SAPIENZA
UNIVERSITÀ DI ROMA

Corso di laurea in Ingegneria dell'Informazione
Indirizzo Informatica

Reti e sistemi operativi

I processi: concetti di base, context switch e scheduling

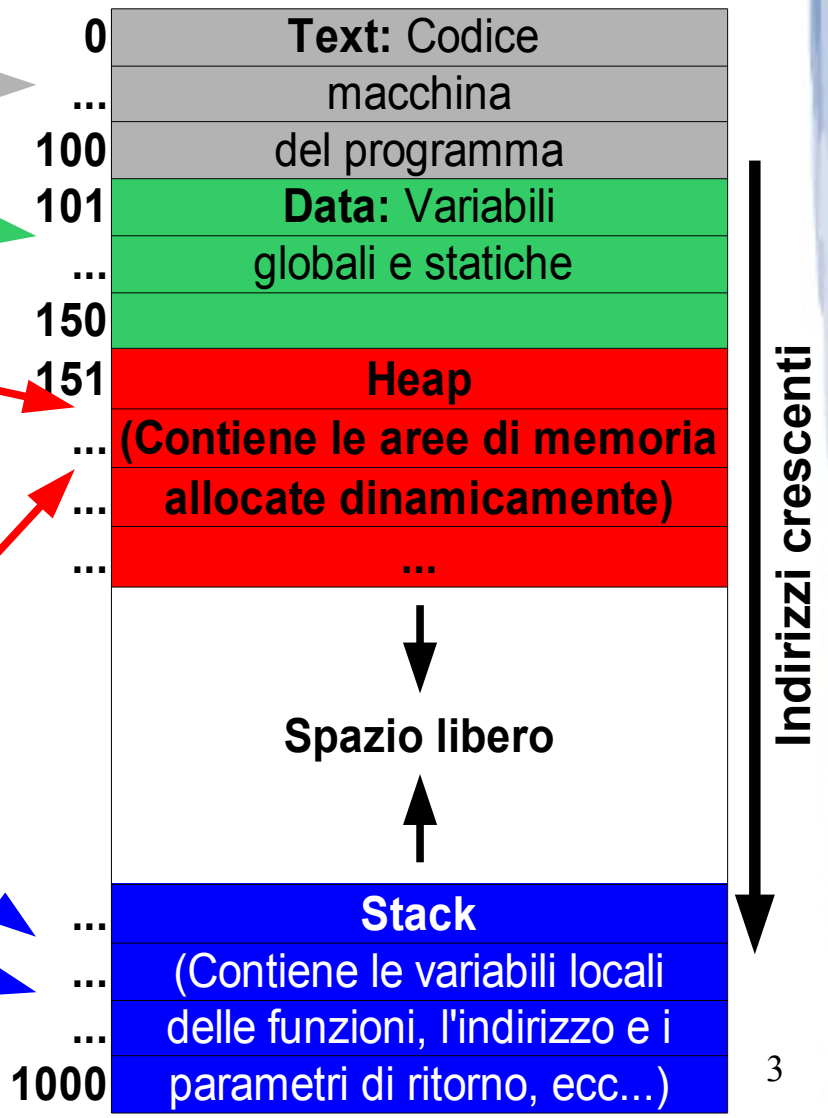
Processo: definizione

- Processo (o *Job*): **Entità attiva che rappresenta il programma in esecuzione**.
- Ci possono essere 2 o più processi che condividono lo stesso programma → sono entità indipendenti.
- Fanno parte di un processo:
  - Il contenuto dei registri di CPU (PC, registri di ALU, …) → almeno il PC varia ad ogni ciclo di esecuzione del processo stesso → il processo "evolve".
  - Una porzione di memoria ad esso dedicata, che include:
    - Il programma o "eseguibile" (nella **text section**) → entità "passiva", non cambia durante l'esecuzione
    - Le variabili globali (nella **data section**)
    - Lo stack
    - L'heap

Un processo in memoria

- Spazio degli indirizzi di un processo (modello semplificato)

