Programmazione e Architetture (Modulo B)

Lezione 11
Gestione della memoria

Luca Manzoni

Astrarre la memoria

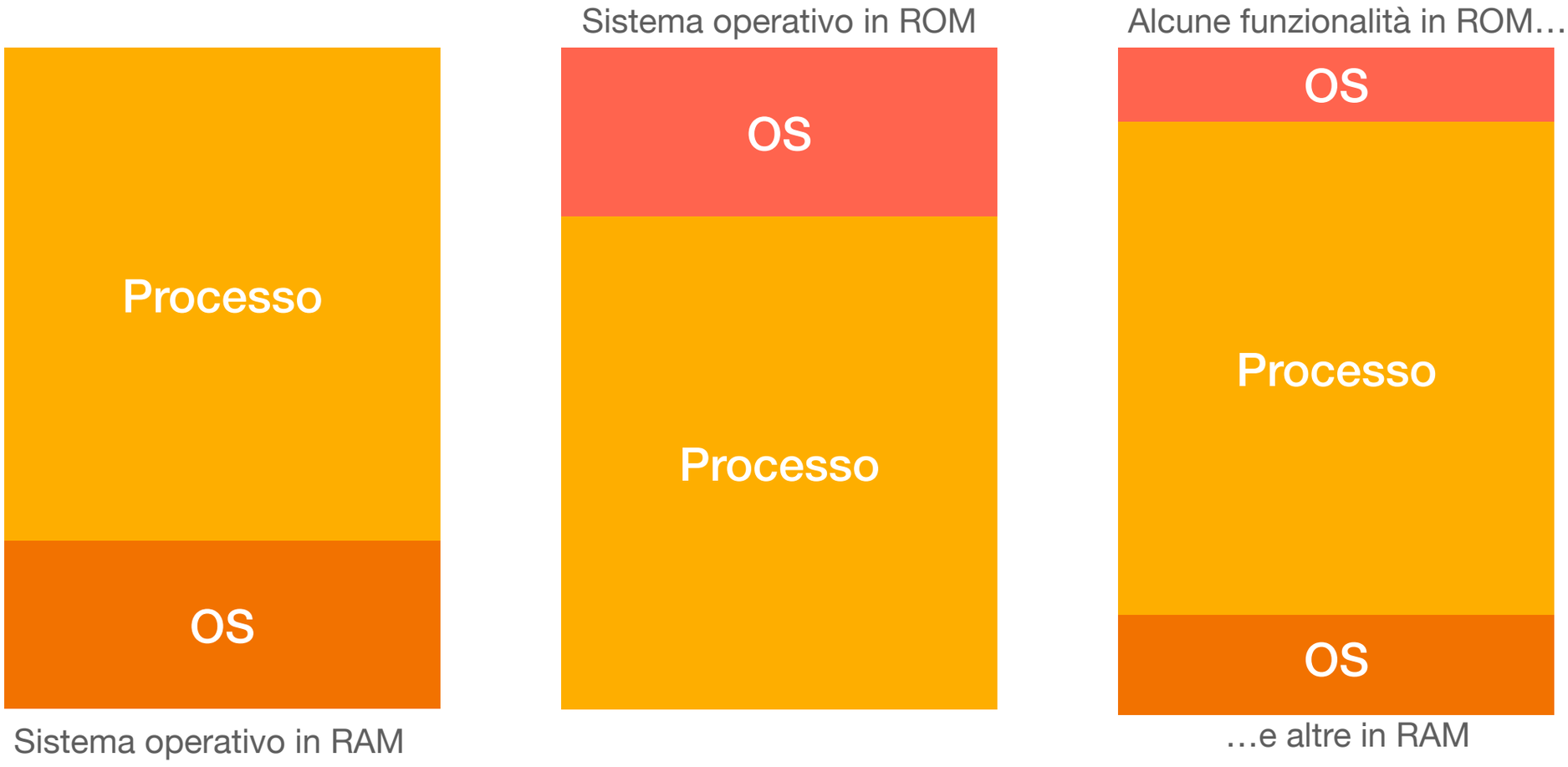
Da nessuna astrazione alla memoria virtuale

