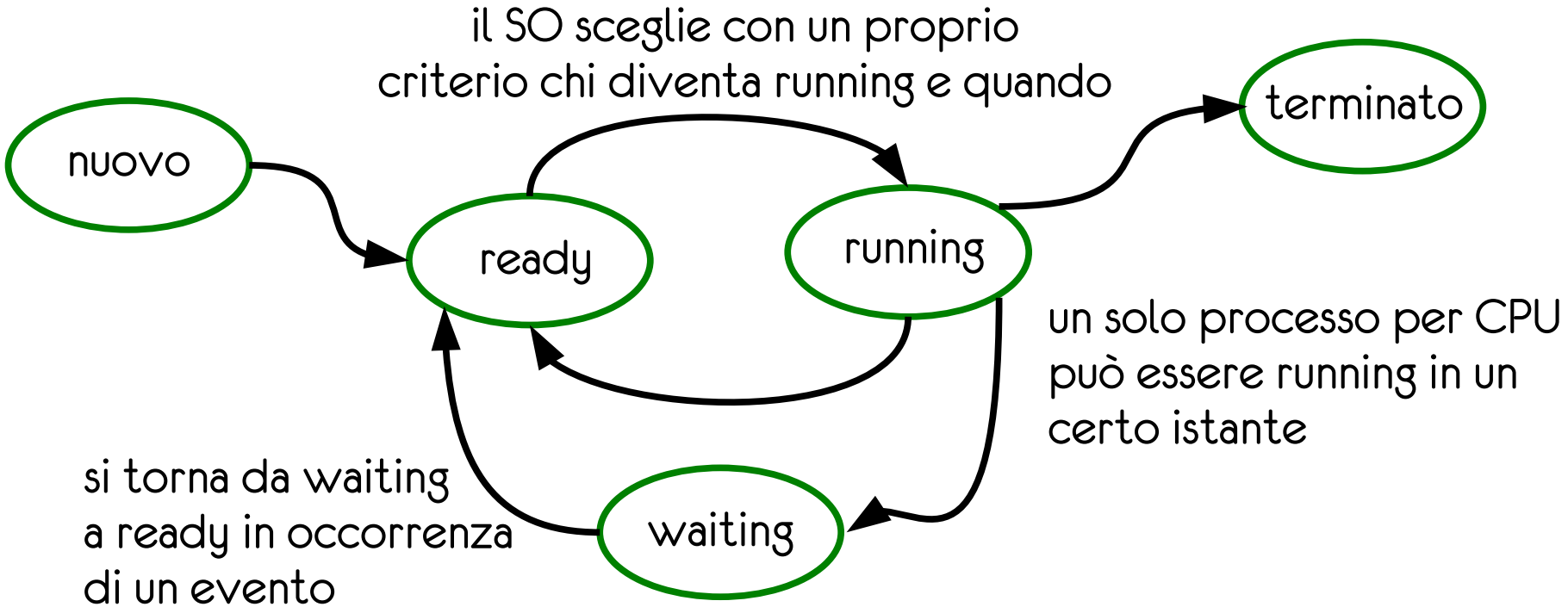
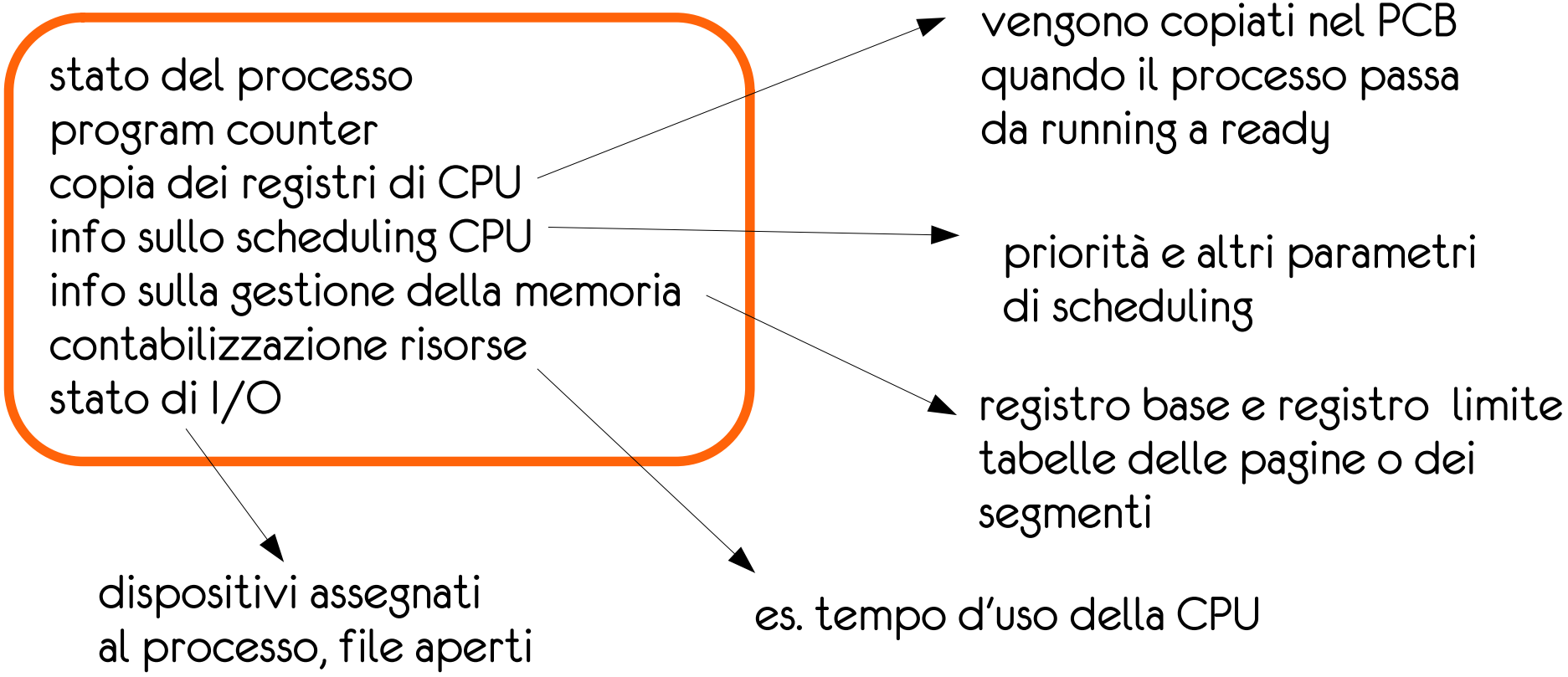
Diagramma di transizione

- Diagramma di transizione degli stati di un processo



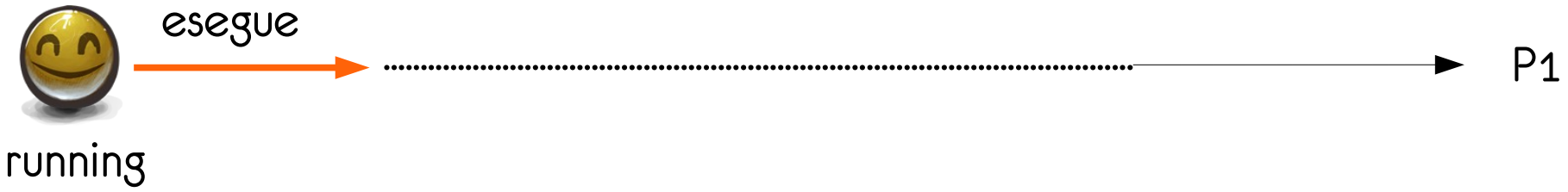
Processi e PCB

- In un SO un processo è rappresentato da un PCB, Process Control Block, contenente queste informazioni:



Es. commutazione della CPU

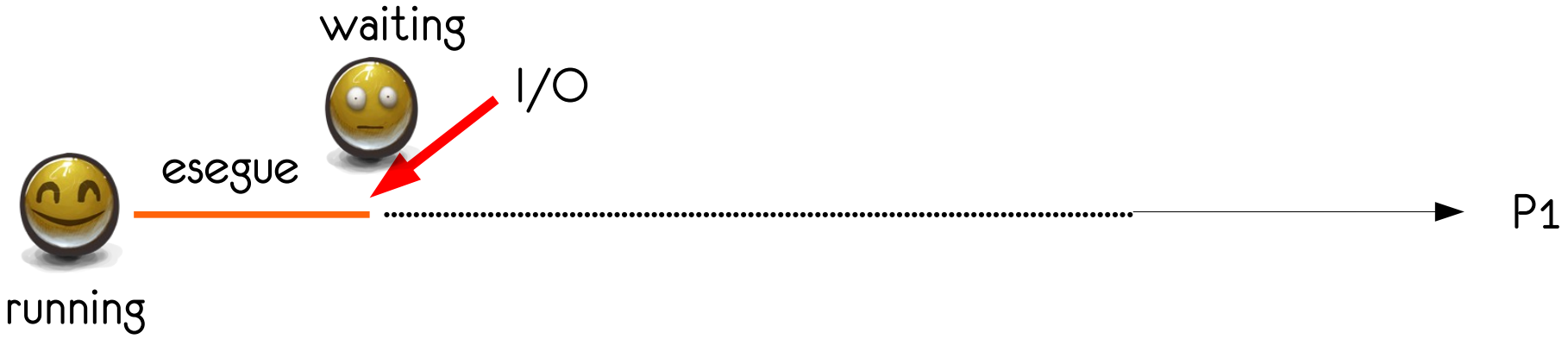
- vediamo cosa succede quando il SO toglie la CPU a un processo running (P1) per passarla a un processo ready (P2)



linee temporali

Es. commutazione della CPU

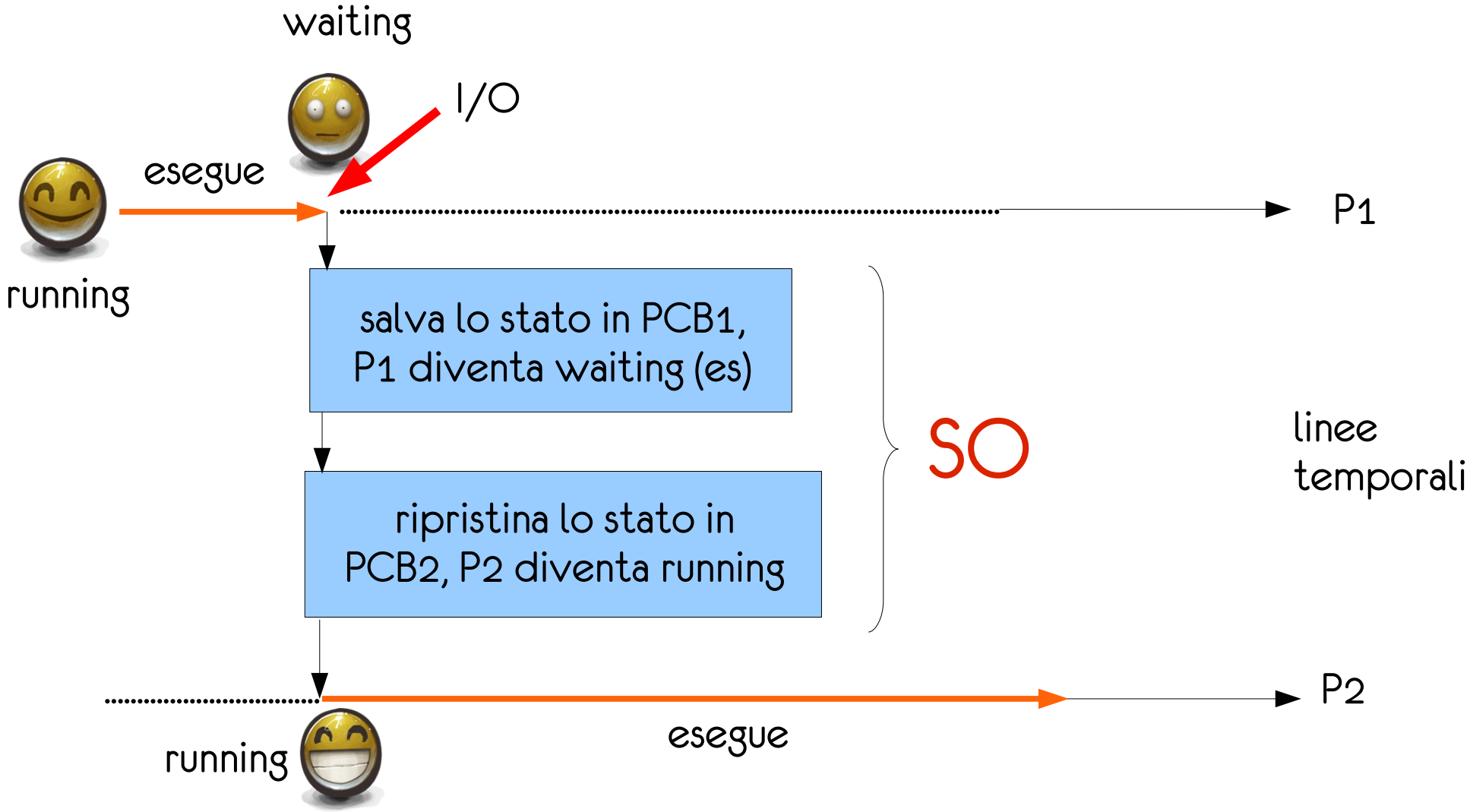
- vediamo cosa succede quando il SO toglie la CPU a un processo running (P1) per passarla a un processo ready (P2)



