On the Implementation of Global Real-Time Schedulers

RTSS'09, Washington, DC
December 3, 2009

Björn B. Brandenburg,
and James H. Anderson

The University of North Carolina at Chapel Hill

Work supported by IBM, SUN, and Intel Corps., NSF grants CNS 0834270, CNS 0834132, and CNS 0615197, and ARO grant W911NF-06-1-0425.

UNC's Implementation Studies (I)

Calandrino et al. (2006)

➡ Are commonly-studied RT schedulers **implementable**?

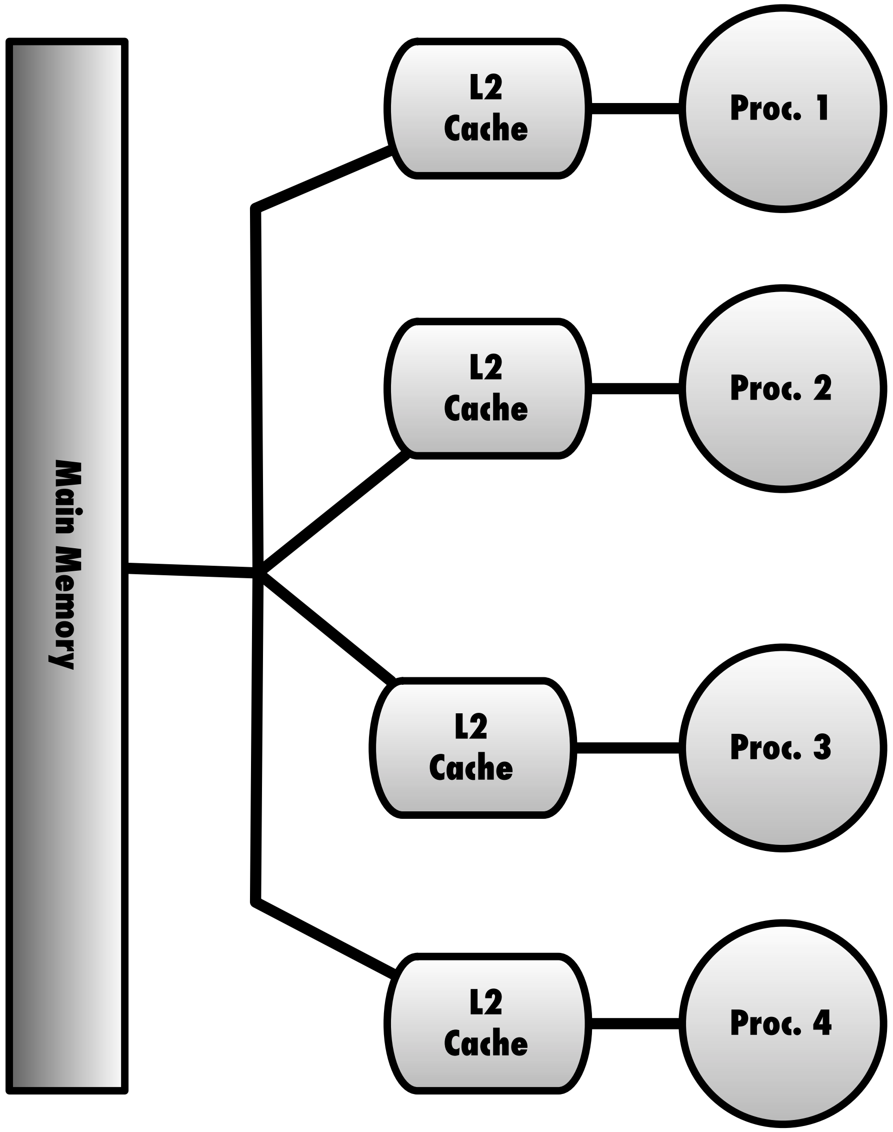
➡ In Linux on common hardware platforms?

Calandrino et al. (2006), LITMUS^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

UNC's Implementation Studies (I)

Calandrino et al. (2006)

- ➡ Are commonly-studied RT schedulers **implementable**?
- ➡ In Linux on common hardware platforms?



Intel 4x 2.7 GHz Xeon SMP

(few, fast processors; private caches)

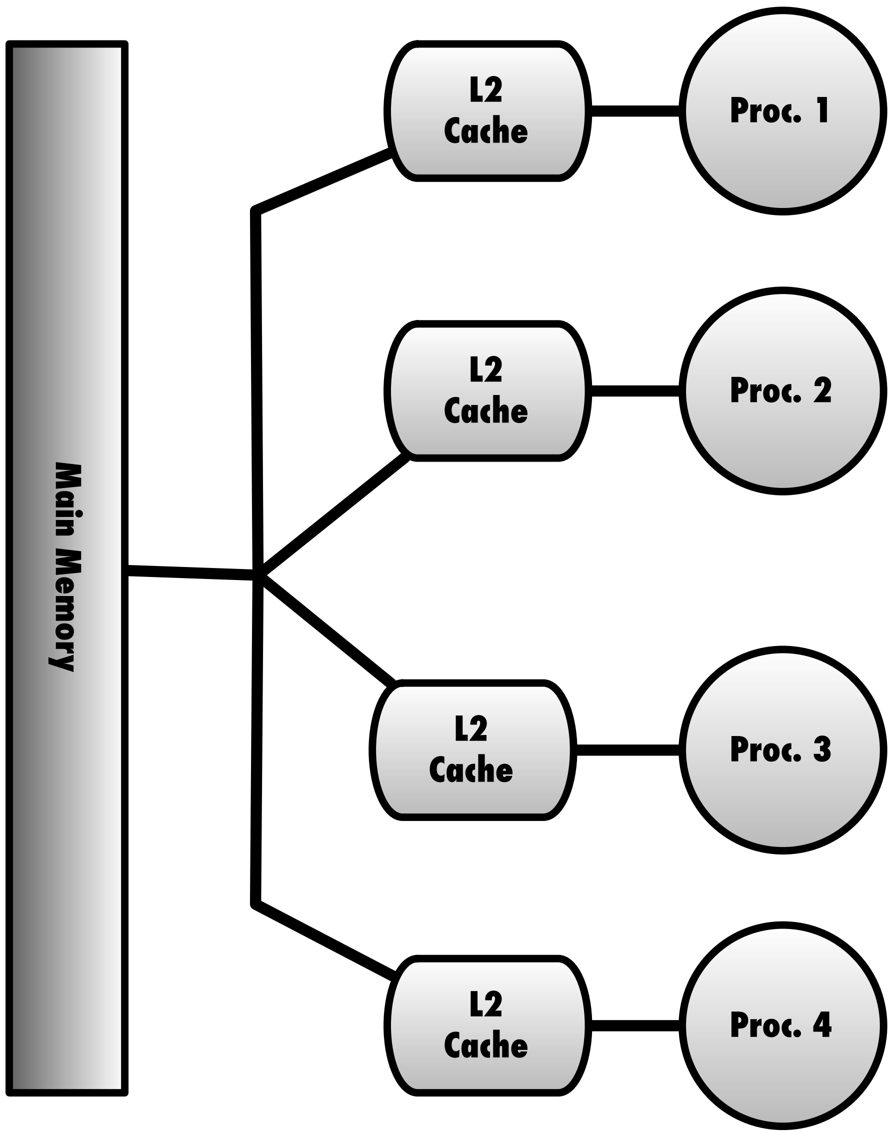
Calandrino et al. (2006), LITMUS[RT]: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

UNC's Implementation Studies (I)

Calandrino et al. (2006)

➡ Are commonly-studied RT schedulers **implementable**?

➡ In Linux on common hardware platforms?



partitioned EDF — **P-EDF**

2 x global EDF — **G-NP-EDF**, **G-EDF**

2 x PFAIR — **PD²**, **S-PD²**

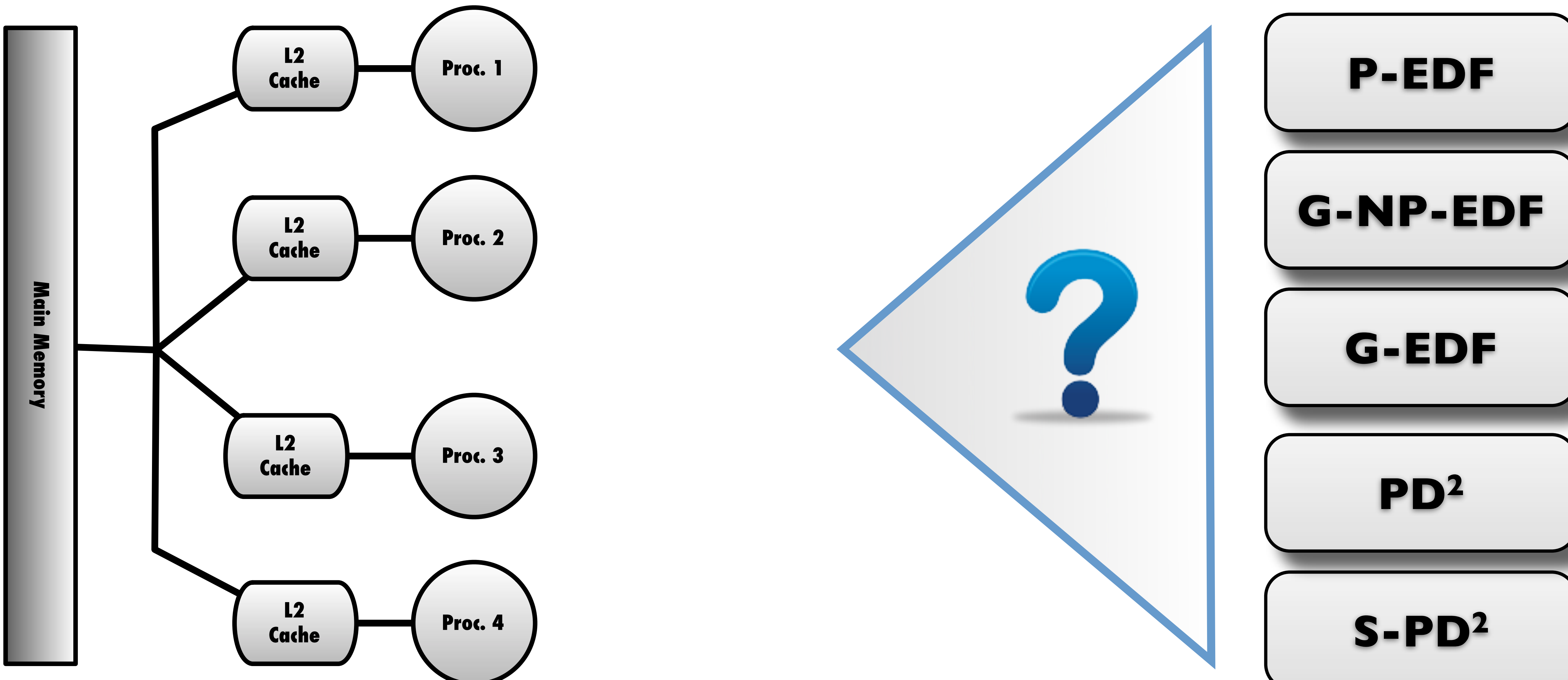
Calandrino et al. (2006), LITMUS^RT: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

UNC's Implementation Studies (I)

"for each tested scheme, scenarios exist in which it is a viable choice"

Calandrino

- ➡ Are commo
- ➡ In Linux on common hardware platforms?



Calandrino et al. (2006), LITMUS^RT: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

UNC's Implementation Studies (II)

Brandenburg et al. (2008)

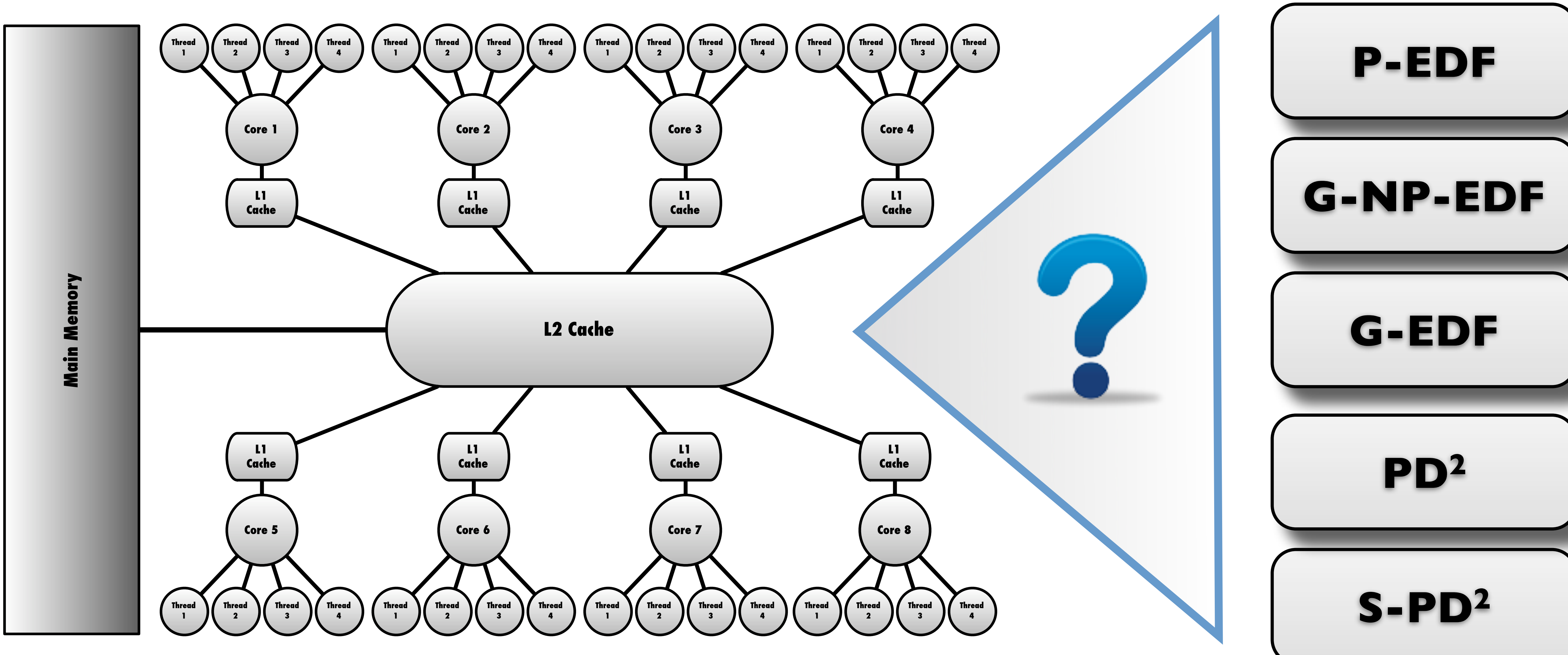
➡ What if there are **many slow processors**?

Calandrino et al. (2006), LITMUS[RT]: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

UNC's Implementation Studies (II)

Brandenburg et al. (2008)

- ➡ What if there are **many slow processors**?
- ➡ Explored **scalability** of RT schedulers on a Sun Niagara.

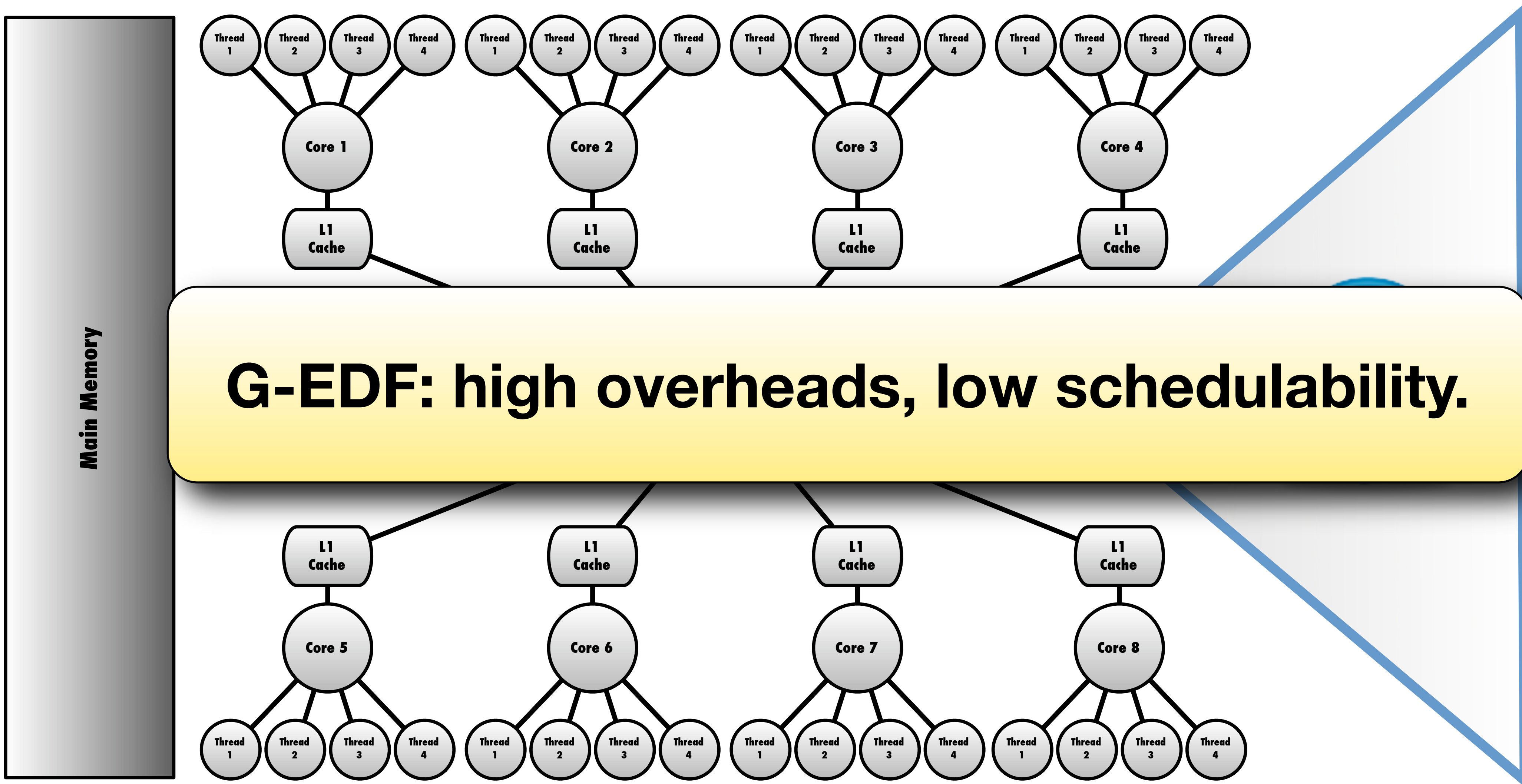


Calandrino et al. (2006), LITMUS[RT]: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

UNC's Implementation Studies (II)

Brandenburg et al. (2008)

- ➡ What if there are **many slow processors**?
- ➡ Explored **scalability** of RT schedulers on a Sun Niagara.



G-EDF: high overheads, low schedulability.

