The picture can't be displayed.

Chapter 8: Memory Management

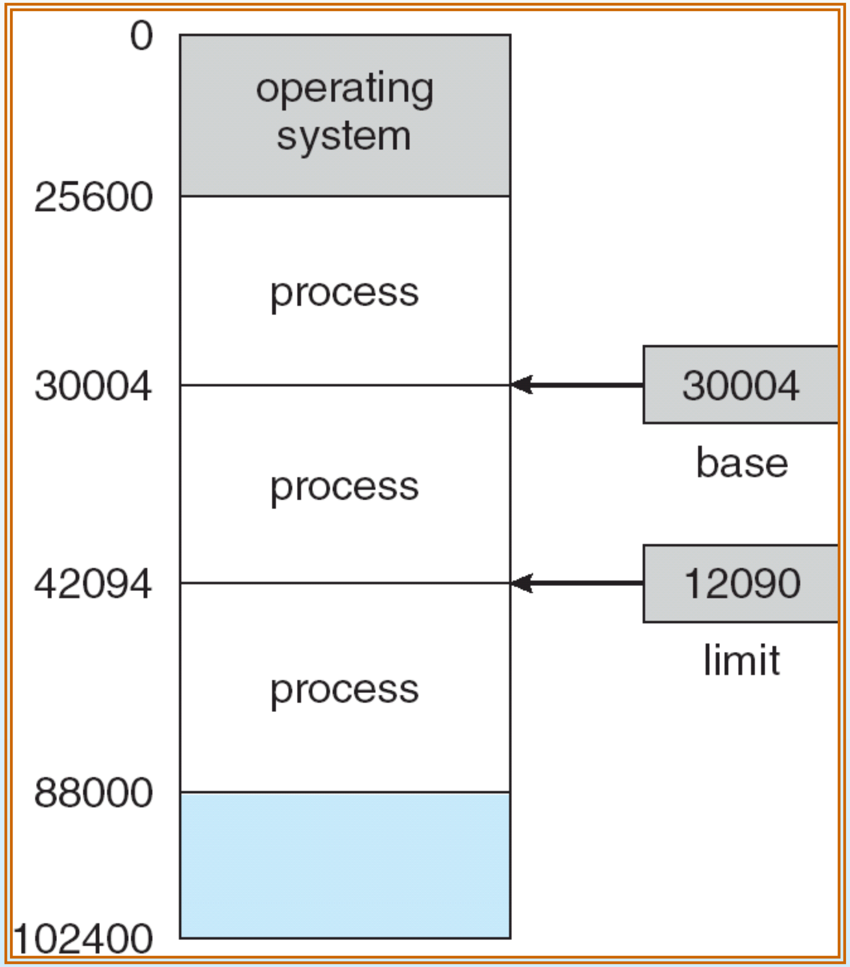
Chapter 8: Memory Management

- Background
- Swapping
- Contiguous Allocation
- Paging
- Segmentation

Background

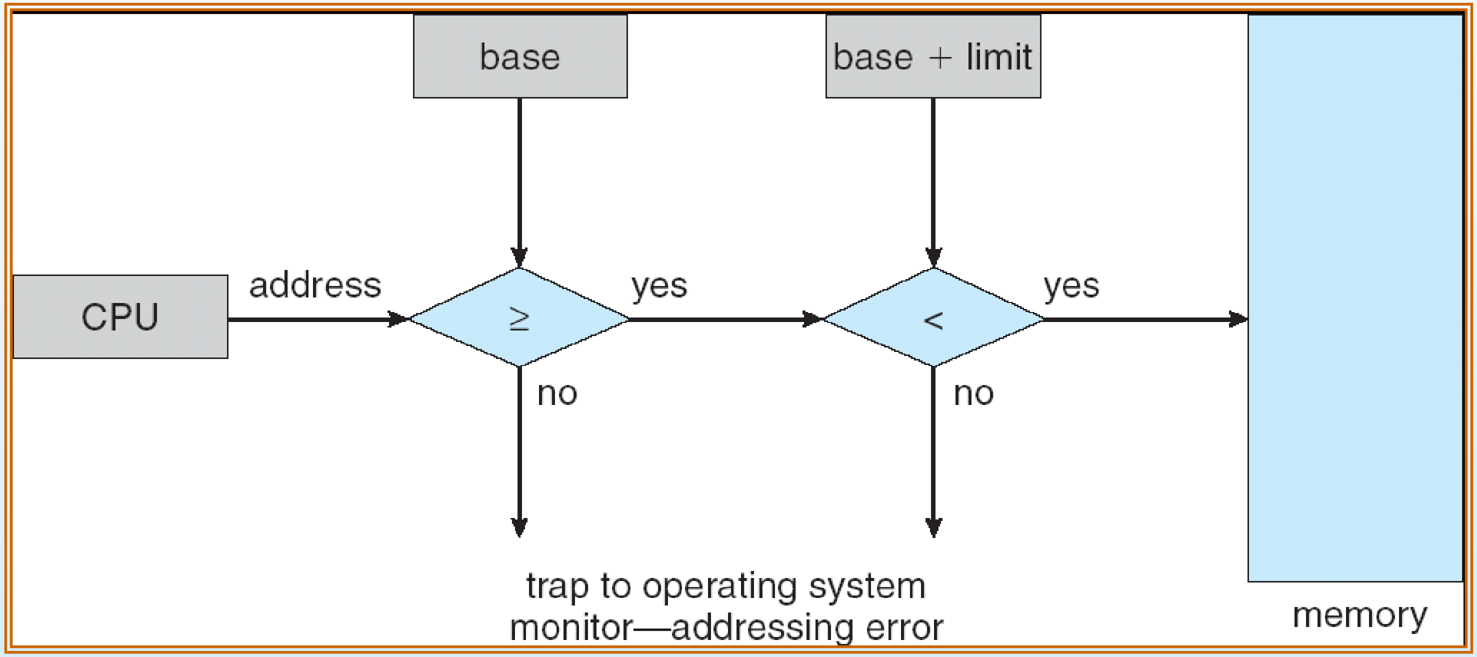
- Program must be **brought into memory** and placed within a process for it to be run
- **Input queue** – collection of processes on the disk that are waiting to be brought into memory to run the program
- Each **process** has a **separate address space**
  - **Base register**: smallest legal physical memory address
  - **Limit register**: size of the range
- Memory Background
  - Stream of memory address.
  - They are used for instruction or data.

A base and a limit register define a logical address space



If the base register holds 30004 and limit register 12090, the program can legally access all address from 30004 through 42094 (inclusive)

