Sistemi Operativi

Modulo 3: Scheduling

Renzo Davoli
Alberto Montresor

1. Scheduler, processi e thread

Introduzione

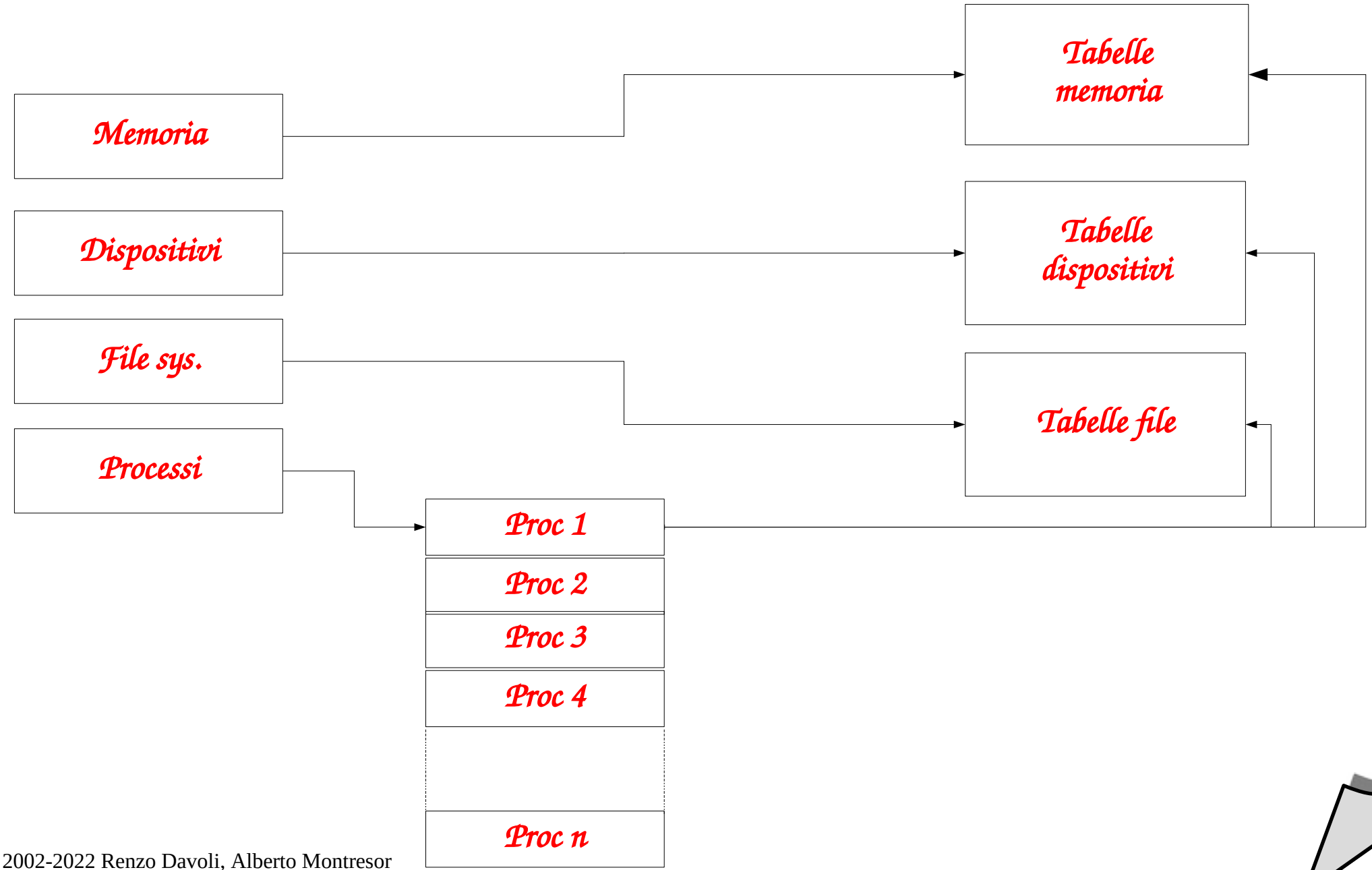
- **Un sistema operativo è un gestore di risorse**
  - processore, memoria principale e secondaria, dispositivi
- **Per svolgere i suoi compiti, un sistema operativo ha bisogno di strutture dati per mantenere informazioni sulle risorse gestite**
- **Queste strutture dati comprendono:**
  - tabelle di 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 è 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, PCB)***

PCB

- **Tabella per la gestione dei processi**
  - contiene i descrittori dei processi (PCB)
  - ogni processo ha un PCB 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

PCB

- **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*

PCB

- **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 schuduling utilizzato
      - priorità, puntatori per la gestione delle code
    - identificatore dell'evento per cui il processo è in attesa

PCB

- **Informazioni di controllo del processo (continua)**
  - informazioni di gestione della memoria
    - Informazioni di configurazione della MMU, es. 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**
  - Quando viene richiamato decide quale processo deve essere in esecuzione
  - interviene quando viene richiesta un'operazione di I/O e quando un'operazione di I/O termina, (l'interval timer si comporta come un device di I/O)
- **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

