Real-time Systems

Chapter 9:
Resource Access Protocol

Ngo Lam Trung

Dept. of Computer Engineering

Contents

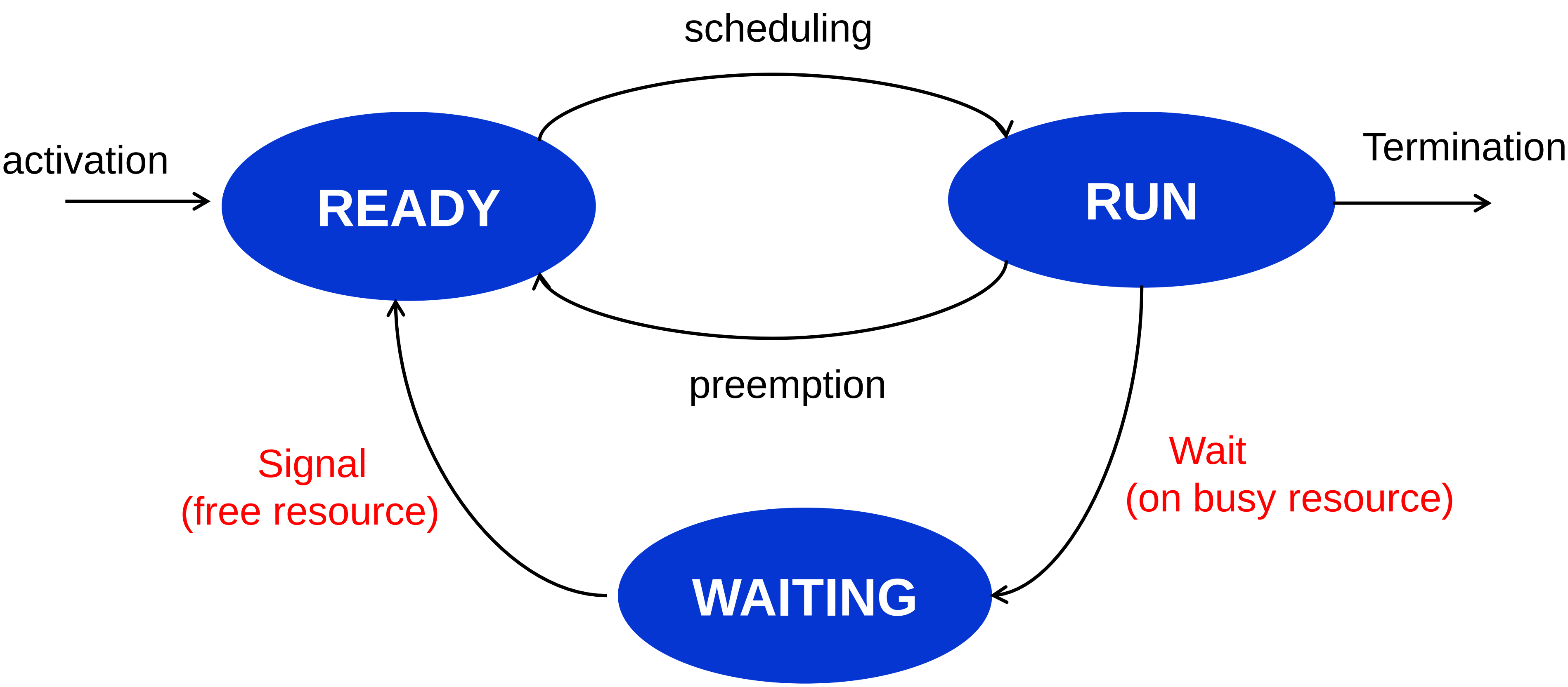
- Introduction
- The priority inversion phenomenon
- Solutions for priority inversion

Resource constraint

- Resource
  - Any software structure that can be used by the process to advance its execution
  - Ex: data structure, variables, main memory area, a file, a piece of program, a set of registers of a peripheral device
- Many shared resources do not allow simultaneous access

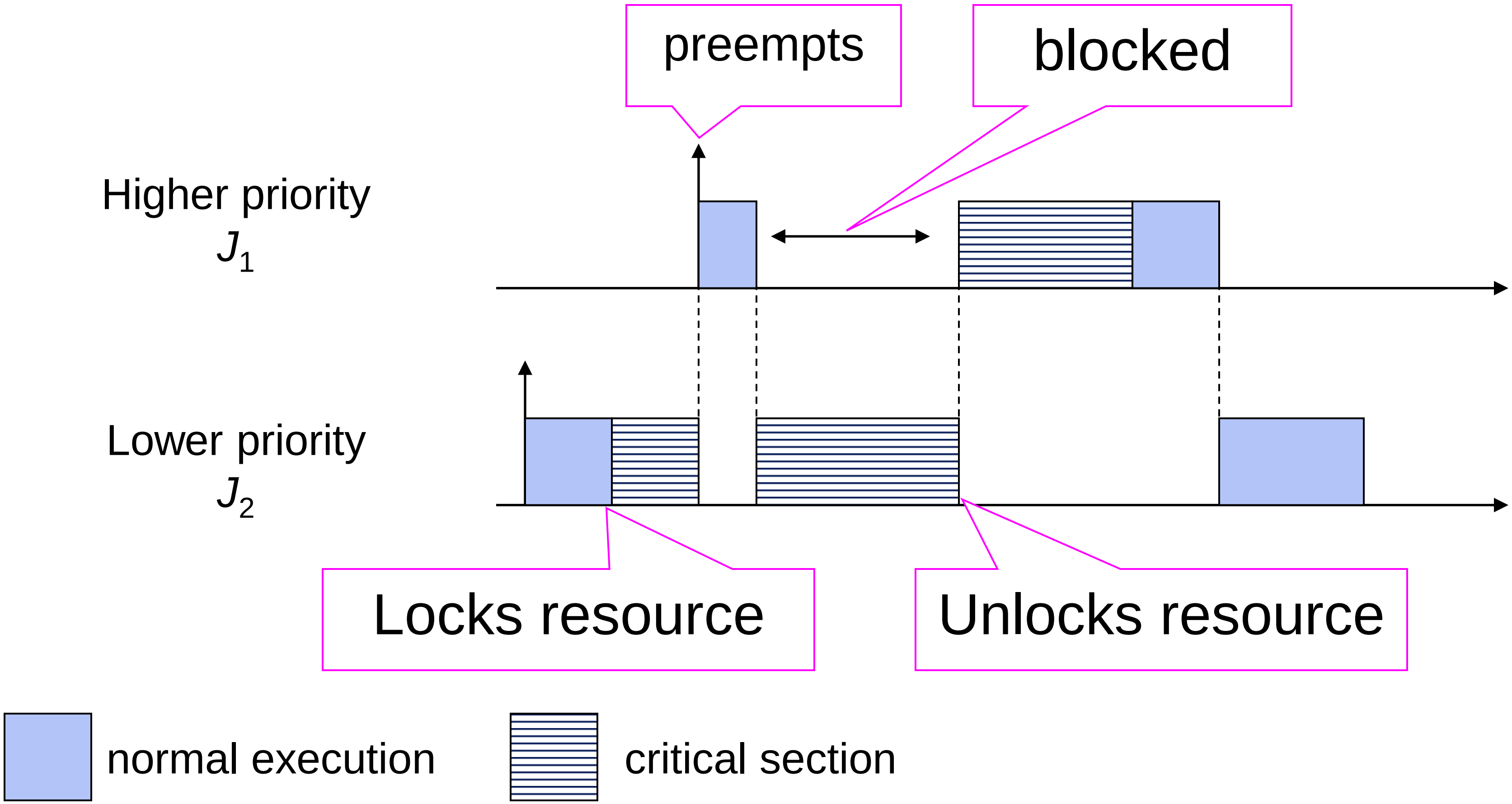
  → require mutual exclusion
- Critical section
  - A piece of code under mutual exclusion constraints
  - Tasks entering critical section have to wait until no other task is holding the resource

