Resource Sharing

CPEN 432 Real-Time System Design

Arpan Gujarati
University of British Columbia

Terminology

- Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ consists of n periodic tasks
- Each task is characterized by a period T_i and worst-case completion time C_i
- The tasks cooperate through m shared resources $R_1, R_2, \ldots, R_m$
- Each resource R_k is guarded by a distinct **binary semaphore** S_k
  - All critical sections using R_k start and end with operations $wait(S_k)$ and $signal(S_k)$
- Each task is assigned a fixed **base priority** P_i (e.g., using RM)
  - Assumption: priorities are unique and $P_1 > P_2 > \ldots > P_n$
- Each task also has an **effective priority** p_i $(\geq P_i)$
  - It is initially set to P_i and can be **dynamically updated**
- B_i denotes the maximum blocking time task τ_i can experience
  - B_i goes into the fixed-priority response-time analysis (recall from previous lectures)
- $z_{i,k}$ denotes any arbitrary critical section of τ_i guarded by semaphore S_k
  - $Z_{i,k}$ denotes the longest among all these critical sections
  - $\delta_{i,k}$ denotes the length of this longest critical section $Z_{i,k}$

The Priority Ceiling Protocol (PCP)

PCP Key Concepts

- **Priority ceilings**
  - Each semaphore S_k is **statically** assigned a priority ceiling $C_{static}(S_k)$
    - $C_{static}(S_k) =$ priority of the highest-priority task that **ever** accesses S_k
- **Current system ceiling**
  - At any time t, a global system ceiling $C_{global}(t)$ is dynamically computed
    - $C_{global}(t)$ = highest priority ceiling among all semaphores locked at time t OR
      (if no semaphores are locked) sentinel value P_0 that is **smaller** than all task priorities
- **Protocol**
  - Task τ_i can acquire semaphore S_k at time t only if
    - Its effective priority $p_i > C_{global}(t)$ OR $p_i = C_{global}(t)$ and τ_i "owns" the ceiling resource
    - OTHERWISE, it transmits its priority to the task τ_j that holds semaphore S_k

Analytically, PCP is better than PIP

- Like PIP ...

