

Deadlock

(sempre legato alla gestione
di risorse condivise)

CONDIZIONI NECESSARIE AL VERIFICARSI DEL DEADLOCK

- ① VARIABILI IN MUTUA ESCLUSIONE risorse condivise
- ② POSSESSO E ATTESA processi che riescono a entrare in possesso di una risorsa ma ne richiedono altre prima di rilasciarle
- ③ ATTESA CIRCOLARE
- ④ ASSENZA DI PRELAZIONE

STRUMENTI DI RILEVAZIONE DEL DEADLOCK

- Mecanismo di rilevare informazioni sull'utilizzo delle risorse da parte dei processi mediante un GRAFO DI ASSEGNAZIONE DELLE RISORSE

NODI di processi
di tipi di risorse OP

□ R quadretto vuoto \rightarrow 1 sola istanza, per ogni istanza aggiuntiva si fa un pallino nel quadratino

ARCHI di richiesta PO \rightarrow □ R
di assegnazione PO \leftarrow □ R

esempio di grafo di assegnazione di risorse:



CONDIZIONE NECESSARIA PER LA PRESENZA DI DEADLOCK:

LOOP

STRATEGIE DI HAVENDER (prevenzione del deadlock)

- Imponne che i processi richiedano tutte le risorse necessarie prima di proseguire con l'esecuzione
 - + Si inibisce la condizione di possesso-Attesa
 - Non consente un uso efficiente delle risorse
 - Rischio di starvation
- Richiesta di risorse in successione. Se una delle risorse richieste risultate occupate, il processo è forzato dall'OS (tramite prelazione) a rilasciare tutte le risorse detenute
 - + Uso più efficiente delle risorse
 - Spesso il processo è costretto alla terminazione
- Assingnemt ad ogni risorsa un numero d'ordine. Viene imposto un vincolo sulle richieste delle risorse da parte dei processi
 - + Si inibisce la condizione di attesa circolare
 - Manca un criterio generale per definire l'ordine
 - Non è scontato che i programmi continuino a funzionare dopo l'affronte di una nuova risorsa

DEADLOCK AVOIDANCE

STATO DEL SISTEMA

↳ allocazione risorse:

- PROCESSI devono dichiarare il numero di risorse necessarie
- TIPI DI RISORSE
 - # di istanze assegnate
 - # di risorse

esempio di tabella di allocazione risorse

	R ₁	
	A	R
P ₁	2	3
P ₂
P ₃

A: assegnate
R: richieste (pendenti)

free n

STATO SICURO

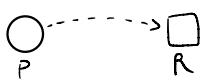
CONDIZIONE

ESISTENZIALE

se esiste un percorso che partendo da una condizione in cui tutti i processi riescano ad ottenere le risorse desiderate. Contrapposti agli STATI INSICURI (DOOMED) che portano necessariamente a deadlock

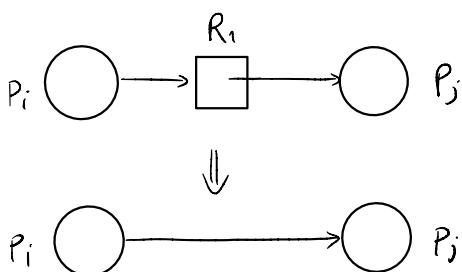
GRAFO DI ALLOCAZIONE RISORSE CON ARCHI DI RECLAMO

arco di reclamo (claim edge)

