Architetture Degli Elaboratori

Lezione 06
Gestione e condivisione risorse: Processi, OS, CPU e memoria

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**, astraendo le risorse”:

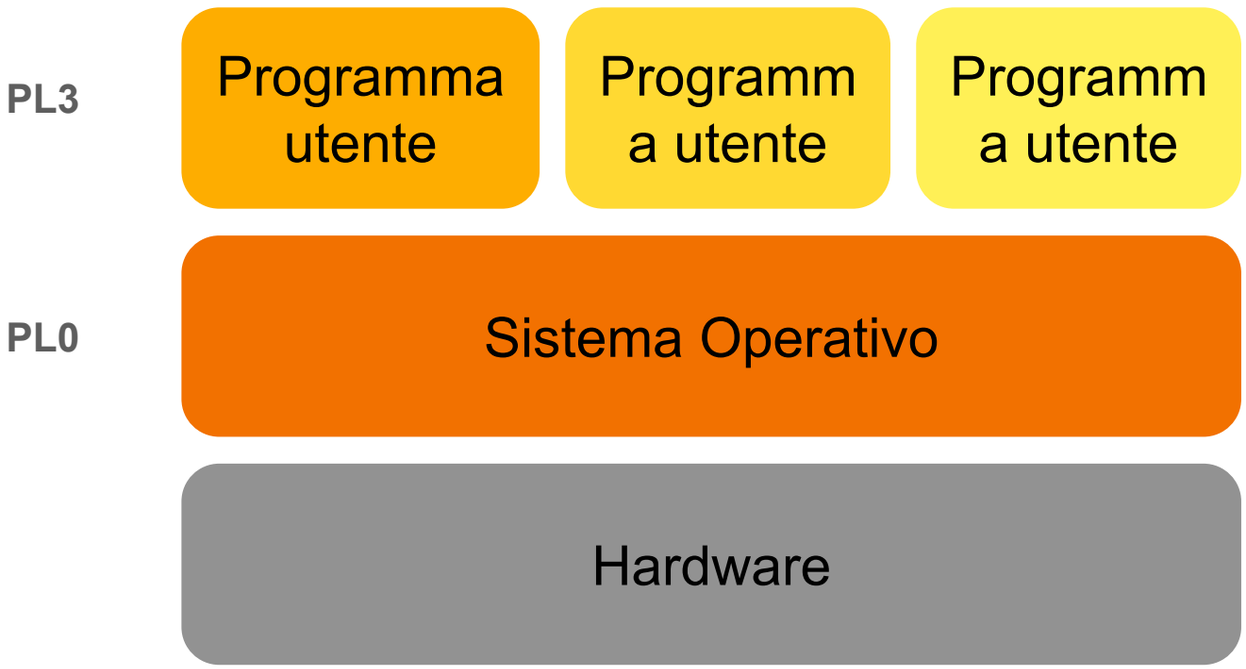
- **Virtualizzare la CPU**
  Garantisce che ogni programma abbia l’illusione di essere l’unico a eseguire sulla CPU

- **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



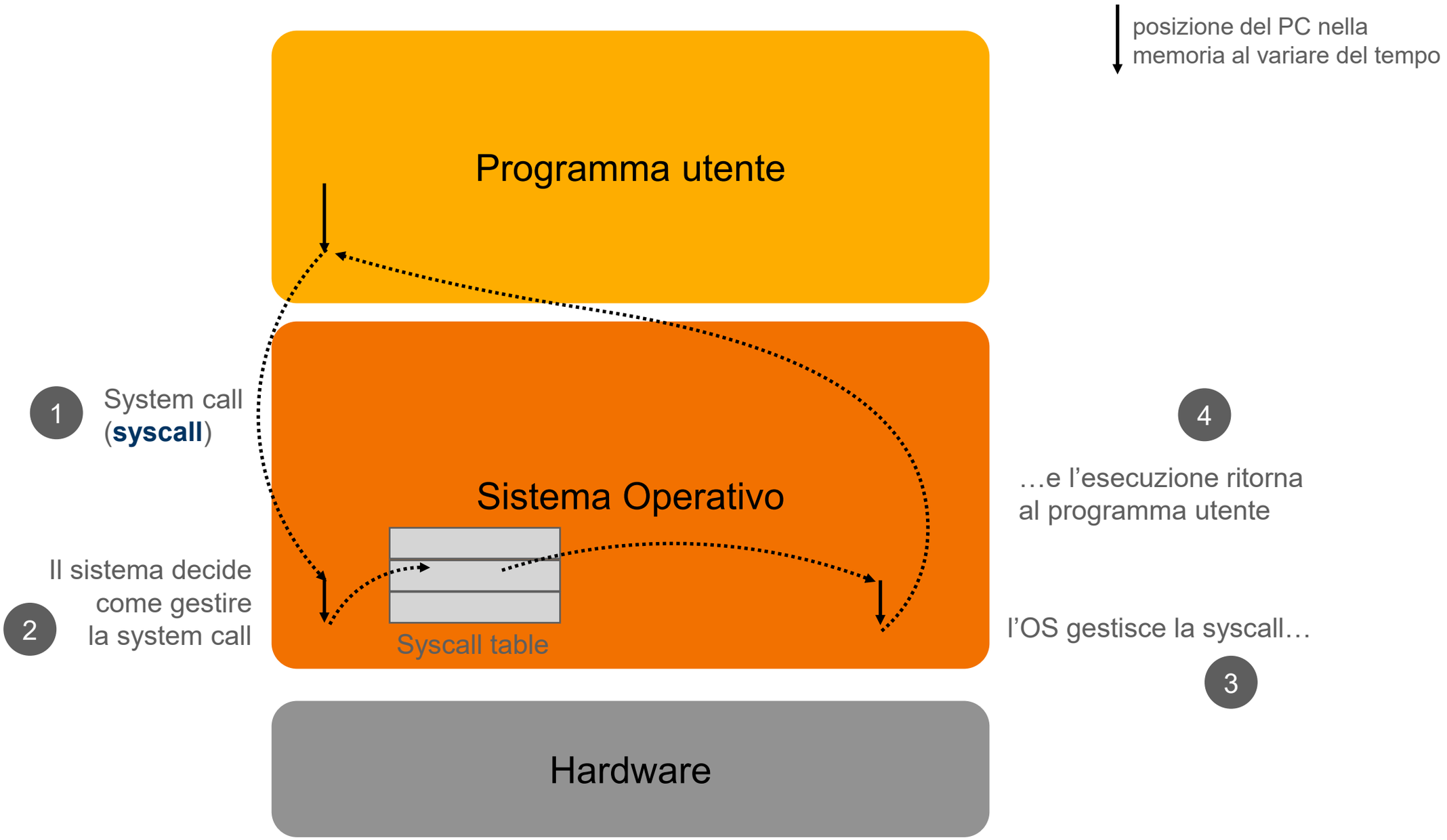
Il sistema operativo si interpone **tra** i **programmi utente** e l'**hardware**

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

Ma come interagiscono i programmi con il sistema operativo?

Chiamate di sistema

Come i programmi interagiscono con l'OS



Chiamate di sistema

Come i programmi interagiscono con l'OS

Si usano **syscall** per fare I/O, aprire/chiudere **file**, creare/eseguire/distruggere **processi**, gestire **memoria** dinamica, controllare **dispositivi**, …

Vantaggi:

- **Sicurezza**: i programmi non hanno accesso diretto all'hw
- **Portabilità**: stessa applicazione può girare su sistemi diversi con stessa 'API'
- **Gestione risorse**: centralizzata e coordinata per **processi multipli**

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 statica 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!)
- Esecuzione può essere sospesa o ripresa in base alle risorse disponibili
- **D**: cosa ci serve per poter sospendere (e poi riprendere) un processo?

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 **ID** univoco, informazioni sui diritti di accesso alla diverse risorse, etc)