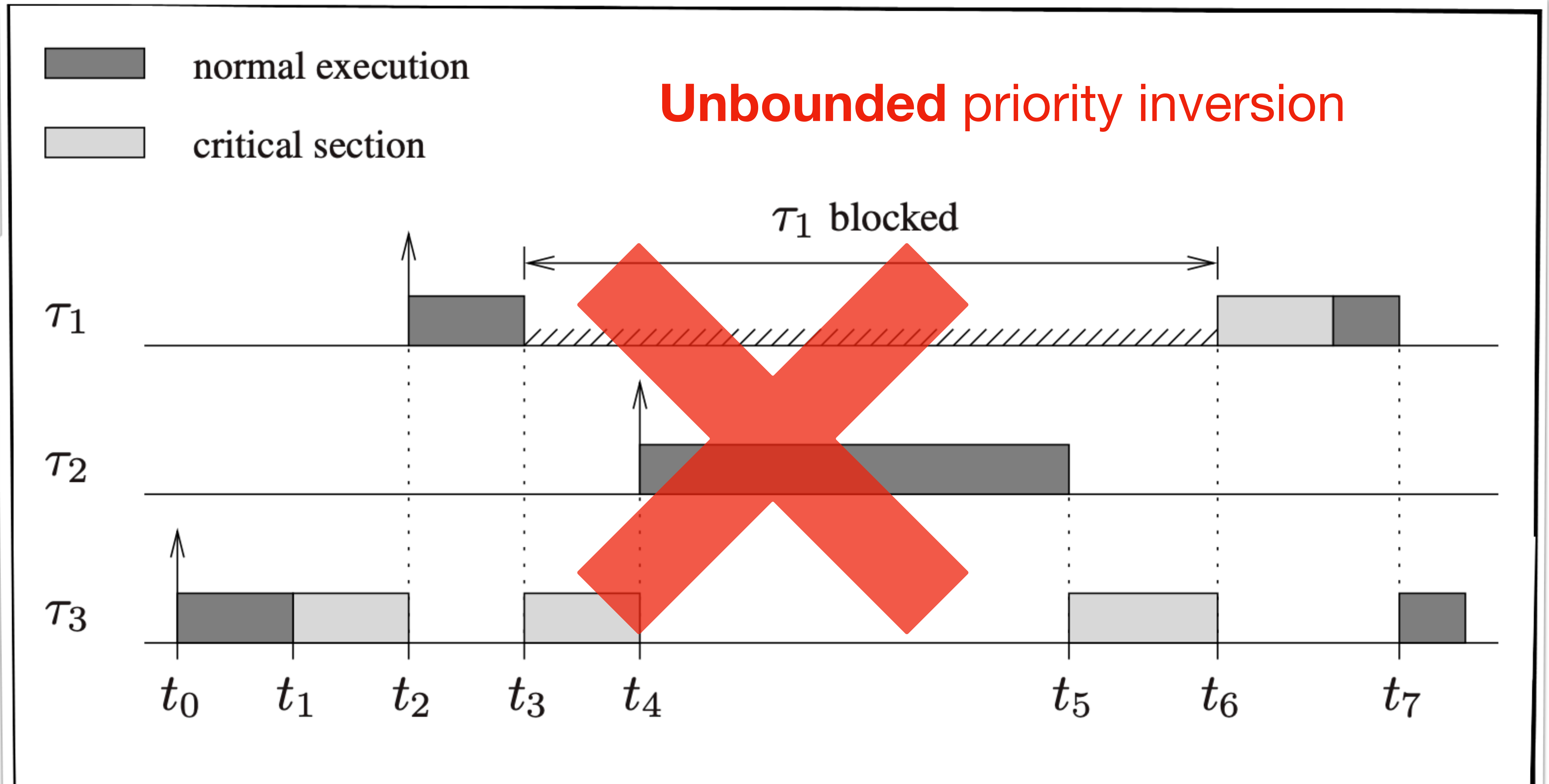


Figure 7.4 An example of priority inversion.

