Challenges in the implementation of MrsP

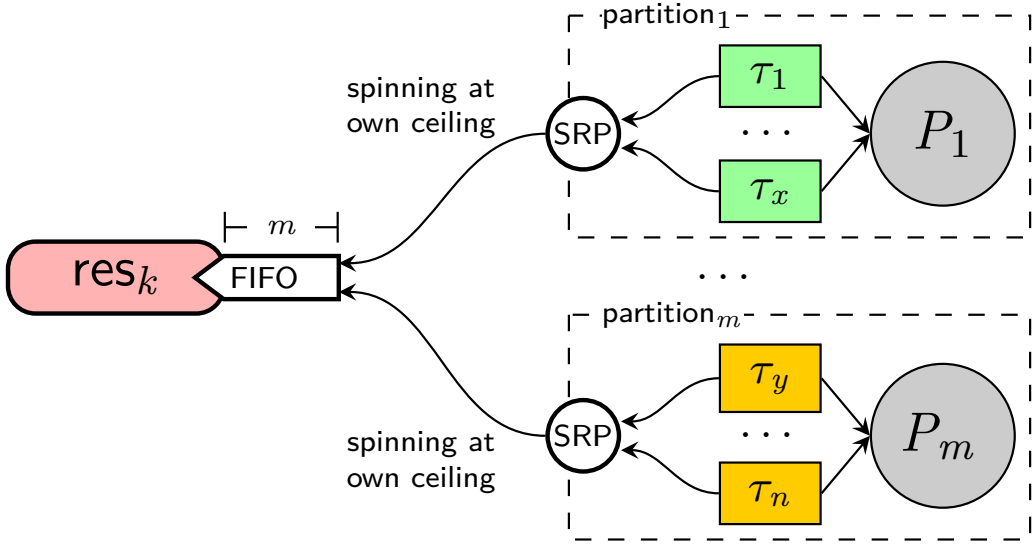
Sebastiano Catellani[1], Luca Bonato[1], Sebastian Huber[2], Enrico Mezzetti[1]

[1]University of Padua, Italy

[2]embedded brains GmbH, Germany

June 25. 2015

What is MrsP?



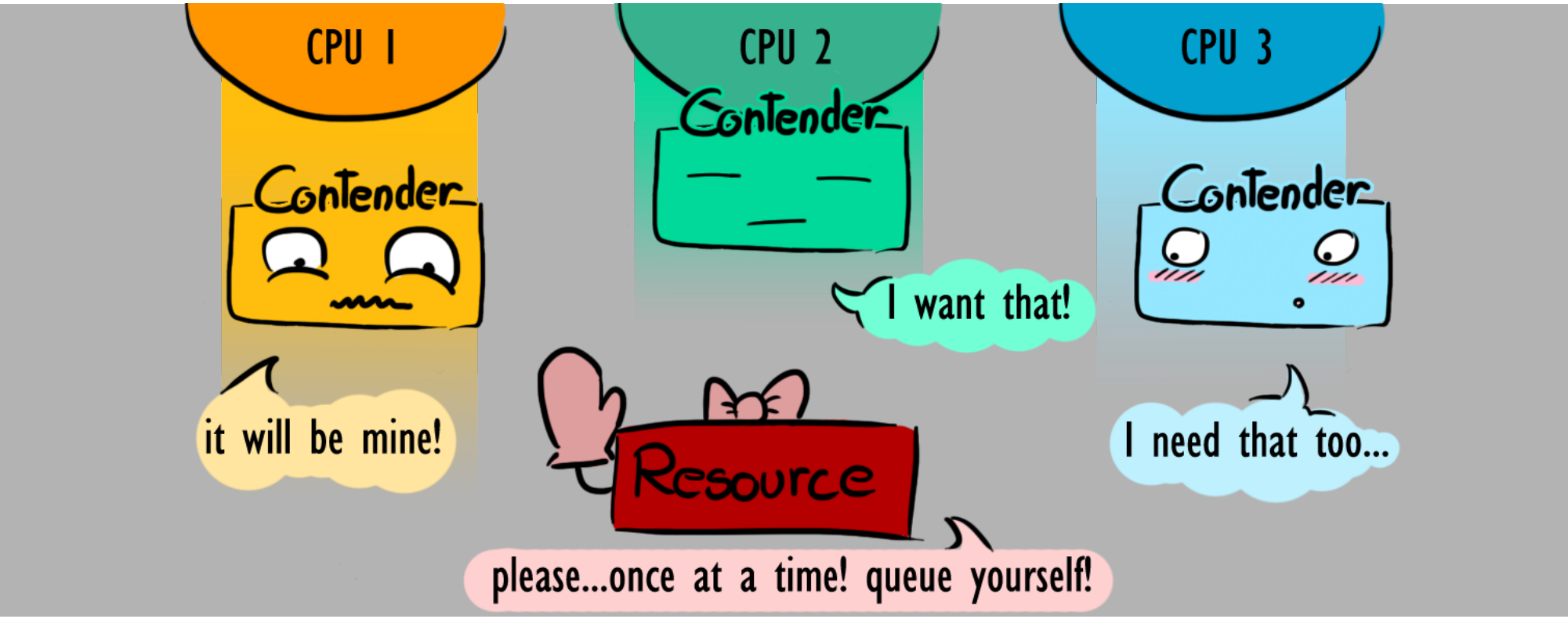
MrsP

MrsP (Multiprocessor resource sharing Protocol) is a semaphore-based protocol devised to work on SMP. It is based on:

