Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

Lezione 10

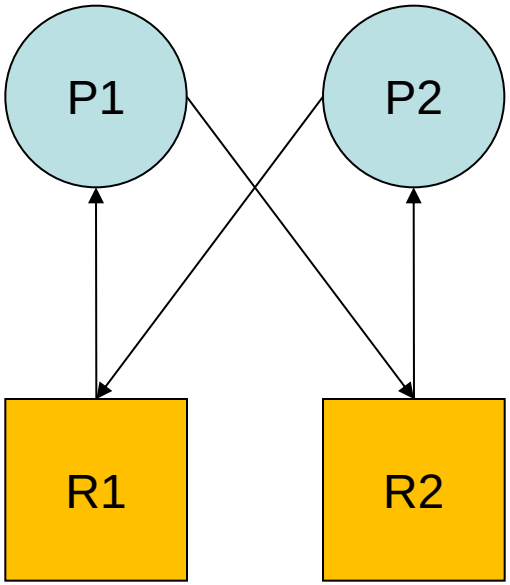
Martedì 15-11-2016

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 ().



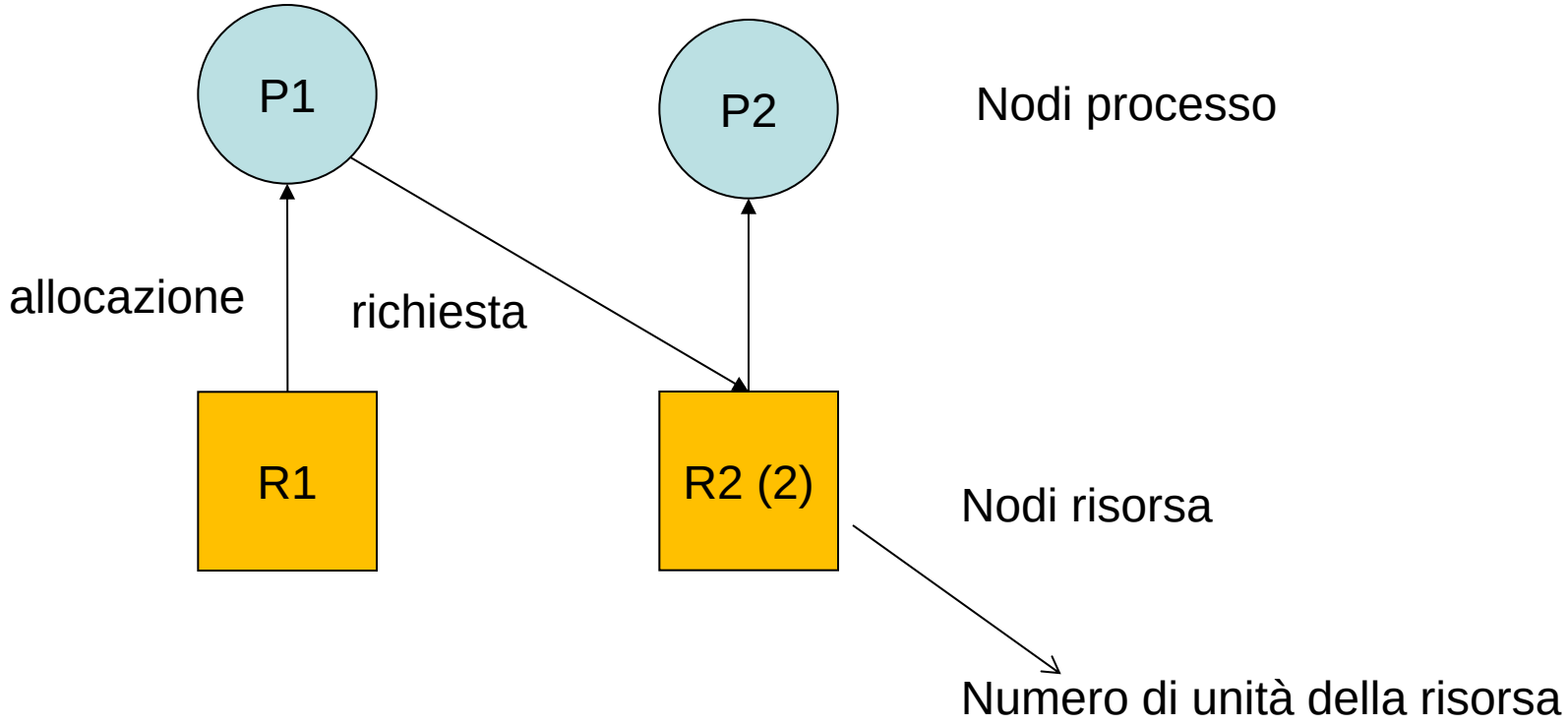
Rappresentazioni dello stato di allocazione delle risorse

- Per stabilire se un certo numero di processi è in stallo è necessario analizzare le informazioni relative alle risorse 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

- un **grafo di richiesta e allocazione risorse** è costituito da due tipi di nodo: i **nodi processo** e i **nodi risorsa**. Un nodo processo è rappresentato da un cerchio e un nodo risorsa è rappresentato da un quadrato (o da un rettangolo).

- Un numero nel nodo risorsa indica il numero di unità di quel tipo di risorsa.



Modelli basati su matrici

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

	R1	R2
P1	0	1
P2	1	0
P3	0	1

Risorse allocate

	R1	R2
P1	1	0
P2	0	1
P3	0	0

Risorse richieste

R1	R2
1	2

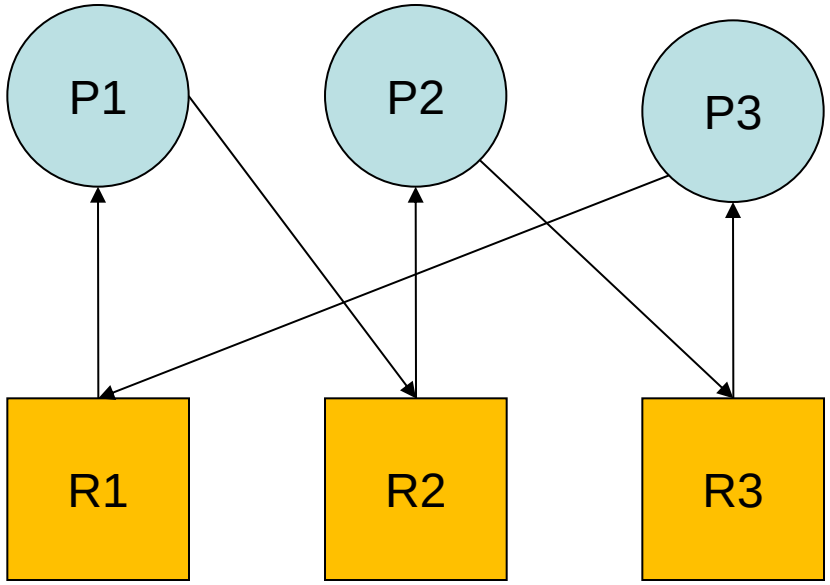
Risorse totali

R1	R2
0	0

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 blocco critico 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 nell'esempio seguente.

