CPU SCHEDULING

Concetti di base: multiprogrammazione

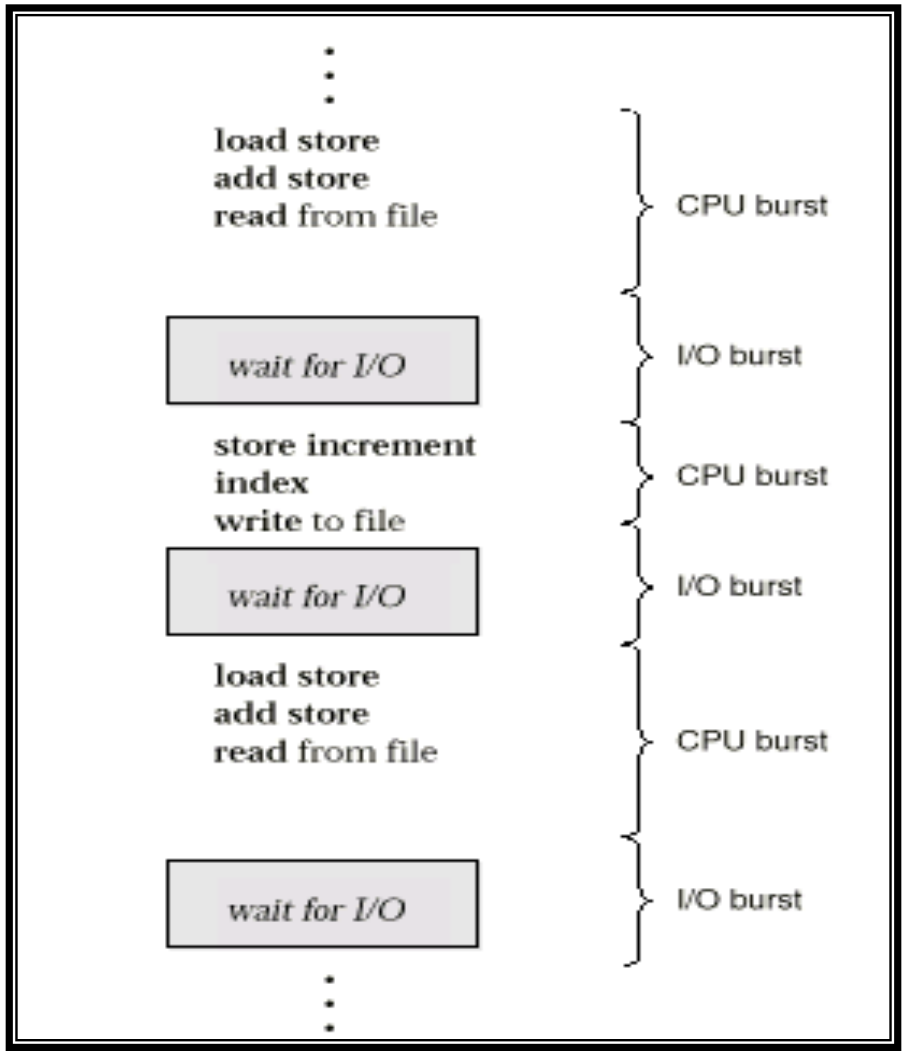
Da ora in poi sul libro di testo…

Multiprogrammazione = Multitasking

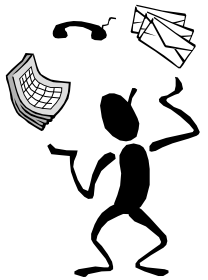
… e cioe' ***piu' di un processo*** alla volta pronto ad essere eseguito viene tenuto in memoria centrale e ***condivide il tempo di CPU*** con tutti gli altri

Concetti di base: CPU burst e I/O burst

Il comportamento tipico di un processo e' di ***alternare*** una sequenza di ***continuo uso di CPU*** (CPU burst) con una sequenza di ***operazioni di I/O*** (I/O burst)



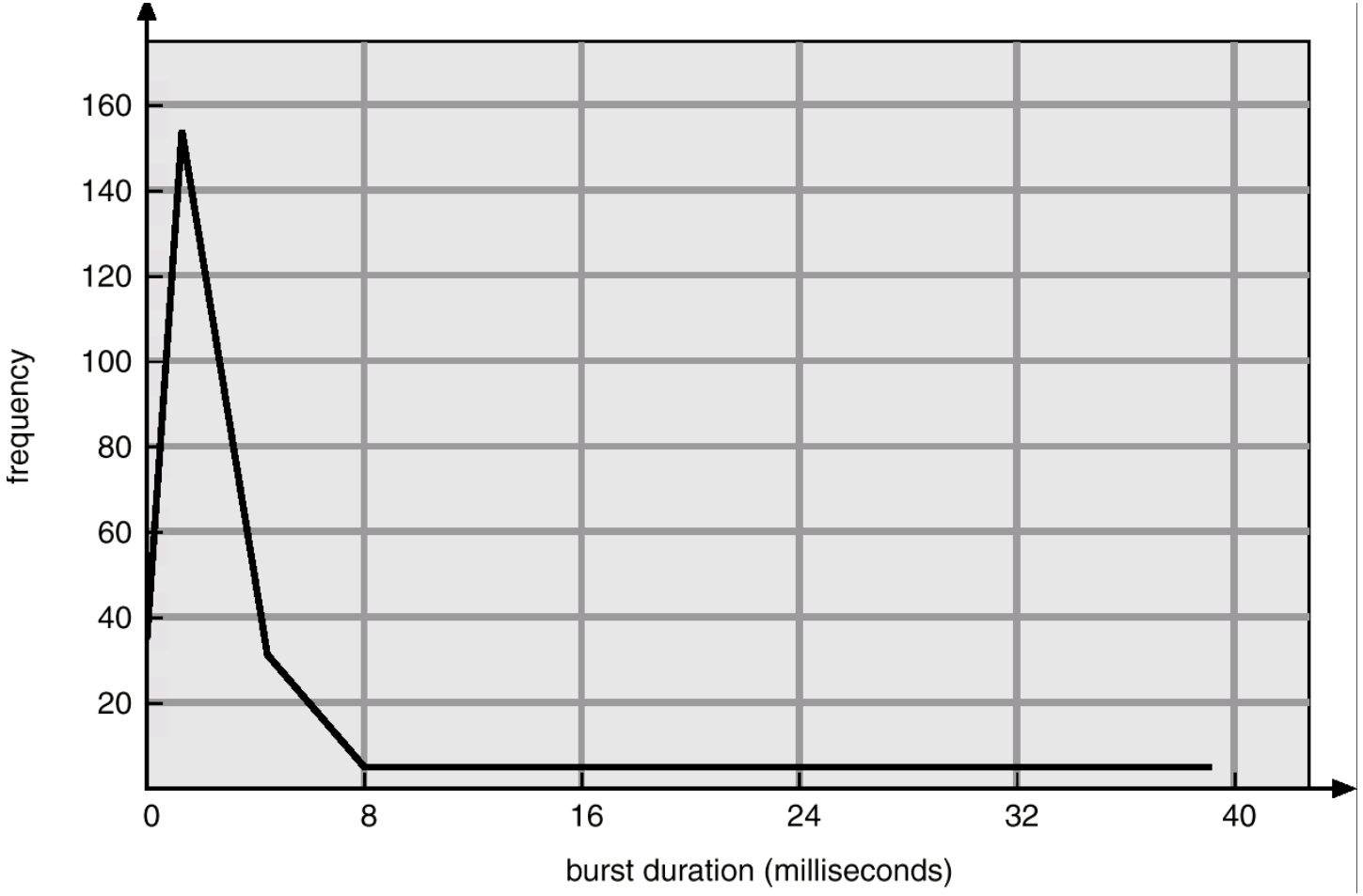
Quindi…



… la possibilita' , durante l' ***I/O burst*** di un processo, di mandare in esecuzione qualche altro processo (***multiprogrammazione***) porta ad una piu' alta utilizzazione della ***preziosa risorsa CPU***

