Gestione dei processi

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

Anno Accademico 2024/2025, Canale San Giovanni

Sommario della lezione

- Sommario della lezione:
  - Definizione di processo
  - Stati di un processo
    - Modello a due stati
    - Modello a tre stati
    - Modello generale
  - Descrittore di un processo
  - Code di processi
  - Cambio di contesto

Riferimenti bibliografici

- Riferimenti
  - P. Ancilotti, M.Boari, A. Ciampolini, G. Lipari, "Sistemi Operativi", Mc-Graw-Hill (Cap.2)
  - www.ostep.org, Cap. 4, Cap. 6 (sez. 6.3)
- Testi di Approfondimento
  - A. Silbershatz, G. Gagne, "Sistemi Operativi" (sesta edizione), Addison-Wesley (Cap. 4)
  - W. Stallings, "Operating Systems : Internals and Design Principles (5th Edition) ", Prentice Hall (Cap. 3)

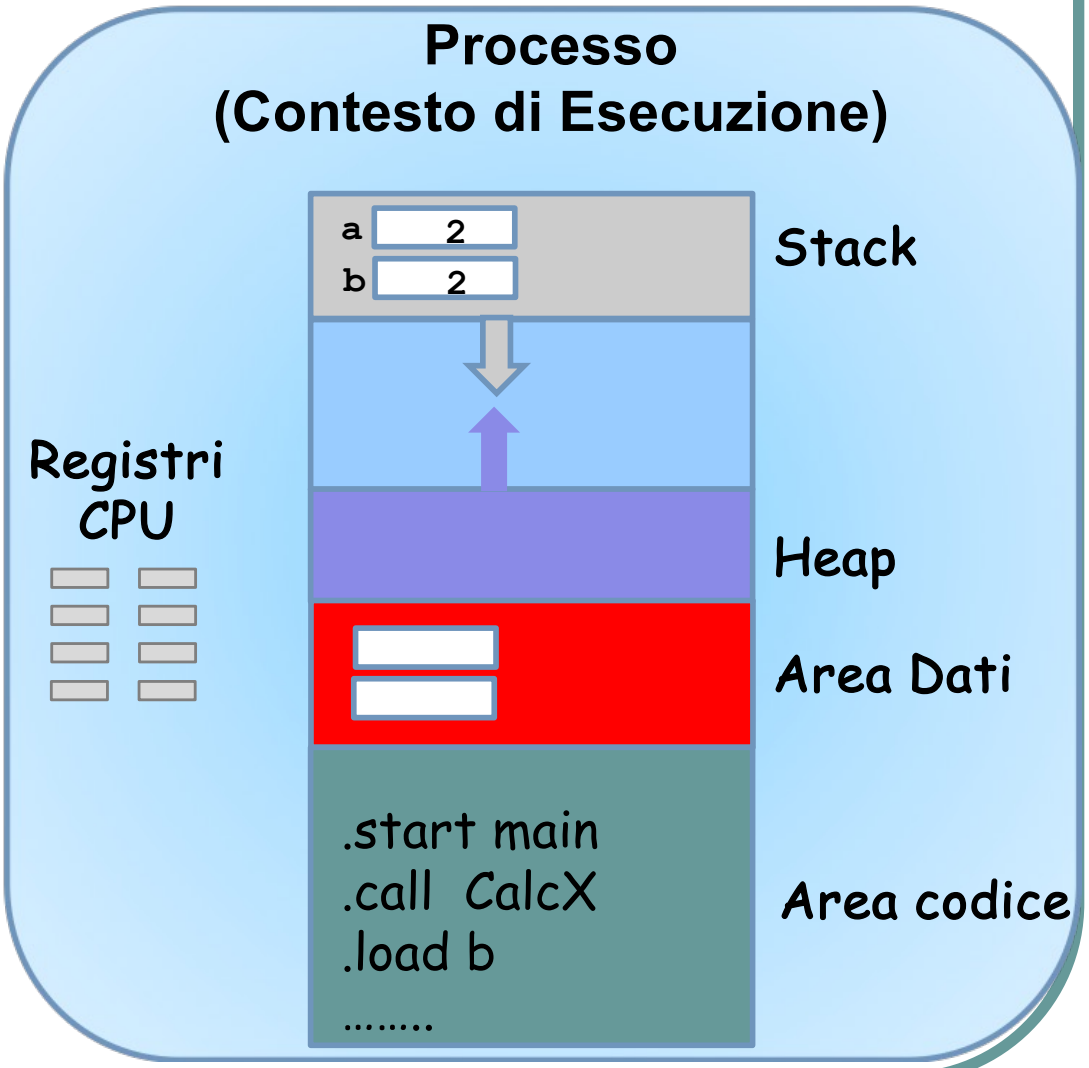
Relazione tra processo e programma

- **Programma**
  - ...è la codifica di un algoritmo in un linguaggio di programmazione, che ne rende possibile l'esecuzione da parte di un elaboratore
  - descrizione **statica** delle elaborazioni da eseguire
- **Processo**
  - ...è l'unità base di esecuzione del SO, che identifica le attività dell'elaboratore relative ad una specifica esecuzione di un programma
  - entità **dinamica** (*programma + contesto di esecuzione*)
- **Processi differenti** possono eseguire **più istanze** di uno **stesso programma**

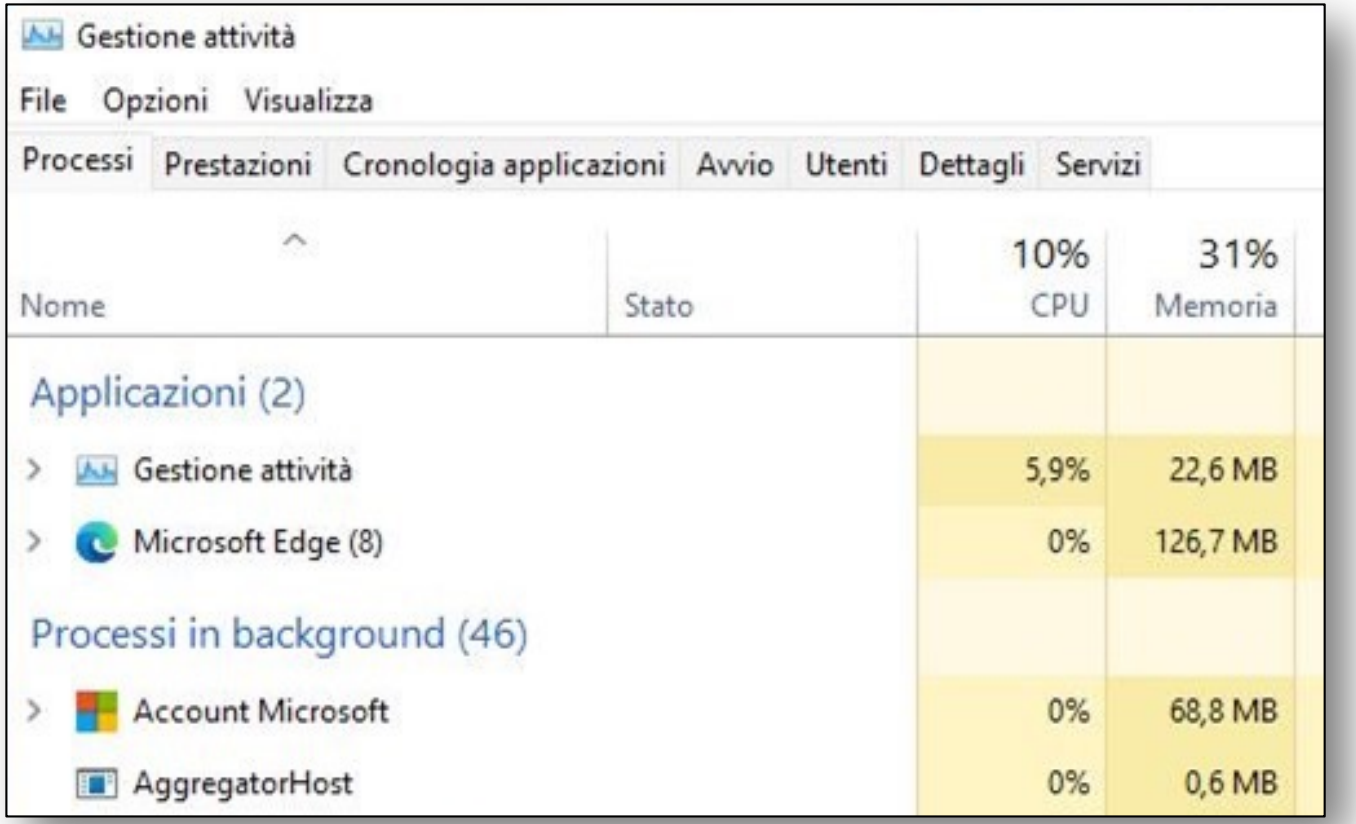
Dal Programma al Processo

Programma

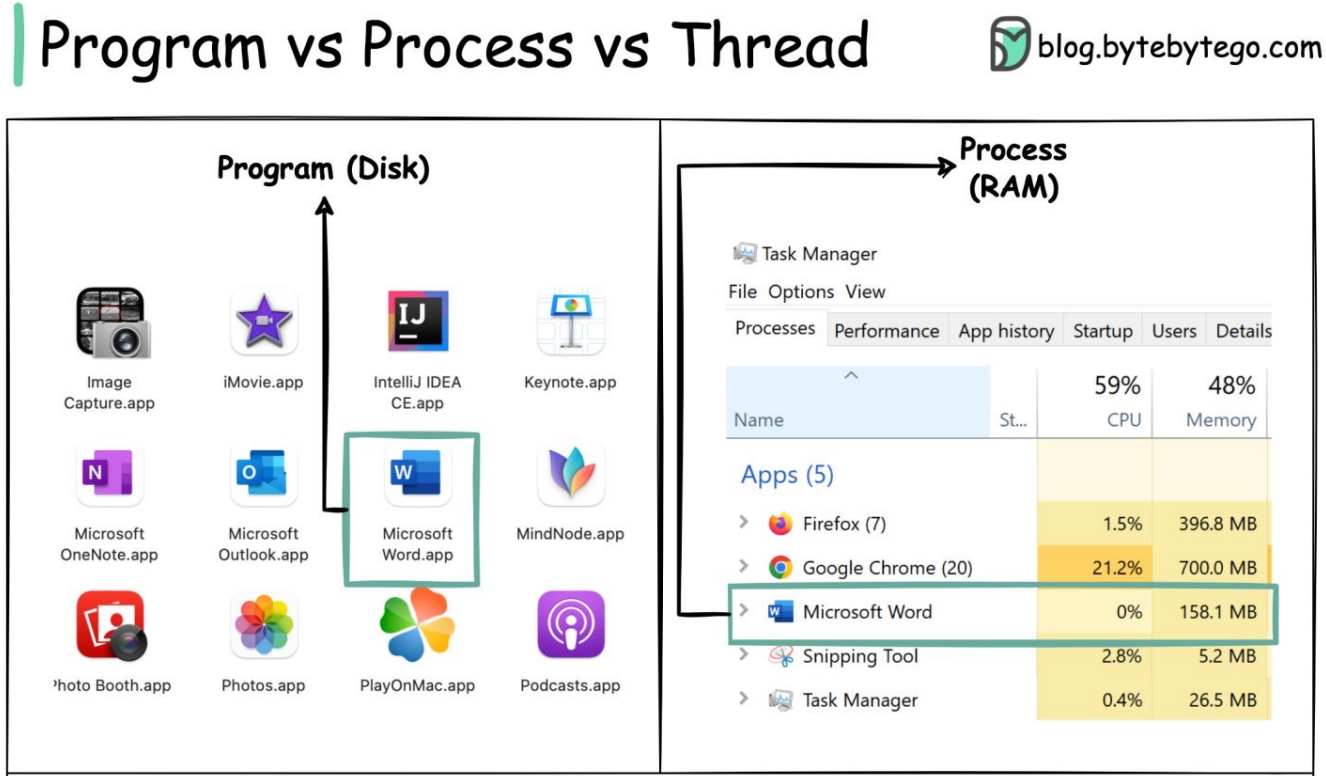
```
void CalcX (int b) {
   …
   if(b == 1) {…}
   …
}

int main() {
   int a = 2;
   CalcX(a);
}
```



