

# FONDAMENTI DI INFORMATICA

Ing. Davide PIERATTONI

Facoltà di Ingegneria  
Università degli Studi di Udine

## Gestione della memoria centrale

# Nota di Copyright

Questo insieme di trasparenze (detto nel seguito slide) è protetto dalle leggi sul copyright e dalle disposizioni dei trattati internazionali. Il titolo ed i copyright relativi alle slides (ivi inclusi, ma non limitatamente, ogni immagine, fotografia, animazione, video, audio, musica e testo) sono di proprietà degli autori prof. Pier Luca Montessoro e ing. Davide Pierattoni, Università degli Studi di Udine.

Le slide possono essere riprodotte ed utilizzate liberamente dagli istituti di ricerca, scolastici ed universitari afferenti al Ministero della Pubblica Istruzione e al Ministero dell'Università e Ricerca Scientifica e Tecnologica, per scopi istituzionali, non a fine di lucro. In tal caso non è richiesta alcuna autorizzazione.

Ogni altro utilizzo o riproduzione (ivi incluse, ma non limitatamente, le riproduzioni su supporti magnetici, su reti di calcolatori e stampe) in toto o in parte è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte degli autori.

L'informazione contenuta in queste slide è ritenuta essere accurata alla data della pubblicazione. Essa è fornita per scopi meramente didattici e non per essere utilizzata in progetti di impianti, prodotti, reti, ecc. In ogni caso essa è soggetta a cambiamenti senza preavviso. L'autore non assume alcuna responsabilità per il contenuto di queste slide (ivi incluse, ma non limitatamente, la correttezza, completezza, applicabilità, aggiornamento dell'informazione).

In ogni caso non può essere dichiarata conformità all'informazione contenuta in queste slide.

In ogni caso questa nota di copyright e il suo richiamo in calce ad ogni slide non devono mai essere rimossi e devono essere riportati anche in utilizzi parziali.

# Gestione della memoria per sistemi multiprogrammati

- Diverse tecniche:
    - partizioni fisse
    - partizioni variabili
    - segmentazione
    - paginazione
    - segmentazione + paginazione
- } memoria virtuale

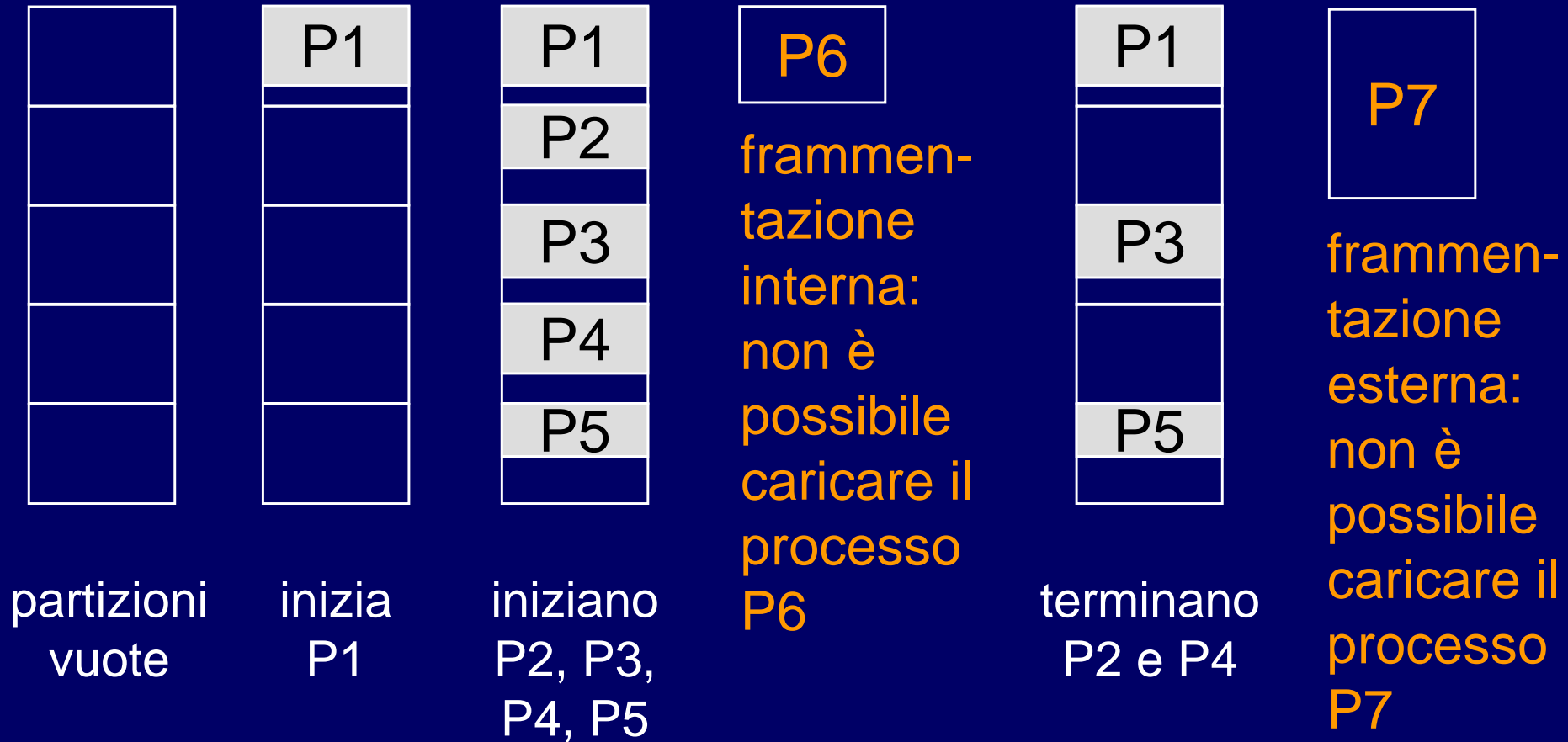
## Obiettivi

- Massimizzare il numero di processi che possono utilizzare la memoria
- Minimizzare lo spazio di memoria inutilizzato
  - frammentazione interna: spazio inutilizzato all'interno di una partizione
  - frammentazione esterna: partizioni non utilizzate perché troppo piccole
- Semplicità ed efficienza

