Unit-5
Memory Management

Outline

1. Basic requirements of Memory Management
2. Memory Partitioning
3. Basic blocks of Memory Management
    1. Paging
    2. Segmentation

The need for memory management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never end to it.
- Memory Management, involves swapping blocks of data from secondary storage.
- Memory I/O is slow compared to a CPU
  - The OS must cleverly time the swapping to maximise the CPU's efficiency

Memory Management

Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

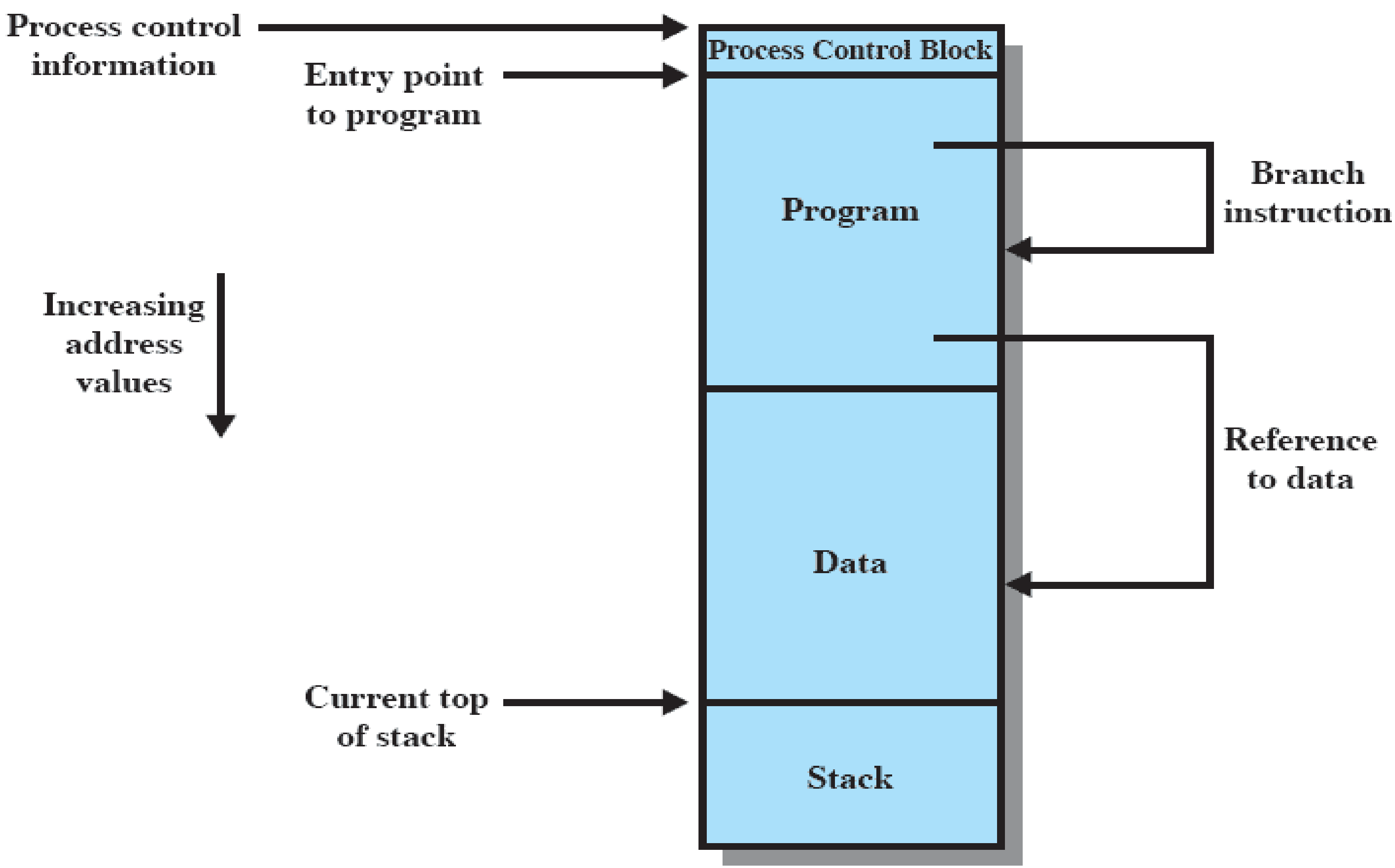
1. Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical organisation
- Physical organisation

Requirements: Relocation

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

Addressing



Process control information
Entry point to program
Process Control Block
Program
Branch instruction
Increasing address values
Reference to data
Data
Current top of stack
Stack

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

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

Requirements: Logical Organization

- Memory is organized linearly (usually)
- 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 helps here

Requirements: Physical Organization

- Cannot leave the programmer with the responsibility to manage memory
- Memory available for a program plus its data may be insufficient
  - **Overlaying (Overlapping)** allows various modules to be assigned the same region of memory but is time consuming to program
- Programmer does not know how much space will be available

2. Memory Partitioning

A. Fixed Partitioning
B. Dynamic Partitioning
C. Simple Paging
D. Simple Segmentation
E. Virtual Memory Paging
F. Virtual Memory Segmentation

A. Fixed Partitioning

- Equal-size partitions Any process whose size is less than or equal to the partition size can be loaded into an available partition
- The operating system can swap a process out of a partition
  - If none are in a ready or running state

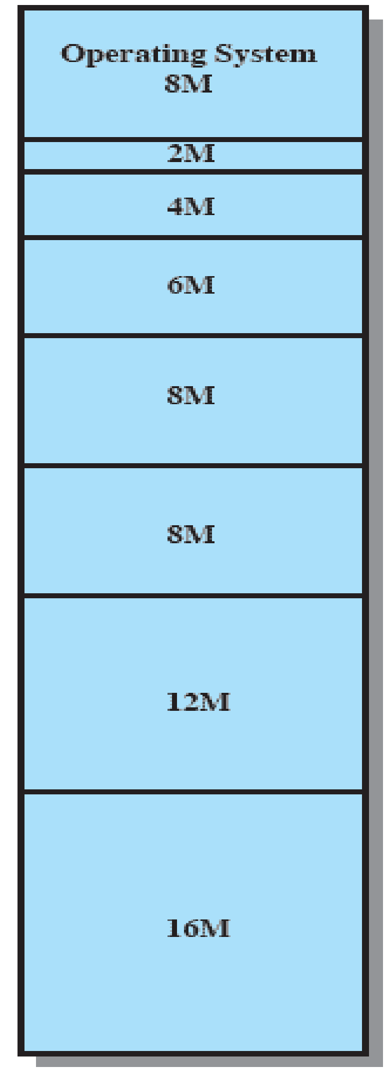
(a) Equal-size partitions

Fixed Partitioning Problems

- A program may not fit in a partition.
  - The programmer must design the program with overlays
- Main memory use is inefficient.
  - Any program, no matter how small, occupies an entire partition.
  - This is results in ***internal fragmentation.***

Solution – Unequal Size Partitions

- But doesn't solve completely
- Programs up to 16M can be accommodated without overlay
- Smaller programs can be placed in smaller partitions, reducing internal fragmentation

