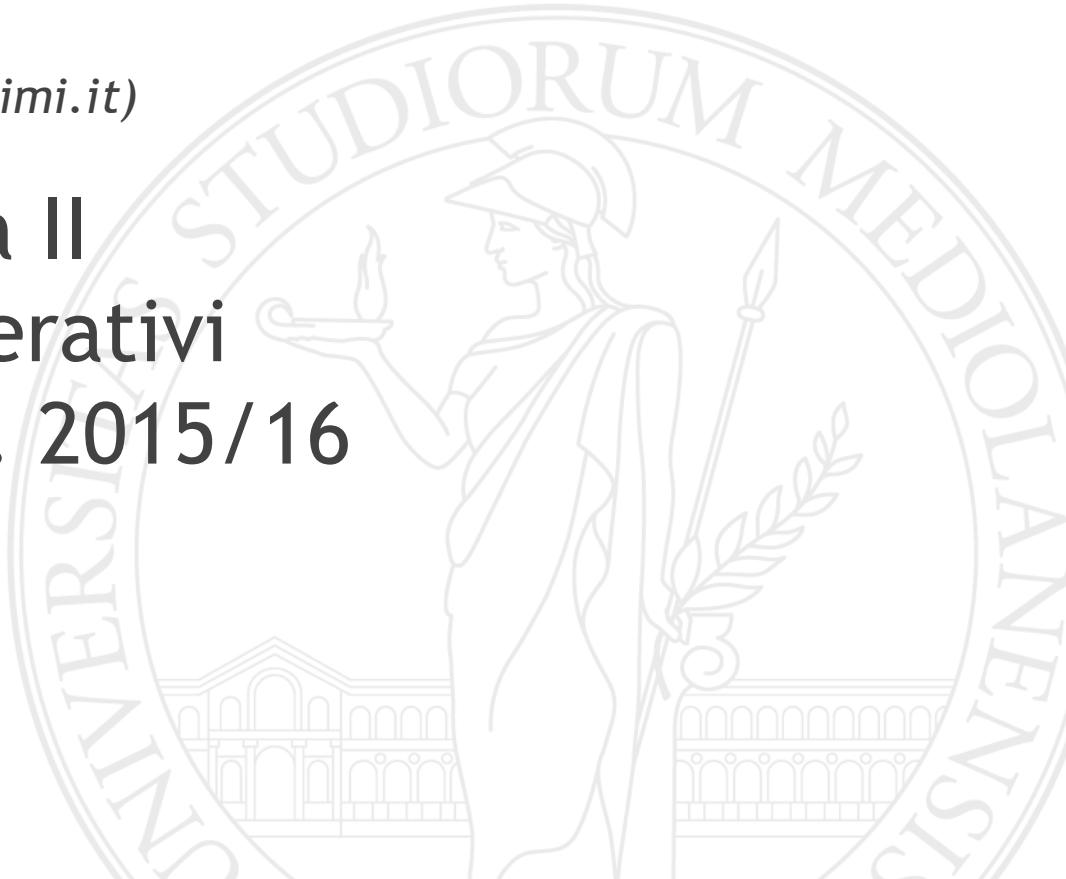
UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Alberto Ceselli
(alberto.ceselli@unimi.it)

Informatica II
Sistemi Operativi
DIGIP - a.a. 2015/16



Sistemi Operativi

(modulo di Informatica II)

Lo stallo (deadlock)

Patrizia Scandurra

Università degli Studi di Bergamo

a.a. 2014-15

Sommario

- Uso di risorse condivise
- Caratterizzazione del deadlock
  - Condizioni per l'occorrenza del deadlock
  - Grafo di allocazione delle risorse
- Metodi di gestione del deadlock
  1. *Ignorare il problema*
  2. *Prevenire il deadlock*
  3. *Evitare il deadlock*
  4. *Rilevazione e ripristino del deadlock*

Uso di risorse condivise

- Vari tipi di risorse R_1, R_2, …
  - cicli di CPU, spazio di memoria, periferiche di I/O, ecc..
  - per ogni risorsa un processo può richiedere delle *istanze*
    - le istanze sono identiche tra loro (e.g. due CPU)
    - al processo non interessa quale specifica istanza ricevere
    - le risorse vengono quindi classificate in classi (tipi)
- Risorse condivise usabili:
  - in modo non esclusivo
  - solo in modo mutuamente esclusivo
- Ogni processo utilizza una risorsa come segue:
  1. Richiesta di uso della risorsa
  2. Uso della risorsa
  3. Rilascio della risorsa

Deadlock ("stallo")

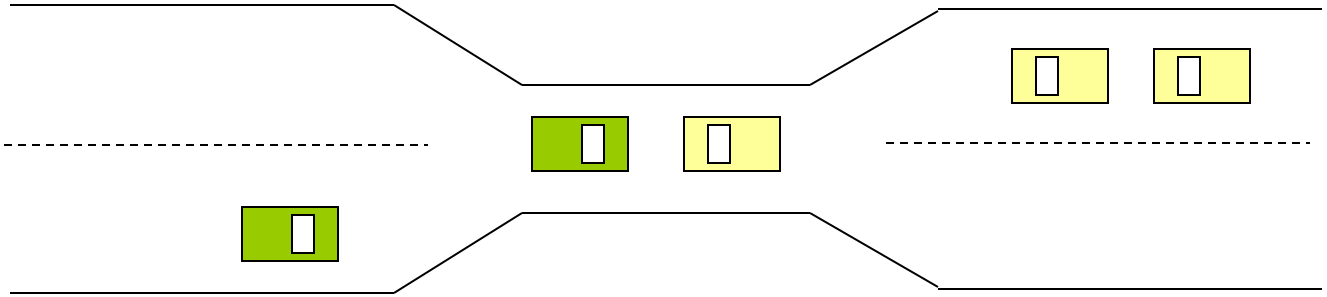
- **Definizione**: un gruppo di processi entra in uno stallo quando tutti i processi del gruppo attendono il rilascio di una risorsa che può essere rilasciata solo da uno dei processi in attesa

- Esempio con due processi e due risorse:
  - un processo P_a possiede una risorsa R_1 e richiede una risorsa R_2 già posseduta da un altro processo P_b; quest'ultimo a sua volta richiede l'uso di R_1

Propagazione del deadlock

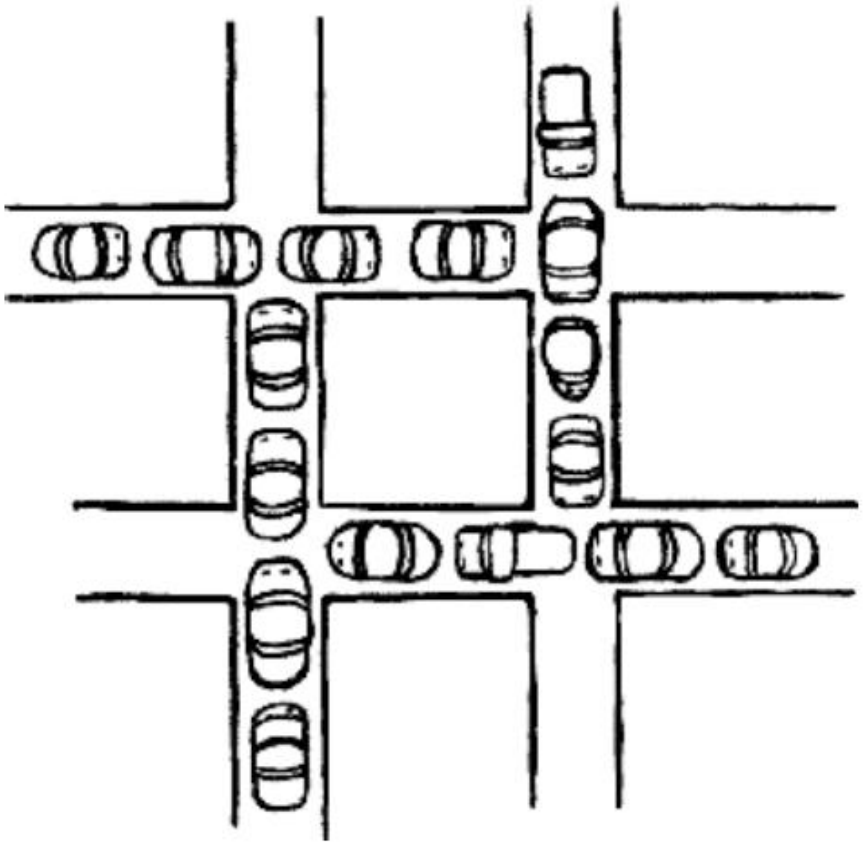
- Nuovi processi possono via via entrare indefinitamente in tale stato se le risorse richieste sono in possesso di altri processi a loro volta in stallo

