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

ECRTS'14
July 9, 2014

Max Planck Institute for Software Systems

Björn B. Brandenburg
bbb@mpi-sws.org

Suspension-Based Locking Protocols

Semaphores in POSIX

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

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

Suspension-Based Locking Protocols

Semaphores in POSIX

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

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

Priority Inversion Blocking

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

Suspension-Based Locking Protocols

Semaphores in POSIX

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

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

Priority Inversion Blocking

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

Blocking Analysis

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

Suspension-Based Locking Protocols

Semaphores in POSIX

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

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

Priority Inversion Blocking

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

Blocking Analysis

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

Blocking Optimality

- In general, what is the **maximum duration of pi-blocking** incurred by **any task in any task set**?

Multiprocessor Real-Time Locking Optimality Classes

Blocking Optimality [— & Anderson, 2010]	suspension **oblivious**	suspension **aware**
How are suspensions analyzed?	CPU demand is **over-approximated**	CPU demand is modeled **accurately**
Advantage	simpler analysis	potentially less pessimistic

[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.

Asymptotically Optimal Locking Protocols

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware EDF w/ Implicit Deadlines	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)			
Global (jobs migrate freely)			
Clustered (jobs migrate only among subset of processors)			

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Asymptotically Optimal Locking Protocols

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware EDF w/ Implicit Deadlines	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]		
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010]		
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011]		

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Asymptotically Optimal Locking Protocols

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware EDF w/ Implicit Deadlines	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [—, 2011]	
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010]		
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011]		

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Asymptotically Optimal Locking Protocols

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware EDF w/ Implicit Deadlines	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** ✓ [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [—, 2011] ✓	
Global (jobs migrate freely)	**G-OMLP** ✓ [— & Anderson, 2010]	**FMLP** ! [Block et al., 2007]	?
Clustered (jobs migrate only among subset of processors)	**OMIP** ✓ [—, 2013] **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

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

JLFP *job-level fixed-prior*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware EDF w/ Implicit Deadlines	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010] ✓	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [—, 2011] ✓	
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010] ✓	**FMLP** [Block et al., 2007] !	?
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011] ✓	?	?

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Asymptotically Optimal Locking Protocols

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	**This Work** The Generalized FMLP⁺ (FIFO Multiprocessor Locking Protocol) + RNLP [Ward & Anderson, 2012] for nested critical sections
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011] + RNLP [Ward & Anderson, 2012]	

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Remainder of This Talk

What is Suspension-Aware PI-Blocking?

➡ Assumptions & quick review

Finding a Suitable Progress Mechanism

➡ How to deal with lock-holder preemptions

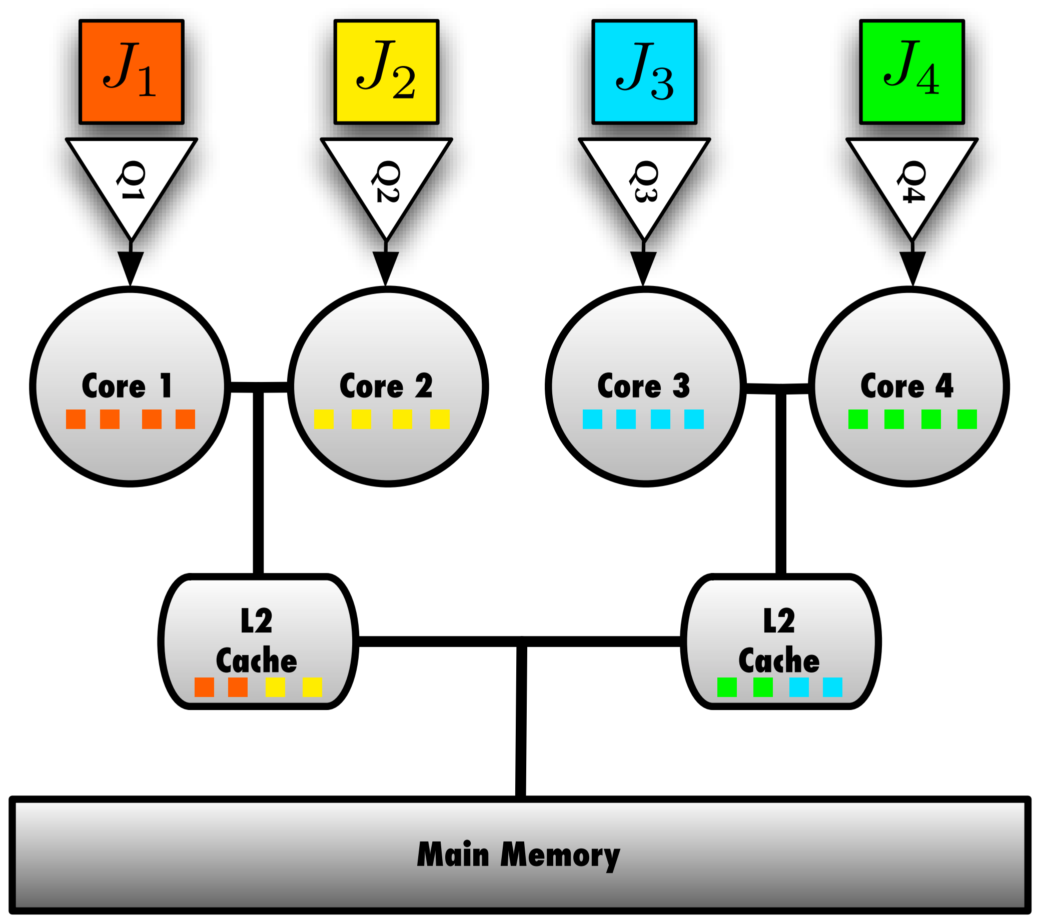
Closing the Suspension-Aware Optimality Gap

➡ New progress mechanism: **restricted segment boosting**

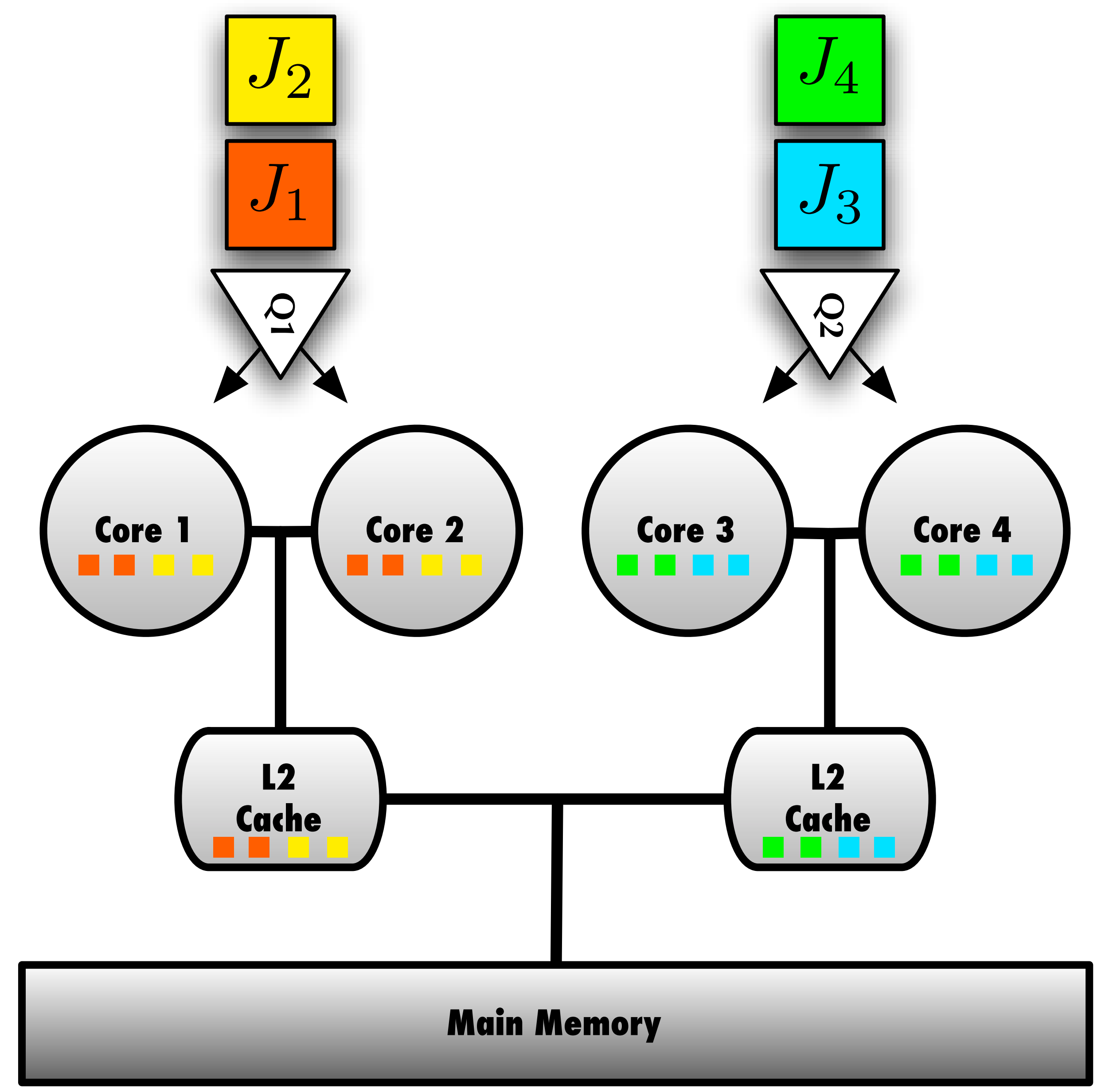
➡ Achieving asymptotic optimality with the **generalized FMLP⁺**

<u>Assumptions</u> & <u>Review</u> of Suspension-Aware PI-Blocking

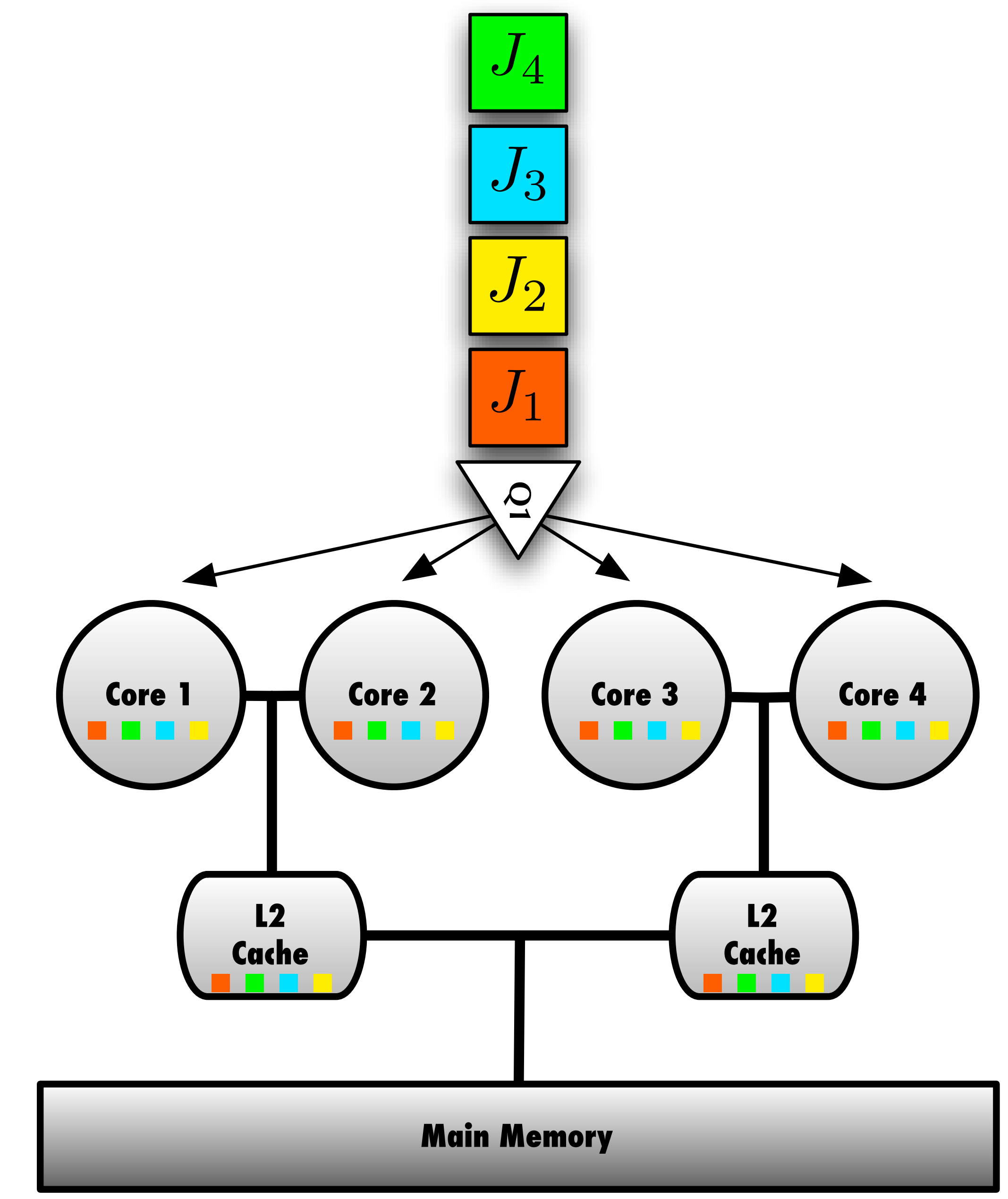
System Model



partitioned scheduling



<u>**clustered scheduling**</u>

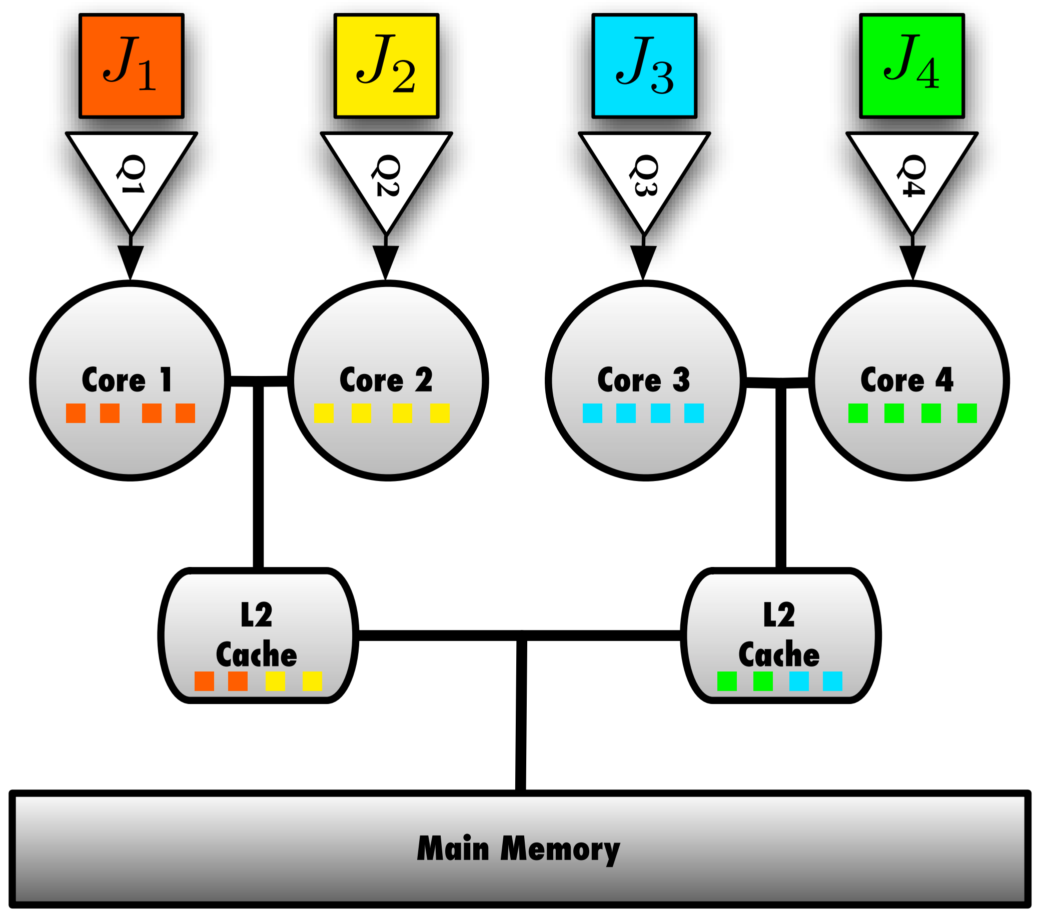


global scheduling

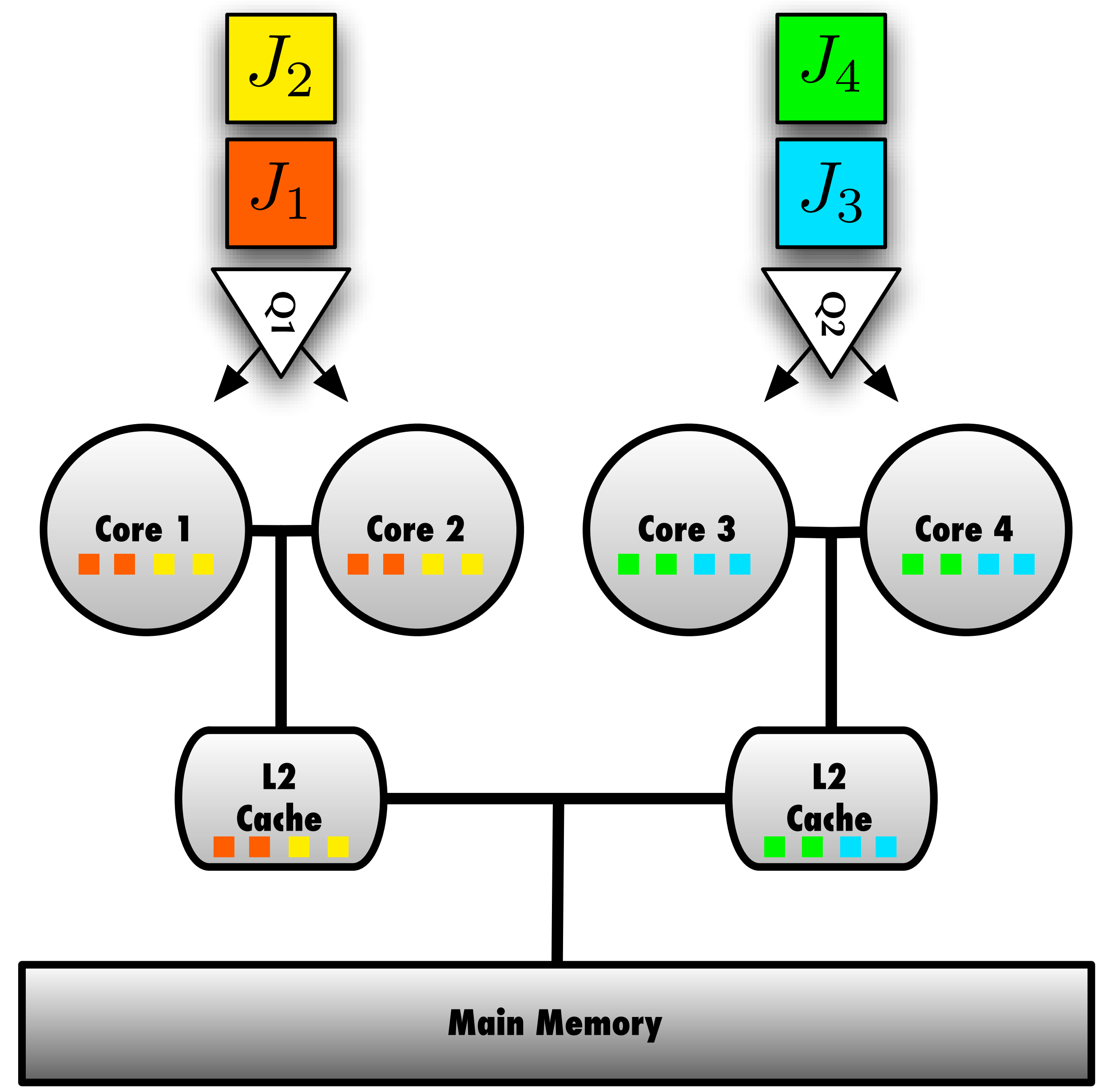
Clustered Scheduling

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

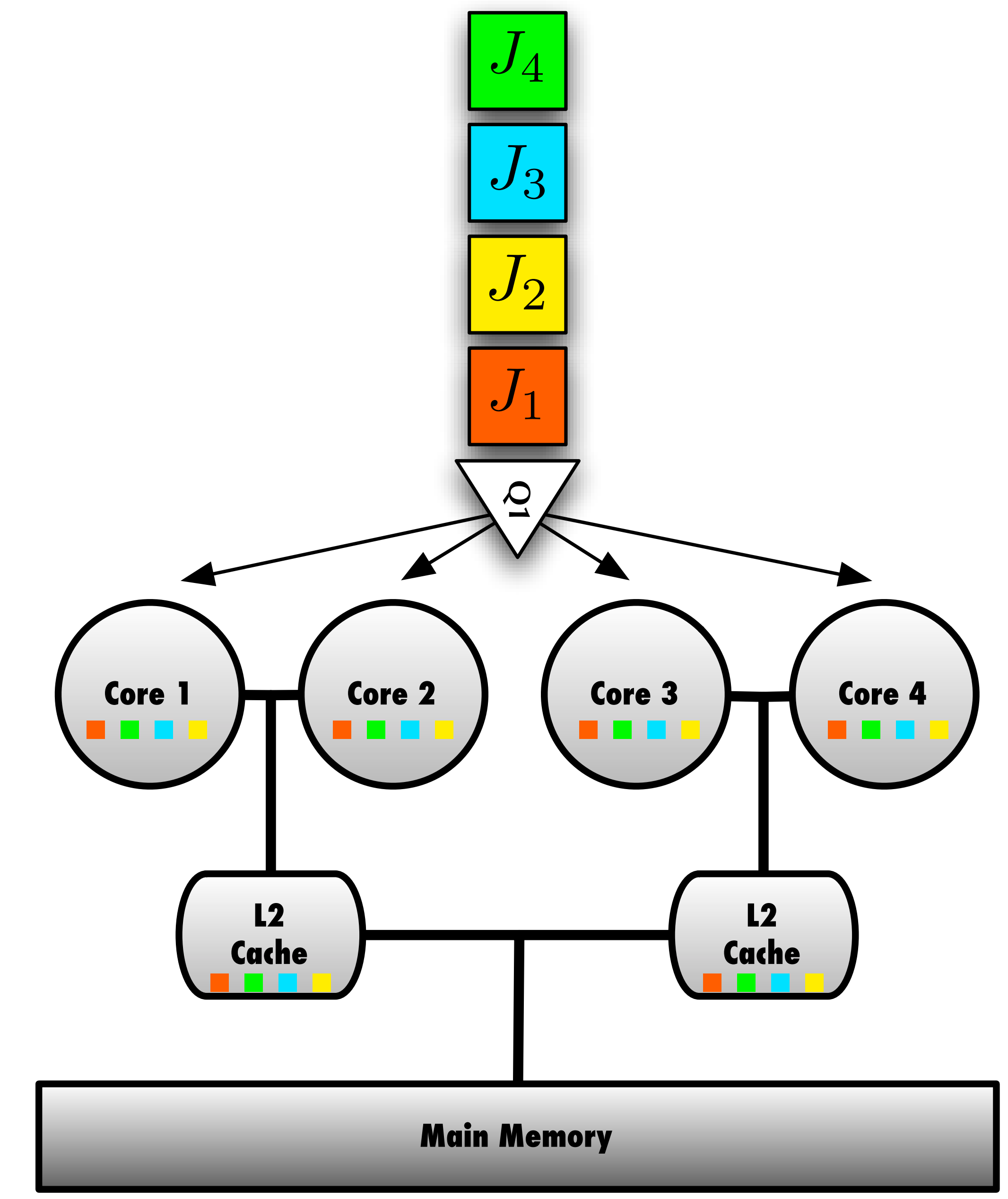
System Model



partitioned scheduling



<u>clustered scheduling</u>



global scheduling

Clustered Scheduling

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

Sporadic Tasks

- **arbitrary** deadlines
- shared resources
  - in the paper: also nested CSs
  - in the talk: <u>only unnested CSs</u>
- *locking-unrelated self-suspensions*

Definition: S-Aware PI-Blocking

*A job **J** assigned to a cluster with **c** CPUs incurs **s-aware pi-blocking** at a time **t** iff*

(1) ***J*** *is* ***not scheduled*** *at time* ***t****, and*

(2) *fewer than* ***c*** ***higher-priority jobs*** *are scheduled.*

[— & Anderson, 2010]

Intuition

➡ Locking-related delays are **not problematic** iff ***J*** would **not** have been scheduled anyway…

[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.