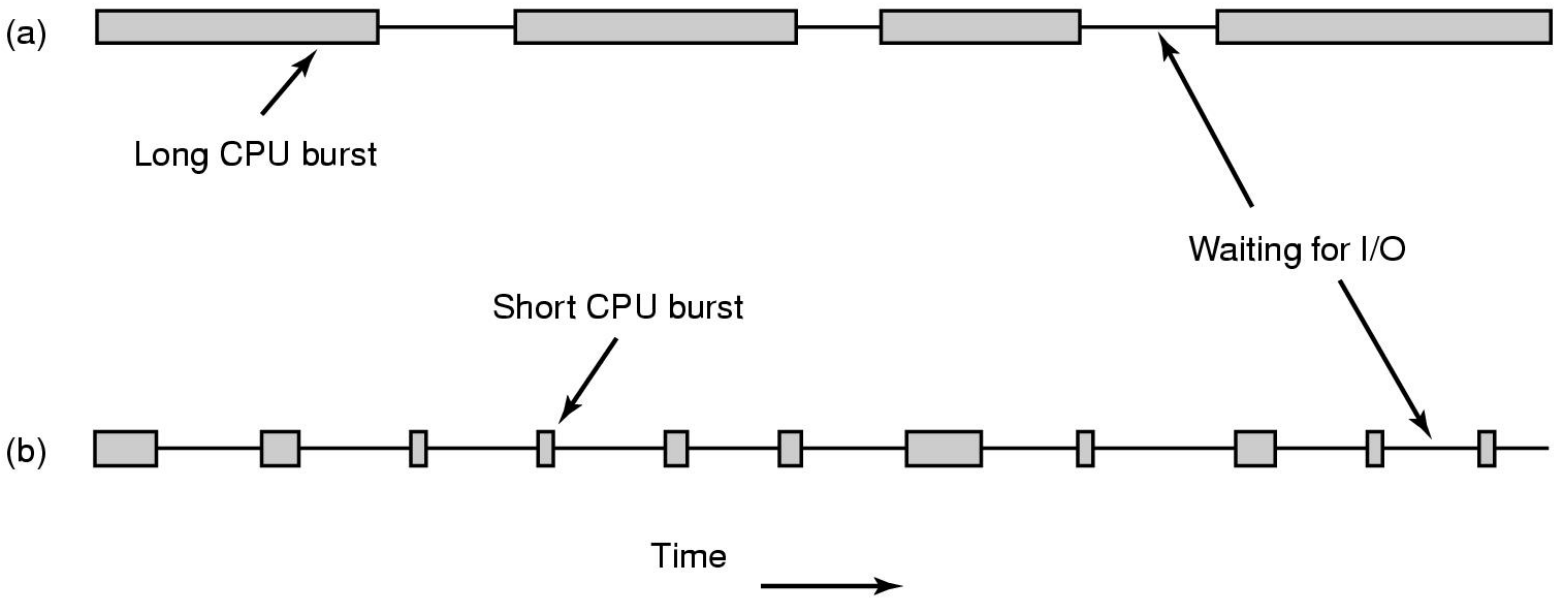
L' andamento dei cicli CPU-I/O burst influenza le ***scelte di gestione dei processi***, effettuate dallo ***scheduler***

Quanto dura un CPU burst?!



Reale andamento ma fittizi valori di tempo!

Quanto dura un CPU burst?!



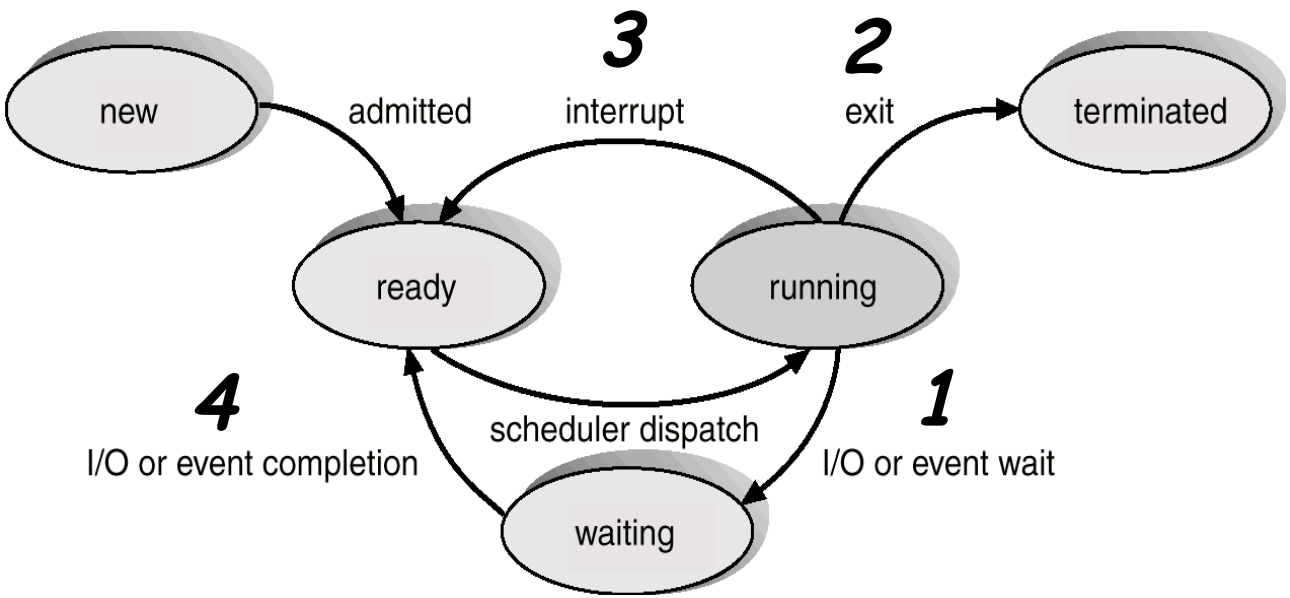
a) Un processo CPU-bound
b) Un processo I/O bound

CPU Scheduler: cosa fa

Quando necessario seleziona, tra i processi in memoria che sono pronti (cioe' nella ***ready queue***) quello che deve ***andare in esecuzione***

Quando necessario?

