DEADLOCK

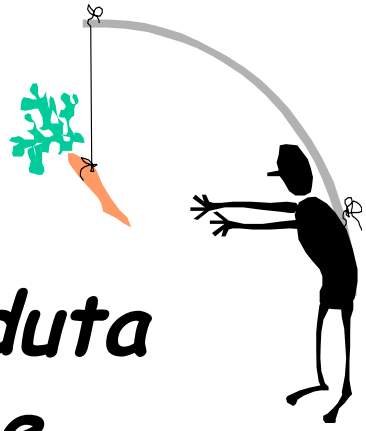
Il problema del deadlock

Esiste un ***insieme di processi*** bloccati, ognuno:

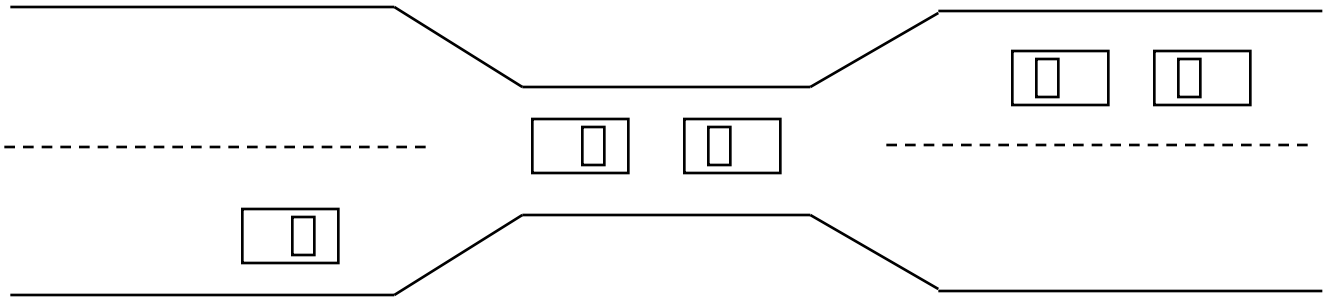
- ***in possesso di una o piu' risorse***

e

- ***in attesa di una risorsa posseduta da un altro processo dell'insieme***



L'attraversamento di un ponte…



- Il traffico sul ponte e' a ***senso unico alternato***
- Ogni ***tratto del ponte*** (es. 100 mt.) puo' essere visto come una ***risorsa***
- In caso di deadlock uno o piu' veicoli possono essere ***ricondotti indietro*** per sbloccare la situazione
- Comunque e' possibile che si verifichi ***starvation***

Di nuovo sui processi…

Definizione formale :

Un insieme di processi e' in deadlock se ogni processo nell' insieme sta attendendo un evento che puo' essere causato solo da un altro processo in quell' insieme (di solito il rilascio di una risorsa)

… un esempio…

Un sistema di calcolo ha ***due unita' a disco*** :

P1 ***mantiene il possesso*** di un disco ed ***ha bisogno*** dell' altro
P2 fa lo stesso

I semafori *A* e *B*, inizializzati a 1, sono di ***mutua esclusione*** sulle due unita'

P1	P2
wait (A)	*wait(B)*
wait (B)	*wait(A)*

P1 : wait(A)
P2 : wait(B)

Deadlock!

… uno piu' complicato…

A

Request R
Request S
Release R
Release S

(a)

B

Request S
Request T
Release S
Release T

(b)

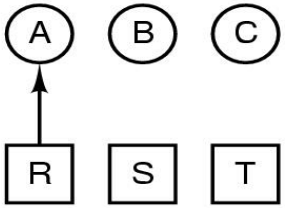
C

Request T
Request R
Release T
Release R

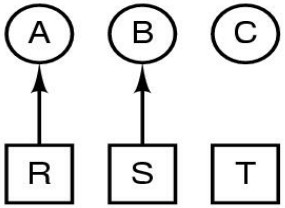
(c)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock

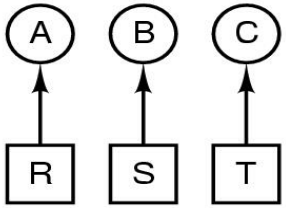
(d)



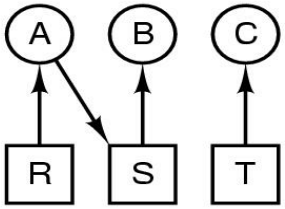
(e)



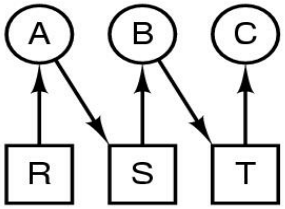
(f)



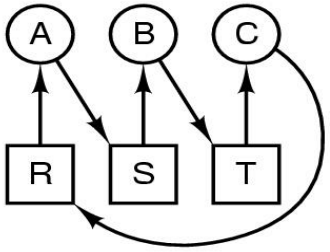
(g)



(h)



(i)



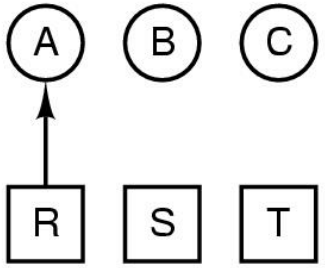
(j)

… che poteva anche non accadere!

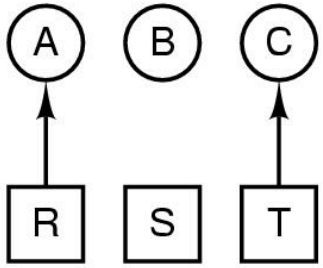
1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S

no deadlock

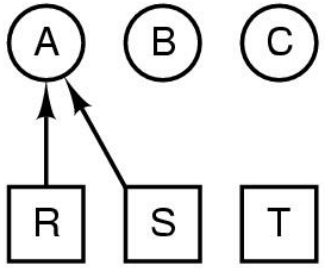
(k)



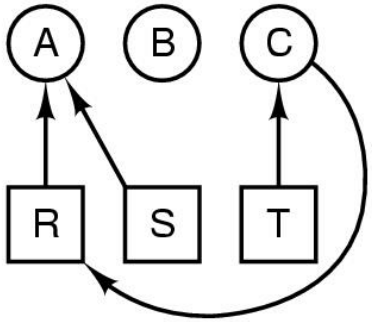
(l)



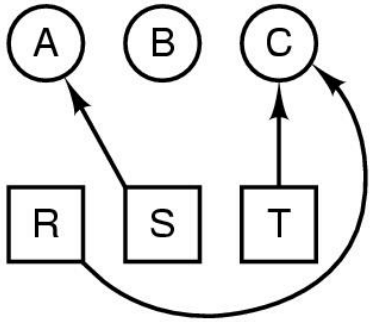
(m)



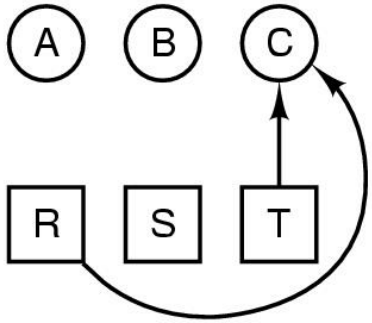
(n)



(o)



(p)



(q)

Caratterizzazione del deadlock…

Un deadlock puo' avvenire ***solo se*** le seguenti condizioni valgono ***simultaneamente*** :

1. **Mutua esclusione**
2. **Possesso e attesa**
3. **Assenza di preemption**
4. **Attesa circolare**