Waiting state caused by resource constraint



Example of blocking on exclusive resource

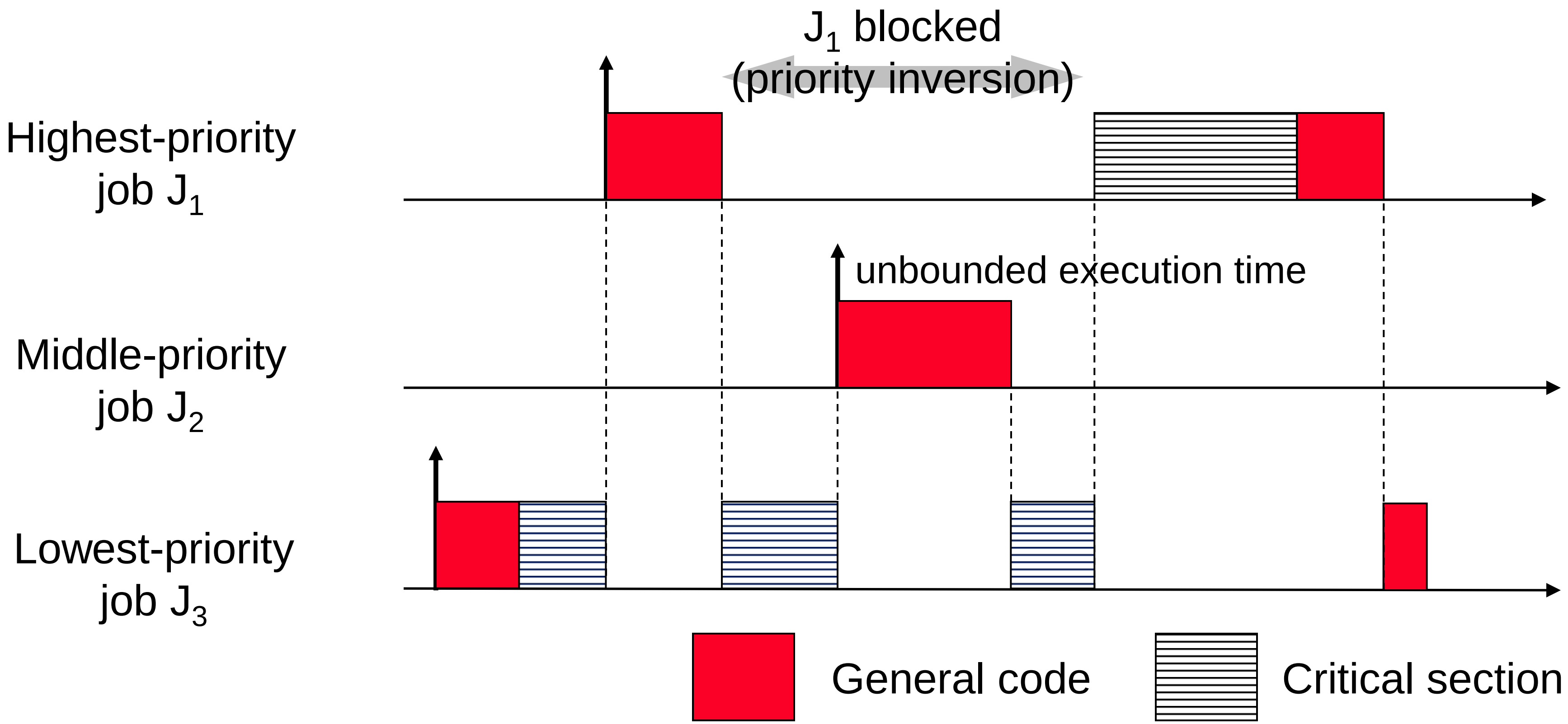
- Scheduling with preemption



Problem: the task with higher priority has to wait for the task with lower priority

The priority inversion phenomenon

- J3 enters critical section first
- J1 is blocked, has to wait until J3 signal the resource
- J2 preempts J3 ➔ J1 has to wait for J2



Problems

- The task with higher priority has to wait for the task with lower priority
- Blocking time is unbounded → the system is not predictable.
- Example of priority inversion: Mars Pathfinder 1997
  - CPU: RAD6000 20MHz ($200K-$300K)
  - OS: VxWork
  - Experienced CPU reset upon touching down on Mars, debugging on Earth detected priority inversion, fixed by new firmware upload.

Problems

- Solutions
  - Non-preemptive Protocol
  - Highest Locker Priority Protocol
  - Priority Inheritance Protocol
  - Priority Ceiling Protocol
  - Stack Resource Policy

Terminology & assumptions(1)

- Periodic task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$
  - $\tau_i = (C_i, T_i)$
  - Relative deadline $D_i = T_i$
- Resources $R_1, \ldots, R_m$
  - Each R_k is guarded by semaphore S_k

- J_i: a job of τ_i
- P_i: nominal priority of τ_i
- $p_i \geq P_i$: active priority of τ_i (initially set to P_i)
- $z_{i,j}$: j-th critical section of J_i
- $d_{i,j}$: duration of $z_{i,j}$
- $S_{i,j}$: the semaphore guarding $z_{i,j}$
- $R_{i,j}$: the resource used in $z_{i,j}$
- Notation $z_{i,j} \subset z_{i,k}$ means $z_{i,j}$ is entirely contained in $z_{i,k}$.

