# Scheduling of CAN Message Transmission when Multiple FIFOs with Assigned Priorities are Used in RTOS Drivers

Michal Lenc Pavel Píša
lencmich@fel.cvut.cz pisa@fel.cvut.cz

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

2024-05-15
international CAN Conference 2024

# Content of Presentation

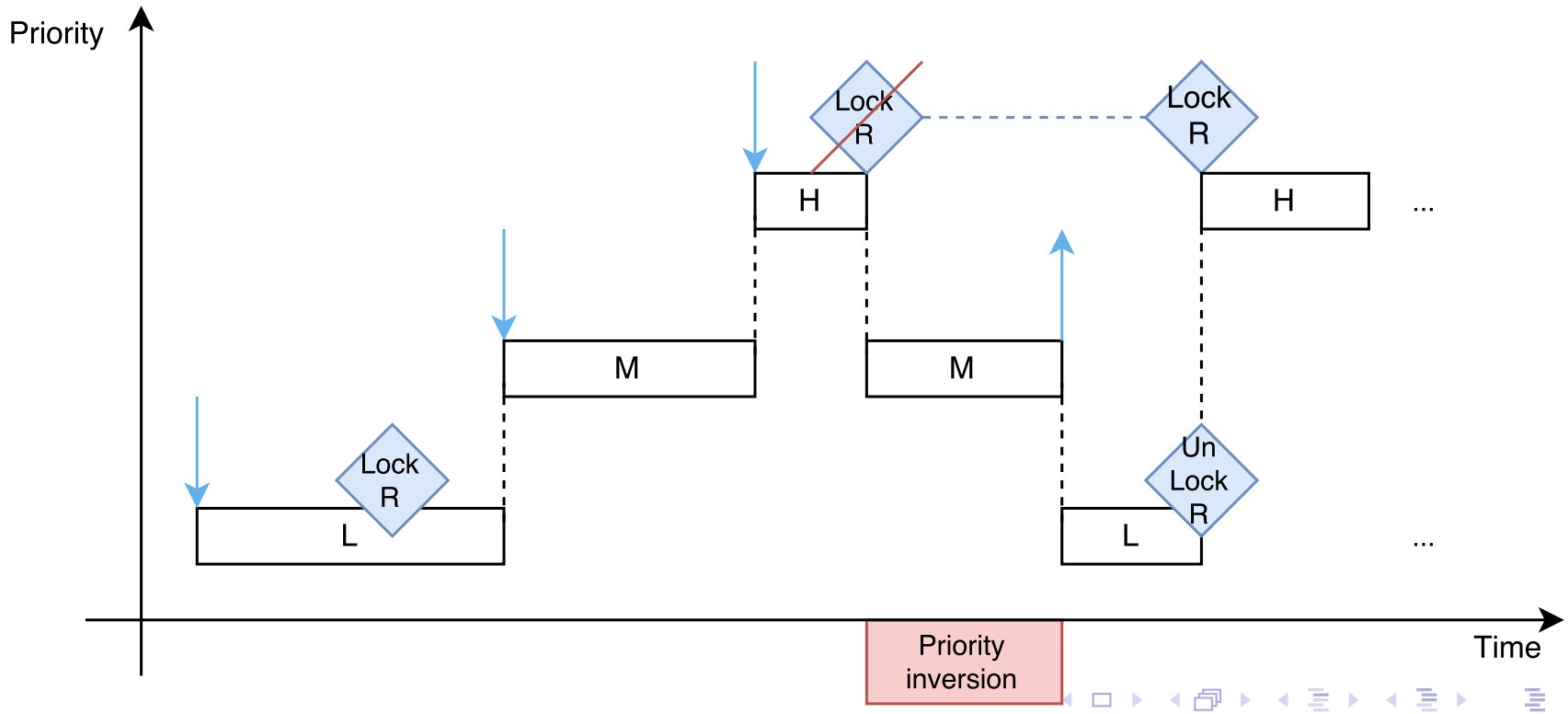
1. Priority Inversion Problem
   - Introduction
   - Mapping Priority Classes to HW Buffers
2. Dynamic Allocation of TX Buffers to Multiple Priority Groups
   - Determining Correct Sequence
   - Determining Correct Sequence – Example
   - Solution Requirements
3. Results
   - Measured Latencies
4. Conclusion
5. References

# Outline

1. Priority Inversion Problem
   - Introduction
   - Mapping Priority Classes to HW Buffers
2. Dynamic Allocation of TX Buffers to Multiple Priority Groups
   - Determining Correct Sequence
   - Determining Correct Sequence – Example
   - Solution Requirements
3. Results
   - Measured Latencies
4. Conclusion
5. References

