Embedded Operating System RTOS resource access protocols

Graziano Pravadelli

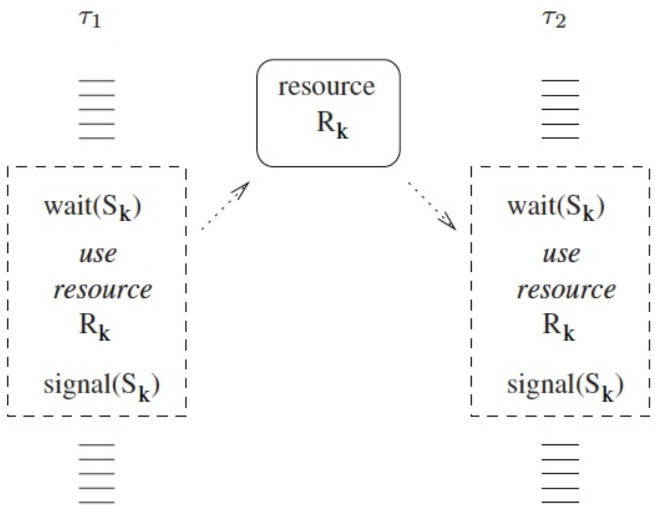
Resource

A resource is any structure necessary to advance processe execution

- Data structure, variables, memory area, peripheral device, …

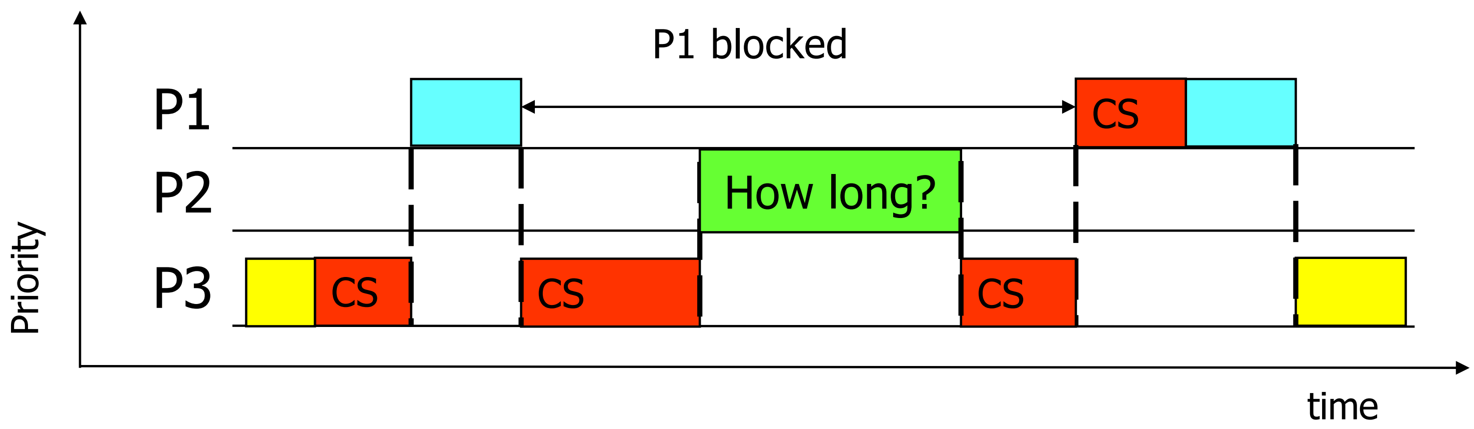
To ensure consistency, access protocols are necessary for mutual exclusion

- Critical section guarantee mutual exclusion generally by semaphores



Priority inversion

- Semaphore mechanism not suited for real time applications
- High priority task is blocked by low priority task for unbounded time



Priority inversion – Solutions

- Non-preemptive protocol
- Highest locker priority protocol
- Priority inheritance
- Priority cealing

