

Operating System

Memory Management

UNIT-IV

Prepared By

Alok Haldar

Assistant professor

Department of Computer Science & BCA

Kharagpur College



Memory Management

- Background
- Swapping
- Contiguous Allocation
- Paging
- Segmentation
- Segmentation with Paging

Background

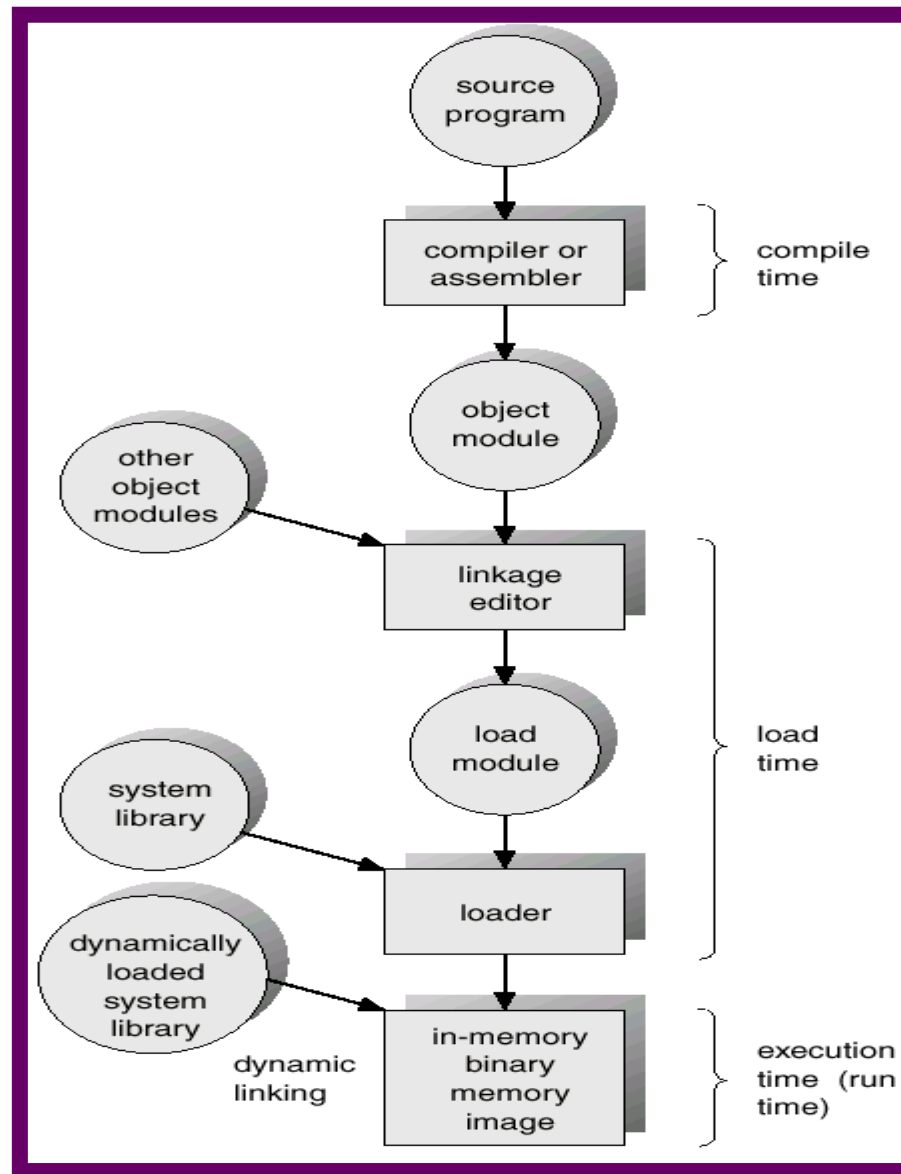
- Program must be brought into memory and placed within a process for it to be run.
- *Input queue* – collection of processes on the disk that are waiting to be brought into memory to run the program.
- User programs go through several steps before being run.

Binding of Instructions and Data to Memory

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 if memory location is not known at compile time.
- **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*).

Multistep Processing of a User Program



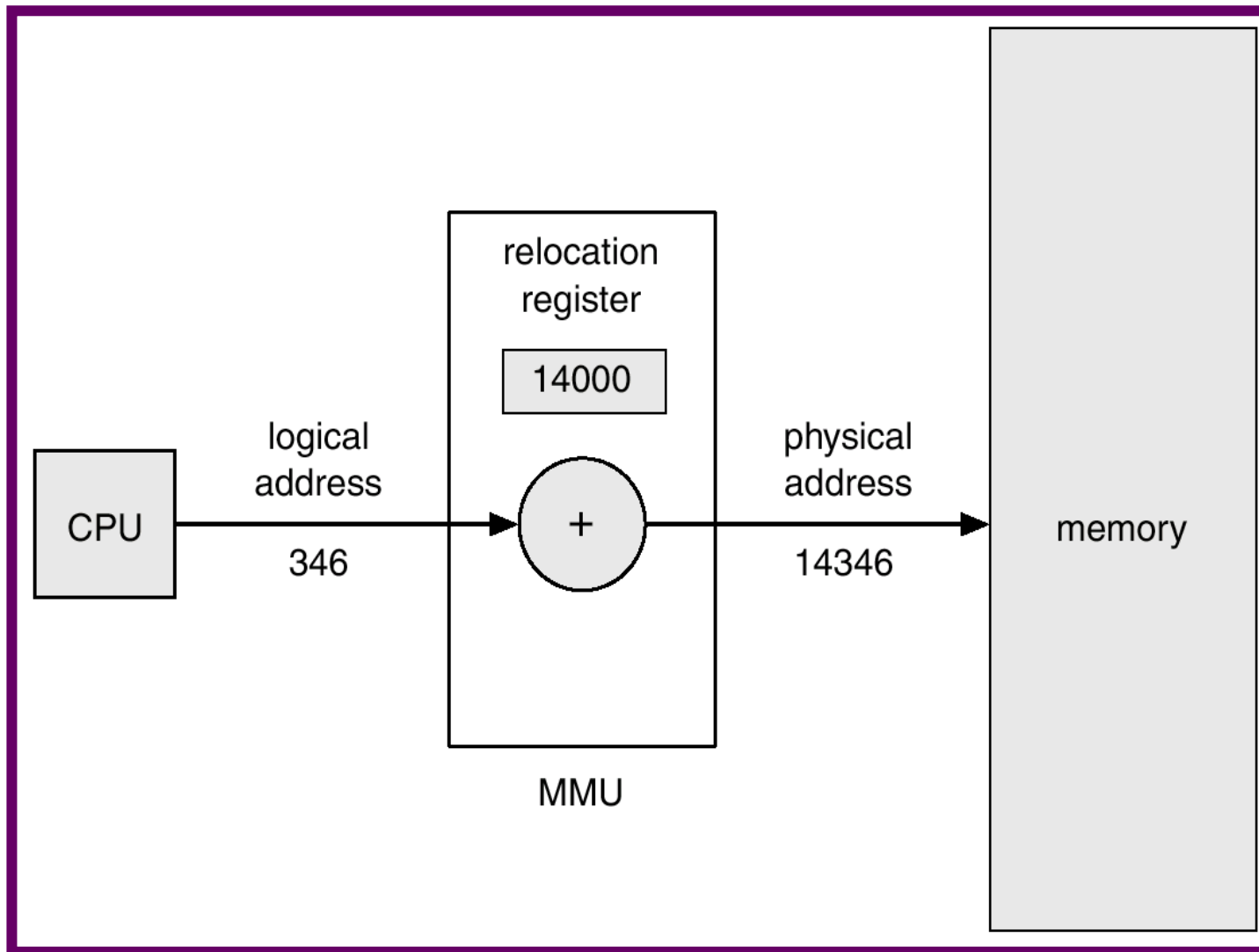
Logical vs. Physical Address Space

- The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management.
 - › *Logical address* – generated by the CPU; also referred to as *virtual address*.
 - › *Physical address* – address seen by the memory unit.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

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 relocation using a relocation register



Dynamic Loading

- 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.

