

Gestione della memoria



Corso di Laurea in Ingegneria Informatica
Università degli Studi di Napoli Federico II
Anno Accademico 2024/2025, Canale San Giovanni

Sommario



- Sommario
 - Aspetti caratterizzanti la gestione della memoria
 - Segmentazione
 - Paginazione
 - Gestione della tabella delle pagine
 - Segmentazione con paginazione
- Riferimenti
 - P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari, "Sistemi Operativi", Mc-Graw-Hill (Cap. 4)
 - www.ostep.org, Capp. 13, 15, 16, 18, 19, 20
 - A. Silberschatz, Galvin, G. Gagne, "Sistemi Operativi" Addison-Wesley (Capitolo 8)

Introduzione



- La **memoria principale** costituisce, insieme alla CPU, una delle risorse per realizzare la astrazione di **processo**
- Il processo dispone di una **area di memoria** ad esso **riservata** (non accessibile da altri processi)



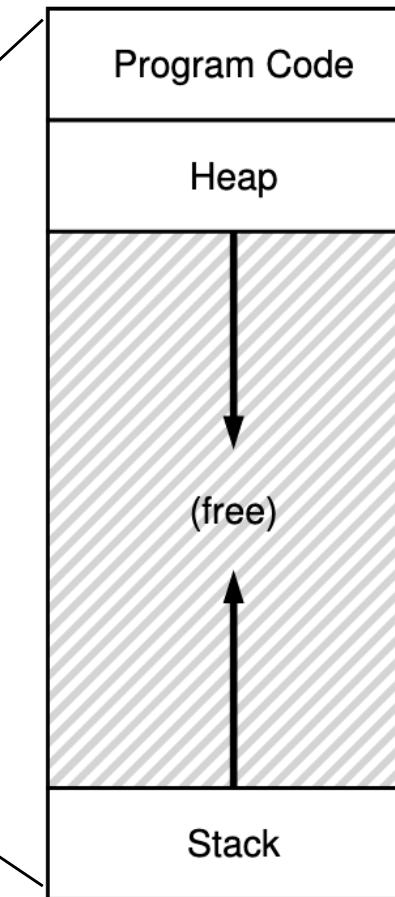
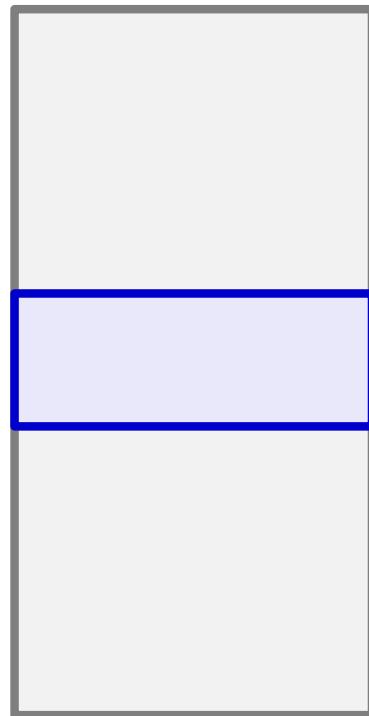
Memoria di un processo – modello concettuale

n bit di indirizzamento
fino a 2^n locazioni

Memoria
RAM

0

2^n





Virtualizzazione della memoria

```
$ cat va.c

#include <stdio.h>
#include <stdlib.h>

int main() {

    int x = 3;
    printf("location of stack: %p\n", &x);

    char * p = malloc(1024);
    printf("location of heap : %p\n", p);

    printf("location of code : %p\n", main);
}

$ ./va

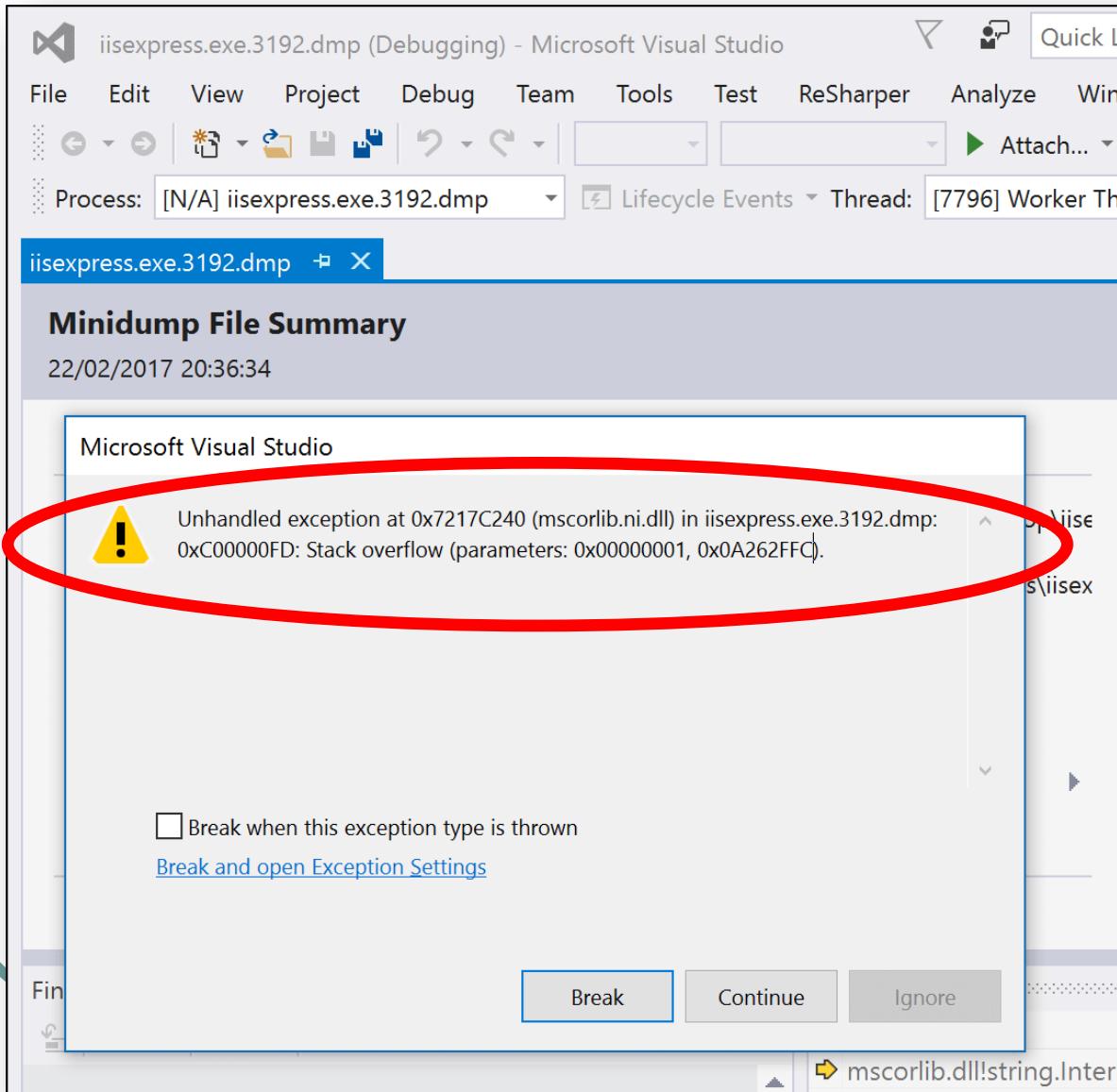
location of stack: 0x7fff691aea64
location of heap : 0x1096008c0
location of code : 0x1095afe50
```



Esiste davvero nella
RAM l'indirizzo
0x7fff691aea64?



Virtualizzazione della memoria





Virtualizzazione della memoria

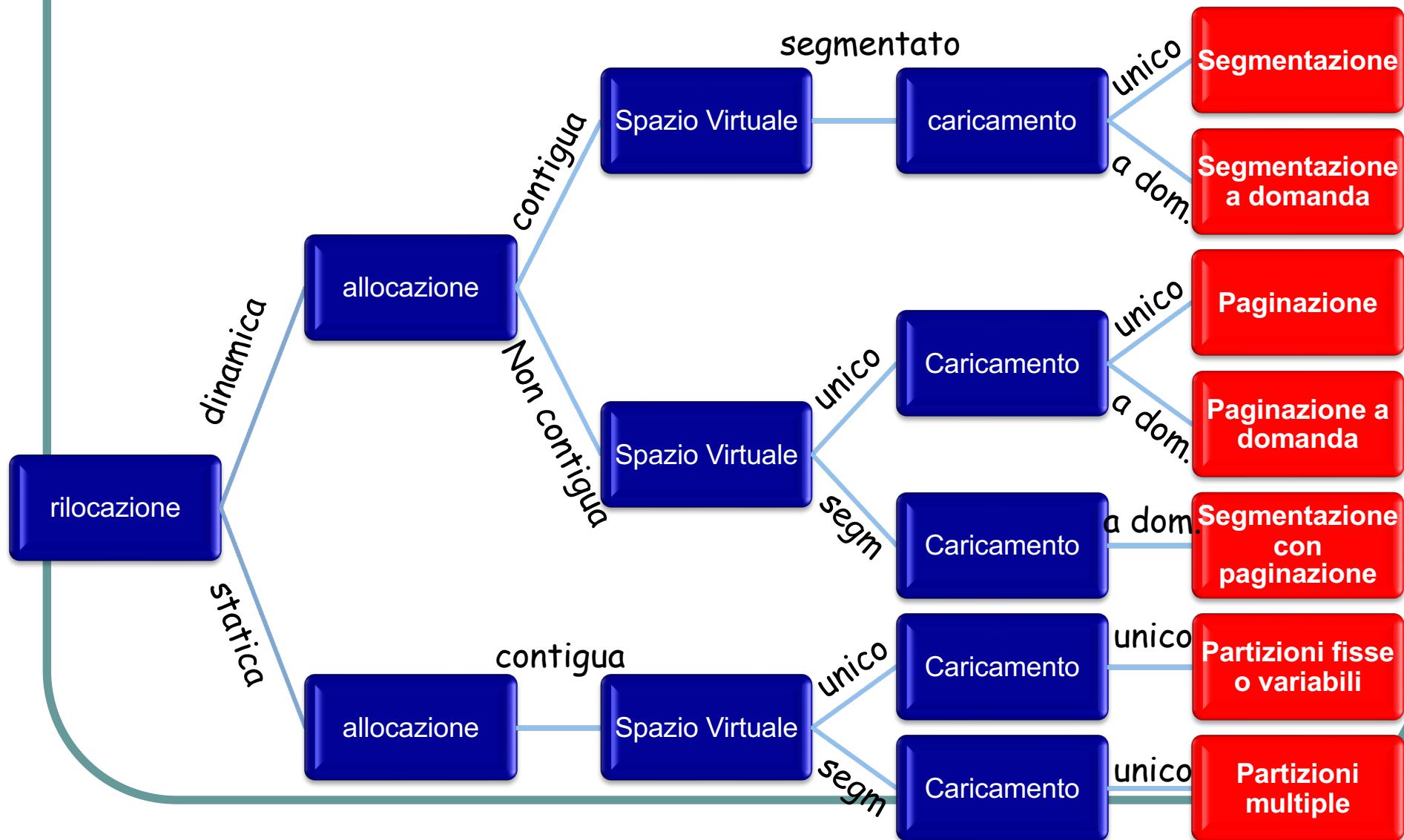
- La **posizione (indirizzi)** di codice e dati nella memoria del processo è una **astrazione (indirizzi "virtuali")**
- La posizione effettiva in **memoria fisica** è gestita dal sistema operativo

Aspetti e parametri caratterizzanti la gestione della memoria



- **Supporto Hardware** per la gestione della memoria (MMU)
- **Organizzazione logica** della memoria virtuale
- **Organizzazione fisica** (allocazione del processo)
- **Dimensione della memoria virtuale** (dimensione dell'immagine di un processo maggiore della dimensione fisica della memoria)
- **Rilocazione**
 - statica
 - dinamica
- **Organizzazione dello spazio virtuale**
 - spazio virtuale unico
 - spazio virtuale segmentato
- **Allocazione**
 - contigua
 - non contigua
- **Caricamento**
 - Tutto insieme
 - A domanda

Tecniche di gestione della memoria



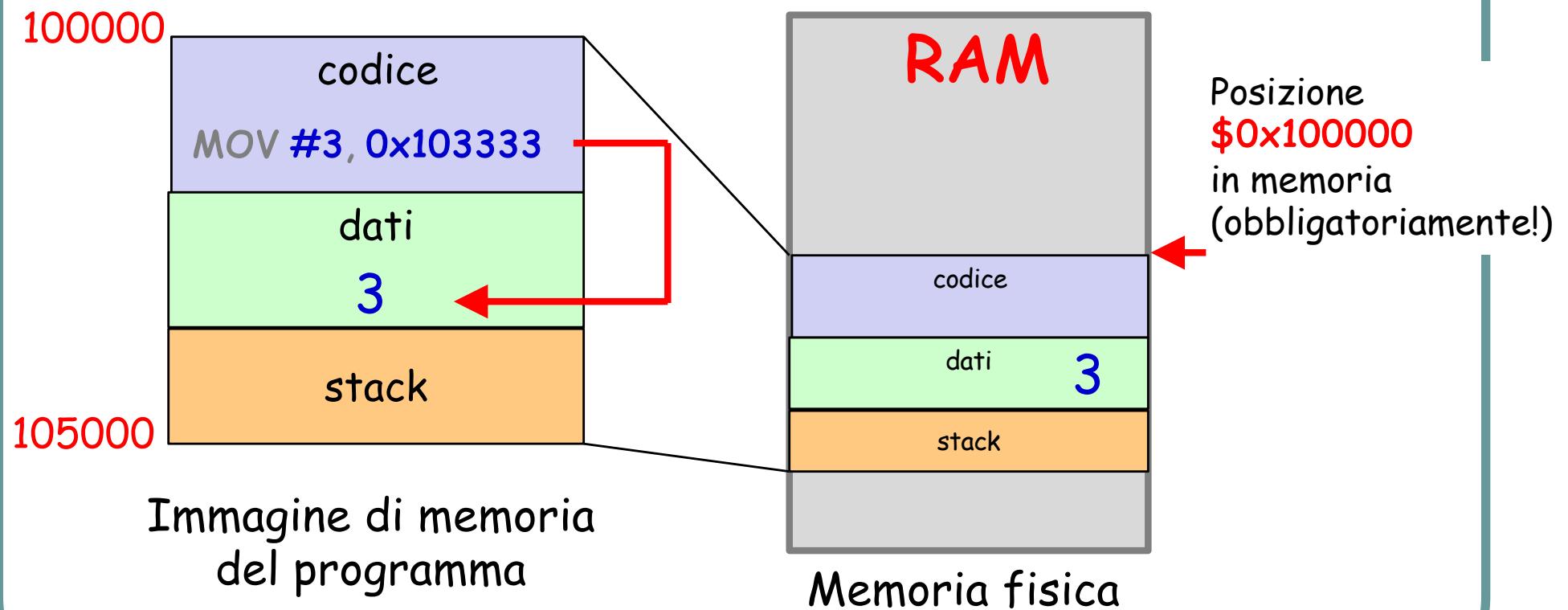


Rilocazione statica

- La **rilocazione statica** stabilisce gli indirizzi di codice e dati al momento della **compilazione o caricamento**
- La posizione di codice e dati **non può più essere modificata**



