blocking

uniprocessor blocking

When a task of lower base-priority is executing instead of a higher priority one.

How blocking impacts schedulability?

- s-oblivious: blocking counts as execution. Safe but unnecessarily pessimistic. Most schedulability test can use this technique. Example:

$$\sum_{\tau_i} \frac{e_i + B_i}{p_i} \leq 1$$

- s-aware: blocking does not count as execution. Safe and tight. Few schedulability test can use this technique. Example:

$$B_j + \sum_{\tau_i} \frac{e_i}{p_i} \leq 1 \qquad \forall \tau_j$$

blocking in multiprocessor

uniprocessor blocking $\neq$ multiprocessor blocking

- processors *can idle*
- access to resources is *parallel*

What to do while waiting for a locked resource?

- suspend: let other tasks execute
- spin: waiting and holding cpu

Not obvious which is best in multiprocessor

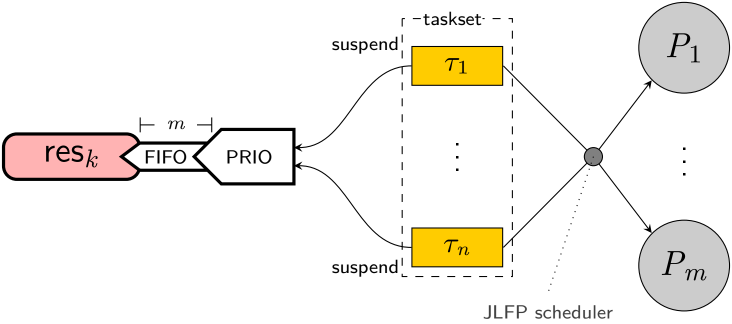
- changing priority or suspending blocked tasks *does not speed up* the release of resources (priorities *might not* have the same meaning across processors)
- spinning *does waste* cpu without letting any task progress

OMLP

$\mathcal{O}(m)$ *Locking Protocol*

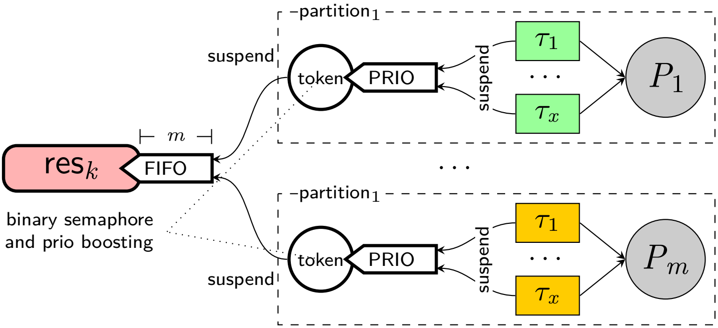
- suspension-based protocol for global and partitioned algorithms
- using queues to achieve optimal $\mathcal{O}(m)$ blocking
  - FIFO queue: serializing access to shared resource, preventing starvation
  - PRIO queue: speeding up higher priority tasks (lower prio tasks are not blocked if higher prio tasks are suspended)
- blocking term usable inside s-oblivious schedulability test
- nested resources only with group locks

OMLP - global



- blocking suffered only by tasks using resources
- per-request blocking is $b_k = 2(m-1)\omega_k$, ω_k length of max critical section for res_k
- all resources are global resources

OMLP - partitioned



- limiting access to global resources: per-partition *contention token*. Must be acquired before requesting *any* global resource (token + PRIO queue shared for all global resources)
- releasing resources as soon as possible: *priority boosting* for tasks queued in global resources (at most 1 per partition)