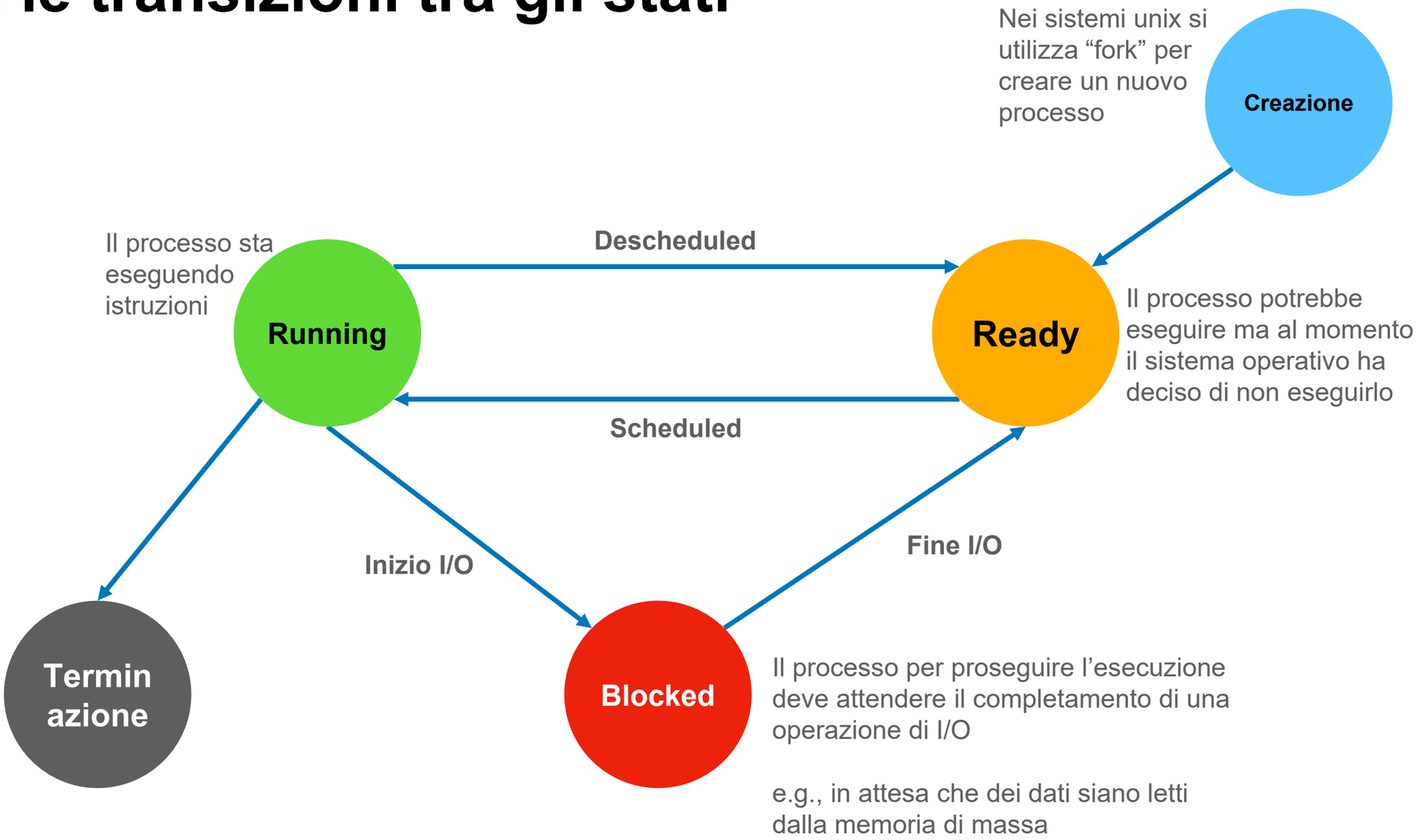
Operazioni sui processi

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

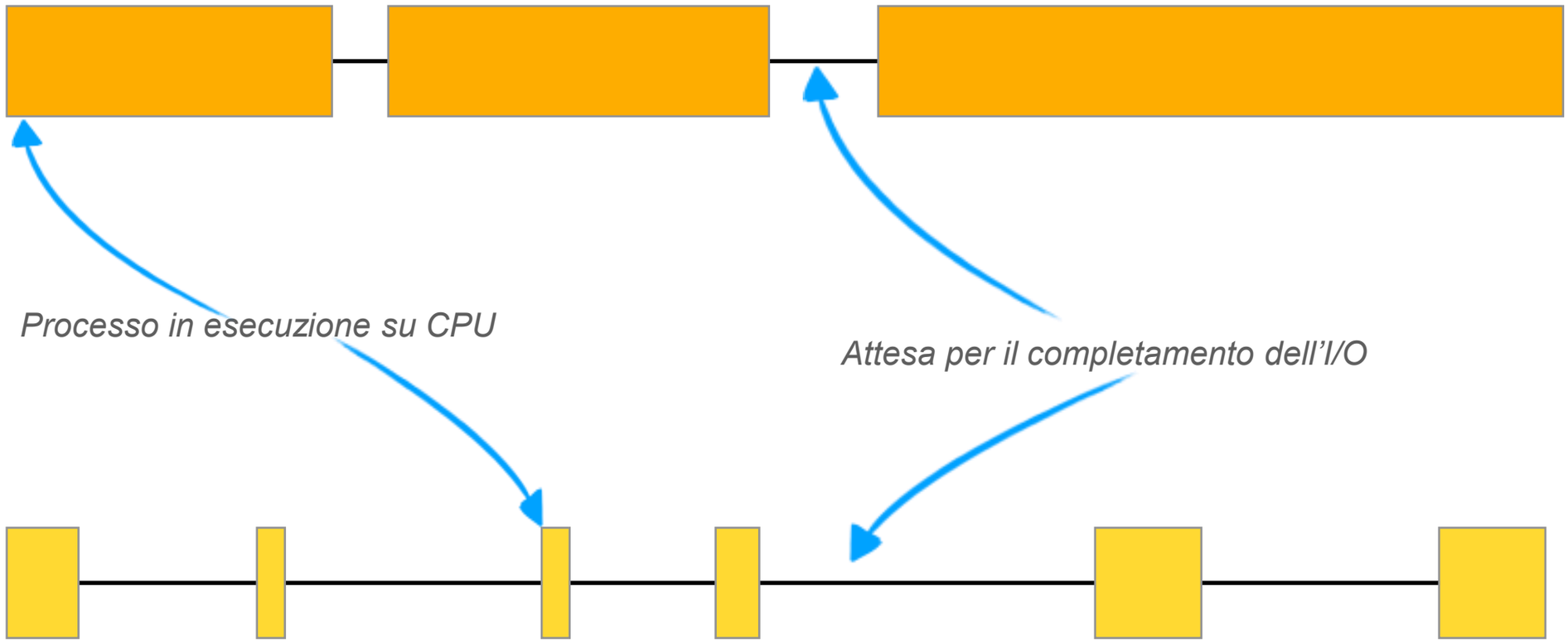
E le transizioni tra gli stati



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 pochi 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

D: in quale categoria sta uno streaming video da internet?

Multitasking

L’illusione di più processi in azione contemporaneamente

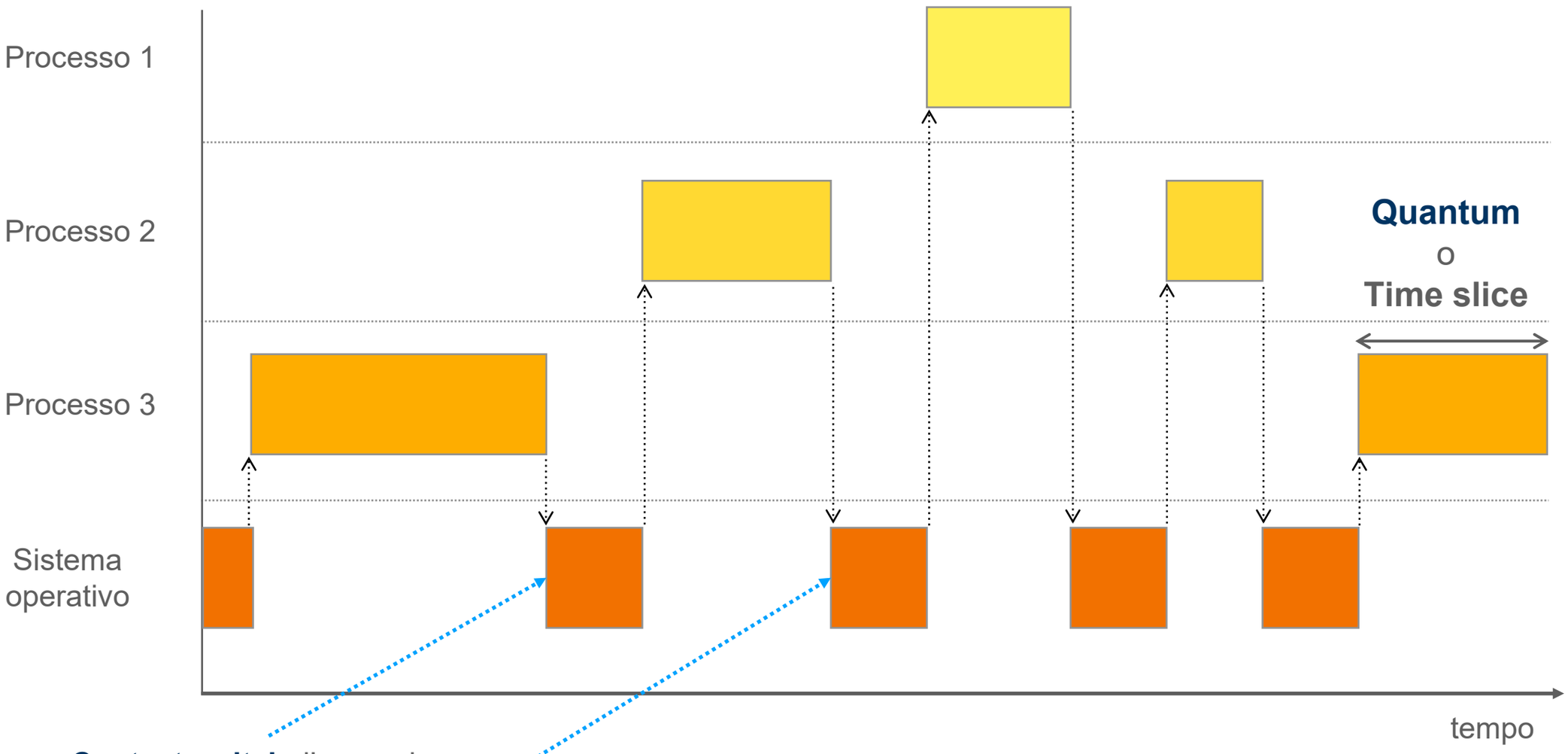
- Invece di caricare in memoria un solo programma è possibile **caricarne più programmi** ed **eseguirli in parallelo**
- Potenzialmente questo può migliorare le performance perché mentre un processo è bloccato per I/O un secondo processo può eseguire
- Possiamo avere **sistemi** che consentono il multitasking 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'**algoritmo 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 sono dei dispositivi (**timer**) 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**. Tempo atteso fra evento e prima assegnazione CPU. Se il tempo di risposta è troppo elevato il sistema non può essere utilizzato in modo interattivo.

D: perché miglioralo può peggiorare il throughput?

