

# 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](http://www.ostep.org), Cap. 4, Cap. 6 (sez. 6.3)

## ● Testi di Approfondimento

- A. Silberschatz, 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**

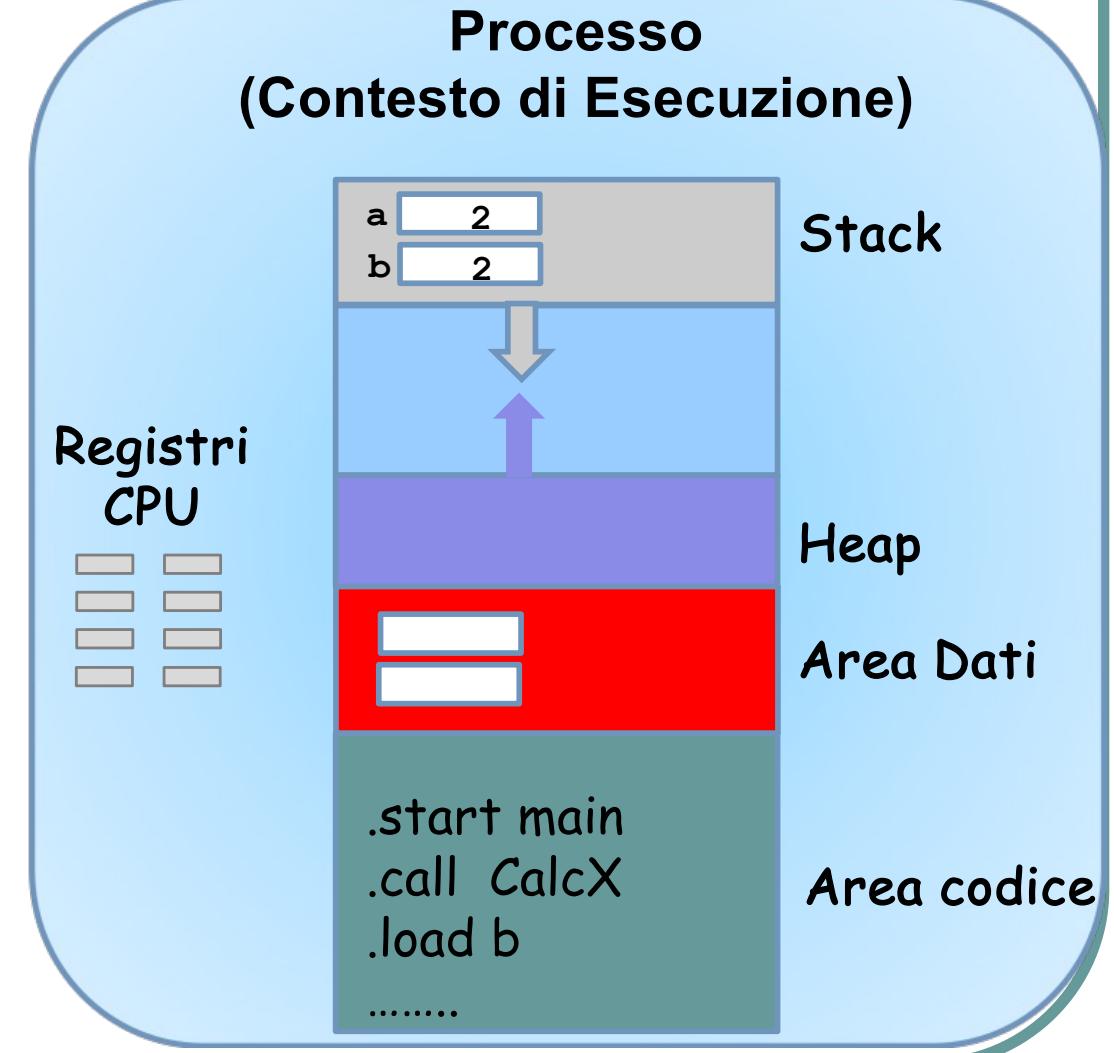
- ...è l'**unità base di esecuzione del SO**, che identifica le attività dell'elaboratore relative ad una specifica esecuzione di un programma
- entità **dinamica** (*programma + contesto di esecuzione*)

- **Processi differenti possono eseguire più istanze di uno stesso programma**

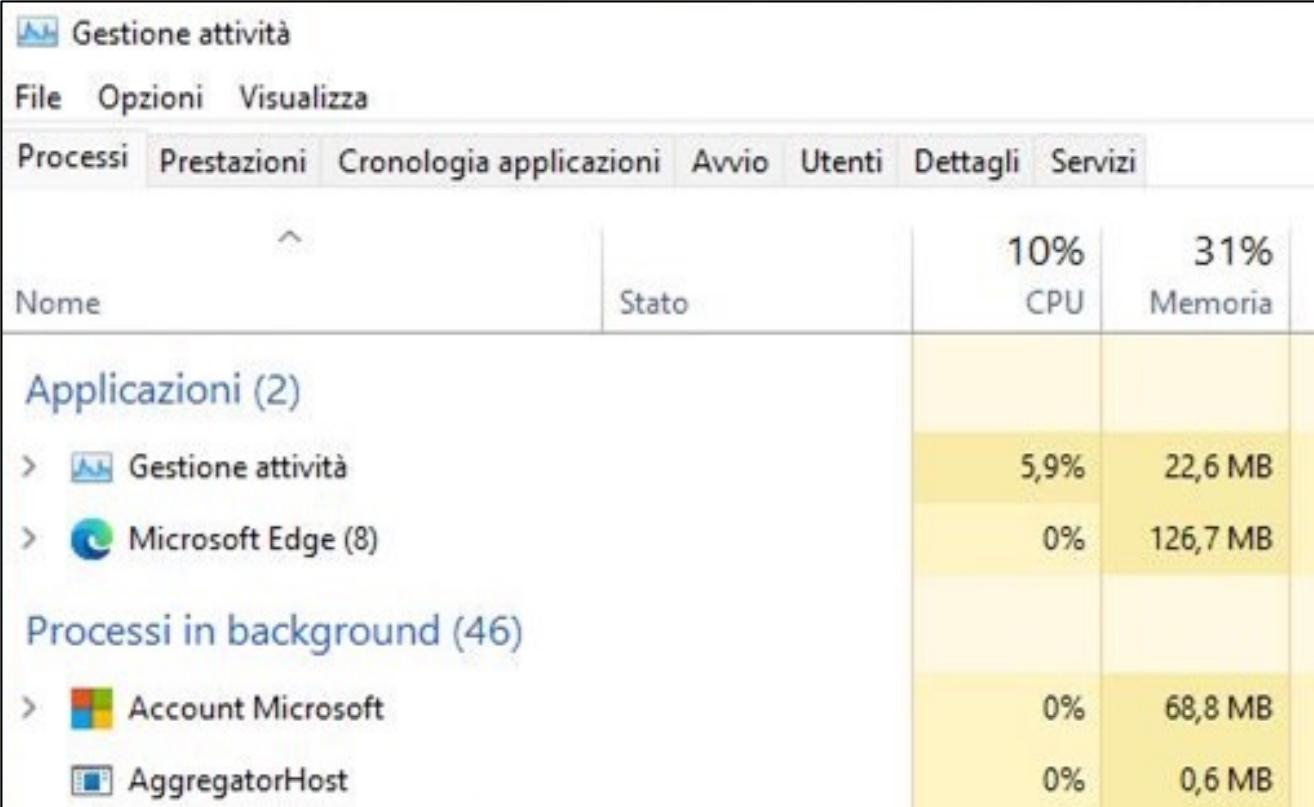
# Dal Programma al Processo

## Programma

```
void CalcX (int b) {  
    ...  
    if(b == 1) {...}  
    ...  
}  
  
int main() {  
    int a = 2;  
    CalcX(a);  
}
```



