The Case for **Migratory Priority Inheritance** in Linux: Bounded Priority Inversions on Multiprocessors

Björn Brandenburg
MPI-SWS

Max Planck Institute for Software Systems

Andrea Bastoni
SYSGO AG

SYSGO EMBEDDING INNOVATIONS

Why “Migratory”?

Classic uniprocessor priority inheritance is **ineffective** under non-global scheduling (Linux).

The **most-studied** multiprocessor real-time locking primitive is **not a good fit for Linux**.

A (simple) "**tweak**" to Linux's **existing** priority inheritance solution **restores predictability**.

Part 1

Classic uniprocessor priority inheritance is **ineffective** under non-global scheduling (Linux).

But it works great on uniprocessors…

Why is **Classic** Priority Inheritance **Effective** on Uniprocessors?

Classic Priority Inheritance

=

Blocking task is scheduled with (at least) the priority of blocked task.

Effective on Uniprocessors

=

“Priority inversion” when blocking on a lock
is limited to duration of one critical section (per lock acquisition).

Analysis of Fixed-Priority Scheduling (`SCHED_FIFO`)

maximum ***response time*** *≤ relative* ***deadline***

Analysis of Fixed-Priority Scheduling (`SCHED_FIFO`)

response time
(time for a task to react to input)

=

own execution
(time to compute response)

\+

execution of higher-priority tasks
(preemptions / scheduling delays due to higher-priority tasks)

Lower-priority *tasks do not cause delays if tasks are* ***independent.***

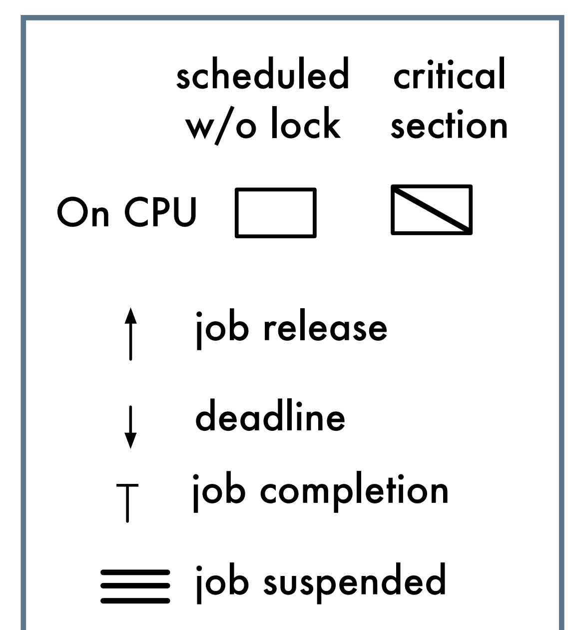
Analysis of Fixed-Priority Scheduling (`SCHED_FIFO`)

response time (time for a task to react to input)	=	**own execution** (time to compute response)
	+	**execution of higher-priority tasks** (preemptions / scheduling delays due to higher-priority tasks)
	+	**durations of priority inversion** (any delay not attributable to higher-priority tasks)

Priority inversion: *any delay due to* ***lower-priority*** *tasks.*

Example Task Set

Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96

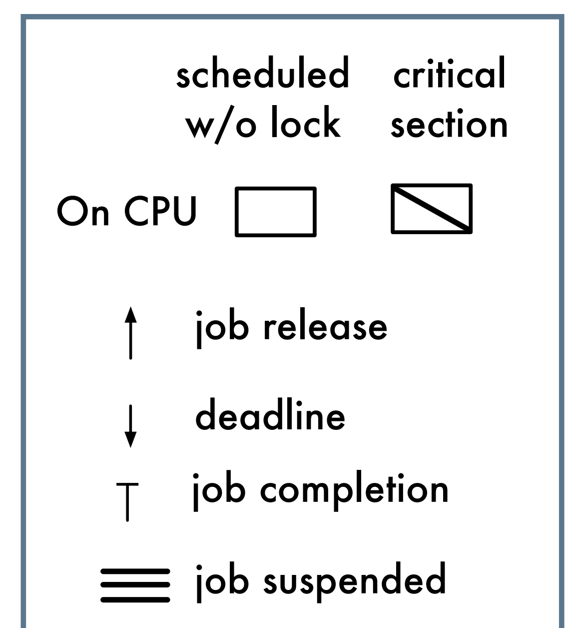


Example Task Set

<u>W</u>orst-<u>C</u>ase <u>E</u>xecution <u>T</u>ime

How much CPU time required to react to input event in the worst case?

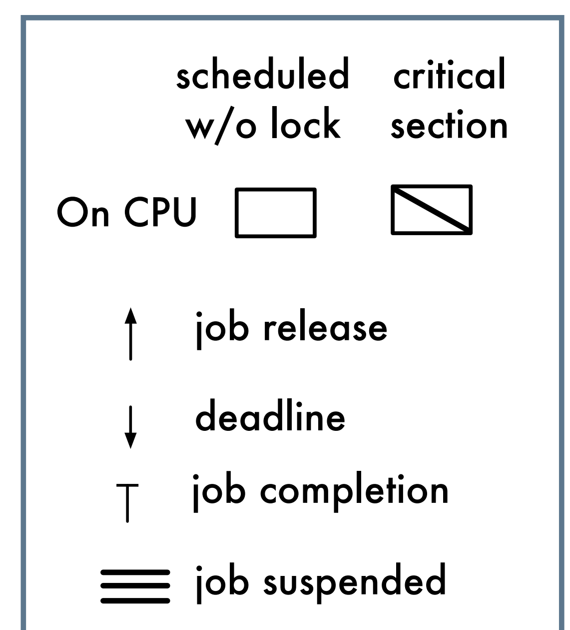
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



Example Task Set

How long may a response be delayed?

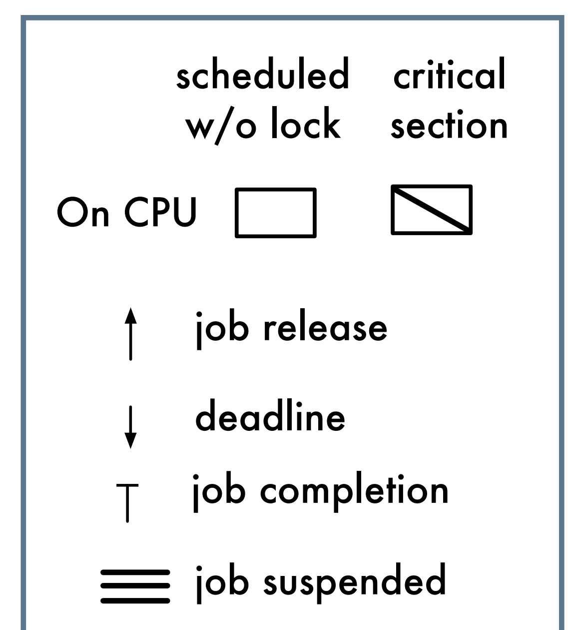
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



Example Task Set

How frequently does new input arrive?

Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



Example Task Set

Deadline-Monotonic Priorities

(shorter relative deadline ⇒ higher priority)

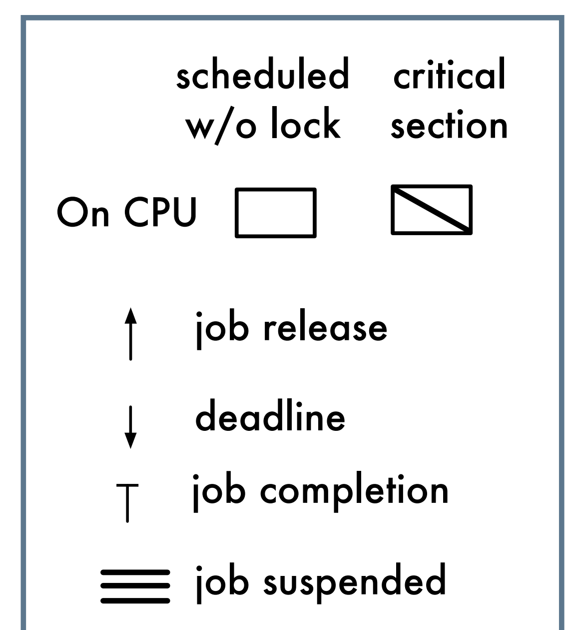
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96

scheduled w/o lock
critical section
On CPU
job release
deadline
job completion
job suspended

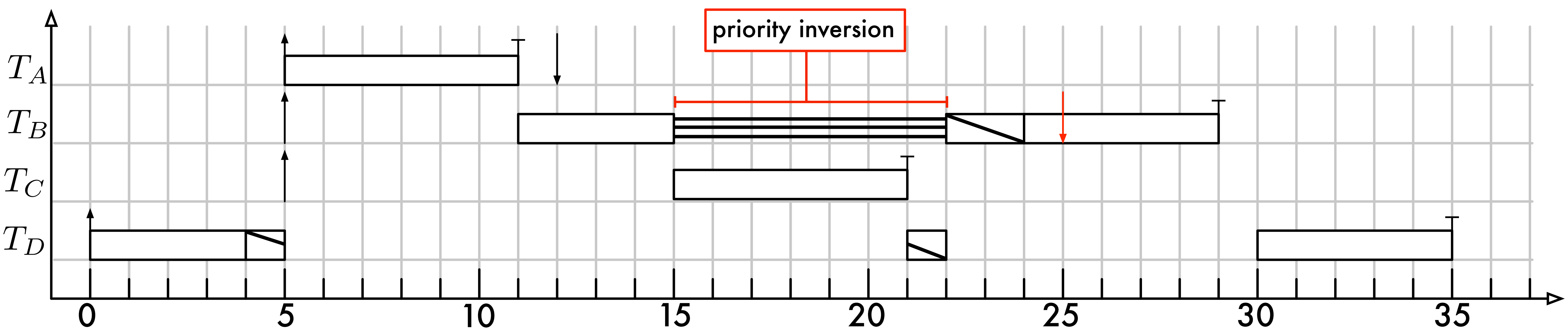
Example Task Set

Tasks T_B and T_D share a data structure protected by a lock.