1. FIFO ordering
2. busy wait at ceiling priority
3. helping mechanism

Why MrsP? - 1

- **optimal**
  - on SMP: critical section lenght $\propto$ number of processors



Why MrsP? - 2

- **RTA friendly**
  - designed to extend uniprocessor RTA

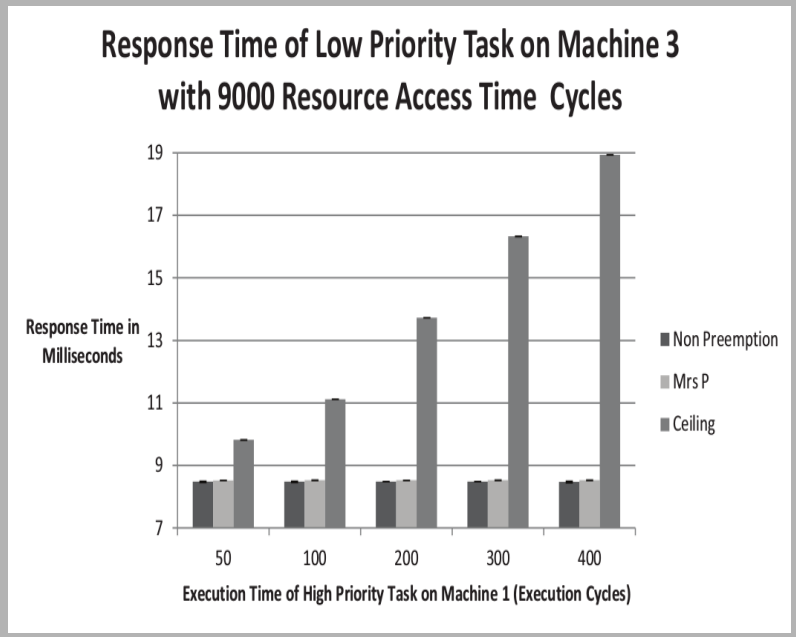
uniprocessor+SRP: $R_i = B_i + C_i + \sum_{j \in hp(i)} \lceil \frac{R_j}{T_j} \rceil C_j$

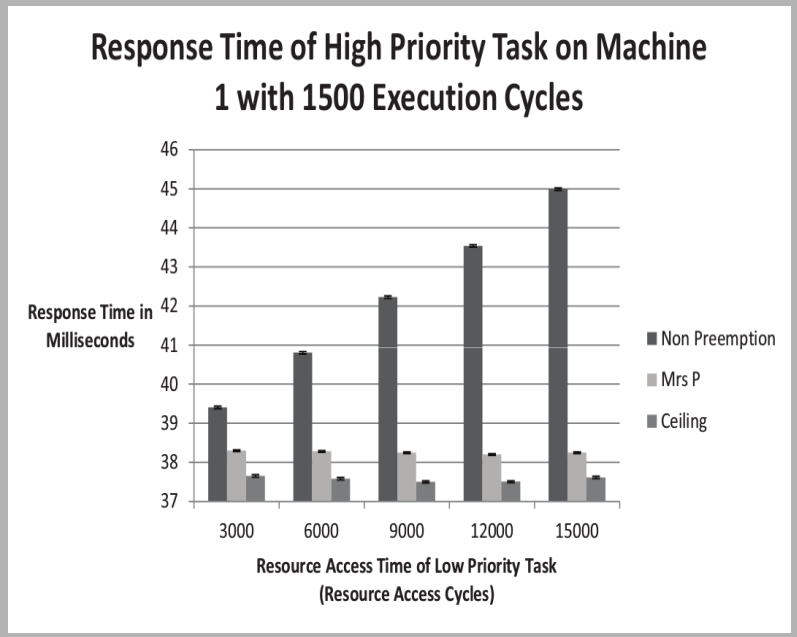
multiprocessor+MrsP: $R_i = \hat{B}_i + \hat{C}_i + \sum_{j \in hpl(i)} \lceil \frac{R_j}{T_j} \rceil C_j$

where $\hat{C}_i$ and $\hat{B}_i$ are inflated to account for the parallel contention (i.e., for the increased critical section length)

