A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications

ECRTS'13
July 12, 2013

Max
Planck
Institute
for
Software Systems

Björn B. Brandenburg
bbb@mpi-sws.org

A Rhetorical Question

On uniprocessors, why do we use

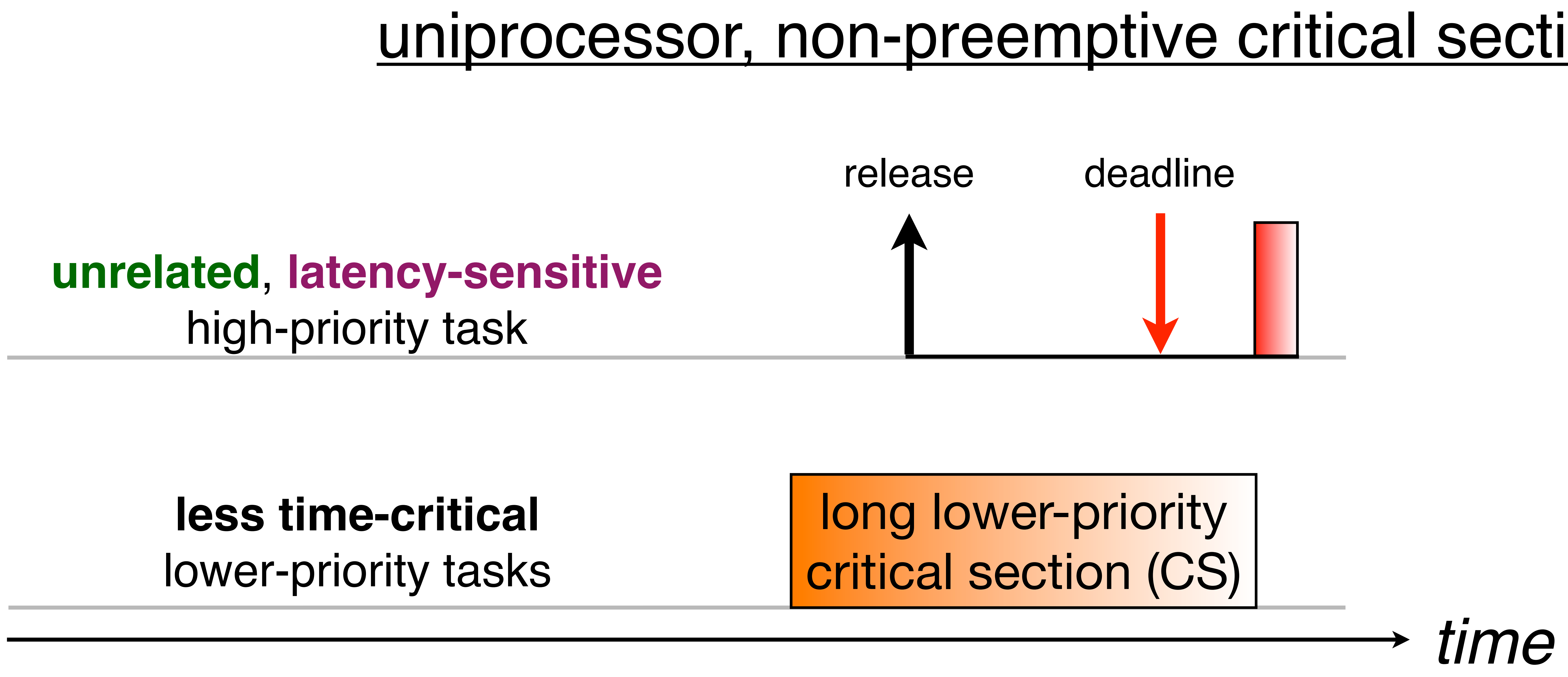
*the **priority inheritance protocol (PIP)***
*or the **priority ceiling protocol (PCP)***

*instead of **simple non-preemptive sections**?*

AUTOSAR Non-Preemptive Critical Section:

```
SuspendAllInterrupts(…);
// critical section
ResumeAllInterrupts(…);
```

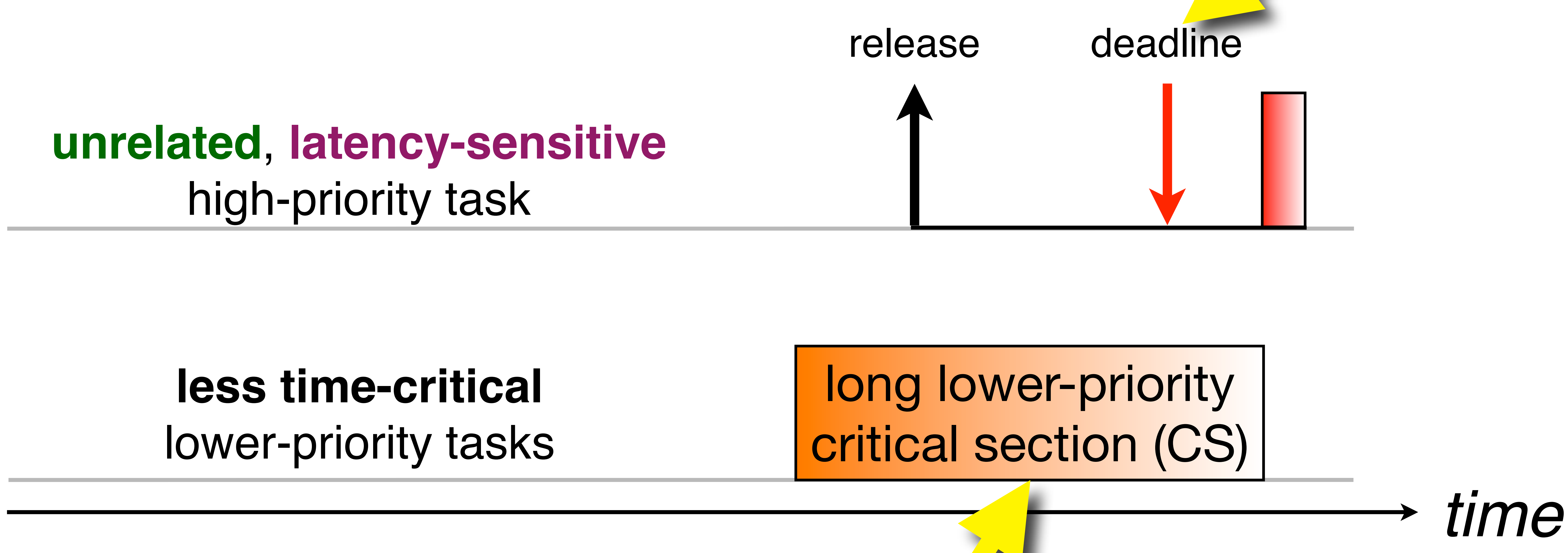
RT 101: Preemptive Synchronization Matters



RT 101: Preemptive Synchronization Matters

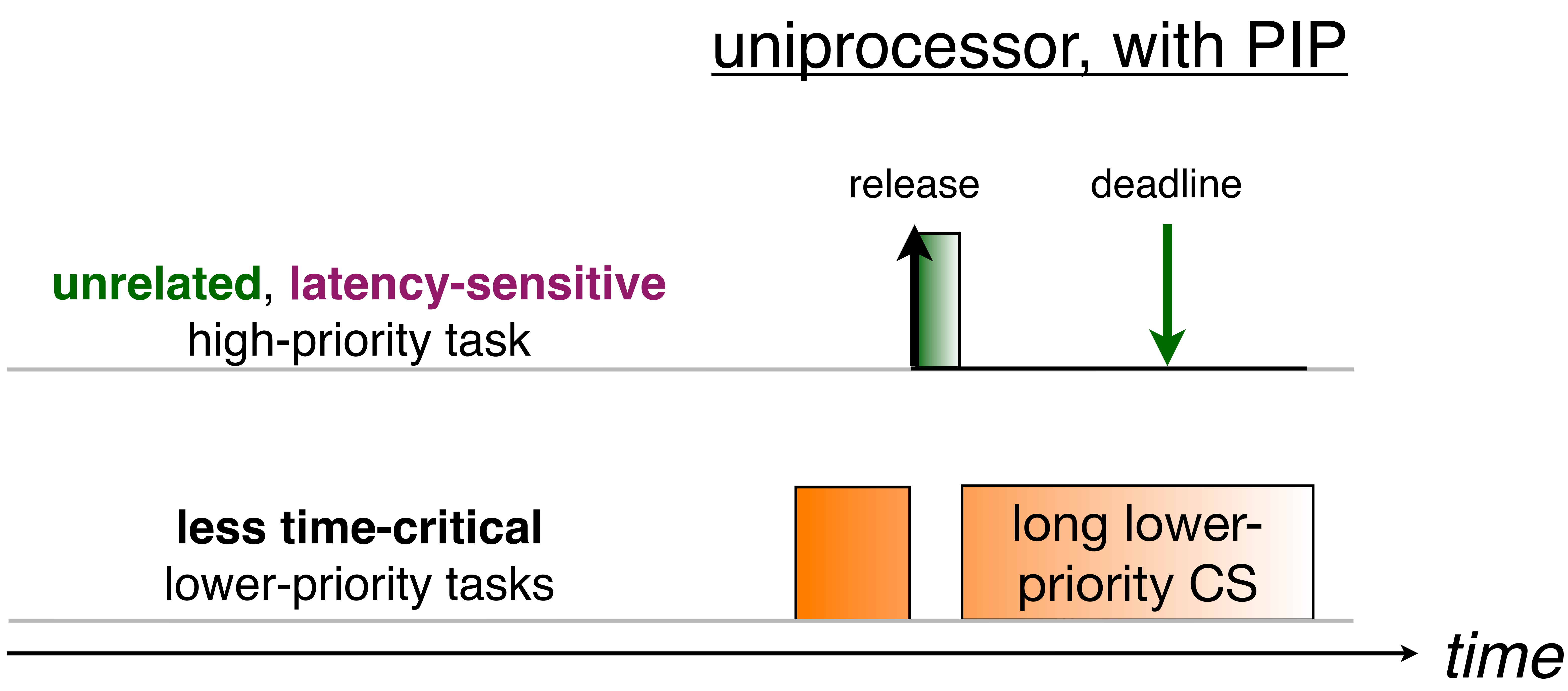
Deadline miss due to **latency increase!**

uniprocessor, non-preemptive critical sections

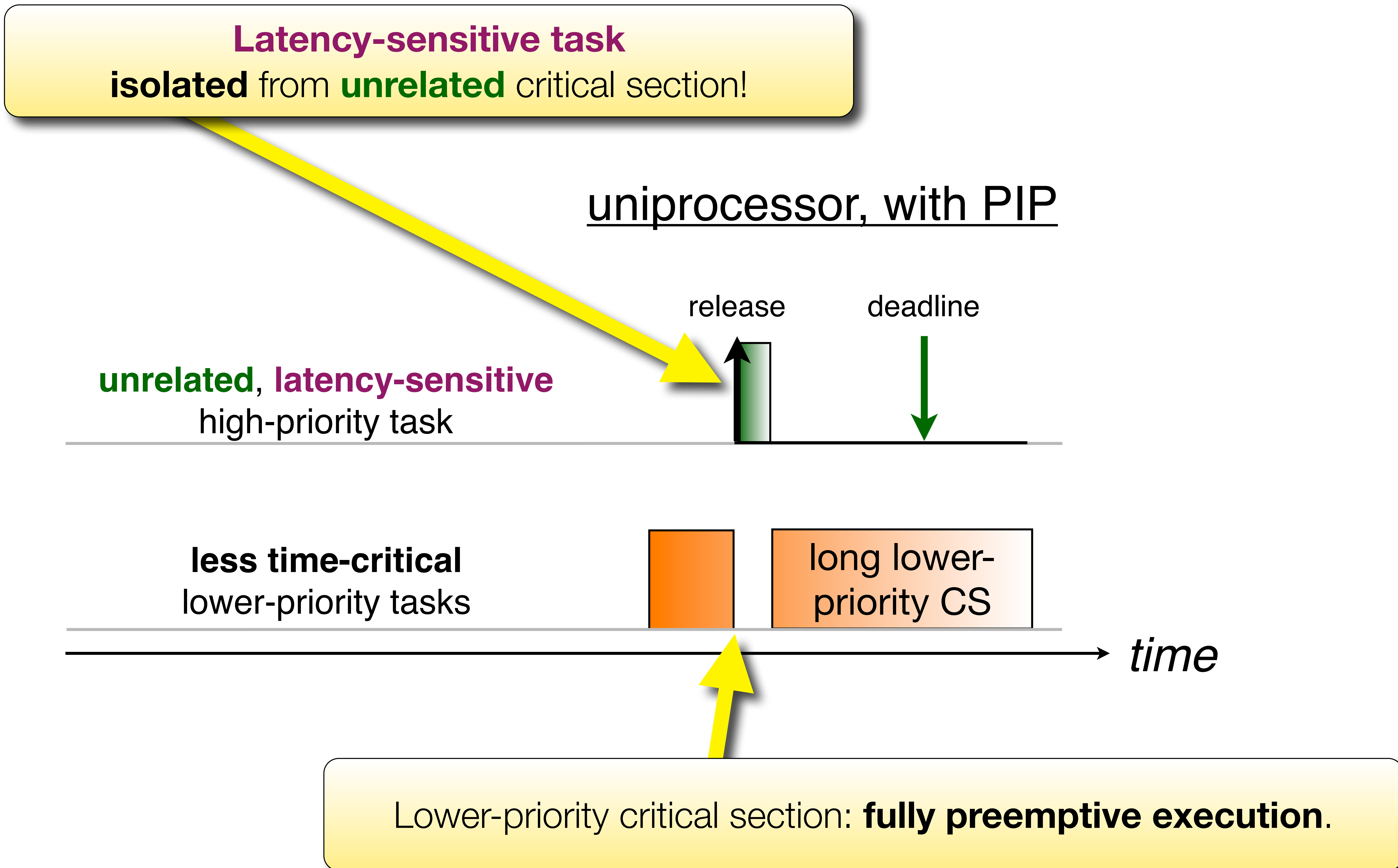


Long **non-preemptive** critical section.

RT 101: Preemptive Synchronization Matters



RT 101: Preemptive Synchronization Matters



The Multiprocessor Case

What if we host the same workload on a multiprocessor?

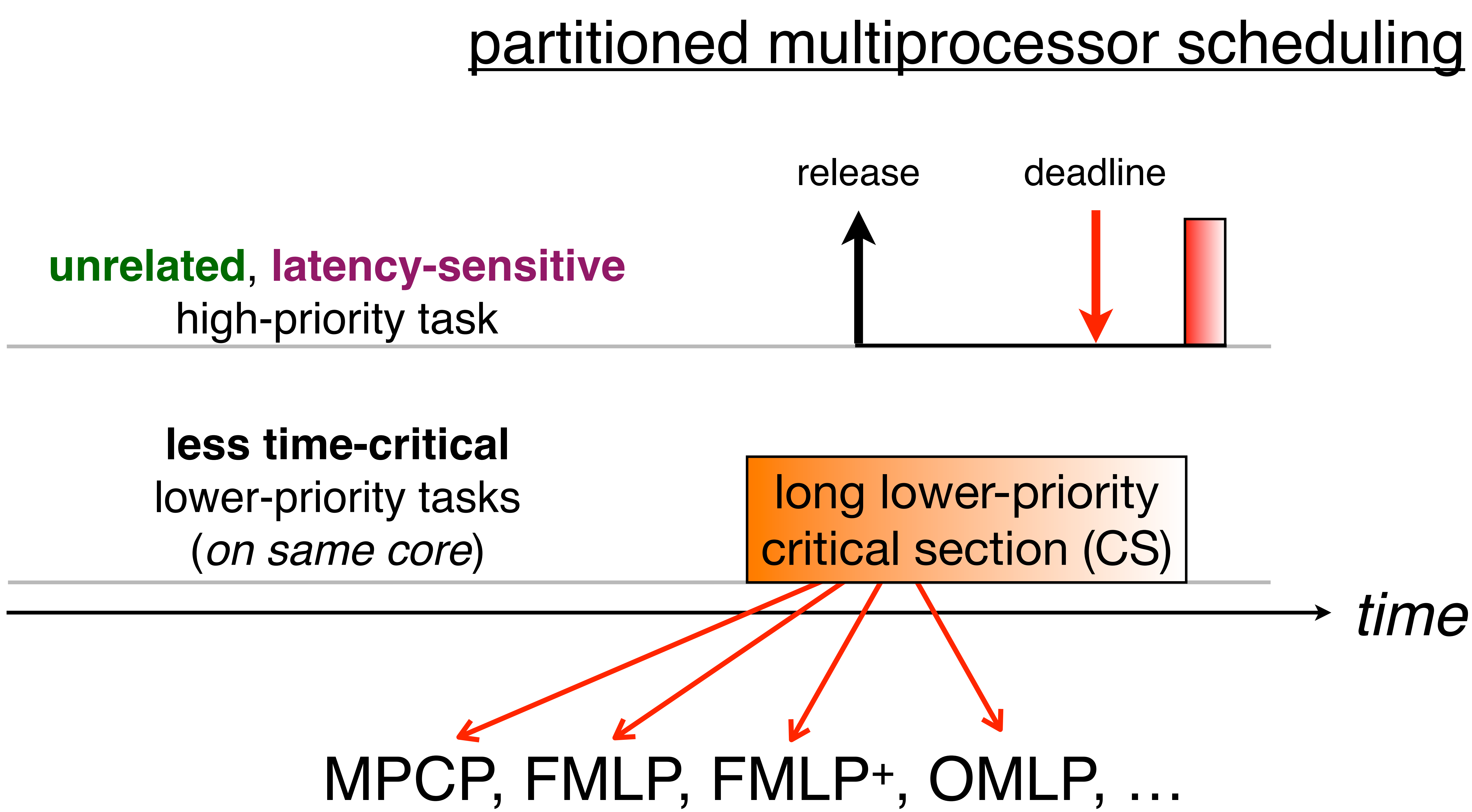
partitioned multiprocessor scheduling

unrelated, **latency-sensitive**
high-priority task

less time-critical
lower-priority tasks
(*on same core*)

time

No existing real-time semaphore protocol for partitioned or clustered scheduling isolates high-priority tasks from unrelated CSs.



This Paper

Independence preservation formalizes the idea that "*tasks should never be delayed by unrelated critical sections.*"

This Paper

Independence preservation formalizes the idea that "*tasks should never be delayed by unrelated critical sections.*"

Independence preservation is **impossible without (limited) job migrations**.

This Paper

Independence preservation formalizes the idea that "*tasks should never be delayed by unrelated critical sections.*"

Independence preservation is **impossible** **without (limited) job migrations**.

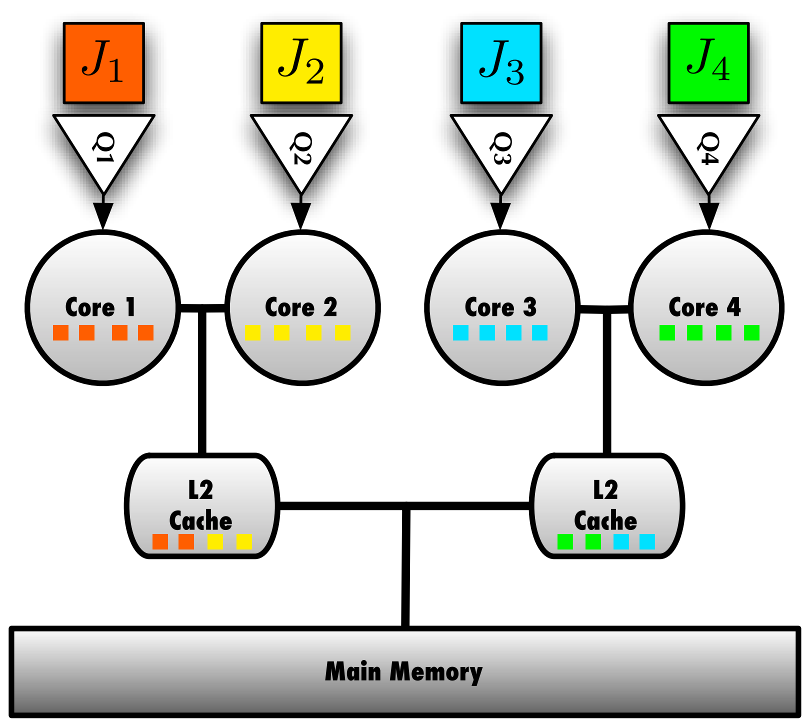
First independence-preserving semaphore protocol for clustered/partitioned scheduling; the protocol also has ***asymptotically optimal blocking bounds***.

