# Deadlock

Il deadlock si presenta quando un processo **è bloccato da un evento che solo un altro processo bloccato può provocare.**

Condizioni per il deadlock:

1. *Mutual exclusion:* solo un processo alla volta può usare la risorsa.
2. *Hold-and-Wait*: un processo mantiene una risorsa mentra aspetta un evento.
3. *No Pre-emption:* un processo non può essere forzato a rilasciare la risorsa.

> Queste sono necessarie ma non sufficenti.

4. *Circular wait:* una catena di processi, in cui almeno uno tiene le risorse necessarie ad un altro.

# Approcci al deadlock:

- **Prevenzione:** elimino le condizioni per averlo.
  - *Indiretta*: evito una delle prime tre condizioni.
  - *Diretta*: evito la quarta.
- **Evitamento** (avoidance): scelte dinamiche in base allo stato attuale del sistema.
  - *Divieto di allocazione*: non do risorse a un processo se queste risorse potrebbero portare a un deadlock.
  - *Divieto di iniziazione:* non faccio partire un processo che potenzialmente potrebbe portare al deadlock.

> Le strategie di **prevenzione** sono molto **conservative e sicure.**
>
> Le strategie di **evitamento** sono più liberali e variano caso per caso.

- **Rilevazione**: rilevarne la presenza e fare qualcosa per recuperare.
  - *Passo 1*: ho un protocollo che rieseguo continuamente, *che mi costa in termini di risorse*, per rilevare un deadlock.
  - *Passo 2:* O elimino tutti i processi in deadlock oppure cerco di ripristinare il sistema ad uno stadio precedente.

Riassunto:

| Approach | Resource Allocation Policy | Different Schemes | Major Advantages | Major Disadvantages |
|---|---|---|---|---|
| Prevention | Conservative; undercommits resources | Requesting all resources at once | •Works well for processes that perform a single burst of activity<br>•No preemption necessary | •Inefficient<br>•Delays process initiation<br>•Future resource requirements must be known by processes |
| | | Preemption | •Convenient when applied to resources whose state can be saved and restored easily | •Preempts more often than necessary |
| | | Resource ordering | •Feasible to enforce via compile-time checks<br>•Needs no run-time computation since problem is solved in system design | •Disallows incremental resource requests |
| Avoidance | Midway between that of detection and prevention | Manipulate to find at least one safe path | •No preemption necessary | •Future resource requirements must be known by OS<br>•Processes can be blocked for long periods |
| Detection | Very liberal; requested resources are granted where possible | Invoke periodically to test for deadlock | •Never delays process initiation<br>•Facilitates online handling | •Inherent preemption losses |