

## GESTIONE MEMORIA

Definizione: Definiamo il modulo della *Gestione delle Memorie* come una MV il cui compito è di distribuire opportunamente la memoria centrale, in modo trasparente ed efficiente, al fine di risolvere le esigenze dei vari processi.

Principali compiti:

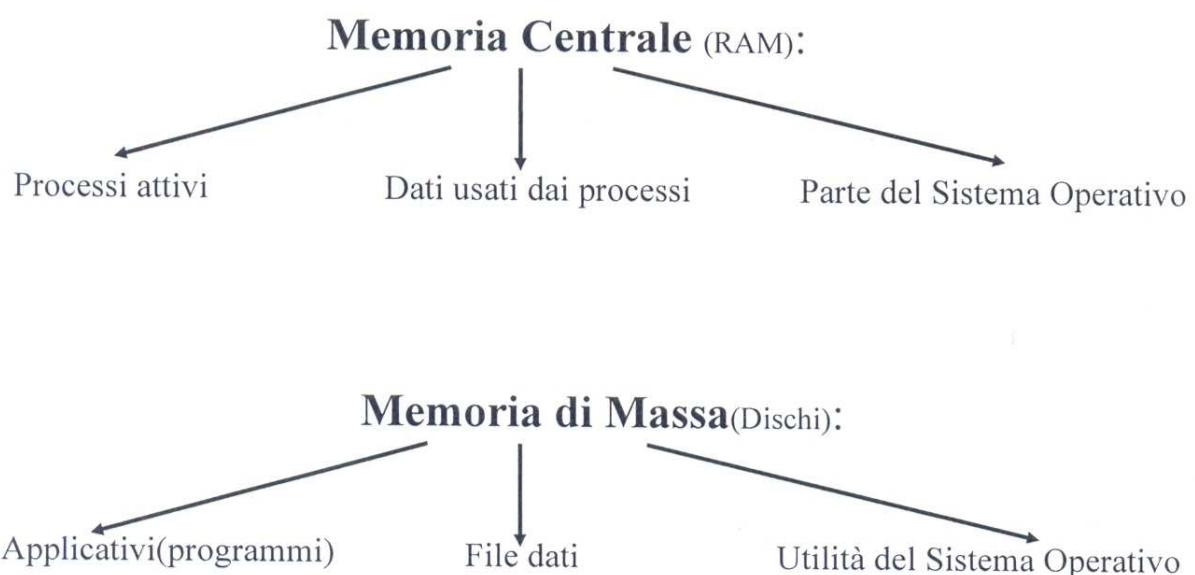
- Proteggere dati e istruzioni.
- Mascherare la locazione fisica dei dati.
- Può sovrapporre gli spazi di memoria associati ai programmi.

Garanzie:

- Garantisce che nessun altro processo possa leggere o modificare i dati contenuti nello spazio di indirizzamento.
- Garantisce l'indirizzamento della memoria centrale più esteso di quello ammissibile (uso della memoria di massa).
- Garantisce la non ridondanza dei dati o dei programmi.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## GESTIONE MEMORIA



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## GESTIONE MEMORIA CENTRALE

### Perché un gestore della memoria centrale?

- Presenza della multi-utenza e della multi-programmazione.
- Presenza di processi di dimensioni più grandi della memoria centrale libera.
- Presenza di dati con dimensioni più grandi della memoria centrale libera.
- Ridurre la quantità di dati e dei processi presenti in memoria centrale.

### Compiti del gestore della memoria centrale

- Risolve le problematiche relative alla multi-programmazione e alla multi-utenza.
- Risolve i problemi di incompatibilità tra dimensione del programma e memoria disponibile.
- Risolve le problematiche relative allo spostamento dell'applicativo tra memoria e CPU

### Strategie

- Consentire il caricamento dei programmi a partire da un indirizzo qualunque.
- Ridurre lo spazio di memoria conservando solo quelle porzioni di processo necessarie.
- Condividere insiemi di istruzioni tra più processi.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Gestione della Memoria

Idealmente si vorrebbe avere per ogni elaboratore una memoria infinitamente grande e infinitamente veloce.

Sfortunatamente ciò non è possibile e la maggior parte degli elaboratori hanno una **Gerarchia di Memoria**:

- Una piccola quantità di cache costosa ma veloce (volatile)
- Una memoria Centrale meno costosa ma anche meno veloce (volatile)
- Una unità a disco molto economica ma lenta (non volatile)

La parte del S.O. che gestisce la Memoria centrale si chiama **Gestore della Memoria**. I suoi compiti sono:

- Tenere conto delle parti di memoria usate e di quelle non utilizzate
- Allocare memoria ai processi quando ne fanno richiesta
- Liberarla quando hanno finito
- Gestire gli scambi tra memoria centrale e disco

I sistemi di gestione della memoria possono essere divisi in due classi:

- Quelli che spostano i processi dalla memoria ai dischi e viceversa
- Quelli che non lo fanno

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

Lo Swapping e la Paginazione sono strategie dettate dall'insufficienza della memoria centrale a contenere tutti i programmi.

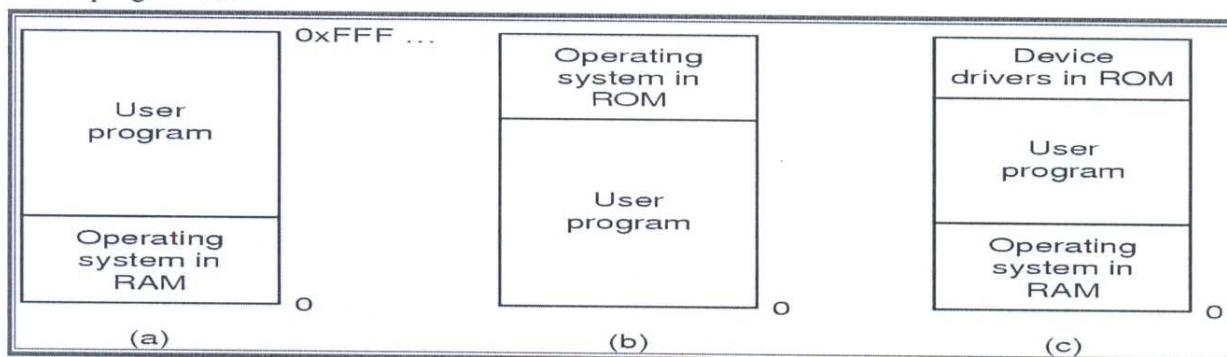
In prima istanza ci occuperemo di un semplice gestore della memoria senza swapping e pagging.

### Un semplice schema

- Lo schema di gestione della memoria più semplice possibile è quello di eseguire soltanto un programma alla volta condividendo la memoria tra programma e sistema operativo.

In uno schema di questo tipo, in un generico istante di tempo, solo un processo può essere caricato in memoria.

Quando un utente richiede l'esecuzione di un programma, attraverso il Prompt dei comandi, questo viene caricato in memoria, al termine del quale la memoria sarà rilasciata restando in attesa del caricamento di un nuovo programma.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Multiprogrammazione con partizioni fisse

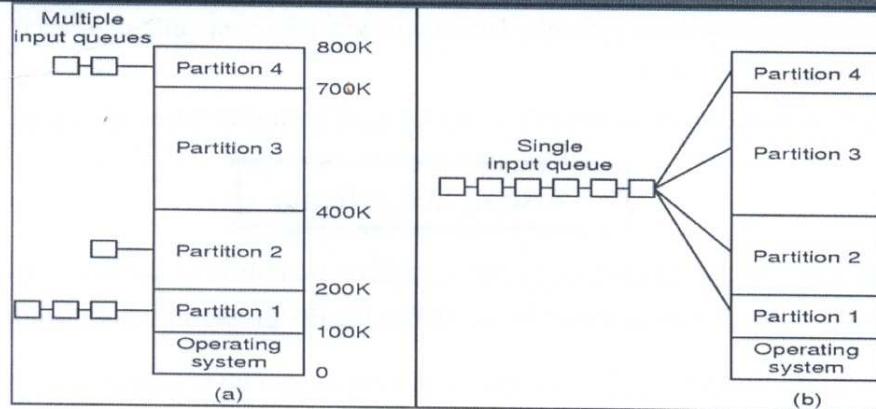
In questo momento quasi tutti i sistemi di elaborazione consentono ai diversi processi di girare contemporaneamente sulla CPU.

Contemporaneamente nel senso che, quando un processo è bloccato (in attesa di una risorsa) la CPU viene assegnata ad un altro processo.

Il modo più semplice per realizzare la multiprogrammazione è di dividere la memoria in n partizioni di dimensioni diverse.

**Come definire e quando effettuare la partizione:** la partizione può essere effettuata quando il sistema sta per attivarsi ed è l'operatore di sistema che può definire le partizioni.

**Come agisce il processo sulla partizione:** il gestore della memoria assegna al processo la più piccola partizione di memoria utile per farlo lavorare, poiché la partizione non è dinamica, al processo sarà assegnata una parte di memoria non utilizzata.



**Svantaggio delle code multiple:** la partizione della memoria in aree diverse, ciascuna con una dimensione diversa dall'altra, implica che in certi momenti alcune partizioni non sono utilizzate in quanto prive di processi in coda, mentre altre saranno estremamente affollate.

**Prima Alternativa:** una singola coda per i processi in attesa della memoria, ogni volta che si libera una partizione viene **caricato il job più vicino alla testa della coda che può entrare nella partizione**.

Poiché non è desiderabile sprecare partizioni grandi per processi piccoli cosa frequente in tale alternativa, una seconda opzione è data dalla seguente:

**Seconda Alternativa:** scegliere tra la coda dei processi in attesa della memoria il processo **più grande che può entrare**. Lo svantaggio di tale strategia e quello di **NON agevolare i processi piccoli**, che di solito sono quelli interattivi e che quindi dovrebbero essere più agevolati.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

200

**Terza alternativa:** Due code una per i processi piccoli e l'altra per i processi più grandi.

**Quarta Alternativa:** Assegnare a un Job un valore numerico che stabilisce il numero di volte che il processo è stato messo da parte. Superata una soglia il processo viene eseguito.

Un gestore della memoria per la multi-programmazione basato su partizioni fisse, qualsiasi sia l'alternativa utilizzata, è IMPENSABILE negli attuali sistemi di elaborazione.

L'attività di programmazione introduce due problemi essenziali che devono essere risolti:

Rilocazione

Protezione

**Esempio:** quando viene effettuato un link, il linker deve conoscere l'indirizzo dove sarà caricato il programma.

**Soluzione 1:** cambiare l'indirizzo ad ogni istruzione. Ai programmi caricati nella partizione 1 si somma il valore 100, a quelli in posizione 2 si somma il valore 200 e così via.

## La rilocazione non risolve i problemi della protezione

La protezione fu risolta, sul 360 dell'IBM, assegnando ad ogni partizione di 2KB un codice di 4 bit (PSW) che il gestore della memoria controlla ogni volta che un processo effettua un accesso ad una specifica locazione di memoria.

**Soluzione 2:** Sono stati definiti due registri Hardware (base, limite) che rappresentano il campo di azione del programma e contengono rispettivamente l'indirizzo dove il programma è stato caricato e il suo limite di azione (indirizzo finale). Tale soluzione risolve contemporaneamente i problemi di rilocazione e di protezione.

