The FMLP+

An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis

ECRTS’14
July 9, 2014
Suspension-Based Locking Protocols

Semaphores in POSIX

```
pthread_mutex_lock(…)
// critical section
pthread_mutex_unlock(…)
```

**semaphore:**
a waiting task suspends,
makes processor available to other tasks
Suspension-Based Locking Protocols

Semaphores in POSIX

- `pthread_mutex_lock(…)
  // critical section
  pthread_mutex_unlock(…)

Semaphores:
- a waiting task suspends,
  makes processor available to other tasks

Priority Inversion Blocking
- Locks cause priority inversions
  \approx extra delay due to lock contention
- Short: pi-blocking
Suspension-Based Locking Protocols

Semaphores in POSIX

```c
pthread_mutex_lock(…)
// critical section
pthread_mutex_unlock(…)
```

**Semaphore:**
- A waiting task suspends, makes processor available to other tasks

**Priority Inversion Blocking**
- Locks cause priority inversions
  - ≈ extra delay due to lock contention
- Short: pi-blocking

**Blocking Analysis**
- For a specific task set, what is the maximum duration of pi-blocking incurred by each task?
Suspension-Based Locking Protocols

Semaphores in POSIX

```c
pthread_mutex_lock(…)
// critical section
pthread_mutex_unlock(…)
```

Semaphore: a waiting task suspends, makes processor available to other tasks

Priority Inversion Blocking
- Locks cause priority inversions
  - ≈ extra delay due to lock contention
  - Short: pi-blocking

Blocking Analysis
- For a **specific task set**, what is the maximum duration of pi-blocking incurred by each task?

Blocking Optimality
- In general, what is the **maximum duration of pi-blocking** incurred by any task in any task set?
# Multiprocessor Real-Time Locking Optimality Classes

<table>
<thead>
<tr>
<th>Blocking Optimality</th>
<th>suspension oblivious</th>
<th>suspension aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>[— &amp; Anderson, 2010]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How are suspensions analyzed?</th>
<th>CPU demand is over-approximated</th>
<th>CPU demand is modeled accurately</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Advantage</th>
<th>simpler analysis</th>
<th>potentially less pessimistic</th>
</tr>
</thead>
</table>

# Asymptotically Optimal Locking Protocols

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>job-level fixed-priority</td>
<td>Any JLFP Scheduler</td>
<td>EDF w/ Implicit Deadlines</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

**Partitioned**  
(no migrations)  

**Global**  
(jobs migrate freely)  

**Clustered**  
(jobs migrate only among subset of processors)  

---

[Block et al., 2007]  
[— & Anderson, 2010]  
[— & Anderson, 2011]  
[—, 2011]  
[Ward & Anderson, 2012]  
[—, 2013]  

Optimality Results for Multiprocessor Real-Time Locking, RTSS 2010.  
# Asymptotically Optimal Locking Protocols

<table>
<thead>
<tr>
<th>JLFP: job-level fixed-priority</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any JLFP Scheduler</td>
<td>EDF w/ Implicit Deadlines</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

### Partitioned (no migrations)
- **P-OMLP**
  - [— & Anderson, 2010](#)

### Global (jobs migrate freely)
- **G-OMLP**
  - [— & Anderson, 2010](#)

### Clustered (jobs migrate only among subset of processors)
- **OMIP**
  - [—, 2013](#)
- **C-OMLP**
  - [— & Anderson, 2011](#)

---

**References**

## Asymptotically Optimal Locking Protocols

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>job-level fixed-priority</td>
<td>Any JLFP Scheduler</td>
<td>EDF w/ Implicit Deadlines</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

### Partitioned
- (no migrations)
- **P-OMLP**
  - [— & Anderson, 2010](#)

### Global
- (jobs migrate freely)
- **G-OMLP**
  - [— & Anderson, 2010](#)

### Clustered
- (jobs migrate only among subset of processors)
- **OMIP**
  - [—, 2013](#)
- **C-OMLP**
  - [— & Anderson, 2011](#)

### SPFP
- (asymptotical tightness)
  - [— & Anderson, 2010](#)

### P-FMLP*
- (practical protocol)
  - [—, 2011](#)

### Other Protocols
<table>
<thead>
<tr>
<th><strong>OMIP</strong></th>
<th><a href="#">— &amp; Anderson, 2010</a></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C-OMLP</strong></td>
<td><a href="#">— &amp; Anderson, 2011</a></td>
</tr>
</tbody>
</table>

### References
- [Block et al., 2007](#)
- [— & Anderson, 2010](#)
- [— & Anderson, 2011](#)
- [—, 2011](#)
- [Ward & Anderson, 2012](#)
- [—, 2013](#)
# Asymptotically Optimal Locking Protocols

<table>
<thead>
<tr>
<th>JLFP (job-level fixed-priority)</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any JLFP Scheduler</td>
<td>EDF w/ Implicit</td>
<td>Any JLFP Scheduler</td>
</tr>
<tr>
<td></td>
<td>Deadlines</td>
<td></td>
</tr>
</tbody>
</table>

| Partitioned (no migrations)    | P-OMLP                | SPFP            |
|                                | [— & Anderson, 2010]  | (asymptotical tightness) |
|                                |                      | [— & Anderson, 2010] |
|                                | **P-FMLP**+ (practical protocol) | [—, 2011] |
|                                |                      |                 |
| Global (jobs migrate freely)   | G-OMLP                | FMLP            |
|                                | [— & Anderson, 2010]  | [Block et al., 2007] |
|                                | **OMIP**              |                 |
|                                | [—, 2013]             |                 |
|                                | C-OMLP                |                 |
|                                | [— & Anderson, 2011]  |                 |

[Block et al., 2007]
[— & Anderson, 2010]
[— & Anderson, 2011]
[—, 2011]
[ward & Anderson, 2012]
[—, 2013]

