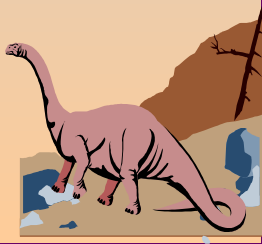
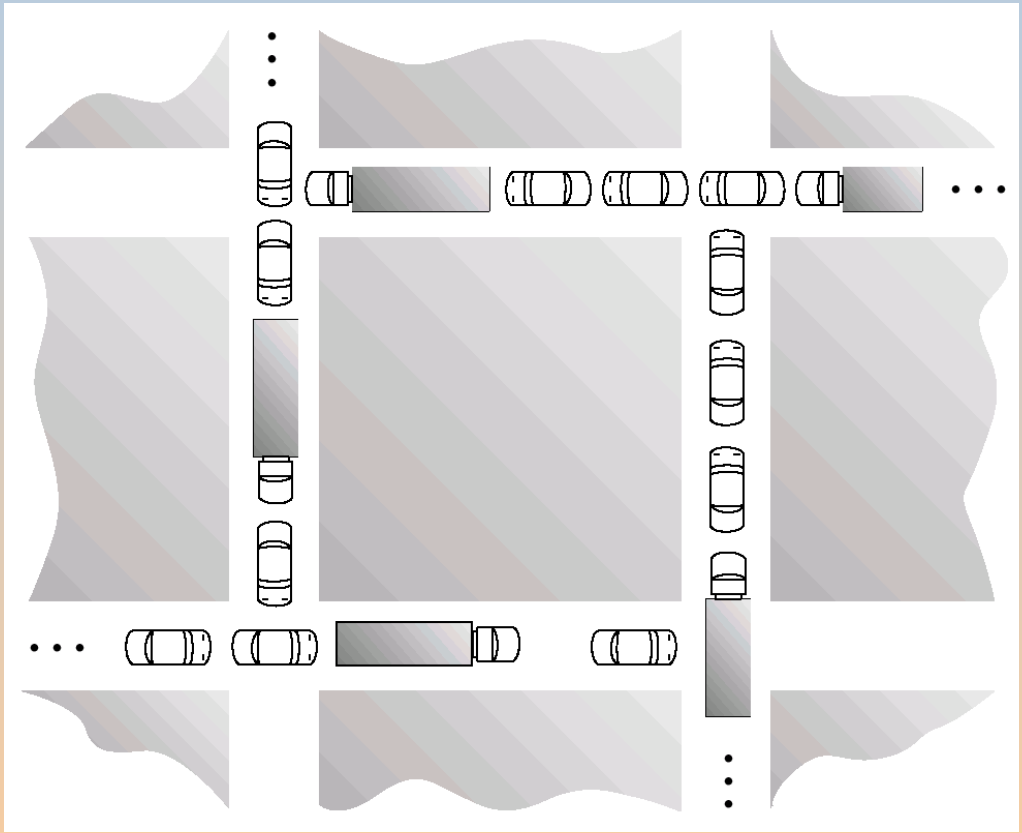
Capitolo 7: Stallo dei processi

- Modello del sistema
- Caratterizzazione delle situazioni di stallo
- Metodi per la gestione delle situazioni di stallo
- Prevenire le situazioni di stallo
- Evitare le situazioni di stallo
- Rilevamento delle situazioni di stallo
- Ripristino da situazioni di stallo

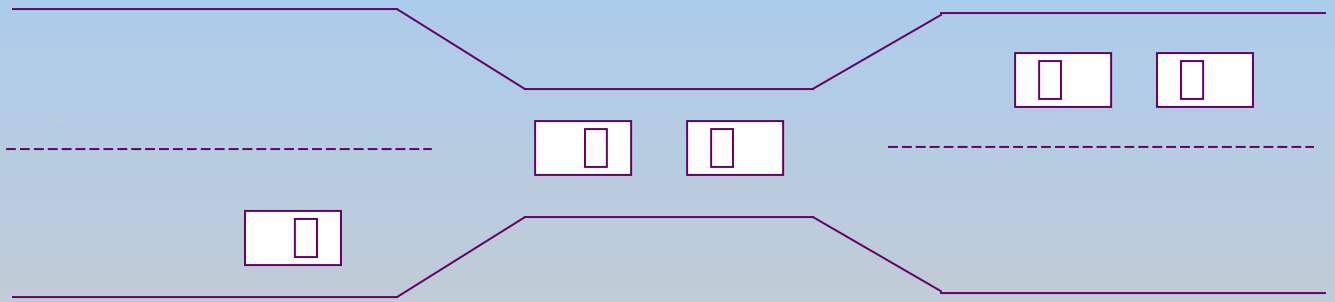


Stallo (*Deadlock*)

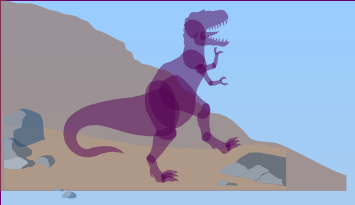
Un insieme di processi e' in stallo se ciascun processo nell'insieme è in attesa di un evento che può essere causato solo da un altro dei processi nell'insieme.



Esempio: attraversamento di un ponte

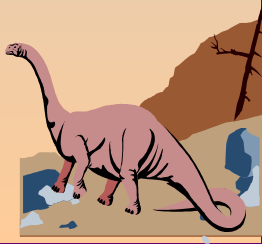


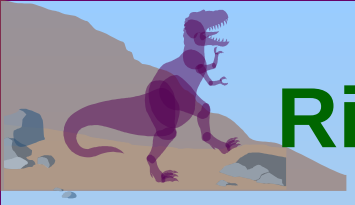
- In ogni istante, sul ponte possono transitare autoveicoli solo in una direzione.
- Ciascuna sezione del ponte può essere vista come una risorsa.
- Se si verifica una situazione di stallo, questa può essere risolta se un'auto torna indietro (rilascia la risorsa ed esegue un ritorno ad uno stato "sicuro" - *rollback*).
- In caso di deadlock, può essere necessario che più auto debbano tornare indietro.
- È possibile si verifichi attesa indefinita (starvation).