Why MrsP? - 3

- **behaves well**
  - vs simple ceiling and vs non-preemptive regions





from Burns&Wellings ECRTS13

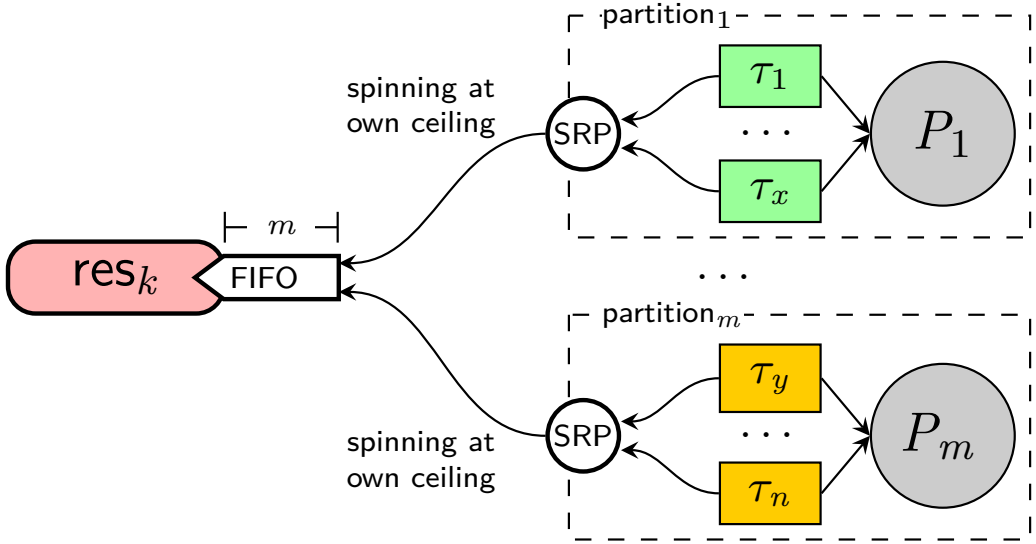
Our Goal

Original work (by Burns & Wellings ECRTS'13) offers end result of a proof of concepts implementation (built above the OS)

Can MrsP be implemented efficiently inside a RTOS on standard HW and SW support? Are there hidden problems?

We implemented it on 2 representative OSes

Recap on MrsP



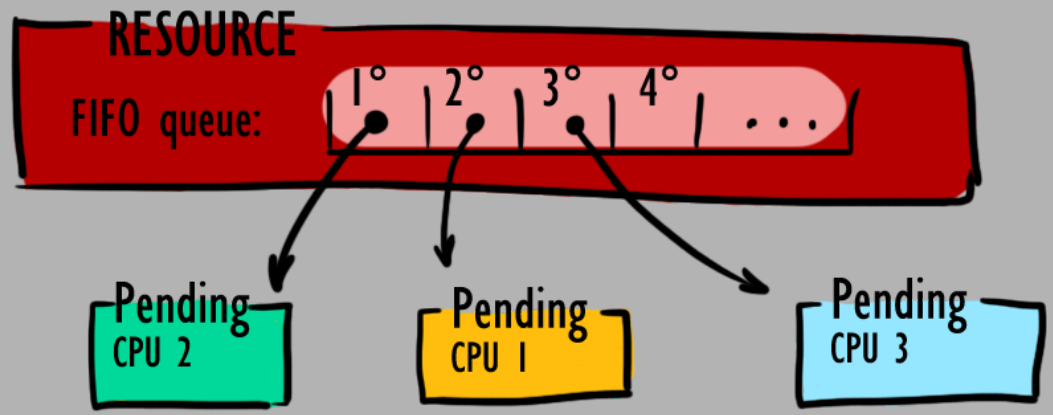
MrsP

MrsP (Multiprocessor resource sharing Protocol) is a semaphore-based protocol devised to work on SMP. It is based on:

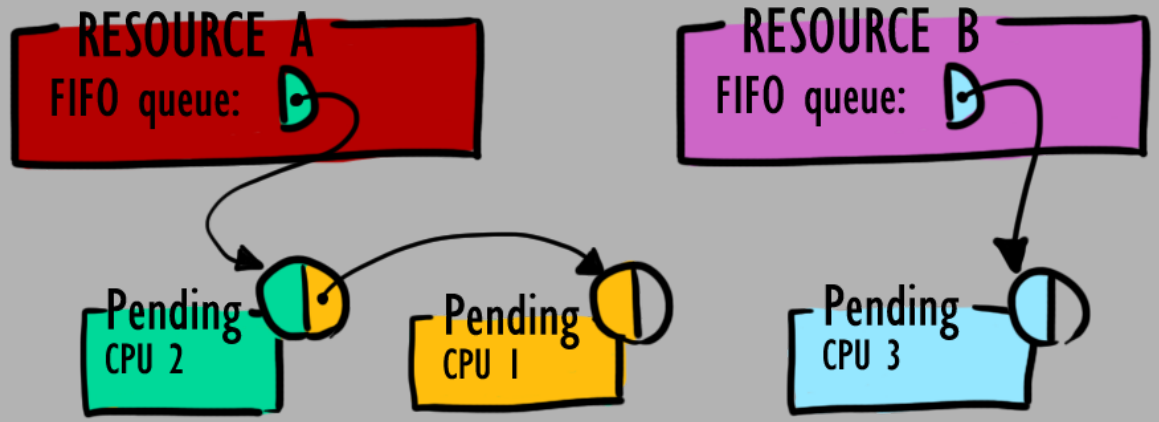
1. FIFO ordering
2. busy wait at ceiling priority
3. helping mechanism