- **P-EDF**
- **G-NP-EDF**
- **G-EDF**
- **PD^2**
- **$S\text{-}PD^2$**

Calandrino et al. (2006), LITMUS^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

This Study

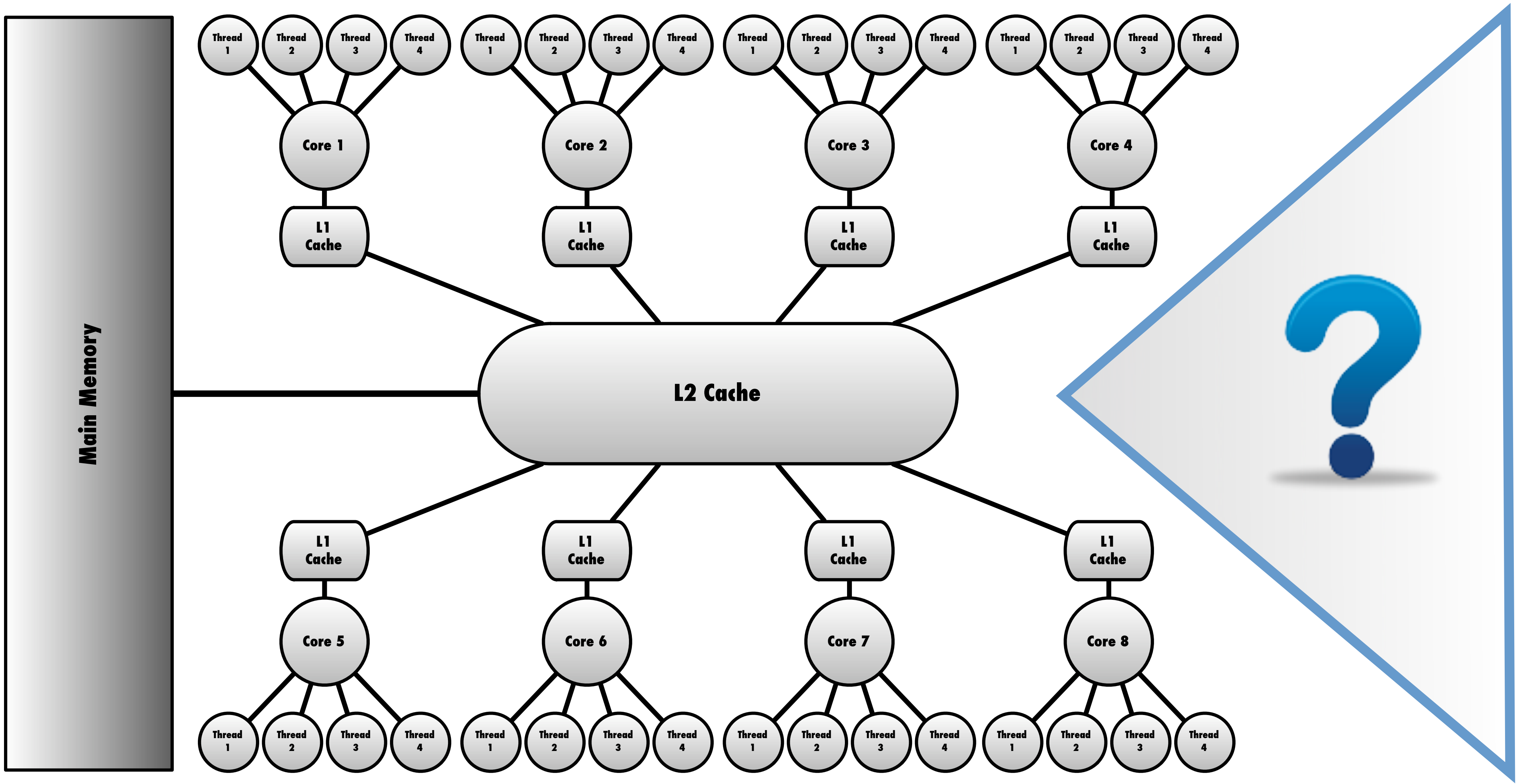
How to implement global schedulers?

Calandrino et al. (2006), LITMUS[RT]: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

This Study

How to implement global schedulers?

➡ Explore how **implementation tradeoffs** affect **schedulability**.



Instead of considering **one** implementation of several **different** scheduling algorithms…

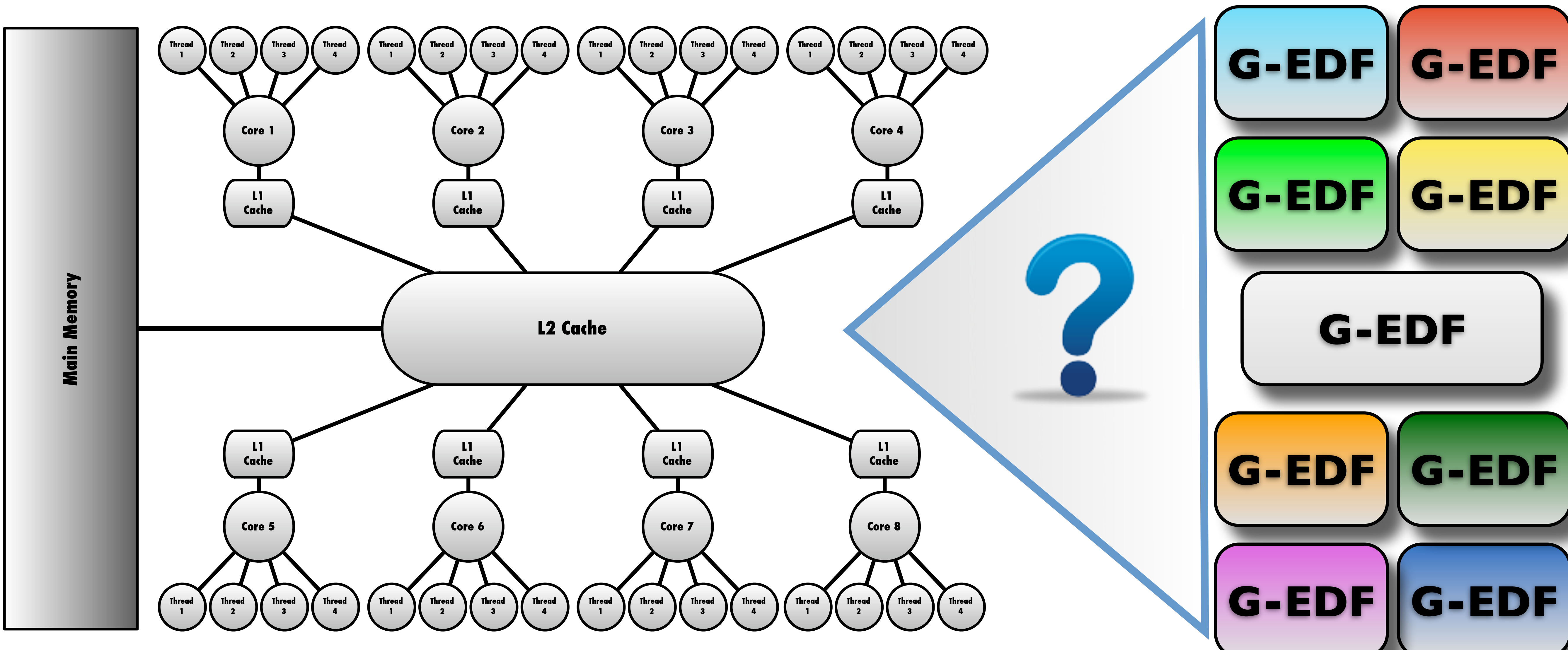
Calandrino et al. (2006), LITMUS[RT]: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

This Study

How to implement global schedulers?

➡ Explore how **implementation tradeoffs** affect **schedulability**.

➡ Case study: **nine G-EDF variants** on a Sun Niagara.



Calandrino et al. (2006), LITMUS[RT]: A testbed for empirically comparing real-time multiprocessor schedulers. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123.
Brandenburg et al. (2008), On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

Design Choices

Design Choices

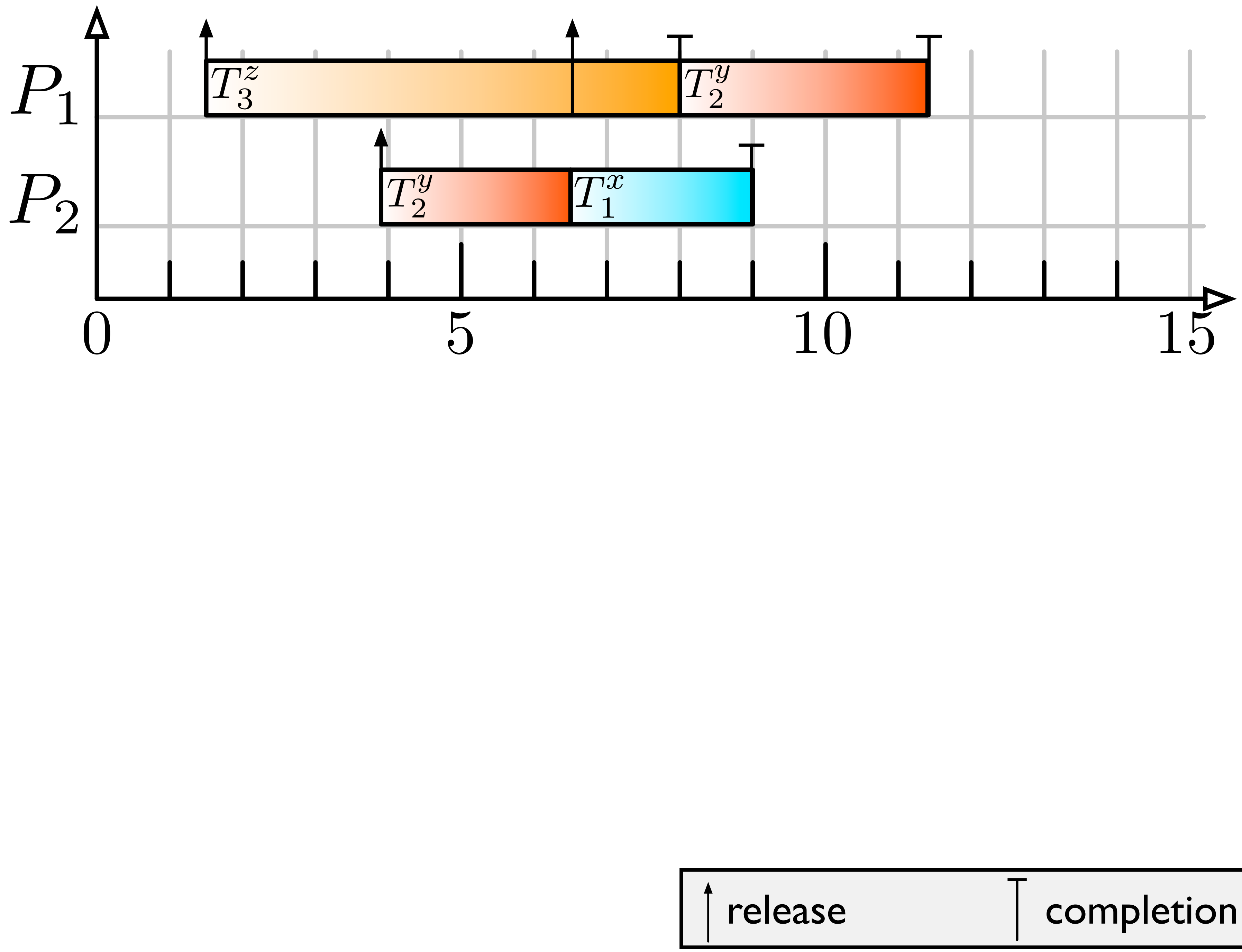
- ➡ When to schedule.
- ➡ Quantum alignment.
- ➡ How to handle interrupts.
- ➡ How to queue pending jobs.
- ➡ How to manage future releases.
- ➡ How to avoid unnecessary preemptions.

Scheduler Invocation

Scheduler Invocation

Event-Driven

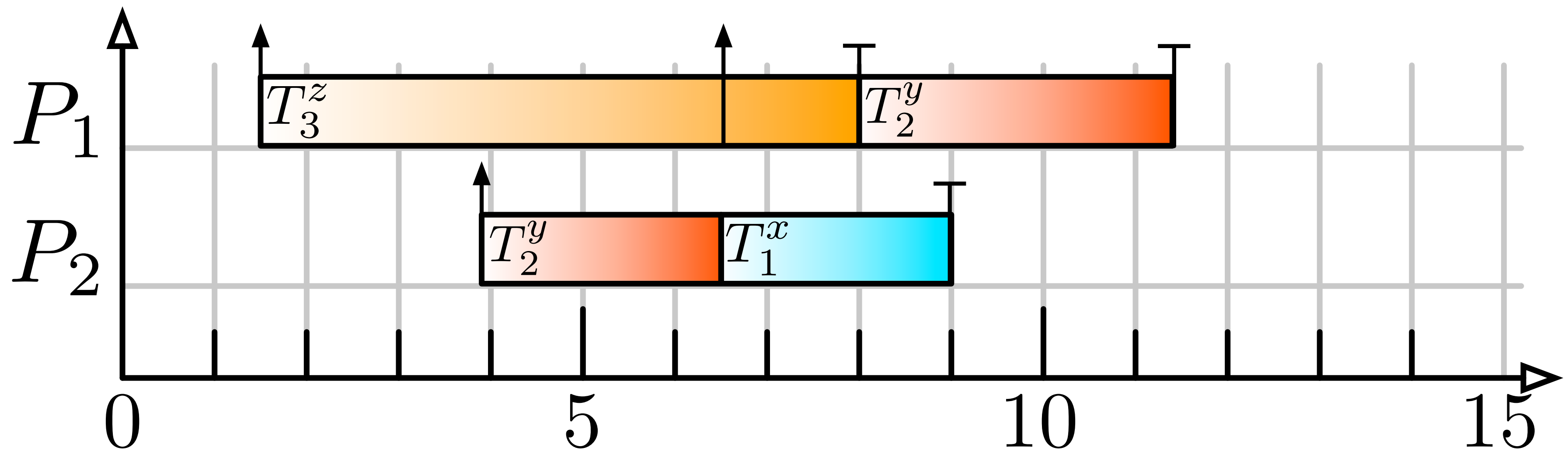
- ➡ on job release
- ➡ on job completion
- ➡ preemptions occur immediately



Scheduler Invocation

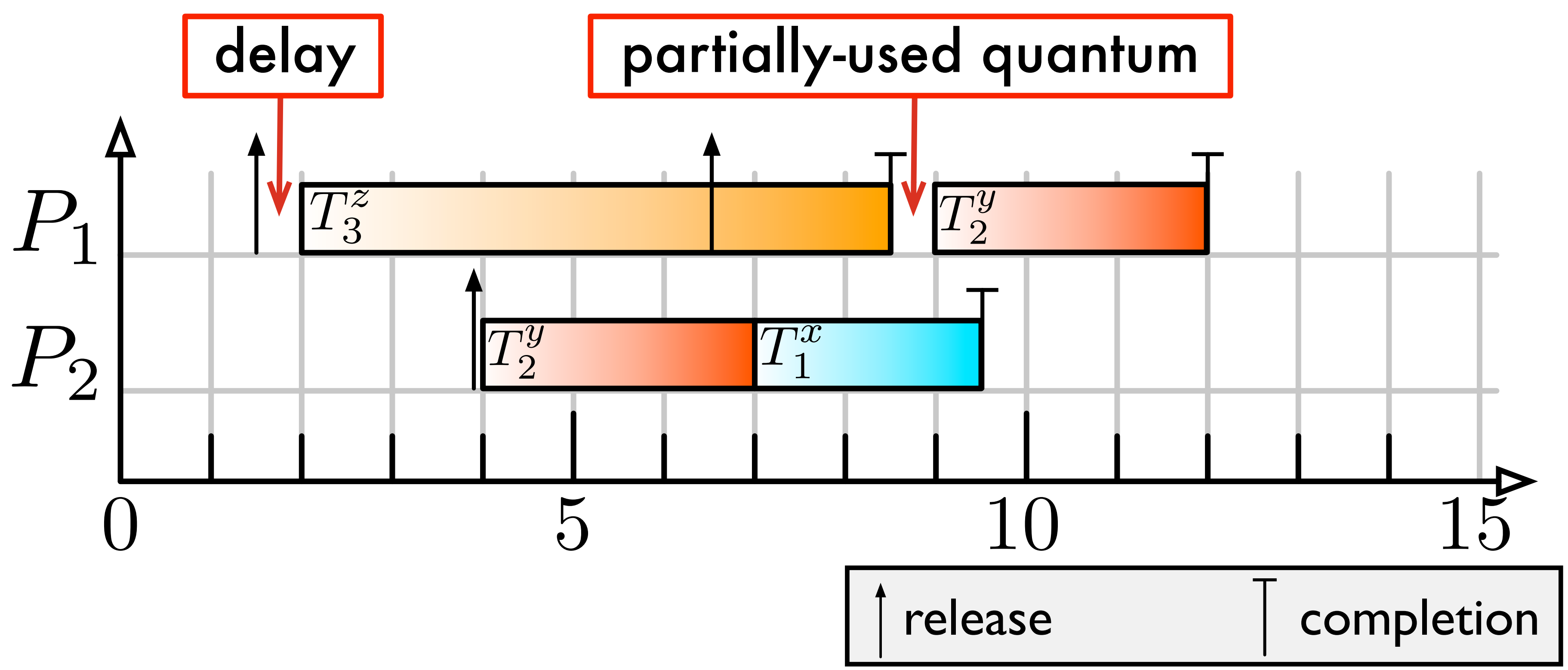
Event-Driven

- ➡ on job release
- ➡ on job completion
- ➡ preemptions occur immediately



Quantum-Driven

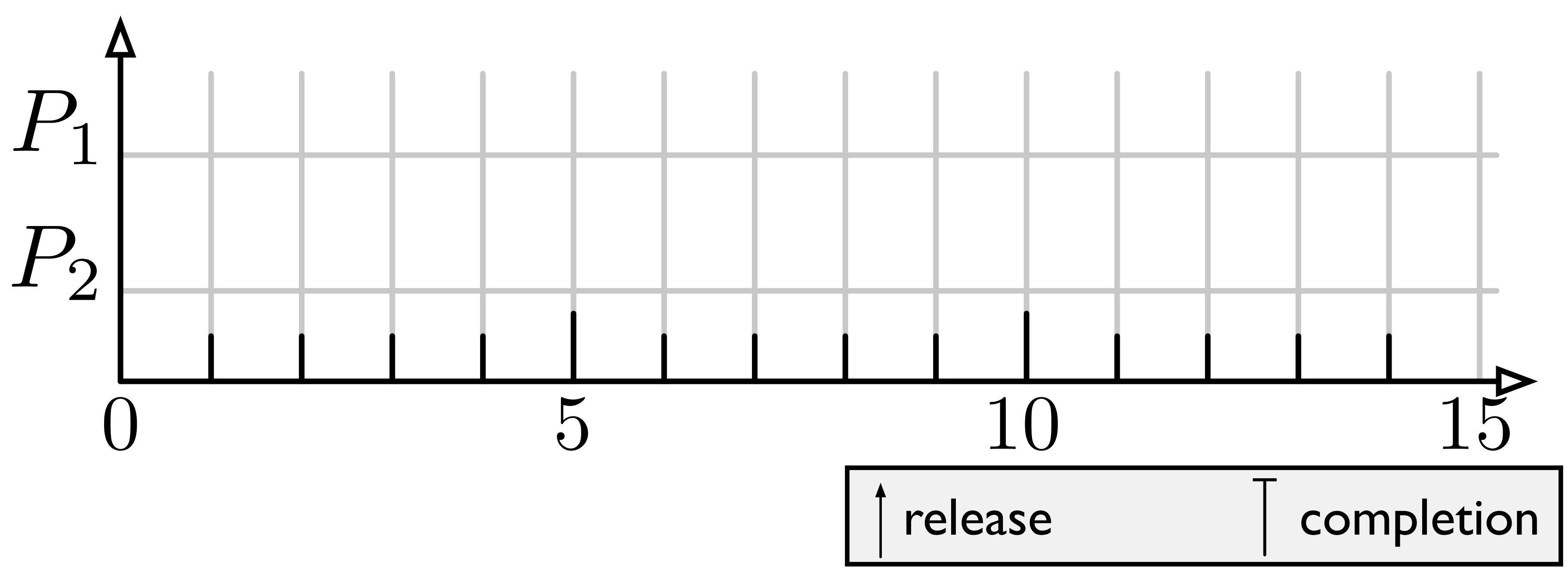
- ➡ on every timer tick
- ➡ easier to implement
- ➡ on release a job is just enqueued; scheduler is invoked at next tick



