

## INTRO A MEMORIA ➤ 9.1.1

La memoria può essere divisa in

- mem. principale: In cui si caricano dati ed istruzioni del prog. in esecuzione. ( $\geq 1 \text{ clock}$ )
- mem. secondaria: In cui si registrano i programmi
- registri: Memorie ad alta velocità ( $\leq 1 \text{ clock}$ )
- cache: Usata per accelerare l'accesso in memoria.

La mem. princ. e i registri sono le uniche memorie ad accesso diretto della CPU.

### PROTEZIONE DELLA MEMORIA

L'hardware deve assicurare una protezione delle memorie riservate ai processi da accessi non autorizzati. Per fare questo si può:

#### USARE REGISTRI BASE E LIMITI

Per ogni processo si definiscono i registri

- base: con l'indirizzo di partenza
- limit: con la dimensione dei registri

Violazioni degli spazi di indirizzamento vengono rilevati dall'hardware e lanciati come tripleccezioni al S.O.

### ADDRESS BINDING 9.1.2

I S.O. moderni multitasking virtualizzano la memoria, allocando ai programmi degli indirizzi logici che non è detto corrispondano a quelli fisici.

Gli indirizzi logici per ogni programma iniziano da 0000 e l'associazione agli indirizzi fisici può essere a:

- Compile Time: Indirizzamento assoluto → Ricompilare
- Load Time: Associazione nel Loader.

Per cambiare indirizzi sarà necessario riaprire

- Execution Time: Se durante l'esecuzione il processo

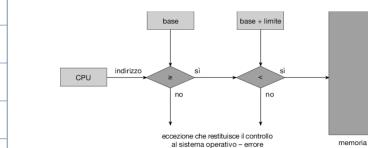
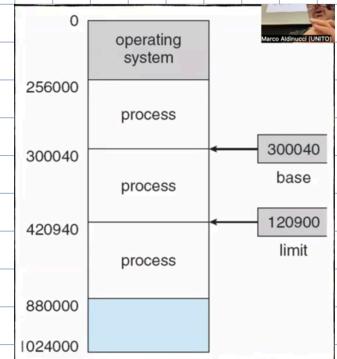


Figura 9.2 Protezione hardware degli indirizzi tramite registri base e limite.

può essere spostato da un segmento all'altro.

L'associazione a compile e load time producono sempre indirizzi logici **uguali** a quelli fisici.

Per l'execution time, viene impiegata la Memory Management Unit (M.M.U.).

Quando un processo utente genera un indirizzo, prima dell'invio in memoria viene sommata una relocation constant (ind. di base) presente in un relocation register all'indirizzo fisico della R.R.EU.

### USO DEL RELOCATION REGISTER

Eseguendo più programmi, il relocation register dovrà essere caricato con valori diversi a seconda del processo.

La relocation constant di ogni processo andrà salvata nel PCB per poter fare i context switch.

Ad ogni switch, la r.constant del prossimo processo dovrà essere caricata nel r.register per poter accedere alla mem del prossimo (e anche per performance).

## CONTIGOUS MEMORY ALLOCATION > Q.2

In memoria vengono caricati sia i processi dell'SO che utente. Il S.O. ha la possibilità di collocarsi nella parte alta o bassa della memoria.

La contiguous mem. allocation è una policy che prevede che ogni processo abbia una porzione continua e unica di memoria assegnata (no interruzioni).

+ vantaggi: Possibilità di uso della logica dei puntatori <sup>→ esterna</sup>

- vantaggi: Può portare a fragmentation in cui si creano aree piccole fra le mem. dei processi non utilizzabili.

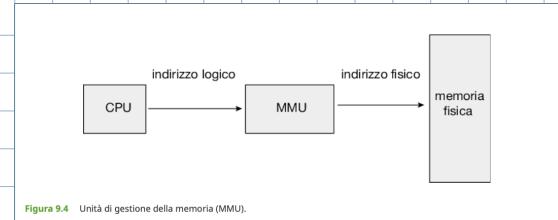


Figura 9.4 Unità di gestione della memoria (MMU).

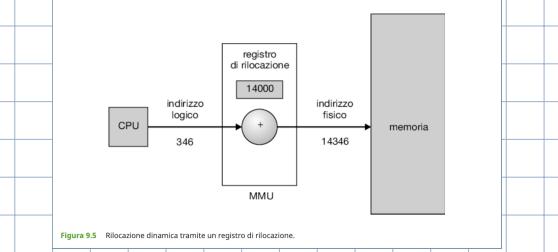


Figura 9.5 Relocalazione dinamica tramite un registro di rilocazione.

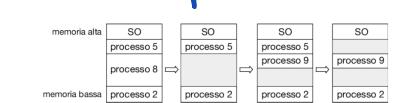


Figura 9.7 Schema di partizione variabile.

La ricerca dell'area di mem può avvenire con diversi mechanism:

- First-fit: Assegno il primo buco disponibile  $O(n)$
- Best-fit: Assegno nel buco più piccolo. Può peggiorare la fragmentation creando buchi sempre più piccoli
- Worst-fit: Assegno nel buco più grande. Può migliorare la fragmentation creando buchi più grandi

Sia first che best fit soffrono di frammentazione esterna.

## FRAGMENTATION 9.2.3

La fragmentation può essere di 2 tipi

- esterna : Dovuta a spazi troppo piccoli liberi non contigui.
- interna :

Parti di memoria inutilizzate assegnate ai processi per "arrotondare per eccesso" le richieste dei processi stessi.

## COMPACTON

Metodo risolutivo della frammentazione che prevede di riordinare il contenuto della memoria. È fattibile solo se il binding è a execution time. È onerosa.

## SEGMENTATION >

È una policy alternativa che consente che l'indirizzamento fisico non sia contiguo.

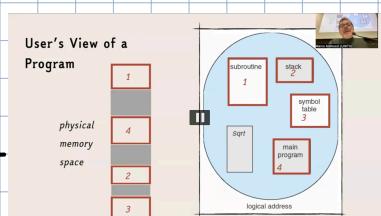
Divide il programma in segmenti non divisibili (es: funzioni stack)

## ARCHITETTURA

Ogni indirizzo avrà un number ed un offset.

I segmenti sono salvati in una segment table che conterrà per ognuno base e limit.

La table a sua volta avrà bisogno di base (STBR) e length (STLR) registri per salvataggio



La segmentation consente di condividere vari segmenti comuni  
(es: printf();)

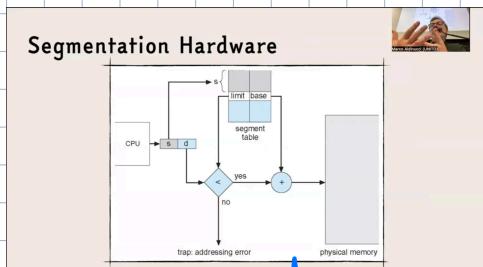
## IMPLEMENTAZIONE BINDING

In questo caso, per ogni indirizzo bisognerà contattare la table per prendere il segment per poi sommare l'offset.

A differenza delle contiguous mem. allocation, non c'è un momento ideale in cui storizzare il segmento e in teoria si dovrebbe fare ad ogni indirizzo.

Un'opzione può essere usare una tabella di registri che però non è una soluzione definitiva → Registri non sufficienti

$d = \text{offset interno al}$   
 $s = \# \text{ segmenti}$



$$\text{ind} = \text{ind} / 2^d = s + r(d)$$