A Partial Overview of Real-Time Synchronization

Real-Time Lunch
Oct 1, 2008

Björn Brandenburg
(with many stolen slides)

The University of North Carolina *at* Chapel Hill

Work supported by IBM and Intel Corps., NSF grants CNS 048996, CCF 0541056, and CNS 0615197, and ARO grant W911NF-06-1-0425.

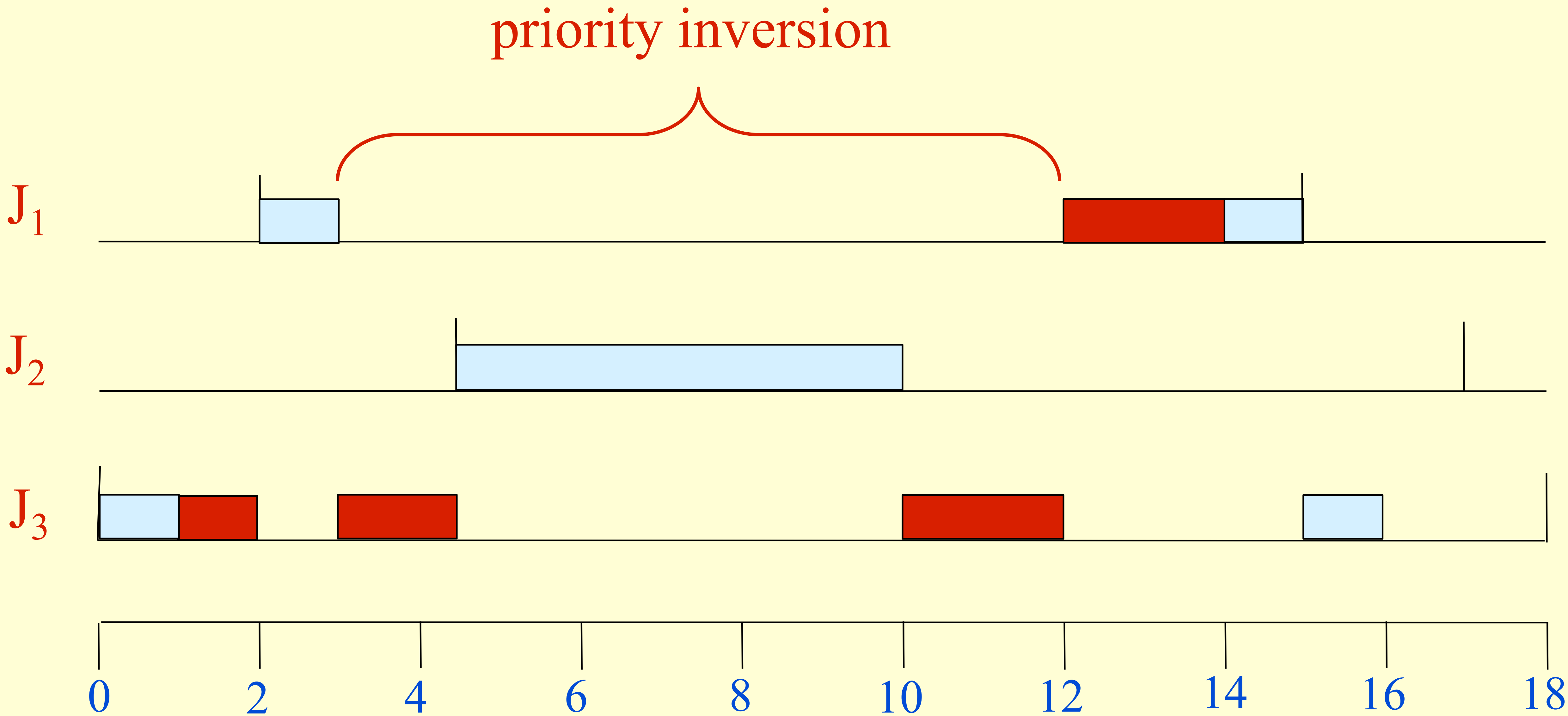
Real-Time Synchronization

(on Uniprocessors)

Priority Inversions

When tasks share resources, there may be priority inversions.

Example:



Quick Review: PCP & SRP

Semaphore protocols based on two concepts

L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

T. Baker, "A stack-based resource allocation policy for realtime processes", *Real-Time Systems*, (3)1:67–99, 1991.

Quick Review: PCP & SRP

Semaphore protocols based on two concepts

priority ceiling
(of a resource *L*)
=
max priority of any job that requests *L*

L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

T. Baker, "A stack-based resource allocation policy for realtime processes", *Real-Time Systems*, (3)1:67–99, 1991.

Quick Review: PCP & SRP

Semaphore protocols based on two concepts

priority ceiling
(of a resource *L*)
=
max priority of any job that requests *L*

system ceiling
(on a processor *P*)
=
max priority ceiling of any resource in use on *P*

L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

T. Baker, "A stack-based resource allocation policy for realtime processes", *Real-Time Systems*, (3)1:67–99, 1991.

Quick Review: PCP & SRP

L. Sha, R. Rajkumar, and J. P. Lehoczky, “Priority inheritance protocols: An approach to real-time synchronization”, *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

T. Baker, "A stack-based resource allocation policy for realtime processes", *Real-Time Systems*, (3)1:67–99, 1991.

Quick Review: PCP & SRP

PCP: Resource request only granted if

1) client priority exceeds system ceiling or

2) client raised system ceiling last.

A resource-holding job is subject to **priority-inheritance**.

L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

T. Baker, "A stack-based resource allocation policy for realtime processes", *Real-Time Systems*, (3)1:67–99, 1991.

Quick Review: PCP & SRP

PCP: Resource request only granted if
1) **client priority exceeds system ceiling** or
2) **client raised system ceiling last**.

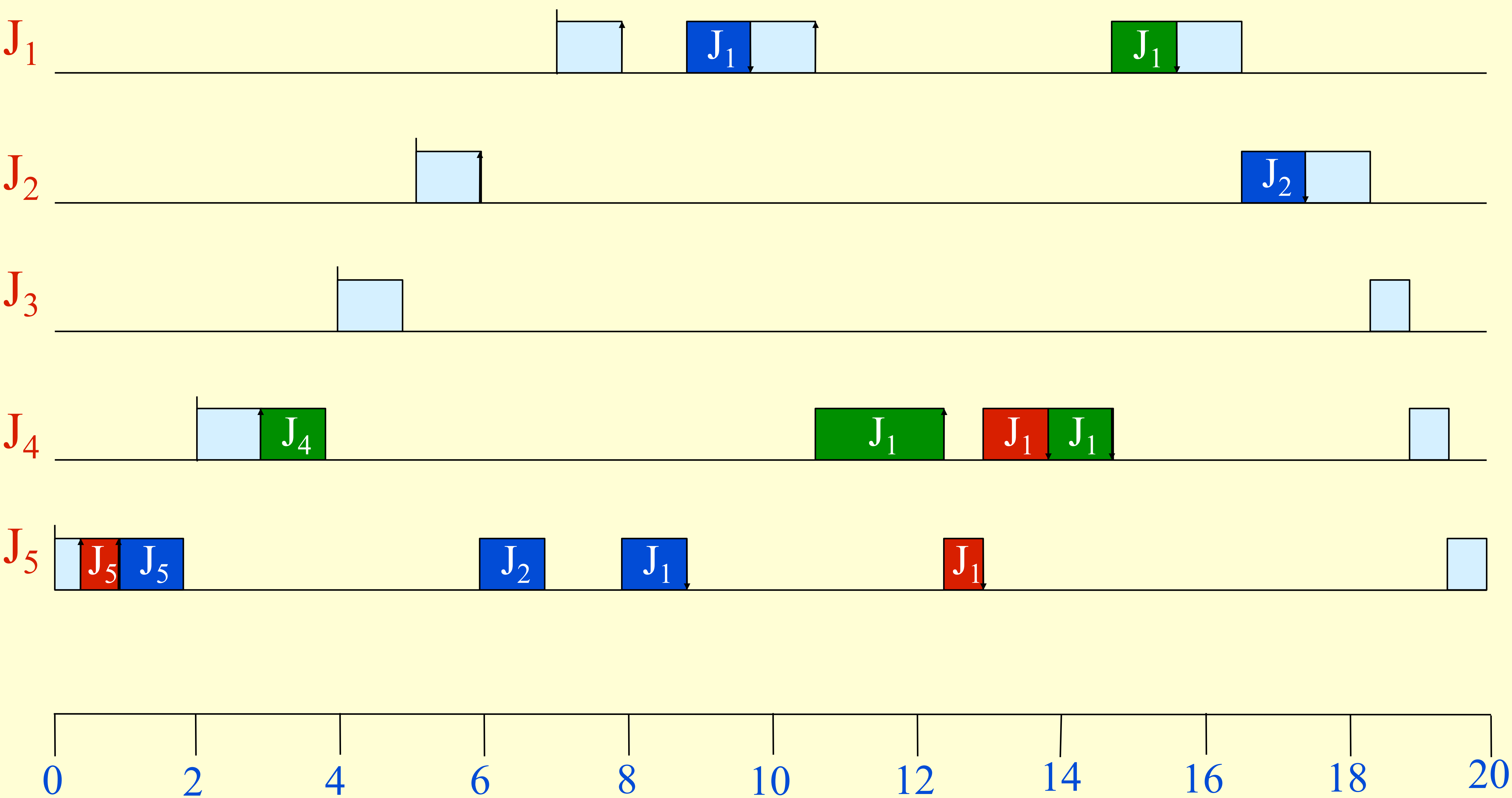
A resource-holding job is subject to **priority-inheritance**.

SRP: A job may not execute unless
1) **its priority exceeds the system ceiling** or
2) **the job executed previously.**

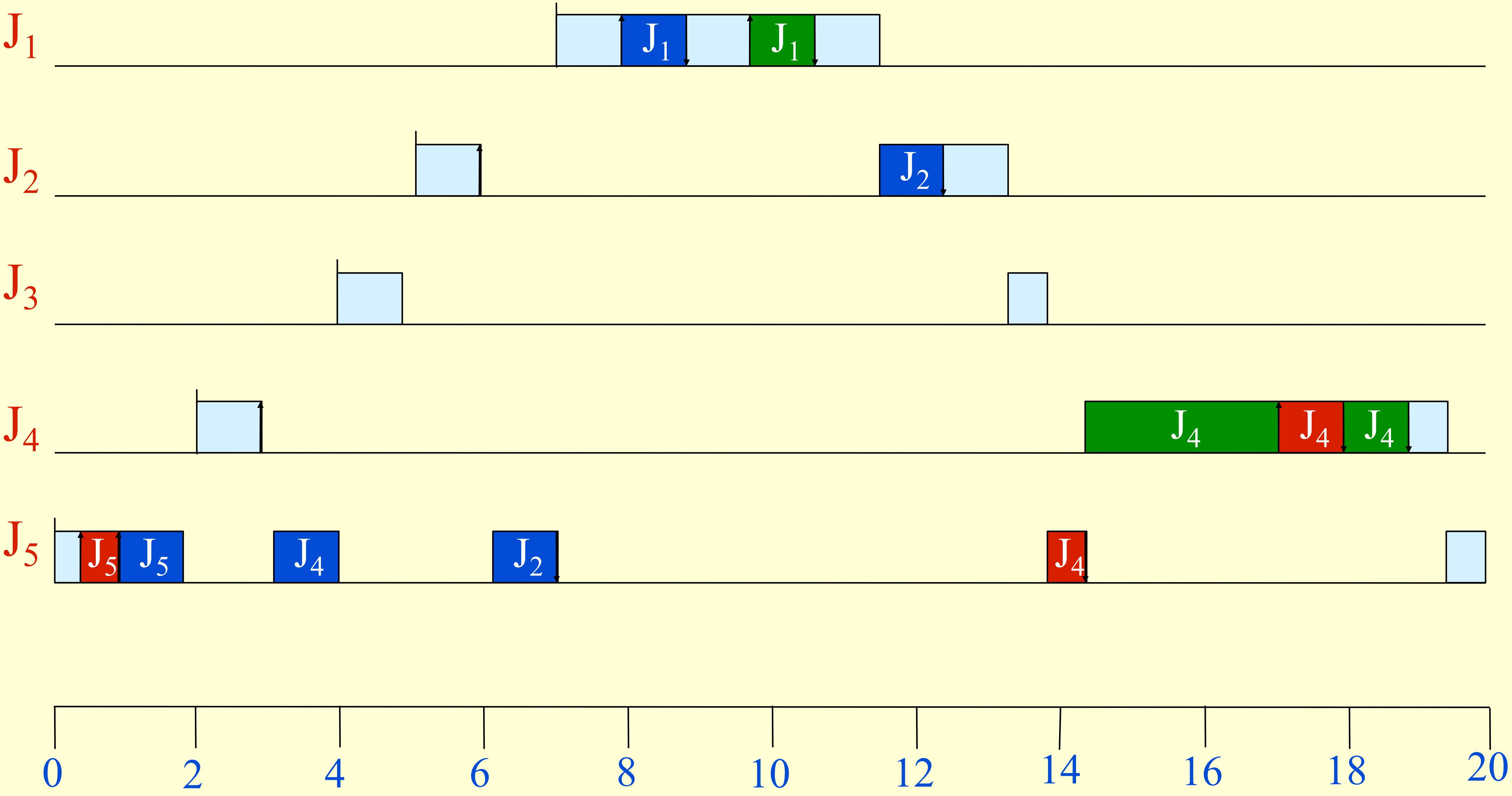
L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

T. Baker, "A stack-based resource allocation policy for realtime processes", *Real-Time Systems*, (3)1:67–99, 1991.

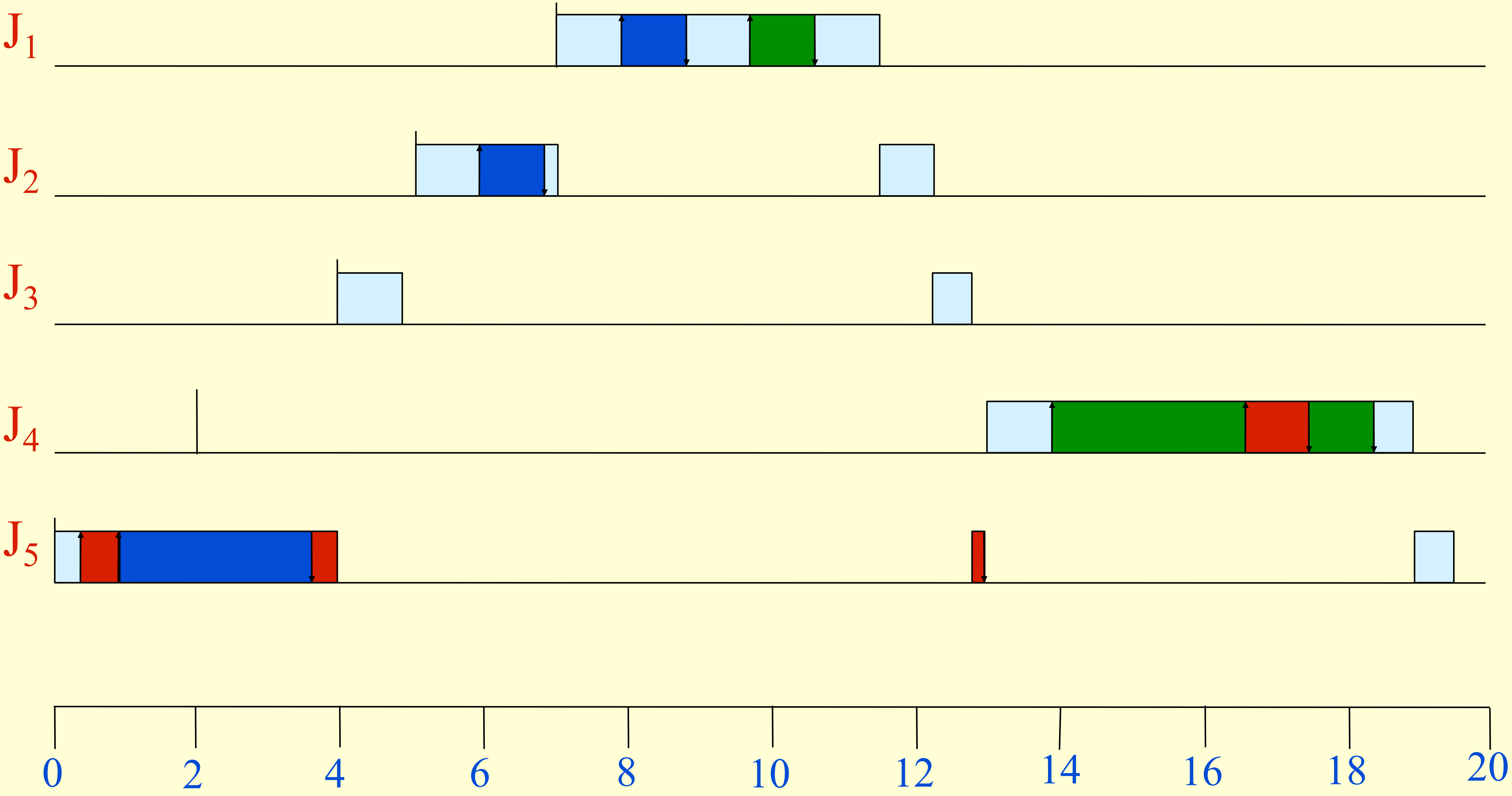
With Priority-Inheritance



With PCP



With SRP



Real-Time Synchronization

(on Multiprocessors)

Real-Time Resource Sharing

On **multiprocessors**,
there are **two kinds** of resources:

Real-Time Resource Sharing

On **multiprocessors**,
there are **two kinds** of resources:

Local
=
all clients on the same processor

Real-Time Resource Sharing

On **multiprocessors**,
there are **two kinds** of resources:

Local
=
all clients on the same processor

Global
=
clients on different processors

Resource Sharing

ltiprocessors,
o kinds of resources:

For local resources, uniprocessor synchronization is sufficient.

(under partitioning)

Local
=
all clients on the same processor

Global
=
clients on different processors

Resou

ltiproc

o kinds

For local resources, uniprocessor synchronization is sufficient.

(under partitioning)

Global resources pose more problems.

In this talk, we focus on global resources.

Local

=

all clients on the same processor

Global

=

clients on different processors

Why are global resources harder to handle?

Why are global resources harder to handle?

Remote blocking:

When processors are no longer independent,
***worst-case analysis becomes pessimistic*.**

Why are global resources harder to handle?

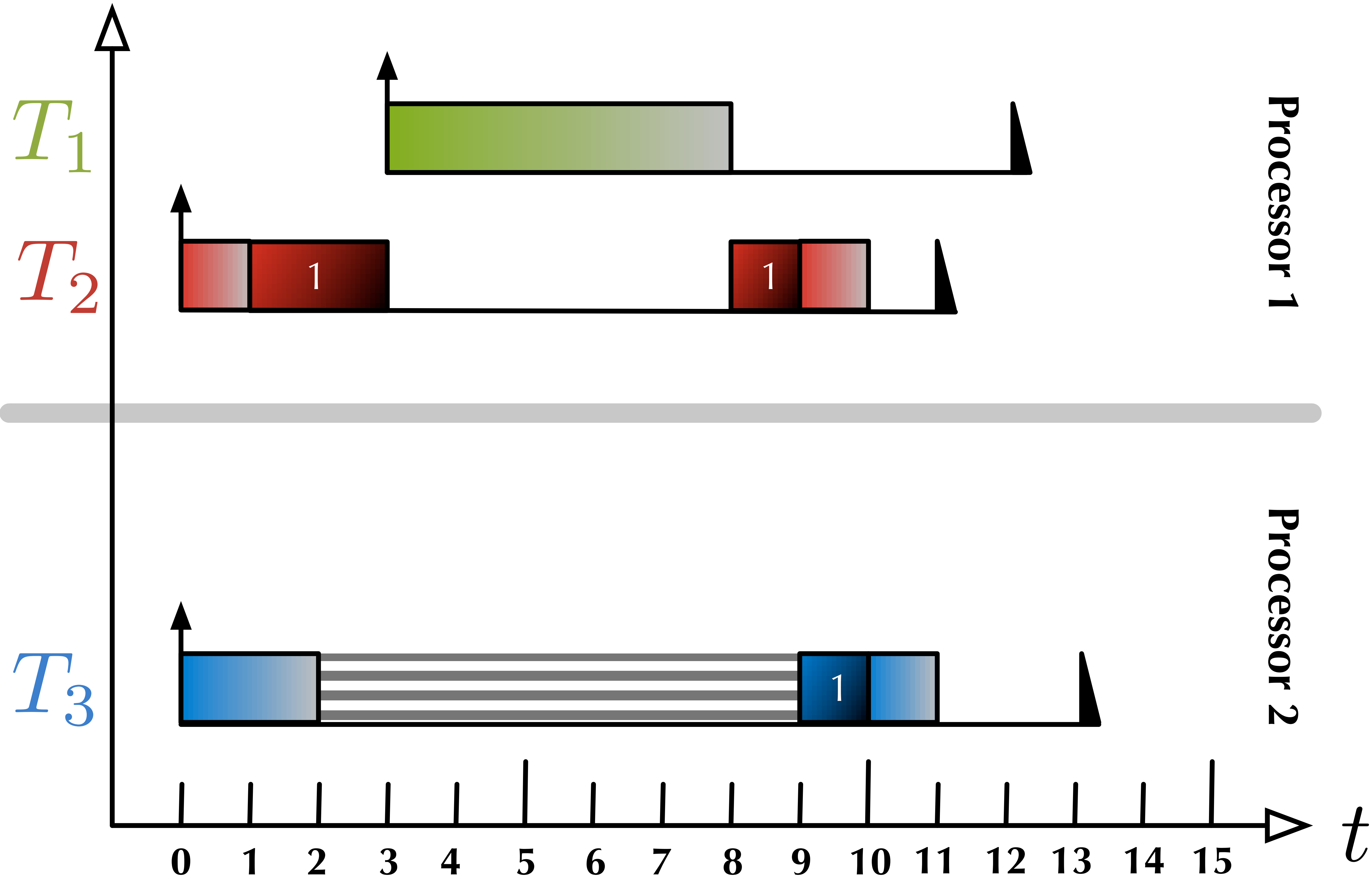
Remote blocking:

When processors are no longer independent, ***worst-case analysis becomes pessimistic****.*

Priority-inheritance is meaningless across processors:

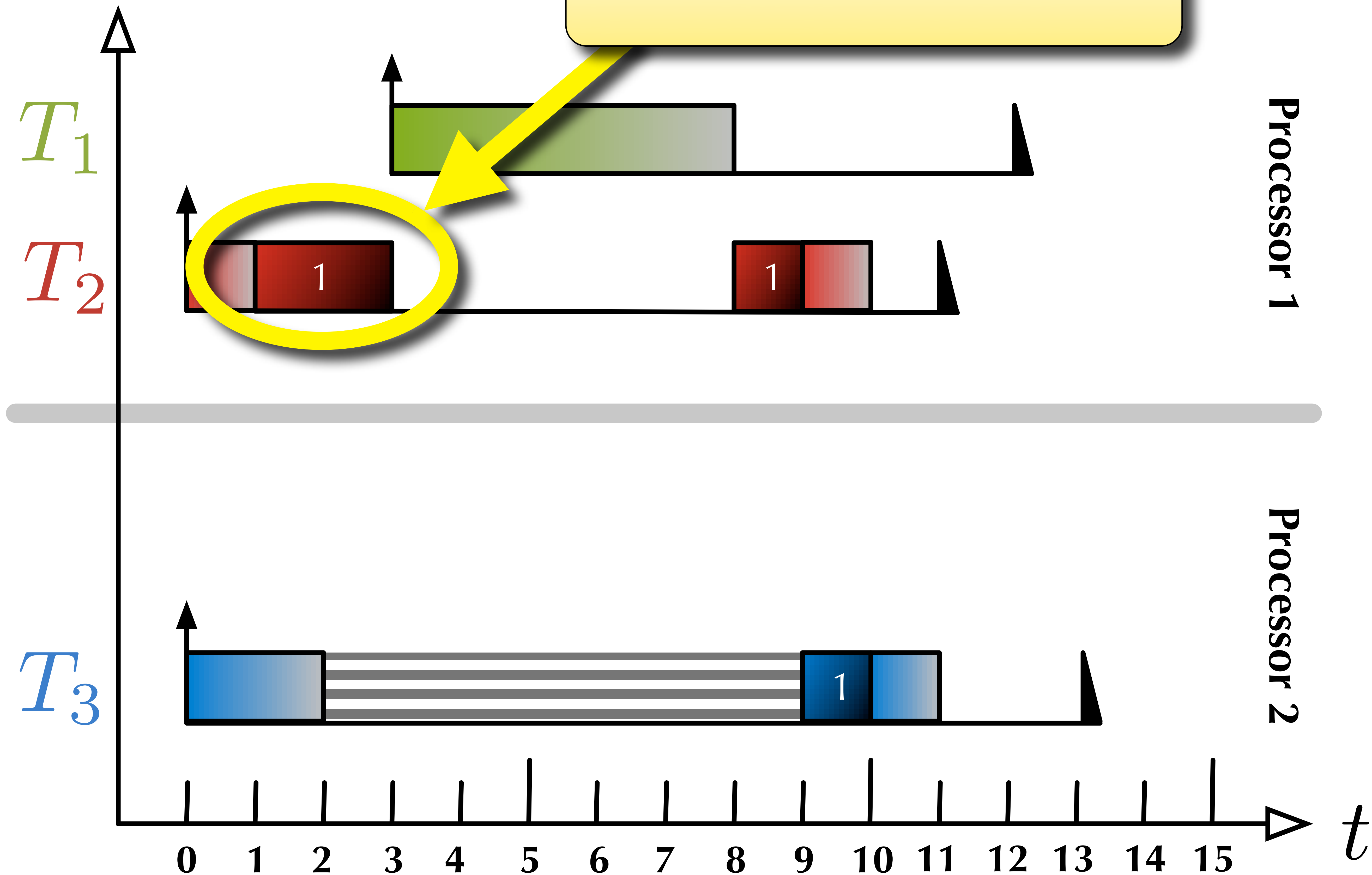
The ***highest priority*** *on processor 1* ***may rank low*** *on processor 2.*

Example: A Naive Approach

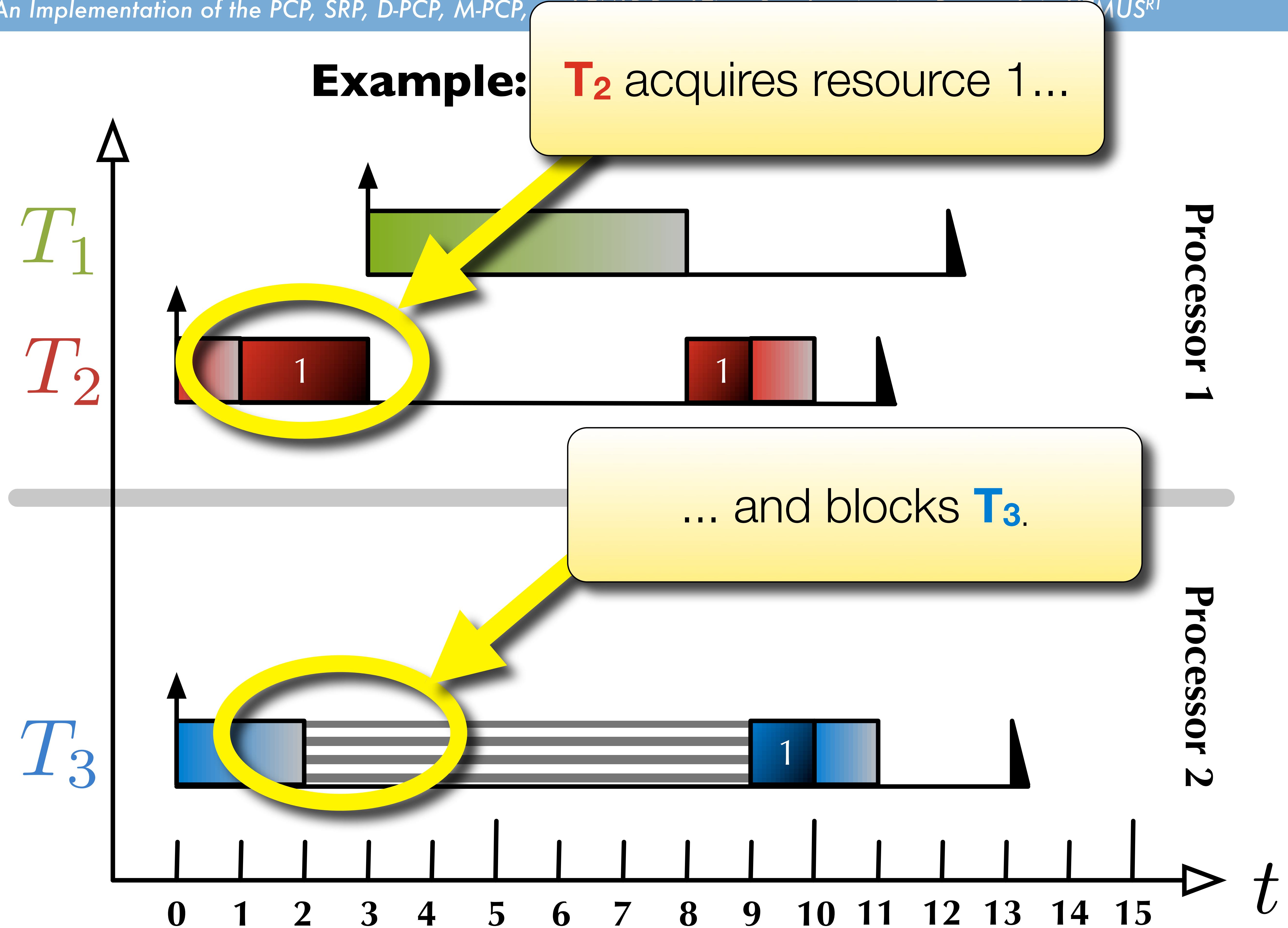


Example:

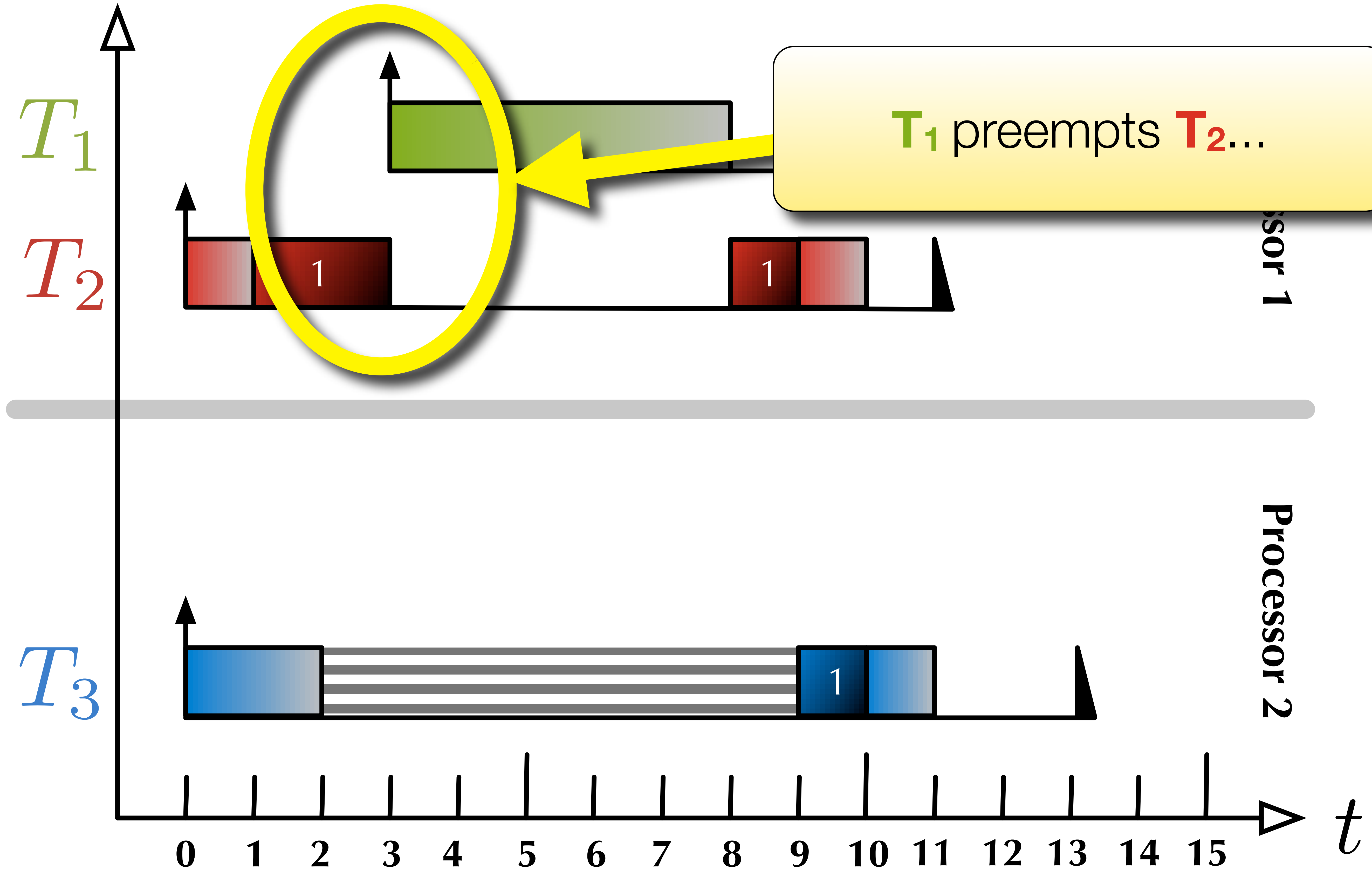
T_2 acquires resource 1...



