

# *Sistemi Operativi*

## *Modulo 4: Scheduling*

Renzo Davoli  
Alberto Montresor

Copyright © 2002-2005 Renzo Davoli, Alberto Montresor

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at:

<http://www.gnu.org/licenses/fdl.html#TOC1>

# Sezione 1



## 1. Scheduler, processi e thread

# Introduzione



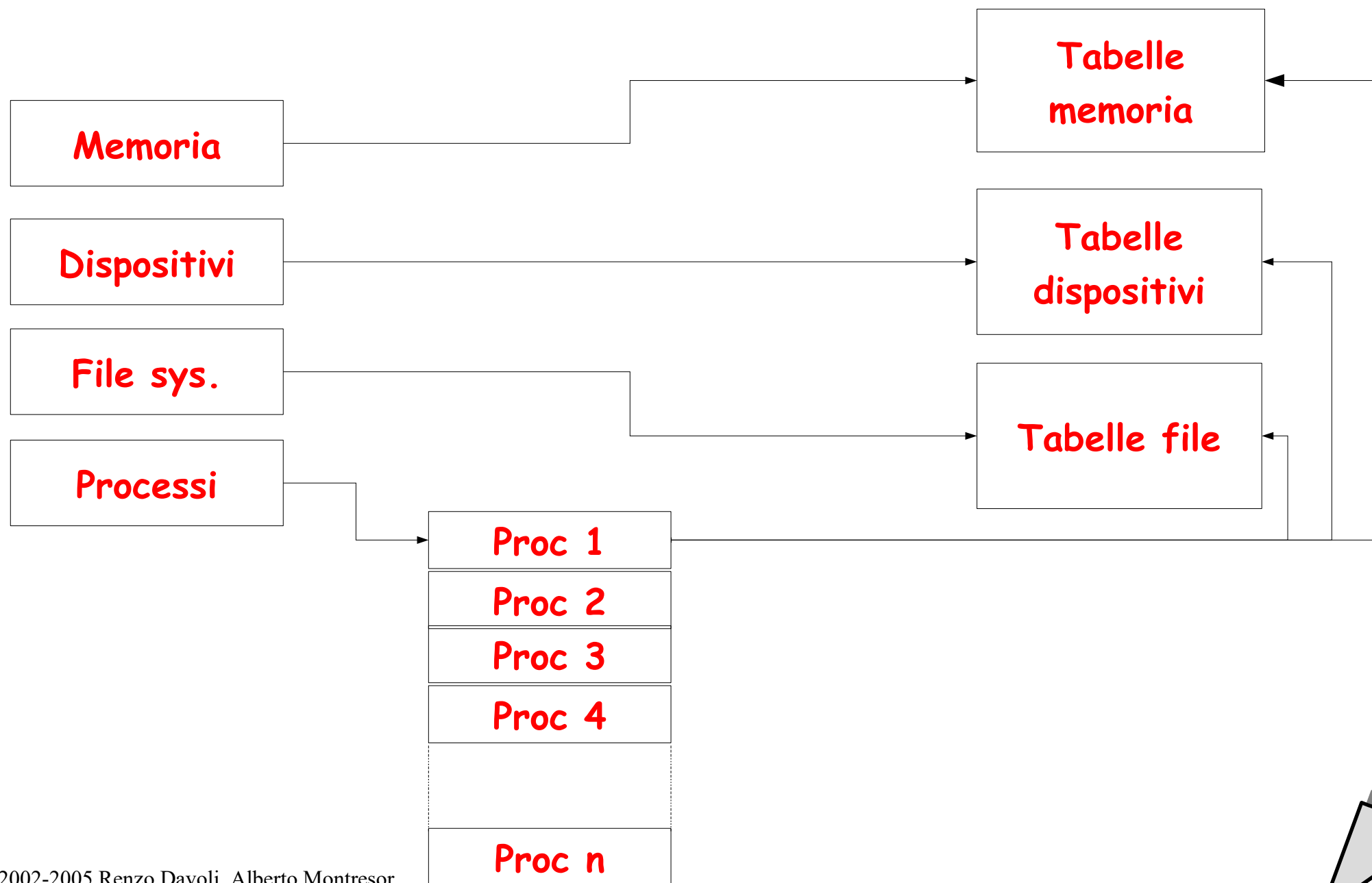
- ♦ **Un sistema operativo è un gestore di risorse**
    - ♦ processore, memoria principale e secondaria, dispositivi
  - ♦ **Per svolgere i suoi compiti, un sistema operativo ha bisogno di strutture dati per mantenere informazioni sulle risorse gestite**
  - ♦ **Queste strutture dati comprendono:**
    - ♦ tabelle di memoria
    - ♦ tabelle di I/O
    - ♦ tabelle del file system
    - ♦ tabelle dei processi
- **Argomento di questo modulo**

# Introduzione



- ◆ **Tabelle per la gestione della memoria**
  - ◆ allocazione memoria per il sistema operativo
  - ◆ allocazione memoria principale e secondaria per i processi
  - ◆ informazioni per i meccanismi di protezione
- ◆ **Tabelle per la gestione dell'I/O**
  - ◆ informazioni sullo stato di assegnazione dei dispositivi utilizzati dalla macchina
  - ◆ gestione di code di richieste
- ◆ **Tabelle per la gestione del file system**
  - ◆ elenco dei dispositivi utilizzati per mantenere il file system
  - ◆ elenco dei file aperti e loro stato

# Introduzione

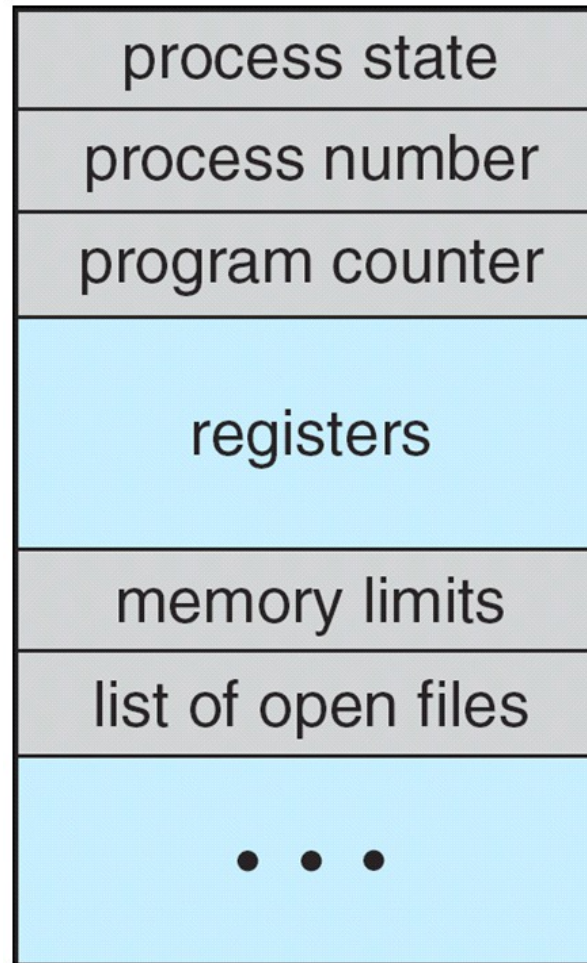


# Descrittori dei processi

---

- ♦ **Qual é la manifestazione fisica di un processo?**
  1. il codice da eseguire (segmento codice)
  2. i dati su cui operare (segmenti dati)
  3. uno stack di lavoro per la gestione di chiamate di funzione, passaggio di parametri e variabili locali
  4. un *insieme di attributi* contenenti tutte le informazioni necessarie per la gestione del processo stesso
    - ♦ incluse le informazioni necessarie per descrivere i punti 1-3
- ♦ **Questo insieme di attributi prende il nome di *descrittore del processo (process control block)***

# Descrittori dei processi



# Descrittori dei processi



- ♦ **Tabella per la gestione dei processi**
  - ♦ contiene i descrittori dei processi
  - ♦ ogni processo ha un descrittore associato
- ♦ **E' possibile suddividere le informazioni contenute nel descrittore in tre aree:**
  - ♦ informazioni di identificazione di processo
  - ♦ informazioni di stato del processo
  - ♦ informazioni di controllo del processo



# Descrittori dei processi

- ◆ **Informazioni di identificazione di un processo**
  - ◆ *identificatore di processo (process id, o pid)*
    - ◆ può essere semplicemente un *indice* all'interno di una tabella di processi
    - ◆ può essere un *numero progressivo*; in caso, è necessario un mapping tra pid e posizione del relativo descrittore
    - ◆ molte altre tabelle del s.o. utilizzano il process id per identificare un elemento della tabella dei processi
  - ◆ *identificatori di altri processi logicamente collegati al processo*
    - ◆ ad esempio, pid del processo padre
  - ◆ *id dell'utente che ha richiesto l'esecuzione del processo*

# Descrittori dei processi



- ◆ **Informazioni di stato del processo**
  - ◆ *registri generali del processore*
  - ◆ *registri speciali, come il program counter e i registri di stato*
- ◆ **Informazioni di controllo del processo**
  - ◆ *Informazioni di scheduling*
    - ◆ stato del processo
      - ◆ in esecuzione, pronto, in attesa
    - ◆ informazioni particolari necessarie dal particolare algoritmo di scheduling utilizzato
      - ◆ priorità, puntatori per la gestione delle code
    - ◆ identificatore dell'evento per cui il processo è in attesa

# Descrittori dei processi



- ◆ **Informazioni di controllo del processo (continua)**
  - ◆ informazioni di gestione della memoria
    - ◆ valori dei registri base e limite dei segmenti utilizzati, puntatori alle tabelle delle pagine, etc.
  - ◆ informazioni di accounting
    - ◆ tempo di esecuzione del processo
    - ◆ tempo trascorso dall'attivazione di un processo
  - ◆ informazioni relative alle risorse
    - ◆ risorse controllate dal processo, come file aperti, device allocati al processo
  - ◆ informazioni per interprocess communication (IPC)
    - ◆ stato di segnali, semafori, etc.

