Bus: Data Format

- Control Bits
  - Message type (Request, Data, Error, Overload)
  - Message length
  - …



The CAN Bus: Data Format

- Payload
  - Restriction to max. 8 Byte per message
  - Transmission time at 500 kBit/s: 260 µs (using 29 Bit ID)
  - i.e., usable data rate 30 kBit/s



The CAN Bus

- Error detection (low level)
  - Sender checks for unexpected signal levels on bus
  - All nodes monitor messages on the bus
    - All nodes check protocol conformance of messages
    - All nodes check bit stuffing
  - Receiver checks CRC

  - If any(!) node detects error it transmits error signal
    - 6 dominant Bits with no stuffing

  - All nodes detect error signal, discard message

The CAN Bus

- Error detection (high level)
  - Sender checks for acknowledgement
    - Receiver transmits dominant “0” during ACK field of received message
  - Automatic repeat of failed transmissions
  - If controller finds itself causing too many errors
    - Temporarily stop any bus access
  - Remaining failure probability ca. 10^{-11}

The CAN Bus: Transport Layer

- Not covered by ISO 11898 (CAN) standards
  - Fragmentation
  - Flow control
  - Routing to other networks

- Add transport layer protocol
  - ISO-TP
    - ISO 15765-2
  - TP 2.0
    - Industry standard
  - …

The CAN Bus: ISO-TP

- ISO-TP: Header
  - Optional: 1 additional address Byte
    - Regular addressing
      - Transport protocol address completely in CAN message ID
    - Extended addressing
      - Uniqueness of addresses despite non-unique CAN message ID
      - Part of transport protocol address in CAN message ID, additional address information in first Byte of TP-Header
  - 1 to 3 PCI Bytes (Protocol Control Information)
    - First high nibble identifies one of 4 types of message
    - First low nibble and addl. Bytes are message-specific

0	1		2	3	4	5	6	7
(opt) Addl. Address	PCI high	PCI low	(opt) Addl. PCI Bytes		Payload			

The CAN Bus: ISO-TP

- ISO-TP: Message type “Single Frame”
  - 1 Byte PCI, high nibble is 0
  - low nibble gives number of Bytes in payload
  - PCI reduces frame size from 8 Bytes to 7 (or 6) Bytes, throughput falls to 87.5% (or 75%, respectively)
  - No flow control

<table>
<tr><th colspan="2">0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td>Len</td><td colspan="7">Payload</td></tr>
</table>

<table>
<tr><th>0</th><th colspan="2">1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>(Address)</td><td>0</td><td>Len</td><td colspan="6">Payload</td></tr>
</table>

The CAN Bus: ISO-TP

- ISO-TP: Message type “First Frame”
  - 2 Bytes PCI, high nibble is 1
  - low nibble + 1 Byte give number of Bytes in payload
  - After First Frame, sender waits for Flow Control Frame

0	1		2	3	4	5	6	7
(Address)	1	Len		Payload				

- ISO-TP: Message type “Consecutive Frame”
  - 1 Byte PCI, high nibble is 2
  - low nibble is sequence number SN (counts upwards from 1)
    - Application layer can detect packet loss
  - No additional error detection at transport layer

0	1		2	3	4	5	6	7
(Address)	2	SN	Payload					

The CAN Bus: ISO-TP

- ISO-TP: Message type "Flow Control Frame"
  - 3 Bytes PCI, high nibble is 3
  - low nibble specifies Flow State FS
  - FS=1: Clear to Send
    - Minimum time between two Consecutive Frames must be ST
    - Sender may continue sending up to BS Consecutive Frames, then wait for new Flow Control Frame
  - FS=2: Wait
    - Overload
    - Sender must wait for next Flow Control Frame
  - Byte 2 specifies Block Size BS
  - Byte 3 specifies Separation Time ST

0	1		2	3
(Address)	3	FS	BS	ST

The CAN Bus: TP 2.0

- Connection-oriented
- Communication based on channels
- Specifies Setup, Configuration, Transmission, Teardown

- Addressing
  - Every ECU has unique logical address;
    additional logical addresses specify groups of ECUs
  - for broadcast und channel setup:
    logical address + offset = CAN message identifier
  - Channels use dynamic CAN message identifier

The CAN Bus: TP 2.0 Broadcast

- Repeated 5 times (motivated by potential packet loss)
- Fixed length: 7 Byte
- Byte 0:
  - logical address of destination ECU
- Byte 1: Opcode
  - 0x23: Broadcast Request
  - 0x24: Broadcast Response
- Byte 2, 3, 4:
  - Service ID (SID) and parameters
- Byte 5, 6:
  - Response: 0x0000
  - No response expected: alternates between 0x5555 / 0xAAAA

0	1	2	3	4	5	6
Dest	Opcode	SID, Parameter			0x55	0x55

The CAN Bus: TP 2.0 Channel Setup

- Byte 0:
  - logical address destination ECU
- Byte 1: Opcode
  - 0xC0: Channel Request
  - 0xD0: Positive Response
  - 0xD6 .. 0xD8: Negative Response
- Byte 2, 3: RX ID
  - Validity nibble of Byte 3 is 0 (1 if RX ID not set)
- Byte 4, 5: TX ID
  - Validity nibble of Byte 5 is 0 (1 if TX ID not set)
- Byte 6: Application Type
  - cf. TCP-Ports

0	1	2	3	4	5	6
Dest	Opcode	RX ID	V	TX ID	V	App

The CAN Bus: TP 2.0 Channel Setup

- Opcode 0xC0: Channel Request
  - TX ID: CAN msg ID requested by self
  - RX ID: marked invalid
- Opcode 0xD0: Positive Response
  - TX ID: CAN msg ID requested by self
  - RX ID: CAN msg ID of original sender
- Opcode 0xD6 .. 0xD8: Negative Response
  - Reports errors assigning channel (temporary or permanent)
  - Sender may repeat Channel Request
- After successful exchange of Channel Request/Response: dynamic CAN msg IDs now assigned to sender and receiver next message sets channel parameters

0	1	2	3	4	5	6
Dest	0xC0		1	TX ID	0	App

The CAN Bus: TP 2.0

- TP 2.0: set channel parameters
  - Byte 0: Opcode
    - 0xA0: Channel Setup Request (Parameters for channel to initiator)
    - 0xA1: Channel Setup Response (Parameter for reverse channel)
  - Byte 1: Block size
    - Number of CAN messages until sender has to wait for ACK
  - Byte 2, 3, 4, 5: Timing parameters
    - E.g., minimal time between two CAN messages
- TP 2.0: misc. channel management and teardown
  - Byte 0: Opcode
    - 0xA3: Test – will be answered by Connection Setup Response
    - 0xA4: Break – Receiver discards data since last ACK
    - 0xA5: Disconnect – Receiver responds with disconnect, too

0	1	2	3	4	5
0xA0	BS	Timing			

The CAN Bus: TP 2.0

- TP 2.0: Data transmission via channels
  - Byte 0, high nibble: Opcode
    - MSB=0 – Payload
      - /AR=0 – Sender now waiting for ACK
      - EOM=1 – Last message of a block
    - MSB=1 – ACK message only (no payload)
      - RS=1 – ready for next message (➔ flow control)
  - Byte 0, low nibble
    - Sequence number
  - Bytes 1 .. 7: Payload

Opcode Nibble			
0	0	/AR	EOM

Opcode Nibble			
1	0	RS	1

0		1	2	3	4	5	6	7
Op	SN	Payload						

Main Takeaways

- CAN
  - Still standard bus in vehicles
  - Message oriented
  - CSMA with bitwise arbitration
    - Impact on determinism
    - TTCAN (TDMA)
  - Error detection
  - Transport layer: ISO-TP vs. TP 2.0
    - Flow control, channel concept

Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN: Local Interconnect Network
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

The LIN Bus

- Local Interconnect Network (LIN)
- 1999: LIN 1.0
- 2003: LIN 2.0
  - Numerous extensions
  - Backwards compatible (only)
- Goal of LIN: be much cheaper than low speed CAN
  - Only reached partially
- Specifies PHY and MAC Layer, API

The LIN Bus

- Very similar to K-Line Bus
- *Master-slave* concept with self synchronization
  - no quartz needed
  - lax timing constraints
- LIN master commonly also part of a CAN bus
  - LIN commonly called a sub bus
- Bidirectional one-wire line, up to 20 kBit/s
- Bit transmission UART compatible
  - 1 Start Bit, 8 Data Bits, 1 Stop Bit
- Message-oriented
  - No destination address

The LIN Bus

- Rudimentary error detection
    - Sender monitors bus
    - Aborts transmission on unexpected bus state
- No error correction
- Starting with LIN 2.0: Response Error Bit
    - Should be contained in periodic messages
    - Set (once) if slave detected an error in last cycle
- Static slot schedule in the master
    - “Schedule Table”
    - Determines cyclic schedule of messages transmitted by master
      → Bus timing mostly deterministic
    - Slaves do not need to know schedule
      → can be changed at run-time

The LIN Bus

- Data request (sent by master)
  - Sync Break (≥13 Low Bits, 1 High Bit)
    - Not UART compliant → uniquely identifiable
  - Sync Byte 0x55 (01010101)
    - Synchronizes bit timing of slave
  - LIN Identifier (6 data Bits (I0 to I5) + 2 parity Bits)
    - Encodes response's expected message type and length
    - 0x00 .. 0x3B: application defined data types, 0x3C .. 0x3D: Diagnosis, 0x3E: application defined, 0x3F: reserved
    - Parity Bits: $I0 \oplus I1 \oplus I2 \oplus I4$ and $\neg (I1 \oplus I3 \oplus I4 \oplus I5)$
- Data request triggers data response (⇨ next slide)

The LIN Bus

- Data response (sent by slave)
  - Slave responds with up to 8 Bytes of data
    - LSB first, Little Endian
    - length was defined by LIN Identifier
  - Frame ends with checksum
    - LIN 1.3: Classic Checksum (only data bytes)
    - LIN 2.0: Enhanced Checksum (data bytes + Identifier)
    - Checksum is sum of all Bytes (mod 256), plus sum of all carries

The LIN Bus

- Types of requests
  - Unconditional Frame
  - Event-triggered Frame
  - Sporadic Frame
  - ...
- Unconditional Frame
  - Most simple frame type
  - Designed for periodic polling of specific data point
  - Exactly one slave answers
  - LIN is a single master system → timing of unconditional frames fully deterministic
  - Sample use case:
    - Request "did state of front left door contact change?" every 15 ms
    - Receive negative reply by front left door ECU every 15 ms

The LIN Bus

- Types of requests
  - Unconditional Frame
  - Event-Triggered Frame
  - Sporadic Frame
  - ...
- Event-Triggered Frame
  - Simultaneous polling of multiple slaves, slave answers if needed
  - Collisions possible (→ non-determinism), detect by corrupt. data
    - master switches to individual polling via Unconditional Frames
  - Use whenever slaves unlikely to respond
  - Sample use case:
    - Request “did state of a door contact change?” every 15 ms
    - Change in state unlikely, simultaneous change extremely unlikely

The LIN Bus

- Types of requests
  - Unconditional Frame
  - Event Triggered Frame
  - Sporadic Frame
  - ...
- Sporadic Frame
  - Sent (by master) only when needed
  - Shared schedule slot with other Sporadic Frames
  - Use whenever polling for specific data only seldom needed
  - If more than one Sporadic Frame needs to be sent, master needs to decide for one → no collision, but still non-deterministic
  - Sample use case:
    - Request "power window fully closed?" every 15 ms
    - ...only while power window is closing