Maximum PI-Blocking

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

$\max\{b_i\}$ — **maximum pi-blocking** of any task in task set

Maximum PI-Blocking

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

$\max\{b_i\}$ — **maximum pi-blocking** of any task in task set

There exist task sets such that under s-aware analysis

$$\max\{b_i\} = \Omega(n)$$

*under **any** suspension-based locking protocol.*

[— & Anderson, 2010]

(assuming constant critical section lengths)

[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.

Maximum PI-Blocking

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

$\max\{b_i\}$ — **maximum pi-blocking** of any task in task set

There exist task sets such that under s-aware analysis

$$\max\{b_i\} = \Omega(n)$$

*under **any** suspension-based locking protocol.*

[— & Anderson, 2010]

(assuming constant critical section lengths)

→ $O(n)$ max. s-aware pi-blocking is asymptotically optimal.

[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.

Objective

Define a locking protocol such that

$$\mathit{max}\,\{b_i\} = O(n)$$

for **any task set** under **any clustered JLFP scheduler**.

Need to define queue order

➡ FIFO works

Need to define progress mechanism

➡ To deal with risk of lock-holder preemption

➡ Ensure timely completion of critical sections

➡ Classic example: priority inheritance

Finding a Suitable Progress Mechanism

Progress Mechanism Choices

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware EDF w/ Implicit Deadlines	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [—, 2011]	
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010]	**FMLP** [Block et al., 2007]	?
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Progress Mechanism Choices

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

job-level fixed priority	Any JLFP Scheduler	Deadlines	Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP⁺** (practical protocol) [—, 2011]	
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010]	**FMLP** [Block et al., 2007]	?
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Progress Mechanism Choices

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

job-l... fixed priority	Any JLFP Scheduler	Deadlines	Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [— 2011]	
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010]	**FMLP** [Block et al., 2007]	?
Clustered (jobs migrate only among subset of processors)	**OMIP** [— 2013] **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] ... *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.

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

Sub-Optimality of Priority Inheritance

It is ***impossible to construct***

an ***asymptotically optimal*** *locking protocol*

(w.r.t. s-aware analysis)

under global JLFP scheduling

based on ***priority inheritance****.*

[—, 2011]

(And hence also under clustered JLFP scheduling.)

[—, 2011]: *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, University of North Carolina at Chapel Hill, 2011.

Sub-Optimality of Priority Inheritance

Priority Inheritance Example Schedule

4 tasks on 2 processors

global fixed-priority scheduling

Task priorities (high to low): $T_1 > T_2 > T_3 > T_4$

	scheduled	critical section
Processor 1		
Processor 2		

job release

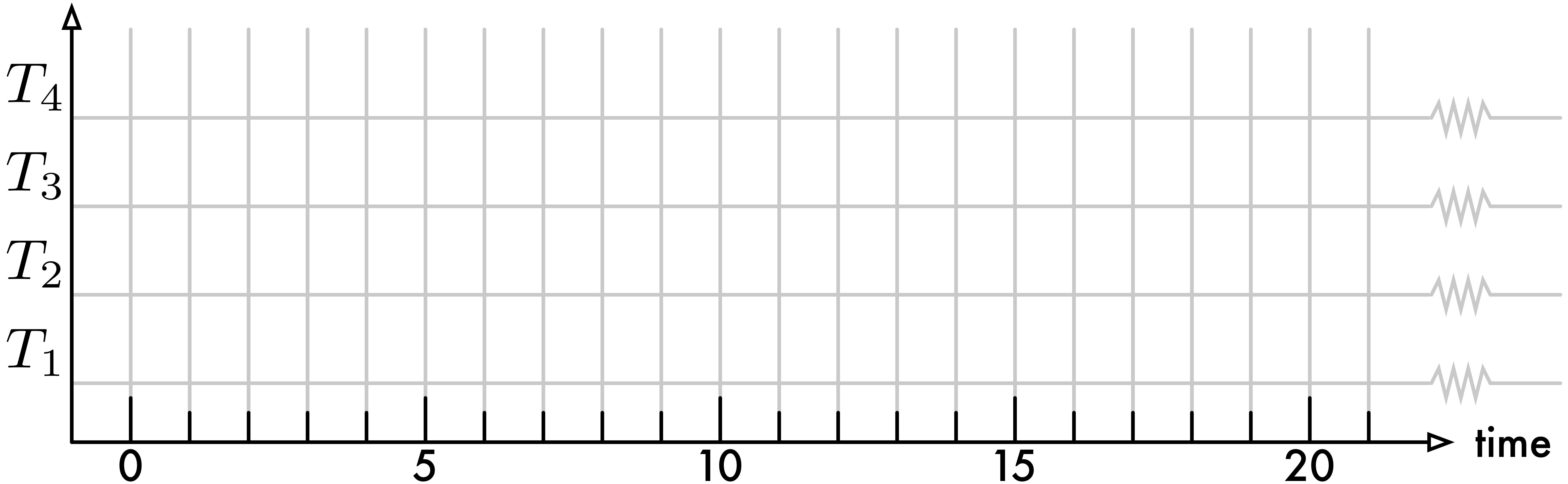
deadline

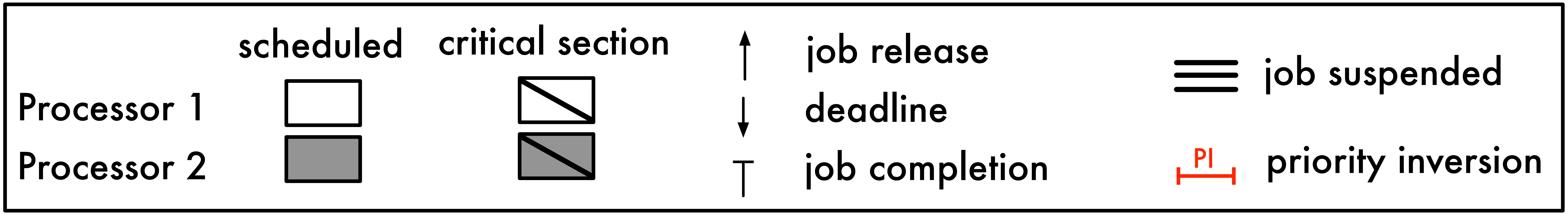
job completion

job suspended

PI priority inversion

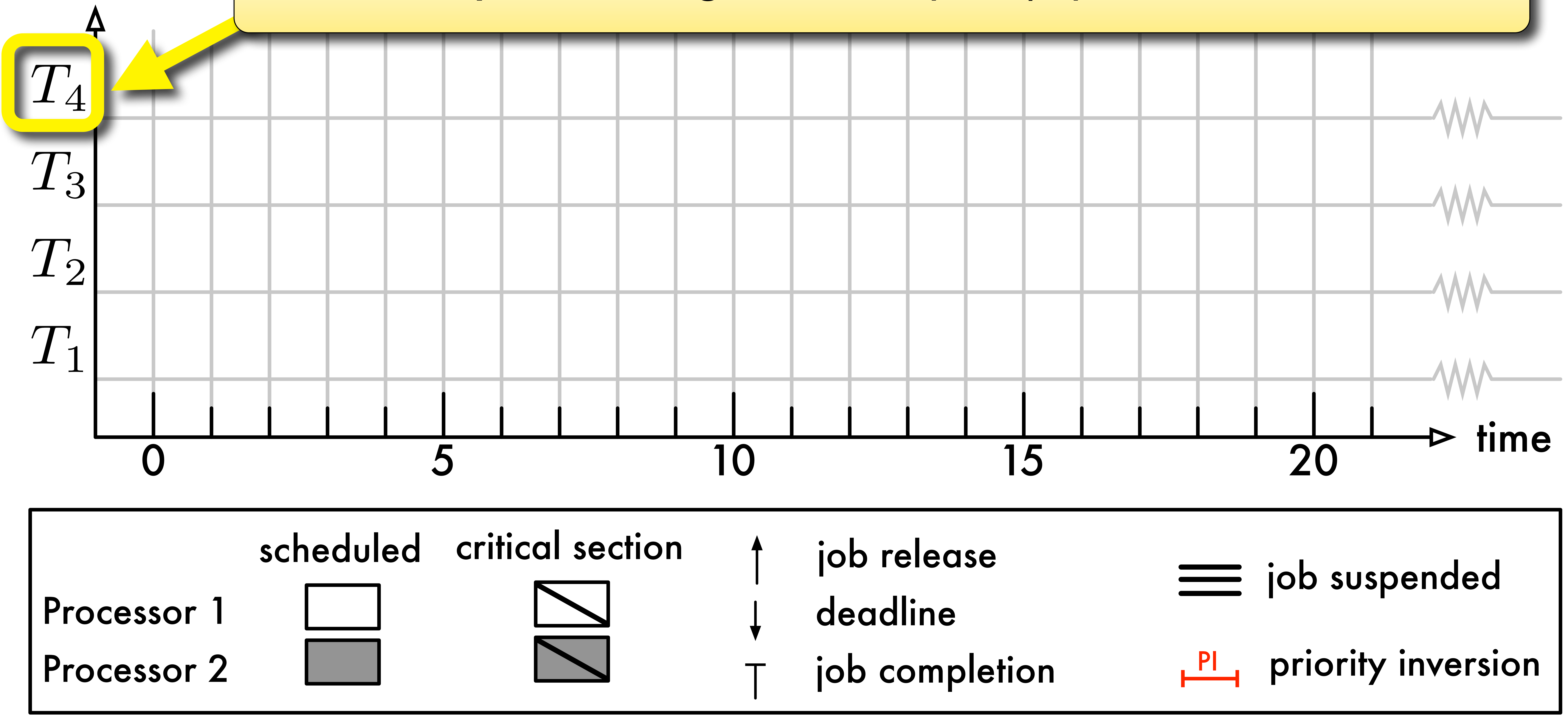
Sub-Optimality of Priority Inheritance



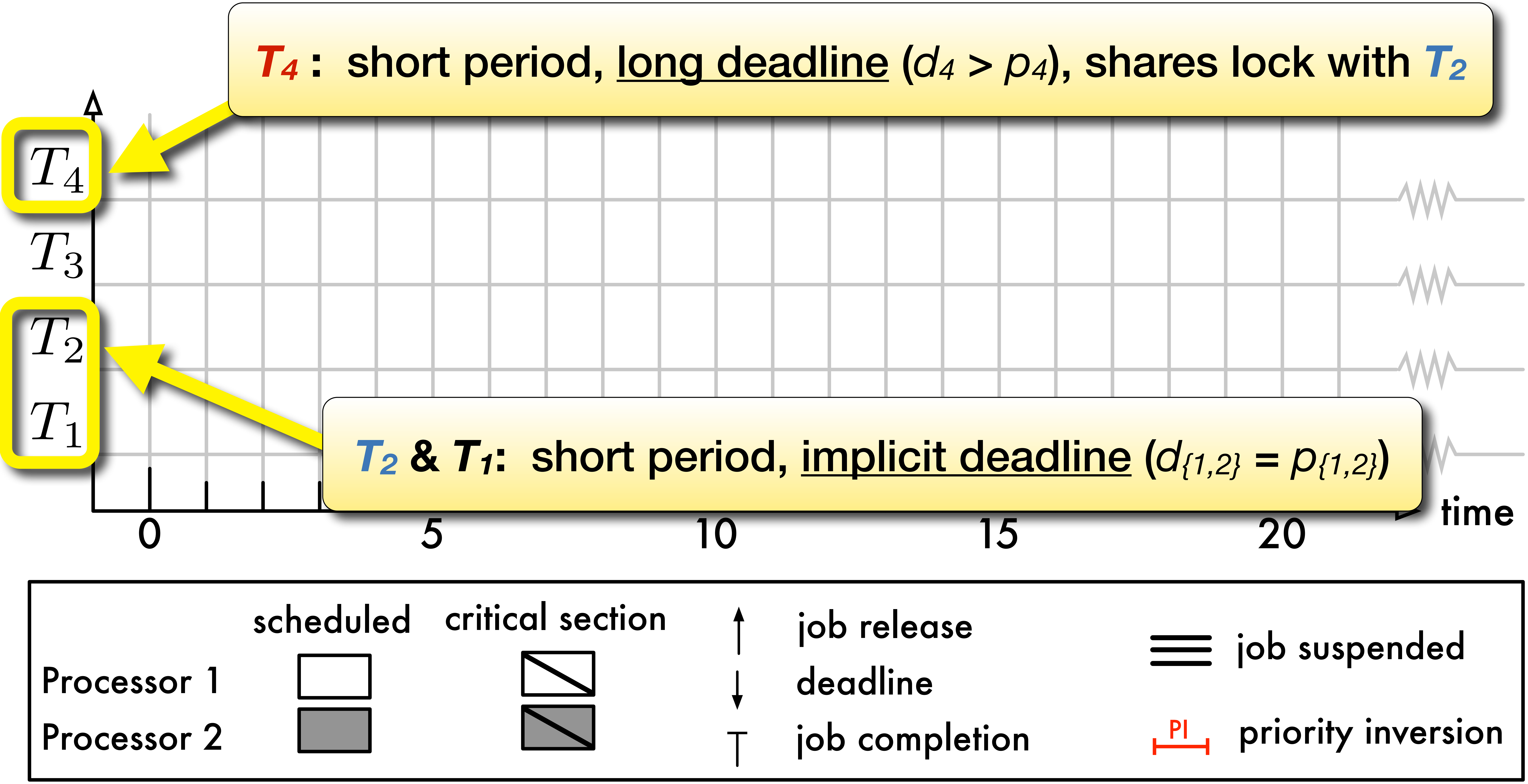


Sub-Optimality of Priority Inheritance

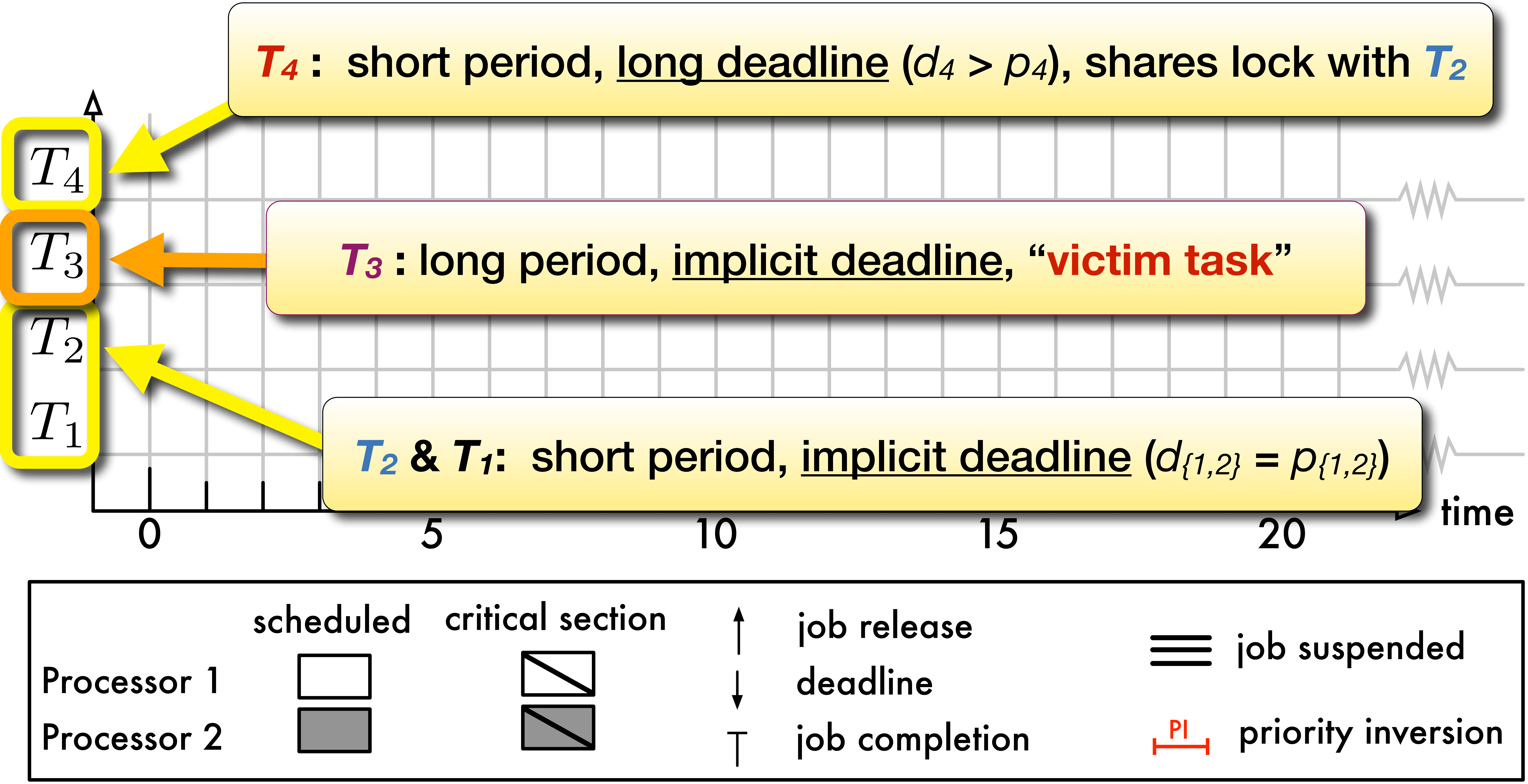
T_4 : short period, long deadline ($d_4 > p_4$), shares lock with T_2



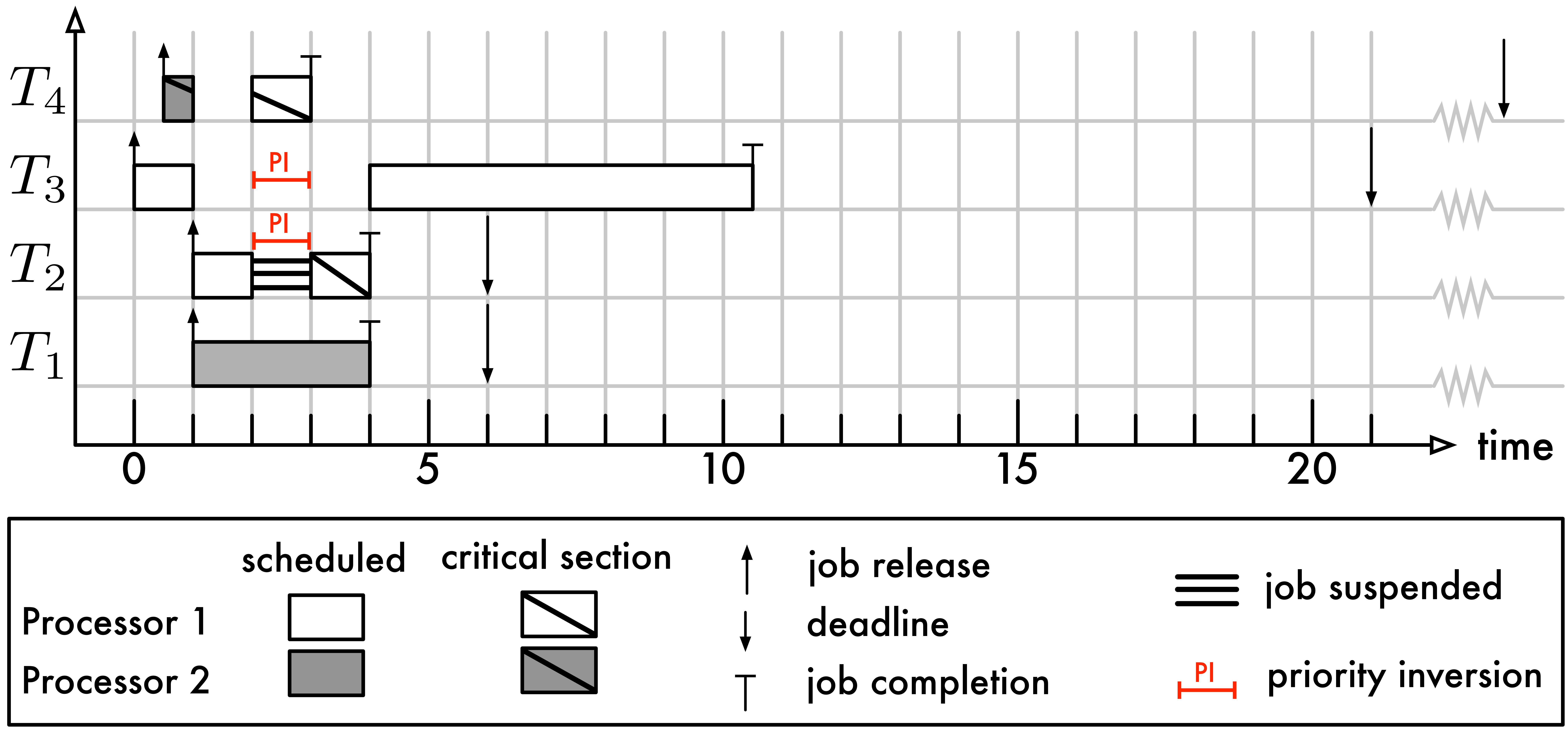
Sub-Optimality of Priority Inheritance



Sub-Optimality of Priority Inheritance

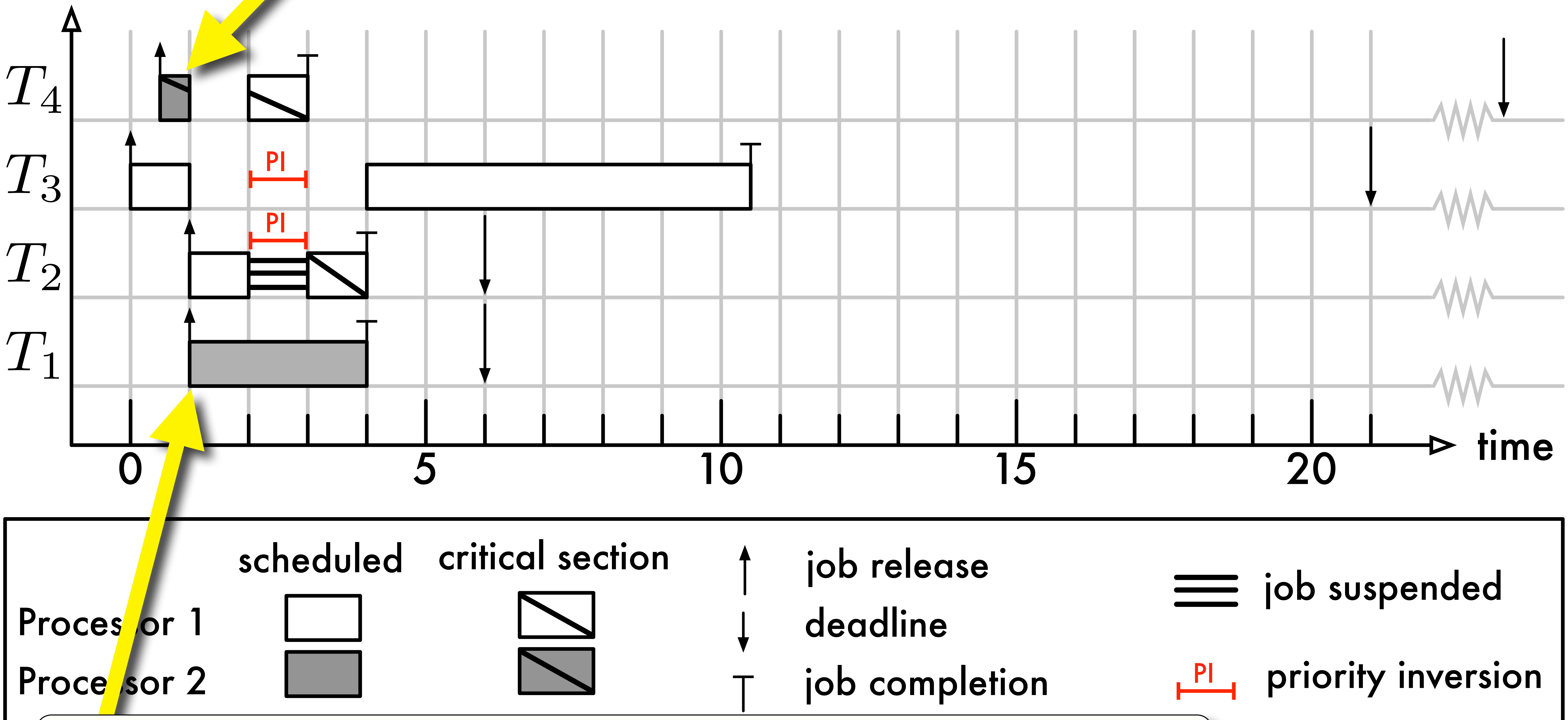


Independent Job Incurs Priority Inversion



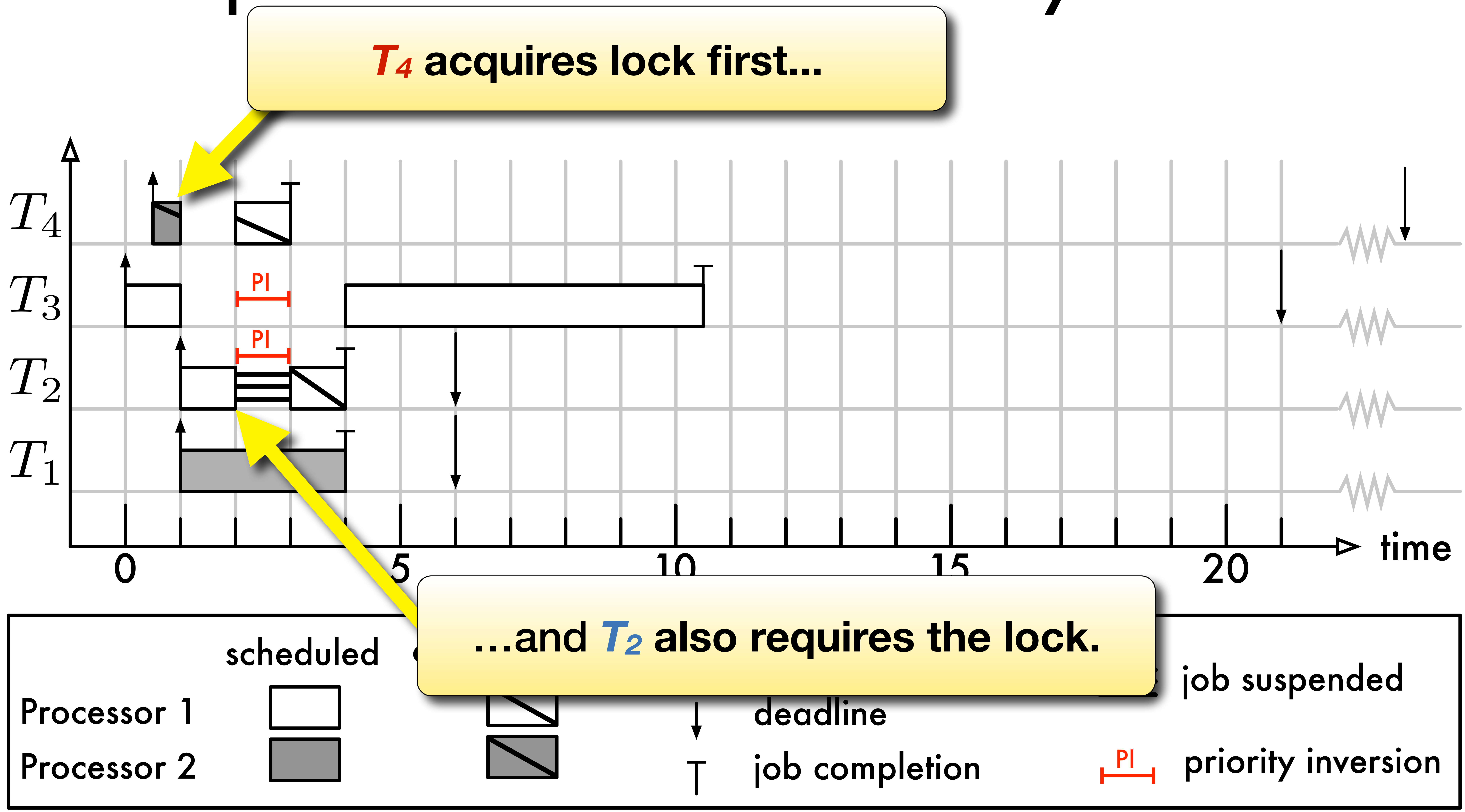
Independent Job Incurs Priority Inversion

T_4 acquires lock first...