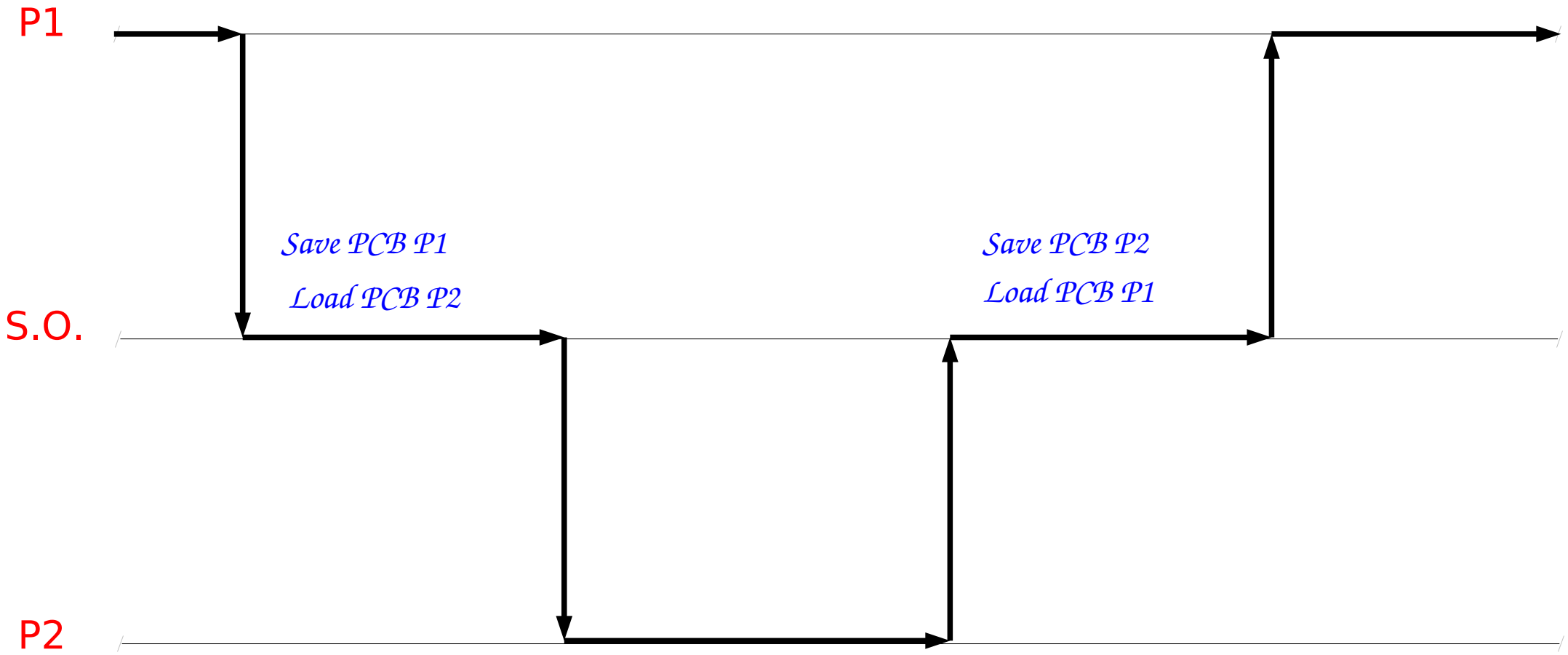
Mode switching e context switching

- **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*

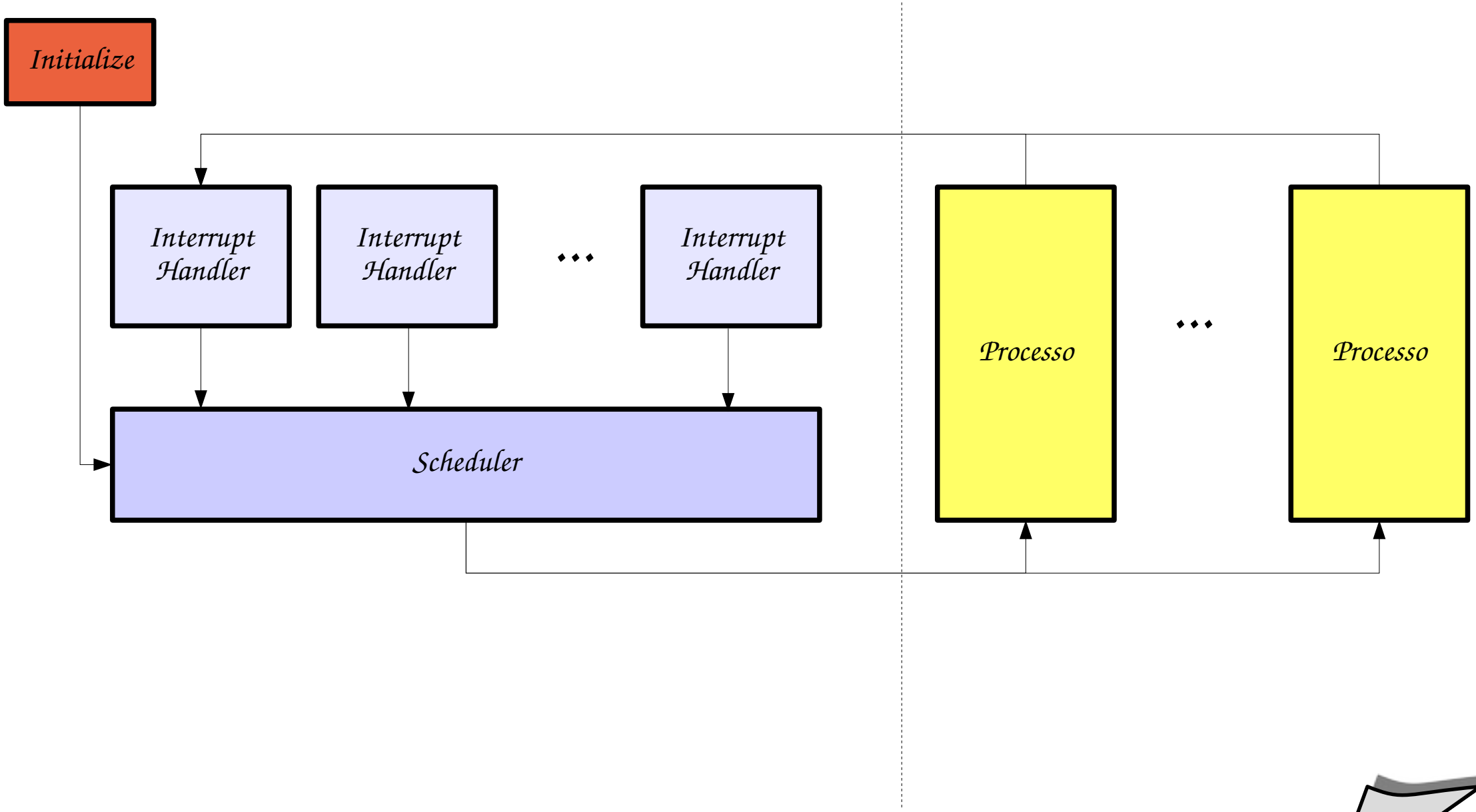
Context switching

