OPERATING SYSTEMS

Subject In-charge

Nidhi Gaur

Assistant Professor

email: nidhigaur@sfit.ac.in

Module 4: Memory Management

Memory Management

- Background and Requirements
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Segmentation

Operating Systems Early Memory Management Schemes

1 Single-User Contiguous

CHARACTERISTICS:

1. Not efficient use of memory
2. Poor performance (SLOW)
3. Single job or program reserves the ENTIRE and CONTIGUOUS memory space

Introduction

- Multiprogramming concept gave rise to memory management.
- Memory as a resource need to be partitioned and allocated to ready processes, such that both processor and memory can be utilized efficiently.
- Memory is partitioned into two parts:

1. One for OS
2. Other for user area

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- Cache sits between

main memory and CPU registers

Memory Management

- Subdividing memory to accommodate multiple processes
- Memory needs to be allocated efficiently to pack as many processes into memory as possible
- handles or manages primary memory and moves processes back and forth between main memory and disk during execution.
- keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.

Memory Management

Two methods for memory allocation:

- Static allocation: compile time
- Dynamic allocation: execution time

Memory Management

Two methods for memory allocation:

- Static allocation: compile time

Two instances:

1. Location of the process is known at compile time. Compiler generates absolute code. If location need to be changed code must be recompiled.
2. Location of the process in memory is not known at compile time, compiler generates relocatable code, the address that is relative to some known point.

In both cases size should be known before start of execution.

Memory Management

Two methods for memory allocation:

- Dynamic allocation: execution time

Memory allocation deferred till the process starts executing.

Logical vs. Physical Address Space

- Dynamic memory allocation method is used to accommodate multi programming concept.
- Two types of addresses are generated:
  - **Logical address** – generated by the CPU; also referred to as **virtual address. As place of allocation of process not known at compile time.**
  - **Physical address** – address seen by the memory unit
  - The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

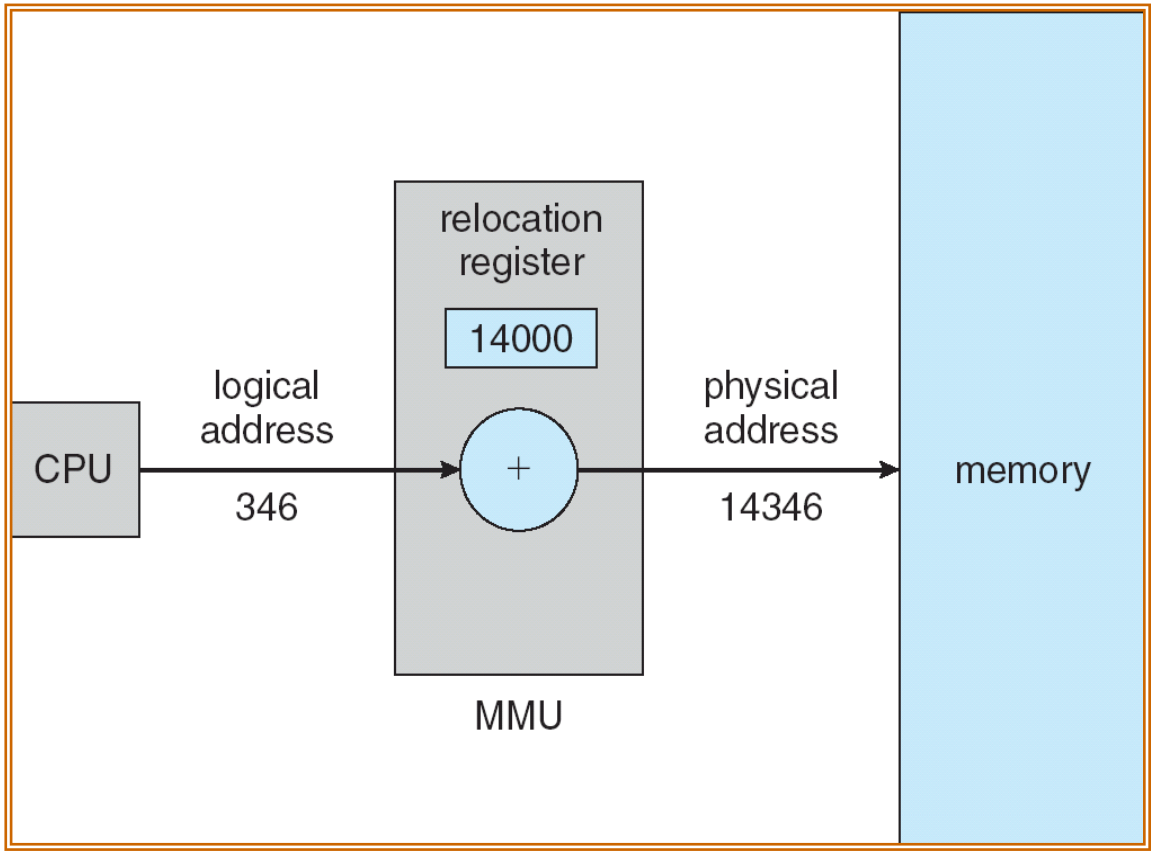
Memory-Management Unit (MMU)

- The runtime mapping from logical to physical address is done by the memory management unit (MMU) which is a hardware device
- 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

Memory Management Requirements

- Relocation
  - Programmer does not know where the program will be placed in memory when it is executed
  - While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
  - Memory references must be translated in the code to actual physical memory address

Dynamic relocation using a relocation register



Dynamic relocation using a relocation register

Example: A process is to be swapped in at the location 20100 in memory. If logical addresses generated by the process are 200, 440 and 550. What are corresponding physical addresses?

Ans:

Relocation register will be loaded with address 20100

So adding relocation register to logical addresses, the corresponding physical addresses are:

20100+200 = 20300

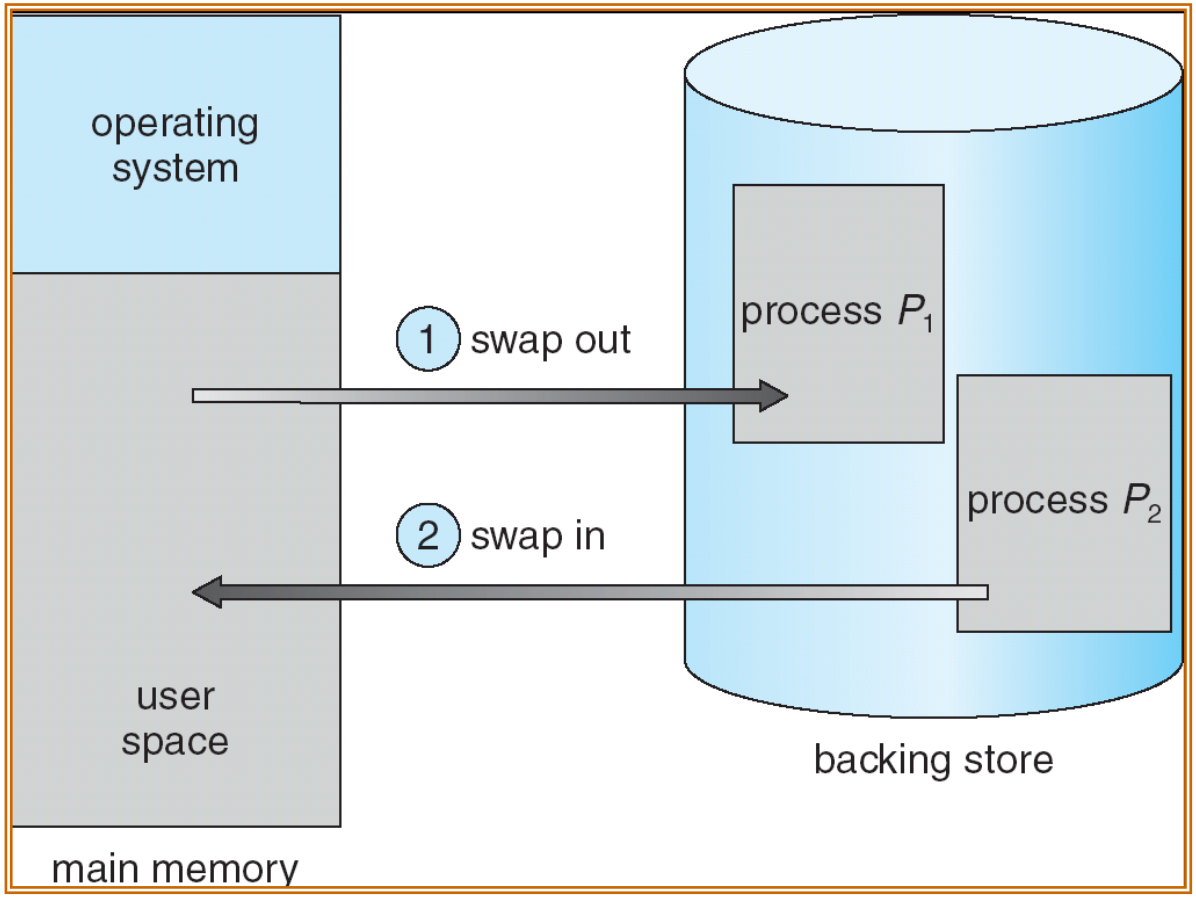
20100+440 = 20540

20100+ 550 = 20650

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users;
- Swap out, swap in – action of taking out process from memory is swap out and action of bringing back swapped out process is called swap in.
- Swap space: a separate space in hard disk is reserved for swapped out processes.
- Swap time : time taken to access the hard disk

Schematic View of Swapping



Swapping

Some of the processes that can be swapped out are:

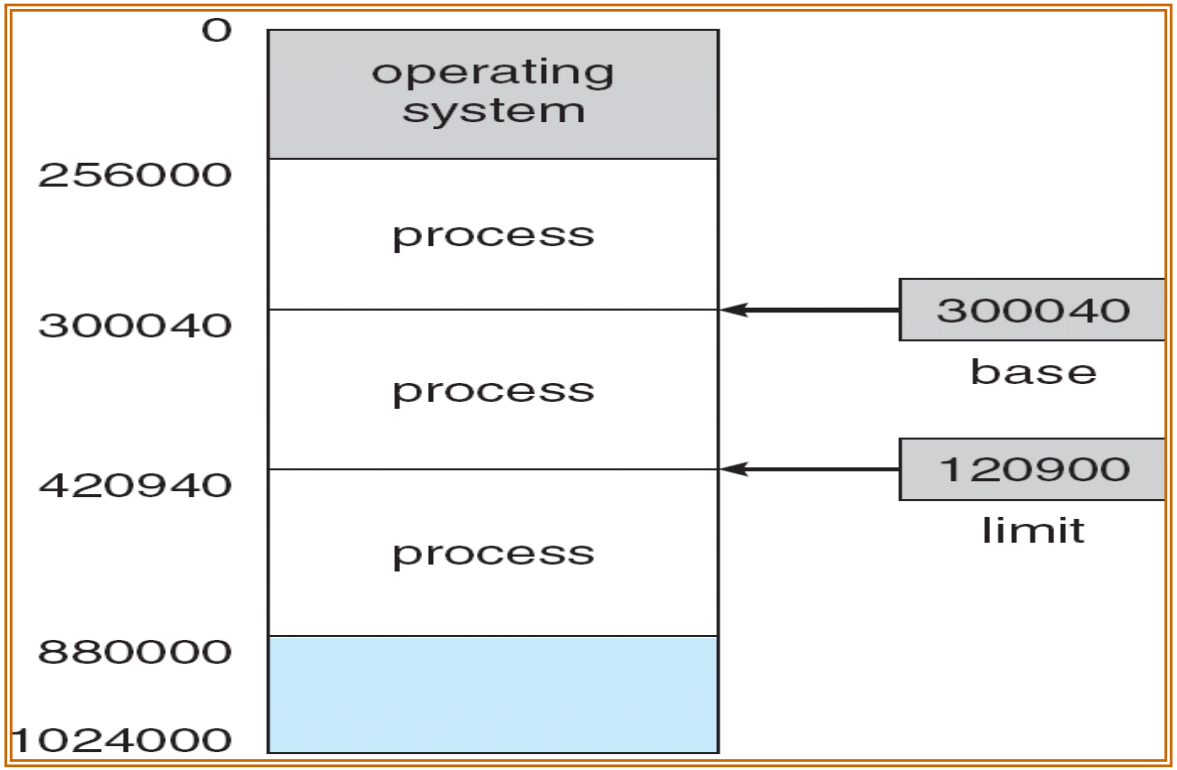
- In RR scheduling, if time quantum expires and the process has not finished execution, it can be swapped out.
- In priority scheduling, if a higher priority process wishes to execute, a lower priority process in memory will be swapped out.
- The blocked processes waiting for I/O can be swapped out.

Contiguous Allocation

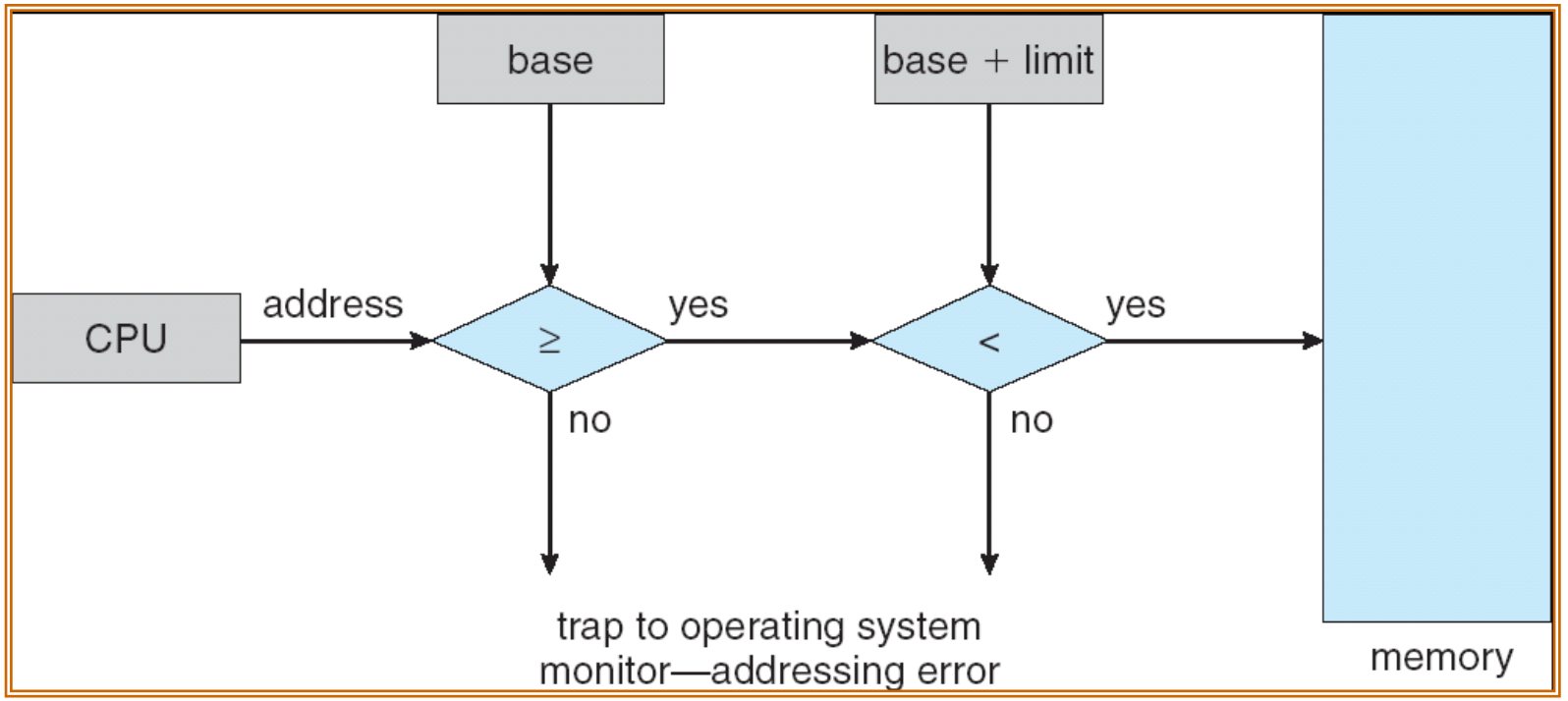
- Main memory usually divided into two partitions:
  - Resident operating system
  - User processes
  - Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register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