⟹ ***Condizioni necessarie !!!***

… e in dettaglio

A. **Mutua esclusione:** solo un processo alla volta puo' utilizzare una (istanza di una) risorsa

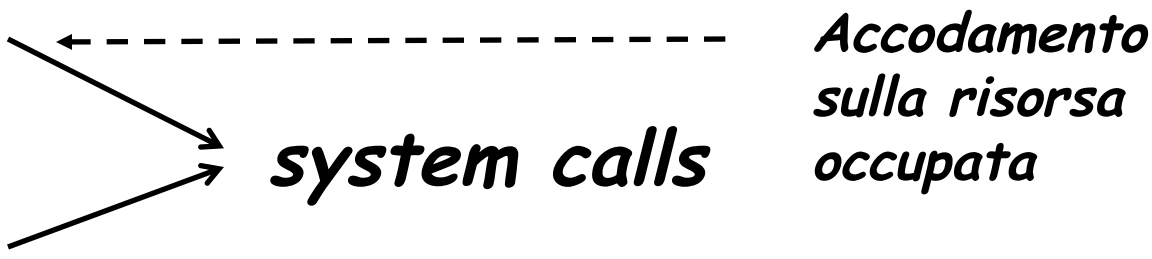
B. **Possesso e attesa:** un processo che possiede almeno una risorsa puo' stare in attesa di acquisire risorse aggiuntive possedute da altri processi

C. **Assenza di preemption:** una risorsa puo' essere rilasciata da un processo che la possiede solo in modo volontario, dopo che il processo ha completato il suo compito

D. **Attesa circolare:** puo' esistere un insieme {P0, P1, …, Pn} di processi in attesa tale che P0 sta attendendo una risorsa posseduta da P1, P1 sta attendendo una risorsa posseduta da P2, …, Pn–1 sta attendendo una risorsa posseduta da Pn, e Pn sta attendendo una risorsa posseduta da P0

Un modello del sistema di calcolo

- *m* ***tipi di risorse*** $R_1, R_2, \ldots, R_m$
  Cicli di CPU, spazio di memoria, dispositivi di I/O, etc.
- Ogni tipo di risorsa R_i ha W_i ***istanze***
- *n* ***processi*** $P_1, P_2, \ldots, P_n$ - ogni processo, per utilizzare una risorsa, esegue :
  - *request*
  - use
  - *release*



Rappresentazione del modello: grafo di allocazione risorse…

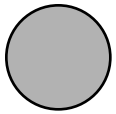
Un grafo e' costituito da un insieme di ***vertici V*** e un insieme di ***archi E***

In un ***grafo di allocazione risorse*** (struttura dinamica):

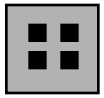
- ***V*** e' partizionato in due sottinsiemi di vertici:
  - $P = \{P_1, P_2, \ldots, P_n\}$, l' insieme dei ***processi***
  - $R = \{R_1, R_2, \ldots, R_m\}$, l' insieme dei ***tipi di risorse***
- ***E*** e' partizionato in due sottinsiemi di archi:
  - Archi di ***richiesta*** $P_i \rightarrow R_j$
  - Archi di ***assegnamento*** $R_j \rightarrow P_i$