Rilocazione statica





Rilocazione dinamica

- Si introduce la seguente distinzione

Indirizzo virtuale

(*virtual address*, o anche *logical address*):

indirizzo acceduto dal programma

(accessi a variabili, salti, etc.) durante l'esecuzione

Indirizzo fisico (*physical address*):

indirizzo visto dalla unità di memoria,

posizione effettiva del dato/istruzione

Rilocazione dinamica



objdump

```
$ objdump -d programma
...
0000000000001149 <func>:
1149: f3 0f le fa
114d: 55
114e: 48 89 e5
1151: 48 83 ec 10
1155: 89 7d fc
1158: 8b 45 fc
115b: 89 c6
115d: 48 8d 3d a0 0e 00 00
1164: b8 00 00 00 00
1169: e8 e2 fe ff ff
116e: 90
116f: c9
1170: c3

0000000000001171 <main>:
1171: f3 0f le fa
1175: 55
1176: 48 89 e5
1179: 48 83 ec 10
117d: 8b 05 8d 2e 00 00
1183: 83 c0 03
1186: 89 45 fc
1189: 8b 45 fc
118c: 89 c7
118e: e8 b6 ff ff ff
1193: b8 00 00 00 00
1198: c9
1199: c3
119a: 66 0f 1f 44 00 00

Il programma finale (main.o + mylib.o) contiene l'immagine di memoria di tutto il codice e dati, stabilendo la loro posizione relativa in memoria

Il caricatore copierà il codice del main() a partire dall'indirizzo di memoria 1171 (virtuale)

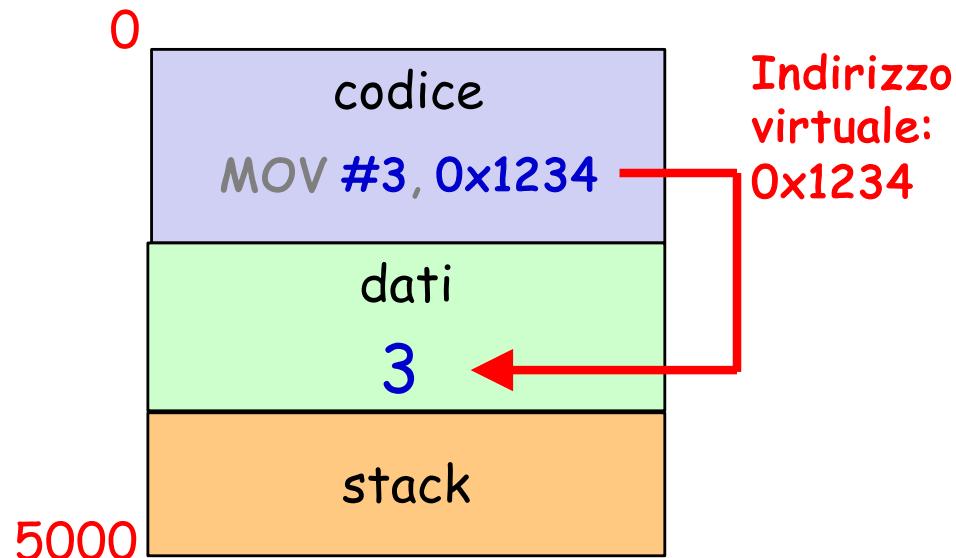
Gli indirizzi precedentemente vuoti sono riempiti dal linker
```

- Vedi lezione sul Makefile e librerie (comando "objdump")

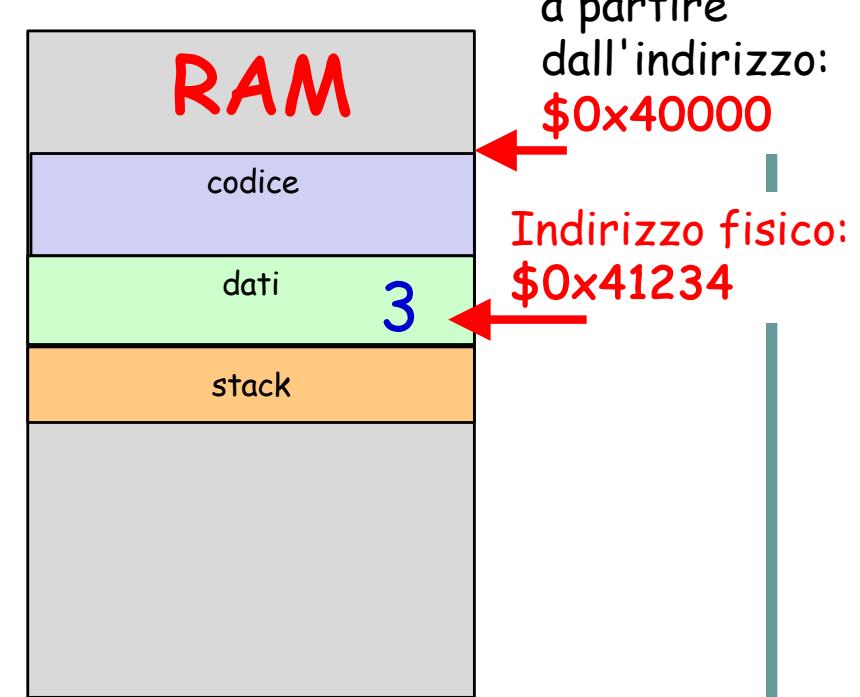
- Il programma contiene indirizzi virtuali "fittizi"

Nota: un processo non ha modo di conoscere gli indirizzi fisici dei propri dati/codice

Rilocazione dinamica



Spazio di memoria virtuale
(indirizzi scelti arbitrariamente)

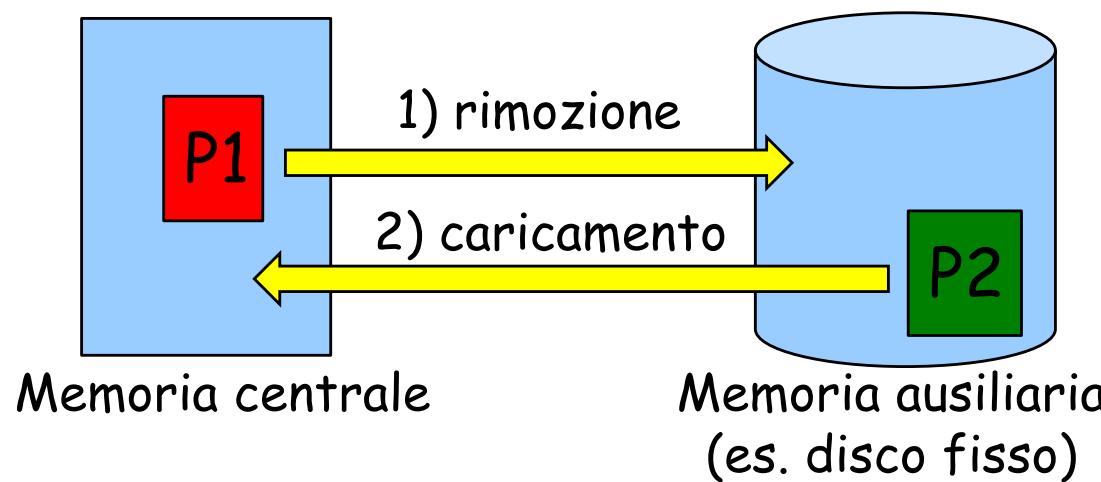


Spazio di memoria fisico
(dal punto di vista della RAM)



Vantaggi della rilocazione dinamica

- La **rilocazione dinamica** consente lo **swapping**
 - Un processo è temporaneamente sospeso e trasferito in **memoria secondaria** (es. dischi)
 - Il processo potrà poi essere ri-caricato in un'**area di memoria differente**





Unità di gestione della memoria (MMU)

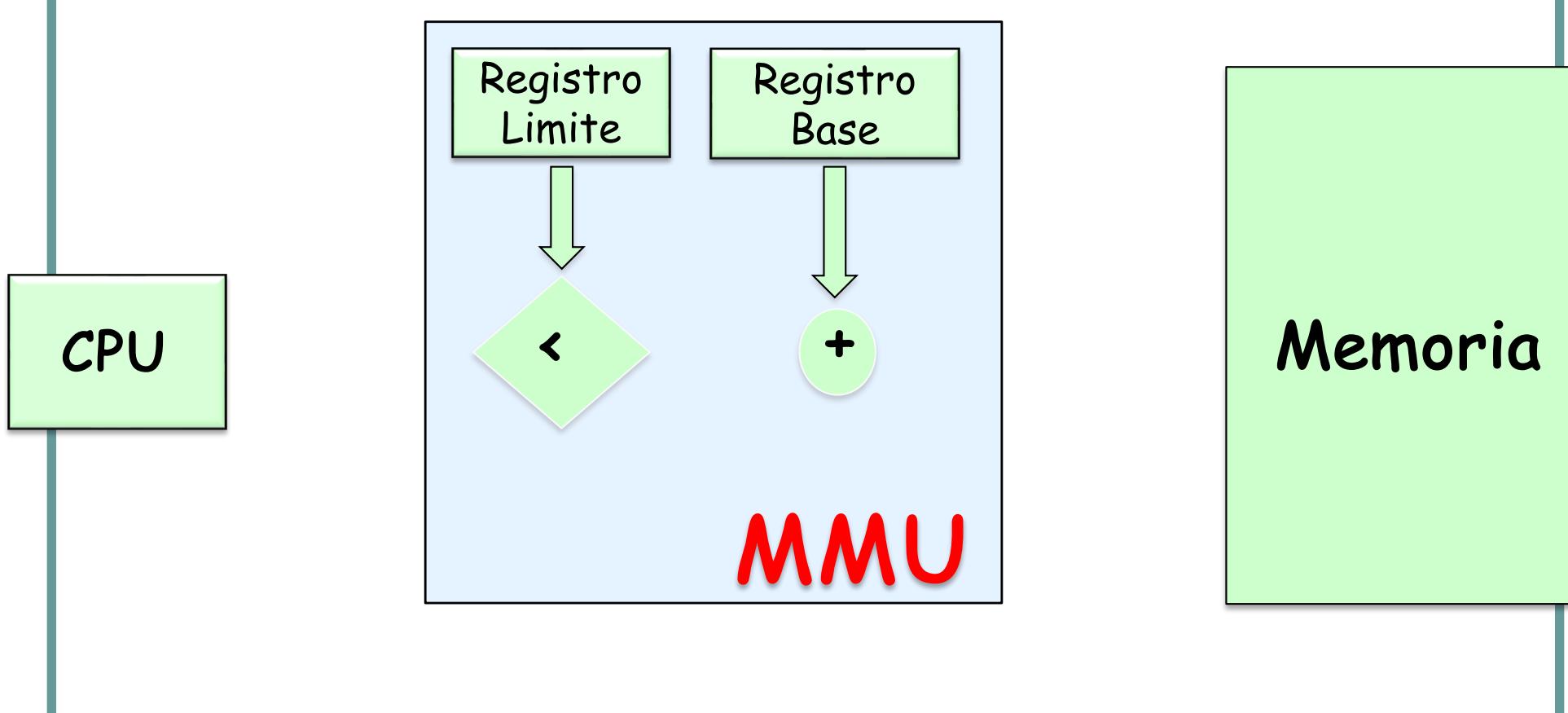
- **MMU (Memory Management Unit):**
 - componente (hardware) della CPU
 - traduce gli indirizzi **virtuali** in **fisici** durante l'esecuzione
 - dopo la traduzione, la CPU accede agli **indirizzi fisici**



Esempio di rilocazione dinamica

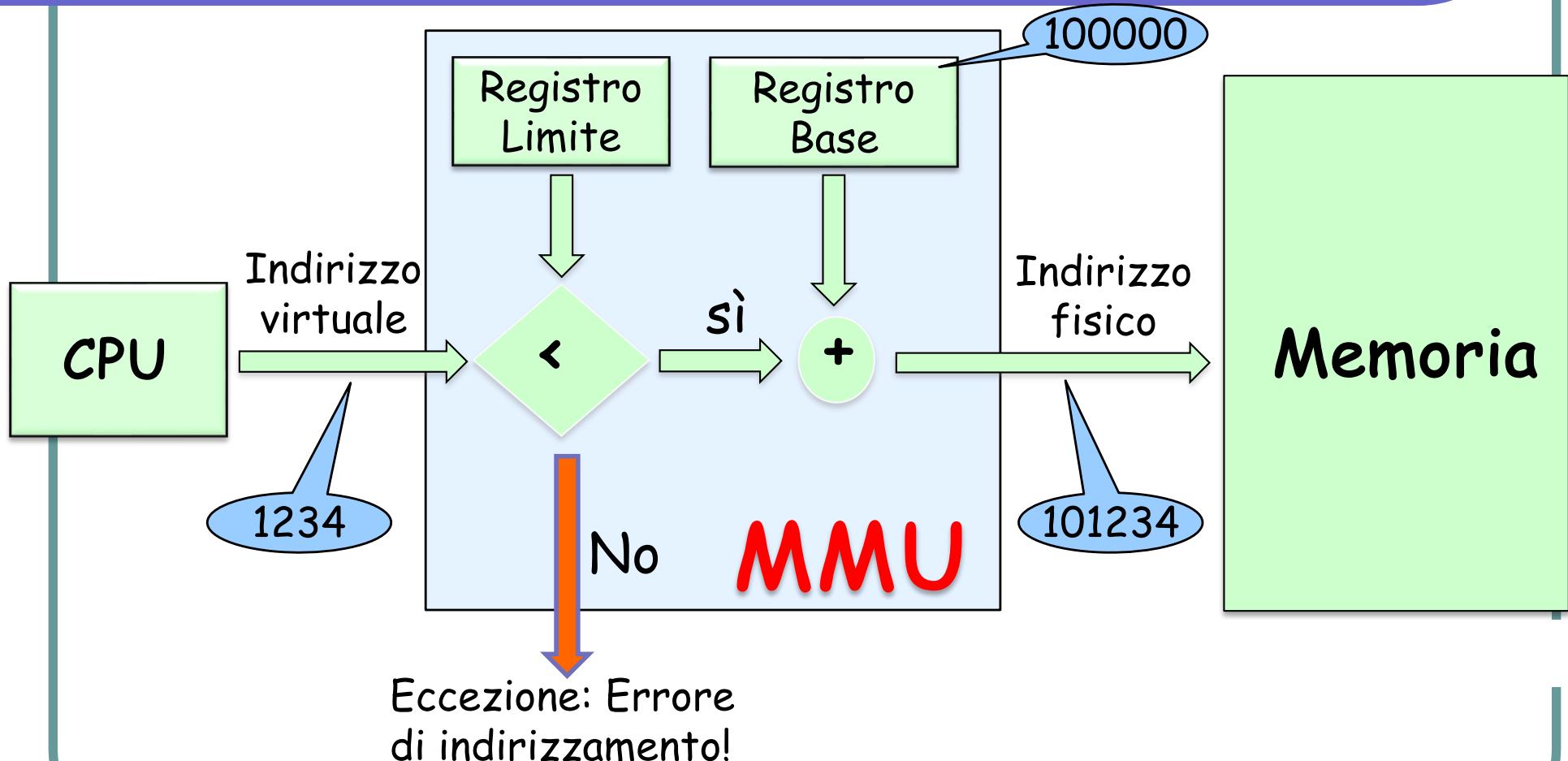
- Caso basilare (obsoleto):
 - **spazio virtuale unico**
 - **allocazione contigua**
- Approccio primordiale di rilocazione dinamica nei primi sistemi

Registri di rilocazione e di limite



La MMU utilizza due registri speciali ("base" e "limite")
indirizzo fisico = indirizzo virtuale + registro base

Registri di rilocazione e di limite





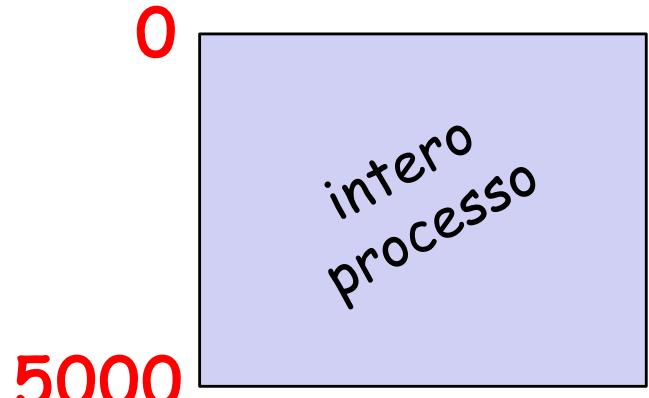
Gestione dello spazio virtuale

- Vi sono due possibili approcci per gestire lo **spazio virtuale** degli indirizzi
 - Uno **spazio unico**
(corrispondente all'intero processo)
 - Un insieme di **segmenti**
(**segmentazione** - la memoria del processo è gestita in porzioni separate)

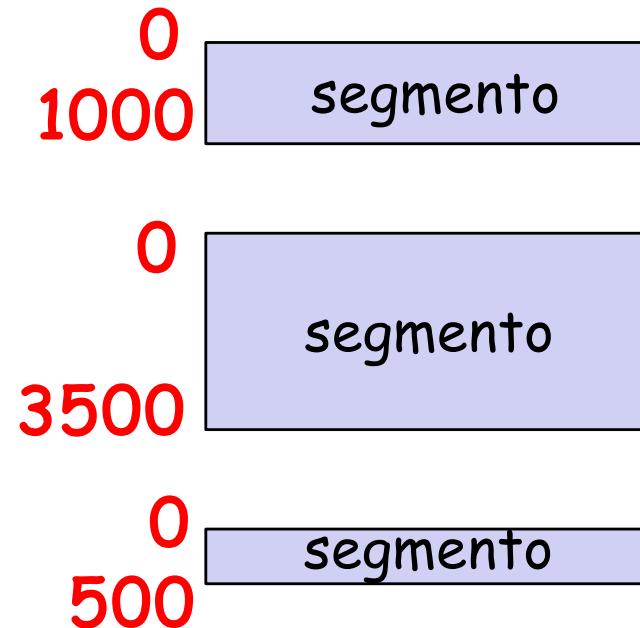


Gestione dello spazio virtuale

- Vi sono due possibili approcci per gestire lo **spazio virtuale** degli indirizzi



spazio unico
(unico segmento)