Terminology & assumptions (2)

- Assumptions
  - $J_1,\ldots,J_n$ are listed in decreasing order of P_i
  - Jobs don't suspend themselves.
  - The critical sections used by any task are properly nested.

$$z_{i,j} \subset z_{i,k} \text{ or } z_{i,k} \subset z_{i,j} \text{ or } z_{i,j} \cap z_{i,k} = 0$$

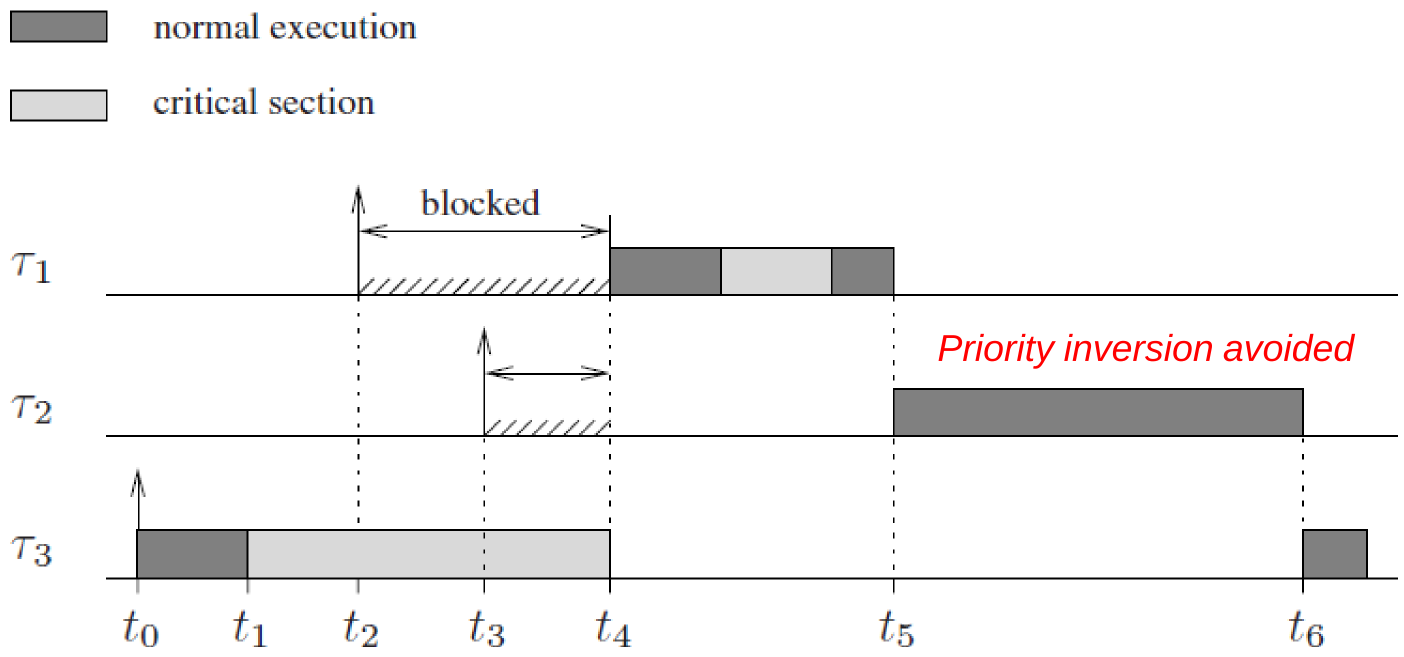
  - Critical sections are guarded by binary semaphores.

The simplest: Non-preemptive Protocol

- Block all other tasks whenever a task enters a critical section

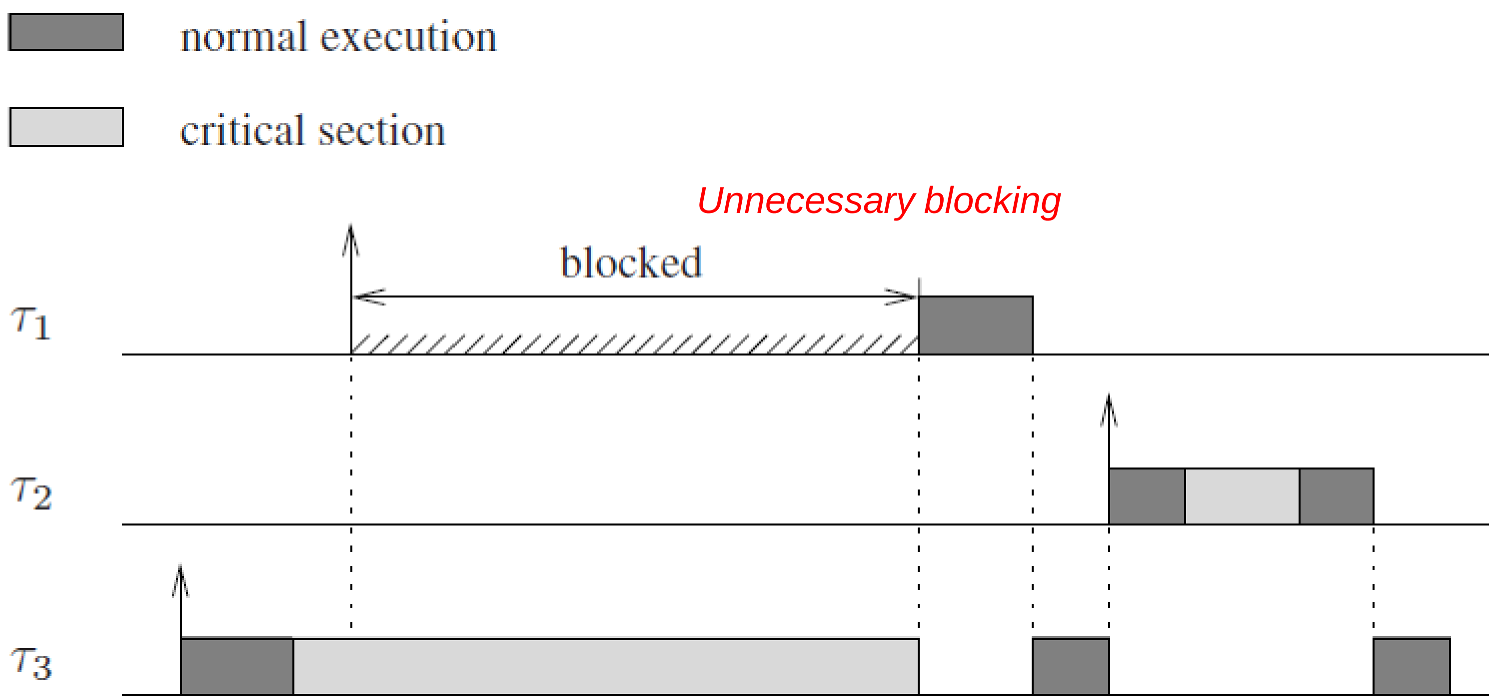
- The dynamic priority of the running task is raised to the highest level

$$p_i(R_k) = \max_h \{P_h\}$$



The simplest: Non-preemptive Protocol (NPP)

- Pros: simple
- Cons: unnecessary blocking



Blocking time of Non-preemptive Protocol

- Given the task T_i, the set of critical sections that can block T_i

$$\gamma_i = \{Z_{j,k} \mid P_j < Pi, k = 1, \ldots, m\}$$

- The maximum blocking time is

$$B_i = \max\{d_{j,k} - 1 \mid Z_{j,k} \in \gamma i\}.$$

➔ Duration of the longest critical section that can block T_i

Highest Locker Priority Protocol (HLP)

- Improves NPP: raising the priority of the task entering a critical section to the highest priority among the tasks sharing that resource.