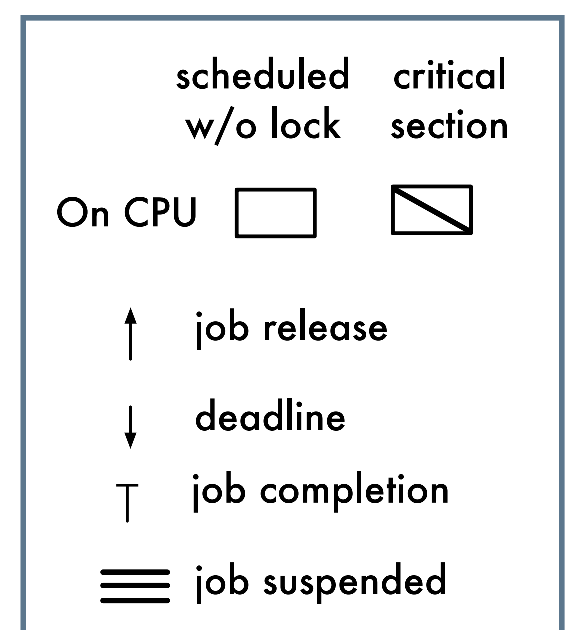
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



Example: Unbounded Priority Inversion



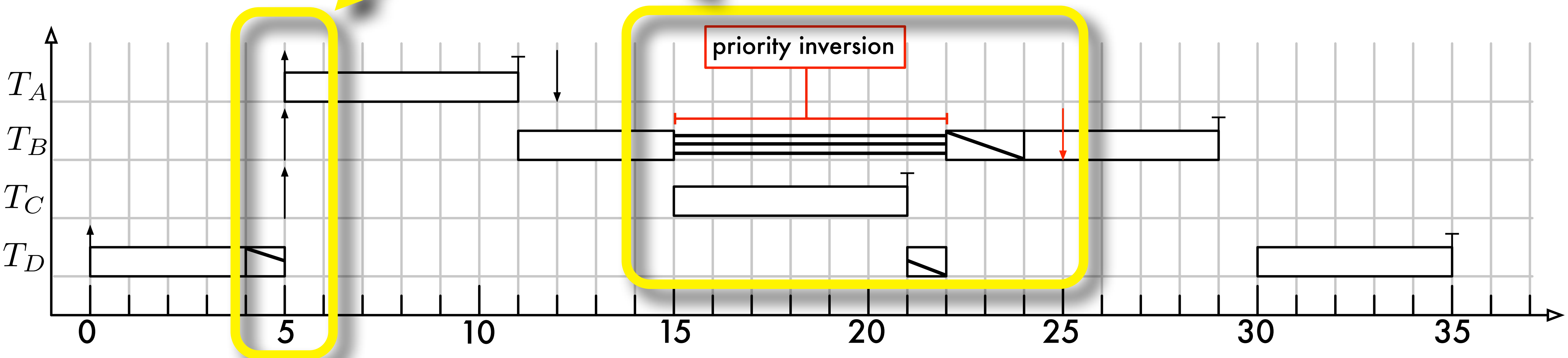
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



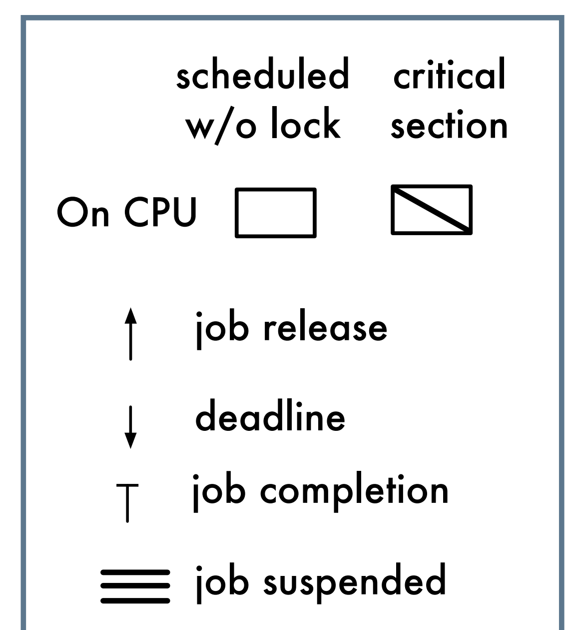
T_D is preempted while holding a lock.

T_B blocks on the lock and is delayed while T_C executes.

Unbounded Priority Inversion

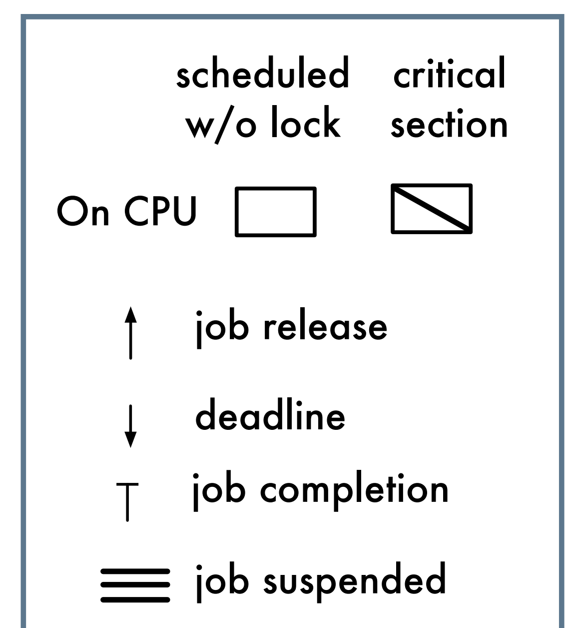


Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96

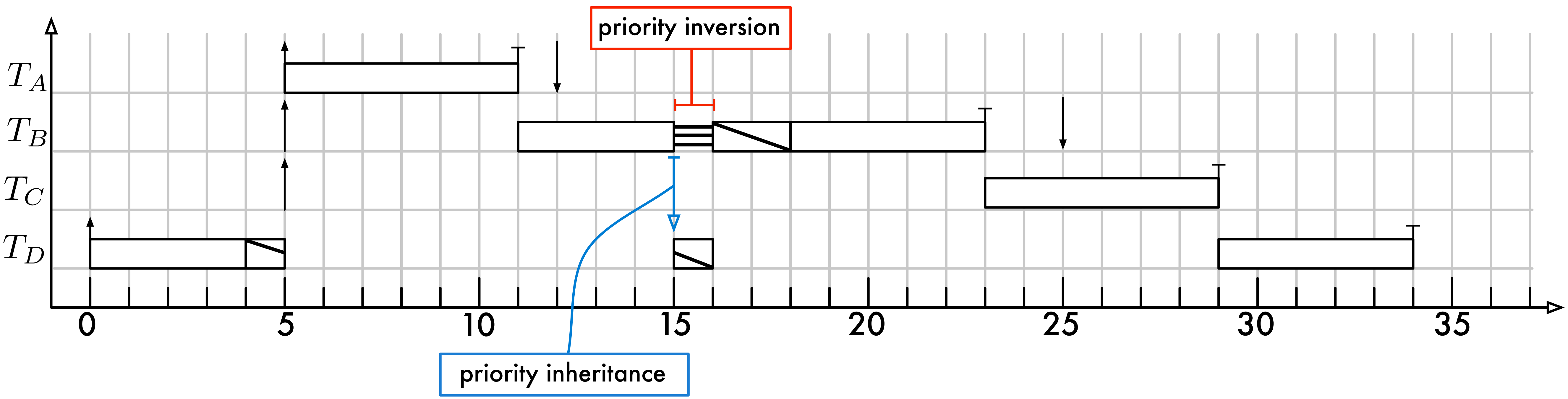


Example: Priority Inheritance

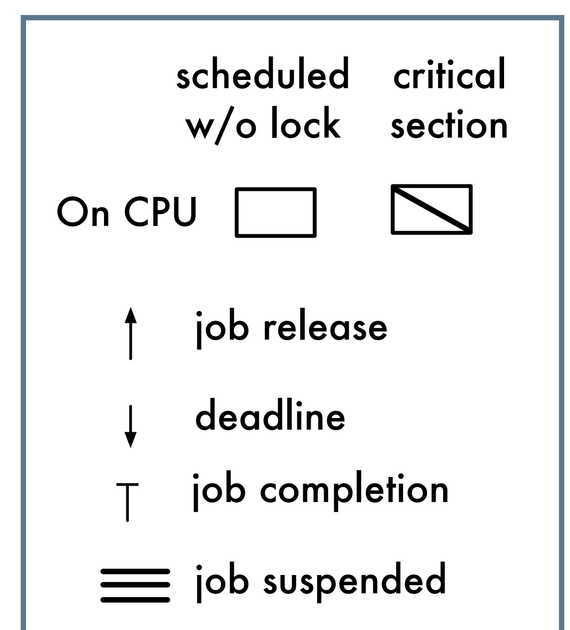
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