Clustered JLFP Scheduling

*Job-Level Fixed-Priority Scheduling (**JLFP**)*

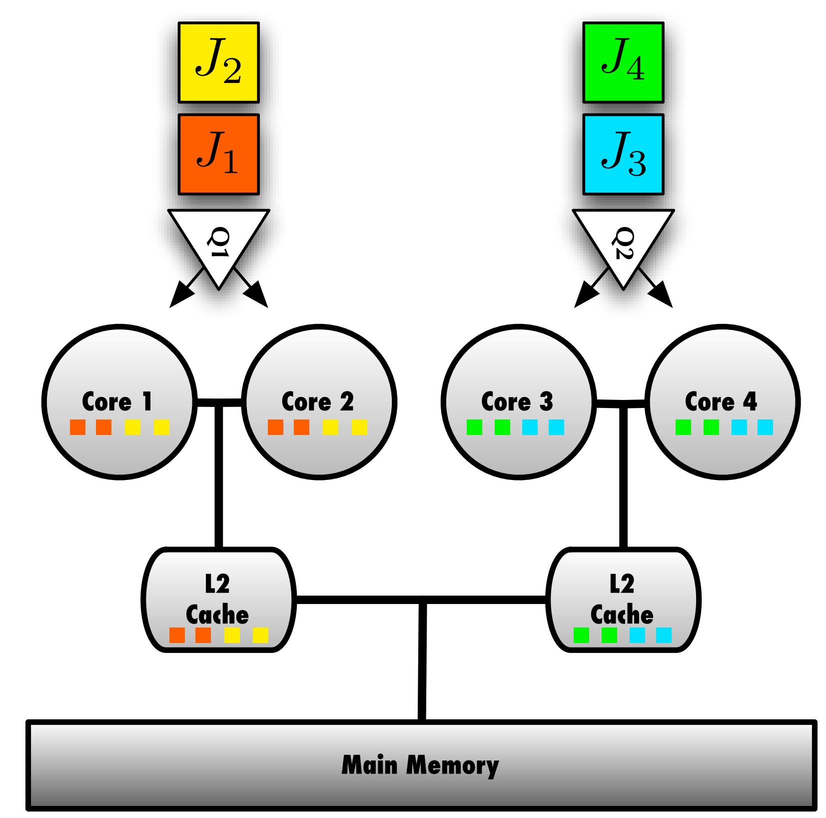
c *… number of processors per cluster*

m *… number of processors (total)*



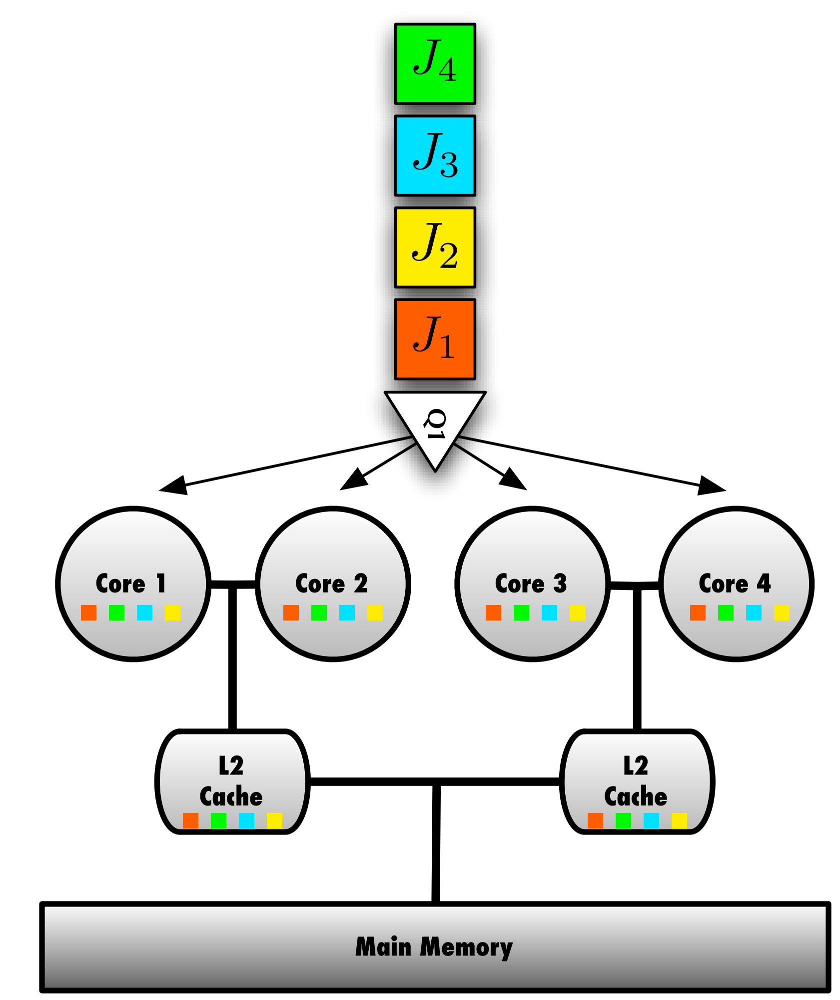
partitioned scheduling

c **= 1**



clustered scheduling

1 ≤ *c* ≤ *m*



global scheduling

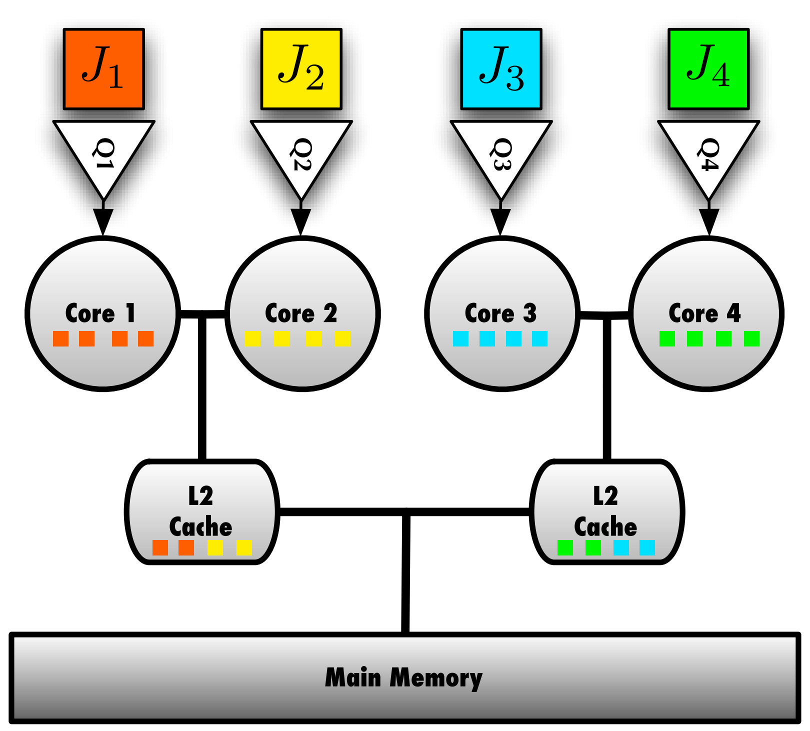
c **=** ***m***

This talk: Partitioned Fixed-Priority (P-FP) Scheduling

*Job-Level Fixed-Priority Scheduling (**JLFP**)*

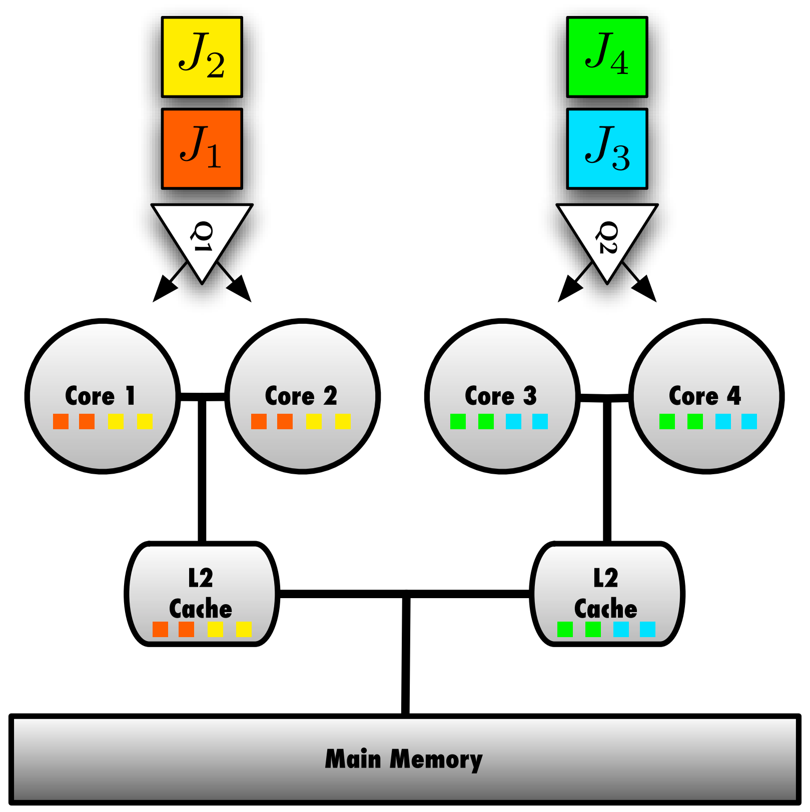
c *… number of processors per cluster*

m *… number of processors (total)*



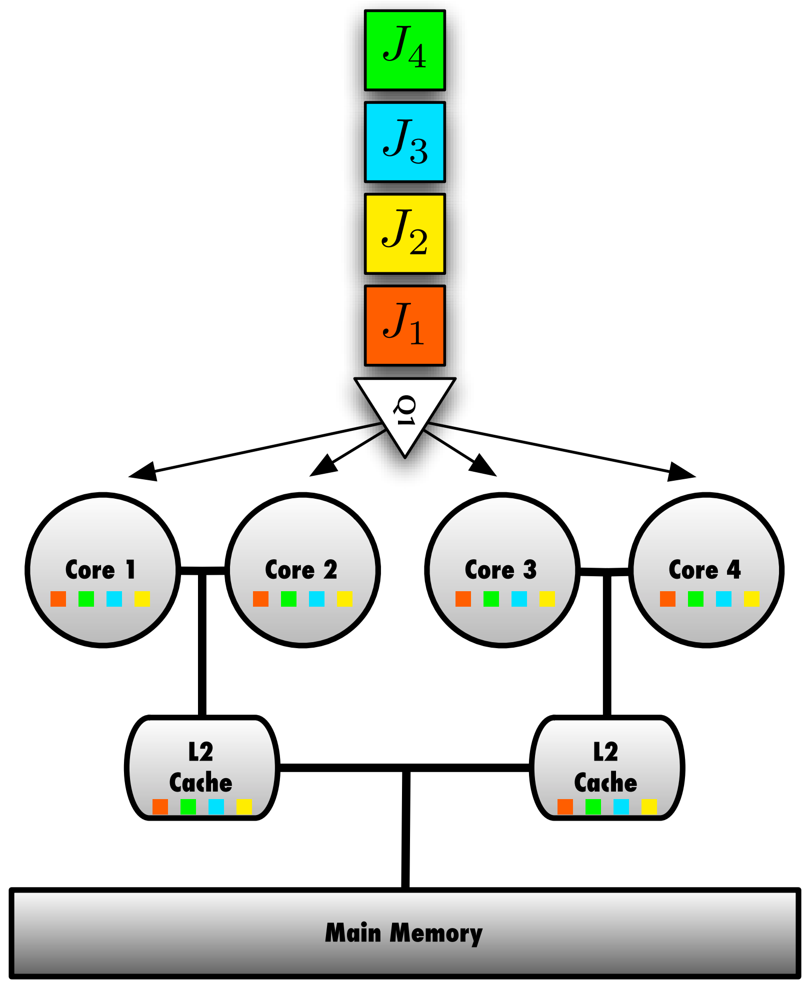
partitioned scheduling

c **= 1**



clustered scheduling

1 ≤ *c* ≤ *m*



global scheduling

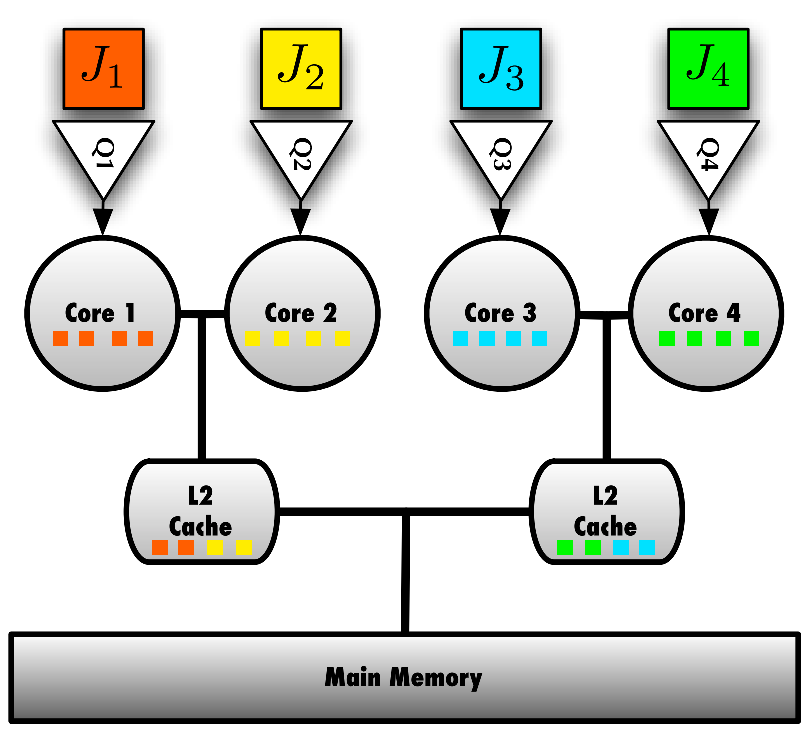
c **=** ***m***

Clustered JLFP Scheduling

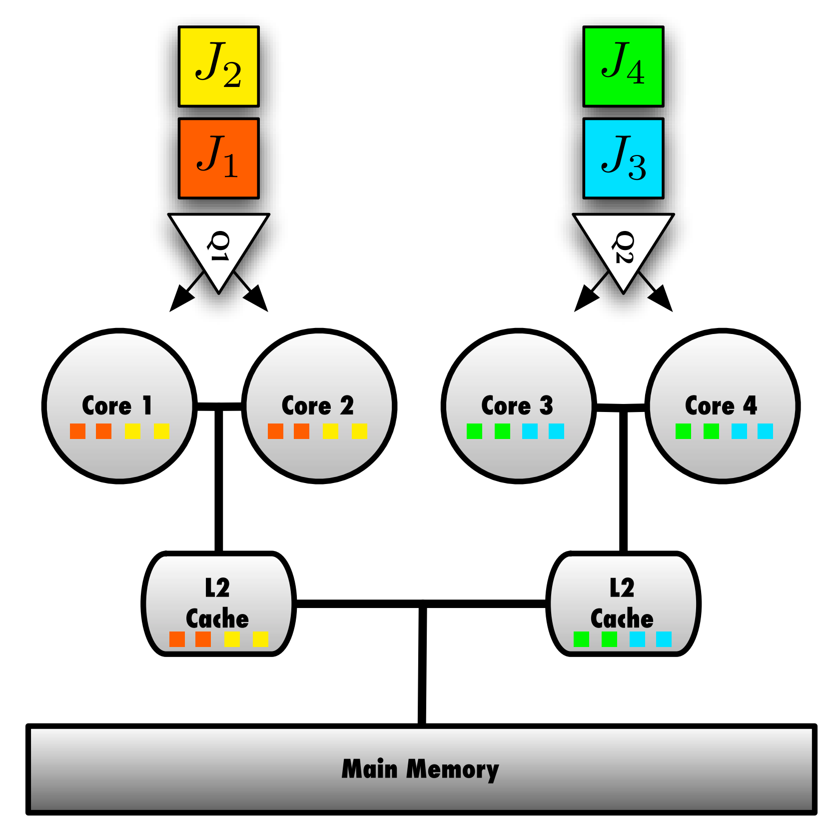
*Job-Level Fixed-Priority Scheduling (**JLFP**)*

***c** … number of processors per cluster*

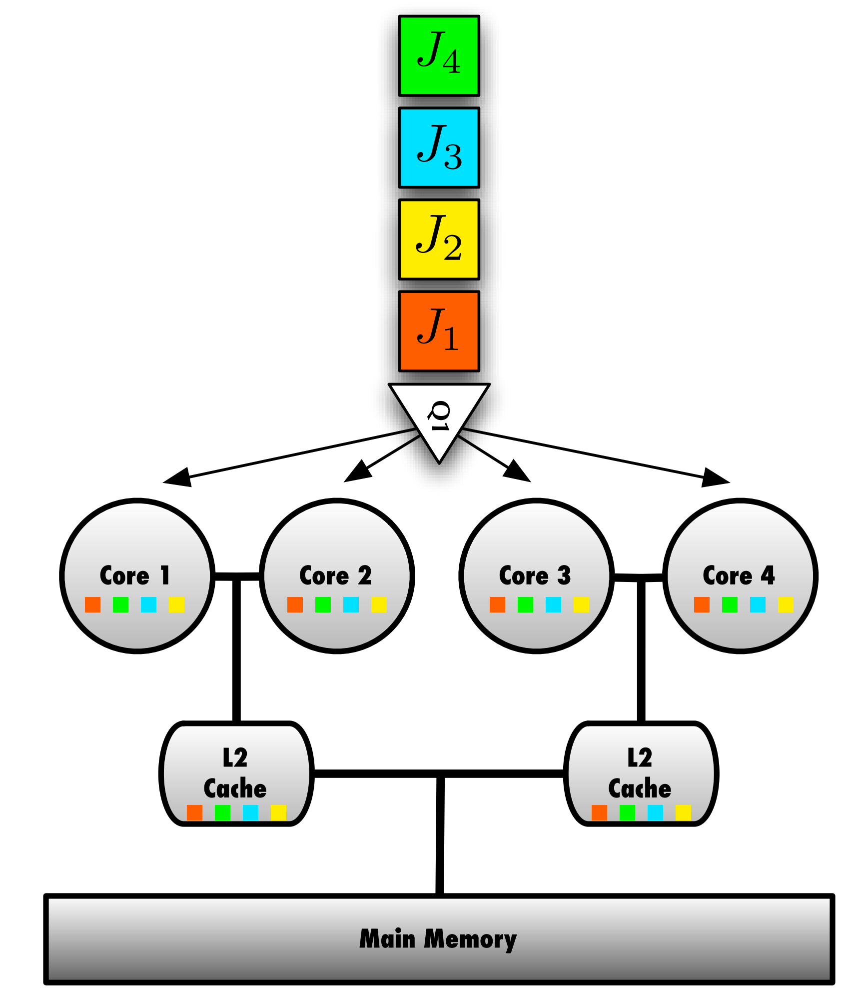
***m** … number of processors (total)*



partitioned scheduling



clustered scheduling



global scheduling

Task model: implicit-deadline sporadic tasks
(choice of deadline constraint irrelevant to results)

Real-Time Semaphore Protocols

Binary Semaphores in POSIX

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

*A blocked task **suspends** & yields the processor.*

Real-Time Semaphore Protocols

Binary Semaphores in POSIX

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

*A blocked task **suspends** & yields the processor.*

Priority Inversion

*A job **should** be scheduled, but **is not**.*

PI-Blocking: increase in worst-case response time due to priority inversions.

Real-Time Semaphore Protocols

Binary Semaphores in POSIX

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

*A blocked task **suspends** & yields the processor.*

Priority Inversion

*A job **should** be scheduled, but **is not**.*

PI-Blocking: increase in worst-case response time due to priority inversions.

Goal: **bounded pi-blocking**.

Bounded in terms of critical section lengths only!

Real-Time Semaphore Protocols

Binary Semaphores in POSIX

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

*A blocked task **suspends** & yields the processor.*

Priority Inversion

*A job **should** be scheduled, but **is not**.*

PI-Blocking: increase in worst-case response time due to priority inversions.

Assumptions

➡ Unnested critical sections.

➡ **Suspension-oblivious** schedulability analysis.

Part 1

Avoiding Delays due to Unrelated Critical Sections

Independence Preservation

(specific to s-oblivious analysis)

“Tasks should never be delayed by unrelated critical sections.”

Independence Preservation

(specific to s-oblivious analysis)

Let $b_{i,q}$ denote the **maximum pi-blocking** incurred by task T_i due to requests for resource q.

Let $N_{i,q}$ denote the maximum number of times that any job of T_i **accesses** resource q.

Under an **independence-preserving** locking protocol,

if $N_{i,q} = 0$, then $b_{i,q} = 0$.

"You only pay for what you use."

Independence Preservation

(specific to s-oblivious analysis)

Let $b_{i,q}$ denote the **maximum pi-blocking** incurred by task T_i due to requests for resource q.

Let $N_{i,q}$ denote the maximum number of times that any job of T_i **accesses** resource q.

Under an **independence-preserving** locking protocol,

if $N_{i,q} = 0$, then $b_{i,q} = 0$.

Isolation useful for:

latency-sensitive workloads (if no delay can be tolerated) or if low-priority tasks contain unknown or untrusted critical sections.

Real-Time Semaphore Protocols

real-time locking protocol	=	*progress mechanism*	+	*queue structure*

Real-Time Semaphore Protocols

real-time locking protocol = *progress mechanism* + *queue structure*

Ensure that a lock holder is scheduled (while waiting tasks incur pi-blocking).

How to **order conflicting critical sections** (e.g., priority queue, FIFO queues).

Real-Time Semaphore Protocols

real-time locking protocol = *progress mechanism* + *queue structure*

partitioned scheduling
priority boosting

clustered scheduling
priority donation

global scheduling
priority inheritance

<u>Priority boosting and Priority Donation</u>:

lock-holding jobs have **higher priority** than **non-lock-holding** jobs

→ effectively non-preemptive → not independence preserving

real-time locking protocol = *progress mechanism* + *queue structure*

<u>partitioned</u> <u>scheduling</u>
priority boosting

<u>clustered</u> <u>scheduling</u>
priority donation

<u>global</u> <u>scheduling</u>
priority inheritance

Real-Time Semaphore Protocols

real-time locking protocol = *progress mechanism* + *queue structure*

partitioned scheduling
priority boosting

clustered scheduling
priority donation

global scheduling
priority inheritance

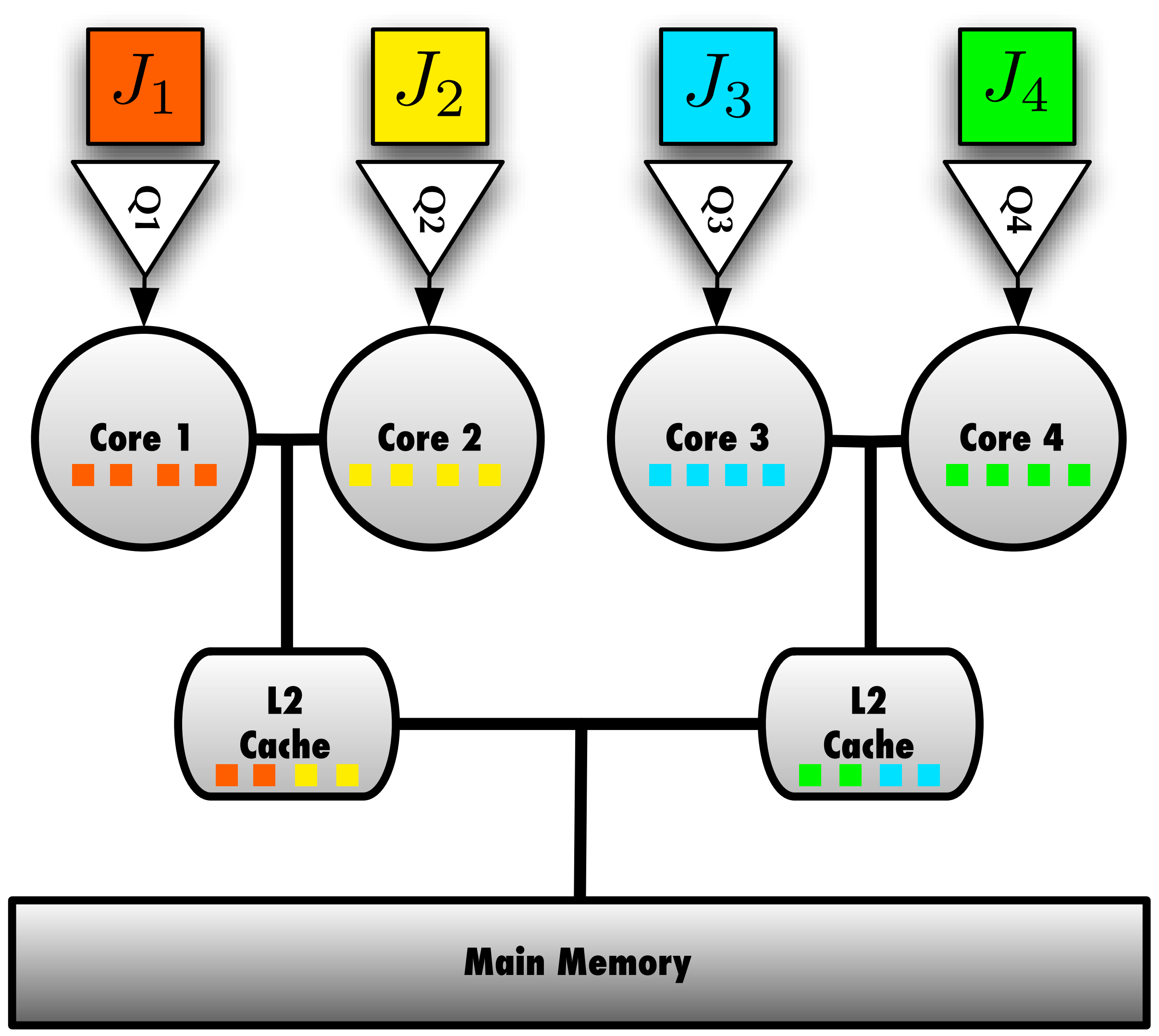
Existing independence-preserving locking protocols:

Global PIP, Global FMLP, Global OMLP, …

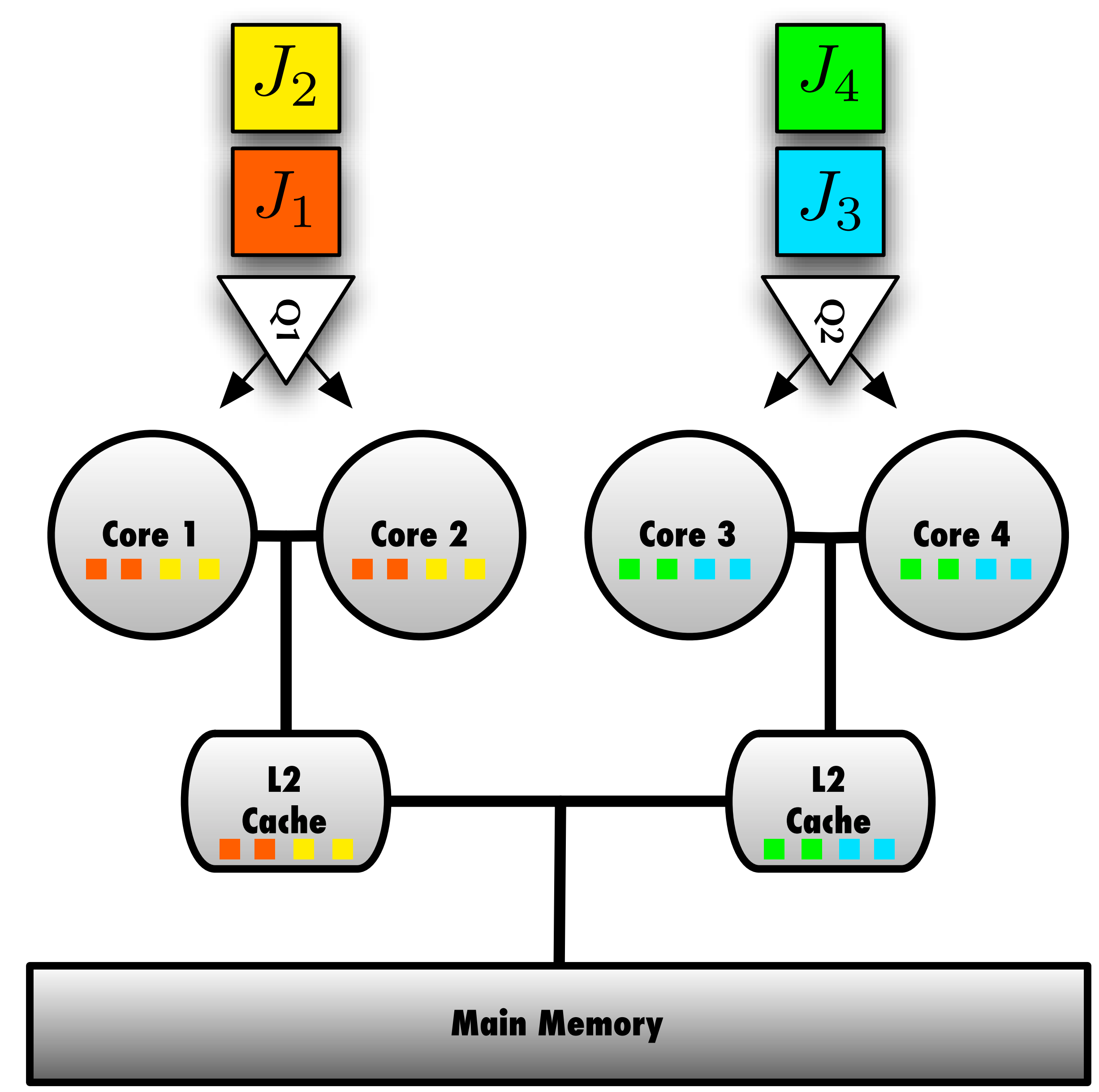
Observation

Independence preservation + bounded priority inversion

requires intra-cluster job migrations.



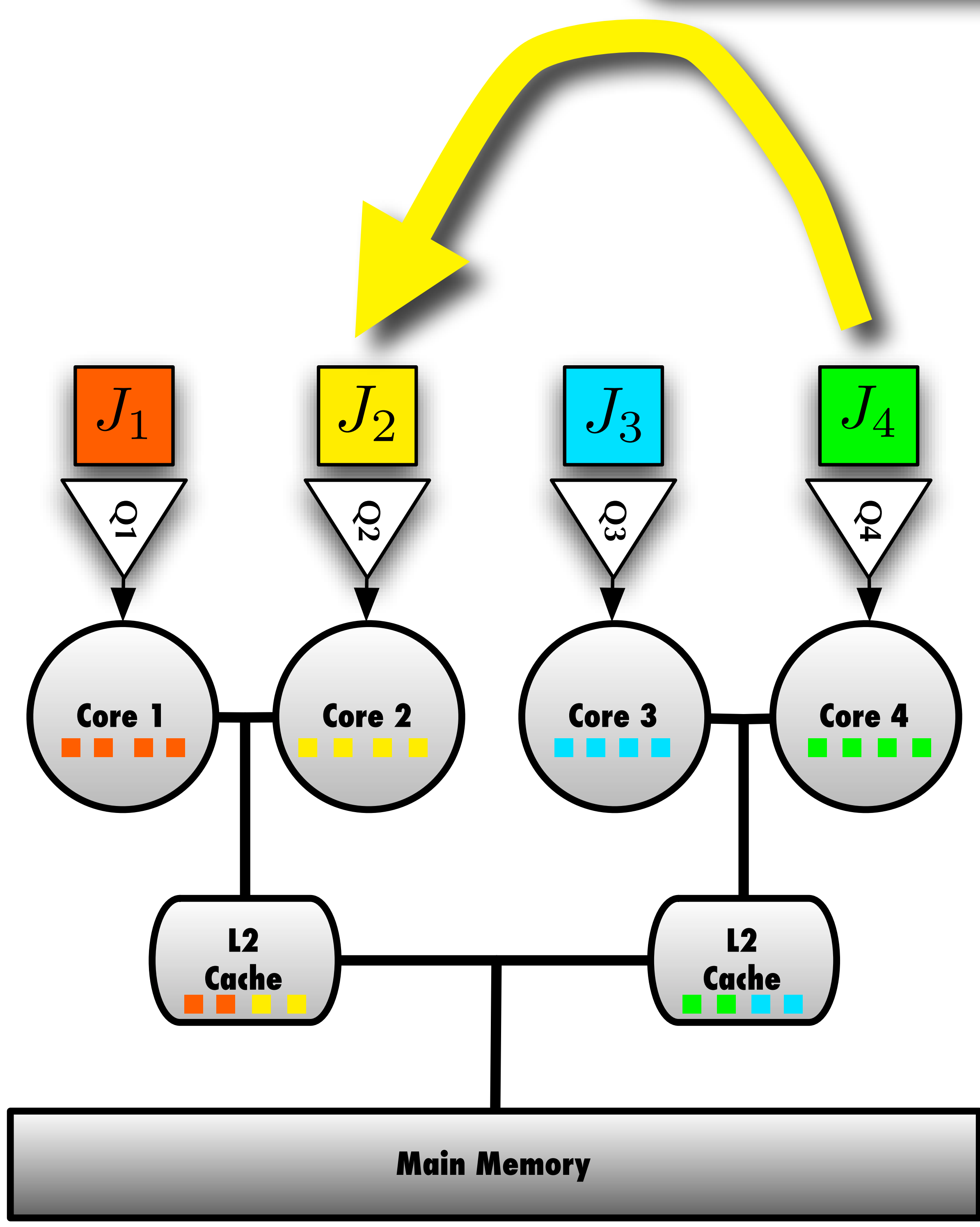
partitioned scheduling



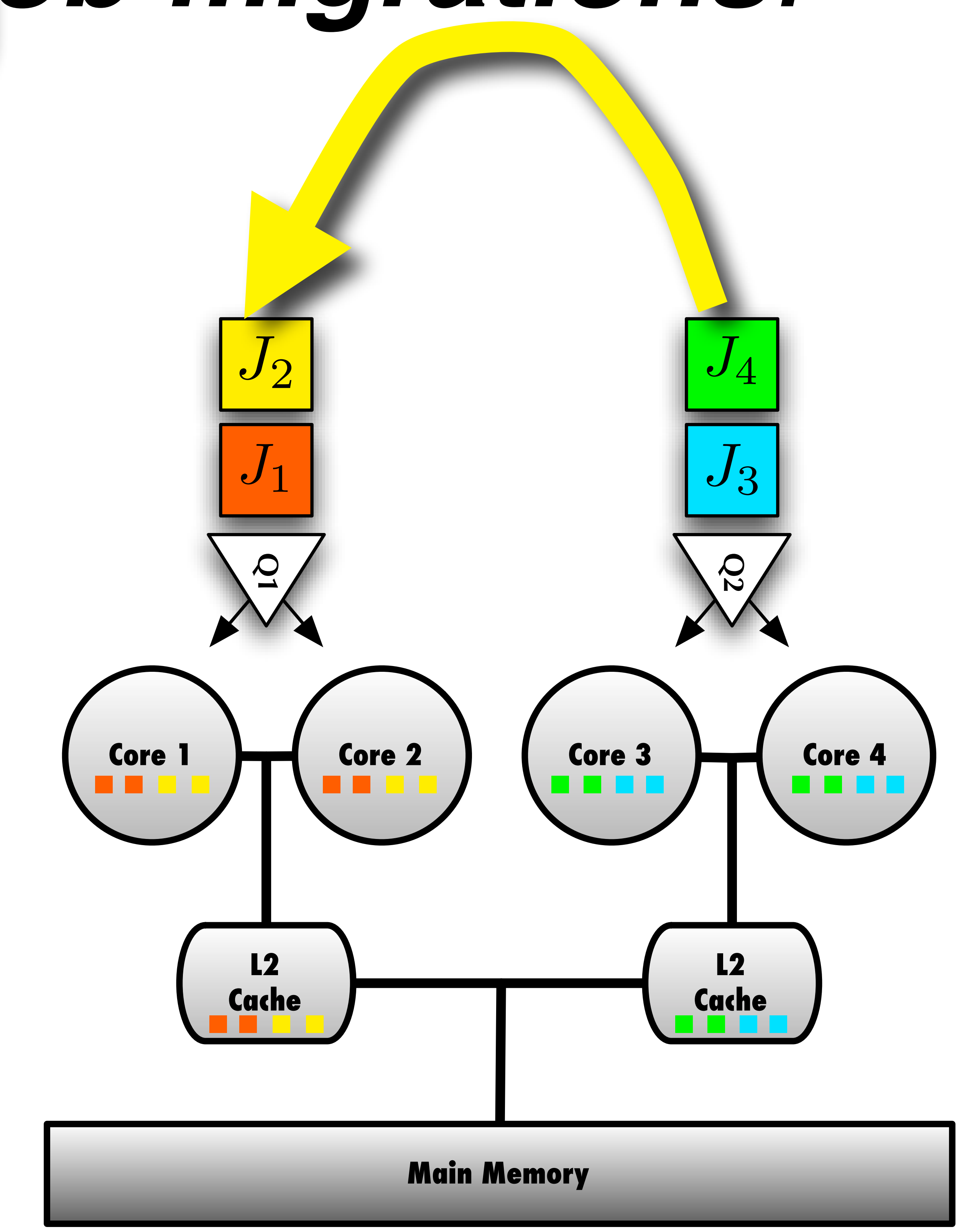
clustered scheduling

Intra-cluster: (temporarily) execute jobs on processors/clusters **they have not been assigned to**.

