Resources Management

Real-Time Industrial Systems

Marcello Cinque

Roadmap

- Priority inversion problem
- Non-Preemptive Protocol and Highest Locker Priority
- Priority Inheritance and Priority Ceiling

- References:
  - Giorgio Buttazzo: “Hard real-time computing systems: Predictable Scheduling Algorithms and Applications”, Third Edition, Springer, 2011

Use of shared resources

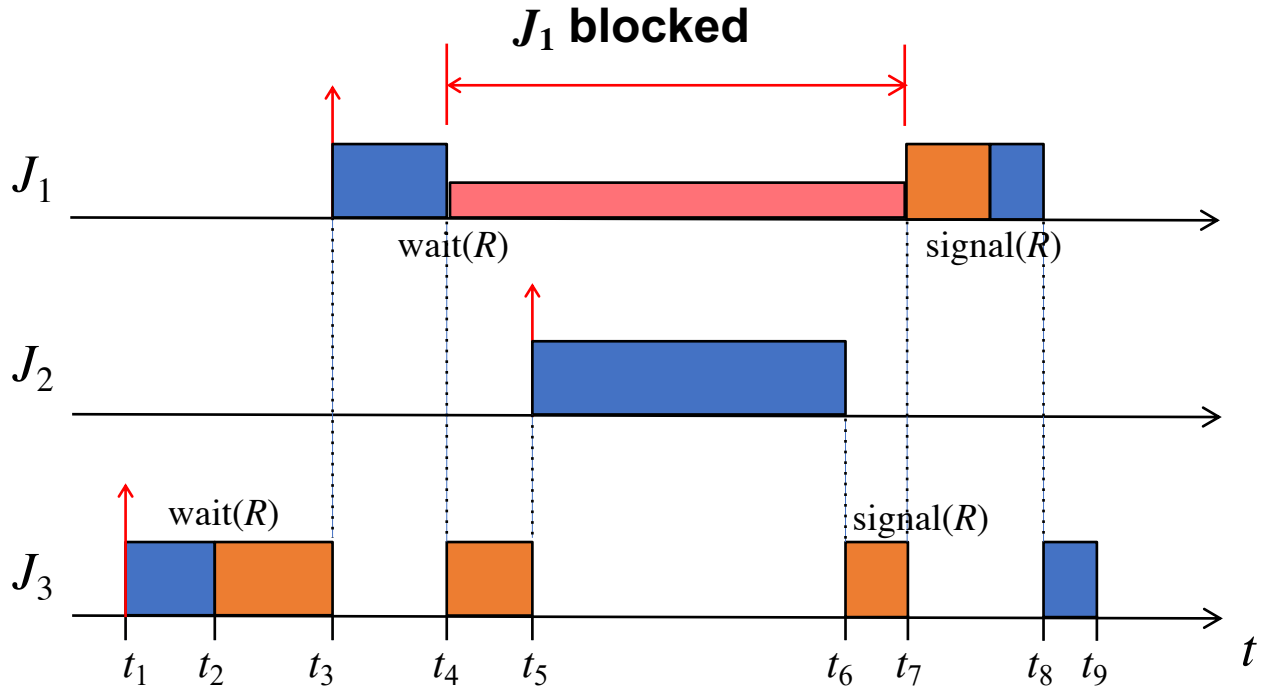
- Concurrent real-time tasks (or kernel threads) can share resources that need to be accessed in mutual exclusion
- When resources are occupied, tasks can be blocked
- The use of classical synchronization mechanisms, like locks, semaphores, and monitors, lead to the priority inversion problem.
- This may harm the schedulability and predictability of the system!