Esempio "Incrocio sul ponte" (traffico in una sola direzione):



Propagazione del deadlock (ancora un altro esempio)

Esempio "Incrocio stradale":



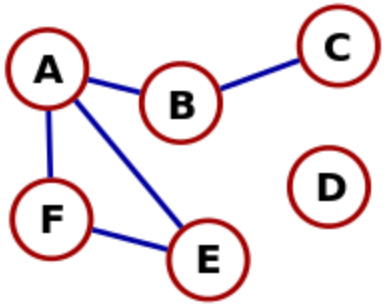
Caratterizzazione del deadlock

Si ha deadlock se si verificano simultaneamente le seguenti condizioni (*condizioni necessarie*):

1. Mutua esclusione (**mutual exclusion**)
2. Possesso e attesa (**hold & wait**)
3. No rilascio anticipato (**no preemption**)
4. Attesa circolare (**circular wait**)

Grafo di allocazione delle risorse (1)

- Grafo: G = (V,E):
  - Insieme di nodi V
  - Insieme di archi E: $E \subseteq V \times V$



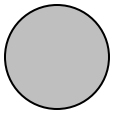
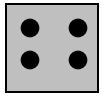
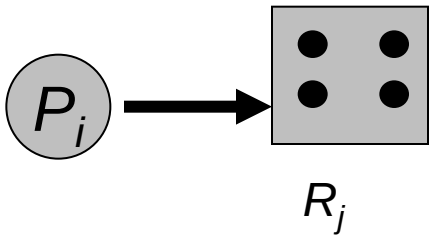
Grafo G di allocazione delle risorse:

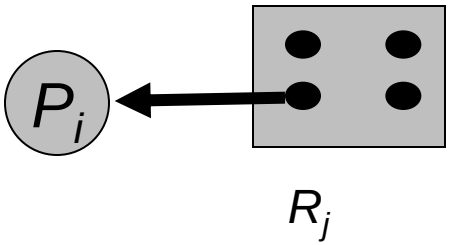
- Nodi V:
  - processi del sistema $P = \{P_1, P_2, \ldots, P_n\}$
  - risorse del sistema $R = \{R_1, R_2, \ldots, R_m\}$
    eventualmente con più istanze identiche

- Archi E:
  - arco di richiesta:
    da processo a risorsa $P_i \rightarrow R_j$
  - arco di assegnazione:
    da risorsa a processo $R_j \rightarrow P_i$

Grafo di allocazione delle risorse (2)

Notazione:

- Processo 
- Tipo di risorsa con 4 istanze 
- Arco di richiesta 

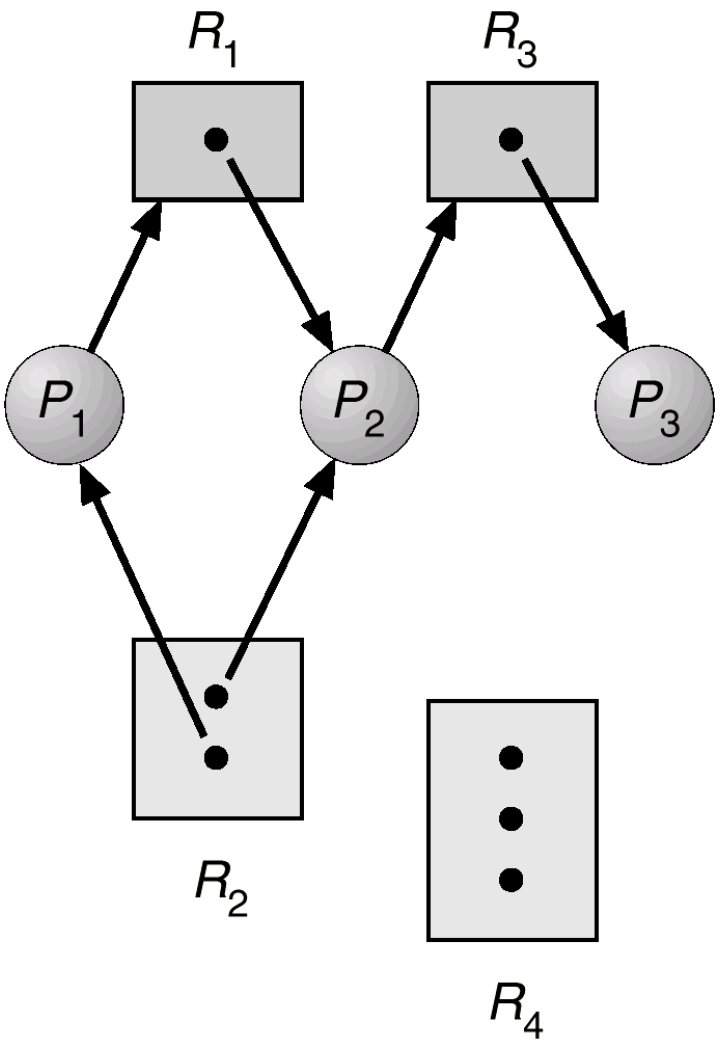
- Arco di assegnazione 

Punti chiave

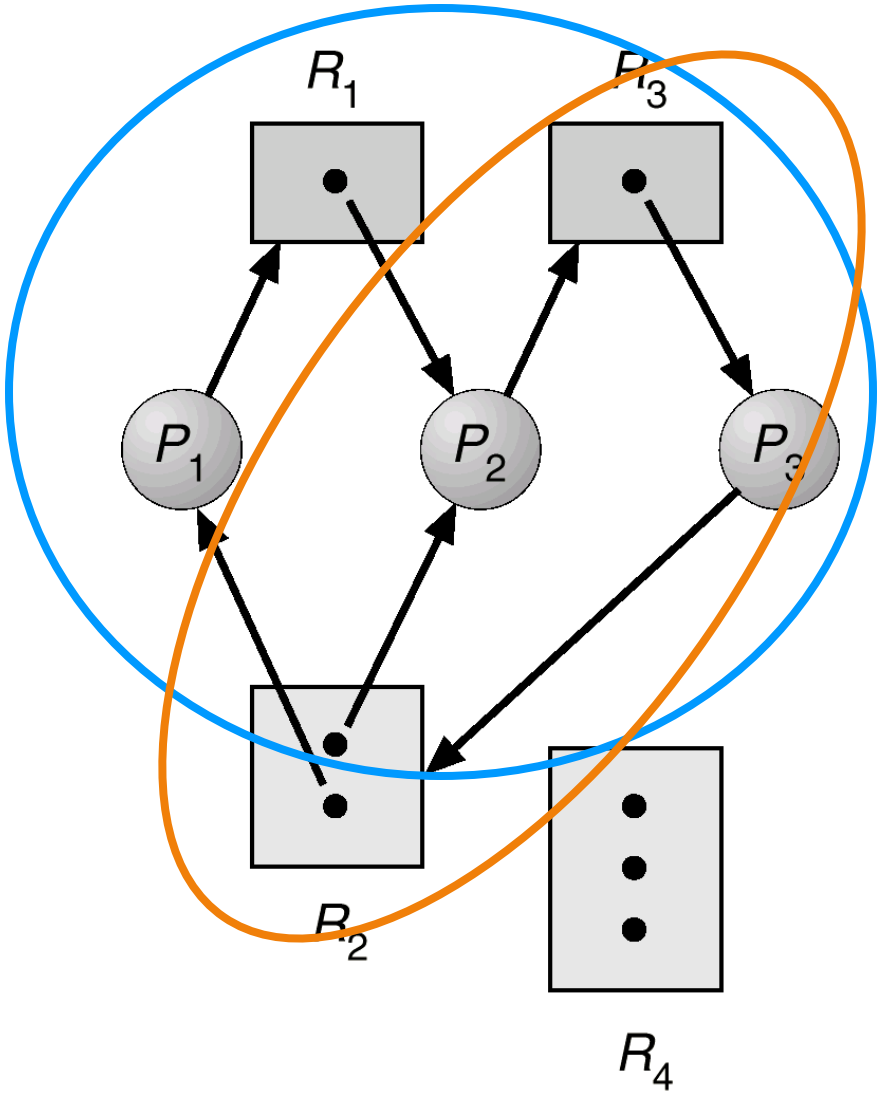
- Se il grafo non contiene cicli ⇒ nessun deadlock
- Se il grafo contiene cicli ⇒
  - se ogni tipo di risorsa nel ciclo ha una sola istanza, allora si ha deadlock
  - se ogni tipo di risorsa ha parecchie istanze, allora si ha possibilità di deadlock

Grafo di allocazione delle risorse (3)

a) Senza deadlock

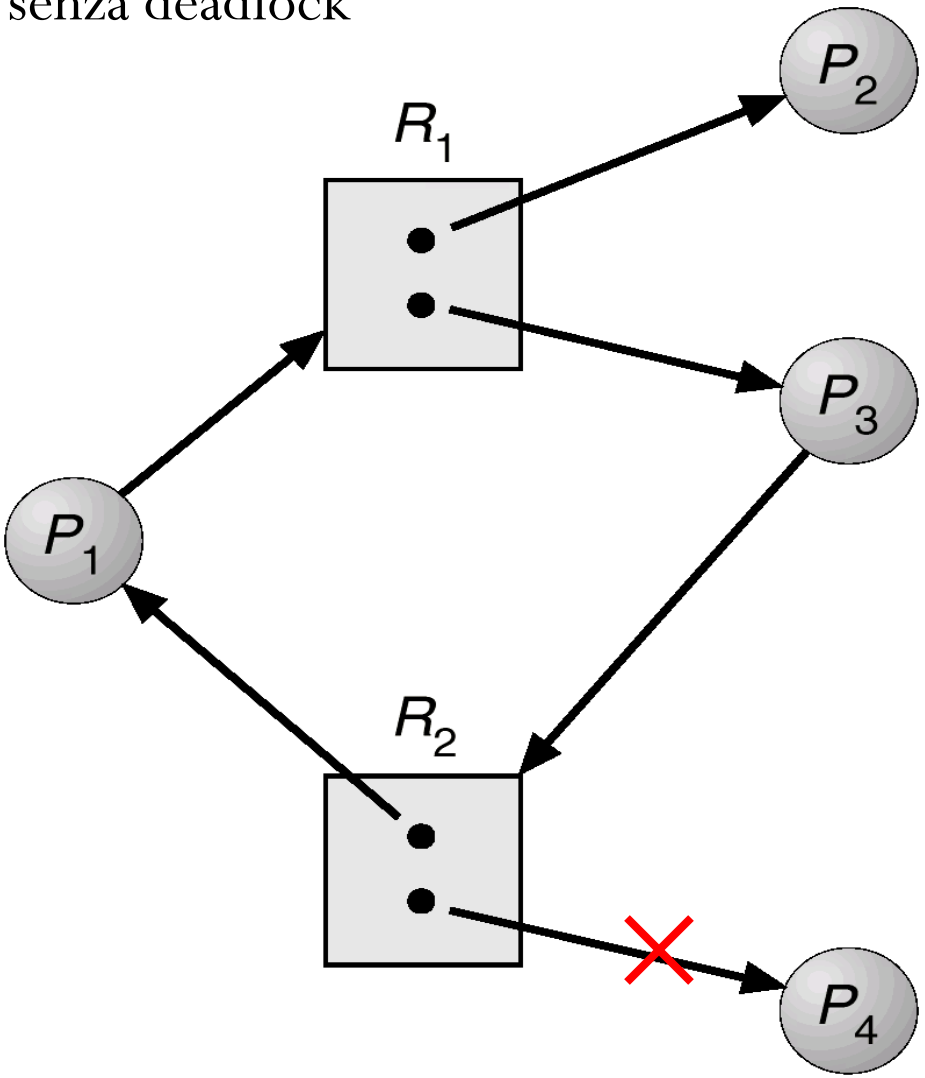


b) Con deadlock (due cicli)

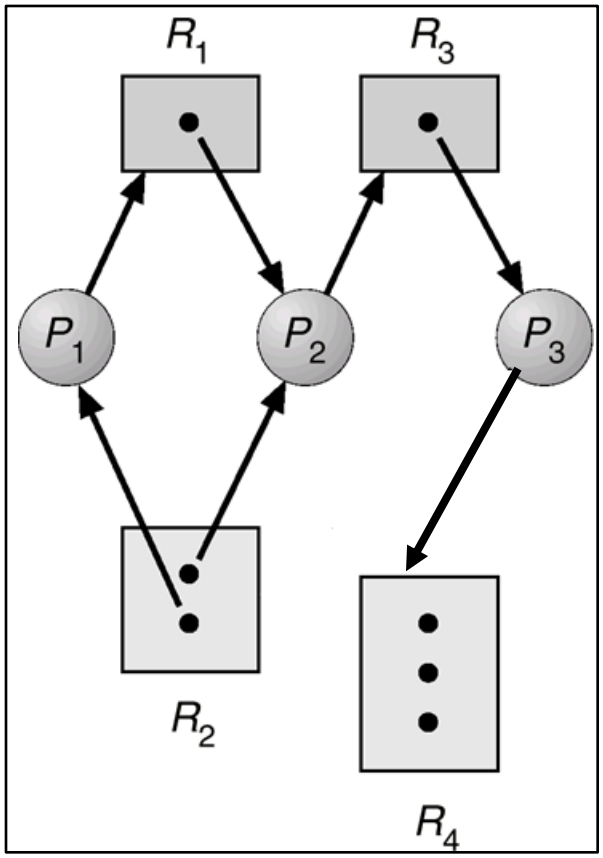


Grafo di allocazione delle risorse (4)