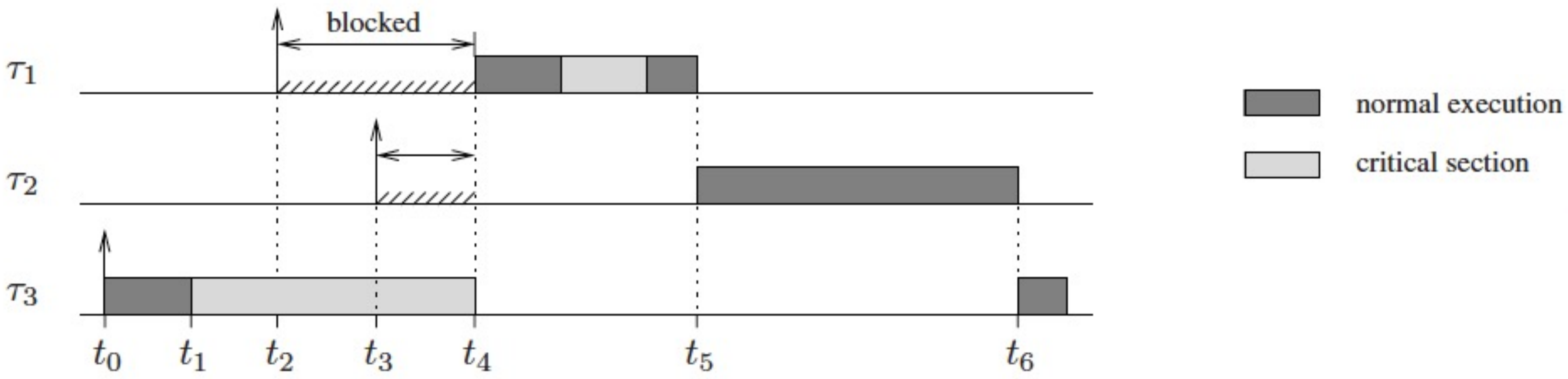
Notation

- P_i = nominal priority of task t_i
- p_i = acquired priority by task t_i

Non-preemptive protocol

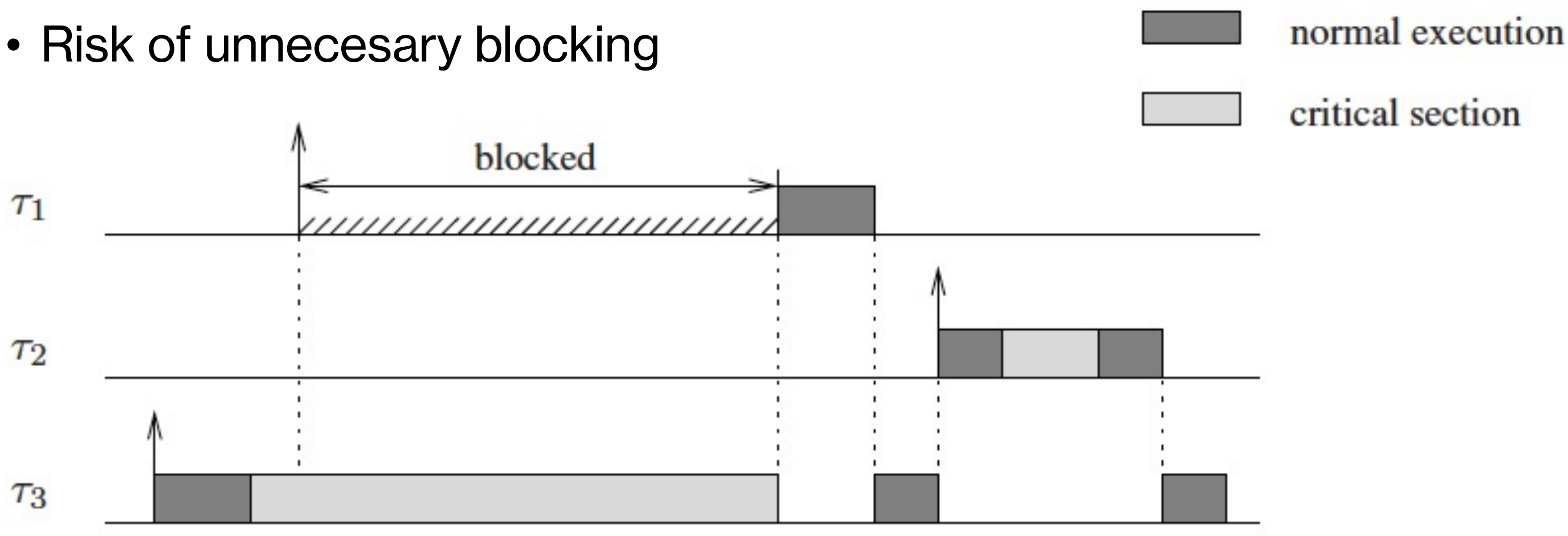
$$p_i(R_k) = \max_h \{P_h\}$$

- Raising the priority of a task to the highest whenever it enters a CS
