Fondamenti dei Sistemi Operativi

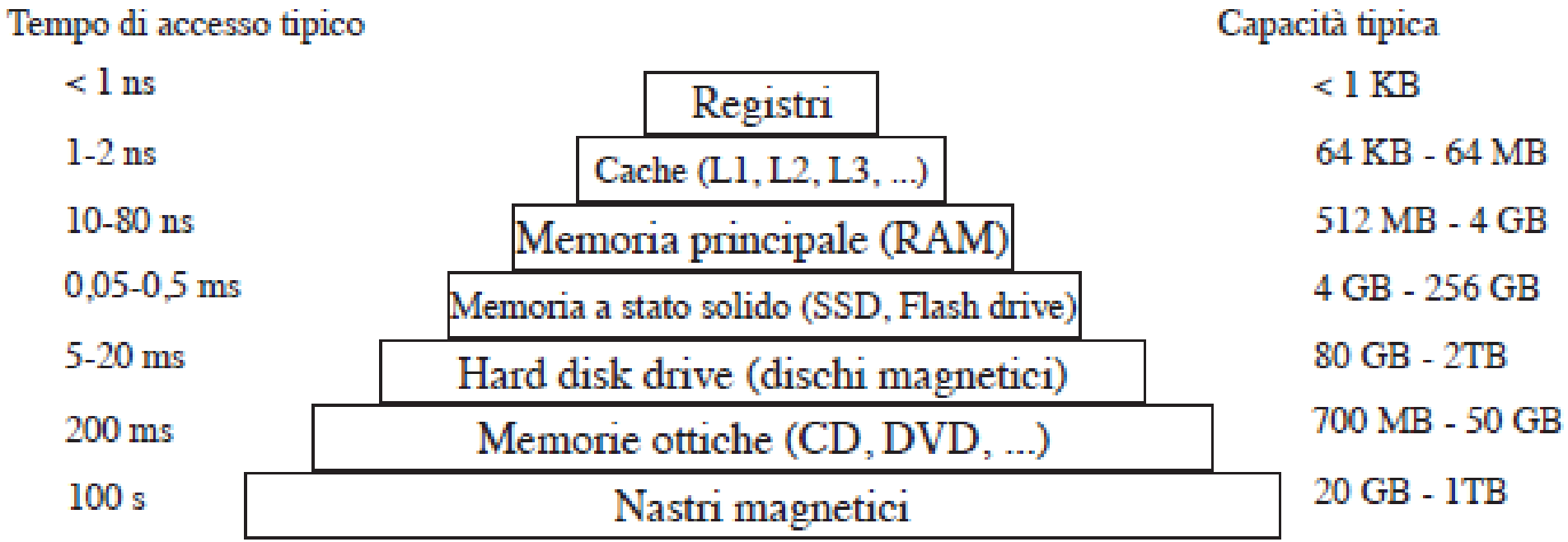
Memory manager

Sommario

La memoria centrale: requisiti della gestione
Dal programma sorgente all'esecuzione: spazio degli indirizzi e caricamento
Indirizzamento e traduzione degli indirizzi: Memory Management Unit (MMU)
Dynamic loading and linking; DLL
Algoritmi di gestione della memoria: mono e multi programmazione
Allocazione contigua di memoria
Partizionamento statico e partizionamento dinamico di memoria
Partizionamento rilocabile e multiplo
Swapping e Rolling. Overlay
Paginazione reale: tabella delle pagine (PMT) e tabella della memoria (MBT)
Traduzione degli indirizzi, Translation Look-aside Buffer (TLB)
Segmentazione: indirizzamento e condivisione dei segmenti
Segmentazione con paginazione
Memoria virtuale: principi di località e relizzazione
Demand paging: page fault, struttura della MMU, Page replacement e algoritmi
Demand segmentation e Demand paged-segmentation
Memoria virtuale multipla

La memoria centrale

La memoria è una **risorsa importante, e limitata**.
"I programmi sono come i gas perfetti: si espandono fino a riempire tutta la memoria disponibile".
Memoria illimitata, infinitamente veloce, economica: non esiste.
Esiste la gerarchia delle memorie, utilizzata dal gestore della memoria (***memory manager***).



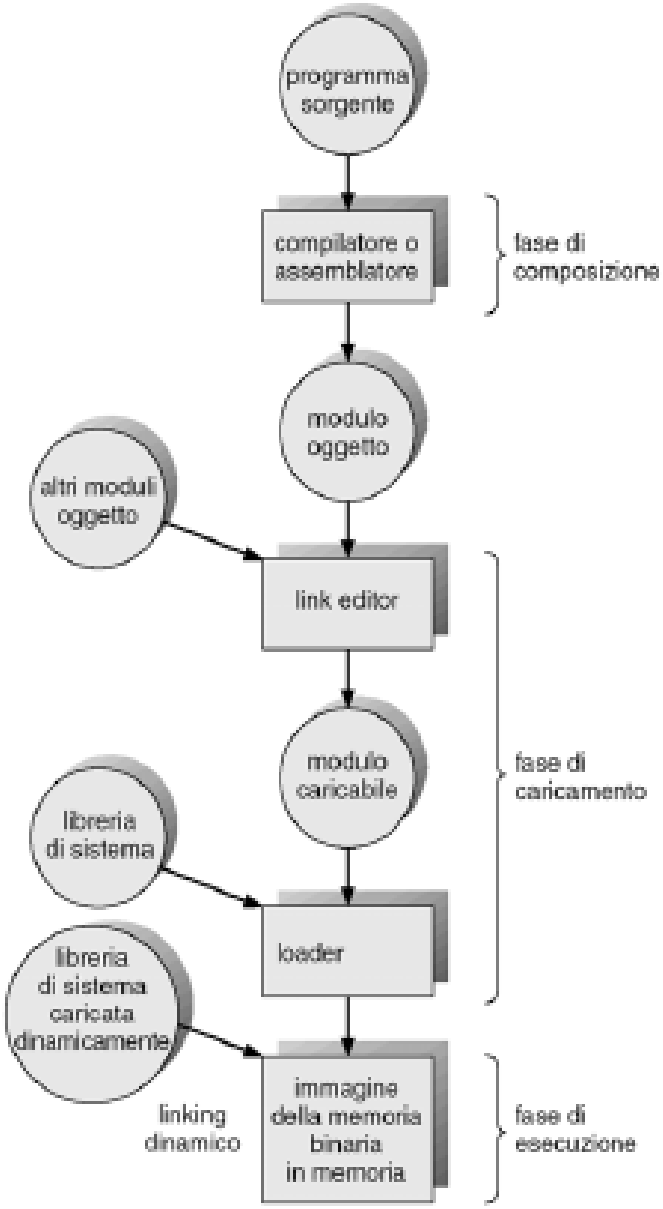
Gestione della memoria: fondamenti

La gestione della memoria mira a soddisfare alcuni ***requisiti*** :

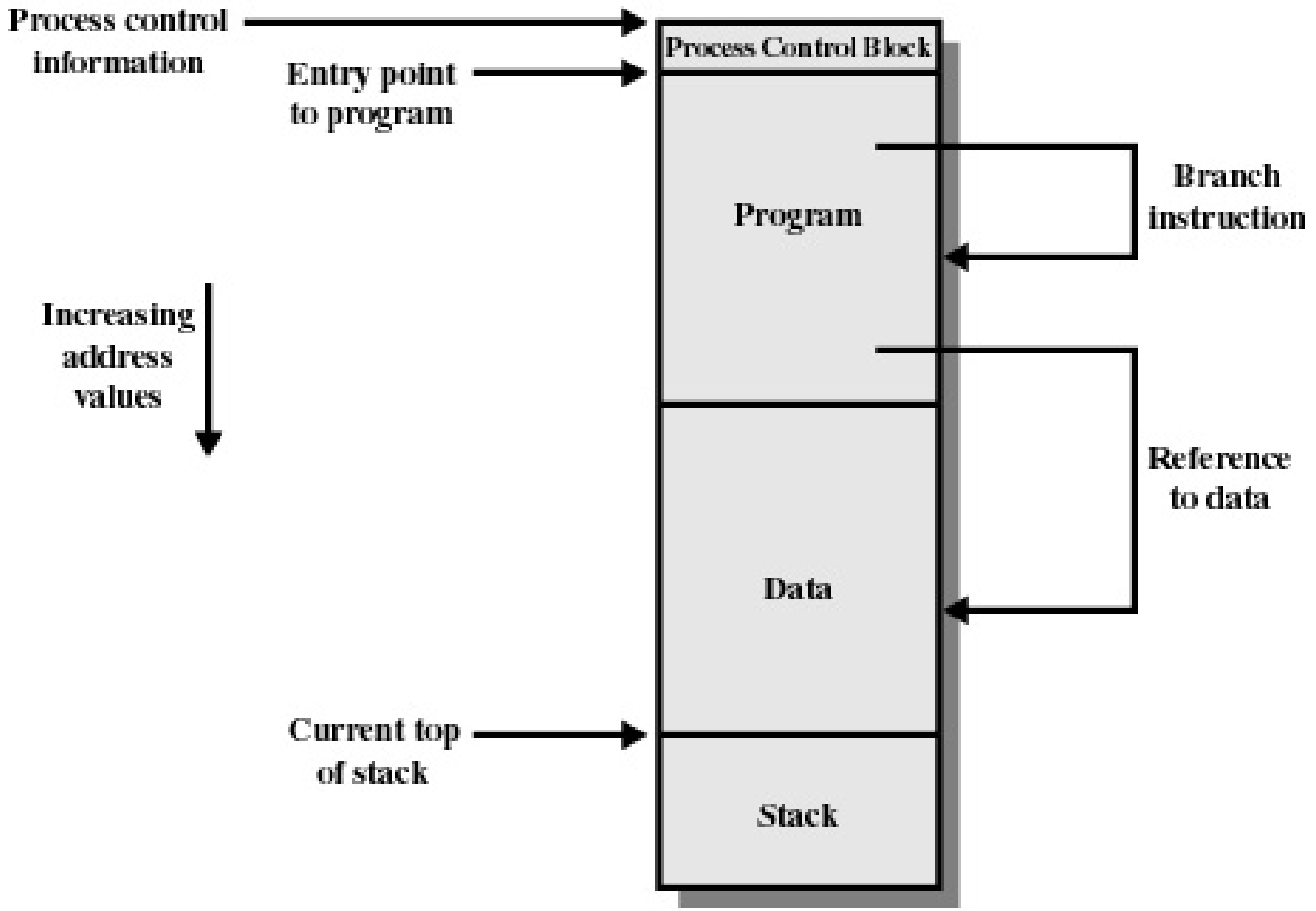
- Organizzazione logica: offrire una visione astratta della memoria, ovvero, *allocare e deallocare* memoria ai processi su richiesta.
- Organizzazione fisica: *tener conto di quale parte della memoria è allocata a ciascun processo*, ed effettuare eventuali scambi con il disco.
- Rilocazione: possibilità di spostare il codice da una parte all'altra della memoria.
- Protezione: della memoria allocata ai processi, e in particolare al sistema operativo.
- Condivisione: della memoria tra i processi, per aumentare l'efficienza d'uso della memoria stessa.

From the source program to the executable program

- Prima di poter essere eseguito, un programma passa attraverso diverse fasi.
  - ✓ Compilazione o assemblaggio
  - ✓ Linkage editing (fusione dei vari moduli)
  - ✓ Loading (caricamento dalla memoria di massa in memoria centrale)
- Ciclo di esecuzione di una istruzione:
  - ✓ Prelievo di una istruzione dalla memoria centrale in base al valore dell'indirizzo contenuto nel program counter;
  - ✓ Decodifica dell'istruzione
  - ✓ Prelievo di operandi dalla memoria ed esecuzione
  - ✓ Memorizzazione dei risultati in memoria



Address space structure



L'immagine della memoria occupata dai processi è costituita da tre segmenti: delle istruzioni (**codice**), dei **dati** e dello **stack**.
Tutti i 3 segmenti fanno parte dello stesso address space, ma è anche supportata la separazione tra spazio del codice e spazio dei dati.

Program addressing and loading

Generalità

Il collegamento delle istruzioni e dei dati agli indirizzi di memoria può essere effettuato in fasi diverse:

Fase di compilazione: se in fase di compilazione si conosce la locazione del processo in memoria, allora si può generare un *codice assoluto*; se la locazione di partenza cambia bisogna ricompilare il codice.

Fase di caricamento: se al momento della compilazione non è nota la locazione del processo in memoria, il compilatore deve generare un *codice rilocabile*.

- Il linking finale è rinviato fino all'istante di caricamento.
- Modificandosi l'indirizzo di partenza, c'è la necessità di modificare il solo codice utente per incorporare il valore cambiato.

Fase di esecuzione: se il processo può essere spostato, durante l'esecuzione, da un segmento di memoria a un altro, allora il collegamento deve essere ritardato fino al momento dell'esecuzione.

- Necessita di supporto hardware per il mapping degli indirizzi (registri base e registri limite).

Program addressing and loading

Il concetto di spazio degli indirizzi logico che viene collegato ad uno spazio degli indirizzi fisico diverso e separato è fondamentale nella gestione della memoria.