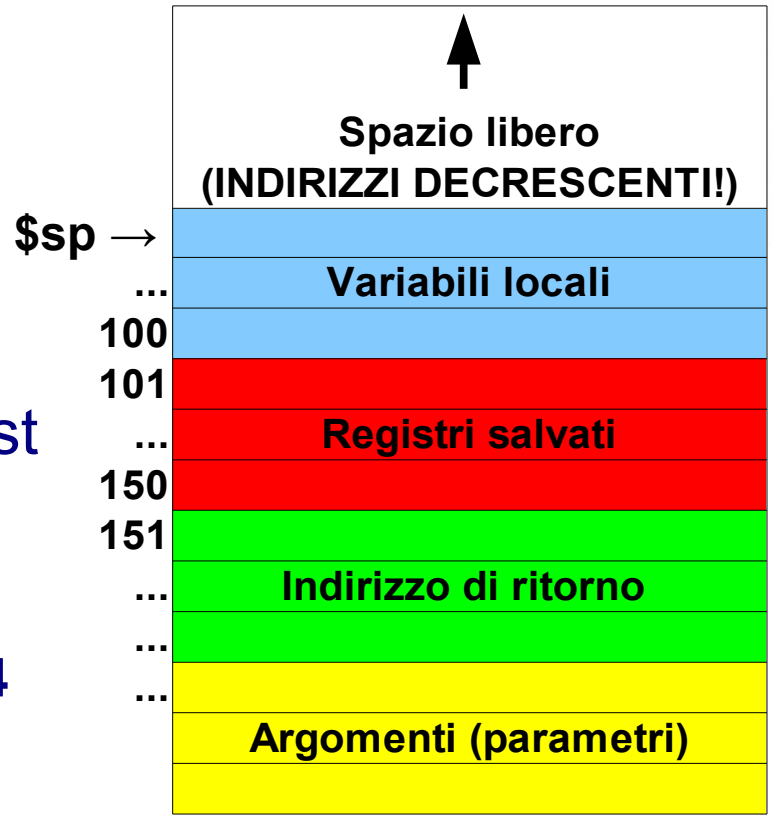
```
int g_counter;

int f(int n)
{
        float *res;
        res = (float *)calloc(n, sizeof(float));
        /* .....*/
        free(res);
        g_counter++;
}
int main()
{
        char *str = ”ciao”;
        int vect[1000];
        int *p;
        p  = (int *)malloc(10*sizeof(int));
        /* ... */
        free(p);
        g_counter++;
}
```



Le frecce indicano ove sono memorizzate le variabili e i blocchi di memoria

Lo stack

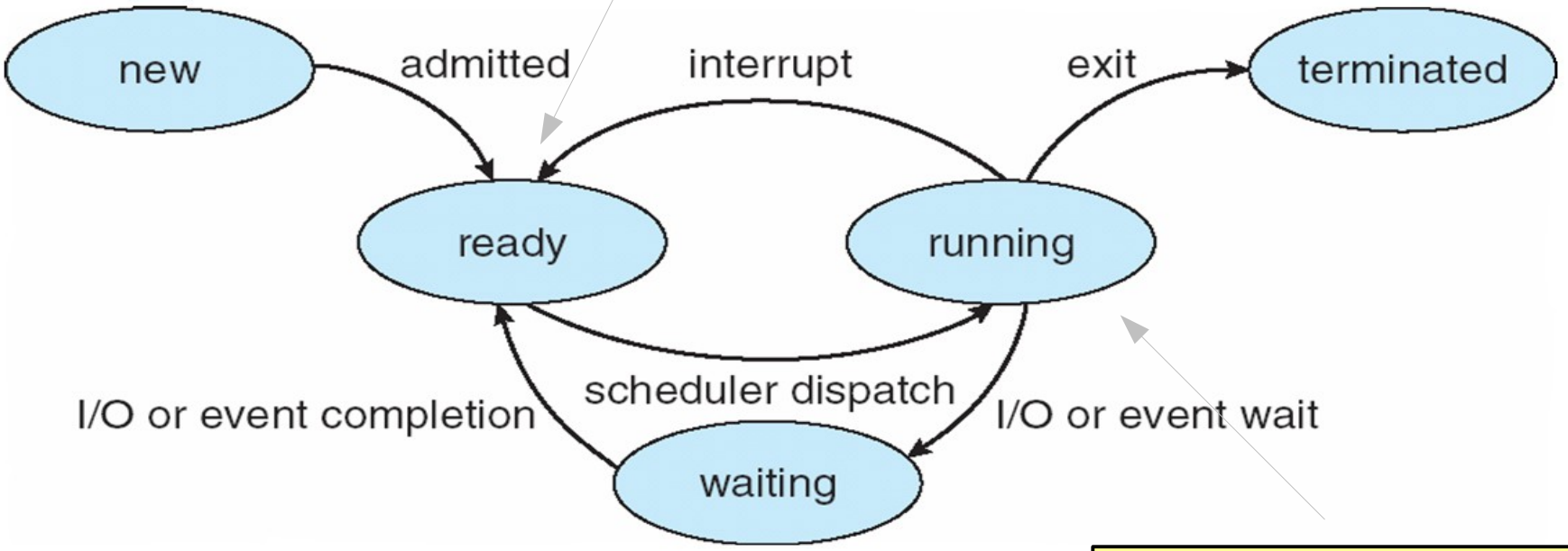
- Lo stack (pila) è una porzione di memoria ben definita che **si espande verso indirizzi decrescenti.**
- Il registro $sp (stack pointer) contiene in ogni momento l'indirizzo della **testa dello stack**.
- Lo stack è una struttura dati LIFO (Last In First Out) in cui si possono definire due operazioni:
  - **Push($ra)**, es. **addi $sp, $sp, -4**
    sw $ra, 0($sp)
  - **$ra = Pop()**, es. **lw $ra, 0($sp)**
    addi $sp, $sp, 4