The LIN Bus

- Sample schedule table

Slot	Type	Signal
1	Unconditional	AC
2	Unconditional	Rain sensor
3	Unconditional	Tire pressure
4	Event triggered	Power window
5	Sporadic	(unused) -OR- Fuel level -OR- Outside temp

The LIN Bus

- Doing Off-Board-Diagnosis of LIN ECUs
  - Variant 1: Master at CAN bus responds on behalf of ECU on LIN
    - Keeps synchronized state via LIN messages
  - Variant 2: Master at CAN bus tunnels, e.g., KWP 2000 messages
    - Standardized protocol
    - LIN dest address is 0x3C (Byte 1 is ISO dest address)
    - Dest ECU (according to ISO address) answers with address 0x3D
    - Independent of payload, LIN frame padded to 8 Bytes
    - LIN slaves have to also support KWP 2000
    - Contradicts low cost approach of LIN
    - “Diagnostic Class” indicates level of support

Main Takeaways

- LIN
    - Goals
    - Deployment as sub bus
    - Message types and scheduling
    - Determinism

Main Takeaways

- Overall (K-Line, CAN, LIN)
  - Design goals
  - Message orientation vs. address orientation
  - Addressing schemes
  - Medium access
  - Flow control
  - Real time guarantees and determinism

Outline

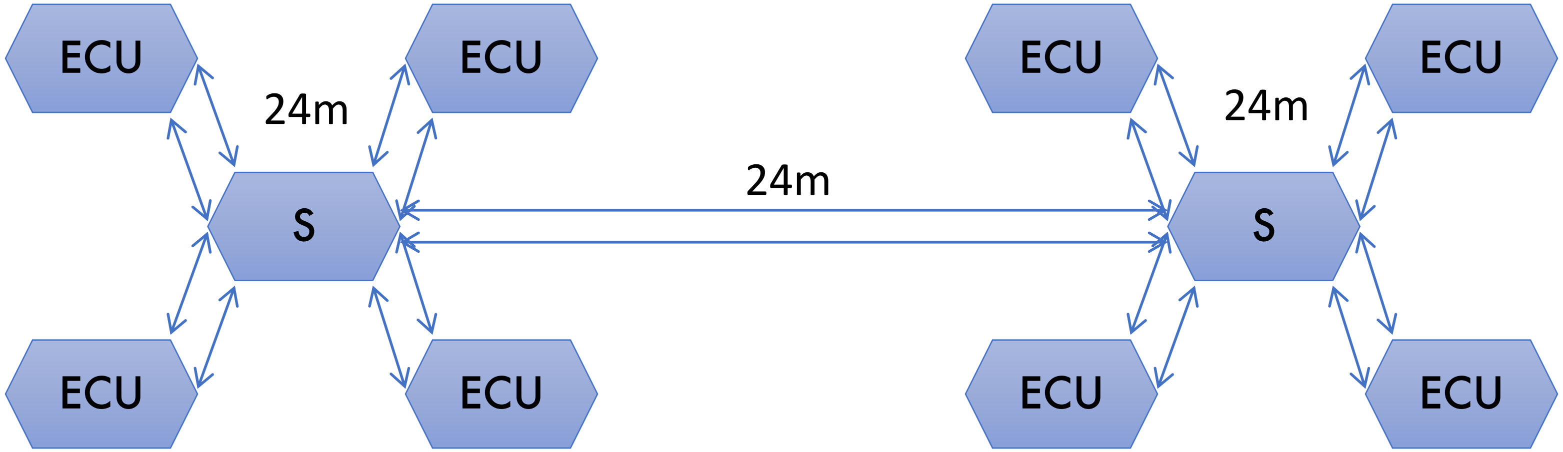
- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST
  - In-car Ethernet
- ECUs
- Safety

FlexRay

- Motivation
  - Drive/Brake/Steer-by-Wire
  - CAN bus is prone to failures
    - Line topology
    - No redundant links
  - CAN bus is slow
  - Need for short bus lines ⇨ deployment expensive, complicated
  - Non-determinism for all but one message class
    - Worst case delay unacceptably high
  - Early solutions by OEMs proprietary
    - TTCAN, TTP/TTA, Byteflight, ...
  - Foundation of consortium to develop new bus: FlexRay
    - BMW, VW, Daimler, GM, Bosch, NXP, Freescale
  - First series deployment at end of 2006 (BMW X5)

FlexRay

- Bus topology
  - Line, Star with bus termination
  - Max. distance per line: 24m
  - Optional use of second channel
    - Higher redundancy or(!) higher speed
    - Up to 10 MBit/s for single channel, 20 MBit/s for dual channel



FlexRay

- Bit transmission
  - Need synchronized clocks in sender and receiver
  - Thus, need additional bits for synchronizing signal sampling at receiver (done with each 1⇨0 flank)
  - Don’t use bit stuffing
    otherwise: message length becomes non-deterministic (cf. CAN)
  - New concept: frame each transmission, each frame, each Byte
    - Bus idle (1)
    - Transmission Start Signal (0)
      - Frame Start Signal (1)
        - Byte Start Signal (1)
        - Byte Start Signal (0)
        - 8 Bit Payload (…)
      - Frame End Signal (0)
    - Transmission End Signal (1)

FlexRay

- Bus access
  - Bus cycle (ca. 1 µs .. 7 µs)
    - Static Segment
    - Dynamic Segment (opt.)
    - Symbol Window (opt.)
    - Network Idle Time
  - Global *Cycle Counter* keeps track of bus cycles passed

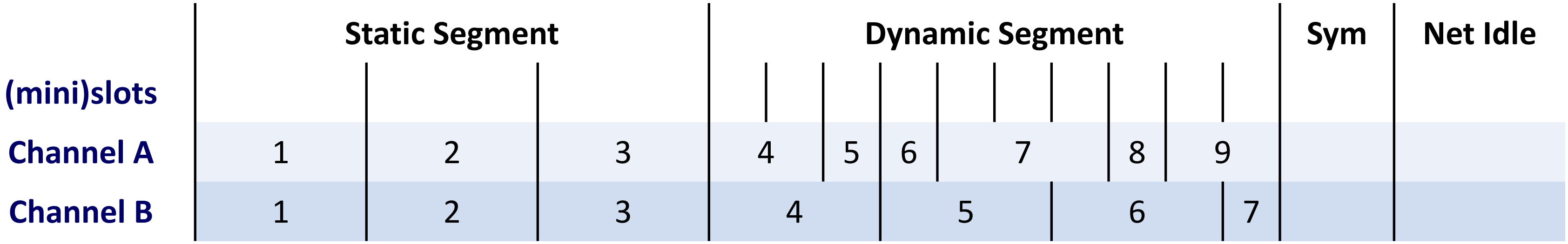
- Static Segment
  - Slots of fixed length (2 .. 1023)
  - One Message per Slot
  - Static assignment (of slot and channel) to ECUs (i.e., TDMA)

  ⇨ bus access is collision free, deterministic

FlexRay

- Dynamic Segment
  - Split into minislots (also statically assigned to ECUs)
  - Messages (usually) take up more than one minislot
  - Slot counter pauses while message is being transmitted (thus, slot counters of channels A and B soon desynchronize)
  - Lower priority messages have higher slot number (thus sent later, or not at all)

- Example:



FlexRay: Message format

- Control Bits
  - Bit 0: Reserved
    - Unused, always 0
  - Bit 1: Payload Preamble Indicator
    - In static segment:
      first 0 .. 12 Byte payload for management information
    - In dynamic segment:
      first 2 Byte payload contains Message ID (cf. UDP Port)

5 Bit	11 Bit	7 Bit	11 Bit	6 Bit		24 Bit
Control Bits	Frame ID	Length	Header CRC	Cycle Counter	Payload	CRC

FlexRay: Message Format

- Control Bits
  - Bit 2: Null Frame Indicator
    - Indicates frame without payload
    - Allows sending “no message” also in static segment (fixed slot lengths!)
  - Bit 3: Sync Frame Indicator
    - Indicates frame may be used for synchronizing clock
    - To be sent by 2 .. 15 “reliable” ECUs
  - Bit 4: Startup Frame Indicator
    - Used for synchronization during bootstrap
    - Sent by cold start node (⇨ later slides)

5 Bit	11 Bit	7 Bit	11 Bit	6 Bit		24 Bit
Control Bits	Frame ID	Length	Header CRC	Cycle Counter	Payload	CRC

FlexRay: Message Format

- Frame ID
  - Identifies message (≙ slot number)
- Length
  - Length of payload (in 16 Bit words)
- Header CRC
- Cycle Counter
  - Global counter of passed bus cycles
- Payload
  - 0 .. 127 16 Bit words (≙ 0 .. 254 Byte of payload)
- CRC

5 Bit	11 Bit	7 Bit	11 Bit	6 Bit		24 Bit
Control Bits	Frame ID	Length	Header CRC	Cycle Counter	Payload	CRC

FlexRay: Time Synchronization

- Need synchronized bit clock + synchronized slot counter
- Want no dedicated time master ⇨ Distributed synchronization
- Configure (typically) three nodes as “cold start nodes”

- Cold start procedure (followed by all cold start nodes):
  - Check if bus idle
    - if bus not idle ⇨ abort (cold start already proceeding or unneeded)
  - Transmit wakeup (WUP) pattern
    - if collision occurs ⇨ abort
    - if no collisions occurred ⇨ this is the leading cold start node
- Cold start procedure (leading cold start node):
  - Send Collision Avoidance Symbol (CAS)
  - Start regular operations (cycle counter starts at 0)
    - Set Bits: Startup Frame Indicator ⊕ Sync Frame Indicator

UNIVERSITÀ DI PARMA

MOTORVEHICLE UNIVERSITY OF EMILIA-ROMAGNA

FlexRay: Time Synchronization

- Cold start procedure (other cold start nodes)
  - Wait for 4 Frames of leading cold start node
  - Start regular operations
    - Set Bits: Startup Frame Indicator ⊕ Sync Frame Indicator
- Cold start procedure (regular ECUs)
  - Wait for 2 Frames of 2 cold start nodes
  - Start regular operations

1	WUP	WUP	CAS	0	1	2	3	4	5	6	7	8	...
2	WUP	ϟ						4	5	6	7	8	...
3	ϟ							4	5	6	7	8	...
4										6	7	8	...
5										6	7	8	...