# Scheduler



- ◆ **E' la componente più importante del kernel**
- ◆ **Gestisce l'avvicendamento dei processi**
  - ◆ decide quale processo deve essere in esecuzione ad ogni istante
  - ◆ interviene quando viene richiesta un'operazione di I/O e quando un'operazione di I/O termina, ma anche periodicamente
- ◆ **NB**
  - ◆ Il termine "scheduler" viene utilizzato anche in altri ambiti con il significato di "gestore dell'avvicendamento del controllo"
  - ◆ possiamo quindi fare riferimento allo "scheduler del disco", e in generale allo "scheduler del dispositivo X"

# Schedule, scheduling, scheduler

---

- ◆ **Schedule**
  - ◆ è la sequenza temporale di assegnazioni delle risorse da gestire ai richiedenti
- ◆ **Scheduling**
  - ◆ è l'azione di calcolare uno schedule
- ◆ **Scheduler**
  - ◆ è la componente software che calcola lo schedule

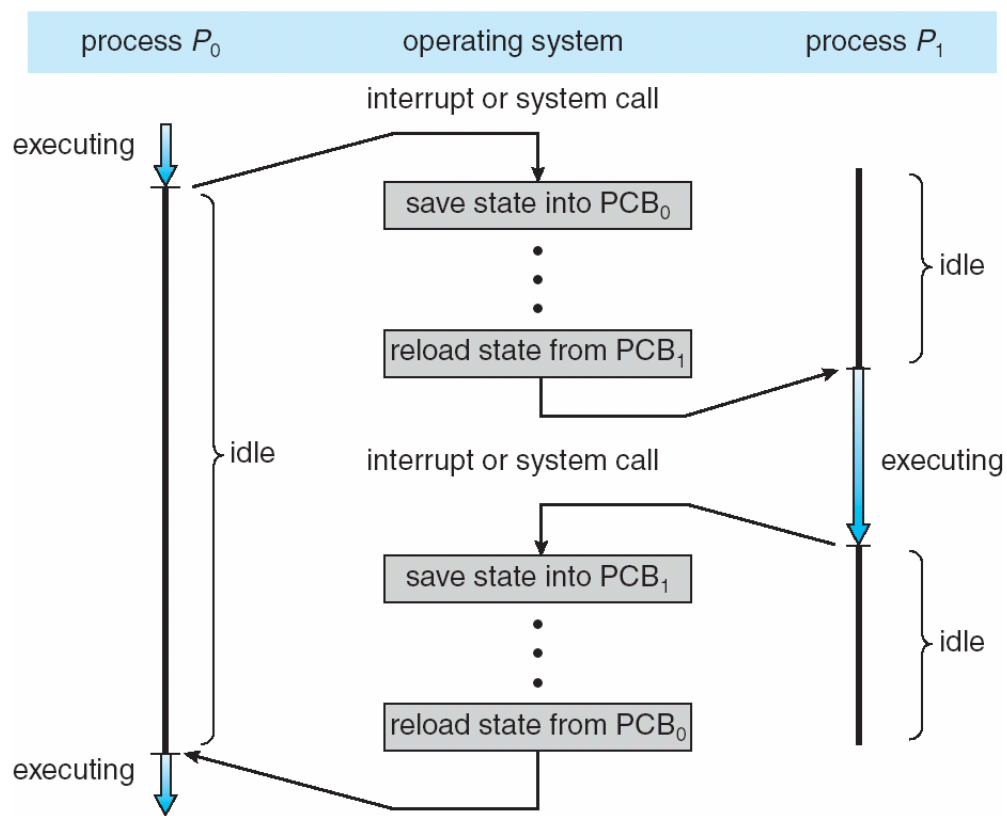
# Mode switching e context switching

---

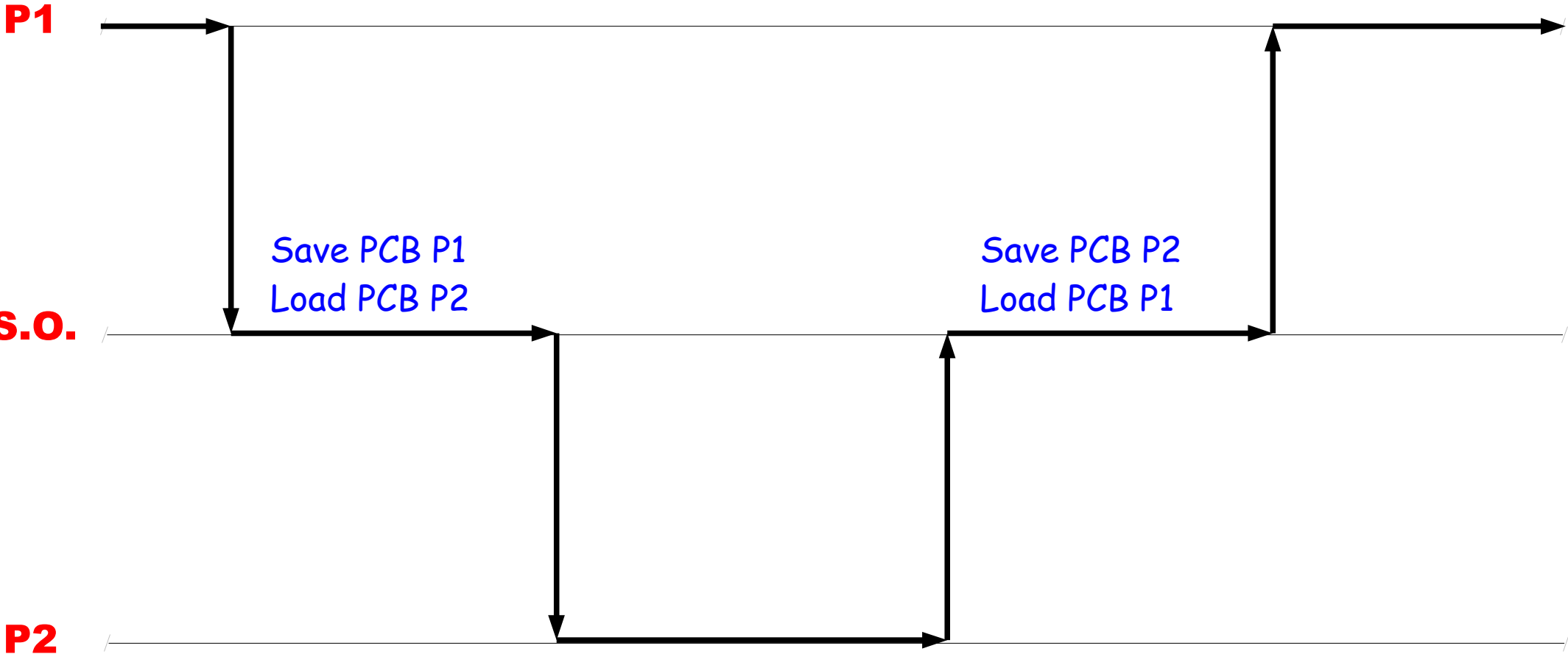
- ♦ **Come abbiamo visto nel modulo precedente**
  - ♦ tutte le volte che avviene un interrupt (software o hardware) il processore è soggetto ad un *mode switching*
    - ♦ modalità utente → modalità supervisore
- ♦ **Durante la gestione dell'interrupt**
  - ♦ vengono intraprese le opportune azioni per gestire l'evento
  - ♦ viene chiamato lo scheduler
  - ♦ se lo scheduler decide di eseguire un altro processo, il sistema è soggetto ad un *context switching*

# Context switching

- **Operazioni durante un context switching**
  - lo stato del processo attuale viene salvato nel PCB corrispondente
  - lo stato del processo selezionato per l'esecuzione viene caricato dal PCB nel processore



# Context switch

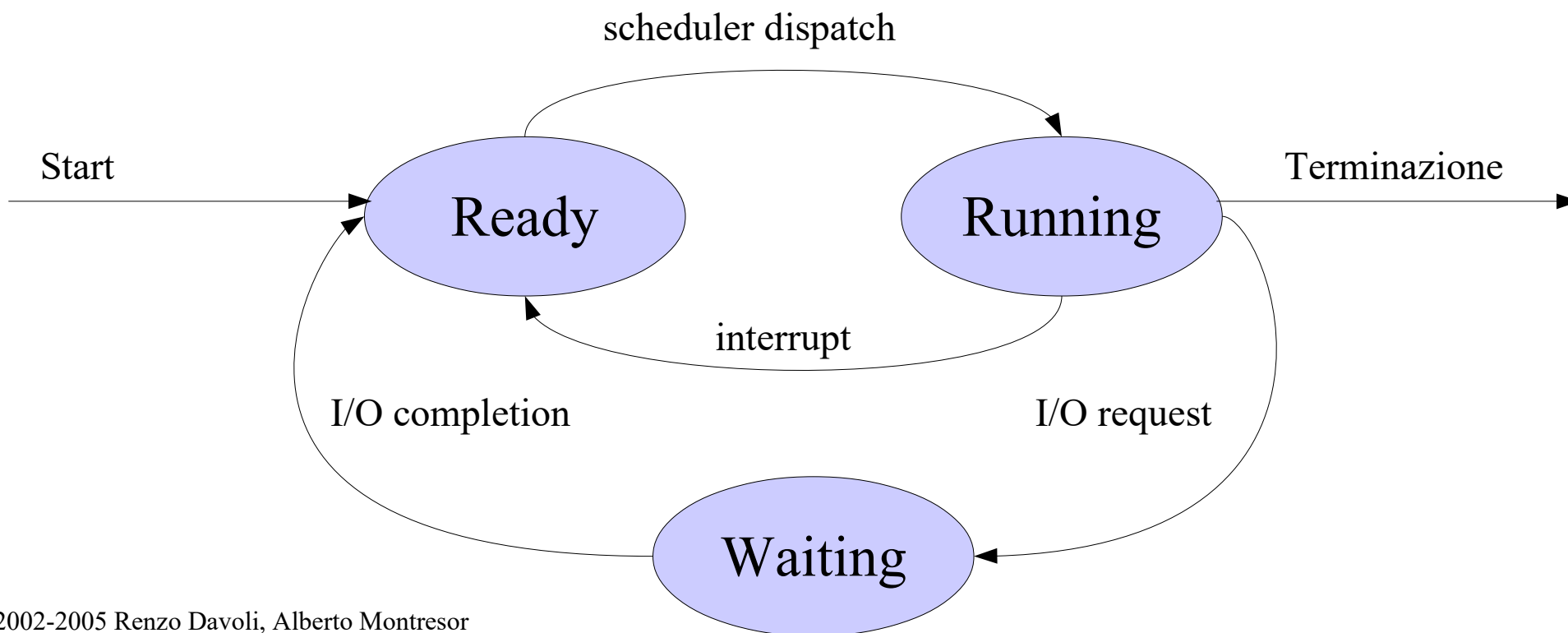




# Vita di un processo

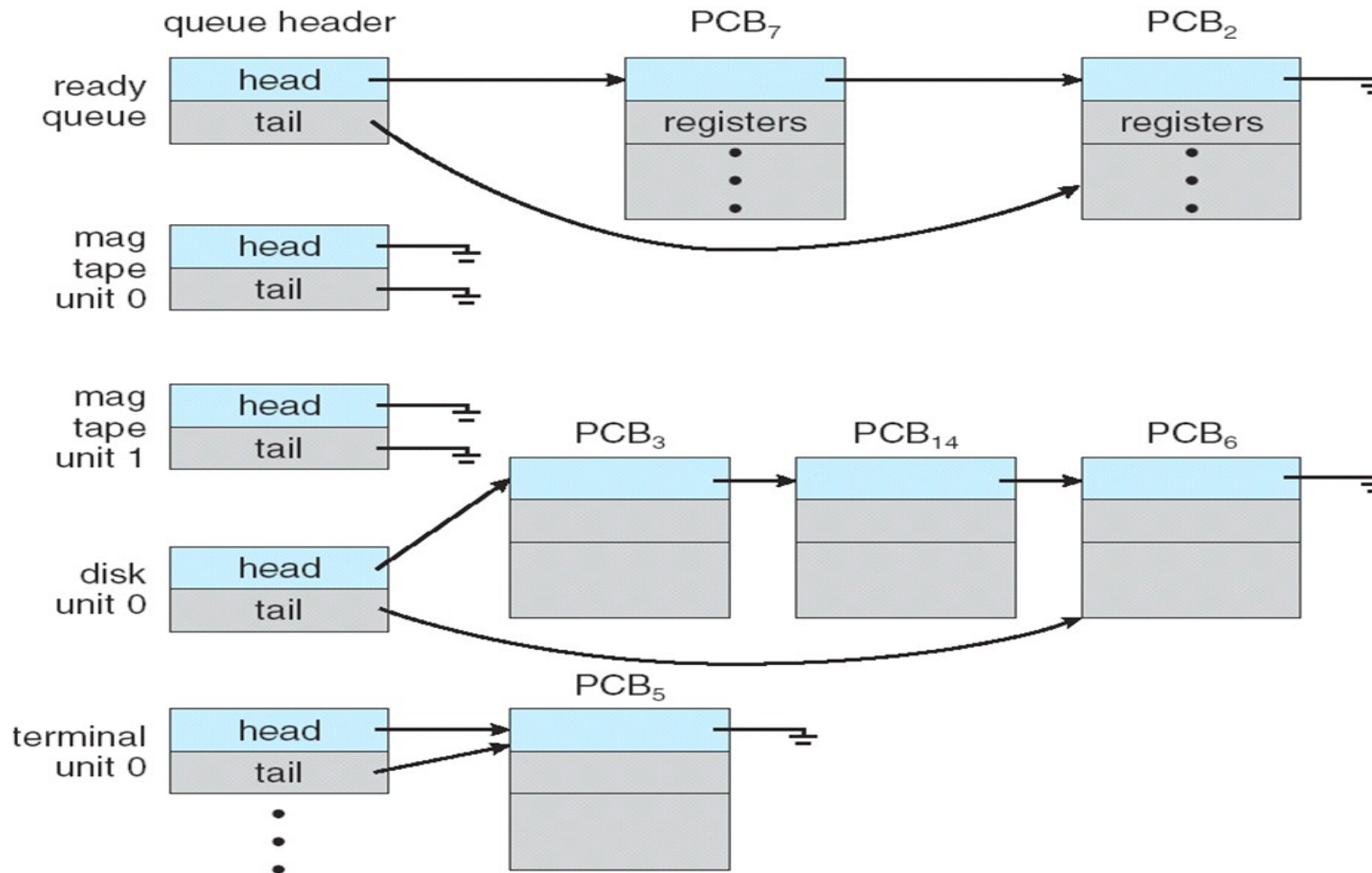
## ◆ Stati dei processi:

- ◆ *Running*: il processo è in esecuzione
- ◆ *Waiting*: il processo è in attesa di qualche evento esterno (e.g., completamento operazione di I/O); non può essere eseguito
- ◆ *Ready*: il processo può essere eseguito, ma attualmente il processore è impegnato in altre attività

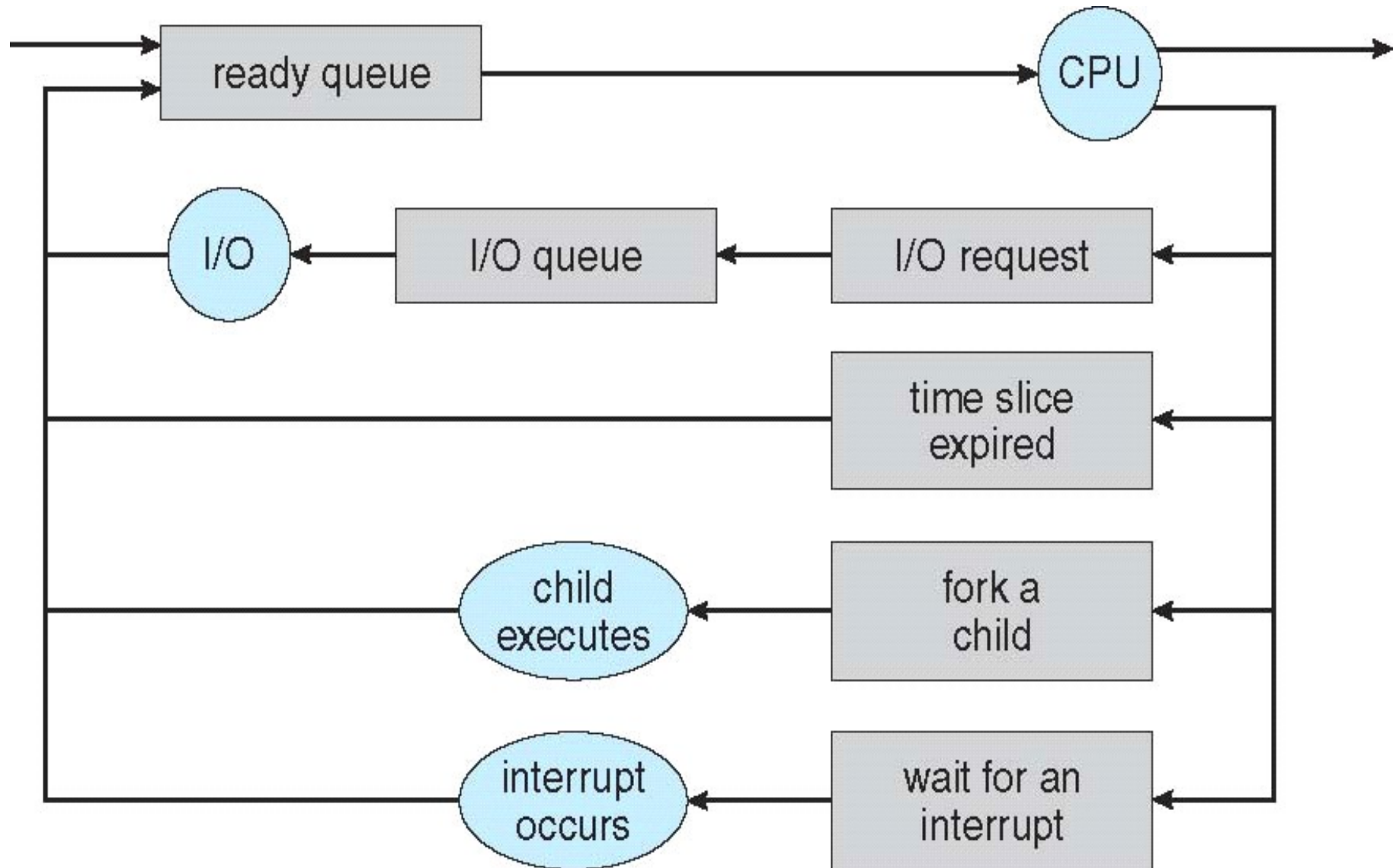


# Code di processi

- Tutte le volte che un processo entra nel sistema, viene posto in una delle code gestite dallo scheduler



# Vita di un processo nello scheduler



# Classi di scheduler



- ♦ **Long-term scheduler**

- ♦ seleziona quali processi creare fra quelli che non hanno ancora iniziato la loro esecuzione
- ♦ viene utilizzato per programmi batch
- ♦ nei sistemi interattivi (UNIX), non appena un programma viene lanciato il processo relativo viene automaticamente creato

- ♦ **Short-term scheduler**

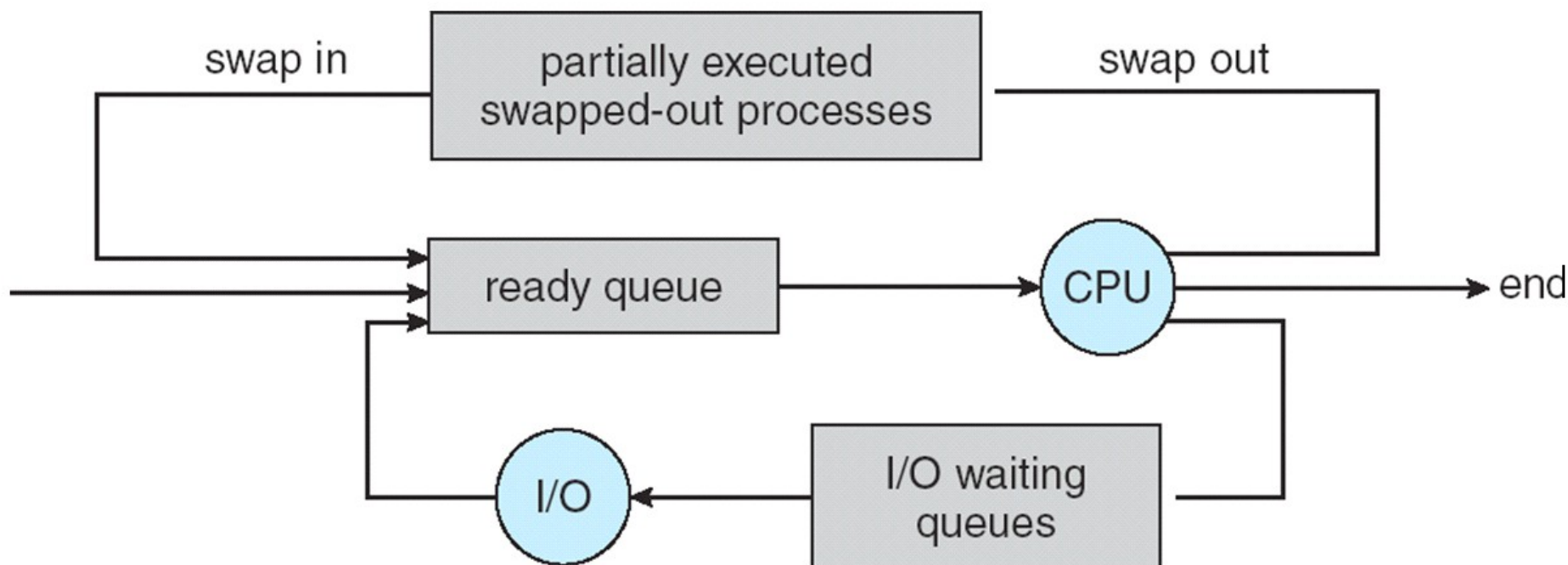
- (o scheduler di CPU, o scheduler tout-court)**

- ♦ selezione quale dei processi pronti all'esecuzione deve essere eseguito, ovvero a quale assegnare il controllo della CPU

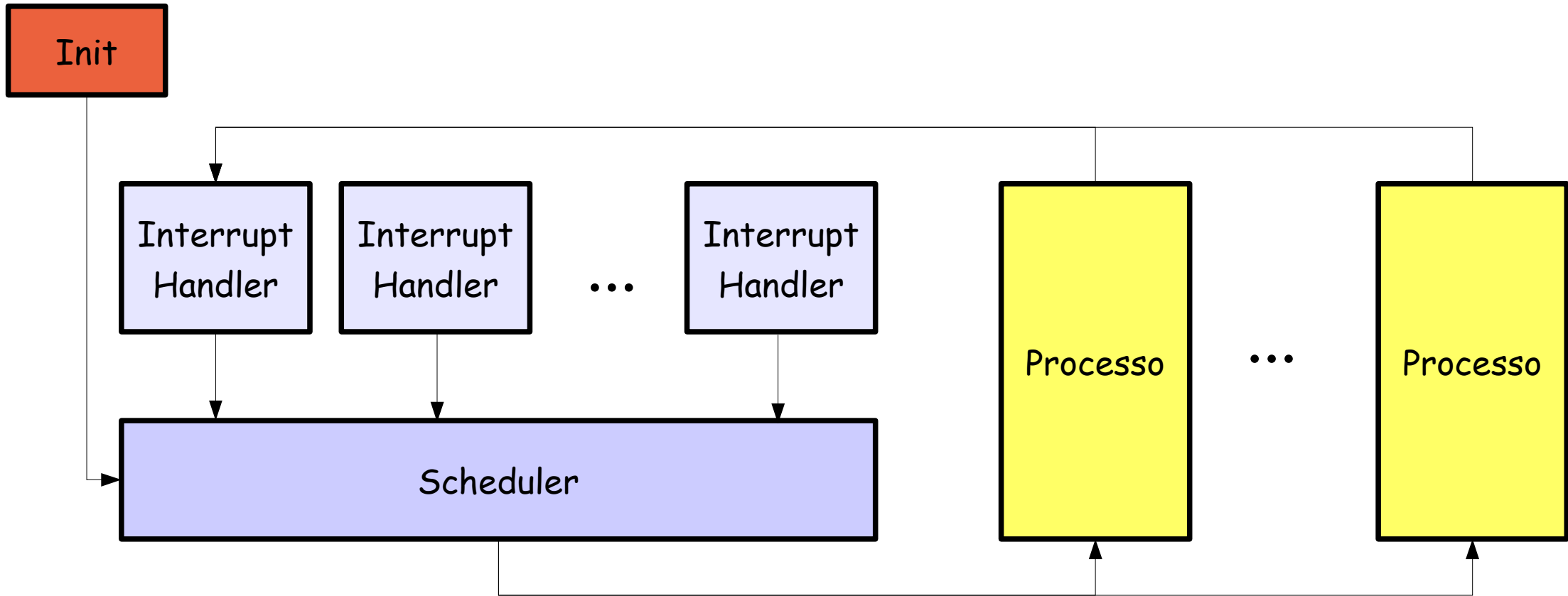
# Classi di scheduler

## ♦ Mid-term scheduler

- ♦ E' anche possibile definire uno scheduler a medio termine che gestisca i processi bloccati per lunghe attese
- ♦ per questi processi l'immagine della memoria può venire copiata su disco al fine di ottimizzare l'uso della memoria centrale



