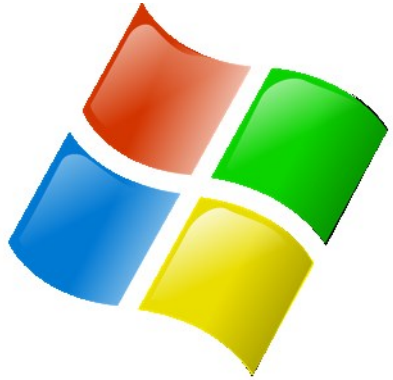
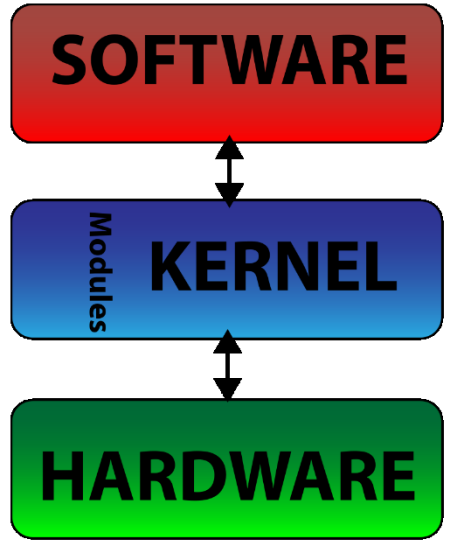
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

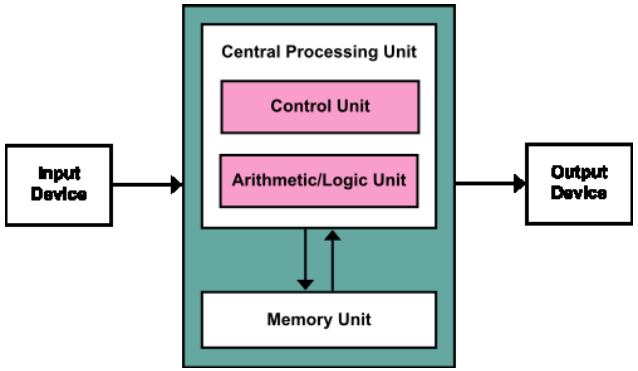


Corso di Sistemi Operativi

Scheduling della CPU



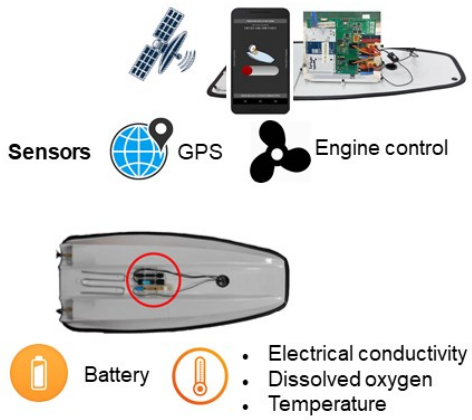
Docente:
Domenico Daniele Bloisi

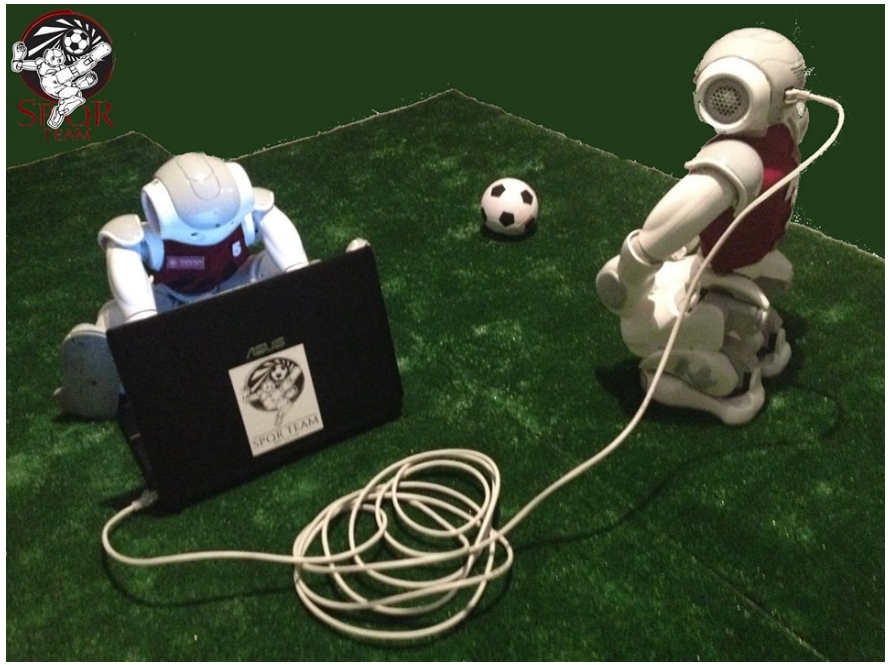


Domenico Daniele Bloisi

- Ricercatore RTD B
  Dipartimento di Matematica, Informatica ed Economia
  Università degli studi della Basilicata
  http://web.unibas.it/bloisi

- SPQR Robot Soccer Team
  Dipartimento di Informatica, Automatica e Gestionale Università degli studi di Roma “La Sapienza”
  http://spqr.diag.uniroma1.it





Informazioni sul corso

- Home page del corso:
  http://web.unibas.it/bloisi/corsi/sistemi-operativi.html
- Docente: Domenico Daniele Bloisi
- Periodo: I semestre ottobre 2020 – febbraio 2021

  - Lunedì 15:00-17:00
  - Martedì 9:30-11:30

Le lezioni saranno erogate in modalità esclusivamente on-line

Codice corso Google Classroom:
https://classroom.google.com/c/MTQ2ODE2NTk3ODIz?cjc=67646ik

Ricevimento

- Su appuntamento tramite Google Meet

Per prenotare un appuntamento inviare una email a domenico.bloisi@unibas.it

Programma – Sistemi Operativi

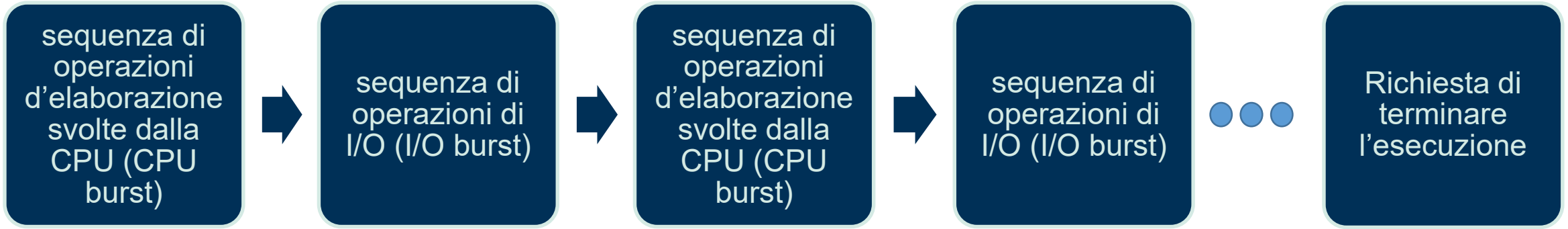
- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

Scheduling

- Lo scheduling è una funzione fondamentale dei sistemi operativi.
- Si sottopongono a scheduling quasi tutte le risorse di un calcolatore.
- La CPU è una delle risorse principali e il suo scheduling è alla base della progettazione dei sistemi operativi.

Ciclicità delle fasi d'elaborazione e di I/O

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



Ciclicità delle fasi d'elaborazione e di I/O

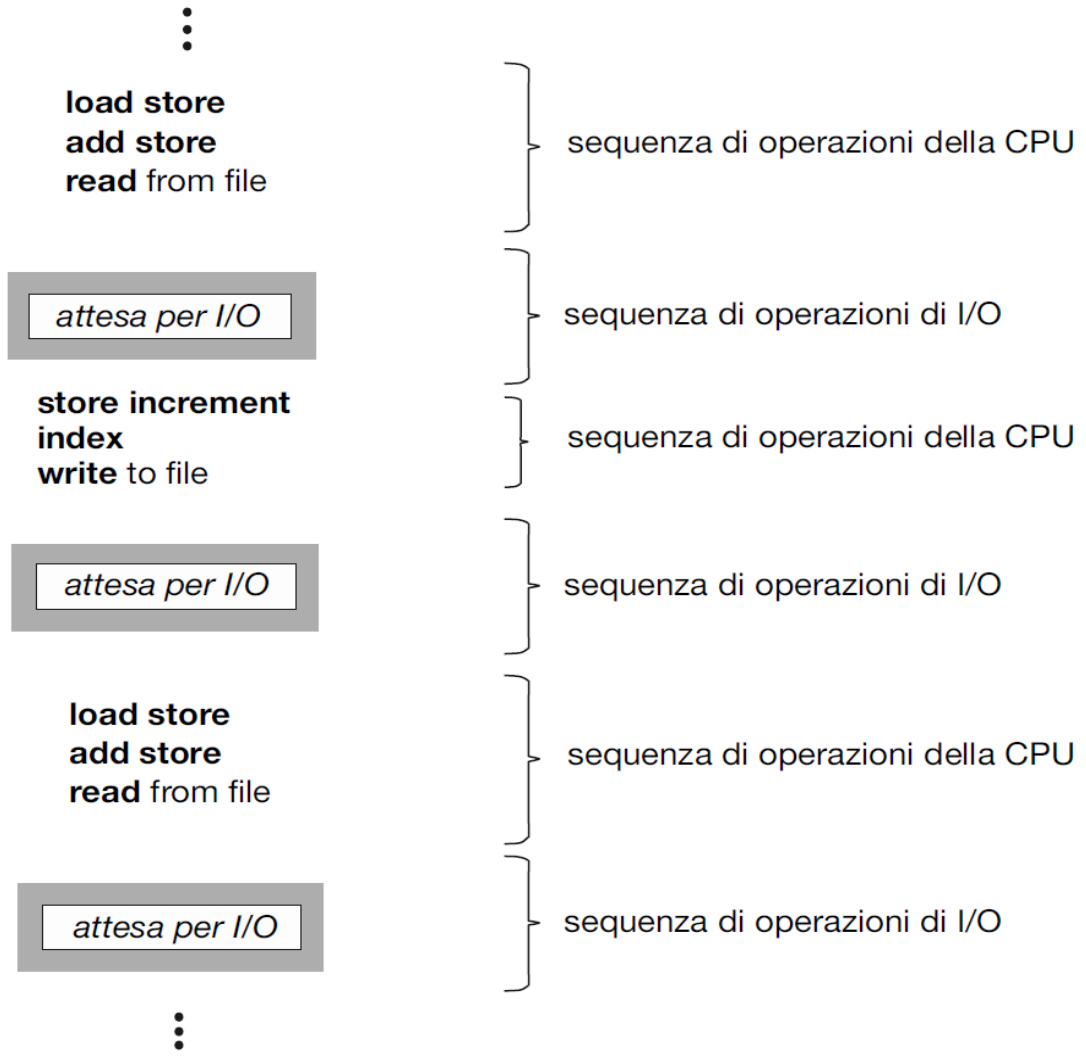


Figura 5.1 Serie alternata di sequenze di operazioni della CPU e di sequenze di operazioni di I/O.

