Sistemi operativi e programmazione concorrente

Docente: Antonio Vito Ciliberto

A.A. 2023/2024

Informazioni sul corso

- Home page del corso: https://learn.unimol.it/course/view.php?id=7209
- Docente: Antonio Vito Ciliberto
- Periodo: II semestre marzo 2024 – luglio 2024
  - Martedì dalle 14:00 alle 17:00 aula Bird (secondo piano)
  - Giovedì dalle 14:00 alle 17:00 aula Bird (secondo piano)
  - Venerdì dalle 10:00 alle 13:00 aula Bird (secondo piano)

Ricevimento

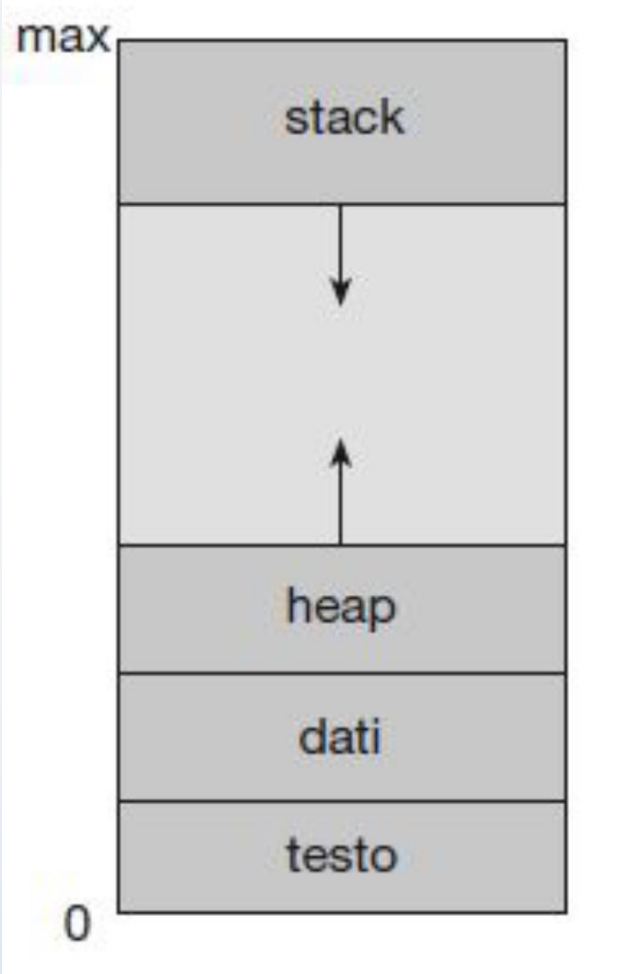
- In presenza, durante il periodo delle lezioni: da concordare con il docente tramite mail
- Tramite google meet : da concordare con il docente tramite mail
- Per prenotare un appuntamento inviare una email a
  - antonio.ciliberto@unimol.it

Concetto di processo

…dalle puntate precedenti!

Un sistema batch (lotti) esegue job (lavori), mentre un sistema time-sharing esegue programmi utente o task; queste attività sono simili per molti aspetti, perciò sono chiamate **processi**.

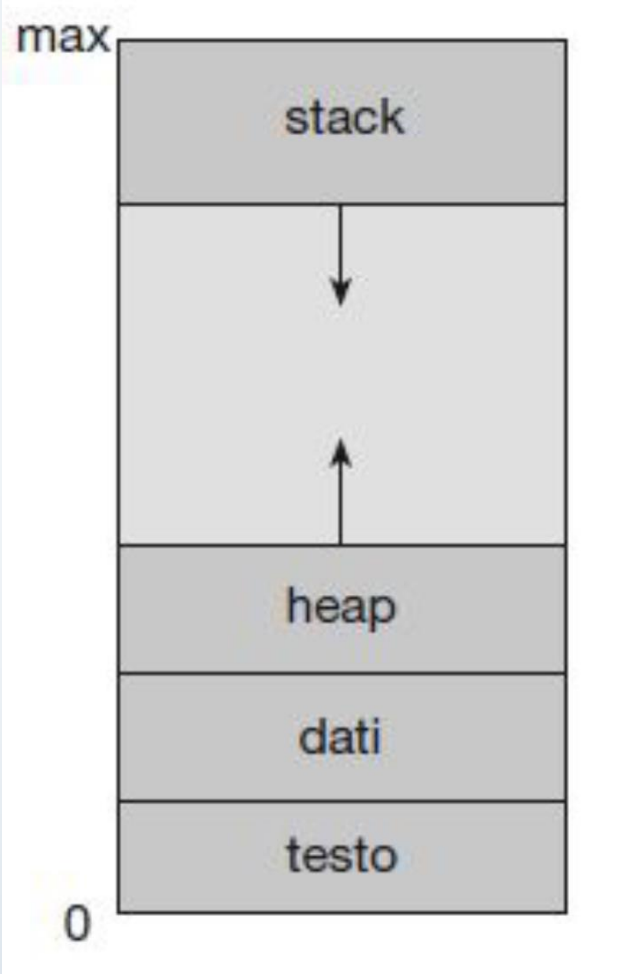
Un **processo** è un programma in esecuzione. La struttura di un processo in memoria è generalmente suddivisa in più sezioni.



Sezioni di processo

…dalle puntate precedenti!

- **Stack:** memoria temporaneamente utilizzata durante le chiamate di funzioni.
- **Heap:** memoria allocata dinamicamente durante l'esecuzione del programma
- **Dati:** contenente le variabili globali
- **Testo:** contenente il codice eseguibile



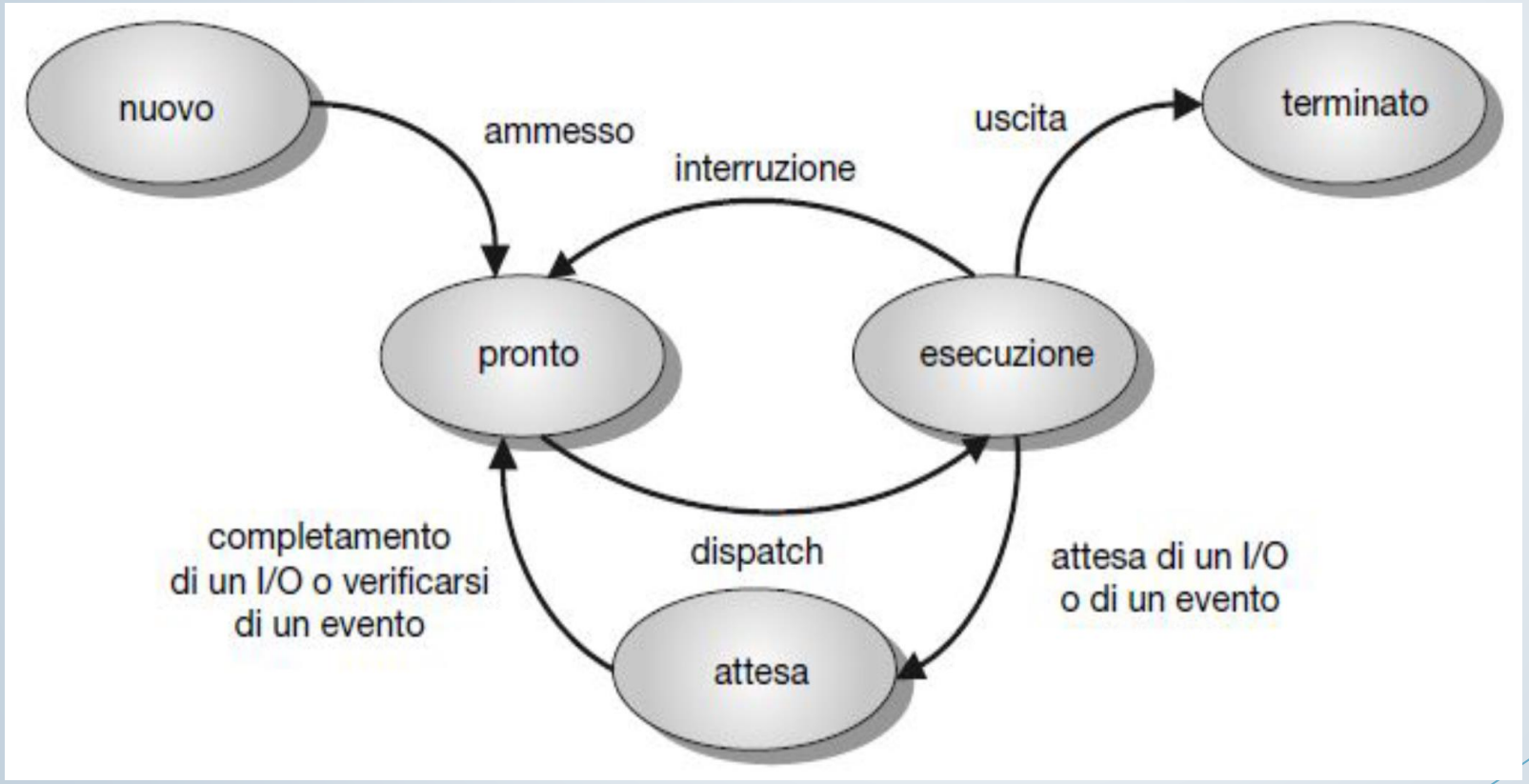
Stato di un processo

…dalle puntate precedenti!

- **Nuovo:** Si crea il processo
- **Esecuzione (running):** Le sue istruzioni vengono eseguite
- **Attesa (waiting):** Il processo attende che si verifichi qualche evento
- **Pronto (ready):** Il processo attende di essere assegnato a un’unità di elaborazione
- **Terminato:** Il processo termina l’esecuzione

Stato di un processo

…dalle puntate precedenti!



PCB – Process Control Block

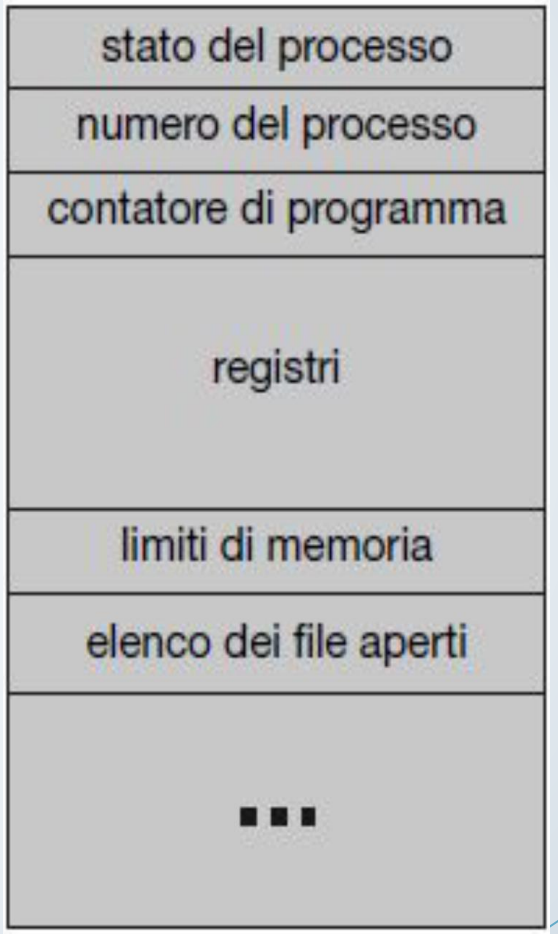
…dalle puntate precedenti!

- Ogni processo è rappresentato nel sistema operativo da un **blocco di controllo** *(process control block, PCB, o task control block, TCB)*
- Il PCB contiene un insieme di informazioni connesse a un processo specifico.

PCB – Process Control Block

…dalle puntate precedenti!

- **Stato del processo:** nuovo, pronto, esecuzione, attesa, arresto
- **Contatore di programma:** che contiene l'indirizzo della successiva istruzione da eseguire per tale processo.
- **Registri della CPU:** accumulatori, registri indice, puntatori alla cima dello stack (*stack pointer),* registri di uso generale e registri contenenti i codici di condizione (*condition codes*)



Come avviene la comunicazione

…dalle puntate precedenti!

Due modelli fondamentali:

- **A memoria condivisa**: si stabilisce una zona di memoria condivisa dai processi cooperanti, che possono cosi comunicare scrivendo e leggendo da tale zona.

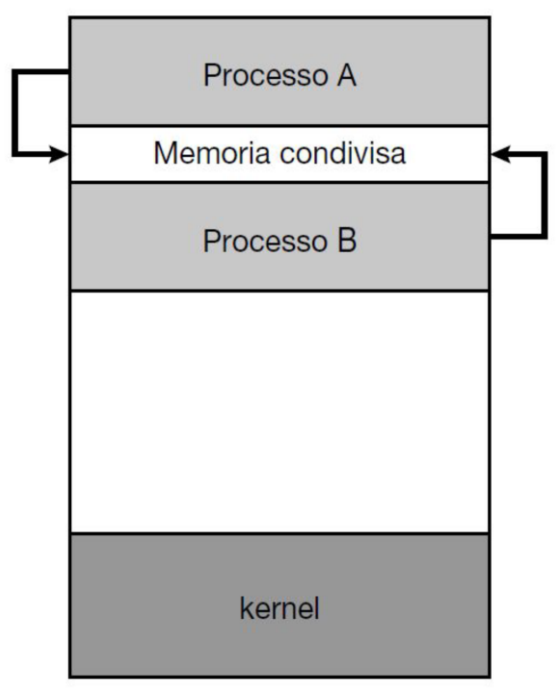
- **A scambio di messaggi**: in questo modello la comunicazione avviene per mezzo di messaggi. Utile per trasmettere piccole quantità di dati, ed è molto facile da realizzare.

Nei sistemi operativi sono diffusi entrambi i modelli, spesso coesistono in un unico sistema.

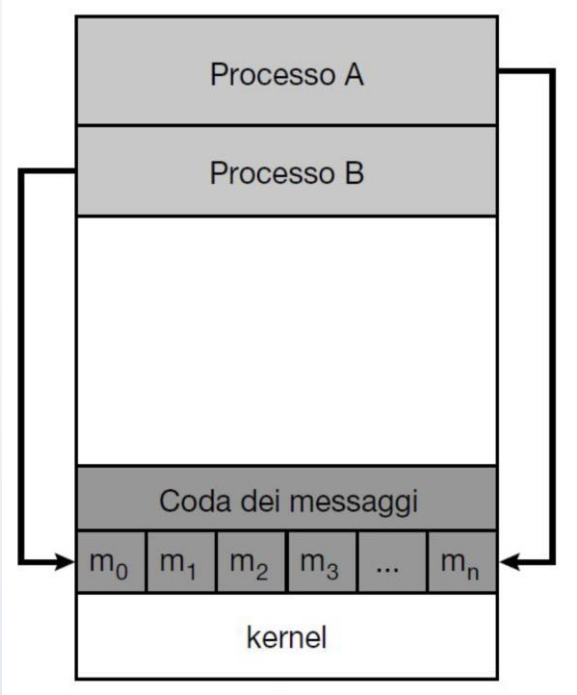
Come avviene la comunicazione

…dalle puntate precedenti!

