

# Il sistema operativo

## Fondamenti di Informatica A

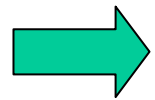
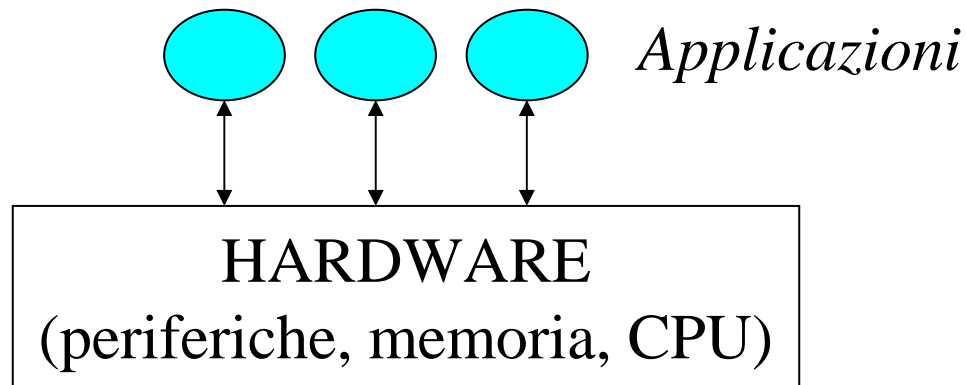
*Percorso di Preparazione agli Studi di Ingegneria*

Università degli Studi di Brescia

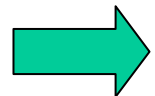
*Docente: Massimiliano Giacomini*

# Cos'è un Sistema Operativo?

Per capirlo, immaginiamo inizialmente che non ci sia:

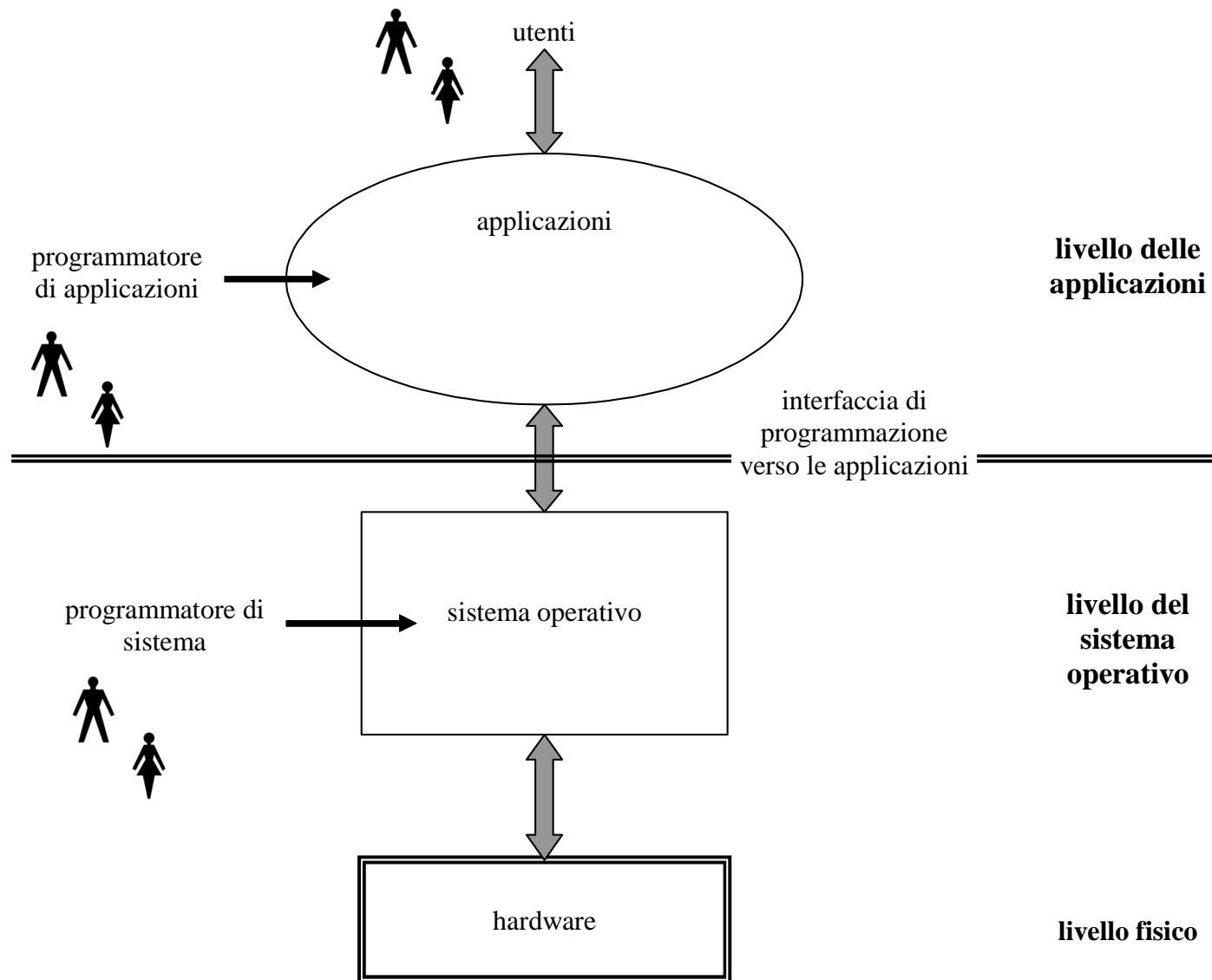


- necessità per il programmatore di conoscere le specificità dell'hardware e di occuparsi della sua gestione  
⇒ aumenta difficoltà nella scrittura dei programmi
- dipendenza applicazioni/hardware