- 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

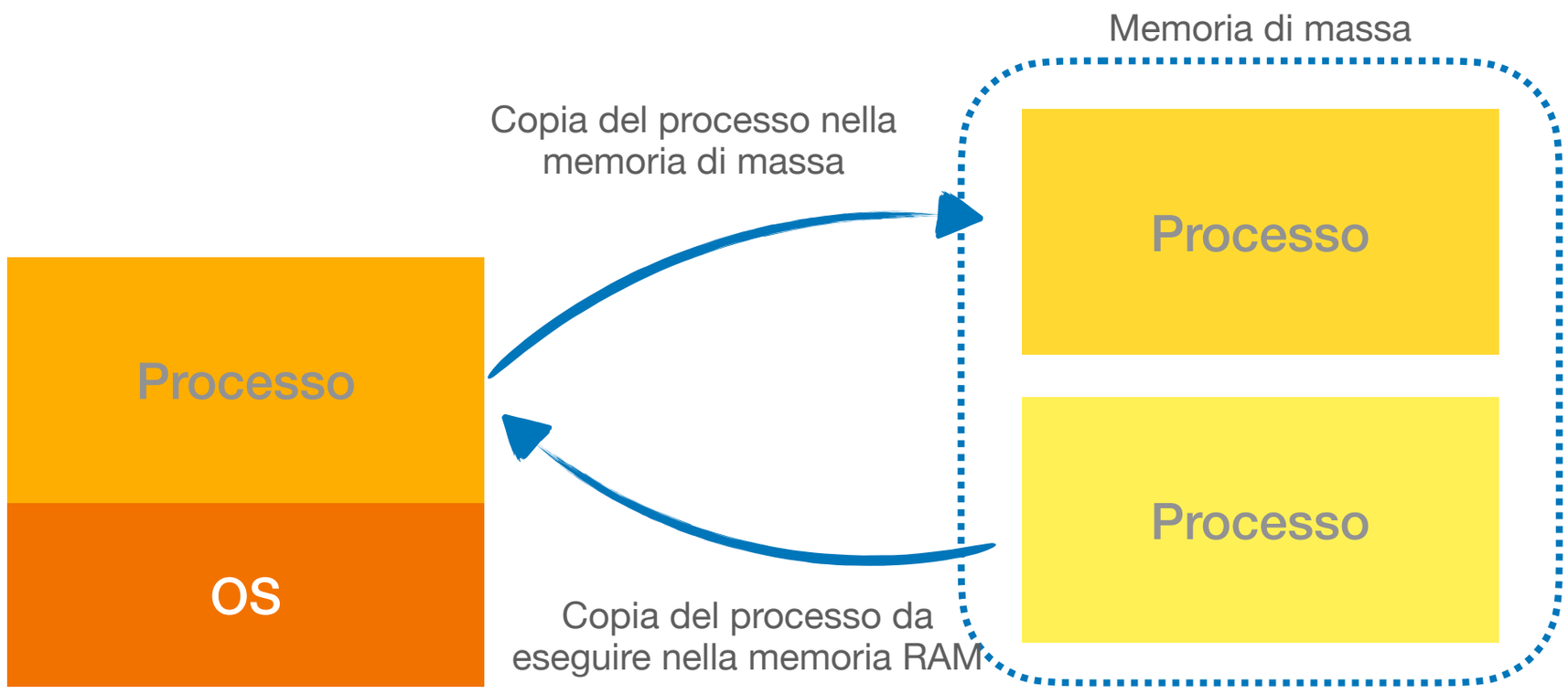
- Un sistema operativo può convivere con un singolo processo anche senza nessuna astrazione della memoria
- Si devono avere delle convenzioni su quali indirizzi sono occupati dal sistema operativo e quali sono usabili dal processo
- Approccio usato nei mainframe (fino agli anni ’60), minicomputer (fino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



Processi multipli

in un sistema senza nessuna astrazione

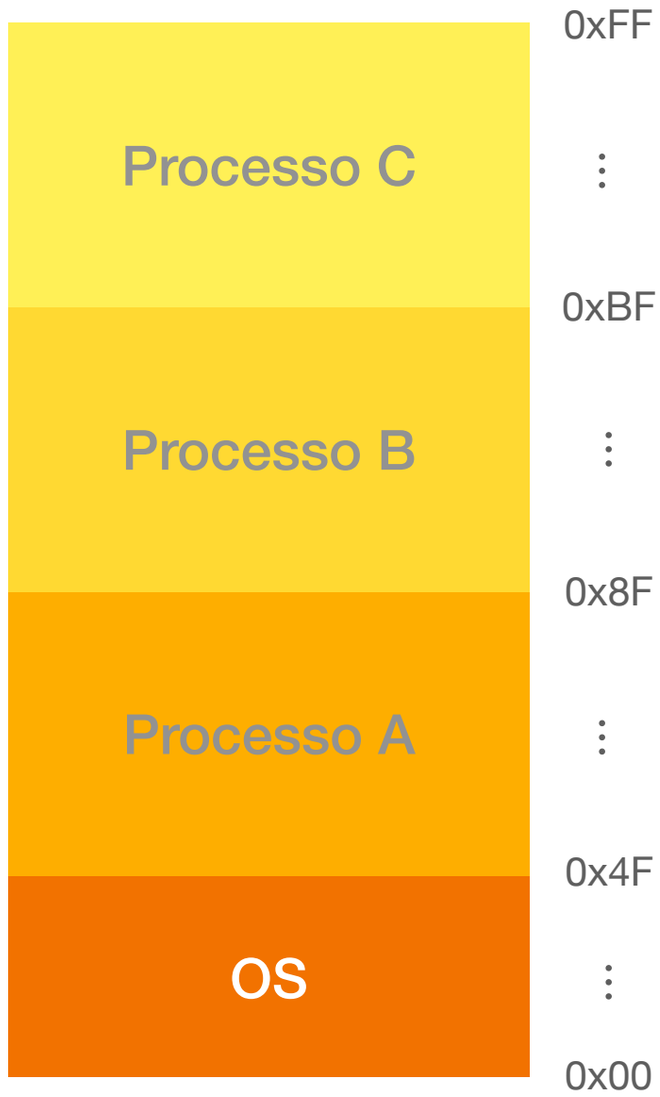
- Anche per sistemi senza nessuna astrazione è possibile gestire più programmi
- Quando un processo deve uscire dallo stato “running”, tutta la sua memoria viene copiata nella memoria di massa…
- …e l’intera memoria del prossimo processo che deve eseguire è copiata dalla memoria di massa alla memoria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