Ciclicità delle fasi d'elaborazione e di I/O

La curva di frequenza dei burst della CPU evidenzia la presenza di:

- molte brevi sequenze di operazioni della CPU
- poche sequenze della CPU molto lunghe

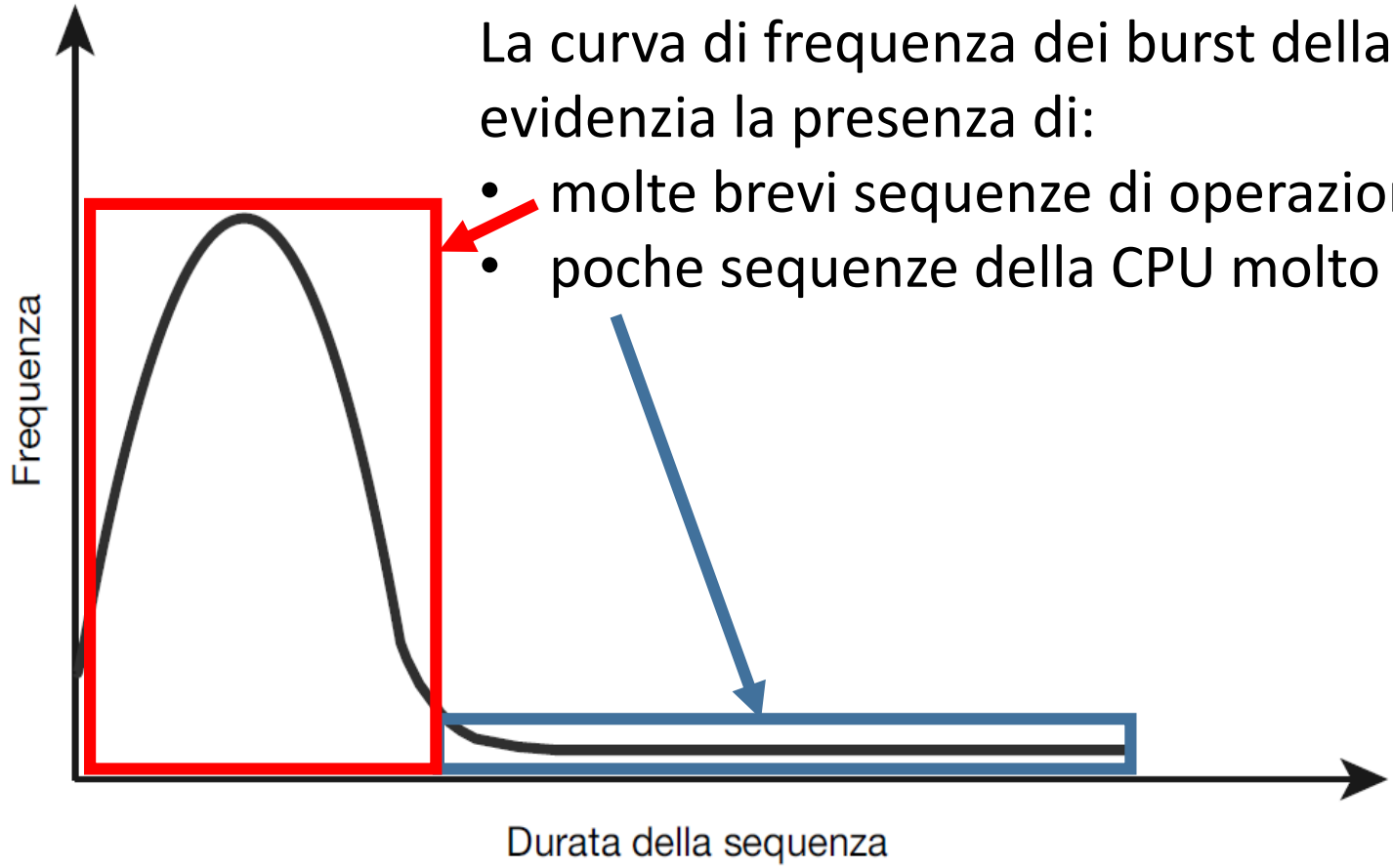


Figura 5.2 Diagramma delle durate delle sequenze di operazioni della CPU.

Lo scheduler

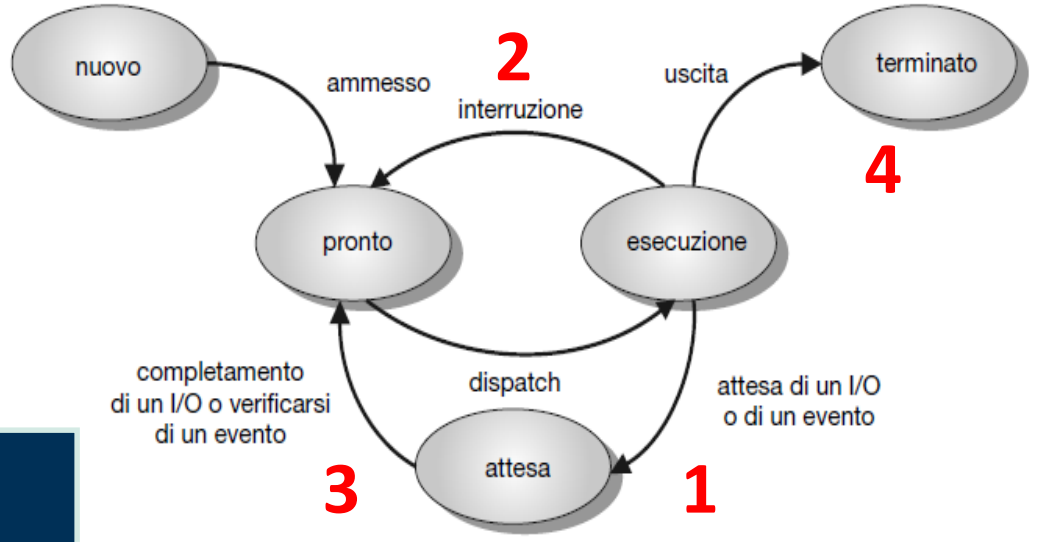
Lo scheduler interviene nelle seguenti situazioni:

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



Scheduling con e senza prelazione

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

Schemi di scheduling:

- **senza prelazione (*nonpreemptive*) o cooperativo (*cooperative*);**
- **con prelazione (*preemptive*)**

Solo casi 1 e 4

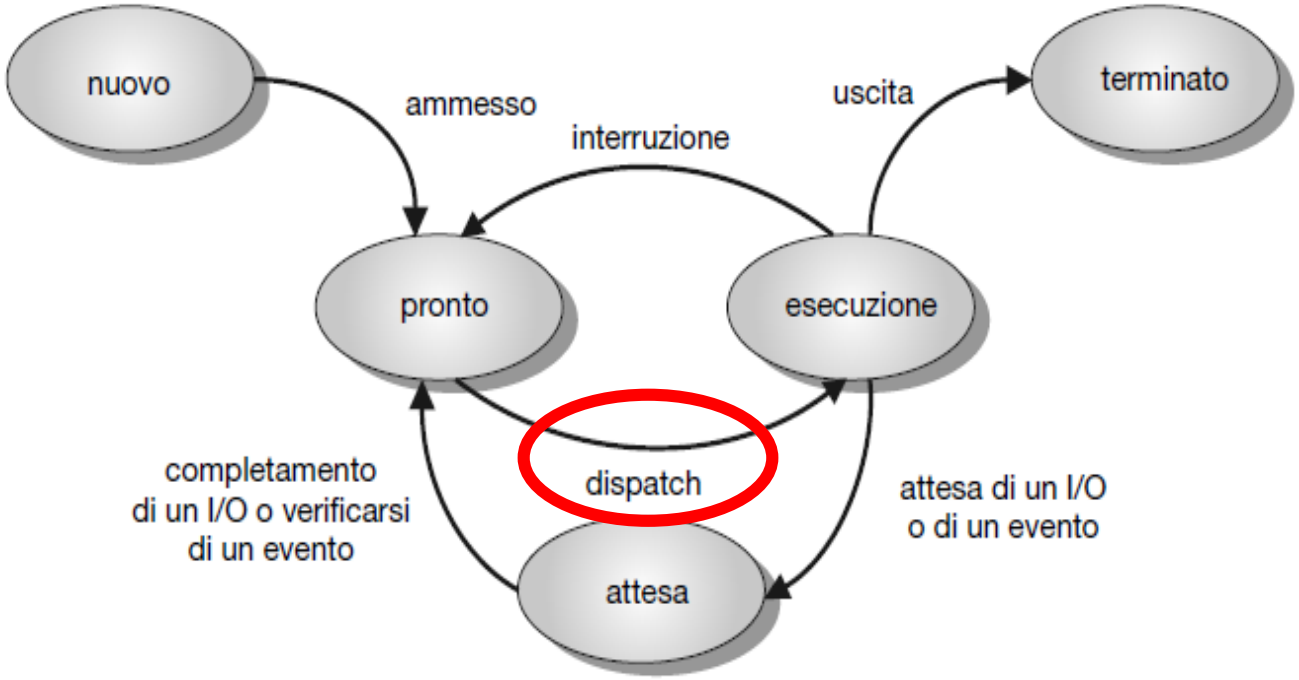
Race condition

Dispatcher

Dispatcher → altro elemento coinvolto nella funzione di scheduling della CPU

Responsabile per:

- Context switch
- 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

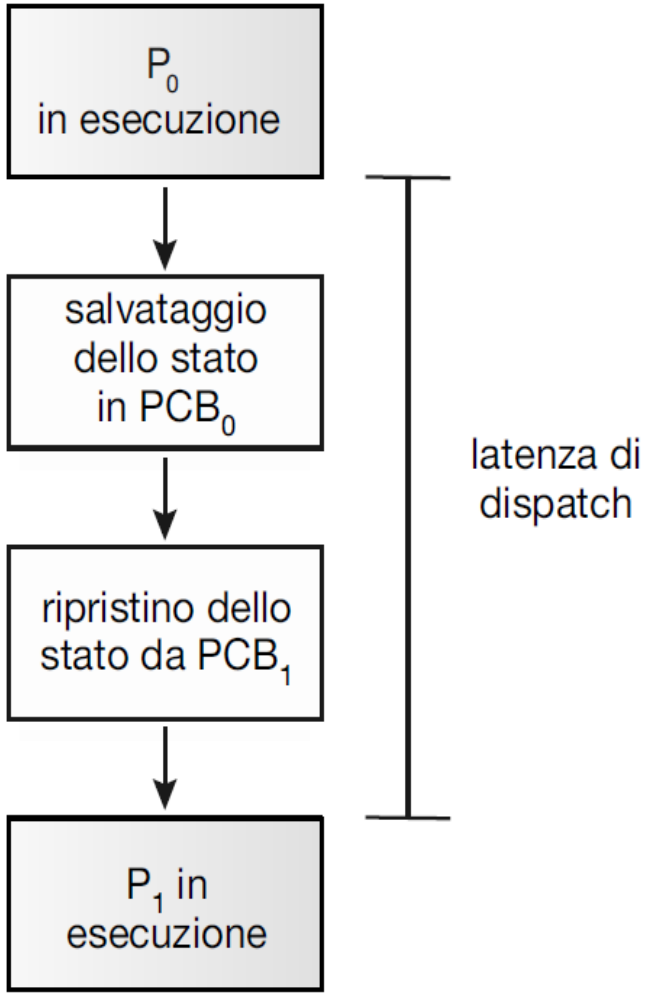
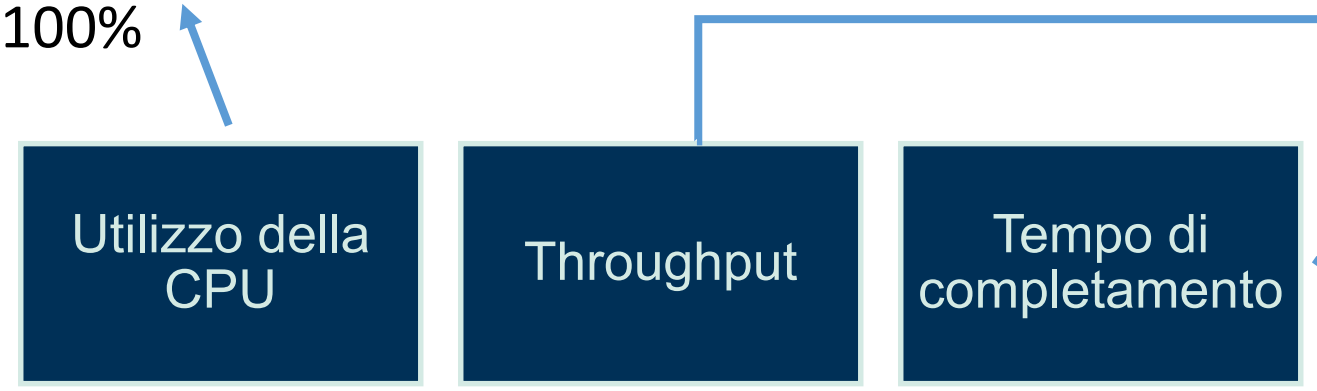


Figura 5.3 Ruolo del dispatcher.