Priority inversion problem

normal execution

critical section

Three tasks J_1, J_2 e J_3 with decreasing priorities ($P_1 > P_2 > P_3$)



Unbounded blocking due to tasks with intermediate priority that can preempt the low priority task while it is using the shared resource

Non- Preemptive Protocol (NPP)

- Tasks in critical sections cannot be preempted
- Achieved by increasing the priority of the task entering the critical section to:

$$P^* = \max_i (P_i)$$

- Priority is taken back to its original value when the task leaves the criticial section

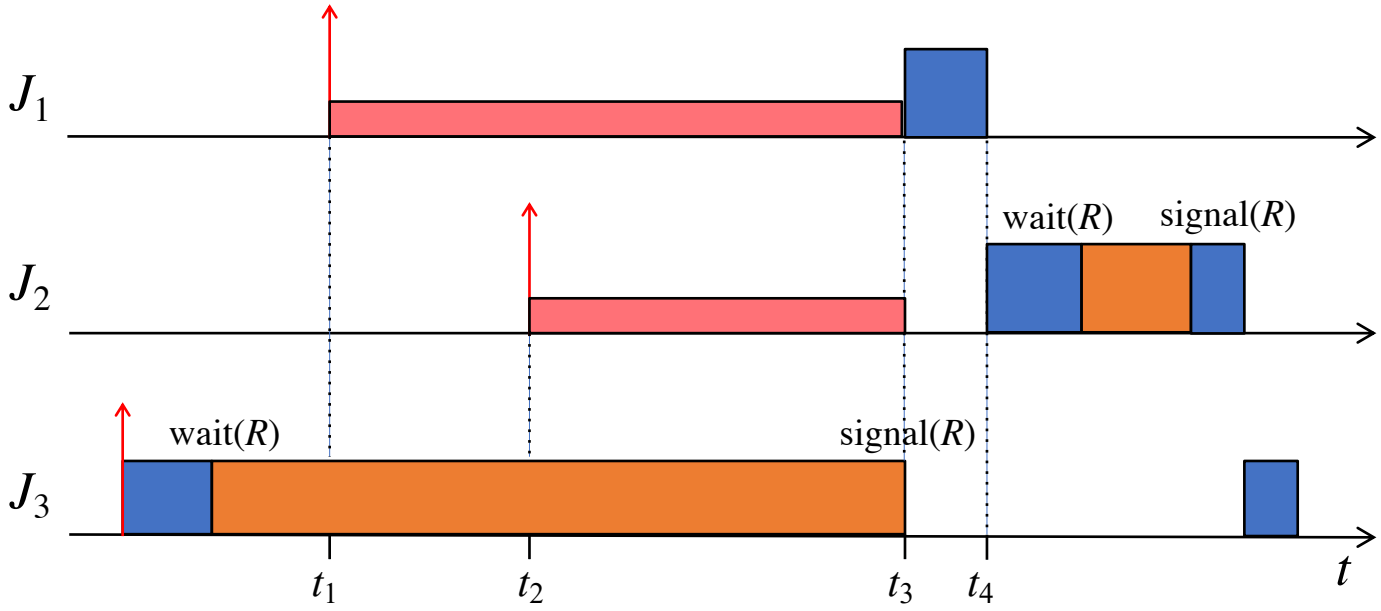
NPP: example

normal execution

crtitical section

P_3 raised to P_1

J_1 blocked, even if it does not use the resource!



Simple implementation but can cause useless blocking.
Recommended only for short critical sections.

Highest Locker Priority (HLP)

- To avoid useless blocking, we can assign the task entering the critical section a priority p^* equal to the largest priority level among the tasks the use that resource:

$$p^*(R) = \max_k (P_k : J_k \text{ uses } R)$$

- The priority must return to its original value when the task exits the critical section and do not cause further blockings