- Back to nominal priority when it exits CS



Non-preemptive protocol - Problem

- NPP suited only for short CS
- Risk of unnecesary blocking



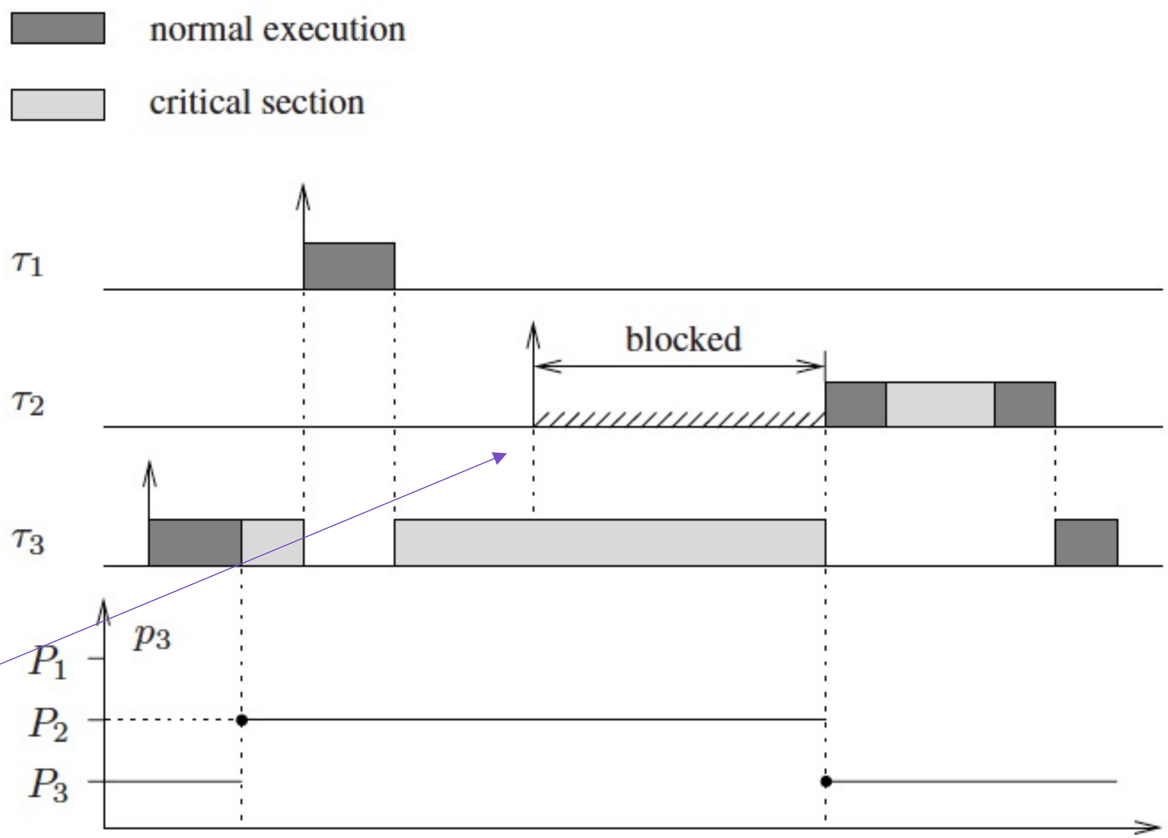
Highest locker priority protocol

- Like NNP, but the priority is raised to the highest priority among tasks sharing the same resource