- **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)

Astrarre la memoria

Astrarre la memoria

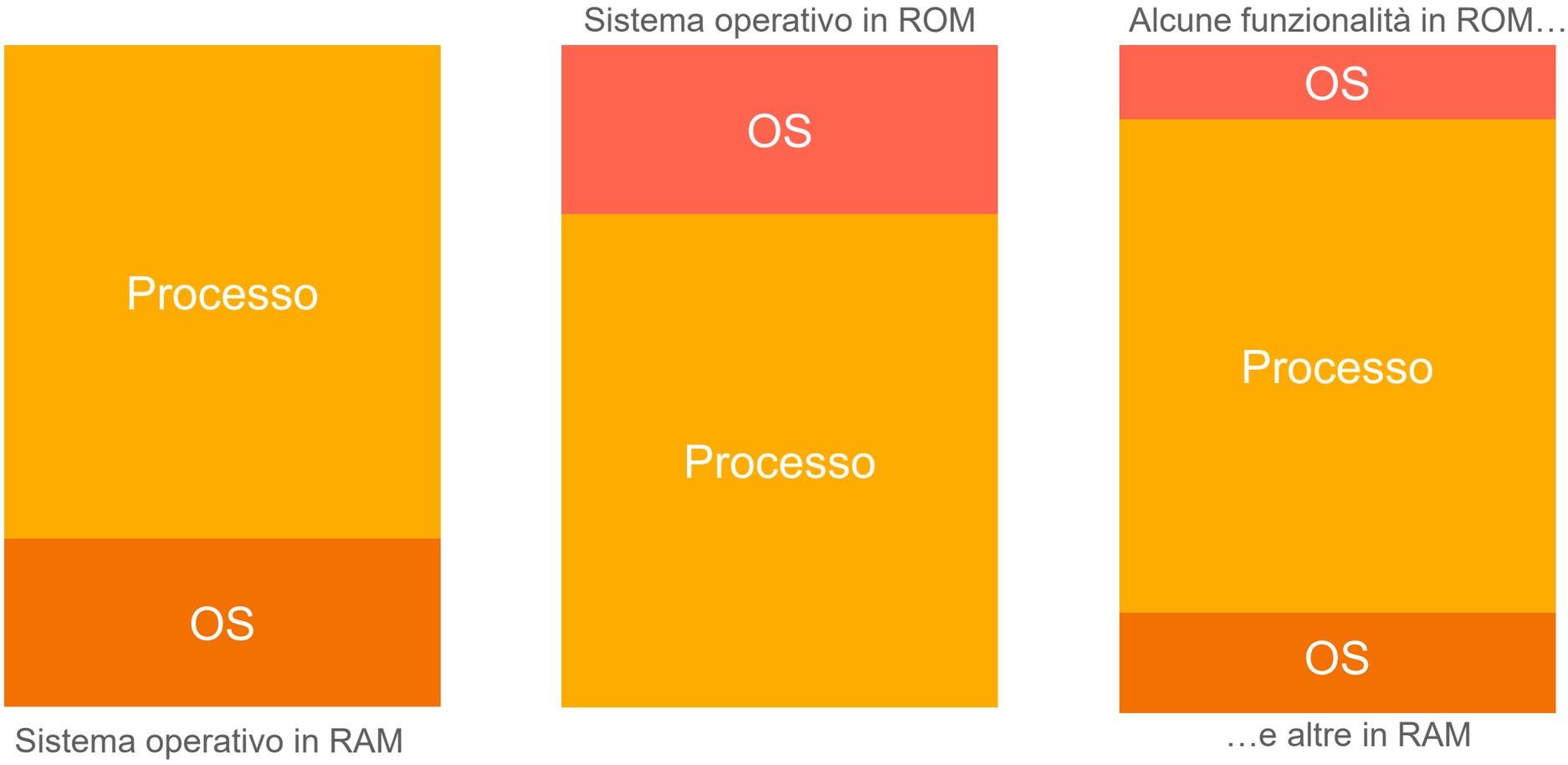
Da nessuna astrazione alla memoria virtuale

- Nessuna astrazione
- Static relocation
- Base + limit
- Swapping
- Gestione della memoria libera
- Memoria virtuale e paging

Programma Singolo

Far convivere sistema operativo e un singolo processo

- Un sistema operativo può convivere con un singolo processo anche senza nessuna astrazione della memoria
- Si devono avere delle **convenzioni** su quali indirizzi sono occupati dal sistema operativo e quali sono usabili dal processo
- Approccio usato nei mainframe (fino agli anni ’60), minicomputer (fino agli anni ’70) e home computer (fino agli anni ’80)

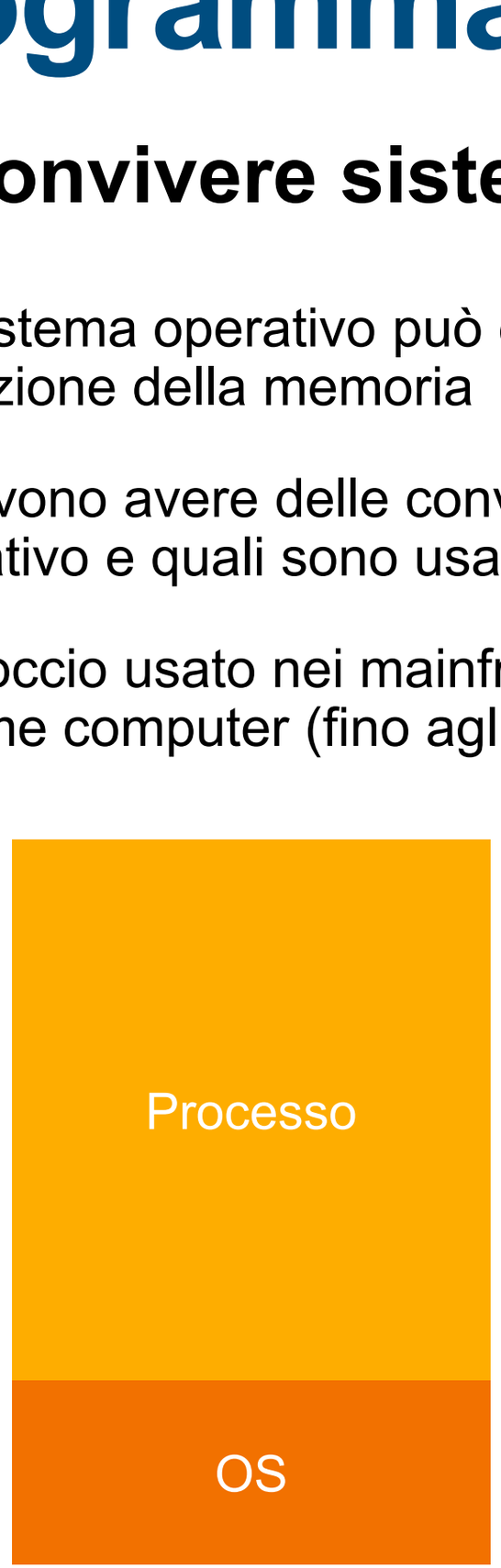


Programma Si

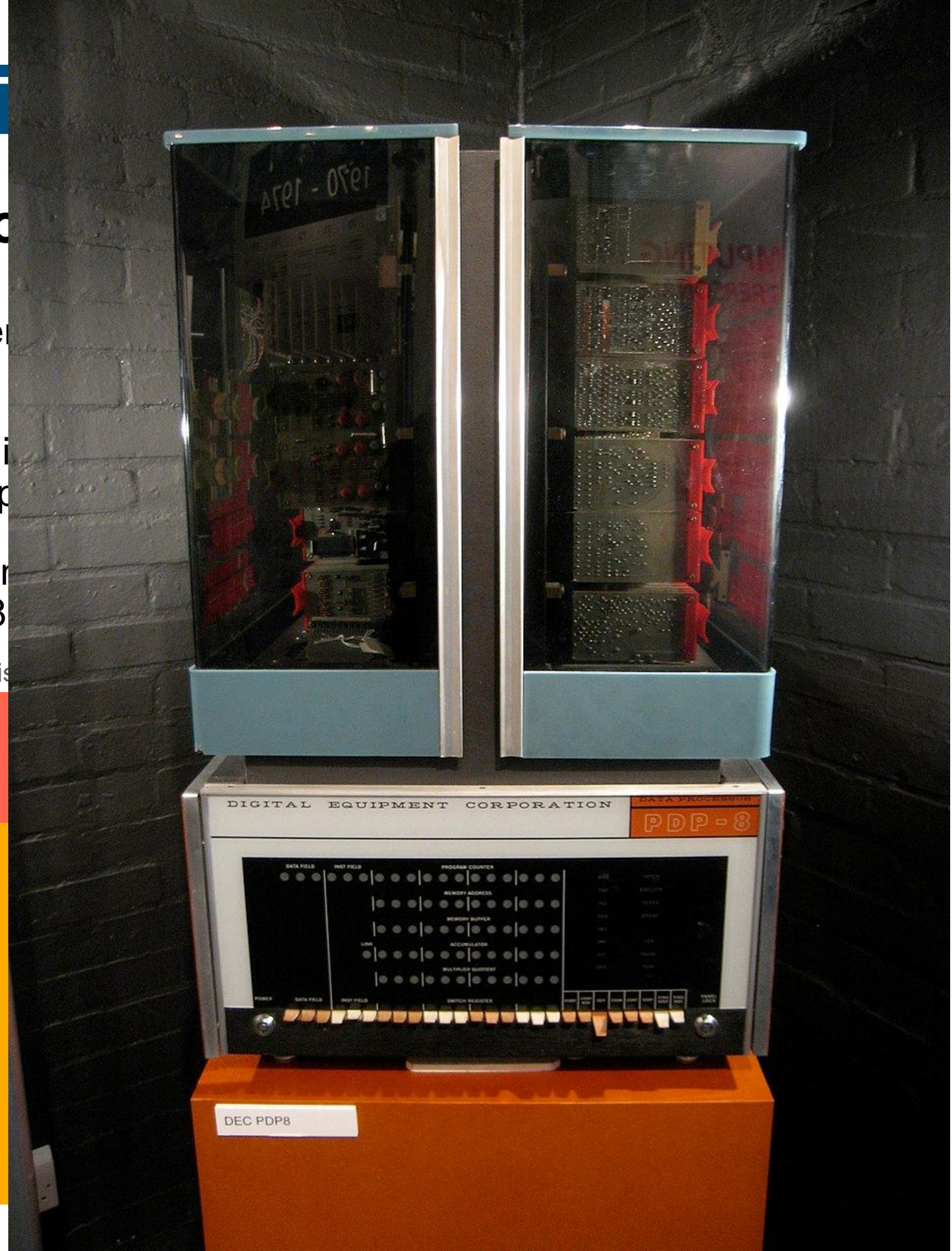
Far convivere sistema c

- Un sistema operativo può convive
  astrazione della memoria
- Si devono avere delle convenzioni
  operativo e quali sono usabili dal p
- Approccio usato nei mainframe (fi
  e home computer (fino agli anni ’8

