MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

CS234 Operating Systems

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Memory Management

- In Uni_programming environment main memory is divided into two parts, one for kernel and other for program being executed currently.
- In multiprogramming environment , user part of memory is further divided into multiple processes.
- The task of subdivision is carried out by OS & is known as Memory management
- Effective memory management is very important in multiprogramming environment.

Memory Management

- Subdividing memory to accommodate multiple processes
  - Done by Memory Management of OS
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical Organization
- Physical Organization

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

Memory Management Requirements

- Protection
  - Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses at compile time
  - Must be checked at run time
  - Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
    - Operating system cannot anticipate all of the memory references a program will make

Memory Management Requirements

- Sharing
  - Allow several processes to access the same portion of memory
  - Better to allow each process access to the same copy of the program rather than have their own separate copy

Memory Management Requirements

- Logical Organization
  - Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes
  - Segmentation satisfies these requirements.

Memory Management Requirements

Physical Organization

Computer memory is organized into at least two levels

- Main Memory: faster access, higher cost, volatile, no permanent storage, holds programs & data currently in use.
- Secondary memory: Slower , cheaper than main memory, not volatile, long term storage of programs & data

Organization of flow of information between main and secondary memory is major system concern

Individual programmer cannot take this responsibility

Memory Management Requirements

- Physical Organization

Main memory available for program and data may not be sufficient.

Overlaying technique can be used by programmer where various modules can be assigned to same region of memory & main program is responsible for switching in & out modules as and when needed. This leads to waste of time.

In multi programming environment programmer does not know at the time of coding, how much space is available for his/her program

Memory Management Schemes

- Memory Partitioning
  - Fixed Partitioning
    - Fixed size
    - Variable size
  - Dynamic Partitioning
- Paging
- Segmentation
- Virtual Memory

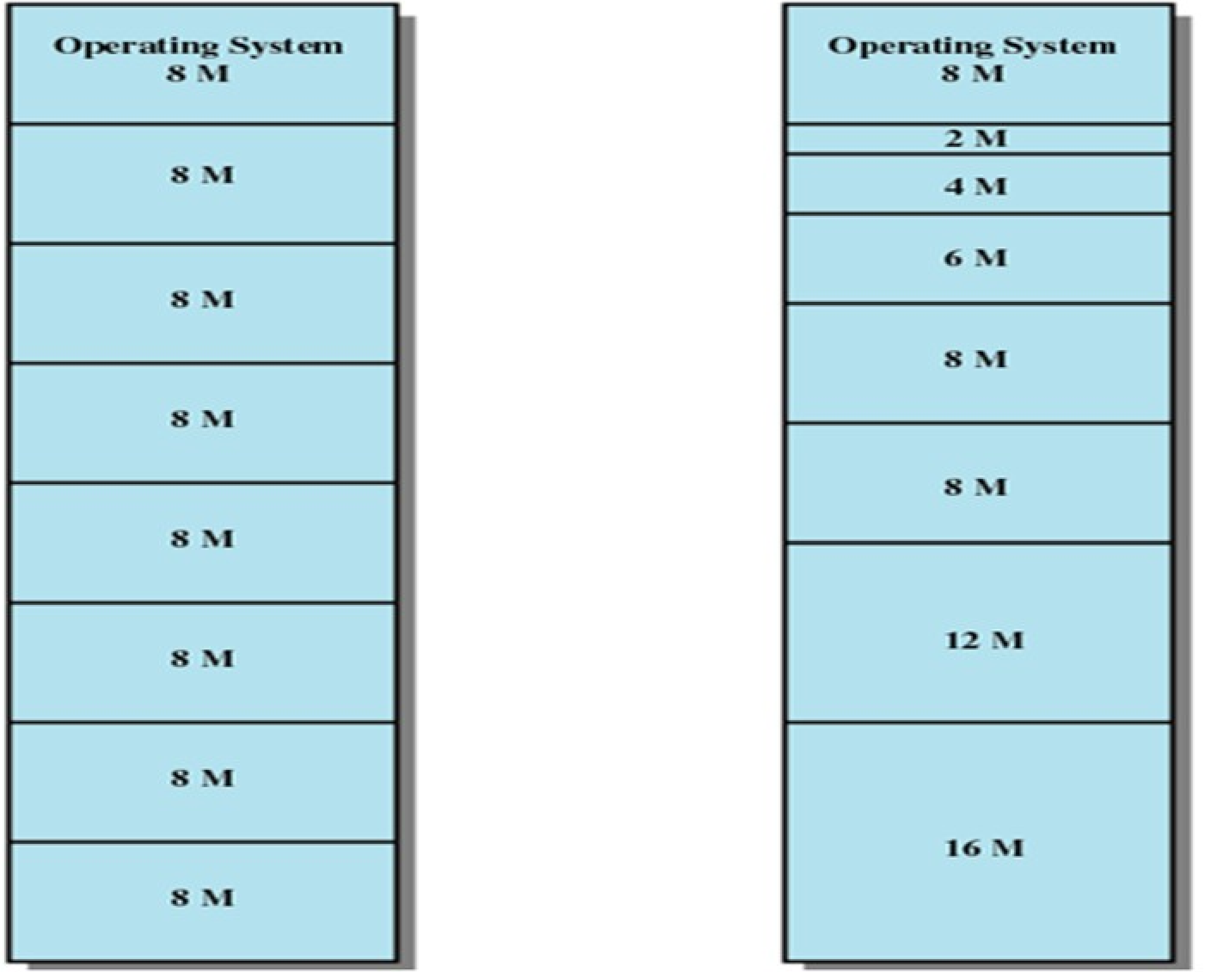
Memory Partitioning

Fixed Partitioning

OS occupies some fixed part of main memory. Rest of memory available for multiple processes.

This method divides main memory into regions with fixed boundaries.

- Make use of equal size partitions
- Make use of unequal size of partitions



(a) Equal-size partitions

(b) Unequal-size partitions

Example of Fixed Partitioning of a 64-Mbyte Memory

Fixed size partitions

Any process whose size is less than or equal to the size of partition size can be loaded into any available portion

Difficulties:

- Program may be too big to fit into memory, so programmer has to design the programs with **overlays**
- **Internal Fragmentation**: Program may be very small, but still gets loaded into partition leaving lot of space free in that partition. Wasted space internal to a partition due to a fact that block of data loaded which is smaller than the size of partition, is known as Internal Fragmentation

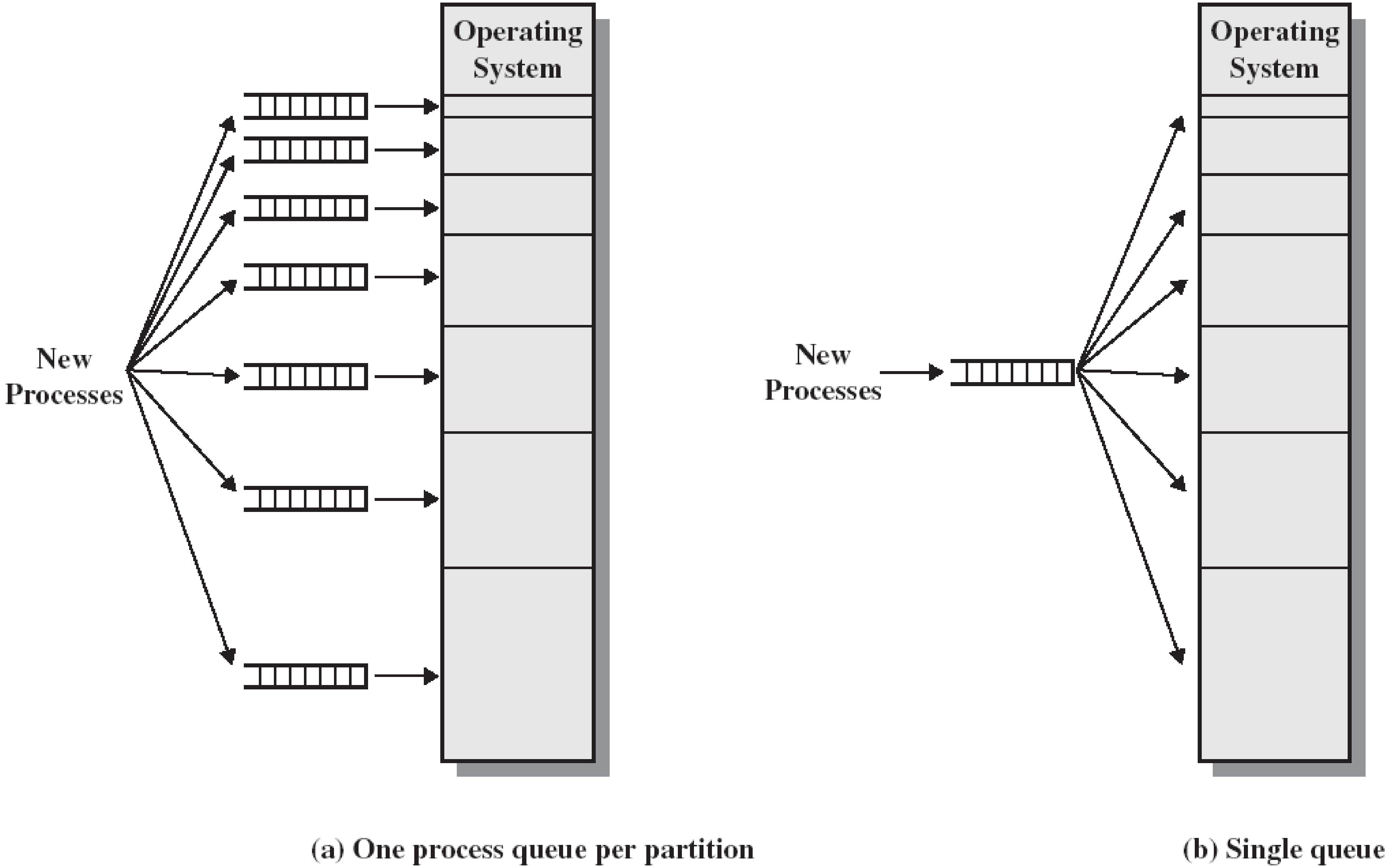
Unequal sized partitions

- Partitions are made but of unequal sizes.
- As long as partition is available , process can be loaded into memory.
- No particular portion is favored.
- When no process is available for execution (all processes blocked), scheduler will take the decision to swap out a process to make space available for new process.

Scheduling Q strategies

Scheduling Q for each partition: Each partition will have its Q of Processes. Processes are always assigned in such a manner that there is minimum internal fragmentation.

Single Q for all processes: Provides a degree of flexibility to fixed partitioning.



Memory Assignment for Fixed Partioning

Disadvantages of fixed partitions

No of partitions are decided at the time system generation. This limits to no. of active processes.

Main storage requirement of all jobs should be known well in advance

Small jobs will not use partition space efficiently, leading to internal fragmentation.

Fixed partitioning is almost unknown today

E.g. Multiprogramming Fixed Tasks (IBM mainframe OS-MFT)

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

Example

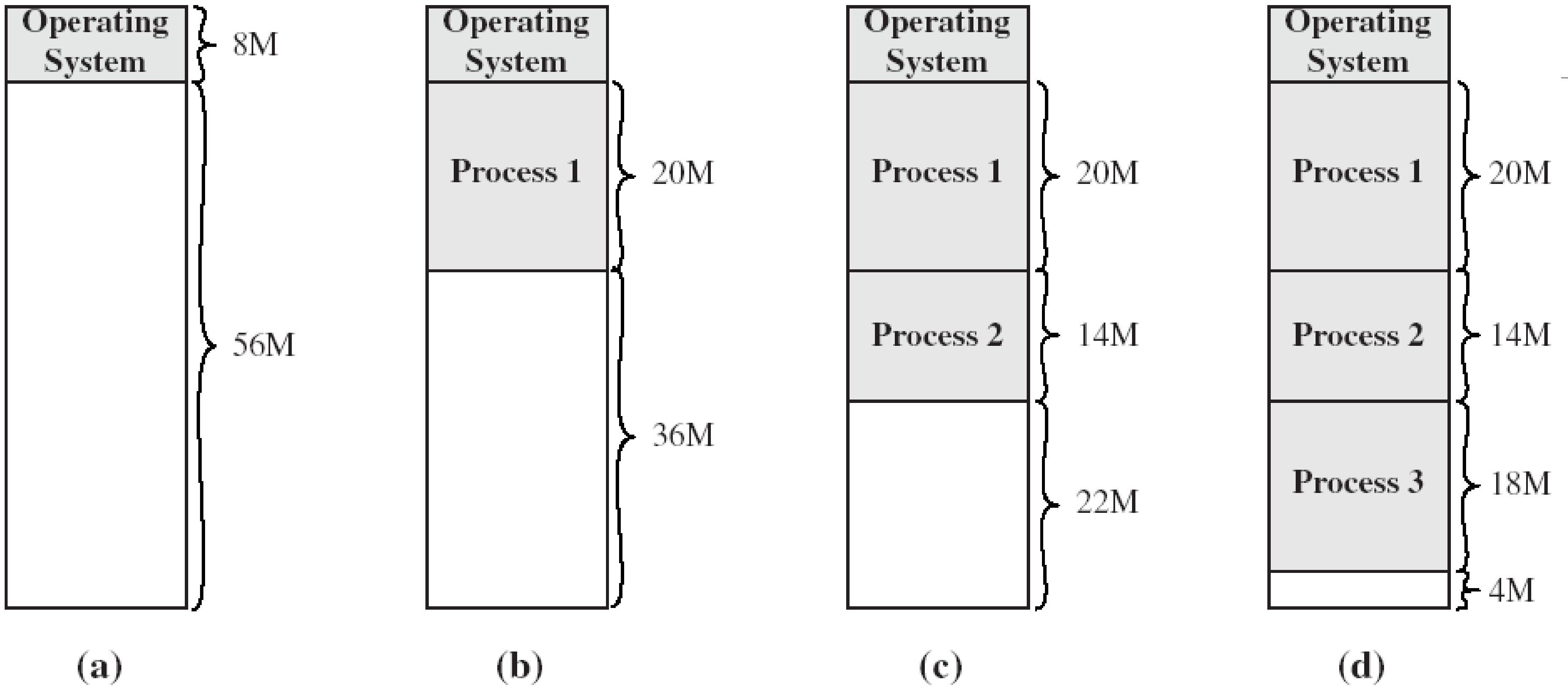
Assume 64m, out of which 8m occupied by OS. Initially no processes in memory. Processes p1,p2,p3 arrive having sizes 20m,14m,18m respectively.