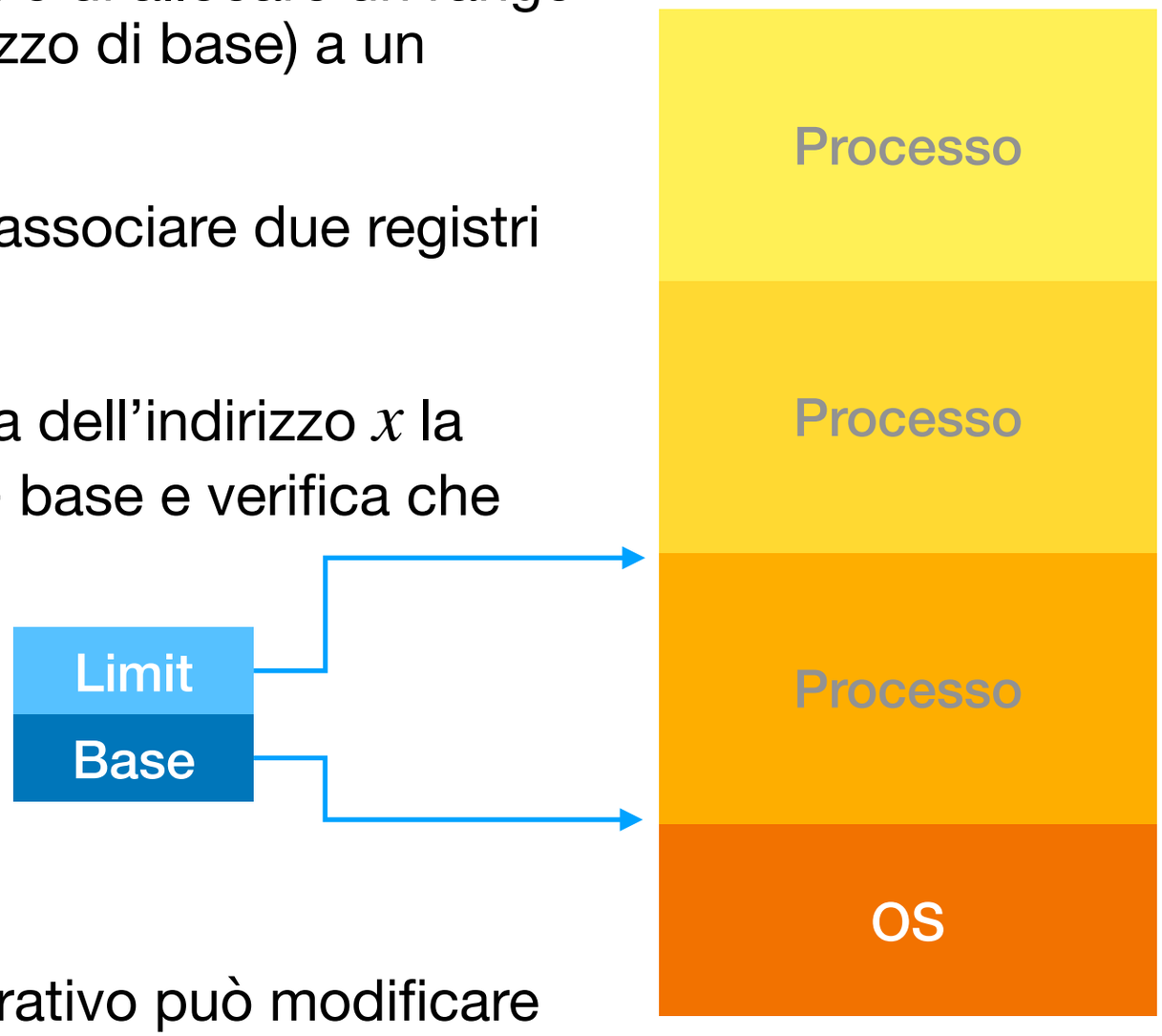
Come nascondere gli indirizzi fisici ai processi

- 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 MMU 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