Base and Limit Registers

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

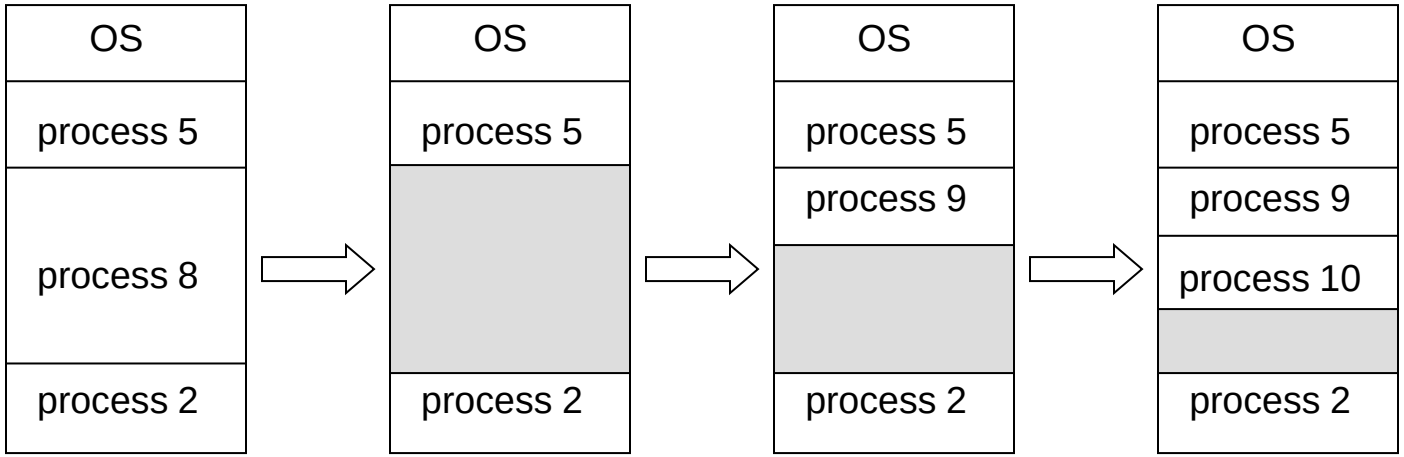


HW address protection with base and limit registers



Contiguous 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)



Contiguous allocation

- PDT(Partition Description Table)

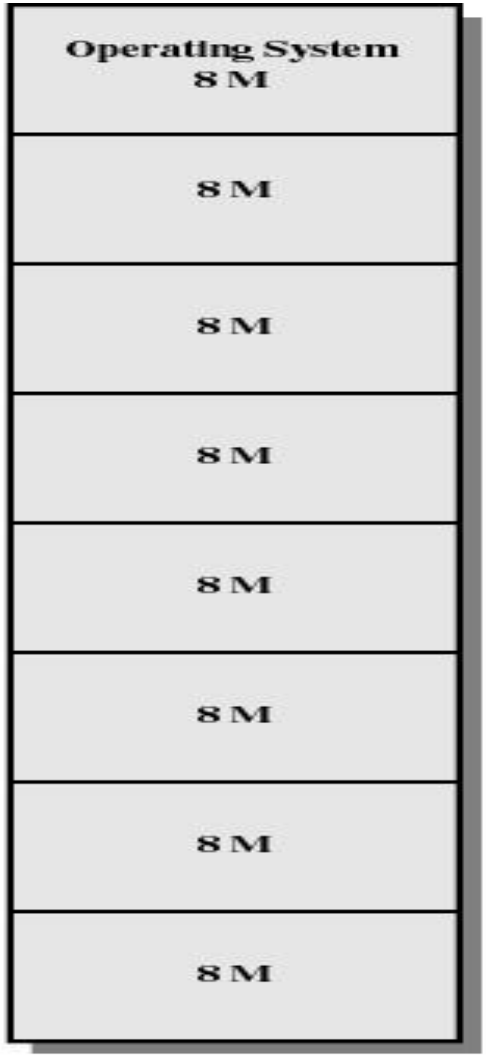
Partition id	Starting address	Size	Allocation status

Types of Partitioning

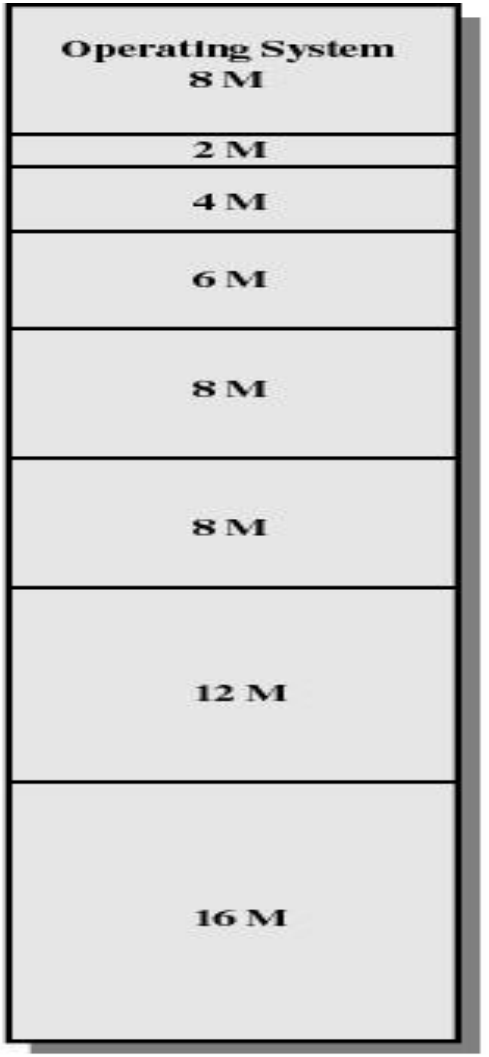
- Fixed Partitioning: at time of system generation; can't be changed.
- Dynamic Partitioning: at run time

Fixed partitioning

- Equal-size partitions
  - As all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - can assign each process to the smallest partition within which it will fit
  - queue for each partition
  - processes are assigned in such a way as to minimize wasted memory within a partition

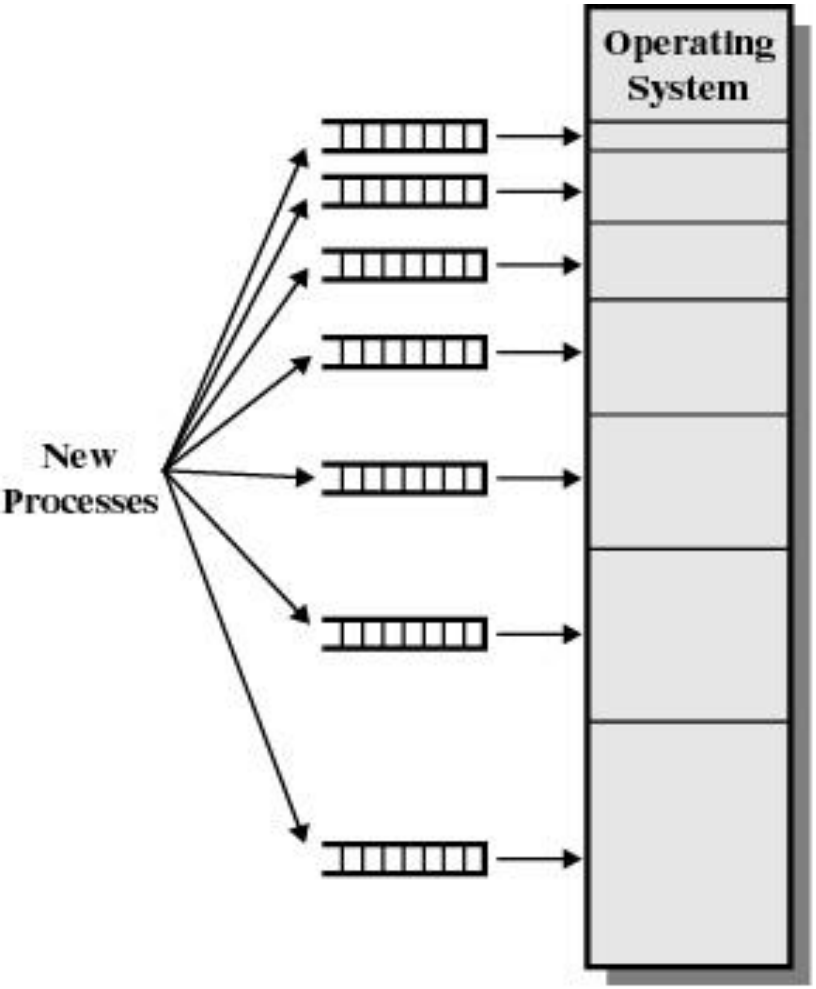


(a) Equal-size partitions

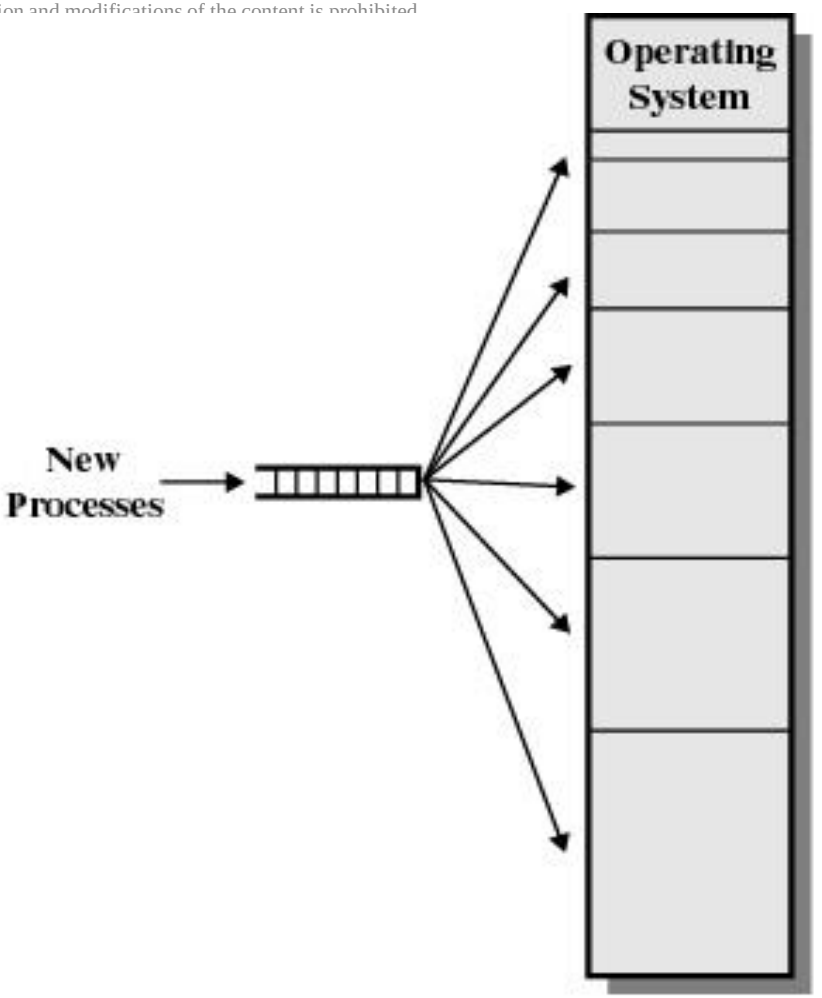


(b) Unequal-size partitions

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory



(a) One process queue per partition



(b) Single process queue

Figure 7.3 Memory Assignment for Fixed Partitioning

Fixed partitioning

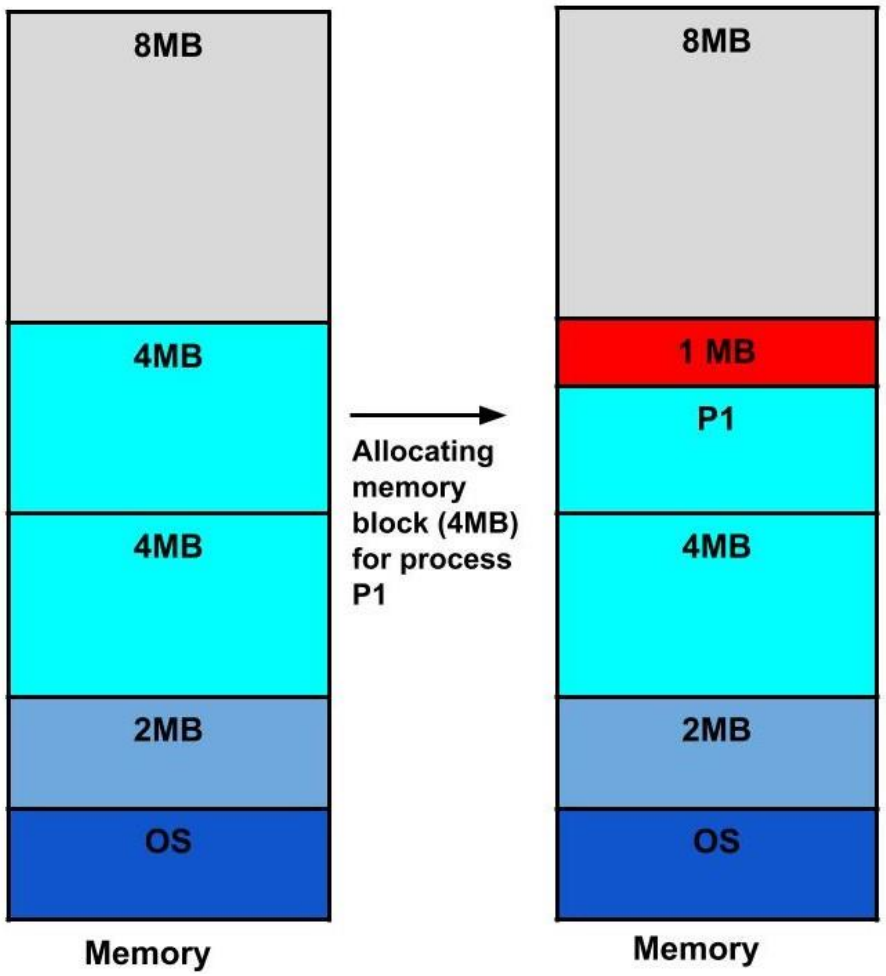
- In **contiguous memory allocation** whenever the processes comes into RAM, space is allocated to them.
- These spaces in RAM are divided either on the basis of **fixed partitioning**(the size of partitions are fixed before the process gets loaded into RAM) or **dynamic partitioning** (the size of the partition is decided at the run time according to the size of the process).

Fixed Partitioning

- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- It happens after sometime that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused.
- This problem is known as Fragmentation.
- When memory block assigned to process is bigger, some portion of memory is left unused, as it cannot be used by another process. This is called internal fragmentation
- Two types: INTERNAL and EXTERNAL fragmentation.