# Dal Programma al Processo



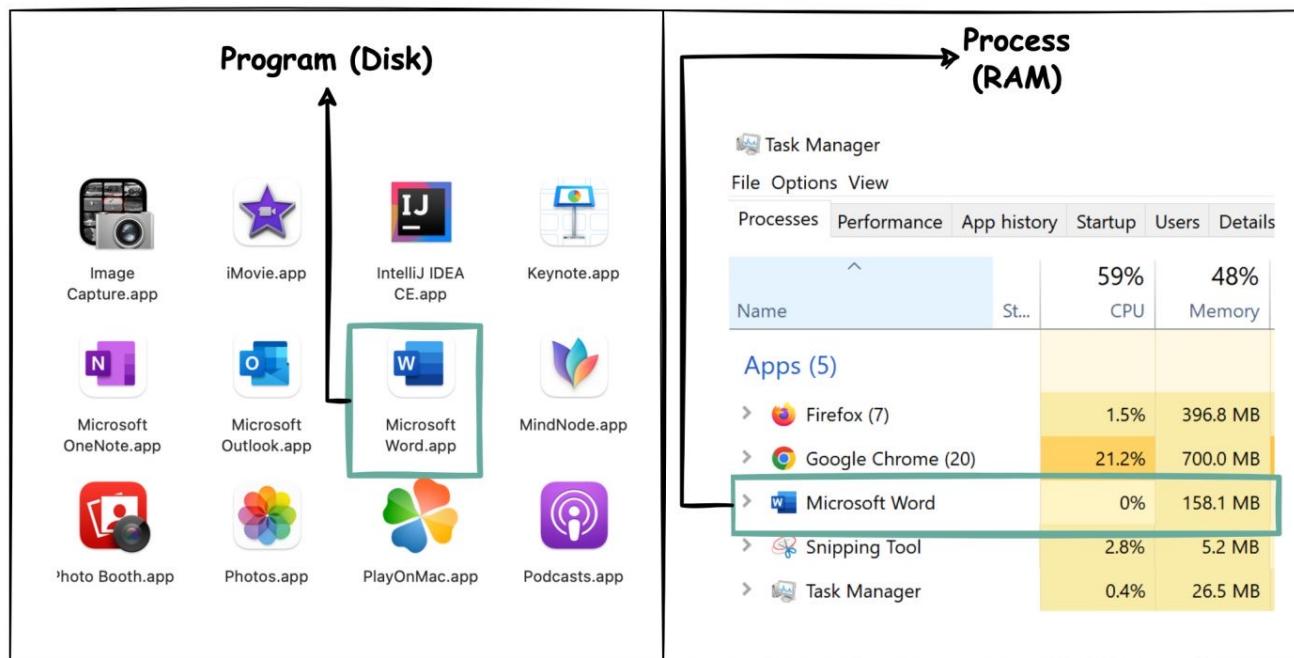
The screenshot shows the Windows Task Manager interface. The title bar reads "Gestione attività". The menu bar includes "File", "Opzioni", and "Visualizza". The top navigation bar has tabs: "Processi" (selected), "Prestazioni", "Cronologia applicazioni", "Avvio", "Utenti", "Dettagli", and "Servizi". Below the tabs is a table with columns: "Nome", "Stato", "CPU", and "Memoria". The table lists two sections: "Applicazioni (2)" and "Processi in background (46)". Under "Applicazioni (2)", there are entries for "Gestione attività" and "Microsoft Edge (8)". Under "Processi in background (46)", there are entries for "Account Microsoft" and "AggregatorHost".

Nome	Stato	CPU	Memoria
Applicazioni (2)			
> Gestione attività		5,9%	22,6 MB
> Microsoft Edge (8)		0%	126,7 MB
Processi in background (46)			
> Account Microsoft		0%	68,8 MB
> AggregatorHost		0%	0,6 MB