## Partizioni fisse

- La memoria fisica viene suddivisa in partizioni predefinite
- Quando un programma viene caricato in una partizione, gli indirizzi vengono ricalcolati in funzione dell'indirizzo di inizio della partizione
- Problemi:
  - quante partizioni, quanto grandi?
  - frammentazione interna ed esterna

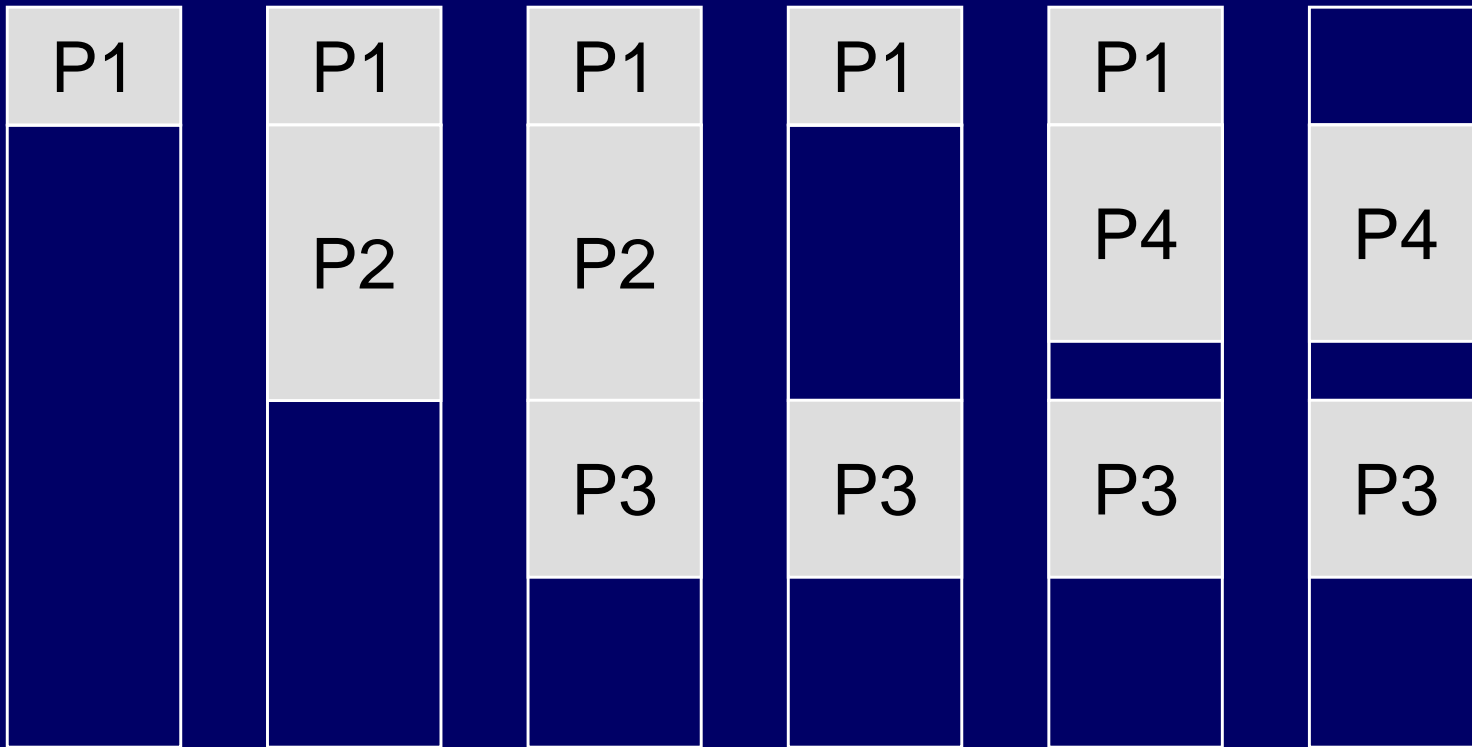
# Partizioni fisse



## Partizioni variabili

- Le partizioni vengono create man mano che i processi richiedono memoria
- Gli indirizzi vengono ricalcolati come nel caso delle partizioni fisse
- Problemi:
  - scelta della partizione da utilizzare
  - frammentazione / ricompattazione