---


**Optimality Results for Multiprocessor Real-Time Locking**, RTSS 2010.


### Asymptotically Optimal Locking Protocols

<table>
<thead>
<tr>
<th>JLFP (job-level fixed-priority)</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partitioned</strong> (no migrations)</td>
<td>Any JLFP Scheduler</td>
<td>EDF w/ Implicit Deadlines</td>
<td>Any JLFP Scheduler</td>
</tr>
<tr>
<td></td>
<td><strong>P-OMLP</strong> ✓</td>
<td><strong>SPFP</strong> (asymptotical tightness) [— &amp; Anderson, 2010]</td>
<td><strong>P-FMLP</strong>+ (practical protocol) [—, 2011]</td>
</tr>
<tr>
<td><strong>Global</strong> (jobs migrate freely)</td>
<td><strong>G-OMLP</strong> ✓</td>
<td><strong>FMLP</strong> [Block et al., 2007]</td>
<td>⚠</td>
</tr>
<tr>
<td></td>
<td><strong>OMIP</strong> ✓ [—, 2013]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Clustered</strong> (jobs migrate only among subset of processors)</td>
<td><strong>C-OMLP</strong> [— &amp; Anderson, 2011]</td>
<td>⚠</td>
<td>⚠</td>
</tr>
</tbody>
</table>

**Support for nested critical sections added by RNLP.** [Ward & Anderson, 2012]

---

- **P-OMLP** [— & Anderson, 2010]
- **SPFP** (asymptotical tightness) [— & Anderson, 2010]
- **P-FMLP**+ (practical protocol) [—, 2011]
- **G-OMLP** [— & Anderson, 2010]
- **FMLP** [Block et al., 2007]
- **OMIP** [—, 2013]
- **C-OMLP** [— & Anderson, 2011]

---


## Asymptotically Optimal Locking Protocols

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>job-level fixed-priority</strong></td>
<td>Any JLFP Scheduler</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

### Partitioned
(no migrations)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P-OMLP</strong></td>
<td>✓</td>
</tr>
<tr>
<td>[— &amp; Anderson, 2010]</td>
<td></td>
</tr>
<tr>
<td>+ RNLP [Ward &amp; Anderson, 2012]</td>
<td></td>
</tr>
</tbody>
</table>

### Global
(jobs migrate freely)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G-OMLP</strong></td>
<td>✓</td>
</tr>
<tr>
<td>[— &amp; Anderson, 2010]</td>
<td></td>
</tr>
<tr>
<td>+ RNLP [Ward &amp; Anderson, 2012]</td>
<td></td>
</tr>
</tbody>
</table>

### Clustered
(jobs migrate only among subset of processors)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OMIP</strong></td>
<td>✓</td>
</tr>
<tr>
<td>[—, 2013]</td>
<td></td>
</tr>
<tr>
<td><strong>C-OMLP</strong></td>
<td>✓</td>
</tr>
<tr>
<td>[— &amp; Anderson, 2011]</td>
<td></td>
</tr>
<tr>
<td>+ RNLP [Ward &amp; Anderson, 2012]</td>
<td></td>
</tr>
</tbody>
</table>

### This Work
**The Generalized FMLP⁺**
(FIFO Multiprocessor Locking Protocol)

- [Block et al., 2007]
- [— & Anderson, 2010]
- [— & Anderson, 2011]
- [—, 2011]
- [Ward & Anderson, 2012]

Optimality Results for Multiprocessor Real-Time Locking, RTSS 2010.
Remainder of This Talk

What is Suspension-Aware PI-Blocking?
- Assumptions & quick review

Finding a Suitable Progress Mechanism
- How to deal with lock-holder preemptions

Closing the Suspension-Aware Optimality Gap
- New progress mechanism: restricted segment boosting
- Achieving asymptotic optimality with the generalized FMLP+
Assumptions & Review of Suspension-Aware PI-Blocking
System Model

Clustered Scheduling

- disjoint clusters of processors
  - special cases: partitioned & global
  - job-level fixed-priority policy (JLFP)
    - e.g., EDF, static task priorities
  - cluster size may be non-uniform
System Model

Clustered Scheduling
- disjoint clusters of processors
  - special cases: partitioned & global
  - job-level fixed-priority policy (JLFP)
    - e.g., EDF, static task priorities
- cluster size may be non-uniform

Sporadic Tasks
- arbitrary deadlines
- shared resources
  - in the paper: also nested CSs
  - in the talk: only unnested CSs
- locking-unrelated self-suspensions
Definition: S-Aware PI-Blocking

A job \( J \) assigned to a cluster with \( c \) CPUs incurs \textit{s-aware pi-blocking} at a time \( t \) iff

1. \( J \) is \textit{not scheduled} at time \( t \), and
2. fewer than \( c \) higher-priority jobs are scheduled.

Intuition

→ Locking-related delays are \textit{not problematic} iff \( J \) would \textit{not} have been scheduled anyway…

Maximum PI-Blocking

\( b_i \) — bound on max. pi-blocking incurred by task \( T_i \)
\[ \text{max } \{ b_i \} \] — maximum **pi-blocking** of any task in task set
Maximum Pi-Blocking

\( b_i \) — bound on max. pi-blocking incurred by task \( T_i \)

\[ \max \{ b_i \} \] — maximum pi-blocking of any task in task set

There exist task sets such that under s-aware analysis

\[ \max \{ b_i \} = \Omega(n) \]

under any suspension-based locking protocol.

[— & Anderson, 2010] (assuming constant critical section lengths)

Maximum Pi-Blocking

\[ b_i \] — bound on max. pi-blocking incurred by task \( T_i \)

\[ \max \{ b_i \} \] — maximum pi-blocking of any task in task set

There exist task sets such that under s-aware analysis

\[ \max \{ b_i \} = \Omega(n) \]

under any suspension-based locking protocol.

\[ \text{[— & Anderson, 2010]} \]

(assuming constant critical section lengths)

\[ \rightarrow O(n) \] max. s-aware pi-blocking is asymptotically optimal.

\[ \text{[— & Anderson, 2010] Optimality Results for Multiprocessor Real-Time Locking, RTSS 2010.} \]
Objective

Define a locking protocol such that

\[ \max \{b_i\} = O(n) \]

for any task set under any clustered JLFP scheduler.