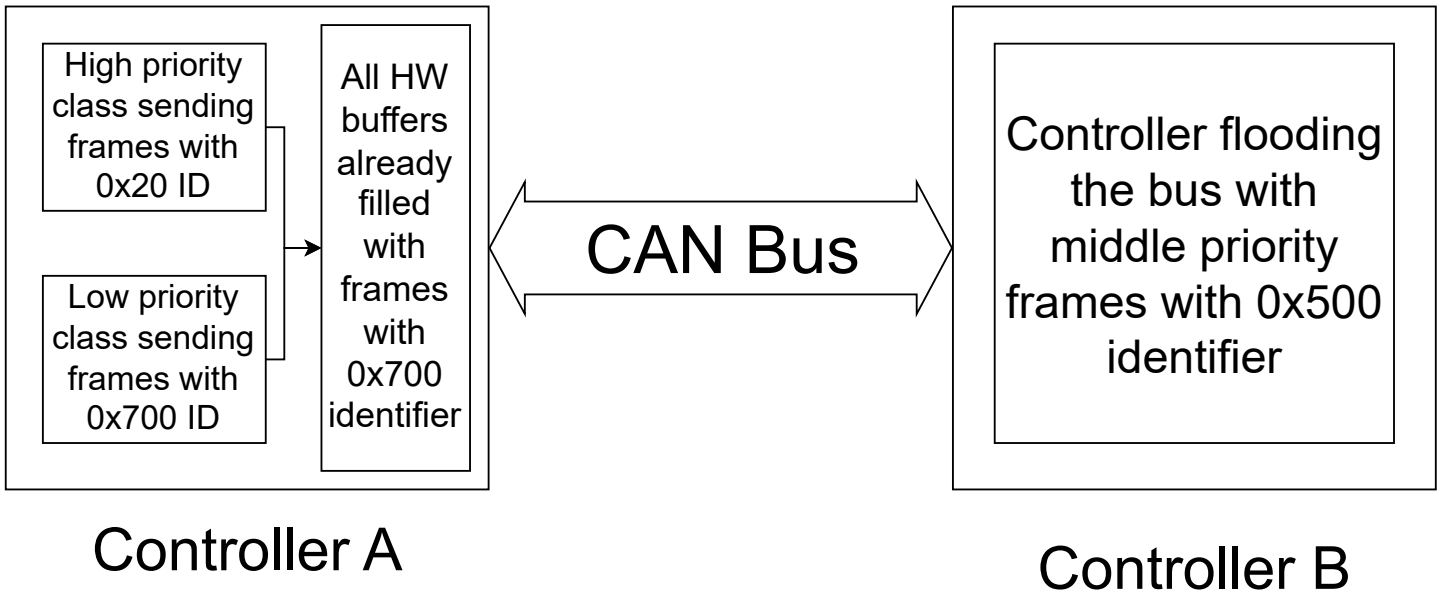
# ...in computer science

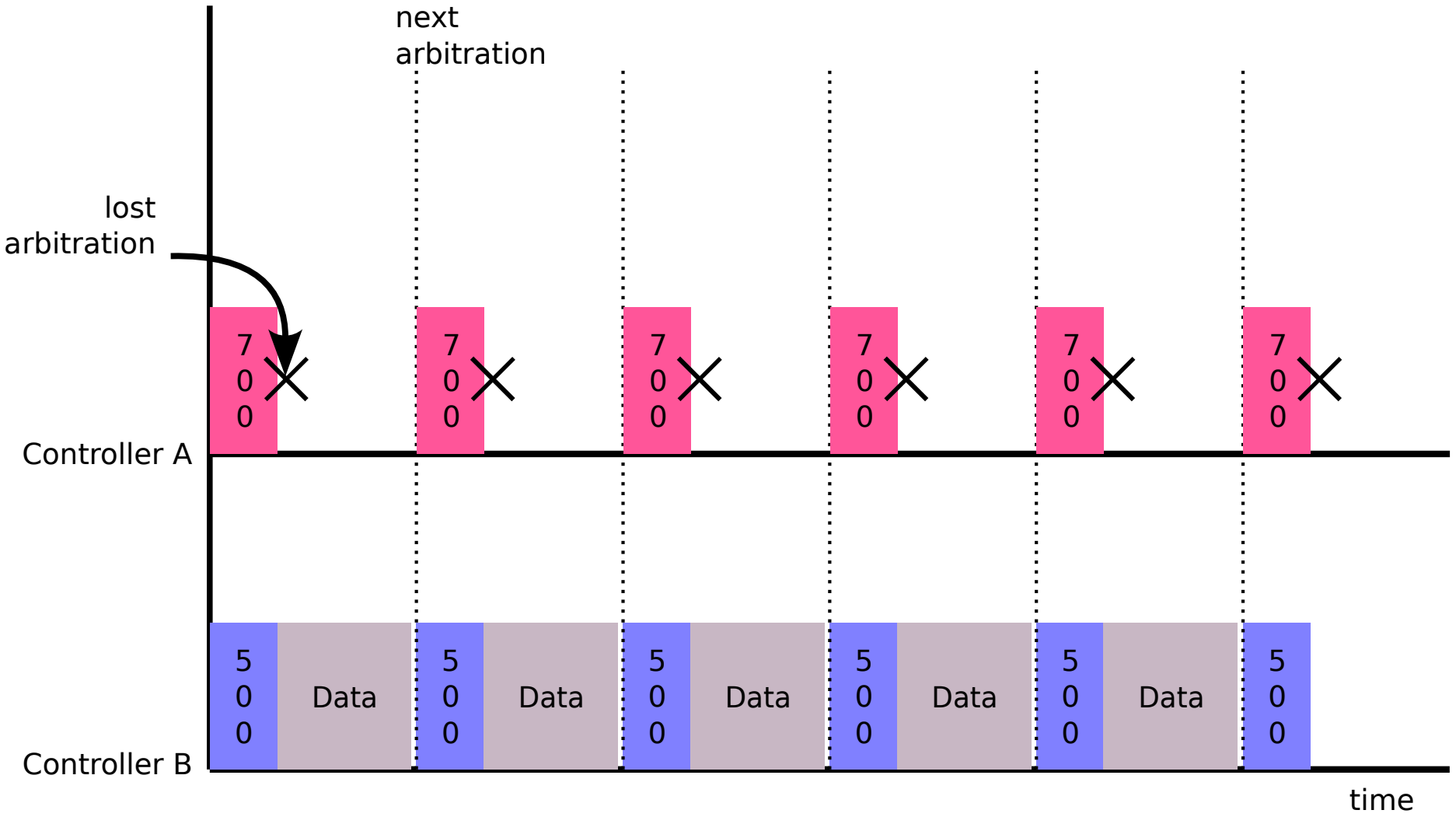
- low priority task, correctly preempted by middle priority task, is holding a resource required by high priority task
- this leads to high priority task incorrectly wait for middle priority task to finish



# ...in CAN bus

- the problem may occur on CAN bus during arbitration phase
- tasks are CAN frames, locked resources are HW buffers
- one controller is flooding the bus with middle priority frames
- second is trying to send mix of low and high priority frames
- if all HW buffers are filled with low priority frames, high priority ones will never get to arbitration phase




next
arbitration
lost
arbitration
7
0
0
Controller A
5
0
0
Data
Controller B
time

# Priority Classes

- common solution is the introduction of priority classes (usually FIFO queues) with assigned priority
- frames are assigned to these classes according to their identifier range
- this way high priority messages can be passed to the controller in their own FIFO
- these classes still has to be mapped to controller's hardware buffers, this causes another problems

# Limitations

- controllers usually allow frame transmission based on their CAN identifiers or in the fixed order determined by the TX buffer index
- applications/protocols may require preservation of message order within the same priority class even if different IDs are used
  - the frames from the highest available priority class have to be sent first and order within the class must be preserved
- order preservation disqualifies frame transmission in identifier order
- current drivers usually limit the transmission to one buffer per class or even to one TX buffer at all
- this highly limits the controller's potential

