Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2018-2019

Pietro Frasca

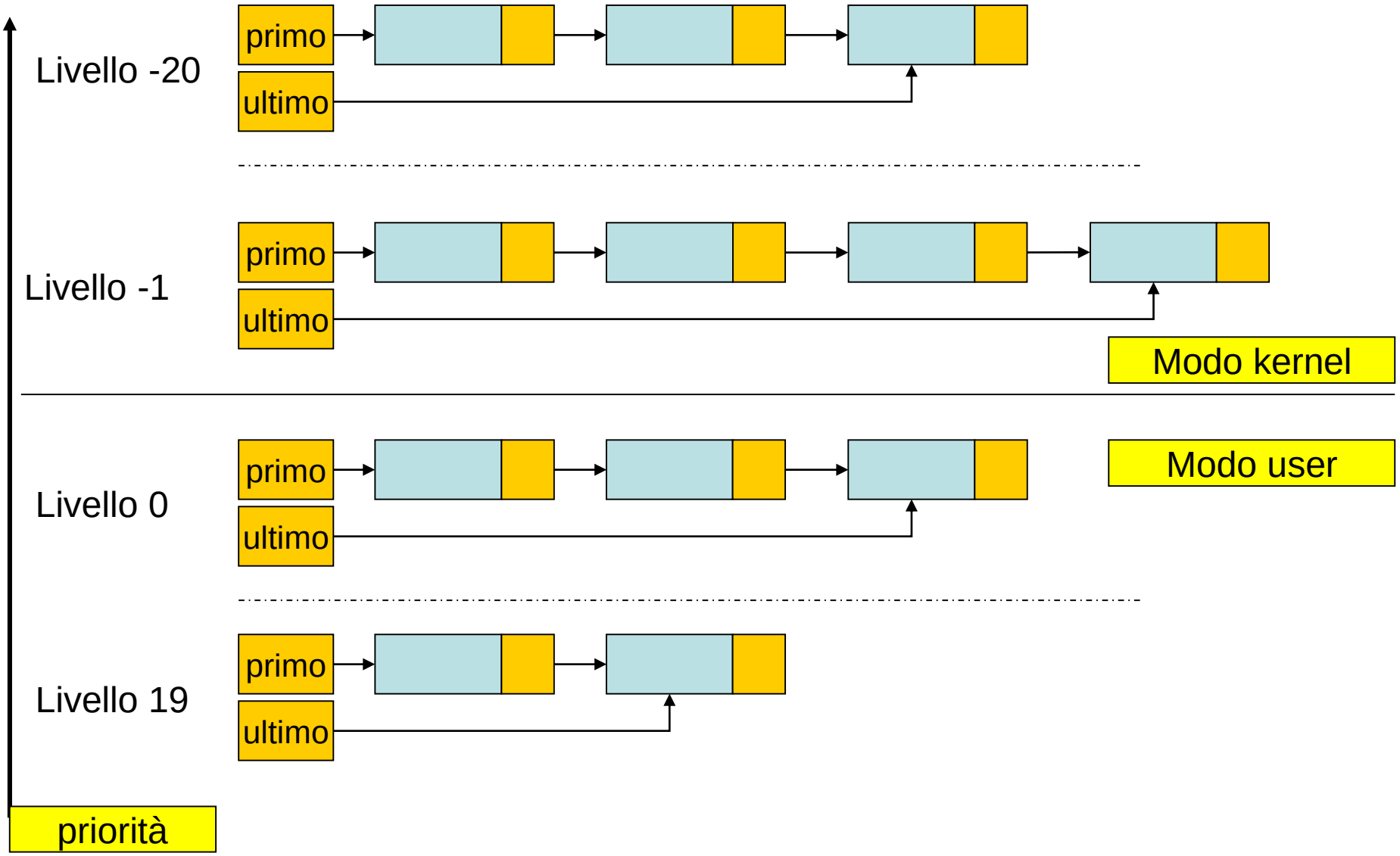
Lezione 14

Giovedì 22-11-2018

Scheduling in UNIX

- Poiché UNIX è un sistema multiutente e multitasking, l'algoritmo di scheduling della CPU è stato progettato per fornire buoni tempi di risposta ai **processi interattivi**.
- I thread sono generalmente a livello di kernel, pertanto lo scheduler si basa sui thread e non sui processi.
- E' un algoritmo a **due livelli** di scheduler.
- Lo **scheduler a breve termine** sceglie dalla coda dei processi pronti il prossimo processo/thread da eseguire.
- Lo **scheduler a medio termine (swapper)** sposta pagine di processi tra la memoria e il disco (area swap o file di paging) in modo che tutti i processi abbiano la possibilità di essere eseguiti.
- Ogni versione di UNIX ha uno scheduler a breve termine leggermente diverso, ma tutti seguono uno schema di funzionamento basato su code di priorità.

- In Unix le priorità dei processi eseguiti in **modalità utente** sono espresse con valori interi positivi mentre le priorità dei processi eseguiti in **modalità kernel** (che eseguono le chiamate di sistema) sono espresse con valori interi negativi.
- I valori **negativi** rappresentano **priorità maggiori**, rispetto ai valori positivi che hanno priorità minore.
- Lo scheduler a breve termine sceglie un processo dalla coda con priorità più alta. Le code sono gestite in modalità **RR**. Il quanto di tempo assegnato al processo per l'esecuzione dura generalmente **20-100 ms**.
- Un processo è posto in fondo alla coda, quando termina il suo quanto di tempo.



scheduling in unix

- I valori delle **priorità sono dinamici** e sono ricalcolati **ogni secondo** in base ad una relazione che dipende dai seguenti parametri:

  - **valore iniziale (base)**
  - **uso_cpu**
  - **nice**

  priorità = f(base, uso_cpu, nice)

- Ogni processo è poi inserito in una coda, come mostrato nella figura precedente, in base alla nuova priorità.
- **Uso_cpu** rappresenta, l'uso medio della cpu da parte del processo durante gli ultimi secondi precedenti. Questo parametro è un campo del descrittore del processo (PCB).

- L’incremento del valore del parametro **uso_cpu** provoca lo spostamento del processo in una coda a priorità più bassa.
- Il valore di **uso_cpu** varia nel tempo in base a varie strategie usate nelle varie versioni di UNIX.
- Ogni processo ha, inoltre, un valore del parametro **nice** associato. Il valore di base è 0 e l’intervallo di valori possibili è compreso tra –20 e +19. Ad esempio, con il comando **nice** (che utilizza l’omonima chiamata di sistema **nice**), un utente può assegnare ad un proprio processo un valore *nice* compreso tra 0 e 19. Soltanto il **superuser** (root) può assegnare i valori di *nice* compresi tra –1 e –20 ad un processo.
- Esempio:

Carattere per esecuzione in background

nice –n 10 mio_calcolo &

esegue il programma **mio_calcolo** in background assegnando al processo un valore nice pari a 10.

- Per quanto riguarda lo scheduling per le estensioni real-time, lo standard P1003.4 di UNIX, cui aderisce anche Linux, prevedono le seguenti classi di thread:
  - **Real-time FIFO;**
  - **Real time round-roubin;**
  - **Timesharing**
- I thread real-time FIFO hanno la priorità massima. A questi thread può essere revocata la cpu solo da thread della stessa classe con più alta priorità.
- I thread real-time RR sono simili ai real-time FIFO, ma viene loro revocata la CPU allo scadere del proprio quanto di tempo, il cui valore dipende dalla priorità.
- Le due classi di thread sono **soft real-time**.
- I thread real-time hanno livelli di priorità da 0 a 99, dove 0 è il livello di priorità più alto.

- I thread standard, non real-time, hanno livelli di priorità compresi tra 100 e 139. In totale, quindi si hanno 140 livelli di priorità.
- Il quanto di tempo è misurato in numero di scatti di clock. Lo scatto di clock è detto **jiffy** e dura 1 ms.