1. Un processo va da running a waiting
2. Un processo in esecuzione termina
3. Un processo va da running a ready
4. *Un processo va da waiting a ready (?!)*



Dispatcher

- Lo ***scheduler*** sceglie il processo da eseguire

- Il ***dispatcher*** da' il controllo della CPU al processo scelto, e cioe':

    - context switching
    - salto alla locazione propria del programma utente scelto per farlo ripartire
    - switching a user mode

- ***Dispatch latency*** – tempo impiegato dal dispatcher per svolgere il suo compito

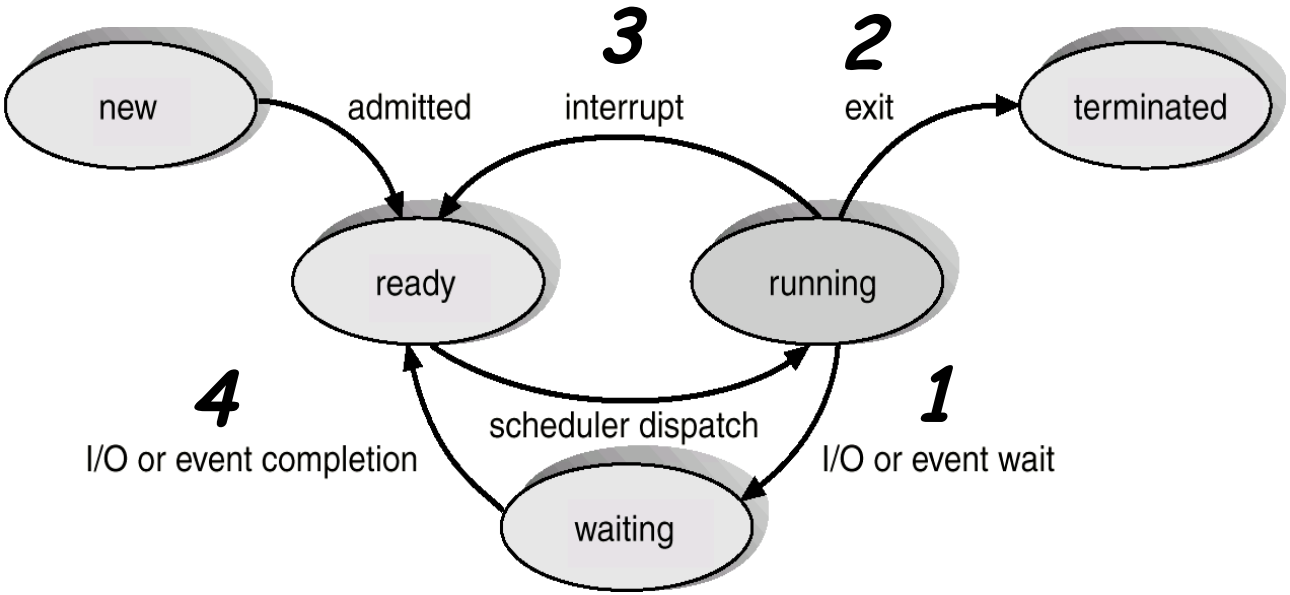
Concetti di base: preemption…

Qualsiasi ***azione*** di scheduling si dice ***preemptive*** se ha ***la possibilita' di interrompere il processo in esecuzione*** per sostituirlo con un altro

Un' azione di scheduling e' quindi ***non-preemptive*** quando ***attende che il processo*** in esecuzione ***abbandoni autonomamente la CPU*** comunque prima di agire

… e qualche considerazione

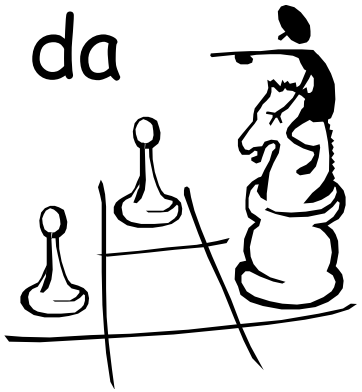
le azioni conseguenti a 1 e 2 sono *non-preemptive*
le azioni conseguenti a 3 e 4 sono *preemptive*



Inconsistenza di dati condivisi – a livello utente oppure, ancor piu' grave, su strutture dati di kernel

Da azione a politica di scheduling

Una ***politica*** di scheduling e' un algoritmo che ***regola ogni azione*** di scheduling e quindi ***decide quando*** e ***come*** selezionare un processo da eseguire



Separazione meccanismo-politica:

Meccanismo $\rightarrow$ scheduler

Politica $\rightarrow$ scheduling

Parametri di confronto tra politiche : a livello di OS

- ***Utilizzazione della CPU*** – percentuale di tempo in cui la CPU e' occupata a fare qualcosa

- ***Throughput*** – numero di processi che completano la loro esecuzione per unita' di tempo

Da massimizzare

Parametri di confronto tra politiche : a livello di processo

- ***Turnaround time*** – tempo necessario ad eseguire un processo

- ***Tempo di attesa*** – tempo totale di attesa di un processo nella ready queue ***(parametro che piu' direttamente e' influenzato dalla politica!)***

- ***Tempo di risposta*** – tempo che intercorre tra una richiesta e l'arrivo della prima risposta (non il completamento dell'output del processo)

Da minimizzare

… e qualche variazione

Potrebbe essere preferibile ottimizzare i ***valori estremi*** di un parametro piuttosto che i medi, per esempio:

invece di tendere a minimizzare il ***tempo di attesa*** medio si potrebbe puntare a ***minimizzare quello massimo***…

… oppure si potrebbe ***minimizzare la varianza*** di un dato parametro

Politiche che consideriamo

1. First Come First Served
2. Shortest Job First
3. Con priorita'
4. Round-Robin
5. Code multilivello (con feedback)

Parametro di confronto :
tempo di attesa medio

Orizzonte di osservazione :
1 CPU burst per processo

1. First-Come, First-Served (FCFS)…

I processi vengono schedulati nello stesso ordine con il quale arrivano nella ready queue

… un esempio…

Processo	Burst Time
P_1	24
P_2	3
P_3	3

- Tempo di attesa per processo:

  $P_1 = 0; P_2 = 24; P_3 = 27$

- Tempo di attesa medio: $(0 + 24 + 27)/3 = 17$

… ed una variazione

Supponiamo invece che i processi siano arrivati nel seguente ordine:

P_2 , P_3 , P_1

- Tempo di attesa per processo:

  $P_1 = 6$; $P_2 = 0$; $P_3 = 3$

- Tempo di attesa medio: $(6 + 0 + 3)/3 = 3$

Convoy effect: processi brevi dietro a processi lunghi

2. Shortest-Job-First (SJF)…

Schedula il processo che presenta il prossimo CPU burst piu' breve

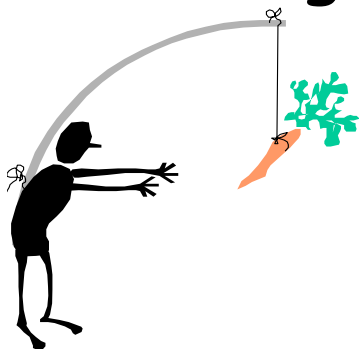
… in realta' si tratta di
shortest next CPU burst!

