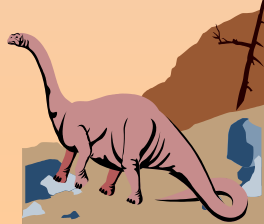




# Capitolo 6: Scheduling della CPU

- Concetti fondamentali.
- Criteri di scheduling.
- Algoritmi di scheduling.
- Scheduling per sistemi multiprocessore.
- Scheduling dei thread.
- Esempi di sistemi operativi. (solo Linux)
- Valutazione degli algoritmi.



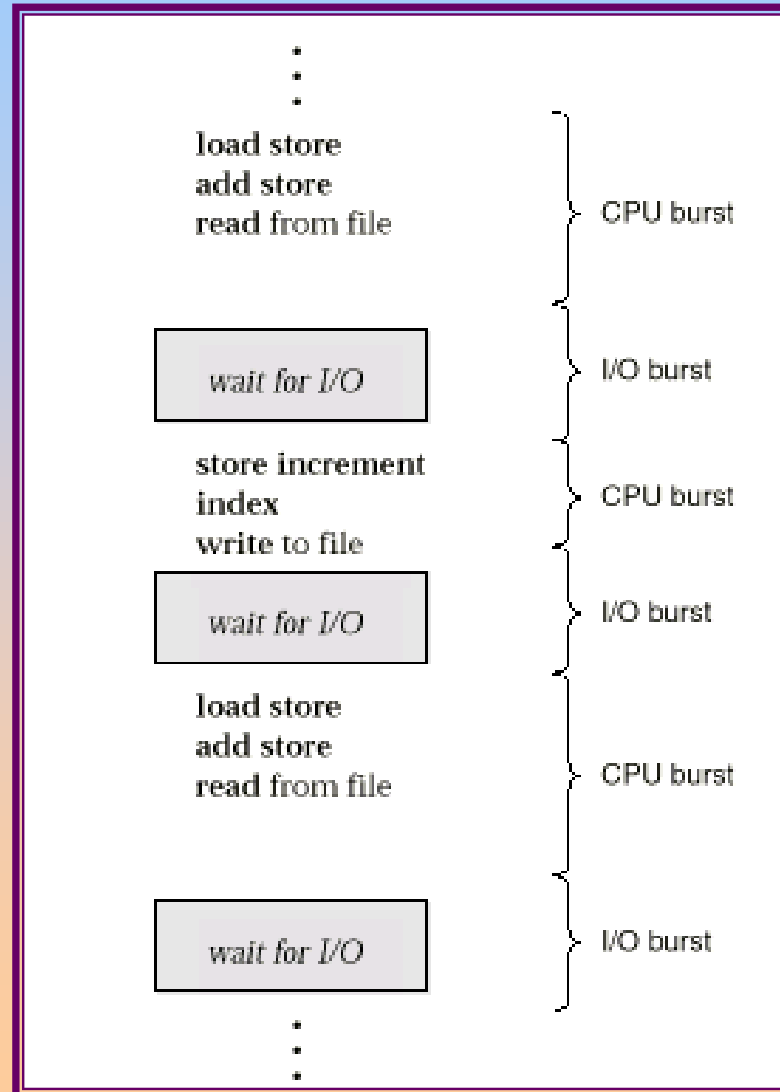


# Concetti fondamentali

- L'obiettivo della multiprogrammazione è:
  - ☞ l'esecuzione concorrente di più processi in modo da massimizzare l'utilizzo della CPU.
- L'obiettivo del time-sharing è:
  - ☞ commutare l'uso della CPU tra i vari processi così frequentemente, che gli utenti possano interagire con ciascun programma in esecuzione
- L'elaborazione di un processo è costituita da:
  - ☞ un ciclo di esecuzione di CPU,
  - ☞ uno di attesa di I/O.
  - ☞ I processi si alternano tra questi due stati.
- Le durate delle sequenze di operazioni della CPU sono state sperimentalmente misurate e la loro curva di frequenza è in genere simile a quella in figura.
- La curva è di tipo esponenziale con molte brevi sequenze di operazioni della CPU e poche sequenze di operazioni di CPU molto lunghe.
- Queste caratteristiche sono spesso utili per la scelta di un appropriato algoritmo di schedulazione della CPU.

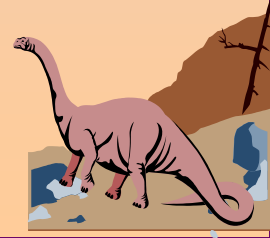
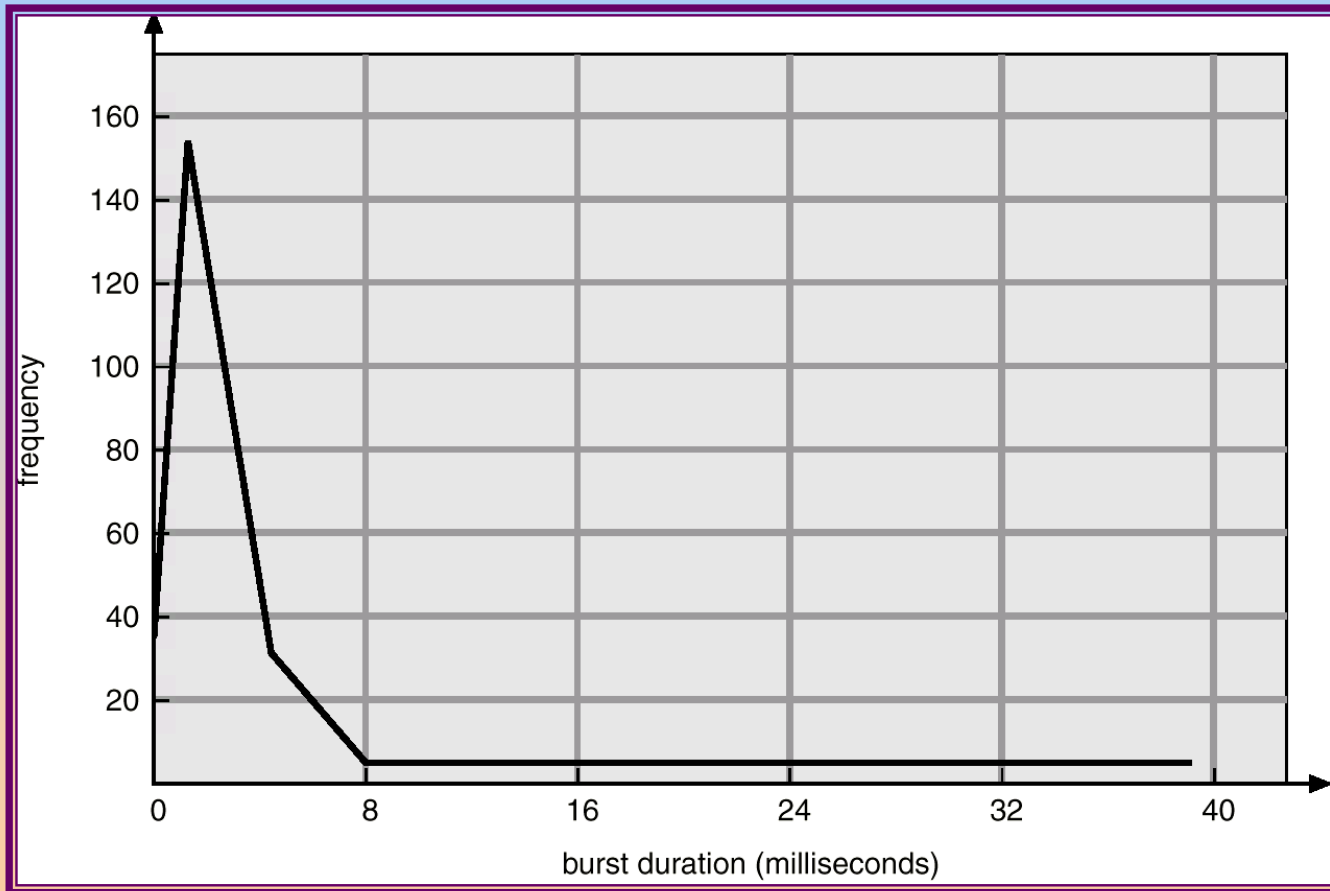