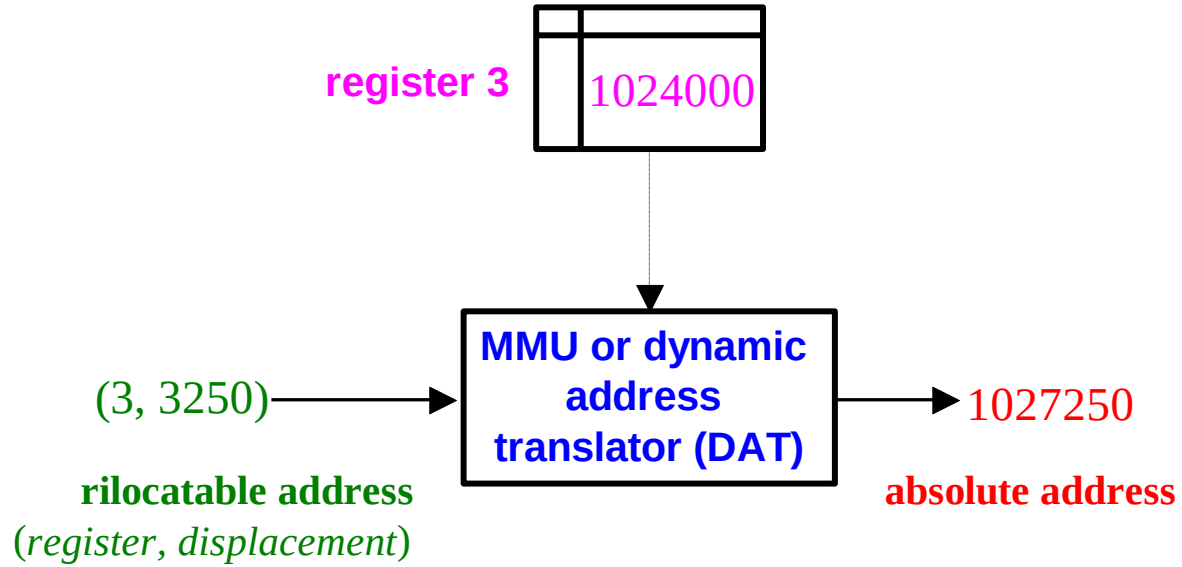
- **Indirizzo logico**: utilizzato dalla CPU. Detto anche indirizzo virtuale.
- **Indirizzo fisico**: indirizzo riferito alla memoria.

- Indirizzi logici e fisici **coincidono** nel caso di collegamento (binding) a compile time o load time.
- Possono essere **differenti** nel caso di collegamento (binding) a tempo di esecuzione. In tal caso, per effettuare dinamicamente la trasformazione da indirizzo virtuale a indirizzo reale, è necessario un apposito hardware di traduzione (Dynamic Address Translator - DAT o anche Memory Management Unit - MMU).

Relocatable addresses and translation

Memory Management Unit (MMU)

- Dispositivo hardware che realizza la trasformazione dagli indirizzi virtuali a quelli fisici in fase di esecuzione.
- Nello schema di una MMU, il valore nel registro di rilocazione è aggiunto ad ogni indirizzo generato da un processo nel momento in cui è trasmesso alla memoria.
- Il programma utente lavora con gli indirizzi logici; non avrà mai percezione degli indirizzi fisici reali.



Dynamic loading into the memory

- Affinché un processo possa essere eseguito, l'intero suo codice ed i suoi dati devono essere in memoria.
  - Ne consegue che la dimensione di un processo è vincolata dalla dimensione della memoria fisica.
- Per ottenere un migliore utilizzo dello spazio in memoria si adopera il dynamic loading:
  - una procedura non è caricata finché non viene richiamata;
  - una procedura inutilizzata non viene mai caricata;
  - tutte le procedure trovano posto su disco in un formato di caricamento rilocabile.
- Il programma principale viene caricato in memoria ed eseguito. Quando deve essere richiamata una procedura non in memoria si richiama il loader per effettuare il caricamento rilocabile.
  - Utile quando sono necessarie grandi quantità di codice per gestire situazioni che si presentano raramente.
  - Non richiede un supporto speciale da parte del sistema operativo, che si limita a rendere disponibili librerie che rendono facile l'uso del caricamento dinamico.

Dynamic linking and DLL

- La fusione è posposta fino alla fase di esecuzione.
- Senza questa funzione, ogni programma dovrebbe avere una copia delle librerie delle procedure del linguaggio.
  - Spreco di spazio sulla memoria di massa e su quella centrale.
- Una piccola parte di codice eseguibile, detta immagine o *stub*, serve per individuare la procedura di libreria desiderata (se residente in memoria) o come caricarla (se non residente).
- Alla prima esecuzione lo *stub* rimpiazza sé stesso con l'indirizzo della procedura (che viene caricata) e la esegue.
  - Vantaggio: è sufficiente sostituire una libreria su disco perché tutti i programmi si riferiscano automaticamente alla versione più recente.
- Il sistema operativo deve controllare se la procedura necessaria rientra nello spazio d'indirizzamento del processo.
  - Il linking dinamico è in grado di abilitare l'accesso ad un'area sottoposta a meccanismi di protezione e di permettere che processi multipli accedano alle stesse locazioni di memoria.
- Il collegamento dinamico è particolarmente utile nell'aggiornamento delle librerie di sistema tramite le *Dynamic Linking Libraries* (DLL).

Memory management algorithms

Mono-programmazione

Allocazione contigua di memoria

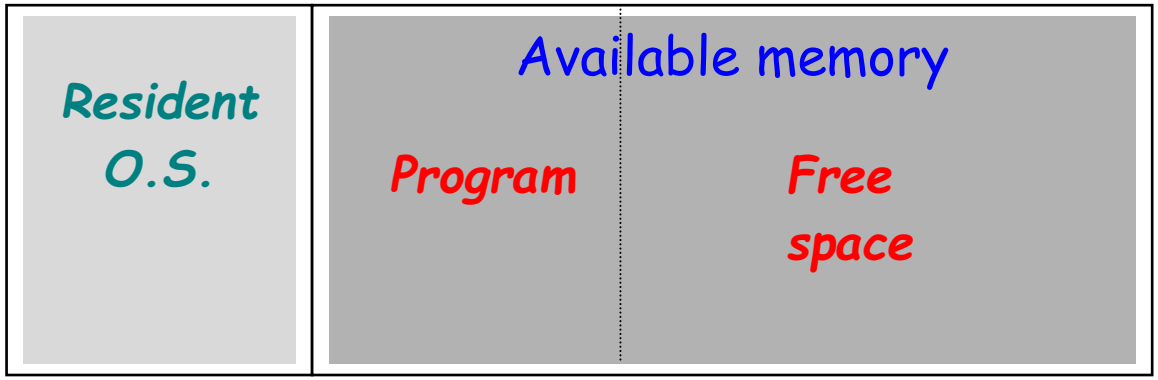
Multi- programmazione

- Partizionamento della memoria
  - Partizionamento statico,
  - Partizionamento dinamico,
  - Partizionamento rilocabile,
  - Partizionamento multiplo,
  - *Swapping*,
  - *Rolling*
- Paginazione (Segmentazione) reale
  - Paginazione reale
  - Segmentazione reale
  - Segmentazione e Paginazione reale
- Paginazione (Segmentazione) virtuale
  - Paginazione virtuale *(demand paging)*
  - Segmentazione virtuale
  - Segmentazione e Paginazione virtuale
  - Memoria virtuale multipla (MVS)

Mono-programmazione

Allocazione contigua di memoria

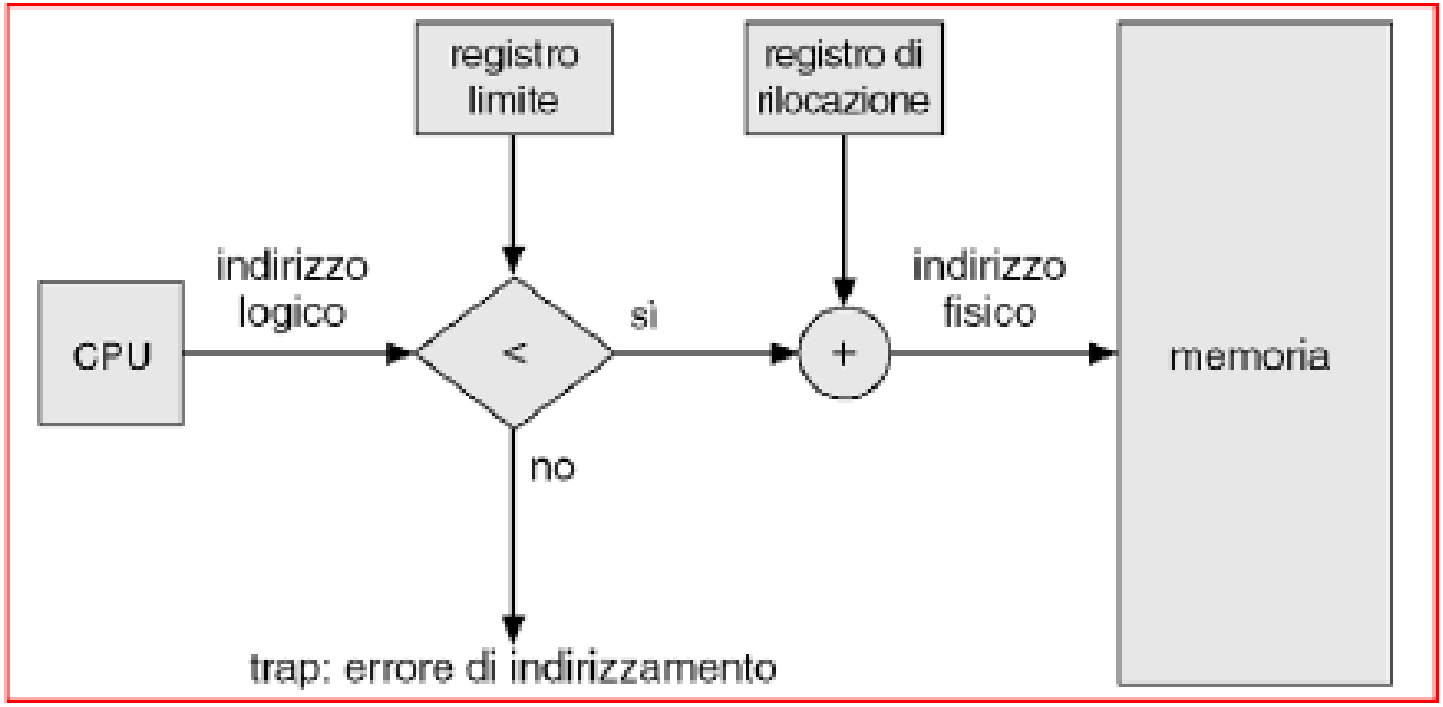
- La memoria centrale è normalmente divisa in due partizioni:
  1. Sistema operativo residente, di solito collocato nella memoria bassa con il vettore di interrupt.
  2. Processo-utente collocato di seguito nella memoria.



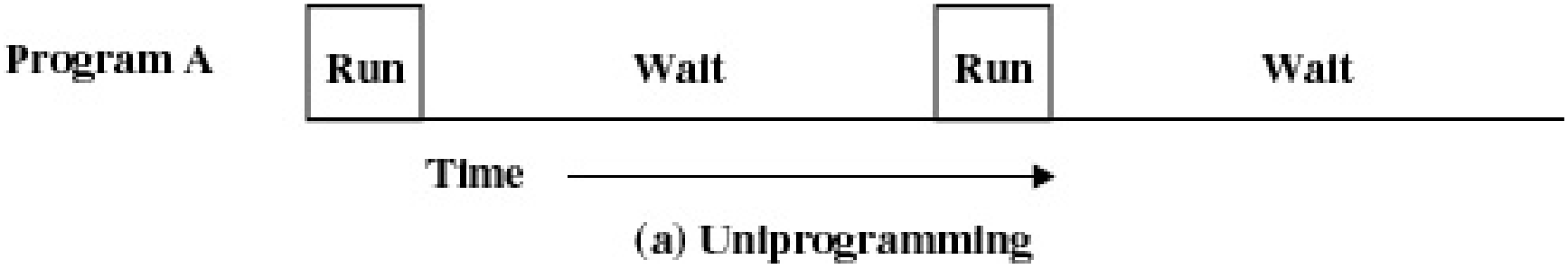
Mono-programmazione

Allocazione contigua di memoria

- Allocazione a partizione singola
  - Il registro di rilocazione serve per proteggere il sistema operativo dal processo-utente.
  - Il registro di rilocazione contiene il valore del più piccolo indirizzo fisico; il registro limite contiene l'intervallo degli indirizzi logici: ogni indirizzo logico deve avere un valore inferiore rispetto a quello del registro limite.