Solo nei batch si puo' stimare il tempo di un intero job

… due possibilita'

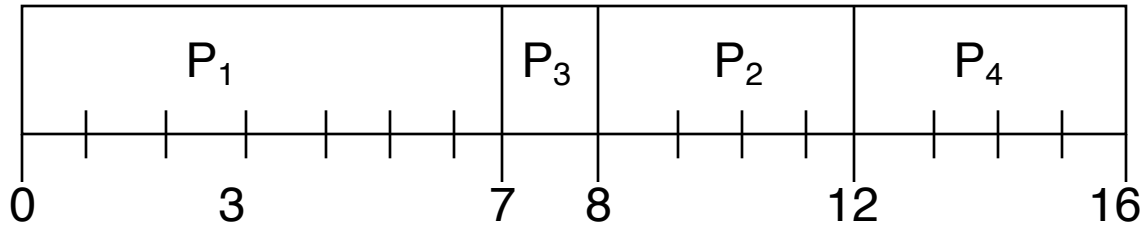
Non-preemptive – una volta che la CPU e' assegnata ad un processo non gliela si puo' togliere finche' non completa il suo CPU burst

Preemptive – se un nuovo processo arriva nella ready queue con ***un burst piu' corto del tempo rimanente*** del corrente processo allora lo si rimpiazza (Shortest-Remaining-Time-First, ***SRTF***)



Non-preemptive (esempio)

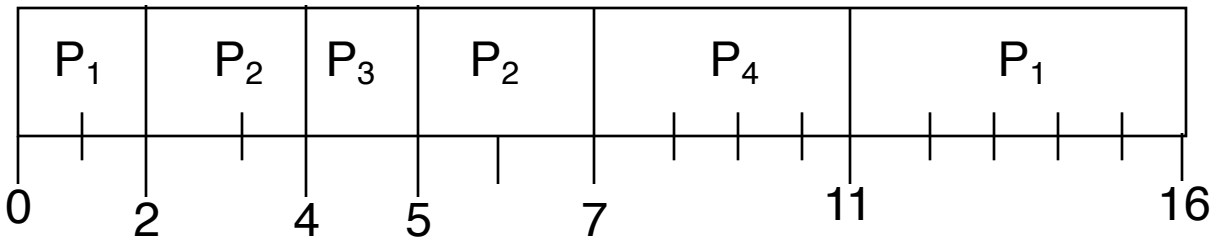
Processo	Istante di arrivo	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



Tempo medio di attesa: (0 + 6 + 3 + 7)/4 = 4

Preemptive (esempio)

Processo	Istante di arrivo	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



Tempo medio di attesa: (9 + 1 + 0 +2)/4 = 3

SJF e' ottimale, cioe' da' il minimo tempo medio di attesa per ogni dato insieme di processi!!!

MA...

… ma come determinare la lunghezza del prossimo CPU burst?!

Si puo' ***stimare*** tenendo traccia dei ***precedenti***

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

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

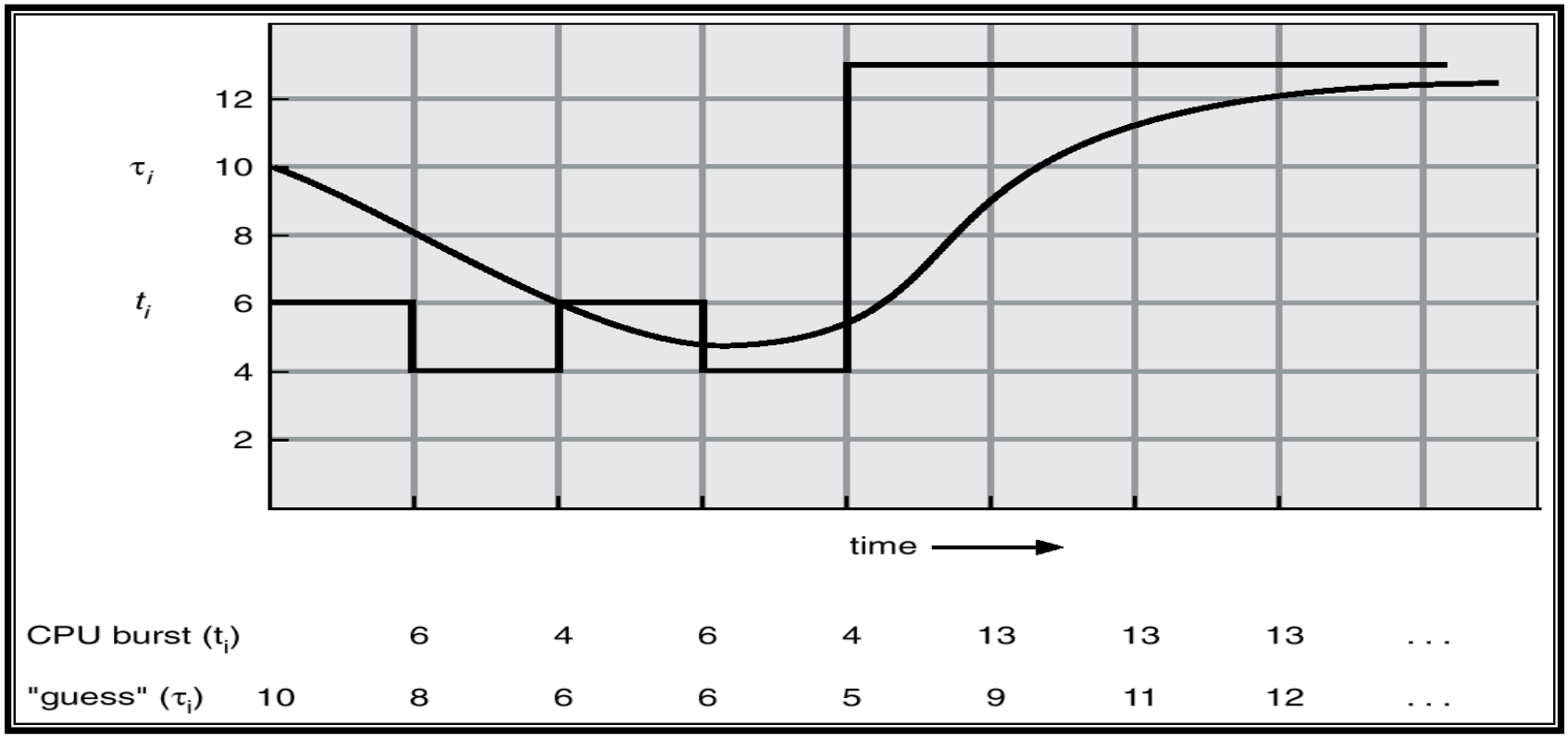
… α rappresenta il peso che vogliamo dare alla storia recente

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La storia recente non conta mai, e quindi il valore rimane costante
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Solo il valore dell' ultimo CPU burst conta
- $\alpha = 1/2$
  - $\tau_{n+1} = (t_n + \tau_n)/2$
  - Storia e ultimo valore hanno lo stesso peso

Questione aperta

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

Provare a espandere la formula per comprendere il significato del peso dato ai valori passati e il comportamento asintotico



3. Con priorita'

- Un ***numero*** (intero) ***di priorita'*** e' assegnato ad ogni processo

- La CPU e' allocata al processo con la piu' alta priorita' (di solito il processo con l'intero piu' piccolo e' quello a piu' alta priorita'! Es. UNIX)

… parametri che possono indurre il livello di priorita' di un processo…

limiti di tempo
requisiti di memoria
numero di file aperti
rapporto tra I/O e CPU burst

INTERNI

importanza del processo
tipologia di utente

ESTERNI

… e alcune considerazioni

SJF puo' essere considerata una politica dove la priorita' e' il tempo del prossimo CPU burst

Piu' in generale, una politica con priorita' puo' essere sia preemptive che non-preemptive

Un problema : la starvation

Processi a bassa priorita'
potrebbero non essere eseguiti mai!

