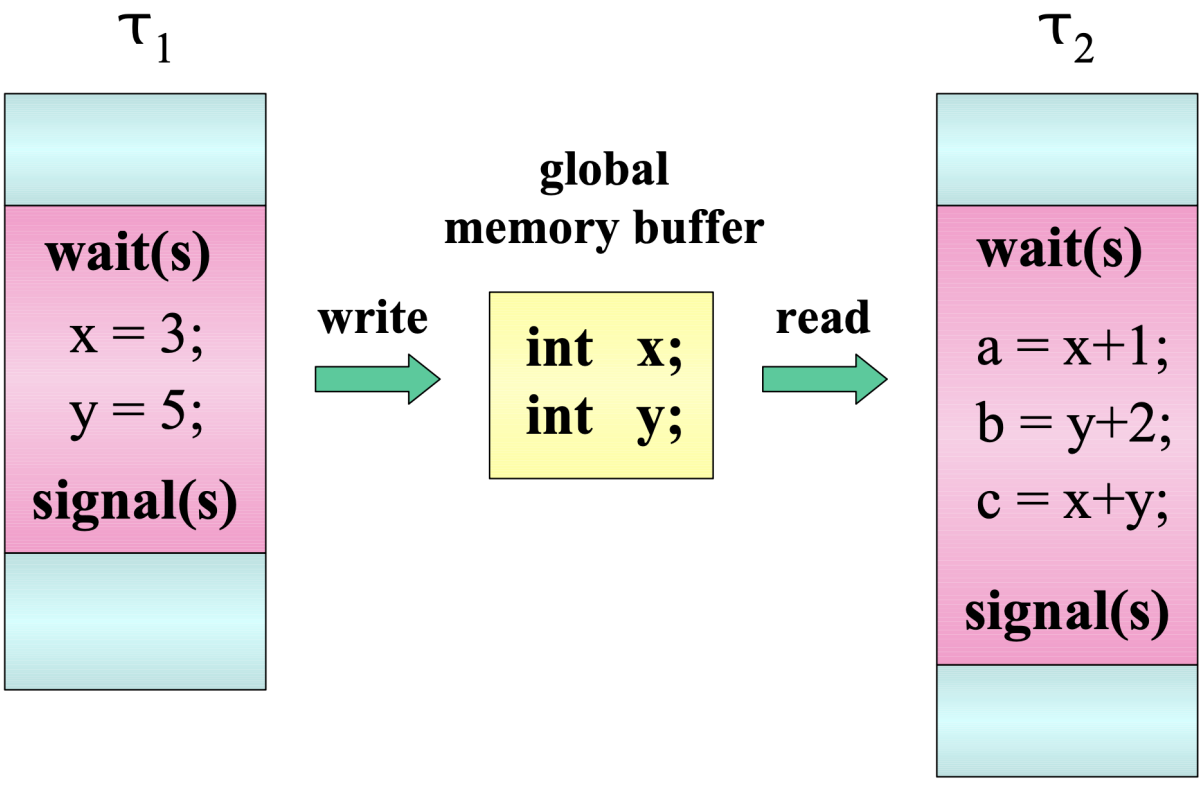
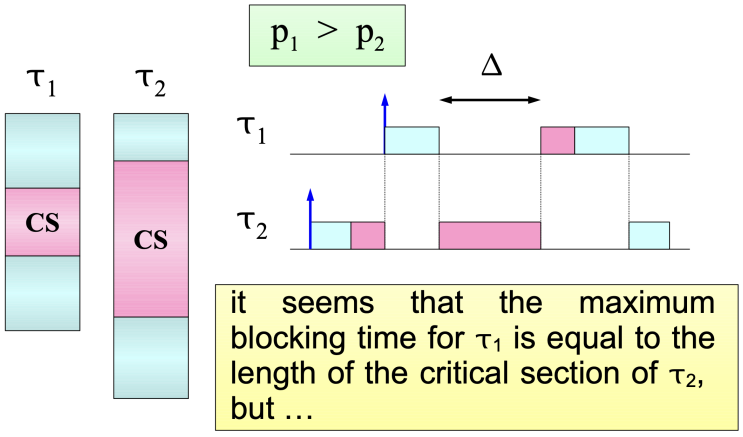
Resource access protocols

Problemi causati dalla mutua esclusione



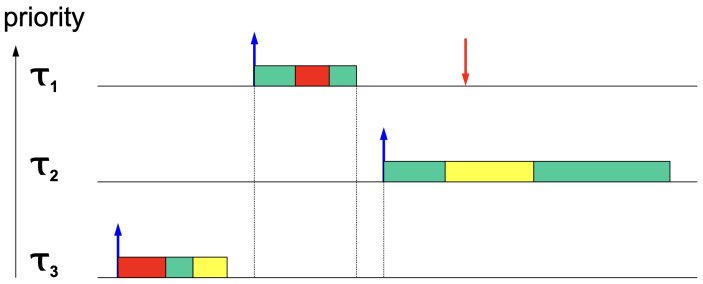


- τ_1 ha priorità più alta, ma esegue prima τ_2.
- τ_2 esegue ed arriva nella sua sezione critica, ma nel mentre viene eseguito τ_1 a cui viene data priorità.
- Arrivato alla wait della sezione critica τ_1 si blocca e si viene ri-preemptati nella sezione critica di τ_2. Appena termina, si torna alla sezione critica di τ_1 ed usciti da essa termina.
- Infine termina anche τ_2.

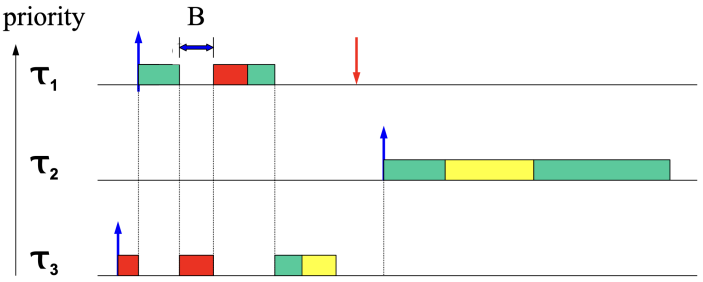
Sembra che il massimo tempo di bloccaggio corrisponde alla sezione critica di τ_2 ***MA*** in molti casi può essere molto peggio.