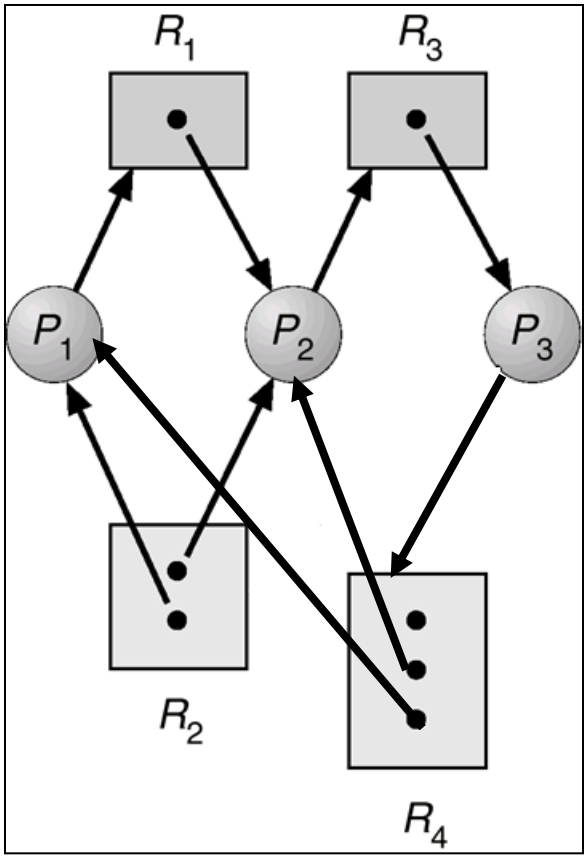
Con ciclo ma senza deadlock



Esercizi

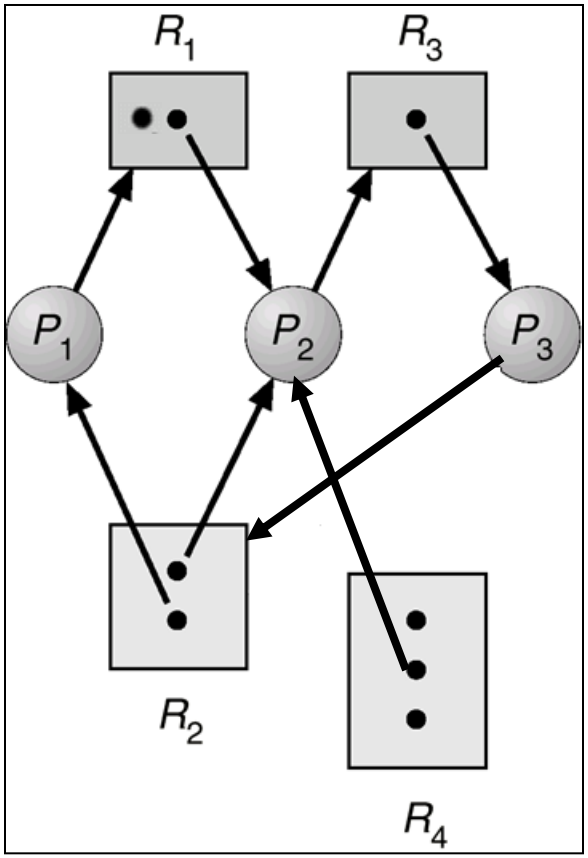


(a)

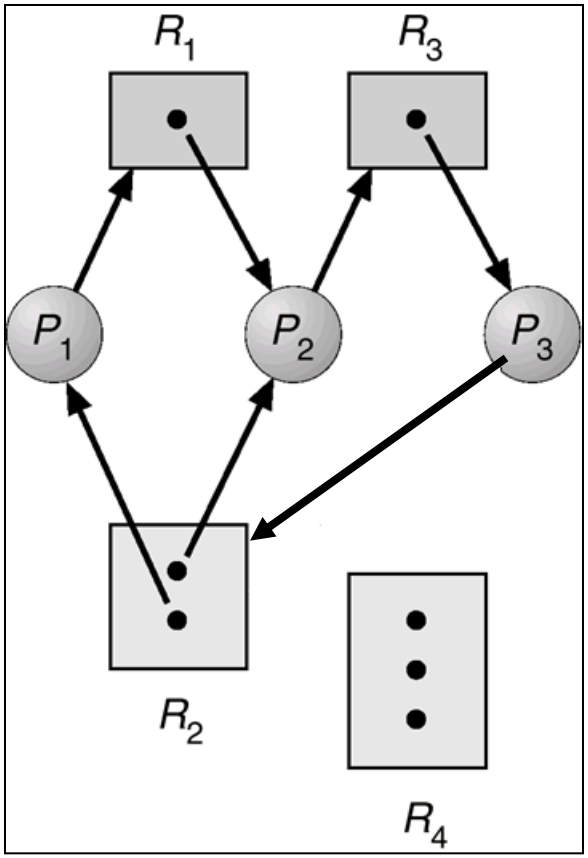


(b)

Esercizi



(c)



(d)

Metodi di gestione dei deadlock

1. **Ignorare** il deadlock
2. **Prevenzione** del deadlock
   (deadlock prevention)
3. **Evitare** il deadlock
   (deadlock avoidance)
4. **Rilevazione e recupero** del deadlock
   (deadlock detection & recovery)

Nessun metodo è ottimale, meglio una loro combinazione a seconda della "classe di risorsa"

1. Ignorare il deadlock

Algoritmo dello struzzo

- Applicato dalla maggior parte dei SO odierni (e dalla JVM)
  - Evita sovraccarico nel sistema: meno info e gestione
- Ma le cose stanno cambiando…per via delle applicazioni multi-thread

2. Prevenzione del deadlock

- **Principio**: prevenire il deadlock facendo sì che almeno una delle condizioni "necessarie" non sia soddisfatta:
  - Mutua esclusione
  - Possesso ed attesa
  - Nessun rilascio anticipato
  - Attesa circolare

Mutua esclusione

- La condizione può essere invalidata rimuovendola per le risorse intrinsecamente condivisibili
- La condizione non può mai essere invalidata per le risorse intrinsecamente non condivisibili

Possesso ed attesa (1)

- La condizione può essere invalidata garantendo che ogni volta che un processo chiede risorse, non possegga già qualche altra risorsa

Possesso ed attesa (2)

- Tecniche:
  1. Un processo chiede e ottiene tutte le risorse prima di iniziare l'esecuzione
  2. Un processo che possiede alcune risorse e vuole chiederne altre deve:
     - rilasciare tutte le risorse che possiede
     - chiedere tutte quelle che servono (incluse eventualmente anche alcune di quelle che già possedeva)
- Problemi:
  - Scarso utilizzo delle risorse (in quanto molte risorse vengono bloccate ma non usate)
  - Possibile starvation (un processo potrebbe attendere indefinitivamente delle risorse)

Possesso ed attesa (3)

Esempio: un processo deve leggere dei file da DVD, copiarli in un file e successivamente stamparli

- Tecnica 1: DVD, file e stampante vengono acquisiti prima di iniziare l'elaborazione
- Tecnica 2: il processo acquisisce DVD e file, finita la copia rilascia le risorse, dopodiché acquisisce file e stampante.