# Schema di funzionamento di un kernel

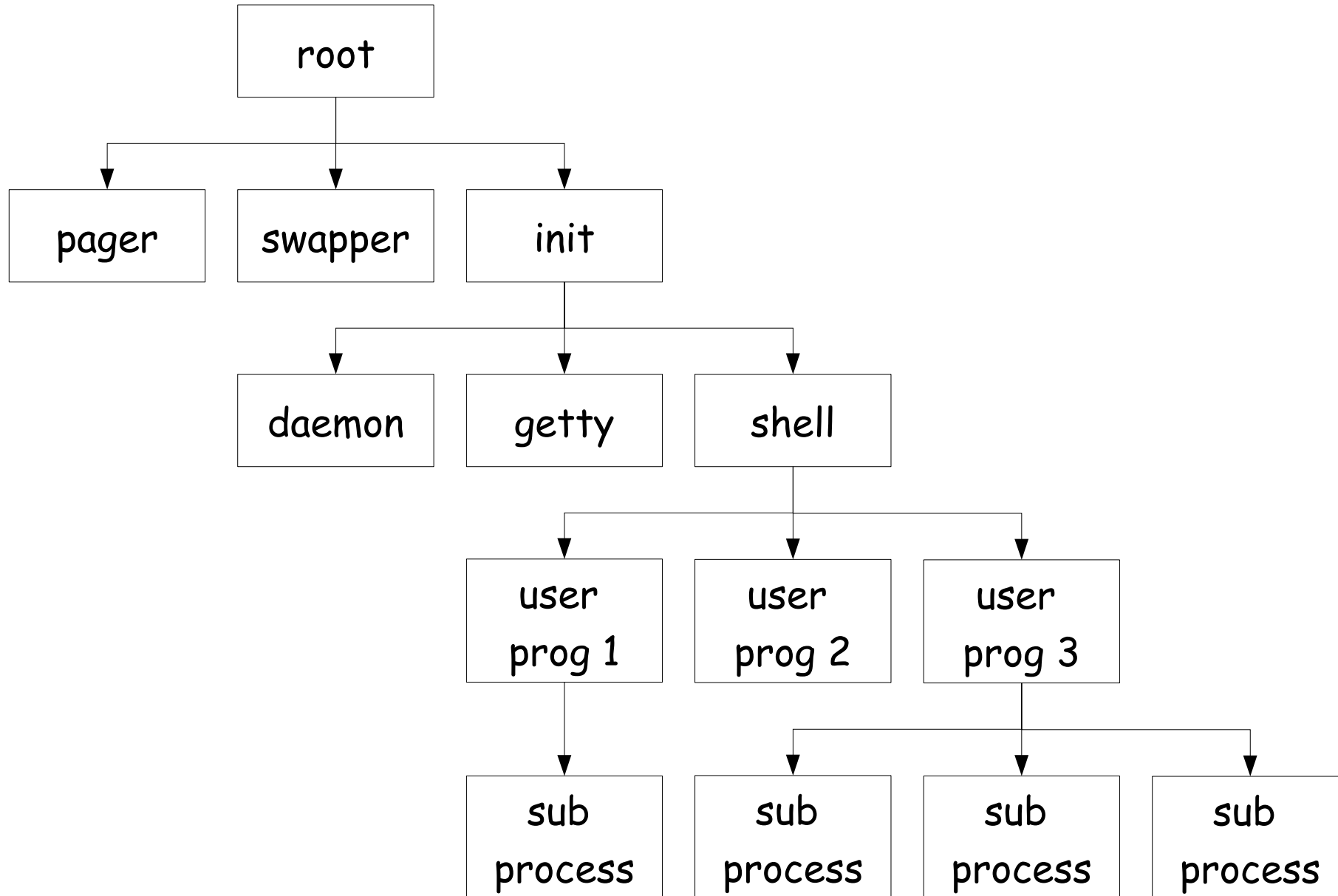


# Gerarchia di processi



- ♦ **Nella maggior parte dei sistemi operativi**
  - ♦ i processi sono organizzati in forma gerarchica
  - ♦ quando un processo crea un nuovo processo, il processo creatore viene detto padre e il creato figlio
  - ♦ si viene così a creare un albero di processi
- ♦ **Motivazioni**
  - ♦ semplificazione del procedimento di creazione di processi
    - ♦ non occorre specificare esplicitamente tutti i parametri e le caratteristiche
    - ♦ ciò che non viene specificato, viene ereditato dal padre

# Gerarchia di processi: UNIX





# Sezione 2

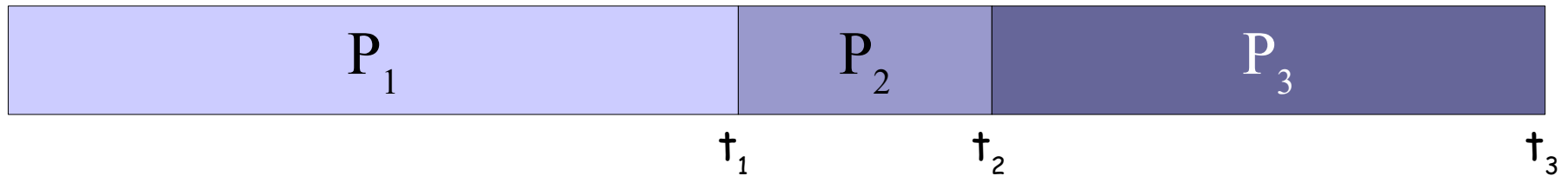


## 2. Scheduling

# Rappresentazione degli schedule

- ◆ **Diagramma di Gantt**

- ◆ per rappresentare uno schedule si usano i diagrammi di Gantt

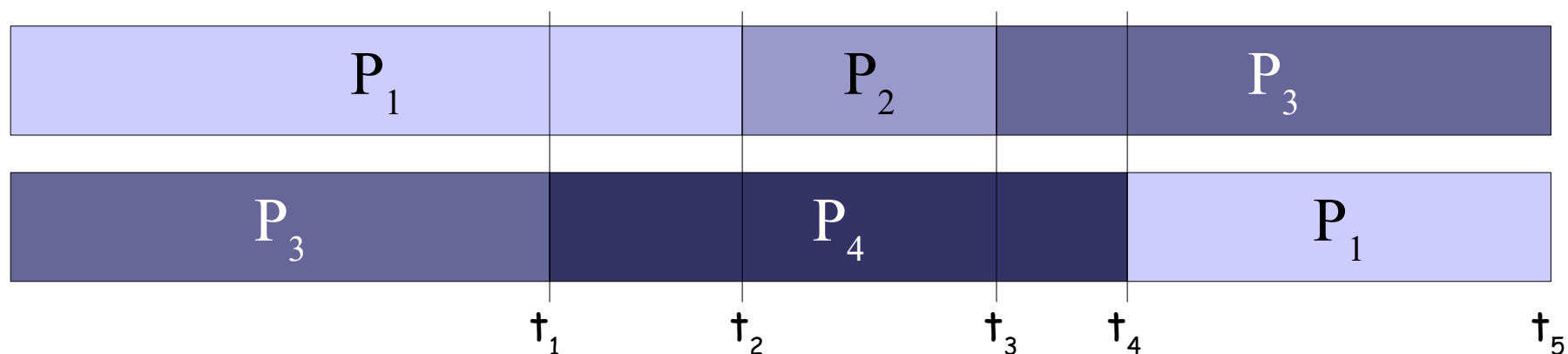


- ◆ in questo esempio, la risorsa (es. CPU) viene utilizzata dal processo  $P_1$  dal tempo 0 a  $t_1$ , viene quindi assegnata a  $P_2$  fino al tempo  $t_2$  e quindi a  $P_3$  fino al tempo  $t_3$

# Rappresentazione degli schedule

- ◆ **Diagramma di Gantt multi-risorsa**

- ◆ nel caso si debba rappresentare lo schedule di più risorse (e.g., un sistema multiprocessore) il diagramma di Gantt risulta composto da più righe parallele



# Tipi di scheduler

- ◆ **Eventi che possono causare un context switch**

1. quando un processo passa da stato running a stato waiting (system call bloccante, operazione di I/O)
2. quando un processo passa dallo stato running allo stato ready (a causa di un interrupt)
3. quando un processo passa dallo stato waiting allo stato ready
4. quando un processo termina

- ◆ **Nota:**

- ◆ nelle condizioni 1 e 4, l'unica scelta possibile è quella di selezionare un altro processo per l'esecuzione
- ◆ nelle condizioni 2 e 3, è possibile continuare ad eseguire il processo corrente

# Tipi di scheduler

---

- ♦ **Uno scheduler si dice *non-preemptive* o *cooperativo***
  - ♦ se i context switch avvengono solo nelle condizioni 1 e 4
  - ♦ in altre parole: *il controllo della risorsa viene trasferito solo se l'assegnatario attuale lo cede volontariamente*
  - ♦ Windows 3.1, Mac OS y con  $y < 8$
- ♦ **Uno scheduler si dice *preemptive* se**
  - ♦ se i context switch possono avvenire in ogni condizione
  - ♦ in altre parole: *è possibile che il controllo della risorsa venga tolto all'assegnatario attuale a causa di un evento*
  - ♦ tutti gli scheduler moderni

# Tipi di scheduler



- ♦ **Vantaggi dello scheduling cooperativo**
  - ♦ non richiede alcuni meccanismi hardware come ad esempio timer programmabili
- ♦ **Vantaggi dello scheduling preemptive**
  - ♦ permette di utilizzare al meglio le risorse

# Criteri di scelta di uno scheduler

---

- ♦ **Utilizzo della risorsa (CPU)**
  - ♦ percentuale di tempo in cui la CPU è occupata ad eseguire processi
  - ♦ deve essere massimizzato
- ♦ **Throughput**
  - ♦ numero di processi completati per unità di tempo
  - ♦ dipende dalla lunghezza dei processi
  - ♦ deve essere massimizzato
- ♦ **Tempo di turnaround**
  - ♦ tempo che intercorre dalla sottomissione di un processo alla sua terminazione
  - ♦ deve essere minimizzato