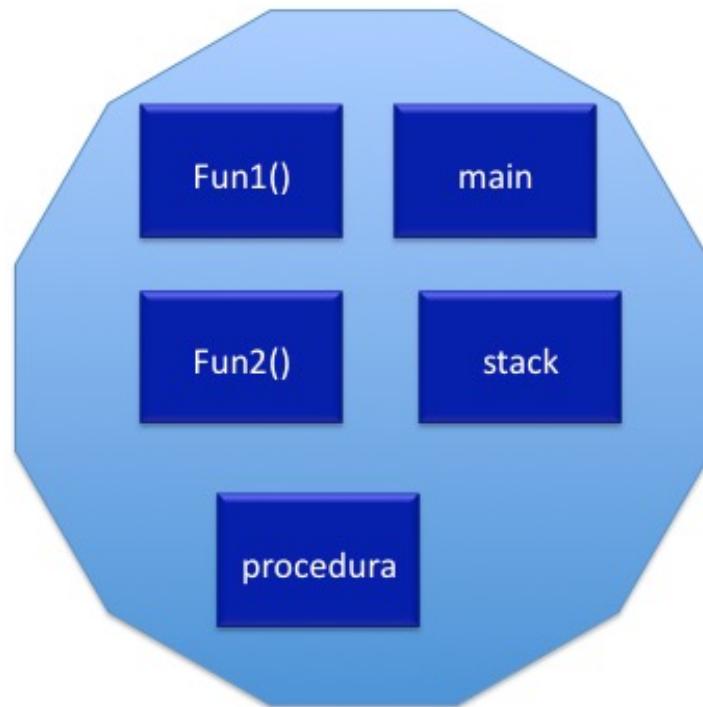


segmentato
(gestito in porzioni separate)

Segmentazione



- A **tempo di compilazione**, si configura lo spazio virtuale segmentato
- Viene creato un diverso **segmento** per ciascun modulo del programma





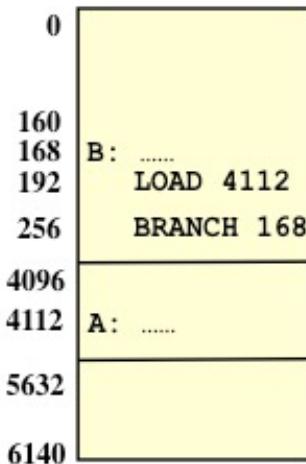
Vantaggi della segmentazione

- **Protezione** dei segmenti
 - es. lettura/scrittura/esecuzione
- **Condivisione** dei segmenti
 - es. libreria di codice usata da più processi
- **Allocazione indipendente** dei segmenti in memoria fisica
 - riduce (ma non elimina) il problema della frammentazione esterna

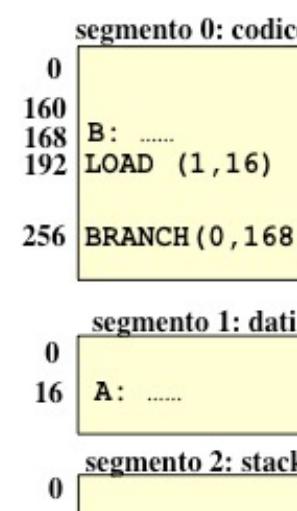


La segmentazione

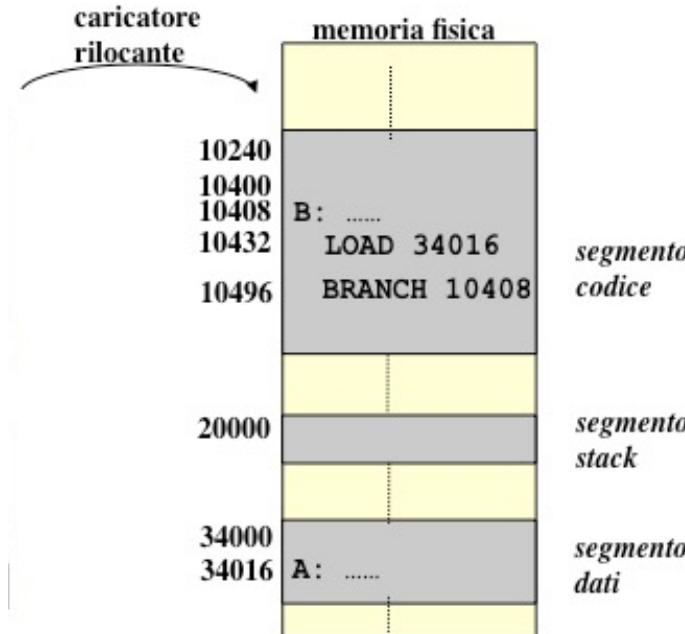
- Un indirizzo virtuale è una coppia fatta da
 - un *identificativo* del segmento
 - uno *scostamento* (offset) all'interno del segmento



a) spazio
virtuale **unico**



b) spazio
virtuale
segmentato



c) immagine del
processo in
memoria fisica

Segmentazione – esempio x86



```
push $0x1234  
pop %es  
mov %es:0x5678, %eax
```



Indirizzo virtuale, è la coppia:

- segmento (0x1234)
- scostamento (0x5678)

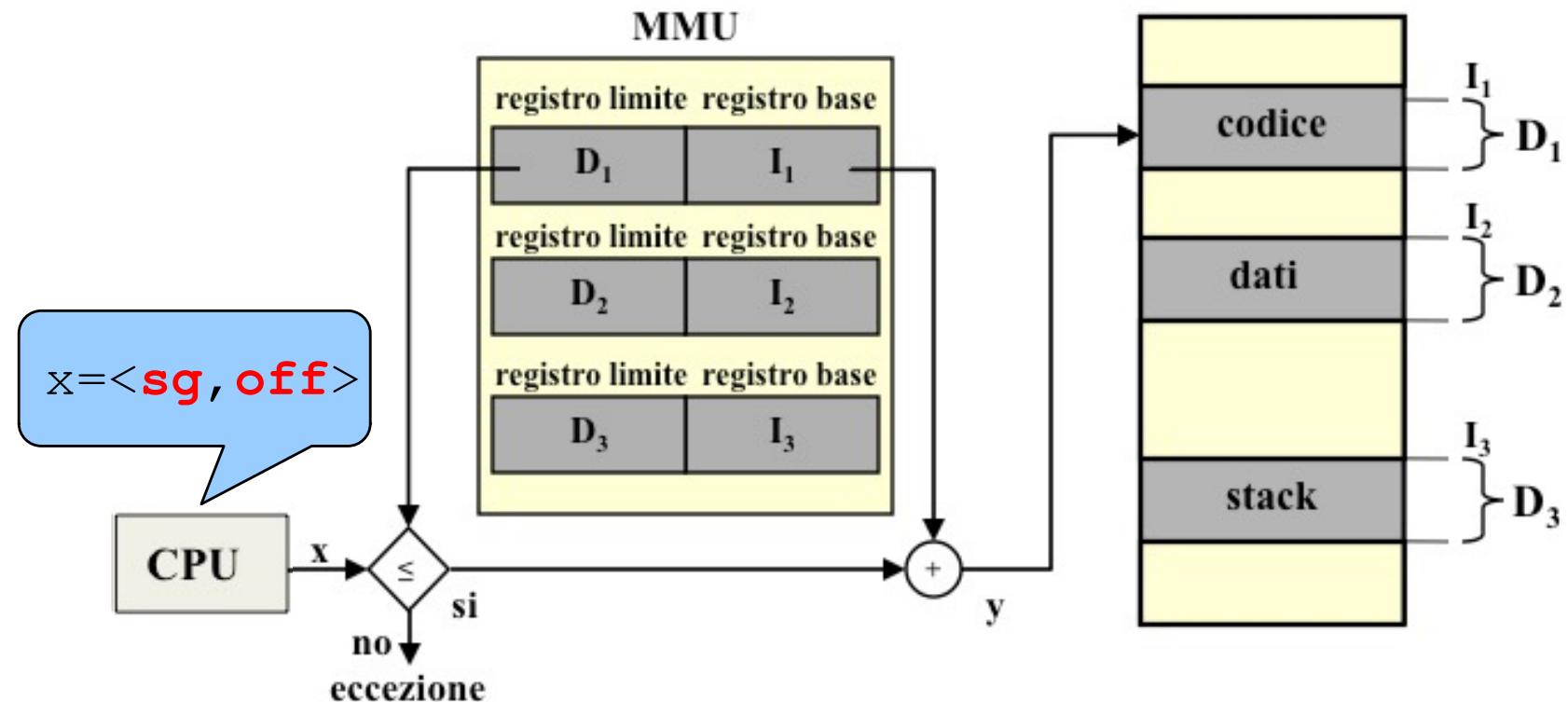


Segmentazione: caso semplice

- Anche la segmentazione si basa su **MMU**
- Tradurre gli indirizzi virtuali in fisici
- Nel caso di **pochi segmenti** (es. 3), è sufficiente avere nella MMU **più coppie di registri base/limite**



Segmentazione: caso semplice





Segmentazione: caso generale

- Per ottenere un **numero arbitrario di segmenti**:
 - non è avere tutti i registri base/limite all'interno della MMU
 - le **copie base/limite** per ogni segmento sono invece mantenute in **memoria RAM**
 - area apposita detta **tabella dei segmenti (segment table)**

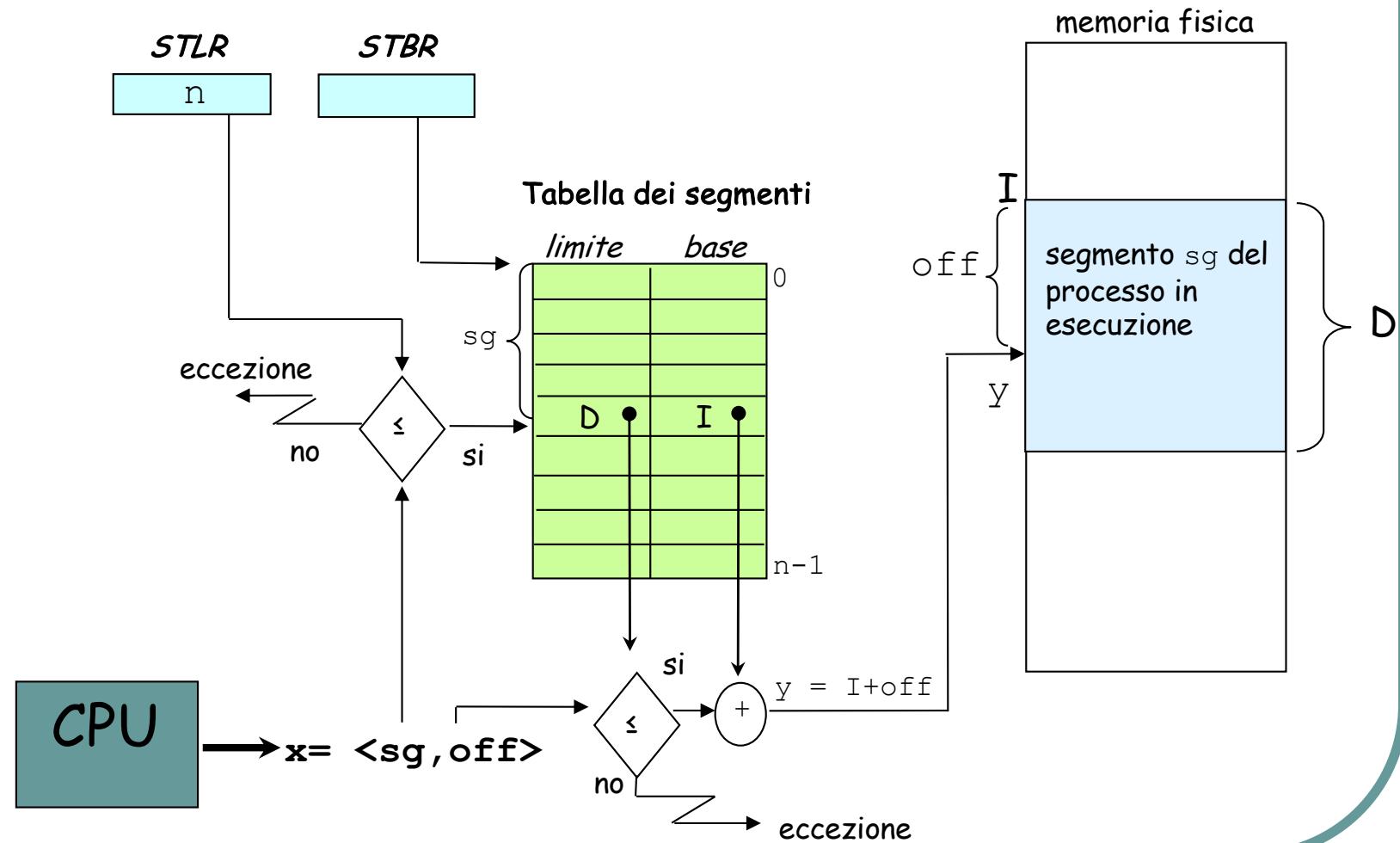


Traduzione degli indirizzi segmentati

- La MMU gestisce la segment table con due appositi **registri**:
 - **STBR (Segment Table Base Register)**: indirizzo in memoria in cui si trova la tabella dei segmenti
 - **STLR (Segment Table Limit Register)**: dimensione della tabella dei segmenti (indica il numero di segmenti del processo)



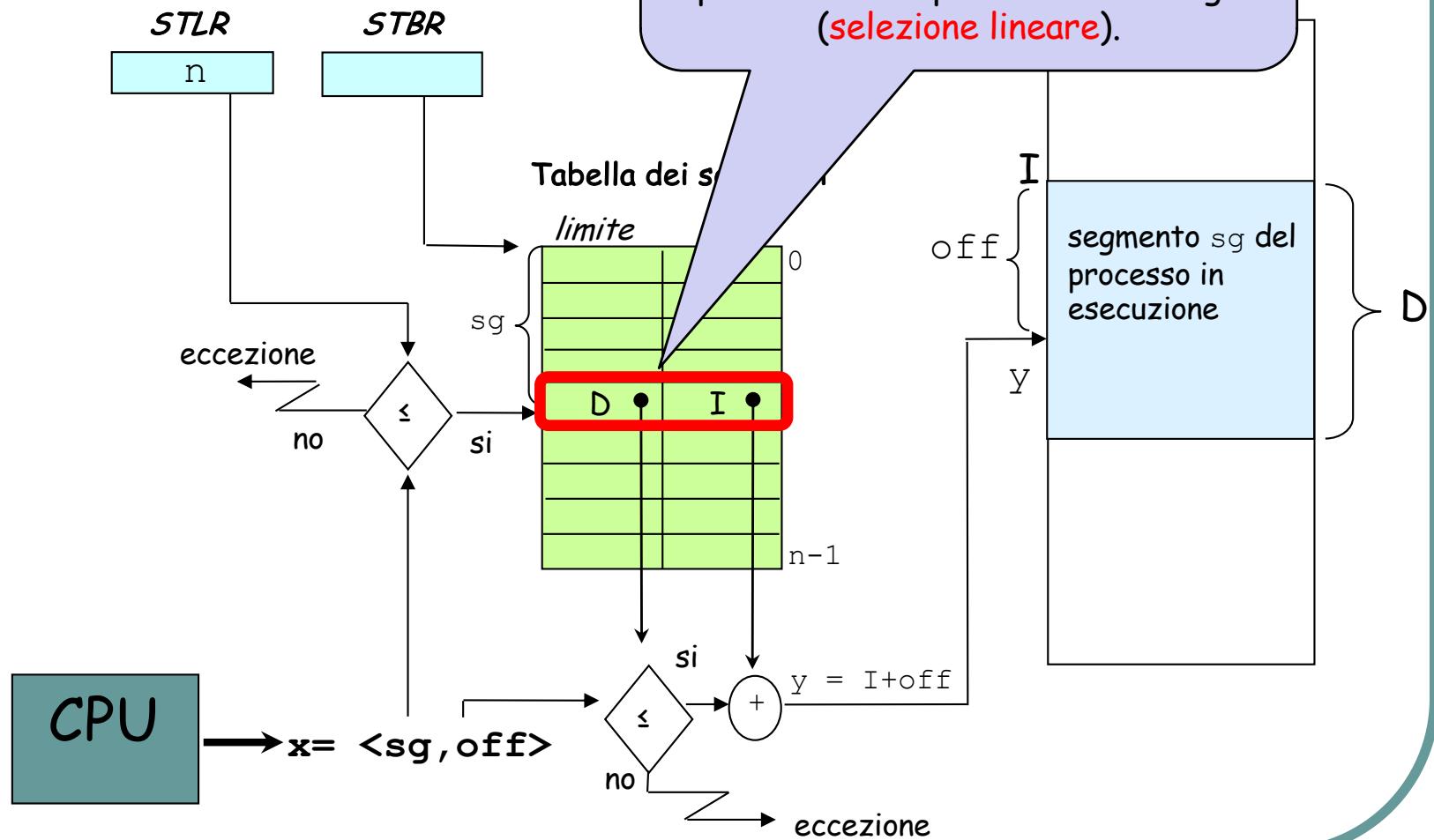
Traduzione degli indirizzi segmentati





Traduzione degli indirizzi segmentati

Nota: La riga del segmento "sg" non contiene il valore di "sg" stesso.
La MMU calcola la somma **STBR+sg** per trovare la posizione della riga
(selezione lineare).