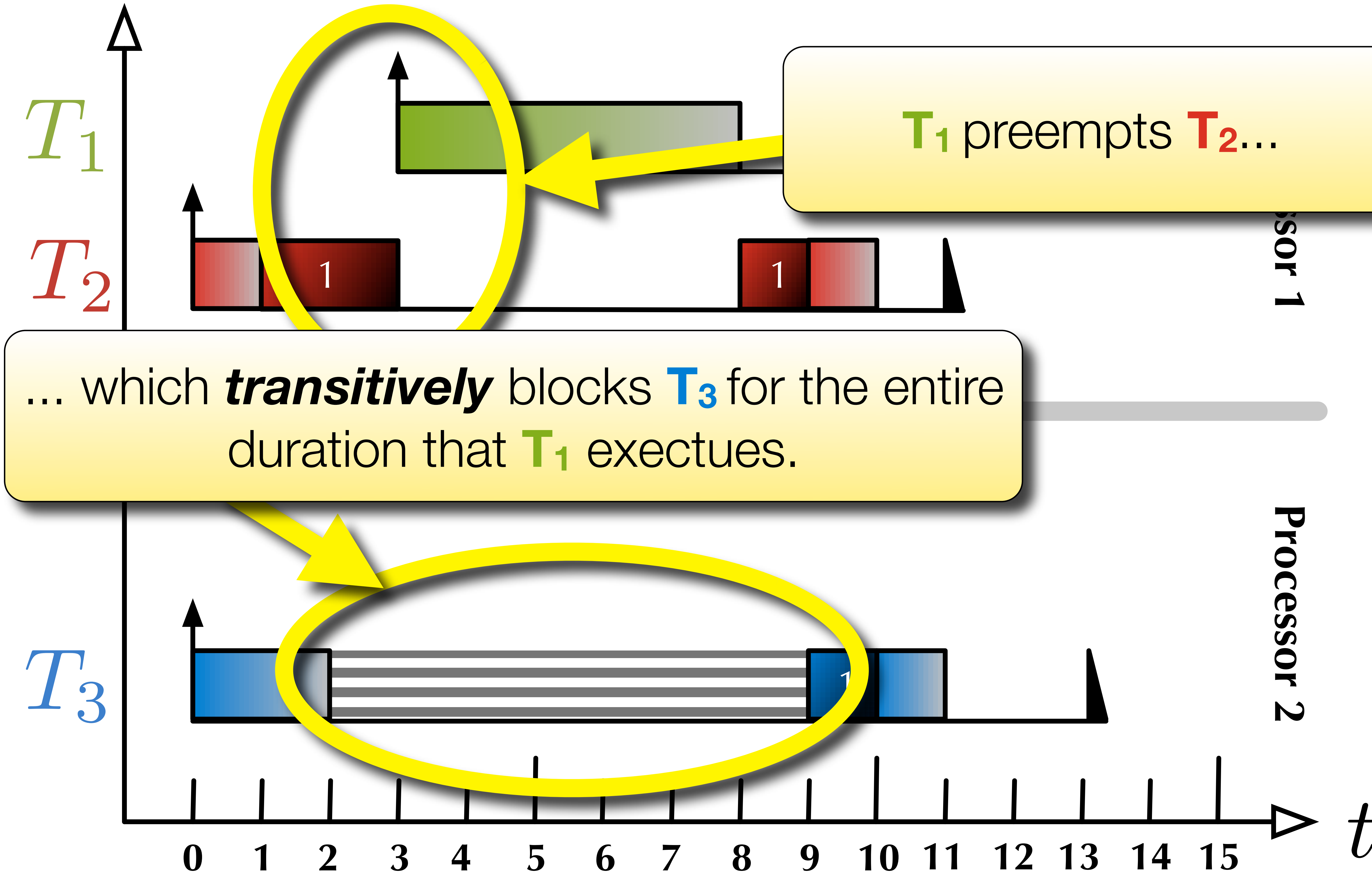
Example:


T2 acquires resource 1...
... and blocks T3.
T1
T2
T3
1
1
1
Processor 1
Processor 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
t

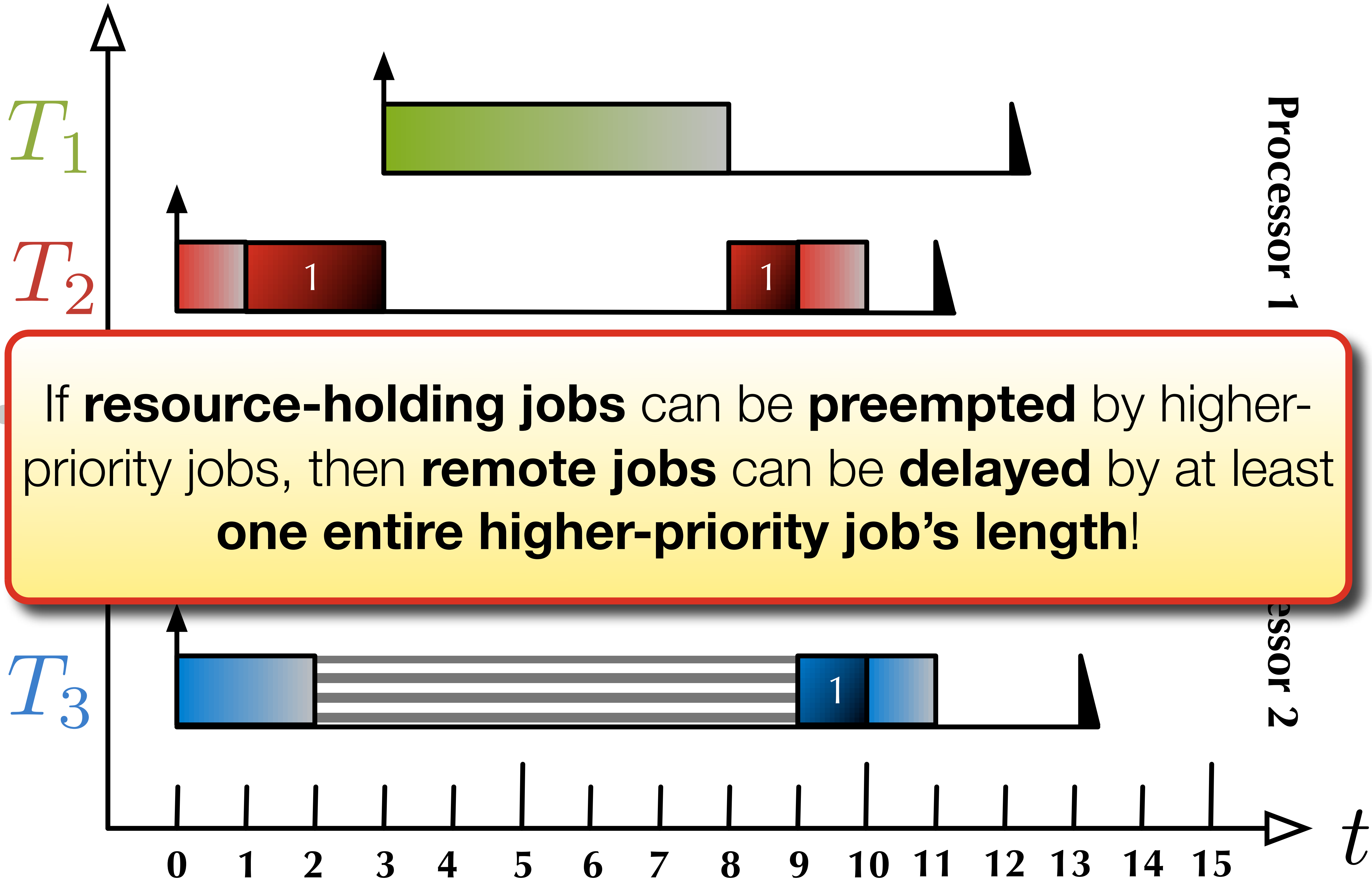
Example: A Naive Approach



Example: A Naive Approach



Example: A Naive Approach



If **resource-holding jobs** can be **preempted** by higher-priority jobs, then **remote jobs** can be **delayed** by at least **one entire higher-priority job's length**!

Quick Review: M-PCP

R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors", *Proceedings of the 10th International Conference on Distributed Computing Systems*, pp.116-123, 1990.

Quick Review: M-PCP

Requests are **ordered by task priority.**

R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors", *Proceedings of the 10th International Conference on Distributed Computing Systems*, pp.116-123, 1990.

Quick Review: M-PCP

Requests are **ordered by task priority.**

Resource-holding jobs have **higher priority** than non-resource-holding jobs; resource-holding jobs **can be preempted** by other resource-holding jobs.

R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors", *Proceedings of the 10th International Conference on Distributed Computing Systems*, pp.116-123, 1990.

Quick Review: M-PCP

Requests are **ordered by task priority.**

Resource-holding jobs have **higher priority** than non-resource-holding jobs; resource-holding jobs **can be preempted** by other resource-holding jobs.

All jobs execute on their **assigned processors**.

R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors", *Proceedings of the 10th International Conference on Distributed Computing Systems*, pp.116-123, 1990.

Quick Review: M-PCP

Requests are **ordered by task priority.**

Resource-holding jobs have **higher priority** than non-resource-holding jobs; resource-holding jobs **can be preempted** by other resource-holding jobs.

All jobs execute on their **assigned processors**.

Doesn't support nesting!

R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors", *Proceedings of the 10th International Conference on Distributed Computing Systems*, pp.116-123, 1990.

Quick Review: D-PCP

R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors", *Proceedings of the 9th Real-Time Systems Symposium*, pp.259-269, 1988.

Quick Review: D-PCP

Requests are **ordered by task priority.**

R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors", *Proceedings of the 9th Real-Time Systems Symposium*, pp.259-269, 1988.

Quick Review: D-PCP

Requests are **ordered by task priority.**

Resource-holding jobs have **higher priority** than non-resource-holding jobs; resource-holding jobs **can be preempted** by other resource-holding jobs.

R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors", *Proceedings of the 9th Real-Time Systems Symposium*, pp.259-269, 1988.

Quick Review: D-PCP

Requests are **ordered by task priority.**

Resource-holding jobs have **higher priority** than non-resource-holding jobs; resource-holding jobs **can be preempted** by other resource-holding jobs.

Resources are **assigned to processors.**
Jobs use RPC to invoke critical sections.

R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors", *Proceedings of the 9th Real-Time Systems Symposium*, pp.259-269, 1988.

Quick Review: D-PCP

Requests are **ordered by task priority.**

Resource-holding jobs have **higher priority** than non-resource-holding jobs; resource-holding jobs **can be preempted** by other resource-holding jobs.

Resources are **assigned to processors.**
Jobs use RPC to invoke critical sections.

Doesn't support nesting!

R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors", *Proceedings of the 9th Real-Time Systems Symposium*, pp.259-269, 1988.

The D-PCP and M-PCP have high implementation overheads.

(in practice, they are used only rarely)

The D-PCP and M-PCP have high implementation overheads.

(in practice, they are used only rarely)

Maybe the **complexity is overkill** in many cases?
Can't we have something **simpler**?

Flexible **M**ultiprocessor **L**ocking **P**rotocol

A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors", *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 47-57, August 2007.

Flexible **M**ultiprocessor **L**ocking **P**rotocol

- Originally proposed for **global** and **partitioned** **earliest-deadline-first (EDF) scheduling.**
- **generalizes** most prior P-EDF schemes
- The FMLP supports both **spin-based** **locks** and **suspension-based** **locks**.
- The FMLP supports **arbitrary nesting** of resources.

A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors", *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 47-57, August 2007.

Flexible **M**ultiprocessor **L**ocking **P**rotocol

- Originally proposed for **global** and **partitioned** **earliest-deadline-first (EDF) scheduling.**
- **generalizes** most prior P-EDF schemes
- The FMLP supports both **spin-based** **locks** and **suspension-based** **locks**.
- The FMLP supports **arbitrary nesting** of resources.

In this work, we **extended the FMLP** to **partitioned static-priority** scheduling.

A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors", *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 47-57, August 2007.

Flexible Multiproce… ol

We call resources protected by **spin-based** locks "**short**."

- Originally proposed f… **earliest-deadline-first (EDF) scheduling.**
- **generalizes** most prior P-EDF schemes
- The FMLP supports both **spin-based locks** and **suspension-based locks**.
- The FMLP supports **arbitrary nesting** of resources.

In this work, we **extended the FMLP** to **partitioned static-priority** scheduling.

A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors", *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 47-57, August 2007.

We call resources protected by **suspension-based** locks “**long**.”

We call resources protected by **spin-based** locks “**short**.”

…e-first (EDF) **scheduling.**

- ➡ **generalizes** most prior P-EDF schemes
- ➡ The FMLP supports both **spin-based locks** and **suspension-based locks**.
- ➡ The FMLP supports **arbitrary nesting** of resources.

In this work, we **extended the FMLP** to **partitioned static-priority** scheduling.

A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors", *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 47-57, August 2007.

FMLP – Design

*"Design a protocol for the common case.
Use the most-simple solution possible."*

FMLP – Design

"Design a protocol for the common case. Use the most-simple solution possible."

Rationale

1. Complex designs are **hard to analyze**.
2. Complex designs are **hard to implement** (and thus tend to have higher overheads).
3. It's **easier to refine** an existing simple protocol then it is to "speed up" a complex protocol.

FMLP – The Common Case

"Most critical sections are short (1-5μs). Nesting is somewhat rare."

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

FMLP – The Common Case

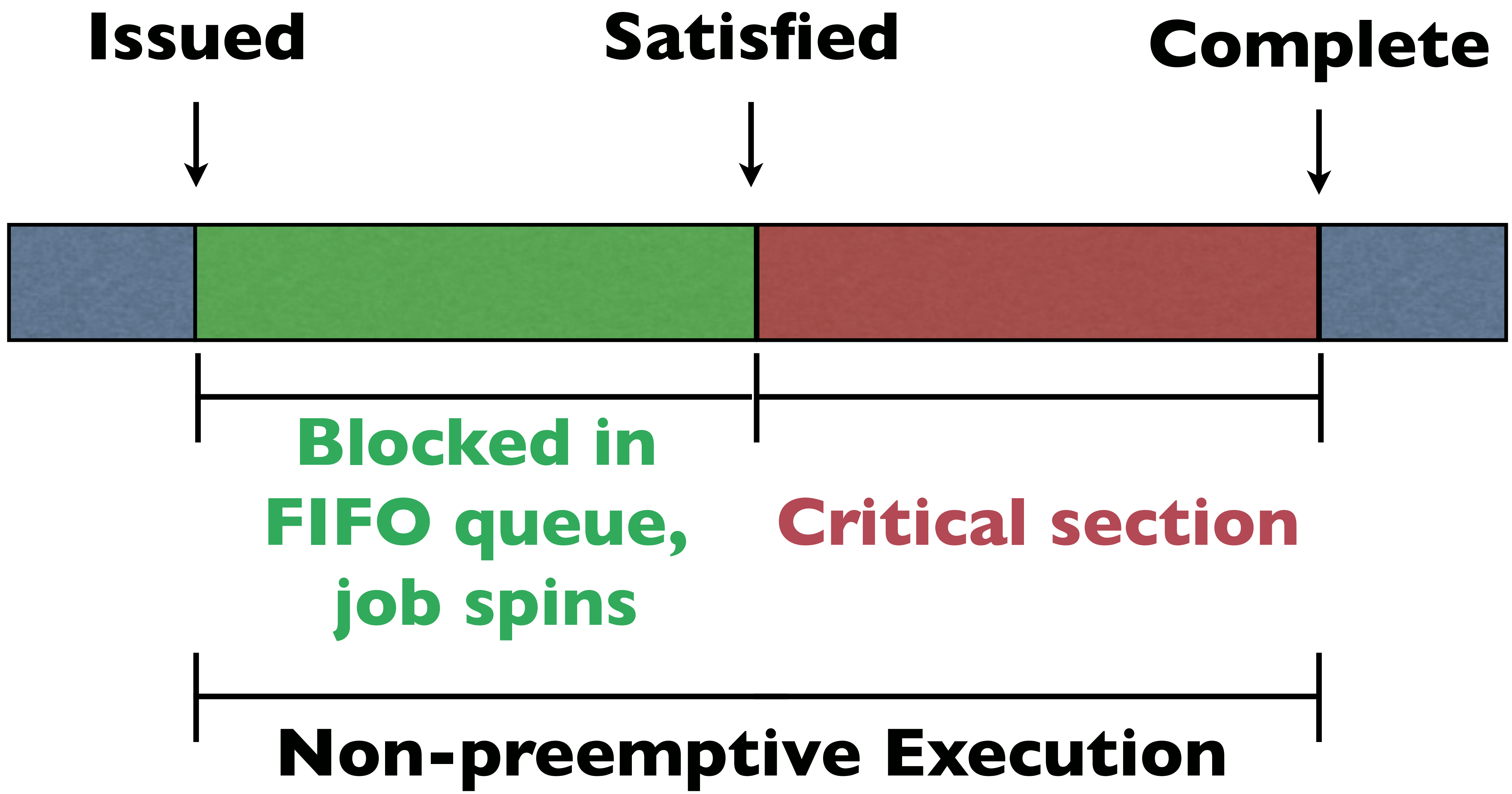
"Most critical sections are short (1-5µs). Nesting is somewhat rare."

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Choices

1. Use **FIFO** everywhere. No priority queues.
2. Use **non-preemptive execution** where possible to simplify analysis.
3. Use a very **simple deadlock avoidance** mechanism.

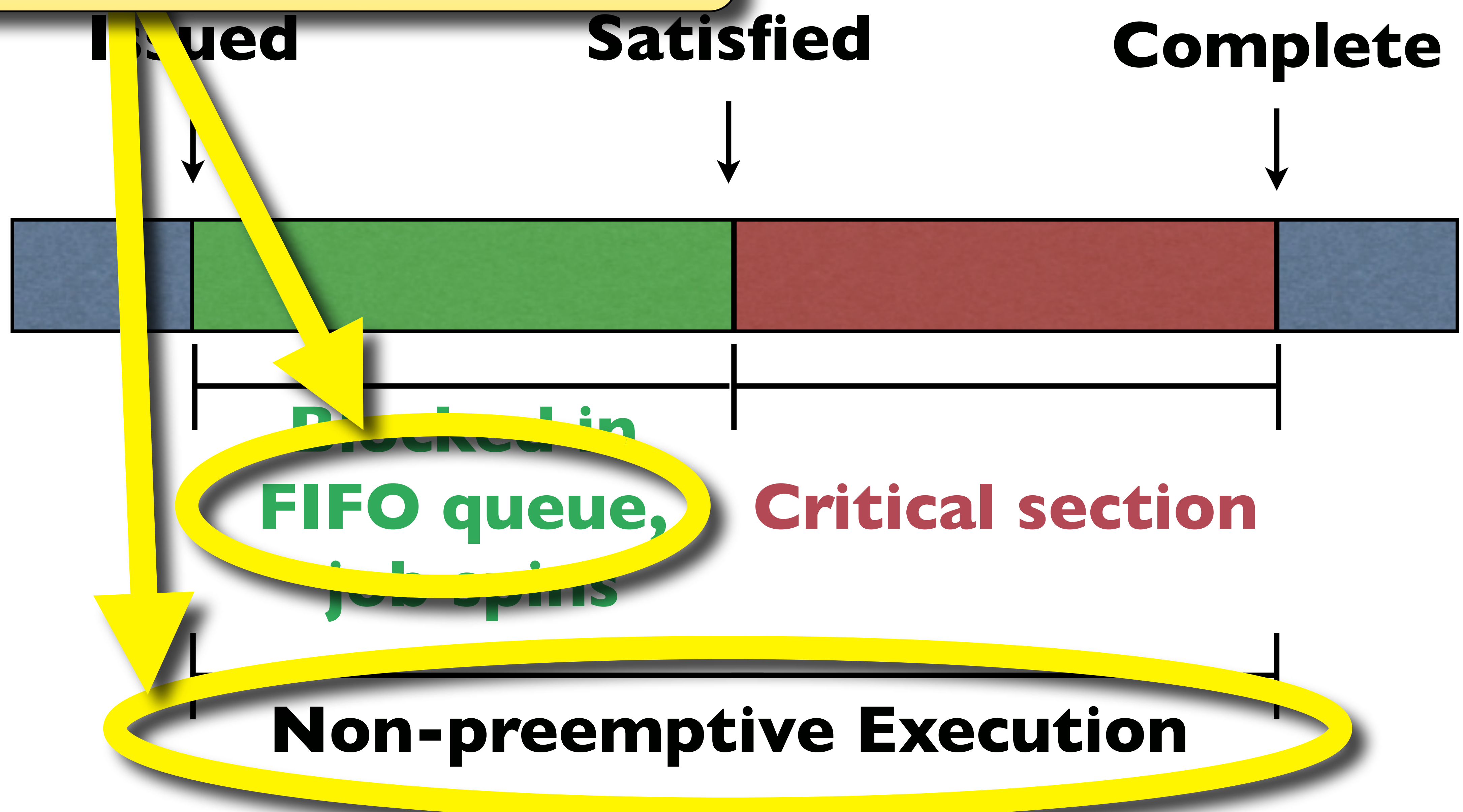
FMLP – Short Resources (Queue Lock)



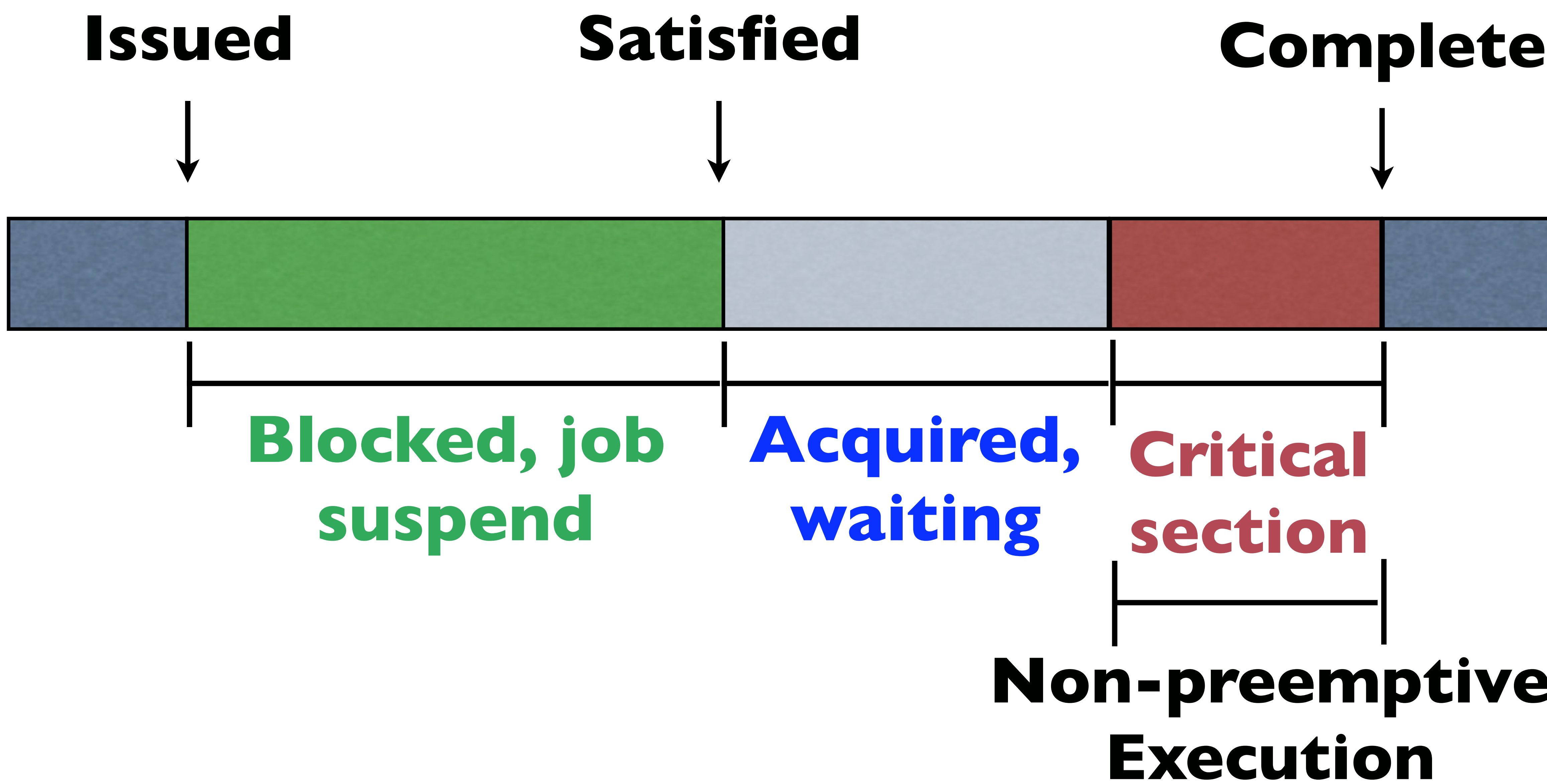
FMLP – Short Resources

e Lock)

This makes analysis easy.

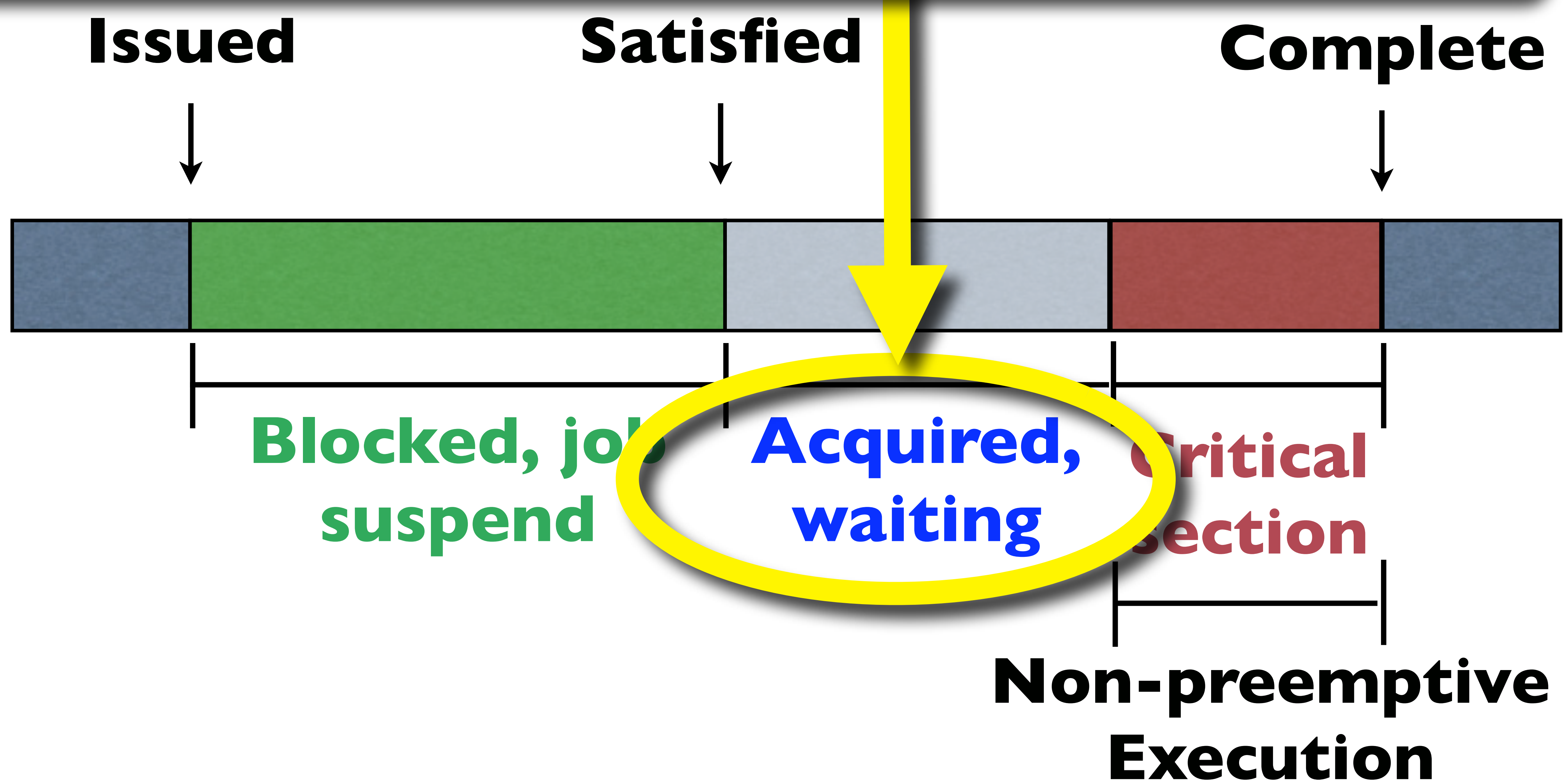


FMLP – Long Resources (Semaphore)



Because the job released the CPU it may be **blocked when it returns**.