P1

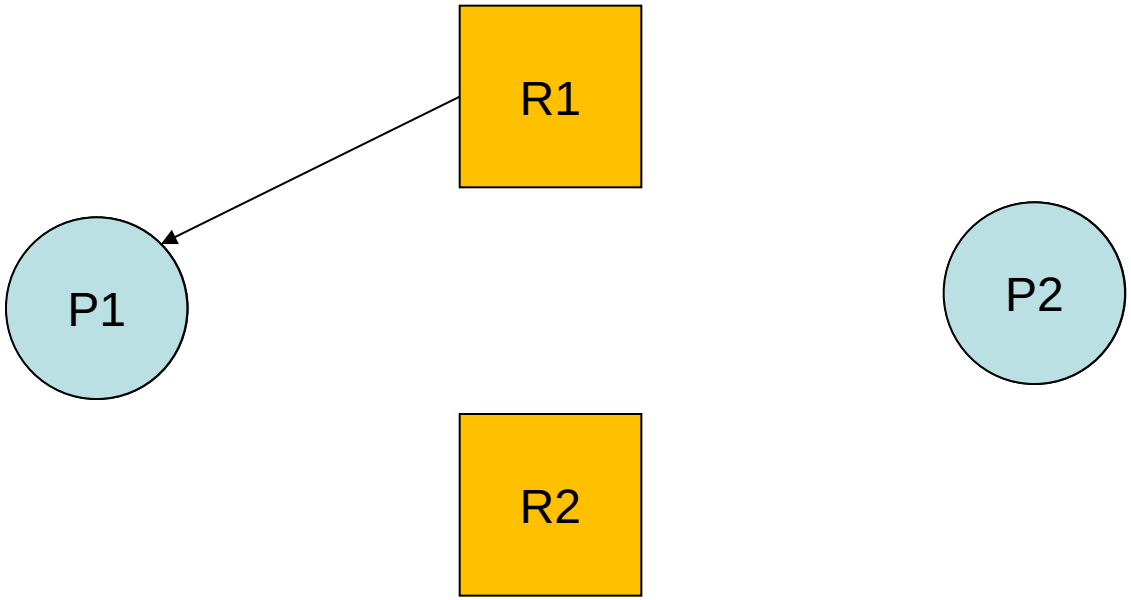
```
wait (mutex1);
wait (mutex2);

<uso risorse R1 e R2>

signal (mutex2);
signal (mutex1);
```

P2

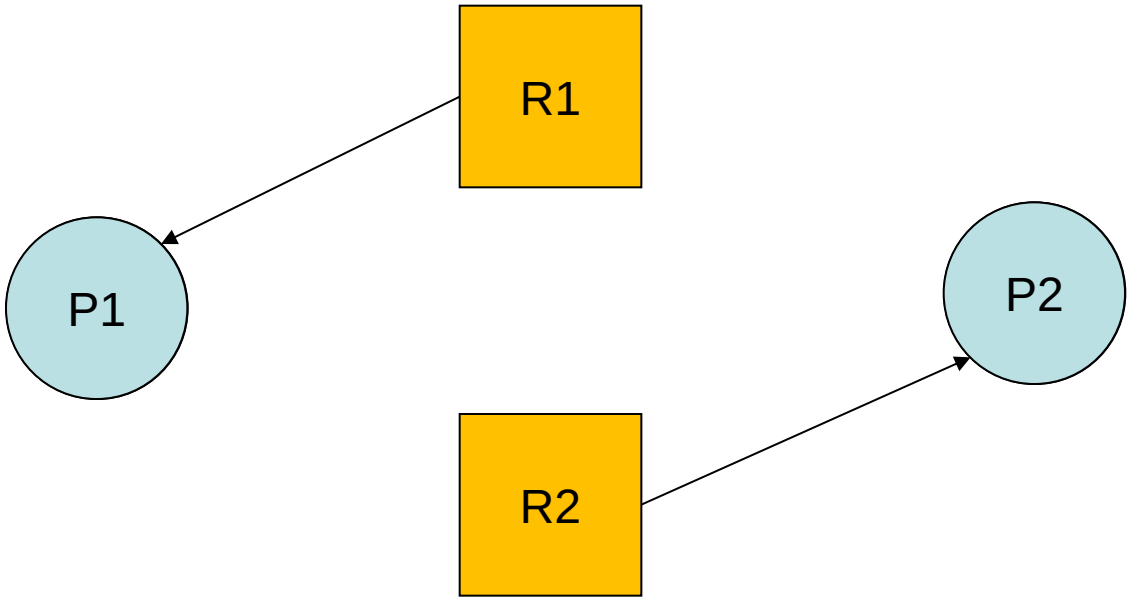
```
wait (mutex2);
wait (mutex1);

<uso risorse R1 e R2>

signal (mutex2);
signal (mutex1);
```

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

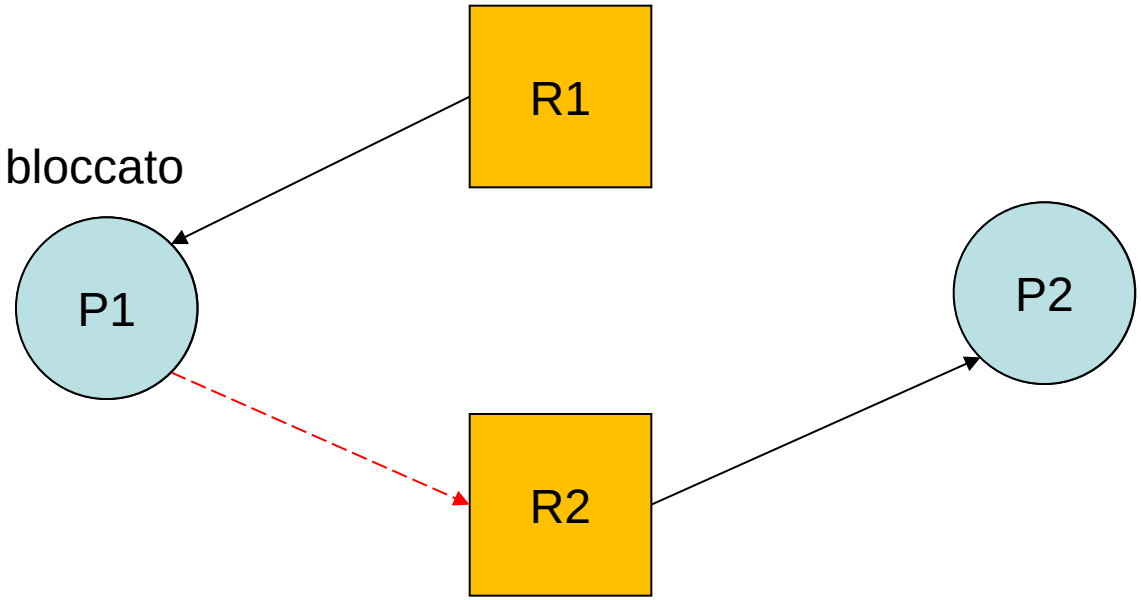
  T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)