Introduciamo la suddivisione sistema operativo/applicazioni

# Organizzazione a livelli di un sistema di elaborazione



In sostanza, un sistema operativo:

1. E' una collezione di programmi (detti anche software di base) finalizzati a rendere utilizzabile l'intera architettura di elaborazione
2. Fornisce una interfaccia (API: Application Programming Interface) al *software applicativo* usato dall'utente, che può invocare il sistema operativo attraverso *chiamate di sistema*
3. Comprende programmi che gestiscono le risorse fisiche del sistema informatico  
(e quindi risolvono i problemi precedentemente visti)

Più in generale, un sistema operativo offre la visione di una *macchina astratta (o virtuale)* più potente e più semplice da utilizzare di quella fisicamente disponibile (macchina fisica).

## Il concetto di macchina virtuale

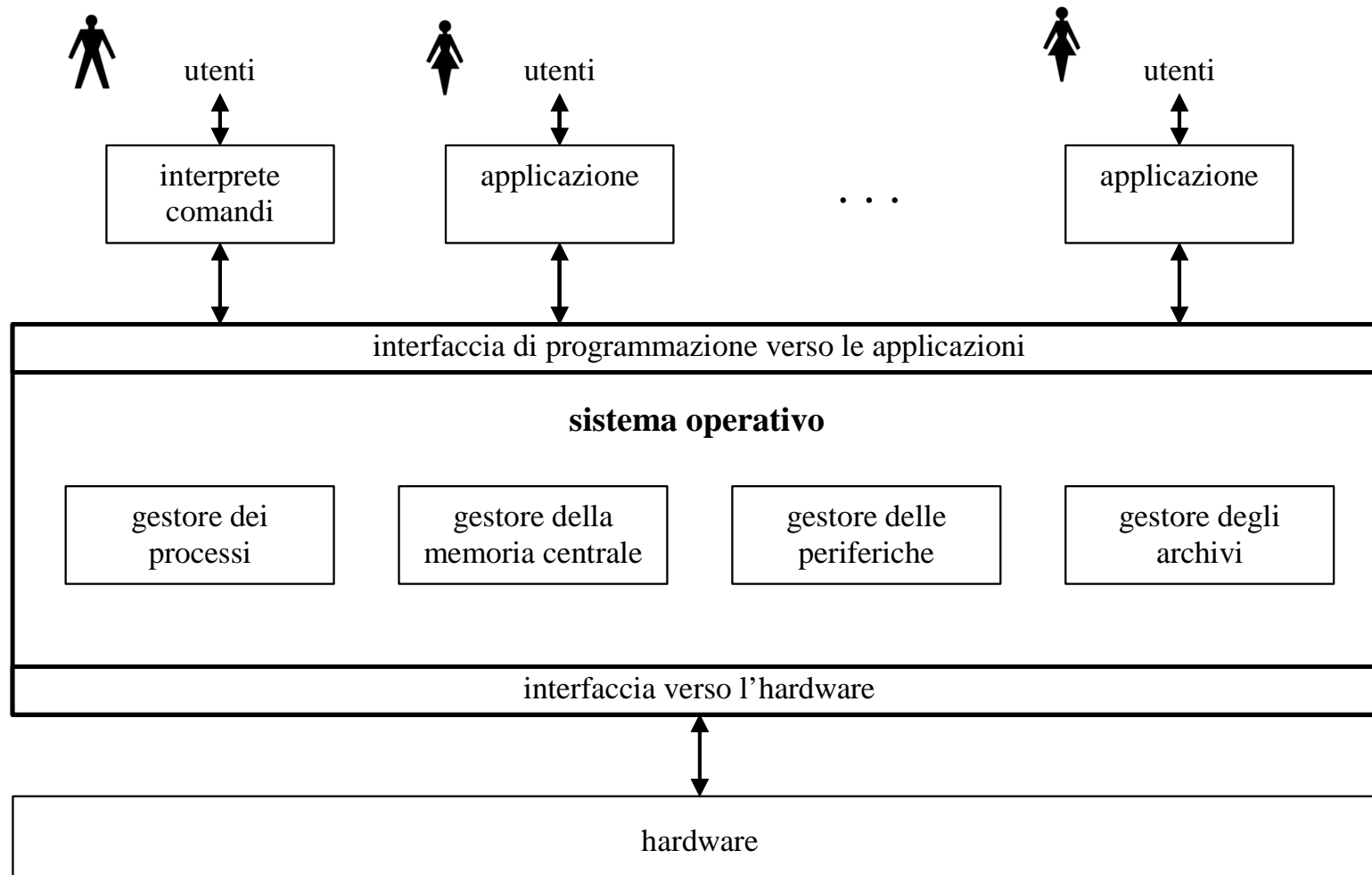
- Interfaccia:  
attraverso di essa gli utenti della macchina virtuale interagiscono con la macchina stessa, utilizzando un insieme di comandi
- Corpo:  
non visibile all'esterno, esegue i comandi ricevuti utilizzando a sua volta i servizi dell'eventuale macchina sottostante

