# DEADLOCK (or deadly embrace)

## Sommario

- Criticità nei sistemi concorrenti
- Risorse e problemi di utilizzo: esempi
- Il modello di descrizione di un sistema
- Processi e allocazione di risorse
- Caratterizzazione del deadlock
- Grafo di allocazione delle risorse: modello e rilevazione di un deadlock
- Esempi
- Metodi di gestione dei deadlock: ignorare?
- Prevenzione
- Astensione: stato sicuro, sequenza sicura di completamento
- Algoritmo del Banchiere e teorema di Habermann: esempi
- Rilevazione e recupero

# Criticità nei sistemi concorrenti

- **Starvation**: una situazione in cui l'esecuzione di uno o più processi (thread) è posticipata indefinitamente.
- **Deadlock**: una situazione in cui tutti i processi (thread) di applicazioni cooperanti sono bloccati.

I processi che sono in attesa possono permanere indefinitamente in tale stato se le risorse che hanno richiesto sono in possesso di altri processi, che non le rilasciano (starvation) o sono a loro volta in attesa.

- Infatti molte ***risorse*** dei sistemi di calcolo e, molto più in generale del vivere corrente, possono essere ***usate solo in modo esclusivo***.
- I sistemi operativi e, più in generale i sistemi di gestione delle comuni risorse, devono ***assicurare l'uso consistente di tali risorse***: le risorse vengono allocate in modo esclusivo, per un tempo limitato. Gli altri richiedenti vengono messi in attesa.
- Ma un processo/utilizzatore di risorse può ***avere bisogno di molte risorse contemporaneamente***.

# Risorse e problemi di utilizzo

Una ***risorsa*** è una componente di un sistema a cui i processi/utilizzatori possono accedere in modo esclusivo, per un certo periodo di tempo.

***Risorse prerilasciabili***: possono essere tolte al processo/utilizzatore allocante, senza effetti dannosi.
Esempio: memoria centrale.

***Risorse non prerilasciabili***: non possono essere cedute dal processo allocante, pena il fallimento dell'esecuzione.
Esempio: stampante, file.

**I deadlock si hanno con le risorse non prerilasciabili.**

# Esempi

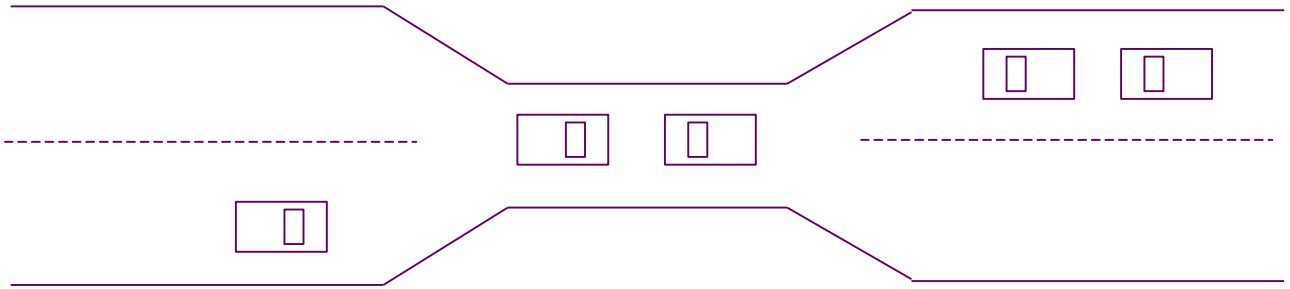
## Esempio dei nastri magnetici

Il sistema ha 2 nastri.

$P_1$ e $P_2$ utilizzano ciascuno un nastro e ciascuno di loro ne richiede un altro.

```
   P1              P2
wait (A);     wait(B)
wait (B);     wait(A)
```

## Esempio del ponte



- Il traffico avviene solo in una direzione alternativamente.
- Ciascuna parte del ponte può essere considerata come una risorsa.
- Per risolvere la situazione può essere necessario retrocedere una o più macchine.
- Per alternare il traffico si può determinare starvation.

# L'abbraccio della morte in città

# Il modello di descrizione di un sistema

- Tipi di risorse $R_1, R_2, \ldots, R_m$

  *CPU cycles, memory space, I/O devices*

- Ogni risorsa di tipo $R_i$ (i=1..m) può avere più istanze identiche $W_{i,j}$ (j=1..p) (esemplari)

- Ogni processo utilizza una risorsa come segue:
  - **Richiesta** (se essa non può essere soddisfatta immediatamente il processo deve attendere fino all'acquisizione);
  - **Uso** (il processo può adoperare la risorsa ed operare su di essa);
  - **Rilascio** (il processo libera la risorsa).

- Richiesta e rilascio di una risorsa sono chiamate di sistema se la risorsa è controllata dall'O.S. (p. es *open* e *close* su un file); in caso contrario esse possono essere implementate con una coppia acquire()/release() di un semaforo

# Risorse di sistema e processi

I processi indirizzano al SO le richieste per le risorse da esso gestite.