Nessun rilascio anticipato (1)

- Invalidare la condizione significa permettere il rilascio anticipato (preemption) per risorse il cui "stato di uso" all'atto del rilascio è *ripristinabile*

- Applicabilità: va bene per risorse il cui stato può essere facilmente salvato e ricaricato
  - Registri CPU e memoria centrale
  - Non va bene ad esempio per nastri o stampanti

Nessun rilascio anticipato (2)

- **Protocollo A:**
- Se un processo P detiene alcune risorse e ne chiede altre:
  - 1. Se **tutte** le risorse richieste sono **disponibili**, gli vengono assegnate
  - 2. Se **non tutte** le risorse richieste sono **disponibili**,
    - il processo richiedente rilascia tutte le risorse in possesso
    - le aggiunge alla lista delle risorse che attende
  - 3. Il processo viene riavviato solo quando può riottenere tutte le risorse

Nessun rilascio anticipato (3)

- **Protocollo B:**

  Se un processo P possiede alcune risorse e ne chiede altre:

  1. se tutte le risorse richieste sono **disponibili**, gli vengono assegnate
  2. se alcune delle risorse richieste non sono disponibili e sono **assegnate ad un processo Q in attesa** di ulteriori risorse, le risorse richieste e possedute da Q vengono
     - rilasciate anticipatamente e assegnate al processo richiedente P
     - inserite tra quelle per cui il processo Q è in attesa
  3. se alcune risorse richieste **non** sono **disponibili e non assegnate a processi in attesa** di altre risorse, il processo richiedente P deve
     - attendere che si liberino
     - ripartire quando ottiene tutte le risorse necessarie

Attesa circolare (1)

- La condizione può essere invalidata impedendo che si creino attese circolari

Un modo consiste nell'imporre che ogni processo chieda le risorse in un *ordine incrementale*

Attesa circolare (2)

- **Tecniche**:
  - Un ordinamento globale univoco viene imposto su tutti i tipi di risorsa R_i
  - Se un processo chiede k istanze della risorsa R_j e detiene solo risorse R_i con $i<j$,
    - se le k istanze della risorsa R_j sono disponibili gli vengono assegnate
    - altrimenti, il processo deve attendere

- **Un processo non potrà mai chiedere istanze della risorsa R_j se detiene risorse R_i con $i>j$**

Attesa circolare (3)

- **Tecniche** (cont.):
  - Se un processo chiede k istanze della risorsa R_j e detiene risorse R_i con $i \geq j$, il processo deve
    - rilasciare tutte le istanze delle risorse R_i
    - chiedere le k istanze della risorsa R_j
    - chiedere le istanze delle risorse R_i ($i>j$) che deteneva precedentemente
- Ordinare le risorse non è sufficiente per evitare l'attesa circolare
  - I programmatori devono scrivere i programmi in modo che l'ordinamento sia rispettato
  - Tipicamente le risorse si ordinano in base all'utilizzo
    - f(disco) > f(stampante) poiché tipicamente si usa prima il disco che la stampante
- Esistono, ad es., alcuni *programmi verificatori dell'ordine dei lock mutex* in un sistema come **Witness** ("testimone") di UNIX BSD

3. Evitare il deadlock

- Verificare a priori se la sequenza di richieste e rilasci di risorse effettuate da un processo porta al deadlock, tenendo conto delle sequenze dei processi già accettati nel sistema

- Obiettivi:
  - Alto sfruttamento delle risorse
  - Alta efficienza del sistema
  - Semplicità di gestione

Informazioni per evitare il deadlock

- Necessità di informazioni a priori sullo stato di allocazione delle risorse e sul comportamento dei processi:
  - il *numero massimo* di risorse per ogni processo
  - risorse disponibili
  - risorse assegnate
  - richieste e rilasci futuri di risorse

Stato di allocazione
(MAX,disponibili,allocate)

Algoritmi per evitare il deadlock

- Un algoritmo generico "per evitare i deadlock" esamina dinamicamente lo stato di allocazione delle risorse per accertarsi che la condizione di attesa circolare non possa mai verificarsi

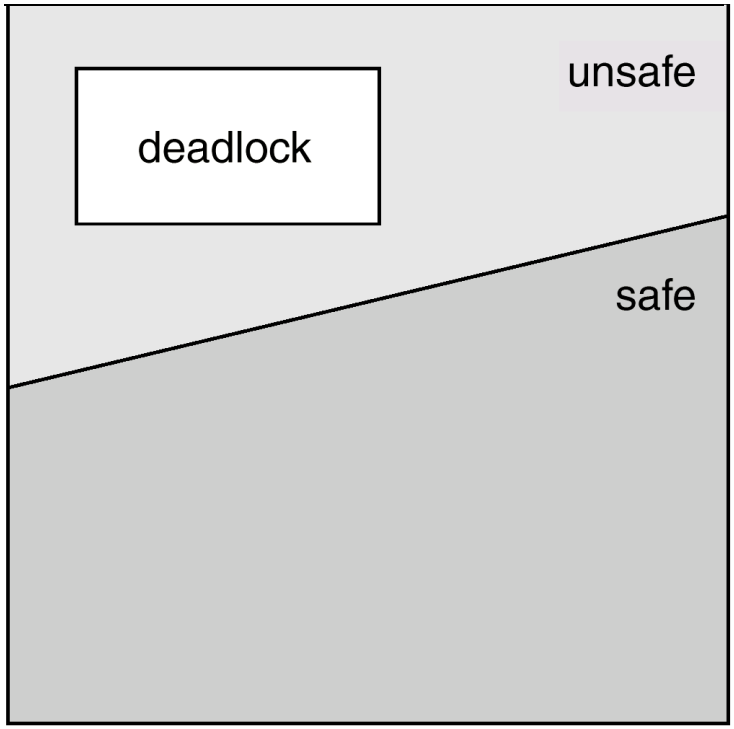
- Due algoritmi:
  - **Algoritmo del grafo di allocazione delle risorse** (per un singolo tipo di risorsa) [Dijkstra, 1965]
  - **Algoritmo del banchiere** [Habermann, 1969] (per risorse multiple)

  Entrambi basati sul concetto di **"stato sicuro"**

Stato sicuro (1)

- **Def.**: Uno stato si dice sicuro se il sistema può allocare le risorse richieste da ogni processo in un certo ordine garantendo che non si verifichi deadlock

  - stato sicuro
    $\Rightarrow$ no deadlock
  - stato non sicuro
    $\Rightarrow$ deadlock possibile



Stato sicuro (2)

- **Def.** :Uno stato è sicuro se esiste una *sequenza di processi sicura*

- **Def**. : Una sequenza di processi $<P_1, P_2, \ldots, P_n>$ è una **sequenza sicura** per l'allocazione corrente se le richieste che ogni processo P_i può fare possono essere soddisfatte dalle **risorse attualmente disponibili più** tutte le risorse **detenute dai processi P_j con $j<i$**

Stato sicuro (3)

- Si considerino i seguenti processi all'istante t_0 in un sistema che può fornire 12 istanze di una risorsa

t_o	Richieste max.	Unità possedute
P_0	10	5
P_1	4	2
P_2	9	2

- All'istante t_0 il sistema è in uno stato sicuro (3 ist. libere)
  - Sequenza P_1, P_0, P_2
    - P_1 riceve tutte le istanze necessarie (2) dopodiché le rilascia (5 istanze libere)
    - P_0 riceve tutte le istanze necessarie (5) dopodiché le rilascia (10 istanze libere)
    - P_2 riceve tutte le istanze necessarie (7) dopodiché le rilascia (12 istanze libere)

Stato sicuro (4)