Criteri di scheduling



Può variare tra 0 e 100%

Numero di processi completati nell'unità di tempo

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

Somma degli intervalli d'attesa passati nella ready queue

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

Algoritmi di scheduling

Esistono molti **algoritmi di scheduling della CPU** differenti:

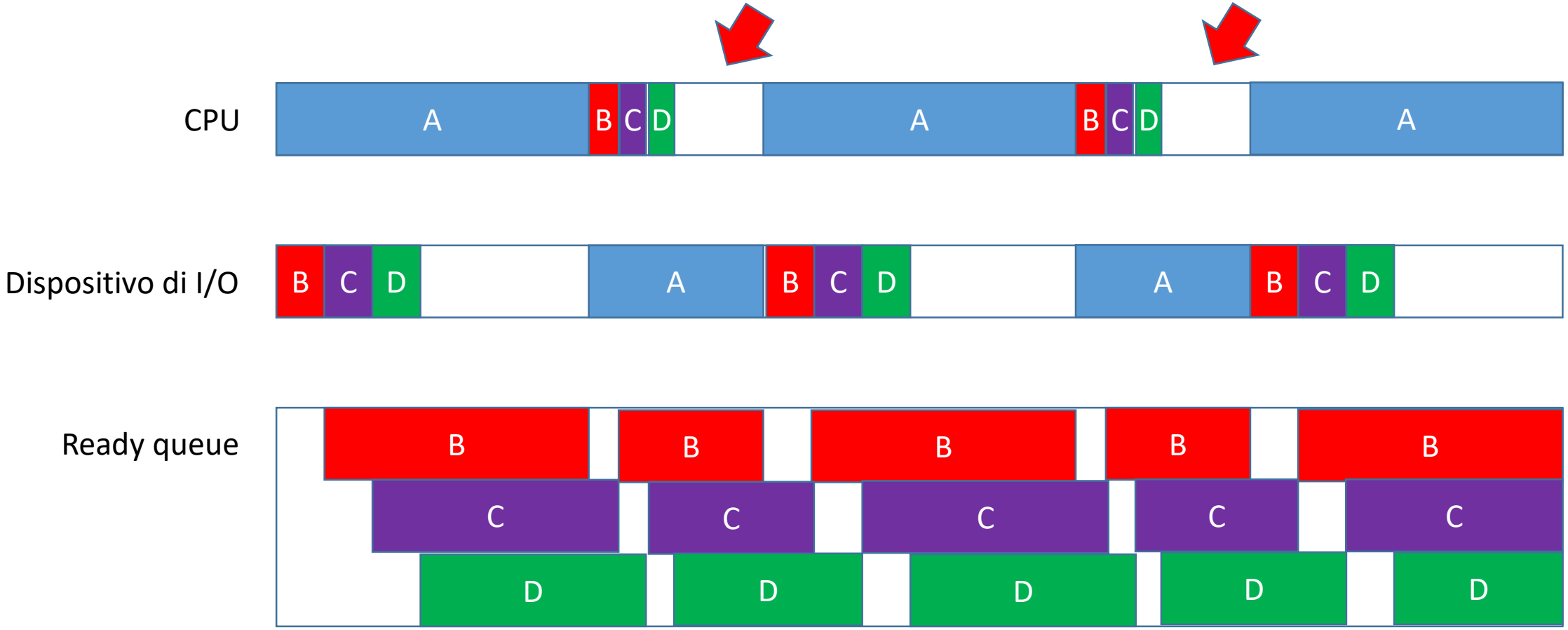
- Scheduling in ordine d'arrivo (first-come, first-served, FCFS)
- Scheduling per brevità (shortest-job-first, SJF)
- Scheduling circolare o (round-robin, RR)
- Scheduling con priorità
- Scheduling a code multilivello
- Scheduling a code multilivello con retroazione

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*)**.

- La CPU si assegna al processo che la richiede per primo
- Senza prelazione
- Tempo medio di attesa spesso abbastanza lungo
- Effetto convoglio

Effetto convoglio



Shortest-Job-First (SJF)

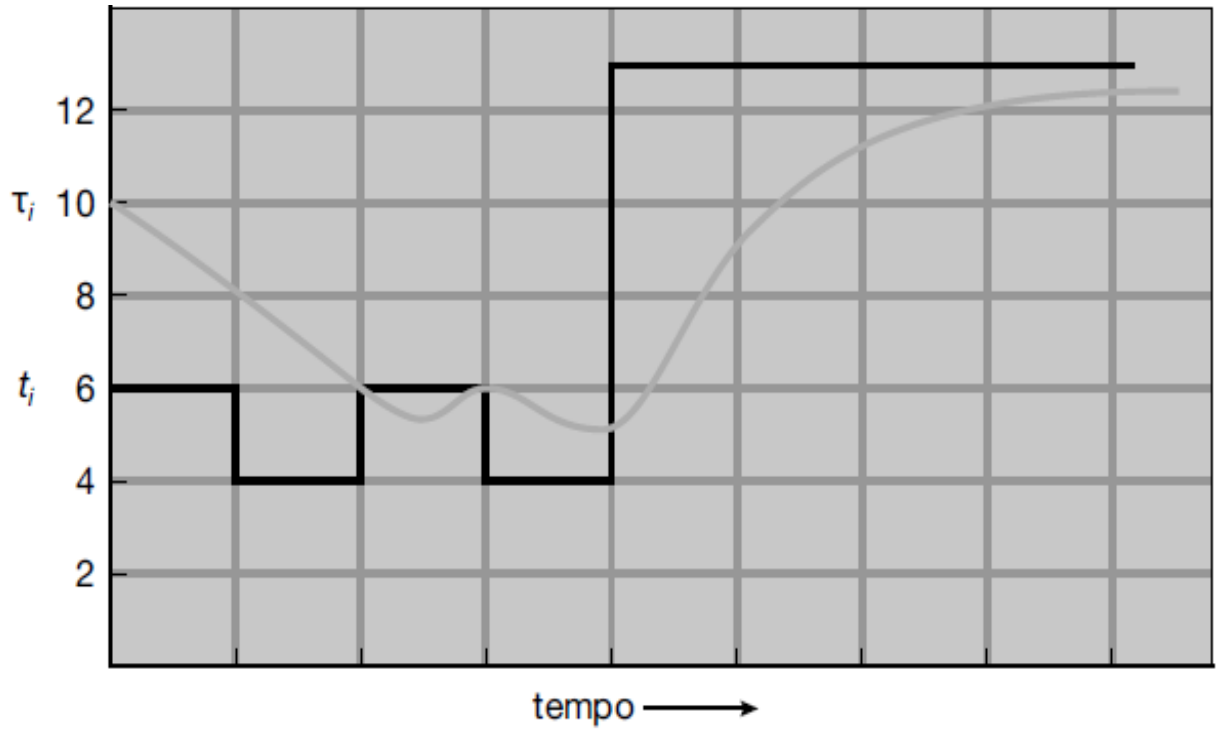
L'**algoritmo di scheduling per brevità** (*shortest-job-first,* ***SJF)*** assegna la CPU al processo che ha la più breve lunghezza della successiva sequenza di operazioni della CPU.

↓

shortest next CPU burst

↓

Si esamina la lunghezza della successiva sequenza di operazioni della CPU del processo e non la sua lunghezza totale



CPU burst (t_i)	6	4	6	4	13	13	13	...
"predizione" (τ_i)	10	8	6	6	5	9	11	12	...

Figura 5.4 Predizione della lunghezza della successiva sequenza di operazioni della CPU (CPU burst).

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.

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

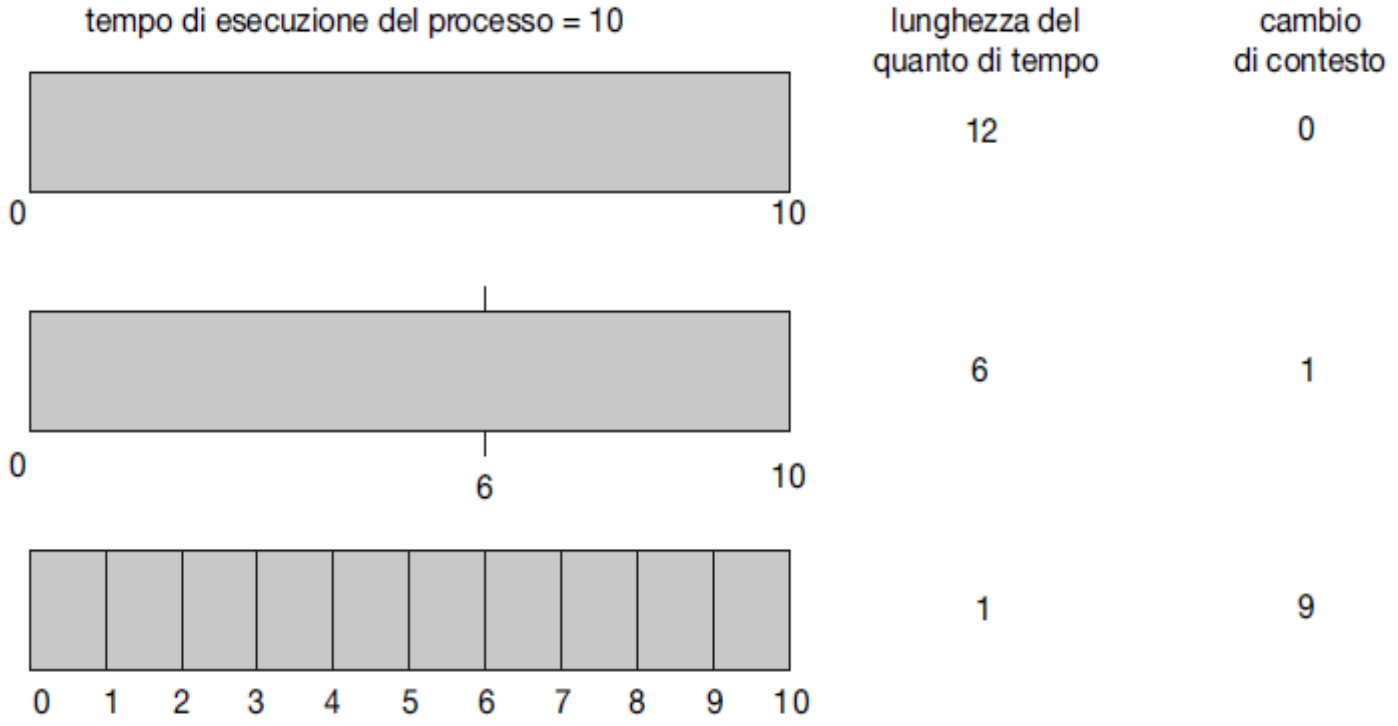
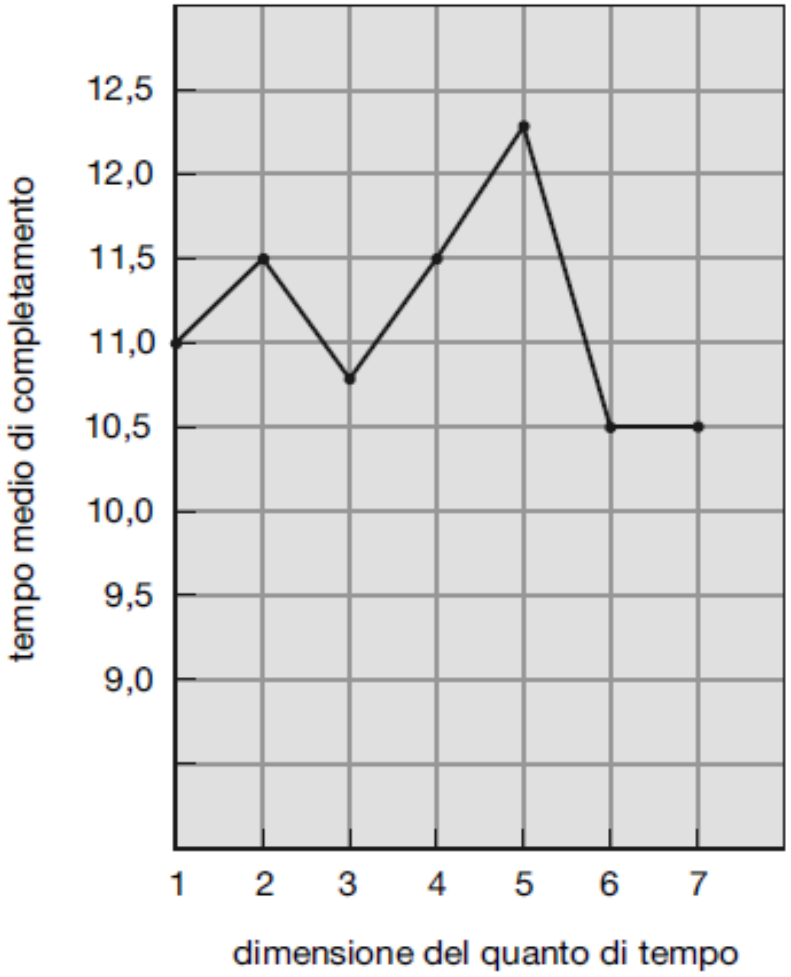


Figura 5.5 Aumento del numero dei cambi di contesto al diminuire del quanto di tempo.