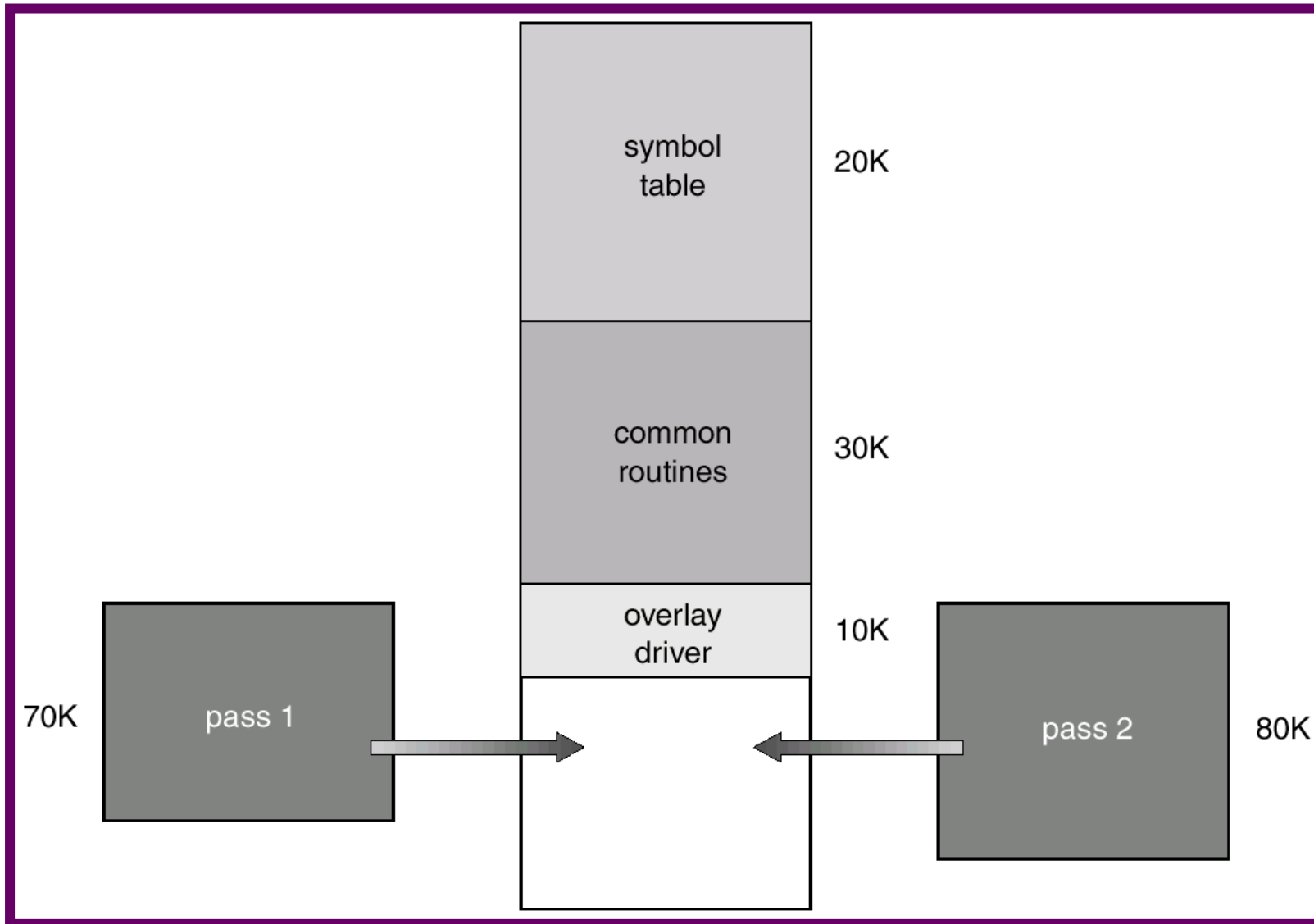
Dynamic Linking

- 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.