Need to define queue order
- FIFO works

Need to define progress mechanism
- To deal with risk of lock-holder preemption
- Ensure timely completion of critical sections
- Classic example: priority inheritance
Finding a Suitable Progress Mechanism
### Progress Mechanism Choices

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partitioned</strong></td>
<td>Any JLFP Scheduler</td>
<td>EDF w/ Implicit Deadlines</td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td>P-OMLP</td>
<td>SPFP (asymptotical tightness)</td>
</tr>
<tr>
<td><strong>Clustered</strong></td>
<td>G-OMLP</td>
<td>FMLP</td>
</tr>
<tr>
<td></td>
<td>[— &amp; Anderson, 2010]</td>
<td>[Block et al., 2007]</td>
</tr>
</tbody>
</table>

The classic choice: (variants of) **priority inheritance**.
Progress Mechanism Choices

(1) The classic choice: (variants of) priority inheritance.

- P-OMLP
  [— & Anderson, 2010]

- SPFP (asymptotical tightness)
  [— & Anderson, 2010]

- P-FMLP* (practical protocol)
  [—, 2011]

(2) The partitioned & clustered choice: (variants of) priority boosting.

- G-OMLP
  [— & Anderson, 2010]

- FMLP
  [Block et al., 2007]

- OMIP
  [—, 2013]

- C-OMLP
  [— & Anderson, 2011]
Sub-Optimality of Priority Inheritance

It is impossible to construct an asymptotically optimal locking protocol (w.r.t. s-aware analysis) under global JLFP scheduling based on priority inheritance.

(And hence also under clustered JLFP scheduling.)

Sub-Optimality of Priority Inheritance

Priority Inheritance Example Schedule

4 tasks on 2 processors

*global fixed-priority scheduling*

Task priorities (high to low): $T_1 > T_2 > T_3 > T_4$
Sub-Optimality of Priority Inheritance

![Diagram showing scheduled times, critical sections, job release, deadline, job completion, and job suspended events.](image-url)
Sub-Optimality of Priority Inheritance

$T_4$: short period, long deadline ($d_4 > p_4$), shares lock with $T_2$
Sub-Optimality of Priority Inheritance

$T_4$: short period, long deadline ($d_4 > p_4$), shares lock with $T_2$

$T_2$ & $T_1$: short period, implicit deadline ($d_{1,2} = p_{1,2}$)

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>scheduled</th>
<th>critical section</th>
<th>job release</th>
<th>job suspended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor 2</td>
<td></td>
<td></td>
<td></td>
<td>job completed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor 2</th>
<th>job suspended</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Processor 2</th>
<th>priority inversion</th>
</tr>
</thead>
</table>

Legend:
- $\uparrow$: job release
- $\downarrow$: deadline
- $\uparrow\downarrow$: job completion
- $\equiv$: job suspended
- $\equiv$: priority inversion
Sub-Optimality of Priority Inheritance

- $T_4$: short period, long deadline ($d_4 > p_4$), shares lock with $T_2$
- $T_3$: long period, implicit deadline, “victim task”
- $T_2$ & $T_1$: short period, implicit deadline ($d_{\{1,2\}} = p_{\{1,2\}}$)

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduled</td>
<td>critical section</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time

$T_4$, $T_3$, $T_2$, $T_1$
Independent Job Incurs Priority Inversion

![Diagram showing independent job incurs priority inversion](image-url)
Independent Job Incurs Priority Inversion

*T4* acquires lock first...

…but is preempted by arrival of higher-priority jobs.

...but is preempted by arrival of higher-priority jobs.
Independent Job Incurs Priority Inversion

$T_4$ acquires lock first...

...and $T_2$ also requires the lock.

scheduled

job suspended

Processor 1

Processor 2

deadline

job completion

priority inversion

Processor 1

Processor 2

$T_4$

$T_3$

$T_2$

$T_1$
The FMLP+: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis

Independent Job Incurs Priority Inversion

Priority inheritance: $T_4$ inherits from $T_2$

$\rightarrow$ $T_3$ is preempted, incurs s-aware pi-blocking

Processor 1

Processor 2

scheduled critical section

job release deadline job completion

job suspended priority inversion
Independent Job Incurs Priority Inversion

How often can this scenario repeat?

How often can this scenario repeat?
"Victim Task" Accumulates PI-Blocking

\[ \Omega(\phi) \text{ pi-blocking is possible, where } \phi = \frac{\text{max response time}}{\text{min period}} \]

The diagram illustrates the accumulation of priority inversion (PI) blocking over time for different tasks. The tasks are labeled as \( T_1, T_2, T_3, T_4 \) and each task has a critical section. The processor states are labeled as Processor 1 and Processor 2, with the scheduled time slot and critical section for each processor shown. The diagram shows the job release, job completion, job suspension, and priority inversion events for each task.
"Victim Task" Accumulates PI-Blocking

\( \Omega(\phi) \text{ pi-blocking is possible, where } \phi = \frac{\text{max response time}}{\text{min period}} \)

Bounded only by the number of jobs released by \( T_1, T_2, \) and \( T_4 \) while \( T_3 \) is pending.

How many jobs? \( \Rightarrow \phi = \frac{\text{max response time}}{\text{min period}} \)
Sub-Optimality of Priority Inheritance

$\Omega(\phi)$ pi-blocking is possible, where $\phi = \frac{\text{max response time}}{\text{min period}}$

$\phi$ is not bounded by the number of tasks $n$.

$\rightarrow$ not asymptotically optimal.
What about Priority Boosting?

(1) The classic choice: (variants of) priority inheritance.

<table>
<thead>
<tr>
<th>Partitioned</th>
<th>P-OMLP</th>
<th>SPFP (asymptotical tightness)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>([— &amp; Anderson, 2010])</td>
<td>([— &amp; Anderson, 2010])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P-FMLP+ (practical protocol)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>([—, 2011])</td>
</tr>
<tr>
<td>Global</td>
<td>G-OMLP</td>
<td>FMLP</td>
</tr>
<tr>
<td></td>
<td>([— &amp; Anderson, 2010])</td>
<td>([Block et al., 2007])</td>
</tr>
<tr>
<td>Clustered</td>
<td>OMIP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>([—, 2013])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-OMLP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>([— &amp; Anderson, 2011])</td>
<td></td>
</tr>
</tbody>
</table>

(2) The partitioned & clustered choice: (variants of) priority boosting.
What about Priority Boosting?

