Programmazione e Architetture (Modulo B)

Lezione 9
Cosa è un sistema operativo

Luca Manzoni

Cosa è un sistema operativo

Eseguire un programma

- L'esecuzione di un programma richiede di eseguire istruzioni (ciclo fetch-decode-execute)…
- …però per rendere un sistema semplice da usare questo non è sufficiente
- Esiste del software che permette di eseguire dei programmi (garantendo anche l'illusione di eseguirne più in contemporanea), condividendo la memoria e garantendo l'accesso ai dispositivi
- Questo è il **sistema operativo** (*Operating System - OS*).

Compiti di un OS moderno

Virtualizzare le risorse

- Il compiuto base di un sistema operativo è quello di fornire ai programmi utente una macchina più “semplice”:

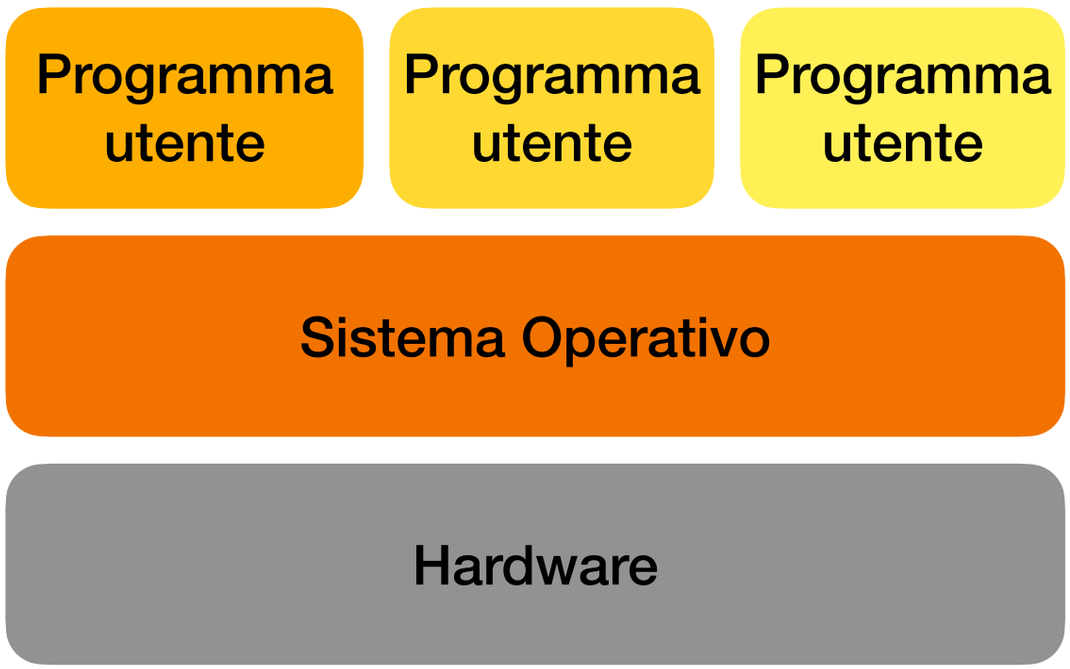
- **Virtualizzare la CPU**
  Garantisce che ogni programma abbia l’illusione di essere l’unico a eseguire sulla CPU

- **Virtualizzare la memoria**
  Garantisce che ogni programma abbia l’illusione di avere tutta la memoria a disposizione e non interferisca con gli altri programmi

- **Facilitare e regolare l’accesso alle risorse**
  Invece di comunicare direttamente coi dispositivi i programmi possono usare una interfaccia “standard” senza bisogno di sapere i dettagli