- Prima per calcolare il tempo di risposta di τ_1 prendevamo il suo computing time più interferenza dei task a più alta priorità, ma in questo caso l'interferenza è causata da task a priorità inferiore.

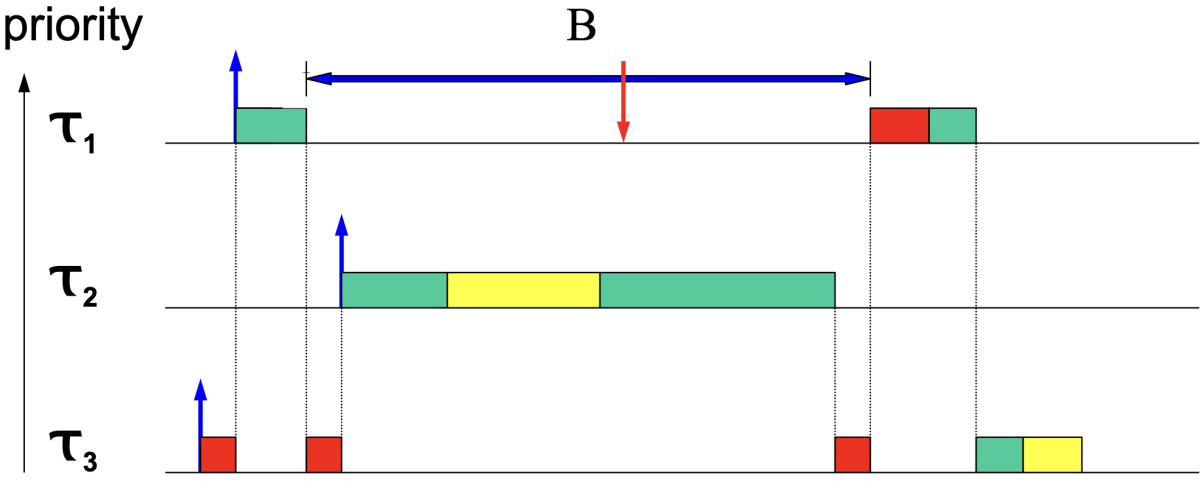
$R_i = C_i + I_i + B_i$?????



Caso senza conflitti



Primo task spostato più indietro



Secondo task spostato più indietro, caso peggiore

- In questo caso il tempo di bloccaggio corrisponde alla durata di tutto τ_2 (Problema del Mars Rover → reset continuo che non permetteva di inviare dati)

Priority Inversion → Un task ad alta priorità è bloccato uno o più task a bassa priorità per un tempo corrispondente a tutta la durata di uno dei task.

Per sistemare questo problema è necessario introdurre dei meccanismi di accesso alle risorse condivise andando ad arricchire le wait controllando quali sono i semafori attualmente bloccati e a seconda del check che fanno mi permettono di fare o no il lock.

I protocolli sono i seguenti: (valgono per single core)

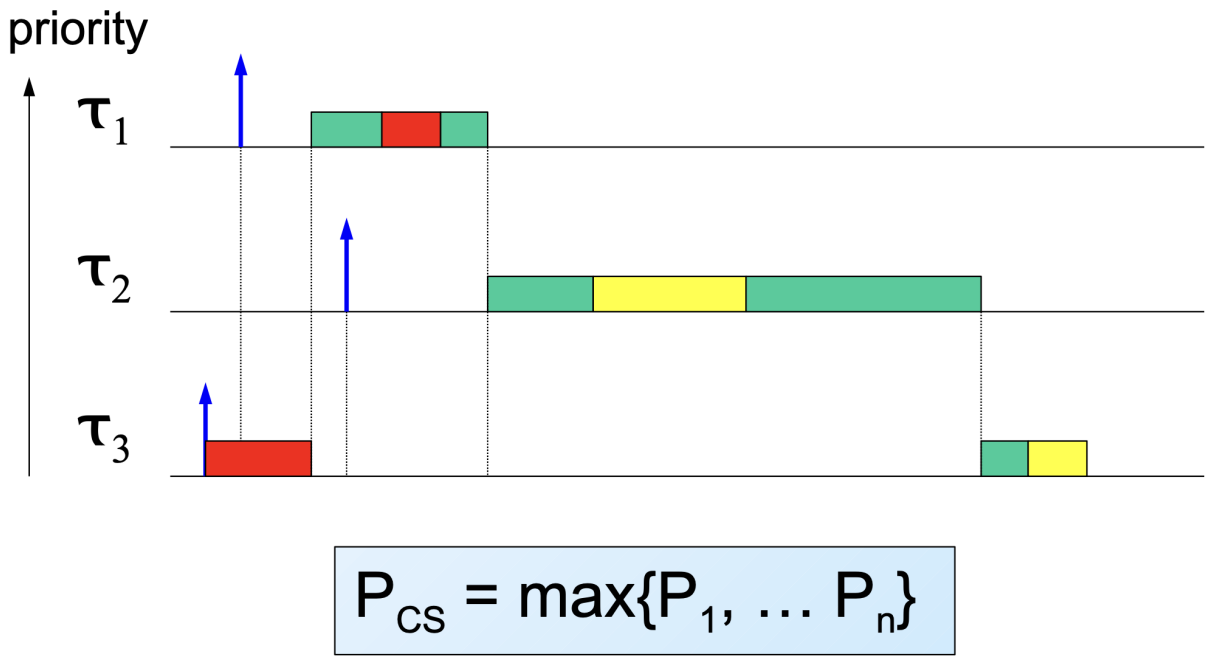
- **Non Preemptive Protocol (NPP)**
- **Highest Locker Priority (HLP)**
- **Priority Inheritance Protocol (PIP)**
- **Priority Ceiling Protocol (PCP)**
- **Stack Resource Policy (SRP)**