Mono-programmazione

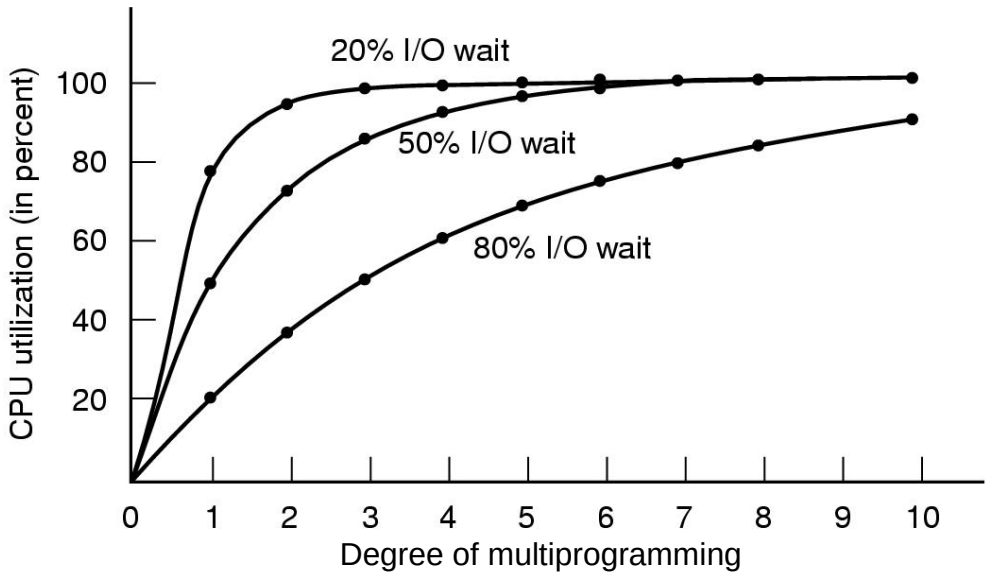


- La monoprogrammazione non sfrutta adeguatamente la CPU.
  - Idea: se un processo usa la CPU al 20%, 5 processi la usano al 100%.
  - Più precisamente, sia p la percentuale di tempo in attesa di I/O di un processo. Con n processi:

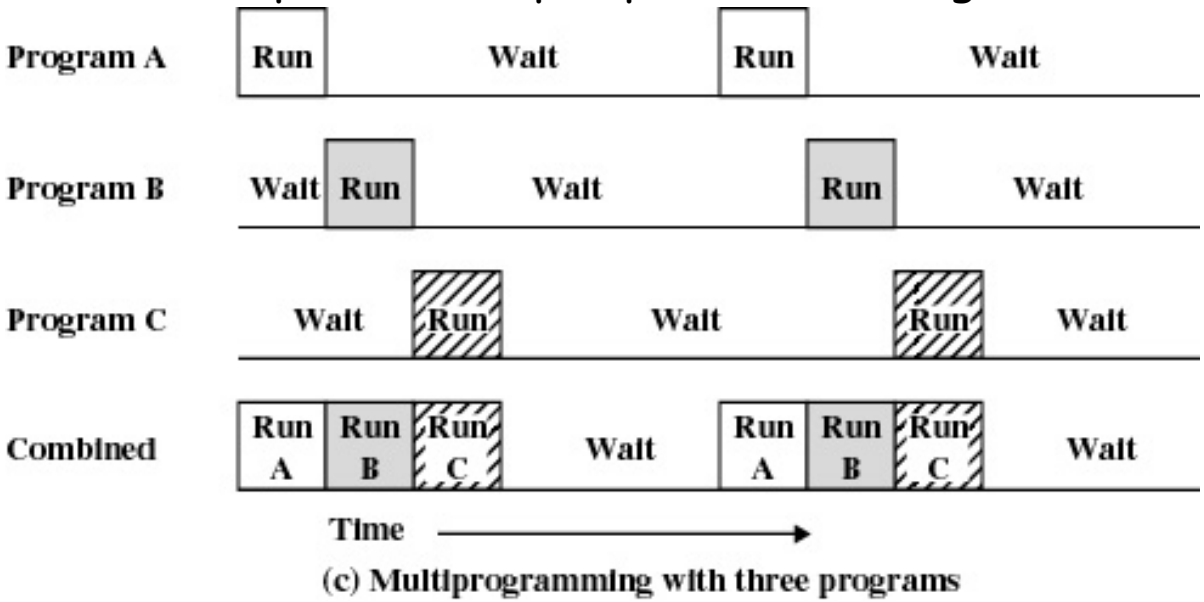
    utilizzo CPU = $1 - p^n$
- Maggiore il grado di multiprogrammazione, maggiore l'utilizzo della CPU.
  - Il modello è ancora impreciso: in realtà i processi non sono indipendenti;
  - un modello più accurato si basa sulla teoria delle code (cfr. di seguito).

Multi-programmazione

- **Pro**



- Quando un processo va in wait, il processore può passare ad eseguire un altro task



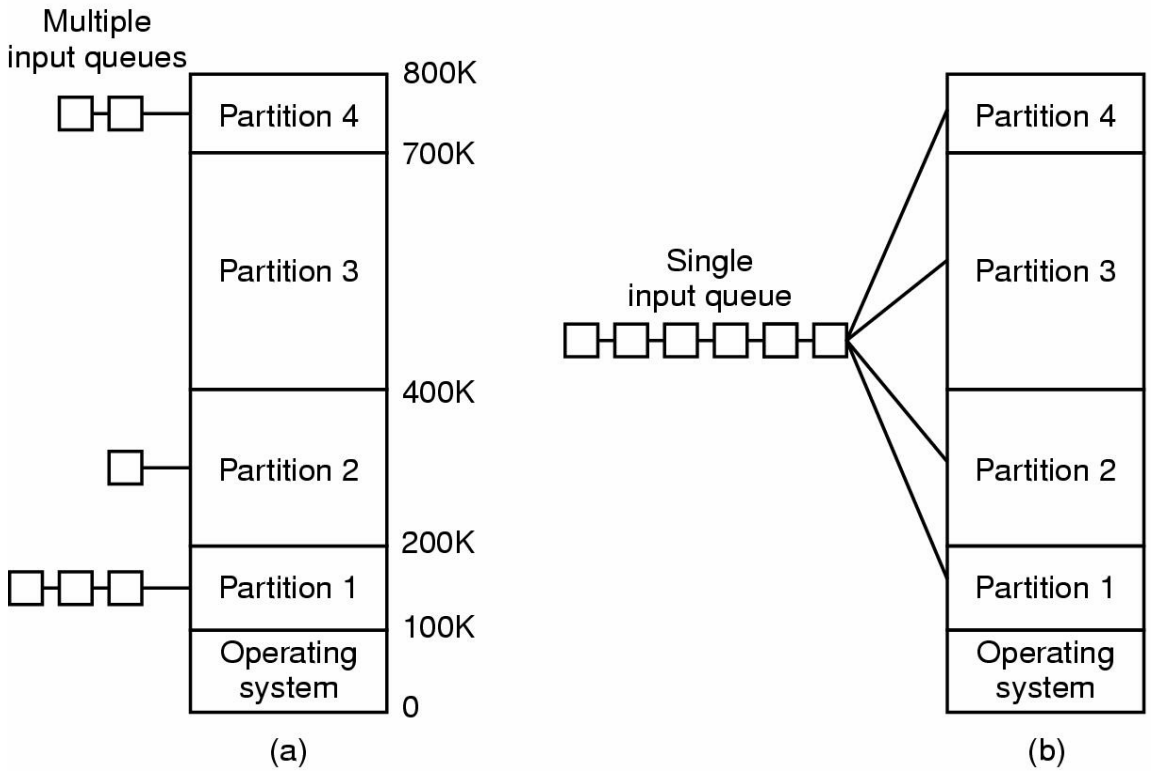
Multi-programmazione: Static partitioning

- La memoria disponibile è divisa in partizioni di ampiezza fissa (uguali o diverse).
- Il sistema operativo mantiene informazioni sulle partizioni allocate e quelle libere.

Partition number	Progr. Id.	Dimension	First byte	Status bit
1	Alfa	100k	100k	1
2		200k	200k	0
3	Word	300k	400k	1
4	Beta	100k	700k	0

- Quando arriva un processo, viene scelta una partizione tra quelle libere e gli viene completamente allocata.
- Porta a ***frammentazione interna***: la memoria allocata ad un processo è superiore a quella necessaria, e quindi in parte non è usata.
- Oggi questa politica viene usata solo su hardware povero o in sistemi real-time.

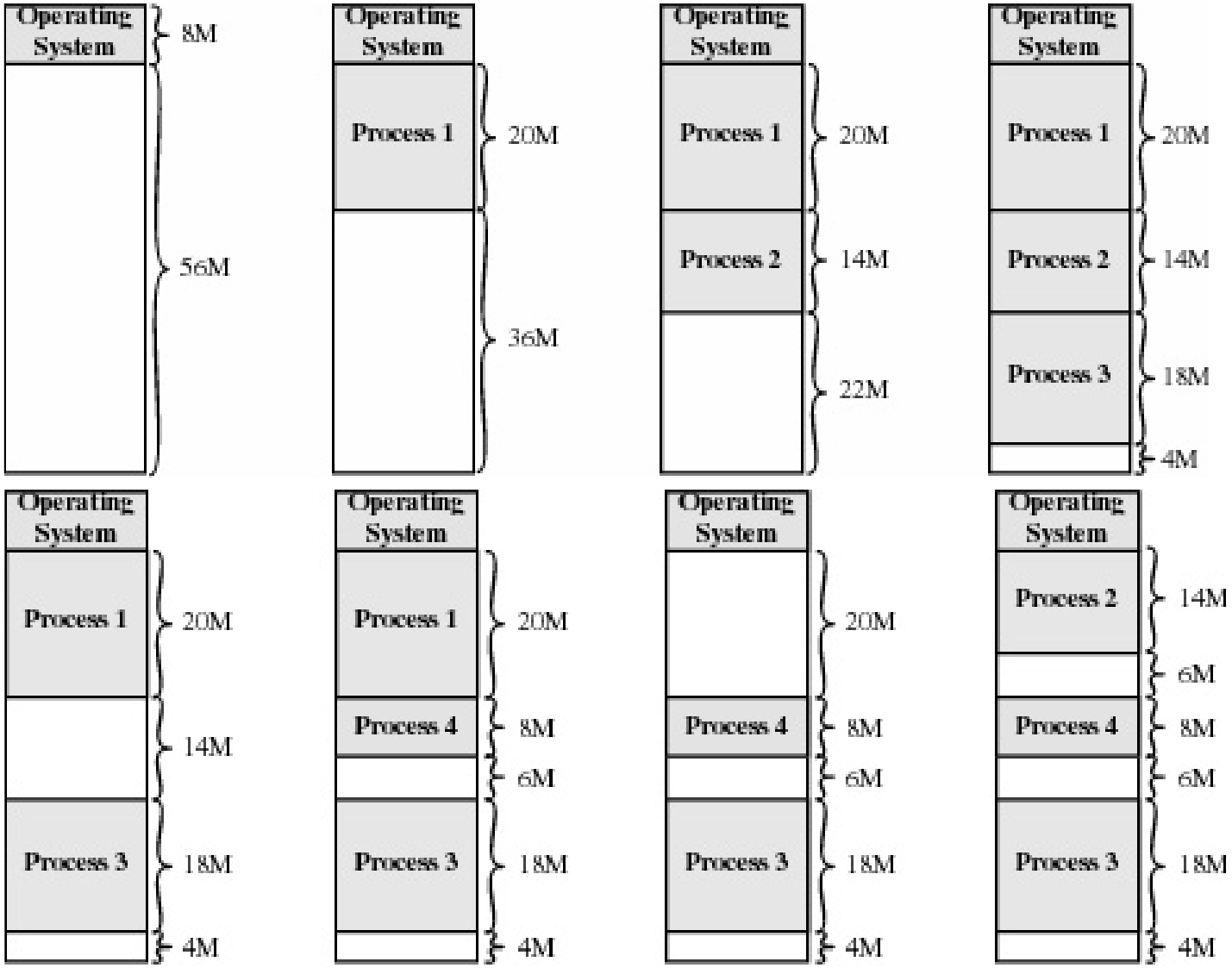
Multi-programmazione: Static partitioning



- Una coda per ogni partizione: possibilità di inutilizzo di memoria.
- Una coda per tutte le partizioni: come scegliere il job da allocare?
  - ***first-fit***: per ogni partizione libera, il primo job che ci entra,
  - ***best-fit***: il più grande che ci entra. Penalizza i job piccoli.

Multi-programmazione: Dynamic partitioning

Allocazione a partizioni multiple:



Multi-programmazione: Dynamic partitioning

- A ogni processo viene associata una partizione di dimensione esattamente pari alla quantità di memoria richiesta dal processo.
- Le partizioni sono variabili in numero e dimensioni
- Si possono formare buchi inutilizzati di memoria. Tali buchi costituiscono la **frammentazione esterna**.
- *Hole* (buco vuoto) – partizione di memoria centrale disponibile; blocchi di varie dimensioni sono sparsi nella memoria centrale.
- Quando un processo arriva, il sistema cerca un blocco libero abbastanza grande da ospitare il processo.
  - Allocato il processo, l'O.S. modifica le informazioni su:
  - a) partizioni allocate b) partizioni libere (*hole*)
- La memoria è assegnata fino a che le richieste possono essere soddisfatte.

Multi-programmazione: Dynamic partitioning

Tabella partizioni allocate

Partition numb.	Progr. Id.	Dimension	First byte	Status bit
1	Process2	14M	8M	1
2				0
3	Process4	8M	28M	1
4				0
5	Process3	18M	42M	1

Tabella aree libere (*hole*)

Hole numb.	Dimension	First byte	Status bit
1	6M	22M	1
2	6M	36M	1
3	4M	60M	1
4			0
5			0

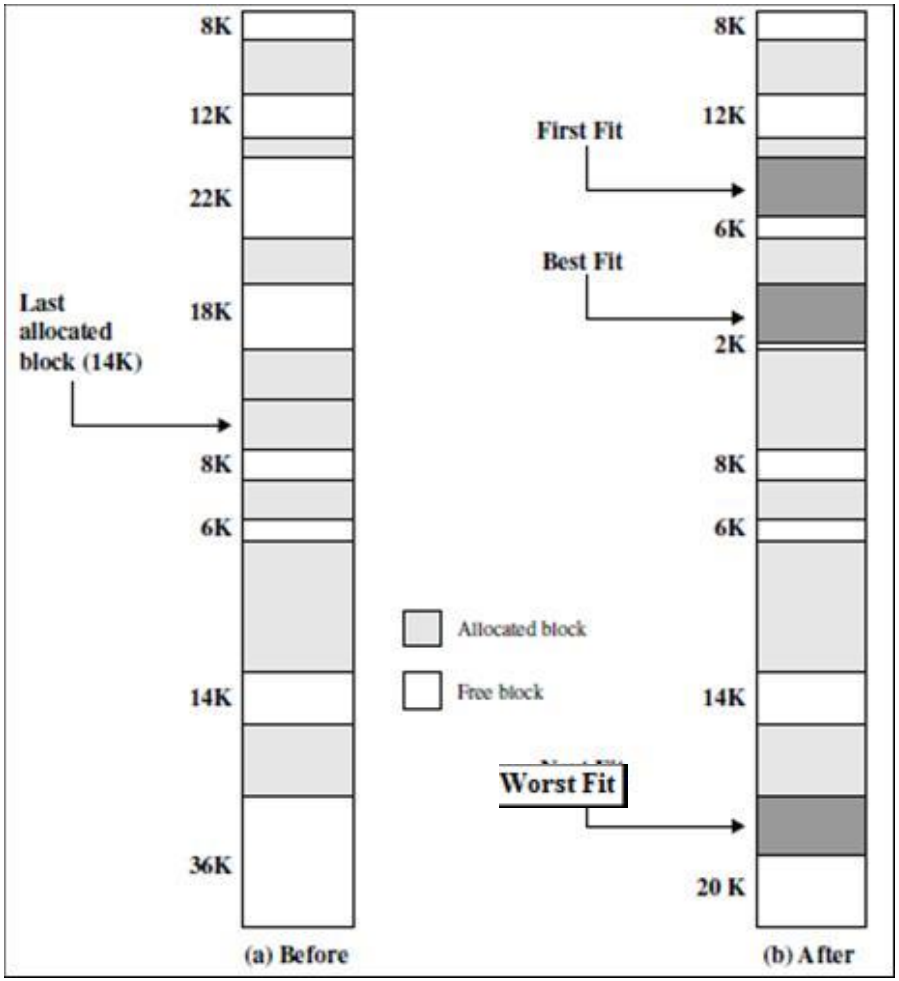
- Se non c'è disponibilità di una partizione libera sufficientemente grande per un processo:
  - l'O.S. può attendere fino al free di uno spazio sufficiente
  - l'O.S. può scorrere la coda di entrata alla ricerca di un processo con richieste inferiori.