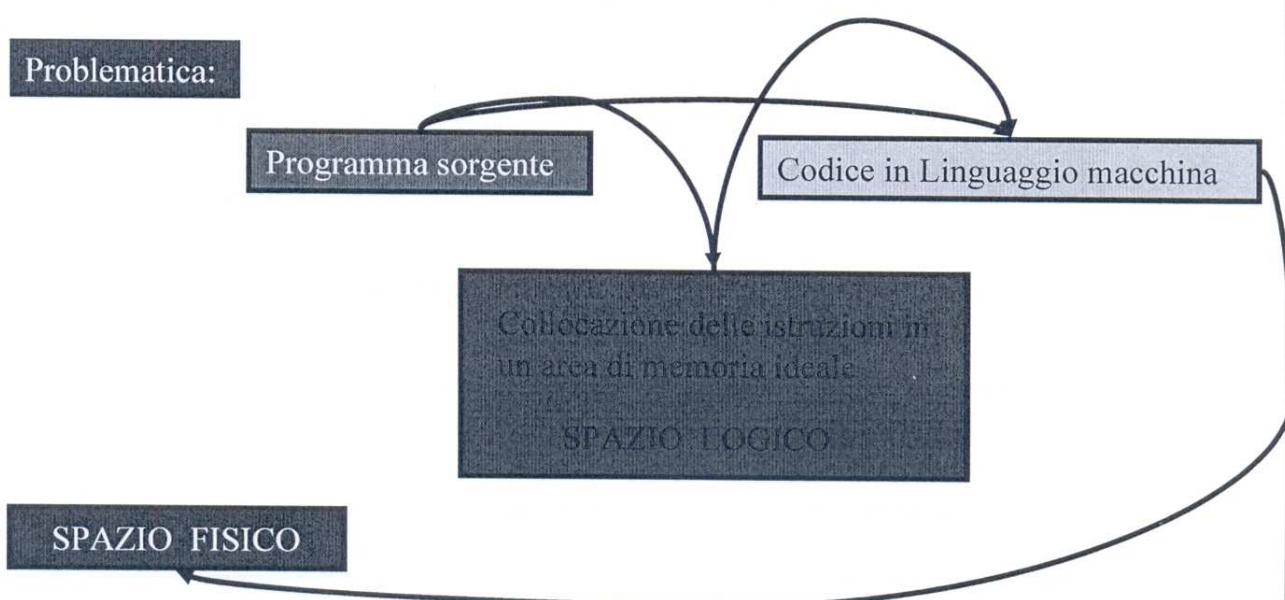
L'Hardware protegge i registri BASE e LIMITE per evitare che qualcuno li possa modificare.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

# GESTIONE MEMORIA

## La rilocabilità del codice dell'applicativo

Problematica:



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## GESTIONE MEMORIA

**SPAZIO LOGICO**

**NON  
COINCIDONO**

**SPAZIO FISICO**

La memoria *ideale* non coincide con la memoria *reale* in cui saranno caricati i codici durante l'esecuzione.

Definiamo *Rilocazione* del codice la trasformazione che permette di caricare il codice di un applicativo a partire da una zona di memoria arbitraria.

Definiamo *Spiazzamento o Offset* il valore sommato agli indirizzi logici calcolato come differenza tra il valore che effettivamente assumerà nella memoria centrale e l'indirizzo generale di partenza degli applicativi (generalmente è 0).

LOGICO		SPAZIO FISICO	
1	Istruzione 1	2AB1	Istruzione 1
2	Istruzione 2	2AB2	Istruzione 2
3	Istruzione 3	2AB3	Istruzione 3
4	Istruzione 4	2AB4	Istruzione 4

SPAZIO FISICO			
0000	Istruzione	2AB1	Istruzione 1
0001	Istruzione	2AB2	Istruzione 2
•	Istruzione	2AB3	Istruzione 3
2AB0	Istruzione	2AB4	Istruzione 4

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## GESTIONE MEMORIA

### RILOCAZIONE

Rilocazione Statica: La rilocazione *statica* avviene in fase di creazione del codice in linguaggio macchina.

Nota: la rilocazione statica è accettabile in sistemi dedicati o uniprogrammati, in cui il S.O. risiede sempre nella stessa partizione di memoria e i programmi sono caricati in indirizzi di memoria fissi.

Rilocazione Dinamica: La rilocazione *dinamica* avviene durante l'esecuzione dell'applicativo.

Nota: la rilocazione dinamica è necessaria quando si è in presenza di sistemi multiprogrammabili dove i processi si creano e si distruggono in maniera dinamica e le zone di memoria sono continuamente utilizzate da processi diversi, da dati diversi e da utenti diversi.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## SWAPPING

Problematica:

Immaginiamo che la dimensione del codice da eseguire sia maggiore della dimensioni della memoria centrale libera.....e più in generale immaginiamo di avere un numero di processi tale che la somma delle dimensioni del loro codice sia maggiore della memoria centrale disponibile.

I questi casi il S.O. NON è in grado di caricare il processo, sia nel primo sia nel secondo caso dovrà attendere la fine di qualche processo per liberare la memoria.

Soluzione:

La soluzione tipica per evitare questi problemi è abilitare il S.O. a trasferire il contenuto di un'area di memoria centrale in un area della memoria di massa (tipicamente della area di swap) al fine di rendere disponibile la memoria resa libera.

Questa tecnica, denominata tecnica di swapping, generalmente è applicata ai processi che sono in attesa.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Swapping

Nei sistemi timesharing o nei personal computer, in generale, la memoria centrale non è sufficiente a mantenere tutti i processi che richiedono di andare in esecuzione.

La soluzione a tale problema è quella di mantenere i processi attivi, che non rientrano in memoria, sul disco e di richiamarli quando possono utilizzare la CPU.

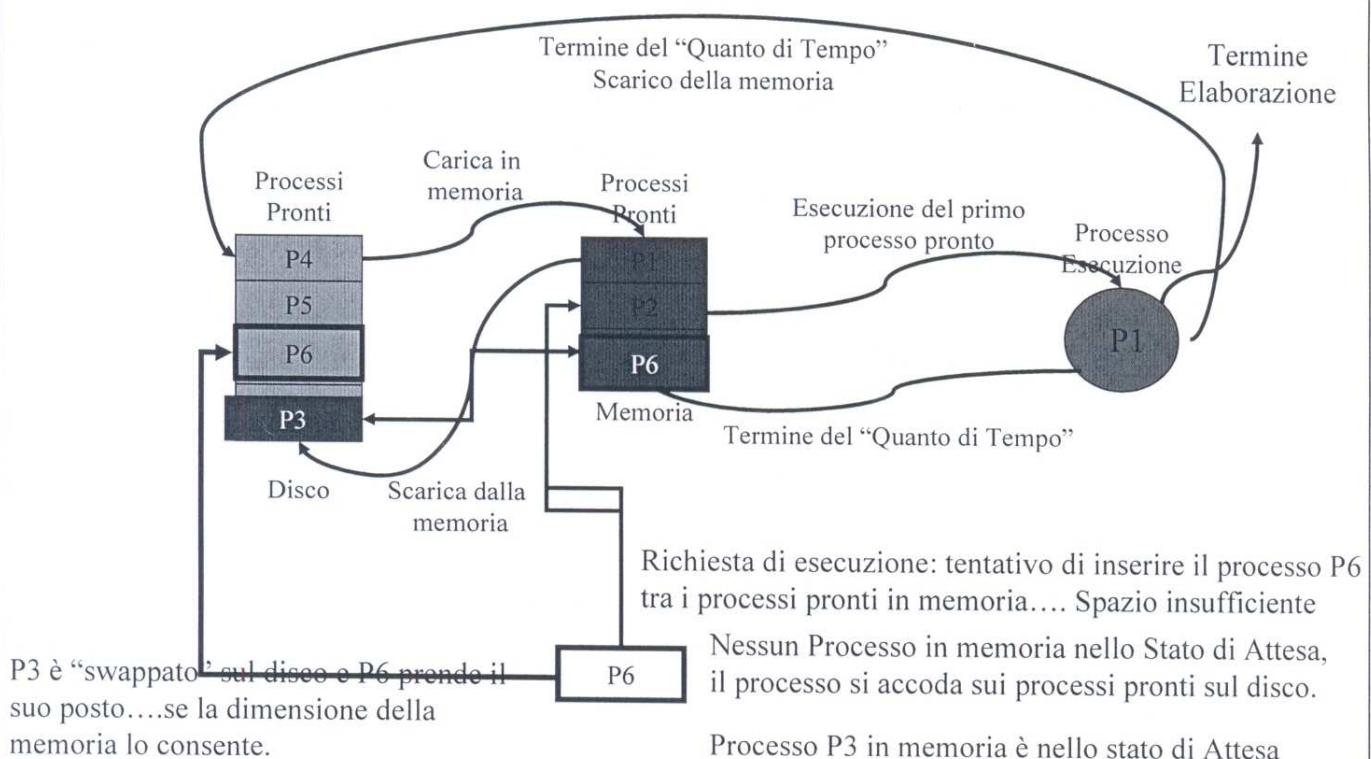
### Strategia SWAPPING

Questa strategia, chiamata **SWAPPING**, carica in memoria l'intero processo, lo manda in esecuzione per un tempo minore o uguale al suo quanto di tempo, quindi lo sposta nuovamente in una speciale memoria sul disco chiamata swap disco.

### Strategia MEMORIA VIRTUALE

Questa strategia, chiamata **MEMORIA VIRTUALE**, consente di caricare in memoria centrale solo porzioni del processo.

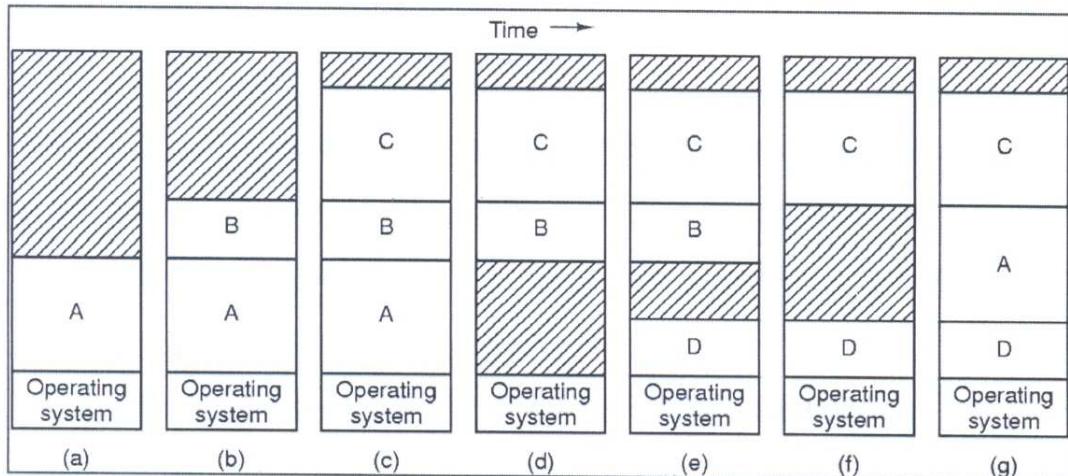
Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Swapping

- il Processo A viene caricato in memoria.
- i Processi B,C sono anch'essi caricati in memoria (notare la dinamicità della memoria)
- il processo A viene scaricato dalla memoria per fare posto ad altri processi
- viene caricato il Processo D
- il processo B viene scaricato liberando una porzione di memoria in cui può essere caricato il processo A
- viene caricato il Processo A in una posizione diversa da quella originale.



Nota: in questo contesto il Sistema Operativo sta nella parte bassa della Memoria.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

La flessibilità è data dal non essere legati alla specifica dimensione della partizione della memoria ma alle dimensioni del processo.

- La flessibilità crea qualche problema di rilocazione e di rilascio della memoria.
- Si deve tener conto di quale parte della memoria è stata resa libera.

Due problemi possono sorgere nel caricamento dei processi in memoria:

- La gestione della comparsa di buchi di memoria non usati da nessun processo.
- La gestione di quei processi che hanno dimensione variabile nel tempo (variabili: crescono dinamicamente)

Caricare e scaricare processi in e dalla memoria può provocare buchi di memoria non riutilizzabili, per evitare tali disfunzioni il gestore della memoria ha un modulo preposto a compattare i buchi presenti in memoria: Compattazione della Memoria.