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

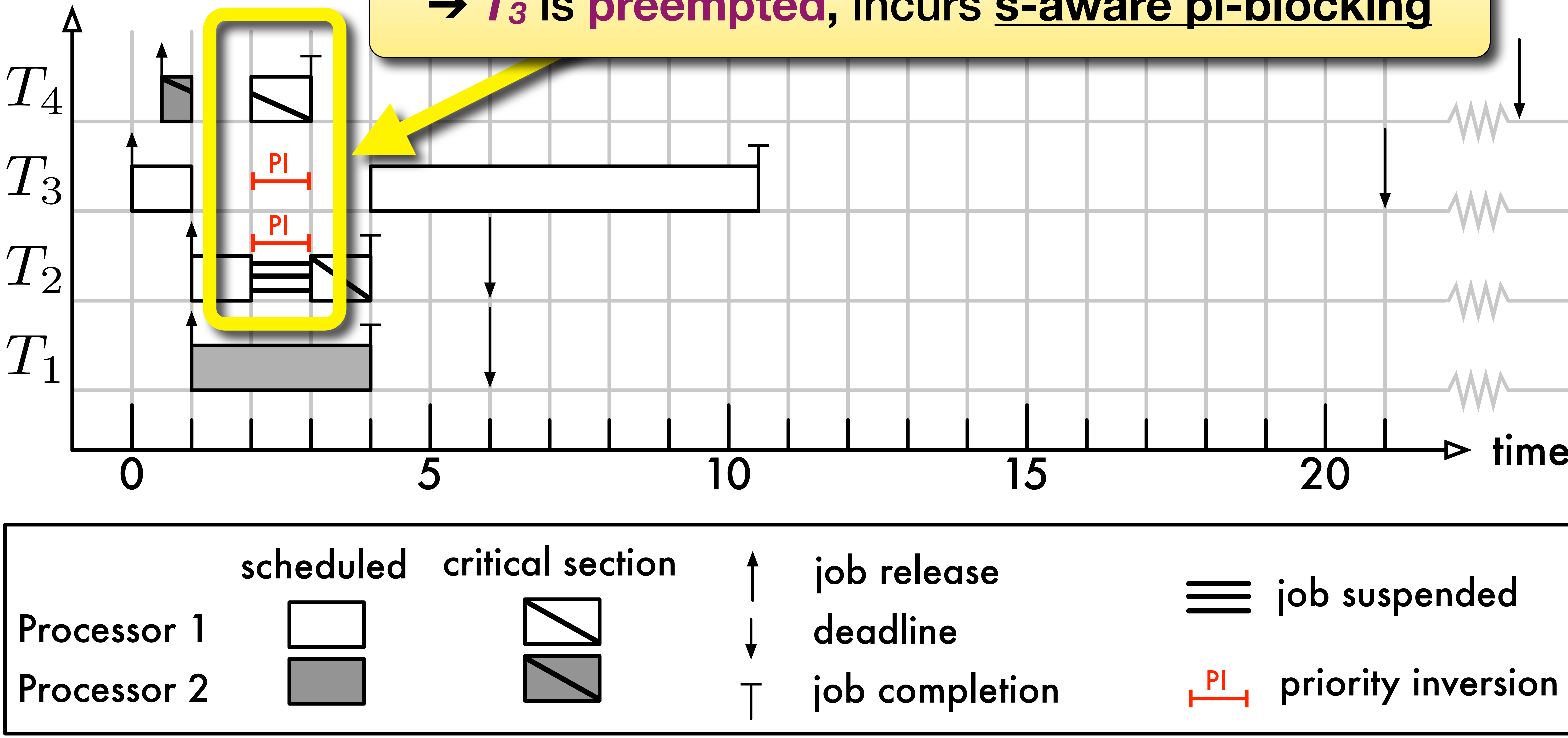
Independent Job Incurs Priority Inversion



Independent Job Incurs Priority Inversion

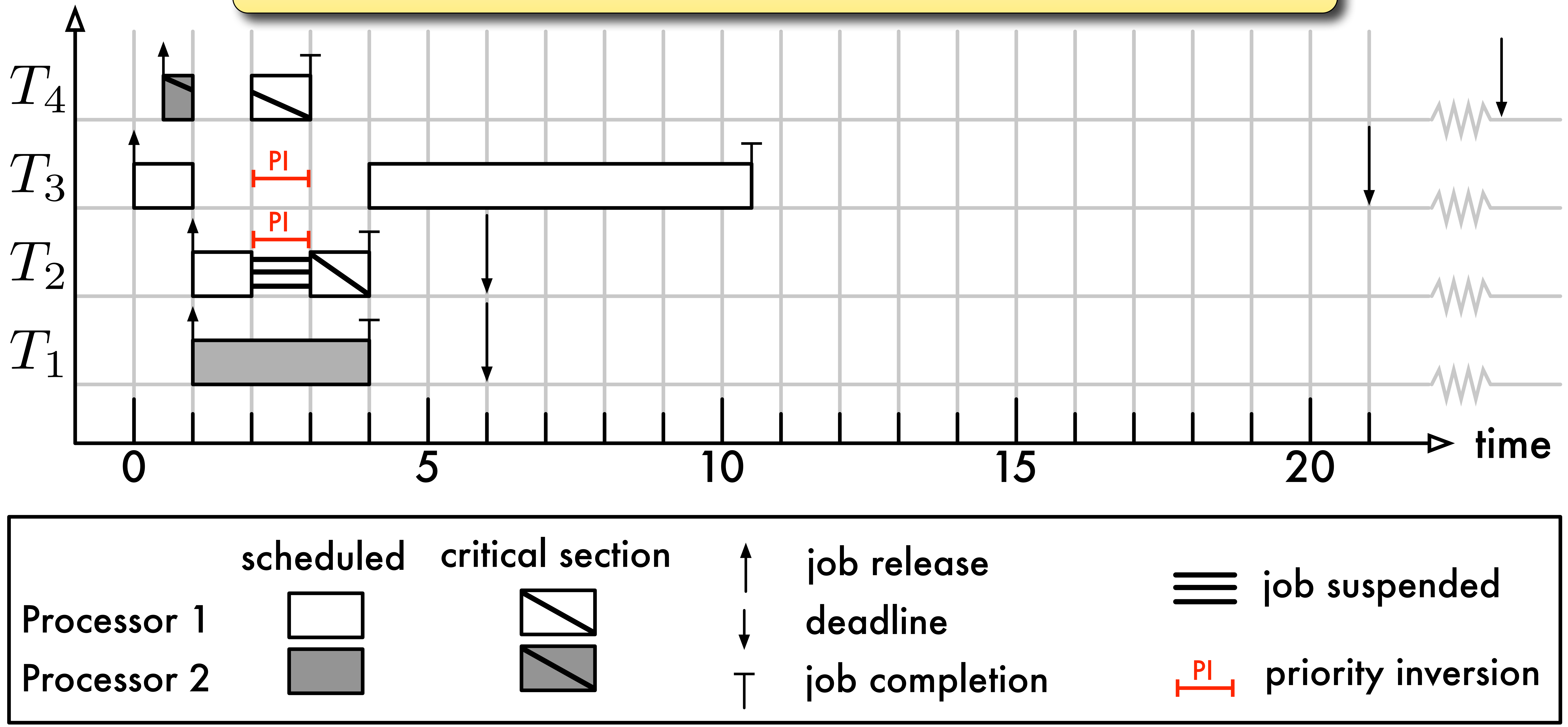
Priority inheritance: T_4 inherits from T_2

→ T_3 is **preempted**, incurs s-aware pi-blocking



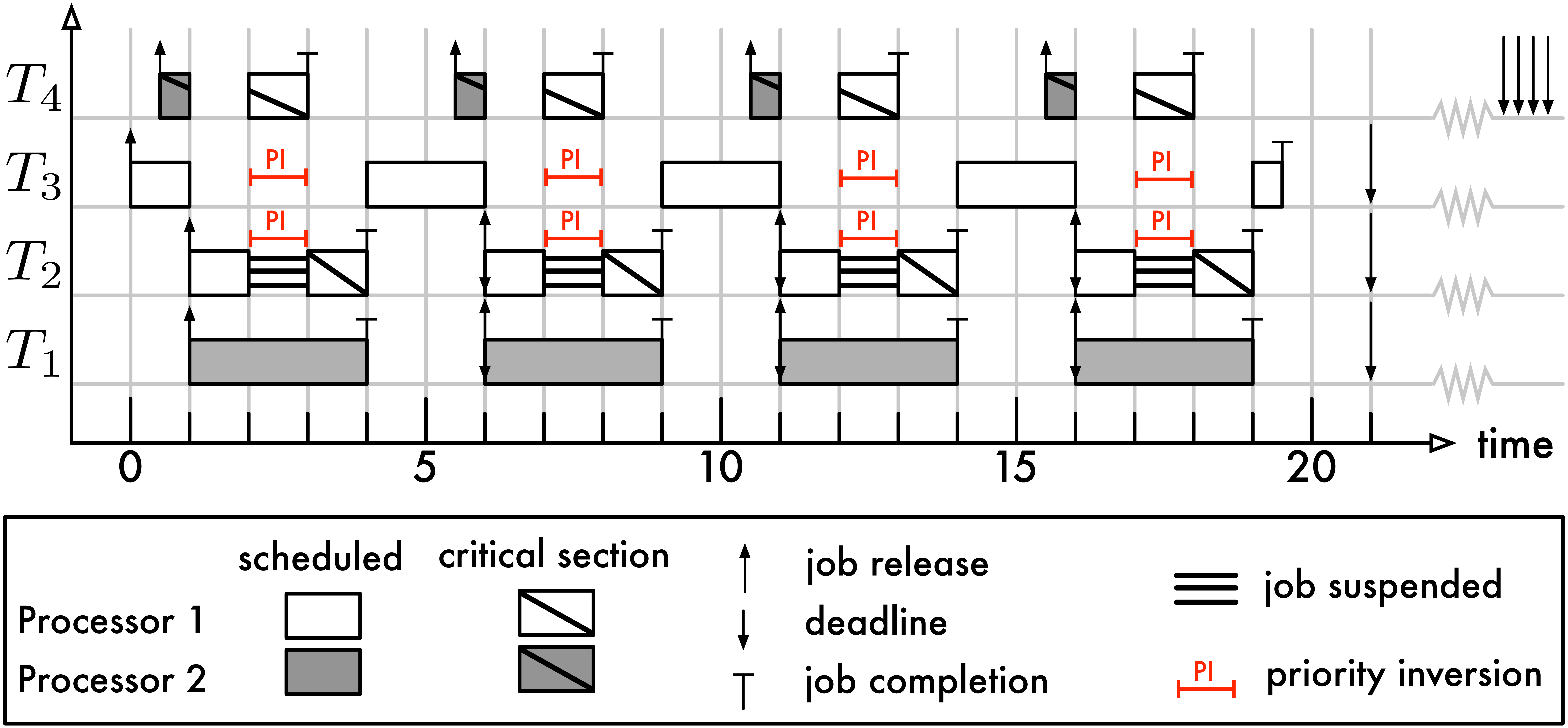
Independent Job Incurs Priority Inversion

***How often** can this scenario **repeat**?*



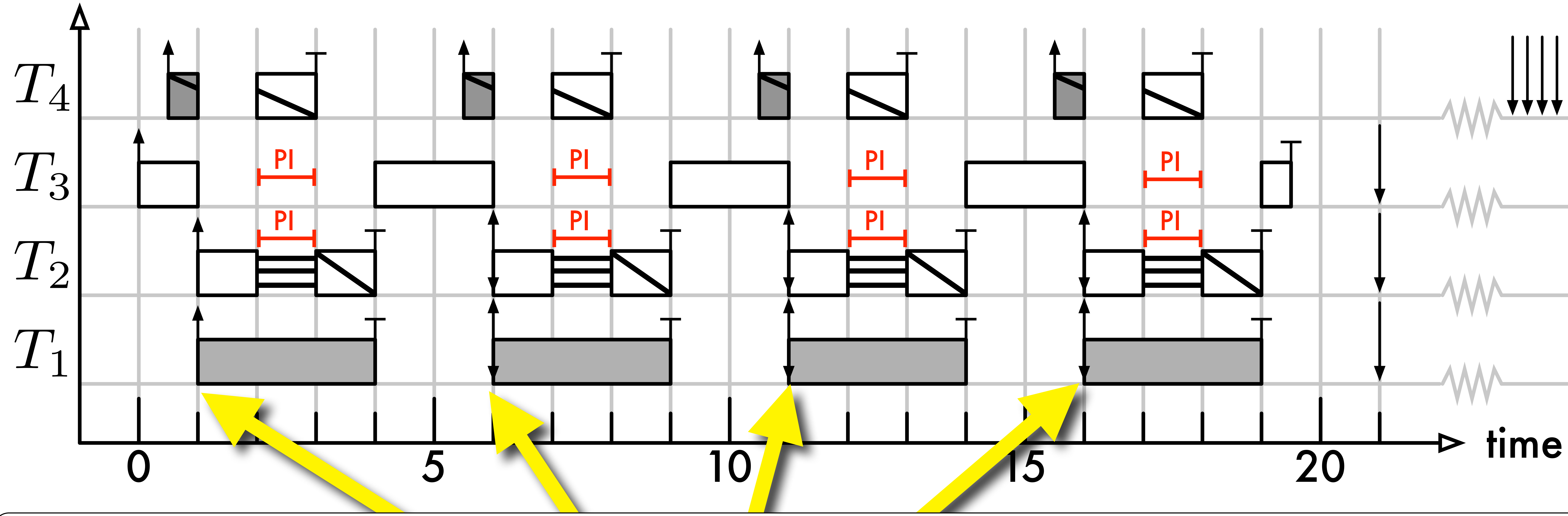
"Victim Task" Accumulates PI-Blocking

Ω(ϕ) pi-blocking** is possible, where **ϕ = {max response time} / {min period}



"Victim Task" Accumulates PI-Blocking

Ω(ϕ) pi-blocking** is possible, where **ϕ = {max response time} / {min period}

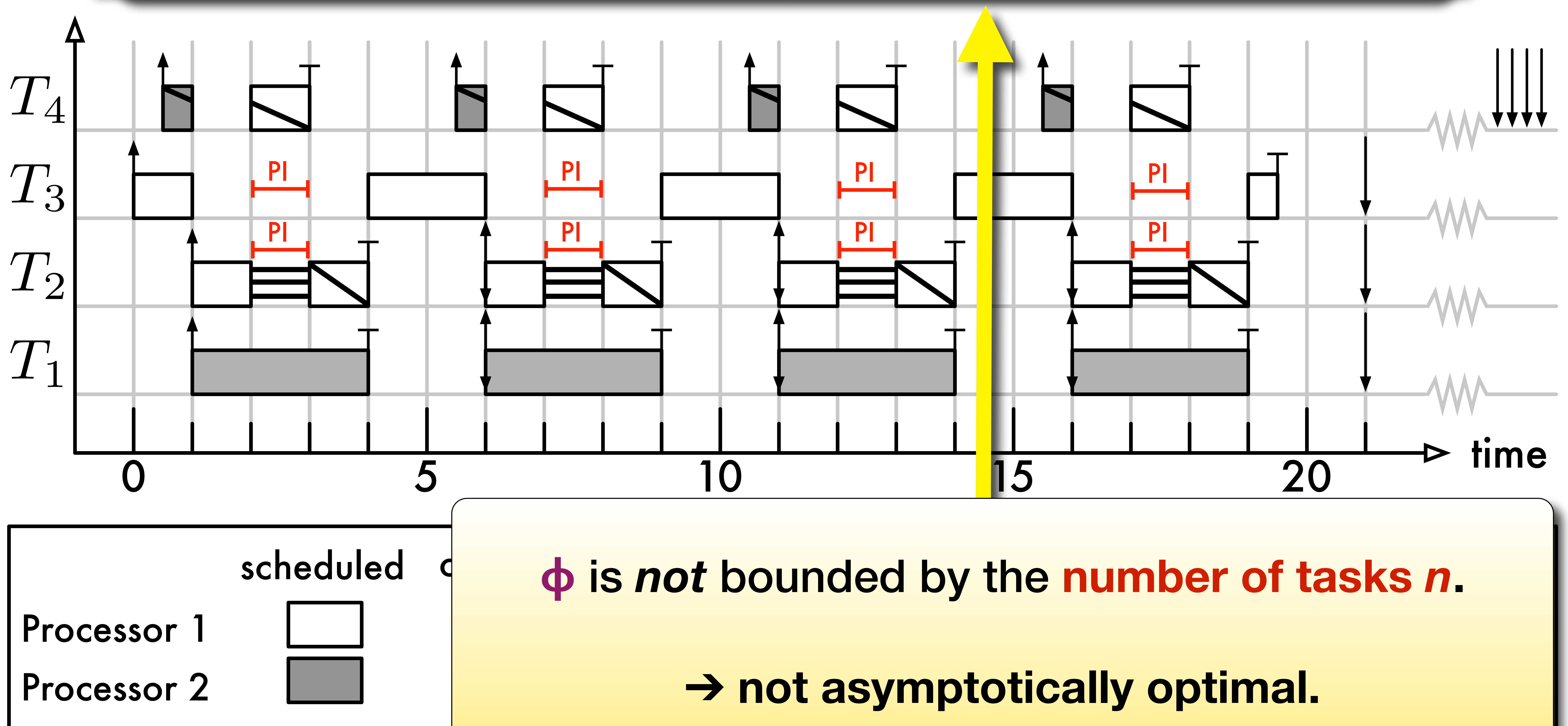


Bounded only by
the number of jobs released by *T1*, *T₂*, and *T₄* while *T₃* is pending.

How many jobs? ➔ **ϕ** = ***{max response time} / {min period}***

Sub-Optimality of Priority Inheritance

Ω(ϕ) pi-blocking** is possible, where **ϕ = {max response time} / {min period}



ϕ is *not* bounded by the number of tasks *n*.

→ not asymptotically optimal.

What about Priority Boosting?

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

job-level fixed priority	Any JLFP Scheduler	Deadlines	Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [— 2011]	
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010]	**FMLP** [Block et al., 2007]	?
Clustered (jobs migrate only among subset of processors)	**OMIP** [— 2013] **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] A Flexible Real-Time Locking Protocol for Multiprocessors, RTCSA 2007.

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

What about Priority Boosting?

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

job-level fixed priority	Any JLFP Scheduler	Deadlines	Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [— 2011]	
Global (jobs migrate freely)	~~**G-OMLP** [— & Anderson, 2010]~~	~~**FMLP** [Block et al., 2007]~~	?
Clustered (jobs migrate only among subset of processors)	~~**OMIP** [— 2013]~~ **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.

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

Sub-Optimality of Unrestricted Priority Boosting

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

*It is **impossible to construct***

*an **asymptotically optimal** locking protocol*

(w.r.t. s-aware analysis)

under global JLFP scheduling

*based on **unrestricted priority boosting**.*

[—, 2011]

(And hence also under clustered JLFP scheduling.)

[—, 2011]: *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, University of North Carolina at Chapel Hill, 2011.