Non Preemtive Protocol

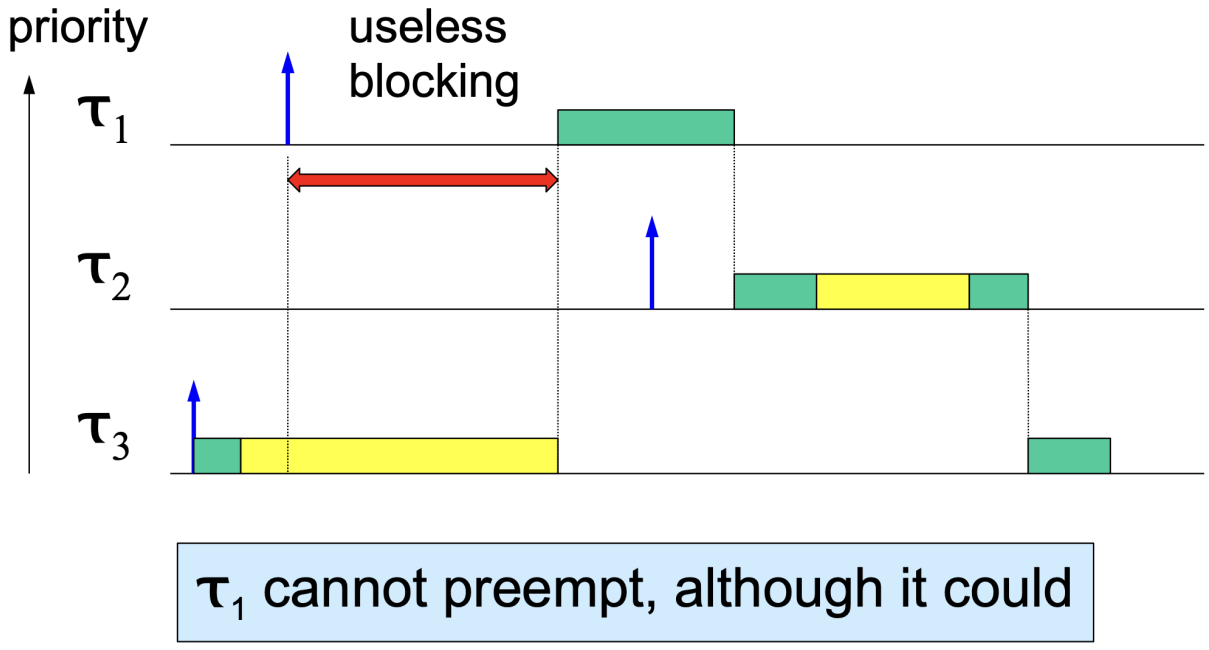
La preemtion è vietata durante una sezione critica!!!

Per fare ciò, la wait entra ed alza la priorità del task a quella massima di tutto il sistema, così fino alla signal non posso essere interrotto. Infine riabbasso la priorità.

E' molto semplice da implementare, abbiamo dei tempi di bloccaggio molto bassi ***MA*** i task ad alta priorità che non usano le sezioni critiche sono comunque bloccati per l'importo della sezione critica più grande tra tutti i task a bassa priorità.



Caso senza preemption



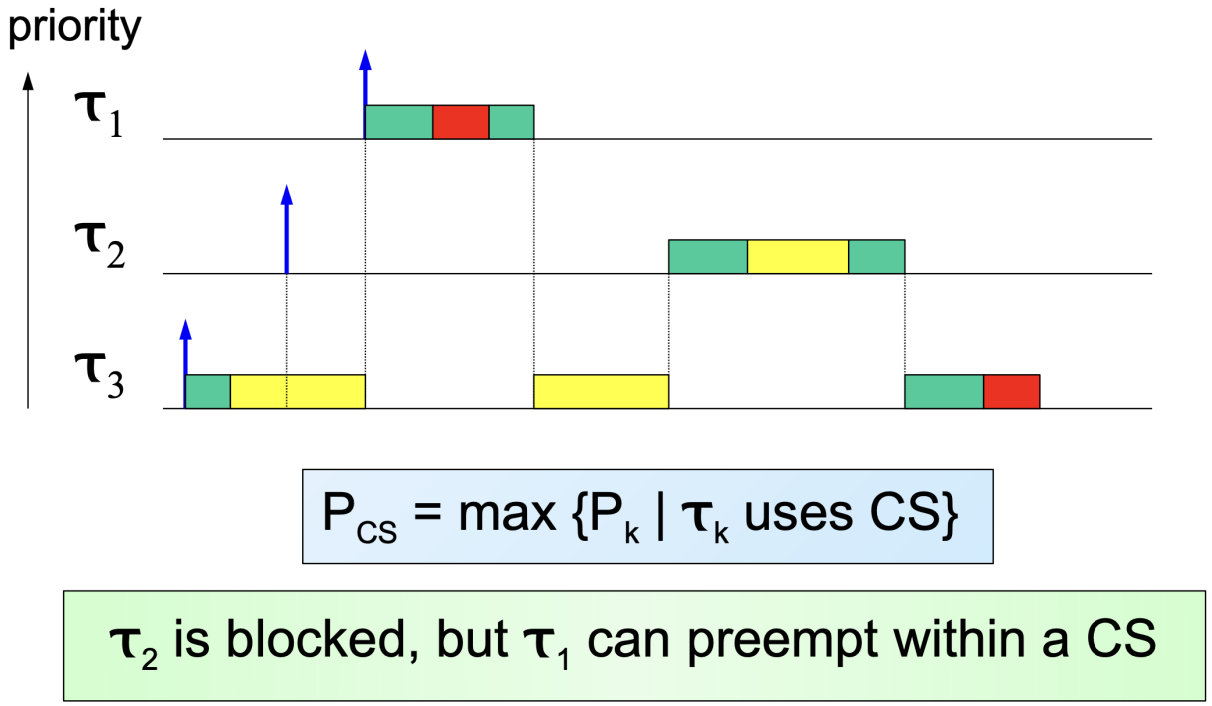
- Può causare un deadline miss di τ_1 a causa del bloccaggio.
- Potrei non alzare alla priorità più alta in assoluto, ma a quella più alta tra i task che possono bloccare quella risorsa → **Highest Locker Priority**