Example: Priority Inheritance



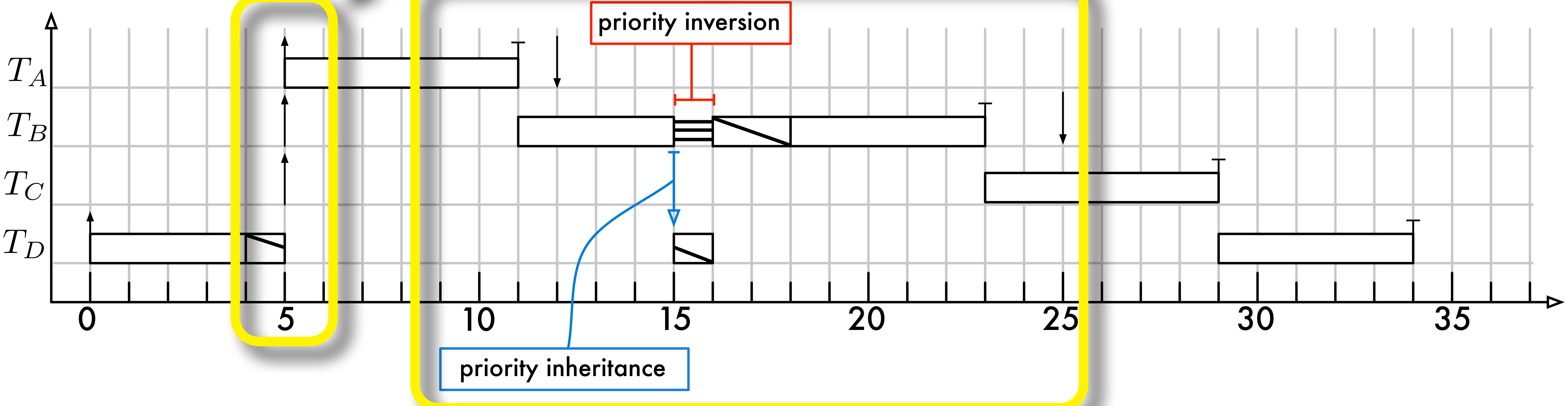
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



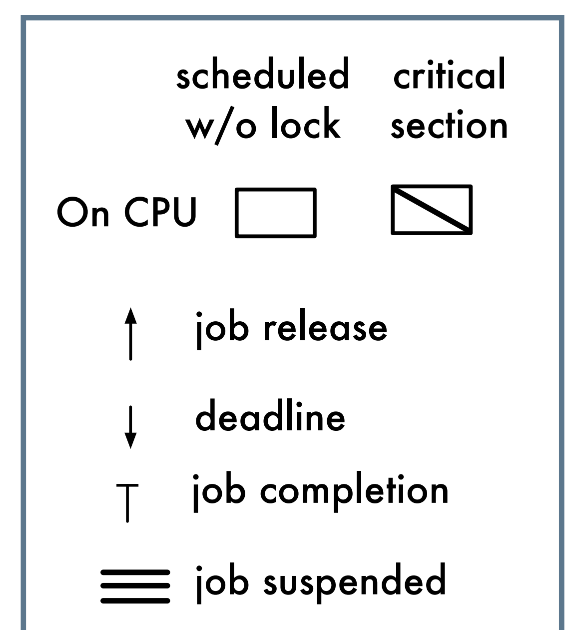
T_D is preempted while holding a lock.

T_C cannot preempt T_D while T_B blocks on the lock due to priority inheritance.

Priority Inheritance



Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96



Why is Priority Inheritance **Ineffective** under **Non-Global** Scheduling?

non-global multiprocessor scheduling
=
not every task may execute on every processor

*This talk: **partitioned** scheduling = each task assigned to one CPU.*

Why is Priority Inheritance **Ineffective** under **Non-Global** Scheduling?

non-global multiprocessor scheduling
=
not every task may execute on every processor

*This talk: **partitioned** scheduling = each task assigned to one CPU.*

priority inheritance is **ineffective**
=
priority inversions are *not always*
limited to the lengths of critical sections

Analysis of **Partitioned** Fixed-Priority Scheduling

response time
(time for a task to react to input)

=

own execution
(time to compute response)

+

execution of local, higher-priority tasks
(preemptions / scheduling delays due to local, higher-priority tasks)

Lower-priority *and* ***remote*** *tasks do not cause delays if tasks are* ***independent.***

Analysis of **Partitioned** Fixed-Priority Scheduling

response time
(time for a task to react to input)

=

own execution
(time to compute response)

\+

execution of <u>local</u>, higher-priority tasks
(preemptions / scheduling delays due to local, higher-priority tasks)

\+

durations of priority inversion
(any delay not attributable to local, higher-priority tasks)

<u>Priority inversion</u>: *any delay due to **local**, **lower-priority** or **remote** tasks.*

Analysis of **Partitioned** Fixed-Priority Scheduling

response time
(time for a task to react to input)

=

own execution
(time to compute response)

\+ **execution of <u>local</u>, higher-priority tasks**
(preemptions / scheduling delays due to local, higher-priority tasks)

\+ **durations of priority inversion**
(any delay not attributable to local, higher-priority tasks)

Remote higher-priority tasks are problematic…

<u>Priority inversion</u>: *any delay due to* ***local***, ***lower-priority*** *or* ***remote*** *tasks.*

Motivation: Increased Frequency

Uniprocessor Task Set

Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96

Multiprocessor Task Set

Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	**1**
T_B	11	20	20	2	**97**	**1**
T_C	6	**20**	**7**	—	**98**	**2**
T_D	11	**20**	**20**	2	96	**2**

Motivation: Increased Frequency

Uniprocessor Task Set

Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96

Operating frequency of tasks T_C and T_D is **scaled up** from **5 Hz** to **50 Hz.**

Switched priorities: T_C has a shorter deadline now.

Multiprocessor Task Set

Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	**1**
T_B	11	20	20	2	**97**	**1**
T_C	6	**20**	**7**	—	**98**	**2**
T_D	11	**20**	**20**	2	96	**2**

Motivation: Increased Frequency

Uniprocessor Task Set

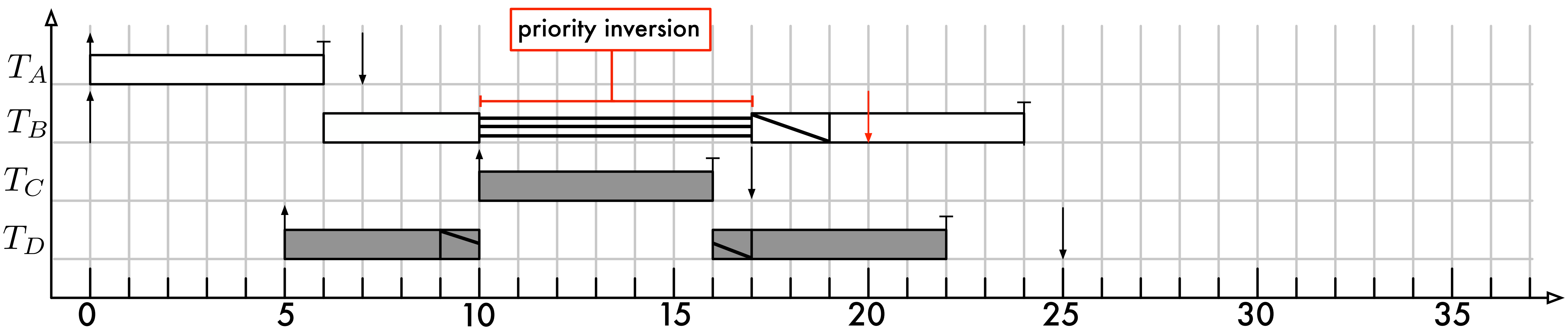
Task	WCET	Period	Deadline	Critical Section	Priority
T_A	6	20	7	—	99
T_B	11	20	20	2	98
T_C	6	200	70	—	97
T_D	11	200	200	2	96

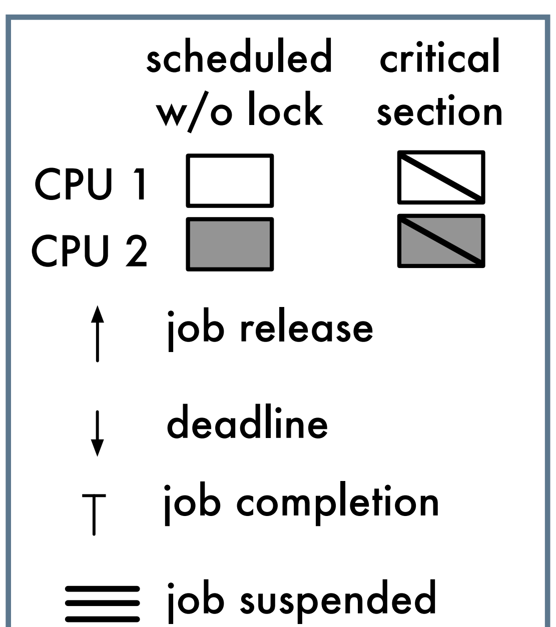
Multiprocessor Task Set

Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	**1**
T_B	11	20	20	2	**97**	**1**
T_C	6	**20**	**7**	—	**98**	**2**
T_D	11	**20**	**20**	2	96	**2**

Symmetric workloads on the two processors.