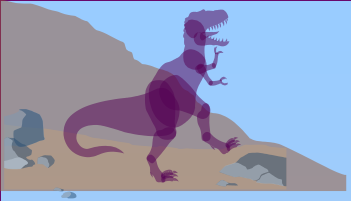
# Serie alternata di sequenze di operazioni della CPU e di sequenze di operazioni di I/O





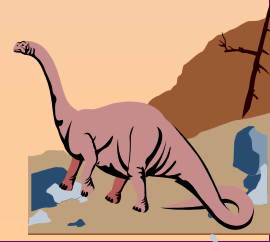
# Diagramma delle durate delle sequenze di operazioni della CPU





# Scheduler della CPU

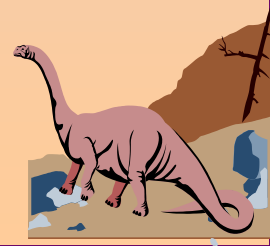
- Lo scheduler a breve termine (scheduler della CPU) sceglie il processo a cui assegnare la CPU tra quelli in memoria pronti per l'esecuzione,
- Le decisioni riguardanti lo scheduling della CPU vengono prese nelle seguenti circostanze:
  1. Quando un processo passa da uno stato running allo stato waiting:
    - 📄 in caso di richiesta di I/O, oppure di attesa per la terminazione di uno dei processi figli.
  2. Quando un processo passa da uno stato di running ad uno stato ready:
    - 📄 in caso di interrupt.
  3. Quando un processo passa da uno stato waiting ad uno stato ready:
    - 📄 in caso di completamento di un I/O.
  4. Quando un processo termina.





# Scheduler della CPU (II)

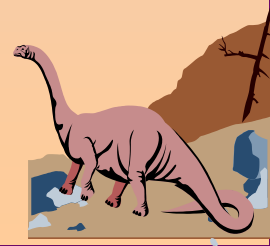
- Quando lo schema di scheduling interviene nelle condizioni 1 e 4 si dice che è senza diritto di prelazione (*non-preemptive*), altrimenti è con diritto di prelazione (*preemptive*).
- Gli algoritmi di scheduling vengono suddivisi quindi in due classi principali: *preemptive* e *non-preemptive*.
- Uno scheduling è **non-preemptive** se ogni processo a cui viene assegnata la CPU rimane in possesso della CPU fino a quando o termina la sua esecuzione oppure passa in uno stato di waiting.
- Altrimenti è **preemptive**.
- Se la politica di scheduling è preemptive allora bisognerà anche disporre di meccanismi per la sincronizzazione dei processi.





# Dispatcher

- Il dispatcher è il modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine.
- Il dispatcher opera quindi:
  - ☞ Il cambio di contesto
  - ☞ Il passaggio al modo d'utente
  - ☞ Il salto alla giusta posizione del programma utente per riavvianne l'esecuzione.
- Il tempo richiesto dal dispatcher per fermare un processo e avviare l'esecuzione di un altro è noto come *latenza di dispatch*.





# Criteri di scheduling

## ■ Utilizzo della CPU:

- ☞ la CPU deve essere più attiva possibile.

## ■ Produttività (throughput):

- ☞ # di processi che completano la loro esecuzione nell'unità di tempo.

## ■ Tempo di completamento (turnaround time):

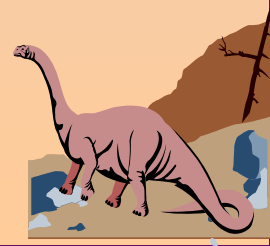
- ☞ intervallo che intercorre tra la sottomissione del processo ed il completamento dell'esecuzione.
- ☞ E' la somma dei tempi passati in attesa dell'ingresso nella memoria, nella coda dei processi pronti, durante l'esecuzione nella CPU e nel compiere operazioni di I/O.

## ■ Tempo di attesa:

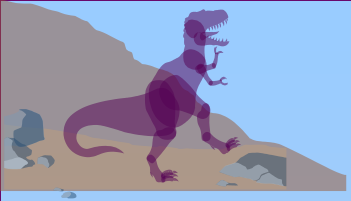
- ☞ la somma degli intervalli di attesa passati nella coda dei processi pronti.

## ■ Tempo di risposta:

- ☞ tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta (non l'output finale...).

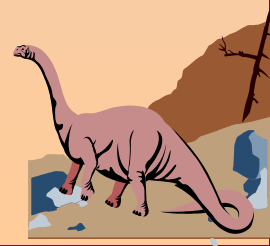






# Criteri di ottimizzazione

- Utilizzo massimo della CPU.
- Produttività massima.
- Minimo tempo di completamento.
- Minimo tempo di attesa.
- Minimo tempo di risposta.





# Scheduling First-Come, First-Served (FCFS)

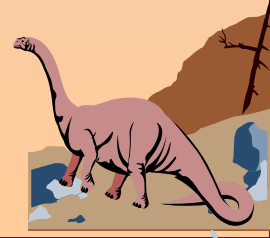
- Con questo schema la CPU si assegna al processo che la richiede per primo.
- Ad es.