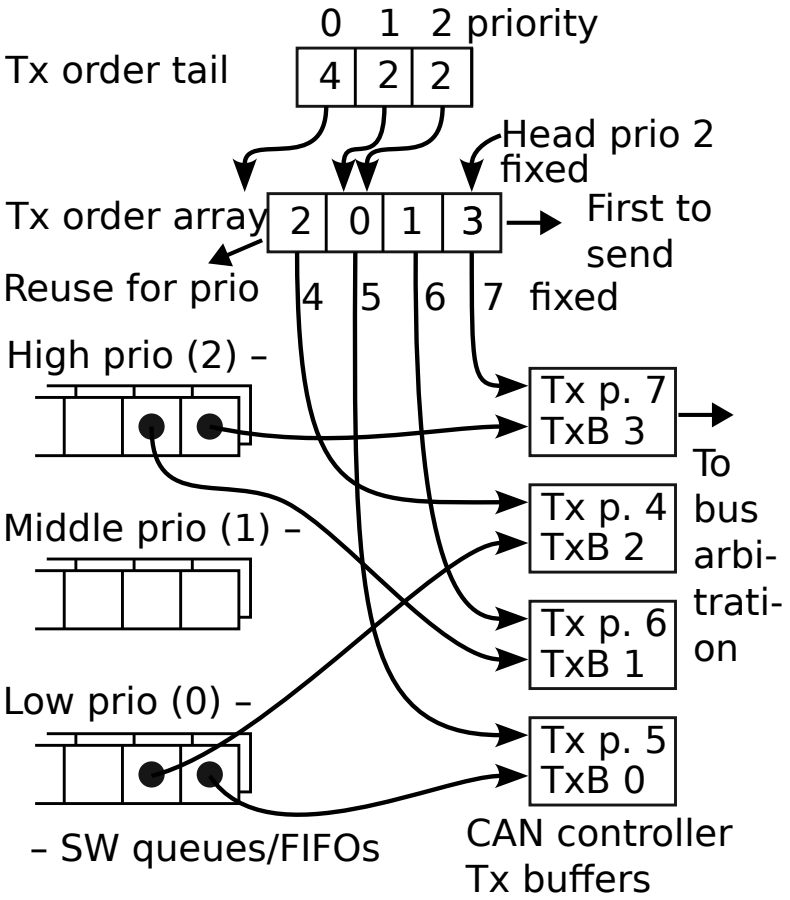
# Outline

1. Priority Inversion Problem
   - Introduction
   - Mapping Priority Classes to HW Buffers
2. Dynamic Allocation of TX Buffers to Multiple Priority Groups
   - Determining Correct Sequence
   - Determining Correct Sequence – Example
   - Solution Requirements
3. Results
   - Measured Latencies
4. Conclusion
5. References

# Introduction

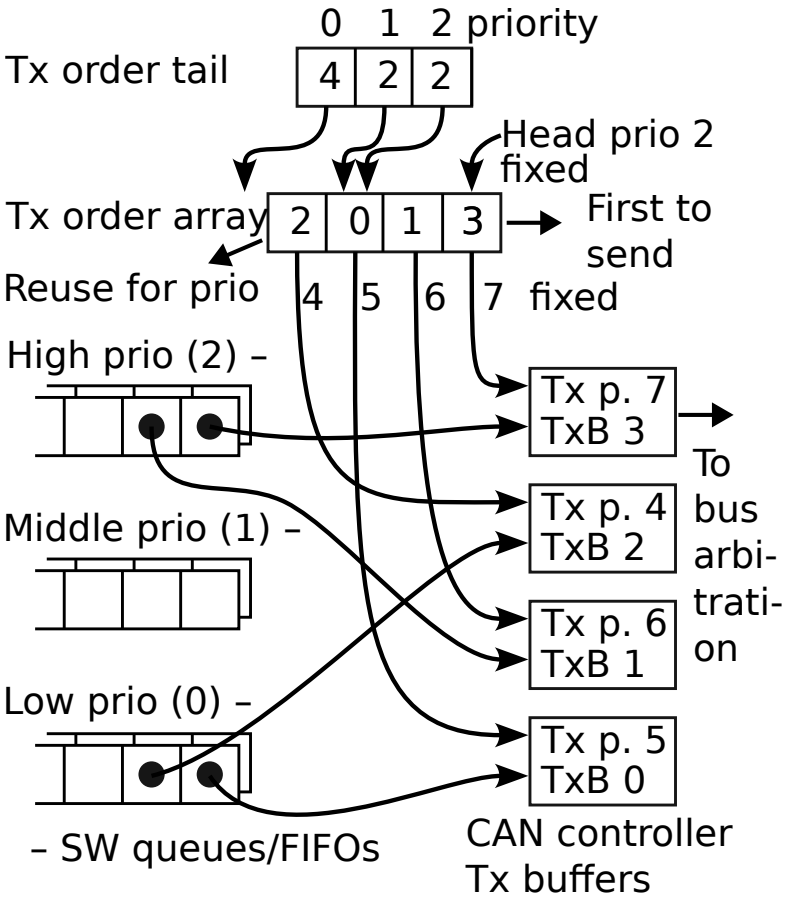
- our design extends the common solution of FIFO queues for each priority class
- dynamic redistribution of hardware transmission buffers to these classes is added
- this process ensures the hardware buffers are assigned the correct priority – order in which they are sent to the network
- this order is determined by both priority classes and order within them
- **the buffer with a newly inserted message has to be inserted in the transmit sequence after all messages of the same or higher priority class but before all messages of a lower priority class**

# Determining Correct Sequence



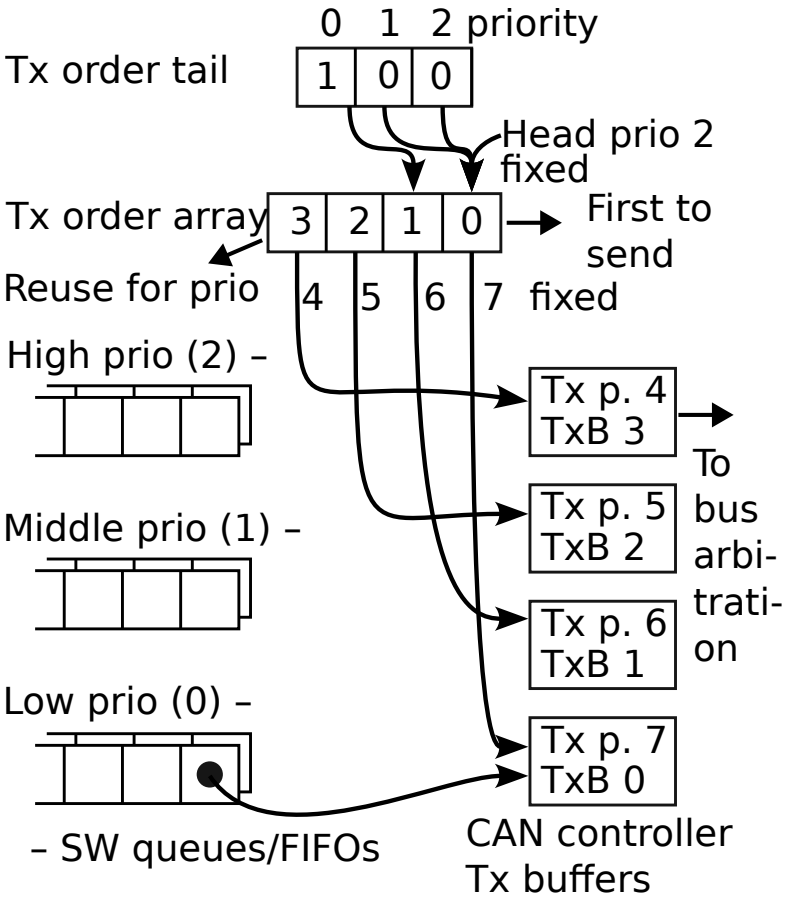

- TX order array holds the numbers of hardware buffers as they should be processed.
- TX order tail array points to the tail for a given priority class – place where new frame should be inserted
- head for the highest priority is fixed and heads for lower priorities are at the exactly same position as previous priority tails
- if new frame inserted, all frames from lower priority classes are moved left