Stato di un processo

Il processo viene creato

Processo in stato dormiente in attesa di essere messo in esecuzione: attende il S.O.



Processo in stato dormiente in attesa di qualche evento. Attende, ad esempio, un evento di I/O.

In ogni istante c'è un numero di processi in stato running **minore o uguale** a N , dove N è il numero di CPU presenti nel sistema

Process Control Block (PCB) (1/2)

- Il PCB contiene tutte le informazioni utili al S.O. per gestire un determinato processo, soprattutto utili durante il **context switch** e lo **scheduling**.
- Di solito è una struttura (es. struct in C) che può essere inserita in strutture dati dinamiche quali linked list, alberi, ecc...
- Il PCB appartiene al S.O. ed è un'entità distinta rispetto all'area di memoria del processo: contiene però i riferimenti a quest'ultima!

Process Control Block (PCB) (2/2)

- Il PCB è generalmente composto dai seguenti campi:
  - Process ID (PID, identificatore unico del processo)
  - User ID (UID, utente proprietario del processo)
  - Stato del programma (ready, waiting, running, …)
  - Program counter
  - Area per il salvataggio dei registri: SP, accumulatori, general purpose registers, condition codes, ecc...
  - Informazioni per lo scheduling: priorità, puntatori, ecc..
  - Informazioni per la gestione della memoria di processo: indirizzo base, limite, page tables, ecc...
  - Informazioni di accounting: tempo effettivo speso dalla CPU, l'ultima volta che è stato messo in esecuzione, ...
  - Informazioni sullo stato di I/O del processo, ad esempio la lista dei file aperti, ecc...

PCB in Linux

```
struct task_struct
{
  pid_t pid; /* process identifier */
  long state; /* state of the process */
  unsigned int time_slice; /* scheduling information */
  unsigned int  rt_priority; /* scheduling information */
  struct task_struct *parent; /* this process's parent */
  struct list_head children; /* this process's children */
  struct files_struct *files; /* list of open files */
  struct mm_struct *mm; /* address space of this proc. */
  /* … */;
}
```