Tempo di completamento

Il **tempo di completamento (*turnaround time*)** dipende dalla dimensione del quanto di tempo: com'è evidenziato nella 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

Figura 5.6 Variazione del tempo di completamento in funzione del quanto di tempo.

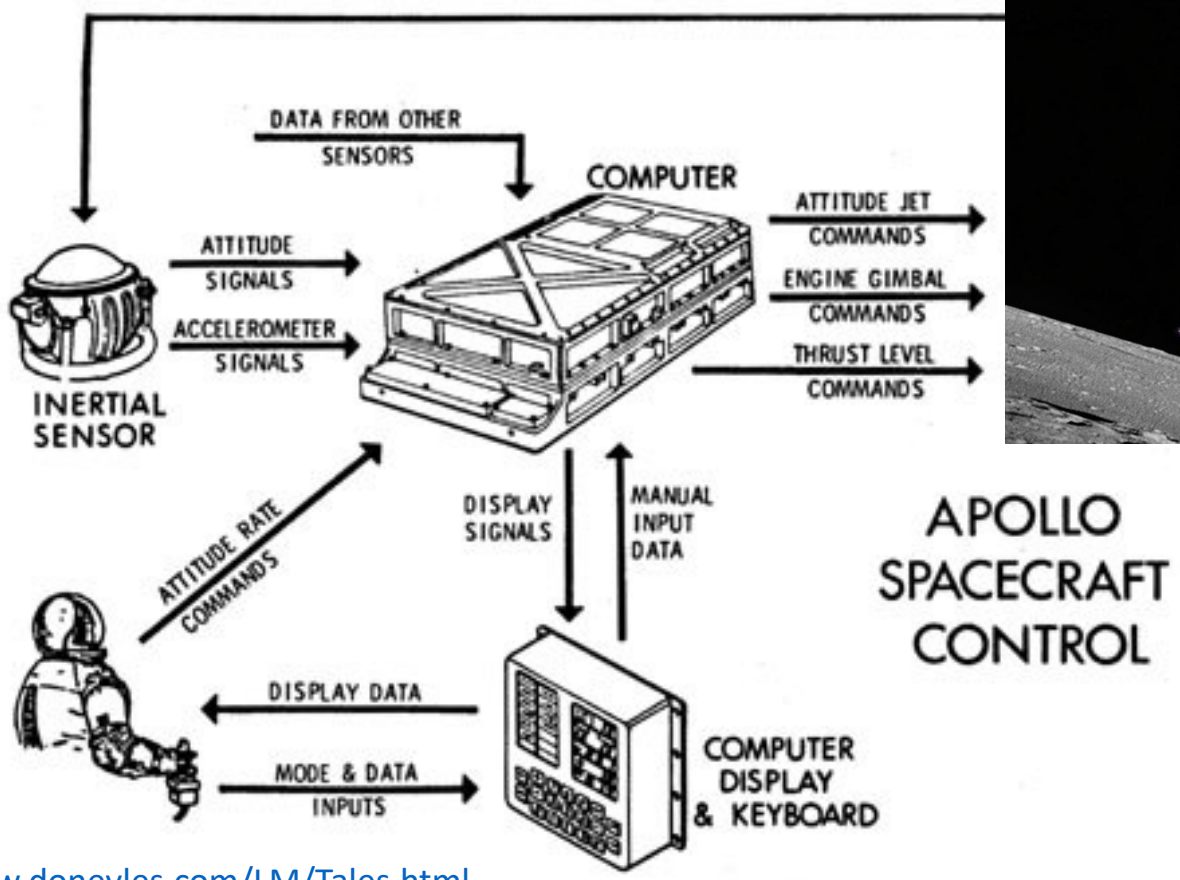
Scheduling con priorità

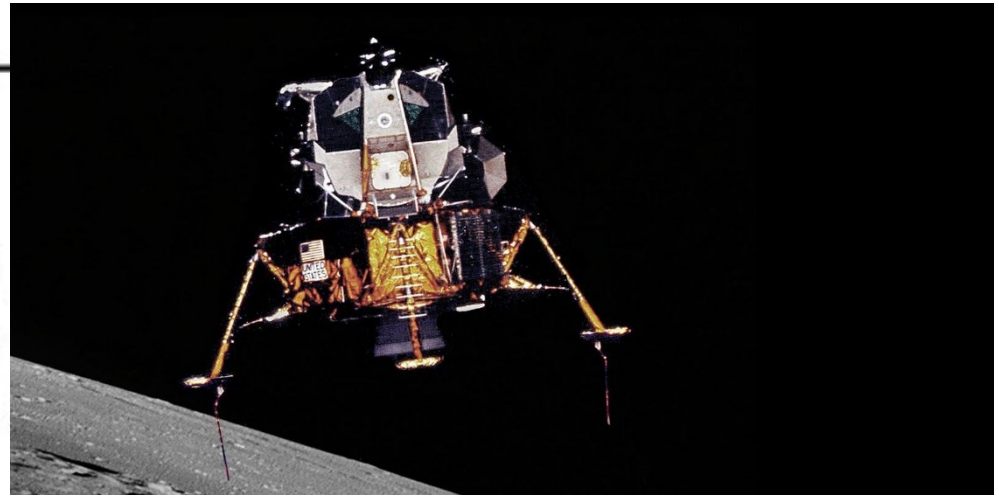
algoritmo di scheduling con priorità → associa una priorità a ogni processo e assegna la CPU al processo con priorità più alta

- Le priorità sono fissate da un intervallo fisso di numeri
- Per esempio, numeri bassi indicano priorità alte
- Può generare starvation (attesa indefinita) → soluzione aging (invecchiamento)

Scheduling con priorità

algoritmo di scheduling con priorità → associa una priorità a ogni processo e assegna la CPU al processo con priorità più alta





https://peterbrandt.space/blog/2019-07-20-Apollo-50-years

https://www.doneyles.com/LM/Tales.html

Scheduling con code multilivello

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

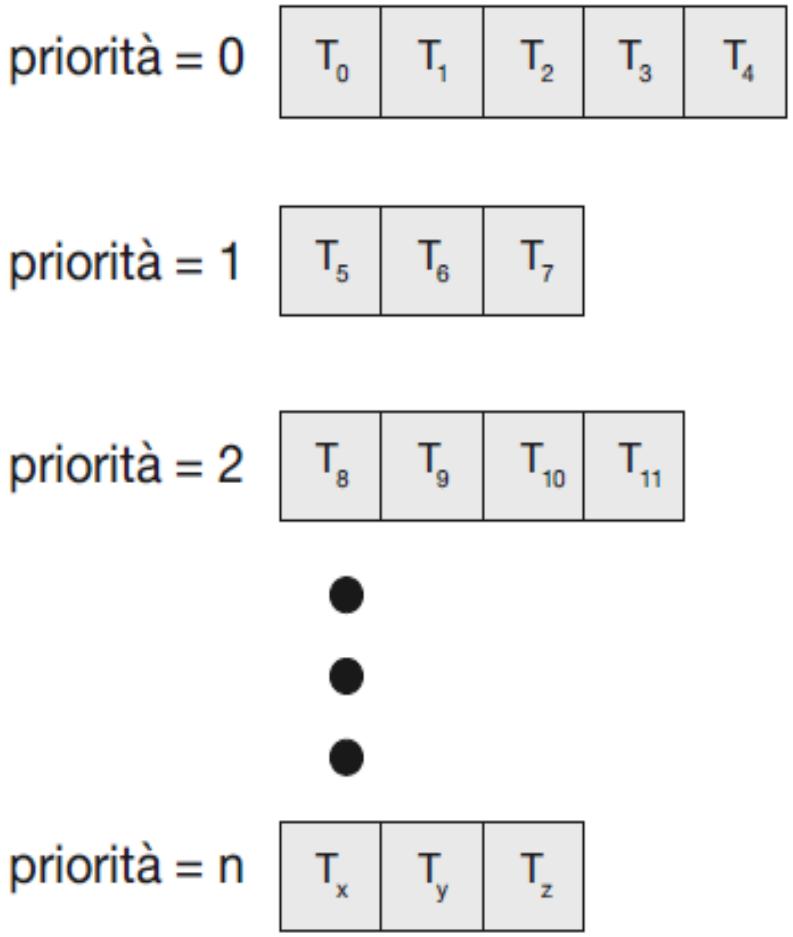


Figura 5.7 Code distinte per ogni priorità.

Scheduling con 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.

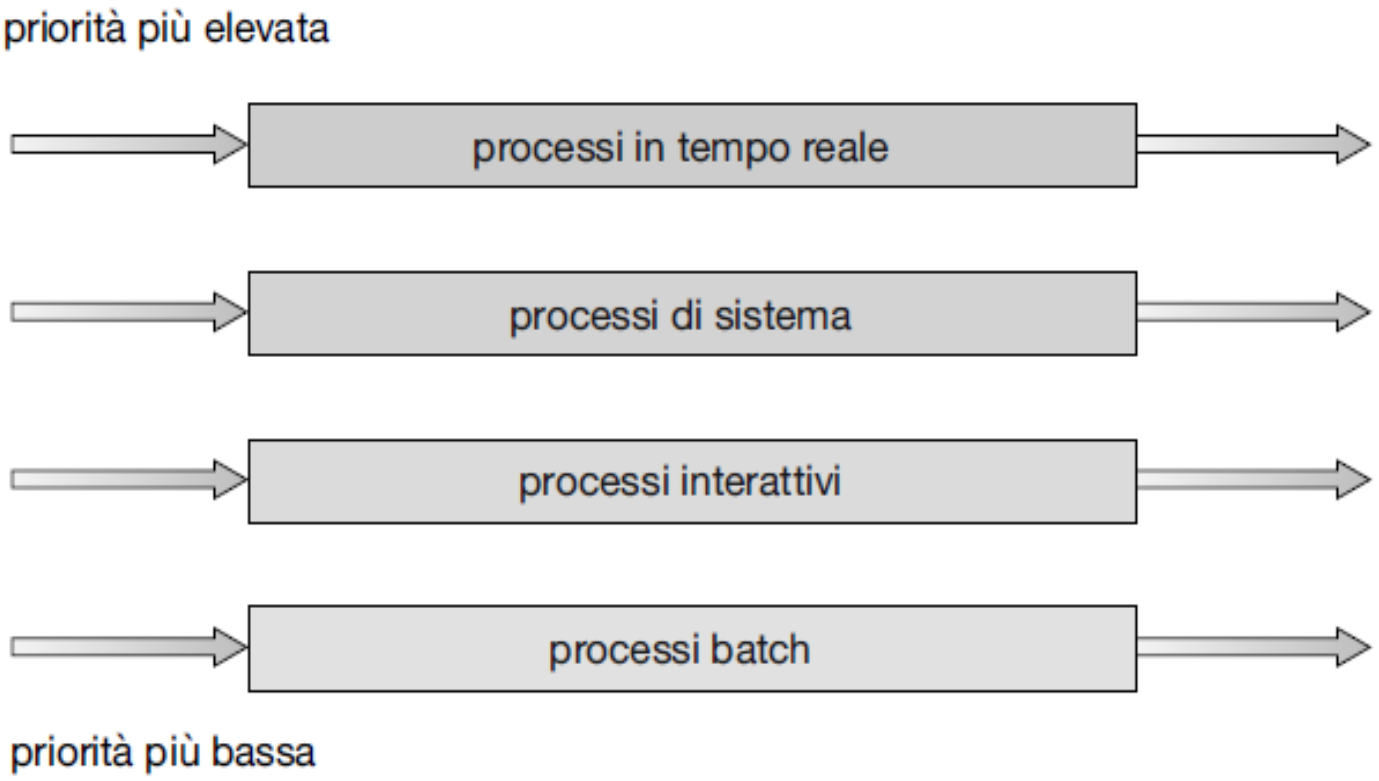


Figura 5.8 Scheduling a code multilivello.