- 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)**

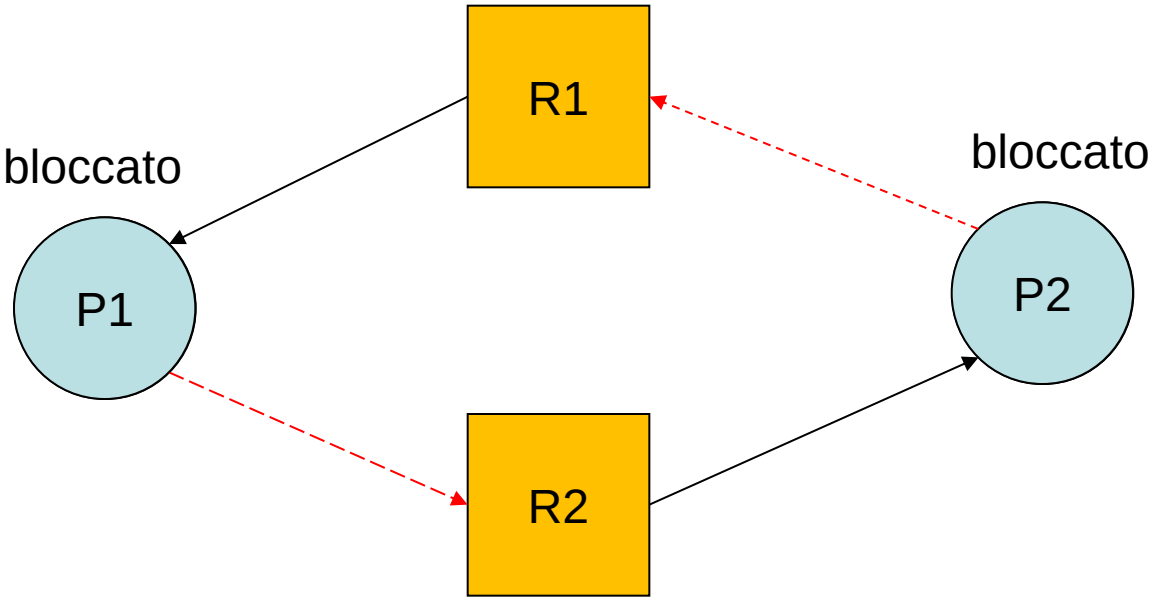


- Se i due processi P1 e P2 eseguono la seguente sequenza di operazioni nel tempo:
  - T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)
  - T1: P2 esegue wait(mutex2) (acquisisce la risorsa R2)
  - **T2: P1 esegue wait(mutex2) (P1 si blocca)**



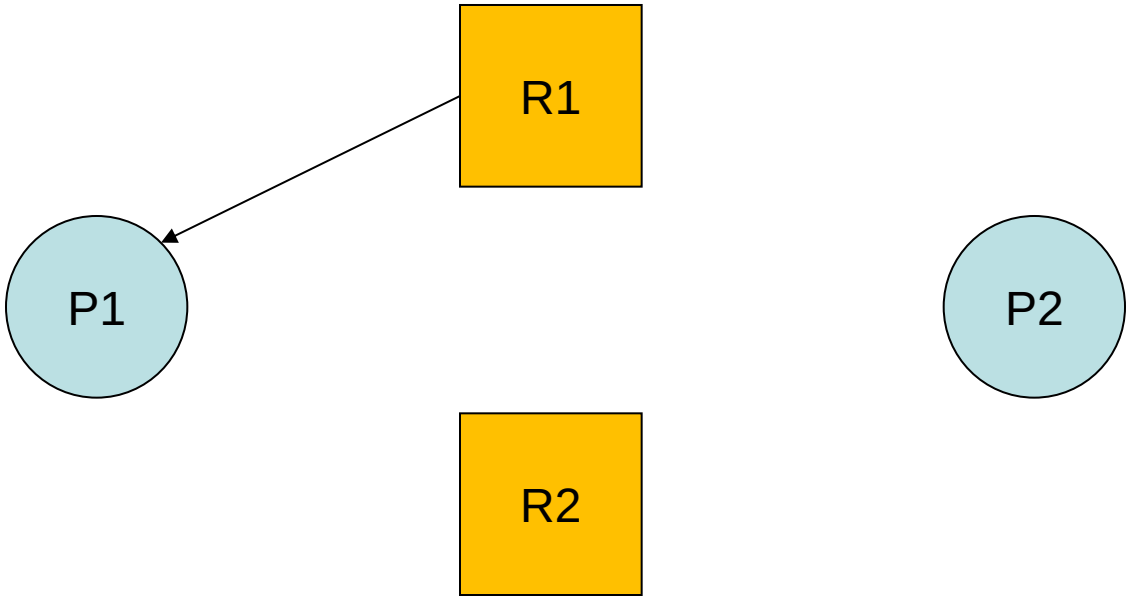
- Se i due processi P1 e P2 eseguono la seguente sequenza di operazioni nel tempo:
  - T0: P1 esegue wait(mutex1) (acquisisce la risorsa R1)
  - T1: P2 esegue wait(mutex2) (acquisisce la risorsa R2)
  - T2: P1 esegue wait(mutex2) (P1 si blocca)
  - **T3: P2 esegue wait(mutex1) (P2 si blocca)**

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:

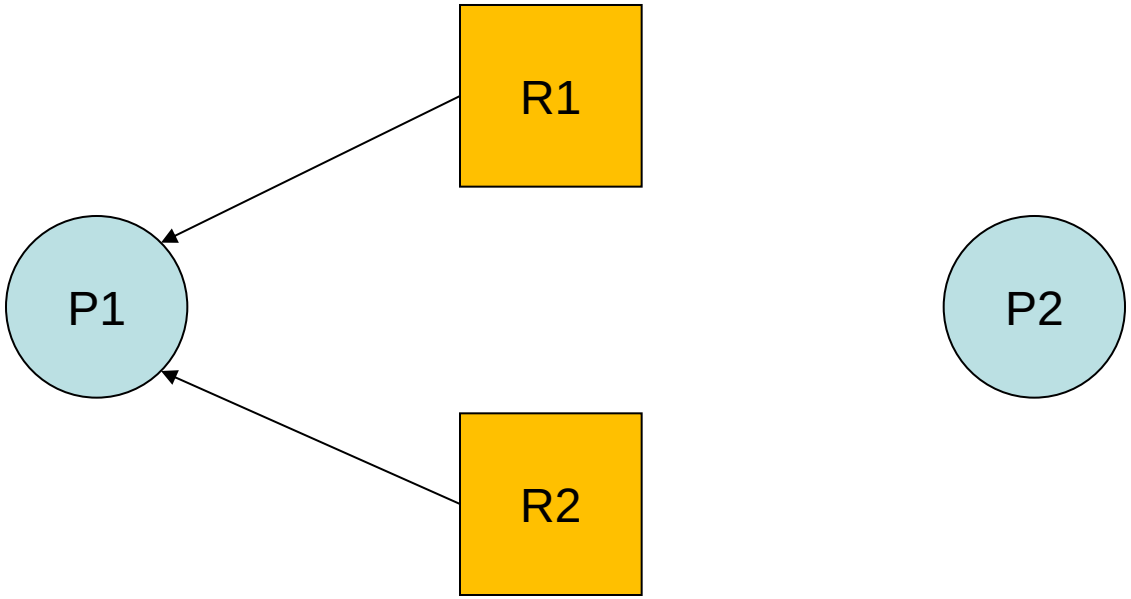
  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)