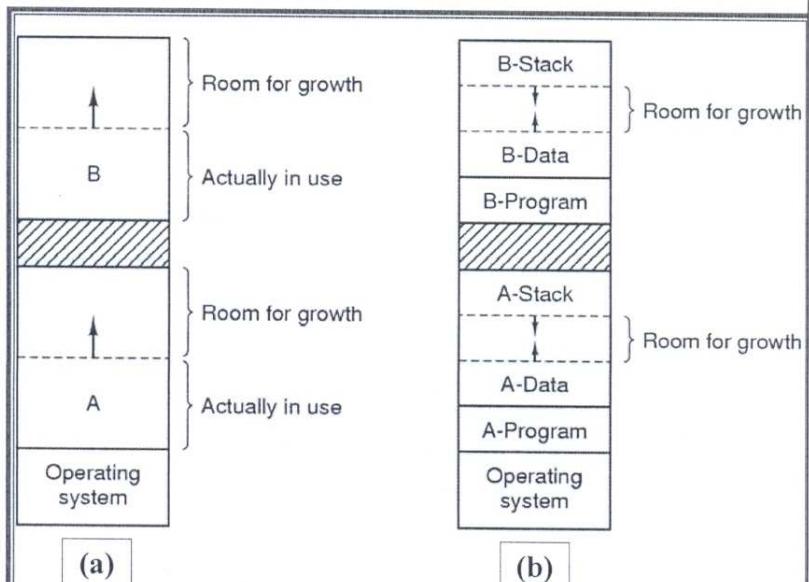
NOTA: Questa tecnica raramente viene usata in quanto necessita di molto tempo di CPU.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

Quando si tenta di caricare in memoria un processo che cresce dinamicamente, ad esempio quando fa uso di variabili dinamiche, si possono verificare alcuni problemi:

1. se il processo si trova in prossimità di un buco questo può crescere facilmente espandendosi occupando il buco.
2. Se il processo è invece adiacente ad un processo, o si sposta il processo in crescita verso un area di memoria che può contenerlo o si spostano i suoi processi adiacenti in modo da creare un buco.
3. Se il processo non può crescere in memoria, o se l'area di swap è piena allora il processo deve attendere la risorsa memoria o sarà killato (se in deadlock).

Se ci si aspetta che il processo cresca è buona norma allocare più memoria di quanto il processo necessita alla sua partenza. Un'unica area per i dati o suddividerla in due aree distinte: area per variabili dinamiche(a), area di crescita condivisa(b).



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## NOTE SULLO SWAPPING

**Efficienza:**

Si predilige il caricamento dei processi dalla memoria centrale piuttosto che quelli presenti sul disco, ciò per evitare l'onerosa operazione preliminare di caricamento dal disco.

**Miglioramento:**

Si suddivide il programma in un certo numero di sezioni di dimensioni fisse (*pagine logiche*) e parallelamente si definiscono delle sezioni di medesima dimensione nella memoria centrale (*pagine fisiche*).

**Vantaggi:**

- Tenere in memoria solo la porzione del codice che serve.
- Il processo può essere allocato in porzioni di memoria non contigue.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Gestione della Memoria con Mappe di Bit

### Mappe di bit Liste concatenate

Le **mappe di bit** forniscono informazioni sull'allocazione della memoria, l'informazione è binaria ed assume valore 0 oppure 1 secondoché la memoria risulta essere libera o occupata. Inoltre la memoria viene ripartita in unità di allocazione la cui lunghezza varia da 32bit in poi.

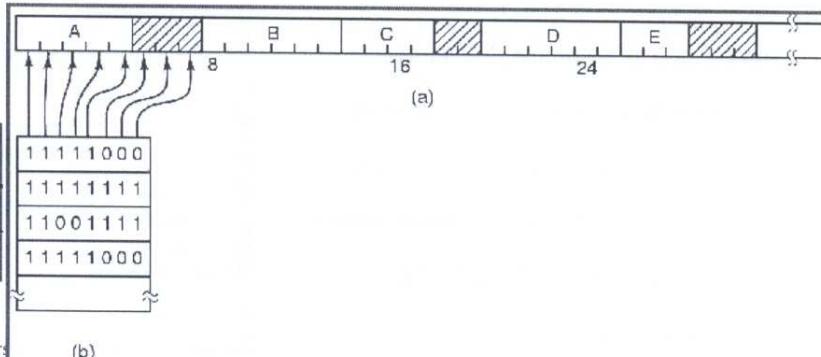
Quindi se la memoria ha  $n$  unità di allocazione (qualsiasi sia la lunghezza di ciascuna allocazione) occorrono  $n$  bit per definire la mappa dei bit. In generale la percentuale di memoria occupata per la mappa dei bit sarà

$$\alpha = \frac{1}{\lambda + 1}$$

dove  $\lambda$  è la cardinalità o dimensione dell'unità di allocazione, (+1) perché serve un bit per ciascuna unità di allocazione.

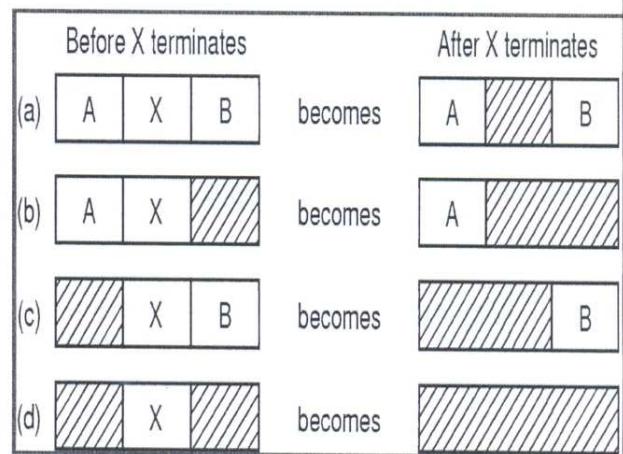
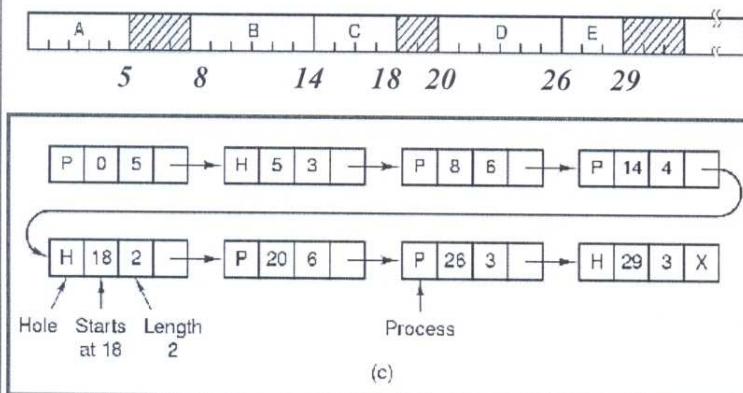
### Esempio

$mem/\lambda + 1$	32bit	64bit	128bit
256k	7,75%	3,93%	1,98%
512k	15,51%	7,87%	3,96%



Questi handout sono da intendersi

## Liste concatenate



Configurazioni possibili quando un processo termina.

Quando i buchi sono inseriti in una lista concatenata, ordinata per indirizzo, si possono usare diversi algoritmi per allocare la memoria ad un processo appena creato o ricaricato dal disco.

1. First Fit

3. Best Fit

5. Quick Fit

2. Next Fit

4. Worst Fit

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Gestione della Memoria con Mappe di Bit

## Liste concatenate

1. First Fit

Il gestore della memoria scorre la lista di segmenti fino a che non trova un buco abbastanza grande; il buco viene quindi diviso in due partizioni, una assegnata al processo, l'altra parte di memoria non utilizzata ritorna a essere buco.

2. Next Fit

Analogia al FIRST FIT con la differenza che si ricorda dove era l'ultimo buco assegnato ed è da lì che comincia a cercare i buchi liberi. N.F. peggiore di F.F.

3. Best Fit

Simile ai precedenti con la particolarità che per ogni processo cerca il buco che si adatta meglio al processo. (il più piccolo buco che può contenere il processo). Più lento del F.F. ed inoltre spreca più memoria in quanto crea molti piccoli buchi.

4. Worst Fit

Per evitare che il gestore generi troppi piccoli buchi, un altro algoritmo fa uso della strategia di assegnare al processo il più grande buco che può contenere il processo. In questo modo i buchi che si formano hanno la dimensione massima che si può ricavare, con conseguente riutilizzo per altri processi.

5. Quick Fit

Mantiene liste separate di buchi. La lista generale punta ad una serie di liste coordinate per dimensione.

Avremo per esempio una lista per i buchi da 4K, una per quelli da 8k , 16K...

**Nota: Ognuna di queste strategie può essere coadiuvata dai seguenti accorgimenti:**

- Separare la lista dei processi dalla lista dei buchi

- Ordinare la lista dei buchi per dimensione

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

L'idea di base che sta dietro alla memoria virtuale è che la dimensione ottenuta dall'unione di programmi, dati e stack può eccedere la dimensione della memoria fisica per essi disponibile.

Il S.O. mantiene in memoria le parti che sono in uso in un certo momento mentre le altre parti sono mantenute sul disco.

Esempio: un applicativo da 16MB può girare in una macchina con 4MB di memoria caricando opportunamente le parti che interessano.

La memoria virtuale può funzionare con sistemi multiprogrammati, caricando porzioni di processi per ciascun processo attivo.

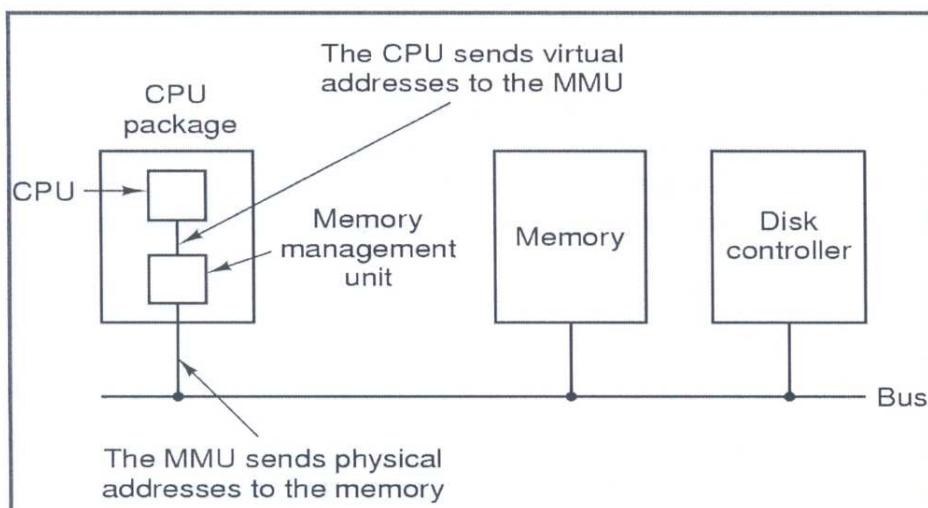
Alla Memoria Virtuale viene associato il concetto di **Pagina**, essa rappresenta lo spazio di indirizzamento virtuale (una pagina generalmente è costituita da 512Byte detta anche blocco); ovviamente alla pagina virtuale viene agganciato il concetto di pagina reale e generalmente la lunghezza delle due pagine è uguale.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Paginazione

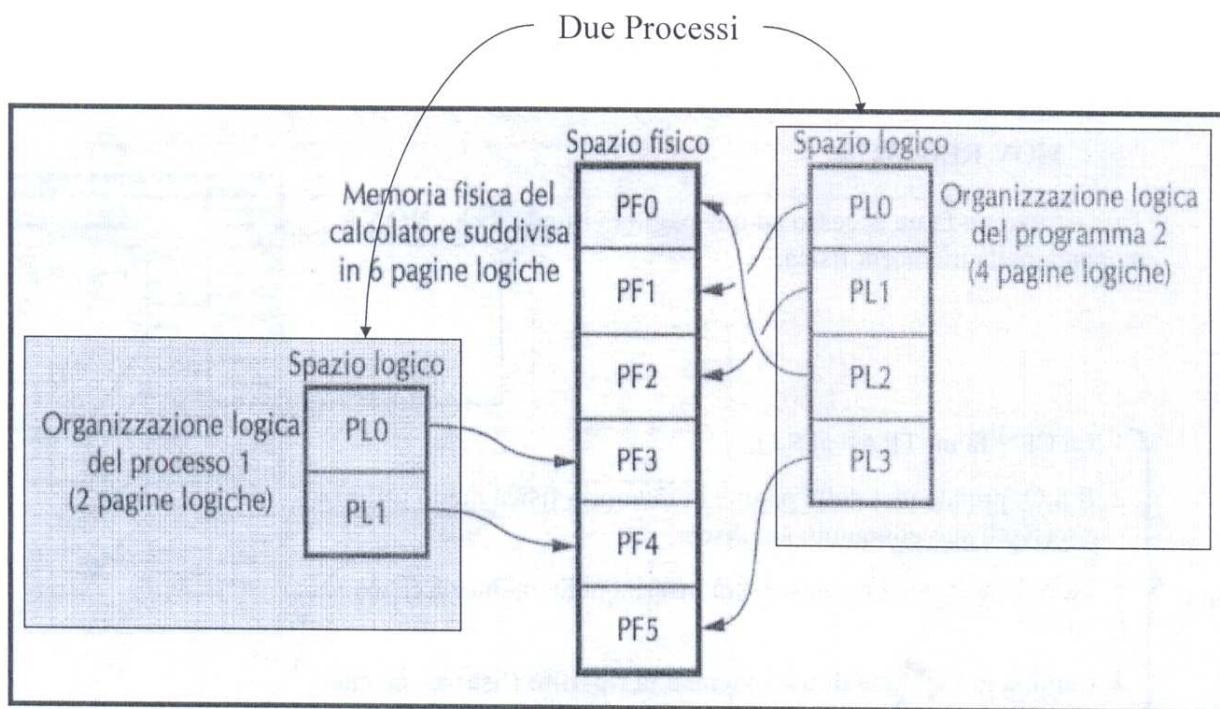
La maggior parte dei sistemi con memoria virtuale usa la tecnica della PAGINAZIONE. La tecnica della paginazione si basa sul **concetto di spazio di indirizzamento virtuale** che contiene gli indirizzi virtuali prodotti dai programmi.