Bounding this as tightly as possible is crucial to performance: The FMLP uses priority-boosting.



FMLP – Deadlock Avoidance

We use a very simple mechanism to avoid deadlock:

1. Assign short/long resources to **groups**
2. Two resources are in the same group if requests for them may be nested
3. Associate a **group lock** with each group
4. **Before accessing a resource, must first acquire its group lock**.

FMLP Deadlock Avoidance

A "classic" deadlock scenario:

Job A	**Job B**	Time
Acquire resource **Y**	Acquire resource **X**	↓
Blocked trying to acquire **X**	Blocked trying to acquire **Y**	

Deadlock!

4. **Before accessing a resource, must first acquire its group lock**.

Group locks solve this problem

Job A	**Job B**
Acquire group lock "**XY**"	
Access **Y**	
Access **X**	
Release group lock "**XY**"	
	Acquire group lock "**XY**"
	Access **X**
	Access **Y**
	Release group lock "**XY**"

Time

4. **Before accessing a resource, must first acquire its group lock.**

Group locks solve this problem

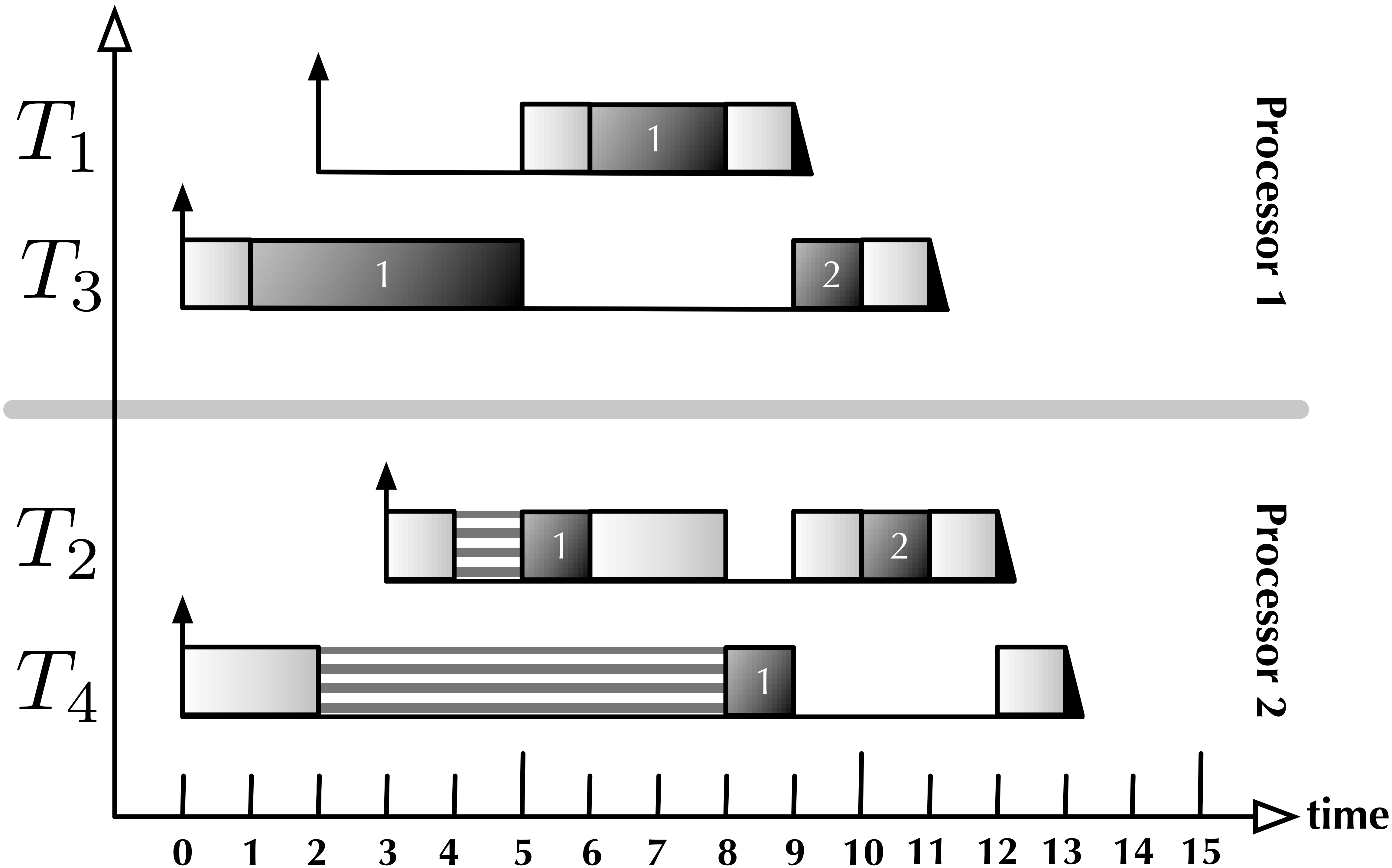
Job A	**Job B**
Acquire group lock "**XY**"	
Access **Y**	
Access **X**	
Release group lock "**XY**"	
	Acquire group lock "**XY**"
	Access **X**
	Access **Y**
	Release group lock "**XY**"

Time

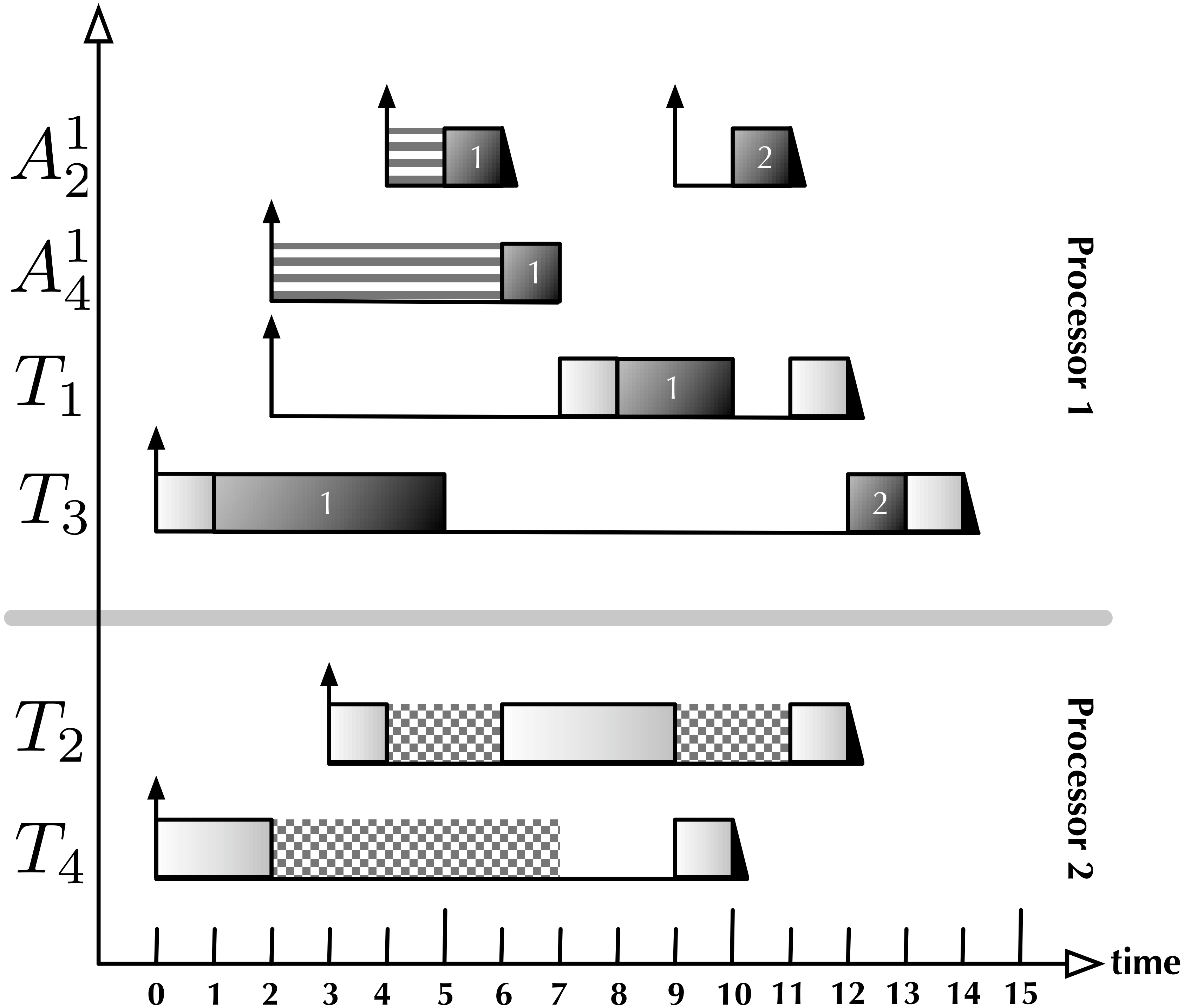
Embarrassingly simple. But:

- Prior multiprocessor work **doesn't support nesting** at all.
- Obtaining ***provably*** better mechanisms is non-trivial.

M-PCP

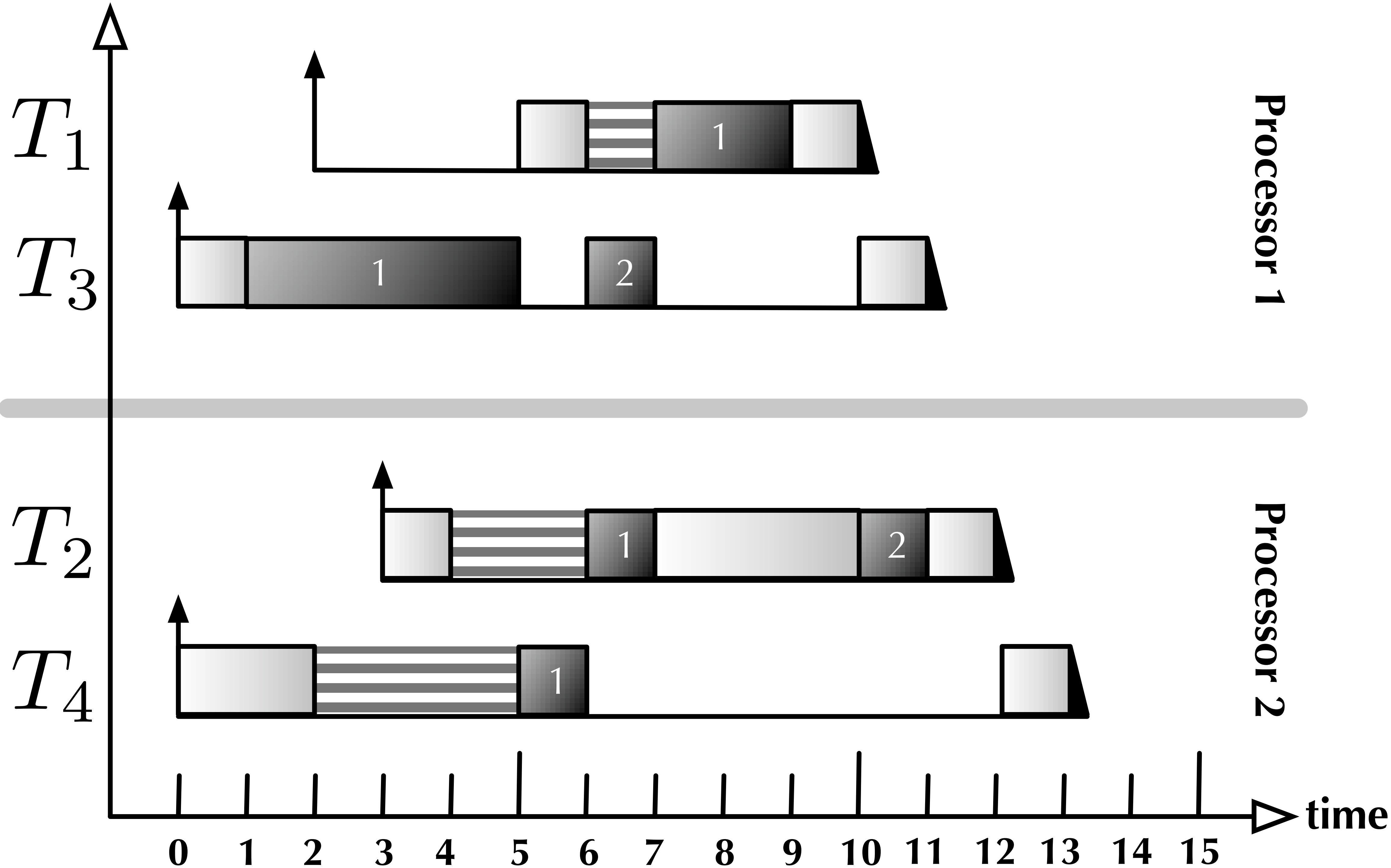

T_1
T_3
T_2
T_4
1
1
2
1
2
1
Processor 1
Processor 2
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

D-PCP



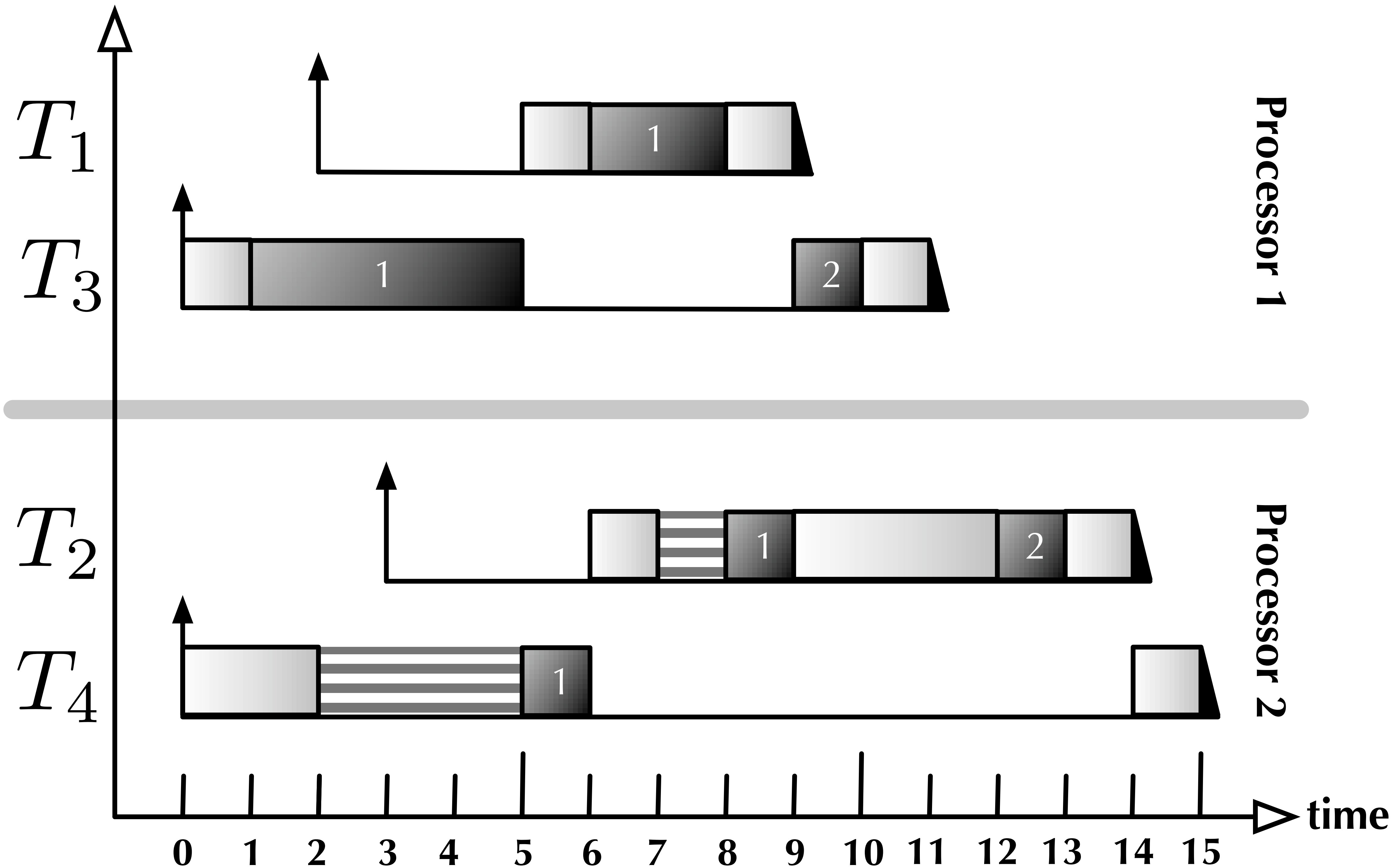
A_2^1
A_4^1
T_1
T_3
T_2
T_4
1
2
1
1
1
2
Processor 1
Processor 2
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

FMLP (long)



T_1
T_3
T_2
T_4
1
1
2
1
2
1
Processor 1
Processor 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
time

FMLP (short)



T_1
T_3
T_2
T_4
1
1
2
1
2
1
Processor 1
Processor 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
time

Some Results

M-PCP vs. D-PCP vs. FMLP-L vs. FMLP-S

Some Results

M-PCP vs. D-PCP vs. FMLP-L vs. FMLP-S

Does the FMLP's simplicity sacrifice performance?

Methodology

feather trace

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

feather trace

1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}
2. Generated lots of **random task sets**.

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}
2. Generated lots of **random task sets**.
3. Executed task sets on LITMUS^{RT}; **traced overheads** with *Feather-Trace*.

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}
2. Generated lots of **random task sets**.
3. Executed task sets on LITMUS^{RT}; **traced overheads** with *Feather-Trace*.
4. Distilled **overhead formulas** from trace data.

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

feather trace

1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}
2. Generated lots of **random task sets**.
3. Executed task sets on LITMUS^{RT}; **traced overheads** with *Feather-Trace*.
4. Distilled **overhead formulas** from trace data.
5. Accounted for overheads in **schedulability analysis** and **blocking term** calculations.

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}
2. Generated lots of **random task sets**.
3. Executed task sets on LITMUS^{RT}; **traced overheads** with *Feather-Trace*.
4. Distilled **overhead formulas** from trace data.
5. Accounted for overheads in **schedulability analysis** and **blocking term** calculations.
6. Generated over **13 million random task sets** (in total) and tested whether they remained **schedulable** with blocking terms/overheads.

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

Methodology

feather trace

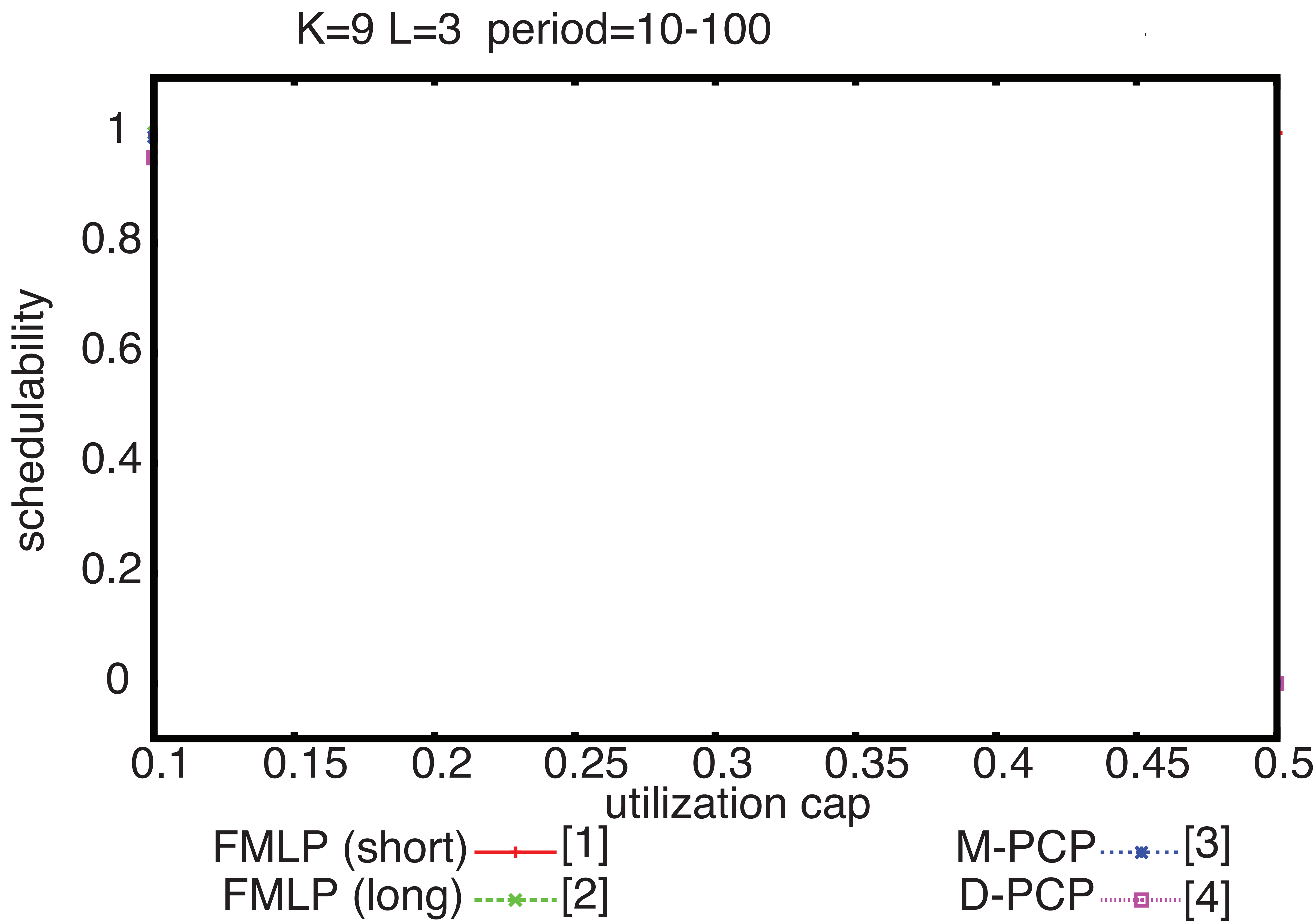
1. **Implemented** PCP, SRP, D-PCP, M-PCP, FMLP in LITMUS^{RT}
2. Generated lots of **random task sets**.
3. Executed task sets on LITMUS^{RT}: **traced overheads** with *Feather-*
4. Distille
5. Accour ... **s** and **blocki**
6. Generated over **13 million random task sets** (in total) and tested whether they remained **schedulable** with blocking terms/overheads.

Our platform:

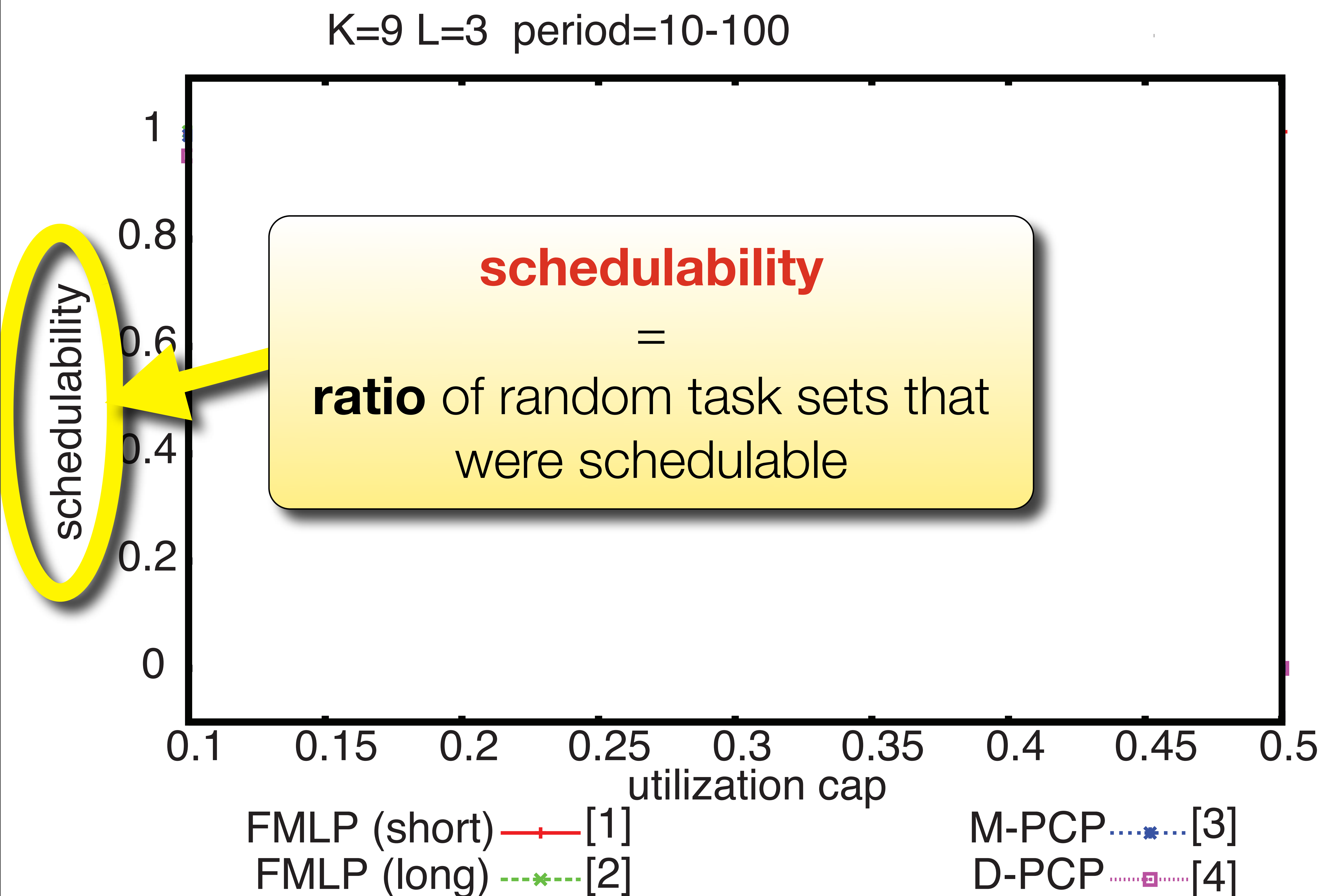
4-way 2.7 GHz Intel Xeon SMP
512K L2 cache per processor
2 Gb RAM

B. Brandenburg and J. Anderson, "Feather-Trace: A Light-Weight Event Tracing Toolkit", *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 20-27, July 2007.