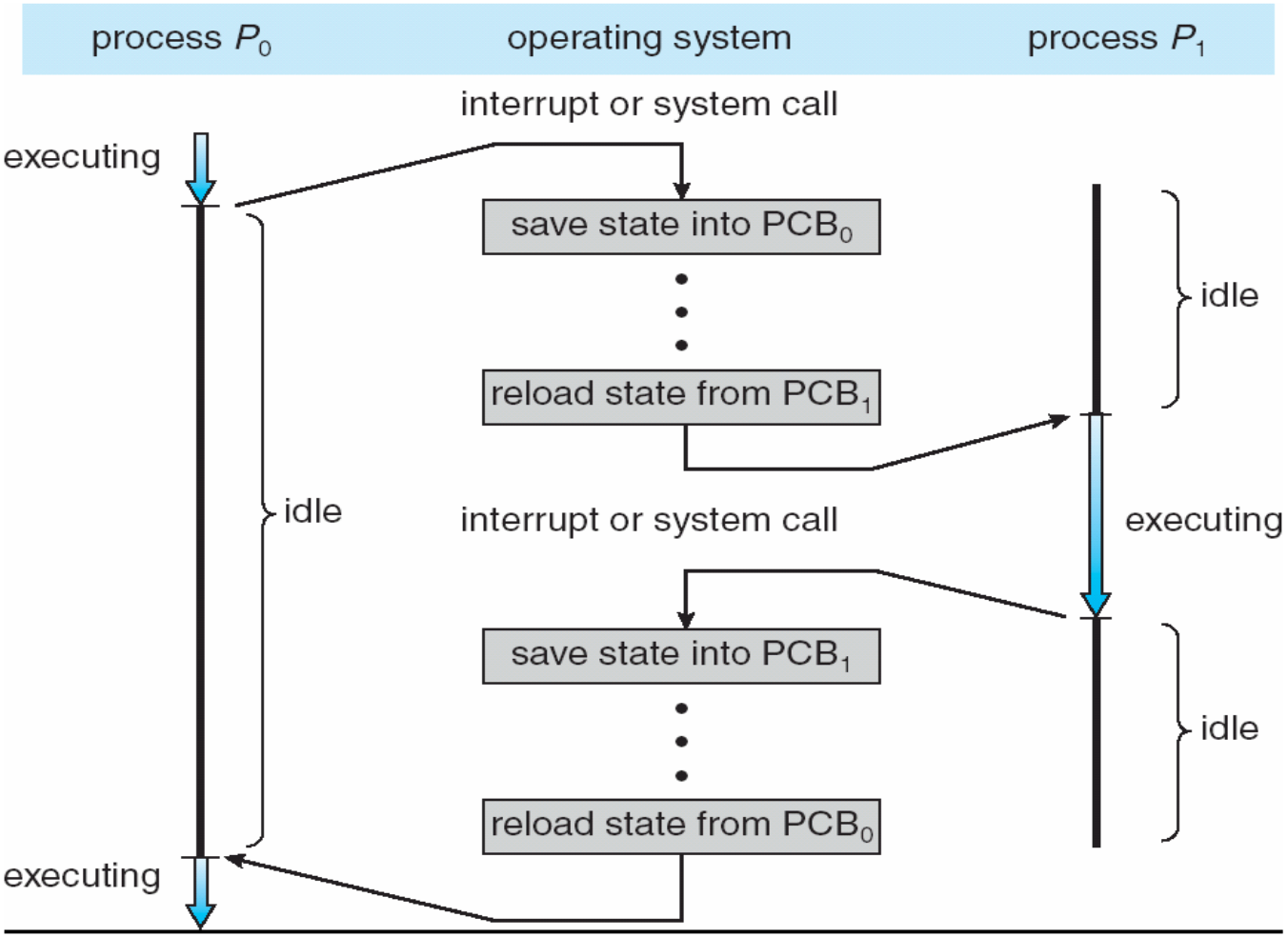
Context switch (1/3)

- Per poter rimuovere dall'esecuzione un processo (es. PID = 0), prima di mettere in esecuzione un nuovo processo (es. PID = 1) il SO deve salvare in memoria una serie di informazioni sullo **stato corrente del processo**, che saranno ripristinate quando il processo con PID = 0 verrà rimesso in esecuzione. Tra di esse si possono segnalare:
  - Contenuto dei registri (PC, SP, registri generici, registri dell'ALU e della FPU, ecc...).
  - Lista dei file aperti, con tutte le informazioni associate
  - …
- Tutte queste informazioni saranno salvate sul **PCB** del processo in esecuzione: (CPU → PCB_0, PCB_1 → CPU).
- Prima di mettere in esecuzione il processo con PID = 1, tutte queste informazioni dovranno essere ripristinate dalla memoria di conseguenza (CPU → PCB_1, PCB_0 → CPU).

Context switch (2/3)

- Il context switch è una fonte di un overhead (surplus di lavoro) non indifferente, generato ad esempio:
  - Tempo impiegato a salvare in memoria lo stato.
  - Blocco e riattivazione delle pipeline di calcolo del processore.
  - **Svuotamento e ripopolamento delle cache**.
- Evidentemente, è necessario evitare di effettuare un numero eccessivo di context switch.