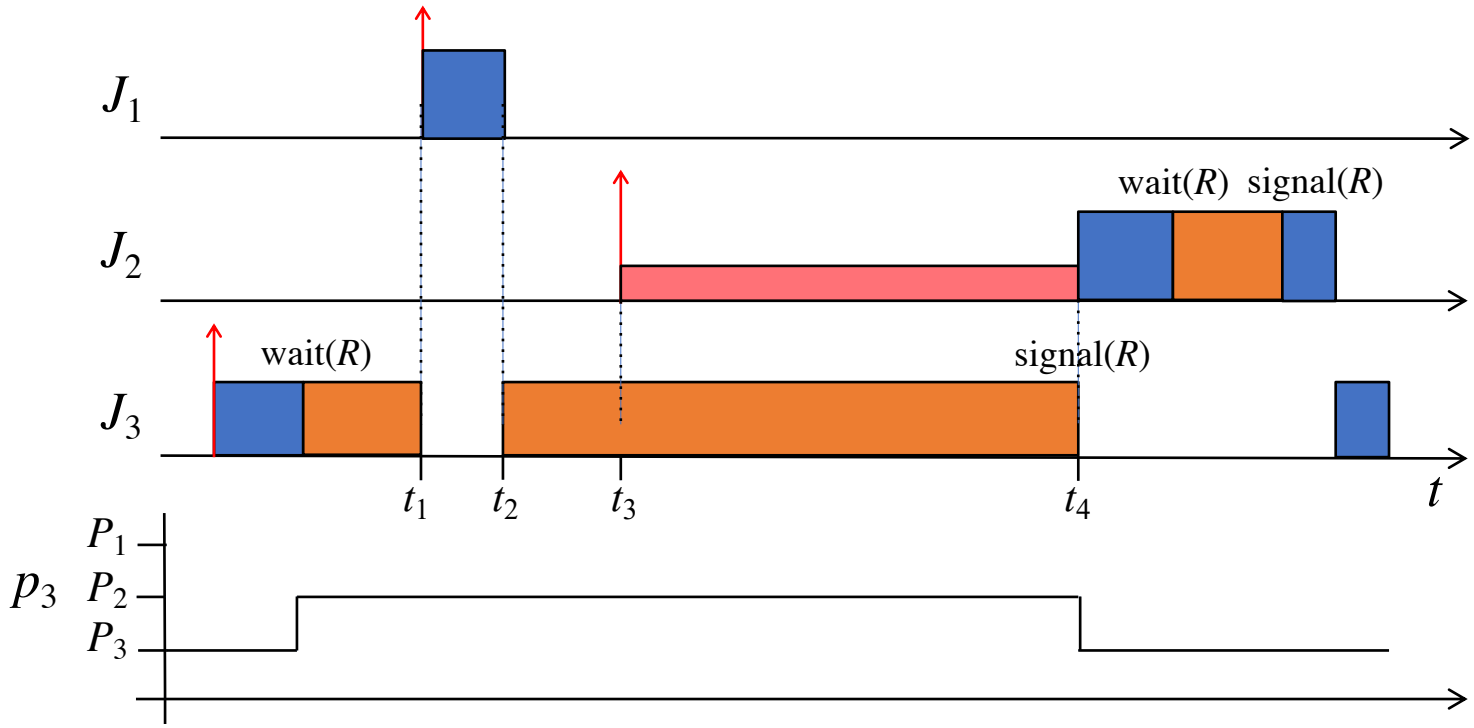
HLP: example

normal execution

critical section

P_3 raised to P_2

J_1 does not block



It is based on anticipated blocking:

- J_2 blocks when it tries to preempt J_3, not when it asks the resource with wait(R)
- What if J2 does not use the resource (wait(R) is in the body of a if statement)?
- Can we do any better?

Eredito la priorità

Priority Inheritance (PI)

- Key idea: raise the priority of a task only when needed, i.e., when a high priority task requests the resource used by a low priority task

In pratica per gli esempi di prima: J3 ha la risorsa ed ha P=4, J1 vuole usare la risorsa ed ha P=1, dovrebbe prendersela lui dato che è più elevata ma non lo fa, J1 trasferisce la sua priorità pari ad 1 a J3 che quindi cambia priorità solo quando richiesto, non subito

- High priority tasks block when the resource is occupied, to assure the consistency of the resource (*direct blocking*)

- Medium priority tasks block when they try to preempt low priority tasks using a resource requested by a high priority task, to avoid priority inversion (*indirect* or *push-through blocking*)

PI: rules

- When a task J_a blocks when accessing a critical resource, it transmits its priority to task J_b that is using it
  - J_b inherits the highest priority of the tasks it currently blocks

Ji block: intende che Ji è più prioritario e vuole usare una risorsa critica ma si blocca perchè la sta usando un altro task.