# Dal Programma al Processo

Program vs Process vs Thread

 blog.bytebytogo.com

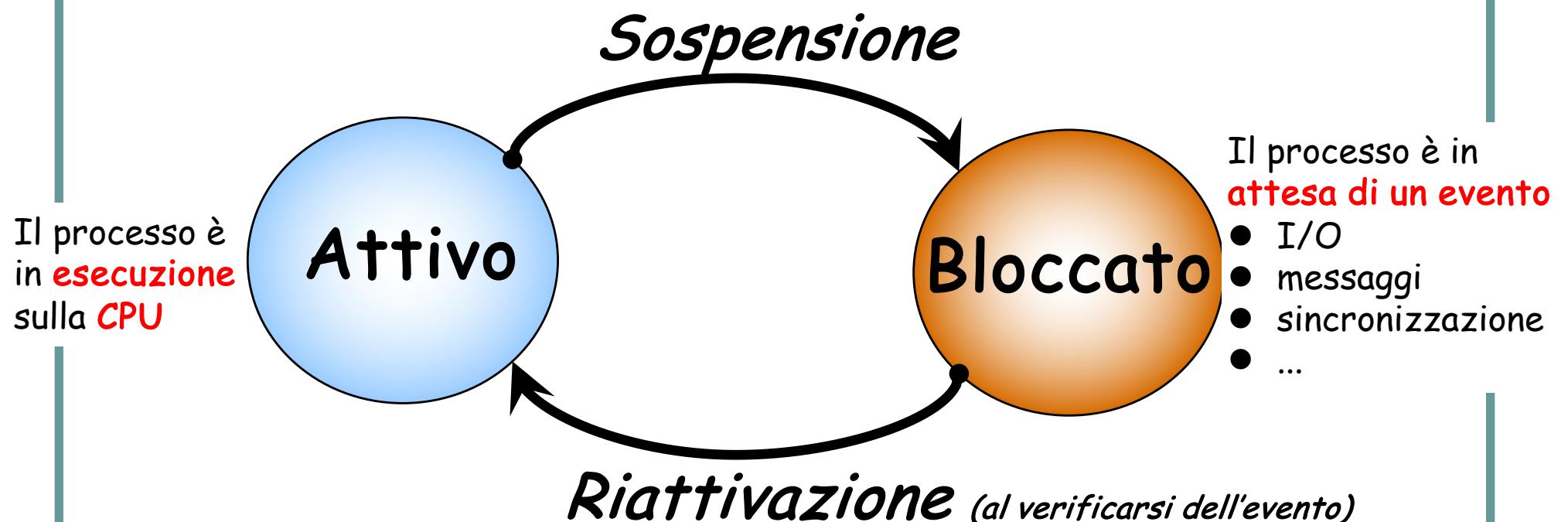


# Stati di un processo

- 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

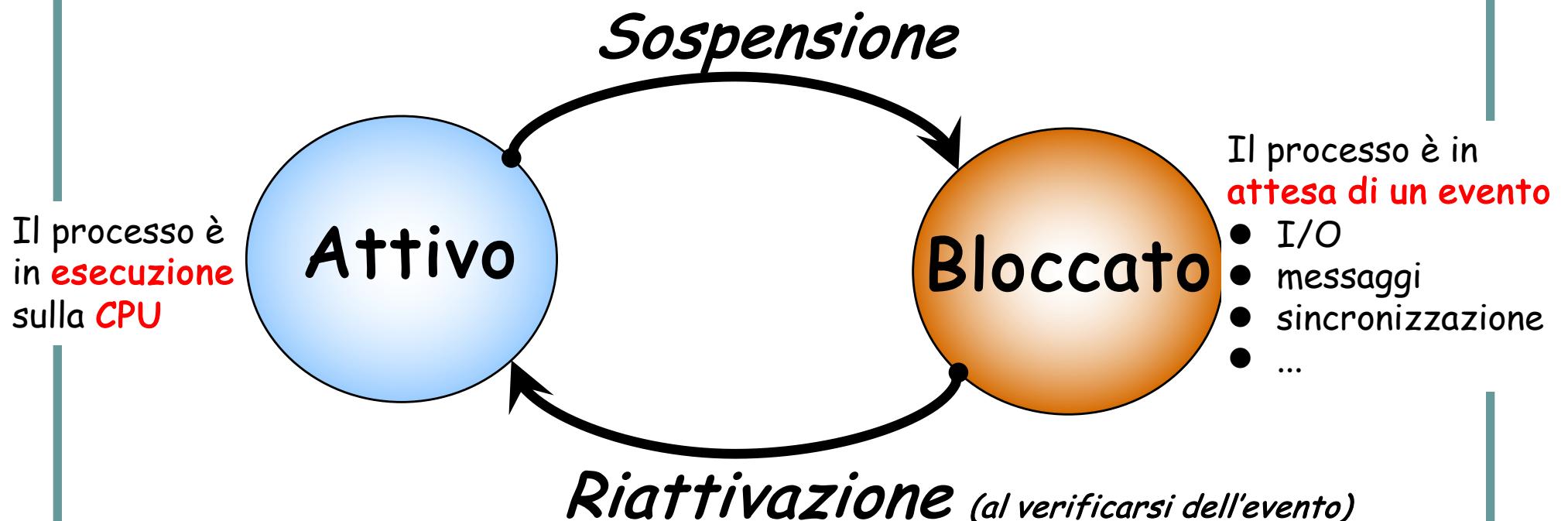
# Modello a due stati

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



# 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

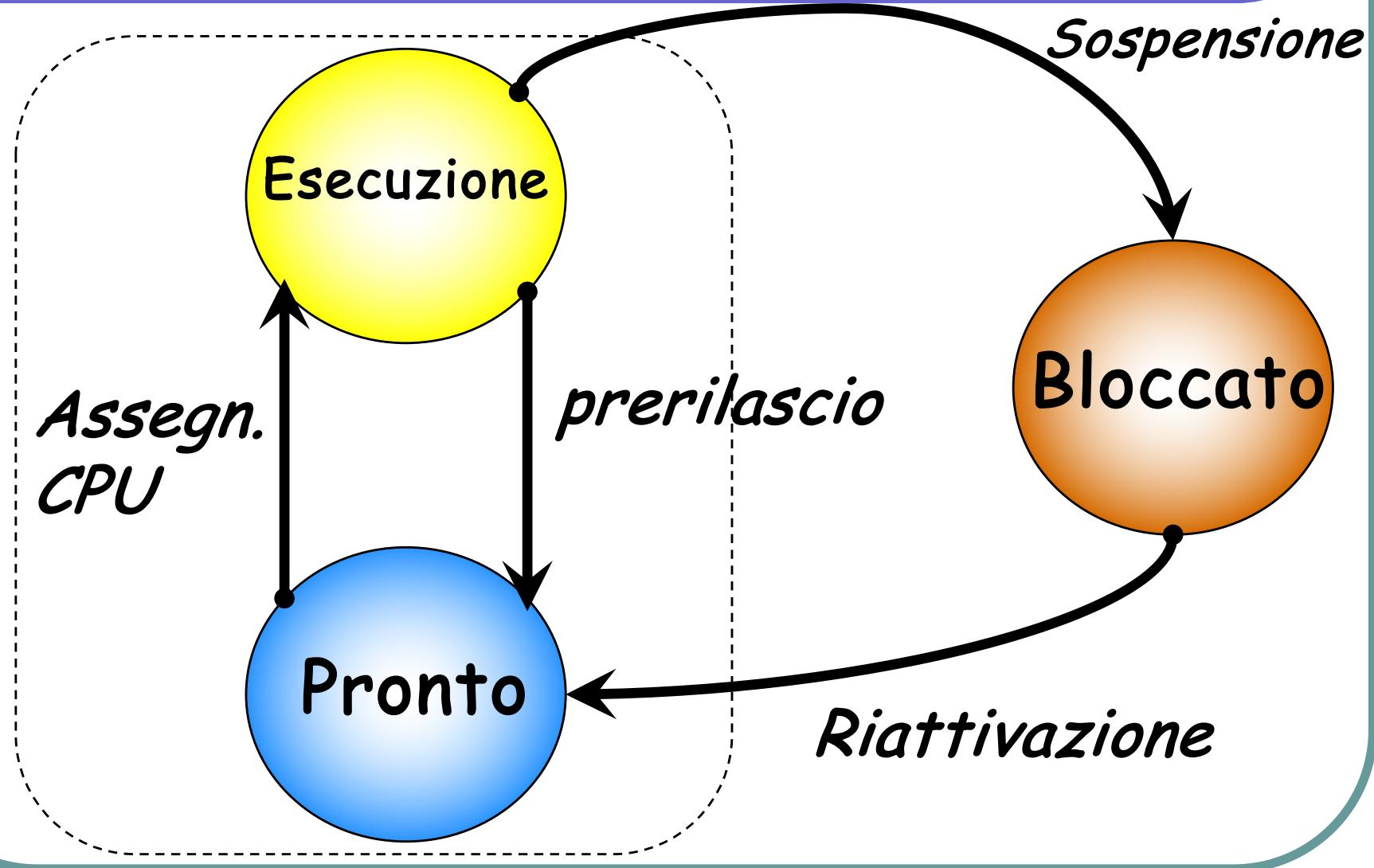
numero di unità di  
elaborazione



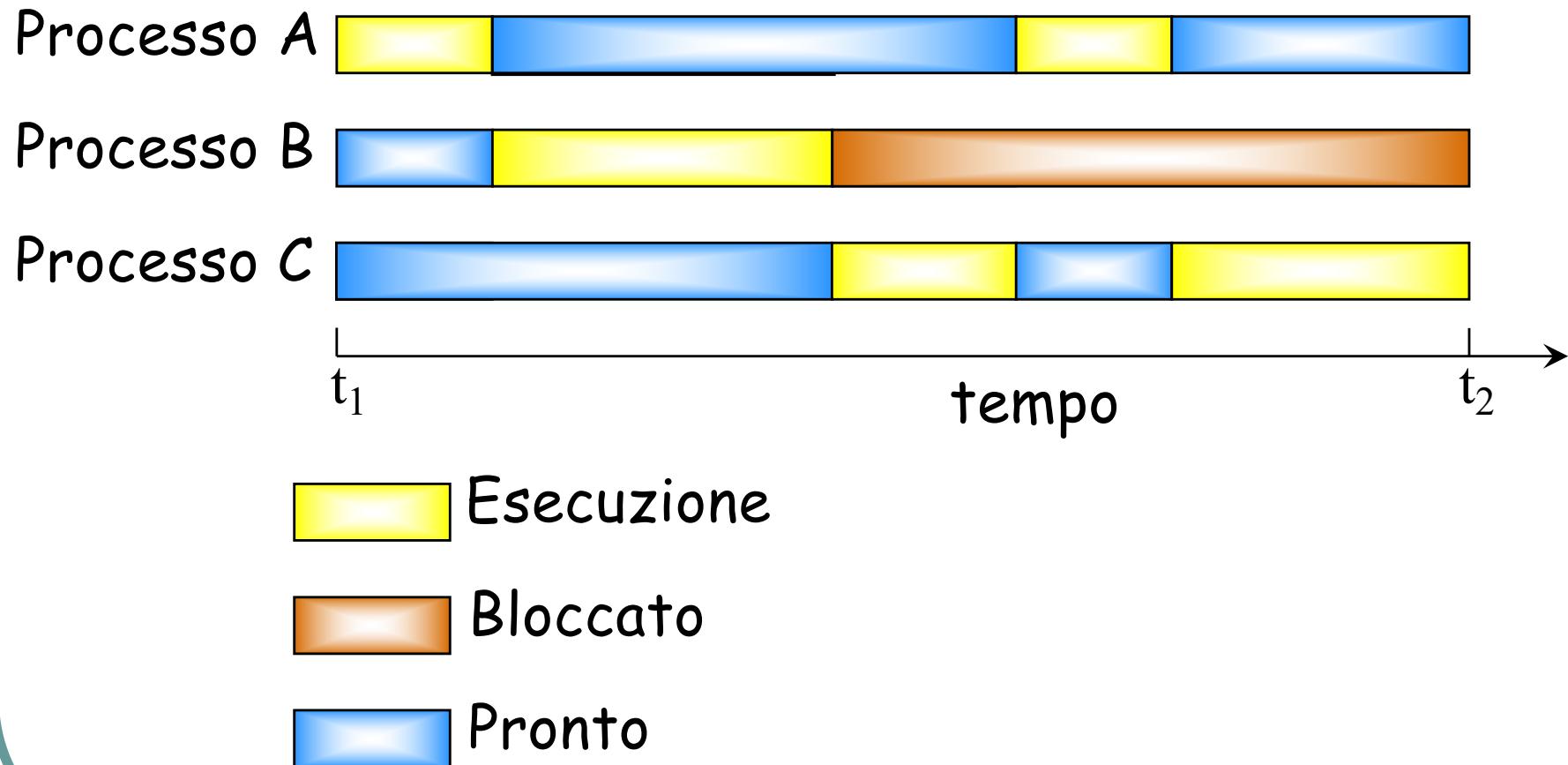
numero di processi  
nel sistema

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

# Modello a tre stati



# Un primo esempio...

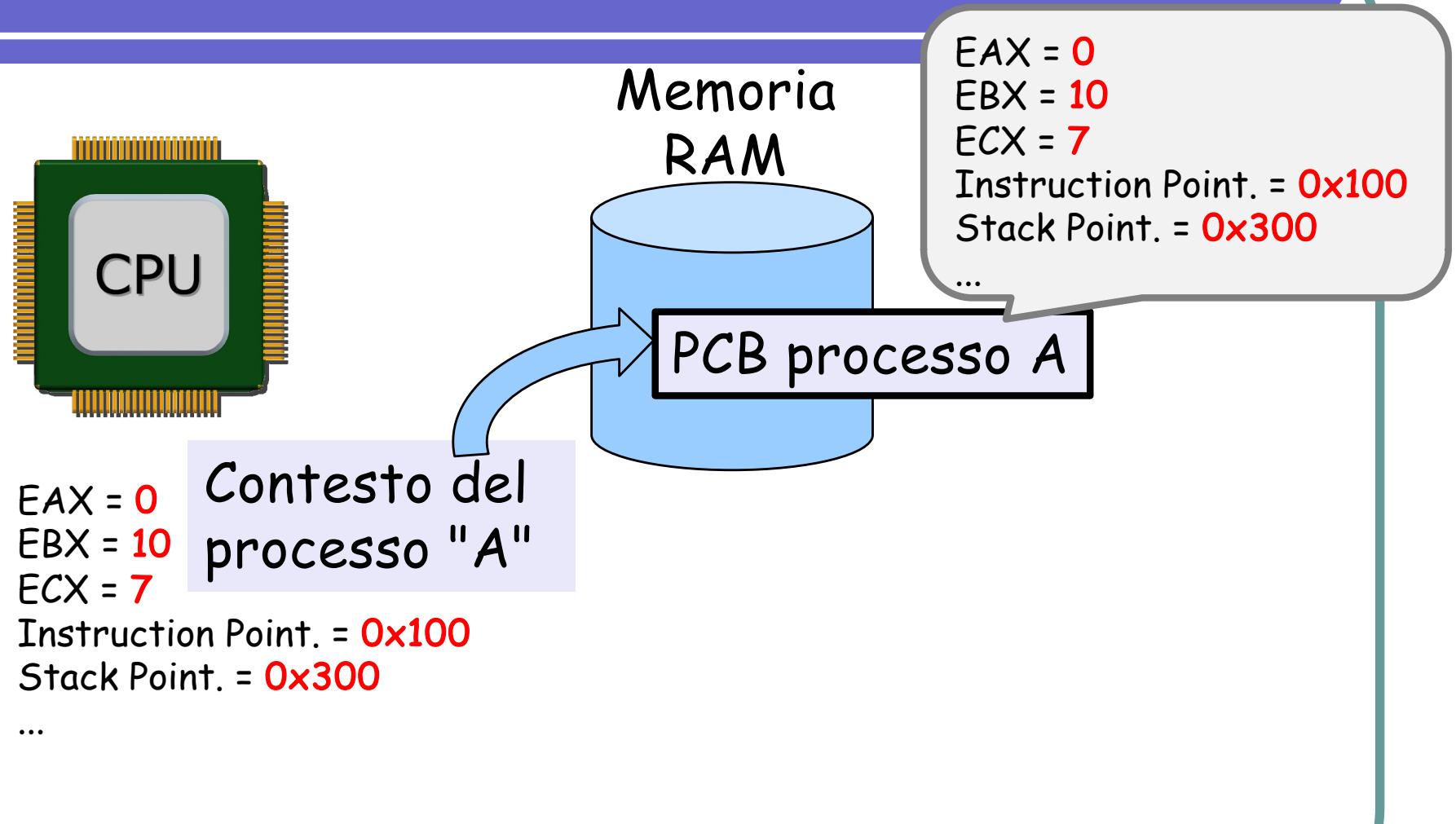


# Il cambiamento di contesto

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