At some time all 3 processes are blocked and hole of 4m is created, which is not sufficient to accommodate a new process,p4 of size 8m.

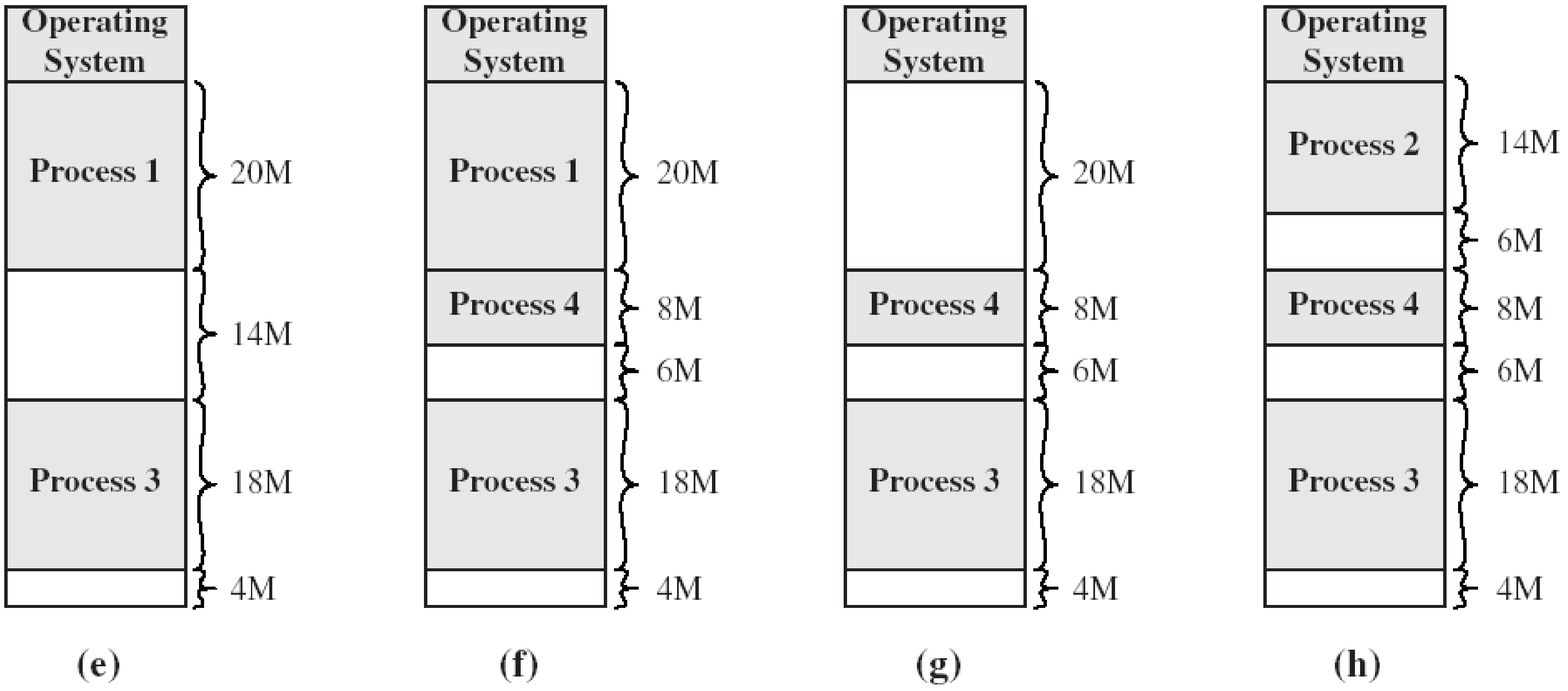
P2 is swapped out and p4 occupies that space again creating a new hole of size 6m.

Suppose p4 is also blocked but p2 is ready.

Swap p1 & place p2 another hole of 6m created



The Effect of Dynamic Partitioning



The Effect of Dynamic Partitioning

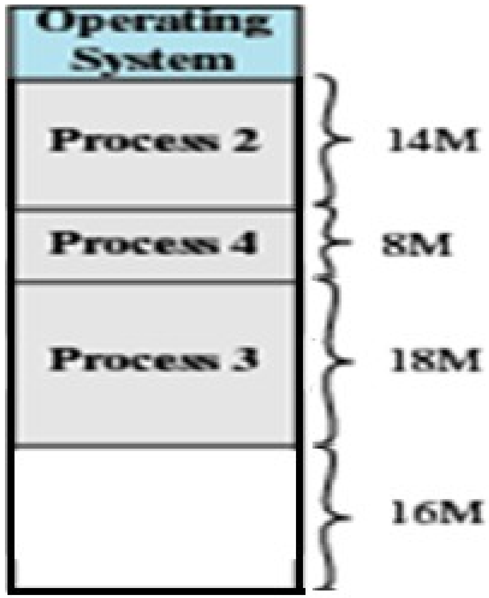
Dynamic partitioning contd

As time goes, memory becomes more & more fragmented.

External Fragmentation: Memory that is external to partitions becomes increasingly fragmented.

Compaction:

- OS shifts the processes, so that all free memory is available in one block.
- Time consuming process and wasteful of processor time
- Compaction requires dynamic relocation capability.



Dynamic Partitioning

- Compaction is time consuming
- Set of holes of various sizes, is created throughout memory.
- When new process arrives, & needs memory, system searches for set of holes & finds a hole that is large enough to for that process.
- If hole is too large, it is split into two parts, one part is allocated to process while, other is returned to a set of holes.
- When process is swapped out/ terminated , it releases its block of memory to the set of holes.
- If new hole is adjacent to other holes, adjacent holes are merged to form a new hole. OS checks whether any waiting process can fit into a combined hole.

Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process.
- Types:
  - Best-fit algorithm
  - First-fit algorithm
  - Worst-fit algorithm
  - Next-Fit algorithm

Strategies for dynamic storage allocation

First Fit: Allocate the first hole that is big enough. Searching can begin at the start of set of holes.

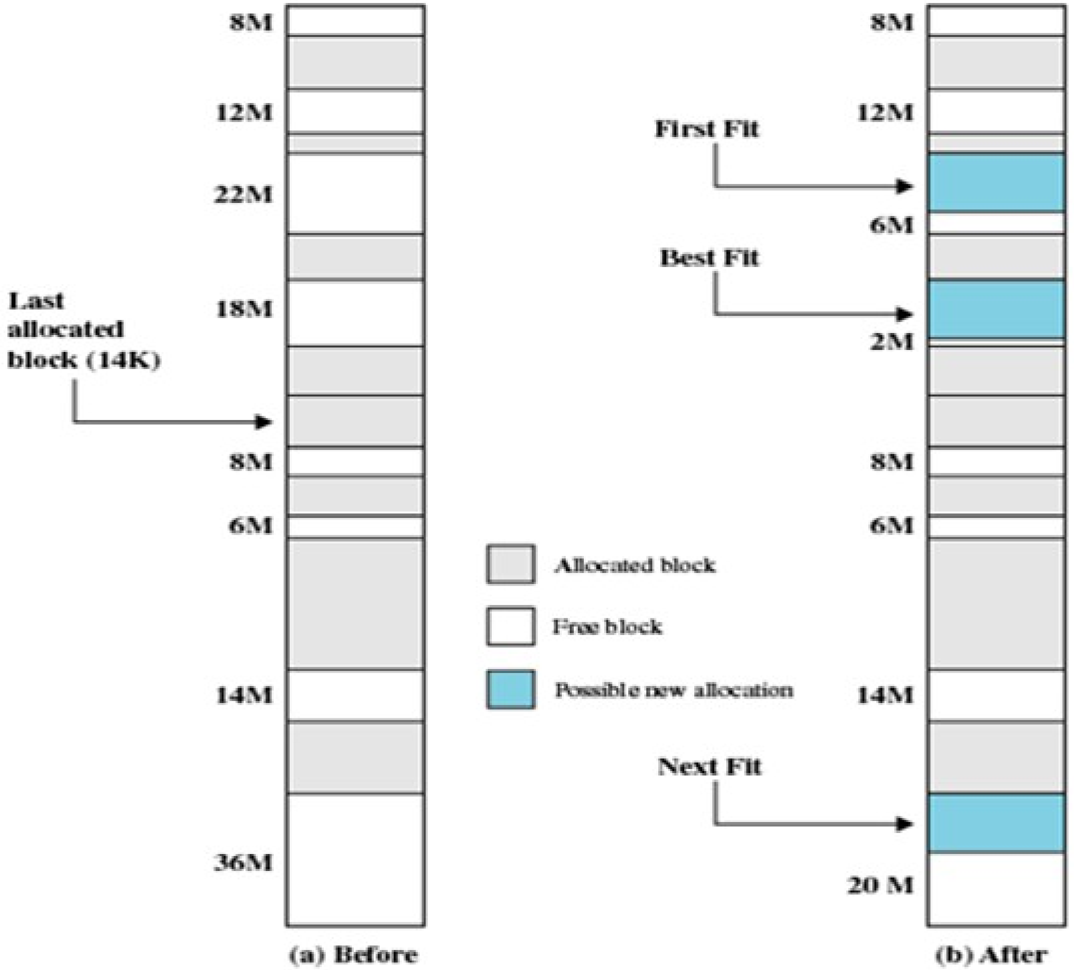
Best Fit: Allocate the smallest hole that is big enough. Search entire list, unless the list is kept ordered by size.

Worst Fit: Allocate the largest hole. Search the entire list unless sorted by size.

Next Fit: Allocate the first hole that is big enough. Searching can begin where the previous search ended.

Algorithm
New process of size 16M to be allocated

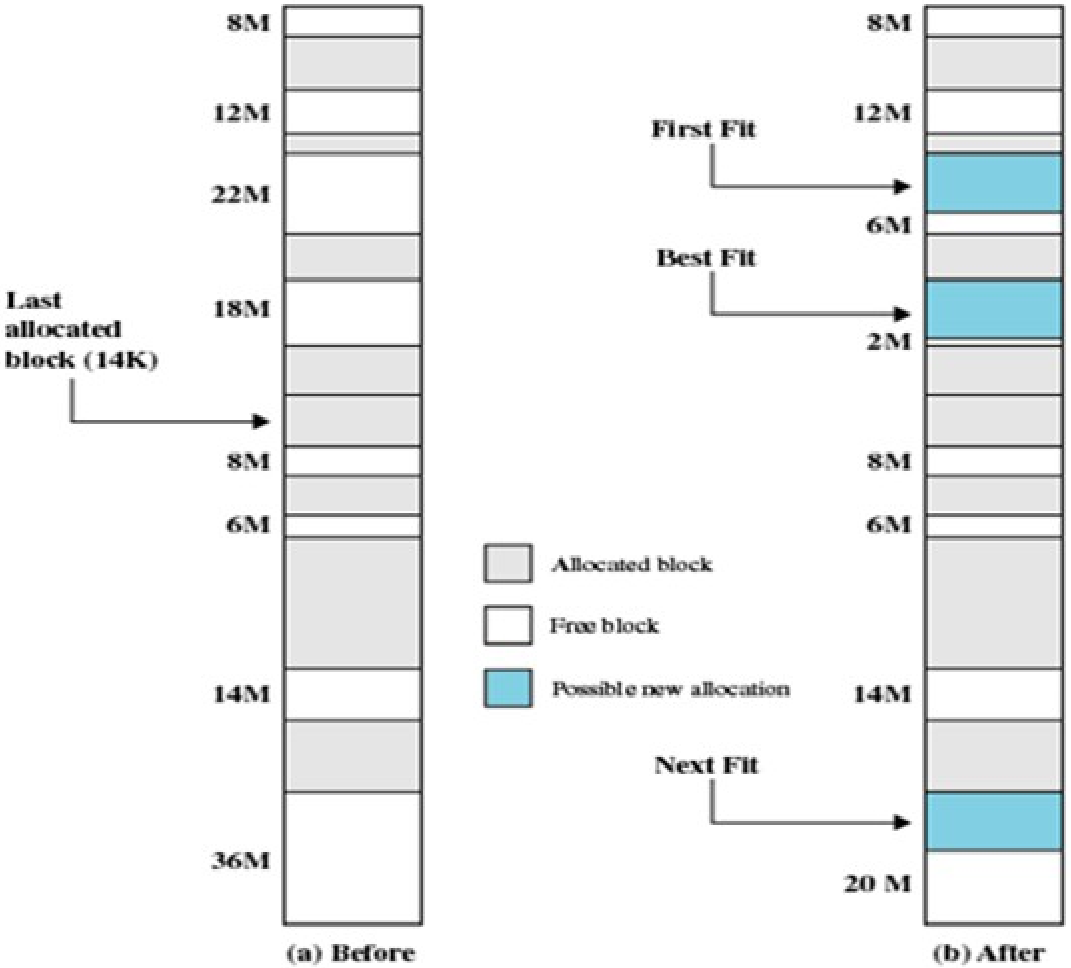
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often



Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Partitioning Placement Algorithm

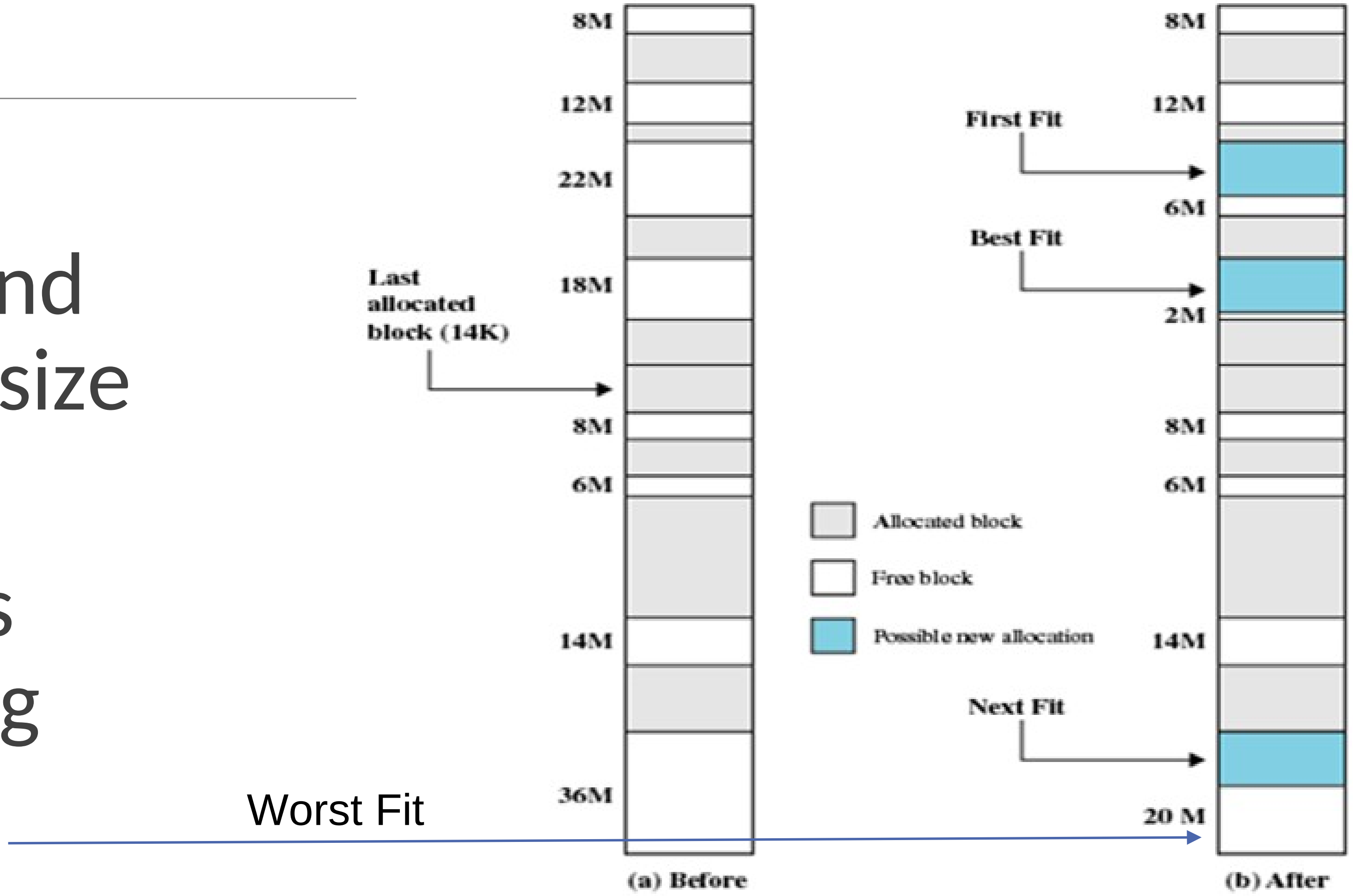
- First-fit algorithm
  - Scans memory from the beginning and chooses the first available block that is large enough
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block



Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Partitioning Placement Algorithm

- Worst-fit algorithm
- Scan the entire list and find the holes having size >= size of process.
- Amongst these holes select that hole having largest size



Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Example 1

Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in first fit, best fit, worst fit?

MIT-WPU

First Fit: Search from beginning
Search ends when you find the
Hole having size >= size of process

Holes → 15 | 10 | 5 | 25 | 30 | 40

Processes

12 → 15: 15-12=3

Modified list of holes - After assigning 15K hole to 12K process

03 | 10 | 5 | 25 | 30 | 40

2 → 03: 3-2=1

Modified list of holes - After assigning 3K hole to 2K process

01 | 10 | 5 | 25 | 30 | 40

25 → 25: 25-25=0

Modified list of holes - After assigning 25K hole to 25K process

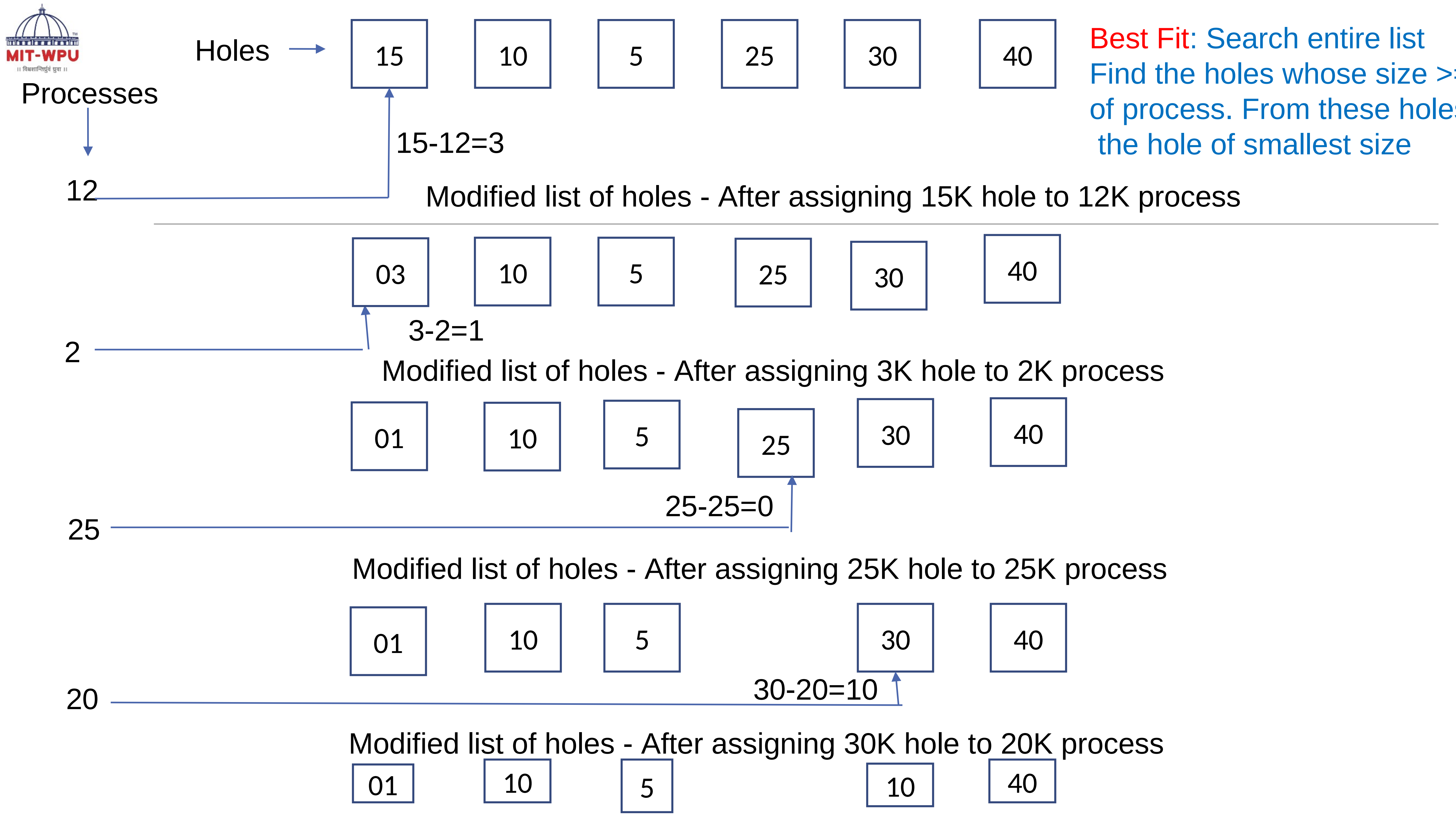
01 | 10 | 5 | 30 | 40

20 → 30: 30-20=10

Modified list of holes - After assigning 30K hole to 20K process

01 | 10 | 5 | 10 | 40

Best Fit: Search entire list
Find the holes whose size >= size of process. From these holes select the hole of smallest size



Worst Fit: Search entire list
Find the holes whose size >= size of process. From these holes select the hole of largest size

Holes →

15	10	5	25	30	40

Processes ↓

12

40-12=28

Modified list of holes - After assigning 40K hole to 12K process

15	10	5	25	30	28

2

30-2=28

Modified list of holes - After assigning 30K hole to 2K process

15	10	5	25	28	28

25

28-25=03

Modified list of holes - After assigning 28K hole to 25K process

15	10	5	25	03	28

20

28-20=08

Modified list of holes - After assigning 30K hole to 20K process

15	10	5	25	03	08

Paging

Paging is a memory management scheme that permits the physical address space of a process to be noncontiguous.

Support for paging has been handled by hardware.

Recent designs have implemented paging by closely integrating H/W & OS, especially on 64 bit processors.

Paging

Partition memory into small equal-size chunks and divide each process into the same size chunks

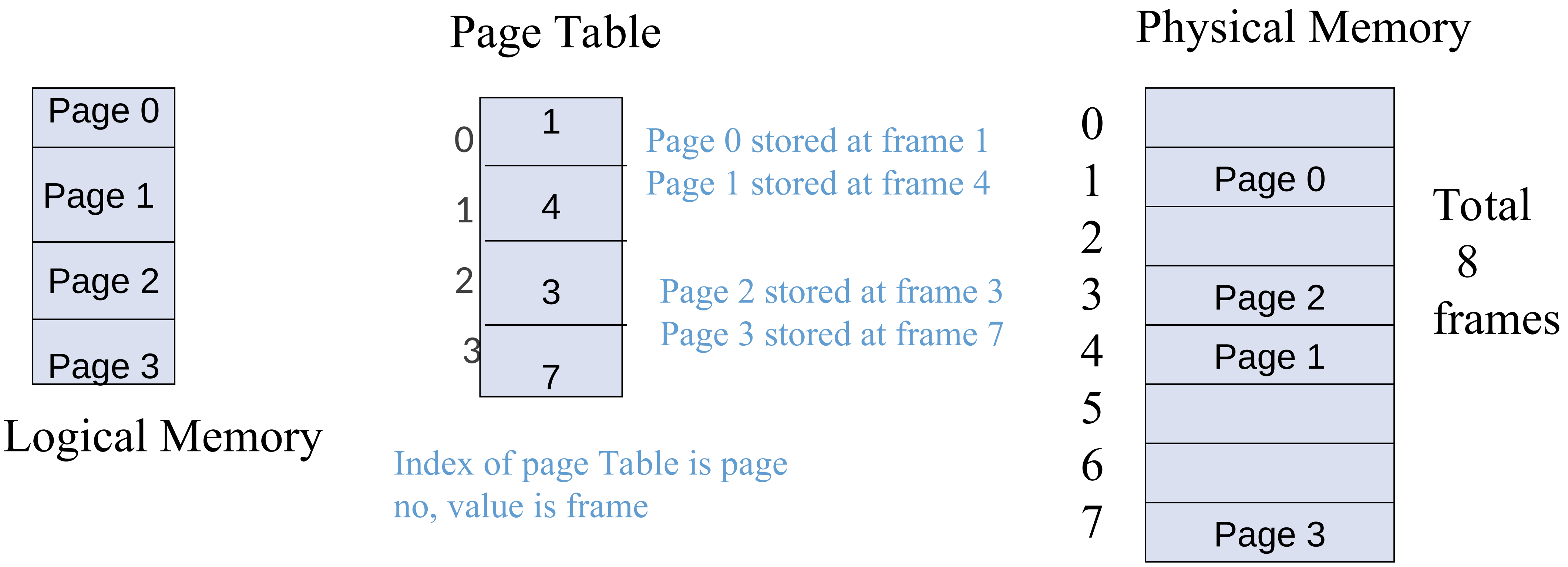
The chunks of a process are called pages and chunks of memory are called frames

Operating system maintains a page table for each process

- contains the frame location for each page in the process
- memory address consist of a page number and offset within the page

Basic Method

Physical memory is broken into fixed sized blocks called frames. Logical memory is broken into blocks of same size called Pages.