- When J_b exits the critical section, it keeps the maximum priority among the tasks it still blocks
  - If no more task is blocked by J_b, it resumes its original priority

Jb dopo avere utilizzato la risorsa non ridà subito la priorità a Ja ma osserva: se non ci sono altri task che bloccano (condizione di sopra) restituisce la priorità a Ja altrimenti la trasmette ad un Jx

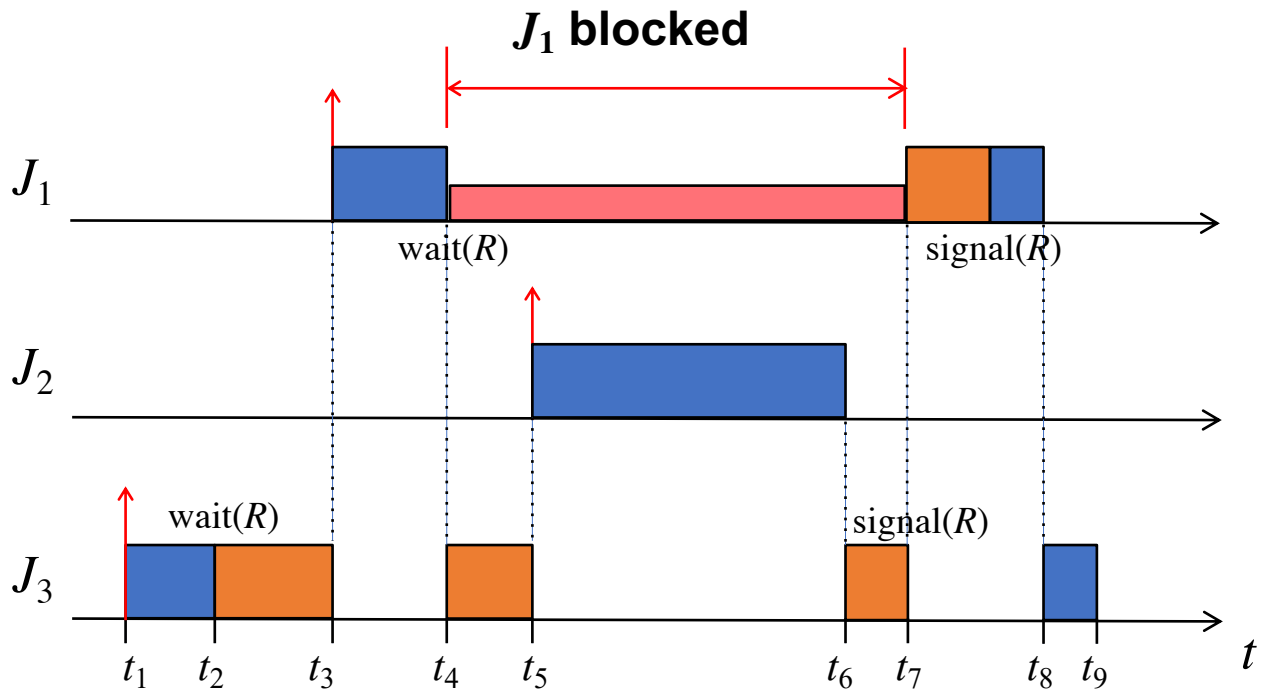
- Priority inheritance is transitive
  - If J_3 blocks J_2 that blocks J_1, then J_3 inherits J_1 priority through J_2

Priority Inversion

normal execution

critical section

Three tasks J_1, J_2 e J_3 with decreasing priorities ($P_1 > P_2 > P_3$)