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.

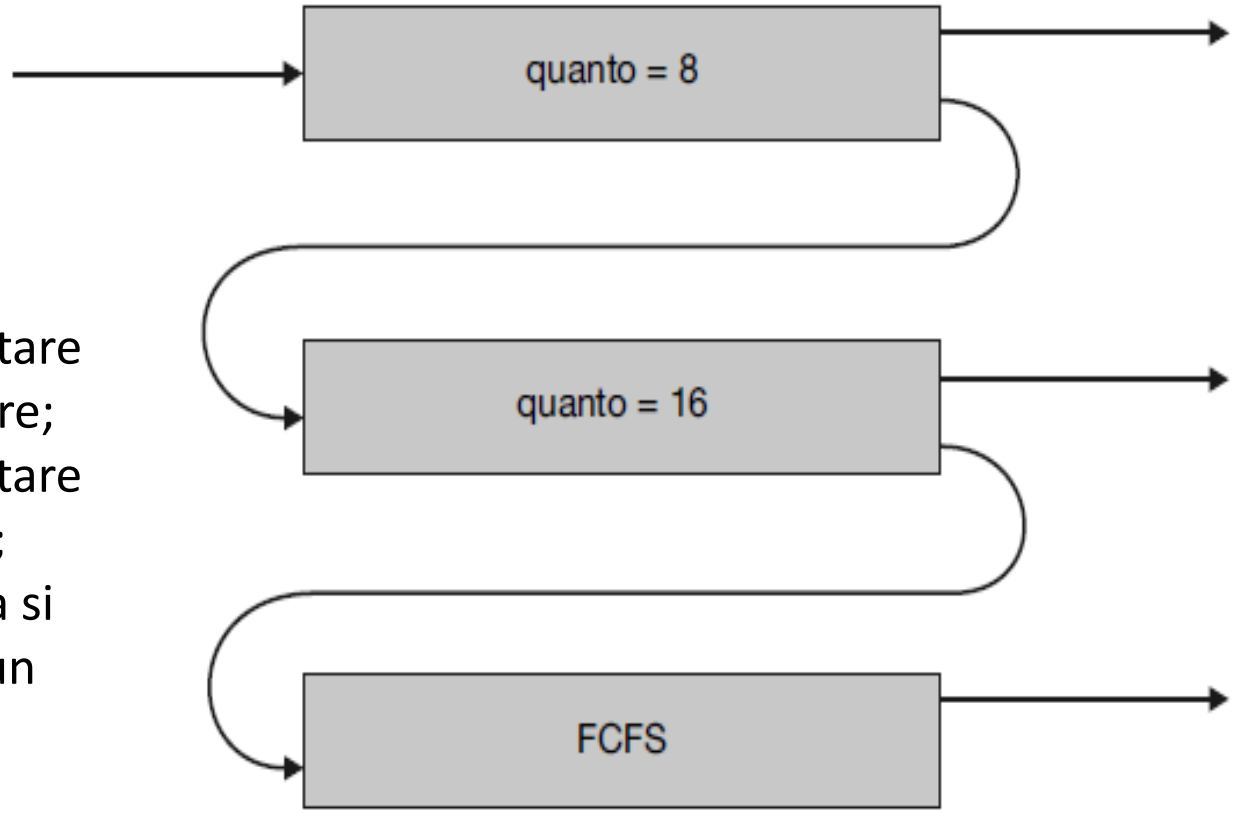


Figura 5.9 Code multilivello con retroazione.

Scheduling dei thread

a livello utente → ambito della contesa ristretto al processo
(*process-contention scope, PCS*)

e

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.

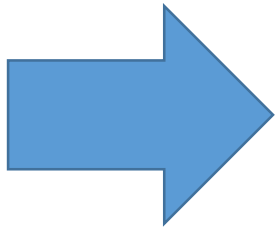
API di scheduling PThreads

```
#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");
   }
```



API di scheduling PThreads

```
   /* 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);
}
```

Figura 5.10 API di scheduling Pthread.

Scheduling per sistemi multiprocessore

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

- CPU multicore
- Core multithread
- Sistemi NUMA (accesso non uniforme alla memoria)
- 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.

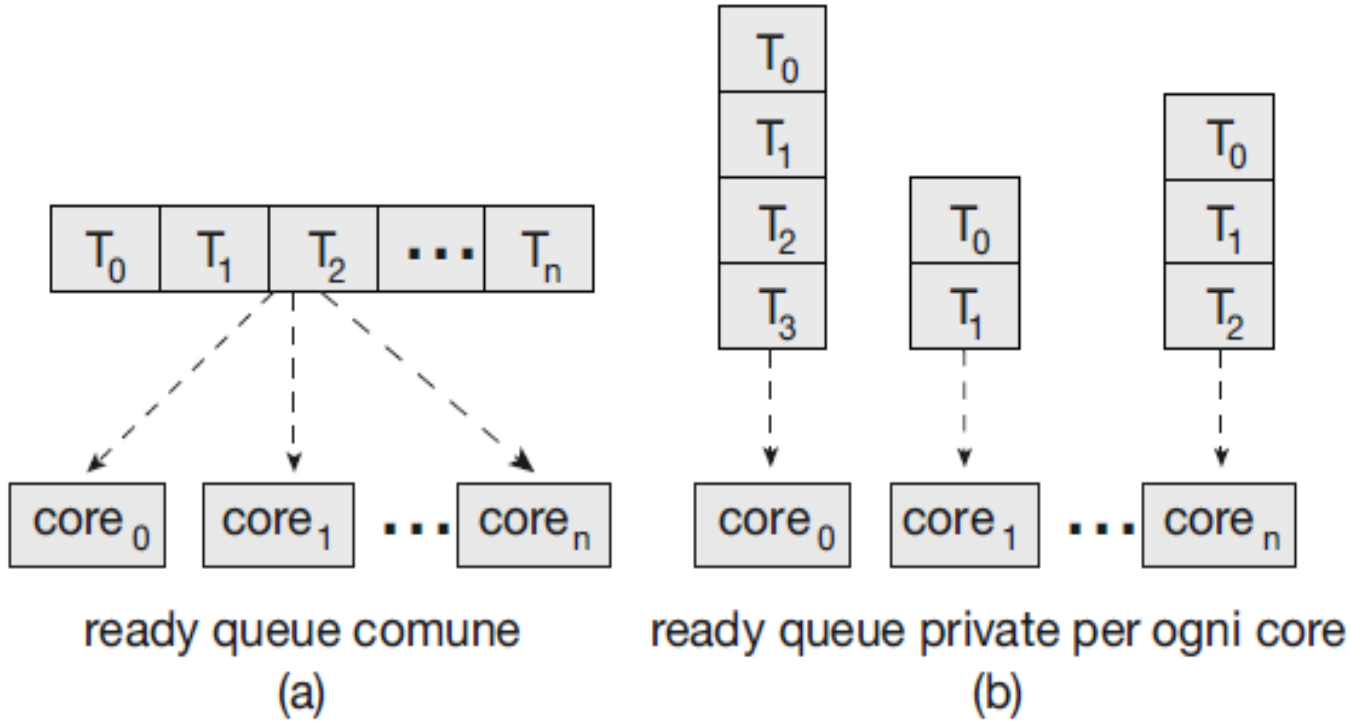


Figura 5.11 Organizzazione di ready queue.

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**

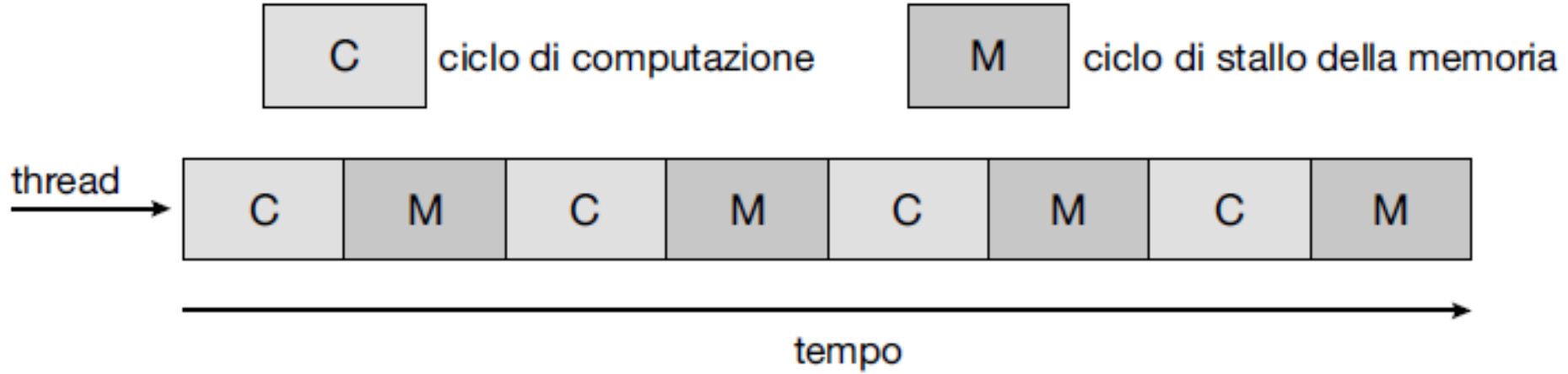


Figura 5.12 Stallo della memoria.

Scheduling per sistemi multicore

La Figura 5.13 mostra un processore a due thread nel quale l'esecuzione dei thread 0 e 1 sono avvicendate nel tempo.

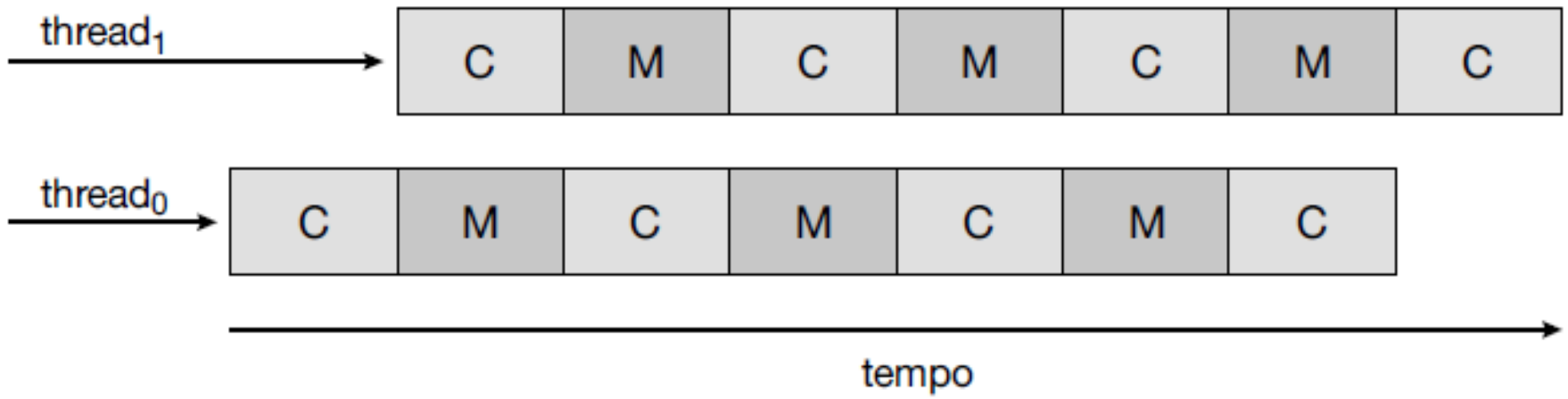


Figura 5.13 Sistema multicore e multithread.

Scheduling per sistemi multicore

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**.