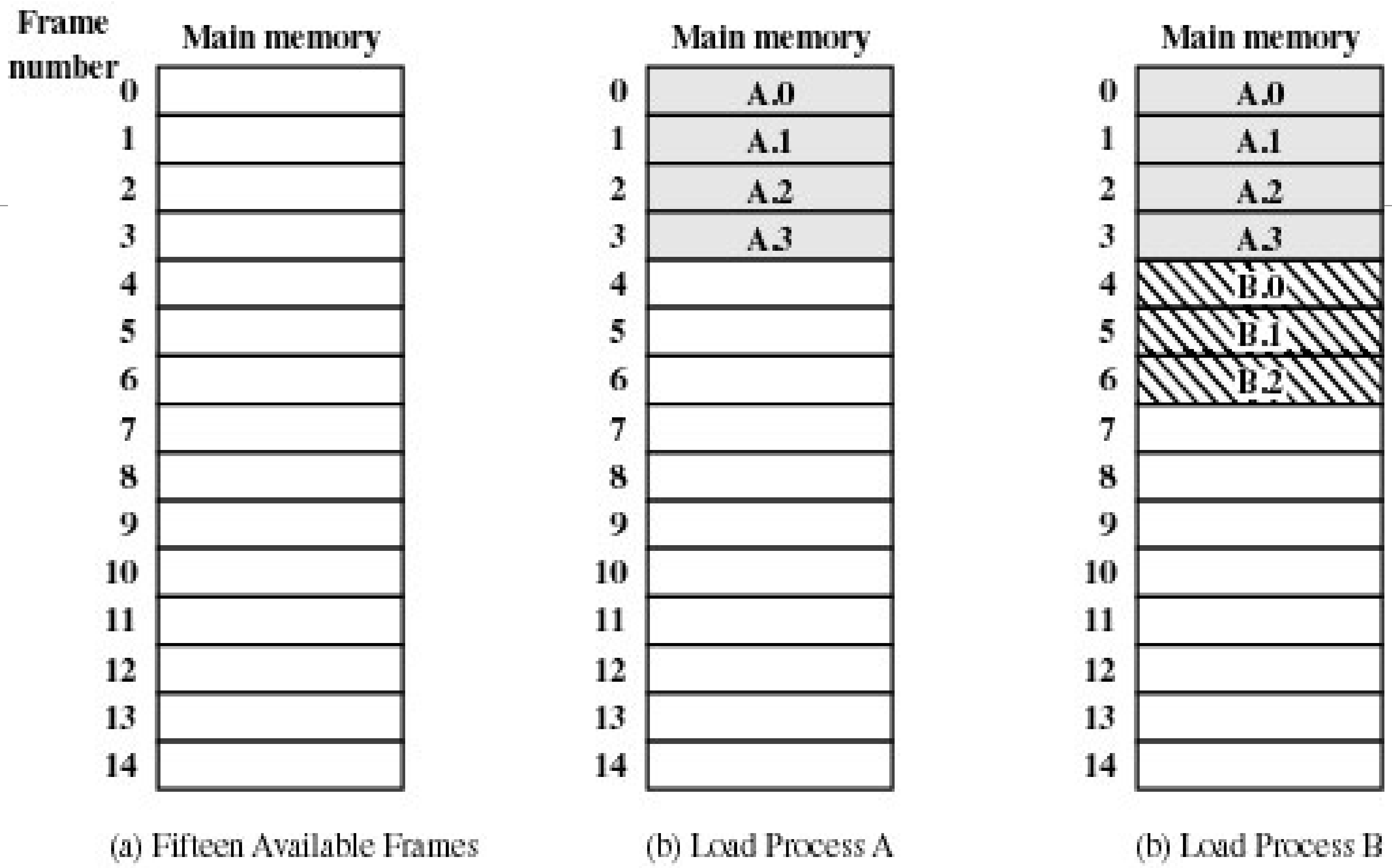


Figure 7.9 Assignment of Process Pages to Free Frames

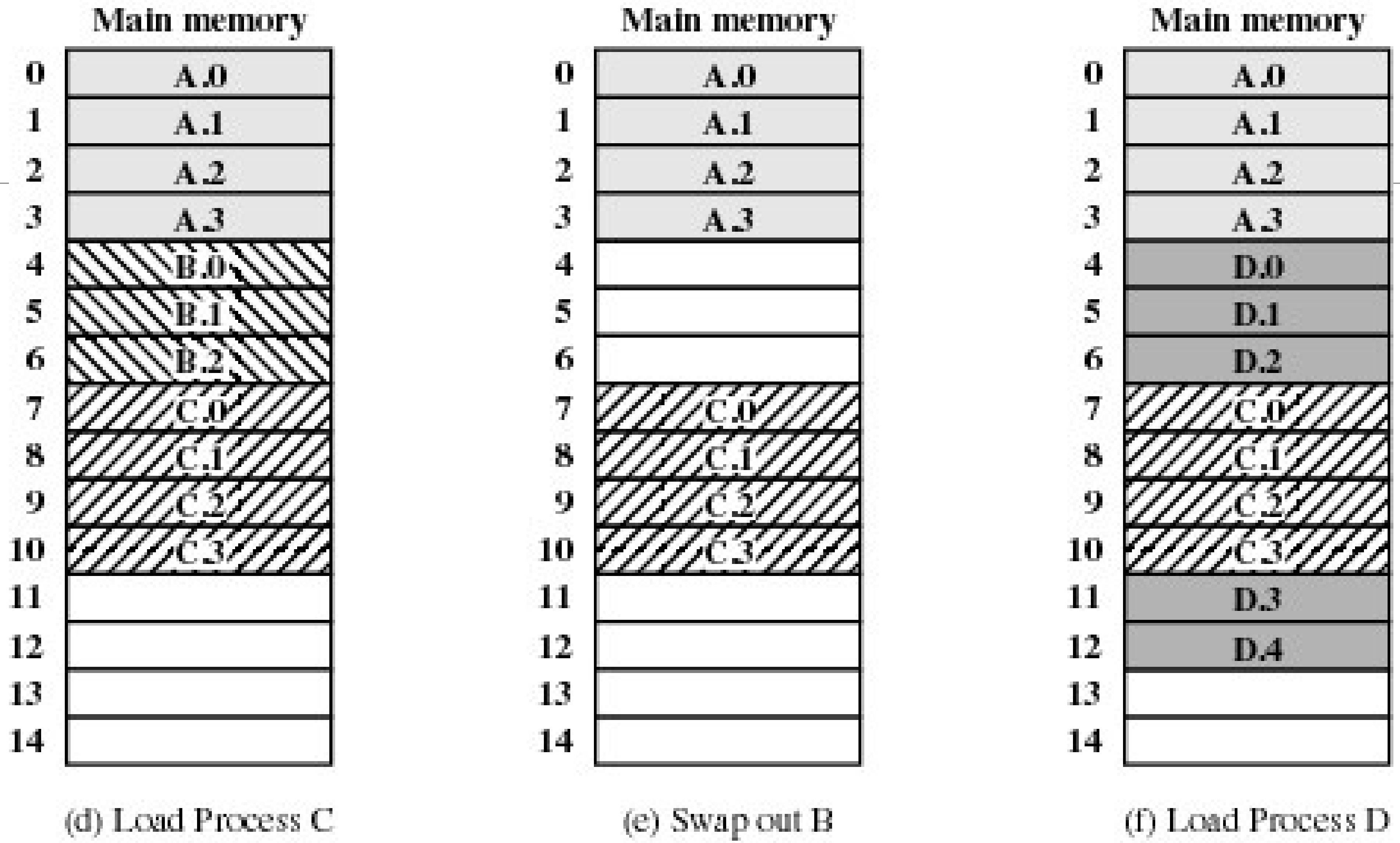


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables

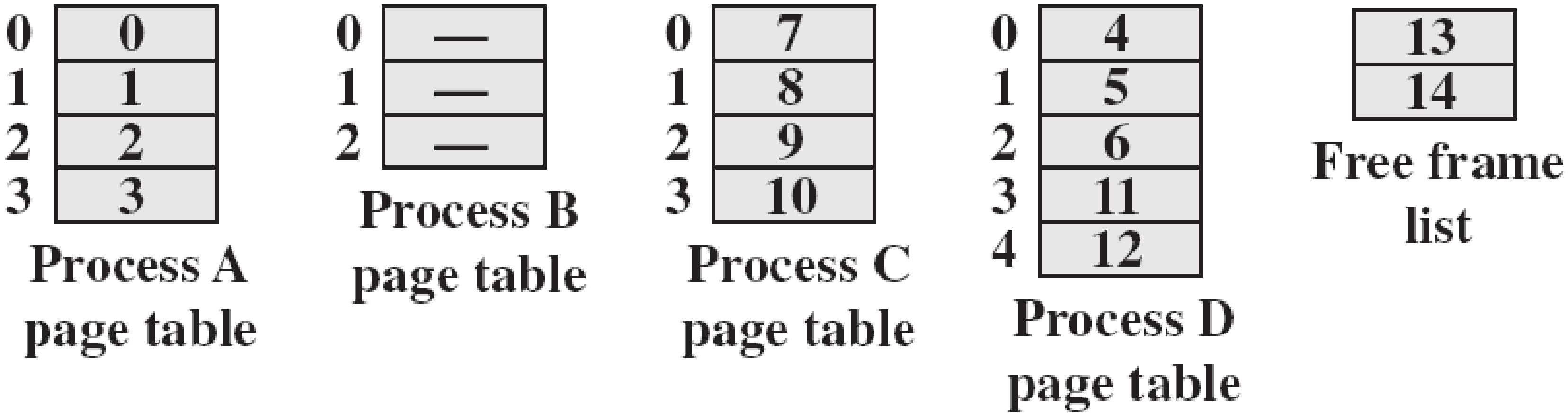


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Paging: Description

Page size defined by H/W, typically power of 2. Selection of power of 2 as a page size makes the translation of logical address into page no. and page offset easy.

No external fragmentation.

Any free frame can be allocated to a process that needs it.

Internal fragmentation may exist.

Example: Internal fragmentation in paging

Process size 72,766 bytes

Page size 2048 bytes

Find the size of internal fragmentation

Ex. Internal fragmentation in paging

Process size 72,766 bytes

Page size 2048 bytes

It will require 36 frames out of which 35 will be fully occupied, while last one giving rise to internal fragmentation of size 962

72766 / 2048 = 35.53

2048 * 35 = 71680 35 total occupied pages

72,767 – 71,680 = 1086 remaining bytes of process

36th page: 2048 – 1086 = 962

Process size – remaining process = int. fragmentation

Issues in page size

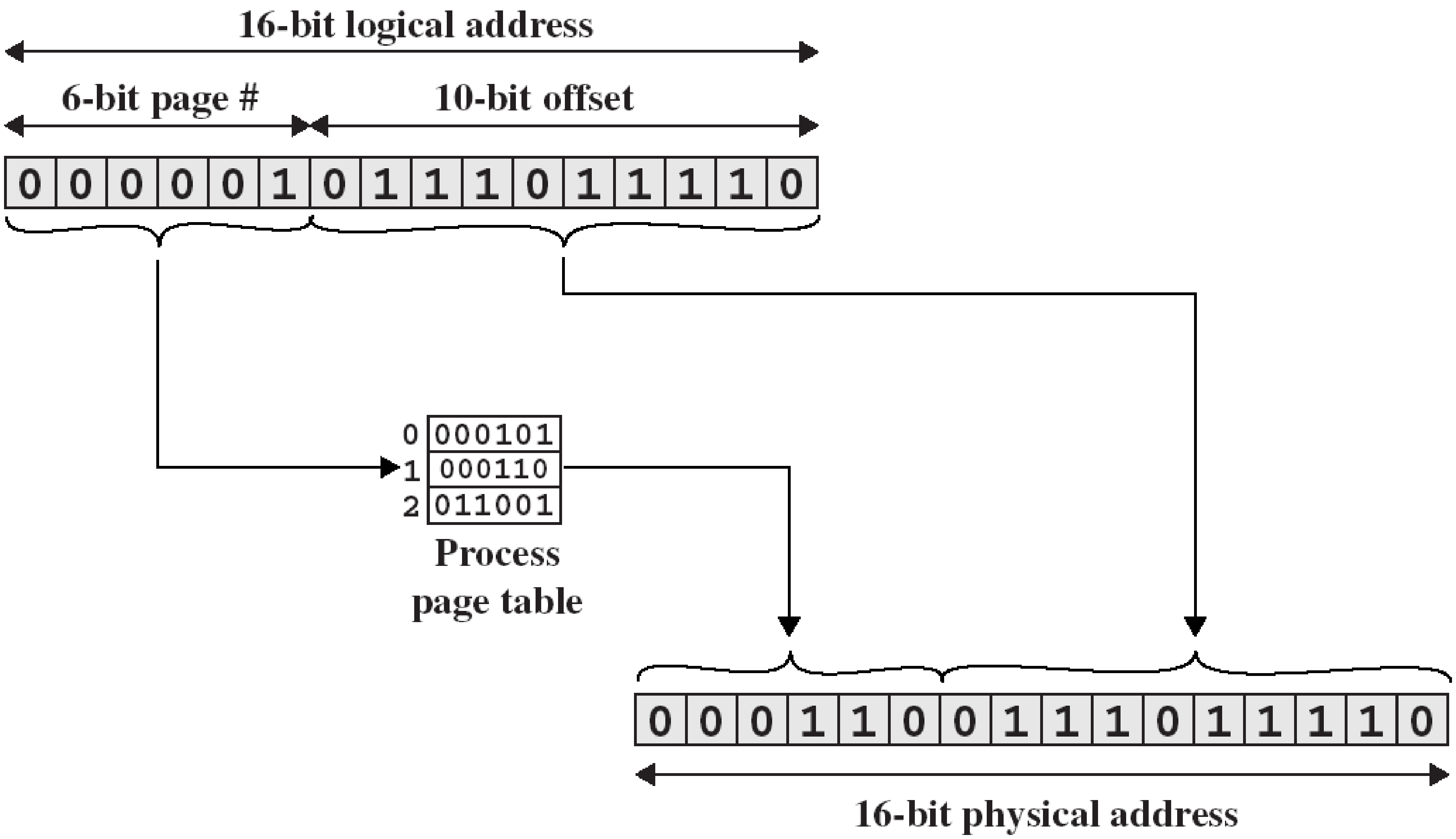
If process size is independent of page size, we expect internal fragmentation to average one half page per process.