Memoria condivisa



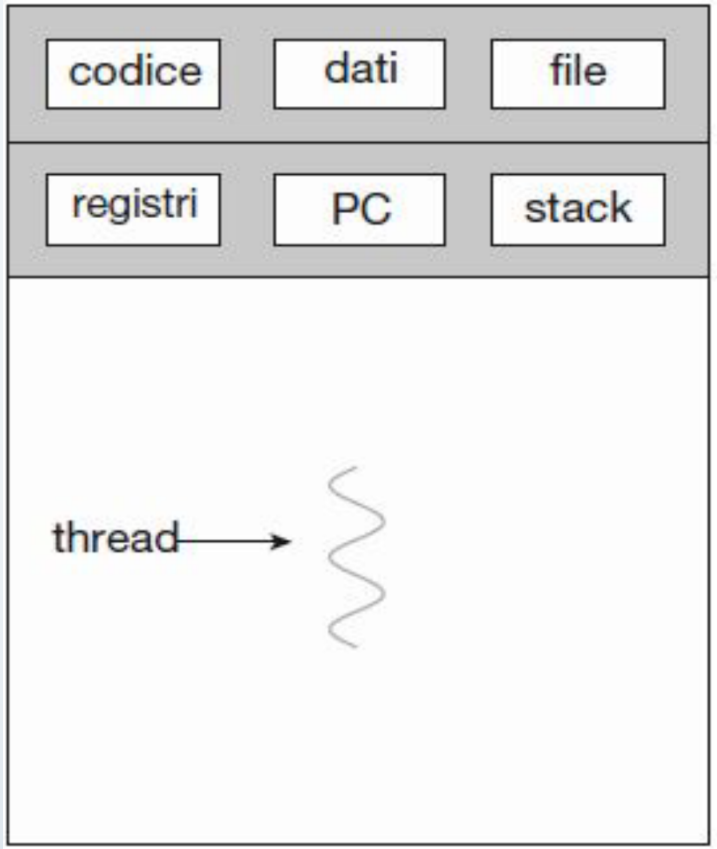
Scambio di messaggi



Definizione di thread

…dalle puntate precedenti!

- Un **thread** è l'unità di base d'uso della CPU e comprende
  - un identificatore di thread (ID)
  - un contatore di programma (PC)
  - un insieme di registri
  - una pila (*stack*)



processo a singolo thread

Multithread process - Vantaggi

…dalle puntate precedenti!

- I vantaggi della programmazione mutithread si possono classificare in 4 grandi categorie
  - **Tempo di risposta:** permettere ad un programma di continuare la sua esecuzione nonostante una parte di esso sia bloccata o stia eseguendo altre operazioni.
  - **Condivisione delle risorse:** Memoria condivisa o scambio di messaggi.
  - **Economia:** vista la capacità dei thread di condividere memoria e risorse non è necessario allocare altro spazio per i nuovi processi.
  - **Scalabilità:** nelle architetture molticore i thread possono essere eseguiti in parallelo.

Gestione dei processi leggeri(thre

…dalle puntate precedenti!

Modello molti a uno

Modello uno a uno

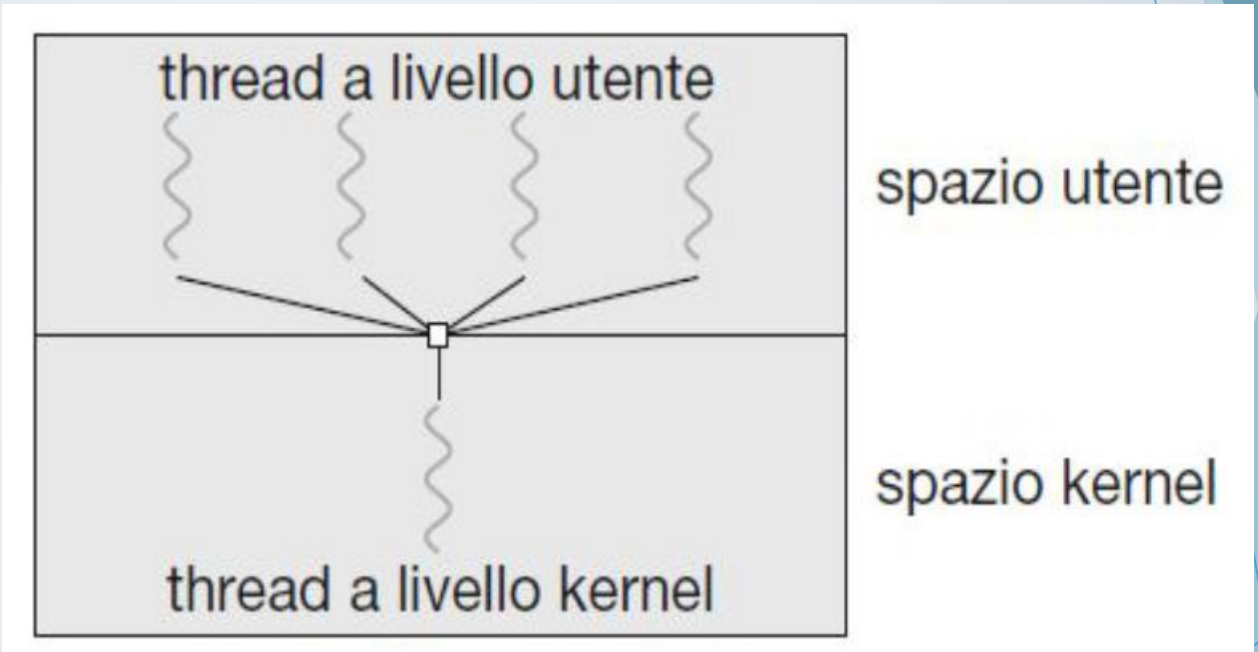
Modello molti a molti

Modello a due livelli

Relazioni tra thread – molti a uno

…dalle puntate precedenti!

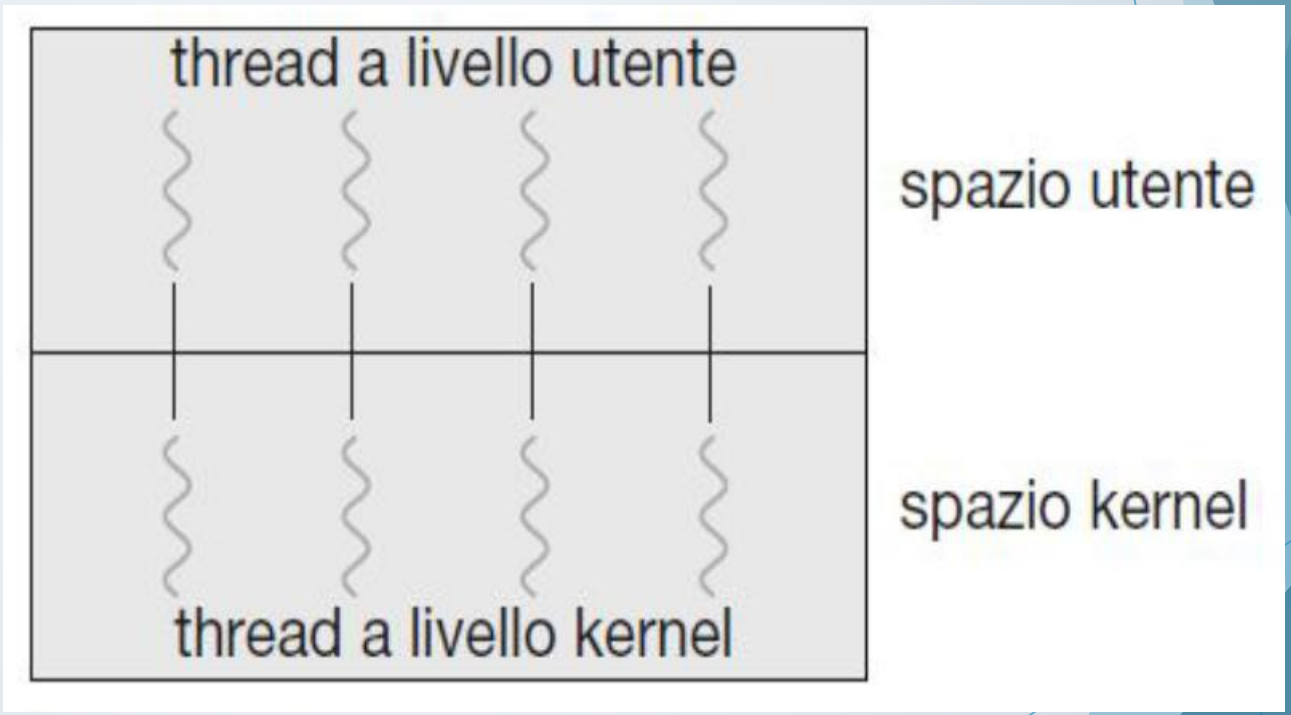
- In questo modello la gestione dei thread risulta efficiente in quanto effettuata per mezzo di una libreria a livello utente. Tuttavia se un thread invoca una chiamata di sistema di tipo bloccante l'intero processo resta bloccato.



Relazioni tra thread – uno a uno

…dalle puntate precedenti!

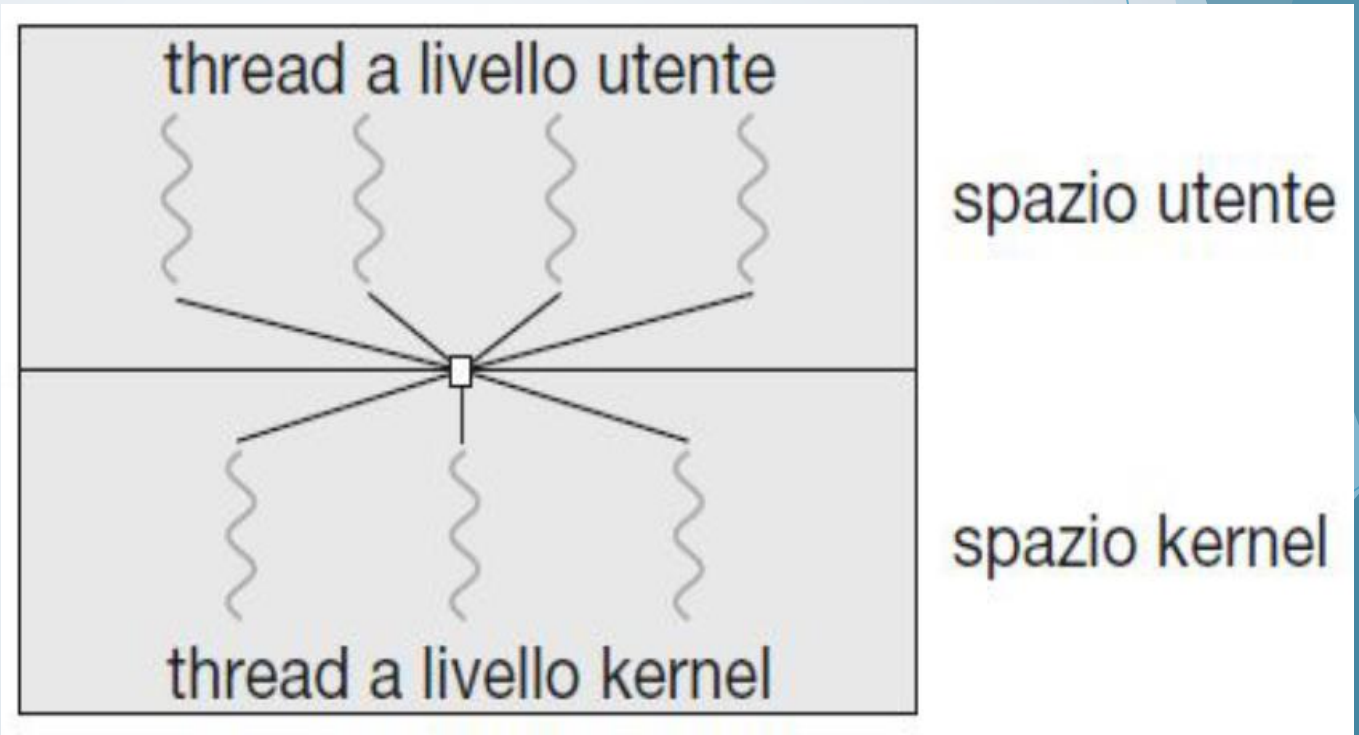
- In questo modello il grado di concorrenza è maggiore, si risolve il problema della chiamata di sistema bloccante, e nei sistemi multicore è possibile anche l'esecuzione in parallelo. Unico svantaggio la creazione di un thread kernel per ogni thread utente.



Relazioni tra thread – molti a molt

…dalle puntate precedenti!

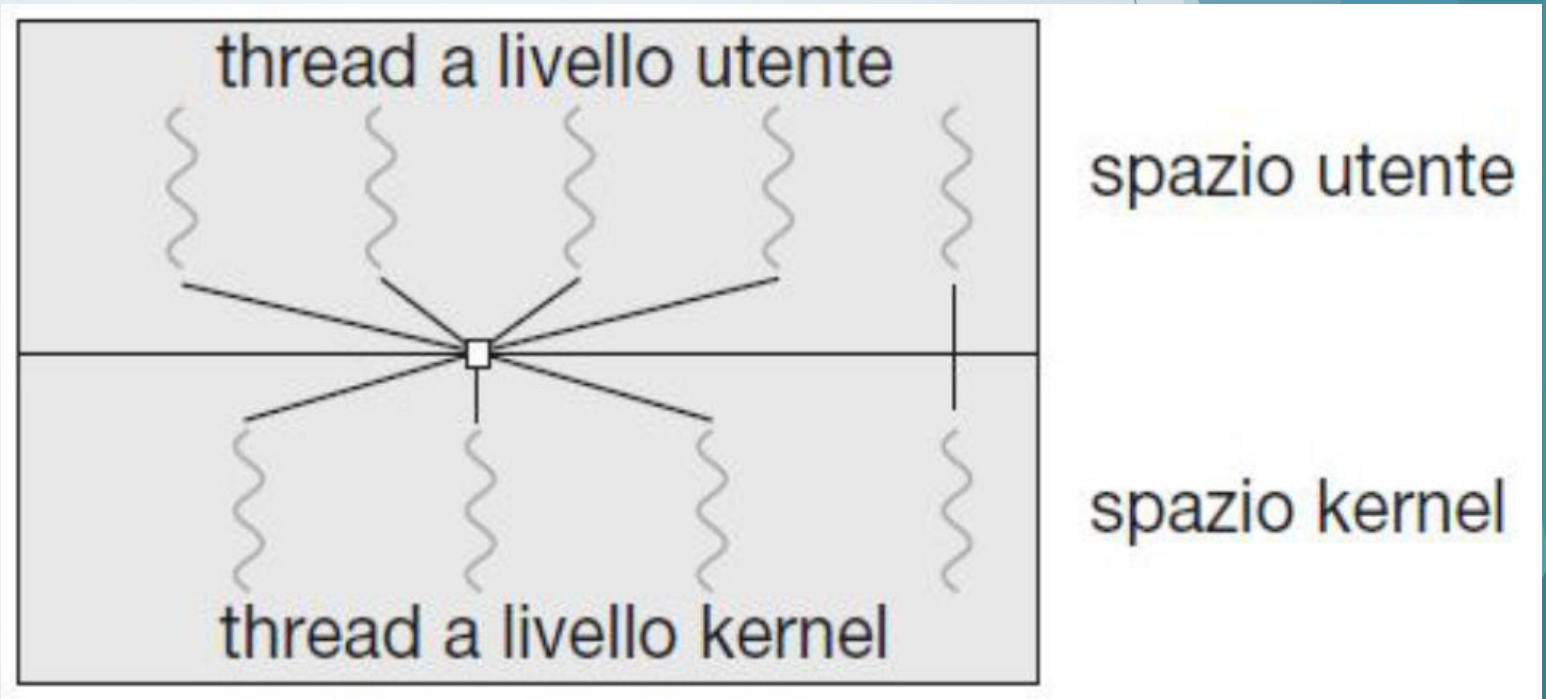
- In questo modella è permessa la creazione di più thread sia a livello utente che a livello kernel, quest'ultimo può essere specifico per una certa applicazione o per un particolare calcolatore. Diversa quantità di assegnazione in base all'architettura 8 core/4 core ecc.