Funzione del sistema operativo

Struttura di base



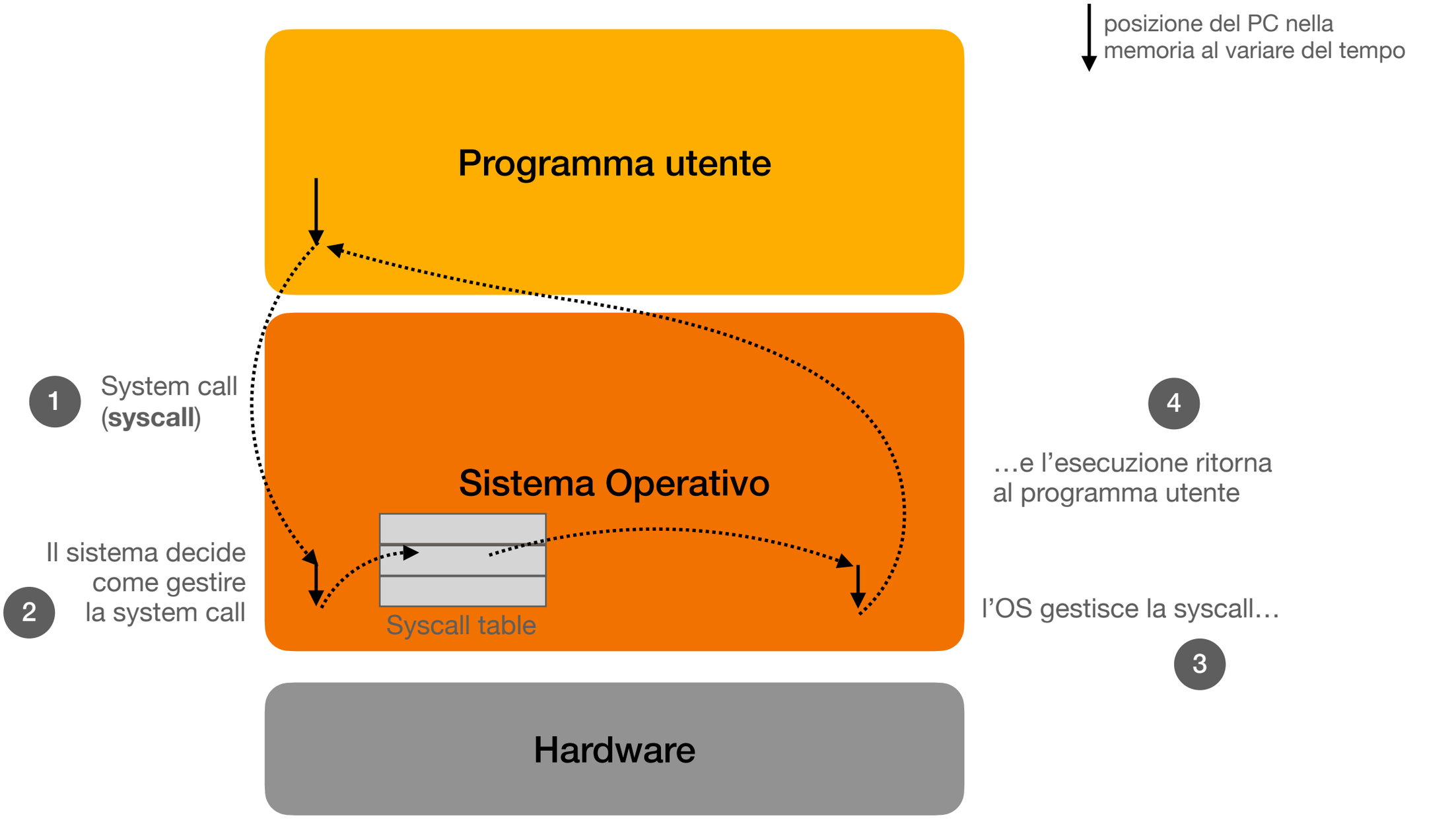
Il sistema operativo si interpone tra i programmi utente e l'hardware

Solitamente il sistema operativo opera con privilegi più elevati dei programmi utente (i.e., alcune operazioni sono disponibili solo al sistema operativo)

Ma come interagiscono i programmi con il sistema operativo?

Chiamate di sistema

Come i programmi interagiscono con l'OS



Processi

Cosa è un processo

Forma dinamica di un programma

- Nella sua definizione più astratta un processo è "*un programma in esecuzione*"
- Un programma "da solo" è semplicemente una serie di dati e istruzioni, spesso salvati nella memoria di massa
- Per essere utile un programma deve poter essere eseguito
- Il programma (dati e istruzioni) deve essere caricato in memoria
- Le istruzioni devono essere eseguite da un processore
- Per l'esecuzione servono anche un serie di risorse aggiuntive (e.g., la memoria!)

Come è composto un processo

Struttura generica

Il sistema operativo terrà traccia di tutto il necessario per permette a un processo di sospendere l'esecuzione e di riprenderla in un secondo momento

Lo stato del processore che sta eseguendo il programma, incluso il punto a cui è arrivata l'esecuzione (il PC)

Viene tenuta traccia di quale memoria sta utilizzando il programma

Le risorse (e.g., file aperti) che il programma sta utilizzando

Un descrittore di processo può contenere una serie di informazioni aggiuntive (e.g., un identificatore univoco, informazioni sui diritti di accesso alla diverse risorse, etc)