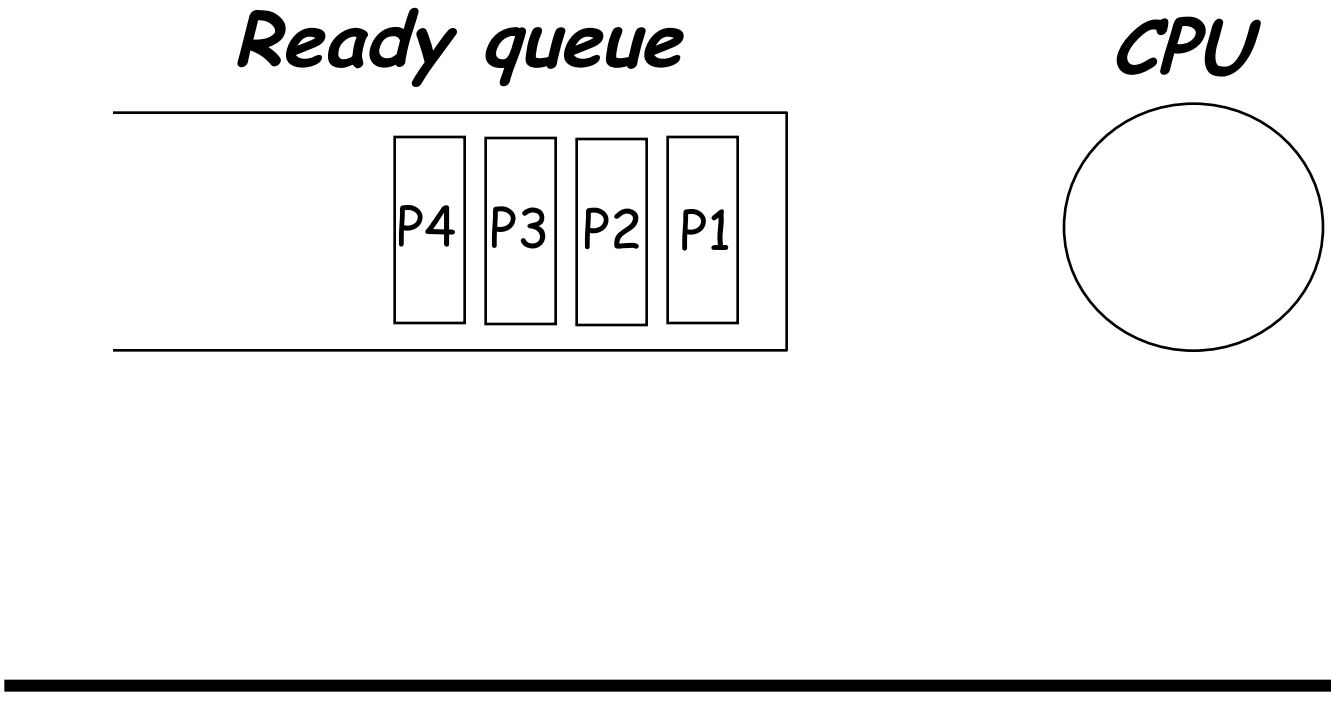
E una possibile soluzione : ***aging***

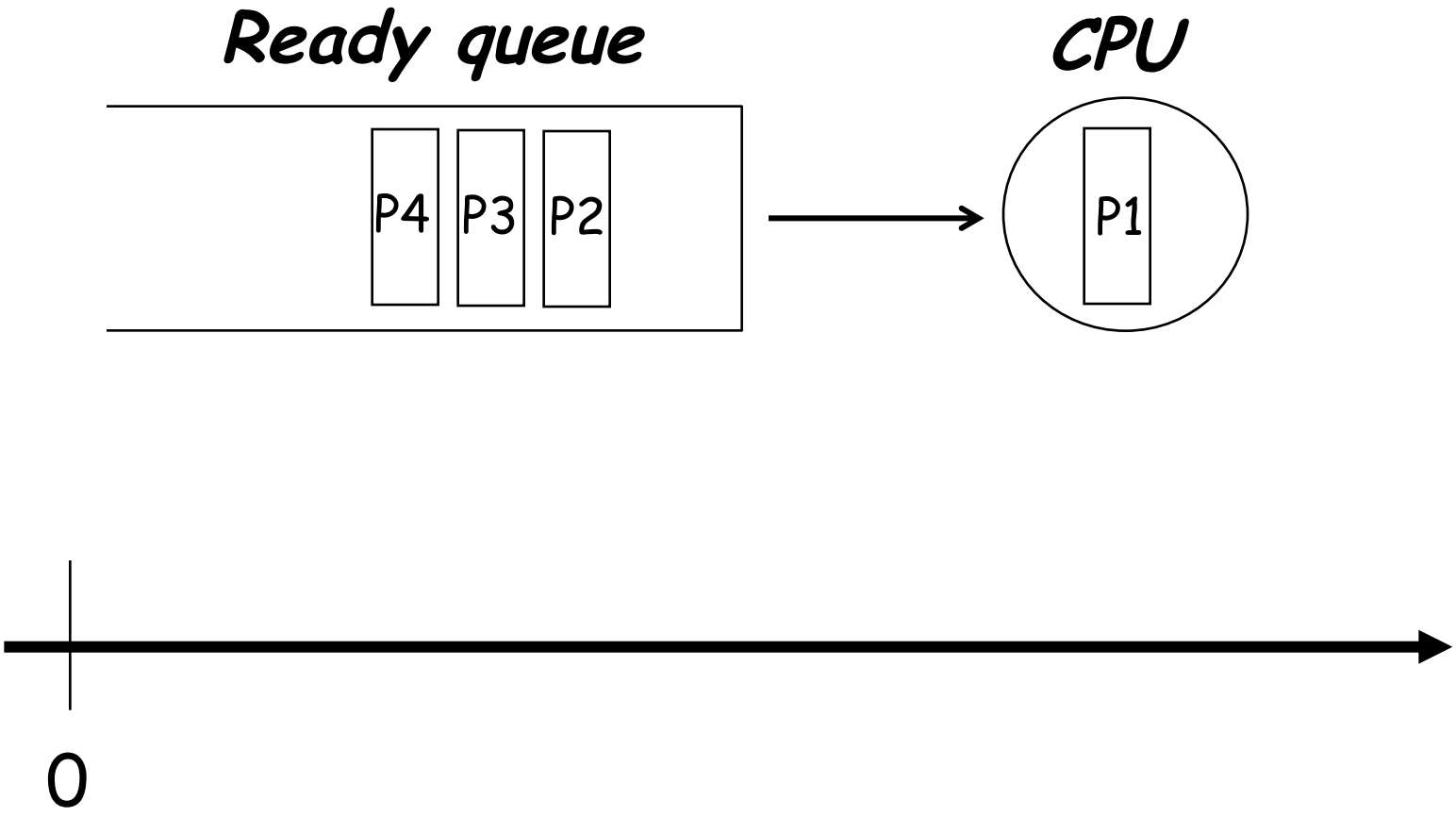
Col passare del tempo la
priorita' di un processo cresce