- 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)

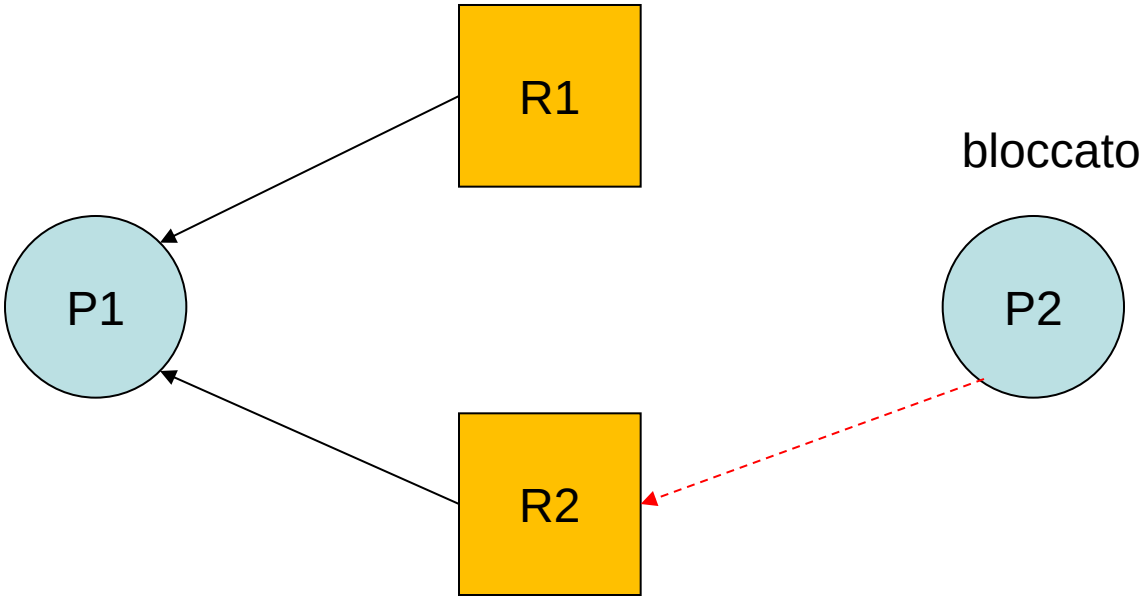


- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  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)



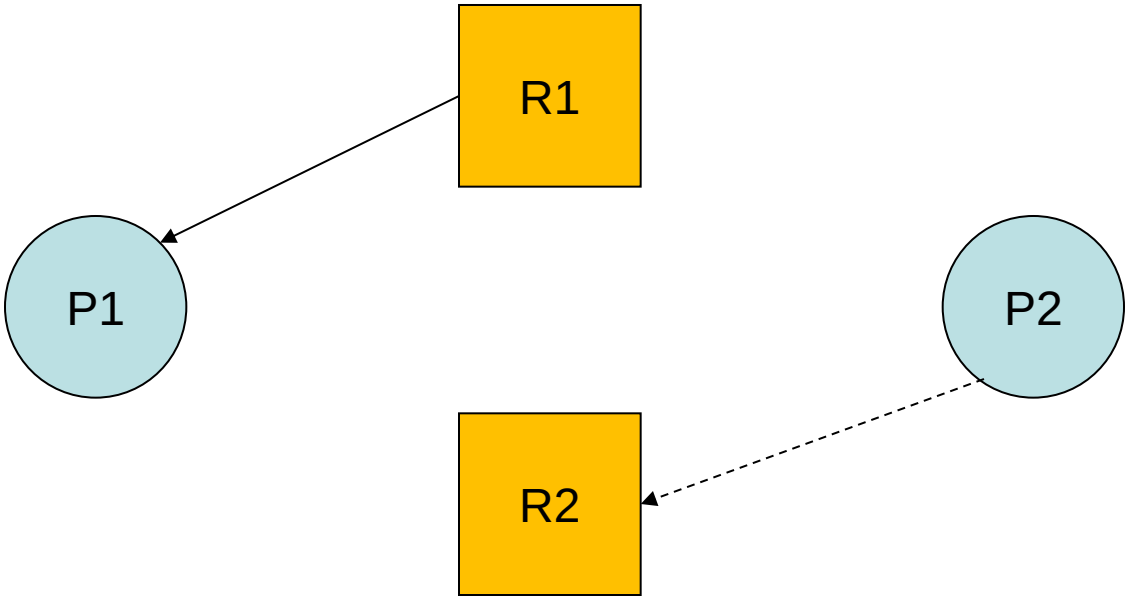
- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  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:

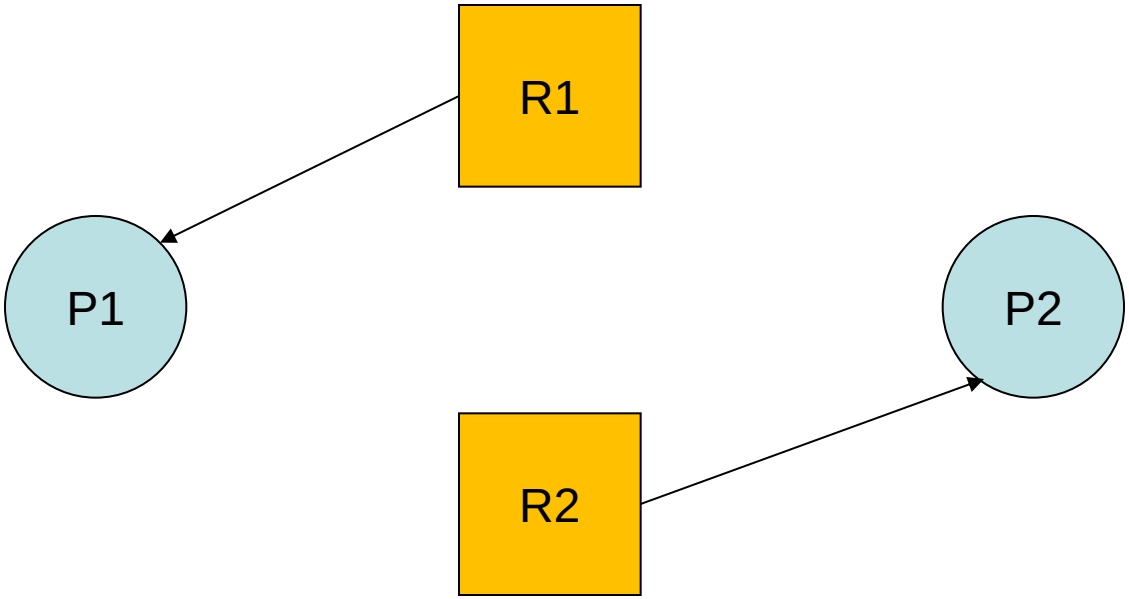
  T0: P1 esegue wait(mutex1) (P1 alloca la risorsa R1)