- **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

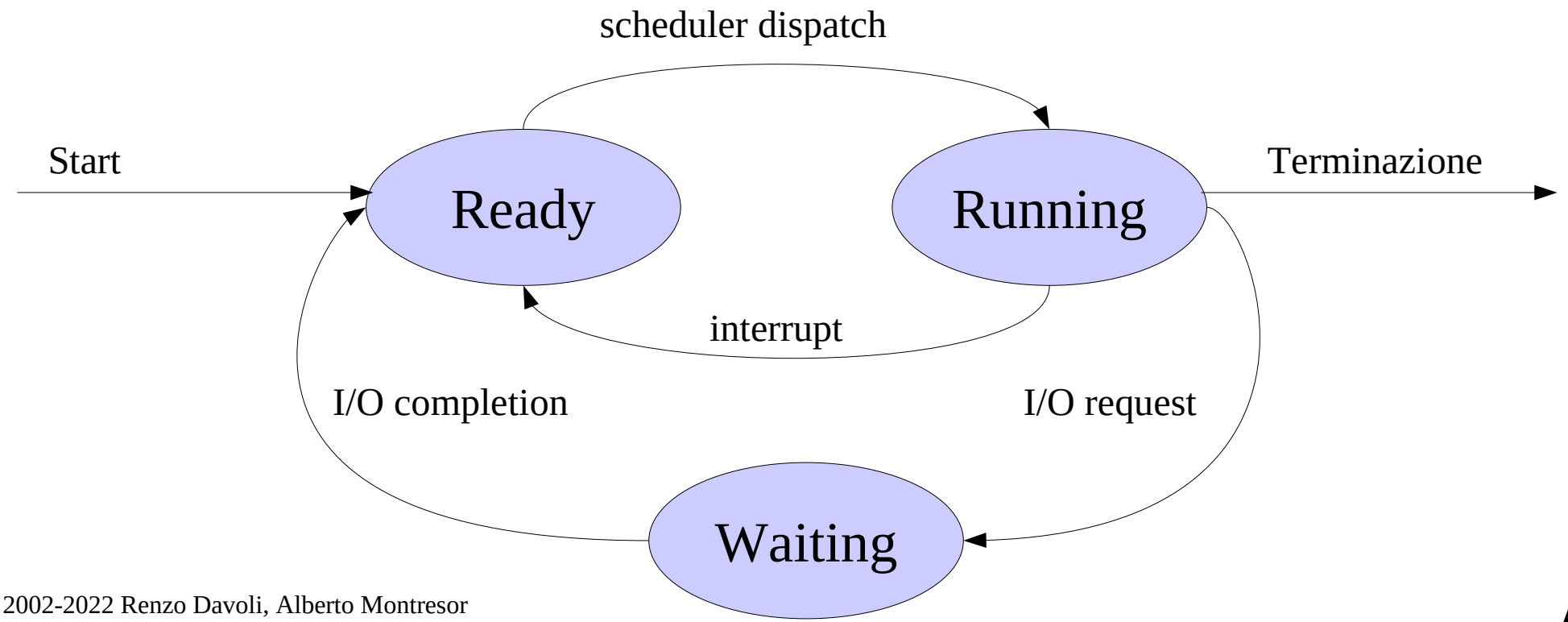


Schema di funzionamento di un kernel



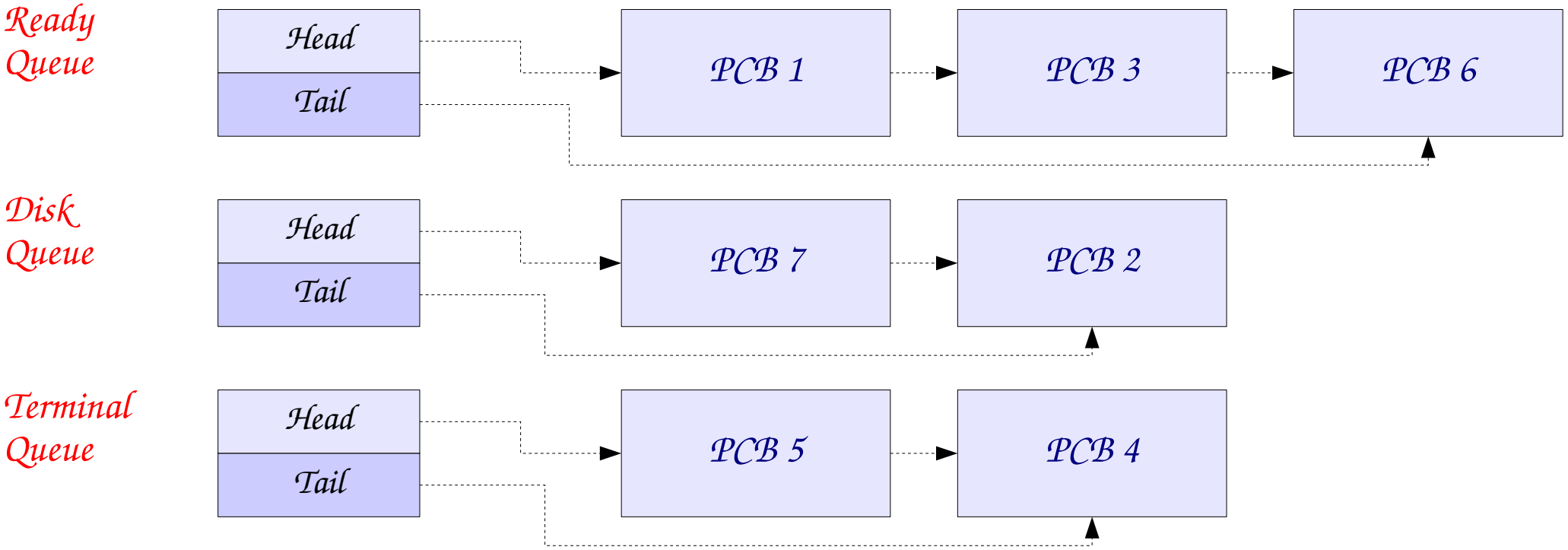
Vita di un processo

- **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

