# Controller Area Network

*Marco Di Natale*
*Scuola Superiore S. Anna- Pisa, Italy*

## CAN bus

# Controller Area Network

- Publicly available standard [1]
  http://www.semiconductors.bosch.de/pdf/can2spec.pdf

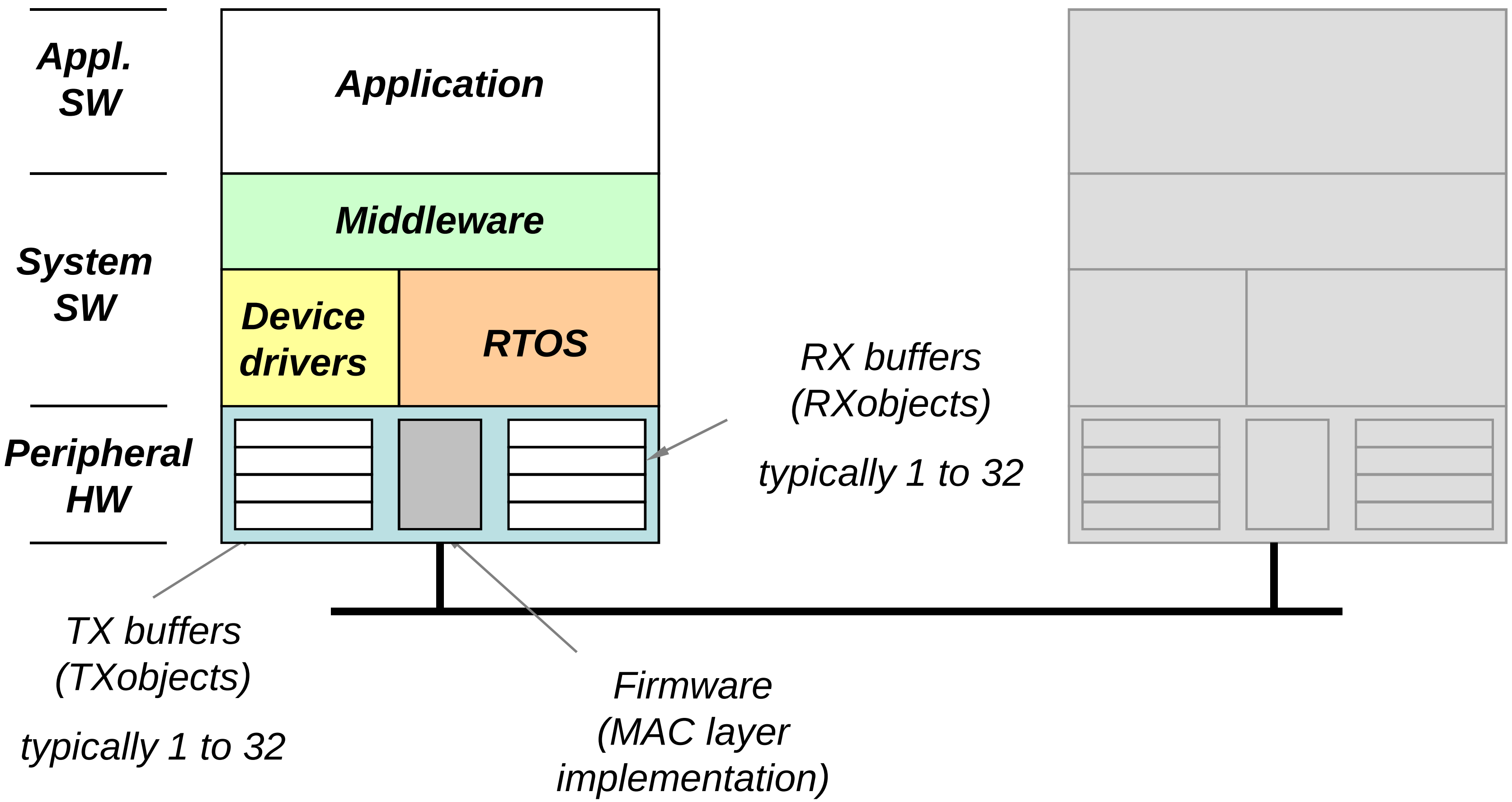
# Serial data bus developed by Bosch in the 80s

- Support for broadcast and multicast comm
- Low cost
- Deterministic resolution of the contention
- Priority-based arbitration
- Automotive standard but used also in automation, factory control, avionics and medical equipment
- Simple, 2 differential (copper) wire connection
- Speed of up to 1Mb/s
- Error detection and signalling

# Purpose of this Lesson

- Yet another communication protocol standard ?
- Develop time analysis for real-time messages
- Study the effect on timing of multiple layers (HW and SW)
- Understand how firmware can affect the time determinism and spoil the priority assignment
- Understand how device drivers and middleware layers influence the timing behavior
- Present multiple views for the time analysis (worst-case, stochastic, simulation-based)

# A CAN-based system



## CAN bus

# CAN standard (MAC protocol)

- Fixed format messages with limited size
- CAN communication does not require node (or system) configuration information (addresses)
  - Flexibility – a node can be added at any time
  - Message delivery and routing – the content is identified by an IDENTIFIER field defining the message content
  - Multicast – all messages are received by all nodes that can filter messages based on their IDs
  - Data Consistency – A message is accepted by all nodes or by no node

# Frame types

## DATA FRAME

- Carries regular data

## REMOTE FRAME

- Used to request the transmission of a DATA FRAME with the same ID

## ERROR FRAME

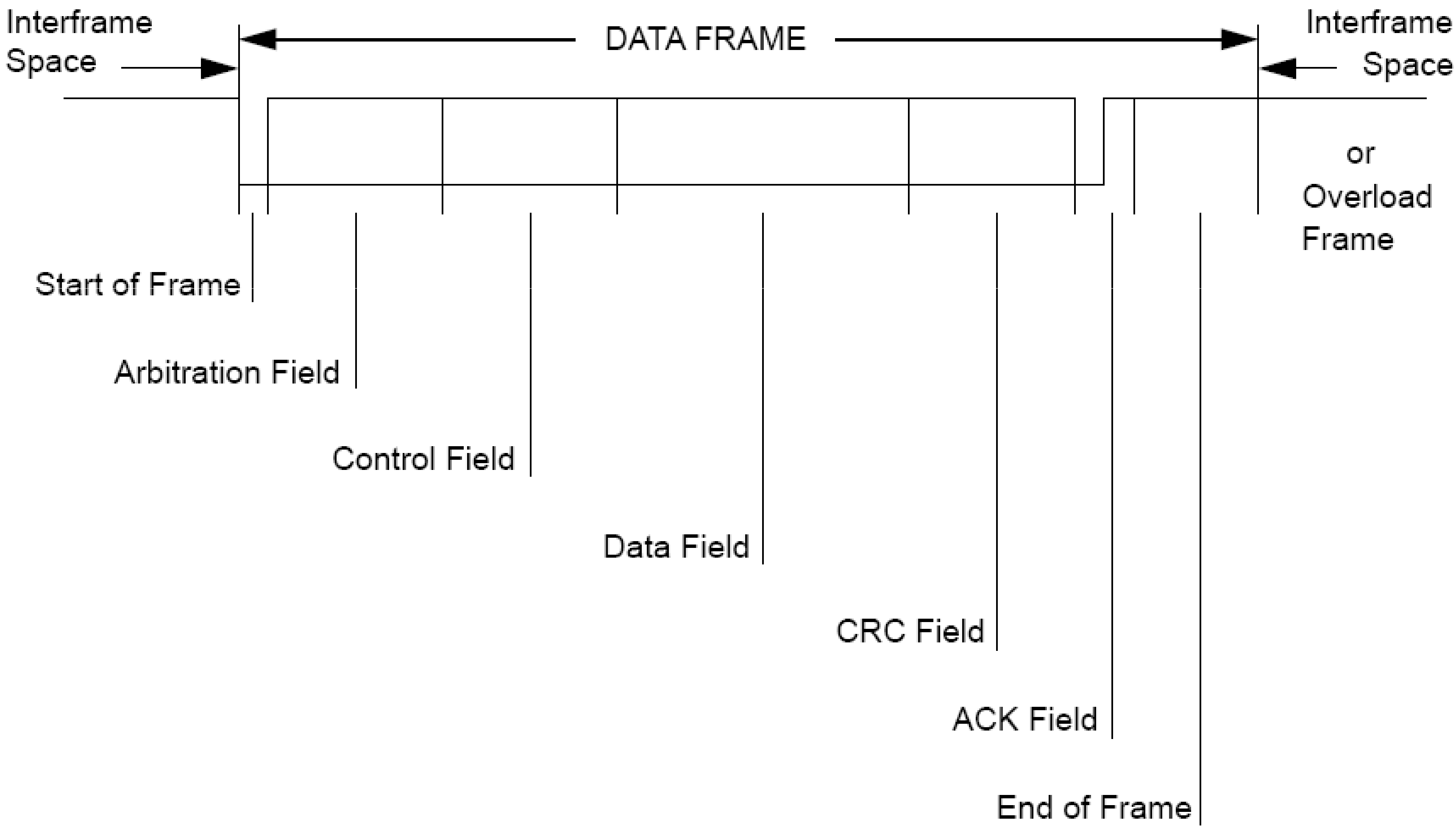
- Transmitted by any unit detecting a bus error

## OVERLOAD FRAME

- Used to force a time interval in between frame transmissions

# CAN bus

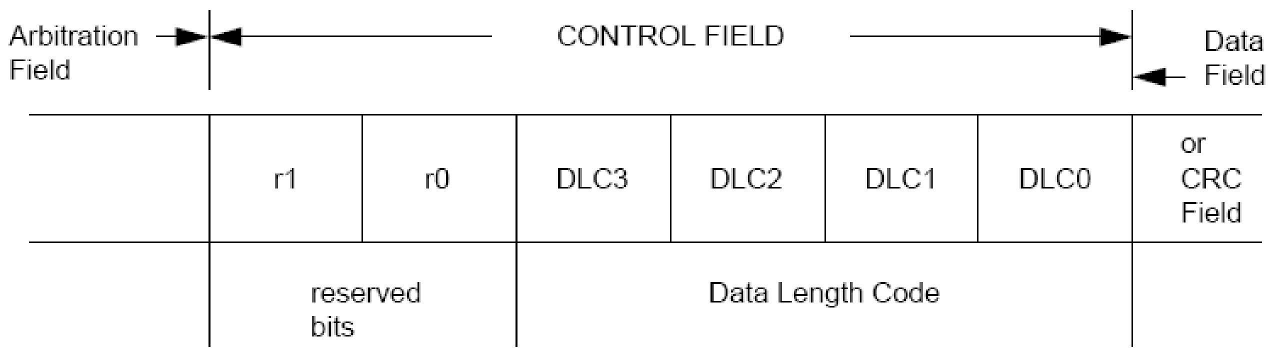
## DATA FRAME



# CAN bus

## DATA FRAME

- *Start of frame* – 1 dominant bit. A frame can only start when the bus is IDLE. All stations synchronize to the leading edge of the SOF bit
- *Identifier* – 11 (or 29 in version 2.0) bits. In order from most significant to least significant. The 7 most significant bits cannot be all recessive
- *RTR* – remote transmission request, dominant for REQUEST frames, recessive for DATA frames
- *CONTROL* – (see figure) maximum data length is 8 (bytes) other values are not used