# Partizioni variabili



P2  
termina

P1  
termina



la memoria è  
frammentata,  
non è  
possibile  
caricare il  
processo P5

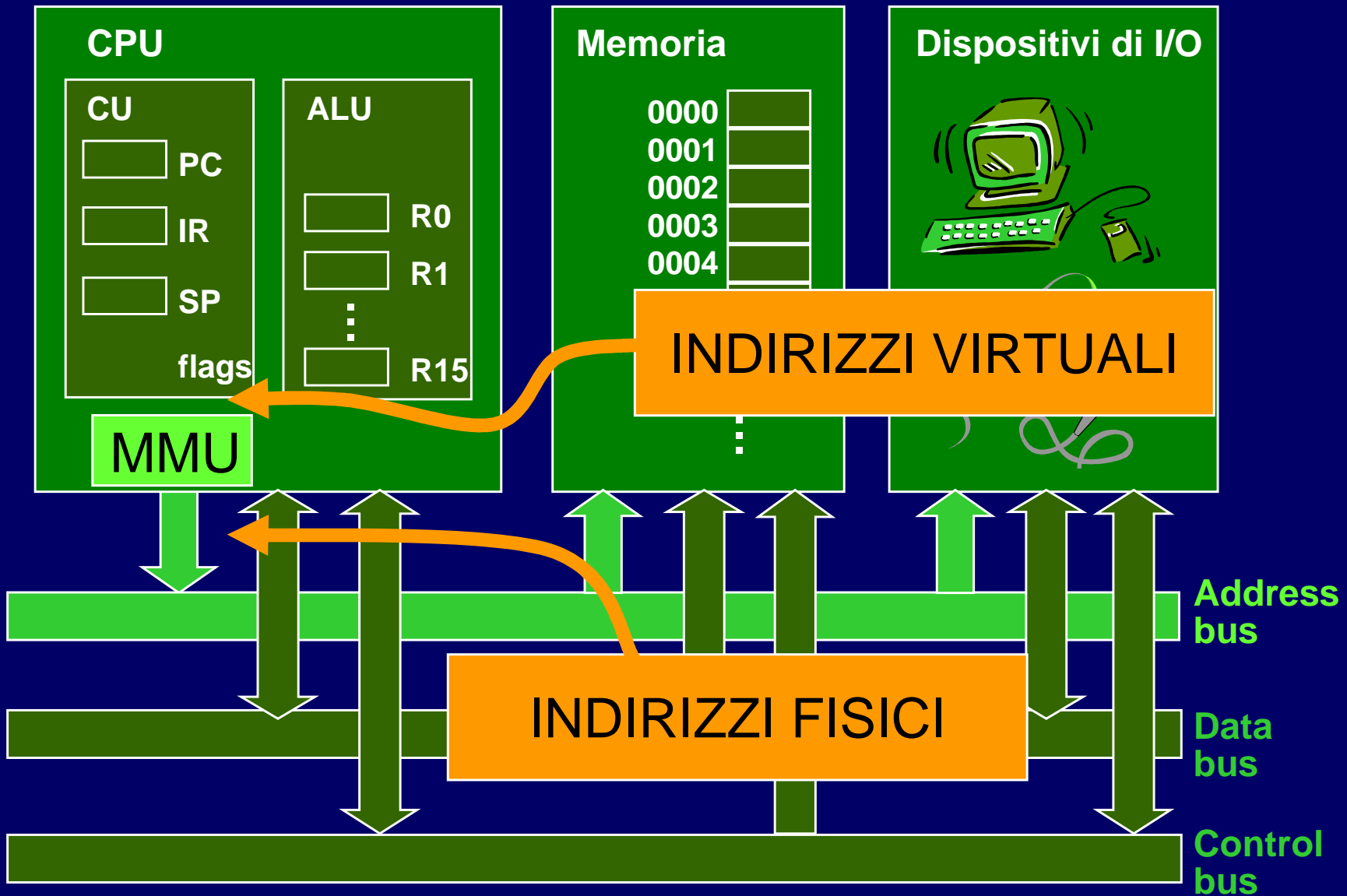
# Memoria virtuale

- 2 aspetti:
  - estensione della memoria RAM utilizzabile dalla CPU oltre la dimensione reale della memoria fisica disponibile (mediante page file)
  - *mapping* tra gli indirizzi virtuali di ciascun processo in ambiente multitasking: tutti i processi usano lo stesso range di indirizzi, ma per ciascuno la memoria realmente indirizzata è diversa

# Memoria virtuale e indirizzi

- **Spazio di indirizzamento virtuale:**
  - determinato dal parallelismo della CPU (es. 32 bit → indirizzi da 0 a 4G)
- **Spazio di indirizzamento reale (in cui vengono tradotti gli indirizzi virtuali):**
  - determinato dall'architettura (numero di fili dell'address bus) e dalla quantità di RAM
- **Massima quantità di parole di memoria utilizzabili contemporaneamente**
  - determinata dal numero di processi attivi e dalla quantità di RAM