Il problema del deadlock

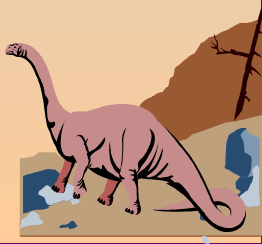
- In un ambiente multiprogrammato più processi possono entrare in competizione per cercare di ottenere risorse condivise.
- Se una risorsa non è correntemente disponibile, il processo richiedente entra in stato di attesa.
- Se le risorse sono trattenute da altri processi, a loro volta in stato di attesa, il processo potrebbe non poter più cambiare il proprio stato.
- Esempio
  - Nel sistema sono presenti solo due unità a nastri
  - P_1 e P_2 hanno già avuto assegnate un'unità ciascuno
  - Entrambi si bloccano e richiedono l'assegnazione di una seconda unità

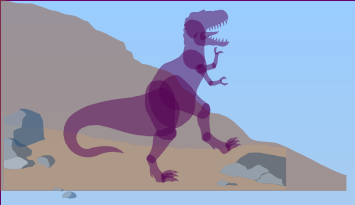




Risorse riutilizzabili e risorse non riutilizzabili

- Risorse riutilizzabili:
  - Questo tipo di risorsa può essere usata in modo sicuro da un processo alla volta,ed essere riutilizzata in seguito da un altro processo
  - Esempi di risorse riutilizzabili : processori, canali I/O, memoria centrale e secondaria, basi di dati, semafori, file, dispositivi
- Risorse non riutilizzabili:
  - Questo tipo di risorsa dopo essere stata creata,viene consumata (distrutta)
  - Esempi di risorse non riutilizzabili : interrupt, segnali, messaggi, dati contenuti nel buffer I/O, etc.



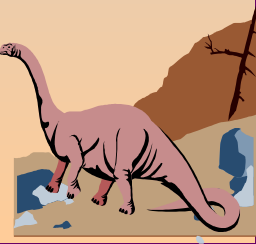


Modello di sistema

- Risorse di tipo $R_1, R_2, \ldots, R_m$

  (Cicli di CPU, spazio di memoria, file, dispositivi di I/O)

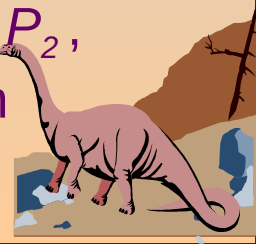
- Sono disponibili W_i istanze di ciascuna risorsa di tipo R_i.
- Nelle ordinarie condizioni di funzionamento un processo può servirsi di una risorsa solo se rispetta la seguente sequenza:
  - **Richiesta** — se la richiesta non può essere soddisfatta immediatamente, perché la risorsa non è libera, il processo richiedente deve attendere fino a che non può acquisire la risorsa.
  - **Utilizzo** — il processo può operare sulla risorsa.
  - **Rilascio** — il processo rilascia la risorsa.
- Richiesta e rilascio di risorse sono realizzati con apposite system call:
  - *request/release device*, *open/close file*, *allocate/free memory*, o *wait* e *signal* su semafori.



Caratterizzazione dei deadlock

Una situazione di deadlock può verificarsi *solo se* valgono ***tutte e quattro*** le seguenti condizioni simultaneamente :

- **Mutua esclusione:** almeno una risorsa deve essere "non condivisibile", cioè può essere usata da un solo processo alla volta. Se un altro processo richiede questa risorsa, si deve bloccare il processo richiedente fino al rilascio della risorsa.
- **Possesso ed attesa:** un processo, in possesso di almeno una risorsa, attende di acquisire ulteriori risorse già in possesso di altri processi.
- **Impossibilità di prelazione:** non esiste un diritto di prelazione. Una risorsa può essere rilasciata dal processo che la possiede solo volontariamente, al termine del suo utilizzo.
- **Attesa circolare:** deve esistere un insieme $\{P_0, P_1, \ldots, P_n\}$ di processi in attesa, tali che P_0 è in attesa di una risorsa che è posseduta da P_1, P_1 è in attesa di una risorsa posseduta da P_2, …, P_{n-1} è in attesa di una risorsa posseduta da P_n, e P_n è in attesa di una risorsa posseduta da P_0 .

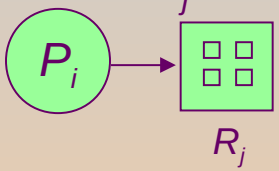


Grafo di allocazione risorse (II)