- When a task enters resource R_k, its dynamic priority is raised to

$$p_i(R_k) = \max_h \{P_h \mid \tau_h \text{ uses } R_k\}$$

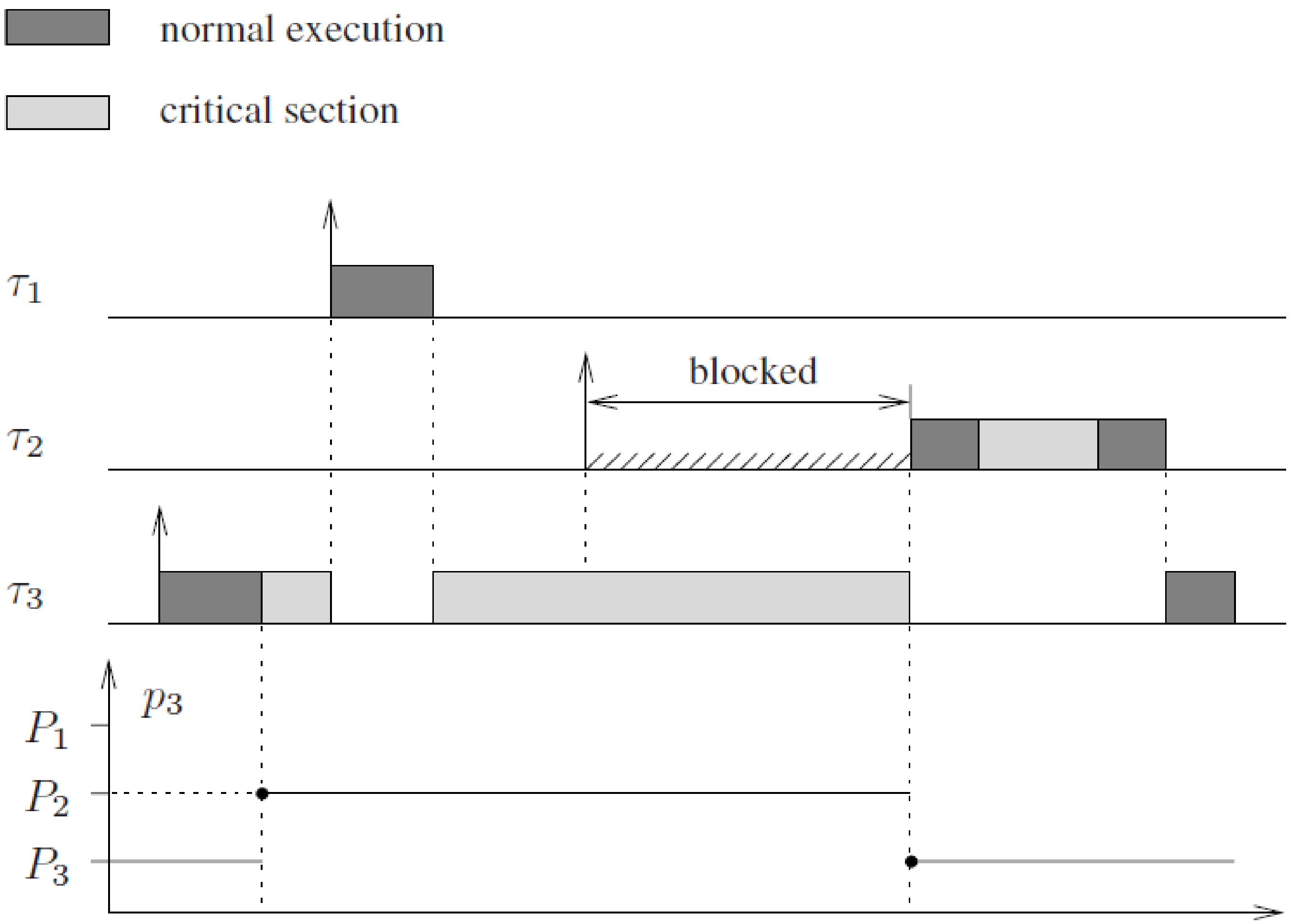
- When the task exits the resource, its dynamic priority is reset to the nominal value P_i

- Priority ceiling can be computed offline

$$C(R_k) \stackrel{\text{def}}{=} \max_h \{P_h \mid \tau_h \text{ uses } R_k\}$$

Highest Locker Priority Protocol

- Example



HLP Blocking time

- The set of critical instants that can block task T_i

$$\gamma_i = \{Z_{j,k} \mid (P_j < P_i) \text{ and } C(R_k) \geq P_i\}$$

- Hence, maximum blocking time is

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

- Problem: what if critical section is access in only one branch of a conditional statement?