# Determining Correct Sequence



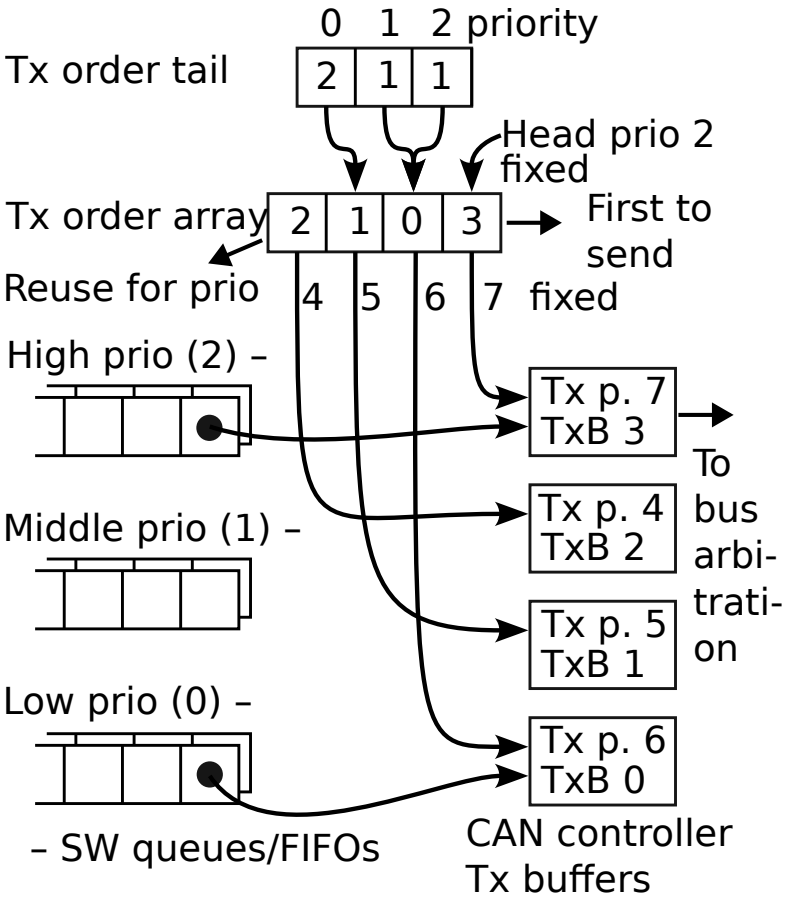

- if no space available in hardware buffers and message of lower priority class (compared to pending message) occupies the buffers, abort must take place
- the oldest message of the lowest priority class is aborted and scheduled for later processing
- new frame is inserted into free buffer and this buffer is reorganized to correct position

# Determining Correct Sequence – Example



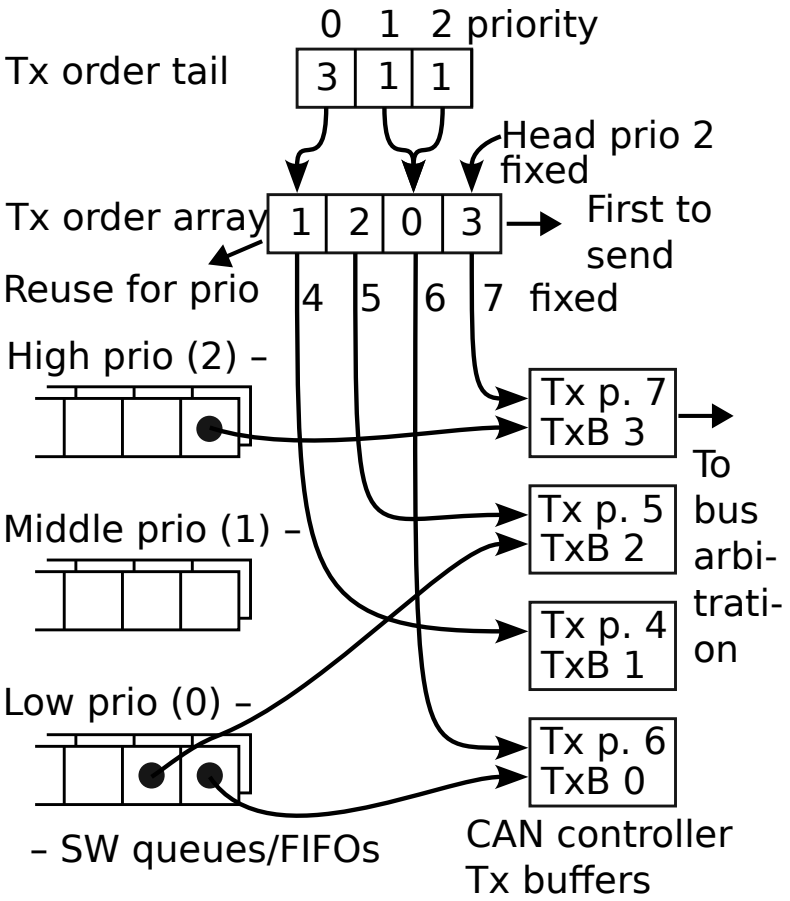

- low priority message assigned to buffer 0
- TX order tail for low priority class moved to position 1
- buffer 0 is first to send with the highest priority 7