Multiprocessor Example:
Priority Inheritance is **Ineffective**



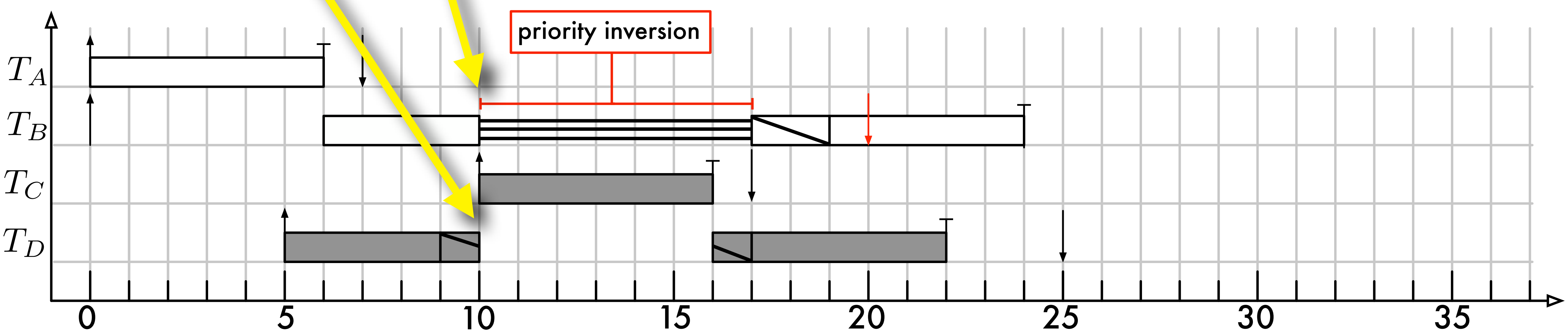


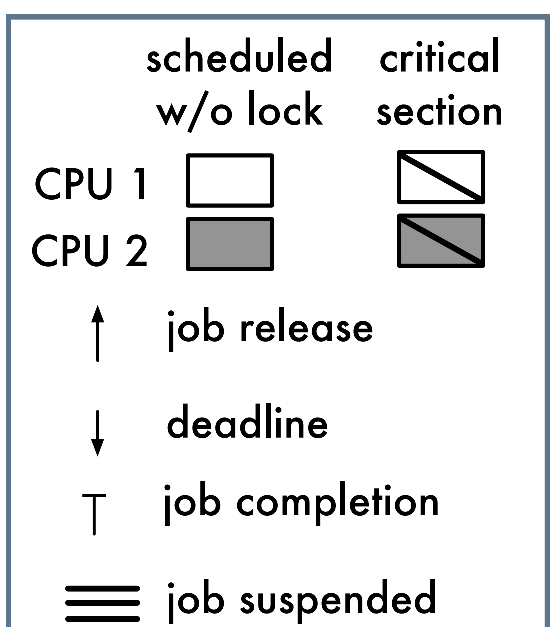
Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

T_D **is again preempted while holding a lock.**

Despite priority inheritance, T_C **preempts** T_D **while** T_B **is blocked.**

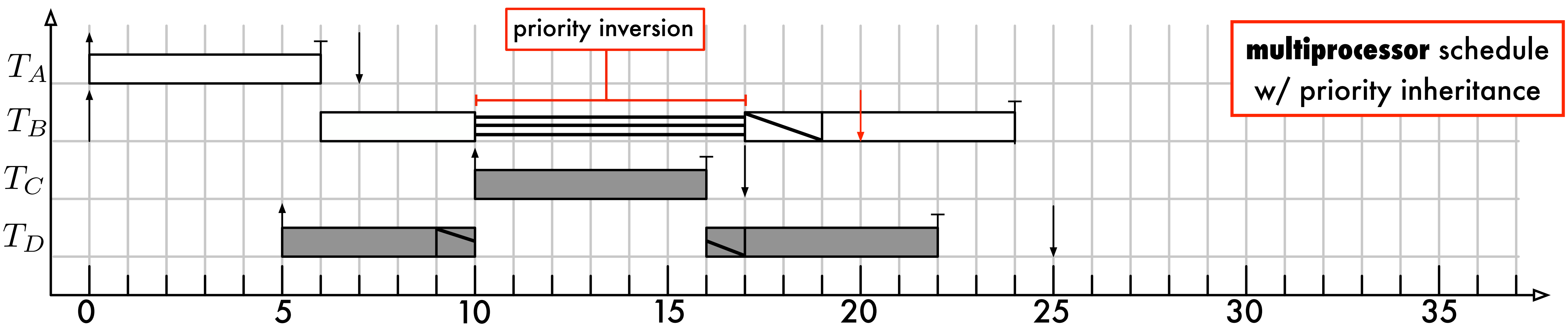
Priority Inheritance is Ineffective

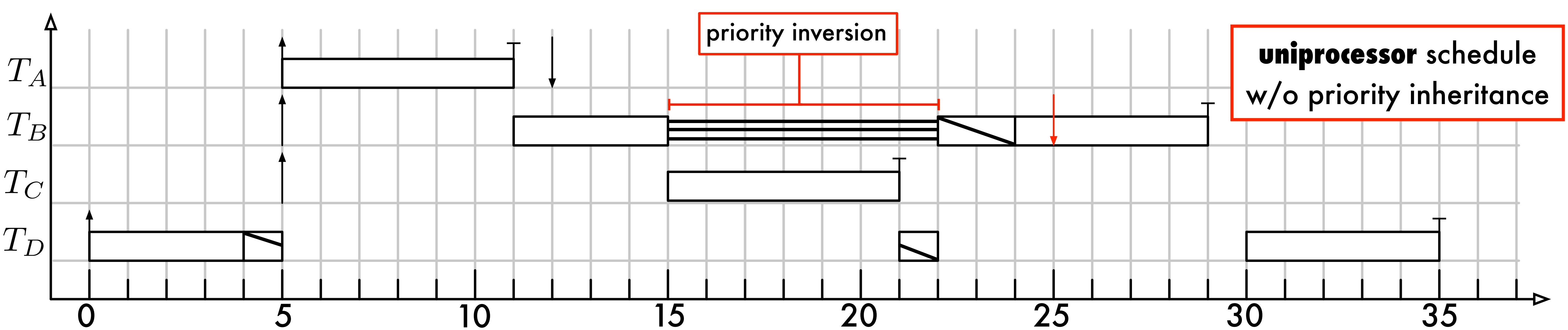




Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

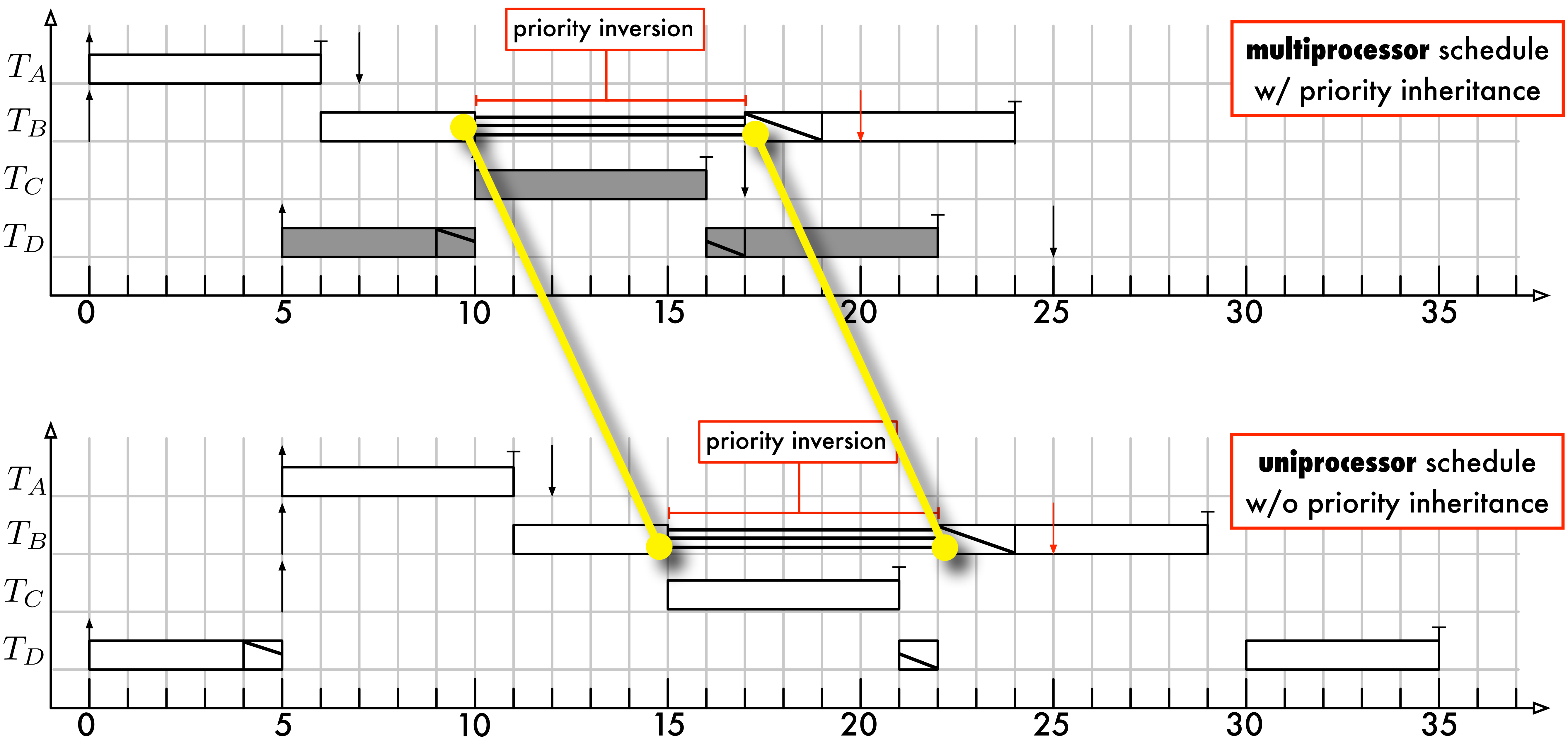
Uniprocessor **w/o** PI vs. Multiprocessor **with** PI