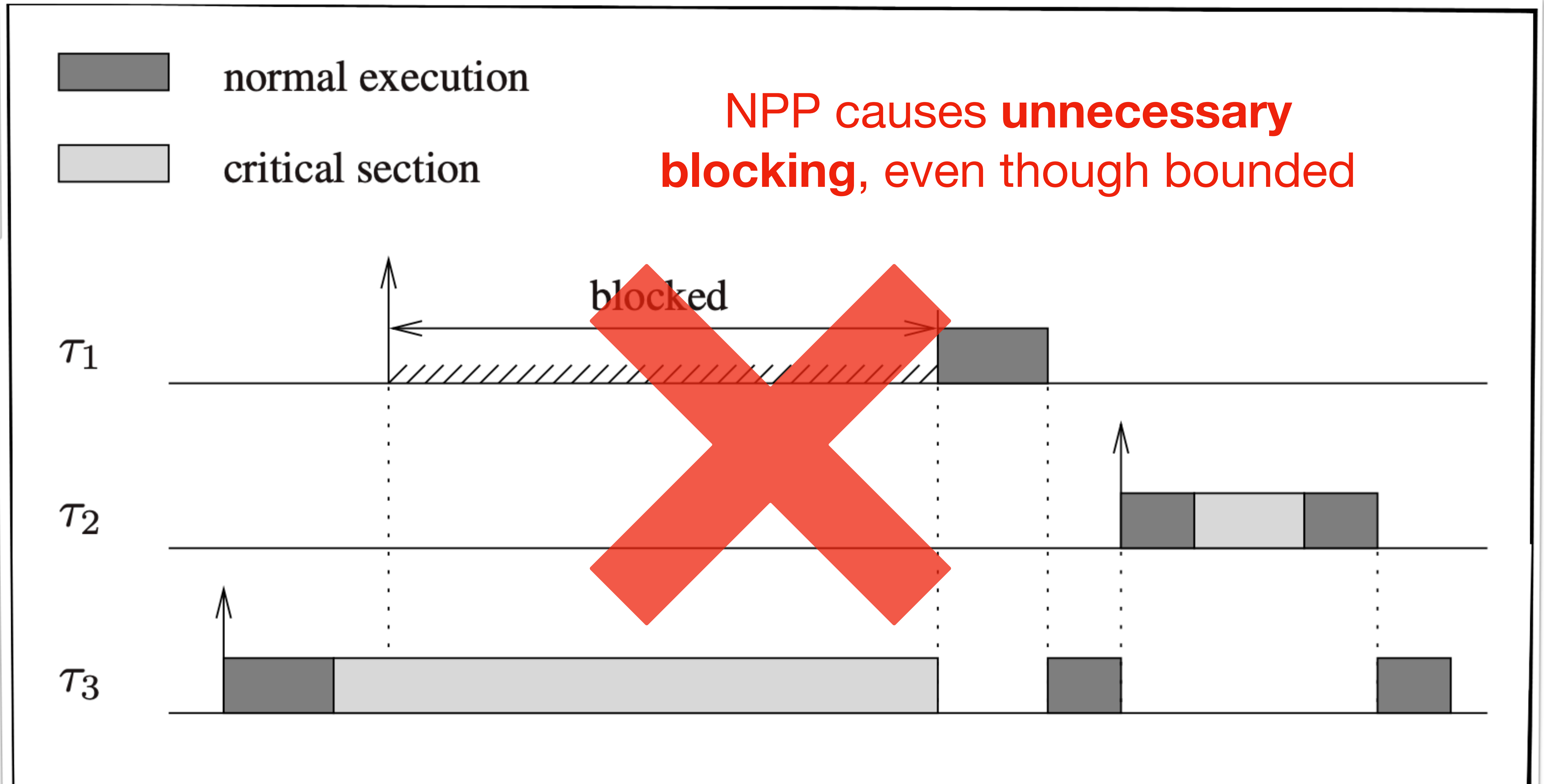


Figure 7.6 Example in which NPP causes unnecessary blocking on τ_1.

Analytically, PCP is better than PIP

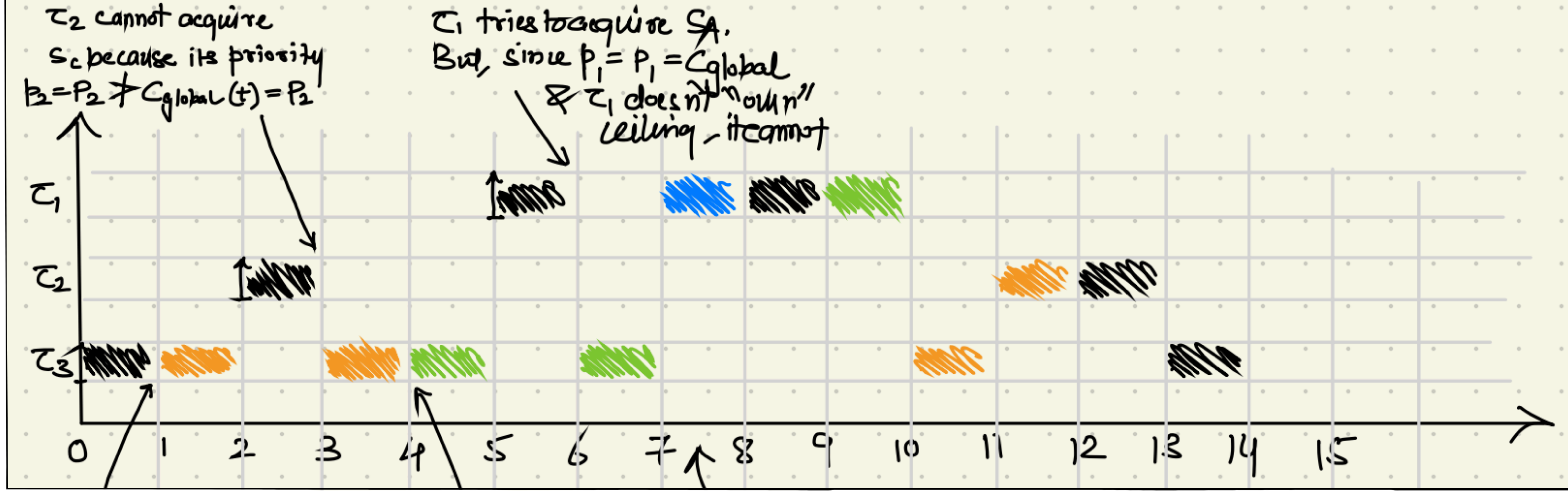
- In addition, unlike PIP
  - PCP prevents transitive blocking
  - PCP prevents deadlocks
  - A task τ_i can be blocked for **at most** the duration of **one** critical section

PCP Example

Task	Priority	Execution Times		Arrival time
τ_1	P_1	[black 1] [A 1] [black 1] [B 1]	Sequential CS	5
τ_2	P_2	[black 1] [C 1] [black 1]		2
τ_3	P_3	[black 1] [C (B) 4] [black 1]	Nested CS	0

$P_1 > P_2 > P_3 > P_4$

$C(S_A) = 1$, $C(S_B) = 1$, $C(S_C) = 2$



What’s Wrong with Context Switches?