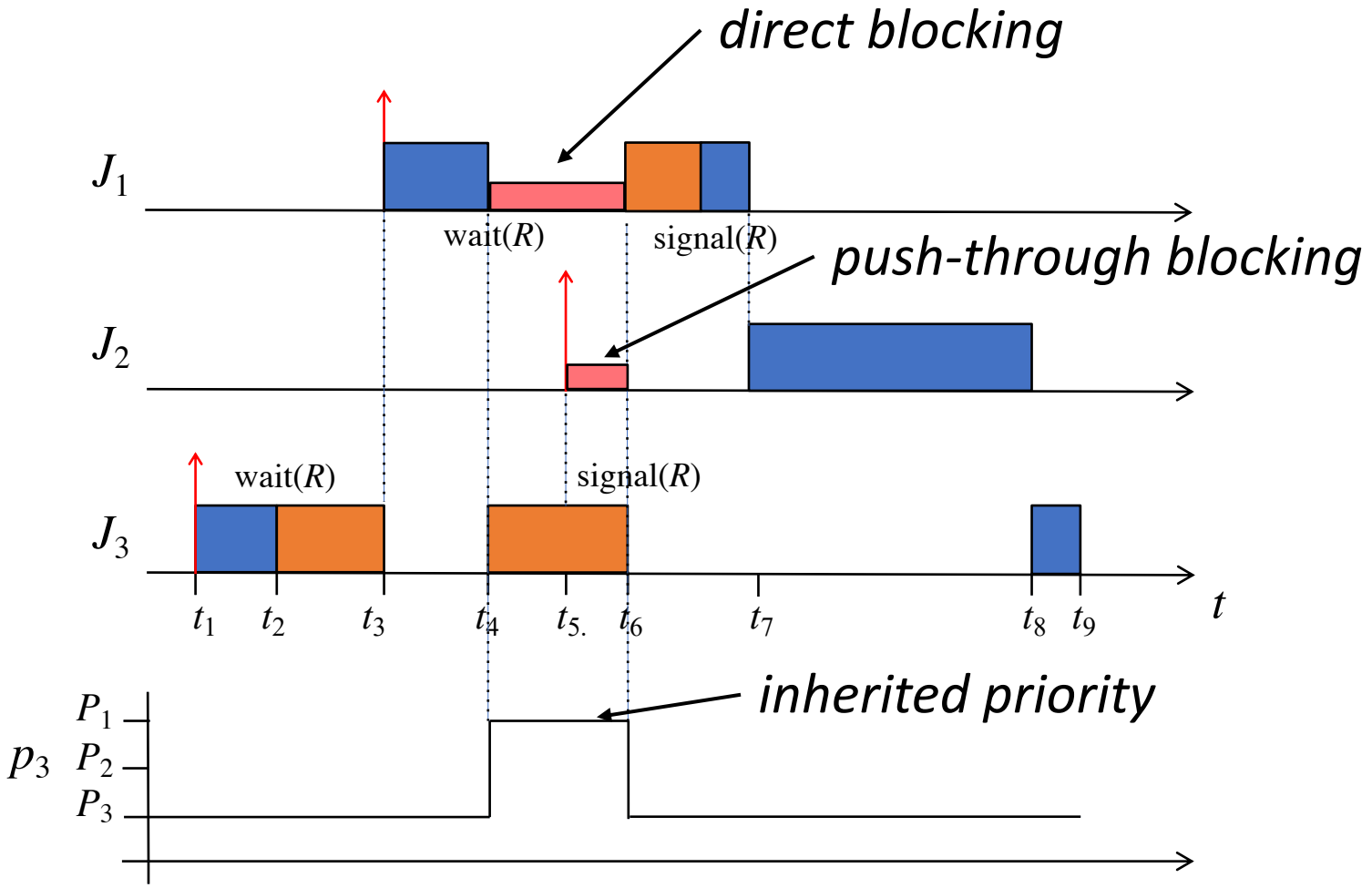


Priority Inheritance Solution

normal execution

critical section

Now J_1, is no longer delayed by J_2 that cannot preempt J_3 while R is occupied



PI: feasibility analysis

- With PI it is possible to calculate an upper bound to the blocking time of a task, allowing to define a schedulability test
- If B_i is the maximum blocking time computed for task τ_i then the task set is feasible if:

time blocked in the worst case

$$\forall i, 1 \leq i \leq n, \quad \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq i(2^{1/i} - 1)$$

We assume Ti=Di

- Response time analysis can be used as well adding B_i to the formula:

$$R_i = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

PI: evaluation of the blocking time

Nella matrice delta possiamo avere anche degli elementi pari a zero, potremo quindi frettolosamente concludere che tale processo non sarà mai bloccato su quella risorsa, ma non è vero potrebbe venire bloccato indirettamente

Matrice Ji x Si (ogni processo quanto potrebbe essere bloccato su di un semaforo Si)