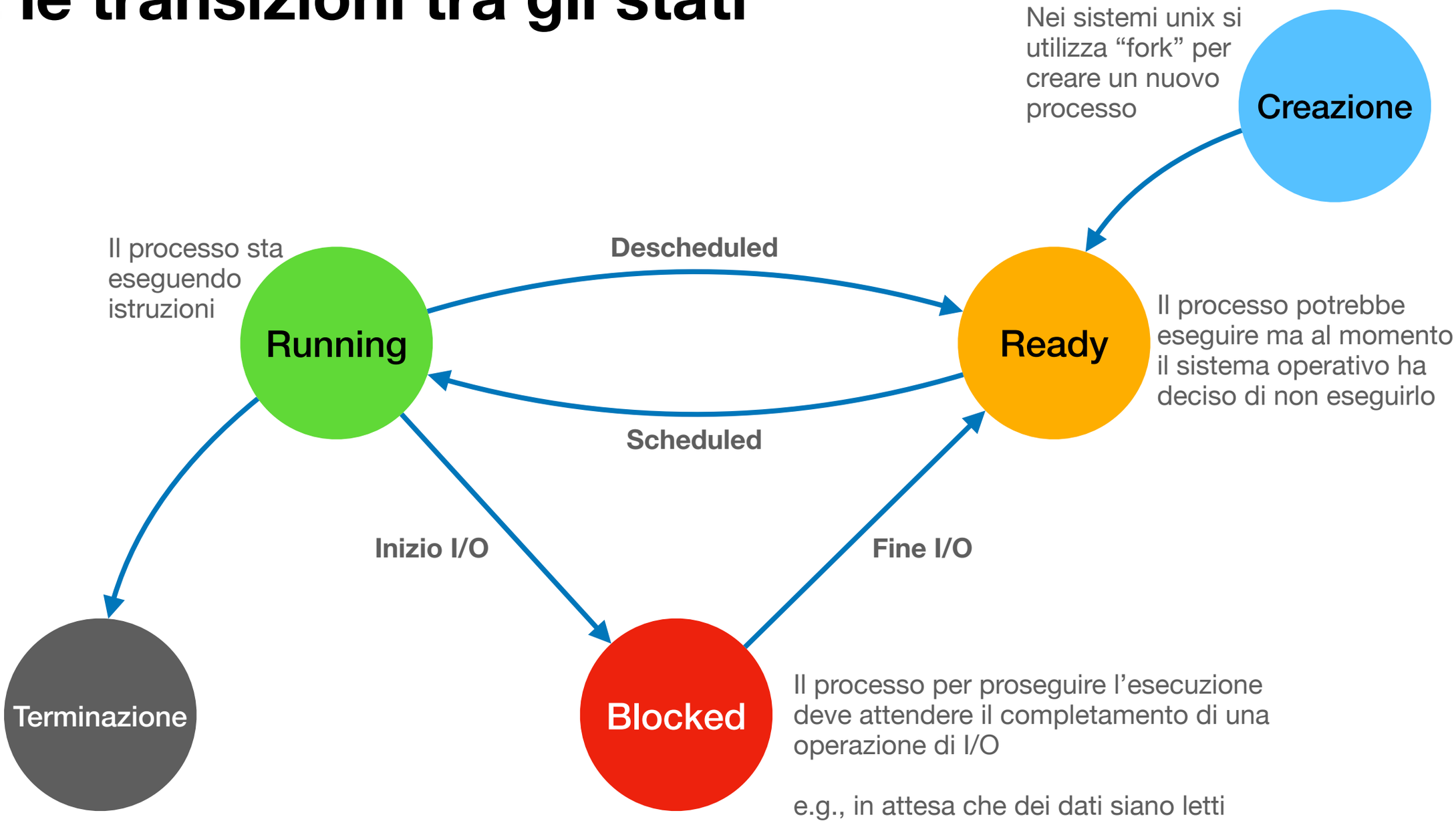
Operazioni sui processi

Forma generale di istruzioni fornite da un OS

- **Creazione** (*create*). Deve essere possibile creare un processo.
- **Distruzione** (*destroy*). In aggiunta alla creazione deve essere possibile distruggere un processo liberando tutte le risorse ad esso associate.
- **Attesa** (*wait*). Potrebbe essere necessario fermare l'esecuzione di un programma fino a quando non si sono verificate alcune condizioni.
- **Stato** (*status*). Quasi tutti i sistemi prevedono la possibilità di avere una serie di informazioni su un processo
- **Altre istruzioni di controllo**. e.g., sospensione o ripristino del processo