Quantum Alignment

Aligned

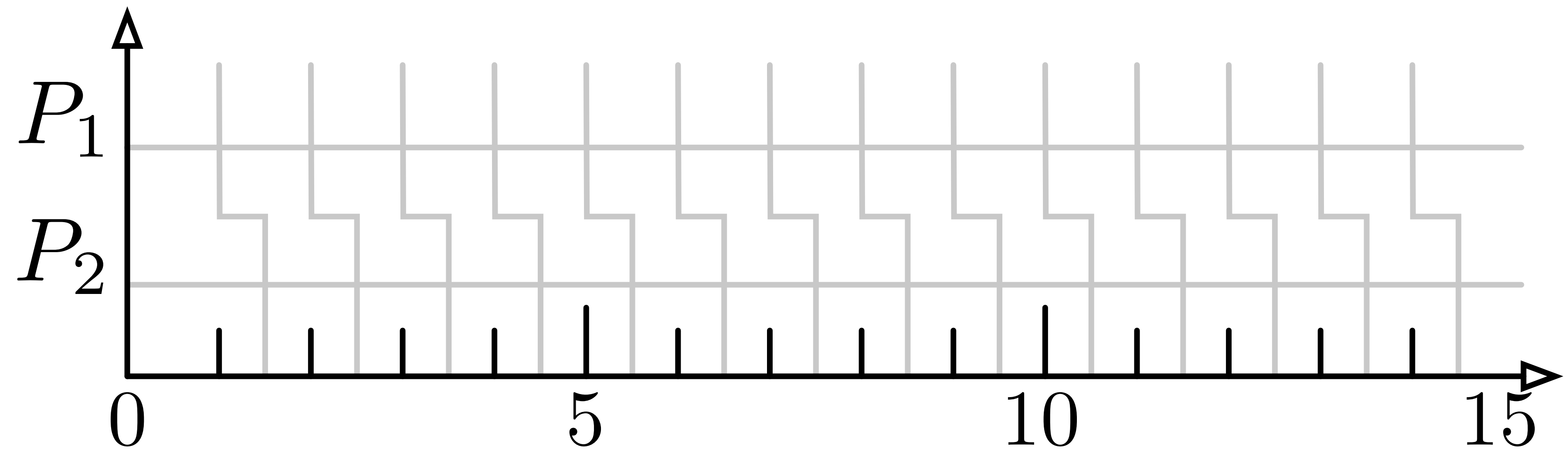
- ➡ Tick **synchronized** across processors.
- ➡ **Contention** at quantum boundary!



Quantum Alignment

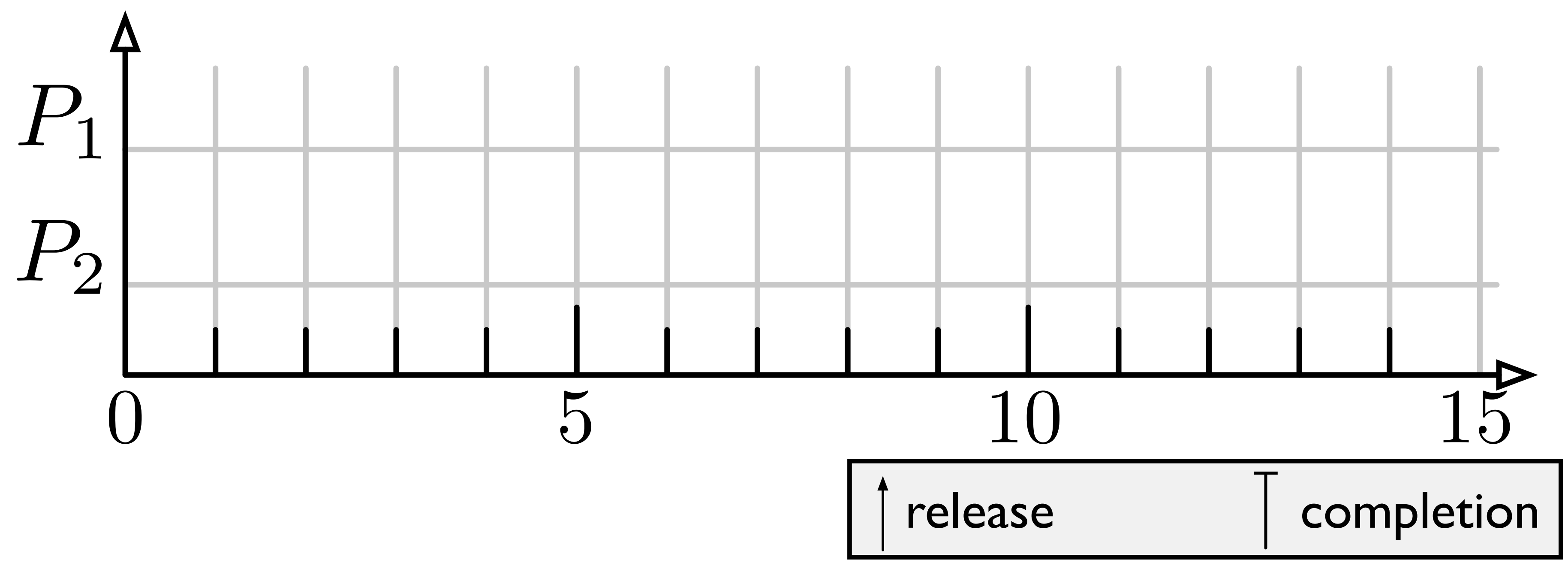
Staggered

- ➡ Ticks spread out across quantum.
- ➡ **Reduced** bus and lock contention.
- ➡ Additional **latency**.



Aligned

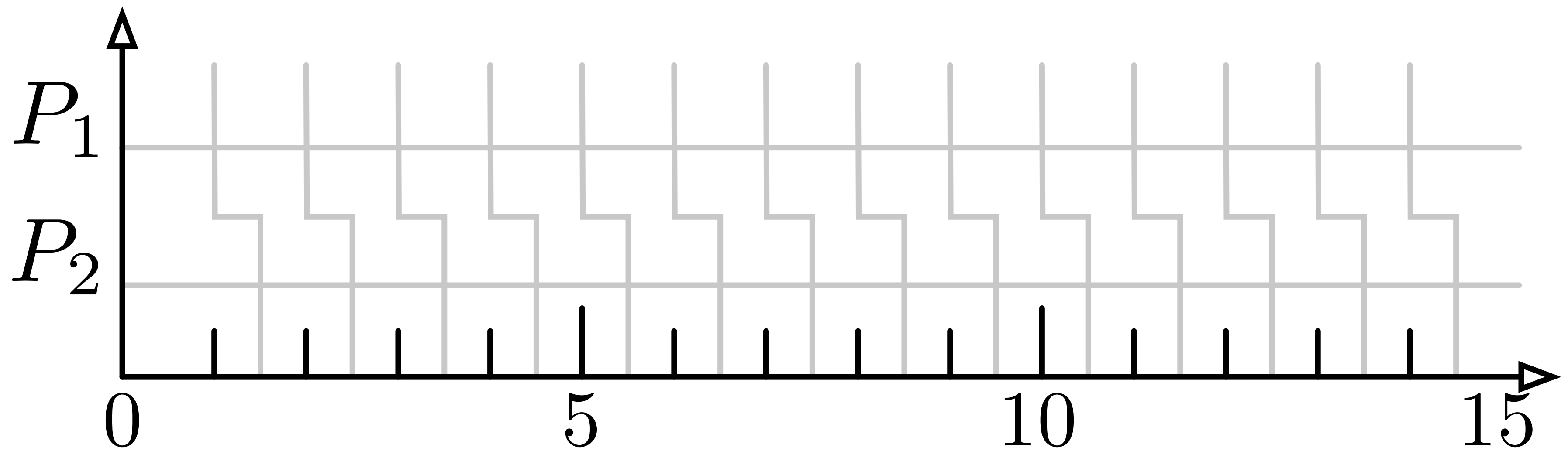
- ➡ Tick **synchronized** across processors.
- ➡ **Contention** at quantum boundary!



Quantum Alignment

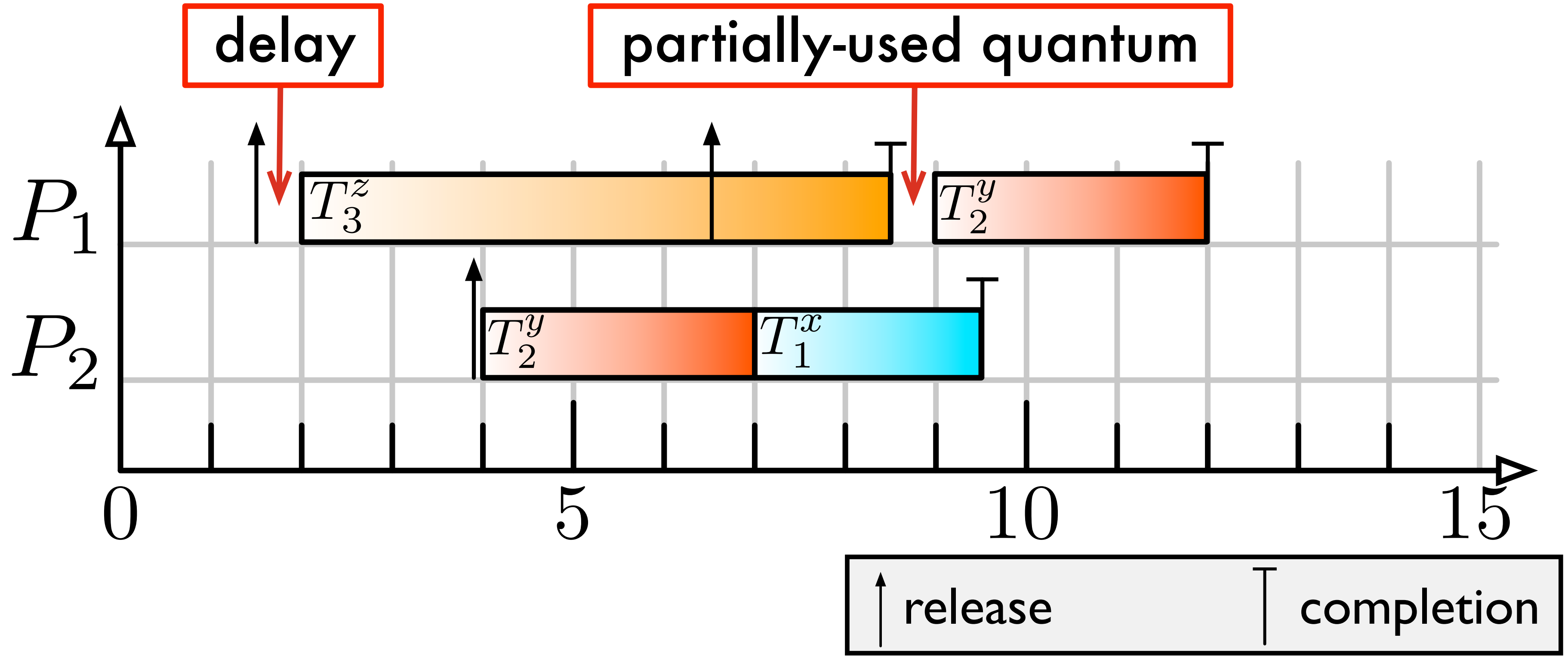
Staggered

- ➡ Ticks spread out across quantum.
- ➡ **Reduced** bus and lock contention.
- ➡ Additional **latency**.



Aligned

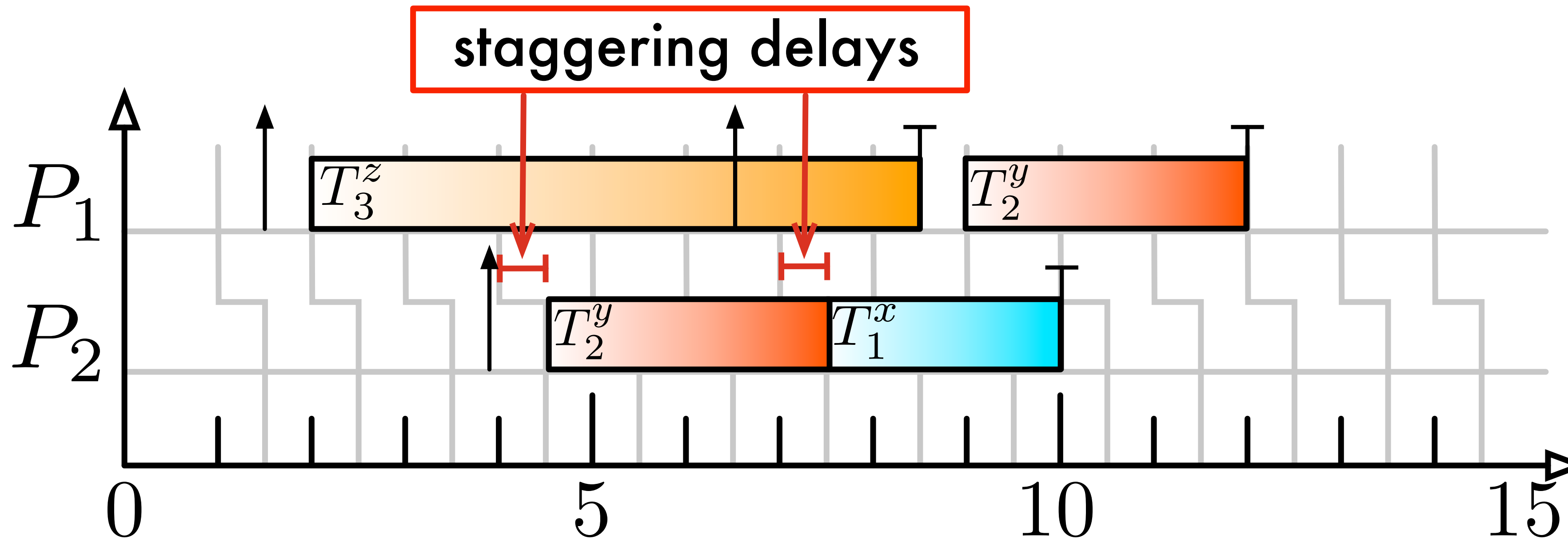
- ➡ Tick **synchronized** across processors.
- ➡ **Contention** at quantum boundary!



Quantum Alignment

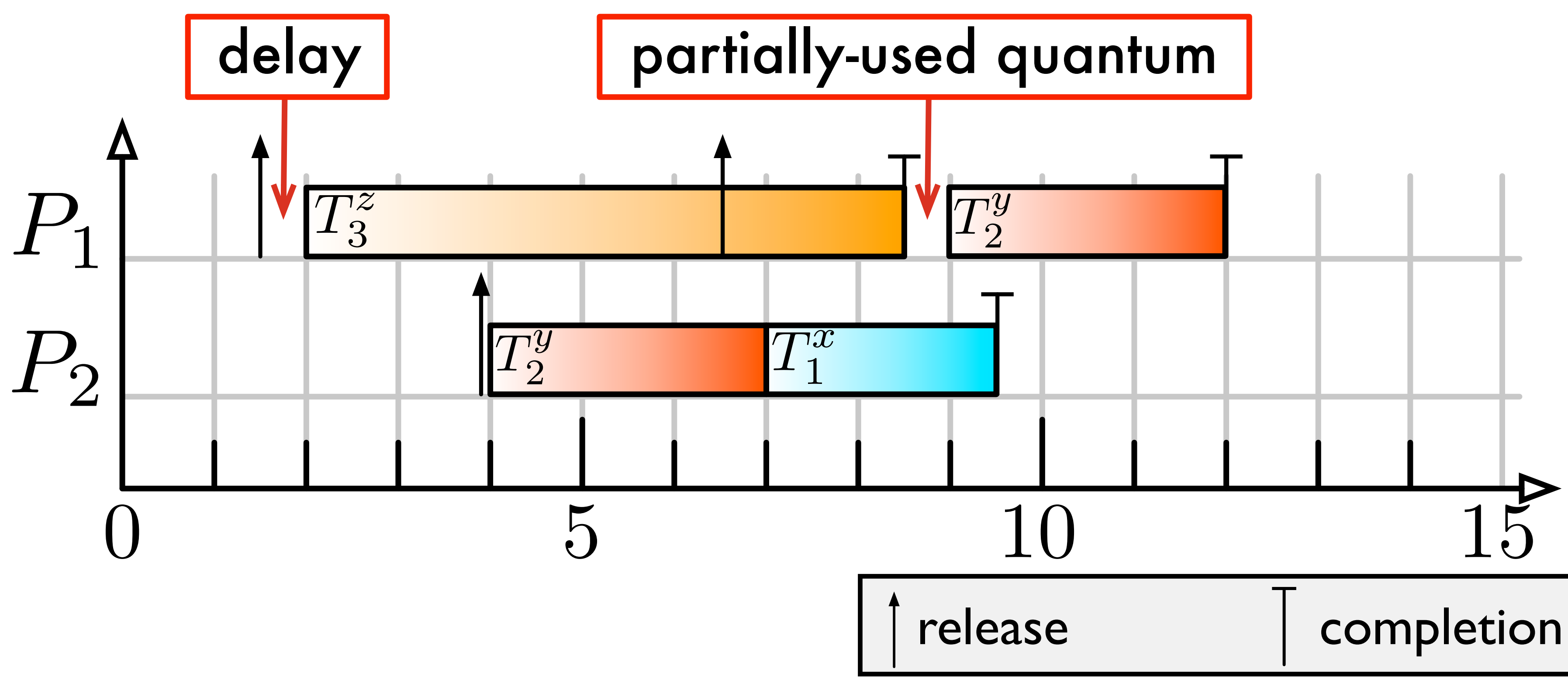
Staggered

- ➡ Ticks spread out across quantum.
- ➡ **Reduced** bus and lock contention.
- ➡ Additional **latency**.



Aligned

- ➡ Tick **synchronized** across processors.
- ➡ **Contention** at quantum boundary!



