Sistemi operativi

3. Deadlock

Mauro Fiorentini

3. Deadlock

Deadlock

- Situazione di stallo, nella quale più processi attendono un evento, che non può accadere.
- Si verifica se due o più processi tengono impegnata una risorsa e ne richiedono contemporaneamente un'altra, detenuta da un altro.
  - La risorsa può essere fisica (memoria, una periferica) o logica (l'accesso a una regione critica).

Condizioni per un deadlock

- Perché si verifichi un deadlock sono necessarie e sufficienti 4 condizioni:
  - **mutua esclusione**;
    - le risorse non possono essere condivise;
  - **hold & wait**;
    - i processi detengono una risorsa mentre ne richiedono un'altra;
  - **non preemption**;
    - le risorse non possono essere sottratte a un processo da un altro;
  - **attesa circolare**.

Gestione dei deadlock

- Un sistema può:
  - scoprire i deadlock;
  - prevenire i deadlock, impedendo con la sua stessa architettura il verificarsi delle 4 condizioni;
  - evitare i deadlock, allocando le risorse in modo da evitare le attese circolari.

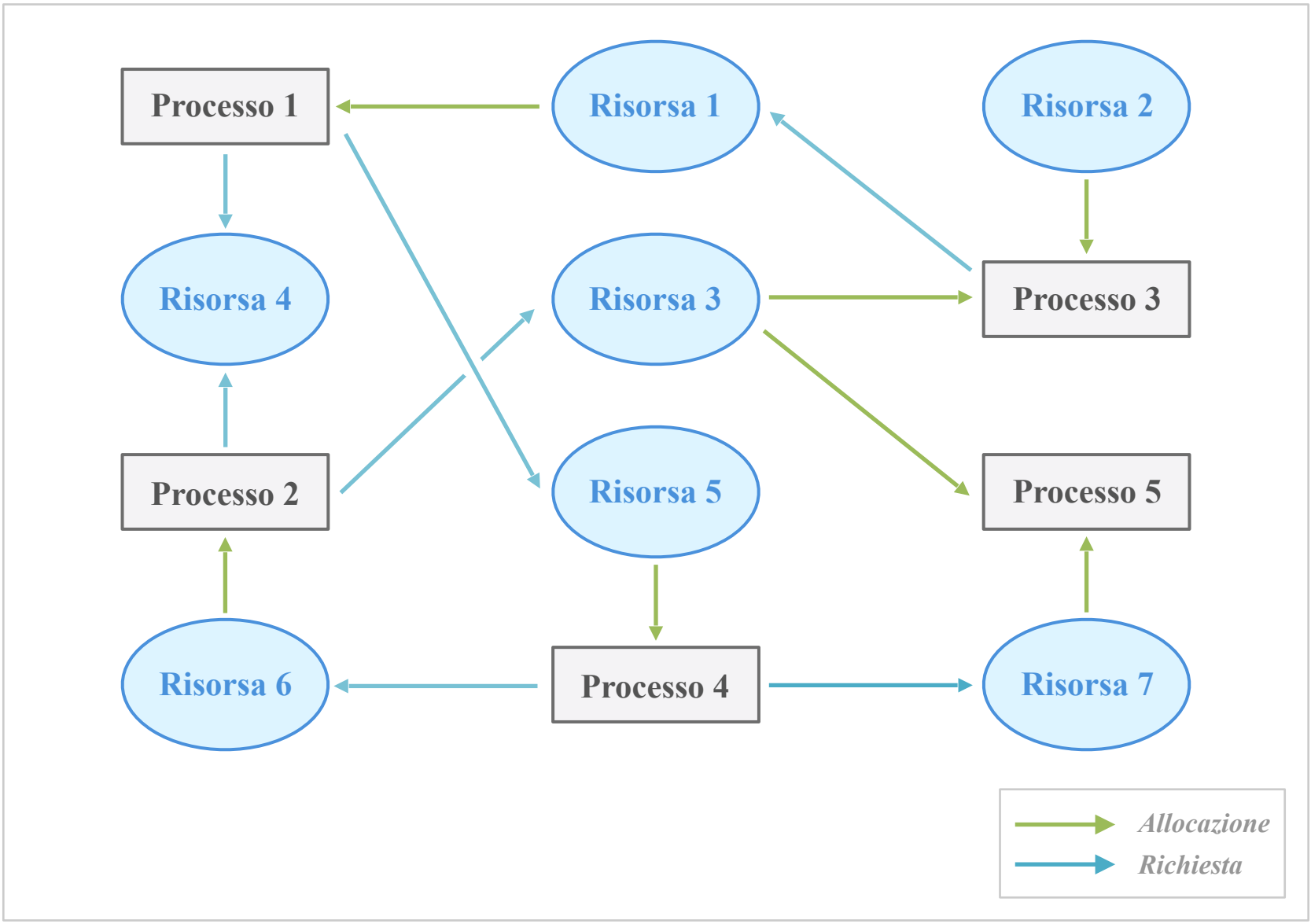
Scoperta dei deadlock

- Un deadlock non coinvolge necessariamente tutti i processi del sistema.
- Per stabilire se alcuni processi sono in deadlock non basta verificare se sono in attesa.
  - In un sistema vi sono normalmente molti processi in attesa, anche per tempi lunghi.

Grafo di allocazione

- Un grafo nel quale processi e risorse costituiscono i nodi; gli archi che vanno dai processi alle risorse rappresentano richieste, quelli che vanno dalle risorse ai processi rappresentano le allocazioni.