Relazioni tra thread – 2 livelli

…dalle puntate precedenti!

- Questa variante del modello molti a molti mantiene la corrispondenza numerica tra thread a livello utente e a livello kernel con la possibilità di vincolare un thread a livello utente con un thread a livello kernel.



API – librerie per thread

…dalle puntate precedenti!

- Una libreria dei thread fornisce al programmatore una API per la creazione e la gestione dei thread. Attualmente sono tre le librerie di thread maggiormente in uso:
  - **Pthread** – estensione dello standard Posix, puo essere realizzata sia a livello utente che a livello kernel
  - **Thread Windows** – libreria a livello kernel per ambiente windows
  - **Java** – gestita direttamente dall'applicazione, la JVM gestisce questa libreria in base al SO ospitante java.

Programma del corso

- Introduzione ai sistemi operativi. Attività e struttura di un sistema operativo.
- I sistemi a processi. Comunicazione: condivisione di memoria, scambio di messaggi. Thread e concorrenza. Scheduling della CPU.
- I semafori. Cooperazione e sincronizzazione. Deadlock.
- Gestione dell'unità centrale.
- Gestione della memoria di massa.
- Il file system. Attributi, operazioni e metodi di accesso.
- Sicurezza. Protezione.

Concetto fondamentale

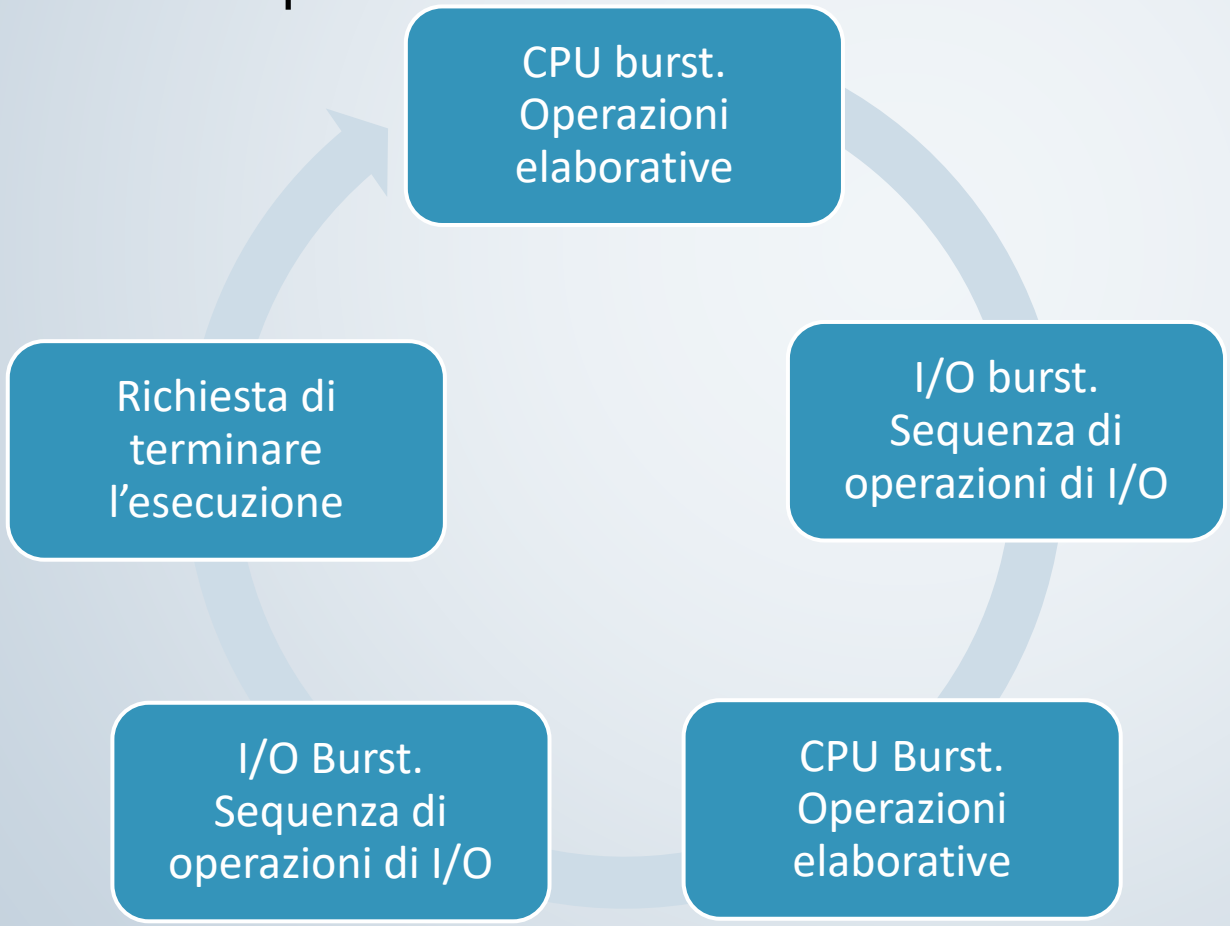
- In un sistema dotato di un singolo core di elaborazione si può eseguire al massimo un processo alla volta, gli altri processi, se ci sono, devo attendere prima di poter usufruire della CPU.

- L'obiettivo della multi programmazione è avere sempre un processo in esecuzione al fine di massimizzare l'utilizzo della CPU.

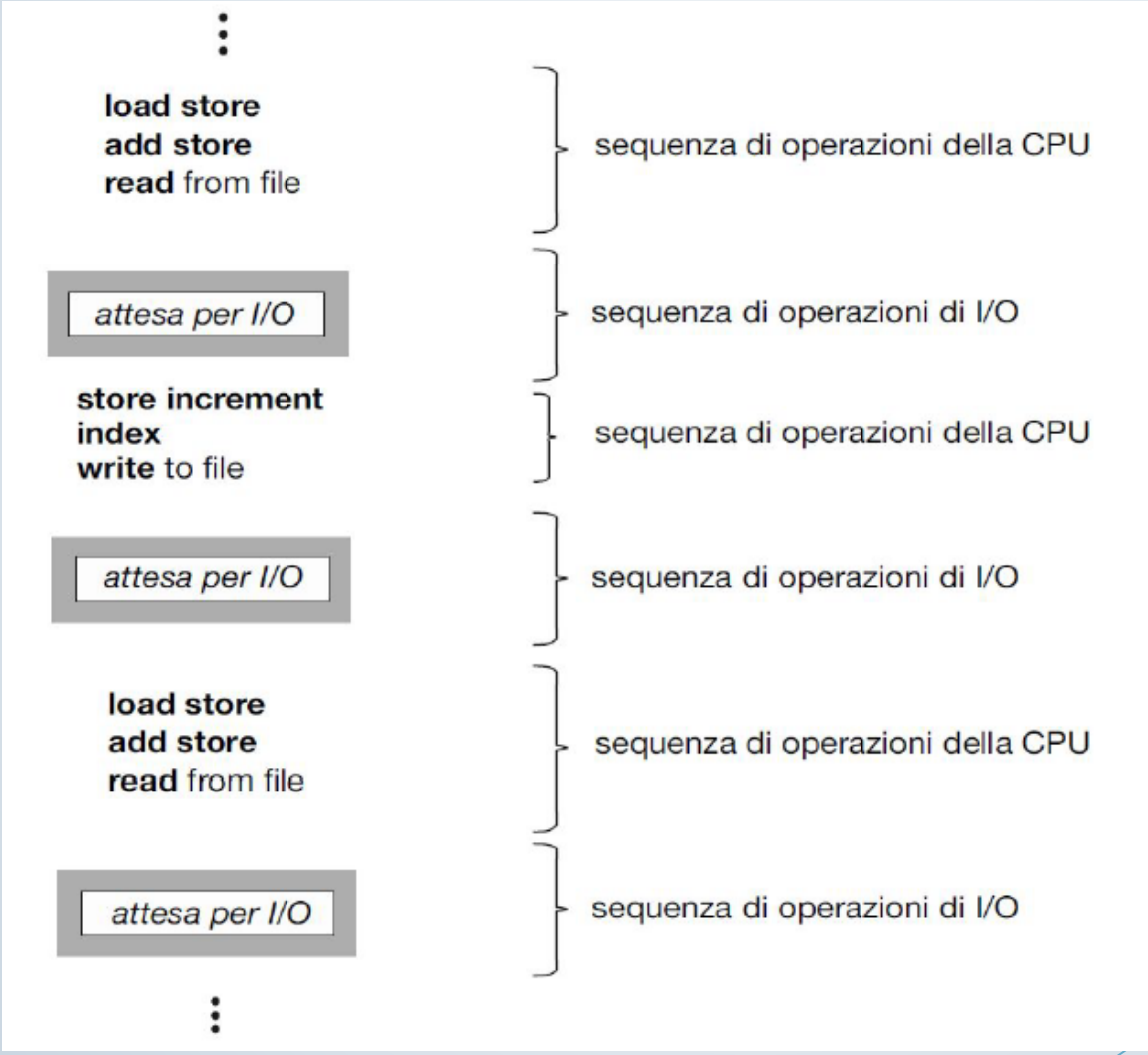
- Soluzione ideale: un processo resta in esecuzione finchè non deve attendere un evento, la multiprogrammazione a questo punto permette ad un altro processo di andare in esecuzione.

Ciclicità delle fasi d'elaborazione

L'esecuzione di un processo consiste in un ciclo di elaborazione:



Ciclicità delle fasi d'elaborazione



Scheduler della CPU

- Ogniqualvolta la CPU passa nello stato d'inattività. Il sistema operativo scegli per l'esecuzione uno dei processi presenti nella ready queue.
- In particolare è lo scheduler a breve termine, o lo scheduler della CPU che sceglie tra i processi in memoria pronti.
- Come vedremo non per forza secondo una politica FIFO.

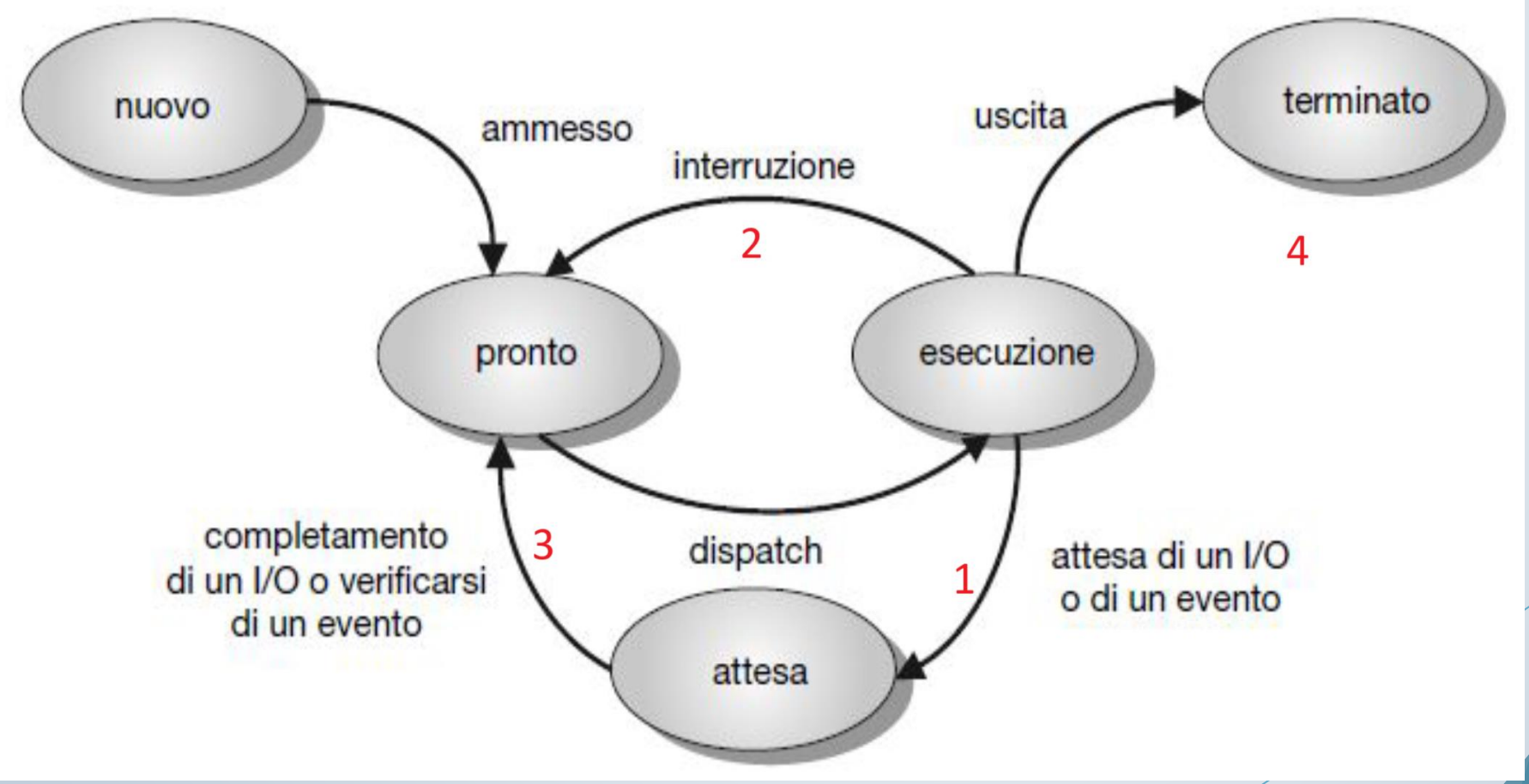
Scheduler della CPU

Lo scheduler interviene nelle seguenti situazioni:

1. Un processo passa dallo stato di esecuzione allo stato di attesa
2. Un processo passa dallo stato di esecuzione allo stato pronto
3. Un processo passa dallo stato di attesa allo stato pronto
4. Un processo termina

Scheduler della CPU

Lo scheduler interviene nelle seguenti situazioni:



Scheduler senza prelazione

- I casi 1 e 4 (si ricordano nell'immagine qui di lato) non danno alternative in termini di scheduling, si deve comunque scegliere un nuovo processo da eseguire.

- In queste condizioni si dice che lo schema di scheduling è senza prelazione (non preemptive) o cooperativo.

1. Un processo passa dallo stato di esecuzione allo stato di attesa

4. Un processo termina

Scheduler con prelazione

- I casi 2 e 3 (si ricordano nell'immagine qui di lato) è possibile scegliere il processo da mandare in esecuzione.