- Each contended critical section causes *two additional context switches*.
  ➞ Regular preemption: LO-HI-LO
  ➞ With critical section: LO-HI-LO-HI-LO

Stack Resource Policy (SRP)

The Stack Resource Policy (SRP)

Observation: if a preempting job requires a locked resource, then a LO-HI-LO-HI-LO context switch sequence becomes **inevitable** *only if the preempting job is allowed to start executing*.

Solution: do not allow jobs to commence execution until all (possibly) required resources are available.
→ No more LO-HI-LO-HI-LO context switch sequences…

SRP Definition[Ba91]

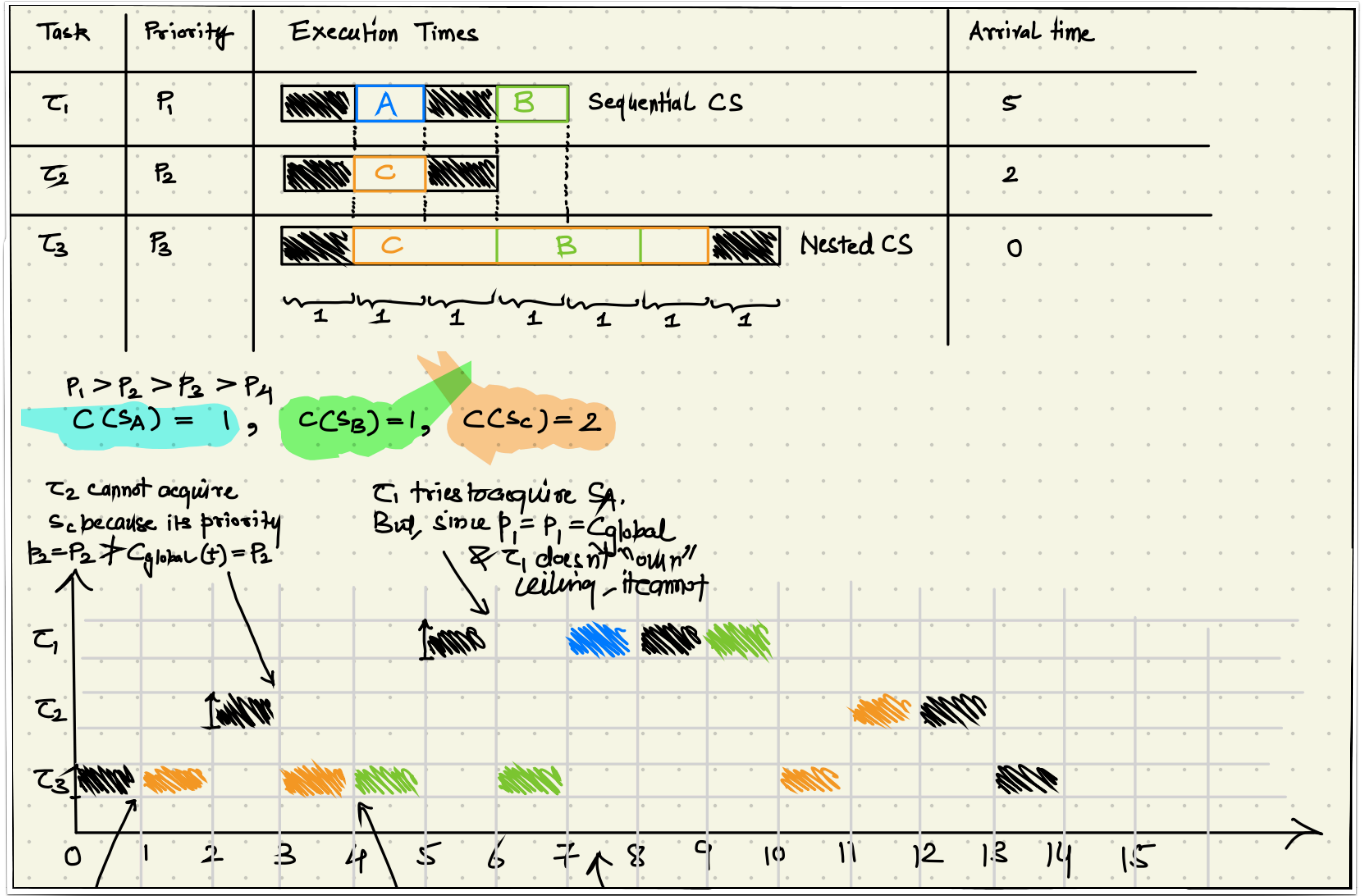
1. Define priority ceilings and system ceilings as under the PCP.
2. When a job is released, it may not commence execution until its (base) priority exceeds the system ceiling (or *preemption threshold*).
3. Whenever a job requires a resource, it gains access immediately.