Fixed partitioning

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



Fixed partitioning

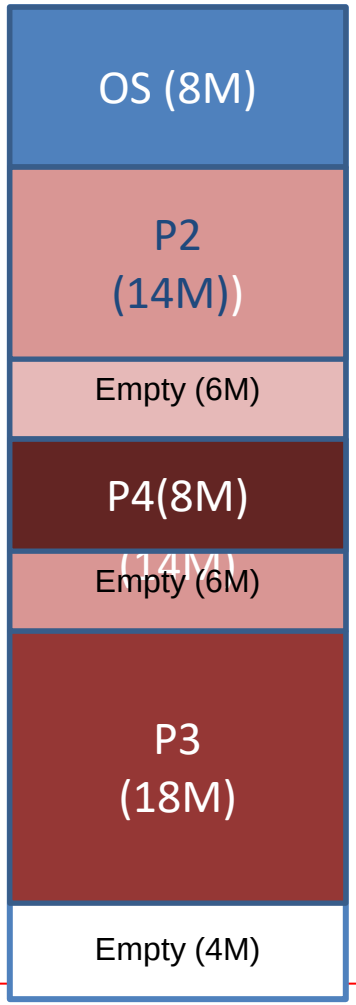
Example: Three processes P1, P2, P3 of size 21900, 21950 and 21990 bytes need space in memory. If equal sized partitions of 22000 bytes are allocated to P1 , P2 and P3, will there be any fragmentation in this allocation?

Ans: YES, 100, 50 and 10 bytes respectively

Dynamic Partitioning

- 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
- Must use compaction to shift processes so they are contiguous and all free memory is in one block

Dynamic Partitioning Example



- *External Fragmentation*
- Memory external to all processes is fragmented
- Can resolve using *compaction*
  - OS moves processes so that they are contiguous
  - Time consuming and wastes CPU time

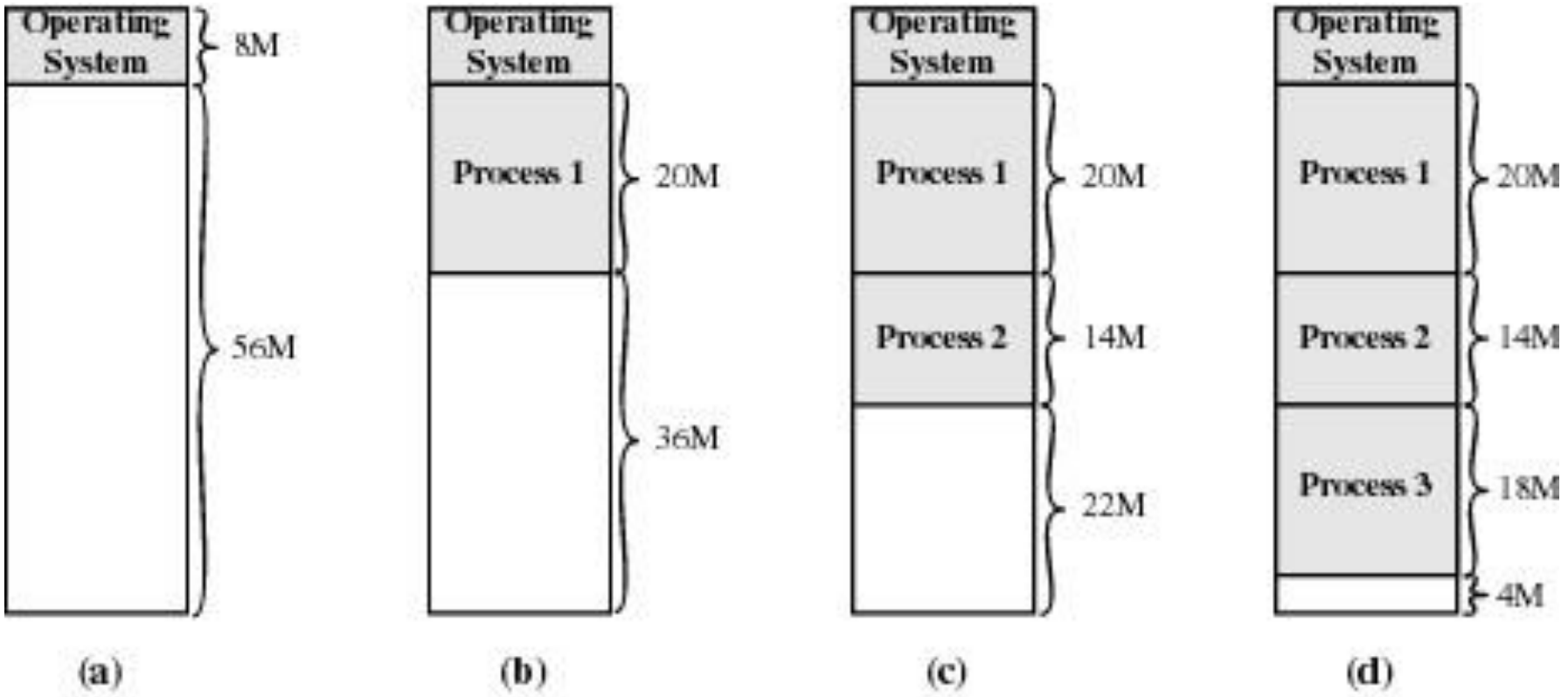


Figure 7.4 The Effect of Dynamic Partitioning

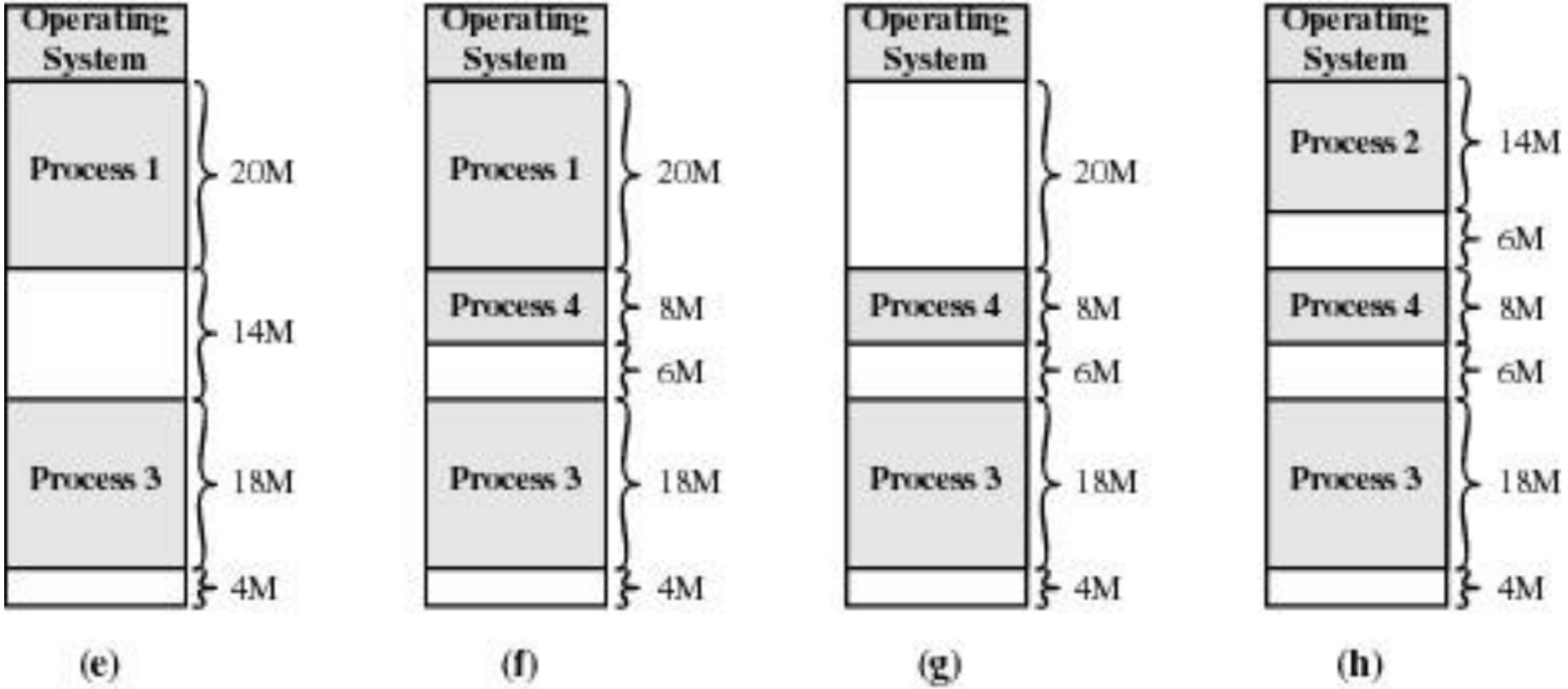
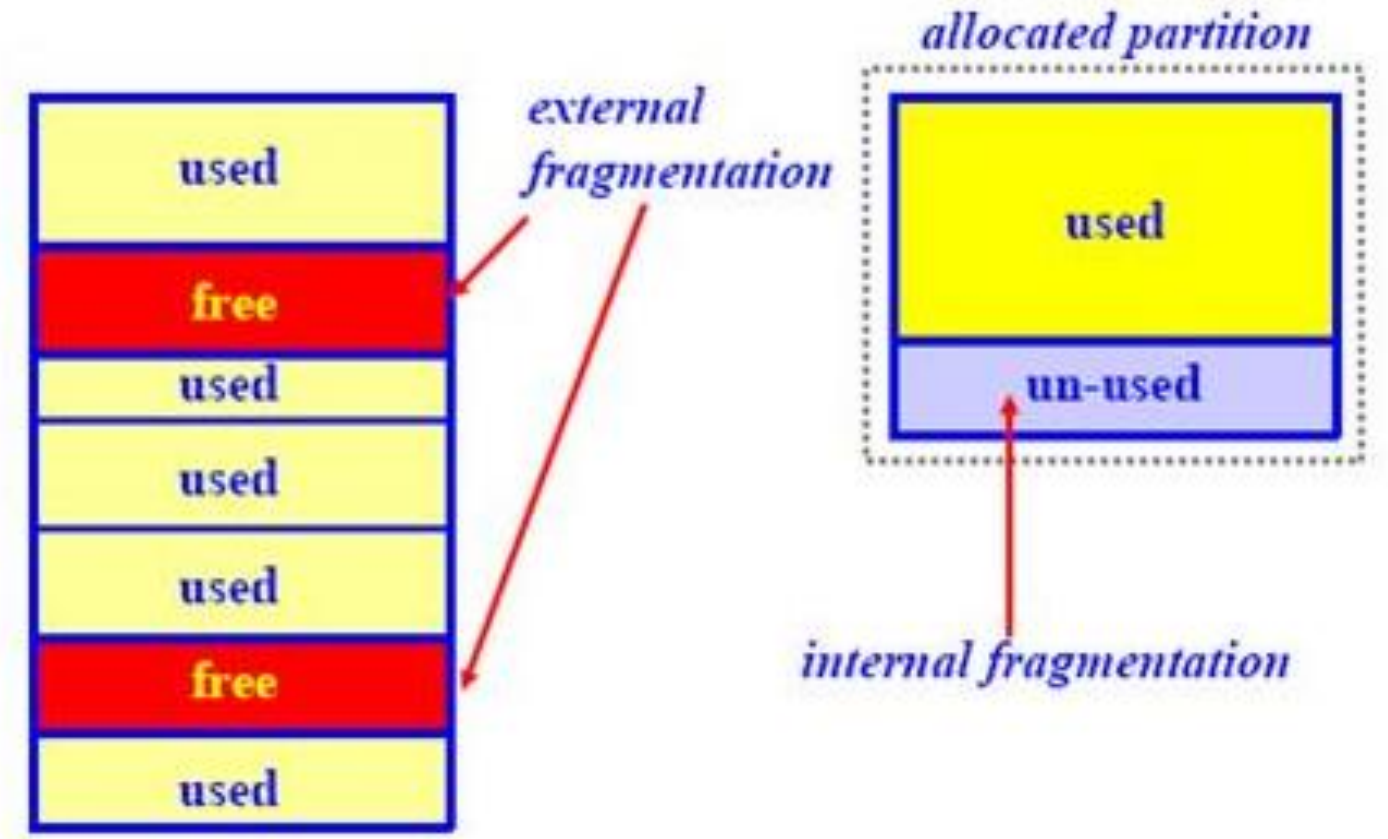


Figure 7.4 The Effect of Dynamic Partitioning

Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block

Fragmentation



Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process.
- Memory allocation techniques are algorithms to satisfy memory needs of a process.
- Also known as partition selection algorithms.

Dynamic Partitioning Placement Algorithm

- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Since smallest block is found for process, the smallest amount of fragmentation is left, memory compaction must be done more often

Dynamic Partitioning Placement Algorithm

- First-fit algorithm
  - Fastest
  - May have many processes loaded in the front end of memory that must be searched over when trying to find a free block

Dynamic Partitioning Placement Algorithm

- Next-fit
  - Allocates a block of memory after the last allocation.
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

Dynamic Partitioning Placement Algorithm

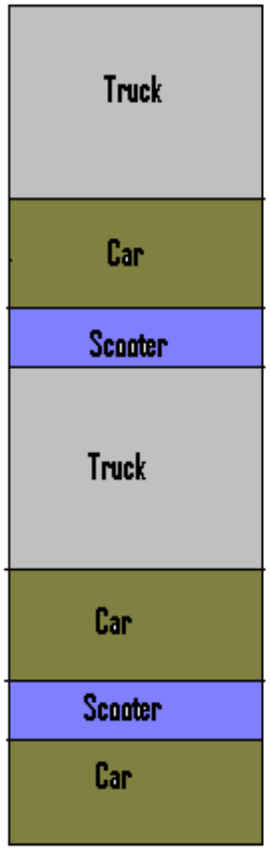
- Worst-fit
  - Allocates the largest block of memory
  - The largest block of memory leaves largest fragment after allocation

Memory Allocation Policies

Example: *Parking Space Management*

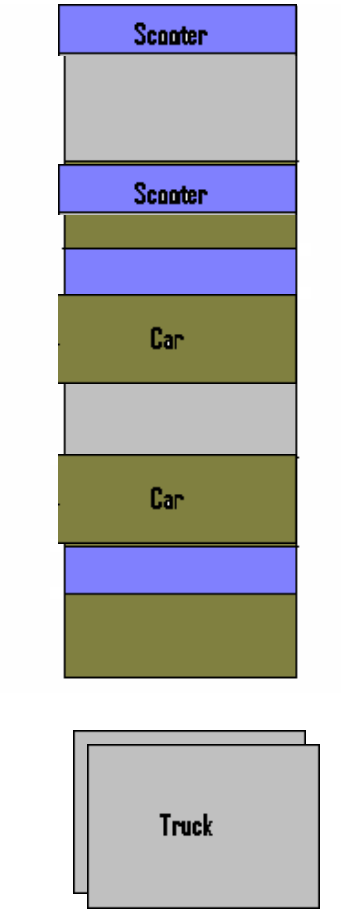
- A scooter, car and a truck are looking out for space for parking. They arrive in the order mentioned above. The parking spaces are available for each one of them as per their size. Truck parking space can accommodate , a car and a scooter or even two scooters. Similarly, In a car parking space two scooters can be parked.