Cfr. lucido 3...

# Funzioni di un Sistema Operativo

- Virtualizzazione dell'hardware (mascherandone i dettagli)
  - periferiche
  - CPU
  - memoria centrale
- Gestione ottimale delle risorse (efficienza)
- Gestione della convivenza (più programmi in esecuzione e più utenti):
  - idealmente, ciascuno opera come se avesse a disposizione l'intero sistema;
  - d'altra parte, gestione delle interazioni tra programmi in esecuzione (es: output utilizzato come input da un altro)
- Gestione della sicurezza (riservatezza e integrità)

# Organizzazione di un Sistema Operativo



- Sistema di **gestione dei processi**
  - Definisce quali programmi sono da eseguire e quali compiti sono da assegnare alla CPU
- Sistema di **gestione della memoria**
  - Controlla l'allocazione della memoria centrale assegnata ai diversi programmi che possono essere contemporaneamente in esecuzione
- Sistema di **gestione delle periferiche**
  - Garantisce l'accesso ai dispositivi di ingresso/uscita
  - Maschera i dettagli di basso livello e gli eventuali conflitti che possono sorgere nel caso di diverse richieste formulate da più utenti/programmi ad uno stesso dispositivo “contemporaneamente”
- Sistema di **gestione della memoria di massa (file system)**
  - Archiviazione e reperimento dati utilizzando le periferiche che costituiscono la *memoria di massa*



## Altri elementi (non riportati in figura)

- Sistema di **gestione della rete**
  - Permette di interfacciarsi a risorse collegate via rete e di comunicare con processi in esecuzione su altri calcolatori

*Se ne parlerà nel contesto delle reti di calcolatori...*

- Sistema di **interazione con gli utenti**  
e gestione dei relativi comandi (*interprete comandi*)

– Permette agli utenti di accedere in maniera semplice e intuitiva alle funzionalità disponibili: linguaggio grafico, ad icone, a comandi (tipico in DOS e UNIX)

*Anche se di fatto fornito con il sistema operativo, concettualmente è meglio vederlo come un'applicazione*

Vediamo in particolare il gestore dei processi, che ha un ruolo fondamentale nei sistemi multiprogrammati e multitasking...

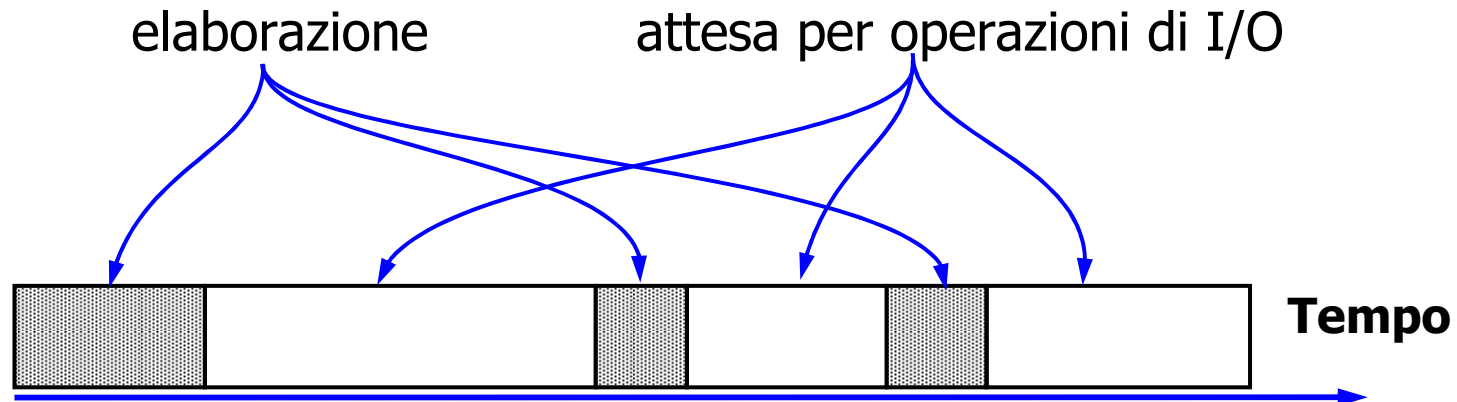
# Sistemi uniprogrammati (*monotasking*)