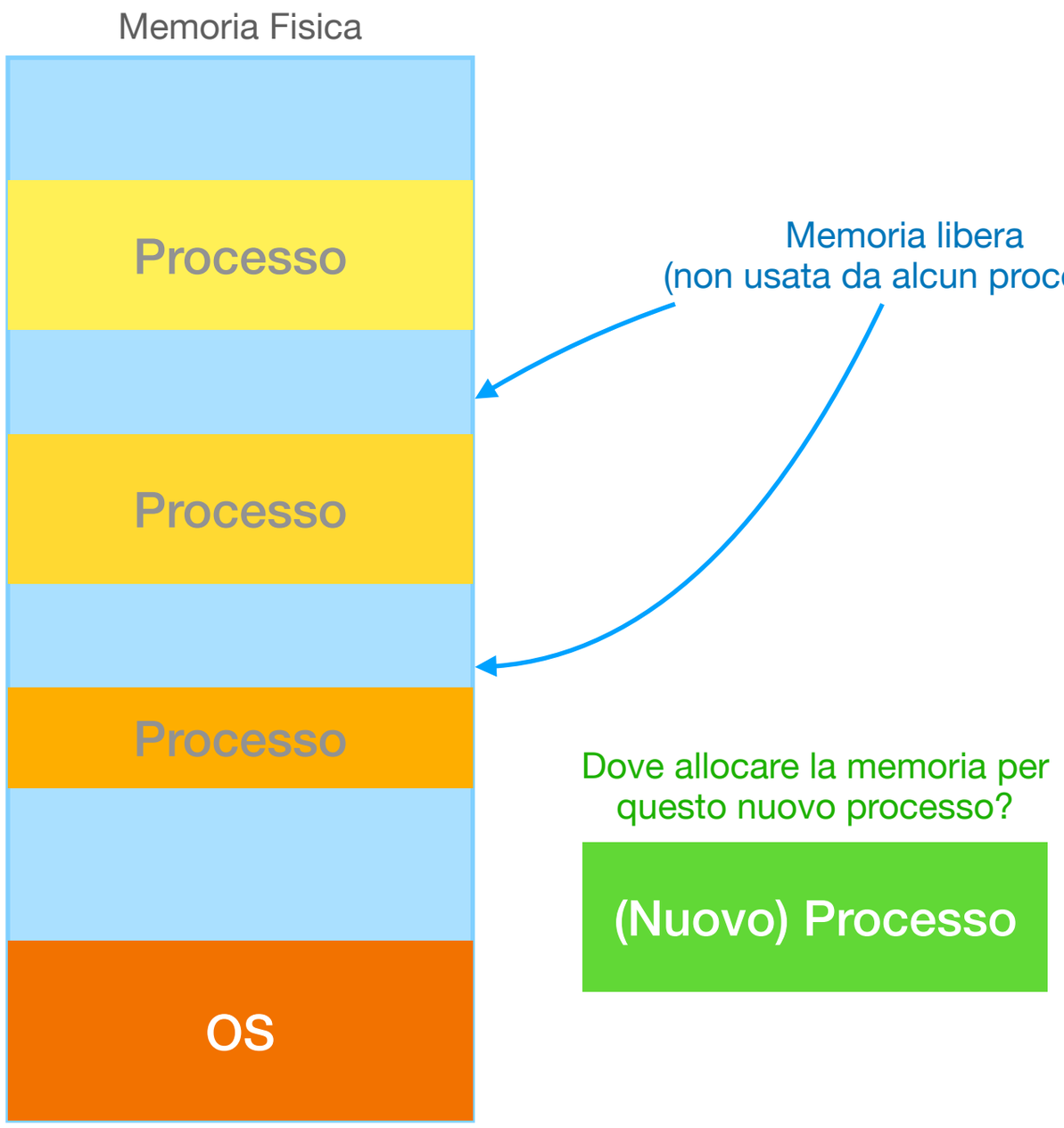
Swapping

Usare più memoria di quella disponibile

- 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

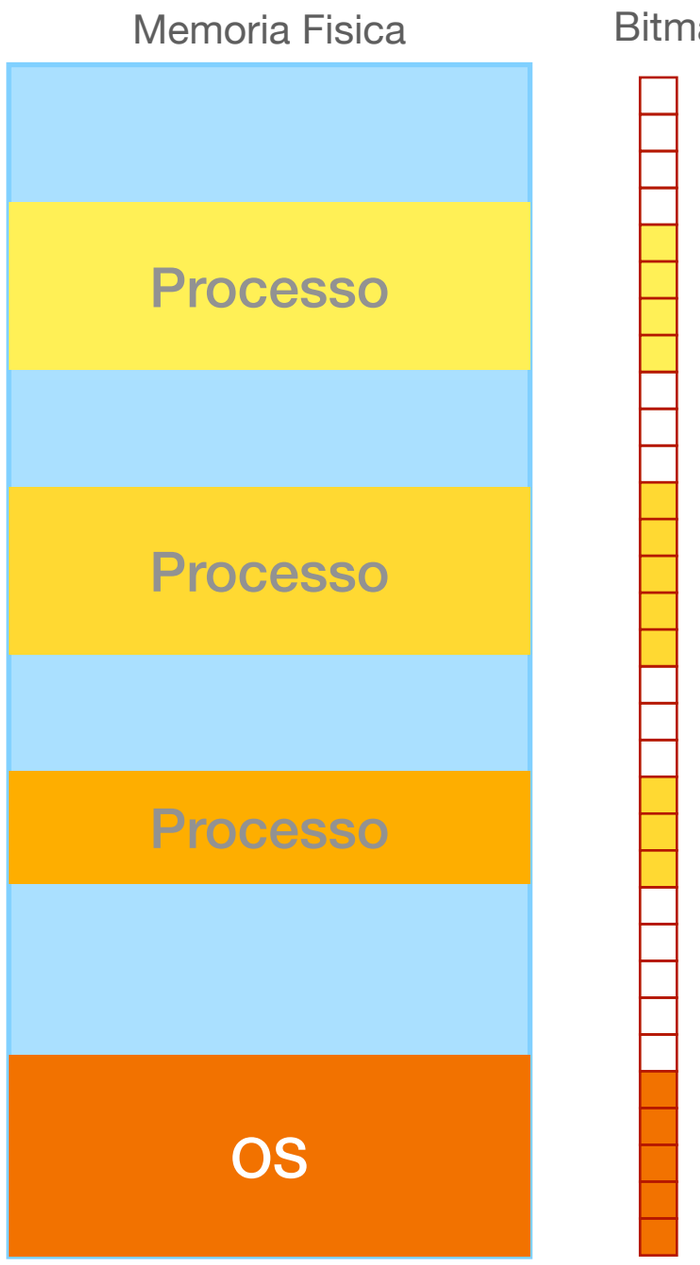
Decidere dove posizionare i processi



- 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

Strutture per gestire la memoria

Bitmap



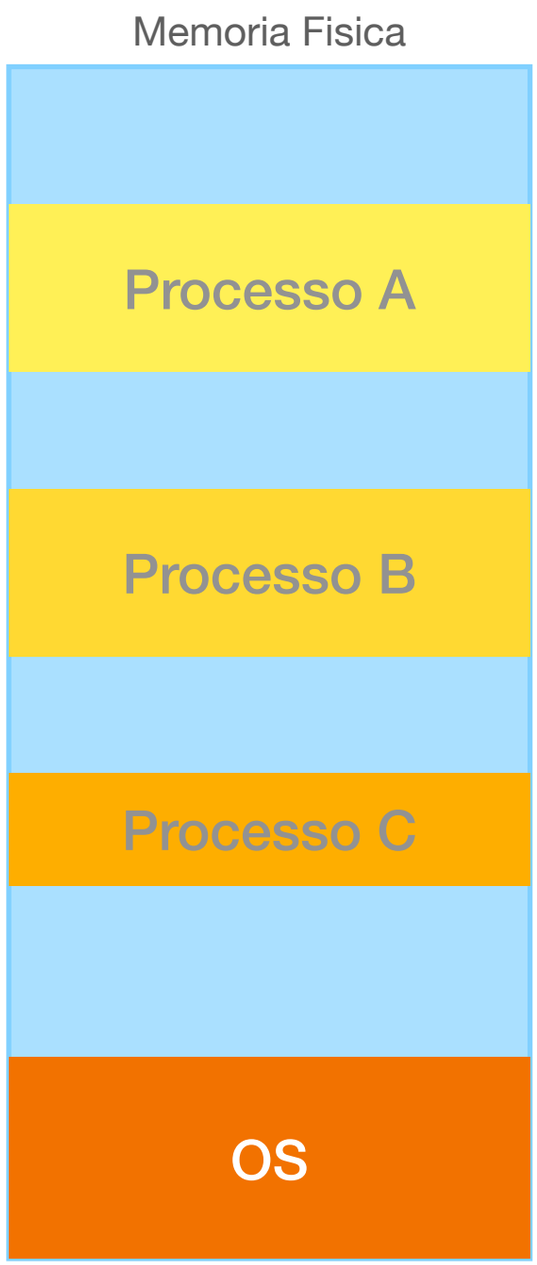