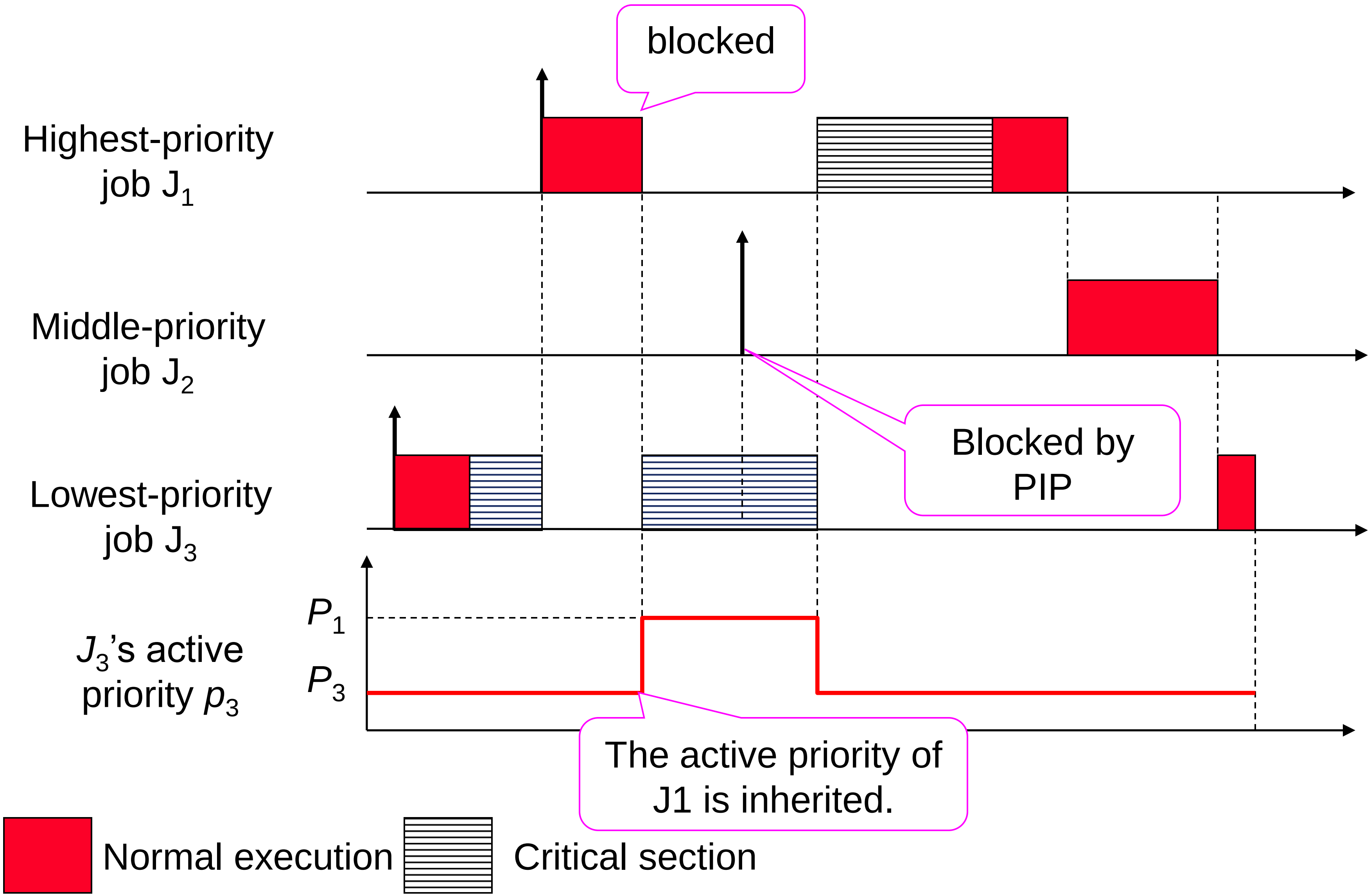
Priority Inheritance Protocol

- Modify the priority of tasks in critical sections
- When a task blocks higher-priority tasks, it temporarily ***inherits*** the highest priority of the blocked tasks.
    - Prevents preemption of medium-priority tasks

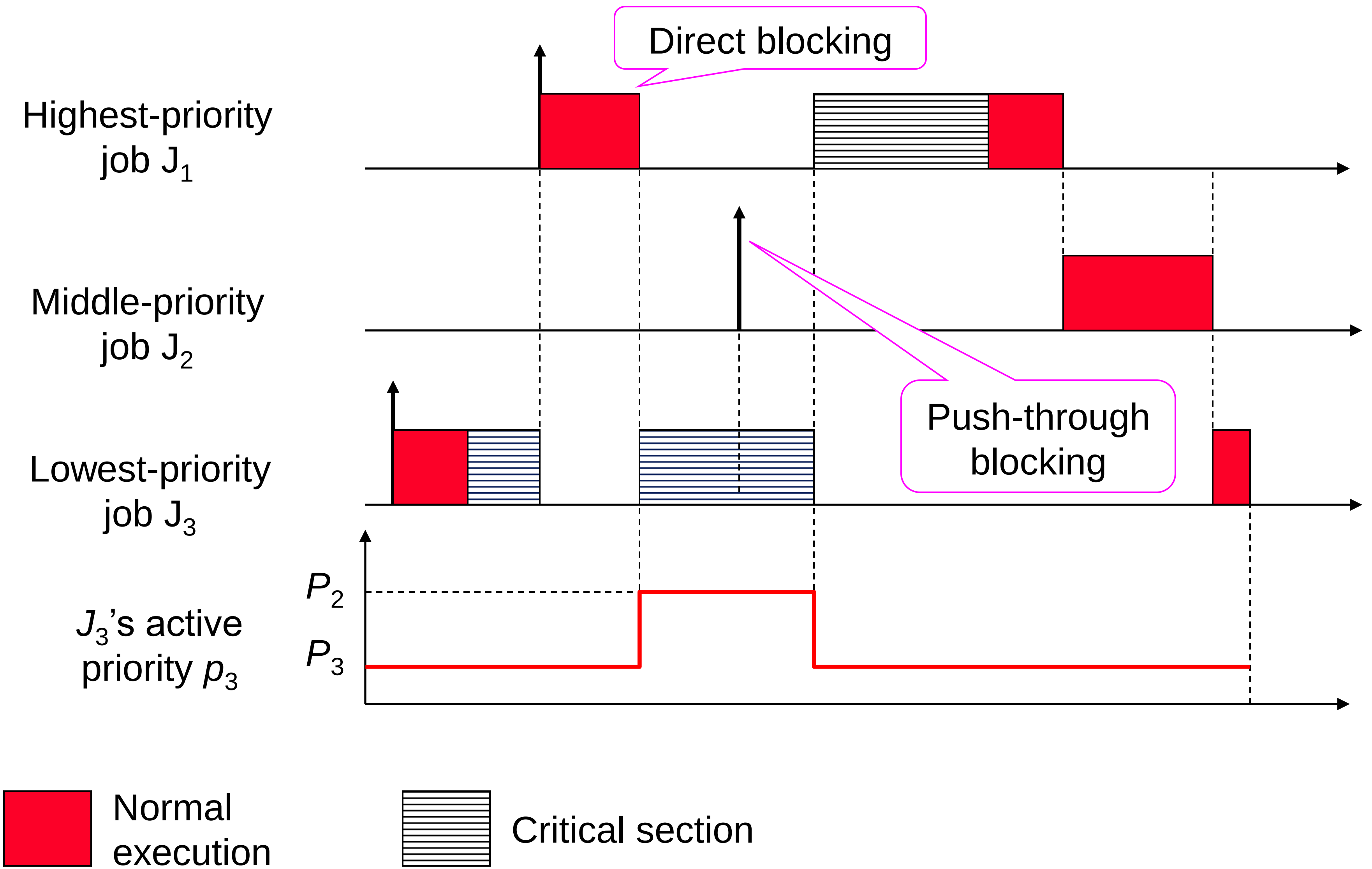
Protocol definition

- Jobs are scheduled based on their active priorities
- When the higher-priority job J_{high} is blocked on a semaphore because the lower-priority job J_{low} is in execution of its critical section, **the active priority p_{high} of J_{high} is inherited** to that of J_{low}.
- The rest of the critical section of J_{low} is executed with the active priority p_{high}.
- In case the medium-priority job J_{medium} activates, it cannot preempt the execution of J_{low} → Unbounded priority inversion is avoided.
- Priority inheritance is transitive; if a job J_3 blocks a job J_2, and J_2 blocks a job J_1, then J_3 inherits the priority of J_1 via J_2.