Dal Programma al Processo



Dal Programma al Processo

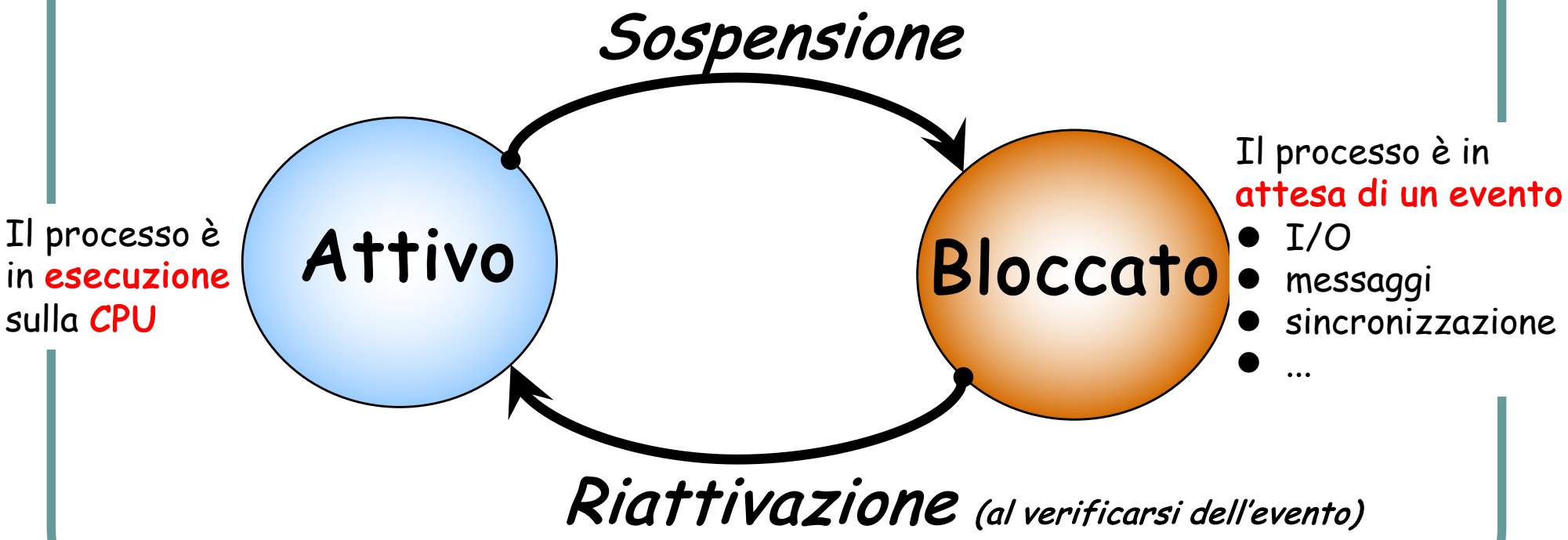


Stati di un processo

- Lo **stato** di un processo rappresenta un'astrazione del suo contesto di esecuzione
- Un processo è soggetto a **transizioni di stato** dovute a:
  - l'attività corrente del processo
  - eventi esterni asincroni con la sua esecuzione

Modello a due stati

In prima approssimazione, un processo può essere caratterizzato da due stati



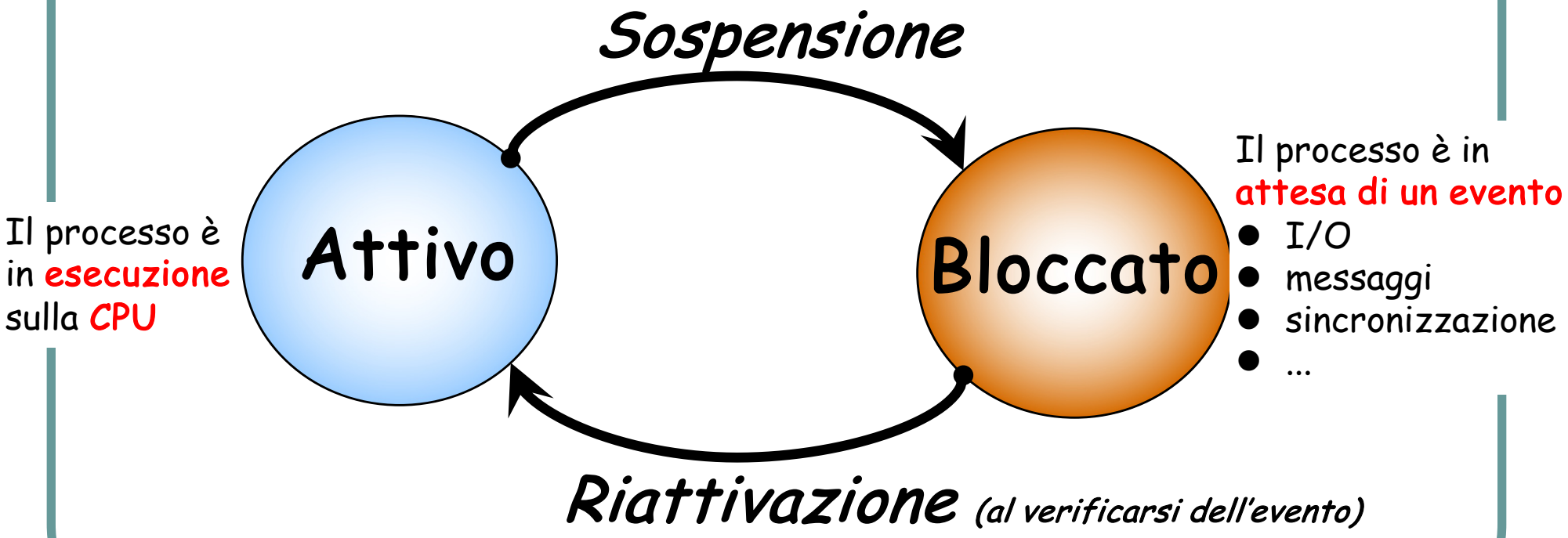
Il processo è in esecuzione sulla CPU

Il processo è in attesa di un evento

- I/O
- messaggi
- sincronizzazione
- ...

Modello a due stati

Problema: come gestire il caso in cui si ha **solo 1 CPU**, e molti **processi attivi**?