… la notazione…

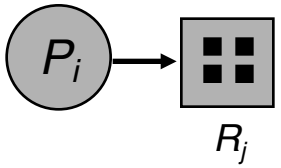
- Processo



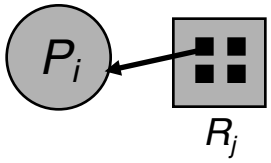
- Tipo di risorsa con istanze



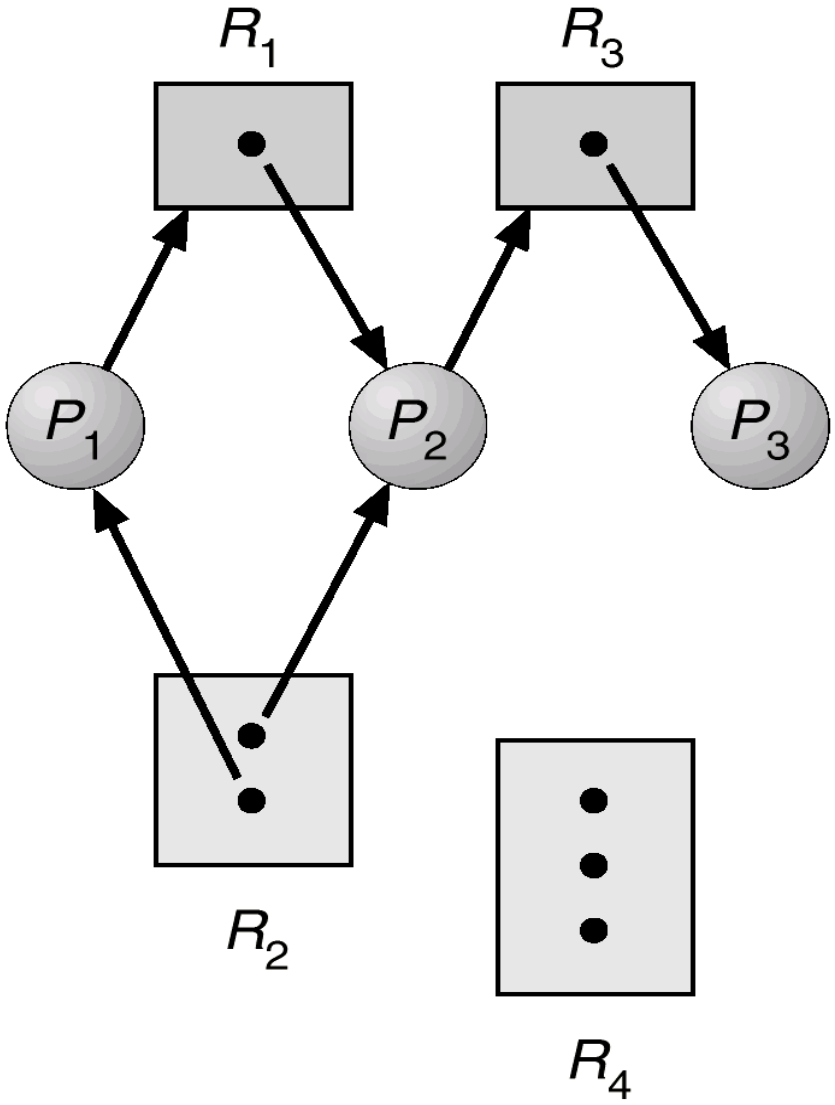
- P_i richiede una istanza di tipo R_j



- P_i possiede una istanza di tipo R_j



… ed un esempio

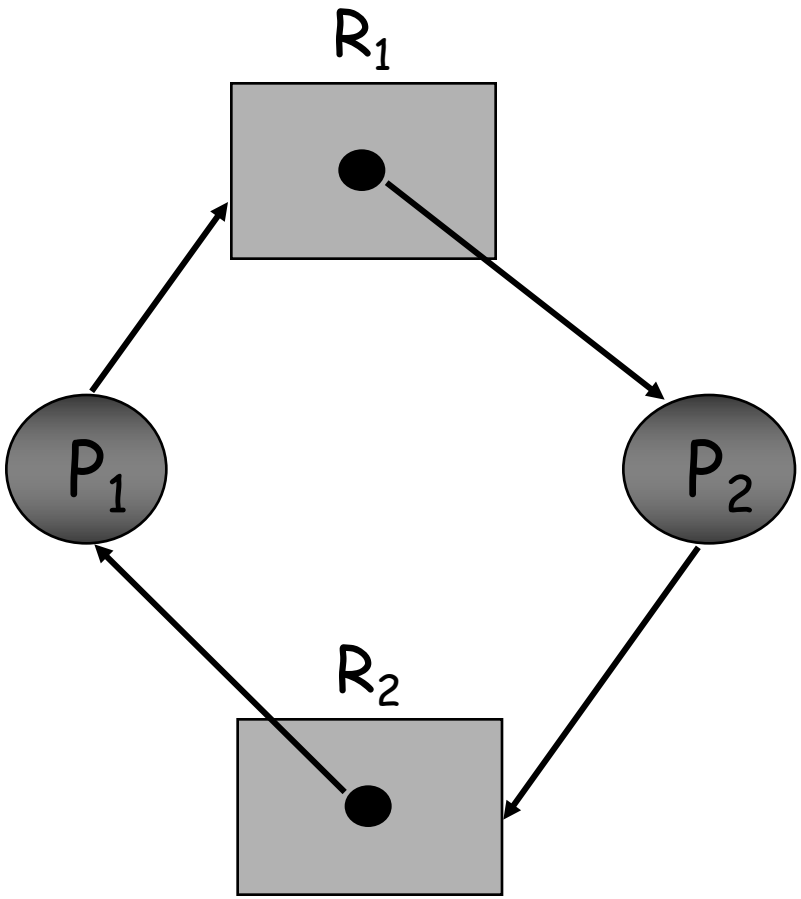


Condizioni per deadlock in un grafo di allocazione risorse

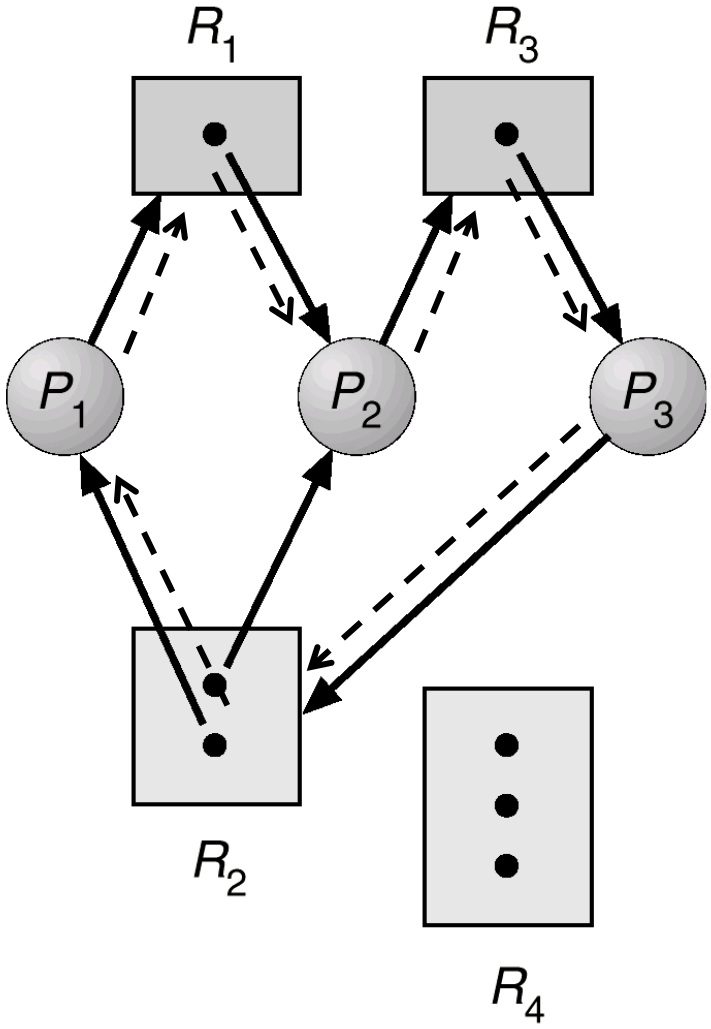
La presenza di un ***ciclo*** in un grafo di allocazione risorse con ***un'unica istanza*** per ogni tipo di risorsa ***implica*** deadlock

La presenza di un ***ciclo*** in un grafo di allocazione risorse con ***multiple istanze*** per ogni tipo di risorsa ***puo' implicare*** deadlock

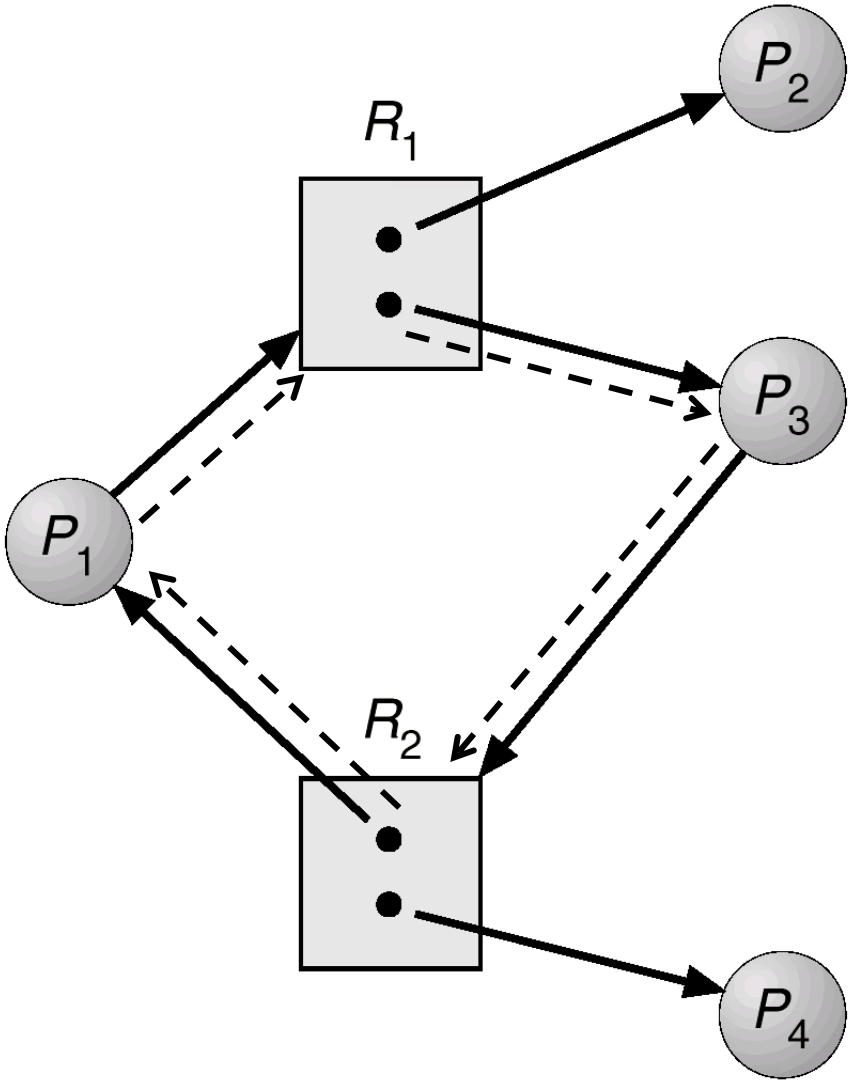
Grafo di allocazione risorse con deadlock (unica istanza)



Grafo di allocazione risorse con deadlock (multiple istanze)…

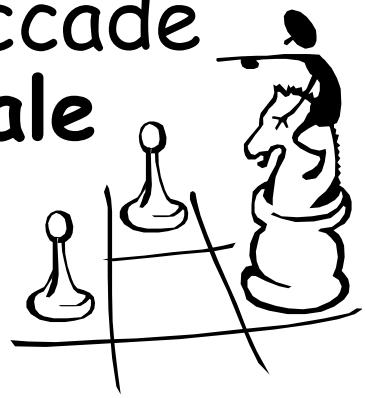


… e grafo con ciclo ma senza deadlock (multiple istanze)



Come trattare il deadlock…

- Costruire il sistema in modo che **non possano esistere stati di deadlock** (***prevenire***)
- Garantire che il sistema **non entri mai in uno stato di deadlock** (***evitare***)
- Permettere che il sistema entri in uno stato di deadlock e quando cio' accade **ripristinare una situazione normale** (***rilevare e ripristinare***)



… o come far finta di niente!

