Implementation and Evaluation of Multiprocessor Resource Synchronization Protocol (MrsP) on LITMUS^{RT}

Junjie Shi[1], Kuan−Hsun Chen[1], Shuai Zhao[2], Wen−Hung Huang[1], Jian−Jia Chen[1], Andy Wellings[2]

[1] Technical University of Dortmund, Germany
[2] University of York, United Kingdom

Outline

- Background and motivations
- Implementation of MrsP [1]
- Evaluation
- Conclusions

Background and motivation

- Resource synchronization protocols are needed
  - Semaphores are used to protect shared resources
  - Priority inversion [2] may destroy the predictability
- Many synchronization protocols are available
  - MPCP [3] DPCP [7] and DNPP (suspension-based)
  - MrsP [1] (spin-based)
- Include runtime overheads into schedulability analysis

Platform

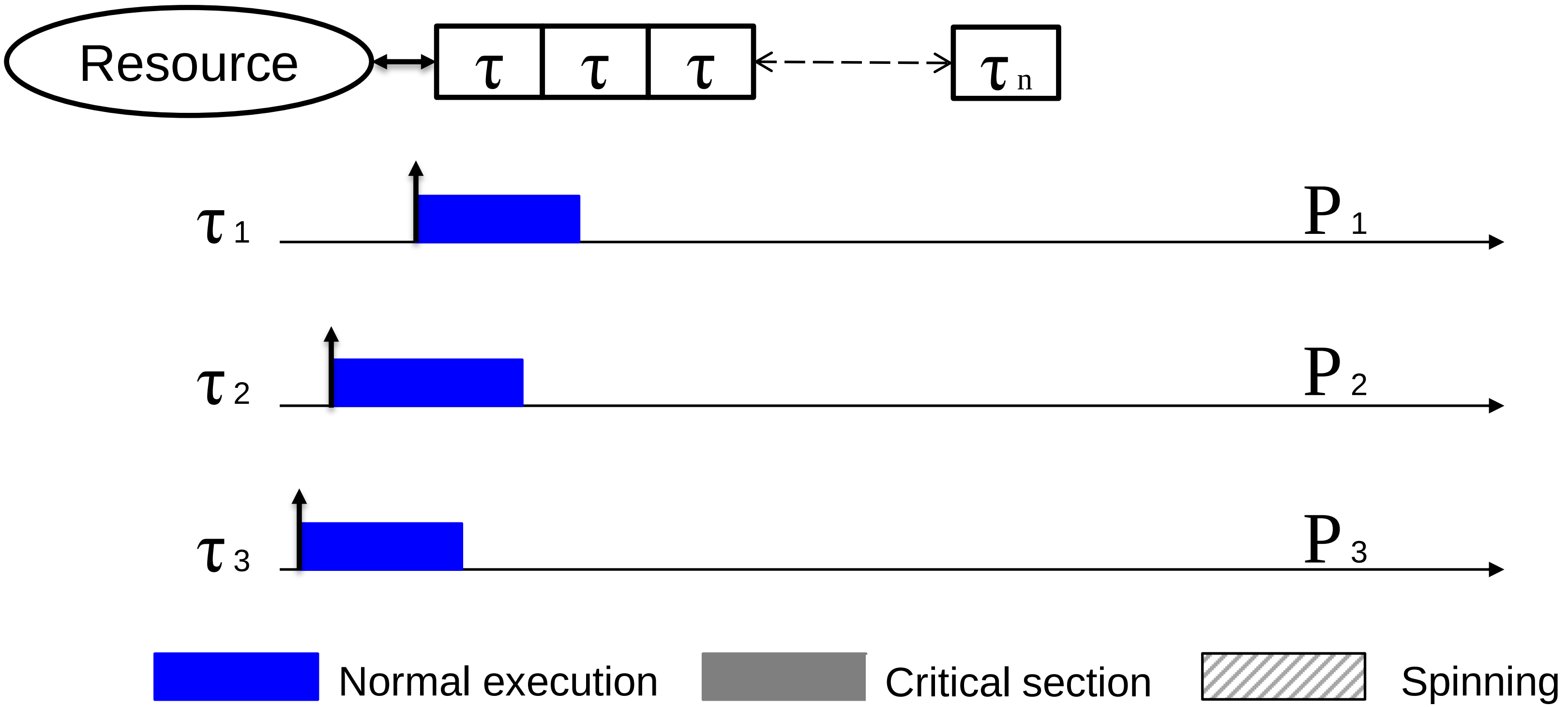
Why LITMUS^{RT} [8] ?

- Open source code
- Well established evaluation platform
- Well distributed, plug-in architecture
- Useful tools provided (overheads and schedule tracing)
- Several protocols have been implemented (MPCP/DPCP)