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

Problem

- τ_2 blocked at the time it attempts to preempt τ_3, before it actually requires the shared resource



Priority inheritance

$$p_j(R_k) = \max\{P_j, \max_h\{P_h | \tau_h \text{ is blocked on } R_k\}\}$$

Postpone the blocking condition at the entrance of the CS rather than at the activation time

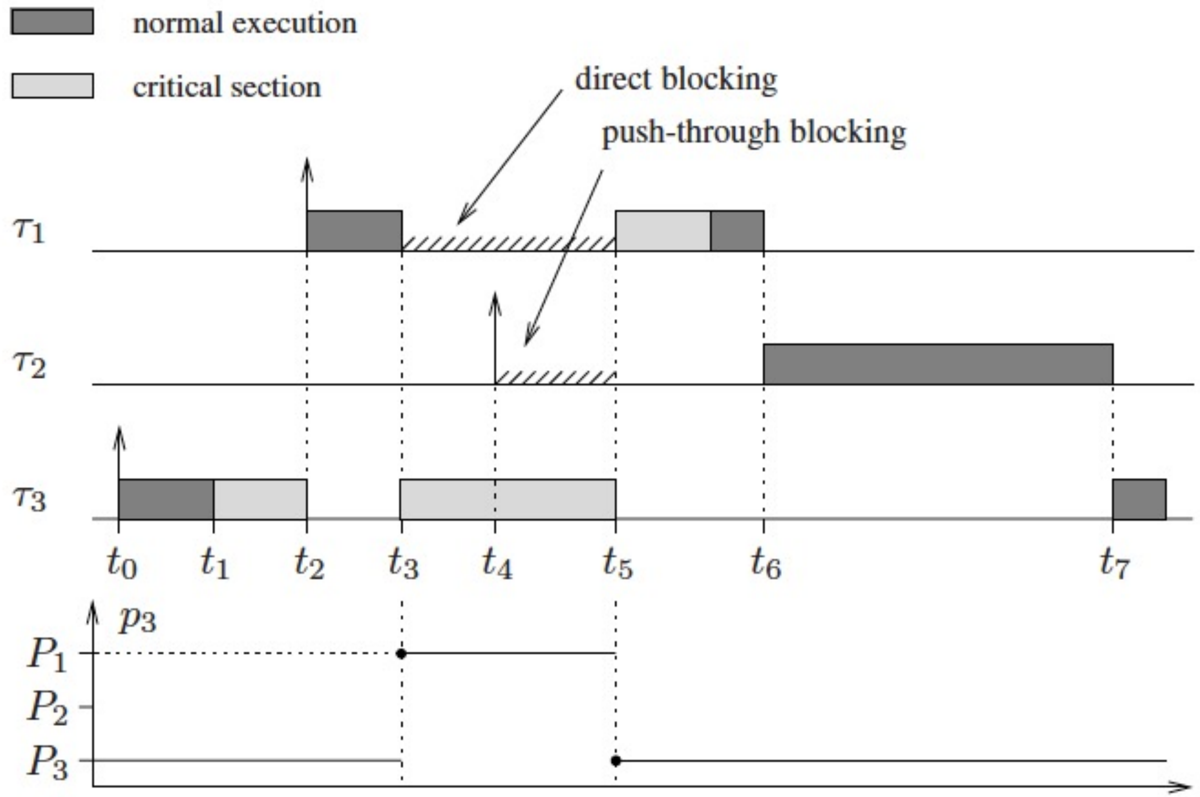
A task τ_j uses its nominal priority until it enters CS and blocks higher priority tasks