Ineffective

Despite priority inheritance, **no reduction in worst-case priority inversion length!**



Summary: Classic Priority Inheritance

- Great solution on **uniprocessors**, essential to Linux's success as a real-time platform.
- The key property of priority inheritance breaks on non-globally scheduled **multiprocessors**.
- Changing priorities or processor assignment **not always a viable workaround**.

Part 2

The **most-studied** multiprocessor real-time locking primitive is **not a good fit for Linux**.

The Standard Solution

in real-time locking protocols for partitioned scheduling

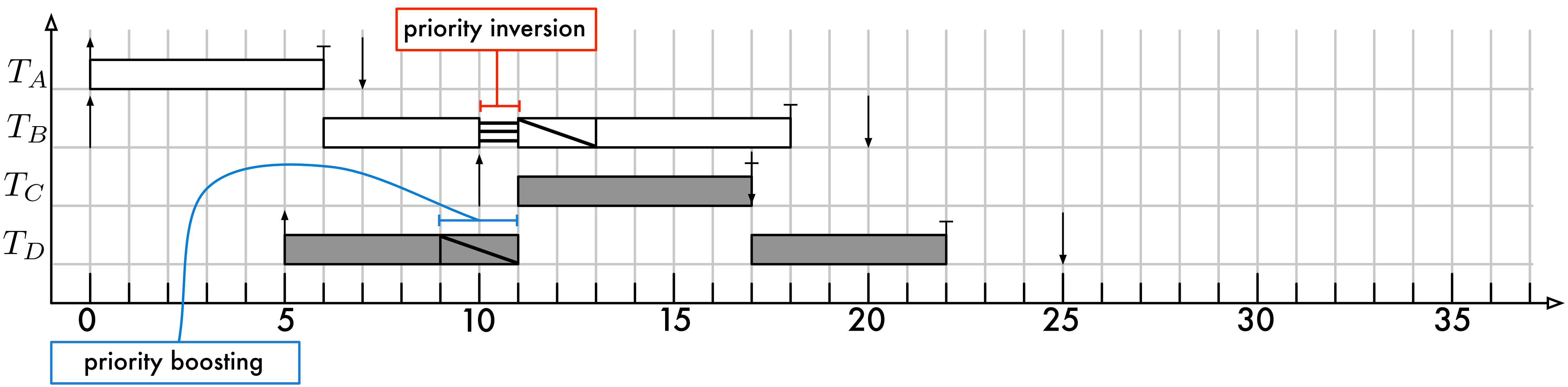
Root problem: preemption of lock-holding tasks.

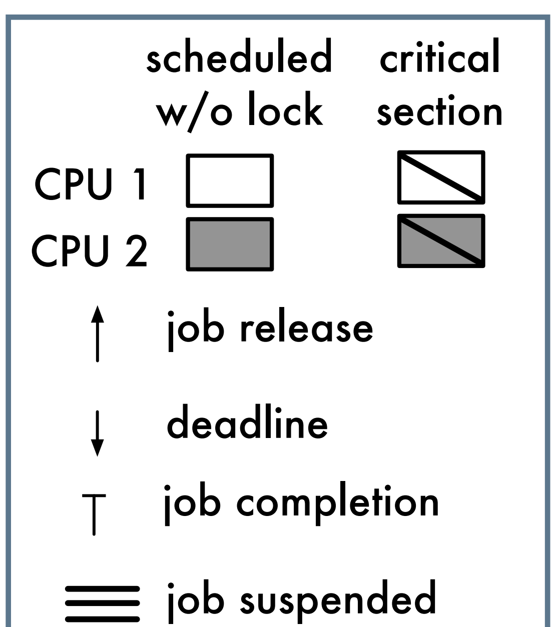
Priority Boosting

Temporarily **raise the effective priority** of tasks in critical sections above that of "normal" tasks.

(Rajkumar et al., 1988; Rajkumar, 1990)

Example: Priority Boosting avoids Lock-Holder Preemptions



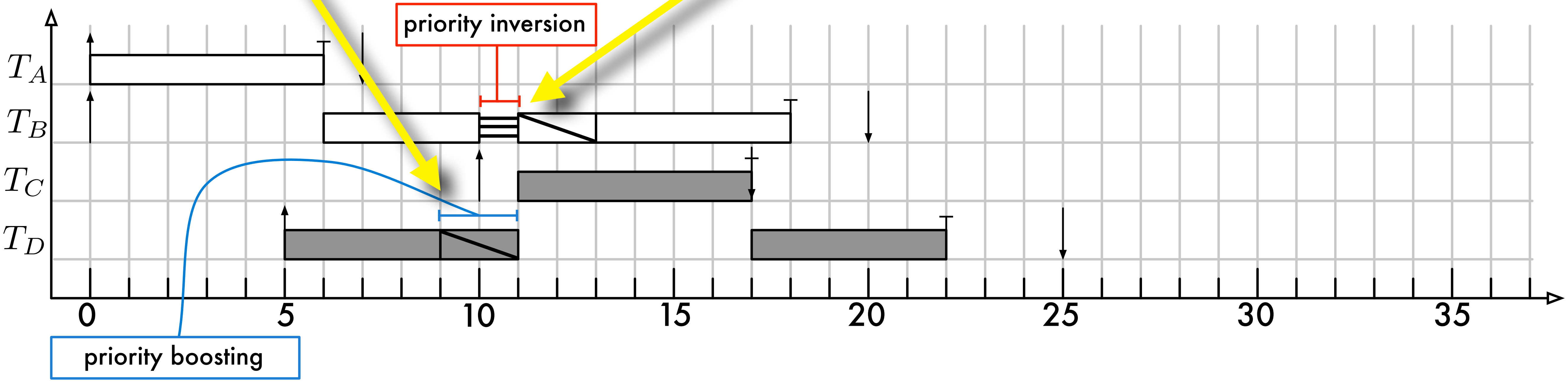


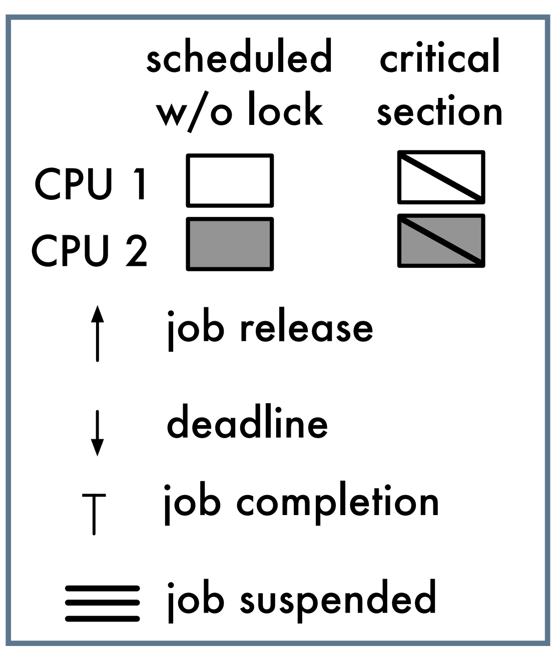
Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

A **higher-priority task** is triggered while T_D is holding a lock.