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

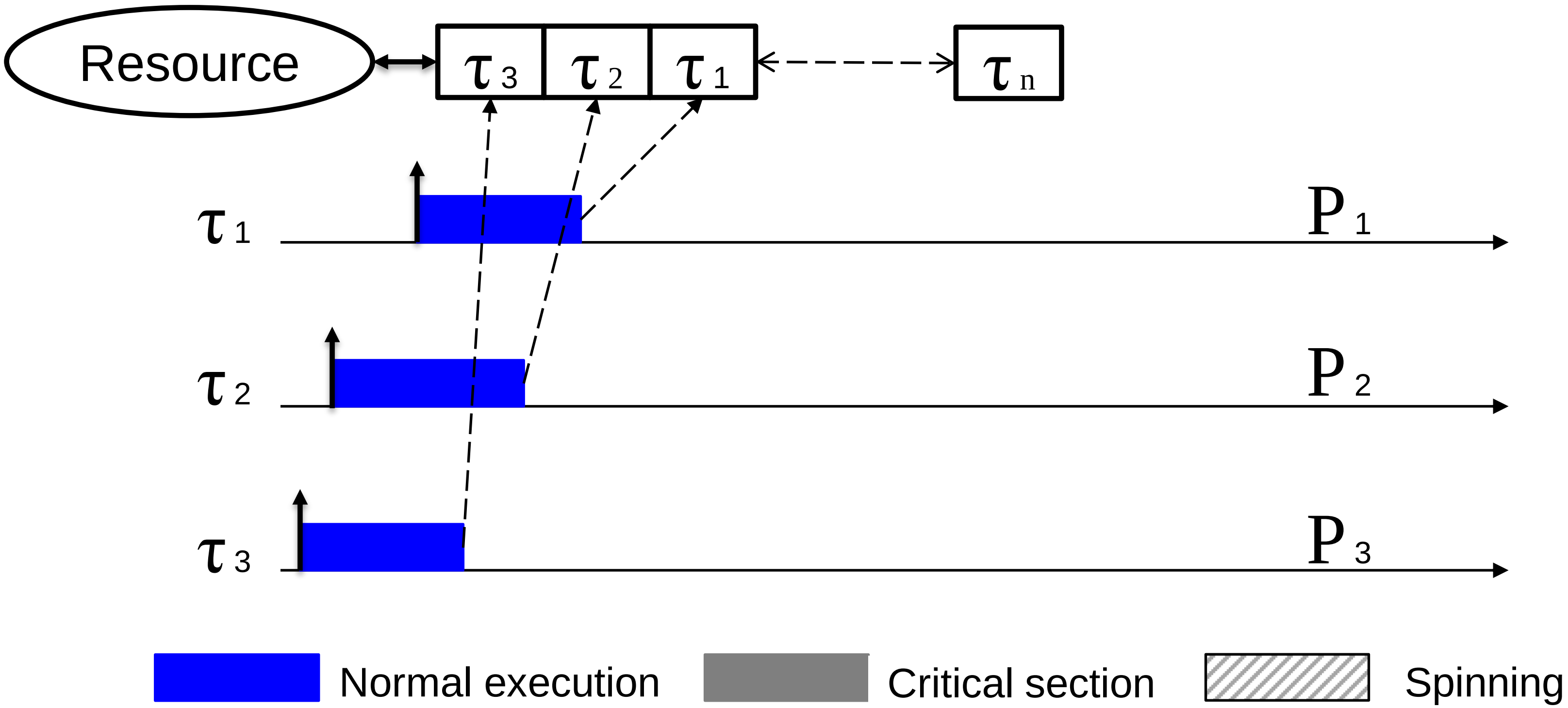
MrsP

- Requests in FIFO queue



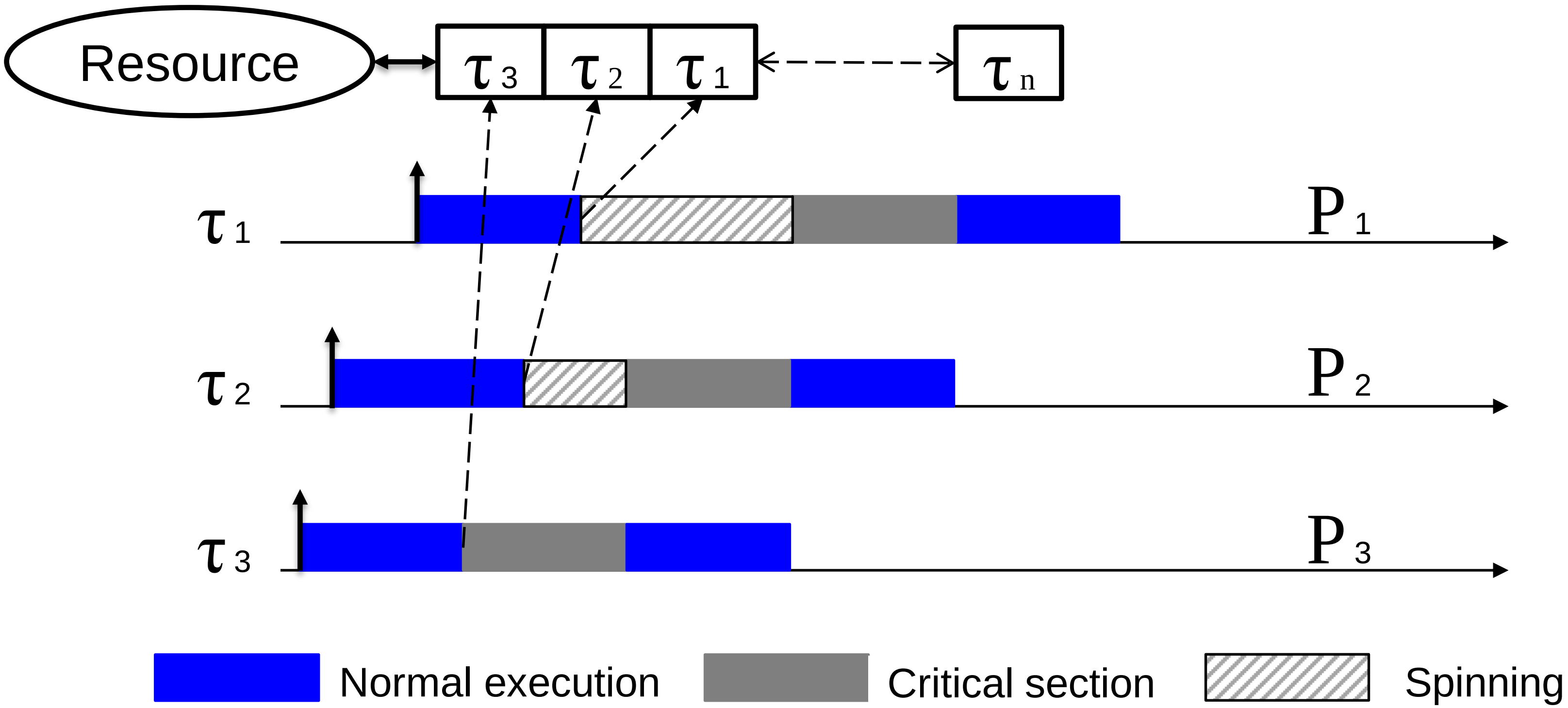
MrsP

- Requests in FIFO queue



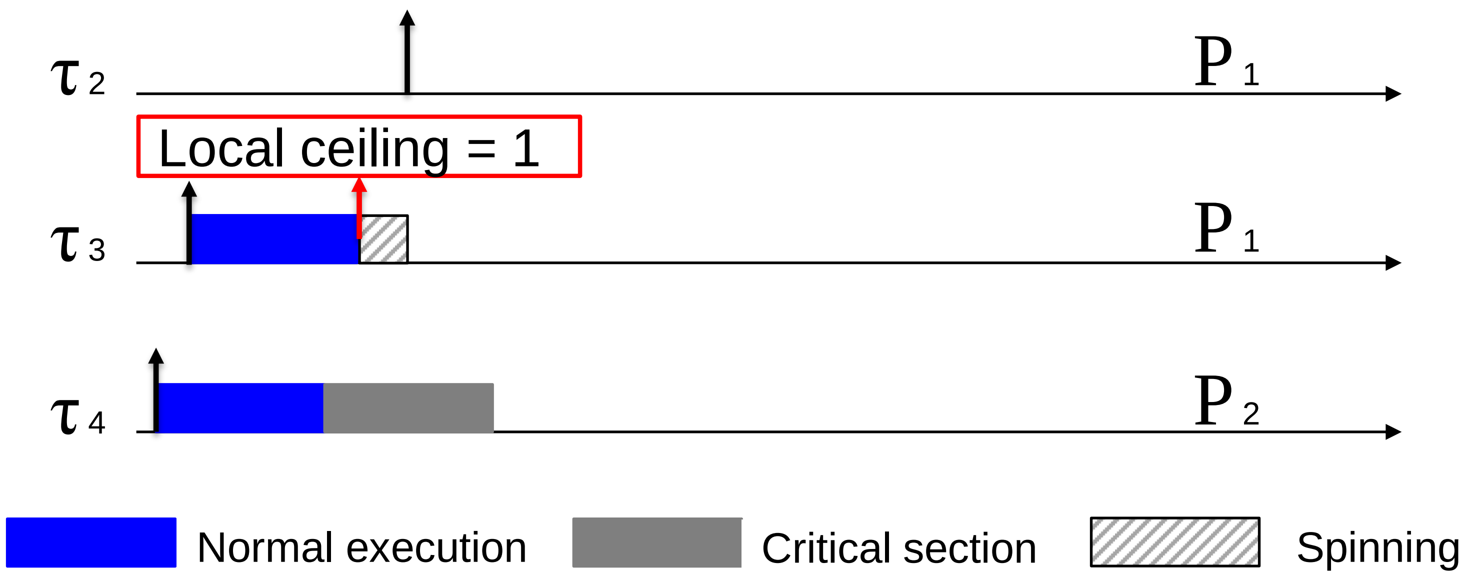
MrsP

- Requests in FIFO queue



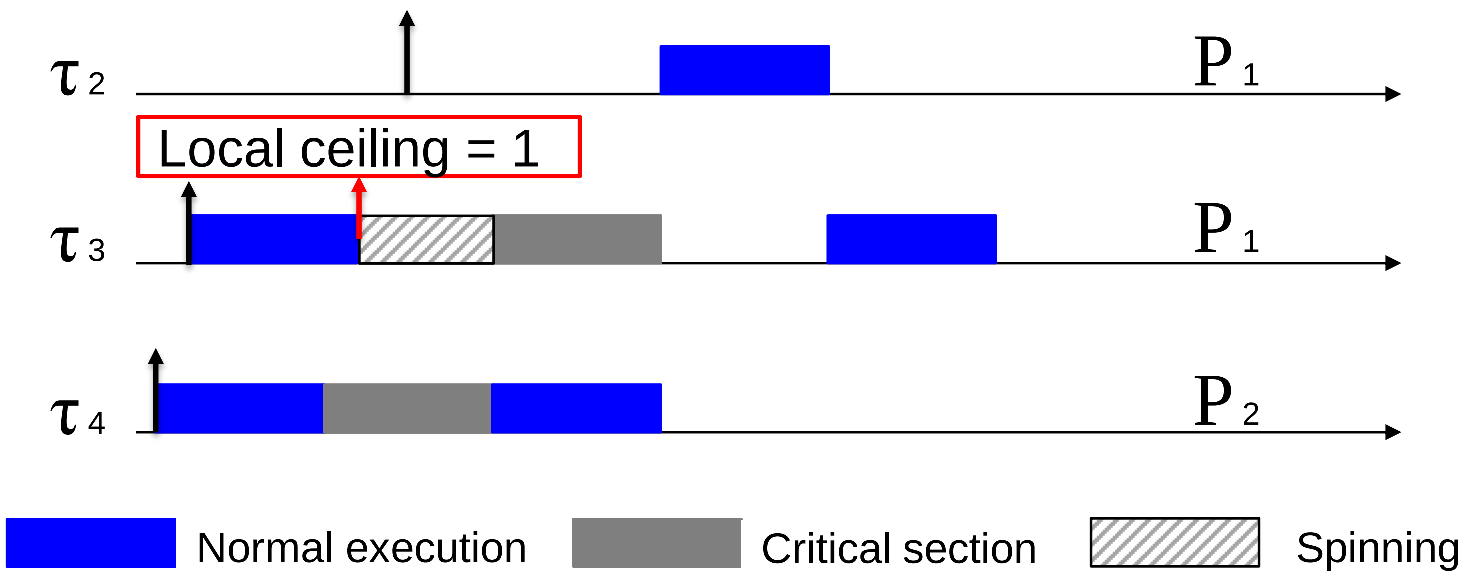
MrsP

- Local ceilings are needed while spin waiting and executing