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

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

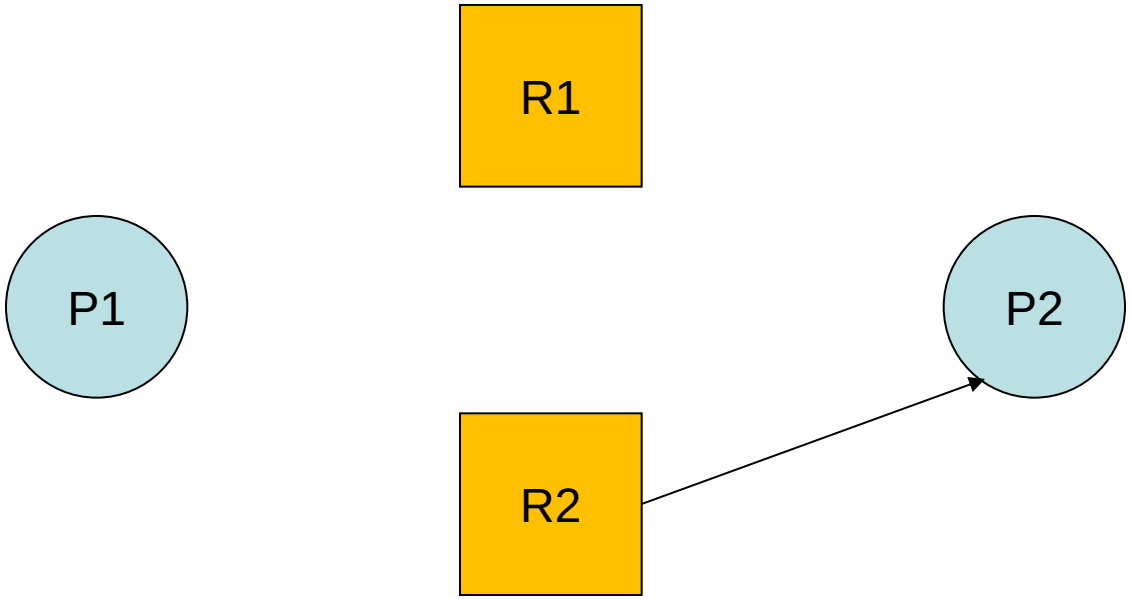
  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)

  T4: P2 alloca la risorsa R2



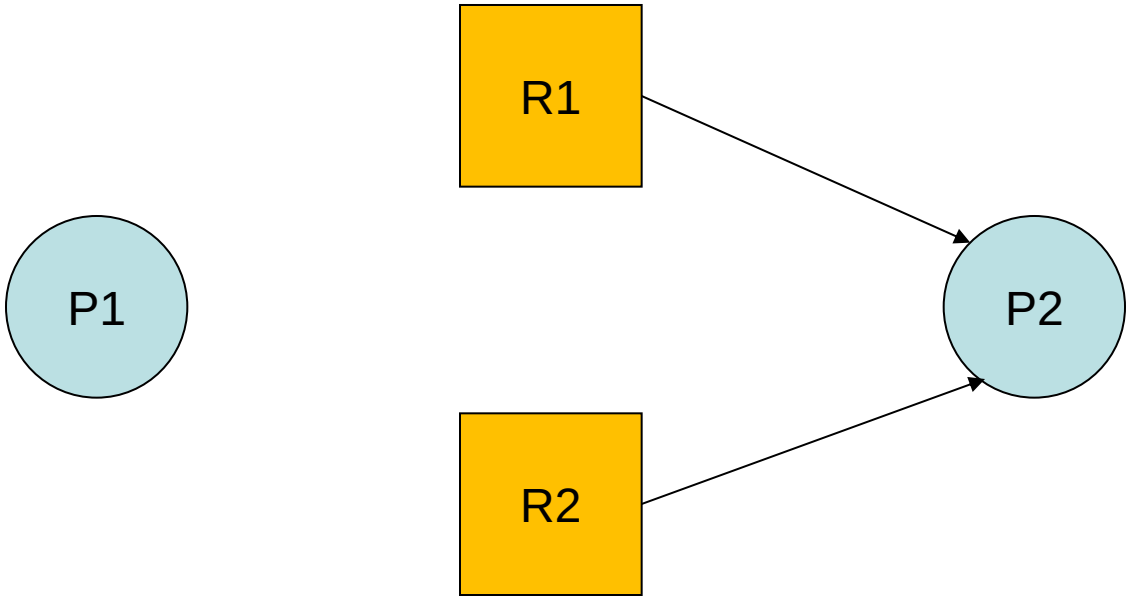
- Quest’altra condizione di velocità relativa non avrebbe portato allo stallo:

  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)

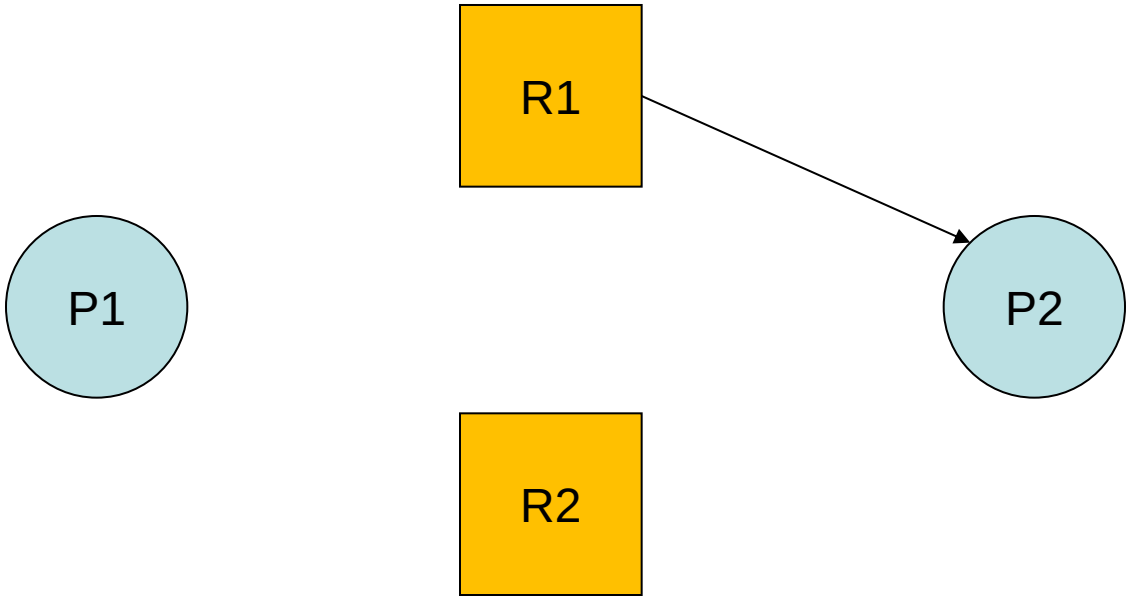
  T3: P1 esegue signal(mutex2) (P1 sblocca il mutex1 e risveglia P2)