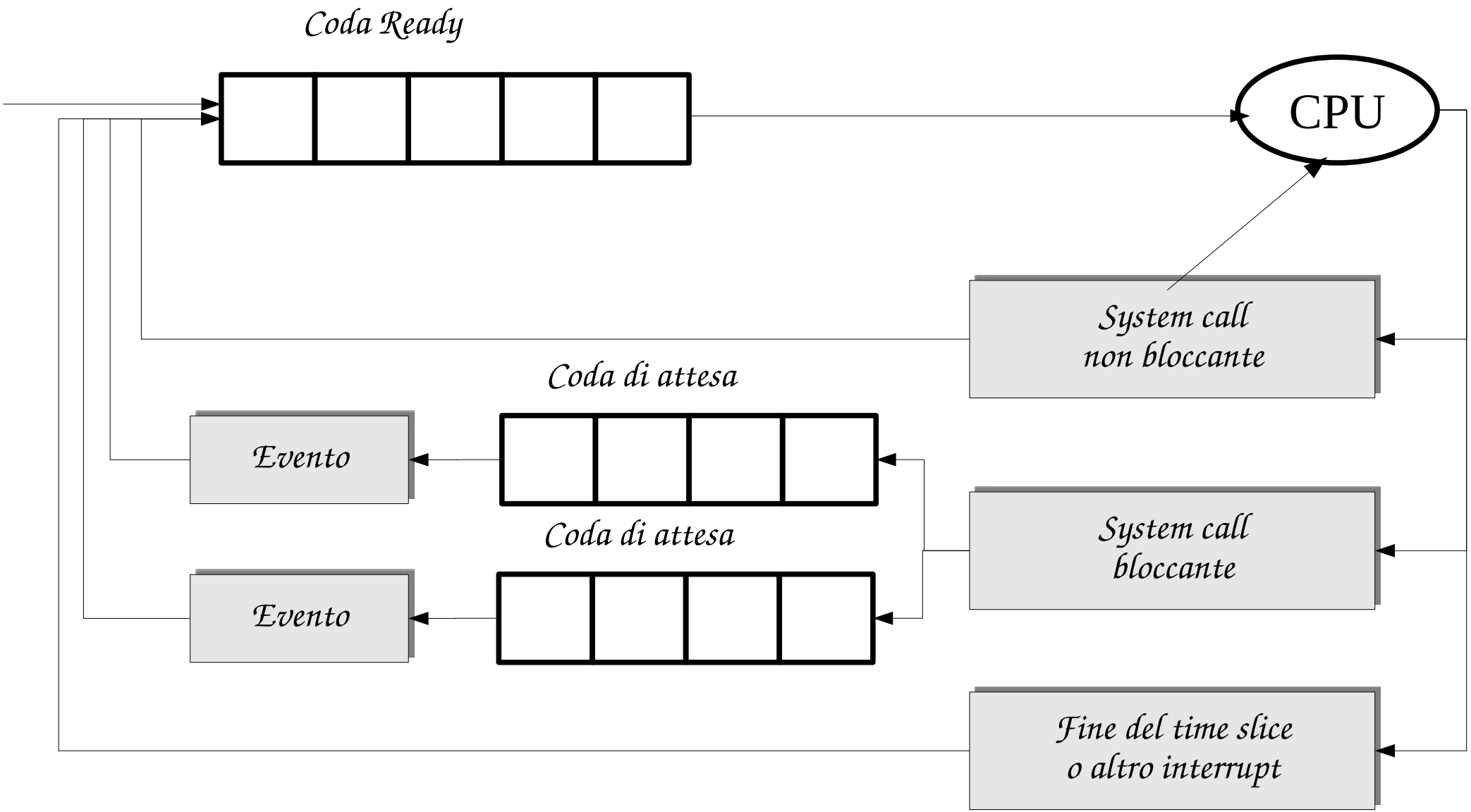


Code di processi

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



Vita di un processo nello scheduler



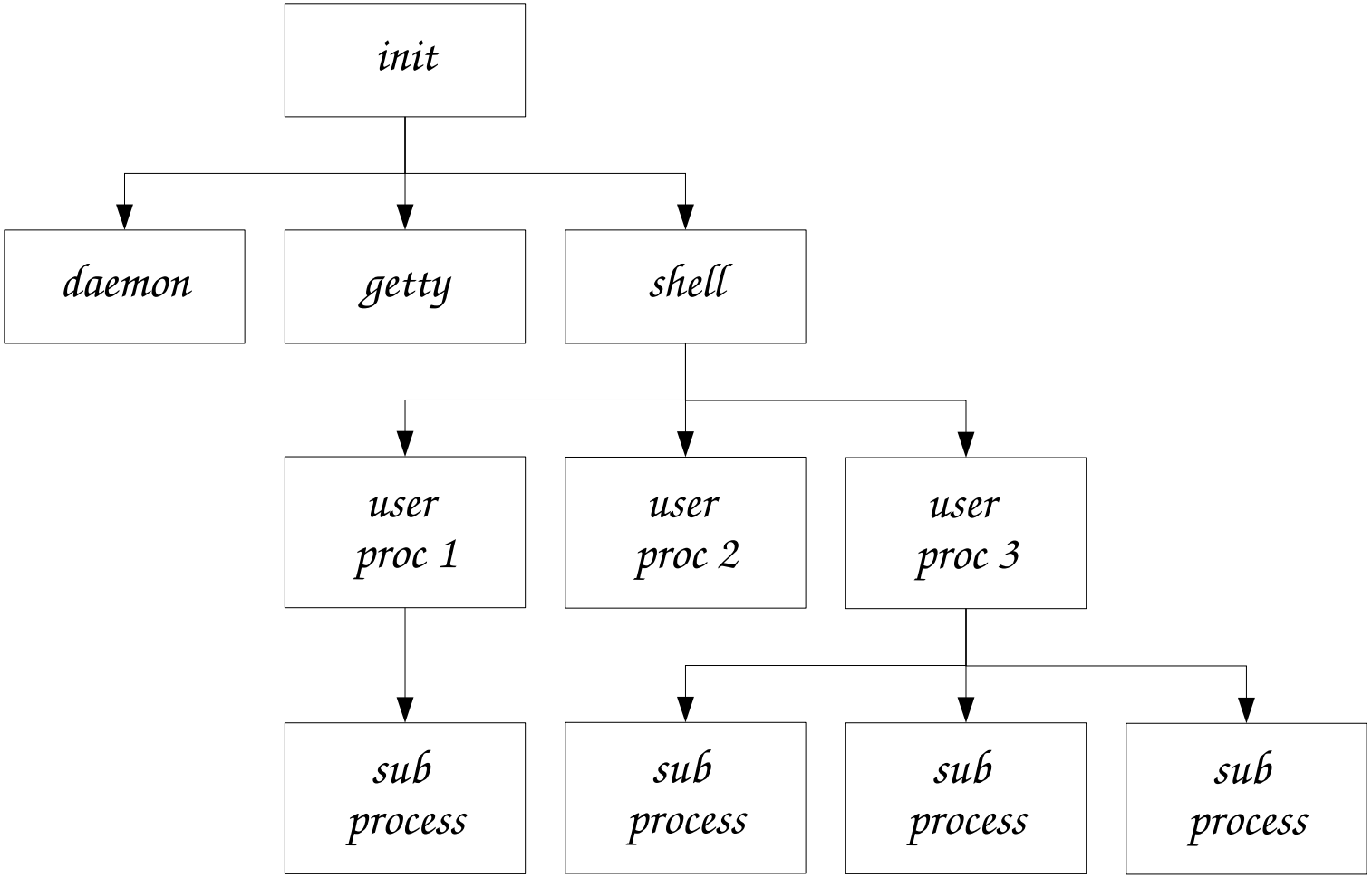
Un precisazione...