Example

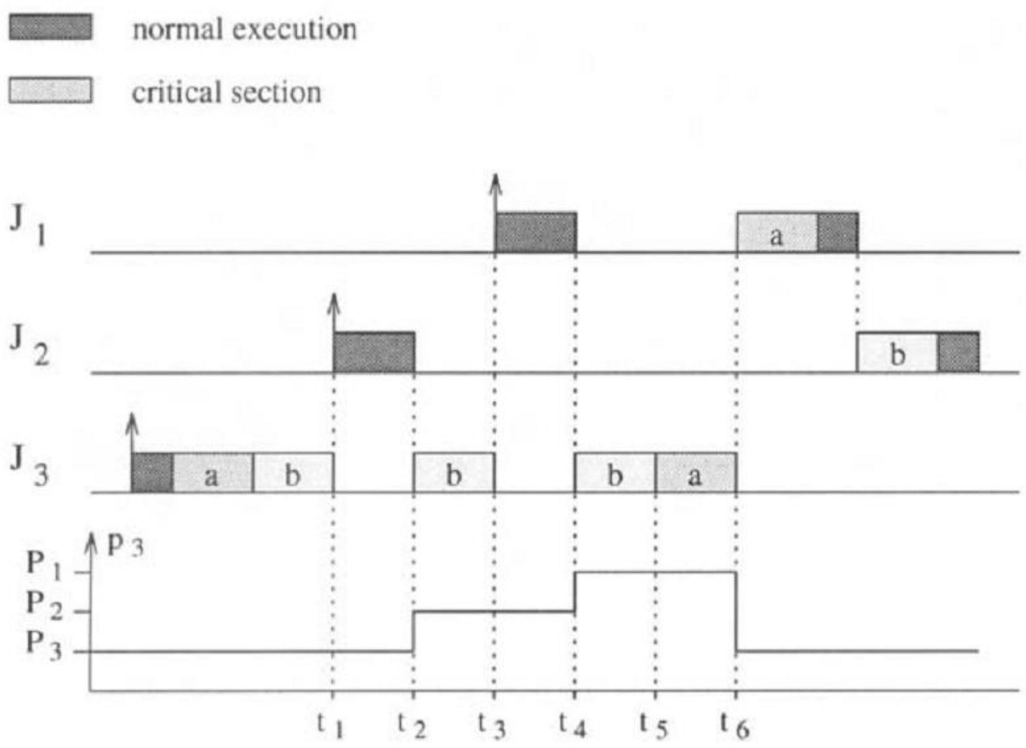

blocked
Highest-priority job J_1
Middle-priority job J_2
Blocked by PIP
Lowest-priority job J_3
P_1
J_3's active priority p_3
P_3
The active priority of J1 is inherited.
Normal execution
Critical section

Direct blocking & Push-through blocking



PIP with nested critical sections

- When the blocking job J_k exits the critical section, the blocked job with the highest priority is awakened.
- J_k replaces its active priority p_k by nominal priority P_k if no other jobs are blocked by J_k, or by the highest priority of the tasks blocked by J_k



Transitive priority inheritance

Properties

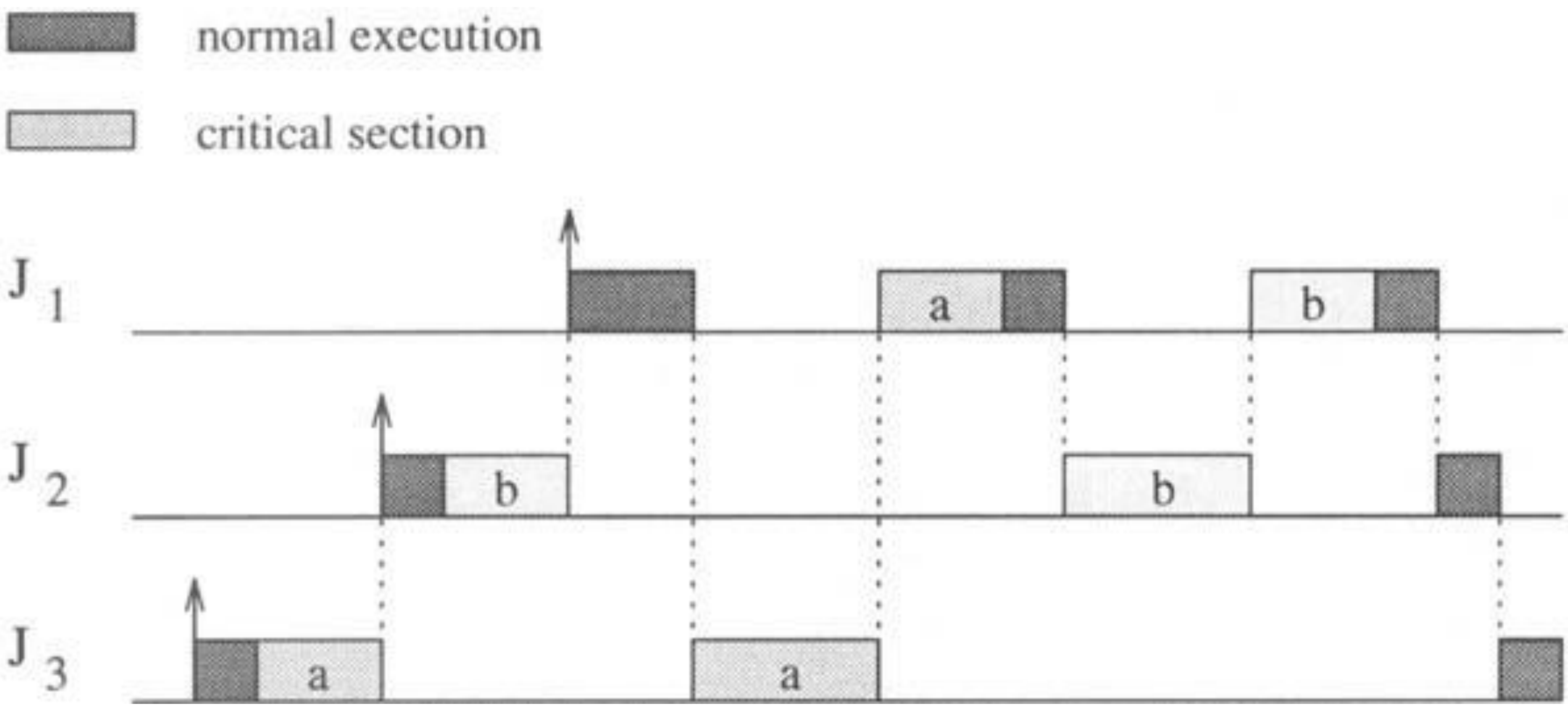
- Push-through blocking to job J_i occurs only if the semaphore is accessed by a job J_{low} with $p_{\text{low}}<p_i$ and by a job J_{high} with p_{high} that can be equal or higher than p_i
- Transitive priority inheritance can occur only in the presence of nested critical sections.
- If there are *n* lower-priority jobs that can block a job J_i, then J_i can be blocked at most the duration of *n* critical sections.
- If there are m distinct semaphores that can block a job J_i, then J_i can be blocked for at most the duration of *m* critical sections.