- Processo

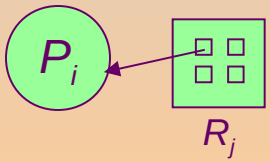
- Tipo di risorsa con 4 istanze

- P_i richiede un'istanza di R_j

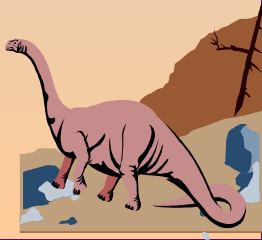


Arco di richiesta

- P_i possiede un'istanza di R_j

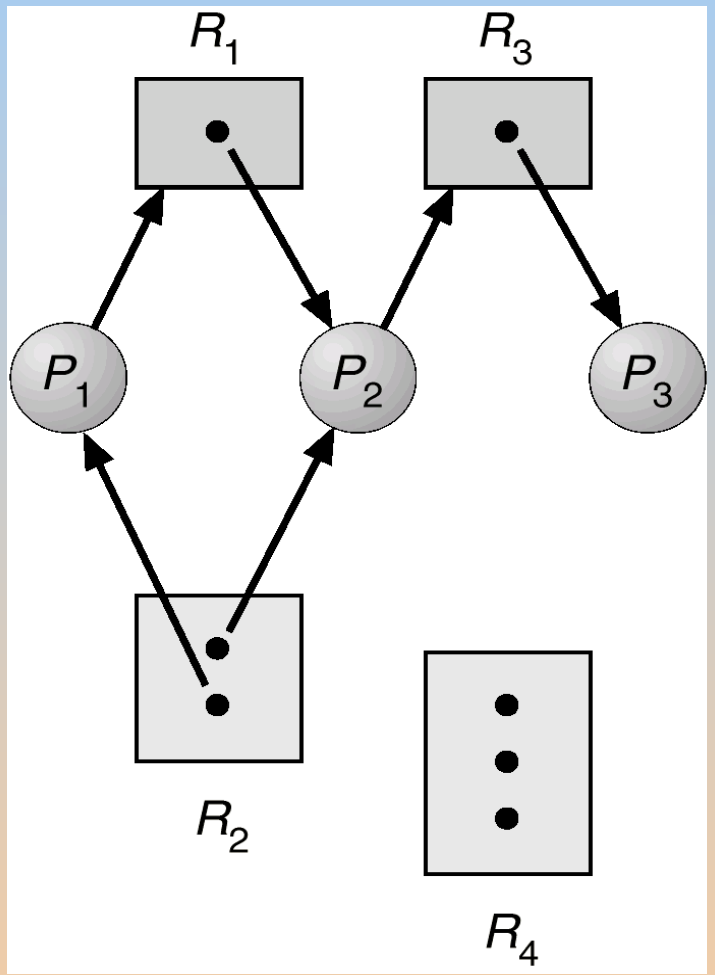


Arco di assegnazione

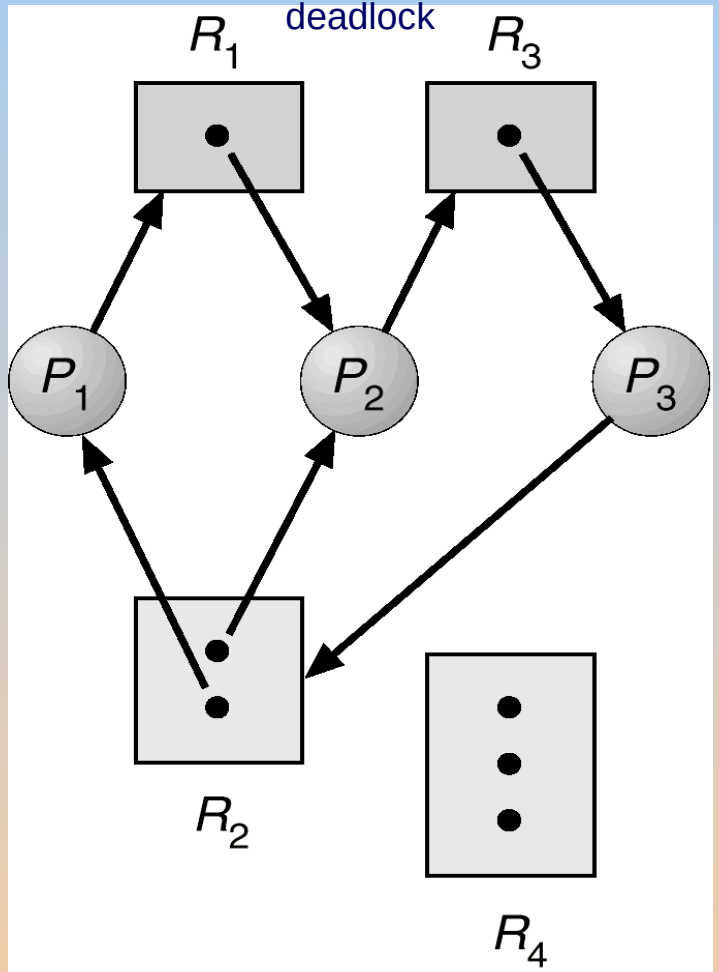


Esempi di grafo di allocazione risorse

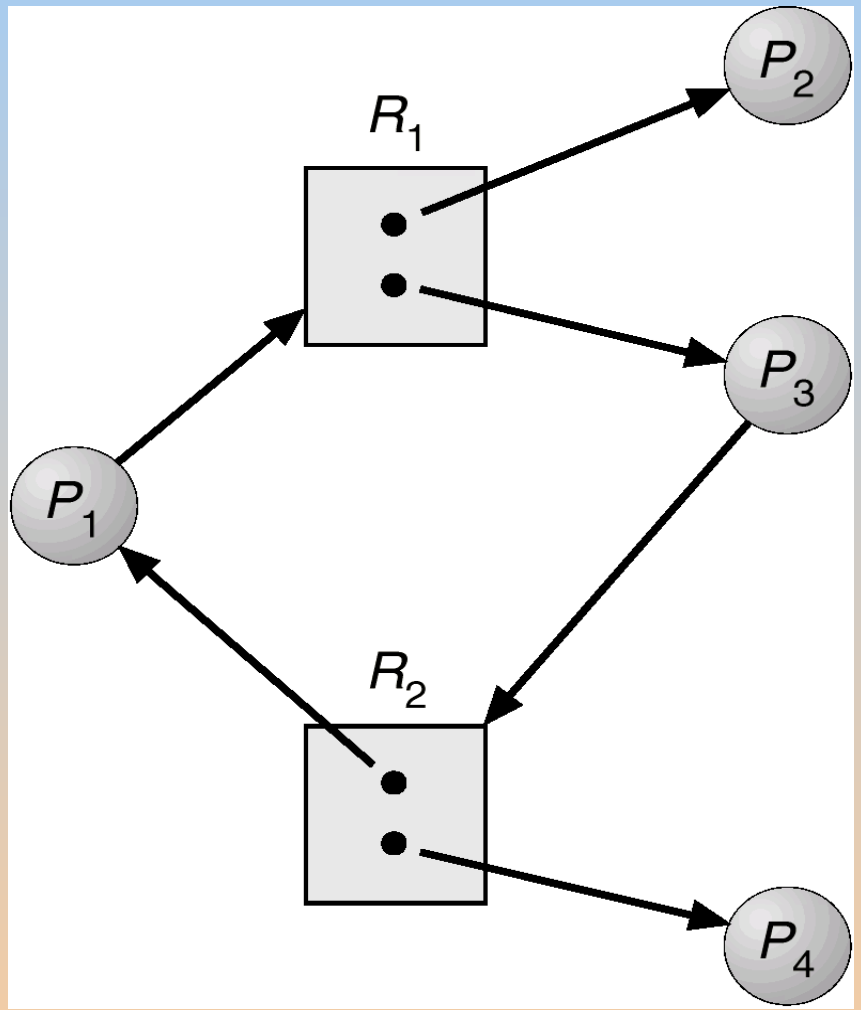
Grafo di allocazione risorse



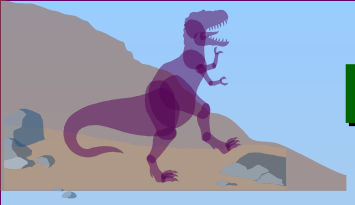
Grafo di allocazione risorse con deadlock



Grafo di allocazione risorse (III)

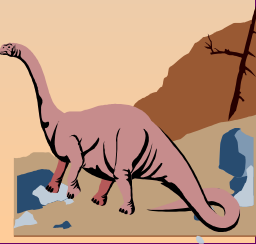


- Se il grafo non contiene cicli ⇒ non ci sono deadlock.
- Se il grafo contiene un ciclo ⇒
  - Se vi è una sola istanza per ogni tipo di risorsa, allora si ha un deadlock.
  - Se si hanno più istanze per tipo di risorsa, allora il deadlock è possibile (ma non certo).
  - Nell'esempio abbiamo un ciclo ma non un deadlock.