# Memory Management Unit (MMU)

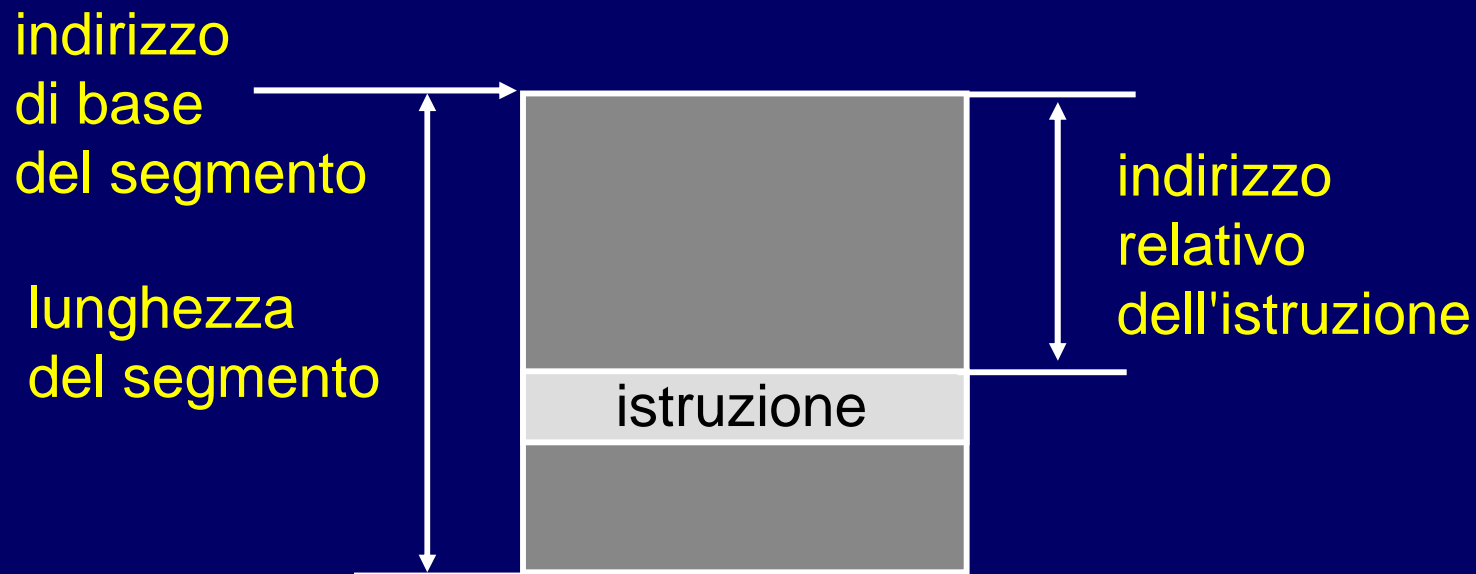


# Segmenti e pagine

- **Segmento**
  - raggruppamento logico di informazioni (subroutine, area dati, ecc.) di lunghezza variabile
- **Page frame**
  - partizione della memoria fisica di dimensione fissa e uguale per tutti i page frame
- **Pagina**
  - partizione della memoria virtuale del processo di dimensione pari ad un page frame

# Segmentazione

- Dati e programmi possono essere suddivisi in segmenti, contenenti parti che possono essere indirizzate relativamente ad un indirizzo di base.