- Let $\boldsymbol{\delta}$ be the matrix of critical sections' maximum durations
  - δ_{ik} is the maximum duration of the critical section of task J_i on resource R_k
- and $c(R_k)$ be the *ceiling* of a resource R_k: $c(R_k) = \max_j\{P_j : J_j \text{ uses } R_k\}$
- The blocking time B_i of the task J_i can be computed as:

iL CEALING è qualcosa di cui abbiamo bisogno prima di fare le nostre analisi, non dinamicamente. Rappresenta la massima priorità di un processo che utilizza la risorsa Rk

$$B_i = \min\{B_i^l, B_i^s\}$$

- where:

$$B_i^l = \sum_{j=i+1}^{n} \max_k\{\delta_{j,k} : C(S_k) \geq P_i\}$$

Sulle righe

Considero quelli sotto a Ji

$$B_i^s = \sum_{k=1}^{m} \max_{j>i}\{\delta_{j,k} : C(S_k) \geq P_i\}$$

Sulle colonne

Example

Regola RM per calcolo priorità:
P1=1 -> J12 priorità maggiore
P2=2
P3=3
P4=4

Matrice Delta

	C_i	T_i	S_1	S_2	S_3
J_1	5	25	1	2	0
J_2	15	60	0	9	3
J_3	20	100	8	7	0
J_4	20	200	6	5	4

Calcoliamo i cealing per capire quali righe o colonne scartare:

c1= 1 (S1 la usano J1,J2,J3 - il cealing sarà pari alla prio di J1
c2= 1 (S2 la usano J1,J2,J3,J4 - il cealing sarà pari alla prio di J1
c3= 2 (S3 la usano J2,J4 - il cealing sarà pari alla prio di J2

ora possiamo calcolare i massimi Blocking Time:

B1L -> consideramo da Ji+1 come da formula quindi da J2 in poi
B1L= 9+8+6 (i massimi delle righe)
B1S-> ora è diverso consideriamo solo le risorse che hanno cealing >= della priorità di J1
B1S= 8+9 (S3 ha cealing pari a 2 che è meno prioritario di J1)

B2L -> considero solo Ji+1 come da formula quindi parto da J3 in poi
B2L= 8+6
B2S -> J2 ha priorità 2 quindi vanno bene risorse con cealing pari a 2 o 1(tutte)
B2S=8+7+4 (si considerano le righe da i+1 in poi)

B3L -> considero solo Ji+1 come da formula quindi parto da J4 in poi
B3L= 6
B3S -> J3 ha priorità 3 quindi vanno bene risorse con cealing pari a 3, 2 o 1(tutte)
B3S= 6+5+4 (si considerano le righe da i+1 in poi)

B4L -> dovrei considerare da J4+1 quindi niente
B4L=0
B4S -> J4 ha priorità 4 quindi andrebbero bene tutte le risorse ma sonno di esso non ho più nulla
B4S=0

$$B_1^l = 9 + 8 + 6 = 23 \qquad B_1^s = 8 + 9 = 17 \qquad \boldsymbol{B_1 = 17}$$

$$B_2^l = 8 + 6 = 14 \qquad B_2^s = 8 + 7 + 4 = 19 \qquad \boldsymbol{B_2 = 14}$$

$$B_3^l = 6 \qquad B_3^s = 6 + 5 + 4 = 15 \qquad \boldsymbol{B_3 = 6}$$

$$B_4^l = B_4^s = 0 \qquad \boldsymbol{B_4 = 0}$$

Calcolo dei Tempi di Bloccaggio B_i in Priority Inheritance

	C_i	T_i
τ_1	5	20
τ_2	10	40
τ_3	15	60

$$\begin{pmatrix} S_1 & S_2 & S_3 & S_4 \\ 1 & 2 & 0 & 0 \\ 0 & 3 & 0 & 2 \\ 4 & 2 & 3 & 1 \end{pmatrix}$$

matrice di

Calcoliamo:

$B_1^l = 3 + 4 = 7$

$B_1^s = 4 + 3 = 7$ $\Rightarrow B_1 = \min(7, 7) = 7$

* Nel calcolo di B_1 considero solo i primi due semafori perché i ceiling di S_3 e S_4 sono inferiori della priorità di τ_1

$B_2^l = 4$

$B_2^s = 4 + 2 + 1 = 7$ $\Rightarrow B_2 = 4$

* Considero solo S_1 - S_2 - S_4 perché S_3 ha un ceiling inferiore alla priorità di τ_2 ($3 < 2$ priorità)

$B_3^l = 0$

$B_3^s = 0$ $\Rightarrow B_3 = 0$

Ora dobbiamo verificare i least Upper Bound di ogni task

① $\frac{5}{20} + \frac{7}{20} \leq 1 \Rightarrow \frac{12}{20} \leq 1$ OK!