FIFO ordering - how to implement the queue

- static list: size known a-priori

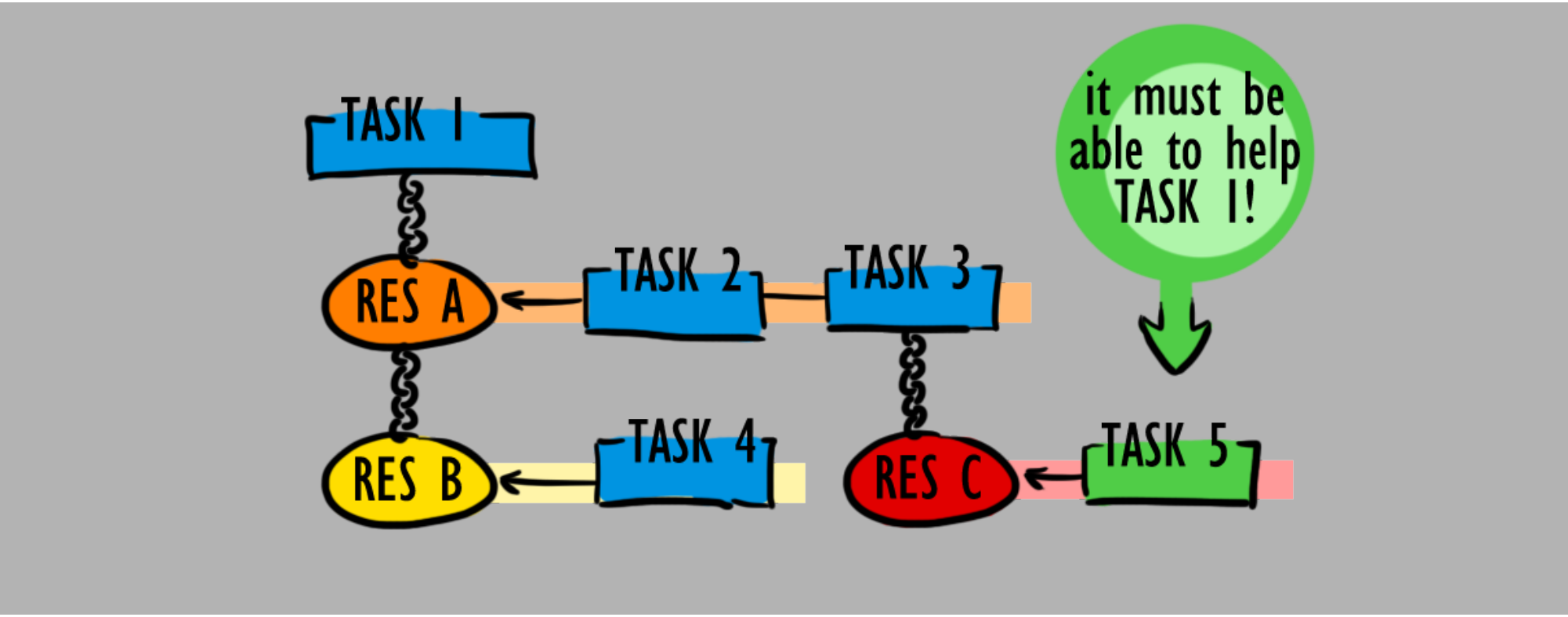


- dynamic list: one node per thread



FIFO ordering - how to manage nesting

Important to note: helping mechanism *must* be transitive!



- Nesting naturally suggests a tree structure
  - can be costly to maintain/inspect!

