

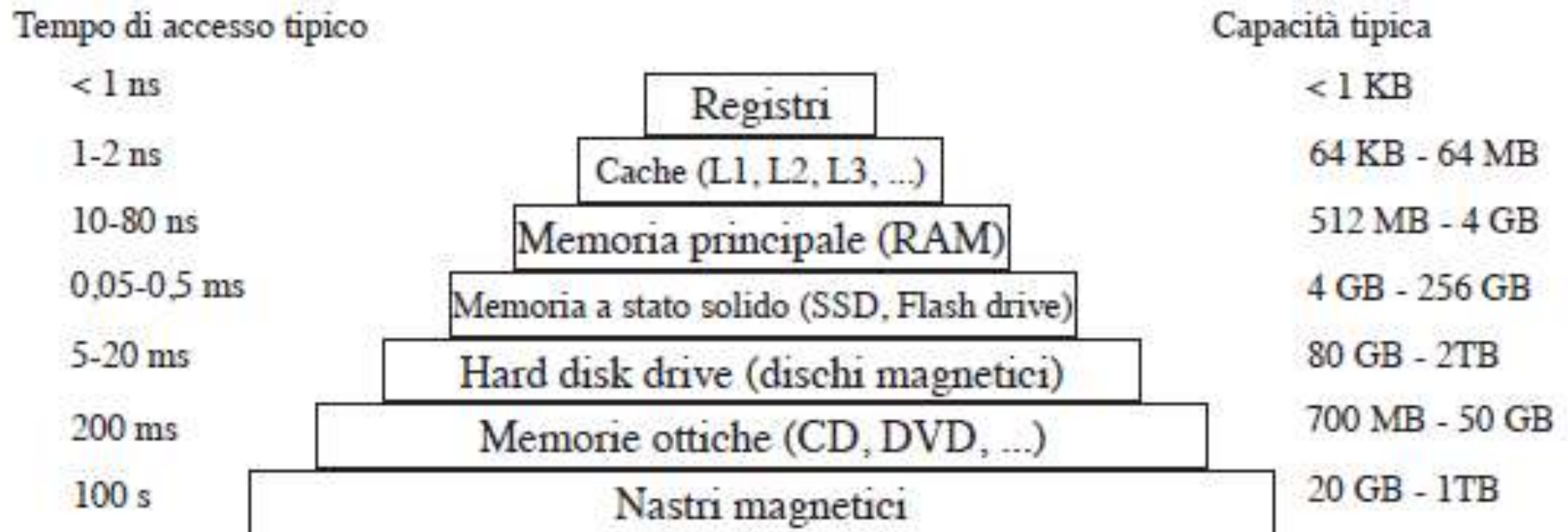
LA MEMORIA CENTRALE

La memoria è una **risorsa importante, e limitata**.

“I programmi sono come i gas reali: si espandono fino a riempire tutta la memoria disponibile”.

Memoria illimitata, infinitamente veloce, economica: non esiste.

Esiste la **gerarchia delle memorie**, gestita dal gestore della memoria (*memory manager*).



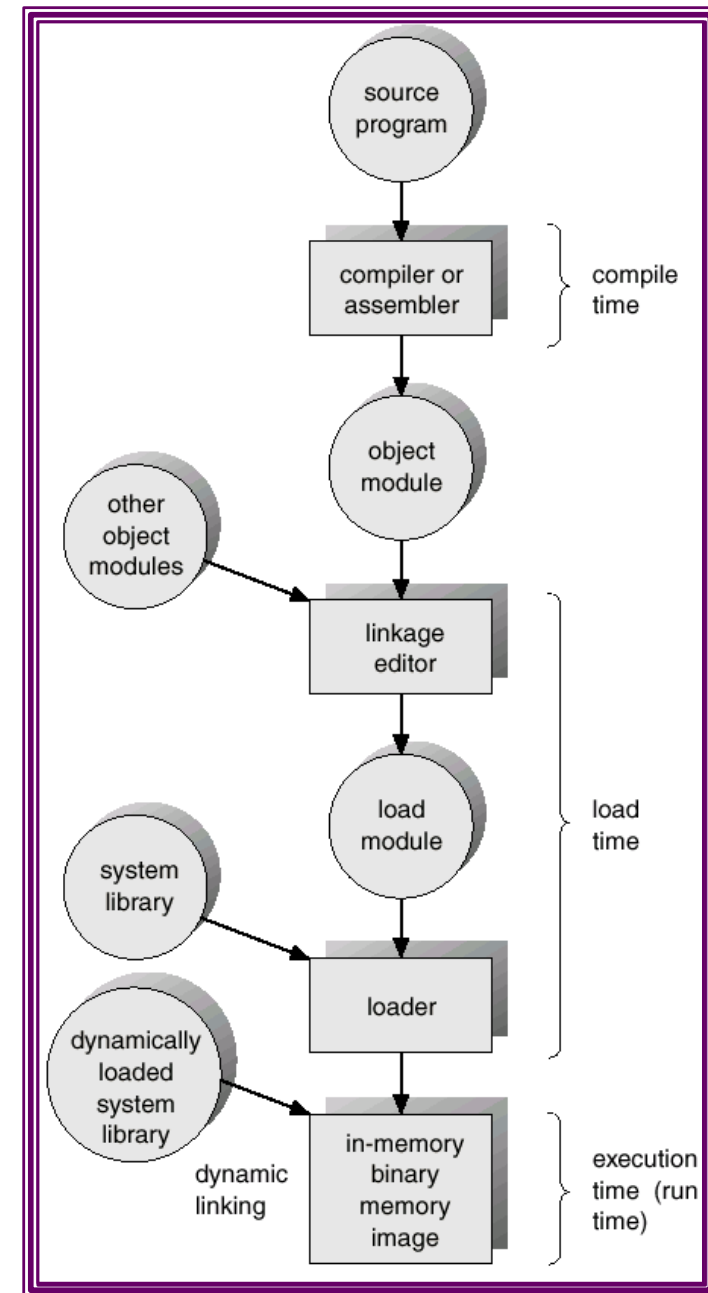
GESTIONE DELLA MEMORIA: FONDAMENTI

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

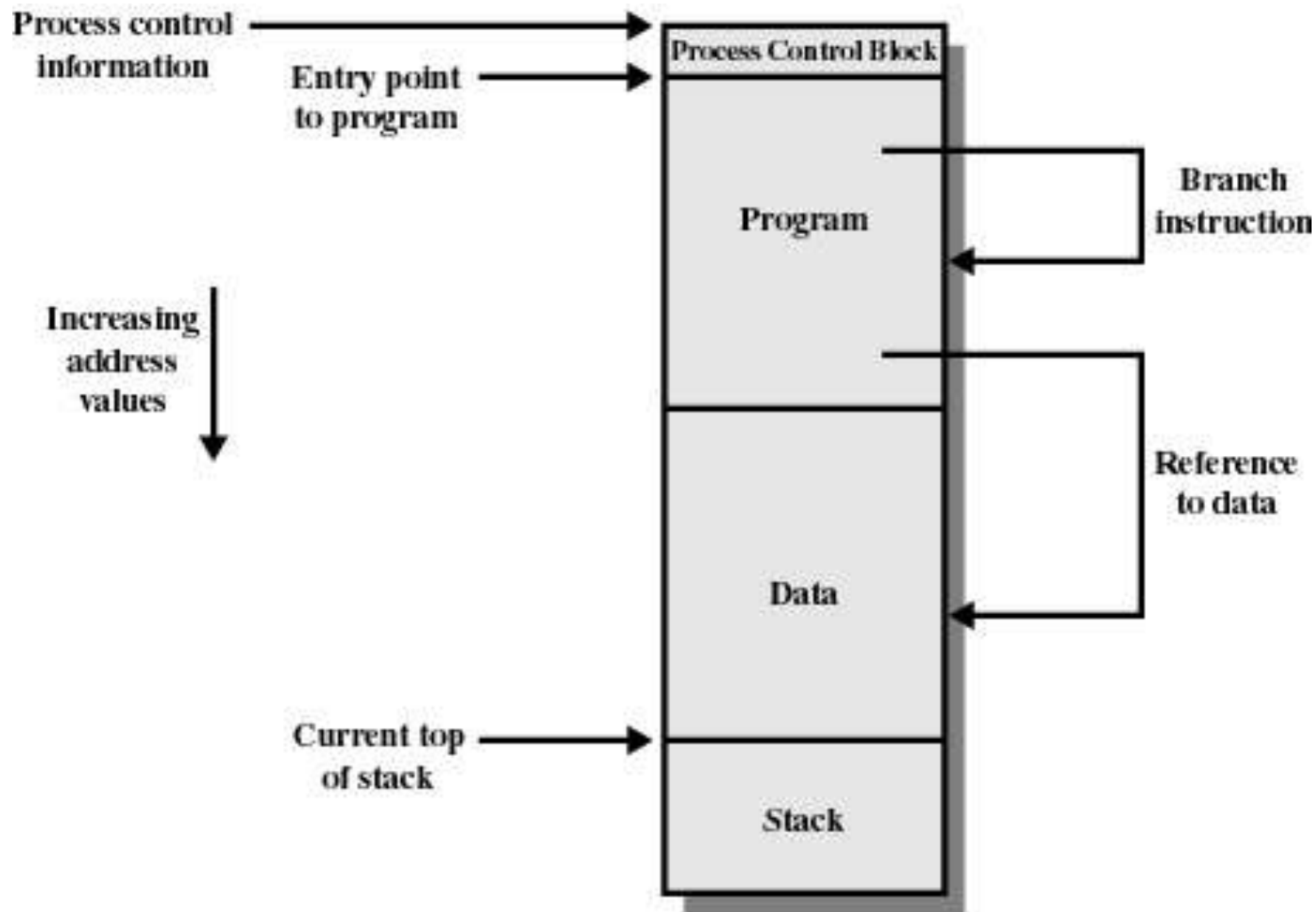
- + **Organizzazione logica:** offrire una visione astratta della gerarchia della memoria, ovvero, *allocare e deallocare* memoria ai processi su richiesta.
- + **Organizzazione fisica:** *tener conto a chi è allocato cosa*, e effettuare gli scambi con il disco.
- + **Rilocazione.**
- + **Protezione:** tra i processi, e per il sistema operativo.
- + **Condivisione:** aumentare l'efficienza.

FROM THE SOURCE PROGRAM TO THE EXECUTABLE PROGRAM

- User programs go through several steps before being run.
 - ✓ Compilation or assembly
 - ✓ Linkage editing
 - ✓ Loading



ADDRESS SPACE STRUCTURE



Processes have three segments: text, data, and stack.
All three are in one address space, but separate instruction and data space is also supported.

PROGRAM ADDRESSING AND LOADING

Generalia

- Address binding of instructions and data to memory addresses can happen at three different stages.
 - ↳ **Compile time:** If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.
 - ↳ **Load time:** Must generate *relocatable* code at loading time if memory location is not known at compile time. Cannot be changed during execution.
 - ↳ **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).
- **Compiler** and relocatable addresses generation.
- **Linker** and individually compiled modules linking; transformation of external variables addresses from symbolic to relocatable.
- The program in executable format: the relocatable format, the size, the entry point.
- **Loader** and *static or dynamic relocation*.

PROGRAM ADDRESSING AND LOADING

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

Indirizzo logico: generato dalla CPU. Detto anche **indirizzo virtuale**.

Indirizzo fisico: indirizzo **visto dalla memoria**.

Indirizzi logici e fisici **coincidono** nel caso di binding al compile time o load time.

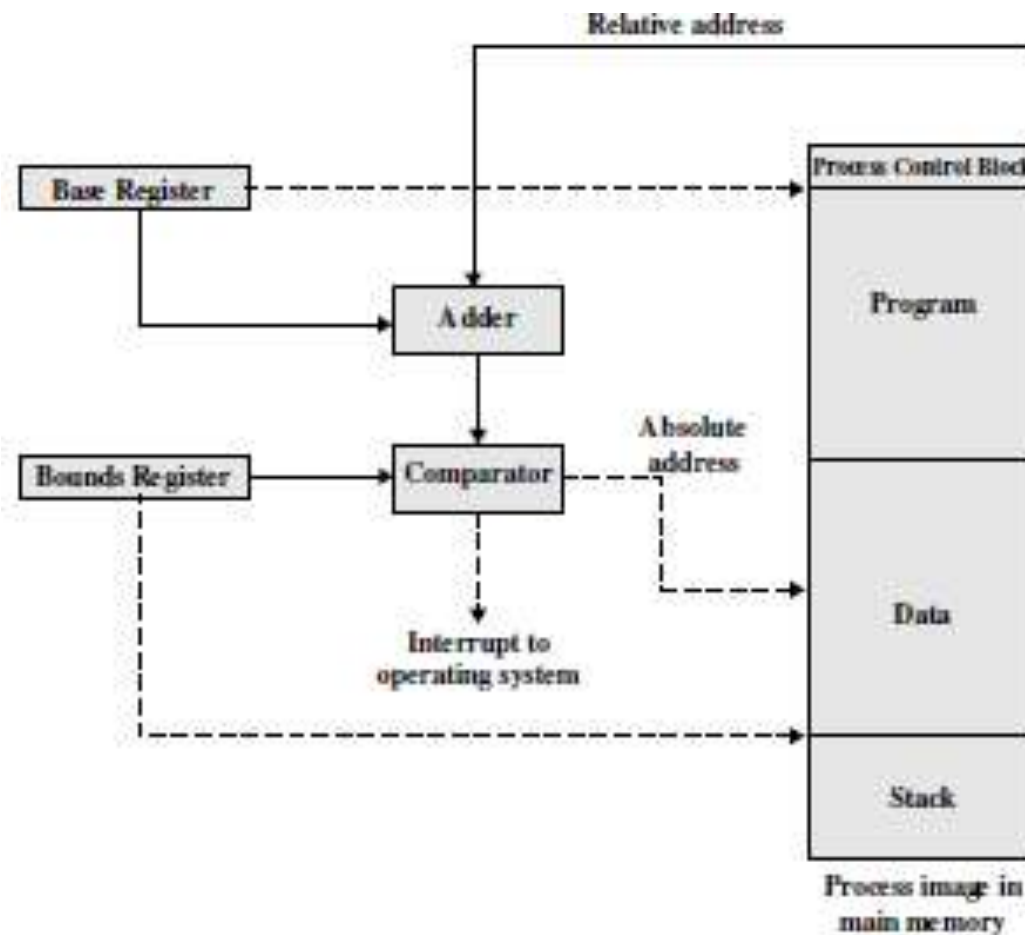
Possono essere **differenti** nel caso di binding al tempo di esecuzione. Necessita di un hardware di traduzione.

RELOCATABLE ADDRESSES OF A PROGRAM

Memory Management Unit (MMU)

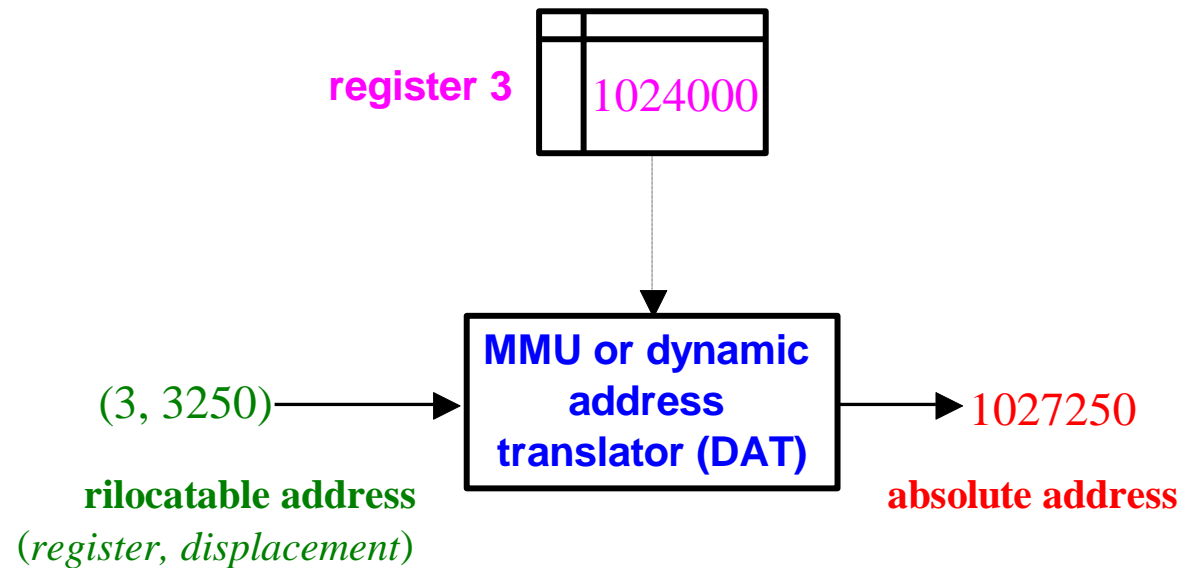
- Hardware device that maps virtual to physical address.
- In MMU basic scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

The user program deals with *logical* addresses; it never sees the *real* physical addresses.



RELOCATABLE ADDRESSES OF A PROGRAM

Memory Management Unit (MMU)



DINAMIC LOADING INTO THE MEMORY

- Any code segment (f.e. a routine) is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases (f.e. errors).
- No special support from the operating system is required. It can be implemented through program design.
- The OS can offer libraries to make it easy dynamic loading.

DINAMIC LINKING

- ↳ Linking postponed until execution time.
- ↳ Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.
- ↳ At the first execution the stub replaces itself with the address of the routine (which is loaded) and the control is transferred to it.
- ↳ Operating system needed to check if routine is in processes' memory address. It has also to give support for code sharing among several processes.
- ↳ Dynamic linking is particularly useful for libraries updating.

MEMORY MANAGEMENT ALGORITHMS

Single-programming

Φ *Contiguous memory allocation*

Multi-programming

↳ *Memory partitioning*

- Φ *static partitioning,*
- Φ *dynamic partitioning,*
- Φ *relocatable partitioning,*
- Φ *multiple partitioning,*
- Φ *swapping,*
- Φ *rolling*

↳ *Real memory paging (segmentation)*

- Φ *paging*
- Φ *segmentation*
- Φ *paging with segmentation*

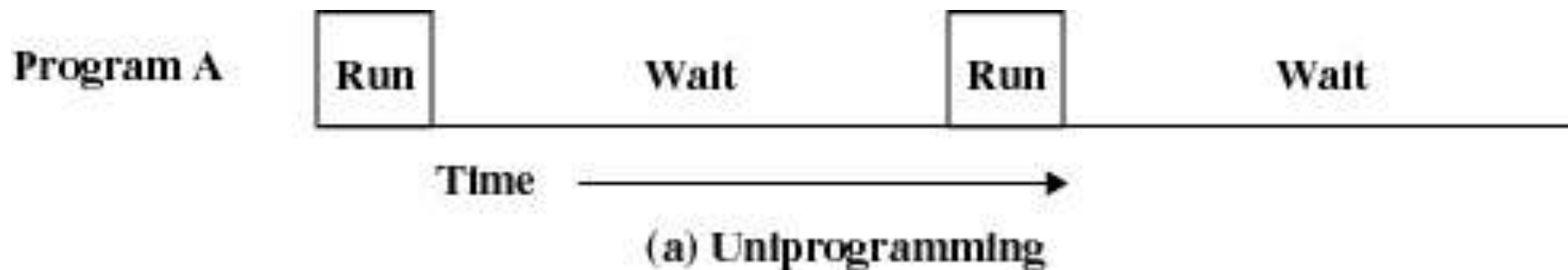
↳ *Virtual storage paging (segmentation)*

- Φ *paged virtual storage (demand paging)*
- Φ *segmented virtual storage*
- Φ *Segmented and paged virtual storage*
- Φ *Multiple virtual storage*

SINGLE-PROGRAMMING

Contiguous memory allocation

- Main memory usually divided into *two partitions*:
 - ⇒ Resident operating system, usually held in low memory with interrupt vector.
 - ⇒ User process then held in high memory.



SINGLE-PROGRAMMING

Contiguous memory allocation

- **Single-partition allocation**

- ↳ *Relocation-register scheme* used to protect user processes from each other, and from changing operating-system code and data.
- ↳ Relocation register contains value of smallest physical address of the process; *limit register* contains range of logical addresses – each logical address must be less than the limit register.
- ↳ Questi registri sono contenuti nella MMU e vengono caricati dal kernel ad ogni context-switch.

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:

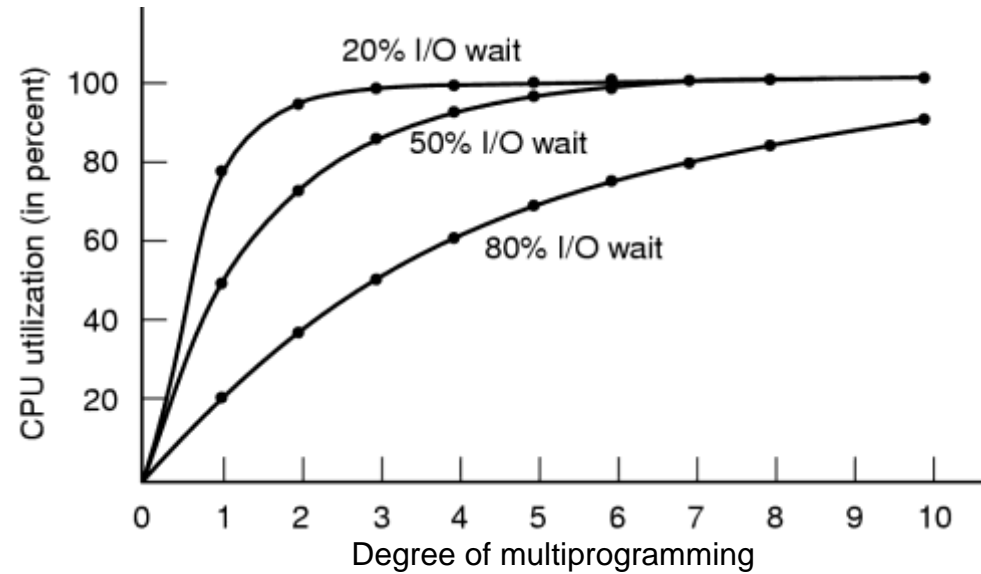
utilizzo CPU = $1 - p^n$

Maggiore il grado di multiprogrammazione, maggiore l'utilizzo della CPU.

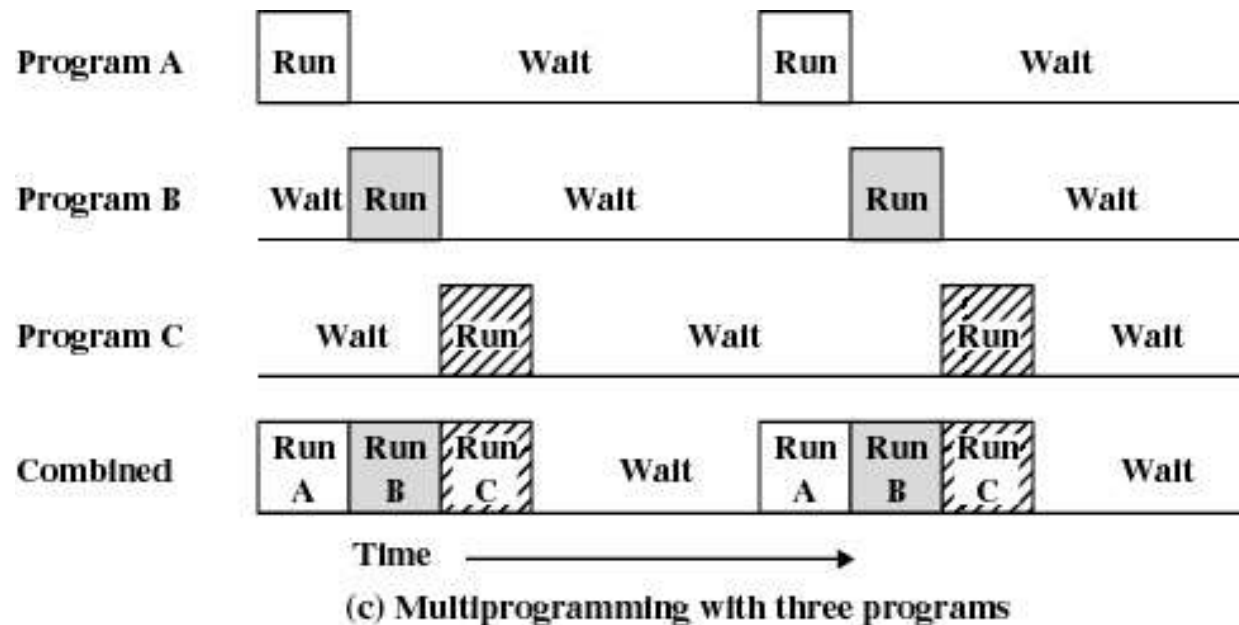
Il modello è ancora impreciso (i processi non sono indipendenti); un modello più accurato si basa sulla teoria delle code.

MULTI-PROGRAMMING

Pro



- When one job needs to wait for I/O, the processor can switch to the other job



O.S. FEATURES FOR SUPPORTING MULTI-PROGRAMMING

⇒ I/O routine supplied by the system.

⇒ Memory management – the system must allocate the memory to several jobs.

⇒ CPU scheduling – the system must choose among several jobs ready to run.

⇒ Allocation of devices.

⇒ Protection

- Processes should not be able to reference memory locations in another process without permission
- Impossible to check absolute addresses in programs since the program could be relocated: must be checked during execution
- Operating system cannot anticipate all of the memory references a program will make

⇒ Sharing

- Allow several processes to access the same portion of memory
- Better to allow each process (person) access to the same copy of the program rather than have their own separate copy

O.S. FEATURES FOR SUPPORTING MULTI-PROGRAMMING

continue

↳ Logical Organization

- Programs are written in modules
- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules

↳ Physical Organization

- Memory available for a program plus its data may be insufficient
- Overlaying allows various modules to be assigned the same region of memory
- Programmer does not know how much space will be available

STATIC PARTITIONING

La memoria disponibile è divisa in partizioni fisse (uguali o diverse).

Il sistema operativo mantiene informazioni sulle partizioni allocate e quelle libere.

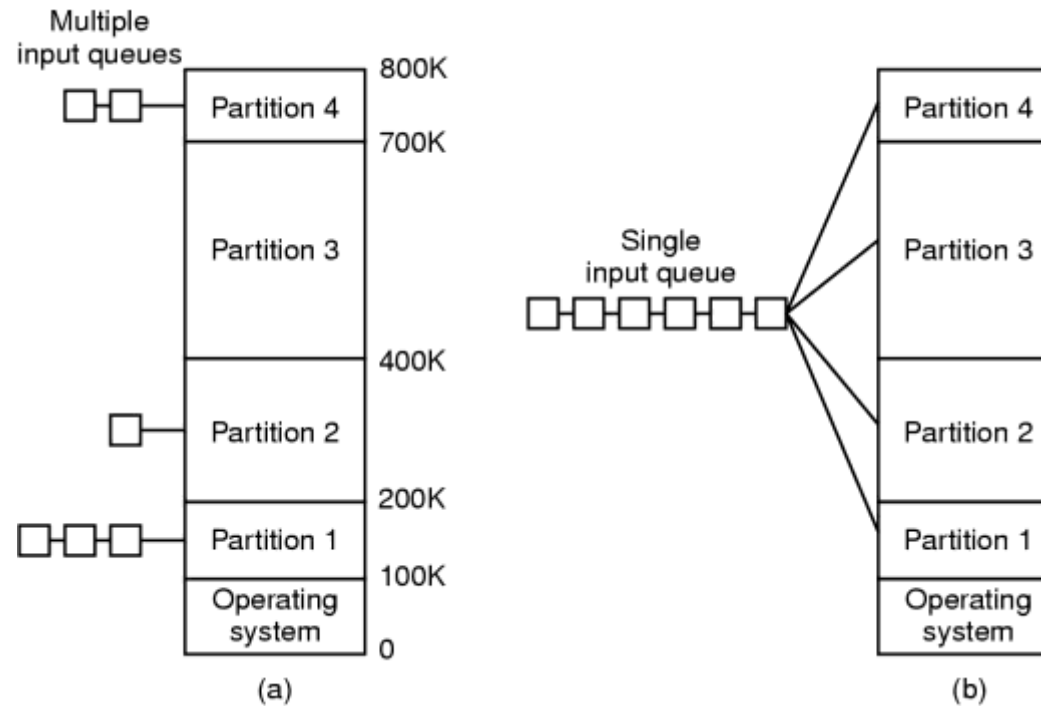
| Partition numb. | Progr. Id. | Dimension | First byte | Status bit |
|-----------------|------------|-----------|------------|------------|
| 1 | Alfa | 100k | 100k | 1 |
| 2 | | 200k | 200k | 0 |
| 3 | Word | 300k | 400k | 1 |
| 4 | Beta | 100k | 700k | 0 |

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.

STATIC PARTITIONING



Una coda per ogni partizione: possibilità di inutilizzo di memoria.

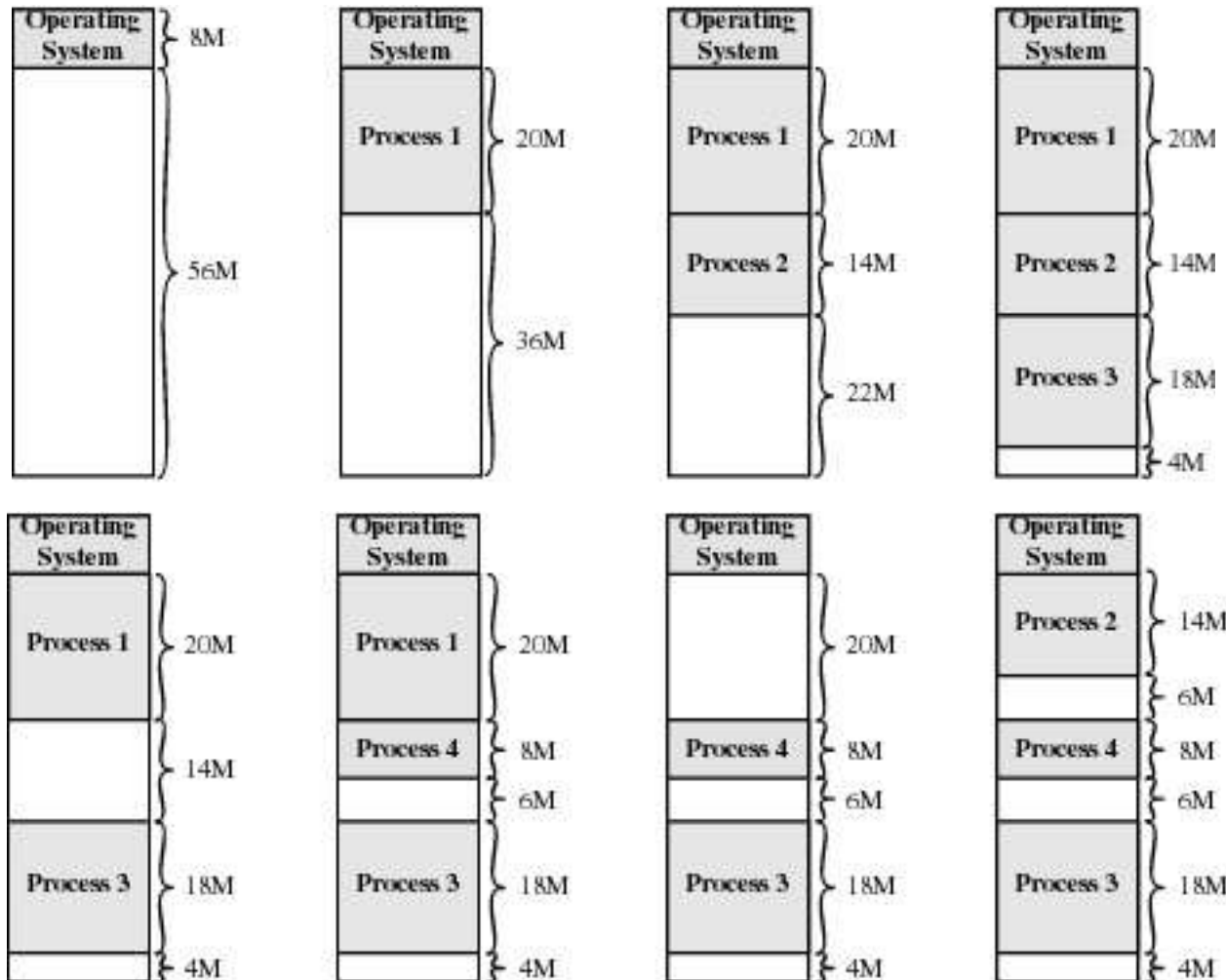
Una coda per tutte le partizioni: come scegliere il job da allocare?

first-fit: per ogni buco, il primo che ci entra,

best-fit: il più grande che ci entra. Penalizza i job piccoli (che magari sono interattivi. . .).

DINAMIC PARTITIONING

Multiple-partition allocation



- *Hole* – block of available memory; free spaces of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Operating system maintains information about:
 - a) allocated partitions
 - b) free space (hole)

DINAMIC PARTITIONING

Allocated Partitions Table

| 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 |

Free Space Table

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

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.

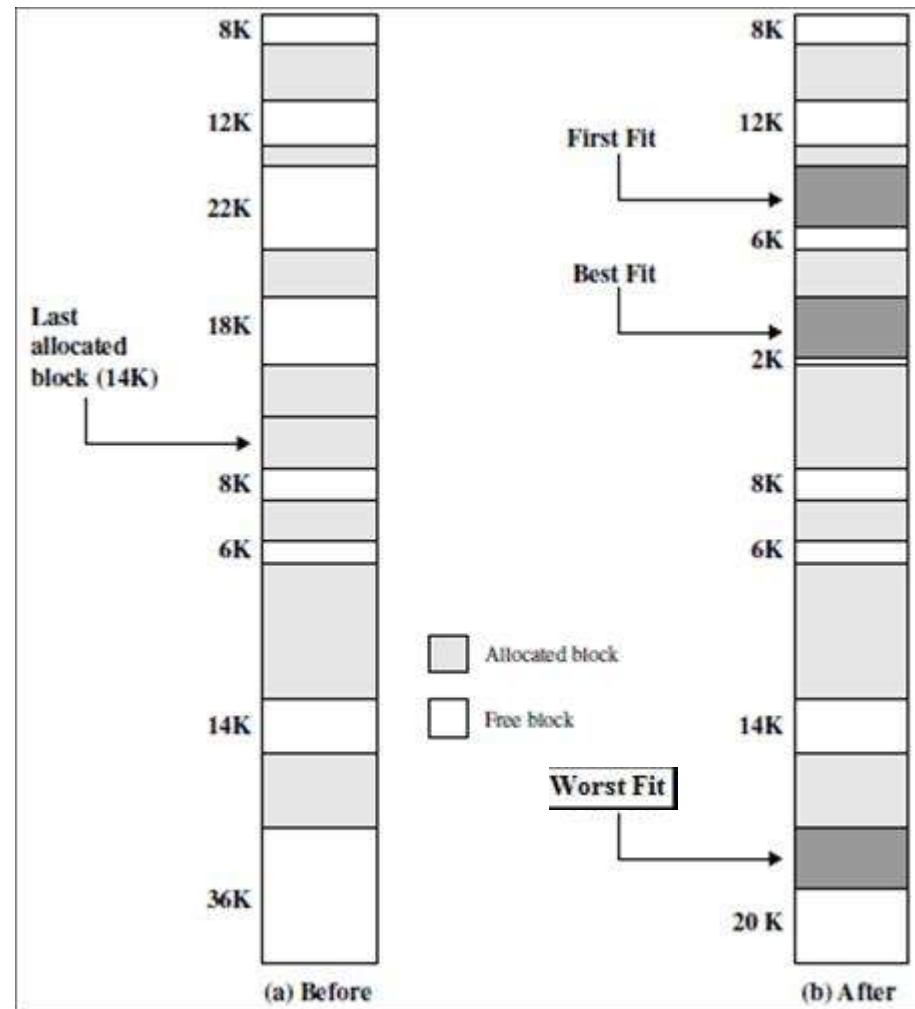
DINAMIC PARTITIONING

➤ According to the sorting method of the Free Space Table, the memory allocation strategy may be:

- **First-fit:** The Free Space Table is sorted according to the Hole First Byte:
 - ✓ allocate the *first* hole that is big enough.
 - ✓ fastest
 - ✓ may have many process loaded in the front end of memory that must be searched over when trying to find a free block
- **Best-fit:** The Hole Table is sorted according to the Hole Dimension:
 - ✓ allocate the *smallest* hole that is big enough;
 - ✓ must search entire list, unless ordered by size;
 - ✓ produces the smallest leftover hole,
 - ✓ since smallest block is found for process, the smallest amount of fragmentation is left,
 - ✓ memory compaction must be done more often,
 - ✓ worst performer overall
- **Worst-fit:** Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

DINAMIC PARTITIONING

Si suppone di dover allocare 16 K



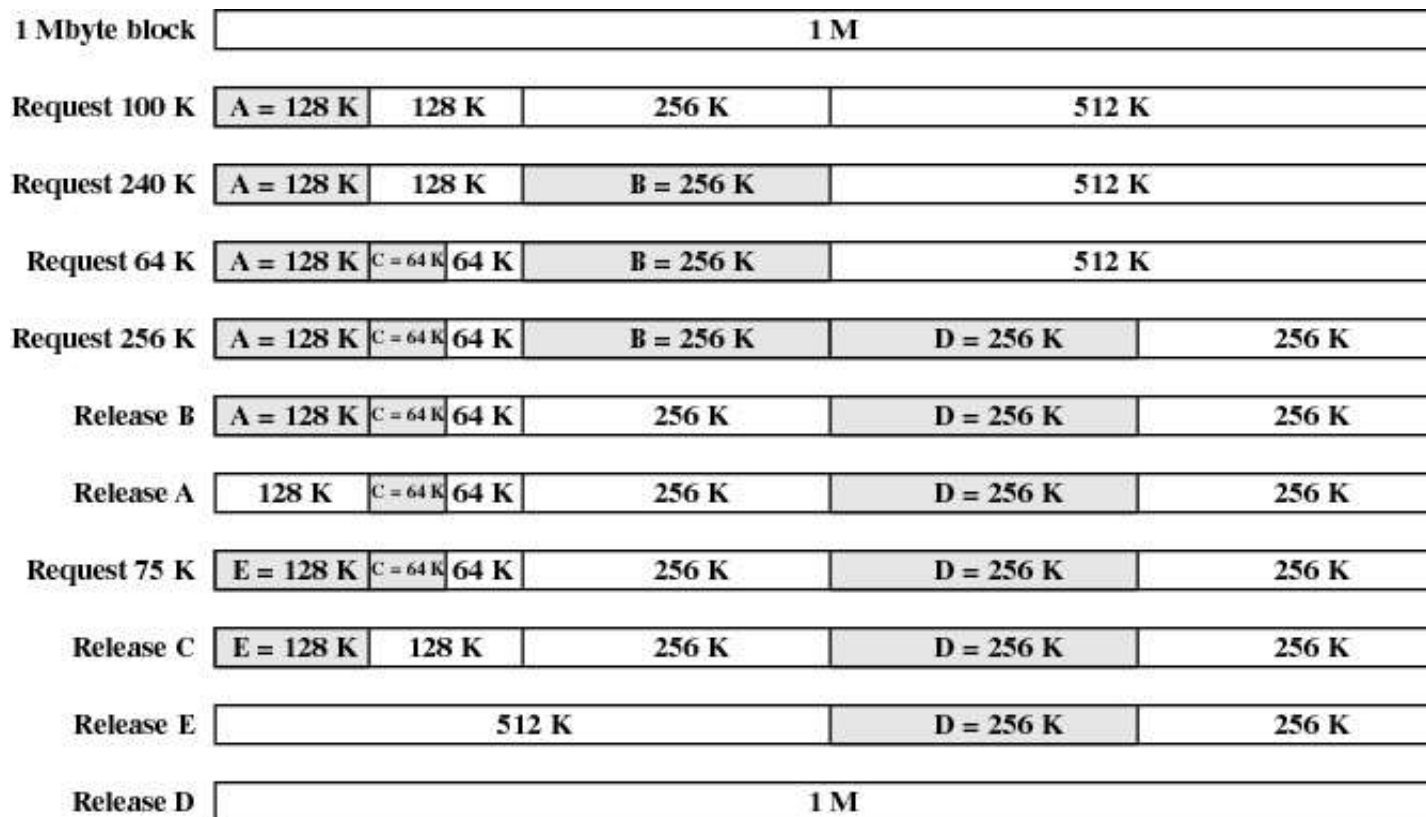
In generale, l'algoritmo migliore è il first-fit.

Best-fit tende a frammentare molto. Worst-fit è più lento.

DINAMIC PARTITIONING

Instead of Allocated Partition Table and Free Space Table, the **Buddy System approach** can be used.

- Entire space available is treated as a single block of 2U
- If a request of size s such that $2^{U-1} < s \leq 2^U$, entire block is allocated
 - ↪ Otherwise block is split into two equal buddies
 - ↪ Process continues until smallest block greater than or equal to s is generated



RILOCATABLE PARTITIONING

↪ use compaction to shift processes so they are contiguous and all free memory is in one block

↪ reduce external fragmentation by compaction

- Shuffle memory contents to place all free memory together in one large block.
- Compaction is possible *only* if relocation is dynamic, and is done at execution time.
- I/O problem
 - ☞ Latch job in memory while it is involved in I/O.
 - ☞ Do I/O only into OS buffers.

MULTIPLE PARTITIONING

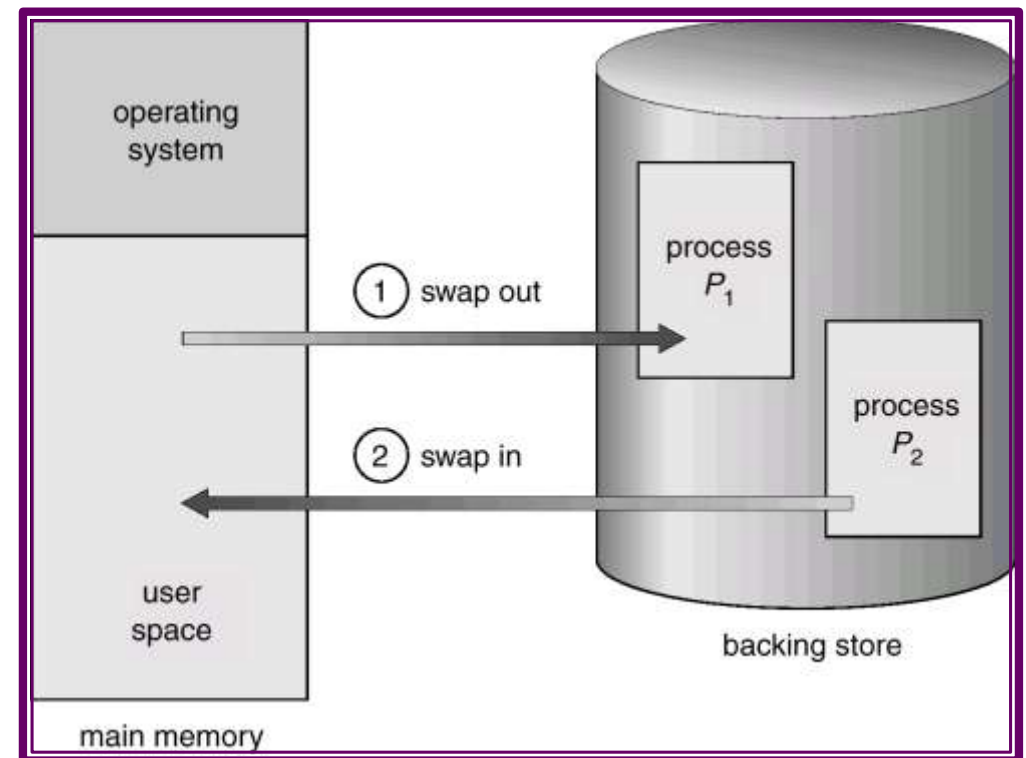
Si basa sull'assunzione che lo spazio degli indirizzi di un programma non debba essere tutto contiguo in memoria centrale.

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

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- *Roll out, roll in* – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.
- Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.



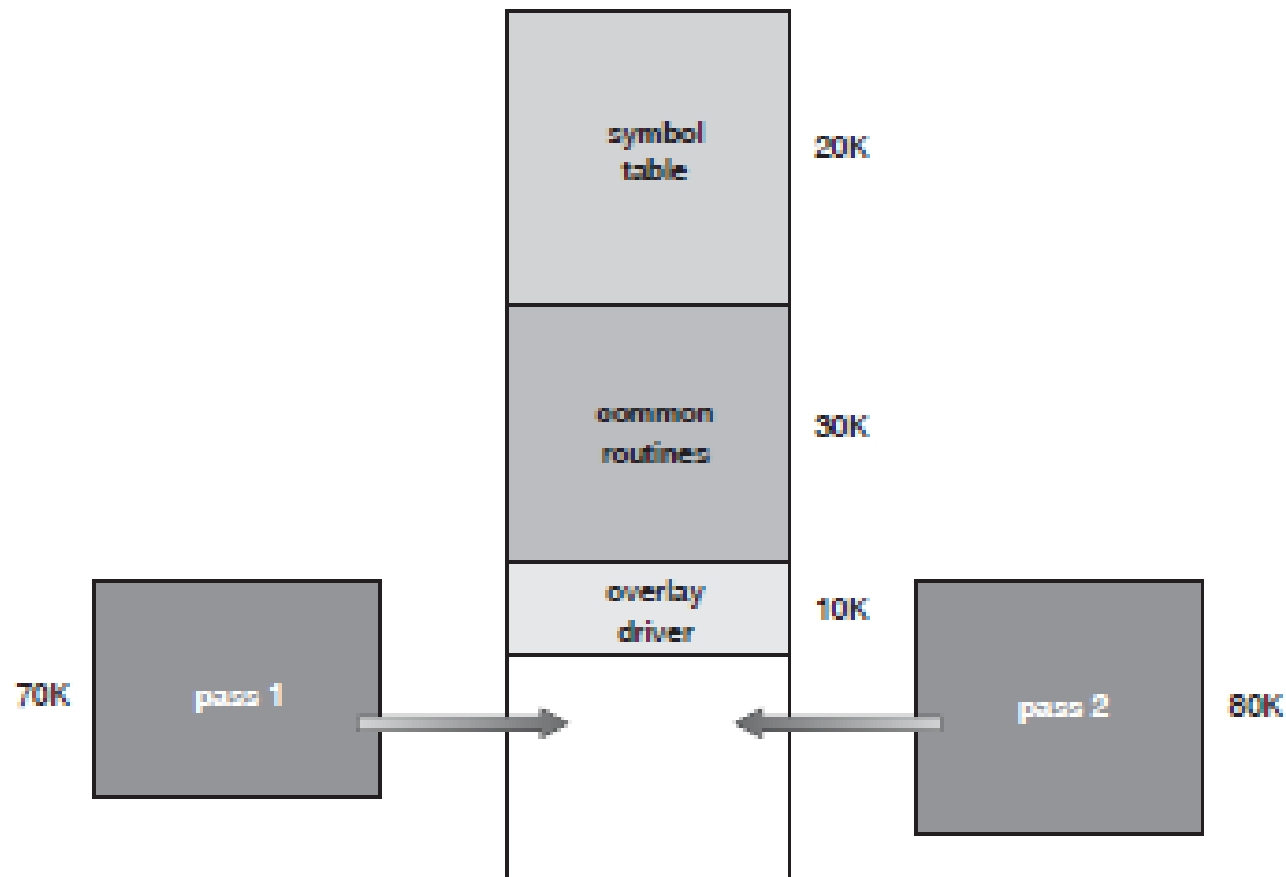
OVERLAY

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

- ↳ *Logical address space of a process can be noncontiguous*; process is allocated in the physical memory whenever the latter is available.
- ↳ *Divide physical memory into fixed-sized blocks* called **frames** (size is power of 2, between 512 bytes and 8192 bytes or more).
- ↳ *Divide logical memory into blocks of same size* called **pages**.
- ↳ *Keep track of all free frames*: to run a program of size n pages, need to find n free frames and load program → The Memory Block Table (MBT)
- ↳ *No external fragmentation. Little internal fragmentation.*
- ↳ Operating system maintains *a page table for each process*
 - contains the frame location for each page in the process
 - memory address consist of a page number and offset within the page

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

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:

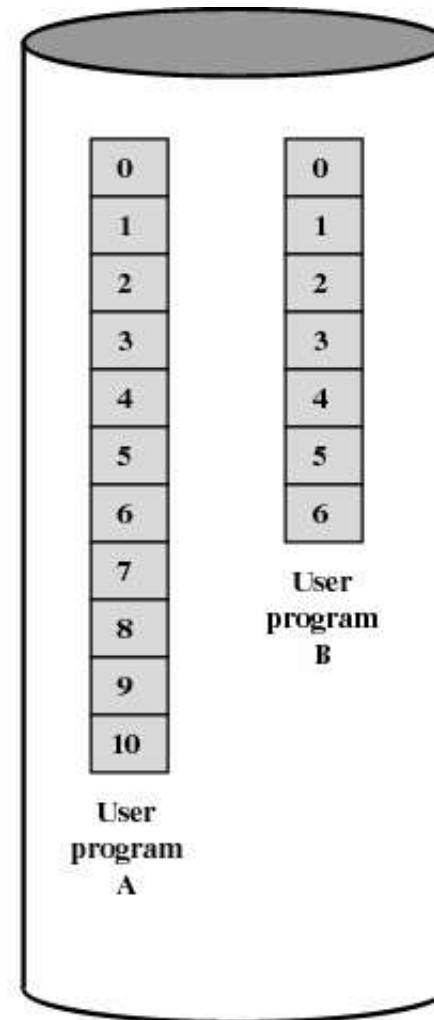
- “**valid**” = indica che la pagina associata è nello **spazio logico del processo**, e quindi è legale accedervi
- “**invalid**” = indica che la pagina non è nello spazio logico del processo) violazione di indirizzi (**Segment violation**)

PAGING (VIRTUAL MEMORY CONCEPT)

| | | | |
|-----|-----|-----|-----|
| A.1 | | | |
| | A.0 | A.2 | |
| | A.5 | | |
| | | | |
| B.0 | B.1 | B.2 | B.3 |
| | | | |
| | | | |
| | | | |
| | | A.7 | |
| | A.9 | | |
| | | | |
| | | A.8 | |
| | | | |
| | | | |
| | | | |
| B.4 | B.5 | B.6 | |
| | | | |

Main Memory

Main memory consists of a number of fixed-length frames, equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.



Disk

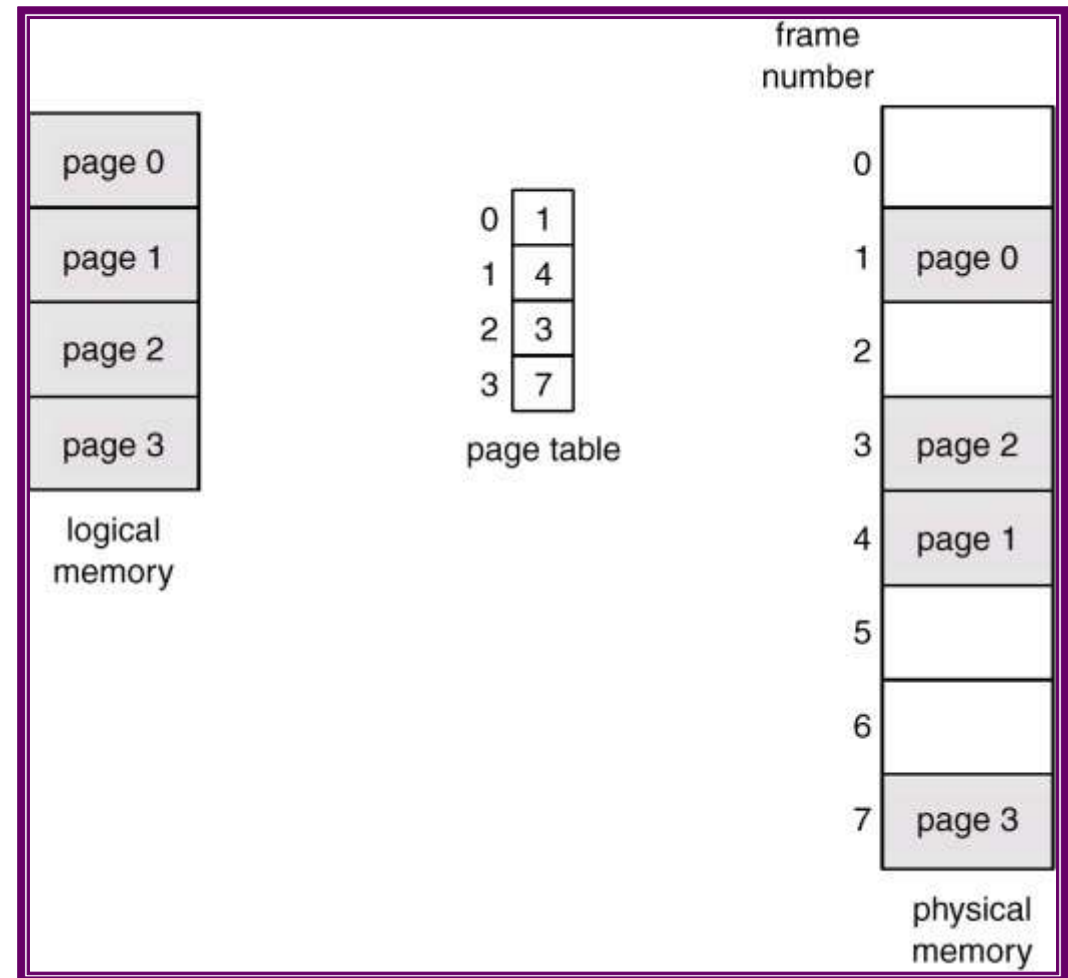
Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

PAGING

➤ Address generated by CPU – (p, d) – is divided into:

- **Page number (p)** – used as an index into a *page table* which contains base address of each page in physical memory.
- **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit.

Tramite la Page Table di ciascun programma il SO tiene traccia del blocco di memoria in cui le varie pagine sono state caricate



PAGING

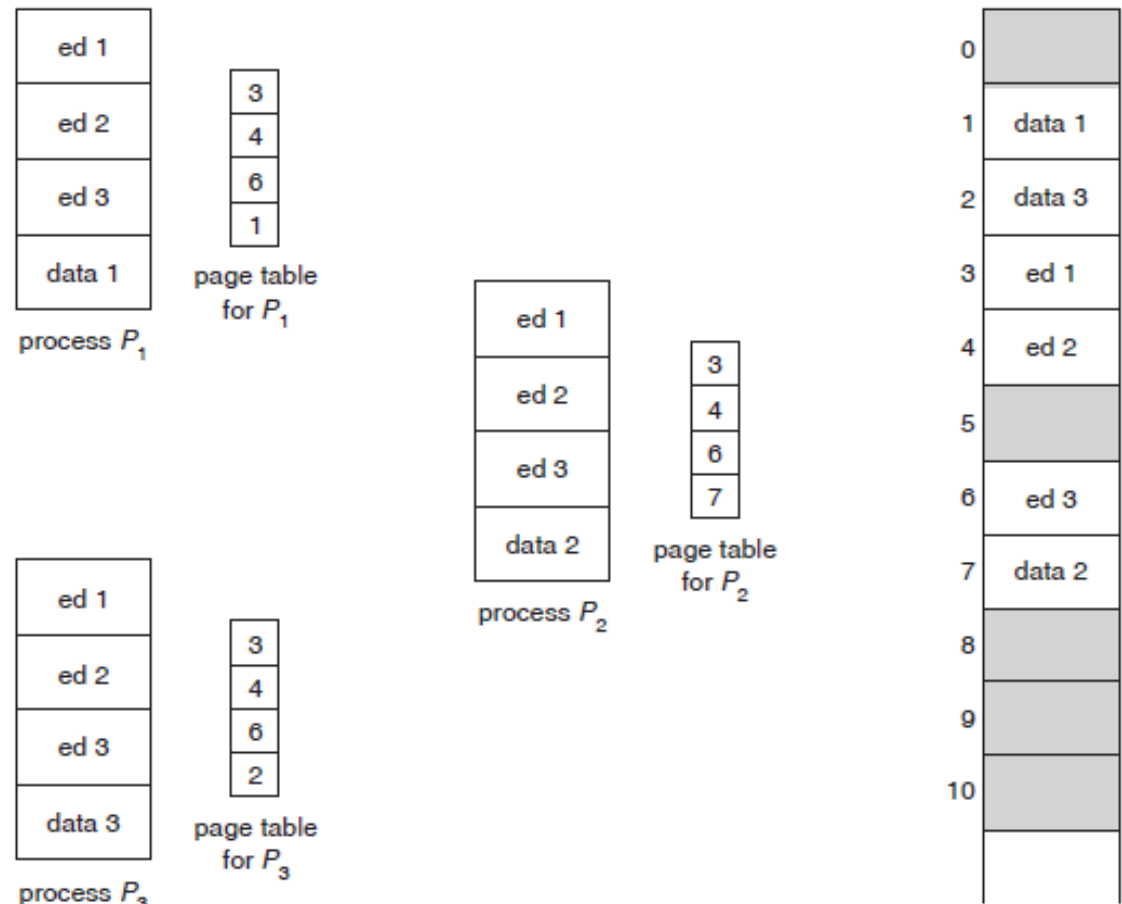
La paginazione permette la **condivisione del codice**.

Una sola copia di codice read-only può essere condivisa tra più processi.

Il codice deve essere rientrante (separare codice eseguibile da record di attivazione). Es.: editor, shell, compilatori, . . .

Il codice condiviso appare nelle stesse locazioni fisiche per tutti i processi che vi accedono.

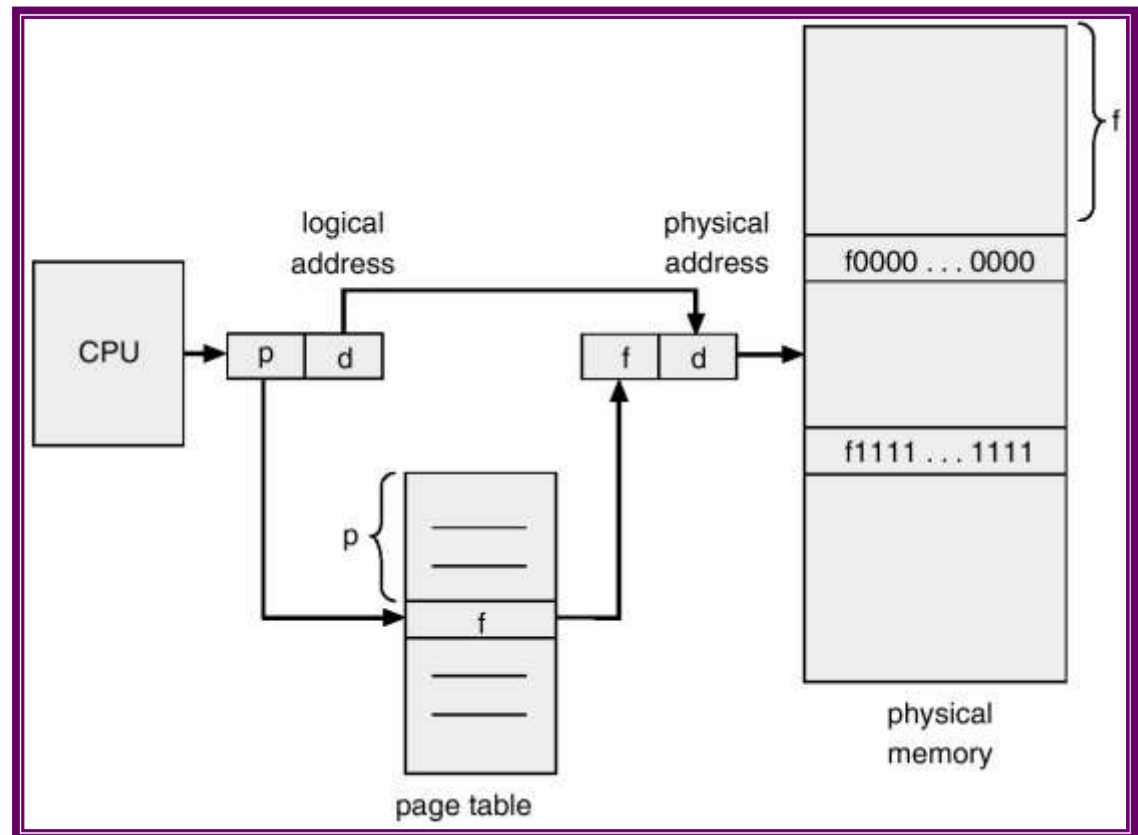
Ogni processo mantiene una copia separata dei propri dati.



PAGING

*The translation from logical address (in the program address space)
to physical absolute addressing (in the main memory)*

- Page table is kept in main memory.
- **Page-Table Base Register (PTBR)** points to the page table.
- **Page-Table Length Register (PRLR)** indicates size of the page table.
- In this scheme every data/instruction access requires **two memory accesses**. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **Translation Look-aside Buffers (TLBs)**
- *The address translation though associative memory is made in parallel.*

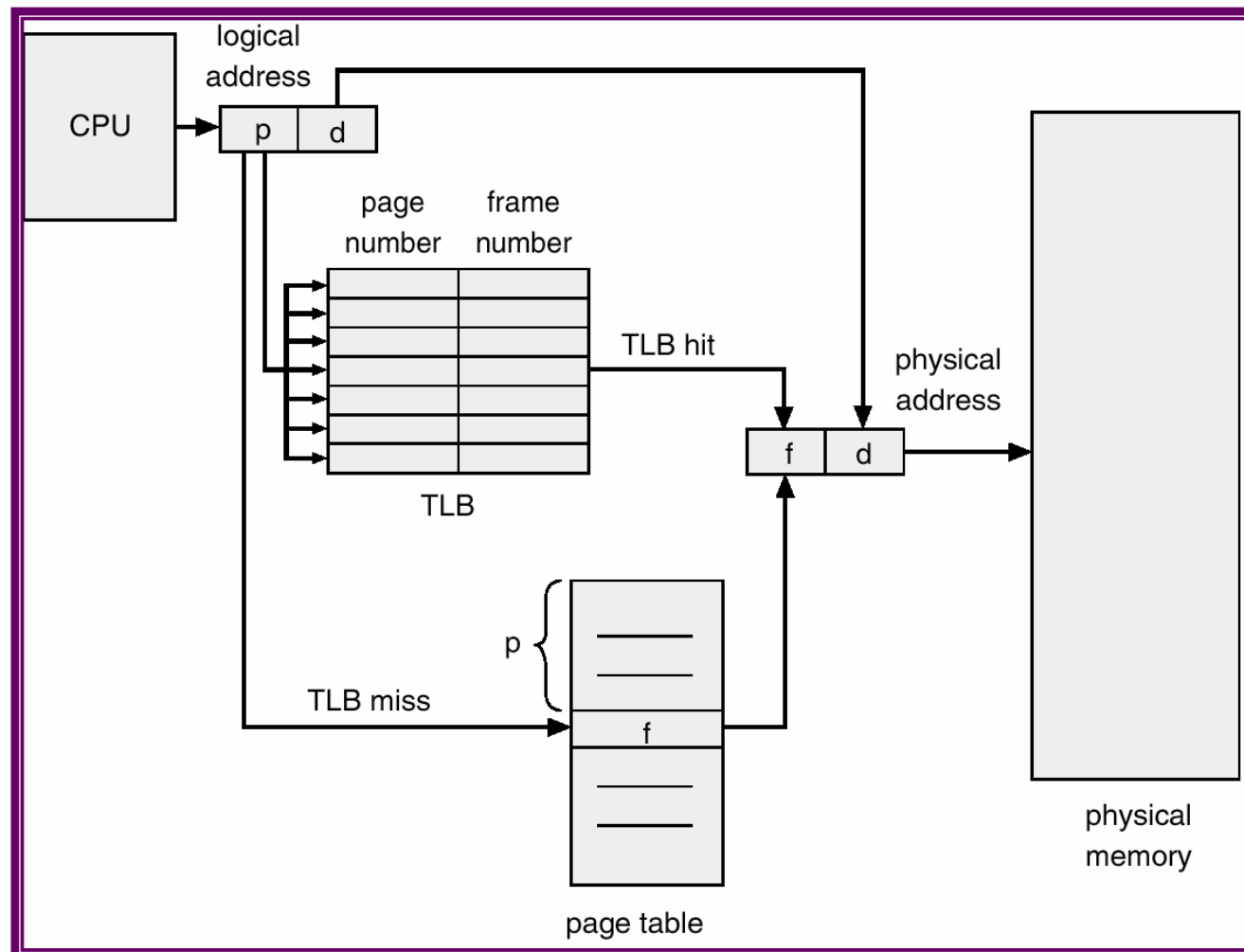


PAGING

Translation Look-aside Buffer

Address translation (A' , A'')

↳ If A' is in associative register, get frame # out, otherwise get frame # from page table in memory



Implementazione della Page Table

Idealmente, *la page table dovrebbe stare in registri veloci della MMU.*

Costoso al context switch (carico/ricarico di tutta la tabella)

Improprio se il numero delle pagine è elevato.

Es: indirizzi virtuali a 32 bit, pagine di 4K: ci sono $2^{20} > 10^6$ entry. A 32 bit l'una \rightarrow 4M in registri.

La page table viene tenuta in memoria principale

Rimane comunque un **grande consumo di memoria** (1 page table per ogni processo).

Con 100 processi \rightarrow 400M in page tables (su 256MB RAM complessivi).

Ogni accesso a dati/istruzioni richiede **2 accessi alla memoria**: uno per la page table e uno per i dati/istruzioni \rightarrow degrado del 100%.

Il virtual page number A' viene confrontato con tutte le entry **contemporaneamente**.

Se A' è nel TLB (**TLB hit**), si usa il **frame #** nel TLB.

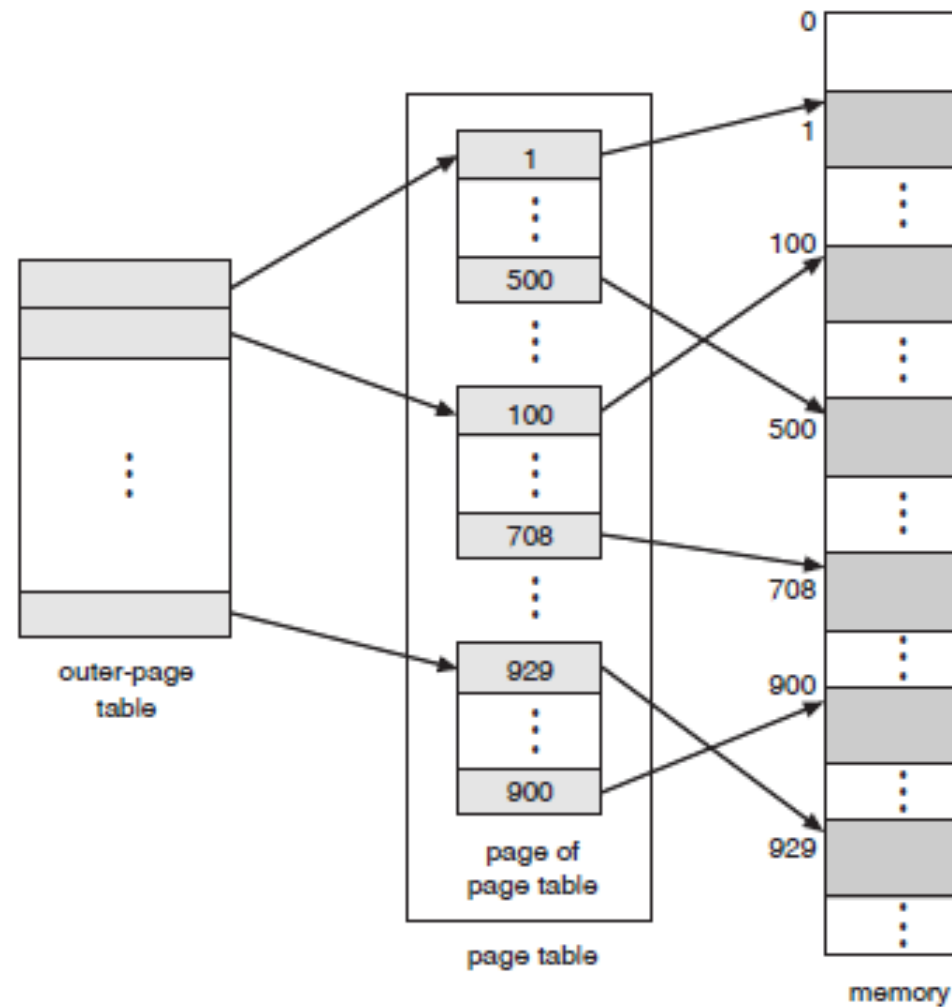
Altrimenti, la MMU esegue un normale lookup nelle page table in memoria, e sostituisce una entry della TLB con quella appena trovata.

Il S.O. viene informato solo nel caso di un page fault.

Paginazione a più livelli

Per ridurre l'occupazione della page table, si pagina la page table stessa.

Solo le pagine effettivamente usate sono allocate in memoria RAM.

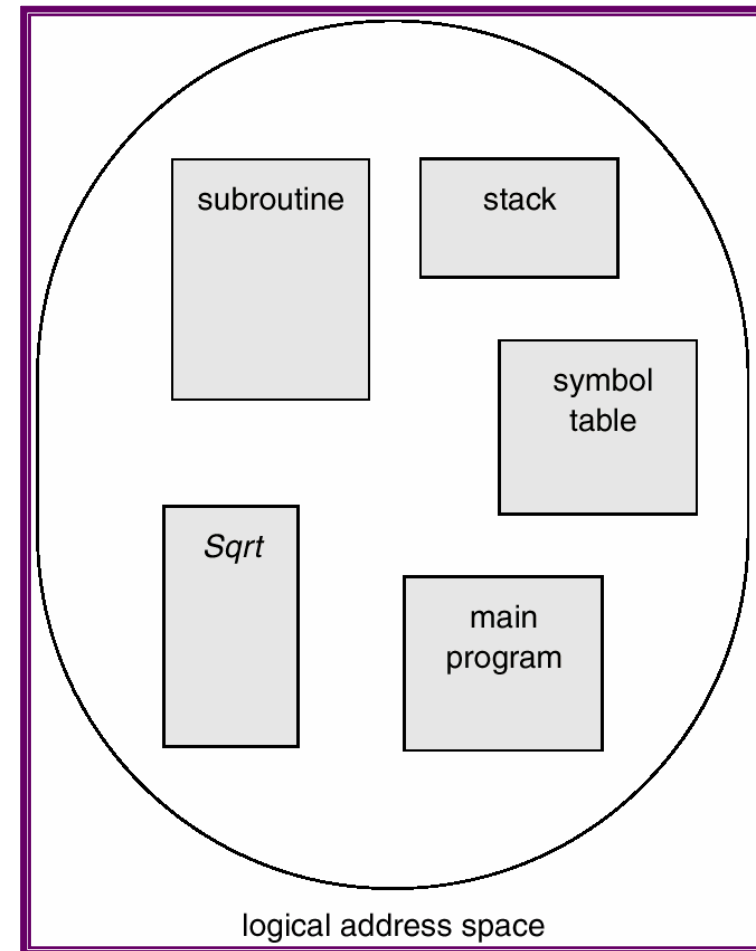


SEGMENTATION

➤ Memory-management scheme that supports user view of memory.

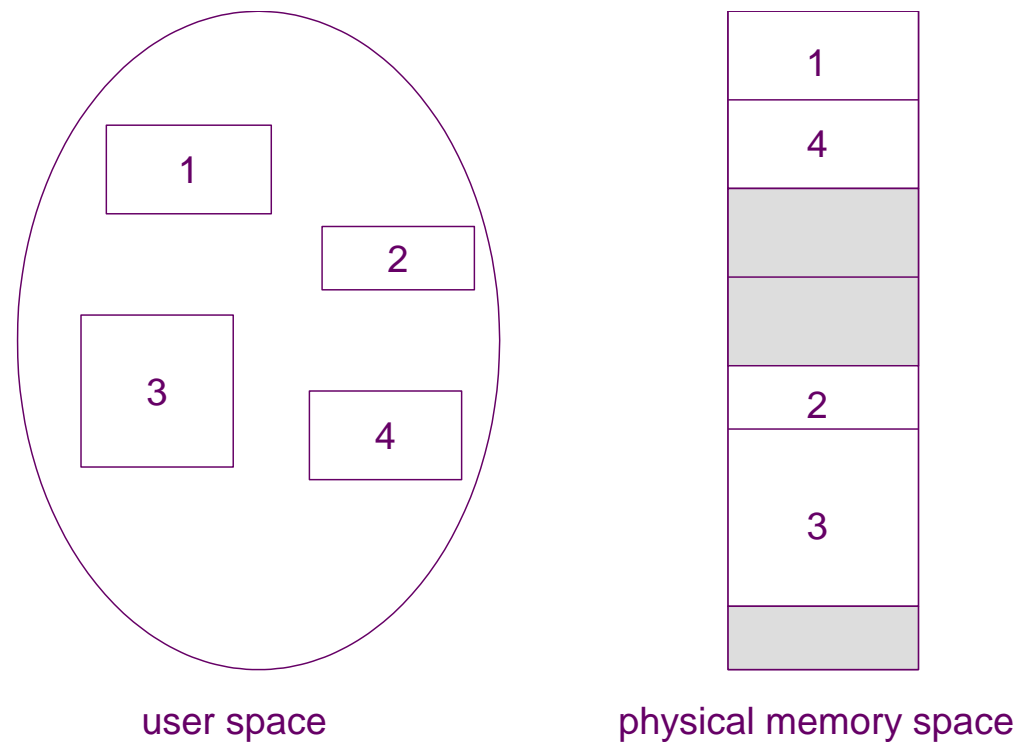
➤ A program is a collection of segments.
A segment is a logical unit such as:

- main program,
- procedure,
- function,
- method,
- object,
- local variables, global variables,
- common block,
- stack,
- symbol table, arrays.



User's View of a Program

SEGMENTATION



Logical View of Segmentation

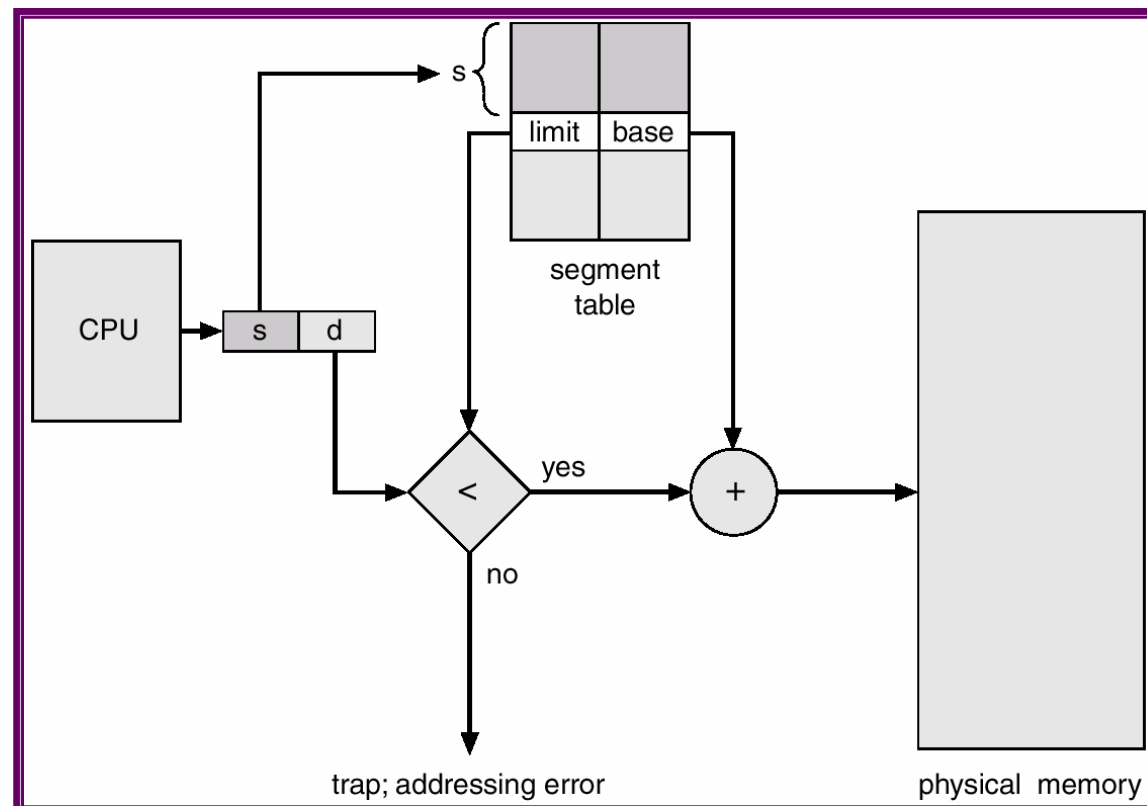
- **All segments of all programs do not have to be of the same length**
- **There is a maximum segment length**
- **Since segments are not equal, segmentation is similar to dynamic partitioning**

SEGMENTATION

⇒ Logical address consists of a two tuple: $\langle \text{segment-number}, \text{offset} \rangle$

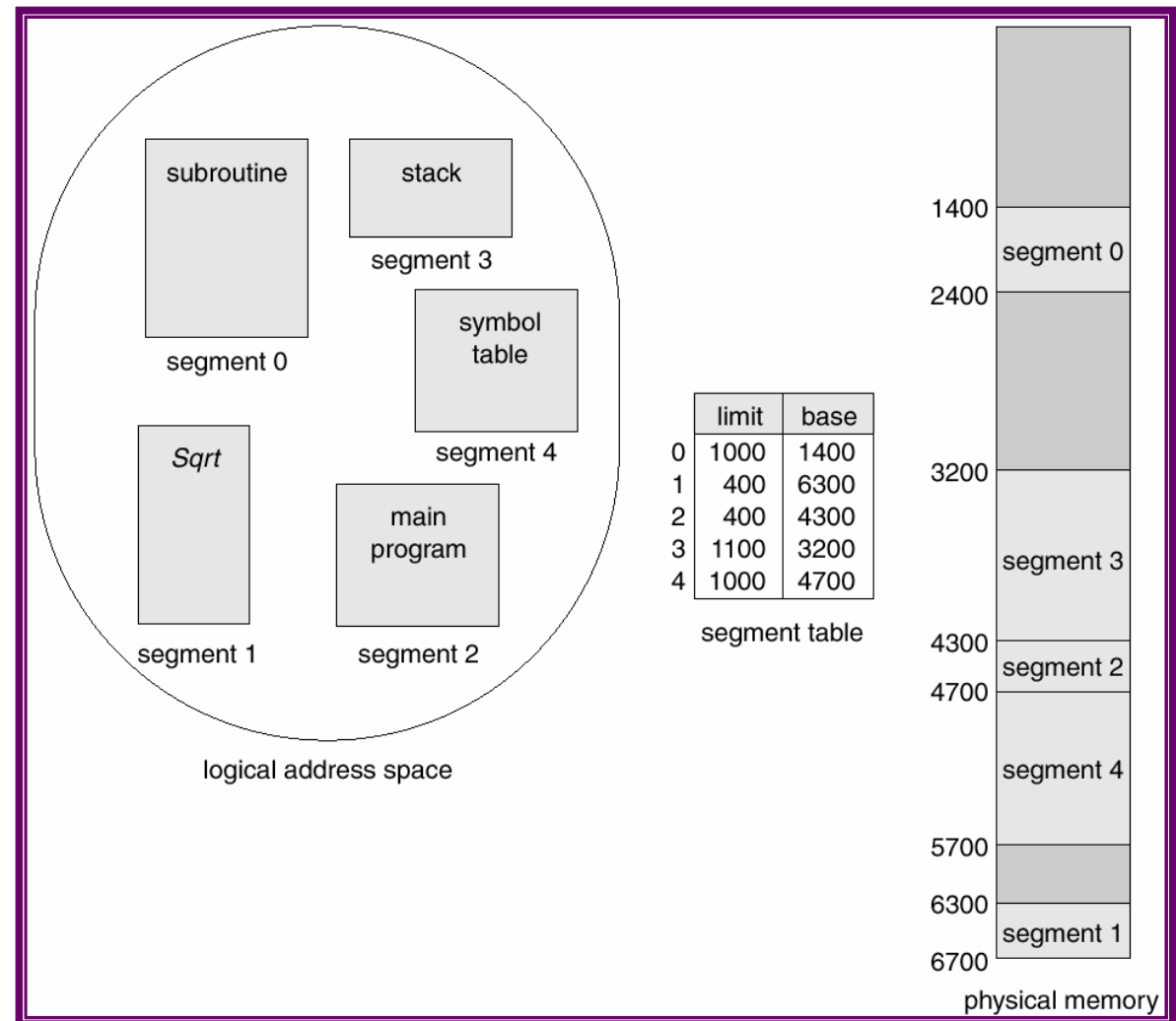
⇒ *Segment table* – maps two-dimensional physical addresses; each table entry has:

- base – contains the starting physical address where the segments reside in memory.
- limit – specifies the length of the segment.
- *Segment-table base register (STBR)* points to the segment table's location in memory.
- *Segment-table length register (STLR)* indicates number of segments used by a program;
- segment number s is legal if $s < \text{STLR}$.

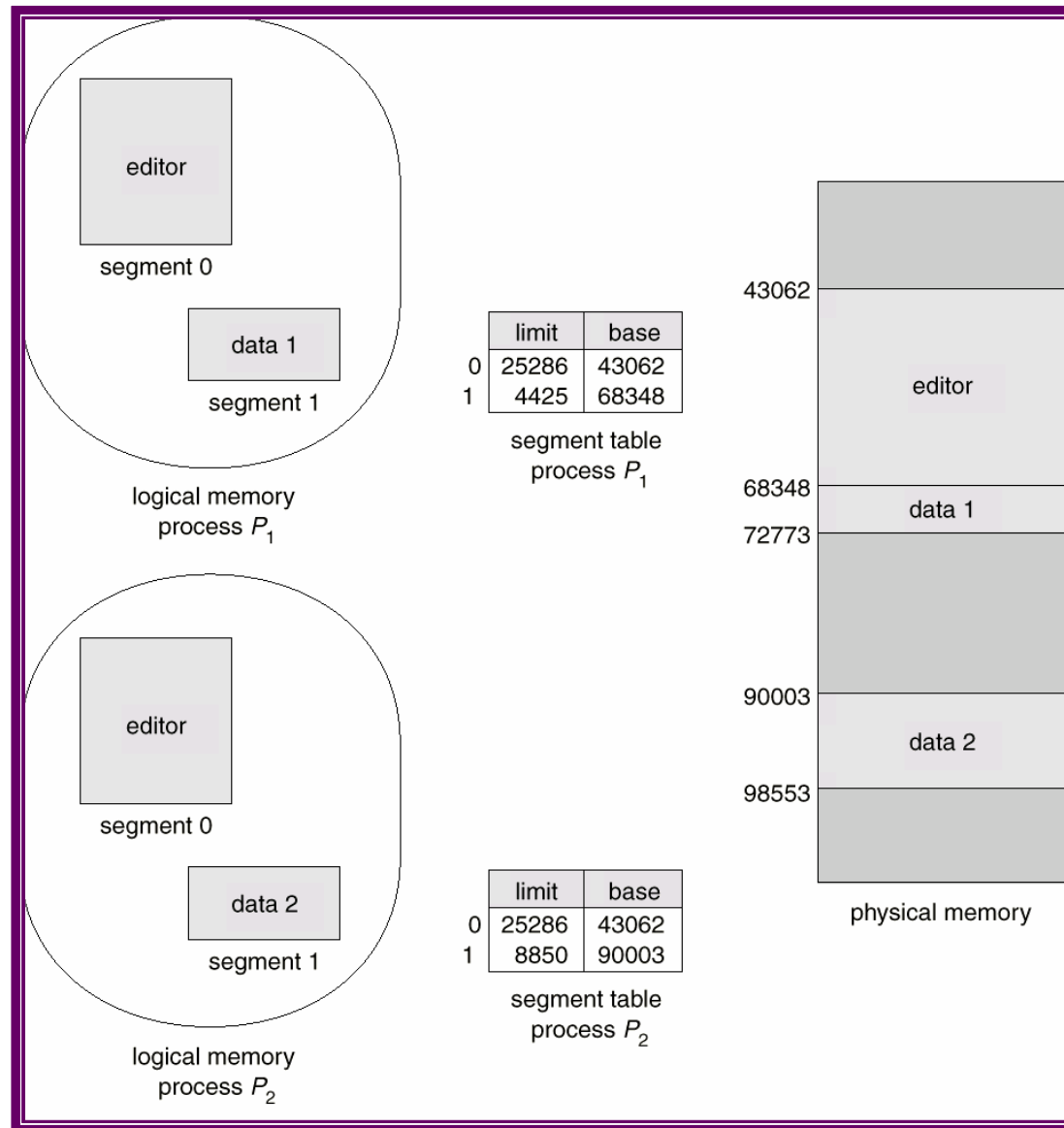


SEGMENTATION

- Relocation.
 - dynamic
 - by segment table
- Sharing.
 - shared segments
 - same segment number
- Allocation.
 - first fit/best fit
 - external fragmentation
- Protection: to each entry a bit indicates
 - the validity: → illegal segment
 - read/write/execute privileges
- Segments can change dimension during execution (f.e. the stack): dynamic memory allocation



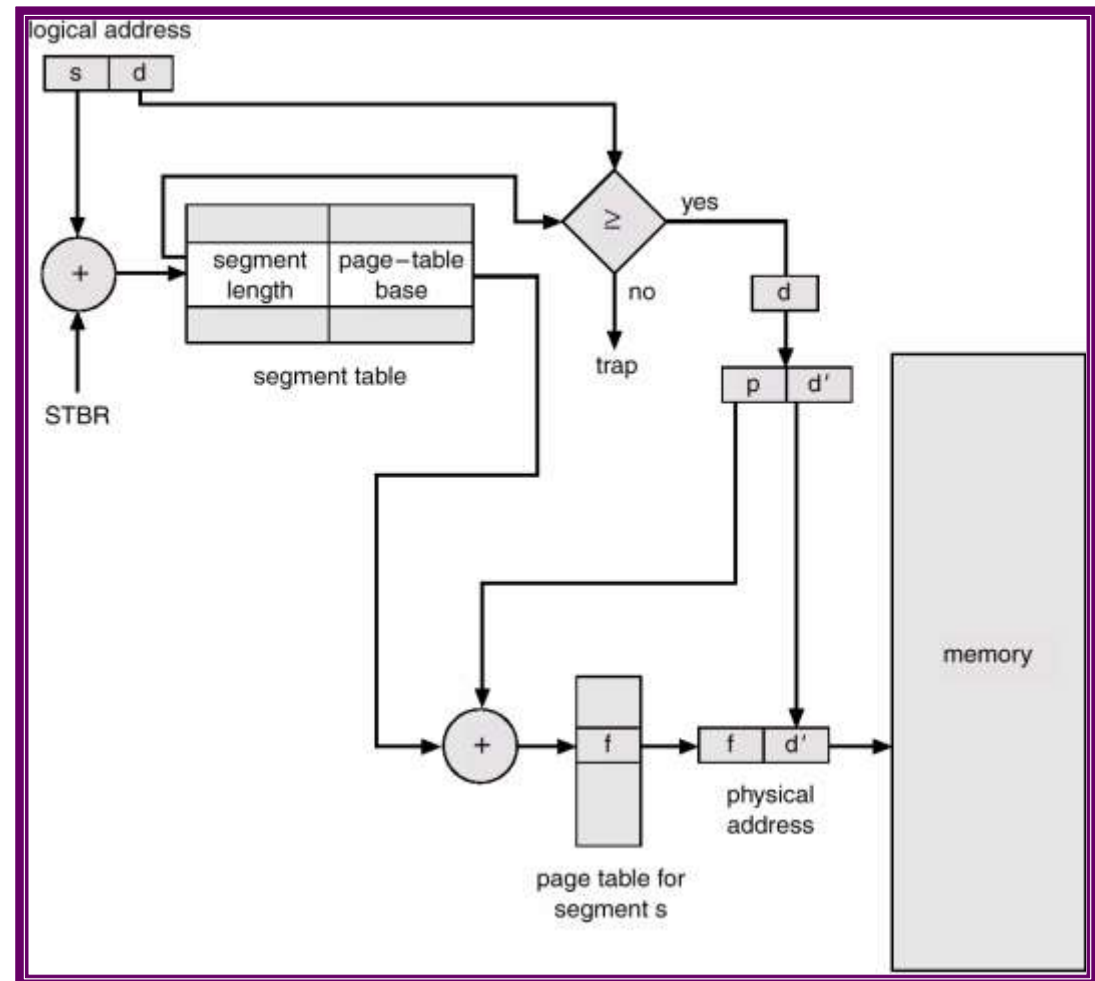
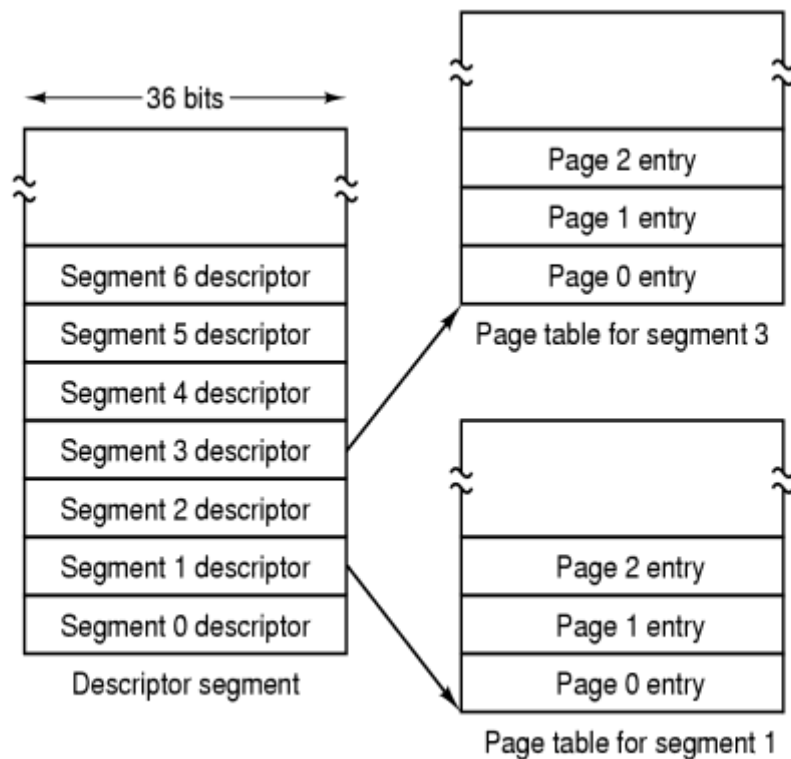
SEGMENTATION



Sharing of Segments

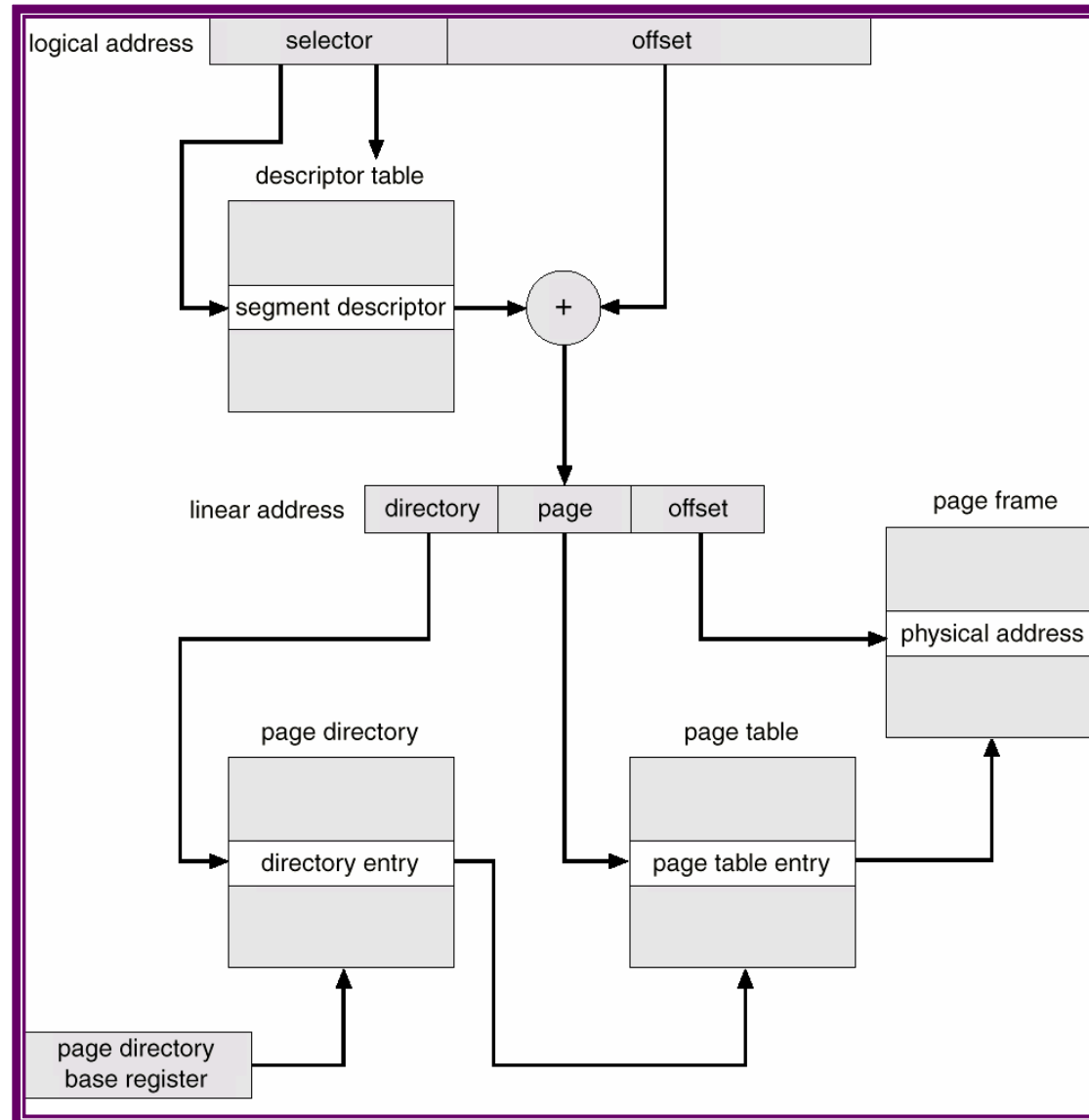
SEGMENTATION AND PAGINATION

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.



SEGMENTATION AND PAGINATION

As shown in the following diagram, the Intel 386 uses segmentation with paging for memory management with a two-level paging scheme.



SEGMENTATION AND PAGINATION

| Consideration | Paging | Segmentation |
|--|--|--|
| Need the programmer be aware that this technique is being used? | No | Yes |
| How many linear address spaces are there? | 1 | Many |
| Can the total address space exceed the size of physical memory? | Yes | Yes |
| Can procedures and data be distinguished and separately protected? | No | Yes |
| Can tables whose size fluctuates be accommodated easily? | No | Yes |
| Is sharing of procedures between users facilitated? | No | Yes |
| Why was this technique invented? | To get a large linear address space without having to buy more physical memory | To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection |

Paging and segmentation characteristics

PRINCIPI DI LOCALITÀ

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

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.

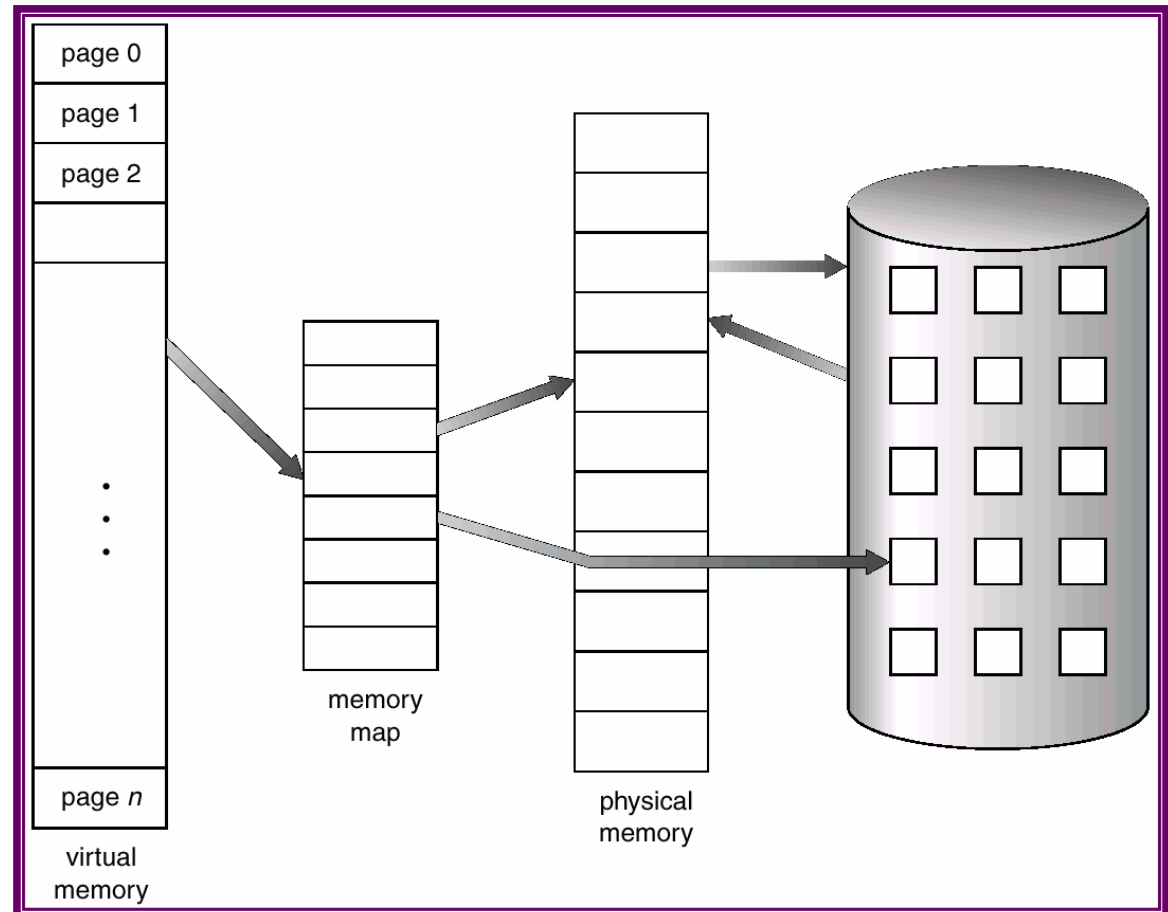
VIRTUAL STORAGE

➤ **Virtual memory** – separation of user logical memory from physical memory.

- 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



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 \Rightarrow reference to it

- ✚ not-in-memory \Rightarrow bring to memory

✚ With each page table entry a valid–invalid bit is associated

(1 \Rightarrow in-memory, 0 \Rightarrow not-in-memory)

✚ Initially valid–invalid bit is set to 0 on all entries.

✚ Example of a page table snapshot.

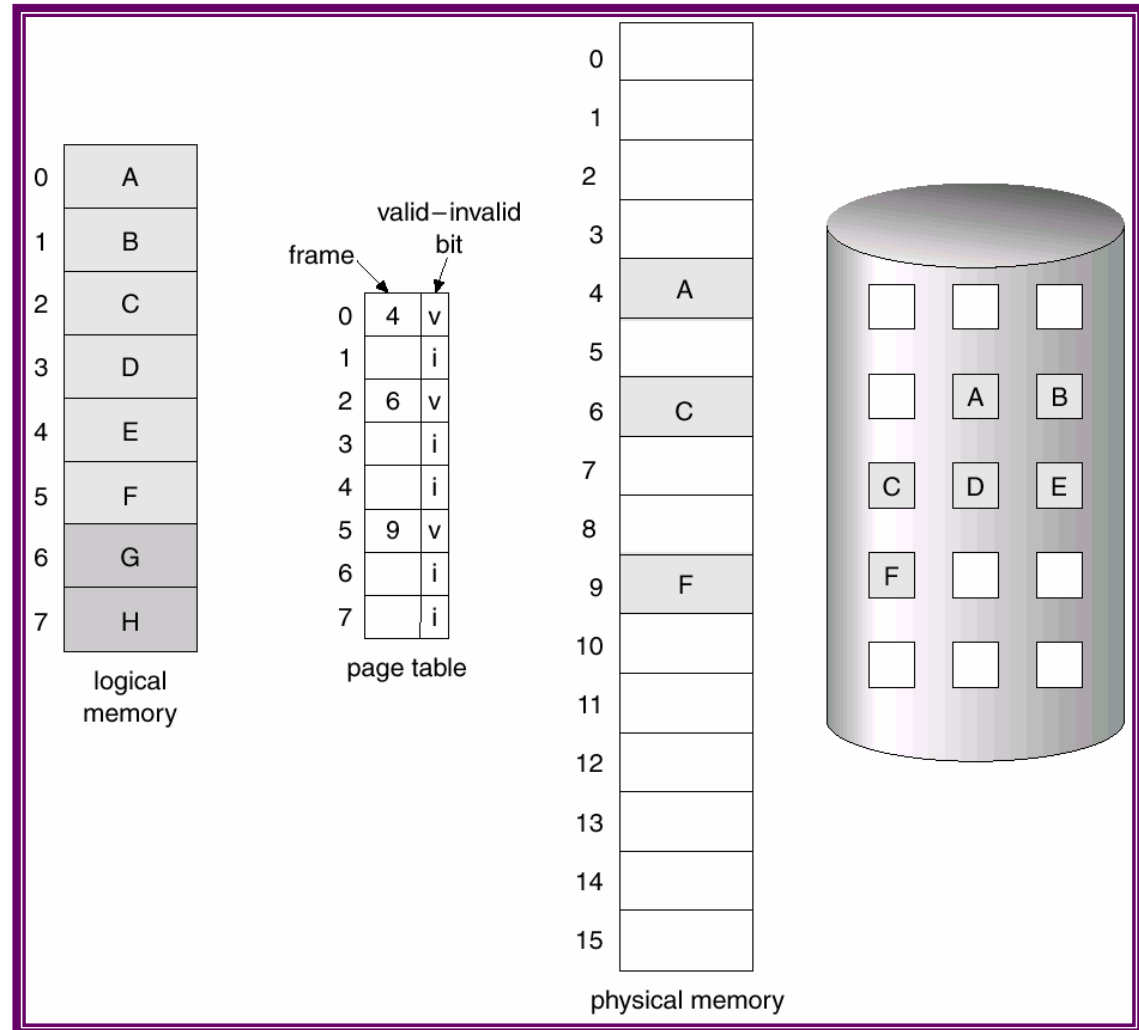
| Page # | Invalid Bit |
|--------|-------------|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 0 |
| 4 | 0 |

Page Table

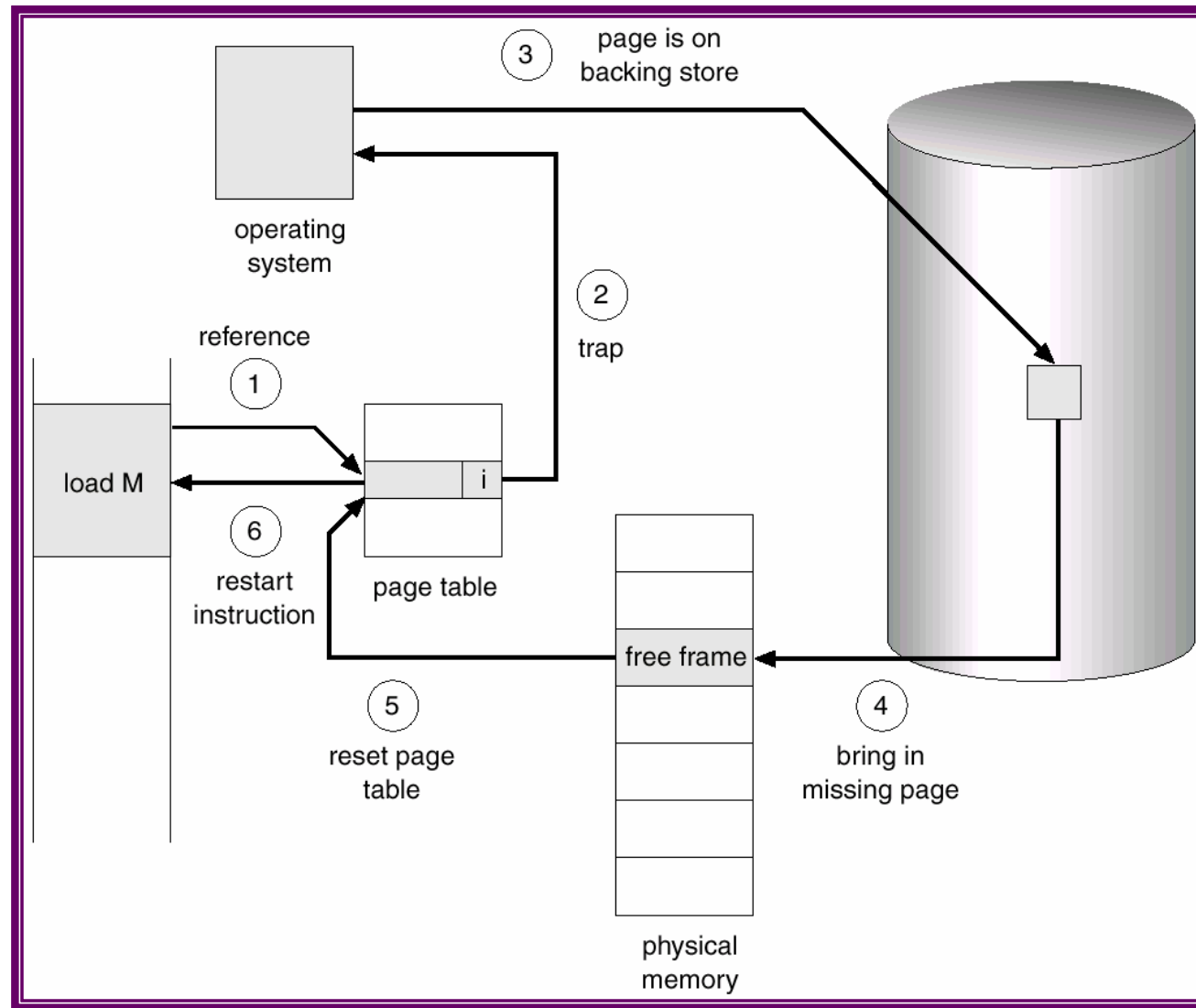
During address translation,
if valid–invalid bit in page
table entry is 0 \Rightarrow page
fault.

DEMAND PAGING

- If there is ever a reference to a page, first reference will trap to OS \Rightarrow **page fault**
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
 - Block move
 - auto increment/decrement location



DEMAND PAGING



Steps in Handling a Page Fault

LA STRUTTURA TABELLARE

| Task ID | # page | ↑ PMT | S bit | | B | Task ID | P | C bit | R bit | S bit |
|---------|--------|-------|-------|--|---|---------|---|-------|-------|-------|
| 63 | 4 | 7 | 0 | | 0 | 51 | 0 | 0 | 0 | 1 |
| 25 | 6 | 9 | 0 | | 1 | 50 | 0 | W | 0 | 1 |
| 44 | 12 | 8 | 0 | | 2 | 54 | 4 | 0 | 0 | 1 |
| 50 | 7 | 2 | 1 | | 3 | 53 | 0 | Z | 1 | 1 |
| 52 | 8 | 6 | 1 | | 4 | 49 | 1 | 0 | 1 | 1 |
| 54 | 6 | 5 | 1 | | 5 | 52 | 6 | 0 | 0 | 1 |
| 51 | 3 | 3 | 1 | | 6 | 52 | 4 | 1 | Z | 1 |
| 49 | 4 | 1 | 1 | | 7 | 49 | 3 | W | Z | 1 |
| 53 | 5 | 4 | 1 | | 8 | 50 | 4 | Z | 0 | 1 |

Job (Task) Table

| | | | | | |
|----|----|---|---|---|---|
| 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

| 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

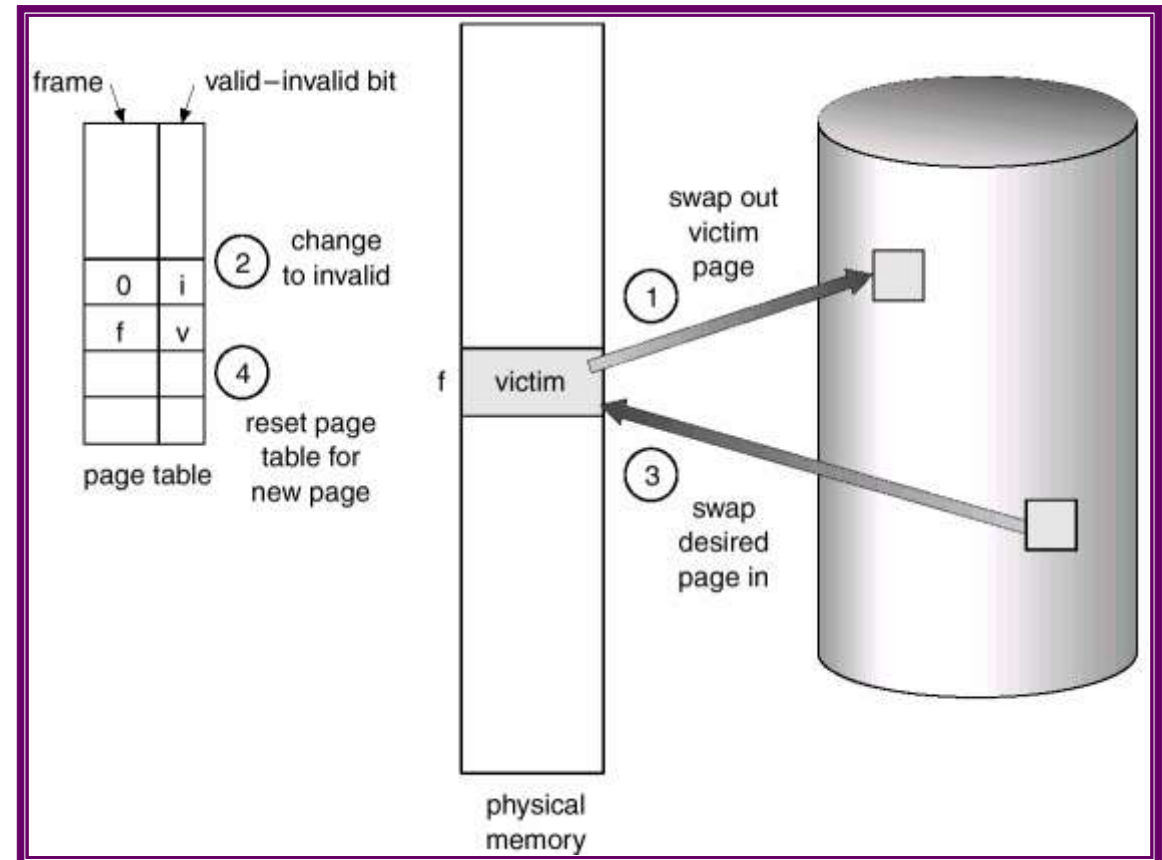
| # | Task ID | P | C bit | C T S | S bit |
|----|---------|---|-------|----------|-------|
| 0 | 51 | 1 | 0 | 13 4 10 | 1 |
| 1 | 51 | 2 | 0 | 99 20 5 | 1 |
| 2 | 53 | 1 | 0 | 22 12 10 | 1 |
| 3 | 49 | 0 | 1 | 4 6 18 | 1 |
| 4 | 53 | 2 | 0 | 14 18 25 | 1 |
| 5 | 49 | 2 | 1 | 105 21 5 | 1 |
| 6 | 53 | 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

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.
- Same page may be brought into memory several times.
- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Change bit (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.



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 Algorithms

☞ Reference 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.

WINDOWS NT

- ⇒ Uses demand paging with clustering. Clustering brings in pages surrounding the faulting page.
- ⇒ Processes are assigned working set minimum and working set maximum.
- ⇒ Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- ⇒ A process may be assigned as many pages up to its working set maximum.
- ⇒ When the amount of free memory in the system falls below a threshold, automatic working set trimming is performed to restore the amount of free memory.
- ⇒ Working set trimming removes pages from processes that have pages in excess of their working set minimum.

DEMAND SEGMENTATION

DEMAND SEGMENTATION AND PAGING

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.