# Criteri di scelta di uno scheduler

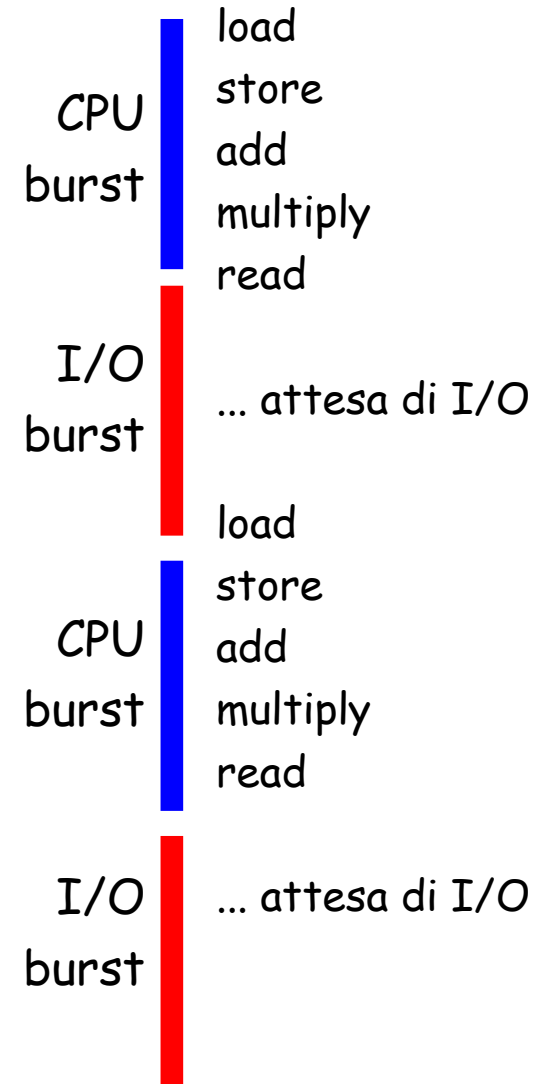


- ♦ **Tempo di attesa**
  - ♦ il tempo trascorso da un processo nella coda ready
  - ♦ deve essere minimizzato
- ♦ **Tempo di risposta**
  - ♦ tempo che intercorre fra la sottomissione di un processo e il tempo di prima risposta
  - ♦ particolarmente significativo nei programmi interattivi, deve essere minimizzato



# Caratteristiche dei processi

- **Durante l'esecuzione di un processo:**
  - si alternano periodi di attività svolte dalla CPU (*CPU burst*)...
  - ...e periodi di attività di I/O (*I/O burst*)
- **I processi:**
  - caratterizzati da CPU burst molto lunghi si dicono *CPU bound*
  - caratterizzati da I/O burst molto lunghi si dicono *I/O bound*



# Scheduling



- ♦ **Algoritmi:**
  - ♦ First Come, First Served
  - ♦ Shortest-Job First
    - ♦ Shortest-Next-CPU-Burst First
    - ♦ Shortest-Remaining-Time-First
  - ♦ Round-Robin

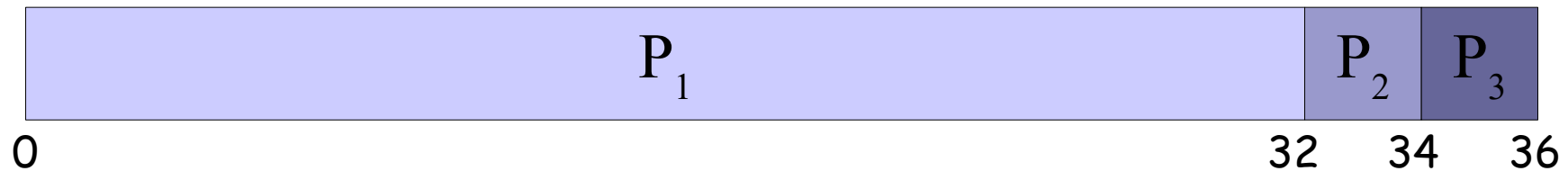
# First Come, First Served (FCFS)



- ♦ **Algoritmo**
  - ♦ il processo che arriva per primo, viene servito per primo
  - ♦ politica senza preemption
- ♦ **Implementazione**
  - ♦ semplice, tramite una coda (politica FIFO)
- ♦ **Problemi**
  - ♦ elevati tempi medi di attesa e di turnaround
  - ♦ processi CPU bound ritardano i processi I/O bound

# First Come, First Served (FCFS)

- ♦ **Esempio:**
  - ♦ ordine di arrivo:  $P_1, P_2, P_3$
  - ♦ lunghezza dei CPU-burst in ms: 32, 2, 2
  - ♦ Tempo medio di turnaround  
 $(32+34+36)/3 = 34$  ms
  - ♦ Tempo medio di attesa:  
 $(0+32+34)/3 = 22$  ms



# First Come, First Served (FCFS)

- Primo arrivato, primo servito (gestito con coda FIFO).

<u>Processo</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

- Supponiamo che i processi arrivino nell'ordine:  $P_1$ ,  $P_2$ ,  $P_3$   
Lo schema di Gantt è:

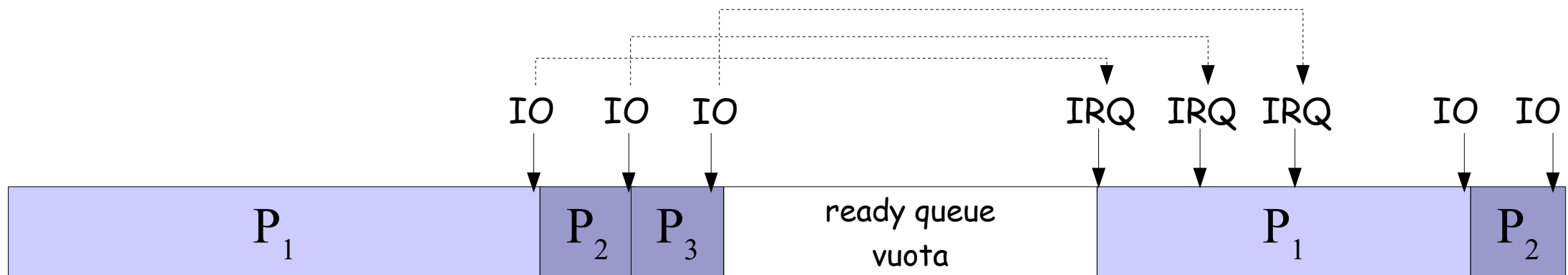


- Tempo di waiting per:  $P_1 = 0$ ;  $P_2 = 24$ ;  $P_3 = 27$
- Tempo di waiting medio:  $(0 + 24 + 27)/3 = 17$

# First Come, First Served (FCFS)

- ◆ **Supponiamo di avere**

- ◆ un processo CPU bound
- ◆ un certo numero di processi I/O bound
- ◆ i processi I/O bound si "mettono in coda" dietro al processo CPU bound, e in alcuni casi la ready queue si puo svuotare
- ◆ *Convoy effect*

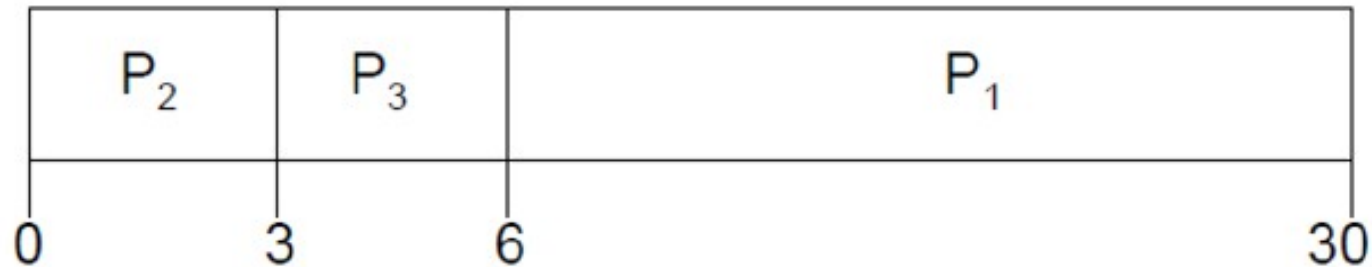


# First Come, First Served (FCFS)

Supponiamo che i processi arrivino nell'ordine

$$P_2, P_3, P_1.$$

- Lo schema di Gantt è:



- Tempo di waiting per  $P_1 = 6; P_2 = 0; P_3 = 3$
- Tempo di waiting medio:  $(6 + 0 + 3)/3 = 3$
- Molto meglio che nel caso precedente.
- *Effetto convoglio*: i processi "brevi" attendono i processi "lunghi".

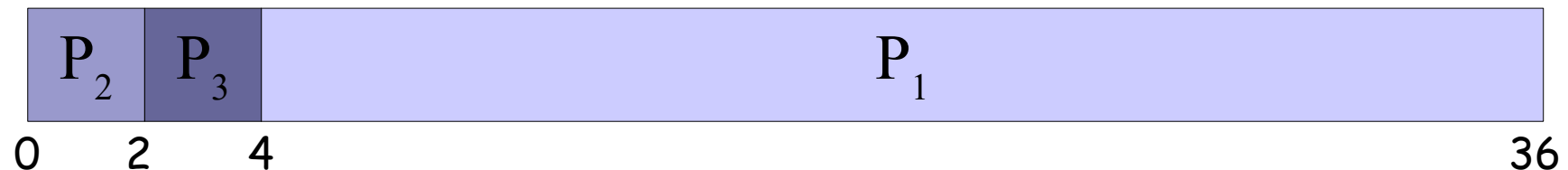
# Shortest Job First (SJF)

- ♦ **Algoritmo (Shortest Next CPU Burst First)**

- ♦ la CPU viene assegnata al processo ready che ha la minima durata del CPU burst successivo

- ♦ **Esempio**

- ♦ Tempo medio di turnaround:  $(0+2+4+36)/3 = 7$  ms
- ♦ Tempo medio di attesa:  $(0+2+4)/3 = 2$  ms





# Shortest Job First (SJF)

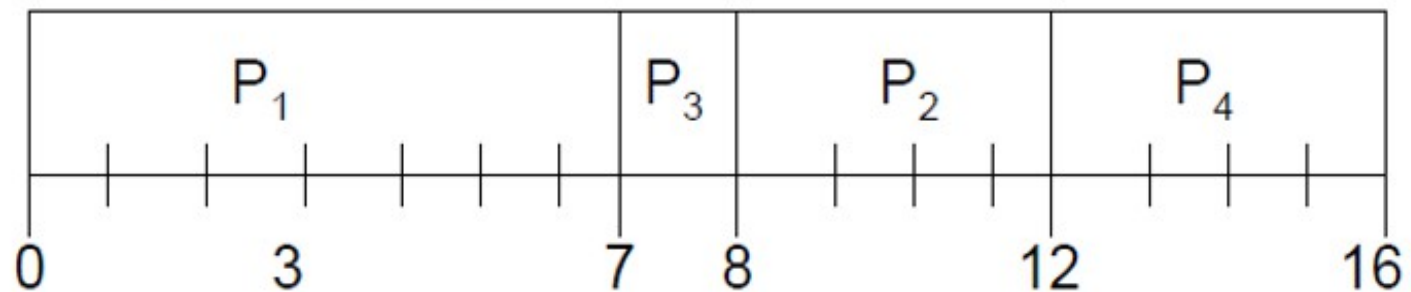
---

- Associa ad ogni processo la lunghezza del prossimo CPU burst. Use questi tempi per schedulare il processo con la lunghezza minima.
- Due schemi:
  - ***nonpreemptive*** – il processo assegnato non può essere sospeso prima di completare il suo CPU burst.
  - ***preemptive*** – se arriva un nuovo processo con un CPU burst più breve del tempo rimanente per il processo corrente, viene servito. Questo schema è conosciuto come ***Shortest-Remaining-Time-First (SRTF)***.
- *SJF è ottimale* – offre il minimo tempo medio di attesa per un insieme di processi.