<u>Processo</u>	<u>Durata della sequenza</u>
$P_1$	24
$P_2$	3
$P_3$	3

- Supponiamo che i processi arrivino in ordine:  $P_1$ ,  $P_2$ ,  $P_3$   
Il diagramma di Gantt per lo scheduling è:



- Tempo di attesa per  $P_1 = 0$ ;  $P_2 = 24$ ;  $P_3 = 27$
- Tempo di attesa medio:  $(0 + 24 + 27)/3 = 17$



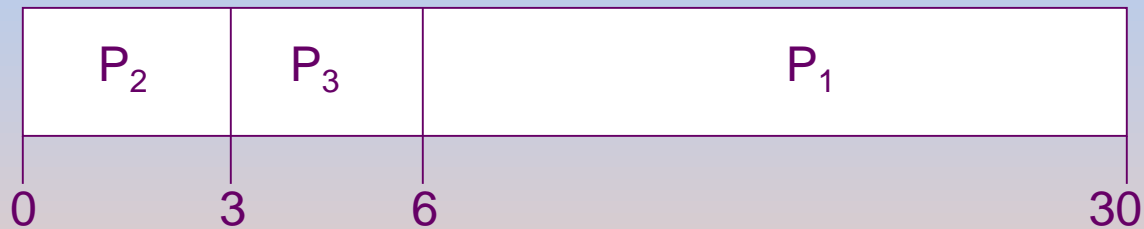


# Scheduling FCFS (II)

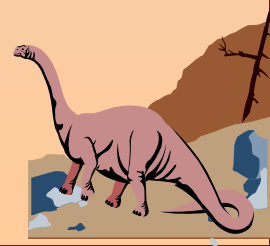
- Supponiamo che i processi arrivano in ordine:

$$P_2, P_3, P_1.$$

- Il diagramma di Gantt per lo scheduling è:



- Tempo di attesa per  $P_1 = 6$ ;  $P_2 = 0$ ;  $P_3 = 3$
- Tempo di attesa medio:  $(6 + 0 + 3)/3 = 3$
- Molto meglio dell'esempio precedente:
  - ☞ in FCFS il tempo di attesa medio può variare notevolmente al variare della durata dei CPU burst dei processi e del loro ordine di arrivo.
- *Effetto convoglio*: processi corti dietro i processi lunghi.
- FCFS è non preemptive.





# Scheduling per brevità (*Shortest-Job-First* - SJF)

- Questo algoritmo associa a ogni processo la lunghezza del suo CPU burst successivo.
- Quando la CPU è disponibile, viene assegnata al processo che ha il CPU burst successivo più breve.
- Se due processi hanno i CPU burst successivi della stessa lunghezza, si applica l'algoritmo FCFS.
- Due schemi:
  - ☞ Nonpreemptive: quando la CPU è allocata al processo non viene deallocata fino al completamento del burst di CPU
  - ☞ Preemptive: se arriva un nuovo processo con burst di CPU più corto del tempo di CPU rimanente al processo correntemente in esecuzione la CPU viene subito deallocata ed allocata al nuovo processo.
- 📄 Detto anche Shortest Remaining Time First: SRTF.
- SJF è ottimale nel senso che rende il tempo medio di attesa minimo per un dato insieme di processi.
- La difficoltà consiste nel conoscere la durata della successiva richiesta della CPU.

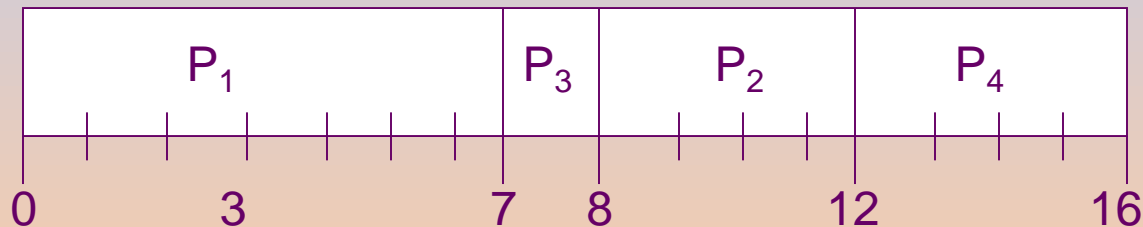




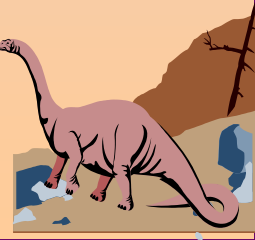
# Esempio di SJF Non-Preemptive

<u>Processo</u>	<u>Tempo di arrivo</u>	<u>Lunghezza del burst</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

## ■ SJF (non-preemptive)



- Il tempo di attesa è la somma degli intervalli di attesa passati nella coda dei processi pronti.
- Tempo di attesa medio =  $(0 + (8-2) + (7-4) + (12-5))/4 = 4$

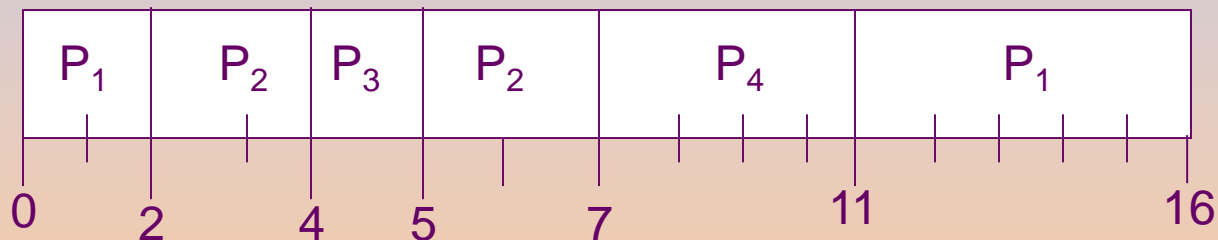




# Esempio di SJF Preemptive

<u>Processo</u>	<u>Tempo di arrivo</u>	<u>Lunghezza del burst</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

■ SJF (preemptive)