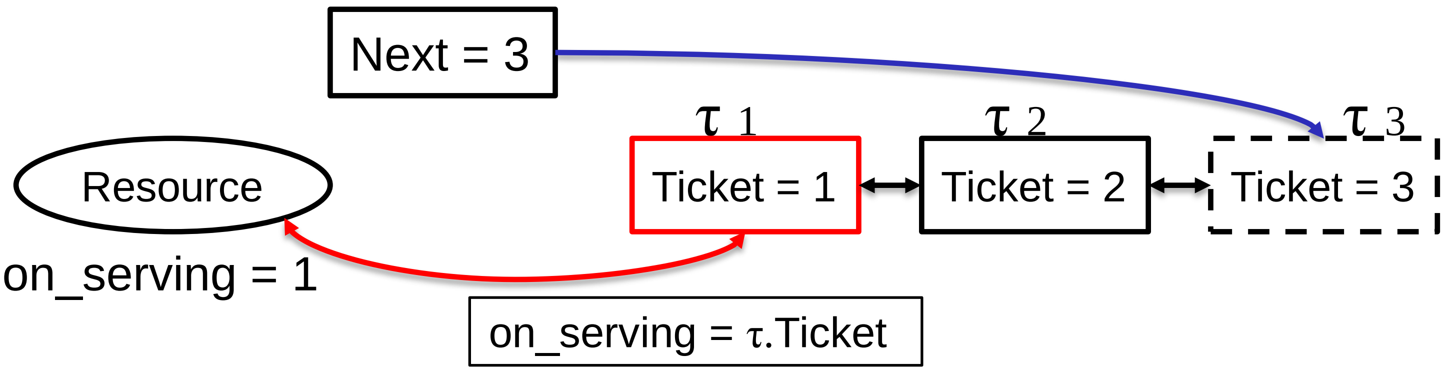
MrsP

- Local ceilings are needed while spin waiting and executing



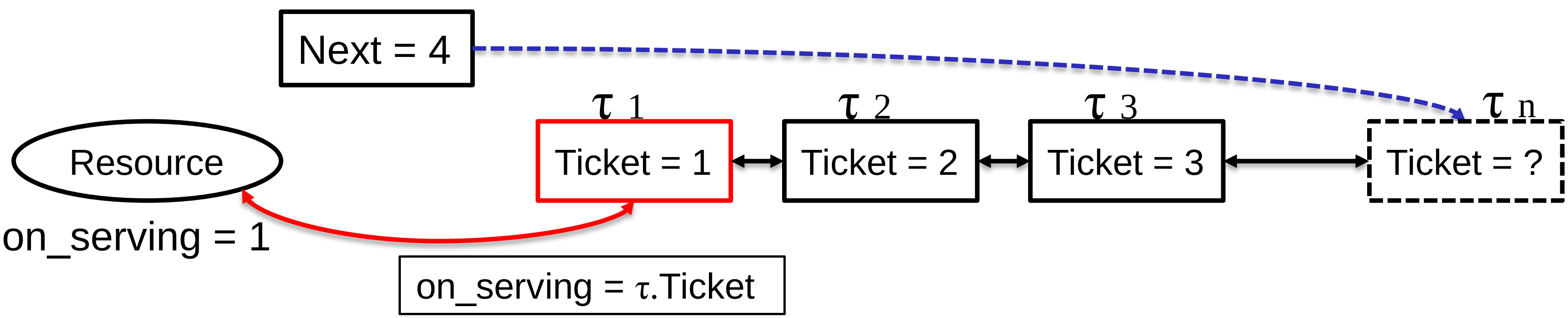
Implementation of MrsP

- Requests in FIFO queue
  - Ticket based spin lock [9]



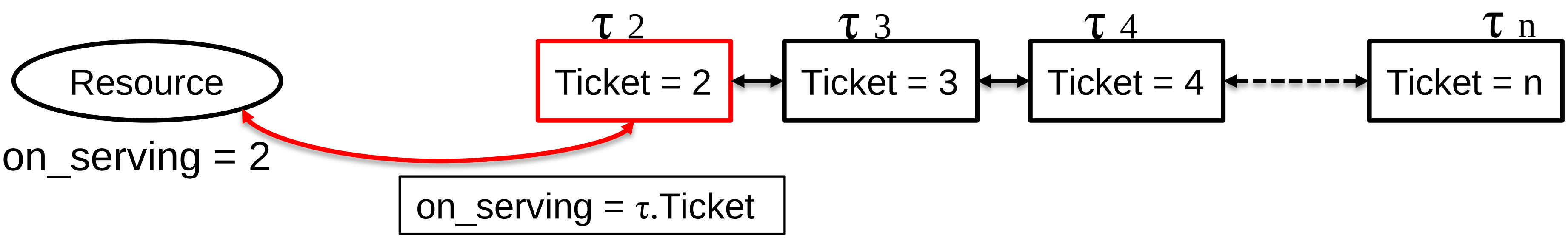
Implementation of MrsP

- Requests in FIFO queue
  - Ticket based spin lock [9]