(1) The classic choice: (variants of) priority inheritance.

<table>
<thead>
<tr>
<th></th>
<th>job-level fixed priority</th>
<th>Any SJFT Scheduler</th>
<th>Deadlines</th>
<th>Any SJFT Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioned</td>
<td>(no migrations)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustered</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **P-OMLP** [— & Anderson, 2010]
- **SPFP** (asymptotical tightness) [— & Anderson, 2010]
- **P-FMLP**+ (practical protocol) [—, 2011]
- **G-OMLP** [— & Anderson, 2010]
- **FMLP** [Block et al., 2007]
- **OMIP** [—, 2013]
- **C-OMLP** [— & Anderson, 2011]

(2) The partitioned & clustered choice: (variants of) priority boosting.
Sub-Optimality of Unrestricted Priority Boosting

It is impossible to construct an asymptotically optimal locking protocol (w.r.t. s-aware analysis) under global JLFP scheduling based on unrestricted priority boosting. (And hence also under clustered JLFP scheduling.)

Sub-Optimality of **Unrestricted** Priority Boosting

**Unrestricted Priority Boosting Example Schedule**

[same task set & arrival sequence as before]

4 tasks on 2 processors

*global fixed-priority scheduling*

Task priorities (high to low): $T_1 > T_2 > T_3 > T_4$
Lock-Holding Jobs Cannot Be Preempted

Graph showing the timeline with events such as job release, deadline, job completion, job suspended, and priority inversion. The figure includes timeseries for Processor 1 and Processor 2 with critical sections and scheduled jobs.
Lock-Holding Jobs Cannot Be Preempted

Priority Boosting

$T_4$ cannot be preempted by newly released, non-resource-holding job...

- scheduled
- critical section
- job release
- deadline
- job completion
- job suspended
- priority inversion
PI-Blocking Shifted, not Avoided

Priority Boosting

$T_4$ cannot be preempted by newly released, non-resource-holding job...

scheduled critical section

job release deadline job completion

job suspended

priority inversion
PI-Blocking Shifted, not Avoided

Priority Boosting

$T_4$ cannot be preempted by newly released, non-resource-holding job...

... but $T_3$ is still preempted by $T_1$.

$\rightarrow$ s-aware pi-blocking

<table>
<thead>
<tr>
<th>scheduled</th>
<th>critical section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor 1</td>
<td></td>
</tr>
<tr>
<td>Processor 2</td>
<td></td>
</tr>
</tbody>
</table>

job release        job suspended

job completion     priority inversion

deadline
"Victim Task" Accumulates PI-Blocking

\[ \Omega(\phi) \text{ pi-blocking is possible, where } \phi = \frac{\text{max response time}}{\text{min period}} \]

Diagram showing time intervals and processor states:
- \(T_4\), \(T_3\), \(T_2\), \(T_1\) represent tasks or processors.
- Scheduled and critical sections indicated.
- Job release, deadline, job completion, and priority inversion marked.

Explanation:
- Processor 1 and Processor 2 have different states and actions marked.
- Time axis from 0 to 20.
Sub-Optimality of Unrestricted Priority Boosting

\[ \Omega(\phi) \text{ pi-blocking is possible, where } \phi = \frac{\text{max response time}}{\text{min period}} \]

Also repeats \( \phi = \frac{\text{max response time}}{\text{min period}} \) times…

→ not asymptotically optimal.
Remark: examples use **single resource** shared by **only two tasks**

→ **queue order irrelevant** (FIFO-ordered, priority-ordered, etc.)

→ **cannot simplify problem** with coarser-grained locking
The FMLP+: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis

We need something new...

<table>
<thead>
<tr>
<th></th>
<th>Any JLFP Sched.</th>
<th>Deadlines</th>
<th>Any JLFP Sched.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>P-FMLP+ (practical protocol) [—, 2011]</td>
</tr>
<tr>
<td>Global</td>
<td>G-OMLP [— &amp; Anderson, 2010]</td>
<td></td>
<td>FMLP [Block et al., 2007]</td>
</tr>
<tr>
<td>Clustered</td>
<td>C-OMLP [— &amp; Anderson, 2011]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) The classic choice: (variants of) priority inheritance.

(2) The partitioned & clustered choice: (variants of) priority boosting.
We need something new…

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(jobs migrate freely)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Partitioned</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(no migrations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Clustered</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(jobs migrate only among subset of processors)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **The classic choice**: (variants of) priority inheritance.
   - **P-OMLP** [Anderson, 2010]
   - **SPFP** (asymptotical tightness) [Anderson, 2010]
   - **P-FMLP** (practical protocol) [Anderson, 2011]

2. **The partitioned & clustered choice**: (variants of) priority boosting.
   - **G-OMLP** [Anderson, 2010]
   - **FMLP** [Block et al., 2007]
   - **OMIP** [Anderson, 2013]
   - **C-OMLP** [Anderson, 2011]
Observation: $O(n)$ PI-Blocking Possible

![Diagram with time axis and processor schedule](image)

- Processor 1: scheduled, critical section
- Processor 2: scheduled, critical section

- Time points: $T_1, T_2, T_3, T_4$
- Events: job release, deadline, job completion
- Symbols: job suspended, priority inversion
Observation: $O(n)$ PI-Blocking Possible

This schedule: *unrestricted* priority boosting.

This schedule: unrestricted priority boosting.

scheduled  critical section  job release  
job suspended  deadline  job completion  priority inversion
Observation: $O(n)$ PI-Blocking Possible

This schedule: unrestricted priority boosting.

PI-blocking cannot be avoided, but it can be shifted to new jobs.