Questo modello presuppone che vi siano tante CPU fisiche quanti sono i processi

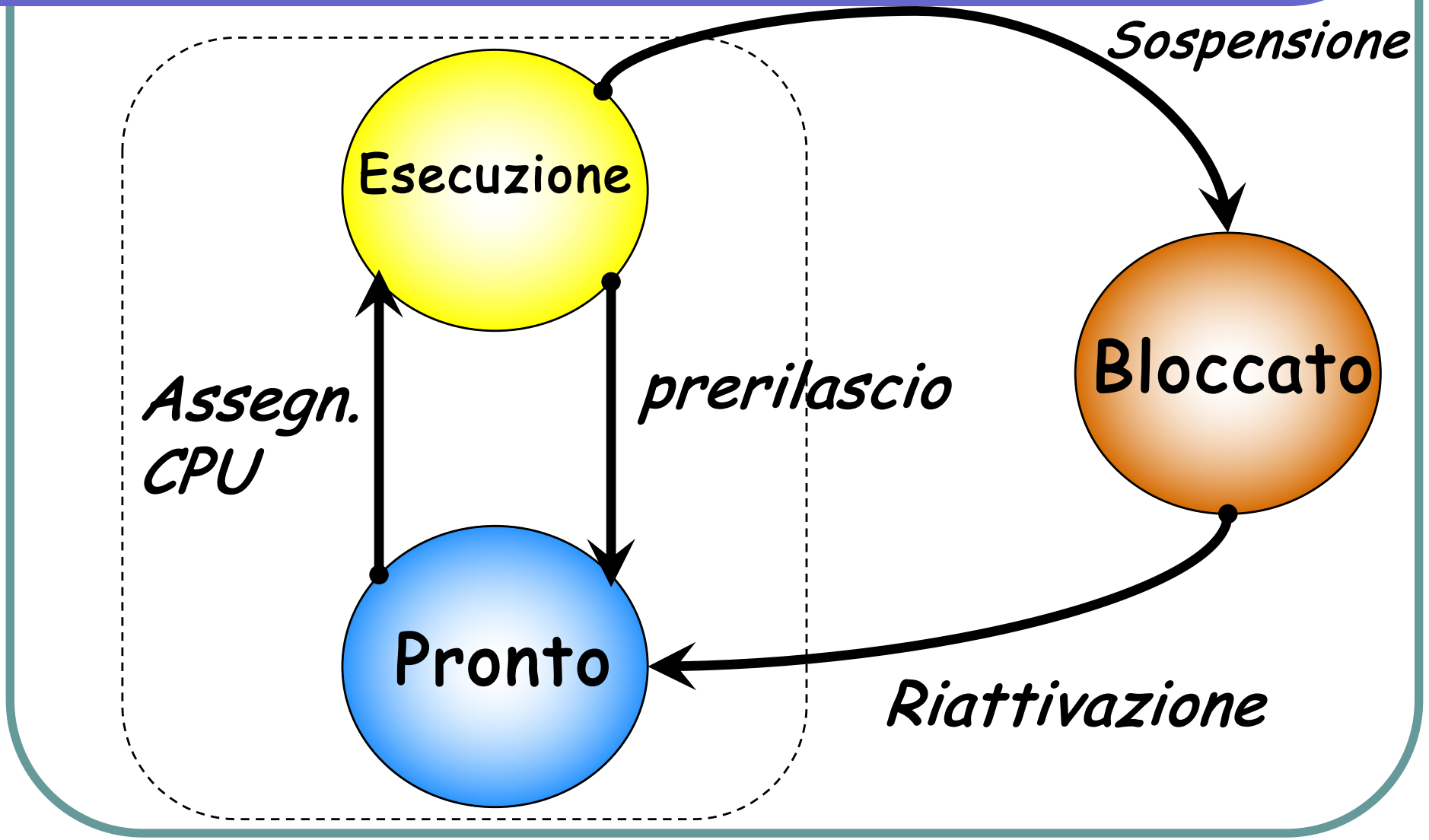
Limiti del modello a 2 stati

- In generale

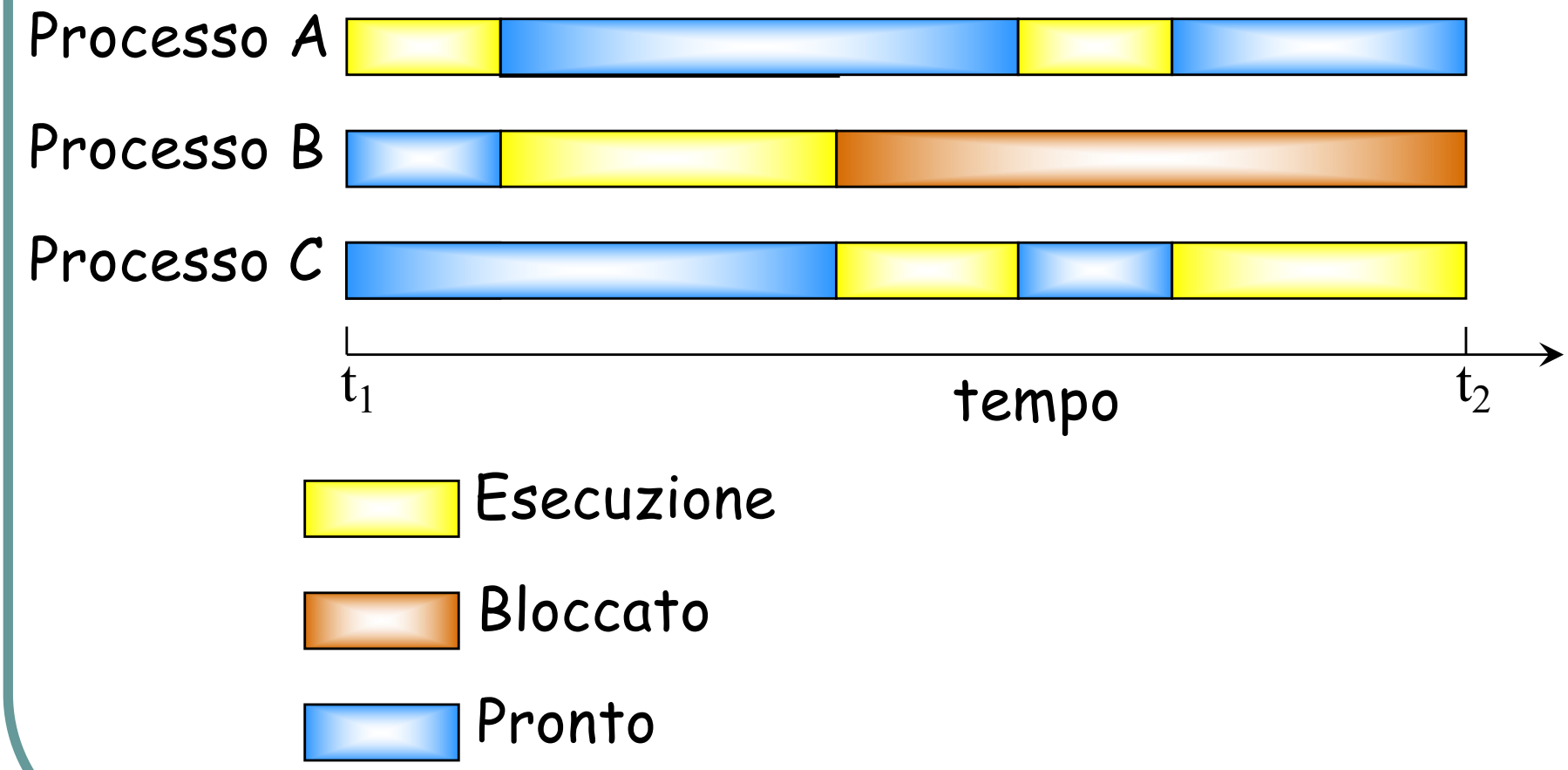
numero di unità di elaborazione **<** numero di processi nel sistema

È necessario distinguere tra processo **pronto** (ad eseguire) e processi **in esecuzione**

Modello a tre stati



Un primo esempio…

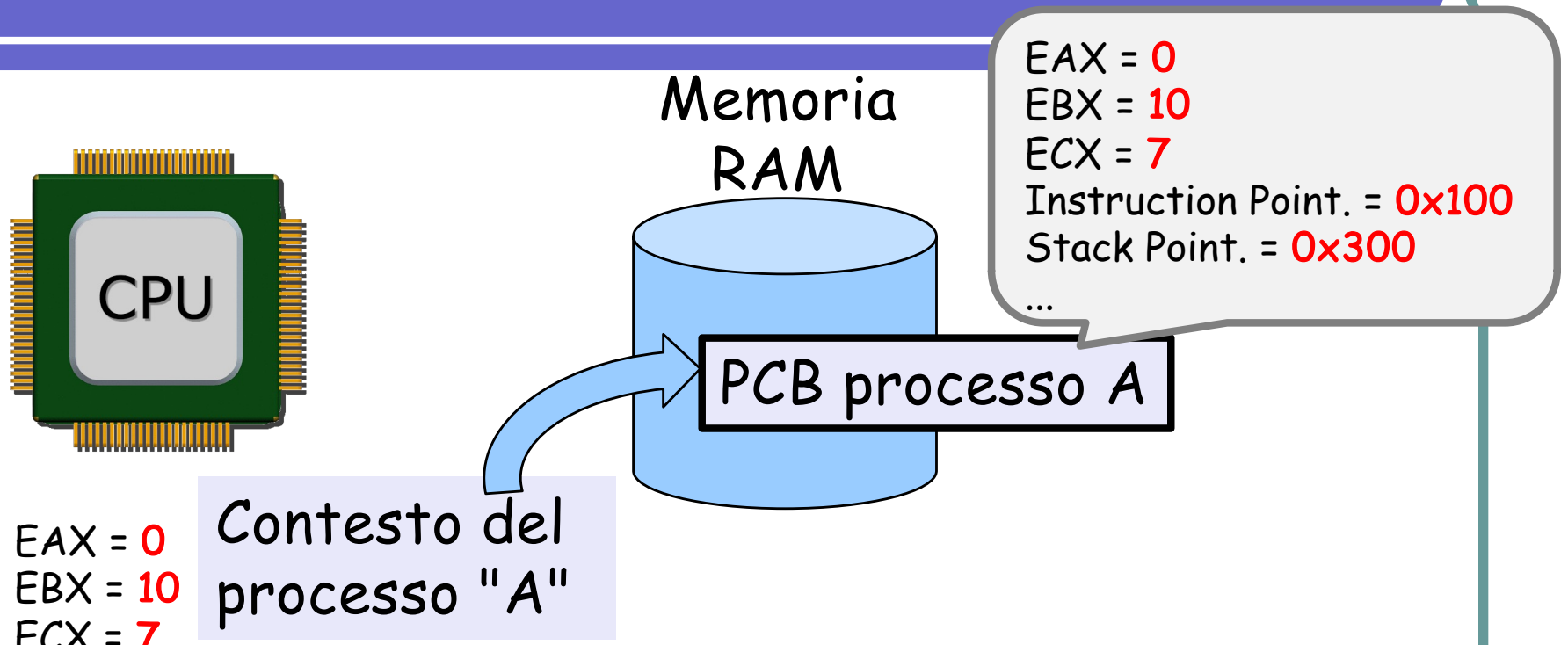


Il cambiamento di contesto

Cambiamento di contesto (**context switch**):
l'insieme di operazioni eseguite dal SO per il **prerilascio** di un processo

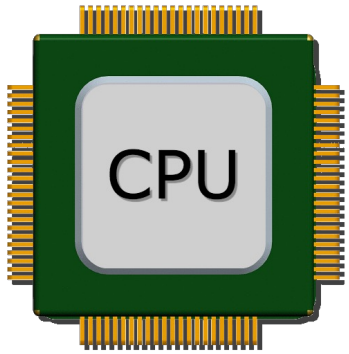
- Il contesto di un processo include le informazioni contenute nei registri del processore
  - Program Counter
  - Stack Pointer
  - Registri general-purpose
  - Registri di gestione della memoria
  - …

Il cambiamento di contesto

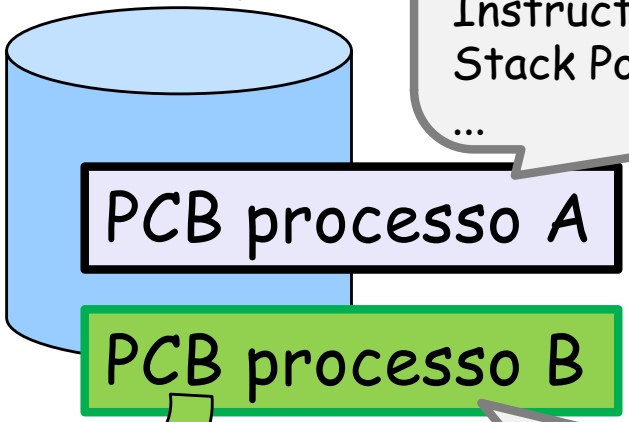


Il **Process Control Block (PCB)** è una **struttura dati del SO** che contiene le informazioni su un processo

Il cambiamento di contesto



Memoria RAM



PCB processo A

PCB processo B

EAX = 0
EBX = 10
ECX = 7
Instruction Point. = 0x100
Stack Point. = 0x300
...

Contesto del processo "B"

EAX = 8
EBX = 20
ECX = 5
Instruction Point. = 0x200
Stack Point. = 0x400
...

EAX = 8
EBX = 20
ECX = 5
Instruction Point. = 0x200
Stack Point. = 0x400
...

Quando avviene un cambio di contesto

A seguito di ….

- Timeout
  - Il quanto di tempo assegnato al processo è scaduto
- System call
  - Il processo richiede un servizio al SO
- Interruzioni di I/O
- Memory fault
  - il processo accede ad un indirizzo di memoria non valido
- Trap
  - CPU exception (può causare la terminazione del processo)

Il cambiamento di contesto

Il cambiamento di contesto è il risultato di tre operazioni

```
Context_Switch() {
        Salvataggio_stato()
        Scheduling_CPU()
        Ripristino_stato ()
}
```

Salva una copia del contesto del processo prelazionato nel PCB

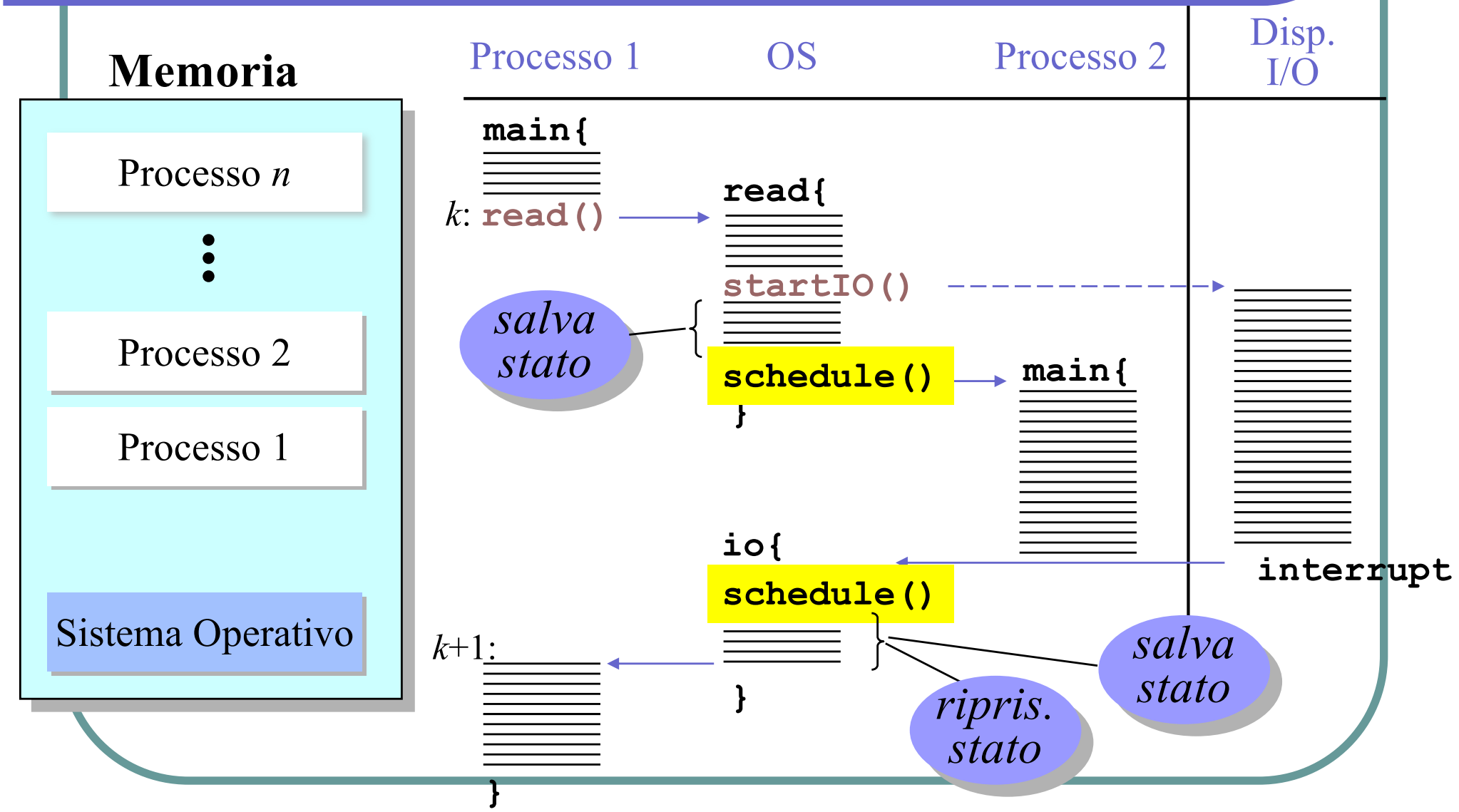
Sceglie il prossimo processo (tra quelli "pronti") da porre in esecuzione