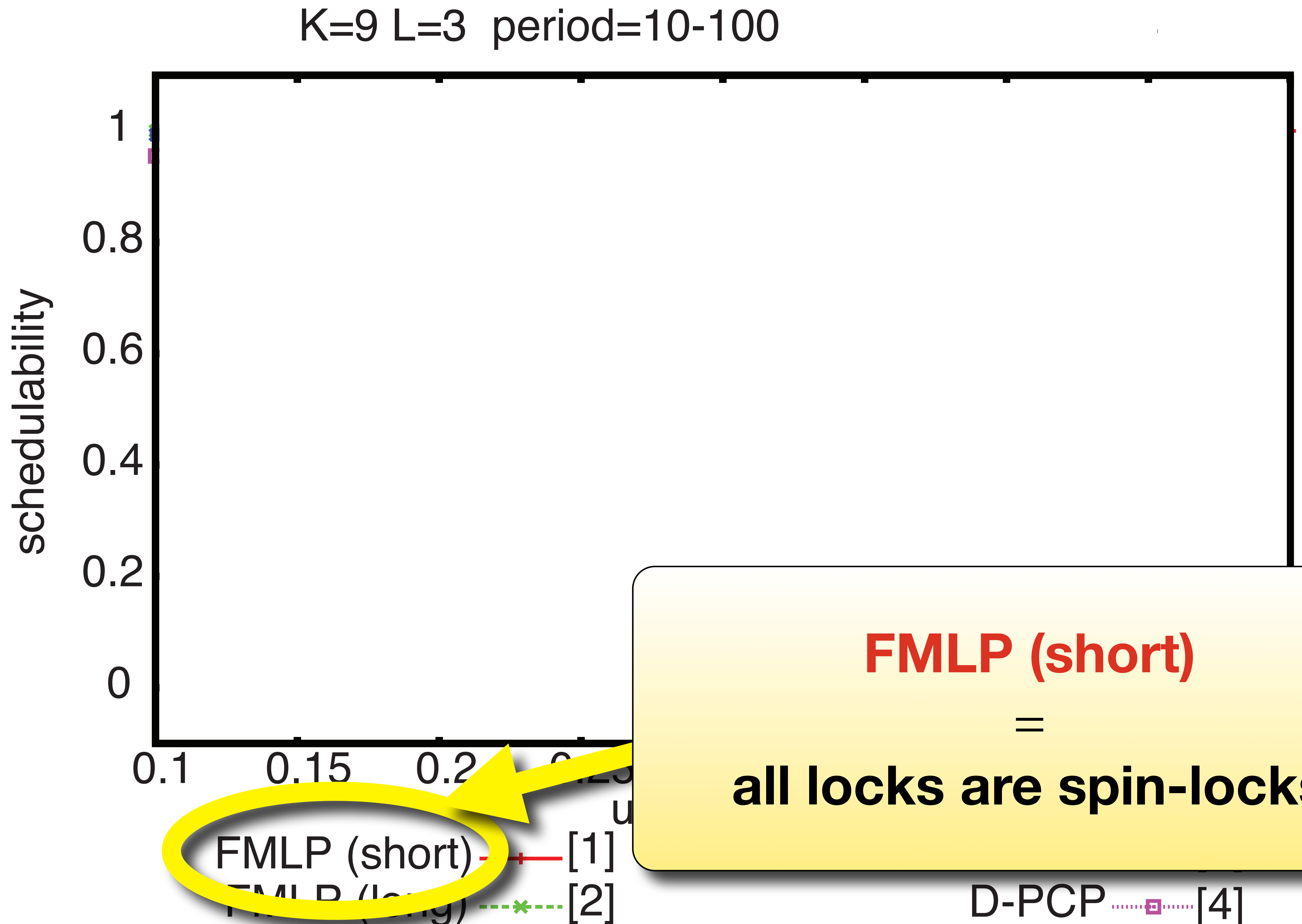
Schedulability vs. Utilization



Schedulability vs. Utilization



Schedulability vs. Utilization

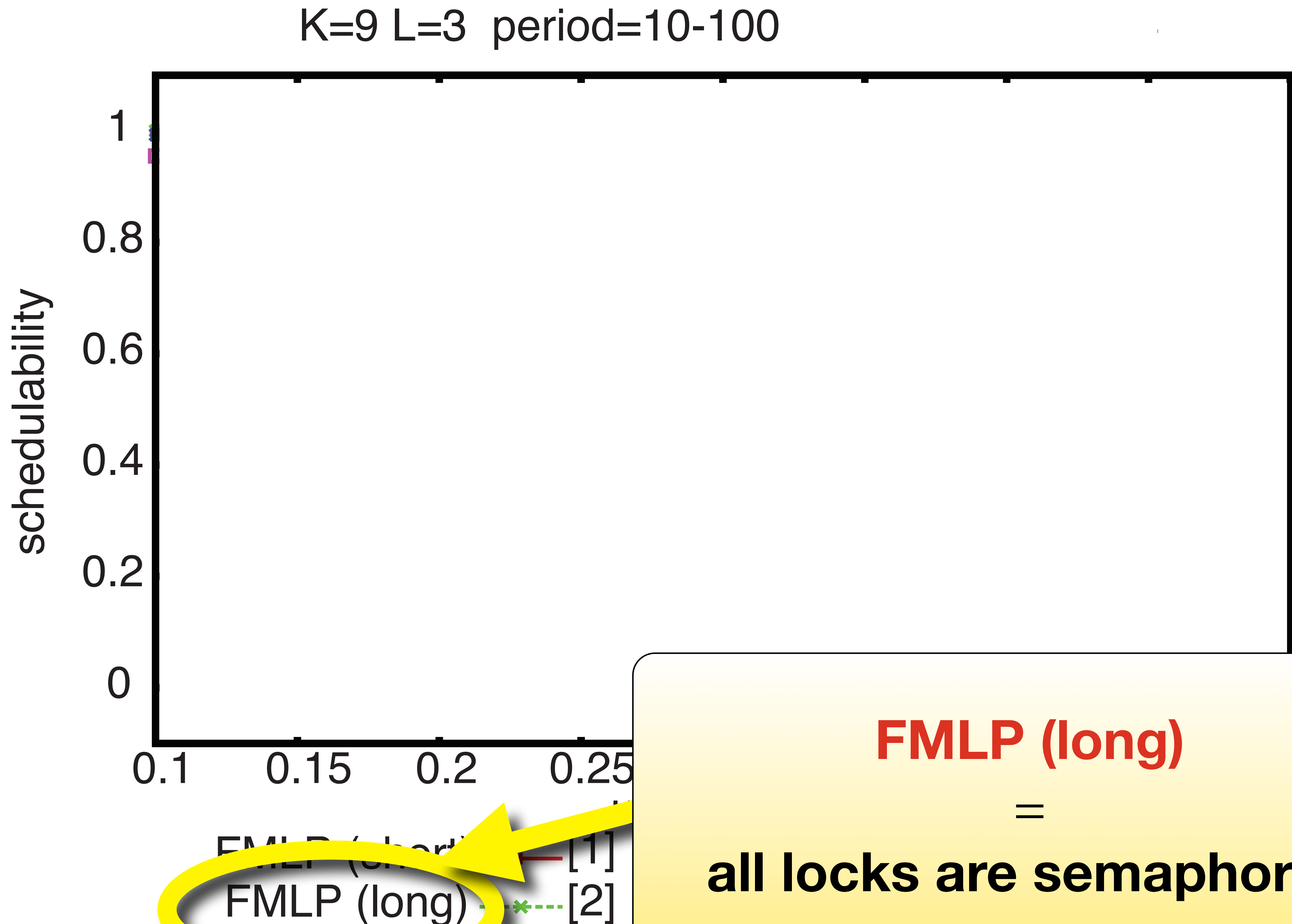


FMLP (short)

=

all locks are spin-locks

Schedulability vs. Utilization

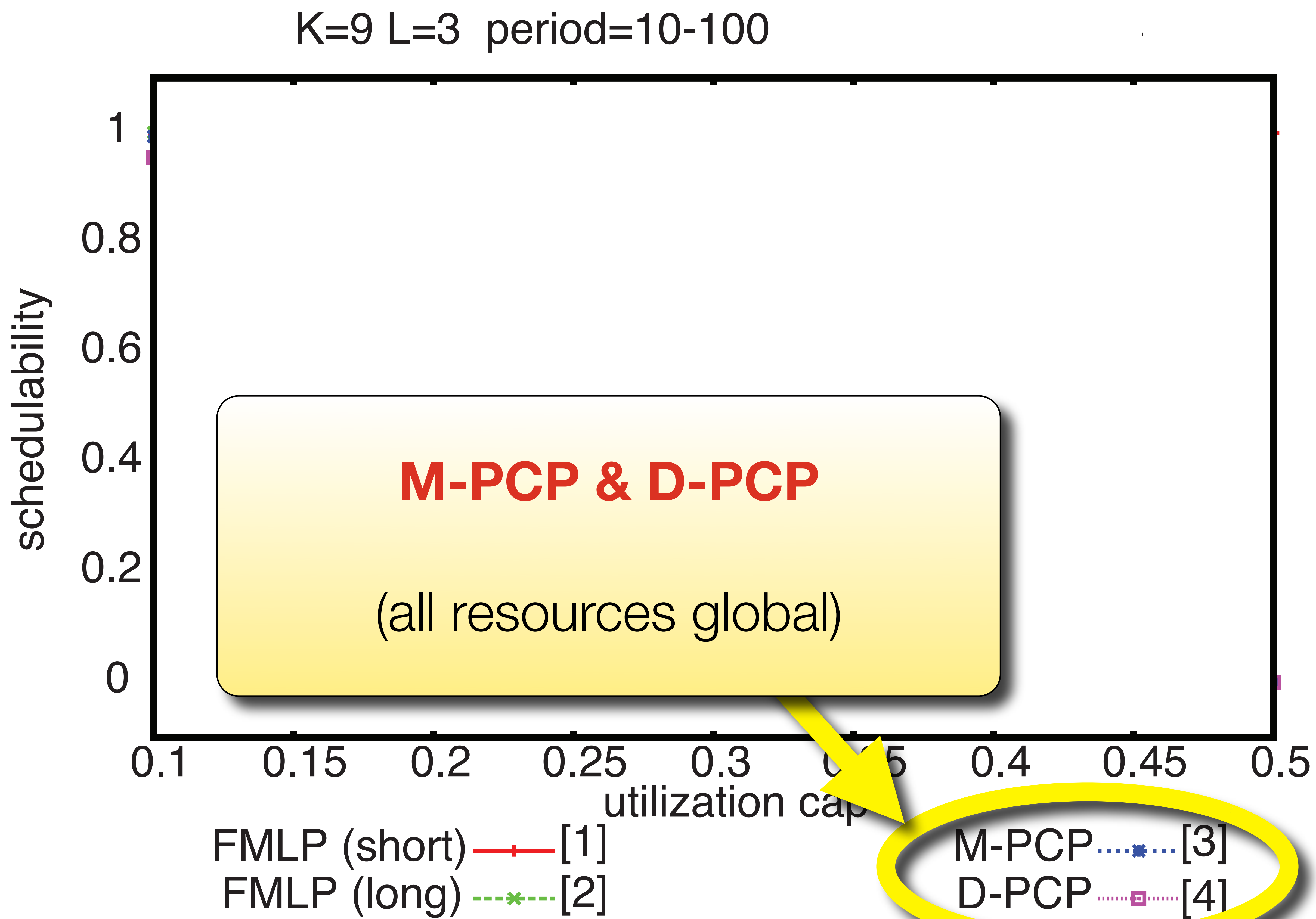


FMLP (long)

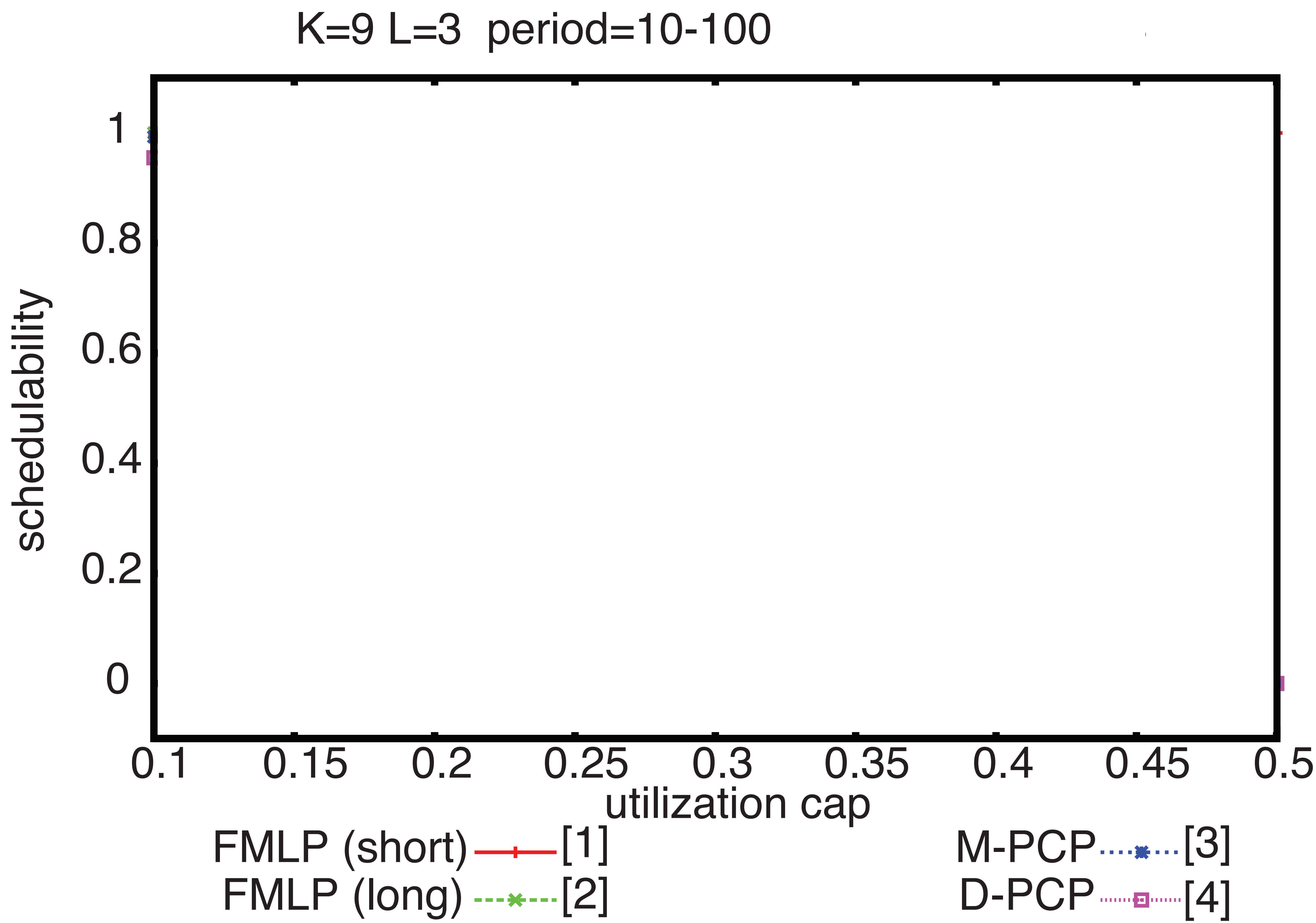
=

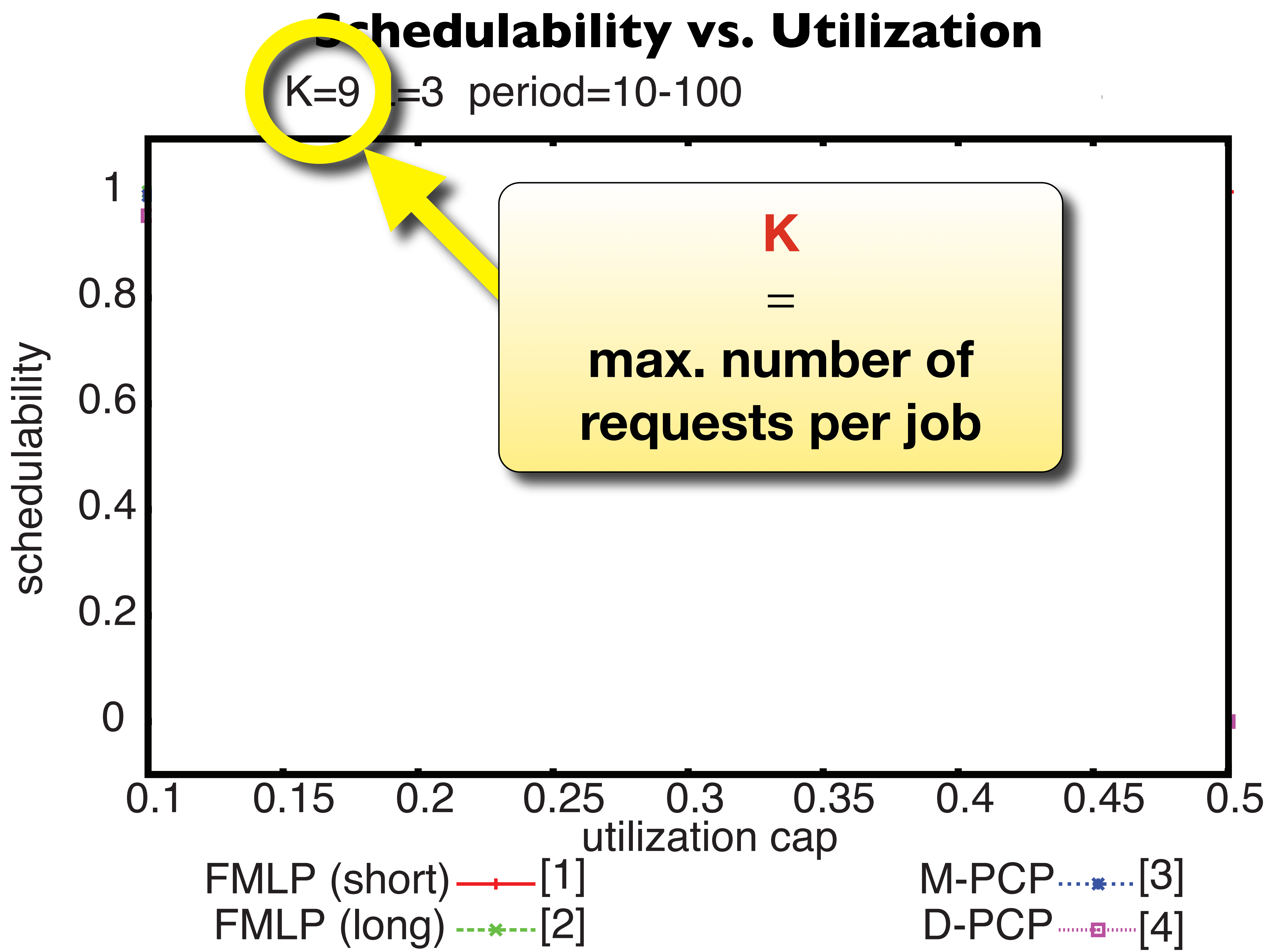
all locks are semaphores

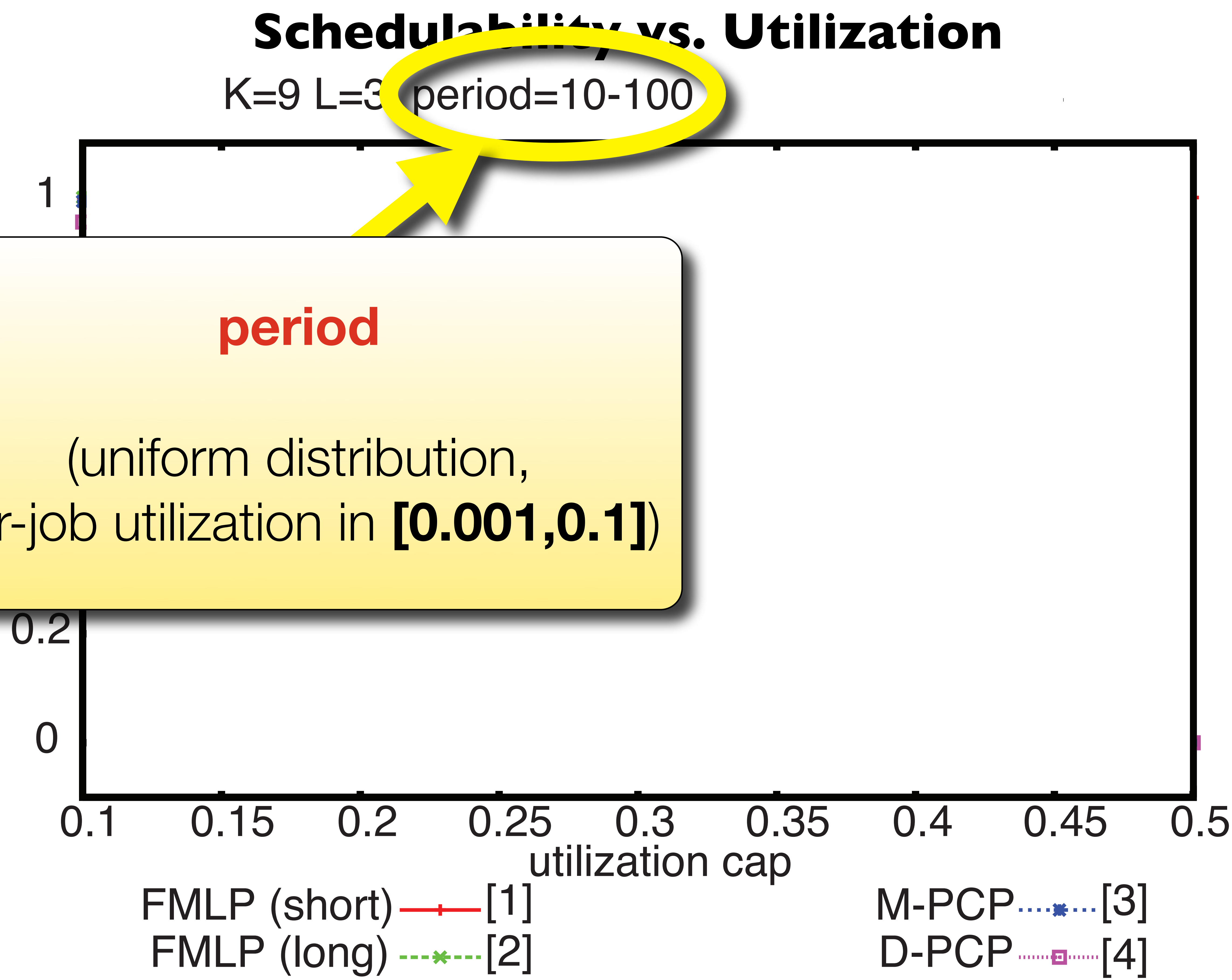
Schedulability vs. Utilization



Schedulability vs. Utilization



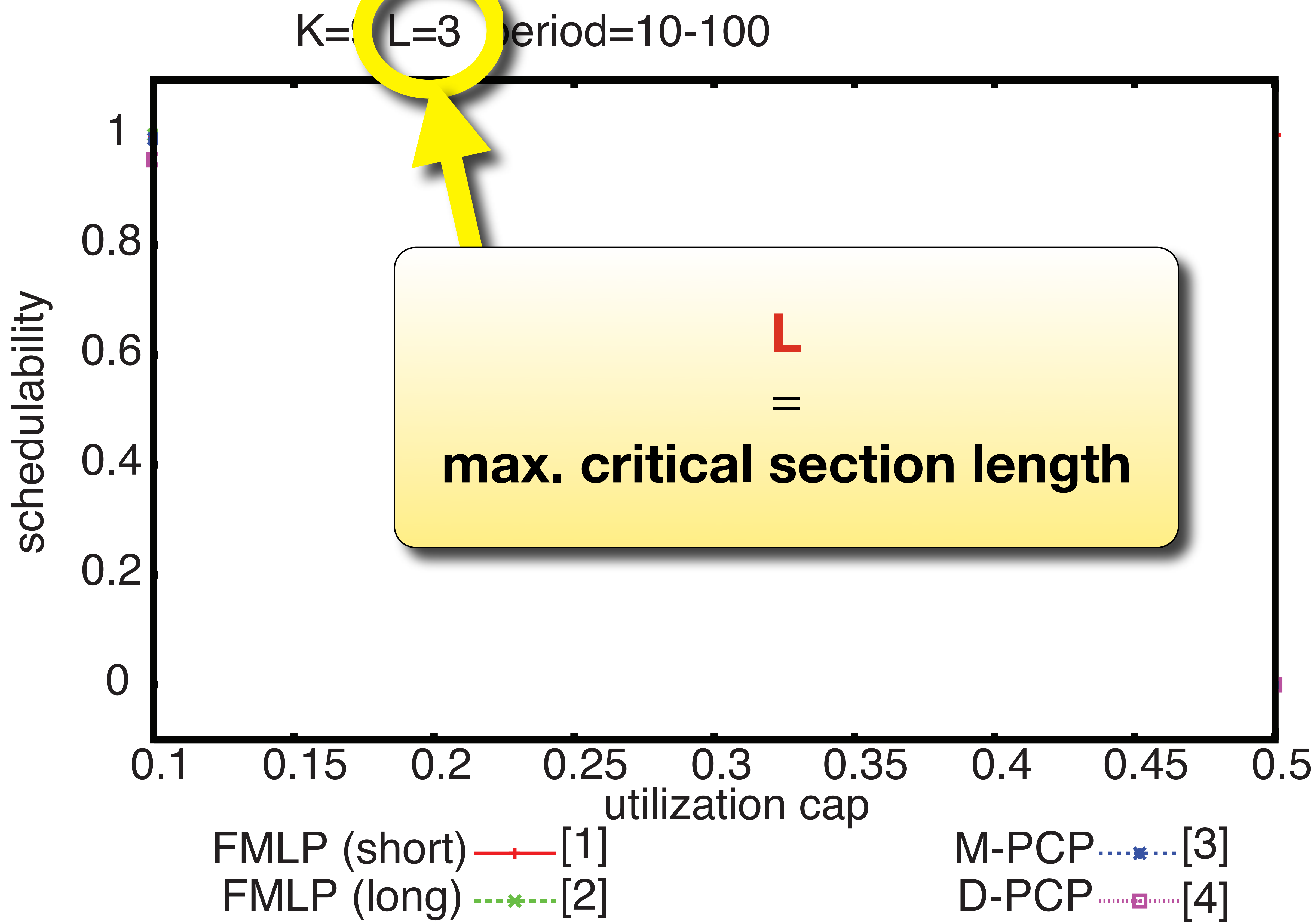

Schedulability vs. Utilization
K=9 L=3 period=10-100
K
=
max. number of requests per job
schedulability
1
0.8
0.6
0.4
0.2
0
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
utilization cap
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]



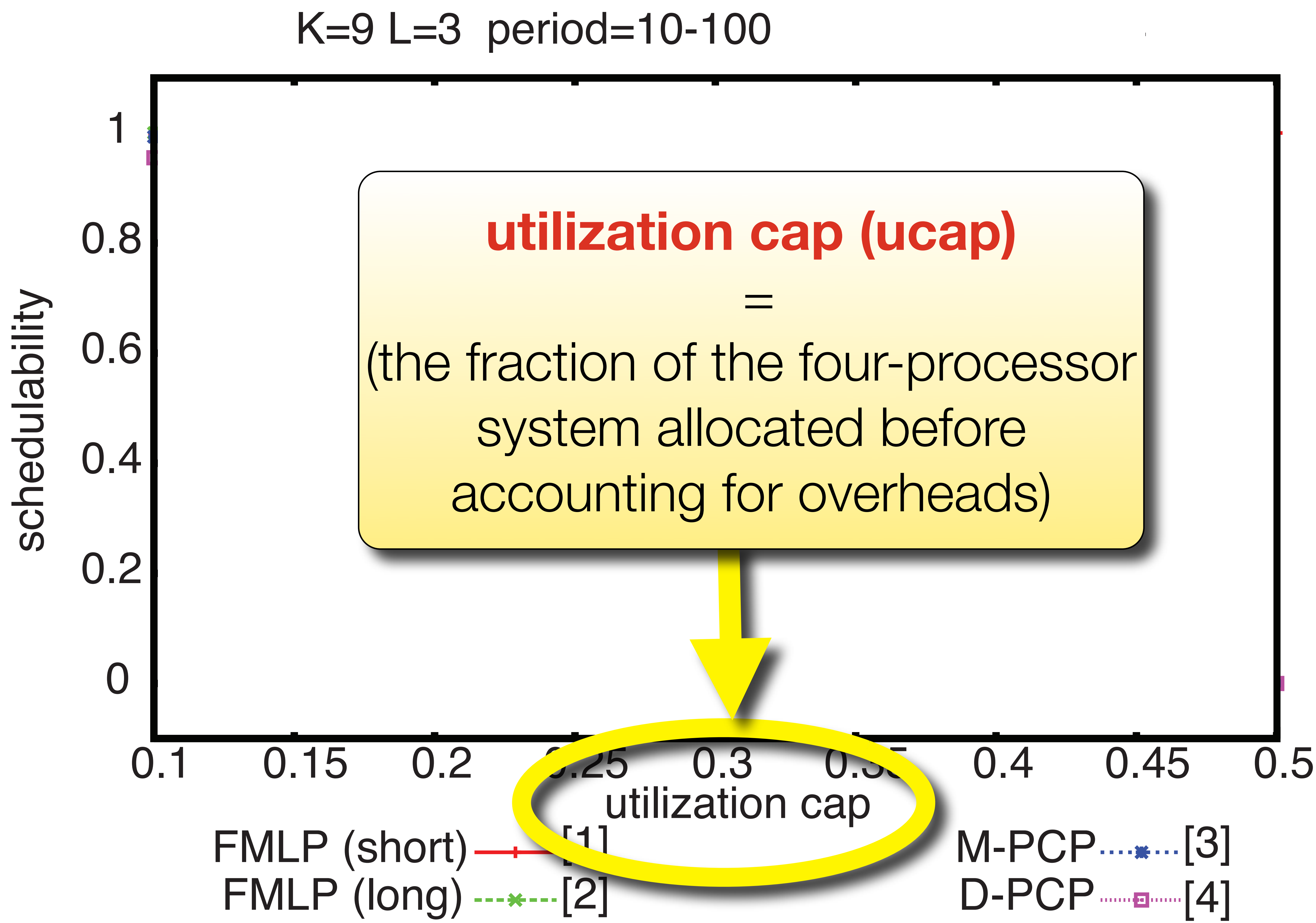
period

(uniform distribution,
per-job utilization in **[0.001,0.1]**)