  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)

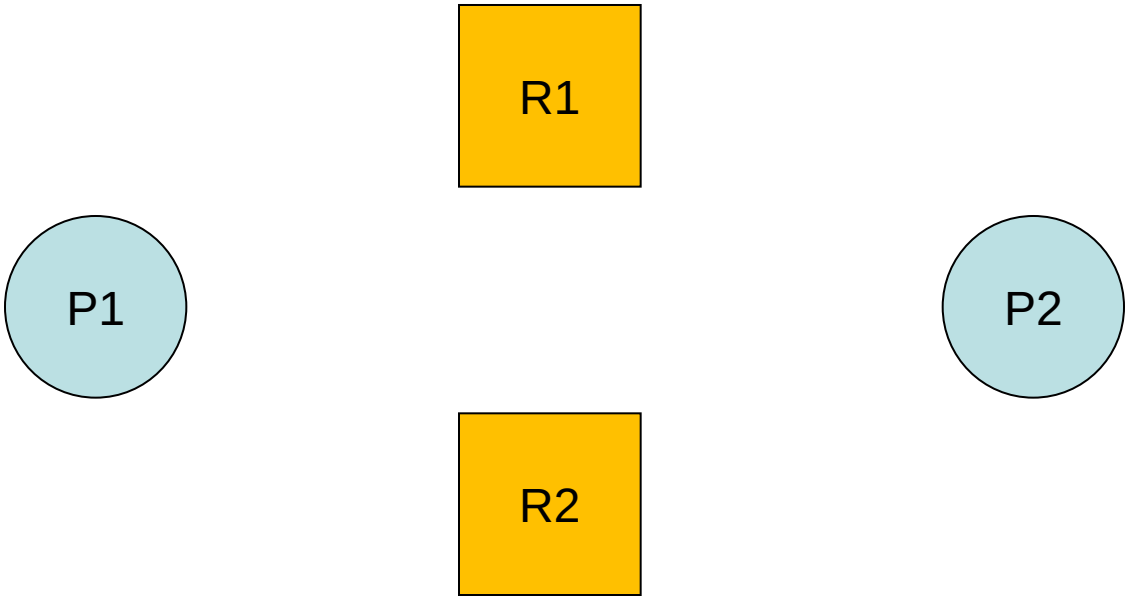
  T4: P2 alloca la risorsa R2

  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)

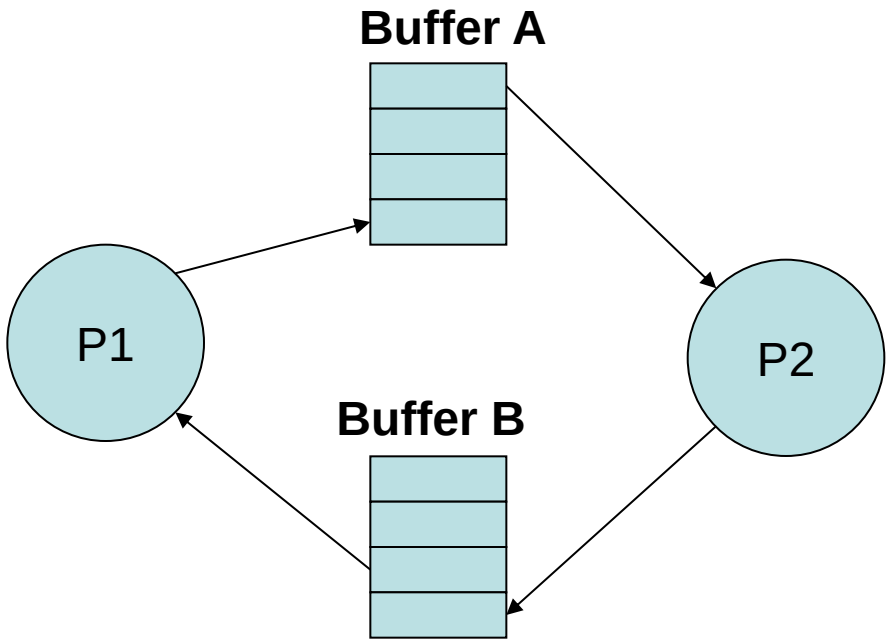


Risorse riusabili, consumabili e condivisibili

- 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.



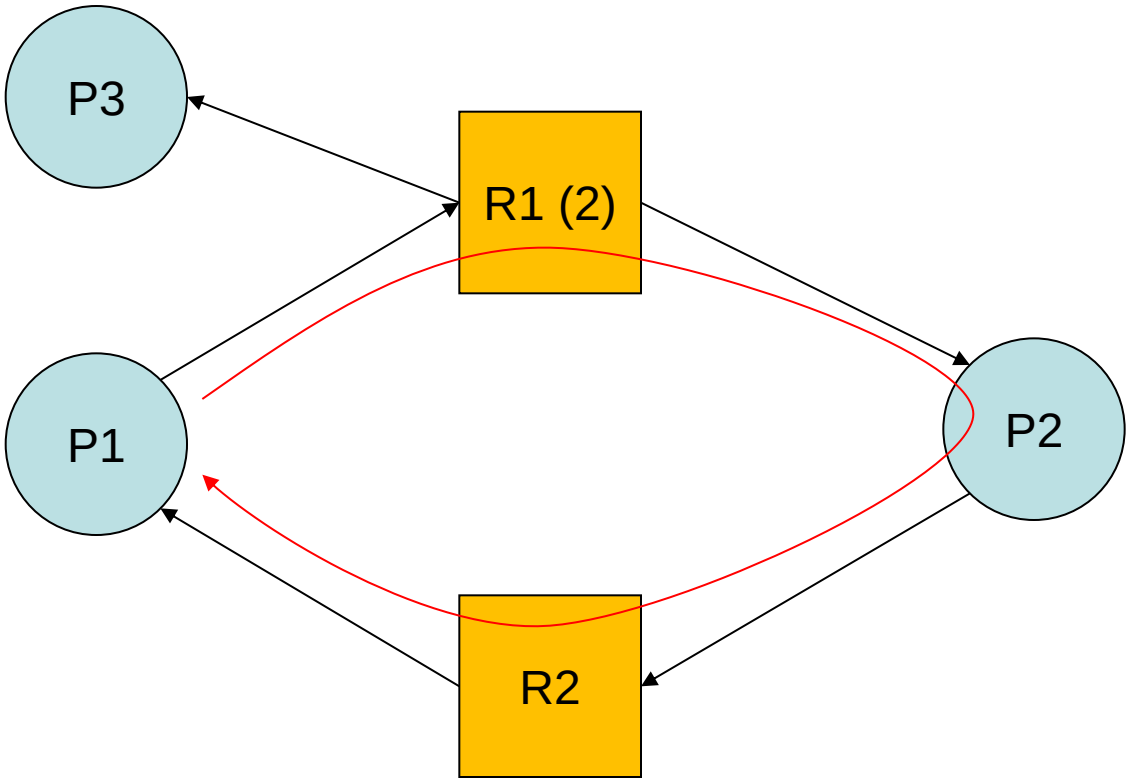
Condizioni per il blocco critico

- 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 blocco critico se risultano vere **contemporaneamente** tutte le seguenti condizioni:

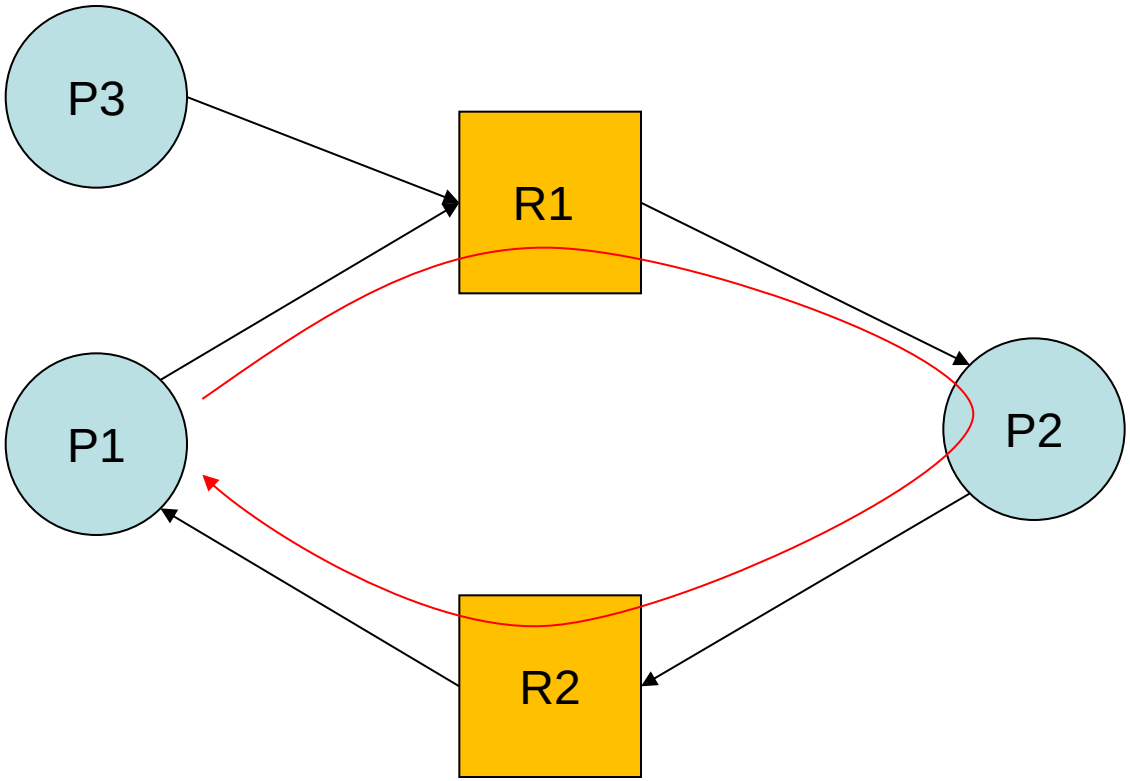
1. **Mutua esclusione**. Le risorse possono essere utilizzate da un solo processo alla volta;
2. **Possesso e attesa**. I processi non rilasciano le risorse che hanno già acquisito e per continuare la loro esecuzione ne richiedono altre;
3. **Mancanza di pre-rilascio**. Le risorse che sono state già 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 riusabile **esista solo una copia.**

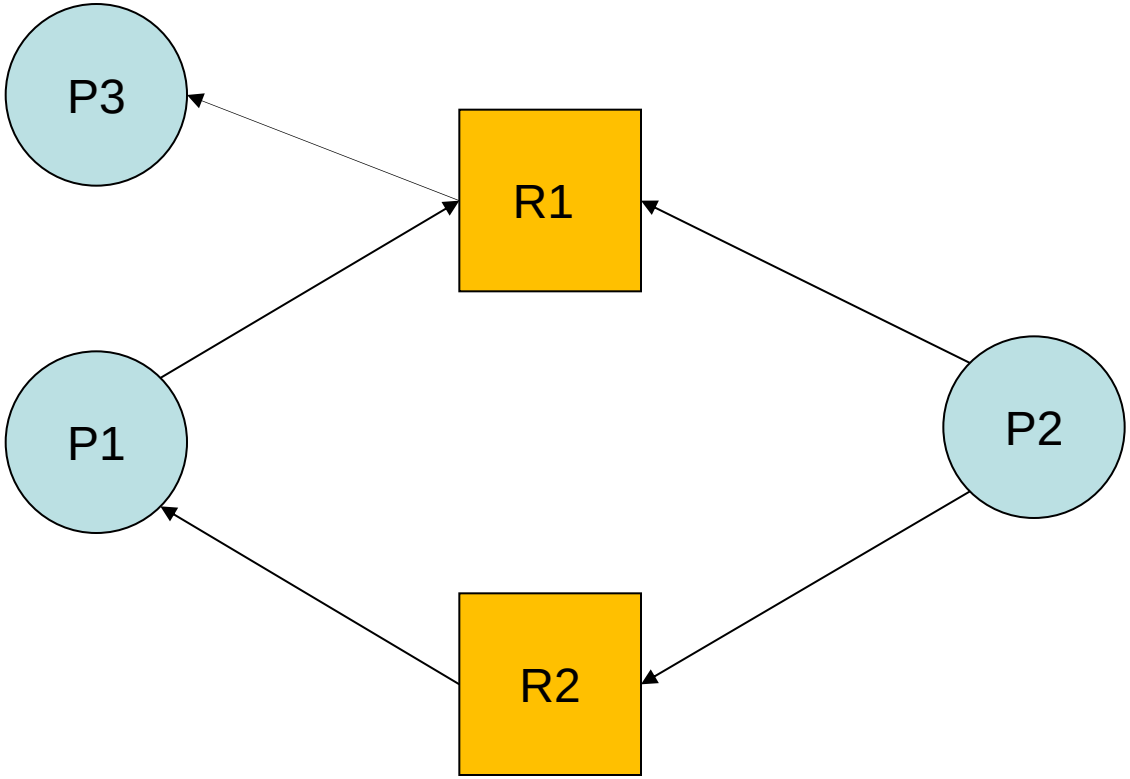
- L’esempio in figura mostra un caso in cui esistono 2 unità della risorsa R1. Il percorso circolare P1-R1-P2-R2 non porta ad una situazione di stallo in quanto il processo P3, dopo aver usato una copia di R1, la può rilasciare e quindi potrà essere allocata al processo P1, eliminando il percorso circolare.