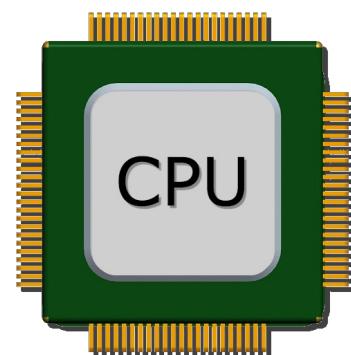
- Il contesto di un processo include le informazioni contenute nei **registri del processore**
  - Program Counter
  - Stack Pointer
  - Registri general-purpose
  - Registri di gestione della memoria
  - ...

# Il cambiamento di contesto



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

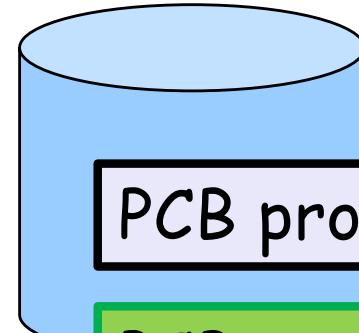
# Il cambiamento di contesto



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

Contesto del processo "B"

Memoria RAM



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

PCB processo B

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

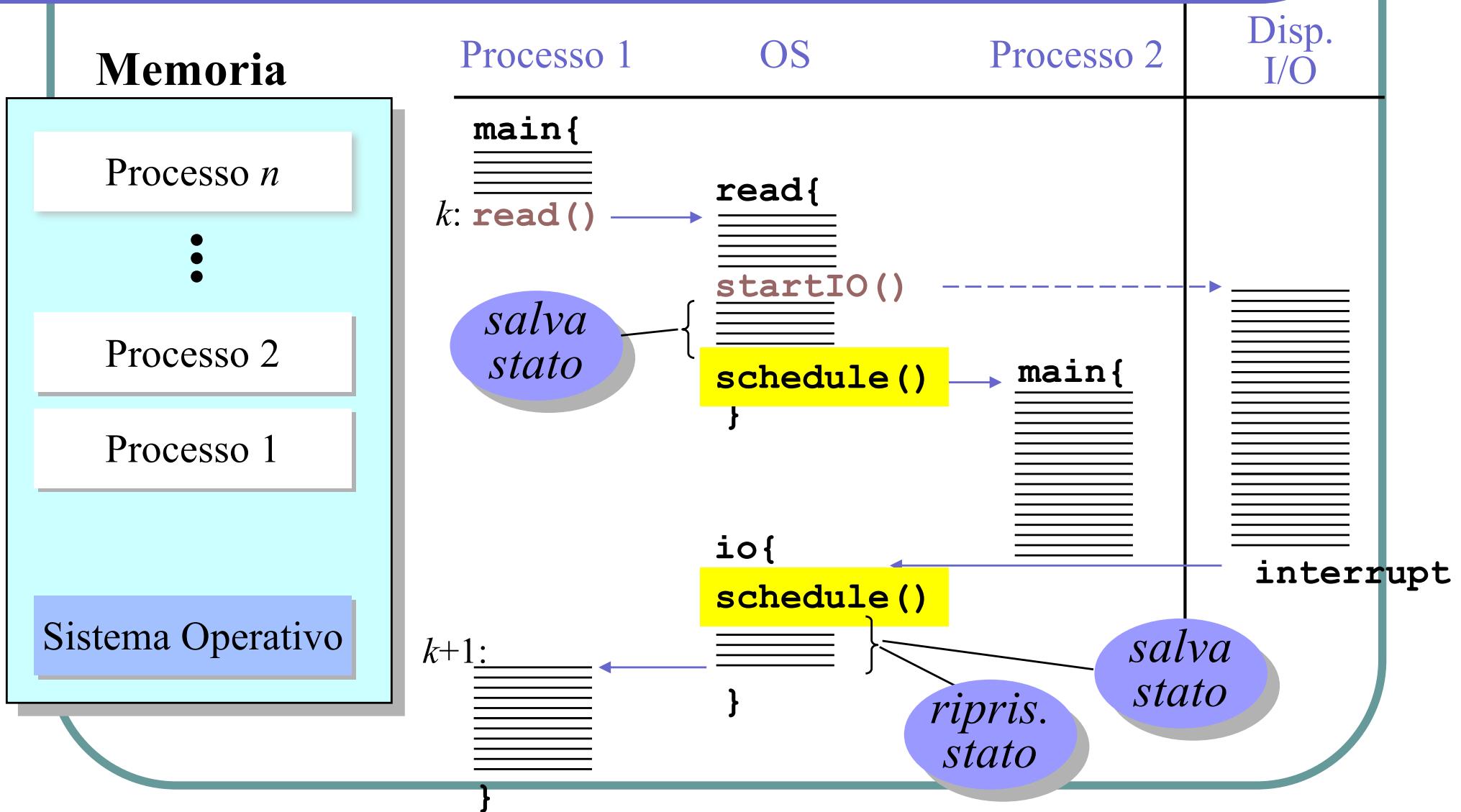
Sceglie il prossimo processo (tra quelli "pronti") da porre in esecuzione