Overlays

- 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

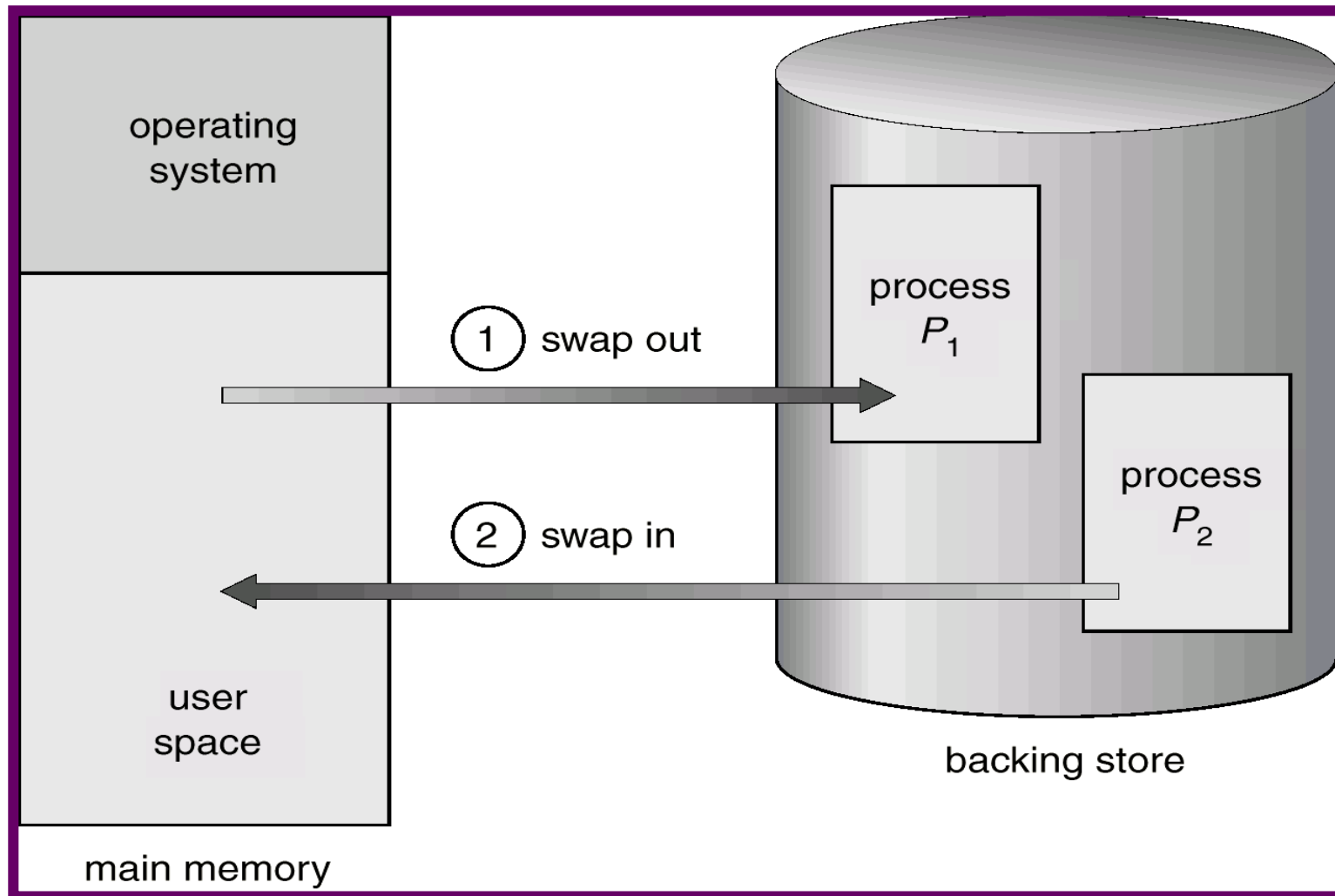
Overlays for a Two-Pass Assembler



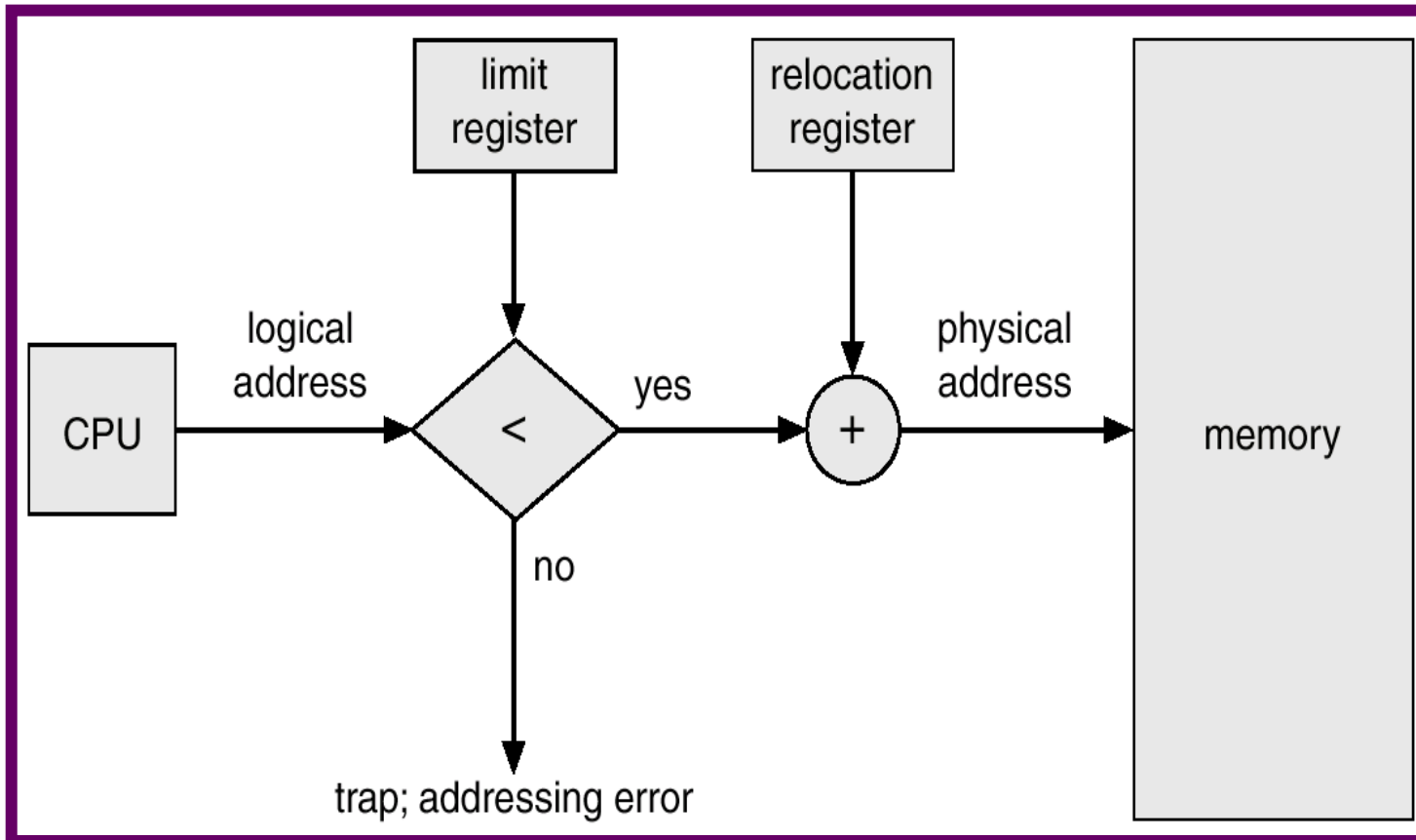
Swapping

- 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.

Schematic View of Swapping

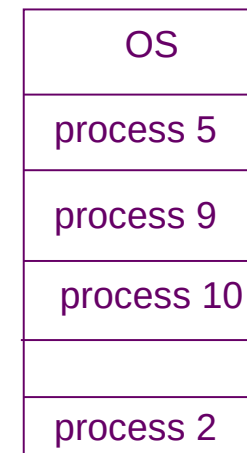
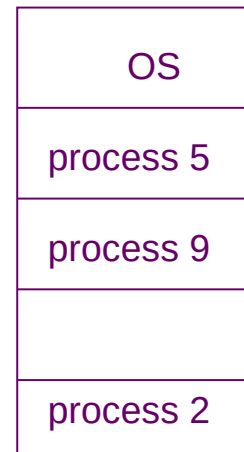
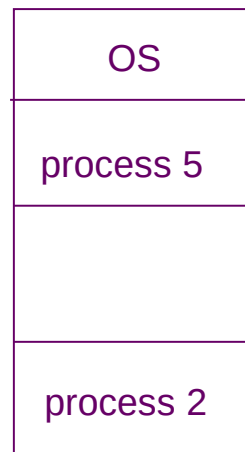
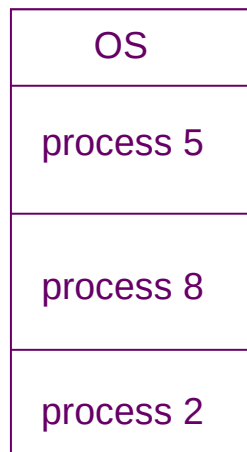


Hardware Support for Relocation and Limit Registers



Contiguous Allocation (Cont.)

- Multiple-partition allocation
 - › *Hole* – block of available memory; holes 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 partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size n from a list of free holes.

- **First-fit:** Allocate the *first* hole that is big enough.
- **Best-fit:** Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- **Worst-fit:** Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- 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.

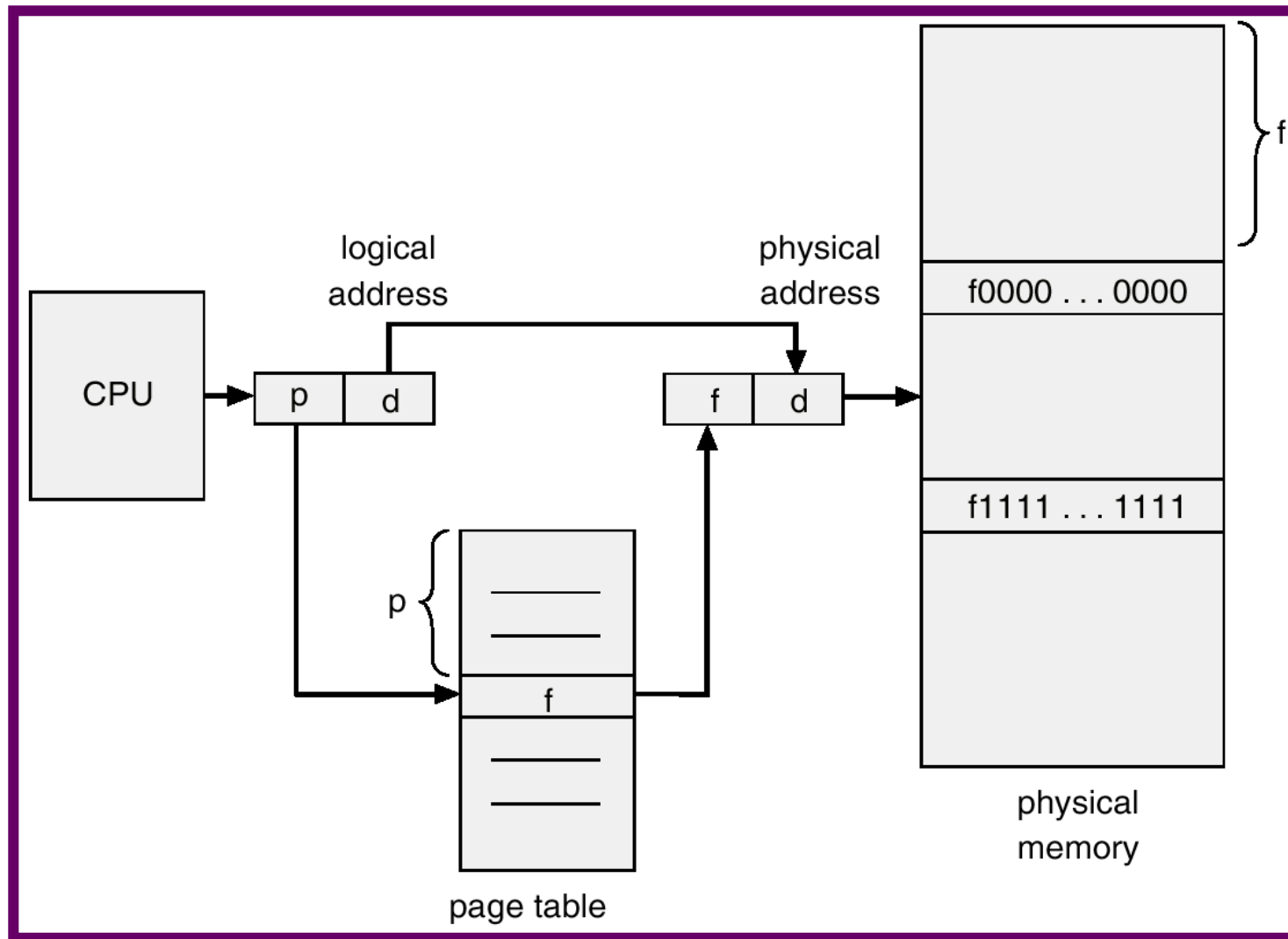
Paging

- Logical address space of a process can be noncontiguous; process is allocated 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.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation.