# CAN bus

## DATA FRAME (conitinued)

*Data* – 0 to 8 bytes of data

*CRC* – 15 CRC bits plus one CRC delimiter bit (recessive)

*ACK* – two bits (SLOT + DELIMITER) all stations receiving the message correctly (CRC check) set the SLOT to dominant (the transmitter transmits a recessive). The DELIMITER is recessive

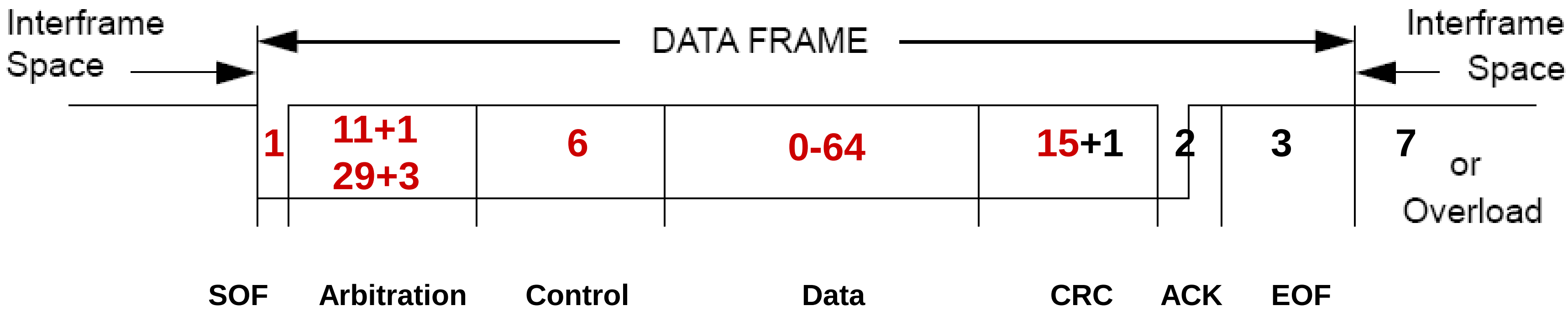
*END OF FRAME* – seven recessive bits

## Bit stuffing

any sequence of 5 bits of the same type requires the addition of an opposite type bit by the TRANSMITTER (and removal from the receiver)

# CAN bus

Some considerations …




*Worst case frame length*

34 bits subject to stuffing

$$64 + \lfloor (64 + 34)/4 \rfloor + 47 = 111 + 24 = \mathbf{135}$$

*Protocol overhead*

(minimum with no stuffing)

64/111 = 0.576 data efficiency (73.4% protocol overhead)

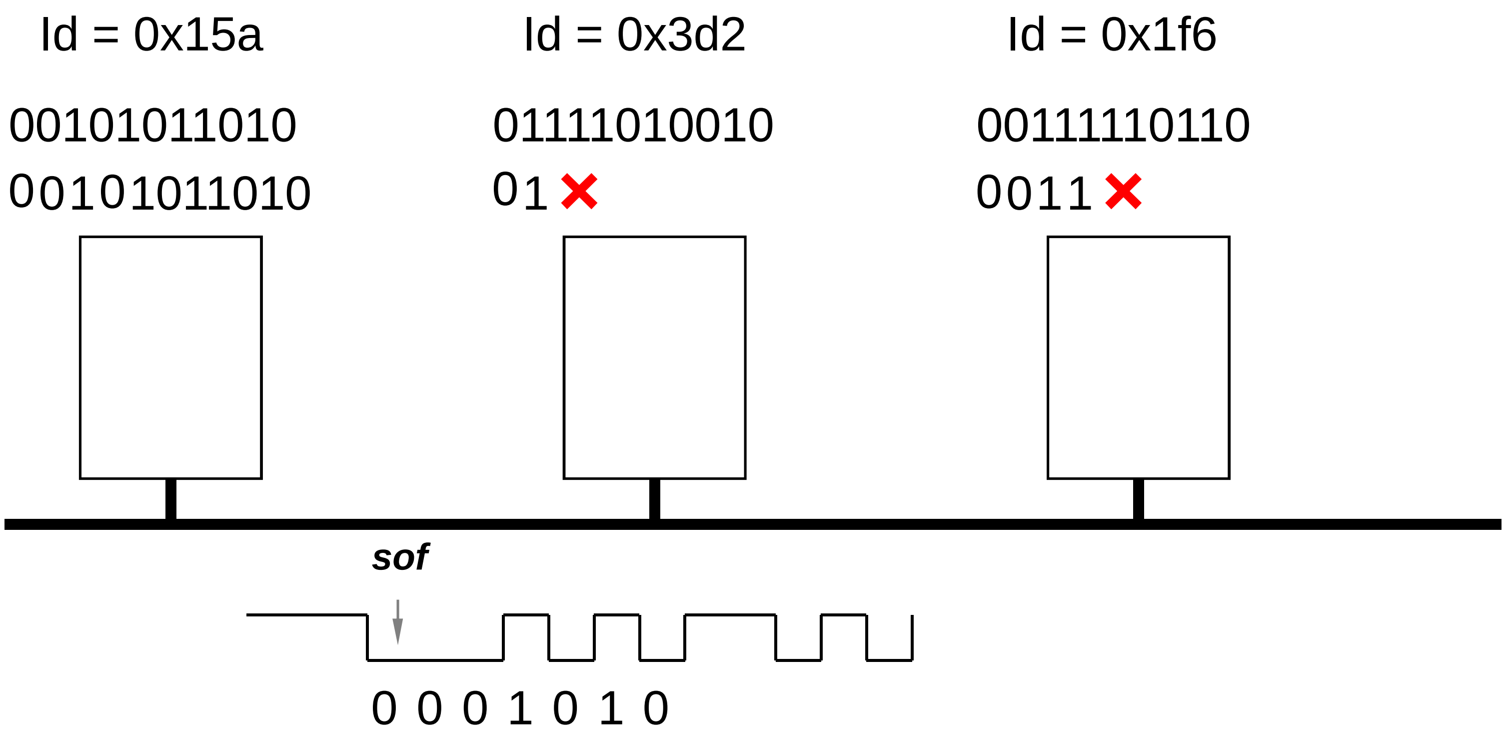
# CAN bus

## Arbitration

*All nodes are synchronized on the SOF bit*

*The bus behaves as a wired-AND (wired-OR)*

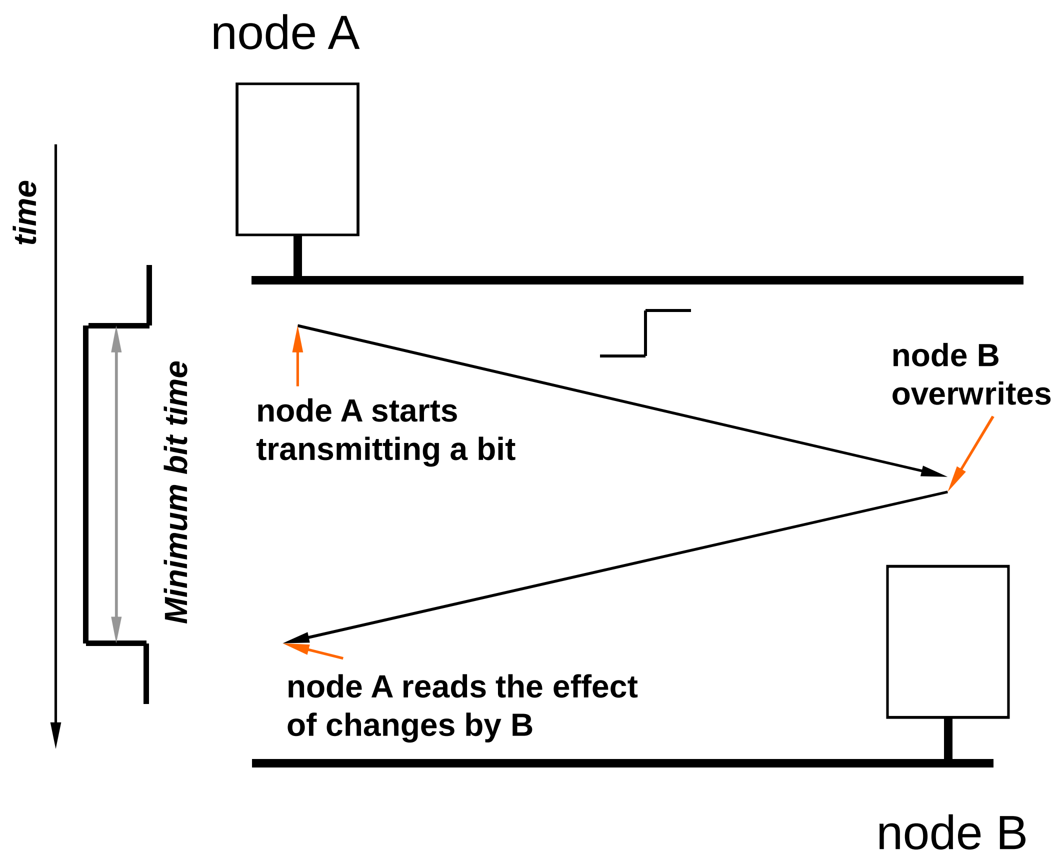
*An example …*



# CAN bus

The type of arbitration implies that the bit time is at least twice the propagation latency on the bus