Stati di un processo

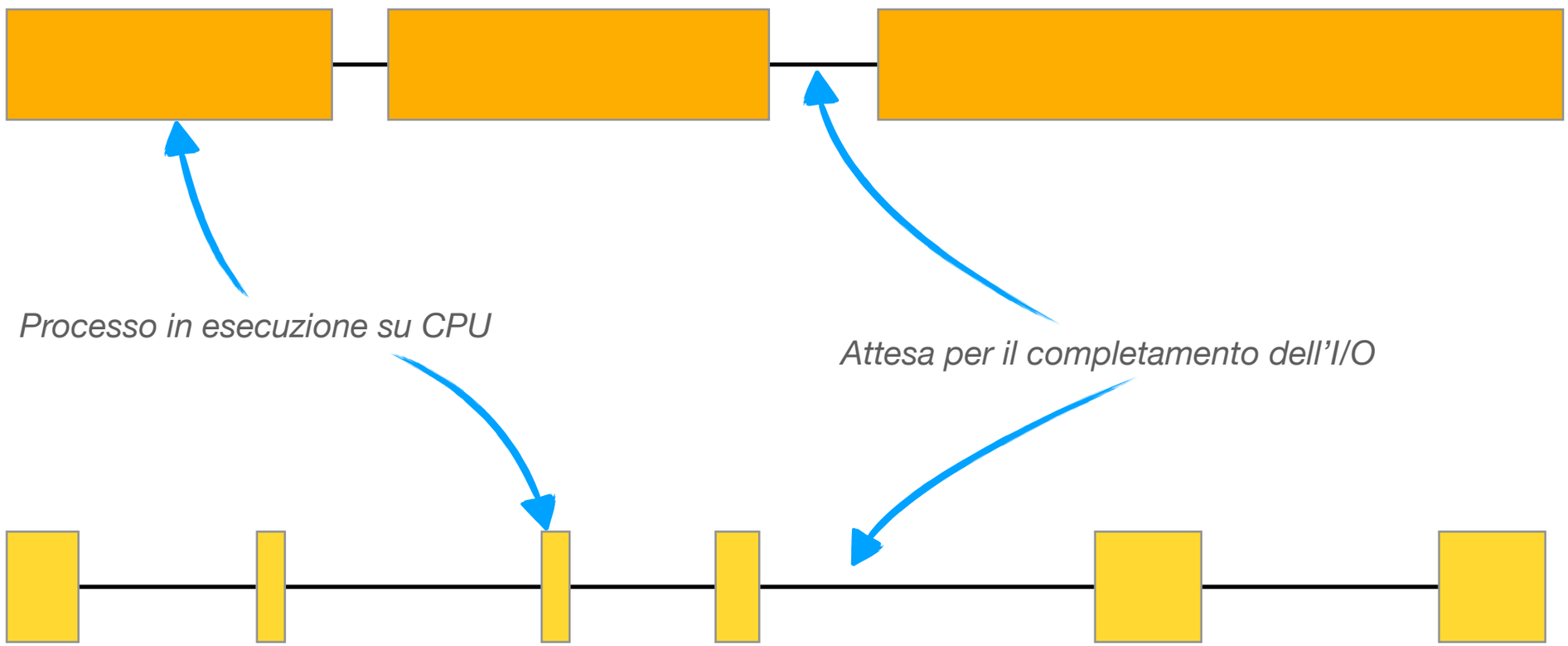
E le transizioni tra gli stati



I/O-bound vs CPU-bound

Due principali tipologie di processi

Un processo CPU-bound è un processo dominato dal tempo speso nell'eseguire su CPU, con pochino ma lunghi intervalli di utilizzo del processore intervallati da brevi attese nell'I/O



Un processo I/O-bound invece ha frequenti e corti momenti in cui usa la CPU intervallati da lunghe attese per il completamento dell'I/O