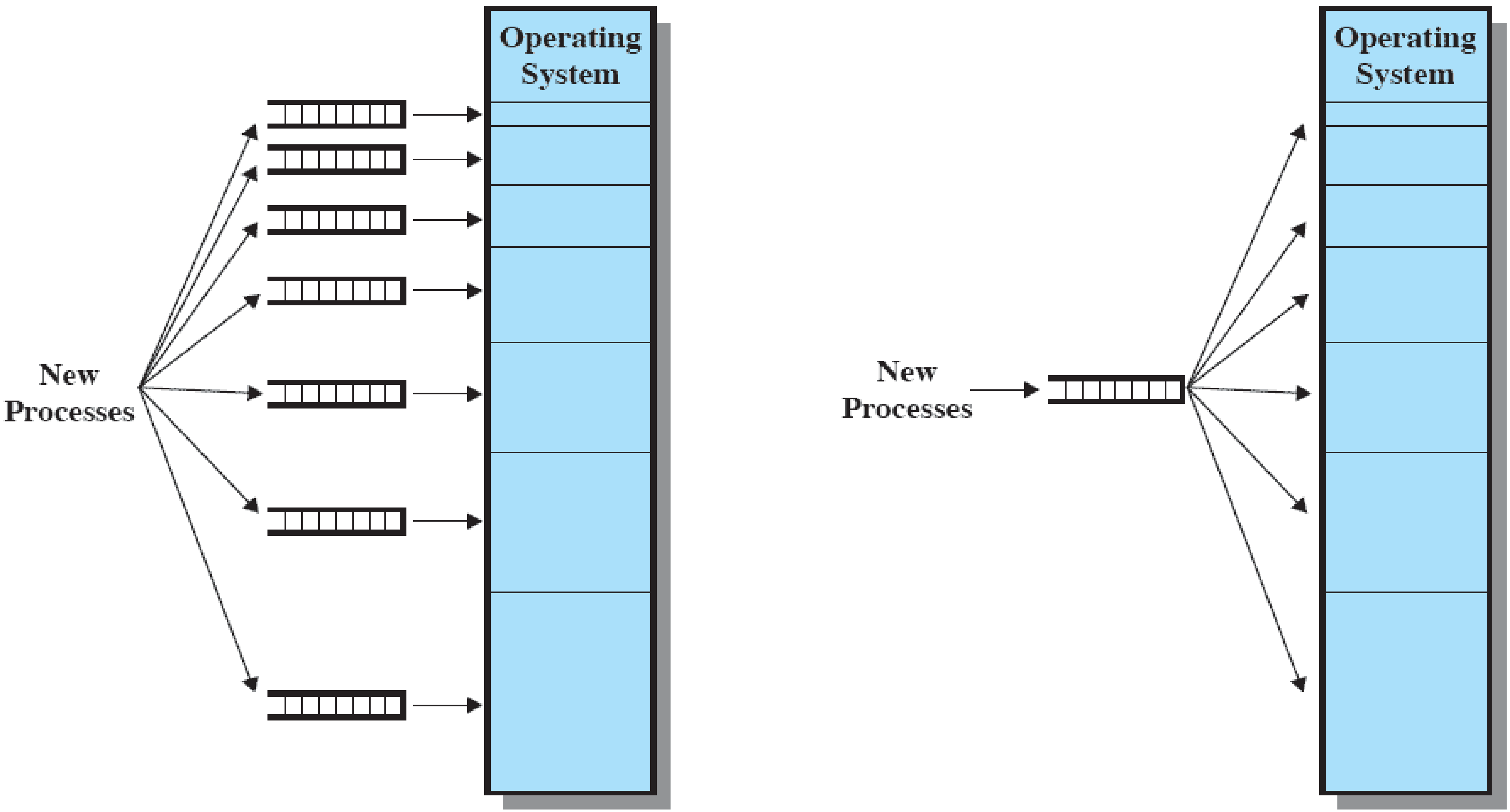


(b) Unequal-size partitions

Placement Algorithm

- Equal-size
  - Placement is trivial (no options)
- Unequal-size
  - 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

Fixed Partitioning



(a) One process queue per partition

(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning

Remaining Problems with Fixed Partitions

- The number of active processes is limited by the system
  - limited by the pre-determined number of partitions
- A large number of very small process will not use the space efficiently
  - In either fixed or variable length partition methods

B. Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required

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

Dynamic Partitioning

- Operating system must decide which free block to allocate to a process
- 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
  - Worst performer overall

Dynamic Partitioning

- First-fit algorithm
  - Scans memory form 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