Dynamic partitioning

- In accordo con il metodo di ordinamento (*sort*) della tabella dei buchi, la strategia di allocazione della memoria può essere:
  - **First-fit**: la tabella degli spazi liberi è ordinata in funzione crescente di indirizzo del primo byte:
    - ✓ assegna il primo blocco libero abbastanza grande da contenere il programma da allocare,
    - ✓ è la strategia più veloce,
    - ✓ può tendere a caricare i processi a indirizzi bassi di memoria.
  - **Best-fit**: la tabella degli spazi liberi è ordinata in funzione crescente della dimensione dell'hole:
    - ✓ assegna il più piccolo blocco libero capace di contenere il programma da allocare,
    - ✓ produce i più piccoli spazi liberi residui e, quindi, la più piccola frammentazione esterna,
    - ✓ richiede, pertanto, la più frequente eventuale compattazione della memoria,
  - **Worst-fit**: la tabella degli spazi liberi è ordinata in funzione decrescente della dimensione dell'hole:
    - ✓ assegna il più grande blocco libero capace di contenere il programma da allocare,
    - ✓ produce i più grandi spazi liberi residui e, quindi, la più grande frammentazione esterna,
    - ✓ è la strategia più lenta.

Dynamic partitioning

Si supponga di dover allocare 16 Kb



- In generale, l'algoritmo migliore è il first-fit.
- Best-fit tende a frammentare molto. Worst-fit è il più lento.

Dynamic partitioning: Fragmentation

- **Frammentazione esterna** – c'è abbastanza spazio totale di memoria centrale per soddisfare una richiesta, ma gli spazi disponibili non sono contigui.

Soluzioni possibili:

- Ridurre la frammentazione esterna attraverso la **compattazione**:
  - Fondere i contenuti della memoria centrale per avere tutta la memoria centrale libera in un grande blocco.
  - La compattazione è possibile *solo se* la rilocazione è dinamica ed è fatta al momento dell'esecuzione (basta variare il valore del registro base e spostare codice e dati).
- Ridurre la frammentazione permettendo **spazi di indirizzamento fisico non contigui** mediante due tecniche principali:
  - Paginazione.
  - Segmentazione.

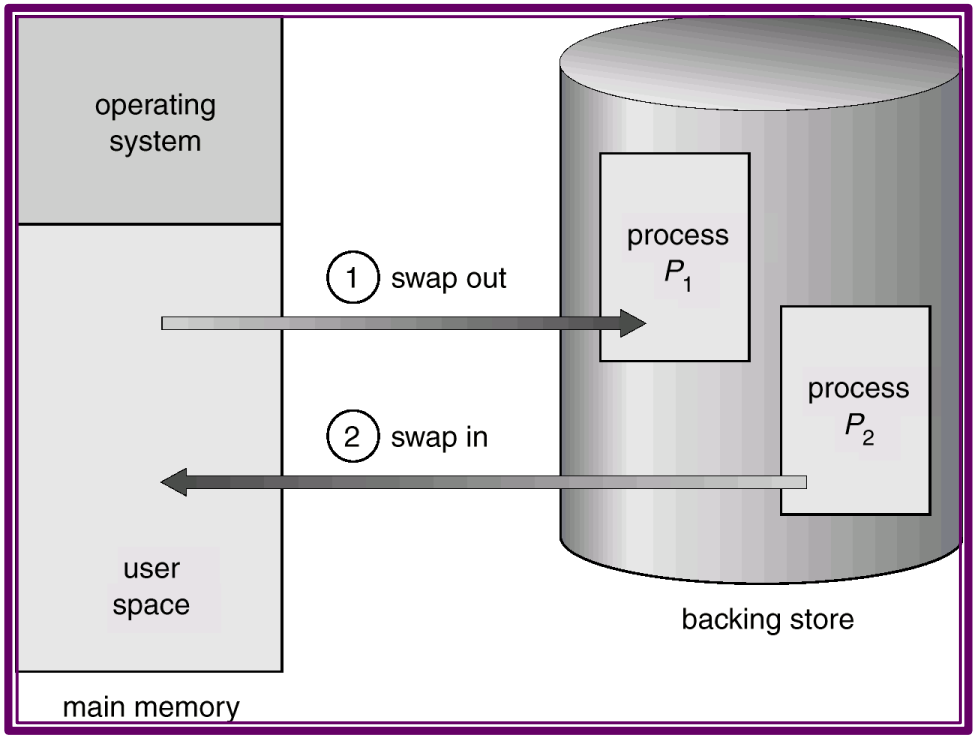
Relocatable partitioning

- Fa uso della **compattazione** per rilocare i processi sì da renderli contigui e far emergere tutta la memoria libera come un unico blocco, riducendo la frammentazione esterna
  - Il compattamento è possibile *solo* se la rilocazione è dinamica e avviene durante l'esecuzione.
  - Problema di I/O
    - ☞ Un programma che è impegnato in un'operazione di I/O viene trattenuto in memoria.
    - ☞ L'I/O viene effettuato solo nei buffer del SO.

Multiple partitioning

- Si basa sull'assunzione che lo spazio degli indirizzi di un programma non debba essere tutto contiguo in memoria centrale.
- Lo spazio degli indirizzi può perciò essere suddiviso in porzioni, che vengono allocate in parti diverse (non contigue) della memoria centrale.
- Un esempio di partizionamento multiplo è quello successivamente presentato come "*Segmentation*".

Swapping and Rolling



- Un processo può essere temporaneamente scambiato di posto (*swapped*) spostandolo dalla memoria centrale ad una memoria temporanea (***Swap-out***), e poi riportato in memoria centrale per continuarne l'esecuzione (***Swap-in***).
- ***Roll out***, ***Roll in*** – variante dello swapping usata per algoritmi di schedulazione basati sulla priorità;
  - un processo a bassa priorità è scambiato in modo che uno ad alta priorità possa essere caricato ed eseguito.

Swapping and Rolling

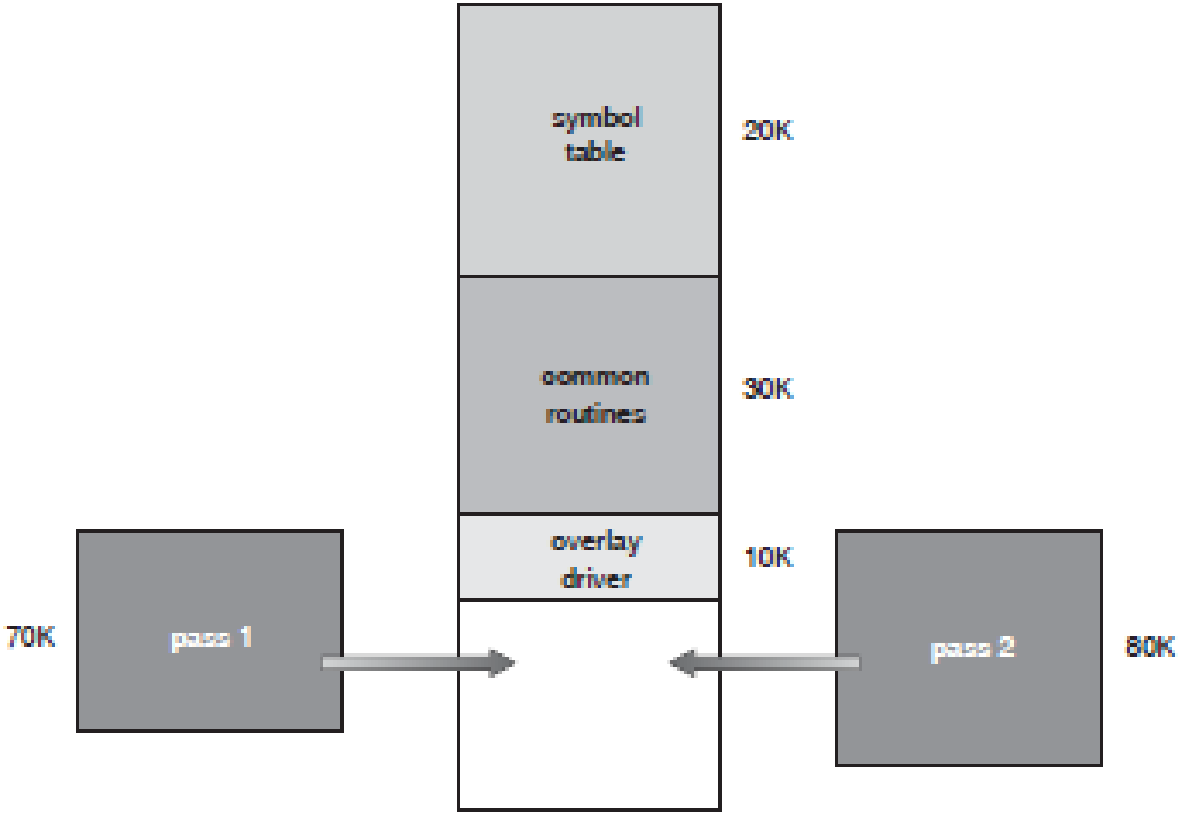
- ***Backing storage*** – Memoria temporanea – disco veloce abbastanza capiente da accogliere le copie delle immagini della memoria centrale per tutti i processi e da poter consentire l'accesso diretto a tali immagini della memoria.
- Non si esegue mai swap quando c'è un I/O in corso.
- Si esegue I/O solo nei buffer del sistema operativo.
- Normalmente un processo scambiato viene preferibilmente ricaricato in memoria sempre nella medesima posizione. Ciò è sempre vero nel caso del Rolling.

Swapping

- In casi di swapping i cambi di contesto sono piuttosto lunghi. Per un efficiente utilizzo della CPU è desiderabile che il tempo di esecuzione di ciascun processo sia lungo rispetto al tempo di swap.
- La maggior parte del tempo di swap è tempo di trasferimento; il tempo totale di trasferimento è direttamente proporzionale alla quantità di memoria spostata.
- Di norma lo swapping è disabilitato, ma quando l'uso della memoria centrale supera una data soglia, la tecnica viene messa in funzione salvo poi ad essere nuovamente disabilitata quando scende il carico del sistema.
- Versioni modificate di swapping si trovano in molti sistemi operativi (ad esempio UNIX, Linux e Windows).

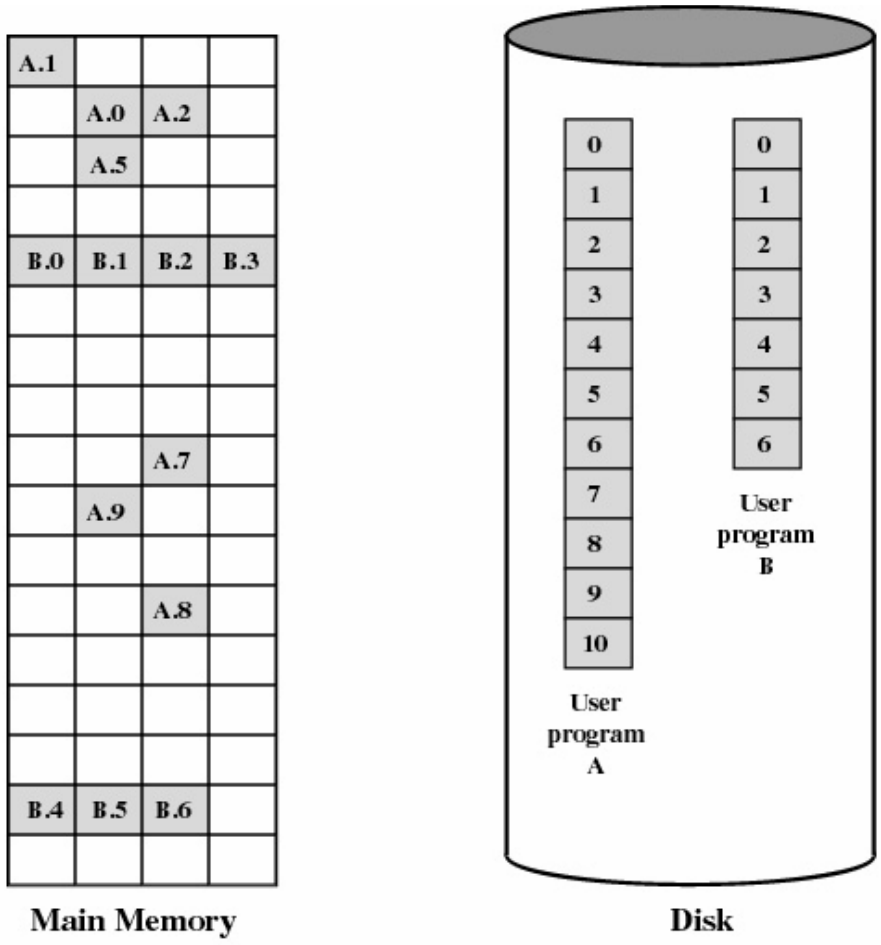
Overlay

- Mantenere in memoria solo le istruzioni e i dati che servono in un dato istante.
- Necessario quando un processo è più grande della memoria allocatagli.
- Gli overlay sono implementati dall'utente, senza supporto particolare dal sistema operativo.
- La programmazione di un programma a overlay è complessa.



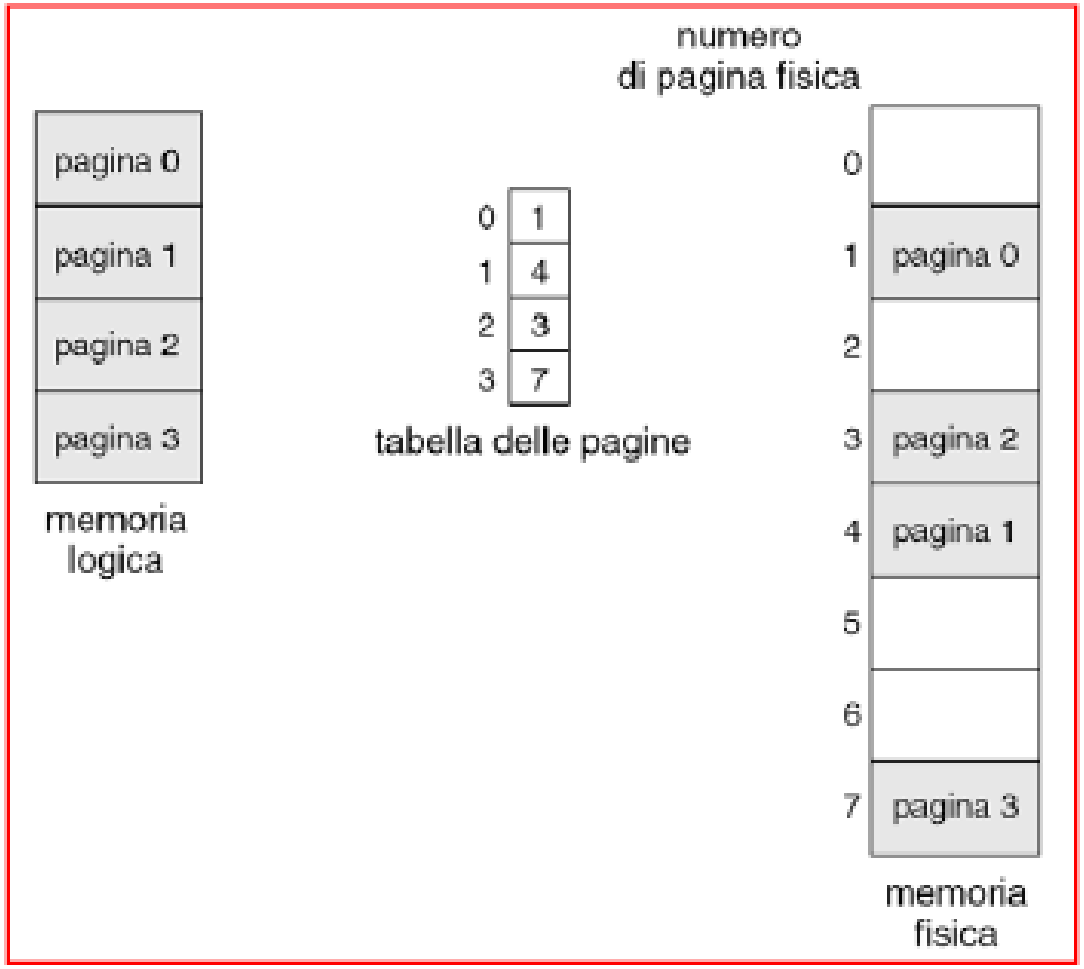
Paging

- Lo spazio degli indirizzi logici (*address space*) di un processo è suddiviso in parti (tutte delle stesse dimensioni) chiamate **pagine**.
- La memoria fisica è divisa in blocchi detti **frame** o *page-frame* (la cui dimensione è una potenza di 2, fra 512B e 16MB). Si mantiene traccia di tutti i frame attraverso la *Memory Block Table - MBT* (cfr. in seguito).



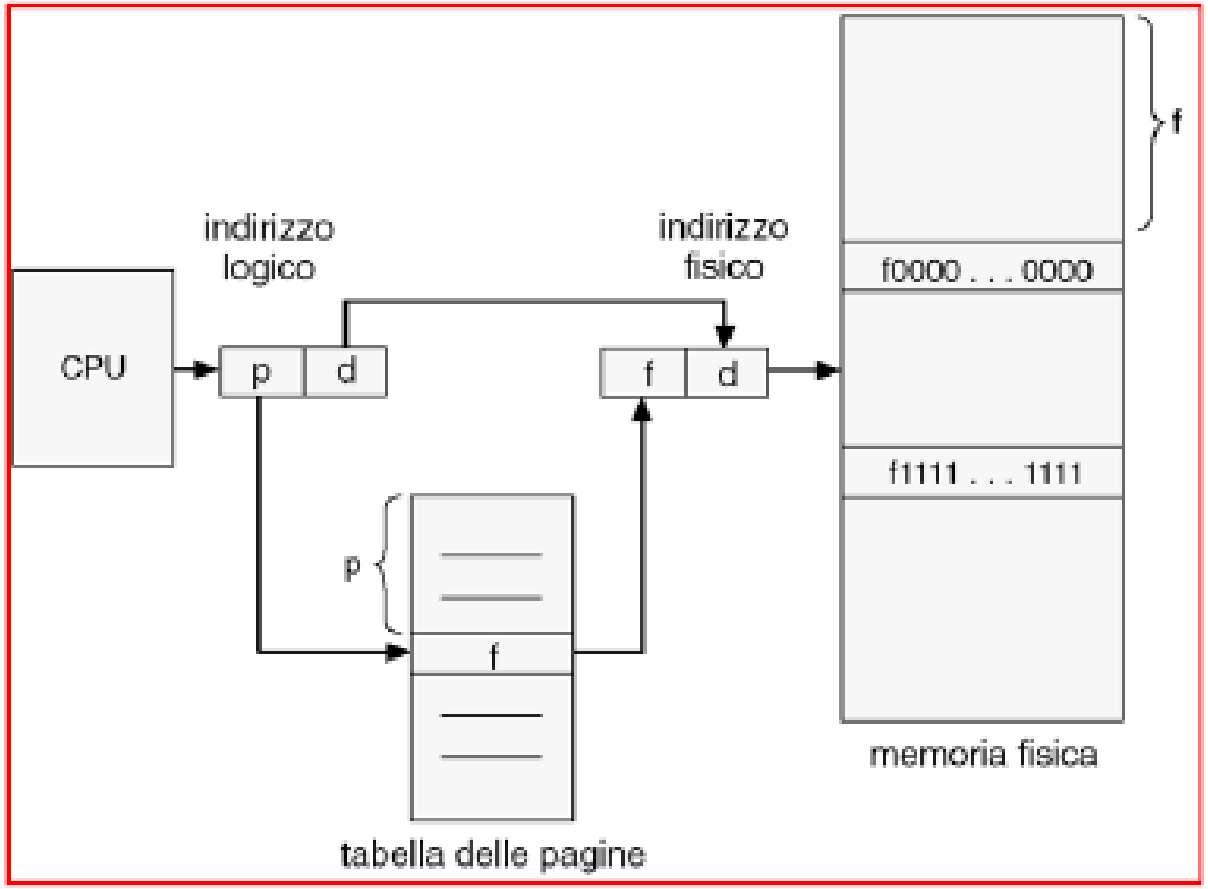
Page Map Table (PMT)

- L'O.S. mantiene una *page map table* per ciascun processo
  - essa tiene traccia del frame in cui è caricata ogni pagina



Address translation

- Ogni indirizzo generato dalla CPU è diviso in due parti:
  - ***Numero di pagina (p)*** – usato come indice nella *tabella delle pagine*
  - ***Spiazzamento nella pagina (d)*** – combinato con l'indirizzo di base per calcolare l'indirizzo di memoria fisica che viene mandato all'unità di memoria centrale.



Paging: memory protection

- La protezione della memoria è implementata associando bit di protezione ad ogni frame.
- **Valid bit** collegato ad ogni entry nella page table:
  - "valid" = indica che il page frame è nello spazio logico del processo, e quindi è legale accedervi
  - "invalid" = indica che il page frame non è nello spazio logico del processo → violazione di indirizzi

Pages and Frames

La dimensione della pagina logica (pari a quella della pagina fisica) è definita dall'hardware.

- Se la dimensione massima di un indirizzo logico è 2^m (dimensione massima della memoria) e abbiamo a disposizione n bit per il sistema di indirizzamento intra page-frame, adopereremo m-n bit per il numero di pagina e n per il displacement.

- Per eseguire un processo di dimensione di n pagine, bisogna trovare n frame liberi e caricare il programma → Memory Block Table.
- Occorre poi impostare una *Page Map Table* - PMT per tradurre gli indirizzi logici in indirizzi fisici.

Non esiste frammentazione esterna, ma frammentazione interna.

Frame fragmentation

- Un processo che abbisogni di n pagine più un byte comporta l'allocazione di n+1 frame con un ultimo frame quasi completamente frammentato (caso peggiore di frammentazione interna).
  - La frammentazione interna in media è di mezza pagina per processo.
  - È auspicabile avere piccole dimensioni di pagina per ridurre la frammentazione interna, tuttavia:
    - con pagine piccole cresce la dimensione della PMT;
    - gli accessi a disco sono maggiormente efficienti con grossi quantitativi di dati da trasferire.

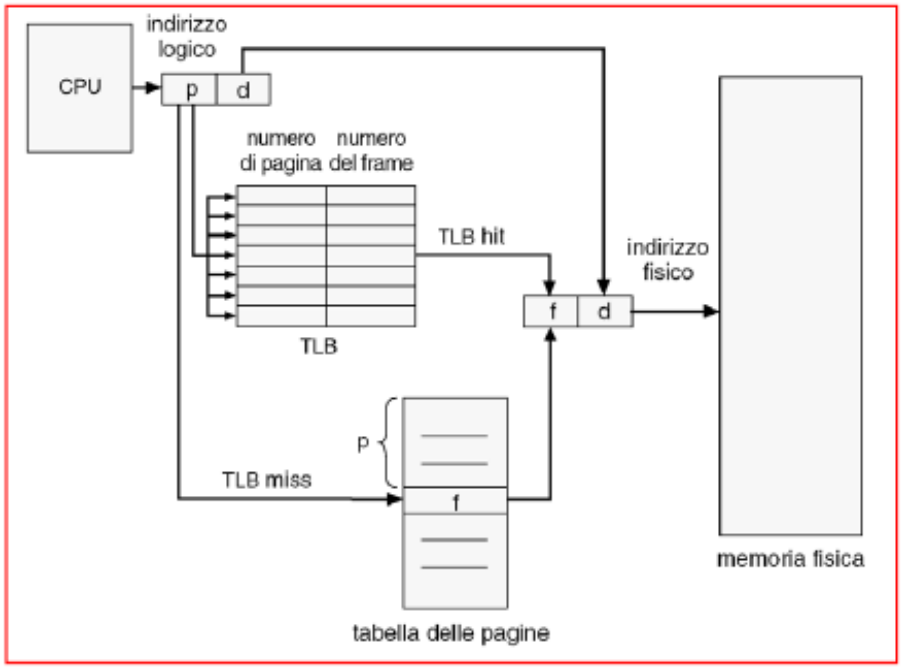
- La dimensione tipica attuale delle pagine è compresa tra 4 kB e 8 kB.
  - Solitamente ogni elemento della PMT è lungo 4 byte (2^{32}frame nella memoria fisica) quindi con frame da 8 kB si può indirizzare una memoria fisica da 2^{45}byte (32 TB).

Page Map Table access

- Un puntatore alla tabella delle pagine è memorizzato nel PCB relativo al processo.
- La tabella delle pagine è mantenuta in registri dedicati (page-map table di piccole dimensioni) o più di frequente nella memoria centrale.
- Il ***Page Table Base Register*** (PTBR) punta alla tabella corrispondente al processo in esecuzione.
  - Il cambiamento di tabella richiede di modificare il solo PTBR riducendo di molto il tempo per il context switching.
- In questo schema ogni accesso a dati/istruzioni richiede due accessi alla memoria: uno per la tabella delle pagine e uno per i dati/istruzioni.
  - L'accesso alla memoria centrale è rallentato di un fattore 2.

Translation Look-aside Buffer (TLB)

- Il problema dei 2 accessi alla memoria può essere risolto attraverso l'uso di una speciale piccola cache per l'indicizzazione veloce, detta *Translation Look-aside Buffer* –TLB.
  - Si tratta di un hardware velocissimo ma molto costoso. Le dimensioni tipiche di un TLB tipicamente consentono di mantenere da 64 a 1024 elementi.
- Memoria associativa per ricerca parallela:
  - Traduzione dell'indirizzo (p, d): Se p si trova nella memoria associativa, si ottiene il frame#. Altrimenti si ottiene il frame# dalla page map table in memoria centrale



Effective Access Time (EAT)

- L'utilizzo di un TLB può richiedere meno del 10% del tempo rispetto al caso di memoria non mappata.
- Per il calcolo del tempo effettivo di accesso alla memoria centrale occorre fare un calcolo pesato di tipo probabilistico.
  - Sia: Associative Lookup = **ε** unità di tempo.
  - Si assuma che il tempo di ciclo di memoria sia **ß** unità di tempo.
  - Tasso di accesso con successo (hit ratio) **α** – frequenza delle volte che un particolare numero di pagina viene trovato nella TLB.

Tempo di accesso effettivo (EAT) = (ß+ε)α + (2ß+ε)(1–α) = 2ß + ε – ßα = ß(2 -α) + ε

Esempio: 20 nsec per accedere al TLB, 100 nsec per accedere alla memoria, hit ratio 80%.
In caso di Hit: 120 nsec per un accesso mappato.
In caso di TLB miss: 20 nsec per l'accesso al TLB (miss), 100 nsec per recuperare la PMT dalla memoria e 100 nsec per l'accesso.