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





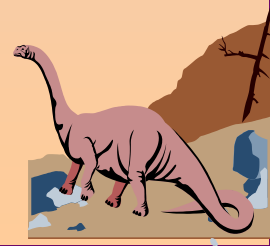
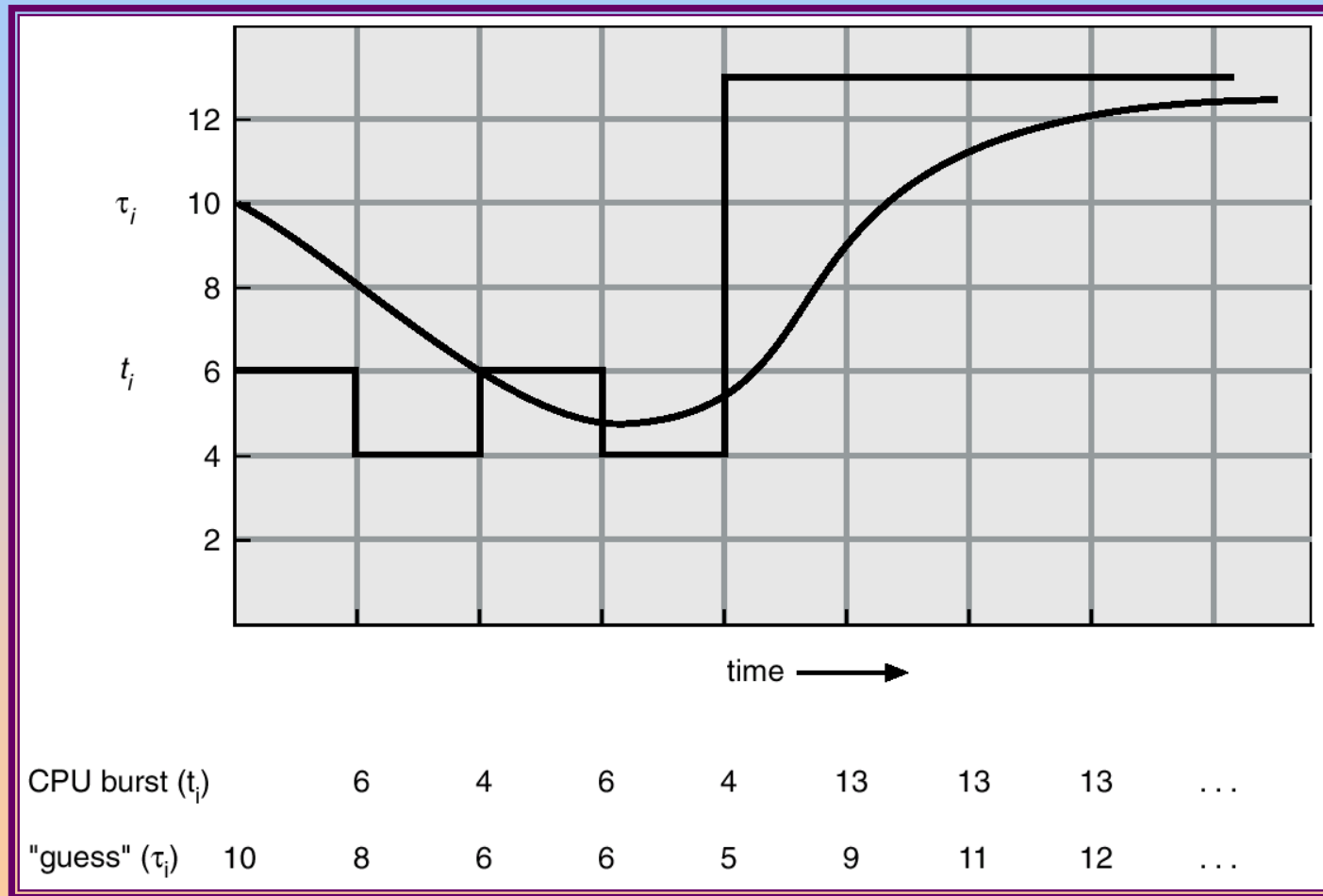
# Determinare la lunghezza del prossimo burst di CPU

- La lunghezza può solo essere stimata.
- Può essere fatto utilizzando le lunghezze dei burst di CPU precedenti:
  - ☞ calcolando la *media esponenziale* delle effettive lunghezze delle precedenti sequenze di operazioni della CPU:
    1.  $t_n$  = lunghezza attuale del  $n^{th}$  CPU burst
    2.  $\tau_{n+1}$  = valore predetto del prossimo CPU burst
    3.  $\alpha$ ,  $0 \leq \alpha \leq 1$
    4. Definiamo :
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n.$$
- $t_n$  contiene le informazioni più recenti,  $\tau_n$  registra la storia passata,  $\alpha$  controlla il peso relativo sulla predizione della storia recente.





# Predizione della lunghezza della successiva sequenza di operazioni della CPU ( $\alpha = 1/2$ )







# Esempi di media esponenziale

## ■ $\alpha = 0$

☞  $\tau_{n+1} = \tau_n$

☞ La storia recente non conta (le condizioni attuali sono transitorie).

## ■ $\alpha = 1$

☞  $\tau_{n+1} = t_n$

☞ Conta solo l'ultimo CPU burst.

## ■ Se espandiamo la formula $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$ otteniamo:

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} t_n \tau_0\end{aligned}$$

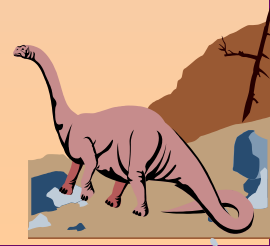
## ■ Giacché entrambi $\alpha$ e $(1 - \alpha)$ sono minori o uguali di 1, ciascun termine successivo ha peso minore del suo predecessore.





# Scheduling per priorità

- Un valore di priorità (intero) viene associato ad ogni processo.
- La CPU viene allocata al processo con priorità più alta (intero più basso  $\equiv$  priorità più alta ).
  - ☞ Preemptive
  - ☞ nonpreemptive
- SJF è uno scheduling a priorità dove la priorità è il valore predetto del prossimo burst di CPU.
- Problema: *blocco indefinito o starvation*:
  - ☞ processi a bassa priorità potrebbero non essere mai eseguiti
- Soluzione: *invecchiamento (aging)*:
  - ☞ con il passare del tempo incrementare la priorità del processo.





# Scheduling circolare (Round Robin - RR)

- Ogni processo ottiene una piccola quantità fissata di tempo di CPU (*quanto di tempo*), in genere 10-100 millisecondi.
- Allo scadere di questo tempo il processo viene sospeso e aggiunto alla fine della coda dei processi pronti.
- Se ci sono  $n$  processi nella coda dei processi pronti e il quanto di tempo è  $q$  :
  - ☞ allora ciascun processo riceve  $1/n$  del tempo di CPU in blocchi di al più  $q$  unità alla volta.
- Nessun processo rimane sospeso per più di  $(n-1)q$  unità di tempo.
- Prestazioni:
  - ☞ Valore di  $q$  grande  $\Rightarrow$  FIFO
  - ☞ Valore di  $q$  piccolo  $\Rightarrow q$  deve essere sufficientemente grande rispetto al tempo necessario al cambio di contesto, altrimenti l'overhead è troppo alto.