Sub-Optimality of Unrestricted Priority Boosting

Unrestricted Priority Boosting Example Schedule

[same task set & arrival sequence as before]

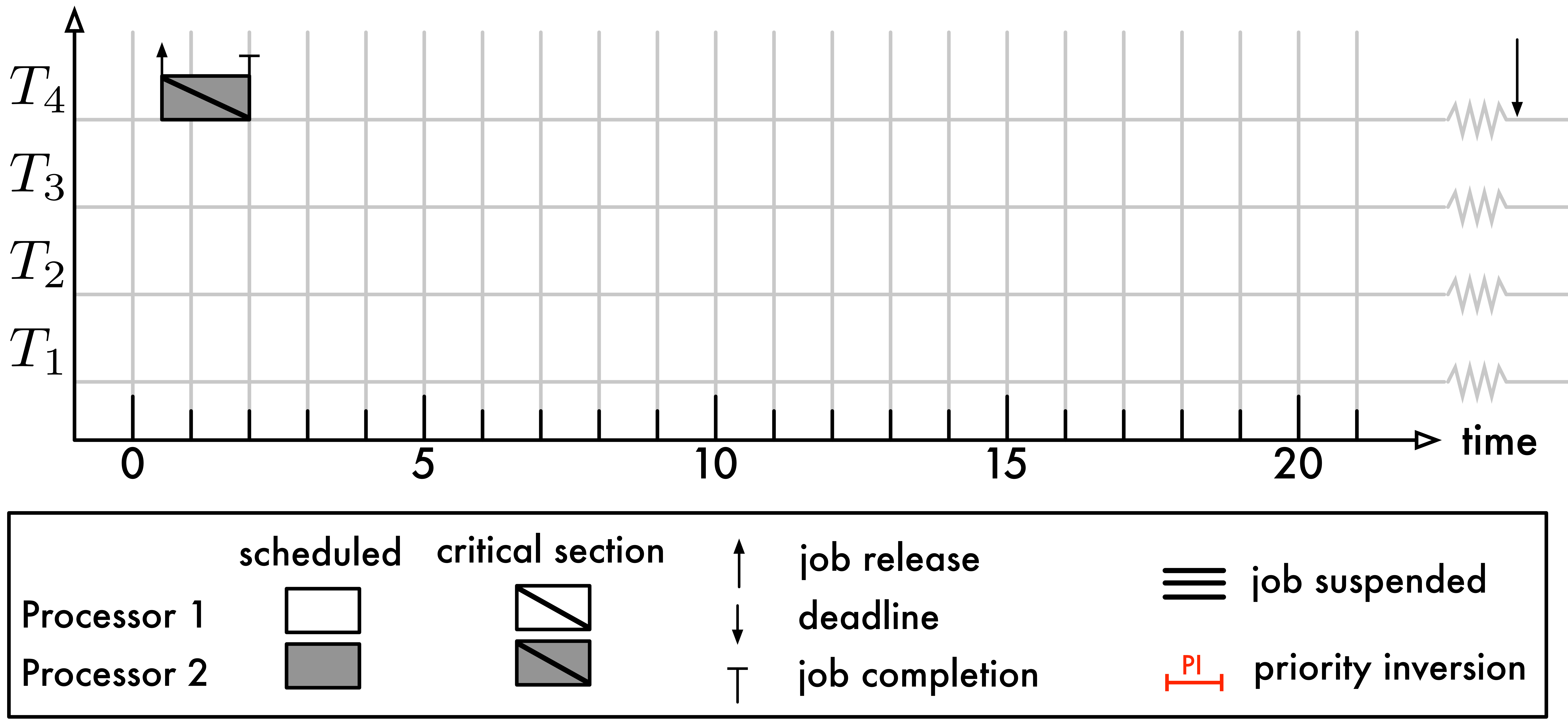
4 tasks on 2 processors

global fixed-priority scheduling

Task priorities (high to low): $T_1 > T_2 > T_3 > T_4$

	scheduled	critical section		
Processor 1			job release	job suspended
Processor 2			deadline	
			job completion	PI priority inversion

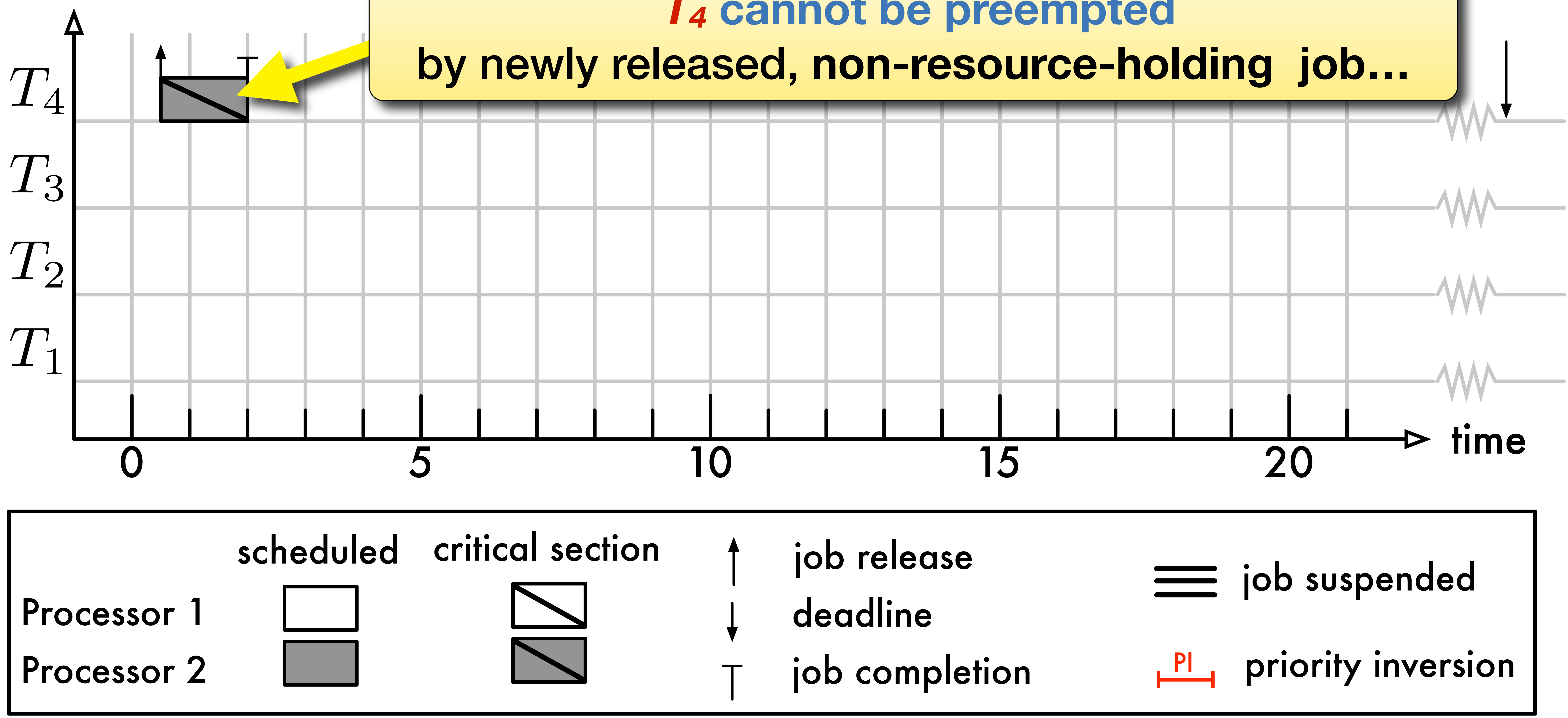
Lock-Holding Jobs Cannot Be Preempted



Lock-Holding Jobs Cannot Be Preempted

Priority Boosting

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

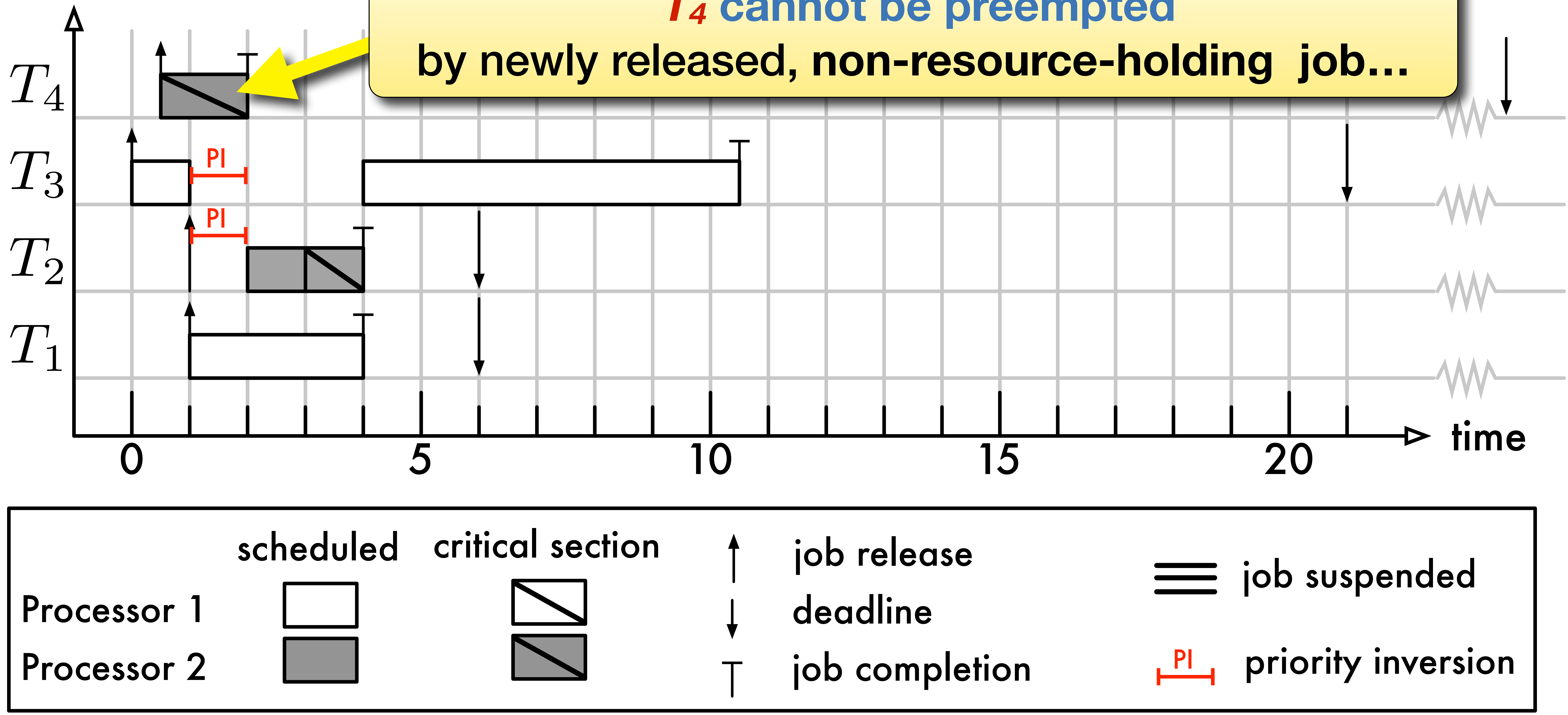


PI-Blocking Shifted, not Avoided

Priority Boosting

***T4* cannot be preempted**

by newly released, non-resource-holding job…

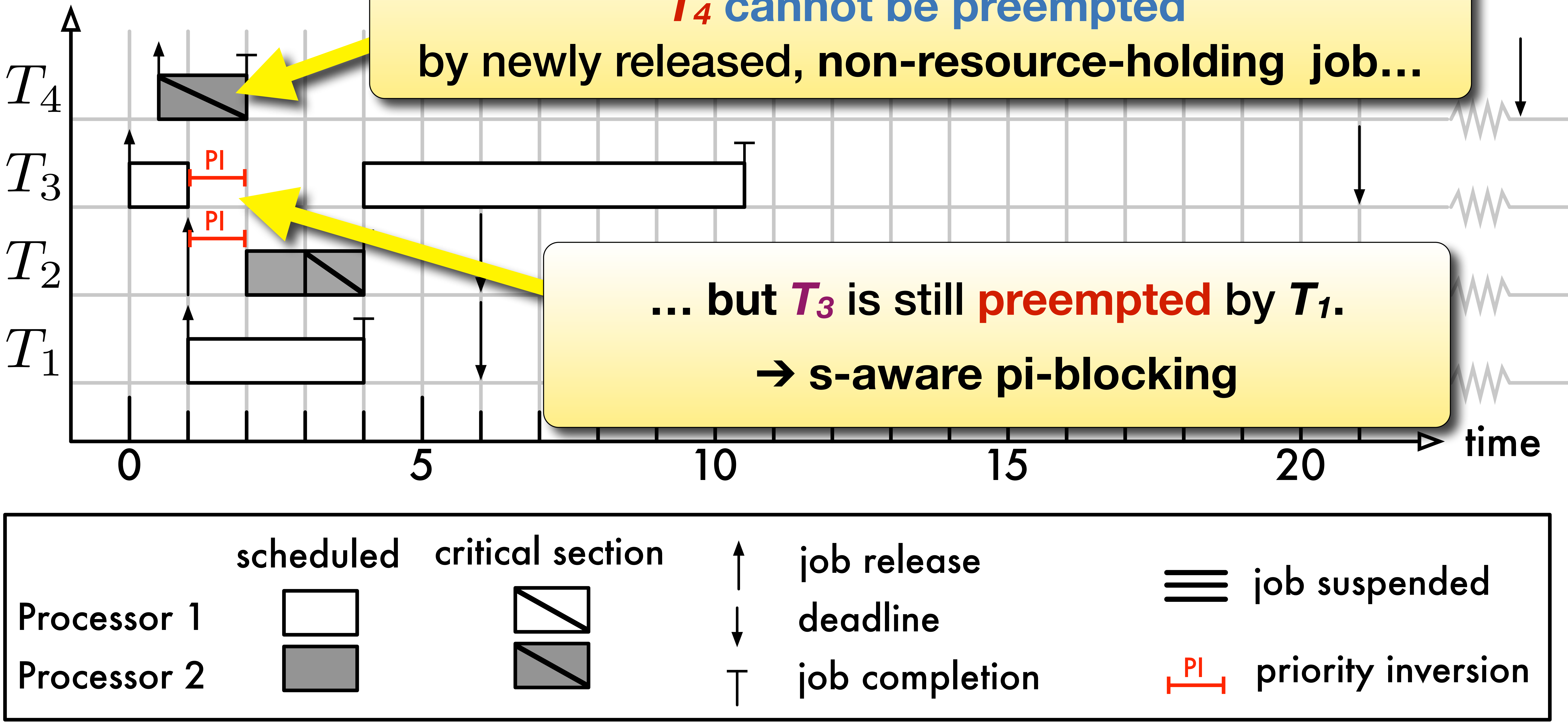


PI-Blocking Shifted, not Avoided

Priority Boosting

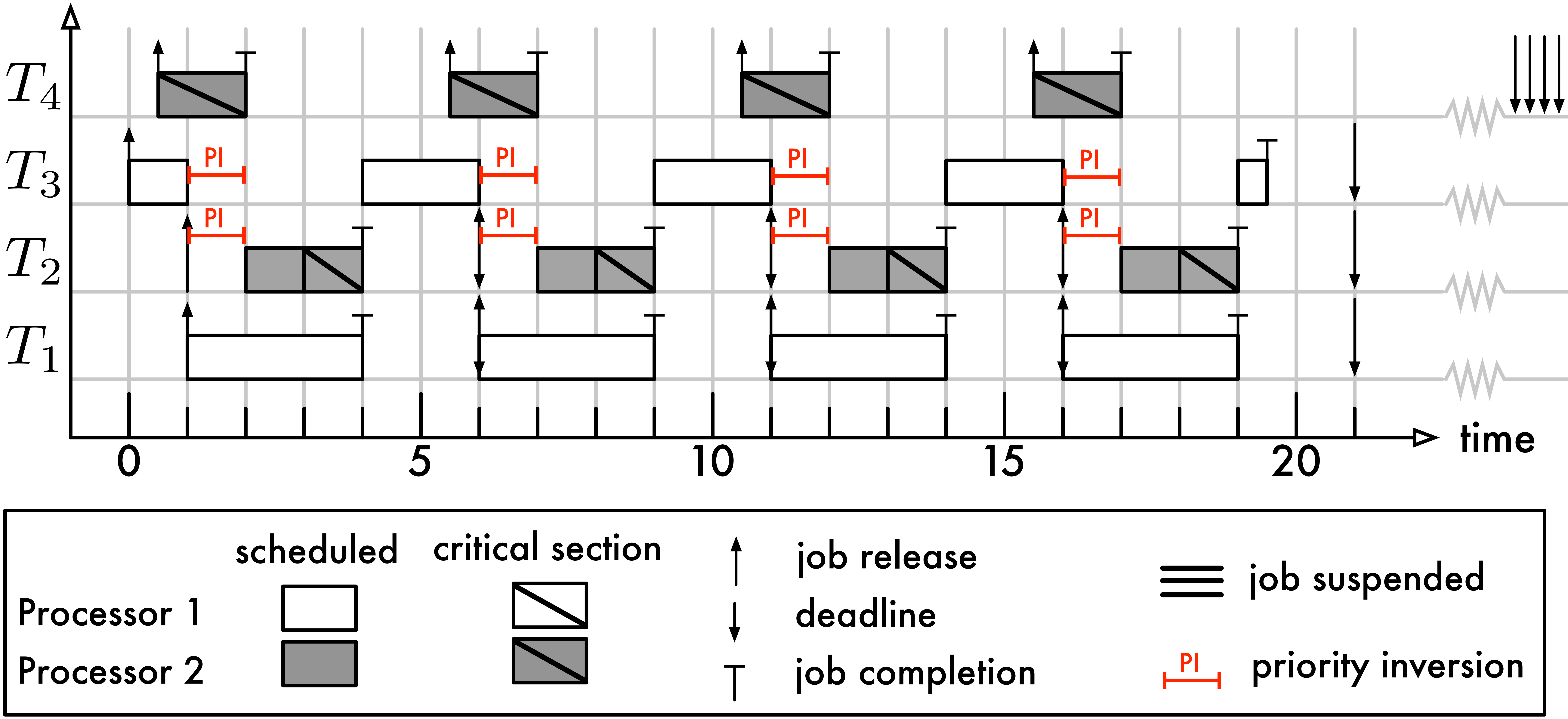
***T_4* cannot be preempted**
by newly released, non-resource-holding job…

… but *T_3* is still preempted by *T_1*.
➔ s-aware pi-blocking



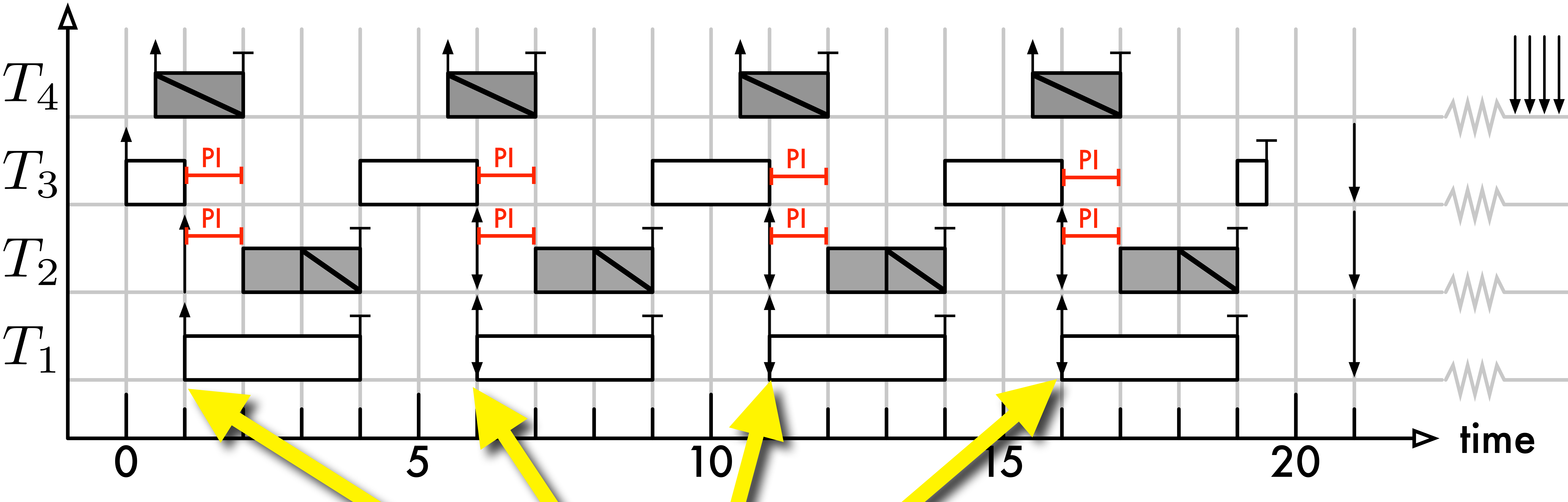
"Victim Task" Accumulates PI-Blocking

Ω(ϕ) pi-blocking** is possible, where **ϕ = {max response time} / {min period}



Sub-Optimality of Unrestricted Priority Boosting

Ω(ϕ) pi-blocking *is possible, where* **ϕ = *{max response time} / {min period}***

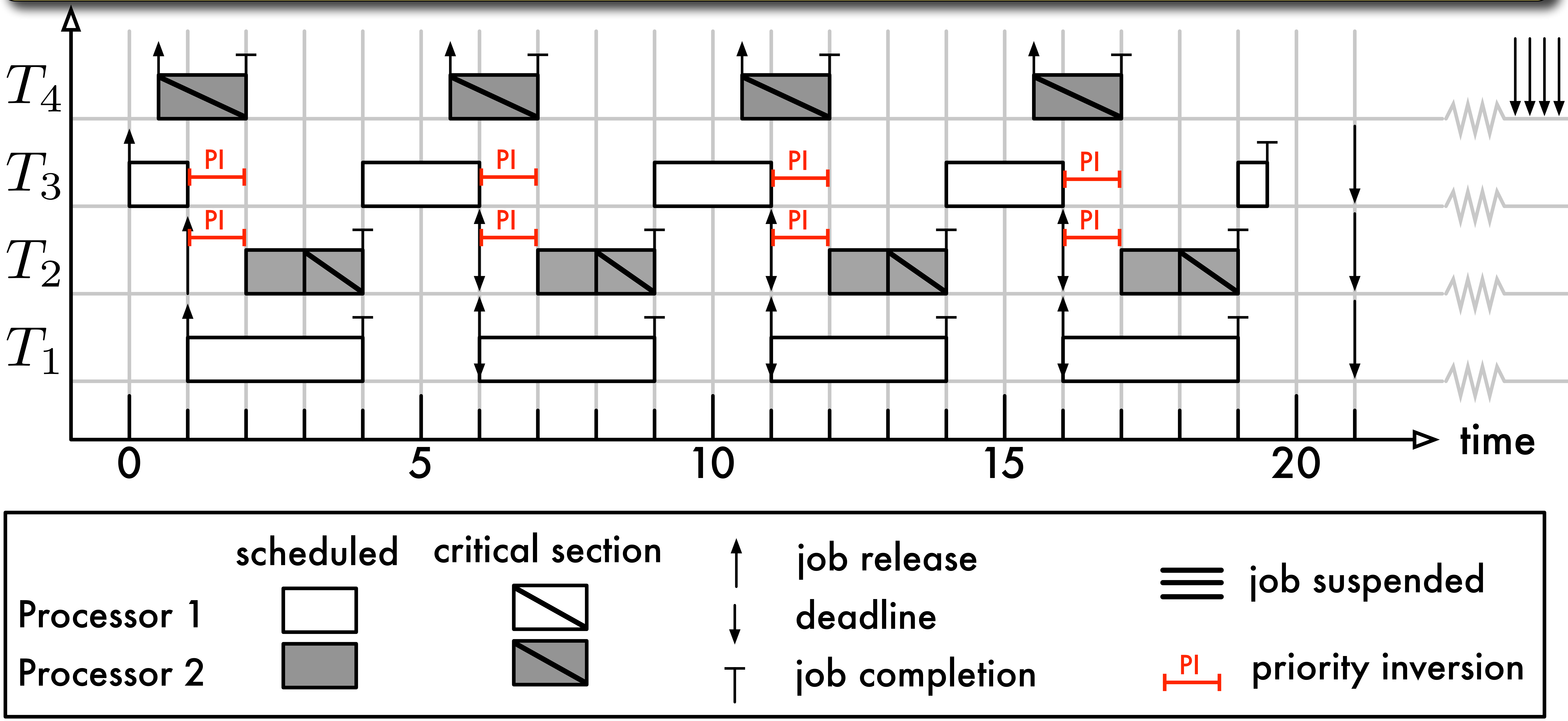


Also repeats **ϕ** = ***{max response time} / {min period}*** times…

➔ **not asymptotically optimal.**

Remark: examples use single resource shared by only two tasks

→ queue order irrelevant (FIFO-ordered, priority-ordered, etc.)
→ cannot simplify problem with coarser-grained locking



We need something new…

~~(1) The classic ch… (variants of) priority inheritance.~~

job-level fixed priority	Any JLFP Scheduler	Deadlines	Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010]	**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP+** (practical protocol) [— 2011]	
Global (jobs migrate freely)	~~**G-OMLP** [— & Anderson, 2010]~~	~~**FMLP** [Block et al., 2007]~~	?
Clustered (jobs migrate only among subset of processors)	~~**OMIP** [— 2013]~~ **C-OMLP** [— & Anderson, 2011]	?	?

[Block et al., 2007] A Flexible Real-Time Locking Protocol for Multiprocessors, RTCSA 2007.