# Shortest Job First (SJF) NON PREEMPTIVE

<u>Processo</u>	<u>Tempo Arrivo</u>	<u>Tempo di 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)

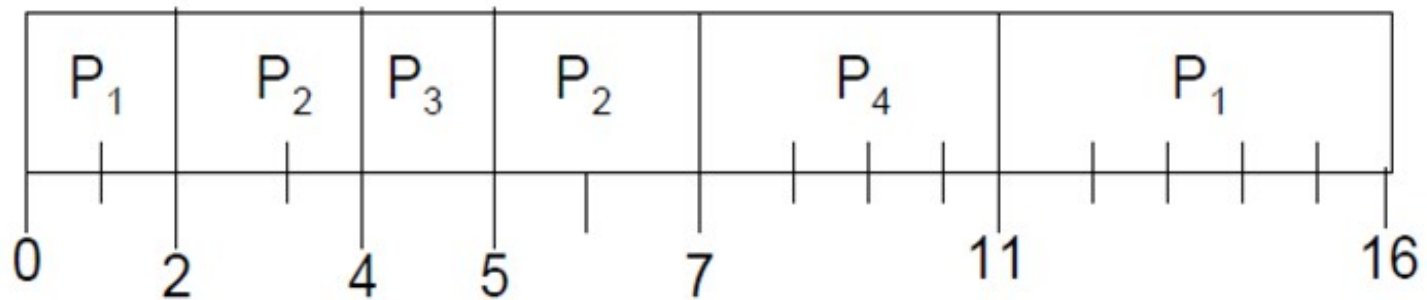


■ Tempo di attesa medio =  $(0 + 6 + 3 + 7)/4 = 4$

# Shortest Job First (SJF) PREEMPTIVE

<u>Processo</u>	<u>Tempo di Arrivo</u>	<u>Tempo di 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 = $(9 + 1 + 0 + 2)/4 = 3$

# Shortest Job First (SJF)



- ♦ **L'algoritmo SJF**

- ♦ è *ottimale* rispetto al tempo di attesa, in quanto è possibile dimostrare che produce il minor tempo di attesa possibile
- ♦ ma è *impossibile da implementare* in pratica!
- ♦ è possibile solo fornire delle *approssimazioni*

- ♦ **Negli scheduler long-term**

- ♦ possiamo chiedere a chi sottomette un job di predire la durata del job

- ♦ **Negli scheduler short-term**

- ♦ non possiamo conoscere la lunghezza del *prossimo* CPU burst... ma conosciamo la lunghezza di quelli *precedenti*

# Shortest Job First (SJF)

- ♦ **Calcolo approssimato della durata del CPU burst**

- ♦ basata su *media esponenziale* dei CPU burst precedenti
- ♦ sia  $t_n$  il tempo dell' $n$ -esimo CPU burst e  $\tau_n$  la corrispondente previsione;  $\tau_{n+1}$  può essere calcolato come segue:

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

- ♦ **Media esponenziale**

- ♦ svolgendo la formula di ricorrenza, si ottiene

$$\tau_{n+1} = \sum_{j=0..n} (1-\alpha)^j t_{n-j} + (1-\alpha)^{n+1} \tau_0$$

da cui il nome media esponenziale

# Shortest Job First (SJF)



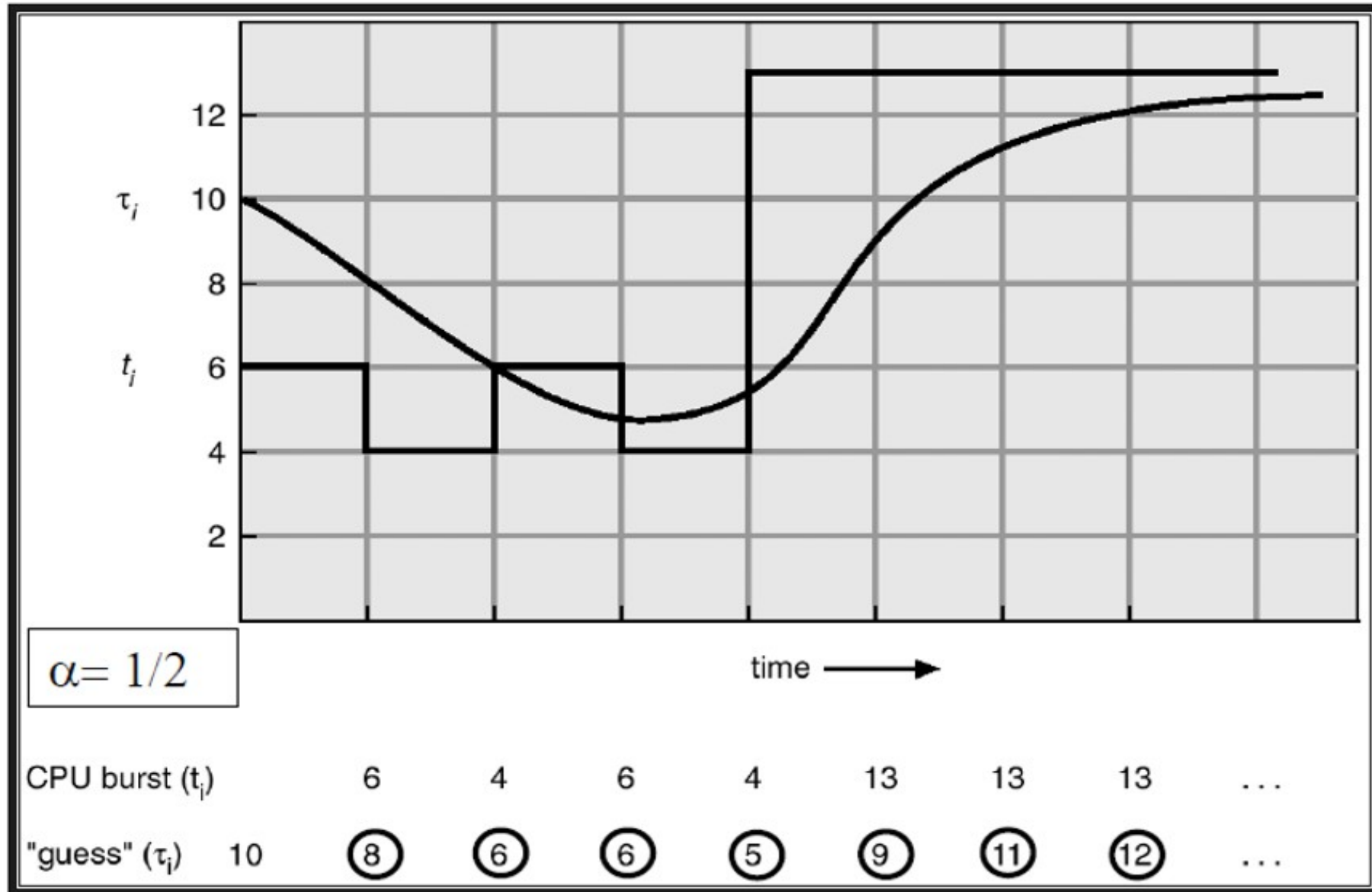
- ◆ **Spiegazione**

- ◆  $t_n$  rappresenta la storia recente
- ◆  $\tau_n$  rappresenta la storia passata
- ◆  $\alpha$  rappresenta il peso relativo di storia passata e recente
- ◆ cosa succede con  $\alpha = 0, 1$  oppure  $\frac{1}{2}$ ?

- ◆ **Nota importante:**

- ◆ SJF (preemptive) può essere soggetto a starvation!

# Shortest Job First (SJF) – predizione del prossimo CPU burst



# Shortest Job First (SJF)



- ♦ Shortest Job First "approssimato" esiste in due versioni:
  - ♦ *non preemptive*
    - ♦ il processo corrente esegue fino al completamento del suo CPU burst
  - ♦ *preemptive*
    - ♦ il processo corrente può essere messo nella coda ready, se arriva un processo con un CPU burst più breve di quanto rimane da eseguire al processo corrente
    - ♦ "Shortest-Remaining-Time First"



# Scheduling Round-Robin



- ◆ **E' basato sul concetto di quanto di tempo (o time slice)**
  - ◆ un processo non può rimanere in esecuzione per un tempo superiore alla durata del quanto di tempo
- ◆ **Implementazione (1)**
  - ◆ l'insieme dei processi pronti è organizzato come una coda
  - ◆ due possibilità:
    - ◆ un processo può lasciare il processore *volontariamente*, in seguito ad un'operazione di I/O
    - ◆ un processo può *esaurire il suo quanto di tempo* senza completare il suo CPU burst, nel qual caso viene aggiunto in fondo alla coda dei processi pronti
  - ◆ in entrambi i casi, il prossimo processo da eseguire è il primo della coda dei processi pronti

# Scheduling Round-Robin

## ♦ Implementazione (2)

- ♦ è necessario che l'hardware fornisca un timer (interval timer) che agisca come "sveglia" del processore
- ♦ il timer è un dispositivo che, attivato con un preciso valore di tempo, è in grado di fornire un interrupt allo scadere del tempo prefissato
- ♦ il timer viene interfacciato come se fosse un'unità di I/O

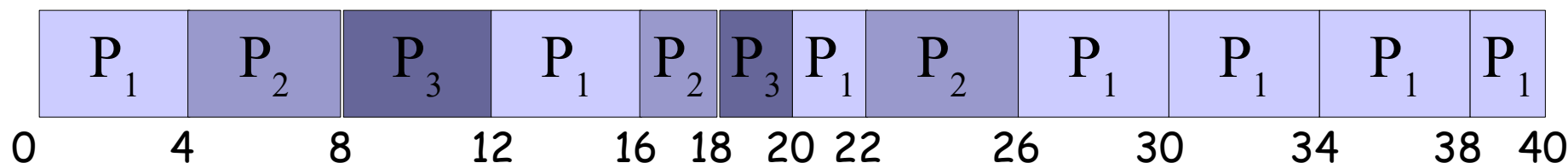
## ♦ La durata del quanto di tempo è un parametro critico del sistema

- ♦ se il quanto di tempo è breve, il sistema è meno efficiente perchè deve cambiare il processo attivo più spesso
- ♦ se il quanto è lungo, in presenza di numerosi processi pronti ci sono lunghi periodi di inattività di ogni singolo processo,
  - ♦ in sistemi interattivi, questo può essere fastidioso per gli utenti