Context switch (3/3)



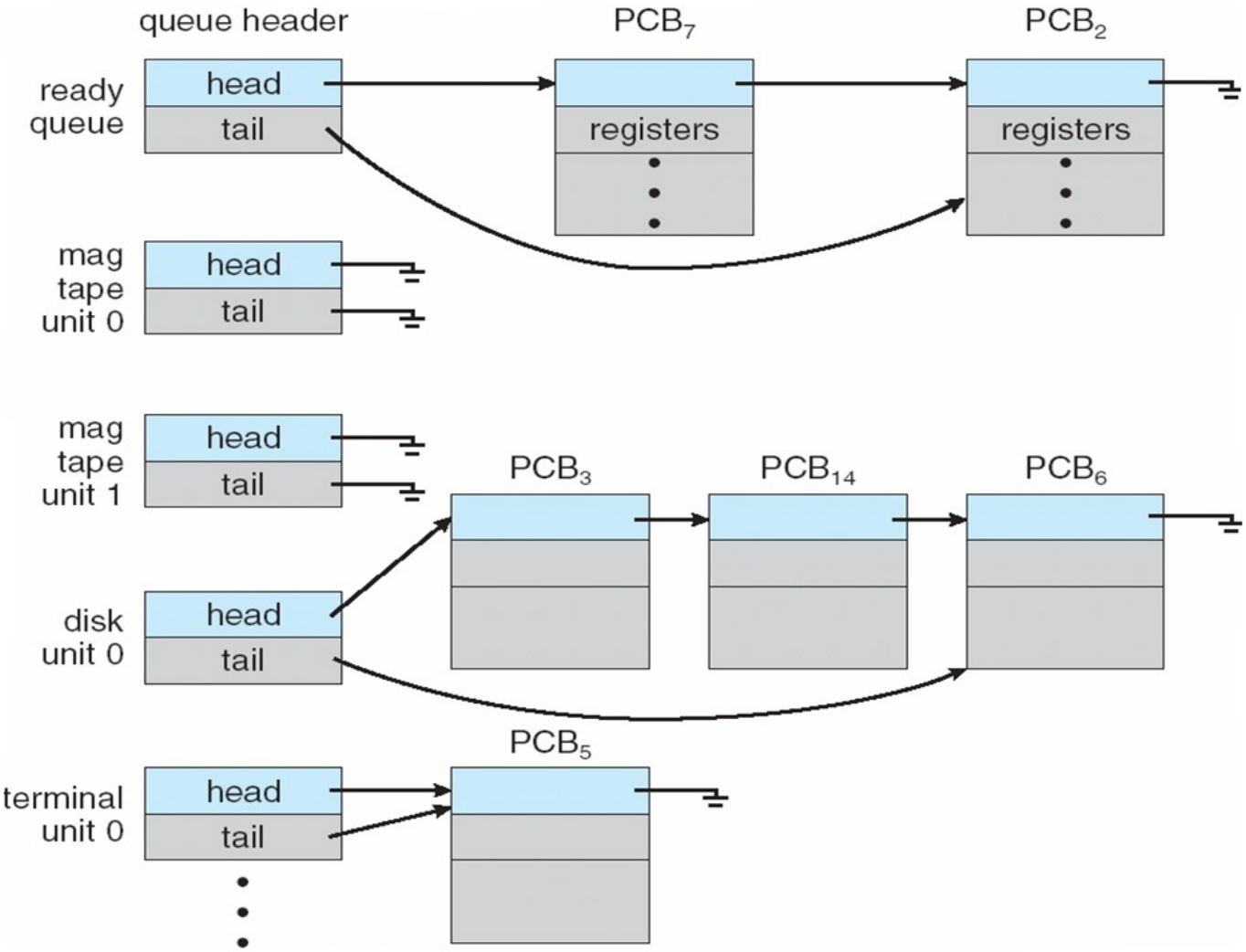
Scheduling dei processi

- Scheduler → sceglie il prossimo processo da eseguire tra quelli in esecuzione
- Dipende dal contesto di utilizzo:
  - Mainframe: massimizzare l'uso delle risorse
  - Desktop e simile: minimizzare i tempi di reazione
- Lo scheduler usa delle code (**queue**) per gestire i processi in esecuzione, di solito implementate come **linked list (o doubly linked list) di PCB**.