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

We need something new…

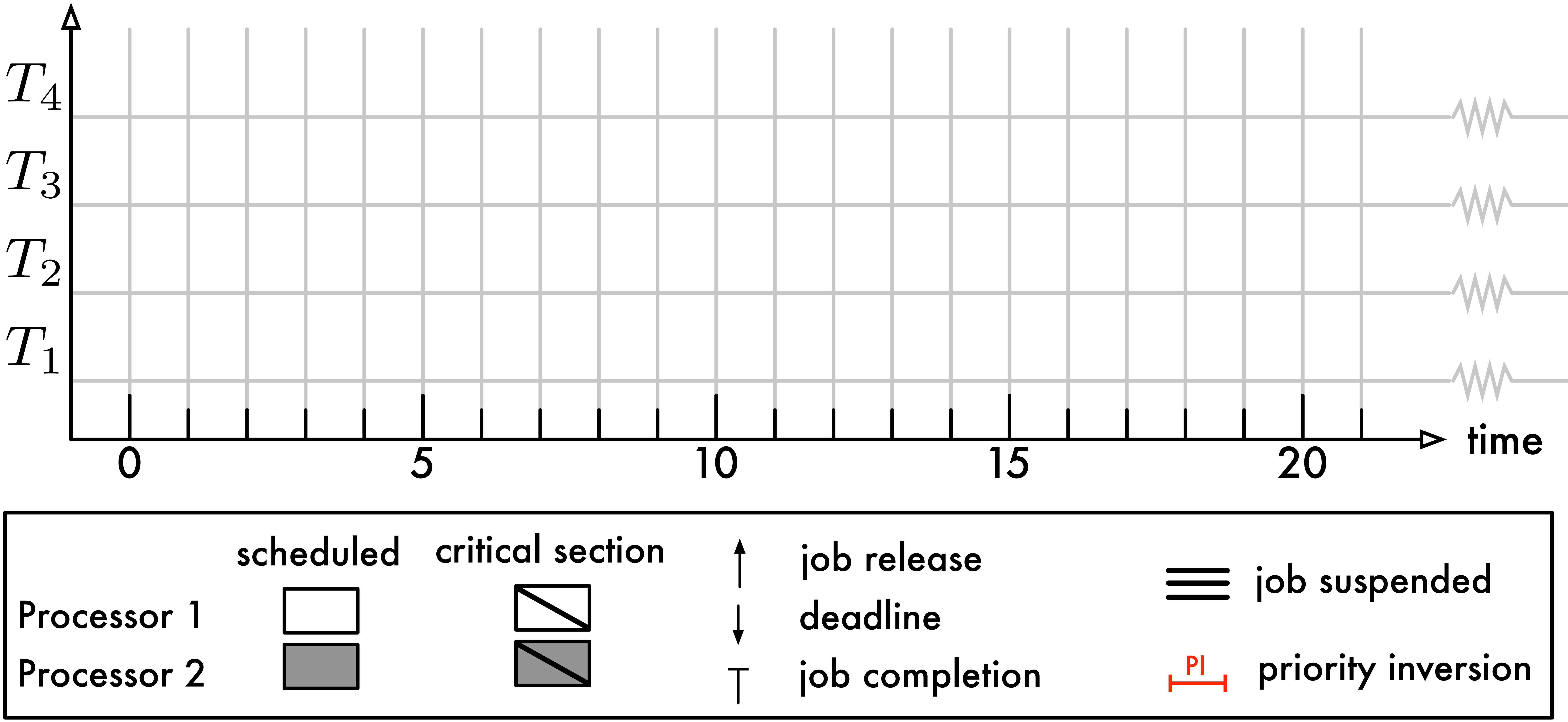
~~(1) The classic choice: (variants of) priority inheritance.~~

job-level fixed priority	Any JLFP Scheduler	Deadlines	Any JLFP Scheduler
Partitioned (no migrations)	~~**P-OMLP** [— & Anderson, 2010]~~	~~**SPFP** (asymptotical tightness) [— & Anderson, 2010] **P-FMLP⁺** (practical protocol) [— 2011]~~	
Global (jobs migrate freely)	~~**G-OMLP** [— & Anderson, 2010]~~	~~**FMLP** [Block et al., 2007]~~	?
Clustered (jobs migrate only among subset of processors)	~~**OMIP** [— 2013]~~ ~~**C-OMLP** [— & Anderson, 2011]~~	?	?

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.

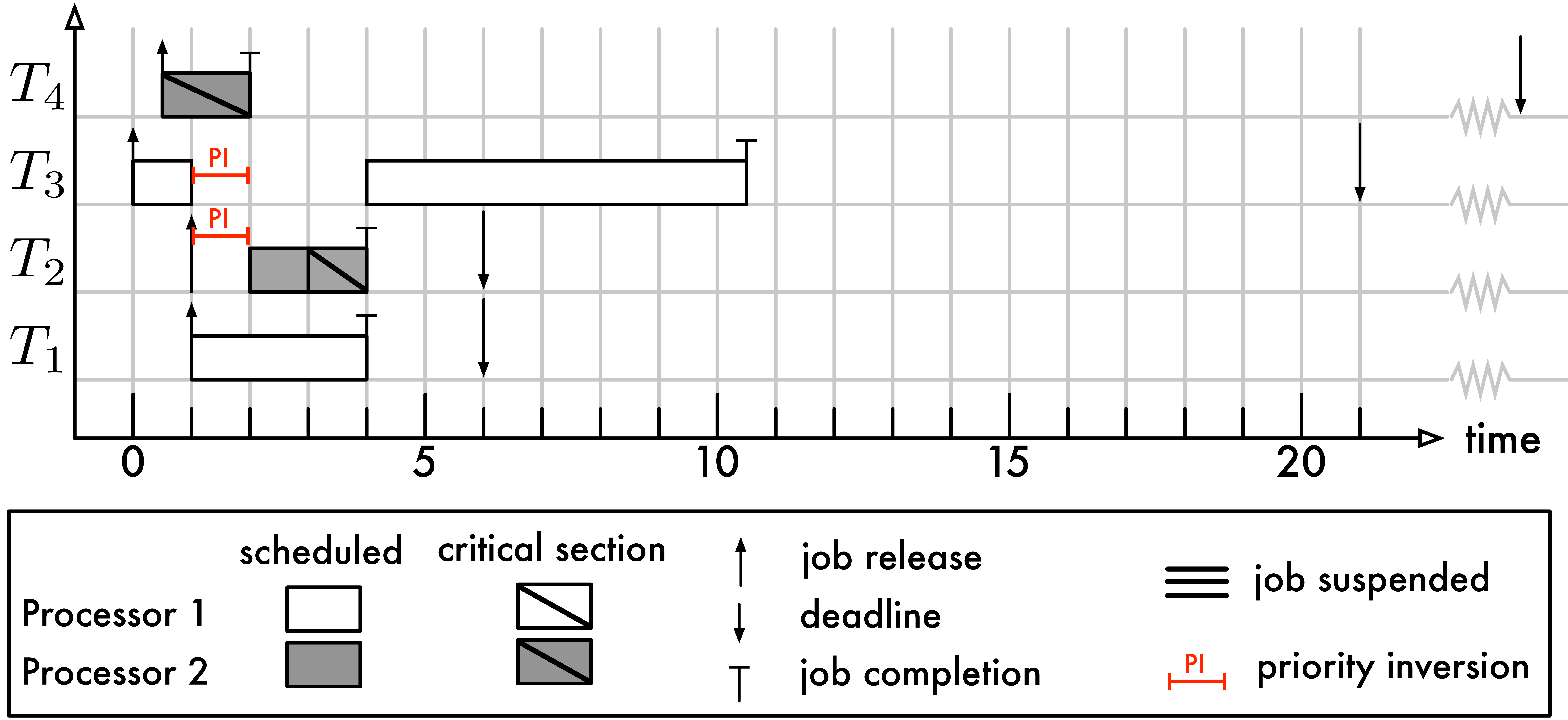
~~(2) The partitioned & clustered choice: (variants of) priority boosting.~~

Observation: O(*n*) PI-Blocking Possible



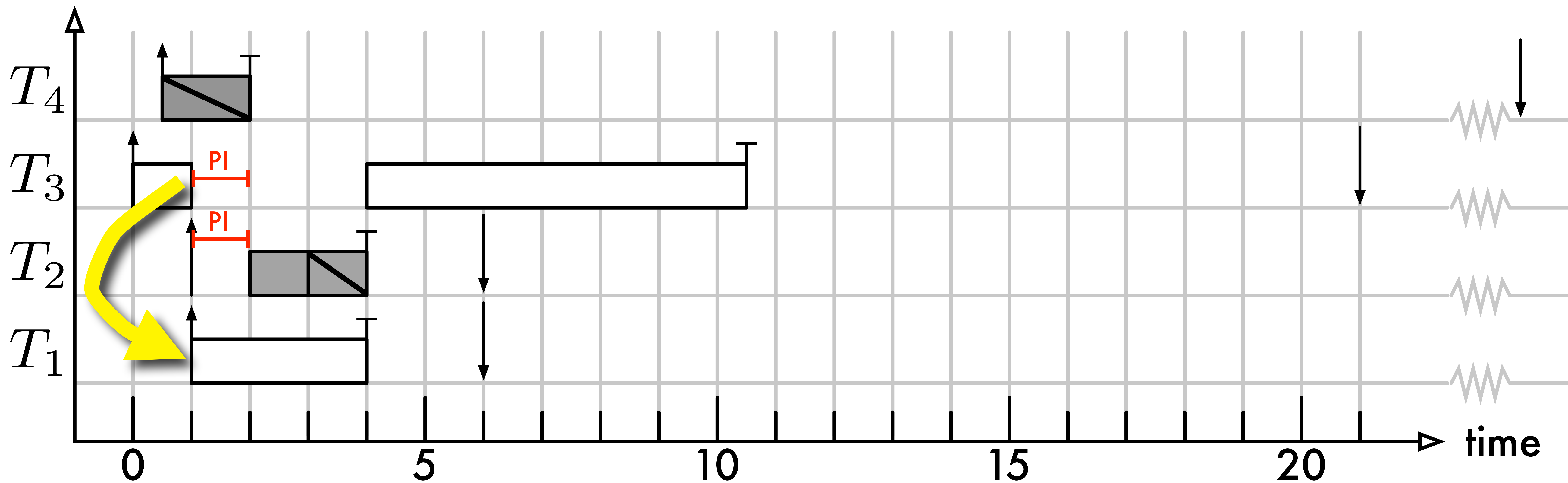
Observation: O(*n*) PI-Blocking Possible

This schedule: unrestricted priority boosting.



Observation: O(*n*) PI-Blocking Possible

This schedule: unrestricted priority boosting.

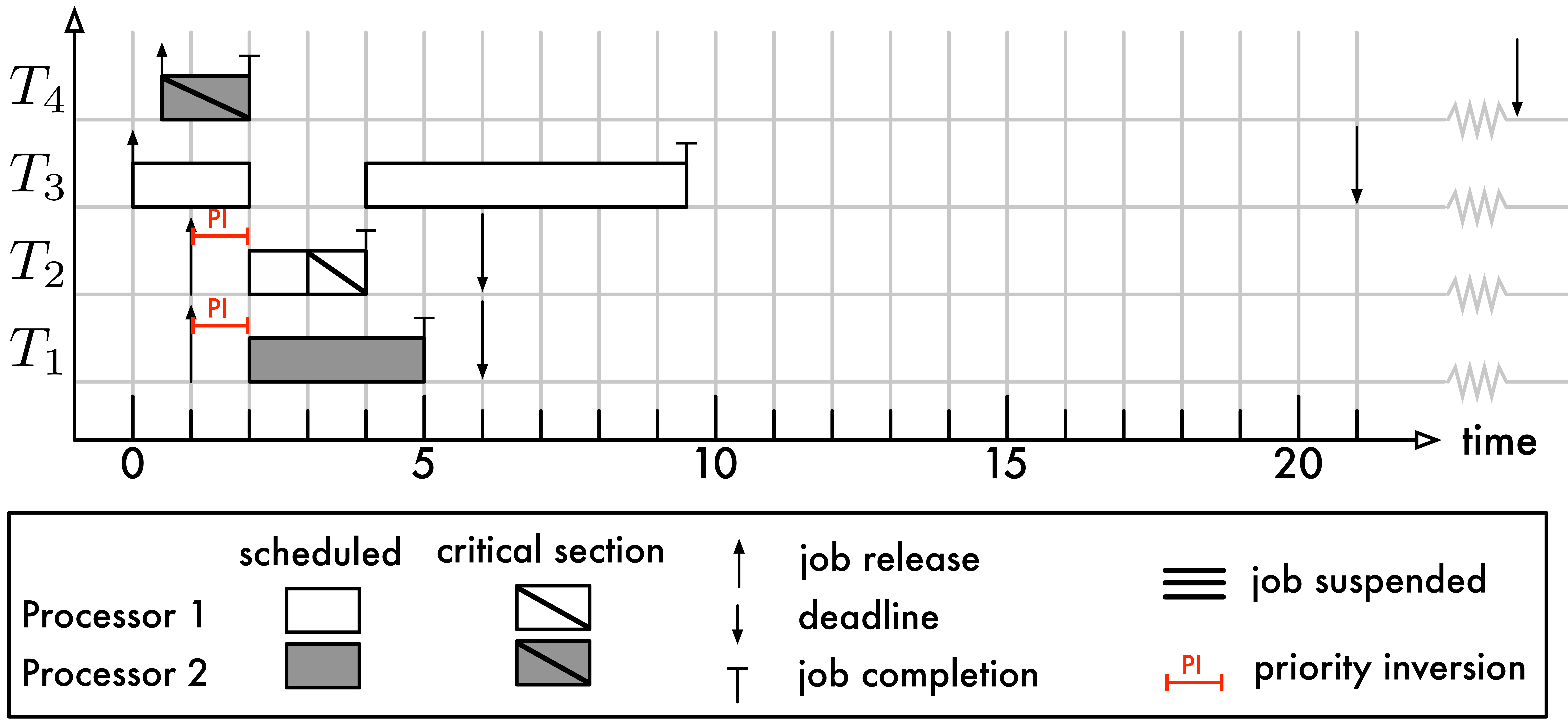


PI-blocking cannot be avoided, but it can be shifted to new jobs.

→ prevent accumulation of pi-blocking in individual jobs.

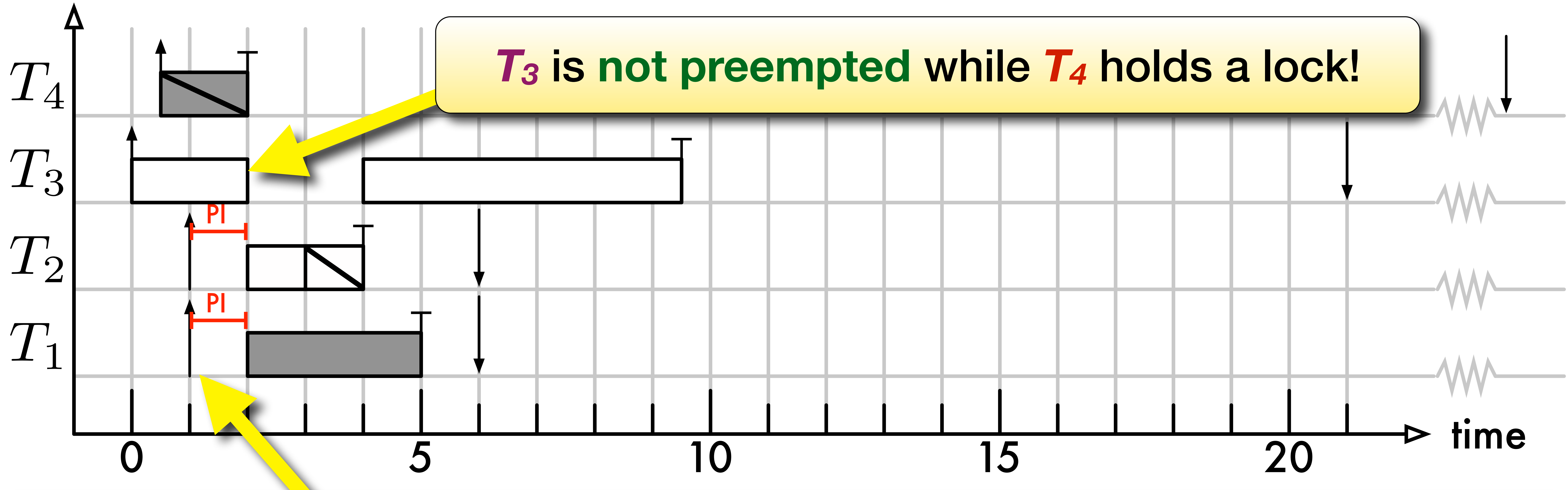
Idea: Protect Existing Independent Jobs

***Alternative* possible schedule.**



Idea: Protect Existing Independent Jobs

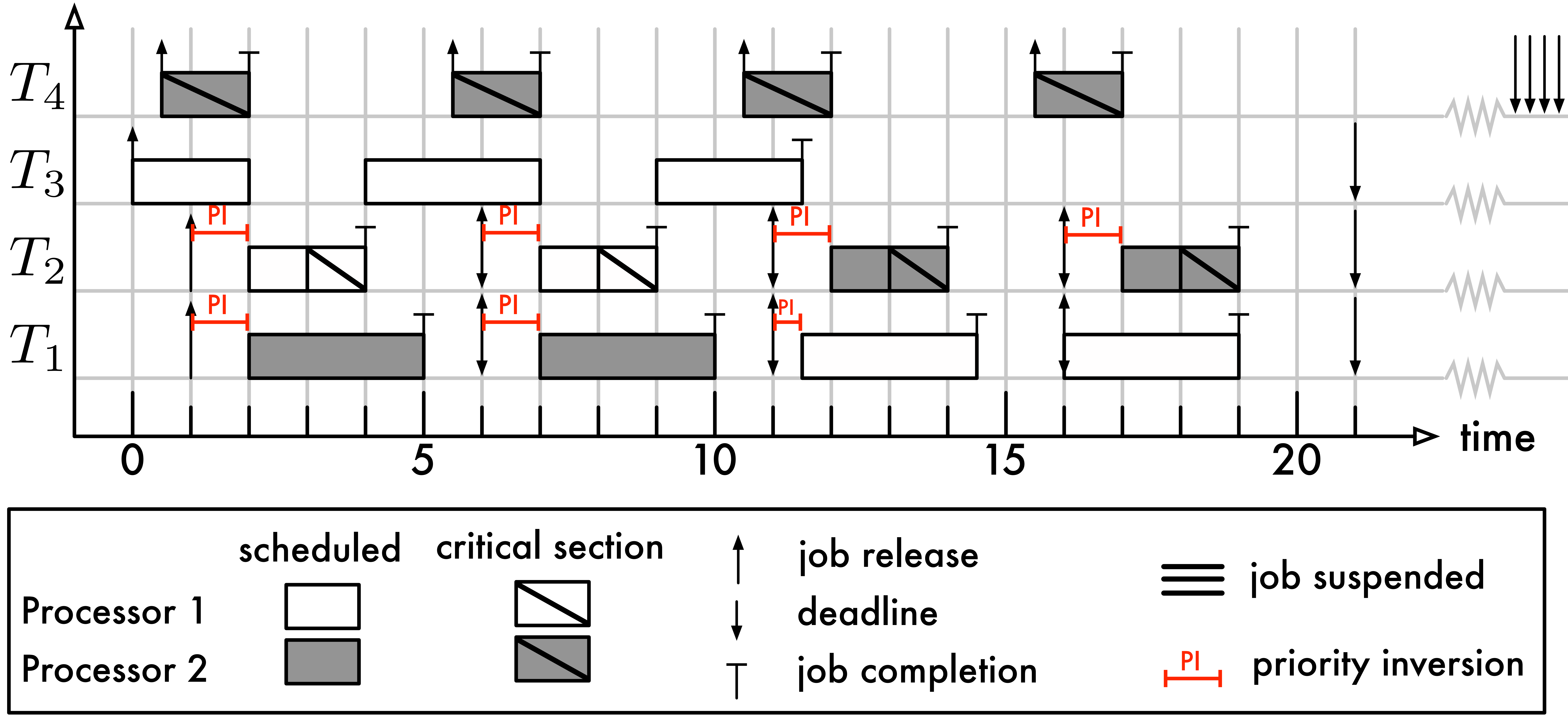
Alternative possible schedule.



Both T_1 and T_2 incur pi-blocking instead.

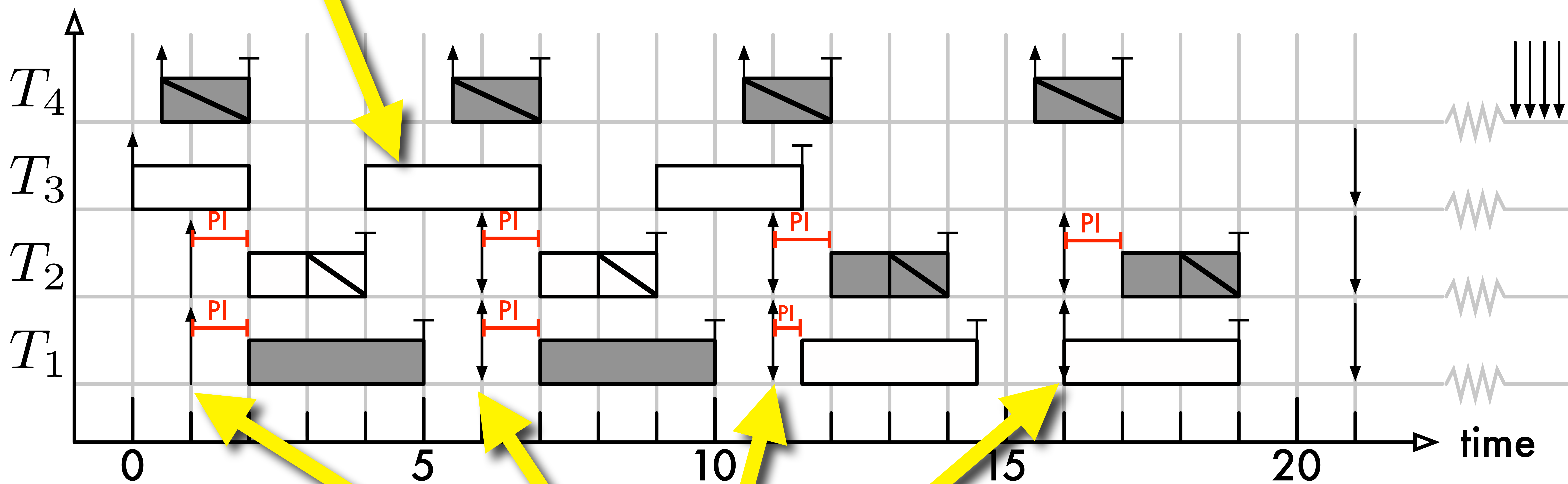
O(*n*) PI-Blocking Per Job

***Alternative* possible schedule.**



***T_3* "protected": incurs no pi-blocking in this example.**

Alternative possible schedule.

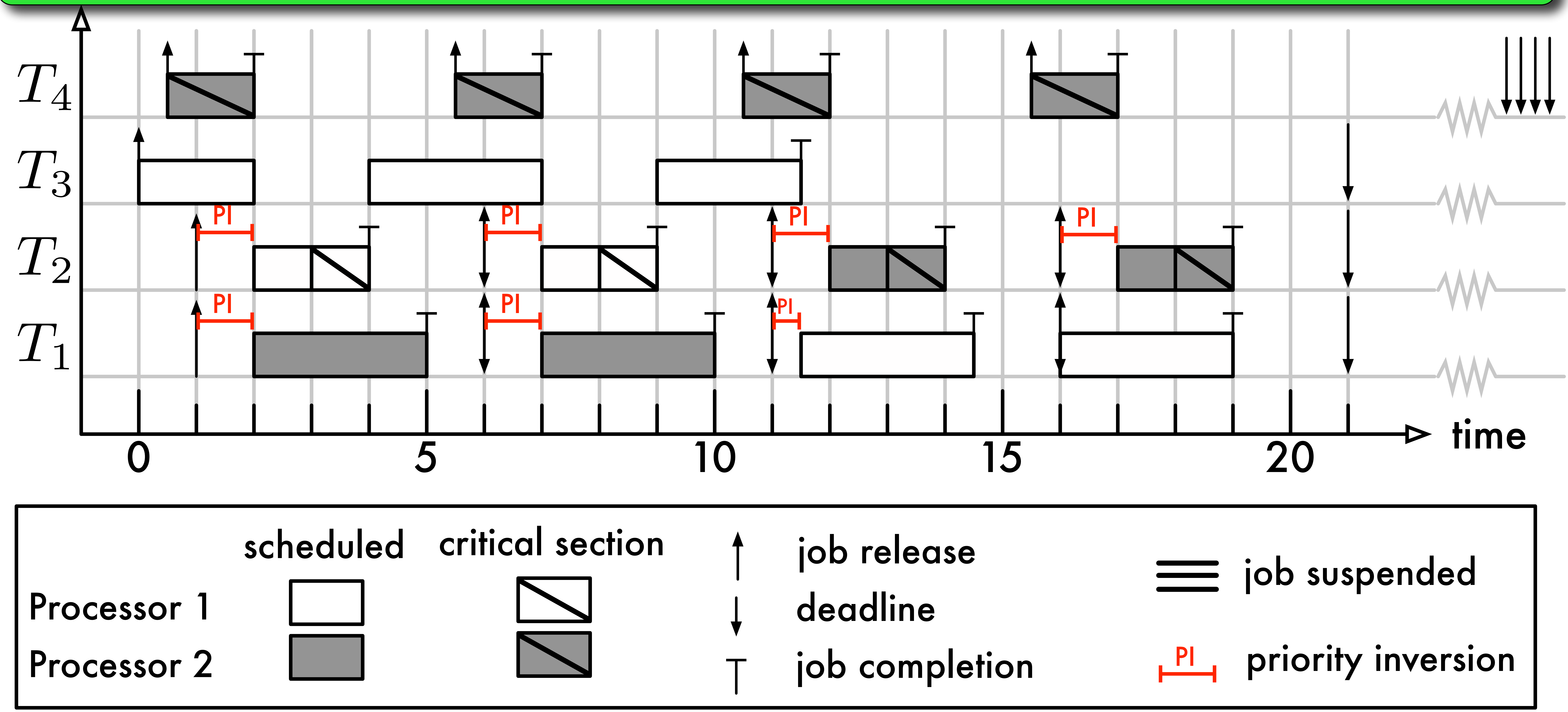


Each job incurs only a **limited amount** of pi-blocking.