- Sistemi **uniprogrammati** (ad es: MS DOS): *in memoria centrale risiede un solo programma applicativo, oltre al SO*

➡ un solo programma in esecuzione in ogni istante

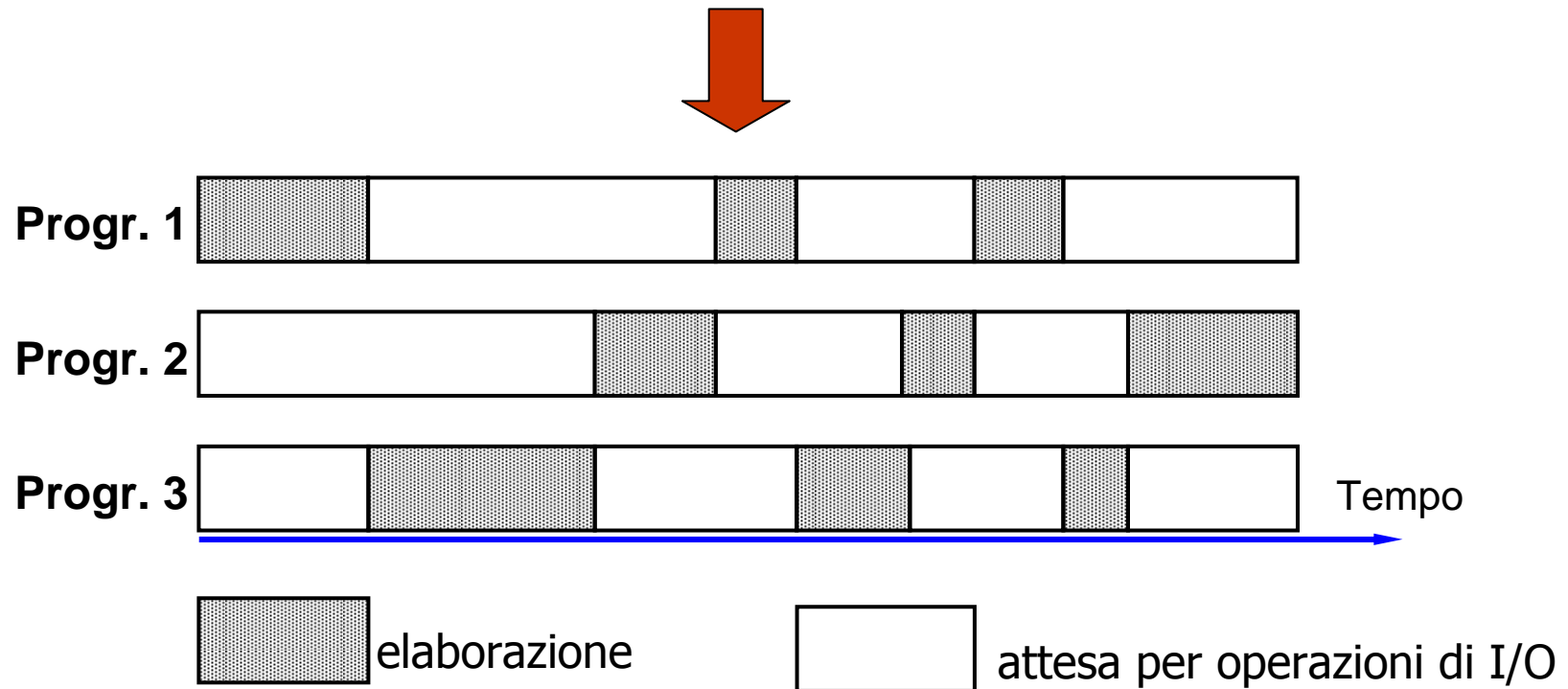


*Uso inefficiente del sistema:* durante le operazioni di input/output con le periferiche **la CPU rimane inattiva**



# Sistemi multiprogrammati e *multitasking*

- Gli attuali SO (come MS Windows XP e UNIX/Linux) sono sistemi **multiprogrammati**: *in ogni istante la memoria centrale può contenere più programmi*