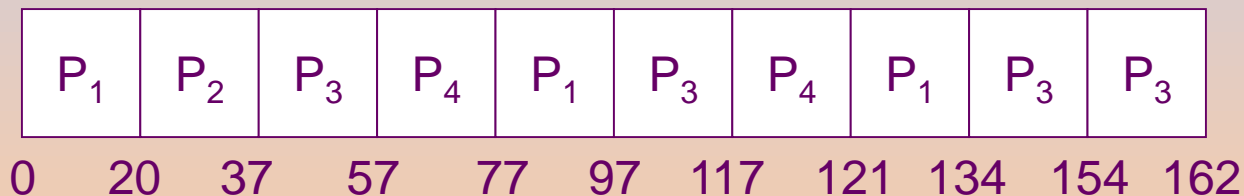




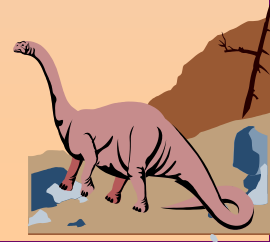
# Esempio di RR con quanto di tempo = 20

<u>Processo</u>	<u>Lunghezza del burst</u>
$P_1$	53
$P_2$	17
$P_3$	68
$P_4$	24

- Il diagramma di Gantt per lo scheduling è:

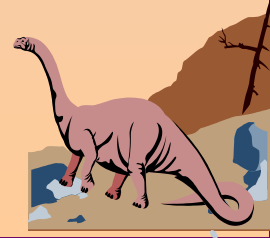
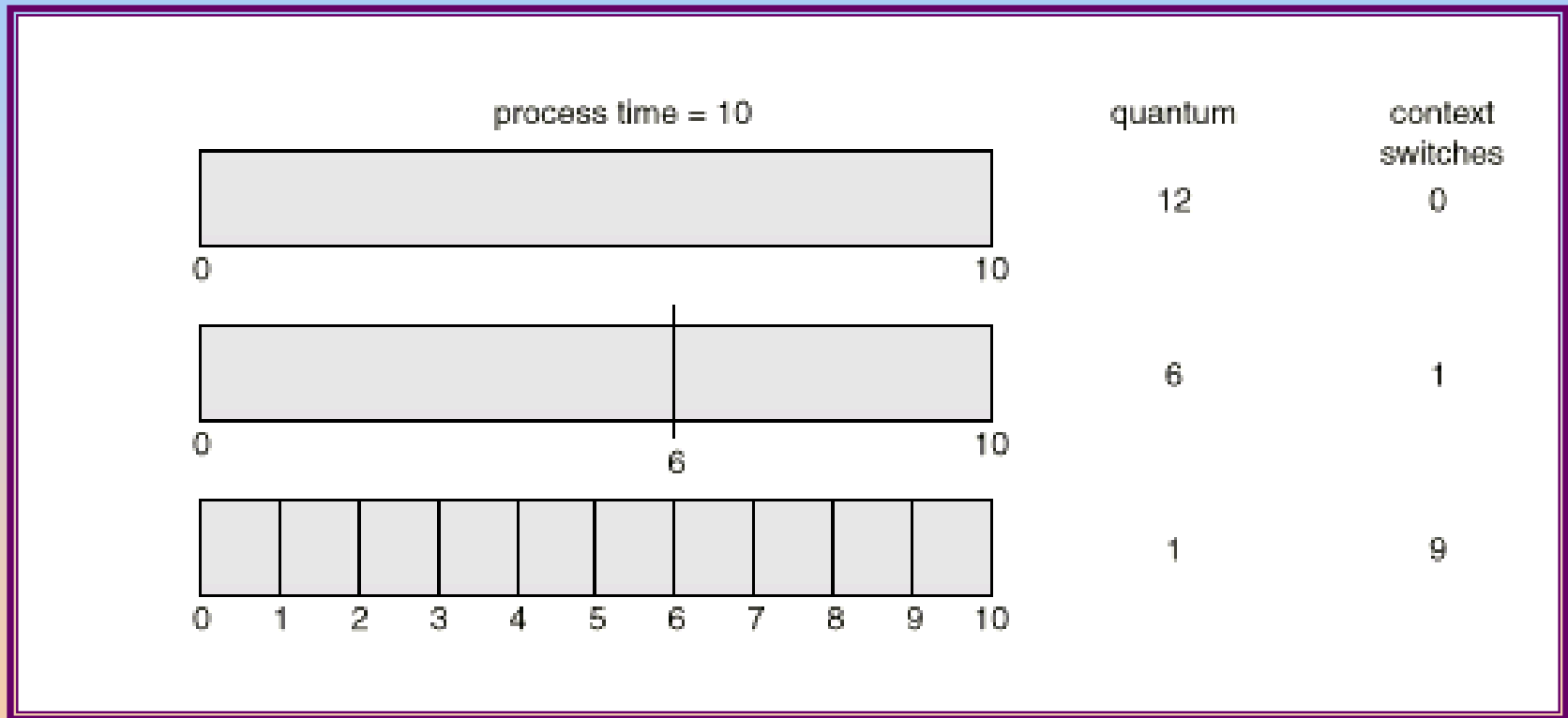


- Generalmente ha tempo di completamento medio più alto di SJF ma tempo di risposta migliore.