→ **asymptotically optimal.**

This is actually a **Generalized FMLP⁺** schedule.

*Key question: how to specify that "**T_3 must be protected**"?*



Closing the S-Aware Asymptotic Optimality Gap

Key Problem: Preemptions due to Later-Started Critical Sections

On Uniprocessors

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

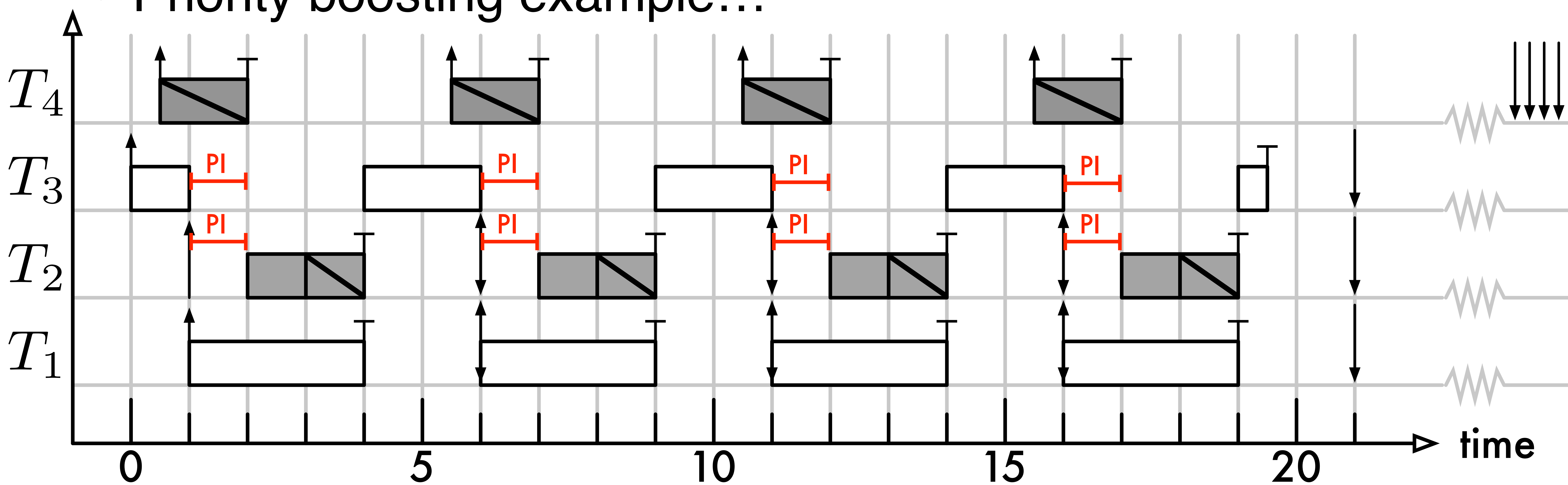
Key Problem: Preemptions due to Later-Started Critical Sections

On Uniprocessors

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

On Multiprocessors

➡ Priority boosting example…



Priority Boosting / Priority Inheritance Examples

T_3 blocked due to ϕ requests issued after T_3 started executing.

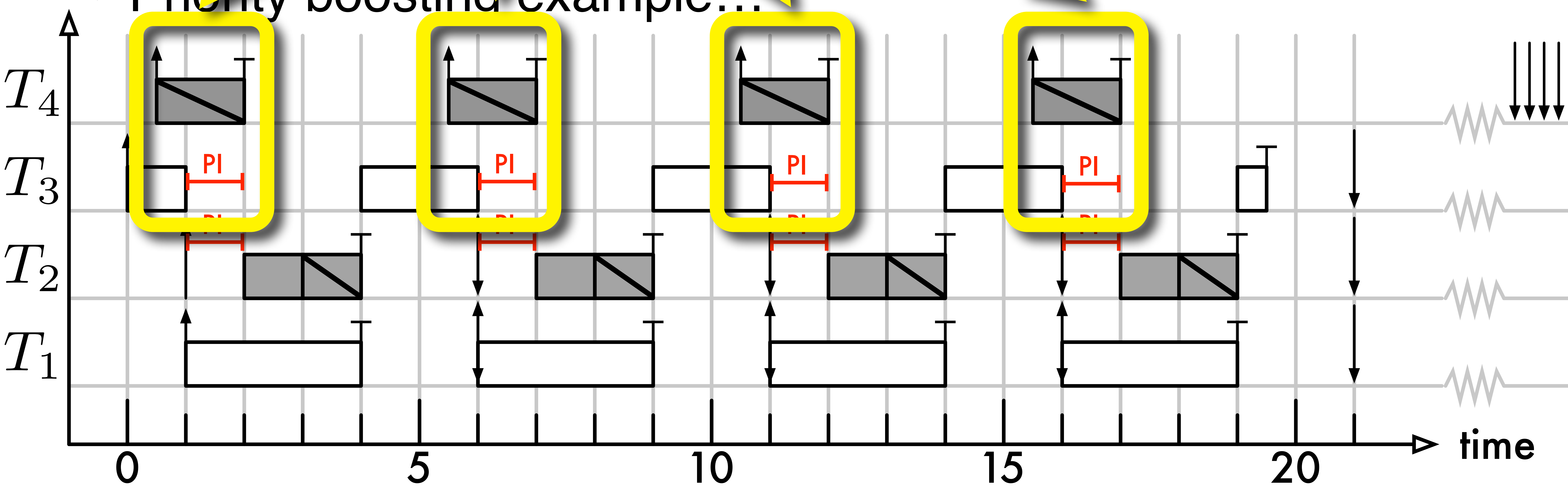
(➔ root cause: parallel scheduling of lower-priority jobs)

On Uniprocessors

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

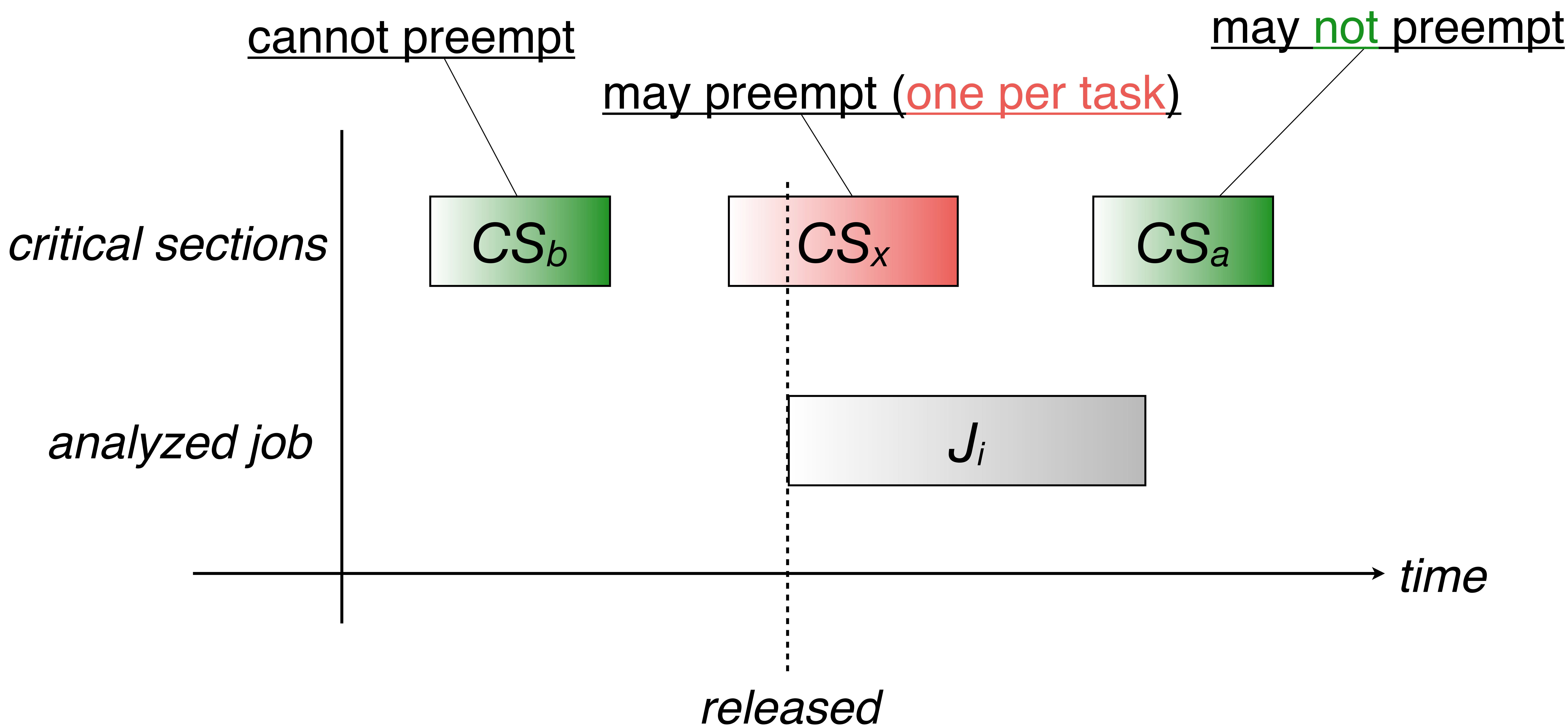
On Multiprocessors

➡ Priority boosting example…



What if this is disallowed...?

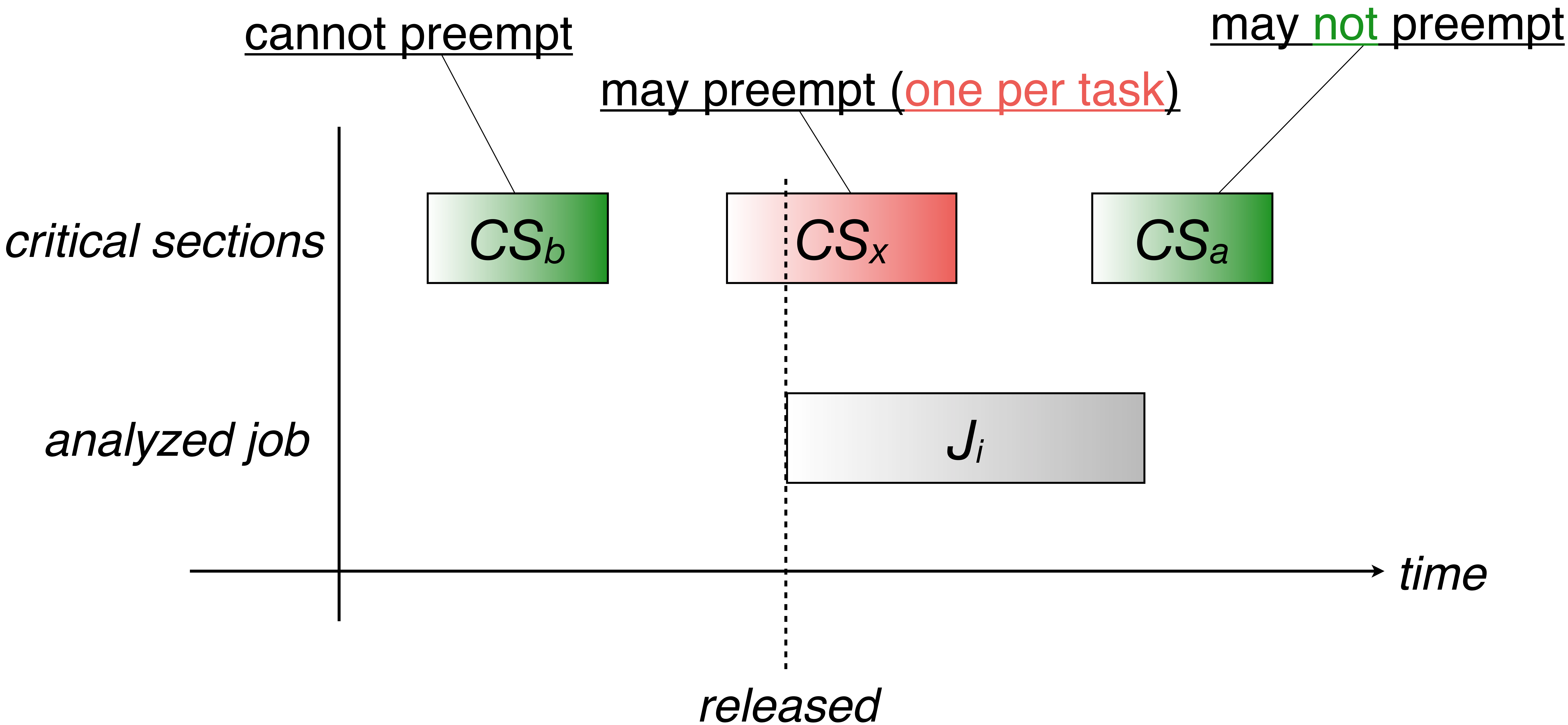
strawman rule: jobs cannot be preempted due to later-issued requests



➡ $O(n)$ preemptions

This is the desired effect, but the simple rule fails in corner cases.

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



➡ $O(n)$ preemptions

Independent & Request Segments

a job at runtime:

Note: exact segments known only at runtime

➡ potentially complex, non-linear **control flow** determines which resources are required and in which order

➡ approach ***not*** limited to linear, branch-free tasks

Key Concept: Segment Start Time

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

a job at runtime:

Note: exact segments known only at runtime

- ➡ potentially complex, non-linear **control flow** determines which resources are required and in which order
- ➡ approach ***not*** limited to linear, branch-free tasks

A Lock Holder's Co-Boosting Set

Key idea underlying the Generalized FMLP+

*If a job is **priority-boosted**,*
*then **certain other jobs** must also be **co-boosted**.*

A Lock Holder's Co-Boosting Set

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

$$\left\{ J_y \;\middle|\; \begin{array}{l} J_y \text{ executes an } \textbf{independent segment} \text{ at } \textbf{time } t \text{ and} \\ J_y \text{ has } \textbf{higher priority} \text{ than } J_b \text{ and} \\ J_y\text{'s current segment } \textbf{started before } J_b\text{'s segment.} \end{array} \right\}$$

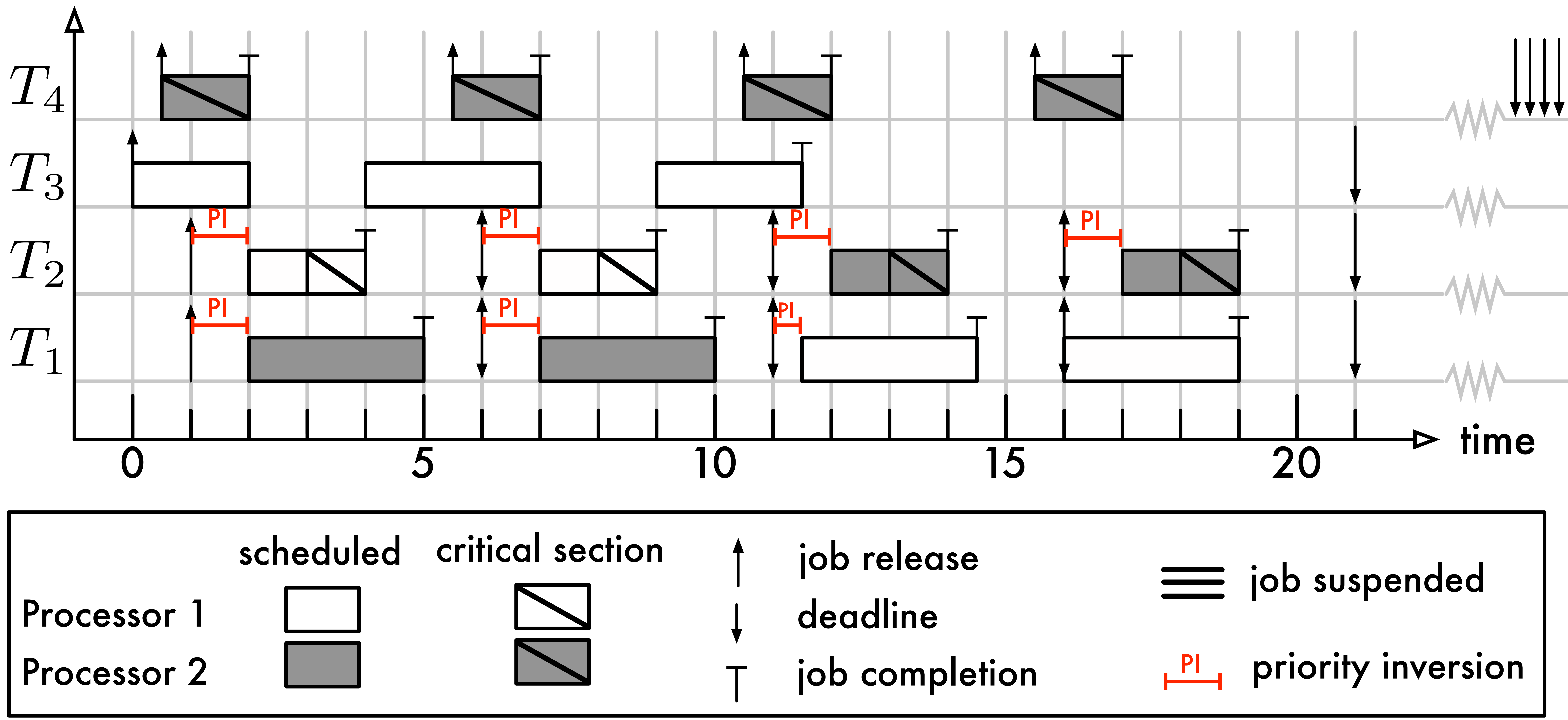
(Note: in this talk, I'll use "task" and "job" interchangeably.)

Intuition

➡ The set of jobs **at risk** of **accumulating pi-blocking** due to J_b.

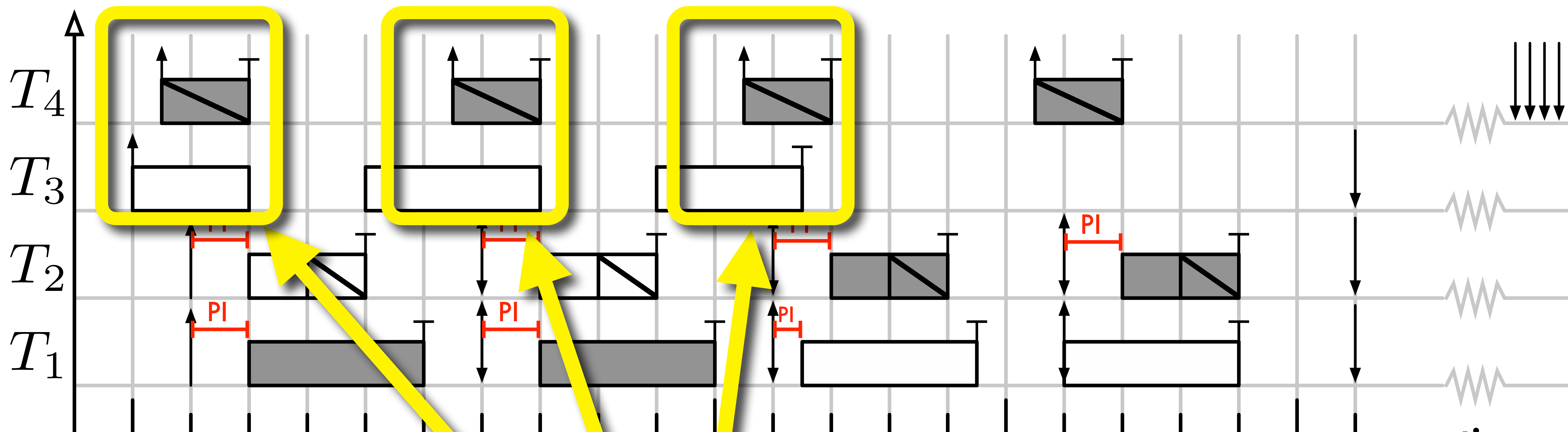
Example: { T_3 } is T_4's Co-Boosting Set

Generalized FMLP⁺ schedule.



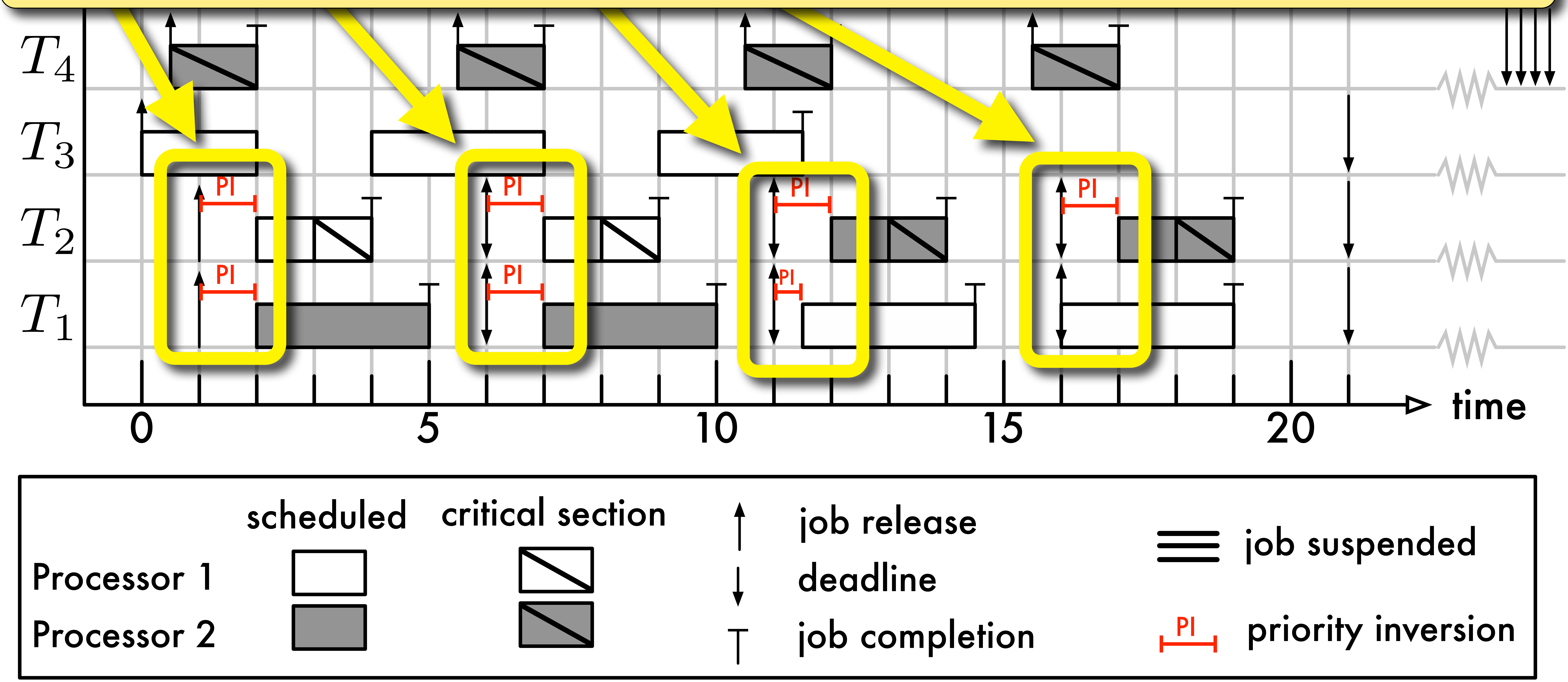
Example: { T_3 } is T_4's Co-Boosting Set

Generalized FMLP⁺ schedule.