# Gestione dei processi in “Time Sharing”

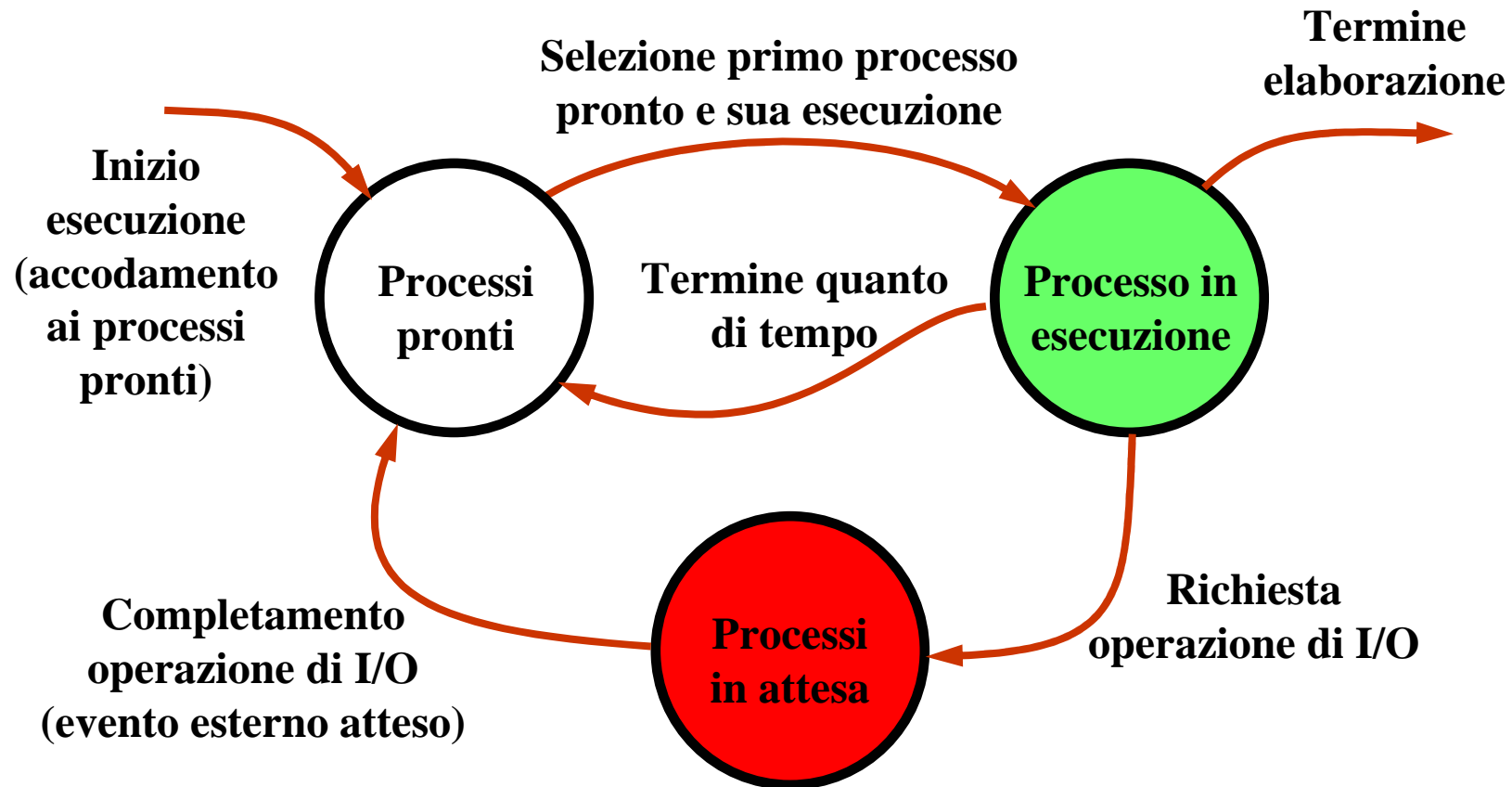
- Permette la condivisione della CPU tra più *processi* (vedi lucido successivo)
- Il tempo di CPU viene suddiviso in unità elementari dette **quanti di tempo**
- I quanti di tempo vengono assegnati ai processi dal SO secondo *opportune politiche* (es. *round-robin*): **apparente parallelismo**
- Ogni processo in esecuzione ha a disposizione un quanto di tempo di utilizzo della CPU, al termine del quale *viene sospeso* per lasciare il posto ad un altro processo in *attesa di esecuzione*

# Processo vs Programma

- **Programma:**  
entità *statica* composta dal codice eseguibile dal processore
- **Processo:**  
entità *dinamica* che corrisponde al programma in esecuzione, composto da:
  - codice
  - dati (memoria e registri unità centrale)
  - stato di evoluzione

**processo = programma + contesto di esecuzione**

# Transizioni tra gli stati di un processo



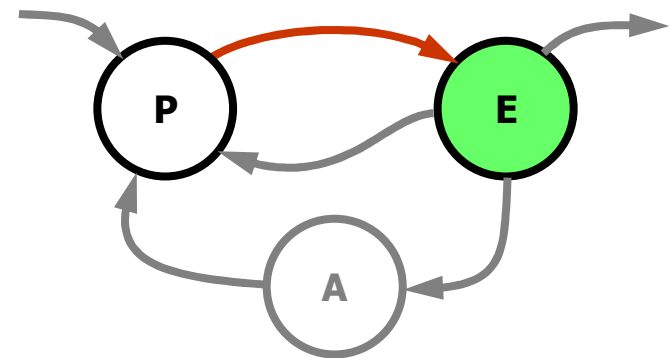
- In ogni istante un processo può essere in uno dei seguenti stati:
  - **in esecuzione** da parte della CPU (un solo processo se vi è una sola CPU)
  - **in attesa** di un evento esterno (ad esempio I/O da tastiera o su schermo)
  - **pronto** ad essere eseguito ed in attesa del proprio quanto di tempo CPU
- I processi in attesa e i processi pronti sono messi in due *code distinte* (contengono i **descrittori** dei processi, che includono il *contesto*: identificativo processo, valori registri unità centrale, priorità, risorse assegnate, ecc.)
- Tipicamente per i processi pronti si usa una coda di tipo **FIFO** (“First in First Out”)