Due to priority boosting, T_C cannot preempt T_D while T_B is blocked.

avoids Lock-Holder Preemptions





Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

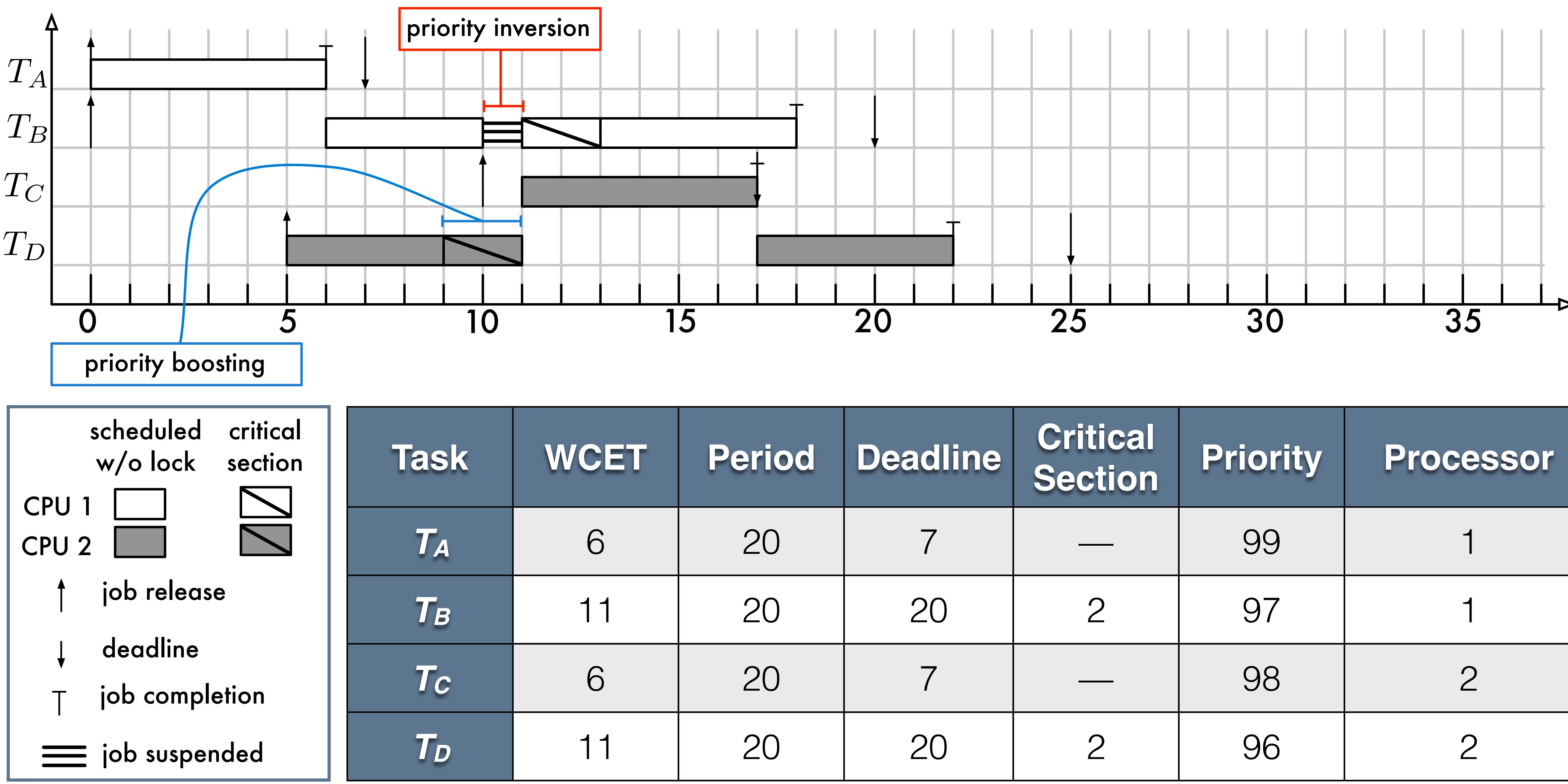
But there's a catch...

What if one of the "normal" tasks is ***urgent***
and ***cannot tolerate delays****?*

How is priority boosting different from turning off interrupts?

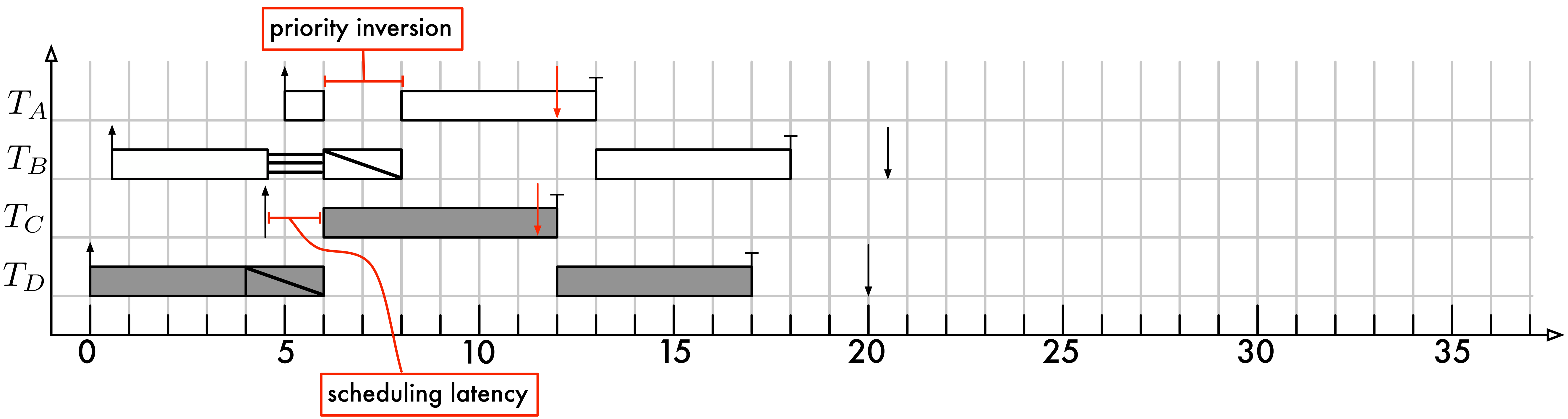
(In the worst case, it isn't.)

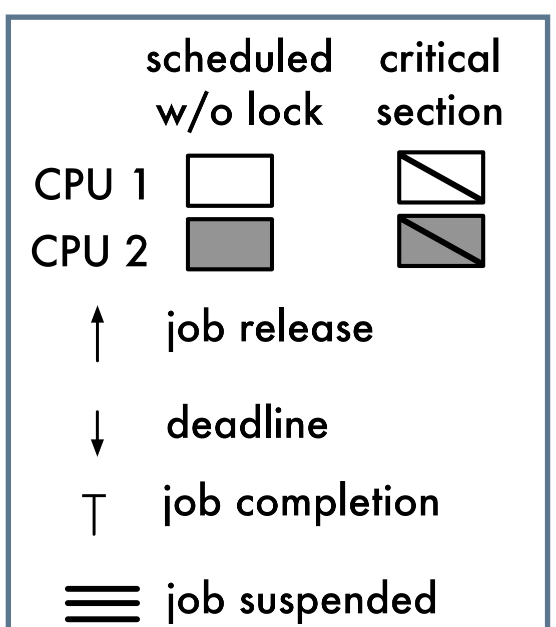
Example: Priority Boosting avoids Lock-Holder Preemptions



Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Example: Priority Boosting Increases Scheduling Latencies



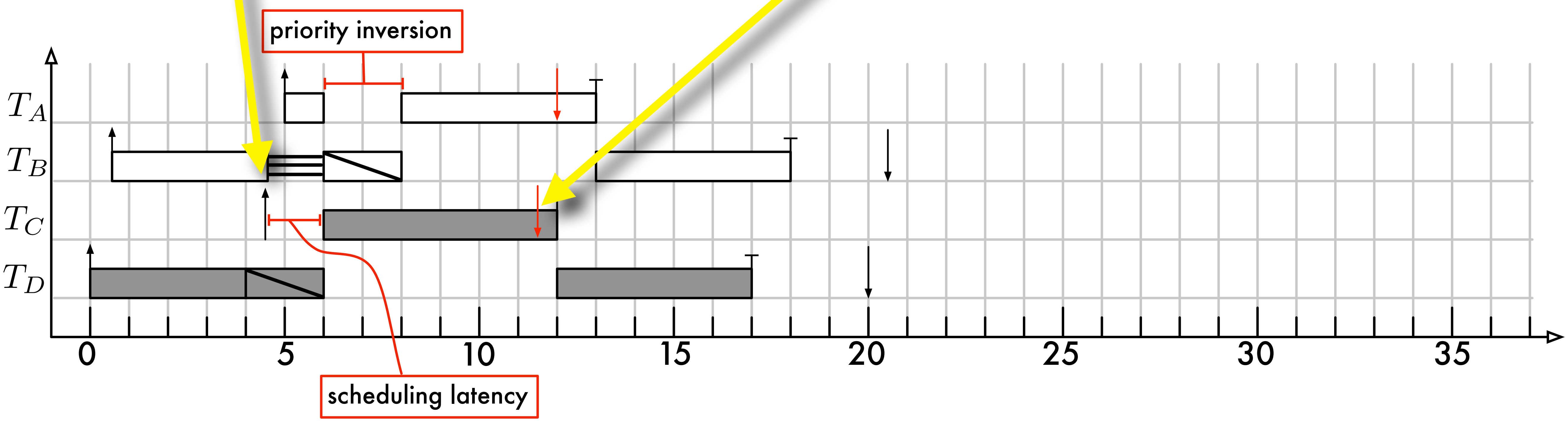


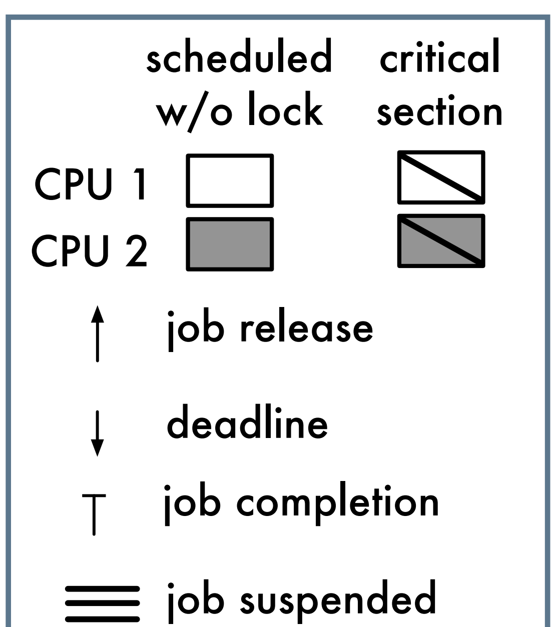
Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Due to priority boosting, T_C cannot preempt T_D immediately.