Small page sizes are desirable, but more overhead involved in each table entry which can be reduced if size of page increases.

Page size 4kb to 8kb

Each table entry is usually 4 bytes long, but that size can vary.

Ex-Page Translation



Paging Contd.

Paging separates user's view of memory & actual memory.

User feels logical memory as one contiguous space where in only his/her program resides.

In physical memory user's program is scattered along with other programs.

Address translation H/W maps user's view to physical address and is hidden from user & controlled by OS.

OS must be aware of allocation details of physical memory.

OS maintains frame table which stores information about total, allocated and available frames.

OS maintains a copy of page table for each process, which is used to translate logical address to physical address, also used by dispatcher to define H/W page table to assign process to CPU

Paging increases context switch time.

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,

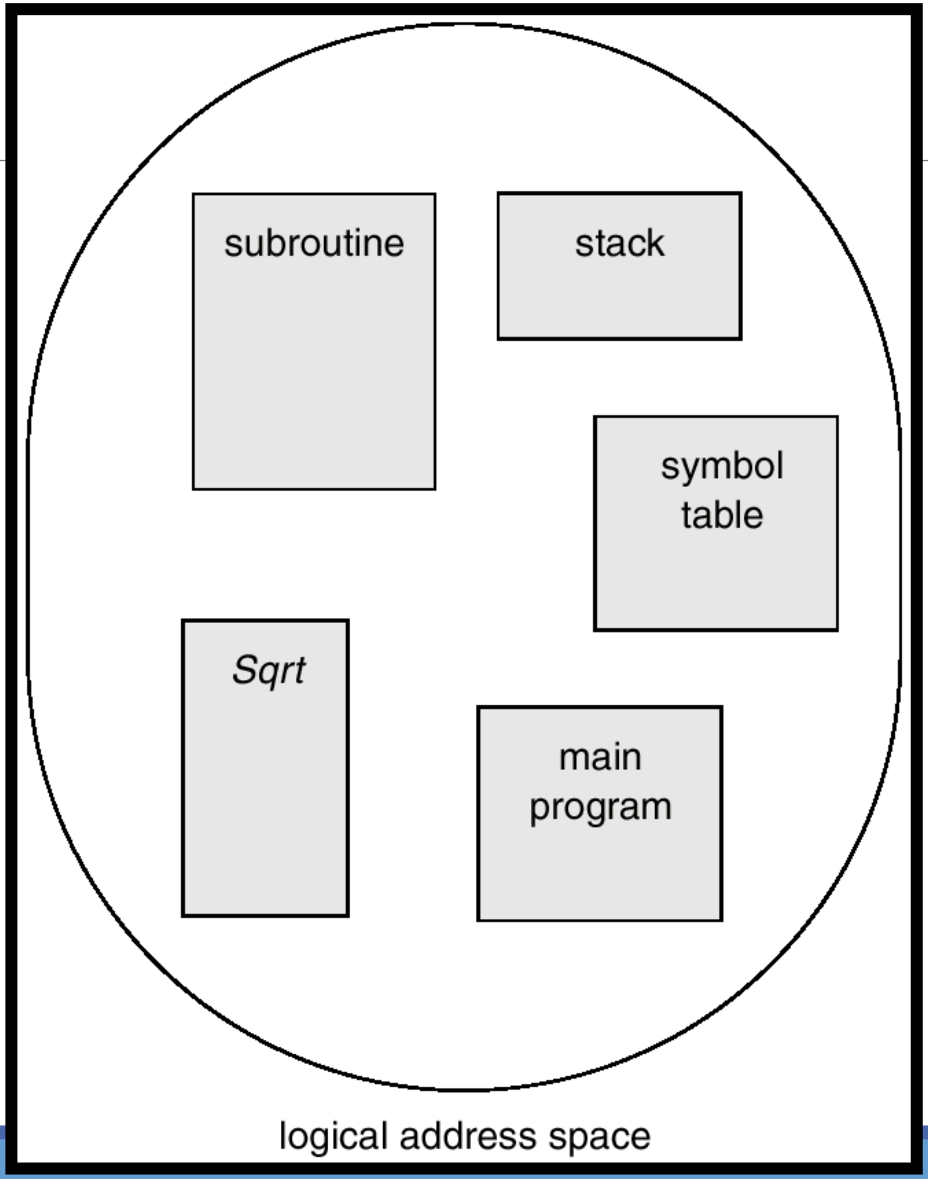
local variables, global variables,

common block,

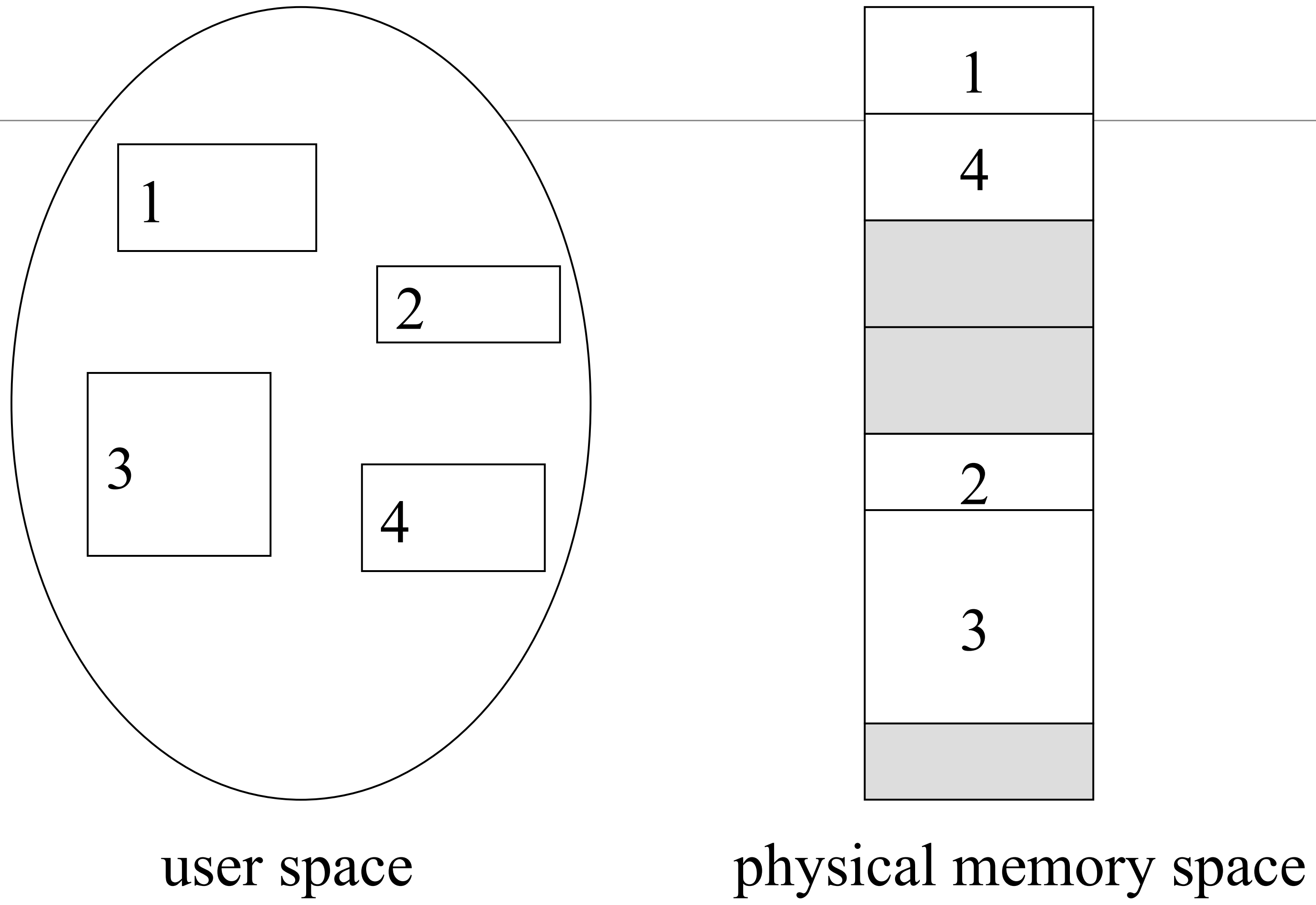
stack,

symbol table, arrays

User’s View of a Program



Logical View of Segmentation



Segmentation Architecture

Logical address consists of a two tuple:

<segment-number, offset>,

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

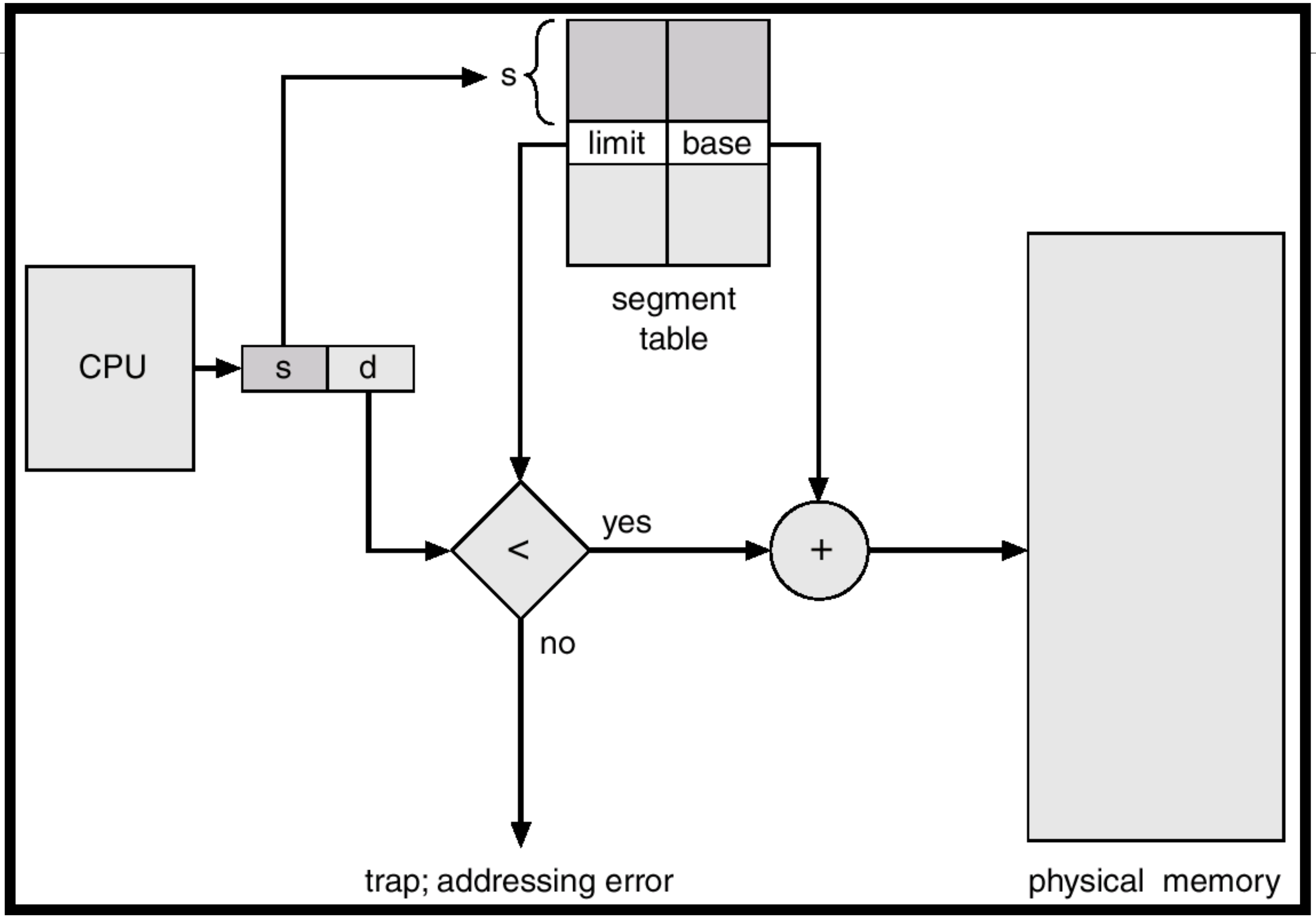
- base – contains the starting physical address where the segments reside in memory.
- *limit* – specifies the length of the segment.

Segment-table base register (STBR) points to the segment table's location in memory.