Multitasking

L’illusione di più processi in azione contemporaneamente

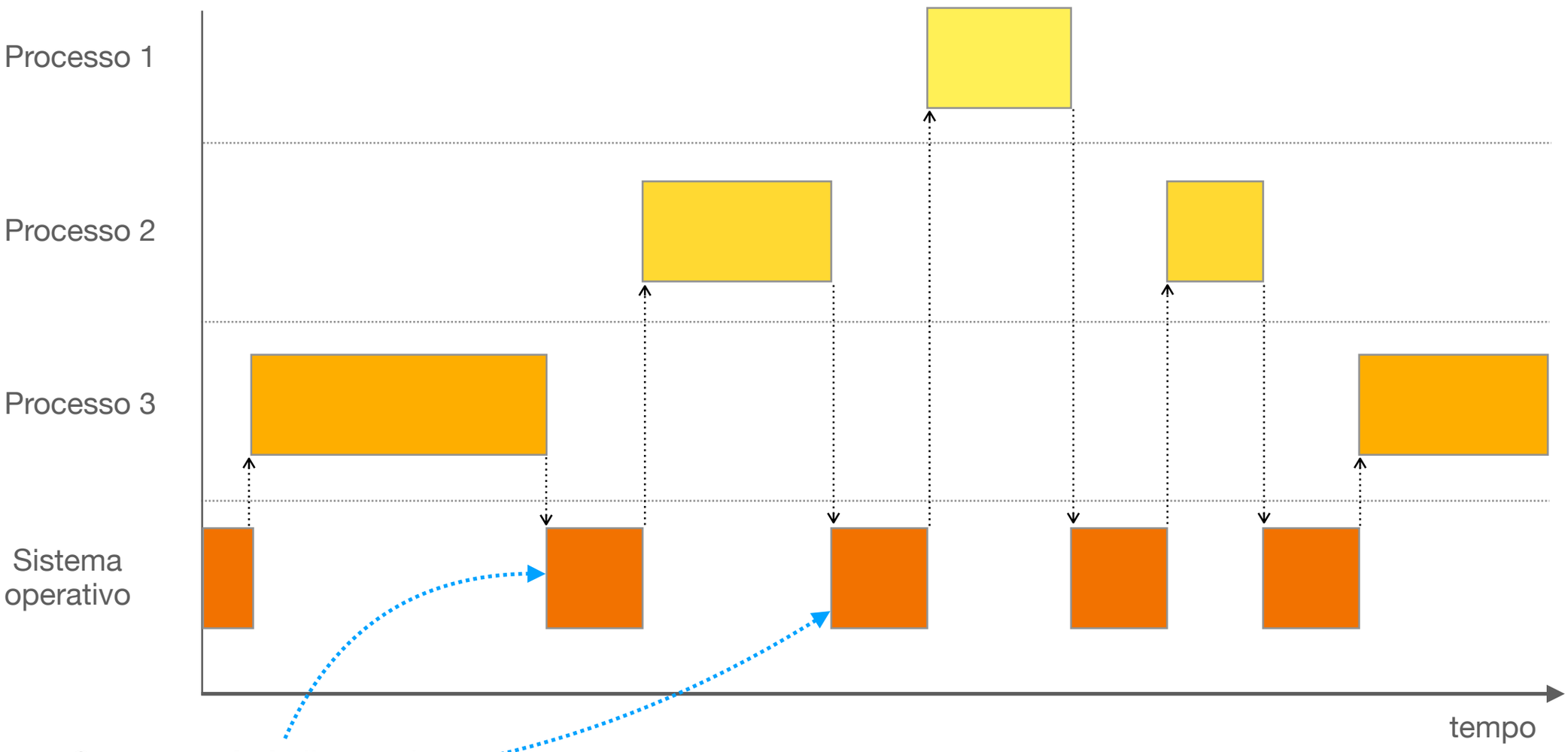
- Invece di caricare in memoria un solo programma è possibile caricarne di più ed eseguirli in parallelo
- Potenzialmente questo può migliorare le performance perché mentre un processo è bloccato per I/O un secondo processo può eseguire
- Possiamo avere sistemi che consentono il multitasking in maniera batch (enfasi sul massimizzare l’utilizzo delle risorse)
- Possiamo anche avere sistemi interattivi, quindi con tempi di risposta già bassi, tramite l’uso di time sharing

Time sharing

Condividere una singola CPU

Sebbene la CPU sia unica questa può venire assegnata a diversi processi

Se gli intervalli temporali sono abbastanza ridotti si ha l'illusione che l'esecuzione avvenga in parallelo



Context switch: l'operazione di salvare lo stato di un processo

Potenzialmente questo è un overhead che, per dare l'illusione di avere una esecuzione parallela, riduce la quantità di lavoro utile che svolge il sistema

Scheduling di processi

Scegliere quale processo eseguire

- Un sistema operativo ha una coda di processi che sono nello stato *ready*, ovvero pronti a eseguire
- Però deve scegliere il prossimo processo da mandare in esecuzione
- Questa scelta dipende dall'algoritmi di scheduling utilizzato
- Ci sono diversi algoritmi di scheduling a seconda degli obiettivi che si vogliono raggiungere

Scheduling di processi

Quando fare scheduling

- Lo scheduling può essere effettuato in diversi momenti:
  - Quando un processo viene creato bisogna scegliere se mandare in esecuzione il processo padre o quello figlio
  - Quando un processo termina bisogna scegliere quale, tra gli altri processi nello stato *ready* mandare in esecuzione
  - Quando un processo effettua dell'I/O bloccante (i.e., deve attendere che l'I/O termini)
  - Quando si riceve un interrupt è possibile effettuare una scelta di che processo mandare in esecuzione.
    Spesso vi è un dispositivo che genera interrupt ad una frequenza fissata (e.g., 50Hz, 60Hz) ed è possibile effettuare uno scheduling ogni k di questi interrupt

Scheduling: cooperative vs preemptive

Quando un processo può essere interrotto

- Lo scheduling può essere diviso a grandi linee in due categorie:
  - **Non-preemptive (o cooperativo)**. Un processo cede il processore solo quando o deve effettuare I/O o quando cede volontariamente il processore (*yield*).
  - **Preemptive**. Il sistema operativo sceglie che processo mandare in esecuzione per un tempo fissato. Se è ancora in esecuzione al termine di quel tempo il sistema operativo può sospendere il processo e mandarne in esecuzione un altro.