Highest Locker Priority

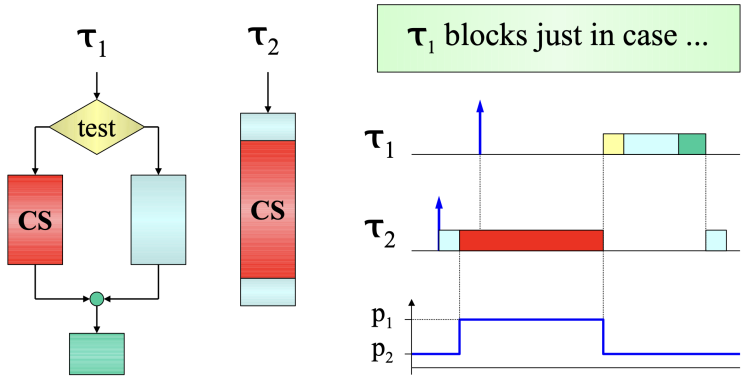
Alzo la priorità del task solo tra quelli che stanno usando quella risorsa.

Si implementa alzando la priorità solo alla più alta tra i task che richiedono quella risorsa, ma è necessario sapere a quali risorse accederanno i task in anticipo. Un task è bloccato quando cerca di preemptare, non quando entra nella sezione critica.

Viene chiamato anche **Immidiate Priority Ceiling** → priorità massima tra i task che possono usare una certa risorsa (detto Immediate perchè alza immediatamente la priorità). Se τ_1 accede a RA, τ_2 accede sia a RA e RB e τ_3 non accede a niente, il **priority ceiling** di RA è 1, di RB è 2. In fase di lockaggio della mia risorsa, alzo la mia priorità al ceiling della risorsa RA.



- Alzo la priorità della risorsa gialla a 2, ma nonostante ciò eseguo τ_1 perchè ha priorità maggiore.



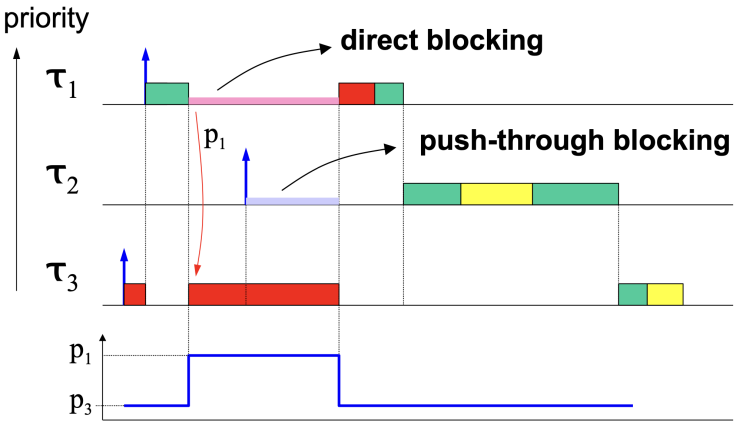
Problemi di **Highest Locker Priority**

- Non è detto che le wait e signal accadano sempre, nel senso che potrebbero avvenire solo in certe condizioni in cui si verificano IF o WHILE.
- Voglio che il task si blocchi solo se effettivamente ci entra nella sezione critica. Risolvo con **Priority Inheritance Protocol**

Priority Inheritance Protocol

Un task su una sezione critica aumenta la sua priorità solo se effettivamente sta bloccando altri task (si dice che blocca **solo quando necessario**), ad esempio nel caso precedente non alzo la priorità perchè non sta bloccando nessuno.

La priorità viene aumentata alla priorità del task che sta bloccando, non applico il Priority Ceiling.



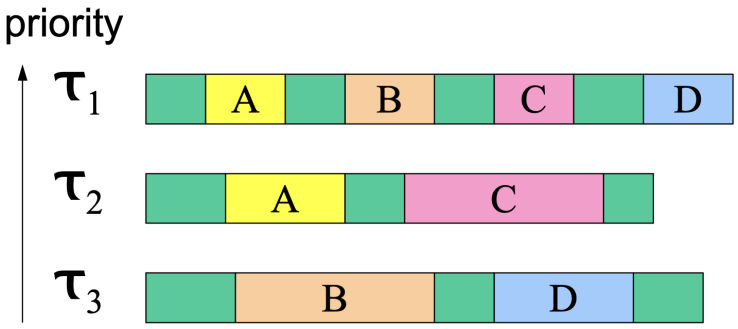
- τ_3 parte ed eseguo la risorsa rossa. Viene preemptata da τ_1 e quando quest'ultimo fa la wait si ridà il controllo della risorsa rossa a τ_3 (**direct blocking →** task bloccato da un semaforo) che stavolta esegue con priorità uguale a τ_1, così che se arrivasse un task Medium Priority esso non blocchi l'esecuzione della sezione critica (**push through blocking →** task bloccato da un task a più bassa priorità che ha ereditato una priorità più alta).

Qual'è il bloccaggio massimo per ogni task τ_i a causa di altri task a bassa priorità?