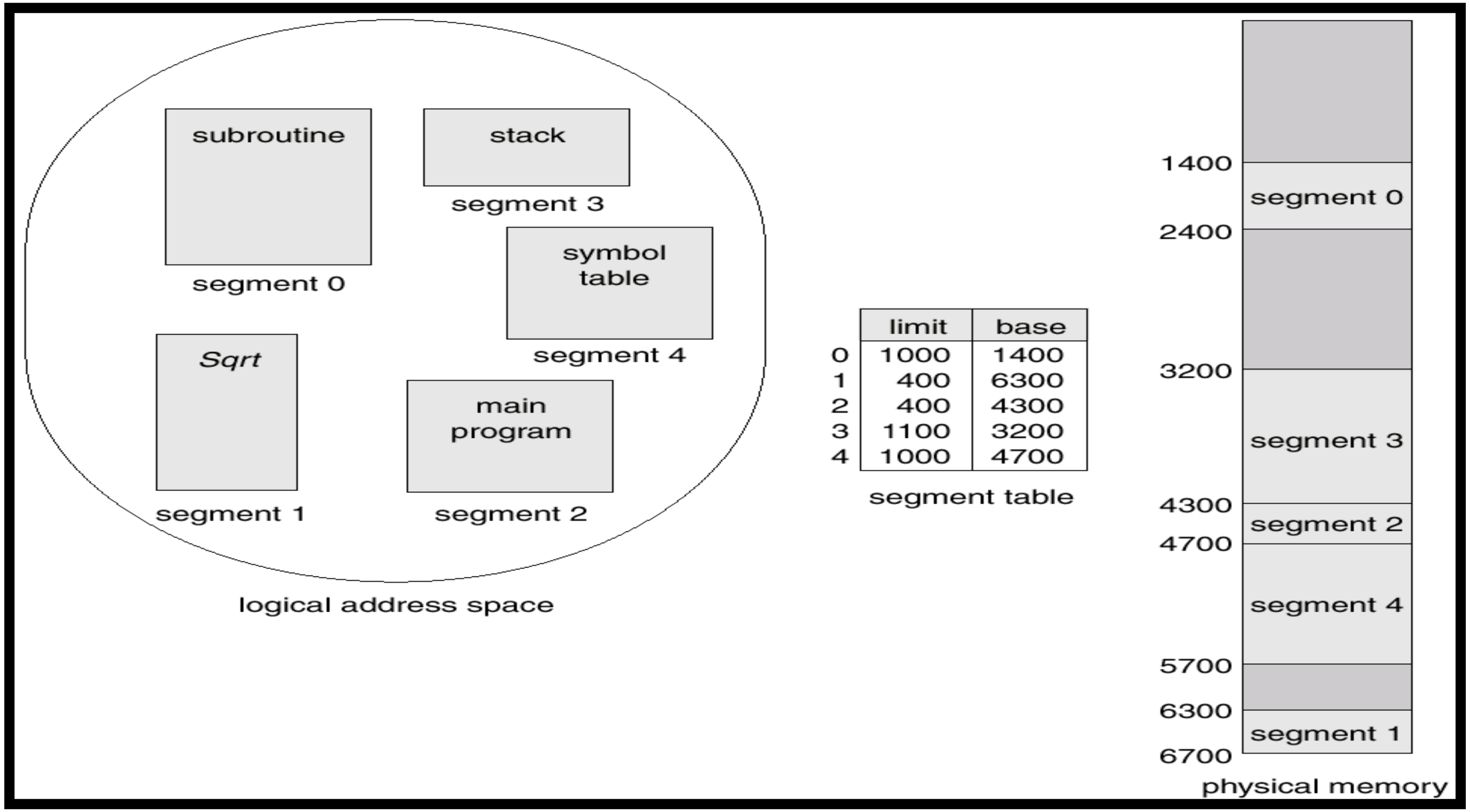
Segment-table length register (STLR) indicates number of segments used by a program;

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

Segmentation Hardware



Example of Segmentation



Example- Segmentation

Segment No	Segment Base Address	Segment Length
0	660	248
1	1752	422
2	222	198
3	996	604

For each of the following logical addresses, determine Physical address or indicate segment fault

a.0,198
b.2,156
c.1,530
d.3,444
e.0,222

Segmentation

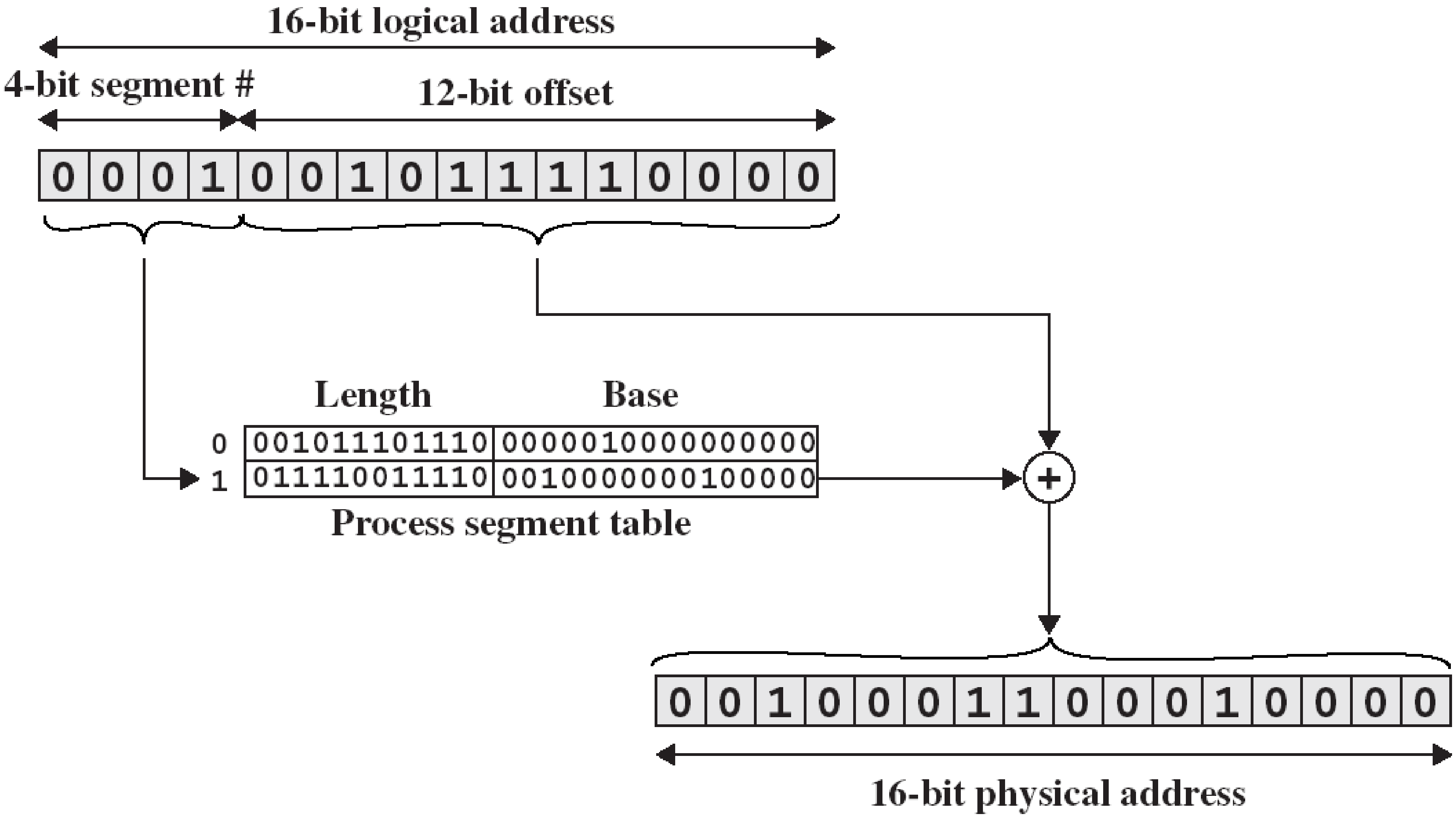
All segments of all programs do not have to be of the same length

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

Segment Translation



Paging vs Segmentation

	Paging	Segmentation
Basic	Page is fixed size block	Segment is of variable size
Fragmentation	Paging may lead to Internal Fragmentation	Segmentation may lead to External Fragmentation
Address	CPU divides the specified address into page no and offset	User specifies the address as segment no and offset
Size	Page size decided by hardware	Segment size specified by user
Table	Page table contains the base address of each table	Segment table contains segment no , base address and length

Virtual 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

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 located contiguously in main memory
- All pieces of a process do not need to be loaded in main memory during execution

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 and it is not in main memory
- Operating system places the process in a blocking state

Continued…

- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready 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

Valid-Invalid Bit

With each page table entry a valid–invalid bit is associated

(1 ⇒ in-memory (and a “legal” page),
0 ⇒ not-in-memory - legal but on disk or illegal)
… distinguish from invalid (illegal) reference -

Example of a page table snapshot.

	Frame #	valid-invalid bit
0		1
1		1
2		1
3		1
4		0
5	...	
		0
n-1		0

page table

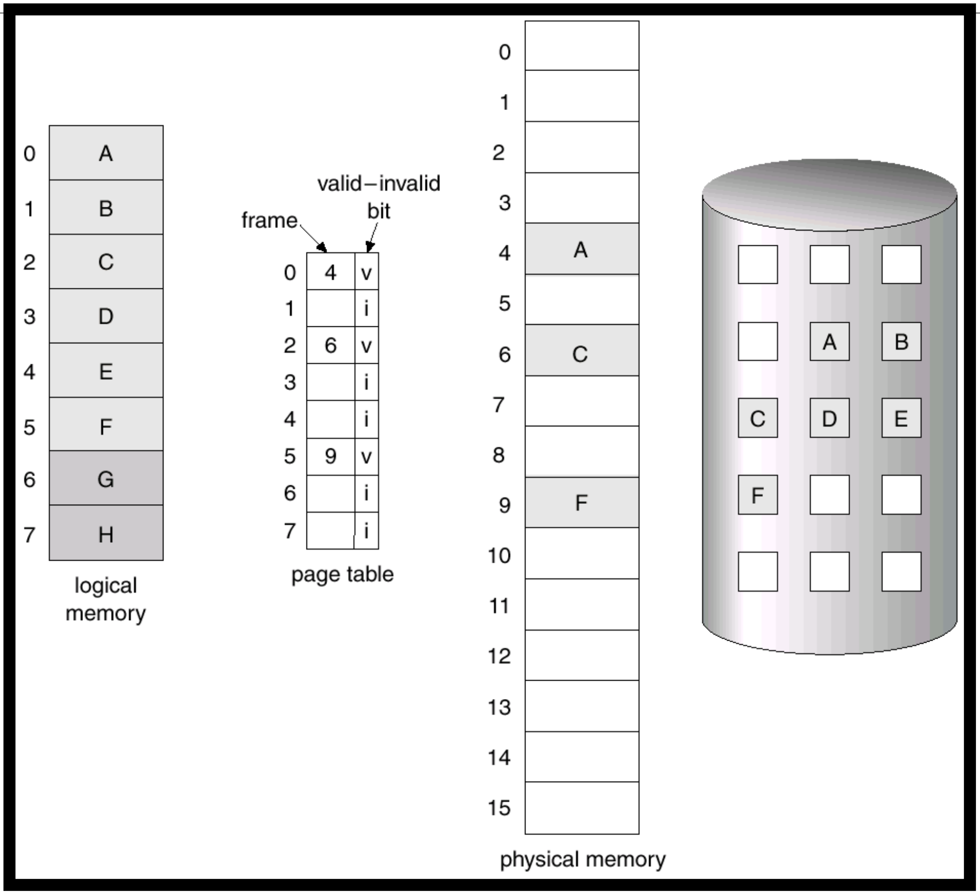
During address translation, CPU splits address in two parts, Page no and offset, page no is used as Index into Page Table. If valid–invalid bit in page table entry is 0, it means page is not in main memory ,which is page fault , or possibly illegal reference

Page Table When Some Pages Are Not in Main Memory and some ouside process space

Meaning of valid bit:
valid means page is both in memory and legal (in address space of process)

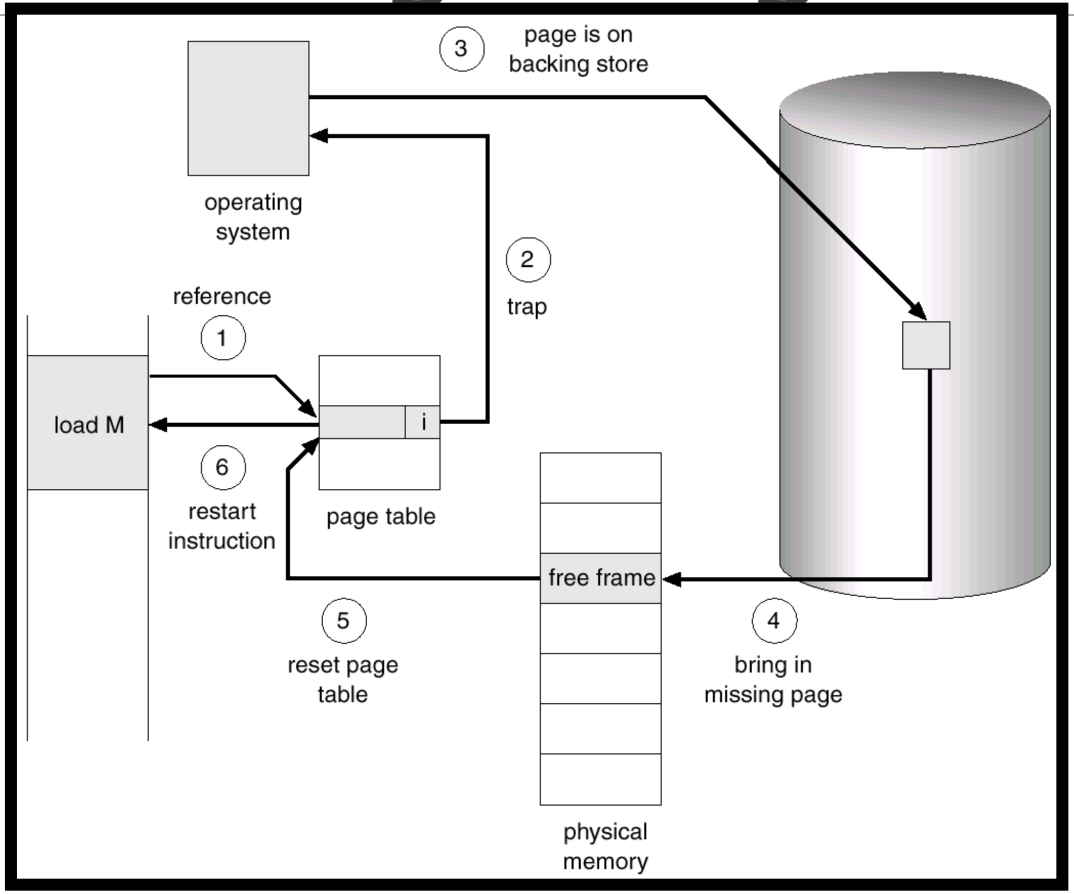
invalid means that page is either outside of the address space of the process (illegal), OR a legal address but not currently resident in memory (a page fault - most common)

NOTE: ref to pages 6 & 7 are illegal.
refs to pages 3 & 4 are page faults both cases marked invalid.