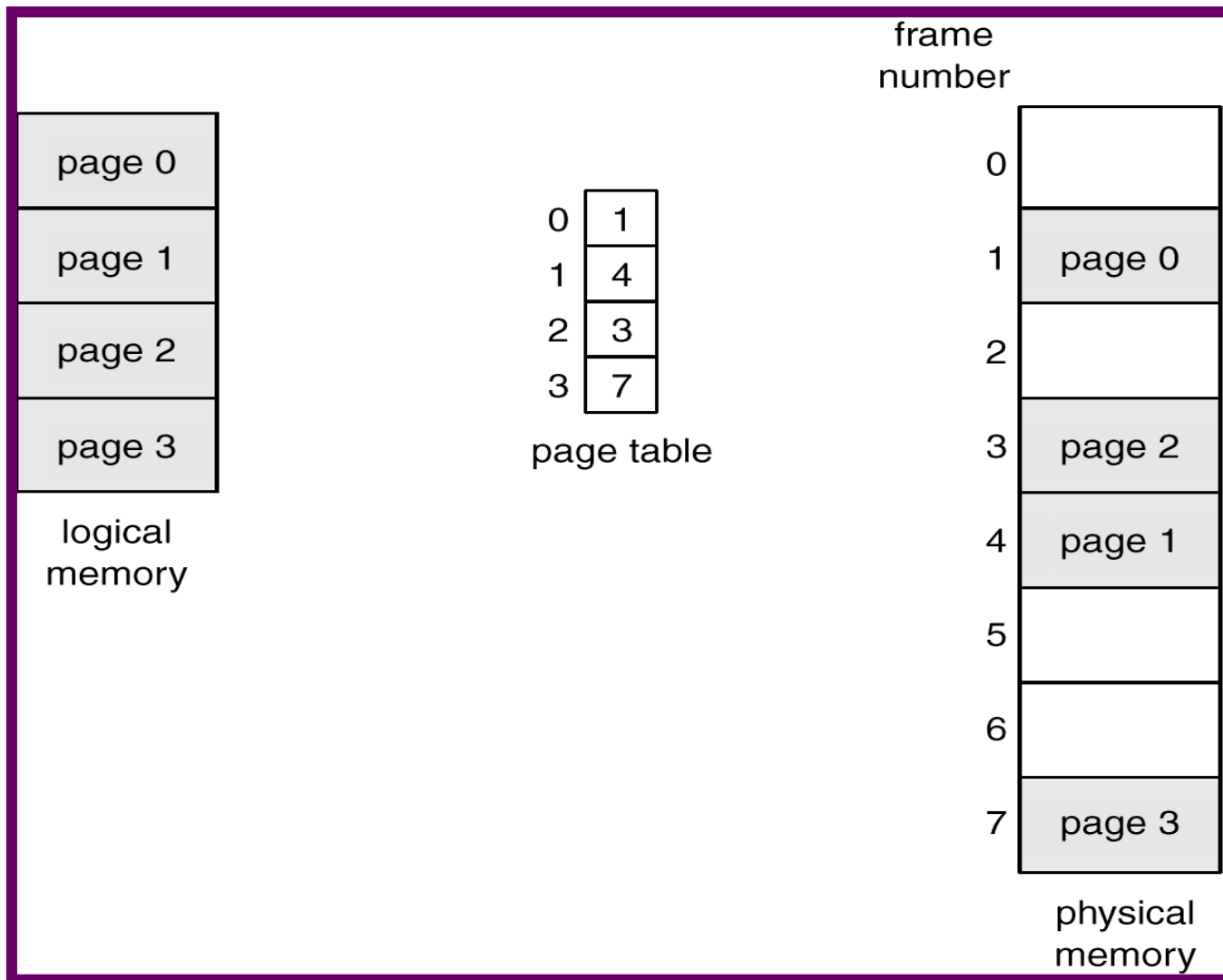
Address Translation Scheme

- 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.