FlexRay

- Illustrative configuration of timing
  - Use fixed payload length of 16 Byte
    (with header and trailer: 24 Bytes; with FSS, BSS, FES: ca. 250 Bits)
  - 10 Mbps data rate ⇨ 25 µs message duration
  - Add 5 µs guard to care for propagation delay and clock drift
    ⇨ 35 µs slot length in static segment
  - One macro tick: 1 µs (can use 1 .. 6 µs)
  - One minislot: 5 macro ticks: 5 µs
  - Tbit = 100 ns, sample rate of bus = Tbit/8 = 12.5 ns

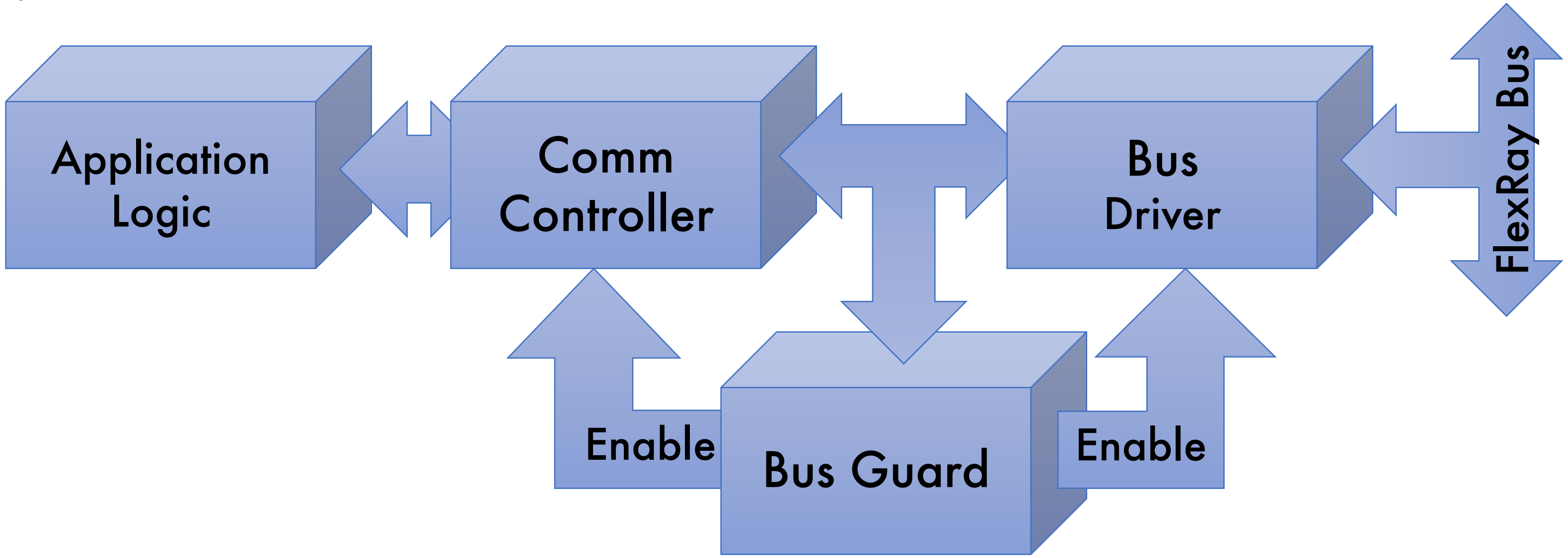
FlexRay

- Illustrative configuration of timing (contd.)
  - Use 64 distinct communication cycles
  - Communication cycle duration: 5 ms
  - Use 3 ms for static segment
  - Remaining 2 ms used for dynamic segment, symbol window, network idle time

- Message repetition interval fully customizable, e.g.:
  - 2.5 ms (one slot each at start and end of static segment)
  - 5 ms (one slot each in every communication cycle)
  - 10 ms (one slot in every second communication cycle)
  - …

FlexRay: Error prevention

- Integrate bus guard
- Implement separately from communication controller
- Follows protocol steps in communication controller
- Can only enable bus driver when allowed to communicate, or permanently disable in case of errors (*babbling idiot problem*)



FlexRay: Error Handling

- Multiple measures for error detection
  - Check cycle counter value
  - Check slot counter value
  - Check slot timing
  - Check header CRC
  - Check CRC
- Reaction to timing errors
  - Do not automatically repeat messages (⇨ non-determinism)
  - Switch to passive state instead
    - Stop transmitting messages
    - Keep receiving messages
      (might allow re-synchronization to bus)
- Reaction to severe, non-recoverable errors
  - Completely switch off bus driver

FlexRay: AUTOSAR TP

- Transport protocol of FlexRay
- Upwards compatible to ISO 15765-2 (ISO TP for CAN)
- Adjusted and extended for FlexRay
- Difference in addressing
  - In CAN: CAN message ID assigned arbitrarily
  - In FlexRay: Frame ID ≙ Slot Number (i.e., not arbitrary)
    ⇨ cannot use source/destination addresses as IDs in lower layer
  - Address encoded only (and completely) in TP header

- Also:
  - New message types

1 .. 2 Byte	1 .. 2 Byte	1 .. 5 Byte	
Target Address	Source Address	PCI	Payload

FlexRay: AUTOSAR TP

- Frame types: Single Frame *Extended* / First Frame *Extended*
- Larger *data length* (DL) field allows for longer payload
  - Four kinds of first frames can indicate payloads of up to 4 GiB

<table>
<tr><th></th><th colspan="2">PCI Byte 0</th><th colspan="2">PCI Byte 1</th><th colspan="2">PCI Byte 2</th><th colspan="2">PCI Byte 3</th><th colspan="2">PCI Byte 4</th></tr>
<tr><td>Single Frame</td><td>0</td><td>DL</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Single Frame Extended*</td><td>5</td><td>0</td><td colspan="2">DL</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>First Frame</td><td>1</td><td colspan="3">DL</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>First Frame Extended*</td><td>4</td><td colspan="2">1</td><td>DL</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>“</td><td>4</td><td colspan="2">2</td><td colspan="3">DL</td><td></td><td></td><td></td><td></td></tr>
<tr><td>“</td><td>4</td><td colspan="2">3</td><td colspan="5">DL</td><td></td><td></td></tr>
<tr><td>“</td><td>4</td><td colspan="2">4</td><td colspan="7">DL</td></tr>
</table>

FlexRay: AUTOSAR TP

- Extended flow control
  - FS values allow triggering abort of ongoing transmission
    - FS=2: Overflow
    - FS=5: Cancel, Data Outdated
    - FS=6: Cancel, No Buffer
    - FS=7: Cancel, Other
  - ST split into two ranges to allow shorter separation times
    - 0x00 .. 0x7F Separation Time in ms
    - 0xF1 .. 0xF9 Separation Time in μs (new!)

<table>
<tr><th></th><th colspan="2">PCI Byte 0</th><th colspan="2">PCI Byte 1</th><th colspan="2">PCI Byte 2</th><th colspan="2">PCI Byte 3</th><th colspan="2">PCI Byte 4</th></tr>
<tr><td>Consecutive Frame</td><td>2</td><td colspan="2">SN</td><td colspan="7"></td></tr>
<tr><td>Consecutive Frame 2*</td><td>6</td><td colspan="2">SN</td><td colspan="7"></td></tr>
<tr><td>Flow Control Frame</td><td>3</td><td>FS</td><td colspan="2">BS</td><td colspan="2">ST</td><td colspan="4"></td></tr>
<tr><td>Acknowledge Frame*</td><td>7</td><td>FS</td><td colspan="2">BS</td><td colspan="2">ST</td><td>ACK</td><td>SN</td><td colspan="2"></td></tr>
</table>

FlexRay: AUTOSAR TP

- Extended flow control
  - CAN: Acknowledgement by transmitting dominant bit in ACK field
  - FlexRay: New Acknowledge Frame (AF)
  - Use after single frame or after all consecutive frames (as ACK) or immediately (as NACK)
  - Functions identical to Flow Control Frame, but adds *ACK* and *SN* nibbles
    - ACK is 1 or 0; SN indicates slot number of first defective frame
  - Sender may repeat failed transmissions at earliest convenience (alternately uses CF and CF2 frames)

<table>
<tr><th></th><th colspan="2">PCI Byte 0</th><th colspan="2">PCI Byte 1</th><th colspan="2">PCI Byte 2</th><th colspan="2">PCI Byte 3</th><th colspan="2">PCI Byte 4</th></tr>
<tr><td>Consecutive Frame</td><td>2</td><td colspan="2">SN</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Consecutive Frame 2*</td><td>6</td><td colspan="2">SN</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Flow Control Frame</td><td>3</td><td>FS</td><td colspan="2">BS</td><td colspan="2">ST</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Acknowledge Frame*</td><td>7</td><td>FS</td><td colspan="2">BS</td><td colspan="2">ST</td><td>ACK</td><td>SN</td><td></td><td></td></tr>
</table>

Outline

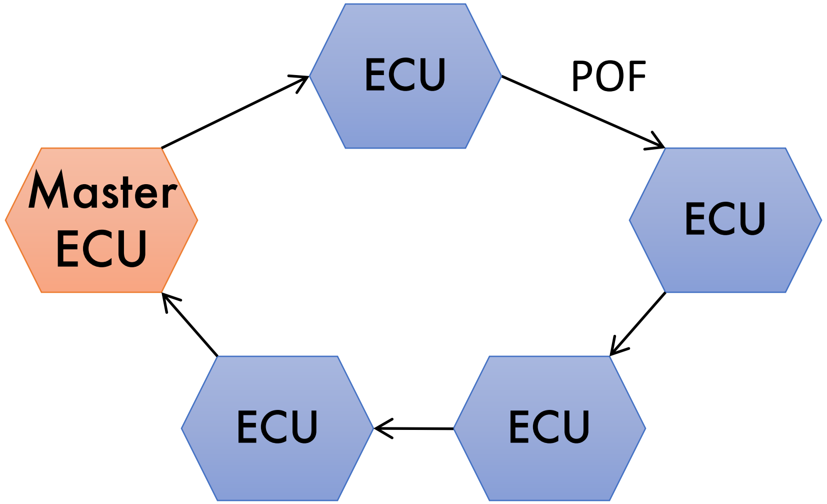
- Bus systems: basics
- Protocols
    - K-Line
    - CAN
    - LIN
    - FlexRay
    - MOST: Media Oriented Systems Transport
    - In-car Ethernet
- ECUs
- Safety

MOST

- Media Oriented Systems Transport
  - specifies ISO layers 1 through 7
  - Does not focus on sensor/actor tasks (e.g., delay, fault tolerance), but on infotainment (e.g., jitter, data rate)
- History
  - Domestic Data Bus (D2B, later: Domestic Digital Bus) developed by Philips, later standardized as IEC 61030 (still in the 90s)
  - Little adoption in vehicles, thus SMSC soon develops a successor
  - 1998: MOST Cooperation standardizes MOST bus (Harman/Becker, BMW, DaimlerChrysler, SMSC)
  - December 2009: MOST 3.0E1 published
  - Today:
    MOST cooperation numbers 60 OEMs, 15 vehicle manufacturers

MOST: Medium

- Plastic Optic Fiber (POF)
  alternative (copper) variant specified, but little used
- Data rates specified from 25 (MOST25) to 150 MBit/s (MOST150)
- Manchester coded bit transmission
- Dedicated timing master ECU (slaves adopt bit timing)
- Logical bus topology: ring of up to 64 ECUs
- Physical bus topology can differ



MOST: Link Layer

- Synchronous bit stream; all clocks synchronized to timing master
- Stream divided into blocks; each block traverses ring exactly once
- Blocks divided into 16 Frames
  - Frame size: 64 Byte (MOST25) to 384 Byte (MOST150)
  - Frame rate static but configurable; recommended: 48 kHz (DVD)
- Frame divided into
  - Header (with boundary descriptor) and Trailer
  - Data: Synchronous Channel, Asynchronous Channel, Control Channel

MOST: Link Layer

- Synchronous Channel
  - Use case: audio or video
  - TDMA divides frame into streaming channels
    ⇨ deterministic
  - Reserved by messages on control channel
  - Thus, no addressing required
  - Maximum number of streaming channels limited by frame size

Streaming Channel 1	Streaming Channel 2	Streaming Channel 3	unused
CD-Audio, Device A	DVD-Video, Device B	...	