Independence preservation + bounded priority inversion ***requires intra-cluster job migrations.***



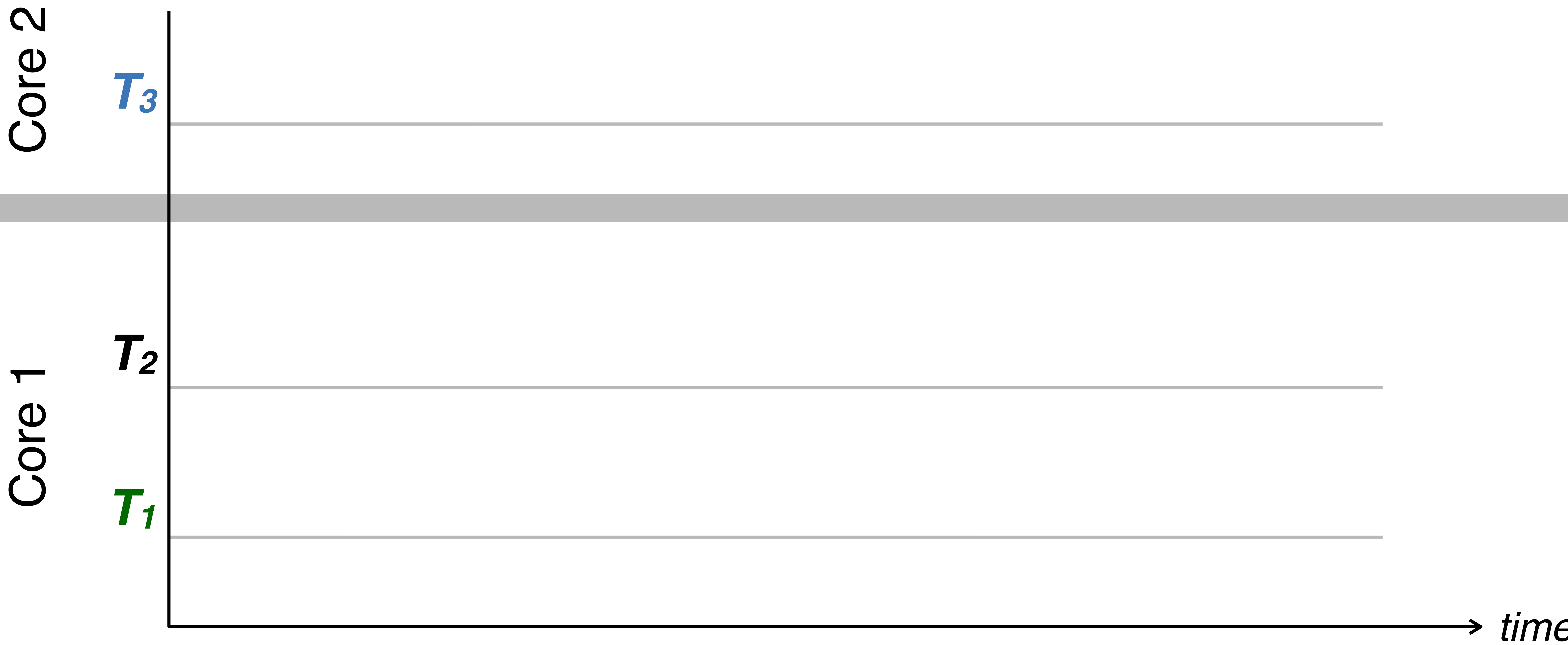
partitioned scheduling



clustered scheduling

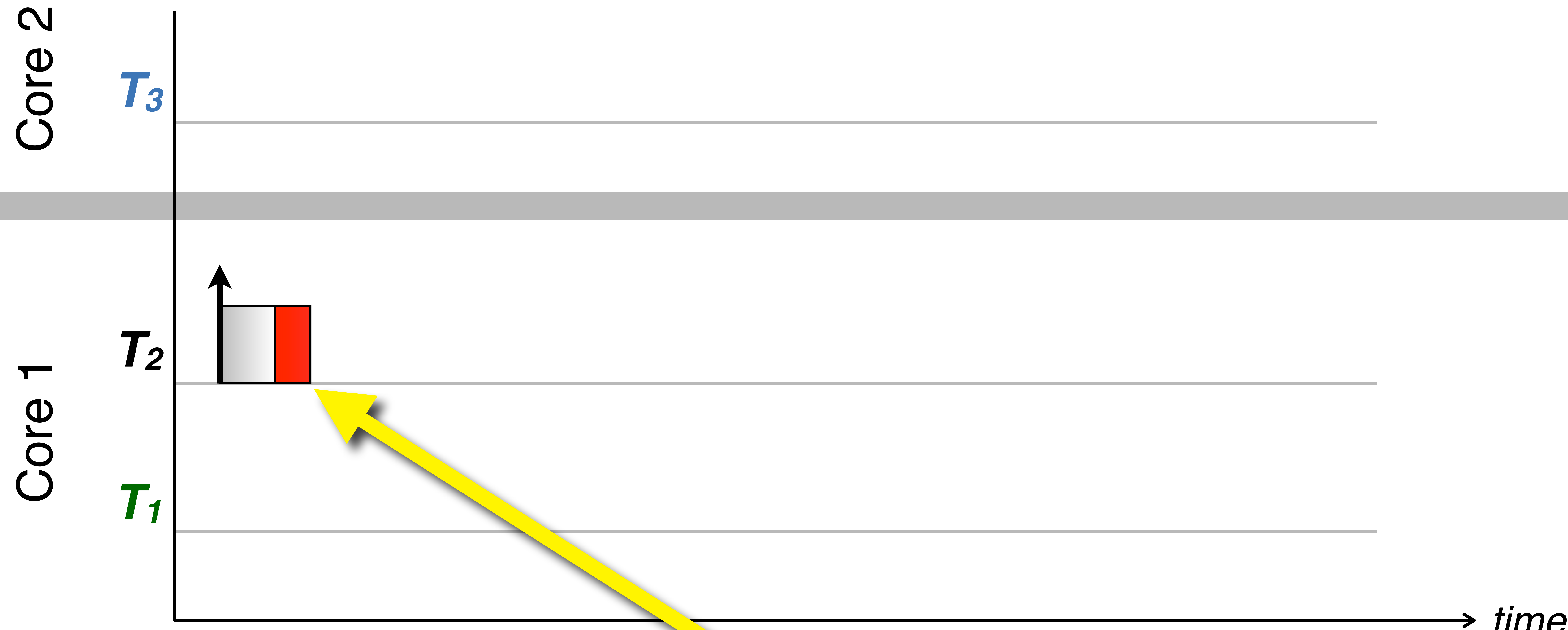
Example: Job Migration is Necessary

three tasks, two cores, one resource, P-FP scheduling



Example: Job Migration is Necessary

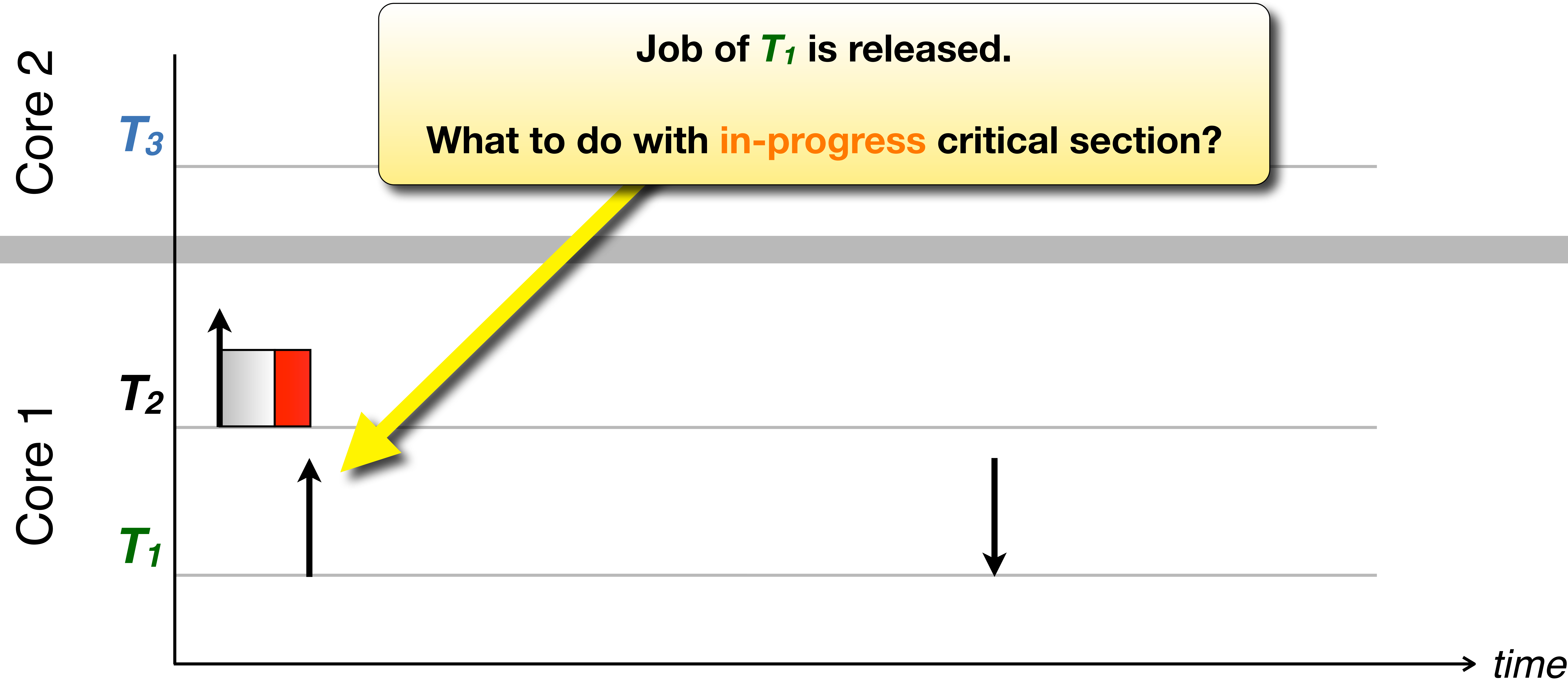
three tasks, two cores, one resource, P-FP scheduling



T_2 starts executing critical section…

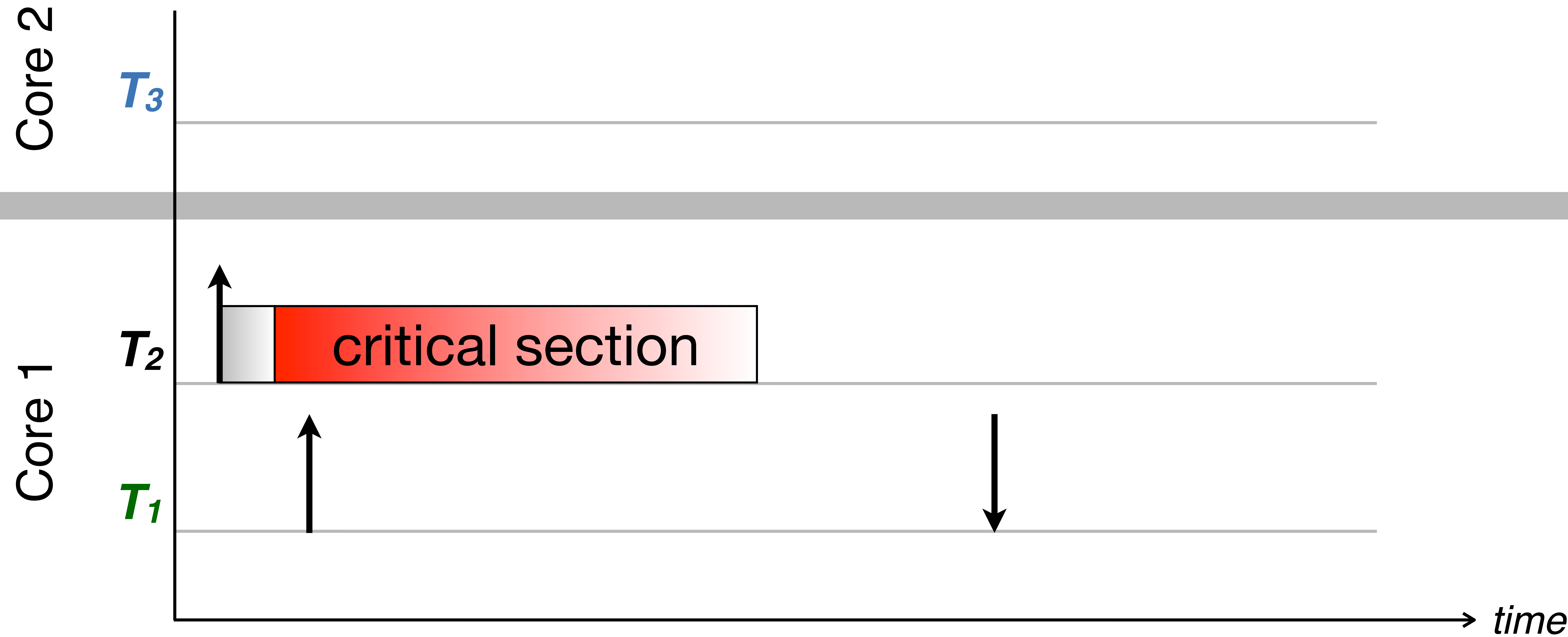
Example: Job Migration is Necessary

three tasks, two cores, one resource, P-FP scheduling



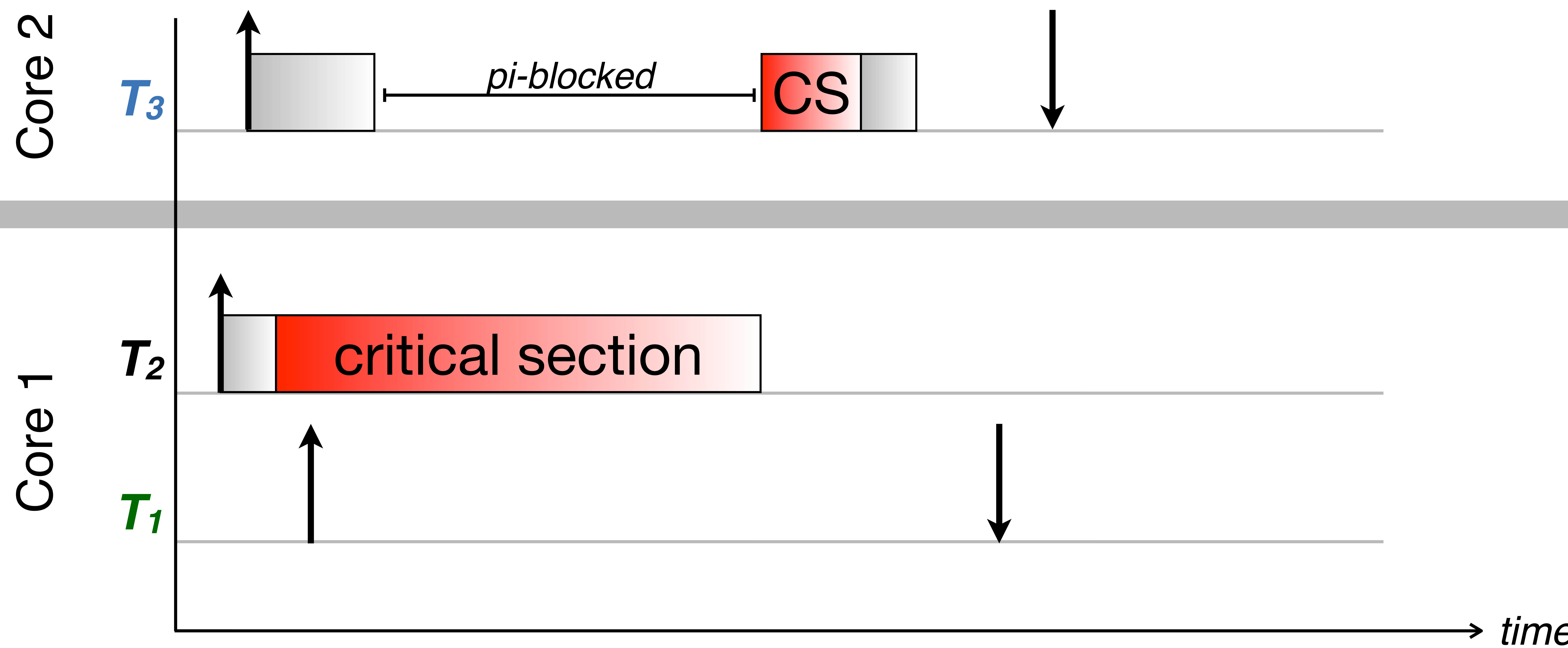
Case 1: priority boosting (=let T_2 continue).

three tasks, two cores, one resource, P-FP scheduling



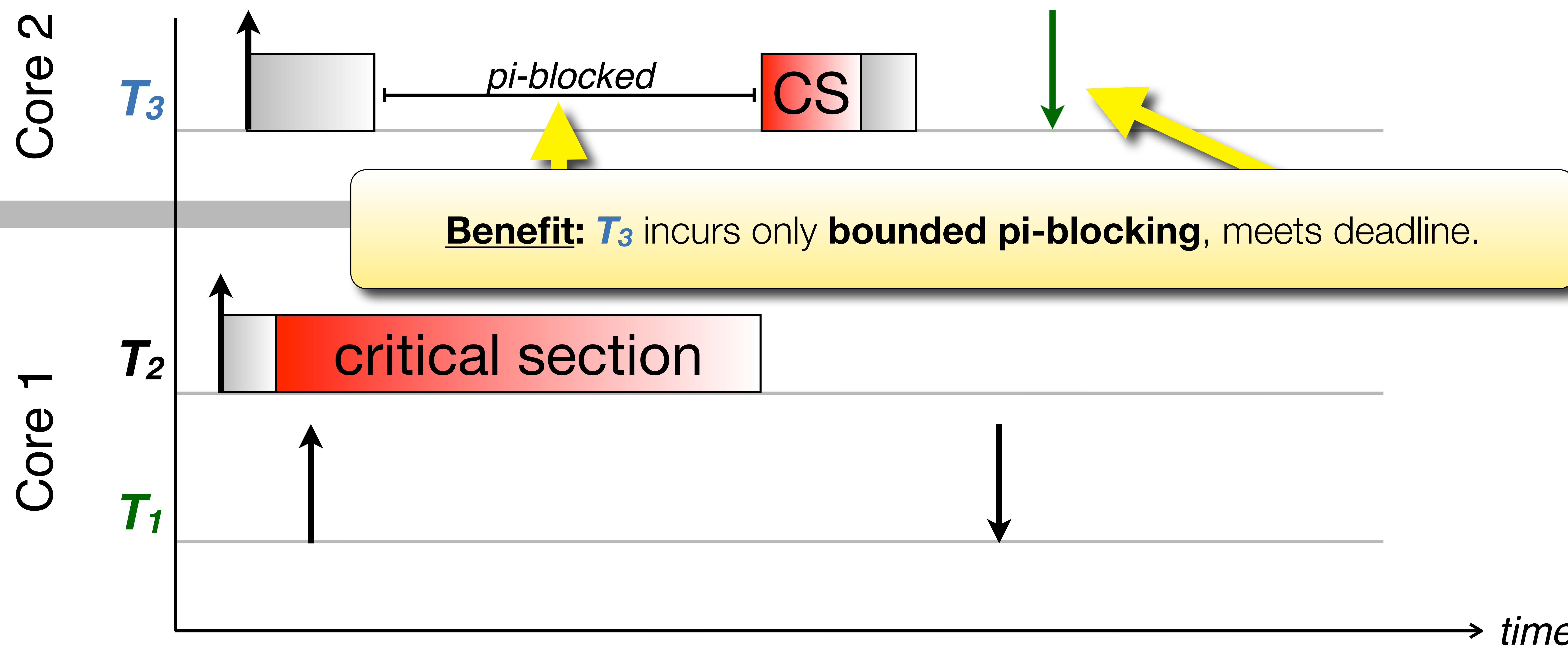
Case 1: priority boosting (=let T_2 continue).

three tasks, two cores, one resource, P-FP scheduling



Case 1: priority boosting (=let T_2 continue).

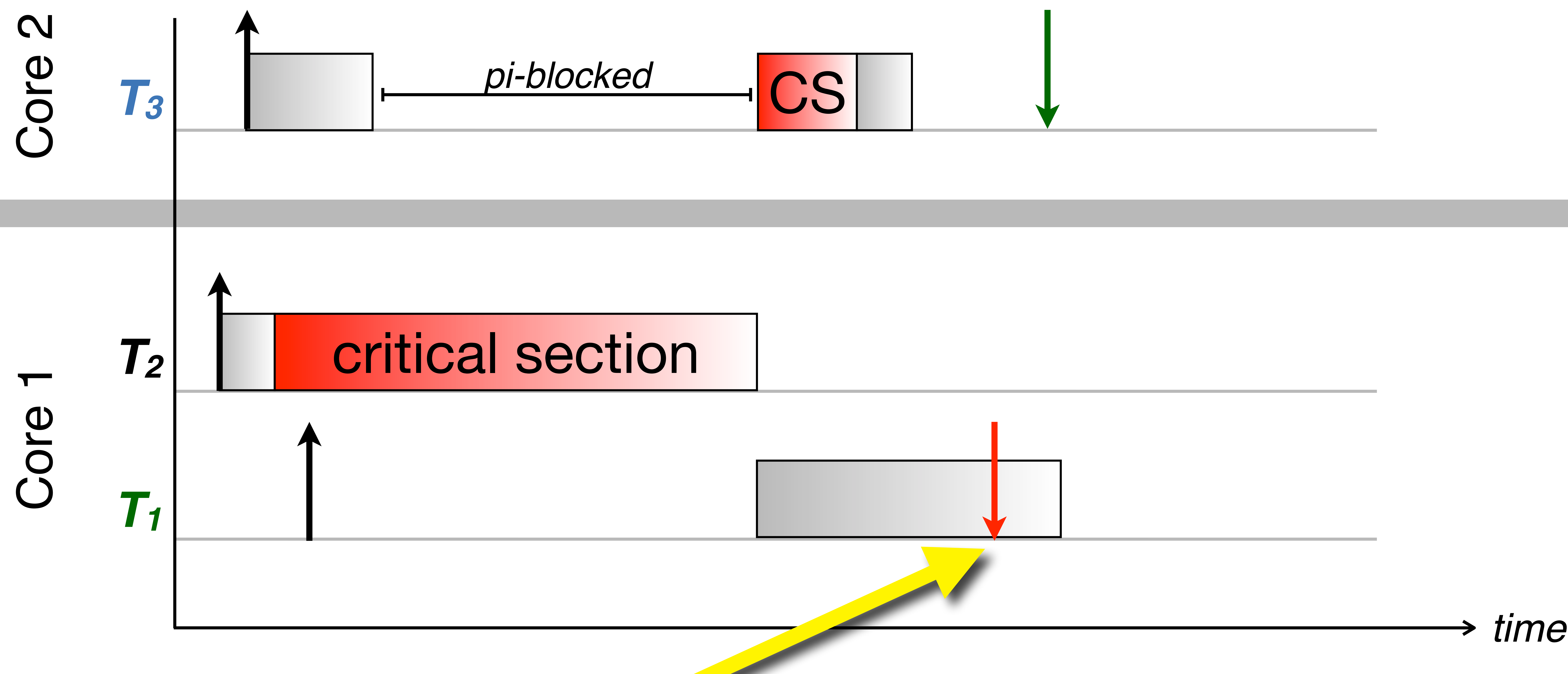
three tasks, two cores, one resource, P-FP scheduling



Benefit: T_3 incurs only **bounded pi-blocking**, meets deadline.

Case 1: priority boosting (=let T_2 continue).

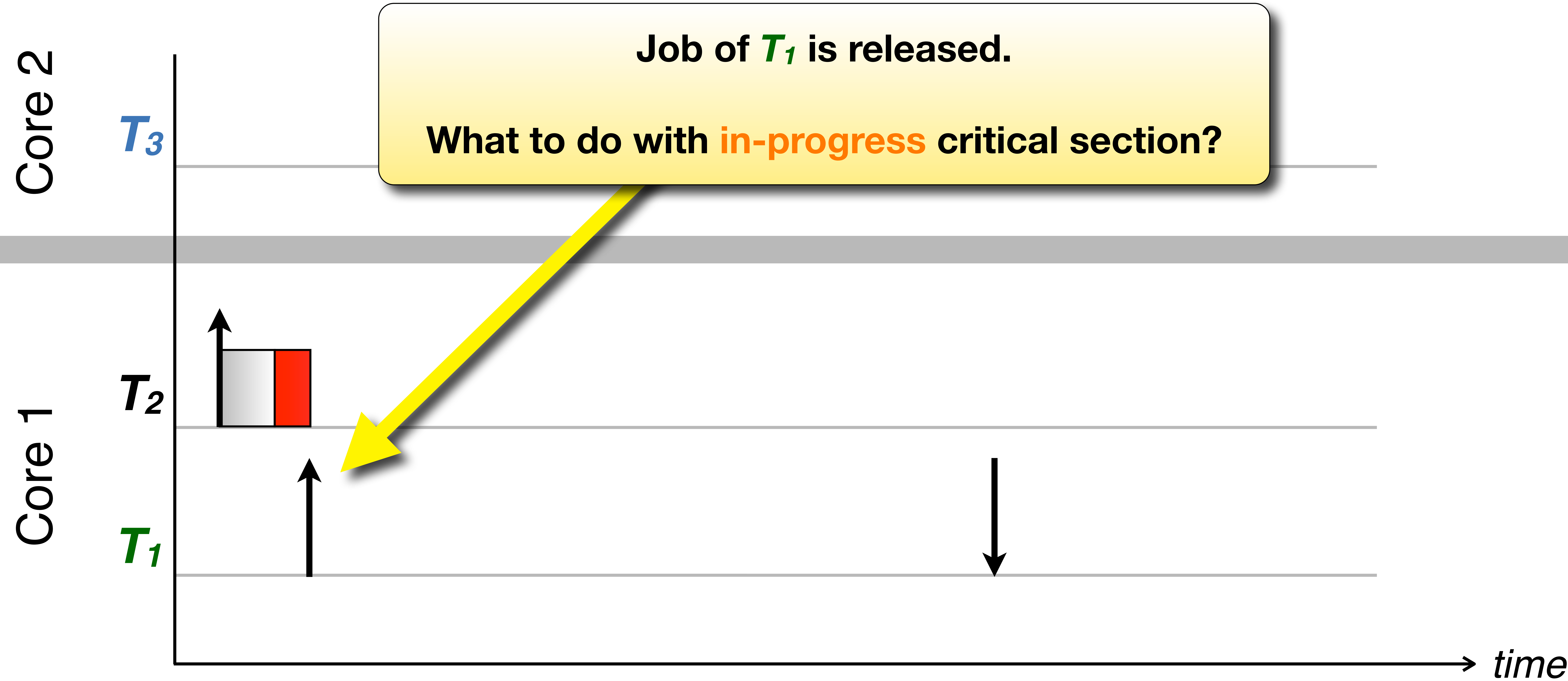
three tasks, two cores, one resource, P-FP scheduling



Problem: T_1 misses its deadline.

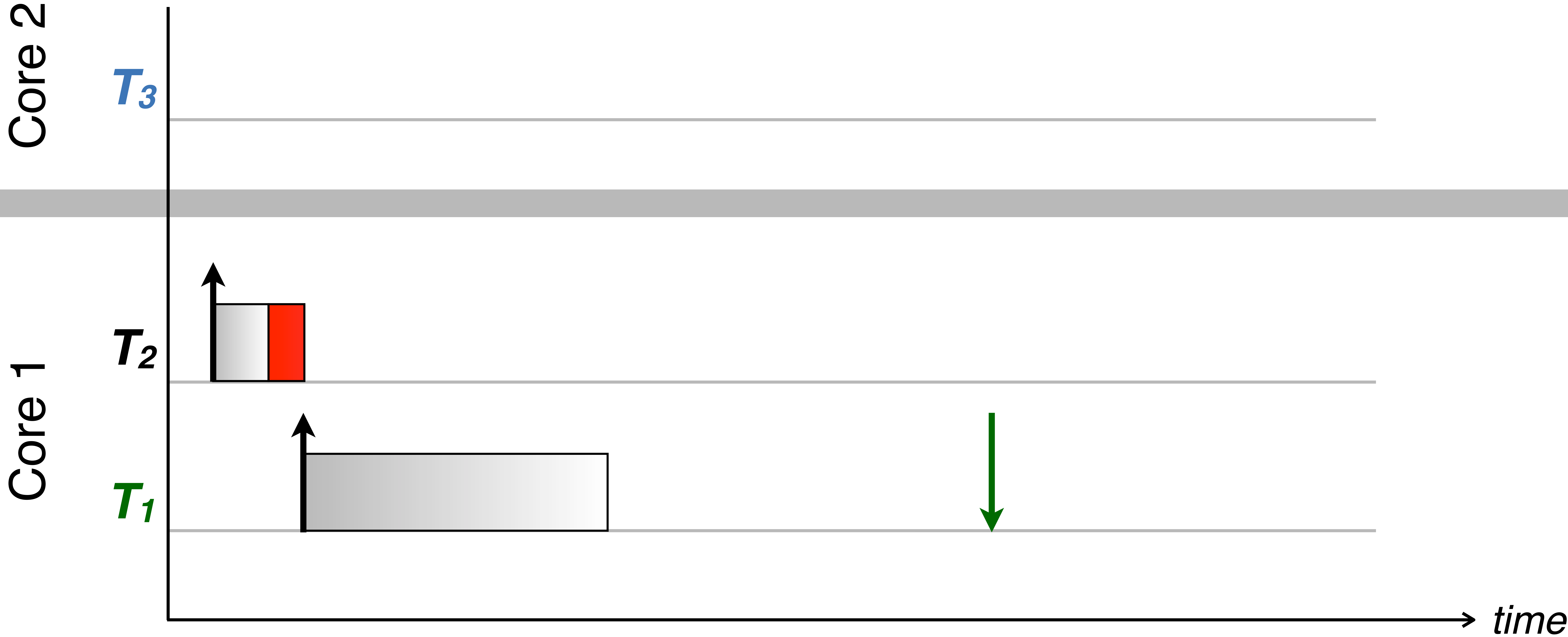
Example: Job Migration is Necessary

three tasks, two cores, one resource, P-FP scheduling



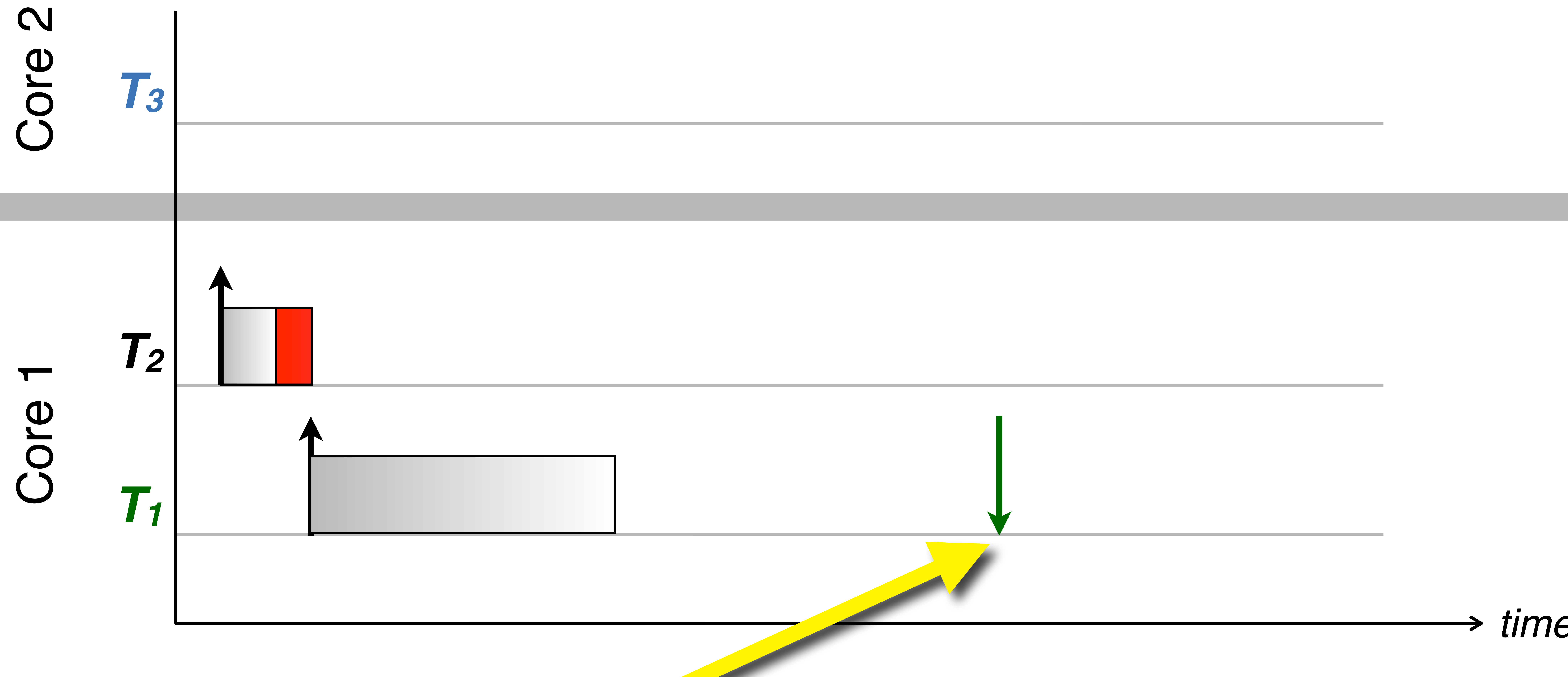
Case 2: **independence preservation** (= preempt T_2).

three tasks, two cores, one resource, P-FP scheduling



Case 2: independence preservation (= preempt T_2).