# Transizioni di stato (1)

- **Pronto → Esecuzione**

- Quando un programma deve essere “eseguito”, viene creato un processo che viene posto in fondo alla coda dei processi pronti
- Il SO stabilisce quale dei processi “pronti” debba essere mandato in “esecuzione”, di solito estraendo il primo nella coda dei “pronti”
- Il processo rimane in esecuzione finché
  - termina il suo quanto di tempo
  - oppure richiede un’operazione di I/O
  - oppure è arrivato all’istruzione finale

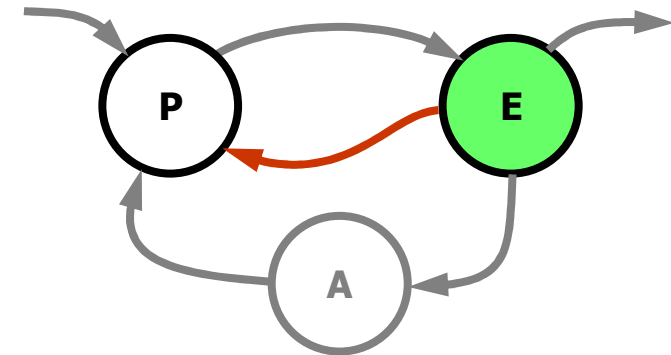


# ...se termina il quanto di tempo...

## Transizioni di stato (2)

- **Esecuzione → Pronto**

- Termina il quanto di tempo ed il processo in “esecuzione” lascia spazio a un altro processo “pronto”
- Il **SO** salva tutte le informazioni per riprendere l’esecuzione del processo dal punto in cui viene interrotta (ad es: salva il contenuto dei registri di CPU) → *salvataggio del contesto*
- Contemporaneamente un altro processo passa da “pronto” a “esecuzione”, ne viene *ripristinato il contesto* se il processo era stato precedentemente interrotto

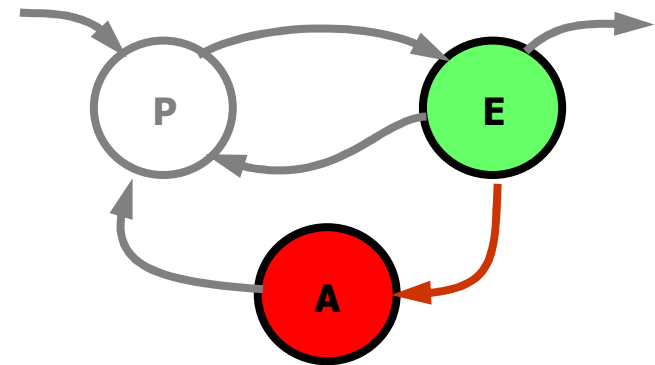


# ...attesa di un'operazione di I/O...

## Transizioni di stato (3)

- **Esecuzione → Attesa**

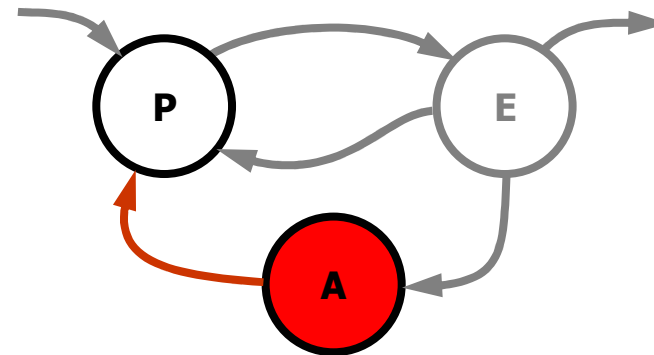
- Il processo deve svolgere un'operazione di I/O che comporta un *notevole tempo di attesa*: es. attende un evento per il completamento dell'operazione (ad es. input da tastiera)
- Il SO salva tutte le informazioni necessarie a riprendere l'esecuzione e l'informazione relativa all'evento atteso



# Transizioni di stato (4)

- **Attesa → Pronto**

- Si verifica l'evento esterno atteso dal processo ed il **SO** sposta quel processo in fondo alla coda dei processi pronti



## **Interruzioni, modo utente e modo supervisore**

Tre tipologie di interruzioni:

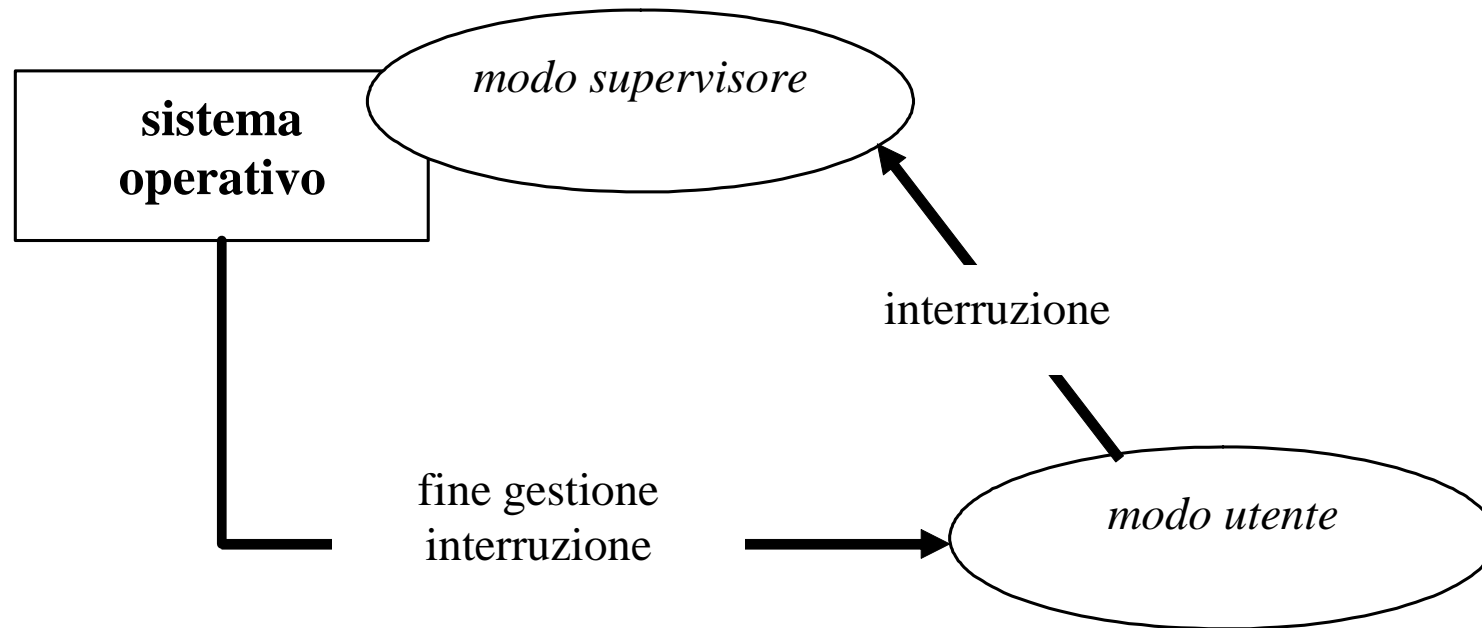
- interne: generate da specifica istruzione macchina per realizzare una chiamata di sistema
- esterne: generate dai componenti fisici del sistema (es. periferiche)
- per eccezione: segnalano situazioni eccezionali

Le transizioni dei processi da uno stato all'altro che abbiamo visto corrispondono a interruzioni esterne o interne.

Vediamo in dettaglio nel prossimo lucido (a noi interessa la parte gialla)



## Il meccanismo di protezione



- nel modo utente, non sono disponibili le istruzioni privilegiate (ad esempio quelle che consentono di interagire direttamente con le periferiche)
- i programmi del sistema operativo in modo supervisore, i programmi utente in modo utente

Vediamo ora altri due componenti: gestione periferiche e gestione della memoria di massa



# Gestione periferiche I/O (1)

- I processi e gli utenti accedono alle periferiche attraverso **comandi ad alto livello**
- L'accesso/controllo delle periferiche avviene attraverso

– **i controller**: dispositivi **hardware** il cui compito è ricevere richieste di operazioni di input/output ed eseguirle fisicamente

- dipendono dalle caratteristiche fisiche delle periferiche che gestiscono (ad es., il controller di una stampante è diverso da quello del lettore di CD ROM)

SISTEMA  
OPERA-  
TIVO

– **i driver**: programmi **software** integrati nel sistema operativo per la gestione dei controller

- mascherano le caratteristiche specifiche dei controller
- forniscono un insieme di **comandi** per la gestione delle operazioni di ingresso/uscita utilizzabili da altri programmi inclusi nel sistema operativo

# Gestione periferiche I/O (2)

- I sistemi operativi includono i driver per la gestione delle periferiche più comuni:
  - tastiera, video, mouse, stampanti, ...
- Ogni aggiunta o modifica alla configurazione standard comporta l'installazione di software aggiuntivo ovvero di *driver aggiuntivi*
  - solitamente prodotti dalla casa costruttrice della periferica (possono essere già disponibili sul CD del sistema operativo)
  - es: driver per gestire informazione acquisita con una videocamera digitale ad alta definizione