Dynamic Partitioning

- Next-fit
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

Allocation

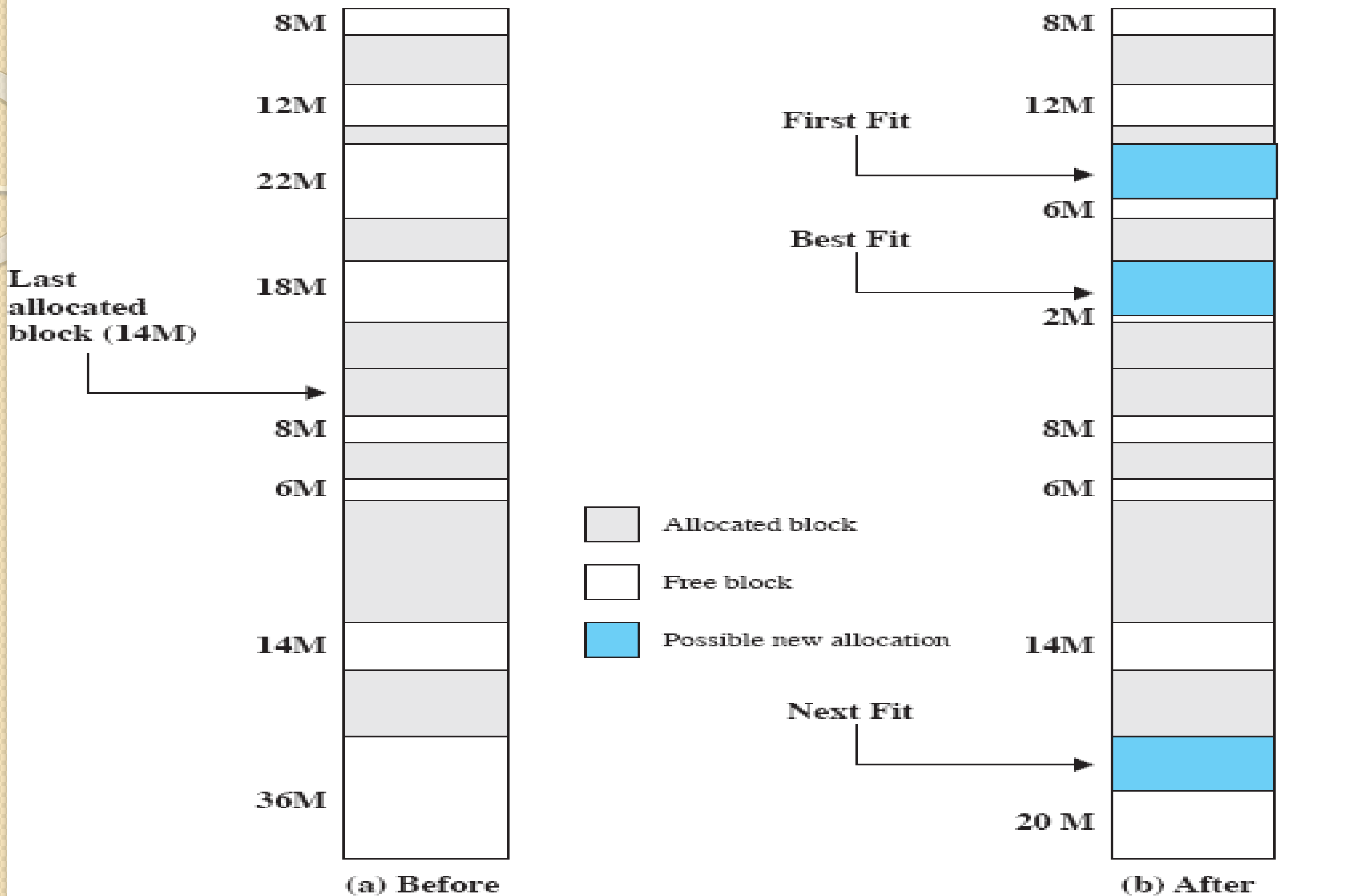


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

- Entire space available is treated as a single block of 2^N
- If a request of size s where $2^{N-1} < s <= 2^N$
  - 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