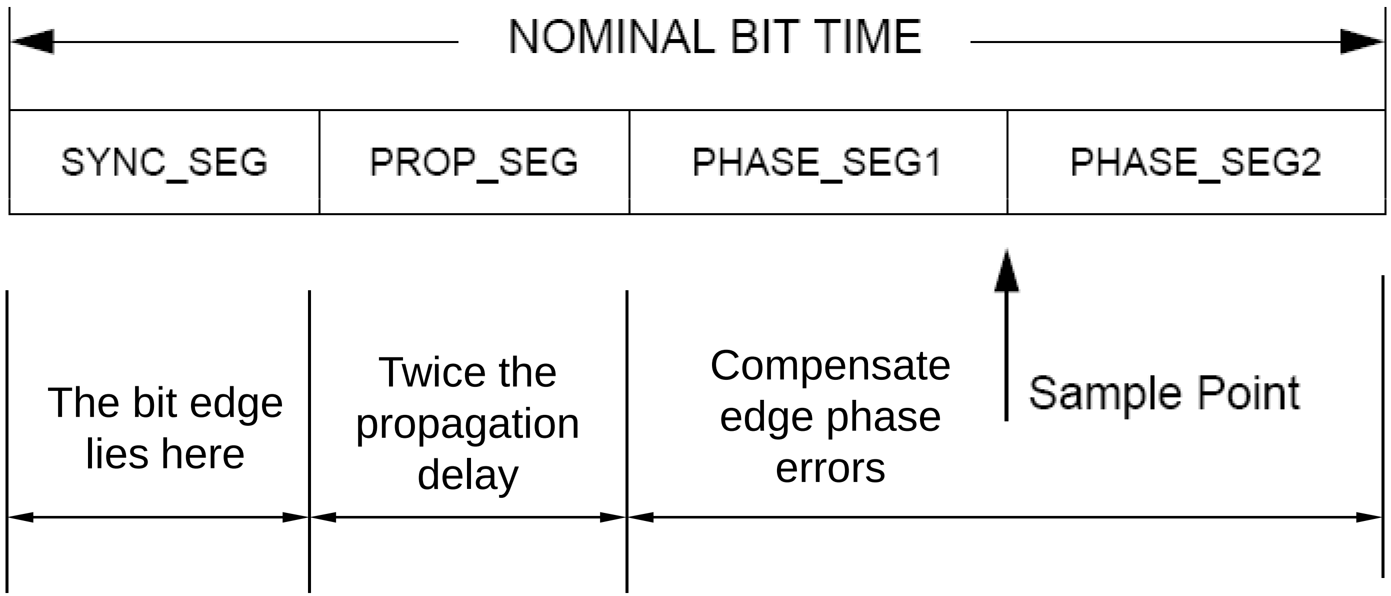
This defines a relation between the maximum bus length and the transmission speed. The available values are




| Bit rate | Bus length |
|---|---|
| 1 Mbit/s | 25 m |
| 800 kbit/s | 50 m |
| **500 kbit/s** | **100 m** |
| 250 kbit/s | 250 m |
| 125 kbit/s | 500 m |
| 50 kbit/s | 1000 m |
| 20 kbit/s | 2500 m |
| 10 kbit/s | 5000 m |

# CAN bus

## Bit time



## CAN bus

Error and fault containment

There are 5 types of error

BIT ERROR

- The sender monitors the bus. If the value found on the bus is different from the one that is sent, then a BIT ERROR is detected

STUFF ERROR

- Detected if 6 consecutive bits of the same type are found

CRC ERROR

- Detected by the receiver if the received CRC field does not match the computed value

FORM ERROR

- Detected when a fixed format field contains unexpected values

ACKNOWLEDGEMENT ERROR

- Detected by the transmitter if a dominant value is not found in the ack slot

## CAN bus

A station detecting an error transmits an ERROR FLAG.

For BIT, STUFF, FORM, ACKNOWLEDGEMENT errors, it is sent in the immediately following bit.

For CRC it is sent after the ACK DELIMITER

The ERROR FLAG is part of an ERROR FRAME

# CAN bus

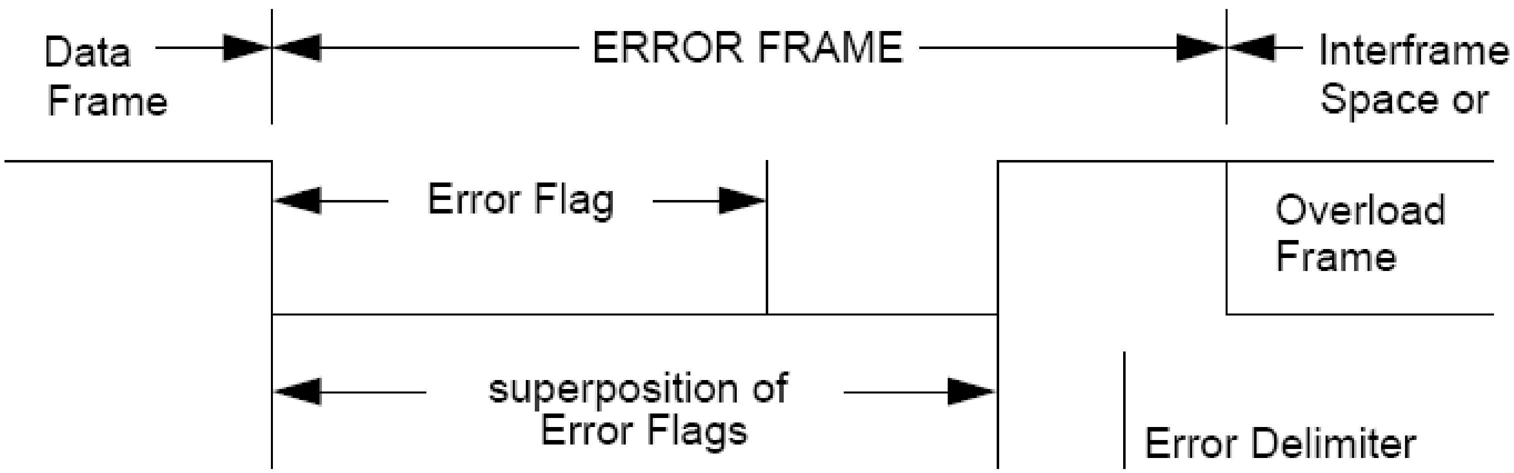
An ERROR FRAME is simply the superposition of ERROR FLAGS from different nodes, plus an ERROR DELIMITER

There are two types of error flags:

- An ACTIVE ERROR flag consists of 6 consecutive dominant bits
- A PASSIVE ERROR flag consists of 6 consecutive recessive bits

The superposition of all the error flags goes from 6 to 12 bits

The error delimiter consists of 8 recessive bits



## CAN bus

### Fault containment

### Each node can be in 3 states:

- Error active
- Error passive: limited error signalling and transmission features
- Bus off: cannot influence the bus

### Each node has two counters:

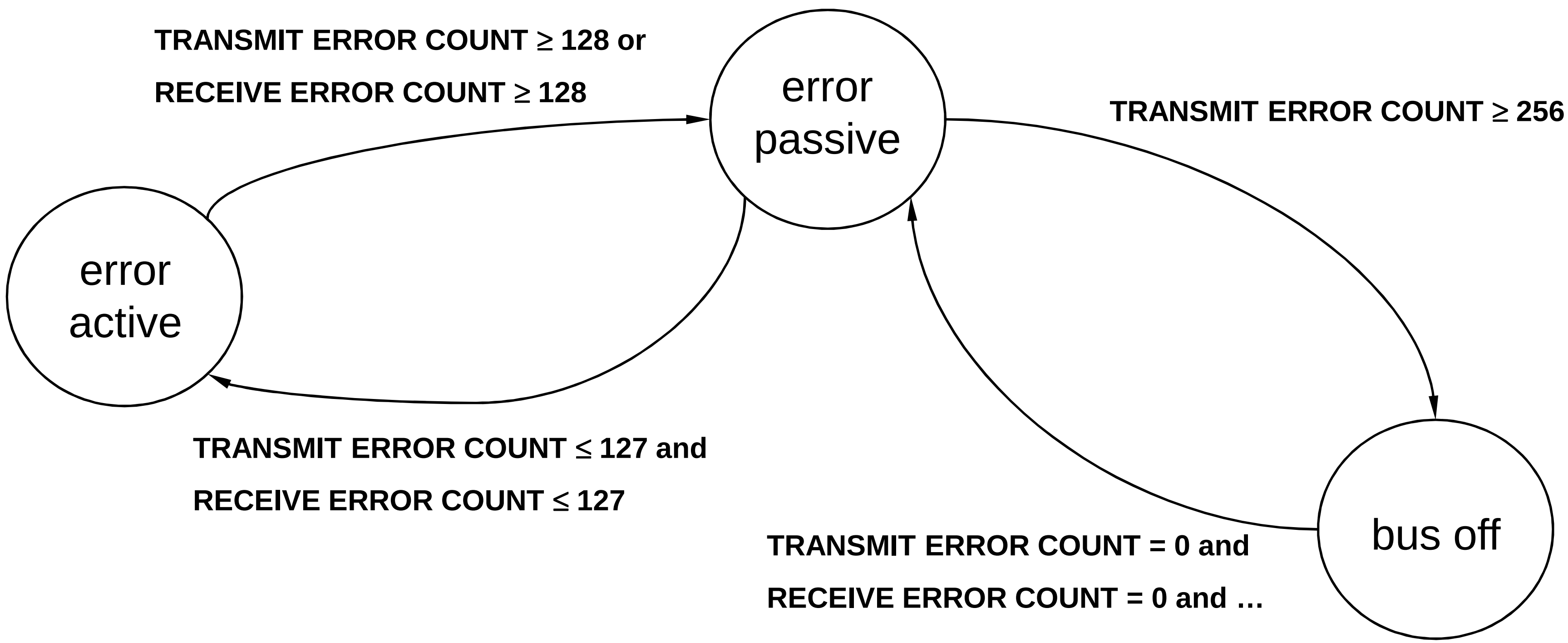
- TRANSMIT ERROR COUNT:
  - increased – (list) by 8 when the transmitter detects an error …
  - decreased – by 1 after the successful transmission of a message (unless it is 0)
- RECEIVE ERROR COUNT:
  - increased – (list) by 1 when the node detects an error, by 8 if it detects a dominant bit as the first bit after sending an error flag …
  - decreased – (if between 1 and 127 by 1, if >127 set back to a value between 119 and 127) after successful reception of a message

# CAN bus

## Fault containment

## Each node can be in 3 states:

- Error active
- Error passive: limited error signalling and transmission features
- Bus off: cannot influence the bus