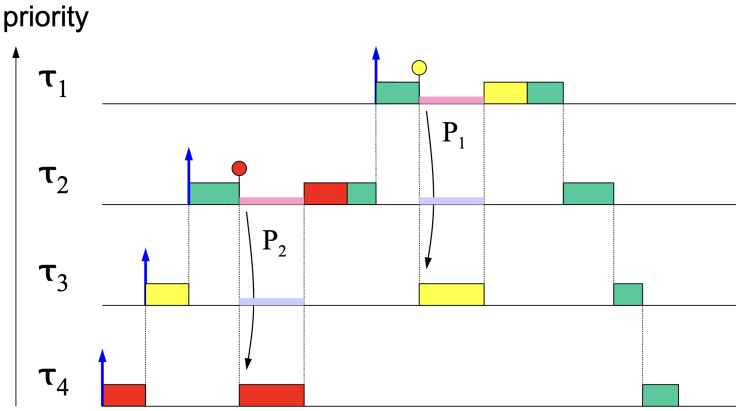
Avviene solo nel caso che i samafori siano presenti nei task a bassa priorità, sono loro che causano i bloccaggi.

Teorema 1: ogni task τ_i può essere bloccato al massimo una volta per ogni semaforo.

Teorema 2: ogni task τ_i può essere bloccato al massimo una volta per ogni task a più bassa priorità.



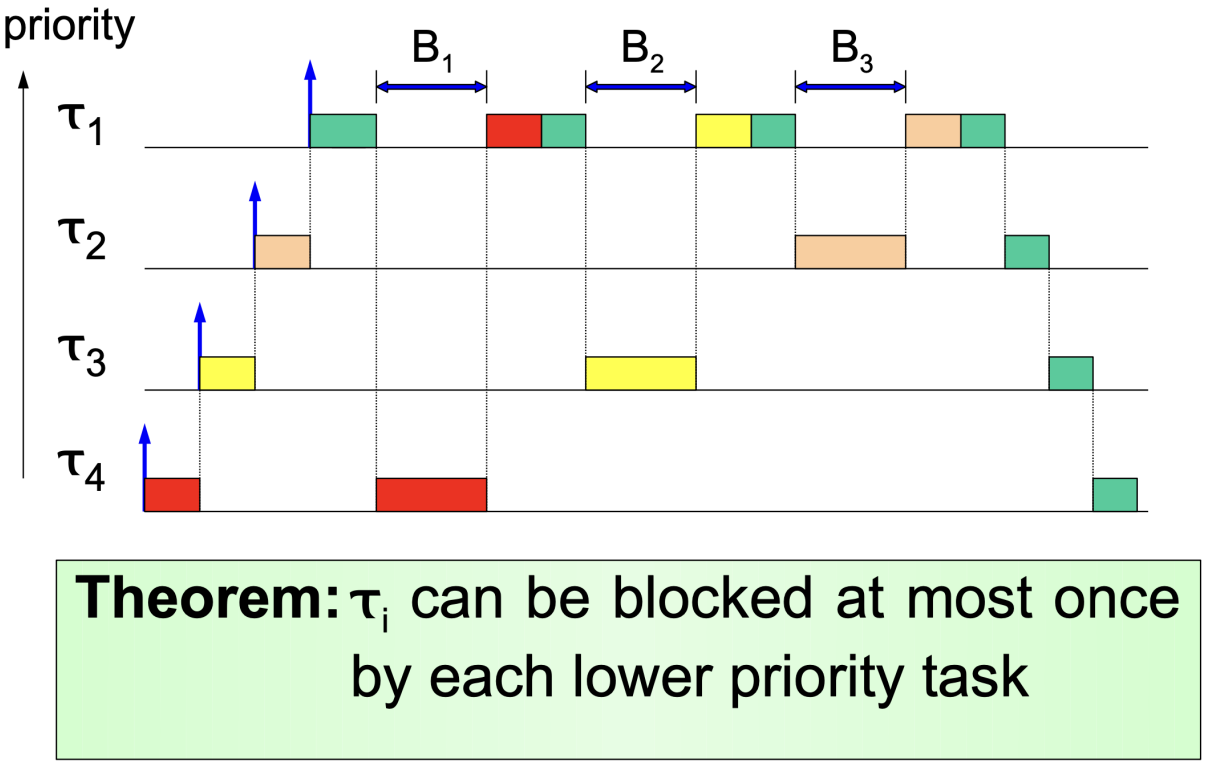
- τ_3 non può mai essere bloccato perchè sono acceduti solo da task ad alta priorità. Il bloccaggio per il task τ_3 è $B_3 = \emptyset$.
- τ_2 può ricevere interferenza da parte di A e C ma non si blocca, si può bloccare con B o D tramite push though blocking. Non si può bloccare su entrambi perchè al più può bloccarsi una volta sola per i task a più bassa priorità come da teorema. $B_2 = \max(B_{3B}, B_{3D})$
- τ_1 si può bloccare su A, B, C e D al massimo due volte (al più una per semaforo e al più una per task a più bassa priorità). $B_1 = \max(B_{2A}, B_{2C}) + \max(B_{3B}, B_{3D})$



- Il pallino indica l'inizio di una sezione critica.
- τ_3 al momento della wait di τ_2 verso la risorsa rossa è bloccato indirettamente, dato che la risorsa rossa in τ_4 assume la priorità 2.
- Stessa cosa accade a τ_2 con la risorsa gialla che viene eseguita con priorità 1 da τ_3.

Vantaggi di PIP → viene detto esplicitamente a quale semaforo accedere (presente anche in Linux, tutto risolto a livello kernel). Limita il Priority Inversion del Mars Rover.

Svantaggi di PIP → ha un problema di chain blocking e per sezioni critiche nested può incorrere in deadlock.