[Ba91] T. Baker (1991). Stack-based scheduling for realtime processes. Real-Time Systems, 3(1):67–99.

~~PCP~~ SRP Key Concepts

- **Priority ceilings**
  - Each semaphore S_k is **statically** assigned a priority ceiling $C_{static}(S_k)$
    - $C_{static}(S_k) =$ priority of the highest-priority task that **ever** accesses S_k
- **Current system ceiling**
  - At any time t, a global system ceiling $C_{global}(t)$ is dynamically computed
    - $C_{global}(t)$ = highest priority ceiling among all semaphores locked at time t OR (if no semaphores are locked) sentinel value P_0 that is **smaller** than all task priorities
- **Protocol**
  - Task τ_i can acquire semaphore S_k ~~**at time t only if**~~ **immediately**
  - **Task τ_i may commence its execution only if**
    - Its effective priority $p_i > C_{global}(t)$ OR $p_i = C_{global}(t)$ and τ_i "owns" the ceiling resource
    - OTHERWISE, it transmits its priority to the task τ_j that holds semaphore S_k

PCP Example



Task
Priority
Execution Times
Arrival time
τ1
P1
A
B
Sequential CS
5
τ2
P2
C
2
τ3
P3
C
B
Nested CS
0
1 1 1 1 1 1 1
P1 > P2 > P3 > P4
C(SA) = 1,
C(SB)=1,
C(Sc)=2
τ2 cannot acquire
Sc because its priority
P2=P2 ≯ Cglobal(t)=P2
τ1 tries to acquire SA.
But, since P1 = P1 = Cglobal
& τ1 doesn't "own" ceiling - it cannot
τ1
τ2
τ3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

SRP Blocking Analysis

The bound on *worst-case* pi-blocking under the SRP is *identical* to the PCP's bound.

$$B_i = max\{Z_{j,k} \mid P_j < P_i \;\; and \;\; C_{global}(S_k) \geq P_i\}$$

- The *actual* pi-blocking differs under the SRP and the PCP.
  → The SRP moves blocking to an earlier point in time.
  → On average, the PCP may yield slightly less blocking. (Why?)