# CAN bus

Error detection

Possible problems on the last but one bit [7]

CAN misbehavior is possible because of the different error detection mechanisms at the transmitter and receiver sites

A message is valid for the transmitter is there is no error until the end of the frame

A message is valid for the receiver is there is no error until the last but one bit of the frame (last bit is do not care)

If the receiver accepts the message, it may have an inconsistent message duplicate

- Use of sequence numbers fixes the duplicate error
- …but does not prevent messages from being received in different orders

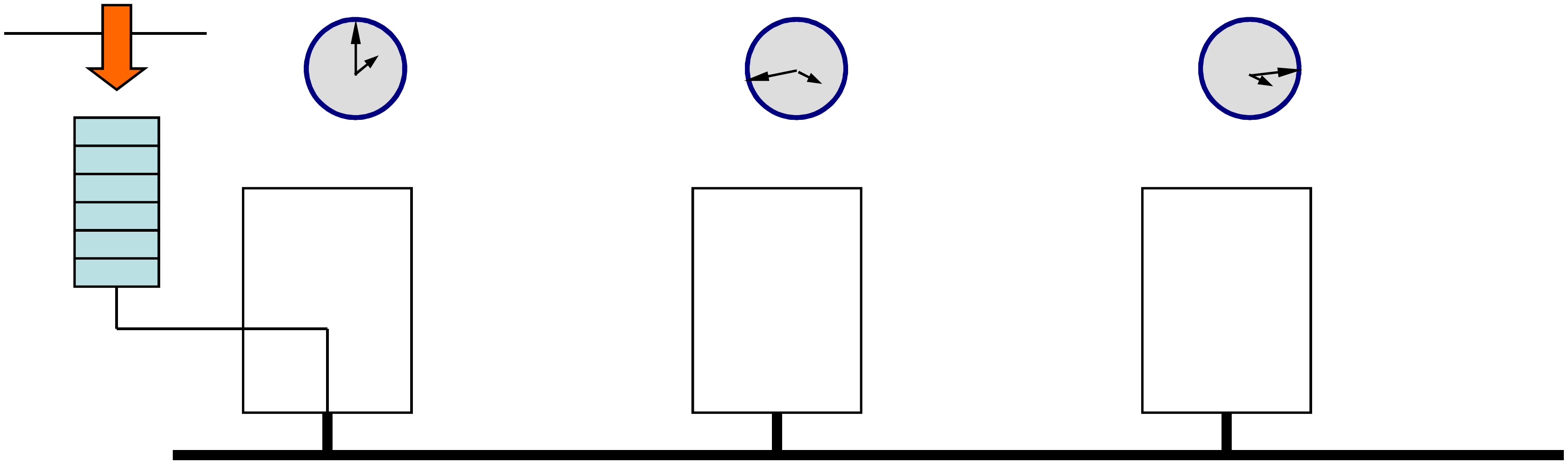
If the sender fails before retransmitting there may be an inconsistent message omission …

# CAN bus

## Timing Analysis (and inversions) – Ideal behavior

Assumption 3: periodic messages, but no assumption on the message phases

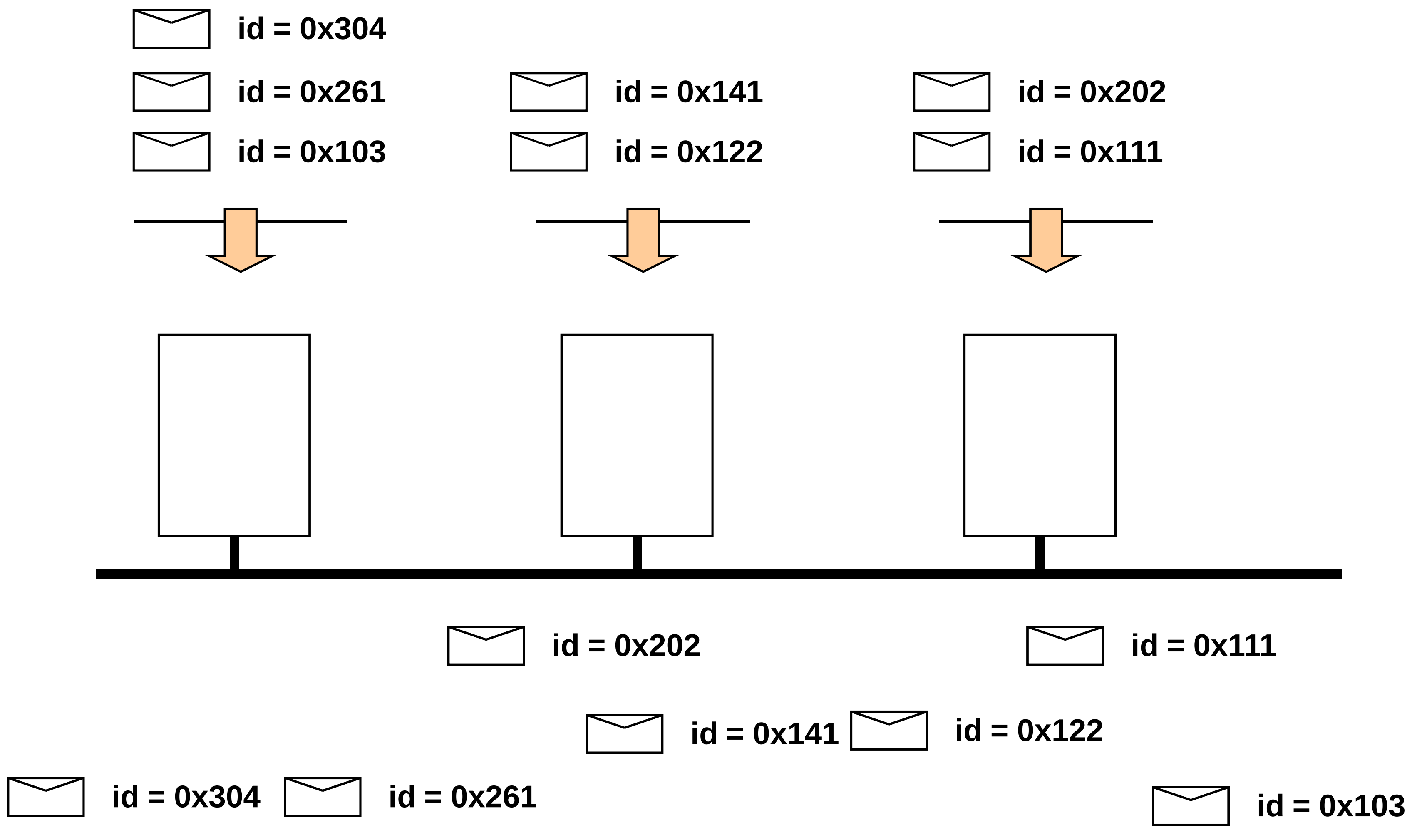
Assumption 1: nodes are not synchronized, nor any assumption on local clocks is used by the MW and driver levels



Assumption 2: messages are always transmitted by nodes based on their priority (ID) – ideal priority queue of messages