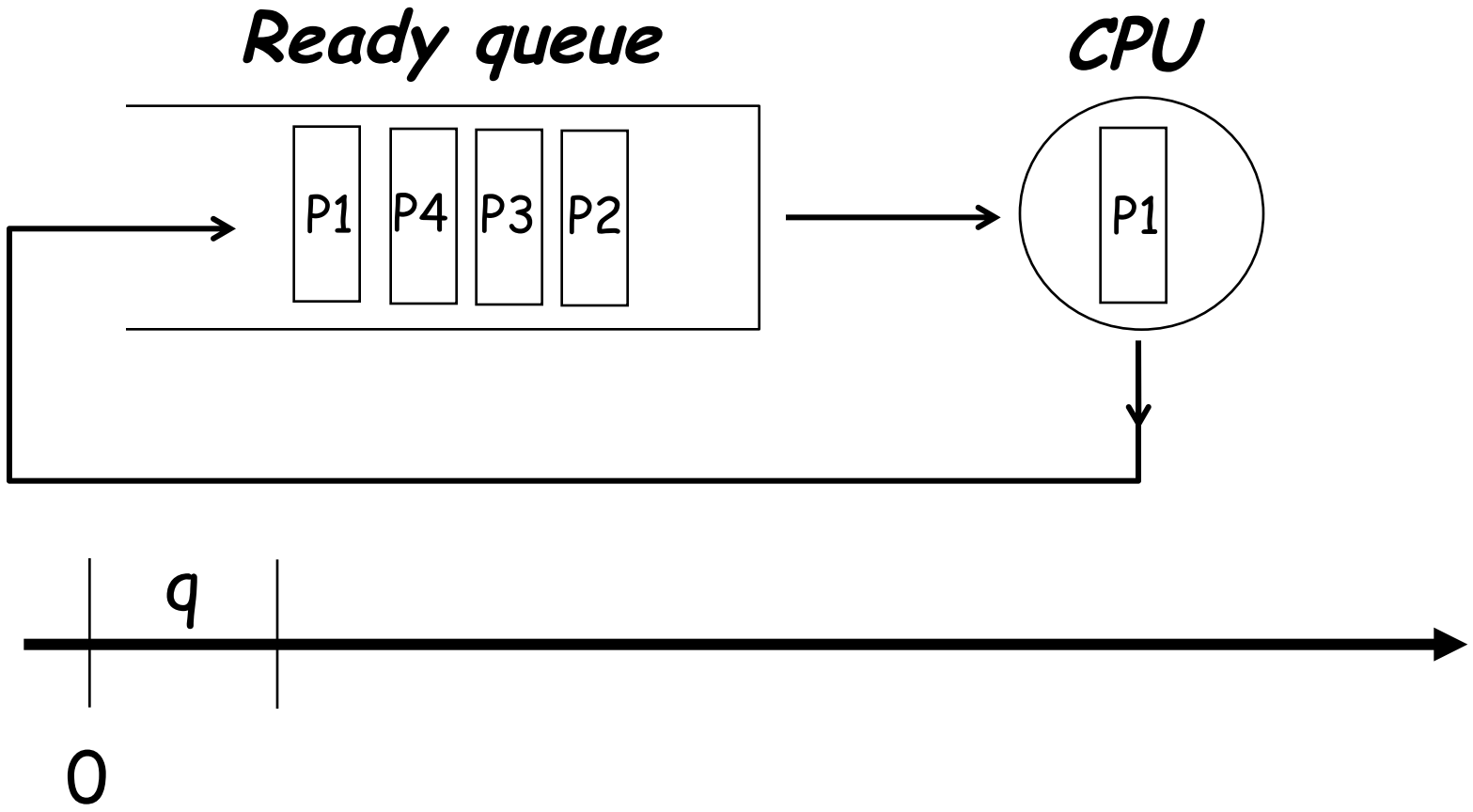
4. Round Robin (RR)

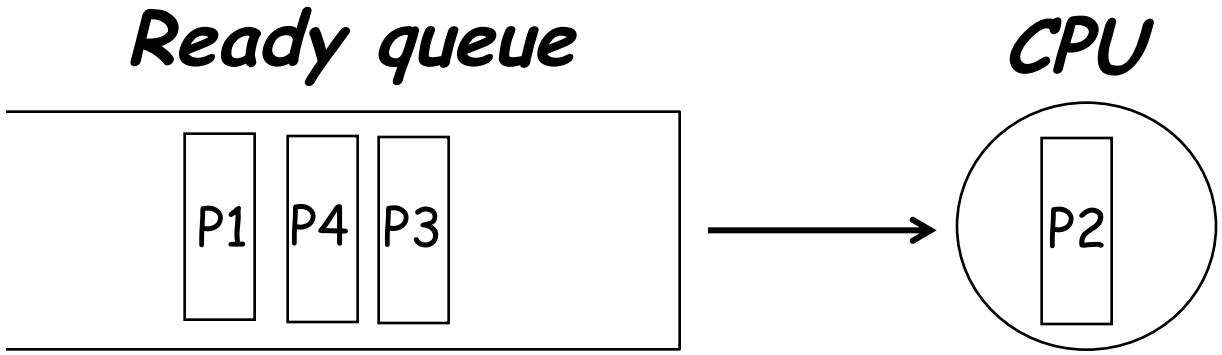
Ad ogni processo viene assegnata una piccola unita' di tempo di CPU (***quanto***), usualmente dell' ordine di alcune decine di millisecondi.

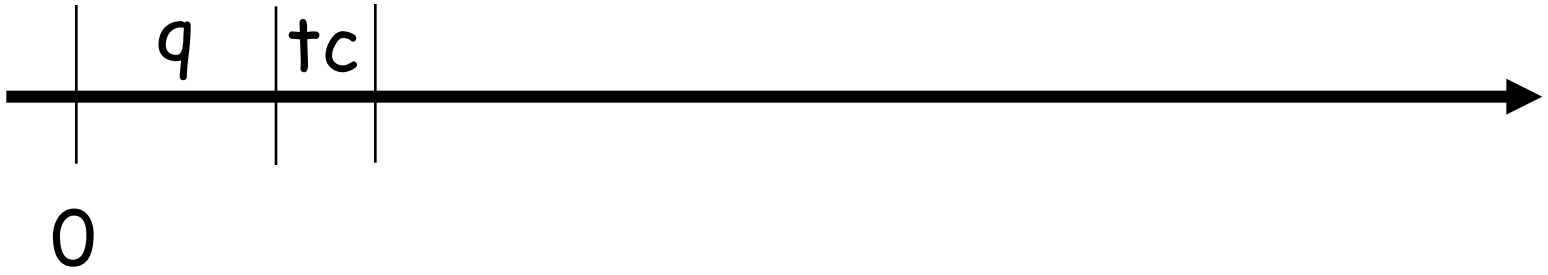
Quando questo tempo scade, il processo e' ***preempted*** e viene inserito ***alla fine della ready queue***


Ready queue
CPU
P4
P3
P2
P1


Ready queue
CPU
P4
P3
P2
P1
0


Ready queue
CPU
P1
P4
P3
P2
P1
q
0

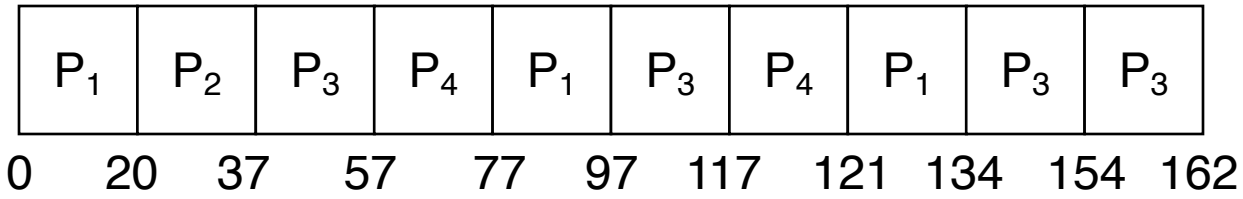




… e cosi' via …

Esempio: q = 20 ; tc = 0

Processo	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24



Usualmente un piu' alto turnaround time medio, ma un piu' breve tempo di risposta