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”

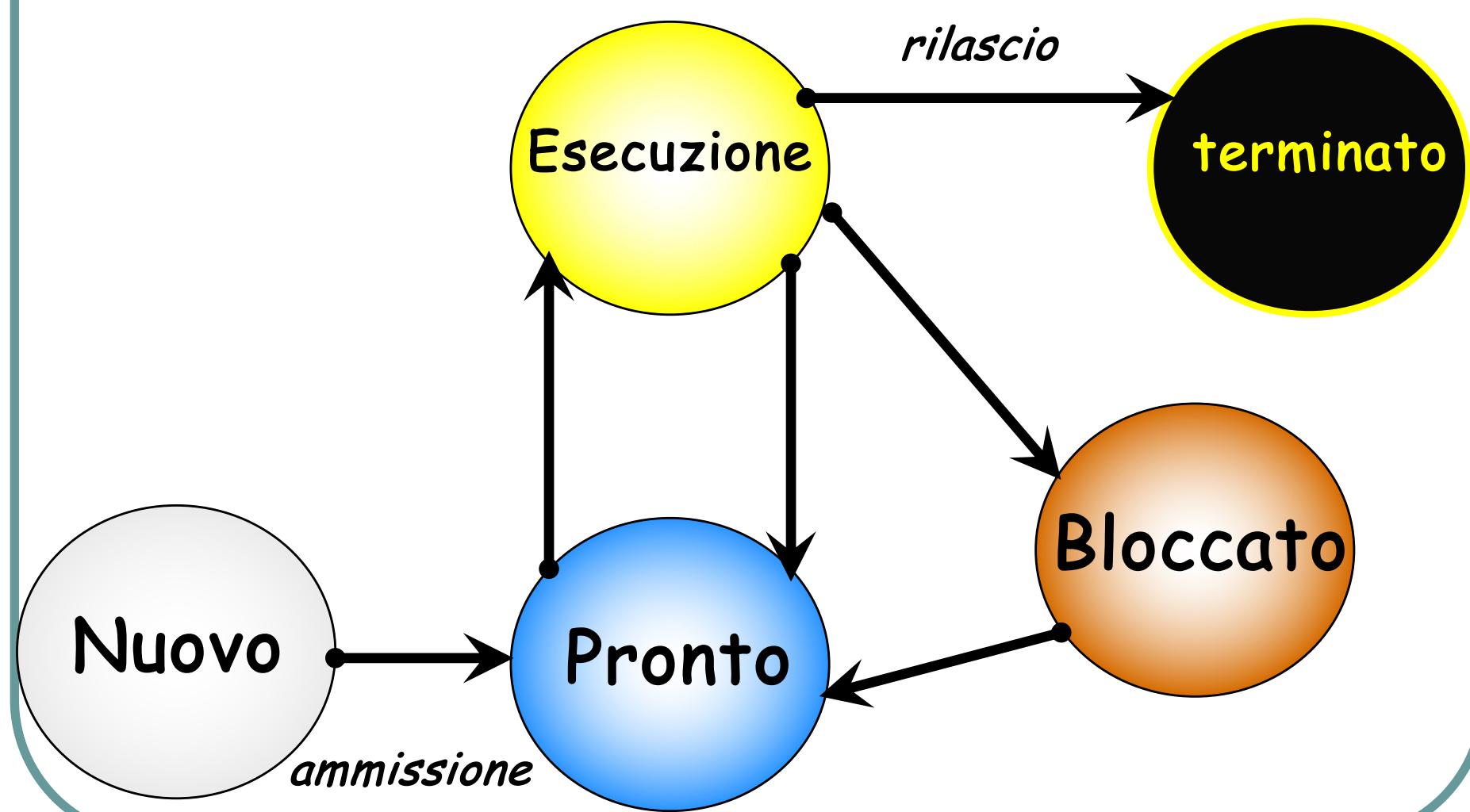
Lo stato “**nuovo**” corrisponde alla creazione di un nuovo processo.

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

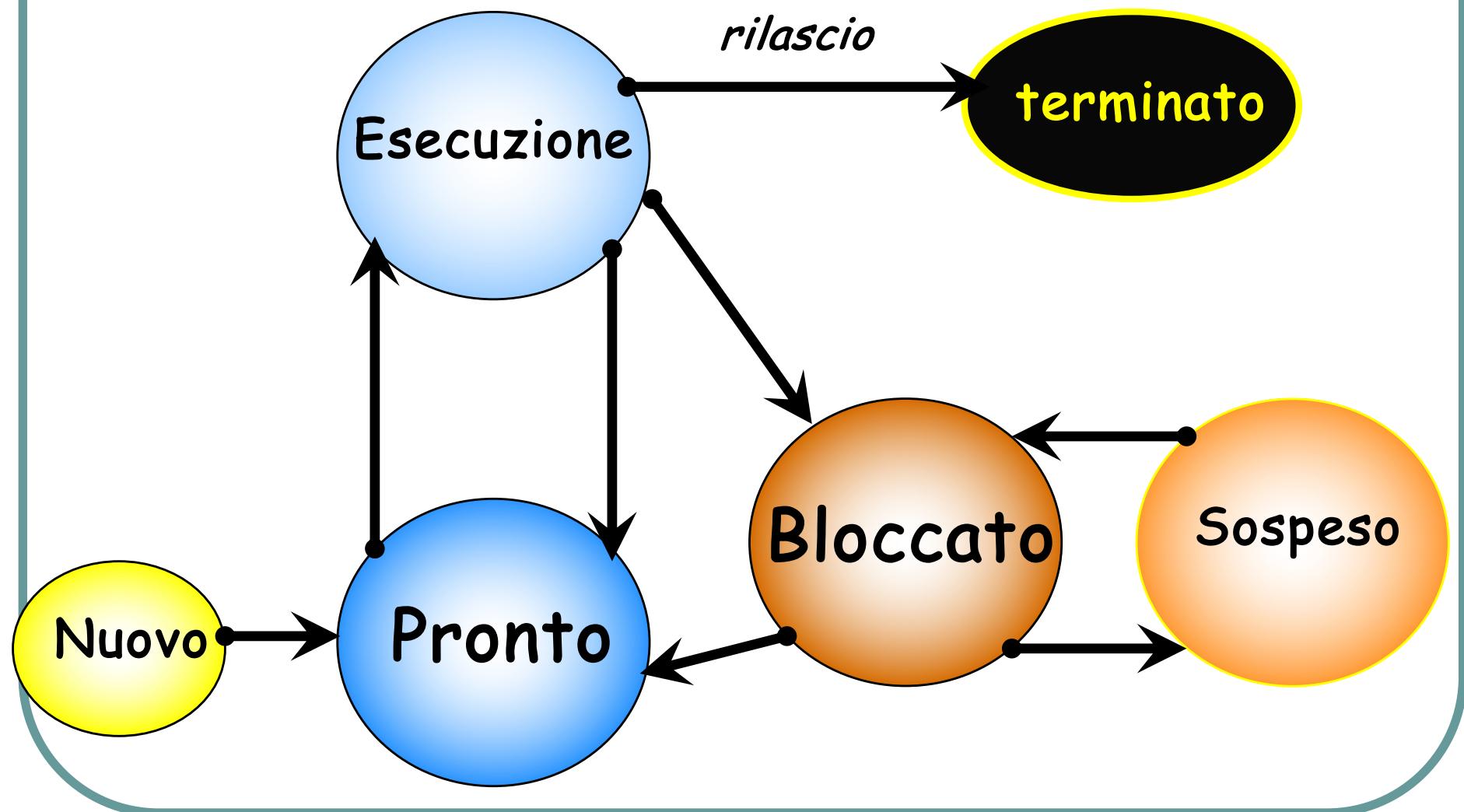
memoria principale  
(RAM)