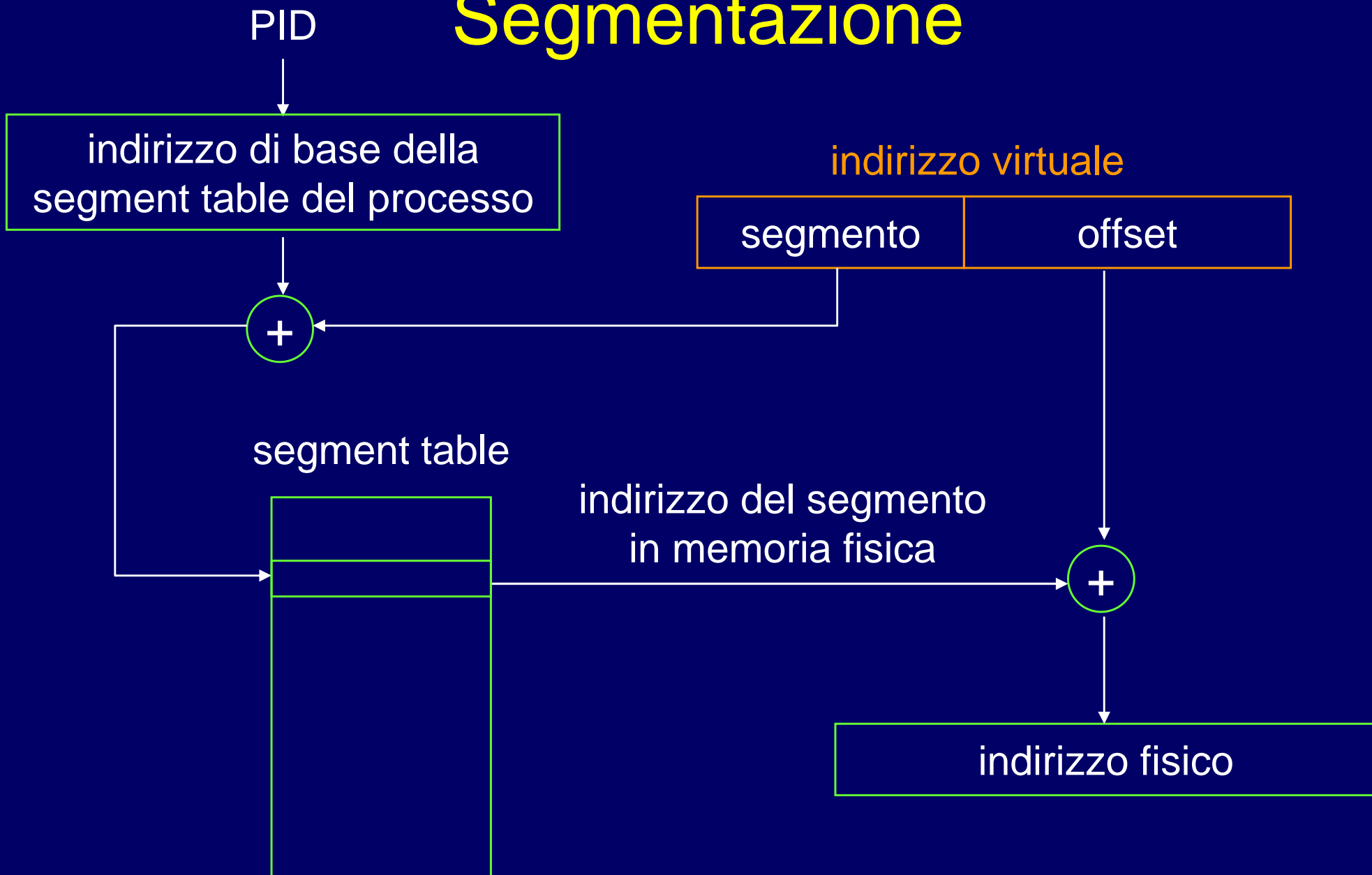
# Segmentazione

- Ogni segmento corrisponde ad una entità logica ben definita (Esempi: codice, dati, funzioni di libreria, sistema operativo o parte di esso)
- Le istruzioni e/o le variabili contenute nel segmento sono identificate tramite il loro indirizzo relativo all'indirizzo di base
- NOTA: Per l'esecuzione di un programma non è necessario che siano presenti in memoria centrale tutti i segmenti: possono essere caricati quando serve

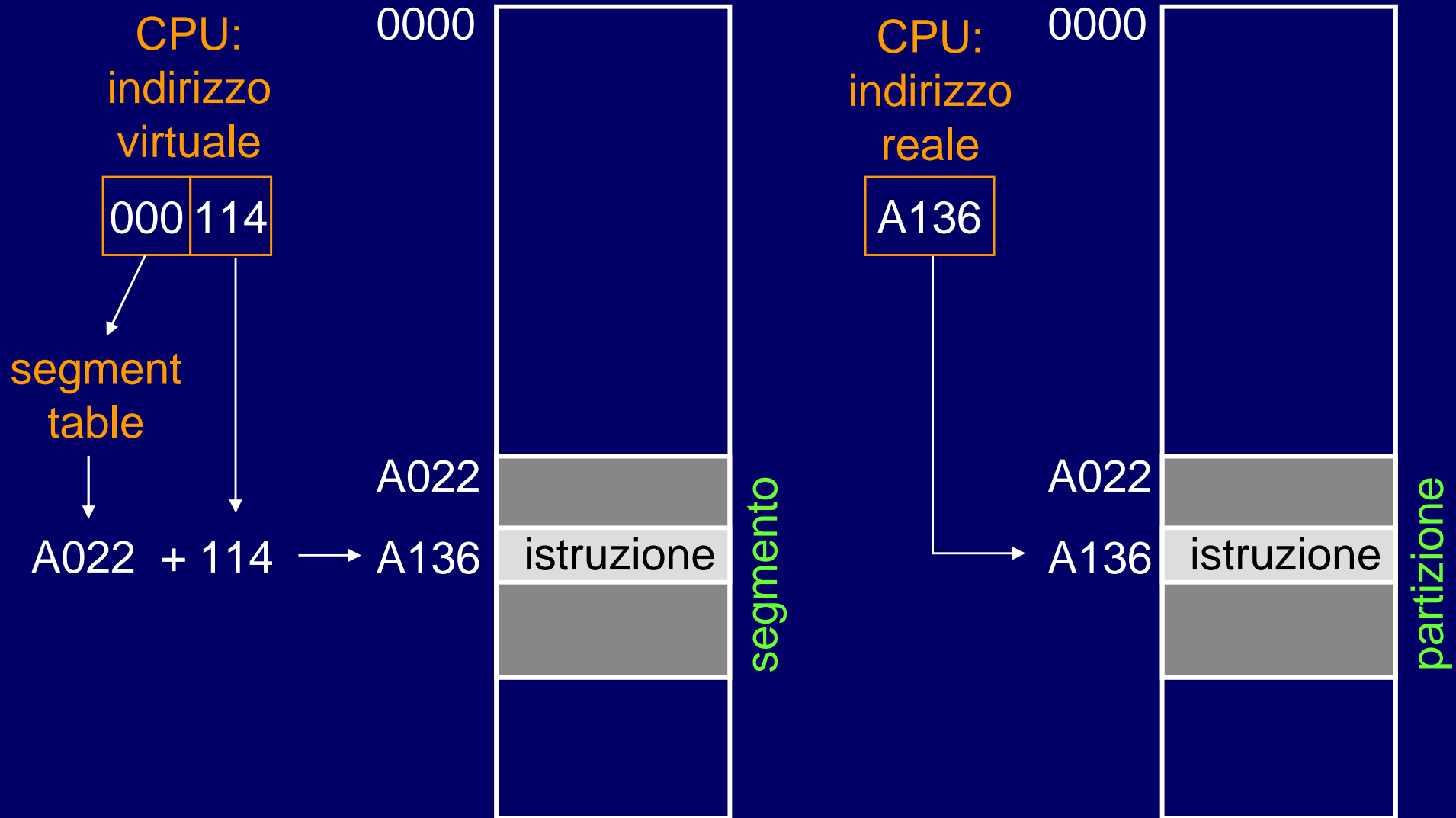
## Vantaggi

- Possibilità di rilocare i segmenti: gli indirizzi sono relativi all'indirizzo di base, e quest'ultimo viene definito al momento del caricamento del segmento in memoria
- L'hardware che gestisce la memoria virtuale provvede alla traduzione degli indirizzi relativi in indirizzi fisici di memoria
- Ad ogni segmento possono essere associate delle protezioni (es. sola lettura o esecuzione) in base alla sua funzione