When enter CS, τ_j inherits PH , the highest priority of the tasks blocked by τ_j	When exits CS, τ_j resumes the priority it had at the point of entry into the CS

Priority inheritance - Characteristics

When τ_j exits CS	if no other task is blocked by τ_j	p_j set to its nominal priority P_j
	otherwise	p_j set to the highest priority of tasks blocked by τ_j
Advantage	Transparent to scheduler	
	Transitive	if τ_1 blocks τ_2 and τ_2 blocks τ_3, τ_1 blocks τ_3 via τ_2
Disadvantage	Deadlock possible in the case of bad use of semaphores	
	Chained blocking	if τ_i accesses *n* resources locked by processes with lower priorities, τ_i must wait for *n* CS

Priority inheritance – Example



At t_3 τ_1 requires CS, but it must wait because τ_3 locks CS

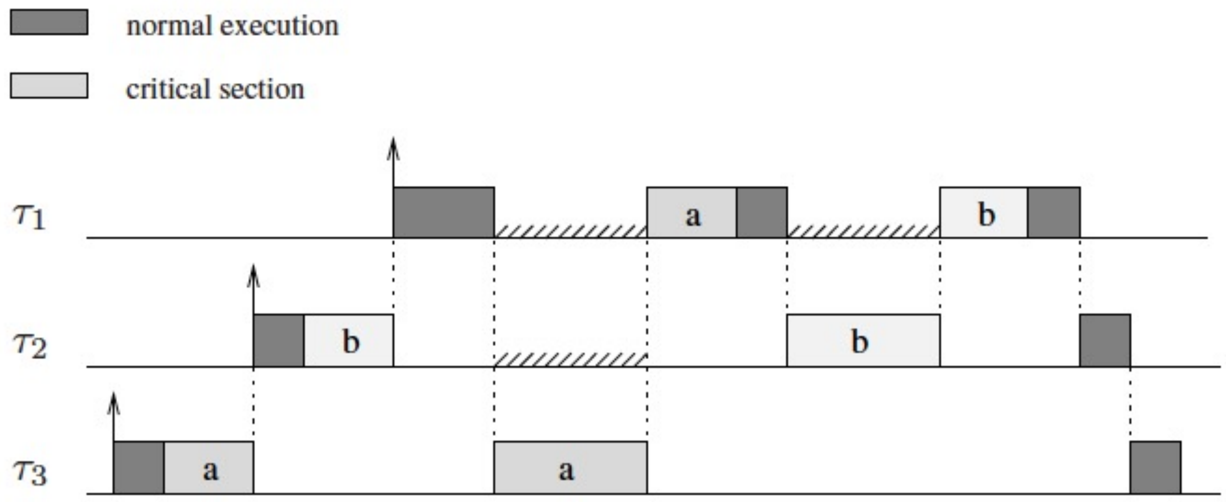
Thus, τ_3 inherits the priority of τ_1 and it can resume its execution

At t_4, τ_2 arrives, but τ_3 cannot be preempted by because it inherited the priority of τ_1