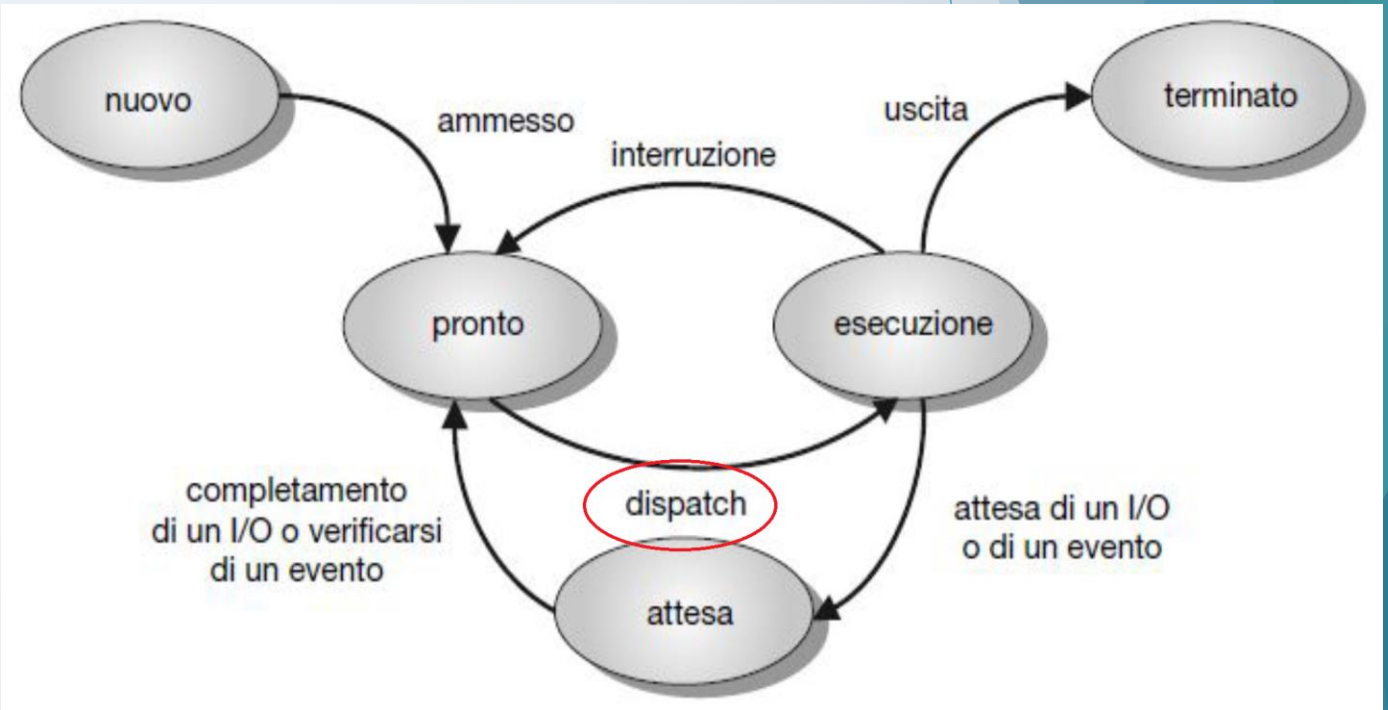
- In queste condizioni si dice che lo schema di scheduling è con prelazione (preemptive)

2. Un processo passa dallo stato di esecuzione allo stato pronto

3. Un processo passa dallo stato di attesa allo stato pronto

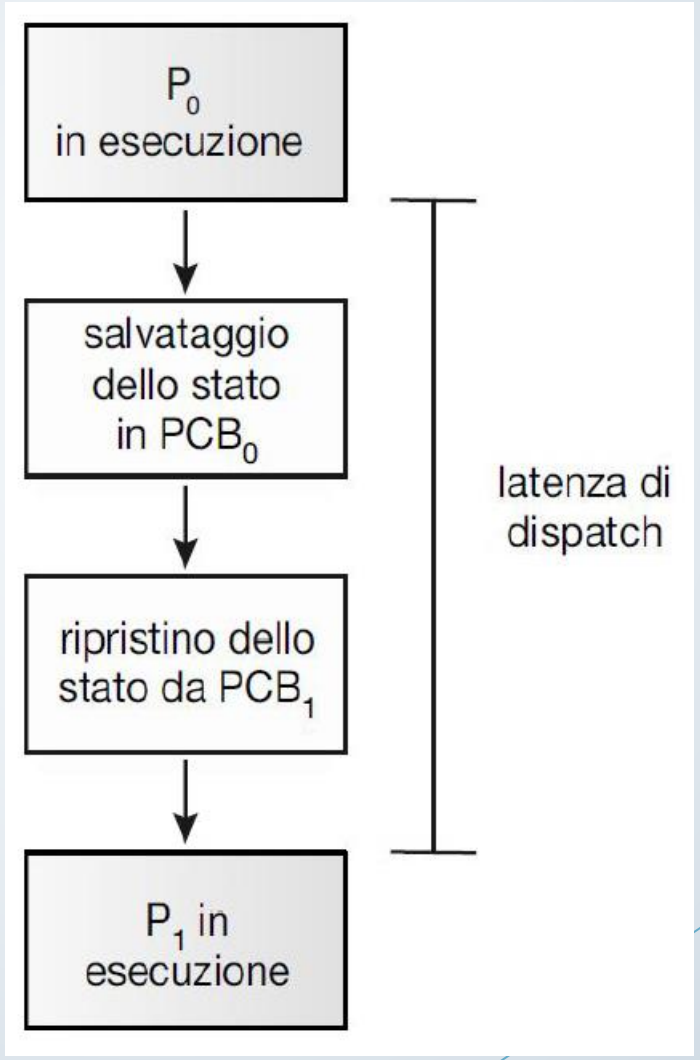
Dispatcher

- Altro elemento coinvolto nella funzione di scheduling è dispatcher. Responsabile di:
  - Cambio di contesto
  - Passaggio alla modalità utente
  - Riavviare esecuzione processo utente



Latenza di Dispatcher

- **Latenza di dispatch:** tempo richiesto dal dispatcher per fermare un processo e avviare l'esecuzione di un altro



Criteri di scheduling

Utilizzo della CPU

- La CPU deve essere più attiva possibile, questo valore viene espresso in percentuale.

Throughput (produttività)

- Rappresenta il numero di processi completati nell'unità di tempo.

Tempo di completamento

- Tempo che intercorre tra la sottomissione del processo e il completamento dell'esecuzione.

Criteri di scheduling

Tempo di attesa

- Somma degli intervalli d'attesa passati in questa coda. (ready queue).

Tempo di risposta

- Tempo che intercorre tra l'effettuazione di una richiesta e la prima risposta prodotta.

N.B. Le caratteristiche elencate in precedenza sono determinanti per la scelta dell'algoritmo di scheduling.

Algoritmi di scheduling

- Sceduling in ordine d'arrivo (FCFS first-come, first-served)
- Scheduling per brevità (SJF - shortest-job-first)
- Scheduling Circolare (RR – Round robin)
- Scheduling con priorità
- Scheduling a code multilivello
- Scheduling a code multilivello con retroazione

First-Come, First-Served FCFS

Il più semplice algoritmo di scheduling della CPU è l'algoritmo di **scheduling in ordine d'arrivo *(scheduling first-come, first-served o FCFS)***

- CPU assegnata al primo processo che la richiede
- Basato su una coda di tipo FIFO
- Tempo medio d'attesa elevato
- Algoritmo senza prelazione

First-Come, First-Served FCFS

Processo	Durata della sequenza
P1	24
P2	3
P3	3

Caso 1: Ordine di arrivo P1,P2,P3
Tempo medio di attesa: 17ms

Calcolo :

0ms per il processo P1
24ms per il processo P2
27ms per il processo P3

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

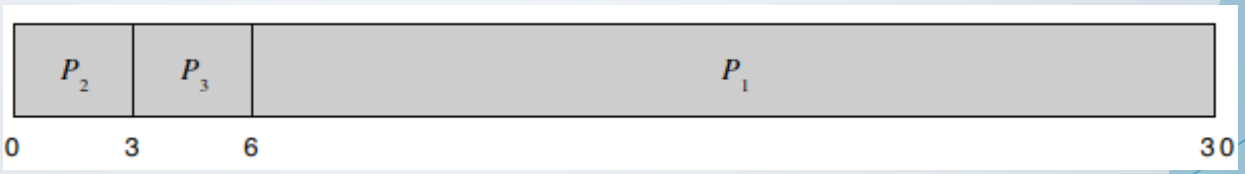
First-Come, First-Served FCFS

Processo	Durata della sequenza
P1	24
P2	3
P3	3

Se i processi fossero arrivati in ordine diverso?

Caso 1: Ordine di arrivo P2,P3,P1
Tempo medio di attesa: 3ms

Calcolo :
6ms per il processo P1
0 per il processo P2
3ms per il processo P3



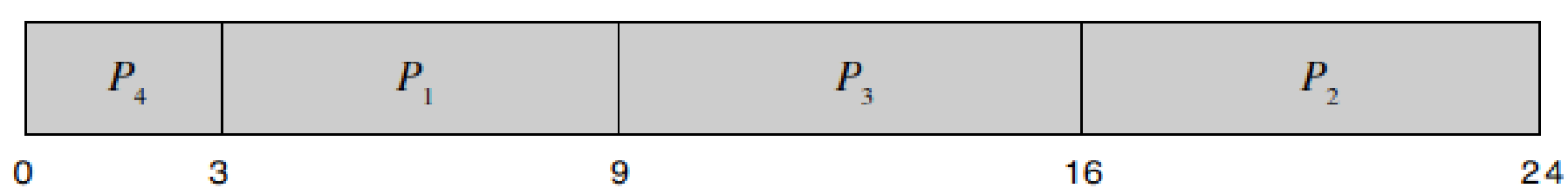
Tempo medio di attesa : (6 + 0 + 3) / 3 = 3ms

Shortest-job-first SJF

- L'**algoritmo di scheduling per brevità** (*shortest-job-first, **SJF)*** assegna la CPU al processo che ha la più breve lunghezza della successiva sequenza di operazioni della CPU. ***shortest next CPU burst.***
- Si esamina la lunghezza della successiva sequenza di operazioni della CPU del processo e non la sua lunghezza totale.
- l'algoritmo sjf non si può realizzare a livello dello scheduling della cpu a breve termine, poiché non esiste alcun modo per conoscere la lunghezza della successiva sequenza di operazioni della cpu. È possibile provare a fare una stima della durata.
- SJF può essere realizzato sia con prelazione che senza prelazione.

Shortest-job-first SJF nonpreemptive

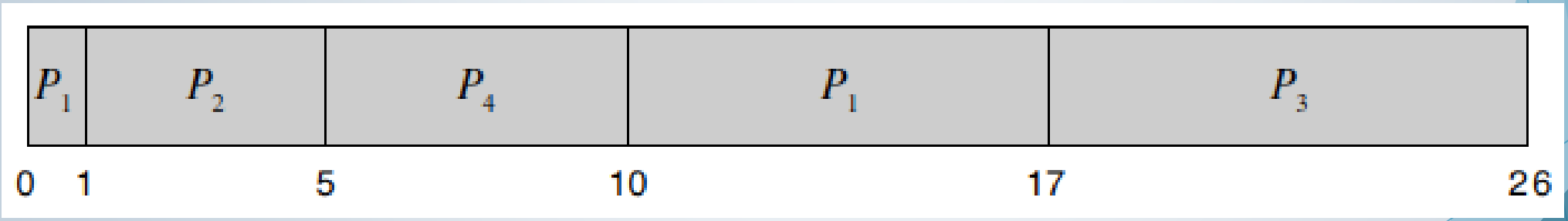
Processo	Durata della sequenza
P1	6
P2	8
P3	7
P4	3



Tempo medio di attesa senza prelazione: (0 + 3 + 9 + 16)/4 = 7.75 ms

Shortest-job-first SJF preemptive

Processo	Durata della sequenza	Instante d'arrivo
P1	8	0
P2	4	1
P3	9	2
P4	5	3



Tempo medio di attesa con prelazione: [(10-1) + (1-1) + (17-2) + (5-3)]/4 = 6.5 ms

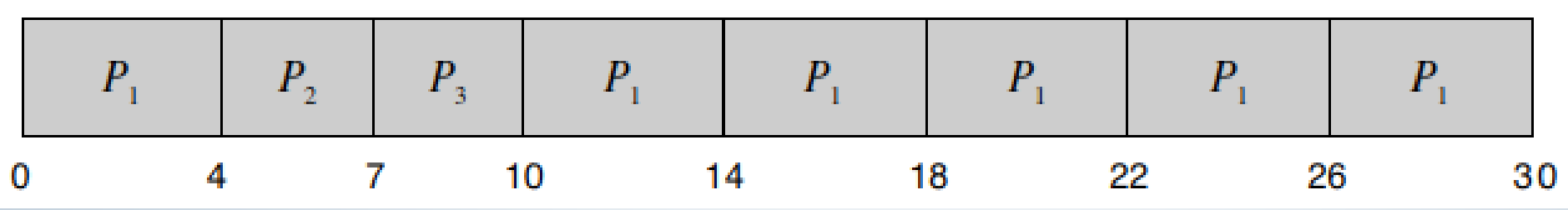
Round-Robin RR

- L'**algoritmo di scheduling circolare (*round-robin, RR*)** è simile allo scheduling FCFS (in ordine di arrivo), ma aggiunge la capacità di prelazione in modo che il sistema possa commutare fra i vari processi.

- Definizione di **quanto di tempo** o **porzione di tempo *(time slice)*** piccola quantità fissata del tempo della CPU ricevuta da un processo

Round-Robin RR

Processo	Durata della sequenza
P1	24
P2	3
P3	3

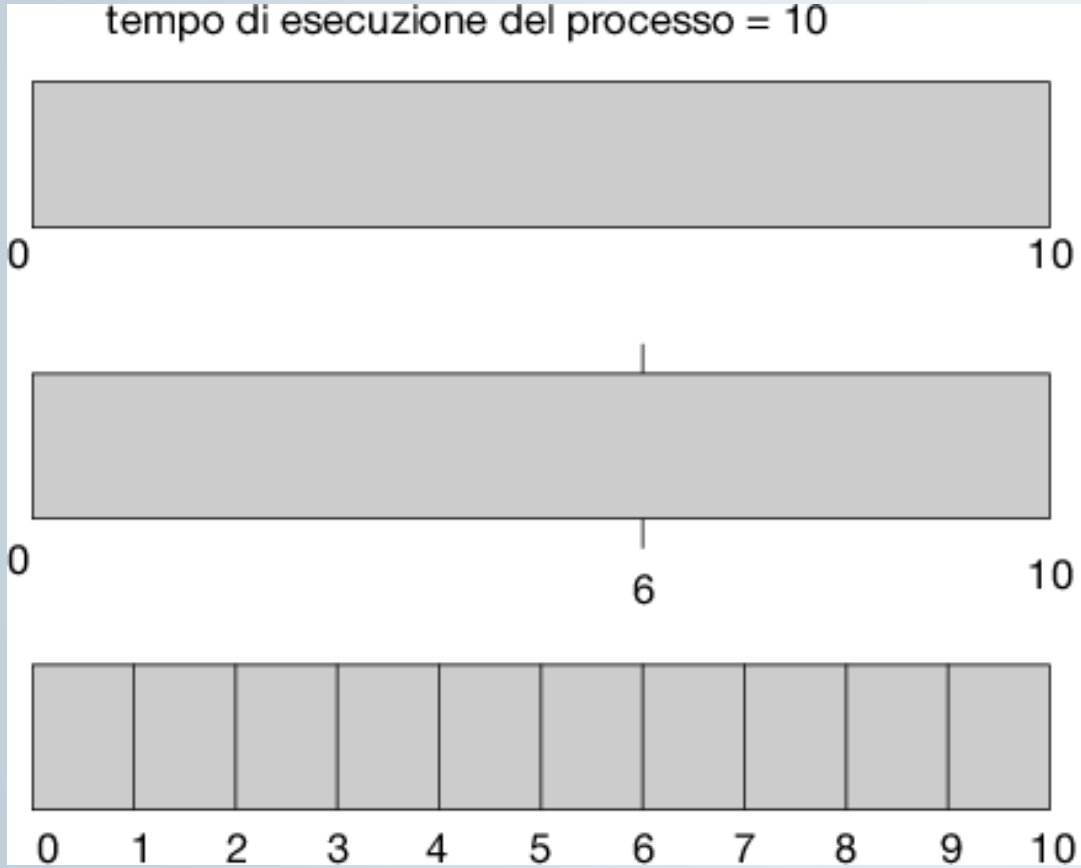


Tempo medio di attesa : [(10-4) + 4 + 7] / 3 = 5,66 ms

Round-Robin RR - quanto

- Le prestazioni dell'algoritmo RR dipendono molto dalla dimensione del quanto di tempo.
- Nel caso limite in cui il quanto di tempo sia molto lungo, il criterio di scheduling RR si riduce al criterio di scheduling FCFS.
- Se il quanto di tempo è molto breve (per esempio, un millisecondo), il criterio RR può portare a un numero elevato di cambi di contesto.