② $\frac{5}{20} + \frac{10}{40} + \frac{4}{40} \leq 2(\sqrt{2} - 1) \approx 0,83 \Rightarrow 0,6 \leq 0,83$ OK!

③ $\frac{5}{20} + \frac{10}{40} + \frac{15}{60} \leq 3(2^{\frac{1}{3}} - 1) \Rightarrow$ OK!

$\Rightarrow$ Possiamo quindi schedulare i tre Task!

PI: discussion

- Requires modification to the classical semaphore mechanism, the implementation is however simple and generally present in RTOSes
- It is based on fixed-priority scheduling
  - Dynamic solutions are available as well, but they are not as adopted as PI

- It may suffer of *blocking chains* if a high priority task needs to access to more than one resource used by more than one low priority task
- It does not prevent *deadlocks*

Priority Ceiling (PC)

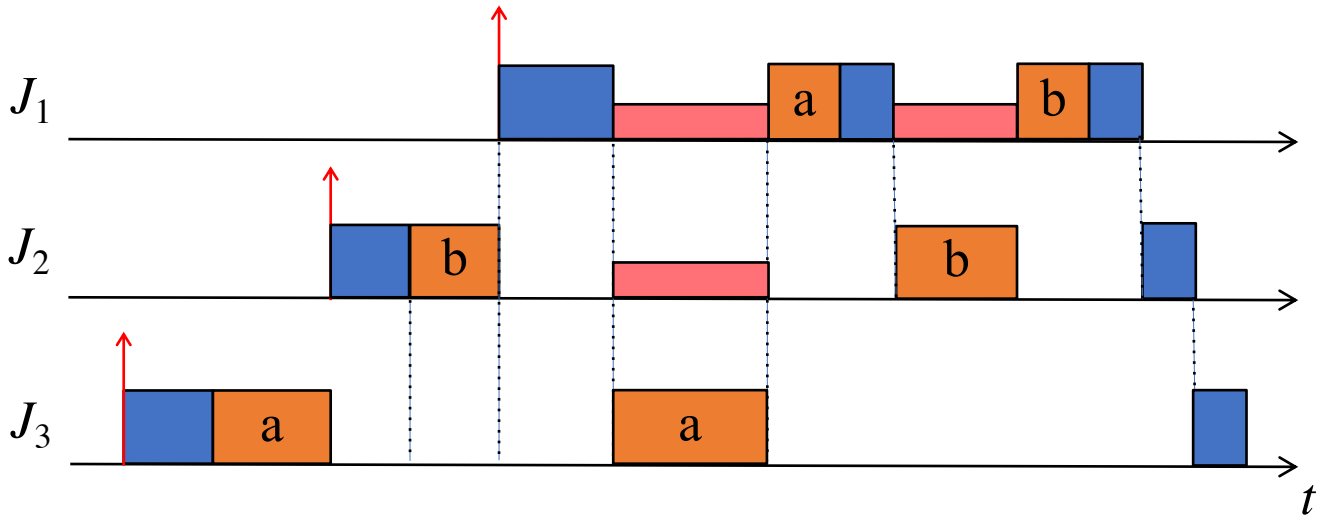
- It introduces a further check when a task tries to access a resource:
  - A task cannot access a resource if there are other resources currently in use that might block a higher priority task
  - Formally, if R^* is the resource with the highest ceiling $c(R^*)$ among all currently used resources, if the task J wants to enter the resource R, different from R^*, it must have a priority strictly higher than $c(R^*)$, else it is blocked

- This form of *dynamic anticipated blocking* (called *celling blocking*) prevents blocking chains and circular waits, and so, deadlocks