Segmentazione

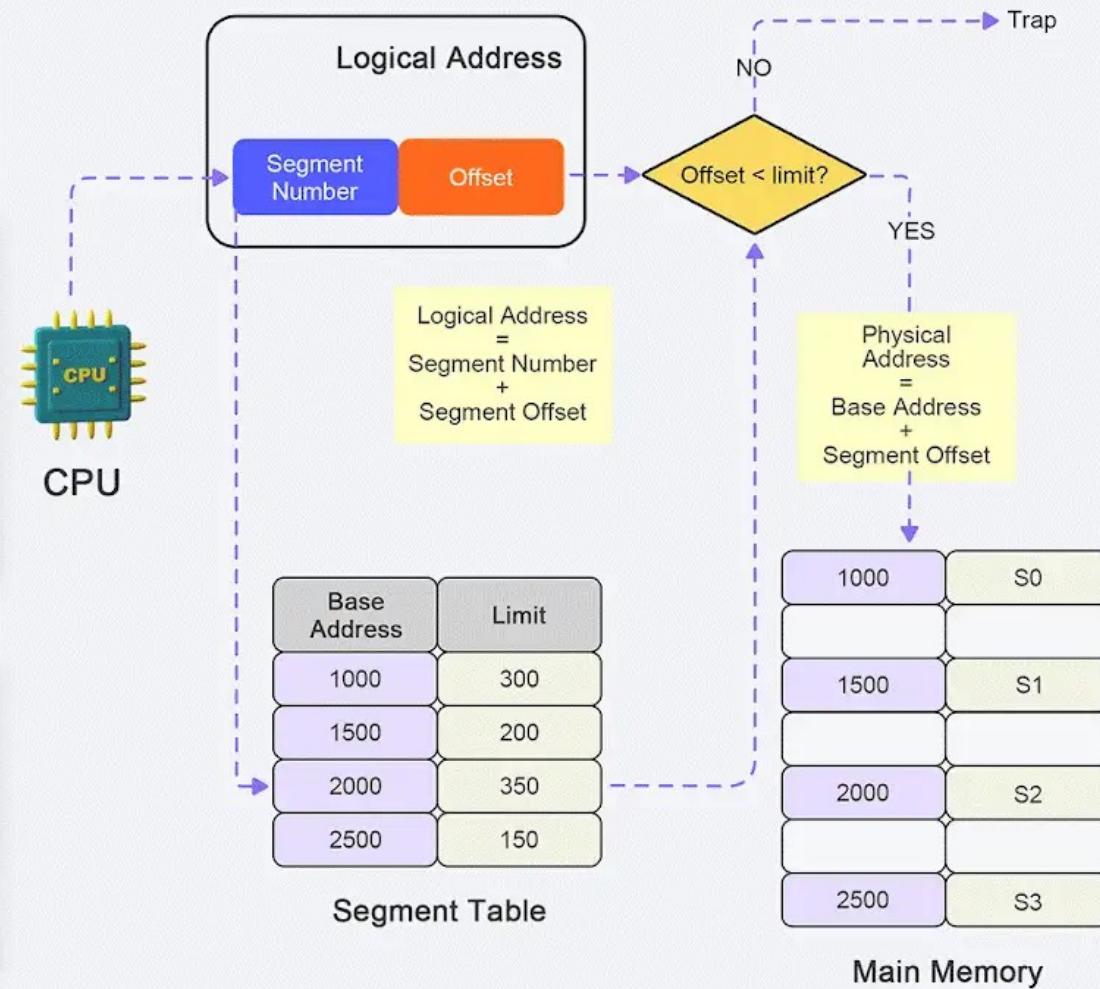


Segmentation

Segmentation is a memory management technique where the memory is divided into **variable-sized** segments based on the **logical divisions of a program**, such as functions, objects, or data arrays.

Advantages

- Provides logical separation of different parts of a program.
- Facilitates protection and sharing of segments.
- Simplifies management of growing data structures.



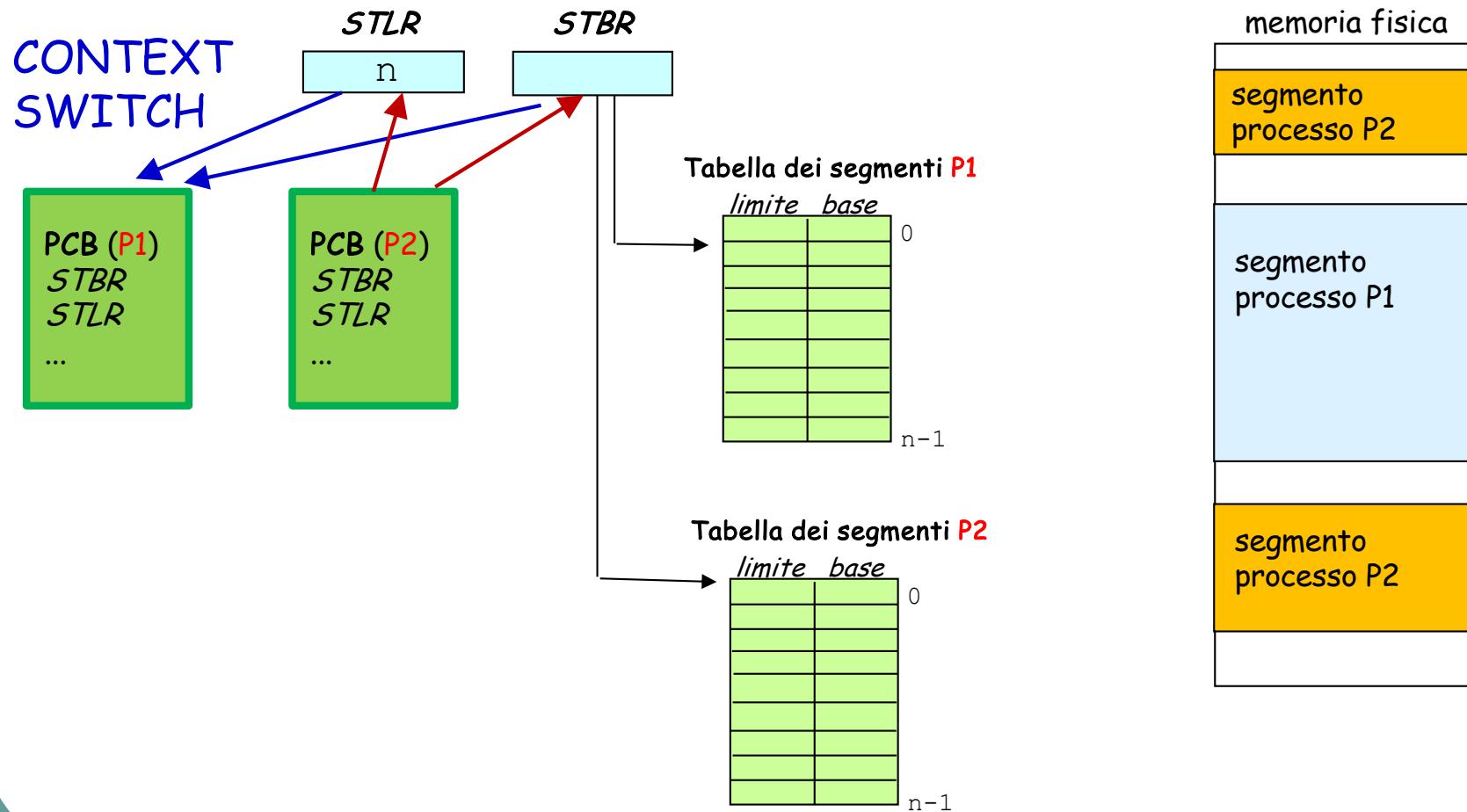


Nota sulla segment table

- Ogni processo ha una **segment table differente**
- I registri STBR/STLR sono configurati ad ogni **context switch** dei processi
- Il SO carica i valori di STBR/STLR dal **PCB (Process Control Block)**



Traduzione degli indirizzi segmentati





Protezione

- Ogni segmento può avere diversi **permessi di accesso**
- Vi sono dei **bit di controllo** in ogni **riga della tabella dei segmenti**
- MMU produce una **exception** se il programma non rispetta i permessi

Descrittore di un segmento



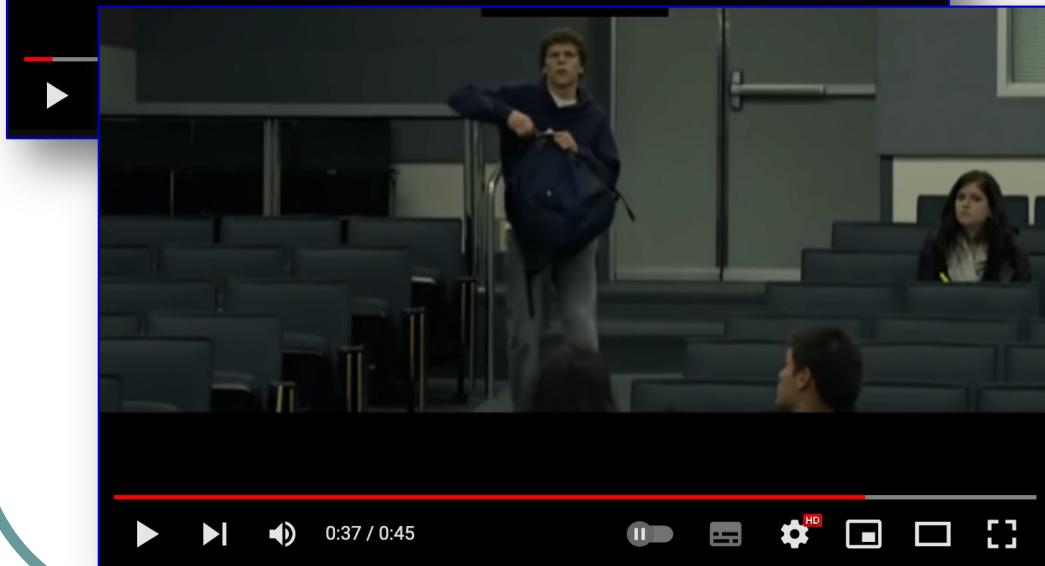
bit di controllo



The Social Network (2010)



"Don't worry Mr. Zuckerberg, brighter men than you have tried and failed at this class"



"1 valid bit, 1 modified bit, 1 reference bit, 5 permission bits"

https://www.youtube.com/watch?v=-3Rt2_9d7Jg



Condivisione

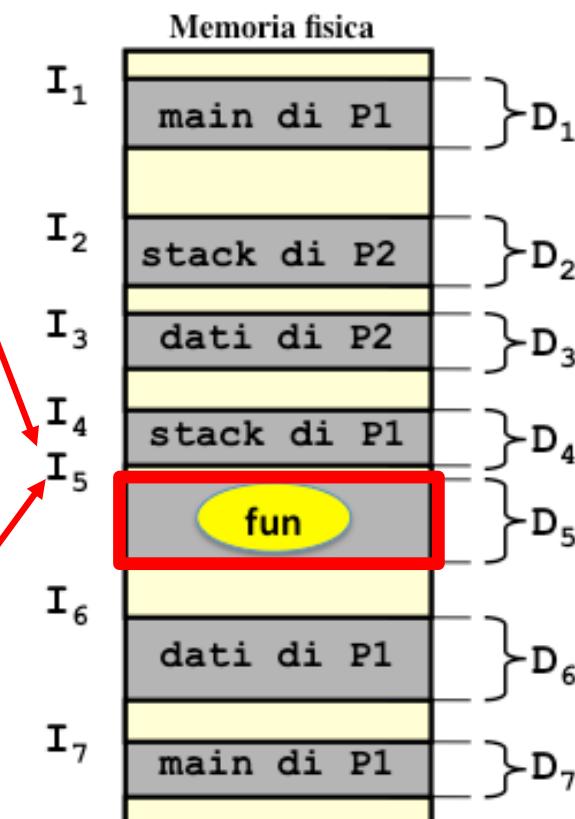
È possibile **condividere dei segmenti** tra più processi, allocando in memoria fisica una sola copia del segmento

	Base	limite	controllo	
main	I ₁	D ₁		0
fun	I ₅	D ₅		1
dati	I ₆	D ₆		2
stack	I ₄	D ₄		3

Tabella dei segmenti di P1

	Base	limite	controllo	
main	I ₇	D ₇		0
fun	I ₅	D ₅		1
dati	I ₃	D ₃		2
stack	I ₂	D ₂		3

Tabella dei segmenti di P2



Allocazione

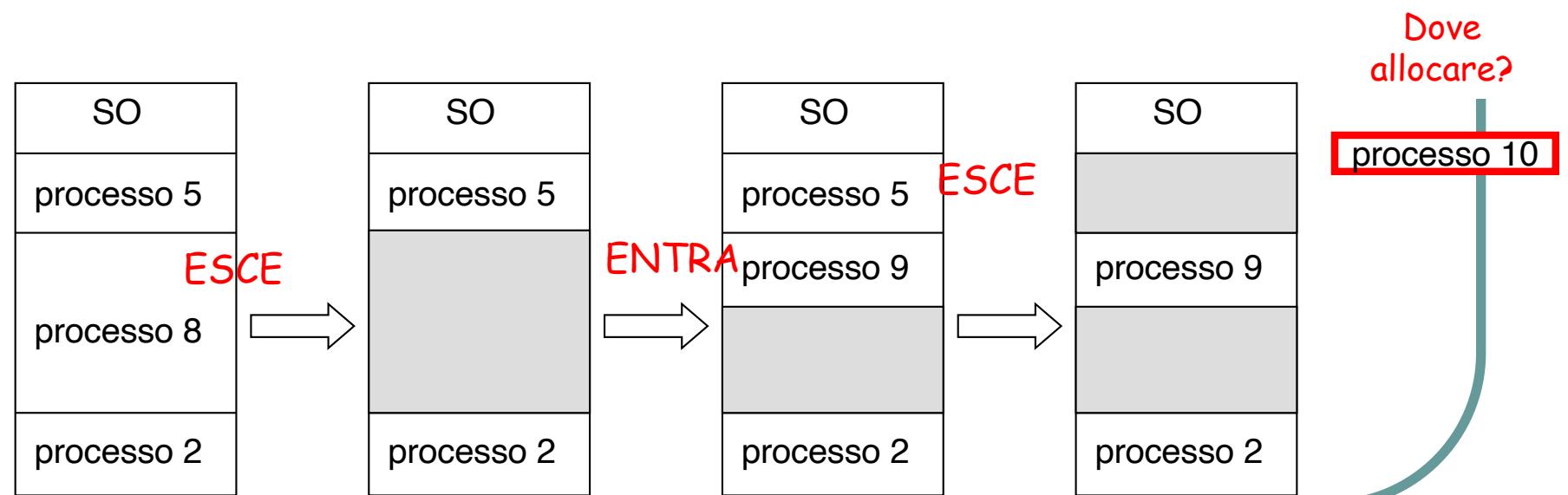


- Uno spazio/segmento di memoria virtuale può essere collocato in memoria fisica in due possibili modi
 1. **Allocazione contigua:**
lo spazio/segmento è **copiato per intero**, in un intervallo di memoria fisica agli indirizzi $[I, I+D]$
 2. **Allocazione non contigua (paginazione)**

Allocazione contigua (partizionamento, a dimensione variabile)



- Il SO colloca il proprio blocco di memoria virtuale, e quelli dei processi, in **intervalli non-sovrapposti** della memoria fisica
- Quando un processo termina, la memoria fisica occupata si libera, creando un "buco" (**hole**)
- Quando si carica un processo, occorre cercare un hole sufficientemente grande da contenerlo



Allocazione contigua (partizionamento, a dimensione variabile)