This results in increased scheduling latency and T_C misses its deadline!

Increases Scheduling Latencies





Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Summary: Priority Boosting

- **Simple solution** to the lock-holder preemption problem.
- In the worst case, no different from simply turning off interrupts: **increased latency**.
- Unconditional boosting of priorities **does not play nice with the FUTEX API.**

Part 3

A (simple) "**tweak**" to Linux's **existing** priority inheritance solution **restores predictability**.

The Desired Property

The **blocking task** **is scheduled**
(on some processor).

A **blocked task** ***should*** be scheduled but ***is not***.

=

A **blocked task** would be the **highest-priority task**
on its assigned processor(s) if it were runnable.

A Simple Solution:
Migratory Priority Inheritance

Priority Inheritance

Blocking tasks are eligible to execute
with the priority of blocked tasks.

A Simple Solution:
Migratory Priority Inheritance

Priority Inheritance

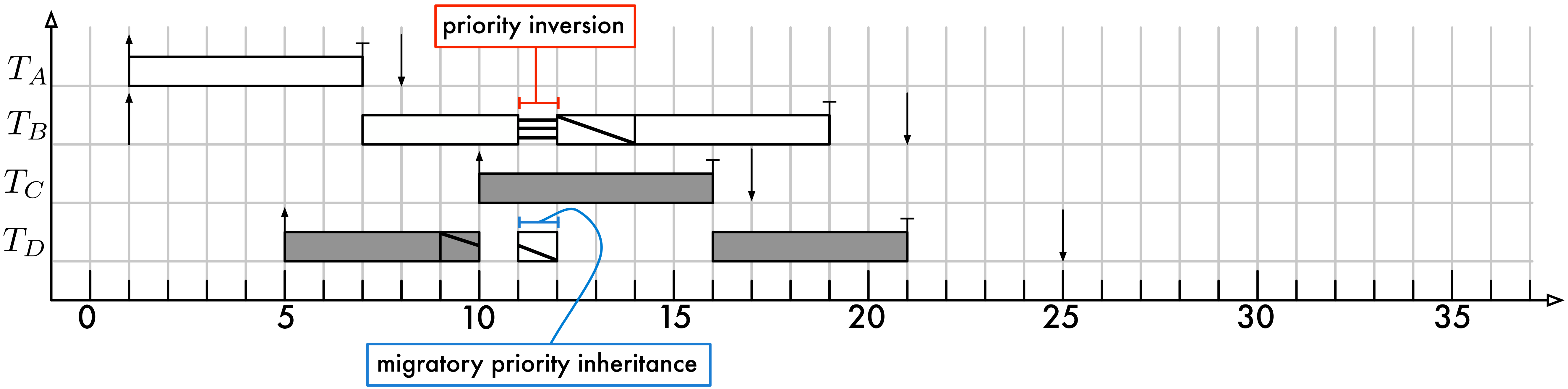
Blocking tasks are eligible to execute
with the priority of blocked tasks.

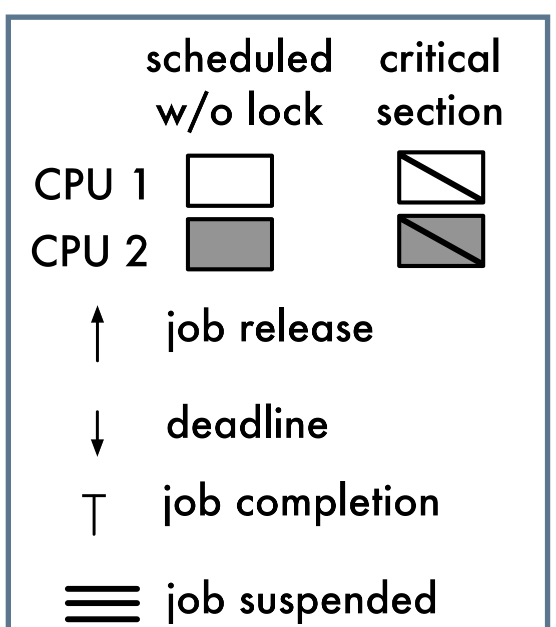
+

Processor Affinity Mask Inheritance

Blocking tasks are eligible to execute
on the processor(s) of blocked tasks.

Migratory Priority Inheritance Bounds Priority Inversions



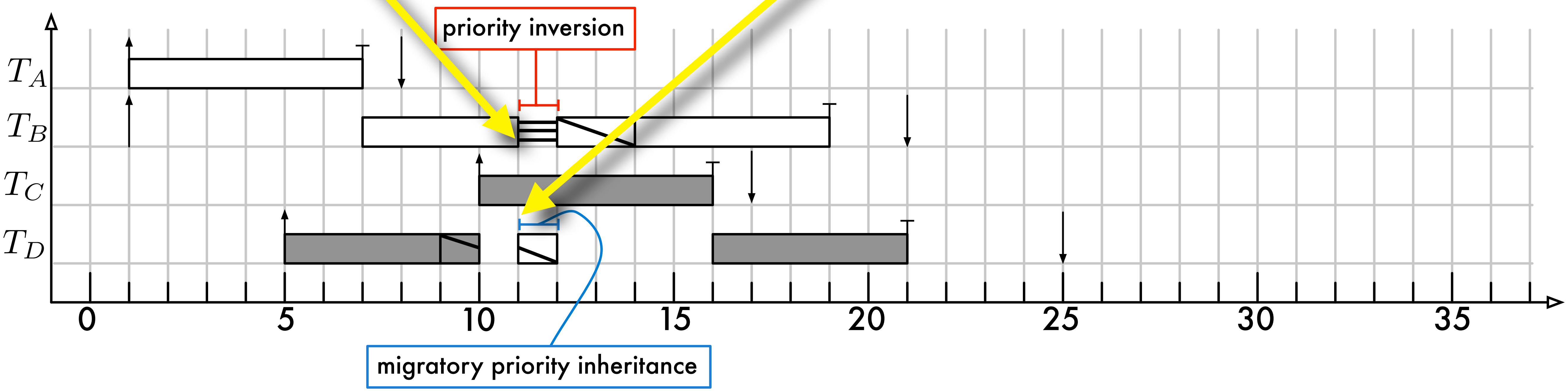


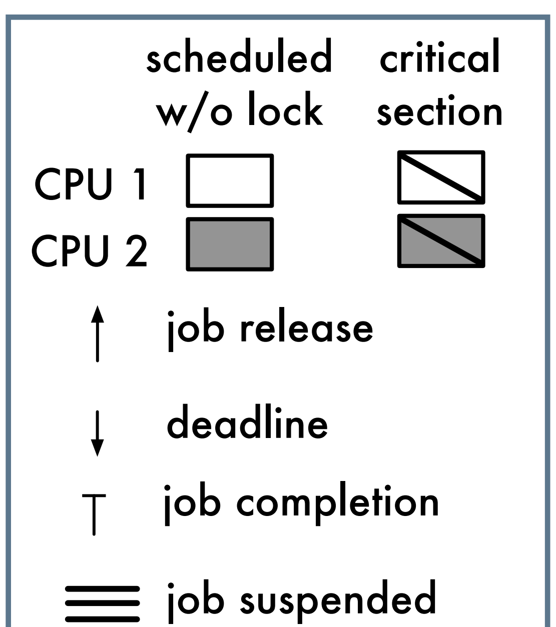
Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Due to processor affinity mask inheritance,

T_D migrates to T_B's CPU when T_B blocks on T_D.