HW address protection with base and limit registers



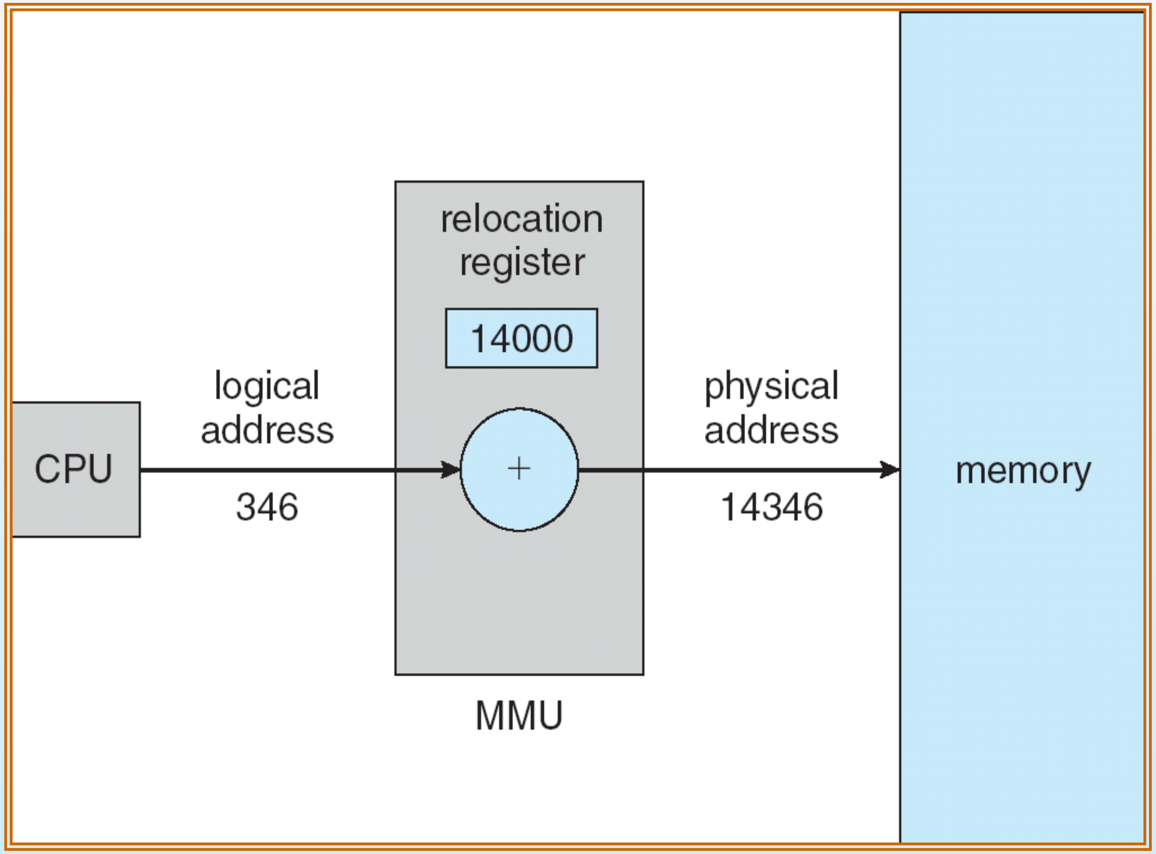
Logical vs. Physical Address Space

- 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

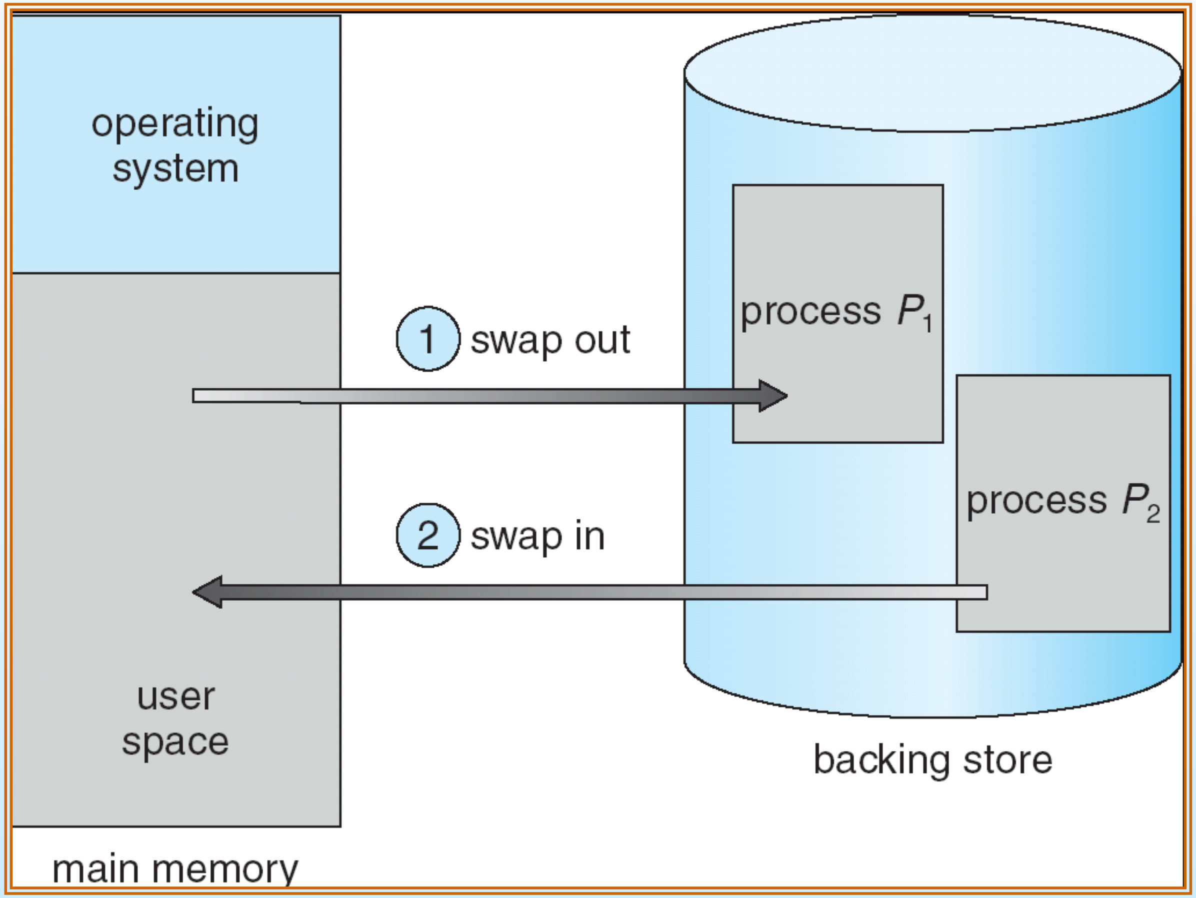
Dynamic relocation using a relocation register



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)

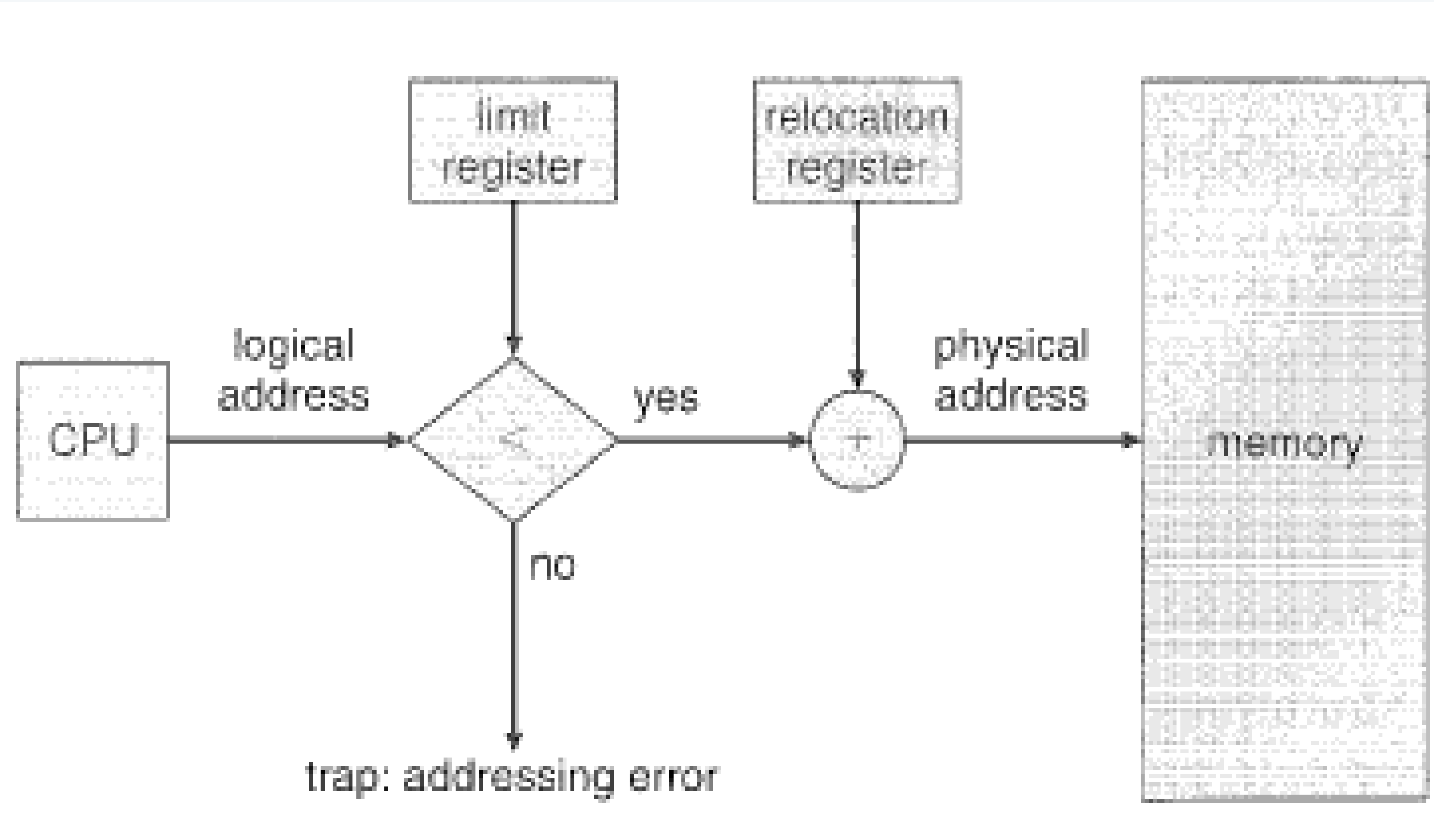
Schematic View of Swapping



Contiguous Allocation

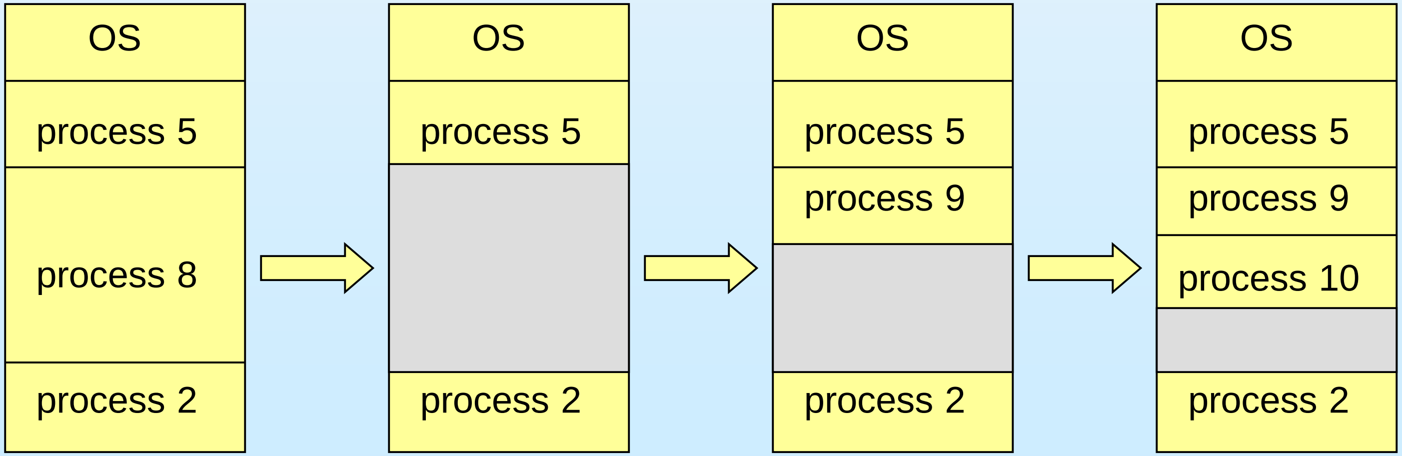
- Main memory usually into **two partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
- Memory mapping and protection
  - **Relocation-register** scheme used to protect user processes from each other, and from changing 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

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

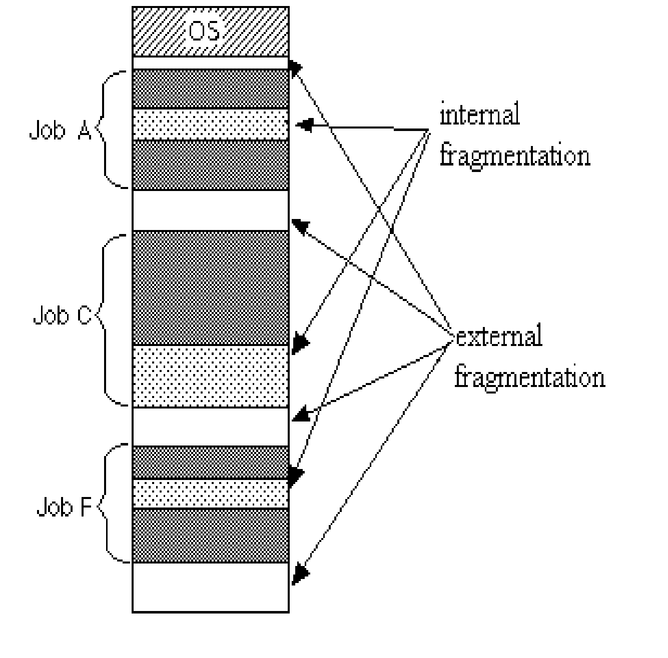
- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the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

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

- Given memory partitions of 200k, 250k,250k,200k, 320k, and 600k (in order, bottom to top), apply first fit, worst fit and best fit algorithms to place processes with the space requirement of 240k, 425k, 112k and 426k (in order). Which algorithm makes the most effective use of memory?

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 overhead



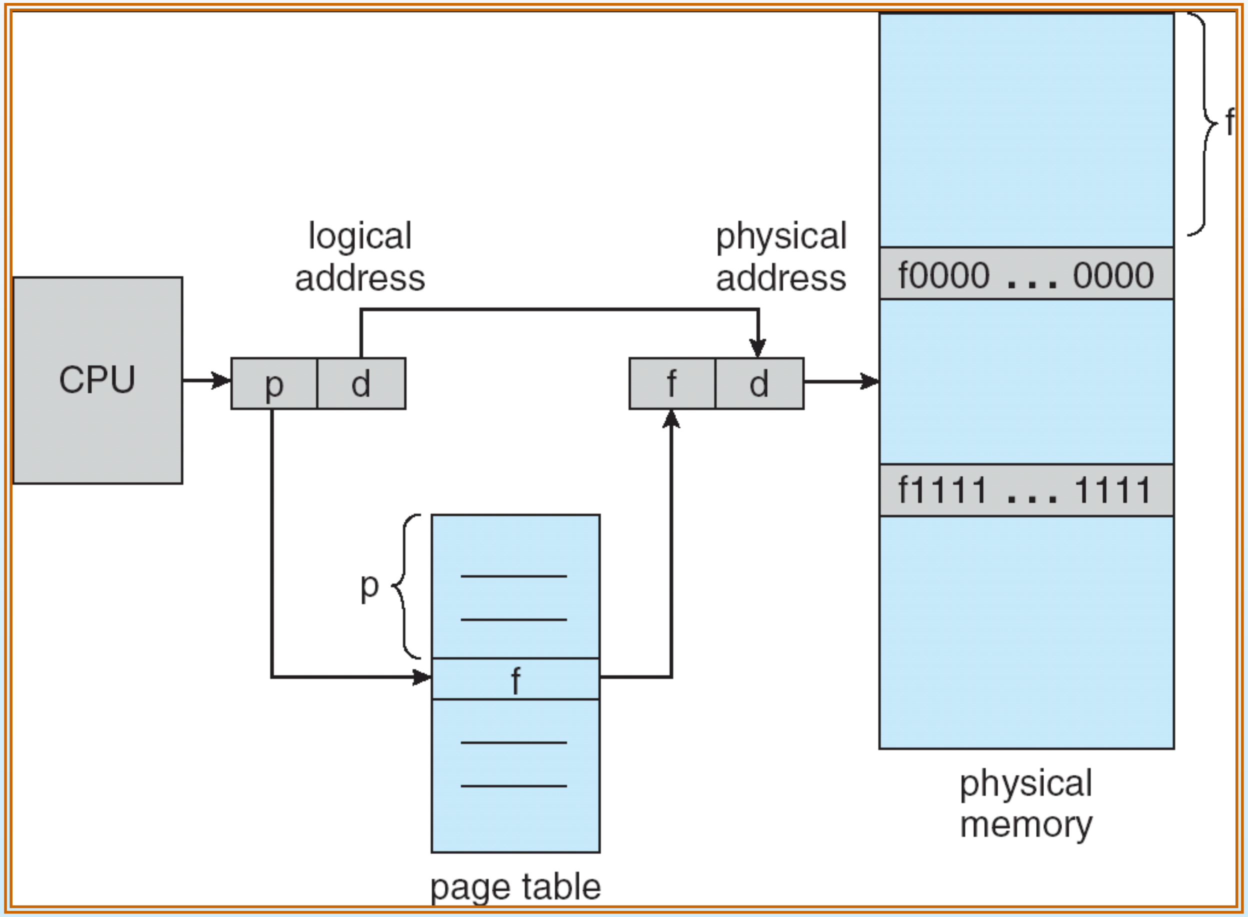
Paging

- Logical address space of a process can be **non-contiguous**; 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**

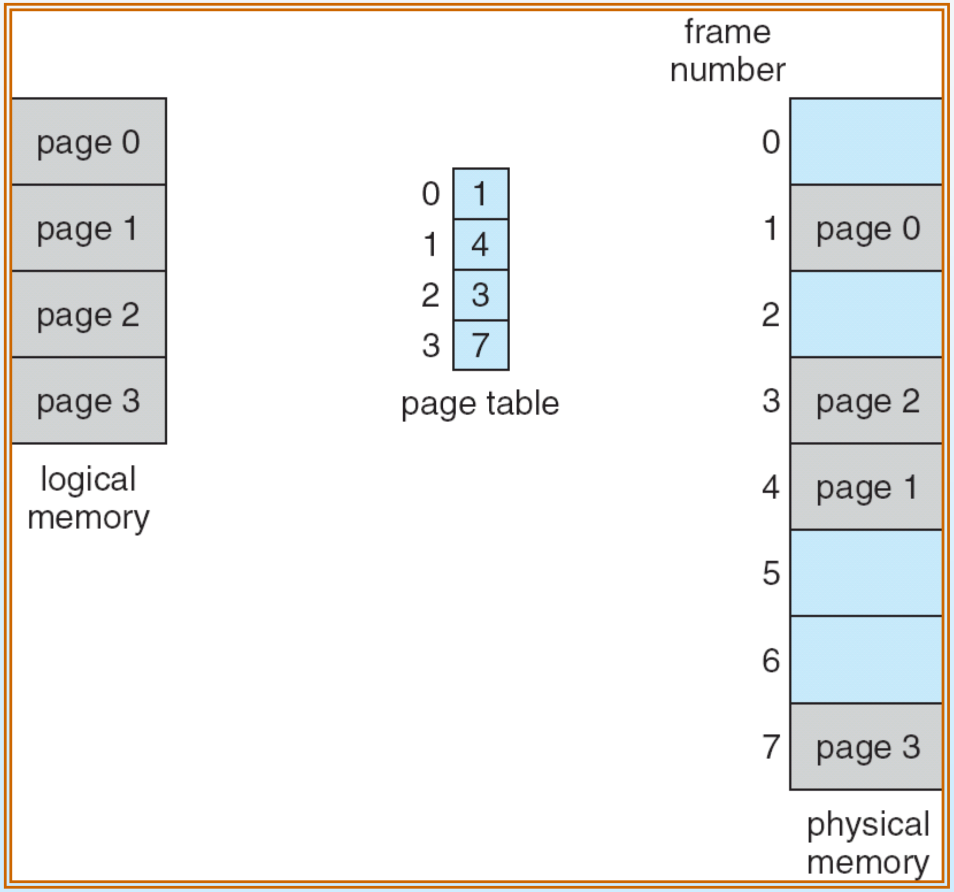
Address Translation Scheme

- 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

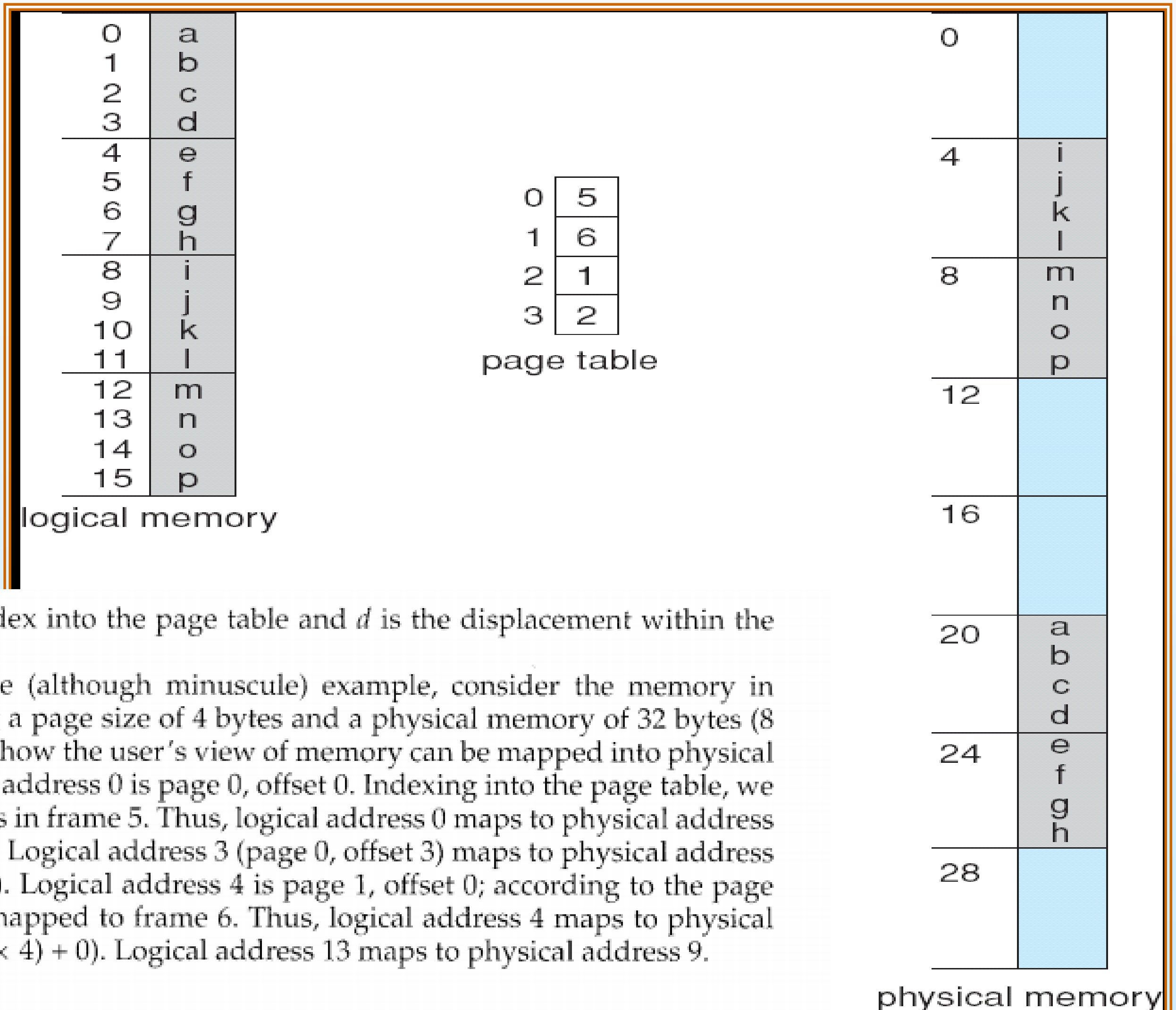
Address Translation Architecture



Paging Example



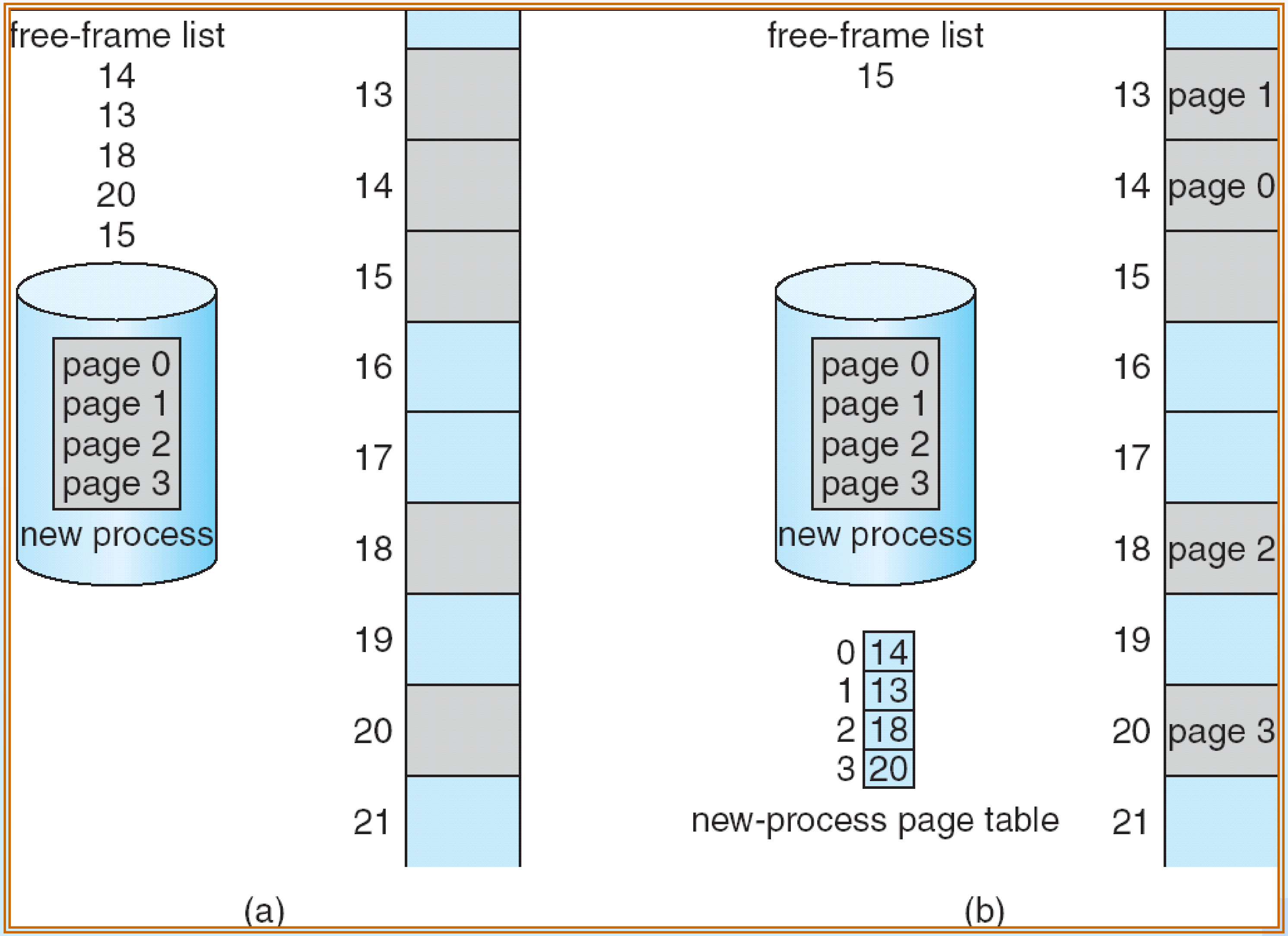
Paging Example



where p is an index into the page table and d is the displacement within the page.

As a concrete (although minuscule) example, consider the memory in Figure 8.9. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the user's view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 (= $(5 \times 4) + 0$). Logical address 3 (page 0, offset 3) maps to physical address 23 (= $(5 \times 4) + 3$). Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 (= $(6 \times 4) + 0$). Logical address 13 maps to physical address 9.

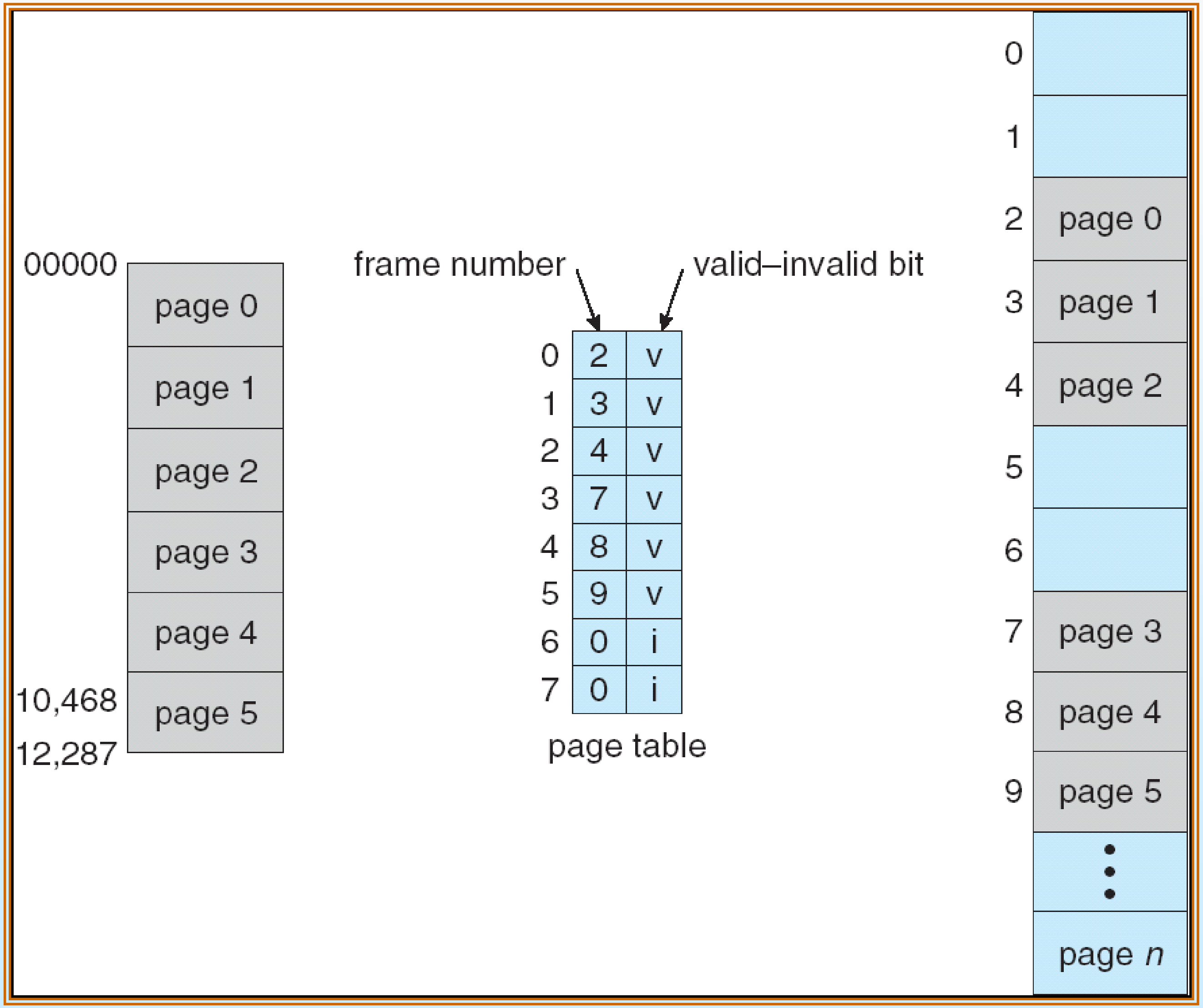
Free Frames


free-frame list
14
13
18
20
15
page 0
page 1
page 2
page 3
new process
13
14
15
16
17
18
19
20
21
(a)
free-frame list
15
page 0
page 1
page 2
page 3
new process
0 14
1 13
2 18
3 20
new-process page table
13 page 1
14 page 0
15
16
17
18 page 2
19
20 page 3
21
(b)