Scheduling di processi

Caratteristiche comuni

- Alcune caratteristiche sono comuni a tutti
  - **Fairness (equità)**. Processi simili devono avere allocate quantità simili di tempo su CPU.
  - **Policy Enforcement (applicazione delle politiche)**. Se, per esempio, si richiede che un certo processo non usi più di una certa percentuale di CPU questo deve essere rispettato.
  - **Balance (equilibrio)**. È necessario tentare di mantenere occupate tutte le risorse del sistema.

Scheduling di processi

Sistemi batch

- I sistemi batch sono caratterizzati dal non avere richieste di interattività, ma di voler completare la maggior mole di lavoro possibile per unità di tempo.

- **Throughput**. Solitamente misurato come numero di lavori completati per intervallo di tempo (e.g., lavori/ora).

- **Turnaround time**. Il tempo atteso tra l'invio di un lavoro e il suo completamento.

- **Utilizzo della CPU**. Si vuole massimizzare l'utilizzo delle risorse di calcolo

Scheduling di processi

Sistemi interattivi

- I sistemi interattivi non richiedono grande throughput, ma richiedono che ogni risposta del sistema all'utente sia rapida.
  - **Tempo di risposta**. Si vuole minimizzare il tempo di risposta anche a discapito del throughput. Se il tempo di risposta è troppo elevato il sistema non può essere utilizzato in modo interattivo.
  - **Proporzionalità**. È necessario rispettare le aspettative dell'utente (anche se possono essere incorrette) sui tempi richiesti per completare certi lavori. E.g., click su una icona richiede risposta immediata, mentre un upload potrebbe anche essere più lento (dato che comunque è considerata una azione che richiede tempo)

Tipologie di scheduler

Per sistemi batch e interattivi

- Per sistemi batch
  - First-come first-served
  - Shortest job first
  - Shortest remaining time next
- Per sistemi interattivi
  - Round-robin
  - Scheduling con priorità
  - Shortest process next
  - Guaranteed scheduling
  - Lottery scheduling
  - Fair-share scheduling

First-Come, First-Served

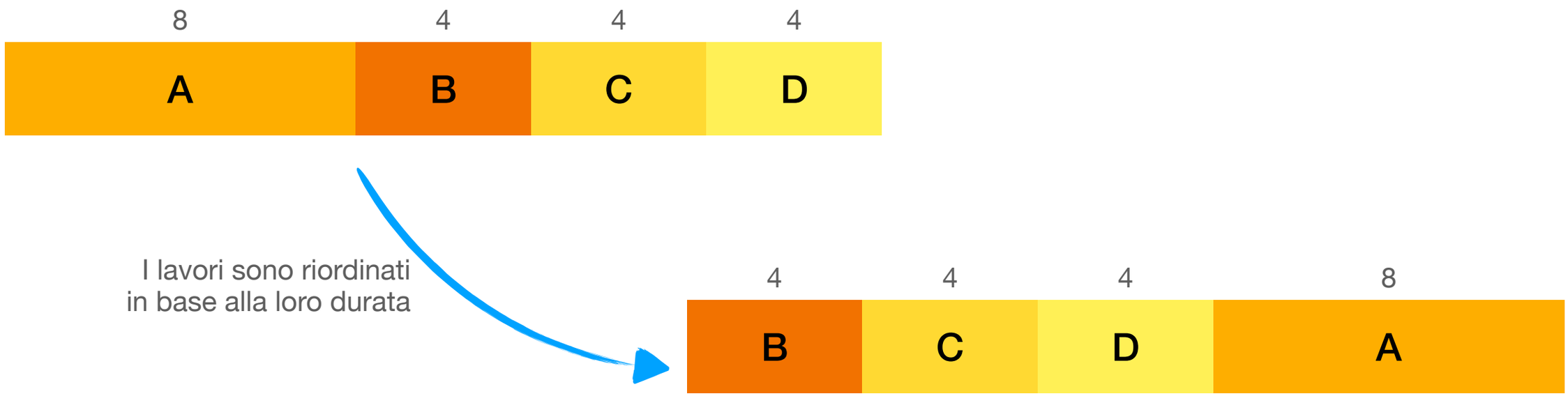
Esecuzione in ordine di arrivo

- I processi sono assegnati alla CPU nell'ordine di arrivo (quindi si ha una unica coda di processi)
- Quando un processo esegue una operazione bloccante viene eseguito il successivo nella coda
- Quando un processo passa diventa *ready* viene inserito in fondo alla coda, come se fosse un processo nuovo
- È molto semplice da implementare ma certamente non minimizza il tempo di attesa per il completamento dei lavori (e.g., pensate se il primo lavoro è lunghissimo e non ha operazioni bloccanti se non alla fine)