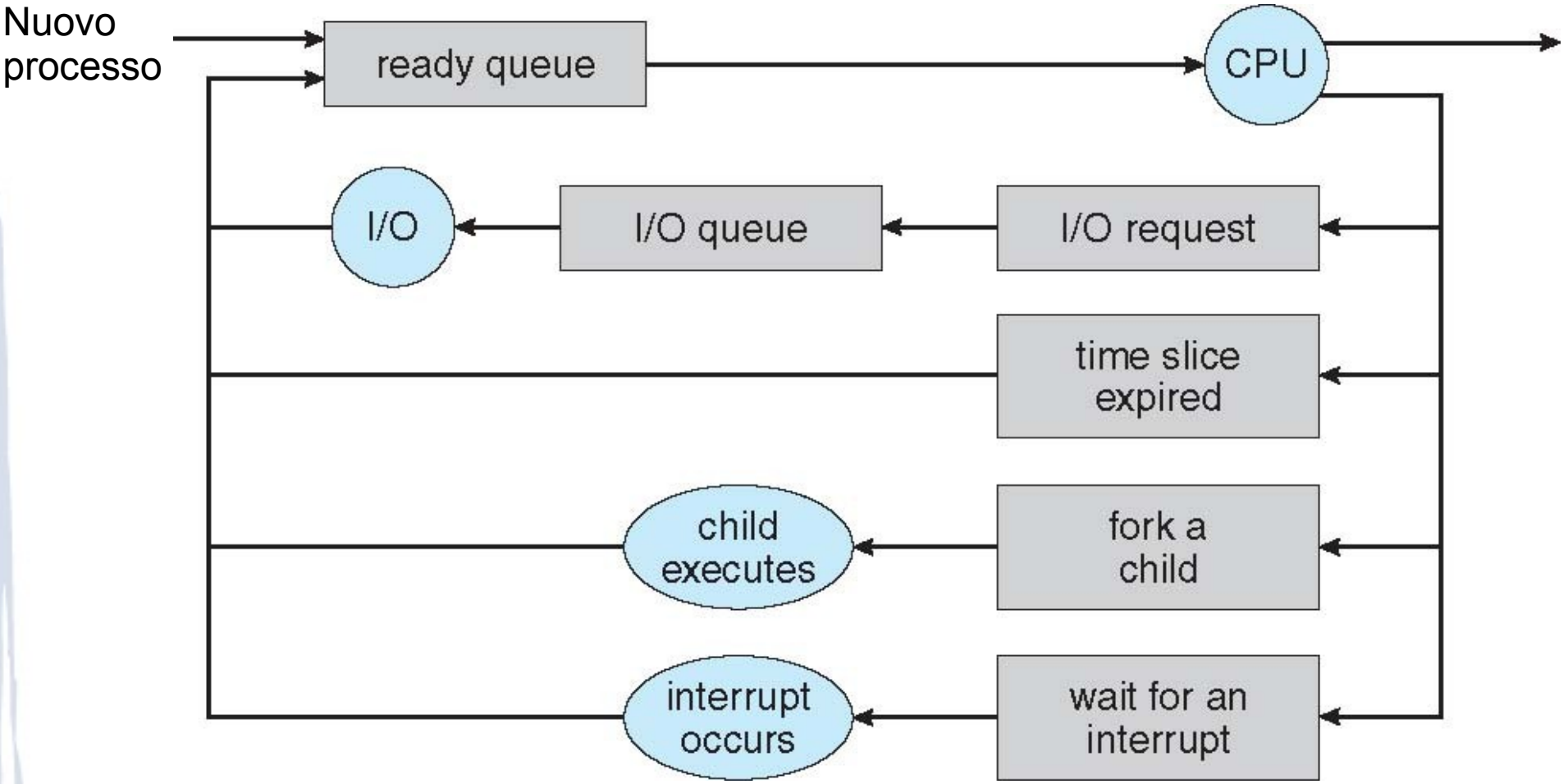
Scheduling queue (1/3)

- Il sistema operativo mantiene aggiornate varie scheduling queues, tra le altre:
  - Job queue: l'insieme di tutti i progetti in esecuzione
  - Ready queue: l'insieme dei processi residenti in memoria principale in attesa, ma pronti per essere eseguiti
  - Device queue: l'insieme dei processi in attesa di qualche operazione di I/O
- Dipendentemente dallo stato del sistema, i processi (ovvero i PCB) vengono spostati da una coda all'altra.

Scheduling queue (2/3)



Scheduling queue (3/3)



Classificazione degli scheduler (1/2)

- Long-term scheduler (o job scheduler): seleziona quali dei processi che possono essere eseguiti immediatamente devono essere spostati in RAM, inserendoli nella ready queue
  - Eseguito sporadicamente
- **Short-term scheduler** (o CPU scheduler): seleziona il prossimo processo da eseguire
  - Eseguito ad ogni intervento del S.O.
  - Spesso l'unico scheduler presente....

Classificazione degli scheduler (2/2)

- Medium-term scheduler: rimuove dalla memoria principale (swap out, RAM → Disk) processi in esecuzione ma inattivi, viceversa riporta in memoria principale (swap in, Disk → RAM) processi che tornano nella ready queue