# Scheduling Round-Robin

## ♦ Esempio

- ♦ tre processi  $P_1$ ,  $P_2$ ,  $P_3$
- ♦ lunghezza dei CPU-burst in ms ( $P_1$ : 10+14;  $P_2$ : 6+4;  $P_3$ : 6)
- ♦ lunghezza del quanto di tempo: 4
- ♦ Tempo medio di turnaround  $(40+26+20)/3 = 28.66$  ms
- ♦ Tempo medio di attesa:  $(16+16+14)/3 = 15.33$  ms
  - ♦ NB (supponiamo attese di I/O brevi,  $< 2$ ms)
- ♦ Tempo medio di risposta: 4 ms



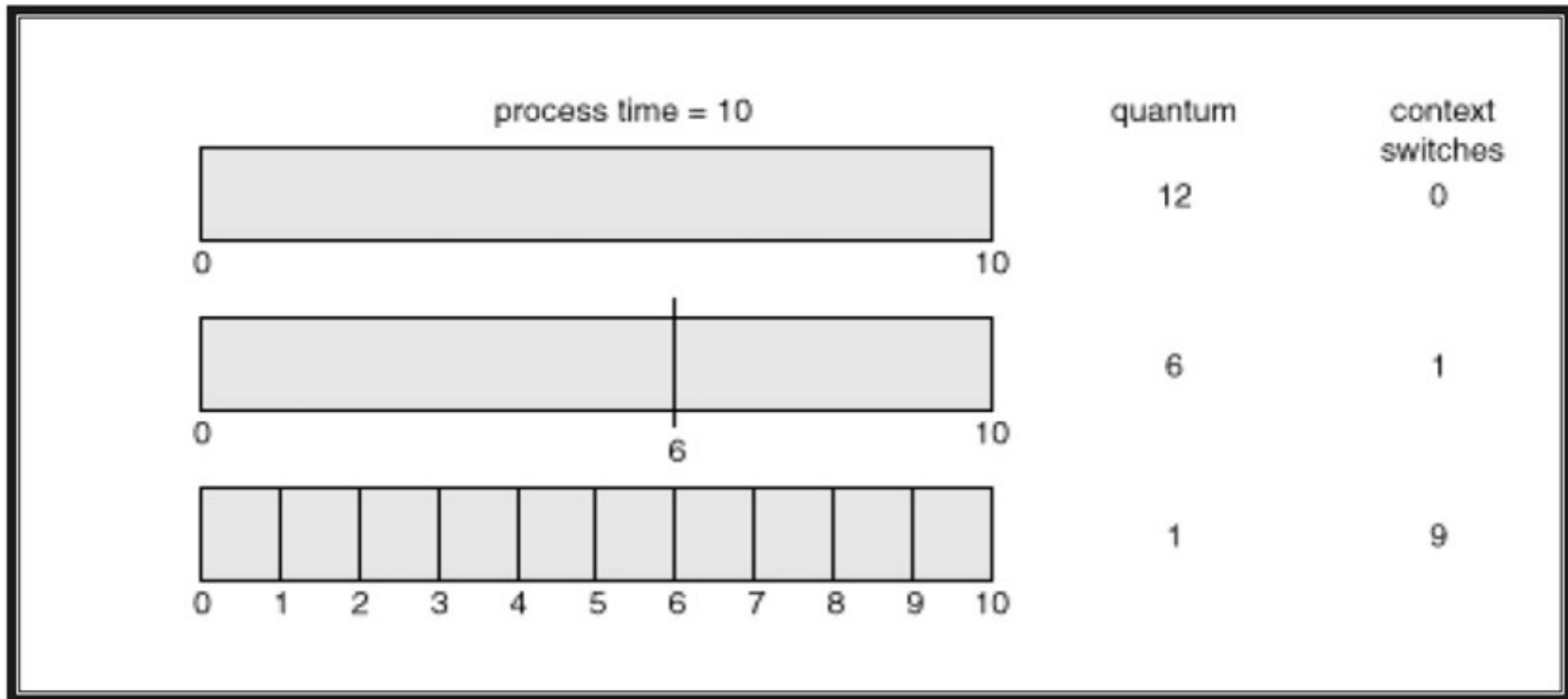
# Scheduling Round-Robin

---

- Se ci sono **N** processi nella ready queue e il quanto di tempo è **Q**, ogni processo ottiene **1/N** del tempo della CPU a blocchi di lunghezza **Q**. Nessun processo attende più di **(N-1)Q** unità di tempo.
- Prestazioni
  - *Q* grande  $\Rightarrow$  FIFO
  - *Q* piccolo  $\Rightarrow$  *Q* deve essere molto più grande del tempo di *context switch*, altrimenti il costo è troppo alto.

# Scheduling Round-Robin

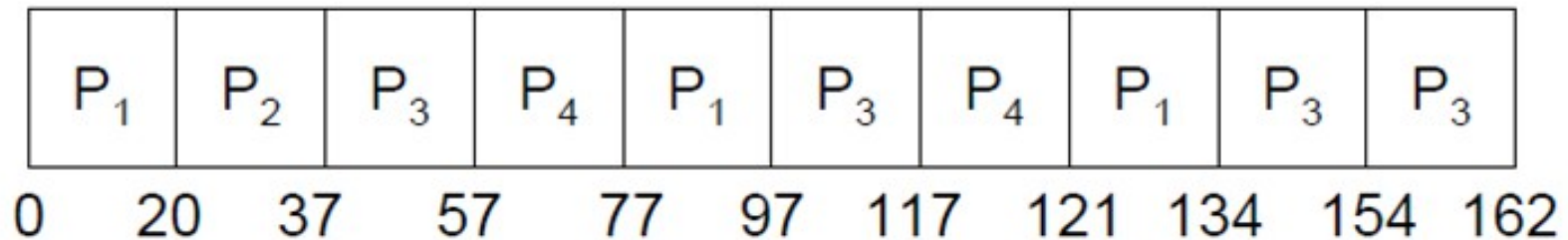
- Il quanto di tempo deve essere molto più grande del tempo di context switch, altrimenti il costo è troppo alto.



# Scheduling Round-Robin – esempio con $Q=20$

<u>Processi</u>	<u>tempo di burst</u>
$P_1$	53
$P_2$	17
$P_3$	68
$P_4$	24

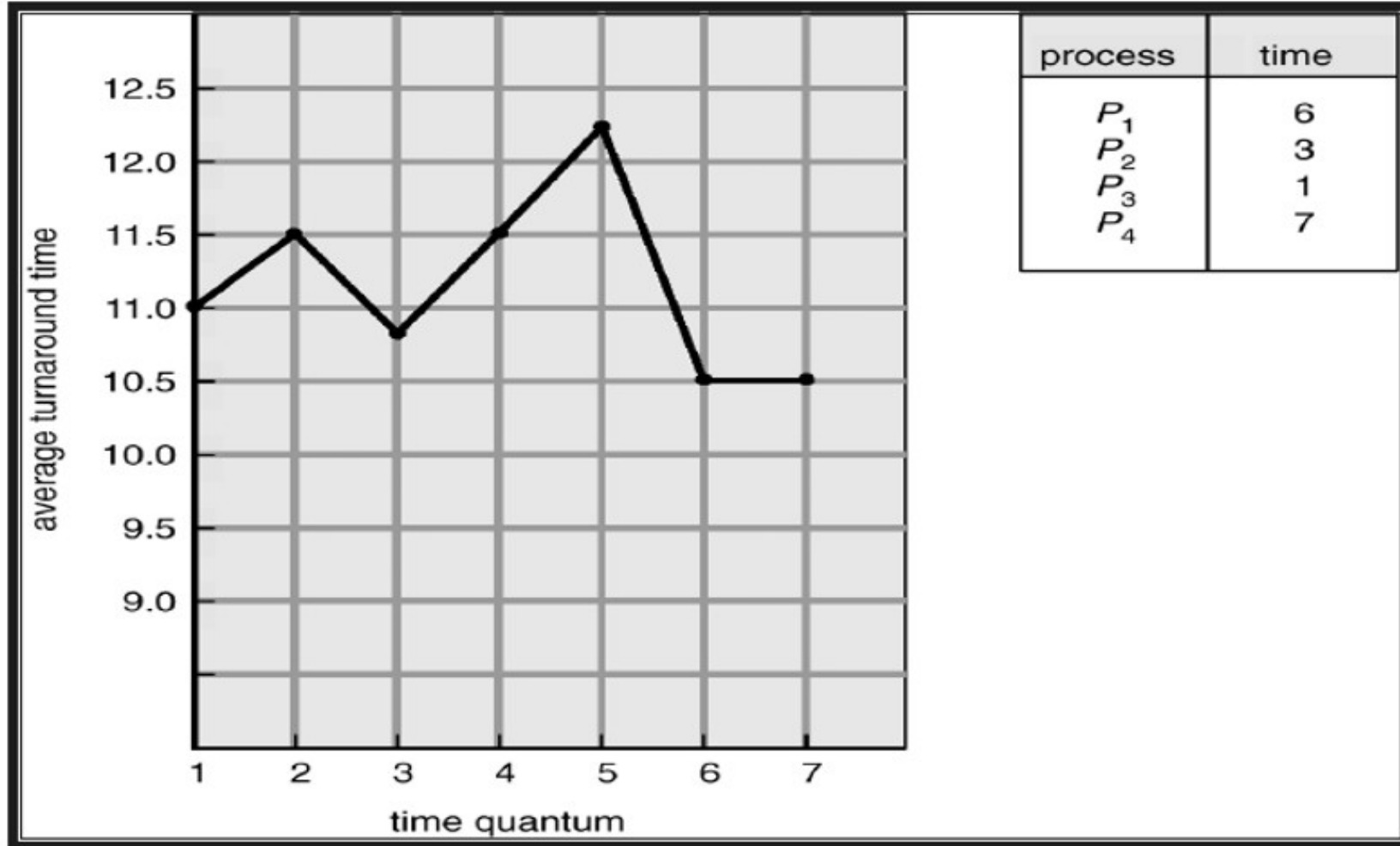
■ Gantt:



- Tempo di *turnaround* maggiore di SJF, ma migliore *tempo di risposta*.

# Scheduling Round-Robin – Tempo di turnaround vs. Q

- Anche il tempo di Turnaround dipende dal quanto di tempo



- Circa l'80% dei CPU burst devono essere più brevi di Q.