Quando in un sistema è presente la memoria virtuale, la richiesta degli indirizzi di memoria non viene effettuata sul bus di memoria ma transitata attraverso l'MMU (Memory Management Unit), che mappa gli indirizzi virtuali sugli indirizzi della memoria fisica.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## ESEMPIO DI PAGING



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

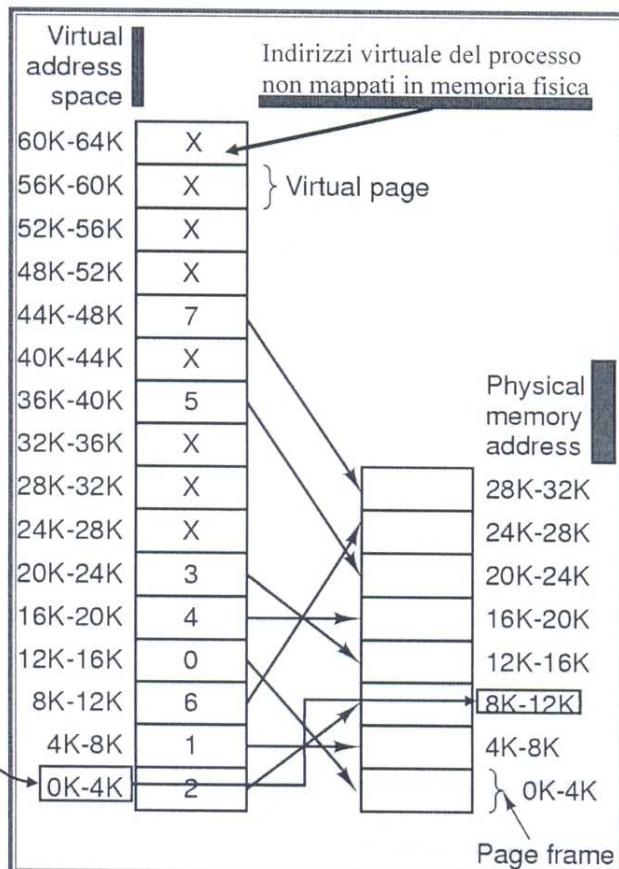
## Paginazione

### Esempio

Vediamo come si comporta un sistema con memoria virtuale, in presenza della seguente istruzione:

**MOV REG, 0**

1. L'indirizzo virtuale viene mandato dalla CPU alla MMU.
2. La MMU si rende conto che l'indirizzo cade nella pagina 0 (da 0 a 4095). Esso corrisponde, secondo la funzione di transizione, alla pagina fisica 2 (8192 12287), quindi invierà sul bus l'indirizzo fisico 8192.
3. La memoria vedrà apparire una richiesta di un indirizzo fisico consentito e lo onorerà.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Fault di Pagina

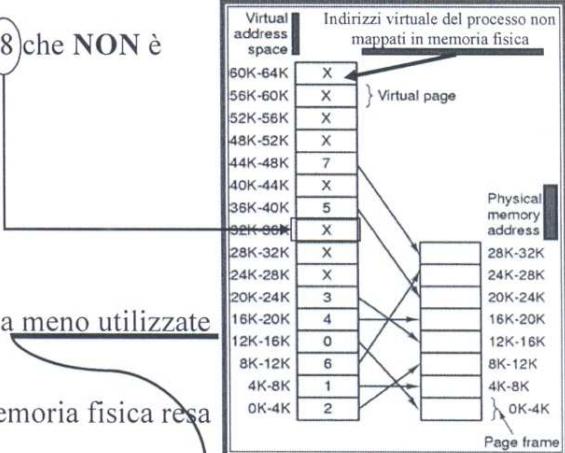
Cosa succede se un programma tenta di usare una pagina virtuale non mappata, per esempio tramite l'istruzione:

**MOV REG, 32780**

Tale istruzione fa un accesso ad una pagina virtuale 8 che **NON** è mappata nella memoria fisica.

Fault di  
Pagina

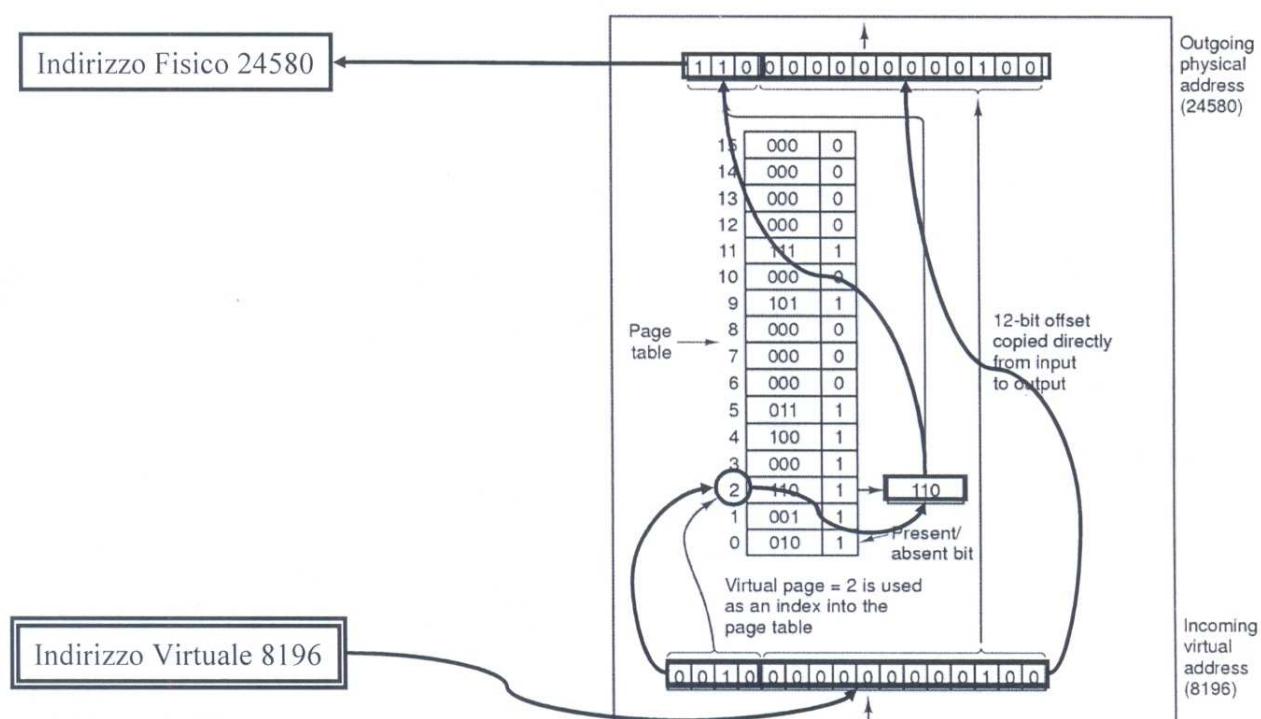
1. La CPU fa un TRAP al S.O.
2. Il S.O. prende una delle pagine in memoria fisica meno utilizzate e scrive il suo contenuto sul disco.
3. La pagina appena richiesta sarà inserita nella memoria fisica resa libera.
4. Cambia la funzione di traduzione e fa ripartire l'istruzione che era stata interrotta.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## M M U

La Memory Management Unit ha un indirizzamento pari ad una potenza di due ed utilizza l'indirizzo virtuale per identificare l'indirizzo fisico.



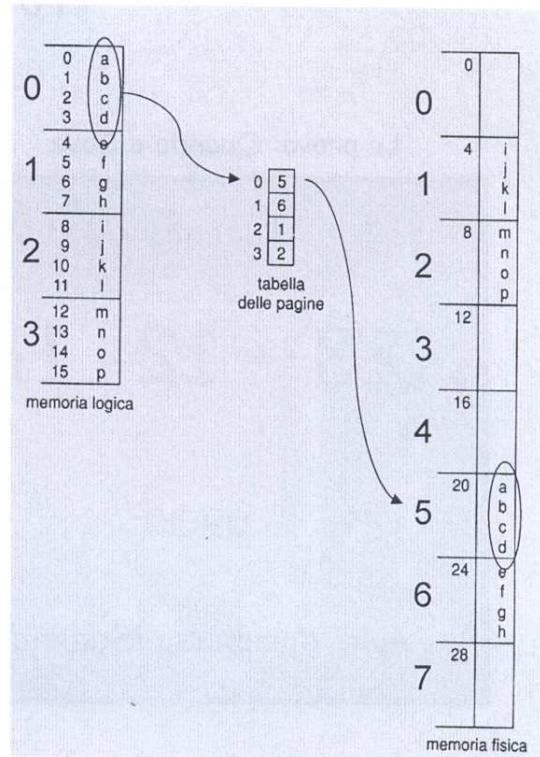
Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Esempio di Paginazione

Paginazione per una memoria a 32 Byte con pagine da 4 Byte

Considerazione:

Il programma utente vede la memoria come un unico spazio contiguo contenente solo il programma, grazie alla netta distinzione tra memoria logica e reale.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

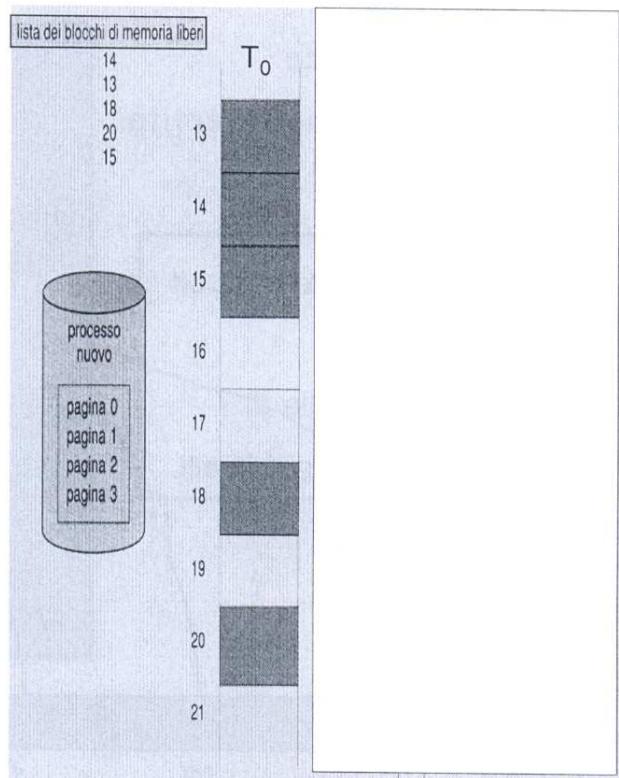
## Esempio dei blocchi liberi nella Paginazione

Esempio di gestione dei blocchi liberi.