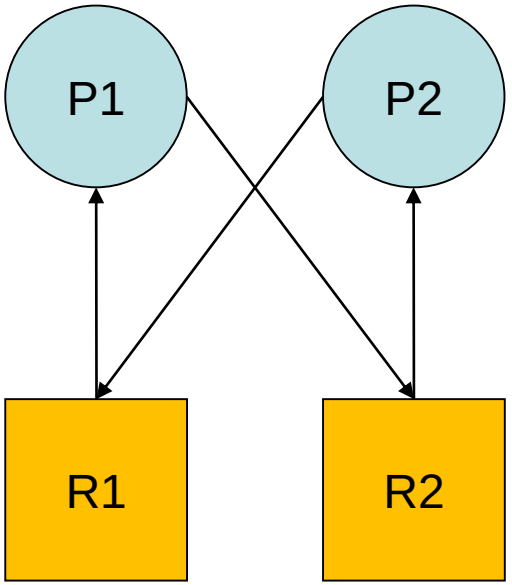
Blocco critico (stallo)

- In un sistema multiprogrammato, durante l'esecuzione, un processo può richiedere risorse condivise per svolgere la sua attività.
- In un determinato istante, la risorsa richiesta potrebbe essere non disponibile perché già allocata in precedenza a un altro processo. In questo caso il processo richiedente passa nello stato di bloccato.
- La situazione di **stallo** (**deadlock**) si può verificare tra due o più processi quando ciascuno dei processi possiede almeno una risorsa e ne richiede altre. Il processo richiedente non può ottenere la risorsa poiché la risorsa richiesta è stata già assegnata ad un altro processo che non la rilascia in quanto è in attesa di un'altra risorsa che è già allocata ad un altro processo ancora.

- Un gruppo di processi è in uno stato di deadlock quando ogni processo è in attesa di un evento che può essere causato solo da un altro processo appartenente al gruppo.
- In un normale funzionamento, un processo può utilizzare una risorsa seguendo la seguente sequenza:

1. **Richiesta**. Il processo richiede la risorsa. Se la richiesta non può essere concessa immediatamente (ad esempio, se la risorsa è utilizzata da un altro processo), allora il processo richiedente deve attendere finché può acquisire la risorsa.
2. **Uso**. Il processo esegue operazioni sulla risorsa
3. **Rilascio**. Il processo rilascia la risorsa.

- La richiesta e il rilascio delle risorse possono essere chiamate di sistema. Esempi di chiamate di sistema per la richiesta e il rilascio sono:
  - open () e close ( ) per file e dispositivi;
  - malloc( ) e free ( ) per la memoria.
- Analogamente, come abbiamo visto , le operazioni di richiesta e di rilascio con i semafori possono essere realizzate mediante le operazioni wait () e signal ().



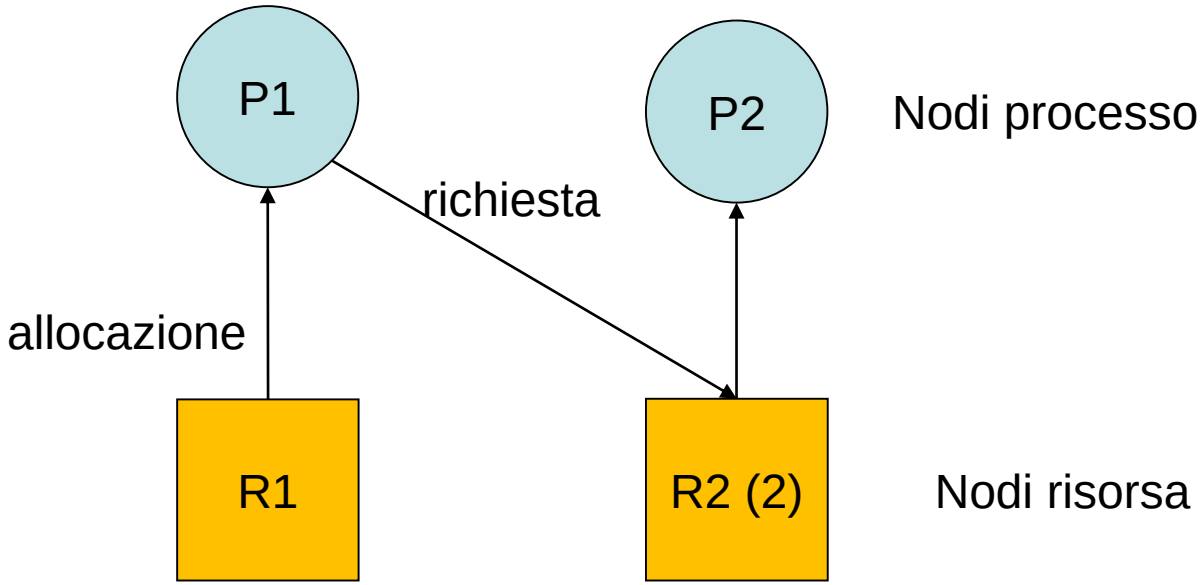
Rappresentazioni dello stato di allocazione delle risorse

- Per stabilire se un certo numero di processi è in stallo è necessario analizzare le informazioni relative alle risorse condivise allocate ai processi e quelle relative alle richieste di risorse in attesa.
- Per rappresentare lo stato di allocazione di un sistema si utilizzano due tipi di rappresentazione:
  - **Modelli basati su grafo**
  - **Modelli basati su matrici**

Modelli basati su grafo

- I deadlock possono essere descritti con un ***grafo orientato***, detto ***grafo di allocazione delle risorse***. Questo grafo è costituito da un insieme di nodi *N* e un insieme di archi *A*.

- L'insieme di nodi *N* è suddiviso in due tipi diversi di nodi: $P = \{P_1, P_2, ..., P_n\}$, l'insieme costituito da tutti i processi attivi nel sistema, e $R = \{R_1, R_2, ..., R_m\}$, l'insieme costituito da tutti i tipi di **risorse condivise** nel sistema.



- Un arco orientato dal processo P_i alla risorsa di tipo R_j, indicato con $P_i \rightarrow R_j$, significa che il processo P_i ha richiesto un'unità di una risorsa della classe R_j ed è ora in attesa per quella risorsa.

- Un arco orientato dal tipo di risorsa R_j verso il processo P_i, espresso con $R_j \rightarrow P_i$, indica che un'unità di risorsa tipo R_j è stata allocata al processo P_i.
- Un arco $P_i \rightarrow R_j$ è detto ***arco di richiesta***; un arco orientato $R_j \rightarrow P_i$ è chiamato ***arco di assegnazione***.
- Graficamente, un processo P_i si rappresenta con un cerchio e ogni tipo di risorsa R_j con un rettangolo o un quadrato.
- Poiché un tipo di risorsa R_j può avere più di un'unità, si indica tale pluralità con un numero intero all'interno del rettangolo.
- Il semplice grafo di allocazione delle risorse mostrato nella figura precedente mostra l'allocazione descritta dai seguenti insiemi: $P = \{P_1, P_2\}$, $R = \{R_1, R_2\}$ e $A = \{ P_1 \rightarrow R_2, R_1 \rightarrow P_1, R_2 \rightarrow P_2\}$.

Modelli basati su matrici

- Con il modello basato su matrici, lo stato di allocazione del sistema è rappresentato dalle seguenti matrici:

	R1	R2	Rm
P1	0	1	2
P2	1	0	3
Pn	0	1	0

Risorse allocate

	R1	R2	Rm
P1	1	0	1
P2	0	1	2
Pn	0	0	1

Risorse richieste

R1	R2	Rm
10	12	9

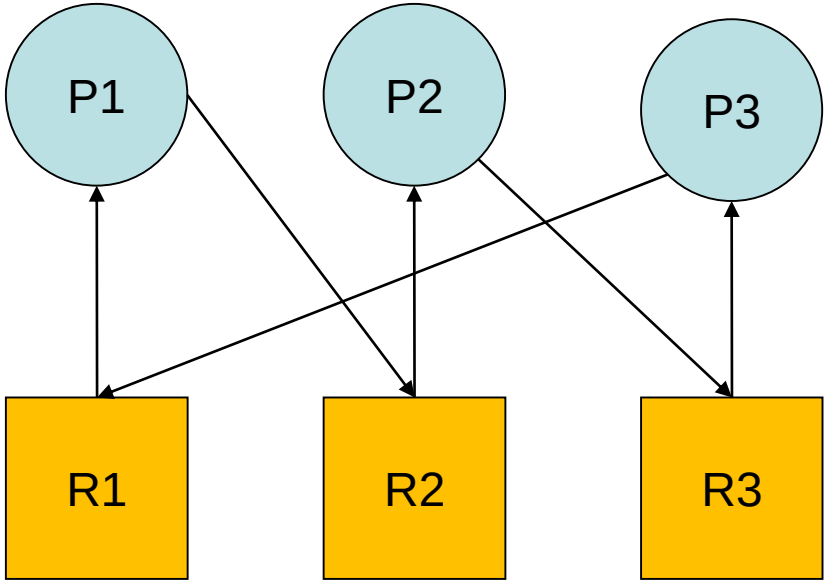
Risorse totali

R1	R2	Rm
0	3	2

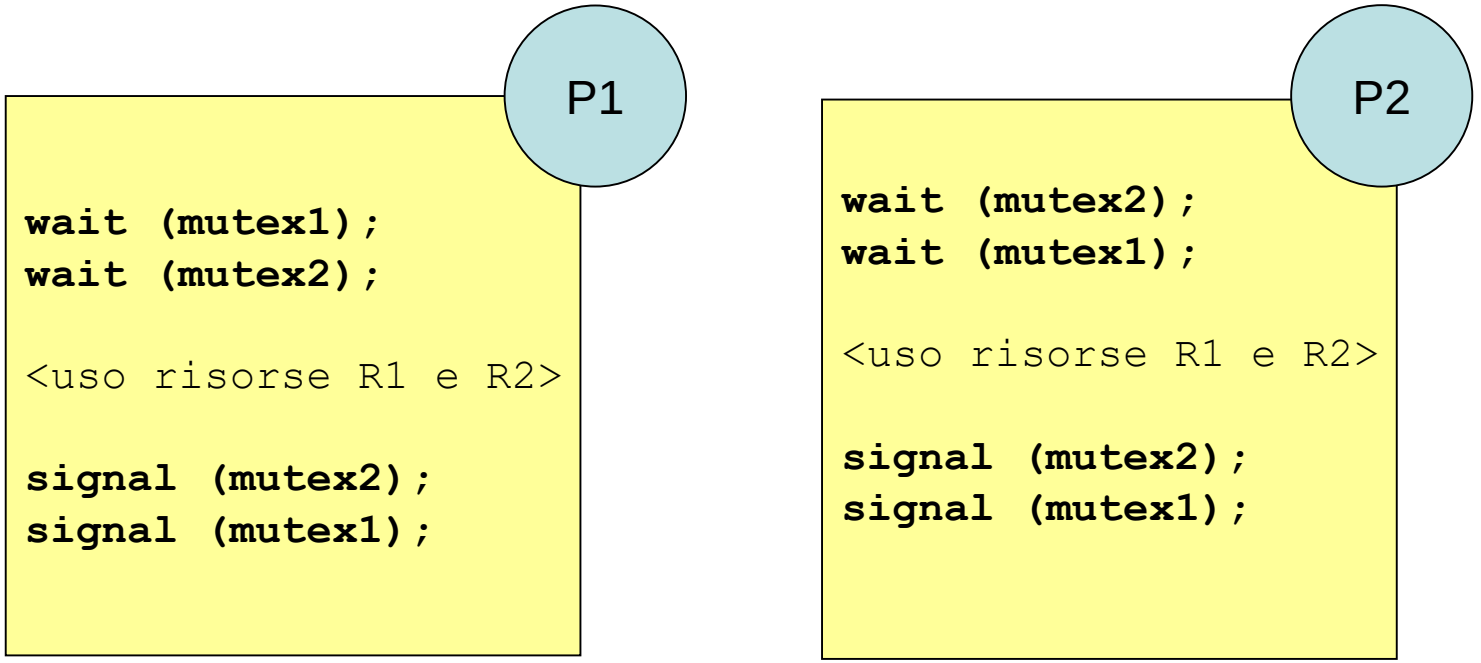
Risorse libere

Esempio di situazione di stallo

- Il seguente grafo di allocazione delle risorse mostra che ciascun processo non può continuare la propria esecuzione in quanto ciascuno è in attesa di una risorsa che è allocata da un altro processo bloccato.

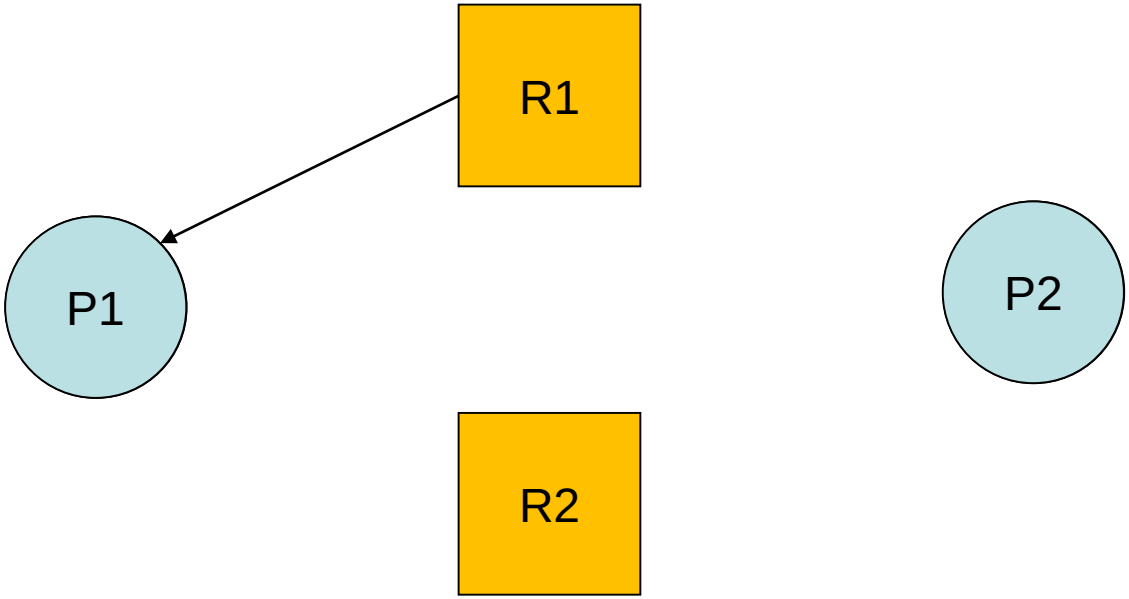


- In certi casi, la situazione di stallo dipende dalla ***velocità relativa*** di esecuzione dei processi.
- Consideriamo ad esempio il caso di due processi **P1** e **P2** che richiedono due risorse **R1** e **R2** nell'ordine mostrato nella figura seguente.



- Se i due processi P1 e P2 eseguono la seguente sequenza di operazioni nel tempo:

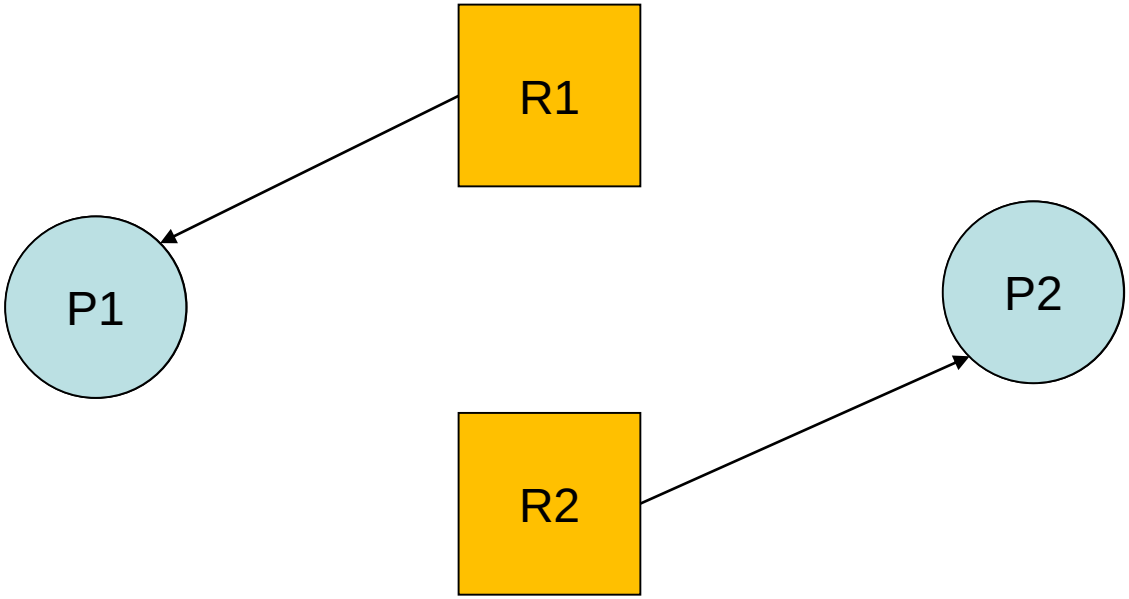
  T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)