Sis



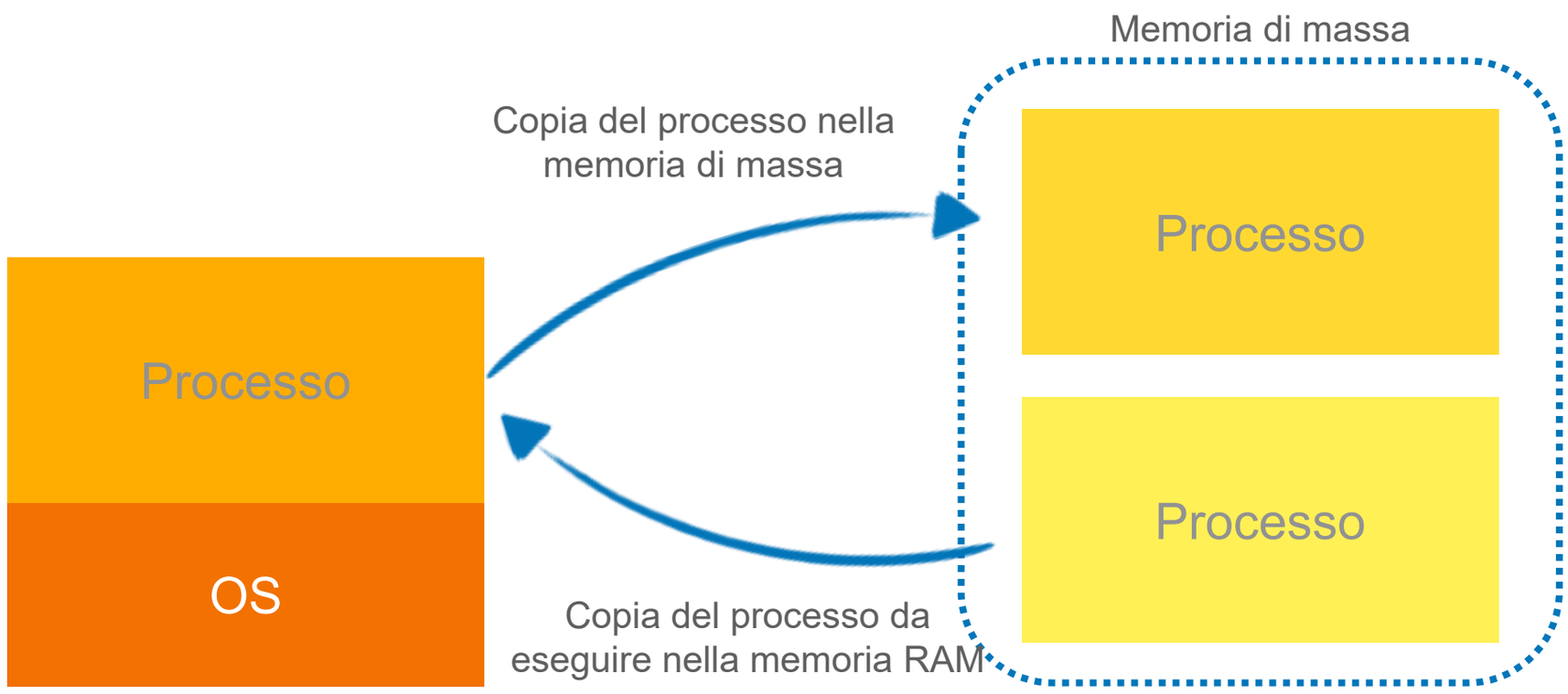
Sistema operativo in RAM



Processi multipli

in un sistema senza nessuna astrazione

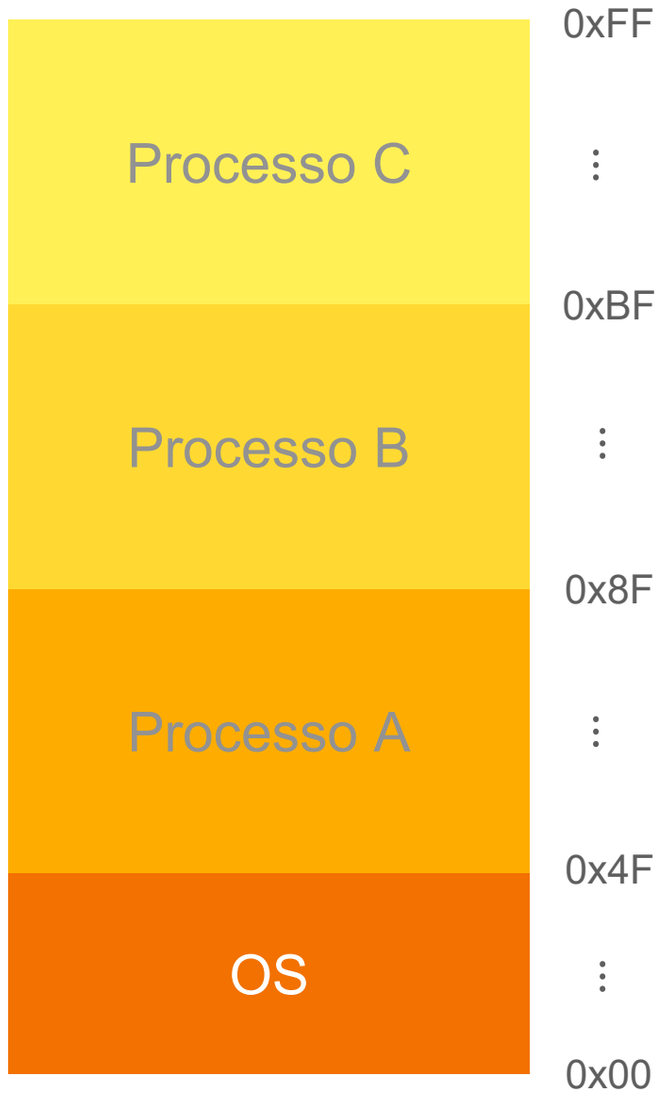
- Anche per sistemi senza nessuna astrazione è possibile gestire più programmi
- Quando un processo deve uscire dallo stato “running”, tutta la sua memoria viene copiata nella memoria di massa…
- …e l’intera memoria del prossimo processo che deve eseguire è copiata dalla memoria di massa alla memoria RAM.



Processi multipli

Senza astrazione della memoria

- Possiamo caricare più programmi in locazioni diverse della memoria…
- …ma come possiamo fare a non fare in modo che si sovrascrivano a vicenda dei dati?
- E.g., se il processo A vuole scrivere all'indirizzo 0x5A, non ci sono problemi (è nella area a lui dedicata)
- Se il processo B vuole scrivere all'indirizzo 0x5A non può, l'indirizzo andrebbe convertito spostandolo "in alto" di 0x4F (dato che la memoria del processo B "inizia" 0x4F più in alto di quanto atteso)
- Una possibilità è far riscrivere al sistema operativo gli indirizzi di memoria quando carica il codice del processo (**rilocazione statica**)
- Non è immediato capire quali sono gli indirizzi e quali i dati!



Spazio degli indirizzi

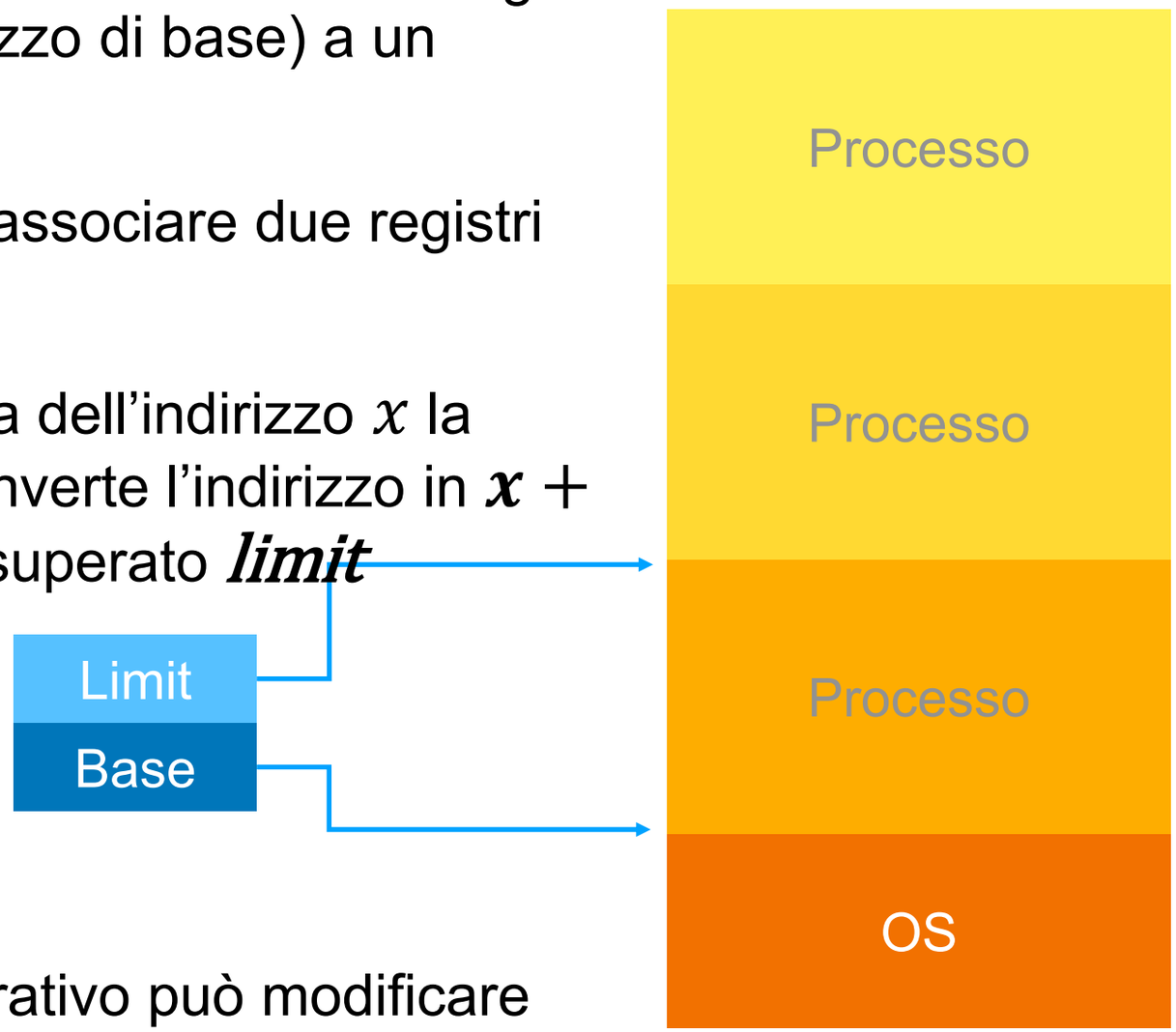
Come nascondere gli indirizzi fisici ai processi