Ogni Processo ha una tabella delle pagine.

Il S.O. gestisce l'attribuzione delle pagine libere.

Il programma utente non può invadere zone che non sono di sua pertinenza.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Prova in "Itinere"

S.O. 2011/12

La prova: Quando e Dove



Il menù della prova



Refresh Argomenti 06 Dicembre 2011

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Esempi di Strutturazione delle Tabelle delle Pagine

- Tabella Gerarchica
- Tabella della pagina Invertita.

Alcune architetture a 32 bit e altre a 64 bit, considerano tabelle di paginazione sino a 4 livelli.

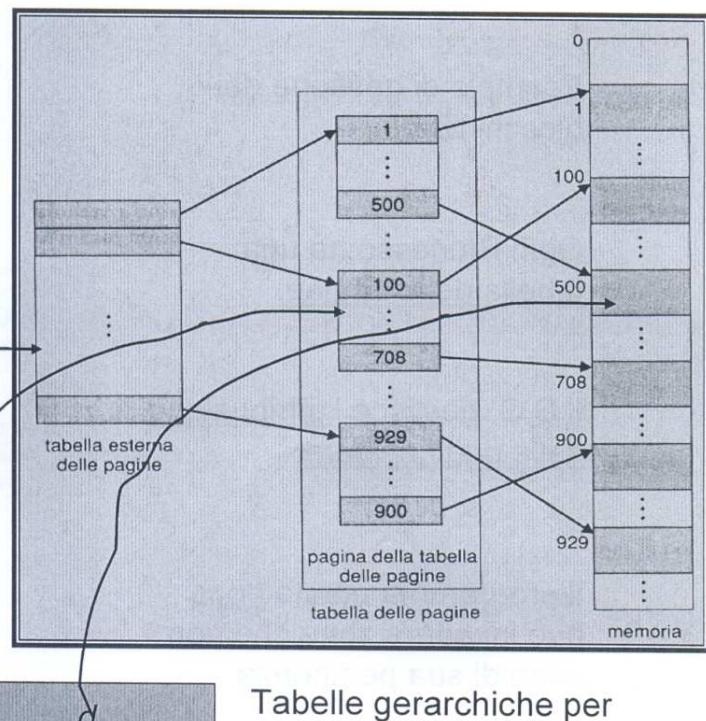


Tabelle gerarchiche per un'architettura a 32 bit

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Tabella della pagina Invertita<sub>1</sub>

225

La gestione gerarchica delle tabelle associa una tabella delle pagine ad ogni processo, tale tabella contiene un elemento per ogni pagina logica che il processo sta utilizzando.

**Pregio:** Facile ritrovamento dell'indirizzo fisico poiché la tabella è ordinata per indirizzi logici.

**Difetto:** la dimensione di ciascuna tabella può occupare grandi quantità di memoria fisica.

Esempio in un'architettura a 64 bit.

Spazio di indirizzamento è  $2^{64}$ ,

Dimensione pagine  $4KB = 2^2 \cdot 2^{10} = 2^{12}$

Dimensione della tabella  $2^{52} = (2^{64}/2^{12})$   
con elementi a 8 byte (64bit)



Dimensione tabella per ogni processo è di circa 30 milioni di GigaByte

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

226

## Tabella della pagina Invertita<sub>2</sub>

### Differenza:

La tabella ha un elemento per ogni pagina reale.

Ciascun elemento è costituito dall'indirizzo logico della pagina memorizzata in una specifica locazione reale e dall'identificatore del processo.

<PID, #Pg, Scostamento>

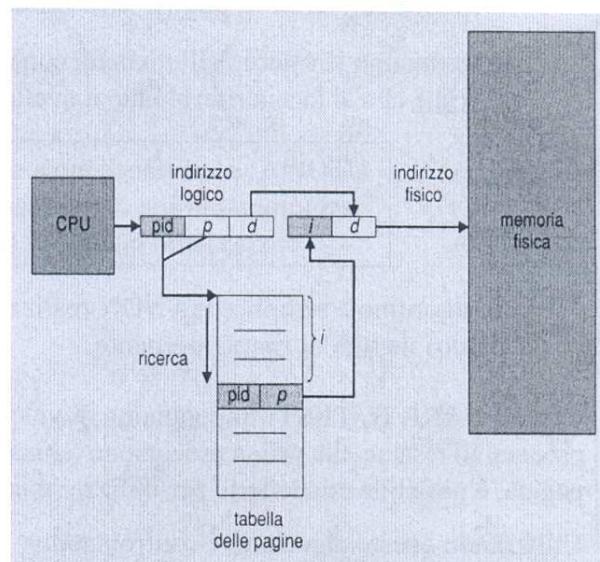
Esempio in un'architettura a 64 bit.

Spazio di indirizzamento è  $2^{64}$ ,

Dimensione pagine  $4KB = 2^2KB$

Dimensione della RAM  $256MB = 2^{18}KB$

Dimensione della tabella  $2^{18}/2^2 = 2^{16} = 65536$  elementi a 8 byte (64bit)



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

Come si osserva nella tabella delle pagine, lo spazio di indirizzamento virtuale è sempre più grande dello spazio fisico (altrimenti non avrebbe motivo di esistere).

Quando non vi è più memoria fisica da allocare per una porzione di memoria virtuale, il S.O. effettua un Fault di pagina e sono proposti i passi che il S.O. compie per rimpiazzare la pagina di memoria.

**Quale pagina scaricare?**

**La pagina da eliminare dalla memoria va conservata?**

**se sì, sotto quali condizioni?....**

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Algoritmi di rimpizzamento delle Pagine

**Descrizione: I° soluzione: Facile a Descrivere ma IMPOSSIBILE a realizzare!!!**

l'algoritmo si basa su concetto di ETICHETTA (valore numerico) da assegnare alla pagina: ogni volta che avviene un FAULT di pagina il sistema SCARICA dalla memoria fisica la pagina che ha un etichetta con valore più ALTO. ?

Definiamo il valore dell'etichetta come il numero delle istruzioni che saranno eseguite prima che si faccia riferimento a quella pagina per la prima volta.

UTOPIA: come facciamo a sapere il numero delle istruzioni che si eseguiranno prima che si faccia riferimento a quella pagina se non conosciamo quando si farà riferimento a quella pagina?



L'algoritmo è semplice ma **NON** realizzabile, è possibile utilizzarlo per fare dei test sulla bontà di altri metodi di rimpiazzamento.

**COME FARE IL TEST:** Supponiamo di voler fare eseguire su di un elaboratore un certo numero di processi di test, se alla prima esecuzione teniamo traccia di come vengono eseguite le istruzioni pagina per pagina, è possibile etichettarle per utilizzarle in una seconda esecuzione.

Utilizzando questo algoritmo e confrontando i risultati con un nuovo algoritmo di rimpiazzamento è possibile capire la sua bontà di esecuzione in termini di percentuale rispetto all'ottimale.

Quindi se il nuovo algoritmo di rimpiazzamento è peggiore dell'1% esso sarà sicuramente accettabile visto che quello ottimale è solo una teorizzazione non realizzabile.

#### Algoritmi di rimpiazzamento delle pagine

1. Non usate di recente.
2. FIFO
3. Della seconda opportunità
4. Dell'orologio
5. Least Recently used
6. Working-set

Definiamo due bit assegnati a ciascuna Pagina Fisica:

R = bit che rappresenta l'informazione di pagina Referenziata (1 = Letta o Scritta)

M = bit che rappresenta l'informazione di pagina Modificata (1 = Scritta)

NOTA: È importante comprendere che i bit R ed M devono essere aggiornati ad ogni riferimento in memoria.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

I bit R ed M formano 4 possibili classi e possono essere usati come segue:

il bit R viene azzerato periodicamente ad ogni interruzione dal clock, l'azzeramento di R stabilisce le pagine che non sono state referenziate di recente.

Le classi formate dalla combinazione di R ed M:

0. **R=0, M=0** implica pagina non usata e non modificata.
1. **R=0, M=1** implica pagina non usata e modificata.
2. **R=1, M=0** implica pagina usata e non modificata.
3. **R=1, M=1** implica pagina usata e modificata.

**Nota1:** la classe 1 sembrerebbe che non possa mai verificarsi, essa si manifesta quando una pagina si trova nella classe 3 e un interrupt del clock riporta tutti i bit **R** delle pagine a 0.

**Nota2:** il bit **M NON** sarà mai azzerato dal clock in quanto è necessario conoscere se la pagina è stata modificata per permettere la sua conservazione sul disco.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

### NON USATE DI RECENTE

Definite le classi, l'algoritmo di rimpiazzamento delle pagine “Non Usate di Recent” rimuove una pagina dalla classe di valore più basso (generalmente dalla classe 0,1,2,3).

**L’Idea di fondo:** è meglio rimuovere una pagina modificata a cui non si è fatto riferimento per un periodo di clock che una pagina usata frequentemente.

- Nota:**
- L’algoritmo è facile da comprendere,
  - Efficiente nella sua implementazione,
  - Non ottimale ma spesso soddisfacente.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Algoritmi di rimpizzamento delle Pagine

### FIFO: First-In, First-Out

Tale algoritmo si basa sulla scelta delle pagine in una lista concatenata contenente i riferimenti alle pagine fisiche, in essa la testa della lista conterrà la pagina più vecchia (in termini temporali), mentre in coda sarà presente la più recentemente utilizzata.

**Tecnicamente:** Quando si attiva un fault di pagina verrà scaricata dalla memoria la pagina che si trova in testa alla lista e la nuova pagina sarà inserita in coda.

**Idea di Fondo:** togliere la pagina più vecchia presente in memoria.

**Nota:** l’algoritmo rischia di togliere pagine molto usate ma da molto tempo in memoria.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

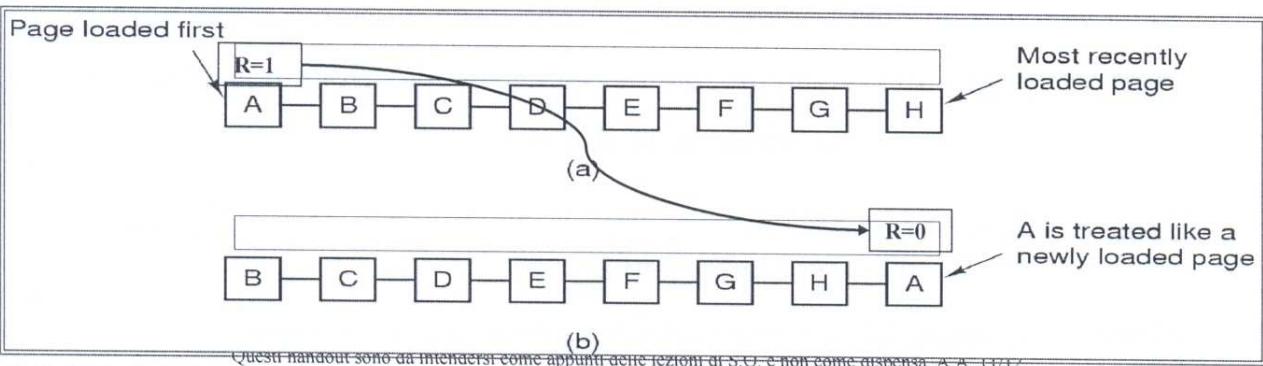
## Della Seconda Opportunità

L'algoritmo è una variante dell'algoritmo FIFO ha la stessa logica di esecuzione ma tiene conto del bit R ed M definiti prima.