T_3 | T_3 *executes an* ***independent segment*** *at* ***time t*** *and* T_3 *has* ***higher priority*** *than* T_4 *and* T_3*'s current segment* ***started before*** T_4*'s segment.*

***T_1* and *T_2* execute *independent segments* at *time t* and**
***T_1* and *T_2* have *higher priority* than *T_4* but**
***T_1* and *T_2*'s current segments *did NOT start before* *T_4*'s segment.**



Restricted Segment Boosting

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

Restricted Segment Boosting

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

A Single Boosted Job J_b

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

(any ties broken arbitrarily but consistently)

Restricted Segment Boosting

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

A Single Boosted Job J_b

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

Up to $c - 1$ jobs from J_b's Co-Boosting Set

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

(any ties broken arbitrarily but consistently)

Restricted Segment Boosting

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

A Single Boosted Job J_b

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

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

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

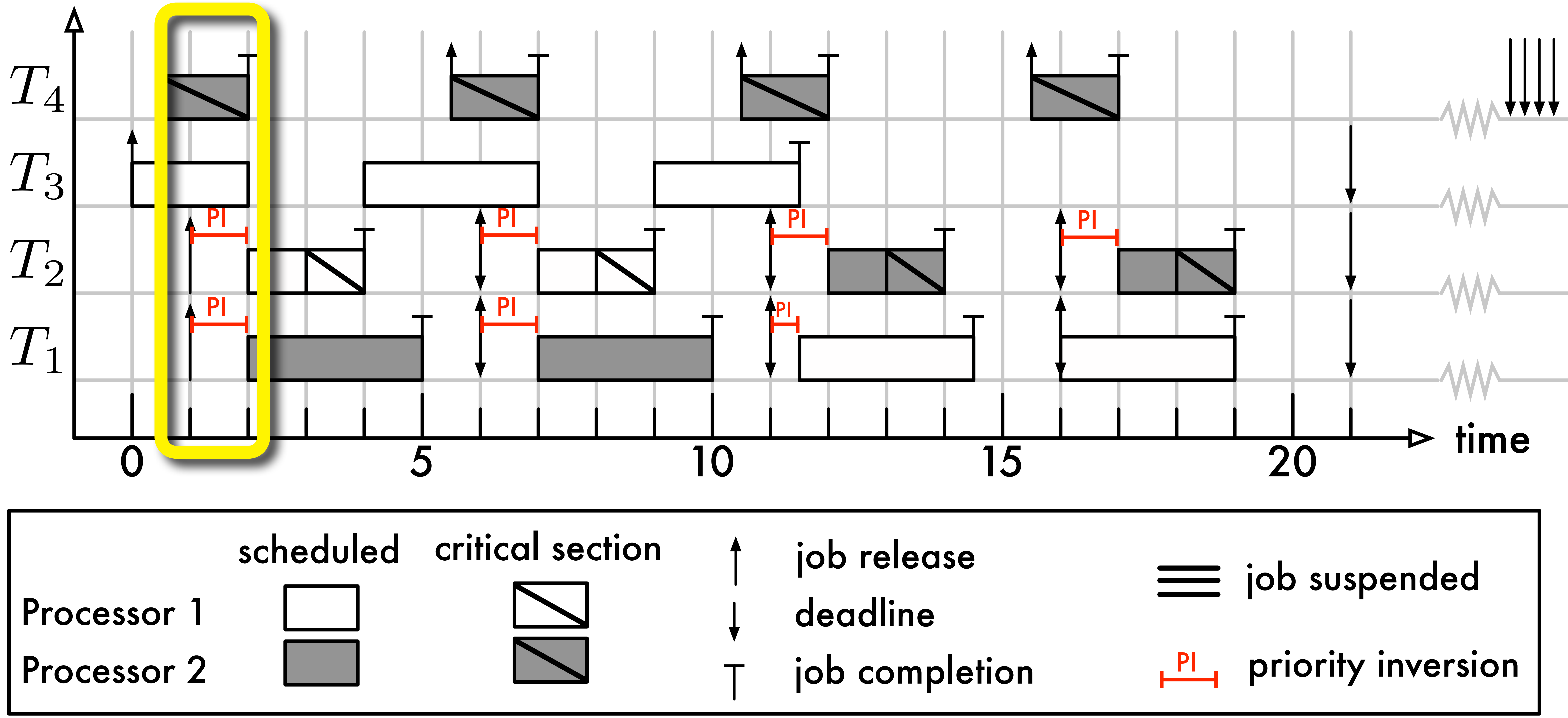
If less than ***c*** jobs scheduled so far: **any other ready jobs**

Select the **highest-priority ready jobs** not yet scheduled (may hold locks).

(any ties broken arbitrarily but consistently)

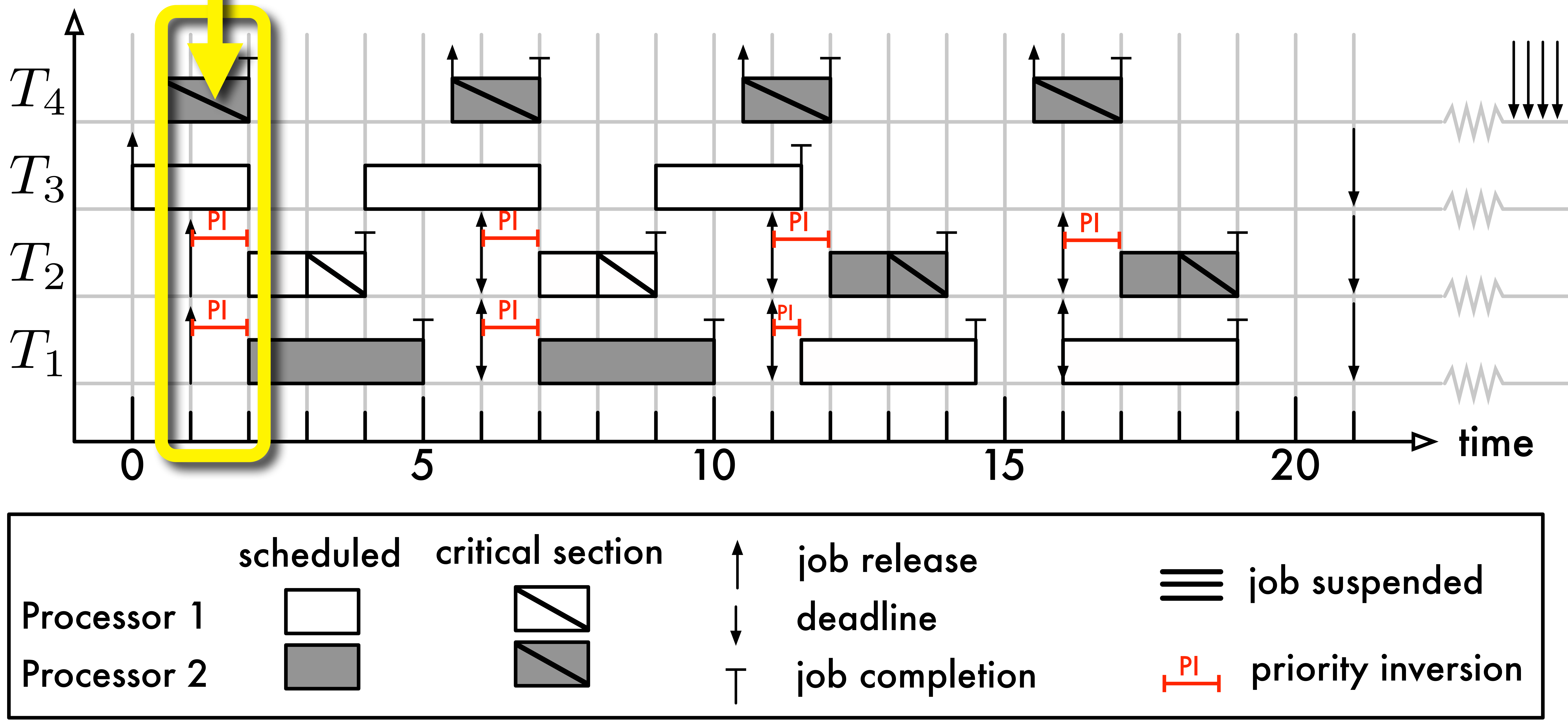
Restricted Segment Boosting at Time 1

Generalized FMLP+ schedule.



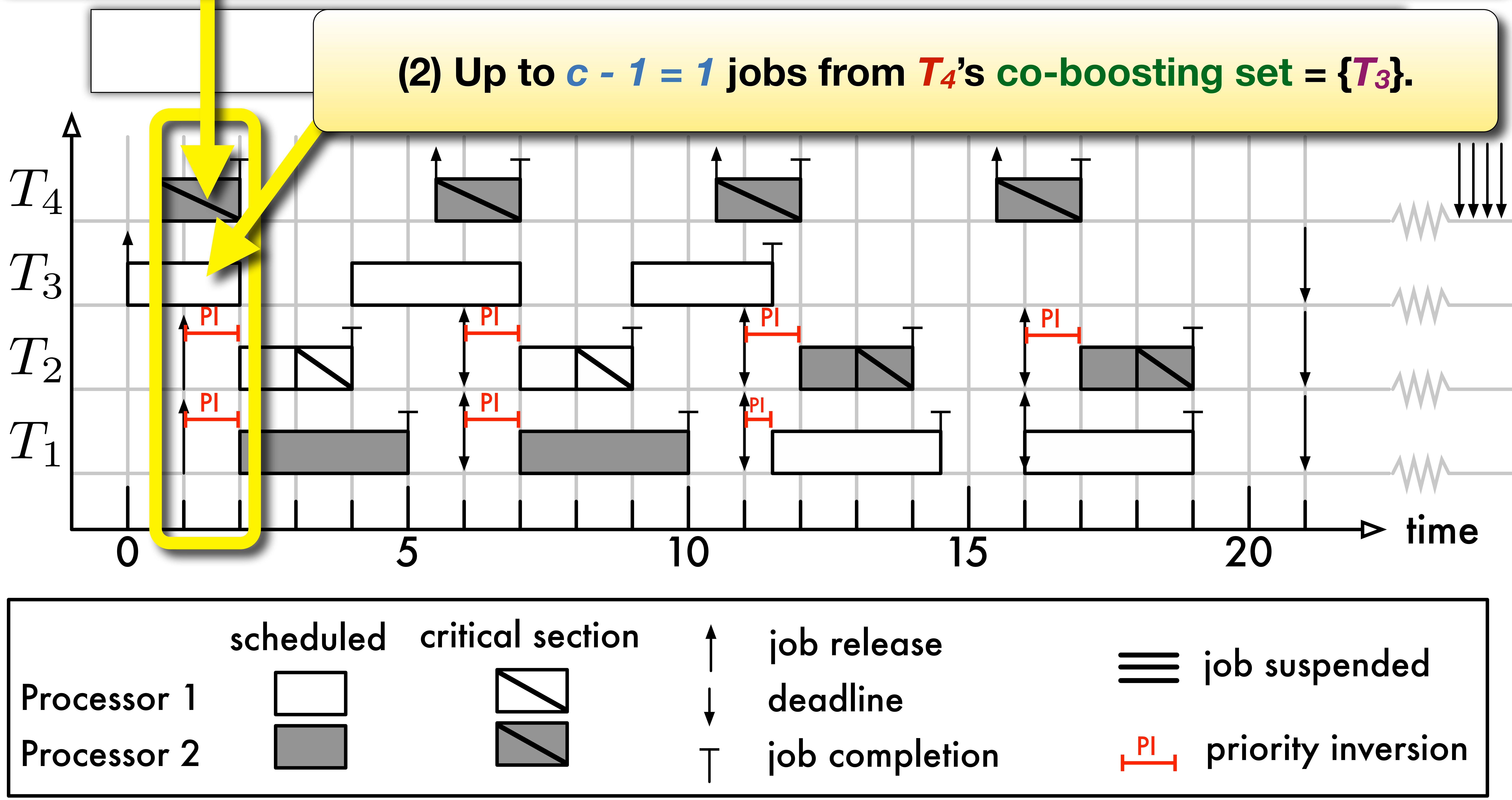
(1) The **lock-holding ready job** (if any) with the **earliest segment start time**.

Generalized FMLP⁺ schedule.



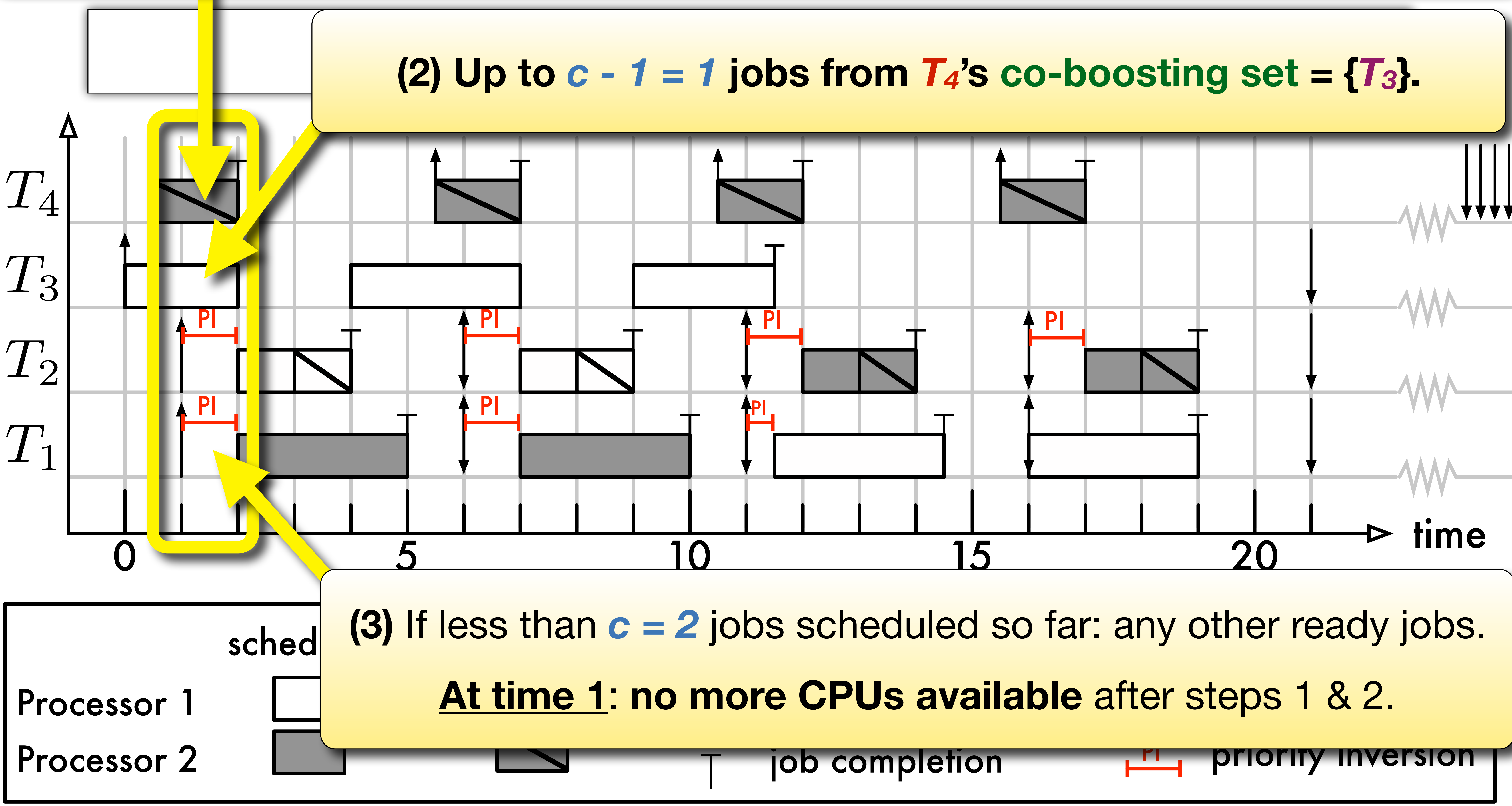
(1) The **lock-holding ready job** (if any) with the **earliest segment start time**.

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



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

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



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

<u>At time 1</u>: no more CPUs available after steps 1 & 2.

The Generalized FIFO Multiprocessor Locking Protocol (FMLP$^+$)

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

The Generalized FIFO Multiprocessor Locking Protocol (FMLP$^+$)

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

S-Aware PI-Blocking per Segment

- …during request segment: ***O(n)***.
  - Proof: rather straightforward → see paper.
- …during independent segment: ***O(n)***.
  - Proof: rather involved → see paper.

The Generalized FIFO Multiprocessor Locking Protocol (FMLP⁺)

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

S-Aware PI-Blocking per Segment

➡ …during request segment: ***O(n)***.
- Proof: rather straightforward → see paper.

➡ …during independent segment: ***O(n)***.
- Proof: rather involved → see paper.

Number of segments

➡ Constant **#requests** and **#self-suspensions** per job
→ **constant number of segments**.

The Generalized FIFO Multiprocessor Locking Protocol (FMLP$^+$)

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

S-Aware PI-Blocking per Segment

- …during request segment: ***O(n)***.
  - Proof: rather straightforward → see paper.
- …during independent segment: ***O(n)***.
  - Proof: rather involved → see paper.

Number of segments

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

Overall Max. S-Aware PI-Blocking

- ***O(n)*** under clustered JLFP scheduling.

Multiprocessor Real-Time Locking Optimality Results

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	The Generalized FMLP⁺ **(restricted segment boosting)** + RNLP [Ward & Anderson, 2012] for nested critical sections
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011] + RNLP [Ward & Anderson, 2012]	

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

The Generalized FMLP⁺ based on Restricted Segment Boosting closes the s-aware asymptotic optimality gap.

*See paper & online appendix for **large-scale empirical evaluation**.*
*(Summary: **the FMLP⁺ works well** if the schedulability analysis is **accurate enough**.)*

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	The Generalized FMLP⁺ **(restricted segment boosting)** + RNLP [Ward & Anderson, 2012] for nested critical sections
Global (jobs migrate freely)	**G-OMLP** [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** [— & Anderson, 2011] + RNLP [Ward & Anderson, 2012]	

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Conclusion

Summary

The Generalized FMLP+

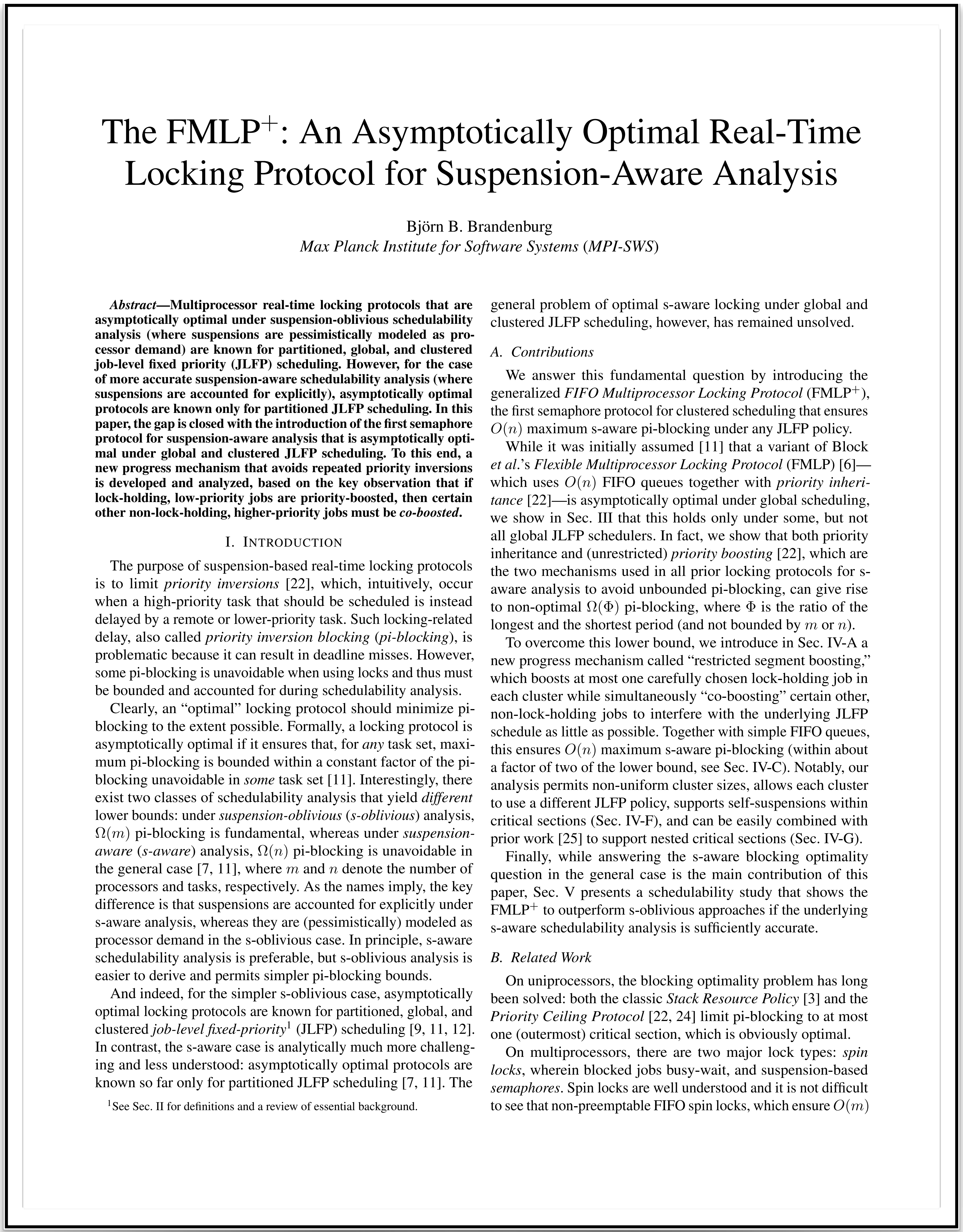
- priority boosting & inheritance **unsuitable**
- based instead on **restricted segment boosting**
  - Key idea: **co-boosting of independent jobs**

In the Paper

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

Online Appendix

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



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

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

***Abstract*—Multiprocessor real-time locking protocols that are asymptotically optimal under suspension-oblivious schedulability analysis (where suspensions are pessimistically modeled as processor demand) are known for partitioned, global, and clustered job-level fixed priority (JLFP) scheduling. However, for the case of more accurate suspension-aware schedulability analysis (where suspensions are accounted for explicitly), asymptotically optimal protocols are known only for partitioned JLFP scheduling. In this paper, the gap is closed with the introduction of the first semaphore protocol for suspension-aware analysis that is asymptotically optimal under global and clustered JLFP scheduling. To this end, a new progress mechanism that avoids repeated priority inversions is developed and analyzed, based on the key observation that if lock-holding, low-priority jobs are priority-boosted, then certain other non-lock-holding, higher-priority jobs must be *co-boosted*.**

I. INTRODUCTION

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

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

And indeed, for the simpler s-oblivious case, asymptotically optimal locking protocols are known for partitioned, global, and clustered *job-level fixed-priority*[1] (JLFP) scheduling [9, 11, 12]. In contrast, the s-aware case is analytically much more challenging and less understood: asymptotically optimal protocols are known so far only for partitioned JLFP scheduling [7, 11]. The general problem of optimal s-aware locking under global and clustered JLFP scheduling, however, has remained unsolved.

A. Contributions

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

While it was initially assumed [11] that a variant of Block *et al.*'s *Flexible Multiprocessor Locking Protocol* (FMLP) [6]—which uses $O(n)$ FIFO queues together with *priority inheritance* [22]—is asymptotically optimal under global scheduling, we show in Sec. III that this holds only under some, but not all global JLFP schedulers. In fact, we show that both priority inheritance and (unrestricted) *priority boosting* [22], which are the two mechanisms used in all prior locking protocols for s-aware analysis to avoid unbounded pi-blocking, can give rise to non-optimal $\Omega(\Phi)$ pi-blocking, where Φ is the ratio of the longest and the shortest period (and not bounded by m or n).

To overcome this lower bound, we introduce in Sec. IV-A a new progress mechanism called "restricted segment boosting," which boosts at most one carefully chosen lock-holding job in each cluster while simultaneously "co-boosting" certain other, non-lock-holding jobs to interfere with the underlying JLFP schedule as little as possible. Together with simple FIFO queues, this ensures $O(n)$ maximum s-aware pi-blocking (within about a factor of two of the lower bound, see Sec. IV-C). Notably, our analysis permits non-uniform cluster sizes, allows each cluster to use a different JLFP policy, supports self-suspensions within critical sections (Sec. IV-F), and can be easily combined with prior work [25] to support nested critical sections (Sec. IV-G).

Finally, while answering the s-aware blocking optimality question in the general case is the main contribution of this paper, Sec. V presents a schedulability study that shows the FMLP$^+$ to outperform s-oblivious approaches if the underlying s-aware schedulability analysis is sufficiently accurate.

B. Related Work

On uniprocessors, the blocking optimality problem has long been solved: both the classic *Stack Resource Policy* [3] and the *Priority Ceiling Protocol* [22, 24] limit pi-blocking to at most one (outermost) critical section, which is obviously optimal.

On multiprocessors, there are two major lock types: *spin locks*, wherein blocked jobs busy-wait, and suspension-based *semaphores*. Spin locks are well understood and it is not difficult to see that non-preemptable FIFO spin locks, which ensure $O(m)$

[1] See Sec. II for definitions and a review of essential background.

[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *Improved Analysis and Evaluation of Real-Time Semaphore Protocols for P-FP Scheduling*, RTAS 2013.

Future Work & Open Questions

Apply this technique to reader-writer locks?

➡ Lower bounds on s-aware pi-blocking?

Apply this technique to k-exclusion locks?

➡ GPUs & other co-processors

➡ Lower bounds on s-aware pi-blocking?

Overheads of restricted segment boosting?

➡ Tracking **segment start times** is simple and cheap.

➡ But… additional preemptions?