In una bitmap la memoria viene divisa in unità discrete (e.g., alcuni KB) e viene allocata una struttura dati con un bit per ognuna di queste unità.
Il bit è 0 se l'unità è libera e 1 se è occupata.

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

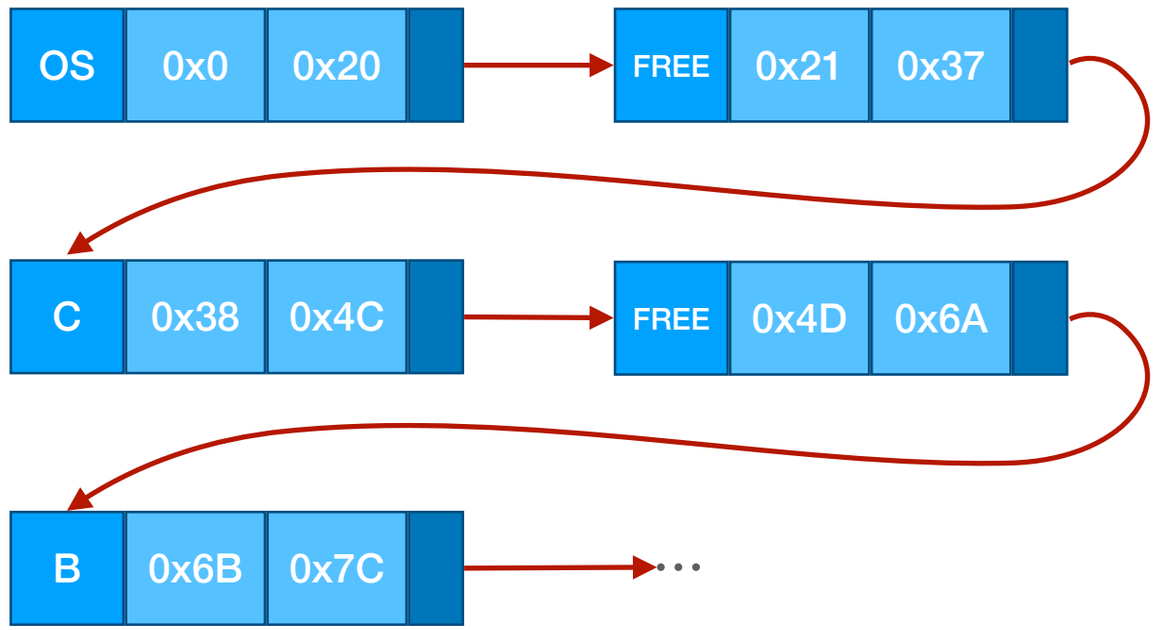
Per soddisfare un processo che richieda k unità di allocazione come spazio serve trovare una sequenza di k zeri consecutivi