Chained Blocking con PIP

- In questo caso τ_1 viene bloccato 3 volte, una per ogni sezione critica.
- Il problema viene detto CHAIN BLOCKING, ovvero un bloccaggio a catena con conseguenti preemption che ci danno fastidio in termini di overhead → Per risolvere questo problema si usa **Priority Ceiling Protocol.**

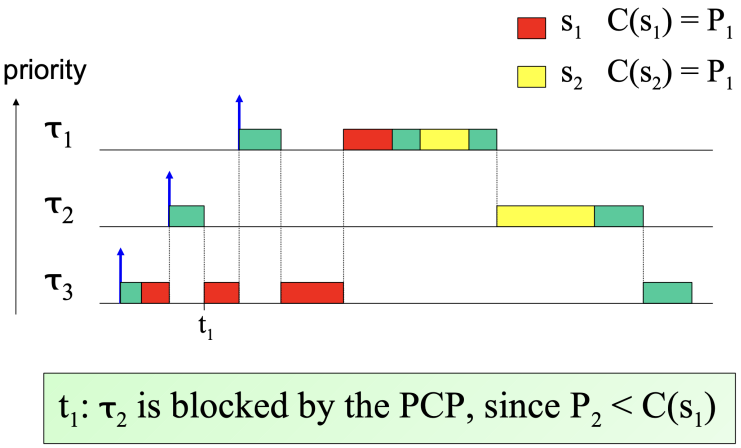
Priority Ceiling Protocol

Identito al PIP + un test in fase di accesso. Un task può entrare in una sezione critica solo se la risorsa è libera e non c'è rischio di bloccaggio a catena.

Devo controllare che tutte le risorse di cui τ_i potrebbe aver bisogno dopo sono libere (devono essere tutte libere se volessi anche solo accedere la prima).

Un task τ_i può entrare in una sezione critica solo se la sua priorità sia più alta del ceiling più alto tra tutte le risorse locked e se il semaforo è libero.

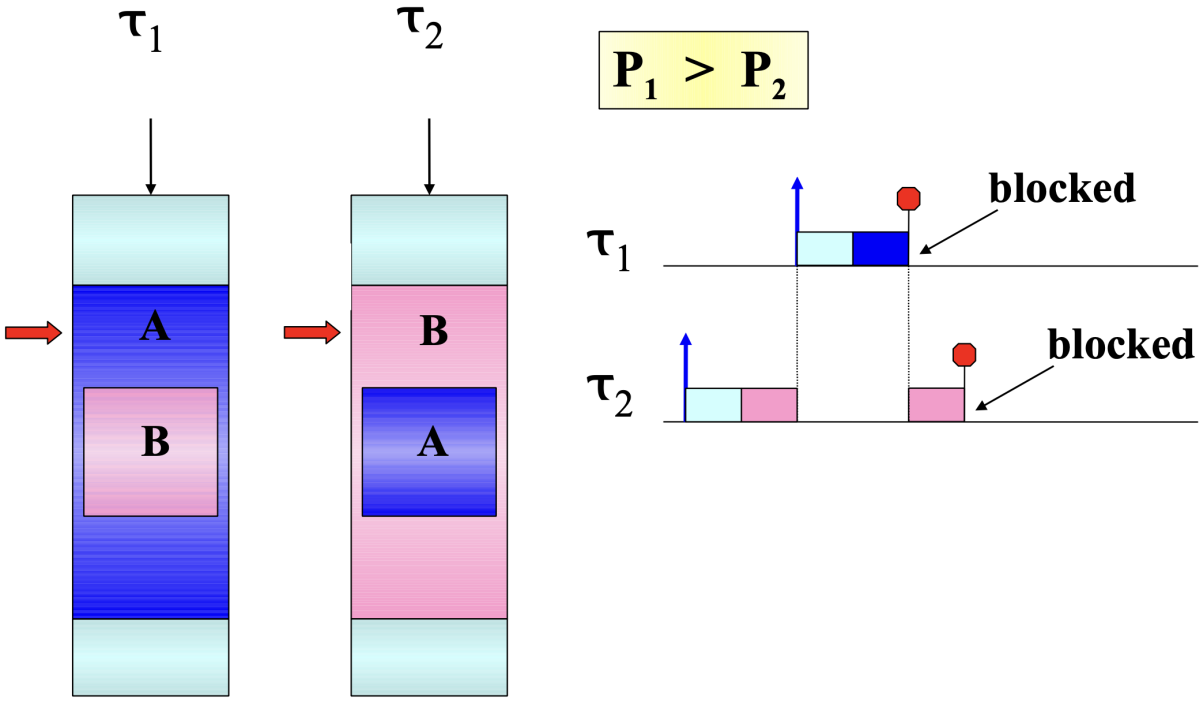
Non permetto di entrare in S_x se c'è qualche risorsa bloccata con un ceiling più alto di me, anche se quella risorsa non la utilizzerò mai.



- τ_1 usa la risorsa rossa e gialla, τ_2 la gialla e τ_3 la rossa.
- Tutti i ceiling della rossa e gialla sono alla massima priorità 1.

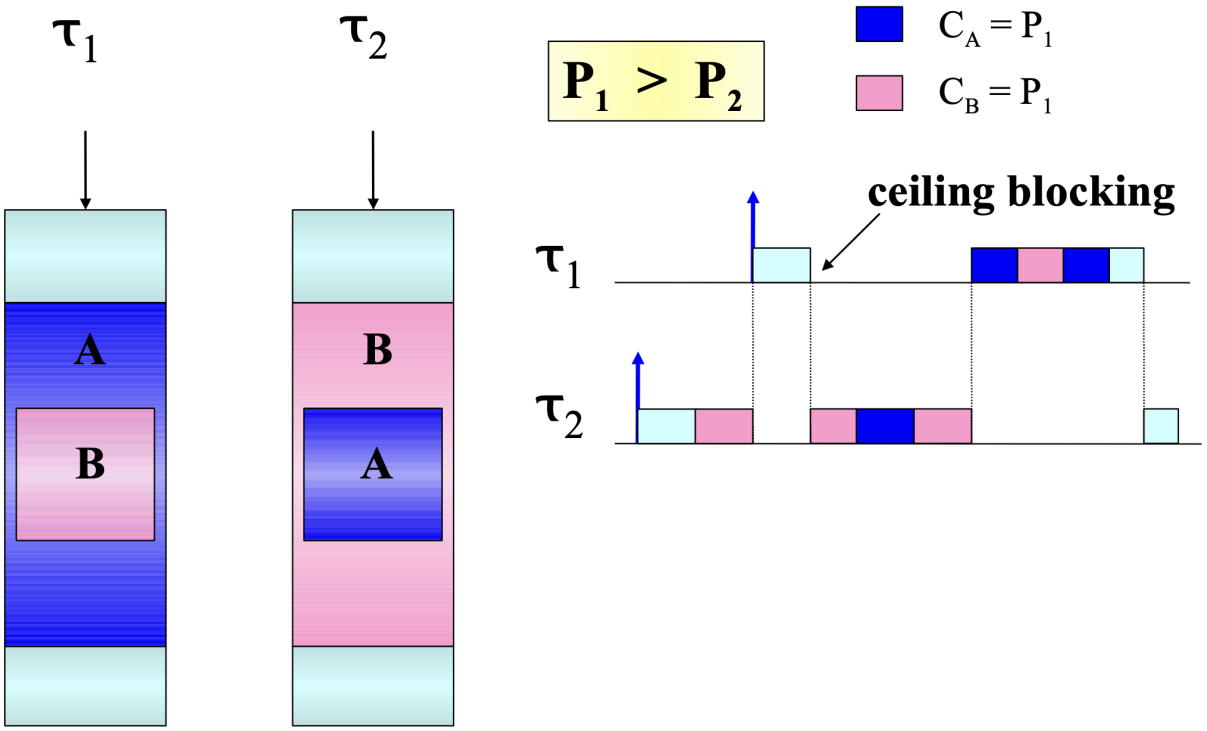
Vantaggi di PCP → il bloccaggio è ridotto ad una sola sezione critica tra i task a più bassa priorità aventi ceiling maggiore di me e previene i deadlocks.