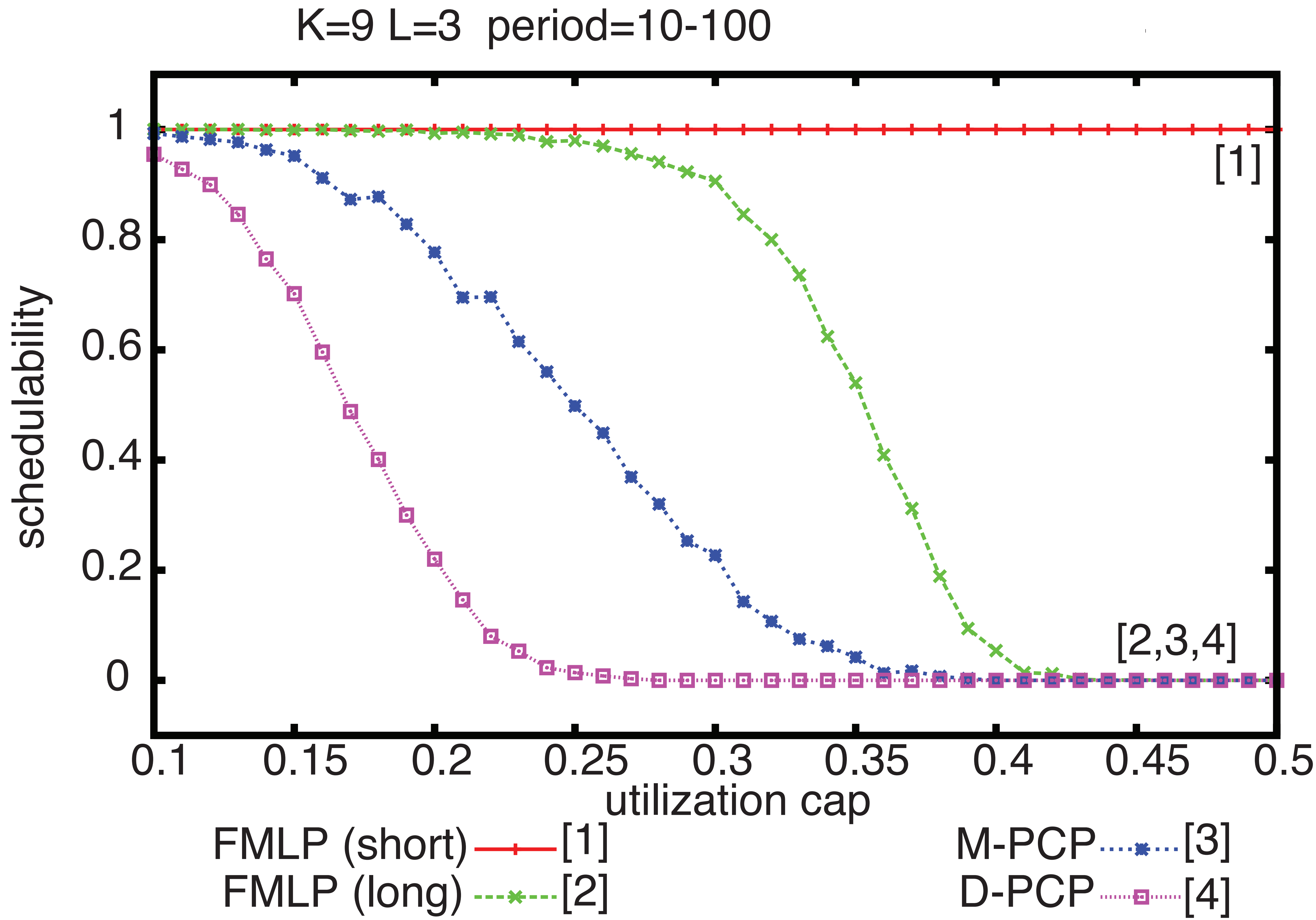
Schedulability vs. Utilization



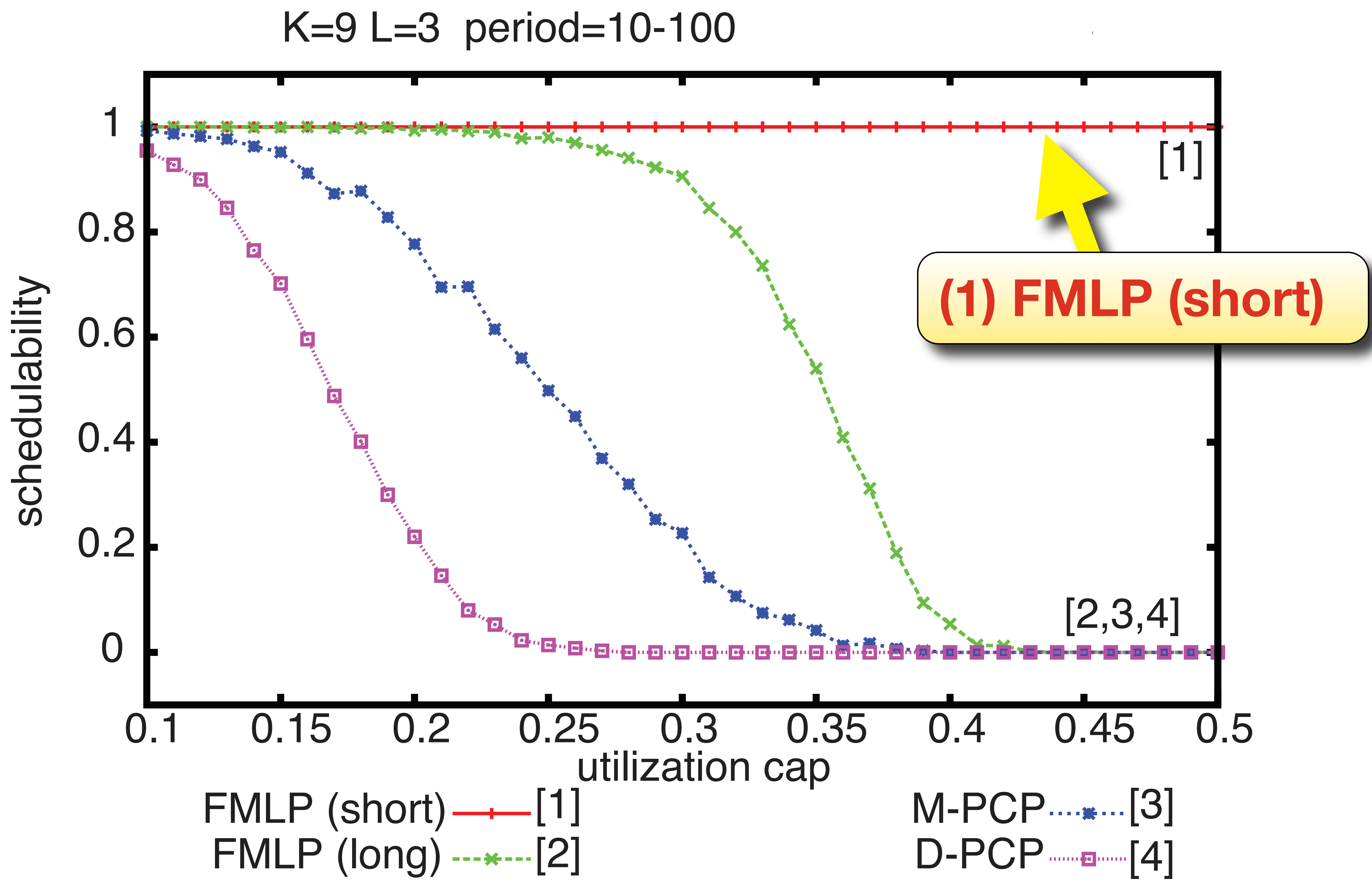
Schedulability vs. Utilization



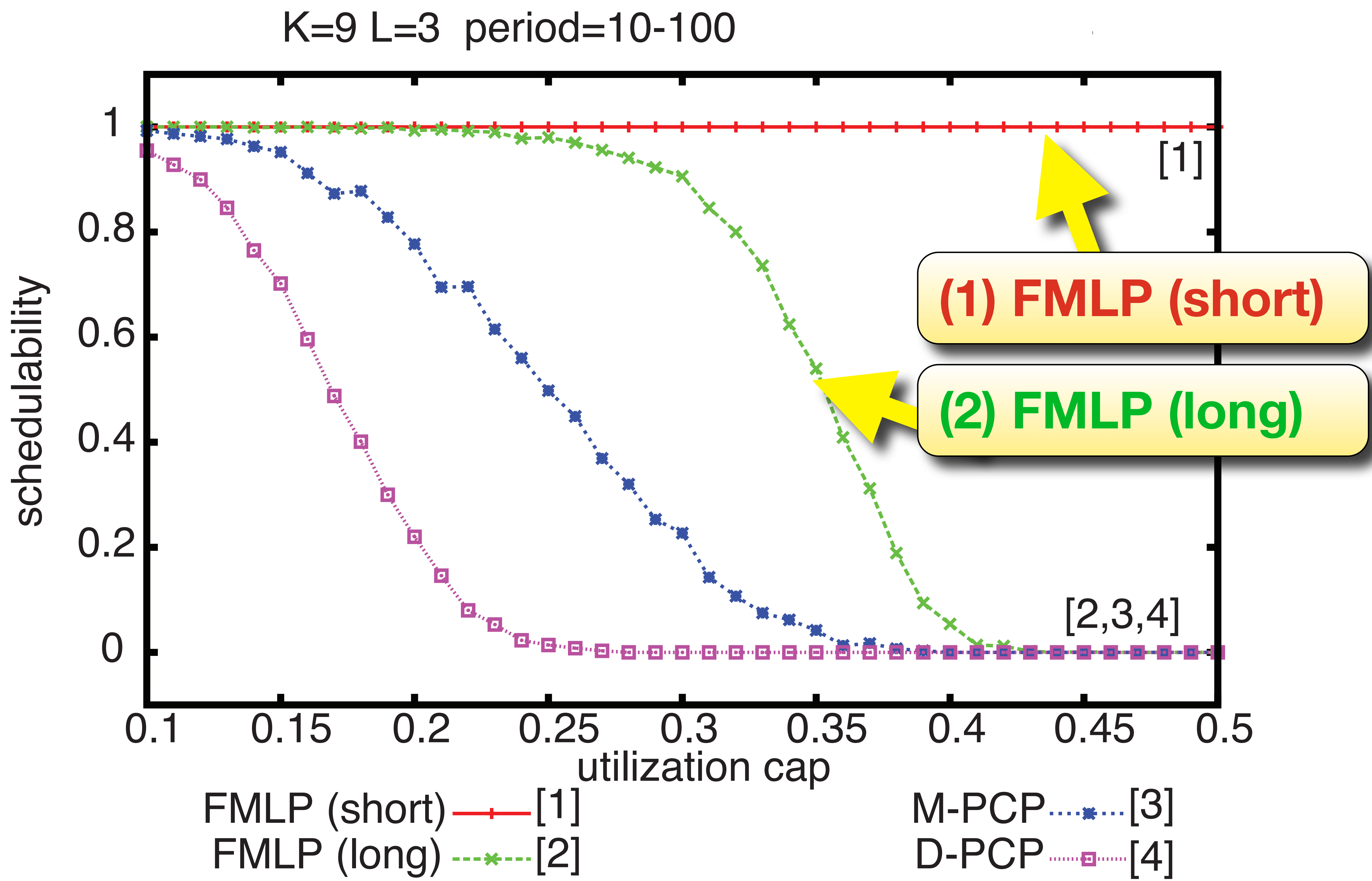
Schedulability vs. Utilization

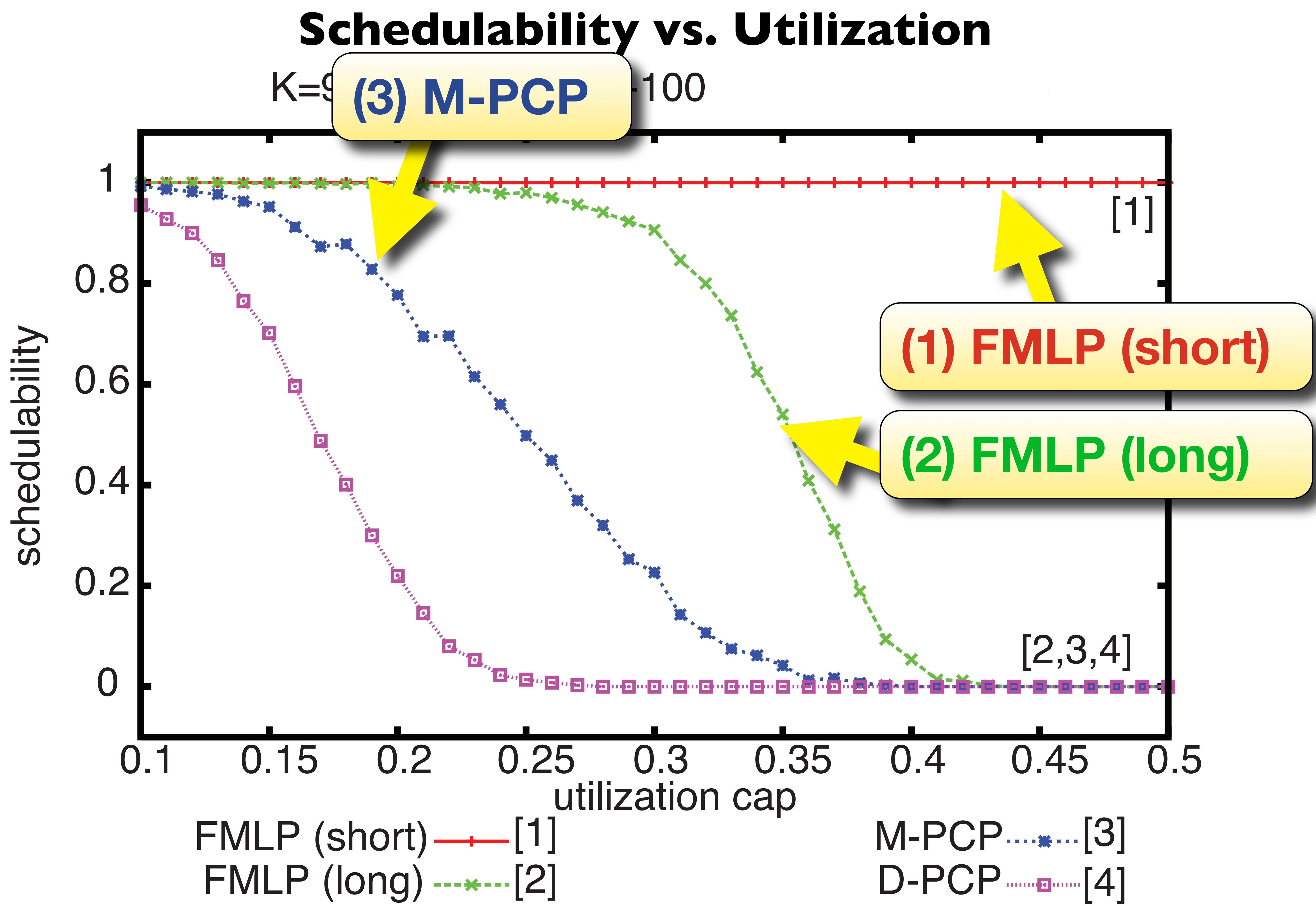


Schedulability vs. Utilization

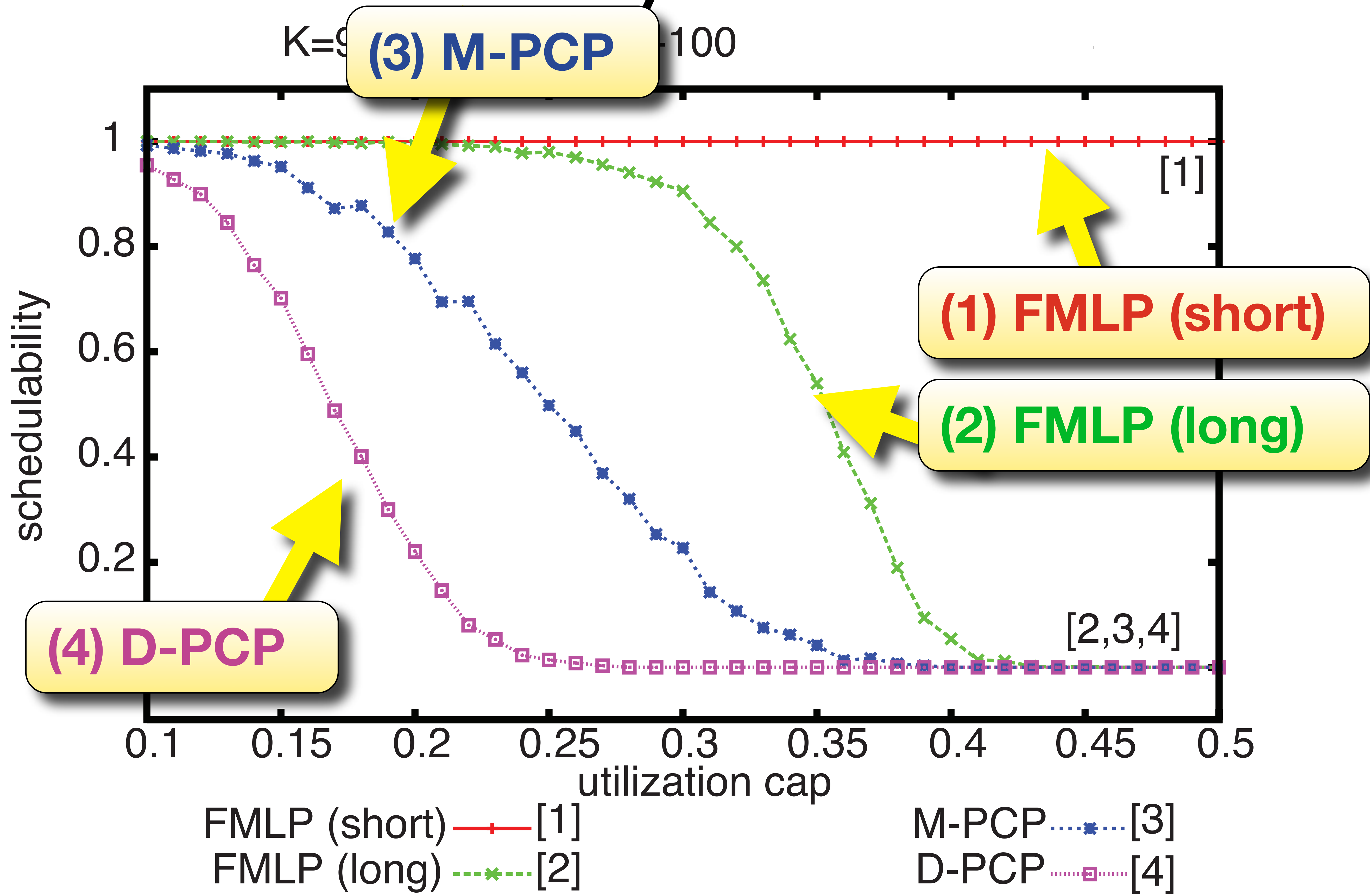


Schedulability vs. Utilization

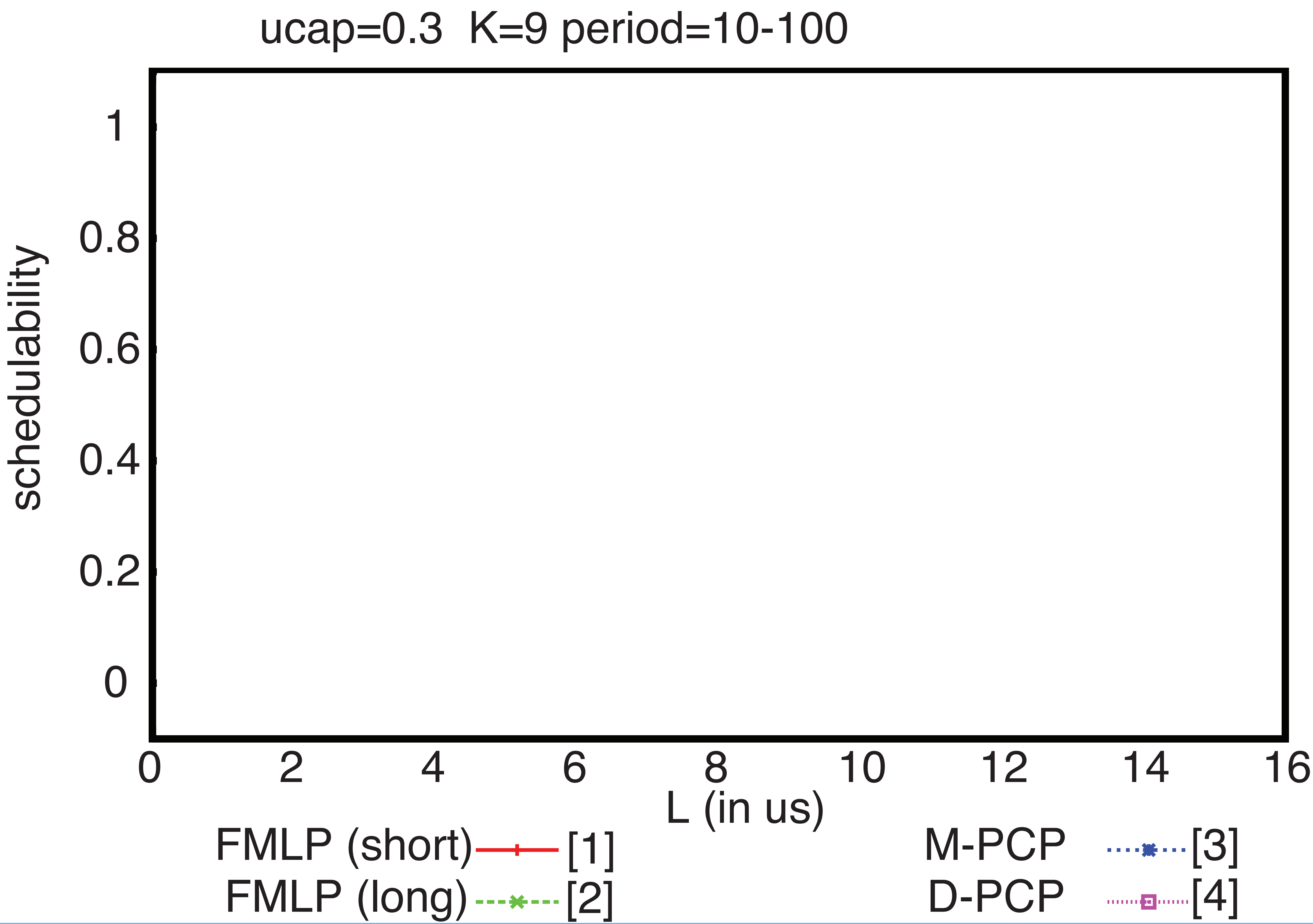



Schedulability vs. Utilization
(3) M-PCP
(1) FMLP (short)
(2) FMLP (long)
schedulability
utilization cap
1
0.8
0.6
0.4
0.2
0
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
[1]
[2,3,4]
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]

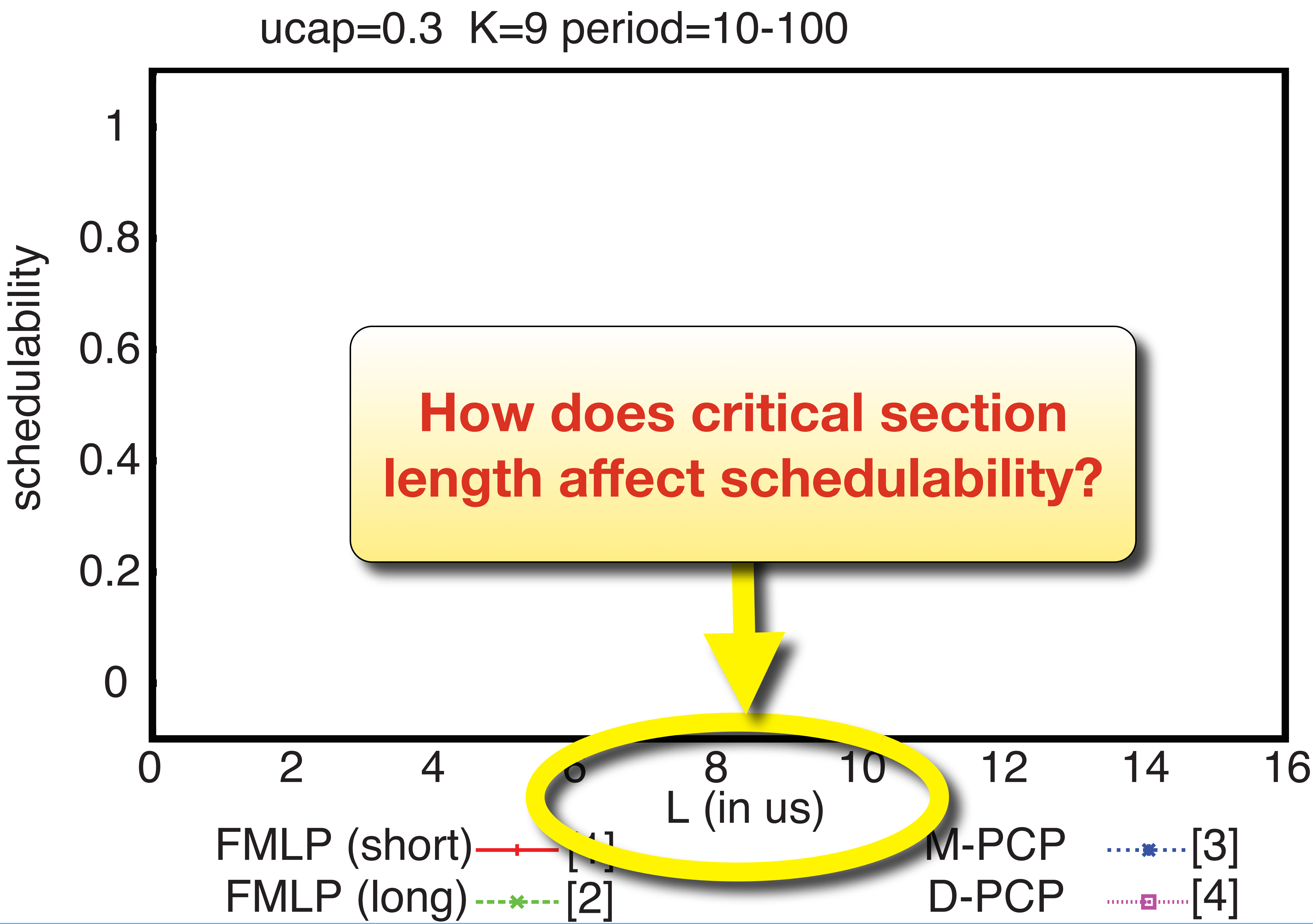
Schedulability vs. Utilization



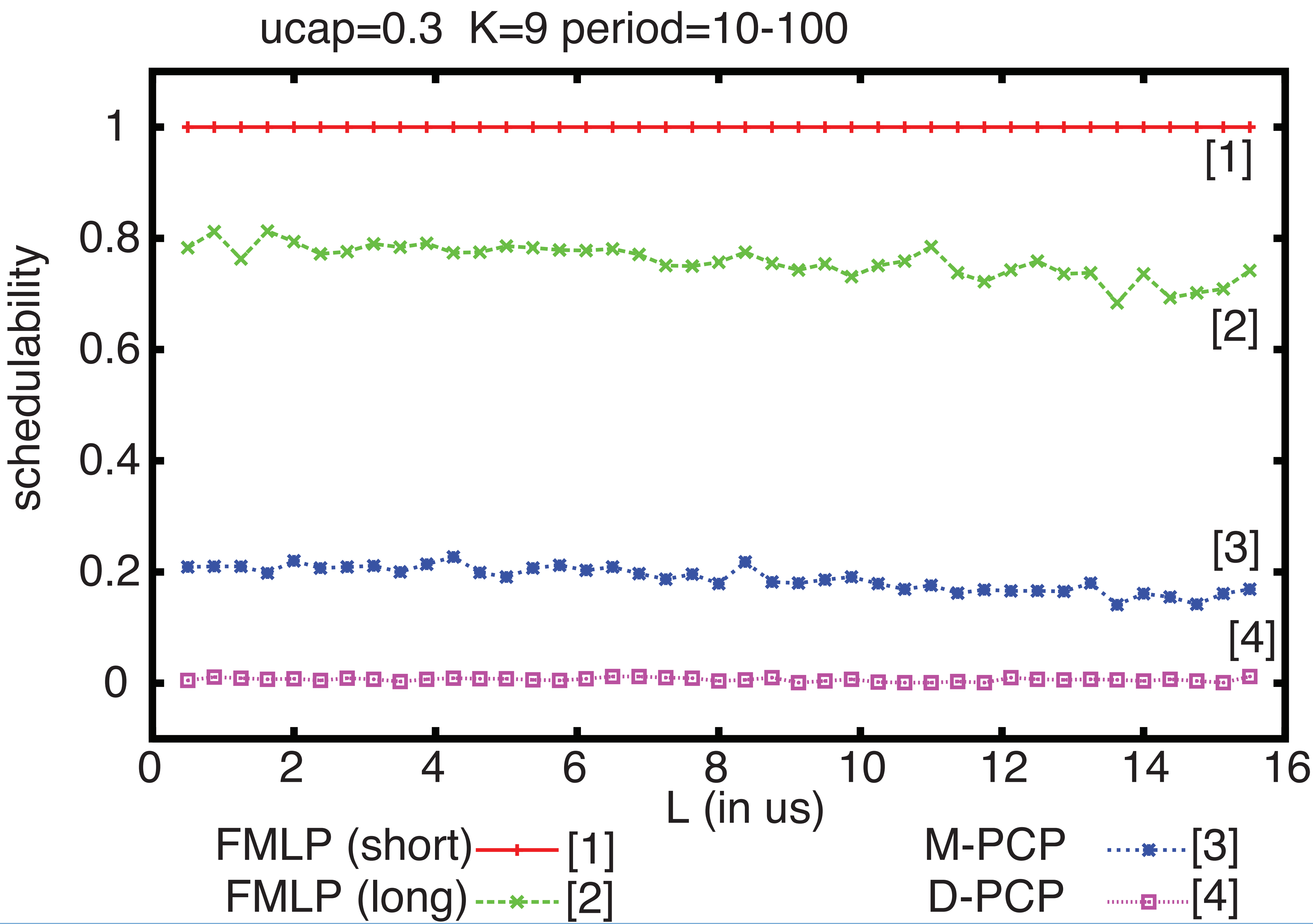
Schedulability vs. Critical Section Length



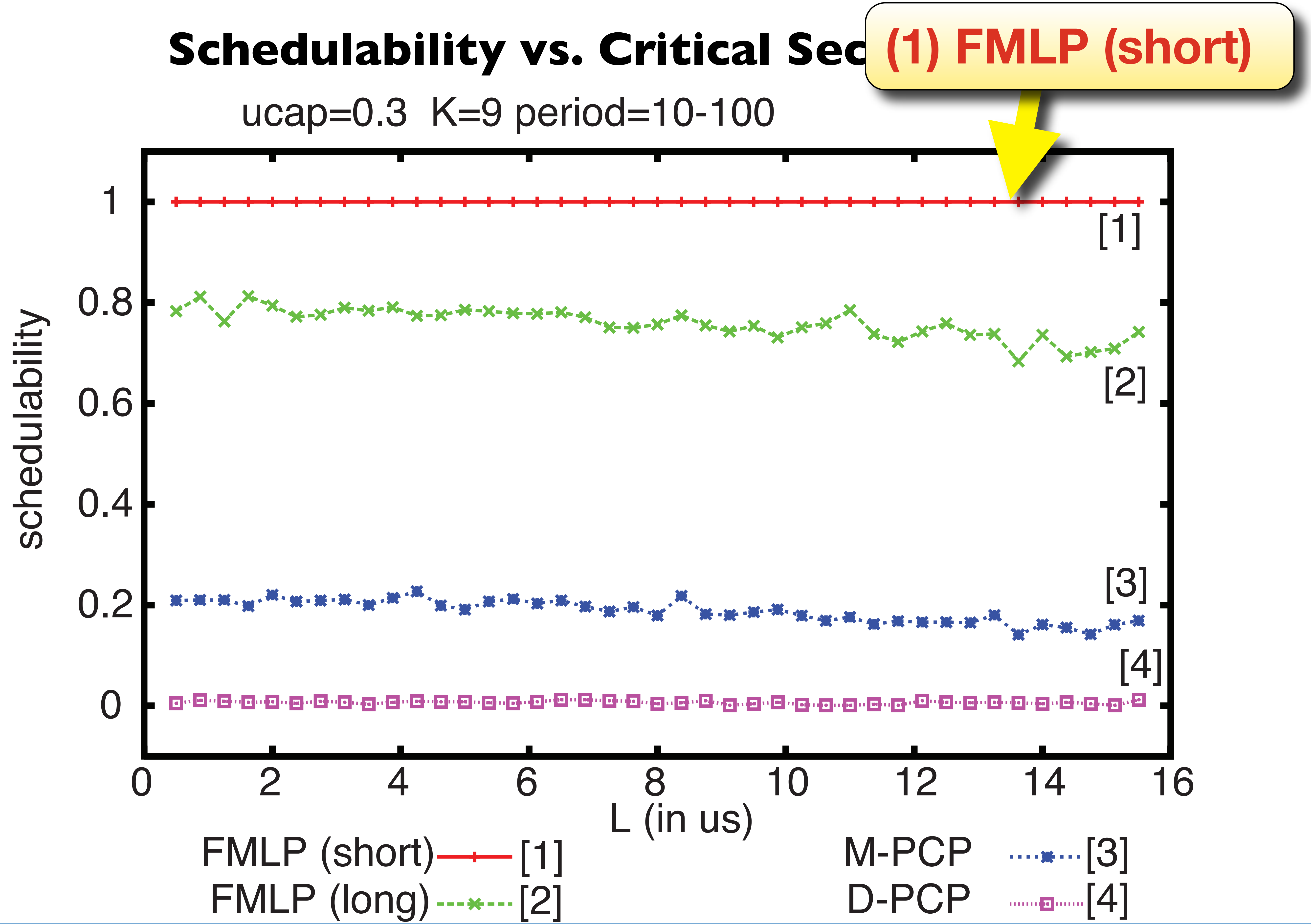
Schedulability vs. Critical Section Length