- Rendere visibili gli indirizzi di memoria reali ai processi è problematico
- Nasce quindi l'idea di uno "**spazio degli indirizzi**" o *address space*. L'idea è che ogni programma ha accesso a un insieme di indirizzi di memoria (e.g., da 0 a $2^{32} - 1$) indipendentemente da quanta sia e come sia organizzata la memoria fisica
- È compito di **hardware** e **sistema operativo** di **convertire** ogni accesso a un indirizzo **virtuale** a un indirizzo **fisico**.
- Uno degli approcci più semplici è quello di utilizzare una **rilocazione dinamica** tramite **due registri aggiuntivi** (base e limit)

Base + Limit

Allocare una frazione della memoria a ogni programma

- Uno degli approcci più semplici è di allocare un range di indirizzi (a partire da un indirizzo di base) a un processo
- Generalmente questo significa associare due registri (base e limit) ad ogni processo
- Quando viene richiesta la lettura dell'indirizzo x la **Memory Management Unit** converte l'indirizzo in $x +$ ***base*** e verifica che non venga superato ***limit***
- Diventa anche possibile permettere ad ogni processo di avere più **segmenti** e di farli crescere in base alle necessità
- Solitamente solo il sistema operativo può modificare base e limit



Swapping

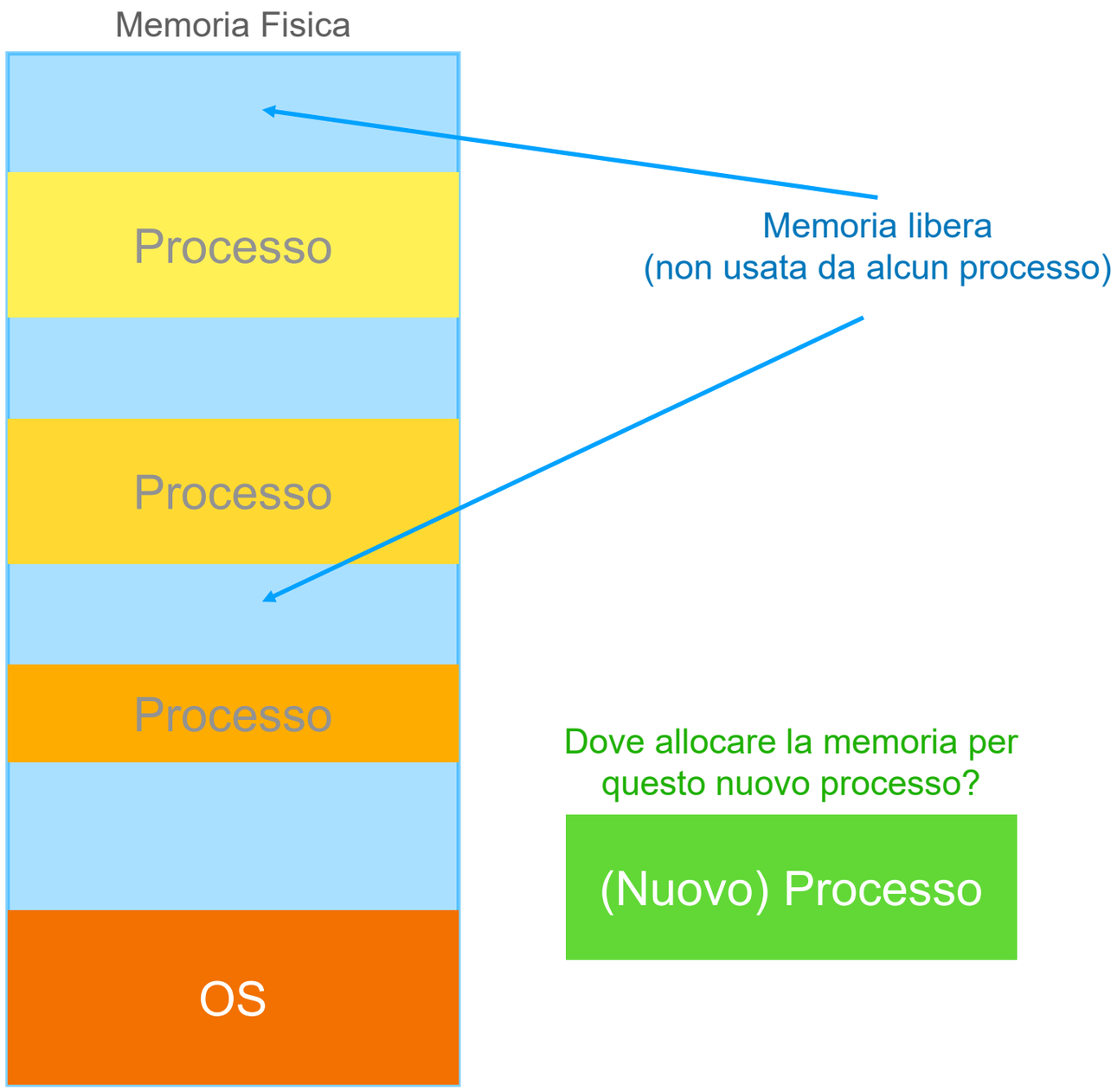
Usare più memoria di quella disponibile

D: se l'ultima allocazione di memoria ha raggiunto le dimensioni della RAM, dobbiamo impedire nuove allocazioni?

- Se abbiamo più processi di quanti ne possono stare in memoria possiamo usare una tecnica chiamata **swapping**
- Lo swapping consiste nel copiare l'intera memoria occupata da un **processo su memoria di massa** (e.g., un disco) e ricopiarlo in memoria quando deve essere nuovamente eseguito
- I **processi** che sono **poco attivi** passeranno la maggior parte del tempo su **disco**, questo permette di avere più processi attivi di quanti la memoria permetterebbe
- Notate che nella sua definizione stretta lo swapping sposta l'intero processo su disco. Un'altra tecnica che permette di spostare solo parte del processo è detta **paging**

Gestire la memoria libera

Decidere dove posizionare i processi



- Ogni processo può avere associati uno o più range di indirizzi di memoria
- Questi range potrebbero non coprire l'intera memoria fisica
- Quando un nuovo processo richiede altra memoria dobbiamo decidere dove allocarla
- Serve anche avere delle strutture dati adeguate per **supportare** l'**allocazione** e la **deallocazione** della **memoria**

Allocazione della memoria

Possibili algoritmi

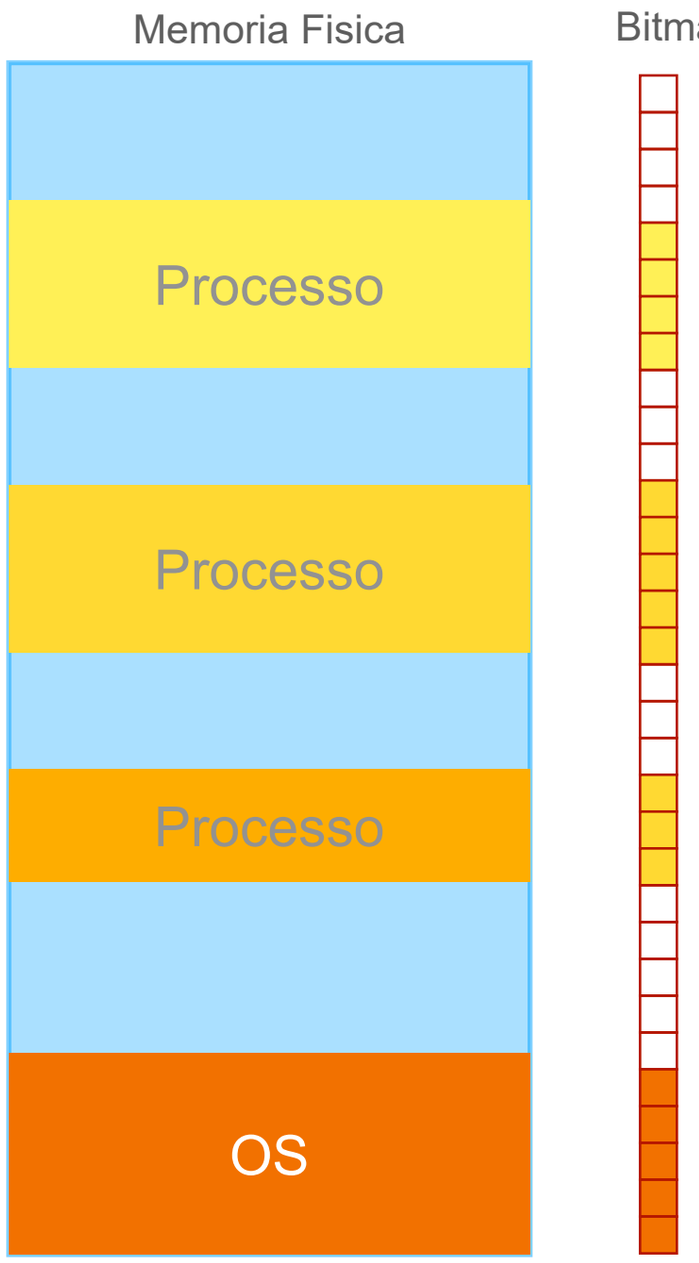
- **First fit**. Si sceglie il **primo slot** disponibile in grado di soddisfare la richiesta di memoria
- **Next fit**. Si tiene traccia di dove era avvenuta l'ultima allocazione e si prosegue la ricerca da quel punto (per il resto uguale a first fit)
- **Best fit**. Cerca lo **slot più piccolo** in grado di soddisfare la richiesta di memoria
- **Worst fit**. Cerca lo **slot più grande** in grado di soddisfare la richiesta di memoria

D: perché??