MOST: Link Layer

- Asynchronous Channel
  - Use case: TCP/IP
  - Random access with arbitration (based on message priority)
    ⇨ non-deterministic
  - Single message may take more than one frame
  - Short additional header contains source/destination address, length
  - Short additional trailer contains CRC
  - No acknowledgement, no automatic repeat on errors

1 Byte	2 Byte	1 Byte	2 Byte		4 Byte
Arbitration	Target Address	Len	Source Address	...	CRC

MOST: Link Layer

- Control Channel
  - Management and control data
  - Random access with arbitration (based on message priority)
  - Message length 32 Byte
    - MOST25 control channel uses 2 Bytes per frame
      ⇨ each message takes 16 Frames = 1 Block
  - Message reception is acknowledged by recipient
  - Failed transmissions are automatically repeated

1 Byte	2 Byte	2 Byte	1 Byte	17 Byte	2 Byte	1 Byte
Arbitration	Target Address	Source Address	Type	Data	CRC	Trailer

MOST: Link Layer

- Control Channel messages
  - Resource Allocation,
    Resource De-allocation:
    - manage streaming channels in synchronous segment
  - Remote Read,
    Remote Write
    - accesses registers and configuration of ECUs
  - Remote Get Source
    - query owner of streaming channels in synchronous segment
  - …
    - Other message types are transparently passed to upper layers

MOST: Link Layer

- Addressing
  - 16 Bit addresses
  - physical address
    - According to relative position in ring
    - Master gets 0x400
    - First slave gets 0x401
    - etc.
  - logical address
    - Assigned by master
    - Typically upwards of 0x100 (Master)
  - groupcast
    - Typically 0x300 + ID of function block
  - broadcast
    - Typically 0x3C8

MOST: Ring Disruption

- Causes
  - ECU stops working
  - Plastic optic fiber gets damaged
- Symptoms
  - Messages either not transmitted to recipient, or not back to sender thus: total failure of bus
- Diagnosis
  - Ring disruption easily detected
  - Reason and affected ECUs impossible to determine
- Workarounds
  - Vendor dependent, proprietary
  - often: use additional single-wire bus for further diagnosis

MOST: Higher Layers

- Object oriented MOST Network Services
  - Function block (= class)
    - e.g. audio signal processing (0x21), audio amplifier (0x22), ...
    - Multiple classes per device, multiple devices per class
    - Every device implements function block 0x01 (MOST Netw. Services)
  - Instance
    - Uniquely identifies single device implementing certain function block
  - Property/Method
    - Property (get/set value)
    - Method (execute action)
  - Operation
    - Set/Get/... (Property), Start/Abort/... (Method)
  - 22.00.400.0 (20) ⇨ amplifier number 0: volume set to 20

MOST: Higher Layers

- System boot and restart
  - Master node announces reset of global state
    (all devices change status to Not-OK and cease operations)
  - Master node initiates system scan
    - Iteratively polls all physical addresses for present function blocks
    - Devices answer with logical address, list of function blocks, and instance numbers
  - Master can detect ambiguous combinations of function blocks and instance numbers ⇨ will then assign new instance numbers
  - Master keeps table of all device's operation characteristics
  - Master reports to all devices: status OK
  - MOST Bus is now operational

MOST: Higher Layers

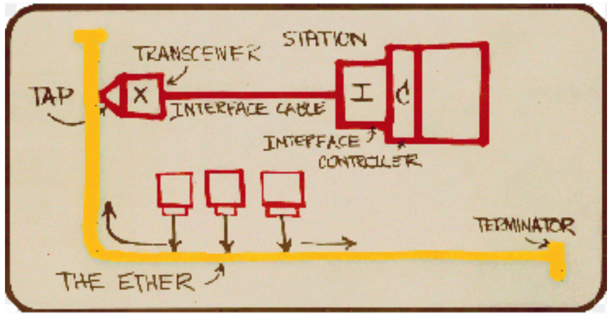
- MAMAC and DTCP
  - Trend towards all-IP in consumer electronics addressed in MOST by introducing MAMAC (MOST Asynchronous Media Access Control)
    - Encapsulates Ethernet and TCP/IP for transmission on MOST bus
    - but: not supported by MOST services; needs to be implemented in software
  - Concerns of music/film industry wrt. digital transmission addressed in MOST by introducing DTCP (Digital Transmission Content Protection)
    - As known from IEEE 1394 (FireWire)
    - Bidirectional authentication and key exchange of sender/receiver
    - Encrypted data transmission

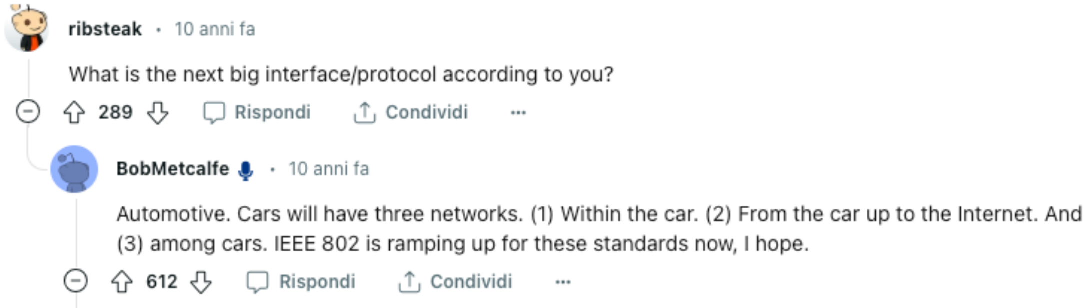
Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST: Media Oriented Systems Transport
  - In-car Ethernet
- ECUs
- Safety

In-Car Ethernet

- IEEE 802.3
- Bob Metcalfe, David R. Boggs
- 1973, Parc CSMA/CD Ethernet
  - 3 Mbit/s, 256 nodes, 1 km coax cable
- 1980- revised to become IEEE Std 802.3
- Next big thing?
  - “Automotive. Cars will have three networks. (1) Within the car. (2) From the car up to the Internet. And (3) among cars. IEEE 802 is ramping up for these standards now, I hope.” --/u/BobMetcalfe on http://redd.it/1x3fiq




ribsteak · 10 anni fa

What is the next big interface/protocol according to you?

289 Rispondi Condividi ...

BobMetcalfe · 10 anni fa

Automotive. Cars will have three networks. (1) Within the car. (2) From the car up to the Internet. And (3) among cars. IEEE 802 is ramping up for these standards now, I hope.

612 Rispondi Condividi ...

In-Car Ethernet

- Why?
  - Old concept:
    - Strictly separated domains
    - Each served by specialized bus
    - Minimal data interchange
  - Current trend:
    - Advanced Driver Assistance Systems (ADAS)
    - Sensor data fusion
      - (in-car, between cars)
    - Ex: Cooperative Adaptive Cruise Control (CACC)
  - Move from domain specific buses ⇨ general-purpose bus

Ethernet

- Physical layers
  - 10BASE5 (aka Thicknet, aka IEEE Std 802.3-1985)
    - Manchester coded signal, typ. 2 V rise
    - 10 Mbit/s over 500m coax cable
    - Nodes tap into core (“vampire tap”)
  - 10BASE2
    - 10 Mbit/s over “almost” 200m coax cable
    - BNC connectors, T-shaped connectors
  - 1000BASE-T
    - 1 Gbit/s over 100m
    - Cat 5e cable with 8P8C connectors,
4 twisted pairs of wires,
multi-level signal (-2, -1, 0, +1, +2),
scrambling, …
    - Medium access
    - No longer shared bus, but point to point
    - Auto-negotiated (timing) master/slave
  - 100GBASE-ER4
    - 100 Gbit/s over 40 km
    - Plastic Optic Fiber (POF)
- Medium access: CSMA/CD
  - Carrier sensed ⇨ medium busy
  - Collision ⇨ jam signal, binary exponential backoff (up to 16 times)

Ethernet

- Link layer
  - Lightweight frame type
  - Optional extensions, e.g., IEEE 802.1Q (identifier 0x8100)
  - Directly encapsulates higher layer protocols, e.g., IPv6 (0x86DD)
  - …or IEEE 802.2 Logical Link Control (LLC) frame (identifier is len)

(in Byte)

  - Error-checked, but only best effort delivery of data

<table>
<tr><th>0</th><th></th><th></th><th></th><th></th><th></th><th></th><th>7</th><th>8</th><th></th><th></th><th></th><th></th><th></th><th></th><th>15</th></tr>
<tr><td colspan="8">Preamble (1010..11)</td><td colspan="6">Destination MAC</td><td colspan="2"></td></tr>
<tr><td colspan="4">Source MAC</td><td colspan="4">(opt) 802.1Q tag</td><td colspan="2">Type/len</td><td colspan="6"></td></tr>
<tr><td colspan="16">Payload (commonly 42-1500 Byte, max 1982 Byte)</td></tr>
<tr><td colspan="4">Checksum</td><td colspan="12">(Idle time)</td></tr>
</table>

In-Car Ethernet

- In-car Ethernet?
  - Almost all “in-car” qualities absent
    - Heavy, bulky cabling
    - Huge connectors
    - Sensitive to interference
    - Needs external power
    - No delay/jitter/… guarantees
    - No synchronization
    - Etc…
  - But:
    - …can be easily extended:
    - New physical layers
    - Tailored higher-layer protocols

In-Car Ethernet

- One-Pair Ether-Net (OPEN) alliance SIG
  - Founded: BMW, Broadcom, Freescale, Harman, Hyundai, NXP
  - 2014: approx. 150 members
  - 100 Mbit/s on single twisted pair, unshielded cable
  - Power over Ethernet (IEEE 802.3at)
  - Manufactured by Broadcom, marketed as *BroadR-Reach*

- Reduced Twisted Pair Gigabit Ethernet (RTPGE) task force
  - Working on IEEE 802.3bp
  - 1 Gbit/s over up to 15m single twisted pair cable

In-Car Ethernet

- Upper layers: TSN
  - Many solutions (e.g., SAE AS6802 “Time Triggered Ethernet”)
  - Current: IEEE 802.1 Time Sensitive Networking (TSN) task group (aka Audio/Video Bridging AVB task group, up until 2012)
  - Promoted by *AVnu Alliance* SIG (cf. IEEE 802.11 / Wi-Fi Alliance)

- Concept
  - Needs TSN-enabled switches / end devices
  - Tight global time synchronization
  - Dynamic resource reservation on *streams* through network
  - IEEE 802.1AS… extensions
    - Layer 2 service
  - IEEE 802.1Q… extensions
    - Frame tagging standard

In-Car Ethernet

- IEEE 802.1AS Time Synchronizing Service
  - Subset of IEEE 1588 Precision Time Protocol (PTP)
  - Syncs clock value/frequency of all nodes
  - Election of "master" time master (grandmaster clock), disseminates sync information along spanning tree
- IEEE 802.1Qat Stream Reservation Protocol (SRP)
  - *Talker* advertises stream (along with parameters)
  - Advertisement is disseminated through network
  - Intermediate nodes check, block available resources, update advertisement with, e.g., newly computed worst case latency
  - *Listeners* check (annotated) advertisement, send registration message back to *Talker*
  - Intermediate nodes reserve resources, update multicast tree