Sharing Runtime Stacks

Example: $prio(\tau_4) > prio(\tau_3) = prio(\tau_2) > prio(\tau_1)$

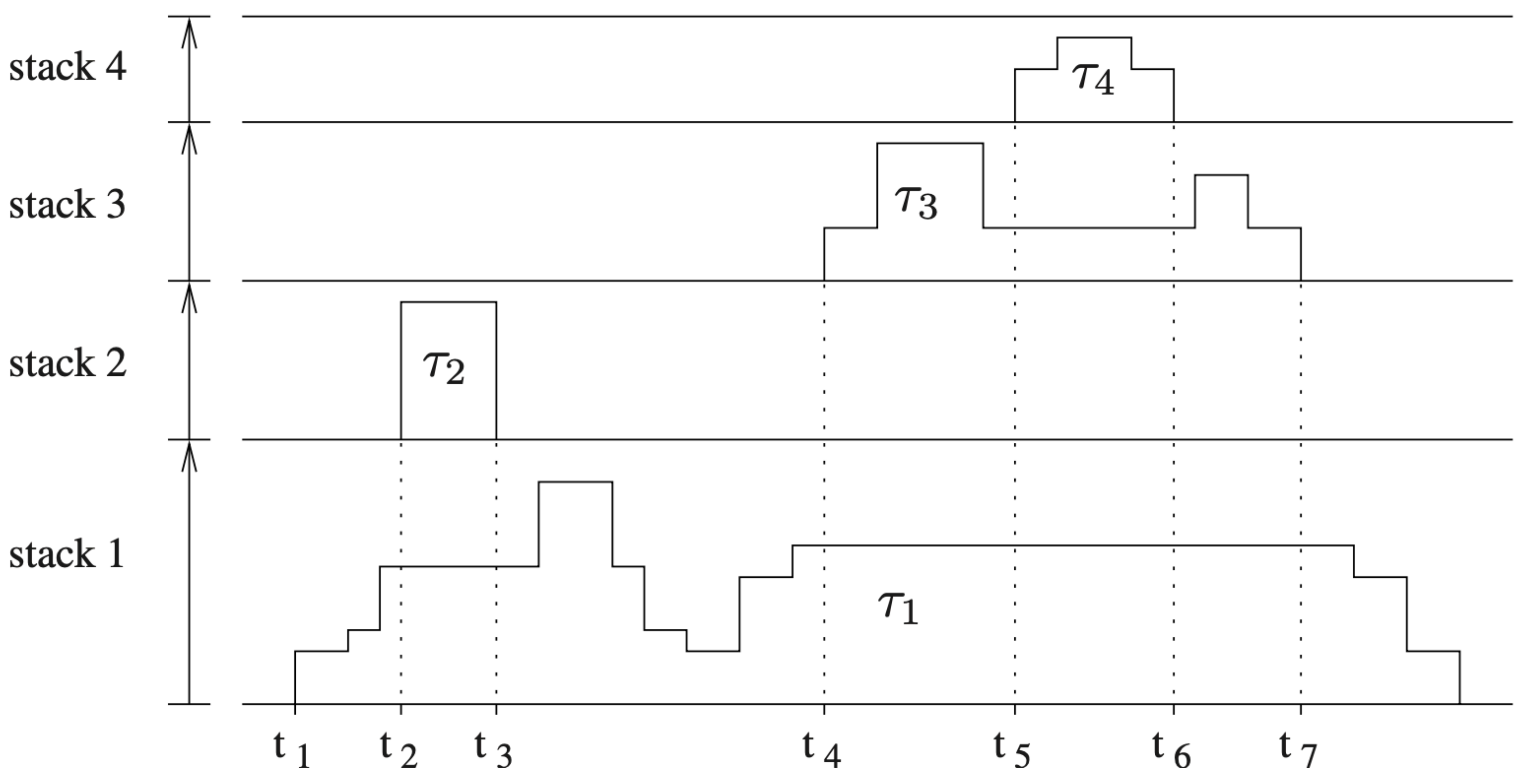


Figure 7.21 Possible evolution with one stack per task.