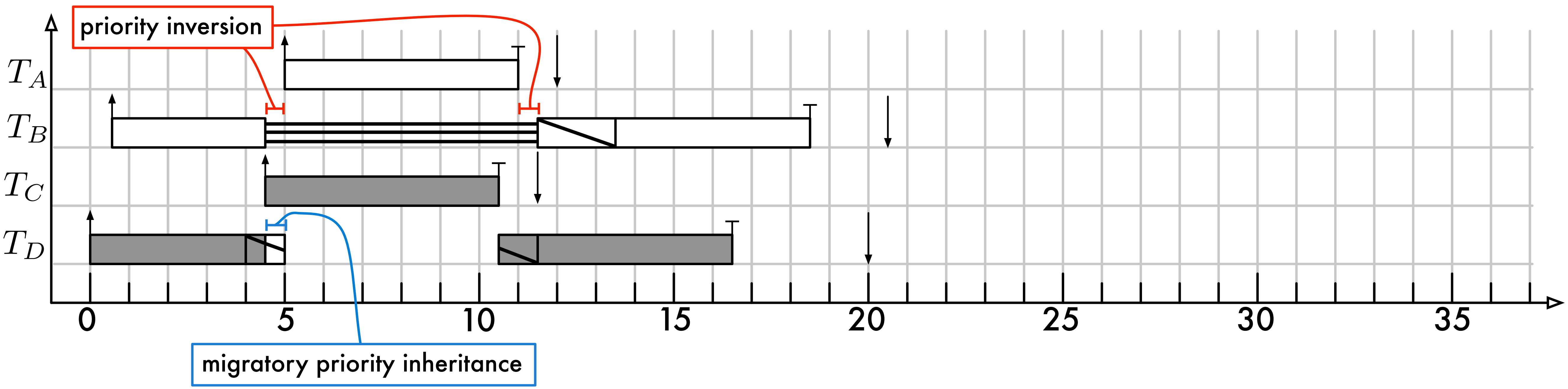
ance
Bounds Priority Inversions

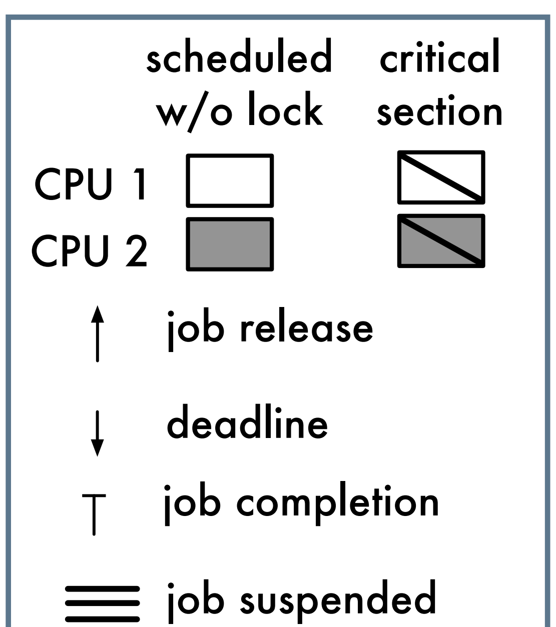




Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Migratory Priority Inheritance Does Not Increase Latencies



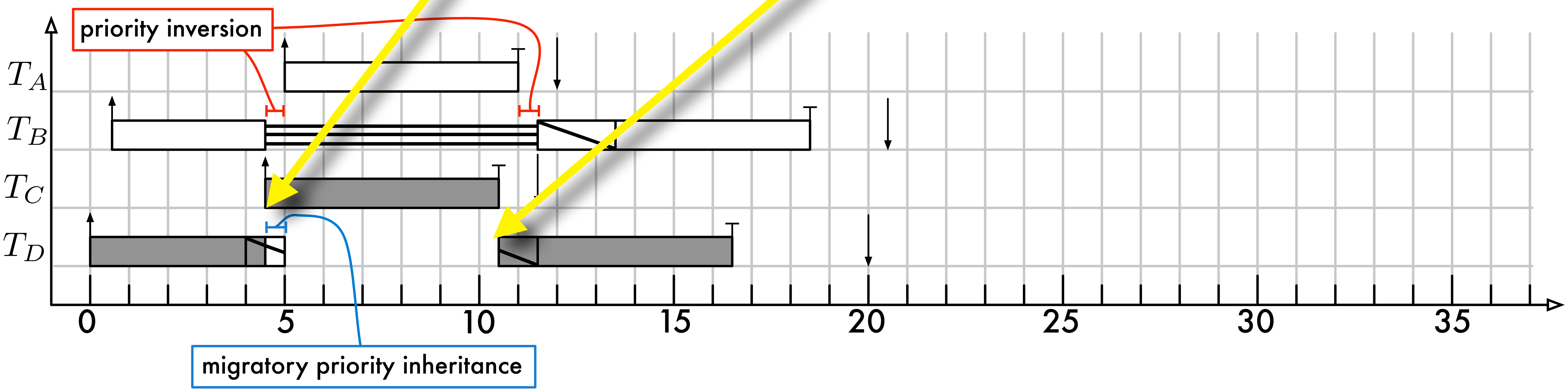


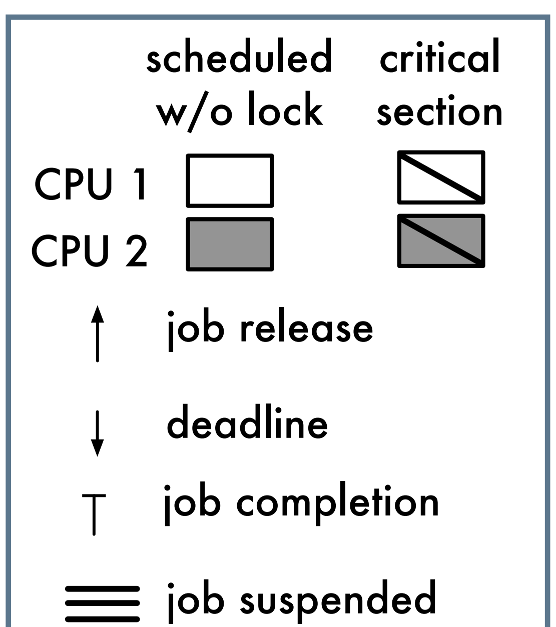
Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Under <u>migratory priority inheritance</u>,

T_C can preempt T_D immediately and T_D migrates to T_B's CPU.

When T_B's CPU becomes unavailable T_D migrates back.





Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2

Properties of Migratory Priority Inheritance

1. **Bounds priority inversions** in all cases, on multiprocessors, with arbitrary affinity masks.
2. **Reduces to classic priority inheritance** on uniprocessors and under global scheduling.
3. Does **not** increase worst-case **scheduling latency**.
4. Takes only effect in case of contention: fully **FUTEX compatible**.
5. **POSIX compliant** and **fully transparent** to developers!

Implementation

implementation complexity vs. analysis accuracy

Two Variants of Migratory Priority Inheritance

Simplified

- Easier to implement, likely **lower overheads**.
- Can **reuse** large parts of Linux's implementation.
- Occurrence of priority inversion is **not minimal.**

Full

- Requires **careful tracking of all inherited priorities** and affinity masks.
- More complicated **push/pull migration logic**.
- **Fewer priority inversions** (similar to uniprocessors).

Simplified Migratory Priority Inheritance

effective priority (priority used on ***all*** processors in mask)	=	**maximum inherited priority** (or its own priority if not blocking higher-priority tasks)

priority and affinity mask are tracked independently

effective processor affinity mask (where is a lock-holder eligible to execute?)	=	**union of all inherited masks** (and a task's own processor affinity masks)

Full Migratory Priority Inheritance

eligibility tuple = (**priority**, **processor affinity mask**)

effective priority on processor *P*	=	**maximum priority** among the **(inherited) tuples** with ***P*** **in the processor affinity mask**

Tracking of processor-specific priorities is difficult in Linux.

Migratory Priority Inheritance

Classic priority inheritance is **ineffective** if tasks have **non-global** processor affinity masks.

Priority boosting is not a good fit for Linux since it **increases worst-case latencies.**

Adding **processor affinity mask inheritance** to Linux's existing priority inheritance implementation **restores predictability**.

Prior Work

Using migrations to "help" preempted lock holders:

- "Local Helping" in **Fiasco/L4**
  (Hohmuth & Peter, 2001)
- Multiprocessor BandWidth Inheritance (**MBWI**)
  (Faggioli et al., 2010)

This principle keeps popping up… time to adopt it!

Thanks!

MPI-SWS is hiring **PhD students**, **post-docs**, and **tenure-track faculty**.

But the user specified the processor affinity mask!

- True, but **the user also specified the priority.**
- To obtain bounded priority inversions, scheduling parameters have to be overridden; this is **no different from classic priority inheritance**.
- At least all **kernel code should tolerate possible migrations** (or call `preempt_disable()`).
- Userspace can get a new policy to opt in: **`PRIO_INHERIT_MIGRATORY`**

This will create huge cache-related migration overheads!

- Not really:

  1. The migrating task **was preempted anyway**.
  2. Only the **working set of the critical section** is relevant, which is likely quite small.

Classic priority inheritance is sufficient if you assign priorities and processors in the right way!

- No, that's not always the case.
- The constructed example task set is **feasible on two processors** (it can be scheduled with migratory priority inheritance without missing deadlines).
- **There does not exist a priority assignment** that ensures that all deadlines will always be met **under classic priority inheritance**. (Try it.)
- Similarly, the task set cannot be scheduled under priority boosting.

Task	WCET	Period	Deadline	Critical Section	Priority	Processor
T_A	6	20	7	—	99	1
T_B	11	20	20	2	97	1
T_C	6	20	7	—	98	2
T_D	11	20	20	2	96	2