EAT = 0.8 x 120 + 0.2 x 220 = 140 nsec

The Memory Block Table (MBT)

Task ID	# page	↑ PMT	S bit
63	4	7	0
25	6	9	0
44	12	8	0
50	7	2	1
52	8	6	1
54	6	5	1
51	3	3	1
49	4	1	1
53	5	4	1

Job (Task) Table

B	Task ID	P	S bit
0	51	0	1
1	50	0	1
2	63	3	0
3	53	0	1
4	49	1	1
5	44	6	0
6	44	4	0
7	25	3	0
8	25	4	0
9	54	3	1
10	52	0	1
11	63	1	0
12	54	1	1
13	35	1	0

Memory Block Table

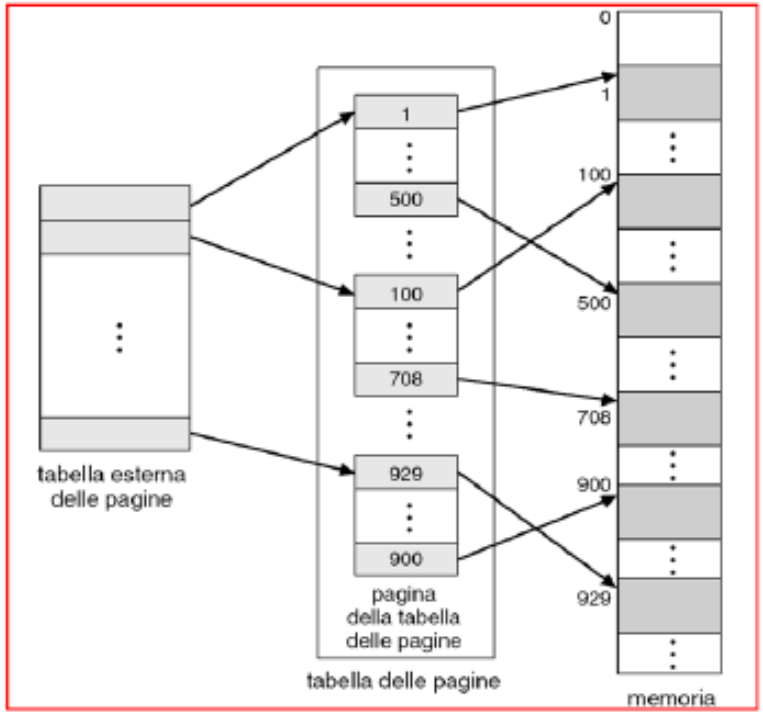
Paginazione a più livelli

- La maggioranza dei PC supporta vasti spazi di indirizzamento logico (da 2^{32} kB a 2^{64} kB) con un conseguente forte impatto sulla dimensione della Page Map Table.
- Con uno spazio a 32 bit (2^{32} kB) e pagine di 4kB (2^{12} kB), la PMT potrebbe avere fino a 1 milione di elementi (2^{20}).
- Se ciascun elemento occupa 4 byte la PMT occuperà 4MB.

- Occorrono dunque tecniche efficienti per la strutturazione della tabella delle pagine,
  - **Paginazione gerarchica**.
  - Tabelle delle pagine con hashing.
  - Tabella delle pagine invertita.

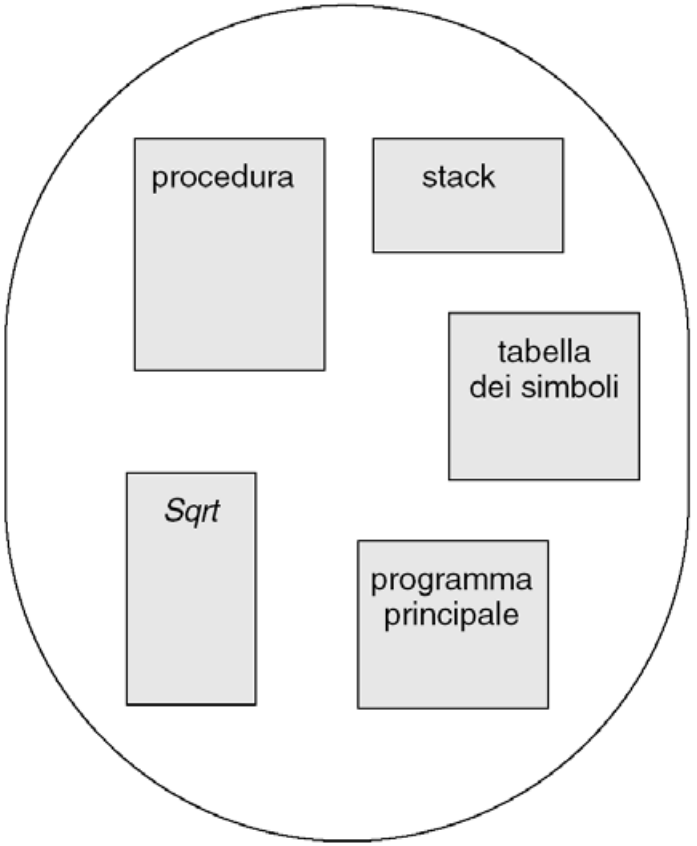
Paginazione gerarchica

- Per ridurre l'occupazione della tabella delle pagine, si pagina la PMT stessa.
- Si suddivide lo spazio degli indirizzi logici in più tabelle di pagine.
  - Occorrerà una External Page Map Table (EPMT) contenente il riferimento alle pagine della PMT.
- Solo le pagine effettivamente usate sono allocate in memoria RAM.



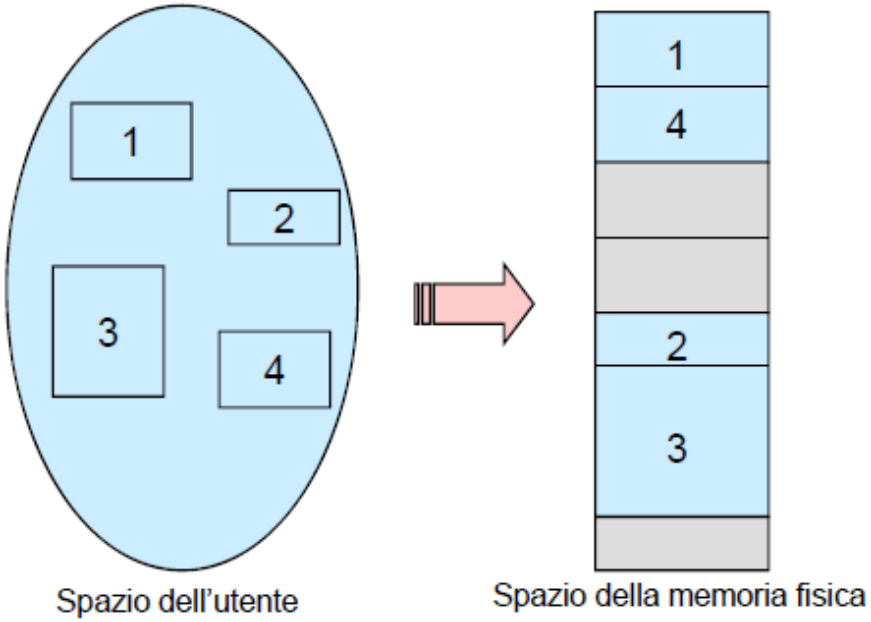
Segmentation

- Schema di gestione della memoria centrale che supporta il punto di vista dell'utente della memoria centrale.
- Lo spazio di indirizzamento logico è un insieme di segmenti. Un segmento è un'unità logica (programma principale, procedura, funzione, metodo, oggetto, variabili locali, stack, array).



User's View of a Program

Segmentation



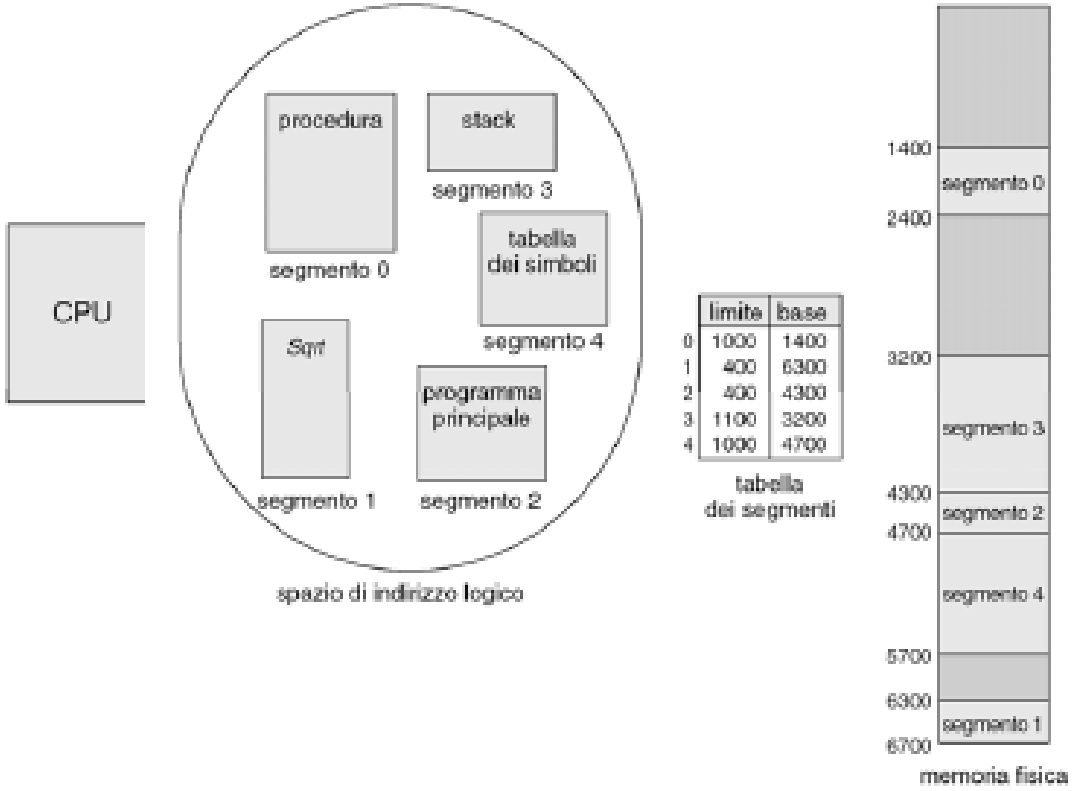
Logical View of Segmentation

La frammentazione

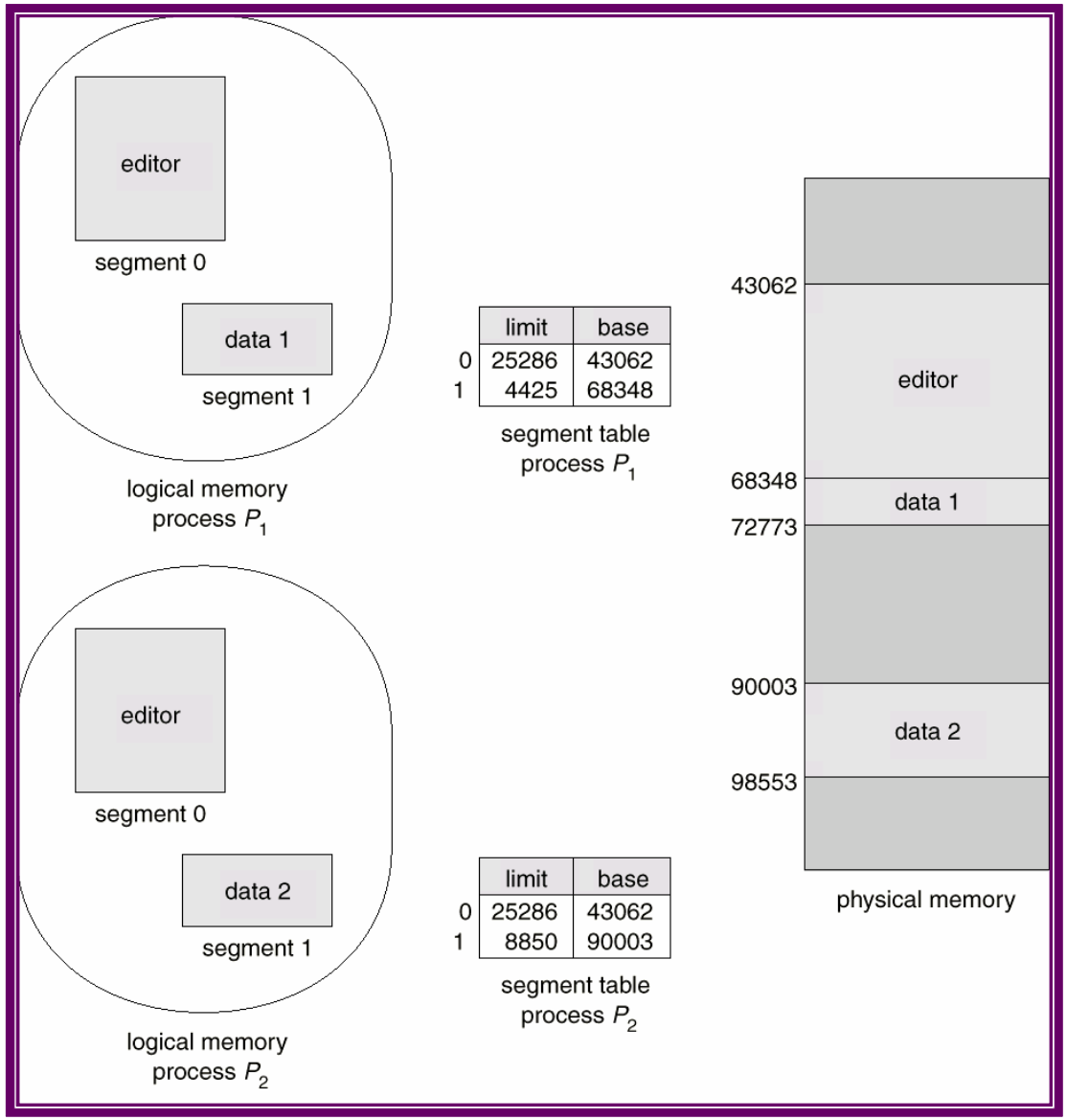
- Poichè i segmenti sono di varia lunghezza, l'allocazione in memoria ha problemi di frammentazione esterna analoghi a quelli del partizionamento dinamico.
  - Quando tutti i blocchi in memoria sono troppo piccoli per ospitare un segmento occorrerà attendere per la liberazione di blocchi o eseguire la compattazione.
- Se la dimensione media dei segmenti è piccola, sarà basso il livello di frammentazione.