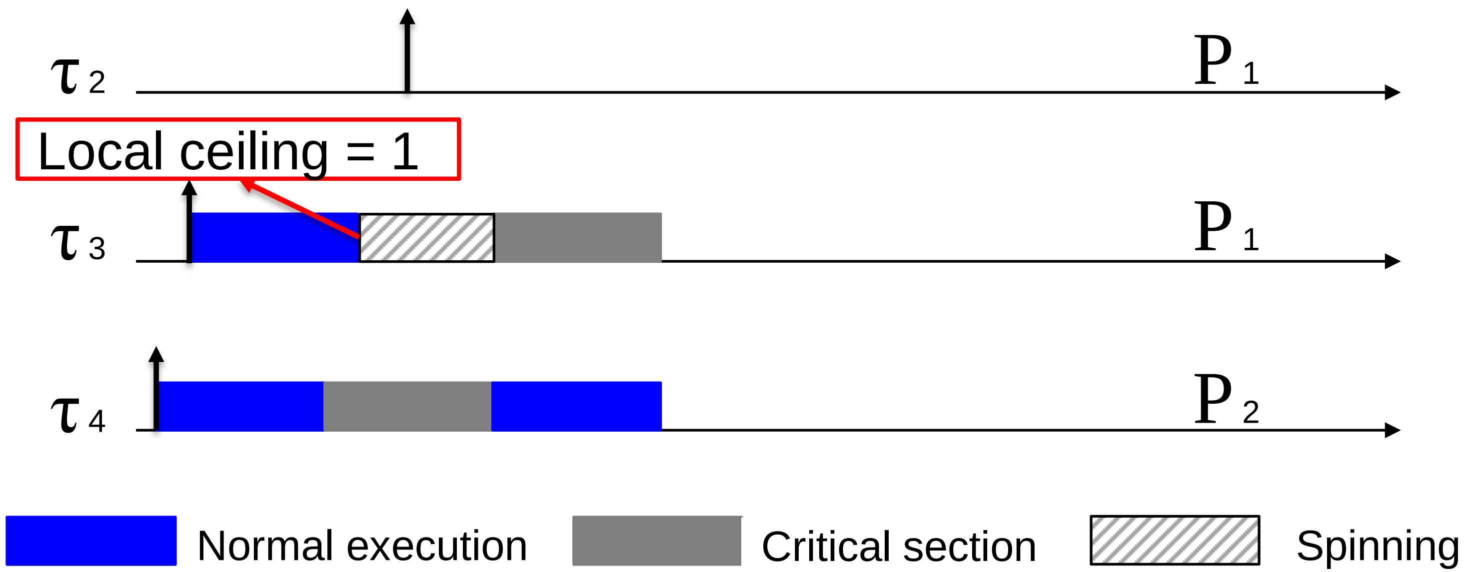
Implementation of MrsP

- Requests in FIFO queue
  - Ticket based spin lock [9]



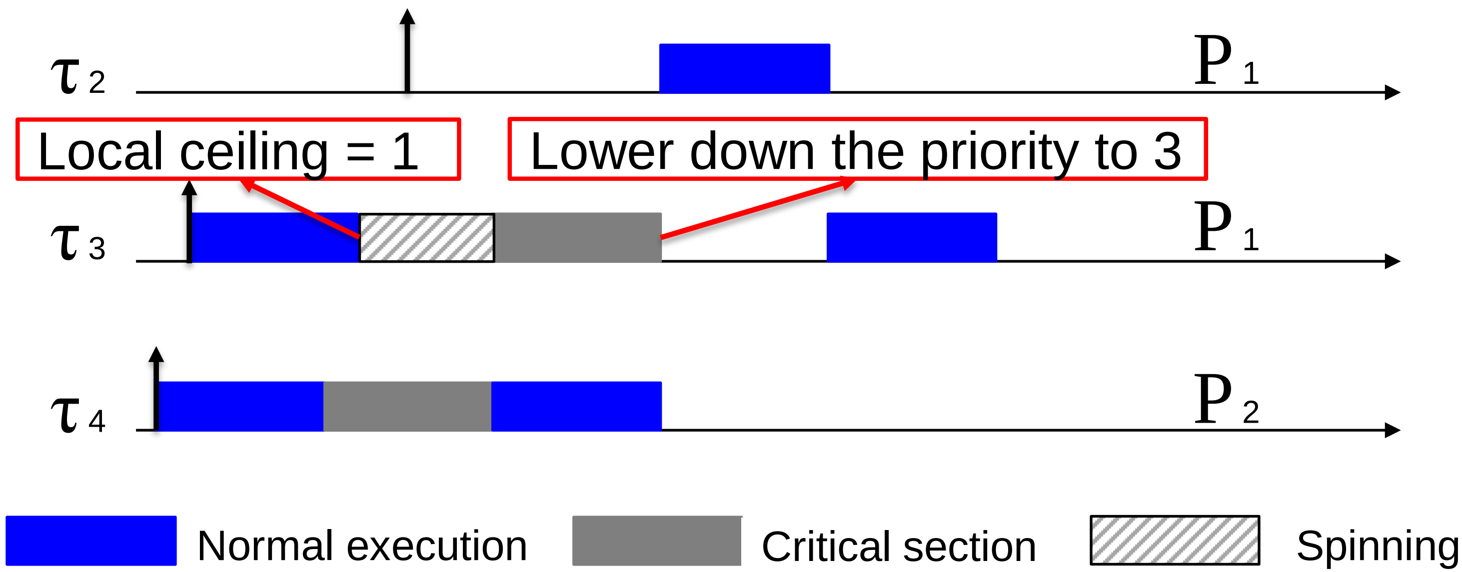
Implementation of MrsP

- Local ceilings are needed while spin waiting and executing
  - Local ceilings are given by users
  - Save the original priority before raising to the ceiling
  - Lower the priority after finishing the critical section



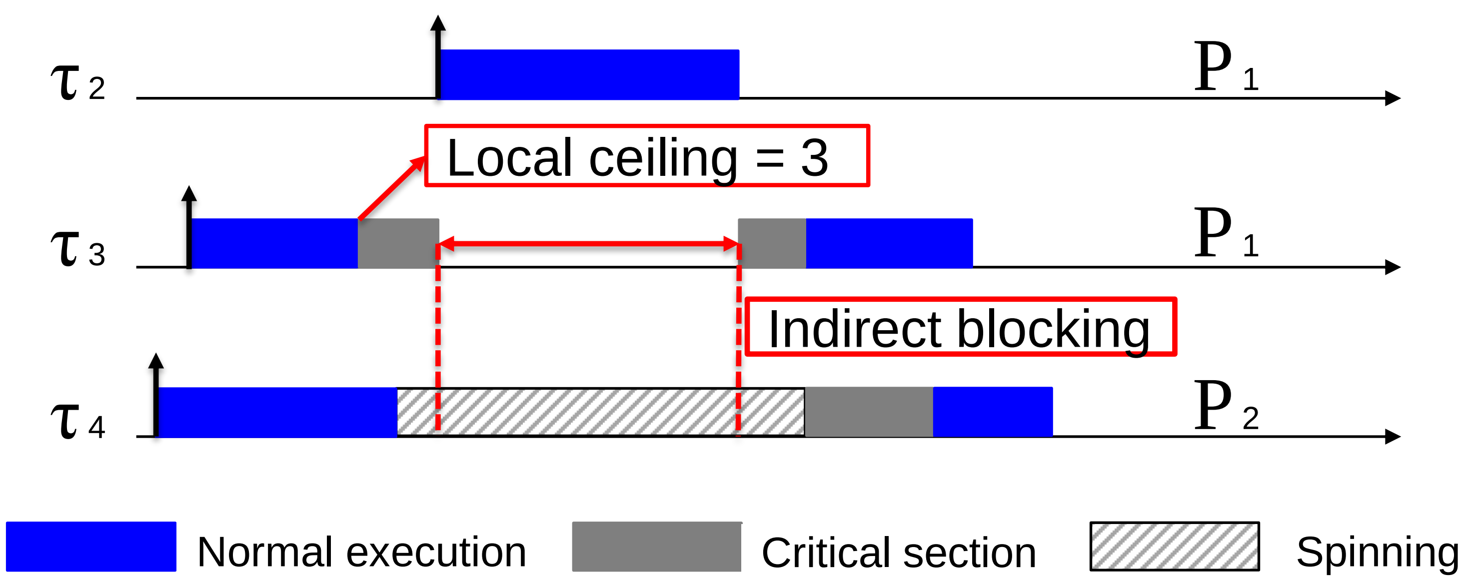
Implementation of MrsP

- Local ceilings are needed while spin waiting and executing
  - Local ceilings are given by users
  - Save the original priority before raising to the ceiling
  - Lower the priority after finishing the critical section