➔ prevent accumulation of pi-blocking in individual jobs.
Idea: Protect Existing Independent Jobs

Alternative possible schedule.

- $T_1$
- $T_2$
- $T_3$
- $T_4$

 Processor 1
- scheduled
- critical section

 Processor 2
- scheduled
- critical section

- job release
- deadline
- job completion

- job suspended
- priority inversion

The FMLP$^+$: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis
Idea: **Protect Existing Independent Jobs**

*Alternative possible schedule.*

Both $T_1$ and $T_2$ incur pi-blocking instead.

$T_3$ is not preempted while $T_4$ holds a lock!
O(n) PI-Blocking Per Job

Alternative possible schedule.

- scheduled
- critical section
- job release
- deadline
- job completion
- job suspended
- priority inversion
Each job incurs only a limited amount of pi-blocking.

→ asymptotically optimal.
This is actually a **Generalized FMLP**+ schedule.

*Key question: how to specify that “T₃ must be protected”?*
Closing the S-Aware Asymptotic Optimality Gap
Key Problem: Preemptions due to Later-Started Critical Sections

On Uniprocessors

- a job is **blocked only** by critical sections that are already in **progress** when the job is released / resumed.
Key Problem: Preemptions due to Later-Started Critical Sections

On Uniprocessors

- A job is **blocked only** by critical sections that are already in progress when the job is released / resumed.

On Multiprocessors

- Priority boosting example…

![Diagram showing priority inversion](image-url)
Priority Boosting / Priority Inheritance Examples

$T_3$ blocked due to $\phi$ requests issued after $T_3$ started executing.

($\Rightarrow$ root cause: parallel scheduling of lower-priority jobs)

On Uniprocessors

- A job is blocked only by critical sections that are already in progress when the job is released / resumed.

On Multiprocessors

- Priority boosting example…

Time

Job $T_1$, $T_2$, $T_3$, $T_4$
What if this is disallowed...?

strawman rule: jobs cannot be preempted due to later-issued requests

cannot preempt

may preempt (one per task)

may not preempt

critical sections

analyzed job

CS<sub>b</sub>  CS<sub>x</sub>  CS<sub>a</sub>

J<sub>i</sub>

released

greater than O(n) preemptions
This is the desired effect, but the simple rule fails in corner cases.

Strawman rule: jobs cannot be preempted due to later-issued requests.

- Critical sections: 
  - $CS_b$ cannot preempt.
  - $CS_x$ may preempt (one per task).
  - $CS_a$ may not preempt.

- Analyzed job: $J_i$ released.

- Time: $O(n)$ preemptions.
Independent & Request Segments

a job at runtime:

```
 independent segment | request segment | independent segment
```

- **holding a lock or suspended**
- **ready & does not require a lock**

**Note:** exact segments known only at runtime
- potentially complex, non-linear **control flow** determines which resources are required and in which order
- approach **not** limited to linear, branch-free tasks
**Key Concept:** *Segment Start Time*

Simply the start time of a job’s current segment.

---

**Note:** exact segments known only at runtime

- potentially complex, non-linear **control flow** determines which resources are required and in which order
- approach **not** limited to linear, branch-free tasks
A Lock Holder’s Co-Boosting Set

Key idea underlying the Generalized FMLP⁺

*If a job is priority-boosted, then certain other jobs must also be co-boosted.*
A Lock Holder’s Co-Boosting Set

If a job $J_b$ holds a lock at time $t$, then its co-boosting set is defined as:

$$\{ J_y \mid J_y \text{ executes an independent segment at time } t \text{ and }$$
$$J_y \text{ has higher priority than } J_b \text{ and }$$
$$J_y's \text{ current segment started before } J_b's \text{ segment.} \}$$

(Note: in this talk, I’ll use “task” and “job” interchangeably.)

Intuition

- The set of jobs at risk of accumulating pi-blocking due to $J_b$. 
Example: \{ T_3 \} \text{ is } T_4 \text{'s Co-Boosting Set}

Generalized FMLP$^+$ schedule.

Processor 1
- scheduled
- critical section

Processor 2
- scheduled
- critical section

↓ job release
↓ deadline
↓ job completion

≡ job suspended

PI priority inversion
Example: \( \{ T_3 \} \) is \( T_4 \)'s Co-Boosting Set

\[
\text{Generalized FMLP}^+ \text{ schedule.}
\]

\[ \begin{align*}
T_1 & \quad T_2 & \quad T_3 & \quad T_4
\end{align*} \]

\( T_3 \) executes an **independent segment** at time \( t \) and
\( T_3 \) has **higher priority** than \( T_4 \) and
\( T_3 \)'s current segment **started before** \( T_4 \)'s segment.
\( T_1 \) and \( T_2 \) execute \textbf{independent segments} at time \( t \) and \( T_1 \) and \( T_2 \) have \textbf{higher priority} than \( T_4 \) but \( T_1 \) and \( T_2 \)'s current segments \textbf{did NOT start before} \( T_4 \)'s segment.
Restricted Segment Boosting

In a cluster with $c$ CPUs, at any point in time $t$, schedule the following jobs:
Restricted Segment Boosting

In a cluster with $c$ CPUs, at any point in time $t$, schedule the following jobs:

- **A Single Boosted Job $J_b$**
  - The lock-holding ready job (if any) with the **earliest segment start time**.

(any ties broken arbitrarily but consistently)
Restricted Segment Boosting

In a cluster with \( c \) CPUs, at any point in time \( t \), schedule the following jobs:

**A Single Boosted Job \( J_b \)**

The lock-holding ready job (if any) with the earliest segment start time.

**Up to \( c - 1 \) jobs from \( J_b \)'s Co-Boosting Set**

Select the (up to) \( c - 1 \) jobs with the earliest segment start times.

(any ties broken arbitrarily but consistently)
Restricted Segment Boosting

In a cluster with \( c \) CPUs, at any point in time \( t \), schedule the following jobs:

- **A Single Boosted Job** \( J_b \)
  - The lock-holding ready job (if any) with the earliest segment start time.