# Determining Correct Sequence – Example



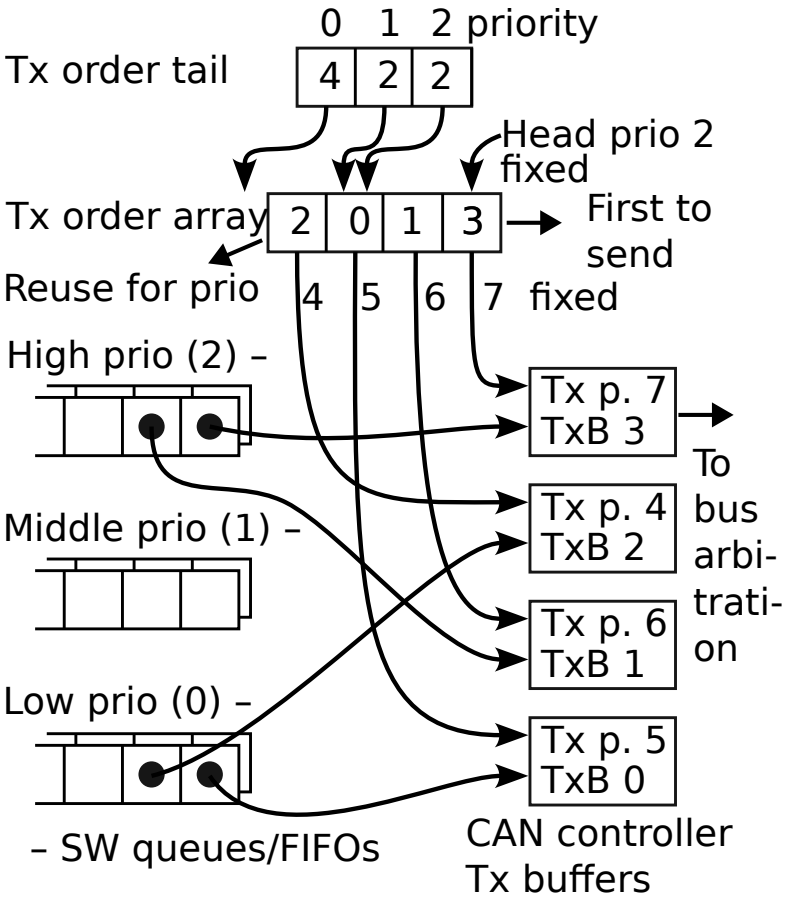

- high priority message assigned to buffer 3
- this frame has to be inserted before low priority one
- TX order tail for high and middle priority class is moved to 1, low priority class to 2
- buffer 3 now has the highest priority 7, buffer 0 has priority 6

# Determining Correct Sequence – Example



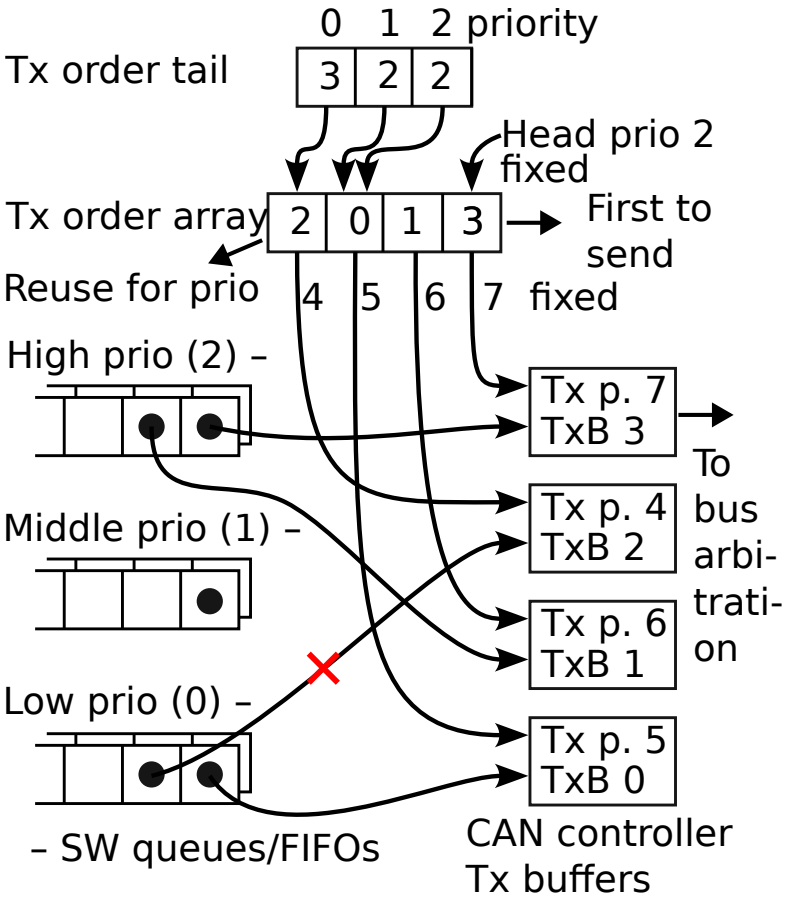

- low priority message assigned to buffer 2
- this frame has to be inserted after all high and middle priority frames and after all previously added low priority frames
- TX order tail for high and middle priority class remains at 1, low priority class is to 2 as new frame is added
- priorities for buffers 2 and 1 are switched

# Determining Correct Sequence – Example



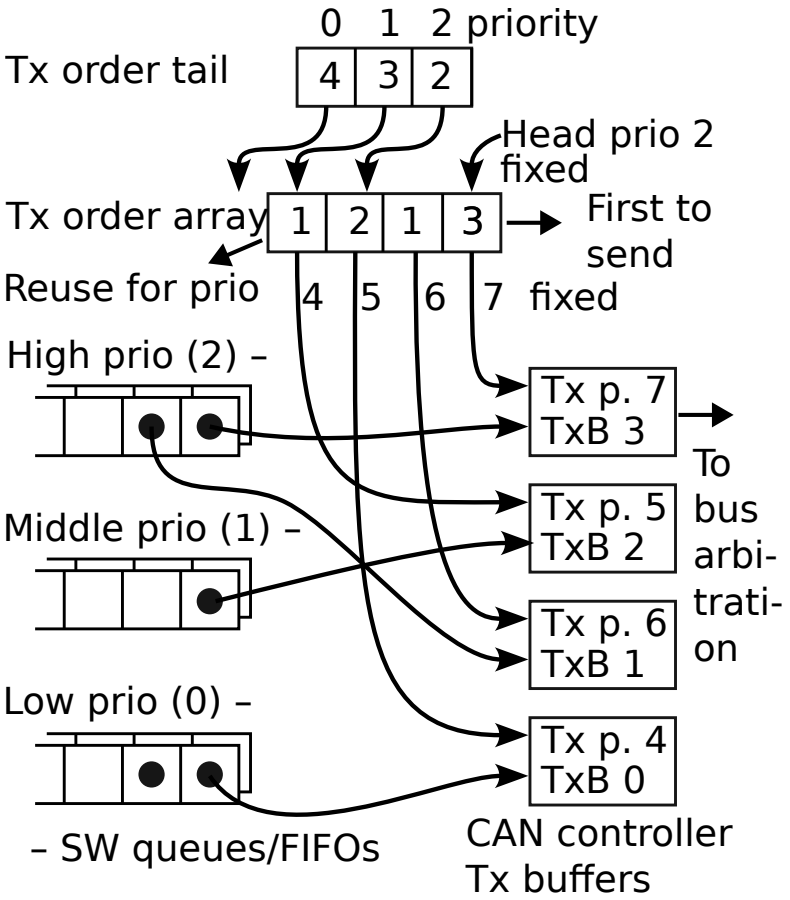

- high priority message assigned to buffer 1
- this frame has to be inserted after all high priority frames and before all low/middle priority frames
- TX order tail for high and middle priority class is moved to 2, low priority frames are moved left, tail to position 4 – no more space
- buffer 1 assigned priority 6 and priorities for lower buffers moved left

# Determining Correct Sequence – Example




- middle priority messages is available in FIFO class
- all HW buffers are full, the oldest message of the lowest priority has to be aborted
- buffer 2 is aborted
- low priority tail is moved right to position 3