Thus, τ_2 must wait until τ_3 exits CS and τ_1 finishes

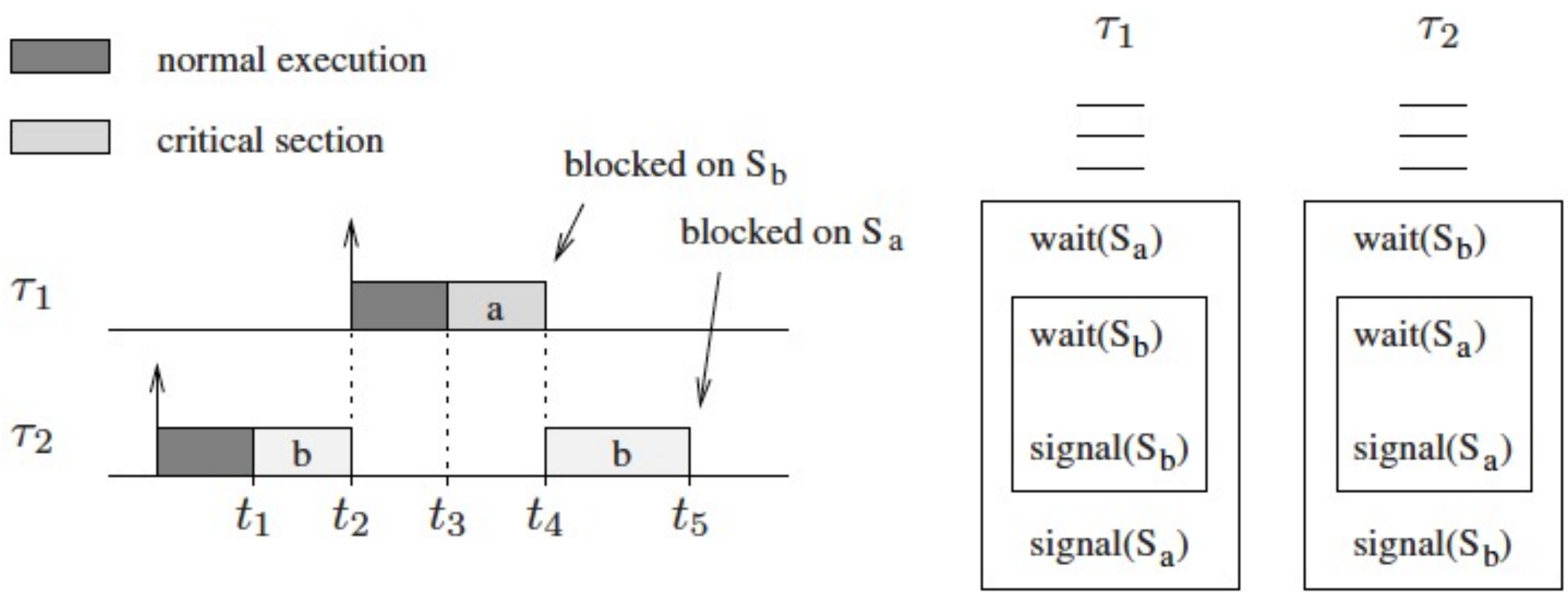
Priority inheritance – Chained blocking

- if τ_1 accesses *n* semaphores locked by *n* lower-priority tasks, τ_1 will be blocked for *n* critical sections



Priority inheritance – Deadlock

- Not a problem of the priority inheritance, but it does not prevent it



Priority ceiling

Extends priority inheritance with a rule for granting a lock on a free semaphore

To avoid multiple blocking, the rule does not allow a task entering CS if there are locked semaphores that can block it

Each resource S_k has a priority ceiling $C(S_k)$ equal to the priority of the highest-priority task that can lock it

A task is allowed entering a CS only if its priority is higher than all priority ceilings of resources locked by the other task

When a task enters CS, no more block by lower-priority tasks

Priority ceiling – Implementation (1)

Let τ_i the task with the highest priority among tasks ready to run

τ_i is assigned to the CPU

Let S^* the resource such that $C(S^*) > C(S_j)$ for all S_j locked by $\tau_n \neq \tau_i$

τ_i acquires S_k iff $p(\tau_i) > C(S^*)$

If $p(\tau_i) <= C(S^*)$, τ_i is blocked on S^* and it cannot acquire S_k

Priority ceiling – Implementation (2)

When τ_i is blocked on R

- it transmits its priority to τ_k that locks R
- τ_k resumes and executes its CS with $p(\tau_i)$