Memory Allocation Policies

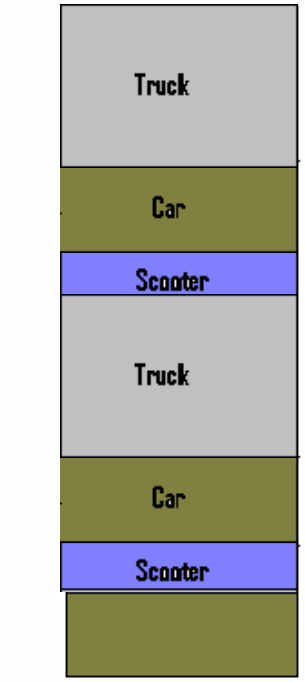


- Alongside is shown the partition in the parking area for Truck, Car and Scooter.
- Now when a scooter, car and truck come in order, parking space is allocated according to algorithm policies

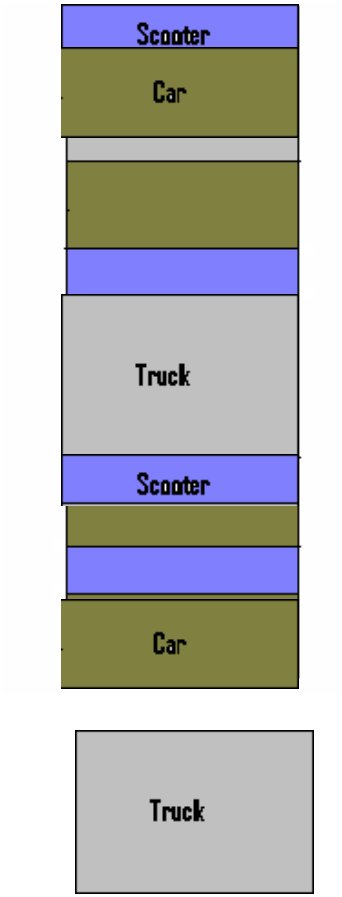
Worst Fit



Best Fit



Next Fit



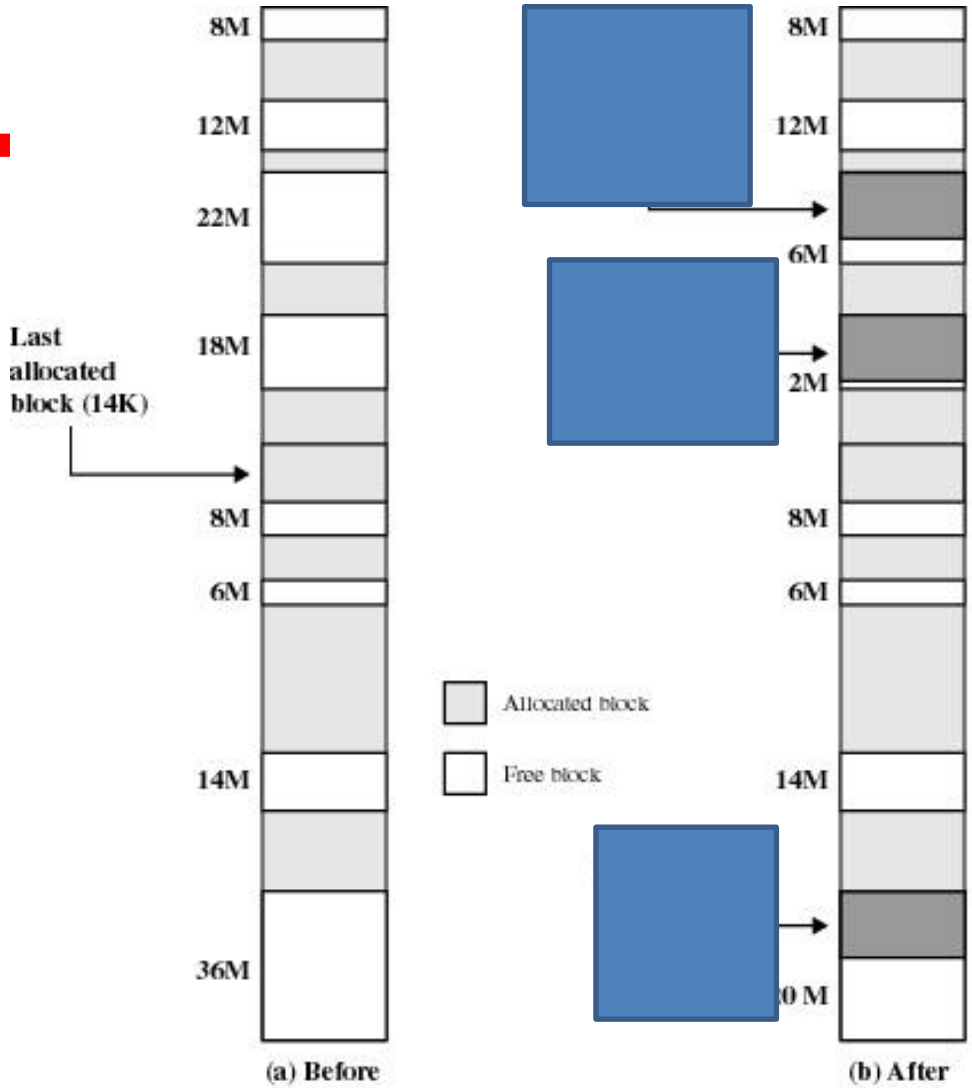


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

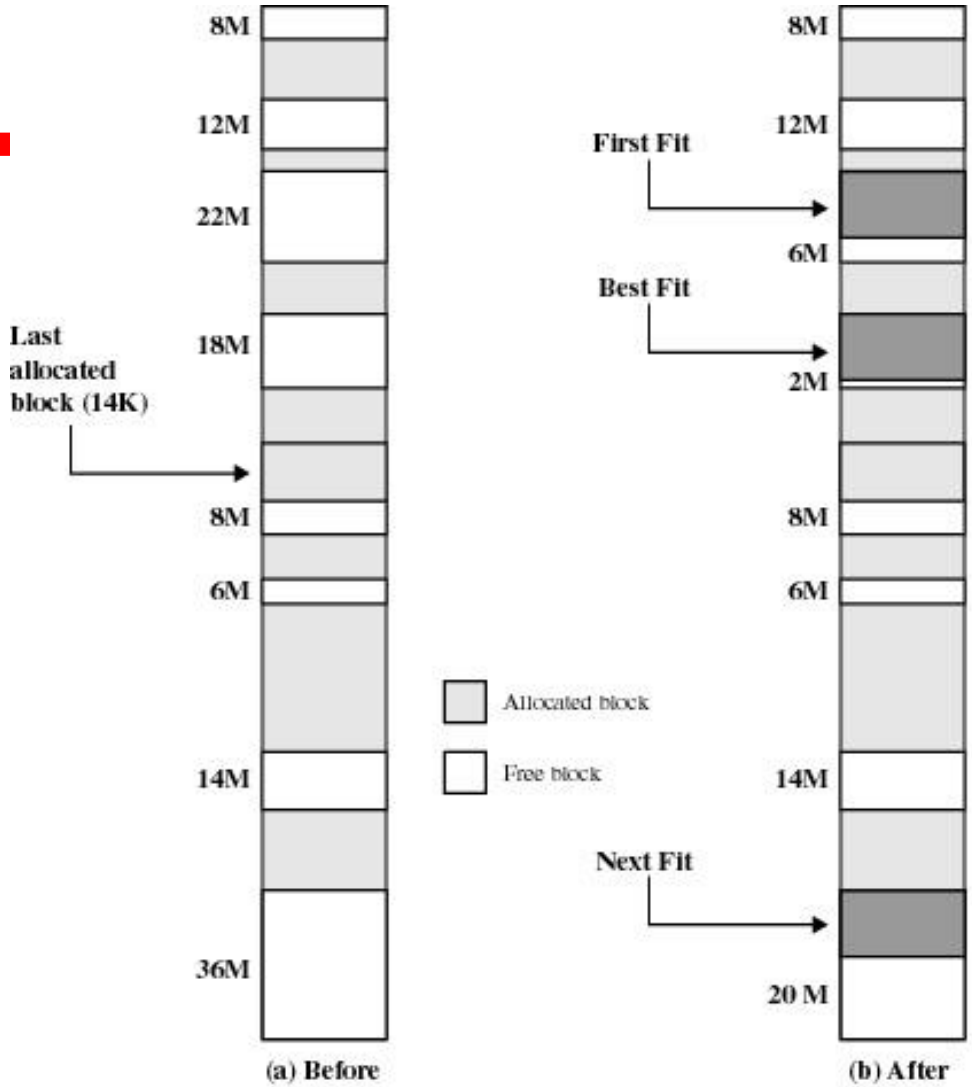


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Storage-Allocation Problem

To summarize:-

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

Dynamic Partitioning Placement Algorithm

Example:

- Allocate memory for request of 4k and 10k(in this order).
- Compare memory allocation using first, best and worst fit.

OS	FREE 10K	25K	FREE 5K	20K	FREE 15K	15K	22K FREE

Buddy System

- Entire space available is treated as a single block of 2^U
- If a request of size s such that $2^{U-1} < s <= 2^U$, entire block is allocated
  - Otherwise block is split into two equal buddies
  - Process continues until smallest block greater than or equal to s is generated

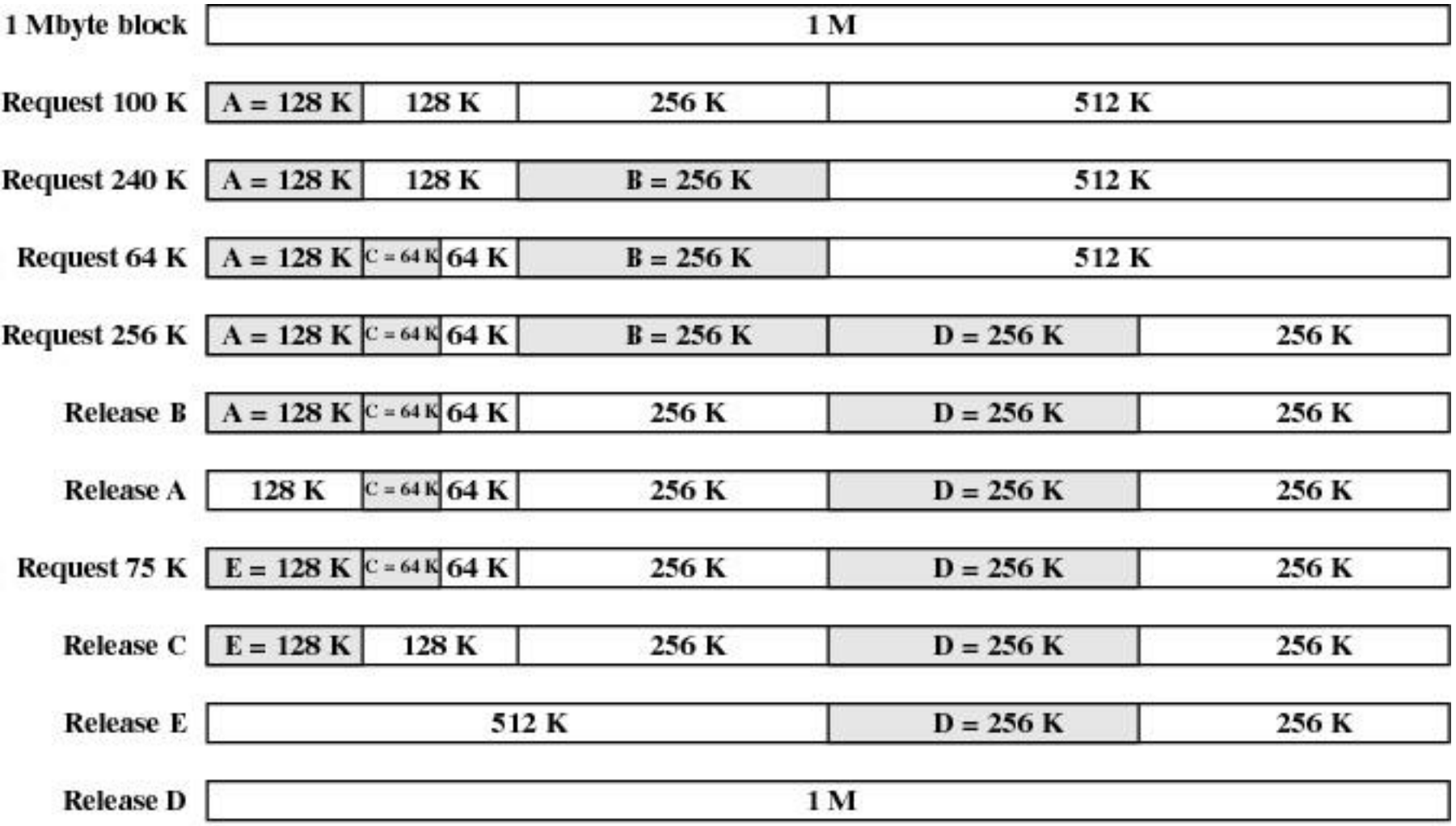


Figure 7.6 Example of Buddy System

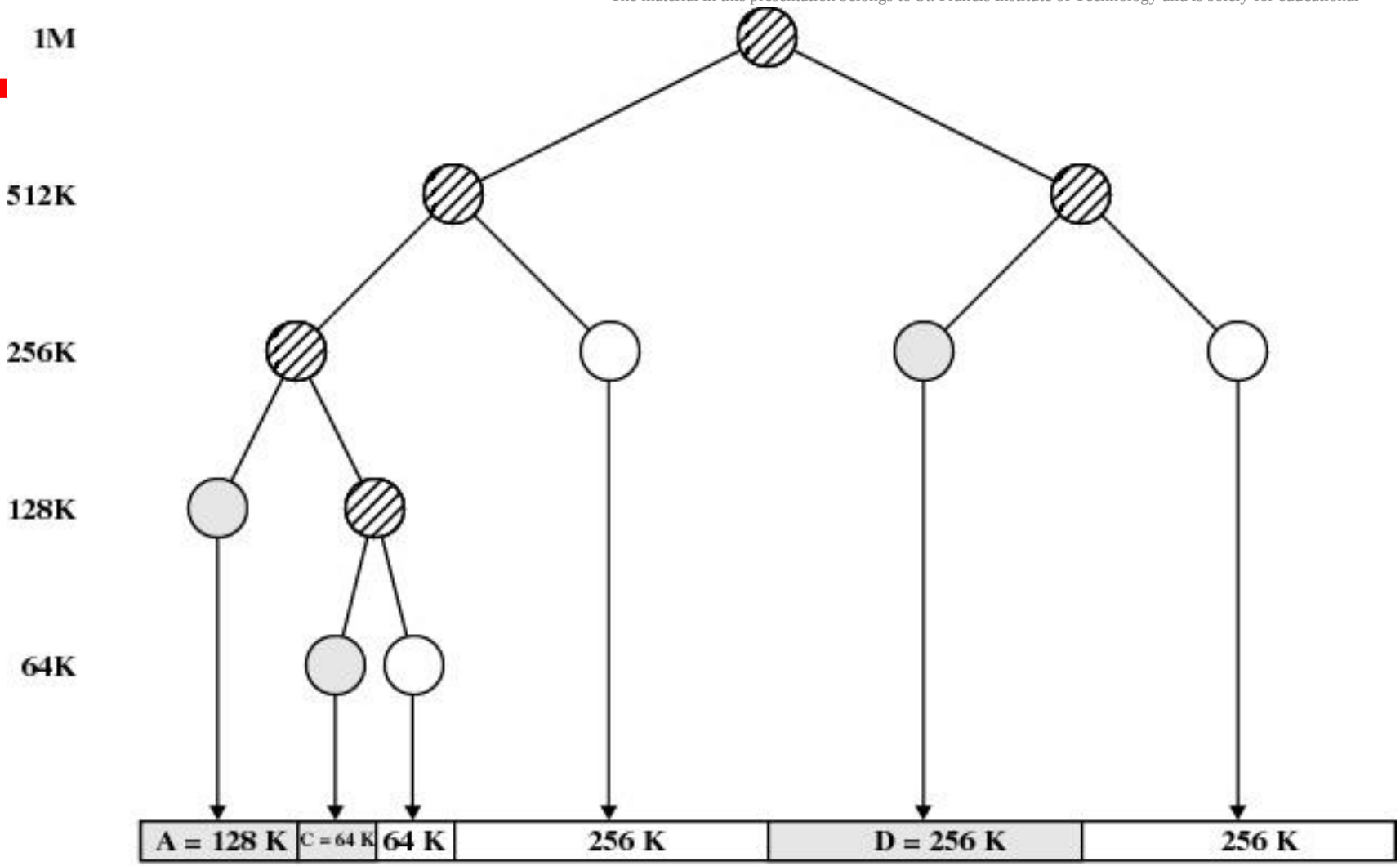


Figure 7.7 Tree Representation of Buddy System

Non contiguous memory allocation

- Contiguous memory allocation suffers from some drawbacks.
- Internal and external fragmentation
- It wastes lot of memory space
- These drawbacks lead us to non contiguous method.
- Two types are:

Fixed partitioning: Paging

Variable partitioning: Segmentation

Paging

- Paging is a logical concept that divides the logical address space of a process into fixed size pages and is implemented in physical memory through frames.
- Memory is divided into equal size partitions
- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called pages

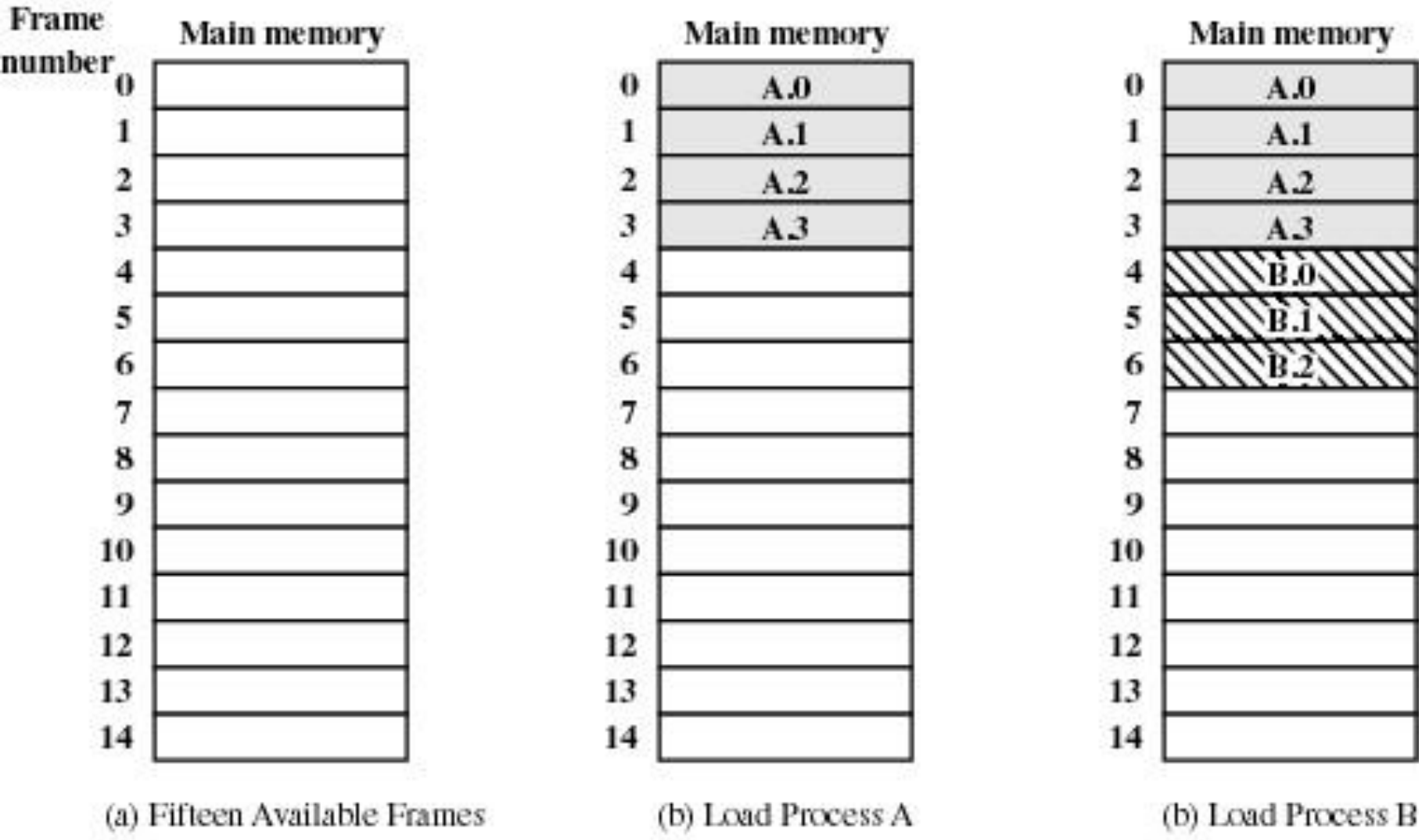


Figure 7.9 Assignment of Process Pages to Free Frames

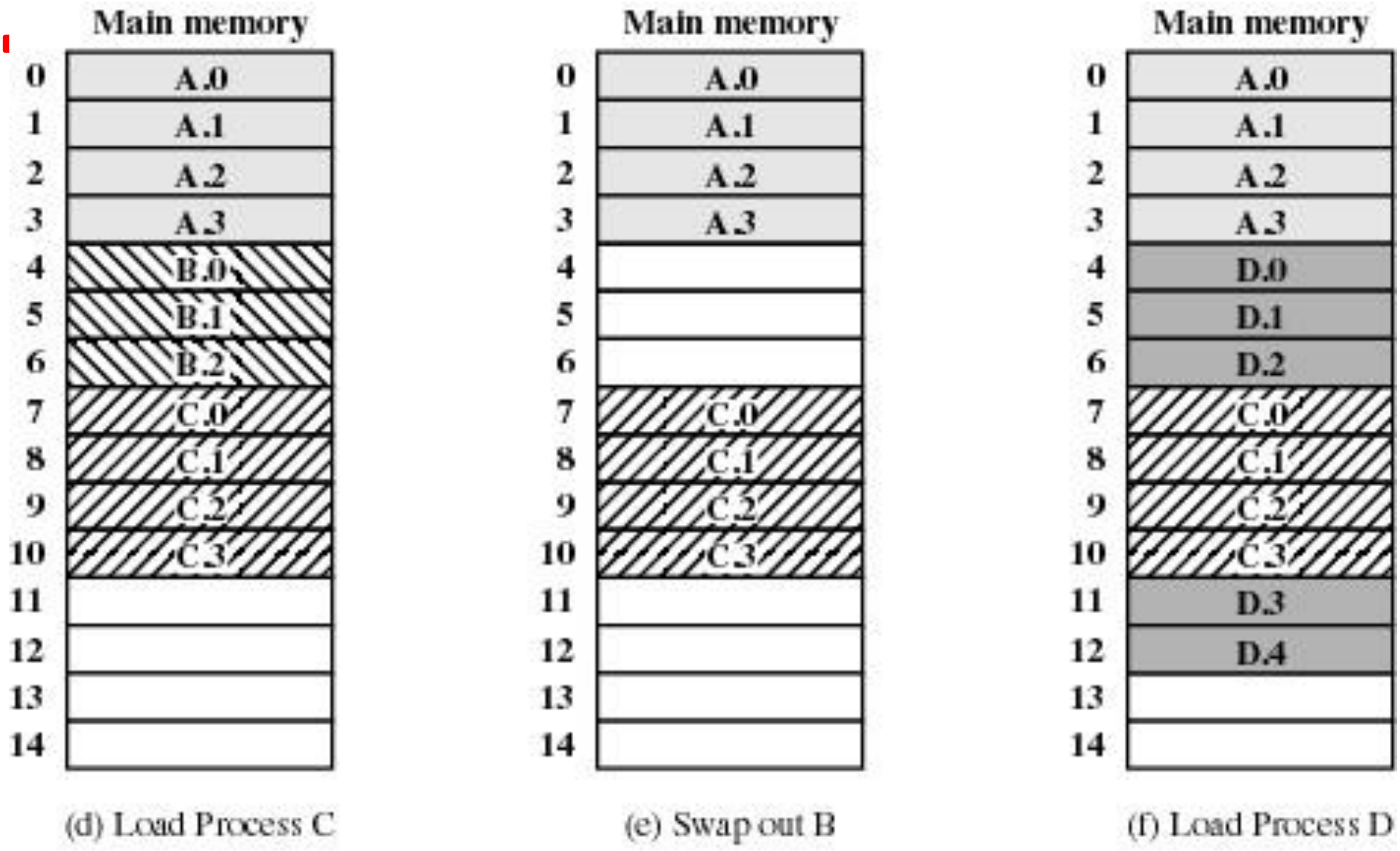


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

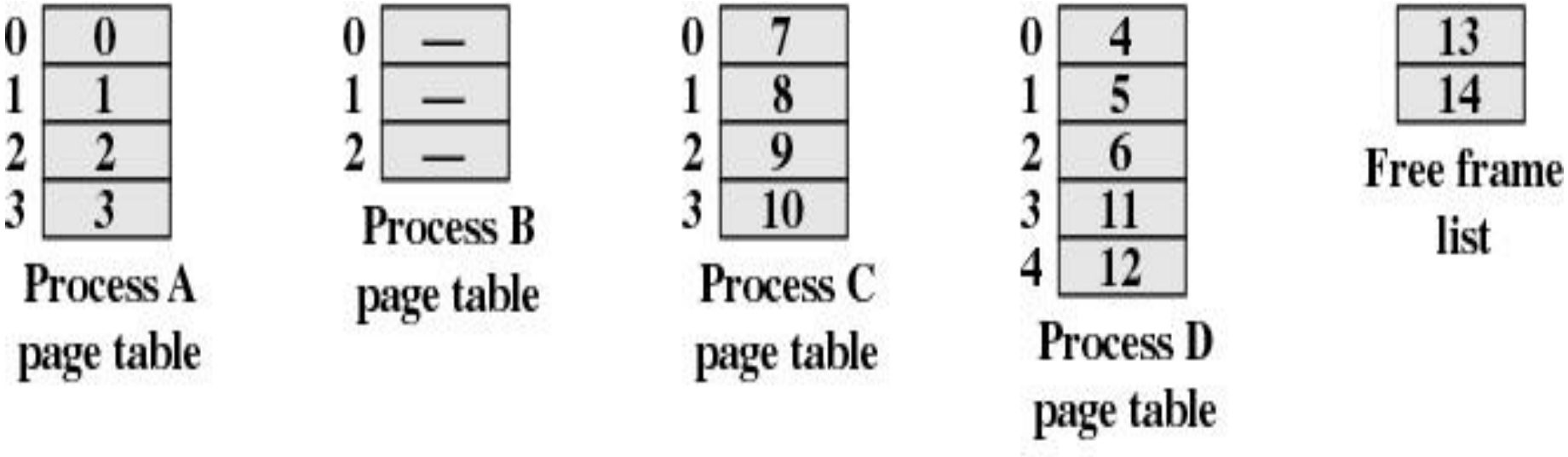


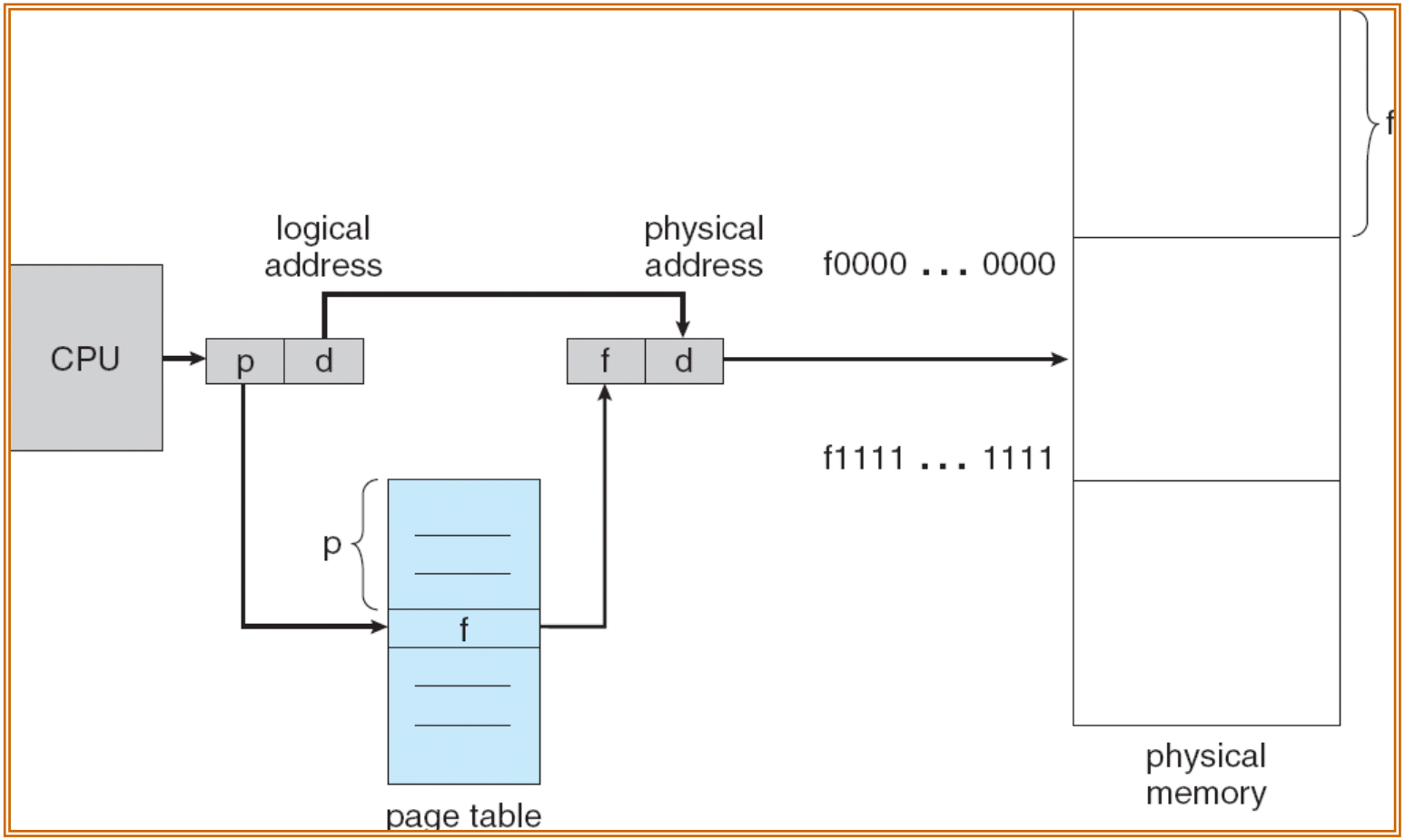
Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Address Translation Scheme