Svantaggi di PCP → è necessario stabilire i ceiling di ogni semaforo, non è detto che lo sappia in caso uso librerie o binari che non conosco.



Problema del programmatore sbadato senza PCP

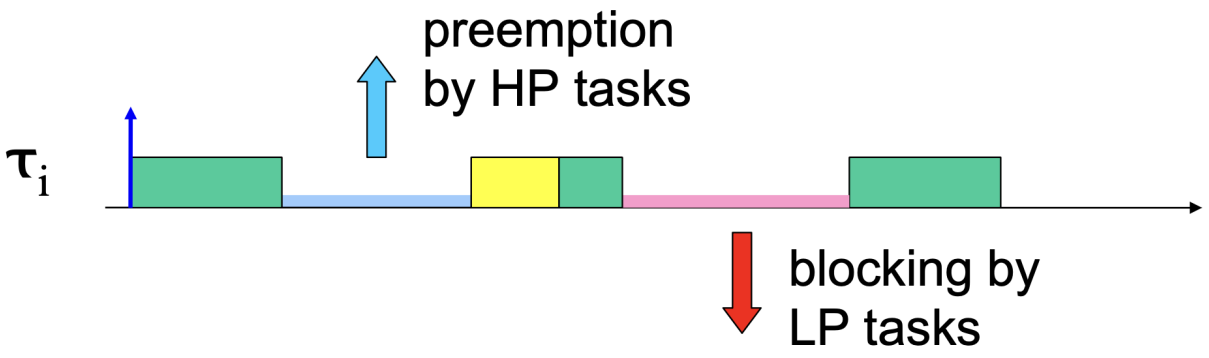
- Parte τ_2 ed esegue quasi fino alla fine di B e viene preemptato da τ_1.
- τ_1 esegue fino alla fine della risorsa blu e chiede la rosa che è ancora bloccata.
- In τ_2 torna in esecuzione la risorsa rosa ma quando termina mi chiede la risorsa blu → deadlock



Problema del programmatore sbadato usando PCP

- Sia il ceiling di A che di B sono al livello di τ_1 a priorità massima.
- Parte τ_2, puo lockare? **La risorsa è libera?** Si. **La mia priorità è più alta del ceiling delle risorse correntemente lockate?** Nessuna risorsa è bloccata quindi posso entrare. (Mi faccio sempre queste domande per sapere se posso accedere ad una certa risorsa!!!)
- τ_1 può entrare nella risorsa blu? E libera ma la priorità dell'unica risorsa lockata, ovvero la rosa, è allo stesso livello, quindi non posso accedervi → **CEILING BLOCKING**
- Mi bocco al più una volta sola.

Come calcolo il tempo di bloccaggio B_i per ogni task?



$$\forall\ i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leqslant i\left(2^{1/i} - 1\right)$$

Nuovo test con RM

- Vecchio test senza bloccaggio: somma delle utilizzazioni dei task a più alta o uguale priorità di me minore o uguale di U_{lub} → $\sum_{hep} U_i \leq U_{lub}$
- Questo test non è più valido → è necessario applicare la formula nella box verde qui sopra.

$$\forall\ i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leqslant 1$$

Nuovo test con EDF

In caso di **Response Time Analysis RTA** → $R_i = I_i + C_i + B_i$

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

In caso di **Demand based test** → $g(0, L) = \sum_{i=1}^{n} \left[\frac{L - D_i + T_i}{T_i}\right] C_i + B(L)$

Esercizi per provare a calcolare il bloccaggio con Priority Inheritance e con Priority Ceiling:

	R1	R2	R3	B_{PIP}	B_{PCP}
τ 1		20		5	5
τ 2	5		10	20	10
τ 3		5	5	15	10
τ 4			5	10	10
τ 5	10	3			

Calcolo bloccaggi con Priority Inheritance:

- τ_5 ha come tempi di bloccaggio 0 perchè non ha nessuno a priorità più bassa.
- τ_4 può bloccarsi per push through per il peggiore tra 10 e 3, ovvero 10.
- τ_3 può bloccarsi per 15, ovvero 10 + 5 (scelto 10, blocco la riga τ_5 e la colonna R_1 perchè su una risorsa posso bloccarmi una sola volta).
- τ_2 si può bloccare su tutte e tre le risorse, al più una per ogni task e al più una per ogni risorsa. Scelgo quindi 10 + 5 + 5, ovvero 20.
- τ_1 si può bloccare solo sulla risorsa R_2 per un tempo 5.