- **Quick fit**. Mantiene un insieme di liste separate per le allocazioni di dimensioni più comuni

Strutture per gestire la memoria

Bitmap



In una bitmap la **memoria** viene **divisa** in **unità discrete** (e.g., alcuni KB) e viene allocata una struttura dati con un bit per ognuna di queste unità.
Il bit è 0 se l'unità è libera e 1 se è occupata.

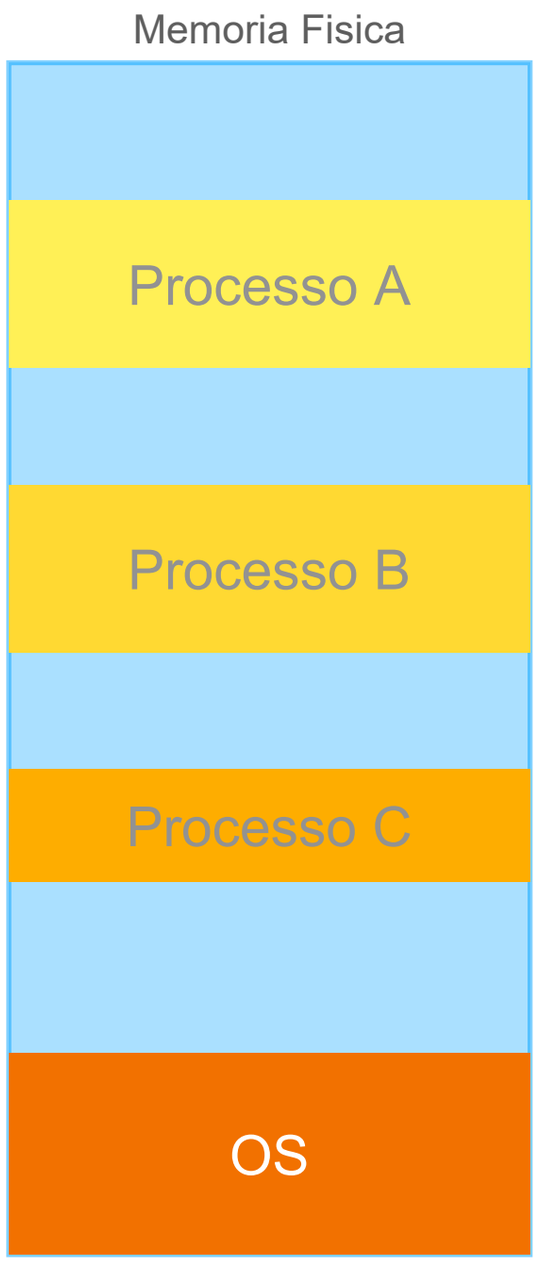
La scelta della dimensione dell'unità di allocazione è importante per **bilanciare granularità** e **occupazione** di memoria da parte della bitmap

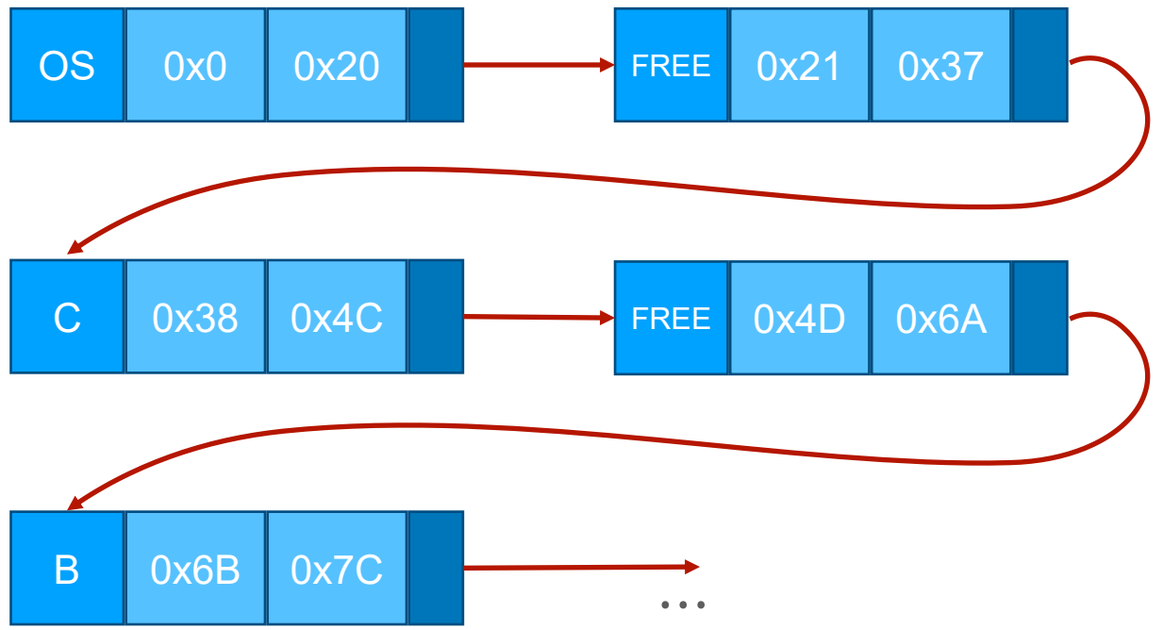
Per soddisfare un processo che richieda k unità di allocazione come spazio serve trovare una sequenza di k zeri consecutivi

Strutture per gestire la memoria

Liste

È possibile tenere una lista della memoria libera e occupata tenendo traccia dell'indirizzo di inizio, quello di fine e del processo che ha possesso di quel range di indirizzi fisici





Alcune osservazioni e domande:

D: cosa devo fare quando si libera un blocco di memoria adiacente ad uno FREE?

- Serve "**unificare**" due nodi consecutivi quando sono entrambi liberi o occupati dallo stesso processo!

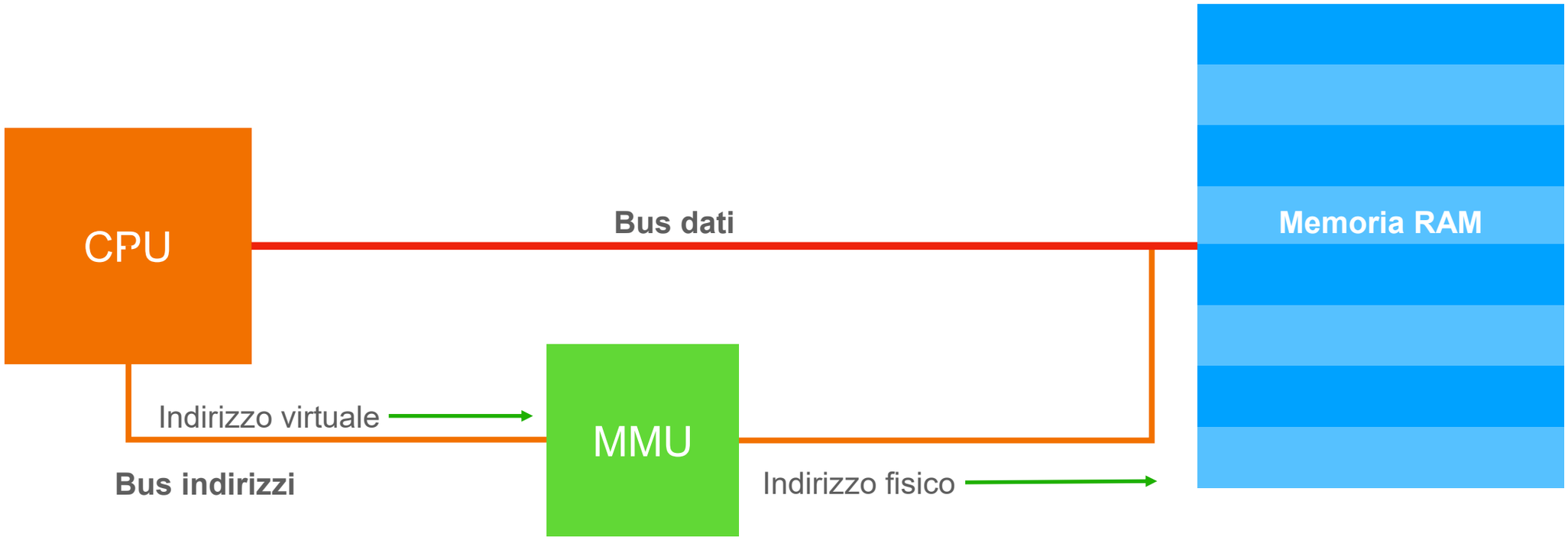
D: cosa devo fare quando alloco un pezzo di un blocco libero?