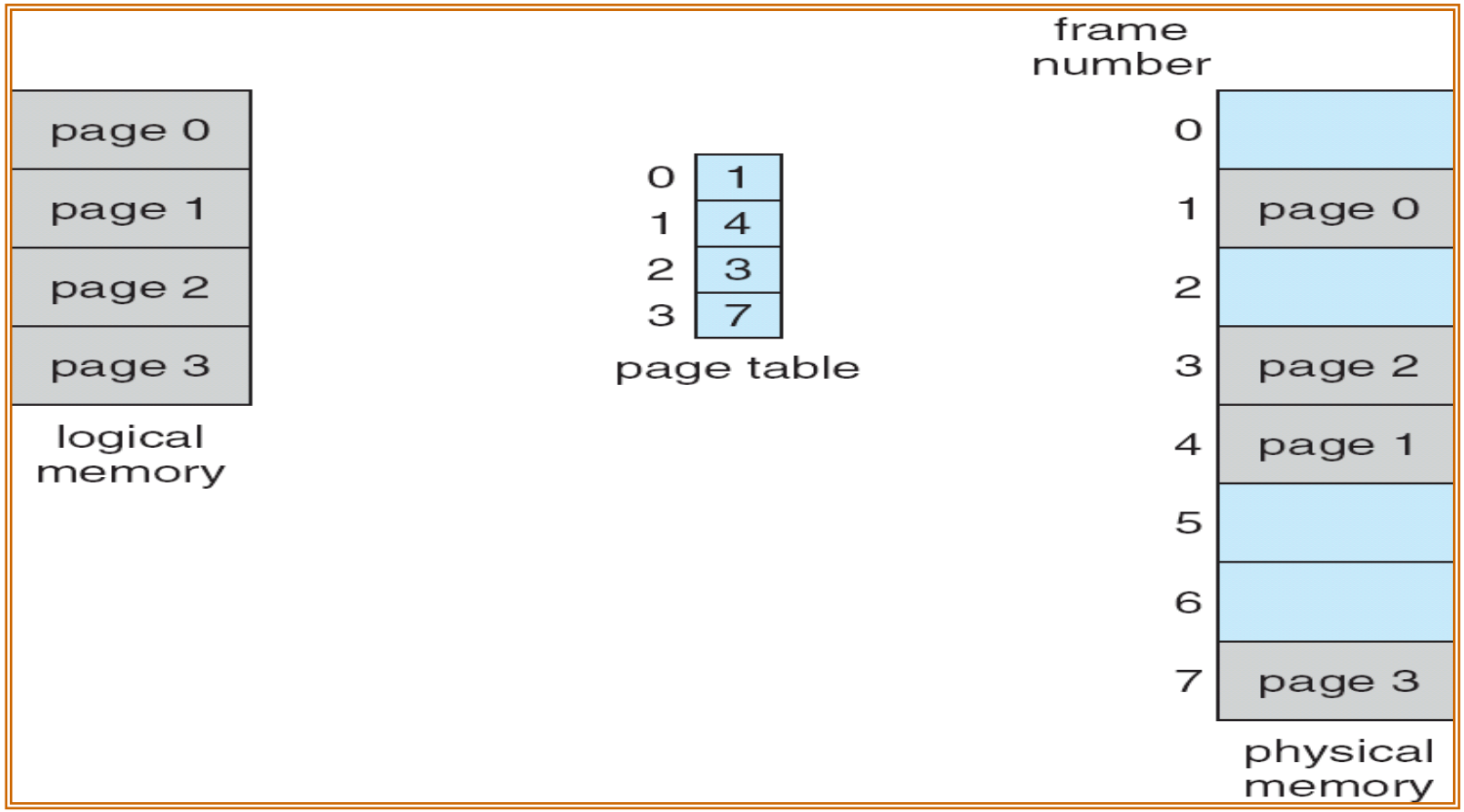
- Address generated by CPU is divided into:
  Page number (*p*) – used as an index into a *page table* which contains base address of each page in physical memory
  Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit
  - For given logical address space 2^m *and page size* 2^n

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

Paging Hardware



Paging Model of Logical and Physical Memory



Paging

Example: A program's logical memory has been divided into five pages and these pages are allocated frames 2, 6, 3, 7 and 5. Show mapping of logical to physical memory

Paging

Example: A program's logical memory has been divided into five pages and these pages are allocated frames 2, 6, 3, 7 and 5. Show mapping of logical to physical memory

pages
Page 0
Page1
Page 2
Page 3
Page 4

Logical memory

	Frame no
0	2
1	6
2	3
3	7
4	5

Page table

	FRAMES
0	
1	
2	PAGE 0
3	PAGE 2
4	
5	PAGE 4
6	PAGE1
7	PAGE 3

Physical memory

Paging Hardware

There are 128 pages in a logical address space, with a page size of 1024 bytes. How many bits will be there in the logical address?

Solution:

Logical address space contains: 128 pages i.e. 2^7 pages, p = 7

Page size =1024 bytes= 2^{10} i.e. n =10

p = m-n

7 = m-10

m = 17

Therefore 17 bits in logical address space.

Paging Example

There is a system with 64 pages of 512 bytes page size and physical memory of 32 frames. How many bits are required in logical address?

Solution: page size = 512 = 2^9

n = 9

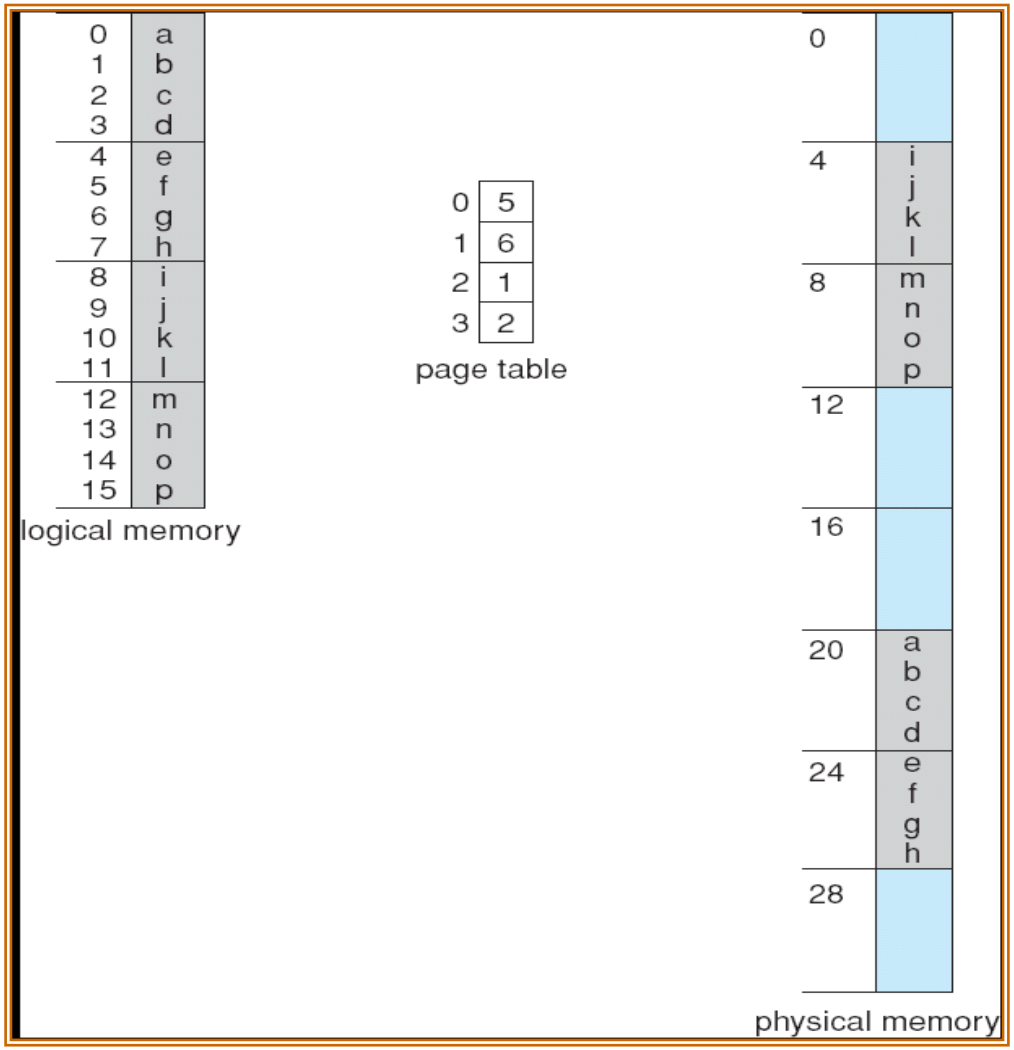
p = 6 // 64 pages = 2^6

p = m - n ;

m = p + n

m = 15

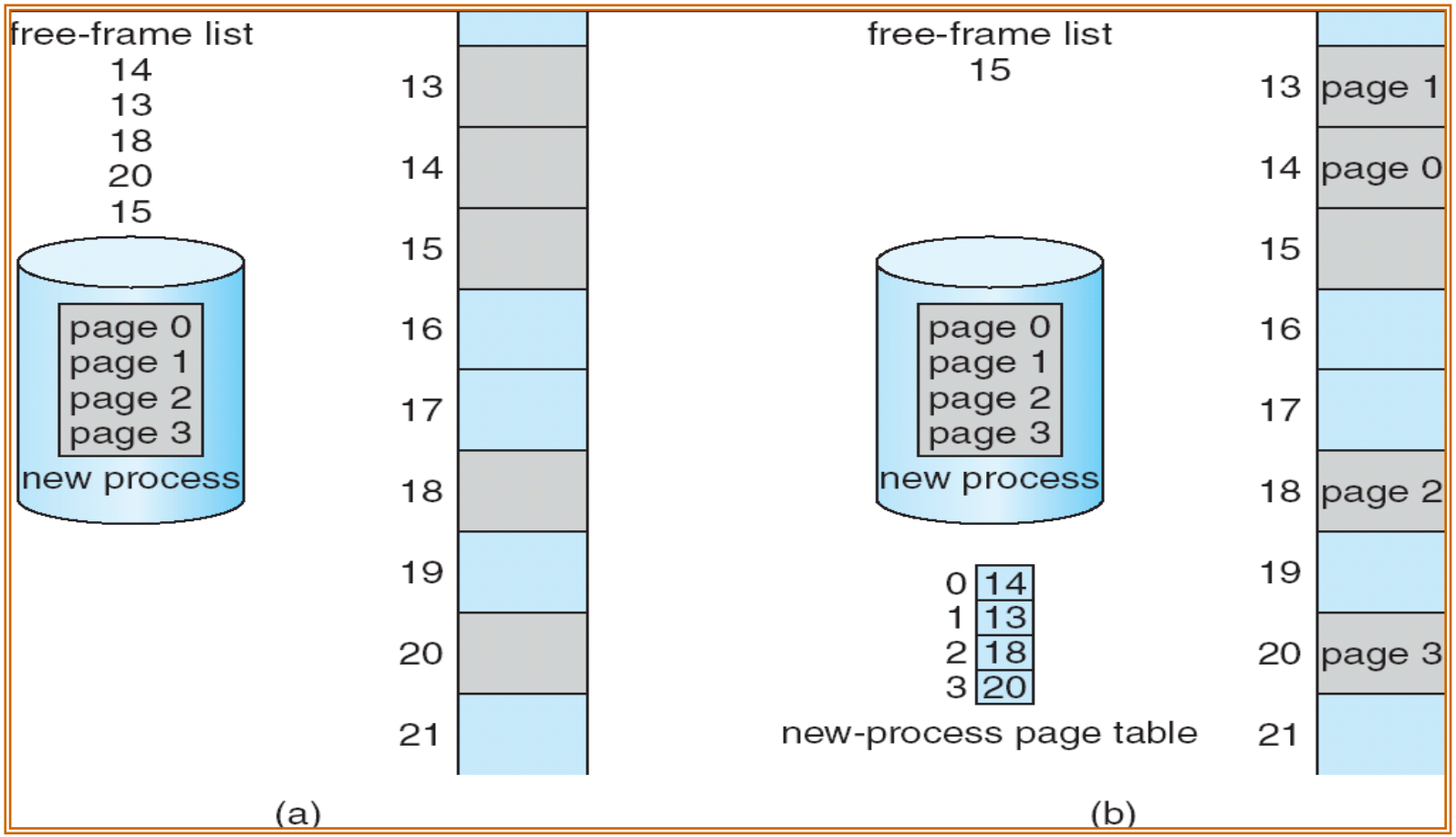
Paging Example



- The page size is of 4 bytes, frame size will also be same.
- Assume logical address (2,2)i.e. page no. is 2 and offset is 2.
- Page no. is used to index into the page table. Page2 is in frame 1.
- Actual location of frame 1 is at 1*4 bytes=4th byte in physical memory.
- So logical address will map to 4+2= 6th byte in physical memory.

32-byte memory and 4-byte pages

Free Frames



Before allocation **After allocation**

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 (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)

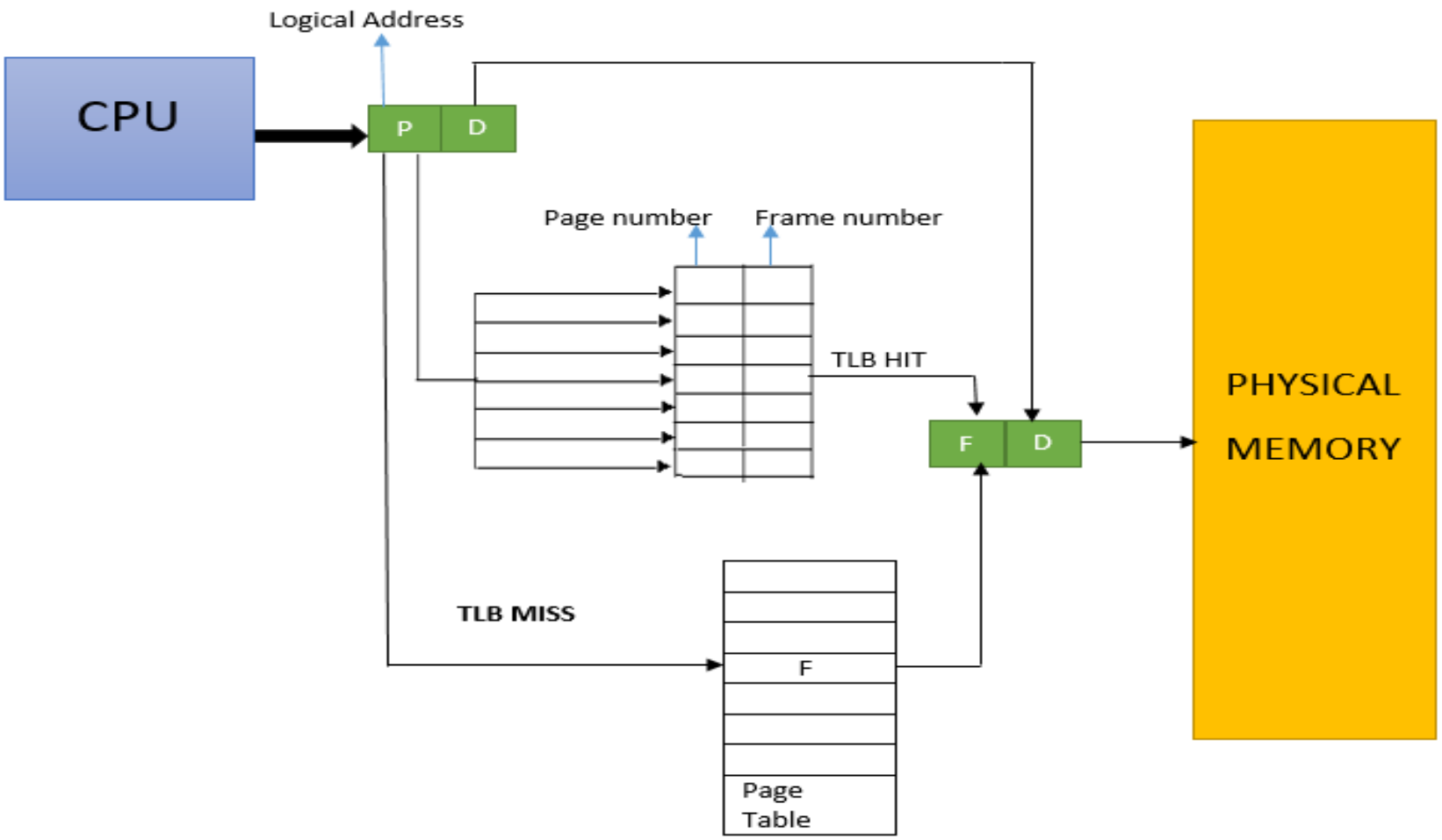
Translation Lookaside Buffer

- Contains page table entries that have been most recently used
- Functions same way as a memory cache