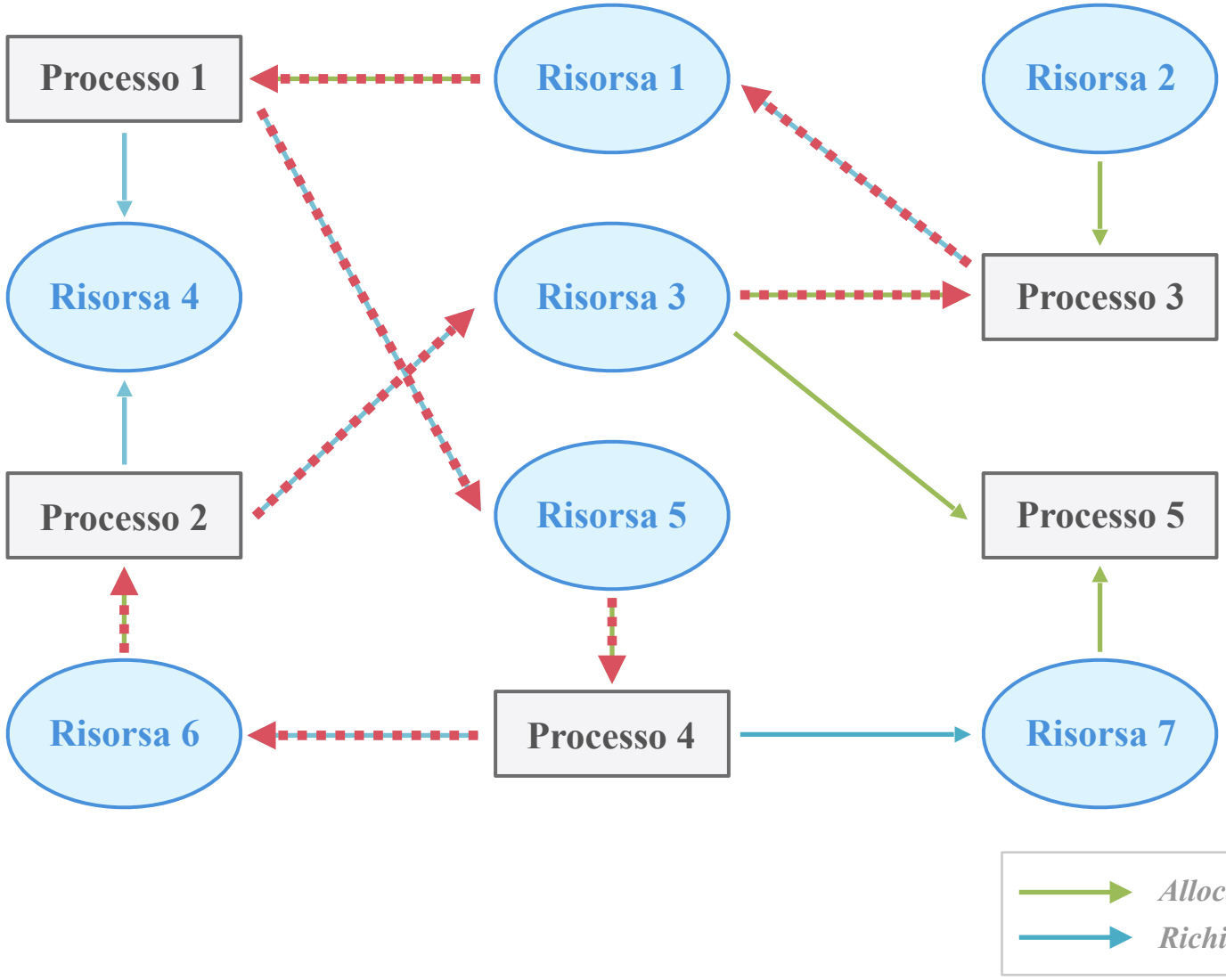
Esempio di grafo di allocazione



Ricerca dei deadlock

- Dato che le prime tre condizioni per un deadlock sono generalmente implicite nell'architettura del sistema, per determinare se vi è un deadlock bisogna costruire il grafo di allocazione e determinare se contiene cicli.
  - Tutti e soli i processi che entrano nei cicli sono in deadlock.
  - Il sistema deve tenere traccia delle richieste, oltre che delle allocazioni.

Esempio di deadlock



Rimedio al deadlock

- Scoperto il deadlock, il sistema non può fare altro che uccidere uno alla volta i processi coinvolti, sino a quando la situazione si sblocca.
  - La scelta del processo non è semplice.
  - Bisogna evitare conseguenze catastrofiche.
  - Si possono annullare tutte le operazioni di I/O del processo da eliminare (rollback) e farlo ripartire.
    - Si rischia la starvation.
- In alternativa il sistema può richiedere l'intervento di un operatore.

Prevenzione dei deadlock

- Si basa sull'impedire il verificarsi di una delle 4 condizioni.
  - Mutua esclusione: non c'è molto da fare.
  - Hold & wait: preallocazione delle risorse o concederle una alla volta.
  - Non preemption: si possono costringere i processi ad allocare le risorse tutte insieme.
  - Attesa circolare: ordinamento delle risorse.

Eliminazione di hold & wait

- Preallocare le risorse vuol dire costringere un processo a dichiarare in anticipo cosa utilizzerà.
  - Poco flessibile, spesso inapplicabile: come può un'applicazione interattiva prevedere le richieste dell'utente?
- Concedere le risorse una alla volta può ridurre il parallelismo.
  - C'é il rischio di attese indefinite.

Eliminazione di hold & wait con i monitor

- Una soluzione nei linguaggi con costrutti simili al monitor consiste nel vietare le chiamate a funzioni di un monitor dall'interno degli altri.
  - Limita la flessibilità dei monitor.
  - Equivale a concedere una risorsa per volta.
  - Usata in CHILL e Modula (Wirth 1977).