- **Up to \( c - 1 \) jobs from \( J_b \)’s Co-Boosting Set**
  - Select the (up to) \( c - 1 \) jobs with the earliest segment start times.

- **If less than \( c \) jobs scheduled so far: any other ready jobs**
  - Select the highest-priority ready jobs not yet scheduled (may hold locks).

(any ties broken arbitrarily but consistently)
Restricted Segment Boosting at Time 1

Generalized FMLP$^+$ schedule.

Time

0 5 10 15 20

T4
T3
T2
T1

scheduled

Processor 1
Processor 2

critical section

job release

job completion

deadline

job suspended

priority inversion
(1) The **lock-holding ready job** (if any) with the **earliest segment start time**.
(1) The lock-holding ready job (if any) with the earliest segment start time.

(2) Up to \( c - 1 = 1 \) jobs from \( T_4 \)’s co-boosting set = \( \{ T_3 \} \).
(1) The lock-holding ready job (if any) with the earliest segment start time.

(2) Up to \( c - 1 = 1 \) jobs from \( T_4 \)'s co-boosting set = \( \{ T_3 \} \).

(3) If less than \( c = 2 \) jobs scheduled so far: any other ready jobs.

At time 1: no more CPUs available after steps 1 & 2.
The Generalized FIFO Multiprocessor Locking Protocol (FMLP⁺)

Restricted Segment Boosting + Per-Resource FIFO Queues
The Generalized FIFO Multiprocessor Locking Protocol (FMLP⁺)

Restricted Segment Boosting + Per-Resource FIFO Queues

S-Aware PI-Blocking per Segment

- ...during request segment: $O(n)$.
  - Proof: rather straightforward ➞ see paper.
- ...during independent segment: $O(n)$.
  - Proof: rather involved ➞ see paper.
The Generalized FIFO Multiprocessor Locking Protocol (FMLP+)

Restricted Segment Boosting + Per-Resource FIFO Queues

S-Aware PI-Blocking per Segment

- …during request segment: $O(n)$.
  ‣ Proof: rather straightforward ➞ see paper.
- …during independent segment: $O(n)$.
  ‣ Proof: rather involved ➞ see paper.

Number of segments

- Constant #requests and #self-suspensions per job
  → constant number of segments.
The Generalized FIFO Multiprocessor Locking Protocol (FMLP⁺)

**Restricted Segment Boosting** + **Per-Resource FIFO Queues**

**S-Aware PI-Blocking per Segment**

- ...during request segment: $O(n)$.
  - Proof: rather straightforward ➞ see paper.
- ...during independent segment: $O(n)$.
  - Proof: rather involved ➞ see paper.

**Number of segments**

- Constant #requests and #self-suspensions per job
  ➞ constant number of segments.

**Overall Max. S-Aware PI-Blocking**

- $O(n)$ under clustered JLFP scheduling.
# Multiprocessor Real-Time Locking Optimality Results

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job-level fixed-priority</td>
<td>Any JLFP Scheduler</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

### Partitioned
- **Partitioned** (no migrations)
  - **P-OMLP**
    - [Anderson & Anderson, 2010]

### Global
- **Global** (jobs migrate freely)
  - **G-OMLP**
    - [Anderson & Anderson, 2010]

### Clustered
- **Clustered** (jobs migrate only among subset of processors)
  - **OMIP**
    - [Anderson & Anderson, 2011]
  - **C-OMLP**
    - [Anderson & Anderson, 2011]

---

**The Generalized FMLP+**

(restricted segment boosting)


---

- [Block et al., 2007]
- [Anderson & Anderson, 2010]
- [Anderson & Anderson, 2011]
- [Anderson & Anderson, 2011]
- [Ward & Anderson, 2012]
- Optimality Results for Multiprocessor Real-Time Locking, RTSS 2010.
The Generalized FMLP$^+$ based on Restricted Segment Boosting closes the s-aware asymptotic optimality gap. See paper & online appendix for large-scale empirical evaluation. (Summary: the FMLP$^+$ works well if the schedulability analysis is accurate enough.)

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>job-level fixed-priority</td>
<td>Any JLFP Scheduler</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partitioned</th>
<th>SUSPENSION_OBLIVIOUS</th>
<th>SUSPENSION_AWARE</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Global</th>
<th>SUSPENSION_OBLIVIOUS</th>
<th>SUSPENSION_AWARE</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Clustered</th>
<th>SUSPENSION_OBLIVIOUS</th>
<th>SUSPENSION_AWARE</th>
</tr>
</thead>
</table>

The Generalized FMLP$^+$ (restricted segment boosting)

Conclusion
Summary

The Generalized FMLP+

- priority boosting & inheritance unsuitable
- based instead on restricted segment boosting
  ‣ Key idea: co-boosting of independent jobs

In the Paper

- Empirical evaluation.
- How to integrate with locking-unrelated self-suspensions...
  ‣ ... also within critical sections.
- How to integrate with Ward & Anderson’s RNLP [2012] for asymptotically optimal pi-blocking given nested critical sections.

Online Appendix

- Fine-grained blocking analysis based on linear programming framework [—, 2013].
- Complete evaluation results (5760 graphs).

The FMLP+: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis

Byron B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)

A. Contributions

We answer this fundamental question by introducing the generalized FIFO Multiprocessor Locking Protocol (FMLP+), the first semaphore protocol for clustered scheduling that ensures (n-ε) maximum s-aware pi-blocking under any ILP policy.

B. Related Work

On uniprocessors, the blocking optimality problem has long been solved both the classic Spin-Resource Poles [3] and the Priority Ceiling Protocol [22, 24] limit pi-blocking to at most one (outermost) critical section, which is obviously optimal. On multiprocessors, there are two major lock types: spin locks, wherein blocked jobs busy-wait, and suspension-based semaphores. Spin locks are well understood and it is difficult to see that non-preemptible FIFO spin locks, which ensure O(1)

— Introduction

The purpose of suspension-based real-time locking protocols is to limit priority inversions [22], which, intuitively, occur when a high-priority task that should be scheduled is instead delayed by a remote or lower-priority task. Such locking-related delays, also called priority inversions or blocking (pi-blocking), are problematic because it can result in deadline misses. However, some pi-blocking is unavoidable when using locks and thus must be bounded and accounted for during schedulability analysis.