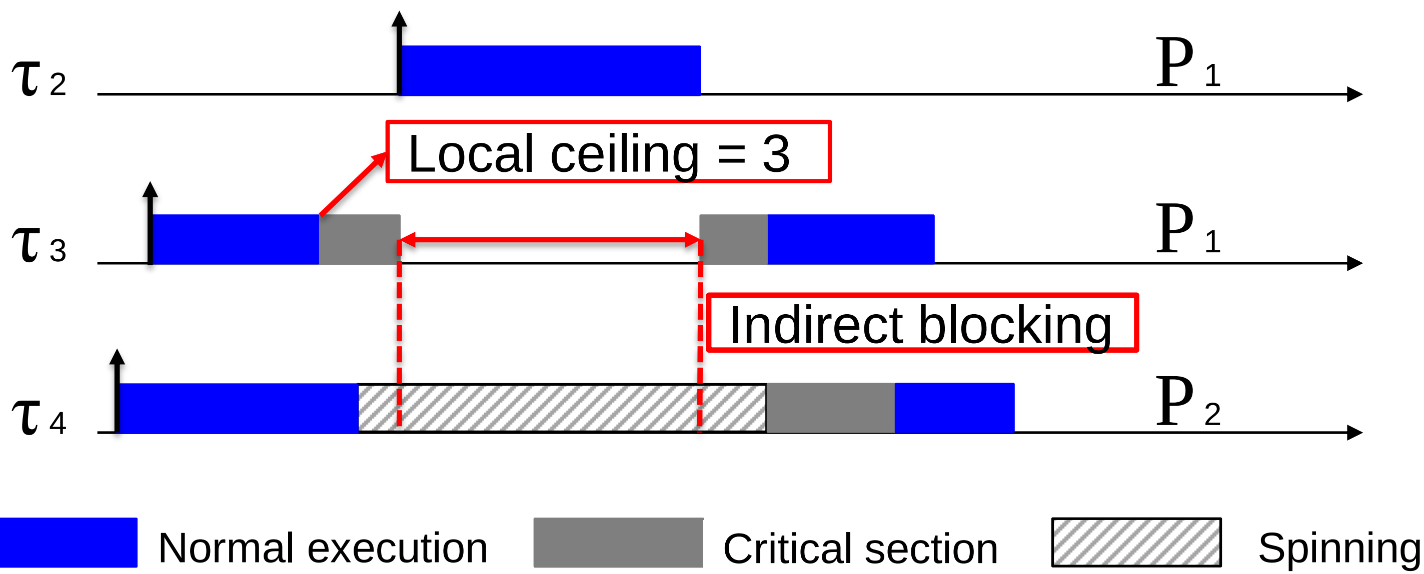
Implementation of MrsP

- Requests in FIFO queue
- Local ceilings are needed while spin waiting and executing



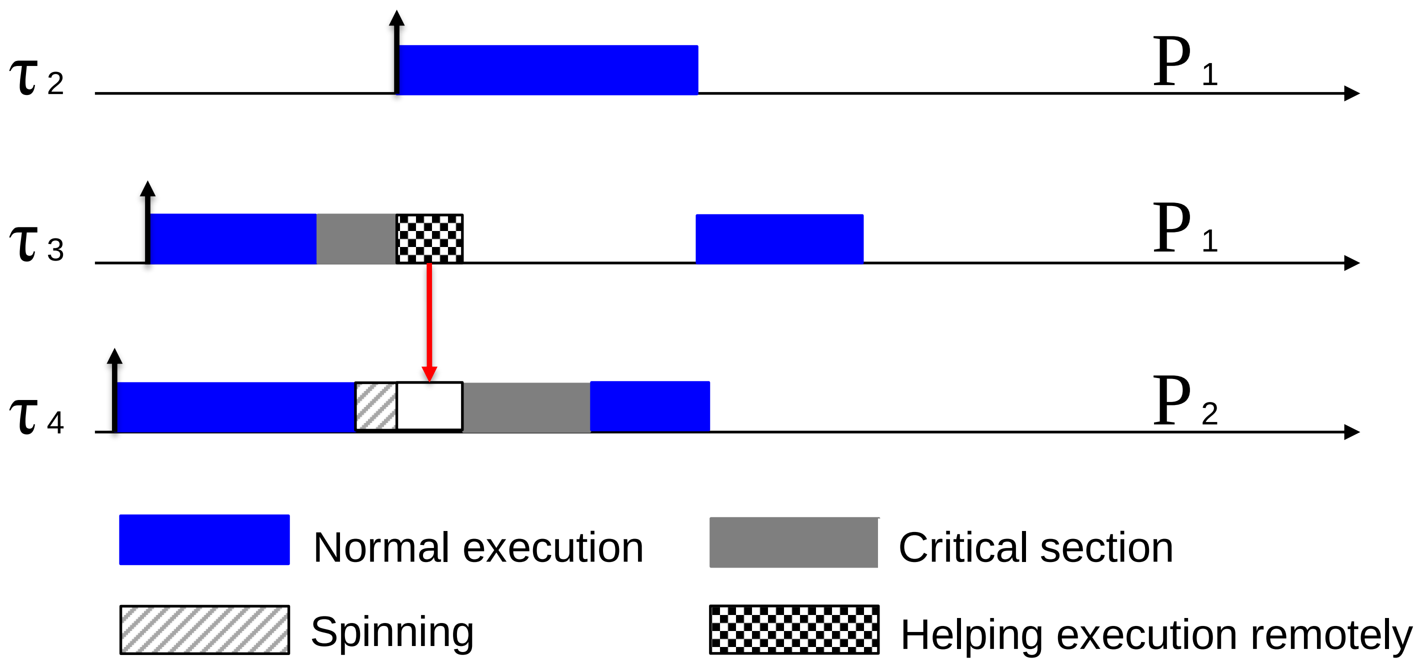
Implementation of MrsP

- Requests in FIFO queue
- Local ceilings are needed while spin waiting and executing
- Help mechanism
  - Spinning tasks help preempted semaphore owners