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:

- Serve "unificare" due nodi consecutivi quando sono entrambi liberi o occupati dallo stesso processo
- Potrebbe essere necessario "spezzare" un nodo se solo una parte della memoria viene deallocata

Allocazione della memoria

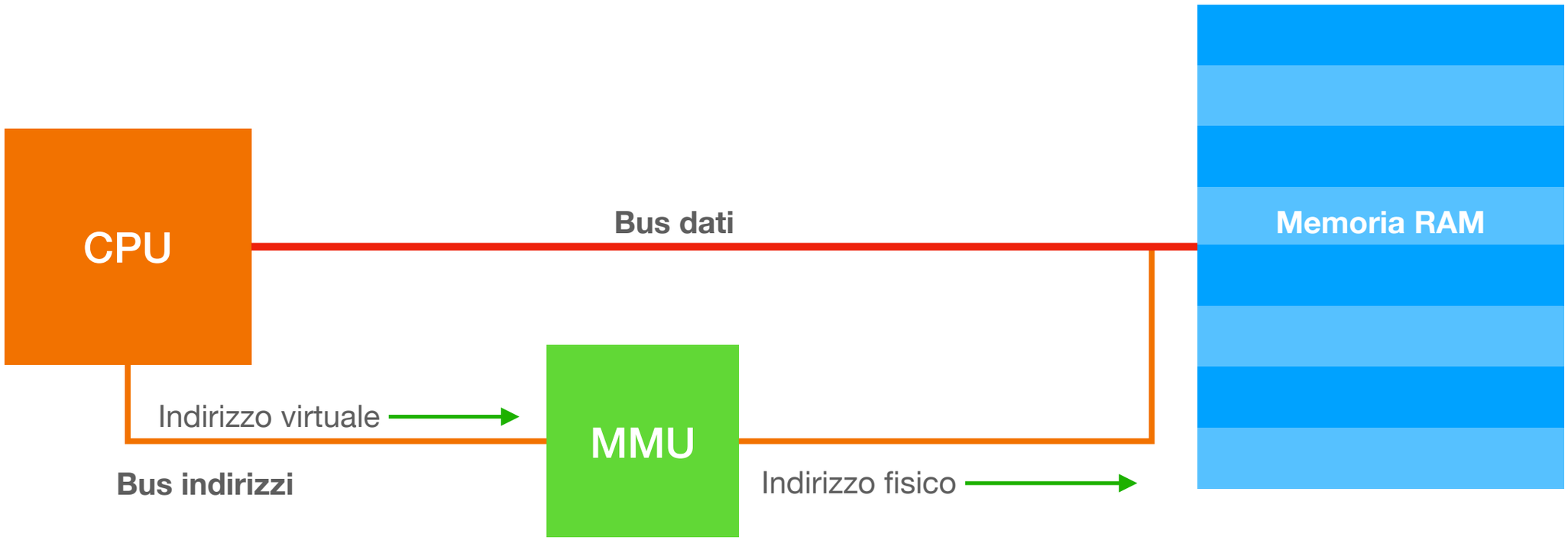
Possibili algoritmi