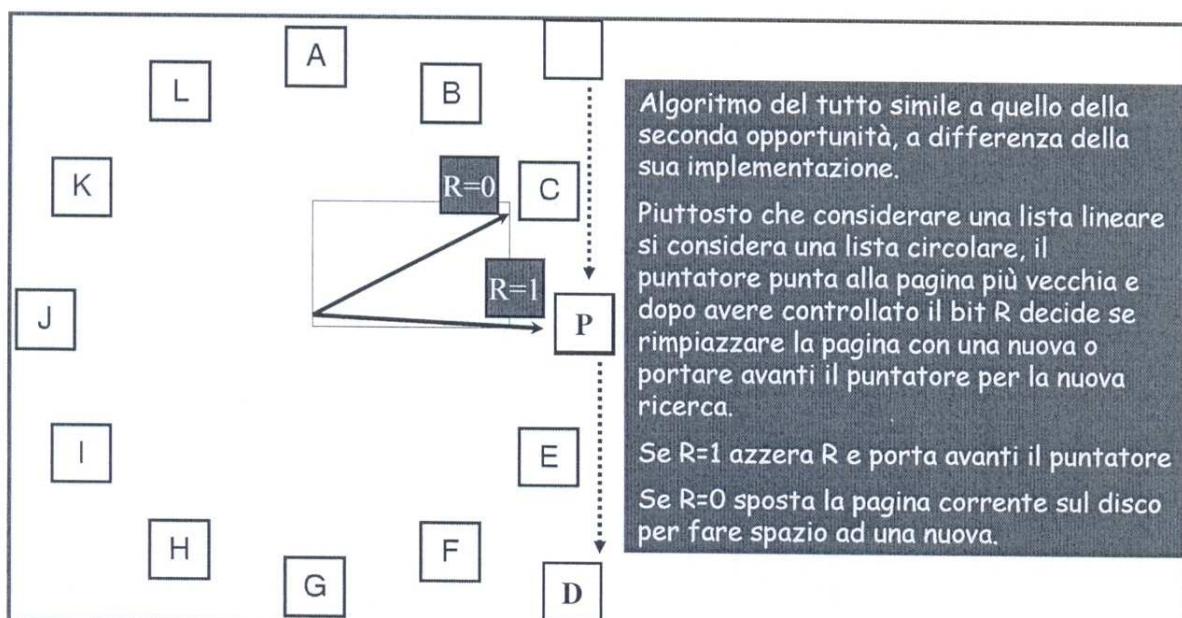
**Tecnicamente:** Quindi l'algoritmo fa riferimento alla pagina in testa alla coda (la pagina più vecchia), inoltre controlla il bit R che riporta se essa è stata usata recentemente, se così è ( $R=1$ ) la pagina viene spostata alla fine della coda,  $R$  posto uguale a 0 in modo che la pagina viene considerata come inserita ex-novo

**Idea di fondo:** Cercare una pagina vecchia che non sia stata usata nel precedente periodo di clock.

**Nota:** se tutte le pagine risultano usate, l'algoritmo li passerà tutte in rassegna, mettendo per ciascuna il bit  $R=0$ , alla fine del ciclo tutte le pagine avranno  $R=0$  e l'algoritmo sarà uguale a quello della FIFO.



## Dell'orologio



## LRU - Least Recently Used

L'algoritmo si basa sulla strategia di eliminare la pagina che non è stata usata da più lungo tempo.

**Tecnicamente:** Implementazione di tale algoritmo richiede che ogni pagina abbia un registro contenente un valore numerico (contatore, tempo trascorso,...) che rappresenta la persistenza della pagina in memoria.

Quando un'istruzione viene referenziata, un contatore viene incrementato e il suo valore viene assegnato alla pagina contenente l'istruzione referenziata (il contatore può essere il *time* della macchina).

Se si verifica un Fault di Pagina il gestore della memoria scaricherà la pagina che ha il valore più piccolo nel caso del contatore numerico, l'orario più vecchio nel caso sia stato usato il timer.

**Idea di Fondo:** Togliere dalla memoria le pagine che sono state usate meno recentemente.

**Nota:** Complesso ma non impossibile da realizzare.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Algoritmi di rimpizzamento delle Pagine

## LRU - Least Recently Used

## Una Implementazione:

L'implementazione dell'algoritmo LRU può essere effettuato con una tabella  $n \times n$  dove  $n$  è il numero delle *pagine della memoria fisica*.

Quando viene referenziata la pagina  $i$  la tabella viene aggiornata nel modo seguente: tutti gli elementi della riga  $i$  sono posti a 1 mentre tutti gli elementi della colonna  $i$  sono posti a 0.

Esempio di pagine referenziate: 0 1 2 3 2 1 0 3 2 3

	Page 0	Page 1	Page 2	Page 3
0	0 1 1 1	0 0 1 1	0 0 0 1	0 0 0 0
1	0 0 0 0	1 0 1 1	1 0 0 1	1 0 0 0
2	0 0 0 0	0 0 0 0	1 1 0 1	1 1 0 0
3	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 0

	Page 0	Page 1	Page 2	Page 3
0	0 0 0 0	0 0 1 1	0 1 1 0	0 1 0 0
1	1 0 1 1	0 0 1 1	0 0 1 0	0 0 0 0
2	1 0 0 1	0 0 0 1	0 0 0 0	1 1 0 1
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0

## Algoritmi di rimpizzamento delle Pagine

### Working Set

Definiamo **Working Set** o **Insieme di Lavoro**, l'insieme delle pagine utilizzate correntemente da un processo.

**NOTA:** Se tutto l'insieme di lavoro fosse caricato in memoria il processo girerebbe senza causare troppi fault di pagina.

Quindi, molti S.O. tendono a caricare in memoria tutto l'insieme di lavoro, **Working-set**, del processo prima di concedergli l'uso della CPU. Tale tecnica è basata sulla **prepaganazione** (il S.O. tende a predire quali pagine saranno utilizzate dal processo), inoltre per far ciò il sistema, per ogni processo, deve conoscere all'istante di tempo  $t$  le pagine di tutti i riferimenti presenti in memoria.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Algoritmi di rimpizzamento delle Pagine

### Working Set

In generale gli applicativi tendono a NON riferire il loro spazio di indirizzamento in maniera uniforme e globale, ma concentrano la loro attenzione in uno spazio di indirizzamento ristretto.

Per ogni istante  $t$  esiste un insieme costituito da tutte le pagine utilizzate dai  $K$  riferimenti più recenti in memoria.

Tale insieme è il Working Set

### Working Set

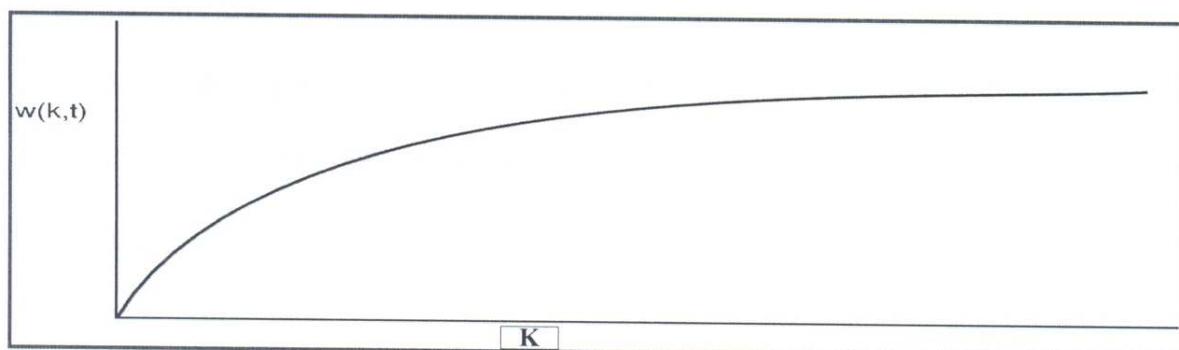
Il Working-set può variare nel tempo sia in termini di dimensione che di contenuto delle pagine.

Quindi il WS può essere definito come una funzione a due variabili:

$$w(k, t)$$

dove  $t$  è il tempo in esame e  $k$  sono i riferimenti più recenti in memoria.

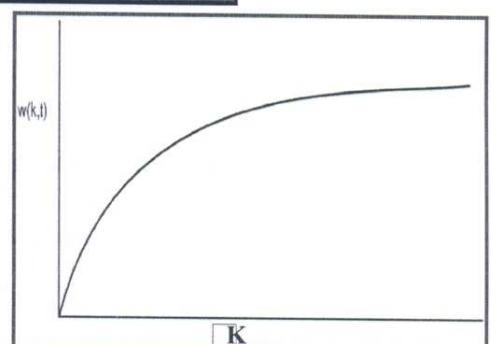
Tale funzione è una funzione monotona non decrescente il cui limite per  $K$  che tende a INFINITO è FINITO.



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

### Working Set

**Giustificazione:** la rapida crescita della curva dipende dal fatto che i programmi in generale accedono in maniera casuale ad un numero piccolo di pagine, e che questo insieme cambia lentamente nel tempo.



Dal grafico ci si rende conto che, finito il periodo di transizione, **esiste un grande intervallo di valori  $K$  per cui l'insieme di lavoro non è cambiato**.

Considerato che l'insieme di lavoro cambia lentamente nel tempo, si può supporre di **conoscere quali pagine saranno necessarie** quando il processo verrà fatto ripartire, sulla base dell'insieme di lavoro che è presente al momento dell'interruzione.

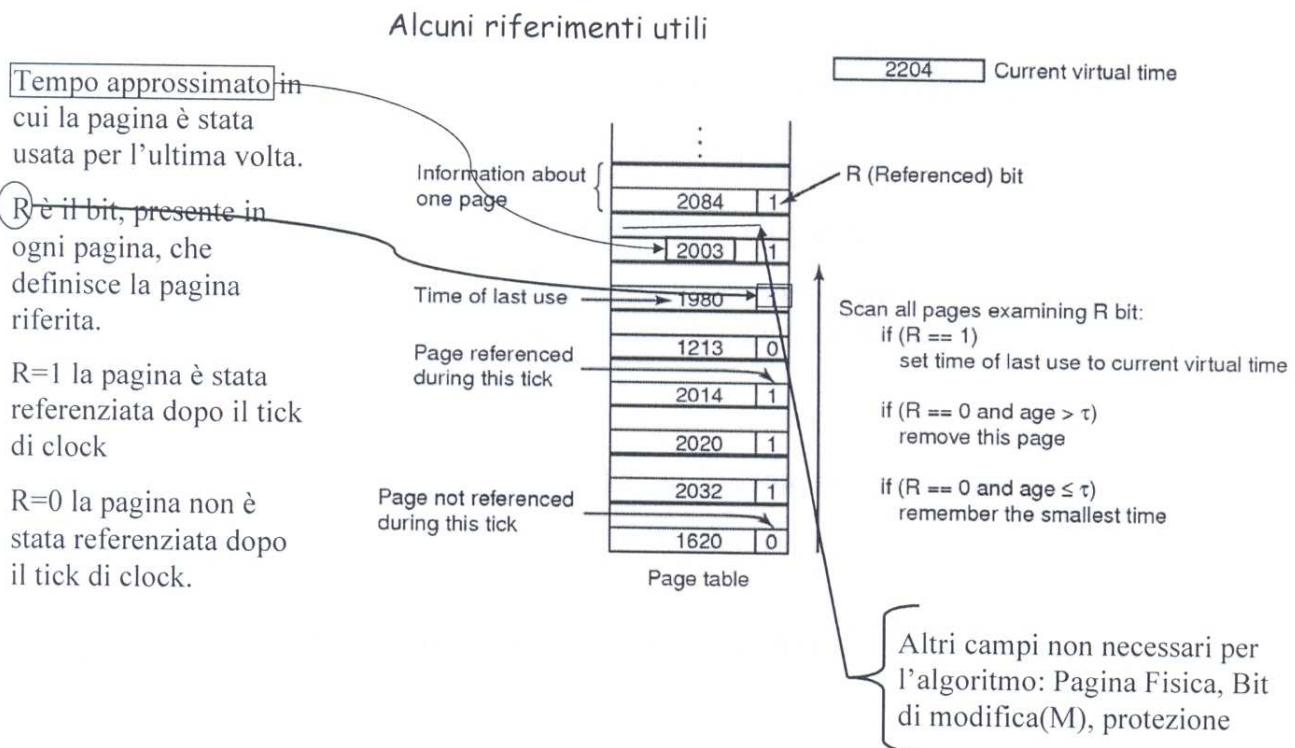
La **Prepaginazione** consente di **caricare tali pagine prima** che il processo venga mandato in esecuzione.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Algoritmo di rimpinzamento delle Pagine basato sul Working-set

241

L'Algoritmo si basa sulla ricerca di una pagina che non è sul working-set e di scaricarla.