OMLP - partitioned

- three kinds of blocking:
  1. b^{prio}: caused by priority boosting (**any** task)

     $$b^{prio} = \max_k \{\omega_k\}$$

  2. b_k^{fifo}: caused by waiting in FIFO queue (only if using res$_k$)

     $$b_k^{fifo} = (m-1)\omega_k$$

  3. b^{trans}: caused by the contention token (if using any global res)

     $$b^{trans} = (m-1)\max_k \{\omega_k\}$$

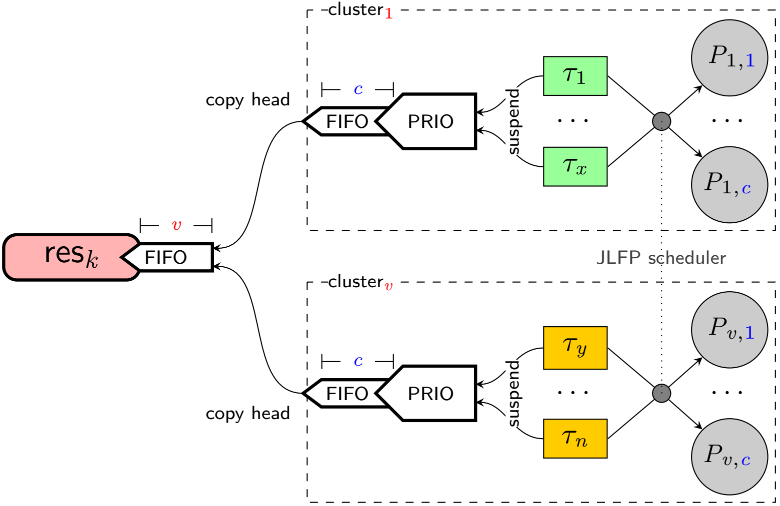
- tasks suffer (extensively) from unrelated critical sections
- some resources can be local (using ICPP/SRP)

OMIP

$\mathcal{O}(m)$ *Independence-preservation Protocol*

- for clustered algorithms
- suspension-based protocol
- generalizes OMLP-glob and OMLP-part (as clustered for global and partitioned schedulers)
- avoids shortcomings from OMLP-part by requiring *intra-cluster migration*
  - theorem: intra-cluster migrations are necessary to not suffer from unrelated critical sections
- blocking term usable inside s-oblivious schedulability tests

OMIP



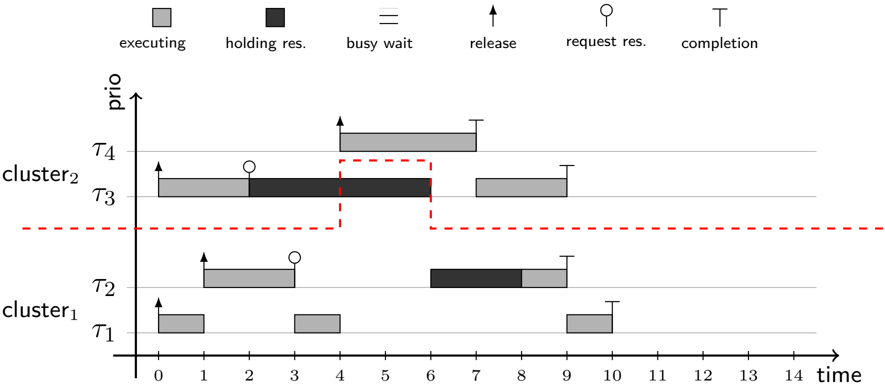
cluster$_1$
c
copy head
FIFO
PRIO
suspend
τ_1
$\cdots$
τ_x
$P_{1,1}$
$\cdots$
$P_{1,c}$
v
res$_k$
FIFO
JLFP scheduler
cluster$_v$
c
FIFO
PRIO
suspend
copy head
τ_y
$\cdots$
τ_n
$P_{v,1}$
$\cdots$
$P_{v,c}$