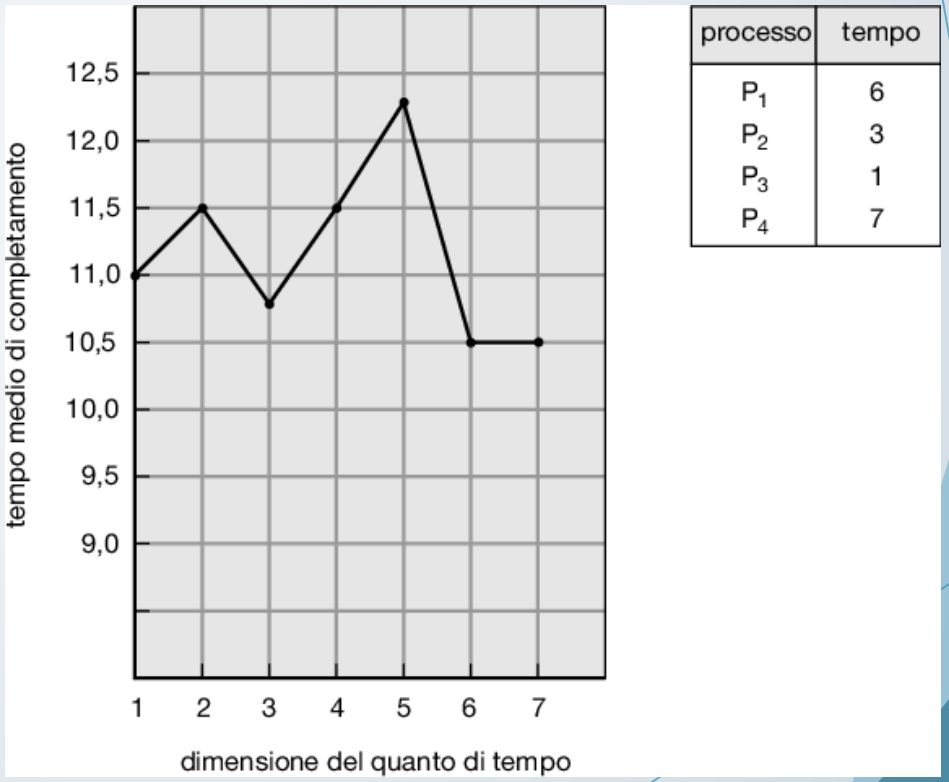
Round-Robin RR - quanto



lunghezza del quanto di tempo	cambio di contesto
12	0
6	1
1	9

Round-Robin RR – Tempo di completamento

Il **tempo di completamento *(turnaround time)*** dipende dalla dimensione del quanto di tempo: com'è evidenziato nella figura a lato, il tempo di completamento medio di un insieme di processi non migliora necessariamente con l'aumento della dimensione del quanto di tempo.



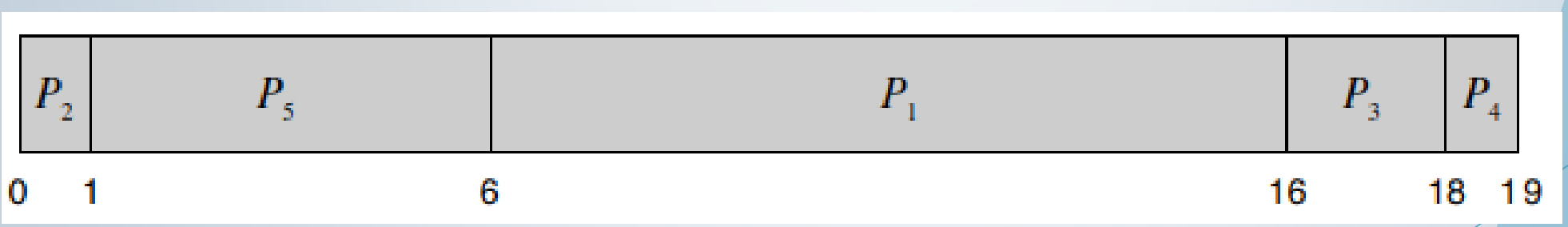
processo	tempo
P_1	6
P_2	3
P_3	1
P_4	7

Scheduling con priorità

- **Algoritmo di scheduling con priorità:** associa una priorità a ogni processo e assegna la CPU al processo con priorità più alta
- Sia con prelazione o senza prelazione
- Le priorità sono fissate da un intervallo fisso di numeri
- Per esempio, numeri bassi indicano priorità alte
- Può generare **starvation** (attesa indefinita) : processo pronto per l'esecuzione ma non dispone della CPU.
- Soluzione aging (invecchiamento) : aumento graduale della priorità dei processi che attendono nel sistema da parecchio tempo.

Scheduling con priorità

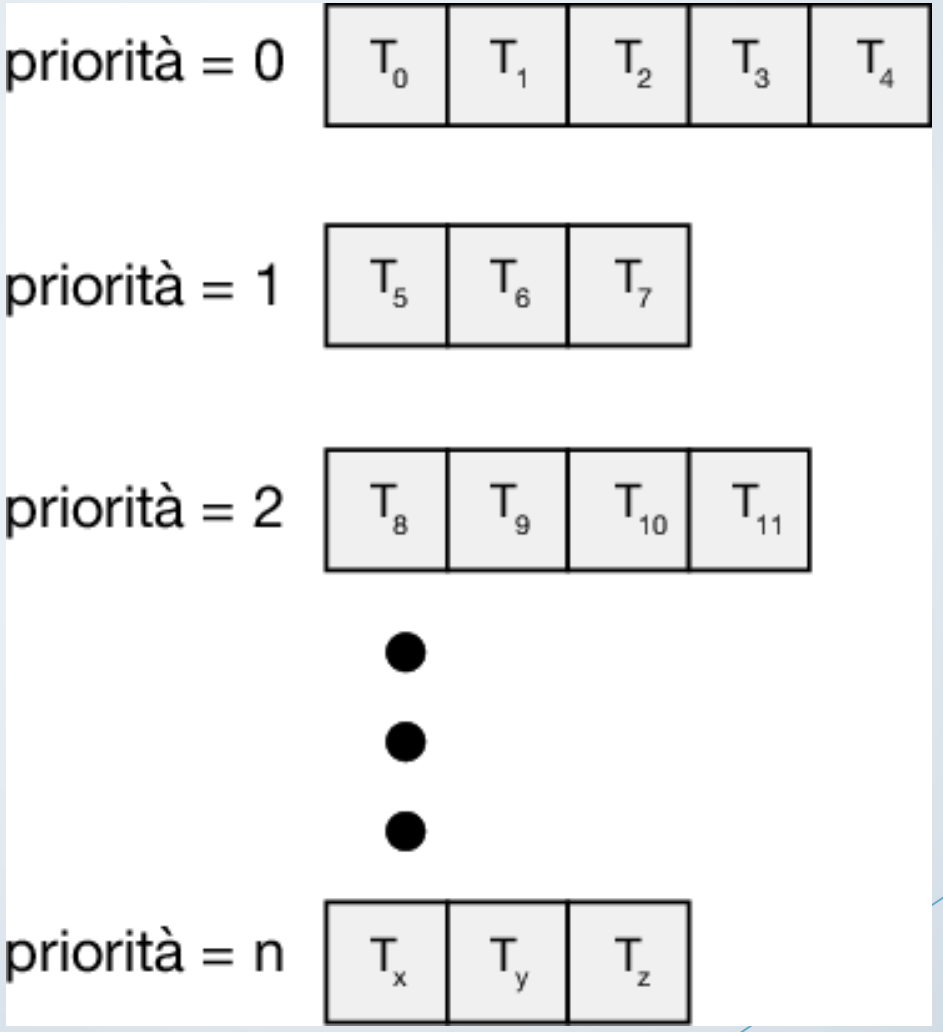
Processo	Durata della sequenza	Priorità
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



Tempo medio di attesa con prelazione: (0 + 1 + 6 + 16 + 18)/5 = 8,2 ms

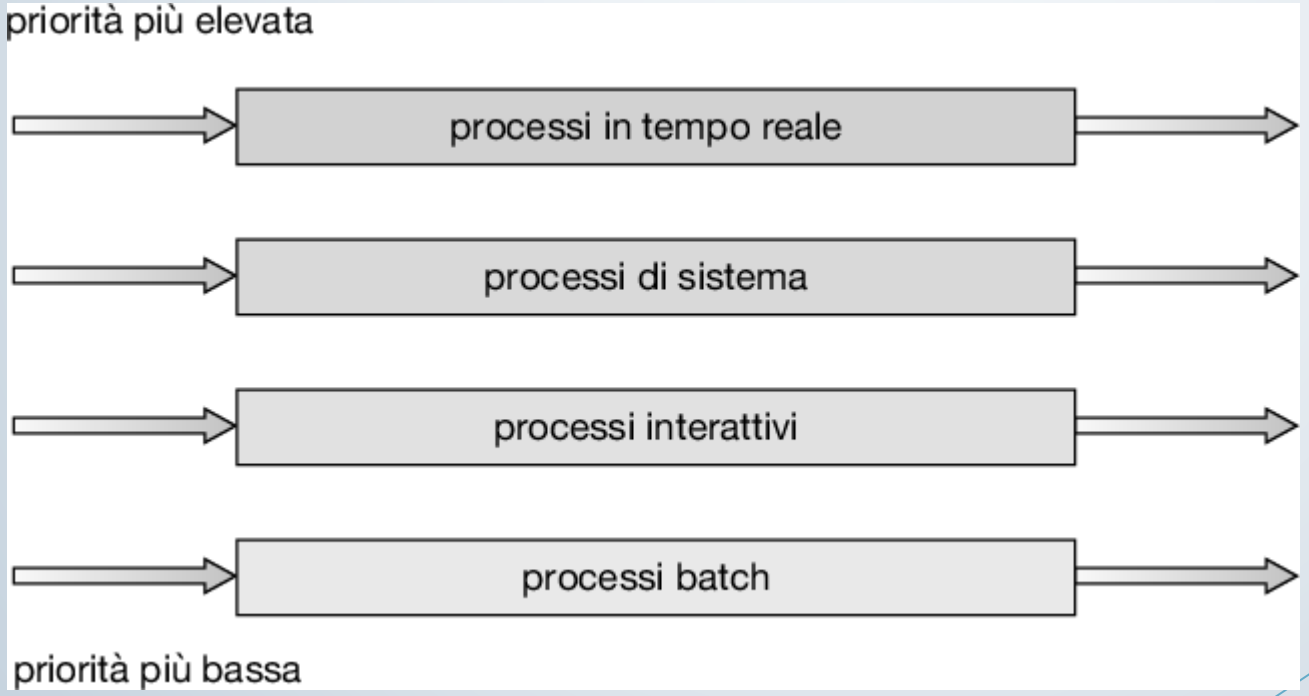
Scheduling a code multilivello

È spesso più semplice disporre di code separate per ciascuna priorità distinta e lasciare che lo scheduling con priorità si occupi semplicemente di selezionare il processo nella coda con priorità più alta



Scheduling a code multilivello

Un **algoritmo di scheduling a code multilivello** può anche essere utilizzato per suddividere i processi in diverse code in base al tipo di processo.

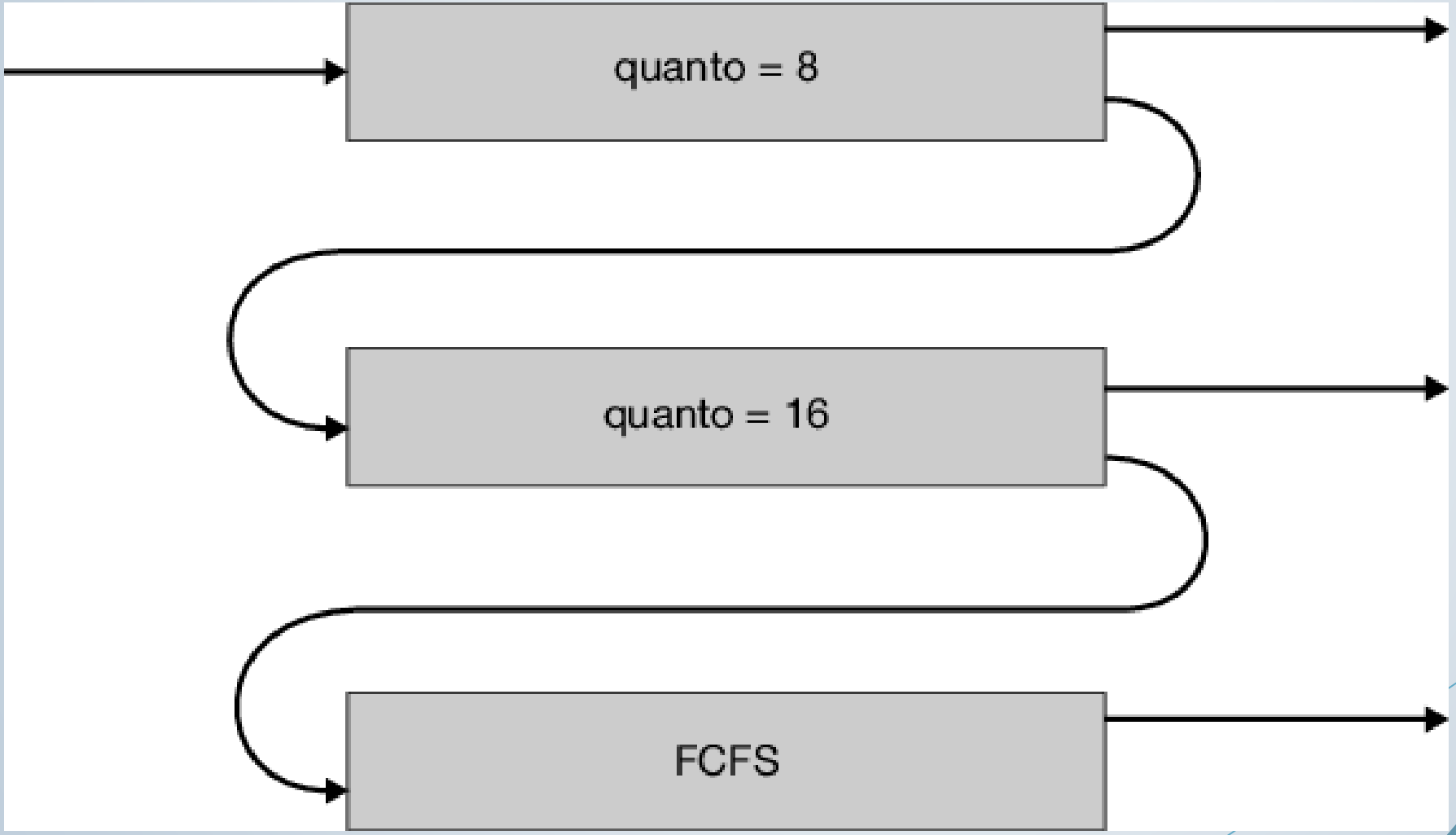


Scheduling a code multilivello con retroazione

Lo **scheduling a code multilivello con retroazione (*multilevel feedback queue scheduling)*** permette ai processi di spostarsi fra le code. È l'algoritmo più complesso. 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.