Metodi per la gestione dei deadlock

- Assicurare che il sistema non entri *mai* in uno stato di deadlock.
  - **Prevenire i deadlock**:
    - evitare che contemporaneamente si verifichino mutua esclusione, possesso e attesa, impossibiltà di prelazione e attesa circolare ⇒ basso utilizzo risorse e throughput ridotto.
  - **Evitare i deadlock**:
    - evitare gli stati del sistema a partire dai quali si può evolvere verso un deadlock.
- Permettere al sistema di entrare in uno stato di deadlock, quindi ripristinare il sistema.
  - **Determinare la presenza di un deadlock**.
  - **Ripristinare il sistema da un deadlock**.
- Ignorare il problema e “fingere” che i deadlock non avvengano mai nel sistema; impiegato dalla maggior parte dei S.O., incluso UNIX.

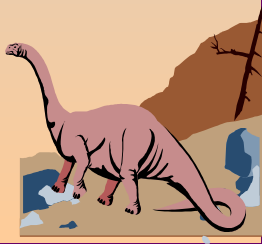


Prevenire le situazioni di stallo

Limitare le modalità di richiesta e accesso alle risorse.

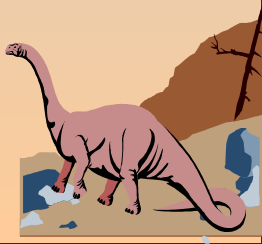
- **Mutua esclusione** - non è richiesta per risorse condivisibili; deve valere invece per le risorse che non possono essere condivise.

- **Possesso e attesa** - occorre garantire che, quando un processo richiede una risorsa, non ne possieda altre.
  - Richiedere ad un processo di allocare tutte le risorse necessarie prima che inizi l'esecuzione, o consentire la richiesta di risorse solo quando il processo non ne possiede alcuna.
  - Basso impiego delle risorse. E' possibile che si verifichi attesa indefinita (starvation).



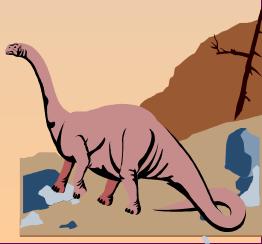
Prevenire le situazioni di stallo (II)

- **Impossibilità di prelazione**
  - Quando un processo, che possiede risorse, richiede un'altra risorsa che non gli può essere allocata immediatamente, potrà rilasciare le risorse possedute se sono richieste da un altro processo.
  - Le risorse rilasciate vengono aggiunte alla lista delle risorse che il processo attende.
  - Il processo viene avviato nuovamente solo quando può ottenere sia le risorse che ha rilasciato che le nuove risorse.
- **Attesa circolare** - si impone un ordinamento totale su tutti i tipi di risorsa e ciascun processo potrà richiedere risorse solo in ordine crescente di numerazione.
  - Supponiamo che $R = \{R_1 , R_2 , \ldots , R_n \}$ sia l'insieme dei tipi di risorse
  - L'insieme delle risorse viene numerato tramite una funzione iniettiva *f* che associa ad ogni risorsa un numero intero distinto.
  - Se un processo possiede *k* istanze di una risorsa R_i, in seguito potrà richiedere allocazioni di risorsa R_j se e solo se $f(R_j) > f(R_i)$.
  - In alternativa, si può stabilire che un processo, prima di richiedere un'istanza di R_j, rilasci qualsiasi R_i tale che $f(R_i) > f(R_j)$.



Prevenire le situazioni di stallo (III)

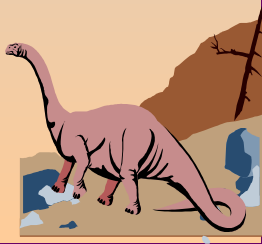
- **Attesa circolare** (cont.) -
  - Se si usa uno di questi due protocolli, la condizione di attesa circolare non può sussistere.
  - Infatti, supponiamo per assurdo che esista un'attesa circolare.
  - Siano i processi coinvolti $\{P_0, P_1, \ldots, P_n\}$, dove P_i attende una risorsa R_i posseduta dal processo P_{i+1} (modulo n: P_n attende una risorsa R_n posseduta da P_0).
  - Poiché P_{i+1} possiede R_i mentre richiede R_{i+1}, è necessario che sia verificata la condizione $f(R_i) < f(R_{j+1})$ per tutti gli *i*, ma ciò implica che $f(R_0) < f(R_1) < \ldots < f(R_n) < f(R_0)$, e quindi, per proprietà transitiva che $f(R_0) < f(R_0)$, il che è assurdo, quindi non può esserci attesa circolare.



Evitare le situazioni di stallo

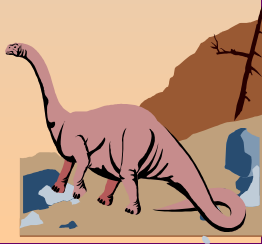
Presuppone che il sistema conosca “a priori” informazioni addizionali su quali saranno le richieste future dei processi.

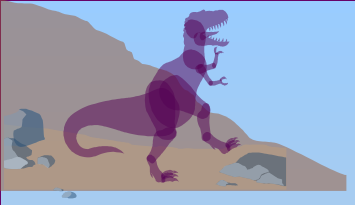
- Il modello più semplice richiede che ciascun processo dichiari il *numero massimo* di risorse di ciascun tipo di cui avrà bisogno.
- Data un’informazione a priori per ogni processo sul numero massimo di risorse di ciascun tipo che il processo potrà richiedere, si può costruire un’agoritmo capace di assicurare che il sistema non entri in stallo.
- L’algoritmo di *deadlock–avoidance* esaminerà dinamicamente lo stato di allocazione delle risorse per assicurare che non si verifichi mai una condizione di attesa circolare.
- Lo *stato di allocazione delle risorse* sarà definito dal numero di risorse disponibili ed allocate, e dal massimo numero di risorse richieste da ciascun processo.