Blocking chain problem with PI

normal execution

critical section

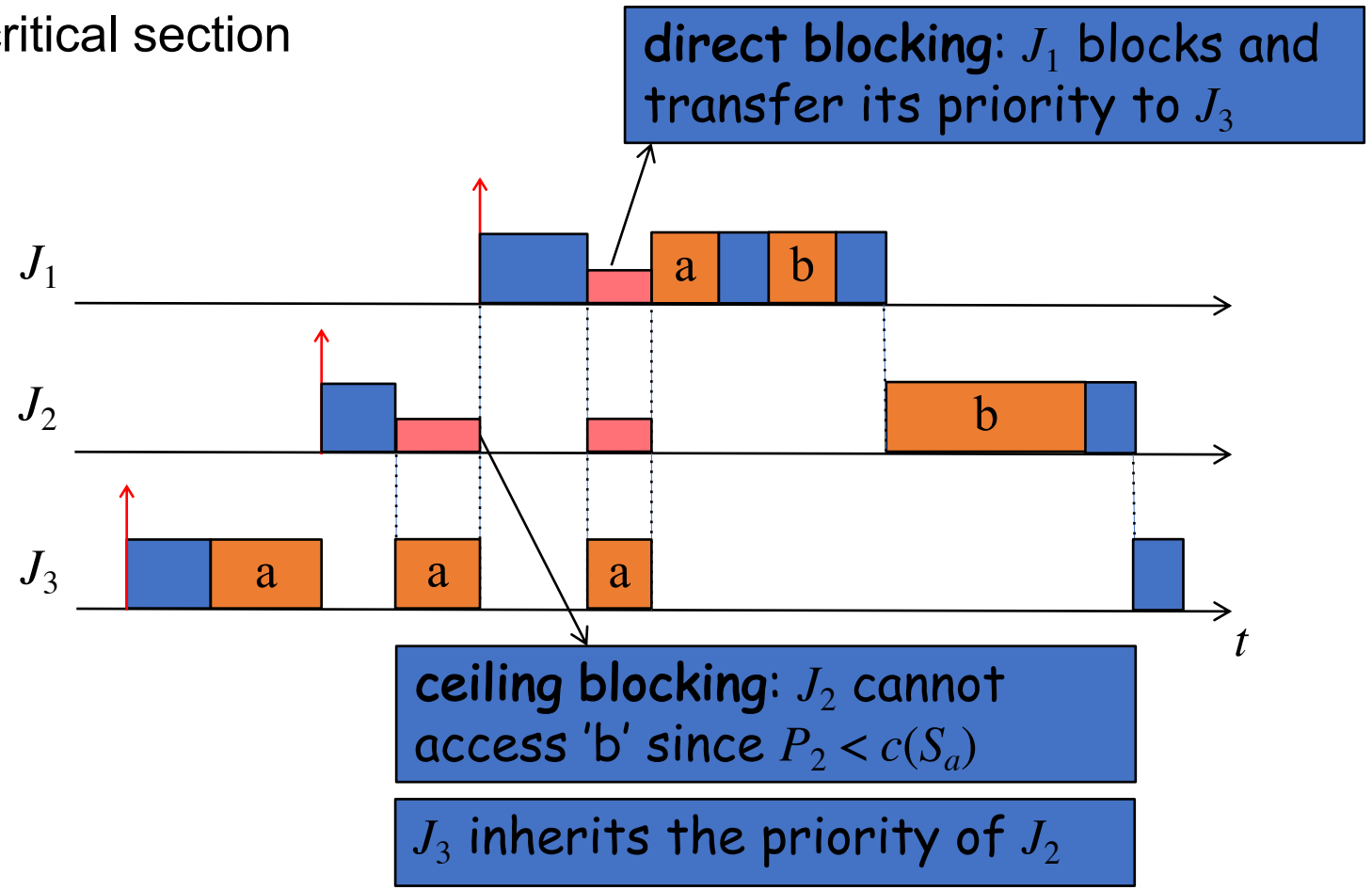


Solution with PC

normal execution

critical section

$c(R_a) = c(R_b) = P_1$



PC: discussion

- The ceiling blocking orders the access to resources in a way that when a high priority task enters the first resource after a blocking it will no more be blocked

Non ci saranno deadlock e circular wait

- Hence, the blocking time can be simply evaluated as the maximum among all crtical sections that can block a task:

$$B_i = \max_{jk}\{\delta_{jk} : P_j < P_i , c(S_k) \geq P_i\}$$

Nell'esempio di prima Bi sarà 9 invece che 17 e lo possiamo utilizzare nella formula della Slide13 del PI

- Then, the same feasibility analysis of PI applies
- It requires fixed-priority scheduling, but its implementation is more complex than PI