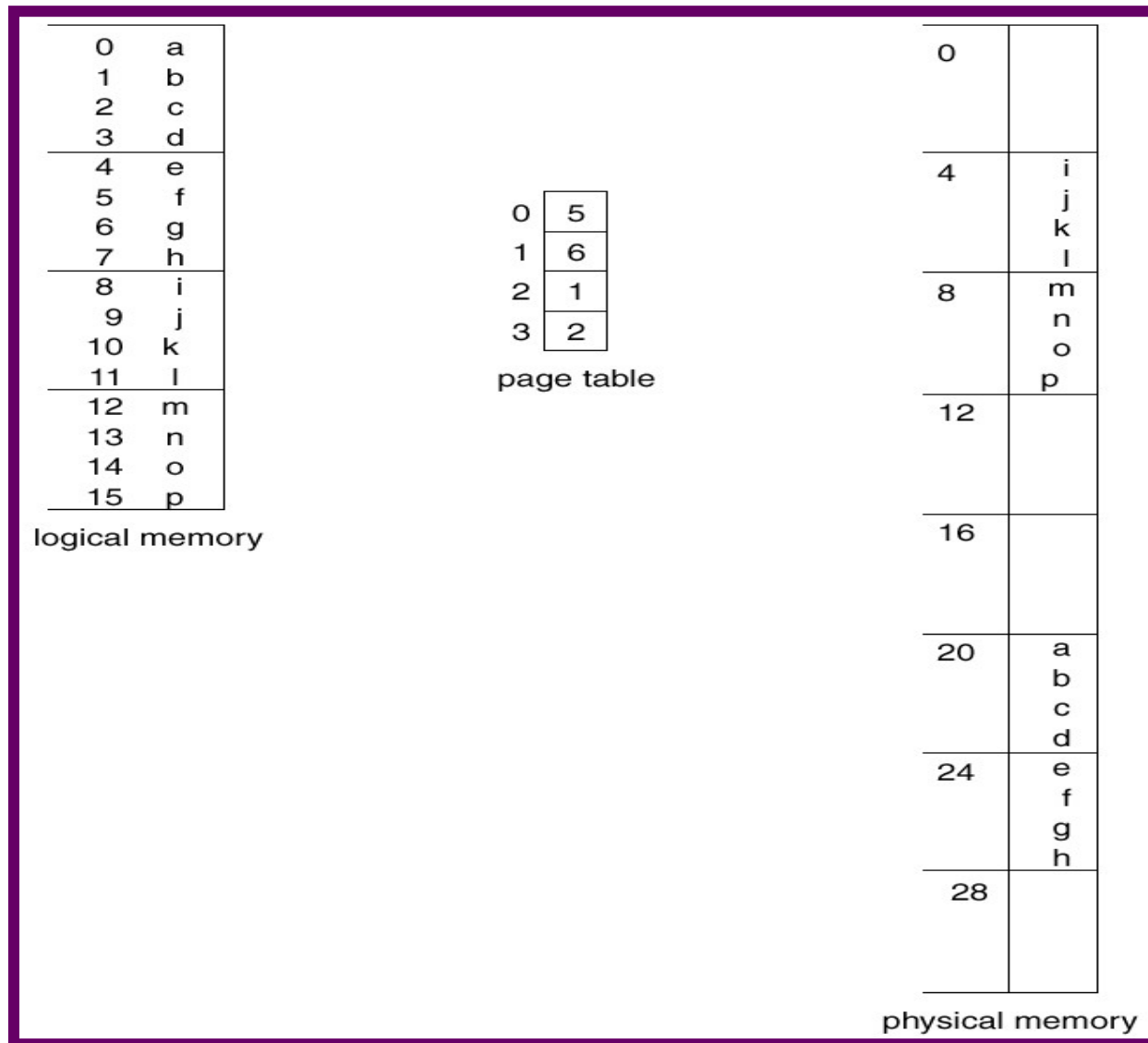
Address Translation Architecture



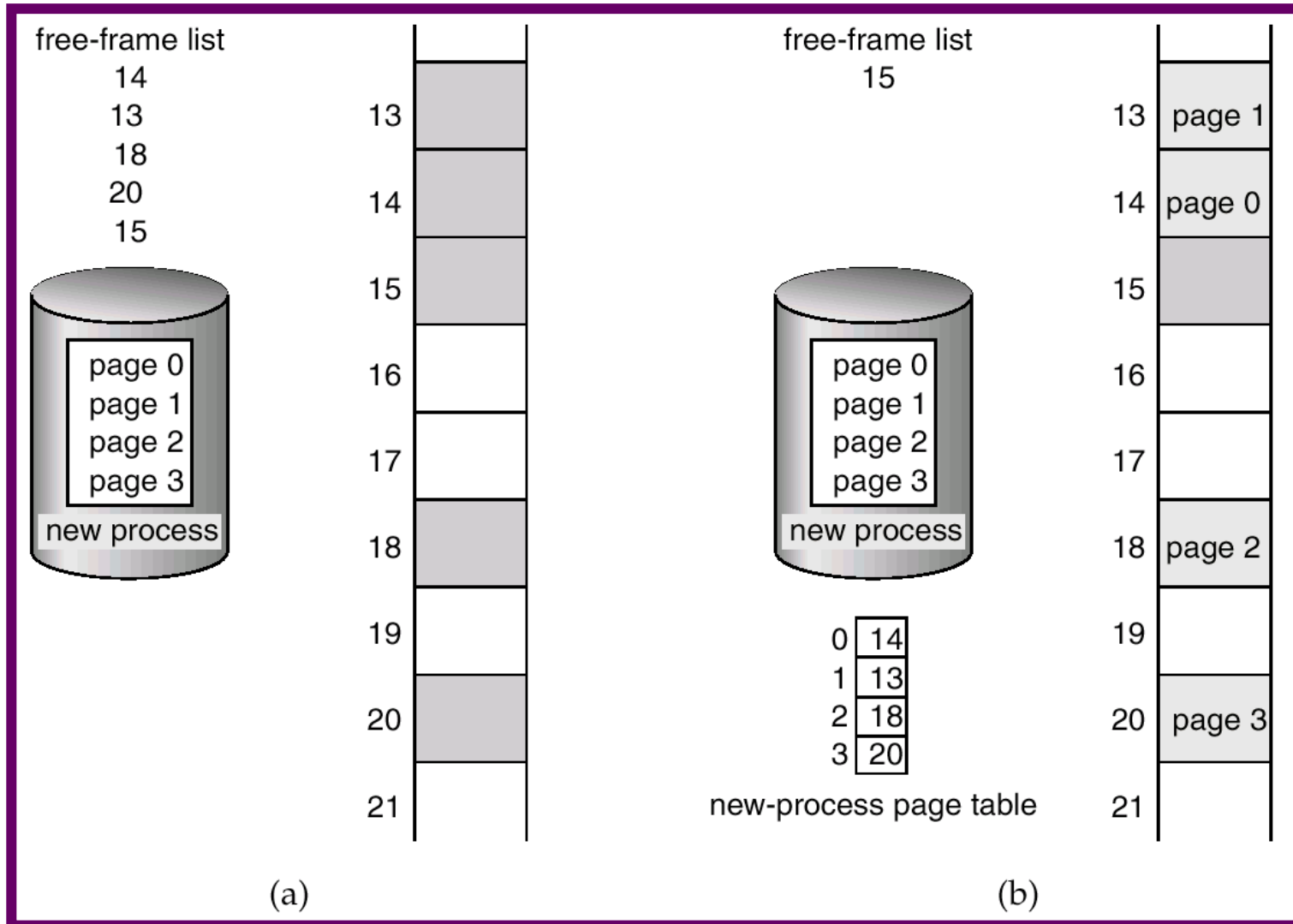
Paging Example



Paging Example



Free Frames



Implementation of Page Table

- 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)*

Associative Memory

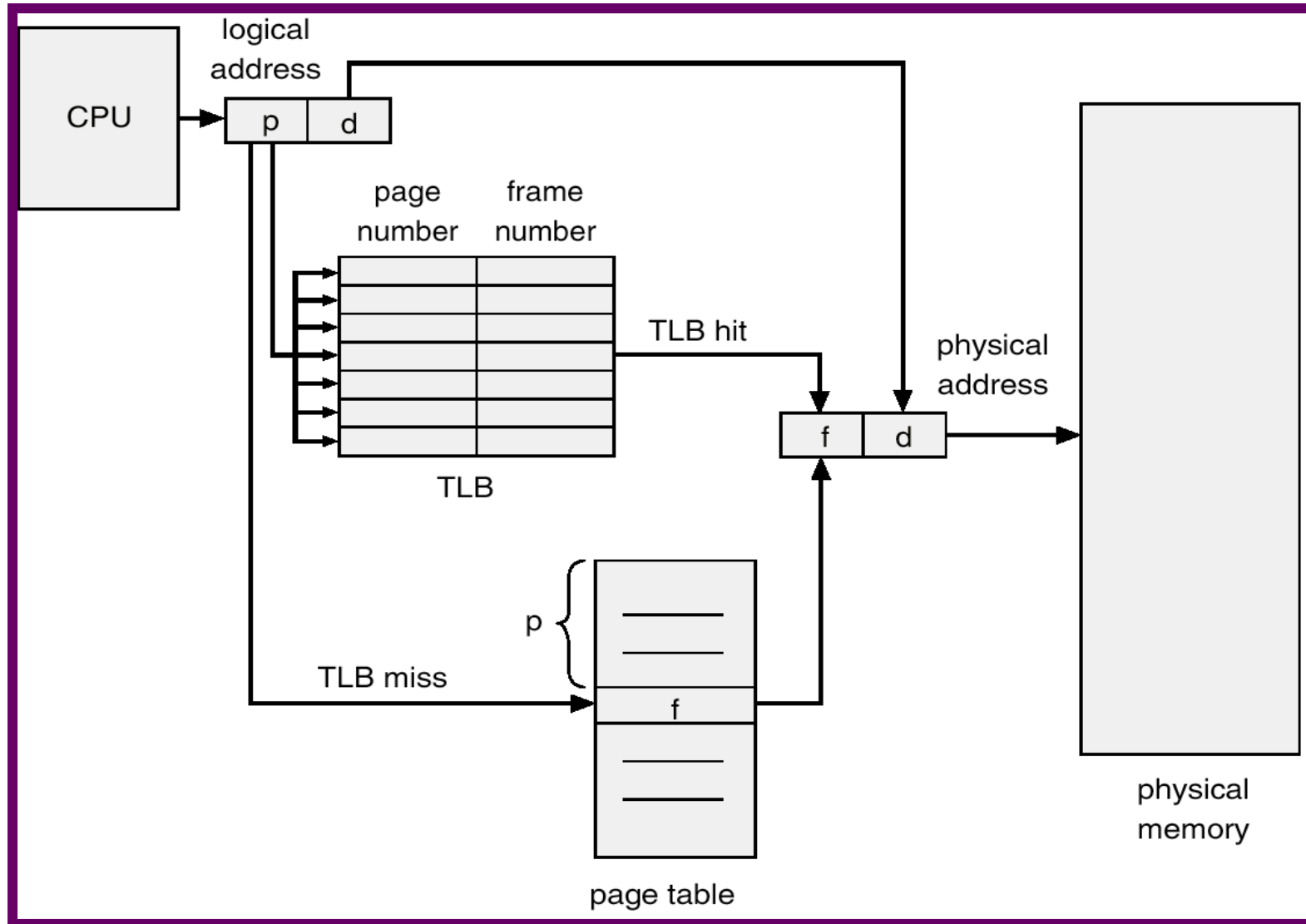
- Associative memory – parallel search

Page #	Frame #

Address translation (A' , A'')

- ◆ If A' is in associative register, get frame # out.
- Otherwise get frame # from page table in memory