When τ_k exits CS

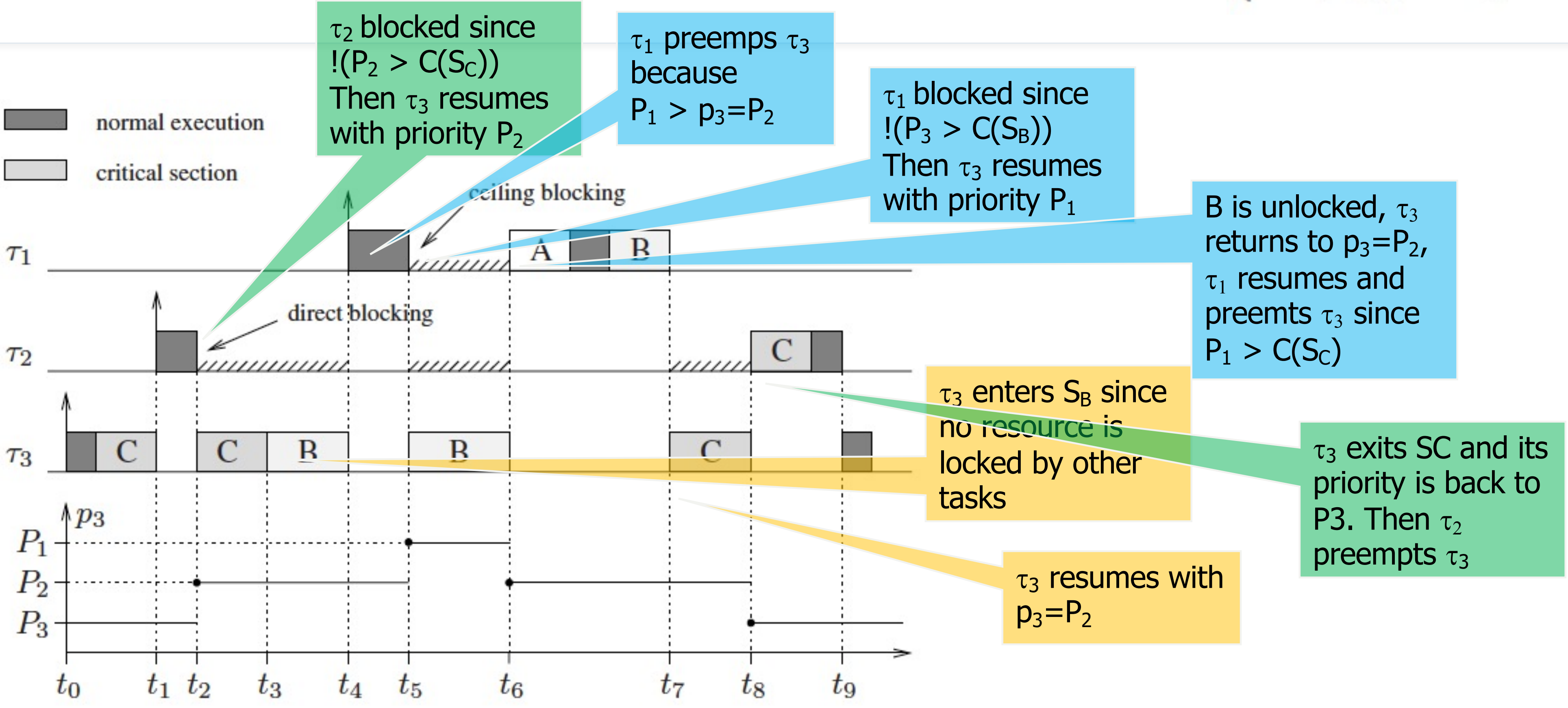
- it unlocks R
- the highest priority job blocked on R is awakened

Priority of τ_k is updated as

- if no task blocked by τ_k, the priority of τ_k is the nominal
- otherwise it is the highest priority of tasks blocked by τ_k

Priority ceiling – Example

$$\begin{cases} C(S_A) = P_1 \\ C(S_B) = P_1 \\ C(S_C) = P_2 \end{cases}$$



Priority ceiling - Characteristics

Priority inheritance is transitive

A high-priority process can be blocked at most once during its execution by lower-priority processes

Transitive blocking is prevented

Deadlocks are prevented

Mutual exclusive access to resources is ensured by the protocol itself (no semaphores etc. required)

Tasks can share resources simply by changing their priorities, thus eliminating the need for semaphores