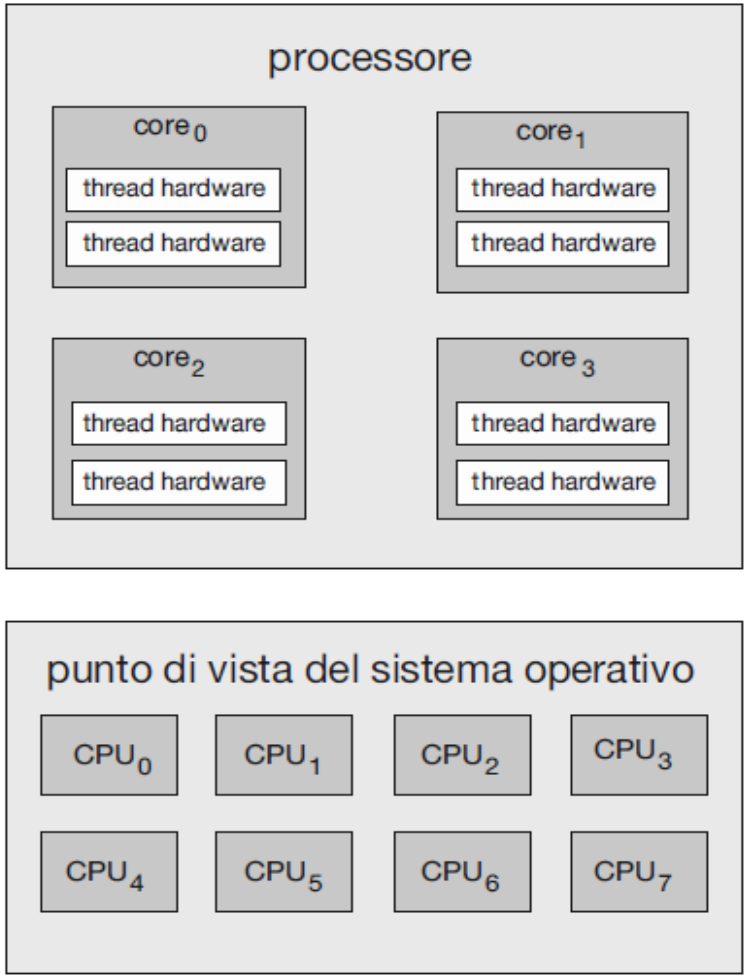
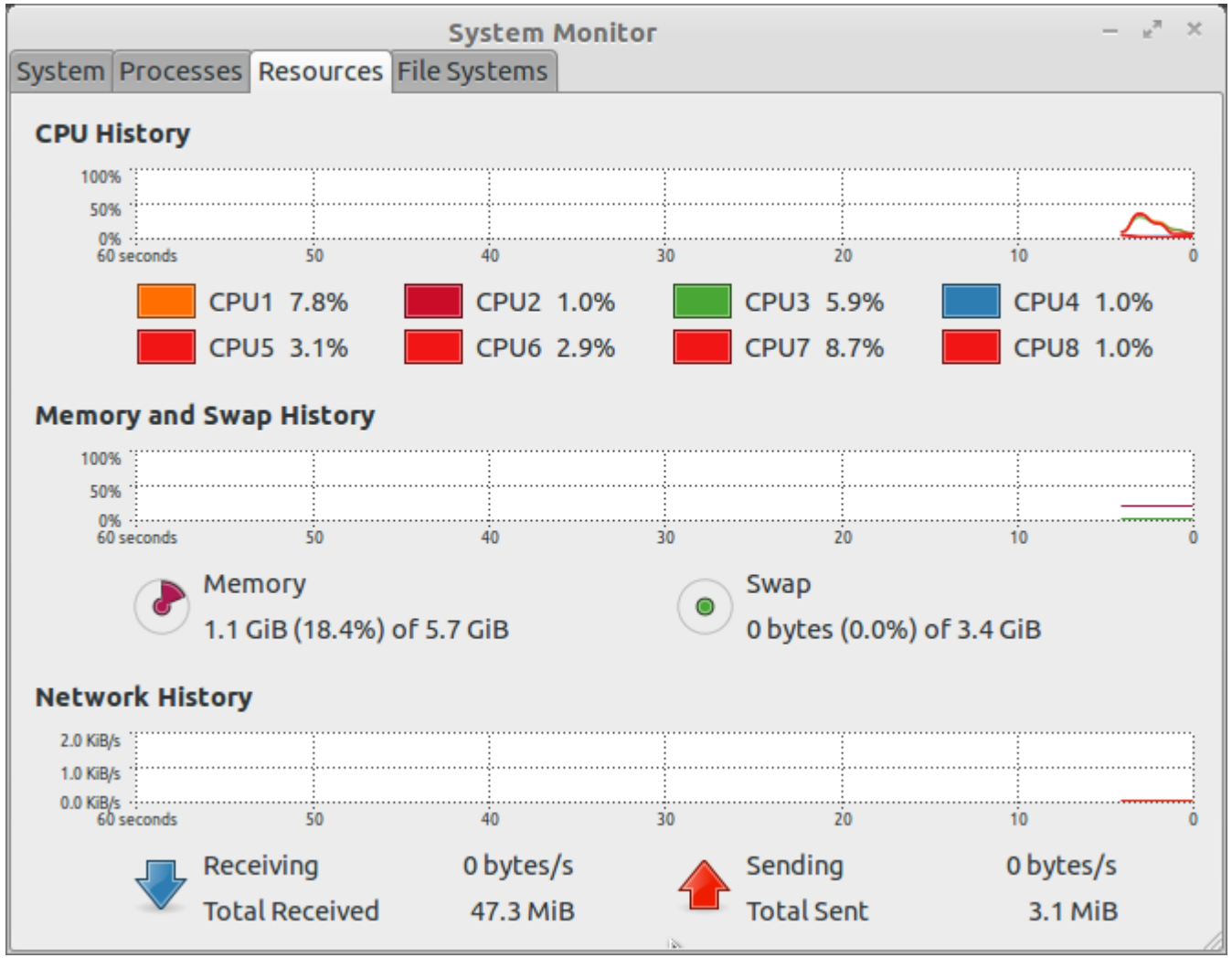


Figura 5.14 Chip multithreading.

Gnome System Monitor



https://askubuntu.com/questions/165955/how-can-i-enable-second-core-of-my-intel-atom-in-my-dell-mini-910

Scheduling a due livelli

Un processore *multithreaded* e *multicore* richiede due diversi livelli di scheduling, come mostrato nella Figura 5.15, che illustra un core di elaborazione dual-threaded.

I due diversi livelli di scheduling mostrati nella Figura 5.15 non sono necessariamente mutuamente esclusivi

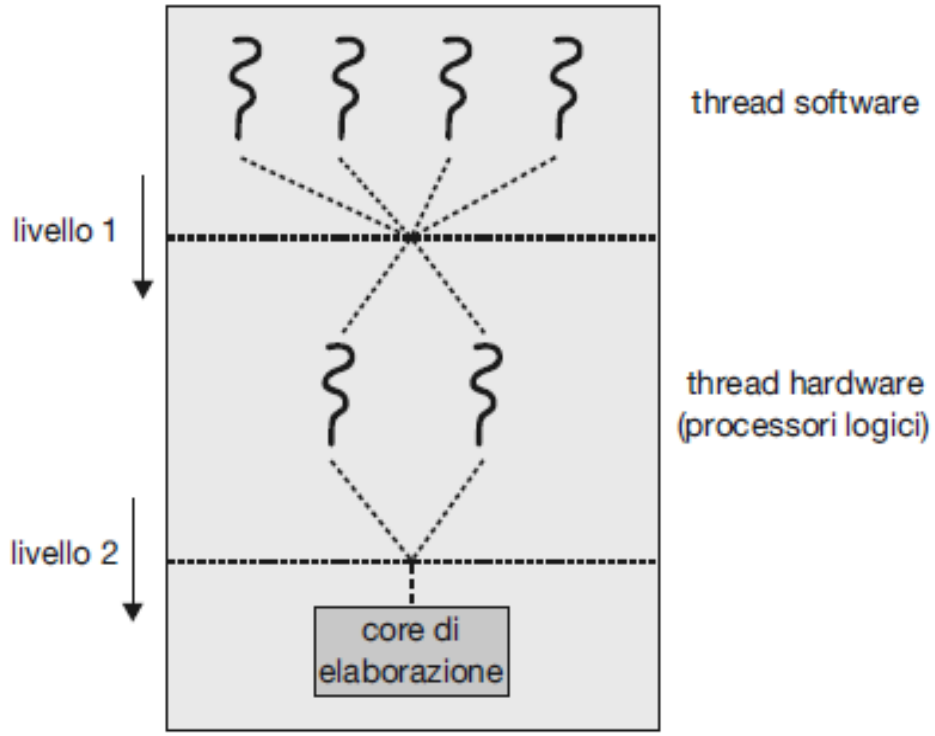


Figura 5.15 Due livelli di scheduling.

NUMA e scheduling della CPU

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

- **predilezione debole (*soft affinity*)**
- **predilezione forte (*hard affinity*)**

La Figura 5.16 mostra 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.

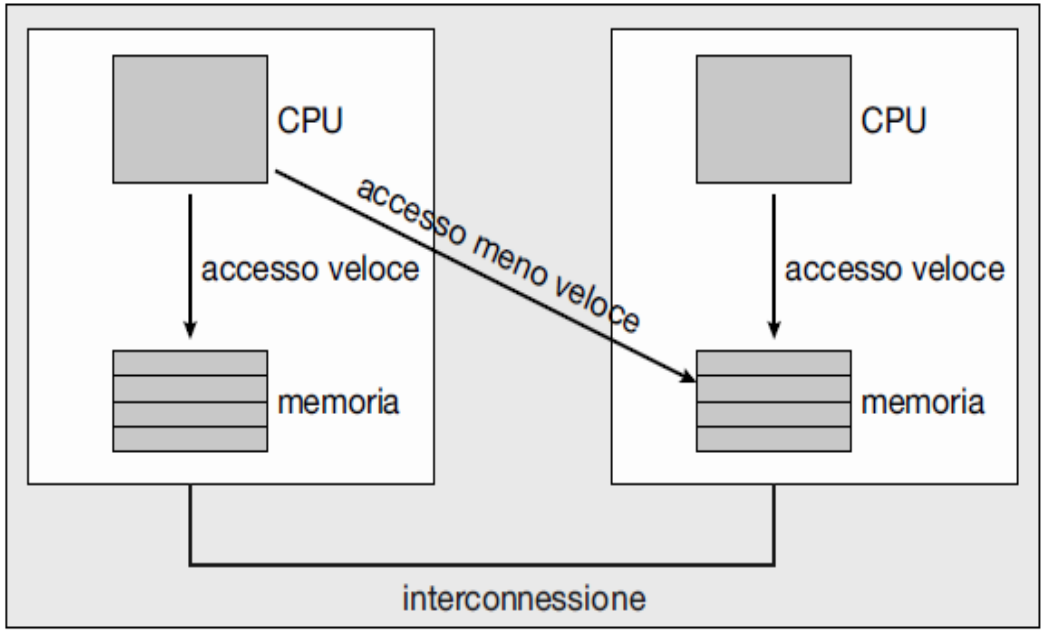


Figura 5.16 NUMA e lo scheduling della CPU.

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.

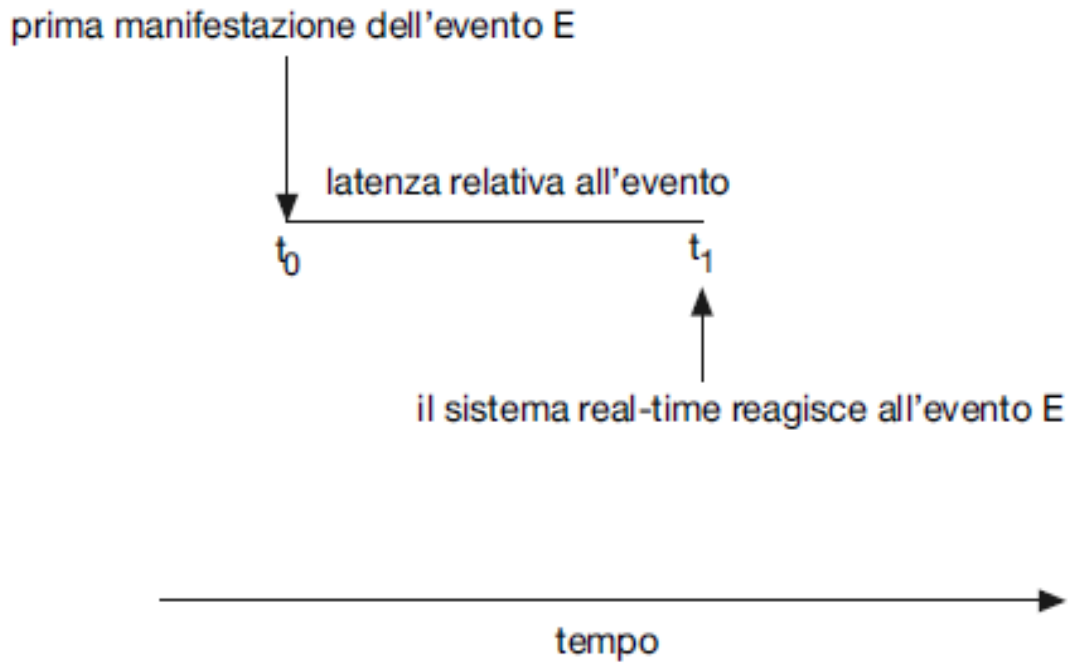


Figura 5.17 Latenza relativa all'evento.