# Segmentazione



# Segmentazione vs. partizioni variabili



# Paginazione

- Risolve il problema della frammentazione esterna, esistente nella segmentazione al pari del sistema a partizioni variabili, unificando la dimensione dei blocchi di codice (pagine) che vengono dinamicamente caricati in memoria (page frame)
- Il programma vede ancora lo spazio di indirizzamento virtuale contiguo all'interno dei segmenti, ma in realtà è immagazzinato in pagine che non sono contigue

# Paginazione

- Pagine e page frame hanno dimensioni pari a  $2^n$ 
  - come nella segmentazione, l'indirizzo viene calcolato sommando l'offset all'indirizzo di base della pagina, ma questo avrà gli ultimi  $n$  bit a zero, quindi sarà sufficiente accostare i bit

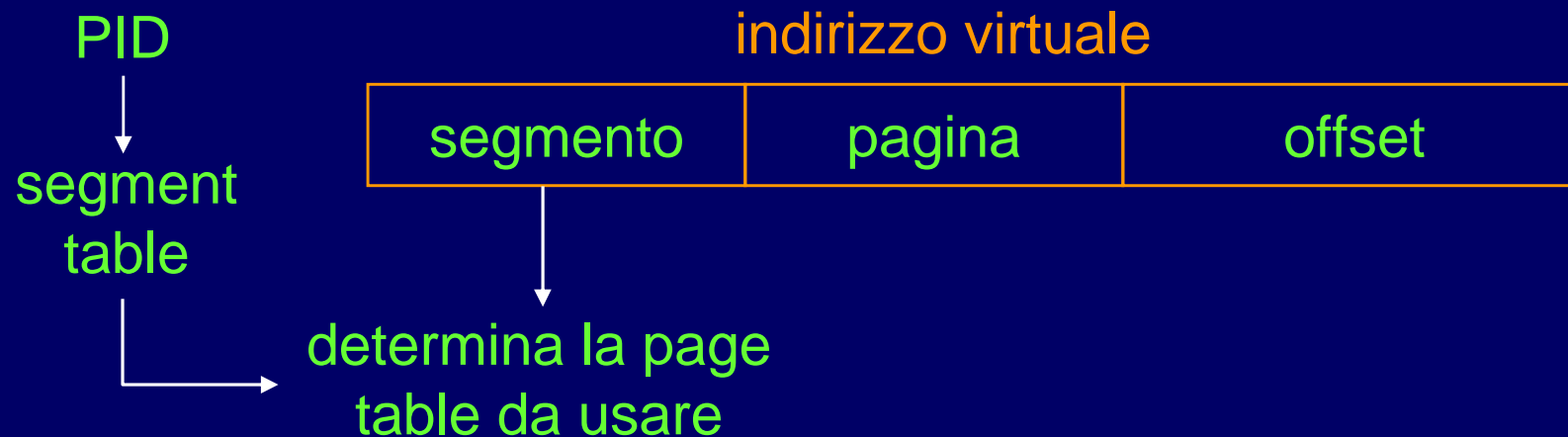
$$\begin{array}{r} 1011110000000000 + \\ \quad \quad \quad 001001011 \\ \hline 101111001001011 \end{array}$$