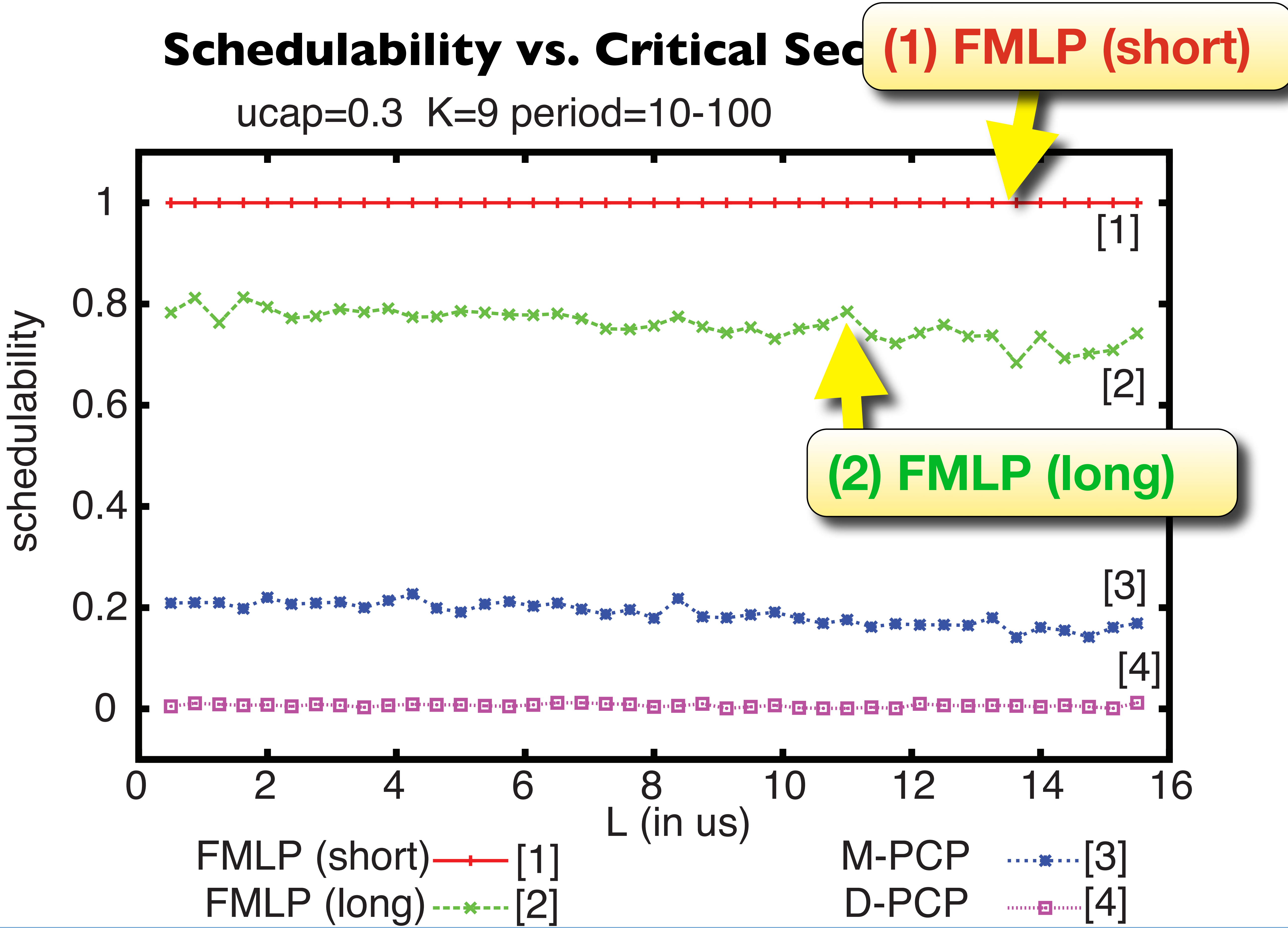


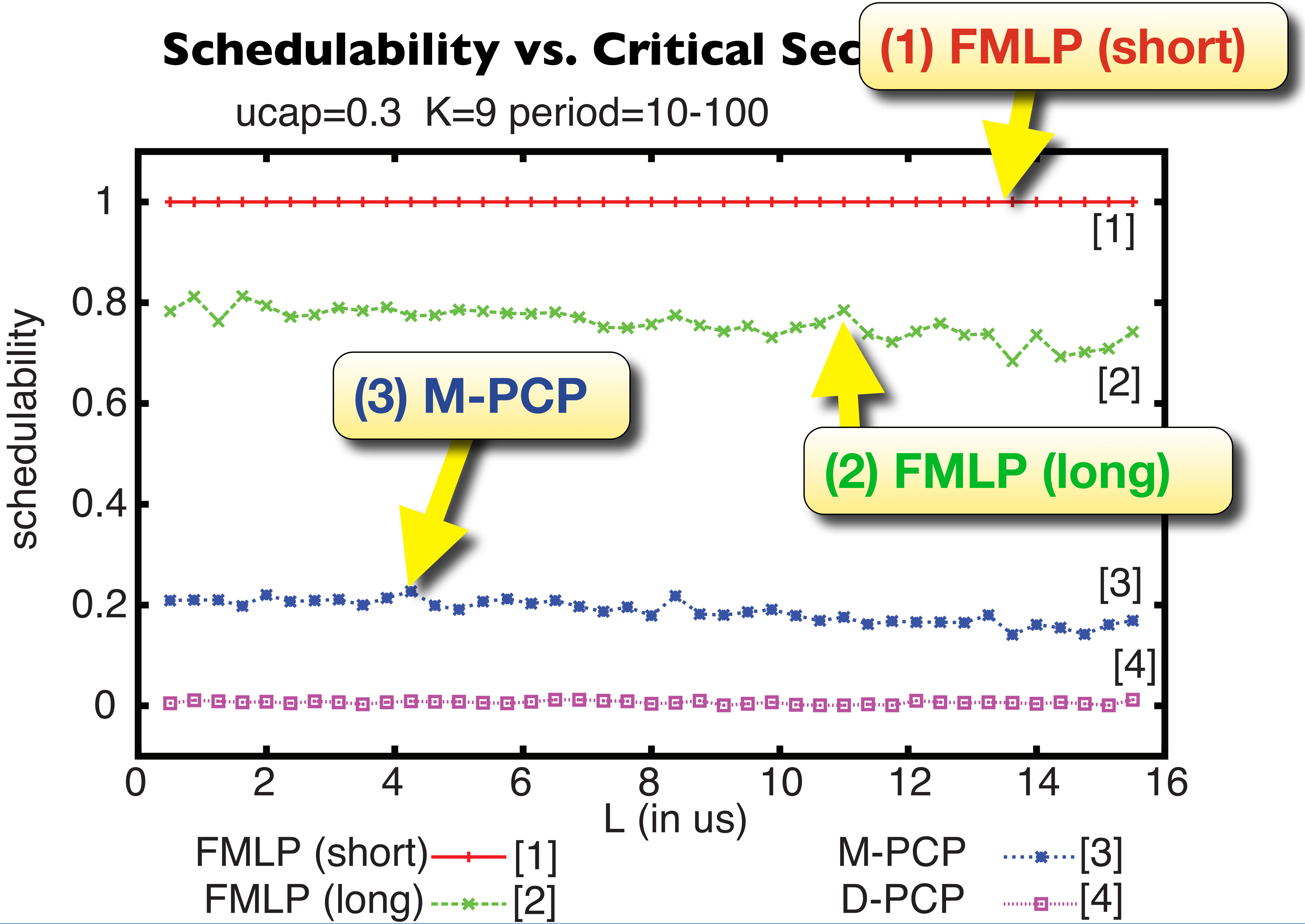
Schedulability vs. Critical Section Length



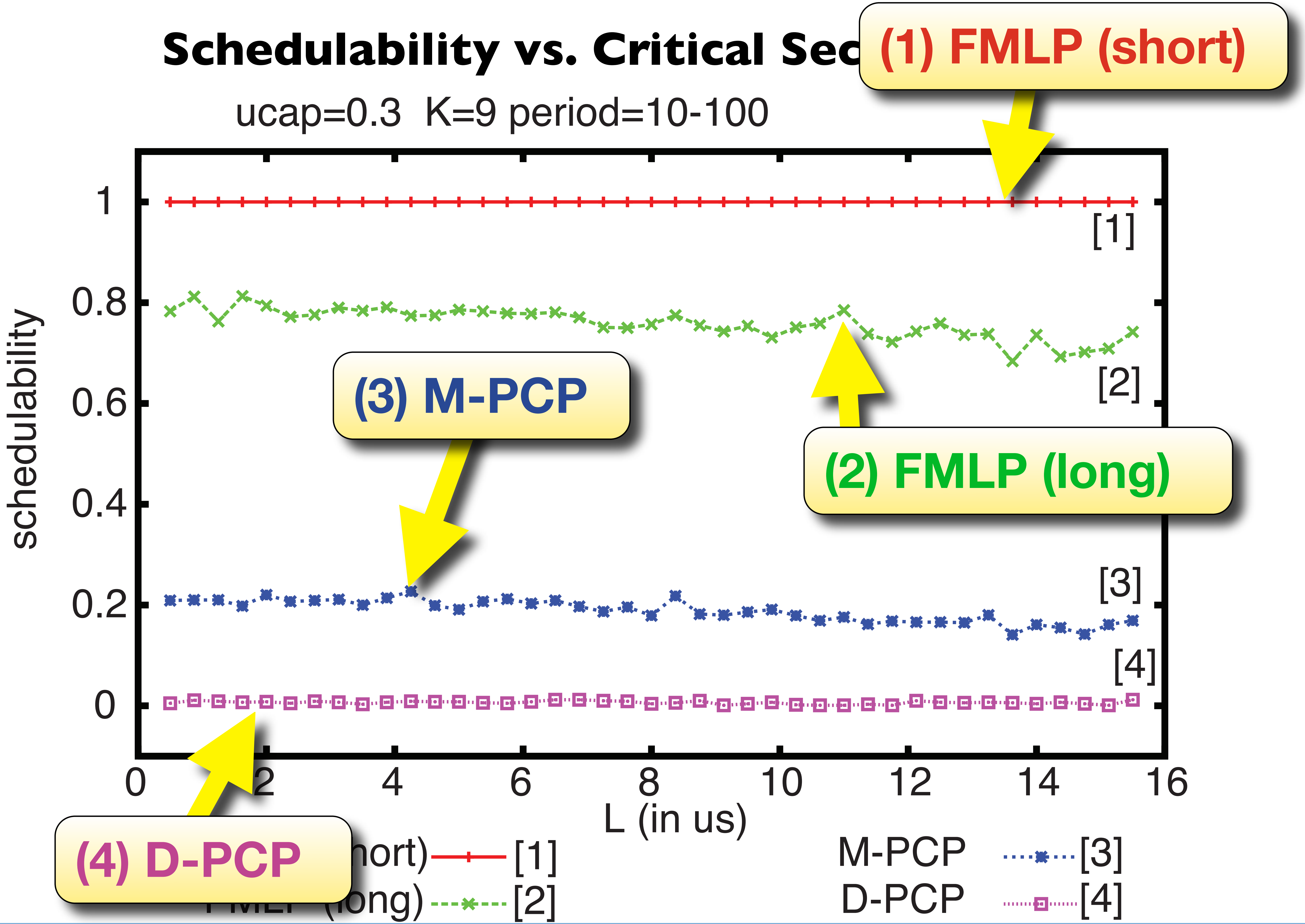
Schedulability vs. Critical Sec



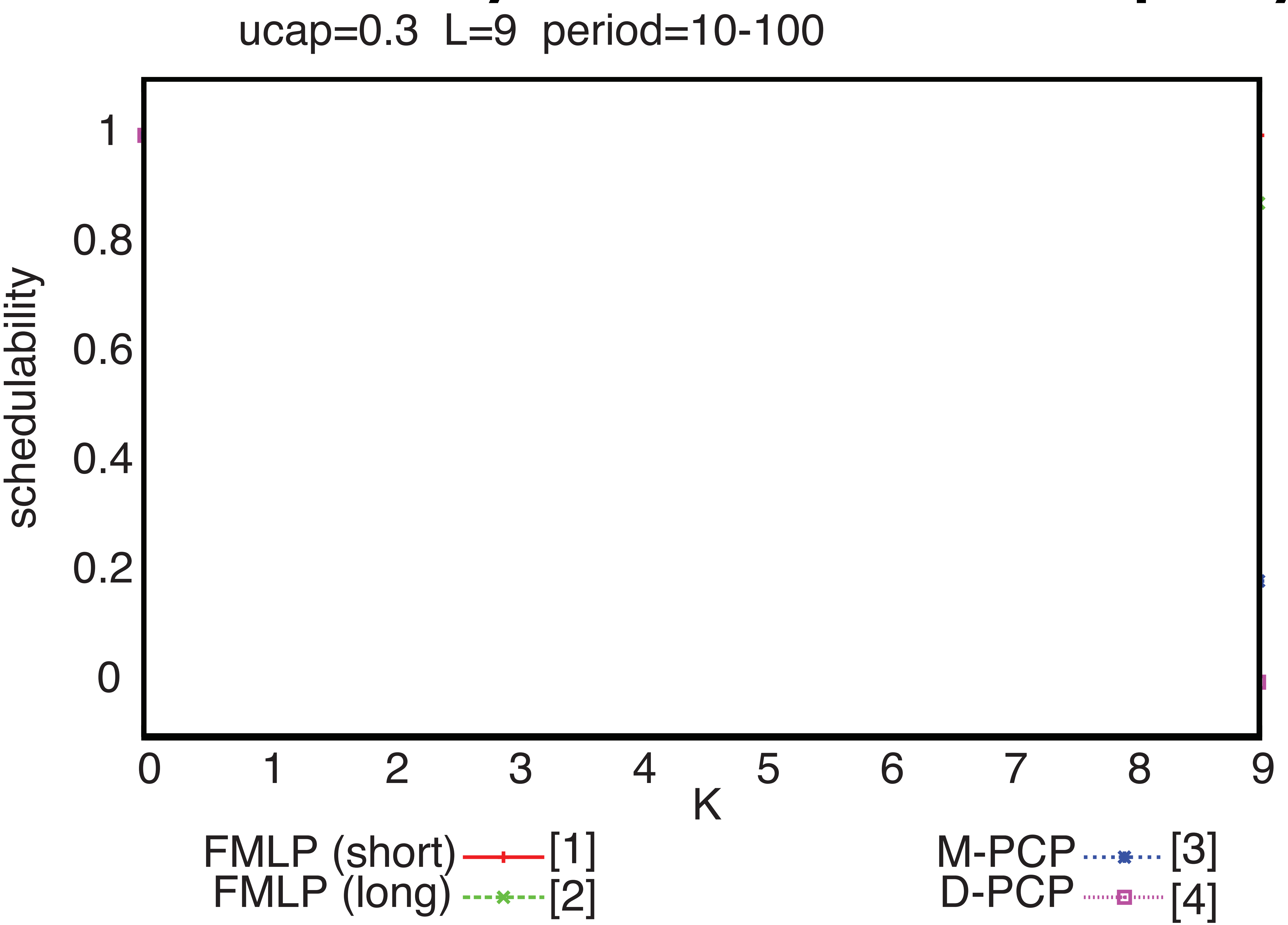

Schedulability vs. Critical Sec
(1) FMLP (short)
ucap=0.3 K=9 period=10-100
schedulability
1
0.8
0.6
0.4
0.2
0
[1]
[2]
[3]
[4]
(2) FMLP (long)
0
2
4
6
8
10
12
14
16
L (in us)
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]



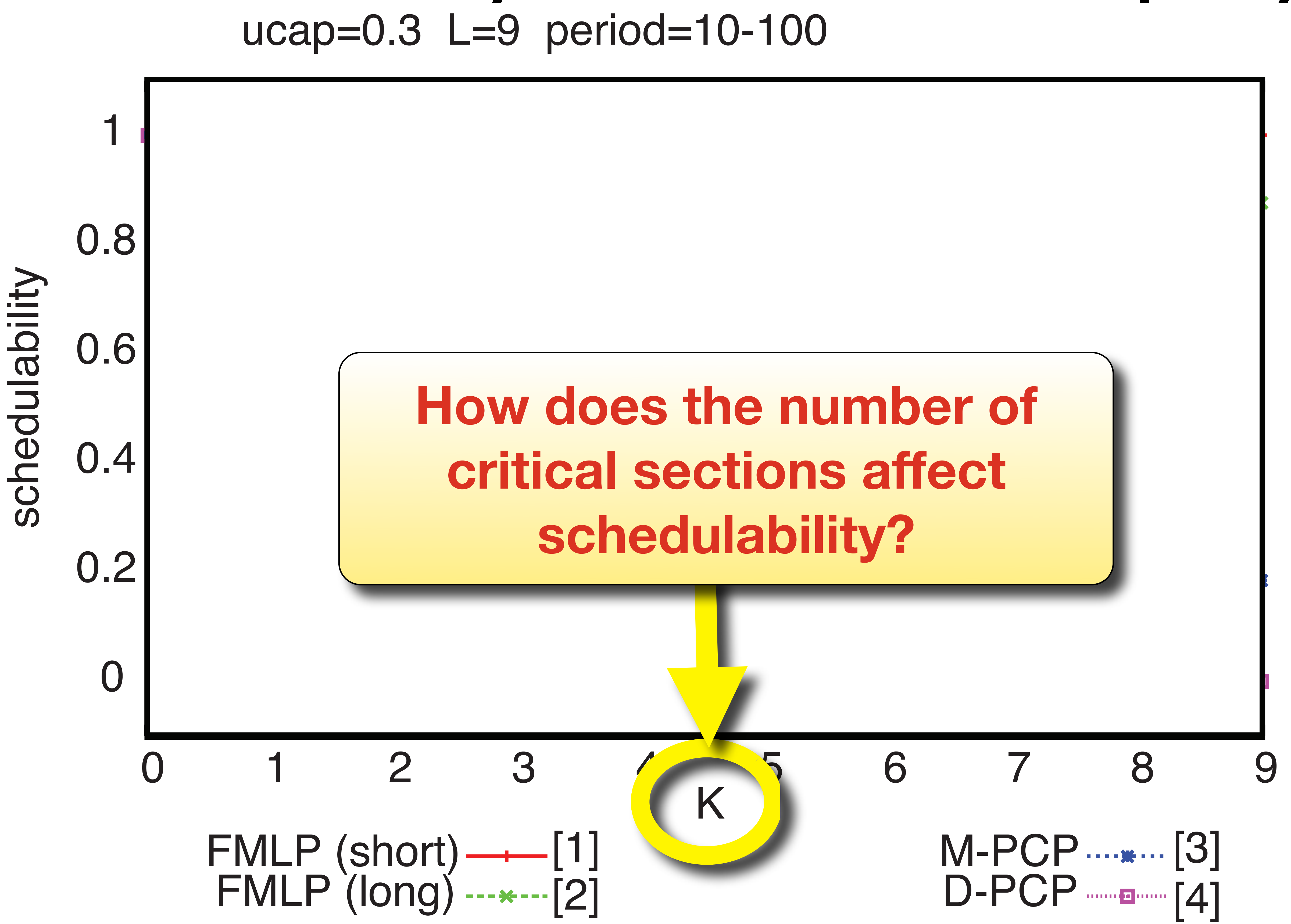
Schedulability vs. Critical Sec
(1) FMLP (short)
ucap=0.3 K=9 period=10-100
(2) FMLP (long)
(3) M-PCP
schedulability
1
0.8
0.6
0.4
0.2
0
[1]
[2]
[3]
[4]
0
2
4
6
8
10
12
14
16
L (in us)
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]


Schedulability vs. Critical Sec
ucap=0.3 K=9 period=10-100
schedulability
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
14
16
L (in us)
[1]
[2]
[3]
[4]
(1) FMLP (short)
(2) FMLP (long)
(3) M-PCP
(4) D-PCP
hort) [1]
(long) [2]
M-PCP [3]
D-PCP [4]

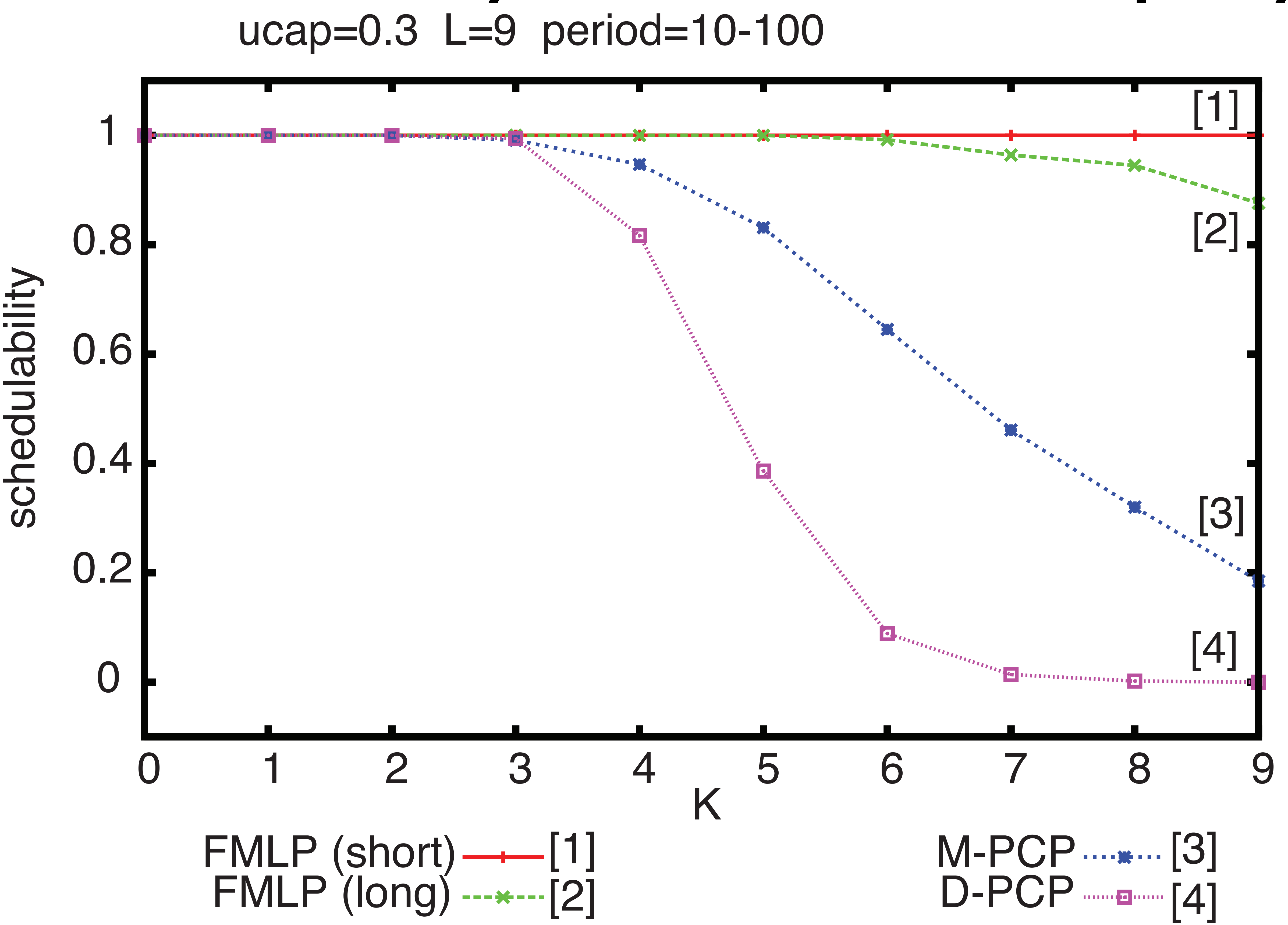
Schedulability vs. Critical Section Frequency