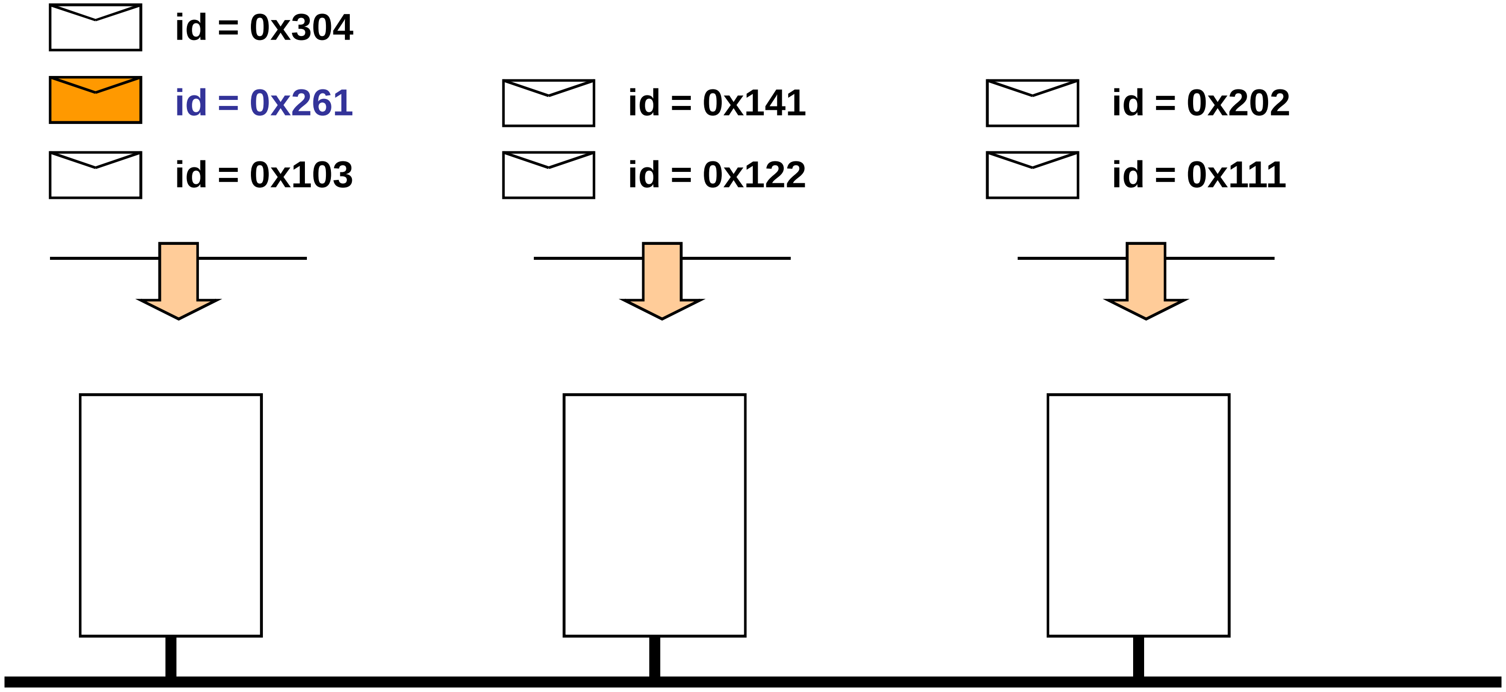
# CAN bus

## Timing Analysis (and inversions) – Ideal behavior



# CAN bus

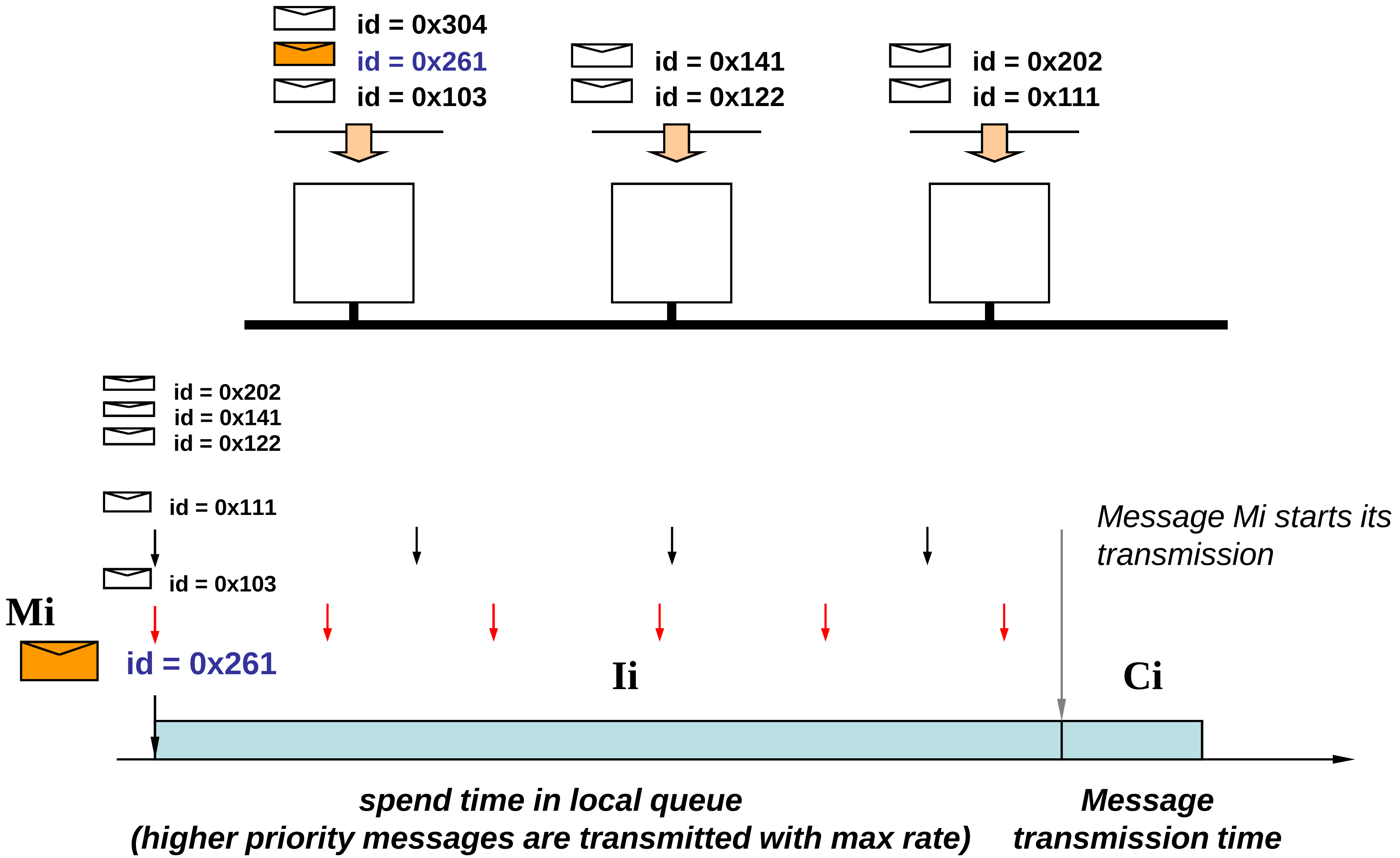
## Timing Analysis – worst case latency – Ideal behavior




Critical instant theorem: for a preemptive priority based scheduled resource, the worst case response time of an object occurs when it is released together with all other higher priority objects and they are released with their highest rate

# CAN bus

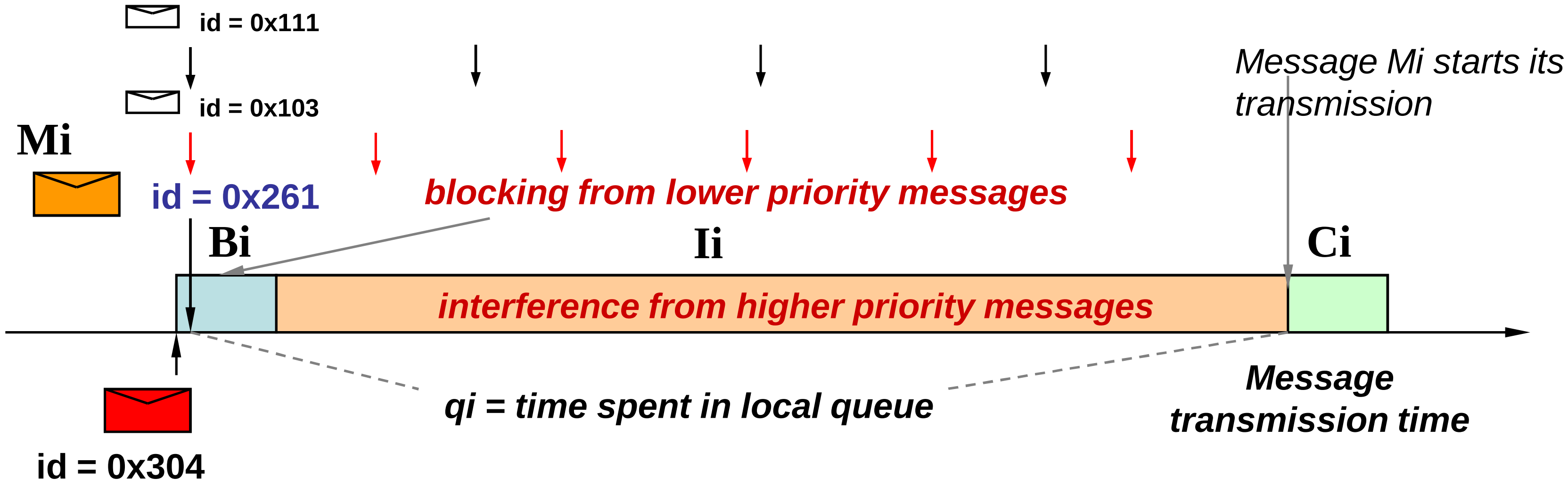
## Timing Analysis – worst case latency – Ideal behavior



## CAN bus

# Timing Analysis – worst case latency – Ideal behavior [2]

The transmission of a message cannot be preempted




$$q_i = B_i + I_i$$

$$w_i = q_i + C_i$$

$$I_i = \sum_{j \in hp(i)} I_{i,j}$$

$$I_{i,j} = \left\lceil \frac{q_i}{T_j} \right\rceil C_j$$

$$q_i = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i}{T_j} \right\rceil C_j$$

Fixed point formula: solved iteratively by setting qi(0)=0 until the minimum solution is found

## CAN bus

The worst case response time analysis has been (partly) refuted and revised in [9]

An example (SAE benchmark) for a 125 kb/s bus [3]

| Msg | Size | T | D | C | R |
|---|---|---|---|---|---|
| **1** | 1 | 50 | 5 | **0.52** | **1.44** |
| **2** | 2 | 5 | 5 | **0.60** | **2.04** |
| **3** | 1 | 5 | 5 | **0.52** | **2.56** |
| **4** | 2 | 5 | 5 | **0.60** | **3.16** |
| **5** | 1 | 5 | 5 | **0.52** | **3.68** |
| **6** | 2 | 5 | 5 | **0.60** | **4.28** |
| **7** | 6 | 10 | 10 | **0.92** | **7.88** |
| **8** | 1 | 10 | 10 | **0.52** | **8.4** |
| **9** | 2 | 10 | 10 | **0.60** | **9** |

| Msg | Size | T | D | C | R |
|---|---|---|---|---|---|
| ***10*** | *3* | *10* | *10* | ***0.68*** | ***9.68*** |
| **11** | 1 | 50 | 20 | **0.52** | **18.6** |
| **12** | 4 | 100 | 100 | **0.76** | **19.28** |
| **13** | 1 | 100 | 100 | **0.52** | **19.8** |
| **14** | 1 | 100 | 100 | **0.52** | **29.24** |
| **15** | 3 | 1000 | 1000 | **0.68** | **29.76** |
| **16** | 1 | 1000 | 1000 | **0.52** | **38.68** |
| **17** | 1 | 1000 | 1000 | **0.52** | **38.68** |

# CAN bus

In reality, this analysis can give optimistic results!