Clearly, an “optimal” locking protocol should minimize pi-blocking to the extent possible. Formally, a locking protocol is asymptotically optimal if it ensures that, for any task set, maximum pi-blocking is bounded within a constant factor of the pi-blocking unavoidable in some task set [11]. Interestingly, there exist two classes of schedulability analysis that yield different lower-bound: under suspension-oblivious (s-oblivious) analysis, s-oblivious pi-blocking is fundamental, whereas under suspension-aware (s-aware) analysis, s-aware pi-blocking is unavoidable in the general case [7, 11], whereas s-aware analysis denote the number of processors and tasks, respectively. As the names imply, the key difference is that suspensions are accounted for explicitly under s-aware analysis, whereas they are implicitly modeled as processor demand in the s-oblivious case. In principle, s-aware schedulability analysis is preferable, but s-oblivious analysis is easier to derive and permits simpler pi-blocking bounds.

And indeed, for the simpler s-oblivious case, asymptotically optimal locking protocols are known for partitioned, global, and clustered JLFP scheduling. However, for the case of non-lock-holding, low-priority jobs are priority-boosted, then certain other non-lock-holding, higher-priority jobs must be co-boosted.

The general problem of optimal s-aware blocking under global and clustered ILP scheduling, however, has remained unsolved.

I. Introduction

The purpose of suspension-based real-time locking protocols is to limit priority inversions [22], which, intuitively, occur when a high-priority task that should be scheduled is instead delayed by a remote or lower-priority task. Such locking-related delay, also called priority inversions or blocking (pi-blocking), is problematic because it can result in deadline misses. However, some pi-blocking is unavoidable when using locks and thus must be bounded and accounted for during schedulability analysis.

Clearly, an “optimal” locking protocol should minimize pi-blocking to the extent possible. Formally, a locking protocol is asymptotically optimal if it ensures that, for any task set, maximum pi-blocking is bounded within a constant factor of the pi-blocking unavoidable in some task set [11]. Interestingly, there exist two classes of schedulability analysis that yield different lower-bound: under suspension-oblivious (s-oblivious) analysis, s-oblivious pi-blocking is fundamental, whereas under suspension-aware (s-aware) analysis, s-aware pi-blocking is unavoidable in the general case [7, 11], whereas s-aware analysis denote the number of processors and tasks, respectively. As the names imply, the key difference is that suspensions are accounted for explicitly under s-aware analysis, whereas they are implicitly modeled as processor demand in the s-oblivious case. In principle, s-aware schedulability analysis is preferable, but s-oblivious analysis is easier to derive and permits simpler pi-blocking bounds.

And indeed, for the simpler s-oblivious case, asymptotically optimal locking protocols are known for partitioned, global, and clustered JLFP scheduling. However, for the case of non-lock-holding, low-priority jobs are priority-boosted, then certain other non-lock-holding, higher-priority jobs must be co-boosted.

The general problem of optimal s-aware blocking under global and clustered ILP scheduling, however, has remained unsolved.

A. Contributions

We answer this fundamental question by introducing the generalized FIFO Multiprocessor Locking Protocol (FMLP+), the first semaphore protocol for clustered scheduling that ensures (n-ε) maximum s-aware pi-blocking under any ILP policy.

While it was initially assumed [11] that a variant of Block et al.’s Flexible Multiprocessor Locking Protocol (FMLP) [16]—which uses FIFO queues together with priority inheritance [22]—is asymptotically optimal under global scheduling, we show in Sec. III that this holds only under some, but not all global ILP schedulers. In fact, we show that both priority inheritance and concurrency (or priority boosting [23], which are the two mechanisms used in all prior locking protocols for s-aware analysis to avoid unbounded pi-blocking, can give rise to non-optimal (10ε) pi-blocking, where ε is the ratio of the longest and the shortest period and not bounded by 1 or ε).

To overcome this lower bound, we introduce in Sec. IV-A a new locking mechanism called “s-outmost segment boosting,” which boosts at most one carefully chosen lock-holding job in each cluster while simultaneously “co-boosting” certain other, non-lock-holding jobs to interfere with the underlying JLFP scheduling and thus prevent non-optimal pi-blocking. By using the s-aware case, asymptotically optimal locking protocols are known for partitioned, global, and clustered JLFP scheduling. However, for the case of nested critical sections, however, the gap is closed with the introduction of the first semaphore protocol for partitioned JLFP scheduling. However, for the case of a high-priority task that should be scheduled is instead delayed by a remote or lower-priority task. Such locking-related delay, also called priority inversions or blocking (pi-blocking), is problematic because it can result in deadline misses. However, some pi-blocking is unavoidable when using locks and thus must be bounded and accounted for during schedulability analysis.

Clearly, an “optimal” locking protocol should minimize pi-blocking to the extent possible. Formally, a locking protocol is asymptotically optimal if it ensures that, for any task set, maximum pi-blocking is bounded within a constant factor of the pi-blocking unavoidable in some task set [11]. Interestingly, there exist two classes of schedulability analysis that yield different lower-bound: under suspension-oblivious (s-oblivious) analysis, s-oblivious pi-blocking is fundamental, whereas under suspension-aware (s-aware) analysis, s-aware pi-blocking is unavoidable in the general case [7, 11], whereas s-aware analysis denote the number of processors and tasks, respectively. As the names imply, the key difference is that suspensions are accounted for explicitly under s-aware analysis, whereas they are implicitly modeled as processor demand in the s-oblivious case. In principle, s-aware schedulability analysis is preferable, but s-oblivious analysis is easier to derive and permits simpler pi-blocking bounds.

And indeed, for the simpler s-oblivious case, asymptotically optimal locking protocols are known for partitioned, global, and clustered JLFP scheduling. However, for the case of non-lock-holding, low-priority jobs are priority-boosted, then certain other non-lock-holding, higher-priority jobs must be co-boosted.