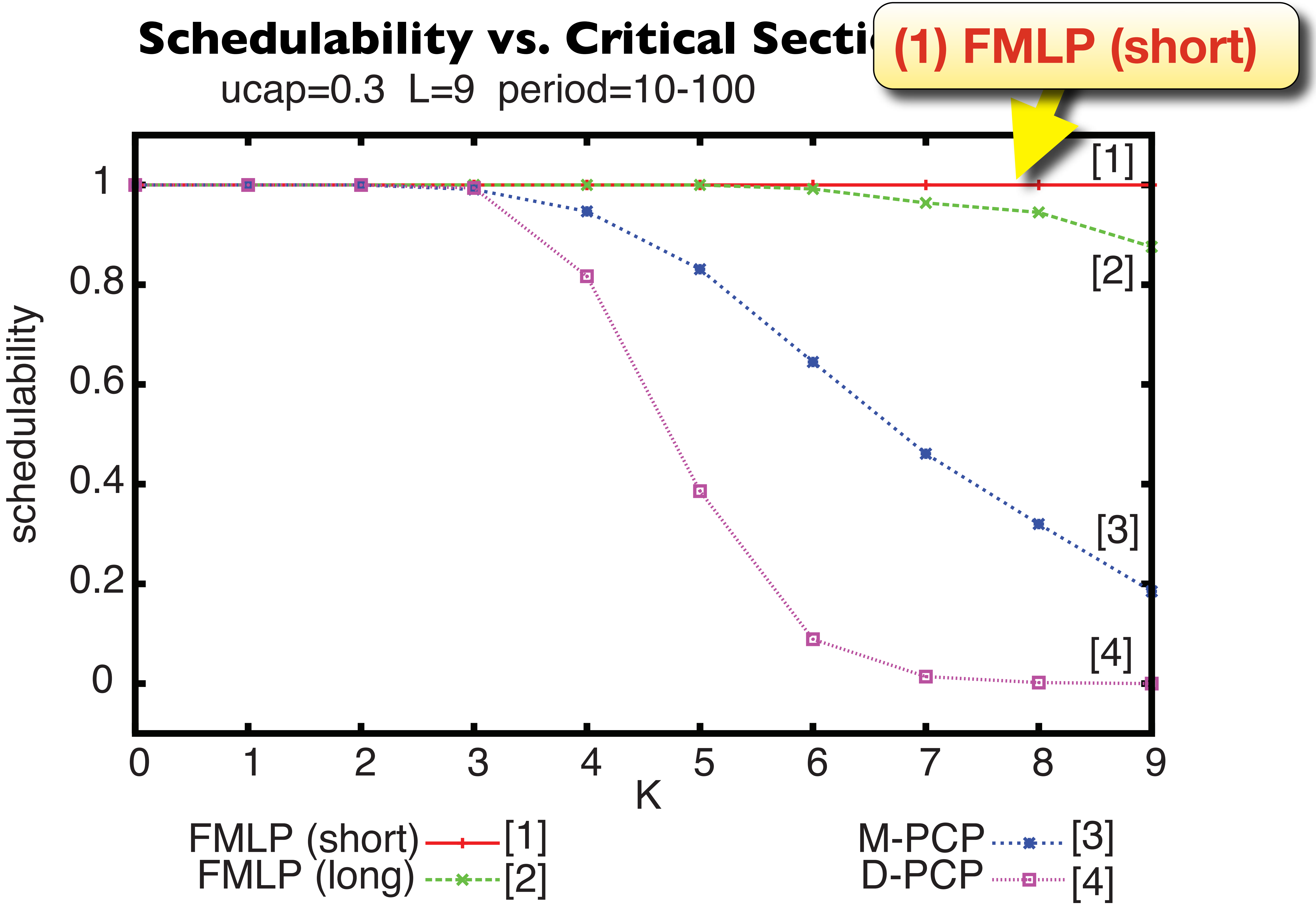


Schedulability vs. Critical Section Frequency

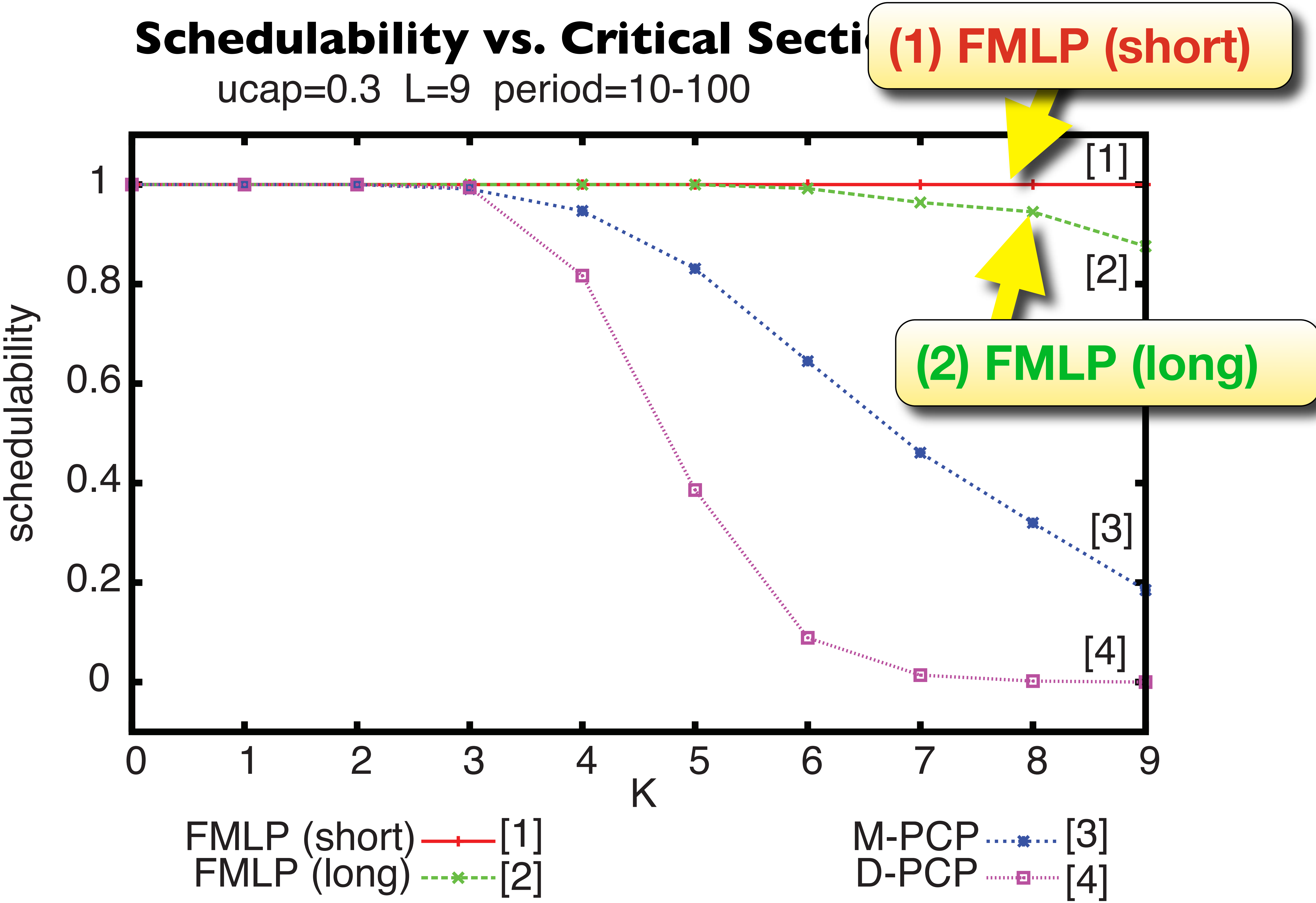


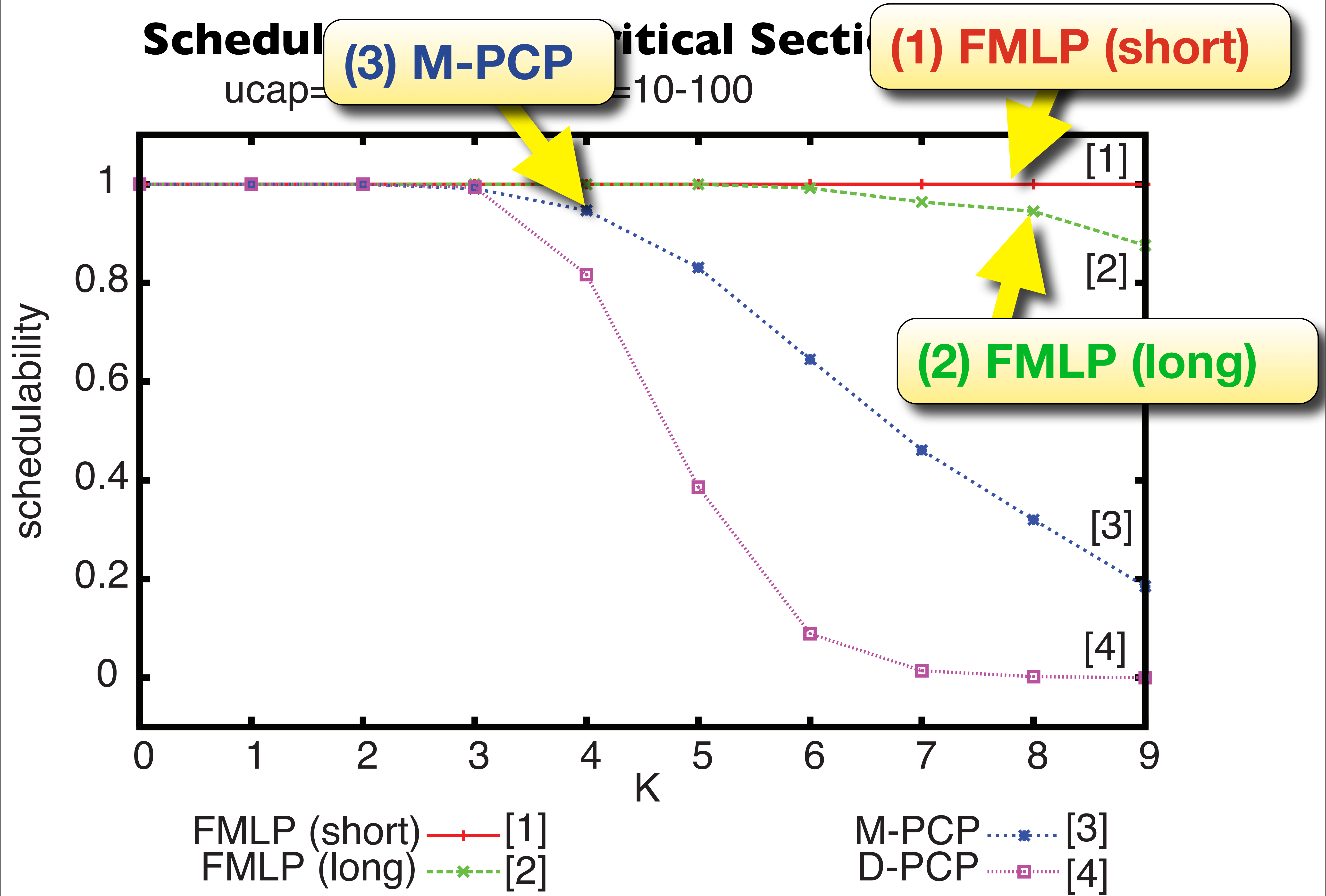
Schedulability vs. Critical Section Frequency

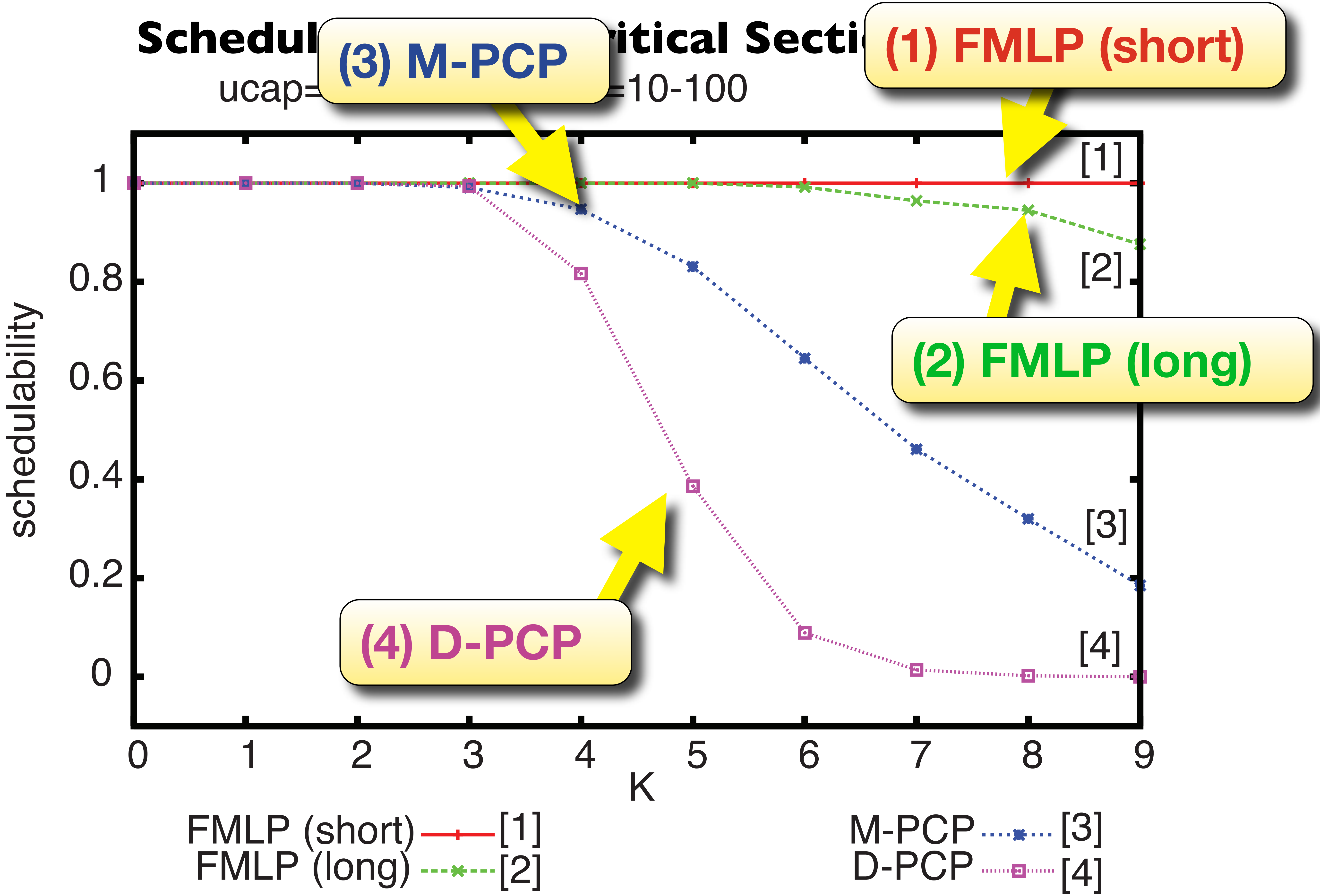




Schedulability vs. Critical Secti
ucap=0.3 L=9 period=10-100
(1) FMLP (short)
schedulability
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9
K
[1]
[2]
[3]
[4]
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]


Schedulability vs. Critical Secti
ucap=0.3 L=9 period=10-100
(1) FMLP (short)
(2) FMLP (long)
schedulability
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8
9
K
[1]
[2]
[3]
[4]
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]


(3) M-PCP
(1) FMLP (short)
(2) FMLP (long)
=10-100
schedulability
K
0
0.2
0.4
0.6
0.8
1
0
1
2
3
4
5
6
7
8
9
[1]
[2]
[3]
[4]
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]


Schedul... ...ritical Secti...
ucap= ... =10-100
(3) M-PCP
(1) FMLP (short)
(2) FMLP (long)
(4) D-PCP
schedulability
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8
9
K
[1]
[2]
[3]
[4]
FMLP (short) [1]
FMLP (long) [2]
M-PCP [3]
D-PCP [4]

FMLP vs. D-PCP & M-PCP

FMLP vs. D-PCP & M-PCP

Non-preemptive FIFO spinlocks are usually the **best synchronization choice** (from a schedulability point of view).

FMLP vs. D-PCP & M-PCP

Non-preemptive FIFO spinlocks are usually the **best synchronization choice** (from a schedulability point of view).

Even with **semaphores**, the **FMLP** usually achieves **higher schedulability**.

FMLP vs. D-PCP & M-PCP

Non-preemptive FIFO spinlocks are usually the **best synchronization choice** (from a schedulability point of view).

Even with **semaphores**, the **FMLP** usually achieves **higher schedulability**.

Simplicity wins

The **FMLP** **outperforms** the "classic" D-PCP and M-PCP most of the time.

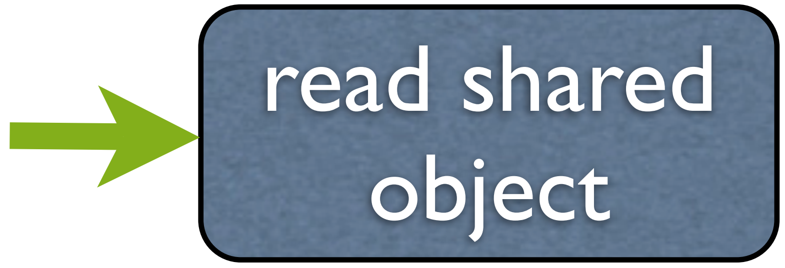
Non-blocking Synchronization

(on Uniprocessors)

Nonblocking Algorithms

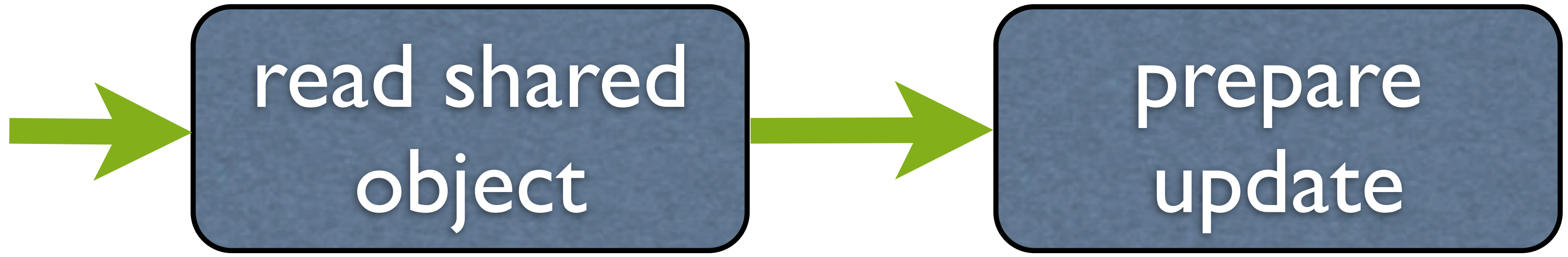
- Two variants:
  - **Lock-free:**
    - Perform operations "optimistically".
    - Retry operations that are interfered with.
  - **Wait-free:**
    - No waiting of any kind:
      - No busy-waiting.
      - No blocking synchronization constructs.
      - No unbounded retries.
- Prior research at UNC has shown how to account for lock-free and wait-free overheads in scheduling analysis.
- First, some background …

Non-Blocking Synchronization: **Lock-Free**



(very high-level view)

Non-Blocking Synchronization: **Lock-Free**



(very high-level view)

Non-Blocking Synchronization: **Lock-Free**

(very high-level view)

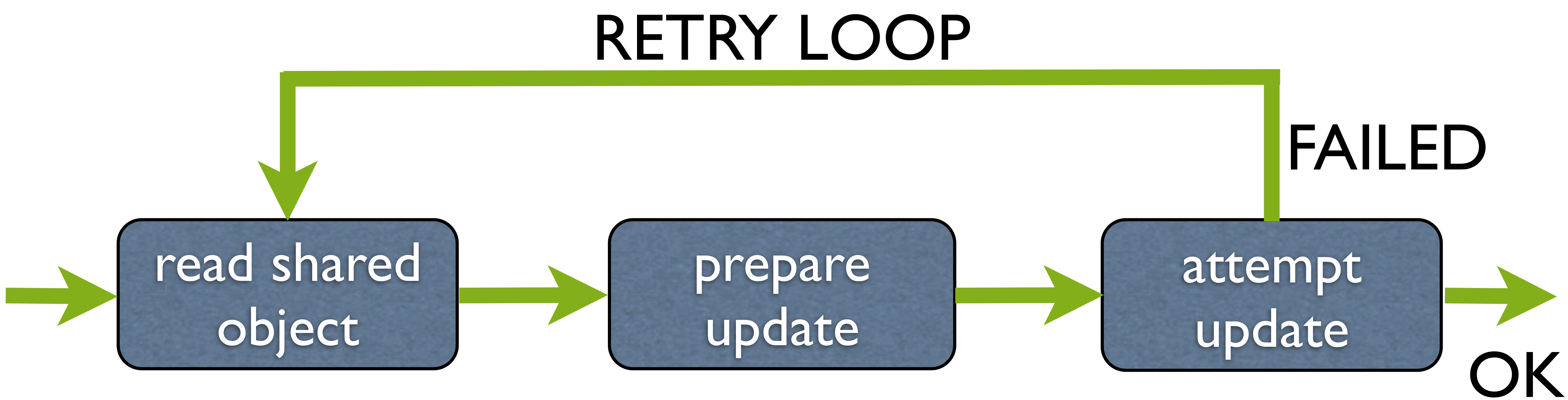
Non-Blocking Synchronization: **Lock-Free**

(very high-level view)

Non-Blocking Synchronization: **Lock-Free**

(very high-level view)

Non-Blocking Synchronization: **Lock-Free**

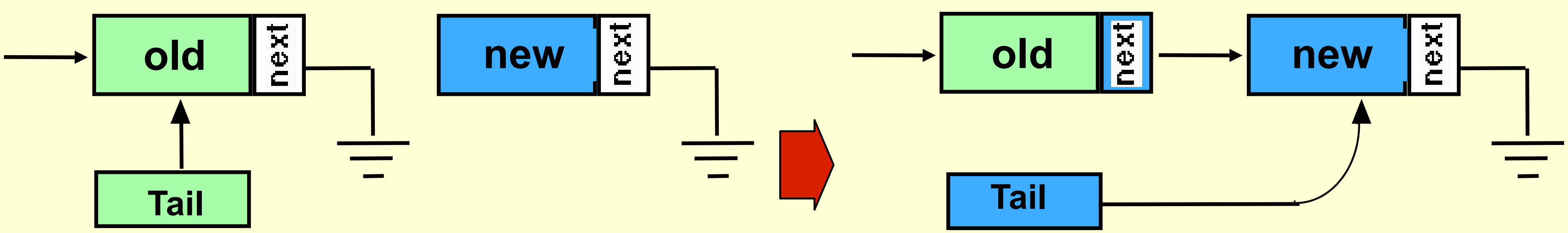


(very high-level view)

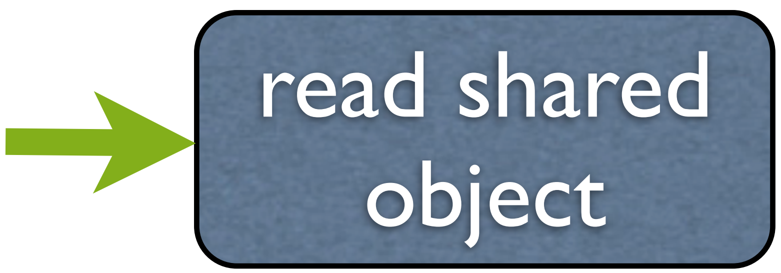
Lock-Free Example

```
type  Qtype = record v: valtype; next:  pointer to Qtype end
shared var  Tail:  pointer to Qtype;
local var  old, new: pointer to Qtype

procedure Enqueue (input: valtype)
     new := (input,  NIL);
    repeat    old := Tail
    until  CAS2(&Tail, &(old->next), old, NIL, new, new)
```

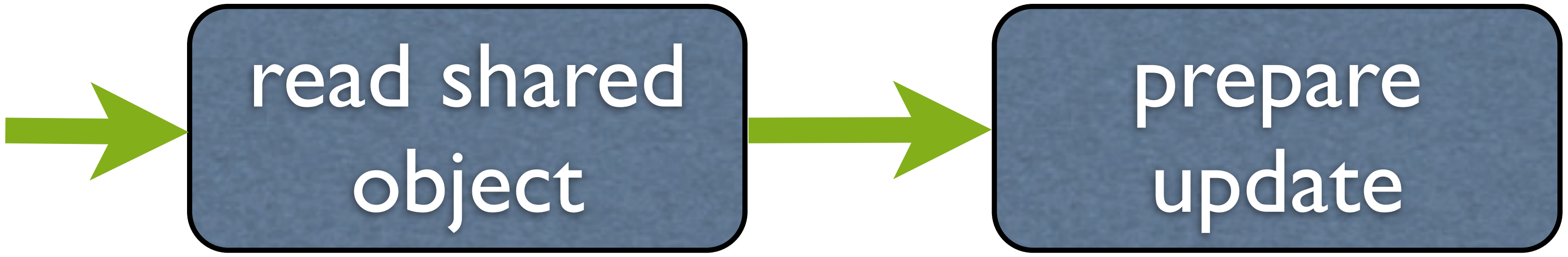


Non-Blocking Synchronization: **Wait-Free**



(very high-level view)

Non-Blocking Synchronization: **Wait-Free**



(very high-level view)

Non-Blocking Synchronization: **Wait-Free**

(very high-level view)

Non-Blocking Synchronization: **Wait-Free**

(very high-level view)

Non-Blocking Synchronization:
Wait-Free

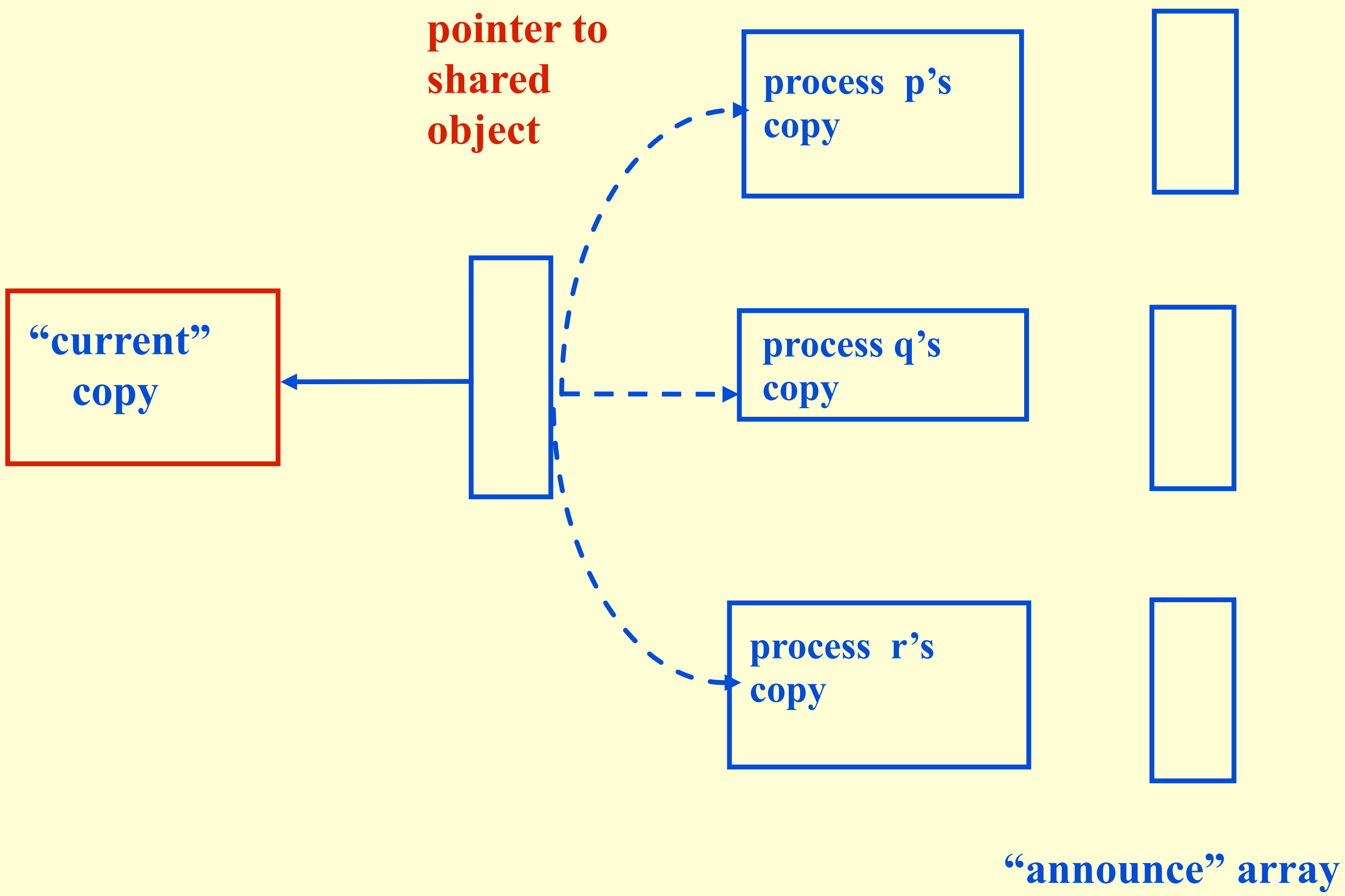
lock-free: **cheap**, but must bound **retry-loops**.

wait-free: **expensive**, but **no retries**, **no blocking**!

(very high-level view)

Wait-Free Algorithms

(Herlihy's Helping Scheme)



Algorithm:

"announce" operation;
retry until done:
create local copy of the object;
apply all announced operations
on local copy;
attempt to make local copy the
"current" copy using a
strong synchronization
primitive

Can only retry once!

Disadvantage: Copying overhead.

Using Wait-Free Algorithms in Real-Time Systems

- On uniprocesors, helping-based algorithms are not very attractive.
  - Only high-priority tasks help lower-priority tasks.
    - Similar to **priority inversion**.
  - Such algorithms can have high overhead due to copying and having to use costly synchronization primitives.
    - Some wait-free algorithms avoid these problems and *are* useful.
    - **Example:** "Collision avoiding" read/write buffers.
- On the other hand, on multiprocessors, wait-free algorithms may be the best choice.

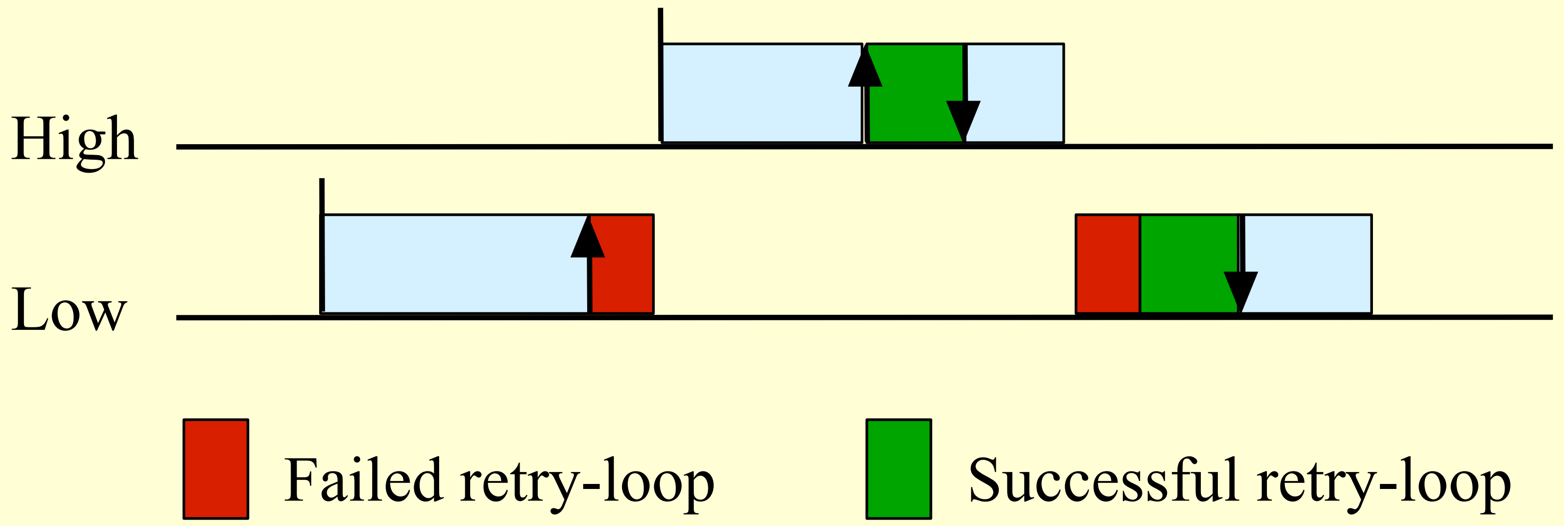
Using Lock-Free Objects on Real-Time Uniprocessors

- **Advantages of Lock-free Objects:**
  - No priority inversions.
  - Lower overhead than helping-based wait-free objects.
  - Overhead is charged to low-priority tasks.
- **But:**
  - Access times are **potentially unbounded**.

Scheduling with Lock-Free Objects

On a uniprocessor, lock-free retries really aren't unbounded.

A task fails to update a shared object only if **preempted** during its object call.



Can compute a bound on retries by counting preemptions.

Lock-Free on Multiprocessors

- same basic approach:

  bound worst-case number of retries

- but:
  - partitioning: tasks of **all priorities** on other CPUS can interfere
  - global: **all tasks** can interfere

(see Uma's thesis for an overview and references)

RTAS'08:

Spinning vs. Suspending vs. Lock-Free vs. Wait-Free

- FMLP under G-EDF and P-EDF
- Lock-Free and Wait-Free in userspace
- Implemented in LITMUS^{RT}
- Obtained various overheads and retry-loop costs for several data structures.

Real-Time Synchronization

Real-Time Synchronization

Blocking

Non-Blocking

Real-Time Synchronization

Blocking **Non-Blocking**

Suspend **Spin** **Lock-Free** **Wait-Free**

Real-Time Synchronization

Blocking **Non-Blocking**

Suspend **Spin** **Lock-Free** **Wait-Free**

Which performs best in terms of schedulability?

Spinning vs. Suspending

(under G-EDF and P-EDF)

B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?", *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 342-353, April 2008.

Spinning vs. Suspending

(under G-EDF and P-EDF)

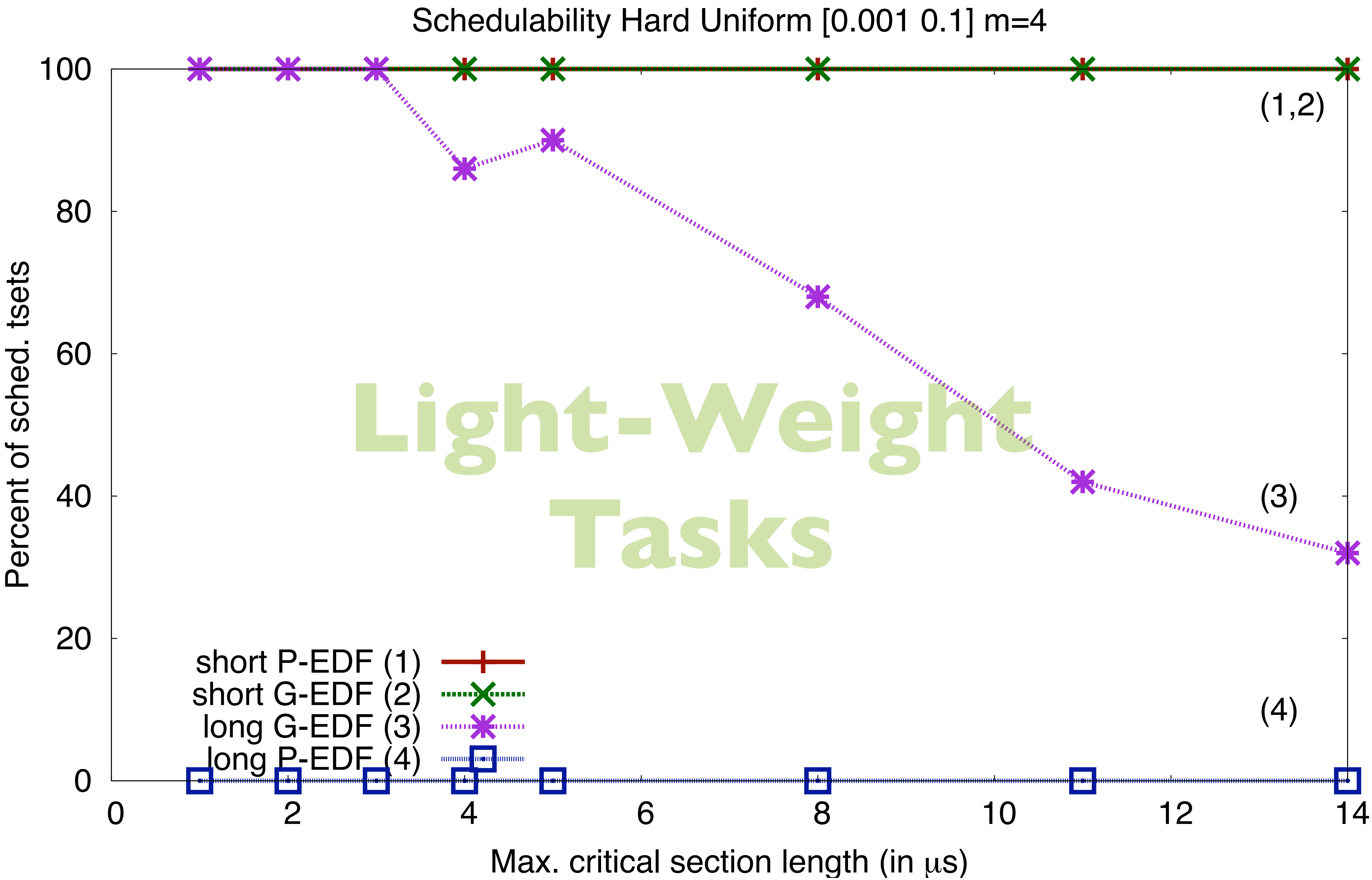
Question:

When, if ever, is suspending preferable to spinning?