Translation Lookaside Buffer

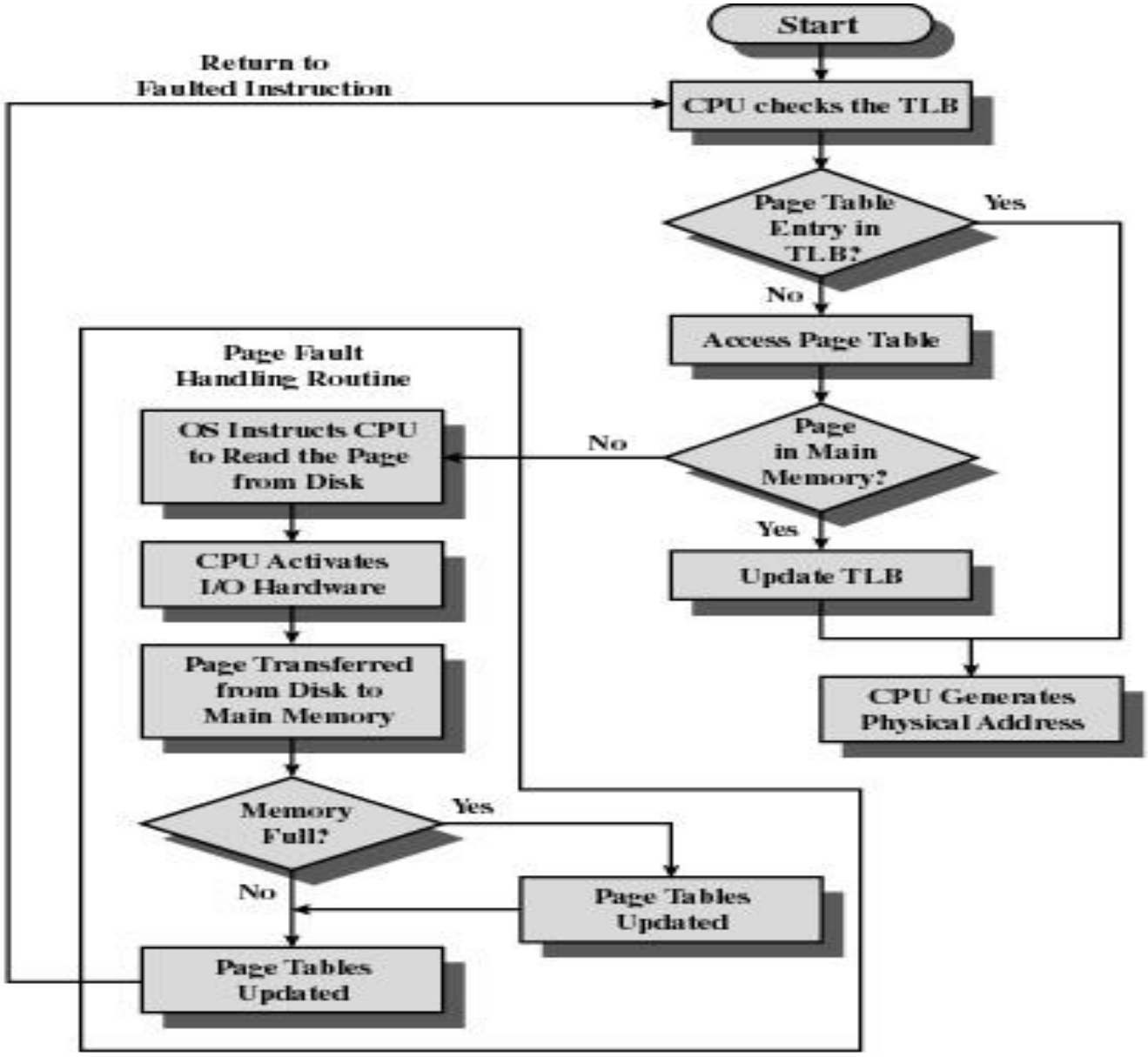
- Given a virtual address, processor examines the TLB
- If page table entry is present (a hit), the frame number is retrieved and the real address is formed
- If page table entry is not found in the TLB (a miss), the page number is used to index the process page table

Translation Lookaside Buffer



Translation Lookaside Buffer

- If TLB miss, OS checks if page is already in main memory
  - if not in main memory a page fault is issued
- The TLB is updated to include the new page entry



gure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

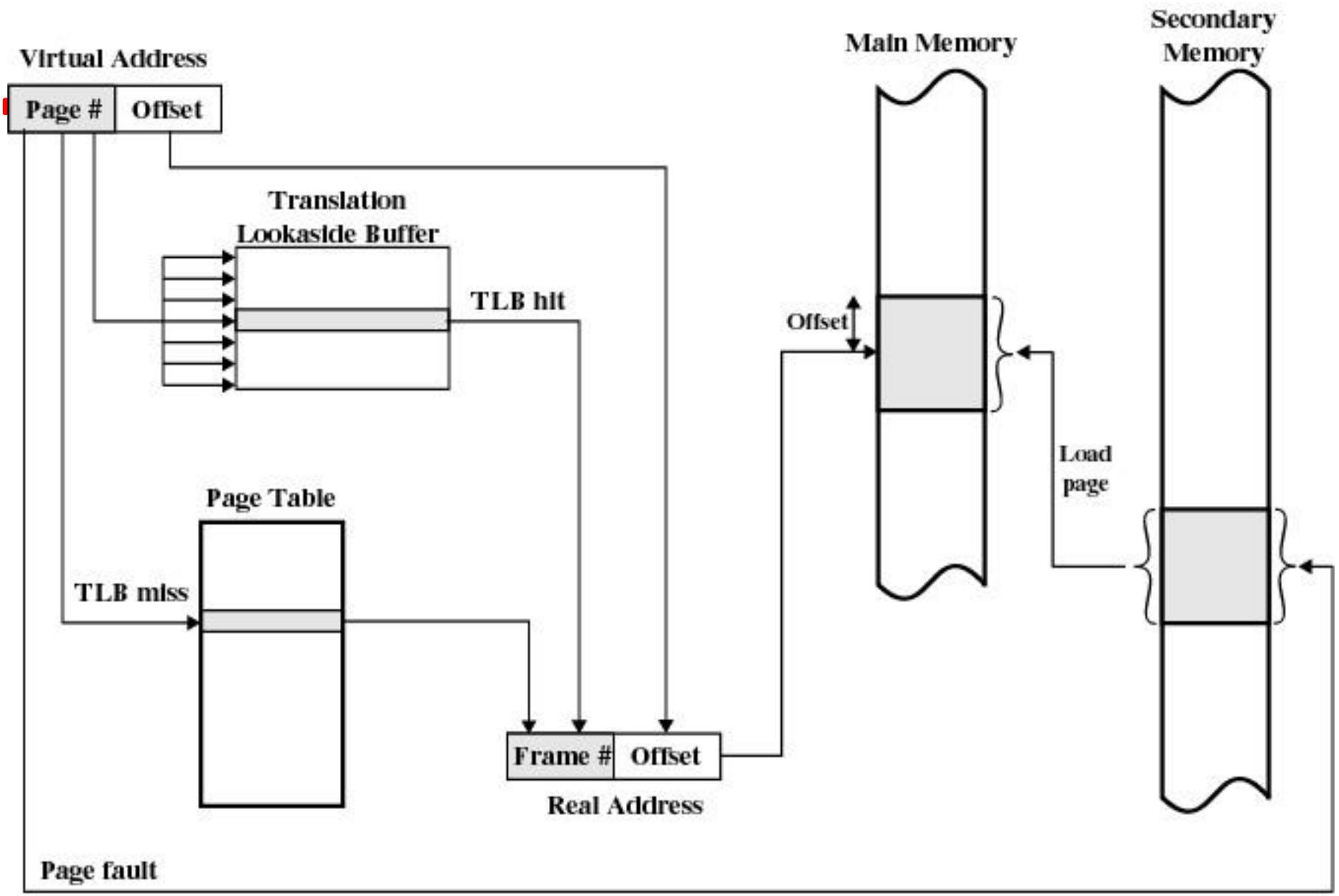


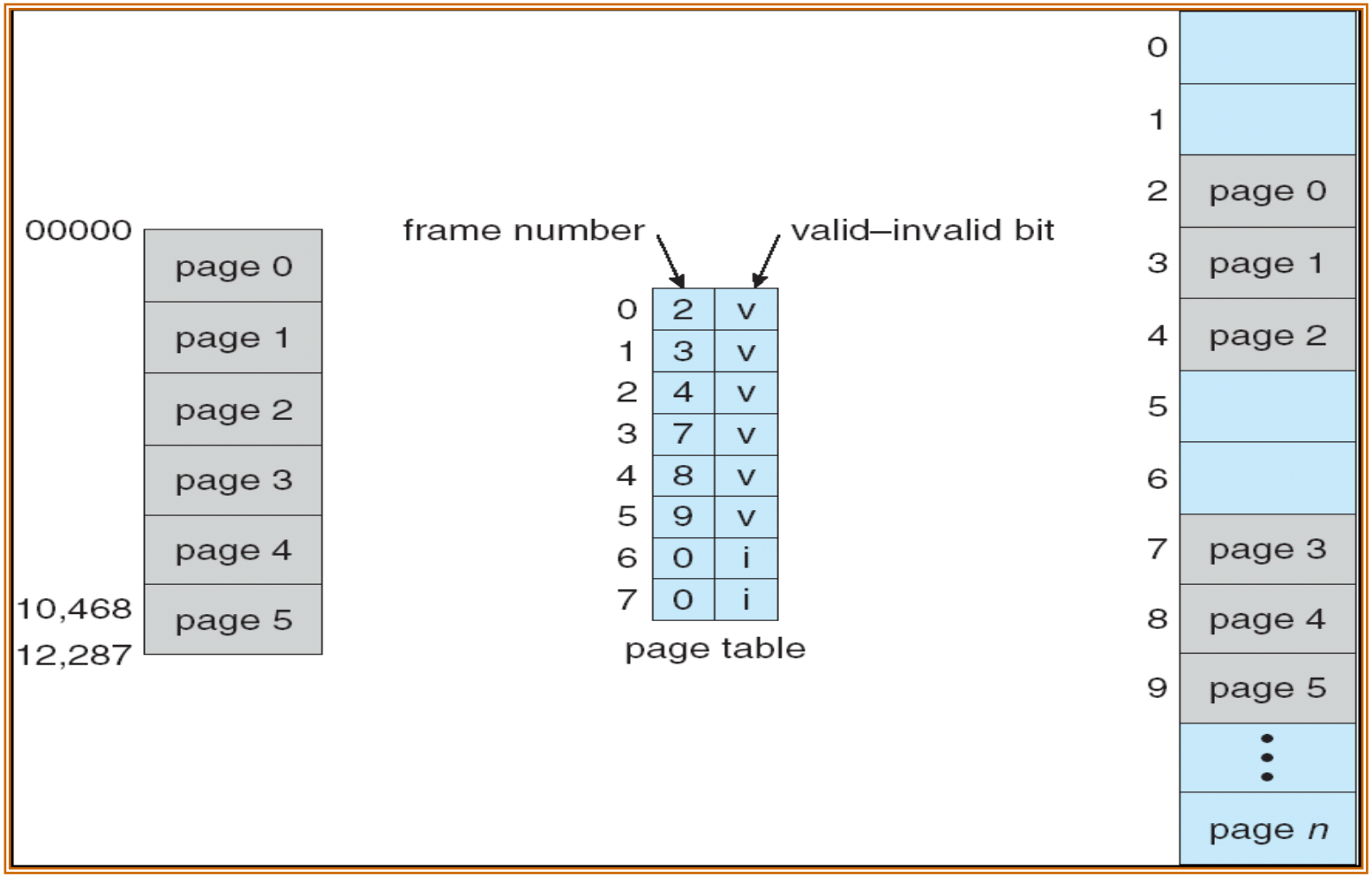
Figure 8.7 Use of a Translation Lookaside Buffer

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

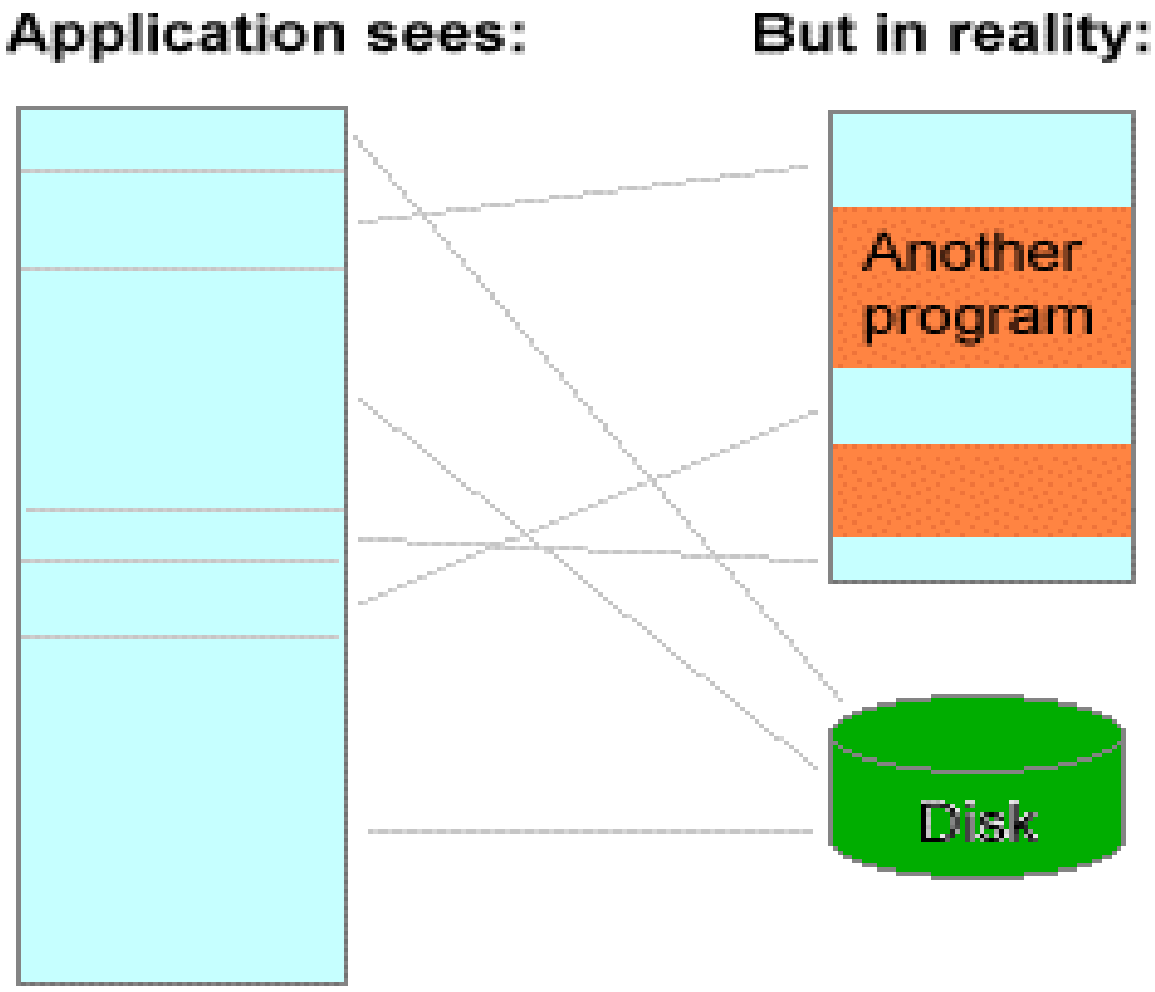


Virtual Memory

Hardware and Control Structures

- Memory references are dynamically translated into physical addresses at run time
  - A process may be swapped in and out of main memory such that it occupies different regions
- A process may be broken up into pieces that do not need to be located contiguously in main memory
  - All pieces of a process do not need to be loaded in main memory during execution