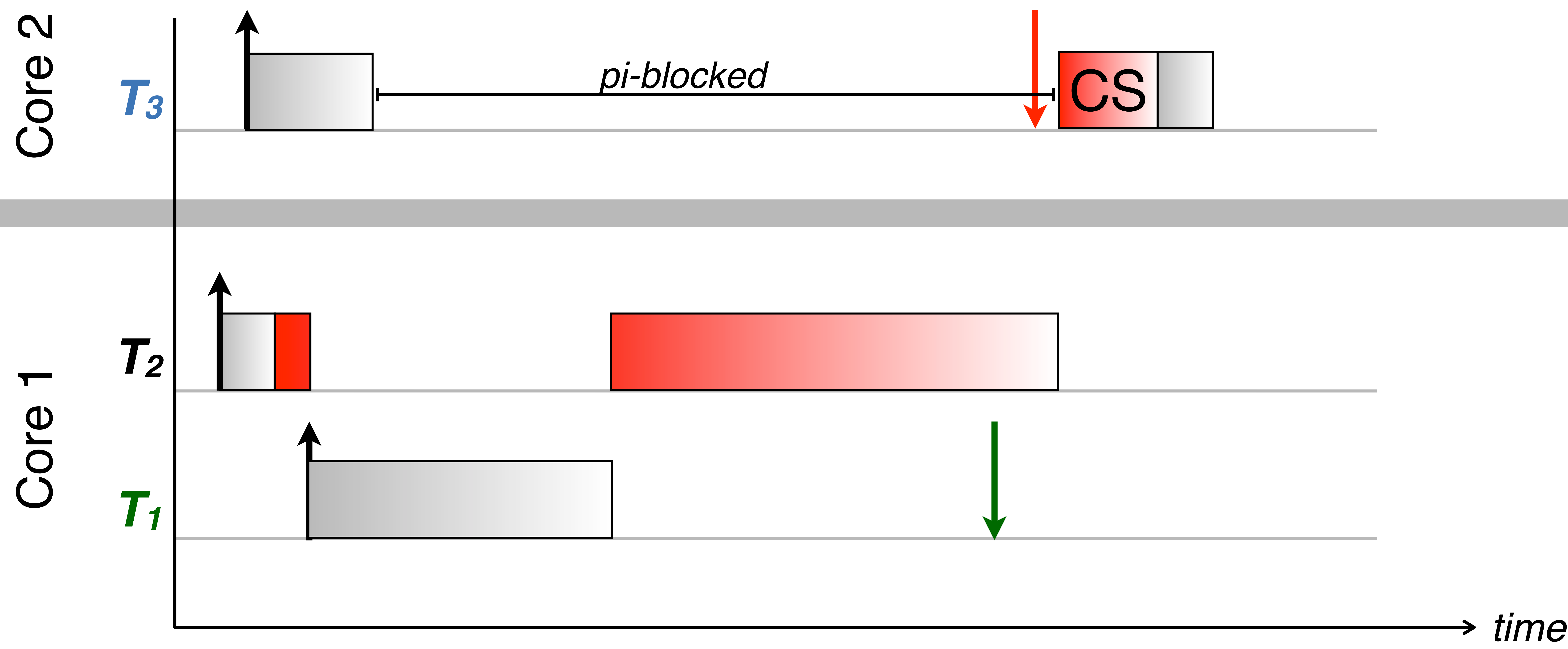
three tasks, two cores, one resource, P-FP scheduling



Independence preservation: T_1 meets its deadline.

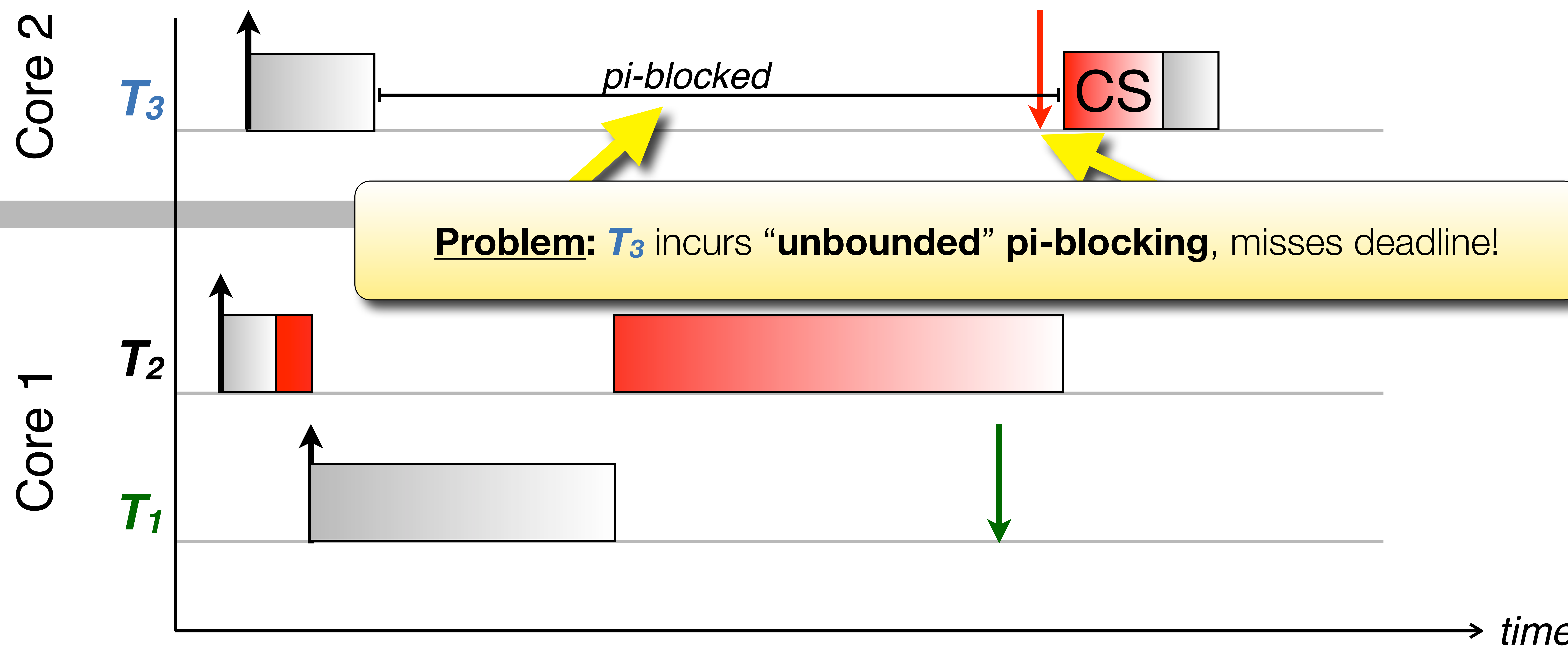
Case 2: independence preservation (= preempt T_2).

three tasks, two cores, one resource, P-FP scheduling



Case 2: independence preservation (= preempt T_2).

three tasks, two cores, one resource, P-FP scheduling

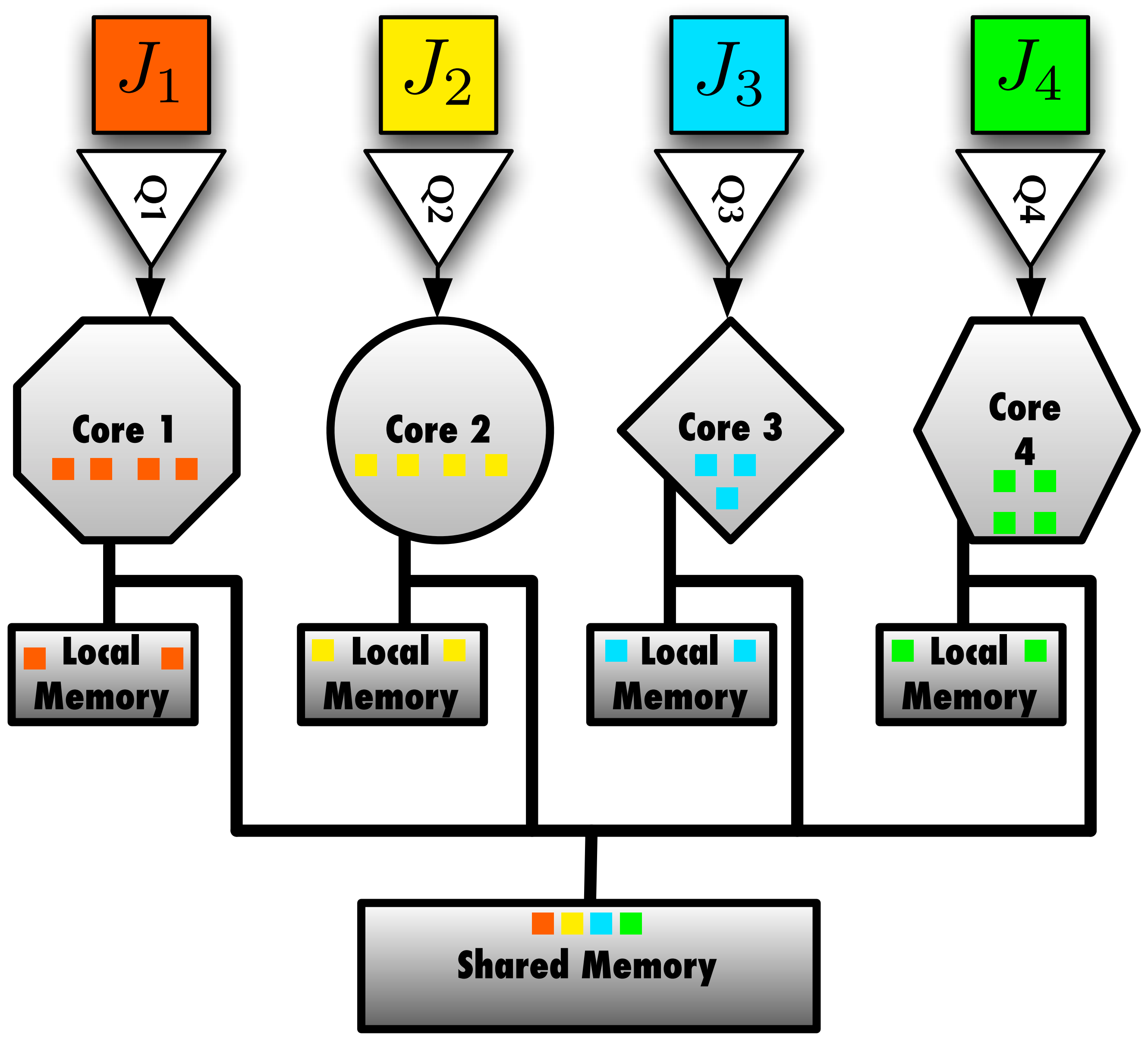


Problem: T_3 incurs "**unbounded**" **pi-blocking**, misses deadline!

Partitioned Scheduling with Migrations?

Partitioned Scheduling with Migrations?

Partitioned By Necessity



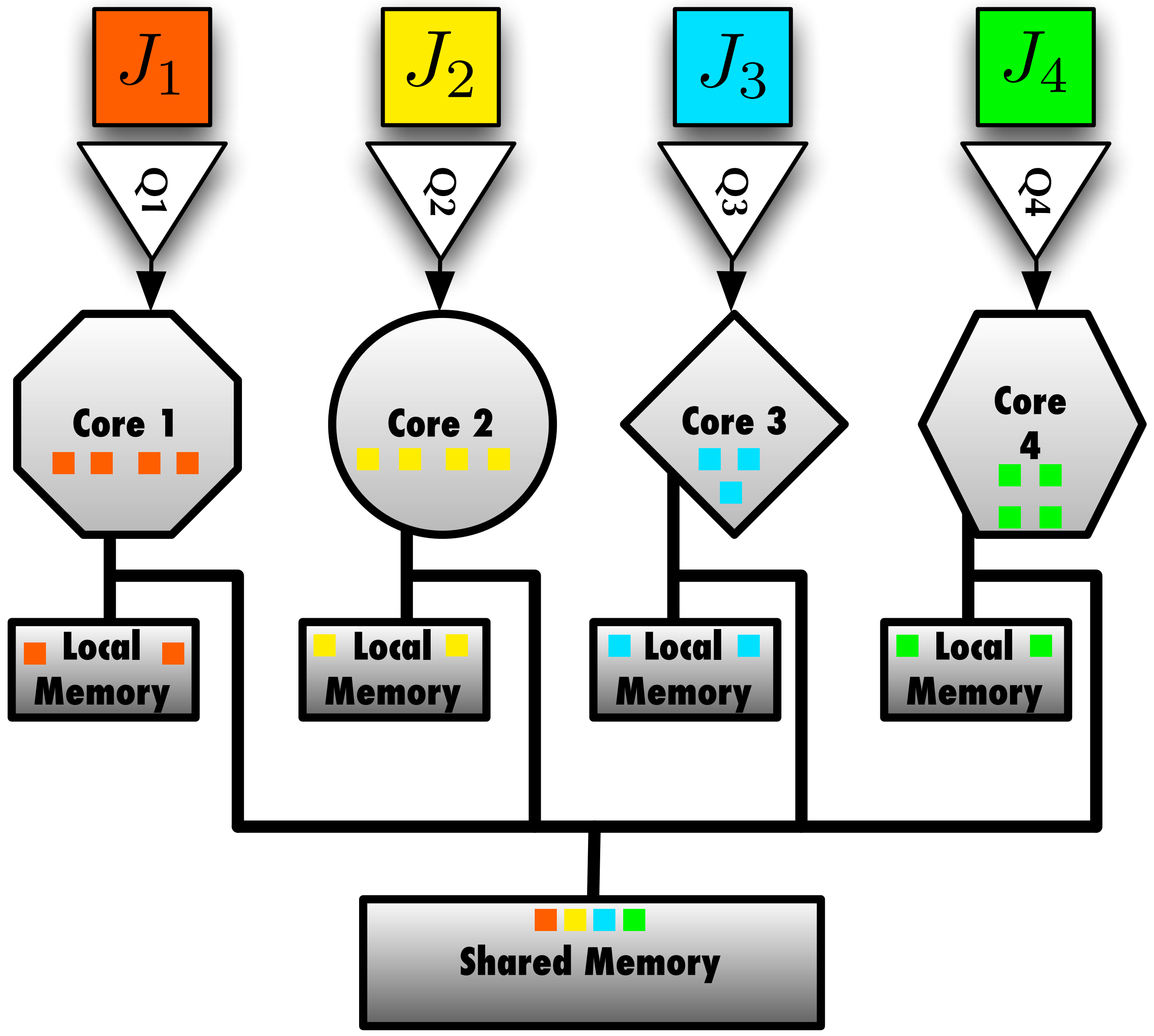
E.g., SoC with **heterogeneous cores** (ARM, PowerPC, x86, MIPS).

migrations **infeasible**

for lack of technical capability

Partitioned Scheduling with Migrations?

Partitioned By Necessity



E.g., SoC with **heterogeneous cores** (ARM, PowerPC, x86, MIPS).

<u>migrations **infeasible**</u>

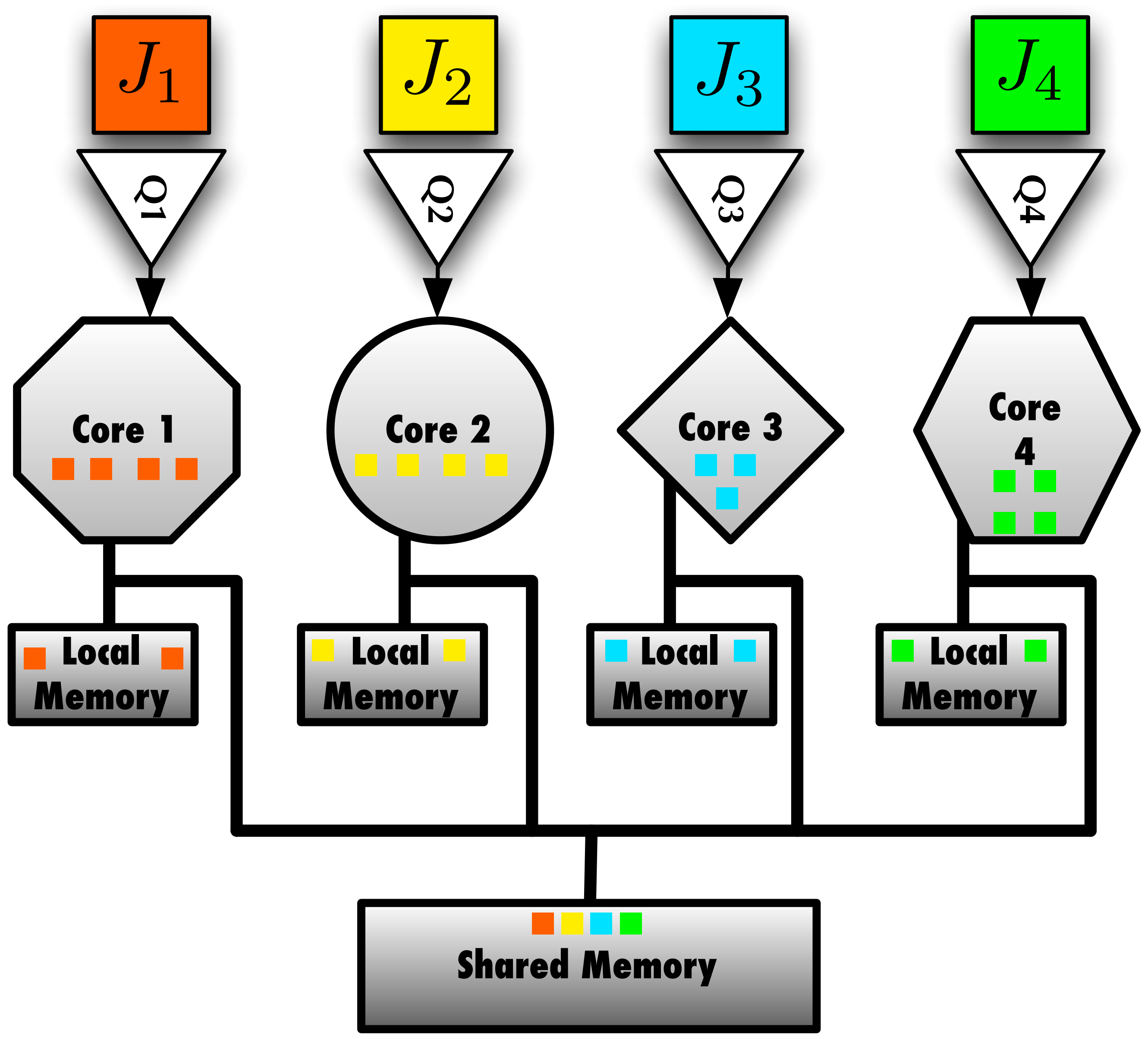
for lack of technical capability

➔ independence preservation **and** bounded priority inversion

impossible to achieve!

Partitioned Scheduling with Migrations?

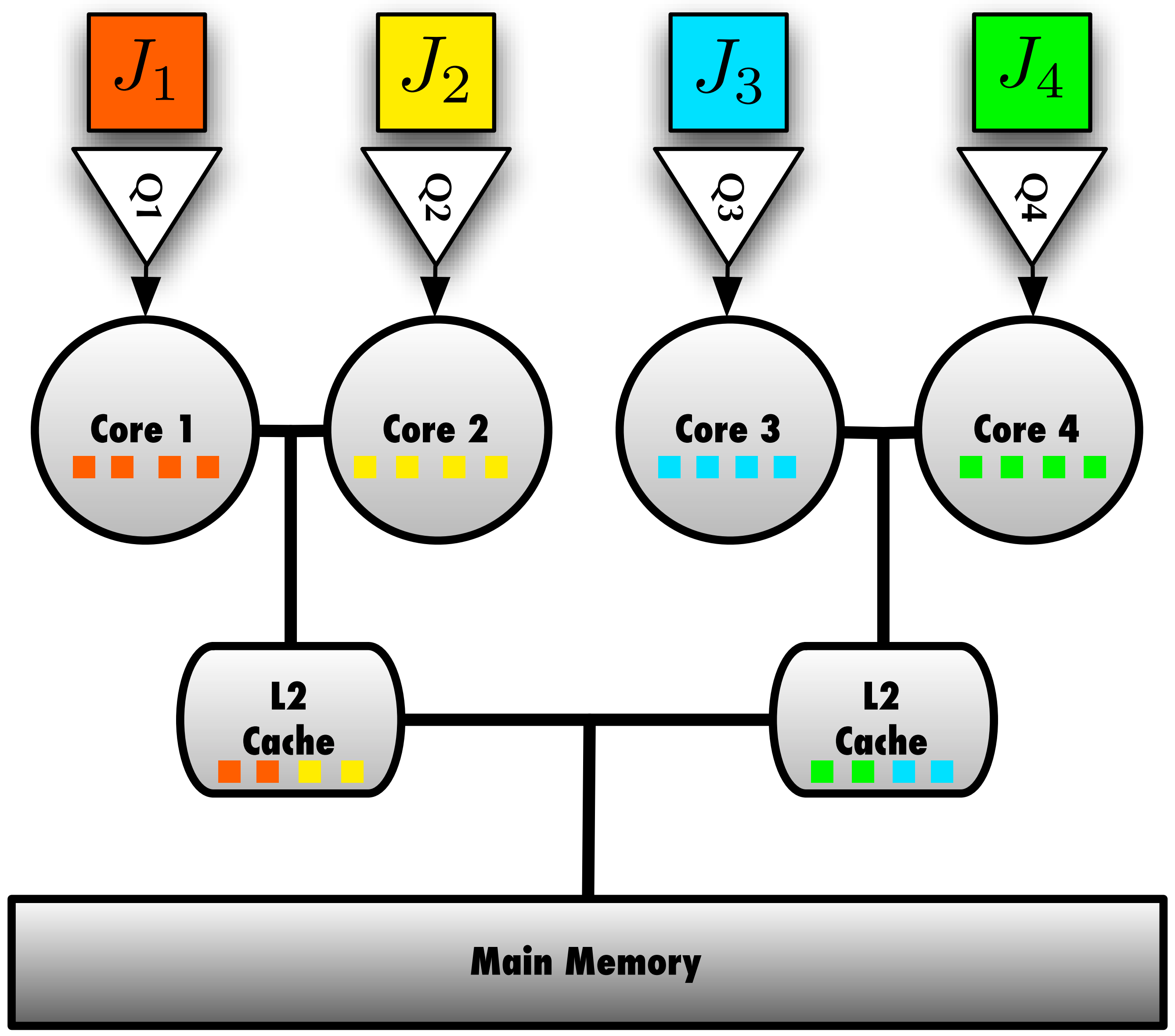
Partitioned By Necessity



migrations **infeasible**

for lack of technical capability

Partitioned By Choice



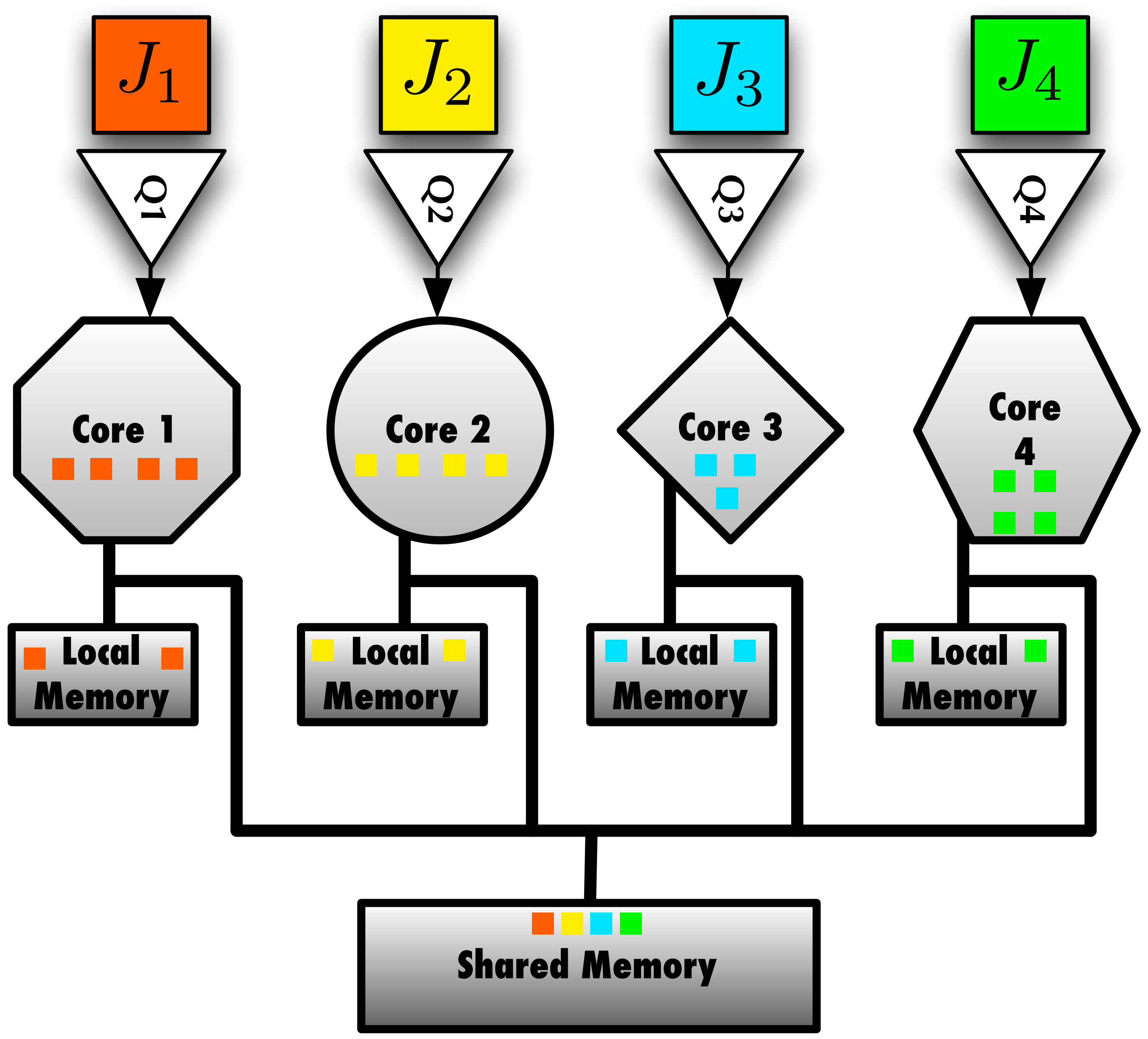
migrations **disallowed**

but technically feasible

Occasional migrations not desirable, but **possible**!

(Focus of this work.)