Example of Buddy System

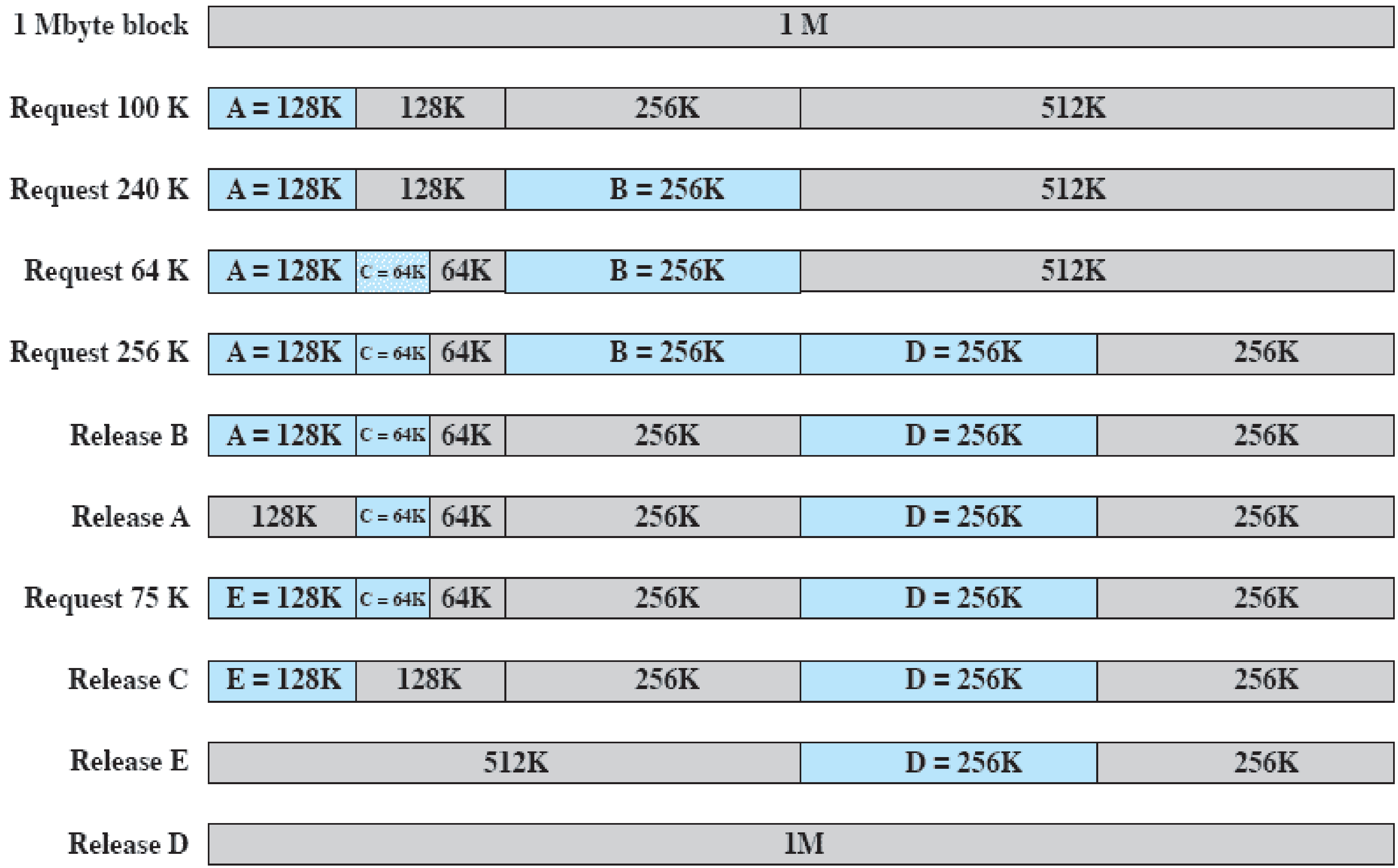


Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

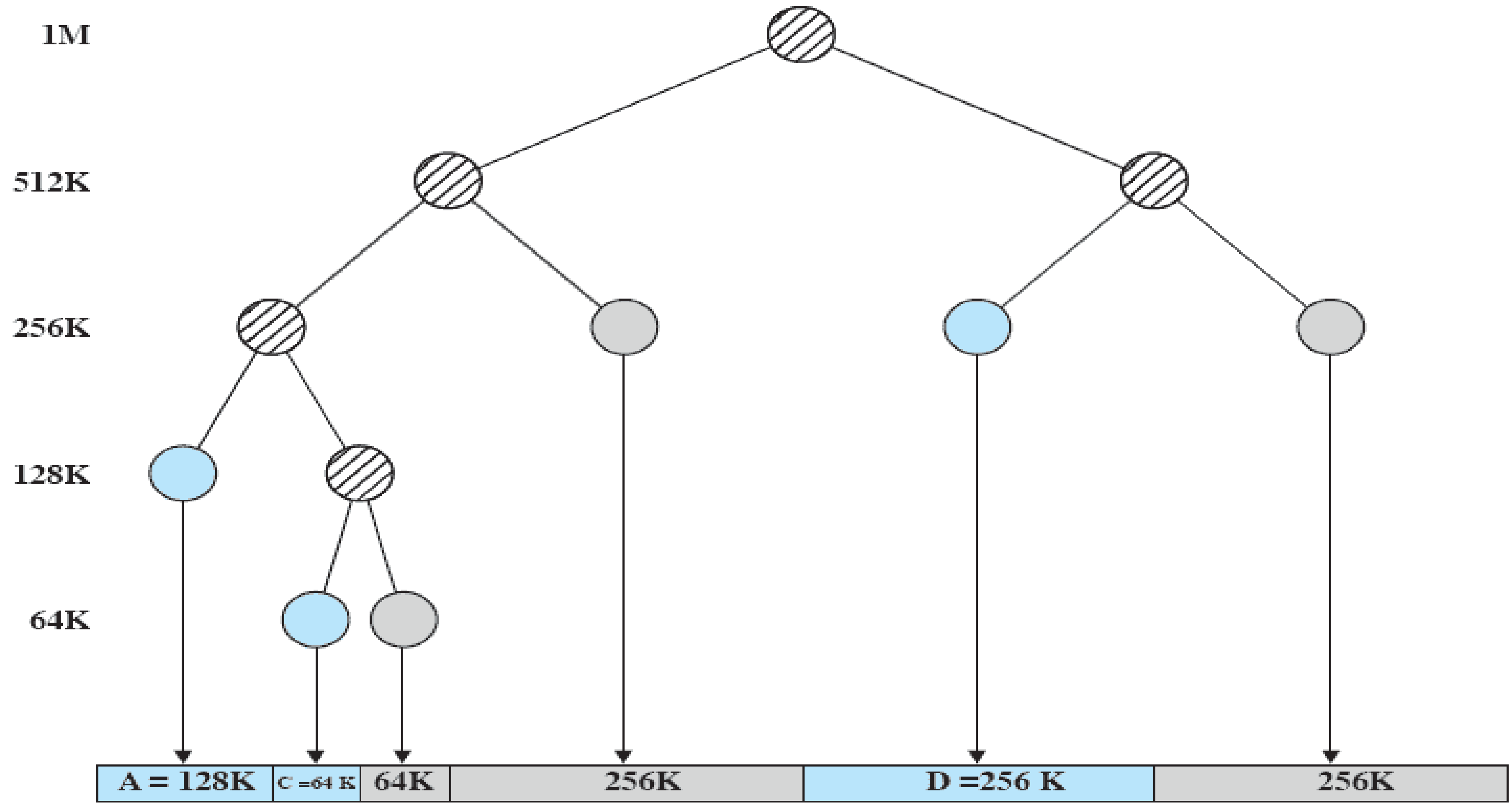


Figure 7.7 Tree Representation of Buddy System

Relocation

- When program loaded into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions which means different absolute memory locations during execution
  - Swapping
  - Compaction

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory.
- Relative
  - Address expressed as a location relative to some known point.
- Physical or Absolute
  - The absolute address or actual location in main memory.

Relocation

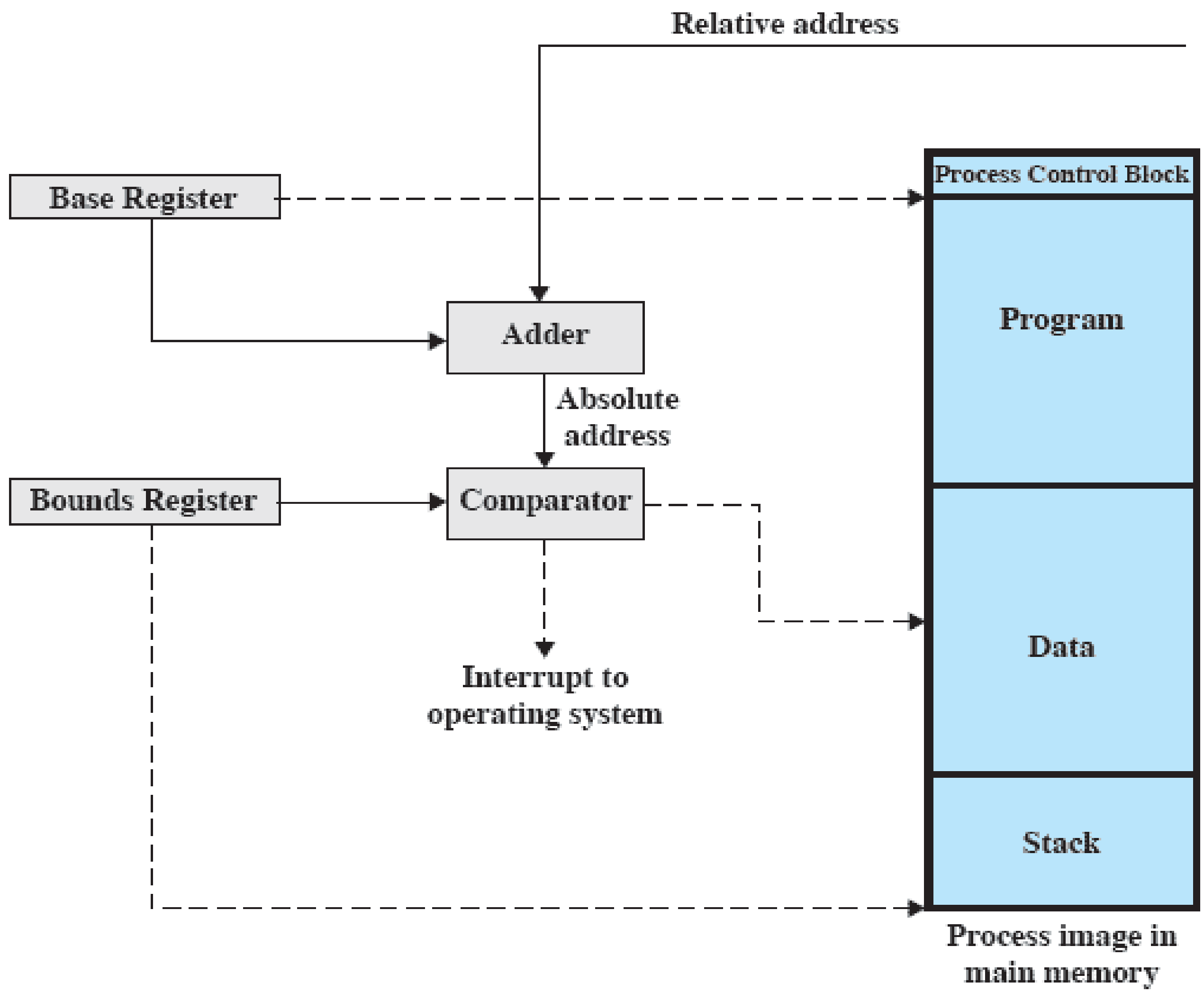


Figure 7.8 Hardware Support for Relocation

Registers Used during Execution

- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded or when the process is swapped in

Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

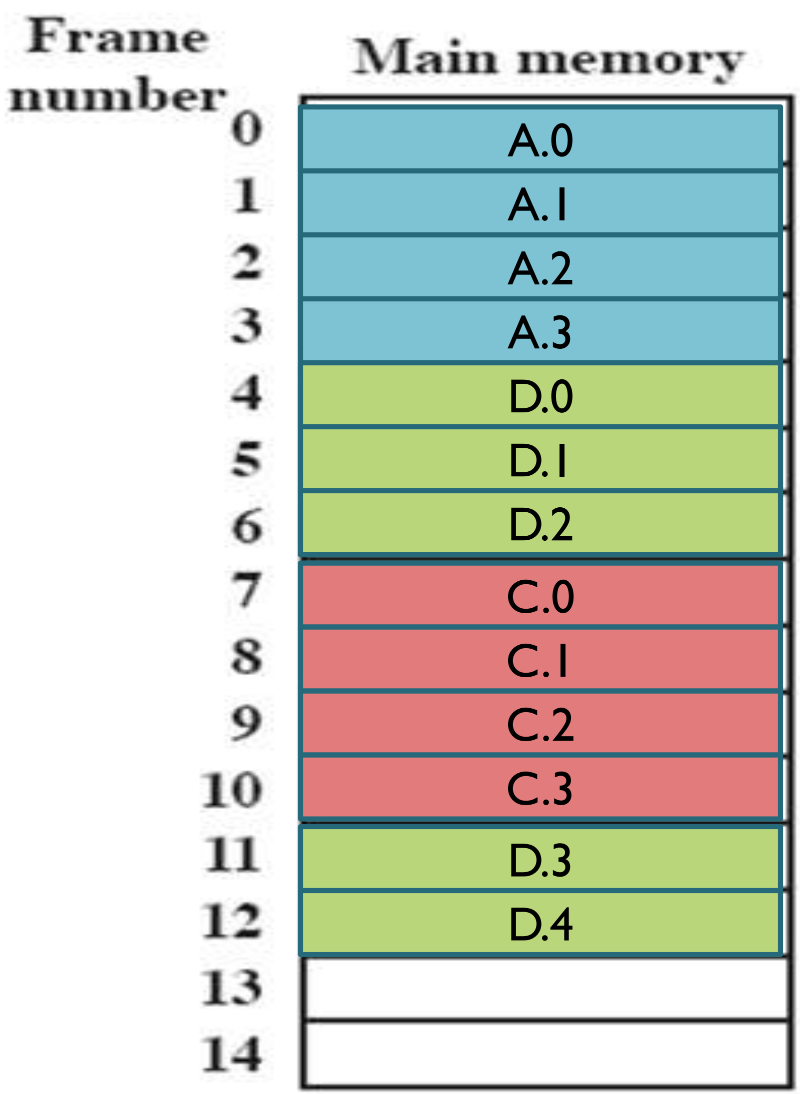
Paging

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called ***pages***
- The chunks of memory are called ***frames***

Paging

- 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

Processes and Frames



Page Table

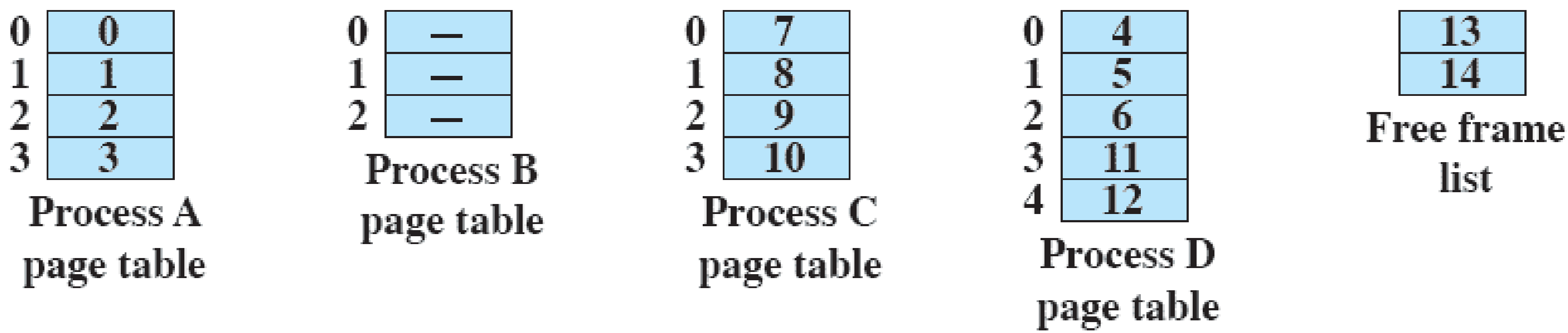


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Segmentation

- A program can be subdivided into segments
  - Segments may vary in length
  - There is a maximum segment length
- Addressing consist of two parts
  - a segment number and
  - an offset
- Segmentation is similar to dynamic partitioning