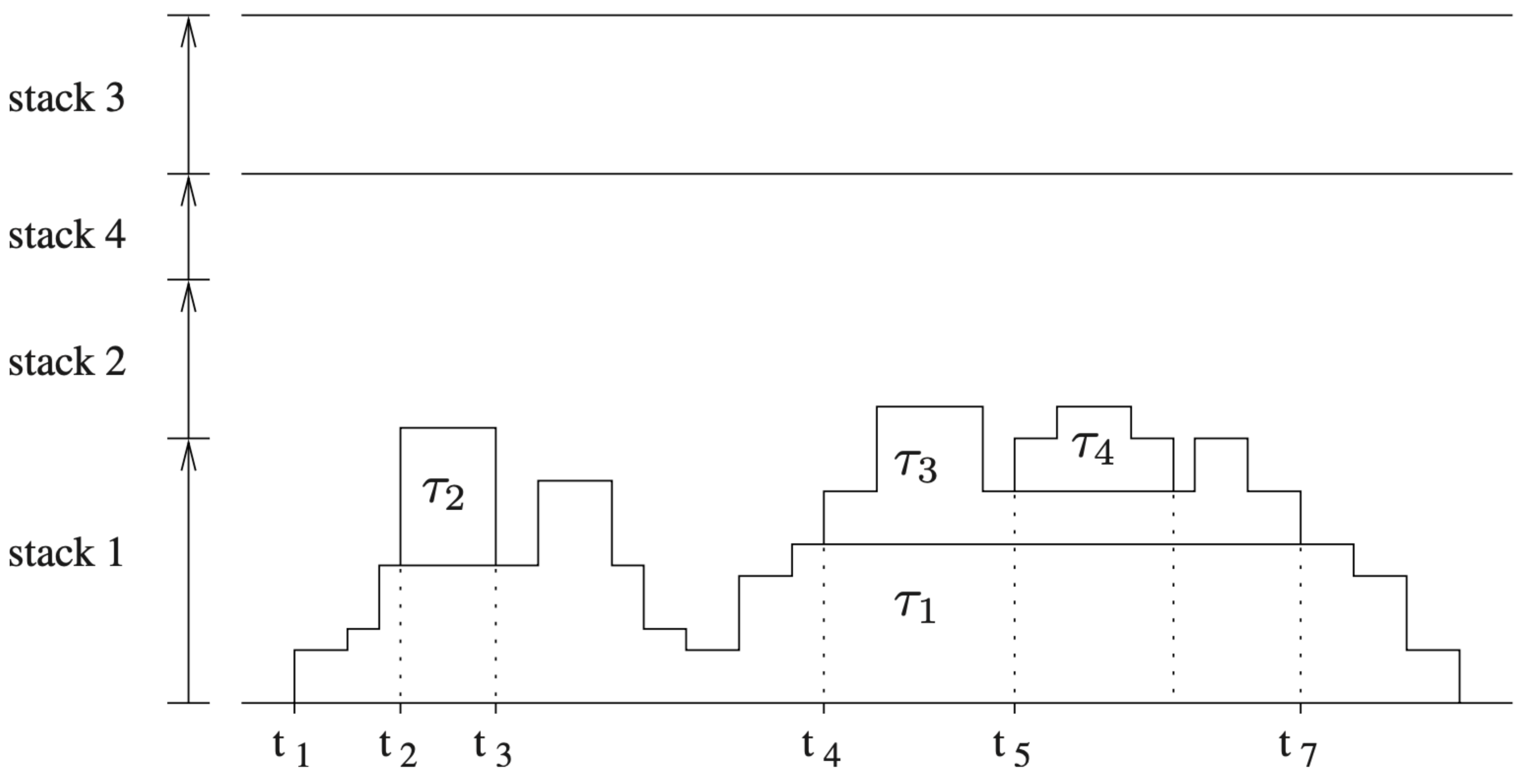


Figure 7.22 Possible evolution with a single stack for all tasks.

SRP with Preemption Levels, Multi-Unit Resources

- Each task τ_i is assigned a priority P_i
  - P_i can be fixed (e.g., RM, DM) or dynamic (e.g., EDF), and is unaffected by the locking protocol (no more inheritance)
- Each task τ_i is assigned a **static** preemption level π_i
  - τ_a can preempt τ_b only if $\pi_a > \pi_b$
- For SRP, we want that
  - *"If τ_a arrives after τ_b and τ_a has a higher priority than τ_b, then τ_a must have a higher preemption level than τ_b"*
  - Under EDF scheduling, $\pi_i > \pi_j \iff D_i < D_j$
- Each resource R_k is allowed to have N_k units that can be concurrently accessed
  - $wait(S_k, r)$ blocks until r units of R_k are available, and the following $signal(S_k)$ releases all locked units of R_k
  - $n_k(t)$ denotes the number of currently available units of R_k (i.e., $N_k - n_k(t)$ units are locked)
  - $\mu_i(R_k)$ denotes the maximum number of units of R_k that can be simultaneously requested by τ_i
- Dynamic resource ceiling of R_k at any time: $C_{R_k}(t) = \max\{\pi_i \mid \mu_i(R_k) > n_k(t)\}$ or $C_{R_k}(t) = 0$ (if $n_k(t) = N_k$)
- Dynamic system ceiling $\Pi_s(t) = \max_k\{C_{R_k}(t)\}$
- SRP preemption test: τ_i is the highest priority ready task and $\pi_i > \Pi_s$