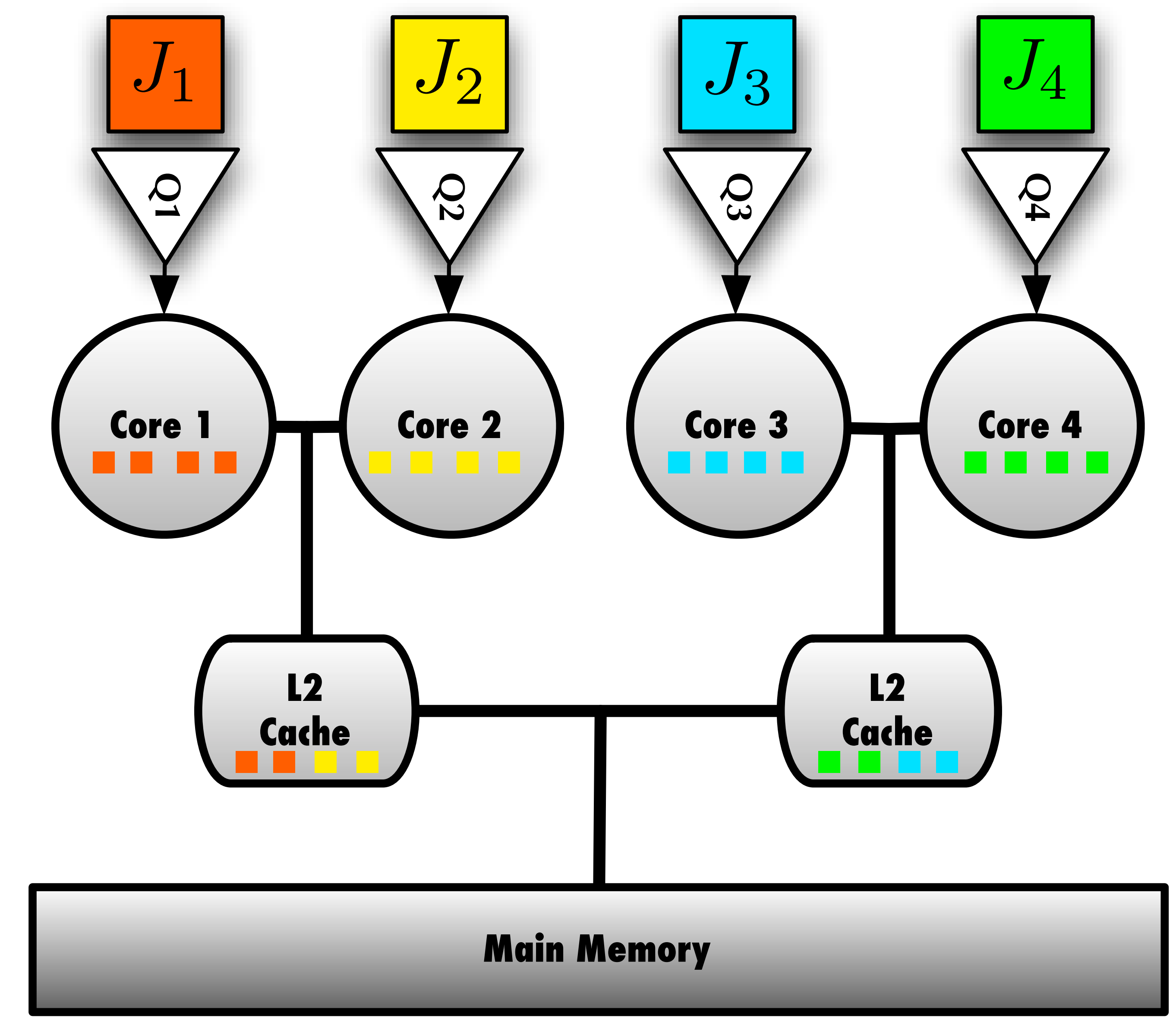
Partitioned By Necessity



migrations **infeasible**

for lack of technical capability

Partitioned By Choice

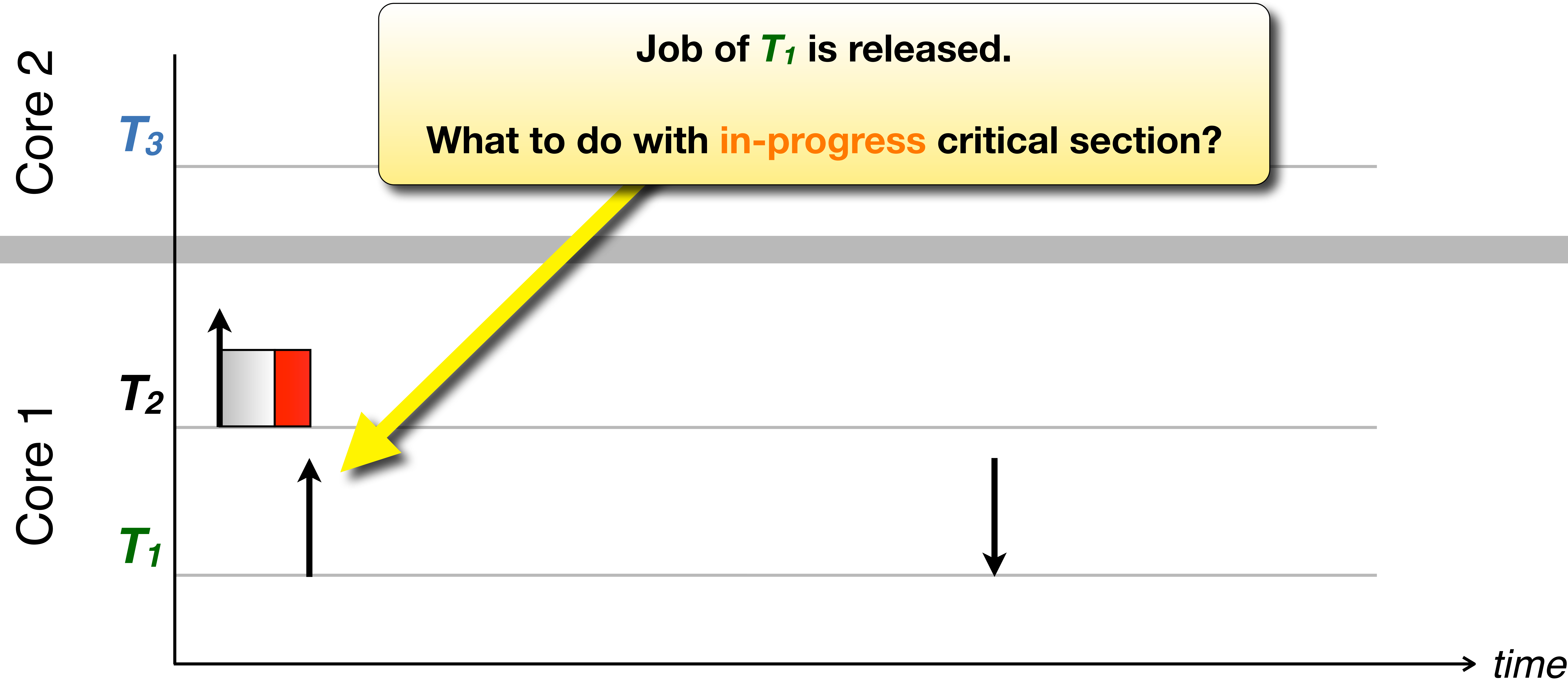


migrations **disallowed**

but technically feasible

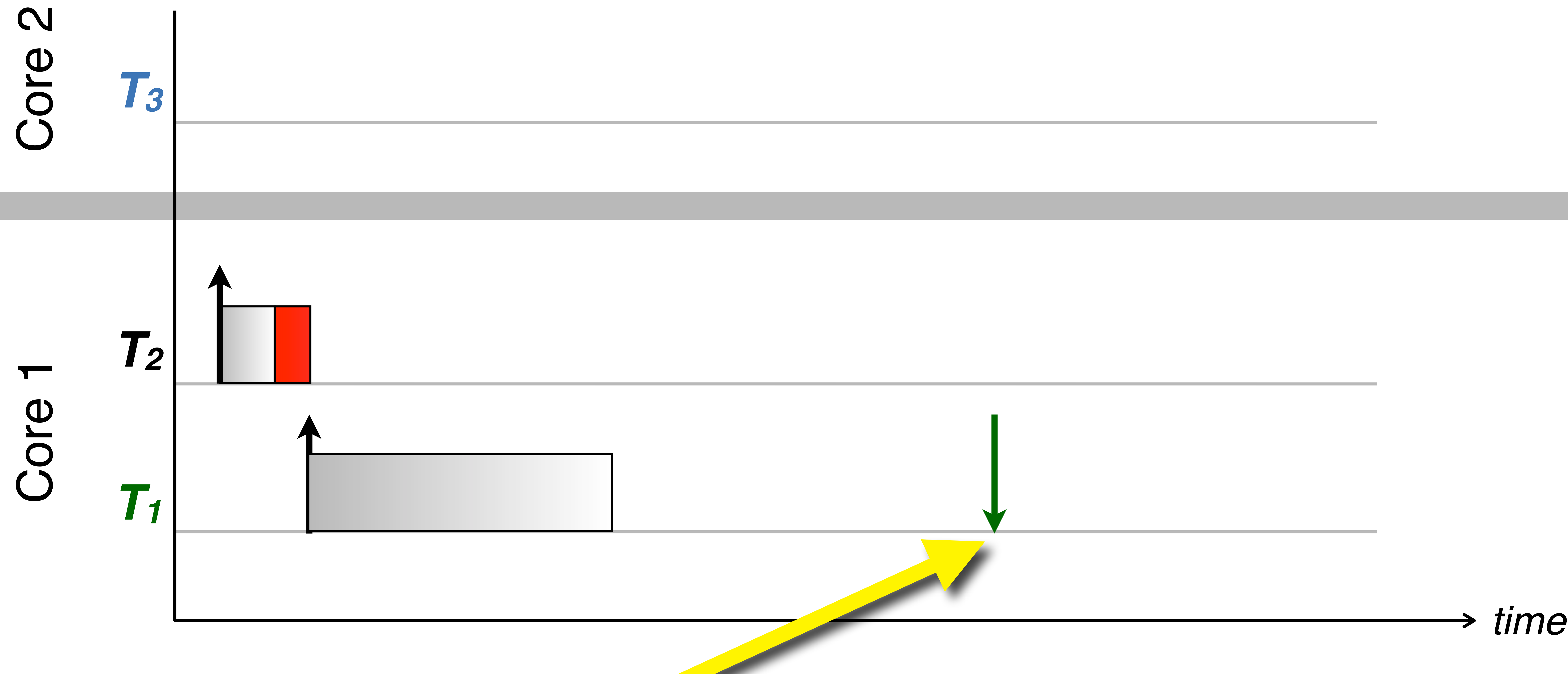
Example: Job Migration is Necessary

three tasks, two cores, one resource, P-FP scheduling



1) Ensure independence preservation (= preempt T_2).

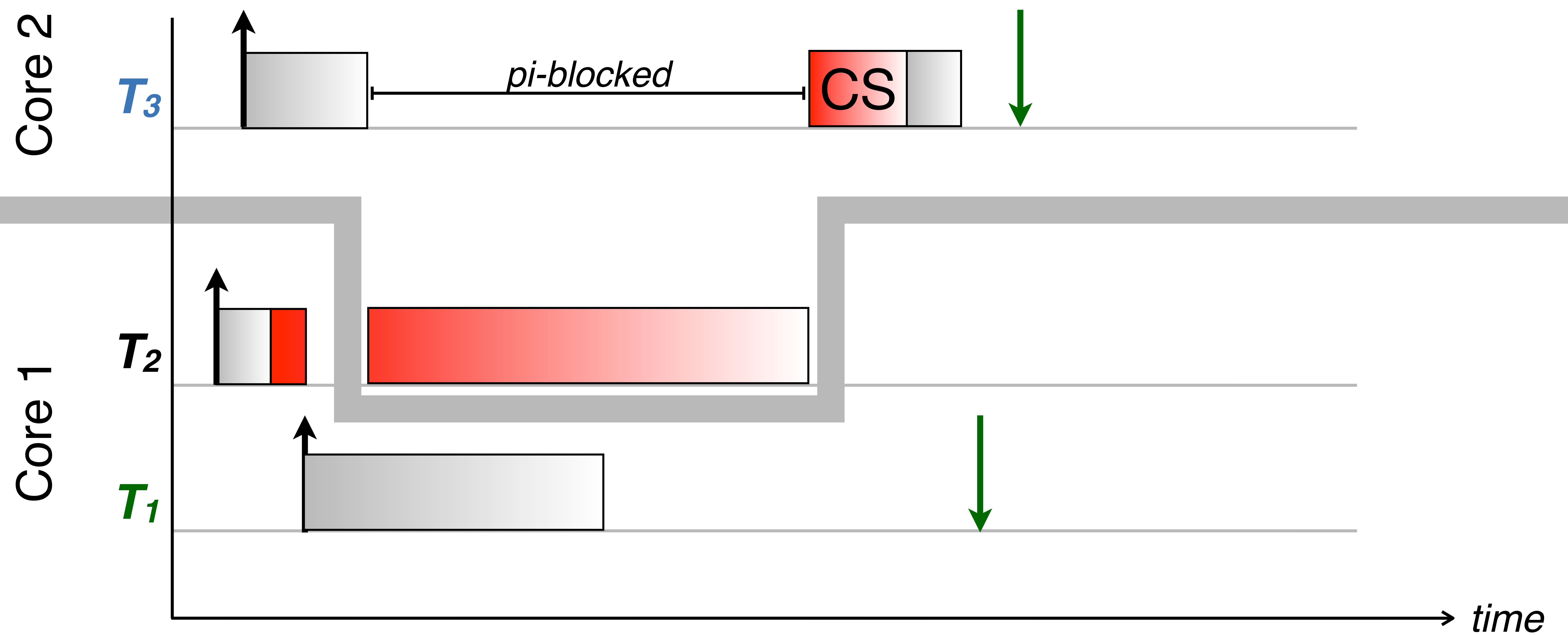
three tasks, two cores, one resource, P-FP scheduling



Independence preservation: T_1 meets its deadline.

2) Ensure bounded pi-blocking (= schedule T_2).

three tasks, two cores, one resource, P-FP scheduling

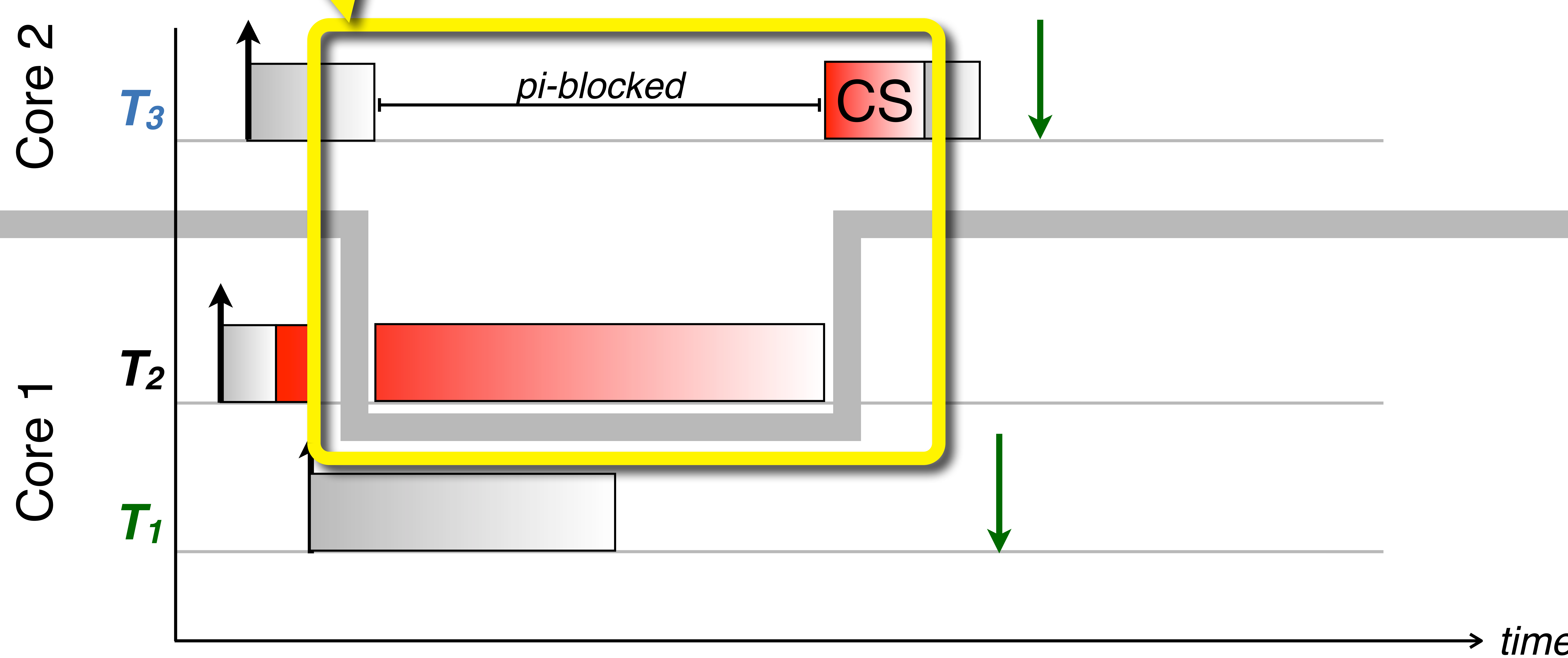


Easy fix: migrate T_2 when T_3 suspends.

Temporarily move T_3 to Core 2…

…igration is Necessary

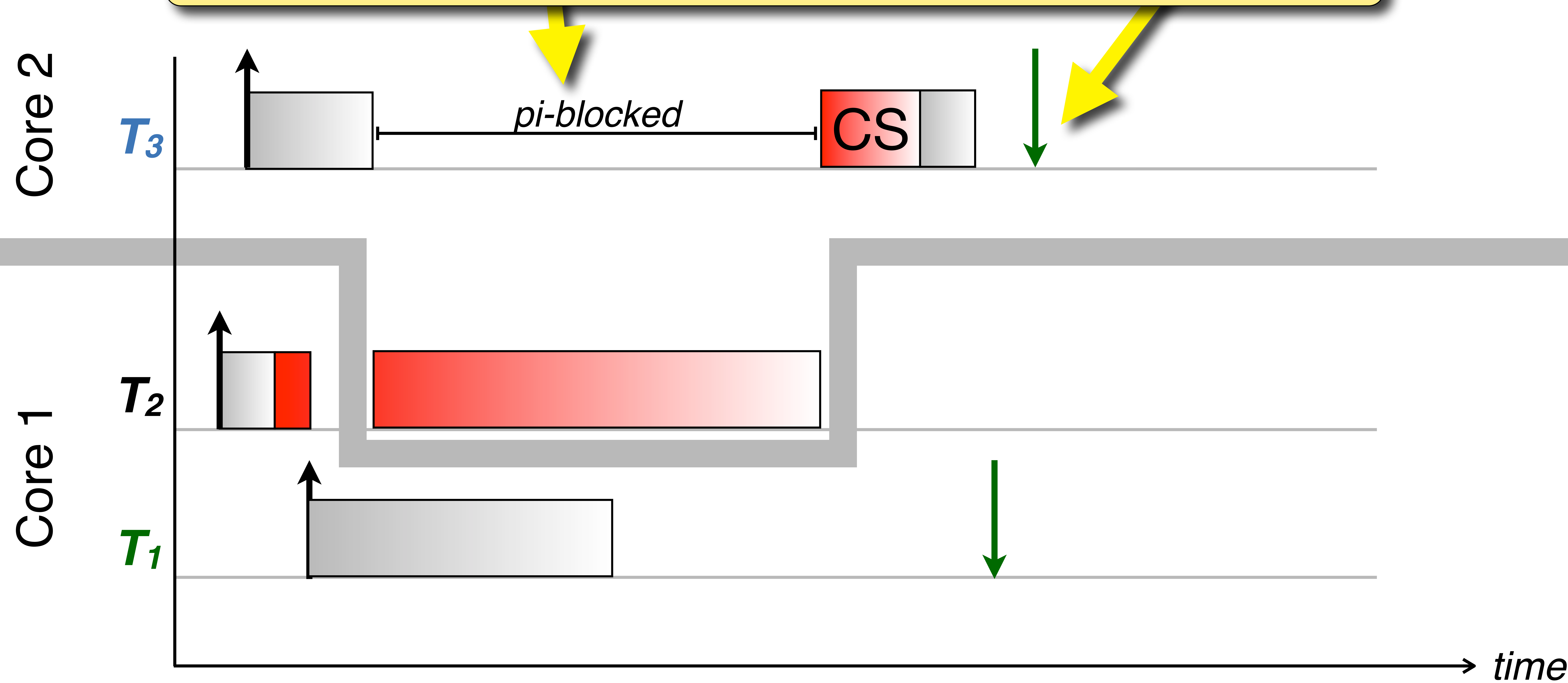
three tasks, two cores, one resource, P-FP scheduling



Easy fix: migrate T_2 when T_3 suspends.

Example: Job Migration is Necessary

Benefit: T_3 incurs only **bounded pi-blocking**, meets deadline.



Easy fix: migrate T_2 when T_3 suspends.

Theorem

*Under non-global scheduling ($c \neq m$), it is **impossible** for a semaphore protocol to simultaneously*

(i) *prevent **unbounded pi-blocking**,*

(ii) *be **independence-preserving**, and*

(iii) *avoid **inter-cluster job migrations**.*

Pick any two…

Combinations of Properties

*Under non-global scheduling ($c \neq m$), it is **impossible** for a semaphore protocol to simultaneously*

(i) *prevent* ***unbounded pi-blocking****,*

(ii) *be* ***independence-preserving****, and*

(iii) *avoid* ***inter-cluster job migrations****.*

(i) & (iii)
➡ MPCP, Part. FMLP, FMLP$^+$, OMLP, …

(ii) & (iii)
➡ Applying PIP to partitioned scheduling (**not sound!**)

(i) & (ii)
➡ **no such protocol known!**

Part 2

Independence Preservation
+
Asymptotically Optimal PI-Blocking

High-Level Overview

real-time locking protocol	=	progress mechanism	+	queue structure

High-Level Overview

real-time locking protocol = *progress mechanism* + *queue structure*

progress mechanism → Must be independence-preserving.

queue structure → Must ensure asymptotic optimality.

High-Level Overview

real-time locking protocol = *progress mechanism* + *queue structure*

Must be independence-preserving.

Must ensure asymptotic optimality.

Adopt intuition from example:

when **lock holder is preempted**,
migrate to blocked task's processor.

Migratory Priority Inheritance

classic priority inheritance

*inherit **priority** of blocked jobs*

Migratory Priority Inheritance

classic priority inheritance

inherit ***priority*** *of blocked jobs*

+

“cluster inheritance”

inherit ***eligibility to execute on assigned clusters*** *from blocked jobs*

Jobs remain **fully preemptive** even in critical sections.

→ enables **independence preservation**

classic priority inheritance

*inherit **priority** of blocked jobs*

+

“**cluster inheritance**”

*inherit **eligibility to execute on assigned clusters** from blocked jobs*

High-Level Overview

real-time locking protocol = *progress mechanism* + *queue structure*

Must be independence-preserving.

Must ensure asymptotic optimality.

High-Level Overview

real-time locking protocol = *progress mechanism* + *queue structure*

Must be independence-preserving.

Must ensure asymptotic optimality.

Resolve (most) contention within clusters: use a multi-level queue.