Scheduling a code multilivello con retroazione



Scheduling a code multilivello con retroazione

- Lo scheduling a code multilivello con retroazione (*multilevel feedback queue scheduling*), invece, permette ai processi di spostarsi fra le code. L'idea consiste nel separare i processi che hanno caratteristiche diverse in termini di cpu burst.
- Se un processo usa troppo tempo di elaborazione della CPU, viene spostato in una coda con priorità più bassa. Questo schema mantiene i processi con prevalenza di I/O e i processi interattivi nelle code con priorità più elevata. Inoltre, si può spostare in una coda con priorità più elevata un processo che attende troppo a lungo in una coda a priorità bassa. Questa forma di aging impedisce il verificarsi di un'attesa indefinita.

Scheduling dei thread

- ***A livello utente:*** ambito della contesa ristretto al processo *(process-contention scope, PCS)*
- ***A livello kernel:*** ambito della contesa allargato al sistema *(system-contention scope, SCS)*
- Nel caso del PCS, lo scheduling è solitamente basato sulle priorità: lo scheduler sceglie per l'esecuzione il thread con priorità più alta.
- La **API pthread di POSIX** consente di specificare *PCS* o *SCS* nella fase di generazione dei thread.

Scheduling dei thread – pthread API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
    int i, scope;
    pthread_t tid[NUM_THREADS];
    pthread_attr_t attr;
    /* ottiene gli attributi di default */
    pthread_attr_init(&attr);
    /* per prima cosa appura l'ambito della contesa */
    if (pthread_attr_getscope(&attr, &scope) != 0)
        fprintf(stderr, "Impossibile appurare l'ambito della contesa\n");
    else {
        if (scope == PTHREAD_SCOPE_PROCESS)
            printf("PTHREAD_SCOPE_PROCESS");
        else if (scope == PTHREAD_SCOPE_SYSTEM)
            printf("PTHREAD_SCOPE_SYSTEM");
        else
            fprintf(stderr, "Valore d'ambito della contesa non ammesso.\n");
   }
    /* imposta l'algoritmo di scheduling a PCS o SCS */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
    /* genera i thread */
    for (i = 0; i < NUM_THREADS; i++)
    pthread_create(&tid[i],&attr,runner,NULL);
    /* adesso aspetta la terminazione di tutti i thread */
    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(tid[i], NULL);
    }