Logical Addresses

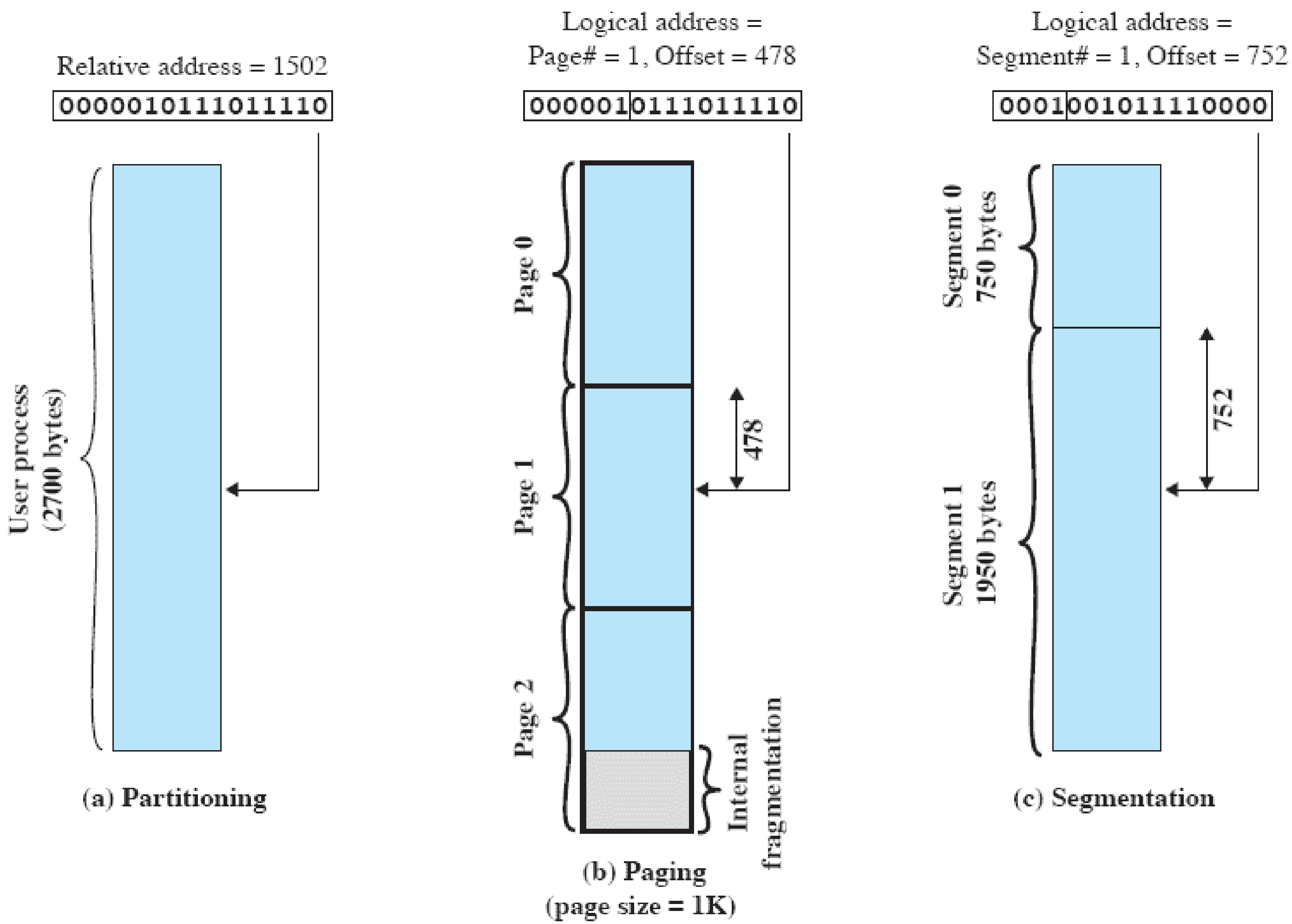
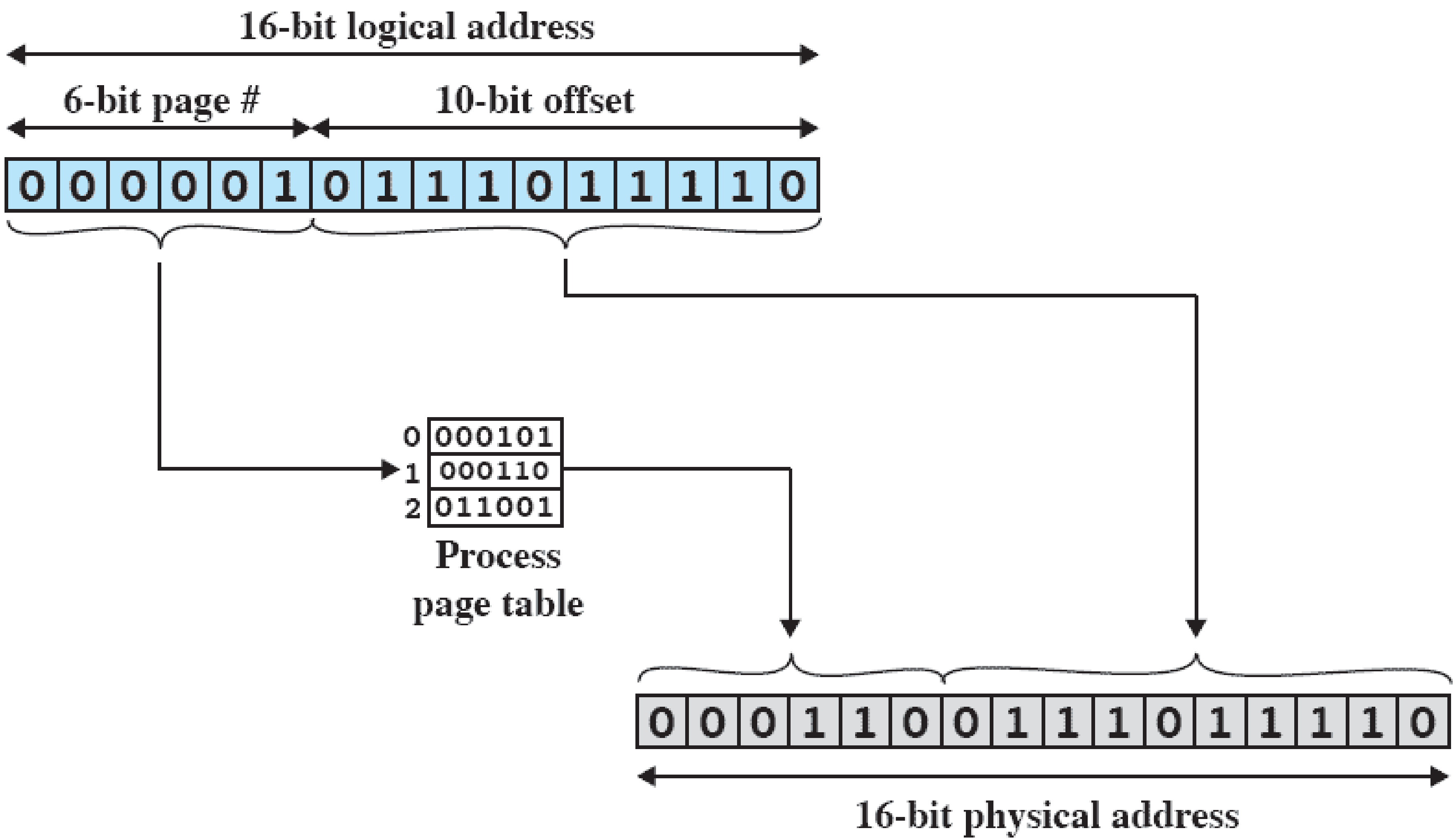