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

Memoria virtuale

MMU

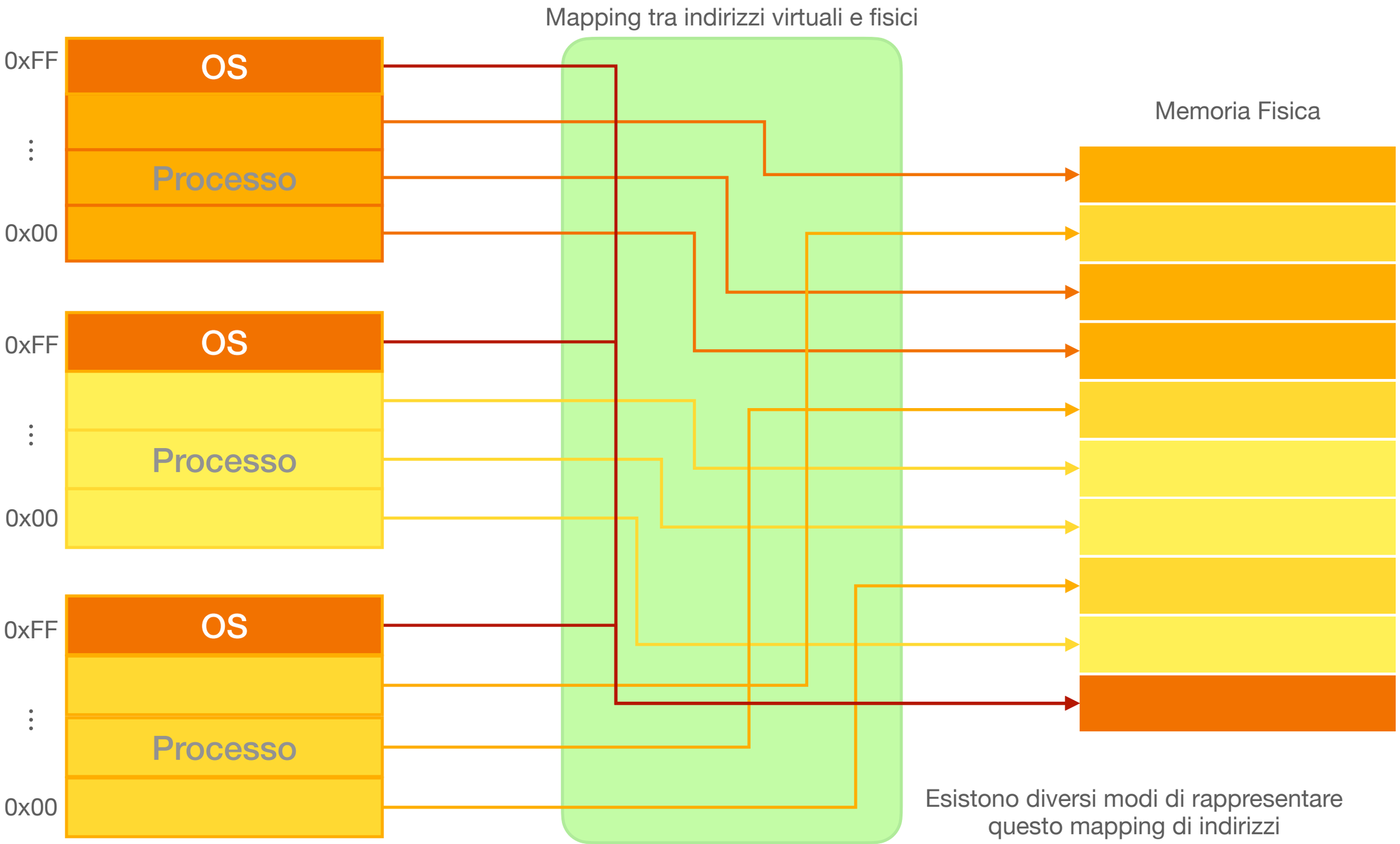
Memory Management Unit



- 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.

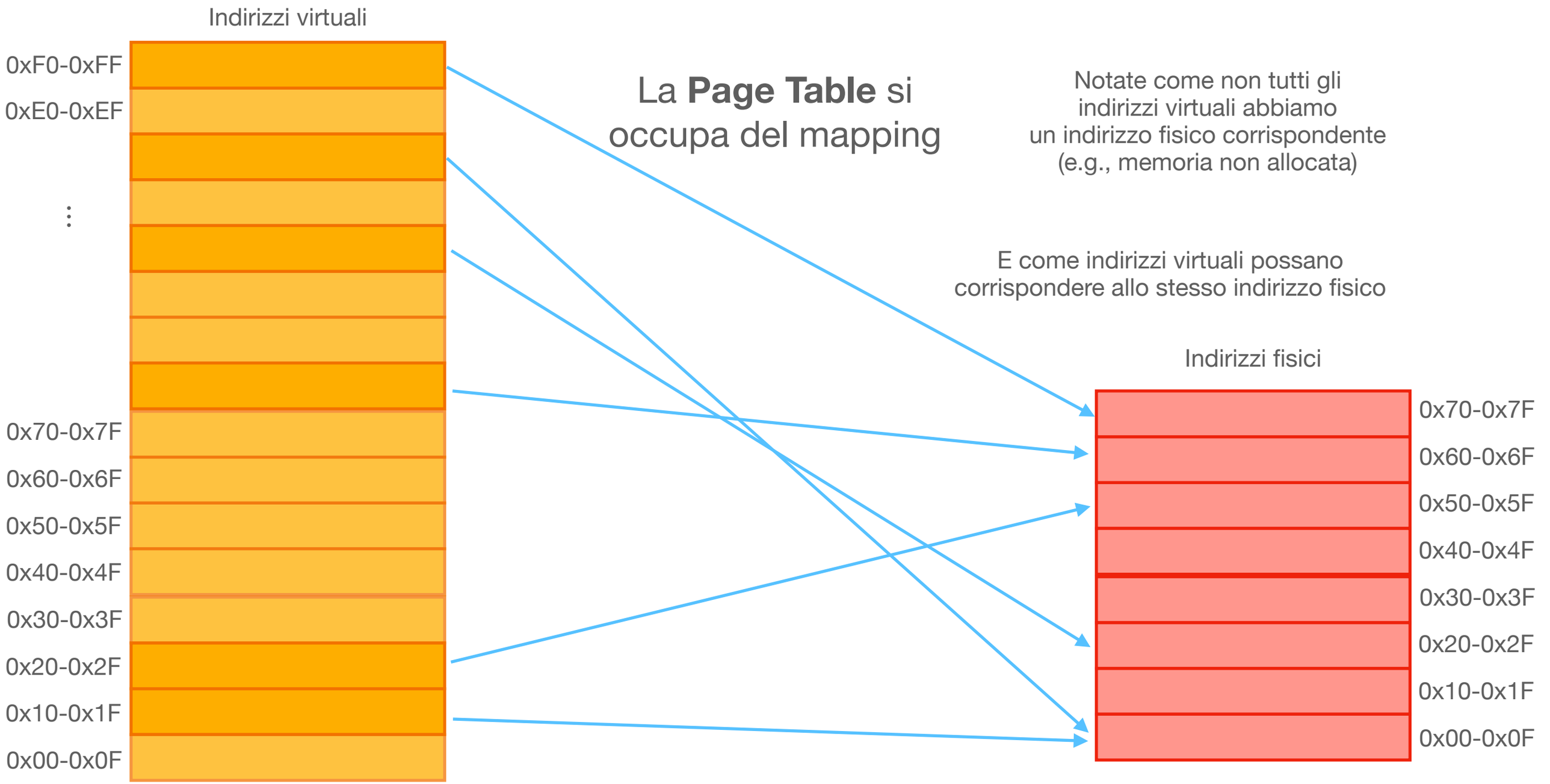
- L'unità corrispondente nella memoria fisica è il **page frame**. I page frame sono solitamente della stessa dimensione delle pagine.