- Un sistema può passare da uno stato sicuro ad uno insicuro
- Si consideri il sistema precedente nel quale all'istante t_1 (P1 ha tutte le istanze necessarie) P2 richiede e riceve un'ulteriore istanza

t_1	Richieste max.	Unità possedute
P_0	10	5
P_1	4	4
P_2	9	3

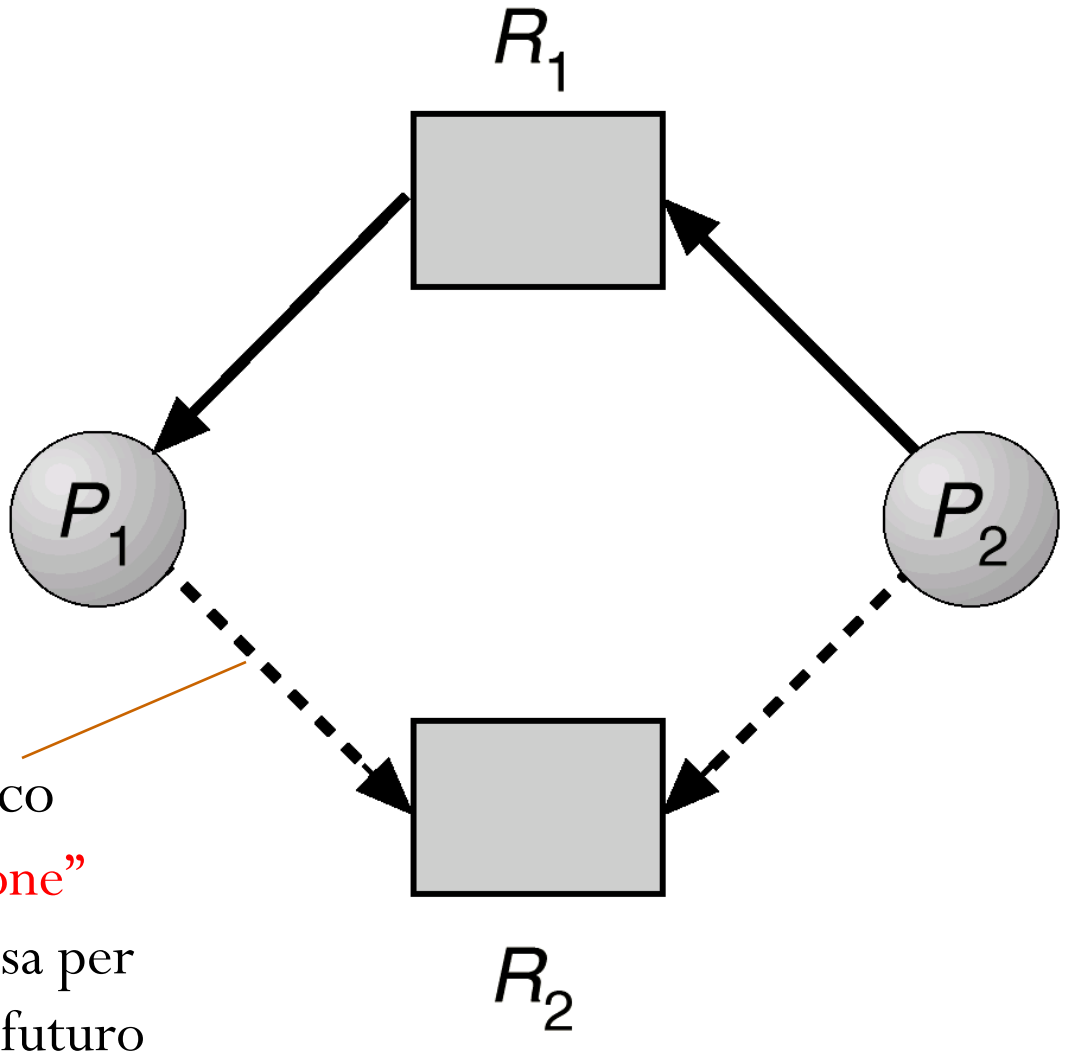
- Quanto P_1 rilascerà le istanze acquisite ci saranno 4 istanze libere e nessuno dei due processi potrà ricevere il massimo di istanze necessarie

Come evitare il deadlock?

- **IDEA**: Garantire che il sistema passi da uno stato sicuro ad un altro stato sicuro quando un processo chiede una nuova risorsa

- **Schema**:
  - Si parte da uno stato iniziale sicuro
  - Una richiesta di risorsa viene soddisfatta se la risorsa è disponibile **e** se il sistema va in uno stato sicuro
  - Se la risorsa non è disponibile, il processo deve attendere

Algoritmo del grafo di allocazione delle risorse (1)



- Nuovo tipo di arco
  "arco di prenotazione"
  da processo a risorsa per qualche momento futuro

Algoritmo del grafo di allocazione delle risorse (3)

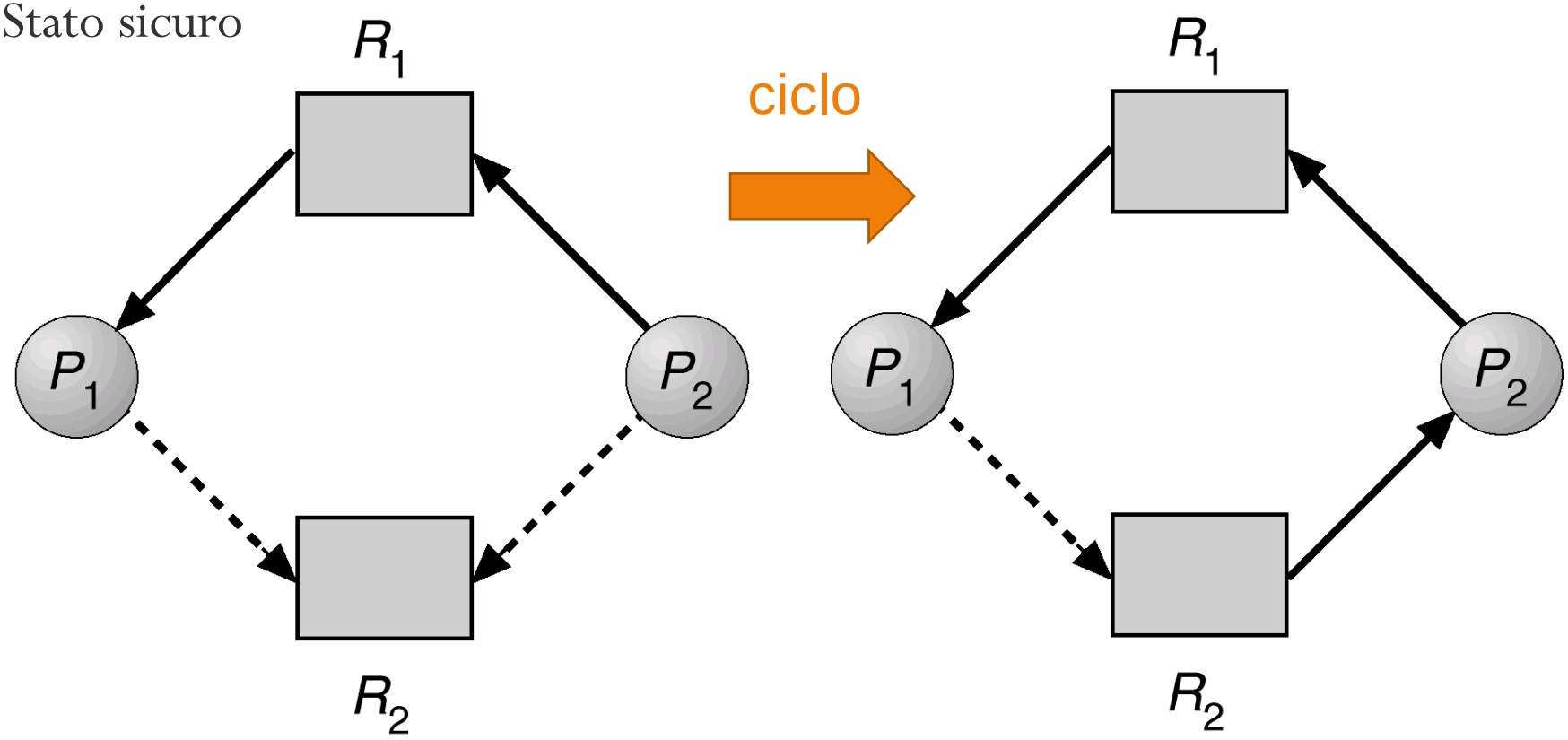
- Inizialmente si costruisce il grafo di allocazione con archi di prenotazione
- Poi, applichiamo le *trasformazioni di archi* :
  - Quando un processo chiede una risorsa, l'arco di prenotazione è convertito in un arco di richiesta
  - Quando una risorsa è assegnata, l'arco di richiesta è convertito in uno di assegnazione
  - Quando una risorsa è liberata dal processo, l'arco di assegnazione è convertito in un arco di prenotazione
- **Se si evidenziano cicli nel grafo --** verifica in **$O(n^2)$ --**, **lo stato non è sicuro** e quindi non si può accettare la richiesta di risorse dell'ultimo processo inserito