- 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 del blocco critico

- 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 riusabili, 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.

Prevenzione dinamica

- Le tecniche di prevenzione dinamica si basano su algoritmi in grado di verificare, in base allo stato corrente di allocazione delle risorse e alle richieste dei processi, se l'assegnazione di risorse dovute ad una nuova richiesta da parte di un processo può portare a una situazione di stallo.
- Un noto algoritmo di prevenzione dinamica, ideato da Dijkstra, è l'**algoritmo del banchiere** (per una certa analogia al comportamento del banchiere) .
- L'algoritmo risulta molto riduttivo per essere usato nei SO di uso generale in quanto è basato sui seguenti vincoli:
  1. il sistema operativo può gestire un numero fisso di processi e un numero fisso di risorse. Inoltre i processi devono dichiarare inizialmente il numero massimo di risorse di cui hanno bisogno durante la loro esecuzione.
  2. I processi possono richiedere nuove risorse mantenendo le unità già in loro possesso.
  3. Tutte le risorse assegnate a un processo sono rilasciate quando il processo termina la sua esecuzione.

- Lo stato del sistema si dice ***sicuro*** se è possibile trovare una sequenza **Ph-Pj..Pk** con cui assegnare le risorse ai processi, detta **sequenza sicura**, in modo tale che tutti i processi possano usare le risorse che richiedono e terminare.
- Se, in un determinato istante, le risorse che un processo **Pi** richiede non sono disponibili, allora **Pi** si blocca fino a che tutti i processi che lo precedono nella sequenza liberino un numero sufficiente di risorse necessarie a **Pi**.
- Se non esiste una sequenza sicura allora lo stato del sistema è detto ***non è sicuro***. Uno stato non sicuro può portare a una condizione di blocco critico.
- L'algoritmo deve quindi consentire l'allocazione delle risorse ai processi solo quando le allocazioni portano a stati sicuri.

- Consideriamo, ad esempio, il caso di un sistema con tre processi P1, P2, P3 in cui siano disponibili 15 unità di un solo tipo di risorsa e che sia noto il massimo numero di risorse che ciascun processo può richiedere: 12 per P1, 3 per P2 e 11 per P3. Lo **stato iniziale** del sistema può essere così rappresentato:

	R1
P1	0
P2	0
P3	0

Risorse allocate

	R1
P1	12
P2	3
P3	11

Risorse richieste

R1
15

Risorse totali

R1
15

Risorse libere

- Se dopo un certo periodo di tempo sono state assegnate 8 unità a p1, 2 a p2 e 11 a P3, lo stato in cui il sistema si trova può essere così descritto:

	R1
P1	8
P2	2
P3	3

Risorse allocate

	R1
P1	4
P2	1
P3	8

Risorse richieste

R1
15

Risorse totali

R1
2

Risorse libere

- Vediamo se questo **stato è sicuro,** verificando se, a partire da questo stato, esiste una sequenza sicura.

	R1
P1	8
P2	2
P3	3

Risorse allocate

	R1
P1	4
P2	1
P3	8

Risorse richieste

R1
15

Risorse totali

R1
2

Risorse libere

- Si può notare che:
  1. il processo **P2** può allocare la risorsa richiesta, potendo quindi completare la sua esecuzione e liberare le sue 3 risorse che aveva allocato, portando quindi a 4 le risorse disponibili.

	R1
P1	8
P2	**3**
P3	3

Risorse allocate

	R1
P1	4
P2	**0**
P3	8

Risorse richieste

R1
15

Risorse totali

R1
1

Risorse libere

- Quindi quando P2 termina la stato di allocazione è il seguente:

	R1
P1	8
P2	0
P3	3

Risorse allocate

	R1
P1	4
P2	0
P3	8

Risorse richieste

R1
15

Risorse totali

R1
4

Risorse libere

2. A questo punto, il processo **P1** può ora ottenere tutte le 4 risorse ancora necessarie e terminare, portando a 12 le risorse disponibili che consentono al processo **P3** di terminare.

- Lo stato indicato è quindi uno ***stato sicuro*** in quanto partendo da esso esiste la ***sequenza sicura*** **(P2, P1, P3).**

- Altre sequenze potrebbero far passare il sistema da uno stato sicuro a uno stato non sicuro.

	R1
P1	8
P2	2
P3	3

Risorse allocate

	R1
P1	4
P2	1
P3	8

Risorse richieste

R1
15

Risorse totali

R1
2

Risorse libere

- Se, ad esempio, il processo **P3** chiede e otteniene una risorsa, il sistema in questo caso passerebbe in **uno stato che non è sicuro**.

	R1
P1	8
P2	2
P3	**4**

Risorse allocate

	R1
P1	4
P2	1
P3	**7**

Risorse richieste

R1
15

Risorse totali

R1
1

Risorse libere

- Infatti, l'unica risorsa rimasta libera può soddisfare soltanto la richiesta del processo P2 consentendogli di terminare l'esecuzione e liberare le 3 risorse in suo possesso.

	R1
P1	8
P2	**3**
P3	4

Risorse allocate

	R1
P1	4
P2	**0**
P3	7

Risorse richieste

R1
15

Risorse totali

R1
0

Risorse libere

- A questo punto nessun altro processo può terminare: P1 non può ottenere le 4 risorse di cui ha bisogno non essendo queste disponibili e quindi deve essere sospeso; analogamente P3 non può ottenere le 7 risorse e quindi anche esso viene sospeso. Si giunge quindi ad una **situazione di stallo**.

Nell’esempio, per evitare lo stallo, a partire dal precedente stato sicuro, la risorsa richiesta da P3 non deve essere ad esso allocata anche se disponibile.

Rilevamento dei blocchi critici

- Se non si prendono adeguati provvedimenti di prevenzione statica o dinamica è possibile che si verifichino situazioni di stallo, coinvolgendo un certo numero di processi e di risorse.
- Spesso si ricorre solo alla rilevazione e alla eliminazione del blocco critico, senza ricorrere ad alcuna tecnica di prevenzione, come nel caso di Windows e Unix.
- L'algoritmo di **rilevazione** viene eseguito dal SO periodicamente con una frequenza che dipende dal tipo di applicazioni o quando ad esempio il grado d'uso della CPU scende sotto una certa soglia in quanto una condizione di blocco critico può rendere inefficienti le prestazioni del sistema.
- L'eliminazione del blocco critico si può ottenere con differenti tecniche, la più semplice ed estrema consiste nel fare terminare tutti i processi coinvolti.

- Una soluzione meno drastica consiste nel far terminare uno alla volta i processi coinvolti e liberando via via le risorse da esso allocate, fino a giungere all'eliminazione dello stallo. In questo caso è possibile usare politiche di selezione dei processi da far terminare, basate ad esempio sulla priorità, sul tempo di cpu utilizzato, sul numero di risorse allocate, etc.
- L'implementazione di tali strategie può portare ad un alto overhead per il SO, in quanto dopo ogni terminazione forzata occorre verificare di nuovo se c'è ancora una situazione di stallo.