- Se i due processi P1 e P2 eseguono la seguente sequenza di operazioni nel tempo:

  T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)

  T1: P2 esegue wait(mutex2) (acquisisce la risorsa R2)

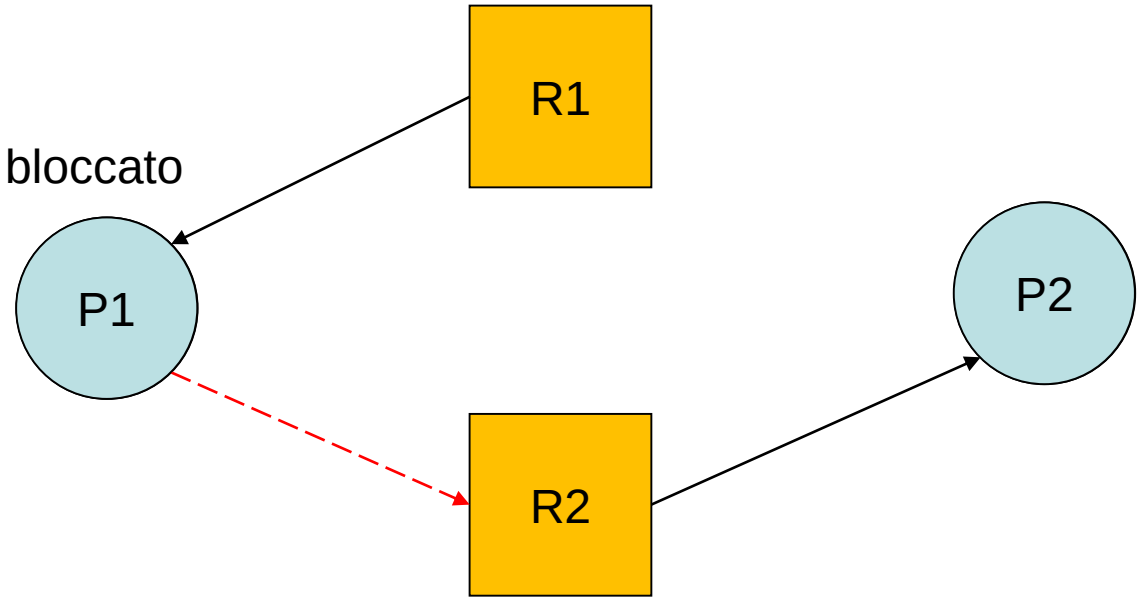


- Se i due processi P1 e P2 eseguono la seguente sequenza di operazioni nel tempo:

  T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)

  T1: P2 esegue wait(mutex2) (acquisisce la risorsa R2)

  T2: P1 esegue wait(mutex2) (P1 si blocca)



- Se i due processi P1 e P2 eseguono la seguente sequenza di operazioni nel tempo:

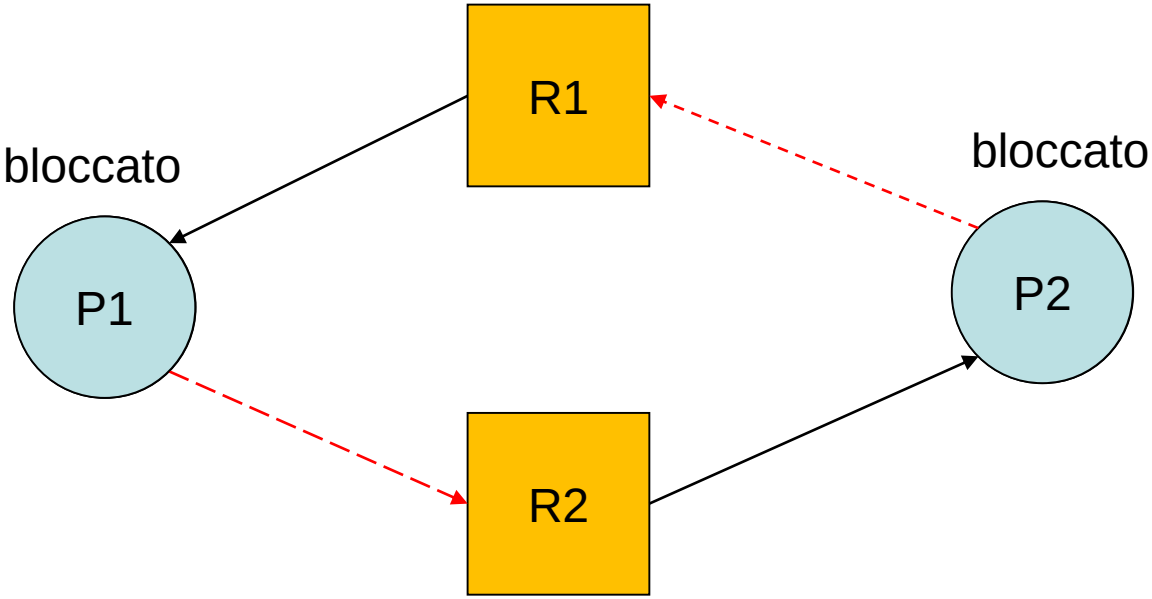
  T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)

  T1: P2 esegue wait(mutex2) (acquisisce la risorsa R2)

  T2: P1 esegue wait(mutex2) (P1 si blocca)

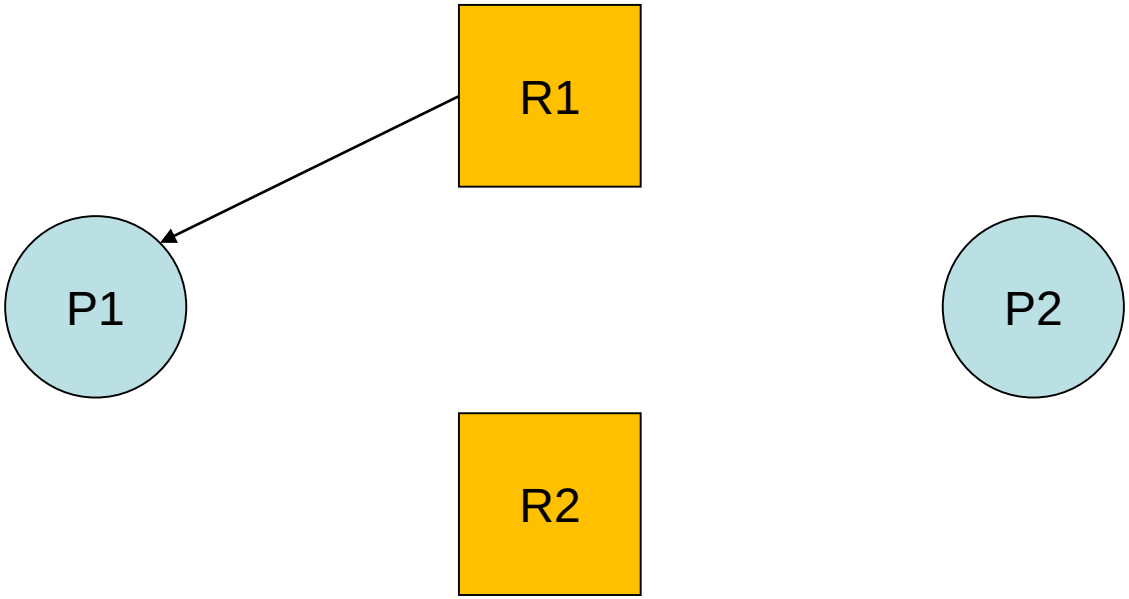
  T3: P2 esegue wait(mutex1) (P2 si blocca)

  i due processi P1 e P2 si bloccano rispettivamente sui semafori mutex2 e mutex1 e non possono uscire dalla situazione di stallo.



- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

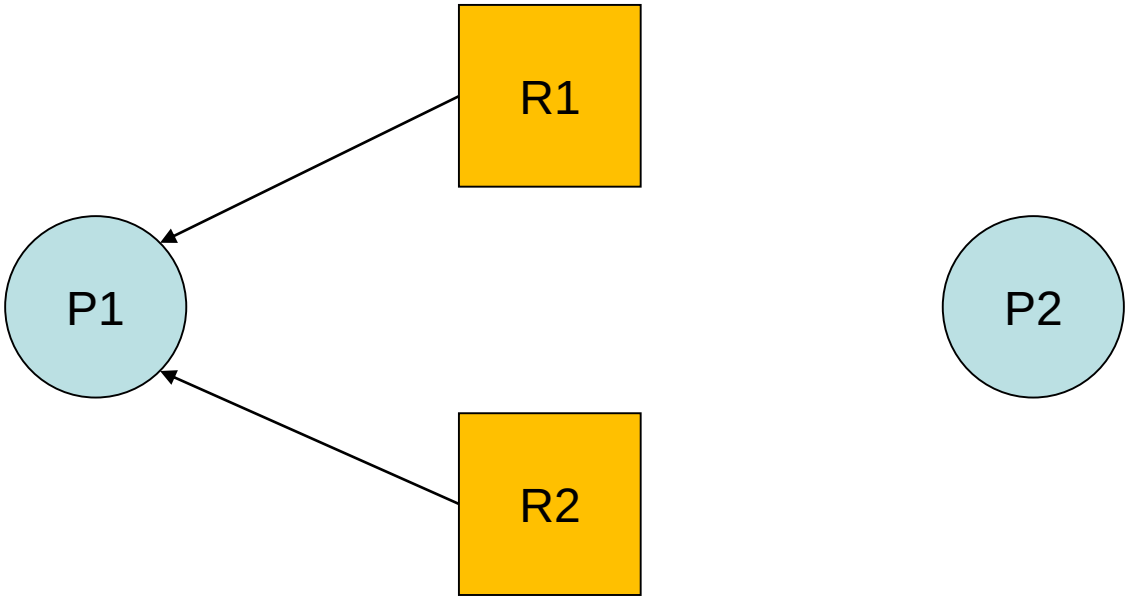
  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)



- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)

  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)

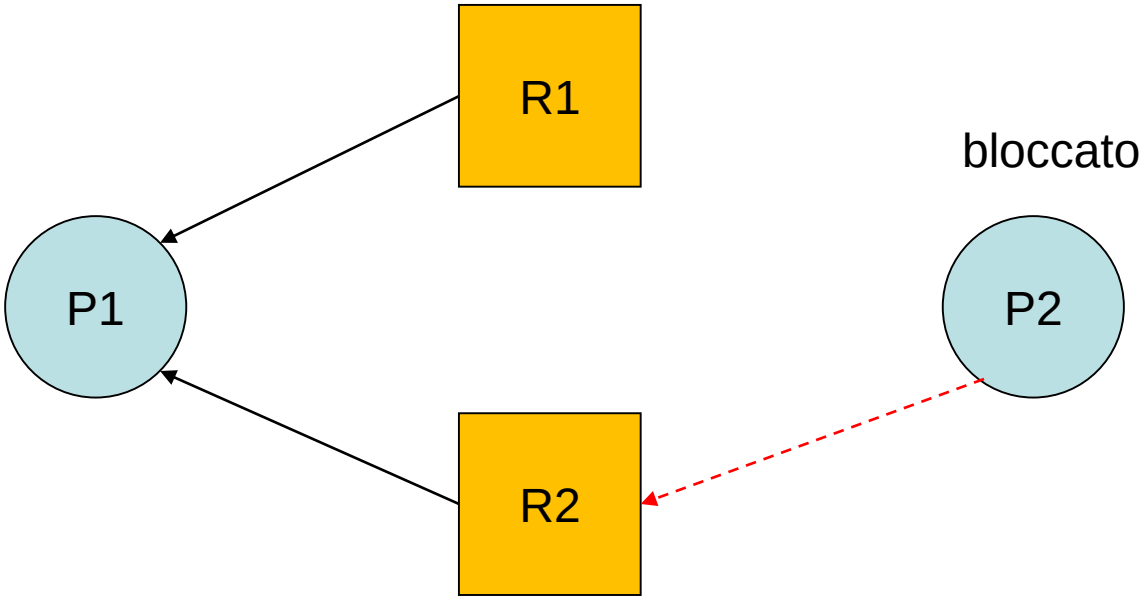


- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)

  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)

  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)



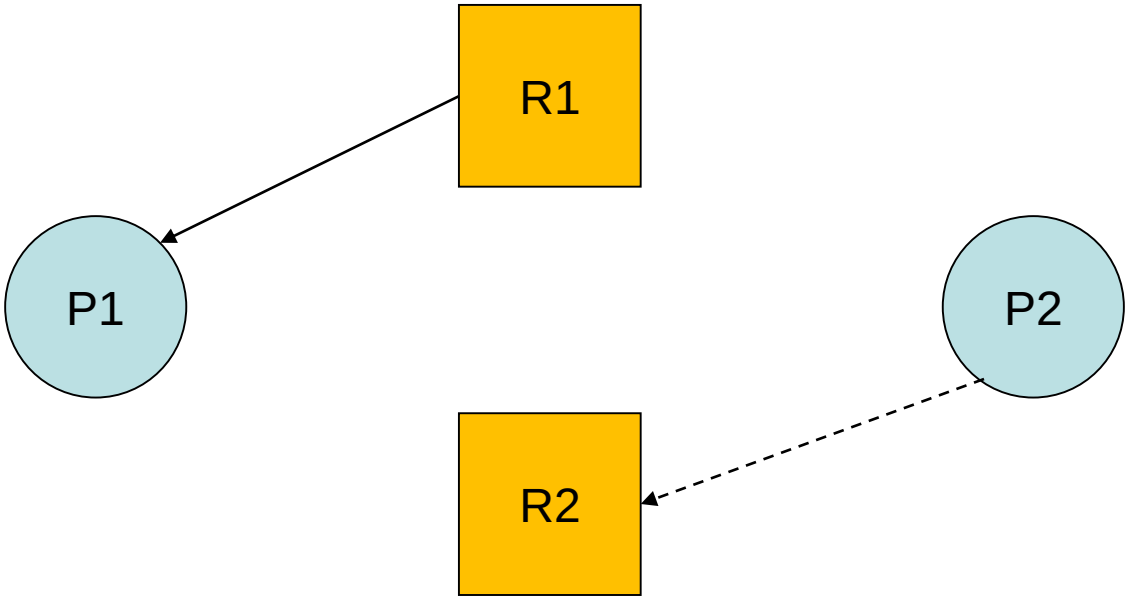
- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)

  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)

  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)

  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex2 e risveglia P2)



- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

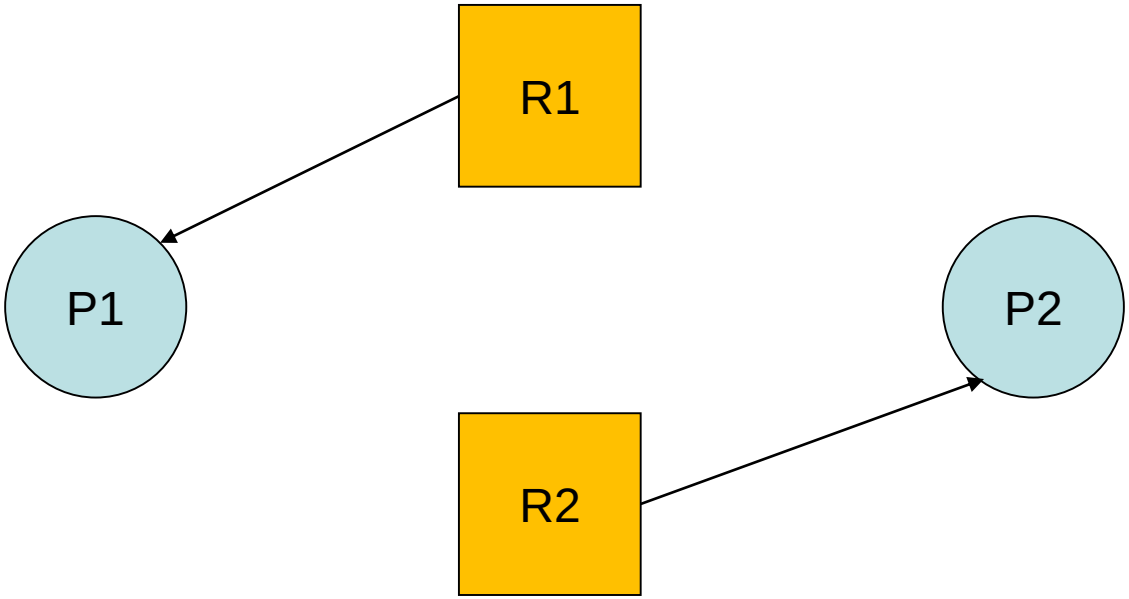
  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)