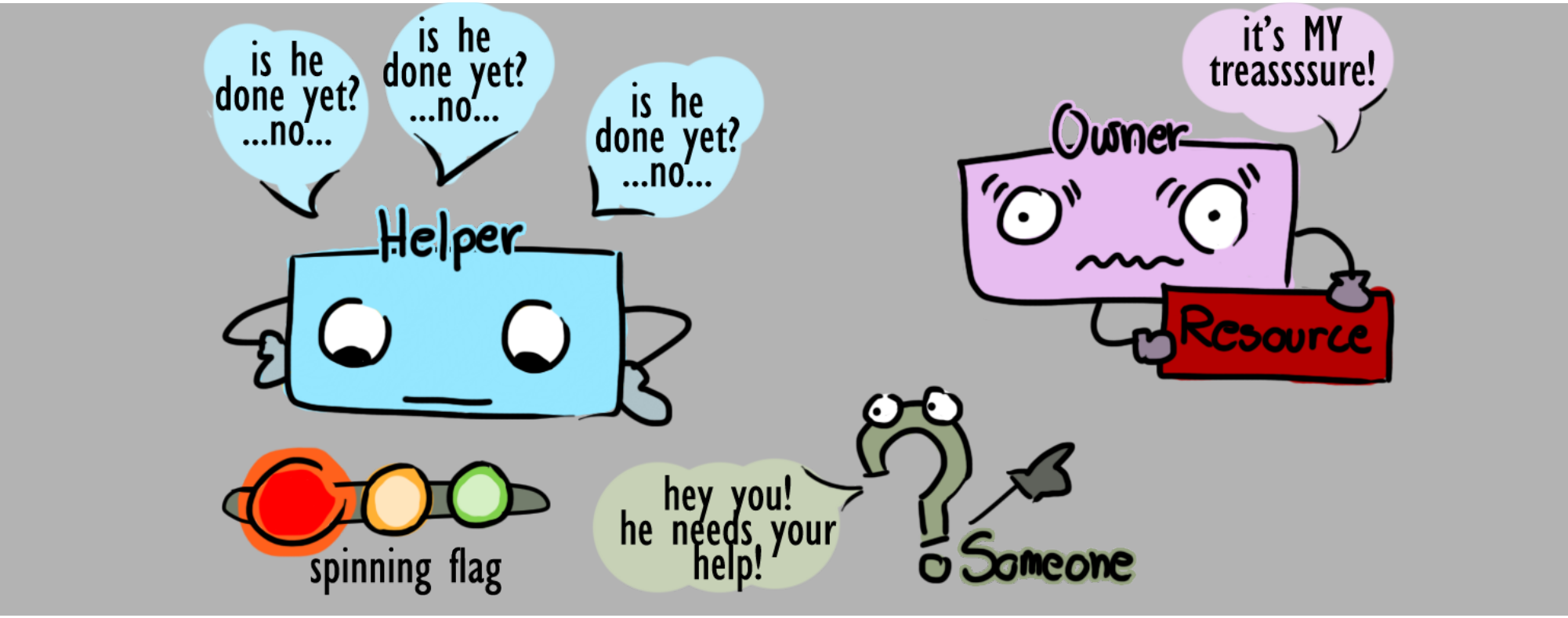
Busy wait at ceiling priority

- busy wait = spinning
  - lots of literature, trivial to implement
- no actual busy wait: just prevent lower priority tasks to execute
  - use a placeholder/idle thread at ceiling priority
  - block all lower priority tasks

Interesting part: *how to use* the busy wait?

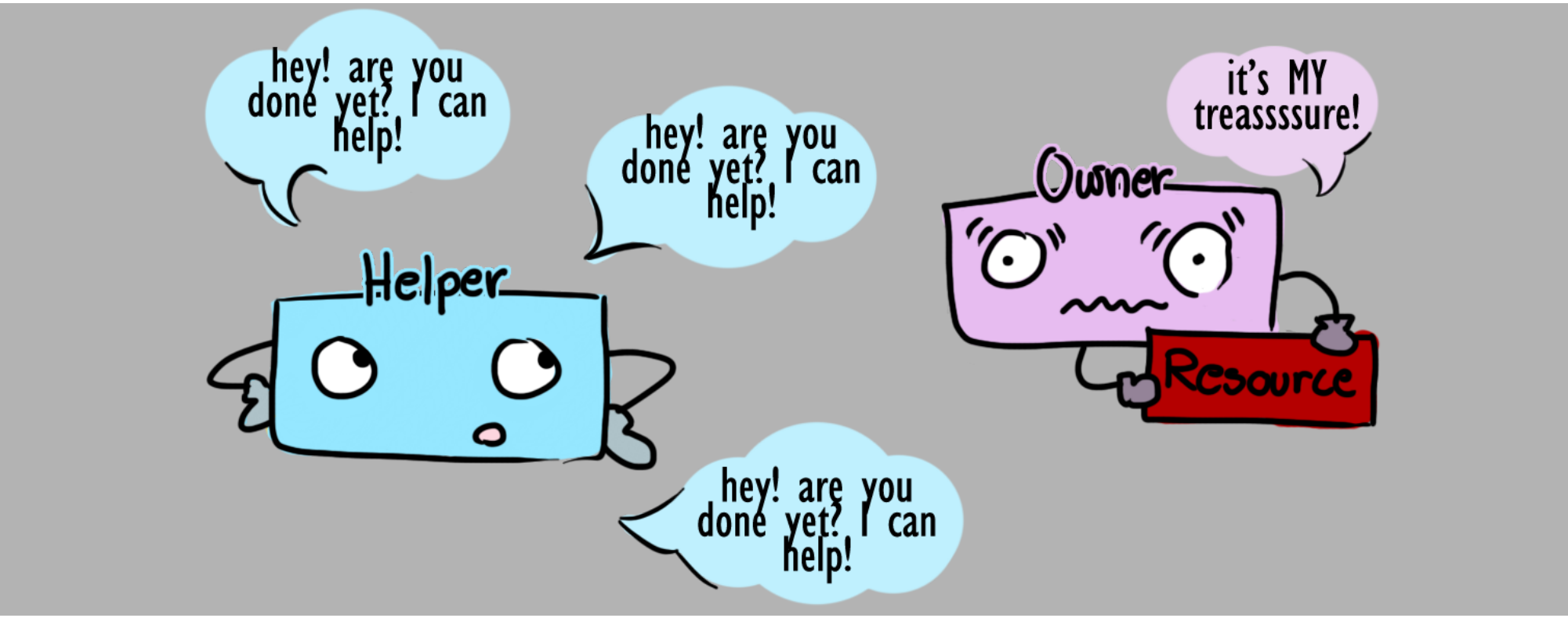
Busy wait: passive spinning

- Just delay the waiting task
  - can be replaced by placeholder
  - someone must decide when to help



Busy wait: active spinning

- Continuously check whether the resource holder is executing
  - polling on a single centralized state
  - can be used to trigger the helping mechanism



Helping mechanism - problem 1

MrsP invariant

resource holder must execute whenever there is a at least one task spinning on the same resource

MrsP invariant must be checked/enforced when:

1. obtaining/releasing the resource
2. resource holder is preempted
3. spinning task is resumed

Note: to trigger the helping mechanism for item 2-3 either:

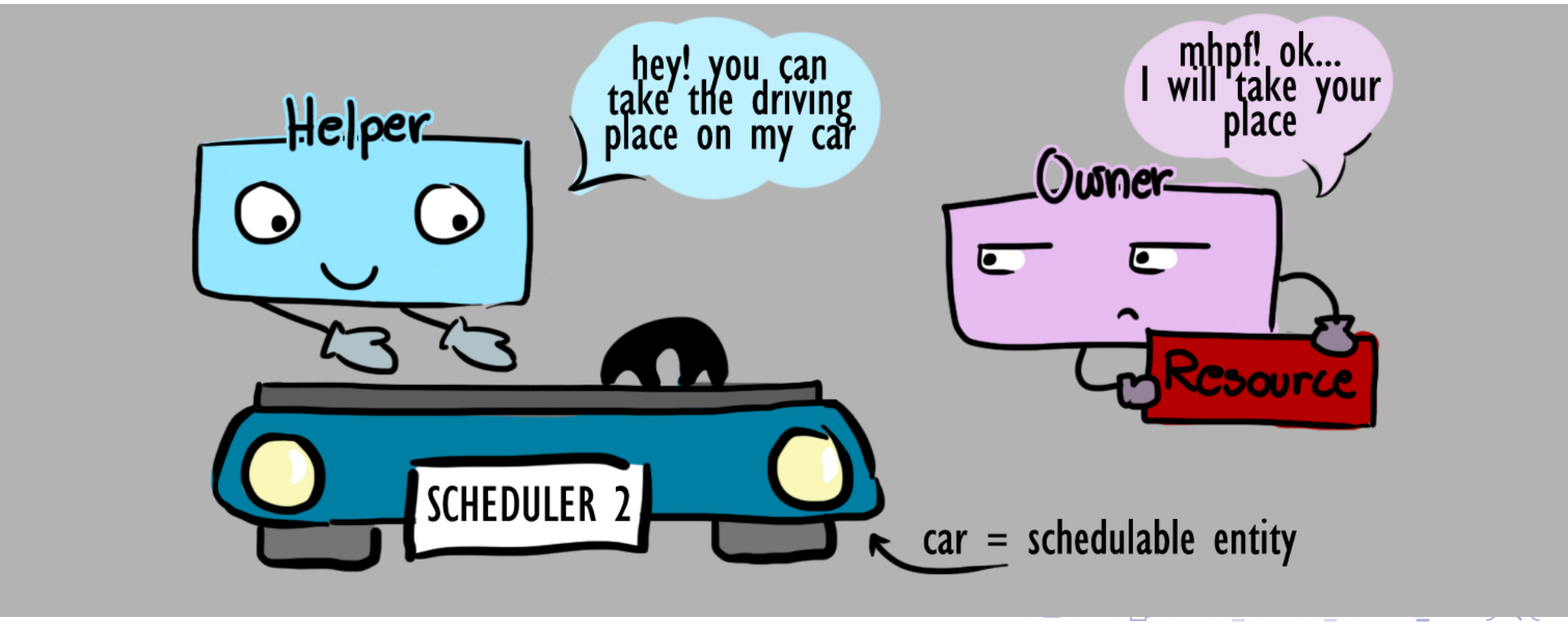
- the spinning task must be able to take action, or
- super partes entity must take action (the scheduler!)

Helping mechanism - problem 2

How to enforce the MrsP invariant?

Assuming migration as helping mechanism, the resource holder (helped) must evict the spinning task (helper)

- strictly dependent on scheduling framework
- halving effective priorities of the scheduler (using +1 when migrated)