A number of issues need to be considered …

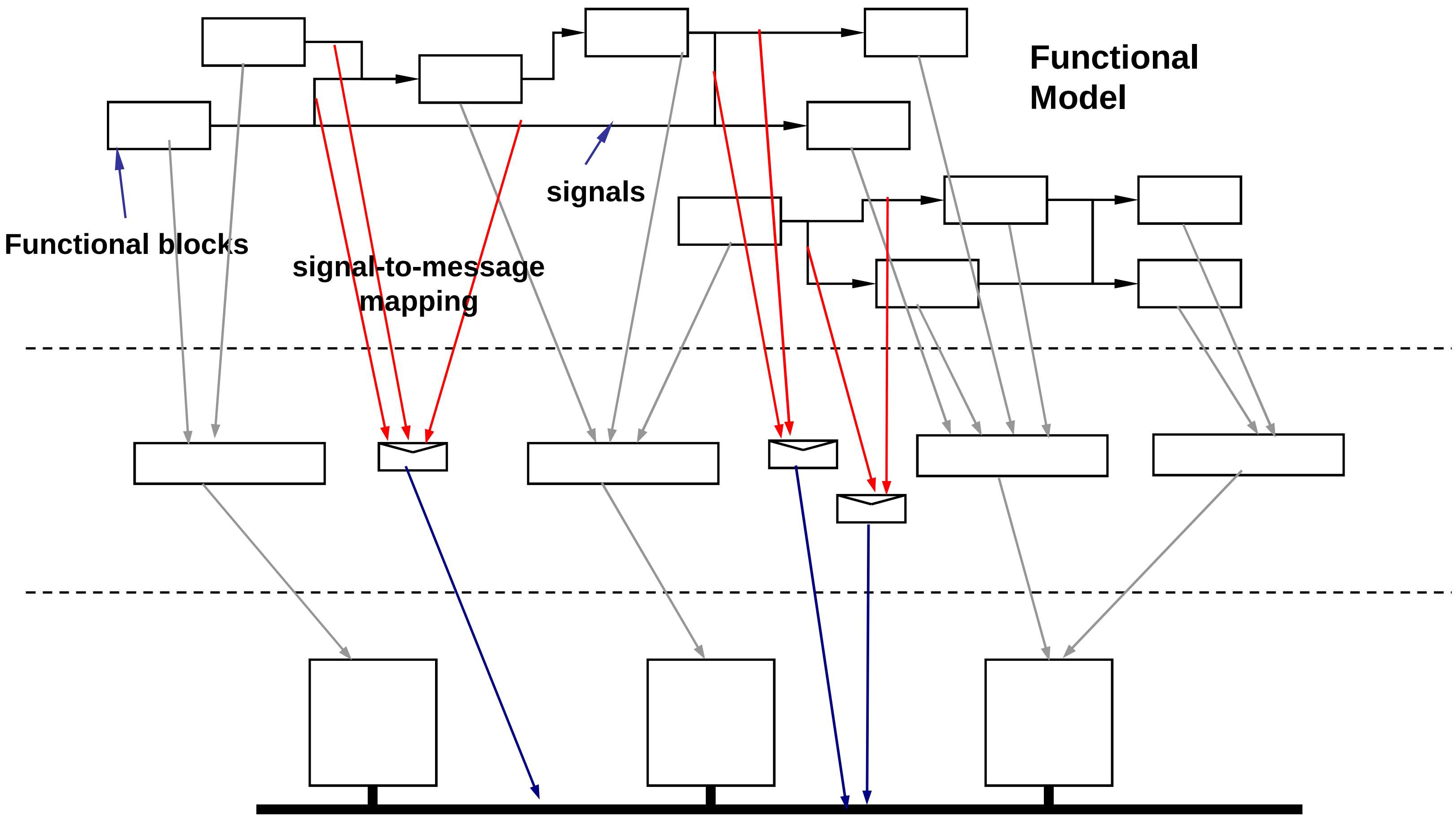
- Priority enqueuing in the sw layers
- Availability of TxObjects at the adapter
- Possibility of preempting (aborting) a transmission attempt
- Finite copy time between the queue and the TxObjects
- The adapter may not transmit messages in the TxObjects by priority

But first ….

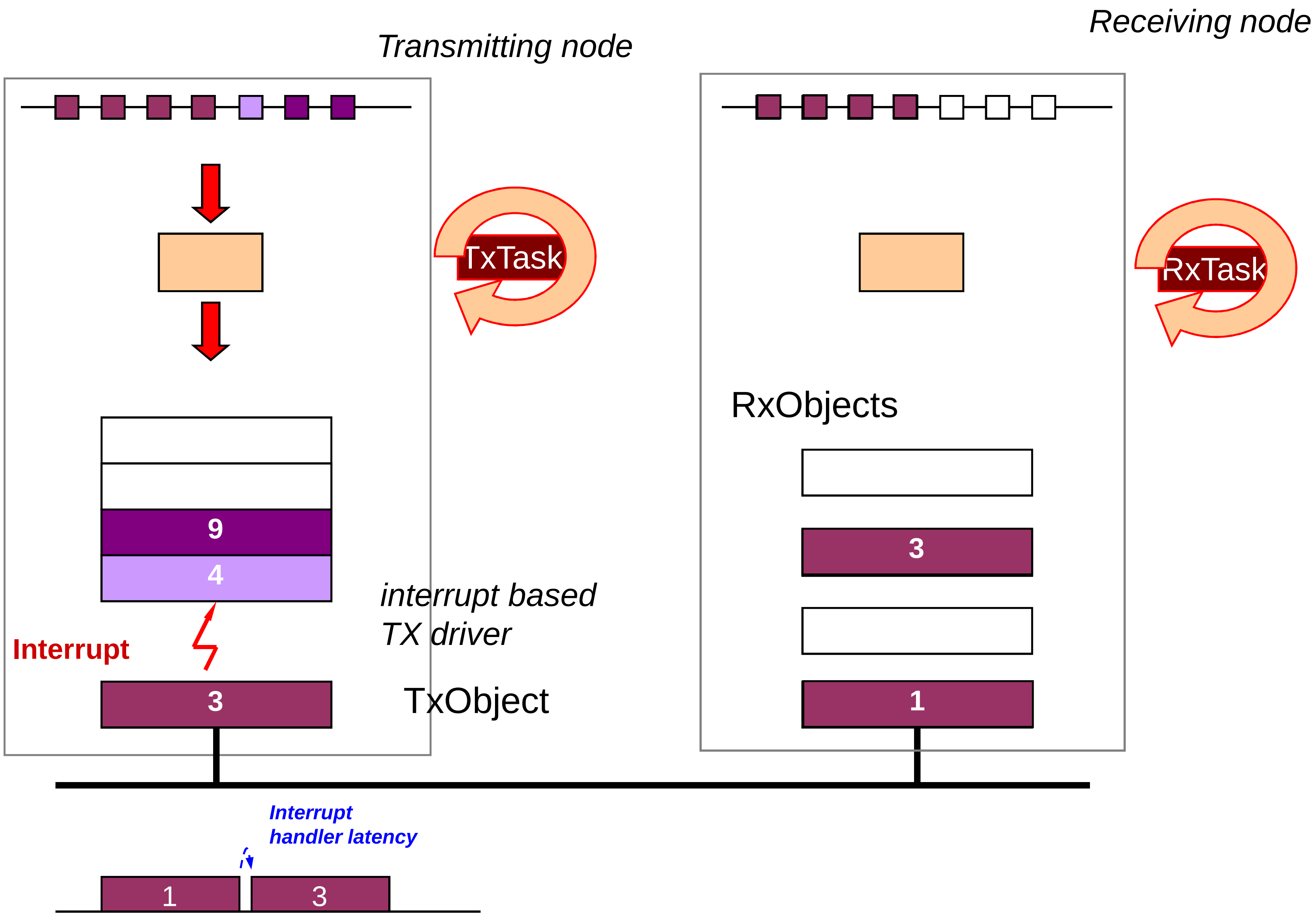
- *Let's examine the functional and architecture-level models and the MW, RTOS and driver management policies*

# CAN bus

Functional and architecture-level models and the MW, RTOS and driver management policies



# Transmission modes (1)



## CAN bus

In reality, this analysis can give optimistic results!

A number of issues need to be considered …

- Priority enqueuing in the sw layers
- …

If the messages are not enqueued by priority, additional priority inversion may occur. This may happen because of the way messages are enqueued in the SW layers (MW/drivers), for example if a FIFO queue is used

# CAN bus

In reality, this analysis can give optimistic results!

A number of issues need to be considered …

- …
- Availability of TxObjects at the adapter
- Finite copy time between the queue and the TxObjects

Adapters typically only have a limited number of TXObjects or RxObjects available

# CAN bus

A number of issues need to be considered …

- …
- Availability of TxObjects at the adapter

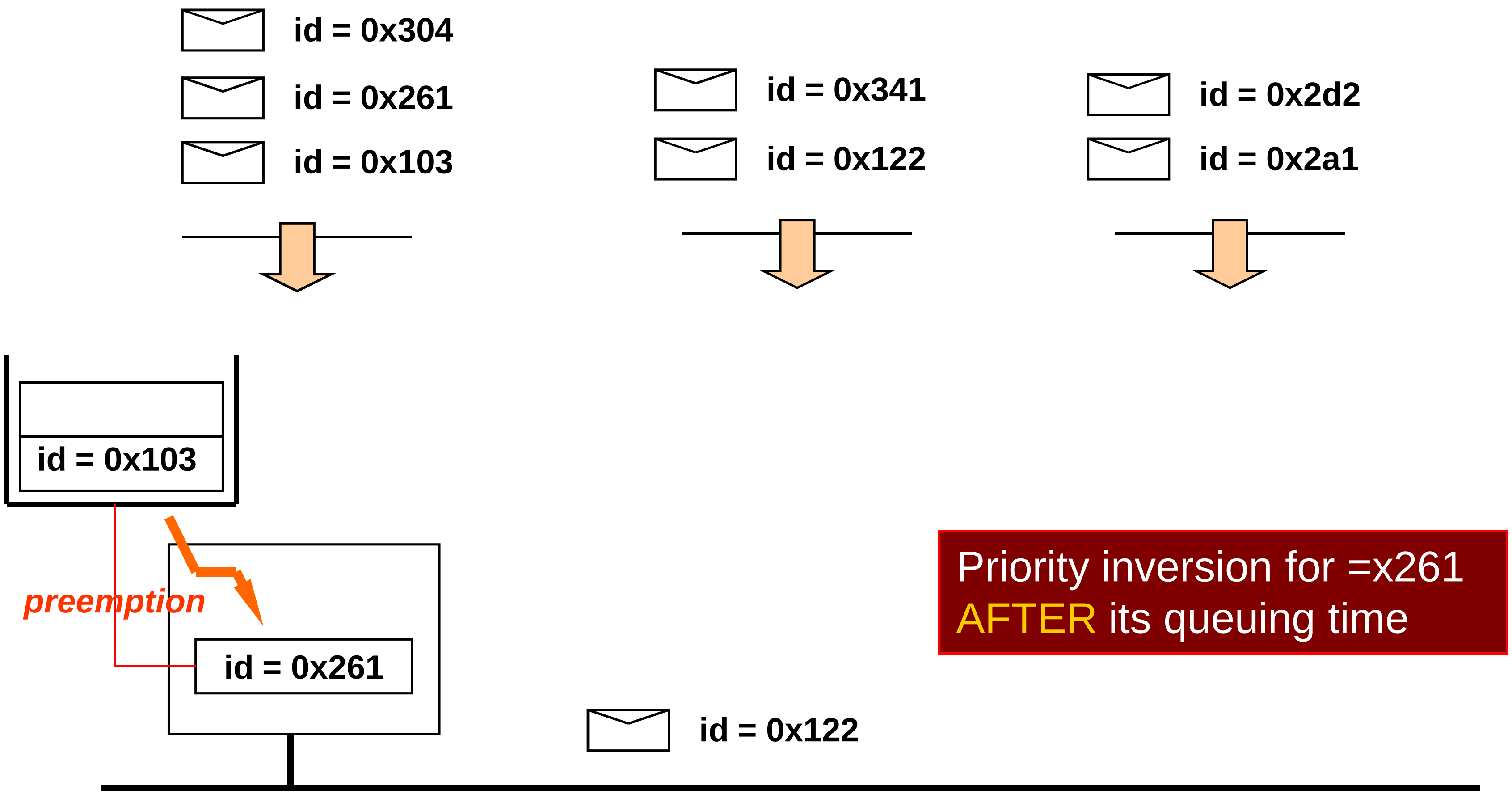
- Let's check the controller specifications!