A 3-Level FIFO/FIFO/PRIO Queue

one 3-level queue for each resource

Cluster 1

Cluster 2

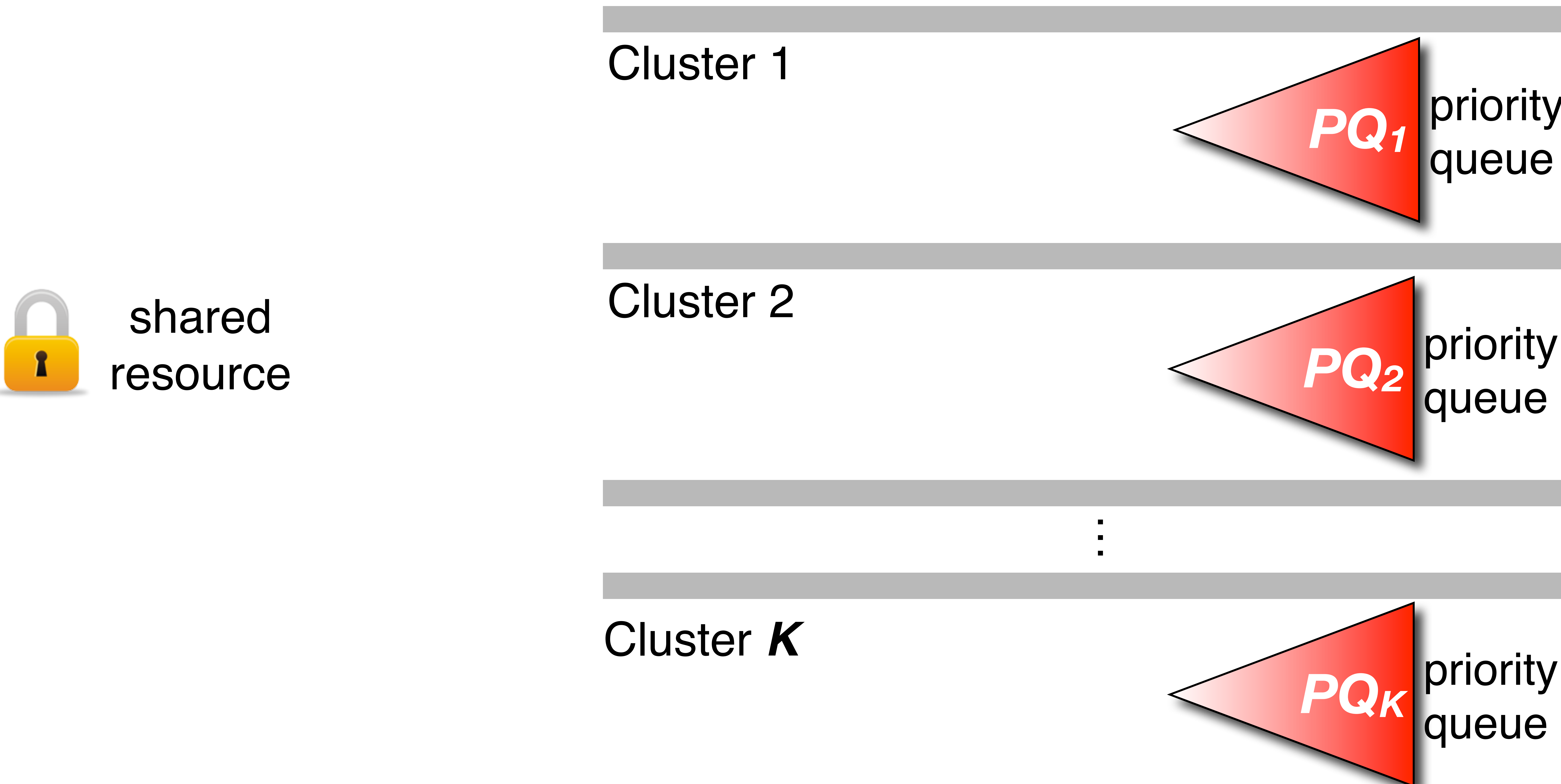
shared resource

⋮

Cluster ***K***

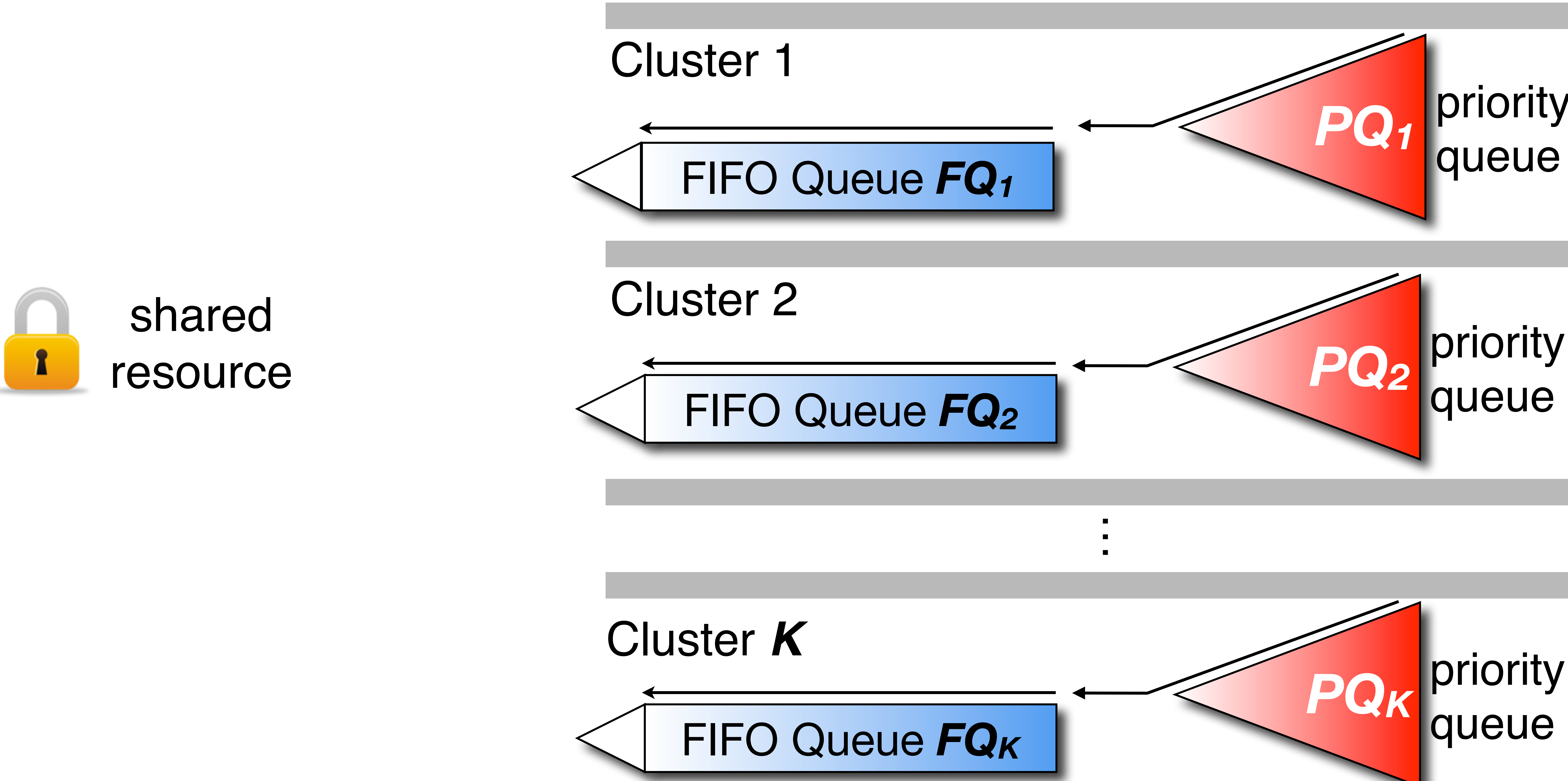
A 3-Level FIFO/FIFO/PRIO Queue

one 3-level queue for each resource



A 3-Level FIFO/FIFO/PRIO Queue

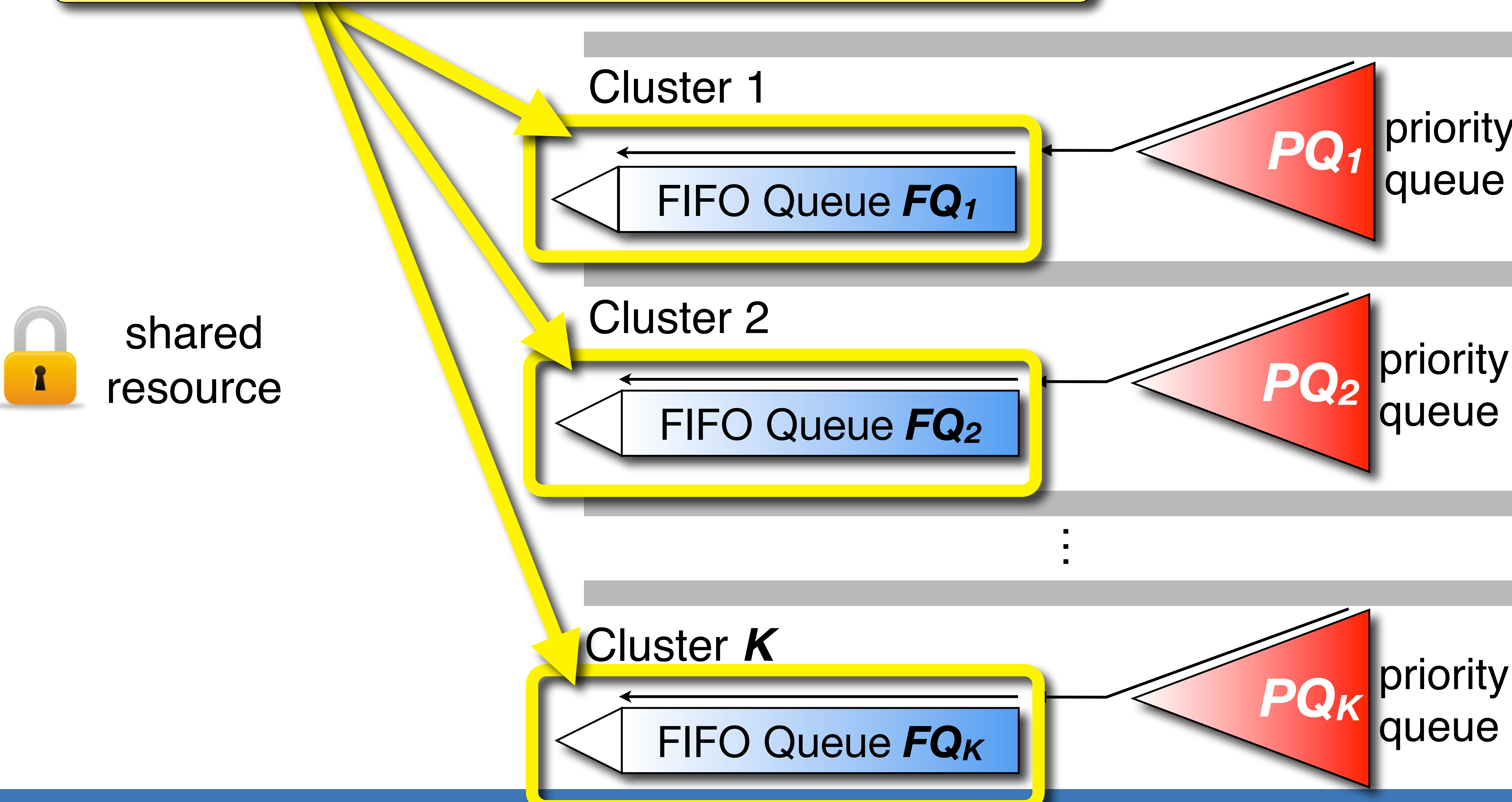
one 3-level queue for each resource



Queue

Bounded length: at most ***c*** jobs (in each cluster).

*(**c** = number of cores in cluster)*

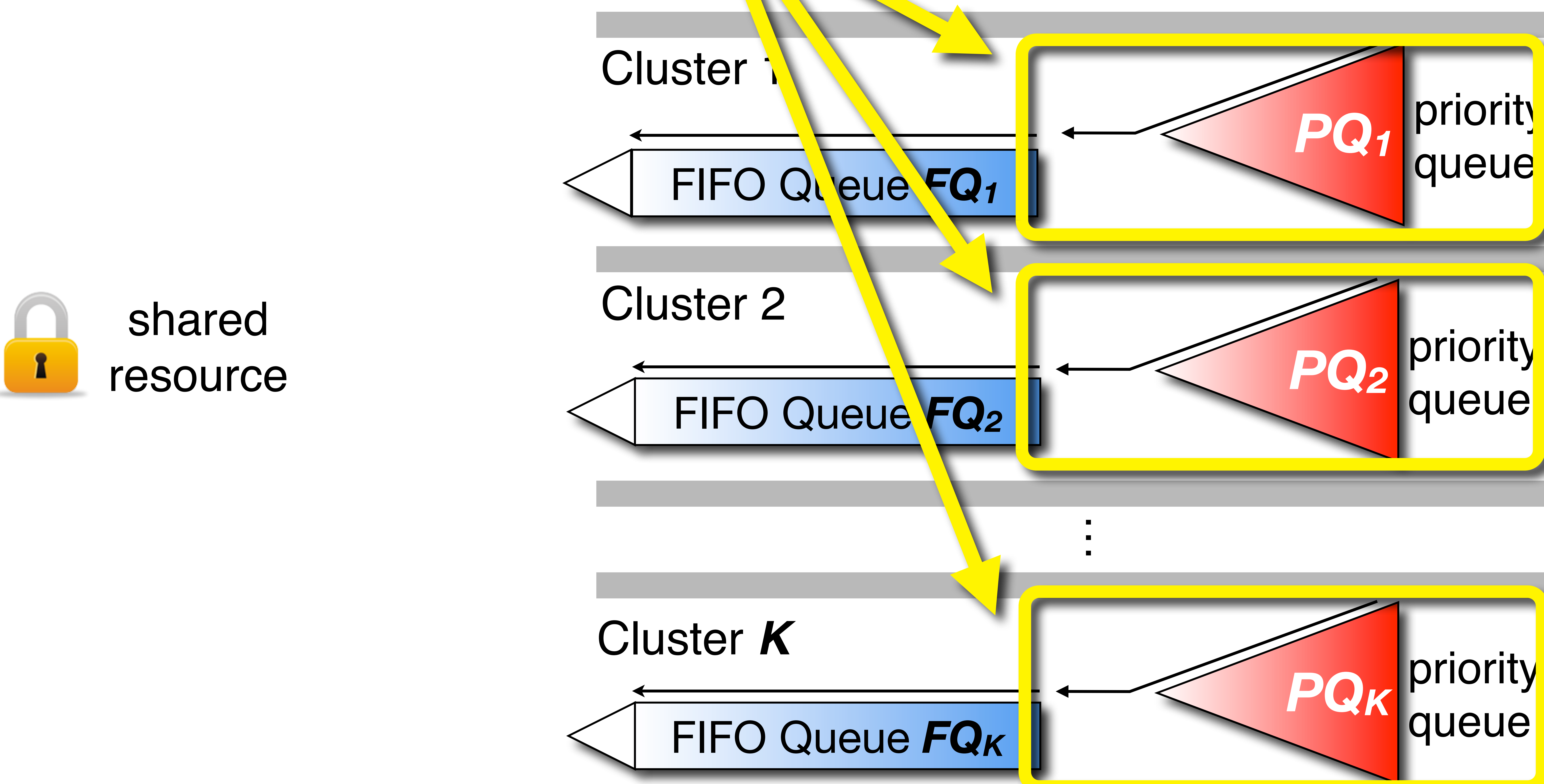


A 3-Le

Priority queue used only if more than ***c*** jobs contend.

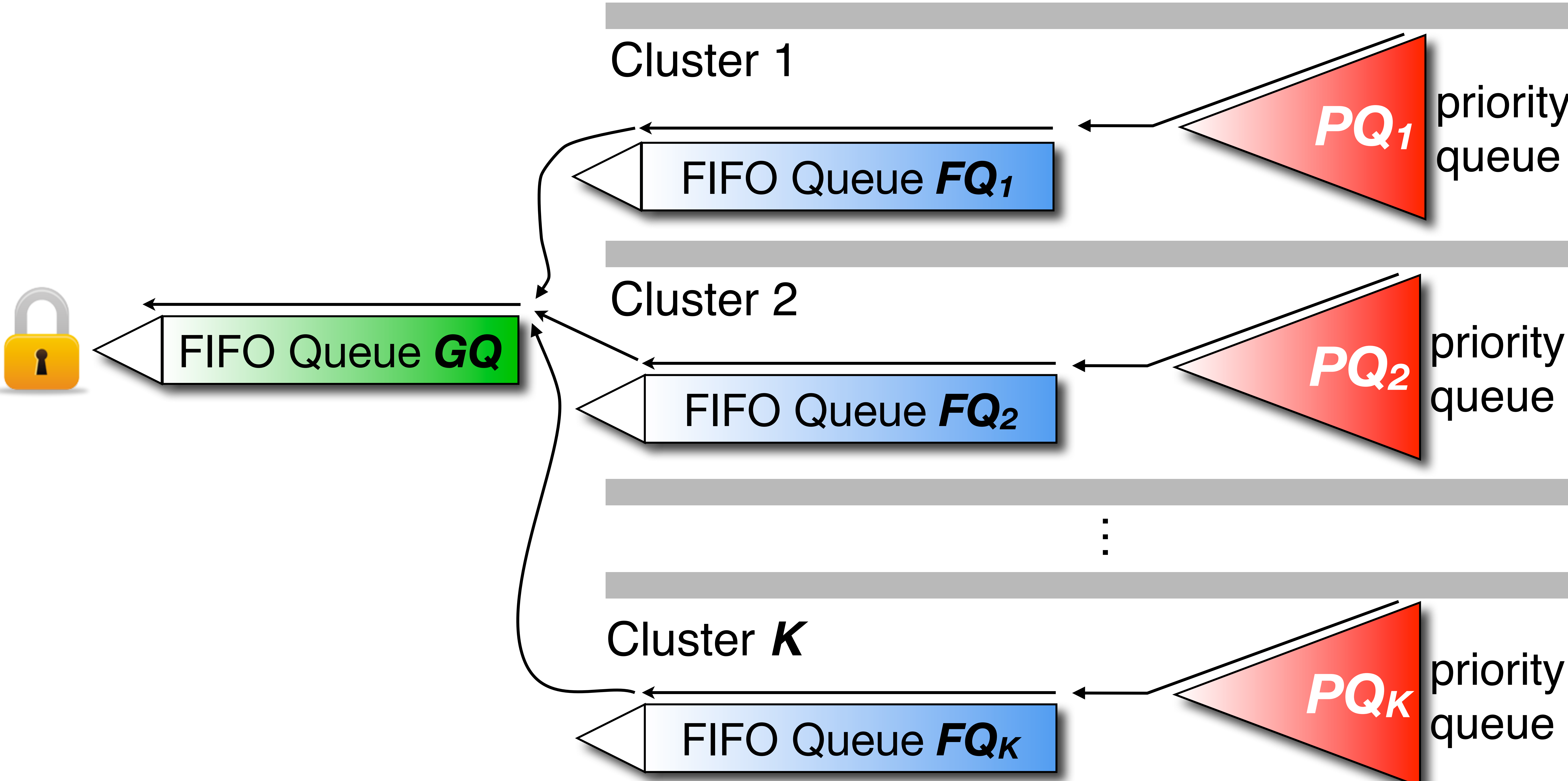
*(**c** = number of cores in cluster)*

one 3-level que e for each resource



A 3-Level FIFO/FIFO/PRIO Queue

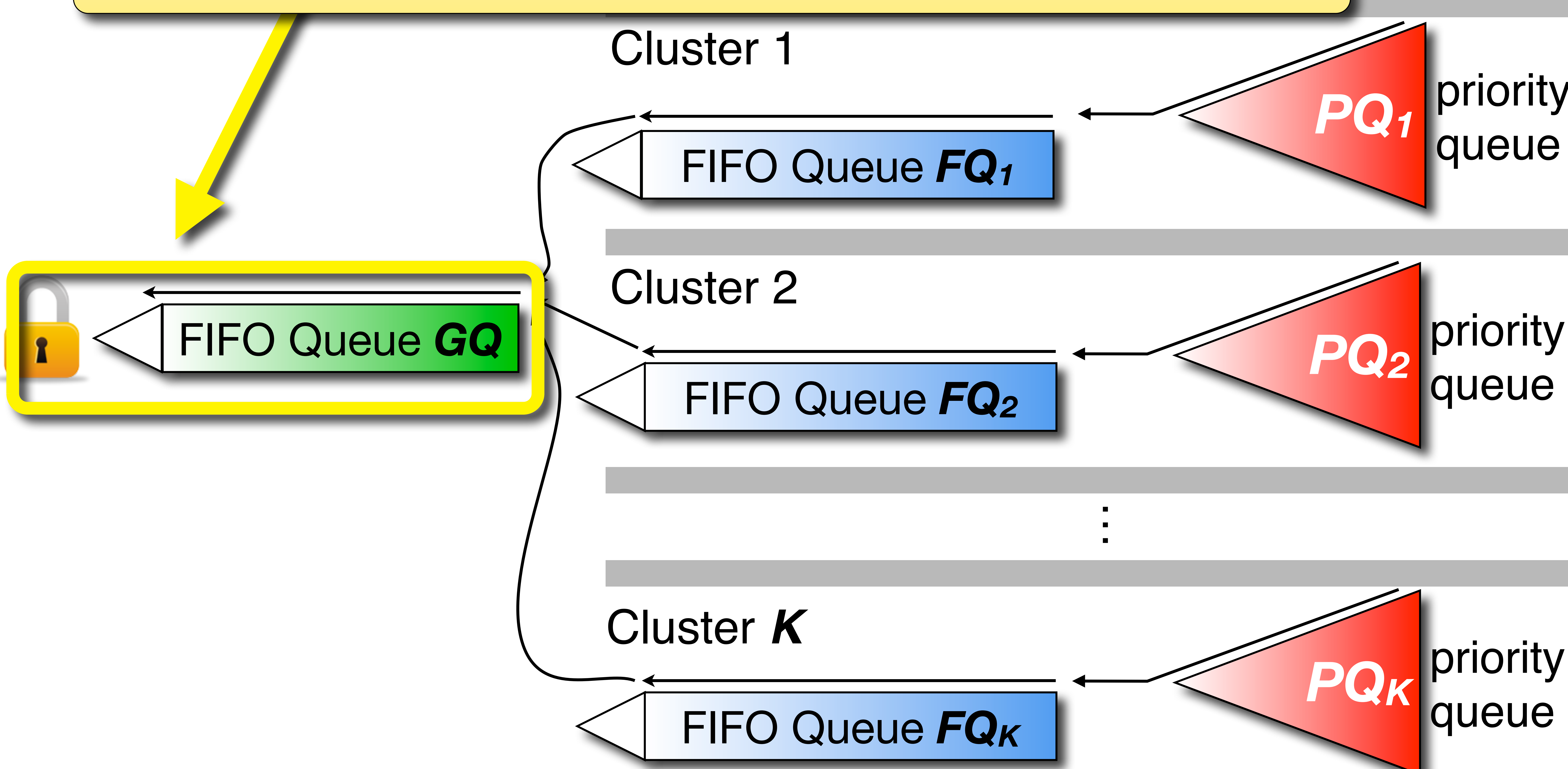
one 3-level queue for each resource



Global FIFO Queue resolves inter-cluster contention.

Bounded length: at most $K = m / c$ jobs (one per cluster).

(m = total number of cores, c = number of cores per cluster)



The O(m) Independence-Preserving Locking Protocol (OMIP)

The OMIP = *migratory priority inheritance* + *3-level F/F/P queue*

independence-preserving

O(m) s-oblivious pi-blocking

The O(m) Independence-Preserving Locking Protocol (OMIP)

The OMIP = *migratory priority inheritance* + *3-level F/F/P queue*

migratory priority inheritance → independence-preserving

3-level F/F/P queue → **O(m)** s-oblivious pi-blocking

Ω(m) lower bound on s-oblivious pi-blocking (— & Anderson, 2010)

➔ The OMIP ensures **asymptotically optimal** s-oblivious pi-blocking.

Part 3

Evaluation

Prototype Implementation

LITMUS RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

www.litmus-rt.org

3-level queues

➡ easy (reuse Linux wait queues)

➡ cheap compared to syscall

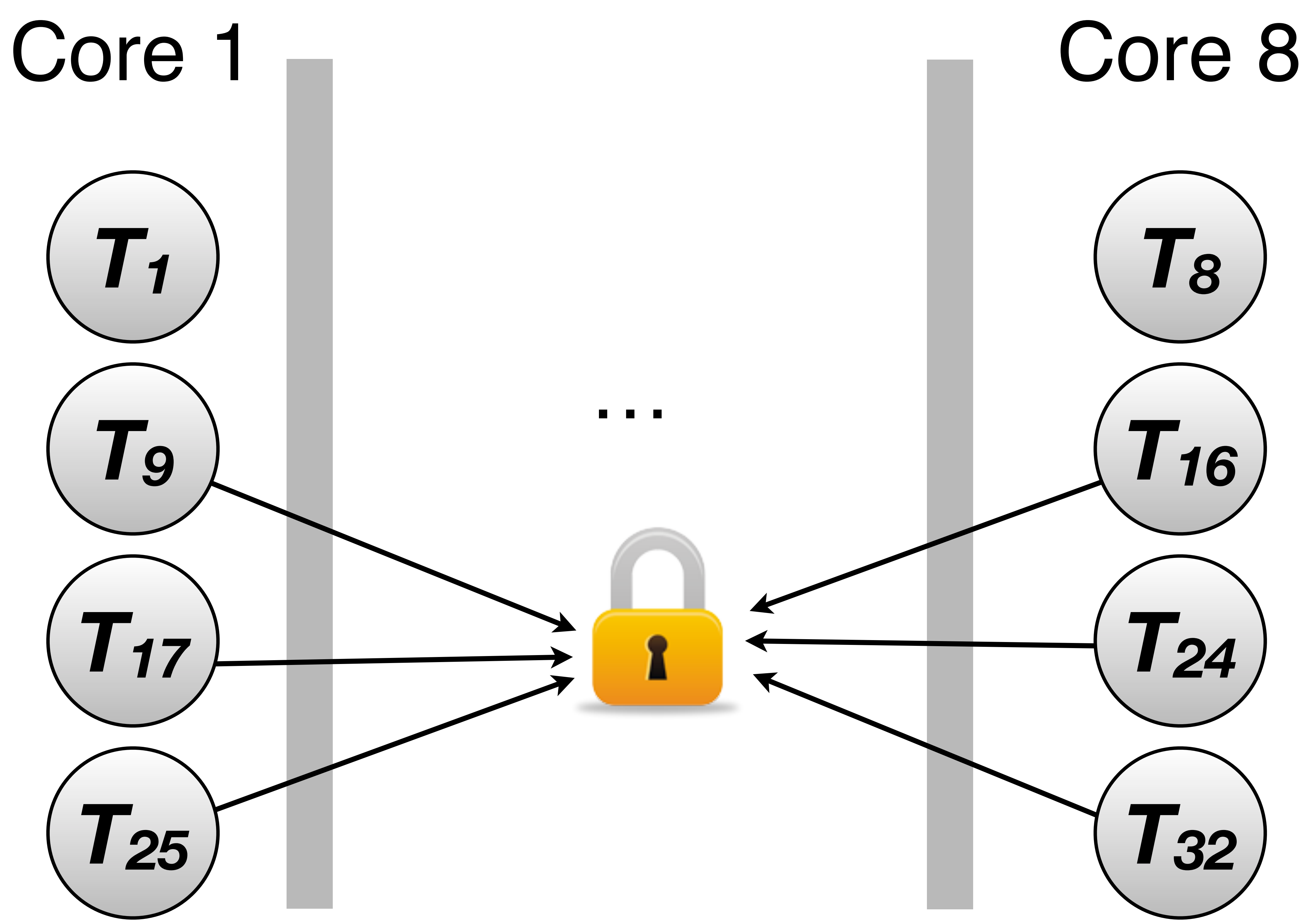
Migratory priority inheritance

➡ more tricky (need to avoid global locks)

➡ store bitmap of cores "offering" to schedule lock holder in each lock

Response Times Experiment

on an 8-core, 2-Ghz Xeon X7550 System



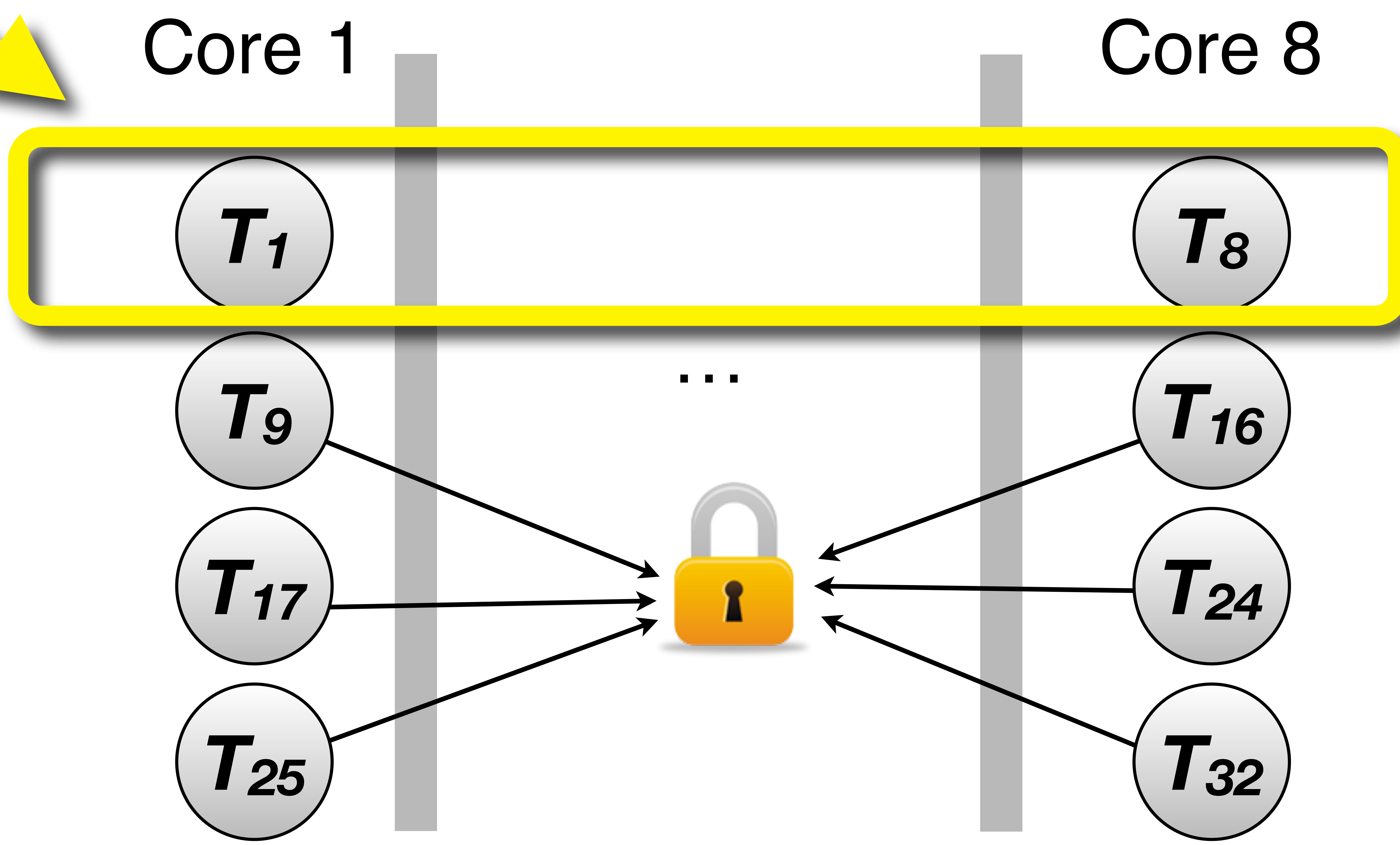
Setup

- ➡ 4 tasks on each core (one independent & latency-sensitive)
- ➡ one shared resource
- ➡ max. critical section length: **~1ms**

One **latency-sensitive**, independent task with **period = 1ms**.

…imes Experiment

…Ghz Xeon X7550 System



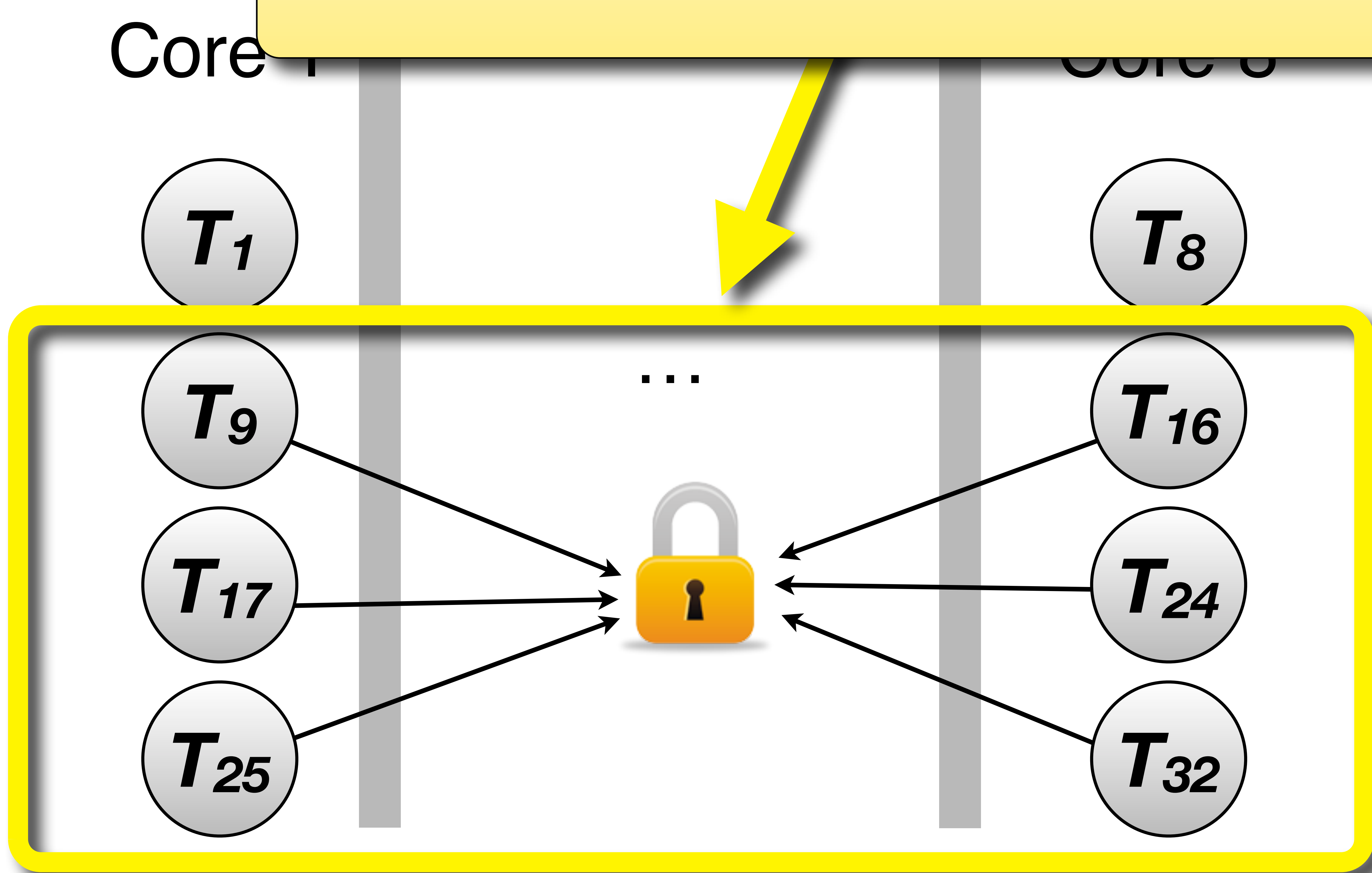
Setup

- ➡ 4 tasks on each core (one independent & latency-sensitive)
- ➡ one shared resource
- ➡ max. critical section length: **~1ms**

Respo

on an 8

Three tasks (on each core) with
periods 25 ms, 100 ms, and **1000 ms**.
Each job of these tasks locks the resource once.