Stato sicuro

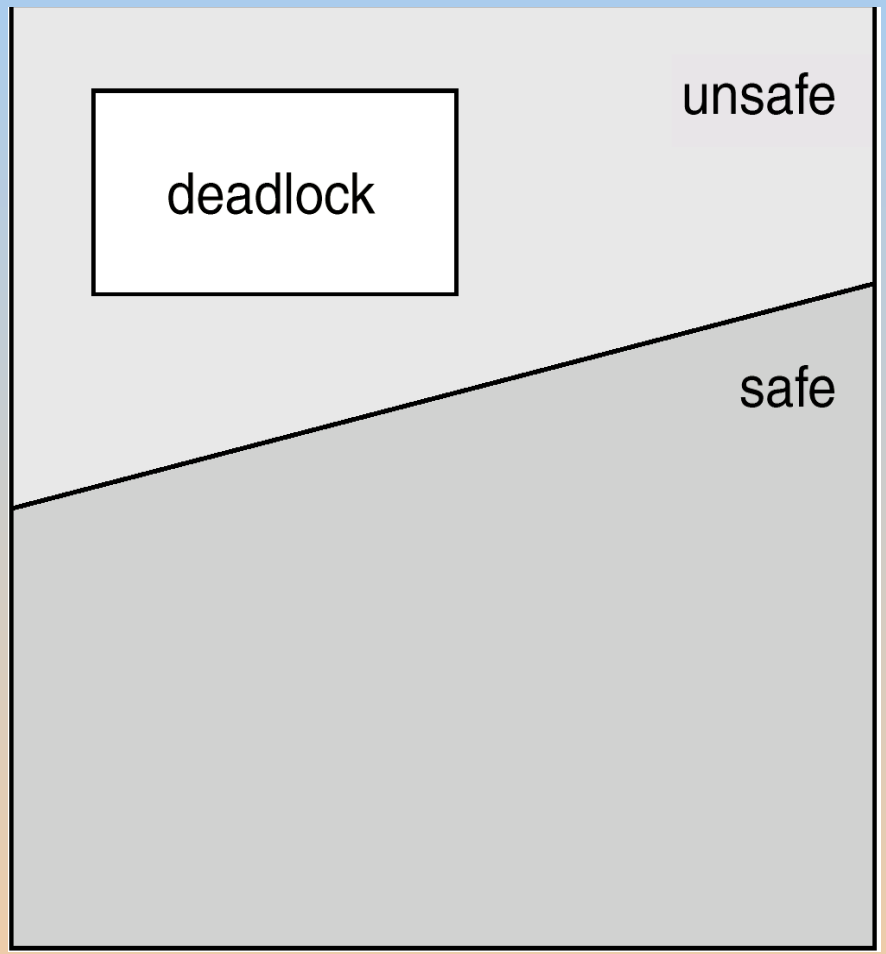
- Quando un processo richiede una risorsa disponibile, il sistema deve decidere se l'allocazione immediata della risorsa lascerà il sistema in uno **stato sicuro**.
- Il sistema si trova in uno stato sicuro solo se esiste almeno una **sequenza sicura** di esecuzione di tutti i processi.
- La sequenza $<P_1, P_2, \ldots, P_n>$ è sicura se, per ogni P_i, le risorse che P_i può ancora richiedere possono essergli allocate sfruttando le risorse disponibili + le risorse possedute da tutti i P_j, con $j < i$.
  - Se le richieste di P_i non possono essere soddisfatte immediatamente, allora P_i può attendere finché P_j ha terminato.
  - Quando P_j ha terminato, P_i può ottenere le risorse richieste, eseguire i suoi compiti, restituire le risorse allocate e terminare.
  - Quando P_i termina, P_{i+1} può ottenere le risorse richieste, etc.

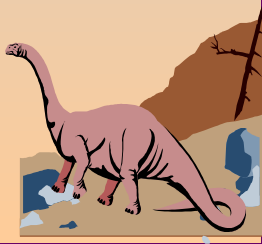




Stato sicuro (II)

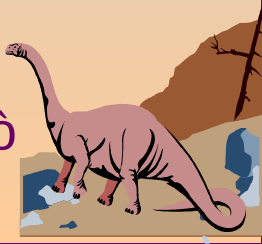
- Se un sistema è in stato sicuro $\Rightarrow$ non si evolve verso il deadlock.
- Se un sistema è in uno stato non sicuro $\Rightarrow$ si può evolvere verso un deadlock.
- Evitare le situazioni di stallo $\Rightarrow$ garantire che un sistema non entri mai in uno stato non sicuro.