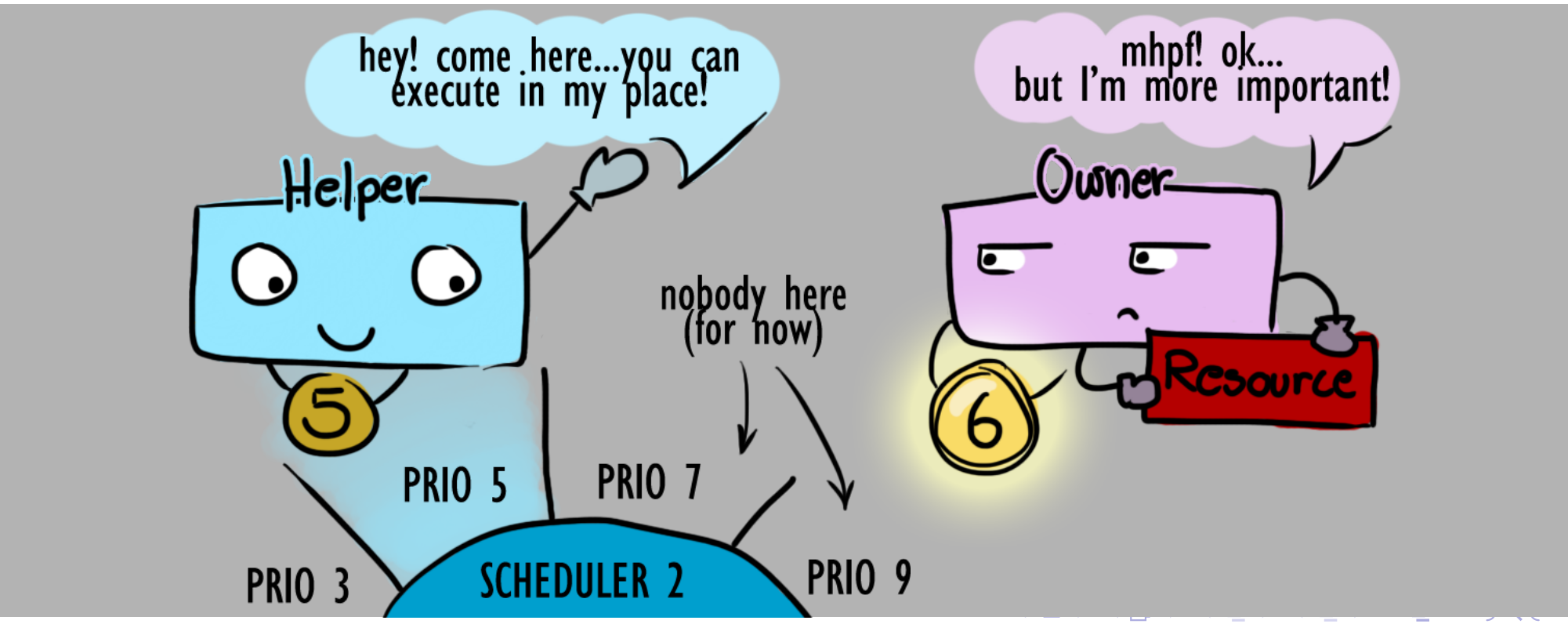


Helping mechanism - problem 2

How to enforce the MrsP invariant?

Assuming migration as helping mechanism, the resource holder (helped) must evict the spinning task (helper)

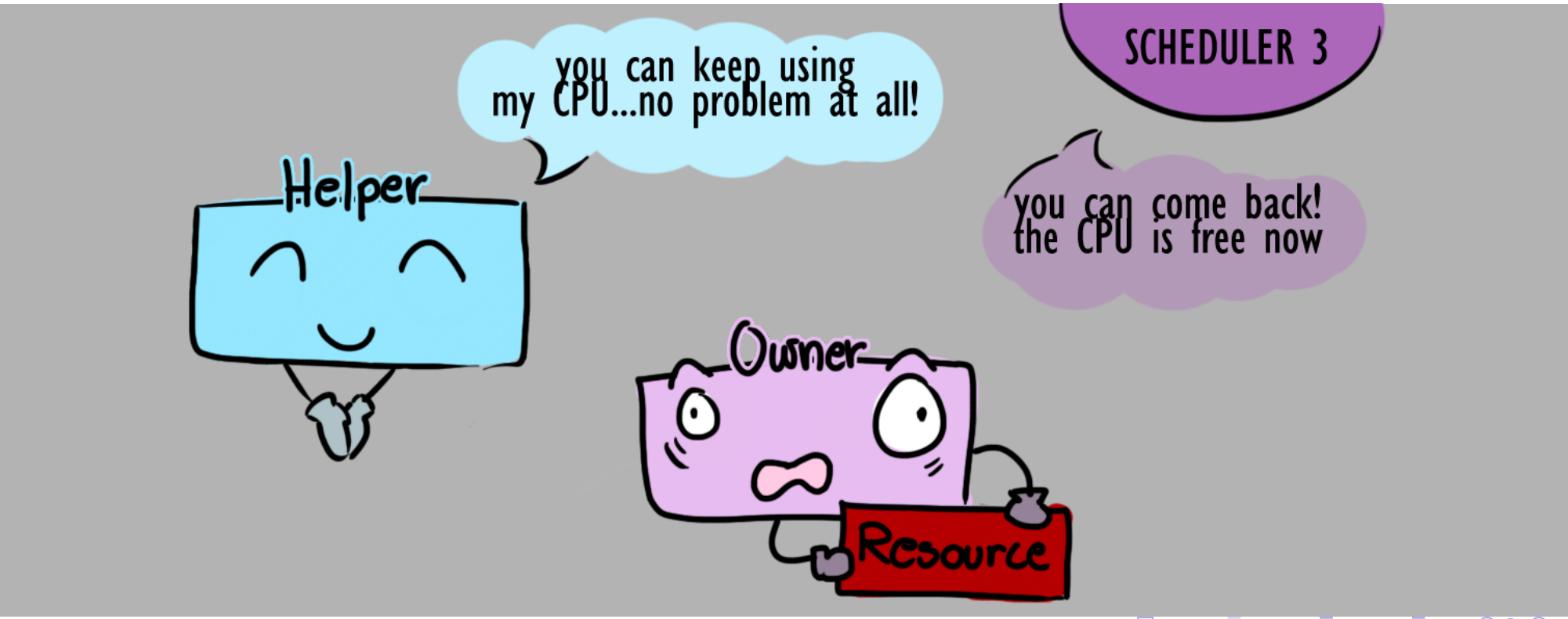
- strictly dependent on scheduling framework
- halving effective priorities of the scheduler (using +1 when migrated)



Helping mechanism - problem 3

What to do when the migrated task can go back to its own partition?

- migrate back :|
- use placeholder and keep executing :)
- suspend lower priority tasks :(



Our implementation

	litmus-rt	**rtems**
FIFO ordering	dynamic list	dynamic list
busy waiting	passive spinning ticket lock	passive spinning MCS lock
helping protocol	scheduler supervision half priorities use placeholder	scheduler supervision use scheduler node of helper use placeholder

Runtime behavior

1. how much does it cost to use a MrsP resource?
2. how much does it cost to have the MrsP infrastructure?
3. how much does it cost to support nested resources?