Steps in Handling a Page Fault

Page fault but No page replacement needed in this case - a free frame was found.



Page Fault: A page fault occurs when a program attempts to access data or code that is in its address space, but is not available in the RAM.

During address translation, CPU splits address in two parts, Page no and offset,

page no is used as Index into Page Table. If valid–invalid bit in page table entry is 0,

it means page is not in main memory ,which is page fault , or possibly illegal reference

For each process no. of frames are assigned. OS maintains list of free frames for

each process. When page fault occurs, page is brought in main memory. If free frame

is available page is placed in the free frame and valid bit for that page is made as 1.

But if free frame is not available, page replacement takes place. From existing pages

Page is selected for replacement.

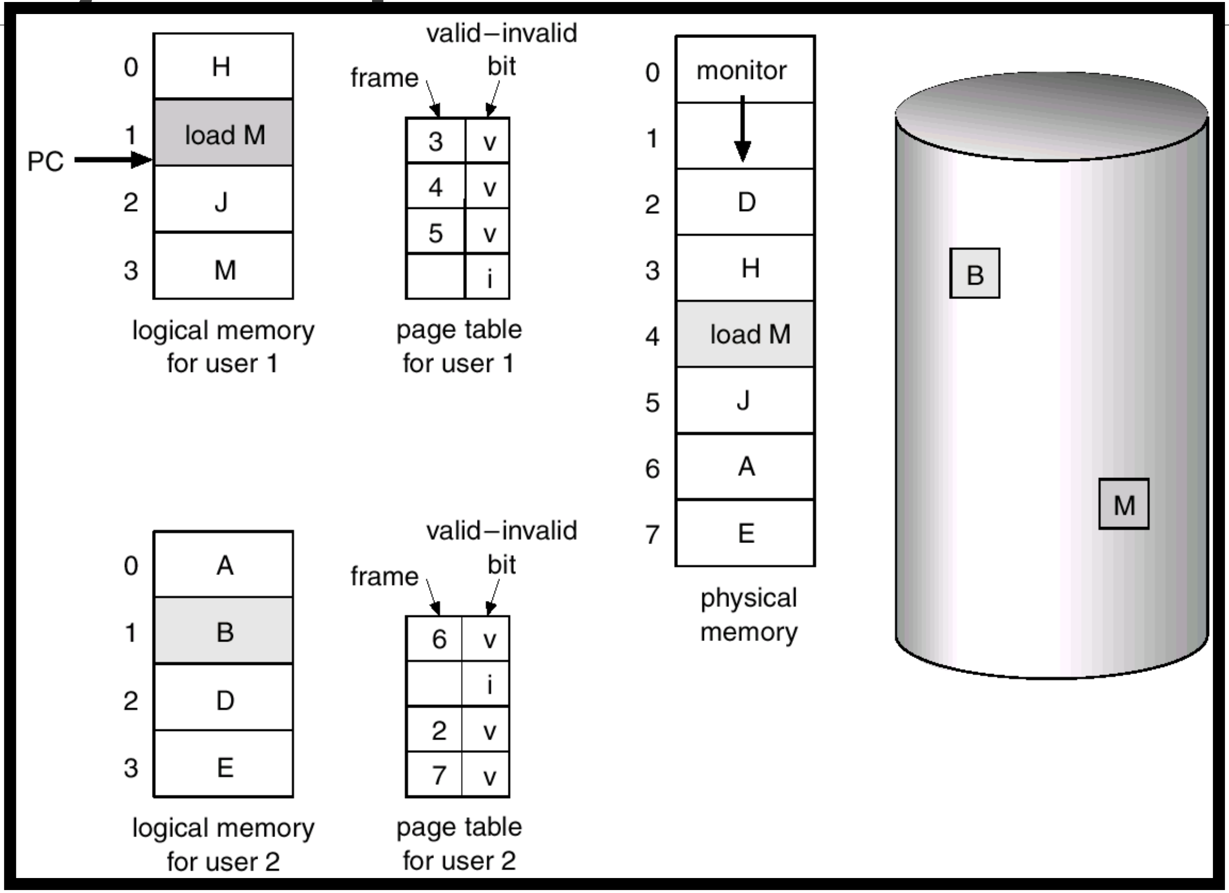
Need For Page Replacement

3 frames per process
Page 3 of user process 1 not in
Main memory
Page 1 of user process 2 not in
Main memory

User Process 2 requires page 1, invalid
Bit, not in memory.
Memory full – Free frame
List is empty … must free up
Space: page replacement

page fault ==>

B needed, but not in memory, also
Page replacement



After Page Replacement

Page 2 is selected for replacement, it was there in frame 2. After replacement in frame 2, Page 1 will reside.
Page 1 becomes valid and page 2 is invalid.

Process 1 Logical memory

0	H
1	Load M
2	J
3	M

Page table For process 1

0	3	V
1	4	V
2	5	V
3		I

Process 2 Logical memory

0	A
1	B
2	D
3	E

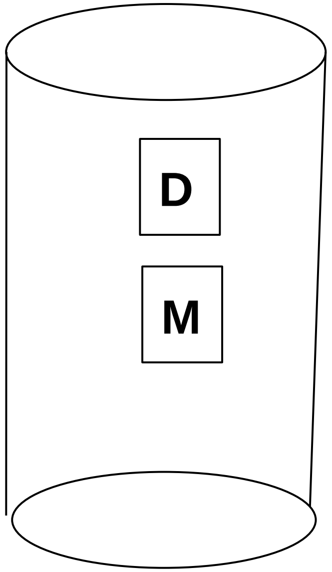
Page table For process 2

0	6	V
1	2	V
2		I
3	7	V

Physical Memory

0	Kernel
1	Kernel
2	B
3	H
4	Load M
5	J
6	A
7	E

Three frames per process



B brought in
D swapped out

Page Replacement and Modify Bit in Page Table

- If free frame is not available, select the page to be swapped out.
- Modify bit in page table indicates 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

Basic Page fault handling

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 … **this is the likelihood**.
3. Read the desired page into the frame. Update the page and frame tables.
   If the “victim” page frame is modified it will have to paged out to the disk.
4. Restart the process **(process was blocked during page fault processing).**

Fetch Policy

- Determines when a page should be brought into memory
- Demand paging only brings pages into main memory when a reference is made to a location on the page
  - Many page faults when process first started
- Prepaging brings in more pages than needed
  - More efficient to bring in pages that reside contiguously on the disk
  - Ineffective as most of the extra pages that are brought in are not referenced.

Replacement Policy

- Frame Locking
  - Associate a lock bit with each frame
  - If frame is locked, it may not be replaced
  - Example:
    - Kernel of the operating system
    - Control structures
    - I/O buffers
    - Time-critical main memory frames

Thrashing

- Swapping out a piece of a process just before that piece is needed
- The processor spends most of its time swapping pieces rather than executing user instructions
- A process is busy swapping pages in and out most of the time **- very little time spent on productive work - most time spent doing paging.**

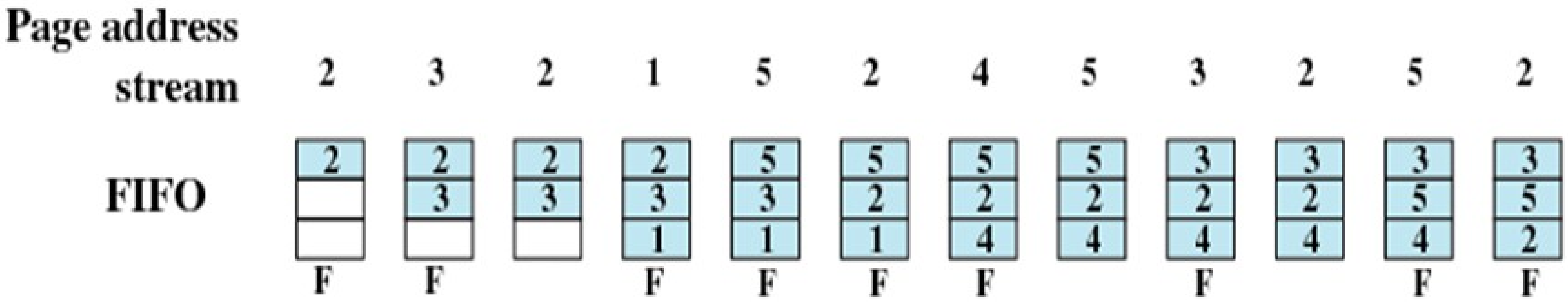
Basic Replacement Algorithms

1. First In First Out (FIFO)
2. Optimal Policy
3. Least Recently Used (LRU)

First-in, first-out (FIFO)

- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
- Simplest replacement policy to implement
- Page that has been in memory the longest is replaced
- These pages may be needed again very soon

Example: FIFO, no of frames 3

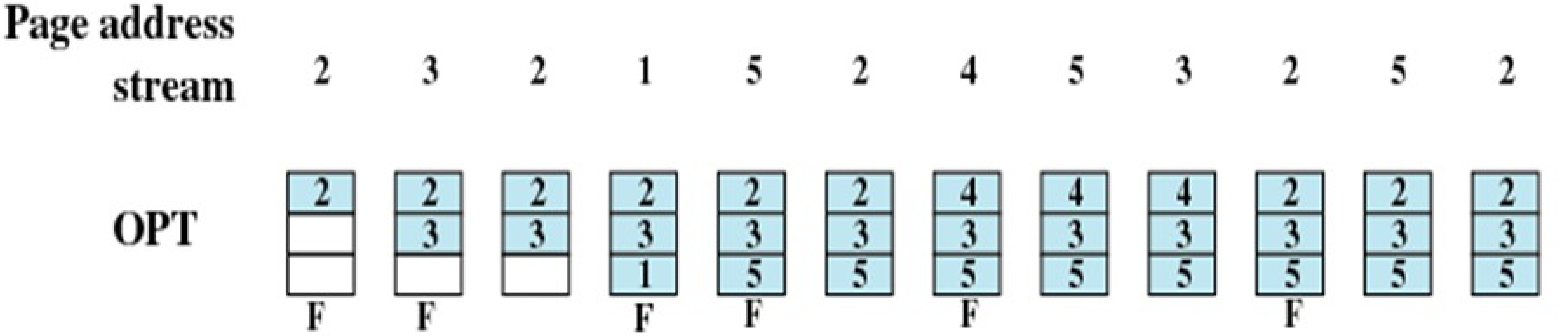


Total page faults = 9

Optimal policy

- Selects for replacement that page for which the time to the next reference is the longest
- Impossible to have perfect knowledge of future events

Example: Optimal policy, no of frames 3



Total page faults = 6

Least Recently Used (LRU)

- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Each page could be tagged with the time of last reference. This would require a great deal of overhead