Alcune considerazioni

Se ci sono ***n*** processi nella ready queue e il quanto e' ***q***, allora ogni processo utilizza una frazione ***1/n*** della CPU in segmenti di durata ***q***

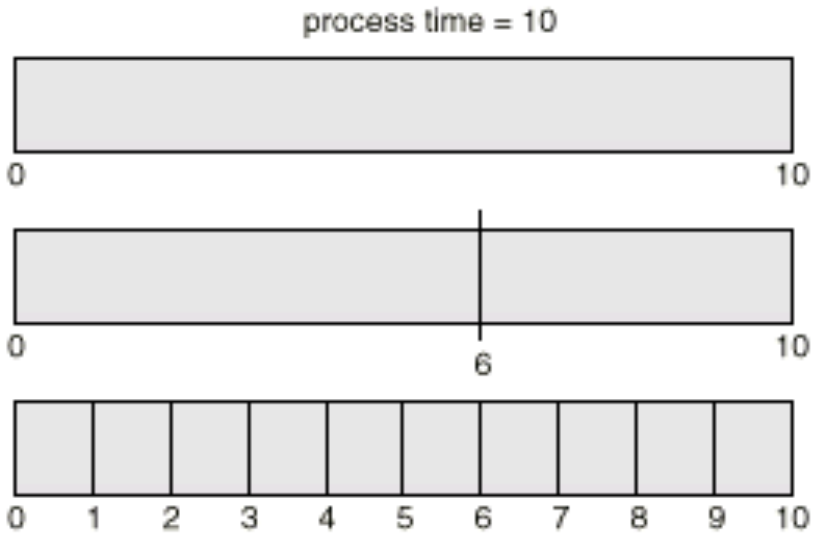
Nessun processo attende piu' di ***(n-1)q*** unita' di tempo per guadagnare la prima volta la CPU

Lunghezza del quanto:

q molto ***grande*** $\Rightarrow$ FCFS

q molto ***piccolo*** $\Rightarrow$ confrontabile con l' overhead introdotto dal context switching

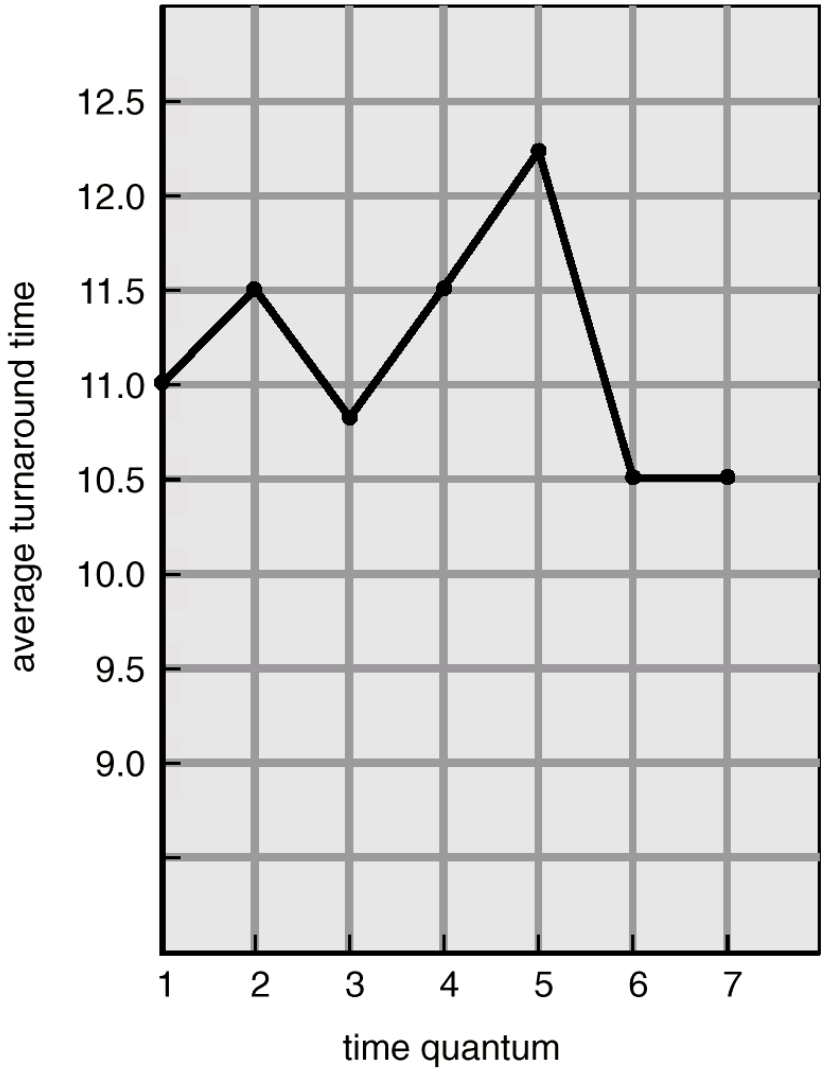
Lunghezza del quanto e numero di context-switching