Ignorare il problema supponendo che il sistema non entri mai in uno stato di deadlock (es. molte versioni UNIX)

IDEA DI BASE: ***l'overhead*** introdotto dal trattamento del deadlock e' ***troppo alto*** e il ***deadlock troppo raro*** per valere la pena di considerarlo

CONTROINDICAZIONE: un ***piccolo insieme di processi*** in deadlock puo' facilmente ***contagiare*** altri e portare tutto il sistema al ***blocco totale***, che in certi casi puo' essere ***piu' dannoso dell'overhead*** di gestione

Operazioni di trattamento del deadlock

Esaminiamo ora nel dettaglio le operazioni di trattamento del deadlock :

1. ***Prevenire***

   oppure

2. ***Evitare***

   oppure

3a. ***Rilevare***

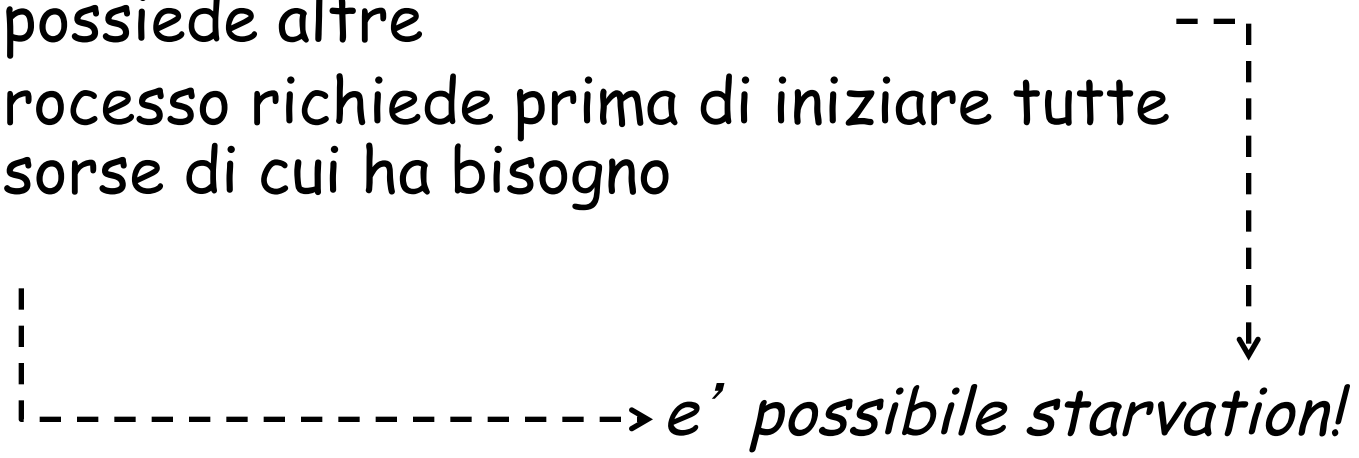
3b. ***Ripristinare***

1. Prevenire

Tecniche per ***prevenire*** che (anche solo) ***una delle 4 condizioni necessarie accada***

A. ***Mutua esclusione*** – si puo' prevenire soltanto su risorse condivisibili quali file in sola lettura

B. ***Possesso e attesa*** – *Due possibilita':*
   - ✓ un processo puo' richiedere risorse solo se non possiede altre
   - ✓ un processo richiede prima di iniziare tutte le risorse di cui ha bisogno



C. Assenza di preemption –

1. se un processo che possiede qualche risorsa richiede un' altra risorsa che non puo' immediatamente essergli allocata, allora tutte le risorse correntemente possedute gli vengono tolte (in maniera preemptive)
2. tali risorse vengono aggiunte alla lista di risorse per le quali il processo sta in attesa
3. il processo ripartira' solo quando potra' riacquisire le vecchie e le nuove risorse

- OPPURE -

La preemption viene applicata solo quando una delle risorse possedute dal processo viene richiesta da un altro processo

Di facile applicazione solo a risorse "leggere" che possono essere allocate/deallocate facilmente

D. Attesa circolare – si ordinano le risorse su numeri naturali (eventualmente per tipologie)

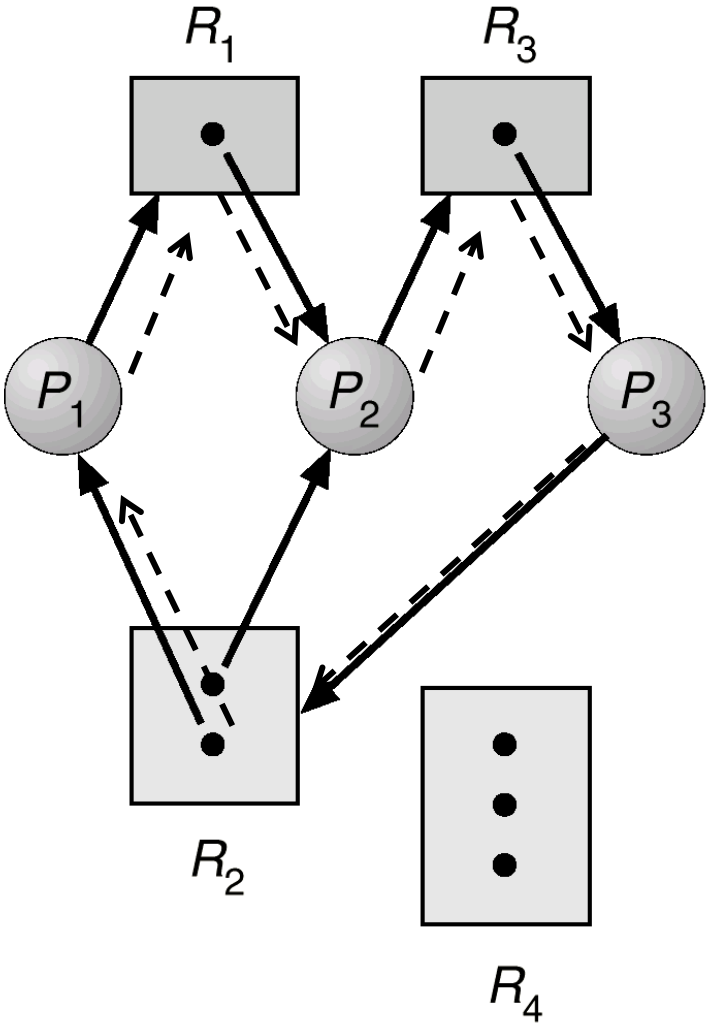
Dischi ad accesso veloce : 1,
Dischi ad accesso lento : 2 , etc.

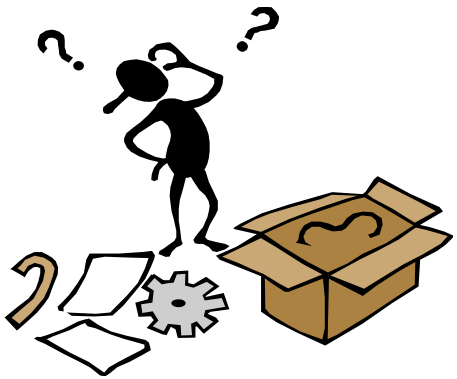
- si formulano le richieste solo in ordine crescente

OPPURE

- si puo' chiedere una risorsa di numero inferiore solo se si rilascia tutto cio' che si possiede di numero superiore

Come sarebbe cambiato questo grafo se una delle 4 "prevenzioni" fosse stata applicata ???