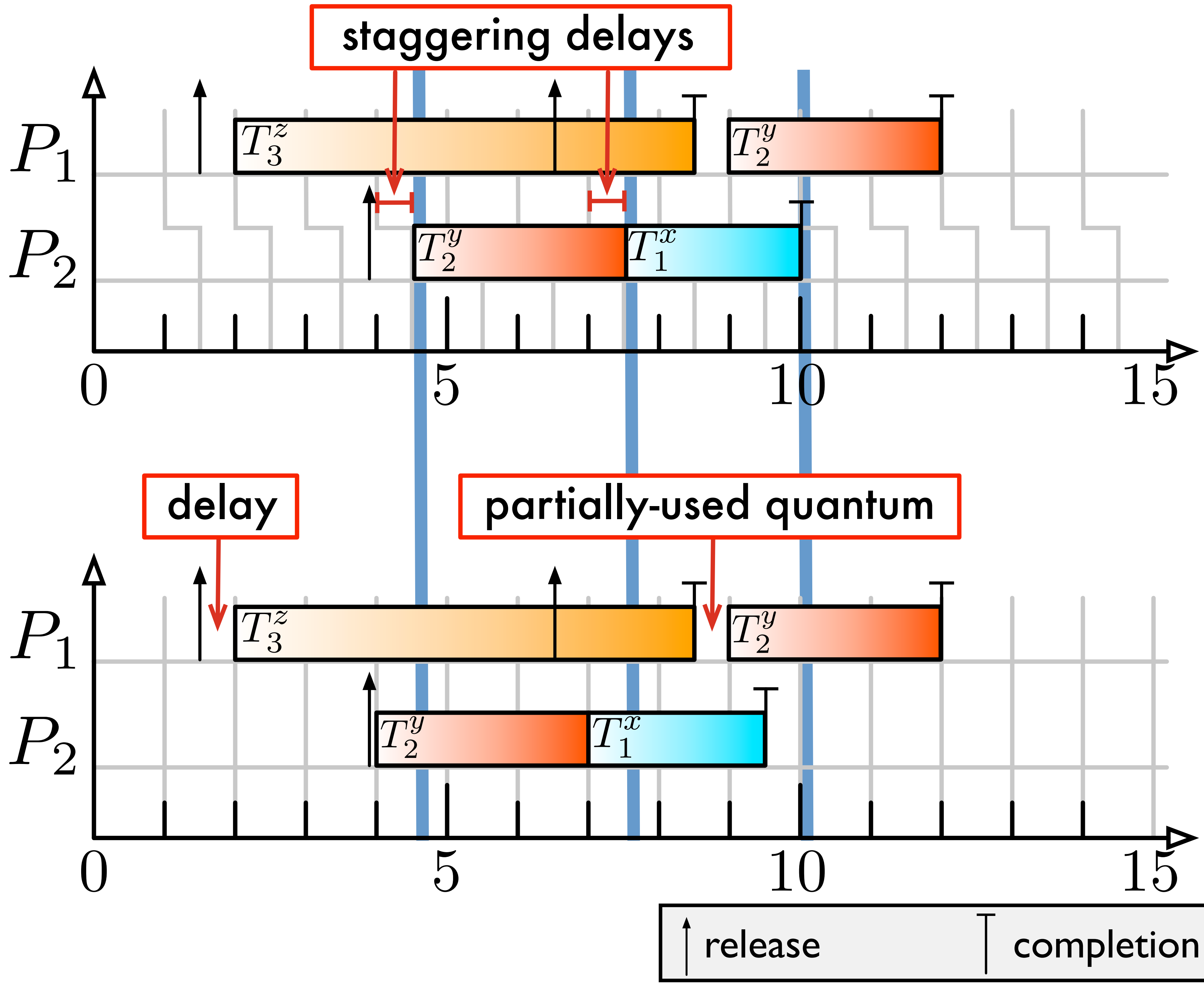
Quantum Alignment

Staggered

- ➡ Ticks spread out across quantum.
- ➡ **Reduced** bus and lock contention.
- ➡ Additional **latency**.

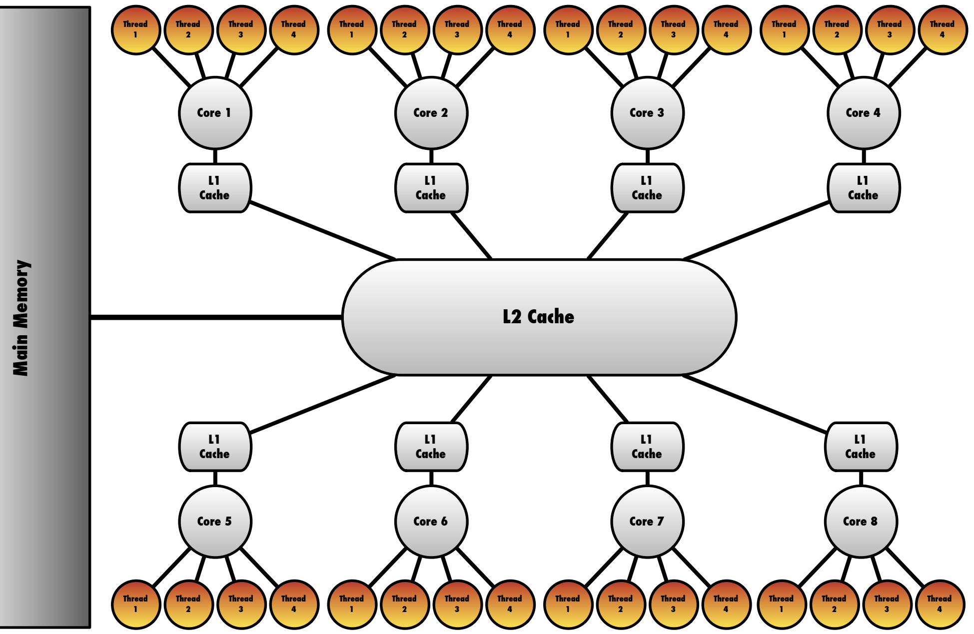
Aligned

- ➡ Tick **synchronized** across processors.
- ➡ **Contention** at quantum boundary!



Interrupt Handling

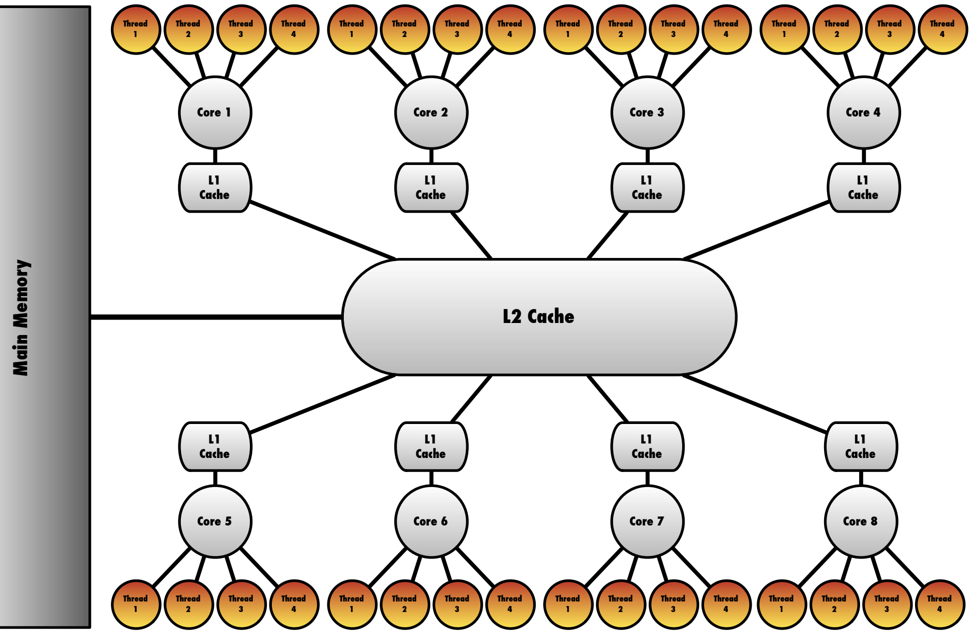
Interrupt Handling



Global interrupt handling.

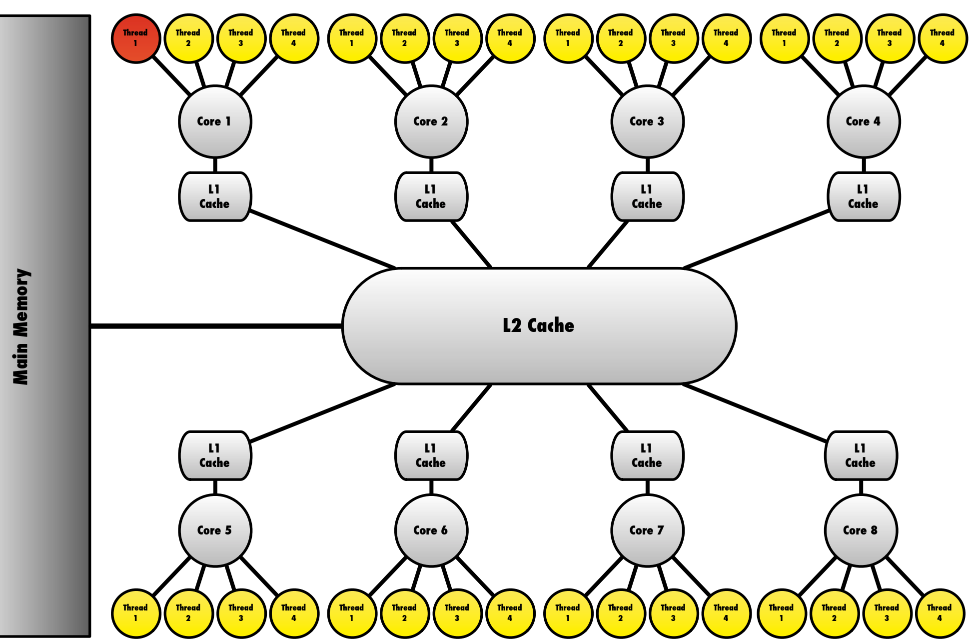
- ➡ Job releases triggered by **interrupts**.
- ➡ Interrupts may fire **on any processor**.
- ➡ Jobs may execute **on any processor**.
- ➡ Thus, in the worst case, a job may be **delayed by each interrupt**.

Interrupt Handling



Global interrupt handling.

- ➡ Job releases triggered by **interrupts**.
- ➡ Interrupts may fire **on any processor**.
- ➡ Jobs may execute **on any processor**.
- ➡ Thus, in the worst case, a job may be **delayed by each interrupt**.

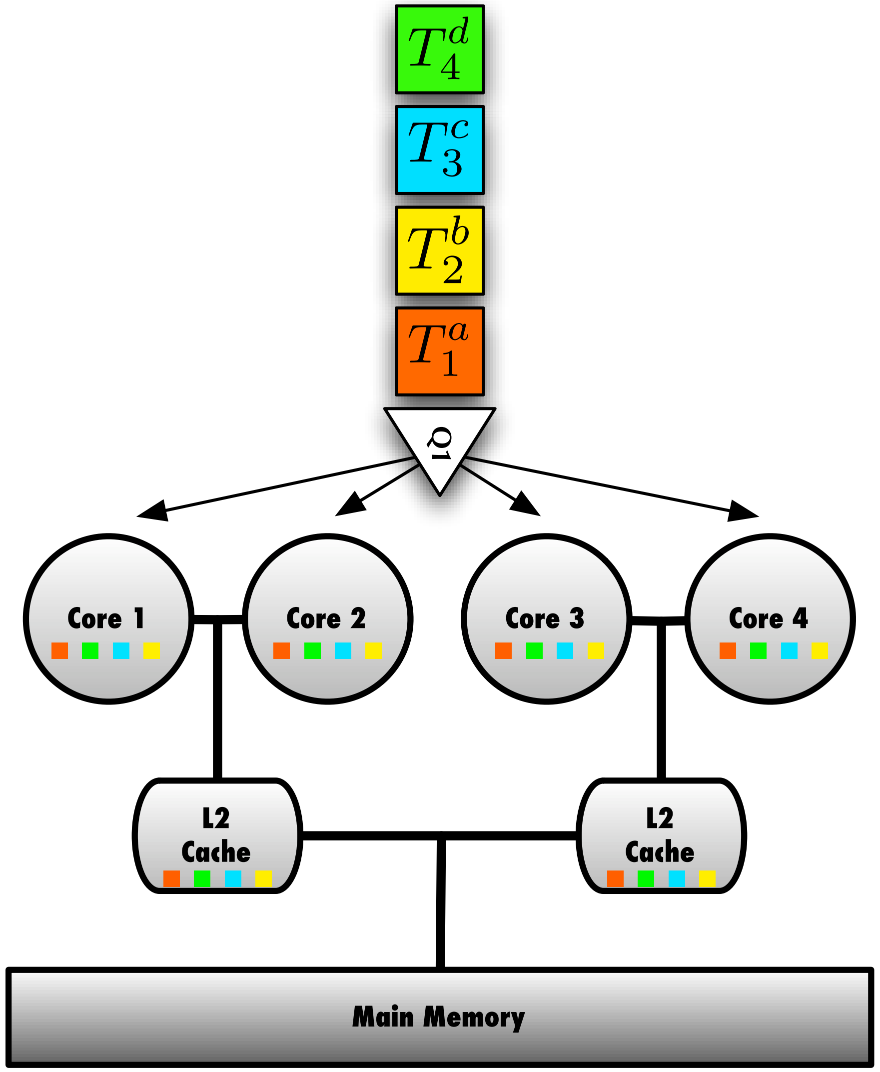


Dedicated interrupt handling.

- ➡ **Only one processor** services interrupts.
- ➡ Jobs may execute **on other processors**.
- ➡ Jobs are not delayed by release interrupts.
- ➡ Well-known technique; used in the **Spring** kernel (Stankovic and Ramamritham, 1991).
- ➡ How does it affect **schedulability**?

J.A. Stankovic and K. Ramamritham (1991), The Spring kernel: A new paradigm for real-time systems. *IEEE Software*, 8(3):62–72.

Ready Queue


T_4^d
T_3^c
T_2^b
T_1^a
Q1
Core 1
Core 2
Core 3
Core 4
L2 Cache
L2 Cache
Main Memory

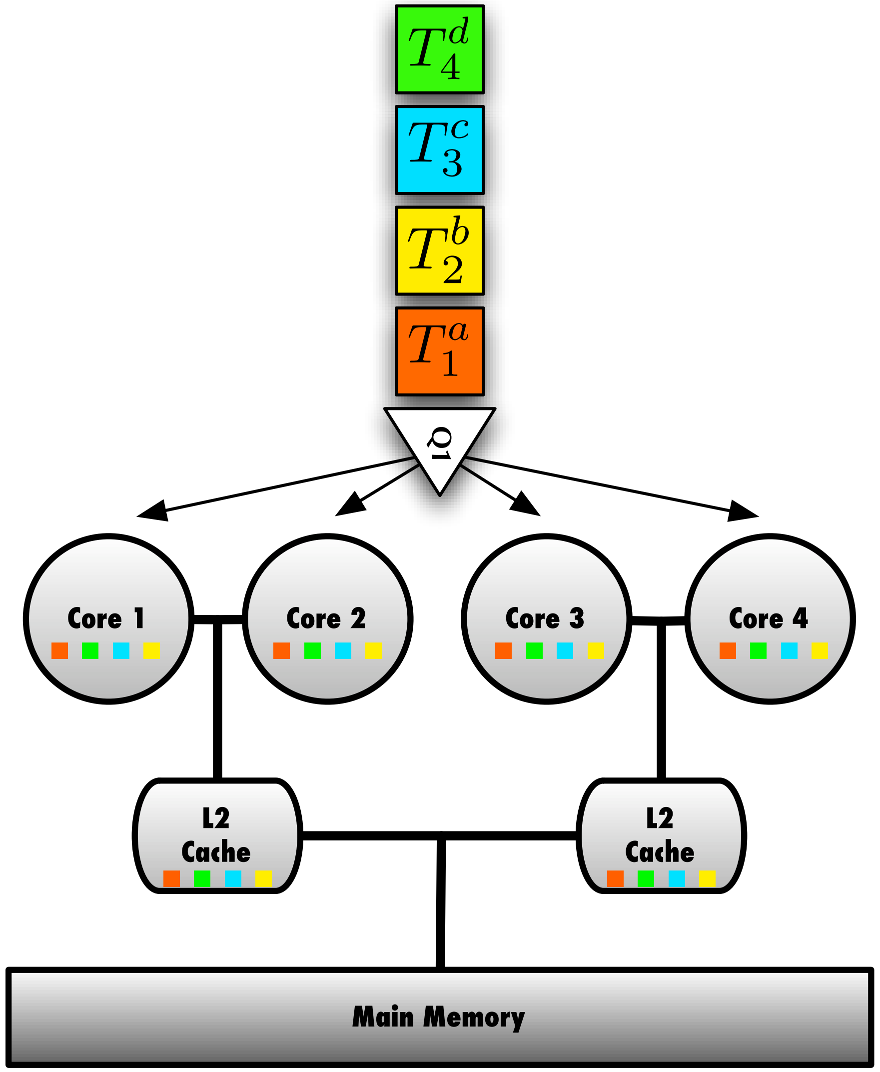
Ready Queue

Globally-shared priority queue.

➡ Problem: **hyper-period boundaries**.

➡ Problem: **lock contention**.

➡ Problem: **bus contention**.



Ready Queue

Globally-shared priority queue.

➡ Problem: **hyper-period boundaries**.

➡ Problem: **lock contention**.

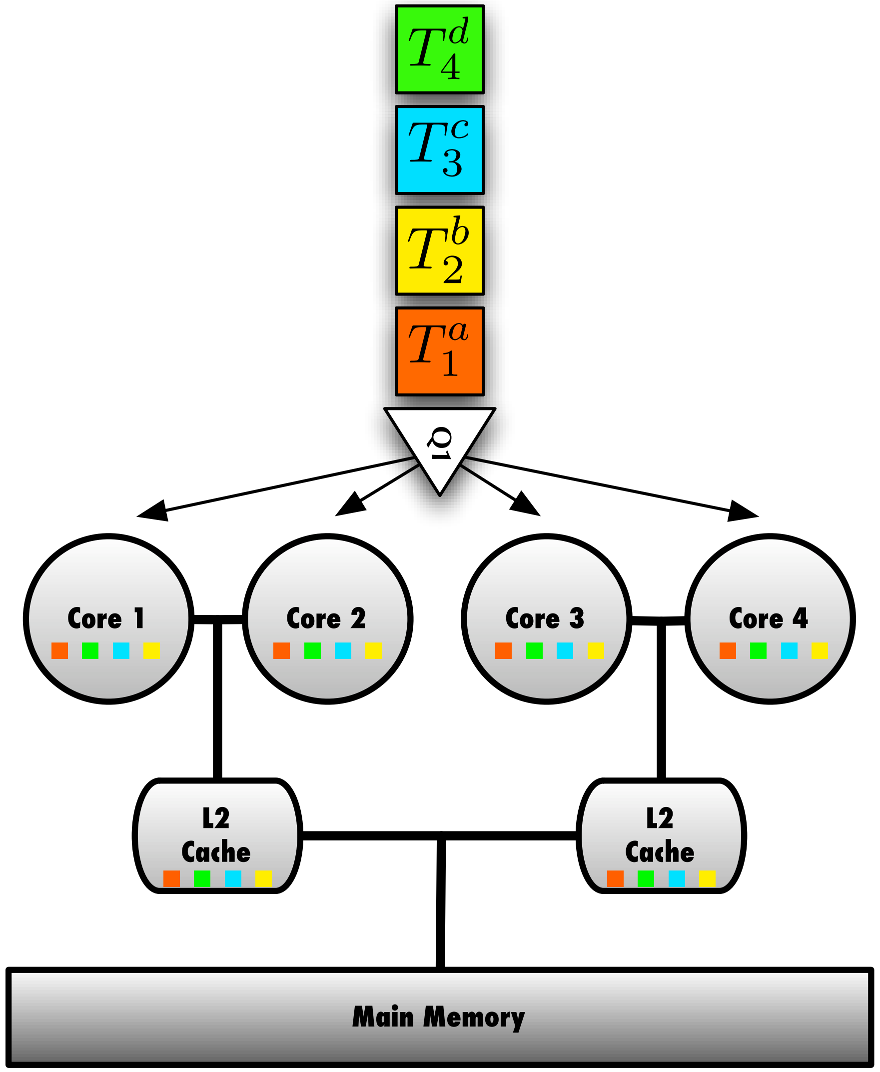
➡ Problem: **bus contention**.

Requirements.

➡ **Mergeable** priority queue: release *n* jobs in O(log *n*) time.