# Paginazione

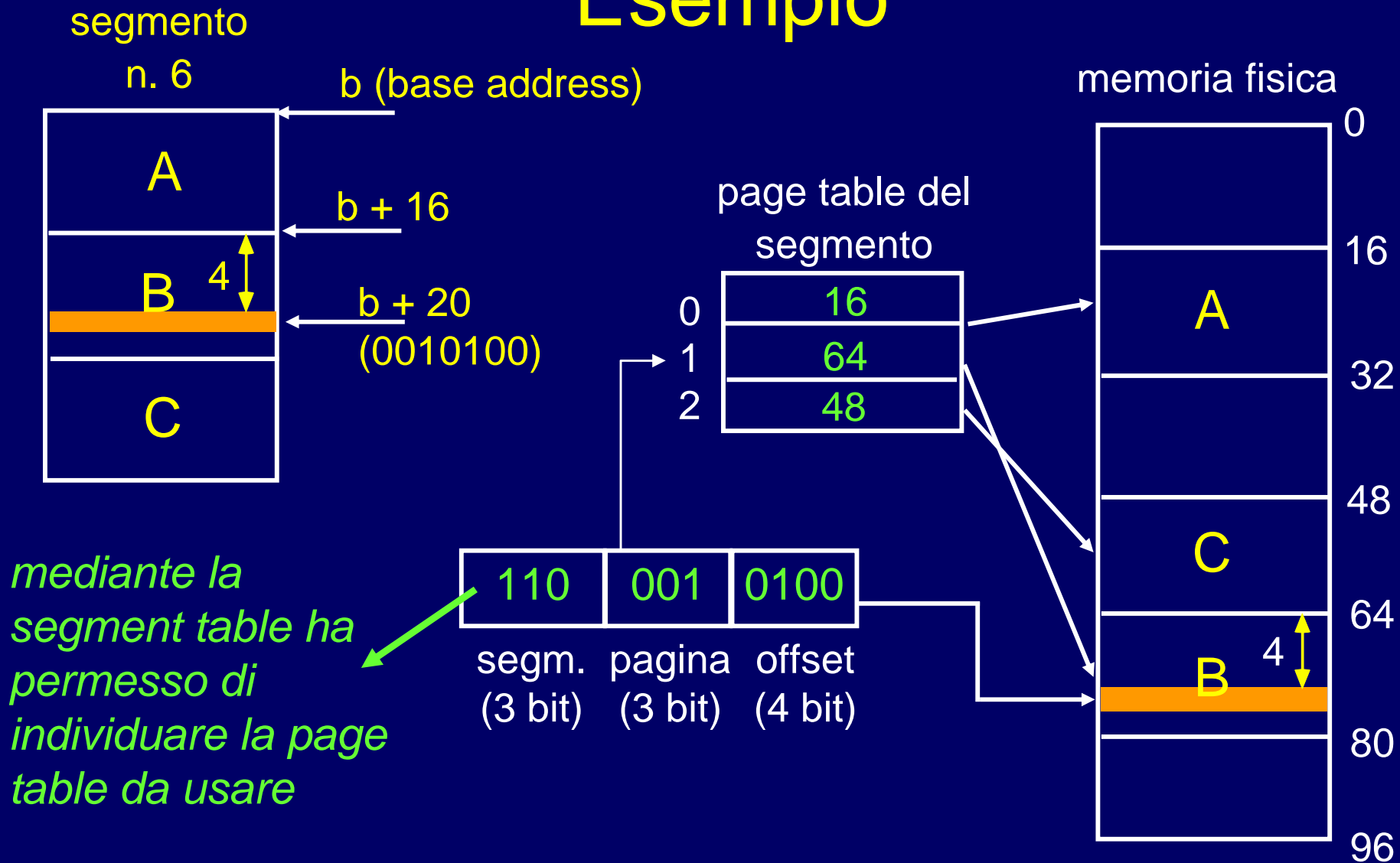


# Paginazione + segmentazione

- La paginazione pura può portare a page table molto grosse
- La segmentazione con paginazione permette di suddividerle in più page table (una page table diversa per ogni segmento)

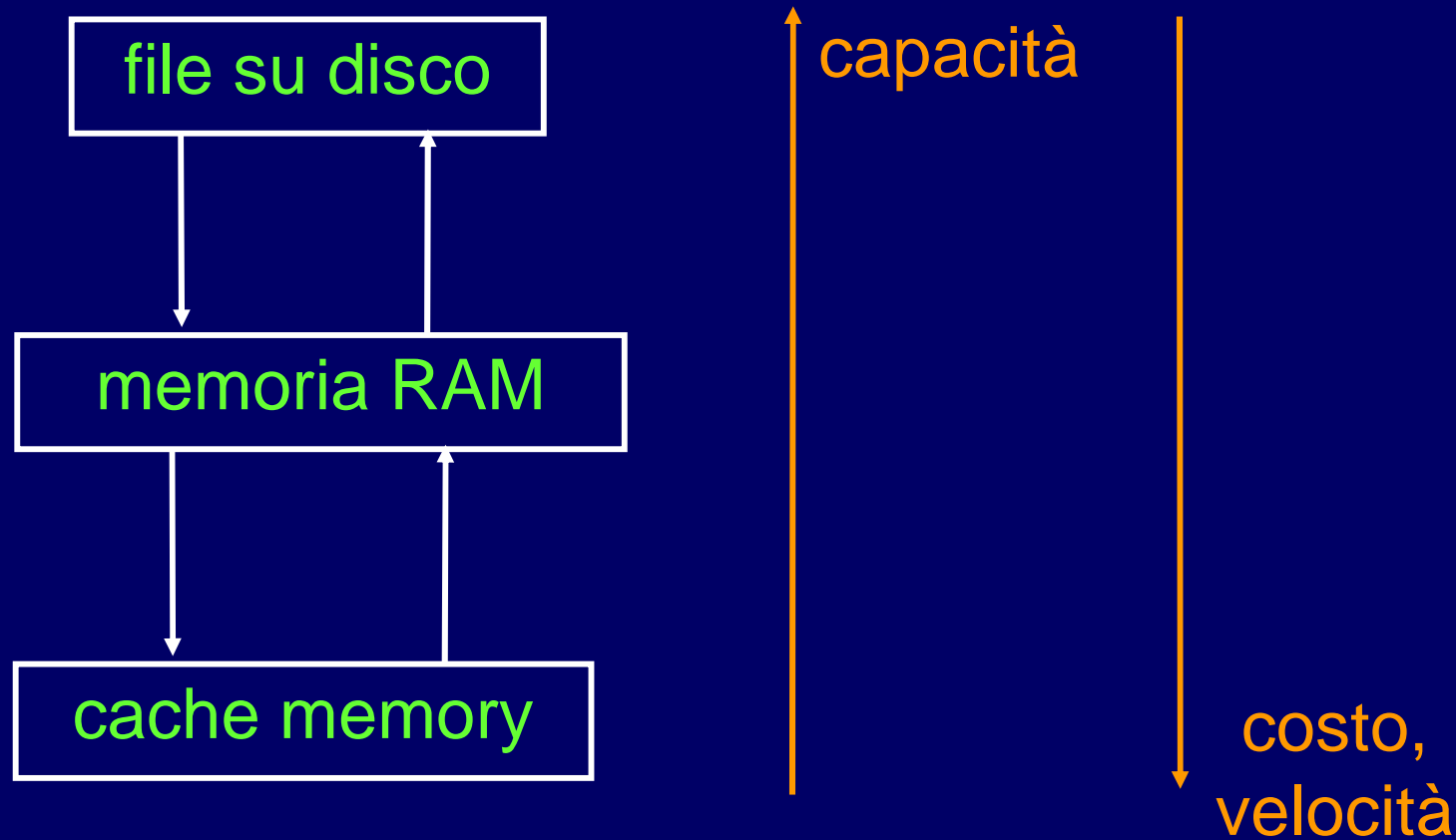


# Esempio



NOTA: n = 4 (pagine da 16 byte)