| Model | Type | Buffer Type | Priority and Abort |
|---|---|---|---|
| Microchip MCP2515 | Standalone controller | 2 RX - 3 TX | lowest message ID, abort signal |
| ATMEL AT89C51CC03 AT90CAN32/64 | 8 bit MCU w. CAN controller | 15 TX/RX msg. objects | lowest message ID, abort signal |
| FUJITSU MB90385/90387 90V495 | 16 bit MCU w. CAN controller | 8 TX/RX msg. objects | lowest buffer num. abort signal |
| FUJITSU 90390 | 16 bit micro w. CAN controller | 16 TX/RX msg. objects | lowest buffer num. abort signal |
| Intel 87C196 (82527) | 16 bit MCU w. CAN controller | 14 TX/RX + 1 RX msg. objects | lowest buffer num. abort possible (?) |
| INFINEON XC161CJ/167 (82C900) | 16 bit MCU w. CAN controller | 32 TX/RX msg. objects (2 buses) | lowest buffer num., abort possible (?) |
| PHILIPS 8xC592 (SJA1000) | 8 bit MCU w. CAN controller | one TX buffer | abort signal |

# CAN bus

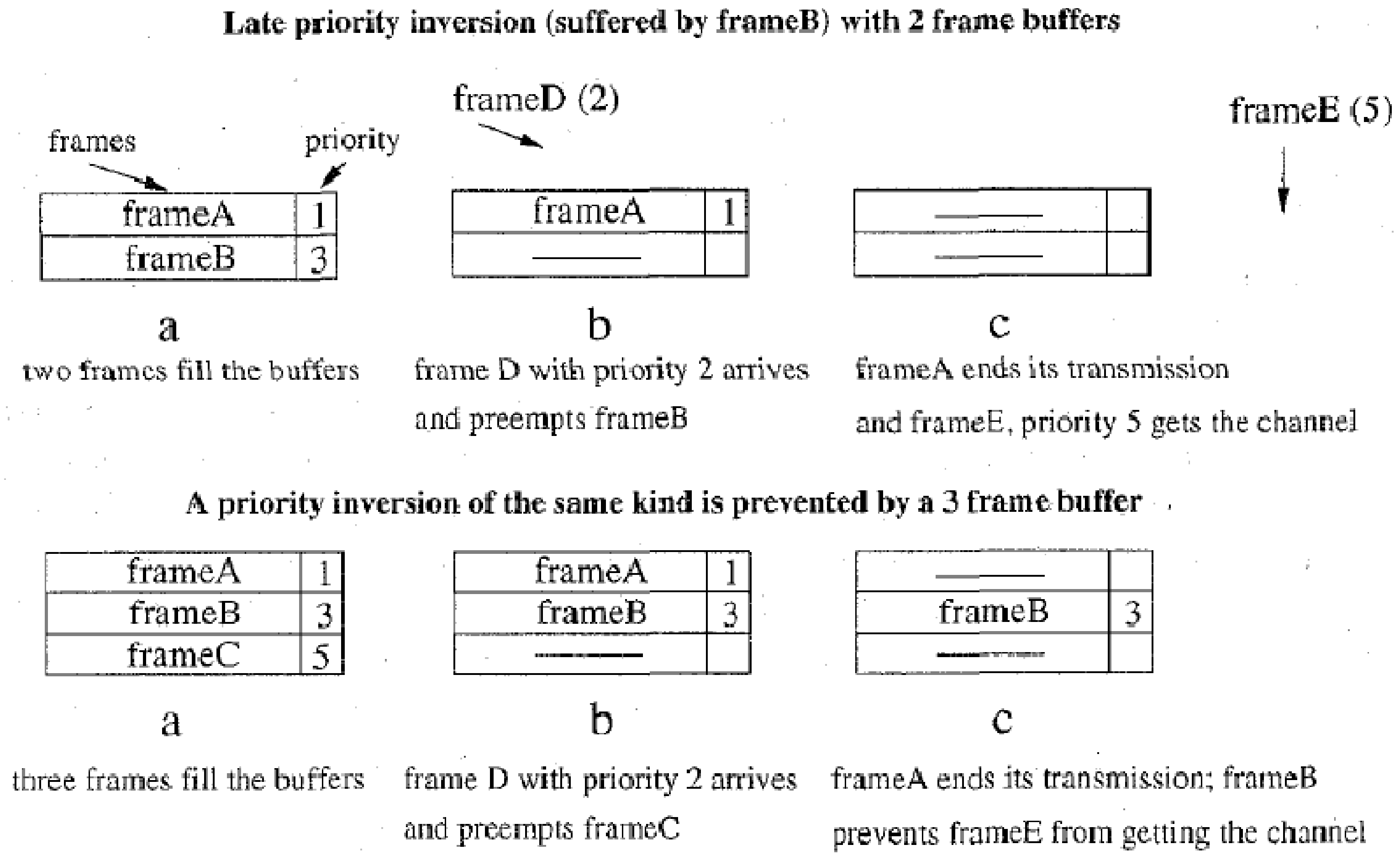
What happens if only one TxObject is available?

– Assuming preempatbility of TxObject



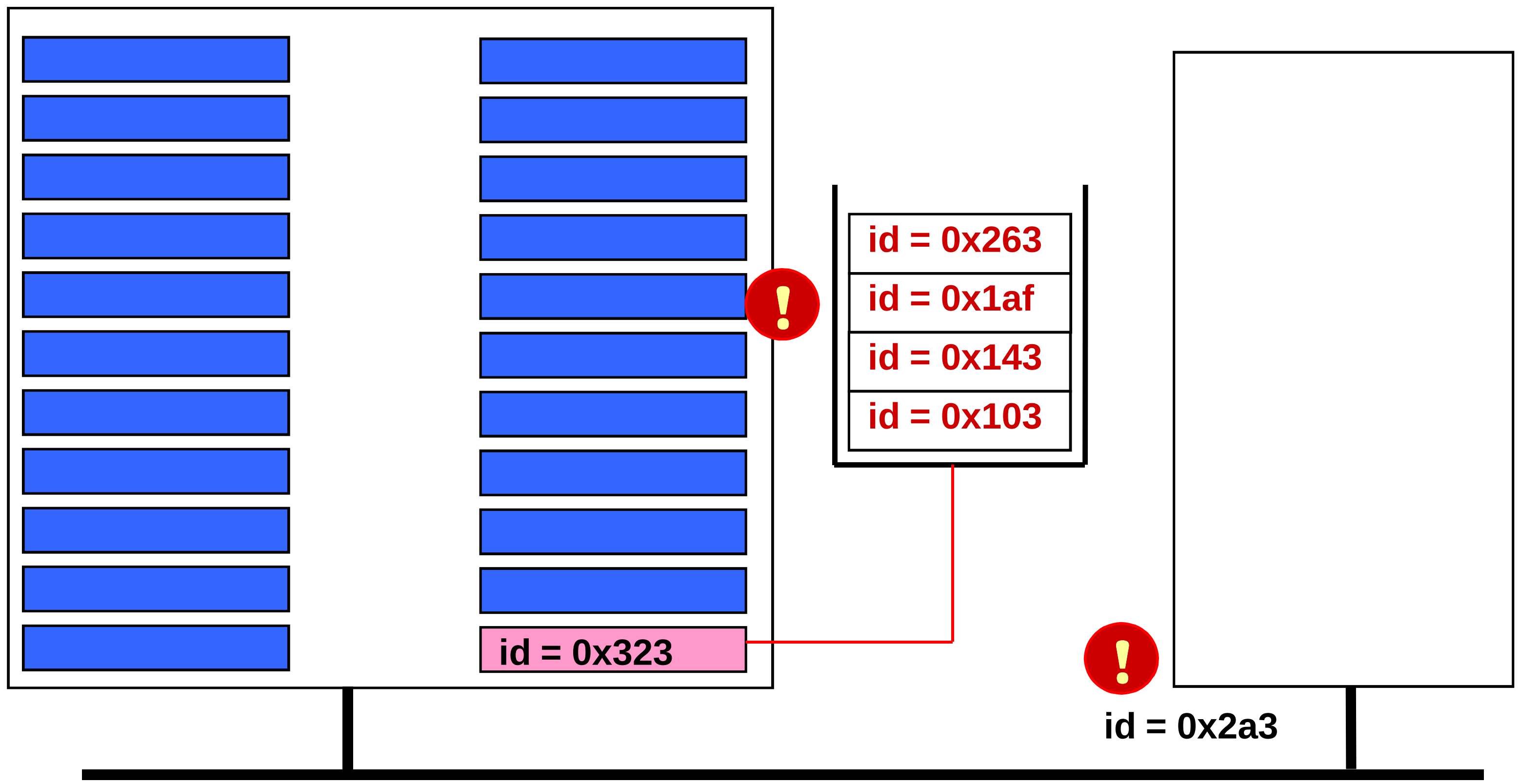
## CAN bus

What happens if two TxObjects are available?



## CAN bus

In reality, because of the polling-based management at the receiving side, designers prefer to use as many Objects as possible for the porpose of receiving messages and only one (or a very limited number) for message transmission !



# CAN bus

In reality, this analysis can give optimistic results!

A number of issues need to be considered …

- …
- Possibility of preempting (aborting) a transmission attempt

And the TxObjects are usually not preempted!