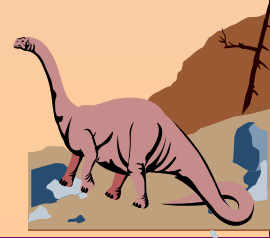
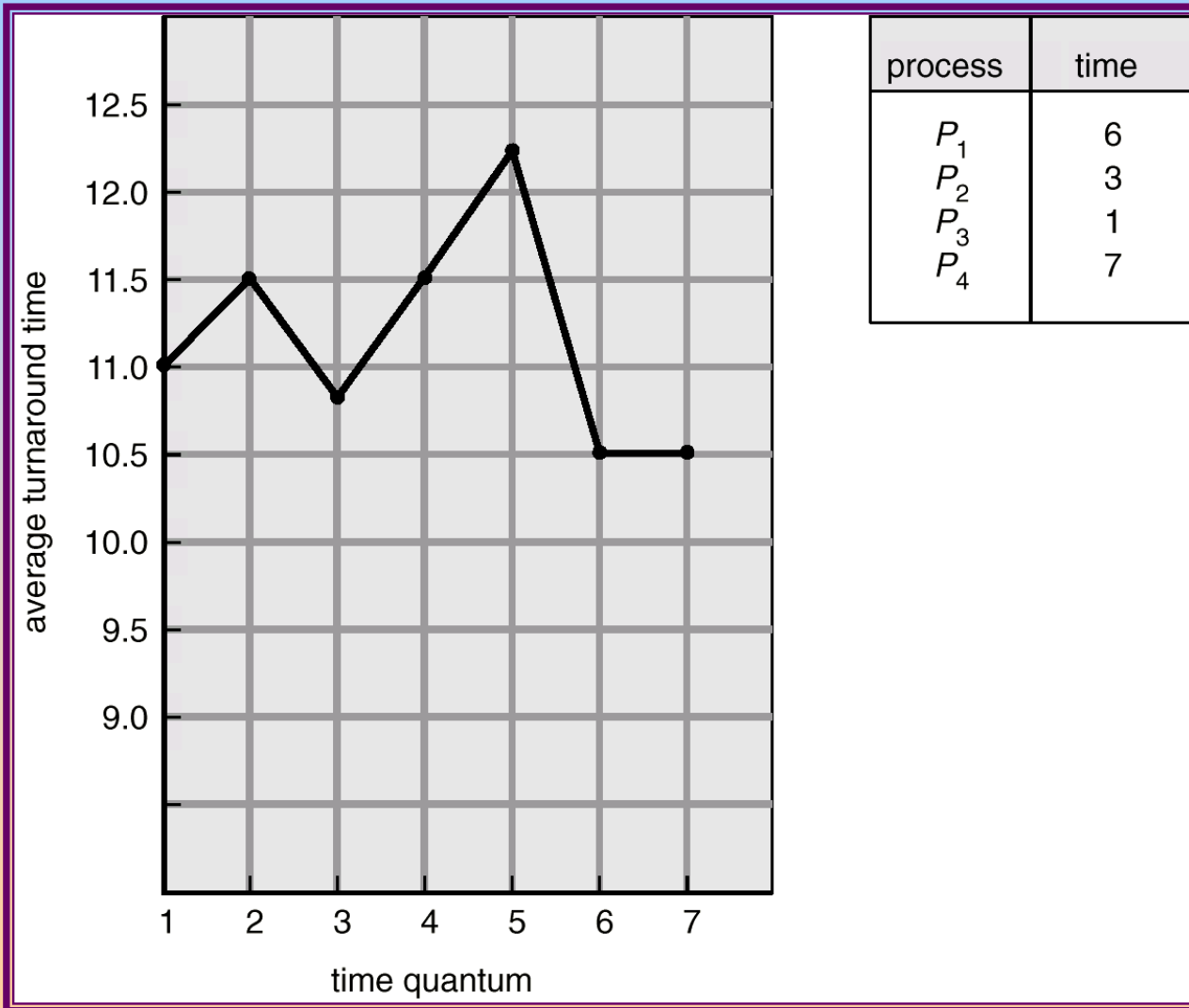


# Un quanto di tempo minore incrementa il numero di cambi di contesto





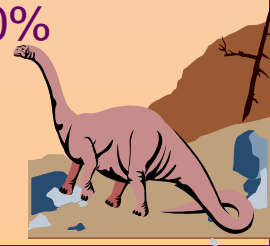
# Variazione del tempo di completamento rispetto alla dimensione del quanto di tempo





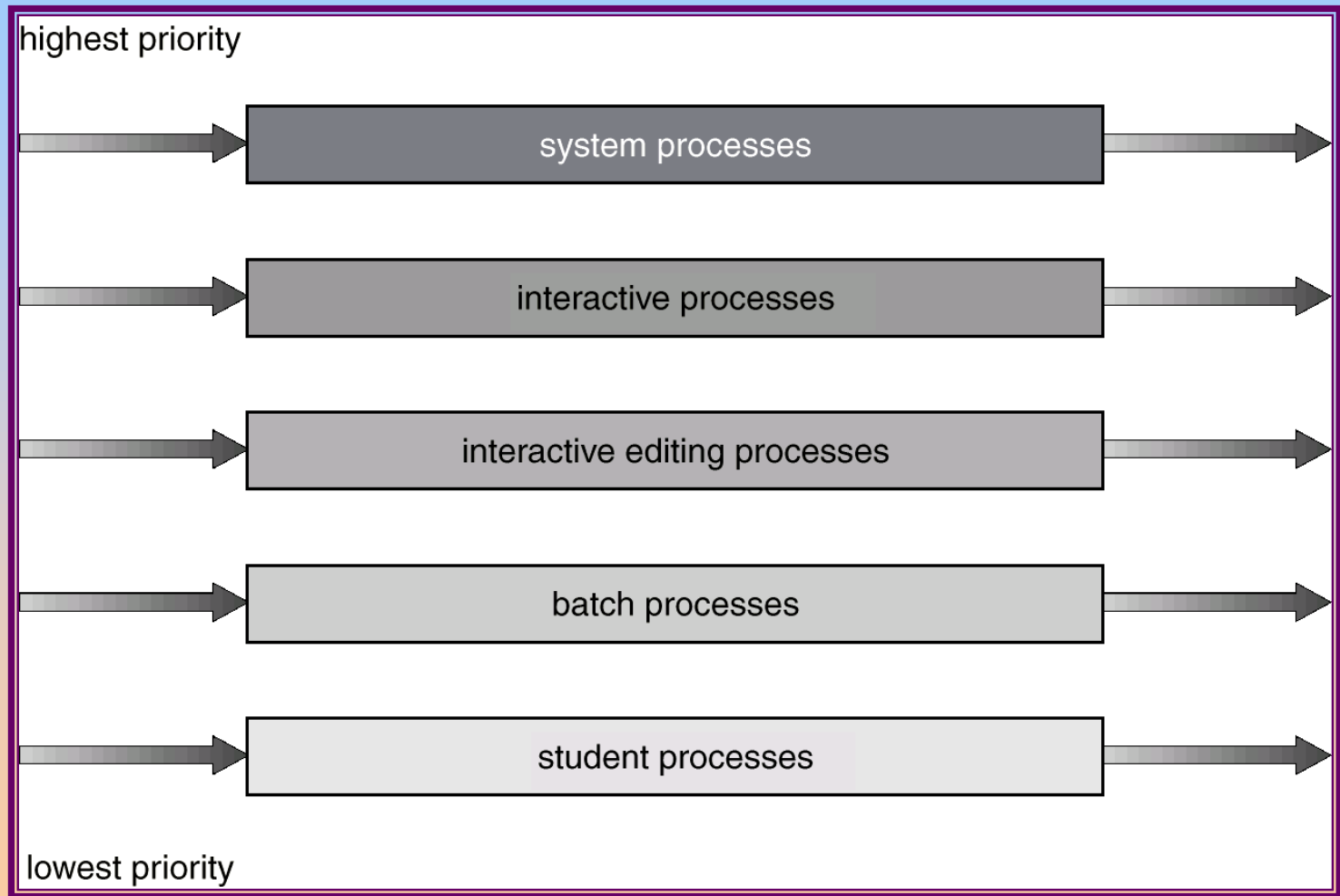
# Scheduling a code multiple

- La coda dei processi pronti è partizionata in più code separate.
- I processi non possono essere spostati tra le varie code.
  - ☞ Ad es. processi che si eseguono in primo piano (foreground) o interattivi e quelli che si eseguono in sottofondo (background), o a lotti (batch).
- Ciascuna coda ha il proprio algoritmo di scheduling.
  - ☞ Ad es: foreground – RR, background – FCFS
- E' necessario avere uno scheduling tra le code:.
  - ☞ Scheduling a priorità fissa;
    - 📄 ad es. eseguire tutti i processi nella coda foreground e passare solo dopo alla background.
    - 📄 Possibilità di starvation.
  - ☞ Quanti di tempo: ciascuna coda ottiene un quanto di tempo della CPU con cui può schedare i processi in coda.
    - 📄 ad es., 80% alla coda dei processi in foreground in RR e 20% all'altra in FCFS