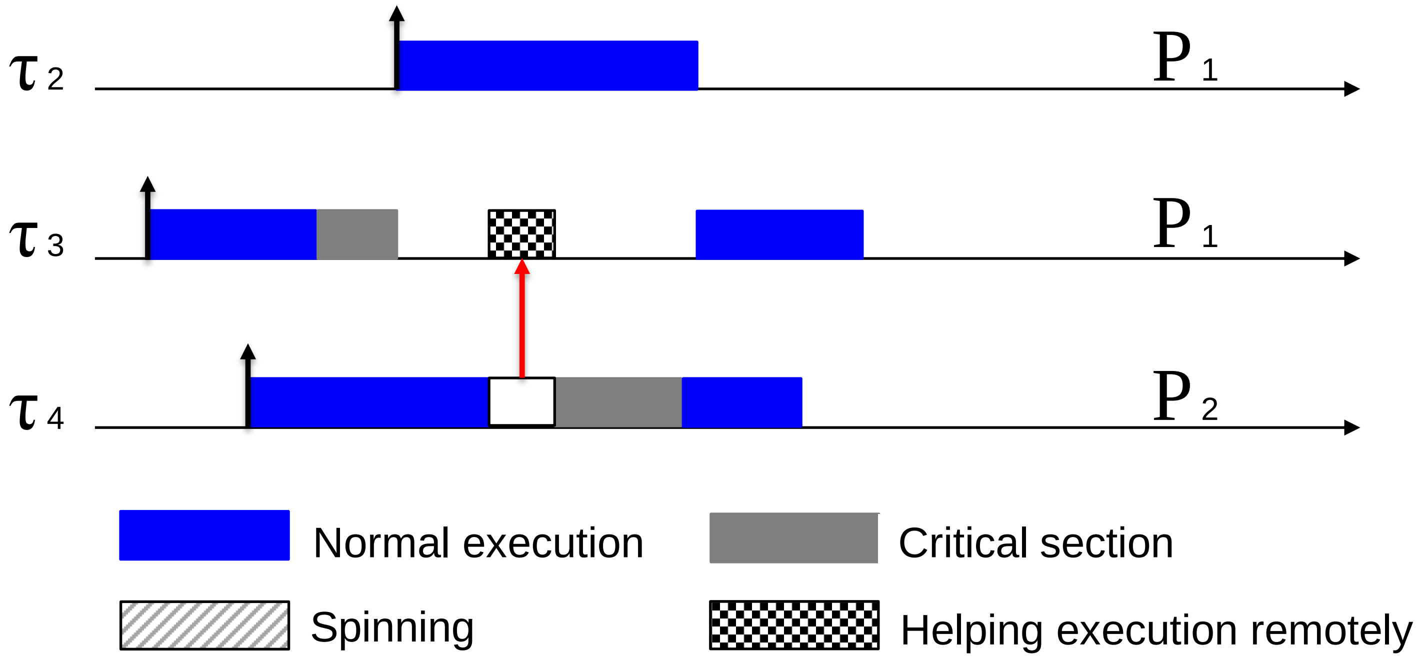
PUSH

- Semaphore owner can migrate to the helper's processor by itself after being preempted



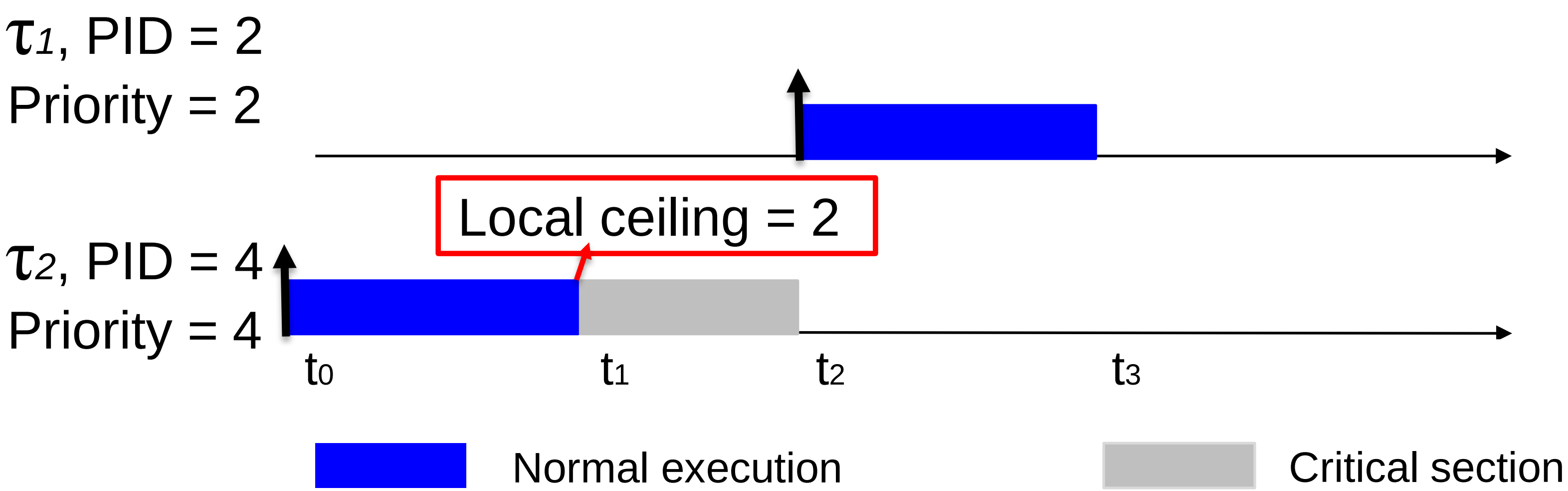
PULL

- Semaphore owner is on the ready queue, helper pulls it to the current processor



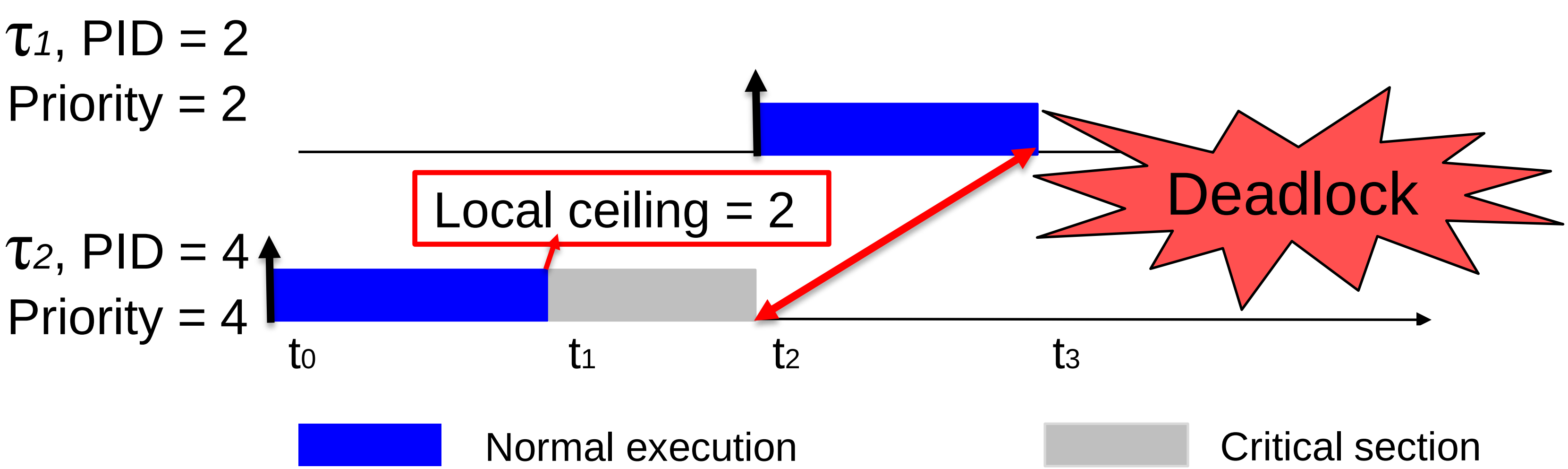
Potential implementation Deadlock

- Which task can be executed when there are two tasks have the same priority?
  - According to the PID number (first created first execute)