# Gerarchia di memoria



## Page file

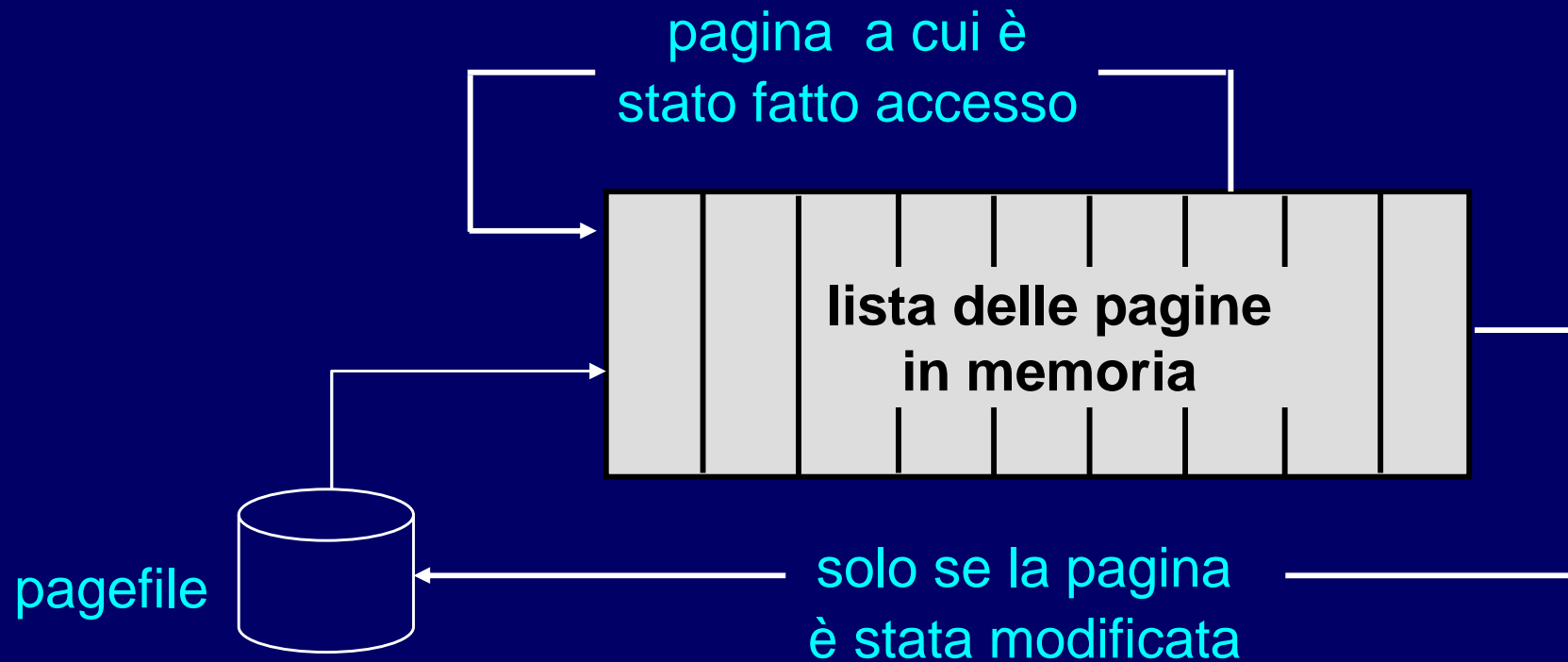
- Il meccanismo di traduzione degli indirizzi permette di far vedere ai programmi uno spazio di indirizzamento virtuale, per esempio da 00000000h a FFFFFFFFh (32 bit)
- La memoria RAM fisicamente presente nel calcolatore avrà normalmente dimensione molto inferiore a quella massima indirizzabile dalla CPU
- L'uso del page file permette di sfruttare la memoria su disco (più lenta, ma più economica) per espandere la memoria RAM centrale

# Page fault

- Quando si legge nella page table l'indirizzo della pagina, questo può essere marcato invalido (bit aggiuntivi oltre all'indirizzo di pagina in memoria fisica)
- Ciò significa che la pagina non è presente in memoria centrale ed è necessario leggerla dal page file su disco
- Troppi page fault provocano una degradazione delle prestazioni del sistema, esiste il problema del bilanciamento tra dimensione del page file e capacità della memoria centrale

# Algoritmo di gestione dei page fault

- Least Recently Used (LRU), altrimenti detto FIFO modificato.



## Attributi di pagina

- In corrispondenza di ogni pagina, nella page table sono contenute diverse informazioni:
  - bit di presenza (in memoria centrale)
  - se è presente, l'indirizzo della pagina di memoria, altrimenti, l'indirizzo all'interno del page file
  - informazioni di protezione (ereditate dal segmento di cui attualmente fa parte)
  - informazioni relative all'utilizzo e alla modifica del contenuto

# Swapping

- I primi sistemi UNIX, per liberare memoria centrale, trasferivano su memoria secondaria interi processi, e non soltanto parti di essi (pagine). Questo modo di procedere è detto swapping
- I sistemi con paginazione usano ancora lo swapping per liberare memoria quando il carico inizia ad essere eccessivo. In queste condizioni il sistema è impegnato quasi al 100% a caricare e scaricare pagine in memoria (trashing)
- I processi vengono nuovamente caricati in memoria quando è possibile riattivarli