Una tabella del SO registra lo stato (libero/allocato) di ogni risorsa e traccia il processo che la impegna. Se un processo richiede una risorsa occupata viene accodato insieme agli processi che ne attendono il rilascio.

Si definisce **deadlock di un sottoinsieme** di processi $\{P_1, P_2, \ldots, P_n\}$ la situazione in cui ciascuno degli n processi $P_i$ e' in attesa del rilascio di una risorsa detenuta da uno degli altri processi del sottoinsieme; si forma cioè una ***attesa circolare*** **(catena circolare)** per cui:

$P_1$ *aspetta* $P_2$ ..... *aspetta* $P_n$ *aspetta* $P_1$

!! Anche se non tutti i processi del sistema sono bloccati, la situazione non è desiderabile in quanto può bloccare delle risorse (quali?) e non soddisfa alcuni utenti !!

**Eventi che generano stalli:**

Acquisizione/rilascio di risorse (fisiche e logiche [file, semafori, monitor]);

Inter Processes Communication (IPC)

# Allocazione delle risorse

- Situazioni di stallo si possono verificare su risorse sia locali sia distribuite, sia software che hardware.
- Allocazione di una risorsa
  - ➔ Si può disciplinare l'allocazione mediante dei semafori, uno per ogni risorsa
- Allocazione di più risorse
  - ➔ come allocarle?
- Non è detto che i due o più programmi che intendono allocare la stessa risorsa siano scritti dallo stesso utente: come coordinarsi?
- Con decine, centinaia di risorse (come quelle che deve gestire il kernel stesso), come determinare se una sequenza di allocazioni è sicura?

# Caratterizzazione del deadlock

Una condizione di deadlock si può verificare se si presentano ***simultaneamente*** le quattro condizioni seguenti (**Coffman theorem**, 1971):

**Mutua esclusione**: soltanto un processo alla volta può usare la risorsa (almeno una risorsa deve essere usata in modo non condivisibile).

**Possesso e attesa**: un processo che detiene almeno una risorsa deve attendere per acquisire ulteriori risorse utili alla sua computazione ed allocate ad altri processi.

**Assenza di preemption**: una risorsa può essere liberata soltanto volontariamente dal processo che la detiene, dopo che ha completato le operazioni su di essa.

**Attesa circolare**: esiste un gruppo di processi $\{ P_1, P_2, ....., P_n\}$ in attesa di risorse, tali che $P_1$ attende una risorsa detenuta da $P_2$, $P_2$ ne attende una detenuta da $P_3$, …, $P_{n-1}$ ne attende una detenuta da $P_n$, e $P_n$ attende una risorsa detenuta da $P_1$.

# Model representation

***Resource-Allocation Graph***

- I deadlock possono essere descritti con l'ausilio di una opportuna rappresentazione grafica costituita da:
  - un insieme di nodi *V*
  - un insieme di archi *E*.

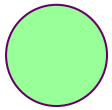
V è diviso in due tipi di nodi: $P = \{P_1, P_2, \ldots, P_n\}$, insieme di tutti i processi del sistema.
$R = \{R_1, R_2, \ldots, R_m\}$, insieme di tutti i tipi di risorse del sistema.

**arco di richiesta** - arco orientato $P_1 \rightarrow R_j$

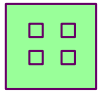
**arco di assegnazione** - arco orientato $R_j \rightarrow P_i$

# Grafo di allocazione delle risorse

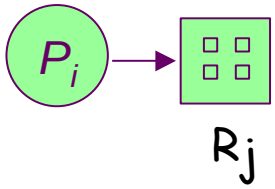
- Processo:



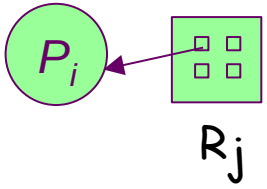
- Risorsa con 4 istanze (esemplari)



- $P_i$ richiede un'istanza di risorsa di tipo $R_j$:




- $P_i$ detiene un'istanza di risorsa di tipo $R_j$:



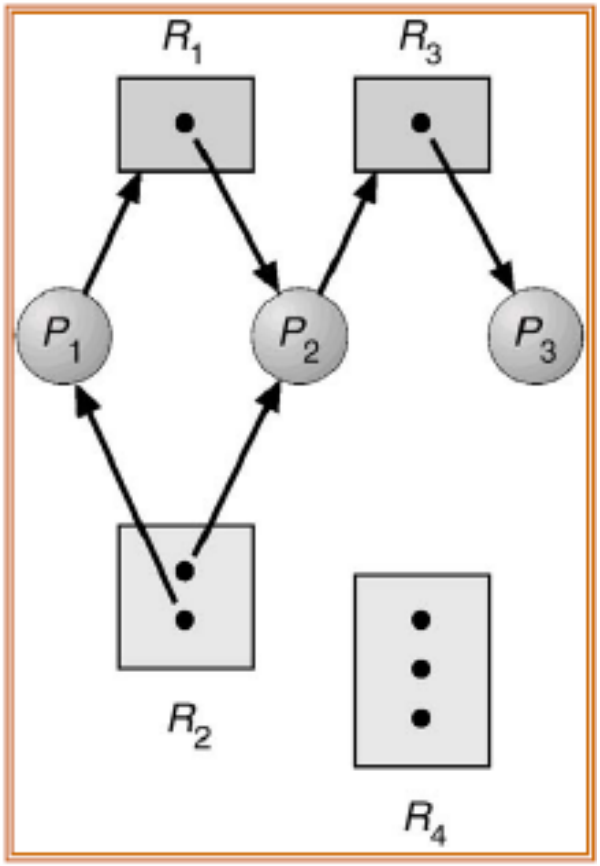
# Grafo di allocazione delle risorse: rilevazione di un deadlock

## Proprietà principale

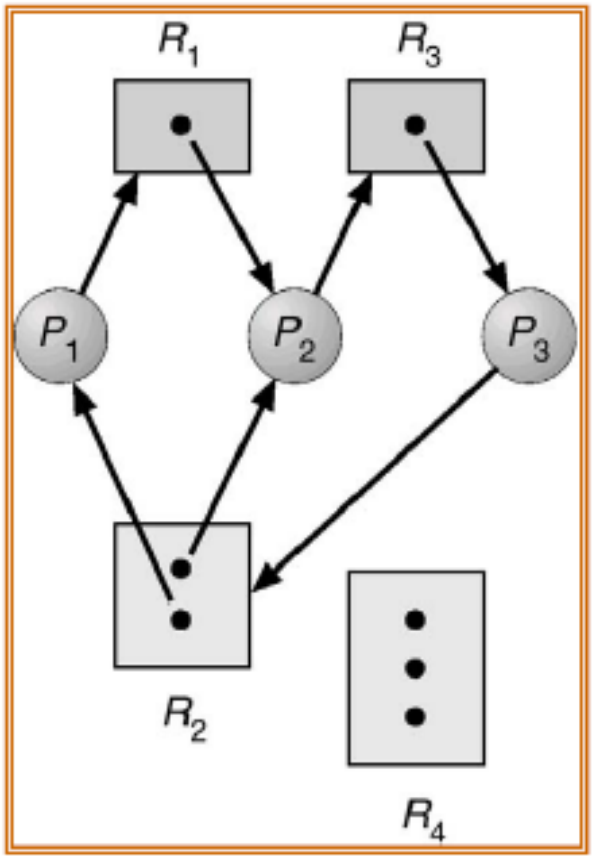
- Se il grafo non contiene cicli non c'è deadlock nel sistema
- Se il grafo contiene cicli ci può essere un deadlock
  - Se il grafo rappresenta un sistema con **una sola istanza per ogni risorsa allora c'è sicuramente deadlock**
  - Se il grafo rappresenta un sistema con **istanze multiple, allora il deadlock è possibile**