Segmentation: the addressing

- Un indirizzo logico è composto da due parti: <numero del segmento, spiazzamento>
- La **Tabella dei segmenti** – mappa gli indirizzi bidimensionali in indirizzi fisici; ogni elemento della tabella ha:
  - Una base – contiene l'indirizzo fisico di partenza in cui il segmento risiede in memoria centrale.
  - Un limite – specifica la lunghezza del segmento stesso.
- Il *Registro base della tabella dei segmenti* (STBR) punta all'indirizzo in memoria della tabella dei segmenti.
- Il *Registro della lunghezza della tabella dei segmenti* (STLR) indica il numero dei segmenti usati dal programma.
- Il *numero del segmento* **s** è legale se **s** < STLR.



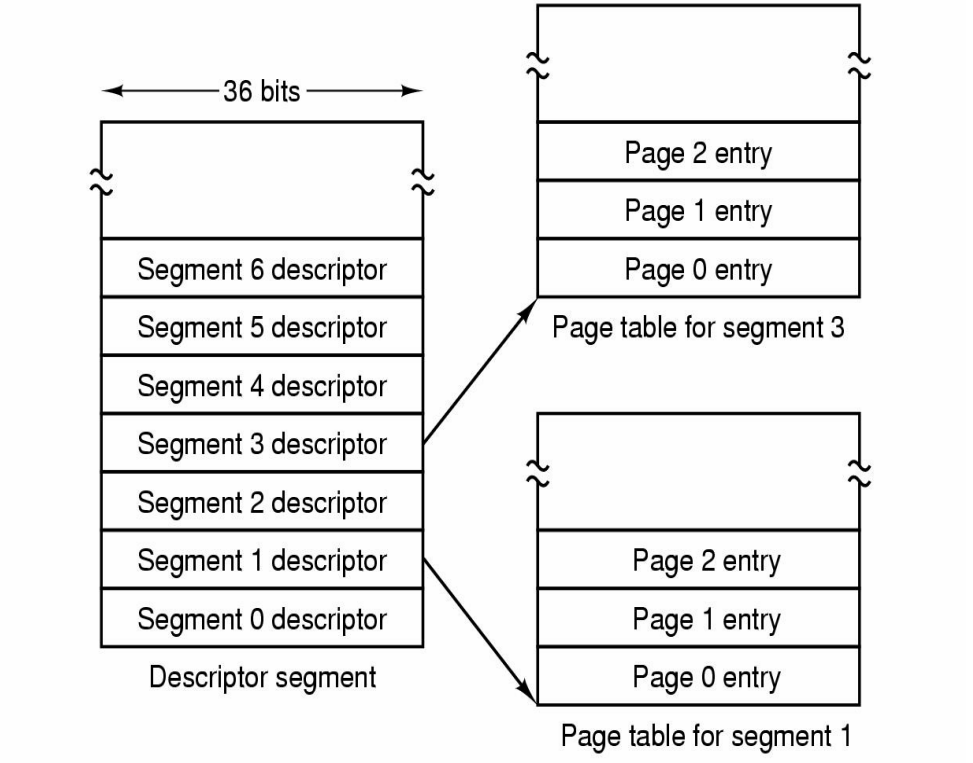
Segmentation: sharing of segments

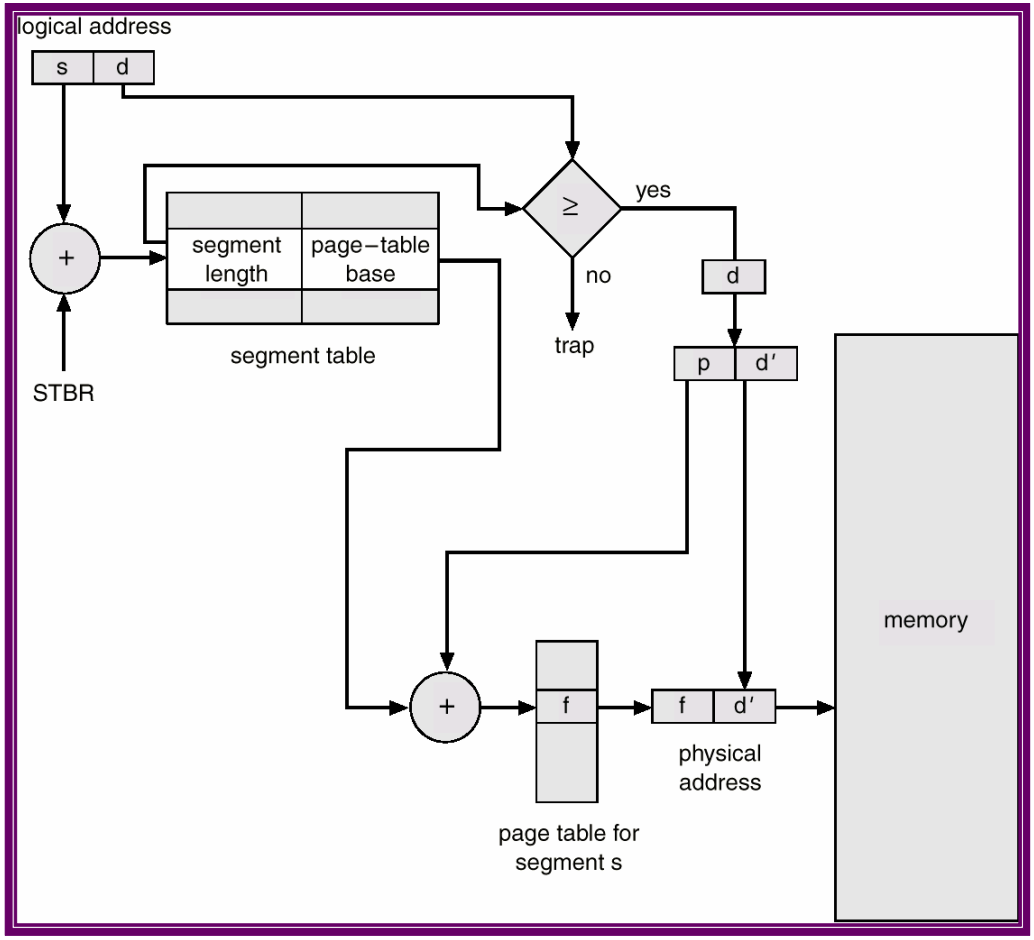


Condivisione dei segmenti

Segmentation and Pagination

- MULTICS (e con lui tutti i SO derivati) risolve i problemi della frammentazione esterna e della lunghezza dei tempi di ricerca attraverso la paginazione dei segmenti.
- Nella segmentazione con paginazione la tabella dei segmenti non contiene l'indirizzo base di un segmento quanto piuttosto l'indirizzo della tabella delle pagine per quel segmento.





Virtual Memory: the locality principles

- I **principi di località** enunciano che, se la CPU sta eseguendo una data istruzione o se un'istruzione sta operando con un certo dato, con molta probabilità le prossime istruzioni da eseguire saranno ubicate nelle vicinanze di quella in corso o faranno riferimento ancora allo stesso dato.

In particolare:

- Il **principio di località spaziale** → se, all'istante t la CPU fa riferimento all'indirizzo di memoria x, allora è molto probabile che all'istante t+dt faccia riferimento all'indirizzo x+dx
- Il **principio di località temporale** → se, all'istante t la CPU fa riferimento all'indirizzo di memoria x, allora è molto probabile che all'istante t+dt faccia riferimento ancora all'indirizzo x

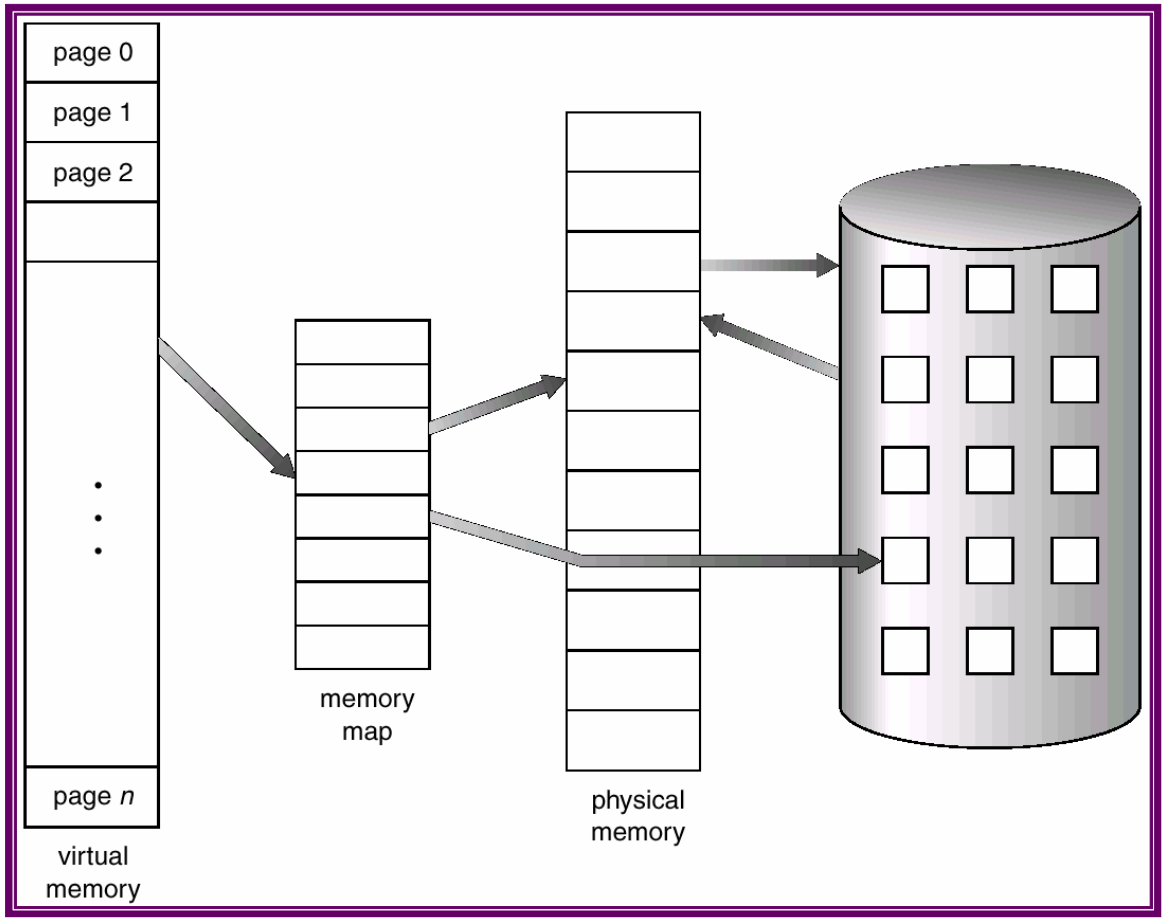
The locality principles

Principi probabilistici e non deterministici

- Il significato: durante la normale esecuzione dei programmi, la CPU passa molto tempo accedendo a zone di memoria ristrette e solo occasionalmente accede a locazioni molto lontane.
- Inoltre, in genere gran parte del codice di cui sono costituiti i programmi viene eseguito solo raramente, al verificarsi di errori o condizioni anomale: quindi succede spesso che di tutto il codice che un programma carica in memoria ne venga realmente eseguita solo una piccola parte.
  - Questo principio è alla base del buon funzionamento della memoria cache e della memoria virtuale: dove tale principio non è rispettato, come per esempio nelle CPU grafiche tridimensionali che devono obbligatoriamente leggere TUTTA la memoria allocata per le texture ad ogni nuovo fotogramma da generare, implementare meccanismi di cache o di memoria virtuale penalizza pesantemente le prestazioni invece di migliorarle.

Virtual Memory

- **Virtual memory**: separazione della memoria logica del programma dalla memoria fisica.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- **Virtual memory** can be implemented via:
  - Demand paging
  - Demand segmentation
  - Demand paged-segmentation



Virtual Memory That is Larger Than Physical Memory

Demand paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - not-in-memory ⇒ bring to memory
- With each page table entry a valid-invalid bit is associated
  - (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid-invalid but is set to 0 on all entries.

Example of a page table snapshot.