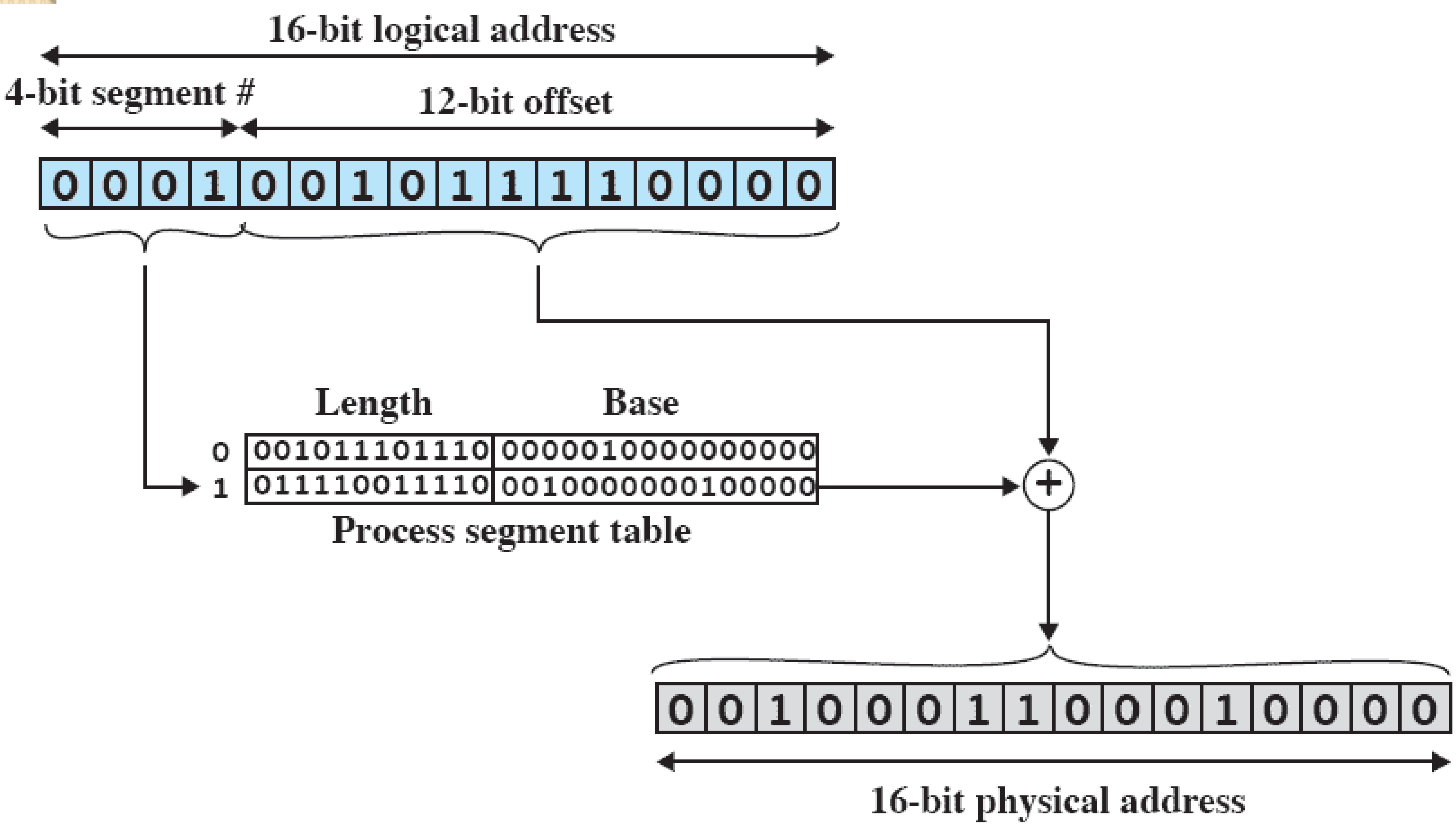
Figure 7.11 Logical Addresses

Paging



(a) Paging

Segmentation



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation

Virtual Memory Management
and
Page Replacement Algorithms

Table 8.1 Virtual Memory Terminology

Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory.

Key points in Memory Management

1) Memory references are logical addresses dynamically translated into physical addresses at run time
   - A process may be swapped in and out of main memory occupying different regions at different times during execution
2) A process may be broken up into pieces that do not need to located contiguously in main memory

Breakthrough in Memory Management

- **If both** of those two characteristics are present,
  - then it is not necessary that all of the pages or all of the segments of a process be in main memory during execution.
- If the next instruction, and the next data location are in memory then execution can proceed
  - at least for a time

Execution of a Process

- 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

Execution of a Process

- 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

Implications of this new strategy

- 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

Real and Virtual Memory

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

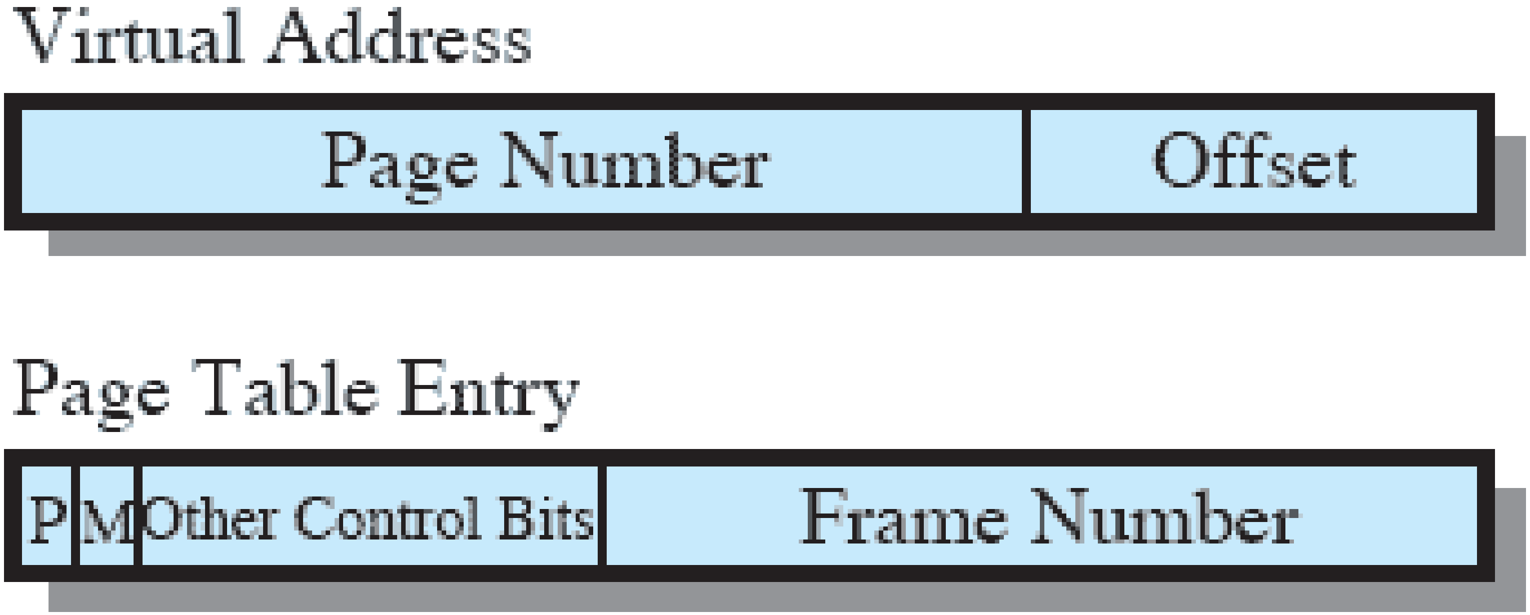
Support Needed for Virtual Memory

- Hardware must support paging and segmentation
- Operating system must be able to manage the movement of pages and/or segments between secondary memory and main memory

Paging

- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- Two extra bits are needed to indicate:
  - whether the page is in main memory or not
  - Whether the contents of the page has been altered since it was last loaded