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

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

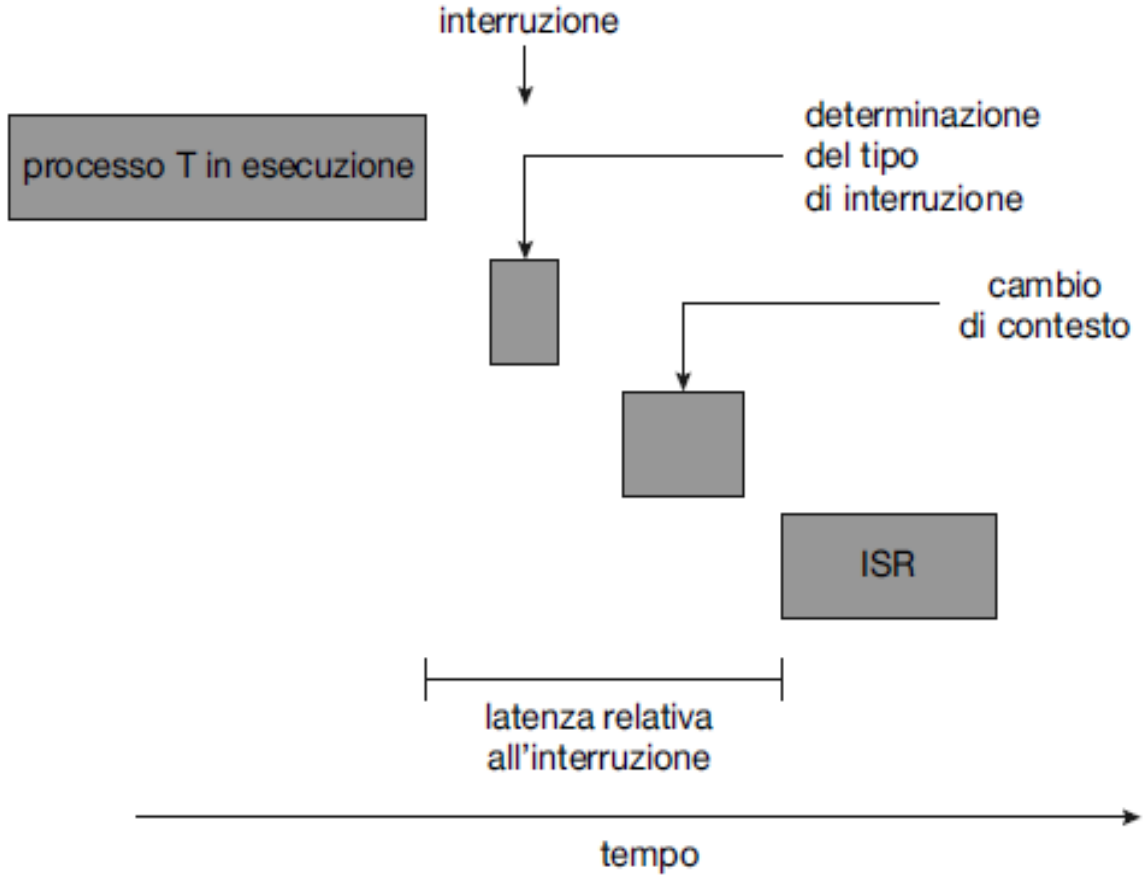


Figura 5.18 Latenza relativa alle interruzioni.

Latenza

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.

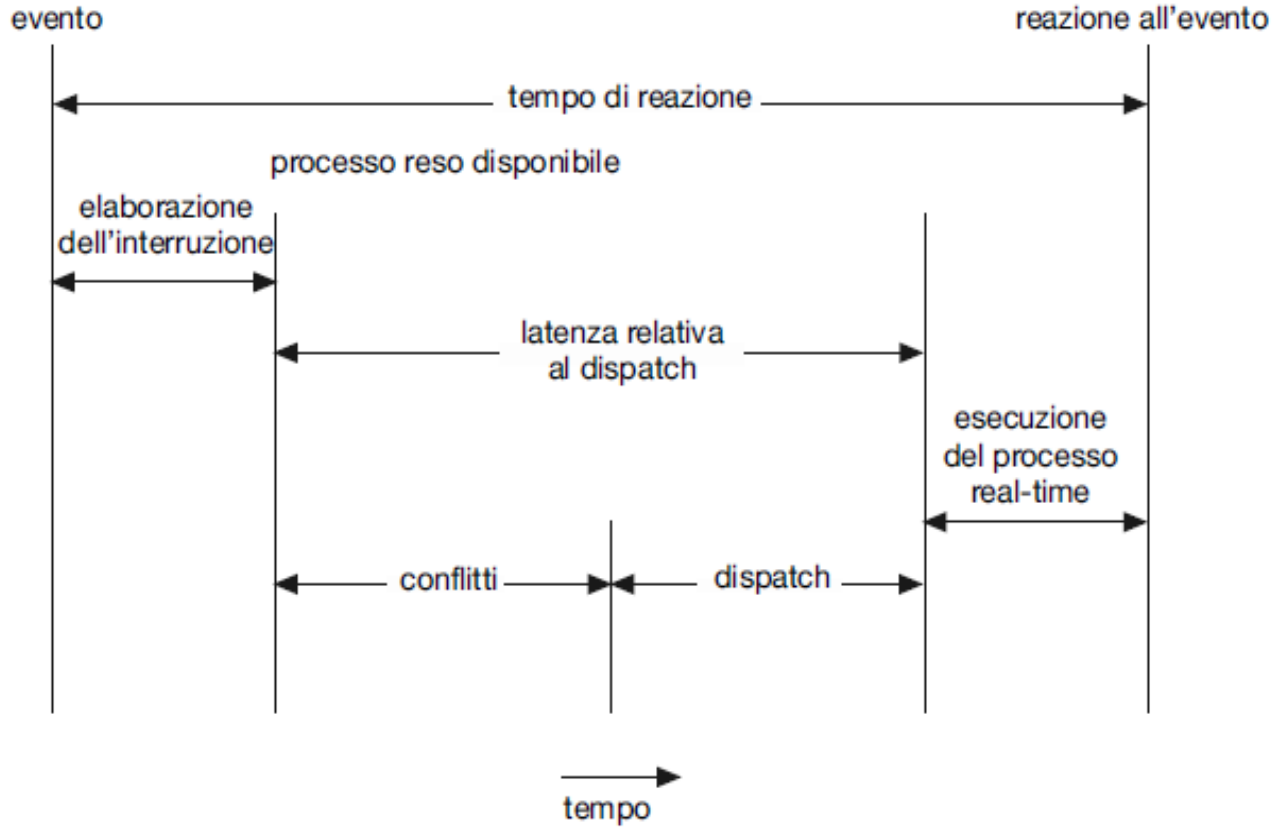


Figura 5.19 Latenza relativa al dispatch.

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)

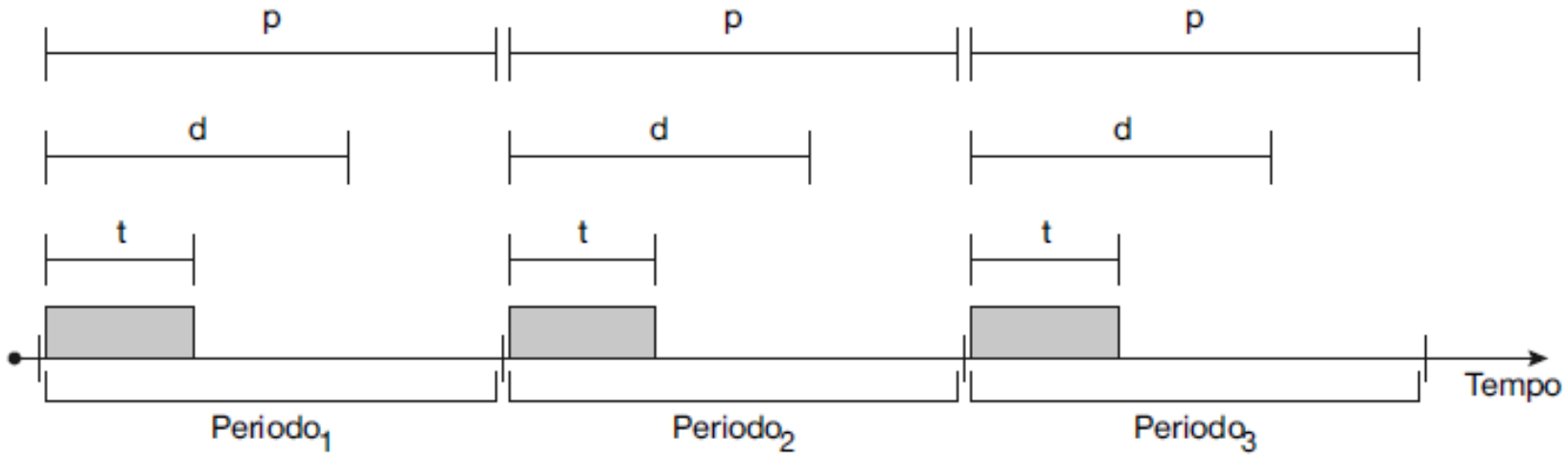


Figura 5.20 Processo periodico.

Scheduling con priorità 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.

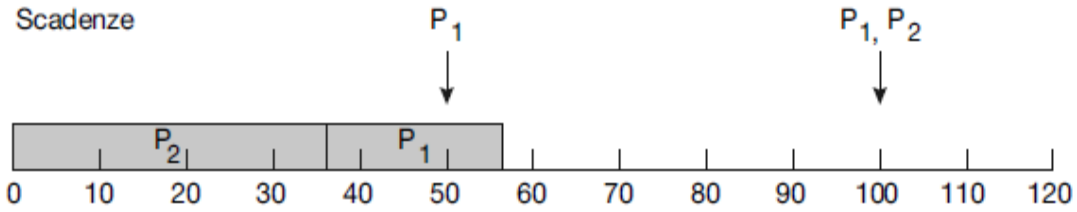


Figura 5.21 Scheduling dei task in caso P_2 abbia priorità maggiore di P_1.

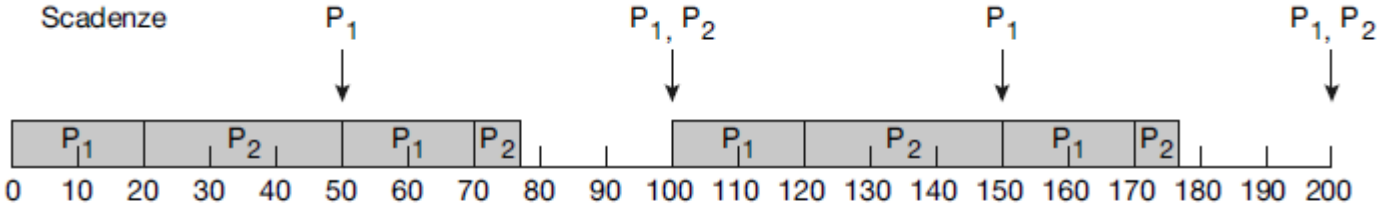


Figura 5.22 Scheduling con priorità proporzionale alla frequenza.

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

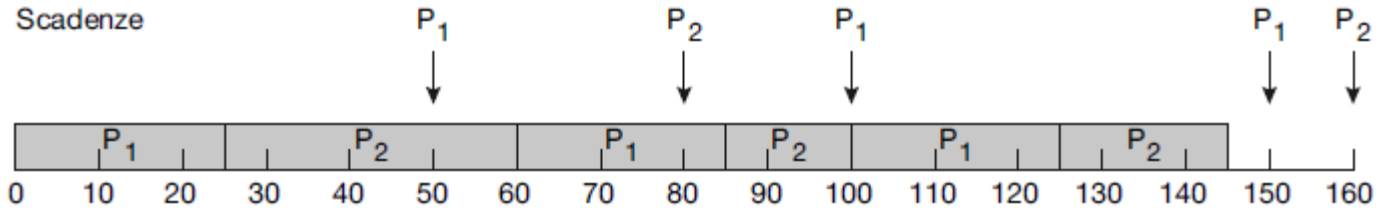


Figura 5.24 Scheduling EDF.