# Grafo di allocazione delle risorse: esempi

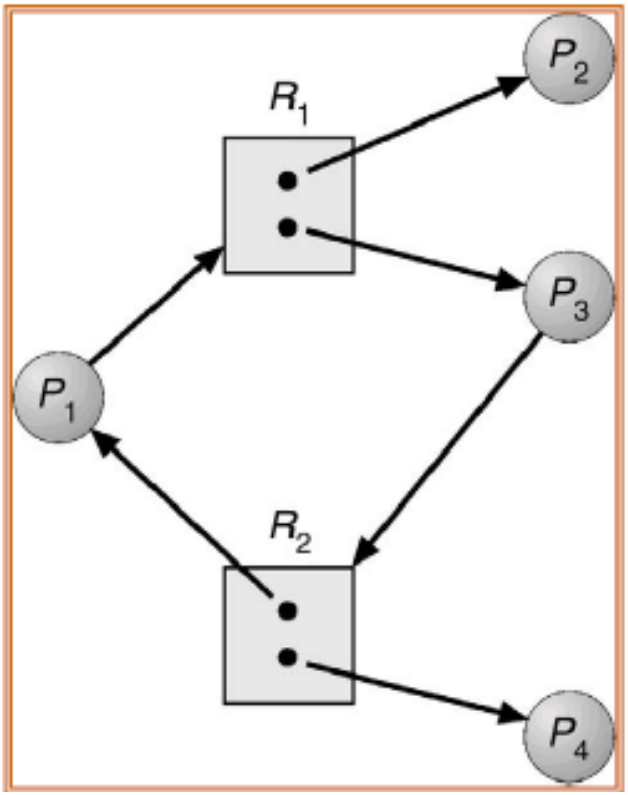
Grafo di allocazione delle risorse




Presenza di cicli con deadlock

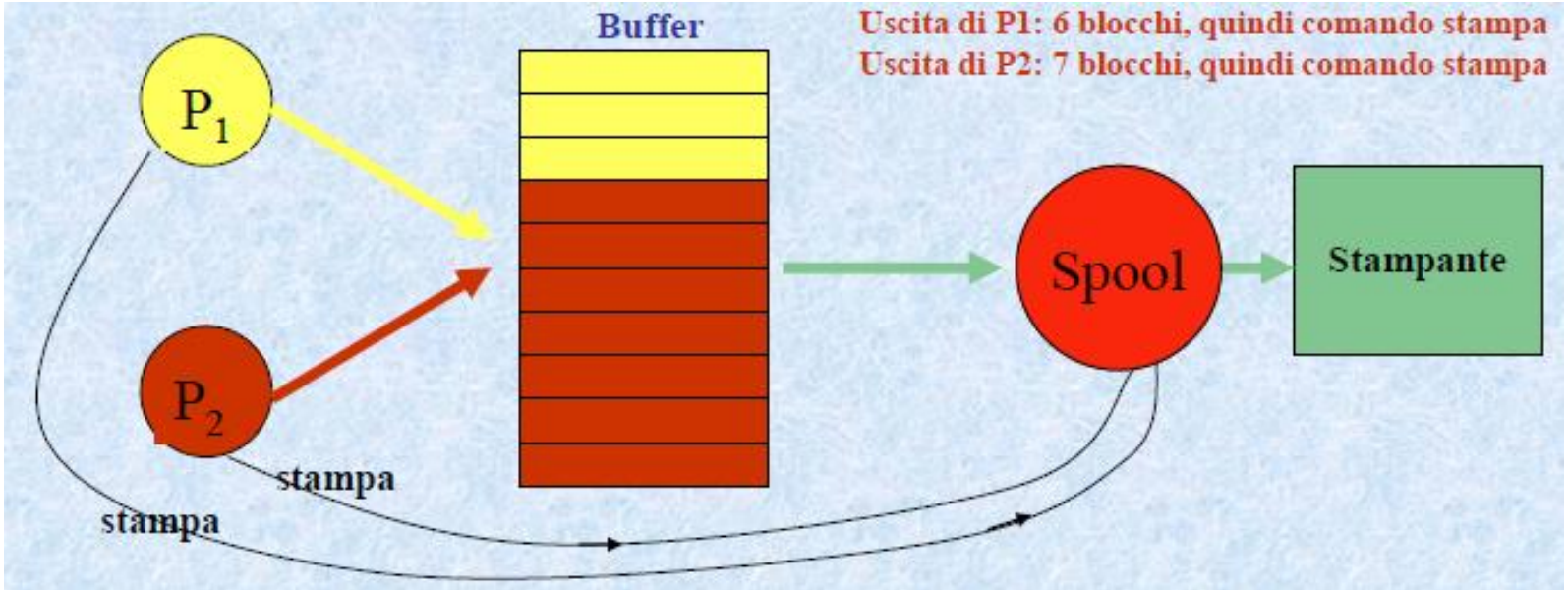



Presenza di cicli senza deadlock



# Un caso di deadlock: il daemon di SPOOL 1/2

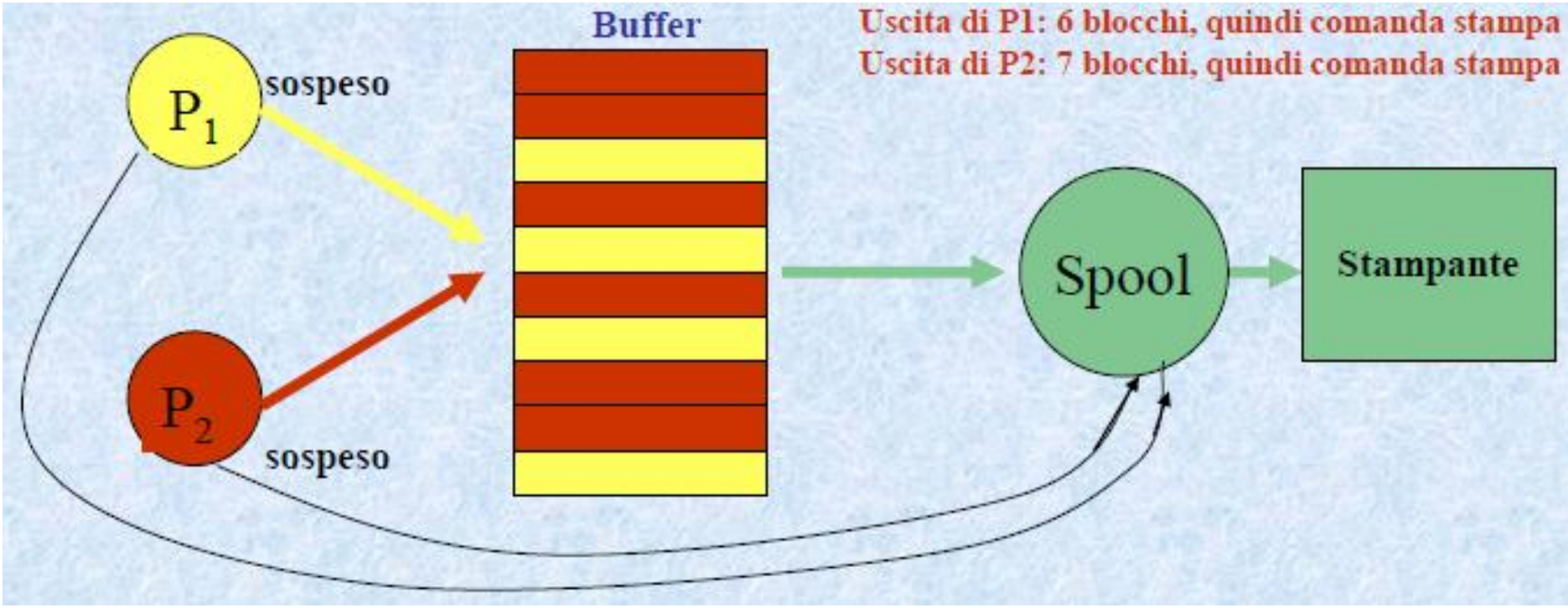
## 1) Sequenza che non provoca stallo




- **Al tempo t: P1 ha inviato 4 blocchi di stampa; P2 ha inviato 4 blocchi di stampa; Dopo tempo t: Buffer P1 invia 1 blocco;**
- **P1 invia 1 blocco e invia comando Stampa**
- **P2 invia 1 blocco e invia comando Stampa**
- **Spool inizia stampa dell'uscita di P1**
- **P2 invia 1 blocco;**
- **P2 invia 1 blocco;**
- **Spool completa stampe**

# Un caso di deadlock: il daemon di SPOOL 2/2

## 2) Sequenza che provoca stallo




- **Al tempo t: P1 ha inviato 4 blocchi di stampa; P2 ha inviato 4 blocchi di stampa;**
- **Dopo tempo t:**
  - **Buffer**
  - **P2 invia 1 blocco**
  - **P2 invia 1 blocco**
  - **P1 invia 1 blocco: P1 sospeso**

# Methods for Handling Deadlocks

È necessario avere dei **metodi** per *prevenire*, *riconoscere* o almeno *risolvere* i deadlock.

## I metodi di gestione dei deadlock

- **Ignorandoli**: quando possibile si cerca di ignorarli
- **Prevention**: si prendono tutte le precauzioni affinché una delle 4 condizioni necessarie non si verifichi mai
- **Detection & Recovery**: si lascia che il sistema possa entrare in uno stato di deadlock; individuato un ciclo attraverso "opportuni" algoritmi di rilevazione, si fa il "*kill*" di uno o più processi coinvolti, che vanno riportati al loro stato iniziale (recovery) prima di essere di nuovo eseguiti