- Se ci sono **più buchi liberi**, ci sono vari criteri per scegliere dove collocare un segmento
- **First-fit**: si assegna il **primo hole** sufficientemente grande
- **Best-fit**: si assegna lo hole **più piccolo** tra quelli sufficientemente grandi
- **Worst-fit**: si assegna lo hole **più grande**



Frammentazione



In generale, gli schemi a partizioni di **dimensione variabile** soffrono del problema della frammentazione esterna

- **Frammentazione esterna:** Spazio di memoria perduto sotto forma di spezzoni
 - lo spazio di memoria totale sarebbe sufficiente per soddisfare una richiesta, ma non è contiguo

Paginazione



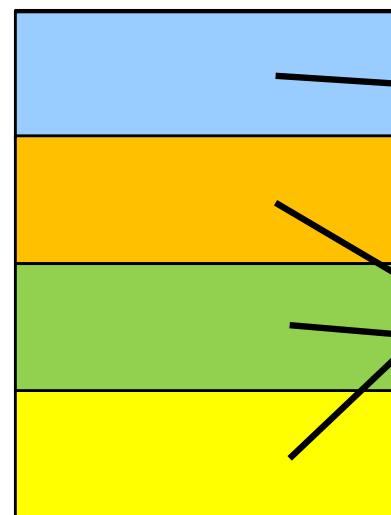
- Paginazione:
 - Tecnica di allocazione **non contigua**
 - Gli spazi/segmenti sono divisi in **blocchi di dimensioni fissa**
 - Evita la frammentazione esterna
 - Introduce frammentazione interna

Paginazione

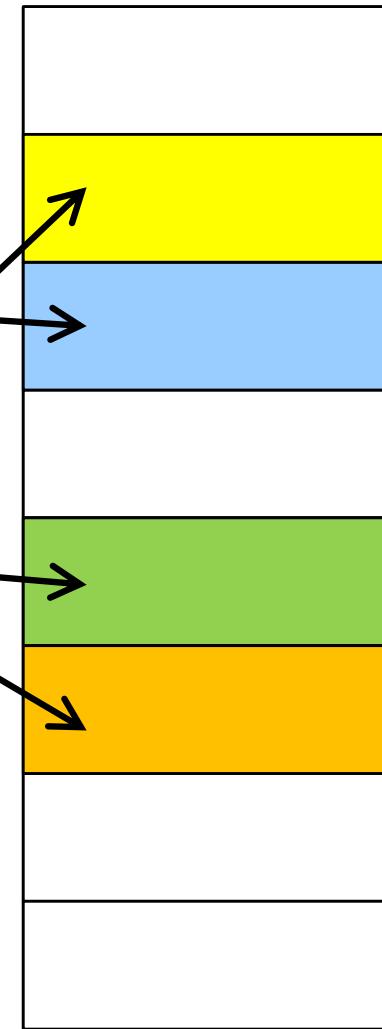


Memoria fisica

Memoria virtuale



*divisa in blocchi di dimensione fissa
(detti pagine virtuali)*



Ogni pagina virtuale è abbinata ad una pagina fisica, tramite una tabella delle pagine

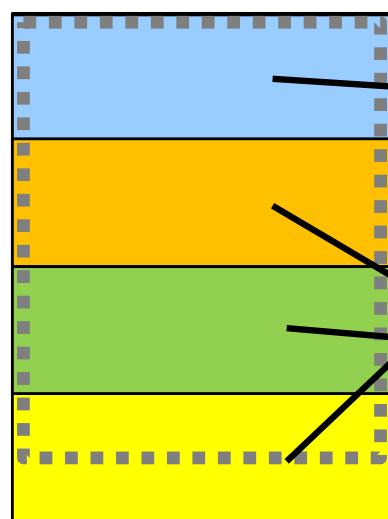
*divisa in blocchi di dimensione fissa
(detti pagine fisiche o frame)*

Paginazione



Memoria fisica

Memoria virtuale



Frammentazione interna:
il processo non utilizza
appieno le pagine assegnate

Ogni pagina virtuale è
abbinata ad una pagina
fisica, tramite una
tabella delle pagine

divisa in blocchi di
dimensione **fissa**
(detti **pagine fisiche**
o **frame**)



Frammentazione interna

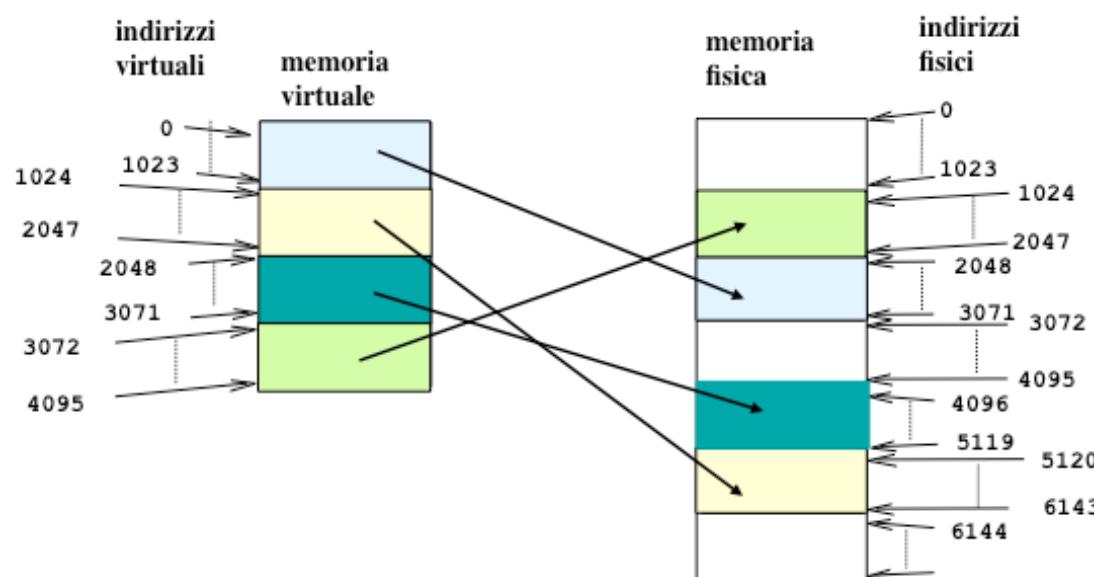
- **Frammentazione interna:**

- Spazio di memoria perduto per un blocco assegnato ma **non utilizzato per intero**
- si verifica se la dimensione del processo non è un multiplo esatto della dimensione dei blocchi
- fenomeno trascurabile se pagine sono piccole



Esempio di paginazione

Tipicamente, la dimensione di pagina è una potenza di 2, compresa tra 512 byte e 16MB





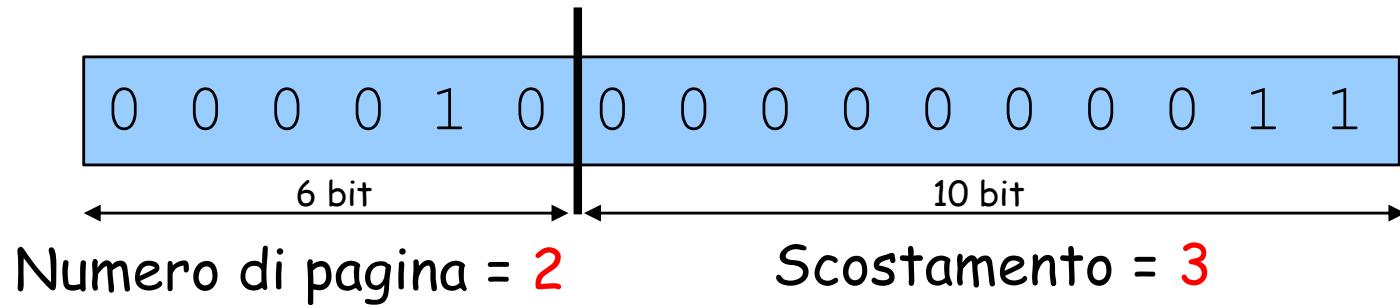
Schema di traduzione degli indirizzi

- Nella paginazione, un indirizzo contiene la **coppia**:
 - **Numero di pagina (p)**: indice della pagina nella memoria fisica
 - **Scostamento di pagina (d)**: indica la posizione dell'indirizzo all'interno della pagina
- A differenza della segmentazione, non sono due valori separati, ma sono contenuti entrambi in un unico valore



Schema di traduzione degli indirizzi

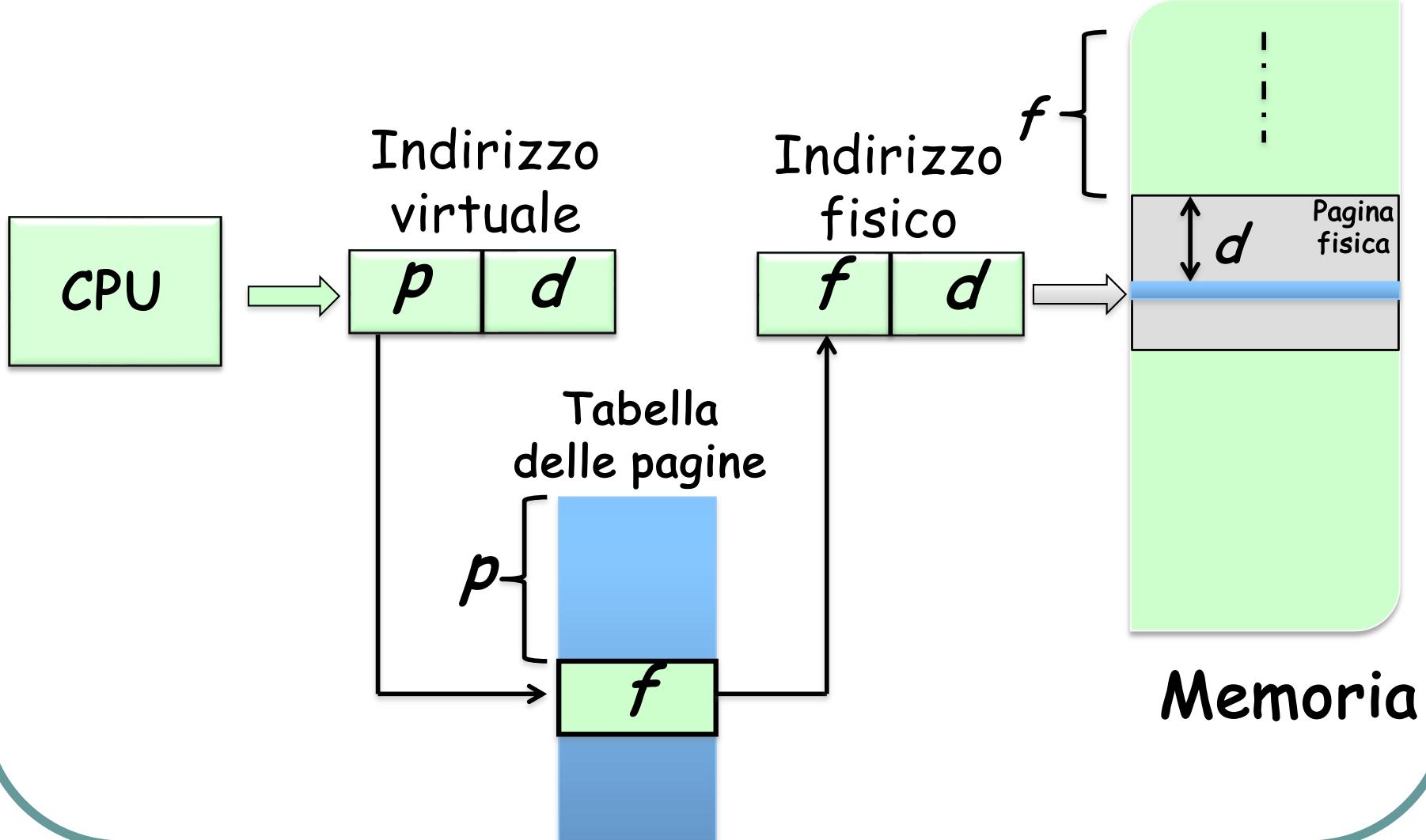
Esempio (indirizzo virtuale a **16bit**): 0x0803



Dimensione delle pagine: $2^{10} = 1024 = 1\text{kB}$

Numero massimo di pagine: $2^6 = 64$ pagine

Architettura di paginazione



Paginazione



Paging

Paging is a memory management scheme that eliminates the need for **contiguous allocation** of physical memory. The process's address space is divided into fixed-size blocks called **pages**, while physical memory is divided into fixed-size blocks called **frames**.

Advantages

- Eliminates external fragmentation.
- Simplifies memory allocation.
- Supports efficient swapping and virtual memory

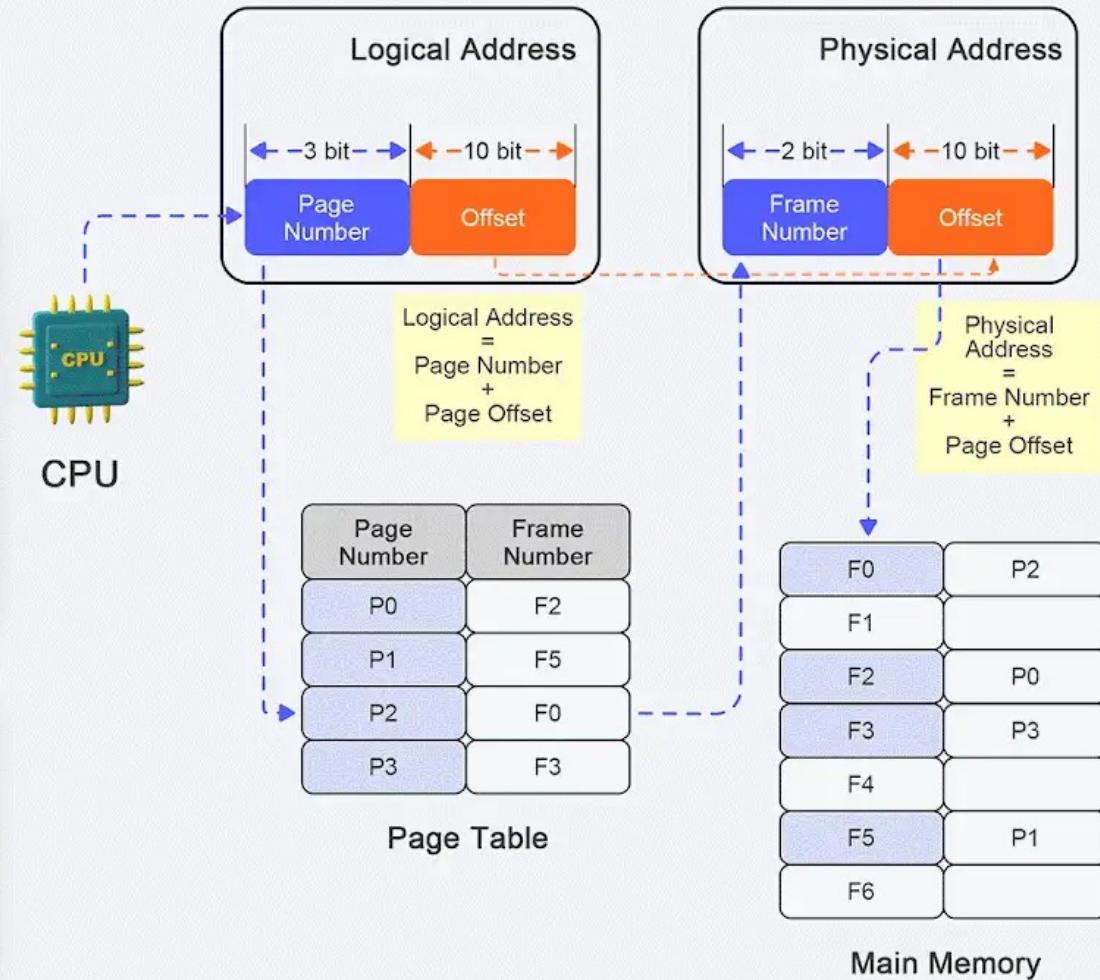




Tabella delle pagine

- La tabella delle pagine ha una riga per **ogni pagina virtuale** del processo
- Contiene
 - l'indice della **pagina fisica**
 - **bit di gestione** (permessi di accesso, etc.)