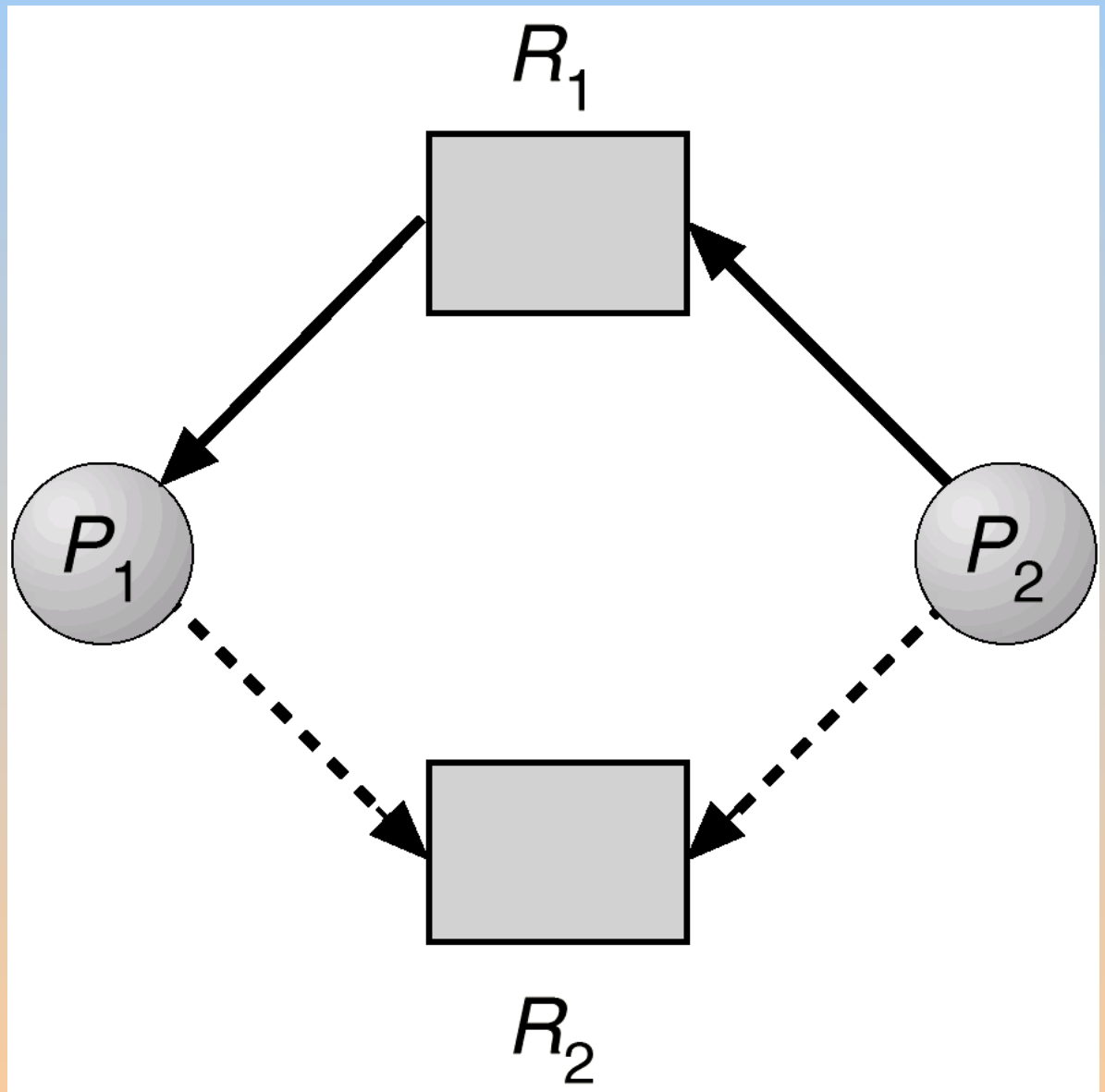


Algoritmo con grafo di allocazione risorse

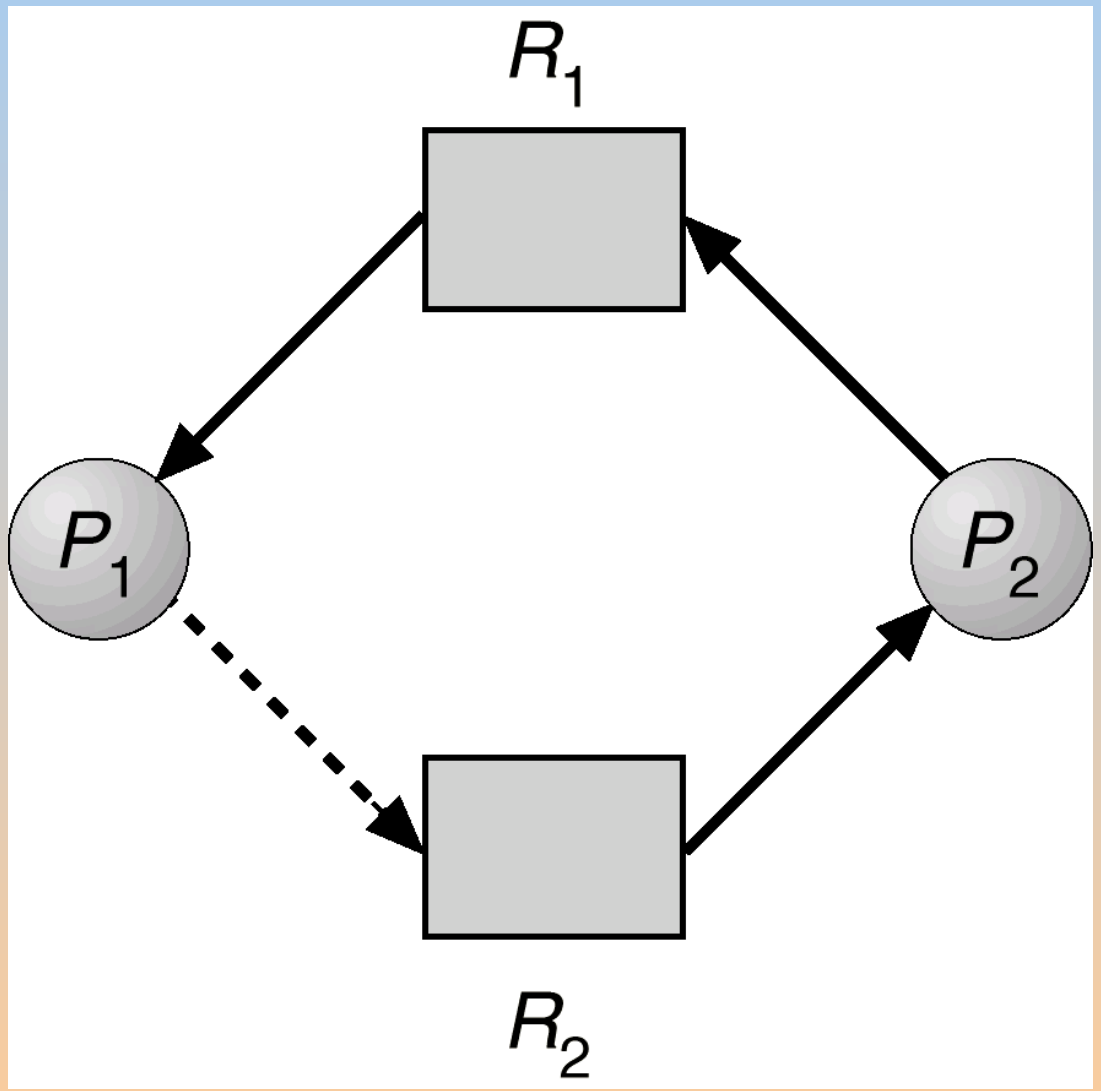
- Situazione in cui è disponibile **solo un'istanza** di ciascuna risorsa.
- Oltre agli archi di assegnazione e di richiesta si introduce **l'arco di reclamo**.
- Un arco di reclamo $P_i \rightarrow R_j$ (rappresentato da una linea tratteggiata) indica che il processo P_i può richiedere la risorsa R_j.
- Un arco di reclamo viene convertito in un arco di richiesta quando il processo P_i richiede effettivamente la risorsa R_j.
- Quando una risorsa viene rilasciata da un processo, l'arco di assegnamento viene riconvertito in un arco di reclamo.
- Le risorse devono venir reclamate *a priori* nel sistema:
  - prima che il processo P_i inizi l'esecuzione, tutti i suoi archi di reclamo devono essere già inseriti nel grafo di allocazione risorse.
- La sicurezza si controlla con un algoritmo per il rilevamento di un ciclo:
  - ($O(n^2)$, dove n è il numero di vertici del grafo).
- Se non esiste alcun ciclo $\Rightarrow$ stato sicuro.
- Se esiste un ciclo $\Rightarrow$ stato non sicuro, l'arco di reclamo non può diventare di assegnazione.



Grafo di allocazione risorse

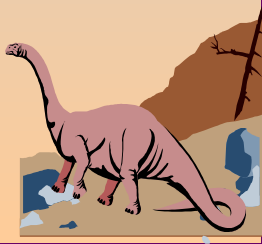


Uno stato non sicuro in un grafo di allocazione risorse



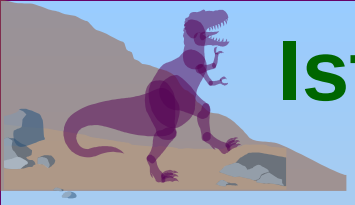
Algoritmo del banchiere

- Permette di gestire situazioni in cui si hanno più istanze di una risorsa (a differenza dell'algoritmo con grafo di allocazione risorse).
- Ciascun processo deve dichiarare a priori il massimo impiego di risorse.
- Quando un processo richiede una risorsa può non essere servito istantaneamente.
- Quando ad un processo vengono allocate tutte le risorse deve restituirle entro un tempo finito.