- **Avoidance**: attraverso un'allocazione oculata delle risorse

Assicurare l'assenza di deadlock impone costi (in prestazioni, funzionalità) molto alti.
Tali costi sono necessari in alcuni contesti, ma insopportabili in altri.
Si considera il rapporto costo/benefici: se la probabilità che accada un deadlock è sufficientemente bassa, non giustifica il costo per evitarlo.

# Deadlock Prevention 1/2

## Negazione di almeno una delle condizioni di Coffman

- **Mutua esclusione**: per eliminare la mutua esclusione è necessario garantire immediatamente l'uso di una risorsa ad ogni processo che lo richiede (difficilmente realizzabile per risorse non condividibili; non richiesta per risorse condividibili); regola di buona programmazione: allocare le risorse per il minor tempo possibile.
- **Hold and Wait**: per prevenire il verificarsi di questa condizione è necessario fare in modo che quando un processo richiede una nuova risorsa non ne possegga altre.
  - richiede che tutte le risorse debbano essere assegnate *a priori* (*resources pre-allocation*) prima dell'esecuzione: implica un basso utilizzo delle risorse
  - un processo potrebbe richiedere le risorse di cui necessita a blocchi: per ogni nuova richiesta dovrebbe provvedere a rilasciare le risorse precedentemente ottenute; possibile starvation di alcuni processi
  - Basso utilizzo delle risorse; possibile starvation