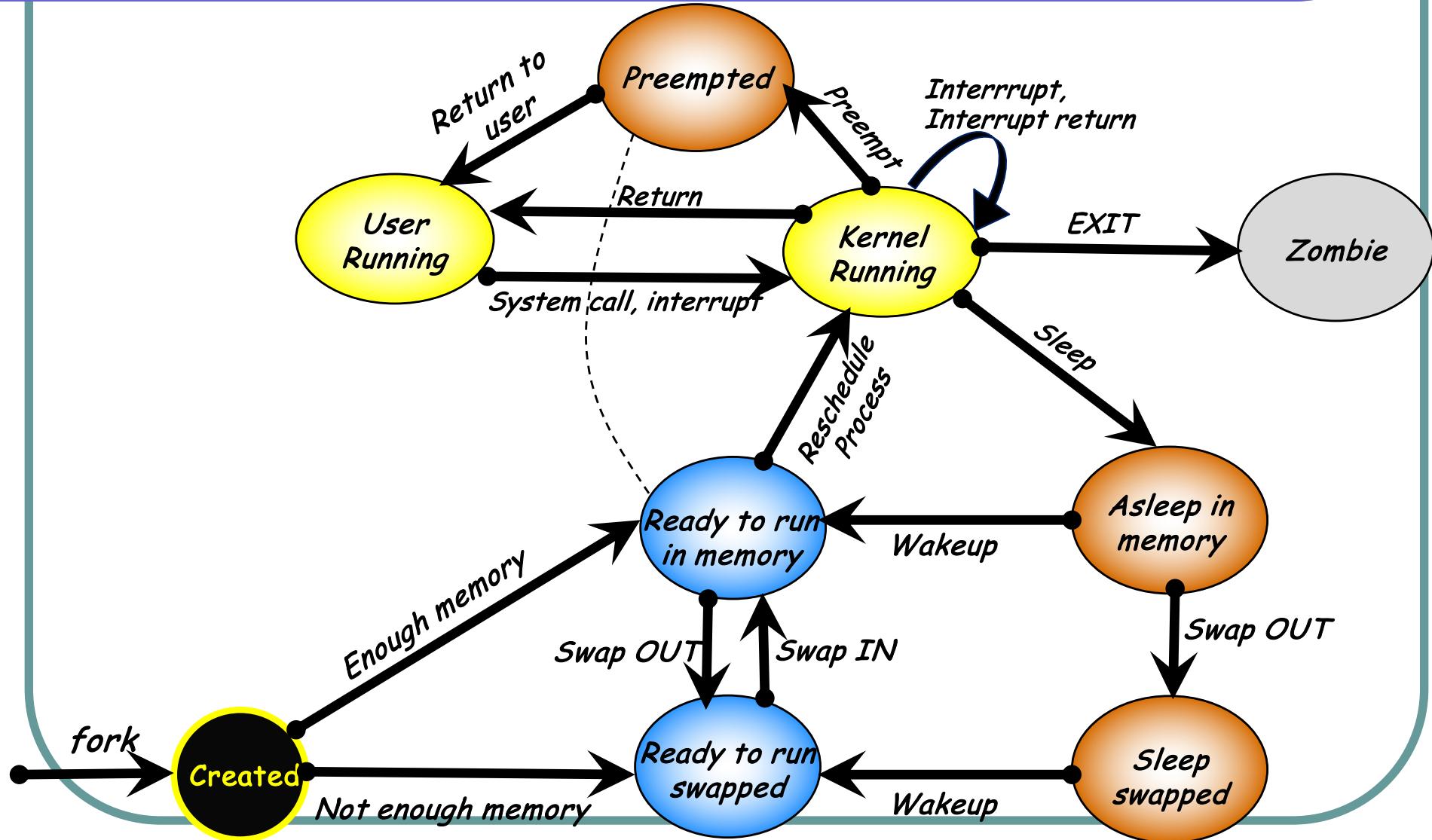
memoria secondaria  
(dischi)

...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

```
so@so-vbox: ~
top - 13:39:41 up 2:16, 1 user, load average: 0,03, 0,03, 0,00
Tasks: 256 total, 1 running, 255 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1,3 us, 0,3 sy, 0,0 ni, 98,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 1987,7 total, 139,5 free, 801,4 used, 1046,9 buff/cache
MiB Swap: 1873,4 total, 1873,4 free, 0,0 used. 1017,2 avail Mem

 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM TIME+ COMMAND
 1149 so        20   0 3625080 304004 118296 S  1,7 14,9 0:20.99 gnome-shell
  901 so        20   0 229372 60880 37780 S  0,3  3,0 0:04.74 Xorg
 1371 so        20   0 292696 42436 31040 S  0,3  2,1 0:12.24 vmtoolsd
 1939 so        20   0 816712 52104 39404 S  0,3  2,6 0:03.13 gnome-terminal-
 3434 so        20   0 12000 3996 3212 R  0,3  0,2 0:00.14 top
  1 root       20   0 102132 11604 8400 S  0,0  0,6 0:03.03 systemd
  2 root       20   0      0     0     0 S  0,0  0,0 0:00.01 kthreadd
  3 root      0 -20      0     0     0 I  0,0  0,0 0:00.00 rcu_gp
  4 root      0 -20      0     0     0 I  0,0  0,0 0:00.00 rcu_par_gp
  6 root      0 -20      0     0     0 I  0,0  0,0 0:00.00 kworker/0:0H-kblockd
  7 root       20   0      0     0     0 I  0,0  0,0 0:10.44 kworker/0:1-events
  9 root      0 -20      0     0     0 I  0,0  0,0 0:00.00 mm_percpu_wq
 10 root      20   0      0     0     0 S  0,0  0,0 0:00.24 ksoftirqd/0
 11 root      20   0      0     0     0 I  0,0  0,0 0:00.65 rcu_sched
 12 root      rt  0      0     0     0 S  0,0  0,0 0:00.06 migration/0
 13 root     -51   0      0     0     0 S  0,0  0,0 0:00.00 idle_inject/0
 14 root      20   0      0     0     0 S  0,0  0,0 0:00.00 cpuhp/0
 15 root      20   0      0     0     0 S  0,0  0,0 0:00.00 kdevtmpfs
 16 root      0 -20      0     0     0 I  0,0  0,0 0:00.00 netns
 17 root      20   0      0     0     0 S  0,0  0,0 0:00.00 rcu_tasks_kthre
```

# Il comando top

```
so@so-vbox: ~
top - 13:39:41 up 2:16, 1 user, load average: 0,03, 0,03, 0,00
Tasks: 256 total, 1 running, 255 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1,3 us, 0,3 sy, 0,0 ni, 98,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 1987,7 total, 139,5 free, 801,4 used, 1046,9 buff/cache
MiB Swap: 1873,4 total, 1873,4 free, 0,0 used. 1017,2 avail Mem

 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
 1149 so 20 0 3625080 304004 118296 S 1,7 14,9 0:20.99 qnome-shell
  901 so 20 0 229372 60880 37780 S 0,3 3,0 0:04.74 Xorg
 1371 so 20 0 292696 42436 31040 S 0,3 2,1 0:12.24 vmtoolsd
 1939 so 20 0 816712 52104 39404 S 0,3 2,6 0:03.13 qnome-terminal-
 3434 so 20 0 12000 3996 3212 R 0,3 0,2 0:00.14 top
   1 root 20 0 102132 11604 8400 S 0,0 0,6 0:03.03 systemd
   2 root 20 0 0 0 0 S 0,0 0,0 0:00.00 ksoftirqd/0
   3 root 0 -20 0 0 0 I 0,0 0,0 0:00.00 rcu_sched
   4 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 migration/0
   6 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
   7 root 20 0 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
   9 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
  10 root 20 0 0 0 0 S 0,0 0,0 0:00.24 ksoftirqd/0
  11 root 20 0 0 0 0 I 0,0 0,0 0:00.65 rcu_sched
  12 root rt 0 0 0 0 S 0,0 0,0 0:00.06 migration/0
  13 root -51 0 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
  14 root 20 0 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
  15 root 20 0 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
  16 root 0 -20 0 0 0 T 0,0 0,0 0:00.00 rcu_tasks_kthre
  17 root 20 0 0 0 0 S 0,0 0,0 0:00.00 rcu_tasks_kthre
```