In-Car Ethernet

- IEEE 802.1Qav etc. Traffic Shaping
  - Prioritize frames according to tags
  - Avoid starvation, bursts, …
  - e.g., Token bucket, with many more proposed
- IEEE 802.1Qbu Frame Preemption
  - Can cancel ongoing transmissions (if higher priority frame arrives)
- IEEE 802.1Qcb Media Redundancy
- …

Main Takeaways

- FlexRay
  - Motivation
  - Single or dual channel operation
  - Distributed operation
  - Static and dynamic segment
- MOST
  - Motivation
  - Topology and implications
  - Centralized operation
  - Synchronous and asynchronous channel
- Ethernet
  - Concept
  - Drawbacks of classic standards
  - New PHY layers
  - New upper layers (TSN)

Outline

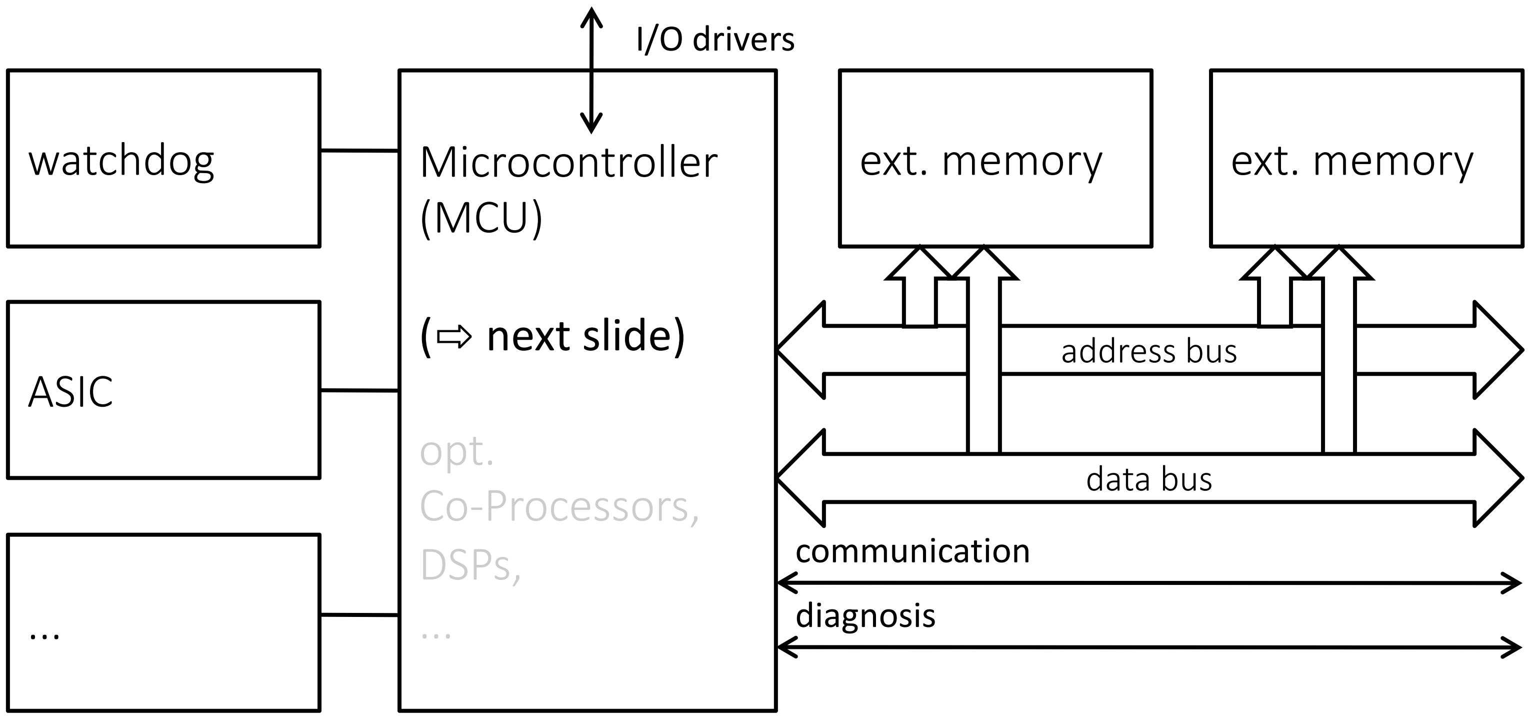
- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST: Media Oriented Systems Transport
  - In-car Ethernet
- ECUs: Electronic Control Units
- Safety

Electronic Control Units (ECUs)

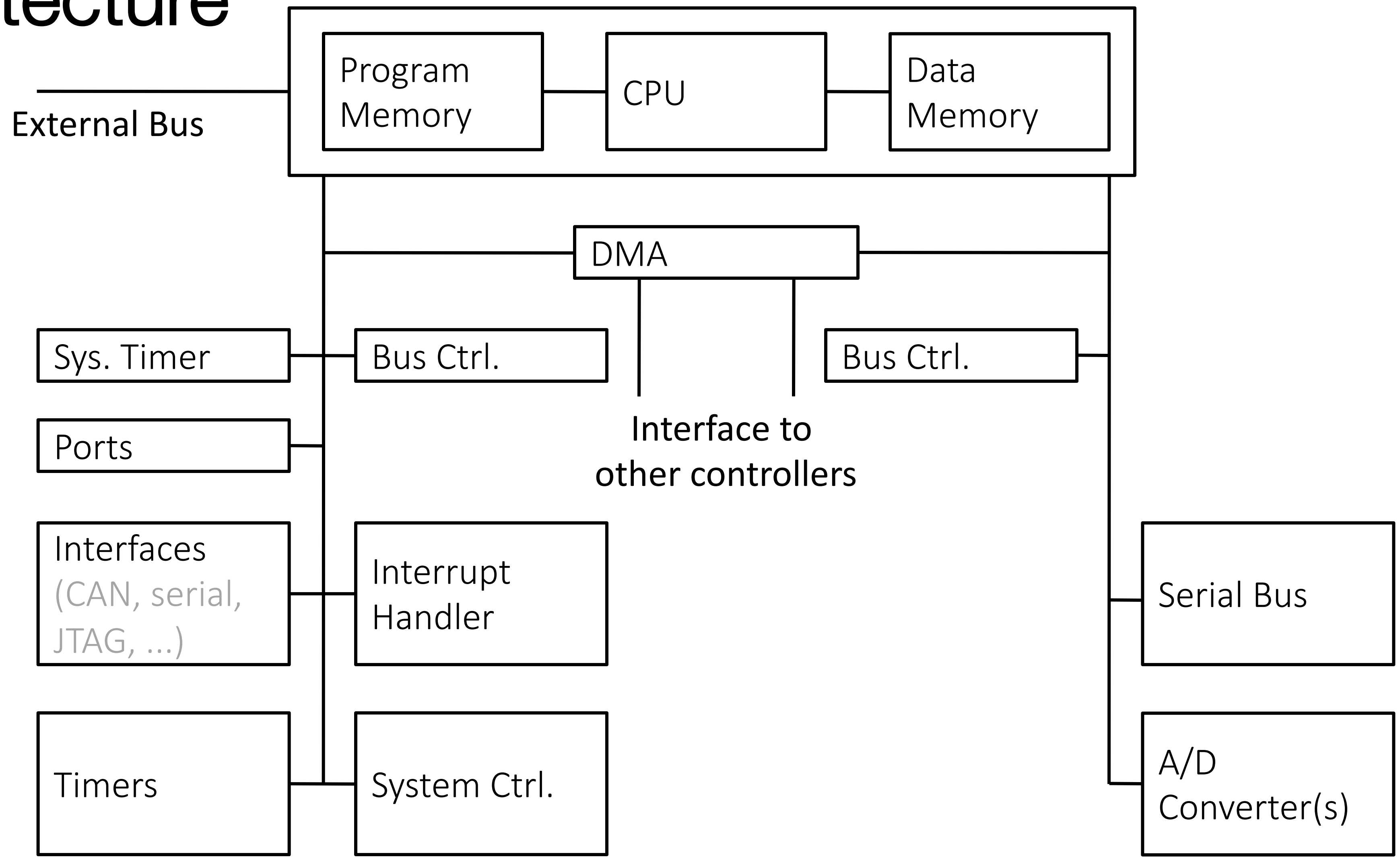
- Middle and upper class vehicles carry 80 .. 100 networked ECUs
- Each consisting of
  - Transceiver (for bus access)
  - Power supply
  - Sensor drivers
  - Actor drivers
  - …and an ECU Core (⇨ next slide)
- Depending on deployment scenario, ECU and components must be
  - Shock resistant
  - Rust proof
  - Water resistant, oil resistant
  - Heat resistant
  - …

ECU Core

- ≙ Personal Computer
- additional external guard hardware (e.g., watchdog) for safety critical applications



Architecture



Architecture

- MicroController Unit (MCU)
  - 8, 16, 32 Bit
  - Infineon, Freescale, Fujitsu, ...
- Memory
  - Volatile memory
    - SRAM (some kByte)
    - Typically integrated into microcontroller
  - Non-volatile memory
    - Flash (256 kByte .. some MByte)
    - Serial EEPROM (some kByte, e.g., for error log)
- Power supply
  - DC/DC converter, e.g., to 5 V or 3.3 V

Architecture

- Clock
  - Quartz Xtal, some 10 MHz (⇨ ECU requires only passive cooling)
- External guard hardware
  - Watchdog
    - Expects periodic signal from MCU
    - Resets MCU on timeout
  - ASIC guard
    - For more complex / critical ECUs
    - ASIC sends question, MCU must send correct answer before timeout
    - Resets (or disables) ECU on timeout or error
- Internal Buses
  - Low-cost ECUs can use shared bus for address and data
  - Parallel

Architecture

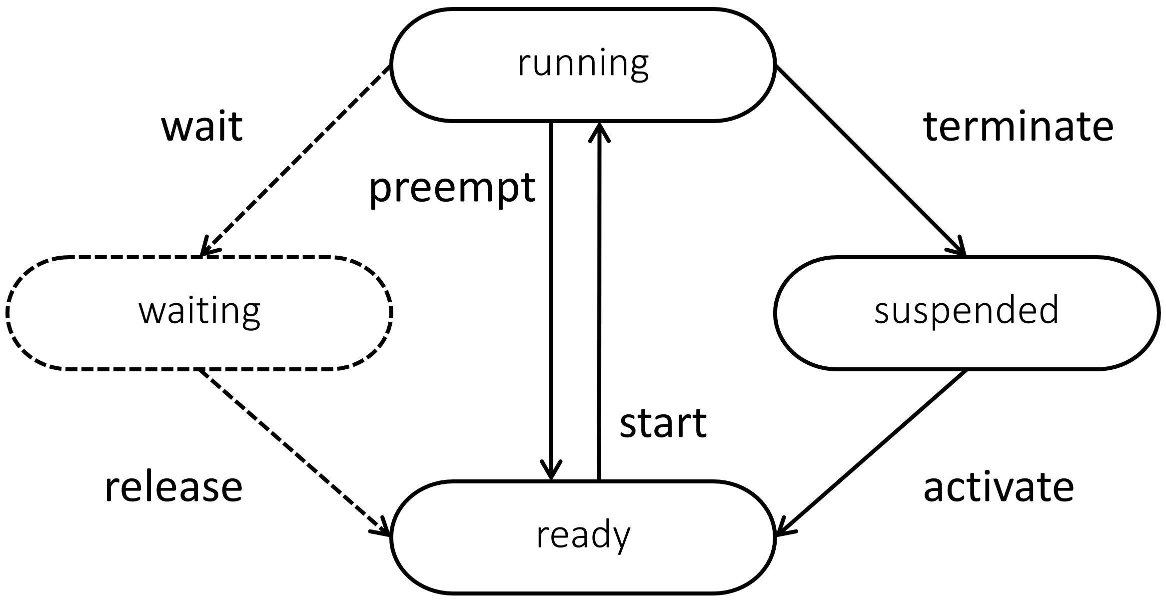
- Sensor drivers
  - Resistive sensors (e.g., simple potentiometer for length, angle)
  - Capacitive, inductive sensors (e.g., pressure, distance)
  - Active sensors (simple voltage / complex data output)
- Actor drivers
  - D/A conversion
  - High-power amplifiers
  - Bridges
- Further requirements
  - Electro-magnetic interference (EMI) characteristics
  - Mechanical robustness
  - Water resistance
  - Thermal resistance
  - Chemical resistance