# Deadlock Prevention 2/2

- **No preemption**: negando la mancanza di prerilascio, si assume che il sistema possa prelazionare (avere, cioè, il diritto di precedenza nell'assegnazione di tutte le risorse); ciò potrebbe avvenire nel caso (poco realistico) che lo stato delle risorse possa essere facilmente salvato e ripristinato: impraticabile per molte risorse
  - Se un processo sta detenendo alcune risorse e ne richiede un'altra che non può essergli assegnata immediatamente, allora tutte le risorse occupate dovranno essere rilasciate.
  - Le risorse rilasciate vengono aggiunte alla lista delle risorse per cui il processo è in attesa.
  - Il processo sarà fatto ripartire soltanto quando può riguadagnare sia risorse precedentemente possedute che quelle che sta richiedendo

- **Circular Wait**: per prevenire l'attesa circolare, alle risorse viene assegnato un identificativo numerico e viene imposto un loro ordinamento totale, cioè che i processi le richiedano rispettando l'ordine prestabilito.
  - Teoricamente fattibile, ma difficile da implementare

# Deadlock Avoidance

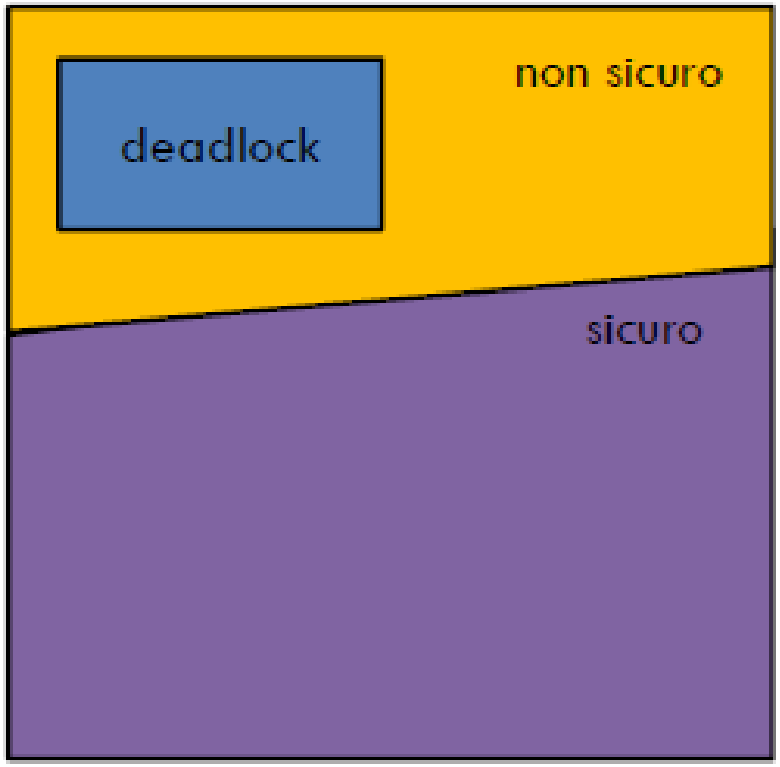
- In questo caso è necessario fornire al sistema informazioni *a priori* sulle risorse (in particolare massime) di cui necessita un processo al fine di garantire che il deadlock non si verifichi mai
  - L'algoritmo esamina dinamicamente lo stato di allocazione delle risorse per assicurare che non si verifichi mai una condizione di attesa circolare
  - Il sistema soddisfa la richiesta di risorse da parte di un processo, solo se sa che il processo potrà soddisfare tutte le sue richieste future
    - Si evitano così eventuali circolarità
    - Molto pesante computazionalmente
    - Difficile determinare in anticipo le risorse di cui necessita un processo
- Lo stato di allocazione delle risorse è definito dal numero di risorse disponibili e assegnate e dal numero massimo di risorse richieste dai processi

# Safe State – Safe Sequence

- Quando un processo richiede una risorsa disponibile, il sistema deve decidere se l'immediata attribuzione della risorsa lascia il sistema in uno stato sicuro
- Safe State è uno stato dal quale è possibile originare una sequenza sicura
- Safe Sequence è un ordinamento di processi che garantisce che tutti i processi possano essere eseguiti sino al loro completamento, anche assumendo che richiedano il massimo delle risorse
- La sequenza $<P_1, P_2, ..., P_n>$ è sicura se per ogni $P_i$, le richieste di risorse che $P_i$ può attualmente fare possono essere soddisfatte da:
  - risorse attualmente disponibili;
  - risorse detenute da tutti i processi Pj, con j < i.
- Se le risorse di cui il processo $P_i$ ha bisogno non sono immediatamente disponibili, allora $P_i$ può aspettare finchè tutti i $P_j$ hanno terminato.
  - Quando hanno finito, $P_i$ può ottenere tutte le risorse necessarie, completare il suo task, restituire le risorse assegnate e terminare.
  - Quando $P_i$ termina, $P_{i+1}$ può ottenere le risorse necessarie, e così via.