- Bisogna "spezzare" un nodo se solo una parte della memoria viene (de)allocata

Memoria virtuale

MMU

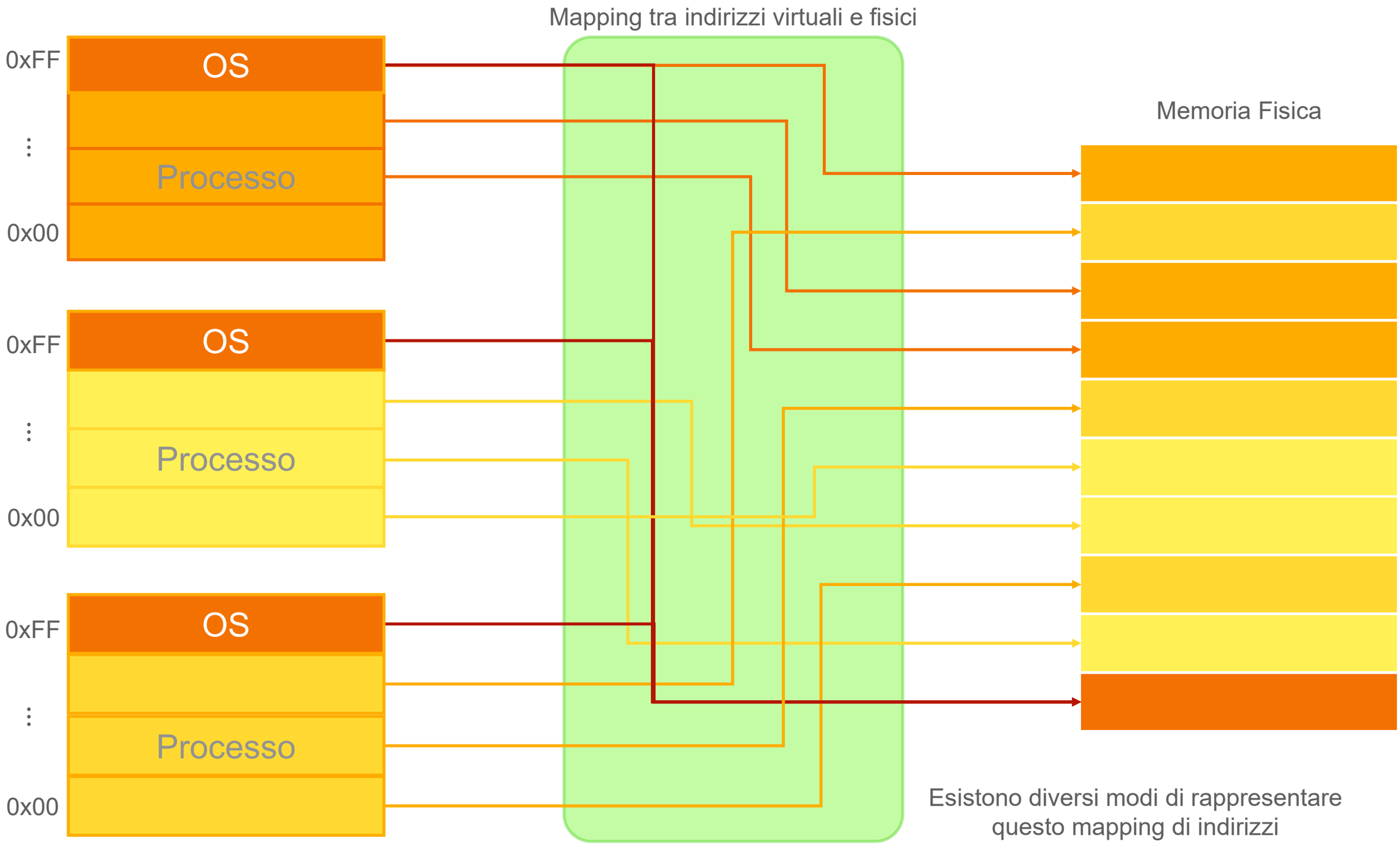
Memory Management Unit



- La MMU si occupa di convertire gli indirizzi che “arrivano” dalla CPU nell’indirizzo di memoria fisico corrispondente
- La MMU fornisce un **mapping** tra **indirizzi virtuali** e i corrispondenti indirizzi **fisici**
- A seconda del tipo di mapping fornito dalla MMU abbiamo diversi modi di astrarre la memoria
- Notate che la MMU è un **dispositivo hardware**

Memoria virtuale

L’illusione che ogni programma abbia la “sua” memoria



Memoria virtuale

Divisione in pagine

- Una idea è quella di dividere l'intera memoria *virtuale* di un processo in unità di dimensione fissata, chiamate **pagine** (**pages**) che possono essere, per esempio, di 4096 bytes (4KB). Altre dimensioni sono ovviamente possibili.

- L'unità corrispondente nella memoria fisica è il **page frame**. I page frame sono solitamente della stessa dimensione delle pagine.

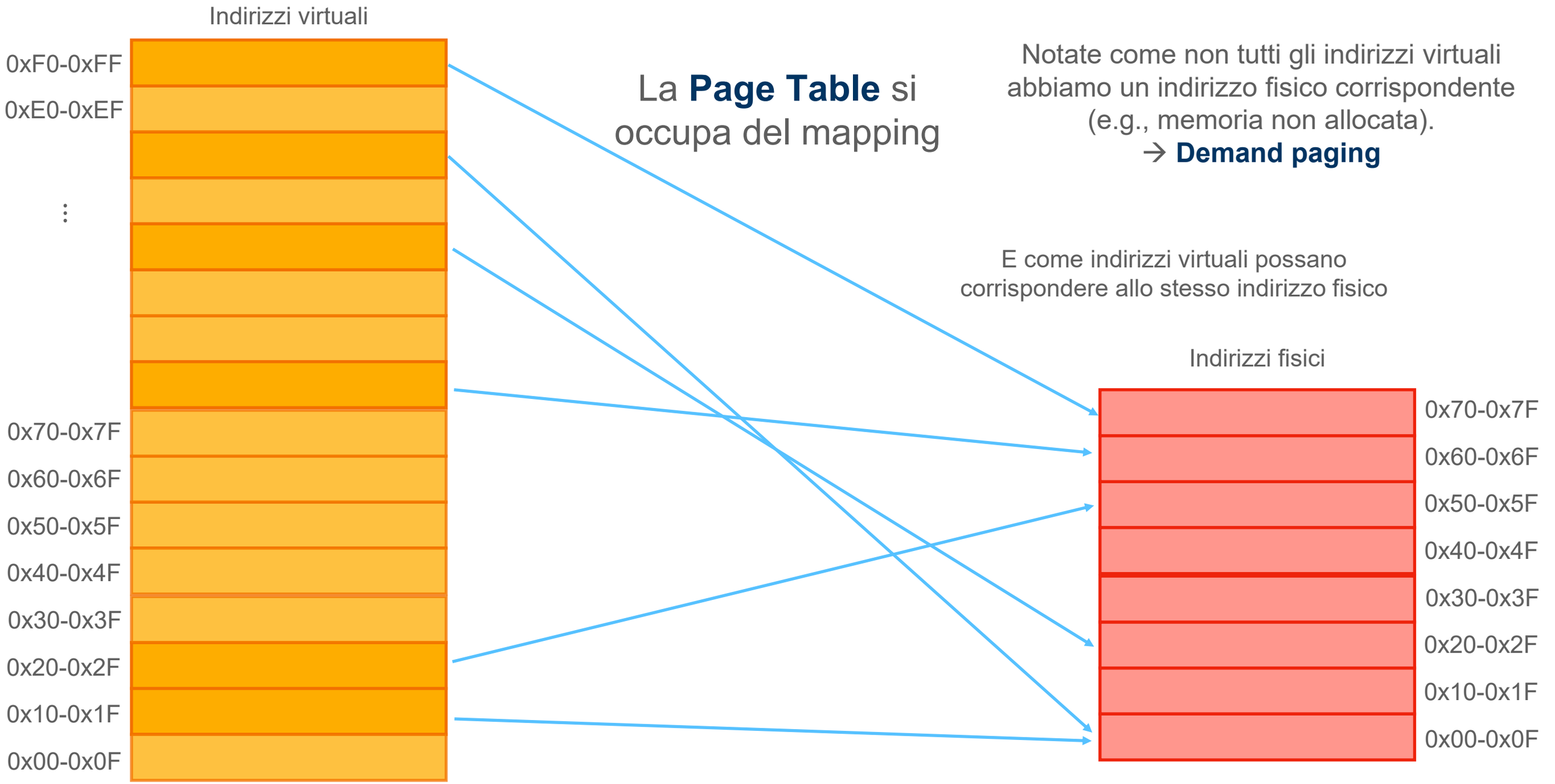
- Il compito della MMU è quello di associare a una pagina il corrispondente page frame (se esiste)

- Questo permette di trasformare gli indirizzi virtuali in indirizzi fisici

D: che vantaggi ci dà allocare blocchi di dimensione fissa?