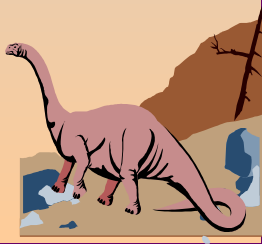
Rilevamento delle situazioni di stallo

- Se un sistema non utilizza un algoritmo per prevenire o evitare i deadlock, permette al sistema di entrare in uno stato di deadlock.
- E' possibile tuttavia utilizzare in questa situazione uno schema di rilevamento e ripristino dal deadlock.
- Sono necessari in questo caso due algoritmi:
  - Algoritmo di rilevamento del deadlock
  - Algoritmo di ripristino dal deadlock
- Uno schema di rilevamento e ripristino dal deadlock richiede un overhead che include:
  - i costi operativi per la memorizzazione delle informazioni necessarie
  - i costi operativi per l'esecuzione dell' algoritmo di rilevamento
  - i potenziali costi legati al ripristino da una situazione di deadlock.



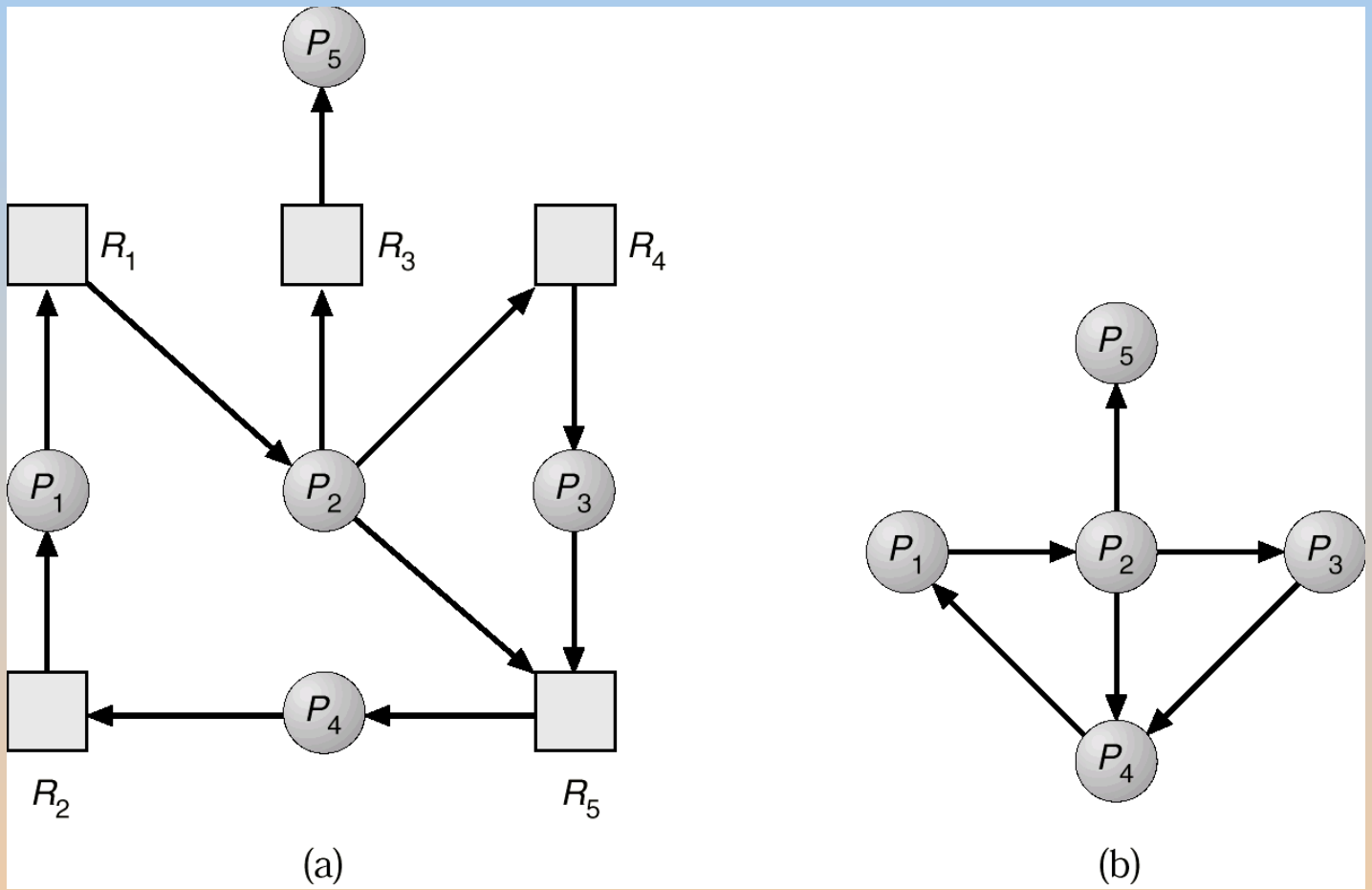
Istanza singola di ciascun tipo di risorsa

- Se tutte le risorse hanno una sola istanza è possibile creare un algoritmo di rilevamento che impiega un ***grafo di attesa***:
  - I nodi rappresentano i processi.
  - L'arco $P_i \rightarrow P_j$ esiste se P_i è in attesa che P_j rilasci una risorsa che gli occorre.
- Periodicamente viene richiamato un algoritmo che verifica la presenza di cicli nel grafo.
- Un algoritmo per rilevare cicli in un grafo richiede un numero di operazioni dell'ordine di n^2, dove n è il numero di vertici del grafo.



Grafo di allocazione risorse e grafo di attesa corrispondente

Un arco $P_i \rightarrow P_j$ esiste nel grafo di attesa, se e solo se il corrispondente grafo di allocazione risorse contiene i due archi $P_i \rightarrow R_q$, $R_q \rightarrow P_j$, per una qualche risorsa R_q.

