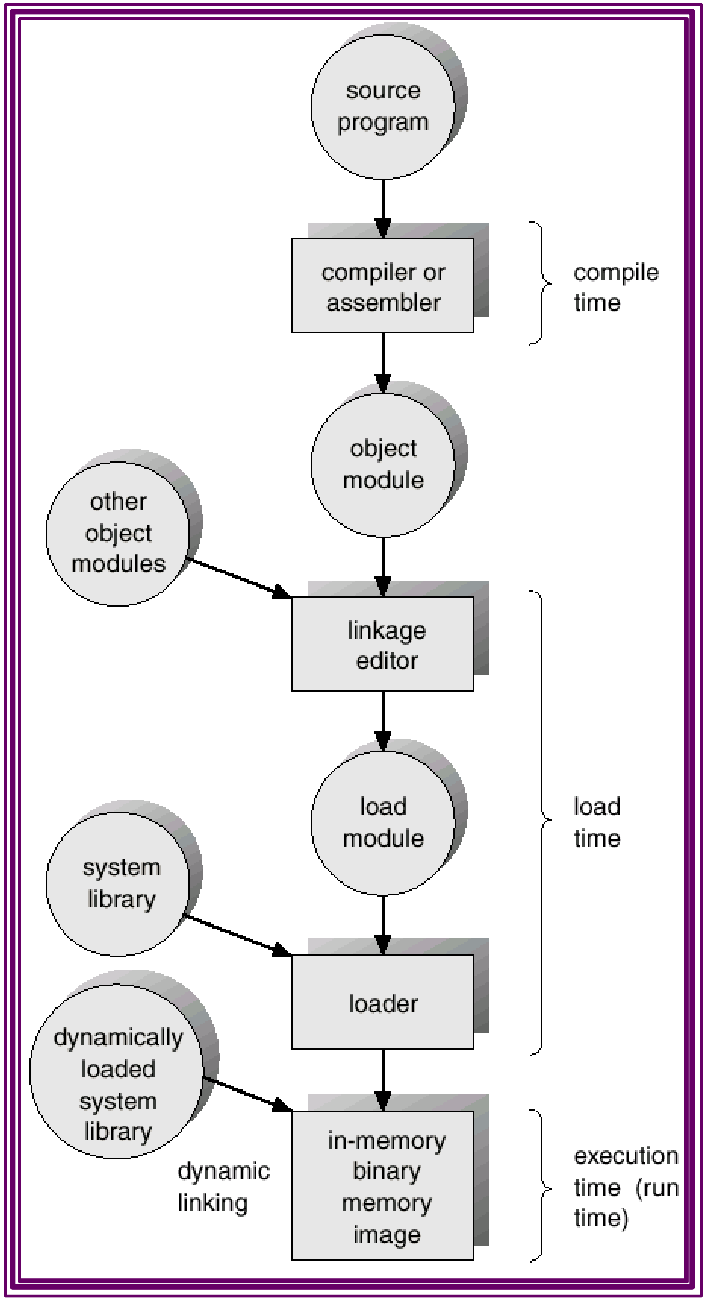
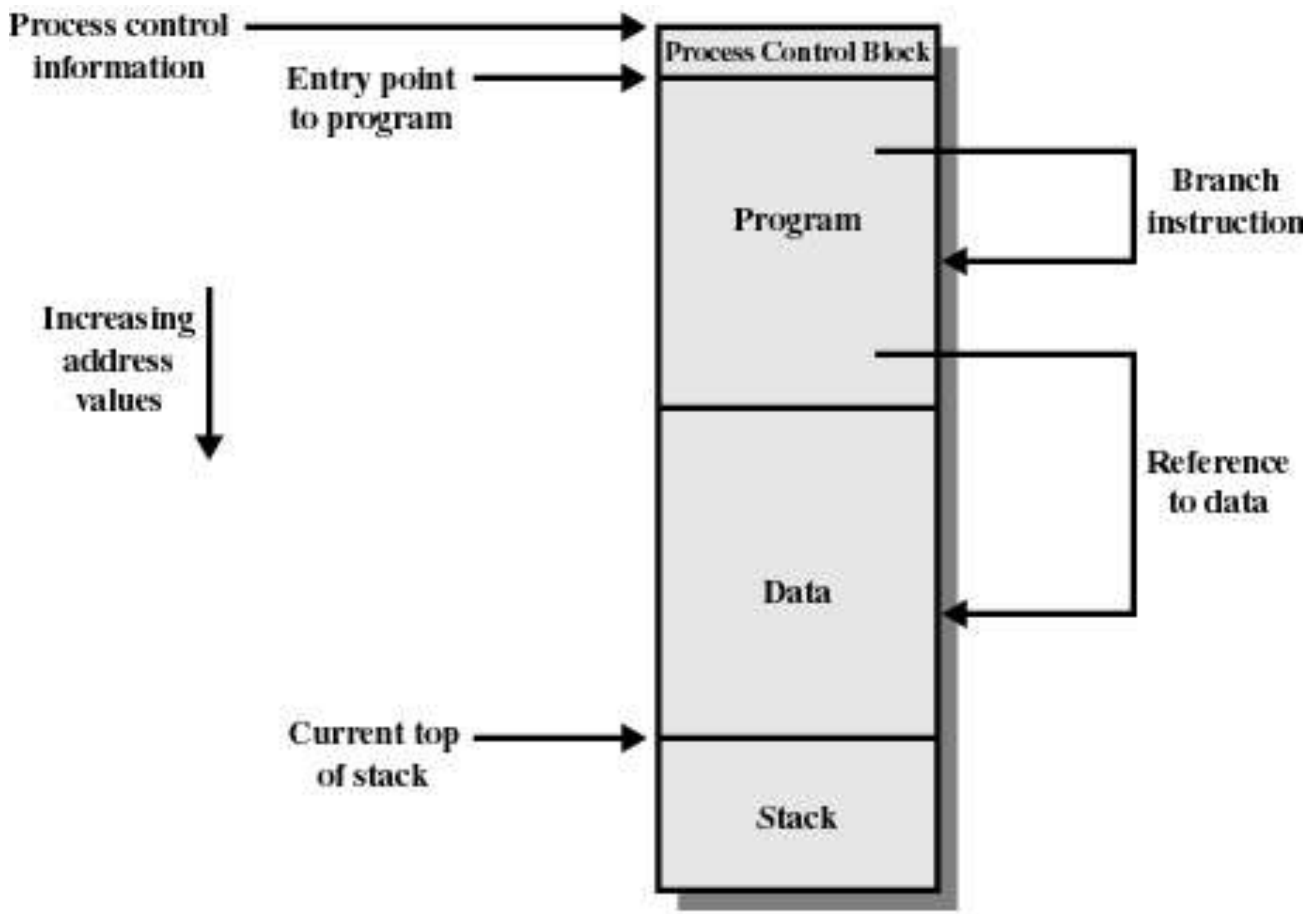
FROM THE SOURCE PROGRAM TO THE EXECUTABLE PROGRAM

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



ADDRESS SPACE STRUCTURE



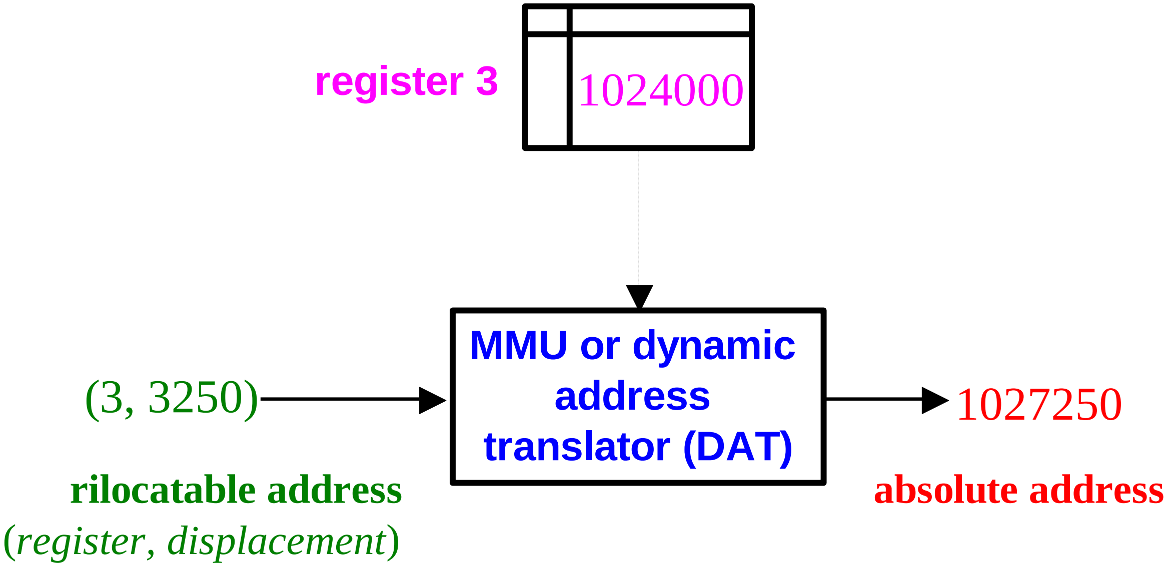
Processes have three segments: text, data, and stack.
All three are in one address space, but separate instruction and data space is also supported.

RELOCATABLE ADDRESSES OF A PROGRAM

Memory Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU 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.



DYNAMIC LOADING INTO THE MEMORY

- 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.
- No special support from the operating system is required
- Implemented through program design.

DYNAMIC LINKING

- Linking postponed until execution time.
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.
- Stub replaces itself with the address of the routine, and executes the routine.
- Operating system needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

OVERLAY

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system, programming design of overlay structure is complex

MEMORY MANAGEMENT ALGORITHMS

Single-programming

- *Contiguous memory allocation*

Multi-programming

- *Memory partitioning*
  - *static partitioning,*
  - *dinamic partitioning,*
  - *rilocatable partitioning,*
  - *multiple partitioning,*
  - *swapping,*
  - *rolling*

- *Real memory paging (segmentation)*
  - *paging*
  - *segmentation*
  - *paging with segmentation*

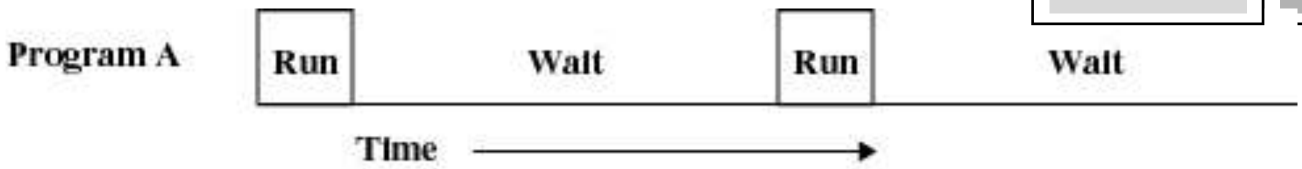
- *Virtual storage paging (segmentation)*
  - *paged virtual storage (demand paging)*
  - *segmented virtual storage*
  - *Segmented and paged virtual storage*
  - *Multiple virtual storage*

SINGLE-PROGRAMMING

Contiguous memory allocation

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

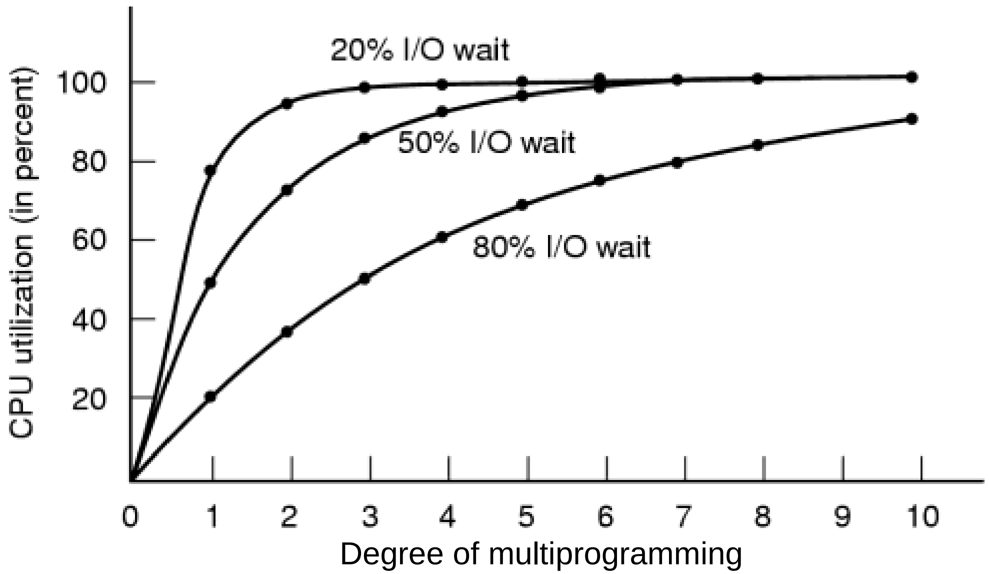
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.



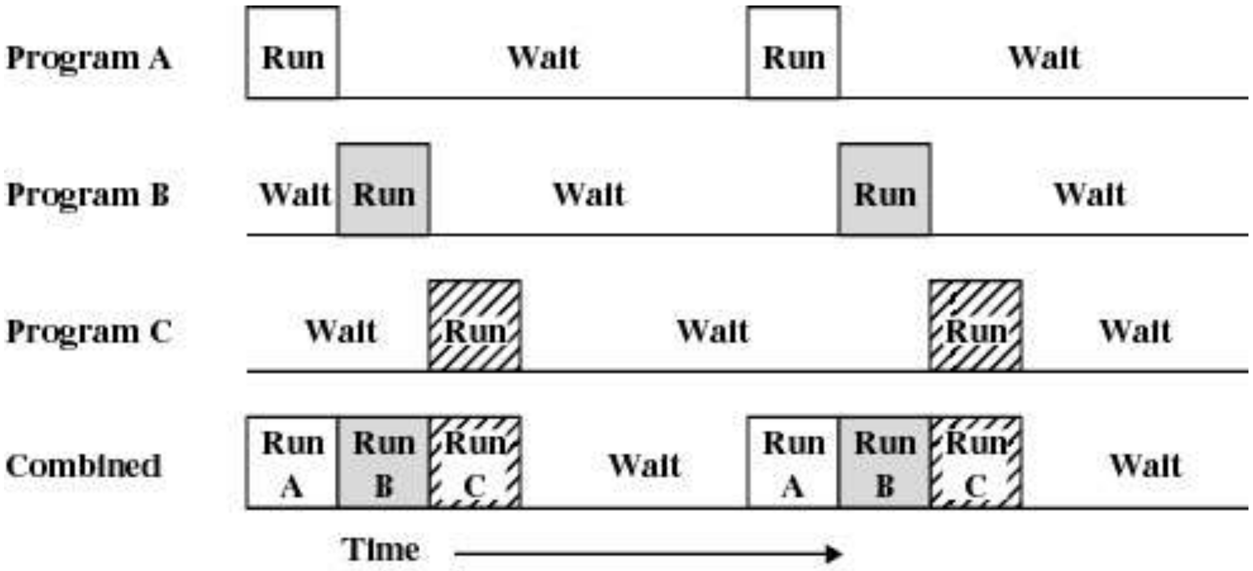
(a) Uniprogramming

MULTI-PROGRAMMIG

Pro



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



(c) Multiprogramming with three programs

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
- 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

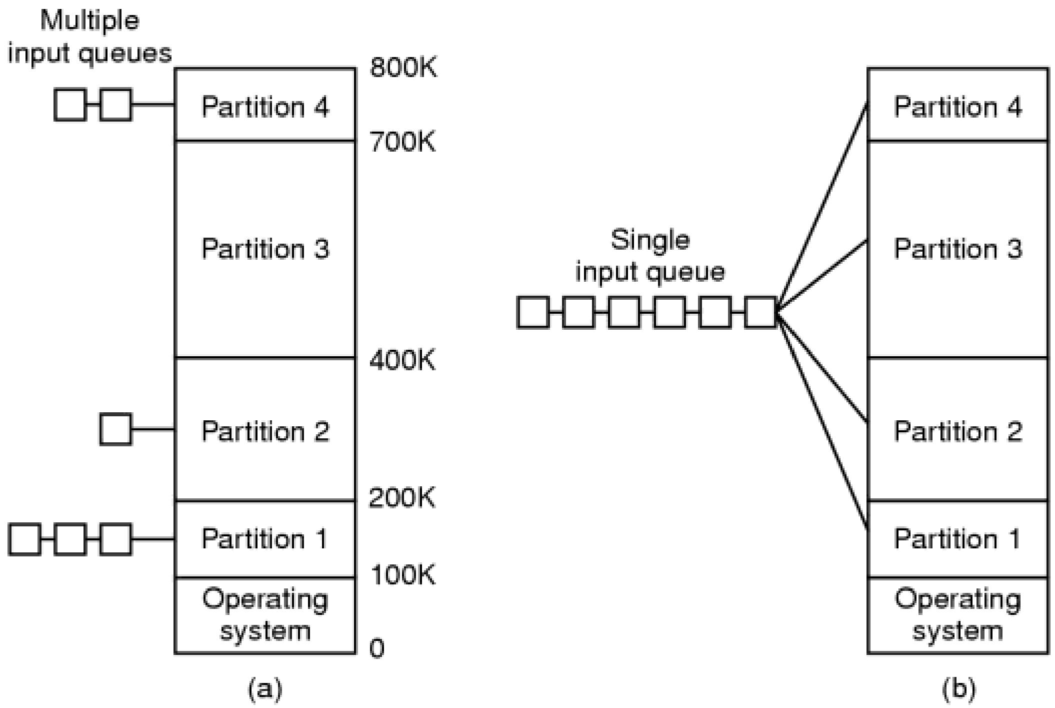
Logical Organization

- 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



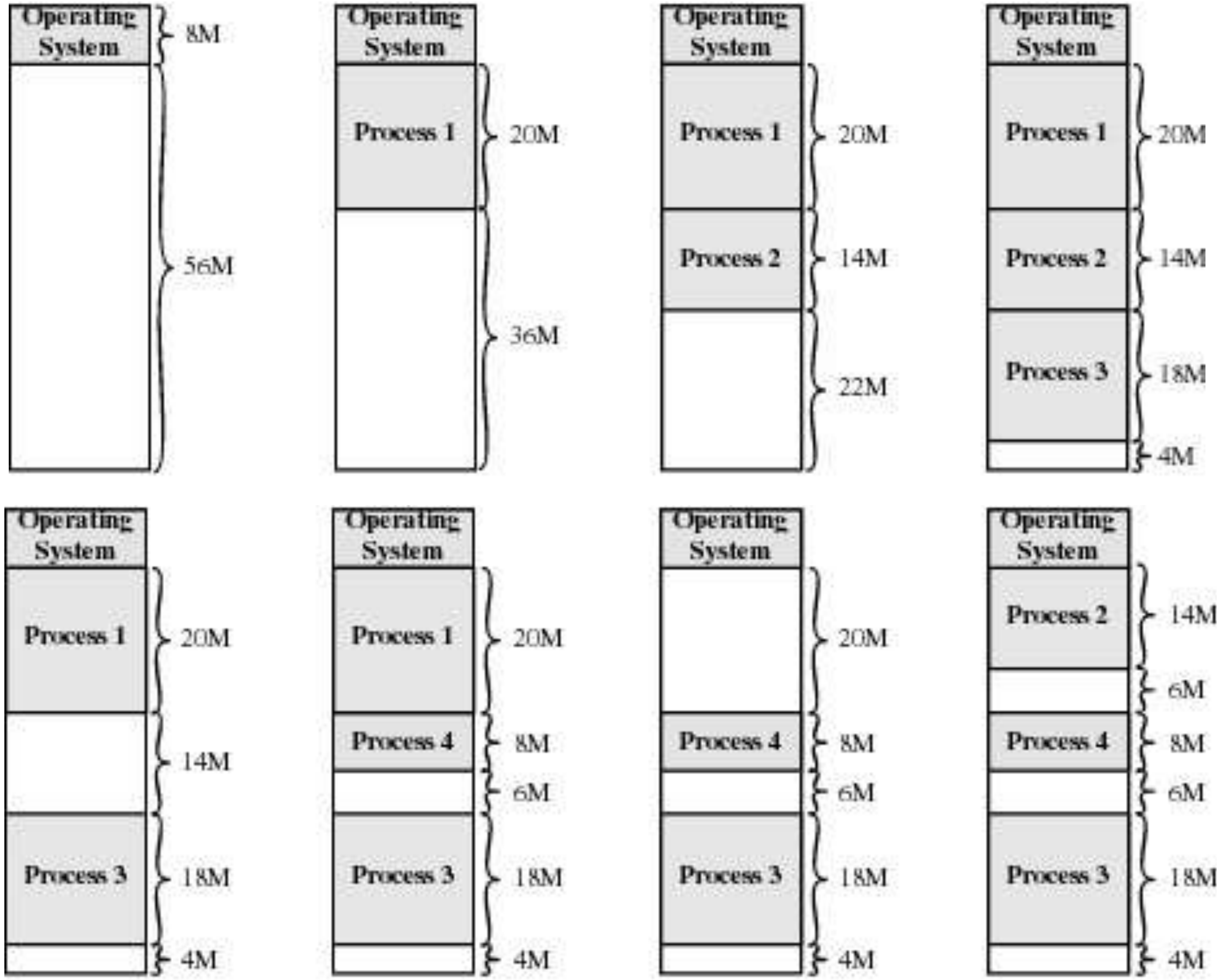
The OS keeps track of the memory through the Partition Table

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

Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

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.

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

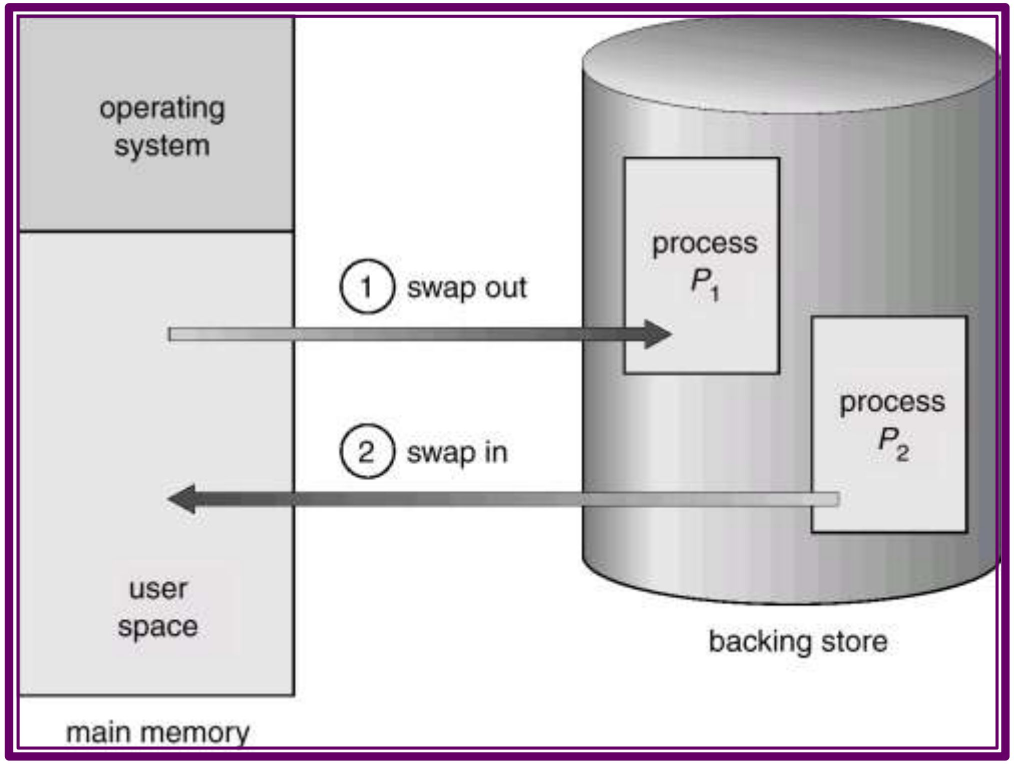
Si basa sull'assunzione che lo spazio degli indirizzi di un programma non debba essere tutto contiguo in memoria centrale.

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

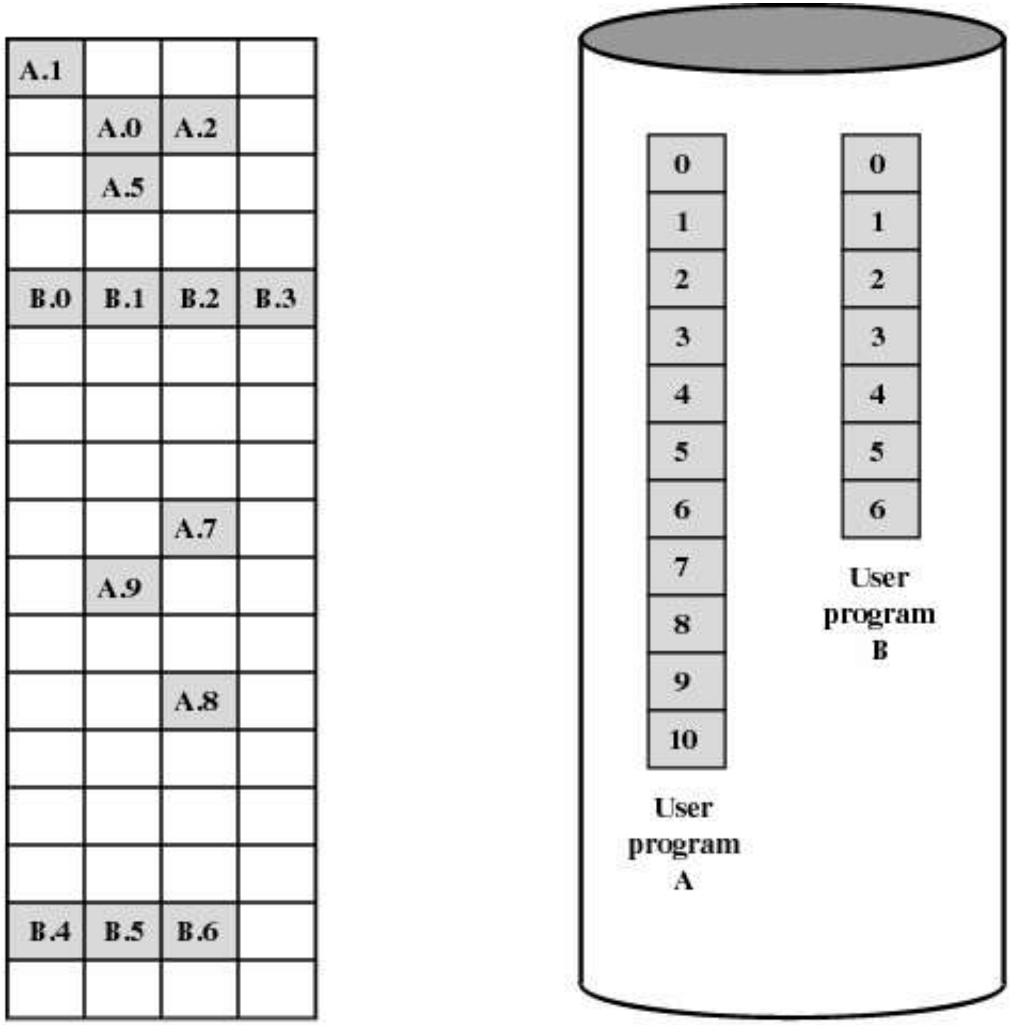
- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- *Roll out, roll 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.



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).
- 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.
- 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

PAGING (VIRTUAL MEMORY CONCEPT)



Main Memory

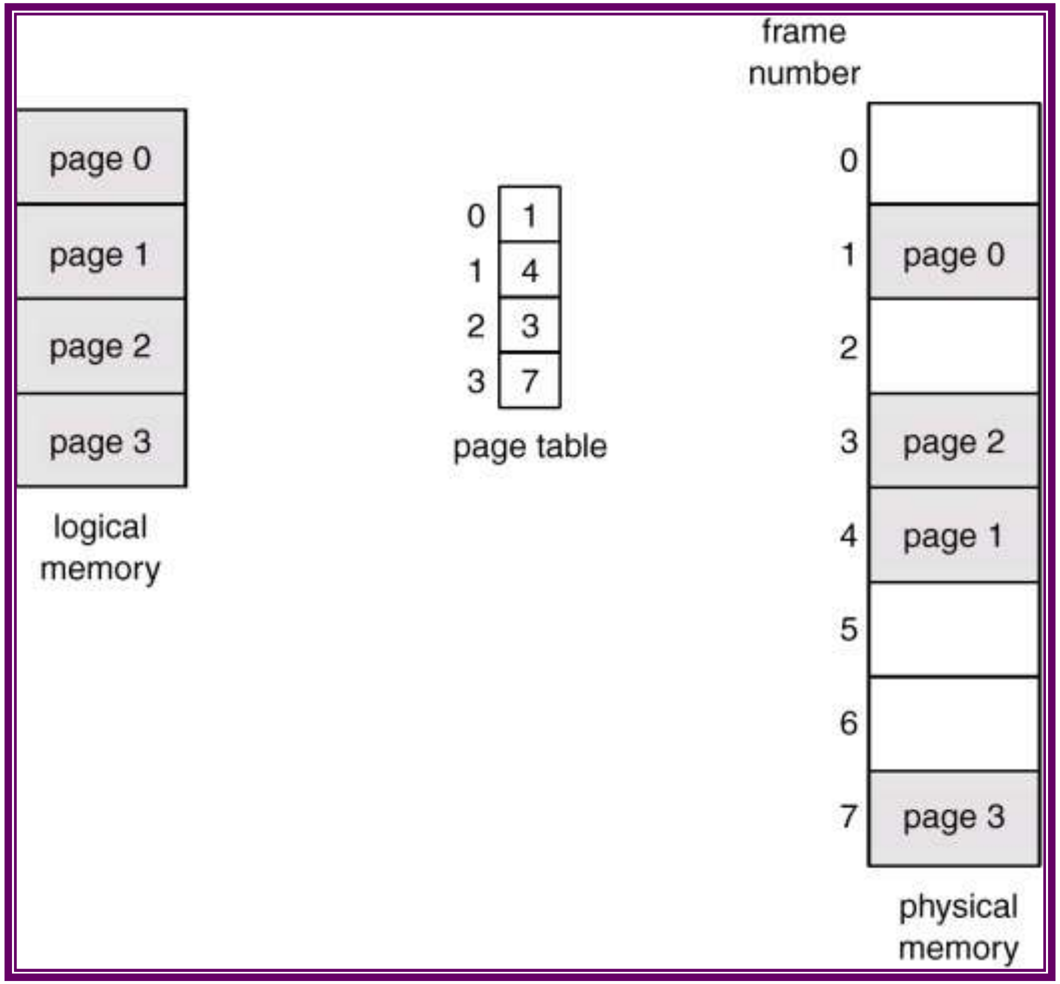
Disk

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.

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 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.

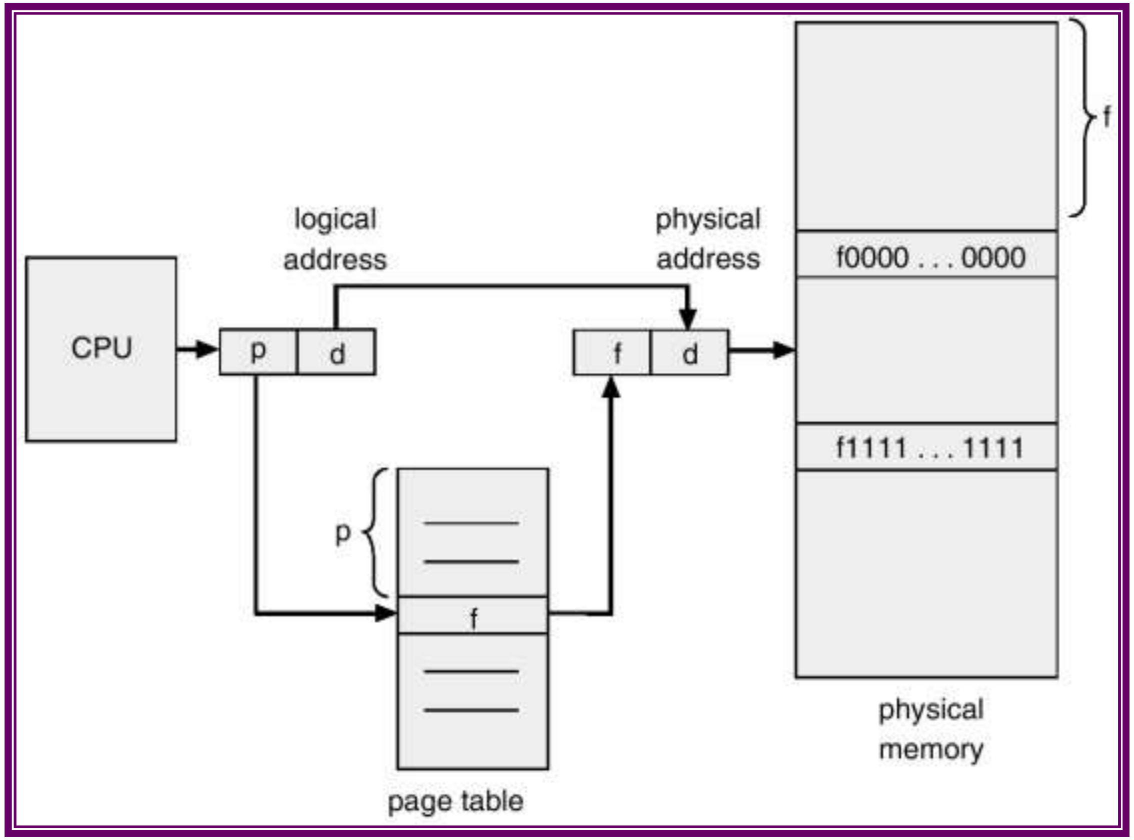


Tramite la Page Table di ciascun programma il SO tiene traccia del blocco di memoria in cui le varie pagine sono state caricate

PAGING

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

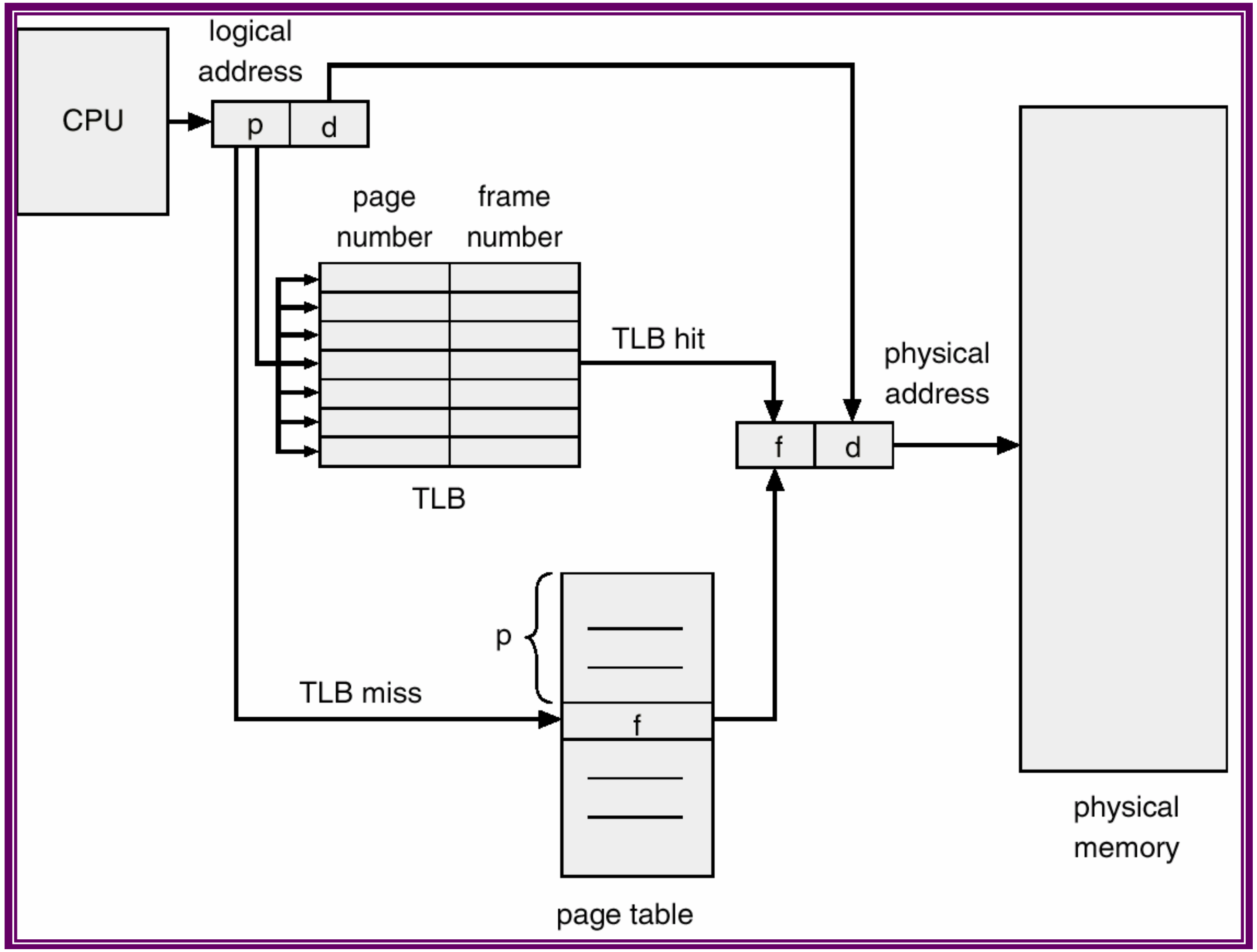
- Page table is kept in main memory.
- *Page-table base register (*PTBR) points to the page table.
- *Page-table length register* (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- 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

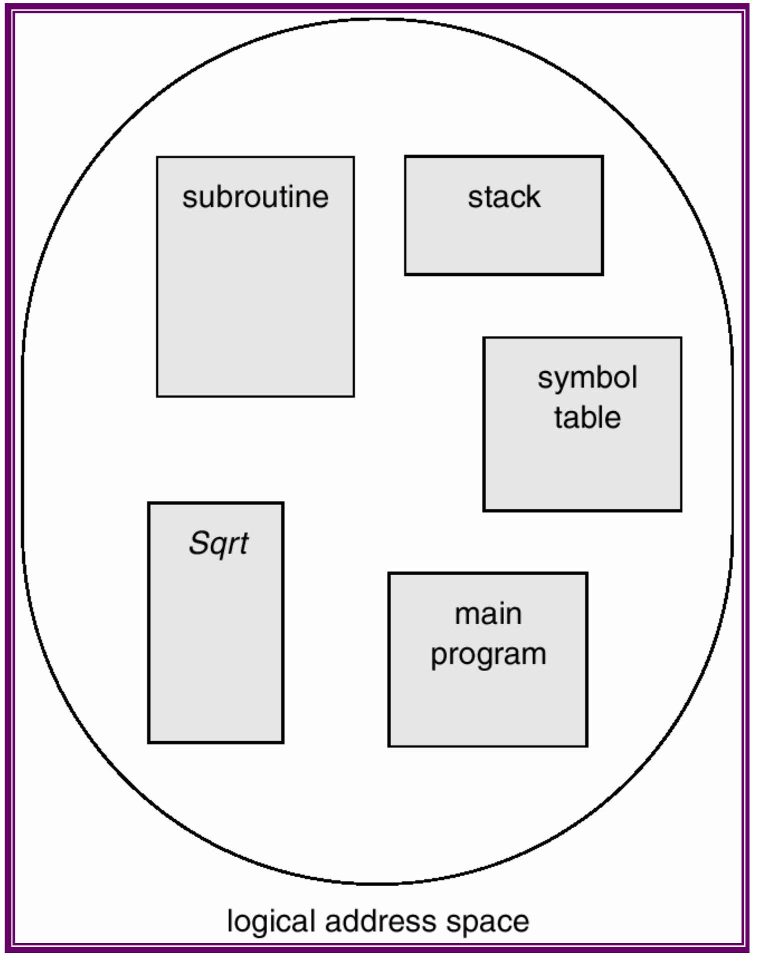
Address translation (A´, A´´)

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



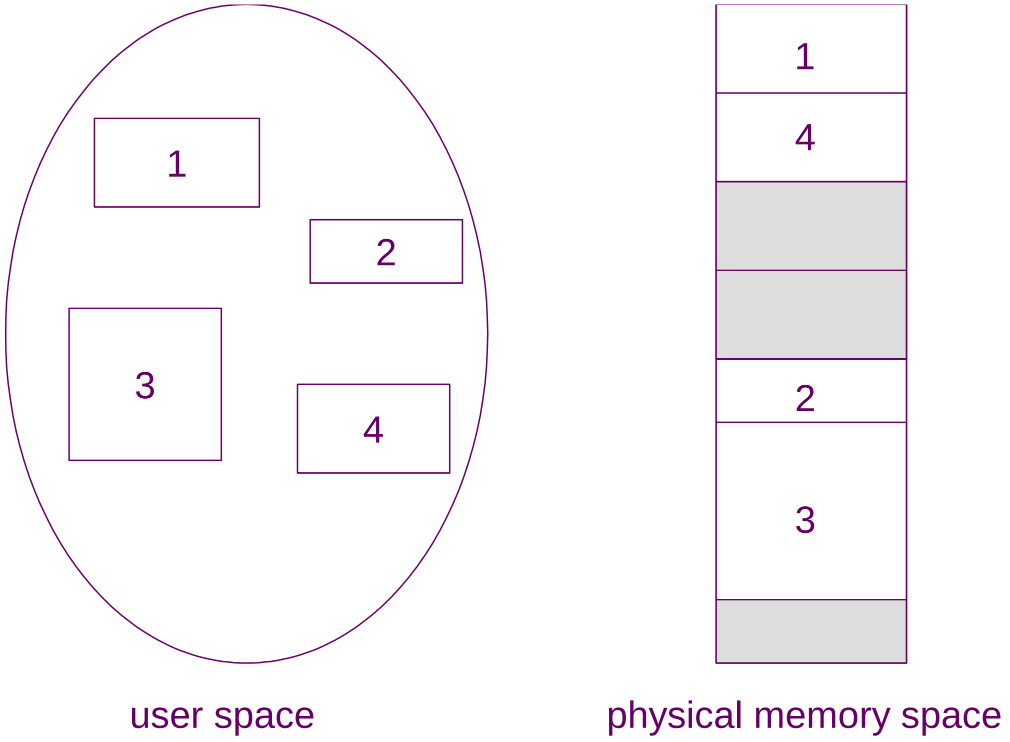
SEGMENTATION

- 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

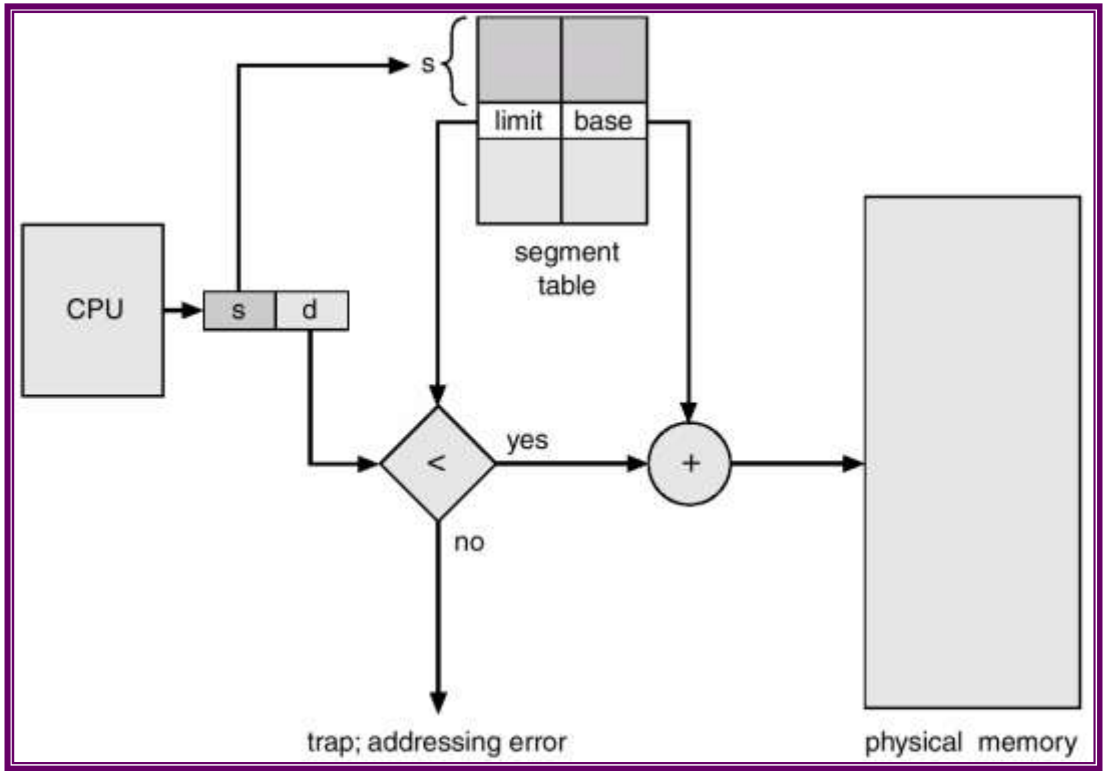
- **All segments of all programs do not have to be of the same length**
- **There is a maximum segment length**
- **Since segments are not equal, segmentation is similar to dynamic partitioning**

SEGMENTATION

- Logical address consists of a two tuple:

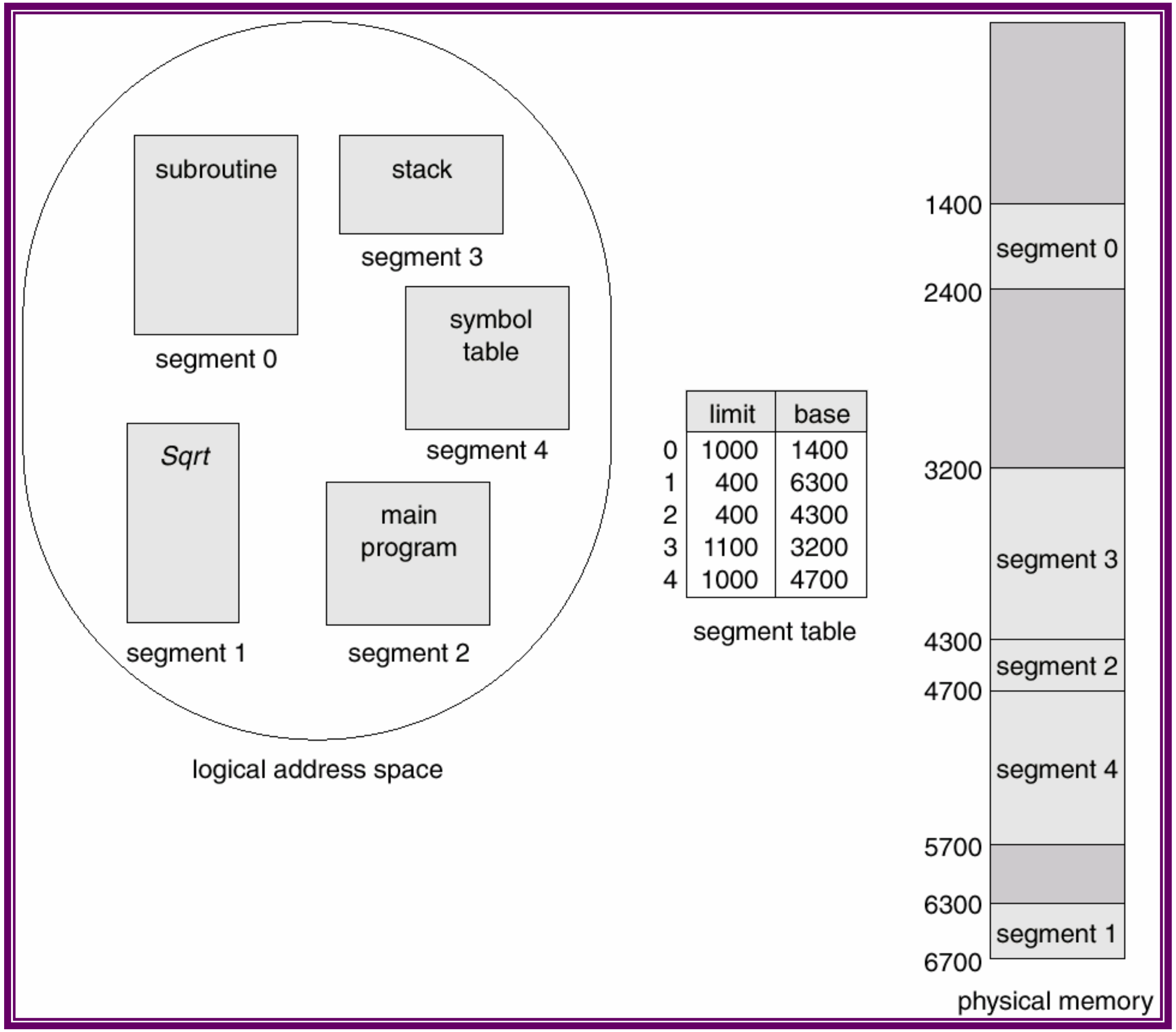
  $<$segment-number, offset$>$

- *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* < STLR.



SEGMENTATION

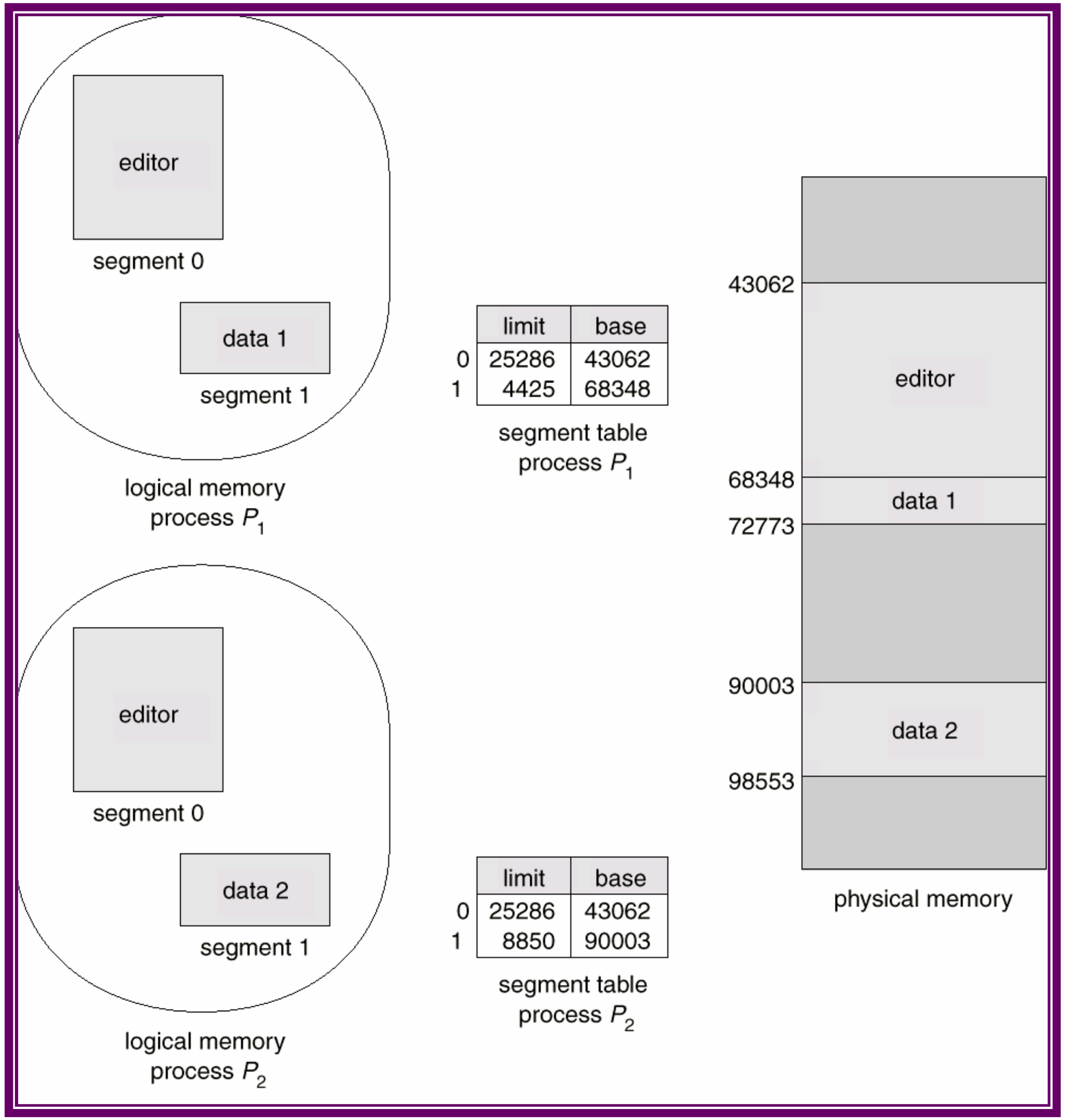
- Relocation.
  - dynamic
  - by segment table
- Sharing.
  - shared segments
  - same segment number
- Allocation.
  - first fit/best fit
  - external fragmentation



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Example of Segmentation

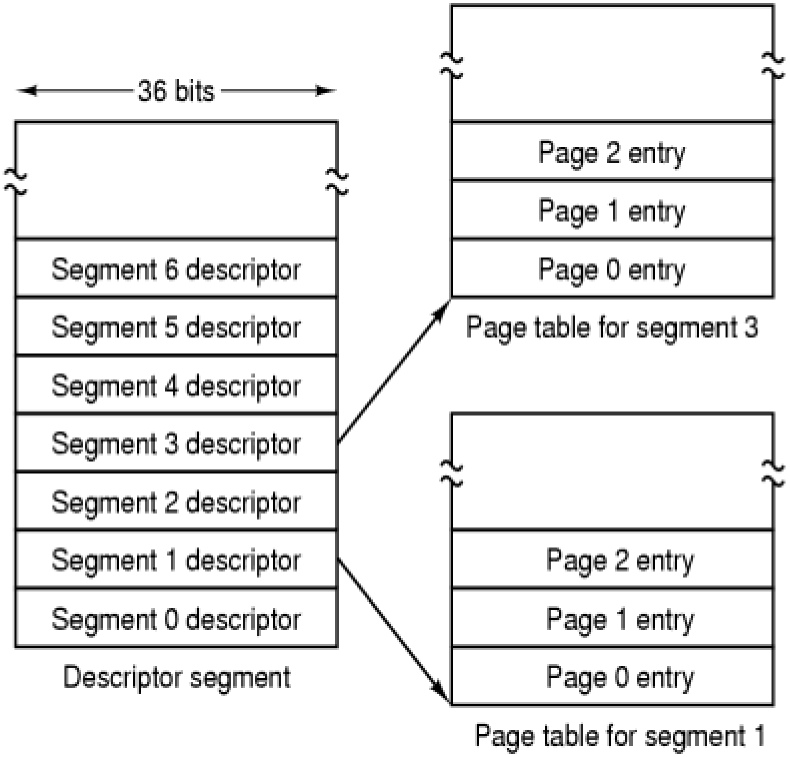
SEGMENTATION

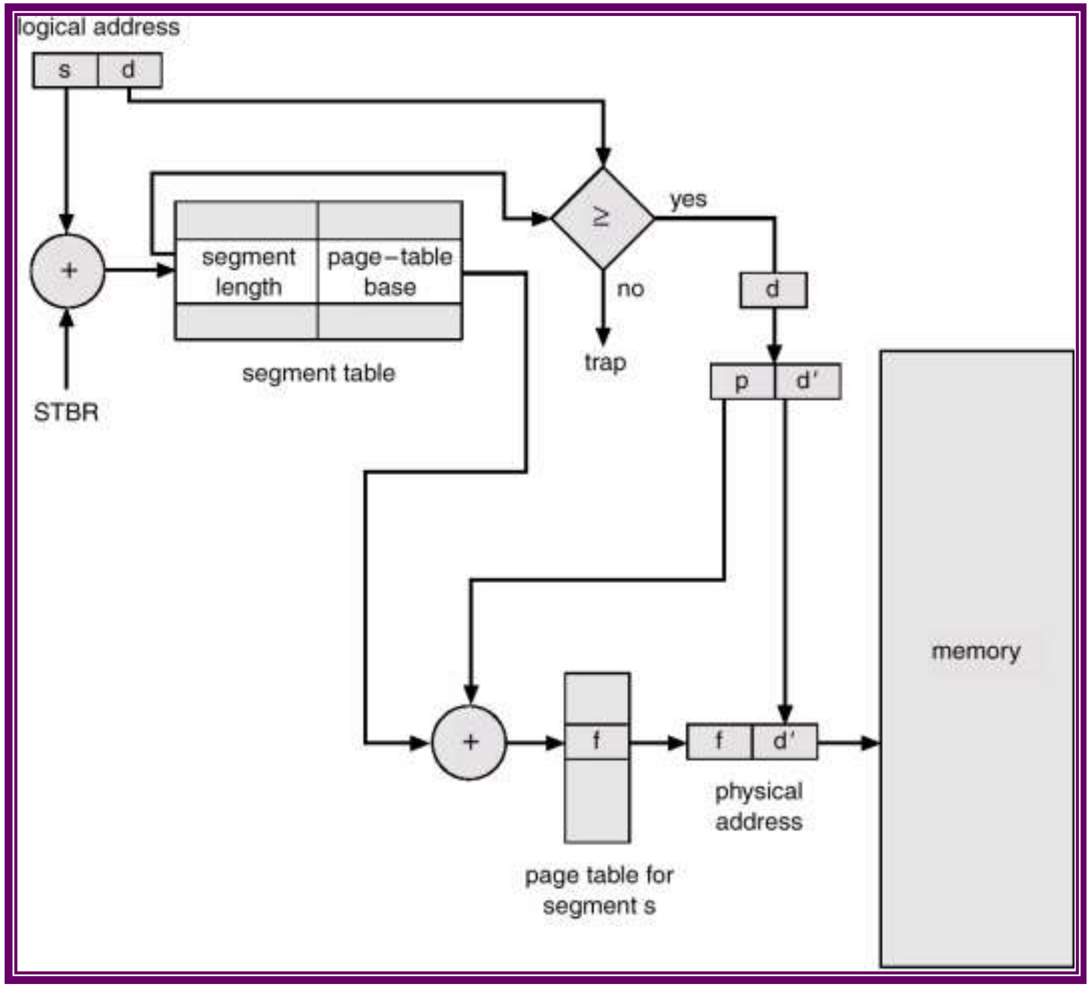


Sharing of Segments

SEGMENTATION AND PAGINATION

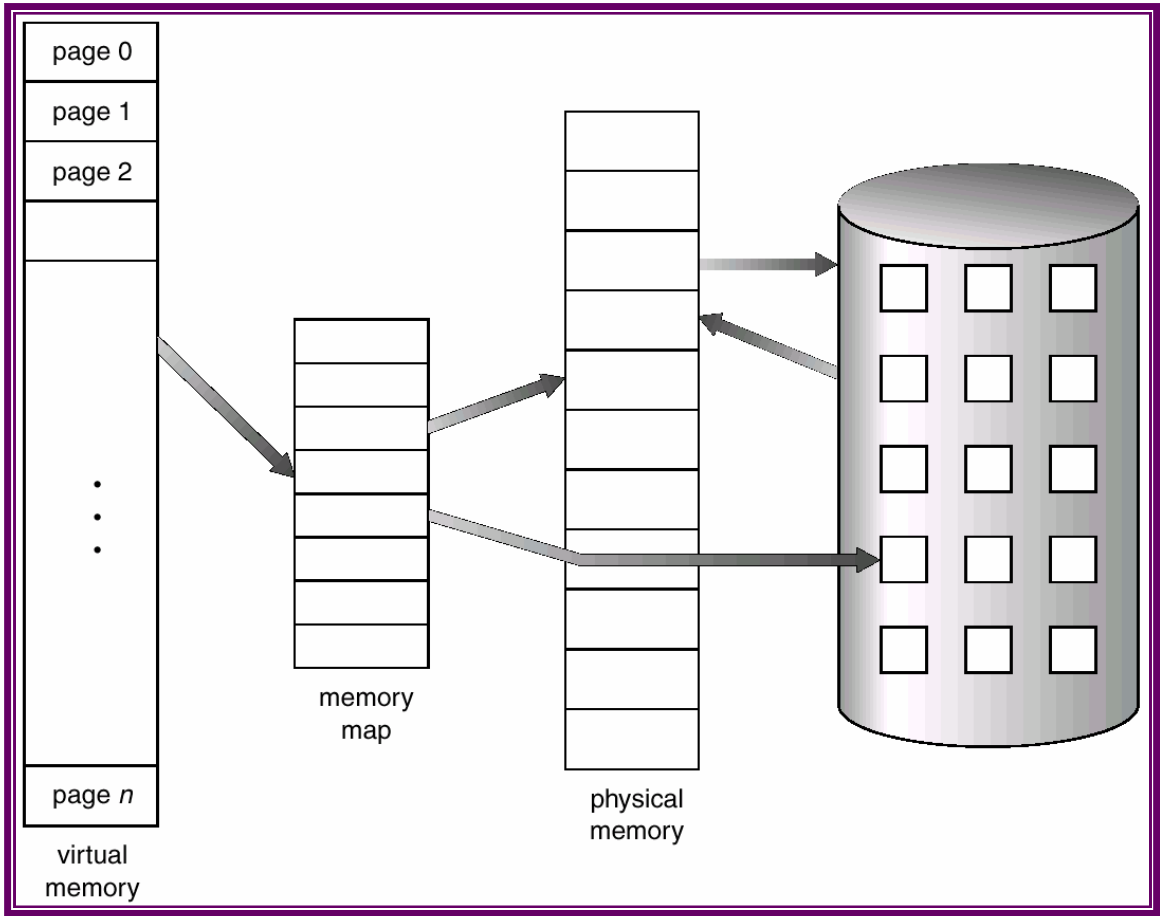
- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.





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 ⇒ 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.

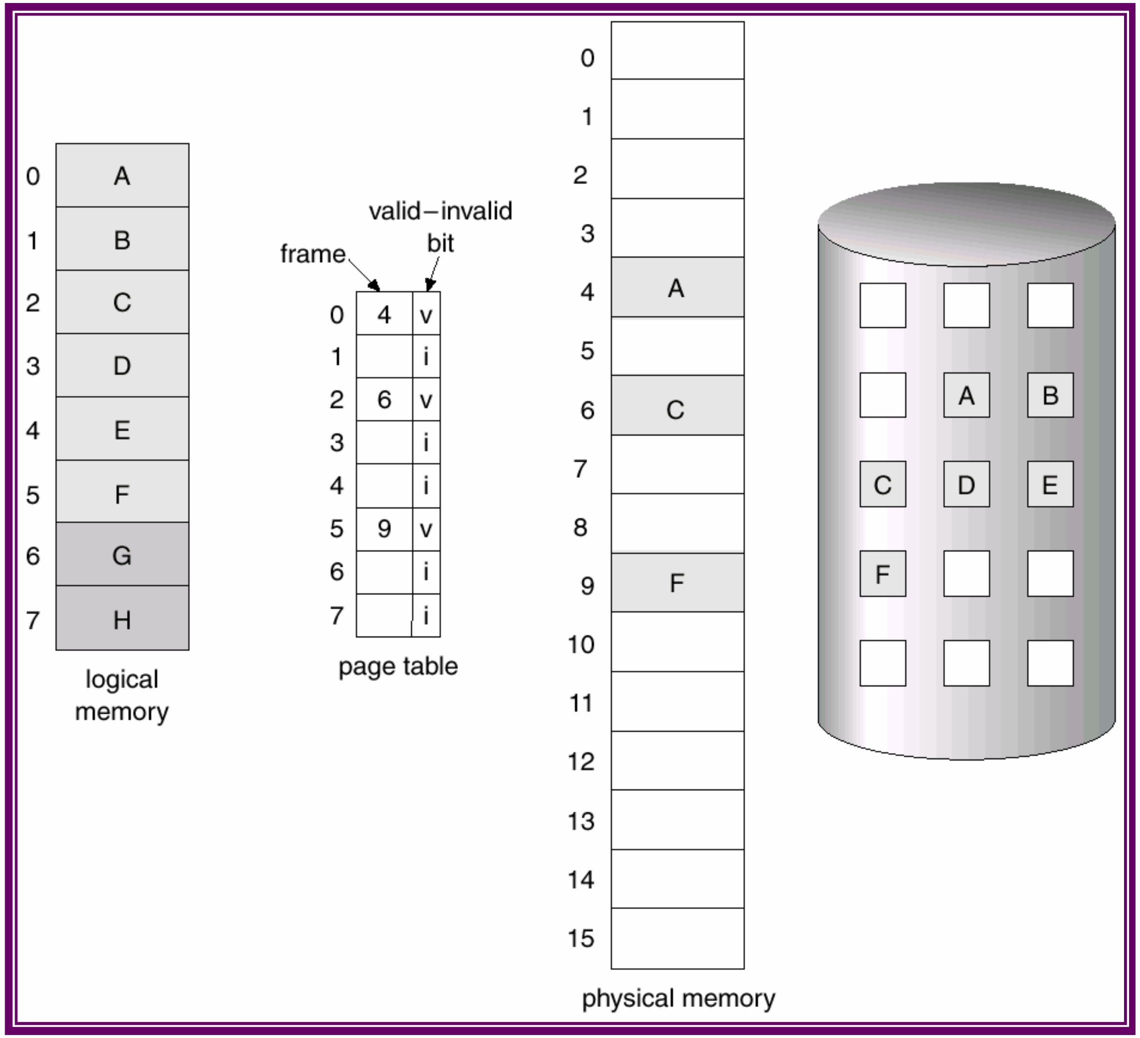
Page #	Invalid Bit
0	1
1	0
2	1
3	0
4	0

Page Table

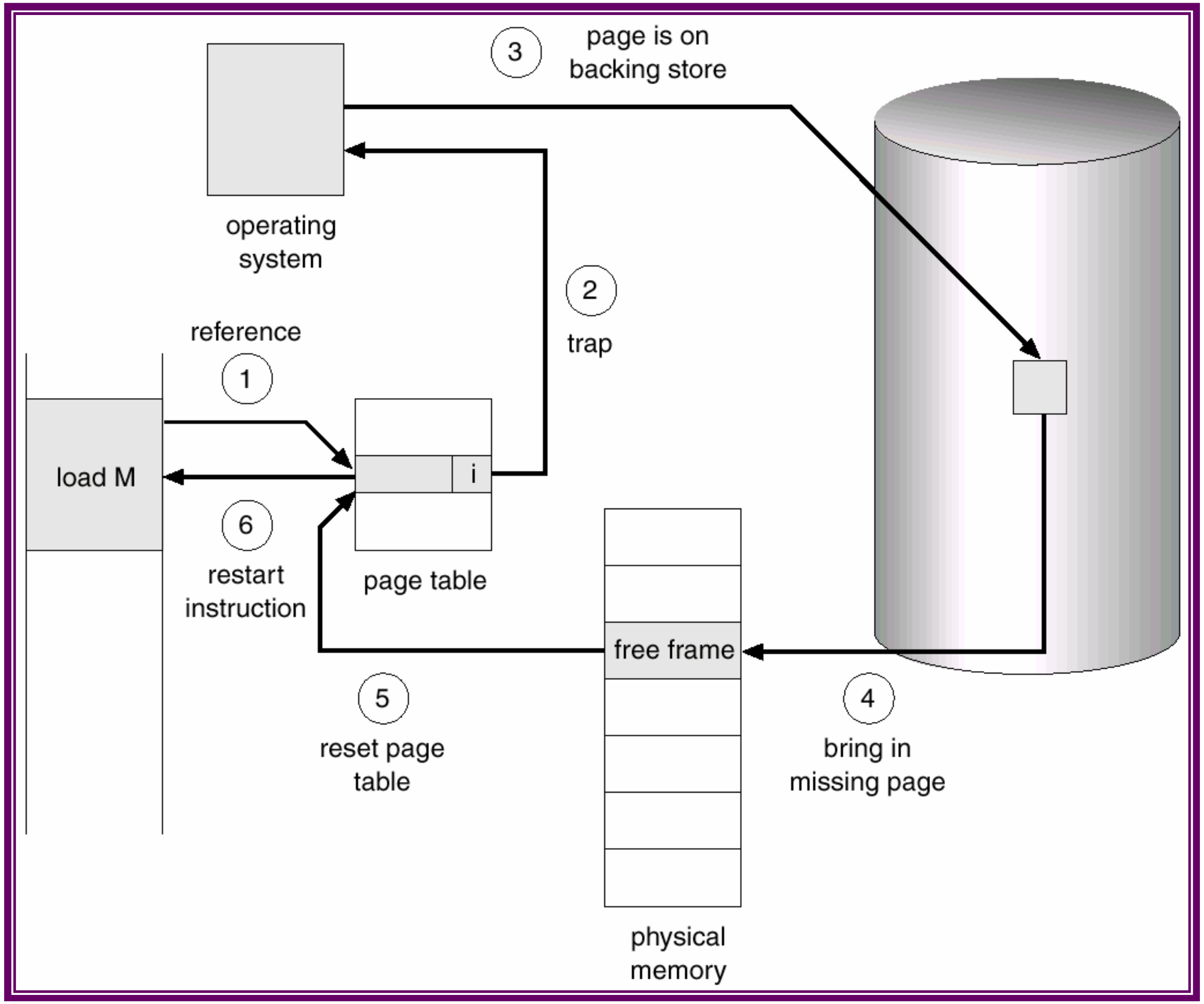
During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

DEMAND PAGING

- 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
  - Block move
  - auto increment/decrement location



DEMAND PAGING

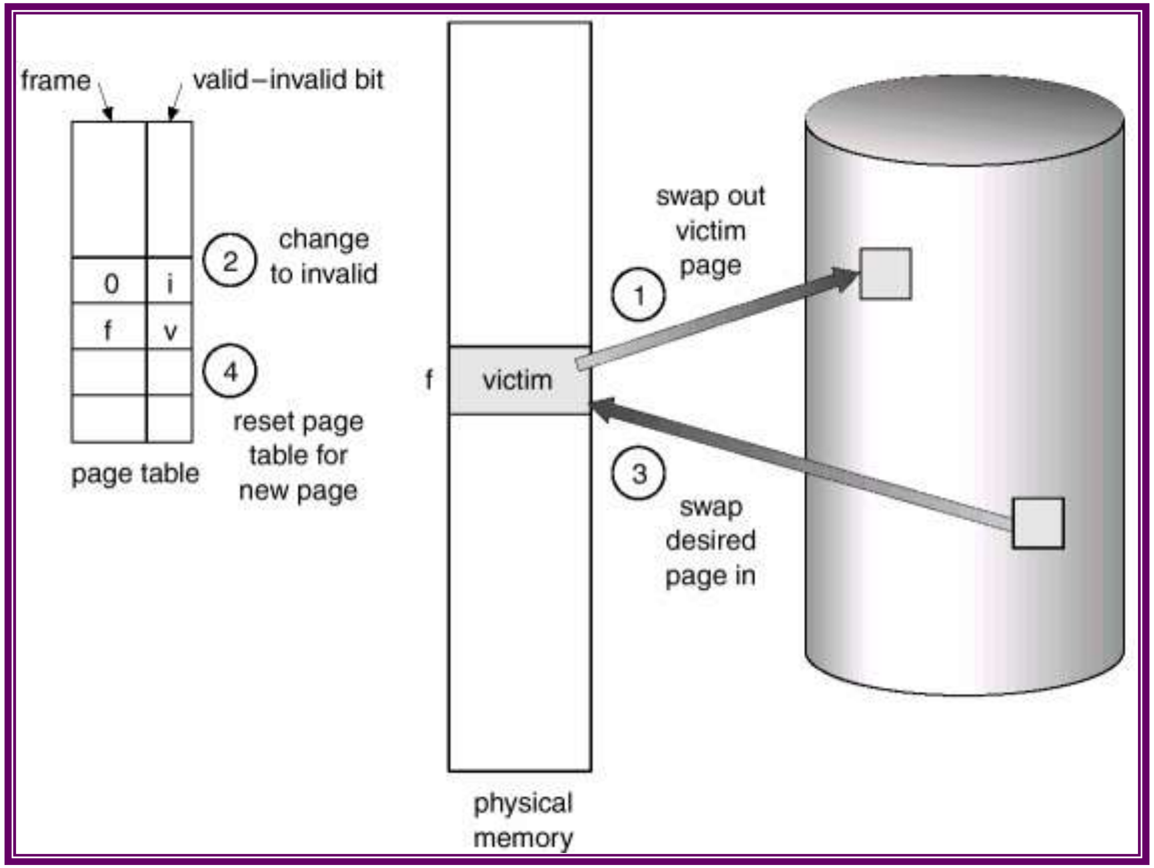


Steps in Handling a Page Fault

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.

TABELLE DEMAND PAGING

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	1	0	1
2	54	4	0	0	1
3	53	0	0	1	1
4	49	1	0	1	1
5	52	6	0	0	1
6	52	4	1	1	1
7	49	3	1	1	1
8	50	4	1	0	1
9	54	3	0	0	1
10	52	0	1	1	1
11	50	6	1	0	1
12	54	1	1	1	1
13	53	4	0	1	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 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.