Automotive Operating Systems

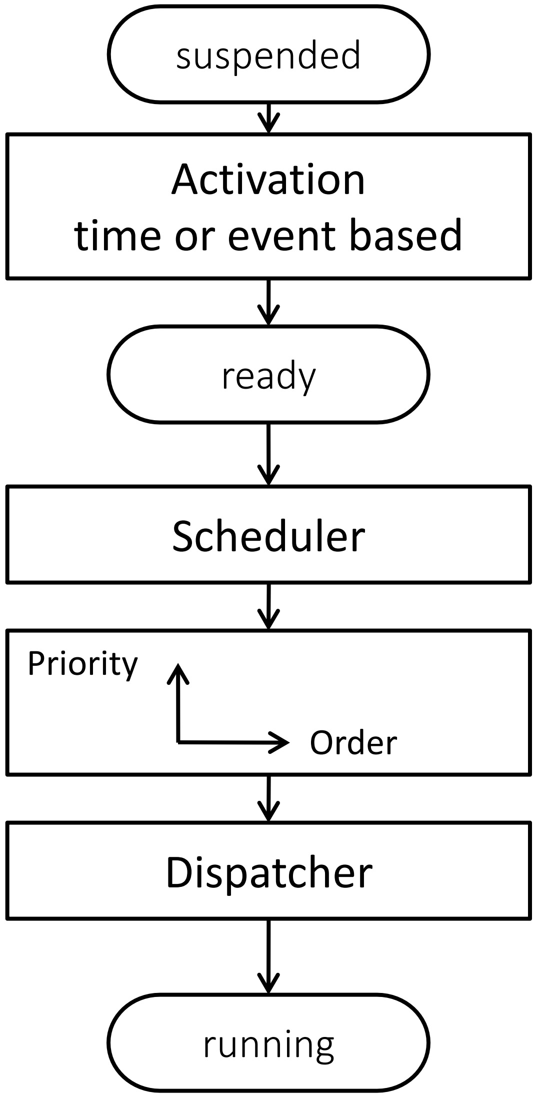
- Hardware abstraction
  - Often missing, hardware accessed directly
  - Recent trends towards operating systems
- Application Programming Interface (API)
  - Common for message transmission over external buses
- Software safeguards
  - E.g., stack overflow
  - Particularly helpful during development

Automotive Operating Systems

- Process States



Scheduling



Set of suspended tasks

Set of ready tasks

Priority queue of ready tasks

Task executed

Automotive Operating Systems

- Scheduling
  - The act of assigning an order of activation, given a process model, activation sequence, and deadlines
    - *dynamic*: Schedule is calculated at run time
    - *static*: Schedule is fixed, e.g., at compile time (⇨ fully deterministic)
  - *Feasible schedule*: all time constraints fulfilled, no deadline violated
  - Dispatcher coordinates context switches

- Context switches
  - For one process to change state to *running*, another process may need to be preempted
  - CPU registers etc. will now be occupied by new process, operating system takes care of persisting information

Real Time Properties

- Latency
  - Time difference from event to reaction
- Jitter
  - Difference of max and min latency
  - High importance in feedback control systems
- Execution time
  - Time difference of task start and end
  - Worst Case Execution Time (WCET)
    - Defined for program aspects, dependent on platform
    - Considers every possible cause of delay (interrupts, caching, …)
    - Important for guaranteeing determinism

Real Time Properties

- Soft deadline
  - Delivering result after soft deadline less helpful (reduced benefit)
  - e.g., car speeds up ⇨ radio gets louder
- Firm deadline
  - Delivering result after firm deadline useless (no benefit)
  - e.g., incoming traffic bulletin ⇨ SatNav powered up
- Hard deadline
  - Delivering result after hard deadline causes damage or harm (negative benefit)
  - e.g., brake pedal is pushed ⇨ car decelerates

Real Time Properties

- Real time systems
  - Internal image of system state in memory
  - State described by set of variables
  - Needs continuous update of image

- Real time architecture
  - Event triggered system
    - Image update with every change of state
  - Time triggered system
    - Image update in fixed intervals
    - internal or global clock (needs synchronization)

OSEK/VDX

- 1993
  - Founded as OSEK – "*Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug*"
  - BMW, Bosch, Daimler Chrysler, Opel, Siemens, VW, Univ. Karlsruhe
- 1994
  - Merged with VDX – "*Vehicle Distributed Executive*"
  - PSA und Renault
- Today
  - More than 50 partners
  - (Parts) standardized as ISO 17356 series
  - Standardizes common communications stack, network management, **operating system** (⇨ next slides), …
  - Many free implementations (freeOSEK, openOSEK, nxtOSEK, …)

OSEK/VDX

- Properties
  - Operating system for single processor
  - Static configuration
    - Tasks
    - Resources
    - Functions
  - Can meet requirements of hard deadlines
  - Programs execute directly from ROM
  - Very low memory requirements
  - Standardized system (⇨ “OSEK conformant ECUs”)

OSEK/VDX

- Configuration
  - Operating system configured at compile time
  - OSEK Implementation Language (OIL)
    - Scheduling strategy
    - Task priorities
    - …

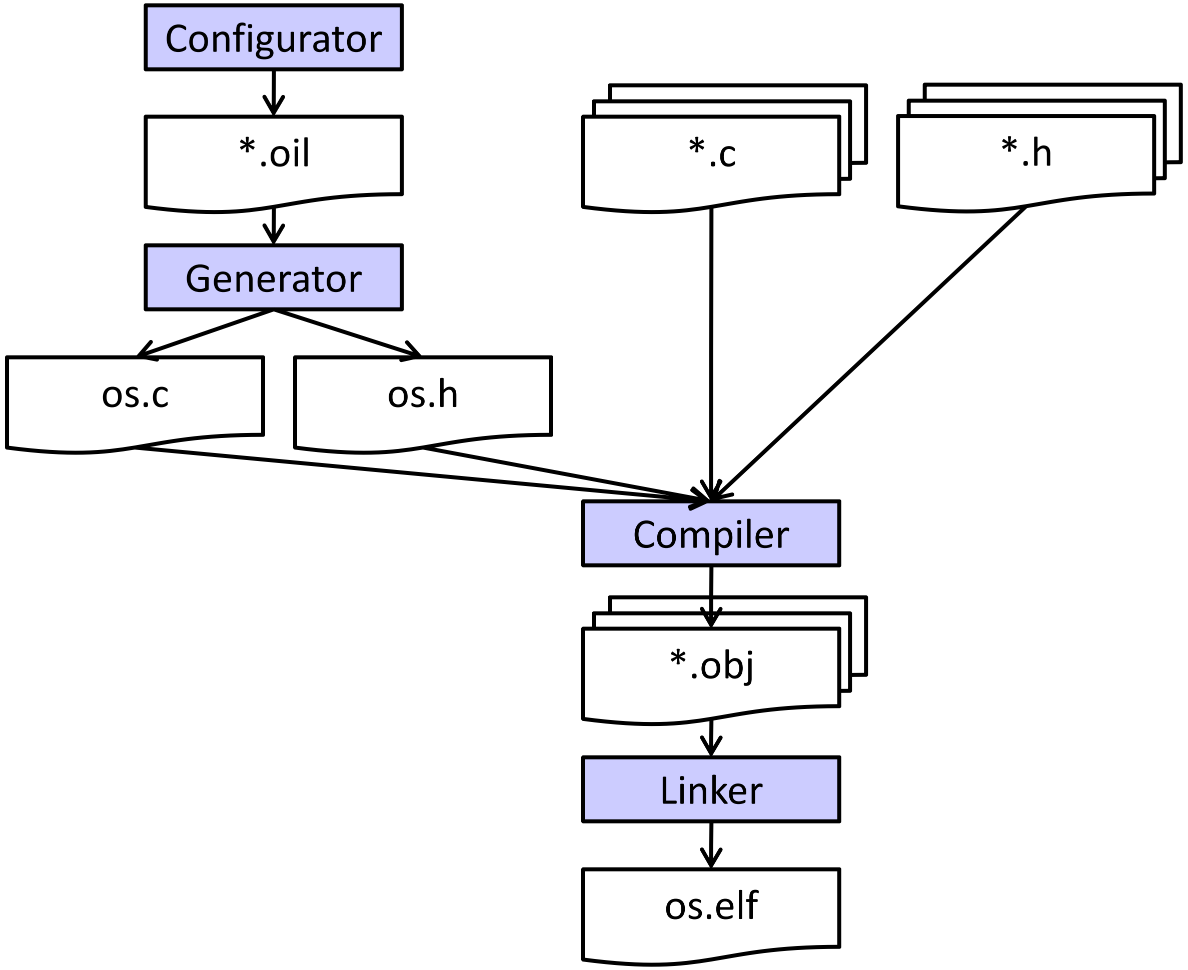
```
CPU OSEK_Demo
{

  OSEK_Example_OS
  {
    MICROCONTROLLER = Intel80x86;
    …
  };

  TASK Sample_TASK
  {
    PRIORITY = 12;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
  };

  …
};
```