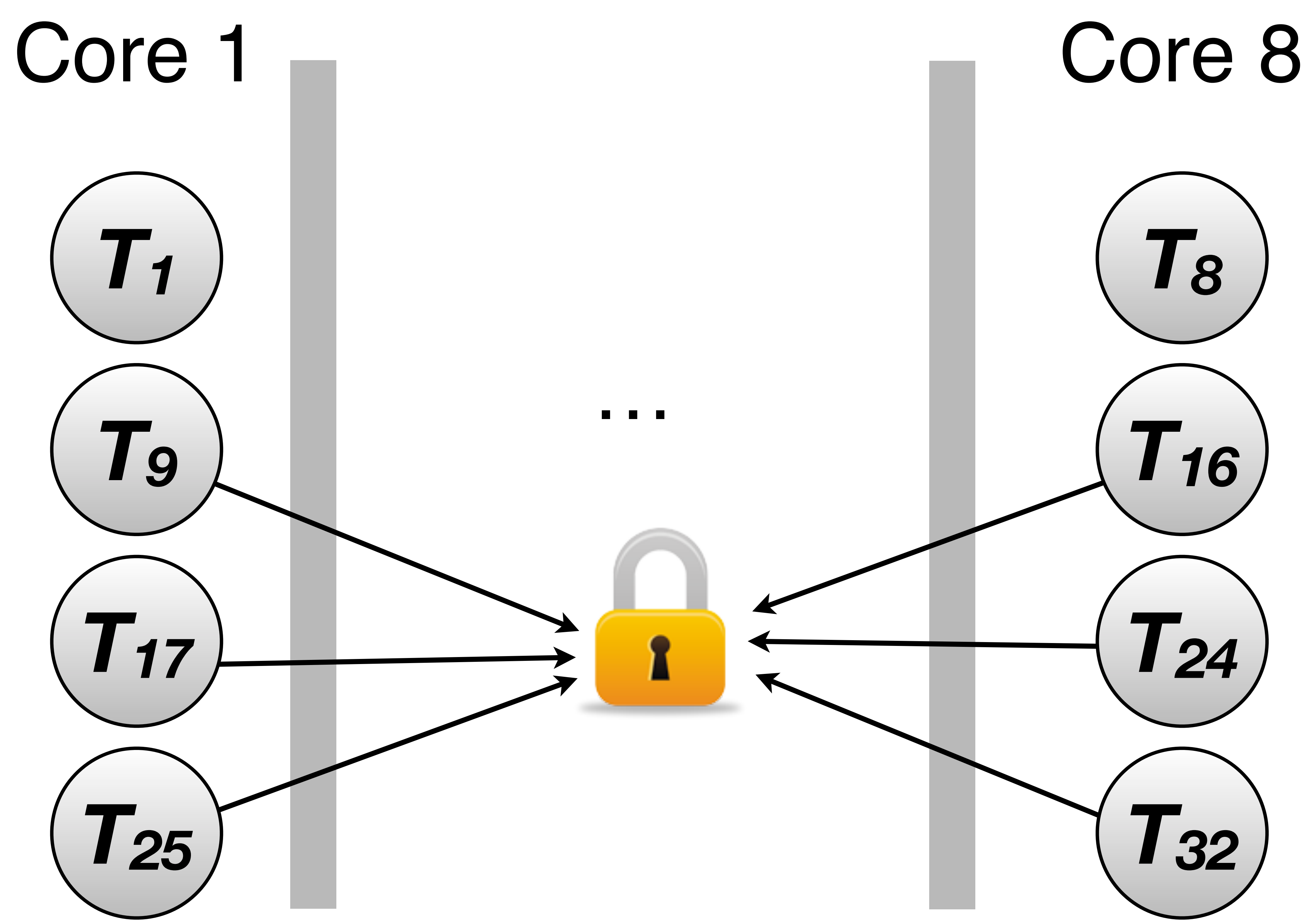


Setup

- ➡ 4 tasks on each core (one independent & latency-sensitive)
- ➡ one shared resource
- ➡ max. critical section length: **~1ms**

Response Times Experiment

on an 8-core, 2-Ghz Xeon X7550 System



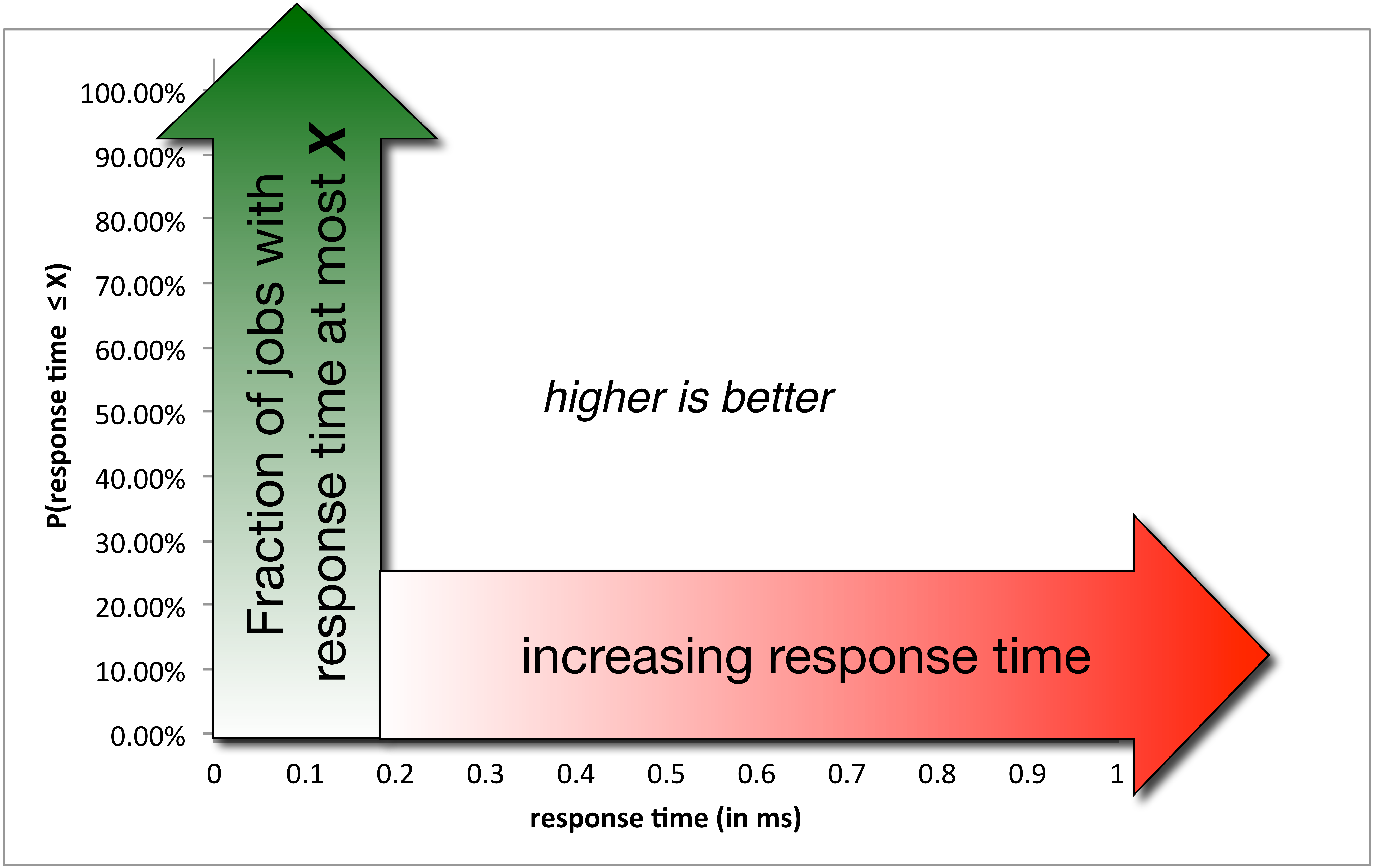
Three Configurations

- ➡ **No locks (unsound!)**
  - no blocking (baseline)
- ➡ **Clustered OMLP**
  - priority donation
- ➡ **OMIP**
  - migratory priority inheritance

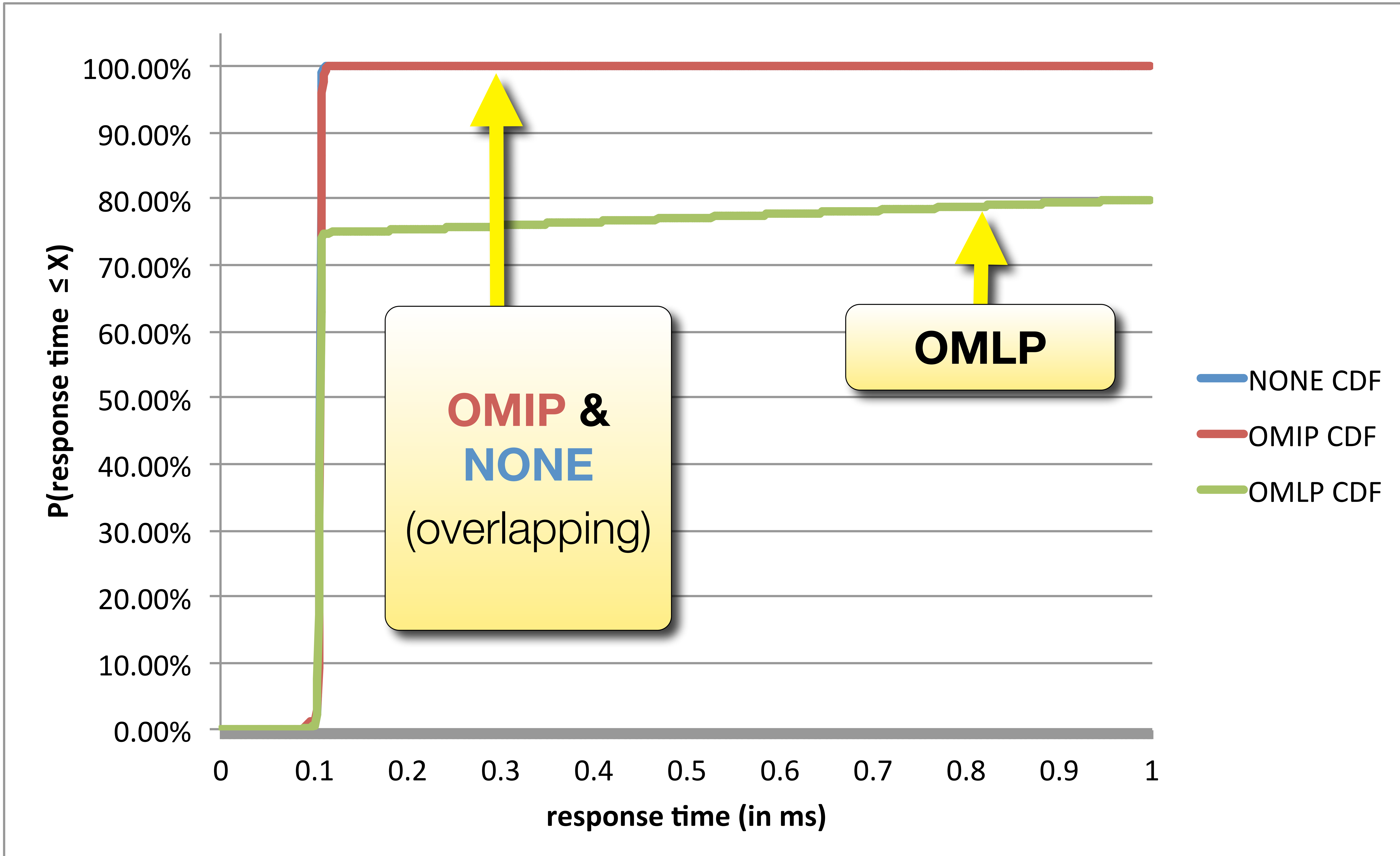
Experiment

- ➡ Measured response times with **`sched_trace`**
- ➡ 30-minute traces
- ➡ more than **45 million jobs**

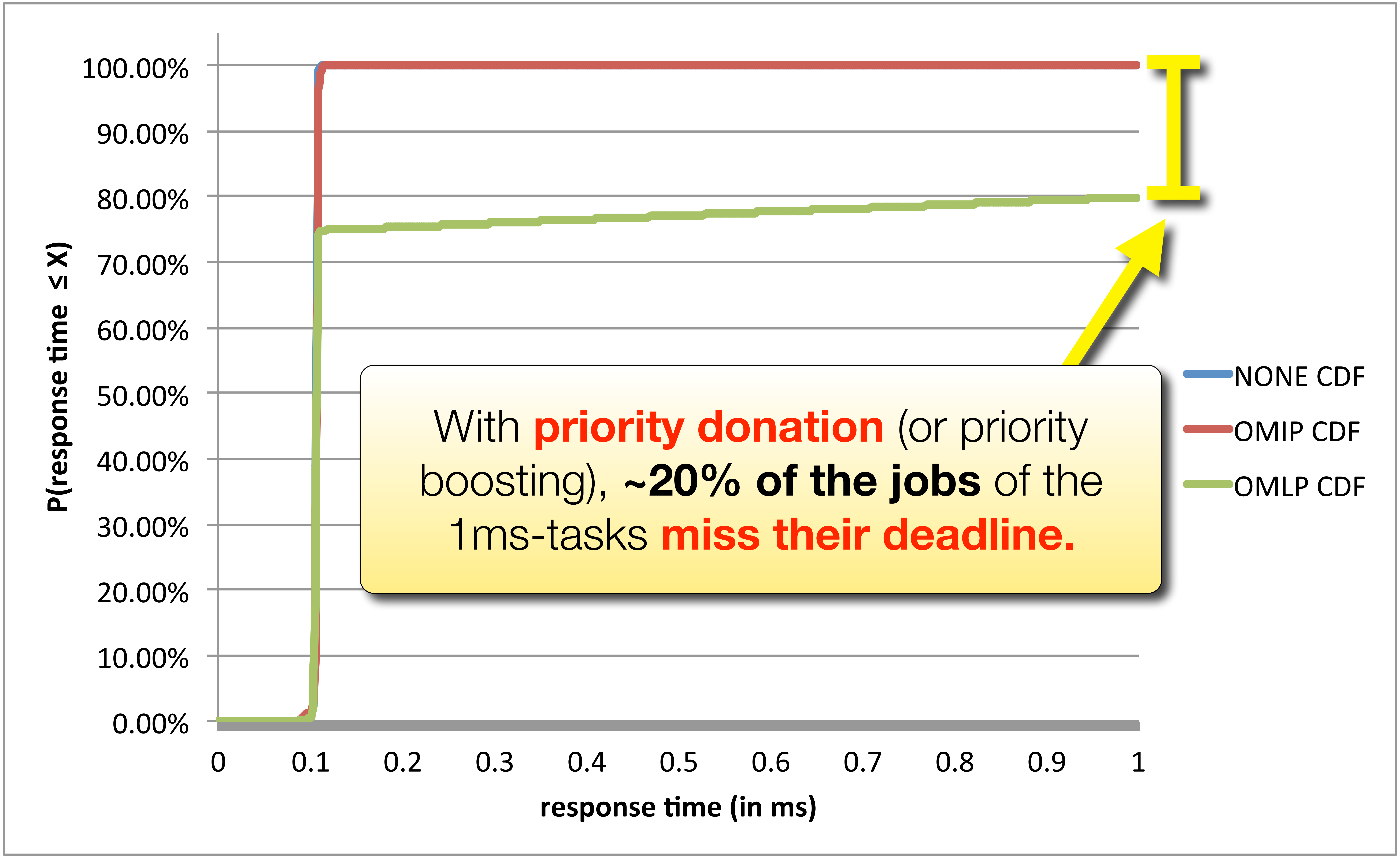
Response Time CDF



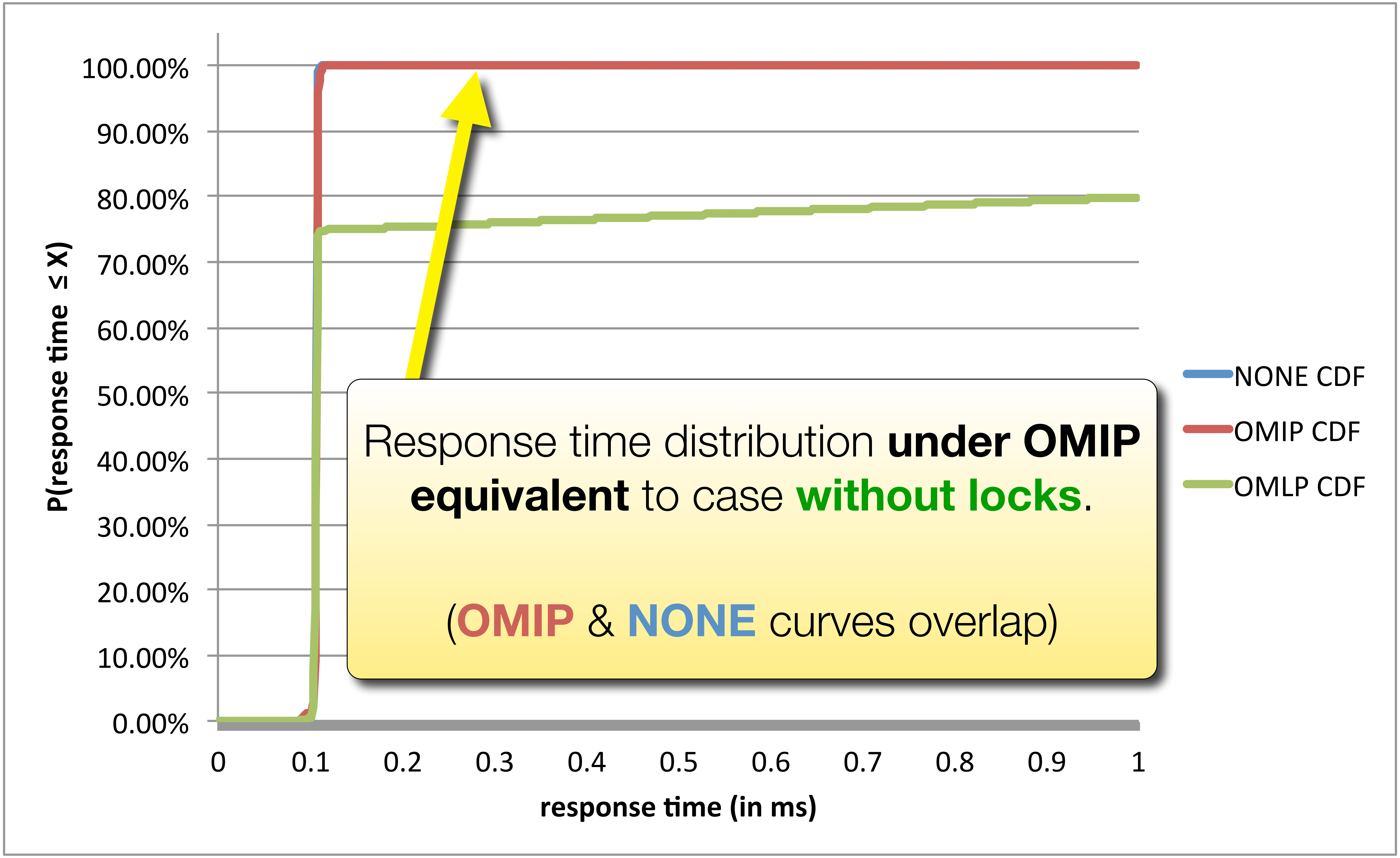
Response Time CDF of 1-ms Tasks



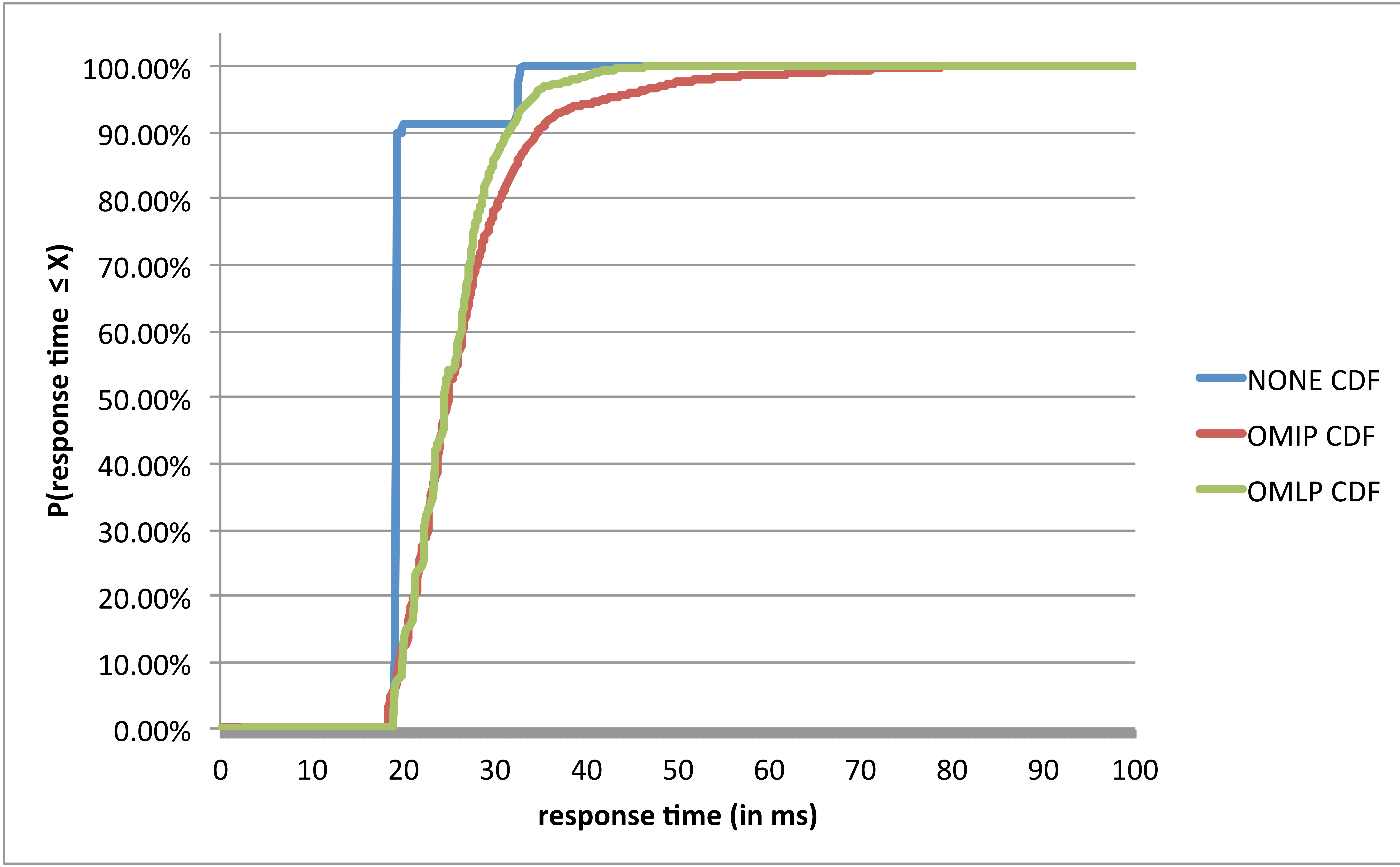
Response Time CDF of 1-ms Tasks



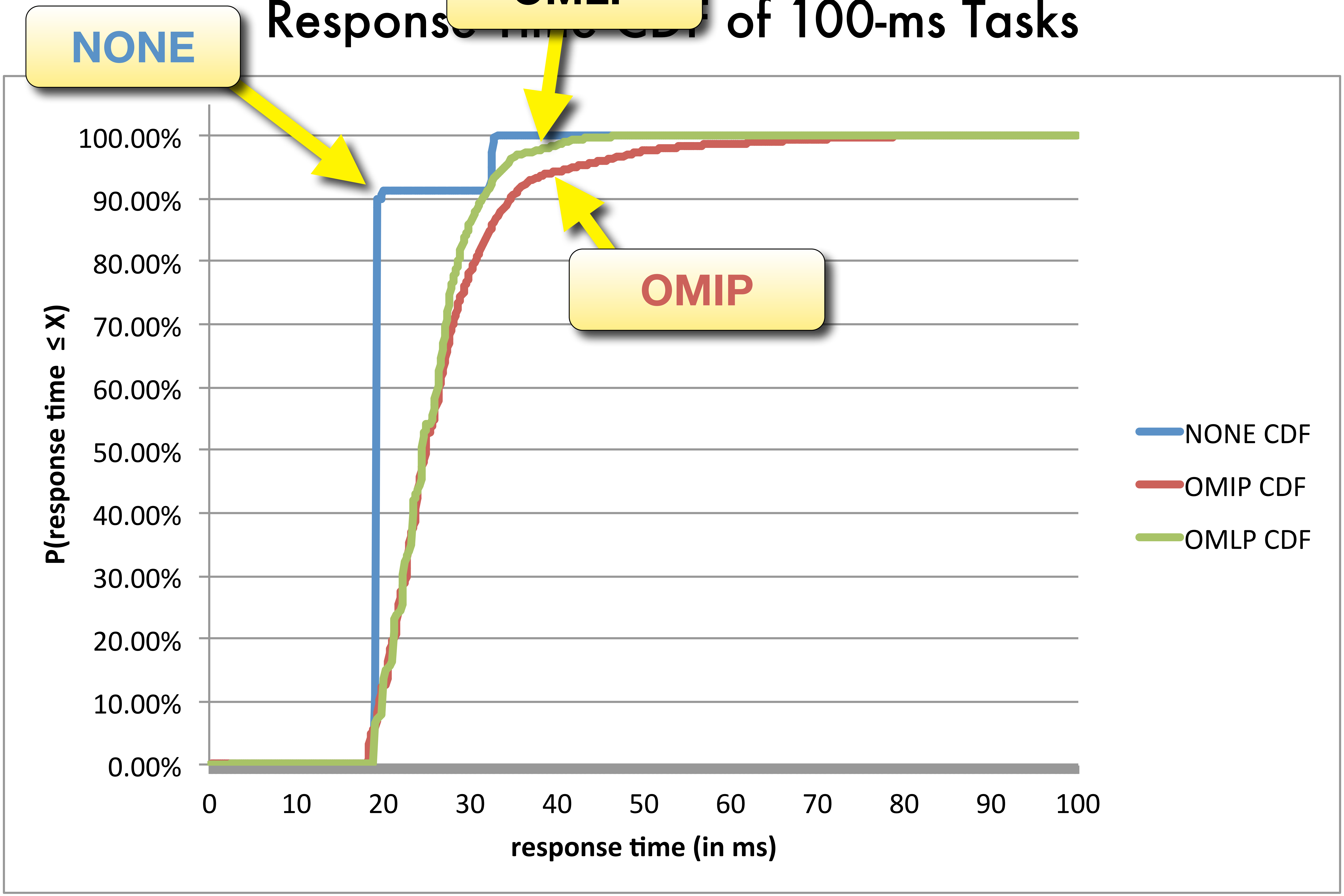
Response Time CDF of 1-ms Tasks



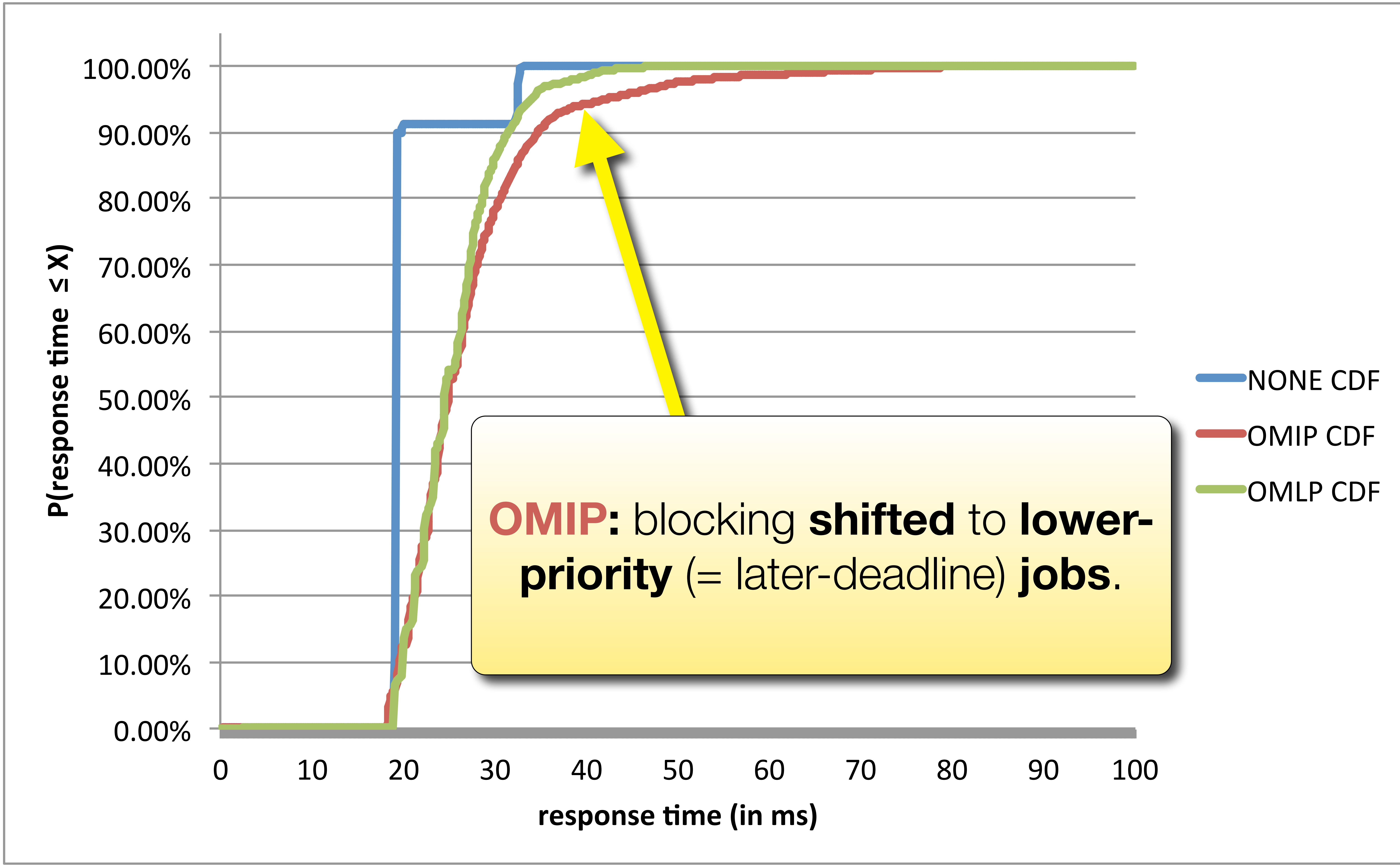
Response Time CDF of 100-ms Tasks



Response-Time CDF of 100-ms Tasks



Response Time CDF of 100-ms Tasks



Analytical Blocking/Latency Tradeoff

Large-scale schedulability experiments

- ➡ Varied #tasks, #cores, #resources, max. critical section lengths, etc.
- ➡ >150,000,000 task sets
- ➡ 678 schedulability plots, available in online appendix