linee temporali

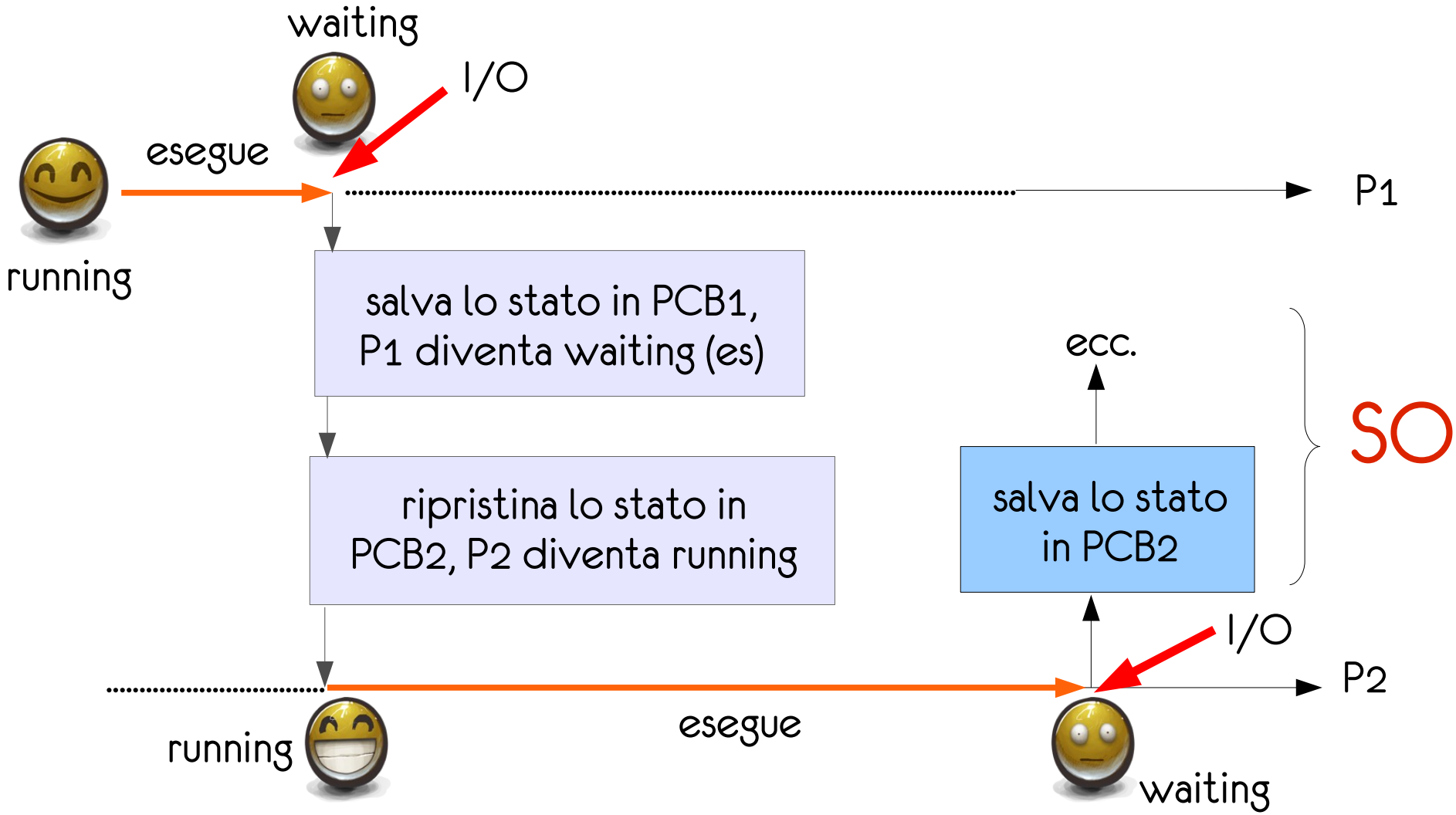
Es. commutazione della CPU

- vediamo cosa succede quando il SO toglie la CPU a un processo running (P1) per passarla a un processo ready (P2)



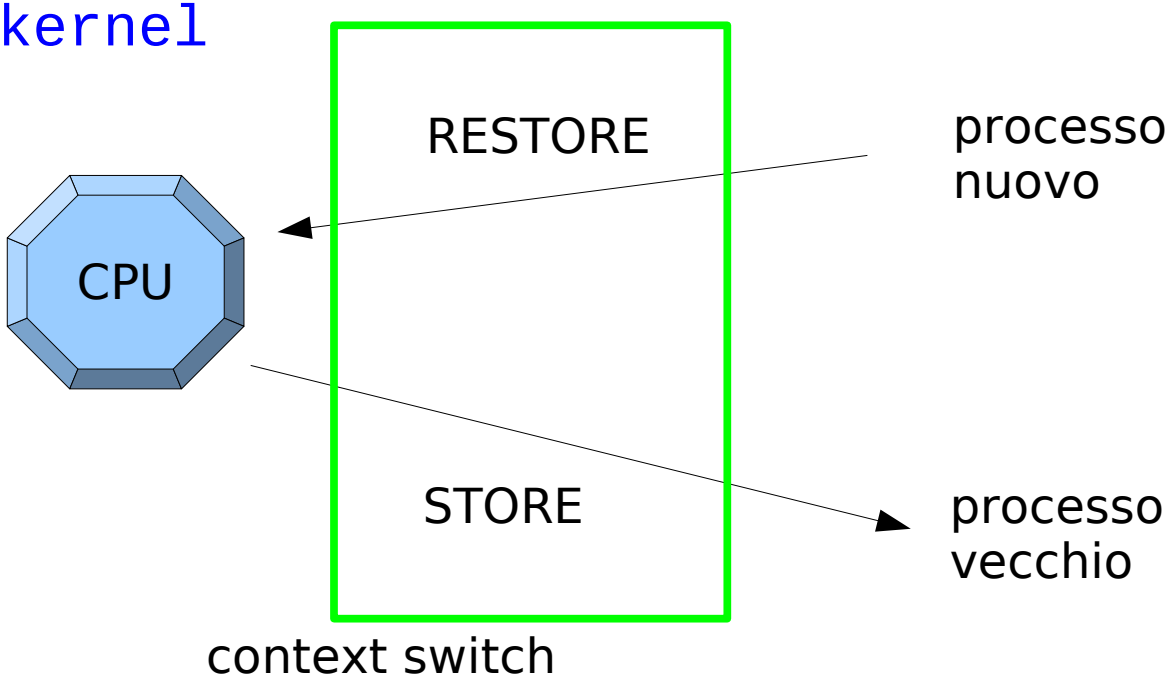
Es. commutazione della CPU

- vediamo cosa succede quando il SO toglie la CPU a un processo running (P1) per passarla a un processo ready (P2)



Context switch

- Il passaggio di un processo da **running** a **waiting** e il concomitante passaggio di un altro processo da **ready** a **running** richiede un **context switch** (cambio del contesto di esecuzione)
- contesto = contenuto dei registri della CPU e del program counter
- il context switch avviene esclusivamente in modalità kernel



Context switch

- ogni volta che un'interruzione causa la riassegnazione della CPU viene effettuato un context switch (cambiamento di contesto):
  - lo stato corrente della CPU viene salvato nel PCB del processo sospeso (se viene sospeso e non terminato)
  - lo stato relativo al processo scelto dallo scheduler a breve termine viene caricato
  - lo stato comprende i valori dei registri
- il tempo per un context switch dipende dall'architettura:
  alcune architetture prevedono diversi set di registri, il context switch indica semplicemente quale set va usato