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

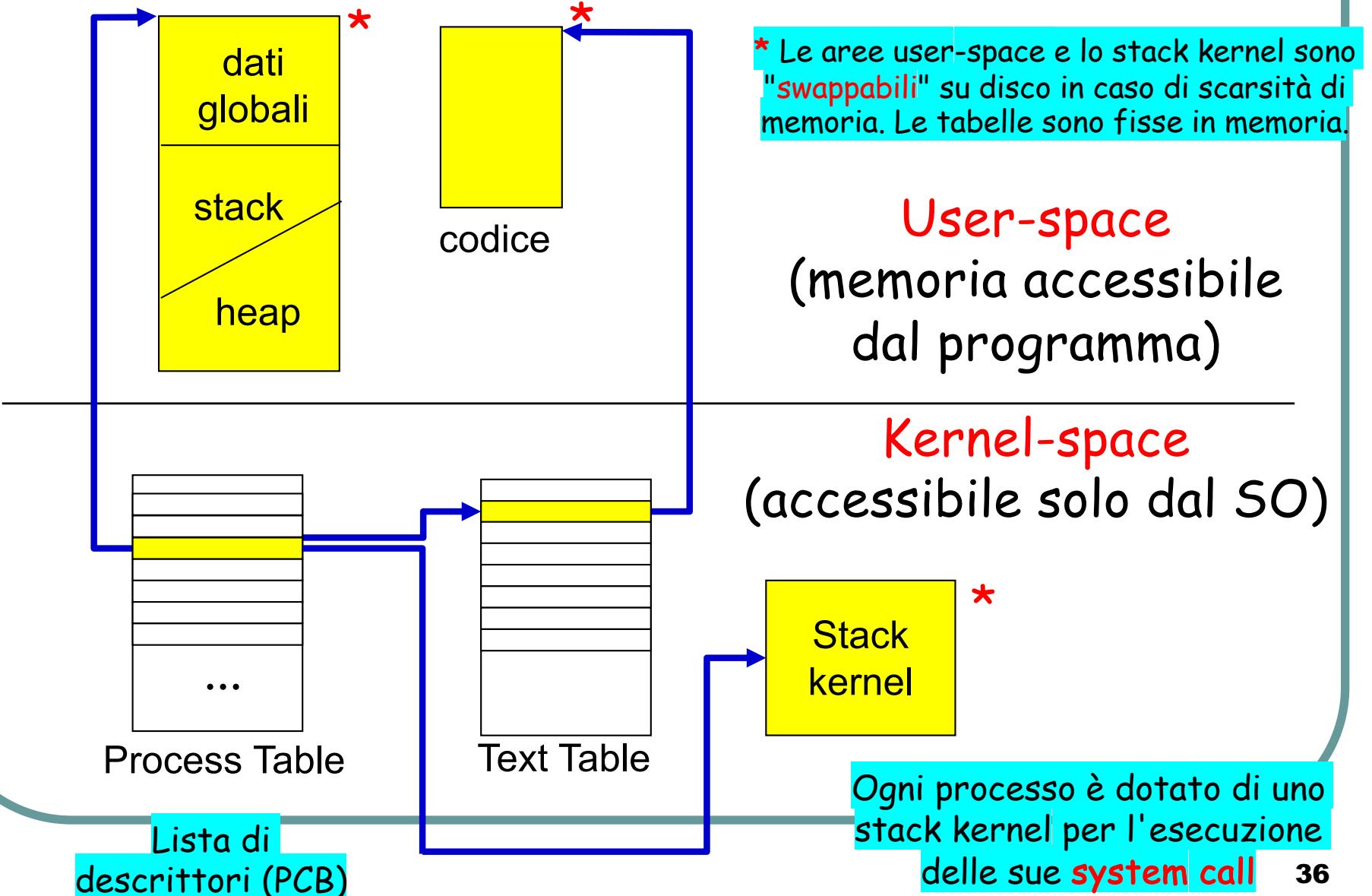
- Process Identifier (PID)
- Stato del processo
- Riferimento a aree dati e stack
- Riferimento indiretto al codice (text-structure)
- PID del processo padre
- Priorità del processo
- Riferimento al prossimo processo in coda
- Puntatore alla U-Area
- ...

**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
- ...

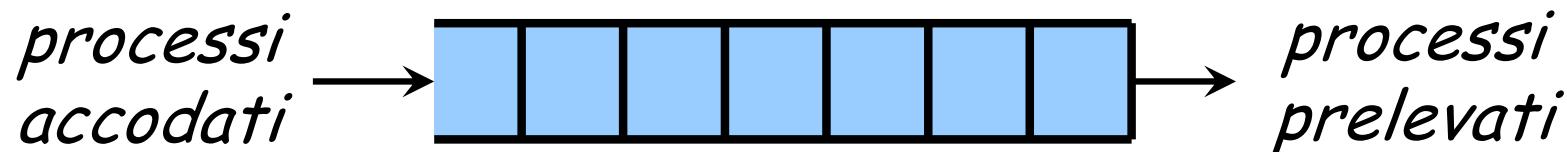
**U-Area**  
(swappabile)

# Immagine di un processo Unix

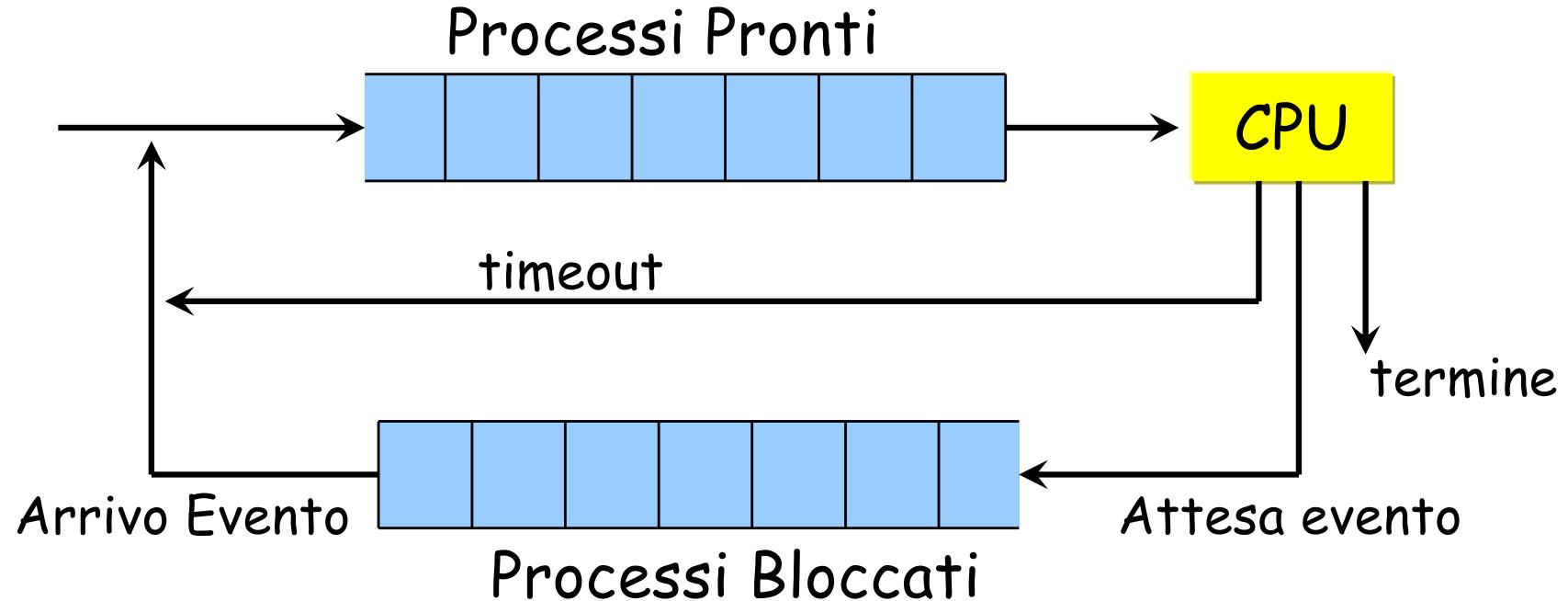


# Code dei processi

- 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

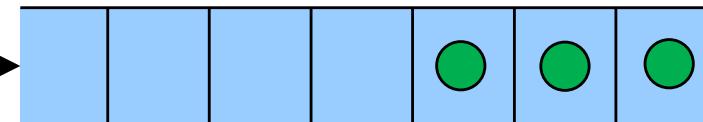


# Code dei processi

Creazione di un nuovo processo

Il prerilascio da parte del SO forza un processo **in esecuzione** nello stato di **pronto**