# Safe State – Safe Sequence

Algoritmi di Deadlock avoidance garantiscono che il sistema si trovi sempre in uno stato sicuro



# Banker's Algorithm (Dijkstra '65)

- Il SO gestisce un array **AVAILABLE** in cui è indicato per ogni risorsa del sistema il numero di istanze di cui il sistema dispone momentaneamente
- Ogni processo dichiara in fase di avvio un array che definisce il numero massimo di istanze, per ogni tipo di risorsa, che intende utilizzare durante la sua esecuzione: l'array bidimensionale **MAX** raggruppa il numero massimo di istanze di risorse j richieste da ogni processo i
- Perchè il sistema sia **ammissibile**, il massimo di istanze richieste per ciascuna risorsa deve essere minore del totale delle istanze disponibili per tale risorsa
- Il SO mantiene un array bidimensionale **ALLOCATION**, che specifica, per ogni processo i e per ogni risorsa j, il numero di istanze di j allocate ad i
- Per ogni processo si calcola un array **NEED** = MAX – ALLOCATION che fornisce il numero di istanze di risorse j ancora necessarie per ogni processo i, anche nel caso che il processo abbia bisogno di disporre del massimo dichiarato di risorse
- Sulla base di questi dati è possibile stabilire se il sistema è in uno **stato sicuro**

# Habermann theorem

- L' idea: assumiamo l'ipotesi peggiore che ogni processo richieda tutte le risorse di cui ha bisogno per terminare, e verifichiamo se questo può accadere, applicando il seguente procedimento

  - Considera l'elenco dei processi che devono ancora terminare
  - Ripeti
    1. Individua un processo i tale che NEED [i] <= AVAILABLE [i], per tutte le risorse; elimina questo processo dall'elenco
    2. AVAILABLE = AVAILABLE + $MAX_i$
  - Finché hai cancellato tutti i processi dall'elenco, o non puoi più soddisfare il passo 1
  - Se l'elenco è vuoto sei in uno stato sicuro !!!

# The Banker's Algorithm

***Example 1***

5 processes $P_1$ through $P_5$; 3 resource types. **Total number of instances for each resource type:** *A* (10 instances), *B* (5 instances), and *C* (7 instances).

*Snapshot at time $T_0$:*

| | *Allocation* | *Max* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| $P_1$ | 0 1 0 | 7 5 3 | 3 3 2 |
| $P_2$ | 2 0 0 | 3 2 2 | |
| $P_3$ | 3 0 2 | 9 0 2 | |
| $P_4$ | 2 1 1 | 2 2 2 | |
| $P_5$ | 0 0 2 | 4 3 3 | |

| | *Need* |
|---|---|
| | *A B C* |
| $P_1$ | 7 4 3 |
| $P_2$ | 1 2 2 |
| $P_3$ | 6 0 0 |
| $P_4$ | 0 1 1 |
| $P_5$ | 4 3 1 |

Il sistema è in uno **stato ammissibile**.

Il sistema è in uno **stato sicuro** perchè la sequenza $< P_2, P_4, P_5, P_3, P_1>$ soddisfa il criterio di sicurezza.

# The Banker's Algorithm

***Example 2***

$P_2$ makes a Request$_1$ = (1,0,2) ≤ Available = (3,3,2) ⇒ *true.*

| | *Allocation* | *Need* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| $P_1$ | 0 1 0 | 7 4 3 | 2 3 0 |
| $P_2$ | 3 0 2 | 0 2 0 | |
| $P_3$ | 3 0 2 | 6 0 0 | |
| $P_4$ | 2 1 1 | 0 1 1 | |
| $P_5$ | 0 0 2 | 4 3 1 | |

Executing safety algorithm shows that sequence <$P_2$, $P_4$, $P_5$, $P_3$, $P_1$> satisfies safety requirement.

Can request for (3,3,0) by $P_5$ be granted?

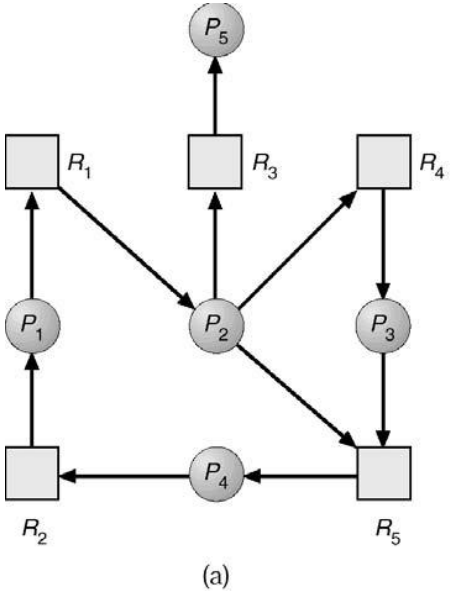
Can request for (0,2,0) by P1 be granted?