## Algoritmo di rimpinzamento delle Pagine basato sul Working-set

242

Se  $R=1$  la pagina non è candidata ad essere rimossa, viene aggiornato il valore del tempo virtuale della pagina, poiché esso indica che in quell'istante la pagina fu usata.

Se  $R=0$  si controlla il tempo della pagina, sottraendo al tempo corrente il tempo trovato nella pagina se questo valore è superiore a  $T$  allora la pagina viene eliminata e sostituita dalla nuova.

Se  $R=0$  e il tempo è inferiore a  $T$  allora la pagina è ancora presente nell'insieme di lavoro.

**Nota:** Se tutta la tabella è stata esaminata e nessun valore è stato trovato a 0, allora tutte le pagine fanno parte del Working-set e si sceglierà una pagina, in modo casuale, per fare spazio alla nuova pagina richiesta.

## SEGMENTAZIONE DELLA MEMORIA

Per ridurre gli sprechi di memoria o la ridondanza nell'uso della memoria, si possono organizzare i programmi in modo da raccogliere a fattor comune alcune sottoparti da destinare a processi distinti.

La struttura di un applicativo può essere suddivisa in tre famiglie:

Codice: contenente le istruzioni del programma.

Dati: contiene sia i dati statici sia quelli dinamici.

Stack: contiene i dati che il processo deve salvare durante la sua esecuzione(chiamata a procedura, cambio di contesto, ...)

Vantaggi:

- Il S.O. può consentire a più processi di condividere lo stesso codice e nel contempo di operare su dati diversi.
- Lo swap agisce sui segmenti di memoria per spostare sul disco le parti dei processi non in uso.

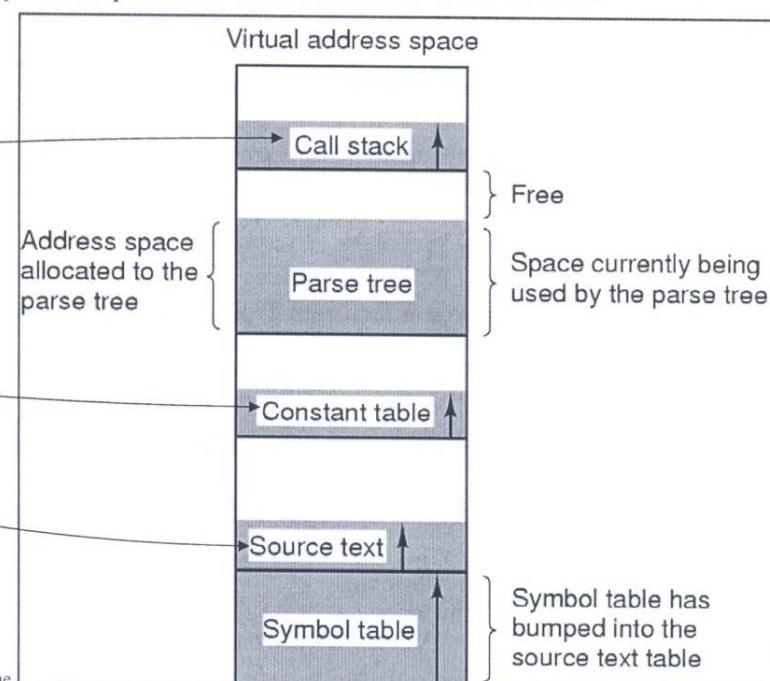
Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Segmentazione

- La caratteristica principale di una memoria Virtuale è di essere unidimensionale, ossia esiste un indirizzo di partenza e uno finale.
- L'idea di avere più aree di memoria magari con dimensioni diverse può agevolare alcuni applicativi che scindono il proprio compito in più compiti.

ESEMPIO: Il Compilatore scinde il proprio compito in almeno 5 Processi con le relative tabelle:

1. Testo sorgente,
2. Tabella dei simboli
3. Tabella delle costanti
4. Albero dell'analisi sintattica
5. Lo Stack chiamate a procedure del compilatore.



Questi handout sono da intendersi come

**Evento Indesiderato:** in compilazione si può avere un numero estremamente elevato di variabili, normalità per il resto.

Il compilatore riempie la tabella dei simboli e non riesce ad andare avanti perché lo spazio è insufficiente. Ne segue che viene mandato un messaggio “memoria insufficiente” ma ciò non è vero perché vi è dell’altra memoria da un’altra parte che può essere recuperata.

Soluzione banale:compatto la memoria prima di mandare il messaggio di errore.

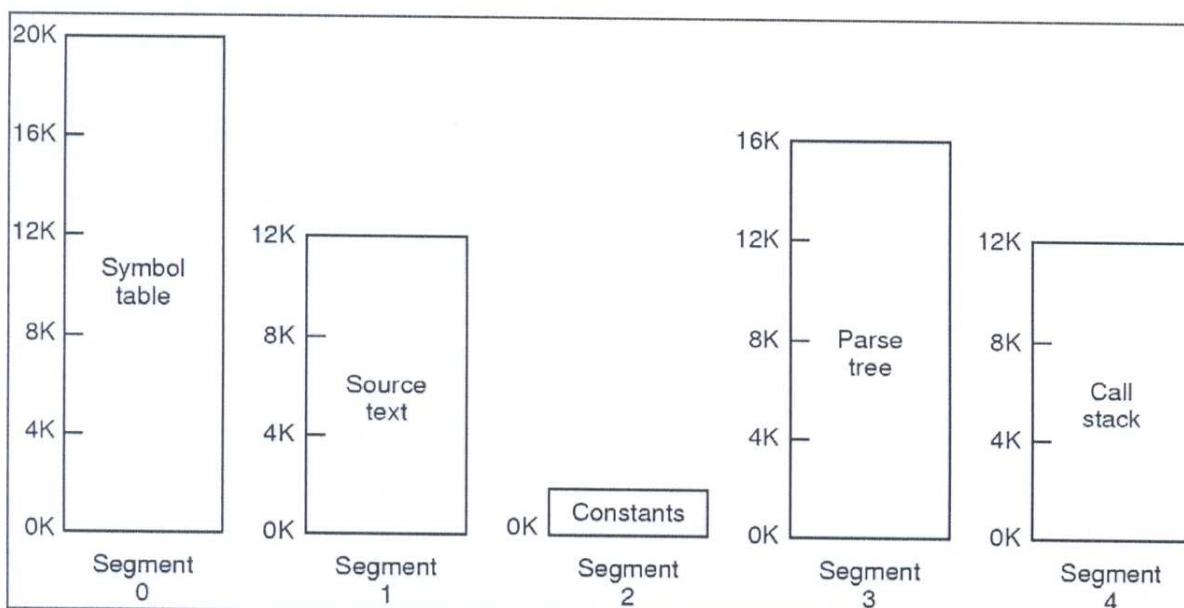
**Soluzione ottimale:**dividere la memoria in molti spazi di indirizzamento con diverse dimensioni, chiamati segmenti.

Ciascun segmento ha una sequenza lineare di indirizzi che generalmente è diversa dagli altri. In essi la lunghezza degli stack cambia dinamicamente liberando e concedendo indirizzi di memoria.

**NOTA:** poiché ciascun segmento costituisce uno spazio di indirizzamento separato, esso può crescere o contrarsi in maniera indipendente dagli altri.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

Per specificare un indirizzo nella memoria segmentata il programma deve dare un indirizzo costituito da due parti: **un numero rappresentante lo specifico segmento e un indirizzo all'interno del segmento.**



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Segmentazione

La Memoria Segmentata oltre ad avere come vantaggio quello di semplificare la gestione dei dati che crescono e si contraggono, semplifica inoltre le chiamate delle procedure:

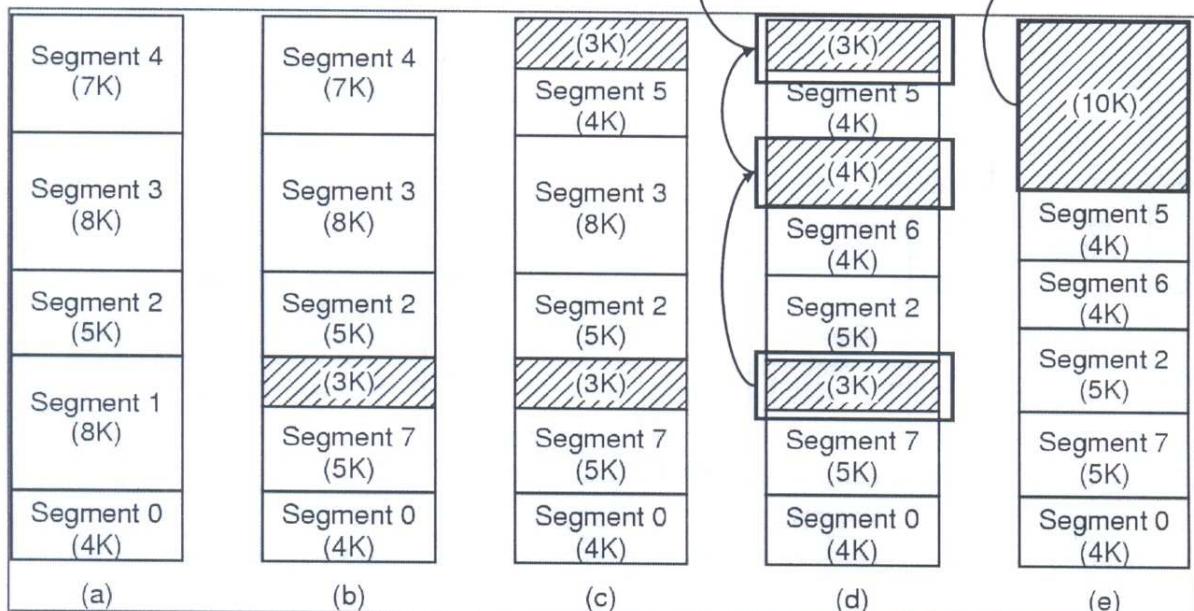
- + Le **n** procedure distinte posso risiedere sugli n segmenti distinti e se **0** è l'indirizzo di partenza della procedura all'interno del segmento allora la procedura sarà richiamata dalla coppia (n,0);
- + Poiché ogni procedura è inserita su di un segmento diverso, una sua modifica implica una sua compilazione ma non la compilazione delle altre e ne tanto meno un loro riposizionamento.
- + La segmentazione facilita la condivisione delle risorse e/o dei dati fra i vari processi.
- + Facilita la protezione e la sicurezza dei dati.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Implementazione della Segmentazione

L'implementazione della segmentazione differisce da quella della paginazione per un punto fondamentale:

- Le **pagine** hanno dimensione **fissa**
- I **segmenti** hanno dimensione **variabile**.

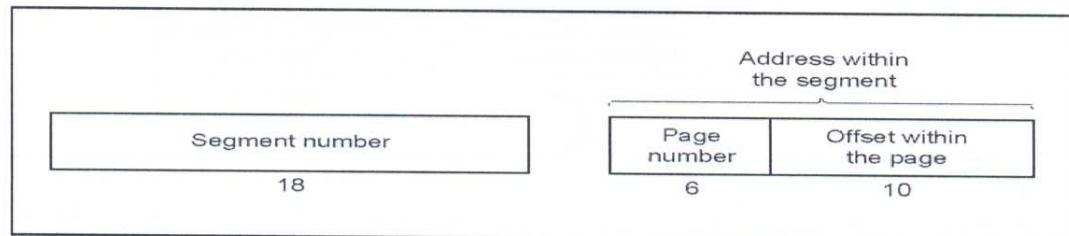


Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Segmentazione con Paginazione

Se i segmenti sono grandi può risultare poco conveniente, se non impossibile, caricarli in memoria nella loro interezza.

La paginazione è una tecnica che consente di risolvere questo problema, in tale senso, secondo i canoni della paginazione, in memoria sono residenti quelle porzioni di segmento che sono necessarie.