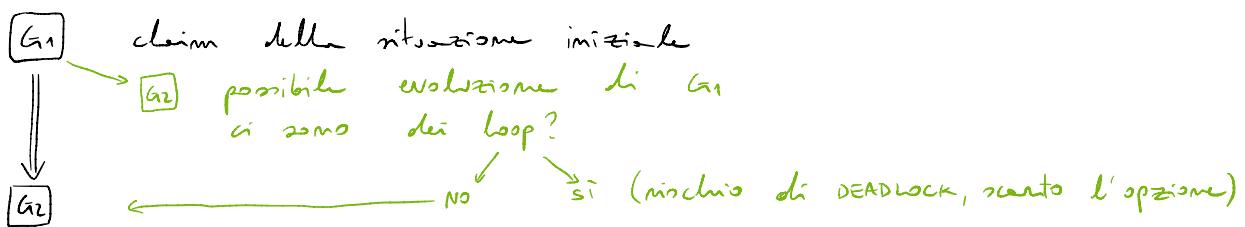
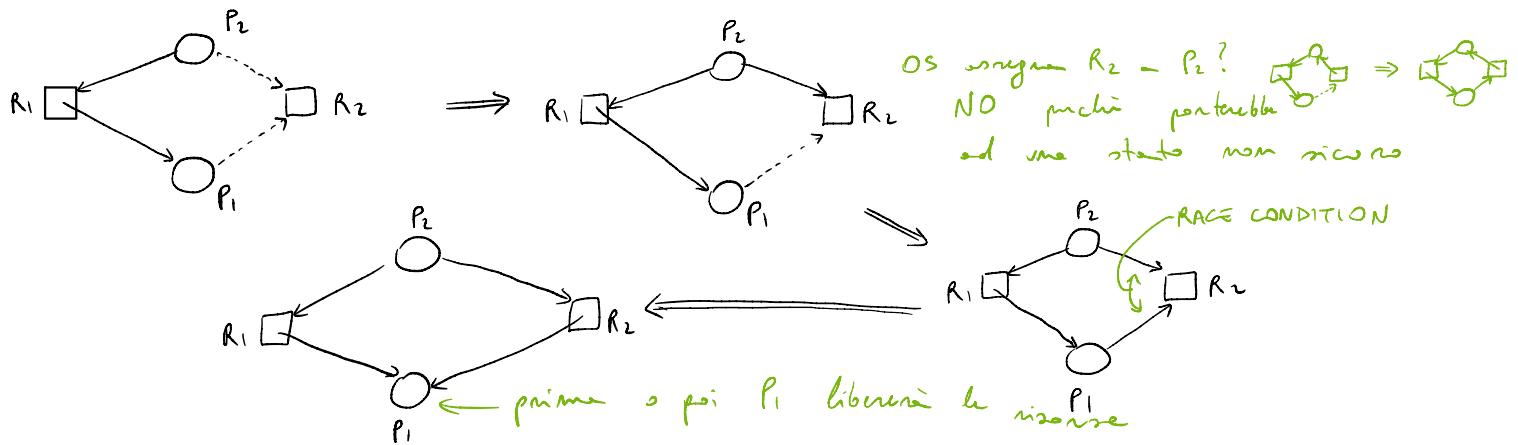