# Determining Correct Sequence – Example




- middle priority class is assigned to free buffer 2
- this buffer is sorted between high and low priority buffers
- middle priority tail is moved to 3, low priority tail to 4
- buffer 2 is assigned priority 5, buffer 0 priority 4
- buffers with messages from high priority class remains unchanged

# Solution Requirements

- on upper layer (common CAN stack)
  - support for multiple priority classes
  - possibility to reschedule aborted frames for later processing – stack should keep slots allocated until transmission done
- on controller
  - possibility to assign sending order to HW buffers
  - possibility to abort HW buffer

# Outline

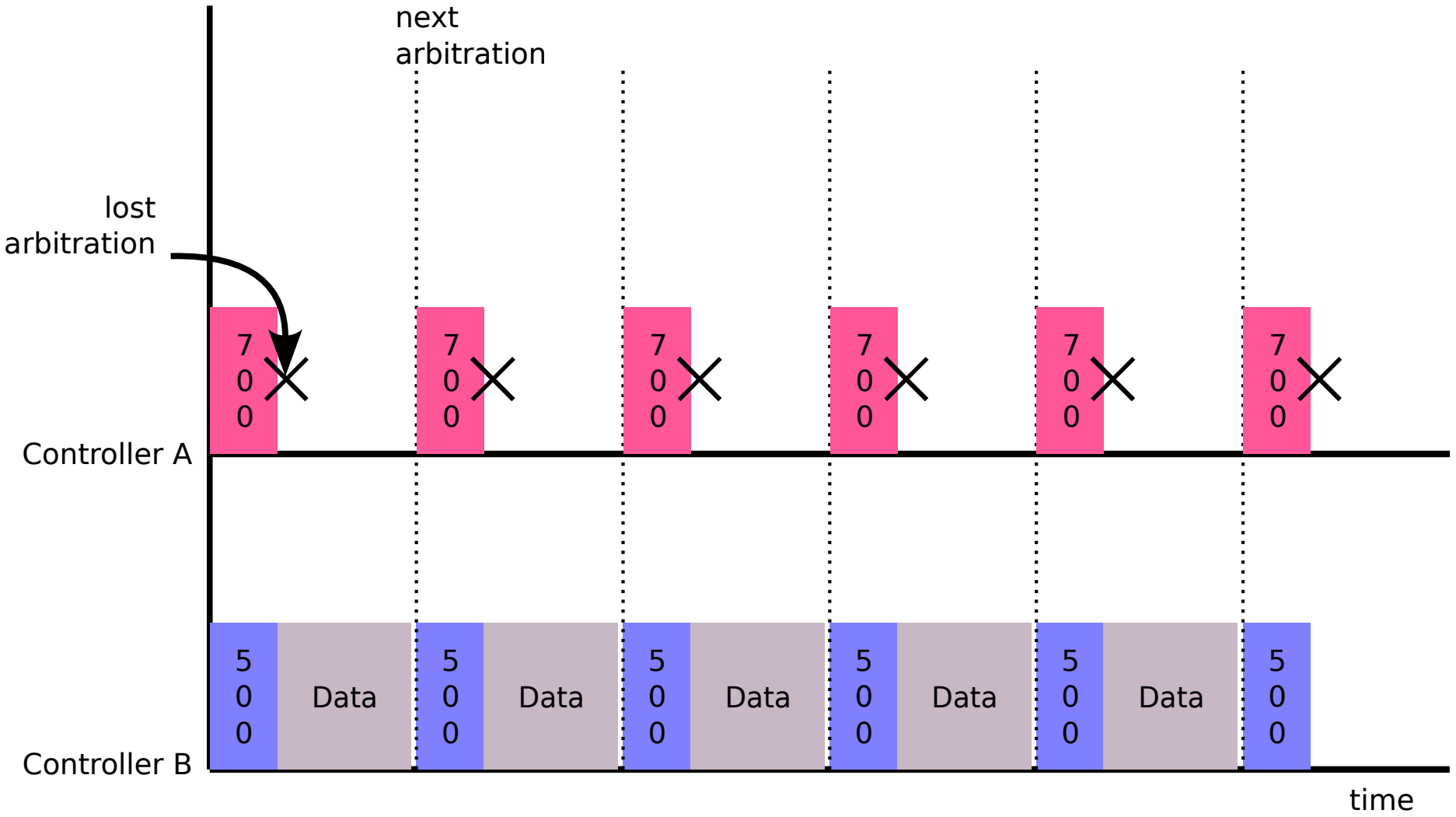
1. Priority Inversion Problem
   - Introduction
   - Mapping Priority Classes to HW Buffers
2. Dynamic Allocation of TX Buffers to Multiple Priority Groups
   - Determining Correct Sequence
   - Determining Correct Sequence – Example
   - Solution Requirements
3. Results
   - Measured Latencies
4. Conclusion
5. References