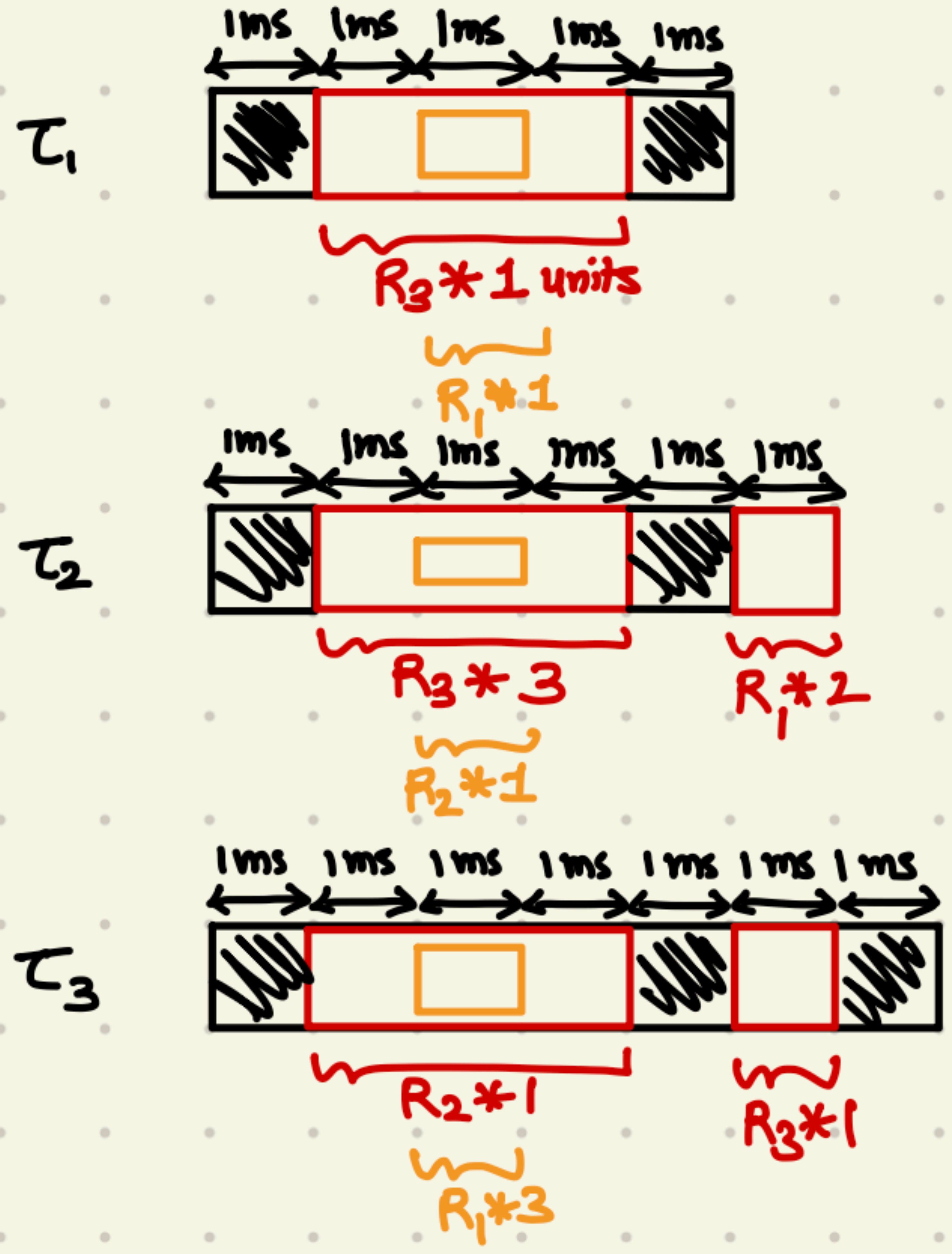
Example

	D_i	π_i	$\mu_i(R_1)$	$\mu_i(R_2)$	$\mu_i(R_3)$	a_i
τ_1	12	3	1	0	1	2.5
τ_2	15	2	2	1	3	1.5
τ_3	19	1	3	1	1	0

$N_1 = 3$

$N_2 = 1$

$N_3 = 3$



Properties of SRP

Lemma 7.9 *If the preemption level of a task τ is greater than the current ceiling of a resource R, then there are sufficient units of R available to*

1. *meet the maximum requirement of τ and*
2. *meet the maximum requirement of every task that can preempt τ.*

Theorem 7.5 (Baker) *If no task τ is permitted to start until $\pi(\tau) > \Pi_s$, then no task can be blocked after it starts.*

Theorem 7.6 (Baker) *Under the Stack Resource Policy, a task τ_i can be blocked for at most the duration of one critical section.*

Theorem 7.7 (Baker) *The Stack Resource Policy prevents deadlocks.*

Summary

	priority	Num. of blocking	pessimism	blocking instant	transpa-rency	deadlock preven-tion	implem-entation
NPP	any	1	high	on arrival	YES	YES	easy
HLP	fixed	1	medium	on arrival	NO	YES	easy
PIP	fixed	α_i	low	on access	YES	NO	hard
PCP	fixed	1	medium	on access	NO	YES	medium
SRP	any	1	medium	on arrival	NO	YES	easy

Table 7.5 Evaluation summary of resource access protocols.