- **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
  - Di solito quando si parla di scheduler ci si riferisce a questo.
- **Long-term scheduler**
  - viene (veniva?) utilizzato per programmi batch
  - seleziona quali processi creare fra quelli che non hanno ancora iniziato la loro esecuzione
  - nei sistemi interattivi (UNIX), non appena un programma viene lanciato il processo relativo viene automaticamente creato
-

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 creante 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



Processi e Thread

- **La nozione di processo discussa in precedenza assume che ogni processo abbia una singola "linea di controllo"**
  - per ogni processo, viene eseguite una singola sequenza di istruzioni
  - un singolo processo non può eseguire due differenti attività contemporanemente
- **Esempi:**
  - scaricamento di due differenti pagine in un web browser
  - inserimento di nuovo testo in un word processor mentre viene eseguito il correttore ortografico

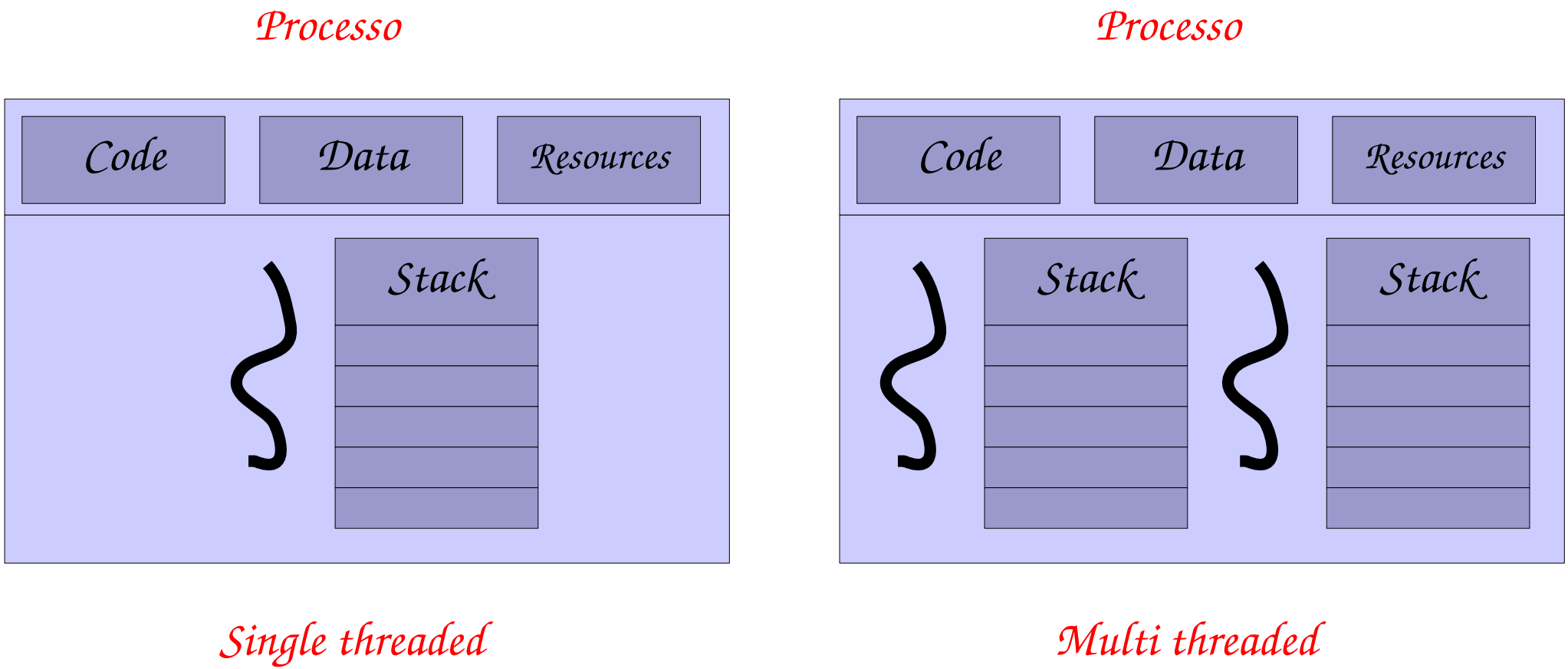
Processi e Thread

- **Tutti i sistemi operativi moderni**
  - supportano l'esistenza di processi multithreaded
  - in un processo multithreaded esistono molte "linee di controllo", ognuna delle quali può eseguire un diverso insieme di istruzioni
- **Esempi:**
  - Associando un thread ad ogni finestra aperta in un web browser, è possibile scaricare i dati in modo indipendente

Processi e thread

- **Un thread è l'unità base di utilizzazione della CPU**
- **Ogni thread possiede**
  - la propria copia dello stato del processore
  - il proprio program counter
  - uno stack separato
- **I thread appartenenti allo stesso processo condividono:**
  - codice
  - dati
  - risorse di I/O

Processi e thread



Benefici dei thread

- **Condivisione di risorse**
  - i thread condividono lo spazio di memoria e le risorse allocate degli altri thread dello stesso processo
  - condividere informazioni tra thread logicamente correlati rende più semplice l'implementazione di certe applicazioni
- **Esempio:**
  - web browser: condivisione dei parametri di configurazione fra i vari thread

Benefici dei thread

- **Economia**
  - allocare memoria e risorse per creare nuovi processi è costoso
  - fare context switching fra diversi processi è costoso
- **Gestire i thread è in generale più economico, quindi preferibile**
  - creare thread all'interno di un processo è meno costoso
  - fare context switching fra thread è meno costoso
- **Esempio:**
  - creare un thread in Solaris richiede 1/30 del tempo richiesto per creare un nuovo processo

Processi vs Thread

- **Thread = processi "lightweight"**
  - utilizzare i thread al posto dei processi rende l'implementazione più efficiente
  - in ogni caso, abbiamo bisogno di processi distinti per applicazioni differenti

Multithreading: implementazione

- **Un sistema operativo può implementare i thread in due modi:**
  - User thread
    (A livello utente)
  - Kernel thread
    (A livello kernel)

User thread

- **Gli user thread vengono supportati sopra il kernel e vengono implementati da una "thread library" a livello utente**
  - la thread library fornisce supporto per la creazione, lo scheduling e la gestione dei thread senza alcun intervento del kernel
- **Vantaggi:**
  - l'implementazione risultante è molto efficiente
- **Svantaggi:**
  - se il kernel è single-threaded, qualsiasi user thread che effettua una chiamata di sistema bloccante (che si pone in attesa di I/O) causa il blocco dell'intero processo