# Scheduling a priorità



- ♦ **Round-robin**
  - ♦ fornisce le stesse possibilità di esecuzione a tutti i processi
- ♦ **Ma i processi non sono tutti uguali:**
  - ♦ usando round-robin puro la visualizzazione di un video MPEG potrebbe essere ritardata da un processo che sta smistando la posta
  - ♦ la lettera può aspettare  $\frac{1}{2}$  sec, il frame video NO



# Scheduling a priorità

---

- ◆ **Descrizione**

- ◆ ogni processo è associato una specifica priorità
- ◆ lo scheduler sceglie il processo pronto con priorità più alta

- ◆ **Le priorità possono essere:**

- ◆ *definite dal sistema operativo*

- ◆ vengono utilizzate una o più quantità misurabili per calcolare la priorità di un processo
- ◆ esempio: SJF è un sistema basato su priorità

- ◆ *definite esternamente*

- ◆ le priorità non vengono definite dal sistema operativo, ma vengono imposte dal livello utente

# Scheduling a priorità



- ◆ **Priorità statica**

- ◆ la priorità non cambia durante la vita di un processo
- ◆ problema: processi a bassa priorità possono essere posti in *starvation* da processi ad alta priorità
- ◆ Leggenda metropolitana: IBM 7094

- ◆ **Priorità dinamica**

- ◆ la priorità può variare durante la vita di un processo
- ◆ è possibile utilizzare metodologie di priorità dinamica per evitare *starvation*

# Scheduling a priorità



- ◆ **Priorità basata su aging**

- ◆ tecnica che consiste nell'incrementare gradualmente la priorità dei processi in attesa
- ◆ posto che il range di variazione delle priorità sia limitato, nessun processo rimarrà in attesa per un tempo indefinito perché prima o poi raggiungerà la priorità massima

# Scheduling a classi di priorità



## ♦ Descrizione

- ♦ e' possibile creare diverse classi di processi con caratteristiche simili e assegnare ad ogni classe specifiche priorità
- ♦ la coda ready viene quindi scomposta in molteplici "sottocode", una per ogni classe di processi

## ♦ Algoritmo

- ♦ uno scheduler a classi di priorità seleziona il processo da eseguire fra quelli pronti della classe a priorità massima che contiene processi

# Scheduling Multilivello

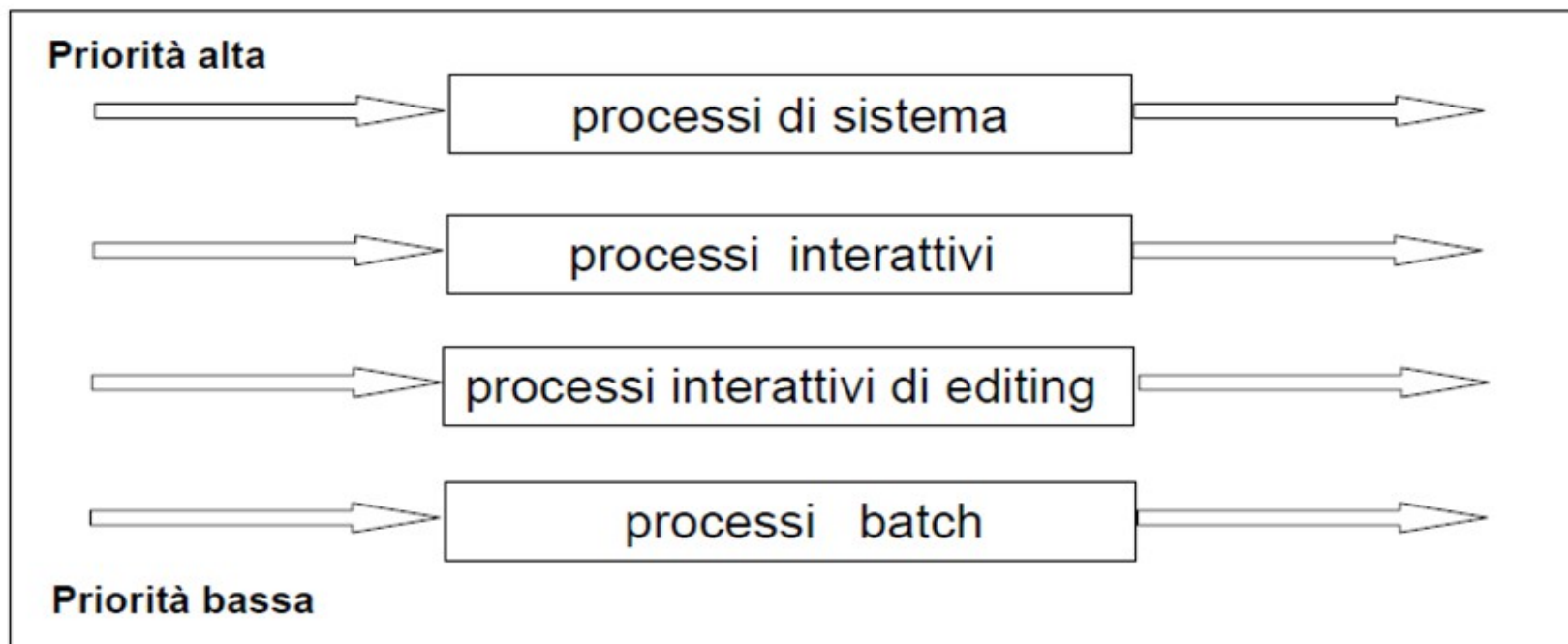


- ◆ **Descrizione**

- ◆ all'interno di ogni classe di processi, è possibile utilizzare una politica specifica adatta alle caratteristiche della classe
- ◆ uno scheduler multilivello cerca prima la classe di priorità massima che ha almeno un processo ready
- ◆ sceglie poi il processo da porre in stato running coerentemente con la politica specifica della classe

# Scheduling Multilivello - Esempio

- ◆ **Quattro classi di processi (priorità decrescente)**
  - ◆ processi server (priorità statica)
  - ◆ processi utente interattivi (round-robin)
  - ◆ altri processi utente (FIFO)
  - ◆ Processi batch (FIFO)



# Scheduling Multilivello con Feedback

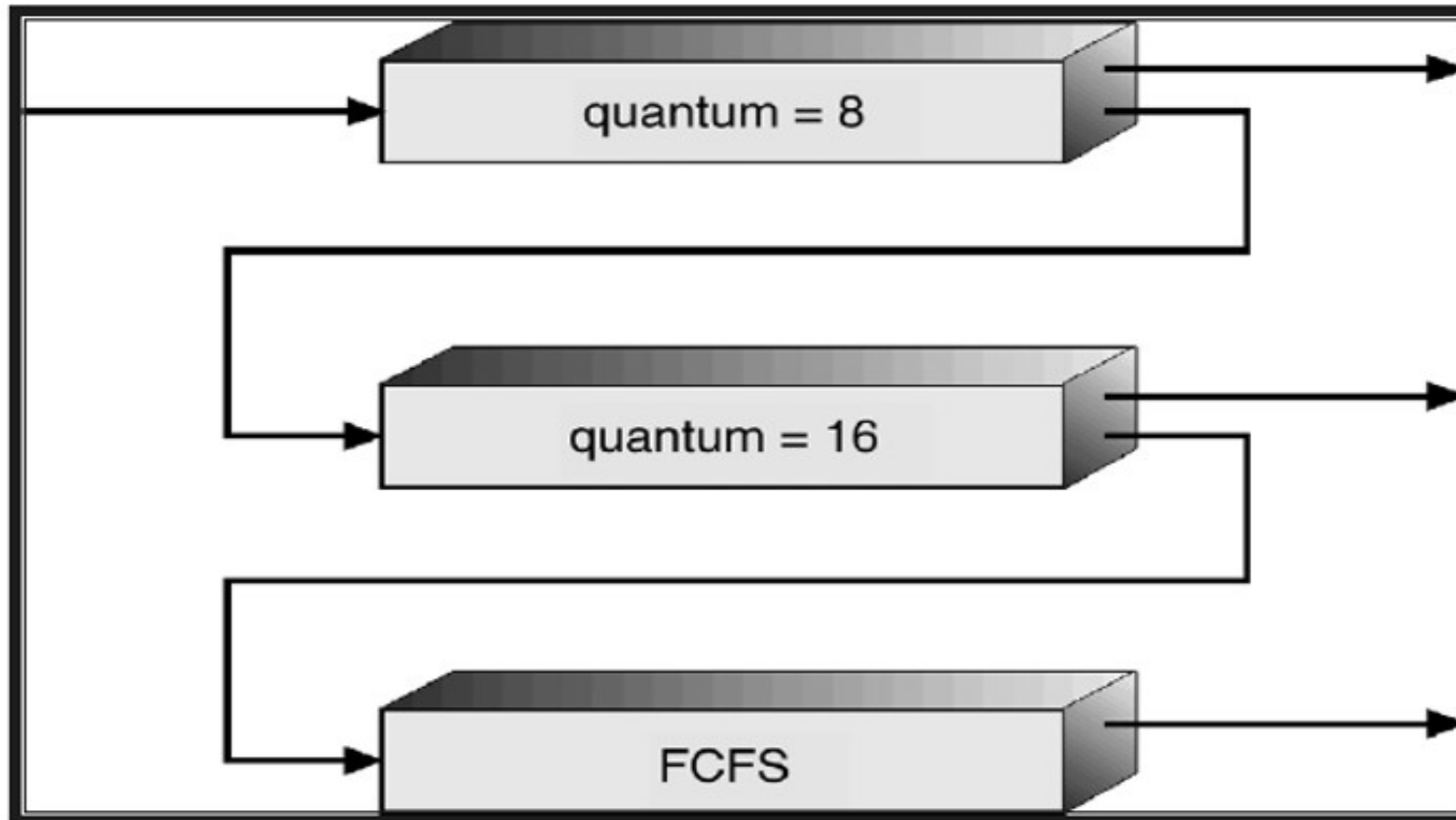
---

- Un processo si può spostare tra le code. Questo evita situazioni di starvation o di eccessivo utilizzo della CPU.
- Lo scheduler che usa code multiple con feedback usa i seguenti parametri:
  - numero di code
  - algoritmi di scheduling per ogni coda
  - un metodo per "promuovere" un processo (--> maggiore priorità)
  - un metodo per "degradare" un processo (--> minore priorità)
  - un metodo per decidere in quale coda inserire un processo quando questo chiede un servizio.

# Scheduling Multilivello con Feedback

## ■ Esempio con tre code:

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





# Scheduling Multilivello con Feedback

- Si servono prima i processi della coda 0, quindi quelli della coda 1 e solo dopo quelli della coda 2.
- Esempio di scheduling
  - un nuovo processo arriva nella coda  $Q_0$  e quando è servito dalla CPU gli viene assegnato un quanto di tempo di 8 millisecondi. Se non completa in questo tempo va in  $Q_1$ .
  - Nella coda  $Q_1$  il processo riceve 16 millisecondi. Se non completa l'esecuzione, dopo questo tempo viene tolto dalla CPU e assegnato alla coda  $Q_2$ .

# Scheduling Real-Time



- ◆ **In un sistema real-time**

- ◆ la correttezza dell'esecuzione non dipende solamente dal valore del risultato, ma anche dall'istante temporale nel quale il risultato viene emesso

- ◆ **Hard real-time**

- ◆ le deadline di esecuzione dei programmi non devono essere superate in nessun caso
- ◆ sistemi di controllo nei velivoli, centrali nucleari o per la cura intensiva dei malati

- ◆ **Soft real-time**

- ◆ errori occasionali sono tollerabili
- ◆ ricostruzione di segnali audio-video, transazioni interattive

# Scheduling Real-Time



- ◆ **Processi periodici**

- ◆ sono periodici i processi che vengono riattivati con una cadenza regolare (periodo)
- ◆ esempi: controllo assetto dei velivoli, basato su rilevazione periodica dei parametri di volo

- ◆ **Processi aperiodici**

- ◆ i processi che vengono scatenati da un evento sporadico, ad esempio l'allarme di un rilevatore di pericolo

# Valutazione di algoritmi di scheduling

## ■ Come scegliere un algoritmo di scheduling adatto/ottimale ?

⇒ Fissare i criteri di ottimizzazione.

✦ Usare metodi di valutazione:

➤ **Modellazione Deterministica** (valutazione analitica)

❖ Fissati i diversi carichi di lavoro definisce le prestazioni dei diversi algoritmi per ognuno dei carichi analizzati.

➤ **Modelli di code**

❖ Il sistema viene modellato come un insieme di server con le code associate; date le distribuzioni degli arrivi delle richieste si calcola la lunghezza media delle code, il tempo medio di attesa, etc.

➤ **Realizzazione**

➤ **Simulazione**

# Valutazione di algoritmi di scheduling