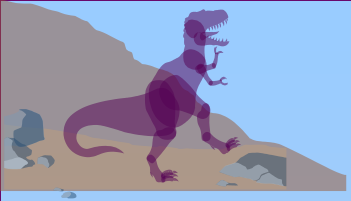




# Scheduling a code multiple (II)

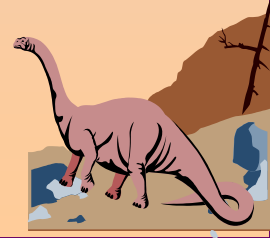






# Scheduling a code multiple con retroazione

- Un processo può essere spostato da una coda ad un'altra, ad es. per *invecchiamento*.
- Lo scheduling a code multiple con retroazione è caratterizzato dai seguenti parametri:
  - ☞ numero di code,
  - ☞ algoritmo di scheduling per ciascuna coda,
  - ☞ metodo usato per determinare quando spostare un processo in una coda con priorità maggiore,
  - ☞ metodo usato per determinare quando spostare un processo in una coda con priorità minore,
  - ☞ metodo usato per determinare in quale coda si deve mettere un processo quando richiede un servizio.





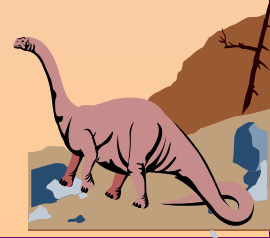
# Esempio di scheduling a code multiple con retroazione

## ■ Tre code:

- ☞  $Q_0$  – quanto di tempo 8 millisecondi
- ☞  $Q_1$  – quanto di tempo 16 millisecondi
- ☞  $Q_2$  – FCFS

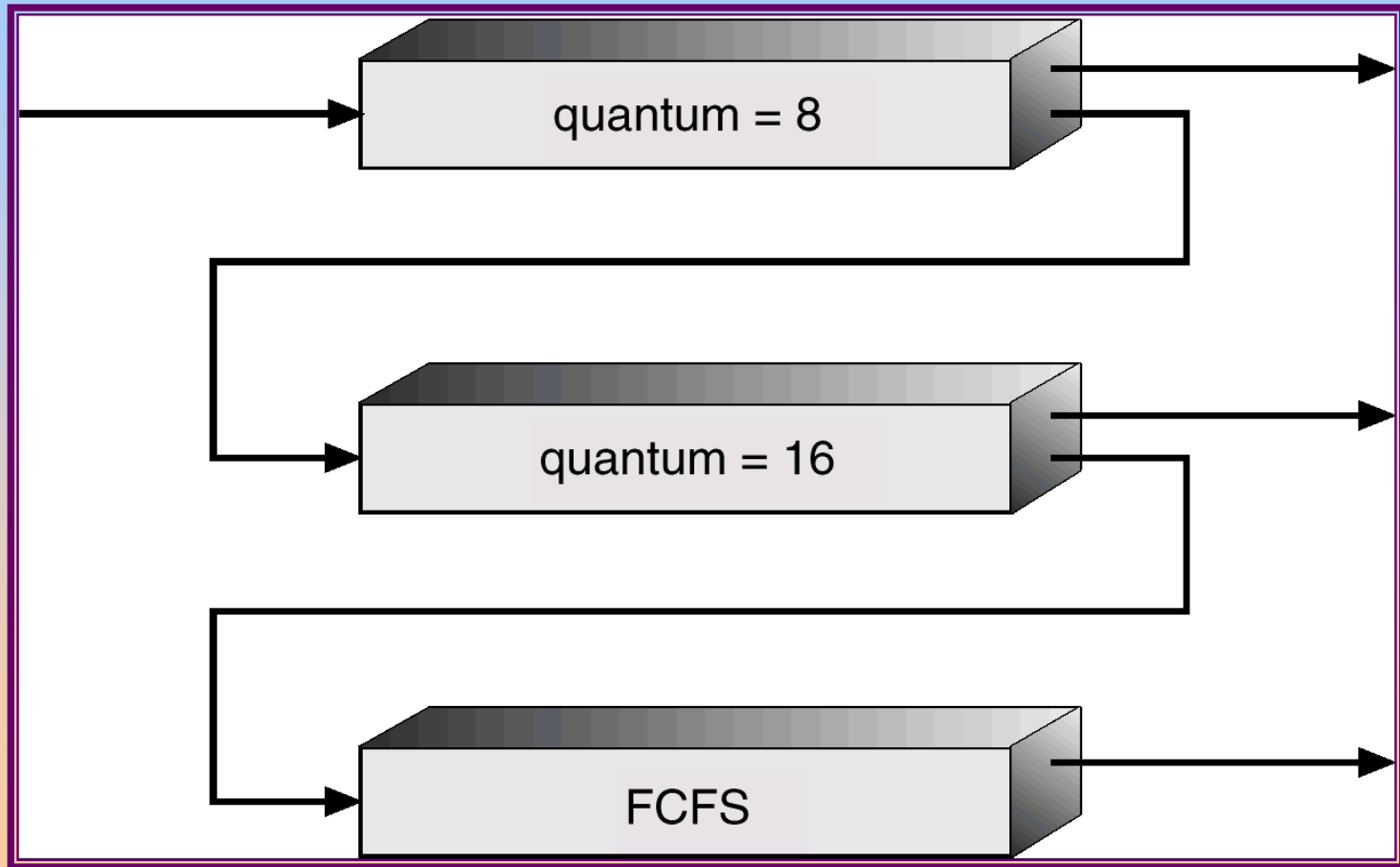
## ■ Scheduling:

- ☞ Un nuovo processo entra nella coda  $Q_0$  schedulata con FCFS.
- ☞ Quando gli viene assegnata la CPU, riceve 8 millisecondi.
- ☞ Se non finisce in 8 millisecondi, viene spostato nella coda  $Q_1$ .
- ☞ Nella coda  $Q_1$  il processo è di nuovo schedulato con FCFS e riceve altri 16 millisecondi.
- ☞ Se non termina entro questo quanto di tempo, viene deallocato e spostato nella coda  $Q_2$ .