Kernel thread

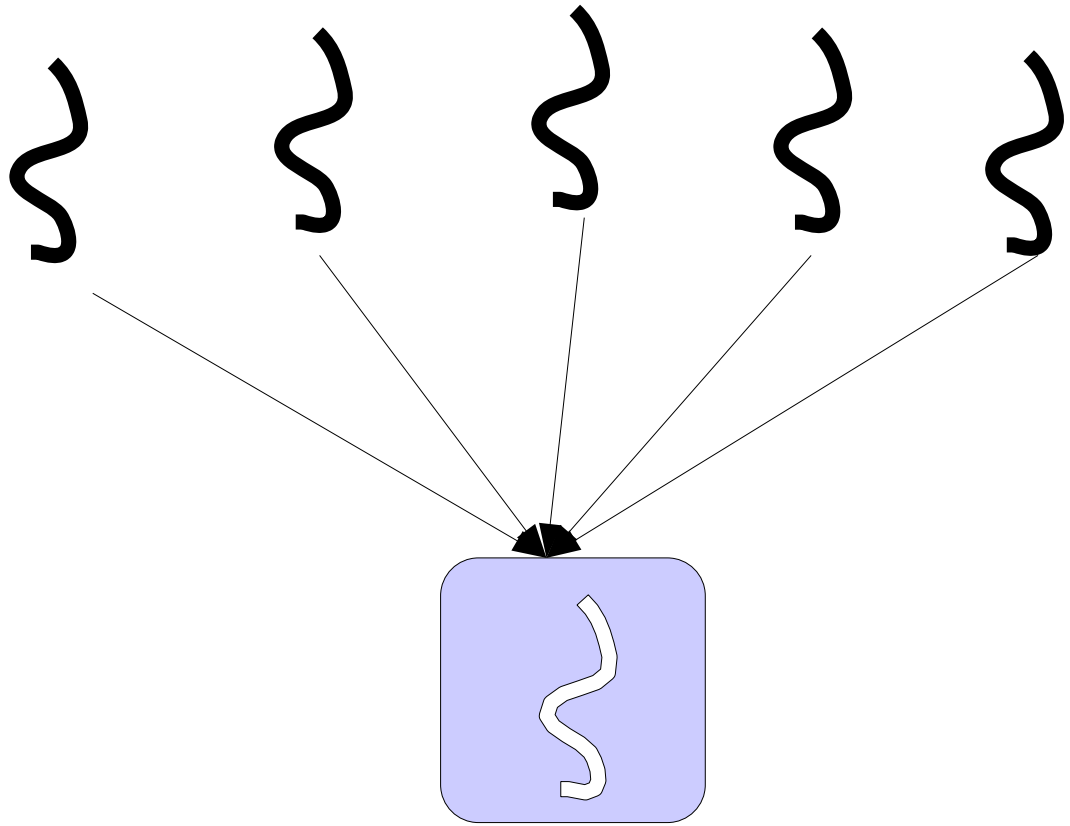
- **I kernel thread vengono supportati direttamente dal sistema operativo**
  - la creazione, lo scheduling e la gestione dei thread sono implementati a livello kernel
- **Vantaggi:**
  - poichè è il kernel a gestire lo scheduling dei thread, se un thread esegue una operazione di I/O, il kernel può selezionare un altro thread in attesa di essere eseguito
- **Svantaggi:**
  - l'implementazione risultante è più lenta, perché richiede un passaggio da livello utente a livello supervisore

Modelli di multithreading

- **Molti sistemi supportano sia kernel thread che user thread**
- **Si vengono così a creare tre differenti modelli di multithreading:**
  - Many-to-One
  - One-to-One
  - Many-to-Many

Many-to-One Multithreading

- **Un certo numero di user thread vengono mappati su un solo kernel thread**
- **Modello generalmente adottato da s.o. che non supportano kernel thread multipli**

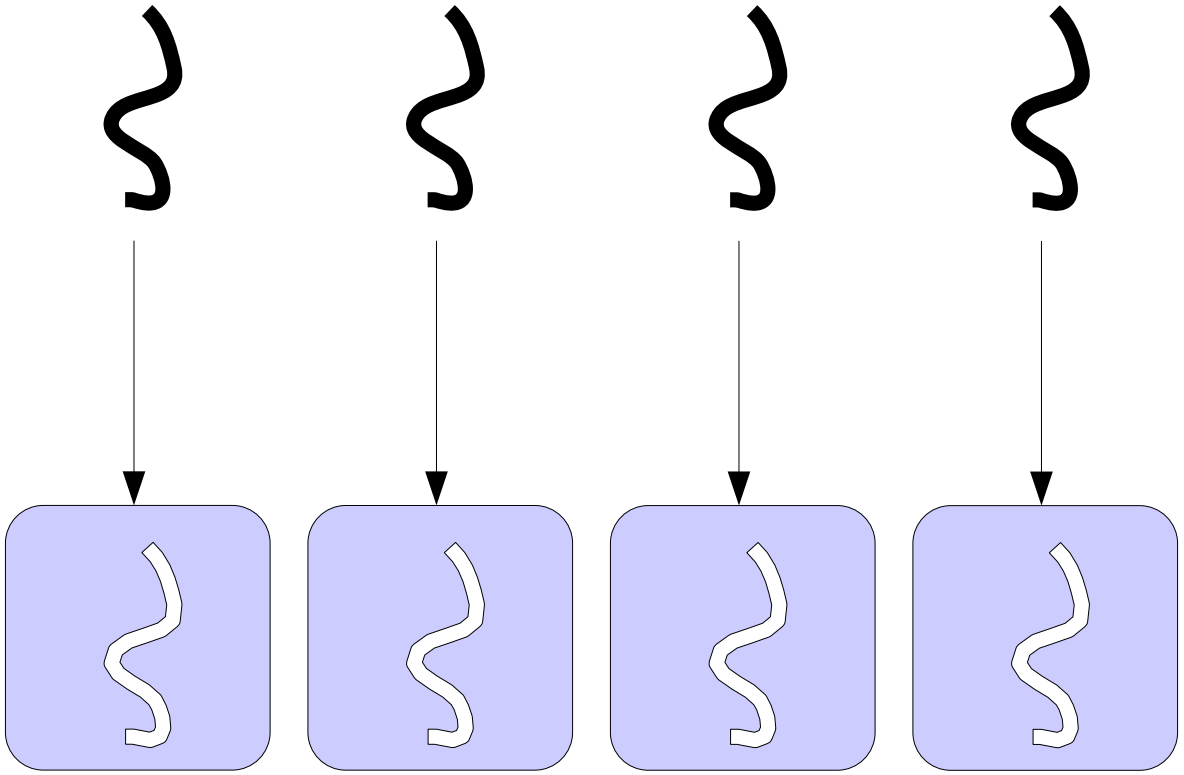


User Thread(s)

Kernel Thread(s)

One-to-One Multithreading

- **Ogni user thread viene mappato su un kernel thread**
- **Può creare problemi di scalabilità per il kernel**