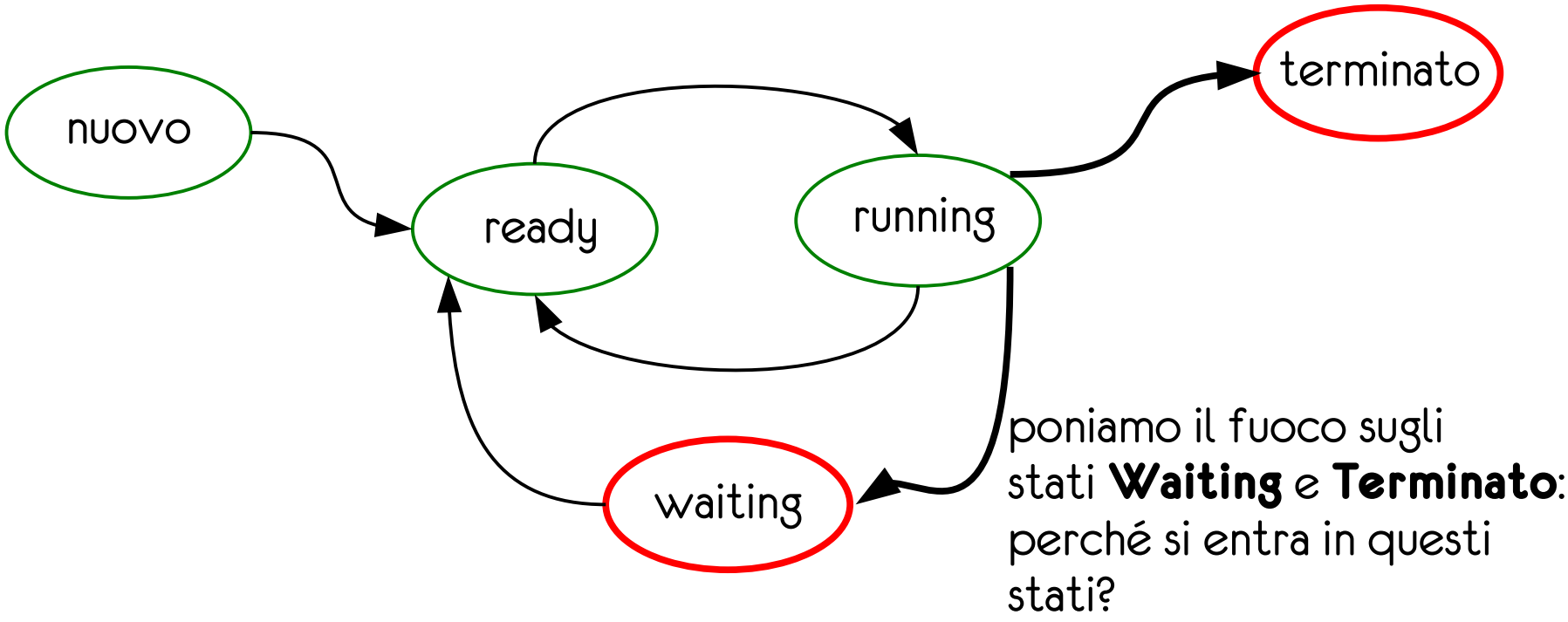
Quanto tempo dura un C.S.?

```
baroglio@esmeralda:~/BACKUP/LAB_SO/Slide$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0      0 994084 157708 572248    0    0   299   205  295  815 12  2 78  7
baroglio@esmeralda:~/BACKUP/LAB_SO/Slide$
```

- 815 cs / sec ≈ 1 c.s. ogni 0.001 ≈ 1 ogni millisecondo (10^{-3} sec)
- un c.s. dura alcuni nanosecondi (10^{-9} sec)
- molti più c.s. (815) che interrupt (295) ⇨ c.s. non sono causati esclusivamente da operazioni di I/O

Diagramma di transizione

- Diagramma di transizione degli stati di un processo



Passaggio a waiting/terminato

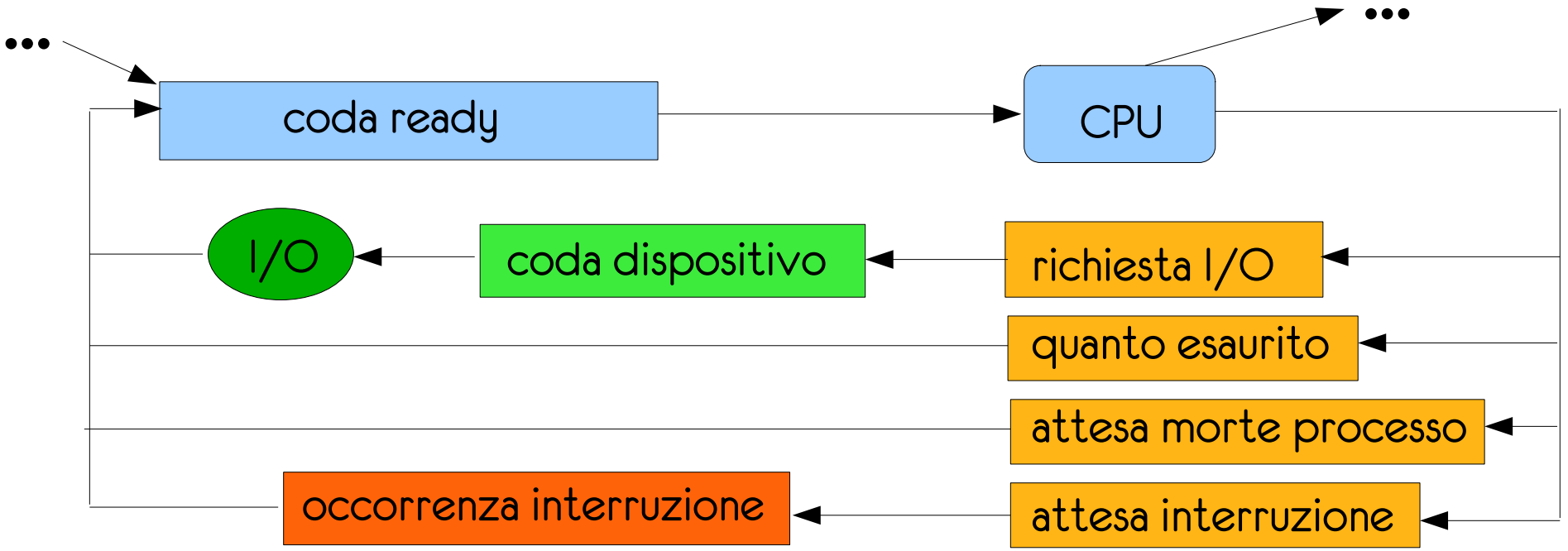
- **waiting**:
  - esecuzione di un comando di I/O
  - sospensione volontaria per un lasso di tempo
  - sospensione volontaria in attesa di un evento (es. morte di un altro processo)
  - sincronizzazione (es. attesa di un messaggio, attesa legata ad altri strumenti di sincronizzazione come i semafori)

Passaggio a waiting/terminato

- **waiting**:
  - esecuzione di un comando di I/O
  - sospensione volontaria per un lasso di tempo
  - sospensione volontaria in attesa di un evento (es. morte di un altro processo)
  - sincronizzazione (es. attesa di un messaggio, attesa legata ad altri strumenti di sincronizzazione come i semafori)
- **terminato**
  - exit
  - abort
  - kill da parte di altri processi
  - interruzione da parte dell'utente