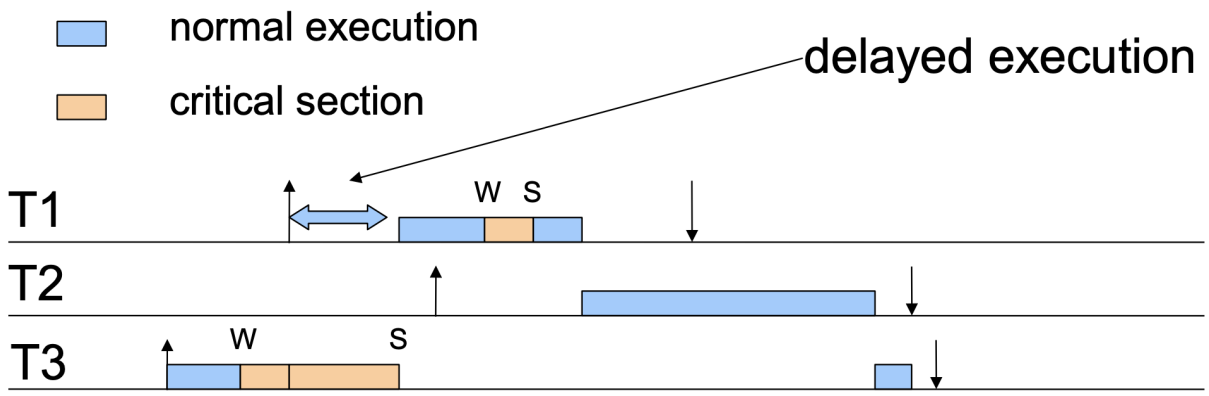
Calcolo bloccaggi con Priority Ceiling:

- Non ho catene di bloccaggio, quindi vado a scegliere il tempo maggiore.

In generale con PIP si ha un tempo di bloccaggio più alto a causa del chain blocking → somma tra sezioni critiche. Problemi in caso di sezioni critiche nested, quindi PIP va bene in caso di sezioni critiche semplici ma ha un implementazione costosa dato l'innalzamento delle priorità nel mentre di una sezione critica.

Stack Resource Policy (SRP)

Uguale all'Immediate Priority Ceiling/Highest Locker Priority, effettua un bloccaggio che avviene ancora prima di iniziare l'esecuzione (non con la wait) → una volta che parto non mi blocco più. Ciò causa un grande vantaggio in termini di overhead.



Assegno ad ogni task una priorità p ed un livello di preemption π (sia su EDF che RM assegno π in maniera inversamente proporzionale ai periodi → $\pi_i = \frac{1}{T_i}$). Il preemtion level sarà più alto ai task che hanno periodo più piccolo.

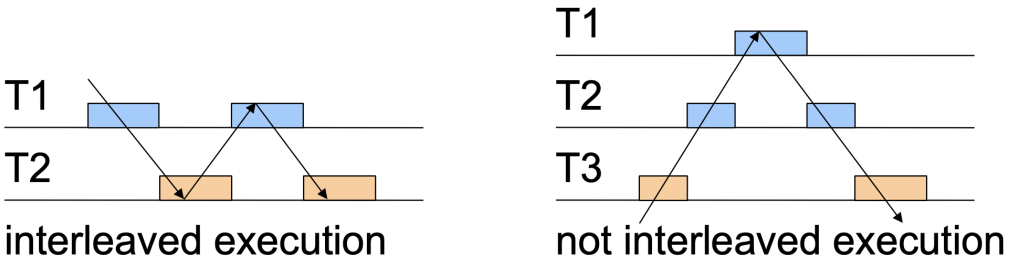
Ad ogni risorsa viene assegnato un **ceiling statico** definito come il preemption level più alto di tutti tra i task che usano quella particolare risorsa.

Inolte definiamo un **ceiling di sistema dinamico** $\prod_s(t)$ definito come il massimo tra tutti i ceiling delle risorse correntemente occupate.

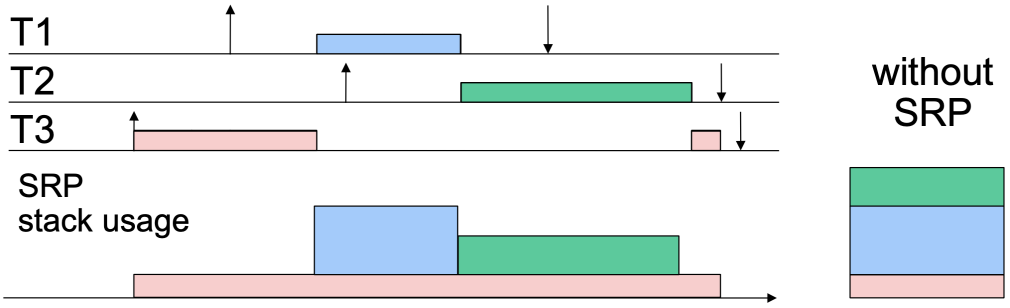
Ad un task a più alta priorità è consentito iniziare la sua esecuzione solo se il suo preemption level è più grande del ceiling di sistema → $\pi > \prod_s(t)$

Aiuta a prevenire i deadlock, le preemtion non necessarie sono evitate e il tempo di bloccaggio B_i è uguale a quello che avevamo trovato con PCP.

Con SRP posso utilizzare lo stesso stack per tutti i task che ho nel sistema.



- In generale lo stack può essere condiviso tra task se possiamo garantire con non saranno mai interlacciati
- E' meglio lo schema a destra in quanto quando scendo di priorità è perchè il task ha finito di eseguire.



- τ_3 mette roba sullo stack, viene preemptato da task a più alta priorità τ_1. Quest'ultimo dopo aver aggiunto dati sullo stack finisce e quindi lo stack decresce dando il controllo a τ_2. Una volta finito lo stack decresce senza intoppi. Senza SRP avrei avuto uno stack dove ogni risorsa è impilata, perciò per deallocare τ_1 sarei andato in conflitto con τ_2.