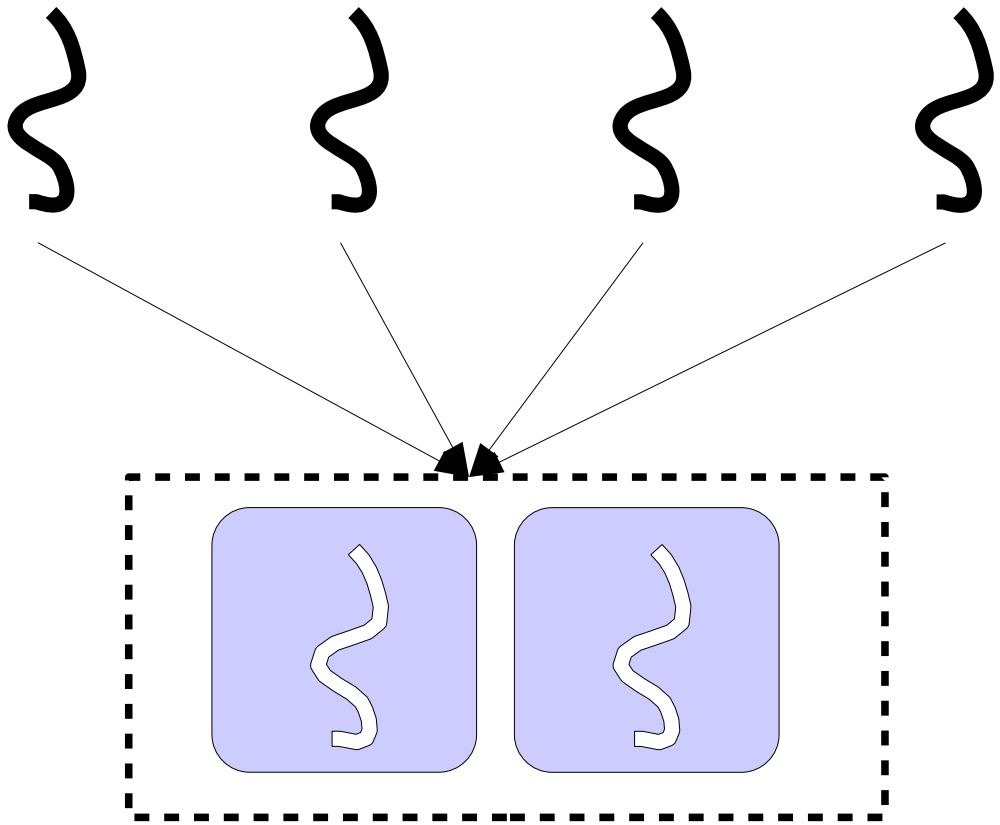


User Thread(s)

Kernel Thread(s)

Many-to-Many Multithreading

- **Riassume i benefici di entrambe le architetture**
- **Supportato da Solaris, IRIX, Digital Unix**



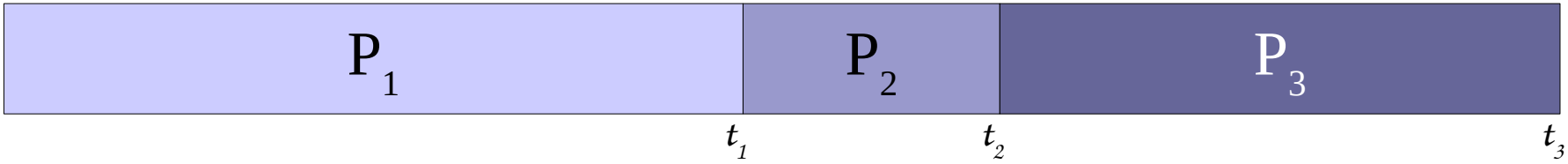
User Thread(s)

Kernel Thread(s)

2. Scheduling

Rappresentazione degli schedule

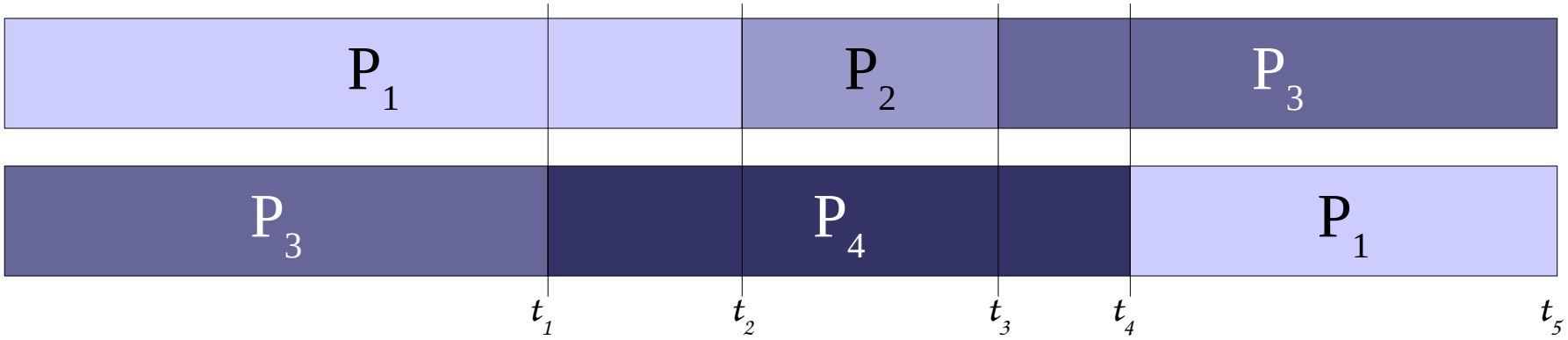
- **Diagramma di Gantt**
  - per rappresentare uno schedule si usano i diagrammi di Gantt



  - 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* <= 9
- **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

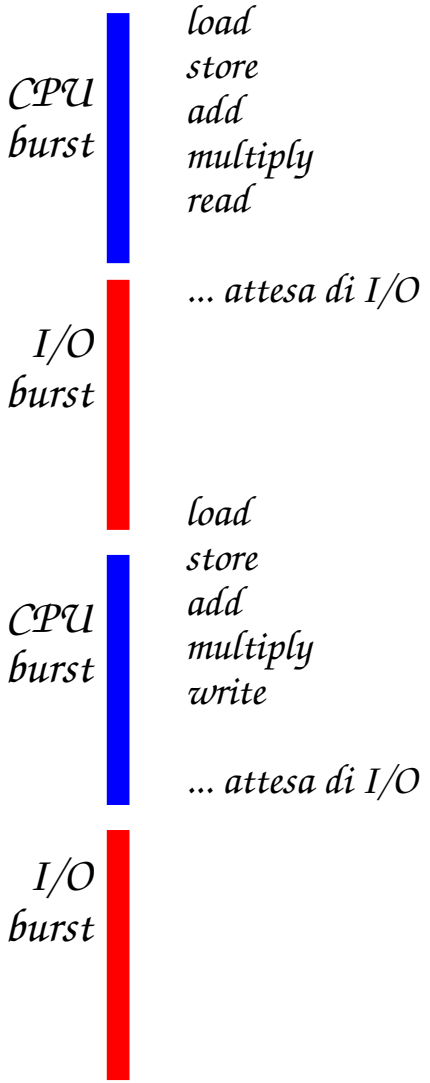
- **Utilizzo della risorsa (CPU)**
  - percentuale di tempo in cui la CPU è occupata ad eseguire processi
  - deve essere massimizzato
- **Throughput**
  - numero di processi completati per unità di tempo
  - dipende dalla lunghezza dei processi
  - deve essere massimizzato
- **Tempo di turnaround**
  - 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 trascurso 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 CPU burst molto brevi 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)