Paging Hardware With TLB



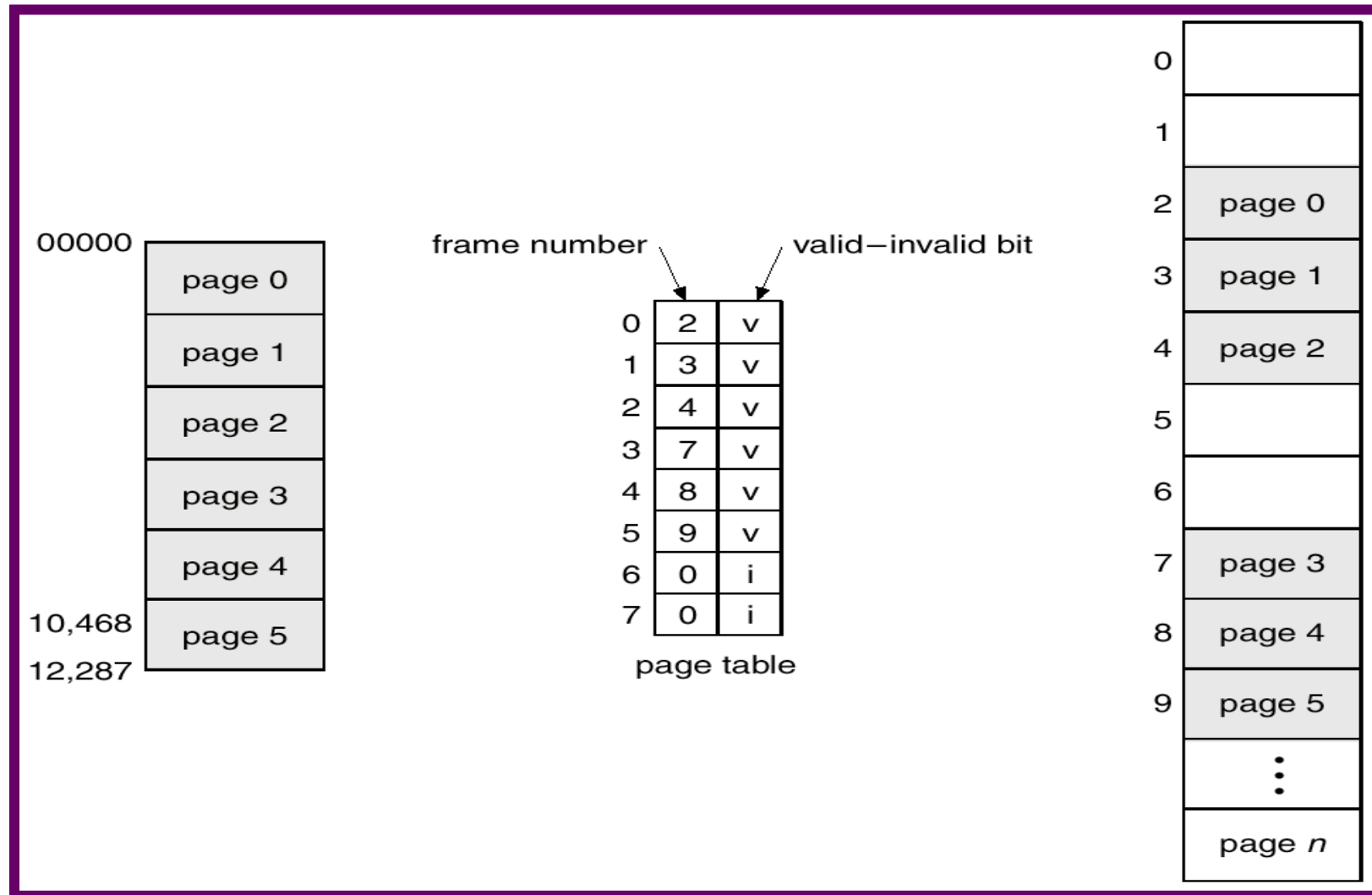
Effective Access Time

- Associative Lookup = $\frac{1}{n}$ time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; rations related to number of associative registers.
- Hit ratio = \rightarrow
- Effective Access Time (EAT)
$$\text{EAT} = (1 + \frac{1}{n}) \rightarrow + (2 + \frac{1}{n})(1 - \rightarrow)$$
$$= 2 + \frac{1}{n} - \rightarrow$$

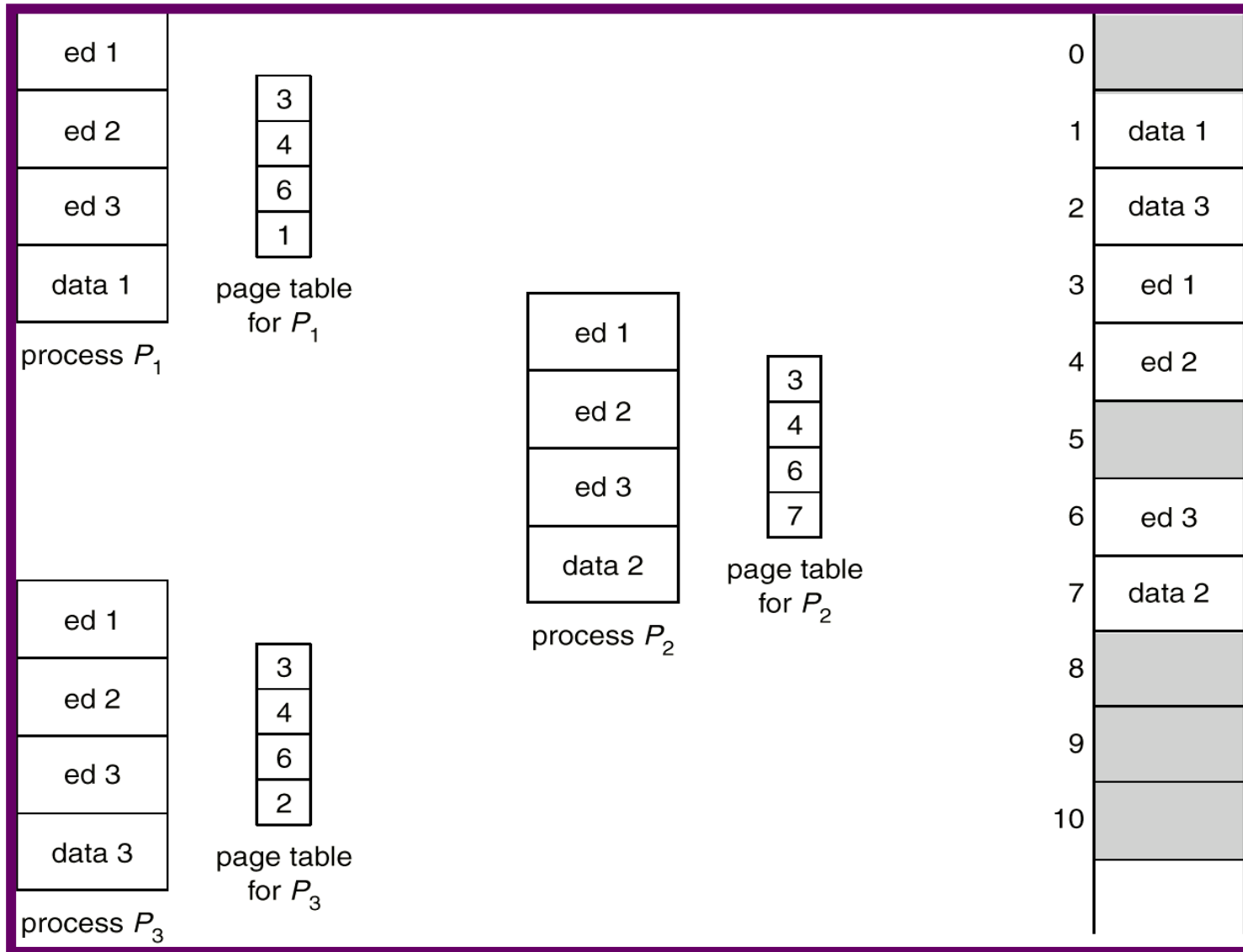
Memory Protection

- Memory protection implemented by associating protection bit with each frame.
- *Valid-invalid* bit attached to each entry in the page table:
 - } “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.
 - } “invalid” indicates that the page is not in the process’ logical address space.