➡ **Parallel** enqueue / dequeue operations.

➡ Mostly **cache-local** data structures.



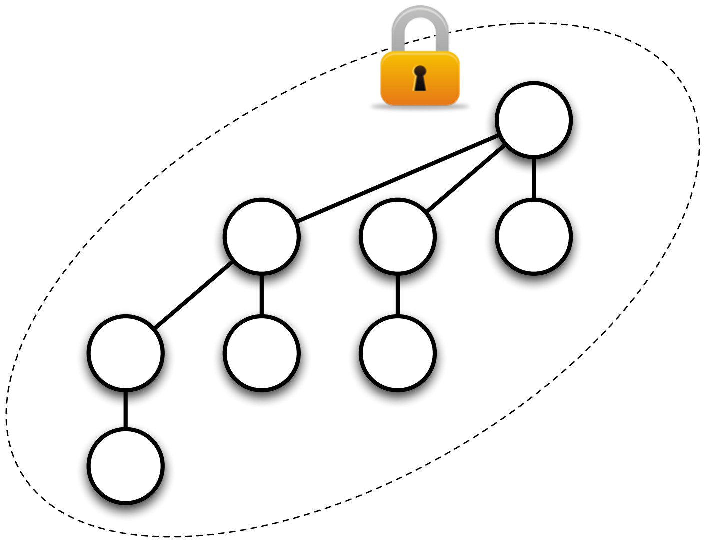
Ready Queue

Globally-shared priority queue.

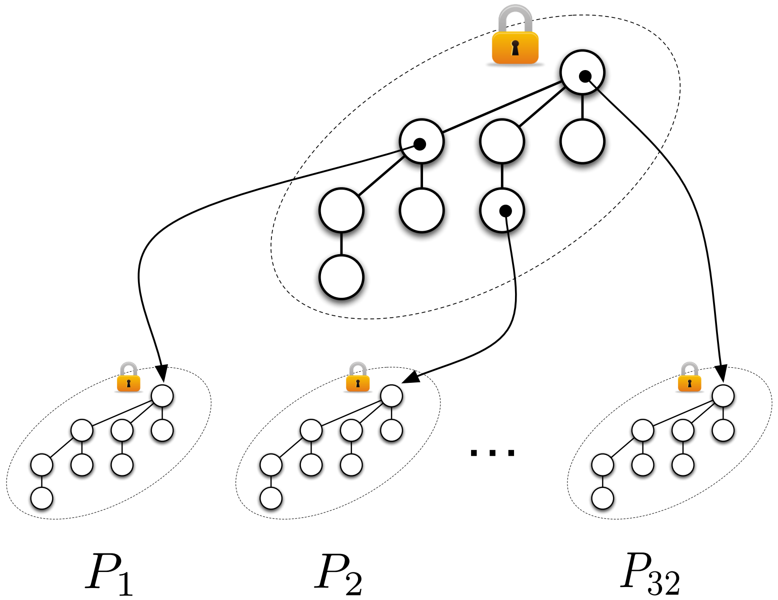
- ➡ Problem: **hyper-period boundaries**.
- ➡ Problem: **lock contention**.
- ➡ Problem: **bus contention**.

In this study, we consider three queue implementations.

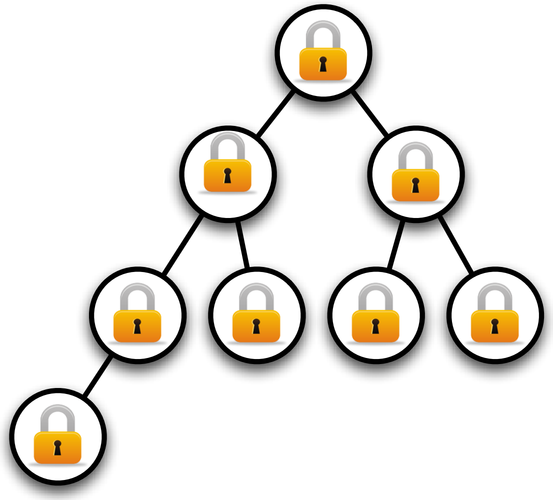
Coarse-Grained Heap



Hierarchical Heaps



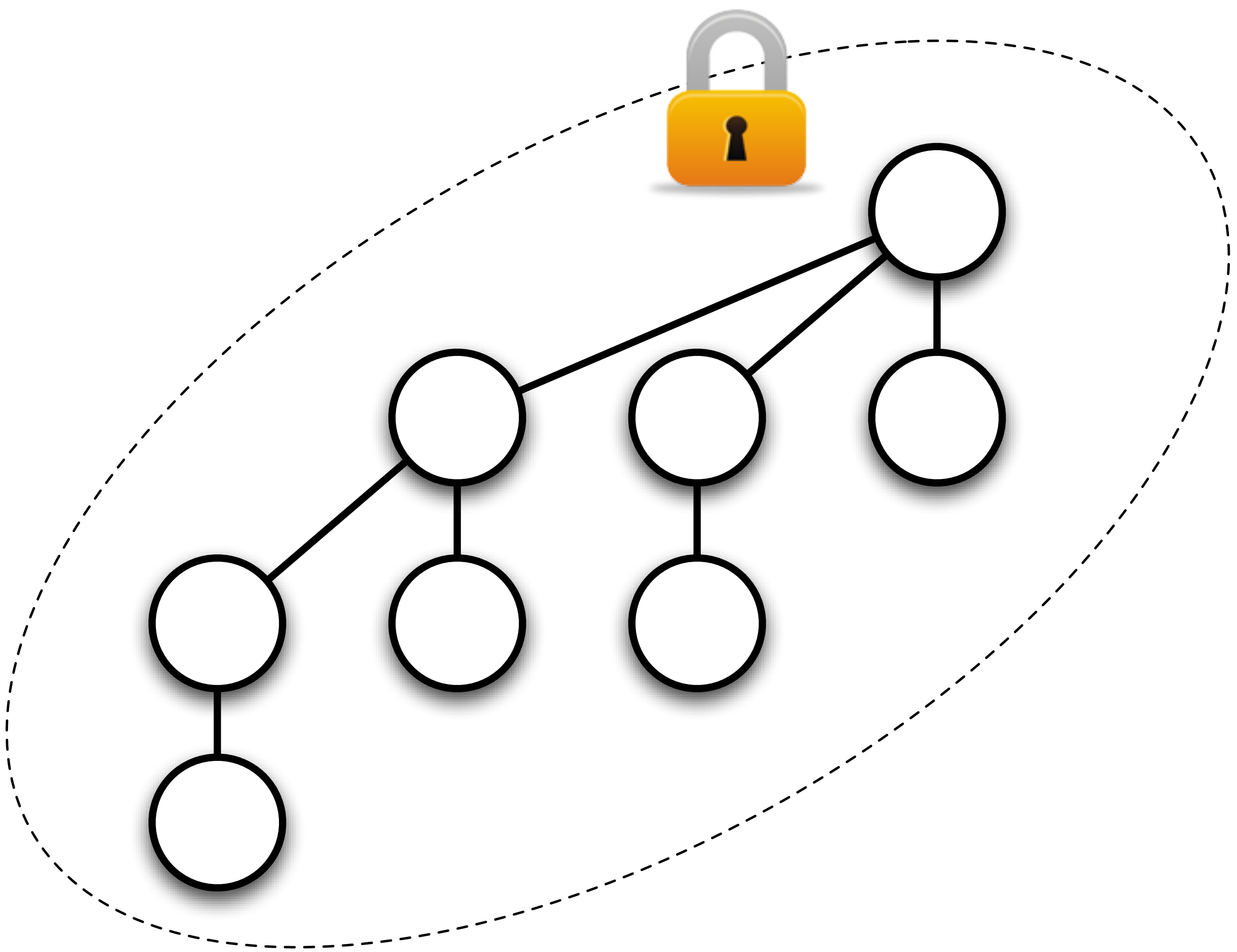
Fine-Grained Heap



Ready Queue: Coarse-Grained Heap

Binomial heap + single lock.

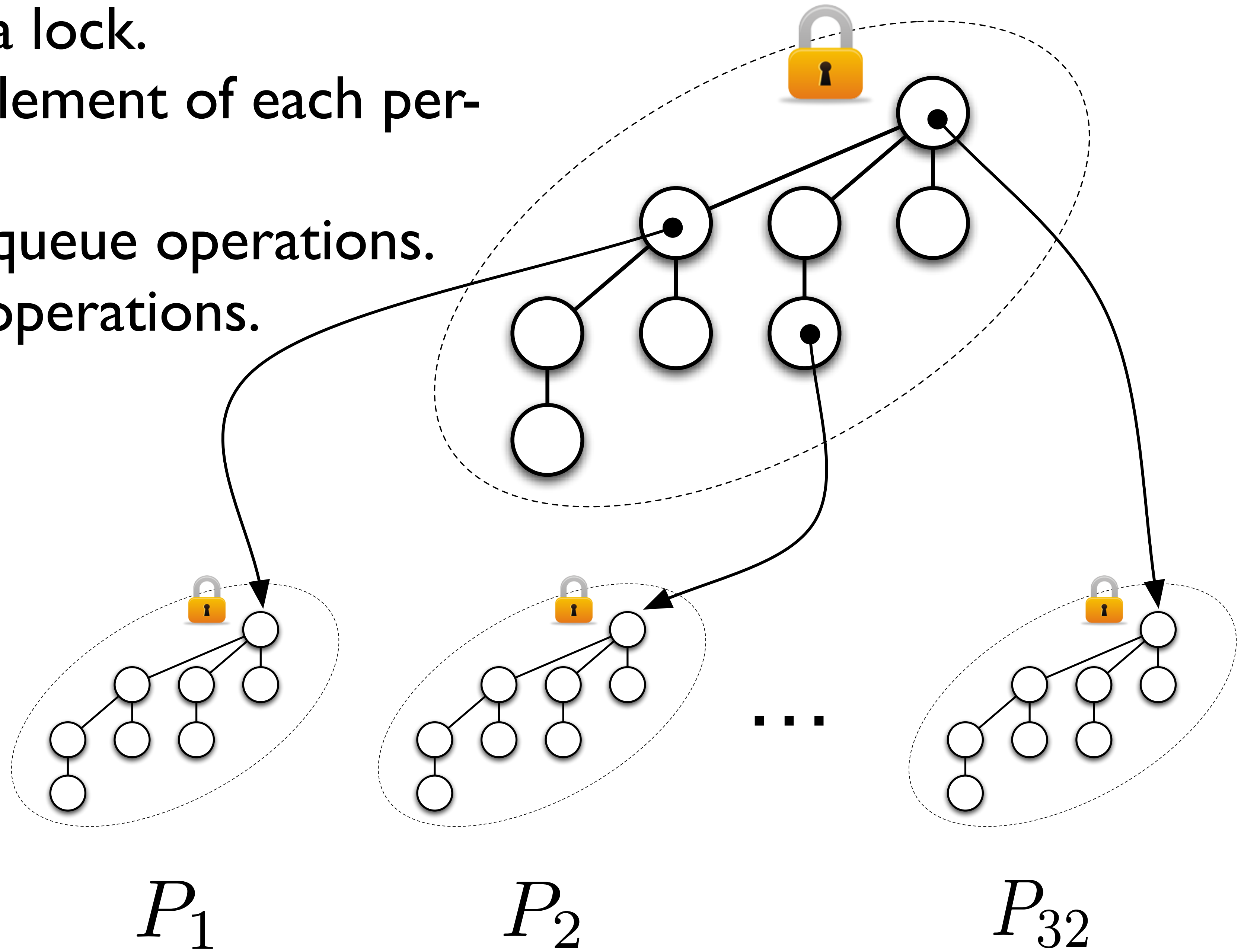
- ➡ Lock used to synchronize all G-EDF state.
- ➡ **Mergeable** queue.
- ➡ No parallel updates.
- ➡ No cache-local updates.
- ➡ Low locking overhead (only single lock acquisition).



Ready Queue: Hierarchical Heaps

Per-processor queues + master queue.

- ➡ Each queue protected by a lock.
- ➡ Master queue holds min element of each per-processor queue.
- ➡ **Global, sequential** dequeue operations.
- ➡ **Mostly-local** enqueue operations.



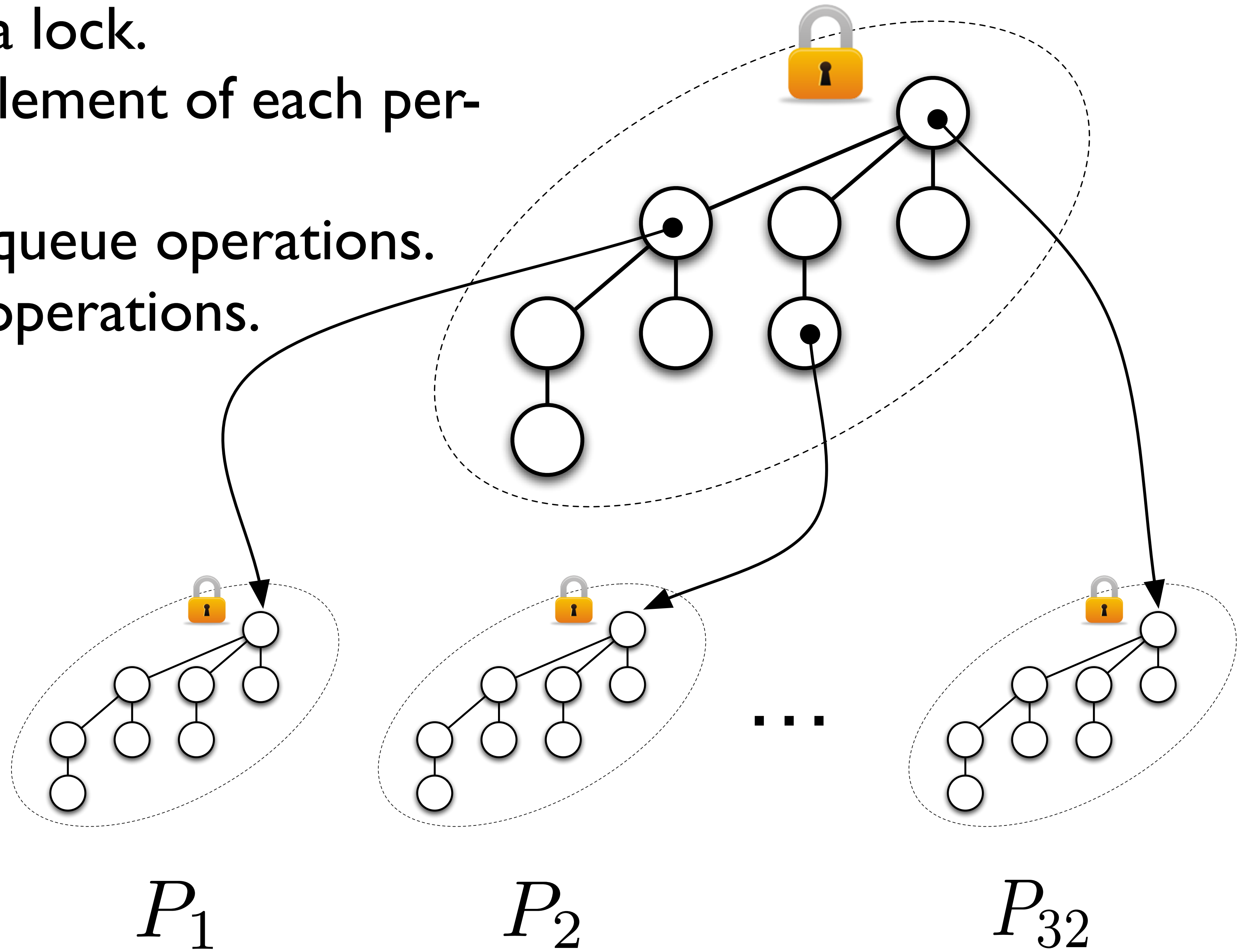
Ready Queue: Hierarchical Heaps

Per-processor queues + master queue.

- Each queue protected by a lock.
- Master queue holds min element of each per-processor queue.
- **Global, sequential** dequeue operations.
- **Mostly-local** enqueue operations.

Locking.

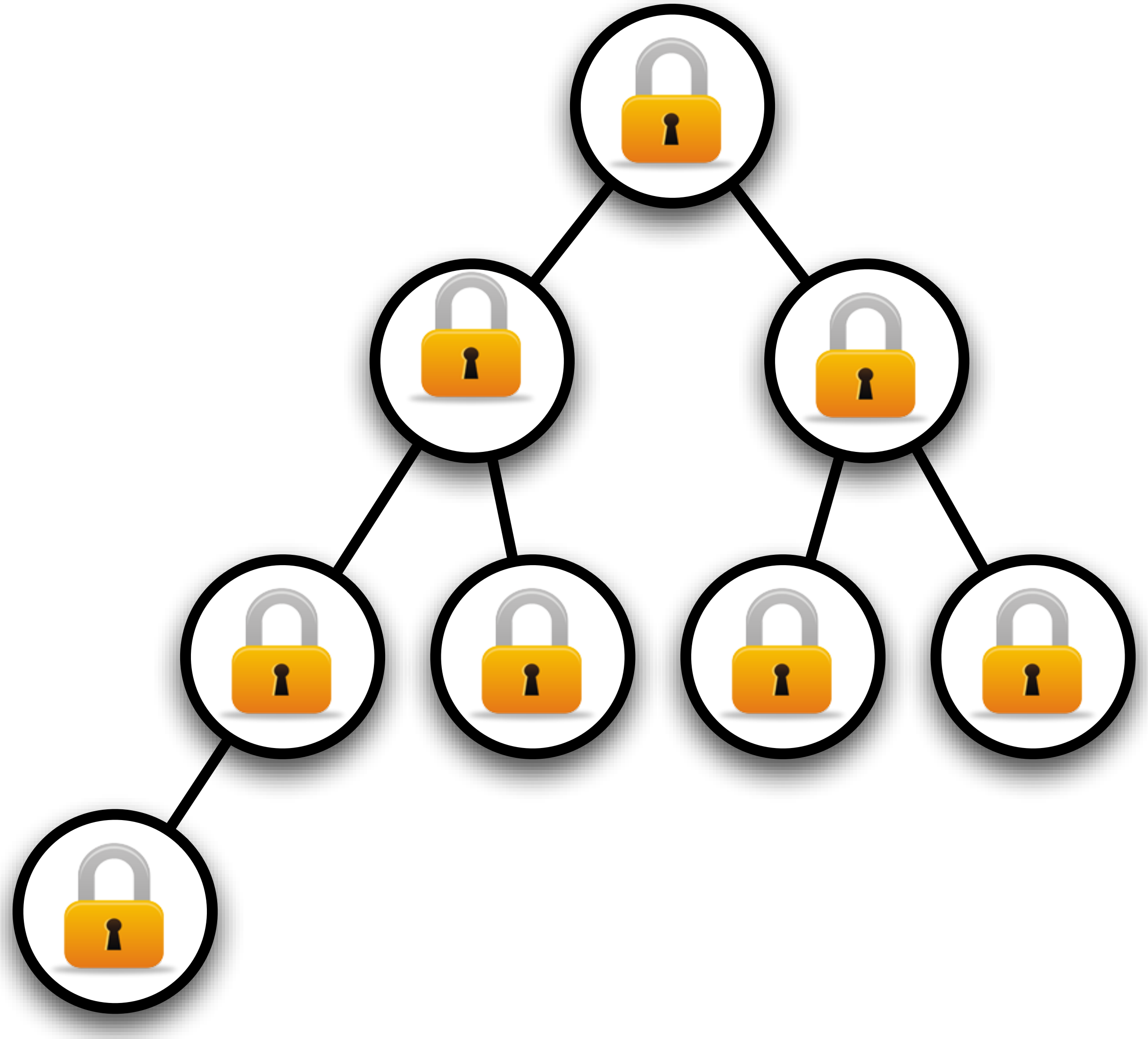
- Dequeue: top-down.
- Enqueue: bottom-up.
- Enqueue may have to drop lock, retry.
- Additional complexity wrt. dequeue (see paper).
- Bottom line: **expensive**.