Code di processi

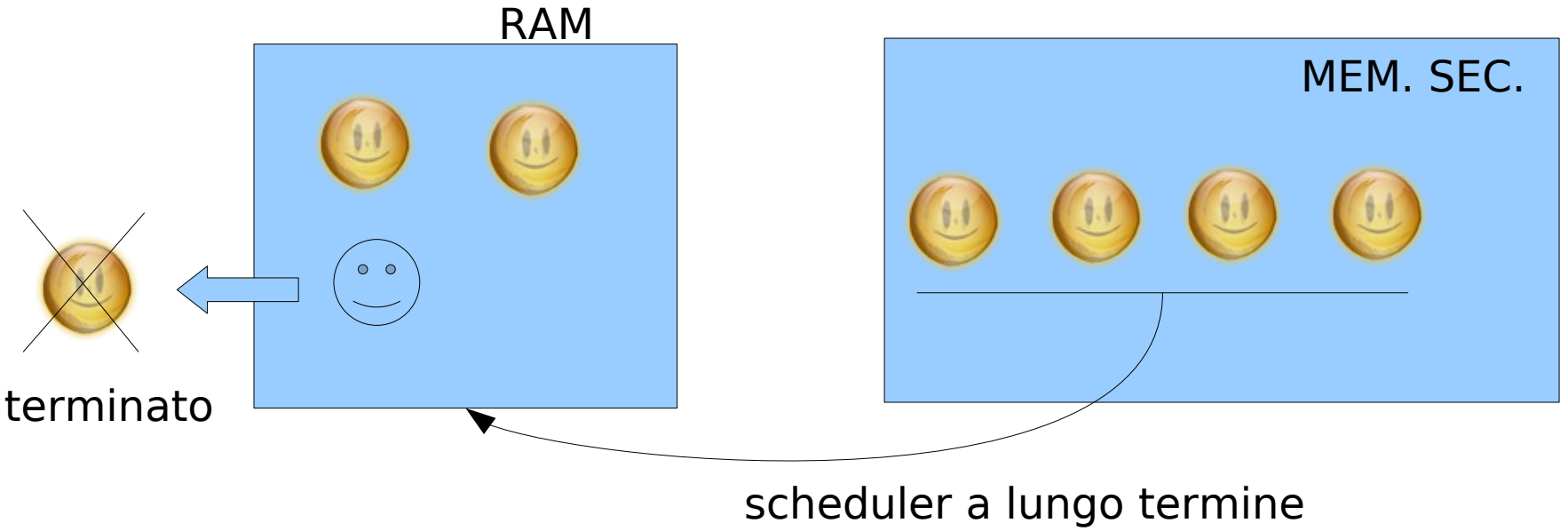
- PCB mantenuti in strutture dati (code):
  - coda ready: contiene tutti i PCB dei processi caricati in memoria e pronti all'esecuzione
  - coda di dispositivo (ne esistono diverse): contiene tutti i PCB dei processi in attesa che si completi un'operazione da loro richiesta su quel dispositivo



Scheduling

- Scheduling a lungo termine:
  presente in **sistemi batch**, in cui la coda dei processi da eseguire era conservata in memoria secondaria. Si attiva quando un processo caricato in memoria principale **termina**, scegliendone uno in memoria secondaria e caricandolo in memoria principale
- Scheduling a medio termine:
  quando il **grado di multiprogrammazione** è troppo alto, non tutti I processi possono essere contenuti in RAM, a turno alcuni vengono spostati in memoria secondaria
- Scheduling a breve termine:
  politica di **avvicendamento alla CPU** dei processi caricati in RAM

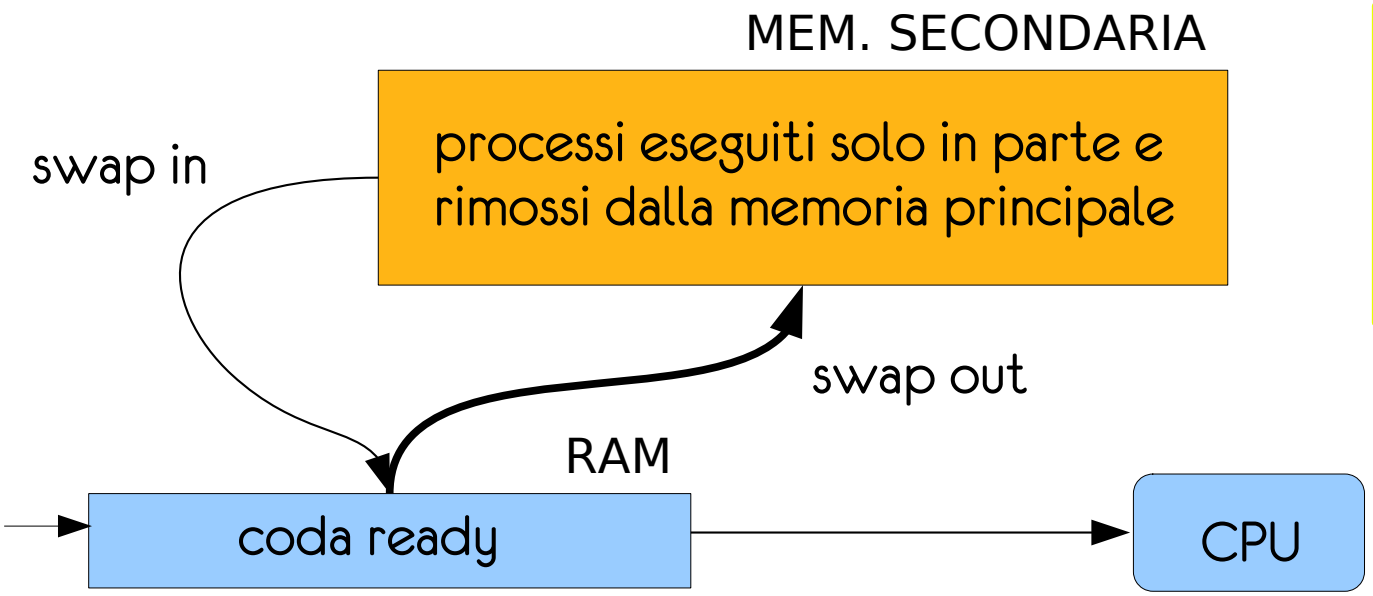
Scheduling a lungo termine



- si limita a sotituire processi terminati, usando uno **scheduler a lungo termine,** si attiva al termine di un processo e carica dalla memoria secondaria un altro processo
- **criterio**:
  mantenere un buon equilibrio fra processi CPU-bound e (che in prevalenza fanno computazione) processi IO-bound (che in prevalenza fanno I/O)

PCB in memoria secondaria

- Un sistema può avere in esecuzione più processi di quanti ne possano coesistere nella memoria principale ...
- **soluzione**: una parte dei processi va trasferita in memoria secondaria fino a quando non sarà possibile eseguirli (**swapping** o avvicendamento dei processi in memoria o scheduling a medio termine)



NB. lo scheduling a lungo termine non prevede l'analogo dello swap out

scheduling della cpu

capitolo 5 del libro (VII ed.), fino a 5.4

Scheduler della CPU

- ... o scheduler a breve termine: seleziona dalla coda ready il processo a cui assegnare la CPU
- NB: la coda ready contiene sempre dei PCB ma può essere implementata in modi diversi, a seconda dell'algoritmo di scheduling usato; non necessariamente è una coda di tipo FIFO
- casi in cui occorre scegliere:

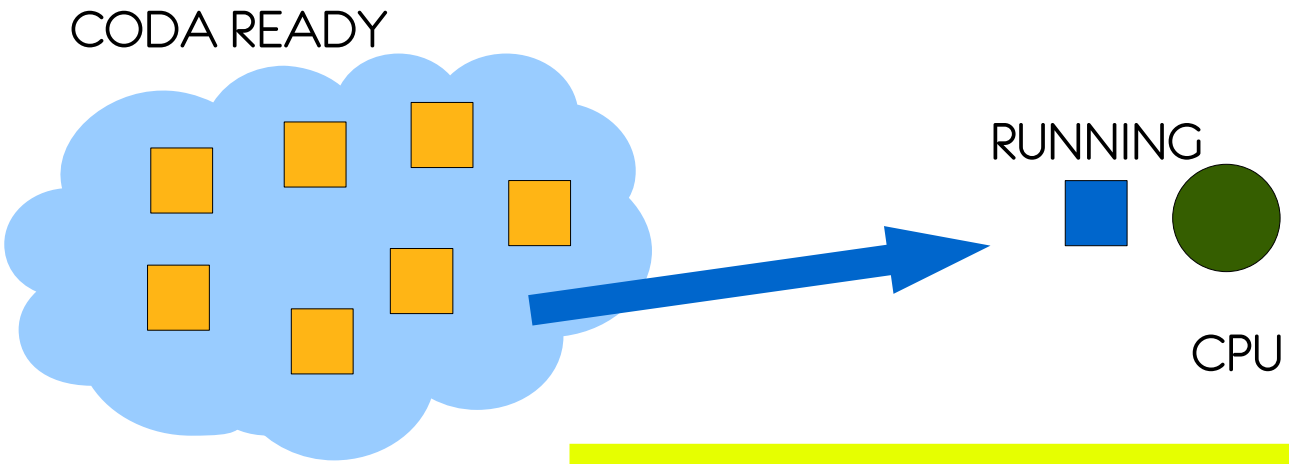
  1) il processo running diventa waiting

  2) il processo running viene riportato a ready (interrupt)

  3) un processo waiting diventa ready

  4) il processo running termina

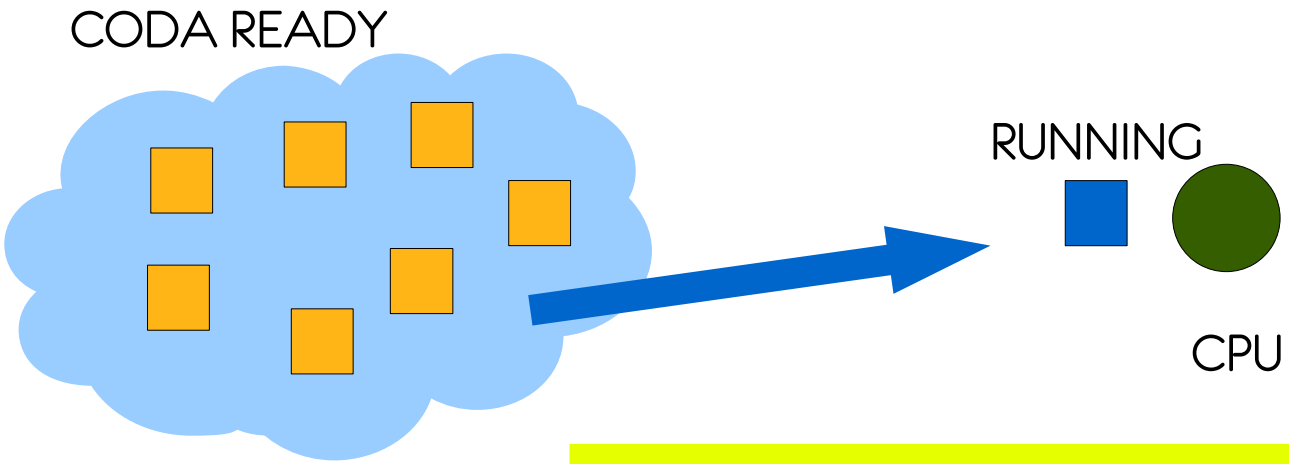
Scheduler della CPU



Lo **scheduler** sceglie un processo ready

Il **DISPATCHER** effettua il cambio di contesto, effettua il posizionamento alla giusta istruzione, passa nella modalità di esecuzione giusta

Scheduler della CPU



Lo **scheduler** sceglie un processo ready

Il **DISPATCHER** effettua il cambio di contesto, effettua il posizionamento alla giusta istruzione, passa nella modalità di esecuzione giusta

condizione che fa scattare lo scheduler
⊕ criterio di selezione ⇒ politica

Algoritmi di scheduling

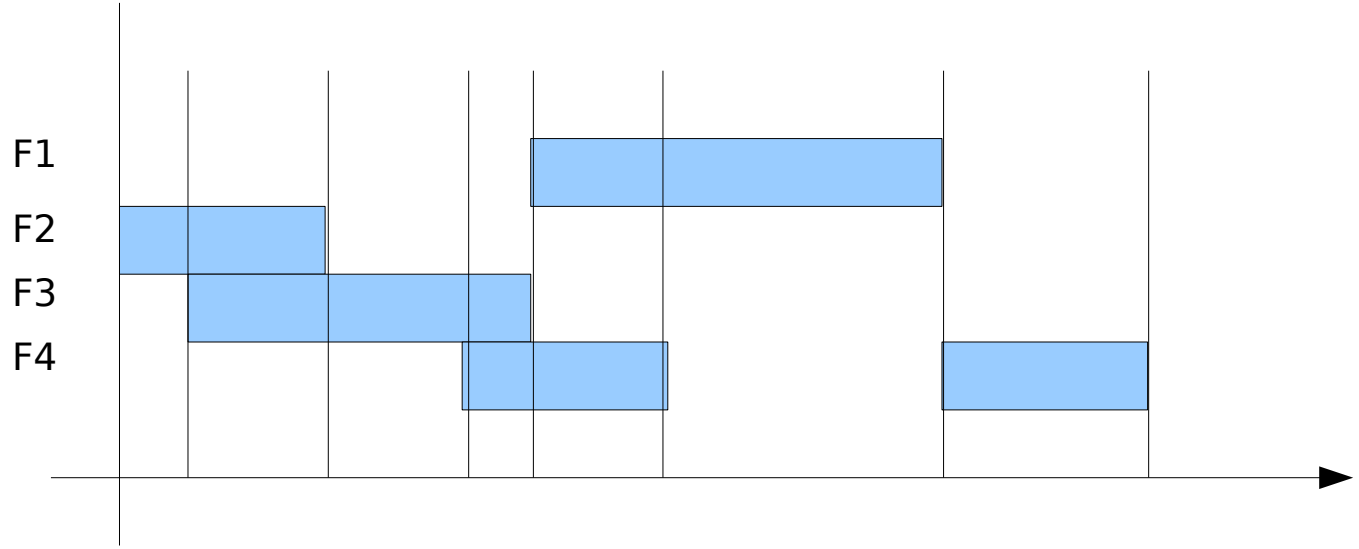
- first-come-first-served (FCFS)
- shortest jobs first (SJF)
- a priorità
- round-robin (RR)
- a code multilivello (multilevel queue scheduling)
- a code multilivello con feedback (multilevel feedback queue scheduling)

Criteri di scheduling

- mantenere la CPU il più attiva possibile
- throughput (produttività): numero di processi completati nell'unità di tempo
- turnaround time: tempo di completamento di un processo, è la somma non solo del tempo di esecuzione e dei vari I/O ma anche di tutti i tempi di attesa legati allo scheduling (attesa di ingresso in main memory, attesa nella coda ready)
- **tempo di attesa**: **visto che l'algo. di scheduling della CPU non influisce sui tempi legati all'esecuzione del processo si può considerare anche il solo tempo di attesa trascorso in ready queue**
- tempo di risposta: nei sistemi interattivi è importante ridurre il tempo che intercorre fra la sottomissione di una richiesta (da parte dell'utente) e la risposta