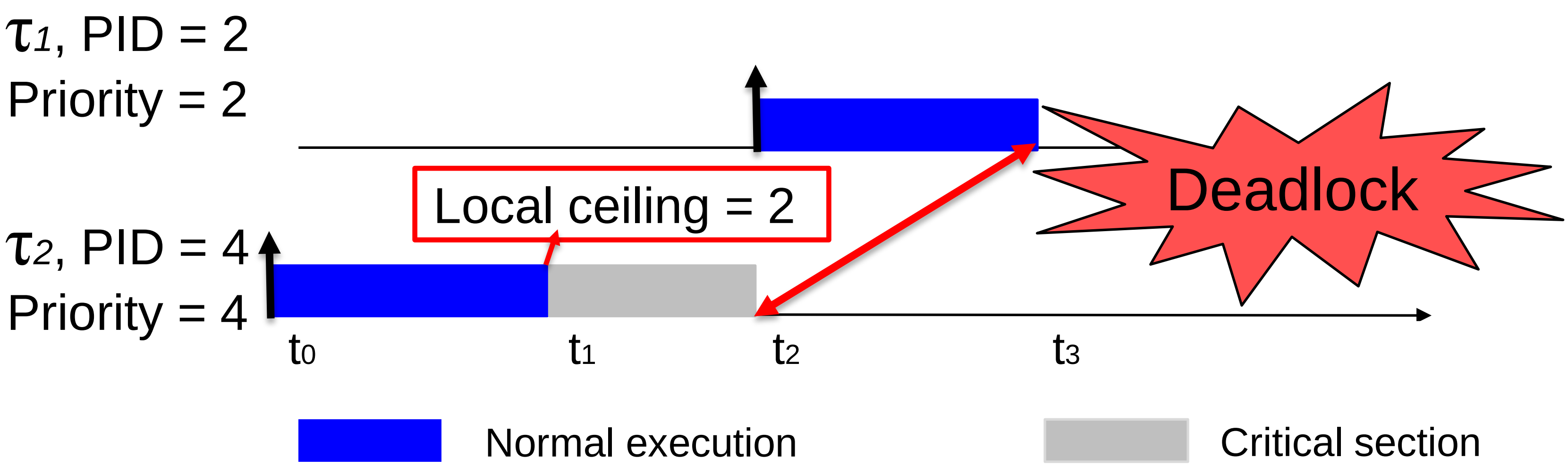
Potential implementation Deadlock

- Which task can be executed when there are two tasks have the same priority?
    - According to the PID number (first created first execute)



Potential implementation Deadlock

- Which task can be executed when there are two tasks have the same priority?
  - According to ~~the PID number (first created~~ first execute)
  - First executing task can continue its execution



Task set construction

- The number of tasks with different periods (40 tasks)

Period (ms)	5	10	20	50	100	200	1000
Num of tasks	2	5	8	10	8	5	2

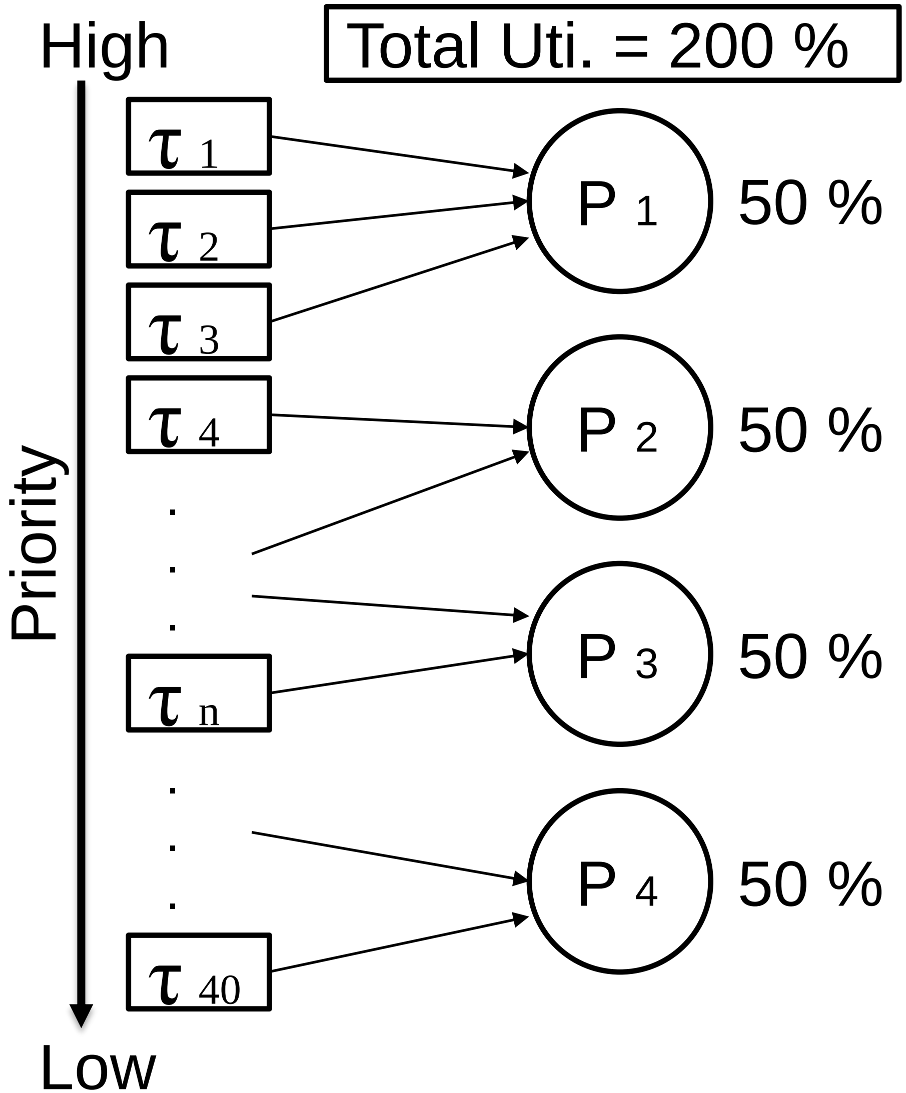
- Utilization for each task: 0.1 % - 10 %
- Total utilization (120 % - 280 %, step 40 %)
- Arithmetic progression is used
- Priorities are assigned under Rate-Monotonic
- BCET = 50 % * WCET, ACET = 90 % * WCET
- Normal distribution to generate real execution time

Shared resources

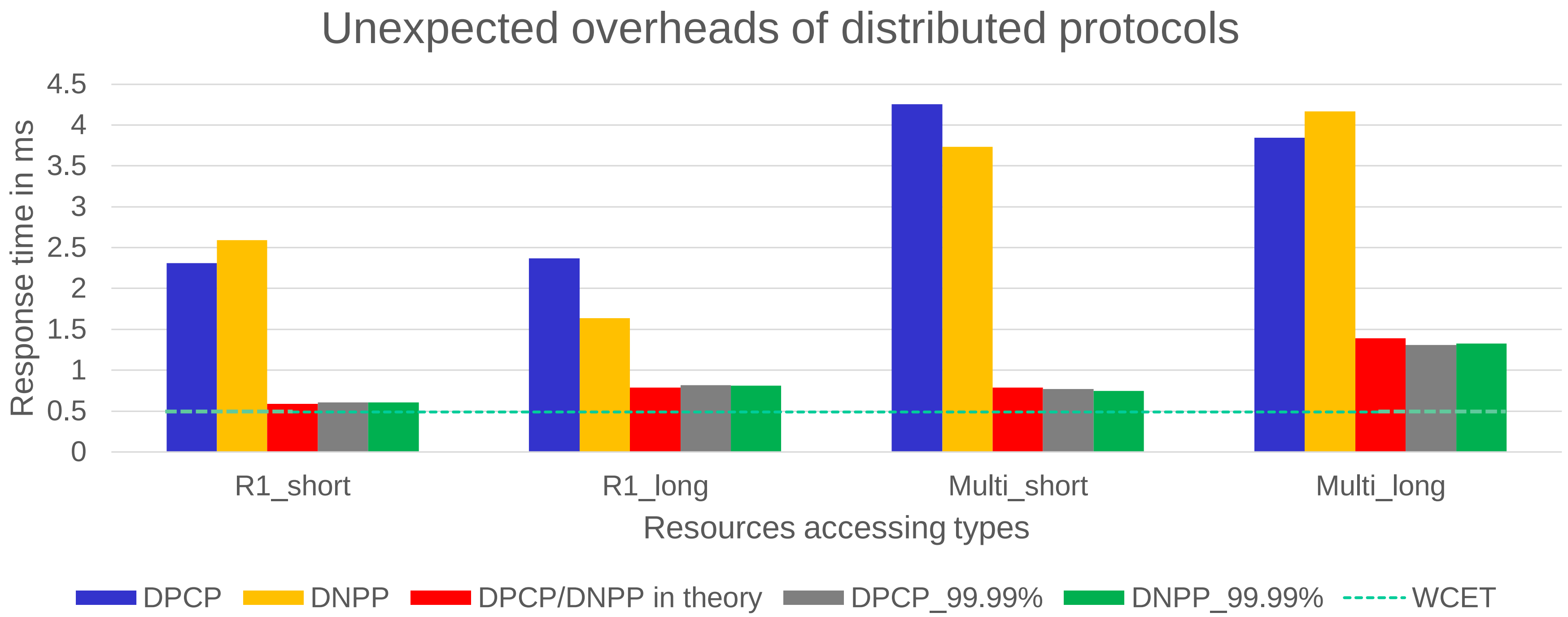
- Length of these shared resources
  - 0 us < R_{short} <= 100 us and 200 us < R_{long} <= 300 us
    - R_{short} = 20 % * execution time
    - Single: R_{long} = 80 % * execution time
    - Multi: R_{long} = 30 % * execution time
- Shared resources allocation
  - Request one short / long resource once
  - Request 3 short / long resources from 6 resources