Multiprocessor Real-Time Locking Optimality Results

JLFP *job-level fixed-priority*	Suspension Oblivious Any JLFP Scheduler	Suspension Aware Any JLFP Scheduler
Partitioned (no migrations)	**P-OMLP** ✓ [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	The Generalized FMLP+ **(restricted segment boosting)** ✓ + RNLP [Ward & Anderson, 2012] for nested critical sections
Global (jobs migrate freely)	**G-OMLP** ✓ [— & Anderson, 2010] + RNLP [Ward & Anderson, 2012]	
Clustered (jobs migrate only among subset of processors)	**OMIP** [—, 2013] **C-OMLP** ✓ [— & Anderson, 2011] + RNLP [Ward & Anderson, 2012]	

[Block et al., 2007] *A Flexible Real-Time Locking Protocol for Multiprocessors*, RTCSA 2007.
[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.
[— & Anderson, 2011] *Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks*, EMSOFT 2011.
[—, 2011] *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*, PhD thesis, UNC, 2011.
[Ward & Anderson, 2012] *Supporting Nested Locking in Multiprocessor Real-Time Systems*, ECRTS 2012.
[—, 2013] *A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications*, ECRTS 2013.

Appendix

Multiprocessor Real-Time Locking
Optimality Classes

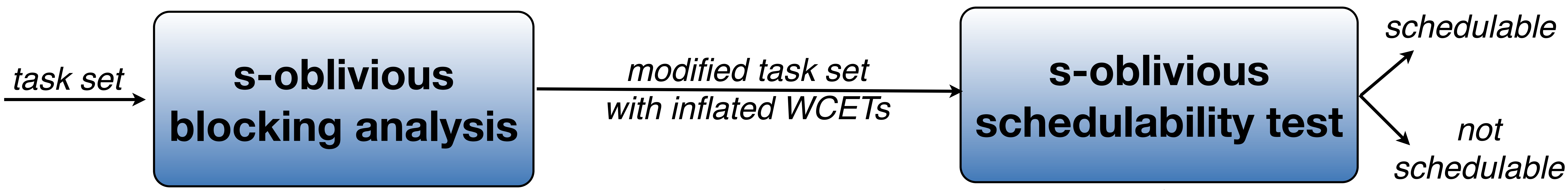
Blocking Optimality [– & Anderson, 2010]	suspension **oblivious**	suspension **aware**
How are suspensions analyzed?	CPU demand is **over-approximated**	CPU demand is modeled **accurately**
Lower bound on maximum priority inversion blocking $max_i\{b_i\}$	**$\Omega(m)$** m = #CPUs	**$\Omega(n)$** n = #tasks

[— & Anderson, 2010] *Optimality Results for Multiprocessor Real-Time Locking*, RTSS 2010.

S-Aware vs. S-Oblivious Analysis

S-Aware vs. S-Oblivious Analysis

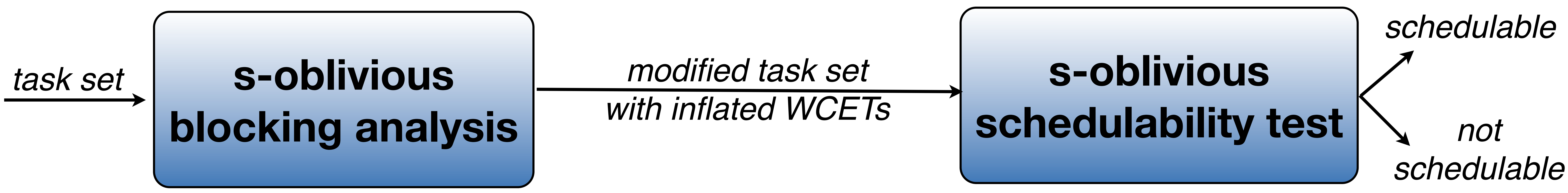
Suspension-oblivious (***s-oblivious***) Analysis



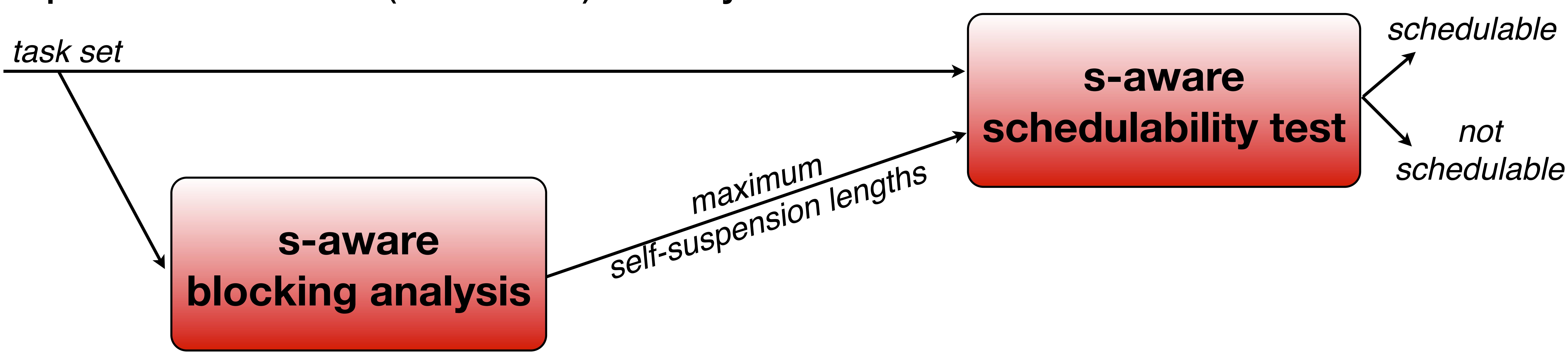
Motivation: *reuse existing schedulability analyses that assume **independent**, **always ready** tasks.*

S-Aware vs. S-Oblivious Analysis

Suspension-oblivious (*s-oblivious*) Analysis



Suspension-aware (*s-aware*) Analysis

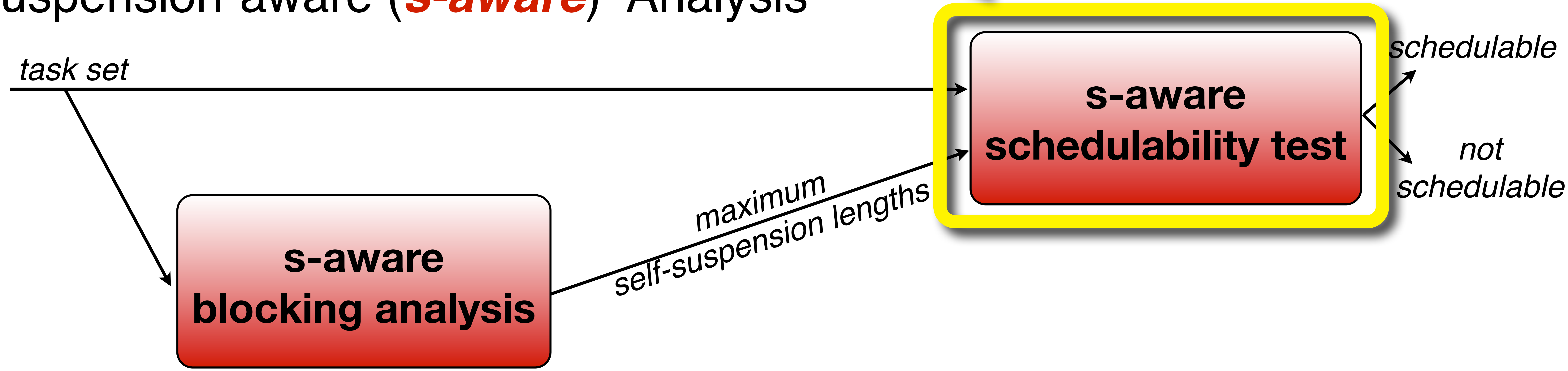


Requires availability of a schedulability test that accounts (reasonably accurately) for self-suspensions.

(…which can be tricky to derive)

Suspension-oblivious (*s-oblivious*) Analysis

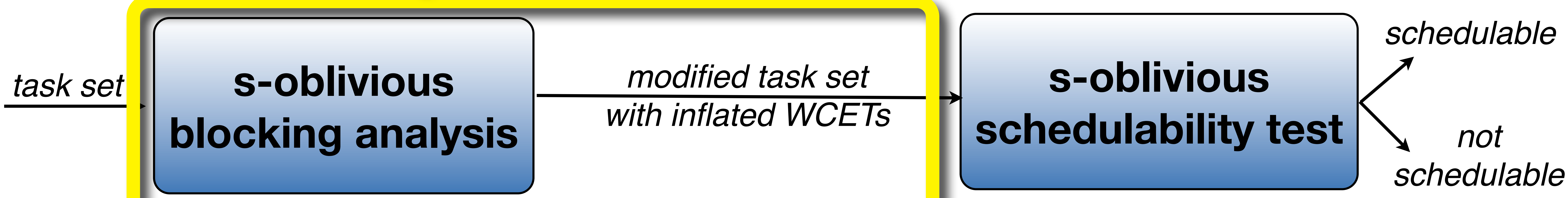
Suspension-aware (*s-aware*) Analysis



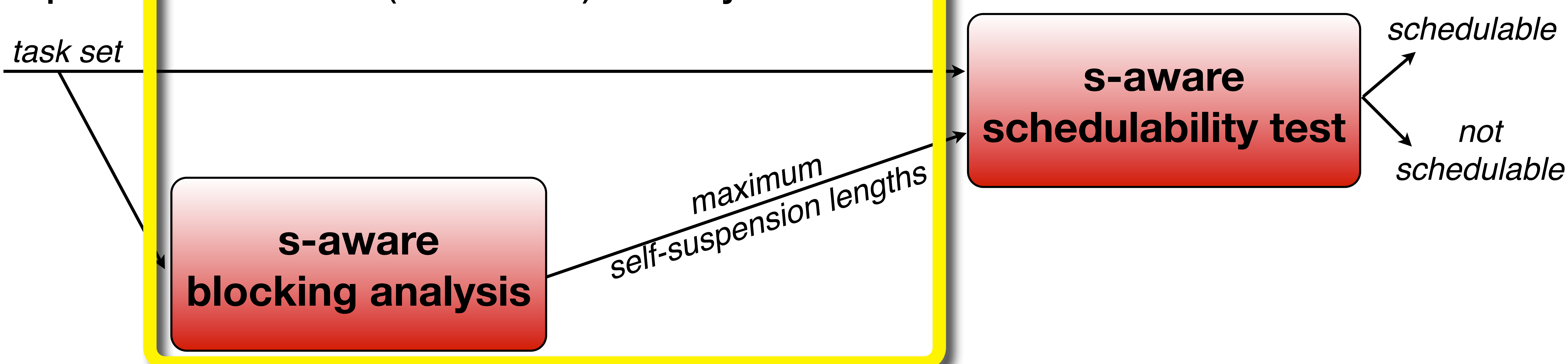
Different notions of “processor demand”
→ different definitions of “priority inversion”.

s Analysis

Suspension-oblivious (*s-oblivious*) Analysis

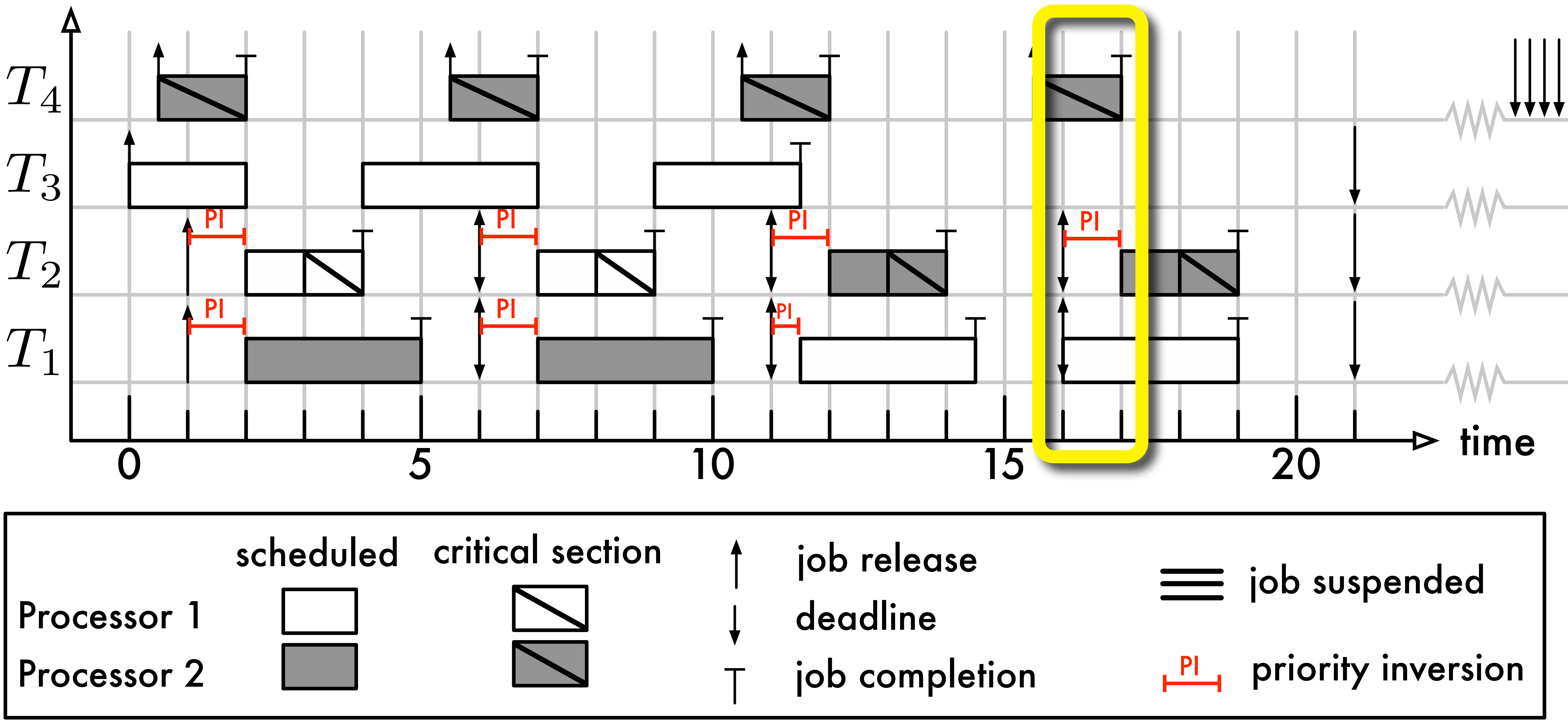


Suspension-aware (*s-aware*) Analysis



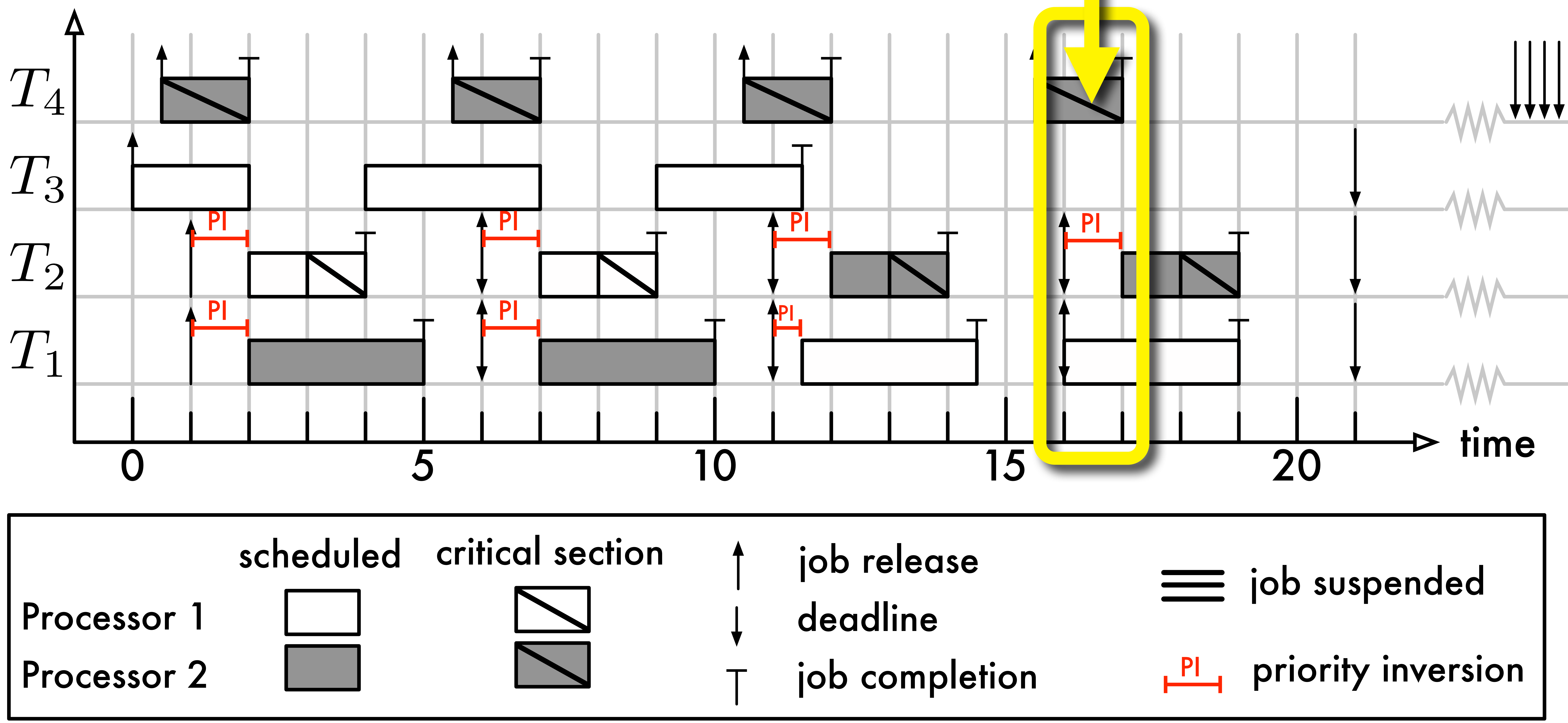
Restricted Segment Boosting at Time 16

Generalized FMLP+ schedule.



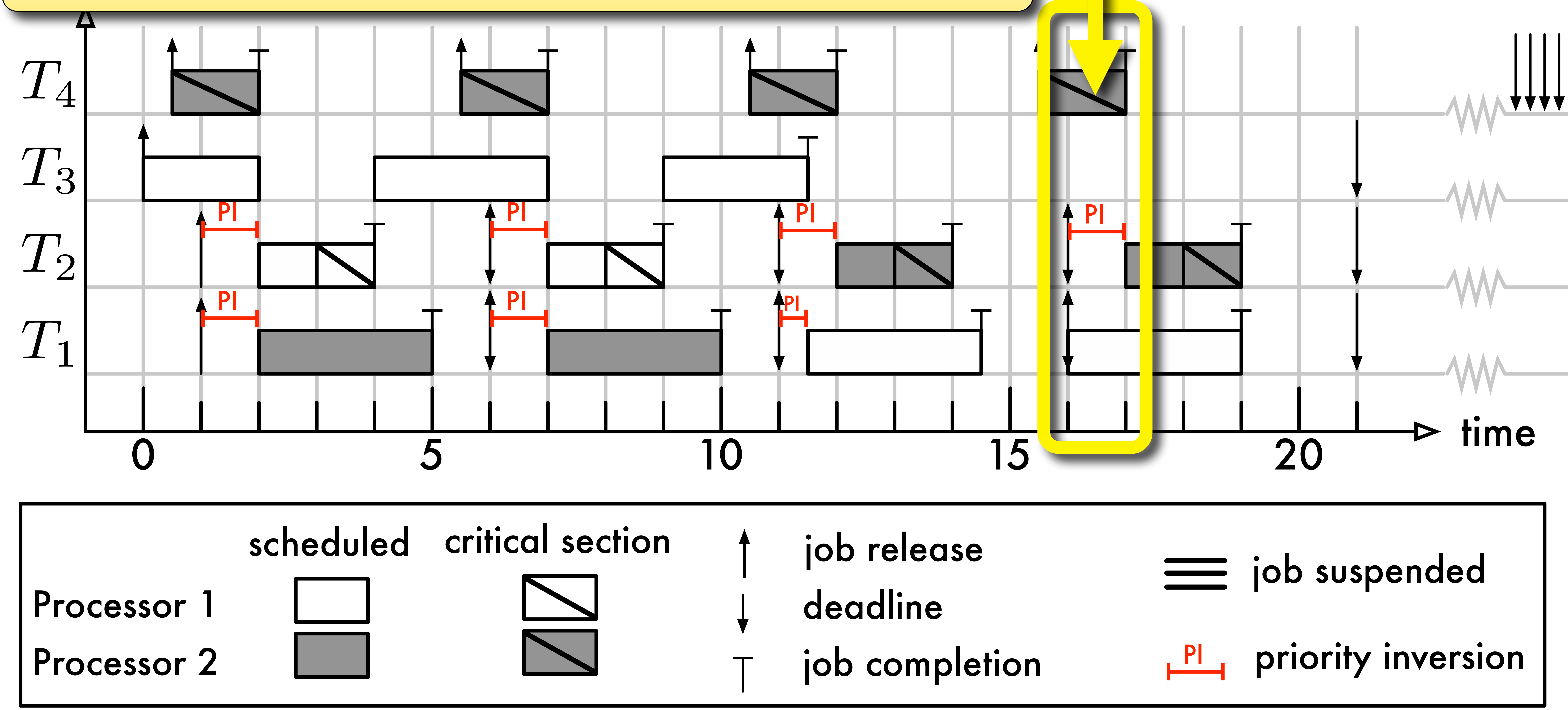
(1) The **lock-holding ready job** (if any) with the **earliest segment start time**.

Generalized FMLP⁺ schedule.



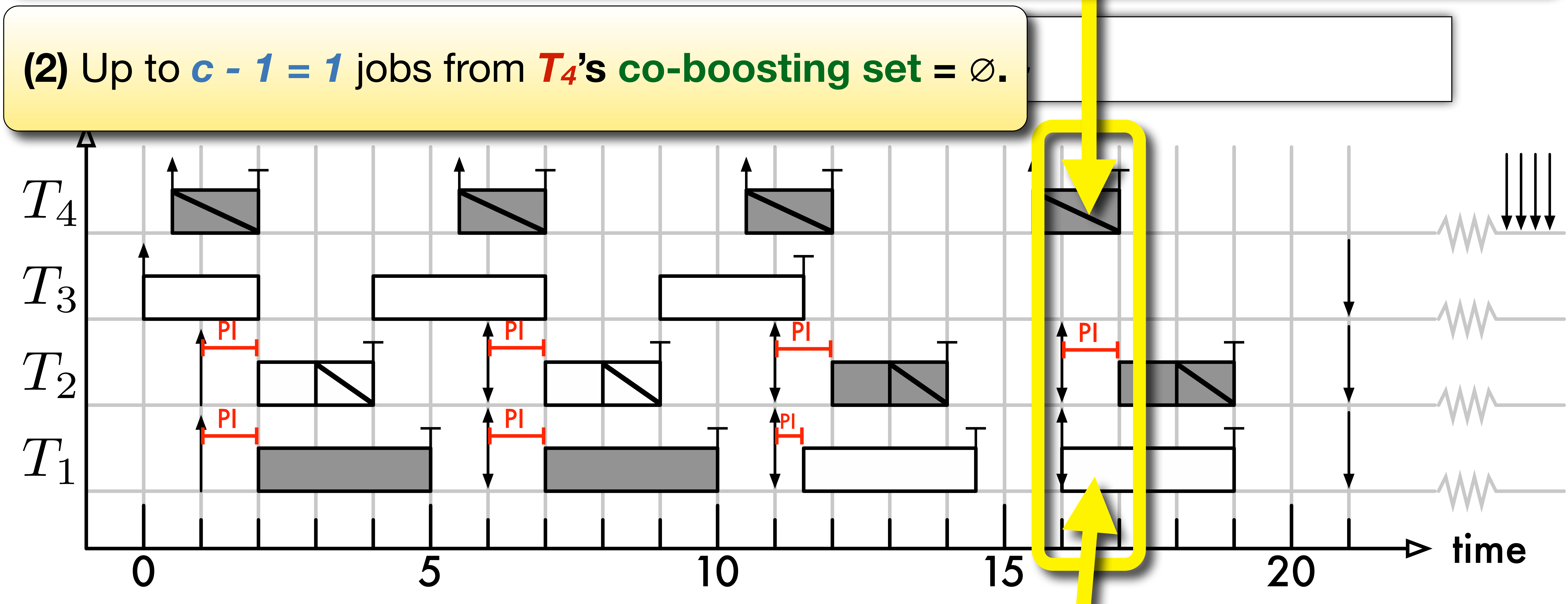
(1) The **lock-holding ready job** (if any) with the **earliest segment start time**.

(2) Up to ***c* - *1* = *1*** jobs from ***T₄*'s co-boosting set = ∅.**



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

(2) Up to ***c* - *1* = *1*** jobs from ***T_4***'s **co-boosting set** = ∅.



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

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

Definition: Job Segments

a job at runtime:

Independent segment

- ➡ starts when a job is **released** or **resumed**, or when it **unlocks a resource**
- ➡ ends when job **completes**, **suspends**, or **requests a lock**

Request segment

- ➡ starts when a job **requests a lock**
- ➡ ends when it **unlocks the resource**