- 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

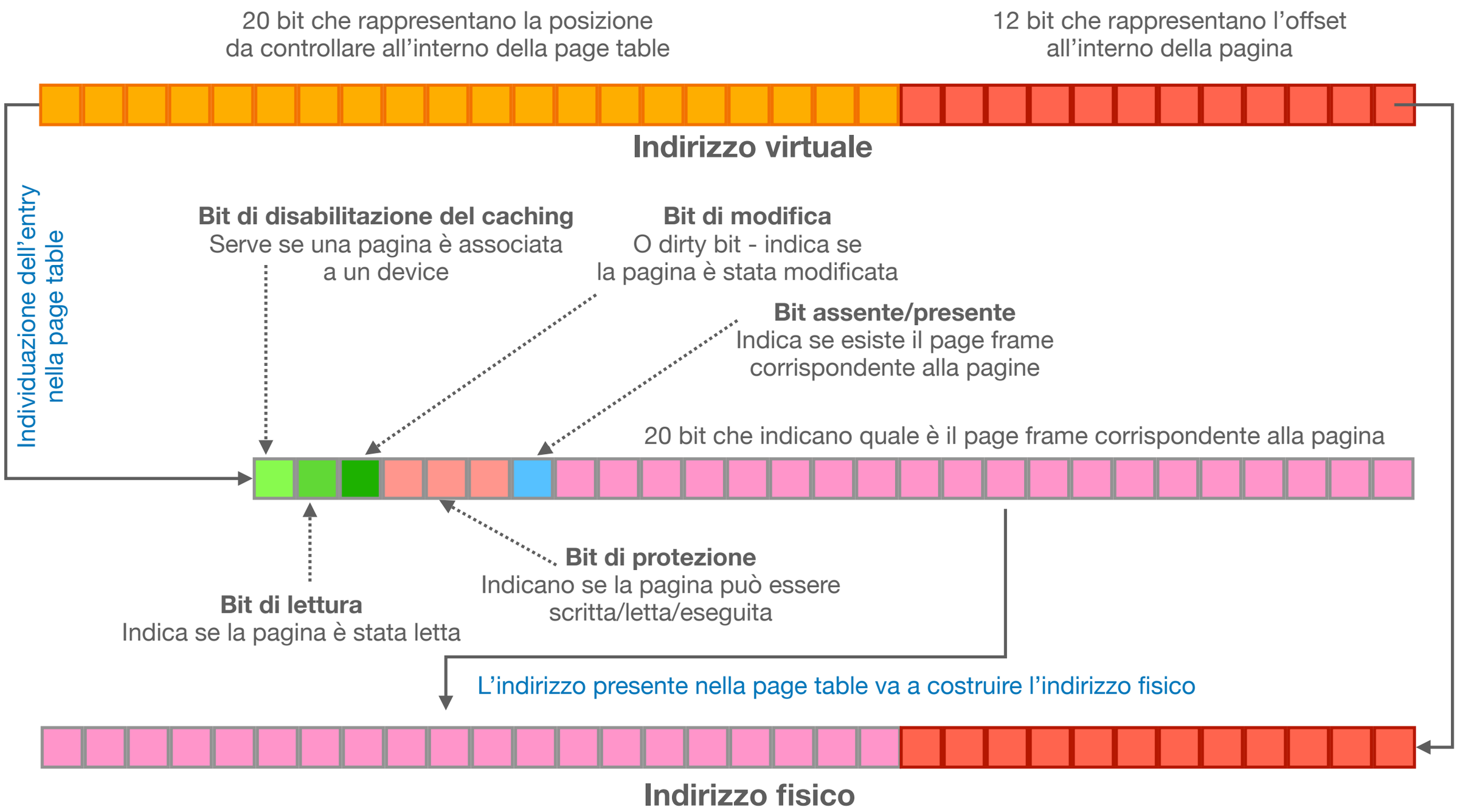
Page Table

La struttura dati che rappresenta il mapping



Indirizzi di memoria

Come effettuare il mapping



Vantaggi del paging

- 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, come vedremo in futuro, ci sono molte cose da tenere in considerazione quando la stessa memoria viene acceduta in modo concorrente)

- 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