  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)

  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)

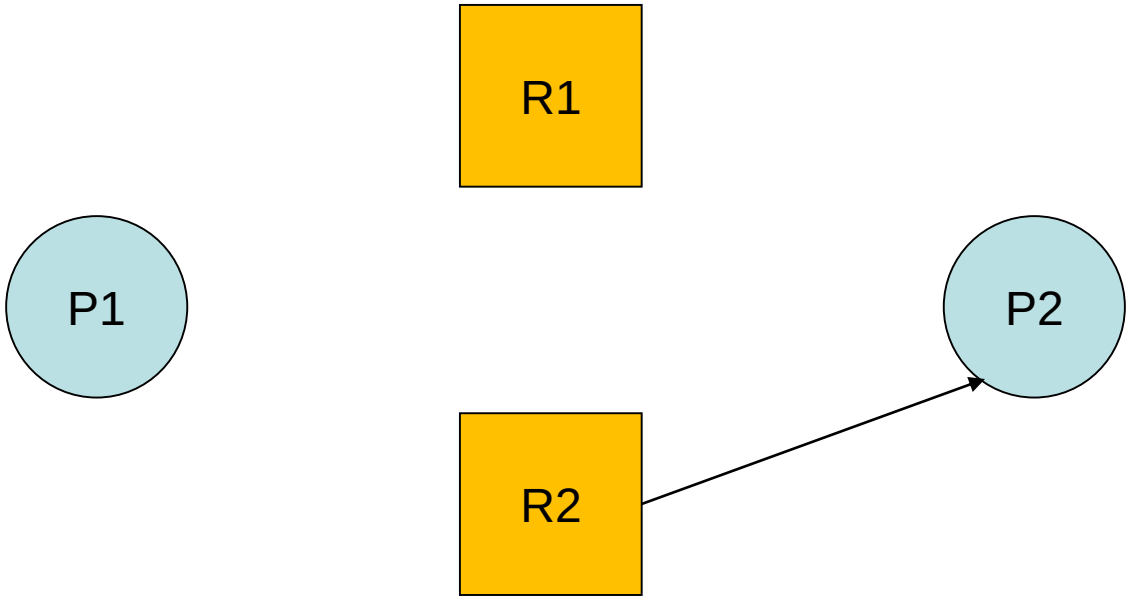
  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)

  T4: P2 alloca la risorsa R2



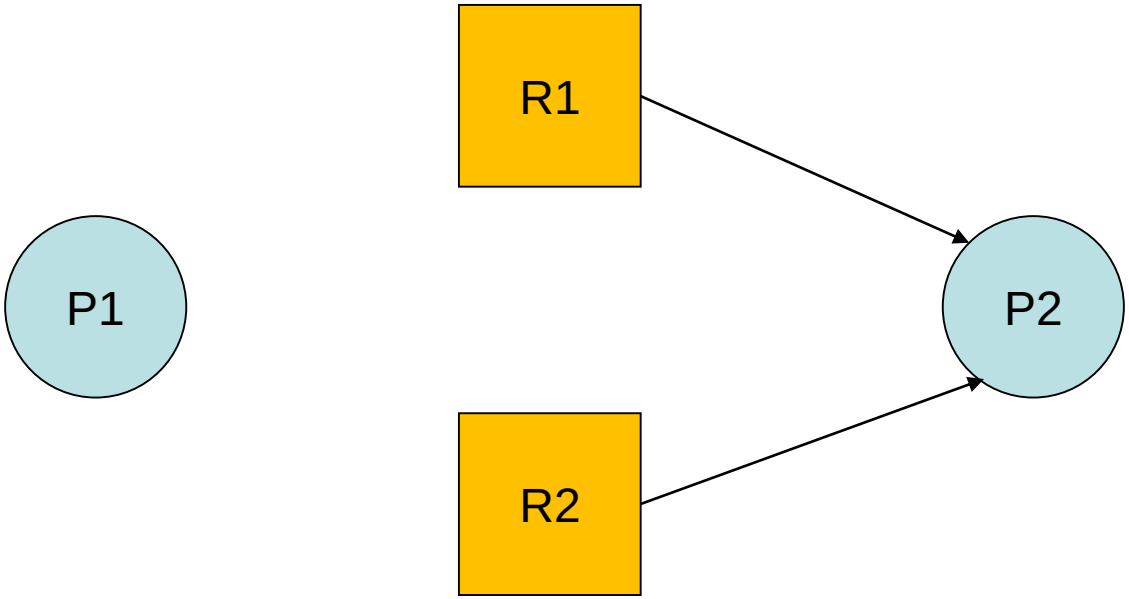
- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)
  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)
  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)
  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)
  T4: P2 alloca la risorsa R2
  T5: P1 esegue signal(mutex1) (P1 rilascia R1)



- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)
  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)
  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)
  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)
  T4: P2 alloca la risorsa R2
  T5: P1 esegue signal(mutex1) (P1 rilascia R1)
  T6: p2 esegue wait(mutex1) (P2 alloca R1)



- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T1: P1 esegue wait(mutex2) (P1 alloca la risorsa R2)

  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)

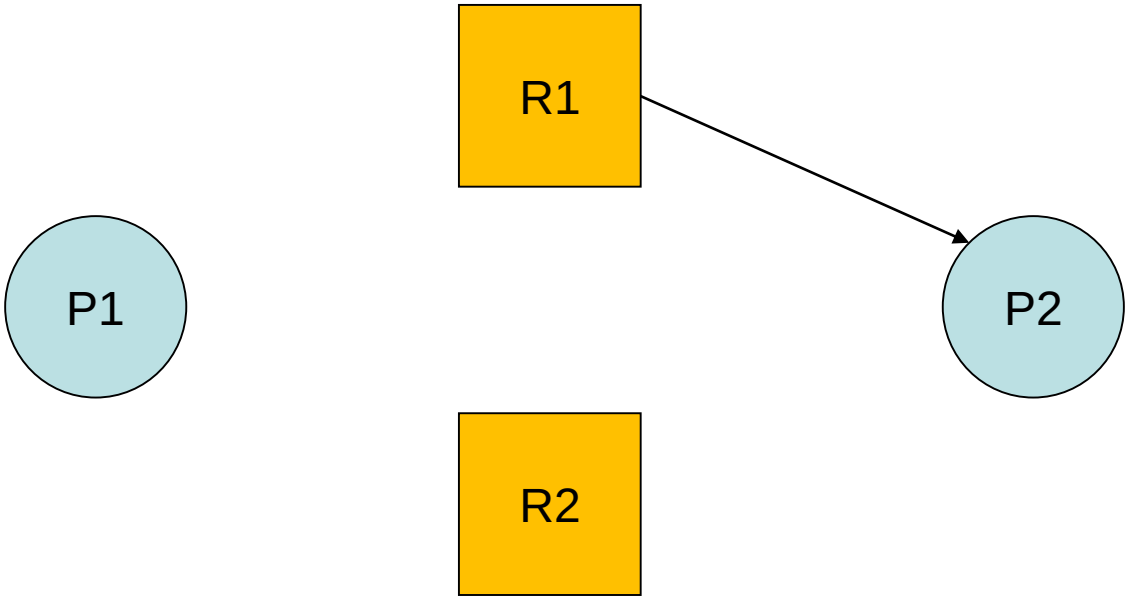
  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)

  T4: P2 alloca la risorsa R2

  T5: P1 esegue signal(mutex1) (P1 sblocca R1)

  T6: P2 esegue wait(mutex1) (P2 alloca R1)

  T7: P2 eseque signal(mutex2) (P2 rilascia R2)



- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  T2: P2 esegue wait(mutex2) (P2 si blocca su mutex2)

  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)