Copia il contesto del processo scelto dal suo PCB ai registri della CPU

L'assegnazione dell'unità di elaborazione

- **Più processi** possono essere nello **stato pronto**
- Si pone il problema di **scegliere quale processo pronto** vada messo in **esecuzione**
- Tale scelta è compiuta dallo **scheduler**, che fa parte in genere del **kernel del SO**

Scheduling e cambio di contesto



Stato “nuovo” e “terminato”

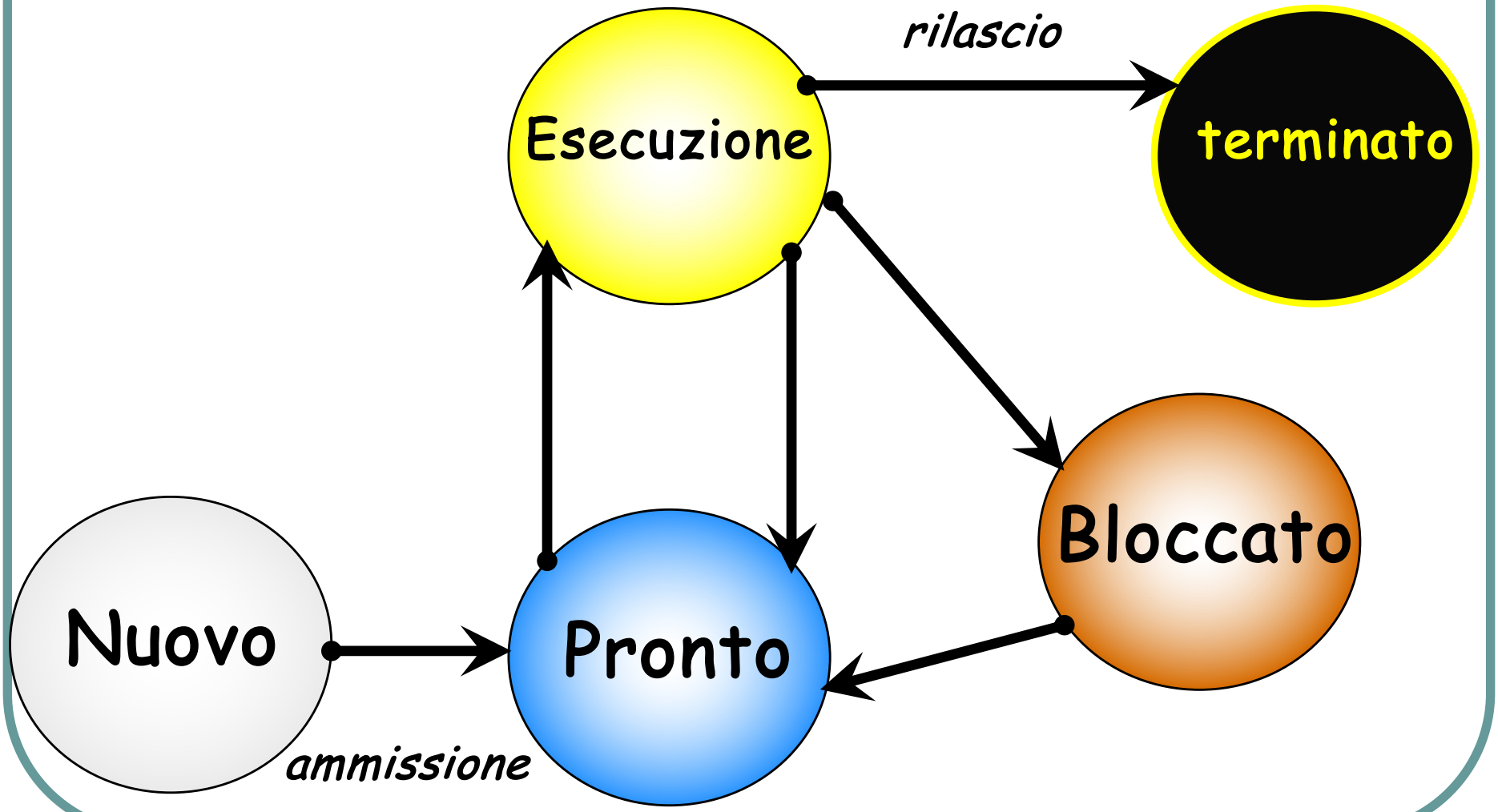
Lo stato “**nuovo**” corrisponde alla creazione di un nuovo processo.

Può avvenire, ad esempio, per la richiesta da parte di un processo già esistente.

Lo stato “**terminato**” corrisponde ad una terminazione del processo, che può essere:

- **Terminazione normale**. Il processo esegue una chiamata al SO per indicare il completamento delle sue attività
- **Terminazione anomala**, che può essere provocata, ad esempio, da un utilizzo scorretto delle risorse

Modello a 5 stati



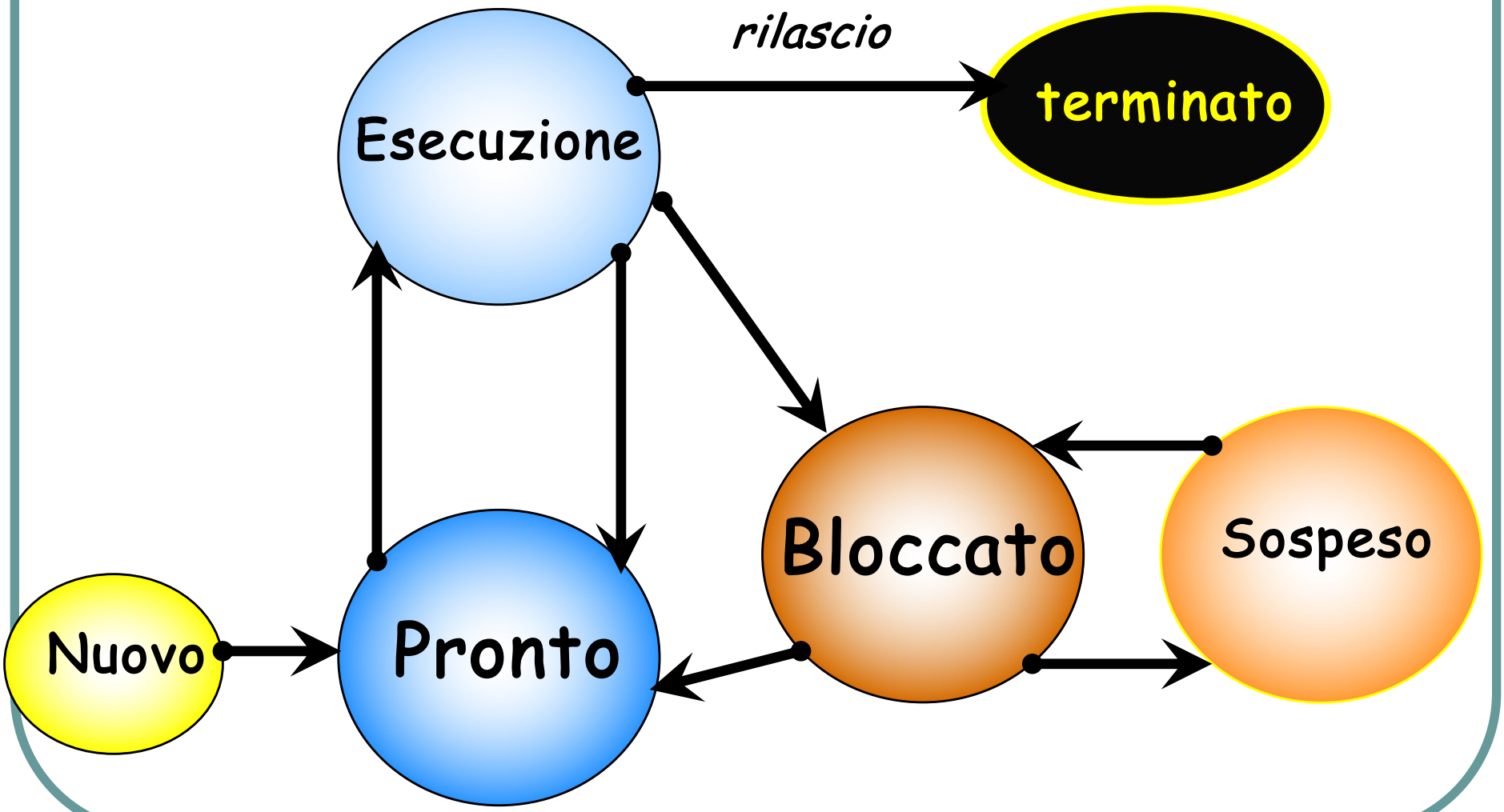
La presenza dei processi swapped

- I SO prevedono la possibilità di **spostare temporaneamente** un processo dalla RAM (**swapping**)
- Si libera spazio per altri processi