The general problem of optimal s-aware blocking under global and clustered ILP scheduling, however, has remained unsolved.

A. Contributions

We answer this fundamental question by introducing the generalized FIFO Multiprocessor Locking Protocol (FMLP+), the first semaphore protocol for clustered scheduling that ensures (n-ε) maximum s-aware pi-blocking under any ILP policy.

While it was initially assumed [11] that a variant of Block et al.’s Flexible Multiprocessor Locking Protocol (FMLP) [16]—which uses FIFO queues together with priority inheritance [22]—is asymptotically optimal under global scheduling, we show in Sec. III that this holds only under some, but not all global ILP schedulers. In fact, we show that both priority inheritance and concurrency (or priority boosting [23], which are the two mechanisms used in all prior locking protocols for s-aware analysis to avoid unbounded pi-blocking, can give rise to non-optimal (10ε) pi-blocking, where ε is the ratio of the longest and the shortest period and not bounded by 1 or ε).
Future Work & Open Questions

Apply this technique to reader-writer locks?
- Lower bounds on s-aware pi-blocking?

Apply this technique to k-exclusion locks?
- GPUs & other co-processors
- Lower bounds on s-aware pi-blocking?

Overheads of restricted segment boosting?
- Tracking **segment start times** is simple and cheap.
- But... additional preemptions?
The FMLP+: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis

### Multiprocessor Real-Time Locking Optimality Results

<table>
<thead>
<tr>
<th>JLFP</th>
<th>Suspension Oblivious</th>
<th>Suspension Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>job-level fixed-priority</td>
<td>Any JLFP Scheduler</td>
<td>Any JLFP Scheduler</td>
</tr>
</tbody>
</table>

| | Partitioned | Global | Clustered |
| | (no migrations) | (jobs migrate freely) | (jobs migrate only among subset of processors) |
| | P-OMLP | G-OMLP | OMIP |

The Generalized FMLP+ (restricted segment boosting)


---

Optimality Results for Multiprocessor Real-Time Locking, RTSS 2010.
Appendix
# Multiprocessor Real-Time Locking

## Optimality Classes

<table>
<thead>
<tr>
<th>Blocking Optimality</th>
<th>suspension oblivious</th>
<th>suspension aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>How are suspensions analyzed?</td>
<td>CPU demand is over-approximated</td>
<td>CPU demand is modeled accurately</td>
</tr>
<tr>
<td>Lower bound on maximum priority inversion blocking $\max_i {b_i}$</td>
<td>$\Omega(m)$ with $m = #\text{CPUs}$</td>
<td>$\Omega(n)$ with $n = #\text{tasks}$</td>
</tr>
</tbody>
</table>

S-Aware vs. S-Oblivious Analysis
S-Aware vs. S-Oblivious Analysis

Suspension-oblivious (s-oblivious) Analysis

Motivation: reuse existing schedulability analyses that assume independent, always ready tasks.
S-Aware vs. S-Oblivious Analysis

Suspension-oblivious (\textit{s-oblivious}) Analysis

\begin{itemize}
  \item Task set\hspace{1cm} s-oblivious blocking analysis\hspace{1cm} modified task set with inflated WCETs\hspace{1cm} s-oblivious schedulability test
  \item Scheduled\hspace{1cm} not scheduled
\end{itemize}

Suspension-aware (\textit{s-aware}) Analysis

\begin{itemize}
  \item Task set\hspace{1cm} s-aware blocking analysis\hspace{1cm} s-aware schedulability test
  \item Scheduled\hspace{1cm} not scheduled\hspace{1cm} maximum self-suspension lengths
\end{itemize}
Suspension-oblivious (s-oblivious) Analysis

- task set → s-oblivious blocking analysis → modified task set with inflated WCETs → s-oblivious schedulability test
  - schedulable
  - not schedulable

Suspension-aware (s-aware) Analysis

- task set → s-aware blocking analysis → s-aware schedulability test
  - schedulable
  - not schedulable

Requires availability of a schedulability test that accounts (reasonably accurately) for **self-suspensions**. 
(…which can be tricky to derive)
Different notions of “processor demand” → different definitions of “priority inversion”.

Suspension-oblivious (s-oblivious) Analysis

- **s-oblivious blocking analysis**
- **modified task set with inflated WCETs**
- **s-oblivious schedulability test**
  - schedulable
  - not schedulable

Suspension-aware (s-aware) Analysis

- **s-aware blocking analysis**
- **maximum self-suspension lengths**
- **s-aware schedulability test**
  - schedulable
  - not schedulable
Restricted Segment Boosting at Time 16

**Generalized FMLP^+ schedule.**

![Graph showing the Generalized FMLP^+ schedule with time intervals and job release, deadline, and job completion events marked.](image)

- **scheduled**
- **critical section**
- **job release**
- **job suspended**
- **deadline**
- **job completion**
- **priority inversion**

Processor 1:
- scheduled
- critical section
- job release
- job suspended
- deadline
- job completion

Processor 2:
- scheduled
- critical section
- job release
- job suspended
- deadline
- job completion

**Legend:**
- **PI** — priority inversion
(1) The lock-holding ready job (if any) with the earliest segment start time.
(1) The lock-holding ready job (if any) with the earliest segment start time.

(2) Up to \( c - 1 = 1 \) jobs from \( T_4 \)'s co-boosting set = \( \emptyset \).
(1) The lock-holding ready job (if any) with the earliest segment start time.

(2) Up to $c - 1 = 1$ jobs from $T_4$'s co-boosting set $= \emptyset$.

At time 16: one CPU available after steps 1 & 2 → schedule highest-priority task $T_1$.

(3) If less than $c = 2$ jobs scheduled so far: any other ready jobs.
Definition: Job Segments

A job at runtime:

- **Independent segment**
  - Starts when a job is *released* or *resumed*, or when it *unlocks a resource*
  - Ends when job *completes*, *suspends*, or *requests a lock*

- **Request segment**
  - Starts when a job *requests a lock*
  - Ends when it *unlocks the resource*