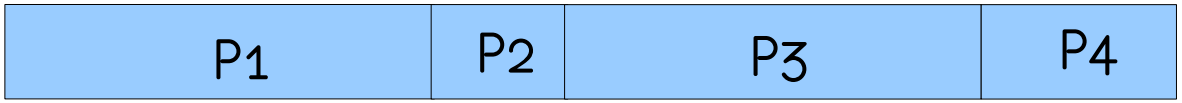
Diagrammi di Gantt

- Ideati nel 1917 come supporto per la gestione di progetti
- Asse orizzontale: tempo
- Asse verticale: progetti oppure fasi di un progetto
- Delle barre orizzontali indicano quando (sia in senso assoluto che in relazioni agli altri progetti/fasi) e per quanto tempo durano le diverse fasi



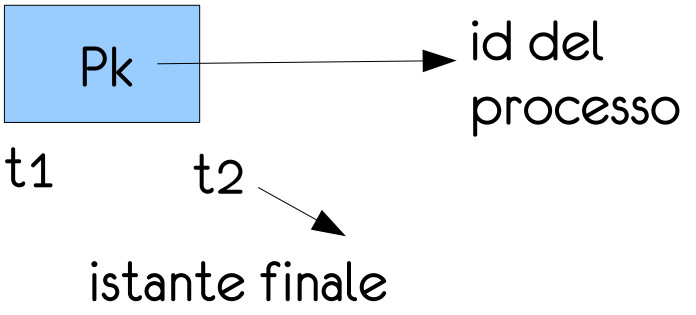
Diagrammi di Gantt

- Rappresenteremo il tempo di occupazione della CPU da parte dei diversi processi
- Una CPU può essere occupata da un solo processo per volta, quindi il Gantt non può contenere sovrapposizioni di fasi
- Si può comprimere a una sola barra contenente blocchetti, ognuno dei quali corrisponde all'esecuzione di un processo o di una sua parte



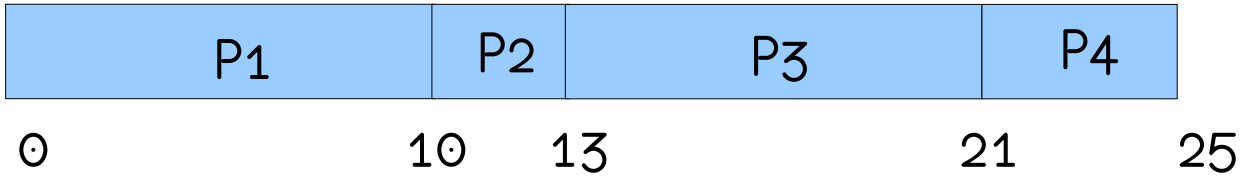
Premessa: diagrammi di Gantt

In un diagramma di Gantt si rappresenta l'occupazione della CPU con un rettangolo così fatto:



Es. dati i processi e relativi CPU burst in tabella, supponendo che la CPU venga allocata nell'ordine con cui sono dati i processi avremo il seguente diagramma

P	D
P1	10
P2	3
P3	8
P4	4



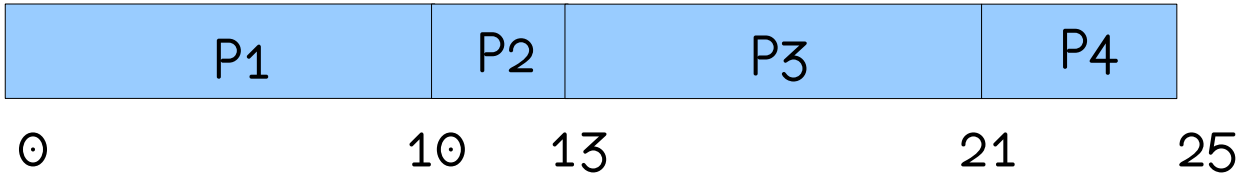
esempio di diagramma di Gantt

P: processo

D: durata del prossimo CPU burst, sequenza contigua di operazioni di CPU fra un I/O e l'altro

First-come-first-served

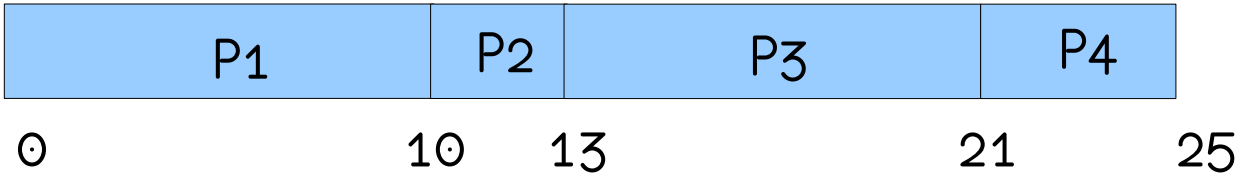
- **FCFS**: PCB organizzati in una coda FIFO, non è preemptive, il primo processo arrivato è il primo ad avere la CPU, la mantiene per un intero CPU burst
- il tempo medio di attesa è abbastanza lungo, es:



- t.m.a. = ???

First-come-first-served

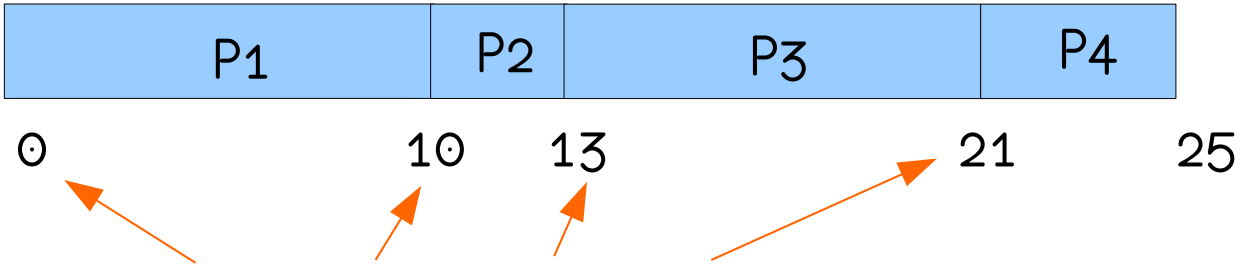
- **FCFS**: PCB organizzati in una coda FIFO, non è preemptive, il primo processo arrivato è il primo ad avere la CPU, la mantiene per un intero CPU burst
- il tempo medio di attesa è abbastanza lungo, es:



- tma = (ta_P1 + ta_P2 + ... + ta_Pn) / num_proc

First-come-first-served

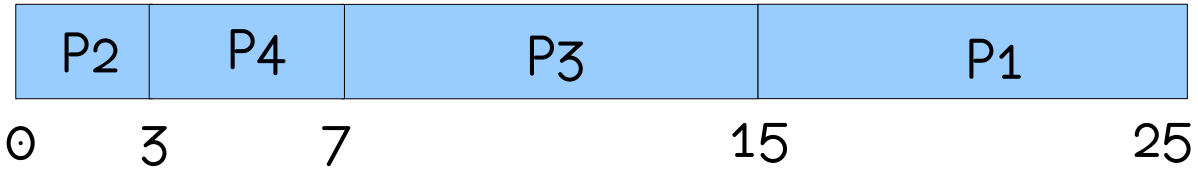
- **FCFS**: PCB organizzati in una coda FIFO, non è preemptive, il primo processo arrivato è il primo ad avere la CPU, la mantiene per un intero CPU burst
- il tempo medio di attesa è abbastanza lungo, es:



- tma = (0 + 10 + 13 + 21) / 4 =
  = 44/4 =
  = 11 millisec

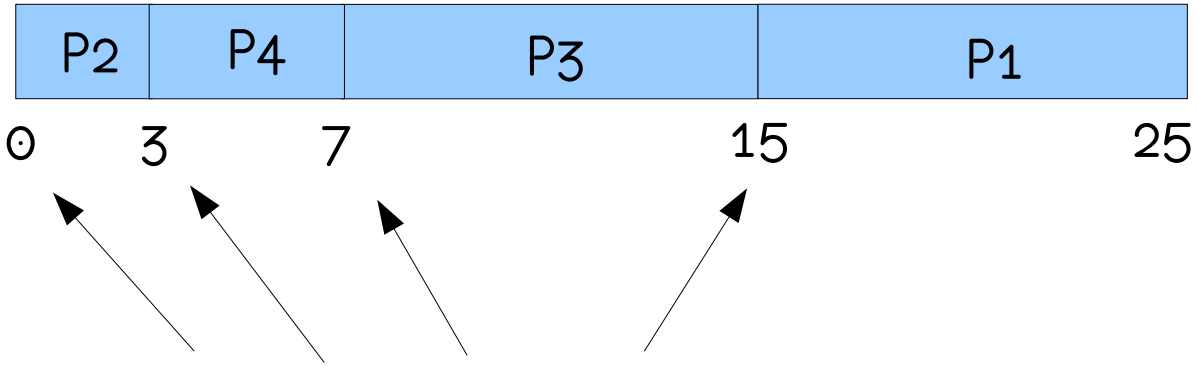
First-come-first-served

- Se i PCB fossero giunti in un altro ordine il tma sarebbe cambiato? Es.



First-come-first-served

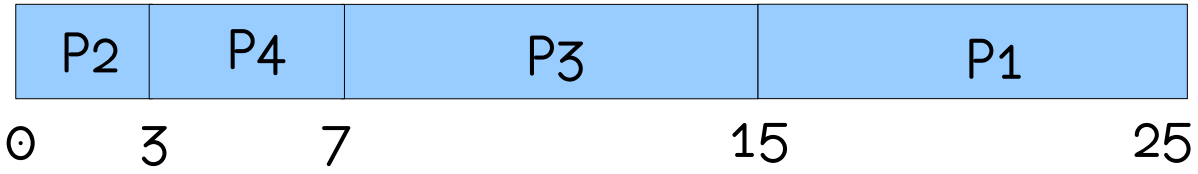
- **SÌ**:



- tma = (0 + 3 + 7 + 15) / 4 = 25/4 = 6,25 millisec

First-come-first-served

- **SÌ**:



- tma = (0 + 3 + 7 + 15) / 4 = 25/4 = 6,25 millisec

- **NB**: FCFS non adeguato a sistemi in time-sharing in cui tutti gli utenti devono disporre della CPU a intervalli regolari

First-come-first-served

- **effetto convoglio**: supponiamo di avere n processi I/O-bound e un processo CPU-bound in coda ready:

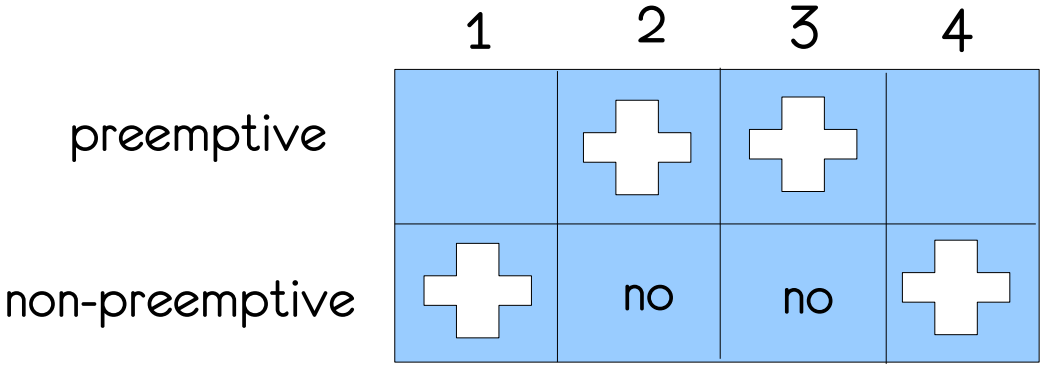
- il processo CPU-bound occupa la CPU per un tempo lungo
- quando fa, per es., I/O la cede
- i processi successivi hanno CPU-burst molto brevi, si alternano in fretta e tornano ad accodarsi

Prelazione di una risorsa

- Meccanismo generale per il quale il SO può togliere una risorsa riservata per un processo anche se:
  - il processo la sta utilizzando
  - il processo utilizzerà la risorsa in futuro
- Nello scheduling a breve termine la risorsa contesa è la CPU

Prelazione

1) il processo running diventa waiting
2) il processo running viene riportato a ready (interrupt)
3) un processo waiting diventa ready
4) il processo running termina



Lo scheduling è **PREEMPTIVE** se interviene in almeno uno dei casi 2 e 3, può occorrere anche in 1 o 4

Lo scheduling è **NON-PREEMPTIVE** se interviene solo nei casi 1 o 4

→ La CPU non viene mai tolta al processo running

Prelazione, supporto HW ...

- La prelazione non è sempre possibile, occorre un'architettura che la consenta.
- In particolare occorrono dei **timer**, se non ci sono l'unico tipo di scheduling che si può avere è non-preemptive (anche detto cooperativo)
- Es. Apple ha introdotto la prelazione nei suoi sistemi operativi con Mac OS X; Windows l’ha adottata con Windows 95

Prelazione, supporto HW … e rischi

- La prelazione richiede l'introduzione di meccanismi di **sincronizzazione** per evitare inconsistenze.
- Se due processi **condividono dati** e uno dei due li sta aggiornando quando gli viene tolta la CPU (a favore dell'altro), quest'ultimo troverà dati inconsistenti.
- Affronteremo il problema più avanti

Shortest job first

- SJF è **ottimale** nel minimizzare il tempo medio di attesa, **seleziona sempre il processo avente CPU burst successivo di durata minima** (shortest next CPU burst)
- **Problema**: la durata dei CPU burst non è nota a priori!
- **Soluzione**: prevedere la durata sulla base dei burst precedenti, combinati con una media esponenziale:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

- espansa la formula diventa:

$$\tau_{n+1} = \alpha t_n + \alpha(1-\alpha) t_{n-1} + \ldots + \alpha(1-\alpha)^j t_{n-j} + \ldots + (1-\alpha)^{n+1} \tau_0$$

- α è un numero compreso fra 0 e 1, quindi gli addendi più vecchi hanno un influsso via via inferiore.
- Maggiore è α maggiore è il peso dato alla storia recente

Shortest job first

- **SJF può essere preemptive o meno**
- **comportamento preeemptive** (anche detto “shortest remaining time left”): quando un nuovo processo diventa ready, lo scheduler controlla se il suo burst è inferiore a quanto rimane del burst del processo running:
  - SÌ: il nuovo processo diventa running, si ha commutazione di contesto
  - NO: il nuovo processo viene messo in coda ready
- **comportamento non-preemptive**: il processo viene messo in coda ready, eventualmente in prima posizione

Priorità e Starvation

- **Problema**: tutti gli algoritmi a priorità sono soggetti a **starvation:**

  un processo potrebbe non ottenere mai la CPU perché continuano a passargli davanti nuovi processi a priorità maggiore

- **Soluzione**: **aging**. La priorità viene alzata con il trascorrere del tempo