Algoritmo del grafo di allocazione delle risorse (4)

- Esempio: Supponiamo che P2 chieda R2

Stato sicuro

Stato non sicuro



Algoritmo del banchiere (1)

- Gestisce istanze multiple delle risorse
- È meno efficiente dell'algoritmo del grafo di allocazione delle risorse
- Il numero massimo di istanze deve essere dichiarato a priori
- Un processo deve restituire in un tempo finito le risorse utilizzate

Algoritmo del banchiere (2)

Strutture dati

- m — # tipi di risorse
- n — # processi
- Available[j] — # risorse disponibili per tipo j
- Max[i,j] — max richiesta per processo i e tipo risorsa j
- Allocation[i,j] — # di risorse j attualmente assegnate ad i
- Need[i,j] — # risorse di tipo j che il processo i può richiedere

(Need [i,j] = Max[i,j] – Allocation [i,j])

Algoritmo del banchiere (3)

Procedura di verifica dello stato sicuro $O(mn^2)$

1. Work[1..m]
2. Finish[1..n]
3. Work := Available; Finish[i] := false per i = 1 .. n
4. Iterativamente:
   1. Se Finish[i] == true per ogni i, termina in stato SICURO
   2. Cerca i tale che:
      a. Finish[i] == false &&
      b. Work ≥ Need[i,*]
   3. Se nessuna i soddisfa (a) e (b), termina in stato NN SICURO
   4. Work := Work + Allocation[i,*]
   5. Finish[i] := true
   6. Passa all'iterazione successiva

Algoritmo del banchiere (4)

Procedura di richiesta delle risorse

- Sia **Request[i,j]** = numero di risorse di tipo j richieste dal processo i

1. Se esiste j tale per cui **Request[i,j] > Need[i,j]**
   Solleva errore: i ha ecceduto il num. max di richieste dichiarato
2. Se esiste j tale per cui **Request[i,j] > Available[j]**
   i deve attendere: risorse non disponibili
3. Si ipotizzi di assegnare le risorse richieste aggiornando lo stato:

   Available := Available – Request[i,*]

   Allocation[i,*] := Allocation[i,*] + Request[i,*]

   Need[i,*] := Need[i,*] – Request[i,*]

   e di lanciare la procedura di verifica dello stato

   se il risultato è «stato SICURO» → allocazione consentita

   se il risultato è «stato NN SICURO» → allocazione non consentita, processo in wait

Algoritmo del banchiere -- Esempio

- Si consideri un sistema con 5 processi (da P_0 a P_4)
  3 tipi di risorse:

  A (10 istanze), B (5 istanze) e C (7 istanze)

- Supponiamo che all'istante T_0 lo stato sia:

	Allocation	*Max*	*Available*
	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Algoritmo del banchiere – Esempio (cont.)

- Il contenuto della matrice *Need* è *Max* – *Allocation*

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- Il sistema è in uno stato sicuro poichè la sequenza **< P_1, P_3, P_4, P_2, P_0> soddisfa i criteri di sicurezza**
  [La verifica è lasciata come esercizio]

Algoritmo del banchiere – Esempio di richiesta di P1 (1,0,2)

- Controlliamo che **Request$_1$ ≤Need$_1$ Request$_1$≤ Available $_1$**

	Allocation	*Need*	*Available*	
	A B C	*A B C*	*A B C*	
P_0	0 1 0	7 4 3	2 3 0	=(3,3,2) - (1,0,2)
P_1	3 0 2	0 2 0		=(1,2,2) - (1,0,2)
P_2	3 0 1	6 0 0		
P_3	2 1 1	0 1 1		=(2,0,0) + (1,0,2)
P_4	0 0 2	4 3 1		

- L'esecuzione dell'algoritmo di verifica dello stato sicuro mostra che la sequenza < P_1, P_3, P_4, P_0, P_2> soddisfa i criteri
  [Dimostrazione: lasciata come esercizio]
- La richiesta (3,3,0) di P_4 può essere soddisfatta? No, perché non ci sono risorse disponibili.
- La richiesta (0,2,0) di P_0 può essere soddisfatta? No, perché porta ad uno stato non sicuro *[Dimostrazione: lasciata come esercizio]*

4. Rilevazione e recupero

- Senza algoritmi di prevenzione o per evitare il deadlock, tale situazione può verificarsi

- Il sistema deve essere in grado di:

  - **rilevare** la presenza di situazioni di deadlock **dopo** che sono avvenute

  - **ripristinare** una situazione di corretto funzionamento eliminando il deadlock

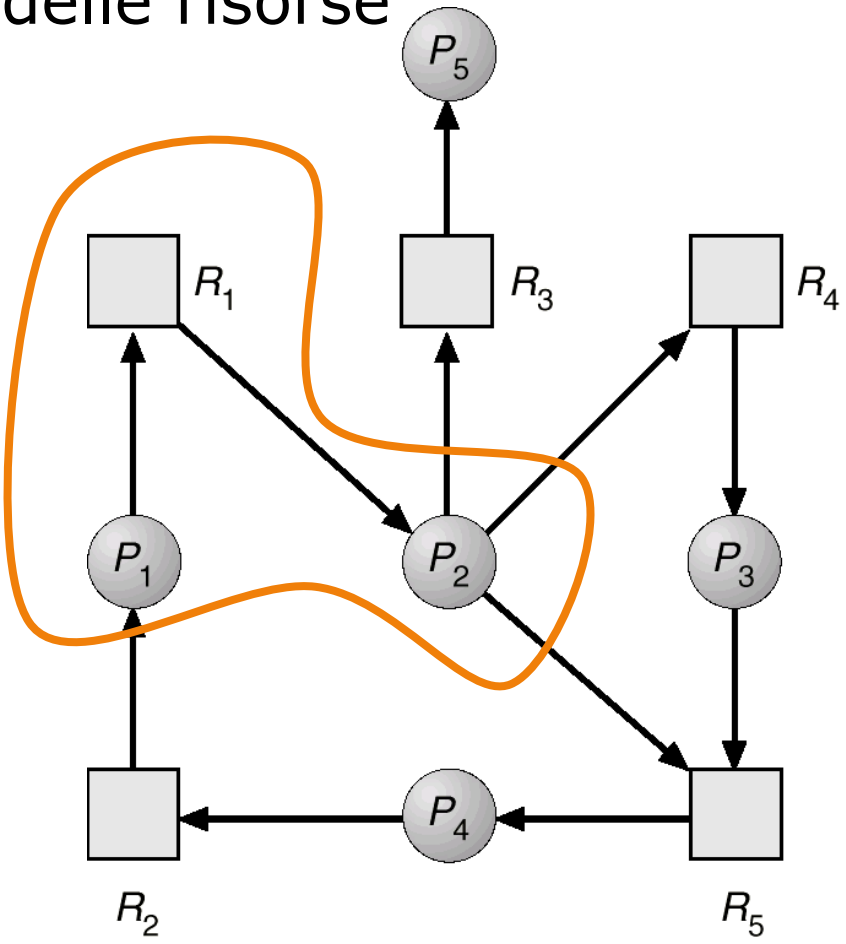
Sistemi con istanze singole o multiple delle risorse