Properties

- Under the priority inheritance protocol, a job *J* can be blocked for at most the duration of min(*n*,*m*) critical sections.
  - n is the number of lower-priority jobs that could block *J*
  - m is the number of distinct semaphores that can be used to block *J*

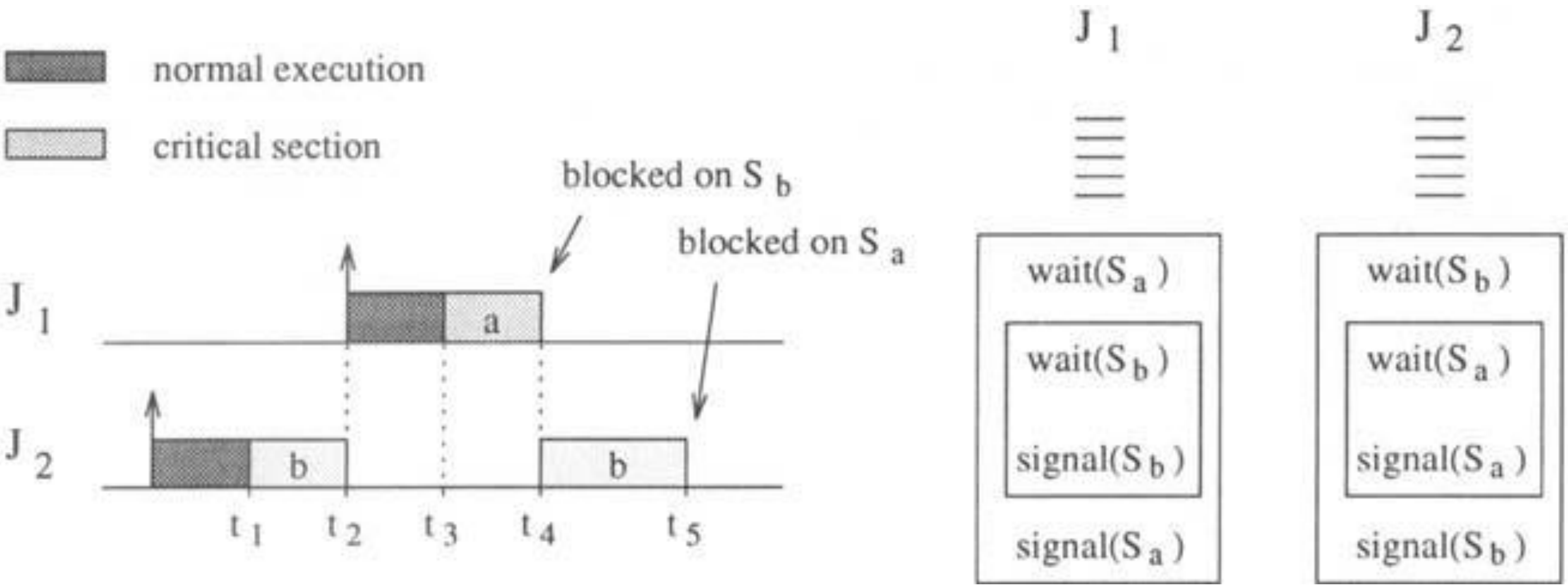
➔ *The maximum blocking time for any task J is bounded*

Remaining problem 1: Chained blocking



➔ J1 can be blocked several times

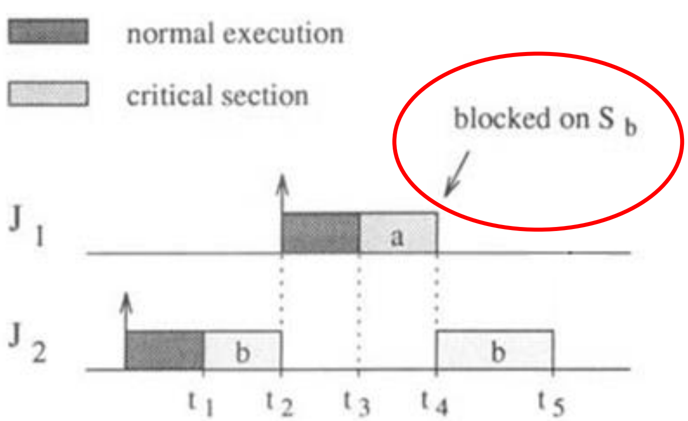
Remaining problem (2): Deadlock

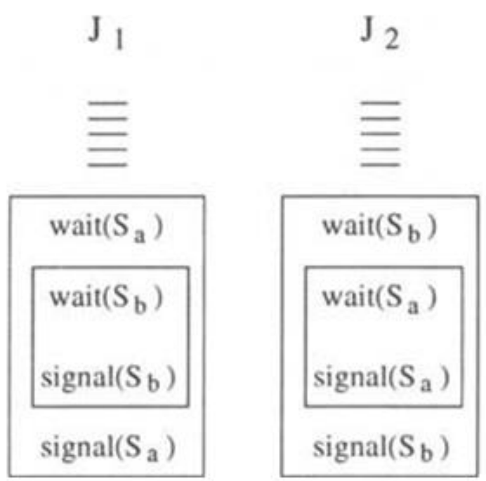


➔ Deadlock caused as J2 enters the nested critical session

Priority Ceiling Protocol

- Extends the Priority Inheritance Protocol
- Assign each semaphore a ceiling priority, equal to the priority of the highest-priority task that can lock it.
- Provided a critical section contains several semaphores, a job J can enter the critical section only when its priority is higher than all priority ceilings of the semaphores already locked by other jobs.





Protocol definition (1)

- S_k: an arbitrary semaphore
- $C(S_k)$: priority ceiling of S_k

$$C(S_k) \stackrel{\mathrm{def}}{=} \max_i \{P_i \mid S_k \in \sigma_i\}$$

This value can be computed offline

- J_i: the job with the highest priority in ready queue
- P_i: the priority of J_i
- S^*: semaphore with the highest priority ceiling among all the semaphores currently locked by jobs other than J_i