Elemento della tabella delle pagine

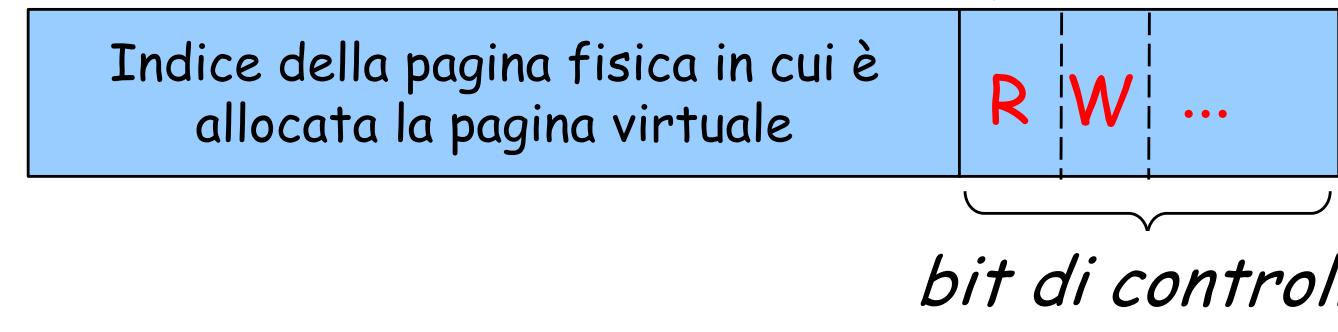
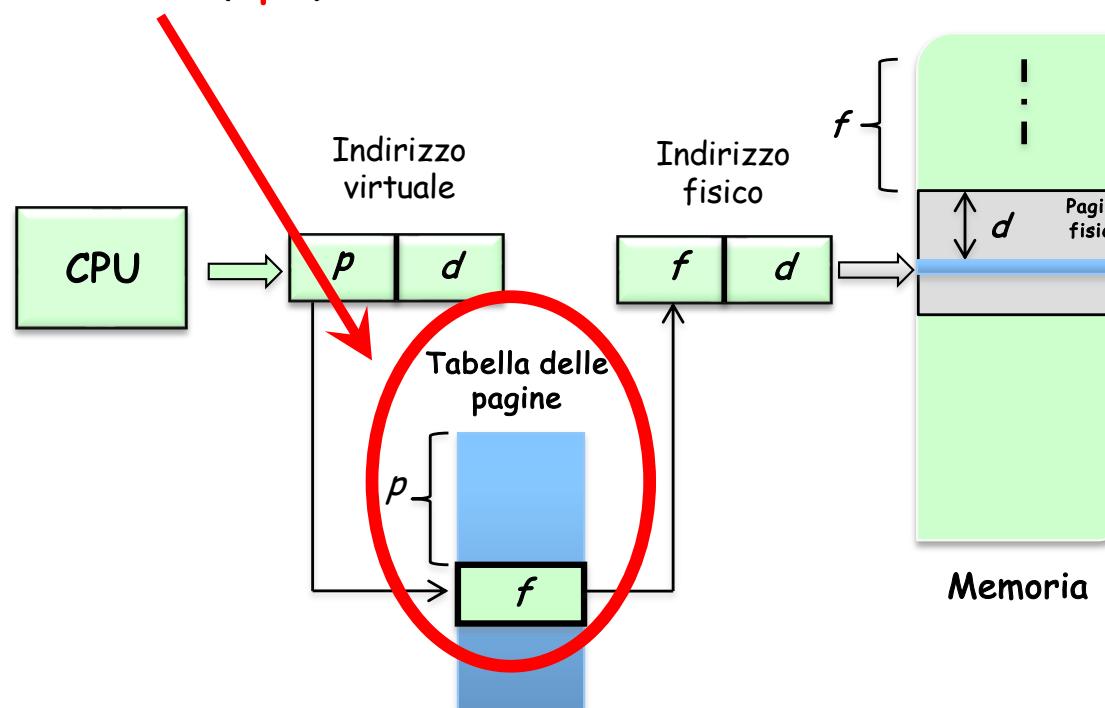




Tabella delle pagine

- La riga contiene l'indice della **pagina fisica ("f")**
- Non contiene l'indice della **pagina virtuale ("p")**



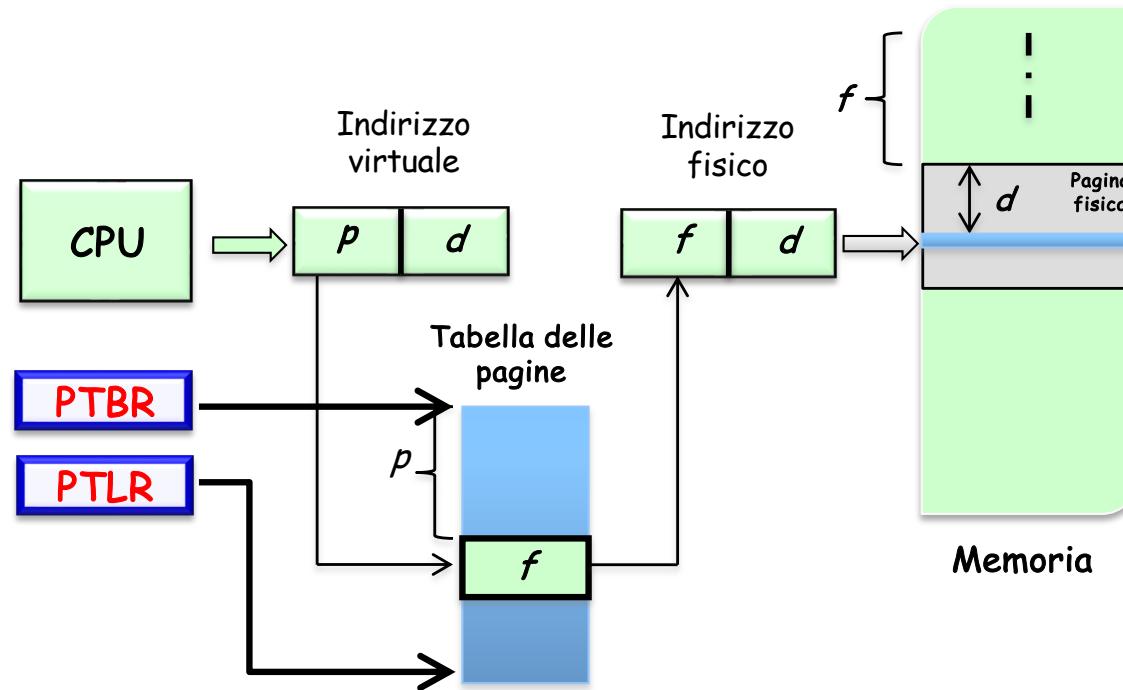


Architettura di paginazione

- La tabella delle pagine è in **memoria principale**
- La MMU usa 2 registri
 - **PTBR (page-table base register)**:
indirizzo base della tabella delle pagine
 - **PTLR (page-table length register)**:
dimensione della tabella delle pagine



Architettura di paginazione



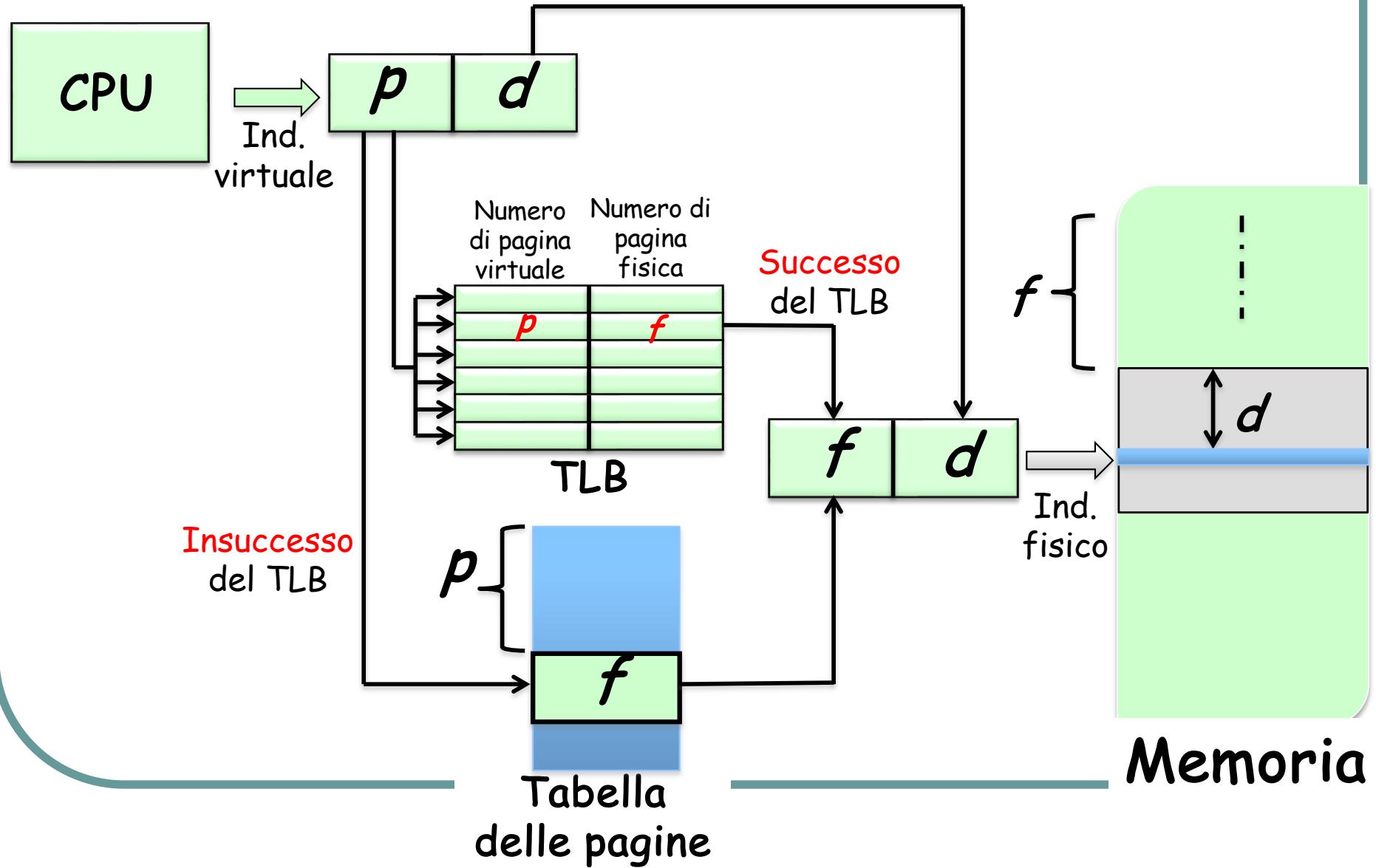


Architettura di paginazione con TLB

- Per accedere alla memoria, occorrono **due accessi**
 - uno per **leggere la tabella delle pagine**
 - uno per **accedere al dato/istruzione vero e proprio**
- **Rallentamento** degli accessi a memoria!
- Per migliorare l'efficienza, si usa una cache associativa detta **TLB (*Translation Look-aside Buffer*)**

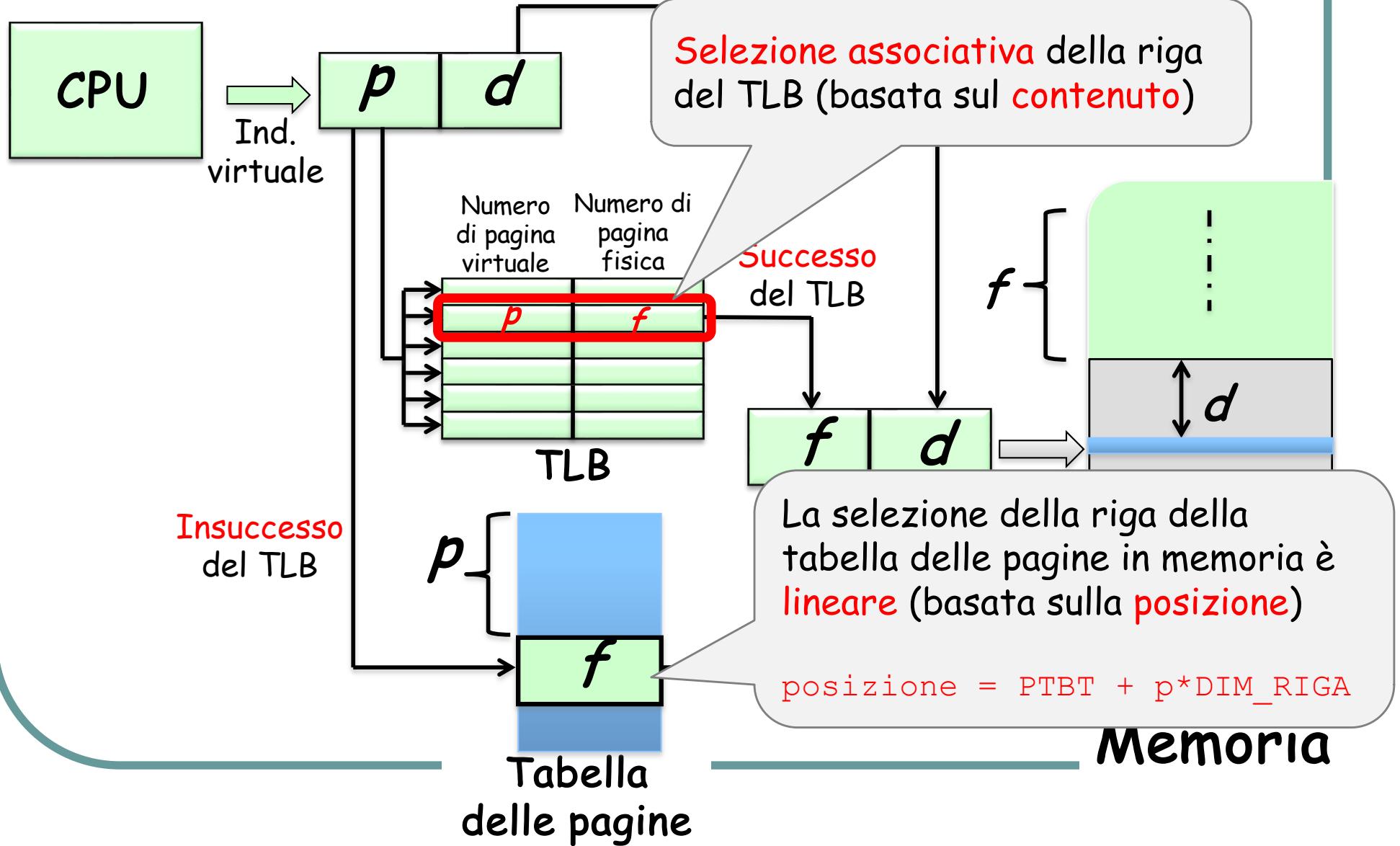


Architettura di paginazione con TLB





Architettura di paginazione con TLB





Tempo effettivo d'accesso

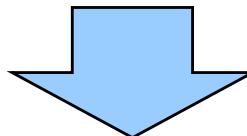
- **Tasso di successo** (*hit ratio*, α): percentuale di volte che un numero di pagina virtuale si trova nel TLB
- Lookup associativo = ε unità di tempo
- Un accesso alla memoria = k unità di tempo
- **Tempo effettivo d'accesso** (*effective access time*)

$$EAT = \underbrace{(k + \varepsilon)\alpha}_{\text{Caso di successo TLB}} + \underbrace{(2k + \varepsilon)(1 - \alpha)}_{\text{Caso di insuccesso TLB}} = (2 - \alpha)k + \varepsilon$$



Dimensione della tabella delle pagine

- Indirizzi a 32 bit
 - Spazio d'indirizzamento = 4Gb (2^{32})
- Usando pagine di 1Kb (2^{10})
 - Dimensione della tabella $2^{22} = 4$ Mb !
 - Frammentazione interna media = 0.5Kb 🤗
- Usando pagine di 64Kb (2^{16})
 - Dimensione della tabella $2^{16} = 64$ Kb 🤗
 - Frammentazione interna media = 32 Kb !



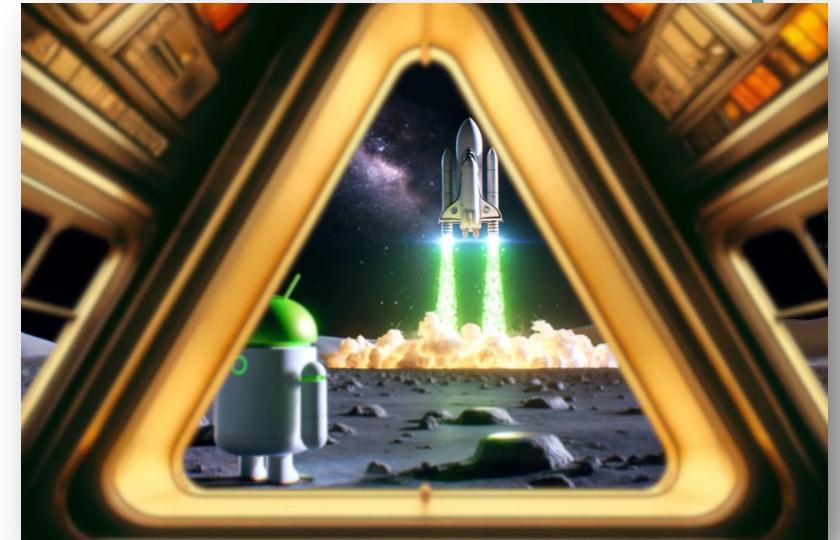
Bisogna scegliere una dimensione di pagina che abbia un buon compromesso tra i due valori



Dimensione della tabella delle pagine

"Android 15 is introducing a significant change in how the operating system manages memory, potentially leading to a 5-10% performance boost.