Example: LRU, no of frames 3

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
	F	F		F	F		F		F	F		

Total page faults = 7

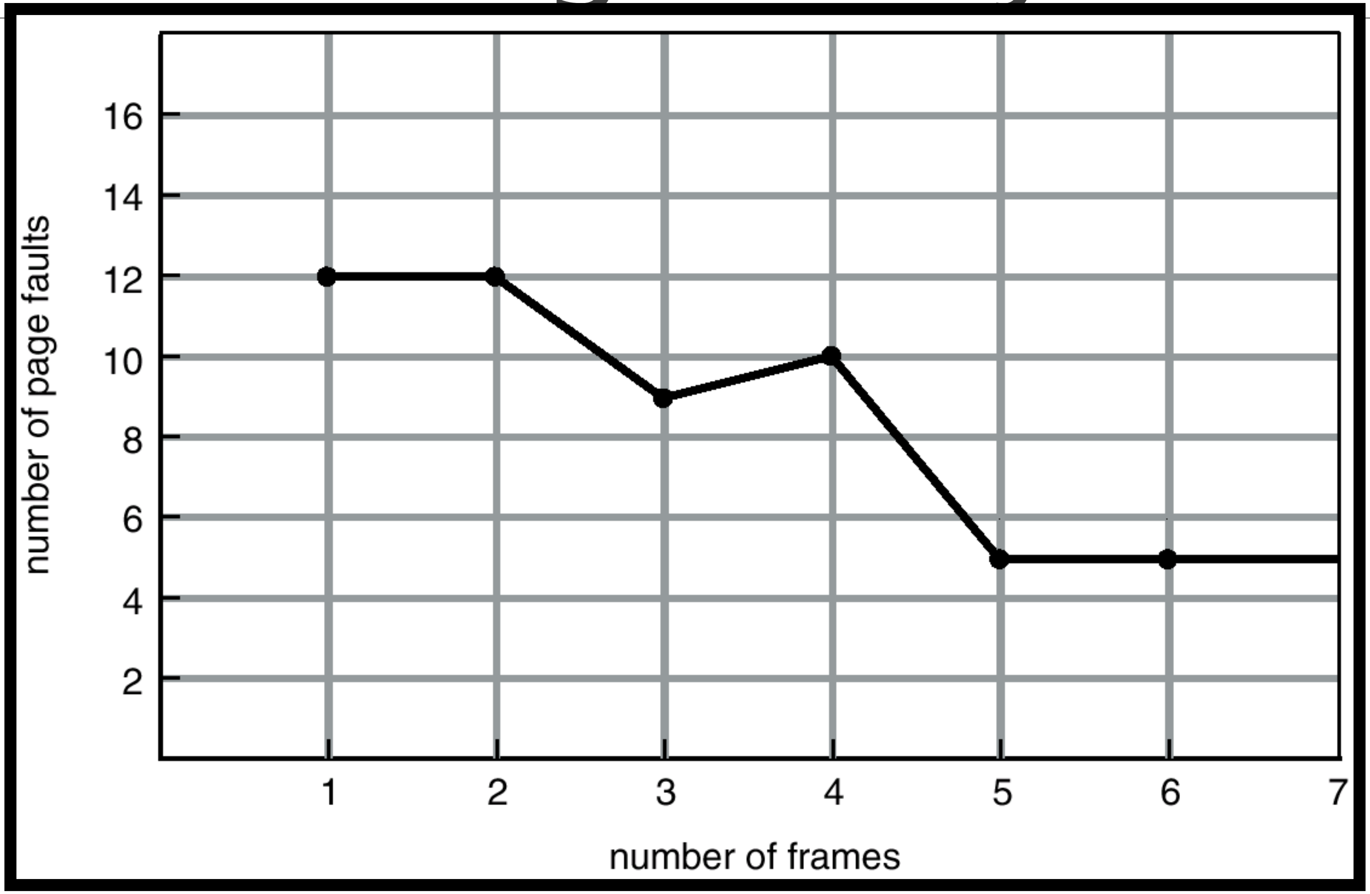
Belady’s Anomaly

Increasing number of frames should decrease number of faults.

In some algorithms it is observed that if number of frames are increased, page faults are increased.

This is called as Belady’s anomaly.

FIFO Illustrating Belady's Anamoly



Translation Lookaside Buffer

Each virtual memory reference can cause two physical memory accesses

- One to fetch the page table
- One to fetch the data

To overcome this problem a high-speed cache is set up for page table entries

- Called a Translation Lookaside Buffer (TLB)
- Contains page table entries that have been most recently used

Translation Lookaside Buffer

Given a virtual address, processor examines the TLB

If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed

If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table

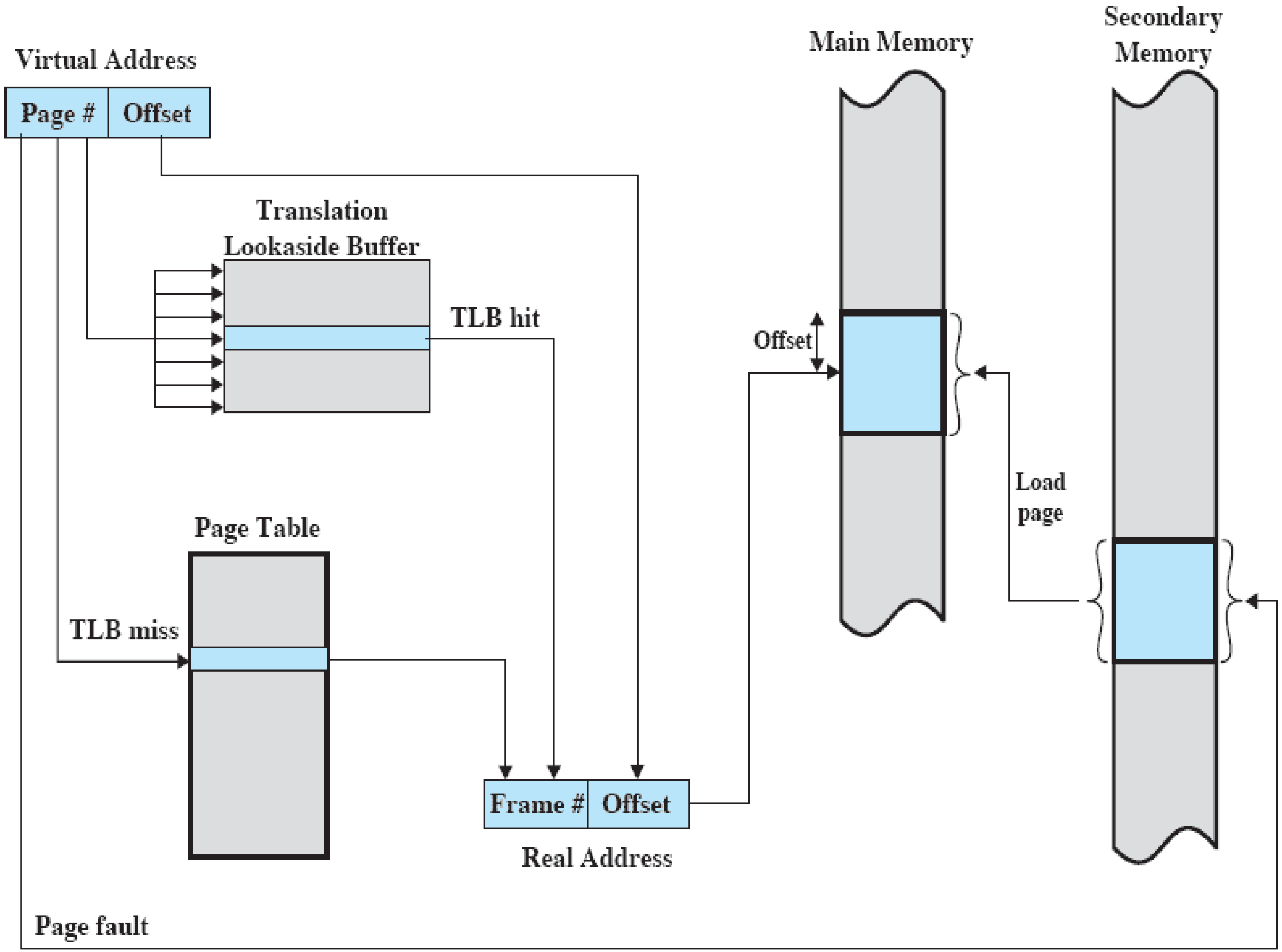
Translation Lookaside Buffer

First 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

Translation Lookaside Buffer



Translation Lookaside Buffer

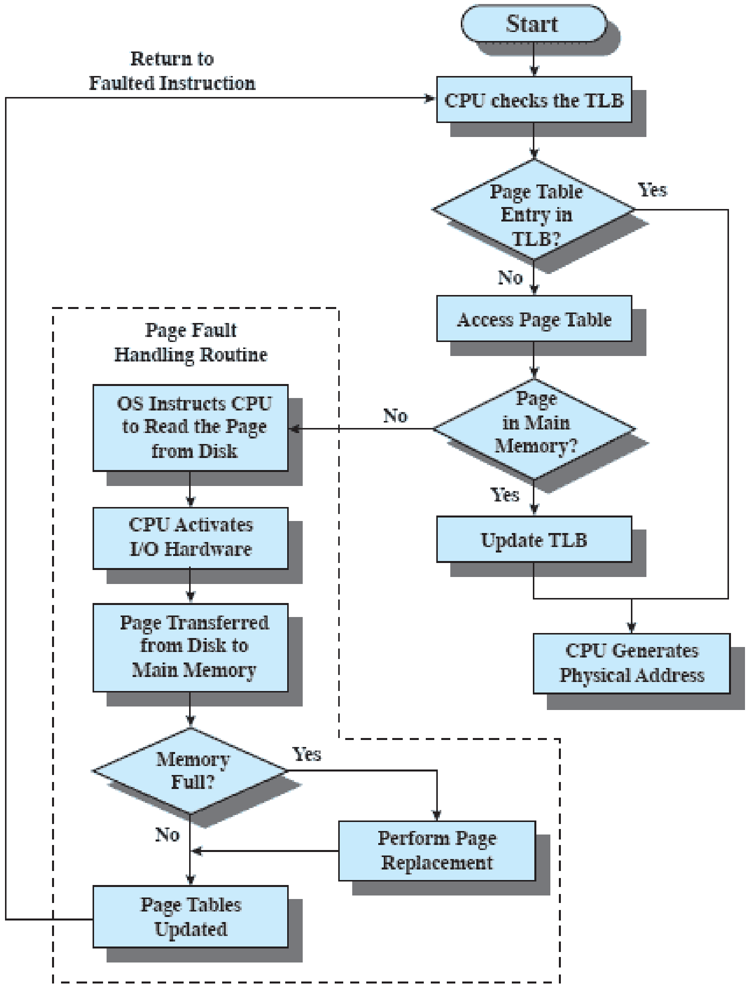


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Translation Lookaside Buffer

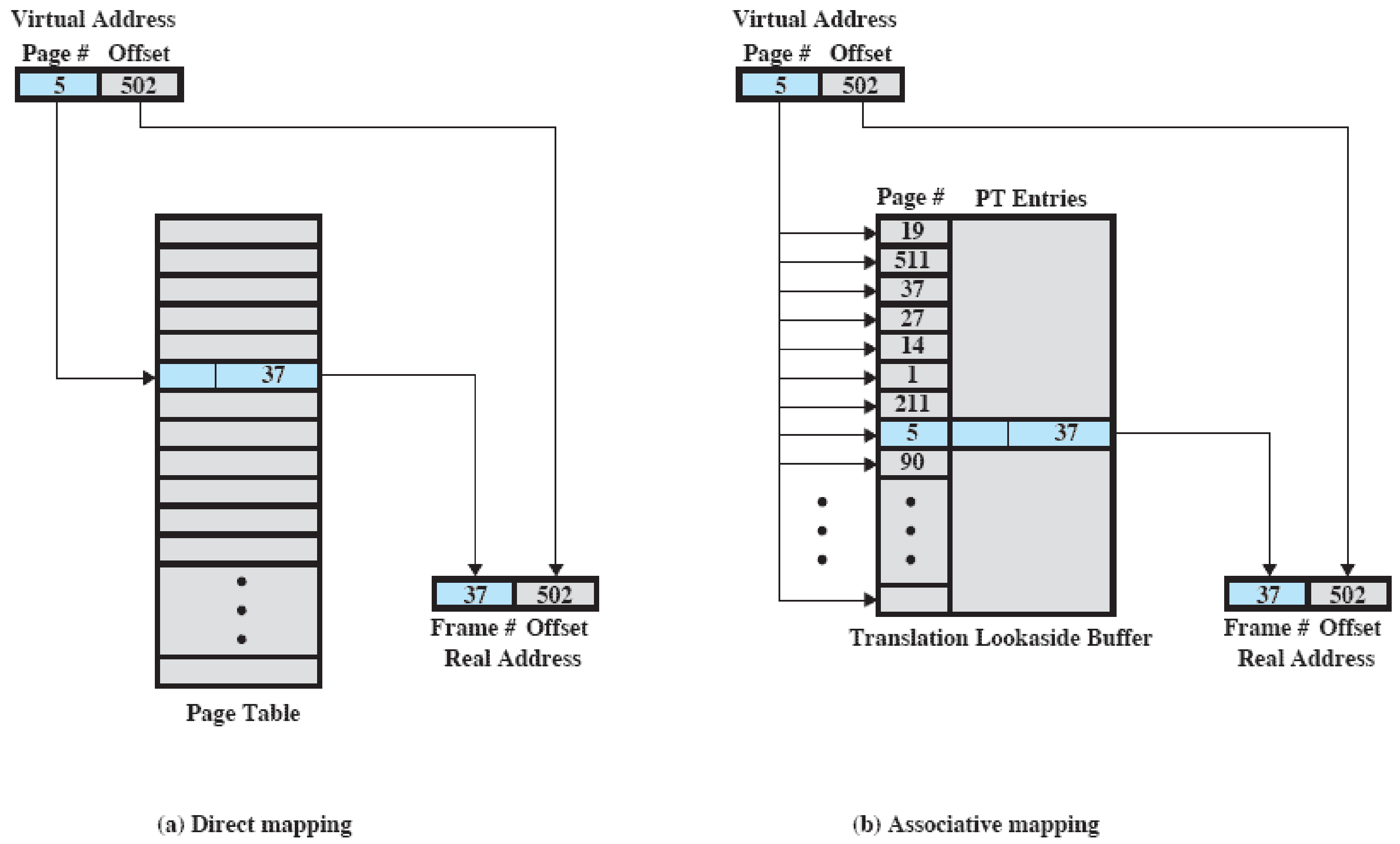


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

END