Ready Queue: Fine-Grained Heap

Parallel binary heap.

- ➡ One lock per heap node.
- ➡ Proposed by Hunt et al. (1996).
- ➡ **Not mergeable**.
- ➡ **Parallel enqueue / dequeue**.
- ➡ **No cache-local data**.



Hunt et al. (1996), An efficient algorithm for concurrent priority queue heaps. *Information Processing Letters*, 60(3):151–157.

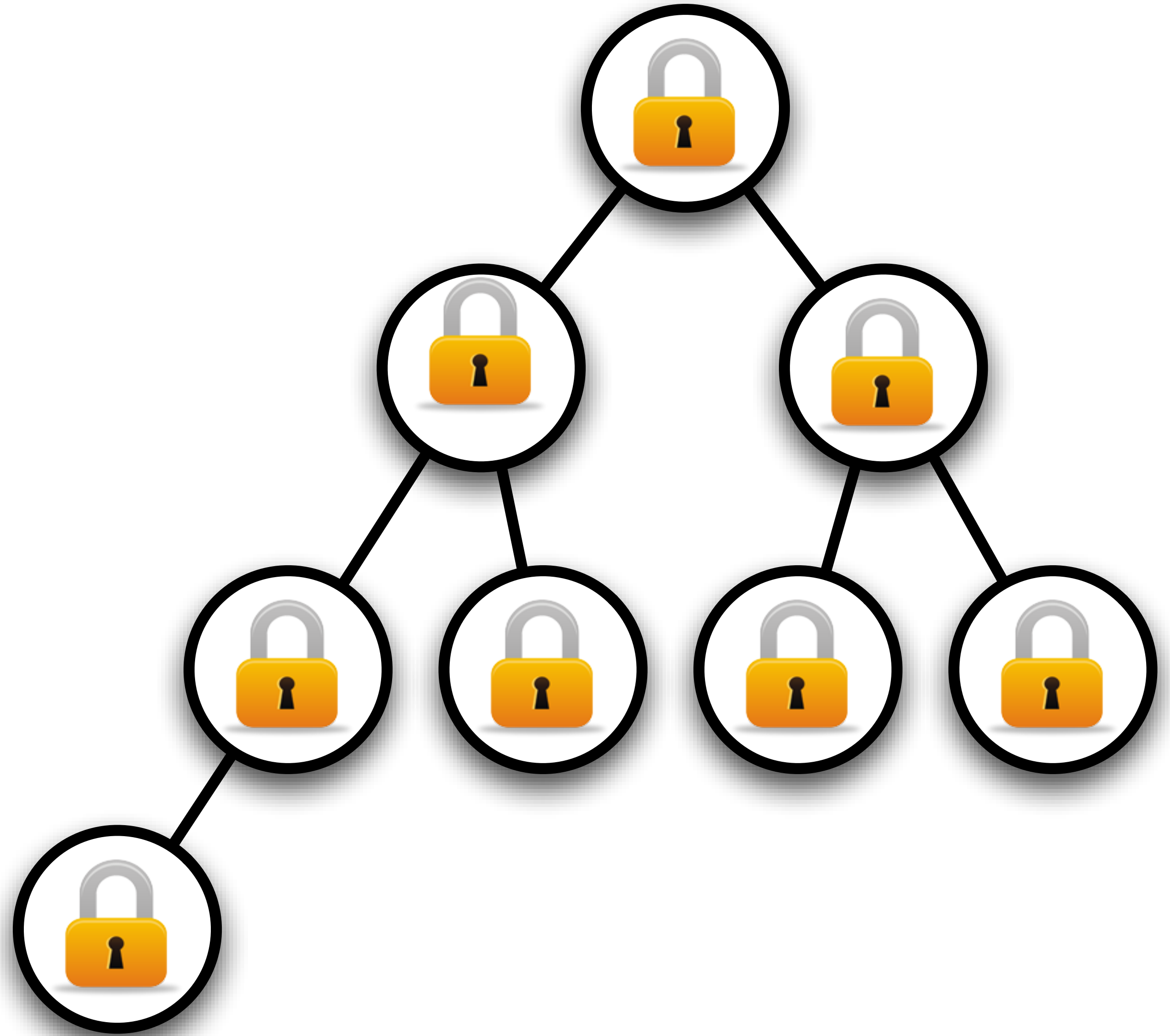
Ready Queue: Fine-Grained Heap

Parallel binary heap.

- ➡ One lock per heap node.
- ➡ Proposed by Hunt et al. (1996).
- ➡ **Not mergeable**.
- ➡ **Parallel enqueue / dequeue**.
- ➡ **No cache-local data**.

Locking.

- ➡ Many lock acquisitions.
- ➡ Atomic **peek+dequeue** operation needed to check for preemptions.



Hunt et al. (1996), An efficient algorithm for concurrent priority queue heaps. *Information Processing Letters*, 60(3):151–157.

Additional Components

Release queue.

➡ Support mergeable queues.
➡ Support dedicated interrupt handling.

Job-to-processor mapping.

➡ Quickly determine whether preemption is required.
➡ Avoid unnecessary preemptions.
➡ Used to linearize concurrent scheduling decisions.

(Details in the paper.)

Implementation in *LITMUS*[RT]

LITMUS^{RT}

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Linux Testbed for Multiprocessor Scheduling in Real-Time systems

LITMUS RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Linux Testbed for Multiprocessor Scheduling in Real-Time systems

UNC's Linux patch.

➡ Used in several previous studies.

➡ On-going development.

➡ Currently, based off of Linux 2.6.24.

LITMUS^RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Linux Testbed for Multiprocessor Scheduling in Real-Time systems

UNC's Linux patch.

- ➡ Used in several previous studies.
- ➡ On-going development.
- ➡ Currently, based off of Linux 2.6.24.

Scheduler Plugin API.

- ➡ `scheduler_tick()`
- ➡ `schedule()`
- ➡ `release_jobs()`

Considered G-EDF Variants

Name	Ready Q	Scheduling	Interrupts

Considered G-EDF Variants

Name	Ready Q	Scheduling	Interrupts
CEm	coarse-grained	event-driven	global
CQm	coarse-grained	quantum (aligned)	global
S-CQm	coarse-grained	quantum (staggered)	global
HEm	hierarchical	event-driven	global
FEm	fine-grained	event-driven	global

Co... ...nts

Baseline from
(Brandenburg et al., 2008)

Name	Ready Q	Scheduling	Interrupts
CEm	coarse-grained	event-driven	global
CQm	coarse-grained	quantum (aligned)	global
S-CQm	coarse-grained	quantum (staggered)	global
HEm	hierarchical	event-driven	global
FEm	fine-grained	event-driven	global

No fine-grained heaps + quantum-driven scheduling.
(Parallel updates not beneficial due to quantum barrier.)

Name	Ready Q	Scheduling	Interrupts
CEm	coarse-grained	event-driven	global
CQm	coarse-grained	quantum (aligned)	global
S-CQm	coarse-grained	quantum (staggered)	global
HEm	hierarchical	event-driven	global
FEm	fine-grained	event-driven	global

Considered G-EDF Variants

Name	Ready Q	Scheduling	Interrupts
CEm	coarse-grained	event-driven	global
CQm	coarse-grained	quantum (aligned)	global
S-CQm	coarse-grained	quantum (staggered)	global
HEm	hierarchical	event-driven	global
FEm	fine-grained	event-driven	global
CE1	coarse-grained	event-driven	dedicated
CQ1	coarse-grained	quantum (aligned)	dedicated
S-CQ1	coarse-grained	quantum (staggered)	dedicated
FE1	fine-grained	event-driven	dedicated

No hierarchical heaps + dedicated interrupt handling.
(Hierarchical heaps not beneficial if only one proc. enqueues.)

Name	Ready Q	Scheduling	Interrupts
CEm	coarse-grained	event-driven	global
CQm	coarse-grained	quantum (aligned)	global
S-CQm	coarse-grained	quantum (staggered)	global
HEm	hierarchical	event-driven	global
FEm	fine-grained	event-driven	global
CE1	coarse-grained	event-driven	dedicated
CQ1	coarse-grained	quantum (aligned)	dedicated
S-CQ1	coarse-grained	quantum (staggered)	dedicated
FE1	fine-grained	event-driven	dedicated

Schedulability Study

Objective

Compare the discussed implementations in terms of the ratio of randomly-generated task sets that can be shown to be schedulable ***under consideration of system overheads****.*

Scheduling Overheads

Scheduling Overheads

Release overhead.

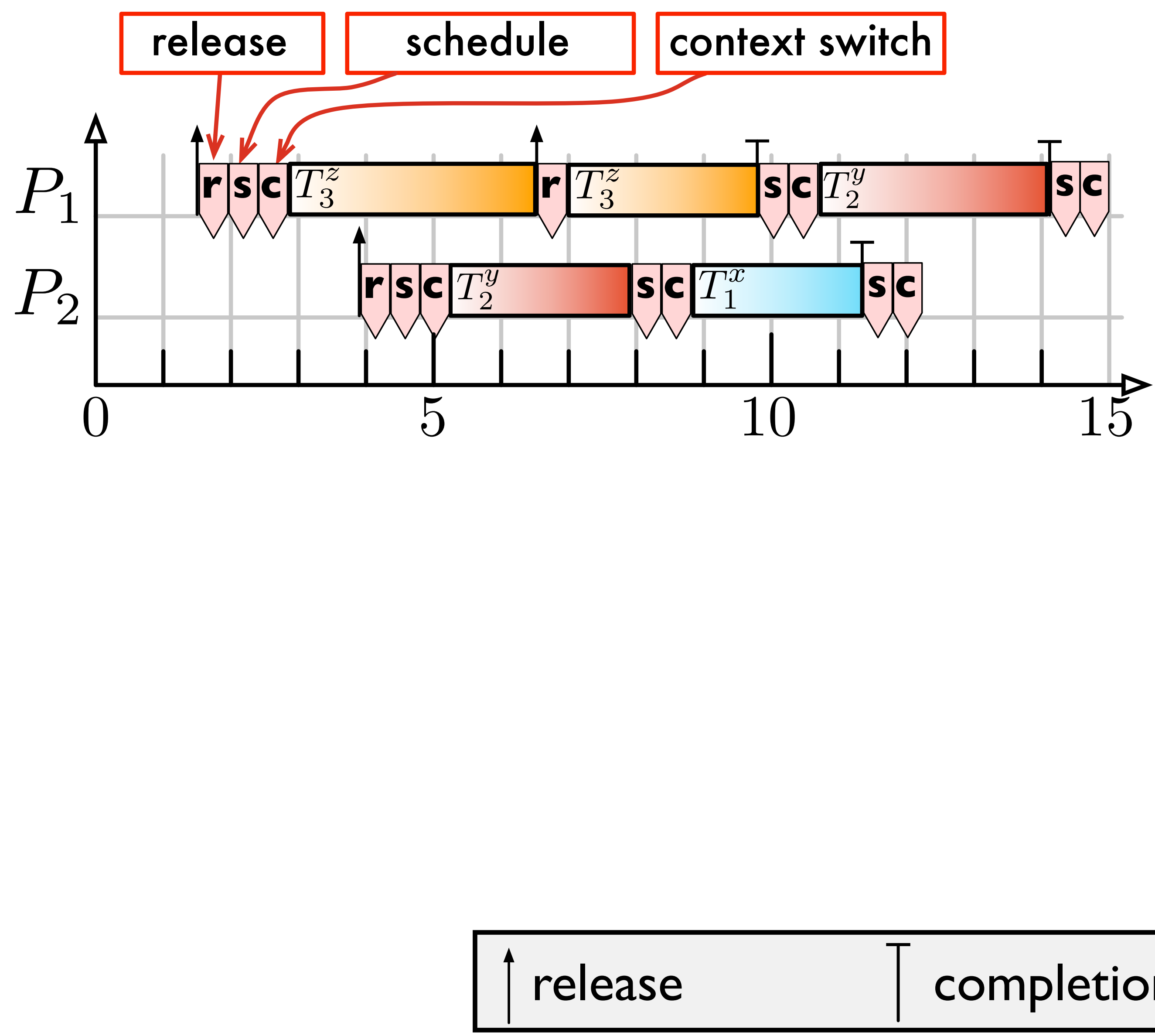
➡ The cost of a one-shot timer interrupt.

Scheduling overhead.

➡ Selecting the next job to run.

Context switch overhead.

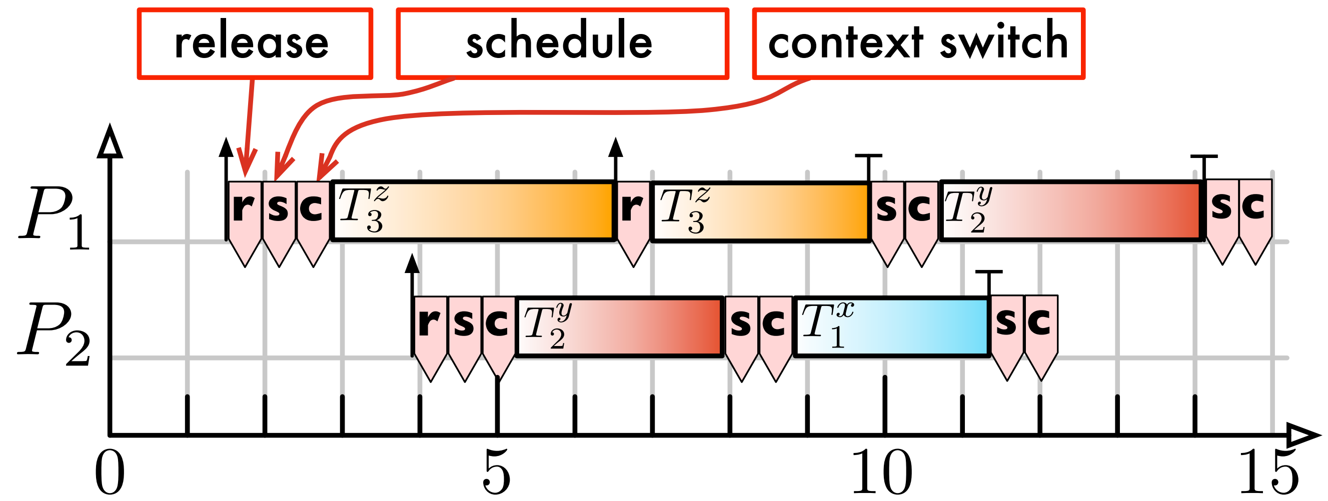
➡ Changing address space.



Scheduling Overheads

Release overhead.
➡ The cost of a one-shot timer interrupt.

Scheduling overhead.
➡ Selecting the next job to run.



Context switch overhead.
➡ Changing address space.

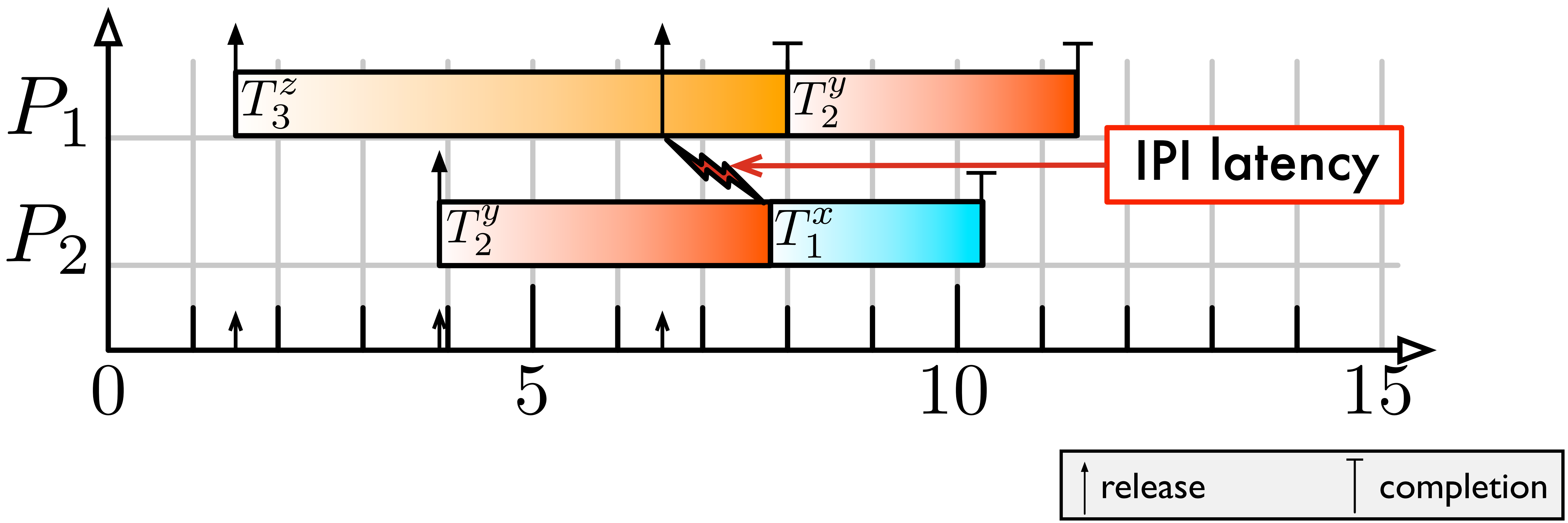
Tick overhead.
➡ Cost of a periodic timer interrupt.
➡ Beginning of a new quantum.

Preemption and migration overhead.
➡ Loss of cache affinity.
➡ Known from (Brandenburg et al., 2008).

IPI Latency

Inter-processor interrupts (IPIs).

- ➡ Interrupt may be processed by a processor different from the one that will schedule a newly-arrived job.
- ➡ Requires notification of remote processor.
- ➡ **Event-based scheduling incurs added latency**.



Test Platform

LITMUS^{RT}

➡ UNC's Linux-based Real-Time Testbed

Sun UltraSPARC T1 "Niagara"

➡ 8 cores, 4 HW threads per core = 32 logical processors.

➡ 3 MB shared L2 cache

Test Platform

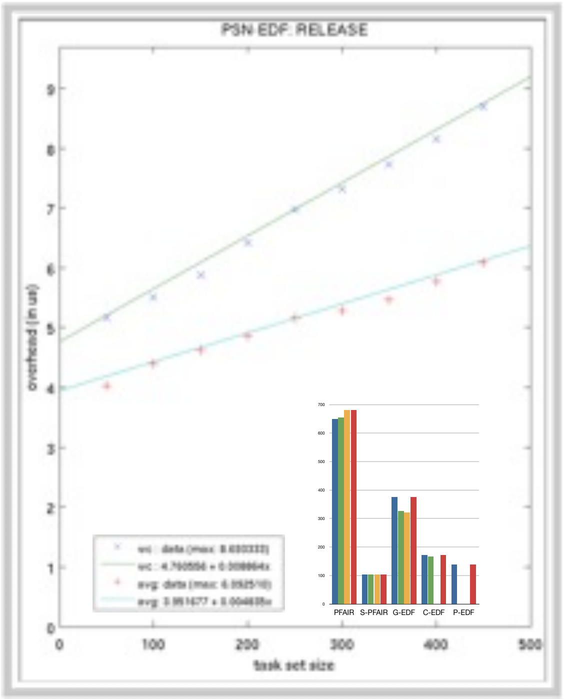
LITMUS^RT

➡ UNC's Linux-based Real-Time Testbed

Sun UltraSPARC T1 "Niagara"

➡ 8 cores, 4 HW threads per core = 32 logical processors.