Traditionally, Android has used a 4 KB page size, but now it will support a larger 16 KB page size, enabled by newer ARM CPUs."



[Android 15's Page Size Change Could Improve Performance](https://www.yahoo.com/tech/android-15s-page-size-change-135617551.html)
<https://developer.android.com/guide/practices/page-sizes>



Dimensione della tabella delle pagine

This shift to a larger page size means less overhead for the operating system when allocating memory to apps.

The result is more resources available for enhancing your videos, games, and overall app experience

Android 15's Page Size Change Could Improve Performance

<https://www.yahoo.com/tech/android-15s-page-size-change-135617551.html>

<https://developer.android.com/guide/practices/page-sizes>



Validità delle pagine virtuali

- Raramente un processo usa tutto il suo spazio di indirizzamento virtuale
 - Dimensione spazio virtuale = $2^{32} = 4 \text{ Gb}$
 - Quantità di memoria usata tipicamente da una applicazione desktop = $\sim 100 \text{ Mb}$

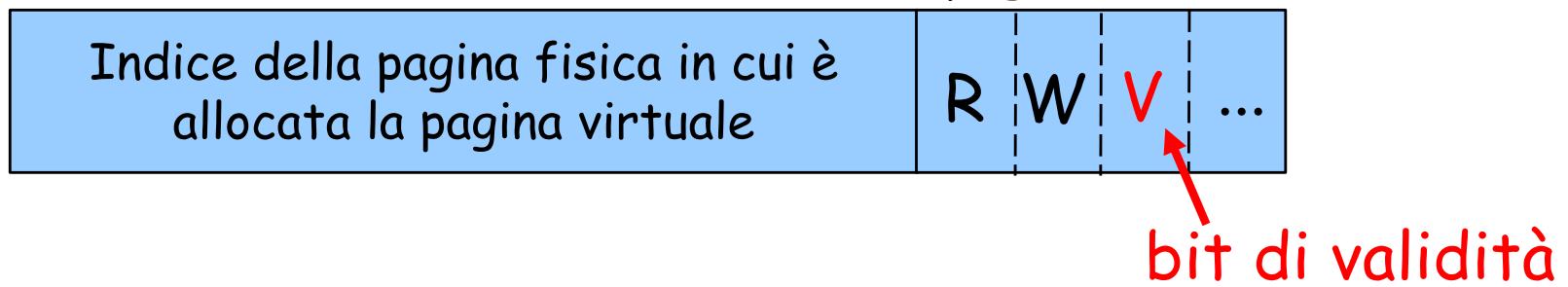
Name	Status	52% CPU	67% Memory	2% Disk	0% Network
Apps (9)					
> Adobe Acrobat Reader DC (32 ...)		0%	59.6 MB	0 MB/s	0 Mbps
> Google Chrome	繁忙	1.3%	2,550.8 MB	0.1 MB/s	0.1 Mbps
> Microsoft Edge	忙	0%	1,099.3 MB	0.1 MB/s	0.1 Mbps
> Microsoft Excel		0%	12.3 MB	0 MB/s	0 Mbps
Microsoft Word (2)		0.5%	264.5 MB	0 MB/s	0 Mbps
Microsoft Word		0.5%	261.7 MB	0 MB/s	0 Mbps
Print driver host for applications		0%	2.8 MB	0 MB/s	0 Mbps
> Settings	忙	0%	0 MB	0 MB/s	0 Mbps
> Snipping Tool		2.0%	4.2 MB	0 MB/s	0 Mbps
> Task Manager		1.1%	27.4 MB	0 MB/s	0 Mbps
> Windows Explorer		1.1%	67.2 MB	0 MB/s	0 Mbps



Validità delle pagine virtuali

- Il SO può marcare le **pagine virtuali in uso** usando un **bit di validità** nella page table
- Il bit viene attivato nel momento in cui la pagina è **allocata** dal processo (es. tramite **malloc()**)

Elemento della tabella delle pagine





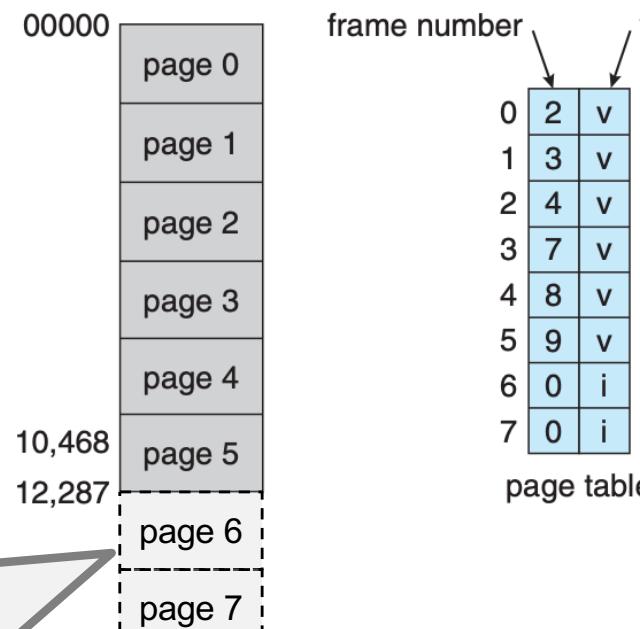
Validità delle pagine virtuali

Esempio:

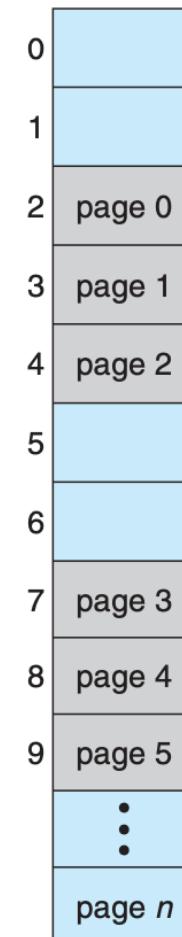
- Indirizzi a 14 bit
- Spazio virtuale: 16Kb
- Dim. pagina: 2Kb

Se il programma tentasse di accedere a queste pagine, la **MMU** genererebbe una **exception**.

Il SO ucciderebbe il processo ("**Segmentation fault**")!



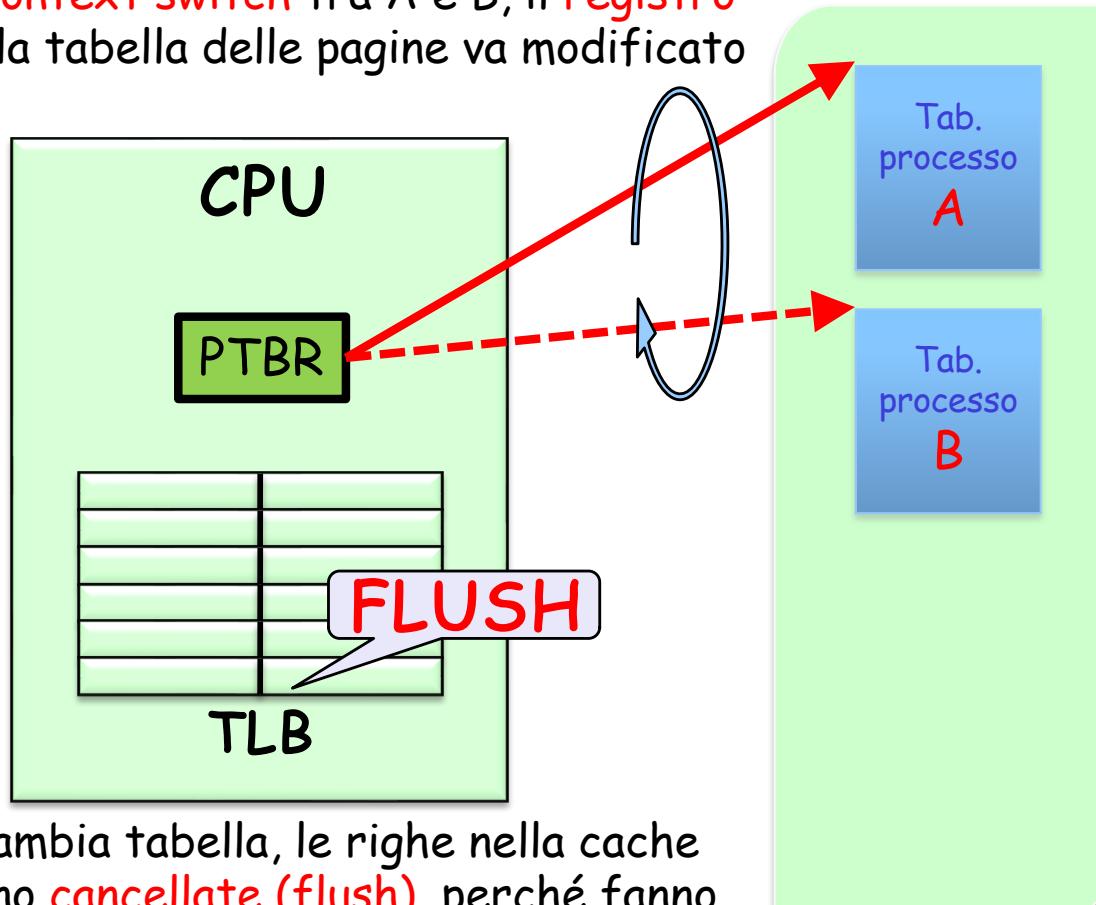
Inutilizzate





Numerosità delle tabelle delle pagine

In caso di **context switch** tra A e B, il **registro base** verso la tabella delle pagine va modificato



Quando si cambia tabella, le righe nella cache del TLB vanno **cancellate (flush)**, perché fanno riferimento alla **tabella precedente**

Ogni processo ha una **tabella differente**

Memoria



Struttura della tabella delle pagine

- Problemi: le tabelle delle pagine
 - hanno grosse dimensioni
 - sono numerose (una per processo)
 - sono "sparse" (poche pagine valide)
- Soluzioni:
 - Paginazione gerarchica
 - Tabella delle pagine basata su hash
 - Tabella delle pagine invertita

Paginazione gerarchica

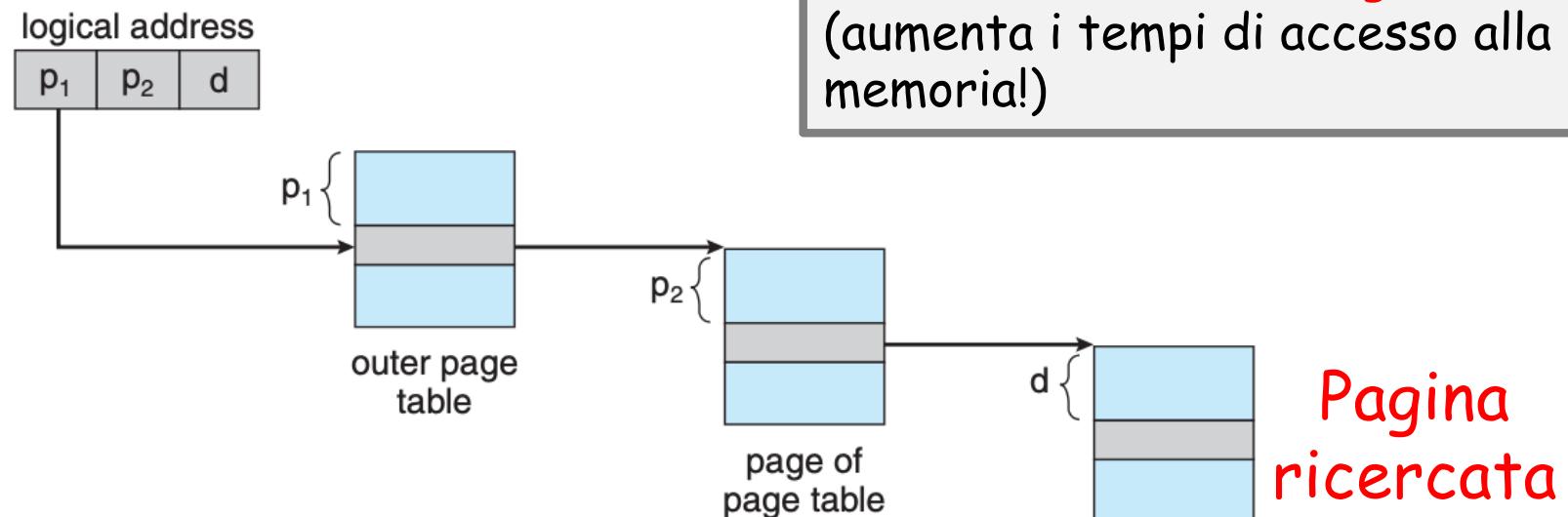


- Suddivide la tabella delle pagine in parti più piccole, secondo una organizzazione **gerarchica**

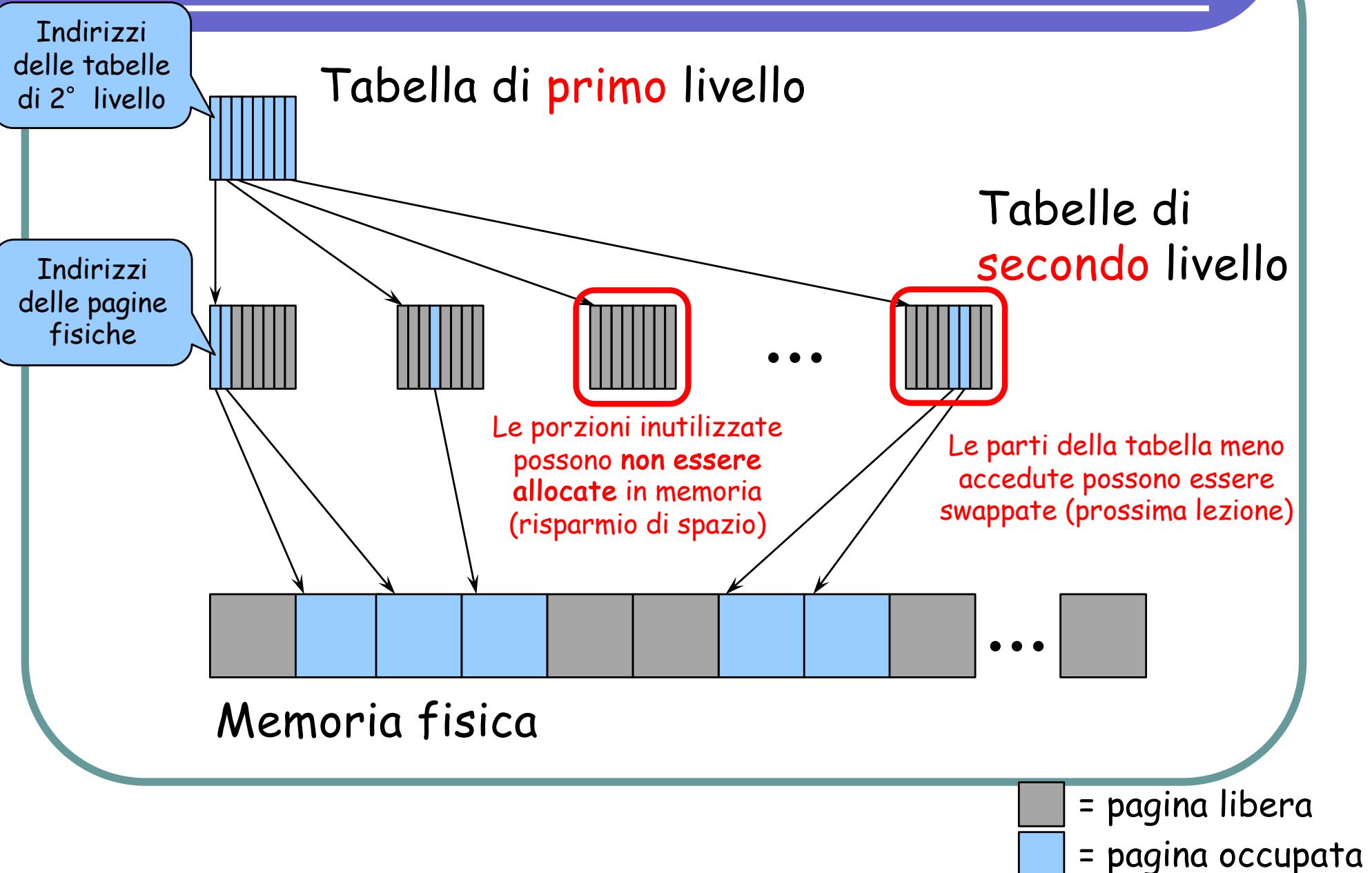


Paginazione gerarchica

- La MMU divide l'**indirizzo di pagina** in più parti (p_1, p_2)
- Nella tabella di primo livello, trova l'indirizzo della tabella di secondo livello