# Deadlock Detection 1/2

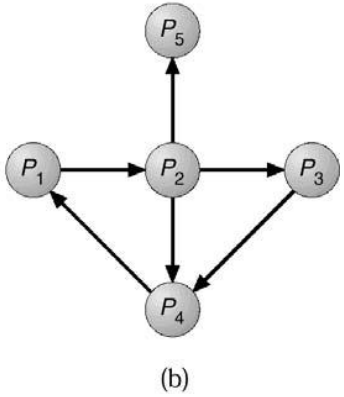
- Permettere al sistema di entrare in uno stato di deadlock.
- Applicare un algoritmo di rilevazione
- Possedere uno schema di recupero

## Il caso di risorse a singola istanza

- Algoritmo di rilevazione
  - Mantiene un grafo di attesa (wait-for graph).
  - I nodi delle risorse sono rimossi e rimangono solo i nodi dei processi.
  - $P_i \rightarrow P_j$ se $P_i$ è in attesa di $P_j$.
- Invocare periodicamente l'algoritmo che cerca un ciclo nel grafo wait-for.
- Un algoritmo per rilevare un ciclo in un grafo richiede $n^2$ operazioni, dove $n$ è il numero di nodi del grafo.




Grafo di allocazione delle risorse




Grafo di attesa

# Deadlock Detection 2/2

## Il caso di risorse a istanze multiple

- Il wait-for graph non è applicabile nel caso di risorse ad istanze multiple.
- Un comune algoritmo per la rilevazione di un deadlock adopera strutture dati variabili nel tempo simili a quelle dell'algoritmo del banchiere:
  - **Available**: vettore di lunghezza *m*,indica il numero di risorse disponibili per ogni tipo.
  - **Allocation**: matrice *n* x *m*, definisce il numero di risorse di ciascun tipo attualmente assegnate a ogni processo.
  - **Request**: matrice *n* x *m*, indica la richiesta corrente di ogni processo. Se *Request* [*i*, *j*] = *k*, allora il processo $P_i$ richiede altre *k* istanze della risorsa di tipo $R_j$.
- L'algoritmo di rilevazione controlla ogni possibile sequenza di allocazione per i processi da completare.

# Deadlock Detection examples 1/2

## Esempio (1/2)

Consideriamo un sistema con 5 processi da $P_1$ a $P_5$ e 3 tipi di risorse:

- A (7 istanze),
- B (2 istanze),
- C (6 istanze).

Supponiamo che all'istante $T_0$ si abbia il seguente stato di allocazione delle risorse:

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| $P_1$ | 0 1 0 | 0 0 0 | 0 0 0 |
| $P_2$ | 2 0 0 | 2 0 2 | |
| $P_3$ | 3 0 3 | 0 0 0 | |
| $P_4$ | 2 1 1 | 1 0 0 | |
| $P_5$ | 0 0 2 | 0 0 2 | |

La sequenza $<P_1, P_3, P_4, P_2, P_5>$ provoca *Finish*[*i*] = true per ogni *i*.

# Deadlock Detection examples 2/2

## Esempio (2/2)

Supponiamo che $P_3$ faccia una richiesta supplementare per un'istanza di tipo C:

| | *Request* | | |
|---|---|---|---|
| | *A* | *B* | *C* |
| $P_1$ | 0 | 0 | 0 |
| $P_2$ | 2 | 0 | 1 |
| $P_3$ | 0 | 0 | 1 |
| $P_4$ | 1 | 0 | 0 |
| $P_5$ | 0 | 0 | 2 |

Stato del sistema?

- Anche se possiamo riprendere le risorse del processo $P_1$, il numero di risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi.
- C'è un deadlock costituito dai processi $P_2$, $P_3$, $P_4$, e $P_5$.

# Recupero dal deadlock (1/2)

- Il ripristino dal deadlock può avvenire:
  - a carico dell'utente che viene informato dal sistema dell'esistenza di uno stallo o per via automatica;
  - abortendo uno o più processi o requisendo risorse detenute da processi in deadlock.

- Abort di processi:
  - Abortire tutti i processi in deadlock (i risultati parziali dell'elaborazione di tutti i processi coinvolti devono essere scartati).
  - Abortire un processo alla volta fino ad eliminare il ciclo di deadlock (è notevole l'overhead poiché dopo ogni abort si deve lanciare l'algoritmo di rilevazione).
  - In quale ordine devono essere terminati i processi?
    - Priorità del processo.
    - Per quanto tempo il processo ha elaborato e per quanto tempo ancora il processo proseguirà prima di completare l'operazione pianificata.
    - Quanti e quali tipi di risorse sono state adoperate dal processo.
    - Di quante altre risorse il processo necessita per completare l'elaborazione.
    - Quanti processi devono essere terminati.
    - Caratteristica del processo (interattivo o batch).

# Recupero dal deadlock (2/2)

- Rilascio anticipato delle risorse (prelazione).
  - Selezione della vittima (minimizzazione del costo).
  - Rollback.
    - Prelazionando una risorsa da un processo occorre riportarlo ad uno stato sicuro e far ripartire il processo da quello stato.
    - La soluzione più semplice è un rollback totale del processo.