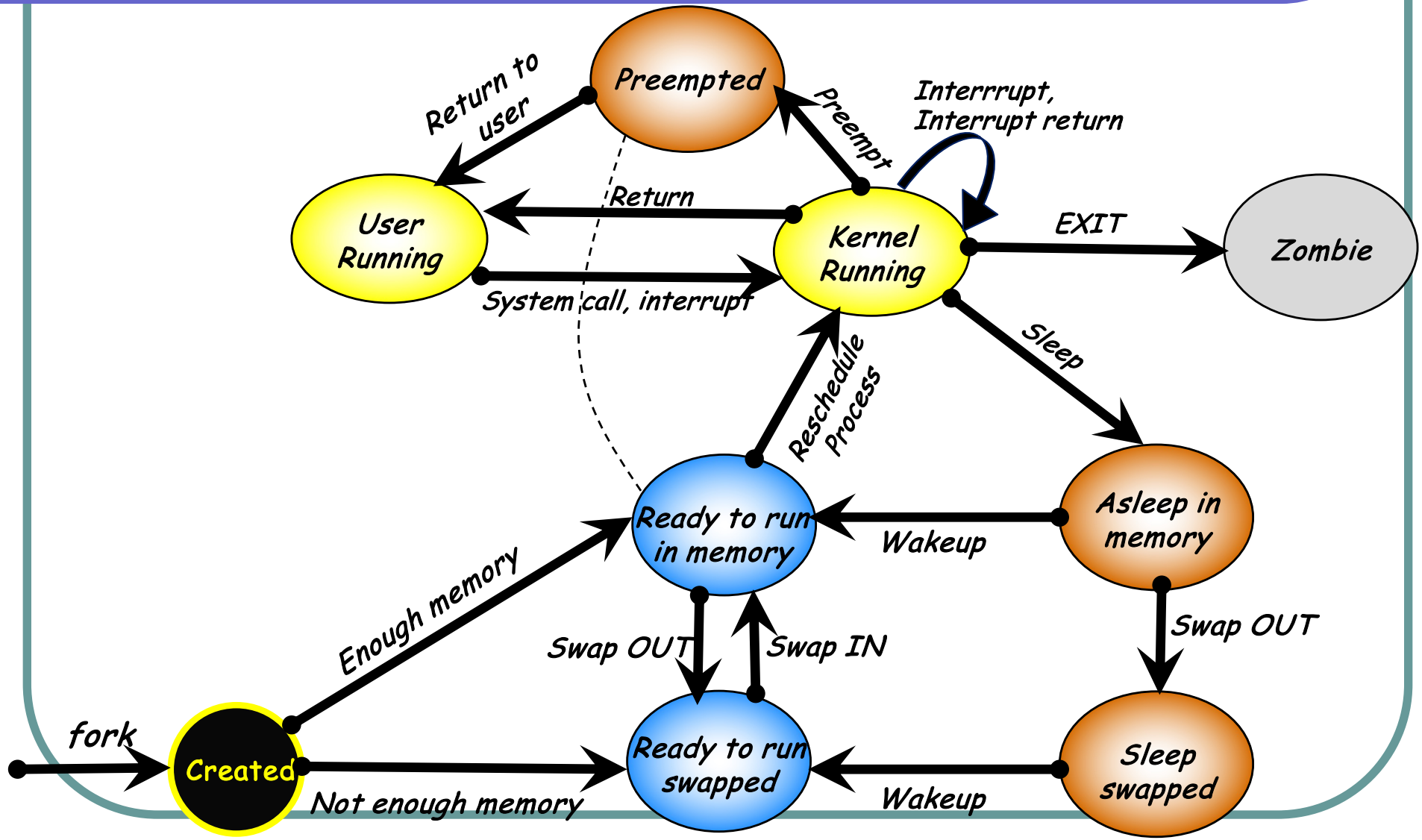
memoria principale (RAM) → memoria secondaria (dischi)

…per tali sistemi occorre aggiungere un ulteriore stato detto "**sospeso**"

Modello generale



Stati di un processo Unix



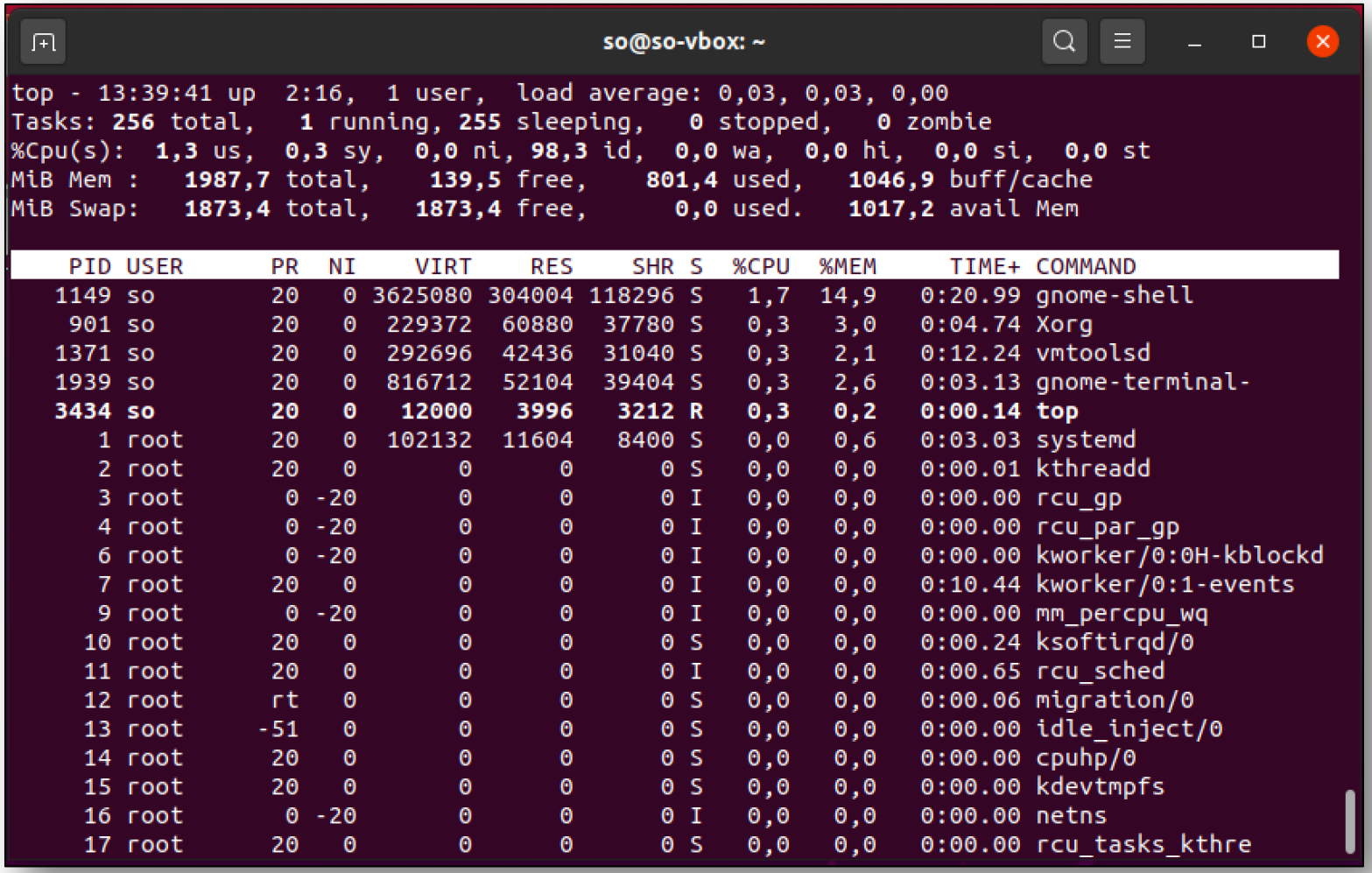
UNIX: lo stato *Zombie*

- Stato in cui un processo ha **terminato**
- Ma non può essere ancora eliminato perché la sua **immagine di memoria è ancora necessaria**

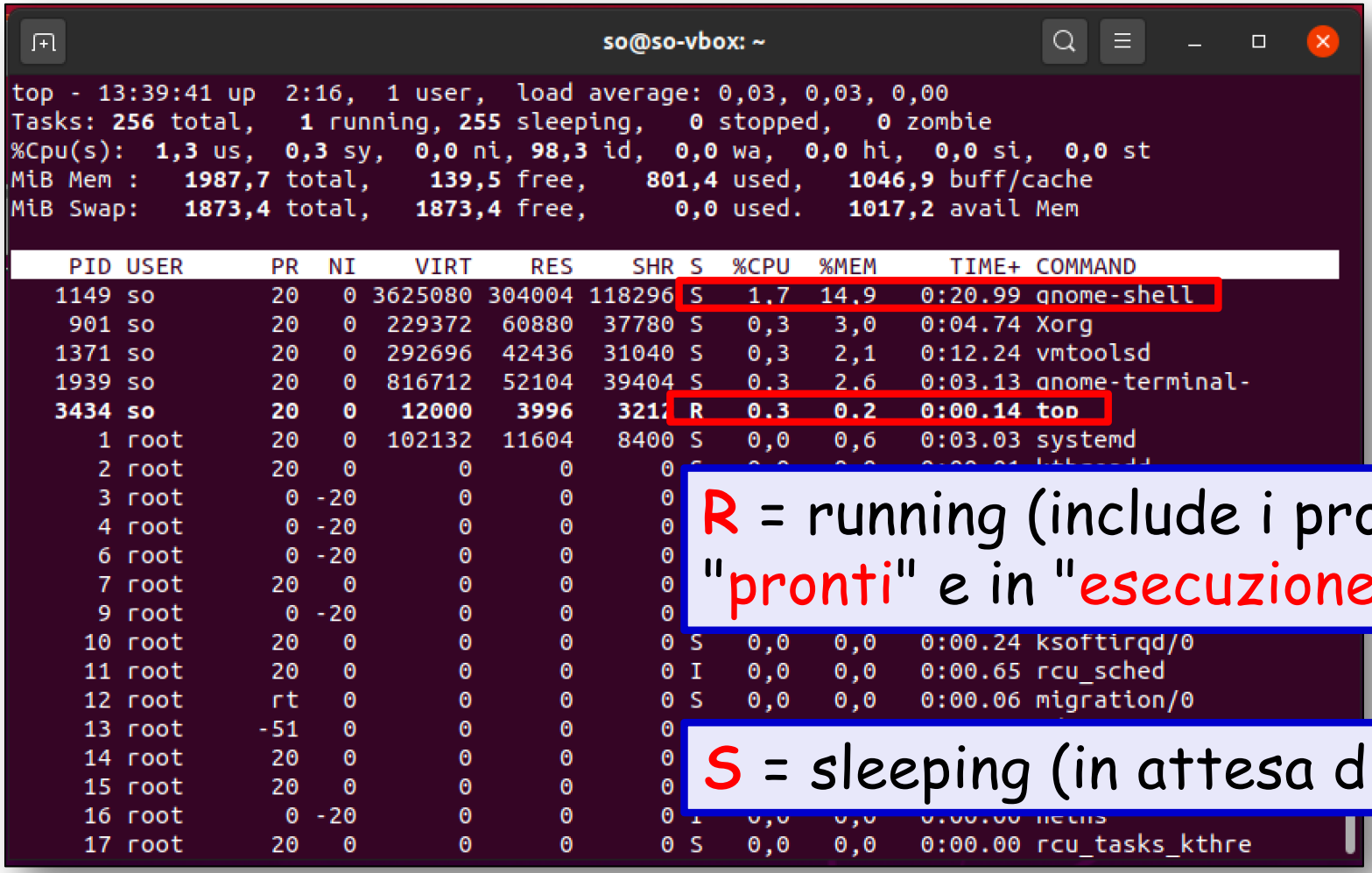
Esempio tipico:

- un processo (padre) avvia un altro processo (figlio), per **delegargli delle attività**
- il processo figlio **termina (prima del padre)**
- il processo figlio è "*zombie*" finchè il padre non ne raccoglie lo **stato di terminazione** (esito dell'esecuzione)

Il comando top



Il comando top



R = running (include i processi "pronti" e in "esecuzione")

S = sleeping (in attesa di I/O)