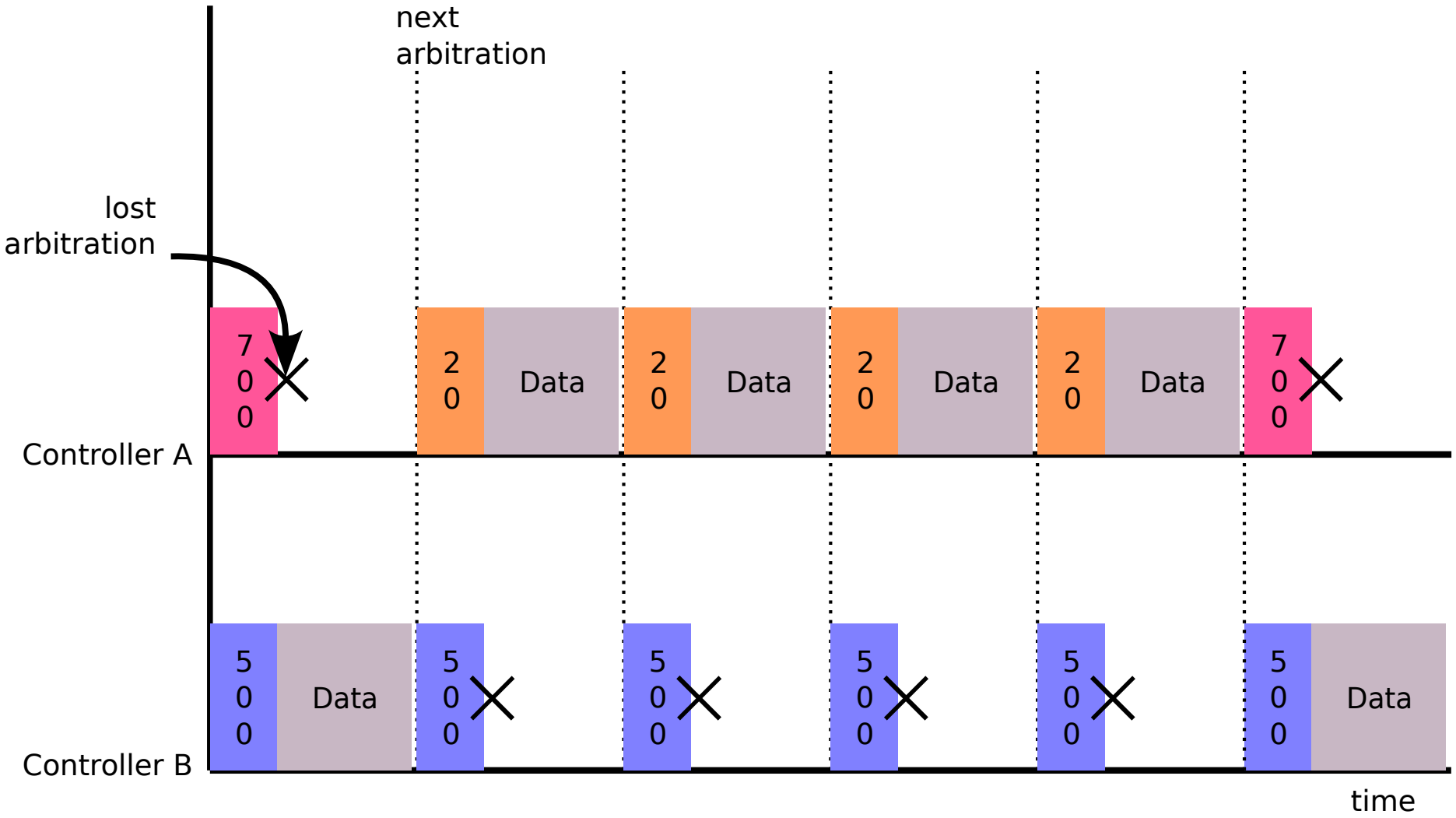
# Target Platform

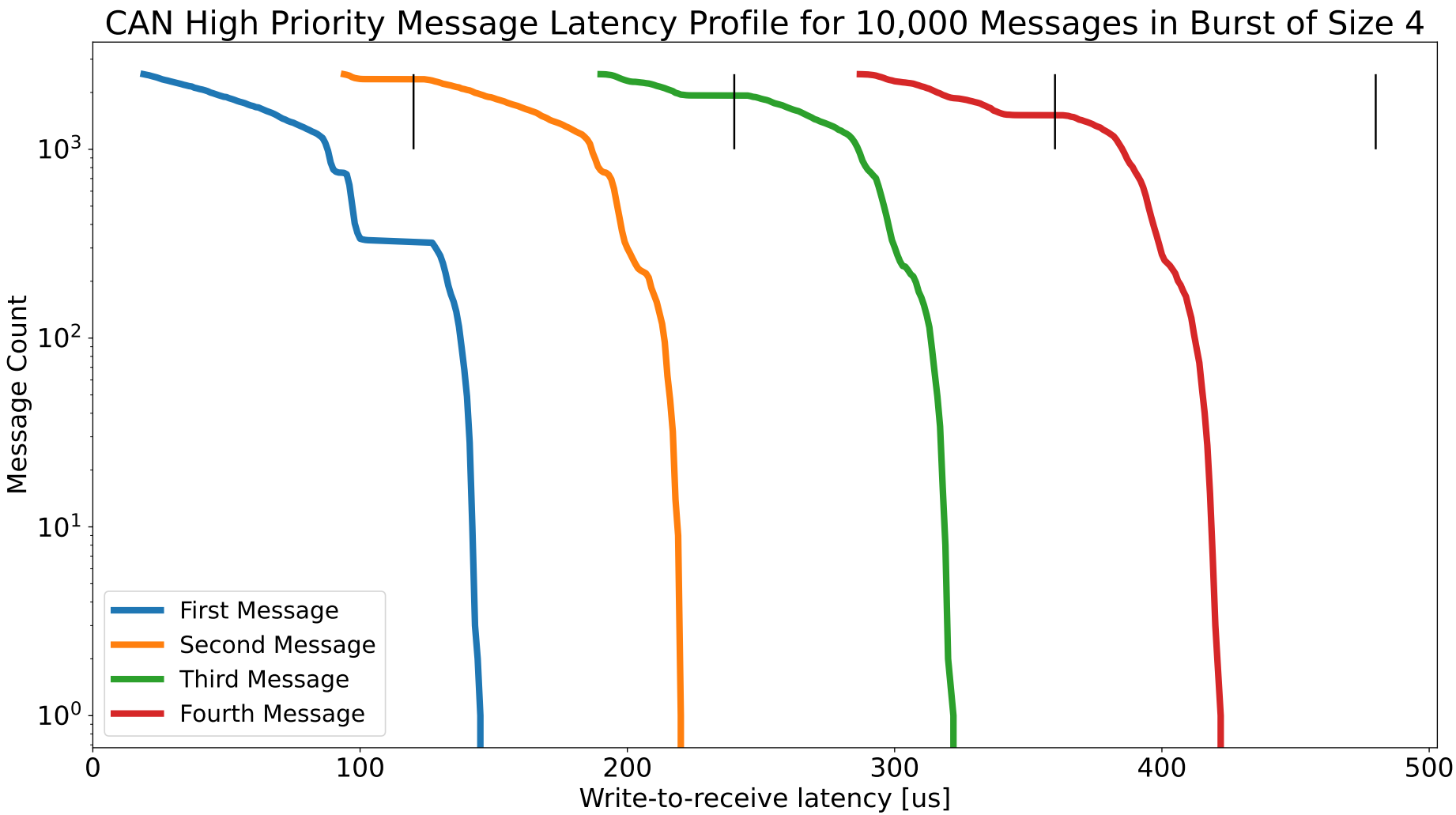
- new full-featured CAN/CAN FD stack for RTEMS executive
  - `https://www.rtems.org/`
- implemented as a character device driver, POSIX compliant
- tested with CTU CAN FD open source IP core
  - provides precise timestamping with 10 ns resolution for latencies measurements
- used hardware: educational kit MicroZed APO based on MicroZed evaluation kit with Xilinx Zynq-7000 system on chip
- two CAN controllers used on the board
- four HW transmission buffers on each

# Test Framework

- controller A fully loading the CAN bus with 8-byte long messages with 0x500 identifier – middle priority
- controller B accessed from two applications
  - one sending messages with a 0x700 identifier – low priority
  - other sending 8-byte long messages with a 0x20 identifier – high priority
- high priority messages send in burst of size four
- wait between bursts long enough to all frames to the bus
- write to receive latency measured with receive HW timestamping at SOF bit
- **test simulated fully loaded bus by one controller and another controller trying to access this bus**


next
arbitration
lost
arbitration
7
0
0
Controller A
5
0
0
Data
Controller B
time


next
arbitration
lost
arbitration
7
0
0
2
0
Data
2
0
Data
2
0
Data
2
0
Data
7
0
0
Controller A
5
0
0
Data
5
0
0
5
0
0
5
0
0
5
0
0
5
0
0
Data
Controller B
time