OMIP

- head of per-cluster FIFO queue participates in global FIFO queue
- each global resource has a private per-cluster FIFO+PRIO queues
- head of global FIFO queue can migrate and inherit priority of other tasks queued in global FIFO queue
- per-request blocking b_k (only if using res$_k$)

$$b_k = (2m - 1)\omega_k$$

OMIP - runtime example



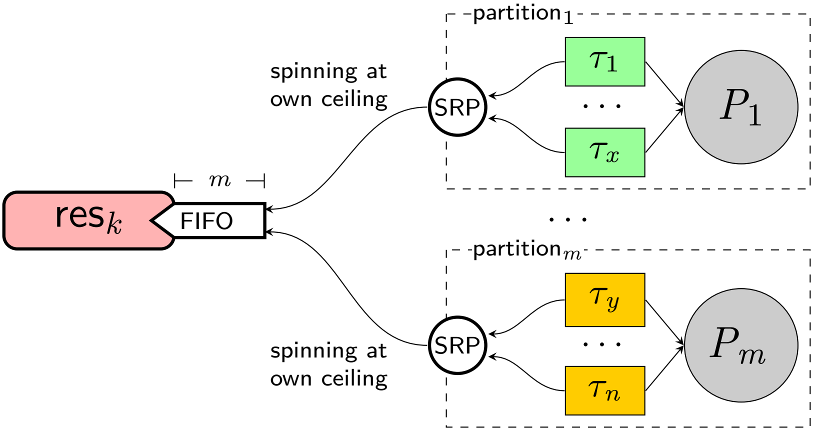
- $t = 3$: task τ_2 suspends and task τ_1 resumes execution
- $t = 4$: task τ_3 migrates to cluster$_1$ and preempts task τ_1

MrsP

Multiprocessor resource sharing Protocol

- for partitioned algorithms
- spinning-based protocol
- generalizes uniprocessor RTA
- assumes availability of helping mechanism
  - *task migration approach*: task migrates and executes in place of the "helper"
  - *duplicated execution approach*: assuming resources have internal status and their use does not produce side effects
- nesting by:
  - using resources always in the same order (avoid circular wait) and ad-hoc analysis
  - group locks

MrsP



partition$_1$
τ_1
τ_x
P_1
SRP
spinning at
own ceiling
m
res$_k$
FIFO
partition$_m$
τ_y
τ_n
P_m
SRP
spinning at
own ceiling

- worst-case resource usage must consider parallelism

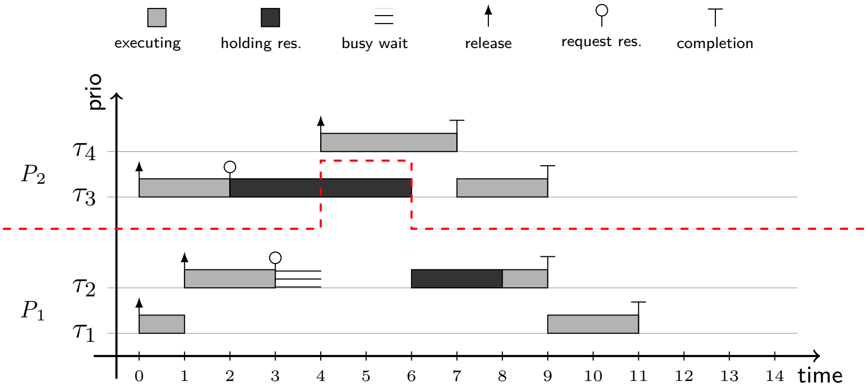
$$\hat{\omega}_k = m\omega_k \Rightarrow \hat{e}_i = e_i + \sum_{\mathsf{res}_k \in \tau_i} (m-1)\omega_k$$

- access to global resources through local SRP
- waiting for locked global resources by *spinning at local ceiling* (remaining preemptable)
- head of global FIFO queue, if preempted, executes in place of other queued spinning tasks
- per-partition RTA equation

$$R_i = \hat{e}_i + B_i + \sum_{hp(\tau_i)} \left\lceil \frac{R_i}{p_j} \right\rceil \hat{e}_j,$$

$B_i =$ uniprocessor SRP-like blocking term using $\hat{\omega}_k$

MrsP - runtime example



- $t = 3$: task τ_2 start spinning at ceiling priority
- $t = 4$: task τ_3 migrates to P_1 and executes in place of τ_2