Shortest Job First

Esegue il lavoro che richiede meno tempo

- Supponiamo che i tempi necessari per completare ogni lavoro siano noti a priori
- Diventa possibile riordinare i lavori prima di eseguirli
- Minimizza il tempo di attesa medio per i risultati **se** tutti i lavori sono noti a priori



Shortest remaining time next

Esegue il lavoro che terminerà prima

- Una versione preemptive di **shortest job first** è **shortest remaining time next**.
- Quando un nuovo lavoro arriva il suo tempo di completamento è comparato col tempo rimanente degli altri processi
- Se il nuovo processo ha un tempo di completamento minore di quello attualmente in esecuzione il processo corrente è sospeso e l'esecuzione prosegue col nuovo processo arrivato

Round-Robin Scheduling

Condividere la CPU tra più processi

- Uno degli scheduler più semplici per sistemi interattivi
- A ogni processo viene allocato un intervallo temporale, detto **quanto** (*quantum*)
- Vi è una coda di processi e viene selezionato il primo della coda per eseguire per un quanto di tempo
- Al termine del quanto di tempo se il processo non è terminato (o non ha eseguito operazioni bloccanti) perde comunque l'utilizzo del processore e viene messo in fondo alla coda
- Dato che il context switch non è eseguito in tempo zero, diventa importante scegliere la dimensione del quanto: più è piccolo maggiori sono l'overhead e la reattività, più è grandi minori sono overhead e reattività

Round-Robin Scheduling

Condividere la CPU tra più processi

Scheduling con Priorità

Fornire diverse priorità a lavori diversi

- Non tutti i processi hanno la stessa priorità, vorremmo poter eseguire alcuni processi più spesso e altri meno spesso

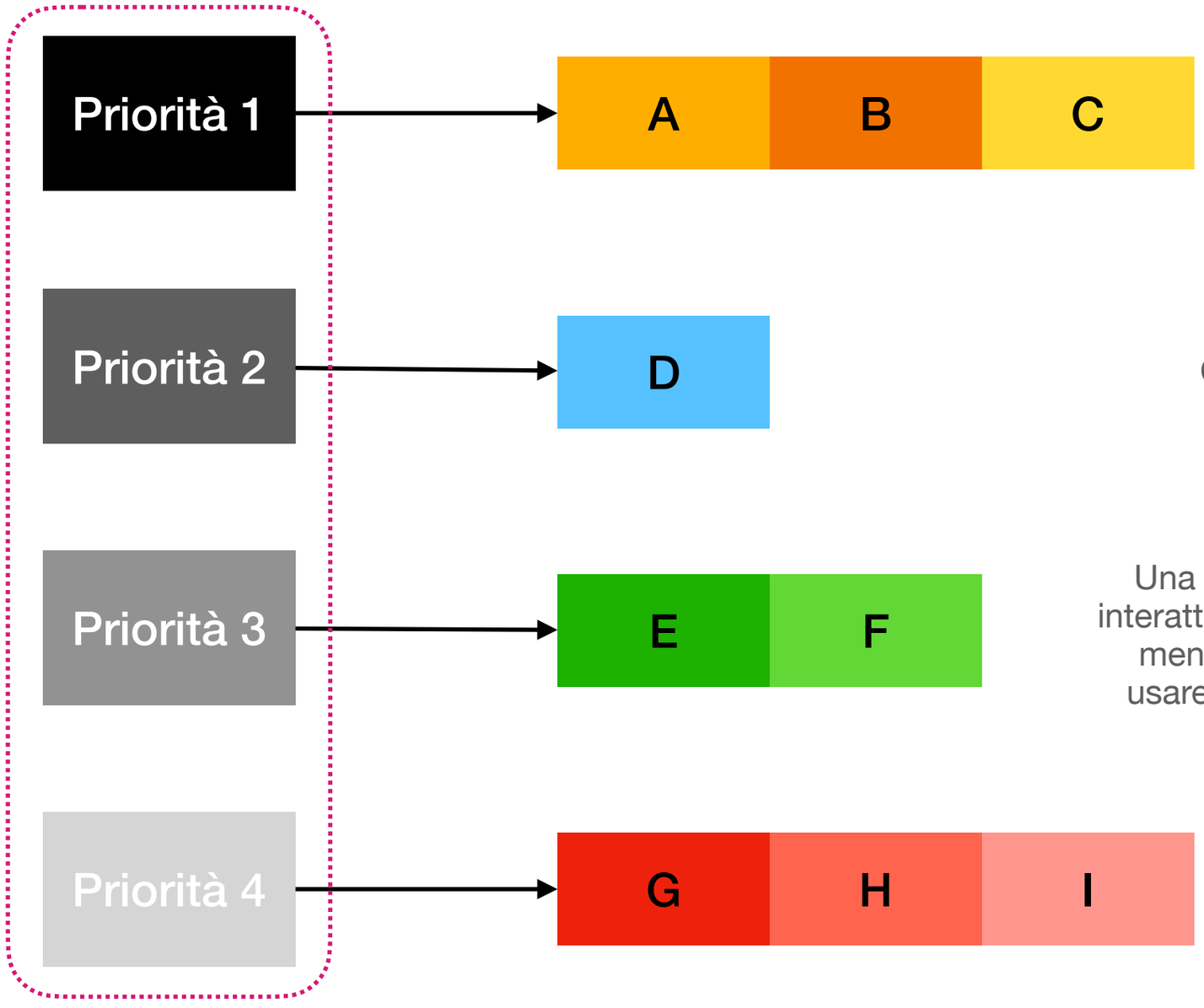
- Per garantire che ogni processo esegua (e non venga sempre “superato” da uno a priorità maggiore) ci sono diverse strategie:

  - Scelta di una policy per cui ogni priorità ottiene del tempo su CPU (questo consente di non cambiare le priorità)

  - Un processo che esegue ha la sua priorità diminuita. Quando non è più il processo a massima priorità si passa al successivo (che ora è quello a priorità massima). In questo caso le priorità cambiano nel tempo

Classi di priorità

Combinare priorità con altri scheduler



All’interno di ogni classe di priorità viene usato uno scheduler come round-robin

Classi di priorità diverse possono avere, per esempio, durata dei quanti diversa.

Una osservazione è che processi I/O bound interattivi beneficiano di tempi di risposta minori, mentre processi CPU bound beneficiano di usare la CPU per intervalli di tempo maggiori

I processi sono divisi in classi di priorità

Shortest Process Next

Shortest Job First per sistemi interattivi

- Nei sistemi batch “shortest job first” produce il miglior tempo di risposta
- Per trasferire l’idea a sistemi interattivi dovremmo sapere quale processo richiede tempo minore per terminare tra quelli ready
- Una possibilità è quella di stimare il tempo a partire da precedenti esecuzioni:
  - Inizieremo con una stima $T_S \leftarrow T_0$
  - Alla successiva esecuzione otteniamo un tempo T_1, aggiorniamo la nostra stima come $T_S \leftarrow \alpha T_0 + (1-\alpha)T_1$ con $\alpha \in [0,1]$
  - In generale prendiamo la nostra stima attuale e, ottenuto un nuovo tempo di esecuzione T_i aggiorniamo la stima come $T_S \leftarrow \alpha T_S + (1-\alpha)T_i$
  - Facile da implementare, specialmente con $\alpha = 0.5$

Guaranteed Scheduling

Garantire una certa quantità di risorse

- Possiamo voler garantire che ognuno di n processi riceva una frazione $1/n$ del tempo del processore
- Possiamo calcolare il rapporto $\frac{\text{tempo CPU consumato}}{\text{tempo CPU a cui si ha diritto}}$
- Se il rapporto è < 1 allora dobbiamo dedicare più tempo a quel processo
- Se il rapporto è > 1 allora allora dobbiamo dedicare meno tempo a quel processo
- Possiamo eseguire il processo col rapporto minore (e cambiarlo con un altro processo quando non è più tale)

Lottery Scheduling

Una semplice implementazione per garantire risorse

- Implementare il guaranteed scheduling non è banale
- Una possibile soluzione è assegnare a ogni processo un certo numero di “biglietti”
- Per decidere il prossimo processo che potrà fare uso della CPU si estrae uno dei biglietti. Il processo che lo possiede avrà accesso alla CPU
- In questo caso la frazione *attesa* di uso della CPU dipende dal numero di biglietti di un processo rispetto al totale

Fair Share Scheduling

Dividere il processore tra più *utenti*

- Fino ad ora abbiamo visto come il tempo su CPU è diviso tra più processi
- In sistemi multi-utente possono esserci più utenti ognuno con un diverso numero di processi in esecuzione
- E.g., Utente 1 con 20 processi e Utente 2 con 2 processi. Non vogliamo che l'utente 1 riceva 10 volte più tempo su CPU dell'utente 2
- Per questo il fair share scheduling utilizza la conoscenza degli utenti a cui appartengono i processi per garantire equità tra gli *utenti,* non solo tra i processi