Analytical Blocking/Latency Tradeoff

Large-scale schedulability experiments

- ➡ Varied #tasks, #cores, #resources, max. critical section lengths, etc.
- ➡ >150,000,000 task sets
- ➡ 678 schedulability plots, available in online appendix

*In the presence of **latency-sensitive tasks**, the **OMIP** is generally the **only viable option**.*

Analytical Blocking/Latency Tradeoff

Large-scale schedulability experiments

- Varied #tasks, #cores, #resources, max. critical section lengths, etc.
- >150,000,000 task sets
- 678 schedulability plots, available in online appendix

*In the presence of **latency-sensitive tasks**, the **OMIP** is generally the **only viable option**.*

***Without** latency-sensitive tasks, the **OMIP** does **not** offer substantial improvements.*

Conclusion

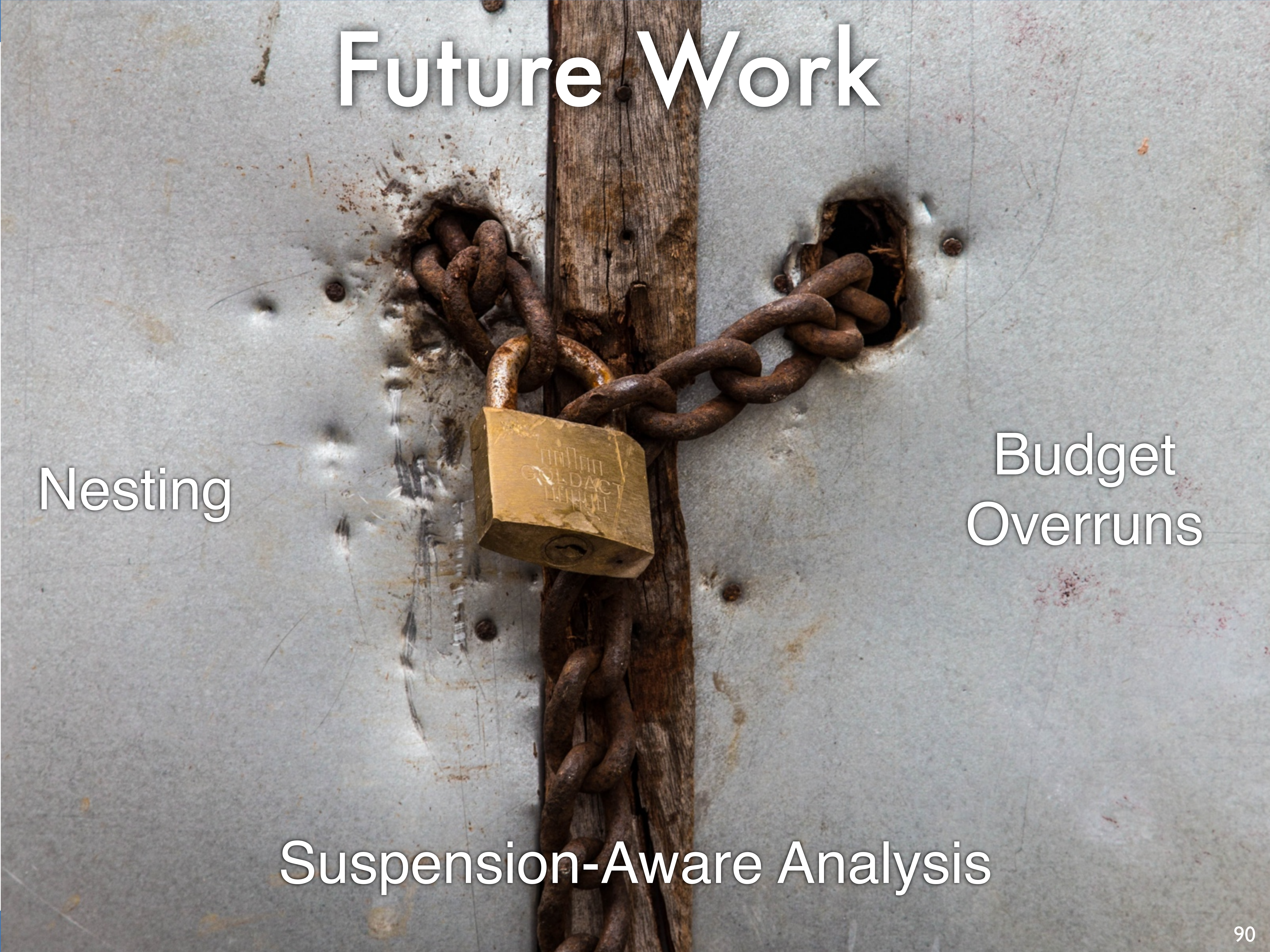
Summary

Independence preservation formalizes the idea that "*tasks should not be delayed by unrelated critical sections.*"

Independence preservation is **impossible** **without (limited) job migrations**.

The **OMIP** is the first independence-preserving semaphore protocol for clustered scheduling. It ensures **asymptotically optimal s-oblivious pi-blocking**.

Future Work



Nesting

Budget
Overruns

Suspension-Aware Analysis

Thanks!

LITMUS^RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

www.litmus-rt.org

SchedCAT

Schedulability test **C**ollection **A**nd **T**oolkit

www.mpi-sws.org/~bbb/projects/schedcat

Appendix

Design Inspirations

Migrate to Blocked Task's CPU

➡ "**Local helping**" in TU Dresden's Fiasco/L4

- *Hohmuth & Peter (2001)*

➡ Multiprocessor **bandwidth inheritance** (MBWI)

- *Faggioli, Lipari, & Cucinotta (2010)*

Queue Design

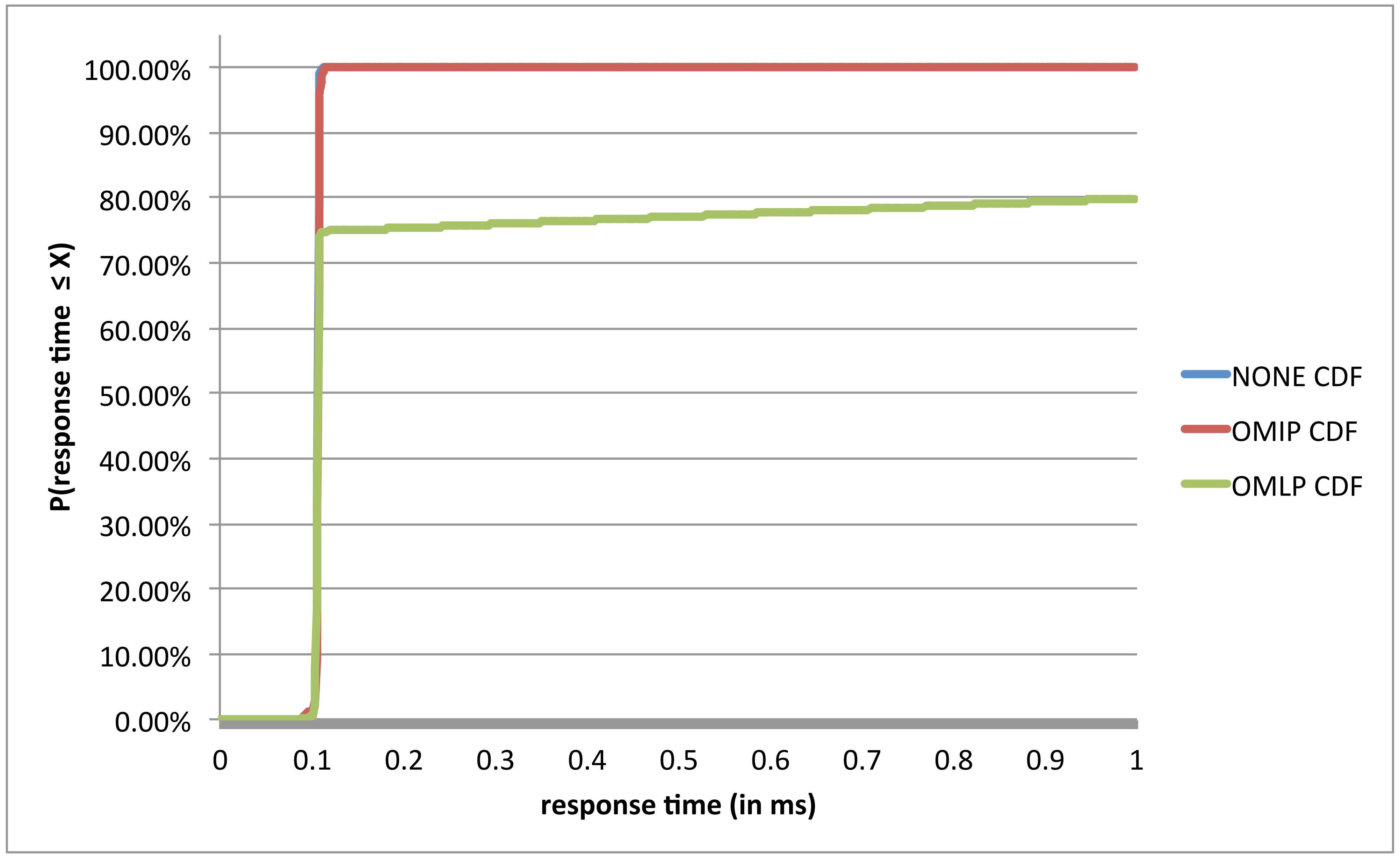
➡ **Intra**-cluster queues adopted from **global OMLP**

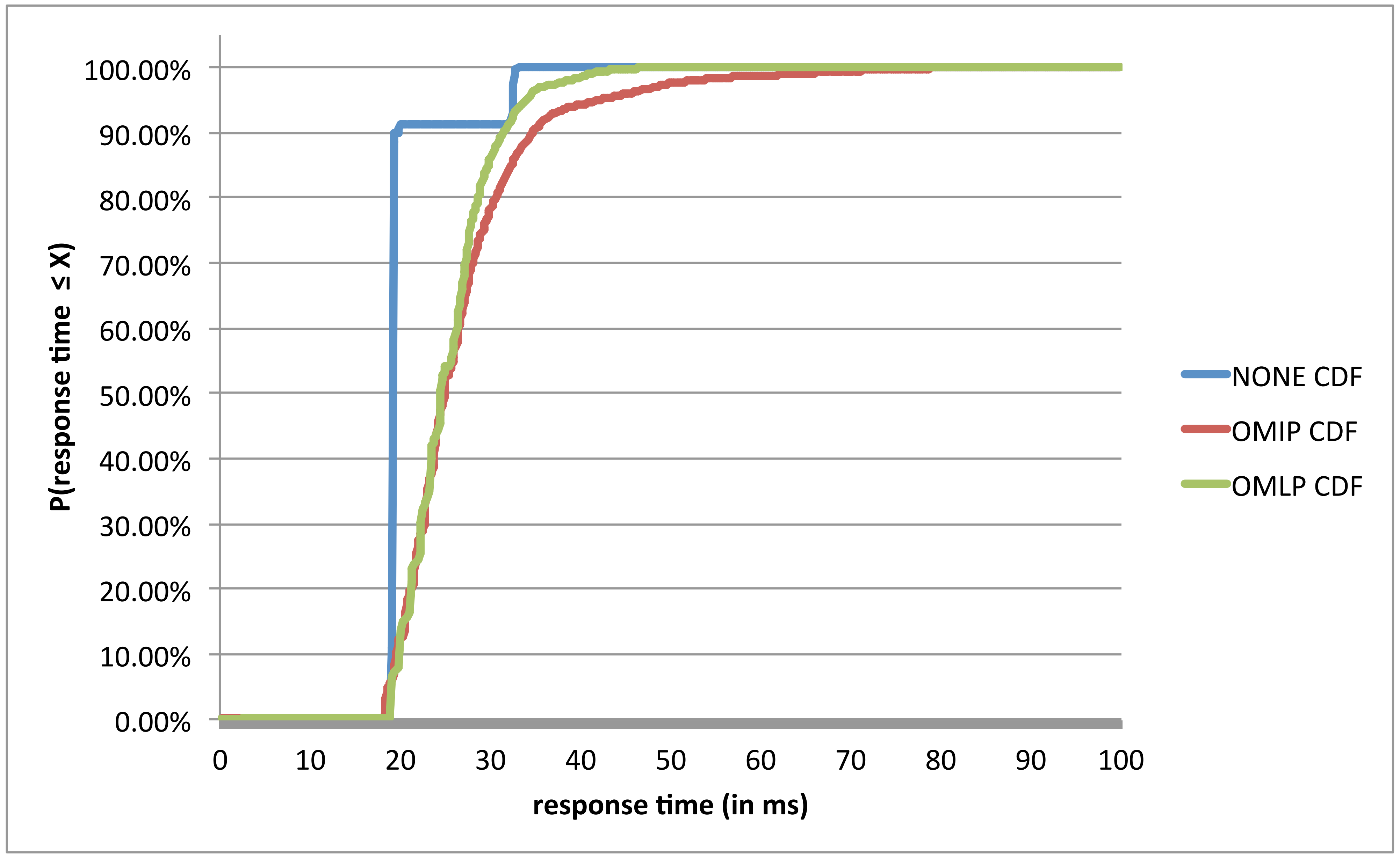
- *— & Anderson (2010)*

➡ **Inter**-cluster queues similar to **clustered OMLP**

- *— & Anderson (2011)*

What about overheads?





Aren't job migrations expensive?

➡ response time experiments **reflect all overheads in real system**

➡ latency-sensitive tasks do not migrate, only lower-priority tasks do

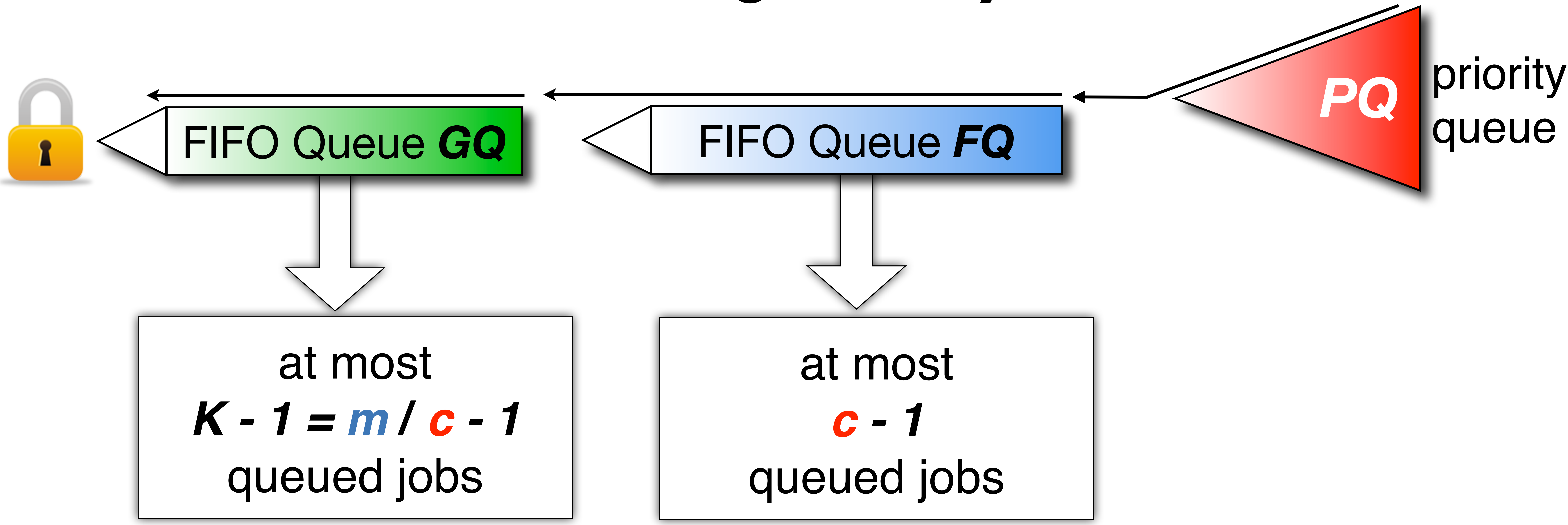
➡ only working set of critical section migrates (likely small), not entire task working set (likely much larger)

➡ the critical section would have been preempted anyway

Blocking Analysis

m *… number of processors (total)* ***c*** *… number of processors per cluster*

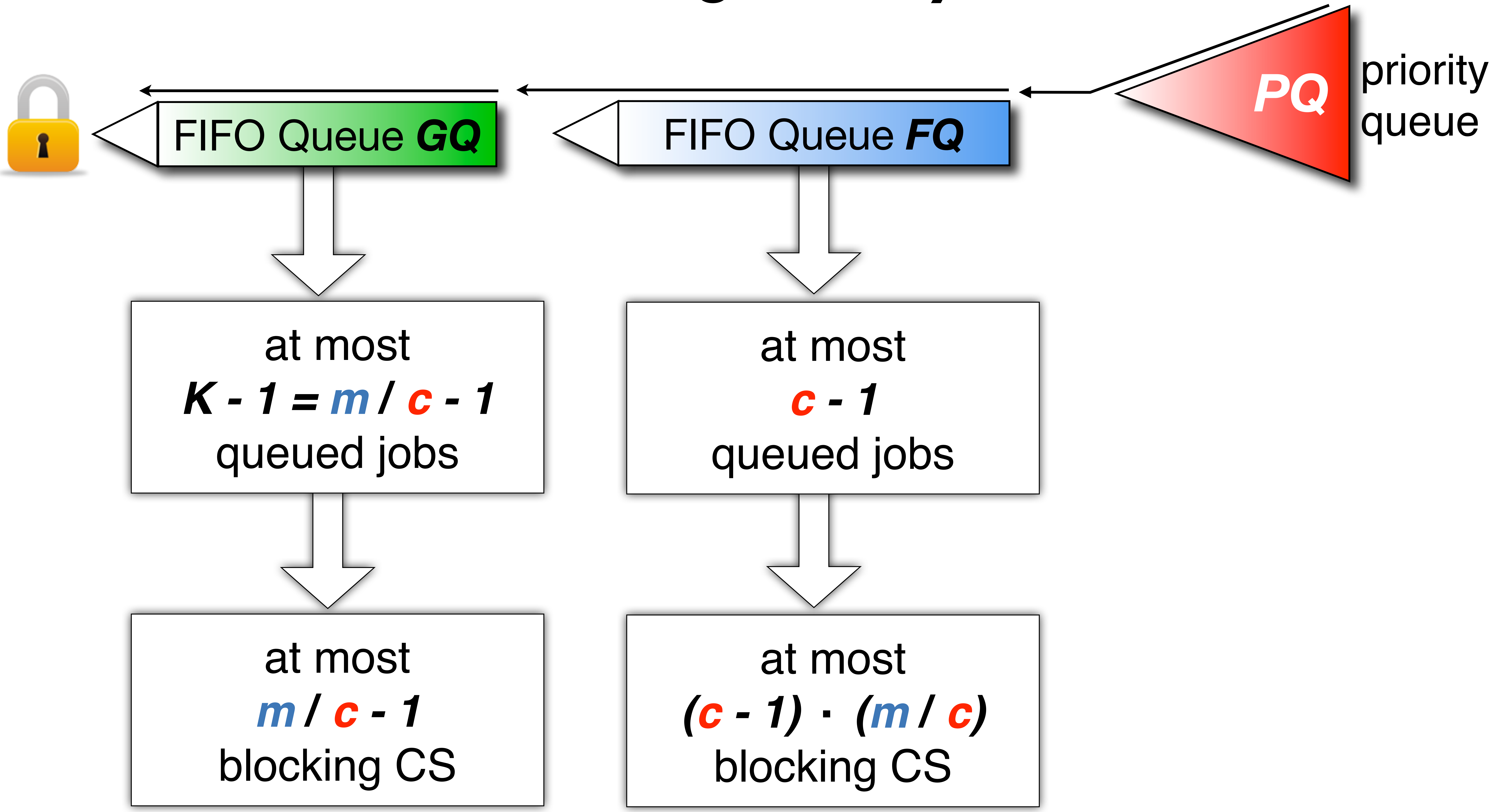
Blocking Analysis



m *… number of processors (total)*

c *… number of processors per cluster*

Blocking Analysis

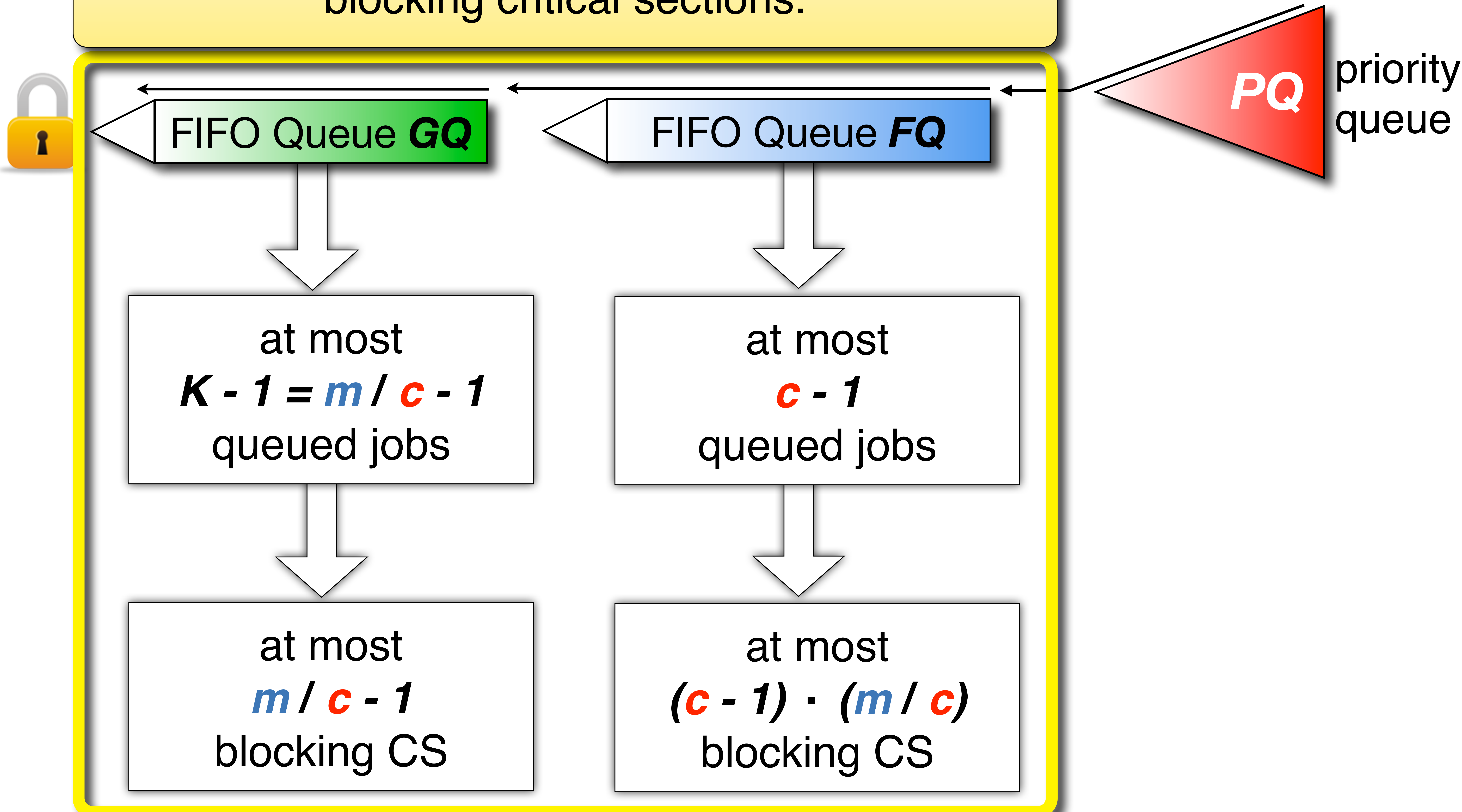


m *… number of processors (total)* ***c*** *… number of processors per cluster*

At most

$m / c - 1 + (c - 1) \cdot (m / c) = m - 1 = O(m)$

blocking critical sections.

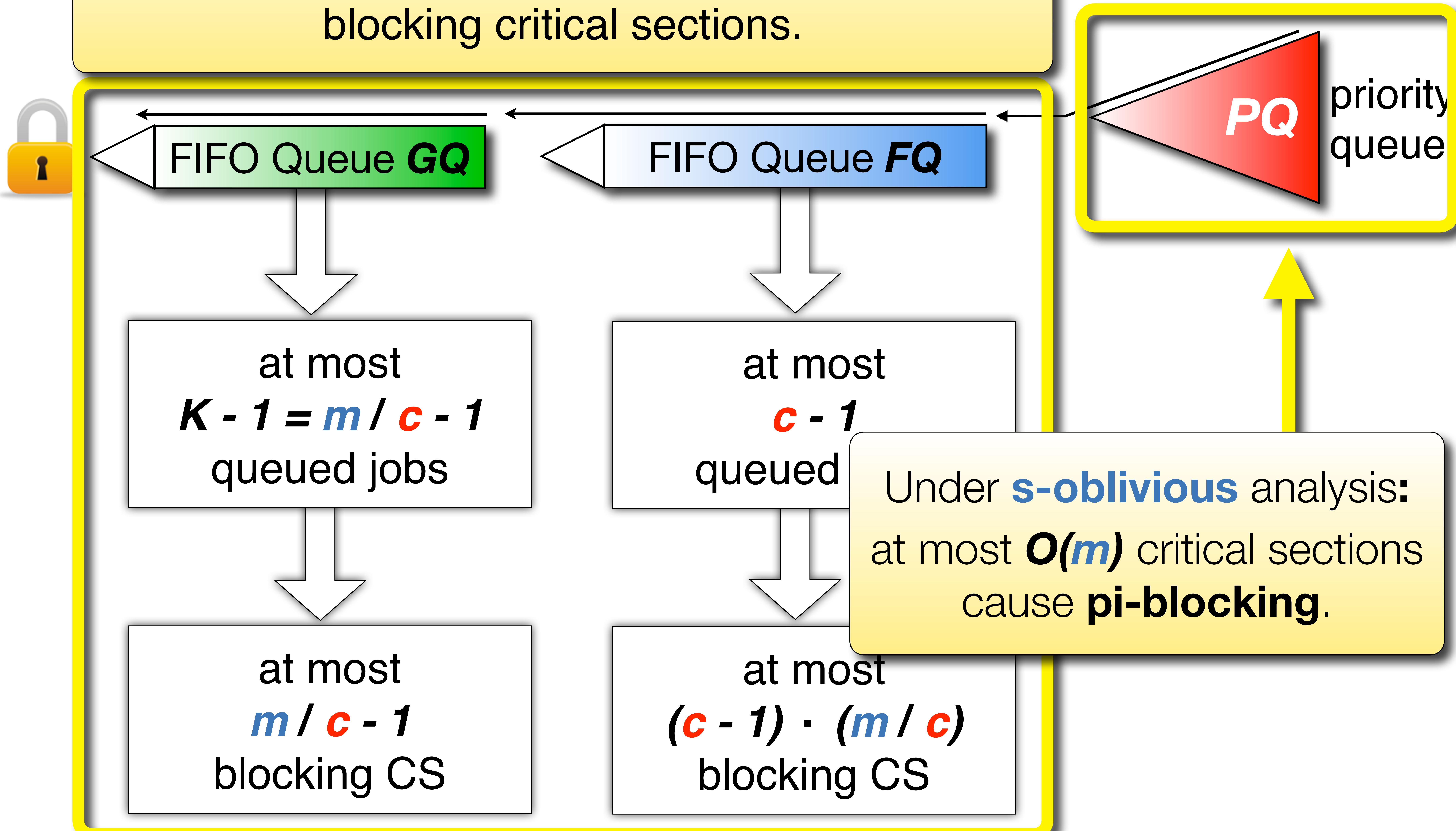


m … number of processors (total)   *c … number of processors per cluster*

At most

$m / c - 1 + (c - 1) \cdot (m / c) = m - 1 = O(m)$

blocking critical sections.



m … number of processors (total) *c … number of processors per cluster*