Processi Pronti



CPU

timeout

EVENTO ESTERNO

Arrivo Evento

Attesa evento

Processi Bloccati

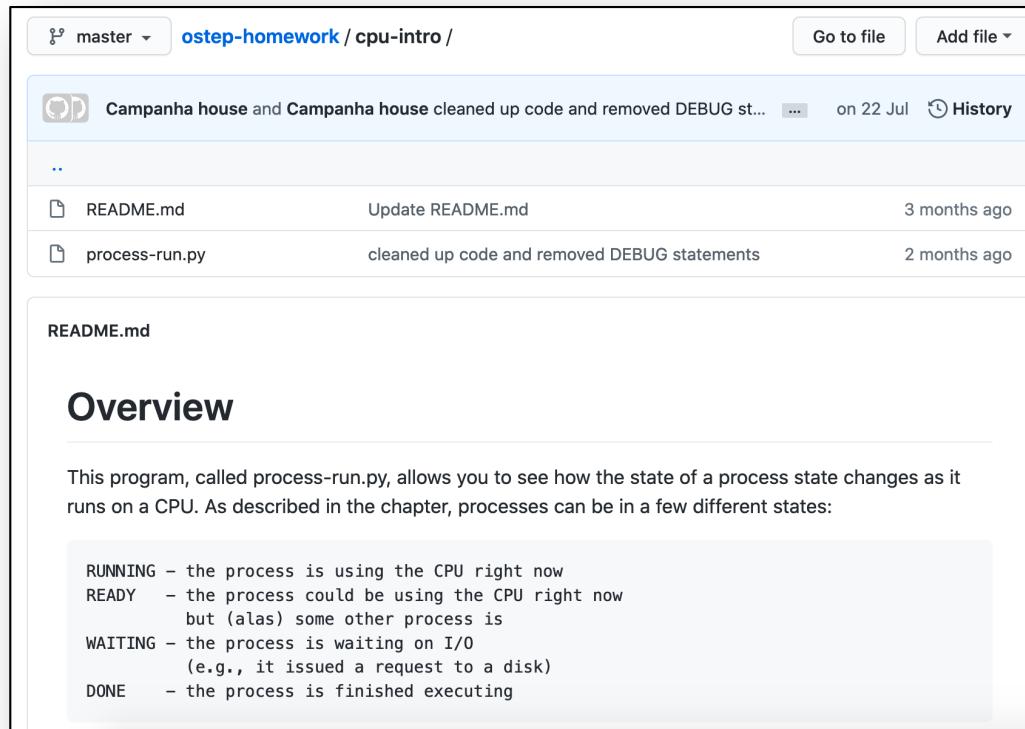
A seguito del verificarsi dell'evento atteso (es. fine lettura), un processo **bloccato** transita nello stato di **pronto**

Le operazioni di I/O (es. chiamata di sistema **read()**) pongono il processo nello stato **bloccato**

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)

# Esempio 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

```
so@Ubuntu-S0:~/ostep-homework/cpu-intro$ python3 ./process-run.py -l 5:50,5:100,5:100  
-L 3 -S SWITCH_ON_IO -c -p -s 1  
Time      PID: 0      PID: 1      PID: 2      CPU      IOs  
 1        RUN:cpu    READY      READY      1  
 2        RUN:io     READY      READY      1  
 3        BLOCKED   RUN:cpu    READY      1      1  
 4        BLOCKED   RUN:cpu    READY      1      1  
 5        BLOCKED   RUN:cpu    READY      1      1  
 6*       READY     RUN:cpu    READY      1  
 7        READY     RUN:cpu    READY      1  
 8        READY     DONE      RUN:cpu    1  
 9        READY     DONE      RUN:cpu    1  
10       READY     DONE      RUN:cpu    1  
11       READY     DONE      RUN:cpu    1  
12       READY     DONE      RUN:cpu    1  
13       RUN:io_done DONE      DONE      1  
14       RUN:io     DONE      DONE      1  
15       BLOCKED   DONE      DONE      1  
16       BLOCKED   DONE      DONE      1  
17       BLOCKED   DONE      DONE      1  
18*       RUN:io_done DONE      DONE      1  
19       RUN:cpu    DONE      DONE      1  
20       RUN:cpu    DONE      DONE      1  
  
Stats: Total Time 20  
Stats: CPU Busy 17 (85.00%)  
Stats: IO Busy  6 (30.00%)  
so@Ubuntu-S0:~/ostep-homework/cpu-intro$
```

# 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

```
so@Ubuntu-S0:~/ostep-homework/cpu-intro$ python3 ./process-run.py -l 5:50,5:100,5:100
-L 3 -S SWITCH_ON_IO -c -p -s 1
Time      PID: 0      PID: 1      PID: 2      CPU      IOs
1         RUN:cpu    READY     READY     1
2         RUN:io     READY     READY     1
3         BLOCKED   RUN:cpu    READY     1
4         BLOCKED   RUN:cpu    READY     1
5         BLOCKED   RUN:cpu    READY     1
6*        READY     RUN:cpu    READY     1
7         READY     RUN:cpu    READY     1
8         READY     DONE      RUN:cpu   1
9         READY     DONE      RUN:cpu   1
10        READY    DONE      RUN:cpu   1
11        READY    DONE      RUN:cpu   1
12        READY    DONE      RUN:cpu   1
13        RUN:io_done DONE      DONE      1
14        RUN:io    DONE      DONE      1
15        BLOCKED   DONE      DONE      1
16        BLOCKED   DONE      DONE      1
17        BLOCKED   DONE      DONE      1
18*        RUN:io_done DONE      DONE      1
19        RUN:cpu   DONE      DONE      1
20        RUN:cpu   DONE      DONE      1

Stats: Total Time 20
Stats: CPU Busy 17 (85.00%)
Stats: IO Busy 6 (30.00%)

so@Ubuntu-S0:~/ostep-homework/cpu-intro$
```

CPU inutilizzata

# 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).

```
so@Ubuntu-SO:~/ostep-homework/cpu-intro$ python3 ./process-run.py -l 5:50,5:100,5:100 -L 3 -S SWITCH_ON_IO -c -p -s 1 -I IO_RUN_IMMEDIATE
Time      PID: 0      PID: 1      PID: 2      CPU      IOs
1        RUN:cpu    READY    READY      1
2        RUN:io    READY    READY      1
3        BLOCKED   RUN:cpu    READY      1
4        BLOCKED   RUN:cpu    READY      1
5        BLOCKED   RUN:cpu    READY      1
6*      RUN:io_done READY    READY      1
7        RUN:io    READY    READY      1
8        BLOCKED   RUN:cpu    READY      1
9        BLOCKED   RUN:cpu    READY      1
10       BLOCKED   DONE     RUN:cpu    1
11*      RUN:io_done DONE     READY      1
12       RUN:cpu    DONE     READY      1
13       RUN:cpu    DONE     READY      1
14       DONE      DONE     RUN:cpu    1
15       DONE      DONE     RUN:cpu    1
16       DONE      DONE     RUN:cpu    1
17       DONE      DONE     RUN:cpu    1

Stats: Total Time 17
Stats: CPU Busy 17 (100.00%)
Stats: IO Busy  6 (35.29%)
```

# 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>