# CAN bus

A number of issues need to be considered …

- …
- The adapter may not transmit messages in the TxObjects by priority

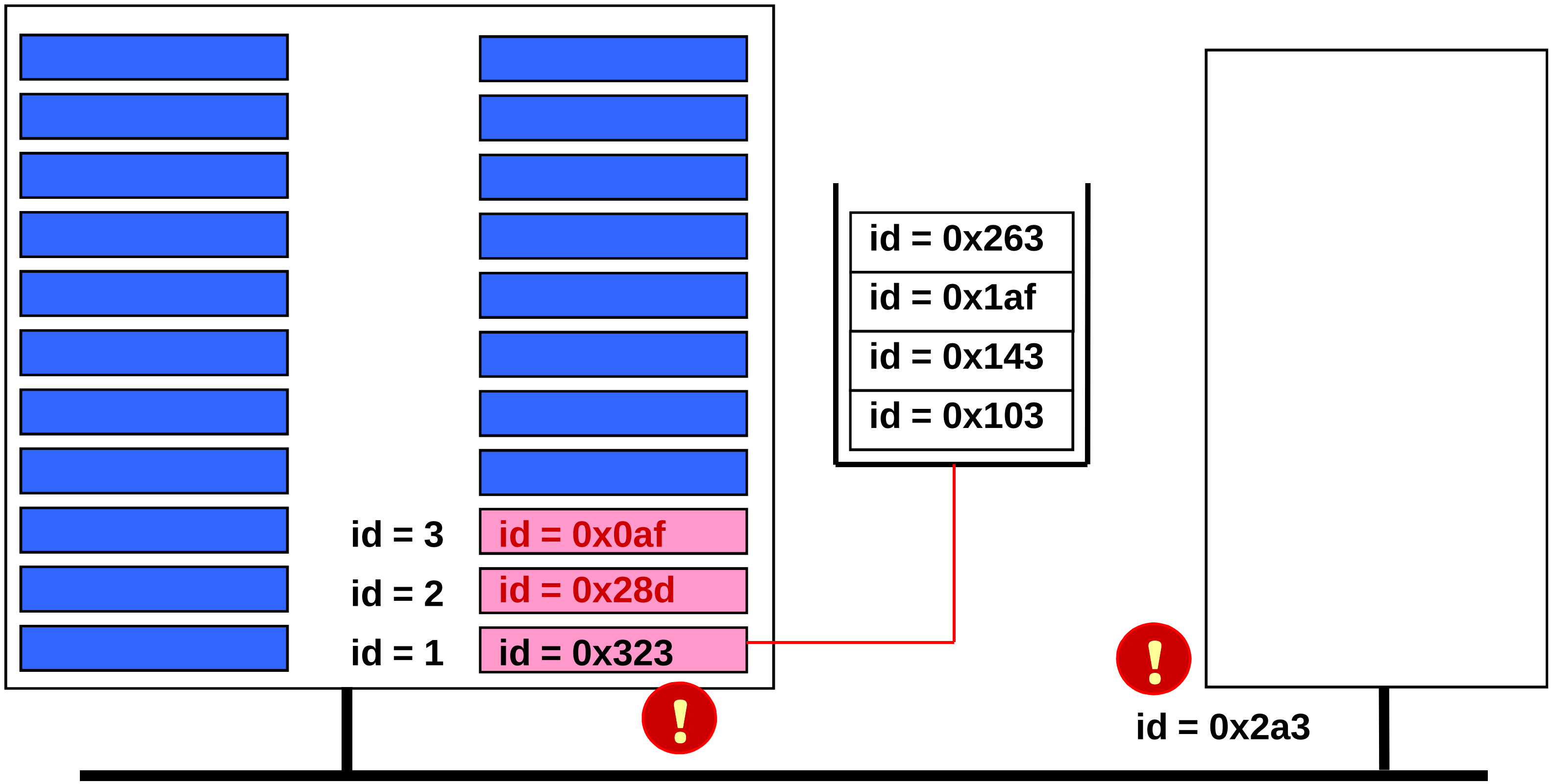
- Let’s check the controller specifications!

| Model | Type | Buffer Type | Priority and Abort |
|---|---|---|---|
| Microchip MCP2515 | Standalone controller | 2 RX - 3 TX | lowest message ID, abort signal |
| ATMEL AT89C51CC03 AT90CAN32/64 | 8 bit MCU w. CAN controller | 15 TX/RX msg. objects | lowest message ID, abort signal |
| FUJITSU MB90385/90387 90V495 | 16 bit MCU w. CAN controller | 8 TX/RX msg. objects | lowest buffer num. abort signal |
| FUJITSU 90390 | 16 bit micro w. CAN controller | 16 TX/RX msg. objects | lowest buffer num. abort signal |
| Intel 87C196 (82527) | 16 bit MCU w. CAN controller | 14 TX/RX + 1 RX msg. objects | lowest buffer num. abort possible (?) |
| INFINEON XC161CJ/167 (82C900) | 16 bit MCU w. CAN controller | 32 TX/RX msg. objects (2 buses) | lowest buffer num., abort possible (?) |
| PHILIPS 8xC592 (SJA1000) | 8 bit MCU w. CAN controller | one TX buffer | abort signal |

## CAN bus

In this case, especially if coupled with non-preemptability of TxObjects, the priority order of the queue may be completely subverted.

- Think of the problems in the implementation of a preemtpive policy!

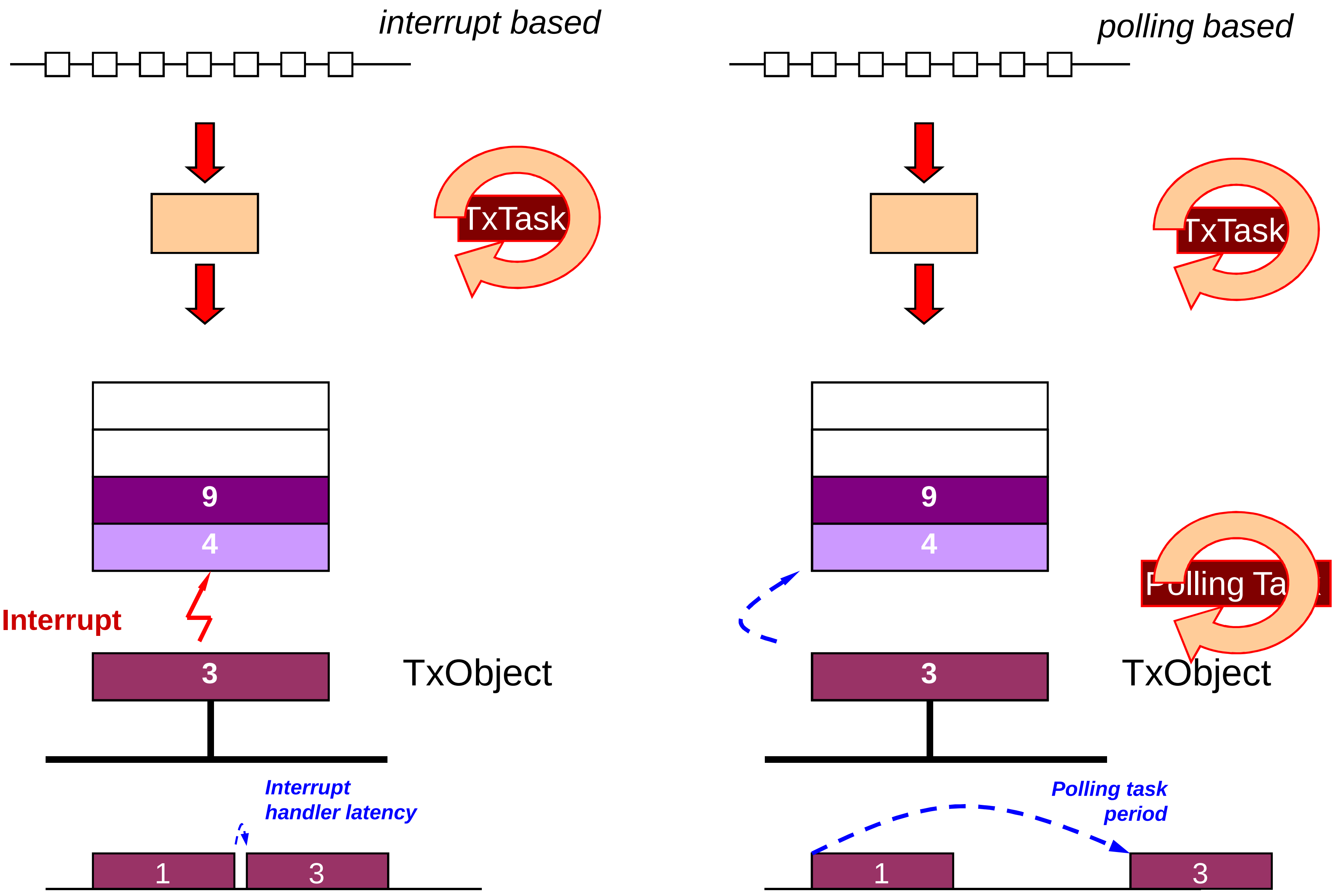


## CAN bus

Finally …

- The driver management policies may be different from what you would expect …

# Transmission modes (1)



## CAN bus

How about the average latency behavior ?

Other types of analysis are possible

By simulation

- Probably the only one that can capture effects like finite copy times, insufficient number of buffers, non-preemptability of TxObjects …

Stochastic analysis

- See recent work with Haibo Zeng [8].
- Suprisingly close to the results of trace analysis with non-preemptable single TxObjects and finite copy times!

# CAN bus

## Bibliography


[1] CAN Specification, Version 2.0. Robert Bosch GmbH. Stuttgard, 1991, http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[2] K. Tindell, H. Hansson, and A. J. Wellings, Analysing real-time communications: Controller area network (can),' Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS'94), vol. 3, no. 8, pp. 259--263, December 1994.

[3] H. Kopetz, A solution to an automotive control system benchmark, Institut fur Technische Informatik, Technische Universitat Wien, Tech. Rep., April 1994.

[4] Gergeleit M., H. Streich. Implementing a Distributed High-Resolution Real-Time clock using the CAN-Bus. Proceedings of the 1st International CAN Conference. Mainz, Germany 1994.

[5] D. Lee and G. Allan. Fault-tolerant Clock synchronisation with Microsecond-precision for CAN Networked Systems. Proceedings of the 9th International CAN Conference, Munich, Germany, 2003.

[6] A. Meschi M. Di Natale M. Spuri Priority Inversion at the Network Adapter when Scheduling Messages with Earliest Deadline Techniques , Euromicro Conference on Real-time systems, L’Aquila, Italy 1996.

[7] Jose Rufino and Paulo Verissimo and Guilherme Arroz and Carlos Almeida and Luis Rodrigues "Fault-Tolerant Broadcasts in CAN", Symposium on Fault-Tolerant Computing", 150-159, 1998.

[8] Stochastic Analysis of Controller Area Network Message Response Times, Haibo Zeng, Paolo Giusto, Marco Di Natale, Alberto Sangiovanni Vincentelli, submitted to the 2008 RTAS

[9] R. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. In RTN06, Dresden, Germany, July 2006.