Eliminazione della non preemption

- Un processo che si pone in attesa rilascia tutte le risorse che detiene.
  - Dovrà riacquisirle tutte di nuovo.
  - Poco efficiente: provoca cicli di acquisizione e rilascio.
  - Talvolta inapplicabile.
  - C'è il rischio di attese indefinite: un processo può aver bisogno di poche risorse, ma non riuscire mai a ottenerle tutte.

Eliminazione dell'attesa circolare

- Si ordinano le risorse gerarchicamente.
- Un processo può chiedere solo risorse di livello **superiore** a quelle che detiene.
  - Altrimenti deve rilasciarne una parte e richiederle di nuovo in ordine.
  - Può essere inefficiente, se un processo non è in grado di prevedere a priori quali risorse utilizzerà.
  - Le risorse possono essere sottoutilizzate, riducendo il parallelismo.
    - Possono essere vietate anche richieste che non provocherebbero deadlock.

Evitare i deadlock

- L'algoritmo del banchiere evita il deadlock:
  - i processi dichiarano in anticipo quali e quante risorse utilizzeranno nel caso peggiore;
  - un semplice conteggio permette di verificare che il totale di risorse che i processi in esecuzione potrebbero richiedere non superi il numero di risorse disponibili;
  - l'esecuzione di un processo che potrebbe portare al superamento dei limiti viene rinviata.

Svantaggi dell'algoritmo del banchiere

- Sottoutilizzo di risorse e parallelismo.
  - Permette l'esecuzione di meno processi di quanti potrebbero effettivamente essere eseguiti.
  - Applicabile solo in caso di risorse disponibili in un certo numero di copie identiche (memoria, stampanti, periferiche).
  - Praticamente inapplicabile se le risorse sono in gran numero e uniche (file, record, aree critiche).
  - Crea problemi di scelta dei processi da eseguire, per evitare la starvation.

Conclusioni

- Molti sistemi operativi (Unix, NT, VMS) **non garantiscono** l'assenza di deadlock.
- L'onere della prevenzione **ricade sulle singole applicazioni**.
  - Nel casi di sistemi real-time il problema è molto delicato.