Il comando ps

ps aux

- Stampa una istantanea dei processi:
  - Identificatore (PID)
  - Lo stato
  - Il terminale assegnato
  - Il tempo di CPU usato
  - Il nome del comando corrispondente al processo
- Per default mostra solo i processi che sono "visibili" dall'utente, ovvero che usano un terminale
- Con l'opzione "x" vengono mostrati anche i processi senza terminale (processi in background o "daemon")

Descrittore di un processo (1/2)

- Ad ogni processo è associata una struttura dati (es. una "struct" in linguaggio C)
- "Descrittore del Processo" (**PCB - Process Control Block**)
- I PCB sono raggruppati in una tabella dei processi (**Process Table**)

Descrittore di un processo (2/2)

- Le informazioni da registrare nel descrittore possono essere così classificate:
  - Nome del Processo
  - Stato del processo
  - Modalità di servizio (ad es. priorità, deadline)
  - Informazioni sulla gestione della memoria
  - Contesto del processo (copia dei registri della CPU)
  - Utilizzo delle risorse
  - Identificatore del processo successivo

PCB in UNIX

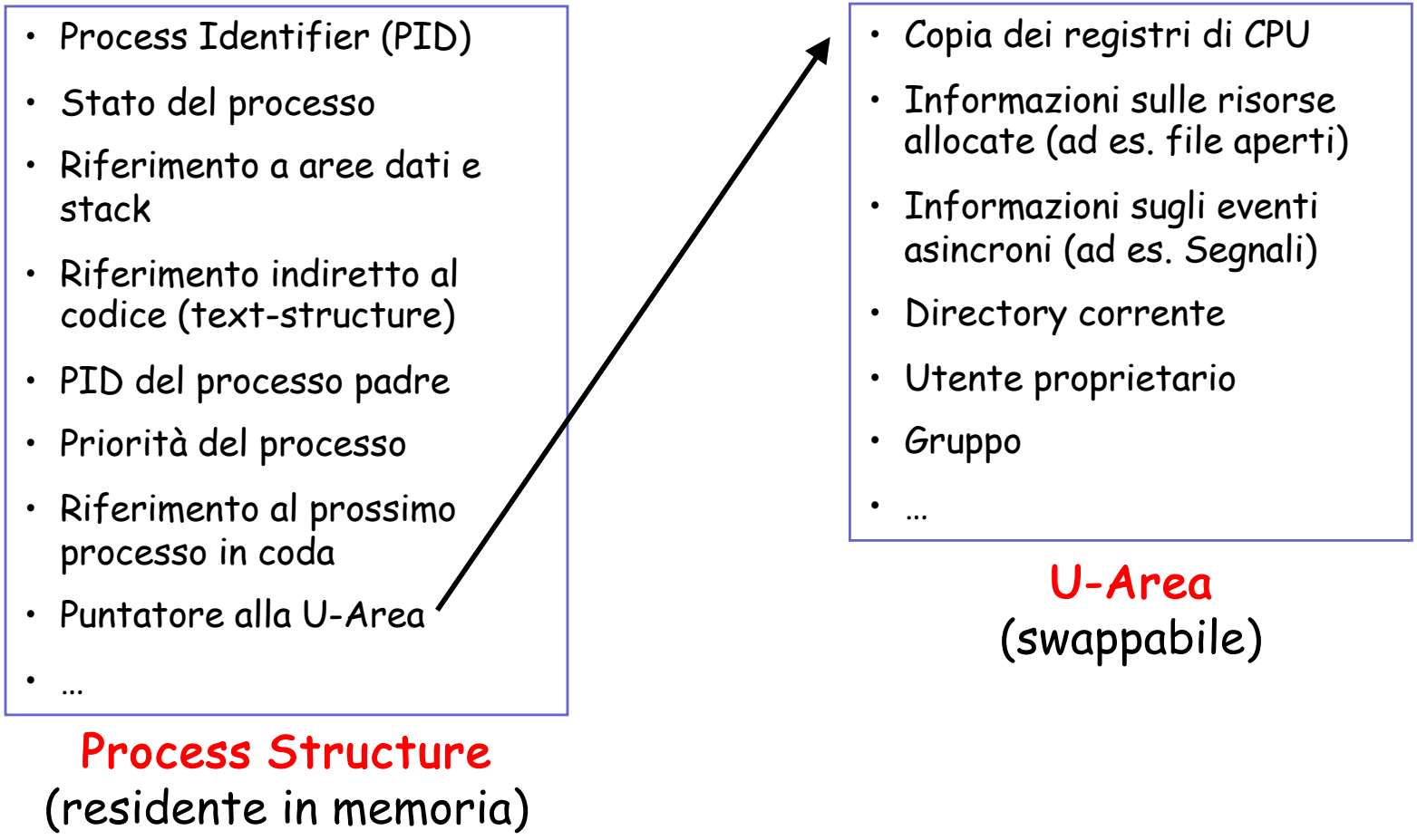
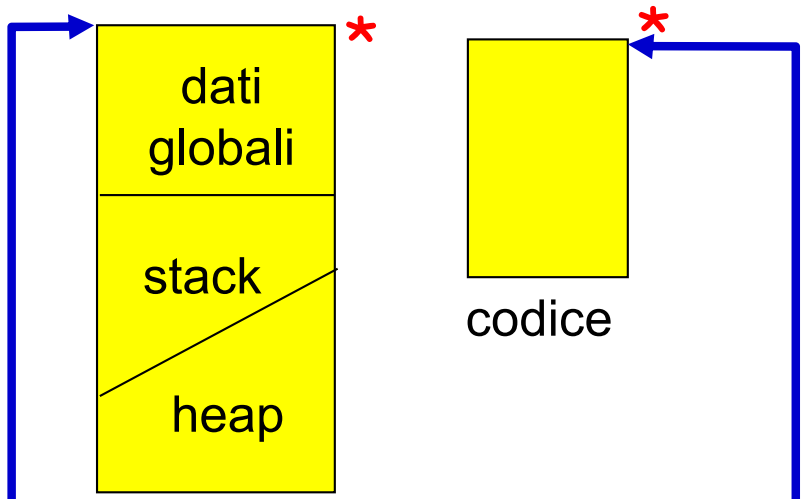

• Process Identifier (PID)
• Stato del processo
• Riferimento a aree dati e stack
• Riferimento indiretto al codice (text-structure)
• PID del processo padre
• Priorità del processo
• Riferimento al prossimo processo in coda
• Puntatore alla U-Area
• …
Process Structure
(residente in memoria)
• Copia dei registri di CPU
• Informazioni sulle risorse allocate (ad es. file aperti)
• Informazioni sugli eventi asincroni (ad es. Segnali)
• Directory corrente
• Utente proprietario
• Gruppo
• …
U-Area
(swappabile)

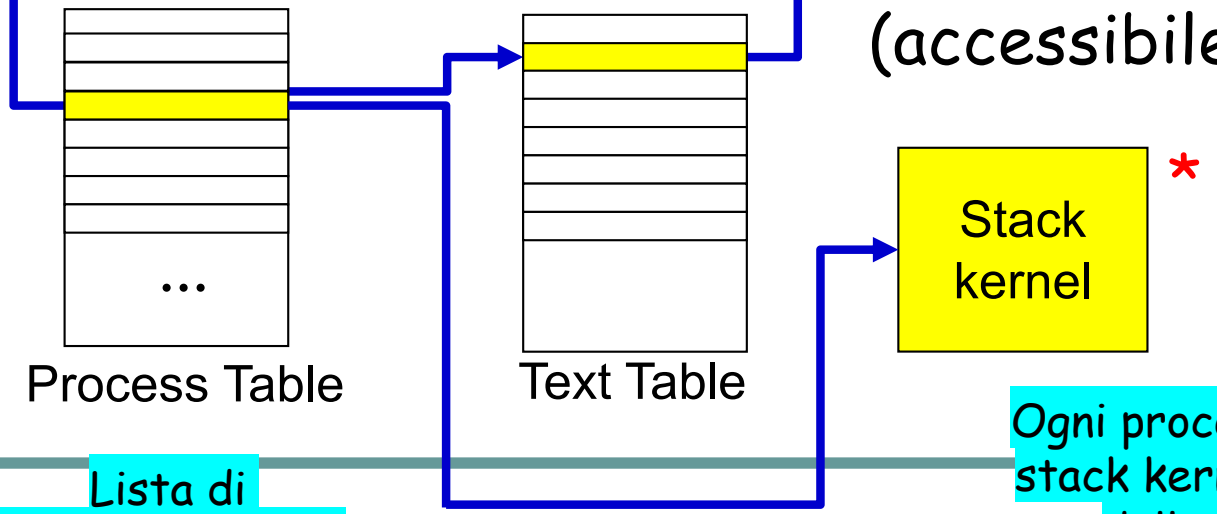
Immagine di un processo Unix



* Le aree user-space e lo stack kernel sono "swappabili" su disco in caso di scarsità di memoria. Le tabelle sono fisse in memoria.

User-space
(memoria accessibile dal programma)

Kernel-space
(accessibile solo dal SO)

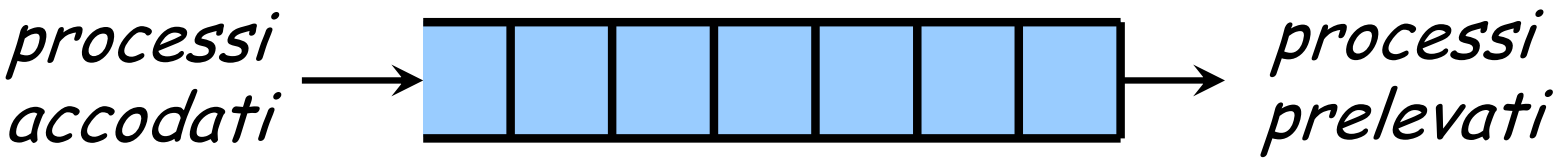


Lista di descrittori (PCB)

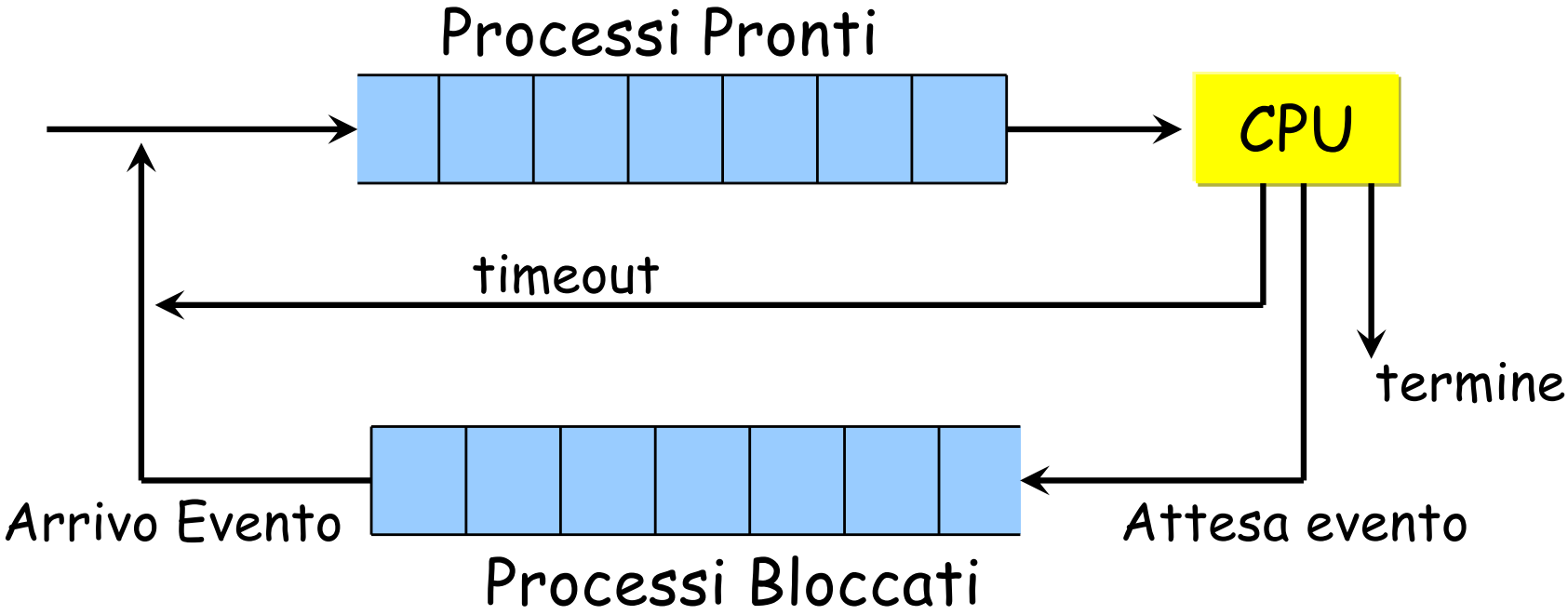
Ogni processo è dotato di uno stack kernel per l'esecuzione delle sue system call