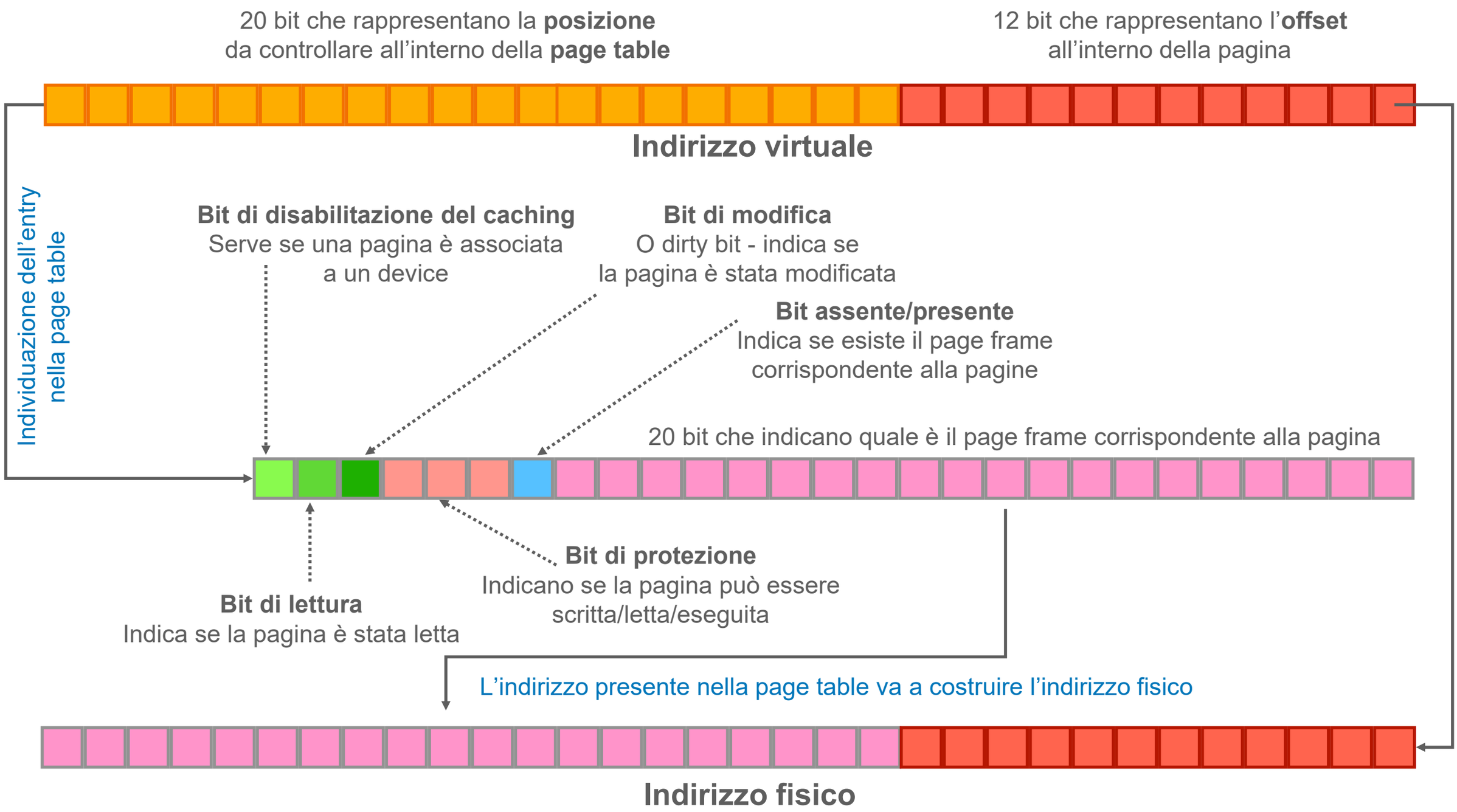
Page Table

La struttura dati che rappresenta il mapping



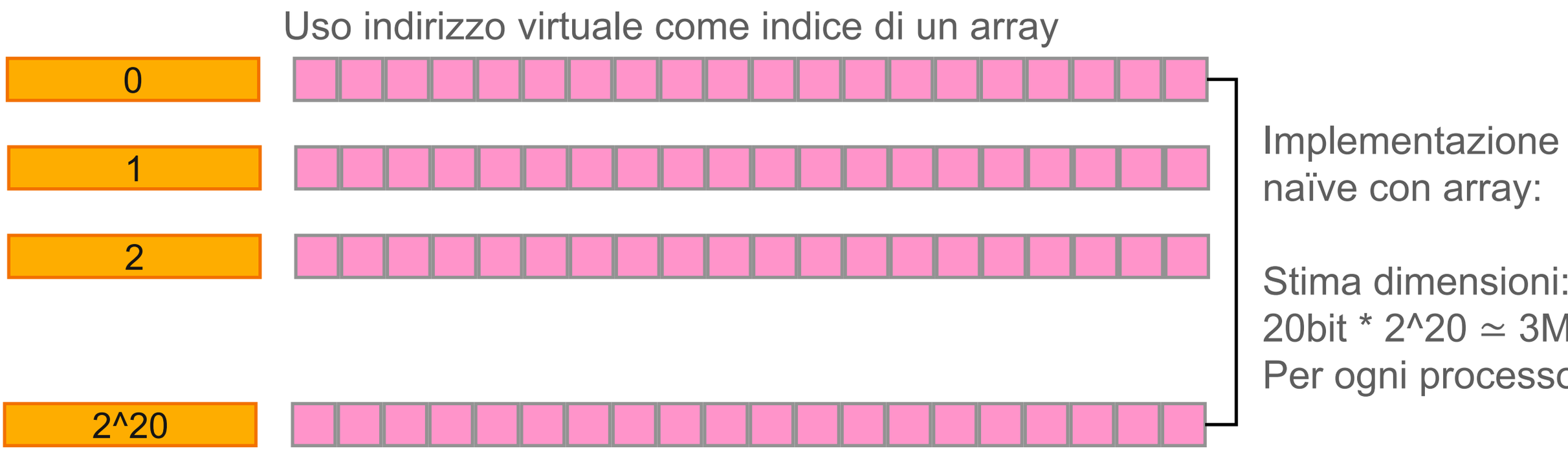
Indirizzi di memoria

Come effettuare il mapping



Risparmiare spazio con gerarchia

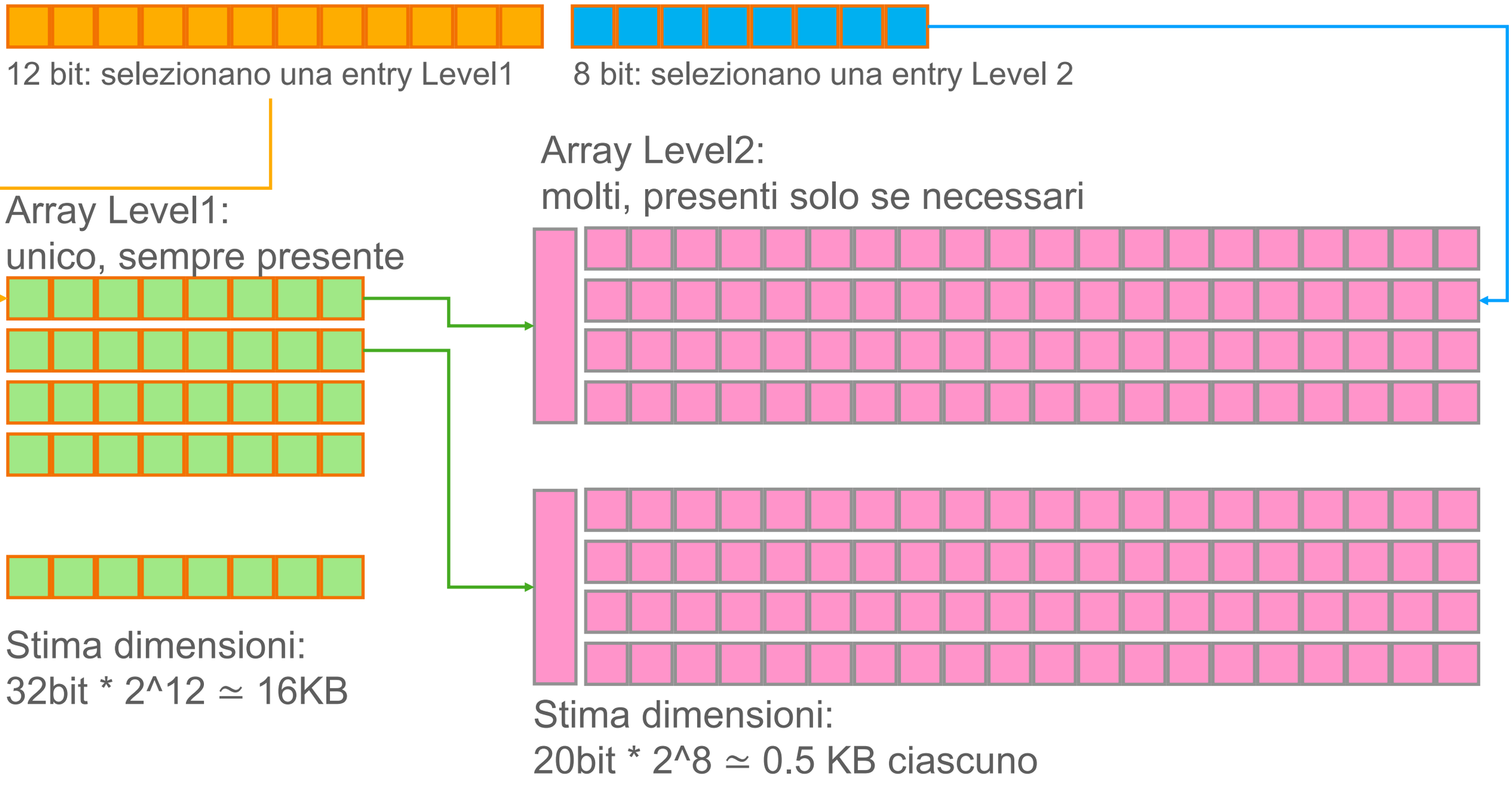
Array naive



Risparmiare spazio con gerarchia

Albero multilivello

Spezzo indirizzo virtuale; le parti indicizzano diversi livelli di un albero



Vantaggi del paging

- **Granularità**: possiamo spostare singole pagine da e verso la memoria di massa, non serve più eliminare dalla memoria un intero processo

- L'**allocazione** è più **semplice**, possiamo allocare **una pagina alla volta** e tenere aggiornato il mapping

- Possiamo permettere a più **processi** di **condividere** la **stessa area** di memoria in sola lettura (e.g., il codice di un programma)

- Possiamo condividere memoria tra processi permettendo la comunicazione (ma con molta attenzione)

- Dato che è necessario convertire ogni indirizzo da virtuale a fisico i processori moderni hanno una cache apposita, chiamata TLB (**Translation Lookaside Buffer**) per alcune delle entry della page table