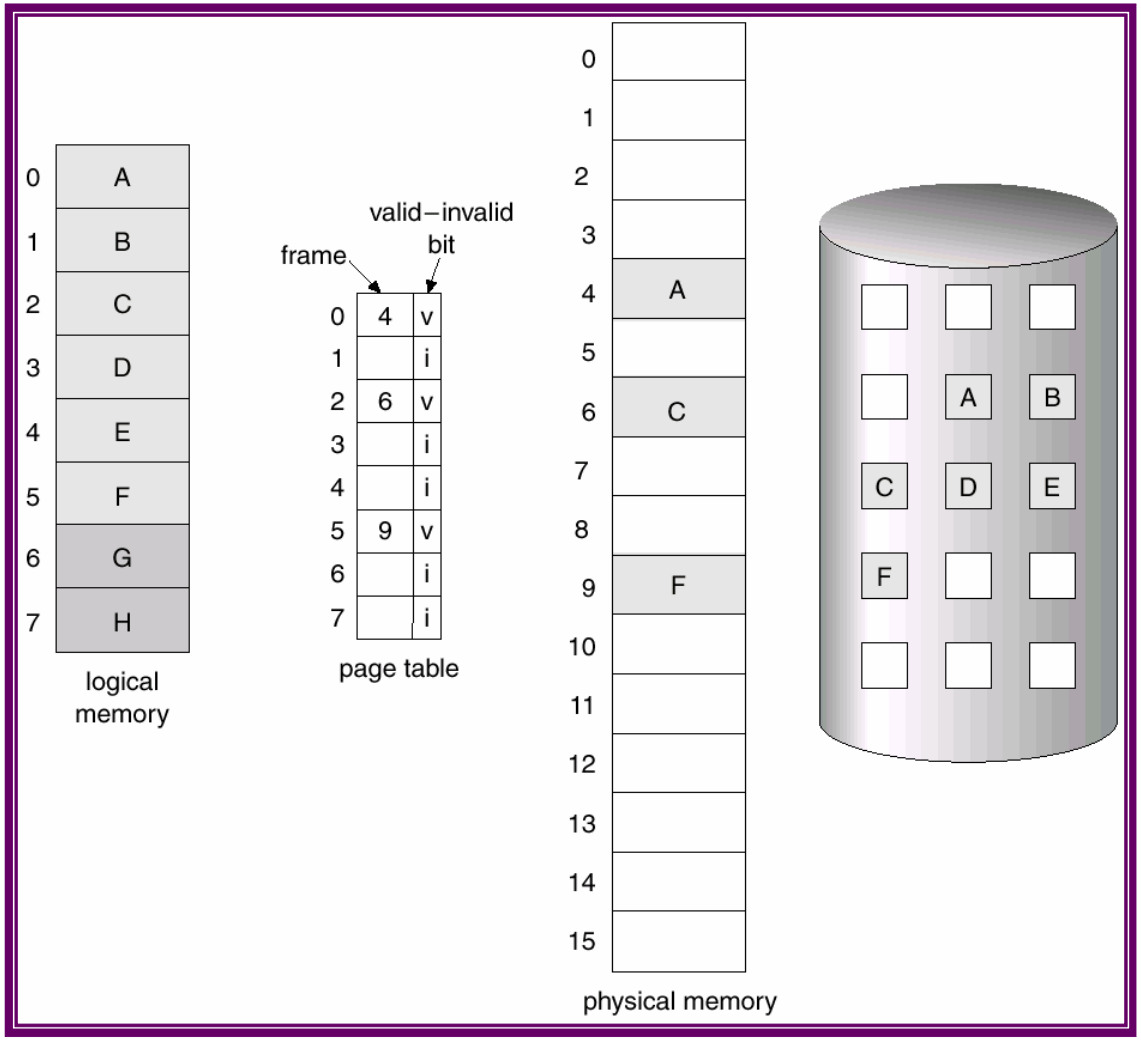
Page #	Invalid Bit
0	1
1	0
2	1
3	0
4	0

Page Table

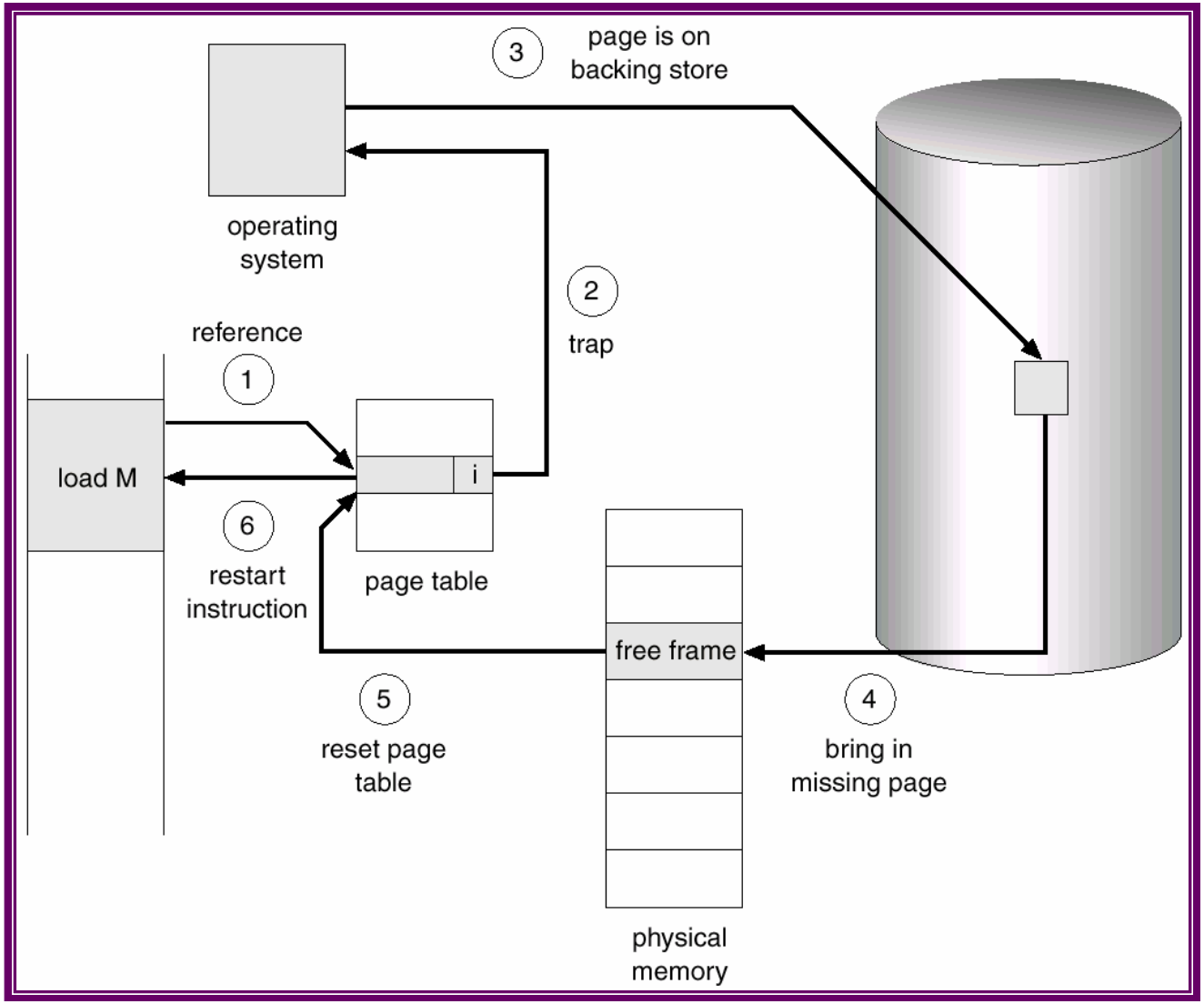
During address translation, if valid–invalid bit in page table entry is 0 ⇒ **page fault**.

Demand paging: the page fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ **page fault**
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used



Demand paging



Steps in Handling a Page Fault

LA STRUTTURA TABELLARE

Task ID	# page	↑ PMT	S bit
63	4	7	0
25	6	9	0
44	12	8	0
50	7	2	1
52	8	6	1
54	6	5	1
51	3	3	1
49	4	1	1
53	5	4	1

Job (Task) Table

B	Task ID	P	C bit	R bit	S bit
0	51	0	0	0	1
1	50	0	W	0	1
2	54	4	0	0	1
3	53	0	Z	1	1
4	49	1	0	1	1
5	52	6	0	0	1
6	52	4	1	Z	1
7	49	3	W	Z	1
8	50	4	Z	0	1
9	54	3	Z	W	1
10	52	0	1	1	1
11	50	6	1	W	1
12	54	1	Z	Z	1
13	53	4	W	W	1

Memory Block Table

#	Task ID	P	C bit	C T S	S bit
0	51	1	0	13 4 10	1
1	51	2	0	99 20 5	1
2	53	1	0	22 12 10	1
3	49	0	1	4 6 18	1
4	53	2	0	14 18 25	1
5	49	2	1	105 21 5	1
6	53	3	1	63 3 17	1
7	52	2	0	21 13 7	1
8	52	5	0	55 6 7	1
9	50	1	1	45 11 9	1
10	54	2	1	17 17 17	1
11	52	3	1	88 25 10	1
12	52	7	0	199 6 13	1
13	50	2	1	33 20 15	1
14	54	5	1	166 11 2	1
15	52	1	0	167 12 1	1
16	54	0	0	68 11 12	1
17	50	3	1	77 13 15	1
18	50	5	0	63 24 12	1

EPMT

P	I bit	↑ EPMT	B
0	0	3	23
1	1	19	4
2	0	5	31
3	1	21	7

PMT 1

P	I bit	↑ EPMT	B
0	1	20	1
1	0	9	25
2	0	13	24
3	0	17	32
4	1	22	8
5	0	18	27
6	1	**32**	11

PMT 2

P	I bit	↑ EPMT	B
0	1	24	0
1	0	0	15
2	0	1	14

PMT 3

P	I bit	↑ EPMT	B
0	1	23	3
1	0	2	19
2	0	4	17
3	0	6	20
4	1	25	13

PMT 4

P	I bit	↑ EPMT	B
0	0	16	18
1	1	30	12
2	0	10	15
3	1	29	9
4	1	28	2
5	0	14	16

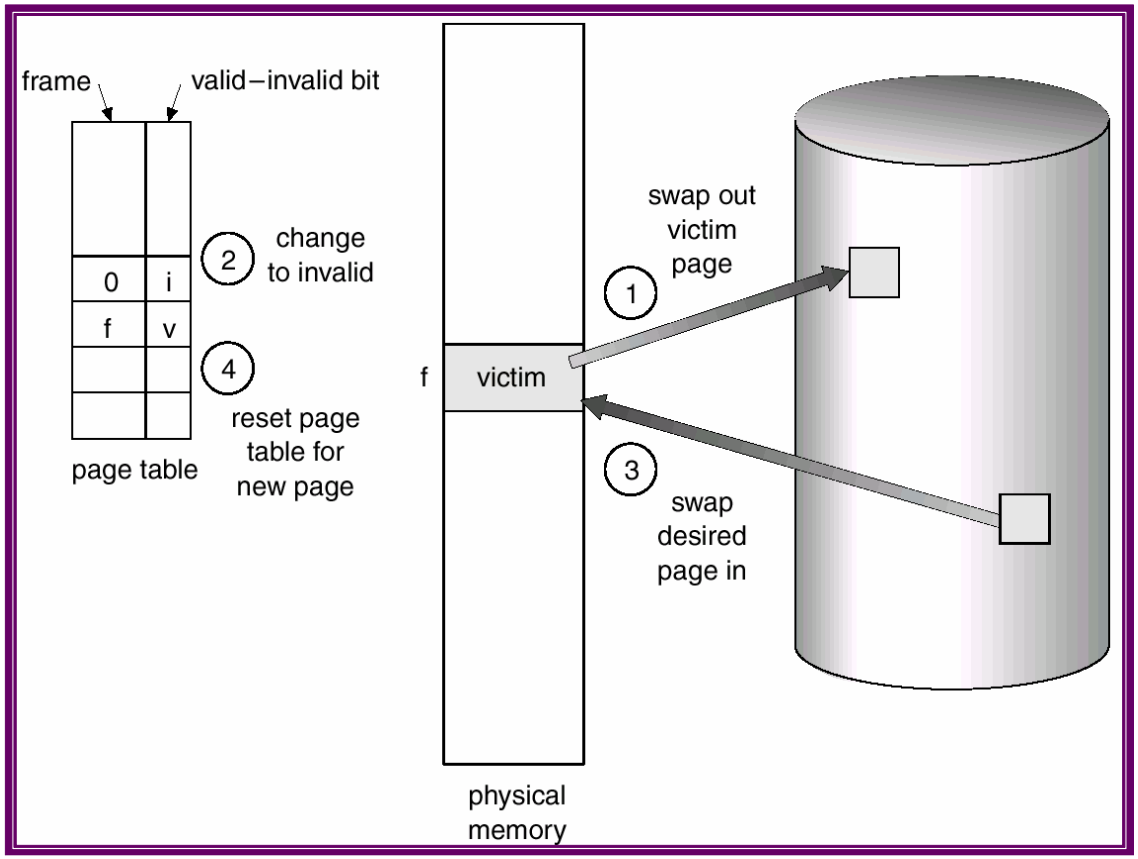
PMT 5

P	I bit	↑ EPMT	B
0	1	27	10
1	0	15	23
2	0	7	21
3	0	11	22
4	1	26	6
5	0	8	26
6	1	31	5
7	0	12	28

PMT 6

Demand paging

- What happens if there is no free frame?
- ✓ **Page replacement** – find some page in memory, but not really in use, swap it out.
  - algorithm: performance – want an algorithm which will result in minimum number of page faults.
- Two bits may be useful for arranging effective page replacement: Reference bit (R bit) and Change bit (C bit or dirty bit).
- Same page may be brought into memory several times.
- Change bit (dirty bit) to reduce overhead of page transfers – only modified pages are written to disk.



Demand paging

Page Replacement algorithms

- **First-In-First-Out (FIFO) Algorithm**
  - Every page entry has a counter; every time page is loaded into the memory through this entry, copy the clock into the counter.
- **Least Recently Used (LRU) Algorithm**
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
- **LRU Approximation Algorithm (algoritmo della seconda possibilità migliorato)**
  - Reference bit & (Change bit)
    - With each page associate a bit, initially = 0
    - When page is referenced bit set to 1.
    - Replace the one which is 0 (if one exists). We do not know the order, however.

Demand segmentation

Demand paged-segmentation

- La gestione dei segmenti (o quella dei segmenti e delle pagine) è basata sugli stessi criteri alla base della segmentazione e della segmentazione e paginazione reale

Multiple Virtual Storage (MVS)

- Ad ogni programma è associata un'intera memoria virtuale ed il SO cambia la memoria virtuale di riferimento quando cambia il contesto computazionale della CPU.