Code dei processi

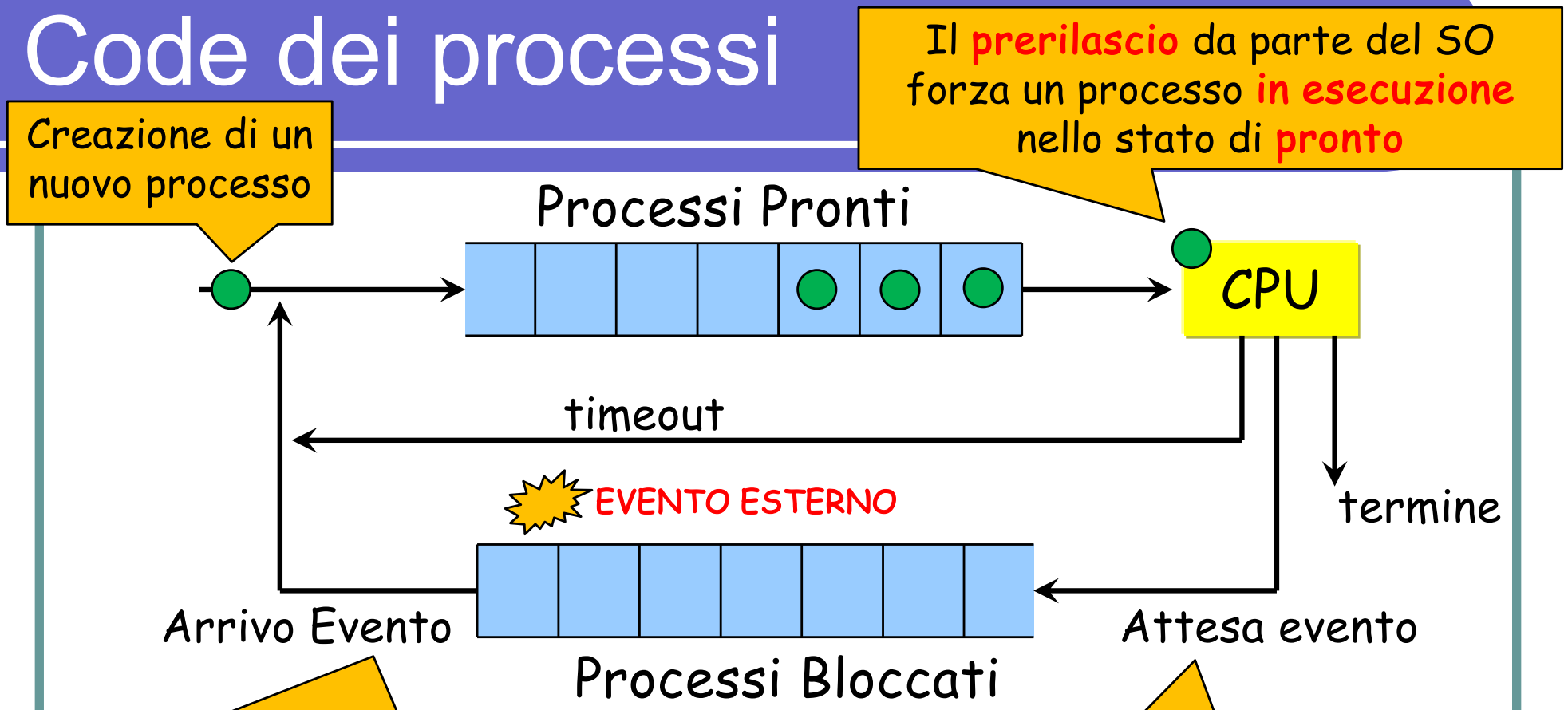
- Il SO tiene traccia dei processi nel sistema utilizzando delle **code**
  - Coda (una o più) dei processi pronti
  - Coda (una o più) dei processi **bloccati** (in attesa di eventi)



Code dei processi



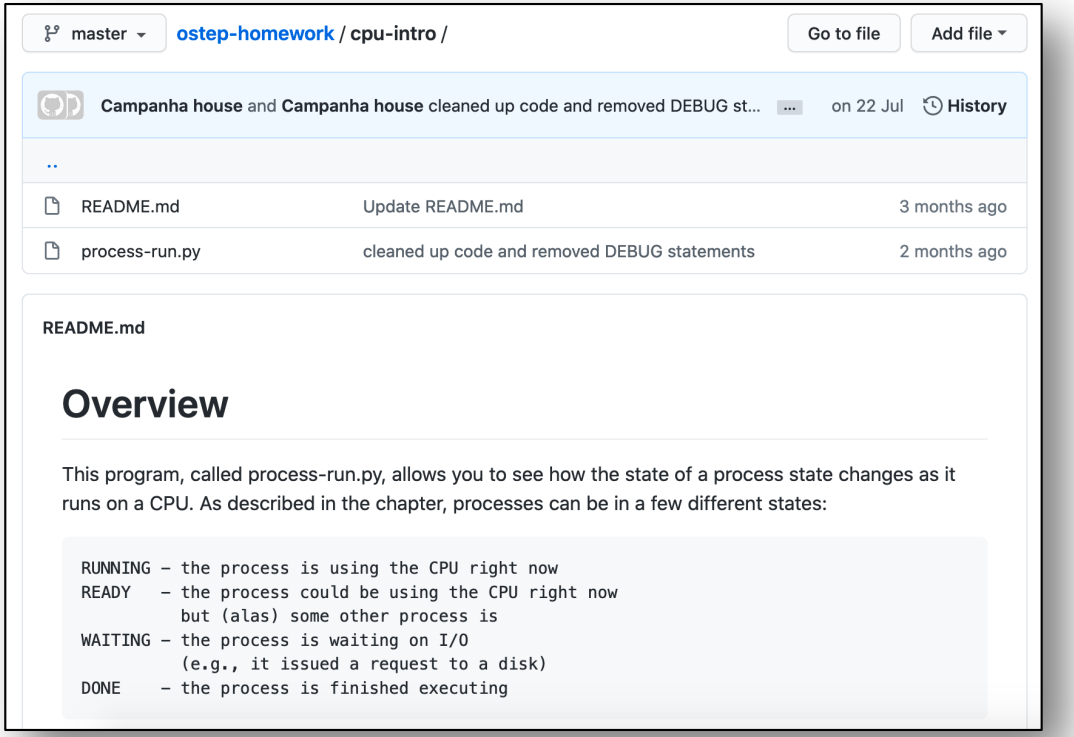
Code dei processi



Domanda: il processo sbloccato va messo in testa oppure in coda tra i processi pronti?

Simulazione processi

- Repository delle simulazioni per il corso: https://github.com/remzi-arpacidusseau/ostep-homework
- Simulatore degli stati dei processi: https://git.io/JUHqQ



Simulazione processi

- Senza l'opzione "`-c`", il programma descrive uno scenario, ma senza darne la soluzione
- Con le opzioni "`-c`" e "`-p`", mostra la soluzione e alcune percentuali
- Altri parametri:
  - `-l X1:Y1,X2:Y2,...` (lista dei processi)
    dove X è il numero di istruzioni, Y è la probabilità di fare I/O
  - `-L <INTERO>` (lunghezza delle operazioni di I/O)
  - `-S SWITCH_ON_END` (simula sistema monoprogrammato)
  - `-S SWITCH_ON_IO` (simula sistema multiprogrammato)
  - `-I IO_RUN_LATER` (dopo I/O, stato RUNNABLE)
  - `-I IO_RUN_IMMEDIATE` (dopo I/O, stato IN ESECUZIONE)

Esemio di simulazione

- Due simulazioni di esempio:
  1. **IO_RUN_LATER**: Un processo che si riattiva dallo stato bloccato, viene posto in fondo alla coda dei processi pronti
  2. **IO_RUN_IMMEDIATE**: Un processo che si riattiva dallo stato bloccato, viene posto in cima alla coda dei processi pronti