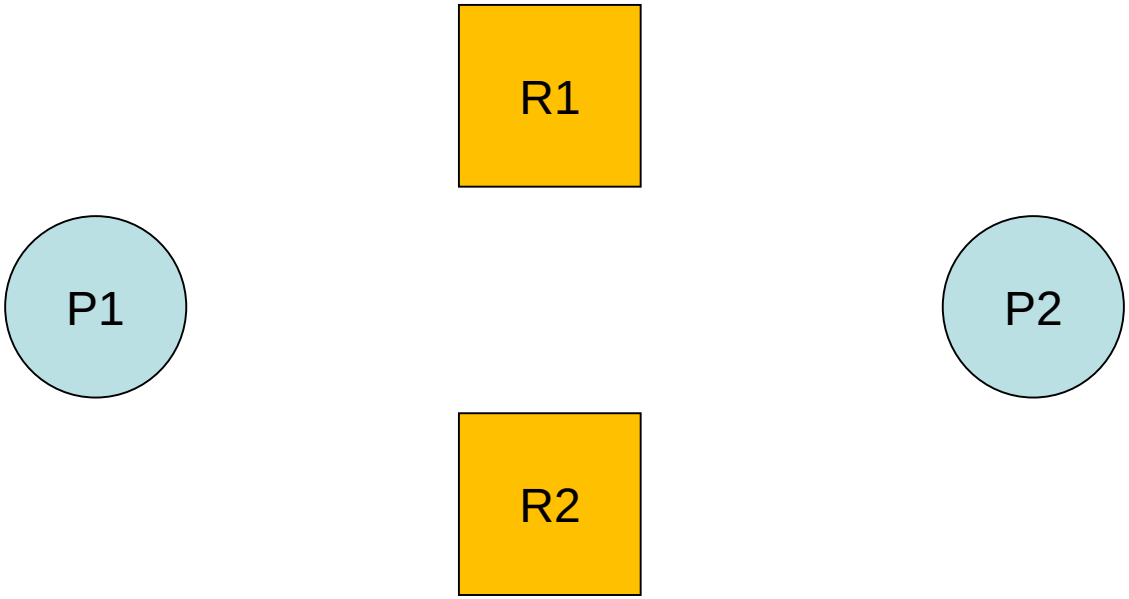
  T4: P2 alloca la risorsa R2

  T5: P1 esegue signal(mutex1) (P1 rilascia R1)

  T6: P2 esegue wait(mutex1) (P2 alloca R1)

  T7: P2 esegue signal(mutex2) (P2 rilascia R2)

  T8: P2 esegue signal(mutex1) (P2 rilascia R1)

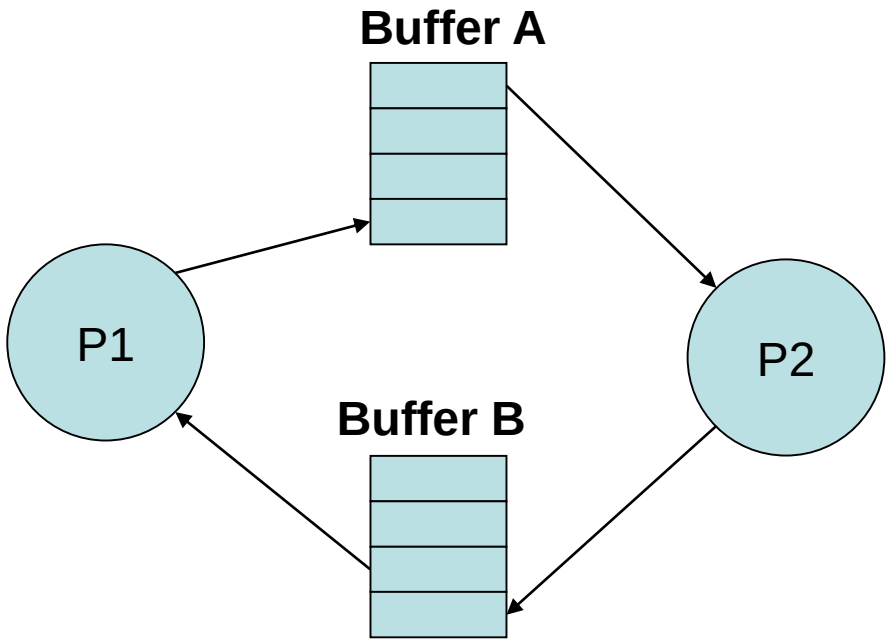


Risorse riusabili, consumabili e condivisibili

- Una risorsa è detta ***riusabile*** quando può essere usata da un processo alla volta e non viene distrutta dopo l'uso. Esempi di risorse riusabili sono risorse hardware come i dischi, i lettori DVD, le stampanti, scanner, etc. e risorse software come file, tabelle, etc.
- Una risorsa è detta ***non riusabile*** o ***consumabile***, quando non può essere riusata. Esempi di risorse consumabili sono i messaggi, i segnali e le interruzioni. Anche l'utilizzo di tali risorse può portare a situazioni di stallo.
- Una risorsa **è condivisibile** quando è riusabile e può essere usata senza ricorrere alla mutua esclusione. Un esempio di risorsa condivisibile è il file con accesso in sola lettura.

Blocco critico con risorse consumabili

- Consideriamo l'esempio in figura in cui i processi P1 e P2 si comportano rispettivamente da produttore e consumatore rispetto al buffer **A** e consumatore e produttore rispetto al buffer **B**.
- Se **i due buffer sono pieni**, P1 non può inserire il suo messaggio nel buffer A e quindi si blocca in attesa che intervenga P2, il quale a sua volta può essere bloccato in quanto impossibilitato ad inserire il messaggio nel buffer B.



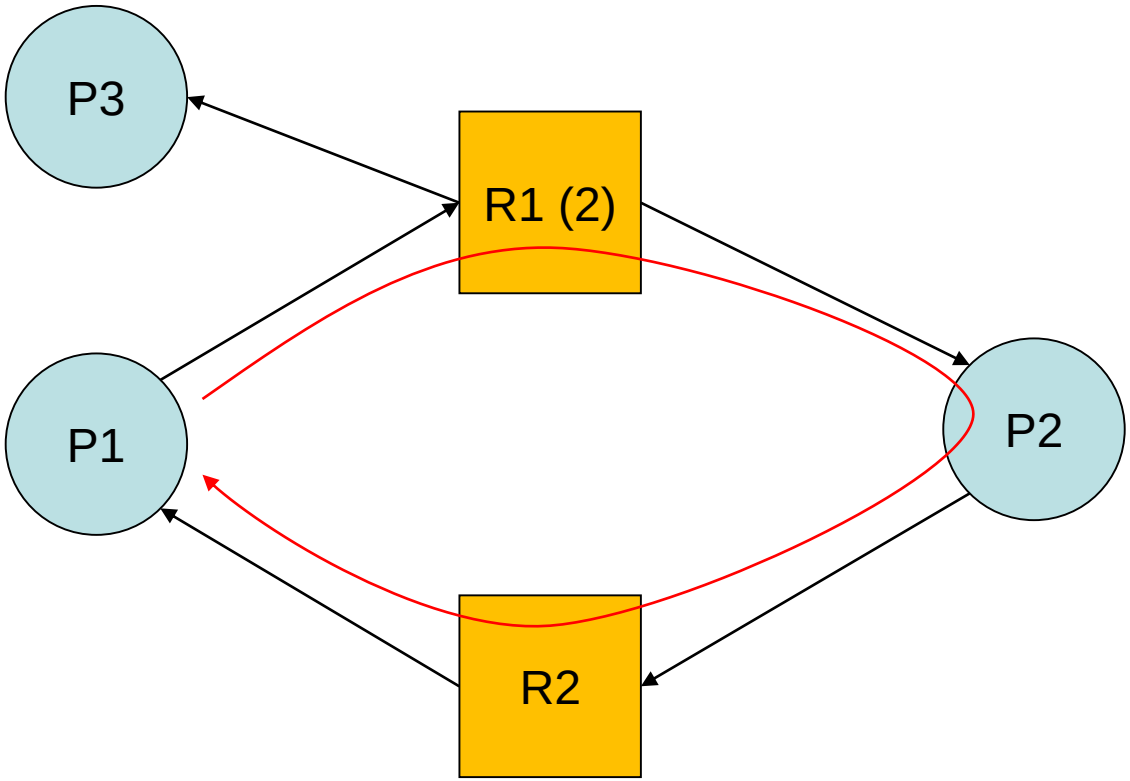
Condizioni per lo stallo

- Considerato un insieme di **N** processi *{P1, P2..PN}* e un insieme di **M** tipi di risorse *{R1, R2,..RM}* **si può verificare** una condizione di stallo se risultano vere **contemporaneamente** tutte le seguenti condizioni:

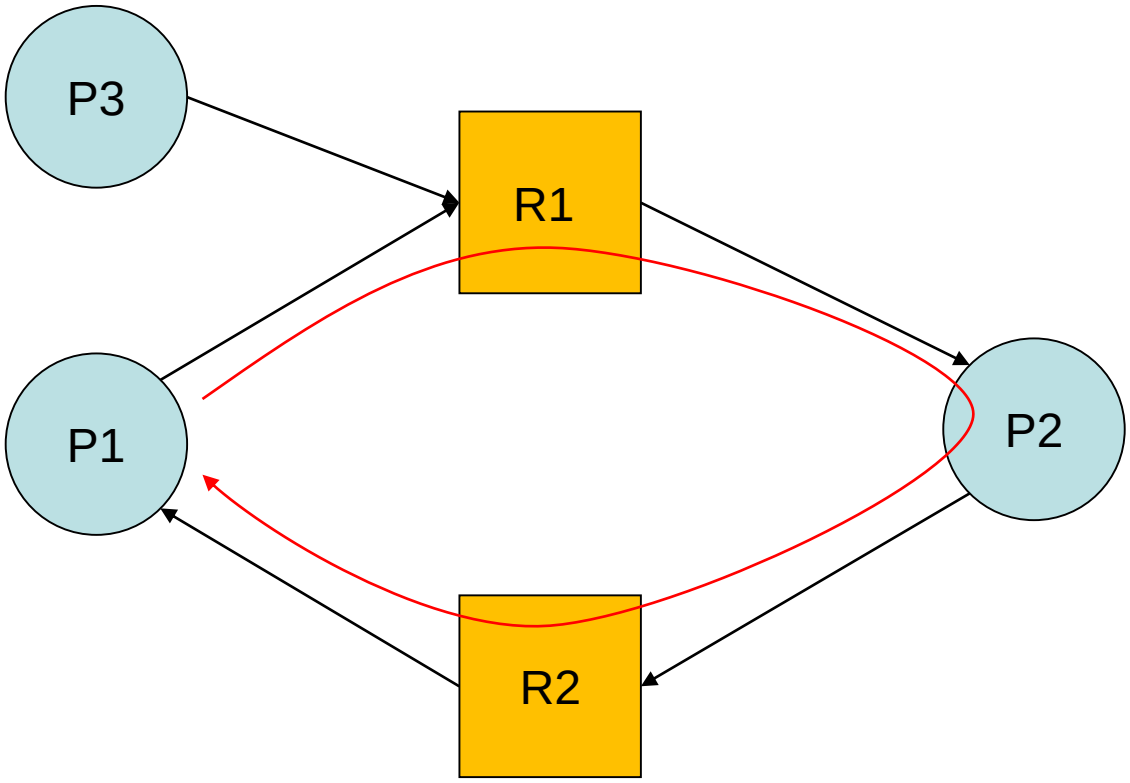
1. **Mutua esclusione**. Le risorse possono essere utilizzate da un solo processo alla volta;
2. **Possesso e attesa**. I processi non rilasciano le risorse che hanno già acquisito e per continuare la loro esecuzione ne richiedono altre;
3. **Mancanza di pre-rilascio**. Le risorse che sono state già assegnate ai processi non possono essere revocate;
4. **Attesa circolare**. Esiste un insieme di processi {Pi, Pi+1,…,Pk}, tali che Pi è in attesa di una risorsa acquisita da Pi+1, Pi+1 è in attesa di una risorsa acquisita da Pi+2,… Pk è in attesa di una risorsa acquisita da Pi.

Le prime tre condizioni sono necessarie ma non sufficienti affinché si verifichi lo stallo. La quarta condizione diventa sufficiente solo nel caso in cui che per ogni tipo di risorsa condivisa esista **solo un'unità.**

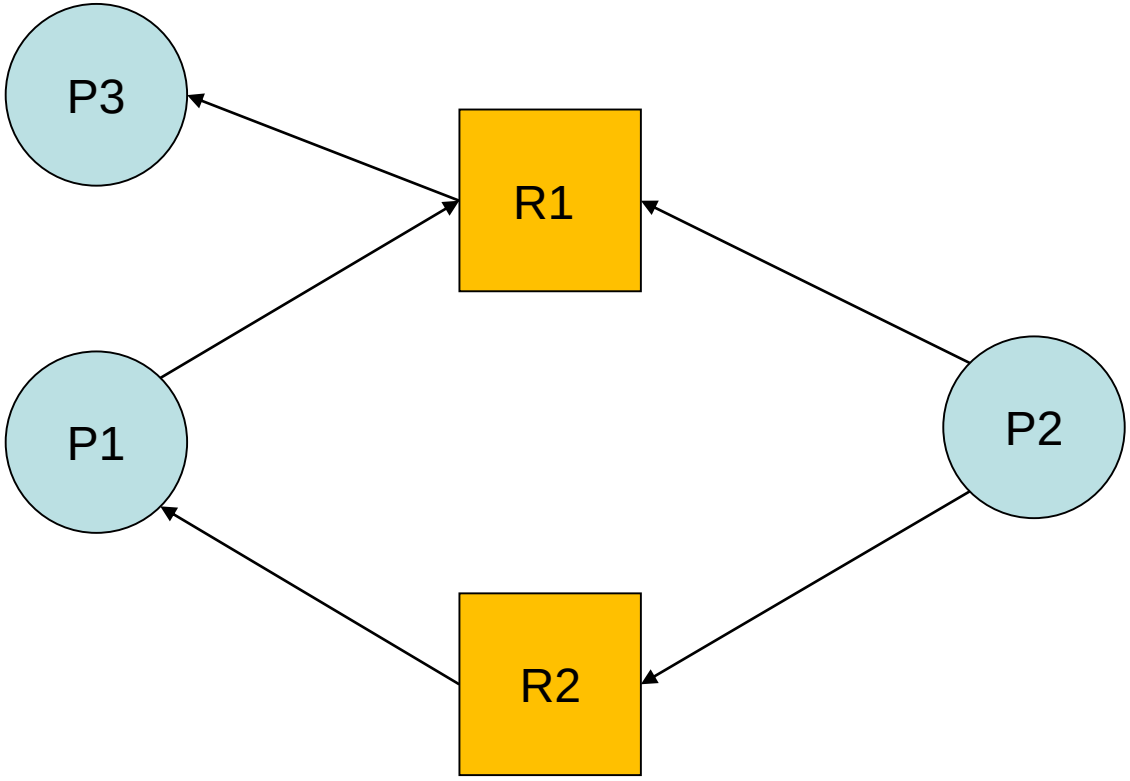
- L’esempio in figura mostra un caso in cui esistono 2 unità della risorsa R1. Il percorso circolare P1-R1-P2-R2 non porta ad una situazione di stallo in quanto il processo P3, dopo aver usato una copia di R1, la può rilasciare e quindi potrà essere allocata al processo P1, eliminando il percorso circolare.



- Questo secondo esempio mostra un caso in cui c'è solo un'unità della risorsa R1. In questo caso il percorso circolare P1-R1-P2-R2 porta ad una situazione di stallo in quanto P1 e P3 sono bloccati in attesa di R1 e P2 è bloccato in attesa di R2.



- In questo terzo caso non è presente un percorso circolare P1-R1-P2-R2 e il sistema non si trova in stallo.
- Tuttavia se successivamente, nel momento in cui P3 libera la risorsa R1 e questa venisse allocata a P2, si formerebbe un percorso circolare P1-R1-P2-R2 e quindi il sistema andrebbe in stallo. Se invece R1 venisse allocata a P1 non si verificherebbe una situazione di stallo.



Metodi per il trattamento dello stallo

- Il problema del blocco critico si può risolvere adottando tecniche di prevenzione:
  - **prevenzione statica**
  - **prevenzione dinamica**

Prevenzione statica

- Consiste nello **scrivere adeguatamente i programmi**, in modo tale che almeno una delle quattro condizioni necessarie non si verifichi. Non considerando la condizione di mutua esclusione, che è fondamentale per l'uso delle risorse condivise, si può intervenire sulle restanti tre condizioni: **possesso e attesa, mancanza di pre-rilascio, attesa circolare.**
- Le tecniche di prevenzione statica sono basate su vincoli sull'acquisizione delle risorse, che possono provocare un uso non efficiente delle risorse ed un rallentamento dei processi.