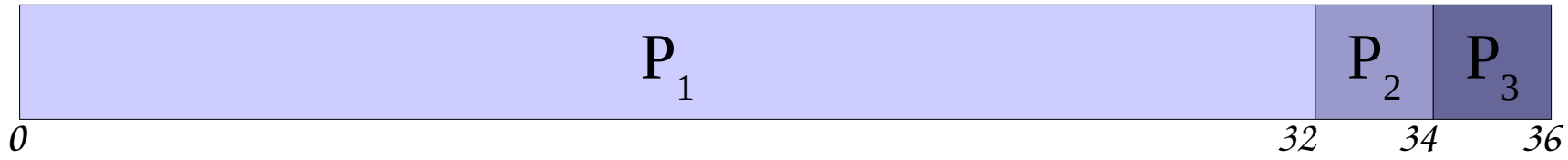
- **Algoritmo**
  - il processo che arriva per primo, viene servito per primo
  - politica senza preemption
- **Implementazione**
  - semplice, tramite una coda (politica FIFO)
- **Problemi**
  - elevati tempi medi di attesa e di turnaround
  - processi CPU bound ritardano i processi I/O bound

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

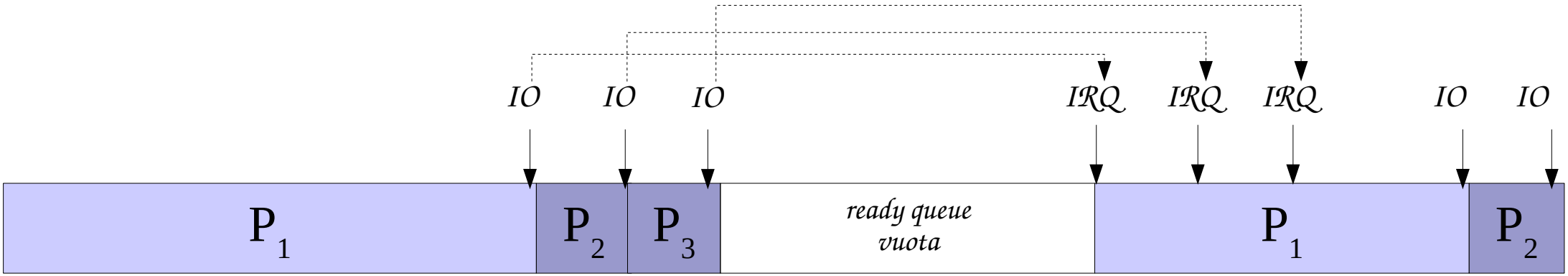
    (32+34+36)/3 = 34 ms
  - Tempo medio di attesa:

    (0+32+34)/3 = 22 ms



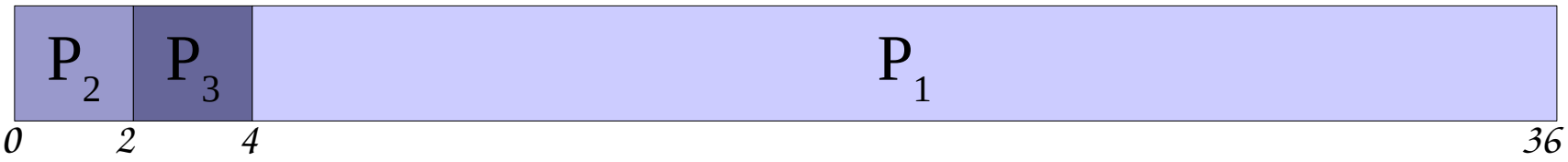
First Come, First Served (FCFS)

- **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*



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
  - politica senza preemption
- **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)

- **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 τ_n la corrispondente previsione; τ_{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

$$\tau_{n+1} = \Sigma_{j=0..n}\, \alpha(1-\alpha)^j t_{n-j} + (1-\alpha)^{n+1} \tau_o$$

  da cui il nome media esponenziale

Shortest Job First (SJF)

- **Spiegazione**
  - t_n rappresenta la storia recente
  - τ_n rappresenta la storia passata
  - α rappresenta il peso relativo di storia passata e recente
  - cosa succede con $\alpha = 0$, 1 oppure ½?
- **Nota importante:**
  - SJF può essere soggetto a starvation!

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 esegure è il primo della coda dei processi pronti

Scheduling Round-Robin

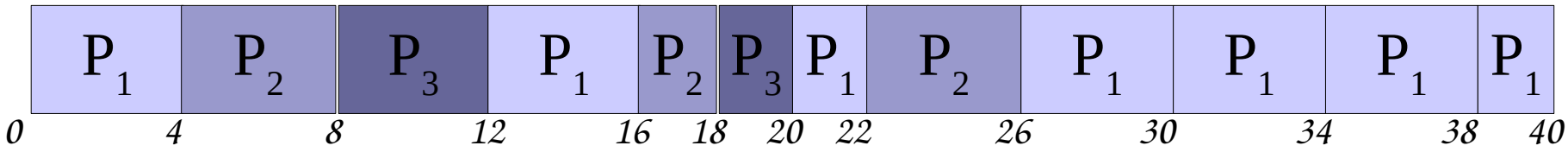
- **La durata del quanto di tempo è un parametro critico del sistema**
  - se il quanto di tempo è breve, il sistema è meno efficiente perchè deve cambiare il processo attivo più spesso
  - se il quanto è lungo, in presenza di numerosi processi pronti ci sono lunghi periodi di inattività di ogni singolo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

Scheduling Round-Robin

- **Implementazione (2)**
  - è necessario che l'hardware fornisca un timer (interval timer) che agisca come "sveglia" del processore
  - il timer è un dispositivo che, attivito 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'unita di I/O

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, < 2ms)
  - Tempo medio di risposta: 4 ms



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 dei un video MPEG potrebbe essere ritardata da un processo che sta smistando la posta
  - la lettera può aspettare ½ 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
- **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)
  - il processo vuoto (FIFO banale)

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

Esempi di scheduler Real-Time

- **Rate Monotonic:**
  - è una politica di scheduling, valida alle seguenti condizioni
    - ogni processo periodico deve completare entro il suo periodo
    - tutti i processi sono indipendenti
    - la preemption avviene istantaneamente e senza overhead
  - viene assegnata staticamente una priorità a ogni processo
  - processi con frequenza più alta (i.e. periodo più corto) hanno priorità più alta
  - ad ogni istante, viene eseguito il processo con priorità più alta (facendo preemption se necessario)
- **Earliest Deadline First:**
  - è una politica di scheduling per processi periodici real-time
  - viene scelto di volta in volta il processo che ha la deadline più prossima
  - viene detto "a priorità dinamica" perchè la priorità relativa di due processi varia in momenti diversi