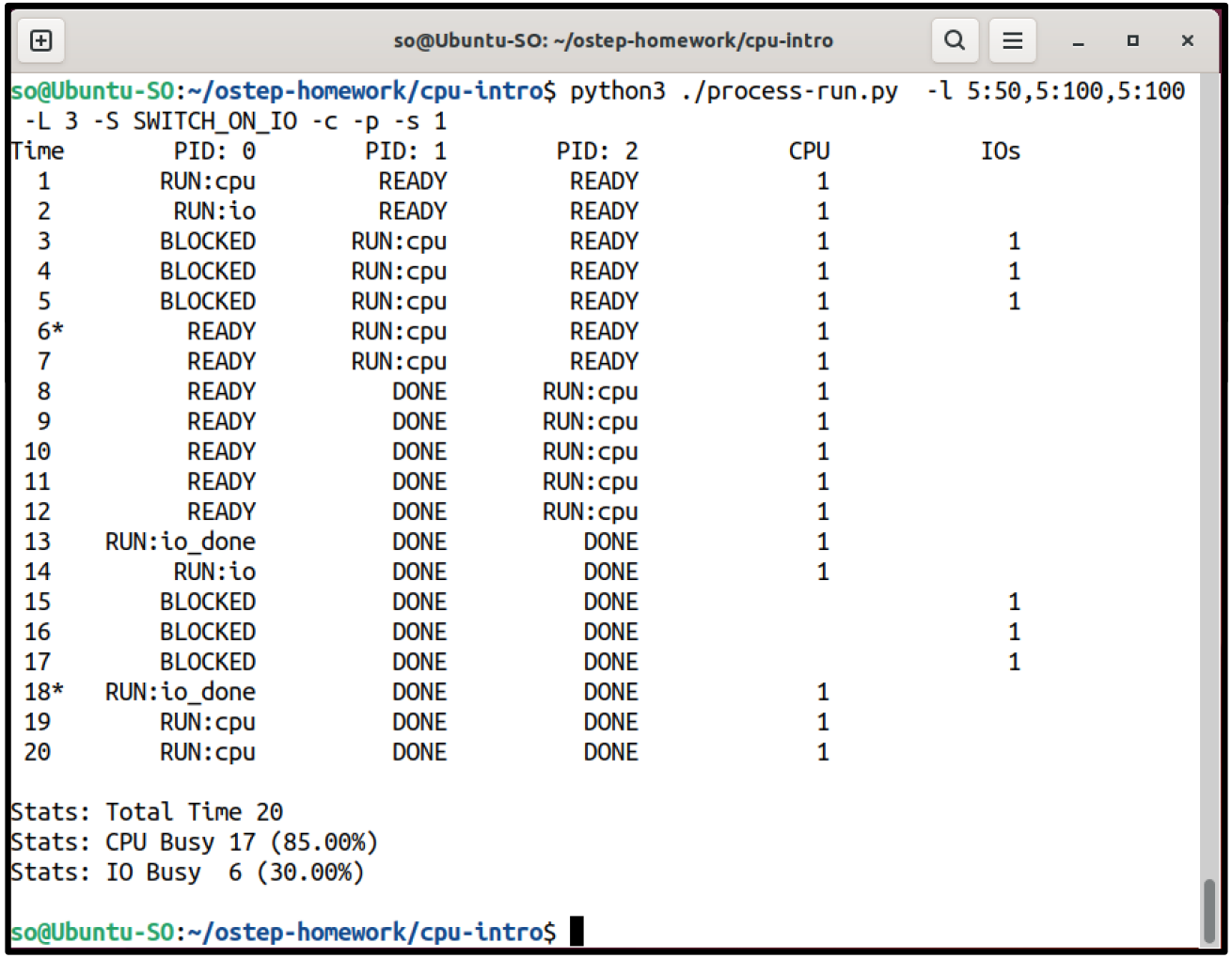
Esempio di simulazione

- Si hanno tre processi
  - Due processi usano **solo la CPU** (100%)
  - Un processo usa **I/O nel 50%** delle operazioni, **50% CPU**
  - Ognuno fa **5 operazioni**
  - Quale sarà la percentuale di **utilizzo della CPU totale**?

```
$ python3 ./process-run.py
          -l 5:50,5:100,5:100
          -L 3
          -S SWITCH_ON_IO
          -c -p
          -s 1
```

"-s" configura il "seed" dei numeri casuali. A parità di seed, verranno prodotti sempre gli stessi valori

Esempio di simulazione

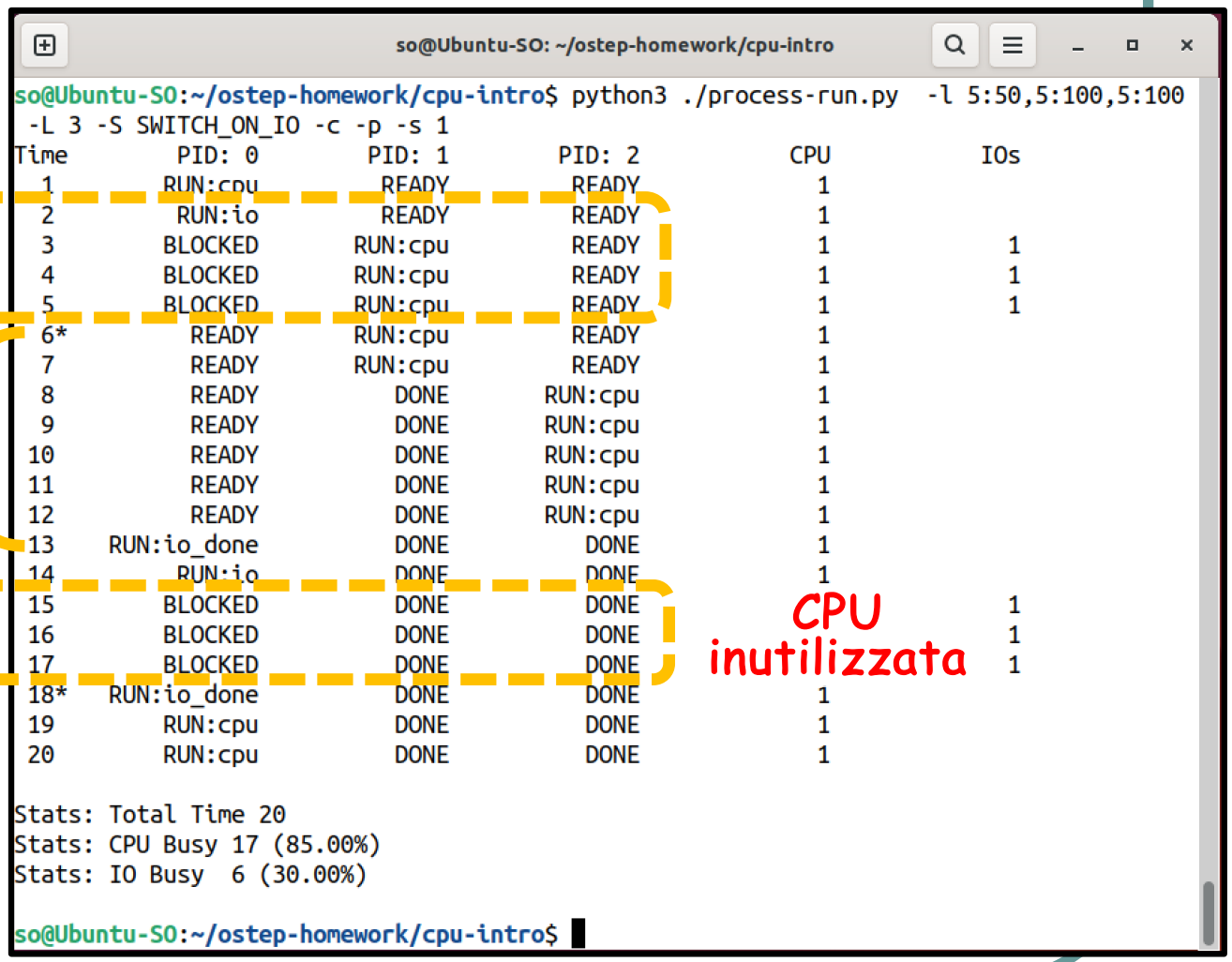


Esempio di simulazione

Mentre il processo 0 si sospende, il processo 1 è pronto a usare la CPU

Il processo 0 è in fondo alla coda dei pronti (IO_RUN_LATER)

Quando il processo 0 si sospende, non c'è nessun processo pronto a usare la CPU



Esempio di simulazione

Diamo ora priorità al processo che fa I/O
(opzione `-I IO_RUN_IMMEDIATE`)

Time	PID: 0
1	RUN:cpu
2	RUN:io
3	BLOCKED
4	BLOCKED
5	BLOCKED
6*	READY
7	READY
8	READY
9	READY
10	READY
11	READY
12	READY
13	RUN:io_done
14	RUN:io
15	BLOCKED
16	BLOCKED
17	BLOCKED
18*	RUN:io_done
19	RUN:cpu
20	RUN:cpu

IO_RUN_LATER

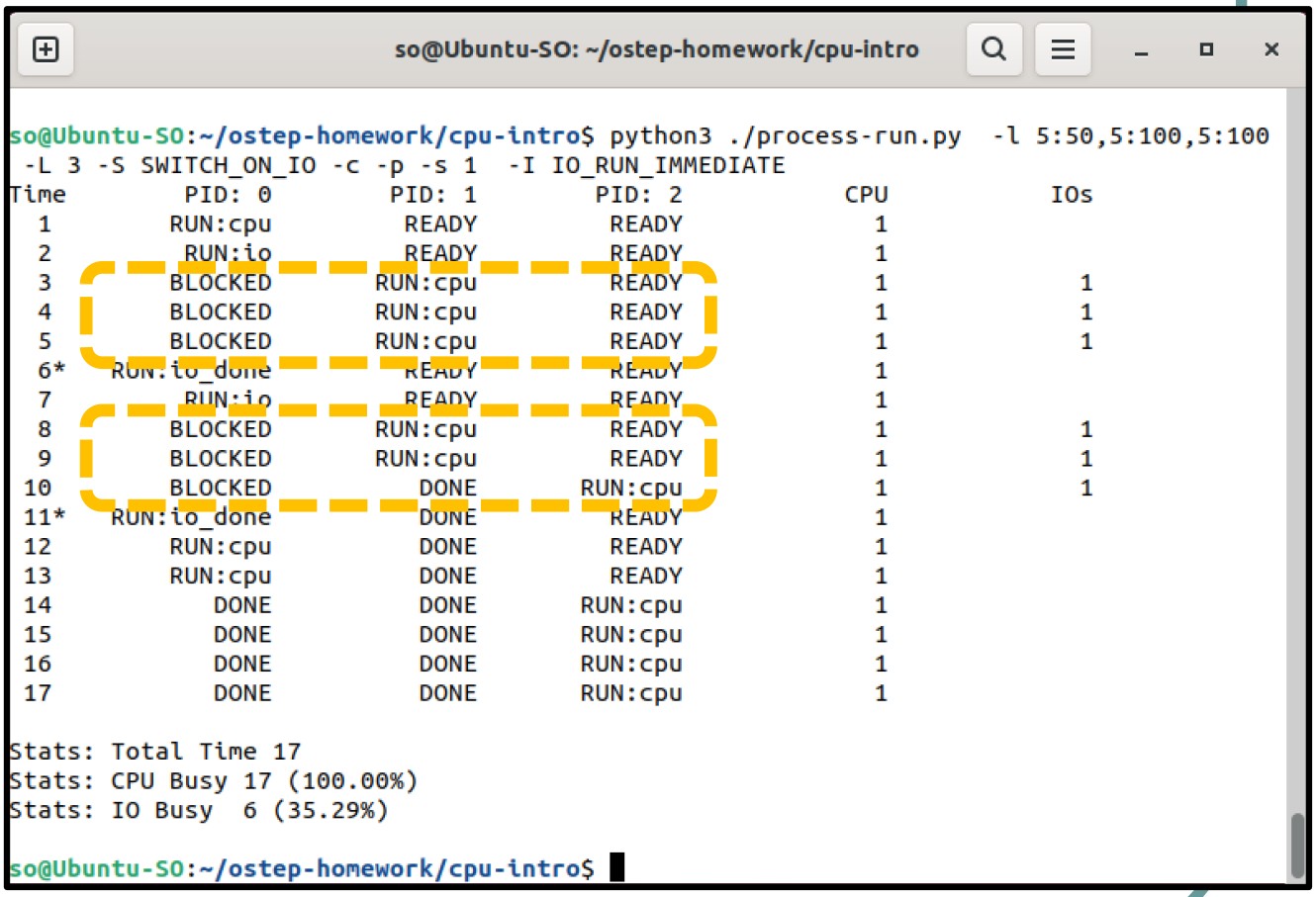
Time	PID: 0
1	RUN:cpu
2	RUN:io
3	BLOCKED
4	BLOCKED
5	BLOCKED
6*	RUN:io_done
7	RUN:io
8	BLOCKED
9	BLOCKED
10	BLOCKED
11*	RUN:io_done
12	RUN:cpu
13	RUN:cpu
14	DONE
15	DONE
16	DONE
17	DONE

IO_RUN_IMMEDIATE

Esempio di simulazione

Mentre il processo 0 è in attesa, la CPU è usata dagli altri (maggiore utilizzo delle risorse).

Inoltre, favorisce un minor tempo di risposta (l'input di I/O è elaborato prima).



Quiz

1. Un processo si dice "BLOCCATO" quando:

- ◯ Vuole utilizzare la CPU, ma c'è già un altro processo che la usa
- ◯ Ha richiesto una operazione di I/O, ma è in attesa che si completi
- ◯ È stato spostato in memoria secondaria

2. È utile dare la priorità ai processi che fanno operazioni di I/O

- ◯ Vero
- ◯ Falso

https://forms.office.com/r/vcFmPa5KEq