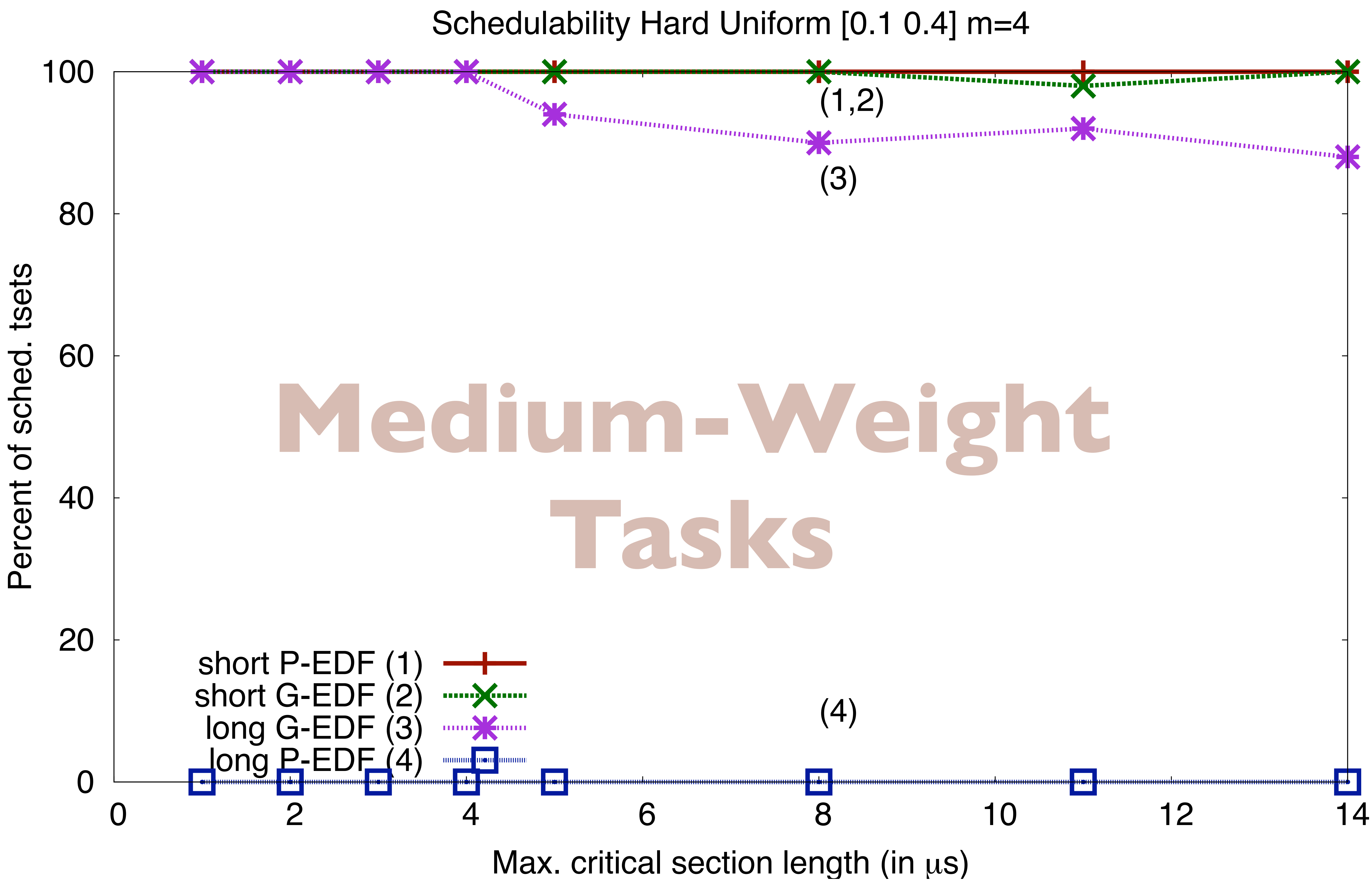
(from the point of view of ***schedulability****)*

B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?", *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 342-353, April 2008.

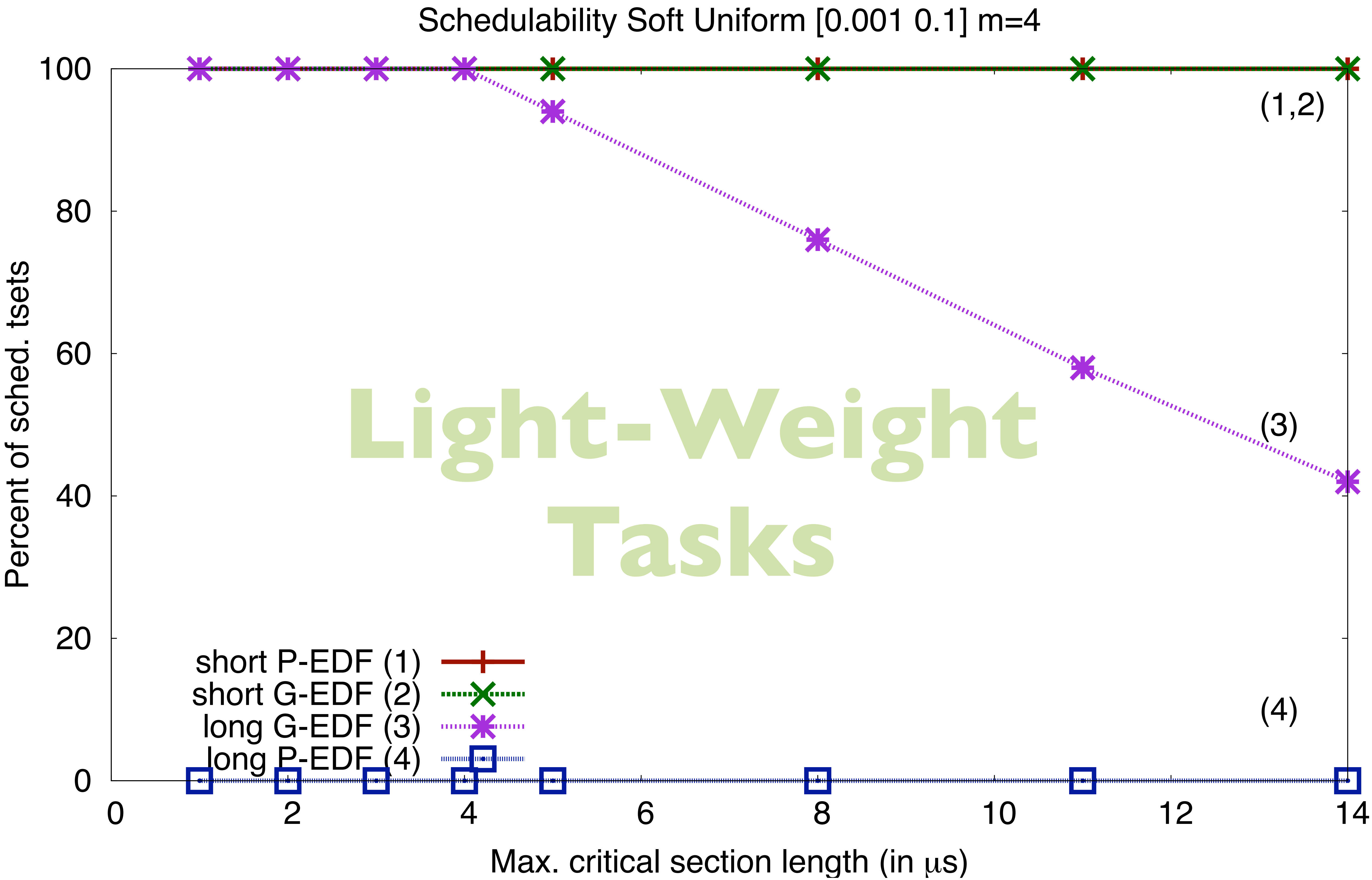
Spinning vs. Suspending: **Hard Real-Time**



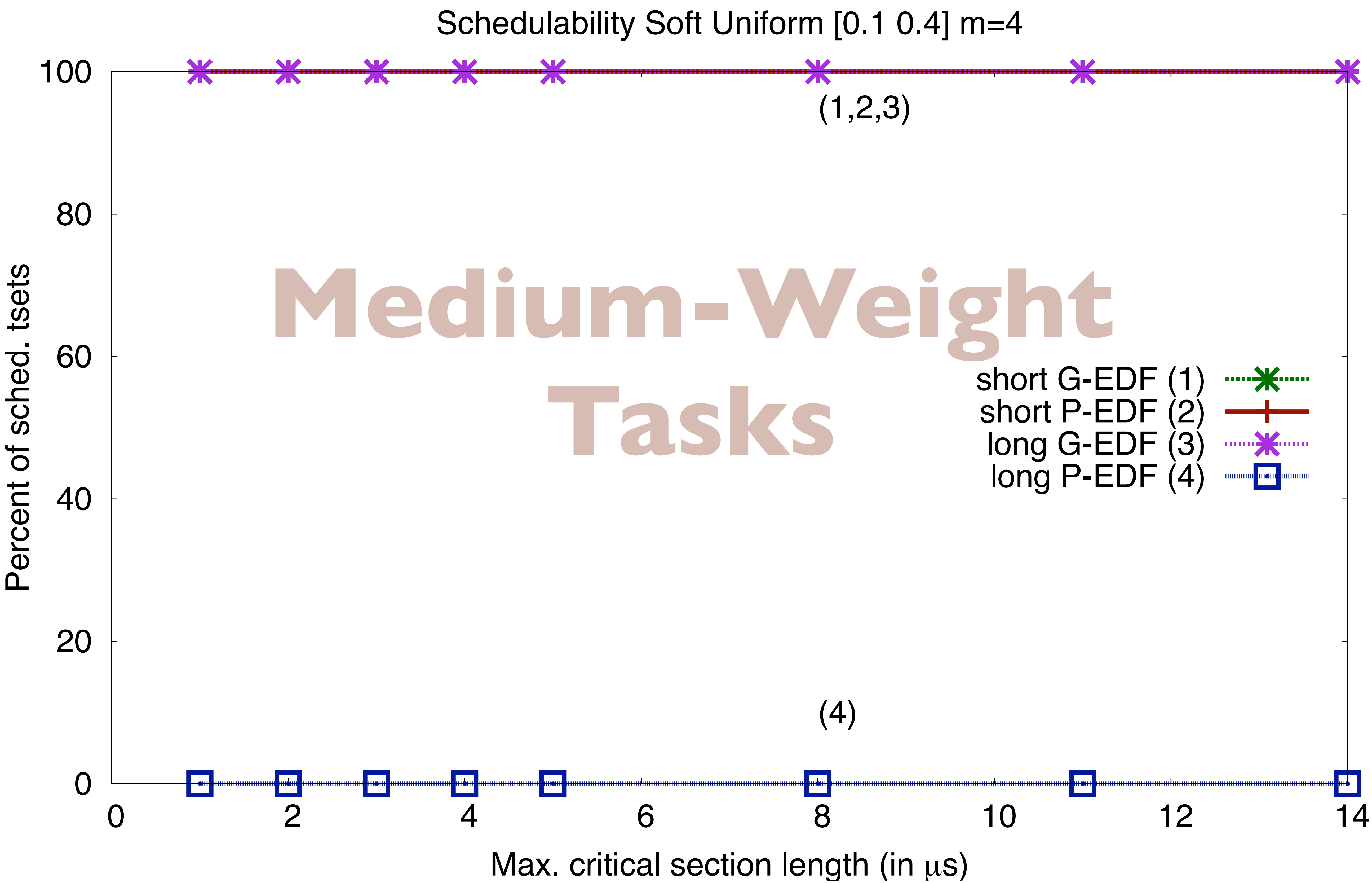
Spinning vs. Suspending: **Hard Real-Time**



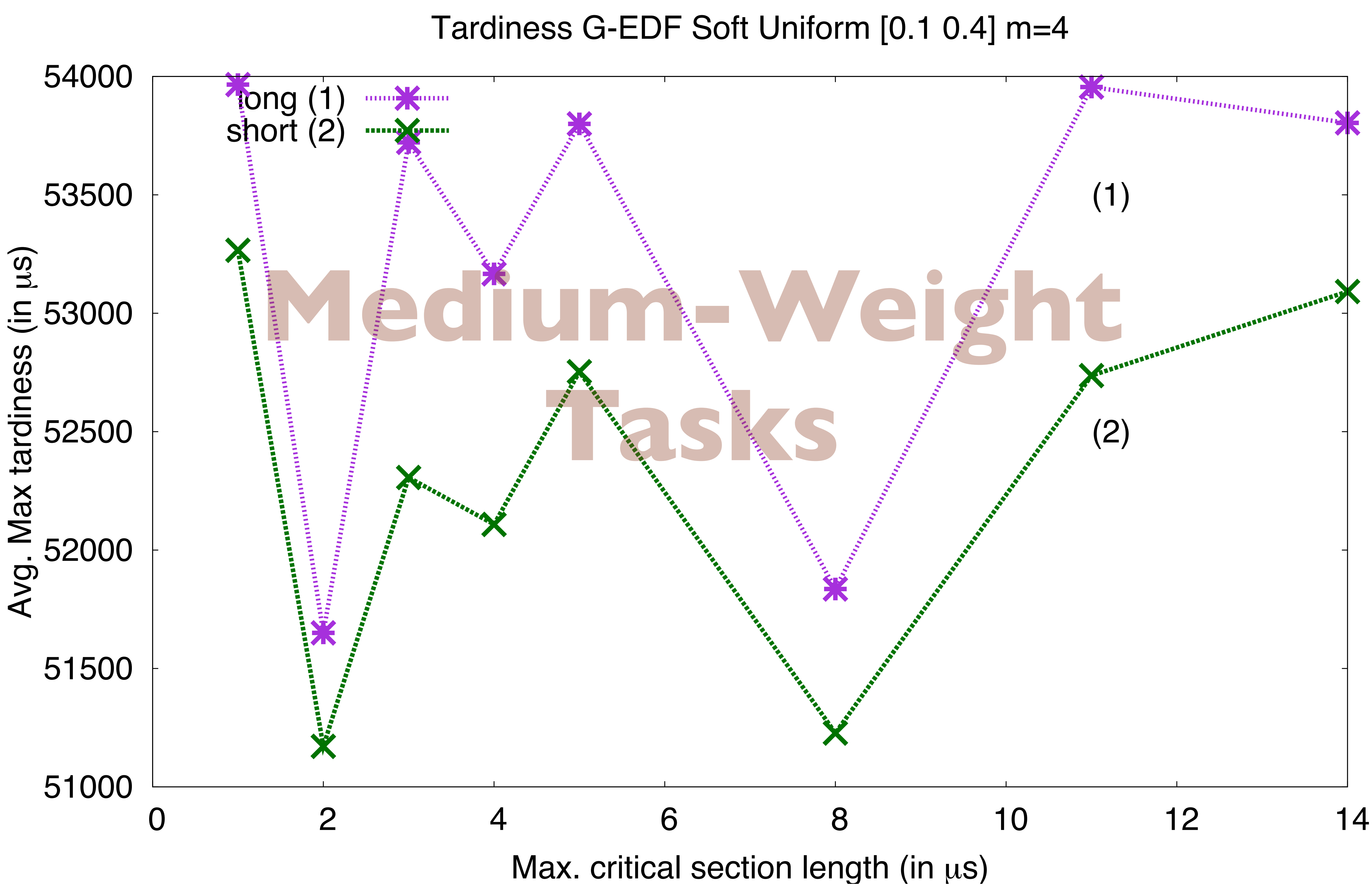
Spinning vs. Suspending: **Soft Real-Time**



Spinning vs. Suspending: **Soft Real-Time**



Spinning vs. Suspending: **Soft Real-Time**



Spinning vs. Suspending

(under G-EDF and P-EDF)

	P-EDF	***G-EDF***
***Spinning* (short)**	Good	Good
***Suspending* (long)**	Generally extremely poor	Only for moderate task counts; tardiness is higher

B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?", *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 342-353, April 2008.

Why is suspending so much worse?

Why is suspending so much worse?

suspension

=

We don't know what happened while the job was gone.

Why is suspending so much worse?

suspension

=

We don't know what happened while the job was gone.

Maybe competing requests?

Why is suspending so much worse?

suspension

=

We don't know what happened while the job was gone.

Maybe competing requests?

Maybe non-preemptive section?

Why is suspending so much worse?

suspension

=

We don't know what happened while the job was gone.

Maybe competing requests?

Maybe non-preemptive section?

Maybe preemption/ migration?

Why is suspending so much worse?

suspension

=

We don't know what happened while the job was gone.

Maybe competing requests?

Maybe non-preemptive section?

Maybe preemption/ migration?

pessimistic analysis

What if we had **better analysis**?

Would suspending become competitive?

What if we had **better analysis**?

Would suspending become competitive?

Well, **we don't know**.

But: This also depends on how "bad" spinning is.

What if we had **better analysis**?

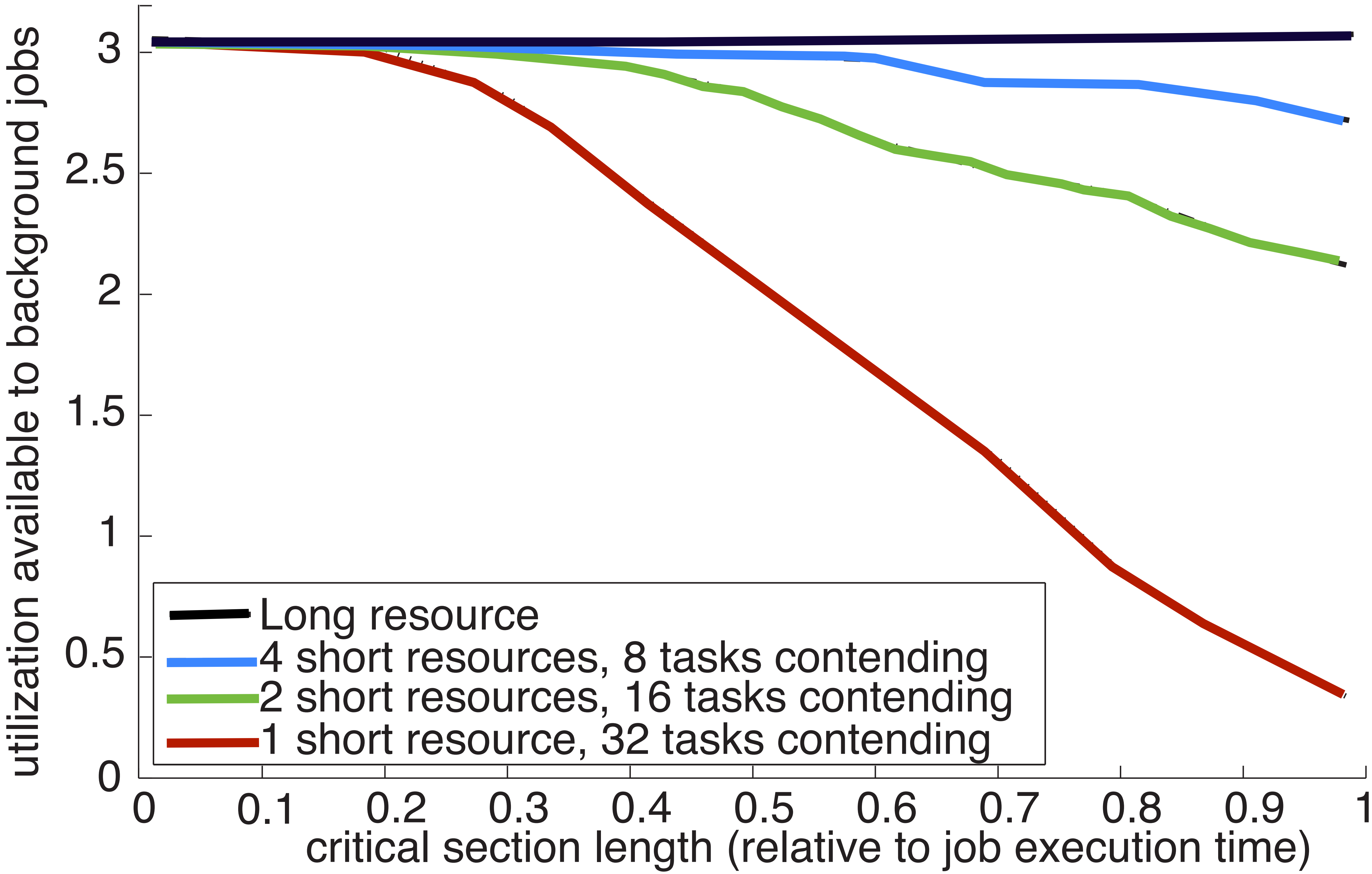
Would suspending become competitive?

Well, **we don't know**.

But: This also depends on how "bad" spinning is.

Experiment:
Measure utilization lost to spinning.

Utilization Loss due to Spinning



So, if we had much **better analysis**…

(conjecture based on empirical evidence)

So, if we had much **better analysis**…

(conjecture based on empirical evidence)

…suspending *might win* if

there is significant contention,

and

the system ***as a whole* spends about**

60% of its time in critical sections.

Spinning vs. Lock-Free vs. Wait-Free

(under G-EDF and P-EDF)

Question:

Are lock-free and wait-free algorithms viable?

If so, when are they preferable to spinning (if ever)?

(from the point of view of ***schedulability****)*

B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?", *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 342-353, April 2008.

Blocking vs. Non-Blocking

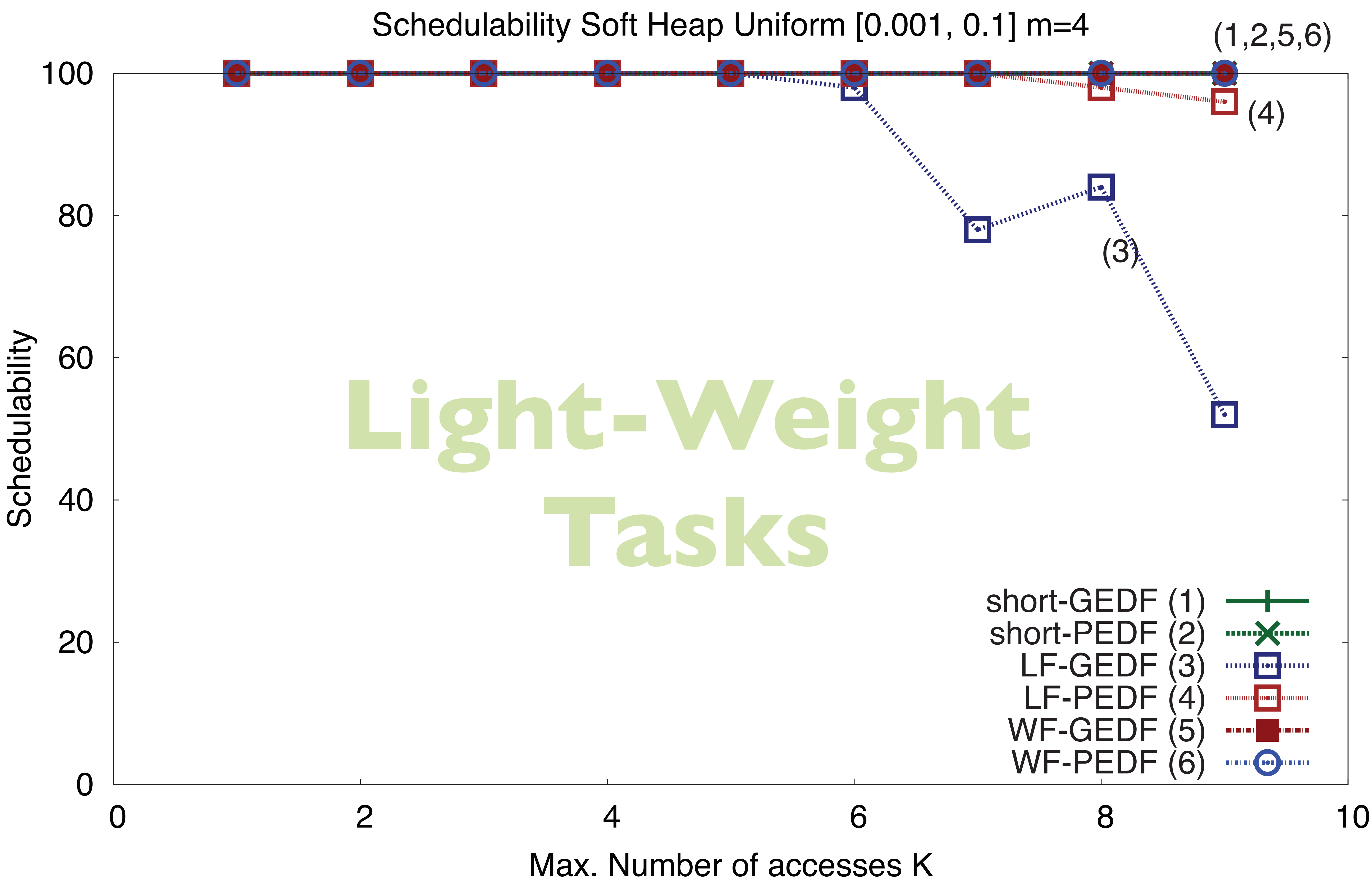
Three Approaches – Three Algorithms

Blocking vs. Non-Blocking

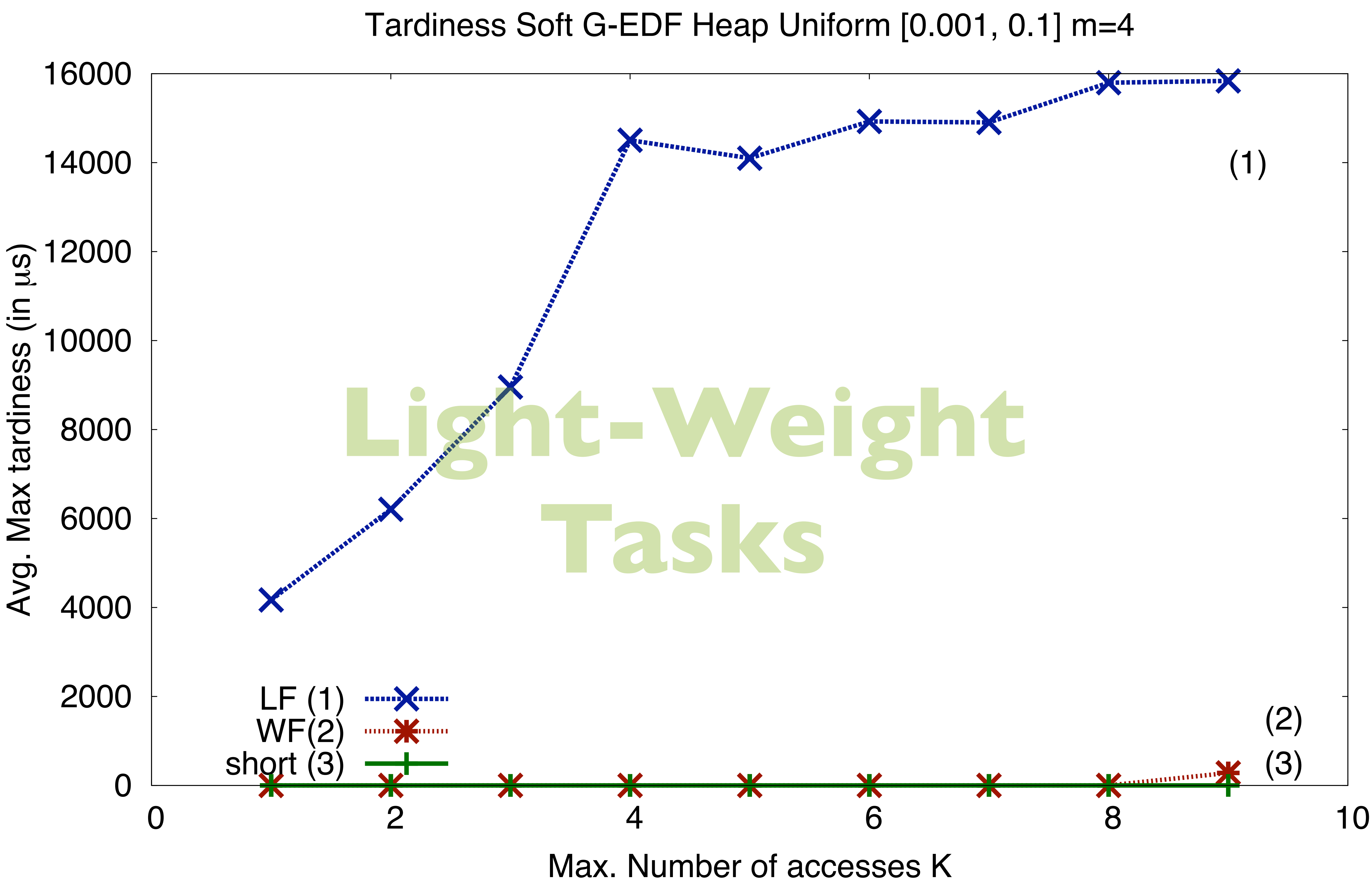
Three Approaches – Three Algorithms

	Buffer	Queue	Heap
Lock-Based	array-based queue-lock [T. Anderson 90]		
Lock-Free	[Tsigas et al. 99]	[Michael et al. 96]	[Anderson and Moir 99]
Wait-Free	[Anderson and Holman 00]	[Anderson and Moir 99]	

Blocking vs. Non-Blocking: **Soft Real-Time**



Blocking vs. Non-Blocking: **Soft Real-Time**



Spinning vs. Lock-Free vs. Wait-Free

(under G-EDF and P-EDF)

	Buffer	**Queue**	**Heap**
Spin-Based	Good, but outperformed by special-purpose algorithms	Good	Good (no copy overhead)
Lock-Free	Good	Good	Retry bounds too pessimistic
Wait-Free	Good	Good	Good (for tested sizes)

B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?", *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 342-353, April 2008.