➡ 3 MB shared L2 cache



Overheads

➡ Traced overheads under each of the plugins.

➡ Collected more than 640,000,000 samples (total).

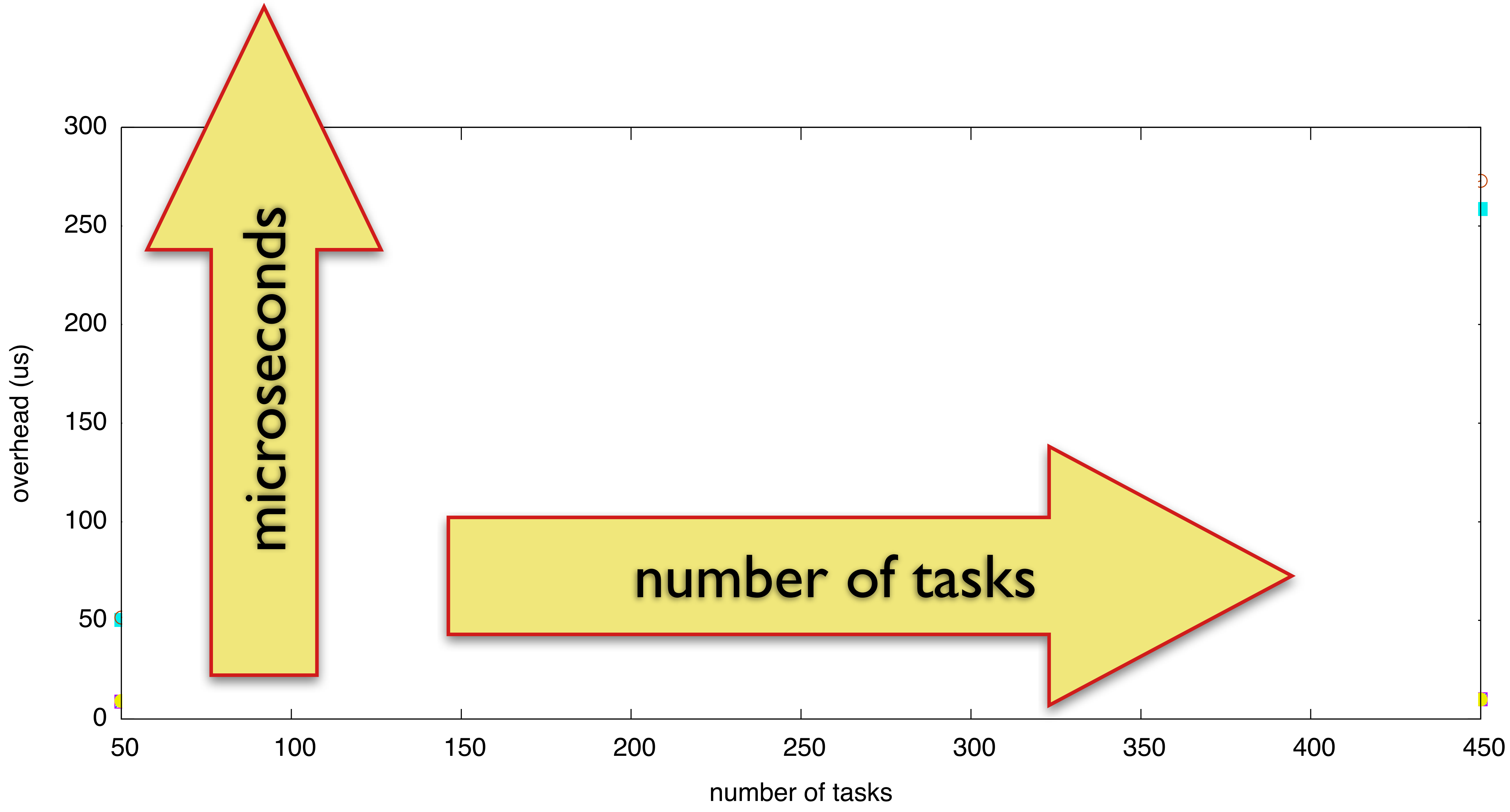
➡ Computed worst-case and average-case overheads.

➡ Over 20 graphs; see online version.

Outliers

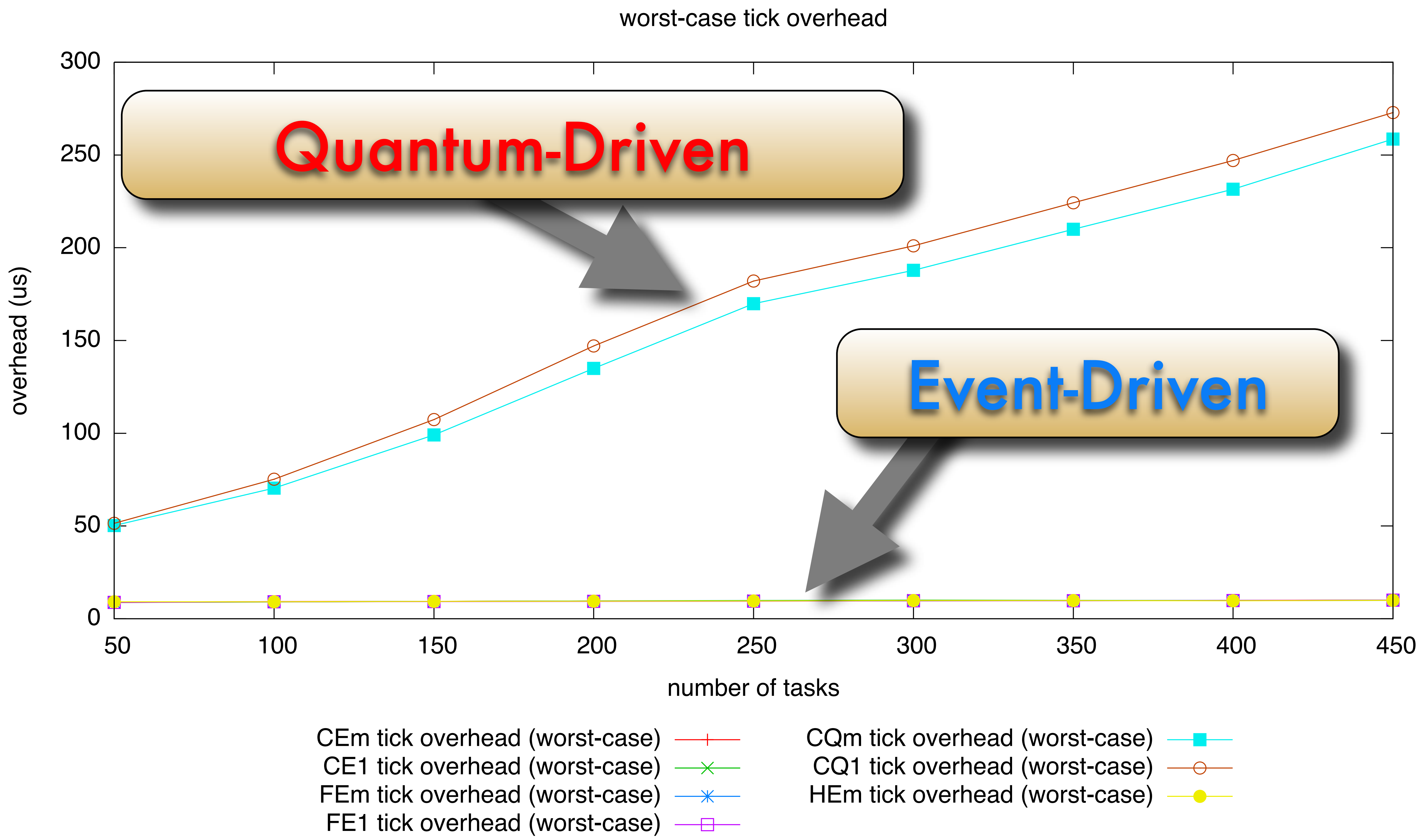
➡ Removed top 1% of samples to discard outliers.

Example: Tick Overhead

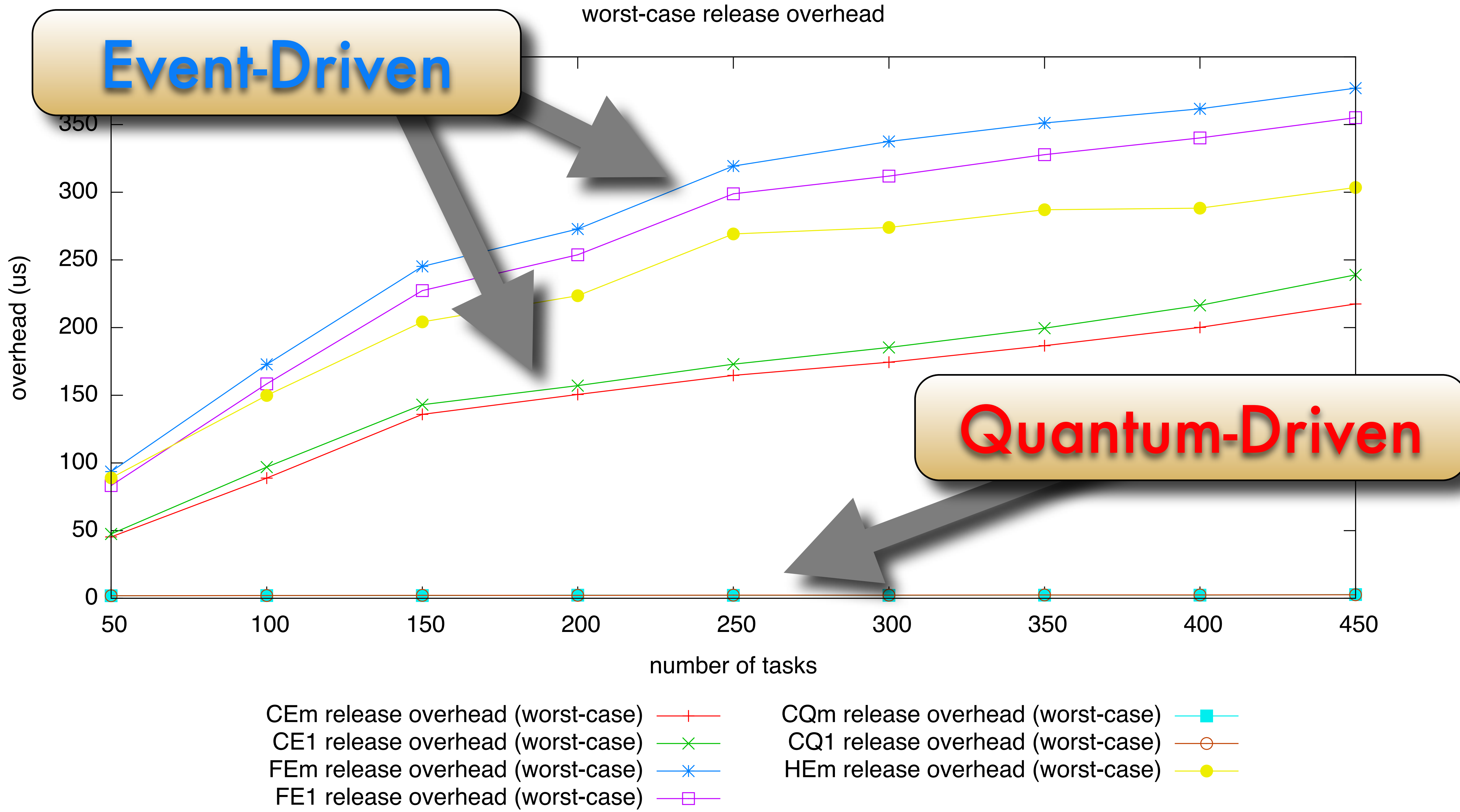


"Higher is worse."

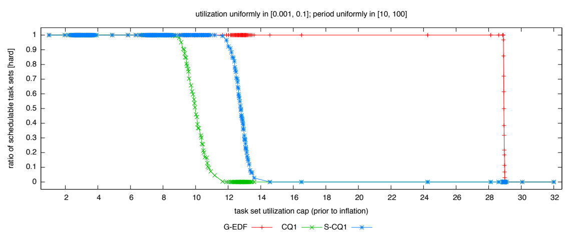
Example: Tick Overhead

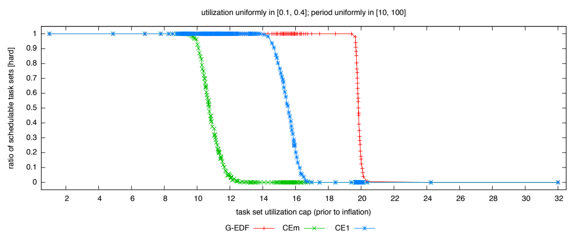


Example: Release Overhead



Study Setup

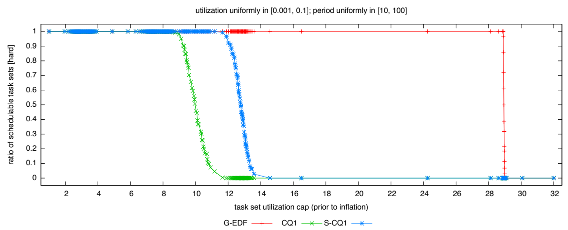


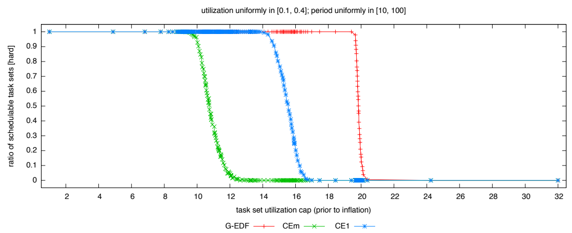


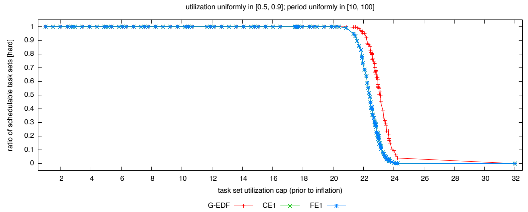
Methodology.

- ➡ Randomly generate task set.
- ➡ Apply overheads (for each G-EDF implementation).
- ➡ Test whether task set can be claimed schedulable (for each G-EDF implementation).

Study Setup







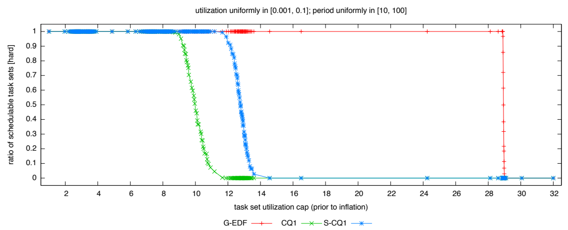
Methodology.

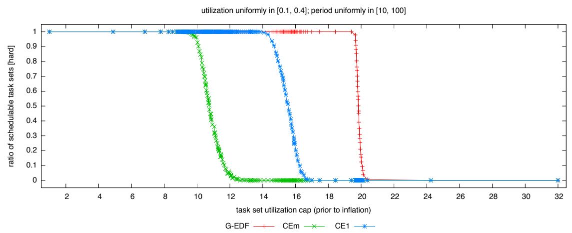
- ➡ Randomly generate task set.
- ➡ Apply overheads (for each G-EDF implementation).
- ➡ Test whether task set can be claimed schedulable (for each G-EDF implementation).

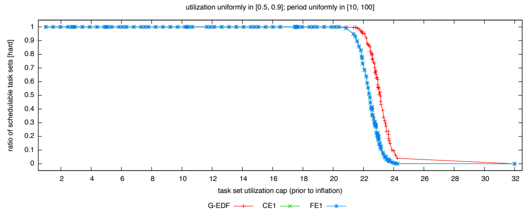
Schedulability.

- ➡ Hard real-time: worst-case overheads, no tardiness.
- ➡ Soft real-time: average-case overheads, bounded tardiness.

Study Setup







Methodology.

- ➡ Randomly generate task set.
- ➡ Apply overheads (for each G-EDF implementation).
- ➡ Test whether task set can be claimed schedulable (for each G-EDF implementation).

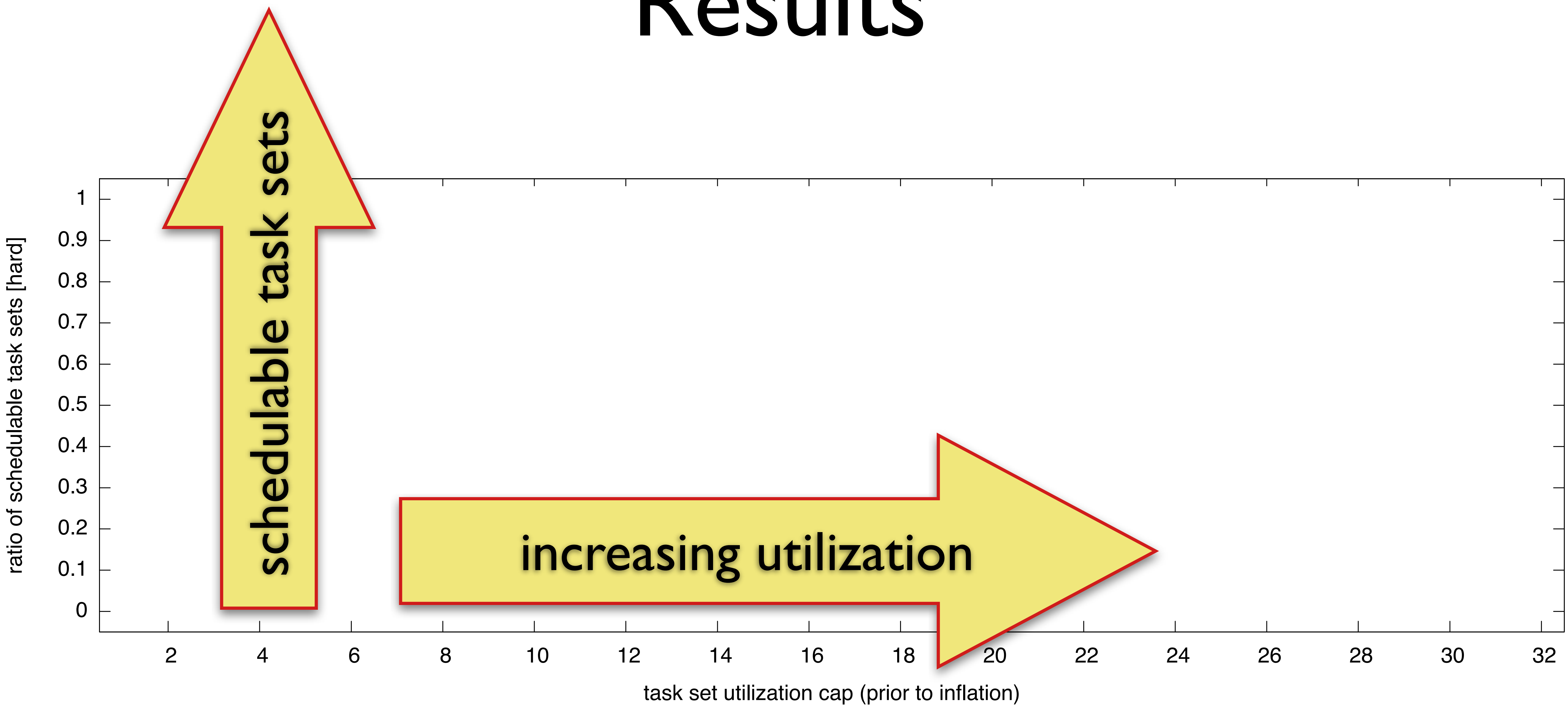
Schedulability.