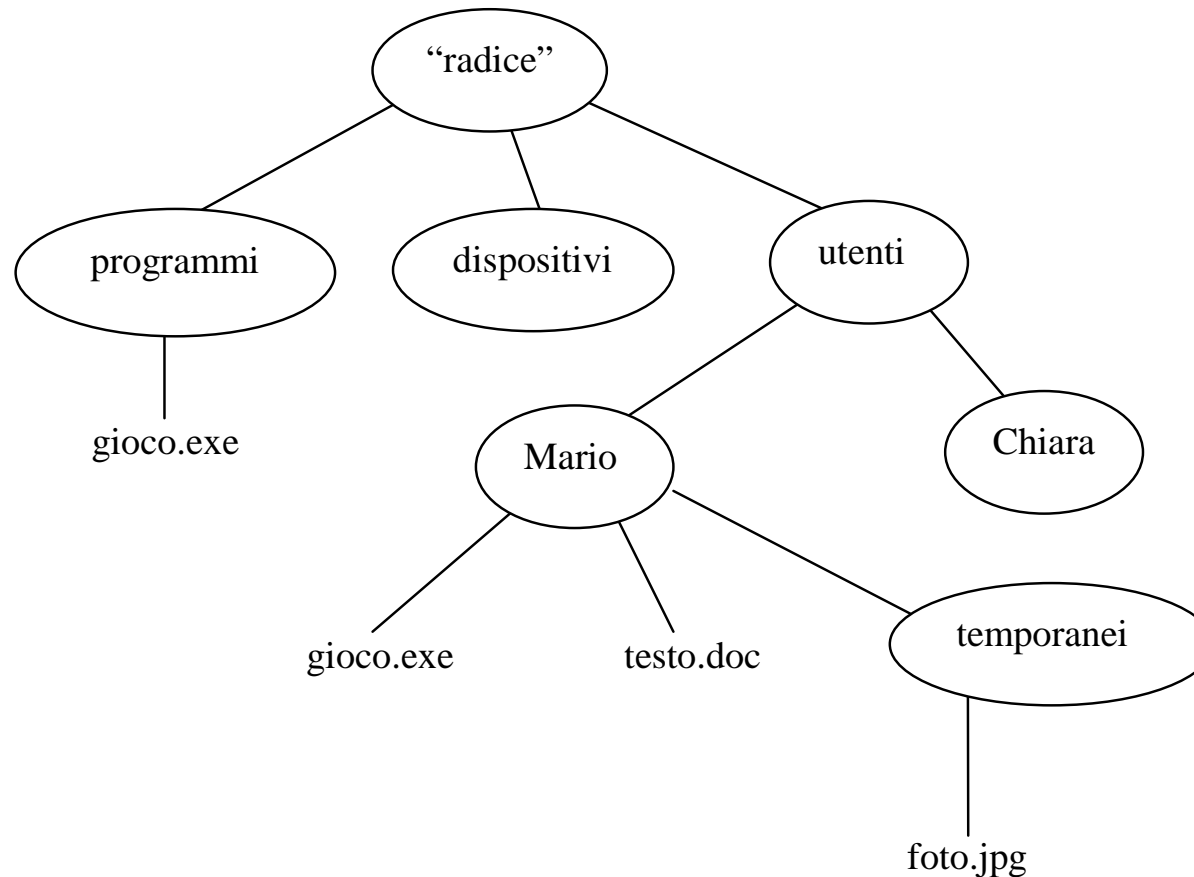
# Sistema di gestione della memoria di massa

- **Obiettivo:** presentare all'utente l'*organizzazione logica* dei dati della memoria di massa e le operazioni che è possibile compiere su di essi
- Operazioni di base di un *file system* (es. Gestione Risorse di Windows)
  - **recupero** di dati precedentemente memorizzati;
  - **eliminazione (cancellazione)** di dati obsoleti;
  - **modifica/aggiornamento** di dati preesistenti;
  - **copia** di dati (e.g. da HD a FD) per backup o per il trasferimento verso altro calcolatore
- I servizi vengono forniti sia ai **programmi applicativi** che direttamente agli **utenti**

## Organizzazione del file system

- Le informazioni vengono memorizzate in contenitori logici detti *archivi* o più comunemente *file*
- Un file è una sequenza di byte: può contenere un programma, un insieme di dati, un testo, un'immagine
- E' caratterizzato da:
  - un **nome** simbolico solitamente costituito di due parti:
    - *nome vero e proprio*, assegnato dall'utente
    - *estensione*, associata al programma che ha generato il file e consente quindi di identificare la tipologia dei dati contenuti nel file (ad es. i file Excel hanno estensione .xls)
  - un insieme di **attributi** (dimensione, data e ora di creazione, permessi di lettura/scrittura/esecuzione, proprietario)
- I file sono generalmente **organizzati in cartelle** (*directory*) e sottocartelle in una **gerarchia ad albero** (o eventualmente a grafo aciclico: in questo caso una cartella o file può far parte di più cartelle)

## Esempio



Ogni file o cartella è individuato dal suo *percorso assoluto (nome completo – pathname)*, ovvero dai nomi dei nodi intervallati da ‘\’ seguiti dal nome del file (es. \utenti\Mario\gioco.exe). Il pathname in pratica identifica la posizione del file o della cartella all’interno dell’albero.