Suvono per dichiarare che il processo richiede la risorsa

GRAFI DI ATTESA

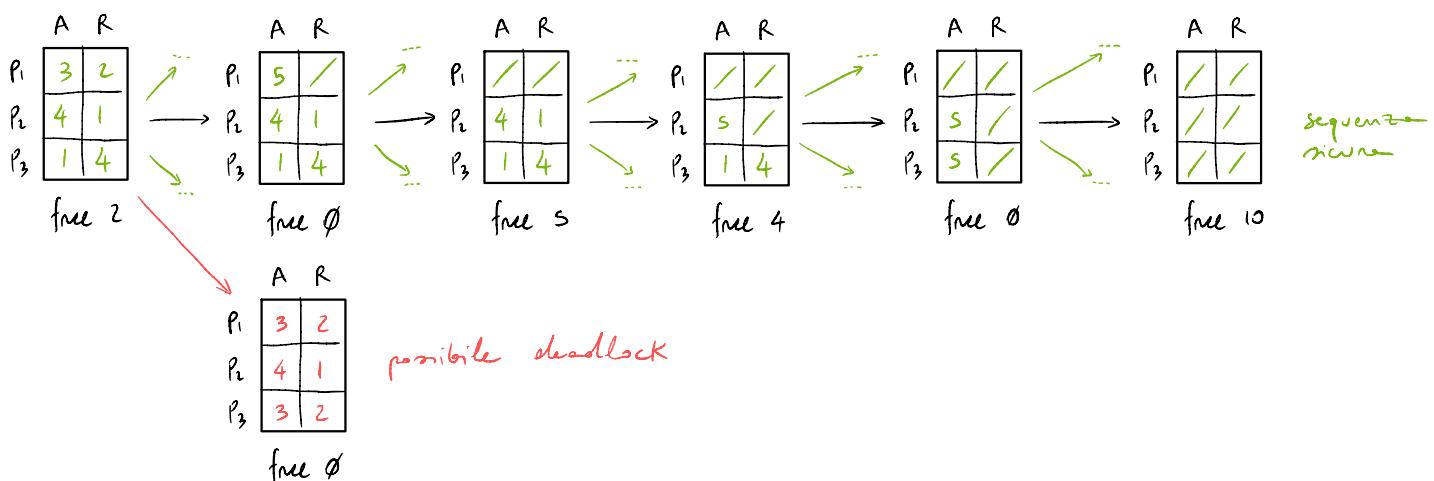


esempio:



- Funziona solo se tutte le risorse hanno una sola istanza

esempio di simulazione con una risorsa a più istanze:



ALGORITMO DEL BANCHIERE

VARIABILI:

M = num. tipi di risorse

N = num processi

DISPONIBILI [M]

MASSIMO [N][M]

ASSEGNAZIONE [N][M]

NECESSARIE [N][M]

CODICE:

- Diviso in due sotto-algoritmi
 - ① → algoritmo che gestisce le richieste
 - ② → algoritmo che verifica che uno stato è sicuro

⑤

1 Siamo LAVORO e FINE due array di lunghezza M ed N rispettivamente
2 LAVORO = DISPONIBILI
3 FINE[i] = FALSO $\forall i \in [i, N]$
4 CERCA $i \in [i, N]$ | FINE[i] == FALSO \wedge NECESSARIE[i] \leq LAVORO
5 se trovato:
6 LAVORO = LAVORO + ASSEGNAZIONE[i]
7 FINE[i] = VERO
8 GOTO (4)
9 se $\forall i \in [i, N]$ FINE(i) = VERO, lo STATO È SICURO

⑥

1 INT RICHIESTE [v_1, v_2, \dots, v_M]
2 Siamo P_j i processi che fanno richiesta v con M risorse diverse
3 NECESSARIE[N][M] con $N = \#$ processi P_j
4 if (RICHIESTE \leq NECESSARIE[j]) {
5 if (RICHIESTE \leq DISPONIBILI) {
6 DISPONIBILI = DISPONIBILI - RICHIESTE
7 STATO ASSEGNAZIONE[j] = ASSEGNAZIONE[j] + RICHIESTE
8 NECESSARIE[j] = NECESSARIE[j] - RICHIESTE
9 if (VERIFICA che lo stato costituito è sicuro (b))
10 Assegnazione confermata
11 else
12 Si ripristinano i valori precedenti di P_j
13 }
14 } else
15 } else {ERRORE}

ALGORITMO GENERALE DI RILEVAZIONE DL

VARIABILI :

M = # tipi di risorse

N = # processi

DISPONIBILI [M]

ASSEGNAME [N][M]

RICHIESTE [N][M]

CODICE :

```
1 int LAVORO[M]
2 boolean FINE[N]
3 LAVORO = DISPONIBILI
4 for (i ∈ [1, N])
5     if (ASSEGNAME[i] = {0, ..., 0})
6         FINE[i] = TRUE;
7     else
8         FINE[i] = FALSE;
9 while (exists i | FINE[i] == FALSE ∧ RICHIESTE[i] ≤ LAVORO)
10    LAVORO = LAVORO + ASSEGNAME[i]
11    FINE[i] = TRUE
12 for (i ∈ [1, N])
13    if (FINE[i] == FALSE)
14        return DEADLOCK
15 return NO DEADLOCK
```

Quando controllare se c'è DEADLOCK?

- Si tenta in tanti - intervalli predefiniti
- Quando un processo richiede una risorsa e la trova occupata
- Quando l'utilizzo delle CPU scende sotto una certa soglia

Cosa fare quando si rileva DEADLOCK?

Possibili soluzioni:

- ① Terminare qualche processo *operazione molto costosa*
 - computazione/risultati parziali
 - transazioni/rollback
 - interattivi
 - priorità del processo
- ② Pulizia e Redistribuzione delle risorse
- ③ Nulla