Memory Management

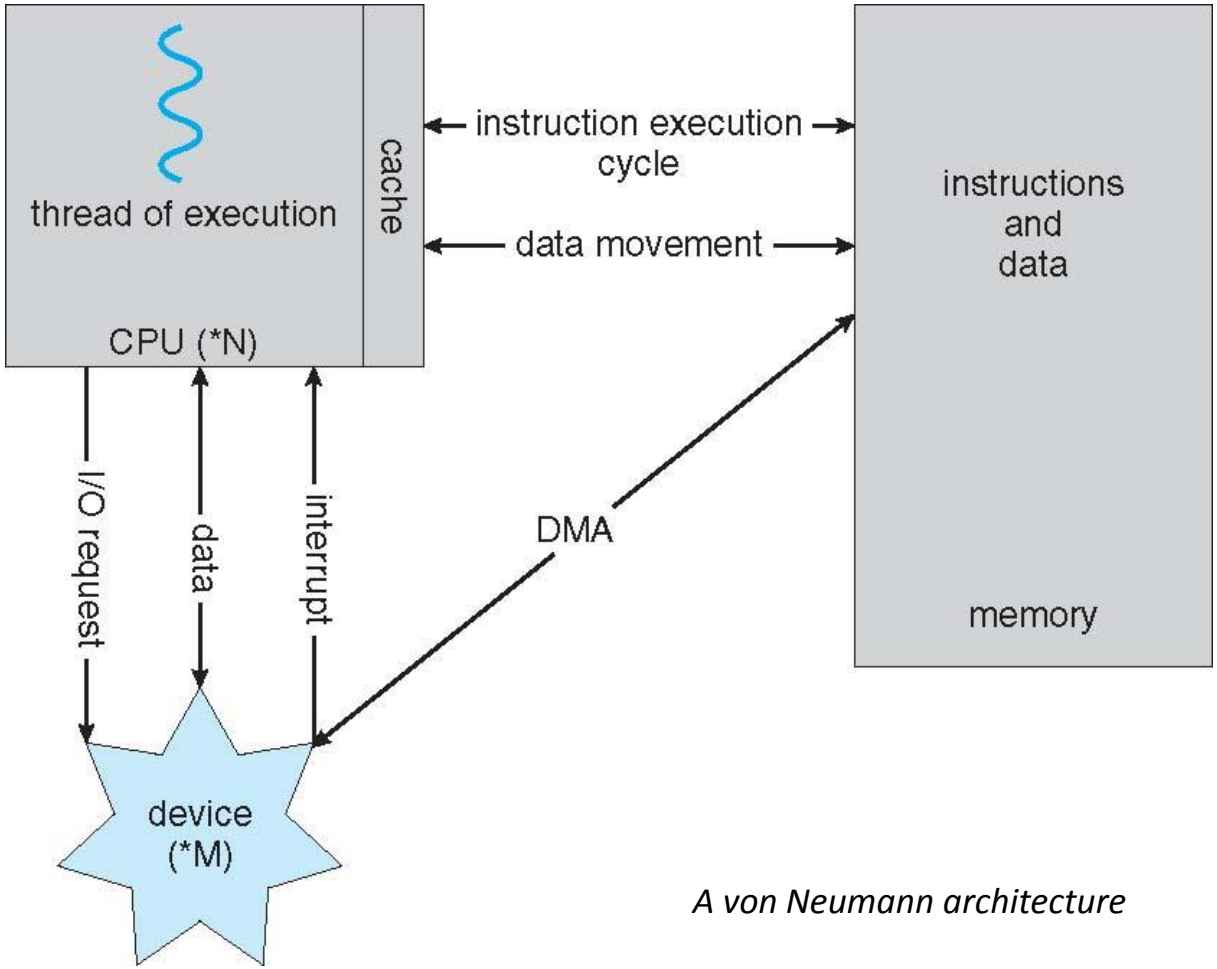
- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is to be in memory
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by which **process**
  - Deciding which processes and data to move into and out of memory
  - Allocating and deallocating memory space as needed by running processes

Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

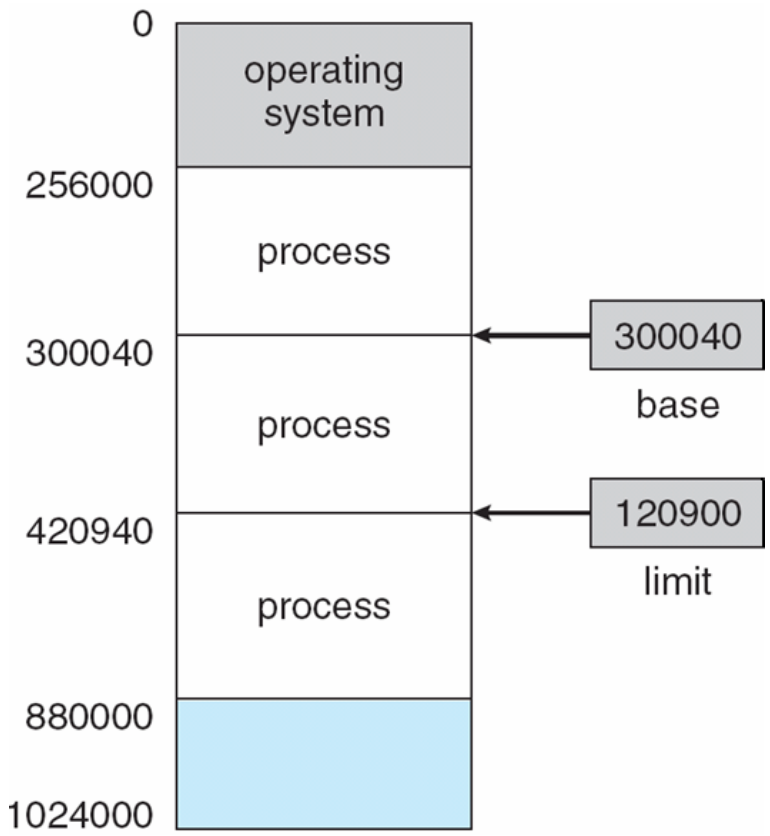
How a Modern Computer Works



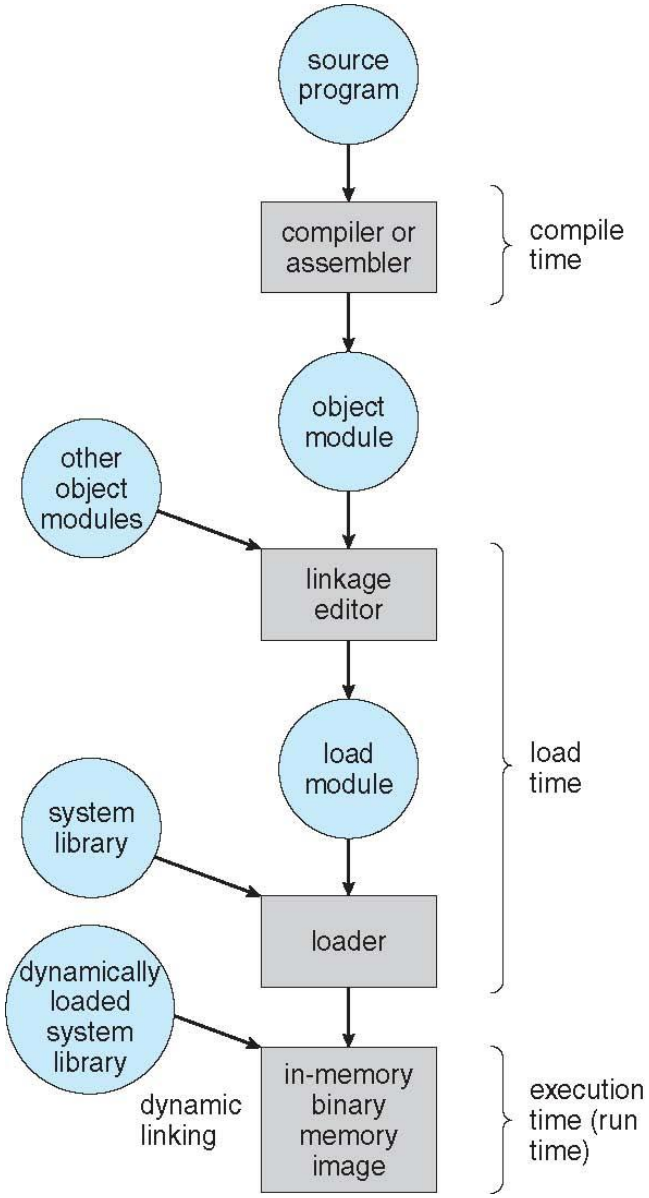
A von Neumann architecture

Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



Multistep Processing of a User Program



Compile time: If bind memory location in compile time; must recompile code if location changes

Load time: Must generate **relocatable code** if memory location is not known at compile time

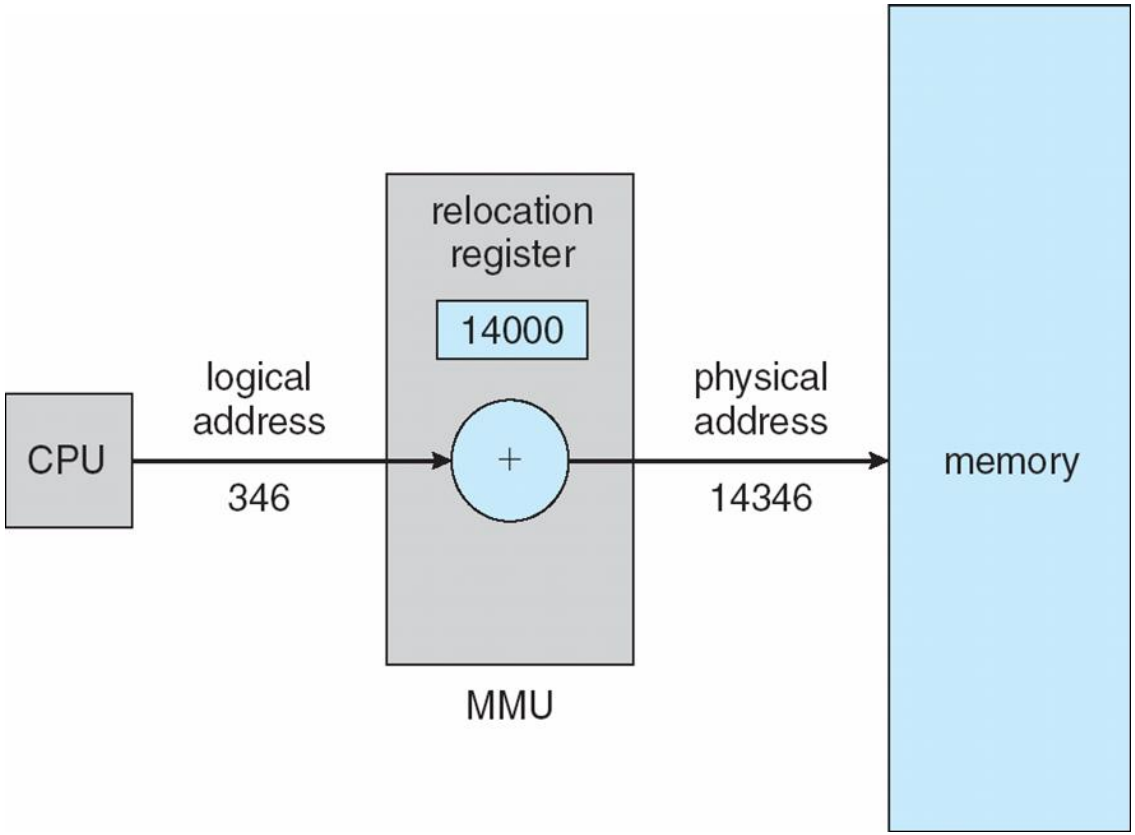
Execution time: Binding delayed until run time; the process can be moved during its execution from one memory segment to another

Need hardware support for address maps (e.g., base and limit registers)

Memory-Management Unit (MMU)

- Hardware device that at run time maps **virtual** address to **physical** address
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Dynamic relocation using a relocation register



Dynamic Loading and Dynamic Linking

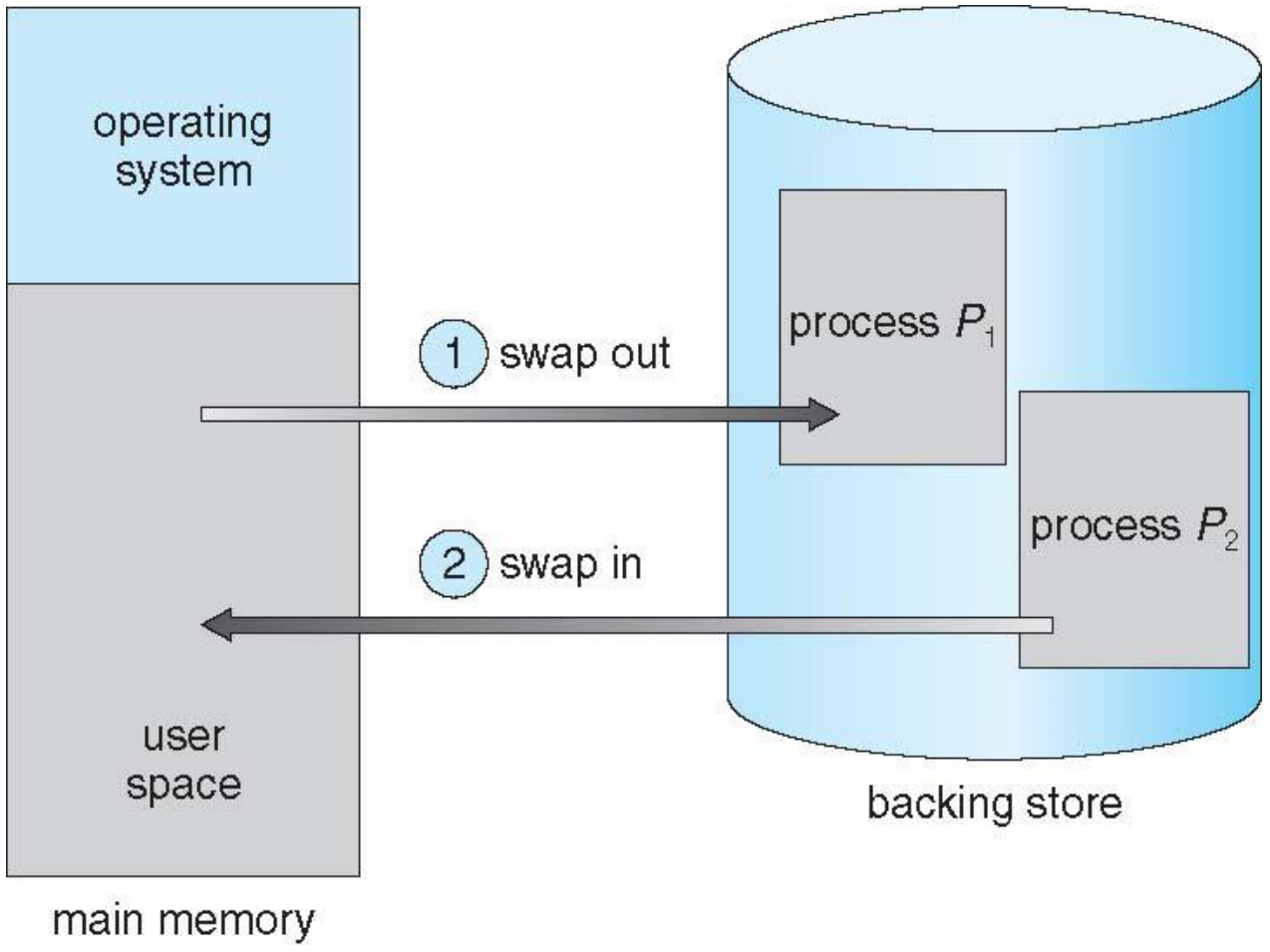
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format

- Static linking – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- Dynamic linking is particularly useful for libraries

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk
- Does the swapped out process need to swap back in to same physical addresses?
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



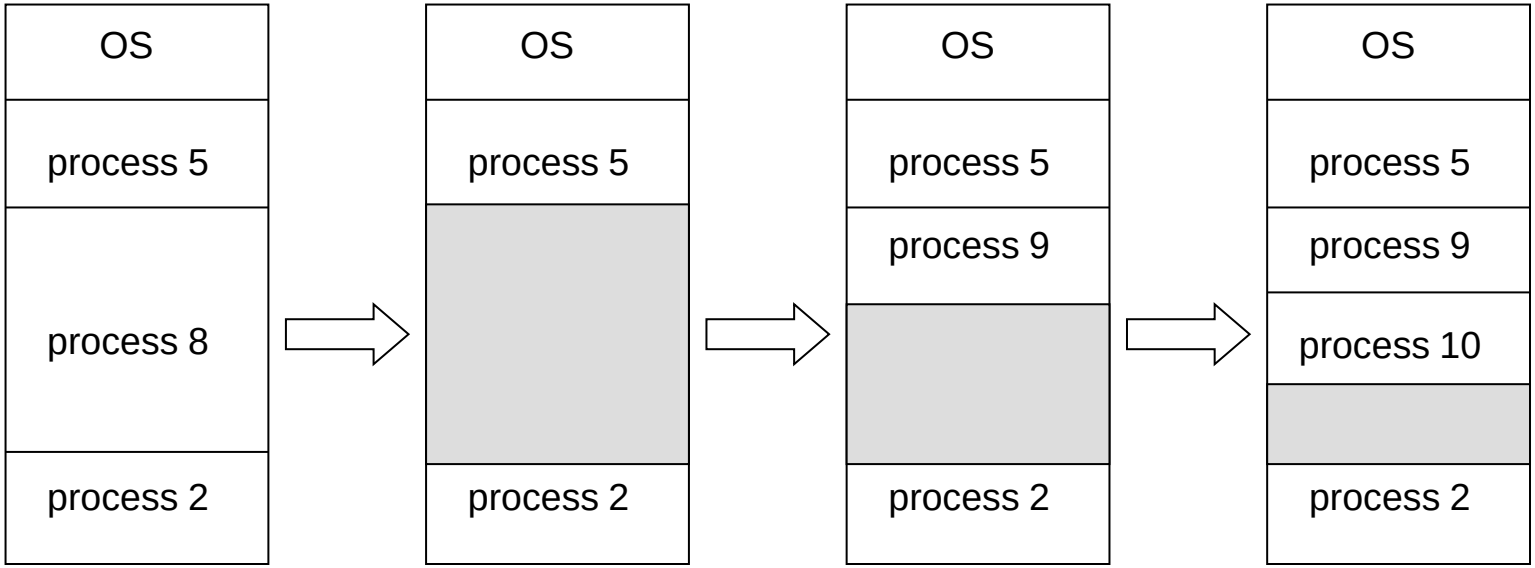
Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high

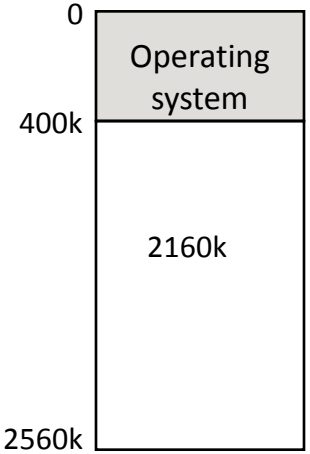
- 100MB process swapping to hard disk with transfer rate of 50MB/sec => 2 sec = 2000 ms
  - Plus disk latency of 8 ms
  - Swap out time of 2008 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4016ms (> 4 seconds)

Contiguous Allocation

- Multiple-partition allocation
  - When a Process arrives, it is allocated memory from a large enough memory free space to place it
  - When a Process terminates, frees its partition, adjacent free partitions combined



Contiguous Memory Allocation Example



job queue

process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

New process is loaded is there is memory for it!

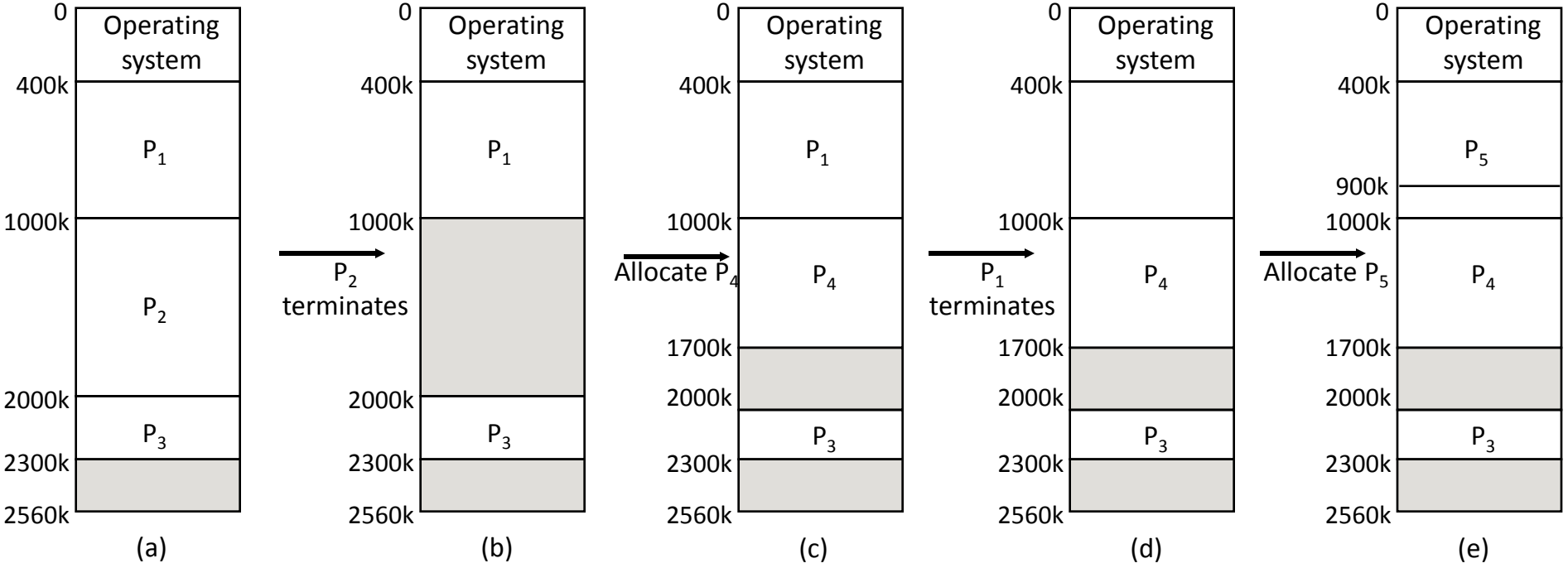


Figure 8.8 Memory allocation and long term scheduling

Dynamic Storage-Allocation Problem

How to satisfy a request of size *n* for a Process 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

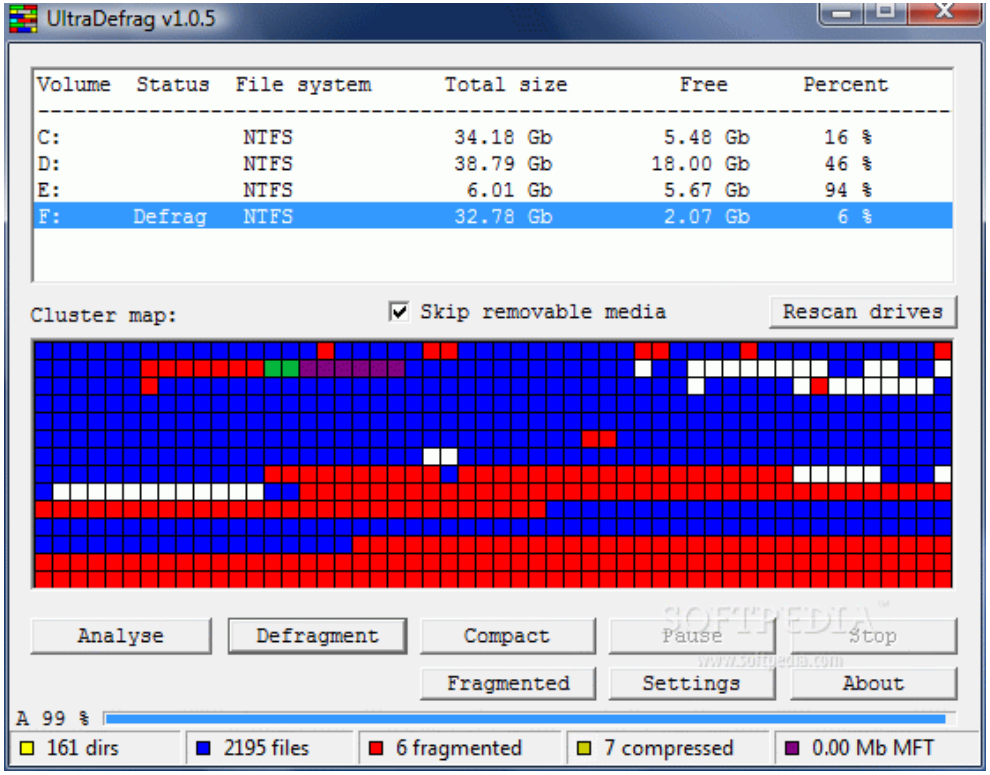
First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Fragmentation

- total memory space exists to satisfy a request, but it is not contiguous
  - **External Fragmentation**

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



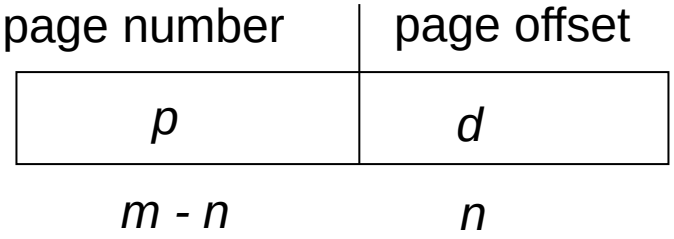
Paging

- Logical address space of a process can be non-contiguous
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes

- 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

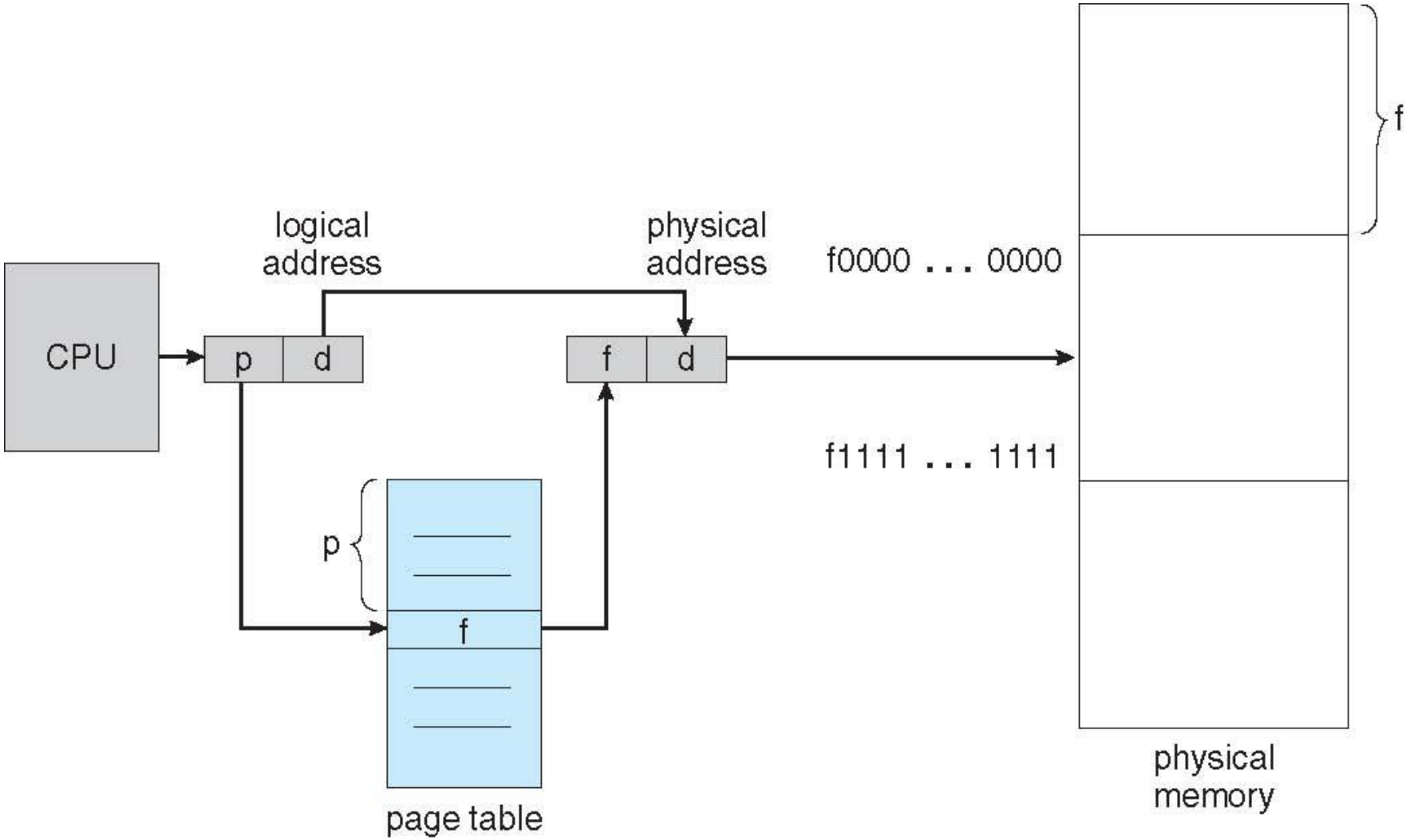
Address Translation Scheme

- 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

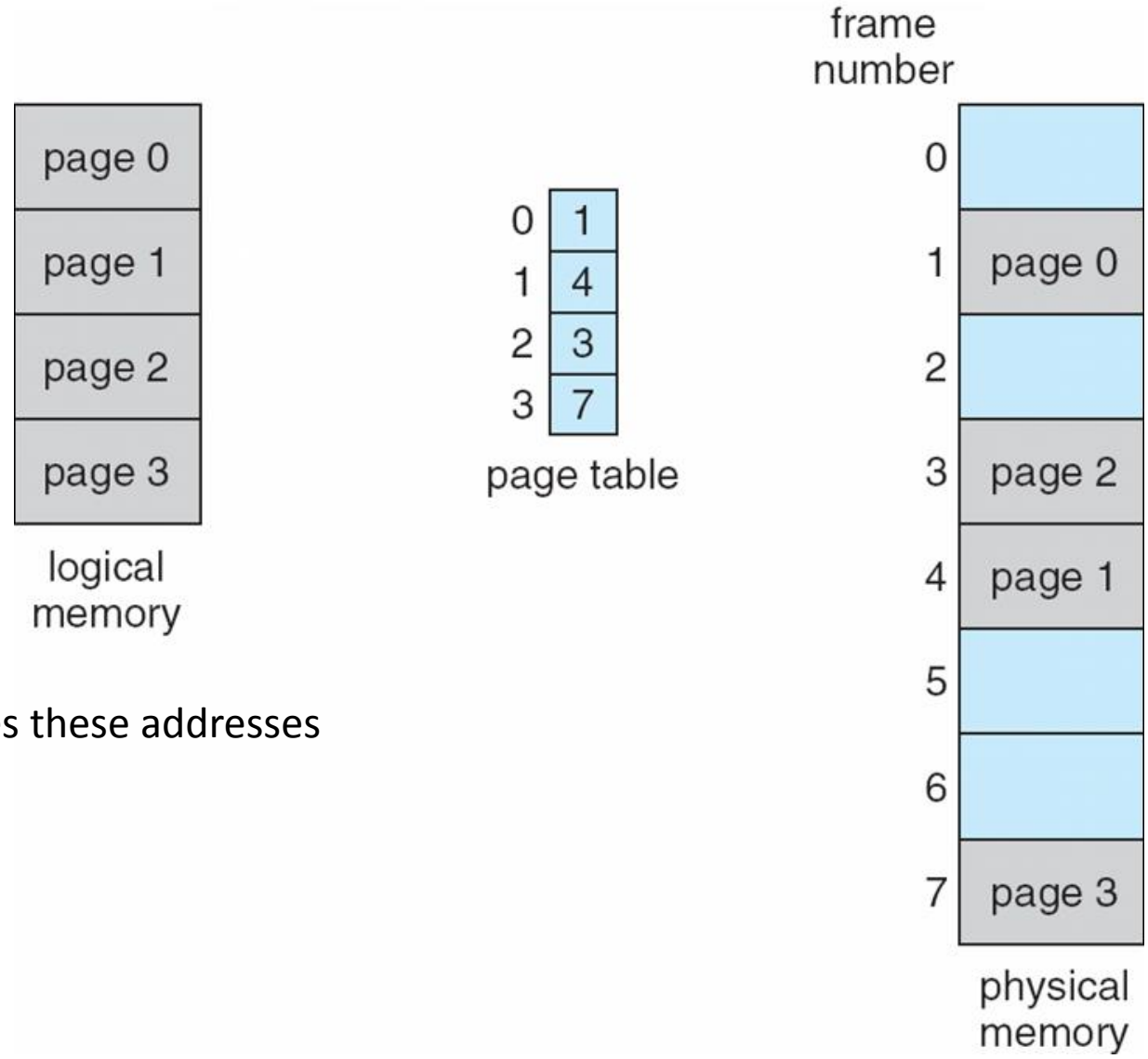


  - For given logical address space 2^m and page size 2^n

Paging Hardware

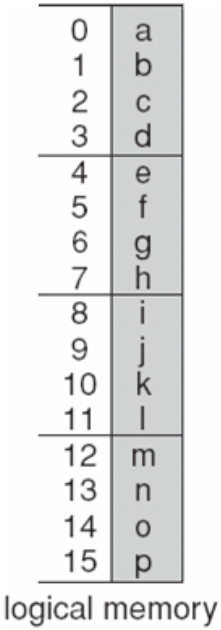


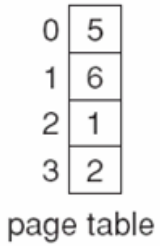
Paging Model of Logical and Physical Memory

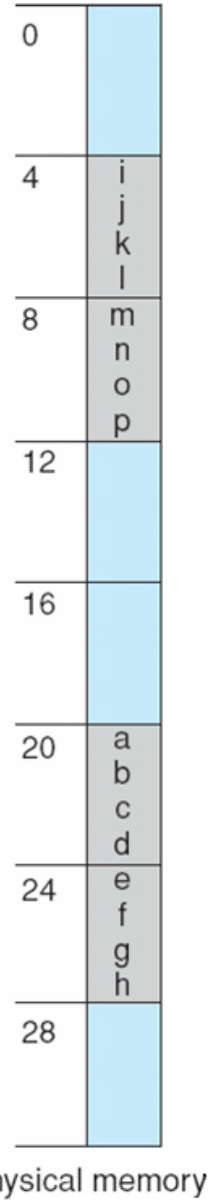


CPU uses these addresses

Paging Example





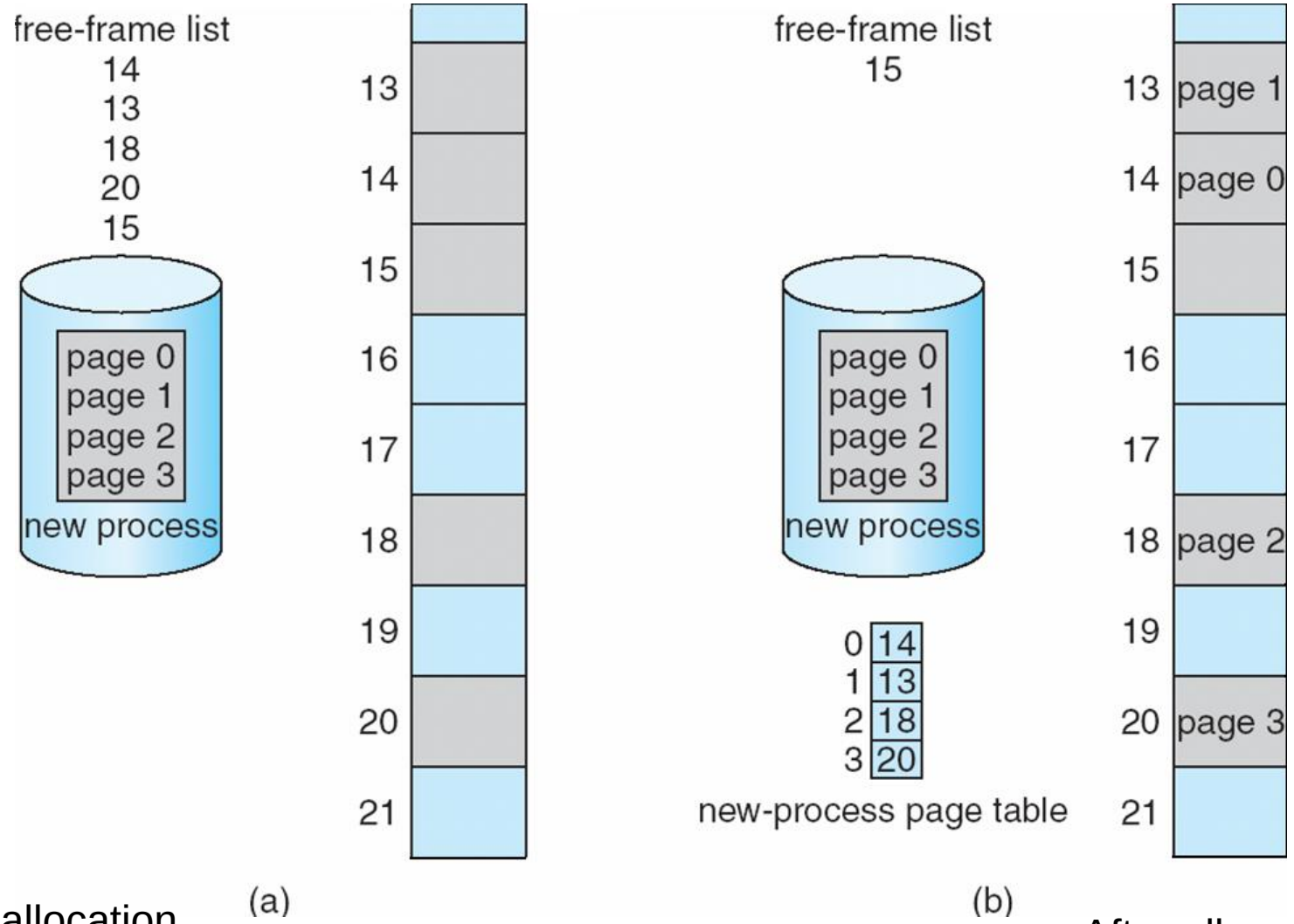


n=2 and *m*=4

32-byte memory and 4-byte pages

page number	page offset
p	*d*
m - n	*n*

Tracking Free Frames



Before allocation

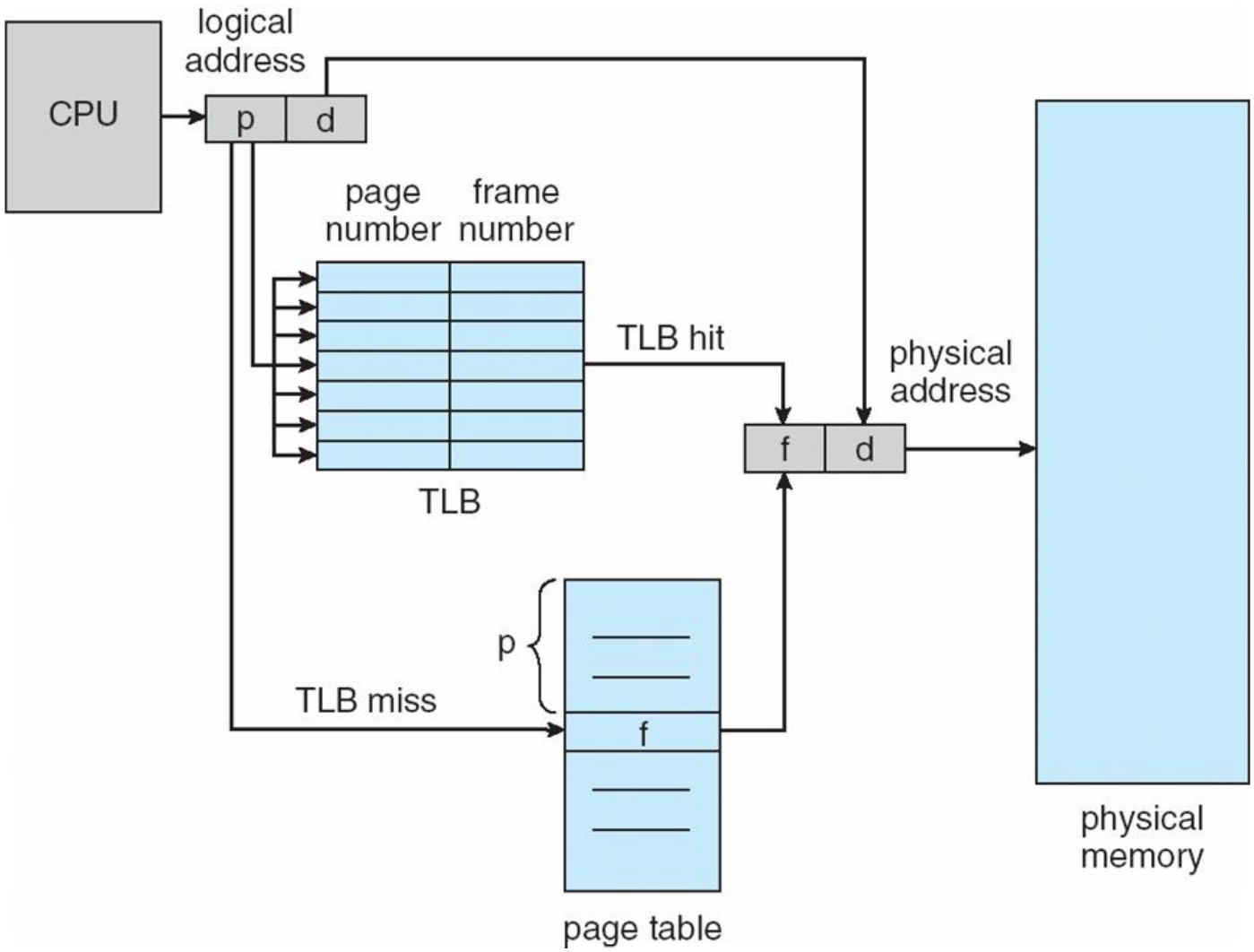
After allocation

Implementation of Page Table

- Page table is kept in main memory
- 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)**

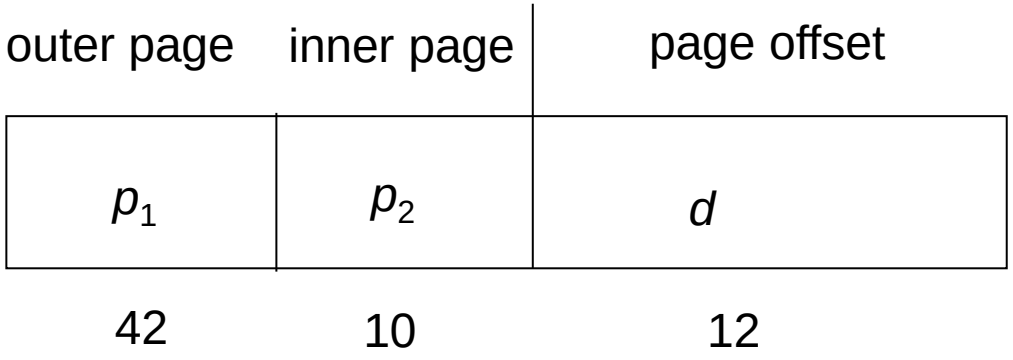
- TLBs typically small (64 to 1,024 entries)

Paging Hardware With TLB



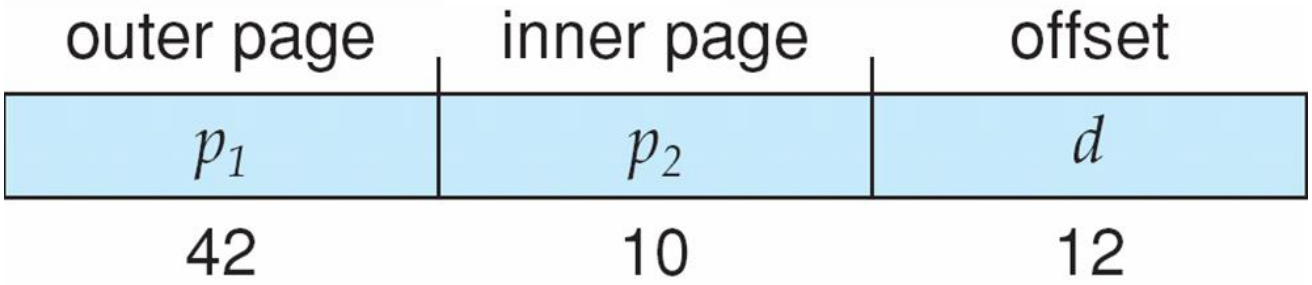
64-bit Logical Address Space

- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like

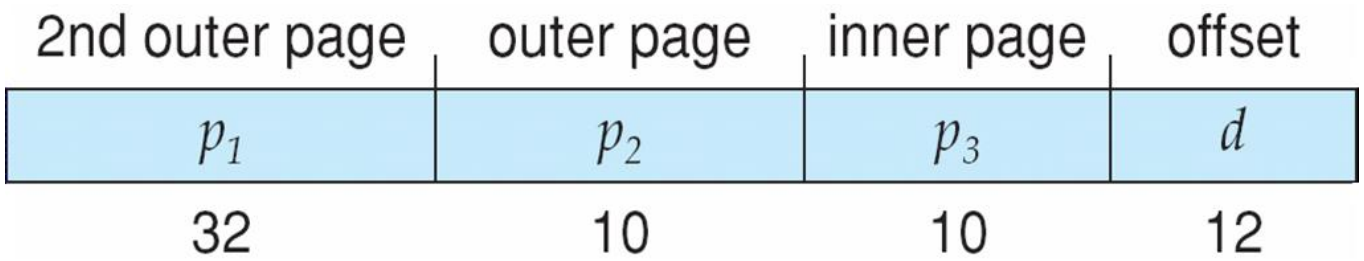


Three-level Paging Scheme

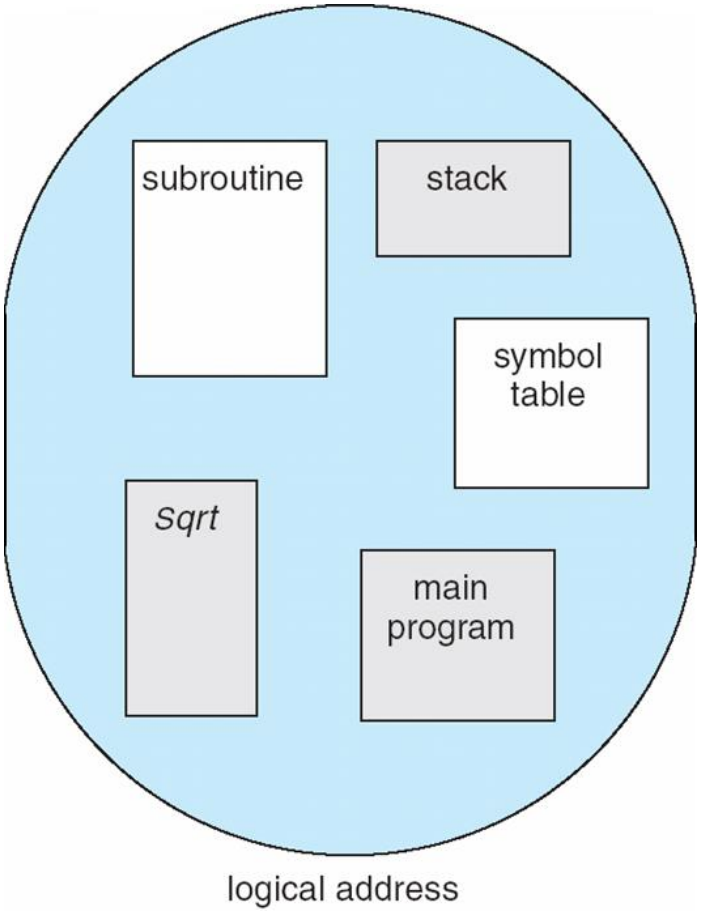
2 page tables, or 2 level caches



3 page tables, or 3 level caches



User's View of a Program

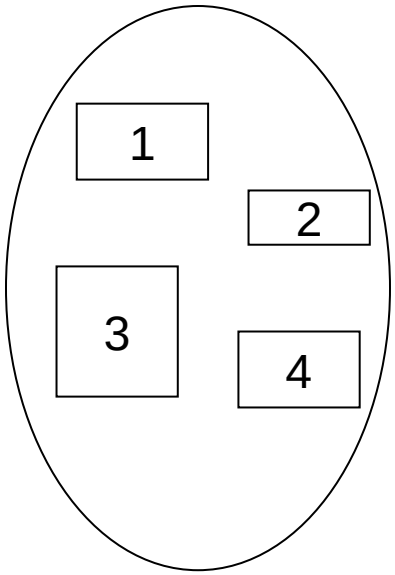


A **program** is a collection of segments

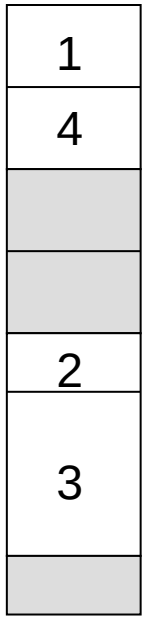
A **segment** is a logical unit such as:

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

Logical View of Segmentation



user space



physical memory space

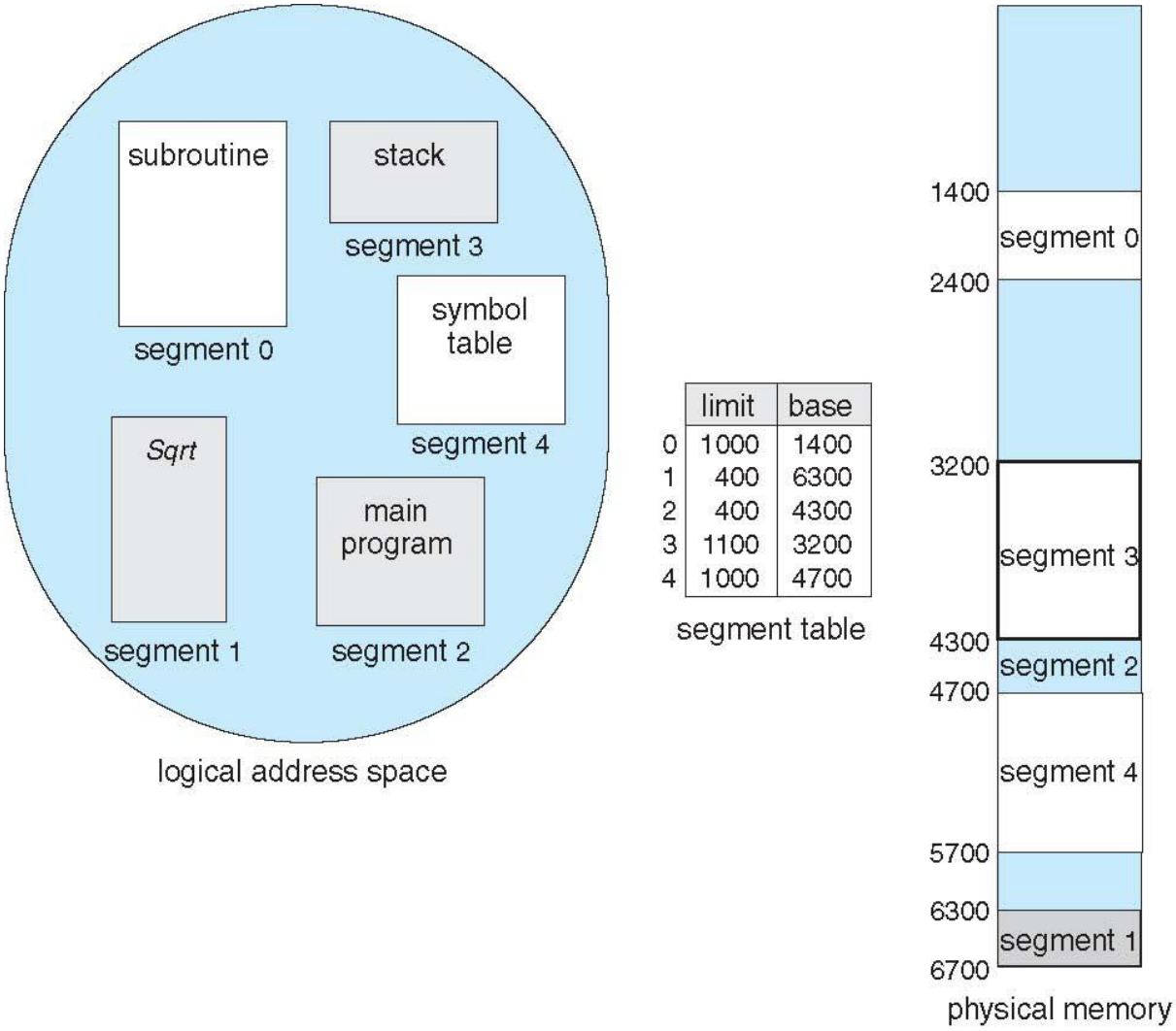
Logical address consists of a two tuple:

<segment-number, offset>,

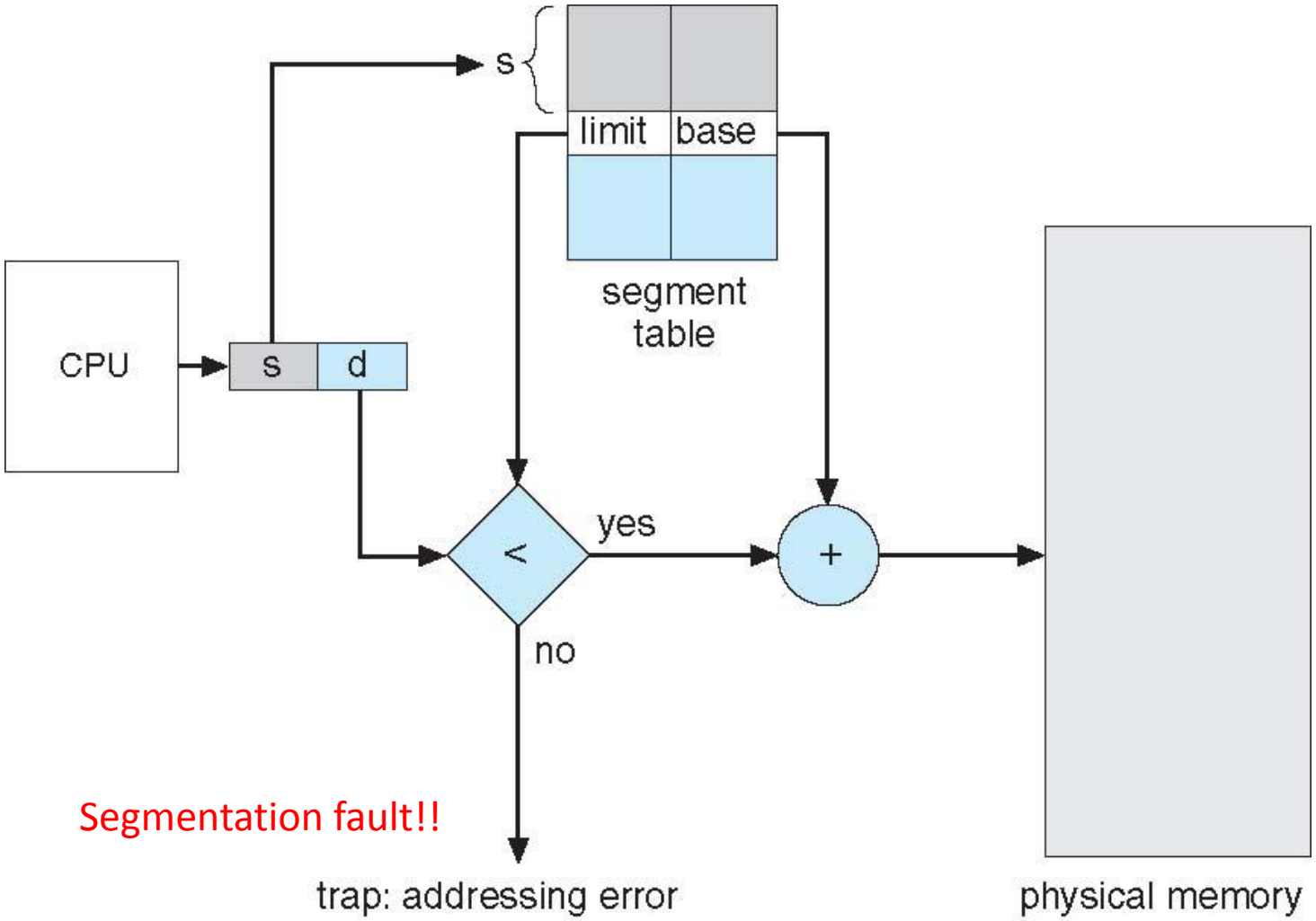
Segment table – maps two-dimensional physical addresses; each table entry has:

- **base** – contains the starting physical address where the segments reside in memory
- **limit** – specifies the length of the segment

Example of Segmentation



Segmentation Hardware



Exercises

Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Which algorithm makes the most efficient use of memory?

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

What are the physical addresses for the following logical addresses?

a. 0,430

b. 1,10

c. 2,500

d. 3,400

e. 4,112