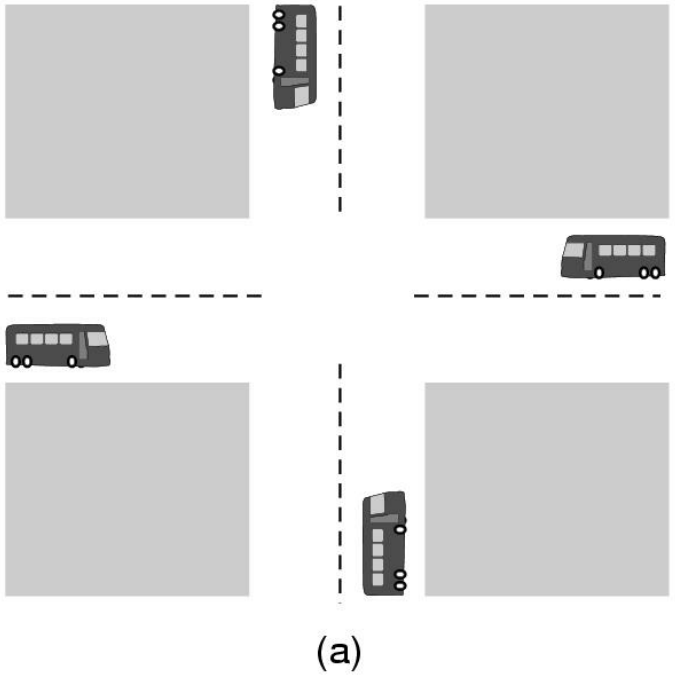
2. Evitare

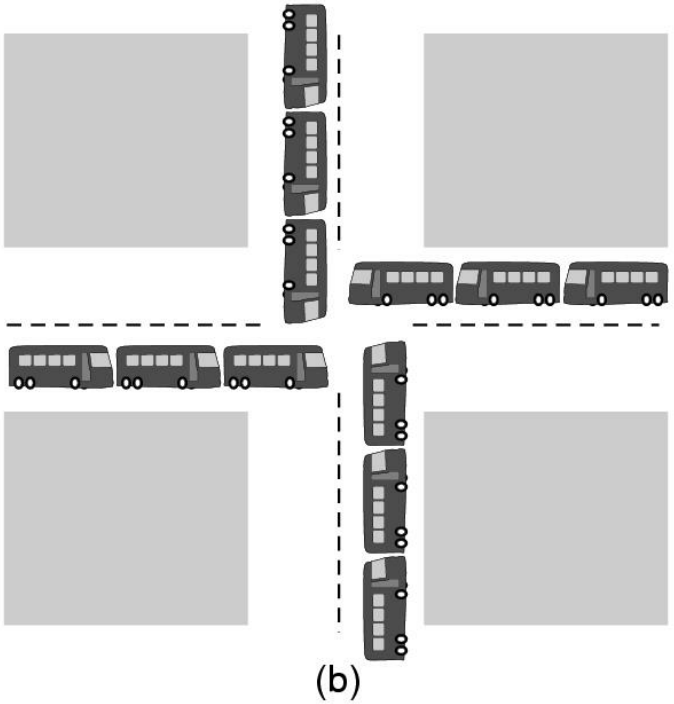
Si richiede che il sistema abbia ***piu' informazioni*** disponibili ***a priori*** sulle richieste di risorse da parte dei processi

Sono necessari ***algoritmi piu' complessi*** per ***decidere se allocare*** o meno una risorsa

Si tratta di ***decisioni dinamiche*** piuttosto che regole a priori come nella prevenzione

Differenza tra situazione "pericolosa" (*unsafe*) e deadlock

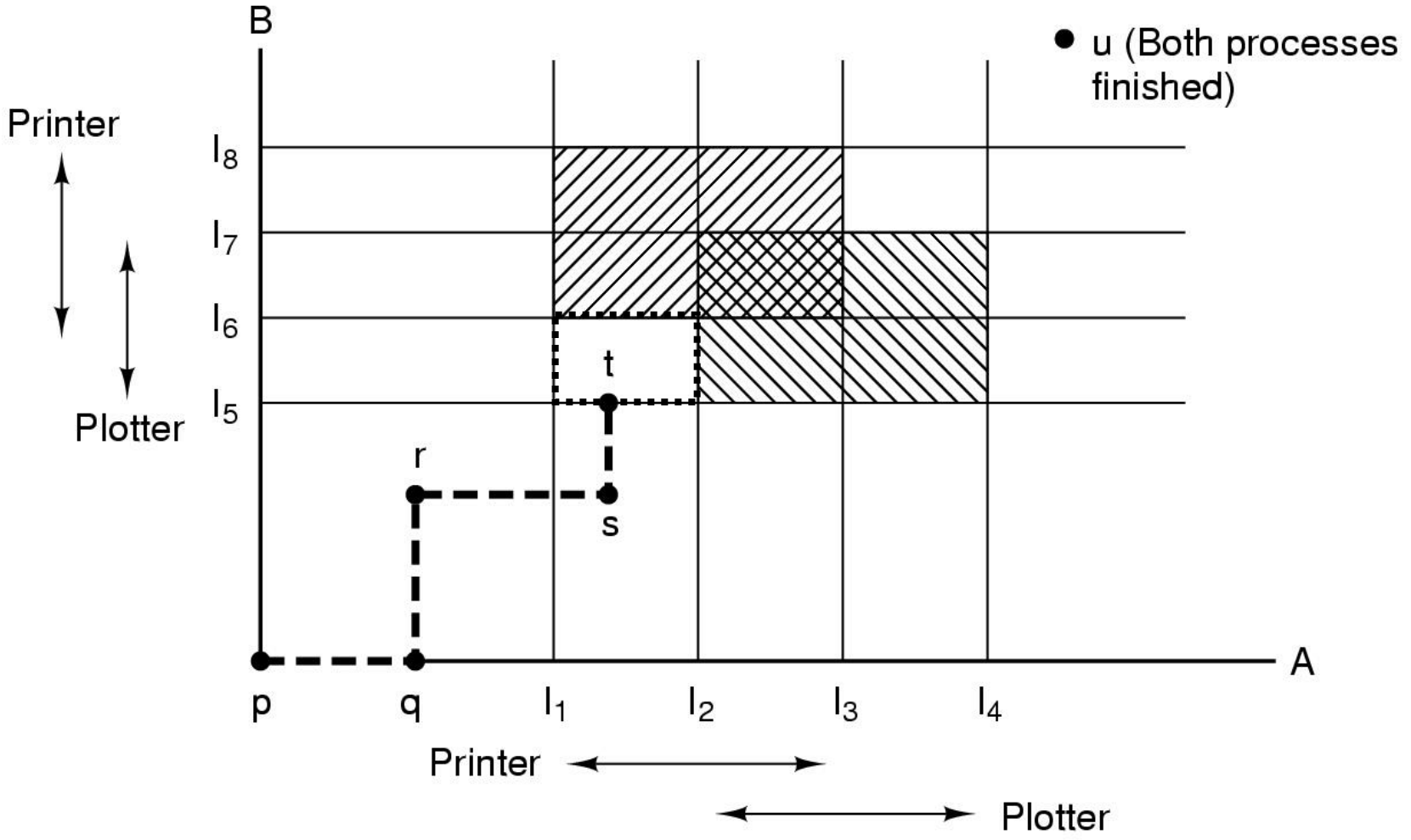




situazione pericolosa

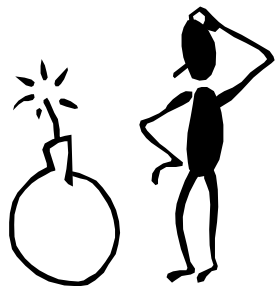
deadlock

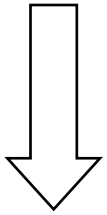
… e pericoli tra due processi



Una soluzione semplice

- Ogni processo dichiara all' inizio il ***massimo numero di richieste*** di ogni tipo di risorsa di cui puo' aver bisogno
- ***Ad ogni richiesta reale di risorse*** si esamina lo ***stato del sistema*** per assicurare che non si andrà mai in una situazione "pericolosa"





Lo ***stato*** e' definito da: numero di risorse ***disponibili***, numero di risorse ***allocate***, ***massima richiesta*** da parte di ogni processo

Definizione di stato *safe*: OS controlla la situazione…

- Il sistema e' in uno ***stato safe*** se ***esiste una sequenza safe di processi***

- La ***sequenza <P_1, P_2, …, P_n> si dice safe*** se per ogni P_i, le risorse che P_i ***<u>puo' ancora richiedere</u>*** possono essere allocate tra le risorse disponibili + le risorse possedute da tutti i P_j, con $j<i$

… e cioe':

- Se le risorse che *Pi* puo' ancora richiedere non sono immediatamente disponibili, allora *Pi* puo' attendere finche' tutti i *Pj* hanno finito

- Quando *Pj* ha finito, *Pi* puo' ottenere le risorse necessarie, eseguire, rilasciare le risorse allocate e terminare