Rilevazione del deadlock

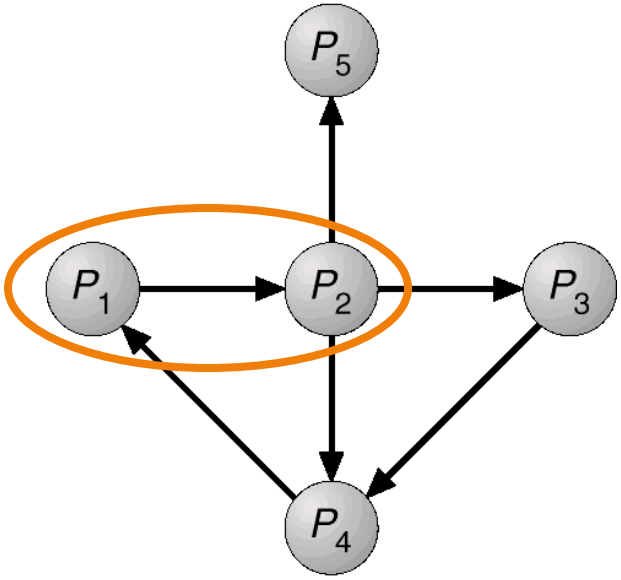
- Algoritmi di rilevamento :
  - Solo singole istanze: *grafo di attesa*
  - Istanze multiple: algoritmo completo [Coffman *et al.*]

Rilevazione con istanze singole delle risorse (1)

Grafo di allocazione delle risorse



Grafo di attesa (wait-for)



Rilevazione con istanze singole delle risorse (2)

- Mantenimento e analisi "periodica" del grafo di attesa
- Se il grafo di attesa contiene cicli -- verifica in tempo $O(n^2)$ -- si ha deadlock
- I processi in deadlock sono quelli coinvolti in ciascun ciclo presente nel grafo

Algoritmo del banchiere (2)

Strutture dati

- m — # tipi di risorse
- n — # processi
- Available[j] — # risorse disponibili per tipo j
- Allocation[i,j] — # di risorse j attualmente assegnate ad i
- Request[i,j] — # risorse di tipo j di cui il processo i è attualmente in attesa

Algoritmo del banchiere (3)

Procedura di rilevazione di deadlock $O(mn^2)$

1. Work[1..m]; Finish[1..n]
2. Work := Available;
3. Per ogni processo i:
   se Allocation[i,*] = 0 allora Finish[i] := true,
   altrimenti Finish[i] := false
4. Iterativamente:
   1. Se Finish[i] == true per ogni i, termina in stato SICURO
   2. Cerca i tale che:
      a. Finish[i] == false &&
      b. Work ≥ Request[i,*]
   3. Se nessuna i soddisfa (a) e (b), termina in stato DEADLOCK
   4. Work := Work + Allocation[i,*]; Finish[i] := true;
   5. Passa all'iterazione successiva

Esempio

- Cinque processi da P_0 a P_4; tre tipi di risorse A (7 instanze), B (2 instanze), e C (6 instanze)

- Supponiamo che **a tempo T_0**:

	Allocation	*Request*	*Work=Available*	*Finish*
	A B C	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	0 0 0	false
P_1	2 0 0	2 0 2		false
P_2	3 0 3	0 0 0		false
P_3	2 1 1	1 0 0		false
P_4	0 0 2	0 0 2		false

Esempio

- Cinque processi da P_0 a P_4; tre tipi di risorse A (7 instanze), B (2 instanze), e C (6 instanze)

- Supponiamo che **a tempo T_0**:

	Allocation	*Request*	*Work=Available*	*Finish*
	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$	
P_0	0 1 0	0 0 0 <= 0 0 0		false
P_1	2 0 0	2 0 2		false
P_2	3 0 3	0 0 0		false
P_3	2 1 1	1 0 0		false
P_4	0 0 2	0 0 2		false

Esempio (Cont.)

- Eseguendo il **passo 4 per P0**:

	Allocation	*Request*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	0 1 0	true
P_1	2 0 0	2 0 2		false
P_2	3 0 3	0 0 0		false
P_3	2 1 1	1 0 0		false
P_4	0 0 2	0 0 2		false

Esempio (Cont.)

- Eseguendo il **passo 4 per P2**:

	Allocation	*Request*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	0 1 0	true
P_1	2 0 0	2 0 2		false
P_2	3 0 3	0 0 0 <=		false
P_3	2 1 1	1 0 0		false
P_4	0 0 2	0 0 2		false

Esempio (Cont.)

- Eseguendo il **passo 4 per P2**:

	Allocation	*Request*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	3 1 3	true
P_1	2 0 0	2 0 2		false
P_2	3 0 3	0 0 0		true
P_3	2 1 1	1 0 0		false
P_4	0 0 2	0 0 2		false

- E così via …
- La sequenza **<P_0, P_2, P_3, P_1, P_4> darà Finish[i] = true per ogni i**

Esempio (Cont.)

- Supponiamo che P_2 faccia invece una richiesta per C

	Request	*Work*
	A B C	*A B C*
P_0	0 0 0	0 1 0
P_1	2 0 2	
P_2	0 0 1	
P_3	1 0 0	
P_4	0 0 2	

Stato del sistema?

- Anche se possiamo riprendere le risorse del processo P_0, *il numero* di risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi
- C'è **deadlock formato da P_1, P_2, P_3,** e P_4

Applicazione della rilevazione

- Quando invocare l'algoritmo di rilevazione?
  - **Ogni volta che una richiesta di allocazione non può essere soddisfatta** immediatamente
    - rilevazione immediata
    - pochi risorse e processi bloccati
    - considerevole sovraccarico computazionale
  - **A intervalli di tempo prestabiliti**
    - minor sovraccarico computazionale
    - rilevazione più complessa
    - molte risorse e processi possono essere bloccati da tempo

Ripristino del deadlock (1)

Terminare processi in deadlock

1. **Abortire tutti i processi in deadlock**
   - Troppi processi terminati
   - Spreco di risorse computazionali
   - Costo elevato

2. **Abortire un processo alla volta fino a eliminare il deadlock**
   - Pochi processi terminati
   - Algoritmo di rilevazione invocato più volte

Ripristino del deadlock (2)

Terminare processi in deadlock

- Ordine di eliminazione dei processi:
  - priorità del processo
  - tempo di elaborazione del processo
  - tipo di risorse utilizzate
  - risorse richieste per terminare l'elaborazione
  - numero dei processi da terminare
  - processo interattivo o batch

Ripristino del deadlock (3)

Rilascio anticipato delle risorse

Selezione della vittima (processo o risorsa)

- la vittima deve essere quella a "costo minimo"

- Rollback del processo (e risorse)
  - all'ultimo stato sicuro, o
  - totale
- Starvation
  - non selezionare sempre la stessa vittima