Building of OSEK/VDX firmware



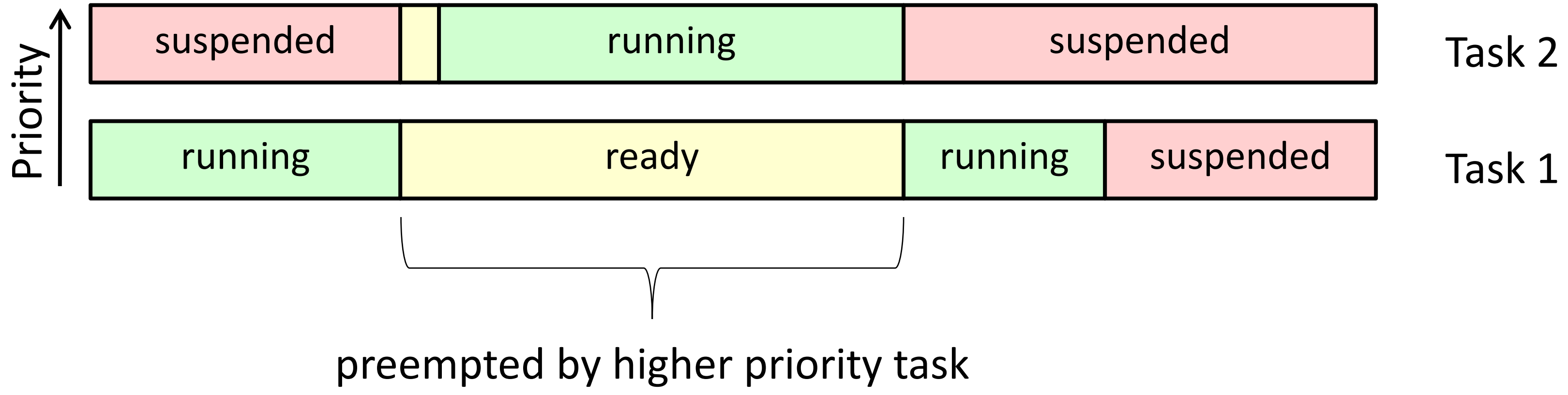
OSEK/VDX

- Tasks
  - Static priority
  - Relationships of tasks
    - Synchronization
    - Message exchange
    - Signaling
  - Support for time triggered services
  - Error management
  - C macros for definition provided

```
DeclareTask(SampleTask);
…
TASK(SampleTask) {
  /* read sensors, trigger actors */
  TerminateTask();
}
```

OSEK/VDX

- Scheduling
  - Scheduler always chooses highest priority task
  - Configurable modes:
    - Non preemptive: Tasks are never preempted
    - Preemptive: Higher priority tasks always preempt lower priority tasks
    - Mixed: Individual configuration of each task



AUTOSAR

- Traditional paradigm:
  one function ⇨ one ECU
  (incl. software and OS, supplied by OEM)

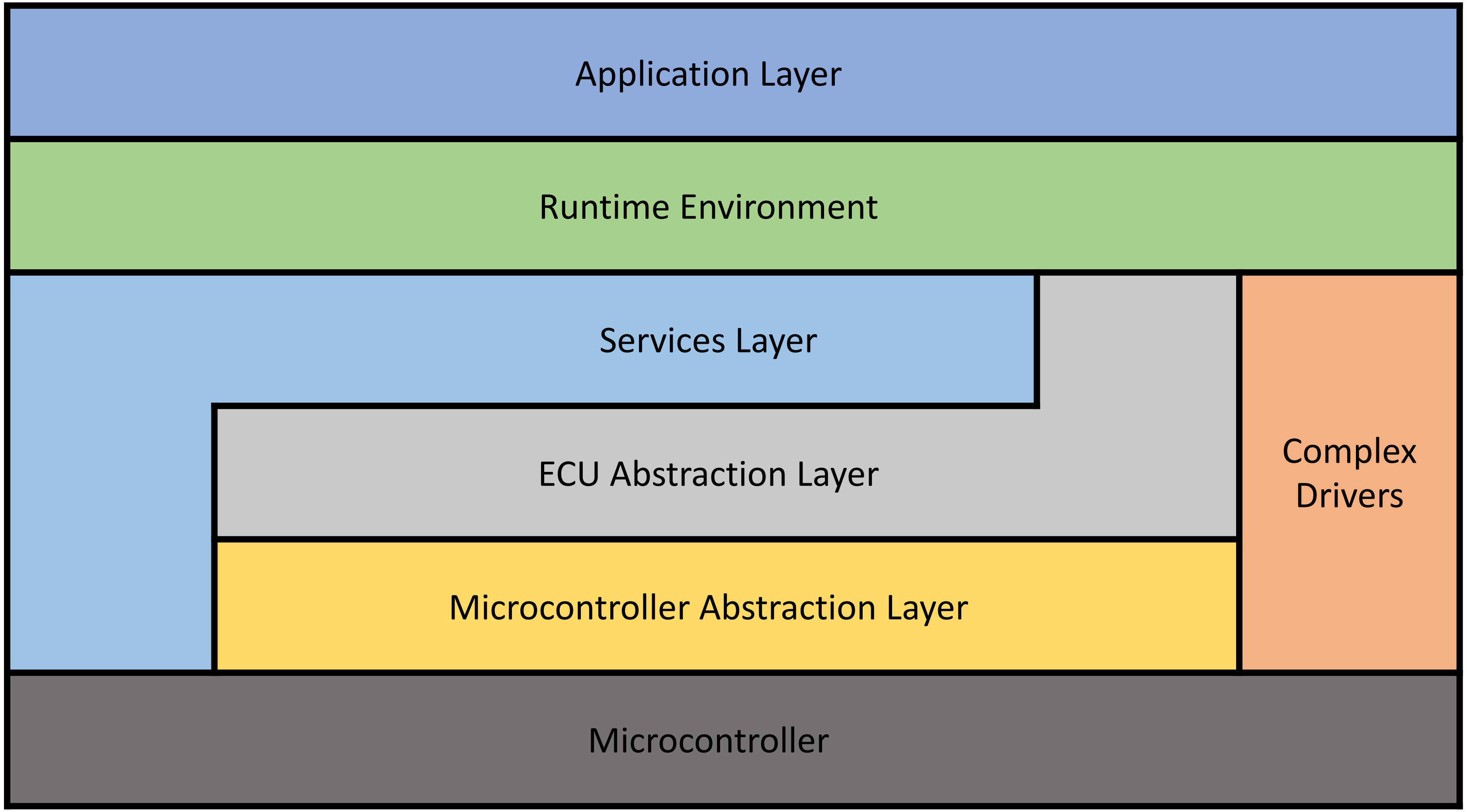
- AUTOSAR (*Automotive Open System Architecture*)
  Initiative of automobile manufacturers to make software development independent of operating system

- Mix and match of hardware and software
  - Integration at manufacturer
  - In-house development of software at manufacturer
  - Independence of/from OEM

AUTOSAR

- AUTOSAR Runtime Environment (RTE)
  - Middleware abstracting away from lower layers
- Application Software Components
  - Rely on strict interfaces, independent of MCU, Sensors, Actors

AUTOSAR


Application Layer
Runtime Environment
Services Layer
ECU Abstraction Layer
Microcontroller Abstraction Layer
Complex Drivers
Microcontroller

Main Takeaways

- ECUs
  - Principles
  - Architecture
  - Real-time properties (hard, firm, soft deadlines)
- OSEK/VDX
  - Motivation
  - Static configuration
  - Scheduling
- AUTOSAR
  - Motivation
  - Run Time Environment
  - Component Principle

Outline

- Bus systems: basics
- Protocols
  - K-Line
  - CAN
  - LIN
  - FlexRay
  - MOST: Media Oriented Systems Transport
  - In-car Ethernet
- ECUs: Electronic Control Units
- Safety

Aspects of Safety

- Errors can lead to
  - material damage
  - bodily injury

- Check if errors might endanger human lives
  - Concerns not just systems for active safety (Airbag, ABS, ...)
  - Also concerns, e.g., engine ECU (sudden acceleration)

- Integral part of ECU development
  - “First and last step” when introducing new functionality

Aspects of Safety: Terminology

- Risk:
  - Quantitative measure of uncertainty
    <risk> = <occurrence probability> × <consequences>
- Limit Risk:
  - Highest still acceptable risk
- Safety:
  - Condition that does not exceed limit risk
    (cf. DIN VDE 31000, Part 2)

Aspects of Safety: Terminology

- Error:
  - Deviation of calculated, observed, or measured value from true, specified, or theoretical value
- Fault:
  - DIN 40041: unpermitted deviation of one or more properties that allows the discrimination of machines or components
  - IEC 61508, Part 4: exceptional condition that might lead to a component no longer fulfilling (or only partly fulfilling) its function
- Failure:
  - DIN 40041: Component ceases to function (within permissible use)
  - IEC 61508, Part 4: System ceases fulfilling the specified function
- Malfunction:
  - (Potentially dangerous) consequence of failure
    - E.g., ABS: failure must not cause wheels to lock; instead: graceful degradation

Aspects of Safety: Terminology

- Functional Safety:
  - Subpart of safety that is reliant on correct function of safety relevant components (as well as external measures for reducing risk)
- Reliability:
  - Probability that a component does not fail within a defined time window
- Redundancy:
  - Duplication of components (where only one would be needed)
    - homogeneous redundancy:
      - components are identical
    - diverse redundancy:
      - components are not identical
  - E.g., dual circuit braking

Aspects of Safety: Laws and Norms

- Laws
  - minimum conditions (in the shape of general requirements)
  - no verification
  - but: product liability laws might require proof that development corresponds to state of the art
  - E.g., German Regulations Authorizing the Use of Vehicles for Road Traffic (StVZO)
- Norms
  - e.g. RTCA DO-178B (ED-12B) for aeronautic software
  - IEC 61508: standard for the development of safety critical systems

Aspects of Safety: IEC 61508

- Two modes of operation
  - Low Demand Mode
  - High Demand Mode or Continuous Mode

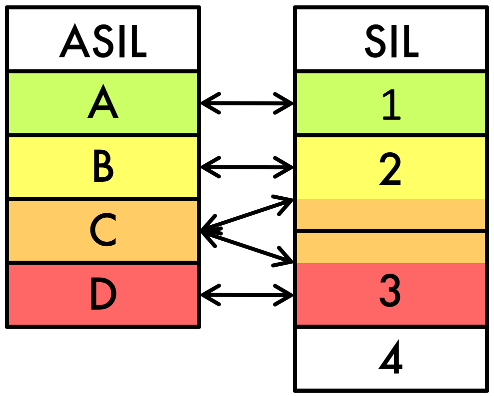
- Low Demand Mode
  - Safety critical system activated no more than twice per year (or maintenance interval)
  - Safety measures are passive (until needed)
  - E.g., airbag deployment on accident
- High Demand Mode or Continuous Mode
  - Safety critical system activated more than twice per year (or maintenance interval)
  - Safety measures keep system within safety margins
  - E.g., ensure that airbag cannot misfire

Aspects of Safety: IEC 61508

- 4 Safety Integrity Levels (SIL)
  - Relate to safety measure, not complete system
  - Describes risk reduction by safety measure
  - SIL 4: highest demands
    system failure triggers catastrophic consequences
- Process:
  - Damage and risk assessment
  - Determination of
    - Hardware Fault Tolerance (HFT)
    - Safety Failure Fraction (SFF)
  - Check if necessary SIL can be reached

Aspects of Safety: ISO 26262

- Adaption of IEC 61508 for automotive engineering
- “Automotive 61508”
- SIL ⇨ ASIL (Automotive Safety Integrity Level)



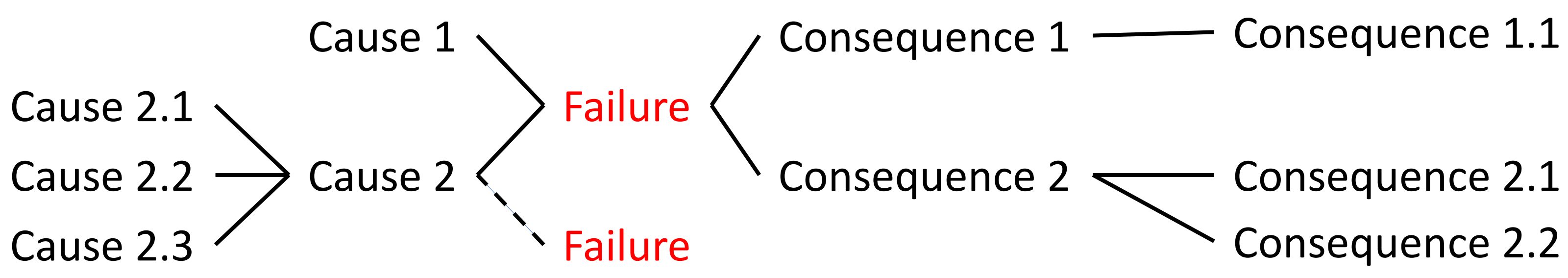
Aspects of Safety

- Methods for analyzing safety and reliability
  - Often rooted in aeronautic and space software development