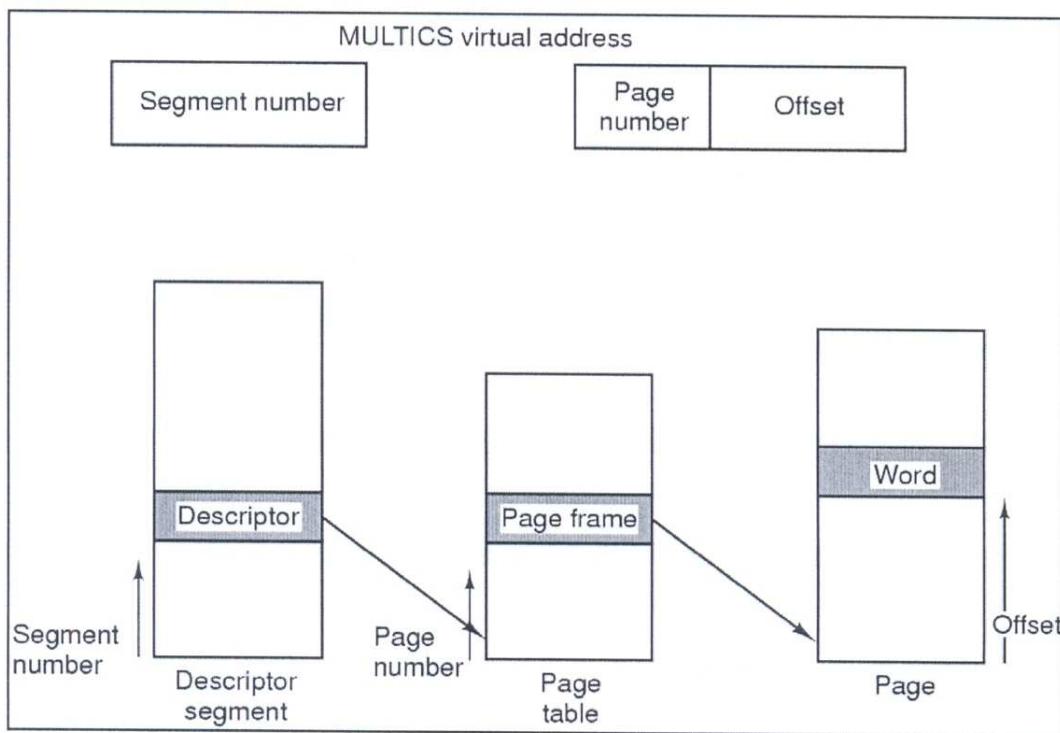


Ciascun segmento rappresenta un normale spazio di indirizzamento virtuale e viene gestito tramite paginazione allo stesso modo in cui era gestita la memoria virtuale.

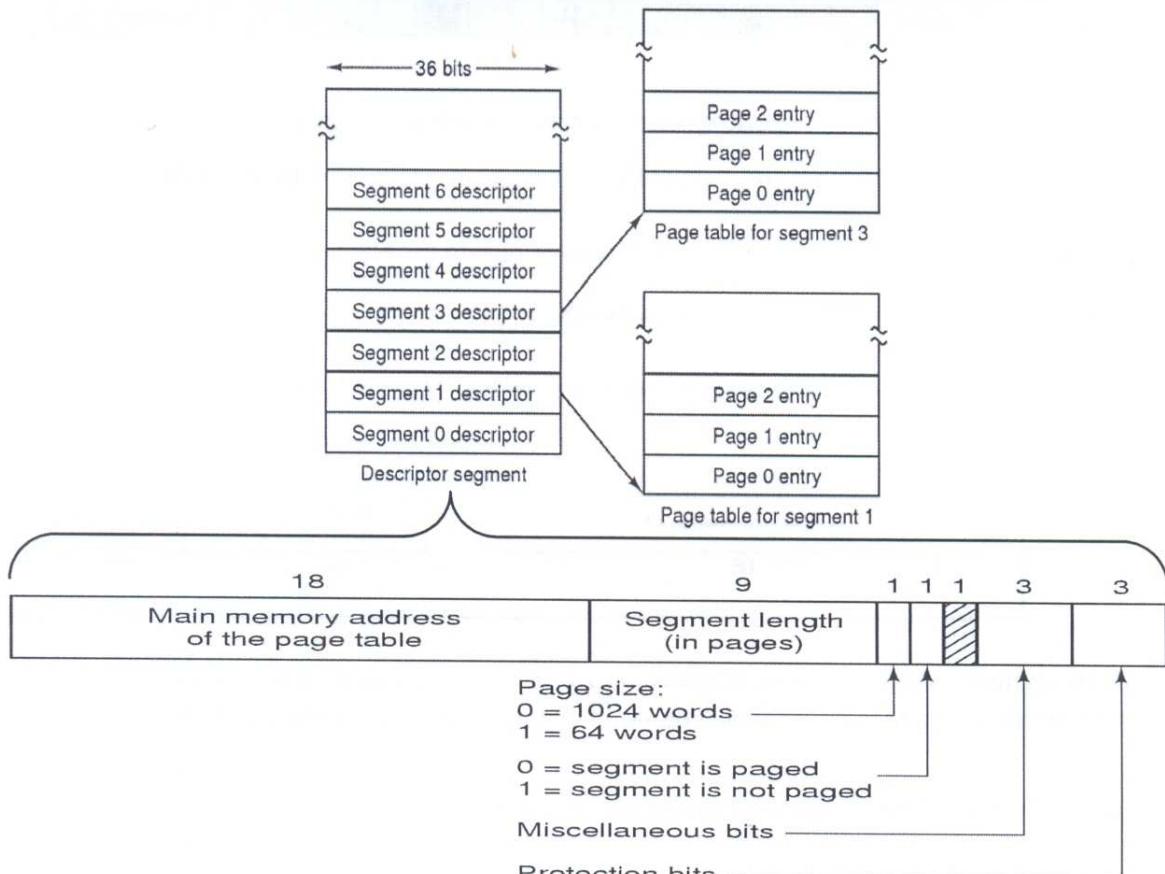
Nota: Il Pentium è un'architettura molto vicina a questo tipo di approccio.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Conversione di indirizzo (multics)



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Segmentazione con Paginazione

Algoritmo è attivato quando si verifica un riferimento alla memoria:

1. Il numero del segmento viene usato per recuperare il descrittore del segmento all'interno del segmento dei descrittori;
2. Si esegue un controllo per verificare se la tabella delle pagine relativa al segmento è presente in memoria.
3. Si esamina l'elemento della tabella delle pagine relativo alla pagina virtuale richiesta.
4. Si somma l'offset all'indirizzo di partenza della pagina per individuare l'indirizzo fisico.
5. Si effettua la lettura o scrittura.

## Segmentazione con Paginazione: Un Esempio

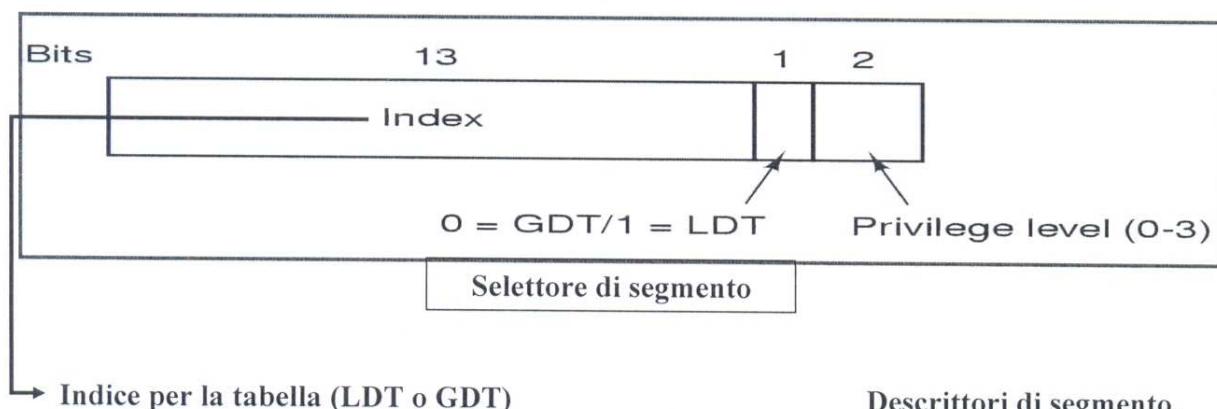
Il cuore della memoria virtuale del Pentium è dotato di due tabelle: la LDT (local Descriptor Table) e la GDT (Global Descriptor Table).

Tutti i programmi hanno una loro LDT ma esiste una sola GDT.

Per accedere ad un segmento, il Pentium carica dapprima un selettore di quel segmento in uno dei suoi registri di segmento della macchina.

Il Registro CS (Code Segment) contiene il selettore per il segmento del codice programma

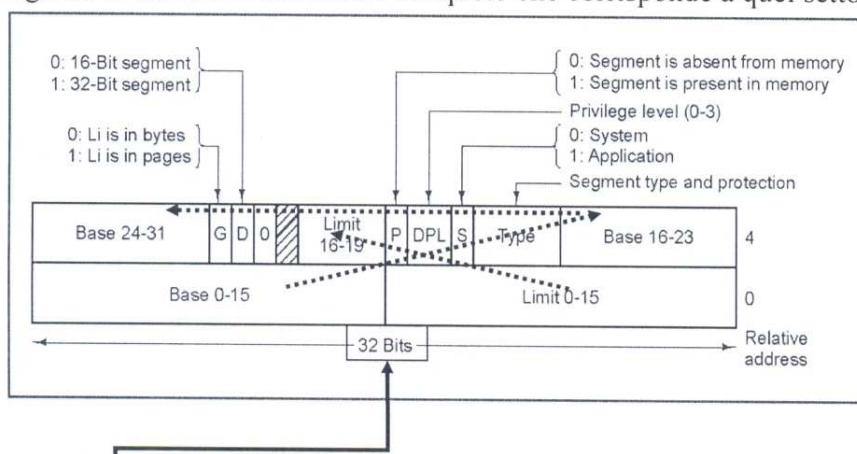
Il Registro DS (Data Segment) quello per il segmento dei dati del programma



Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Descrittore del segmento nel esempio Pentium

Non appena il microprogramma ha ricevuto la richiesta (conosce quale registro del segmento verrà usato) è in grado di trovare il descrittore completo che corrisponde a quel settore.



In seguito controlla se l'offset va a cadere oltre i limiti del segmento

$2^{13}$  (8192=8K) x 32 dimensione della tabella dei descrittori del segmento

→ Spazio di indirizzamento del selettore.

Questi handout sono da intendersi come appunti delle lezioni di S.O. e non come dispensa. A.A. 11/12

## Dimensione della Pagina

$S$  = Dimensione media di un processo espressa in Byte.

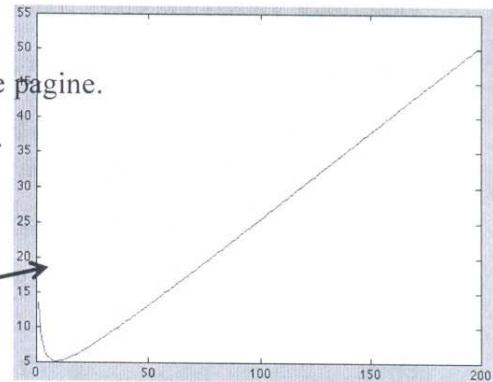
$P$  = Dimensione delle pagine espressa in Byte.

$E$  = Richiesta delle pagine da ciascun elemento della tabella delle pagine.

$S/P$  = Numero approssimato di pagine richieste da ogni processo.

$S \cdot E/P$  = Numero di byte occupati nella tabella delle pagine.

$P/2$  = Memoria sprecata nell'ultima pagina.



$$\text{Overhead}(P) = S \cdot E/P + P/2$$

Esempio:

$$S=1\text{MB}, E=8\text{Byte} \implies P=4\text{KB}$$

Deriviamo rispetto a  $P$  per il calcolo del minimo:

$$\text{Overhead}(P)' = -S \cdot E/P^2 + 1/2$$

$$-S \cdot E/P^2 + 1/2 = 0 \implies P = \sqrt{2 \cdot S \cdot E}$$

Valore utilizzato per una pagina comune è compreso tra 512 Byte e 64Kbyte.

In questo momento le pagine assumono dimensioni pari a 4K o 8K