- Quando *Pi* termina, *Pi+1* puo' ottenere le sue risorse, e cosi' via

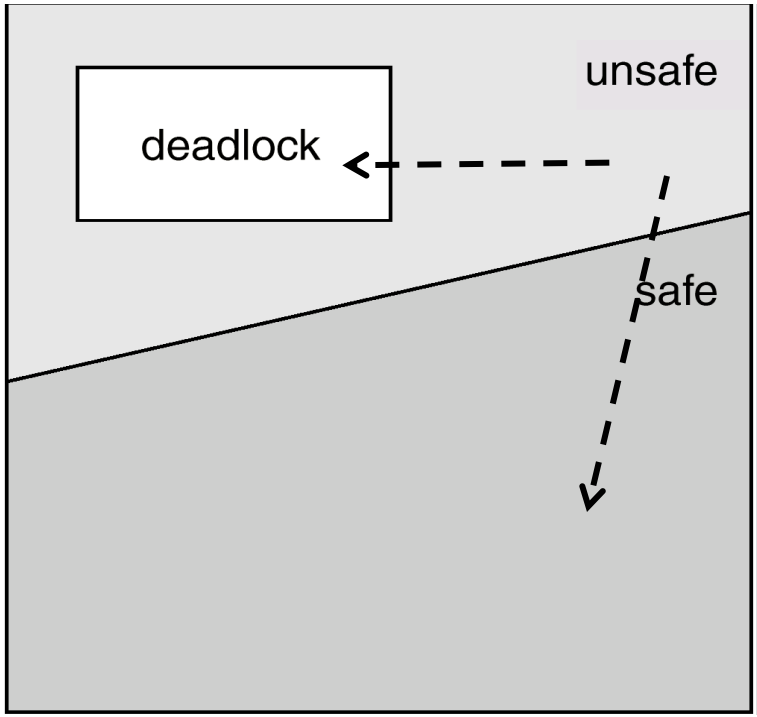
Considerazioni

Stato safe $\Rightarrow$ no deadlock
Stato unsafe $\Rightarrow$ possibilita' di deadlock

Evitare significa assicurare che il sistema ***non entri mai in uno stato unsafe***

E cioe' quando un processo richiede una risorsa, ***il sistema deve decidere*** se ***l'immediata allocazione*** di quella risorsa lascia il sistema ***in uno stato safe***

… e graficamente



Stato unsafe : e' il comportamento dei processi che determina o meno il fatto che accada un deadlock

Stato safe : OS puo' tenere sotto controllo la situazione come dettto prima

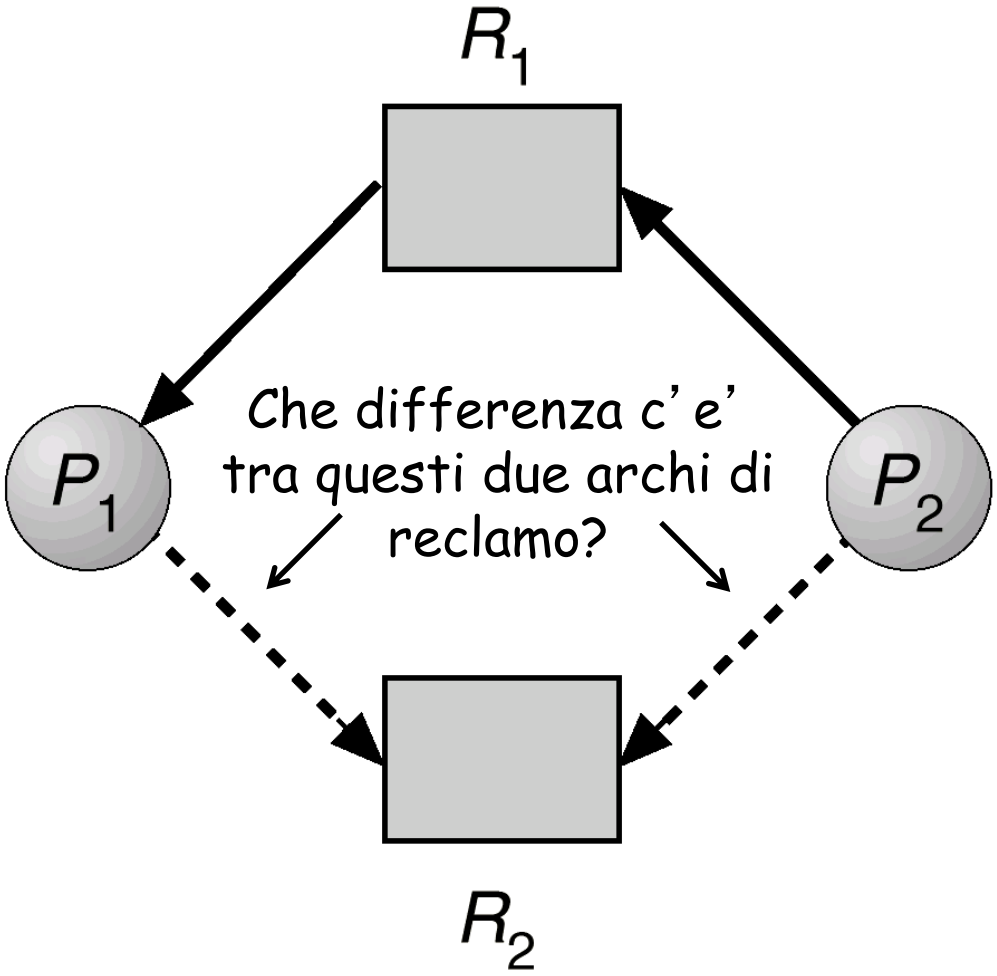
⇩

Quindi come stabilire se esiste una sequenza safe a valle dell' allocazione di una risorsa ?!

Risorse a istanza unica: modifica del grafo di allocazione risorse…

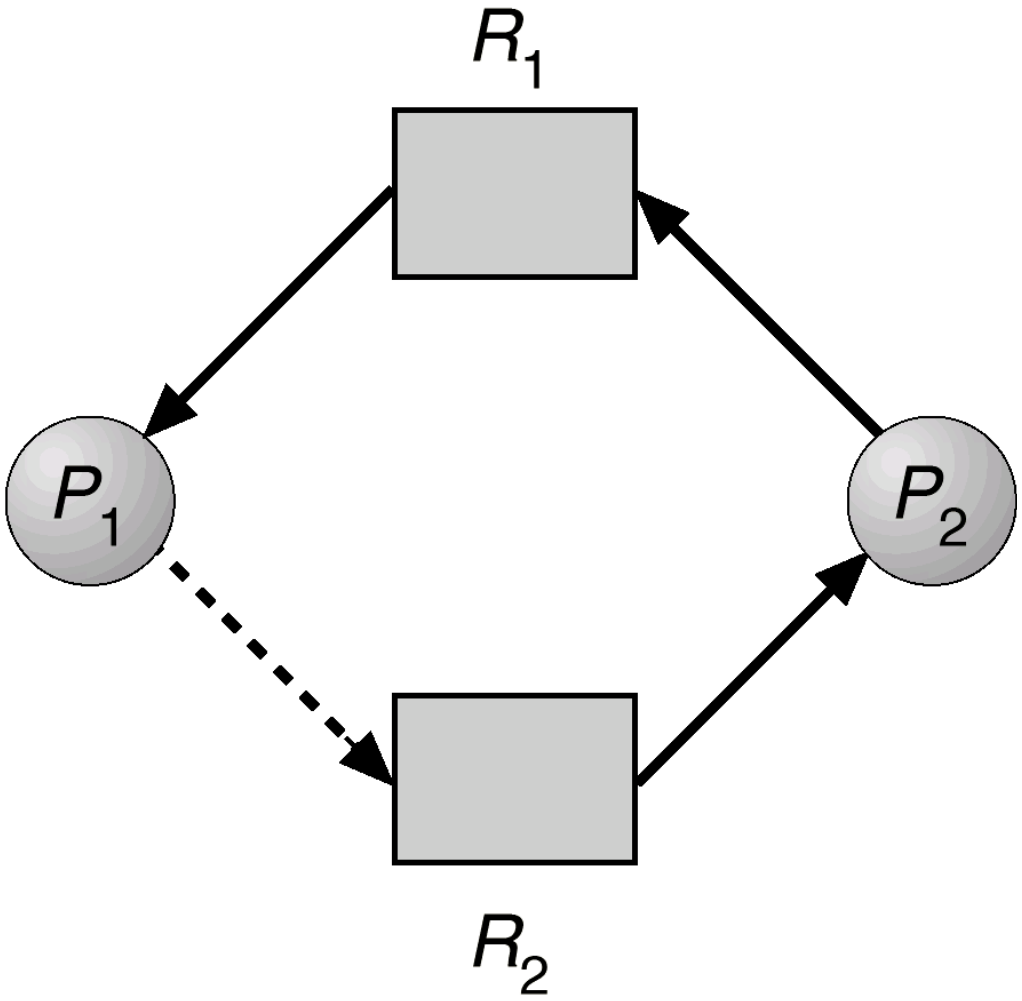
- Nuovo tipo di arco: ***Arco di reclamo*** $P_i \rightarrow R_j$ indica che il processo P_i ***puo' richiedere*** la risorsa R_j
- Un ***arco di reclamo*** diventa ***arco di richiesta*** quando il processo ***richiede*** effettivamente la risorsa
- Quando invece la ***risorsa*** viene ***rilasciata*** l' arco ***di assegnamento*** torna ad essere ***di reclamo***
- Le risorse devono essere ***reclamate a priori***