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.

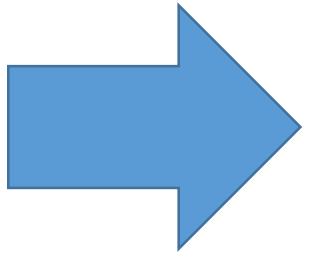
Scheduling real-time POSIX

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

int main(int argc, char *argv[])
{
  int i, policy;
  pthread_t tid[NUM_THREADS];
  pthread_attr_t attr;

  /* appura gli attributi di default */
  pthread_attr_init(&attr);

  /* appura la politica di scheduling corrente */
  if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
    fprintf(stderr, "Unable to get policy.\n");
  else {
    if (policy == SCHED_OTHER)
     printf("SCHED_OTHER\n");
    else if (policy == SCHED_RR)
     printf("SCHED_RR\n");
    else if (policy == SCHED_FIFO)
     printf("SCHED_FIFO\n");
  }
```



Scheduling real-time POSIX

```
   /* imposta la politica di scheduling – FIFO, RR o OTHER */
   if (pthread_attr_setschedpolicy(&attr, SCHED_OTHER) != 0)
     fprintf(stderr, "Unable to set policy.\n");

   /* genera i thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);

   /* ora attende la terminazione di ciascun thread */
   for (i = 0; i < NUM_THREADS; i++)
     pthread_join(tid[i], NULL);
}

/* ciascun thread comincia l'esecuzione da questa funzione */
void *runner(void *param)

{
   /* fai qualcosa ... */

   pthread_exit(0);
}
```

Figura 5.25 Scheduling real-time con la API POSIX.

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.

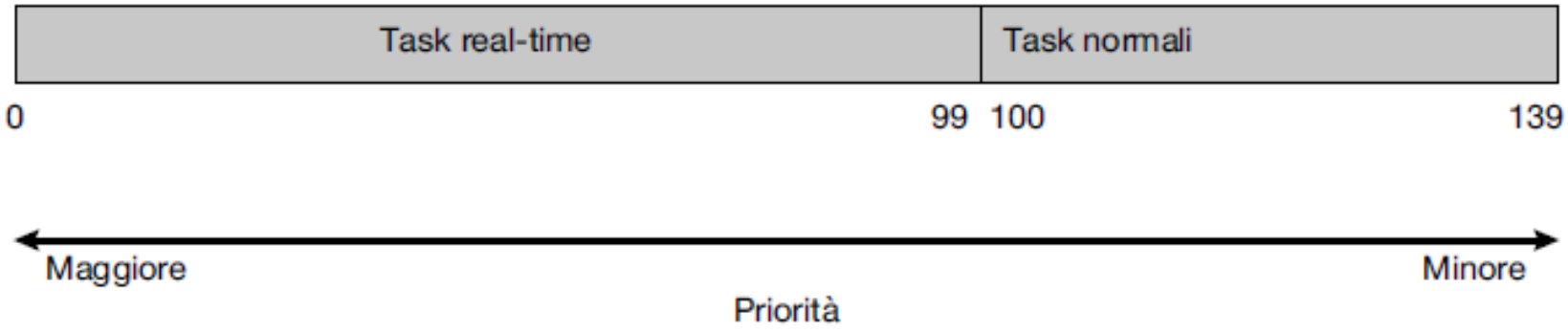


Figura 5.26 Priorità di scheduling in Linux.

Dominio di scheduling in Linux

Un **dominio di scheduling** è un insieme di core che può essere bilanciato l'uno con l'altro, come mostrato nella Figura 5.27.

I core in ciascun dominio di scheduling sono raggruppati in base al modo in cui condividono le risorse del sistema.

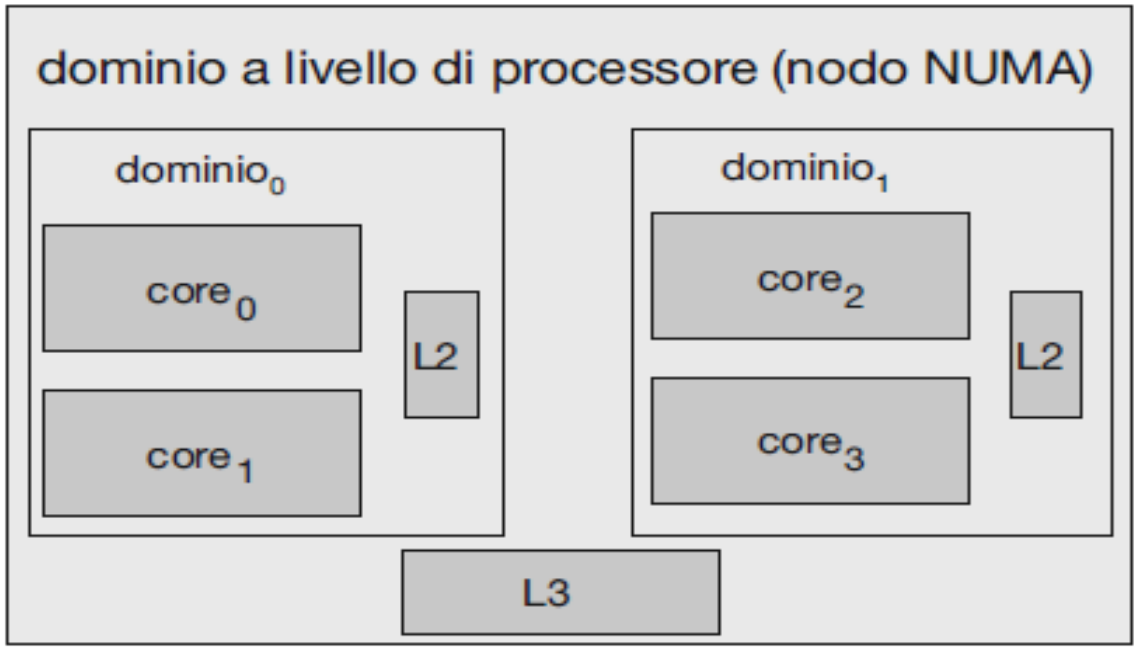


Figura 5.27 Bilanciamento del carico NUMA-aware nello scheduler CFS di Linux.

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.

Classi di priorità nello scheduling in Windows

	real-time	high	above_ normal	normal	below_ normal	idle_ priority
time_critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above_normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below_normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Figura 5.28 Priorità dei thread in Windows.

Valutazione degli algoritmi

metodi di valutazione dell'algoritmo
di scheduling della CPU

Valutazione degli algoritmi

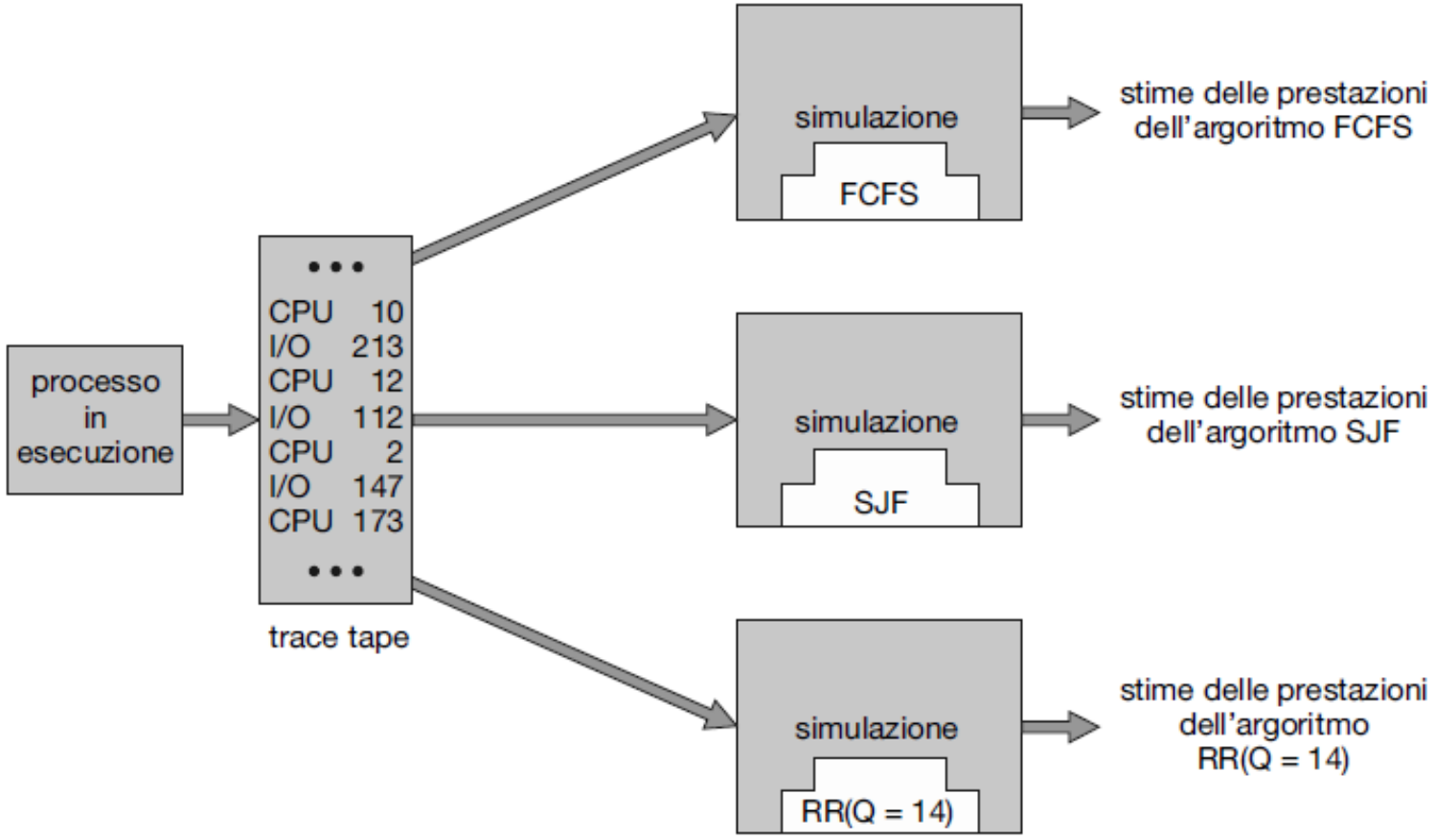
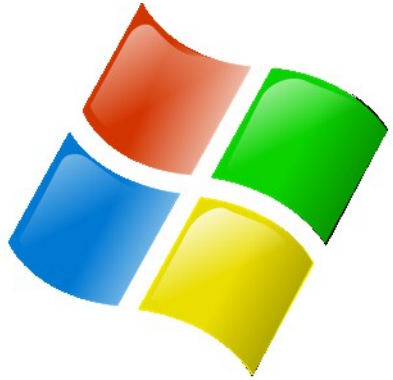


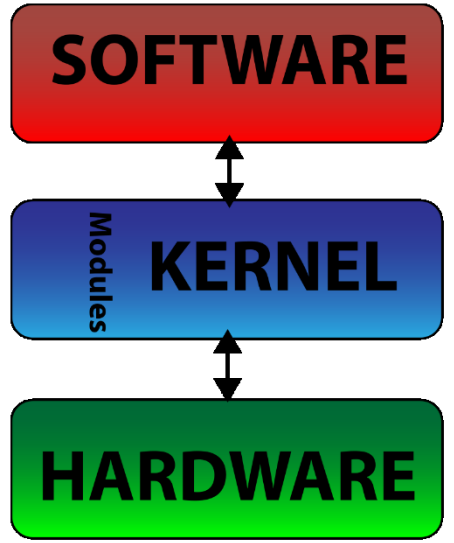
Figura 5.31 Valutazione di algoritmi di scheduling della CPU tramite una simulazione.

UNIVERSITÀ DEGLI STUDI DELLA BASILICATA



Corso di Sistemi Operativi

Scheduling della CPU



Docente:
Domenico Daniele Bloisi