Runtime behavior - 1

how much does it cost to use a MrsP resource?

- to feed realistic overhead in RTA
- to know when it is not convenient to use MrsP

RTEMS		litmus-rt	
`obtain`	$5,376\ ns$	$8,800\ ns$	`lock`
`release`	$5,514\ ns$	$8,500\ ns$	`unlock`
`ask for help`	$1,827\ ns$	$35,000\ ns$	`finish switch`

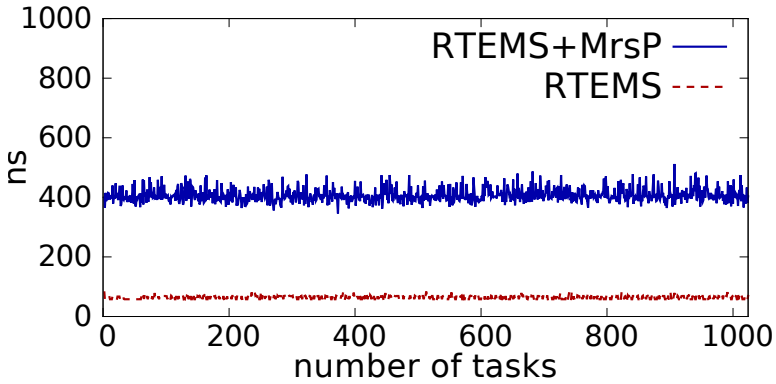
Runtime behavior - 2

how much does it cost to have the MrsP infrastructure?

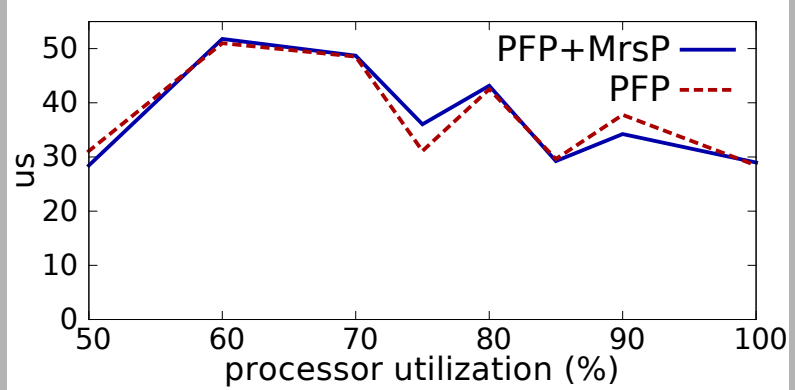
MrsP *must* interact with the scheduler because of the helping protocol

- to check if the overhead of the normal scheduling decisions is still acceptable

RTEMS - block operation



LITMS - schedule operation

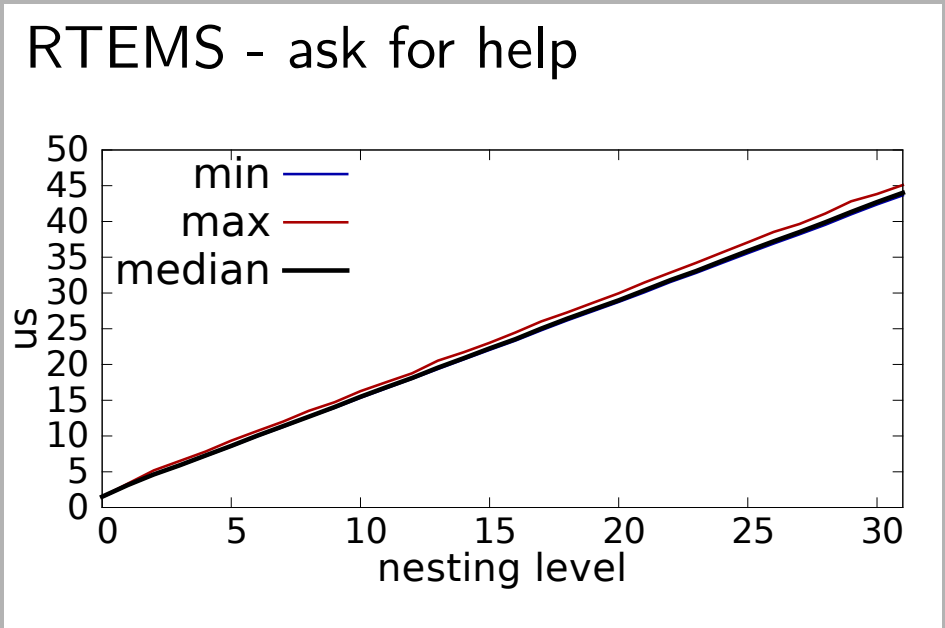


Runtime behavior - 3

how much does it cost to support nested resources?

Support for nested resources depends on the resource tree

- is its induced overhead acceptable?



Conclusions and future work

MrsP *can* successfully be implemented on standard RTOS

- implementation strictly related to the platform and RTOS support
- increased kernel overhead compensated by the improved RTA offered by MrsP

What to do next:

- further analyze MrsP overhead
- smartly account for the overhead in RTA
- develop a more efficent support for nested resources