… ed un esempio



Quando il ***reclamo*** diventa ***richiesta***, prima di ***soddisfarla*** si verifica se genera un ciclo…

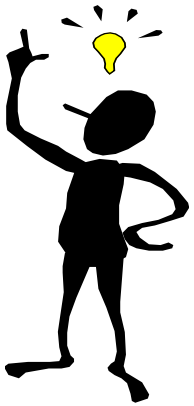
… come in questo caso: stato unsafe



Risorse a multiple istanze

Stessa idea di base del caso di singola istanza

- Ogni processo deve ***dichiarare a priori il massimo uso delle risorse*** (che naturalmente non deve essere superiore al numero totale di risorse del sistema!)
- Quando ***un processo richiede una risorsa*** potrebbe dover attendere anche se la risorsa fosse disponibile, in quanto ***OS deve stabilire se la sua immediata allocazione porta il sistema in uno stato unsafe***



Soluzione: Algoritmo del banchiere

a) Strutture dati principali

b) Verifica della safety

c) Richiesta di risorse

a) Strutture dati principali

n = numero di processi, *m* = numero di tipi di risorse

- *Available[m]* -
  - *Available[j] = k*, ci sono *k* istanze di risorsa di tipo R_j disponibili
- *Max[n,m]* -
  - *Max [i,j] = k*, il processo P_i puo' richiedere al piu' *k* istanze di risorsa di tipo R_j
- *Allocation[n,m]*
  - *Allocation[i,j] = k*, il processo P_i possiede *k* istanze di R_j.
- *Need[n,m]*
  - *Need[i,j] = k*, il processo P_i ***puo' avere bisogno*** ancora di *k* istanze di R_j per completare il suo compito

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j]$$

b) Verifica della safety (utilizzata all'interno del punto c)!!!)

1. *Work[m]* quantita' di risorse non allocate
   Finish[n] terminazione dei processi
   Work := *Available*
   Finish [i] = false forall *i*

   ← - - - - - - *inizializzazioni*

2. Trova un processo *i* tale che:
   (a) *Finish [i] = false*
   (b) $Need_i \leq Work$
   Se tale processo non esiste *go to step 4*

   ← - - - - - - - *un passo della sequenza safe di processi*

3. *Work* := *Work* + $Allocation_i$
   Finish[i] := true
   go to step 2

4. Se *Finish [i] = true* forall *i*, allora il sistema e' in uno stato safe

c) Richiesta di risorse da parte del processo P_i

$Request_i$ = vettore di richieste correnti di P_i

***Request*$_i$ [j] = k** : P_i vuole k istanze della risorsa di tipo R_j

1. Se $Request_i \leq Need_i$ go to step 2.
   Altrimenti segnala l' errore che il processo ha richiesto piu' risorse del suo massimo
2. Se $Request_i \leq Available$ go to step 3.
   Altrimenti P_i deve attendere per risorse non disponibili
3. In pratica simuliamo l' allocazione delle risorse richieste da P_i modificando lo stato come segue:

   $Available := Available - Request_i$

   $Allocation_i := Allocation_i + Request_i$

   $Need_i := Need_i - Request_i$

- VERIFICA DELLA SAFETY -

- Se lo stato raggiunto e' safe $\Rightarrow$ le risorse vengono allocate a P_i
- Se unsafe $\Rightarrow$ P_i deve attendere, e il vecchio stato di allocazione viene ripristinato

Esempio

- 5 processi

  3 tipi di risorse: *A* (10 istanze), *B* (5 istanze), *C* (7 istanze)
- Fotografia del sistema al tempo T_0:

	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	

- Il contenuto della matrice *Need* e' definito come *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 e' in uno stato safe perche' esiste la sequenza safe < P_1, P_3, P_4, P_2, P_0>

P_1 richiede (1,0,2)

- Request ≤ Available : (1,0,2) ≤ (3,3,2) ⇒ *true.*

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

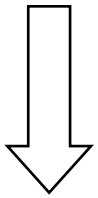
- L'algoritmo di safety a questo punto mostra che $<P_1, P_3, P_4, P_0, P_2>$ e' una sequenza safe

E se P_4 avesse richiesto (3,3,0) ?

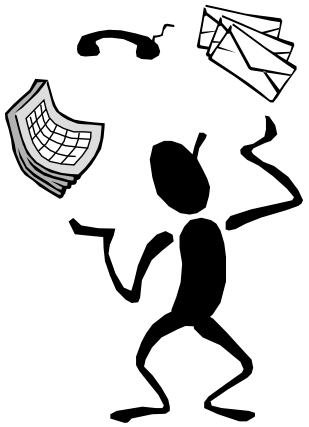
E se P_0 avesse richiesto (0,2,0) ?