Schema di una tabella delle pagine a due livelli



Schema di una tabella delle pagine a due livelli



- 1 tabella di "primo livello", con N righe
- N tavelle di "secondo livello", ognuna a sua volta con N righe
- Le singole tavelle sono più piccole (N righe) rispetto alla tabella non-gerarchica (N^*N righe)



Esempio di paginazione a due livelli

- Nella paginazione gerarchica, il "numero di pagina" nell'indirizzo virtuale viene **a sua volta suddiviso in più parti**

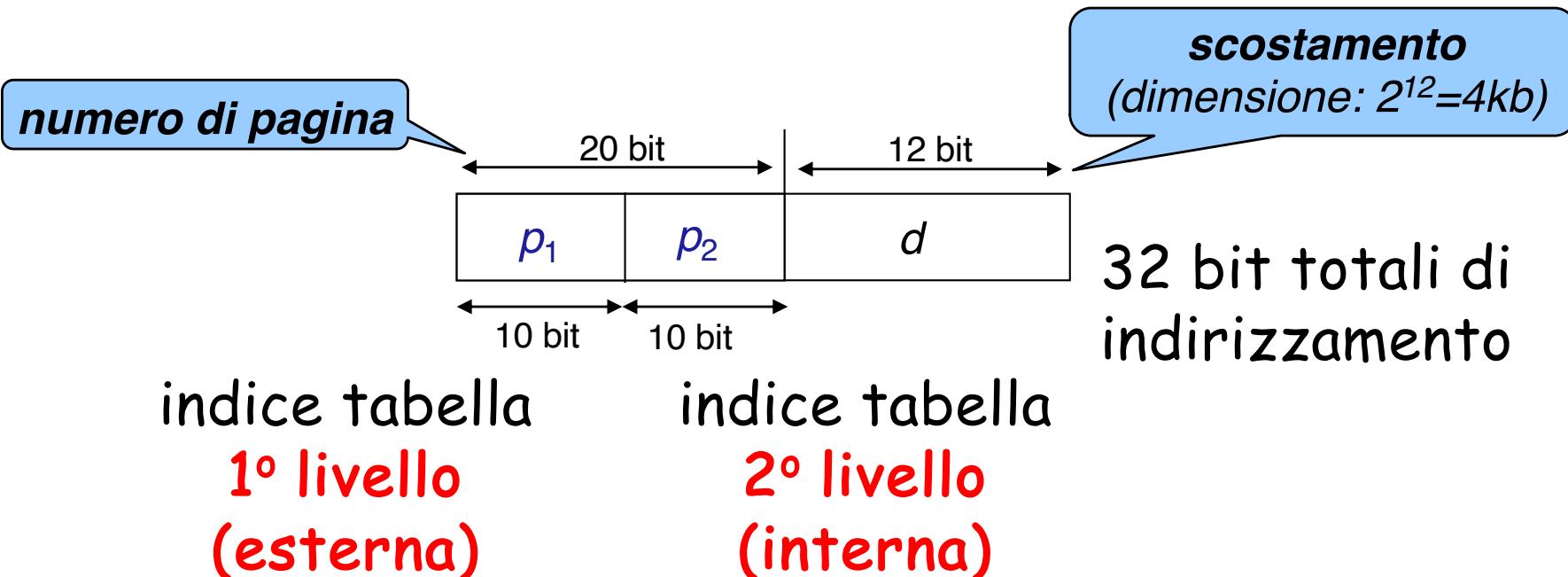


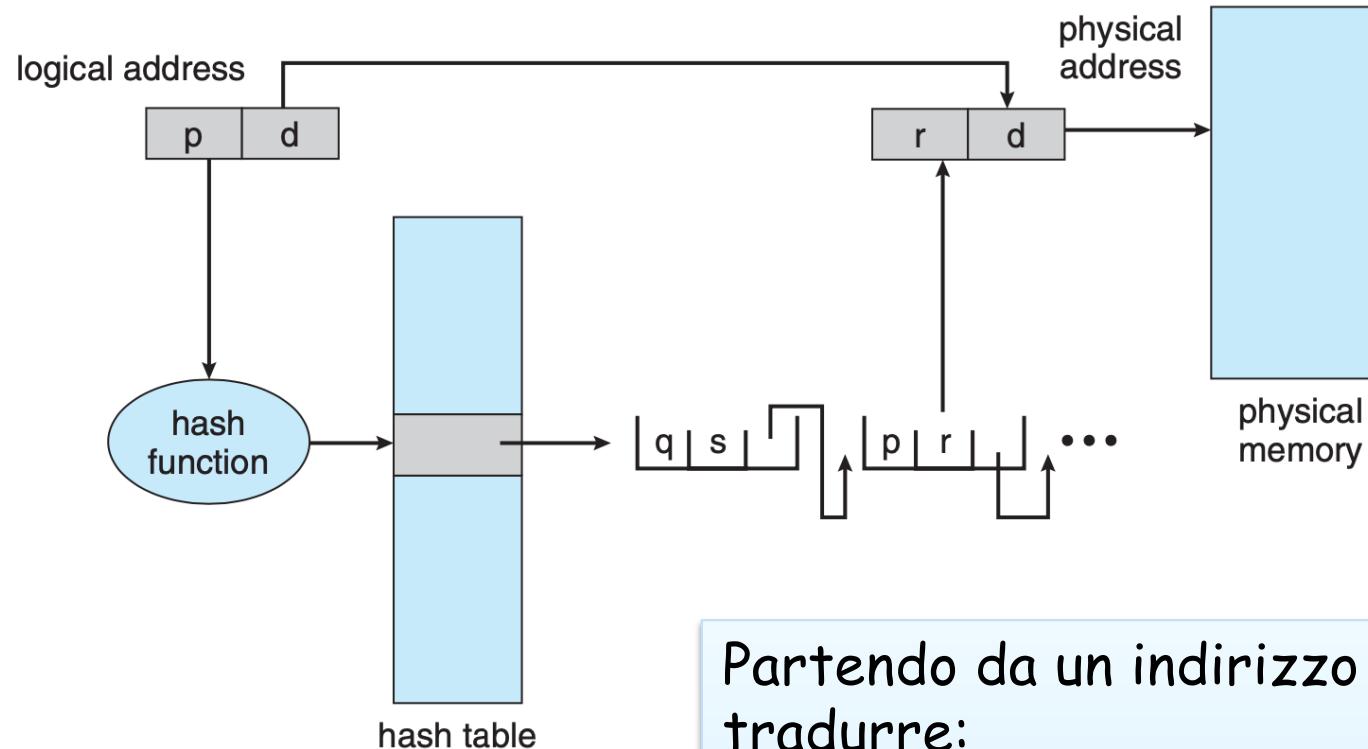


Tabella delle pagine basata su hash

- Le righe della tabella delle pagine sono organizzate usando una **lista concatenata (linked list)**
 - Si memorizzano esclusivamente le righe per le pagine valide
 - Ulteriore **risparmio di memoria**, ma **rallenta la ricerca** (occorre scandire la linked list, ricerca basata sul contenuto)
- Per ottimizzare i tempi di ricerca, si dividono le righe su **tante liste concatenate di piccole dimensioni**
 - Una **funzione di hash** è applicata al num. della pagina virtuale
 - Elementi con lo **stesso valore della funzione di hash** sono collocati nella **stessa lista concatenata**



Tabella delle pagine basata su hash



Partendo da un indirizzo virtuale da tradurre:

- si applica su esso la **funzione di hash** (la stessa usata per dividere le liste)
- viene scandita la **corrispondente lista concatenata** per trovare il num. del blocco di memoria

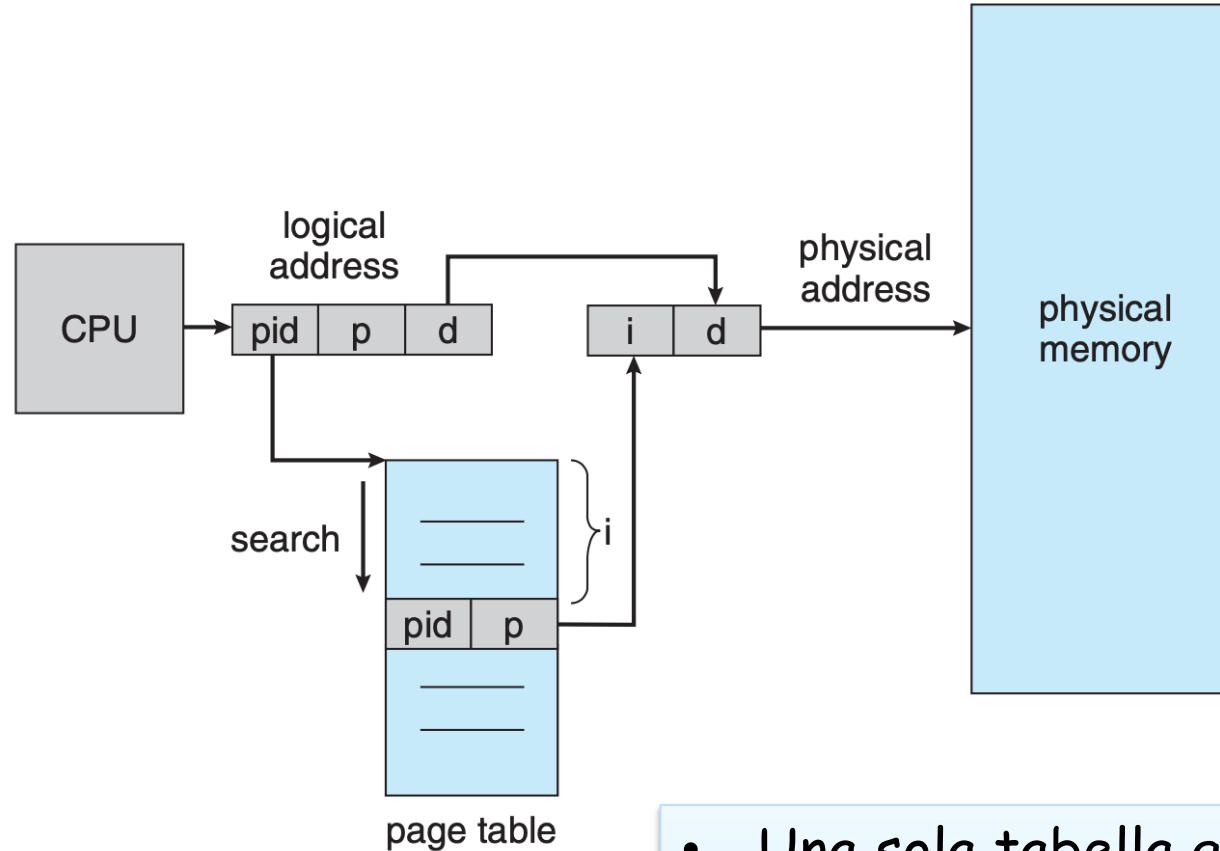


Tabella delle pagine invertita

- Negli schemi precedenti, esiste **una tabella distinta per ogni processo**
- Tabella delle pagine **invertita**
 - **Una** sola tabella delle pagine **comune a tutti i processi**
 - Questa tabella ha un elemento **per ogni pagina fisica**
 - Ogni elemento contiene l'indirizzo **virtuale** della pagina memorizzata **in quella locazione fisica**, con informazioni sul processo che possiede tale pagina
 - Utilizzata nelle architetture UltraSparc (64 bit) e Power PC

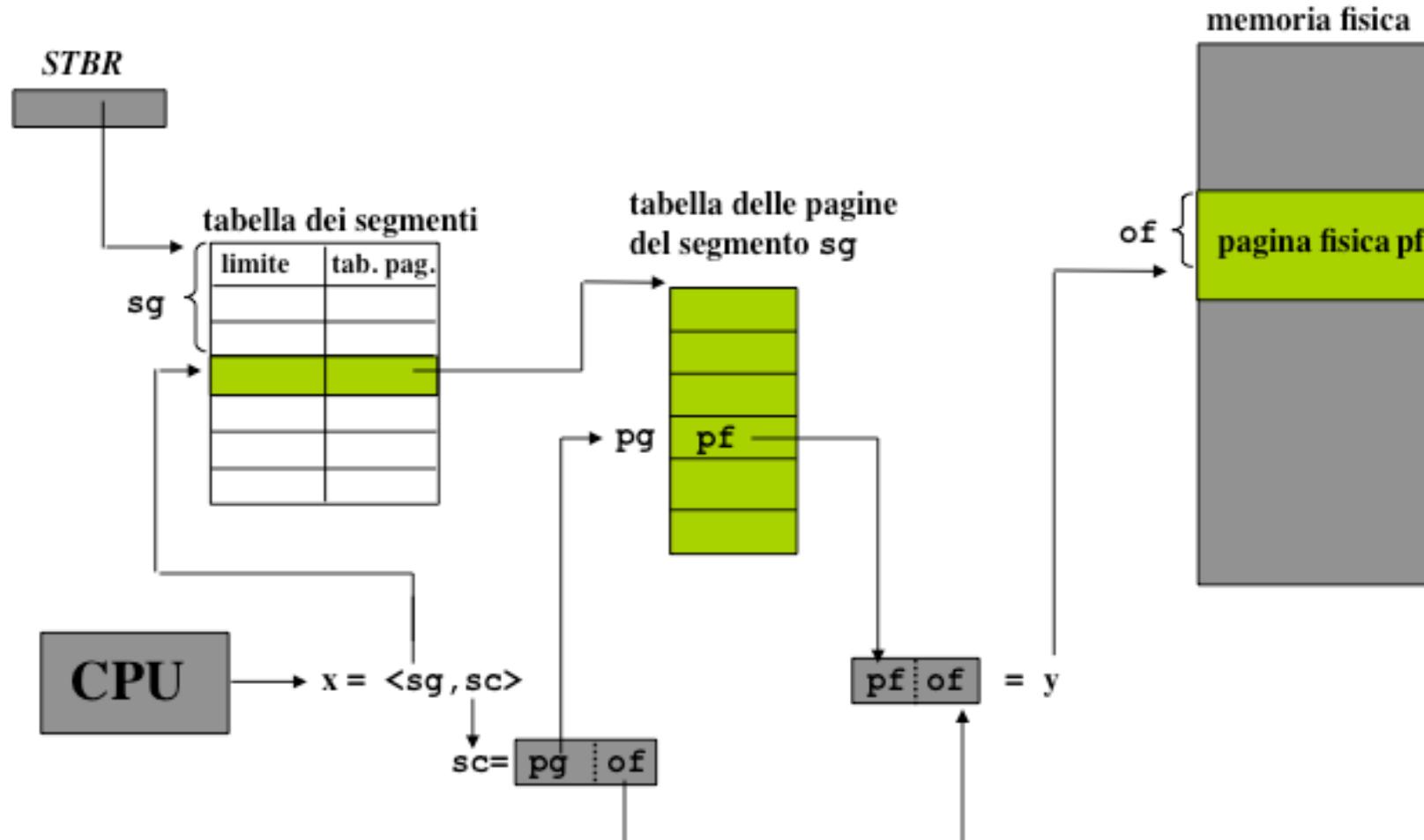


Tabella delle pagine invertita



- Una sola tabella globale
- Il numero di righe è pari al numero di pagine fisiche (invece che virtuali)

Segmentazione paginata



Quiz



1. In un sistema con paginazione, con indirizzi a 32 bit e con pagine di dimensione di 2Kb (2^{11}), quante pagine virtuali può avere un processo?

- Circa 2 migliaia (2^{11})
- Circa 1 milione (2^{20})
- Circa 2 milioni (2^{21})
- Circa 4 miliardi (2^{32})

2. Dopo un cambio di contesto, in alcune CPU è necessario svuotare il contenuto del TLB, anche a costo di penalizzare le prestazioni della CPU

- Vero
- Falso



<https://forms.office.com/r/eT2Kr0eZG7>

Quiz



3. In un sistema con paginazione, la MMU consulta la tabella delle pagine mediante:

- Selezione lineare (indirizzo base sommato a uno scostamento)
- Selezione associativa (itera sugli elementi della tabella, alla ricerca del valore di pagina virtuale)



<https://forms.office.com/r/eT2Kr0eZG7>