# Esempio di scheduling a code multiple con retroazione (II)





# Scheduling per sistemi con più unità di elaborazione

- Lo scheduling della CPU diviene più complesso se sono disponibili più CPU.
- Si considereranno i *sistemi omogenei*:
  - ☞ sistemi nei quali le unità di elaborazione sono, in relazione alle loro funzioni, identiche.
- Quindi si può usare qualunque unità di elaborazione disponibile per eseguire qualunque processo presente nella coda.
- *Condivisione del carico*:
  - ☞ ad esempio si ha una coda dei processi pronti comune e quando un'unità di elaborazione è disponibile le viene assegnato il prossimo processo.
- Due criteri di scheduling:
  - ☞ avere una CPU master che assegna i processi da eseguire alle altre,
  - ☞ oppure ogni unità di elaborazione esamina la coda dei processi pronti e sceglie un processo da eseguire.





# Soluzioni di scheduling per multiprocessori

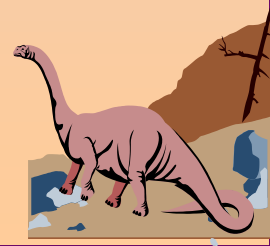
## ■ *Multielaborazione asimmetrica:*

- ☞ una prima strategia di scheduling affida tutte le decisioni, l'elaborazione dell'I/O e le altre attività del sistema ad un solo processore.
- ☞ Gli altri processori eseguono soltanto il codice dell'utente
- ☞ quindi un'unica unità di elaborazione (*master server*) ha accesso ai dati di sistema e gestisce la schedulazione e le attività di sistema, semplificando la condivisione dei dati.

## ■ Quando invece ciascun processore provvede al proprio scheduling si parla di *multielaborazione simmetrica* o di *SMP*.

- ☞ In questo caso i processi pronti sono situati tutti in una coda comune o c'è un'apposita coda per ogni processore.
- ☞ Lo scheduler di ogni processore esamina la coda appropriata da cui seleziona un processo da eseguire.

## ■ La SMP e' messa a disposizione da quasi tutti i S.O. moderni: Windows XP, Solaris, Linux, Mac OS X, etc.





# Soluzioni di scheduling per multiprocessori (II)

## ■ *Predilezione per il processore:*

- ☞ significa tentare di riallocare ad un processo la stessa CPU su cui era in esecuzione prima.
- ☞ Tecnica usata per sfruttare al meglio le cache locale del processore.
  - 📄 I dati del processo già presenti in cache.

## ■ *Bilanciamento del carico:*

- ☞ nei sistemi SMP si tende a distribuire uniformemente i processi tra i processori disponibili (push migration e pull migration).
- ☞ E' necessario solo nei sistemi in cui ogni processore detiene una coda privata di processi da eseguire.
- ☞ Due approcci: migrazione guidata e migrazione spontanea.
- ☞ La prima prevede che un processo apposito controlli periodicamente il carico di ogni processore ed eventualmente riporti il carico in equilibrio.
- ☞ La seconda si ha quando un processore inattivo sottrae ad un processore sovraccarico un processo in attesa.





# Scheduling per sistemi di elaborazione in tempo reale

- *Tempo reale stretto (hard real-time):*
  - ☞ sistemi capaci di garantire il completamento delle funzioni critiche entro un tempo definito.
- Un processo si presenta insieme con una dichiarazione del tempo entro cui deve completare le proprie funzioni.
- Se è possibile garantirne il completamento lo scheduler accetta il processo, altrimenti rifiuta la richiesta (*prenotazione delle risorse*).
- *Tempo reale debole (soft real-time):*
  - ☞ i processi critici devono avere una priorità più alta dei processi ordinari.





# Esempio: scheduling di Linux

- Lo scheduler di Linux ricorre ad un algoritmo di scheduling con prelazione basato sulle priorità.
- Ci sono due gamme di priorità separate:
  - ☞ un intervallo real-time che va da 0 a 99
  - ☞ un intervallo nice compreso tra 100 e 140.
- I valori di questi due range sono mappati in un valore di priorità globale.
  - ☞ Numeri bassi implicano priorità alta.
- Il kernel mantiene una lista di tutti i task in una *runqueue*.
- La runqueue contiene due array di priorità:
  - ☞ attivo e scaduto (active array ed expired array).
    - 📄 il primo contiene tutti i task che hanno ancora tempo da sfruttare,
    - 📄 il secondo i task scaduti.

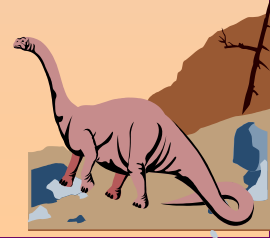






# Relazione fra le priorità e la lunghezza del quanto di tempo

QuickTime™ e un  
decompressore TIFF (LZW)  
sono necessari per visualizzare quest'immagine.





# Valutazione degli algoritmi

## ■ Modelli deterministici:

- ☞ dato un carico di lavoro predeterminato definiscono una formula o un algoritmo che valuta le prestazioni dell'algoritmo di schedulazione per quel carico di lavoro.
- ☞ Necessitano in genere di conoscenze troppo dettagliate ed impossibili da ottenere.

## ■ Reti di code:

- ☞ il sistema di calcolo si descrive come una rete di unità serventi, ciascuna con una coda di attesa:
  - 📄 la CPU è un'unità servente con la propria coda dei processi pronti, il sistema di I/O con le code dei dispositivi, etc..
- ☞ Se sono noti (o stimabili) l'andamento degli arrivi e dei servizi, si possono calcolare l'utilizzo, la lunghezza media delle code, il tempo medio di attesa, etc.
- ☞ Questo tipo di studio si chiama analisi delle reti di code.
- ☞ Il risultato è una approssimazione, non sempre esatta, del sistema reale.





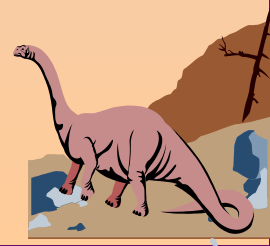
# Valutazione degli algoritmi (II)

## ■ Simulazioni:

- ☞ implicano la programmazione di un modello del sistema di calcolo.
- ☞ I processi sono simulati, ad esempio, attraverso generatori di numeri casuali che simulano le quantità di risorse necessarie ad ogni processo.
- ☞ Il simulatore dispone di una variabile che rappresenta un clock.
- ☞ Con l'aumentare del valore di questa variabile, il simulatore modifica lo stato del sistema in modo da descrivere le attività dei dispositivi, dei processi e dello scheduler.
- ☞ Sono spesso onerose, in termini di risorse di calcolo, e non sempre precise.
- ☞ Durante l'esecuzione della simulazione si raccolgono e si stampano statistiche che indicano le prestazioni degli algoritmi.

## ■ Realizzazione:

- ☞ l'unico modo assolutamente sicuro di valutare un algoritmo di scheduling consiste nel codificarlo, inserirlo nel S.O. ed osservarne il comportamento nelle reali condizioni di funzionamento.





# Valutazione di uno scheduler di CPU per simulazione