/* ogni thread inizia l'esecuzione da questa funzione */
void *runner(void *param){
    /* fai qualcosa ... */
    pthread_exit(0);
}
```

Scheduling per sistemi multiprocessore

- Gli algoritmi di scheduling visto fino ad ora sono da considerarsi validi per tutti i sistemi con un singolo core di elaborazione, notoriamente i sistemi moderni sono caratterizzati da più core, con la possibilità di suddividere il **carico di lavoro (load sharing)** e di conseguenza aumentare la complessità dello scheduling.

Scheduling per sistemi multiprocessore

Il termine **multiprocessore** si applica attualmente alle seguenti architetture di sistema:

- CPU multicore
- Core Multithread
- Sistemi NUMA
- Sistemi multiprocessore eterogenei

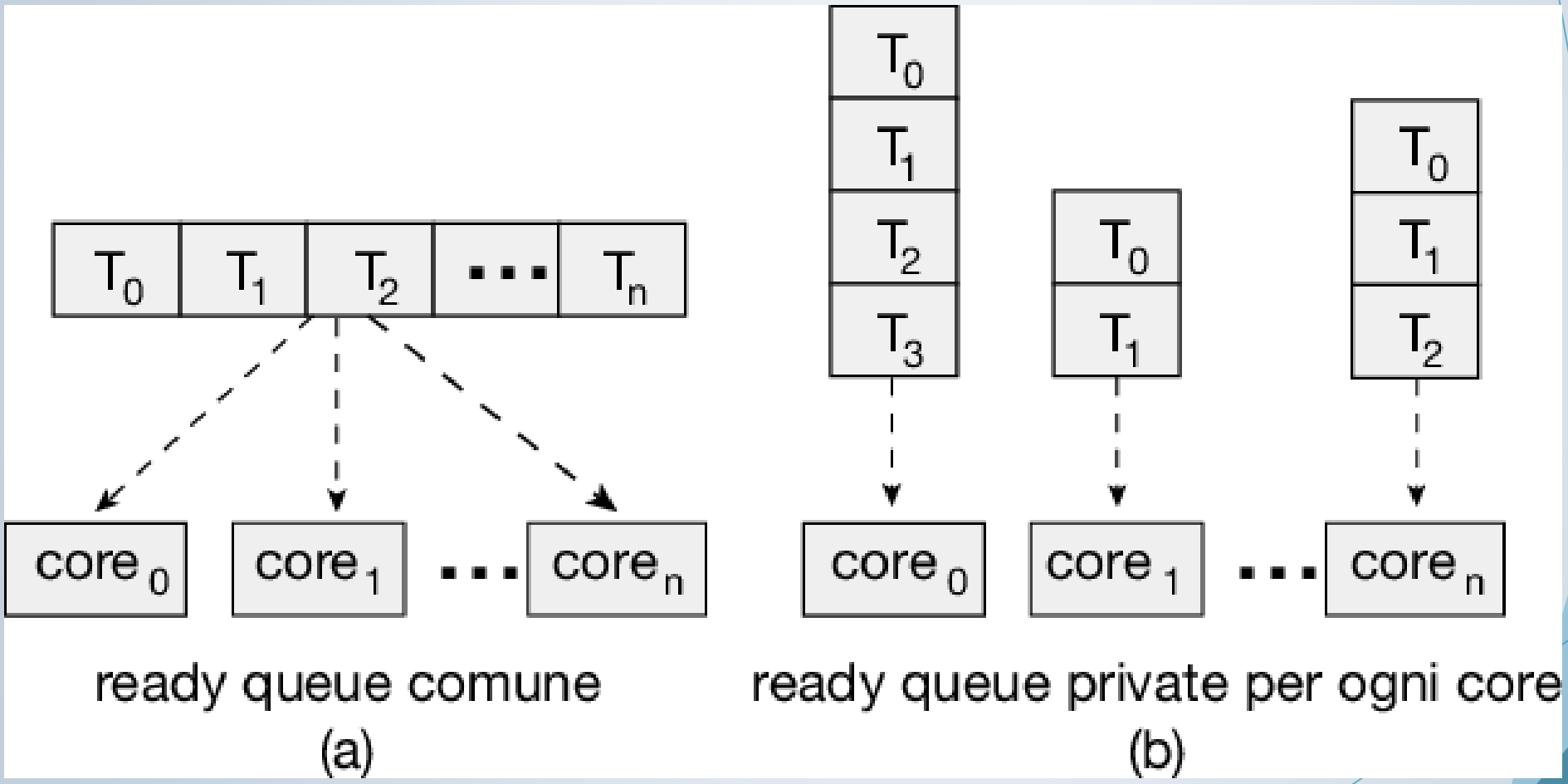
Scheduling per sistemi multiprocessore

L’approccio standard per supportare i multiprocessori è la **multielaborazione simmetrica (SMP),** in cui ciascun processore è in grado di autogestirsi La SMP offre due possibili strategie per organizzare i thread da selezionare per l’esecuzione:

a) Tutti i thread possono trovarsi in una ready queue comune;

b) Ogni processore può avere una propria coda privata di thread.

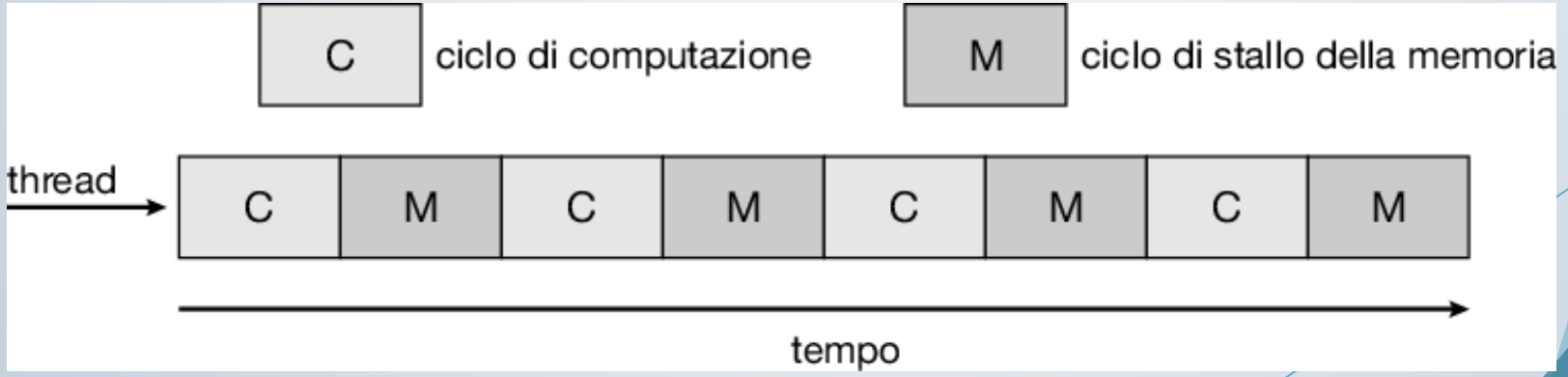
Scheduling per sistemi multiprocessore



Scheduling per sistemi multicore

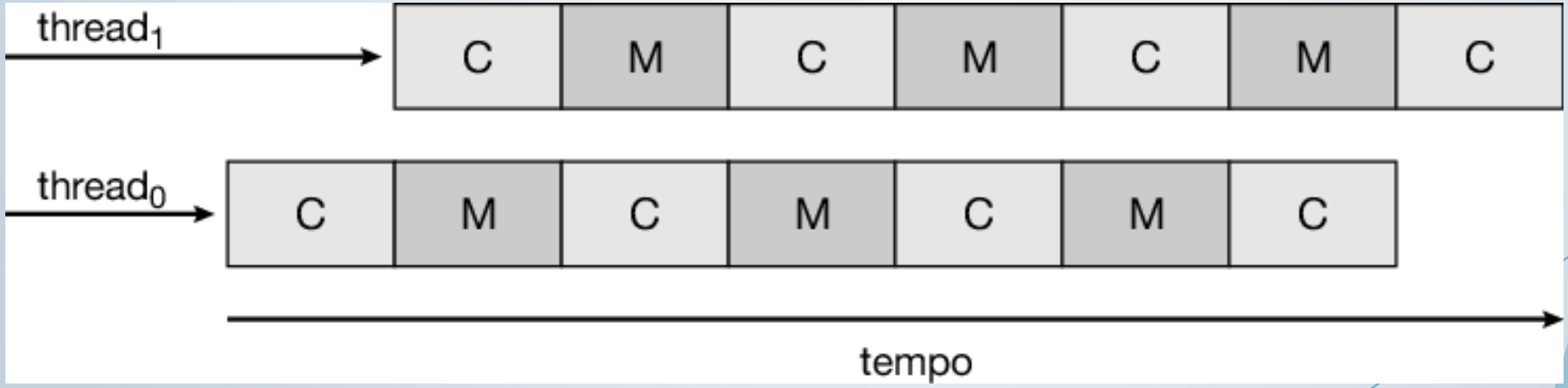
- Inserire più core di elaborazione in un unico chip fisico : **processore multicore**

- Quando un processore accede alla memoria, una quantità significativa di tempo trascorre in attesa della disponibilità dei dati : **stallo della memoria**



Scheduling per sistemi multicore

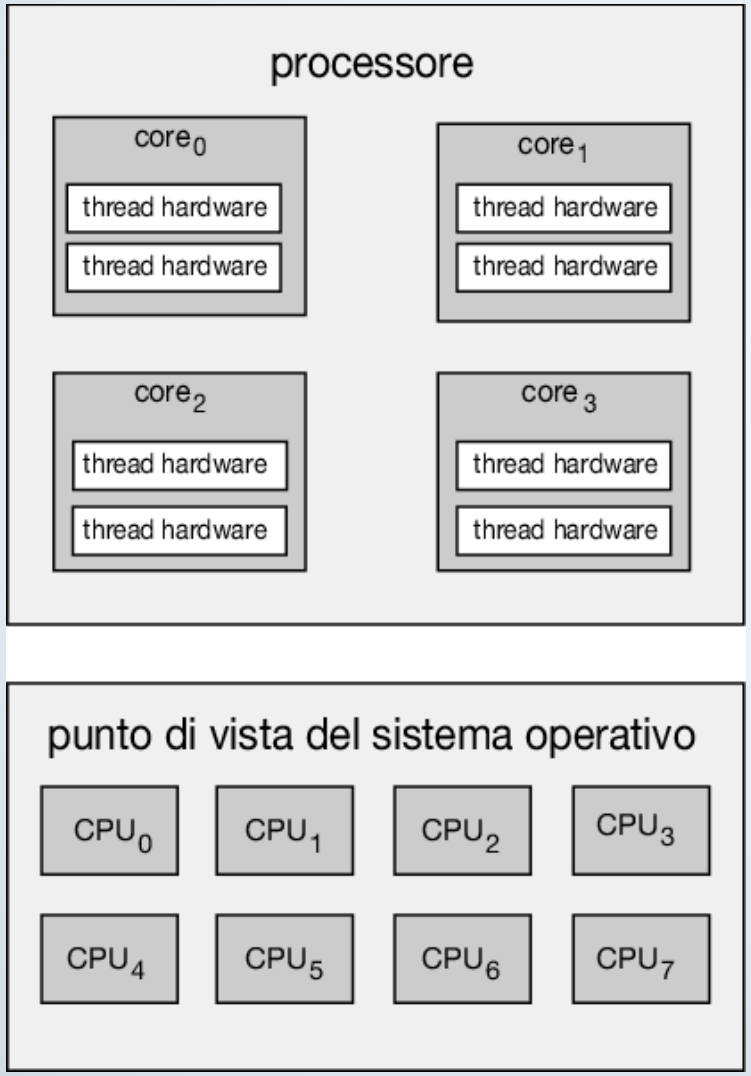
- Per rimediare a questa situazione, molti dei progetti hardware recenti implementano delle unità di calcolo multithread in cui due o più thread hardware sono assegnati a un singolo core. In questo modo, se un thread è in situazione di stallo in attesa della memoria, il core può passare a eseguire un altro thread



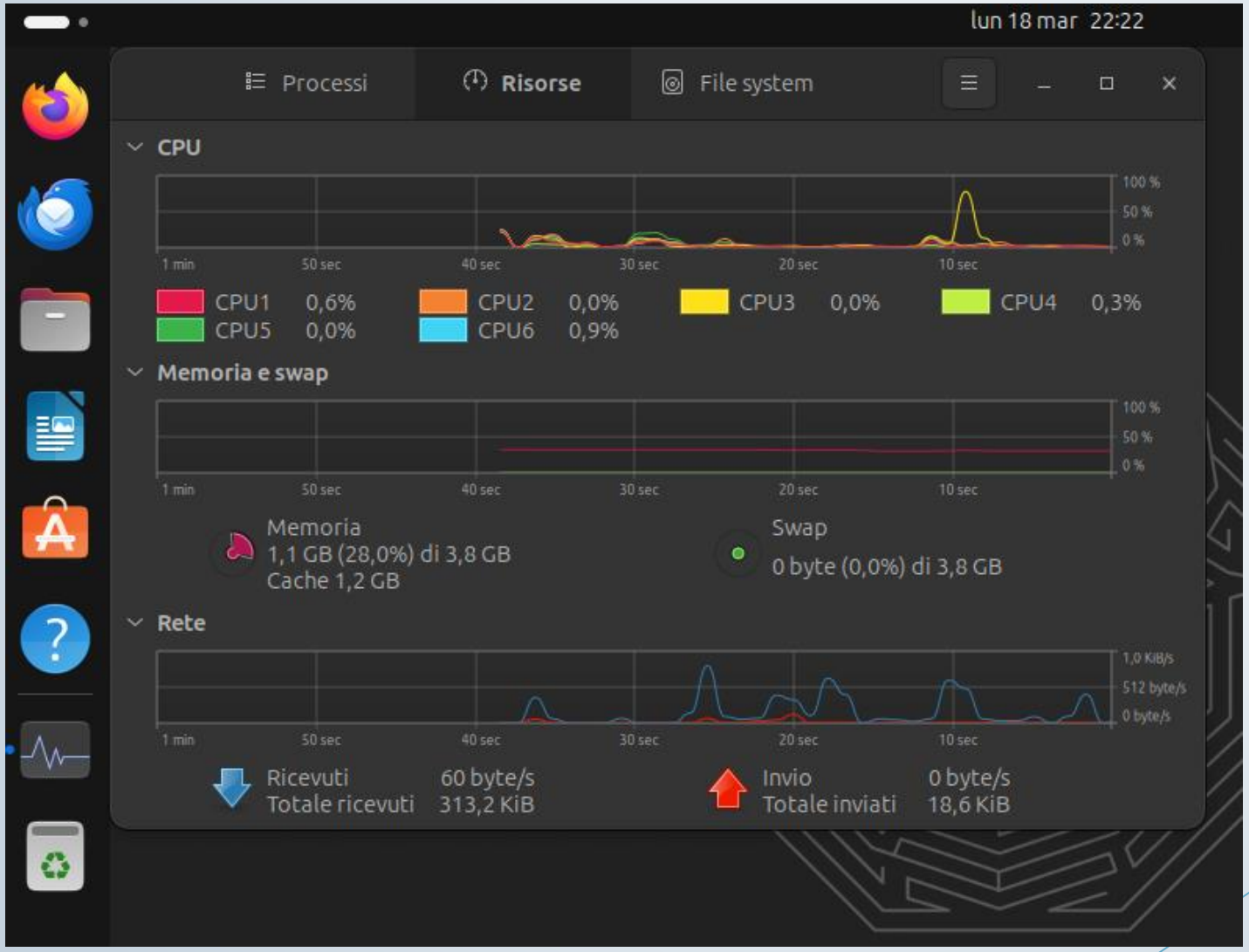
Hyper Threading – chip CMT

- **chip multithreading (CMT) o hyper-threading** un processore contiene quattro *core di elaborazione*, ognuno dei quali contiene due *thread hardware*: dal punto di vista del sistema operativo sono presenti otto **CPU logiche**.
- I processori Intel utilizzano il termine hyper-threading per descrivere l'assegnazione di più thread hardware a un singolo core di elaborazione. I processori Intel contemporanei, come l'i7, supportano due thread per core, mentre il processore Oracle Sparc M7 supporta otto thread per core, con otto core per processore, fornendo così al sistema operativo 64 cpu logiche.

Hyper Threading – chip CMT

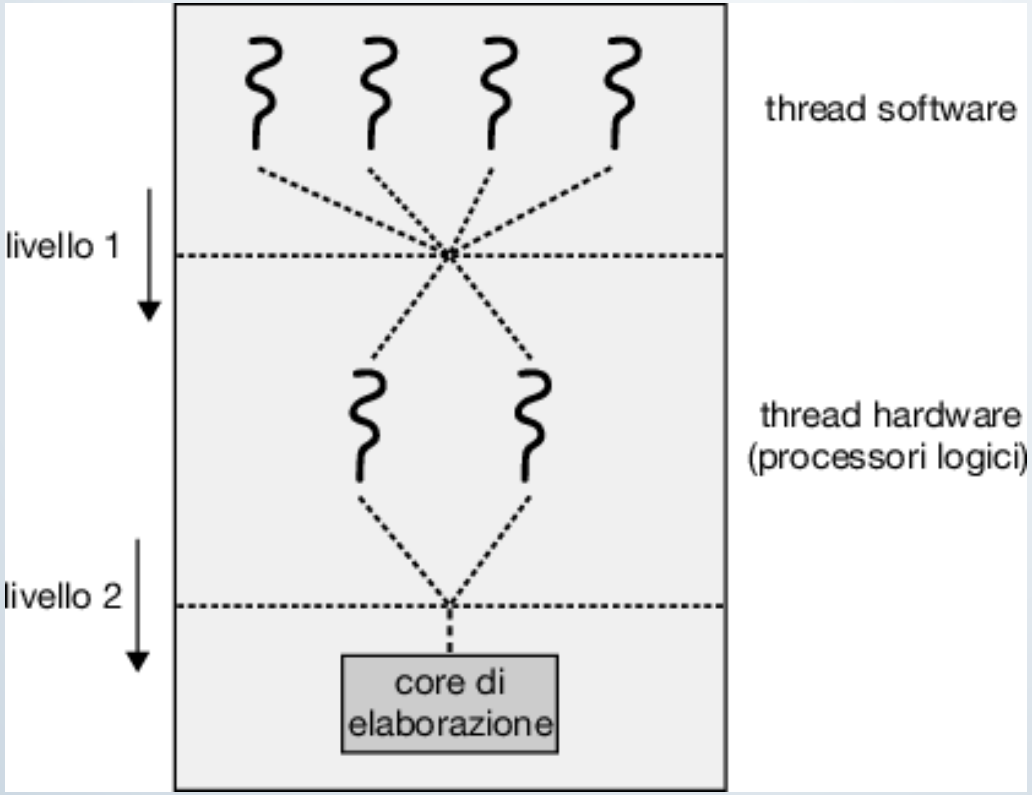


Gnome System Monitor



Scheduling a due livelli

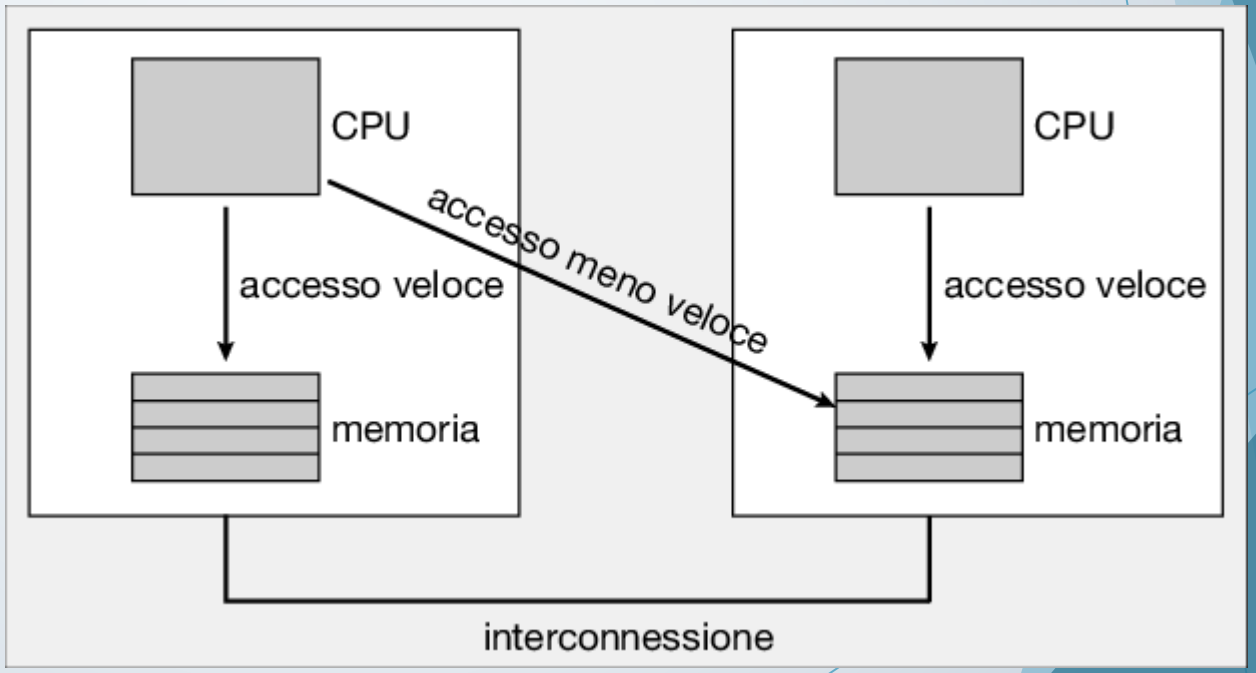
Un processore *multithreaded* e *multicore* richiede due diversi livelli di scheduling, che illustra un core di elaborazione dual-threaded.



NUMA e scheduling della CPU

Predilezione per il processore *(processor affinity)* *:* un processo ha una predilezione per il processore su cui è in esecuzione.

Un’architettura con accesso non uniforme alla memoria (**NUMA**) in cui sono presenti due chip fisici di processore, ciascuno con la propria CPU e memoria locale.



Scheduling real-time della CPU

- I sistemi real-time sono per loro natura guidati dagli eventi: generalmente, il sistema attende che si verifichi un evento in tempo reale.

Scheduling real-time della CPU

Sistemi soft real-time

- non offrono garanzie sul momento in cui un processo critico sarà eseguito, ma assicurano solamente che sarà data precedenza a quest'ultimo piuttosto che ad altri processi non critici.

Sistemi hard real-time

- hanno vincoli più rigidi: i task vanno eseguiti entro una scadenza prefissata ed eseguirli dopo tale scadenza è del tutto inutile.

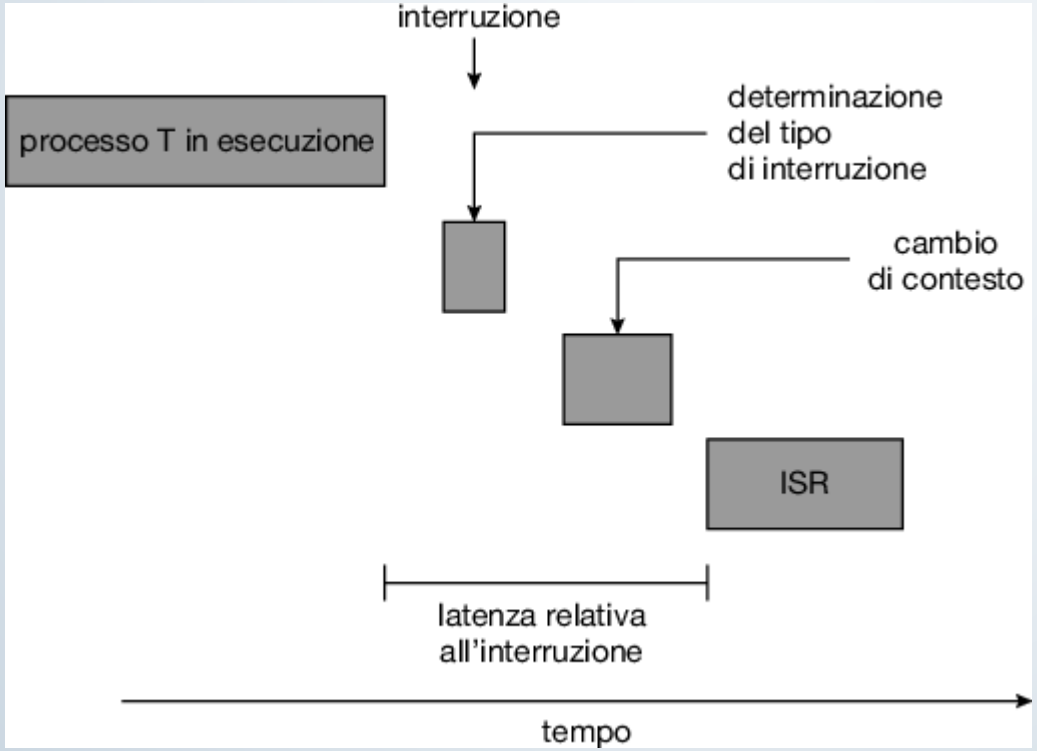
Latenza nei sistemi real-time

Le categorie di latenza che influiscono sul funzionamento dei sistemi real-time sono:

- **Latenza relativa alle interruzioni**
- **Latenza relativa al dispatch**

Latenza relativa alle interruzioni

Si riferisce al periodo di tempo compreso tra la notifica di un'interruzione alla CPU e l'avvio della routine che gestisce l'interruzione (ISR)

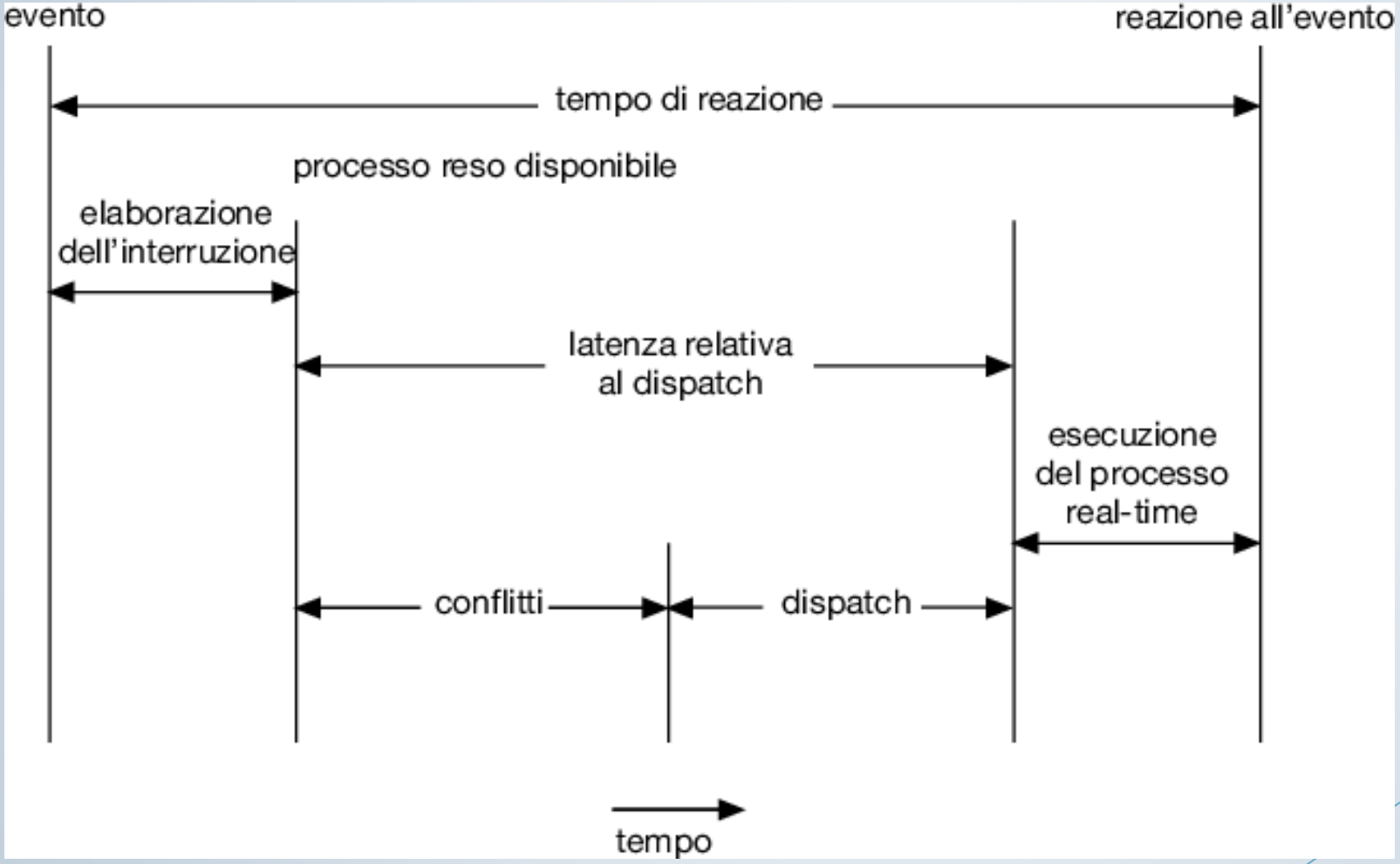


Latenza relativa alle interruzioni

latenza relativa al dispatch: periodo di tempo necessario al dispatcher per bloccare un processo e avviarne un altro. La *fase di conflitto della latenza di dispatch* consiste di due componenti:

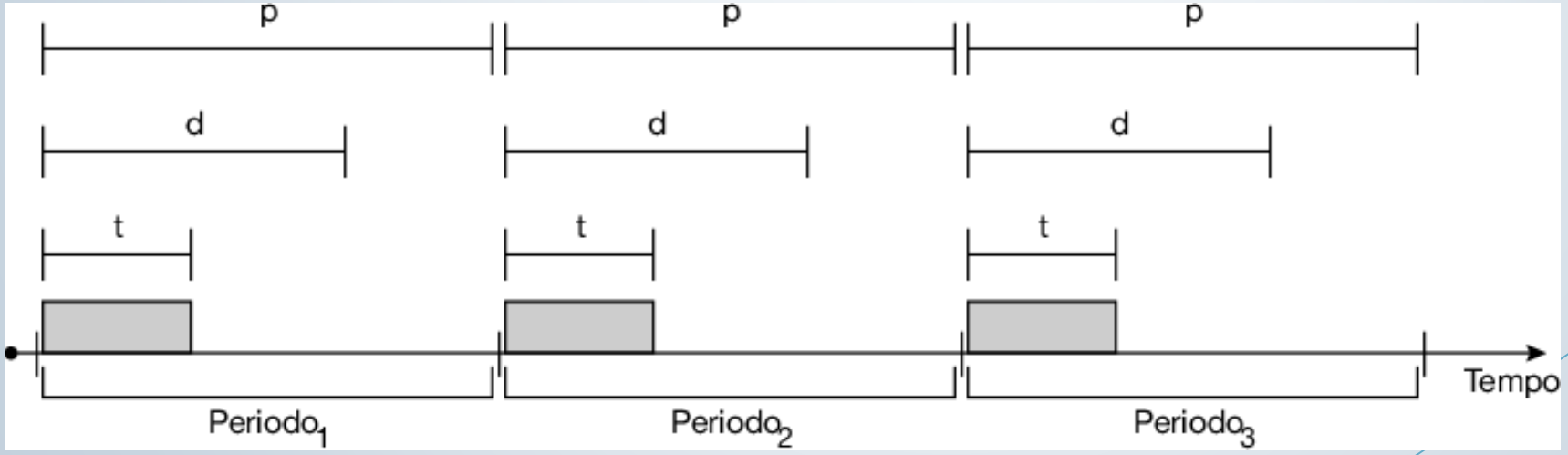
- 1. prelazione di ogni processo in esecuzione nel kernel;
- 2. cessione, da parte dei processi a bassa priorità, delle risorse richieste dal processo ad alta priorità.

Latenza relativa alle interruzioni



Scheduling basato sulla priorità

- Gli algoritmi di scheduling con priorità assegnano a ogni processo una priorità in base alla loro importanza
- I processi sono considerati **periodici**, nel senso che richiedono la CPU a intervalli costanti di tempo (periodi)



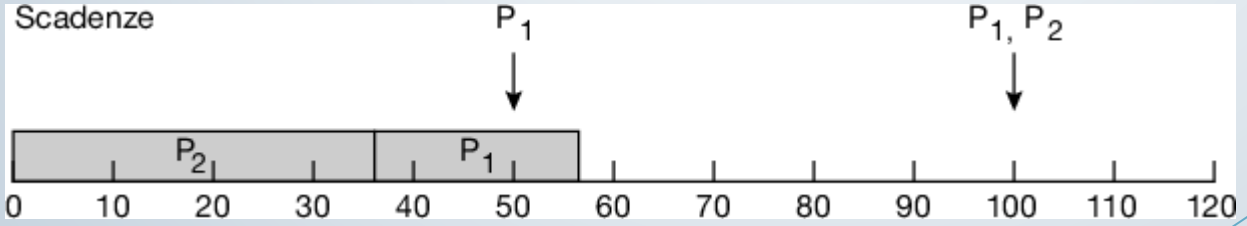
Scheduling proporzionale alla frequenza

Ciascun task periodico si vede assegnare una priorità inversamente proporzionale al proprio periodo:

- Più breve è il periodo, più alta la priorità;
- Più lungo il periodo, più bassa la priorità.
- Eseguo prima P2 e poi P1

P1 con periodo p1 = 50 e
tempo di elaborazione t1 = 20

P2 con periodo p2 = 100 e
tempo di elaborazione t2 = 35



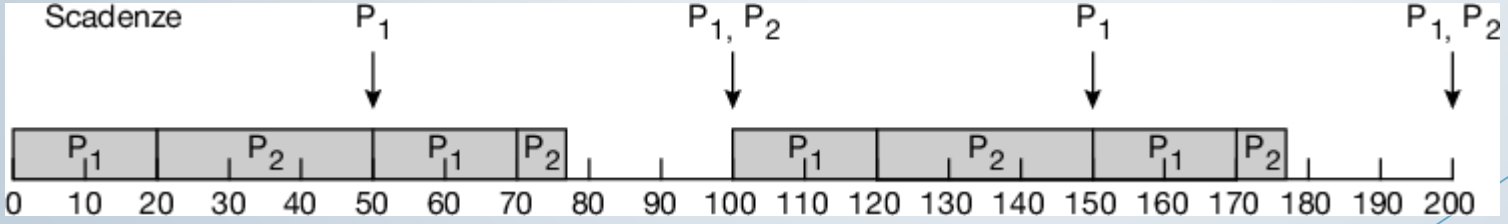
Scheduling proporzionale alla frequenza

Ciascun task periodico si vede assegnare una priorità inversamente proporzionale al proprio periodo:

- Più breve è il periodo, più alta la priorità;
- Più lungo il periodo, più bassa la priorità.
- Prima P1 poi P2 per precedenza di durata

P1 con periodo p1 = 50 e
tempo di elaborazione t1 = 20

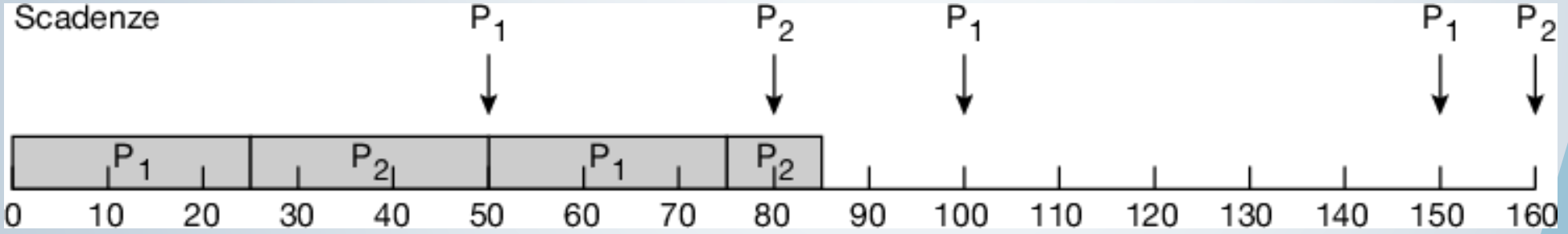
P2 con periodo p2 = 100 e
tempo di elaborazione t2 = 35



Scheduling EDF

Lo **scheduling EDF** *(earliest-deadline-first, ossia "per prima la scadenza più ravvicinata")*, attribuisce le priorità dinamicamente, sulla base delle scadenze.

- Più vicina è la scadenza, maggiore è la priorità



A differenza dell'algoritmo con priorità proporzionale alla frequenza, lo scheduling EDF non postula la periodicità dei processi, e non prevede neanche di impiegare sempre la stesso tempo della CPU per ogni burst.

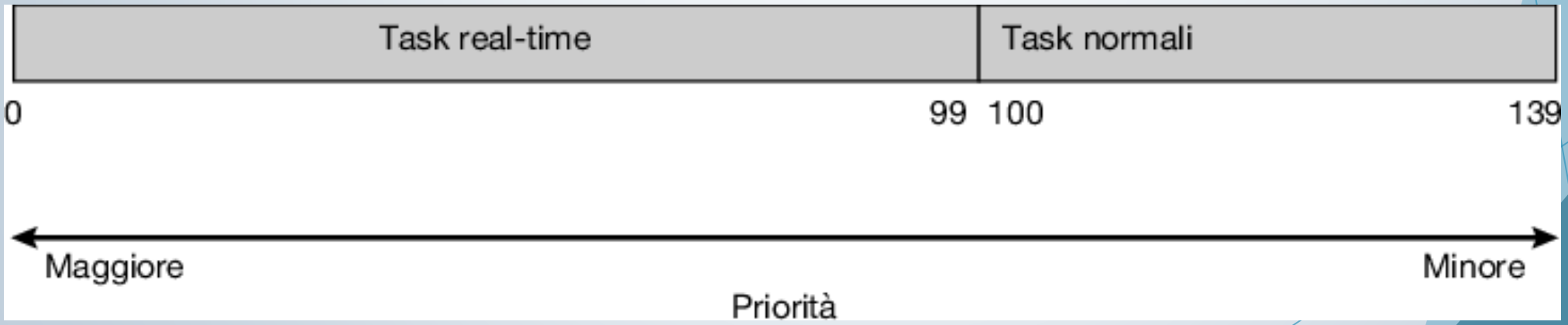
Criteri di scheduling per OS

Linux

Windows

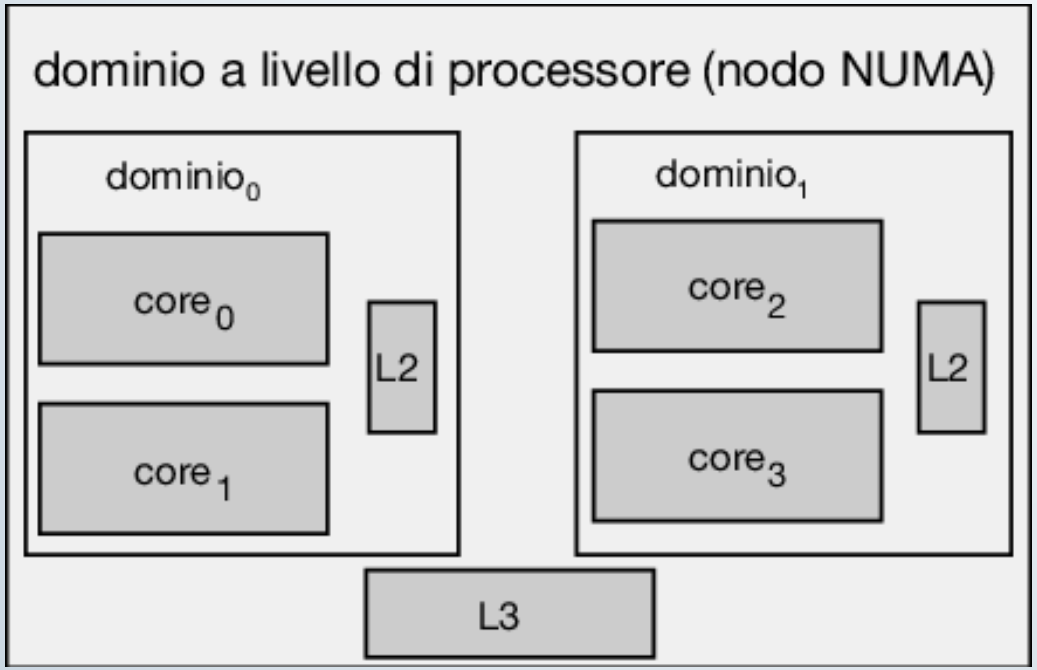
Scheduling in Linux

Nei sistemi Linux lo scheduling si basa sulle **classi di scheduling** Per decidere quale task eseguire, lo scheduler seleziona il task con priorità più alta appartenente alla classe di scheduling a priorità più elevata.



Scheduling in Linux

Un **dominio di scheduling** è un insieme di core che può essere bilanciato l'uno con l'altro. I core in ciascun dominio di scheduling sono raggruppati in base al modo in cui condividono le risorse del sistema.



Scheduling in Windows

- Lo scheduler di Windows assicura che si eseguano sempre i thread a più alta priorità.
- La porzione del kernel che si occupa dello scheduling si chiama ***dispatcher***
- Le priorità sono suddivise in due classi:
  - la **classe variable :** raccoglie i thread con priorità da 1 a 15,
  - la **classe real-time :** raccoglie i thread con priorità tra 16 e 31
- La priorità di ciascun thread dipende dalla priorità della classe cui appartiene e dalla priorità relativa che il thread ha all'interno della stessa classe.

Valutazione degli algoritmi

Metodi di valutazione dell'algoritmo di scheduling della CPU:

1. Valutazione analitica – partendo dall'algoritmo dato e dal carico di lavoro del sistema, fornisce una formula o un numero che valuta le prestazioni dell'algoritmo per quel carico di lavoro.

2. Modellazione deterministica – è un tipo di valutazione analitica, che considera un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per quel carico di lavoro.

Valutazione degli algoritmi

Metodi di valutazione dell'algoritmo di scheduling della CPU:

3. Analisi delle reti di code - Il sistema di calcolo si descrive come una rete di server, ciascuno con una coda d'attesa. La cpu è un server con la propria ready queue, così come il sistema di i/o con le sue code di attesa dei dispositivi.

4. Simulazioni - Le simulazioni implicano la programmazione di un modello del sistema di calcolo; le strutture dati rappresentano gli elementi principali del sistema.

Valutazione degli algoritmi

Metodi di valutazione dell'algoritmo di scheduling della CPU:

5. Codifica dell'algoritmo di scheduling - L'unico modo assolutamente preciso per valutare un algoritmo di scheduling consiste nel codificarlo, inserirlo nel sistema operativo e osservarne il comportamento nelle reali condizioni di funzionamento del sistema.

Riferimenti - fonti

1. © Pearson Italia S.p.A. – Silberschatz, Galvin, Gagne, *Sistemi operativi*