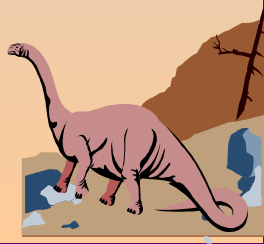


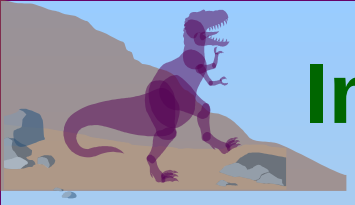
Grafo di allocazione risorse

Grafo di attesa corrispondente

Più istanze di ciascun tipo di risorsa

- Lo schema con grafo di attesa non può essere utilizzato con sistemi con più istanze per risorsa.
- Esiste comunque un altro algoritmo applicabile a tali sistemi che richiede un numero di operazioni dell'ordine di $\vartheta(m \times n^2)$ per determinare se il sistema è in uno stato di deadlock.



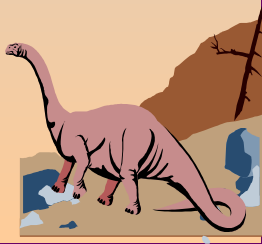


Impiego dell'algoritmo di rilevamento

- Quando e quanto spesso richiamare l'algoritmo di rilevamento dipende da:
  - Frequenza (presunta) con la quale si verificano i deadlock;
  - Numero di processi che sarebbero influenzati da tale deadlock.

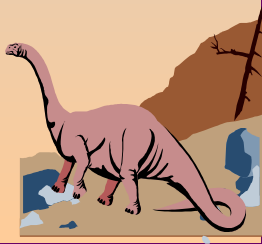
- Non è conveniente richiamare l'algoritmo in momenti arbitrari:
  - nel grafo delle risorse possono coesistere molti cicli e, normalmente, non si può stabilire quale dei processi coinvolti abbia "causato" la situazione di stallo.

- L' algoritmo può essere richiamato:
  - ogni volta che una richiesta di assegnazione non può essere soddisfatta immediatamente
  - quando l'utilizzo della CPU scende al di sotto del 40%.



Ripristino da situazioni di stallo

- Una situazione di deadlock può essere affrontata in vari modi.
- Si può informare l' operatore che ci si trova in una situazione di deadlock,in modo che possa gestirlo manualmente
  - (per Windows Ctrl + Alt + Canc)
- Un' altra soluzione lascia al sistema il ripristino automatico dallo stato di deadlock.
- Un deadlock può essere eliminato in due modi :
  - Il primo prevede l' abort dei processi per eliminare l'attesa circolare.
  - Il secondo richiede la prelazione di alcune risorse.

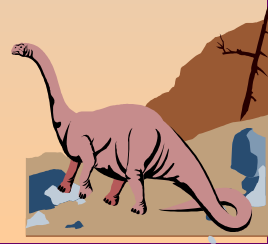


Terminazione di processi

- Per eliminare i deadlock attraverso la terminazione di processi possono essere utilizzati due metodi:

- **Terminazione di tutti i processi in stallo**:
  - questa operazione può risultare molto costosa poichè i processi possono aver già effettuato calcoli per parecchio tempo.

- **Terminazione di un processo alla volta fino all'eliminazione del ciclo di deadlock**:
  - anche questo metodo può risultare costoso, poichè dopo aver terminato un processo si deve richiamare un algoritmo di rilevamento per stabilire se esiste ancora un deadlock.

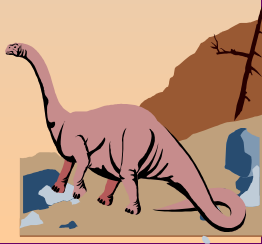


Terminazione di processi (II)

- La terminazione di un processo non è un' operazione banale.
- Se il processo si trova a metà del suo lavoro (aggiornamento di un file, stampa di un dato …) la sua terminazione comporta la necessità di riportare il sistema in uno stato corretto.
- Se viene utilizzato un metodo di terminazione parziale, occorre determinare quale/i processi eliminare per effettuare il ripristino.
- Dovrebbero essere terminati i processi la cui eliminazione richiede il "minimo costo".
- In quale ordine si decide di terminare i processi? I fattori significativi sono:
  - La priorità del processo.
  - Il tempo di computazione trascorso e il tempo ancora necessario al completamento del processo.
  - Quantità e tipo di risorse impiegate.
  - Risorse ulteriori necessarie al processo per terminare il proprio compito.
  - Numero di processi che devono essere terminati.
  - Tipo di processo: interattivo o batch.

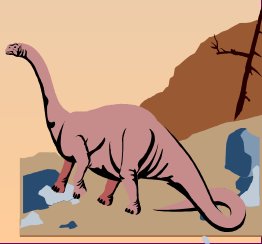
Prelazione di risorse

- Per eliminare un deadlock è possibile prelazionare le risorse dei vari processi fino ad arrivare all'interruzione dello stato di deadlock.
- Le risorse possono cioè essere prese dal processo corrente e allocate ad un altro.
- La scelta del processo da cui prelazionare le risorse dipende dalla facilità con cui si può ritornare in uno stato sicuro.



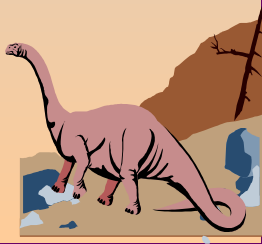
Prelazione di risorse (II)

- È comunque necessario risolvere i seguenti problemi
- **Selezione di una vittima:**
  - È necessario stabilire quali risorse e quali processi sottoporre a prelazione.
  - I costi includono parametri come il numero di risorse possedute da un processo in stallo e il tempo già consumato dal processo.
- **Ristabilimento di un precedente stato sicuro:**
  - Un processo a cui è stata sottratta una risorsa non può continuare normalmente la sua esecuzione.
  - Deve ritornare ad uno stato sicuro da cui ripartire.
  - Essendo difficile stabilire quale stato sia sicuro è più semplice, quando possibile, terminare il processo e poi riavviarlo.



Prelazione di risorse (III)

- **Attesa indefinita:**
  - La selezione della vittima della prelazione è basata su fattori di costo.
  - Può quindi capitare che venga selezionato sempre lo stesso processo.
  - In questo caso il processo non è mai in grado di portare a termine il suo compito.
  - Per evitare starvation le risorse non possono essere sottratte sempre allo stesso processo.
  - È quindi necessario assicurare che il processo sia designato come vittima al più un numero finito di volte includendo il numero di terminazioni nel fattore di costo.



Approccio combinato alla gestione del deadlock

- Gli approcci visti fino a questo momento non possono da soli far fronte a tutti i problemi connessi alla allocazione delle risorse

- Una soluzione è di combinare i tre approcci di base:
  - prevenzione
  - evitare i deadlock
  - rilevamento

  Suddividendo le risorse in classi gerarchicamente ordinate e impiegando le tecniche più appropriate per gestire i deadlock in ciascuna classe.