- Covered in this lecture
  - Failure Mode Effect Analysis (FMEA)
    - Also called: Failure Mode Effect and Criticality Analysis (FMECA)
  - Fault Tree Analysis (FTA)
  - Event Tree Analysis (ETA)

Failure Mode Effect Analysis (FMEA)

- Step 1: list all possible failures
- Step 2: For each failure, list possible causes and consequences
  - Causes have causes
  - Consequences have consequences
  - Results in tree:



- Example: FMEA for Yacht Autopilot
  - Cause: Wind sensor imprecise
    - ⇨ Failure: Wrong wind speed
      - ⇨ Consequence: Wrong drift calculation

Failure Mode Effect Analysis (FMEA)

- Step 3: transform tree to table
  - Risk priority number = Probability × Severity × Detectability

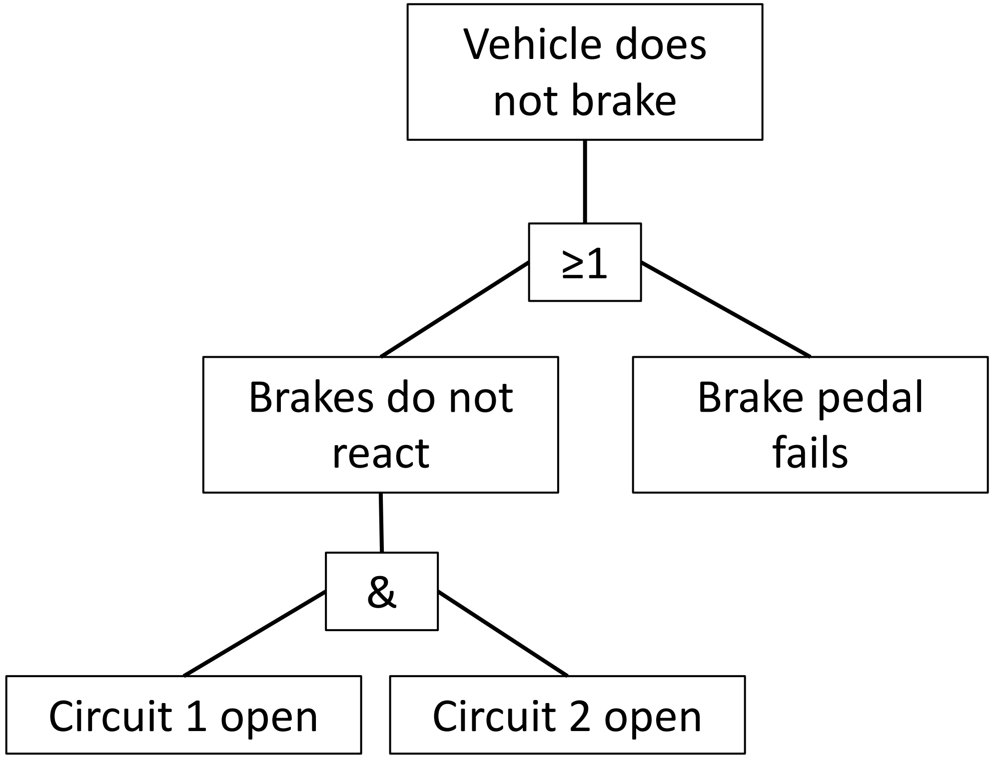
FMEA for Yacht Autopilot		Innsbruck, 2000-04-01			Rating				
Function	**Component**	**Failure**	**Consequence**	**Cause**	**P**	**S**	**D**	**RPN**	**Measures**
Lead ship from A to B	Determine position	Wrong position	Wrong course	Solar storms	1	5	1	5	
				GPS switched off	1	10	9	90	
				GPS precision reduced	1	5	9	45	
				GPS defective	1	10	9	90	
				GPS satellite defective	1	5	9	45	
	Determine wind speed	Wrong wind speed	Wrong drift calculation	Wind sensor imprecise	5	5	1	25	
			Wrong skipper warning	Wind changing too fast	10	9	5	450	
				Wind speed too low	10	9	5	450	
				Wind sensor gone	1	9	9	81	

Adapted from Kai Borgeest: “Elektronik in der Fahrzeugtechnik Hardware, Software, Systeme und Projektmanagement“ Vieweg/Springer, 2008

Fault Tree Analysis (FTA)

- DIN 25424-1,2
- Tree structure of causes
  - i.e., left half of FMEA
- Failures depend on causes
- Leafs: elementary causes
  (those that do not stem from other causes)
- Connect with logical OR/AND
  - Logical OR (if one cause suffices)
    e.g., brake pedal fails ⇨ brake fails (even if other components ok)
  - Logical AND
    e.g., dual circuit brake (both circuits have to fail)

Fault Tree Analysis (FTA)



Adapted from Kai Borgeest: “Elektronik in der Fahrzeugtechnik Hardware, Software, Systeme und Projektmanagement“ Vieweg/Springer, 2008

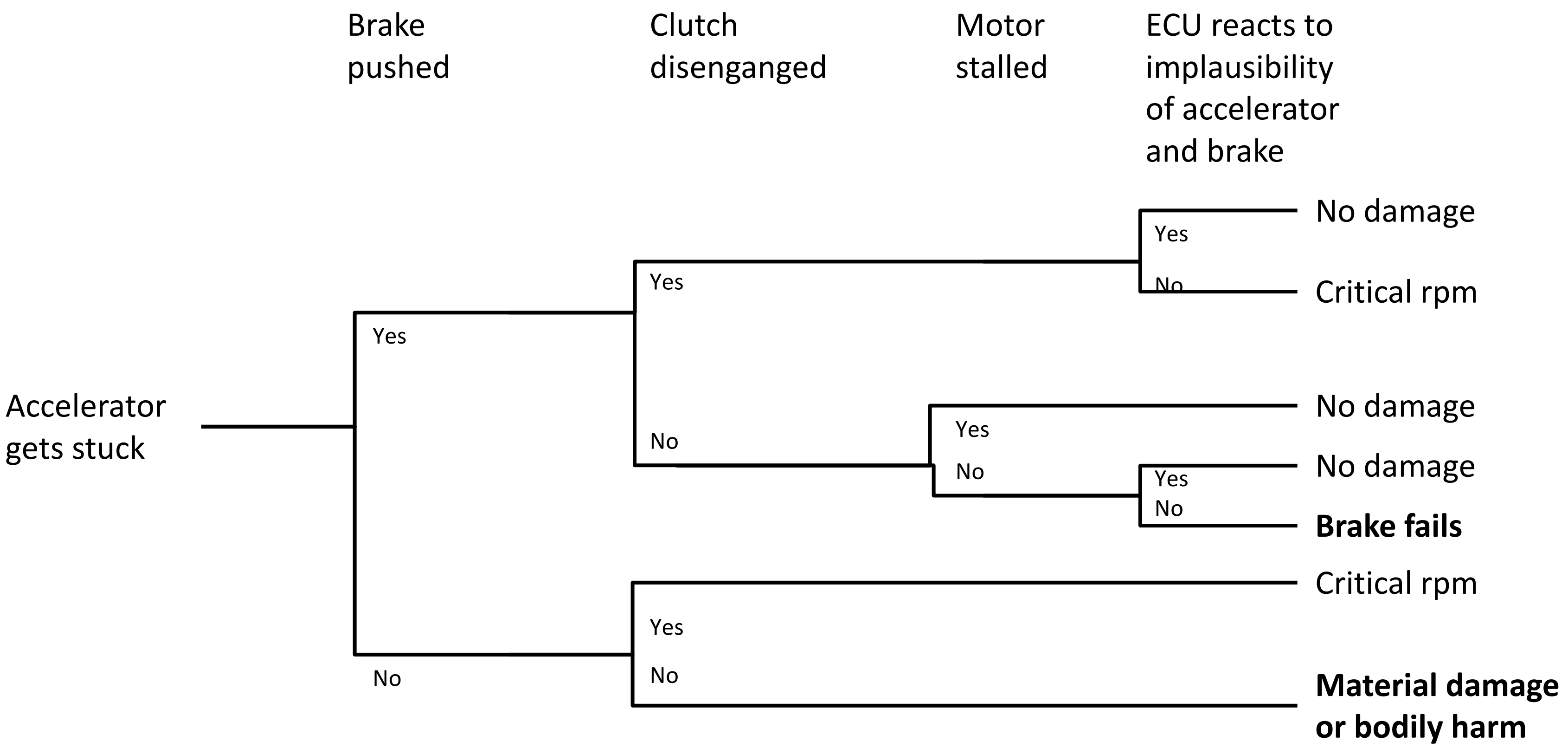
Fault Tree Analysis (FTA)

- Qualitative representation as tree
- Quantitative calculation:
  - OR for exclusive events:
    - $p(c) = p(a) + p(b)$
  - OR for arbitrary events:
    - $p(c) = p(a) + p(b) - p(a \times \mathrm{b})$
  - AND for independent events:
    - $p(c) = p(a) \times p(b)$
  - AND for dependent events:
    - $p(c) = p(a) + p(b|a)$
  - Difficulty: How probable are elementary events?

Event Tree Analysis (ETA)

- Analyzes consequences of faults
  (even if safety measures do not trigger)
- Process:
  - Start with individual fault
  - Fork, depending on which safety measures trigger
  - Multiple endings if more than one safety measure is in place
  - Results in tree of possible consequences
  - Limited quantitative analysis: hard to assign numbers to forks

Event Tree Analysis (ETA)



Adapted from Kai Borgeest: “Elektronik in der Fahrzeugtechnik Hardware, Software, Systeme und Projektmanagement“ Vieweg/Springer, 2008

Main Takeaways

- Aspects of Safety
    - Motivation
    - Terminology
    - Failure Mode Effect Analysis (FMEA)
    - Fault Tree Analysis (FTA)
    - Event Tree Analysis (ETA)
    - Commonalities and differences

UNIVERSITÀ DI PARMA

MUNER MOTORVEHICLE UNIVERSITY OF EMILIA-ROMAGNA

thank you

danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
Баярлалаа
спасибо
mersi
kia ora
barka
welalin
tack
spas
faafetai lava
vinaka
blagodaram
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
kiitos
dankie
спасибі
gracias
хвала
nanni
nandri
dhanyavad
mauruuru
köszönöm
dankon
akun
aĉiū
asante
manana
obrigada
tenki
murakoze
bedankt
enkosi
bayarlalaa
gràçie
hvala
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
go raibh maith agat
sobodi
děkuji
obrigado
sagolun
sukriya
kop khun krap
arigatô
takk
dakujem
trugarez
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
rahmet
kam sah hamnida
rahmat
terima kasih
diolch
dhanyavadagalu
shukriya
mercé
мерси
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci