ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

CHAPTER 3 - MEMORY MANAGEMENT

- Basic memory management
- Swapping
- Virtual memory
- Design issues for paging systems
- Implementation issues
- Segmentation

Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Memory manager handles the memory hierarchy

Basic Memory Management
Logical vs. Physical Address Space

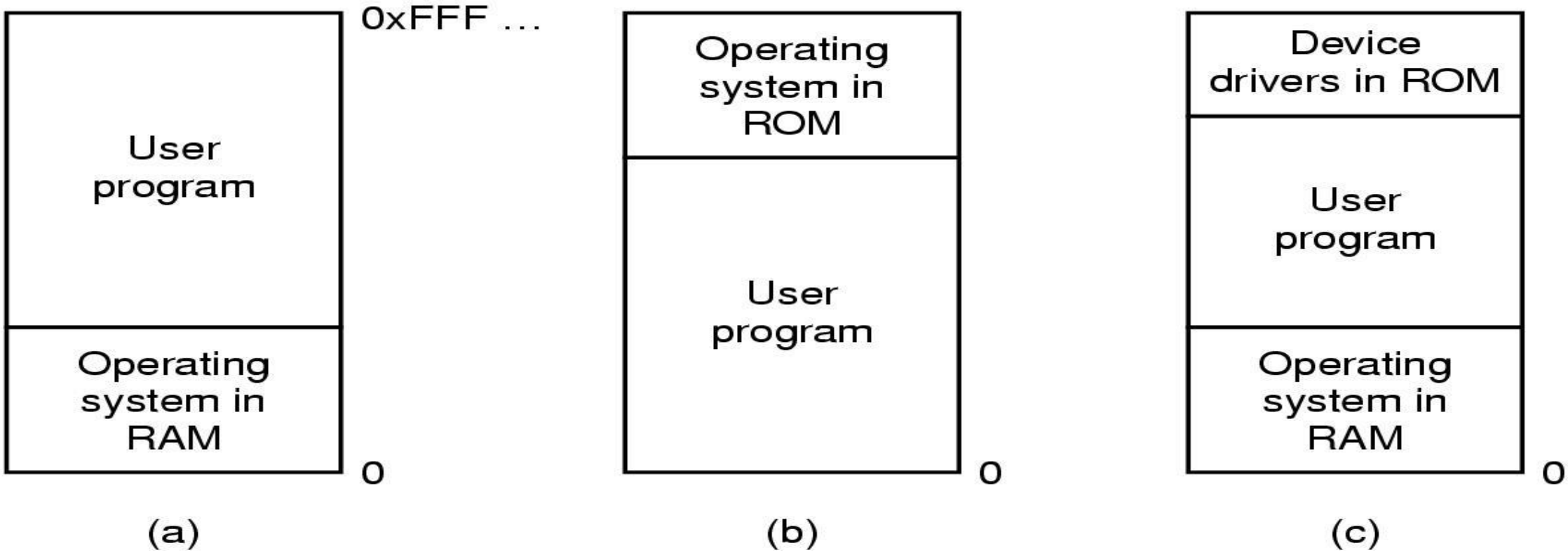
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit

Basic Memory Management

Monoprogramming without Swapping or Paging

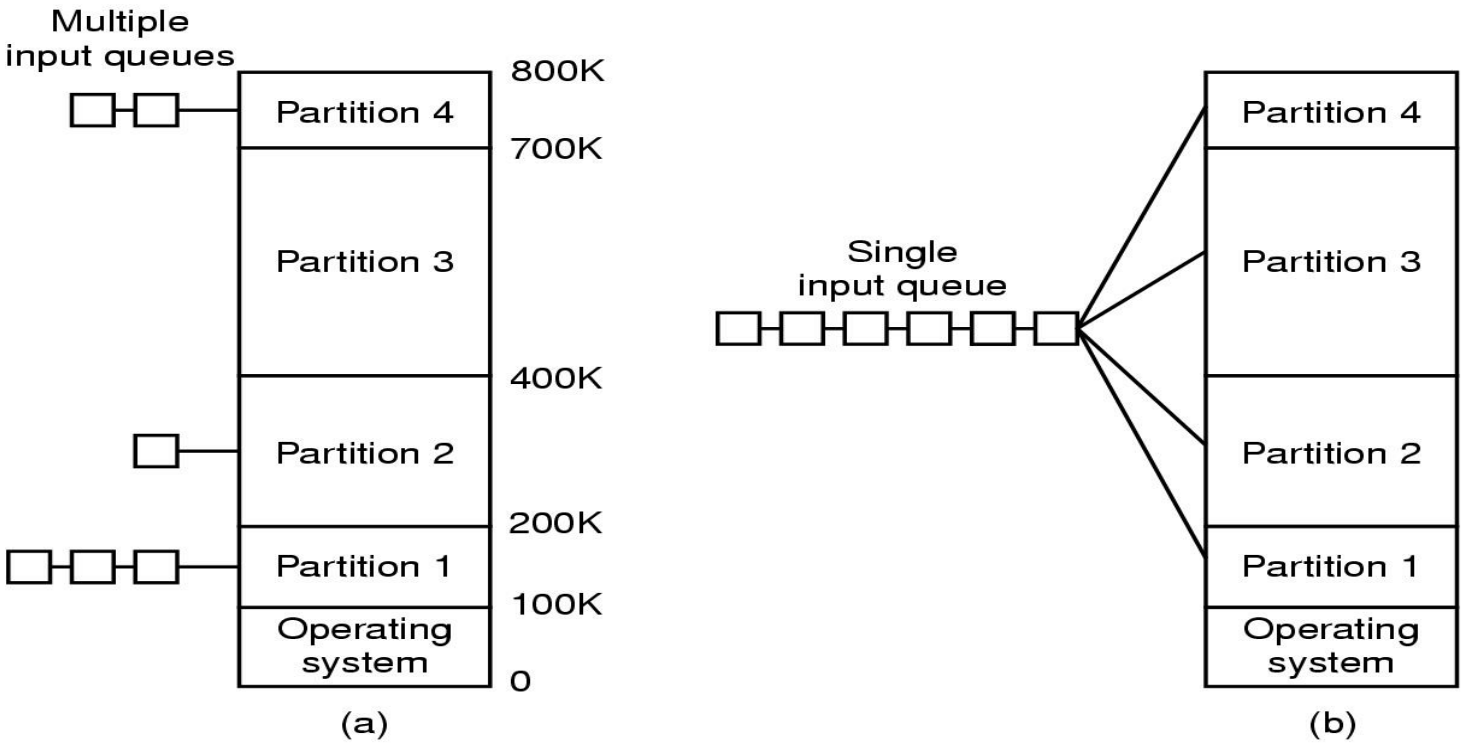
Three simple ways of organizing memory

- an operating system with one user process



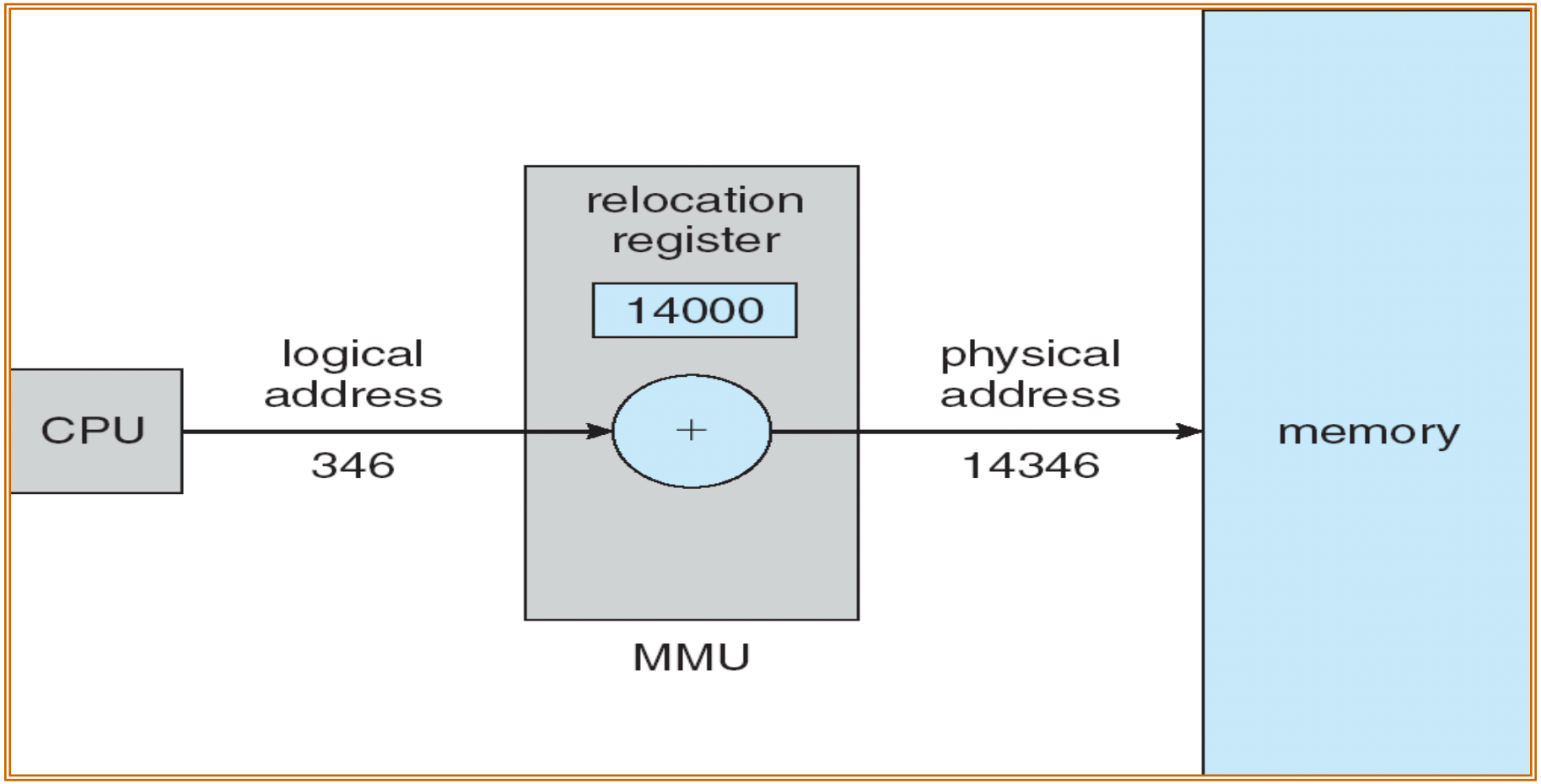
Basic Memory Management
Multiprogramming with Fixed Partitions

- Fixed memory partitions
  - (a) separate input queues for each partition
  - (b) single input queue



Basic Memory Management

Dynamic relocation using a relocation register



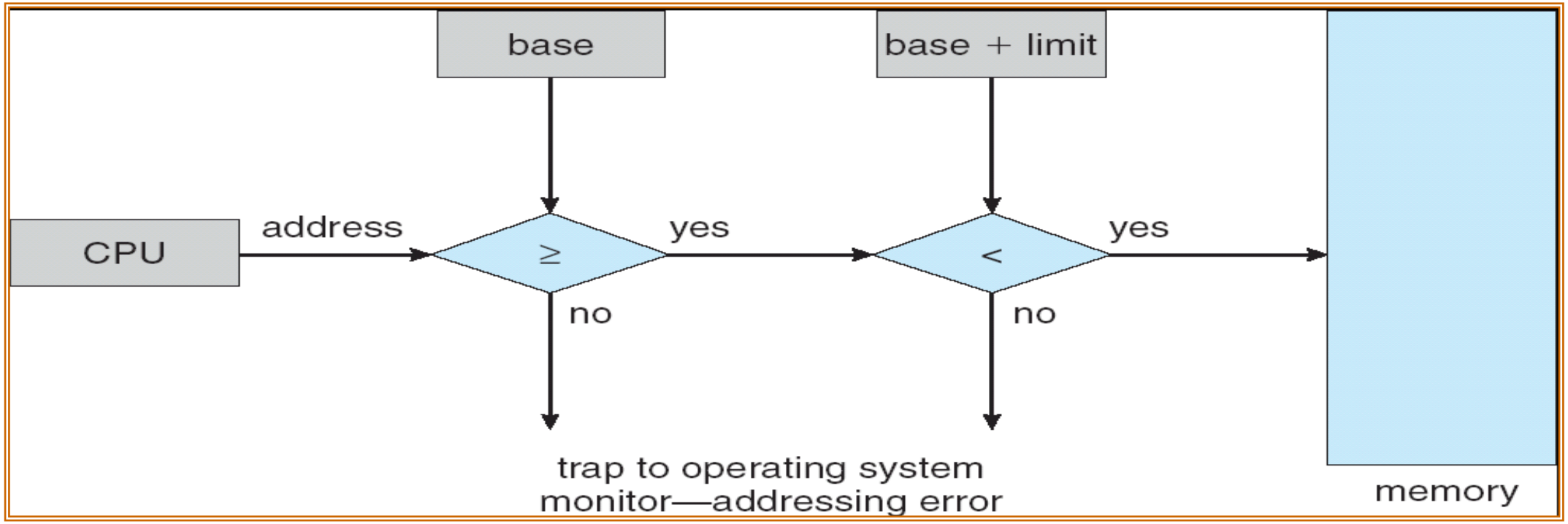
Basic Memory Management
Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - address locations of variables, code routines cannot be absolute
  - must keep a program out of other processes' partitions
- Use base and limit values
  - address locations added to base value to map to physical addr
  - address locations larger than limit value is an error

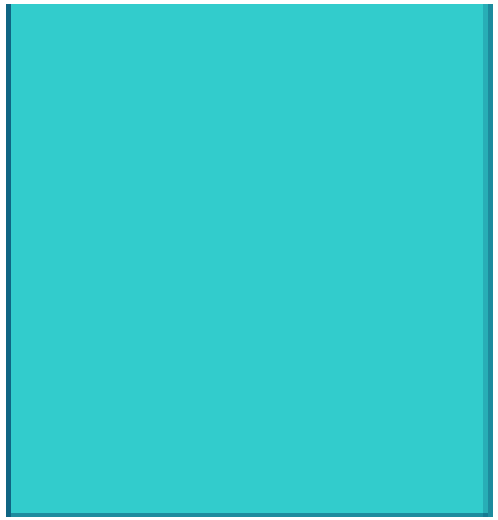
Basic Memory Management
Relocation and Protection

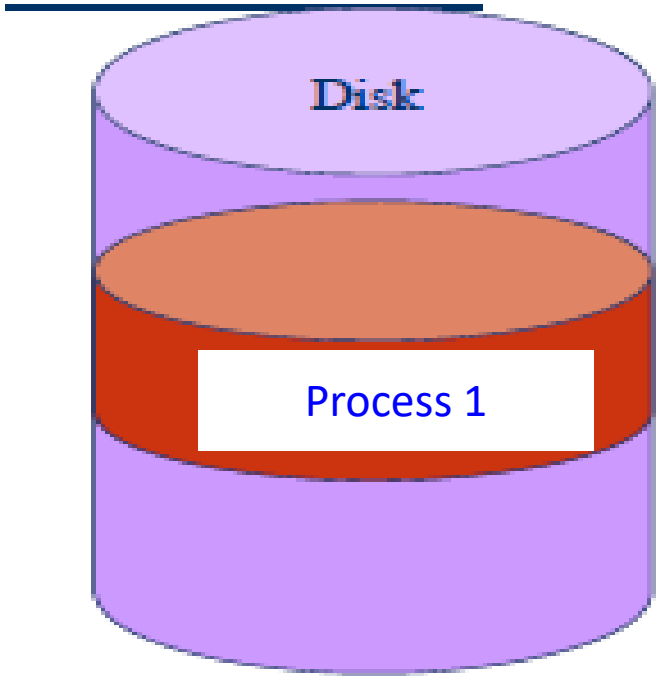
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*

Basic Memory Management
HW address protection with base and limit registers

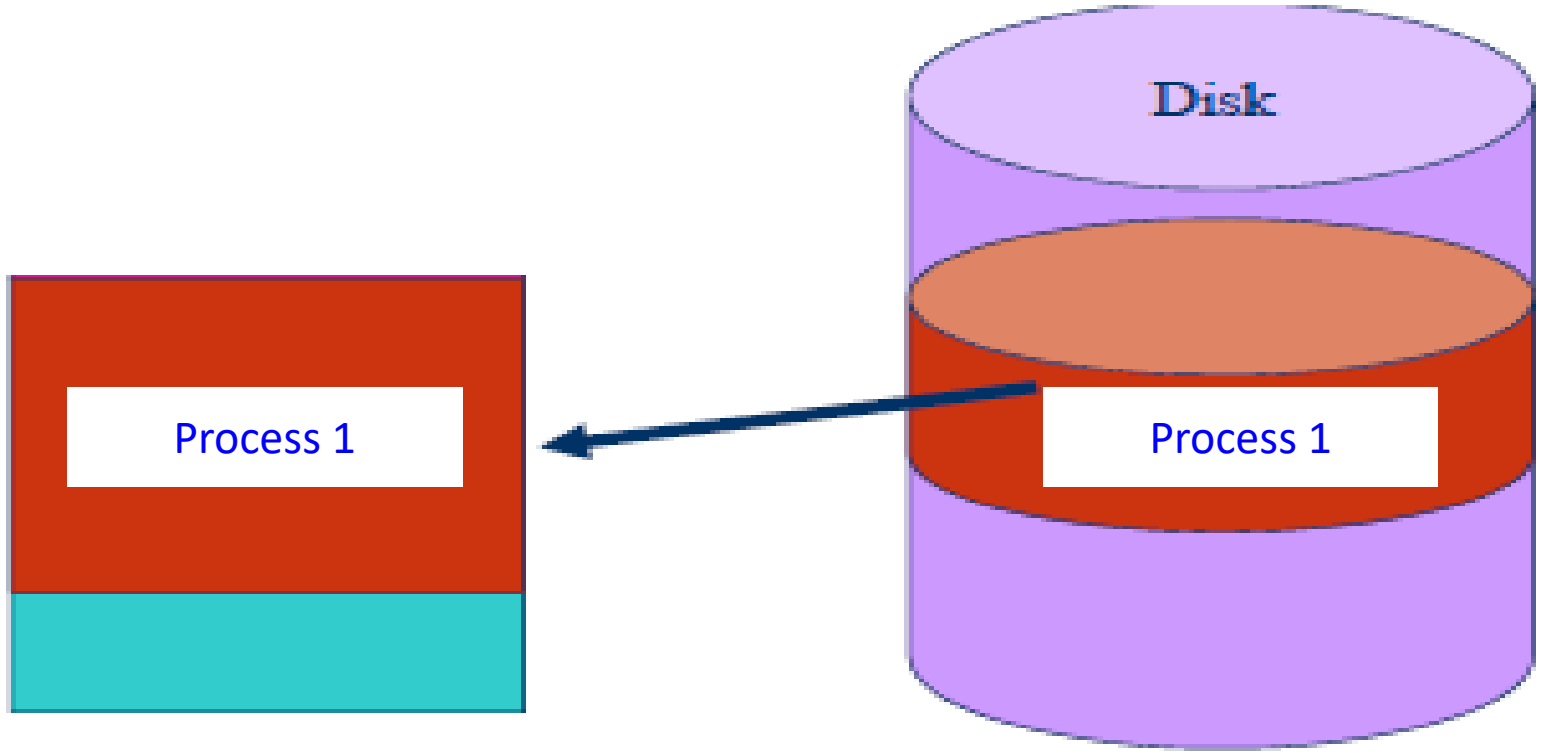


Swapping

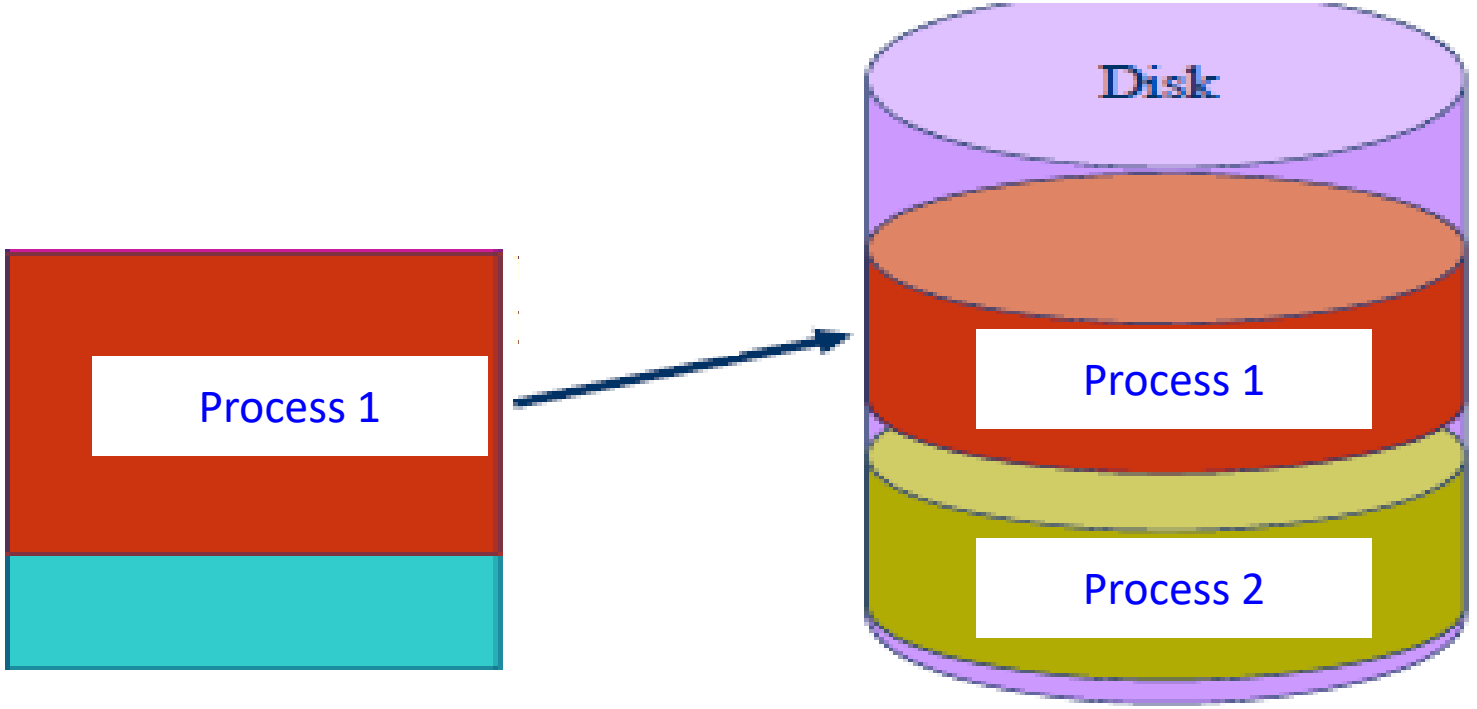




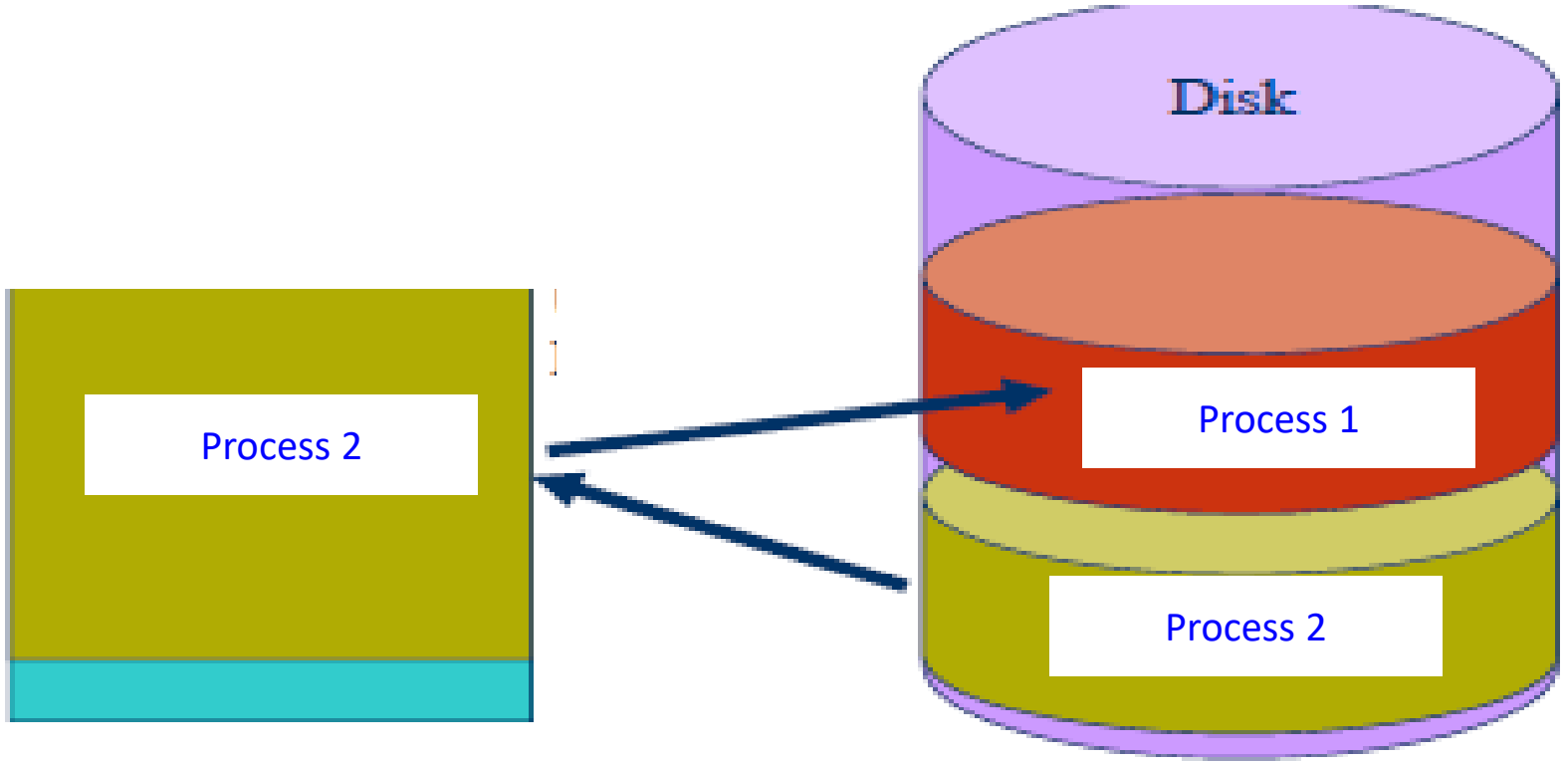
Swapping process 1's data into memory



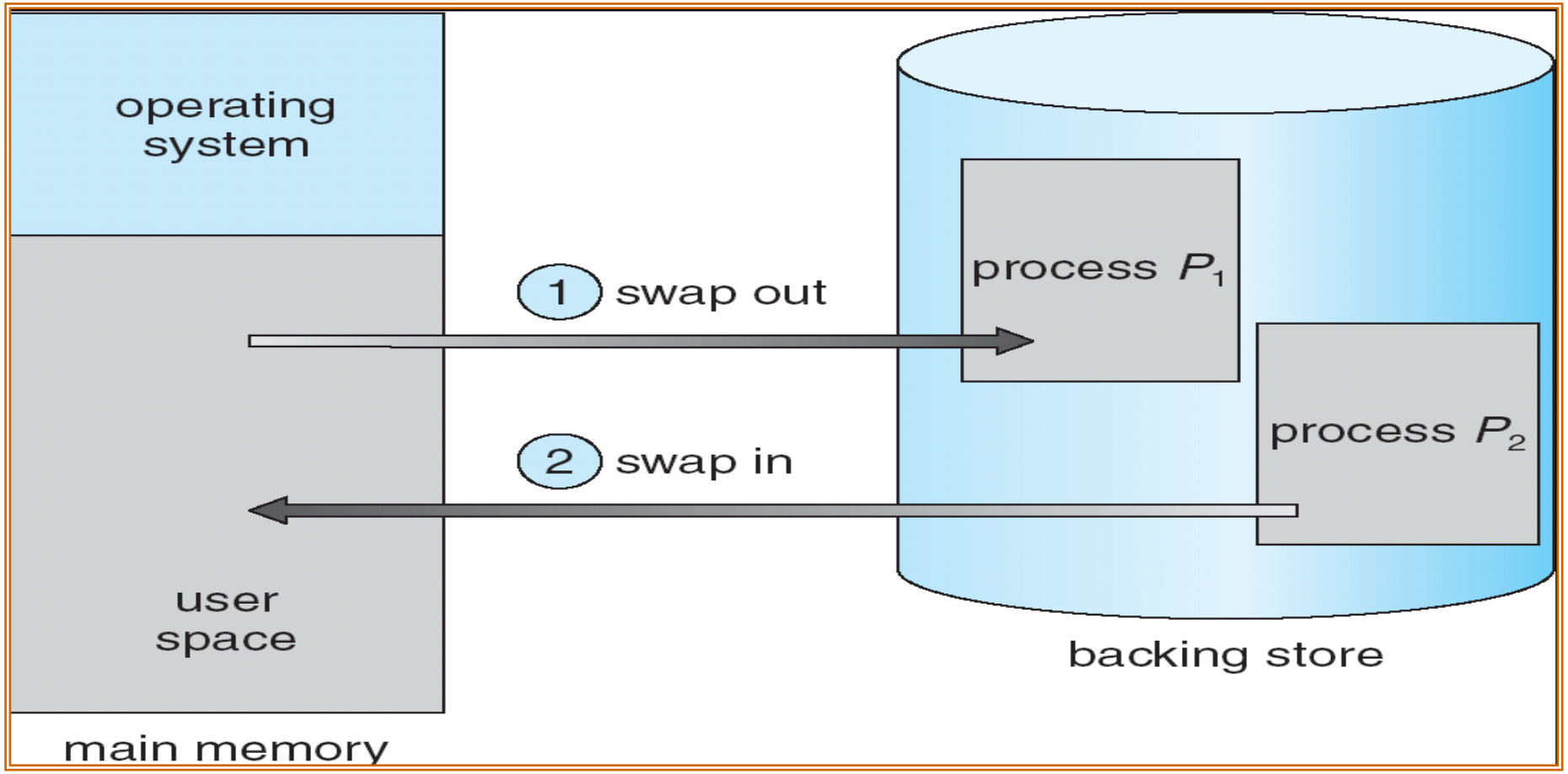
Swapping



Swapping

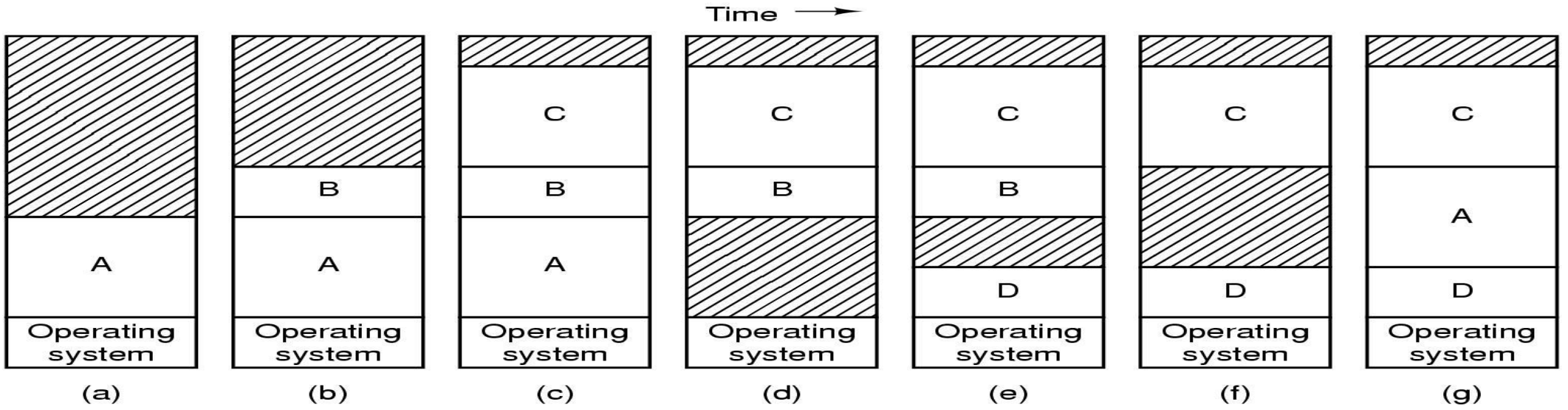


Swapping (1)
Schematic View of Swapping



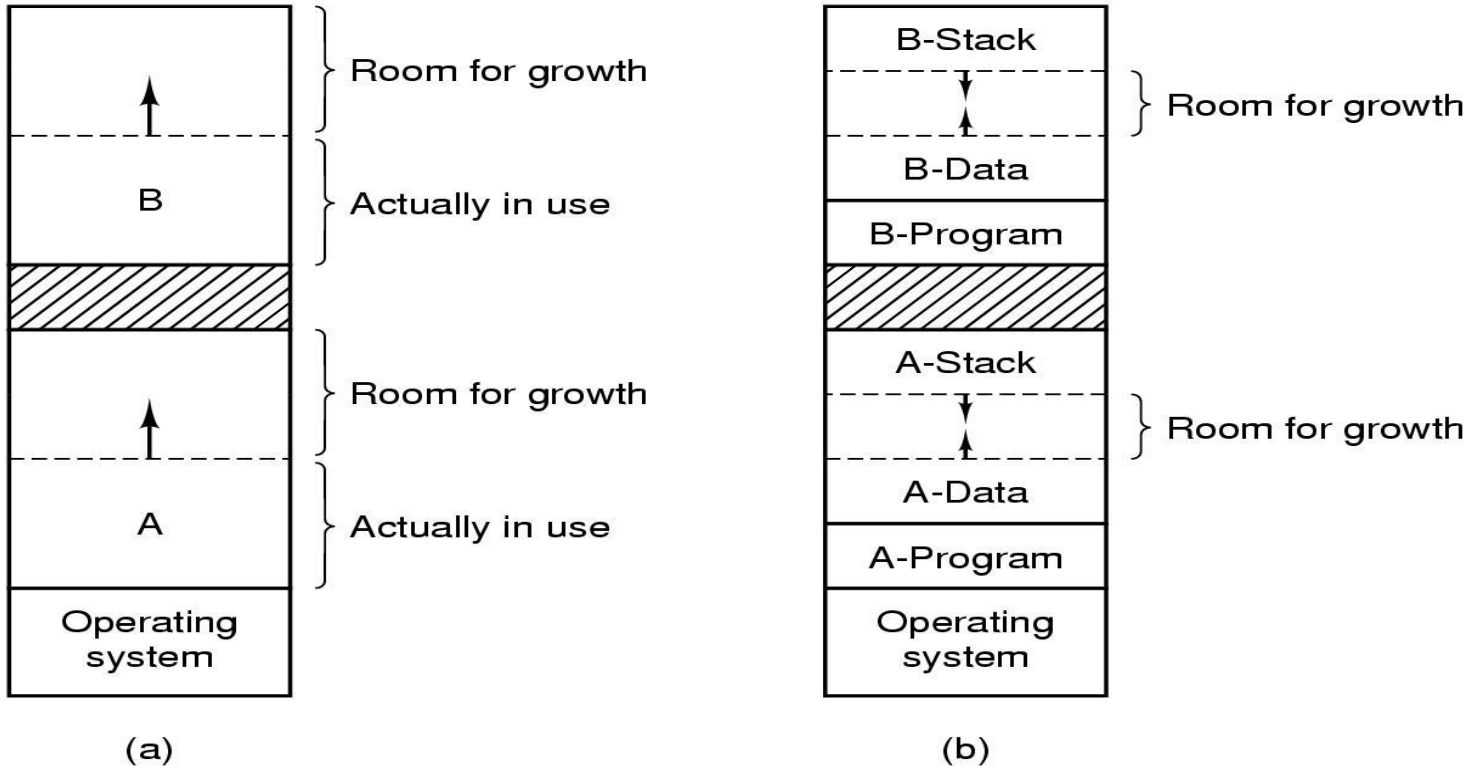
Swapping (2)

- Memory allocation changes as
  - processes come into memory
  - leave memory
- Shaded regions are unused memory
- **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



Swapping (3)

- (a) Allocating space for growing data segment
- (b) Allocating space for growing stack & data segment



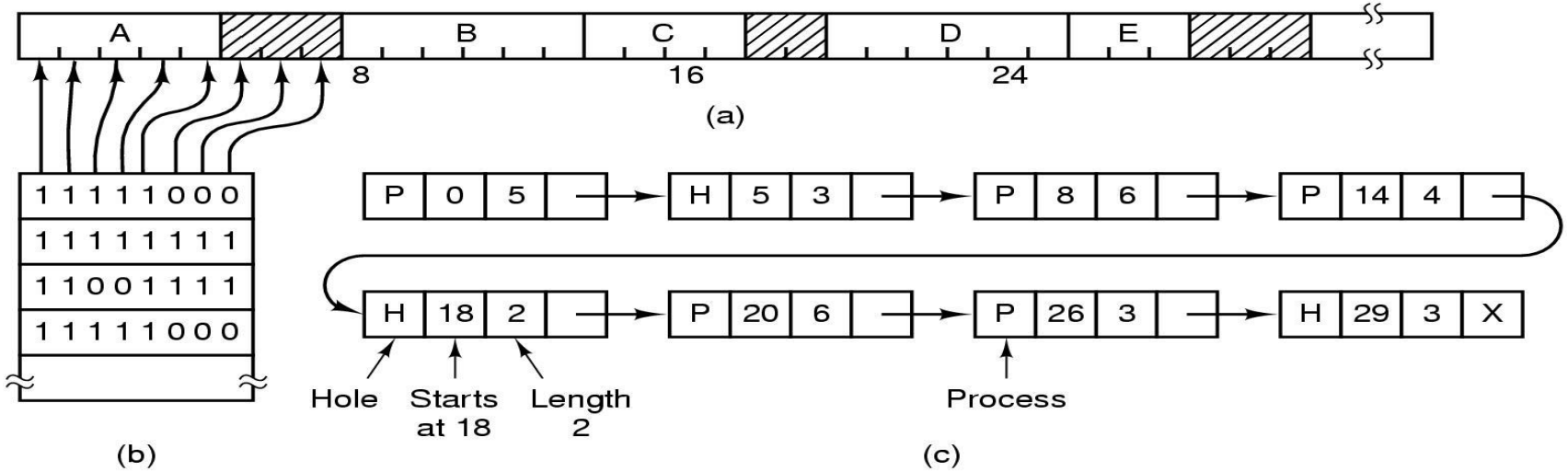
Swapping (4)
Multiple-partition allocation

- 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)
  - There are two ways to keep track of memory usages
    - ✓ Memory Management with Bit Maps
    - ✓ Memory Management with Linked Lists

Swapping (4)
Multiple-partition allocation

Memory Management with Bit Maps

- (a) Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- (b) Corresponding bit map
- (c) Same information as a list



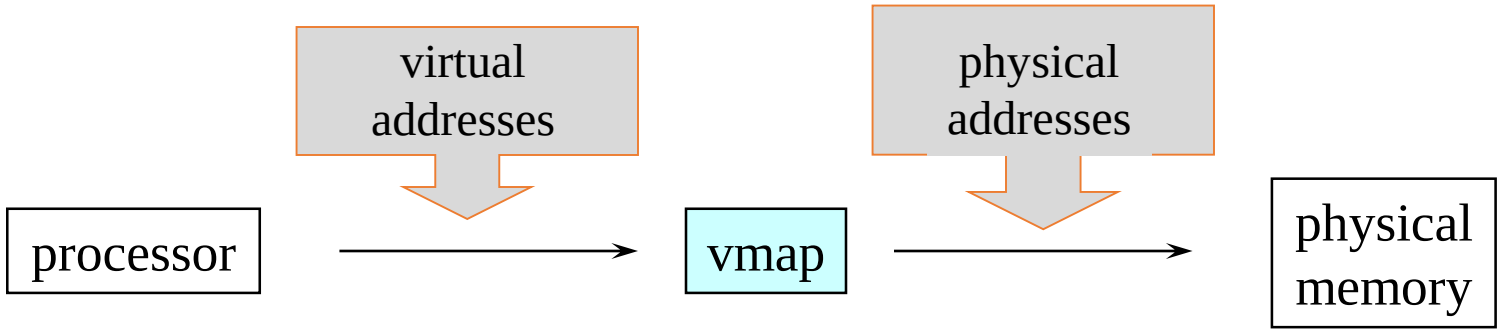
Swapping (5)
Dynamic Storage-Allocation Problem

- First-fit: Allocate the first hole that is big enough
- Next fit: Start seaching the list from the place where it left off last time
- 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

Virtual Memory
Paging

Virtual Memory Addresses

- Many ways to do this translation…
  - Start with old, simple ways, progress to current techniques

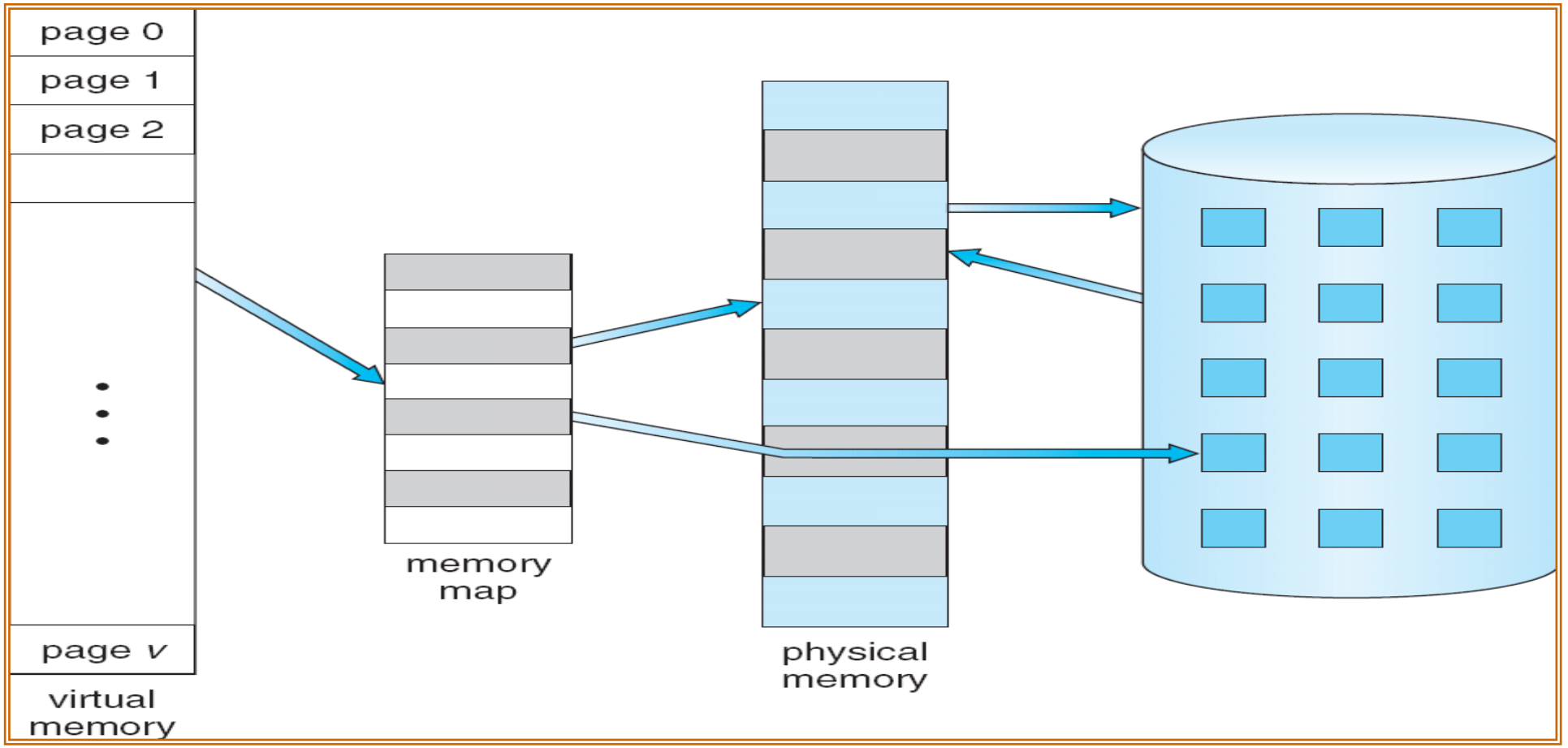


Virtual Memory

- **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**

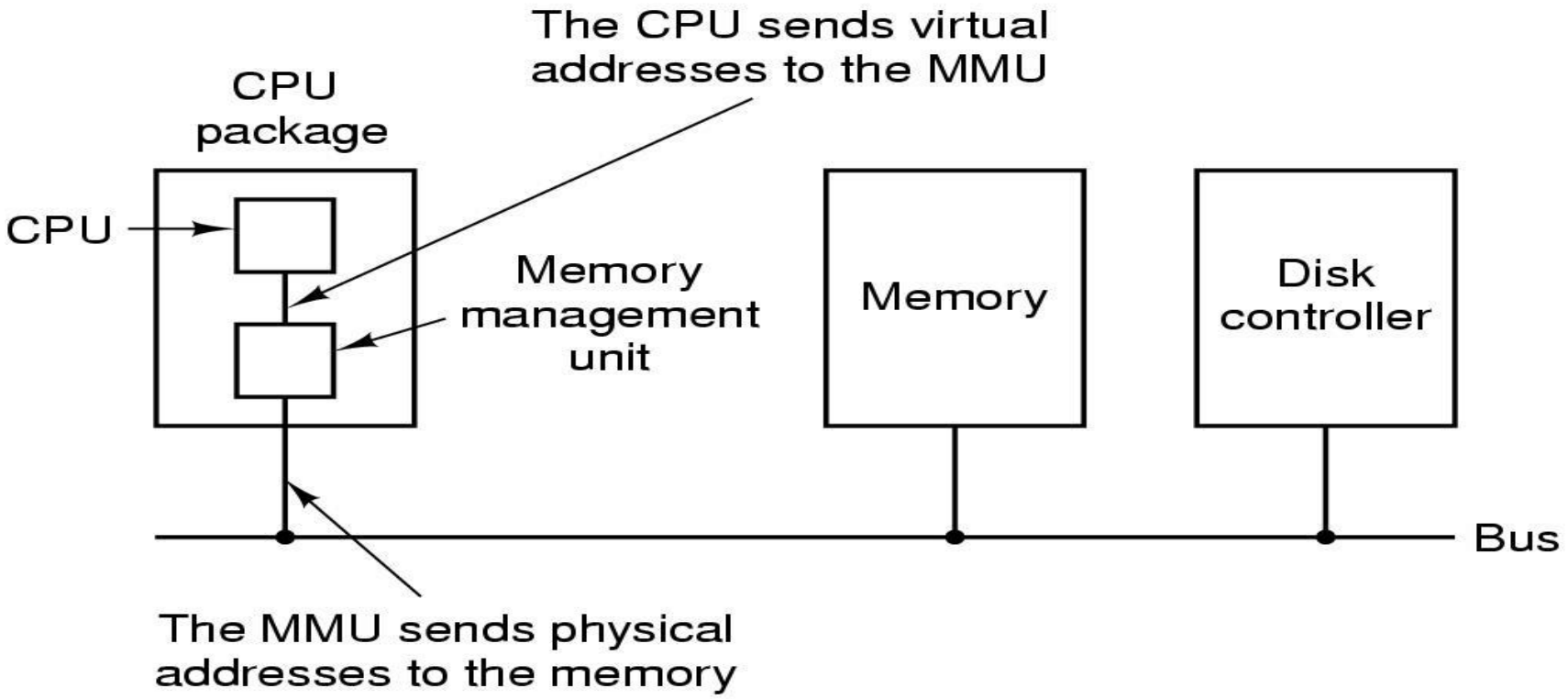
Virtual Memory

Paging



Virtual Memory
Paging

The position and function of the MMU



Virtual Memory
Paging

- Virtual 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 **Page frames** (size is power of 2, between 512 bytes and 8,192 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

Virtual Memory
Address Translation Scheme

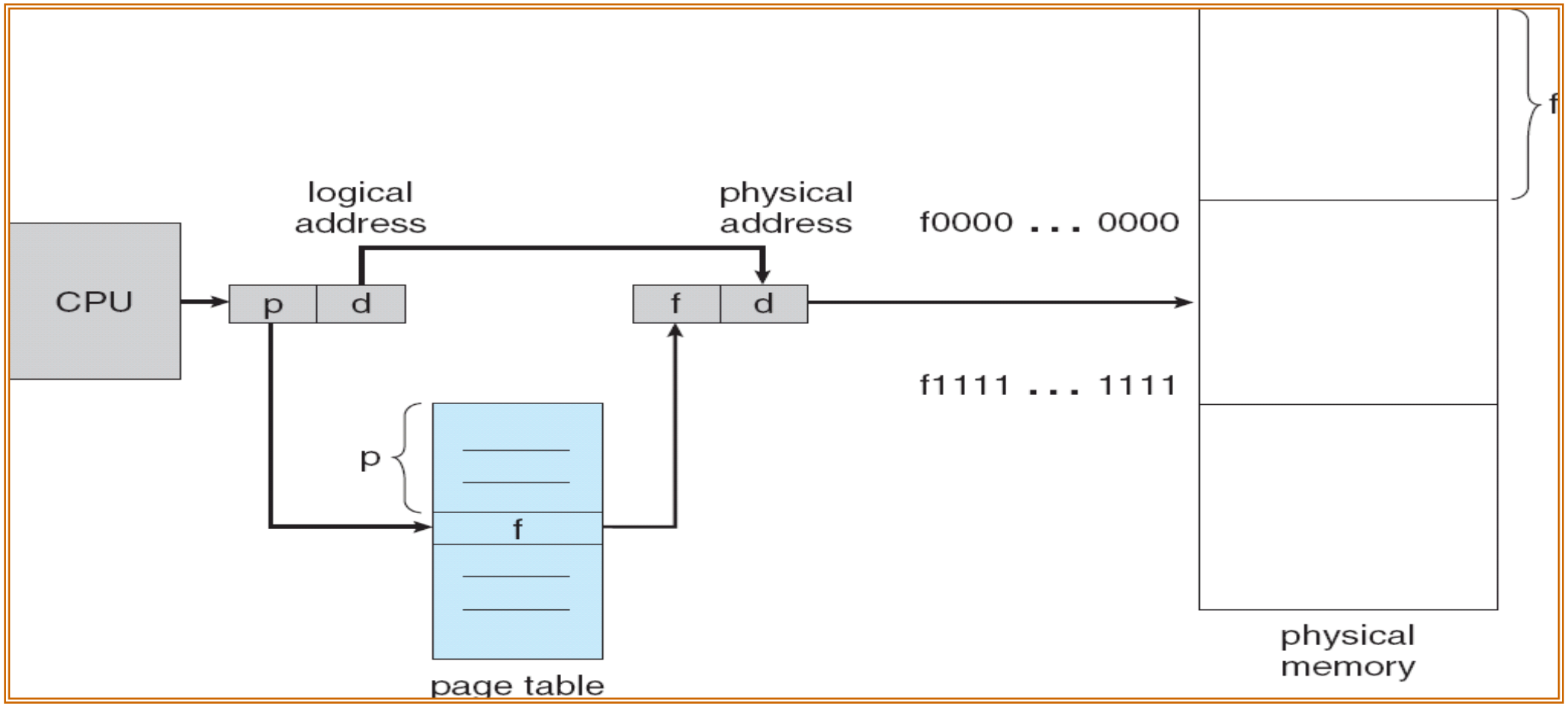
- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory

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

  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit

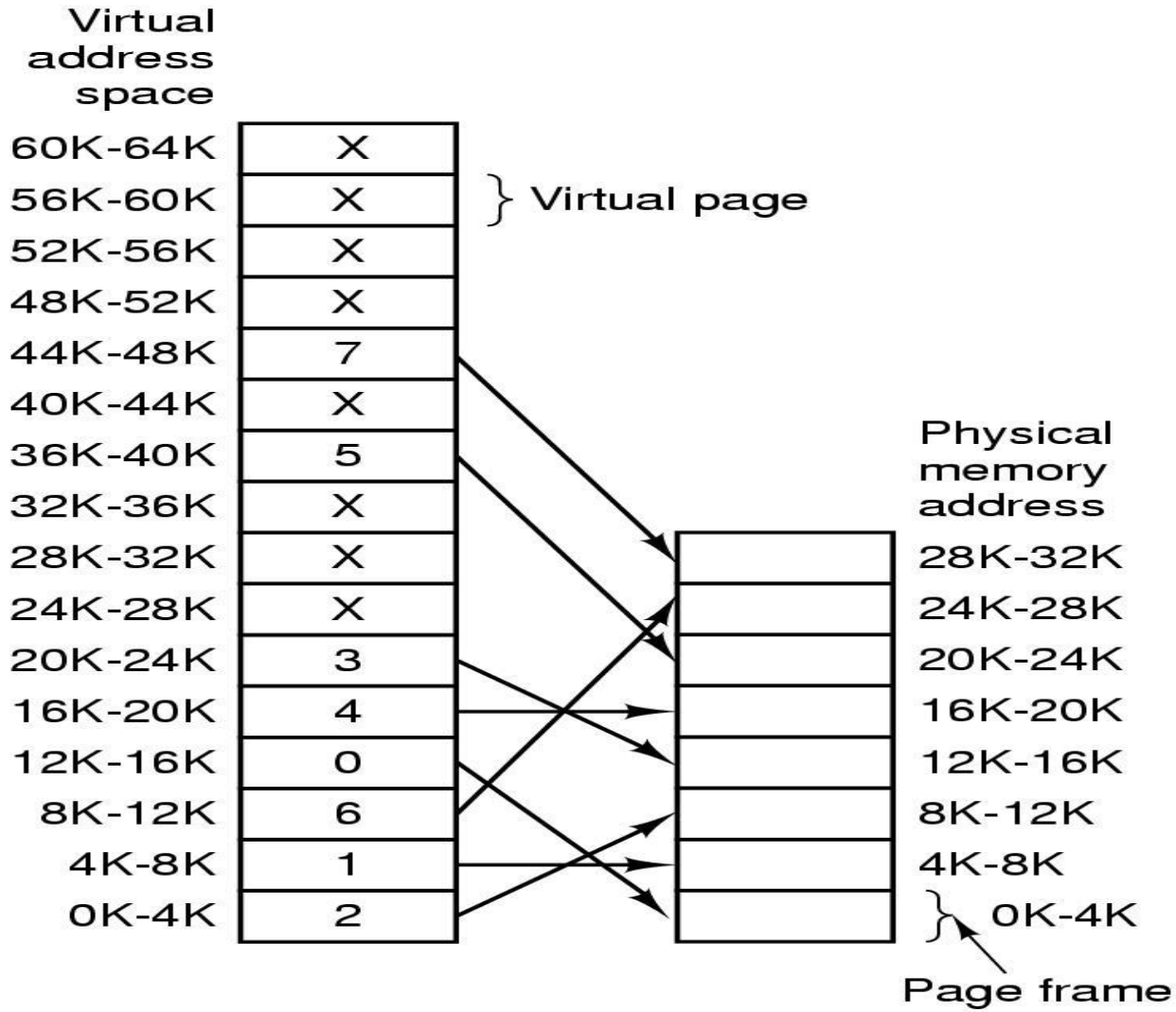
  - For given logical address space 2^m *and page size* 2^n

Virtual Memory
Paging Hardware



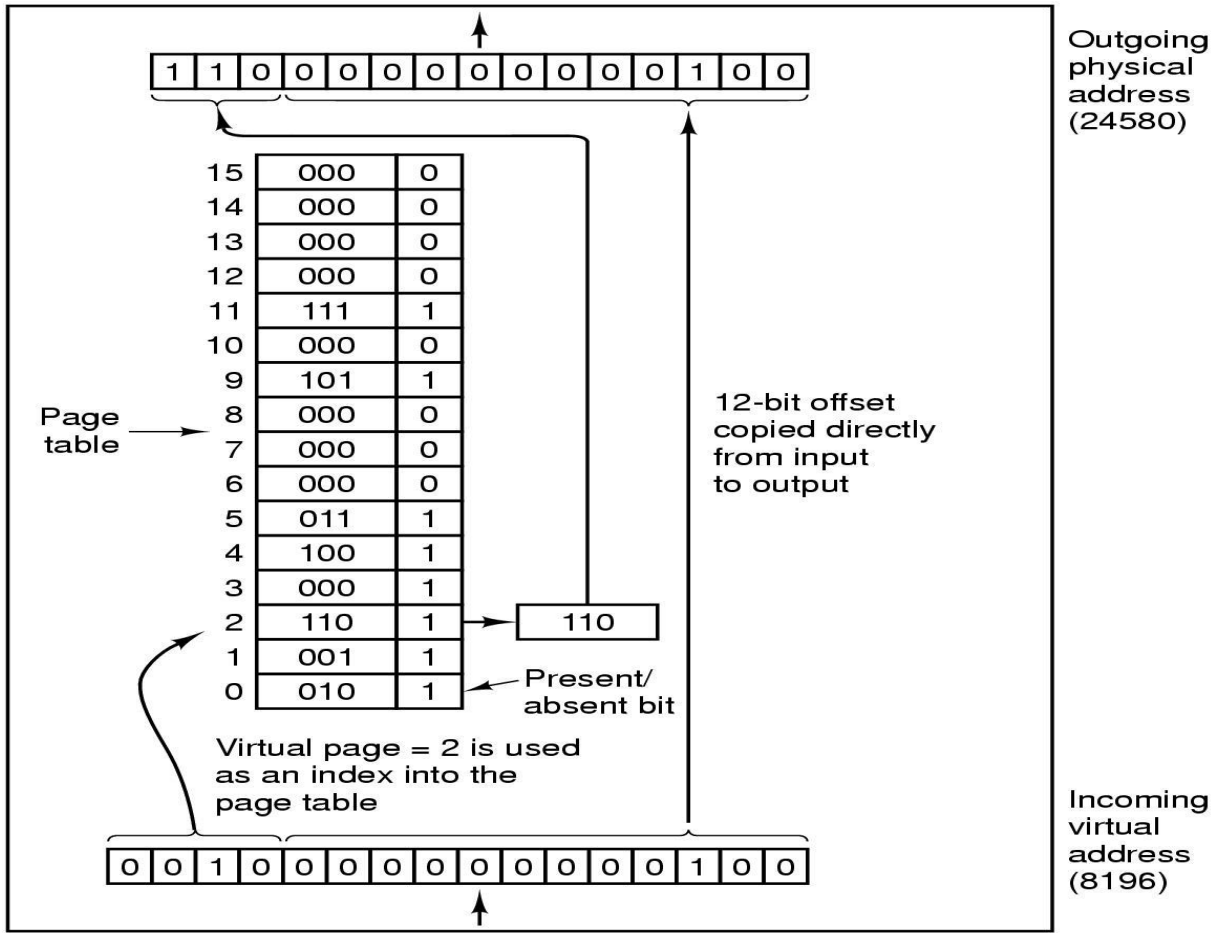
Virtual Memory
Paging: Example

The relation between virtual addresses and physical memory addresses given by page table



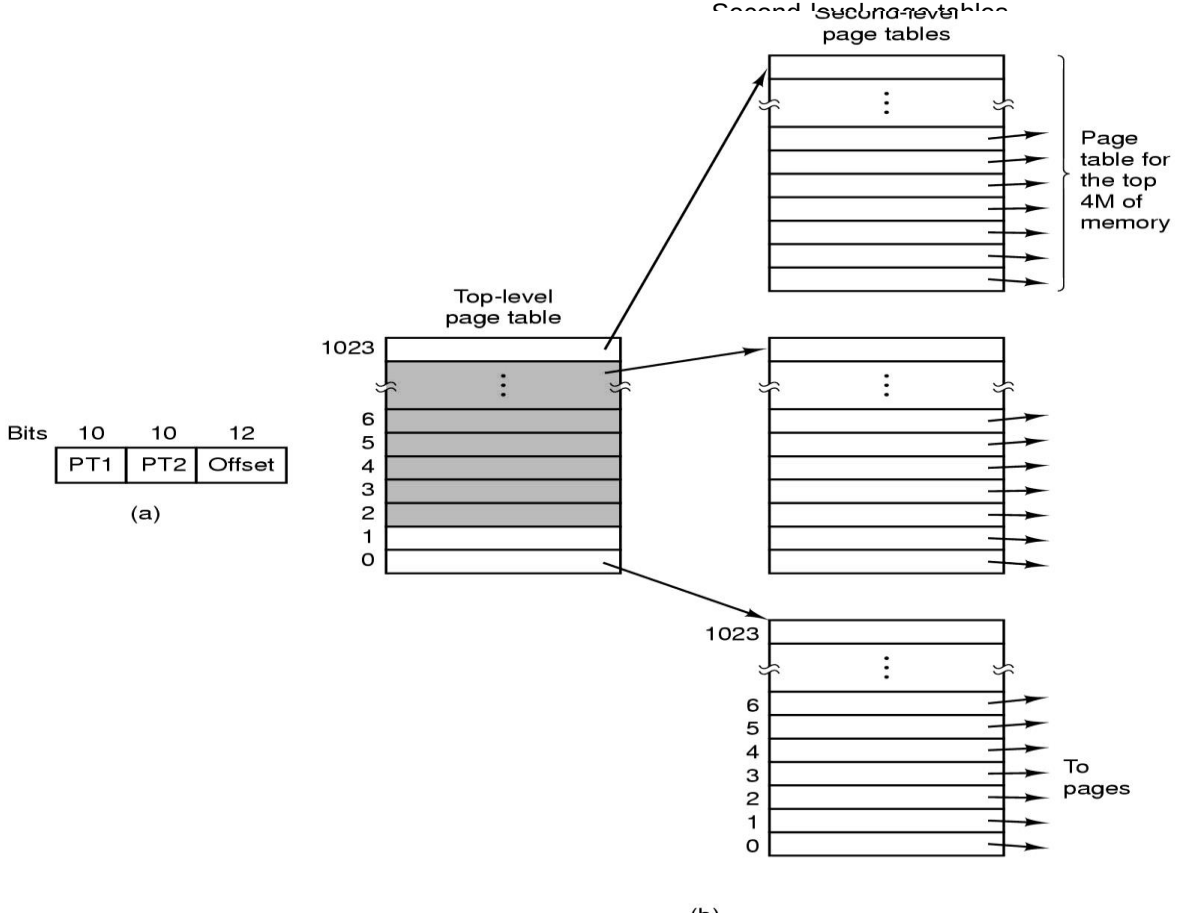
Virtual Memory
Page Tables: Example

Internal operation of MMU with 16 4 KB pages



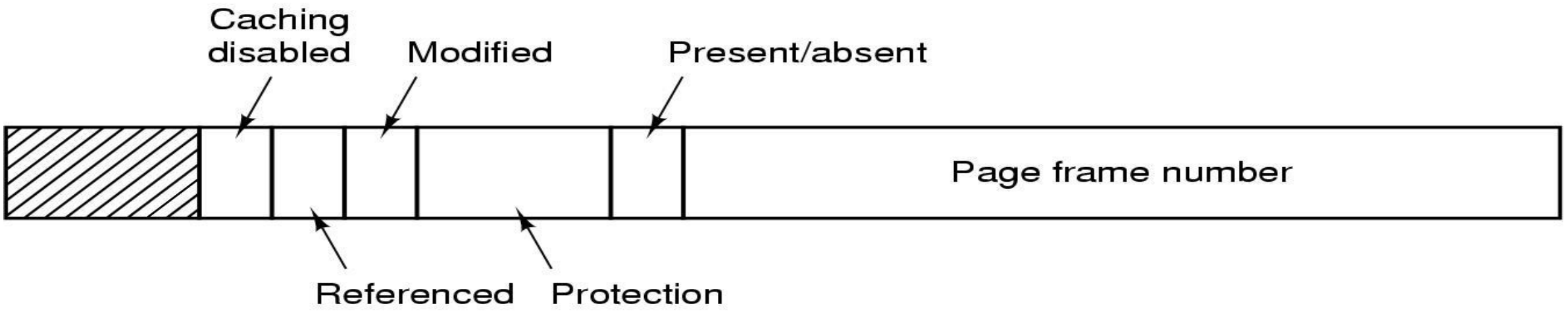
Virtual Memory
Two-level page tables

- 32 bit address with 2 page table fields
- Two-level page tables



Virtual Memory
Typical page table entry

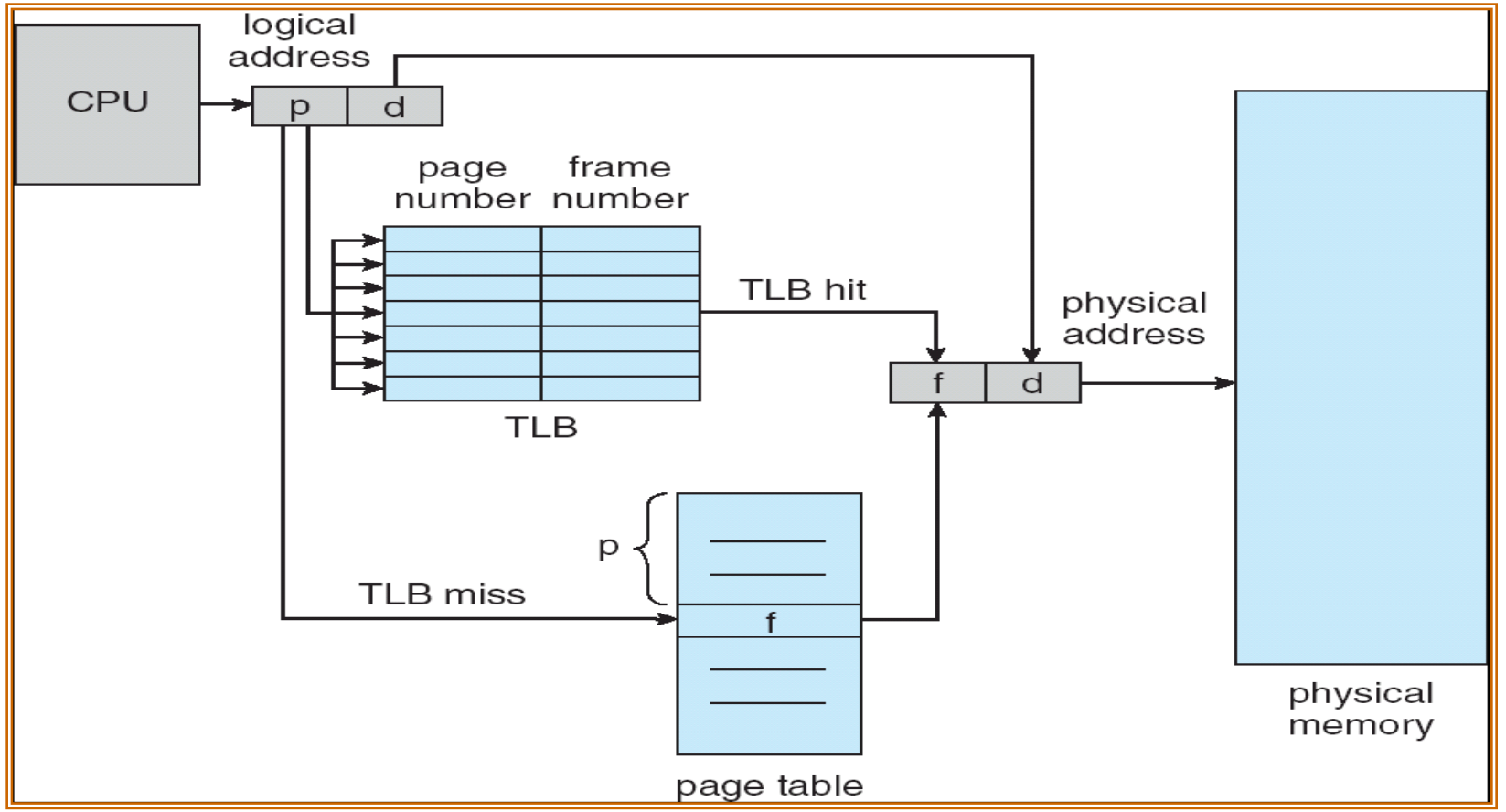
Typical page table entry



Virtual Memory
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)**

Virtual Memory
Paging Hardware With TLB



Virtual Memory
TLBs – Translation Lookaside Buffers

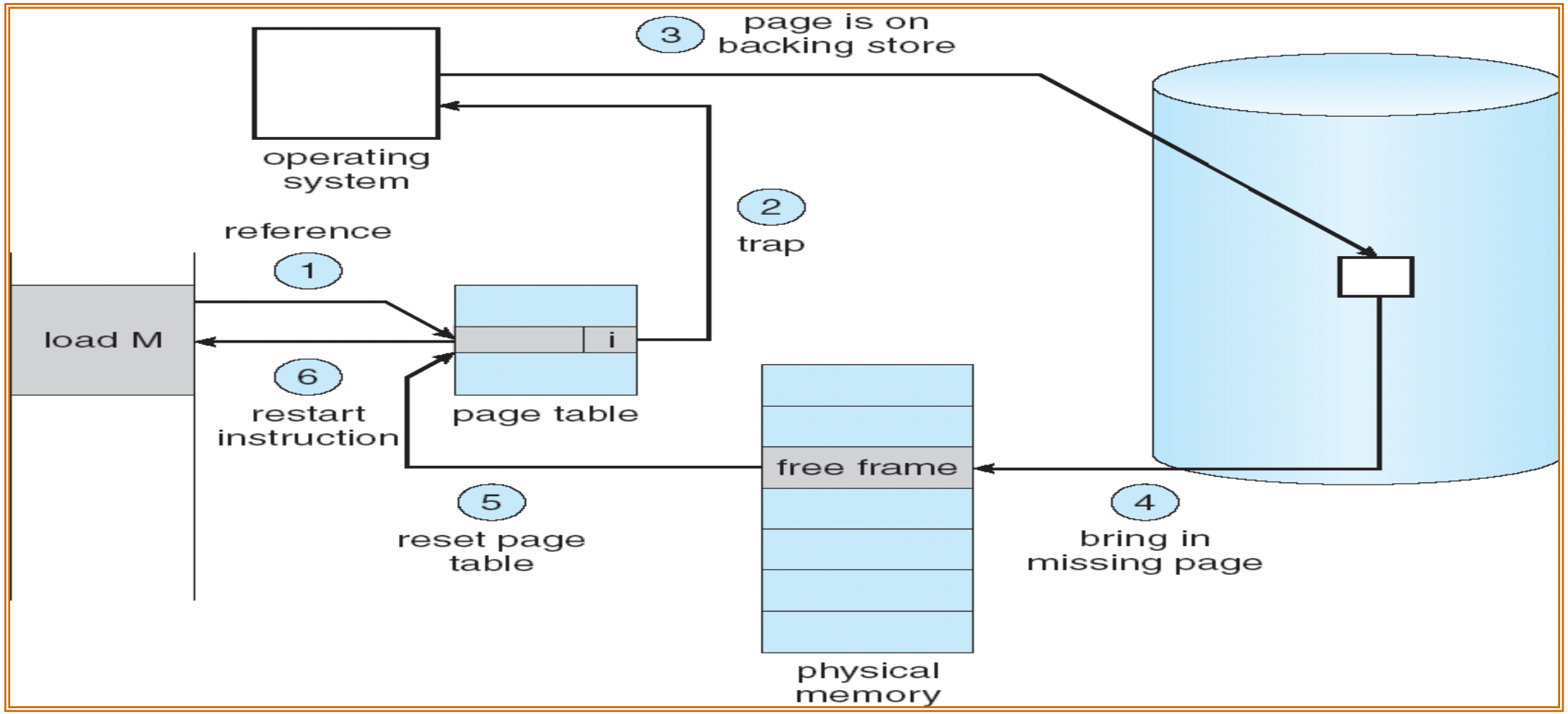
A TLB to speed up paging

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Virtual Memory
Page Fault

1. If there is a reference to a page, Just not in memory: **page fault,**
2. Trap to operating system:
3. Get empty page frame, Determine page on backing store
4. Swap page from disk into page frame in memory
5. Modifies page tables, Set validation bit = **v**
6. Restart the instruction that caused the page fault

Virtual Memory
Steps in Handling a Page Fault



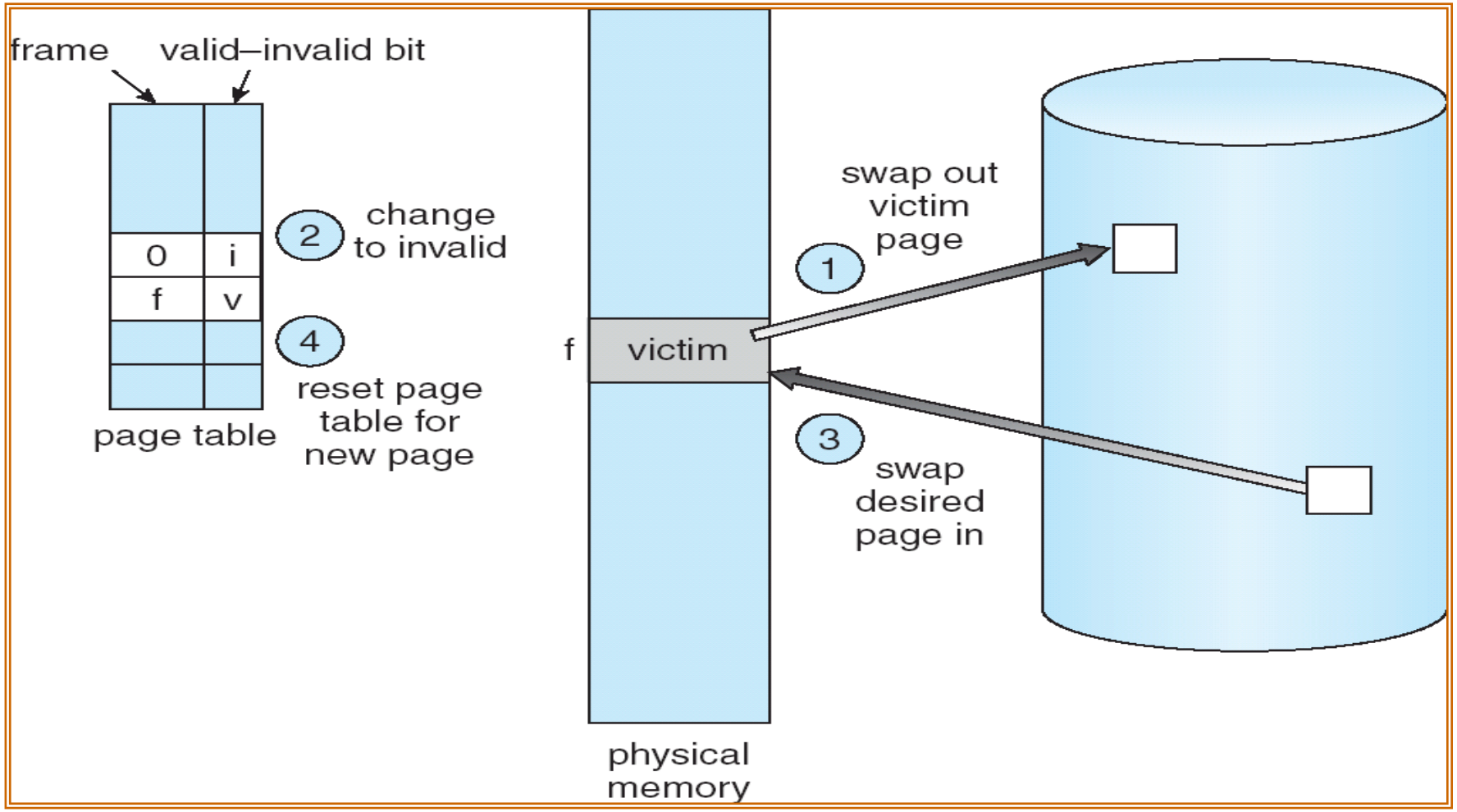
Virtual Memory
Page Replacement Algorithms

- 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

Virtual Memory
Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Restart the process

Virtual Memory
Page Replacement



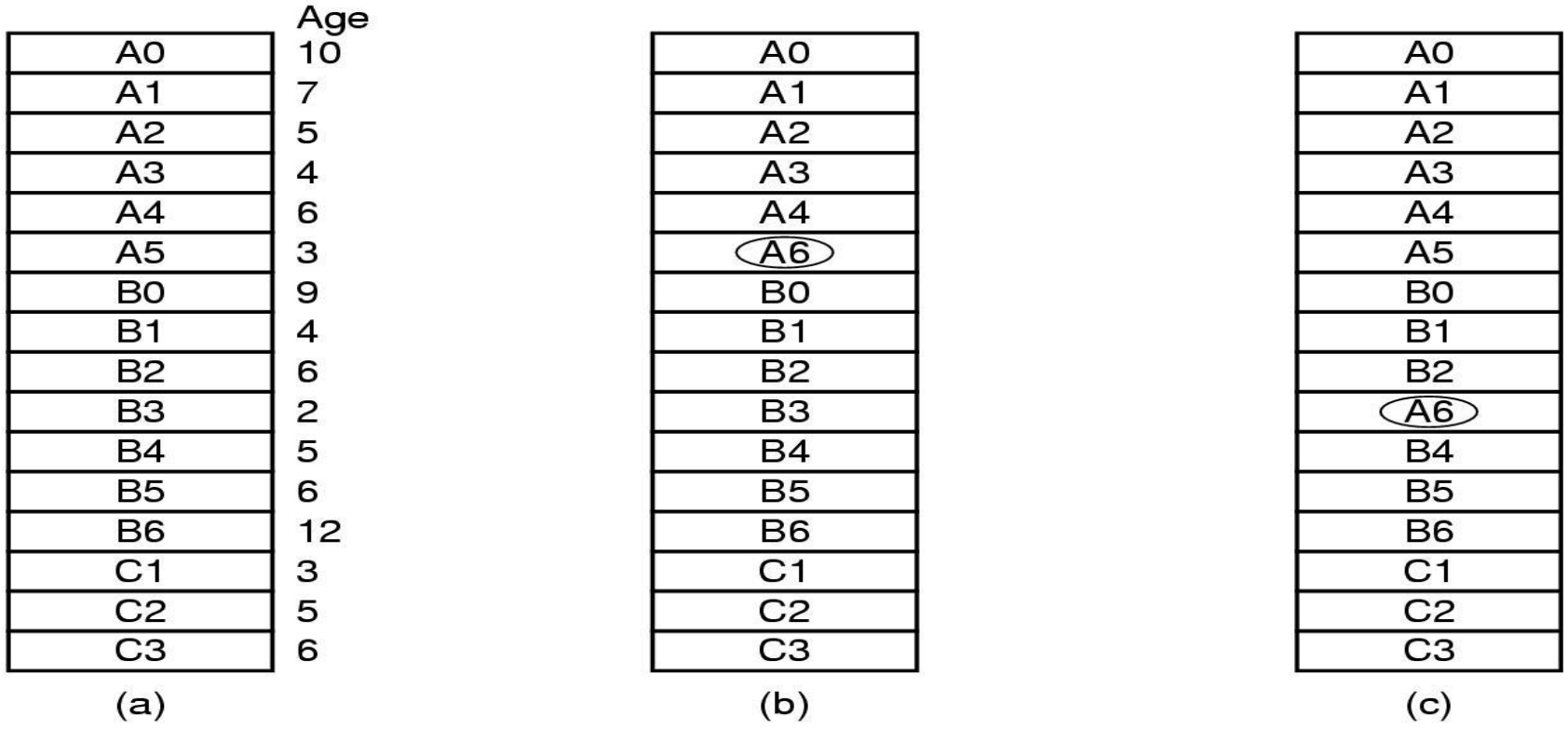
Virtual Memory
Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Virtual Memory

Design issues: Local versus Global Allocation Policies

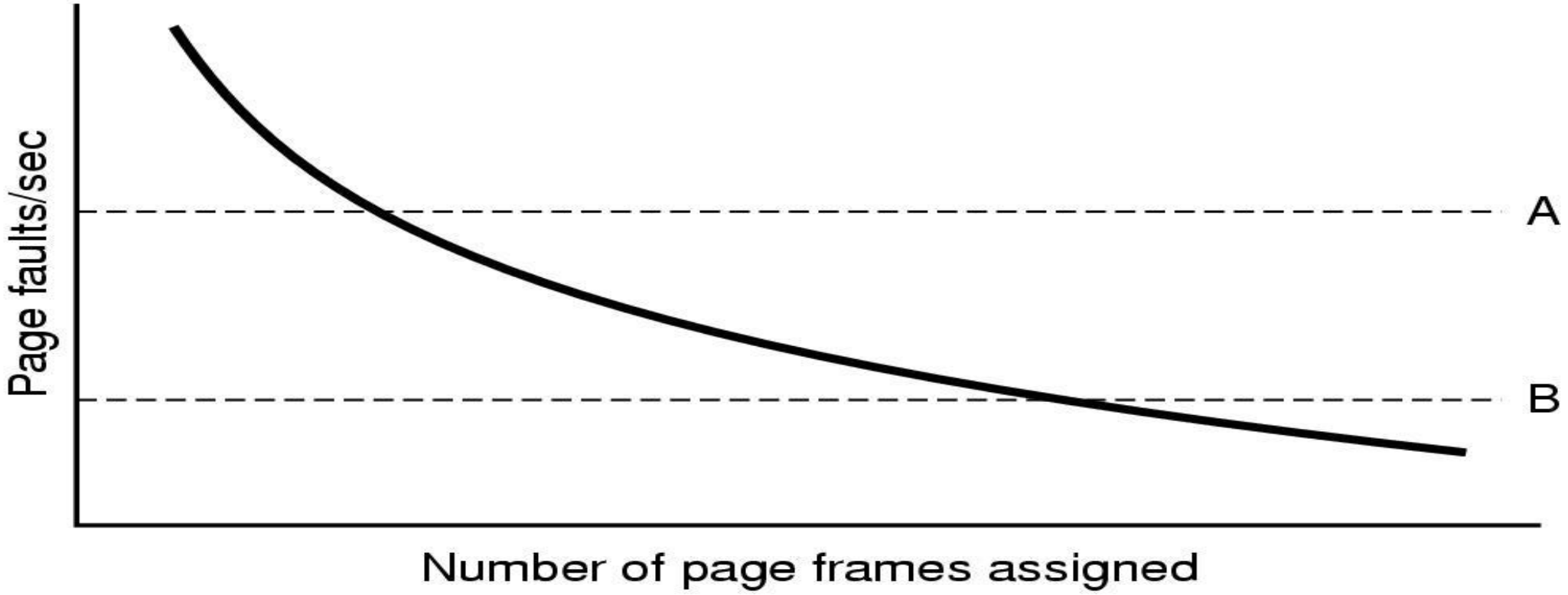
- (a) Original configuration
- (b) Local page replacement
- (c) Global page replacement



Virtual Memory

Design issues: Page Frame Allocation

Page fault rate as a function of the number of page frames assigned



Virtual Memory

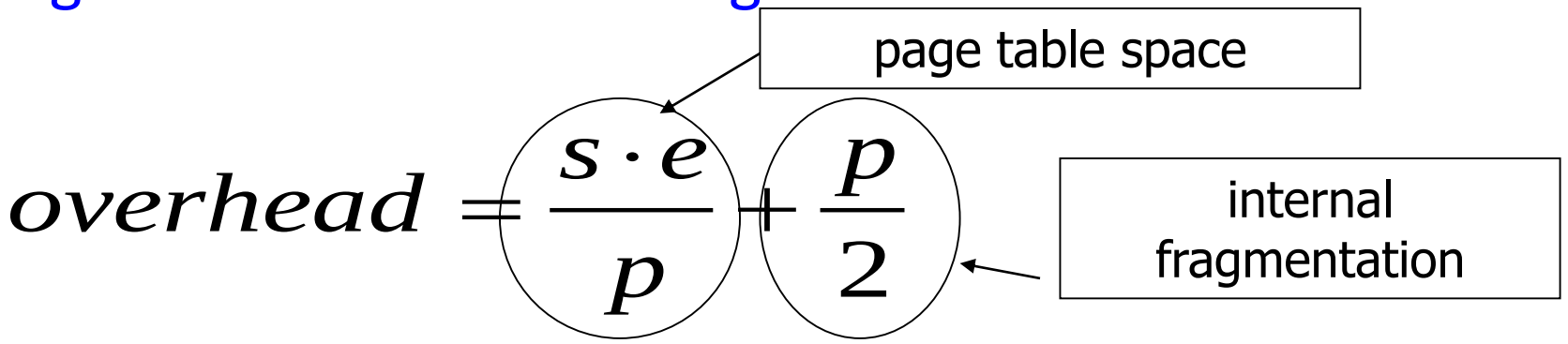
Design issues: Page Size (1)

Small page size

- Advantages
  - less internal fragmentation
  - better fit for various data structures, code sections
  - less unused program in memory
- Disadvantages
  - programs need many pages, larger page tables

Virtual Memory
Design issues: Page Size (2)

- Overhead due to page table and internal fragmentation



- Where
  - s = average process size in bytes
  - p = page size in bytes
  - e = page entry in bytes

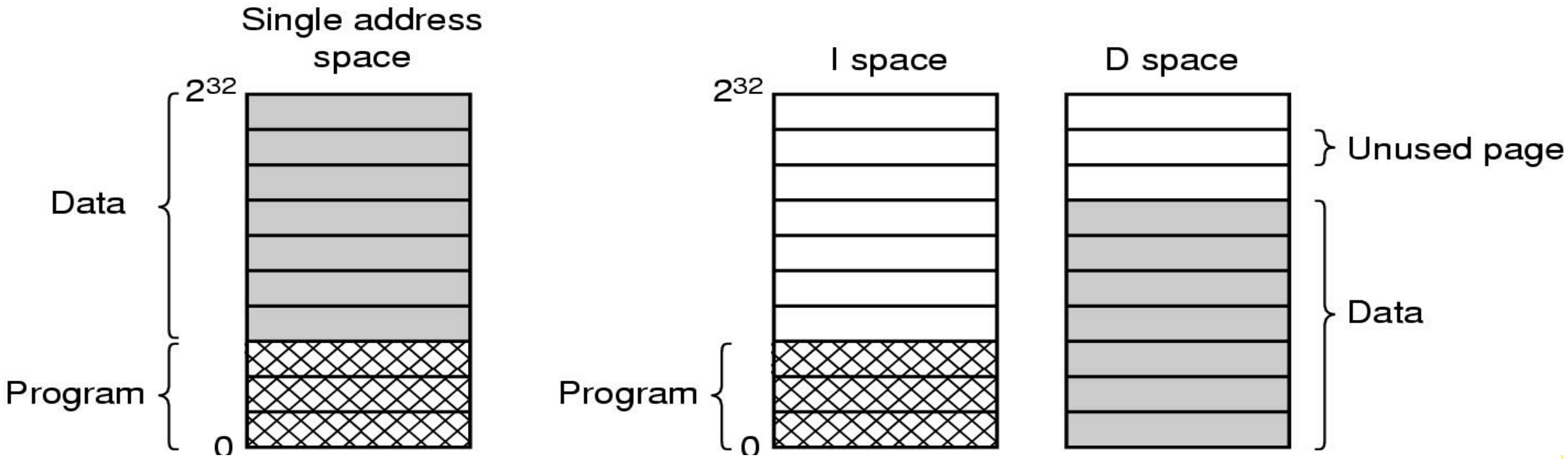
Optimized when

$$p = \sqrt{2se}$$

Virtual Memory

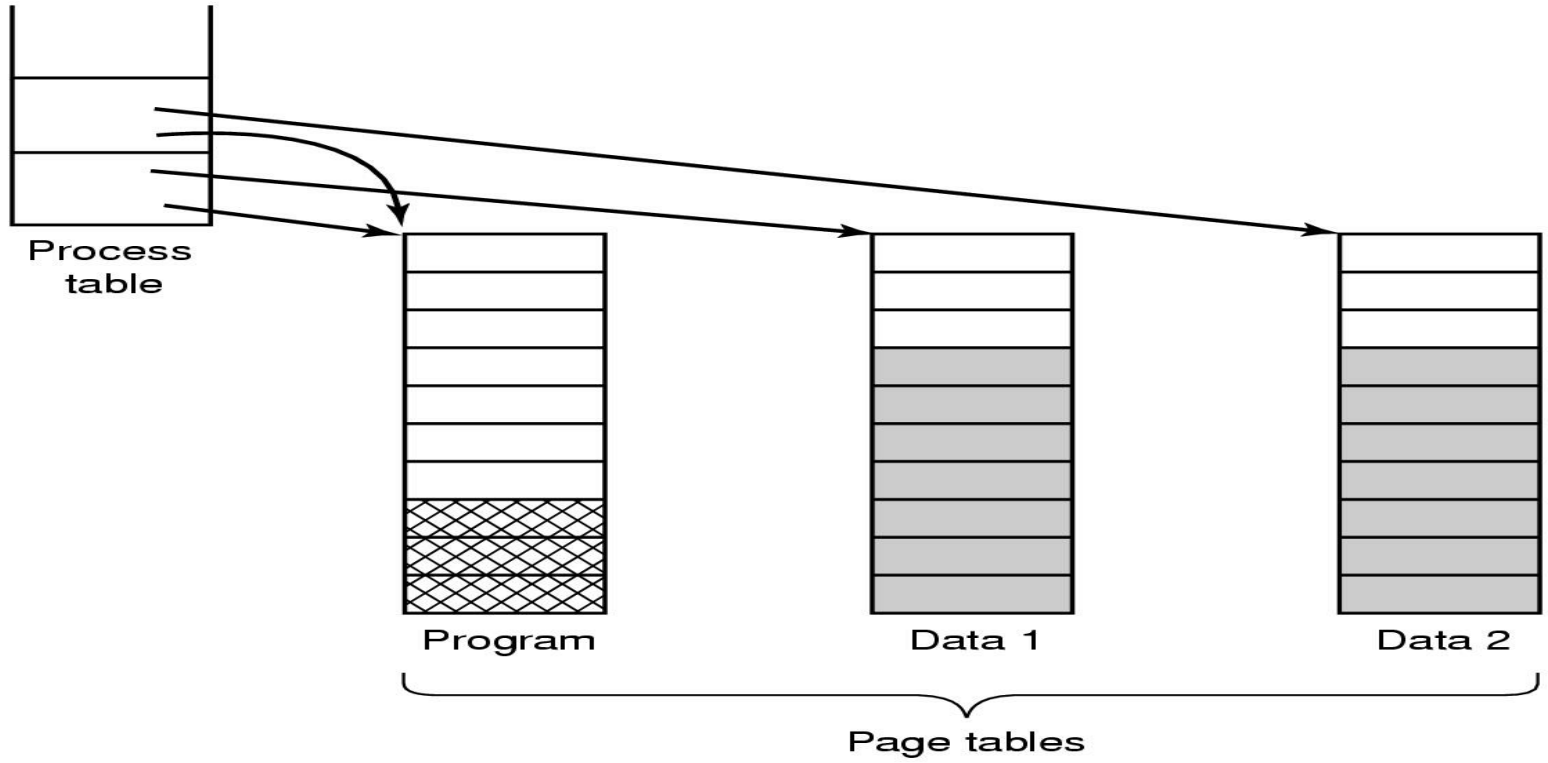
Design issues: Separate Instruction and Data Spaces

- One address space
- Separate I and D spaces



Virtual Memory
Design issues: Shared Pages

Two processes sharing same program sharing its page table



Virtual Memory
Design issues: Cleaning Policy

- Need for a background process, paging daemon
  - periodically inspects state of memory
- When too few frames are free
  - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
  - as regular page replacement algorithm but with diff ptr

Virtual Memory
Implementation Issues
Operating System Involvement with Paging

Four times when OS involved with paging

1. Process creation
   - determine program size
   - create page table
2. Process execution
   - MMU reset for new process
   - TLB flushed
3. Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
4. Process termination time
   - release page table, pages

Virtual Memory
Implementation Issues
Page Fault Handling (1)

1. Hardware traps to kernel
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of address, seeks page frame
5. If selected frame is dirty, write it to disk

Virtual Memory
Implementation Issues
Page Fault Handling (2)

6. OS brings schedules new page in from disk
7. Page tables updated
8. Faulting instruction backed up to when it began
9. Faulting process scheduled
10. Registers restored, Program continues

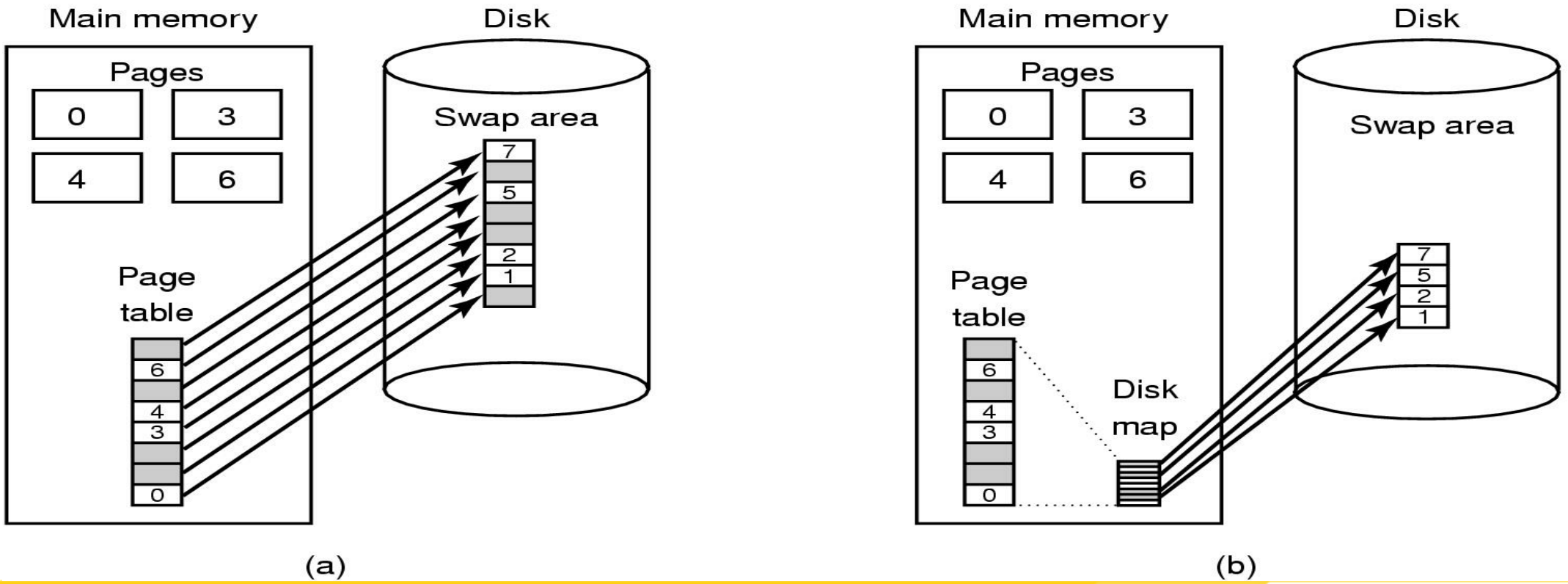
Virtual Memory
Implementation Issues
Locking Pages in Memory

- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first proc may be chosen to be paged out
- Need to specify some pages locked
  - exempted from being target pages

Virtual Memory
Implementation Issues: Backing Store

(a) Paging to static swap area

(b) Backing up pages dynamically

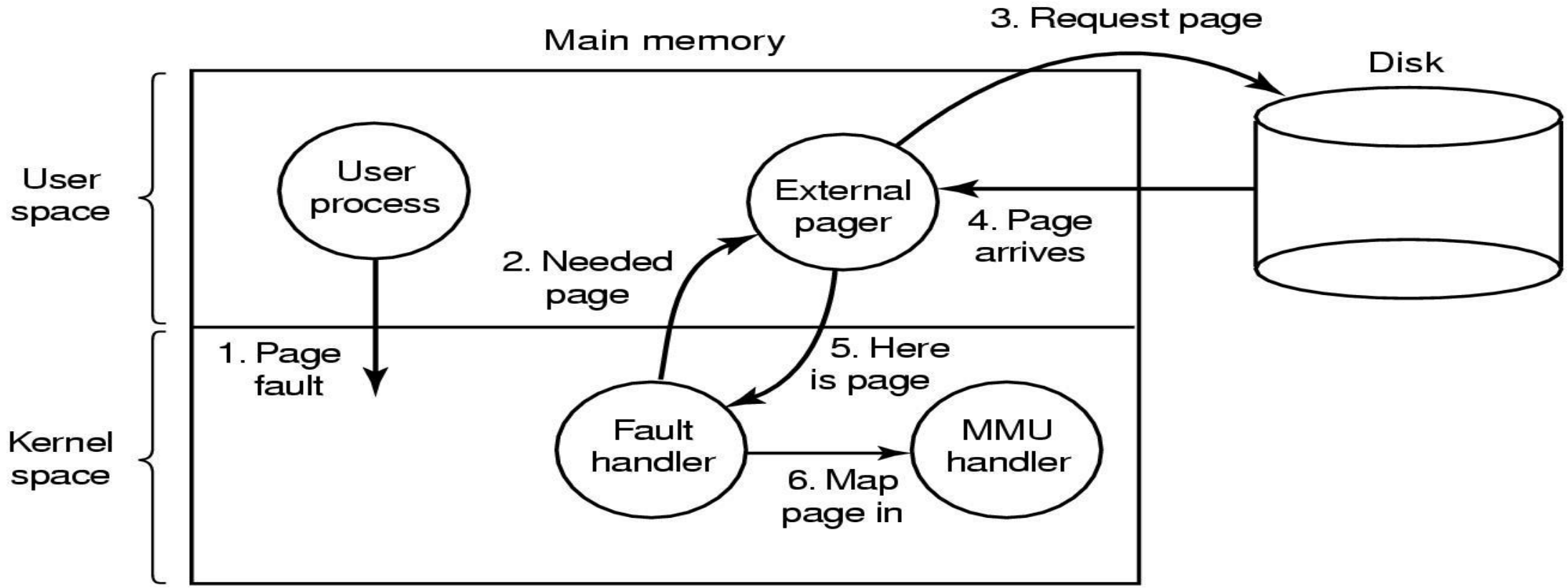


Virtual Memory

Implementation Issues

Separation of Policy and Mechanism

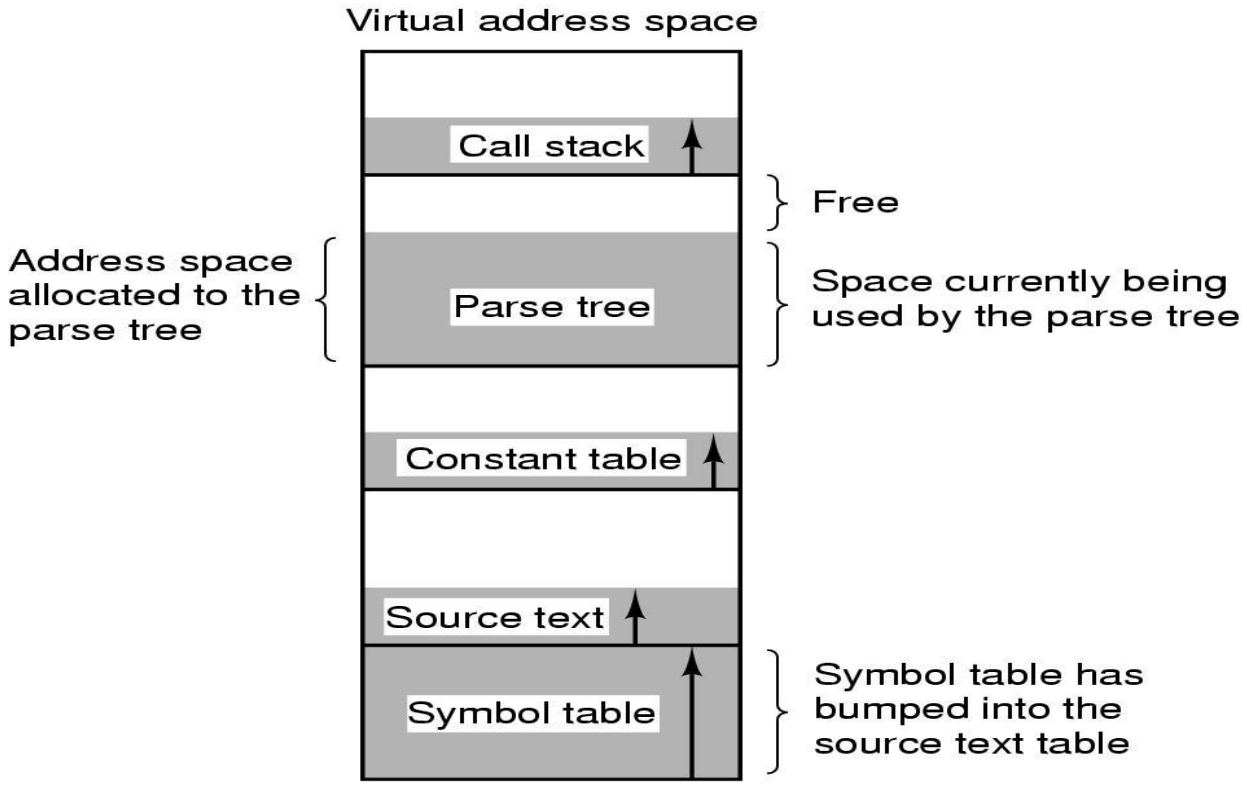
Page fault handling with an external pager



Virtual Memory Segmentation

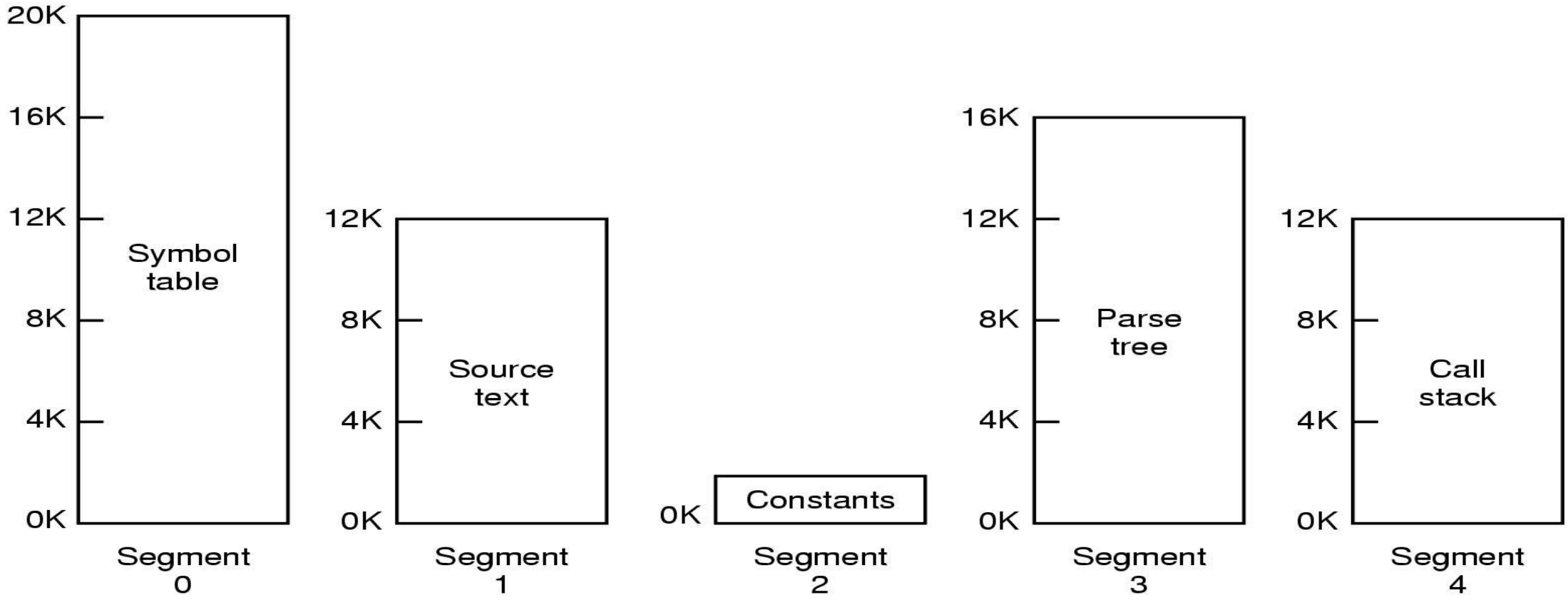
Virtual Memory
Segmentation (1)

- One-dimensional address space with growing tables
- One table may bump into another



Virtual Memory
Segmentation (2)

Allows each table to grow or shrink, independently



Virtual Memory
Segmentation (3)

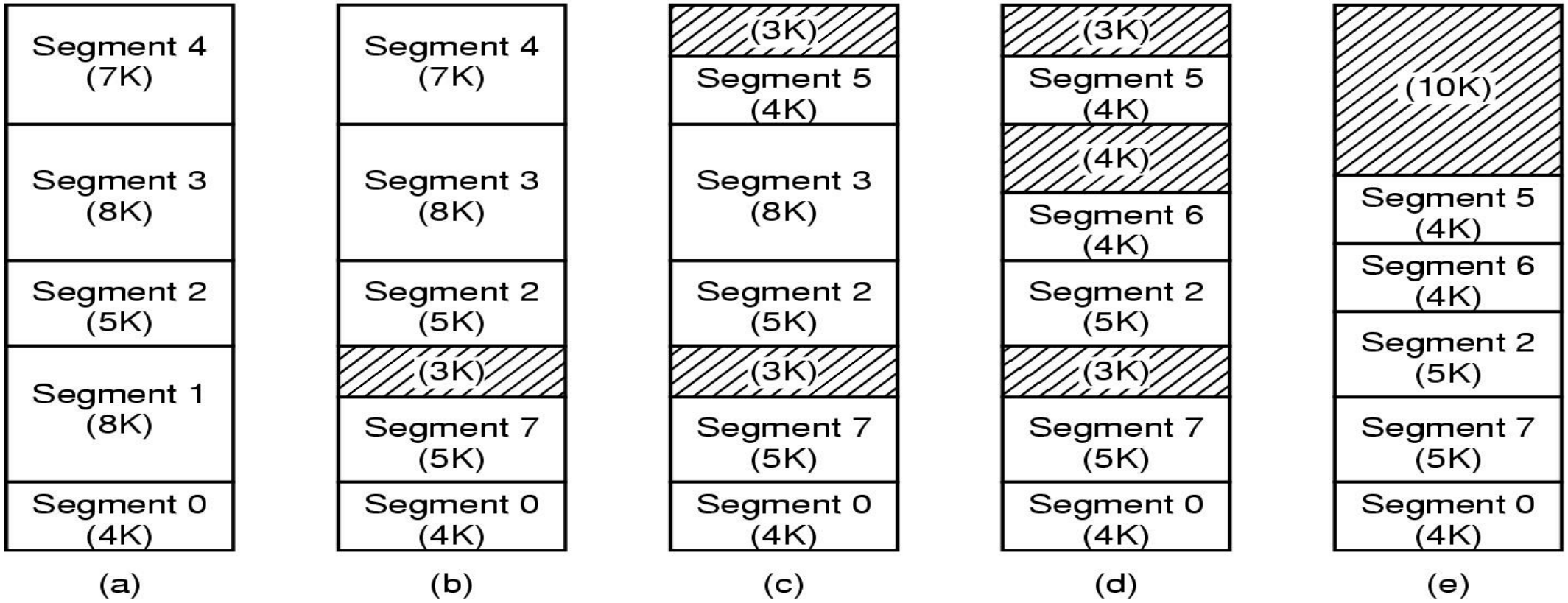
Comparison of paging and segmentation

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

Virtual Memory
Implementation of Pure Segmentation (4)

(a)-(d) Development of checkerboarding

(e) Removal of the checkerboarding by compaction

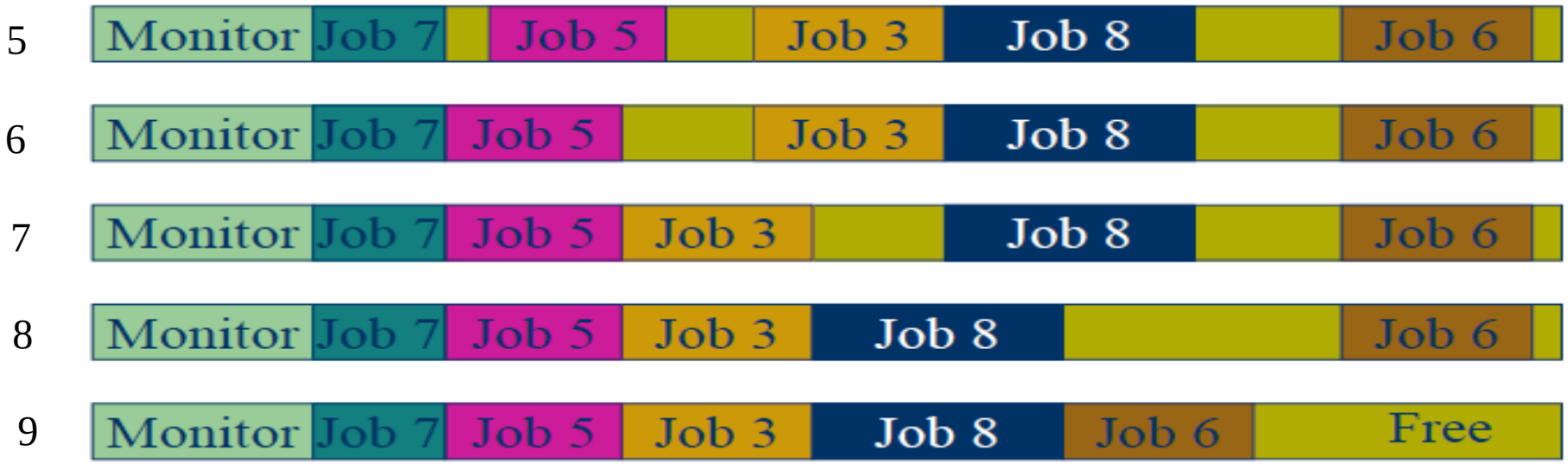


Variable Partitions and Fragmentation

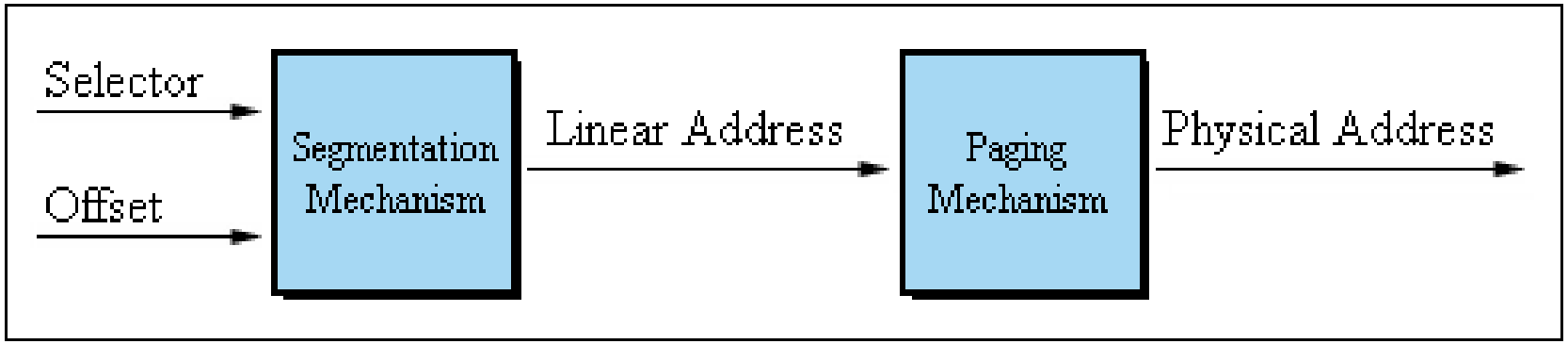
Memory wasted by External Fragmentation

Compaction

- Processes must be suspended during compaction
- Need be done only when fragmentation gets very bad



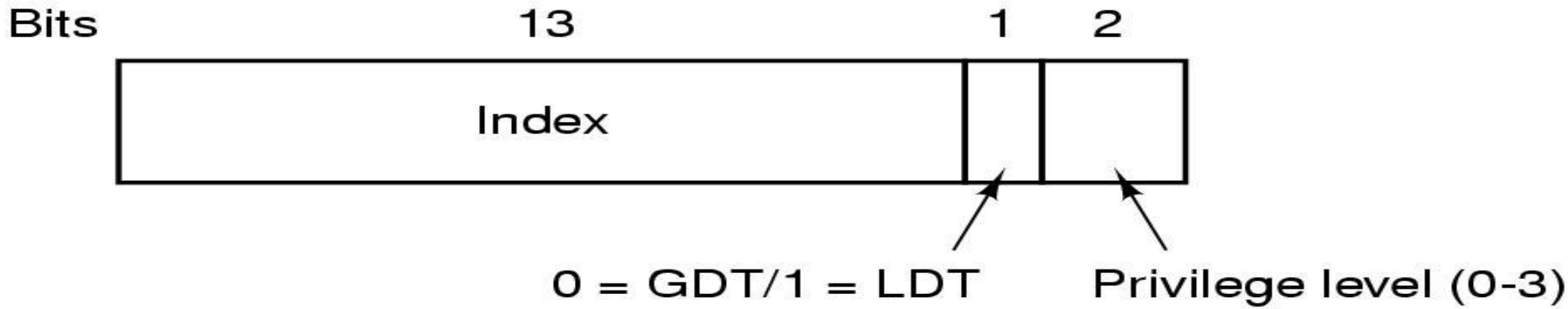
Virtual Memory
Segmentation with Paging: Pentium (1)



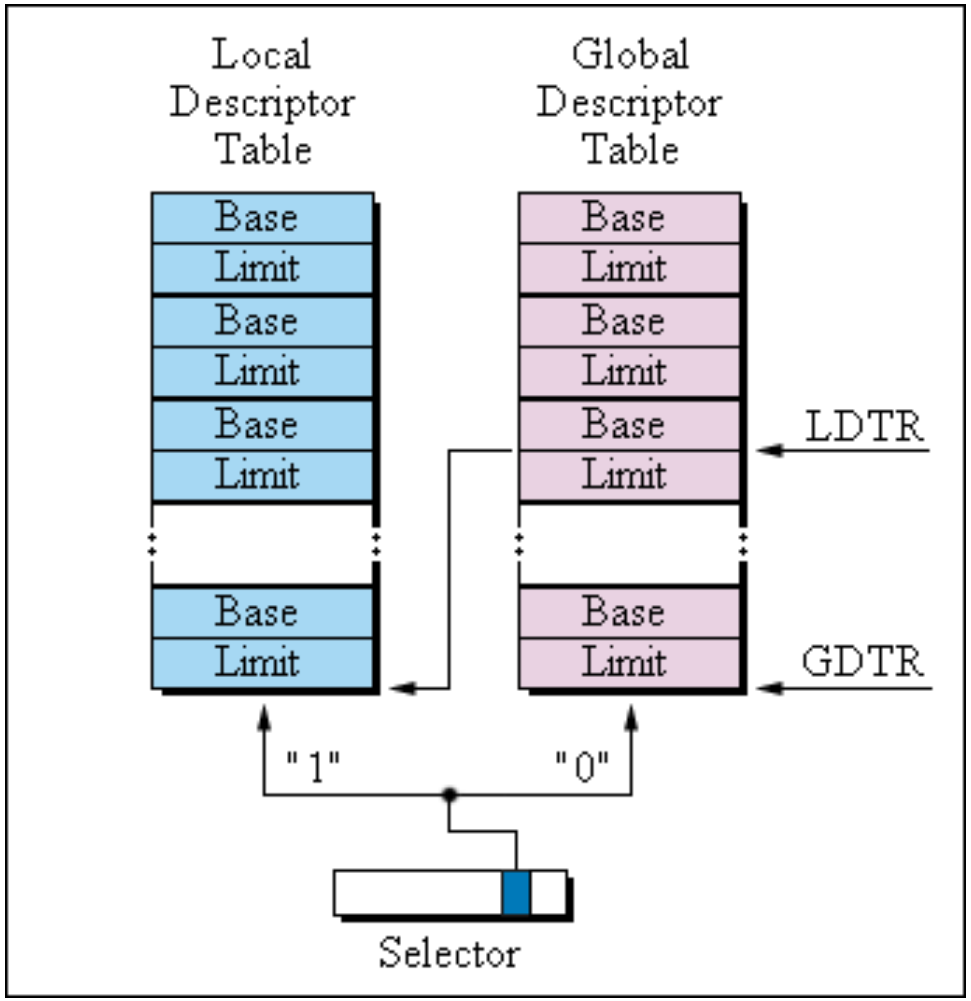
Virtual Memory
Segmentation with Paging: Pentium (2)

A Pentium selector

GDT (Global Descriptor Table), LDT (Local Descriptor Table)

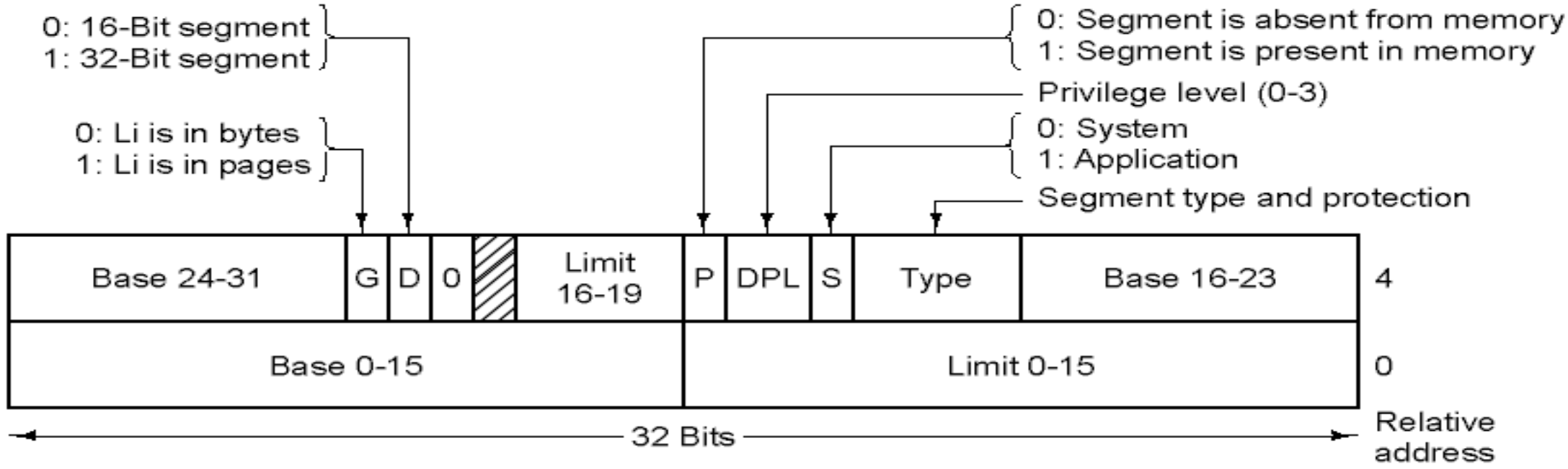


Virtual Memory
Segmentation with Paging: Pentium (3)

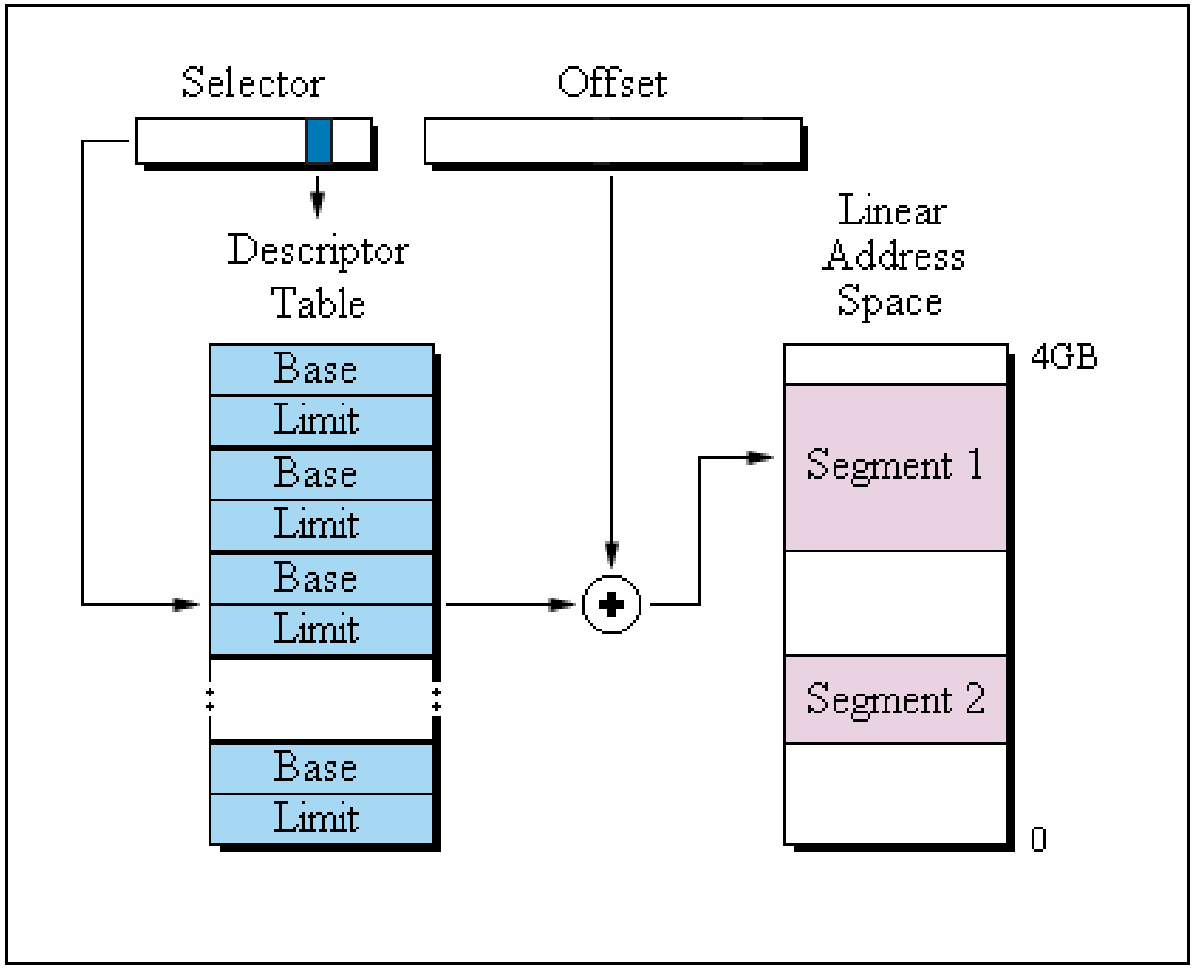


Virtual Memory
Segmentation with Paging: Pentium (4)

- Pentium code segment descriptor
- Data segments differ slightly

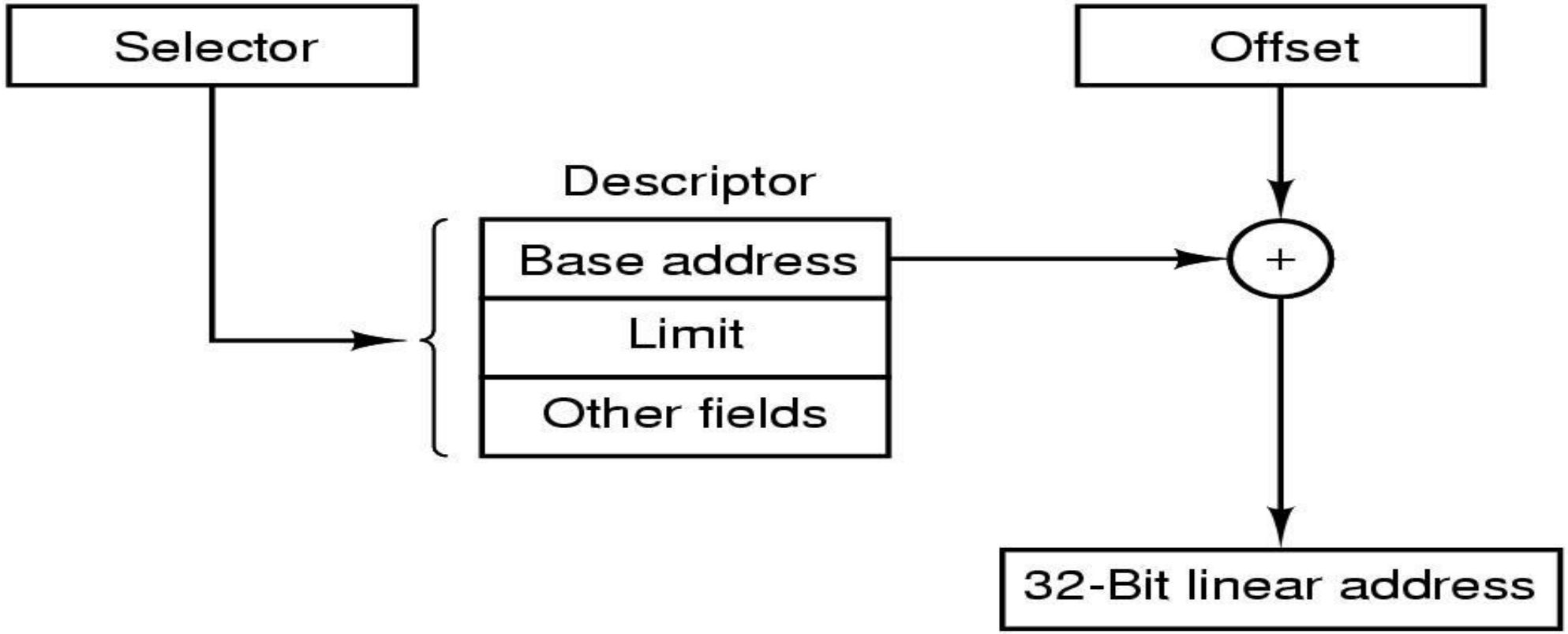


Virtual Memory
Segmentation with Paging: Pentium (5)



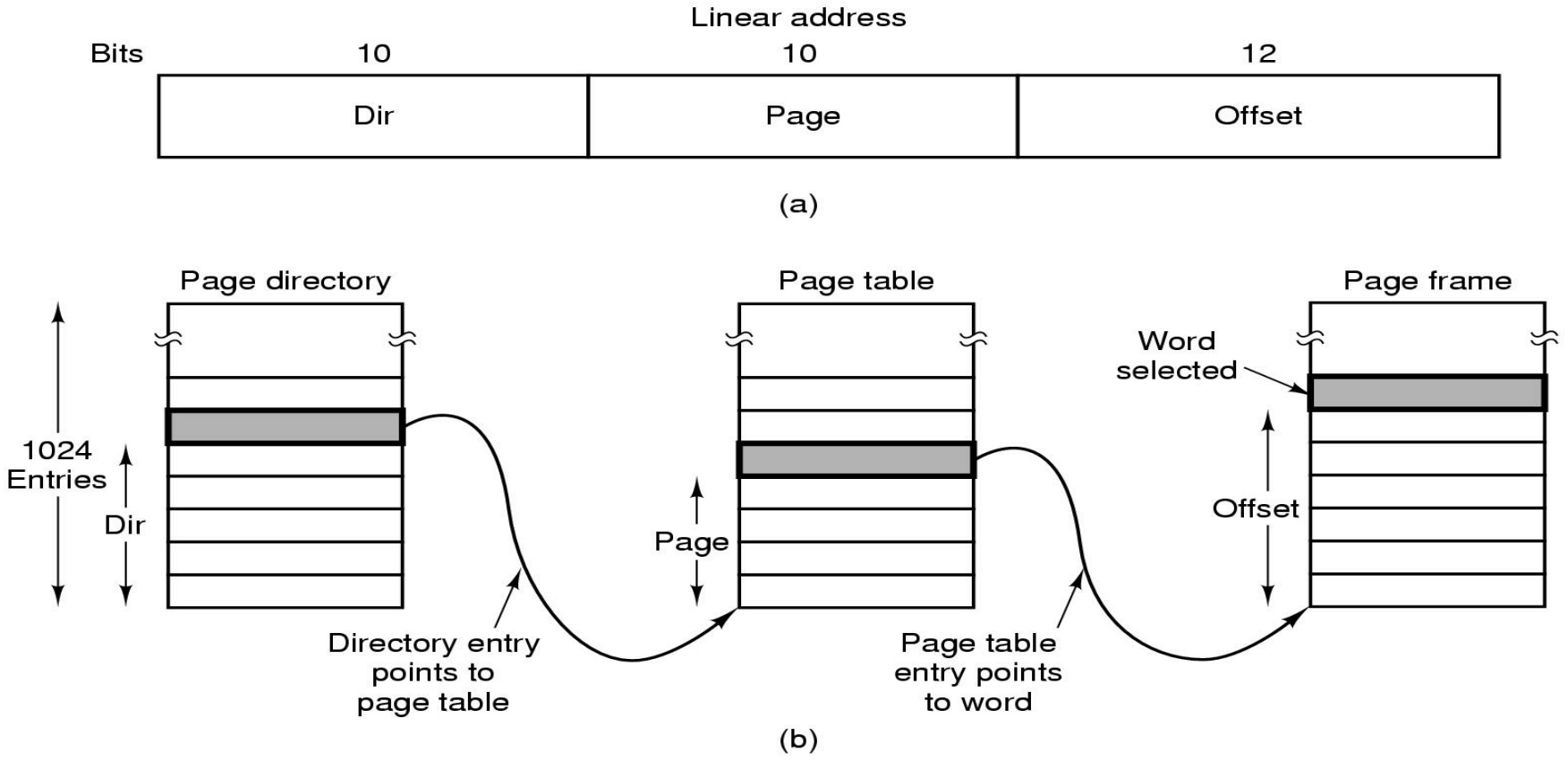
Virtual Memory
Segmentation with Paging: Pentium (6)

Conversion of a (selector, offset) pair to a linear address

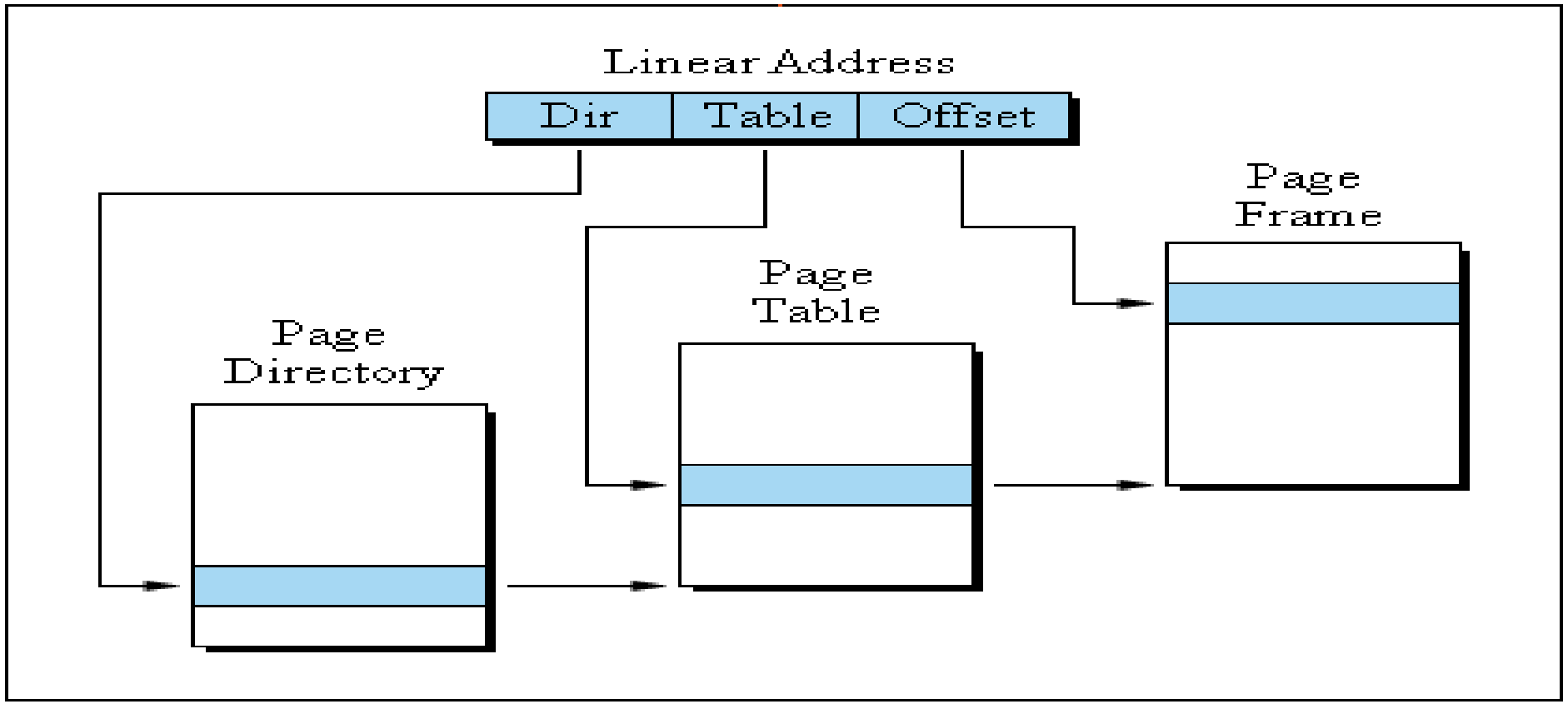


Virtual Memory
Segmentation with Paging: Pentium (7)

Mapping of a linear address onto a physical address

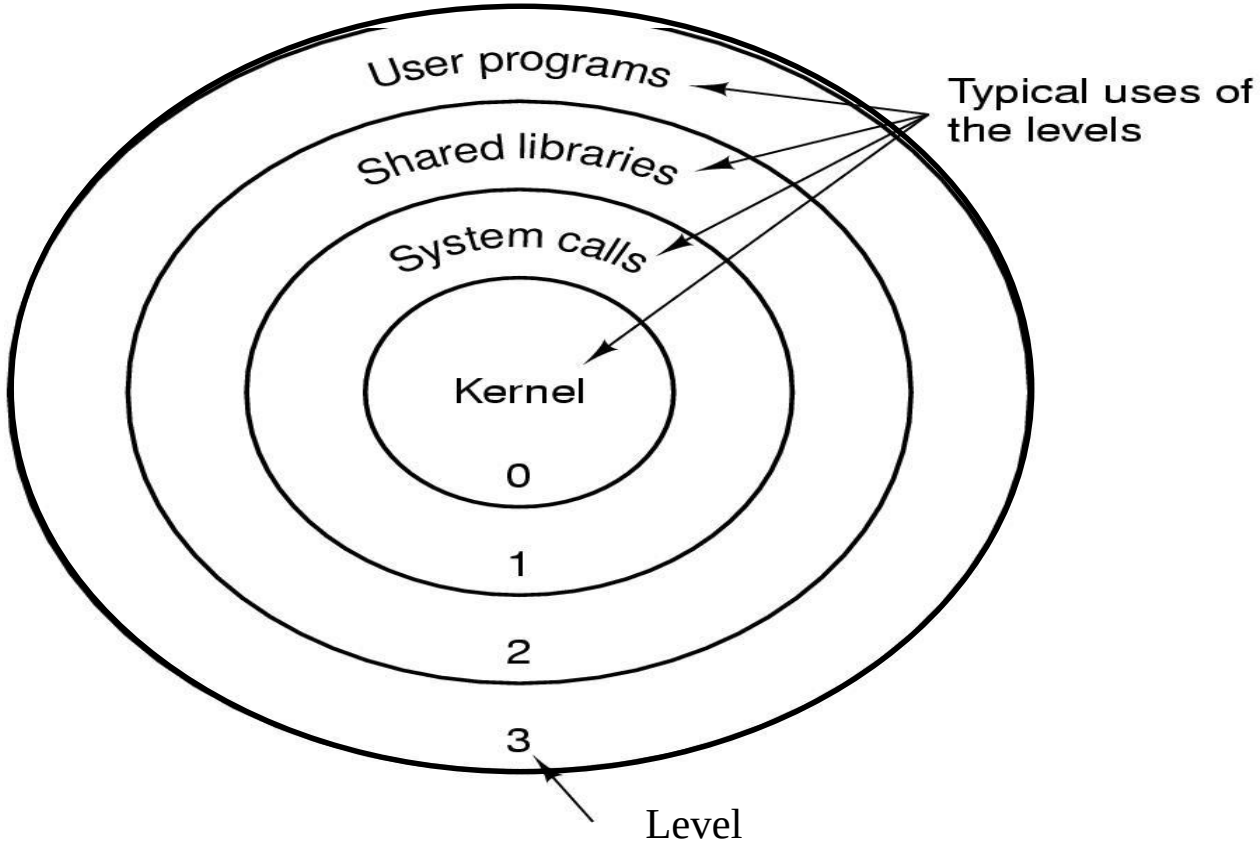


Virtual Memory
Segmentation with Paging: Pentium (8)



Virtual Memory
Segmentation with Paging: Pentium (9)

Protection on the Pentium



SUMMARY

- Basic memory management
- Swapping
- Virtual memory
- Design issues for paging systems
- Implementation issues
- Segmentation