- ➡ Hard real-time: worst-case overheads, no tardiness.
- ➡ Soft real-time: average-case overheads, bounded tardiness.

Task set generation.

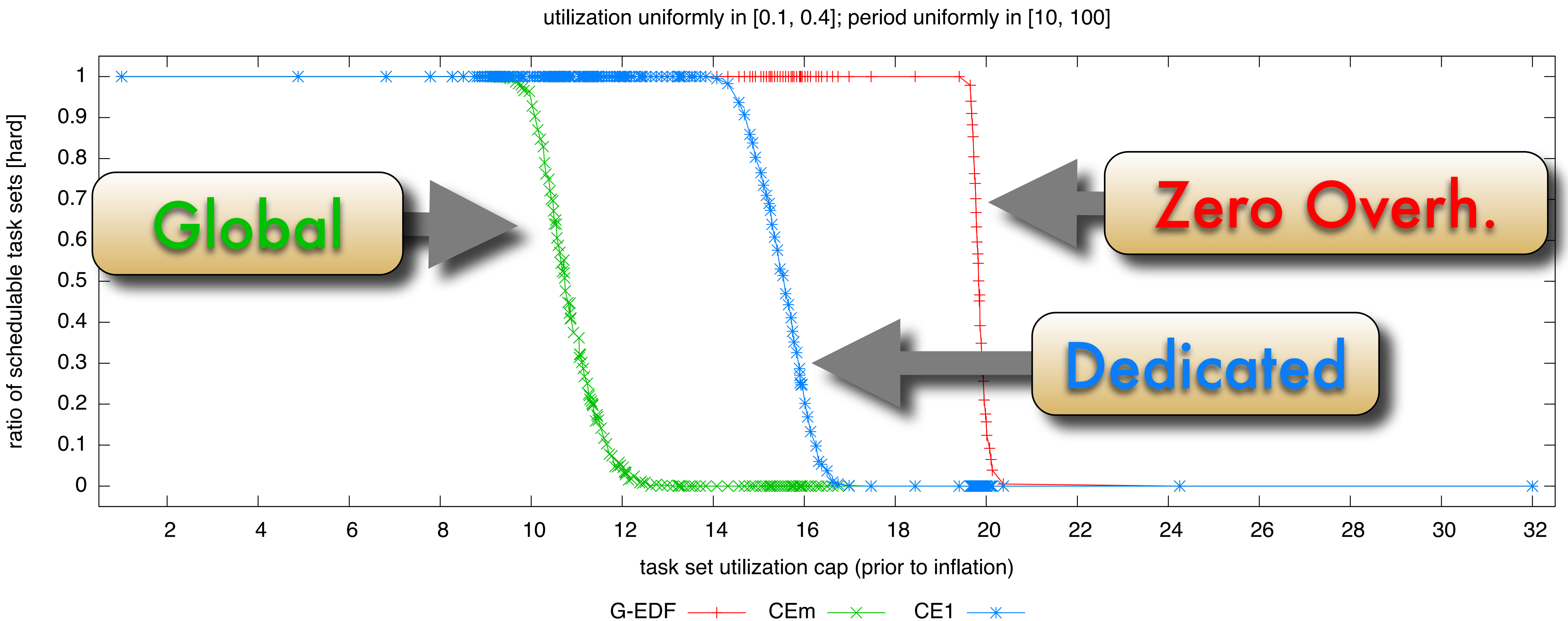
- ➡ Six utilization distributions (uniform and bimodal).
- ➡ Three period distributions (uniform).
- ➡ Over 300 graphs; see online version.

Results



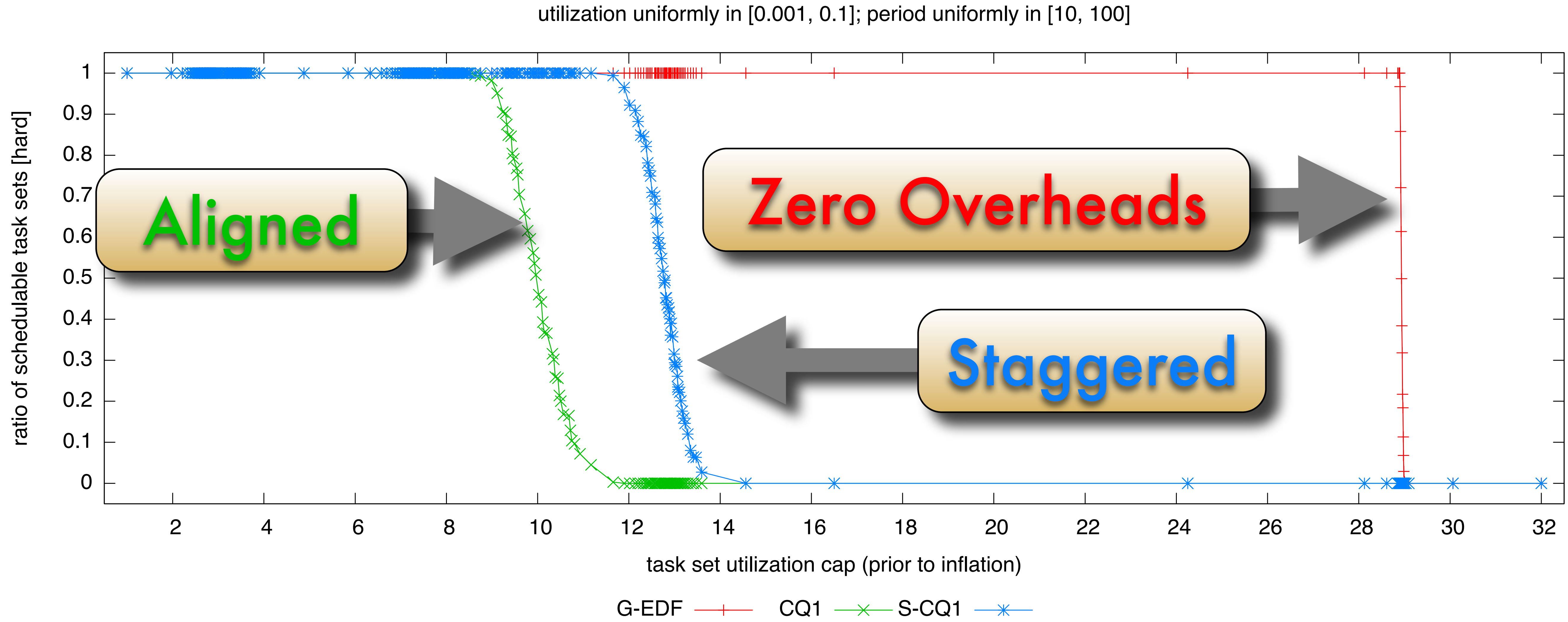
"Higher is better."

Interrupt Handling



Dedicated interrupt handling was generally preferable (or no worse).

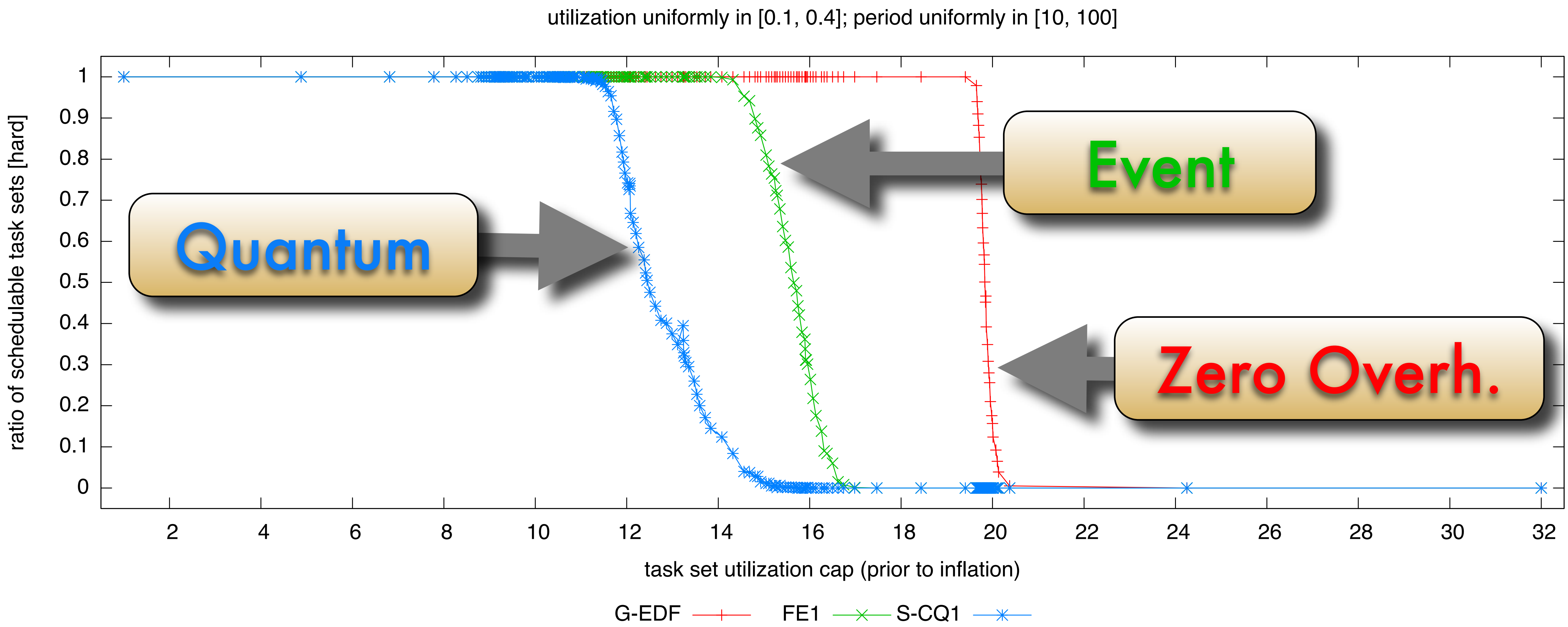
Quantum Staggering



Staggered quanta

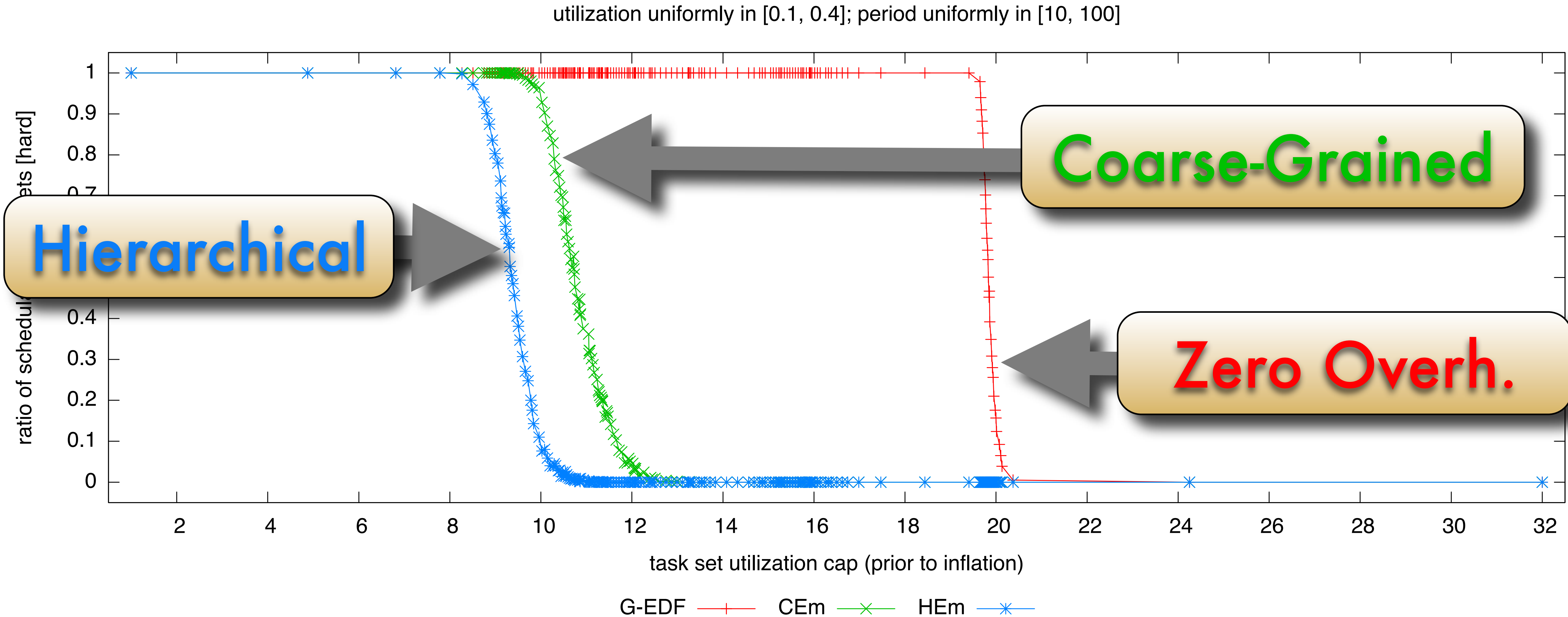
were generally preferable (or no worse).

Quantum- vs. Event-Driven



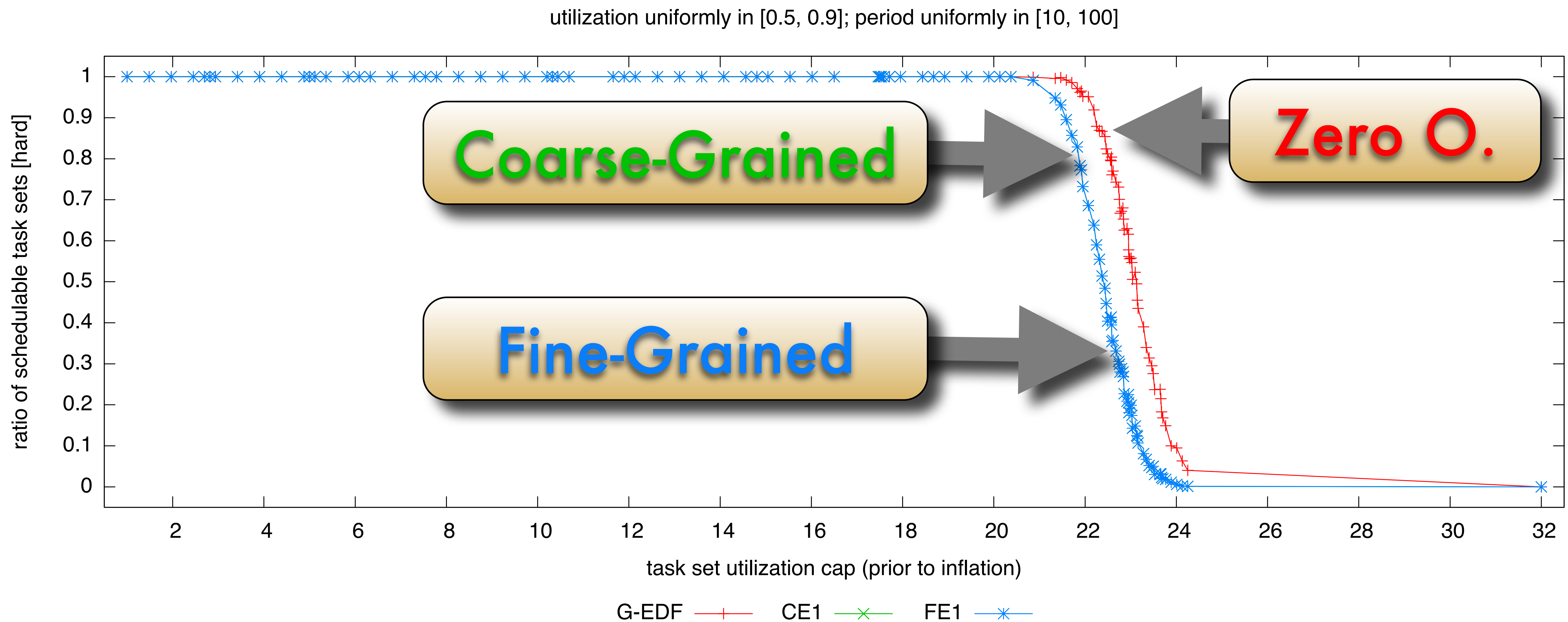
Event-driven scheduling
was preferable in most cases.

Choice of Ready Queue (1)



The **coarse-grained ready queue** performed better than the hierarchical queue.

Choice of Ready Queue (II)



The fine-grained ready queue
performed marginally better than the coarse-grained queue
if used together with **dedicated interrupt handling**.

Conclusion

Summary of Results

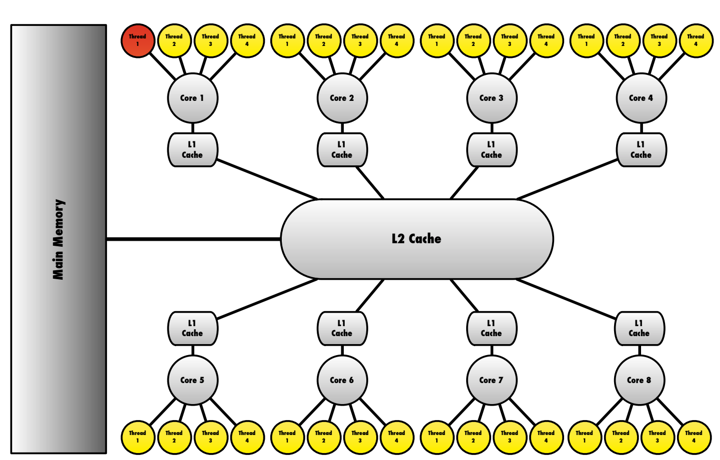
Implementation choices
can impact schedulability as much as
scheduling-theoretic tradeoffs.

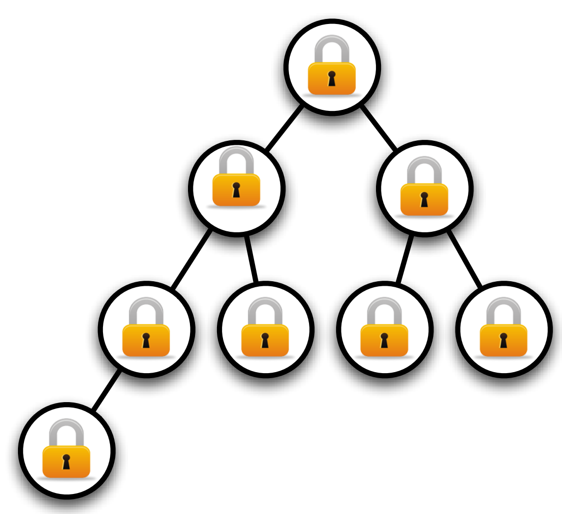
Unless task counts are very high
or periods very short,
G-EDF **can scale** to 32 processors.

Recommendation

Best results obtained with combination of:

fine-grained heap
event-driven scheduling
dedicated interrupt handling





Future Work

Platform.

➡ Repeat study on embedded hardware platform.

Implementation.

➡ Simplify locking requirements.

➡ Parallel mergeable heaps?

Analysis.

➡ Less pessimistic hard real-time G-EDF schedulability tests.

➡ Less pessimistic interrupt accounting.

Thank you!

LITMUS RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

available at

http://www.cs.unc.edu/~anderson/litmus-rt