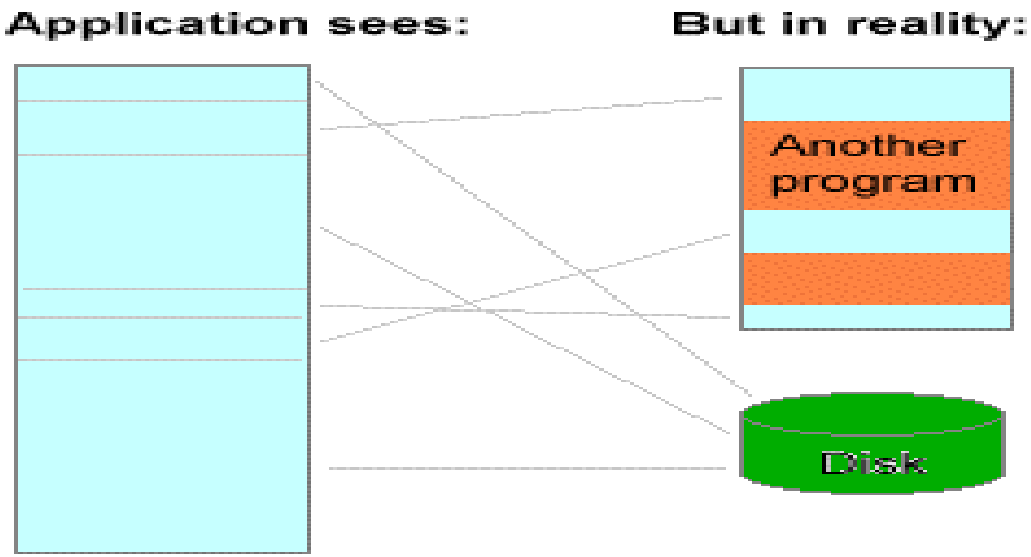
Virtual memory



Main memory

Execution of a Program

- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state



Advantages of Breaking up a Process

- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory

Types of Memory

- Real memory
  - Main memory
- Virtual memory
  - Memory on disk
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory

Modify Bit in Page Table

- A modify bit is needed to indicate if the page has been altered since it was last loaded into main memory
- If no change has been made, the page does not have to be written to the disk when it needs to be swapped out

Page Tables

- The entire page table may take up too much main memory
- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

Page Size

- Smaller page size, more pages required per process
- More pages per process means larger page tables
- Larger page tables means large portion of page tables in virtual memory
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

PROCESS 16K
P1 4K
P2 4K
P3 4K
P4 4K

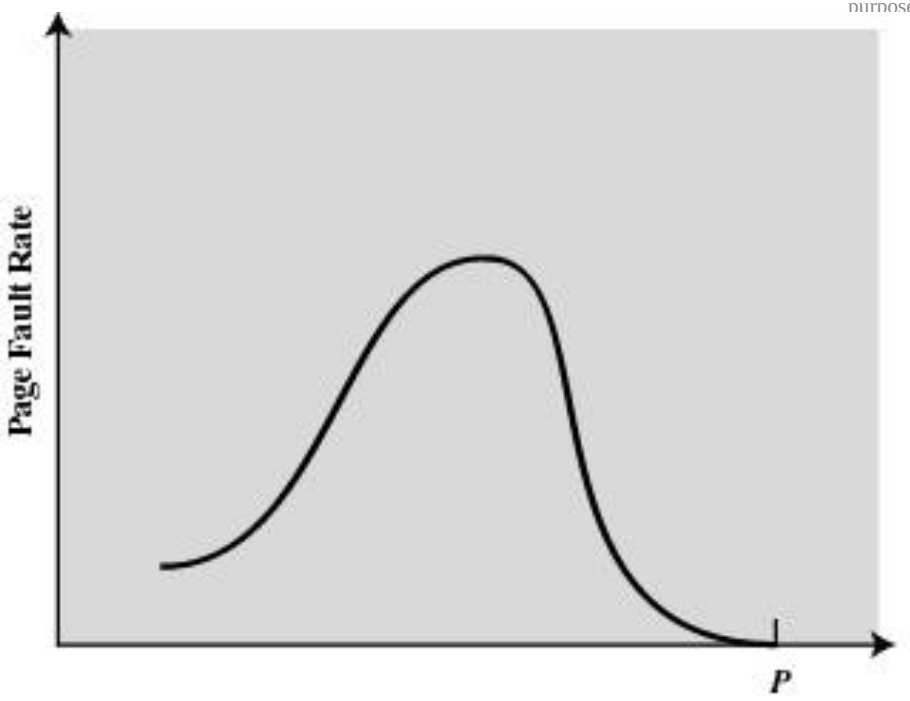
PROCESS 16K
P1 8K
P2 8K

Page Size

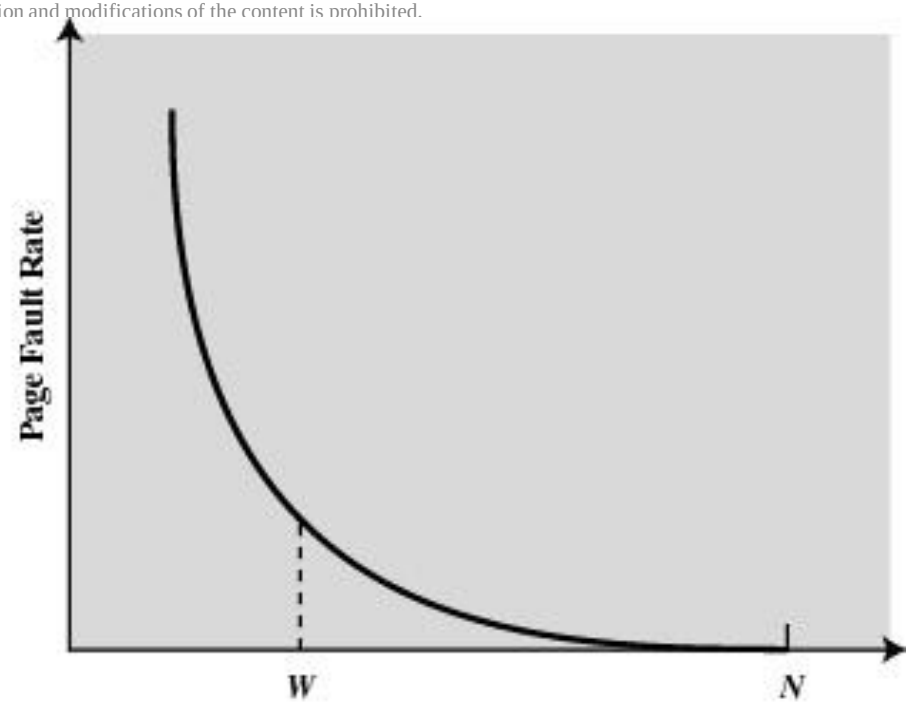
- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process near recent references. Page faults low.
- Increased page size causes pages to contain locations farther from any recent reference.
- Page faults rise.

PROCESS 16K
P1 4K
P2 4K
P3 4K
P4 4K

PROCESS 16K
P1 8K
P2 8K



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process
W = working set size
N = total number of pages in process

Figure 8.11 Typical Paging Behavior of a Program

Example Page Sizes

Table 8.2 Example Page Sizes

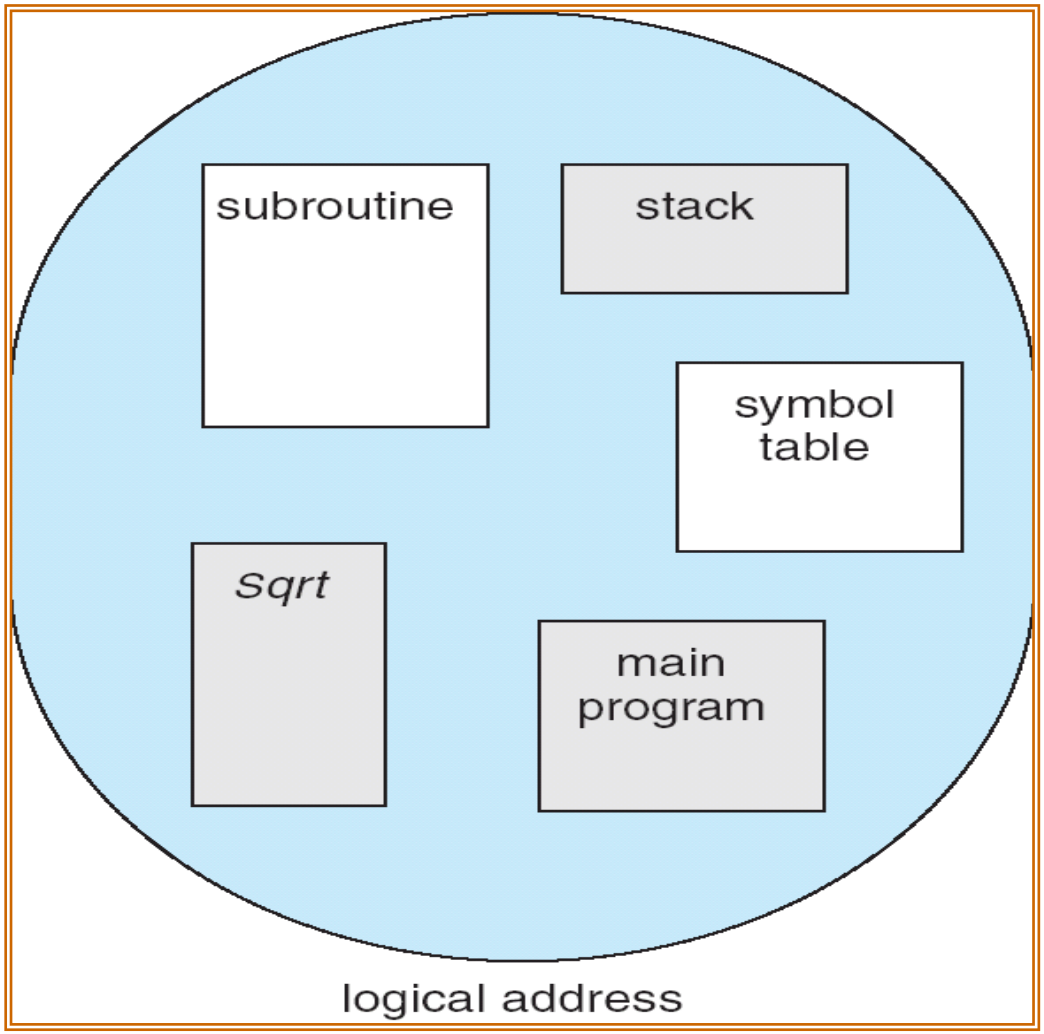
Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes

Segmentation

- Memory-management scheme that supports user's 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

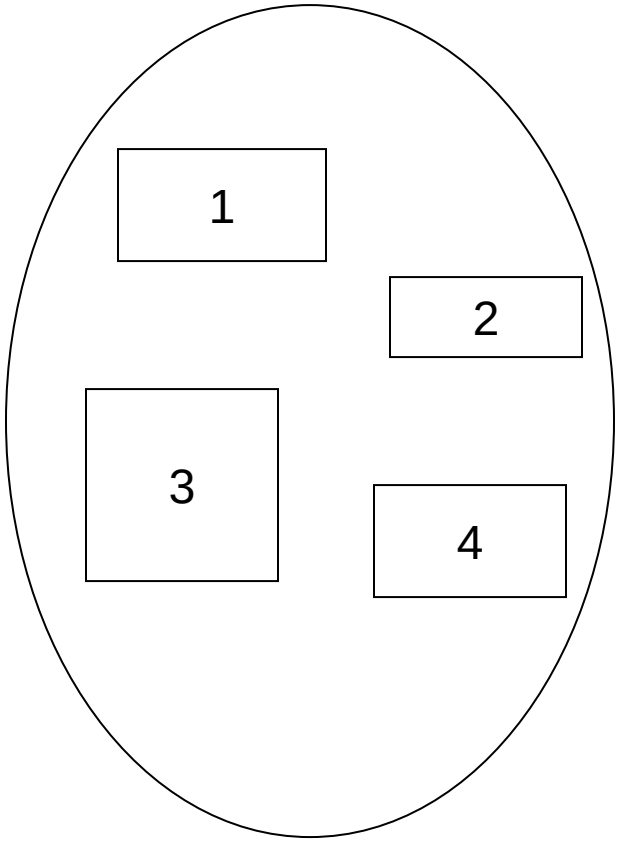
User's View of a Program



Segmentation

- All segments of all programs do not have to be of the same length
- There is a maximum segment length
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning

Logical View of Segmentation



user space

physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

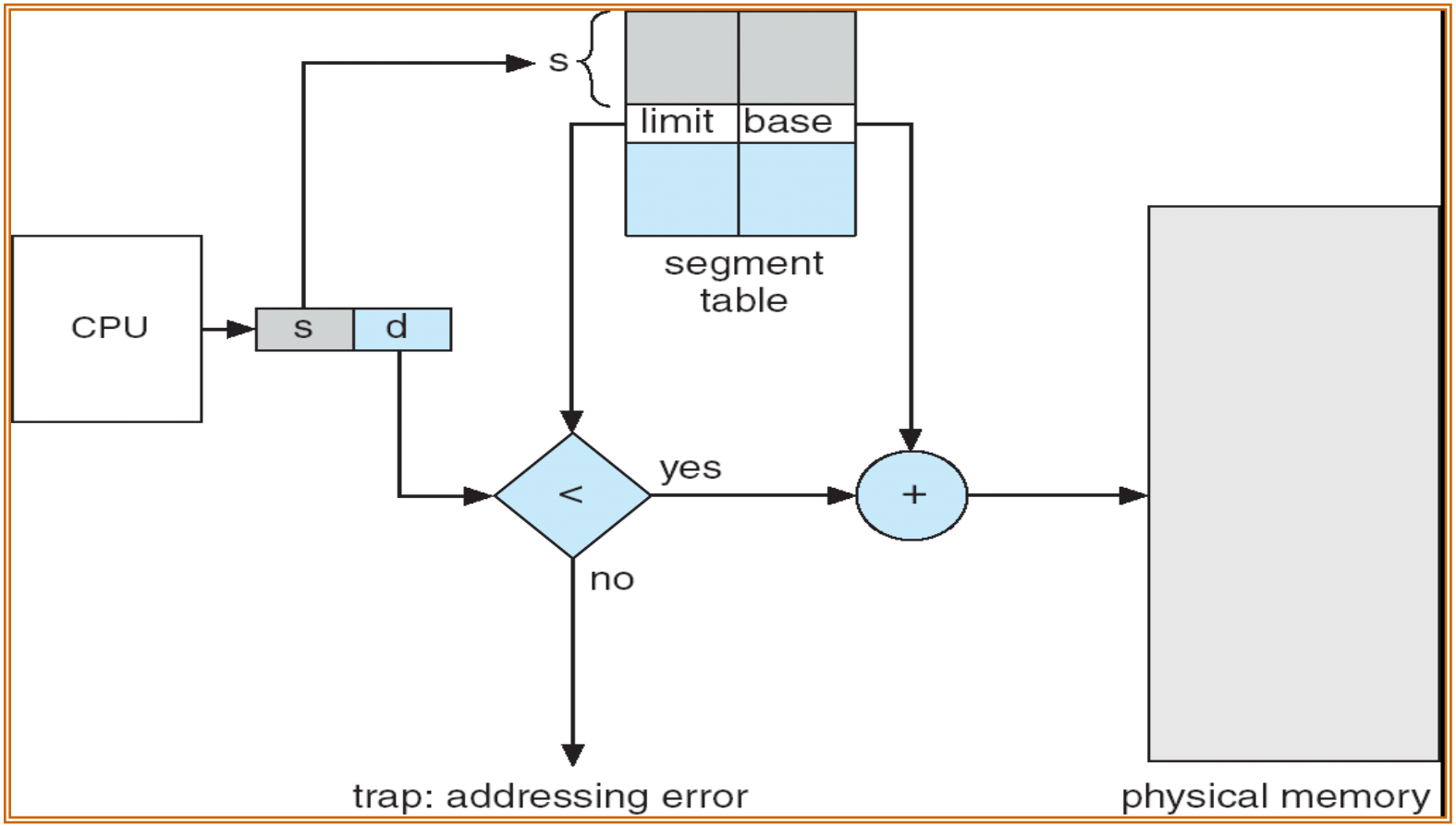
- Segment table – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- Segment-table base register (STBR) points to the segment table's location in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

  segment number *s* is legal if *s* < STLR

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Segmentation Hardware



Example of Segmentation