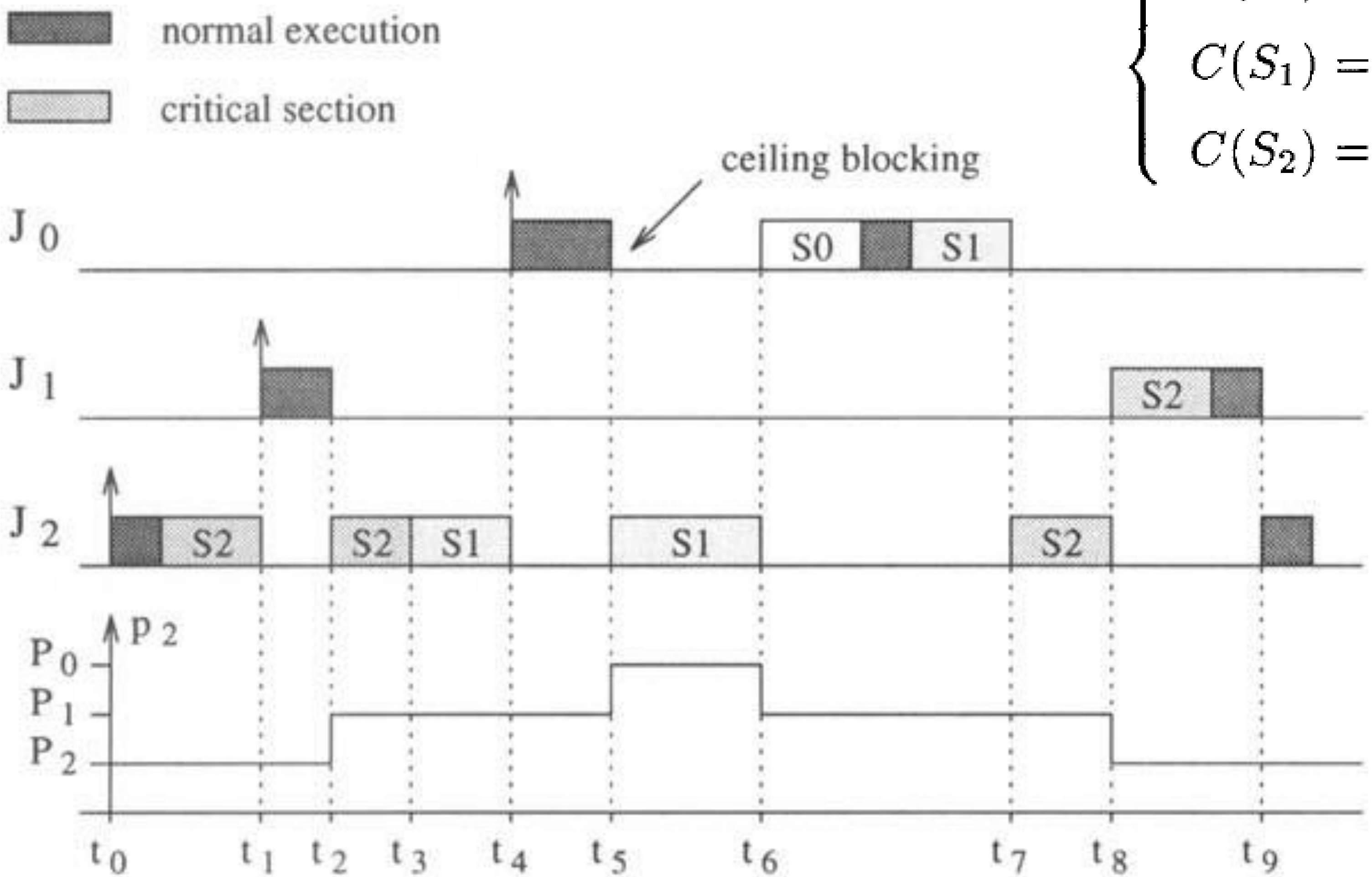
Protocol definition (2)

- When J_i is about to enter a critical section guarded by semaphore S_k,
  - If $P_i \leq C(S^*)$
    - locking on S_k is denied, &
    - J_i is blocked on semaphore S^* by the job holding the lock on S^*.
  - If $P_i > C(S^*)$
    - J_i locks on S_k *and continue execution*
- When J_i is blocked on a semaphore S,
  - The job J_k locking on S inherits the priority p_i
  - Generally, a task inherits the highest priority of the jobs blocked by it.
- When J_k exits a critical section & unlocks the semaphore,
  - If there are blocked jobs, then p_k is the highest active priority of the jobs blocked by J_k
  - Otherwise, p_k is restored to P_k

Example

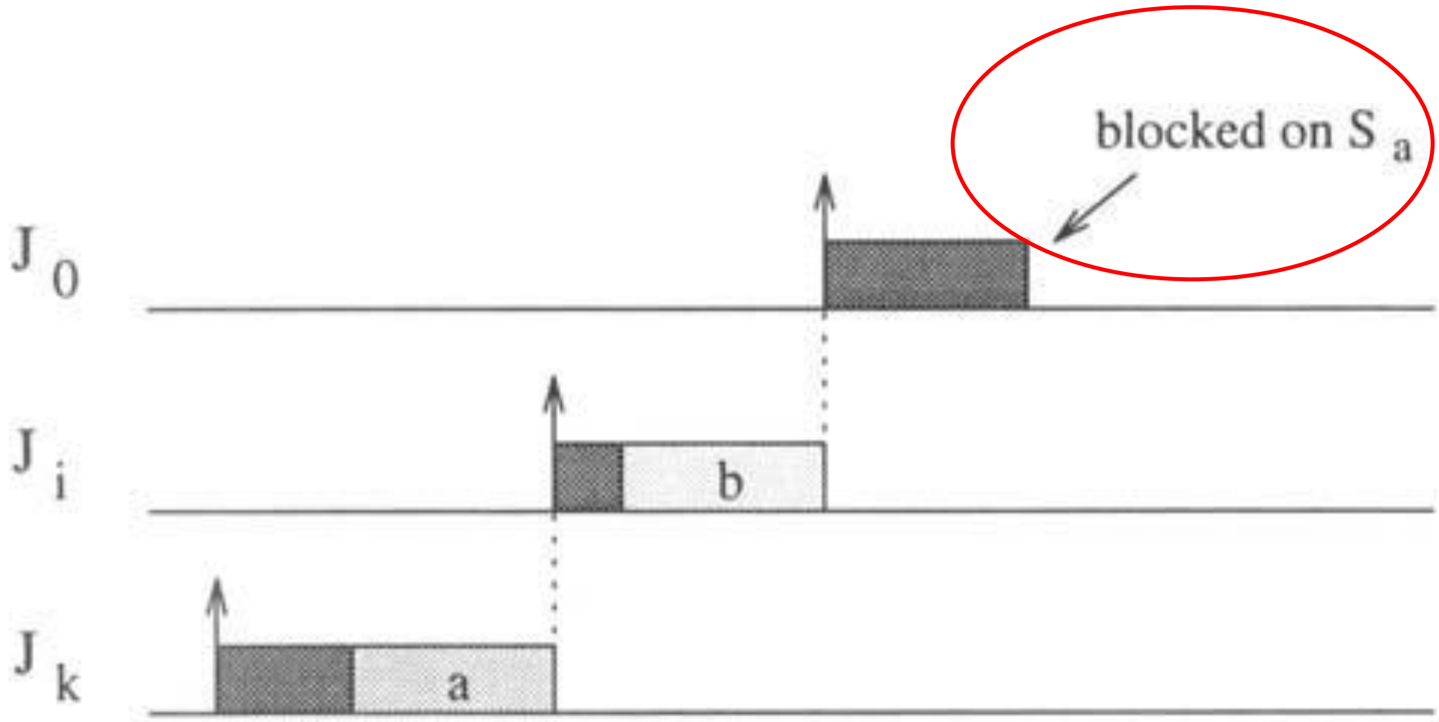
- Ceiling blocking

$$\begin{cases} C(S_0) = P_0 \\ C(S_1) = P_0 \\ C(S_2) = P_1 \end{cases}$$



Ceiling blocking

- A task is blocked by the protocol because of the priority ceiling condition
- Necessary to avoid chained blocking and deadlock



This will never happen with PCP

Properties of the protocol (2)

- The Priority Ceiling Protocol prevents deadlocks.
- Under the Priority Ceiling Protocol, a job J_i can be blocked for at most the duration of one critical section.

→Reduce blocking time
→Avoid unnecessary high-priority tasks blocking
→Avoid deadlock

Comparison

	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

SRP (Stack Resource Protocol): for student's further reading