3. Rilevare – 4. Ripristinare

IDEA : Si permette al sistema di entrare in uno stato di deadlock



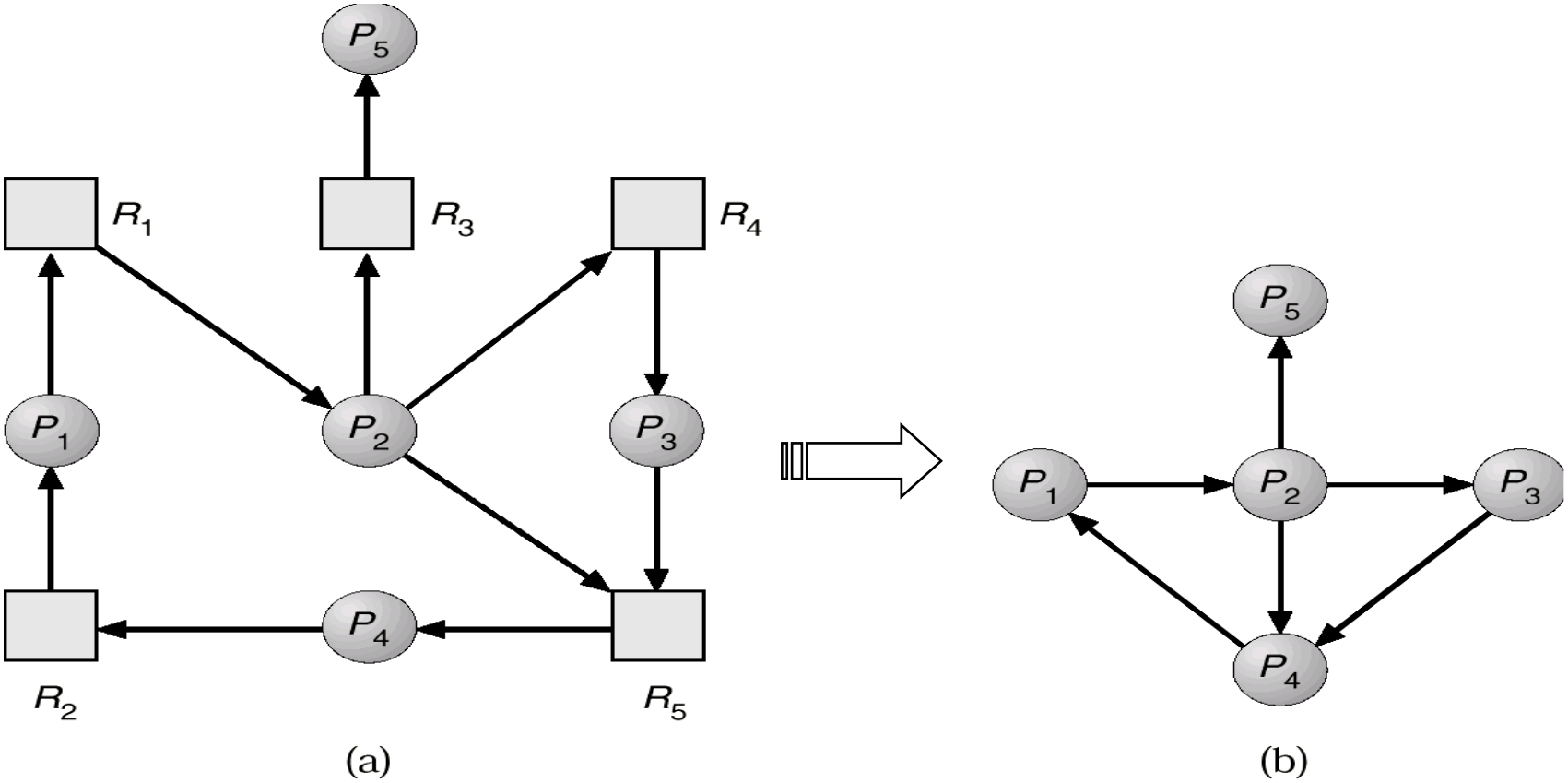
- Algoritmo di detection
- Schema di recovery



Risorse a istanza unica: semplificazione del grafo di allocazione risorse…

- Diventa un ***grafo di attesa***
  - i nodi sono solo processi
  - $P_i \rightarrow P_j$ se P_i ***sta attendendo*** qualche risorsa posseduta da P_j
- Una volta cancellate le risorse, si collassano gli archi
- Periodicamente si invoca un algoritmo che ***ricerca un ciclo nel grafo***

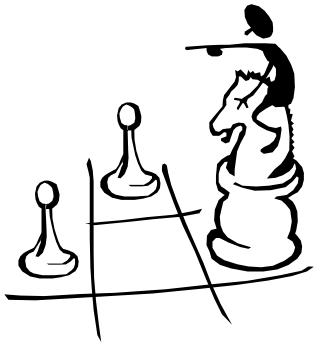
Dal grafo di allocazione al grafo di attesa



Grafo di allocazione risorse

Corrispondente grafo di attesa

Risorse a multiple istanze : un "banchiere on-line"



- *Available[m]*
  - *Available[j] = k*, ci sono *k* istanze di risorsa di tipo *Rj* disponibili
- *Allocation[n,m]*
  - *Allocation[i,j] = k*, il processo *Pi* possiede *k* istanze di *Rj*
- *Request[n,m]*
  - *Request [i,j] = k*, il processo *Pi* sta richiedendo *k* istanze di risorsa di tipo *Rj*

Algoritmo di detection…

IDEA DI BASE

Le risorse vengono ***allocate subito a processi sicuramente non coinvolti*** in un deadlock ($Request_i \leq Work$), supponendo che:

- non ne chiedano altre
- quelle che vengono allocate servano solo per completare il lavoro e quindi restituirle tutte

Se cosi' non e', si potra' verificare un deadlock che verra' rilevato nelle successive invocazioni dell'algoritmo

… step by step

1. *Work* e *Finish* come nel "banchiere"

   Work := *Available*

   For $i = 1,2, \ldots, n$, if $Allocation_i \neq 0$, then
   Finish[i] := false ; else *Finish[i] := true*

2. Trova un processo *i* tale che

   Finish[i] = false

   $Request_i \leq Work$

   Se tale processo non esiste *go to step 4*

3. *Work* := *Work* + $Allocation_i$
   Finish[i] := true
   go to step 2.

Si suppone che un processo termini dopo aver soddisfatto la richiesta corrente

4. Se *Finish[i] = false*, per qualche *i*, $1 \leq i \leq n$, allora il sistema e' in stato di deadlock

   Inoltre, se *Finish[i] = false*, allora il processo P_i e' in deadlock

Esempio

- 5 processi

  3 tipi di risorse: *A* (7 istanze), *B* (2 istanze), *C* (6 istanze)

- Fotografia del sistema al tempo T_i:

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

- La sequenza <P_0, P_2, P_3, P_1, P_4> risultera' in *Finish[i] = true* forall *i*

P_2 richiede una ulteriore istanza di tipo C

Qual e' lo stato del sistema ora?

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

Anche se si reclamassero le risorse possedute dal processo P_0 sarebbero comunque insufficienti per soddisfare le richieste degli altri processi

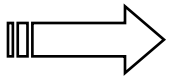
Esiste un deadlock tra i processi
P_1, P_2, P_3, P_4

Uso dell'algoritmo di detection

- ***Quando*** e ***quanto spesso invocare*** l'algoritmo dipende da:
  - ***Quanto spesso un deadlock*** puo' accadere?
  - ***Quanto pesante e' l'overhead*** della procedura di detection/recovery?
  - ***Quanti processi*** dovranno essere ***terminati per rompere il ciclo che determina il deadlock*** ?

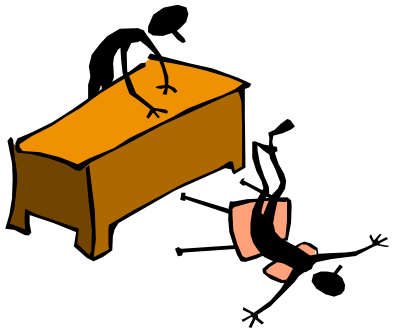
    (e' una questione riguardante il recovery !)

Se l'algoritmo e' invocato in maniera arbitraria ci potrebbero essere troppi cicli nel grafo da non poter identificare un ***processo colpevole***



Schema di recovery 1: terminazione di processi

- Termina ***tutti i processi*** coinvolti nel deadlock
- Termina ***un processo alla volta*** finche' il ciclo di deadlock non si interrompe



Parametri da considerare per stabilire l'ordine di terminazione dei processi ***(selezione della vittima)***

- Priorita' del processo
- Quanto del processo e' stato eseguito, e quanto manca alla terminazione
- Risorse gia' utilizzate e risorse di cui ha bisogno per il completamento
- Processo interattivo o batch
- (Quanti processi devono essere terminati)

Schema di recovery 2: preemption delle risorse

- Di nuovo ***selezione di una vittima***: questa volta un processo e' vittima come conseguenza della scelta di una risorsa che gli viene tolta
- Il processo privato della risorsa deve tornare indietro nella computazione (***rollback*** fino a dove?!)
- ***Evitare starvation*** – un processo dovrebbe essere vittima solo un numero finito di volte (*anche nello schema di soluzione 1!*)
- Si puo' includere ***il numero di rollbacks nel fattore di costo*** calcolato nella selezione della vittima

Idea di un approccio combinato

- Combinare i 3 approcci di base
  - prevenire
  - evitare
  - rilevare e ripristinare

  permettendo ***l'uso dell'approccio ottimale per ogni tipo di risorsa nel sistema***
- Ordinare le classi di risorse e applicare la richiesta ordinata di risorse vista in precedenza
- All'interno poi di ogni classe di risorsa applicare l'approccio piu' appropriato