Valid (v) or Invalid (i) Bit In A Page Table



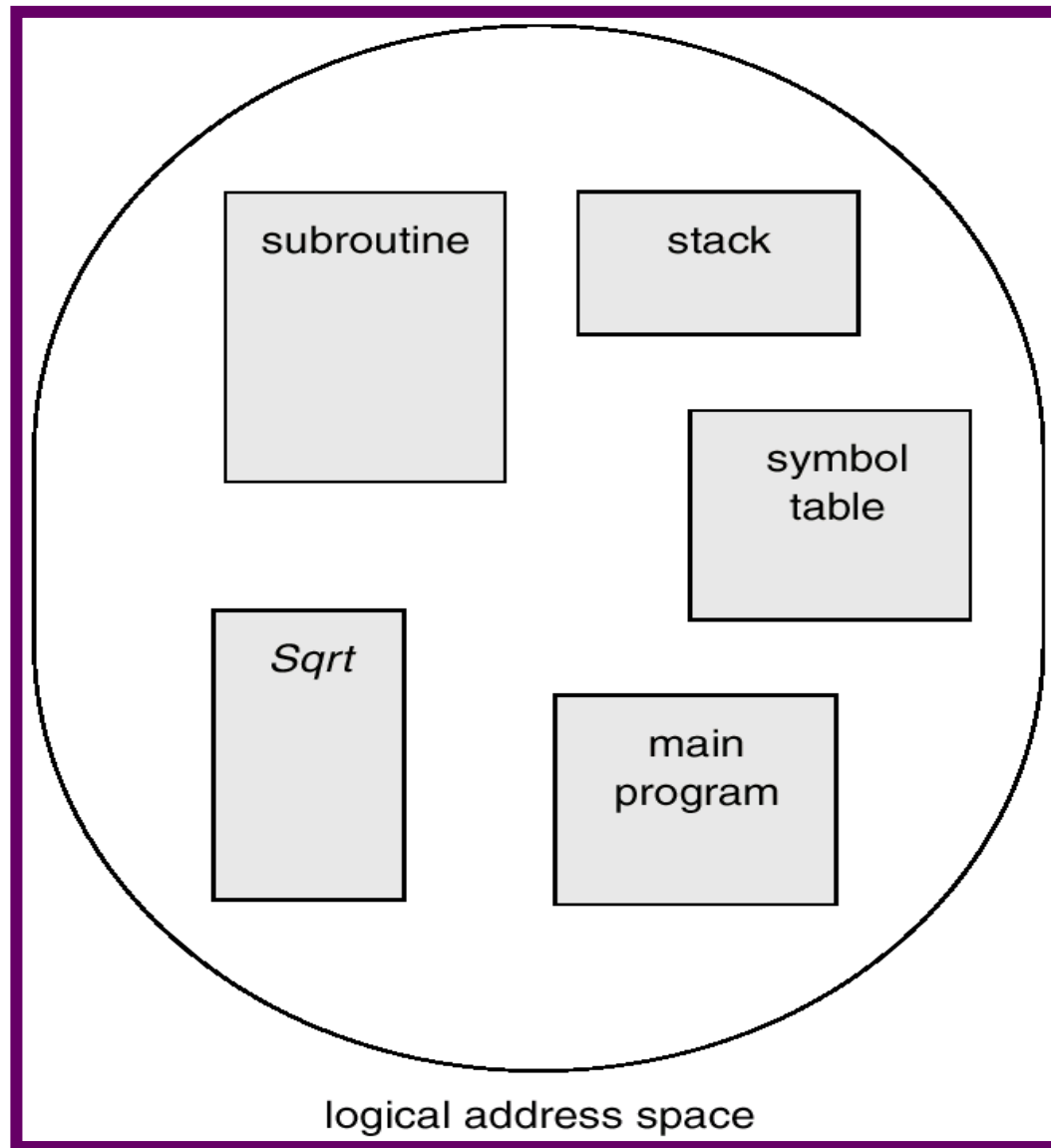
Shared Pages Example



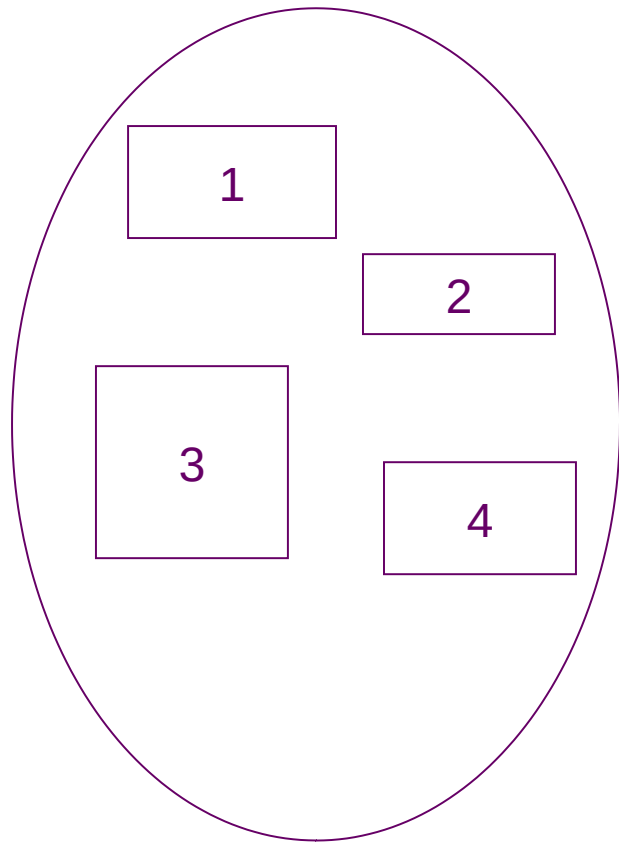
Segmentation

- 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

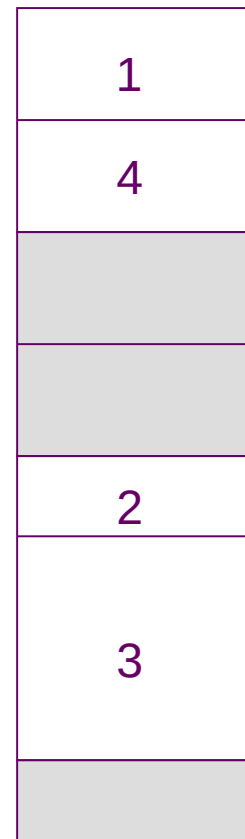
User's View of a Program



Logical View of Segmentation



user space



physical memory space

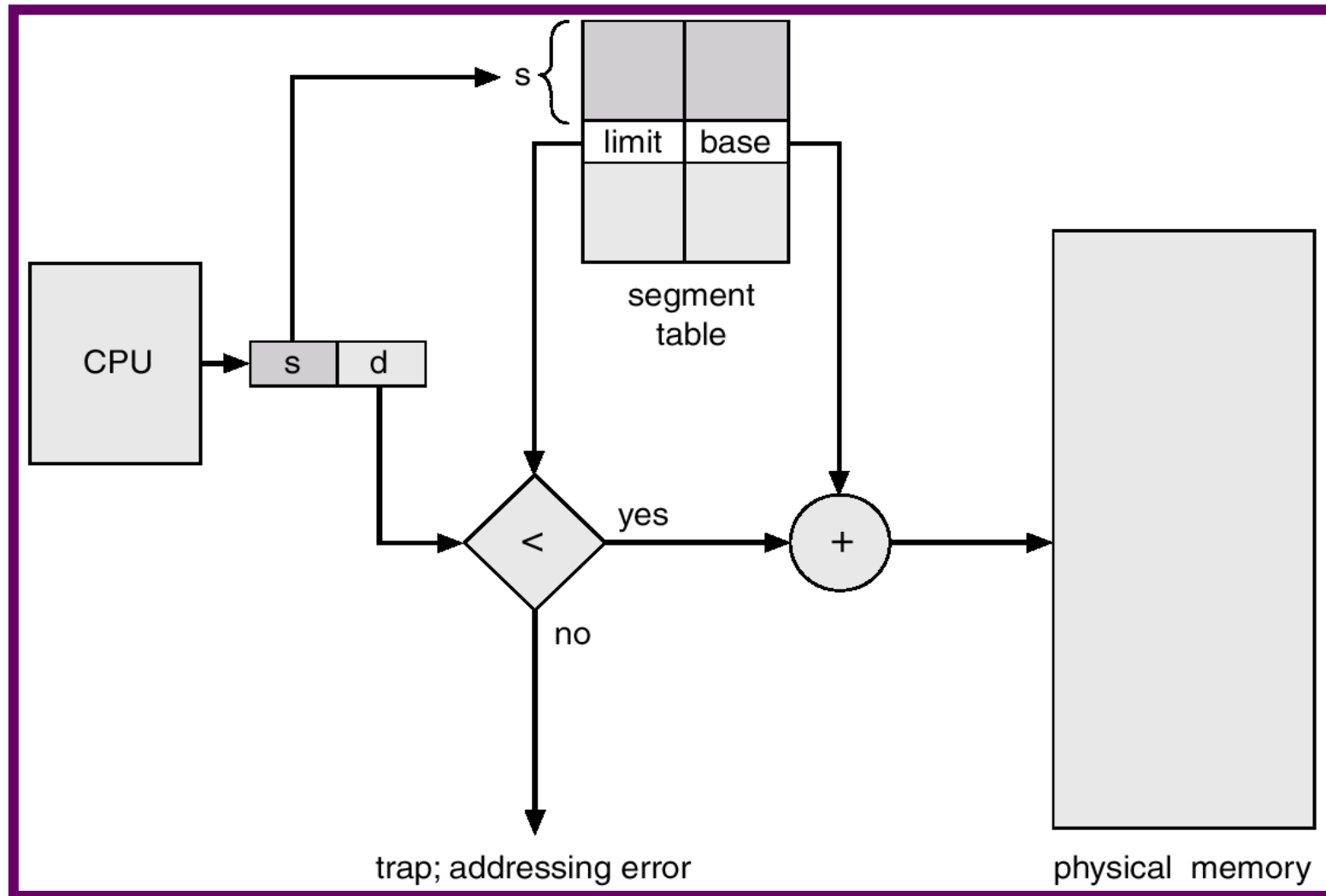
Segmentation Architecture

- 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 $s < \text{STLR}$.