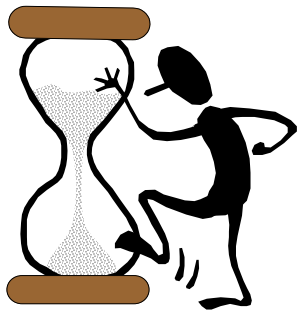
quantum	context switches
12	0
6	1
1	9

E' consigliabile che l'80% dei CPU burst siano piu' piccoli del quanto

Il Turnaround Time dipende dal quanto



process	time
P_1	6
P_2	3
P_3	1
P_4	7

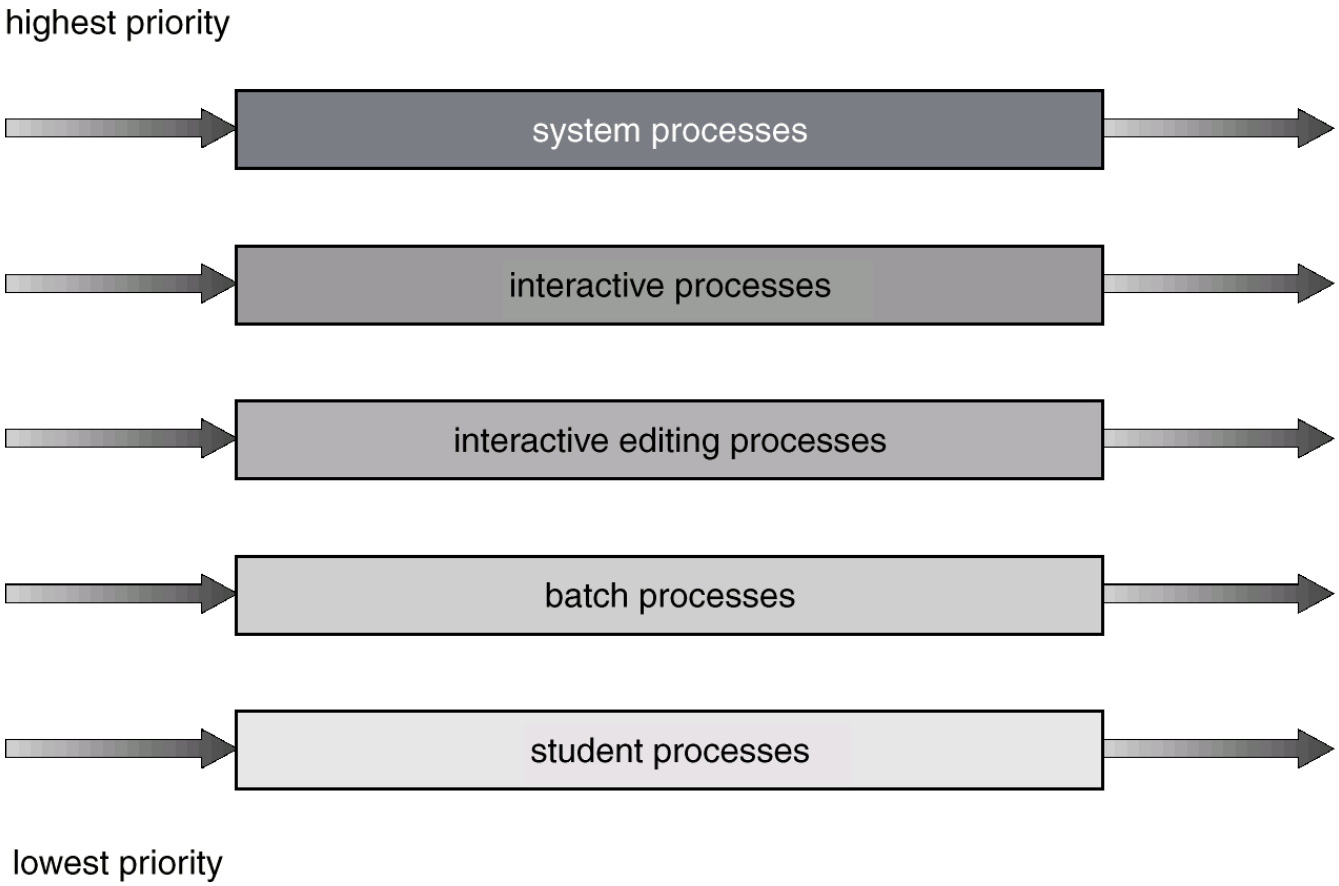


5. Code multilivello

- La ready queue e' partizionata in ***code separate*** e ad ogni coda viene assegnata una ***priorita'***

- Si servono prima tutti i processi in code ad alta priorita' e poi via via gli altri (preemptive o non-preemptive) $\rightarrow$ ***starvation***

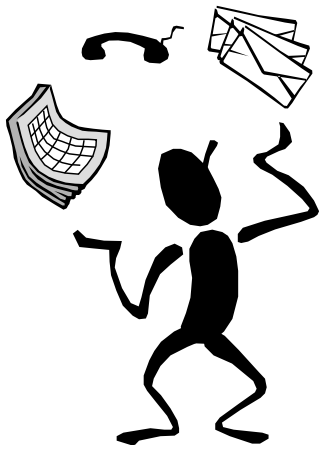
Una rappresentazione grafica



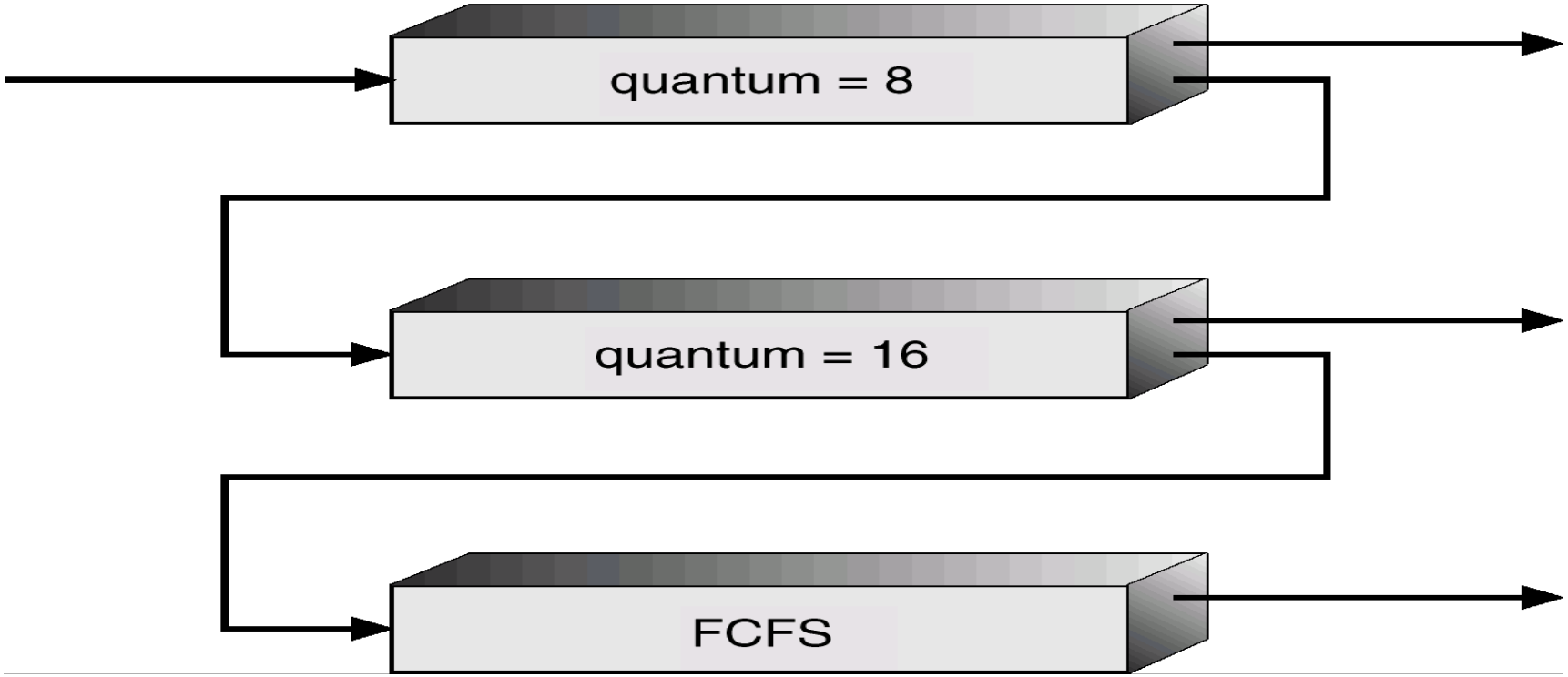
Ogni coda al suo interno ha la
propria politica di scheduling

Soluzioni alla starvation: CPU slicing

Ad ogni coda viene assegnata una certa ***percentuale di tempo di CPU*** che dividera' tra i suoi processi



Soluzioni alla starvation: code con feedback



Una speciale tecnica di ***aging*** :
un processo si puo' muovere tra le code

Esempio

- Tre code:
  - Q_0 – quanto: 8 milliseconds
  - Q_1 – quanto: 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - Un nuovo job entra nella coda Q_0 . Quando guadagna la CPU ci sta per 8 millisecondi. Se non finisce va della coda Q_1 .
  - In Q_1 il job aspetta il suo turno e quindi riceve la CPU per altri 16 milliseconds. Se ancora non ha completato va nella coda Q_2.

Parametri delle code con feedback

- Numero di code
- Algoritmo di scheduling di ogni coda
- Metodo usato per la migrazione dei processi tra code (anche eventualmente verso l'alto!)
- Coda in cui allocare un processo appena entrato