Partition algorithm

- Sort tasks
- Calculate utilizations
- Allocate tasks



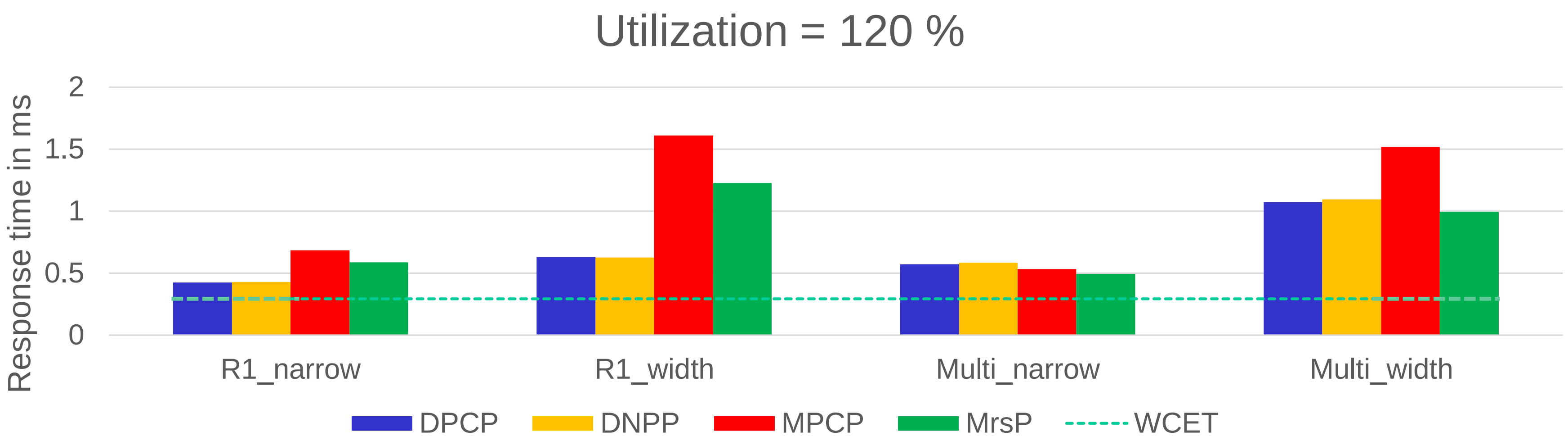
Overheads

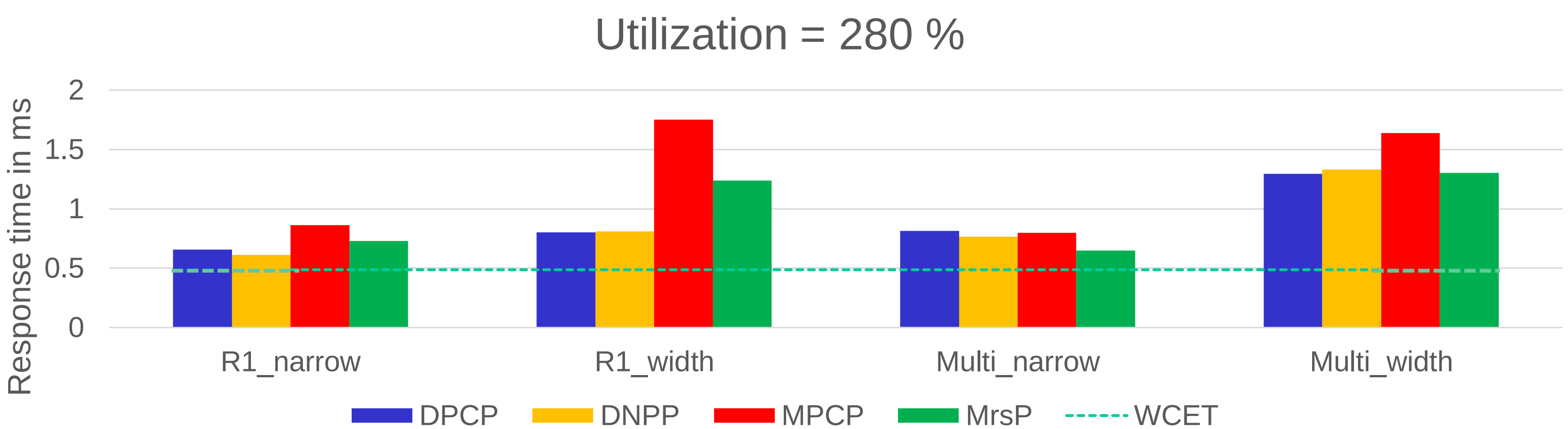


The results of the overheads measurement

	Migration	SCHED (MrsP)	Context switch
Average case	5.6 us	< 1 us	1.5 us

Experiment results





Conclusions and future work

- Conclusions
  - The first publicly available implementation of the MrsP
  - On $LITMUS^{RT}$, frequent migration may cause unexpected overhead
- Future work
  - Eliminate unexpected overhead of the DPCP/DNPP
  - When doing schedulability test, include the overheads and adopt proper partition algorithm.

Reference

[1] Burns, Alan, and Andy J. Wellings. "A Schedulability Compatible Multiprocessor Resource Sharing Protocol--MrsP." 2013 25th Euromicro Conference on Real-Time Systems. IEEE, 2013.

[2] Schmidt, Douglas C., et al. "Software architectures for reducing priority inversion and non-determinism in real-time object request brokers." Real-Time Systems 21.1-2 (2001): 77-125.

[3] Rajkumar, Ragunathan, Lui Sha, and John P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors." RTSS. Vol. 88. 1988.

[4] Kato, Shinpei, and Nobuyuki Yamasaki. "Semi-partitioned fixed-priority scheduling on multiprocessors." Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE. IEEE, 2009.

[5] Lakshmanan, Karthik, Ragunathan Rajkumar, and John Lehoczky. "Partitioned fixed-priority preemptive scheduling for multi-core processors." 2009 21st Euromicro Conference on Real-Time Systems. IEEE, 2009.

[6] B. B. Brandenburg and M. G¨ul. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In Real-Time Systems Symposium (RTSS), 2016 IEEE, pages 99–110. IEEE.

[7] Sha, Lui, Ragunathan Rajkumar, and John P. Lehoczky. "Priority inheritance protocols: An approach to real-time synchronization." IEEE Transactions on computers 39.9 (1990): 1175-1185.

Reference

[8] Calandrino, John M., et al. "LITMUS^ RT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers." 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06). IEEE, 2006.
[9] Solihin, Yan. Fundamentals of parallel computer architecture. Solihin Publishing and Consulting LLC, 2009.
[10] Kramer, S., Ziegenbein, D., Hamann, A.: Real world automotive benchmarks for free. In: Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS), July 2015.