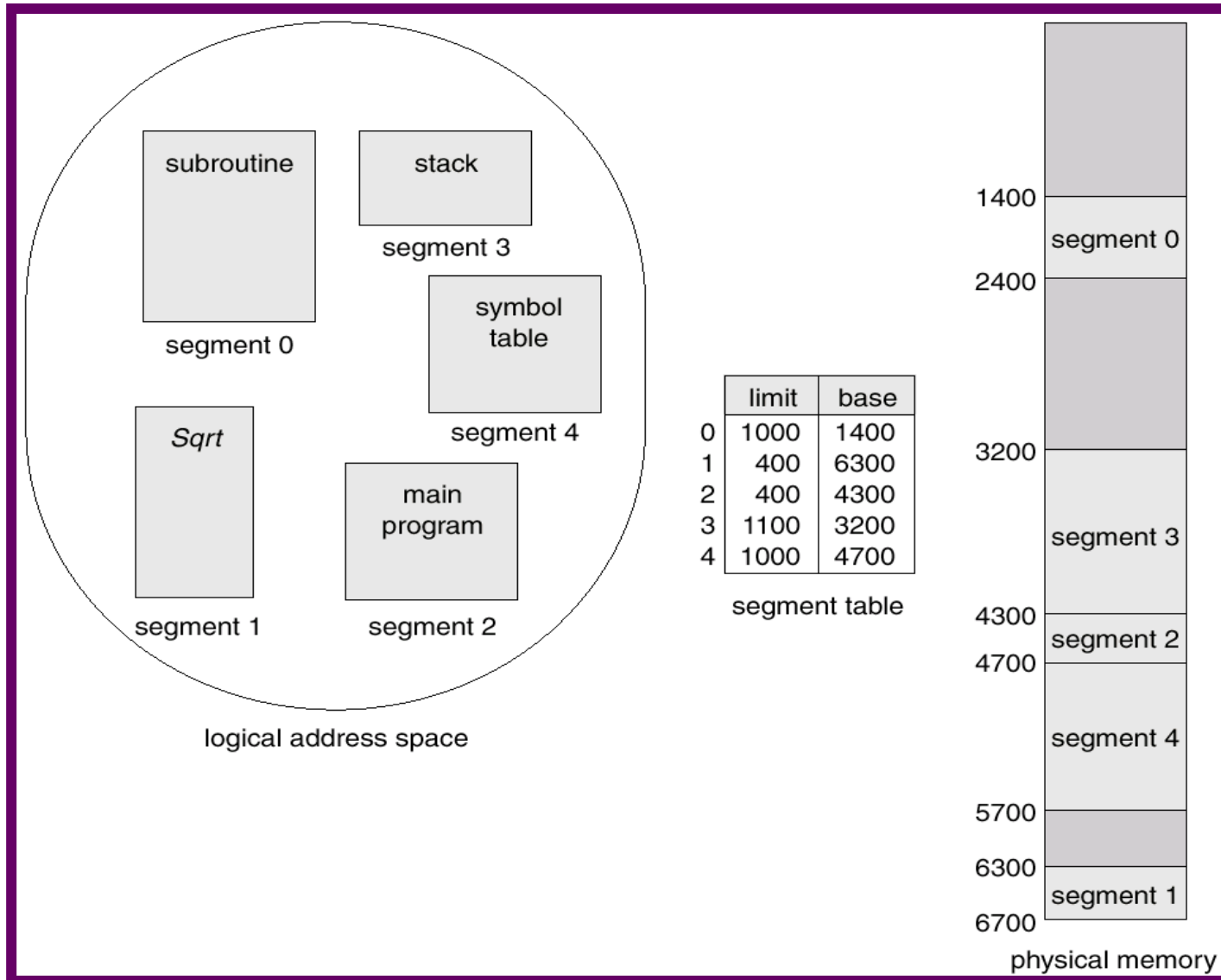
Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
 - } validation bit = 0 \blacktriangle illegal segment
 - } read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.
- A segmentation example is shown in the following diagram

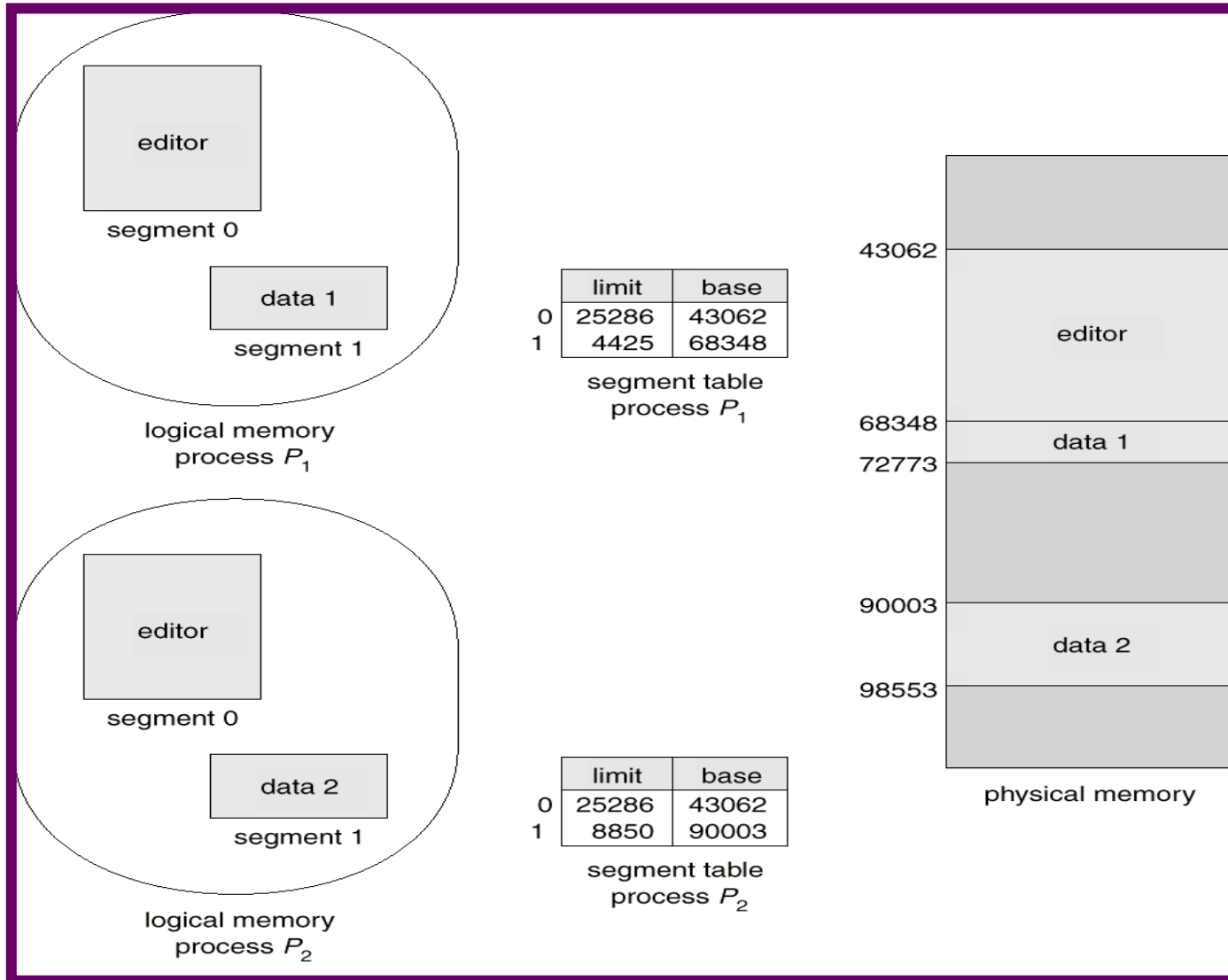
Segmentation Hardware



Example of Segmentation



Sharing of Segments



Paging VS Segmentation

1	Non-Contiguous memory allocation	Non-contiguous memory allocation
2	Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
3	OS is responsible	Compiler is responsible.
4	Paging is faster than segmentation	Segmentation is slower than paging
5	Paging is closer to Operating System	Segmentation is closer to User
6	It suffers from internal fragmentation	It suffers from external fragmentation
7	There is no external fragmentation	There is no external fragmentation
8	Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
9	Page table is used to maintain the page information.	Segment Table maintains the segment information
10	Page table entry has the frame number and some flag bits to represent details about pages.	Segment table entry has the base address of the segment and some protection bits for the segments.