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

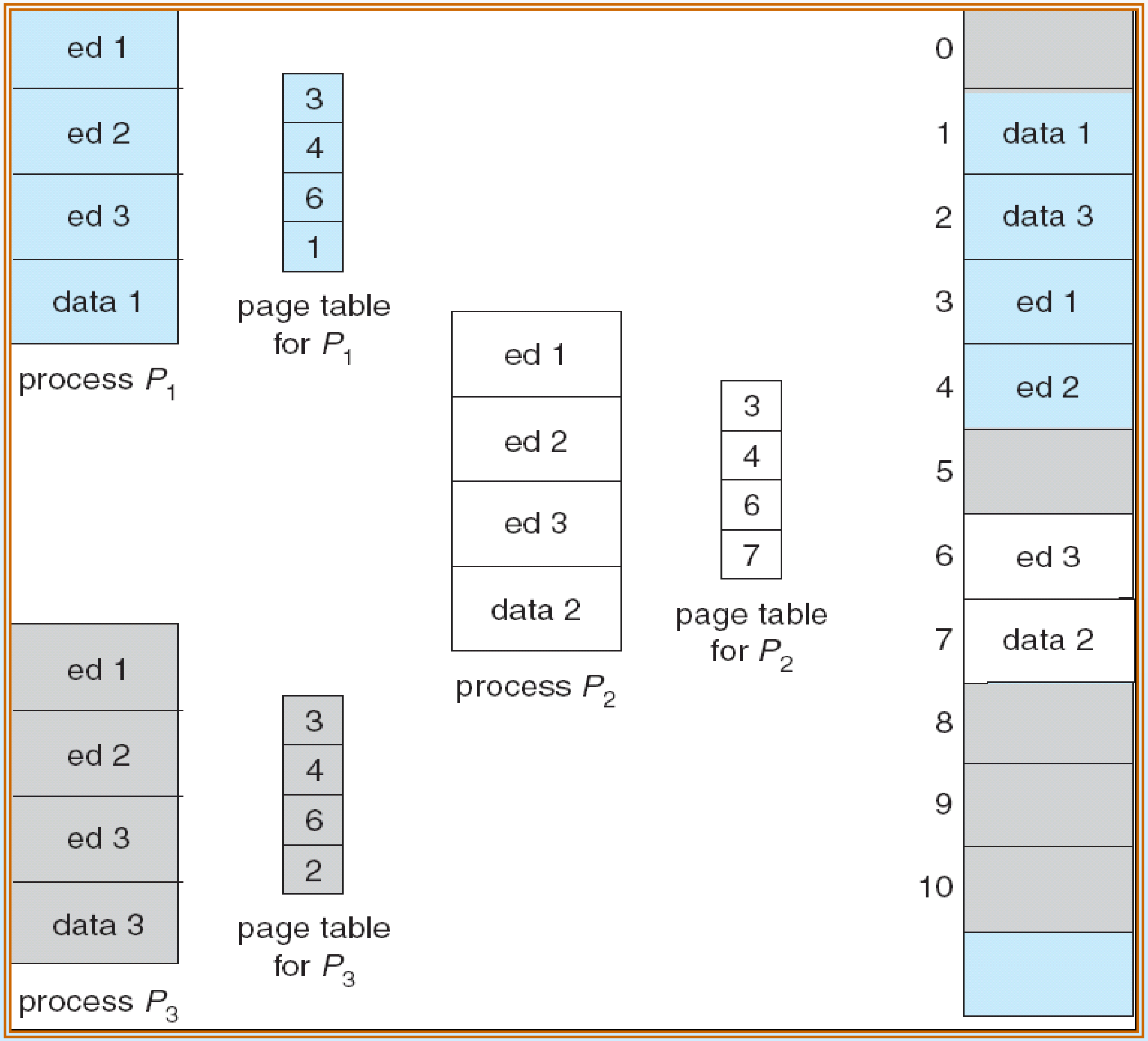
Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



Assume that page size = 2 bytes and Physical Memory = 34 bytes. If CPU generates logical addresses 5, 3, 9, 0, 11 and
respectively then how the users' view of memory can be mapped into physical memory?

P0	country
P1	much
P2	I
P3	very
P4	love
P5	my

Logical Memory

P0	7
P1	9
P2	4
P3	8
P4	5
P5	6

PMT

Main Memory

Segmentation

- Memory-management scheme that <u>supports user view of memory</u>
- A program is a collection of segments. A segment is a logical unit such as:

    main program,

    procedure,

    function,

    method,

    object,

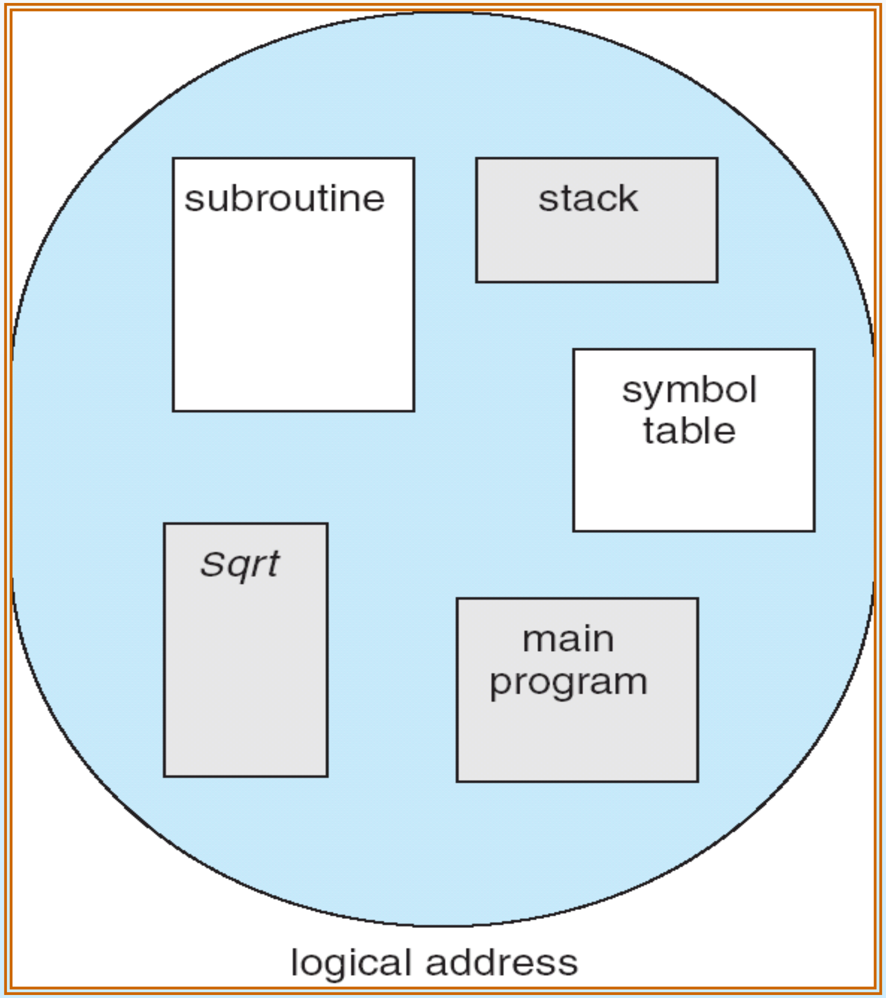
    local variables, global variables,

    common block,

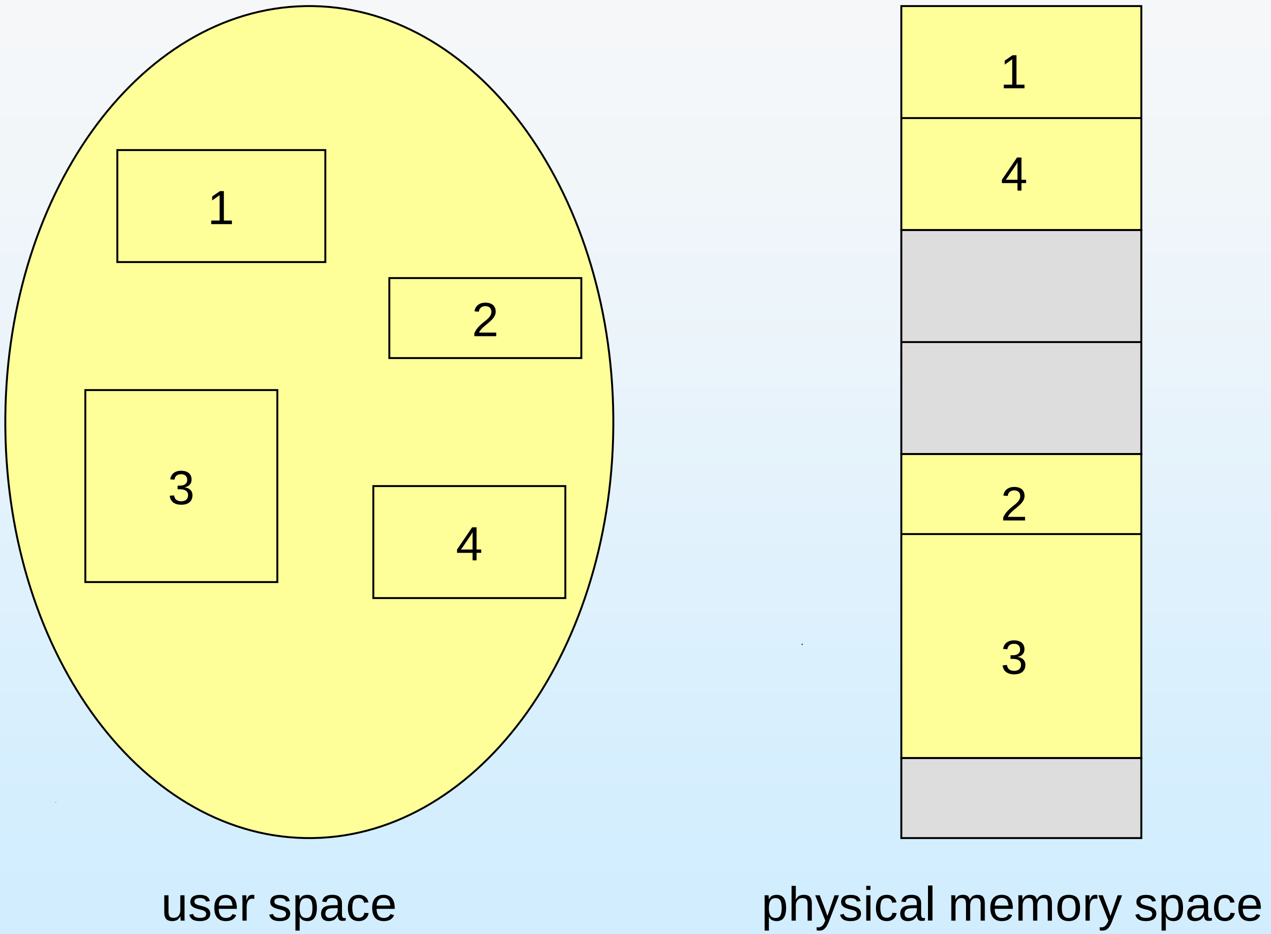
    stack,

    symbol table, arrays

User's View of a Program



Logical View of Segmentation



Segmentation Architecture

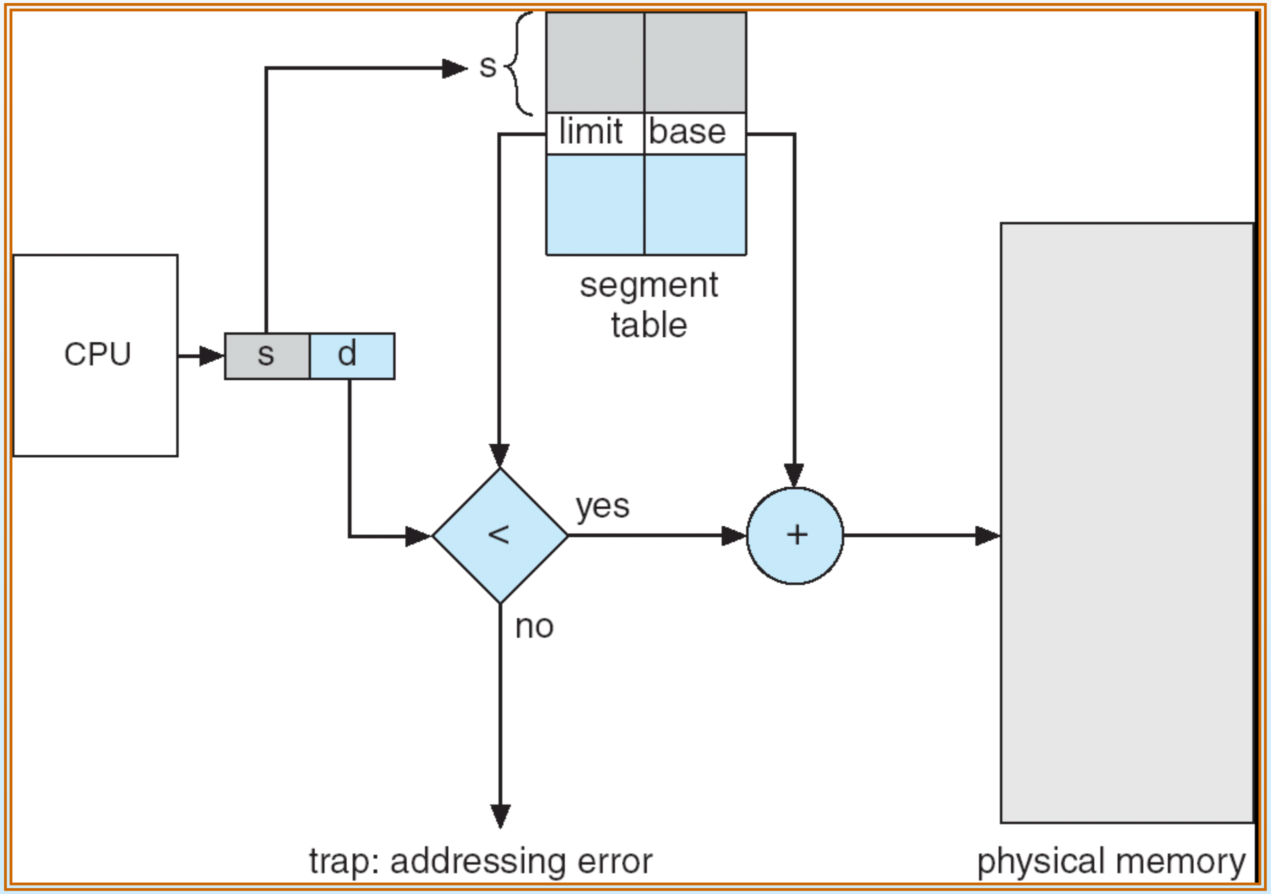
- Logical address consists of a two tuple:

  <segment-number, offset>,

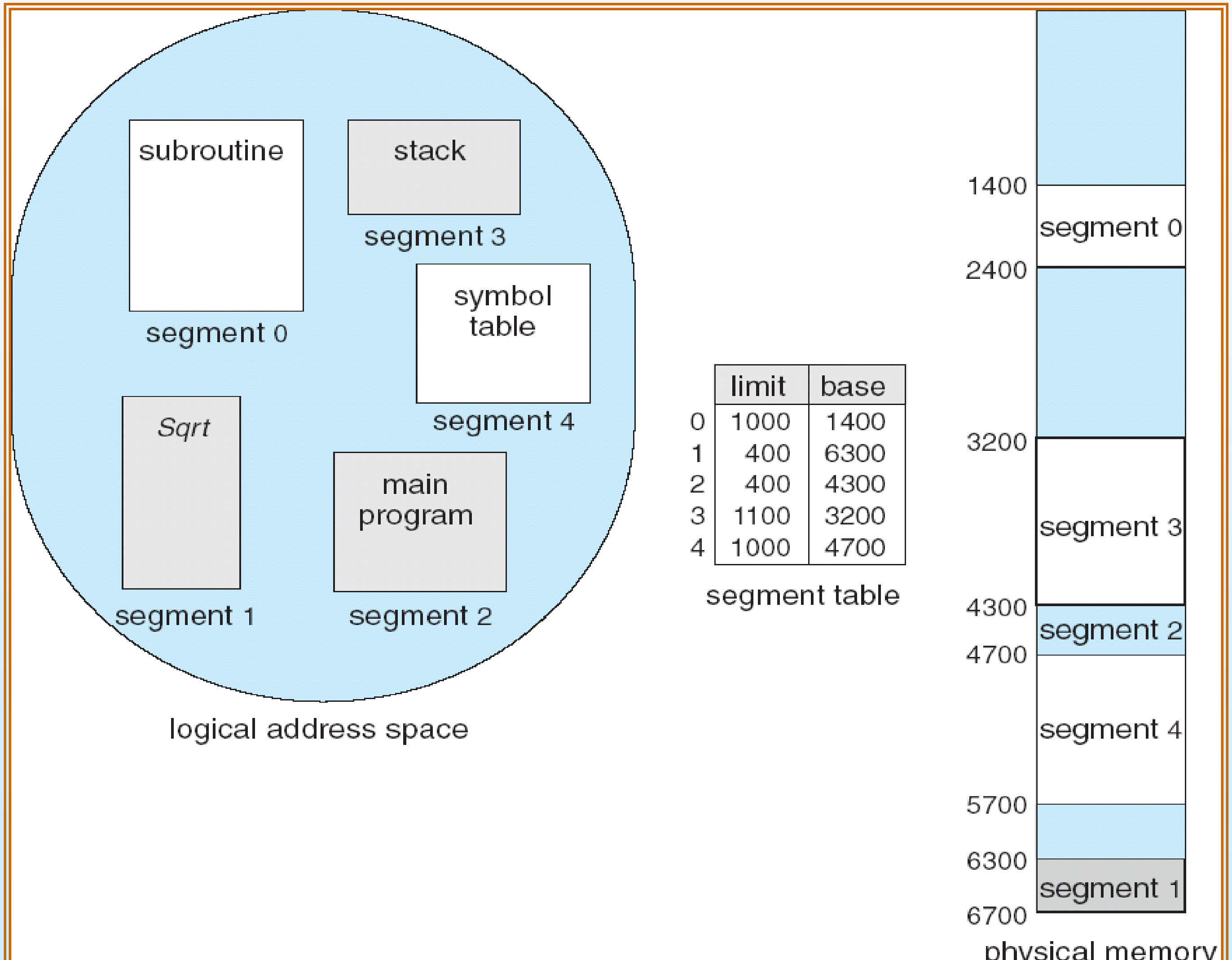
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - *limit* – specifies the length of the segment
- *Segment-table base register (STBR)* points to the segment table's location in memory
- *Segment-table length register (STLR)* indicates number of segments used by a program;

  segment number s is legal if $s < \text{STLR}$

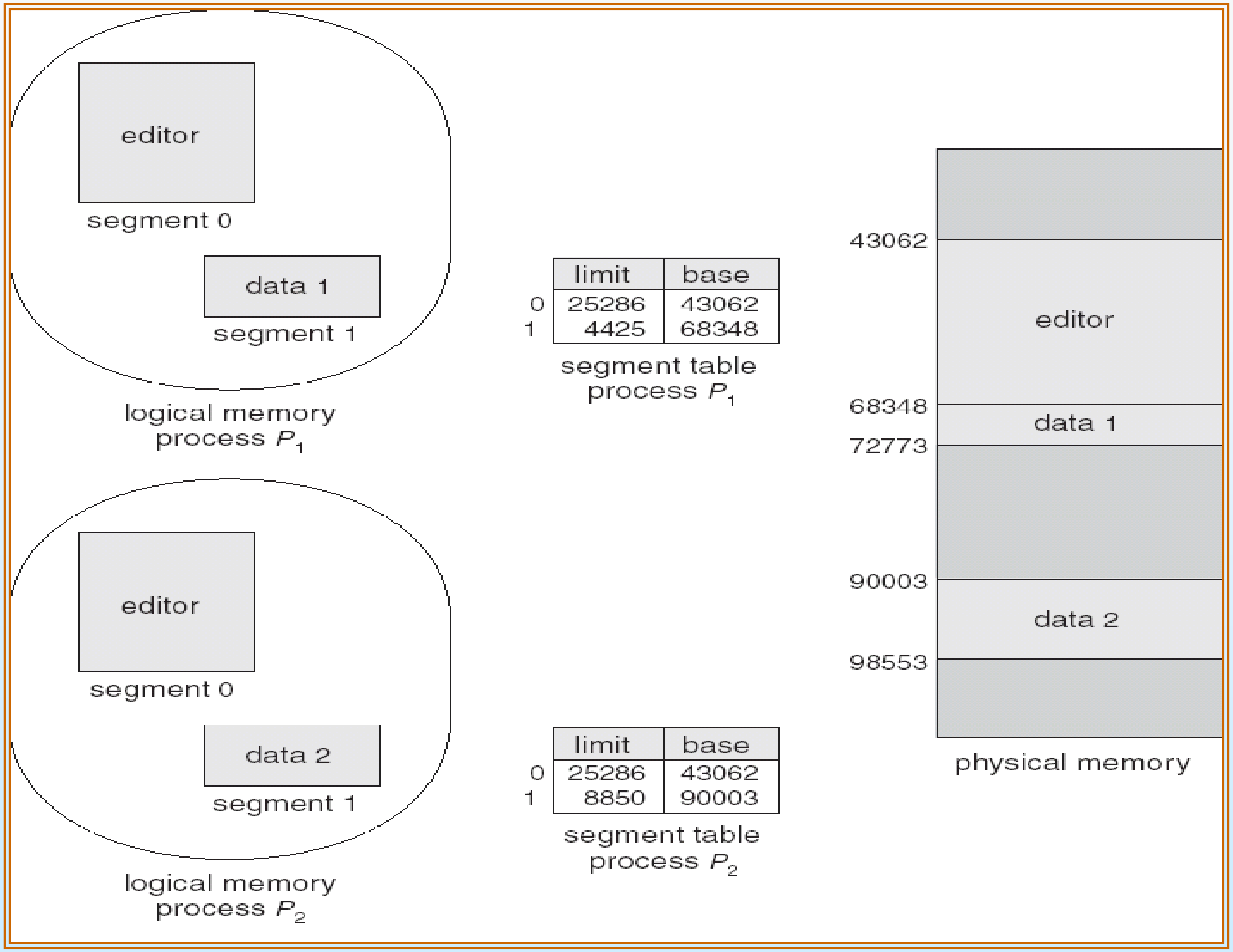
Address Translation Architecture



Example of Segmentation



Sharing of Segments



Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
  - validation bit = 0 $\Rightarrow$ illegal segment
  - read/write/execute privileges
- Since segments vary in length, memory allocation is a **dynamic storage-allocation problem**
- Segmentation leads to **external fragmentation**
- Code sharing occurs at segment level; shared segments

Consider the following segment table:

8.16. Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addressed for the following logical addresses?

(a) 0,430
(b) 1,10
(c) 2,500
(d) 3,400
(e) 4,112

- (a) 219 + 430 = 649
- (b) 2300 + 10 = 2310
- (c) illegal reference; traps to operating system
- (d) 1327 + 400 = 1727
- (e) illegal reference; traps to operating system

End of Chapter 8