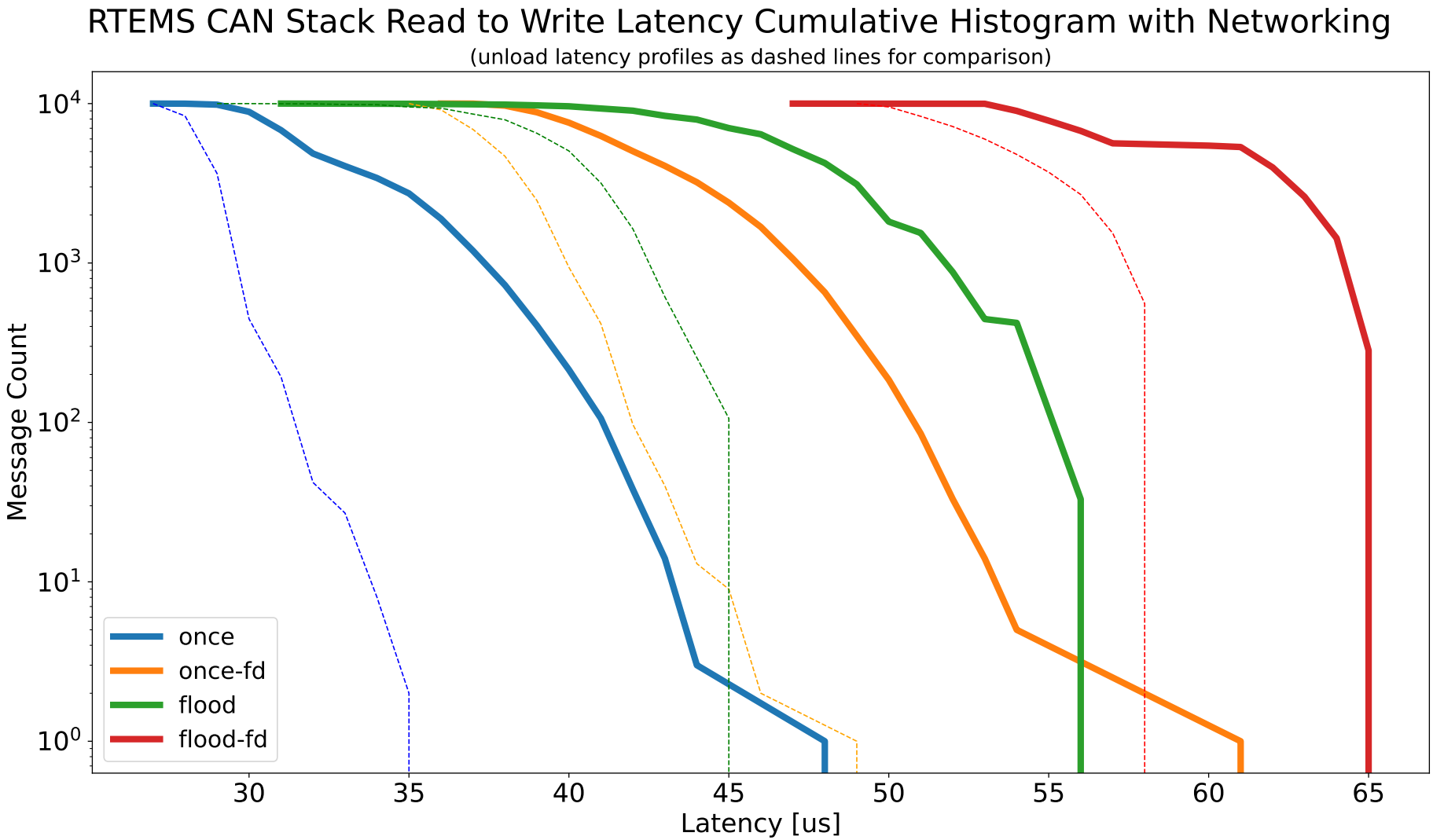

CAN High Priority Message Latency Profile for 10,000 Messages in Burst of Size 4
Message Count
$10^3$
$10^2$
$10^1$
$10^0$
0
100
200
300
400
500
Write-to-receive latency [us]
First Message
Second Message
Third Message
Fourth Message

# Outline

1. Priority Inversion Problem
   - Introduction
   - Mapping Priority Classes to HW Buffers
2. Dynamic Allocation of TX Buffers to Multiple Priority Groups
   - Determining Correct Sequence
   - Determining Correct Sequence – Example
   - Solution Requirements
3. Results
   - Measured Latencies
4. Conclusion
5. References

# Conclusion

- presented algorithm can be used to ensure the correct transmit order of various priority messages
- this is possible even if all controller's HW buffers are used
- the solution is not hardware specific, it may be ported to more controllers and systems
- we aim to contribute this solution and our new CAN/CAN FD stack to RTEMS mainline in upcoming months
  - development branch: gitlab.fel.cvut.cz/otrees/rtems/rtems-canfd
- **CAN bus CTU FEE Projects:** see our CAN projects at `https://canbus.pages.fel.cvut.cz/`


RTEMS CAN Stack Read to Write Latency Cumulative Histogram with Networking
(unload latency profiles as dashed lines for comparison)
Message Count
$10^4$
$10^3$
$10^2$
$10^1$
$10^0$
30
35
40
45
50
55
60
65
Latency [us]
once
once-fd
flood
flood-fd

# Outline

1. Priority Inversion Problem
   - Introduction
   - Mapping Priority Classes to HW Buffers
2. Dynamic Allocation of TX Buffers to Multiple Priority Groups
   - Determining Correct Sequence
   - Determining Correct Sequence – Example
   - Solution Requirements
3. Results
   - Measured Latencies
4. Conclusion
5. References

# References

**CAN bus CTU FEE Projects:**

# References


- CTU CAN FD, CAN for QEMU, CAN/CAN FD Latecy and RTEMS CAN FD stack https://canbus.pages.fel.cvut.cz/
- Linux kernel CAN latencies daily testing https://canbus.pages.fel.cvut.cz/can-latester/
- Pavel Píša, Jiří Novák, Pavel Hronek, and Matěj Vasilevski. Continuous CAN Bus Subsystem Latency Evaluation and Stress Testing on GNU/Linux-Based Systems. embedded world Conference 2024. 2024.
- Priority inversion figure from: Federico Reghenzani, Giuseppe Massari, and William Fornaciari. 2019. The Real-Time Linux Kernel: A Survey on PREEMPT_RT. ACM Comput. Surv. 52, 1, Article 18 (January 2020), 36 pages. https://doi.org/10.1145/3297714