Paging Table



(a) Paging only

Address Translation

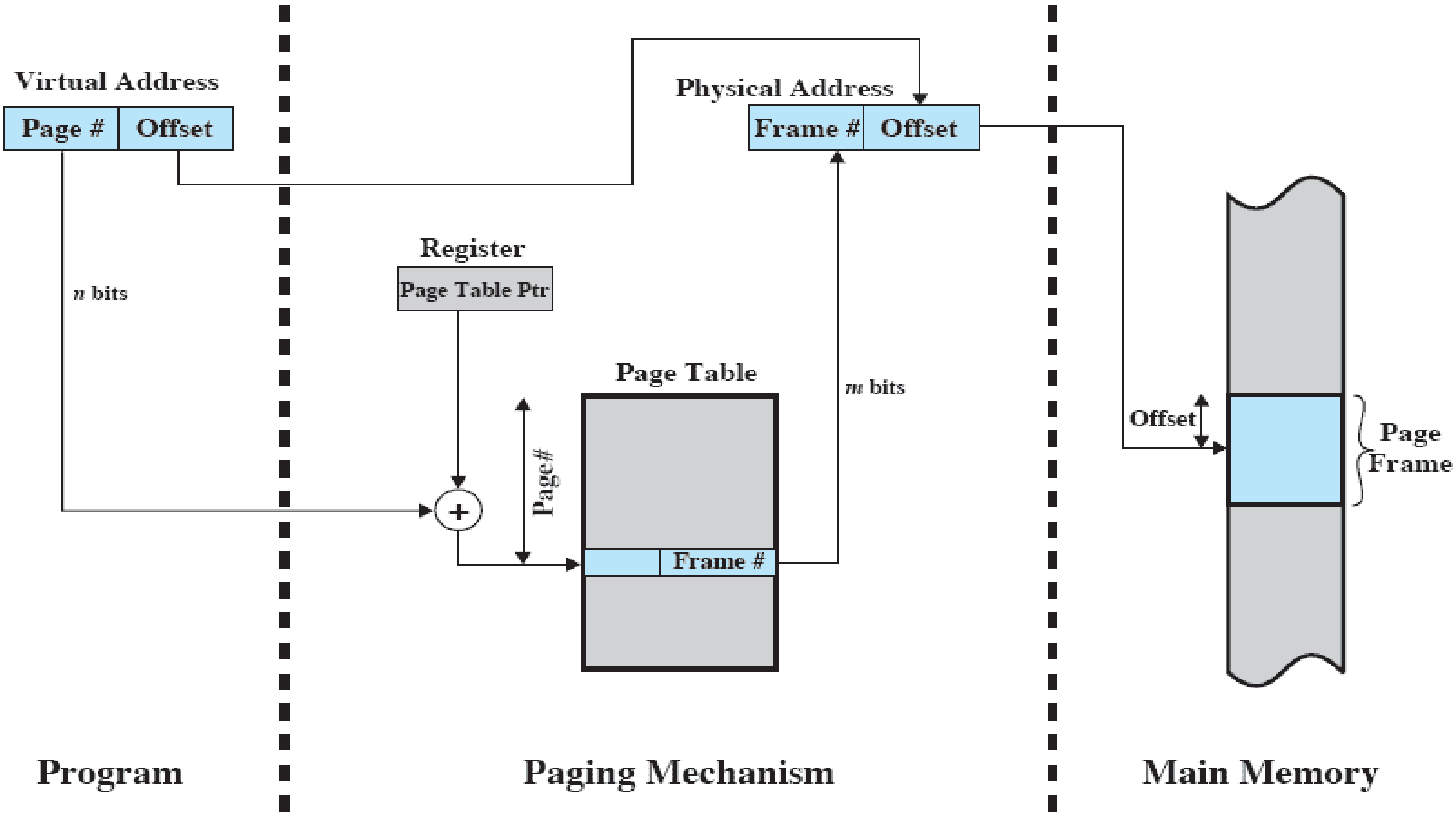
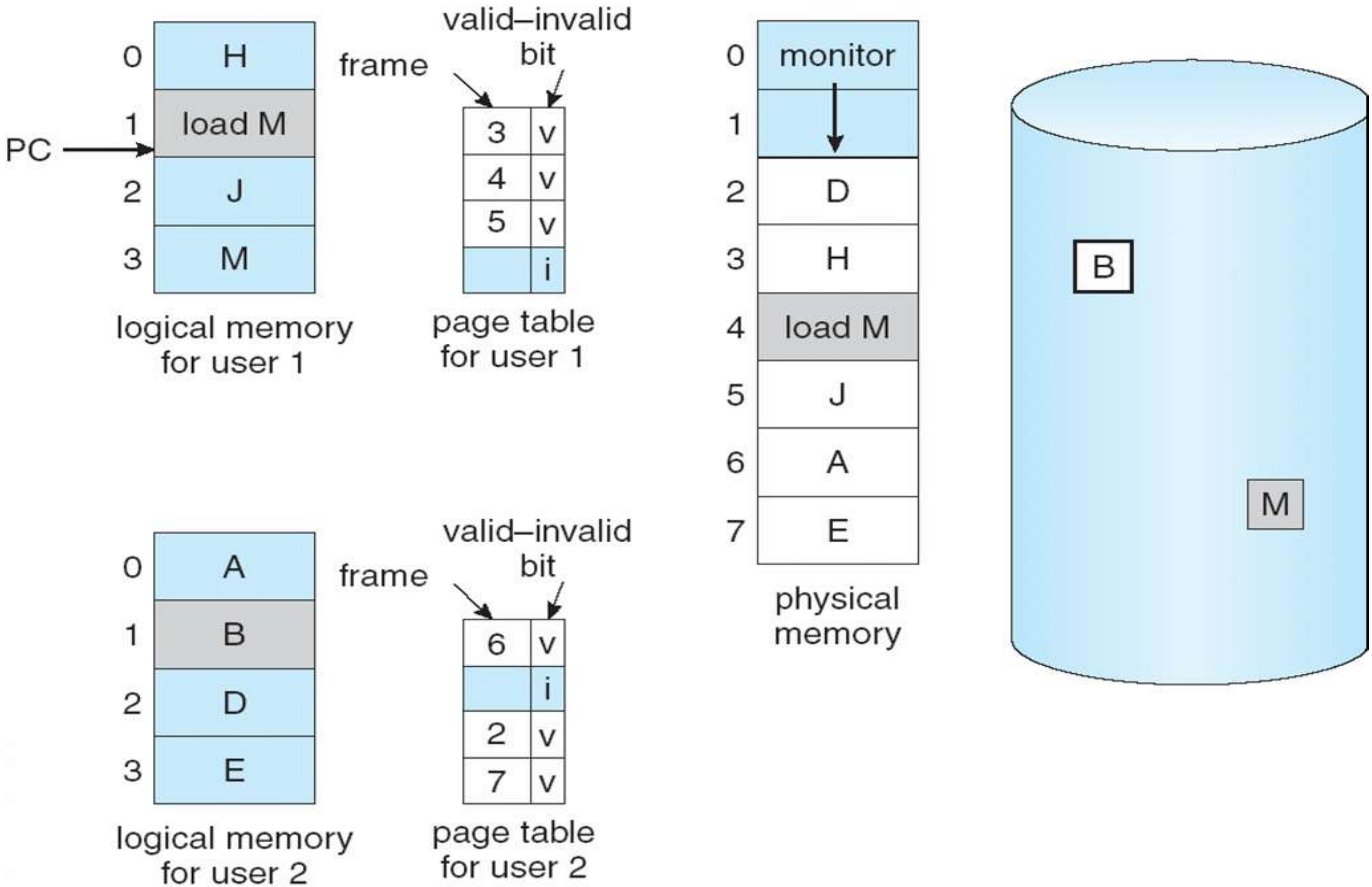


Figure 8.3 Address Translation in a Paging System

Replacement Policy

- When all of the frames in main memory are occupied and it is necessary to bring in a new page, the replacement policy determines which page currently in memory is to be replaced.

Need For Page Replacement



Basic Replacement Algorithms

- There are certain basic algorithms that are used for the selection of a page to replace, they include
  - Optimal
  - Least recently used (LRU)
  - First-in-first-out (FIFO)
- Examples

Examples

- An example of the implementation of these policies will use a page address stream formed by executing the program is
  - 2 3 2 1 5 2 4 5 3 2 5 2
- Which means that the first page referenced is 2,
  - the second page referenced is 3,
  - And so on.

Optimal policy

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

Optimal Policy Example

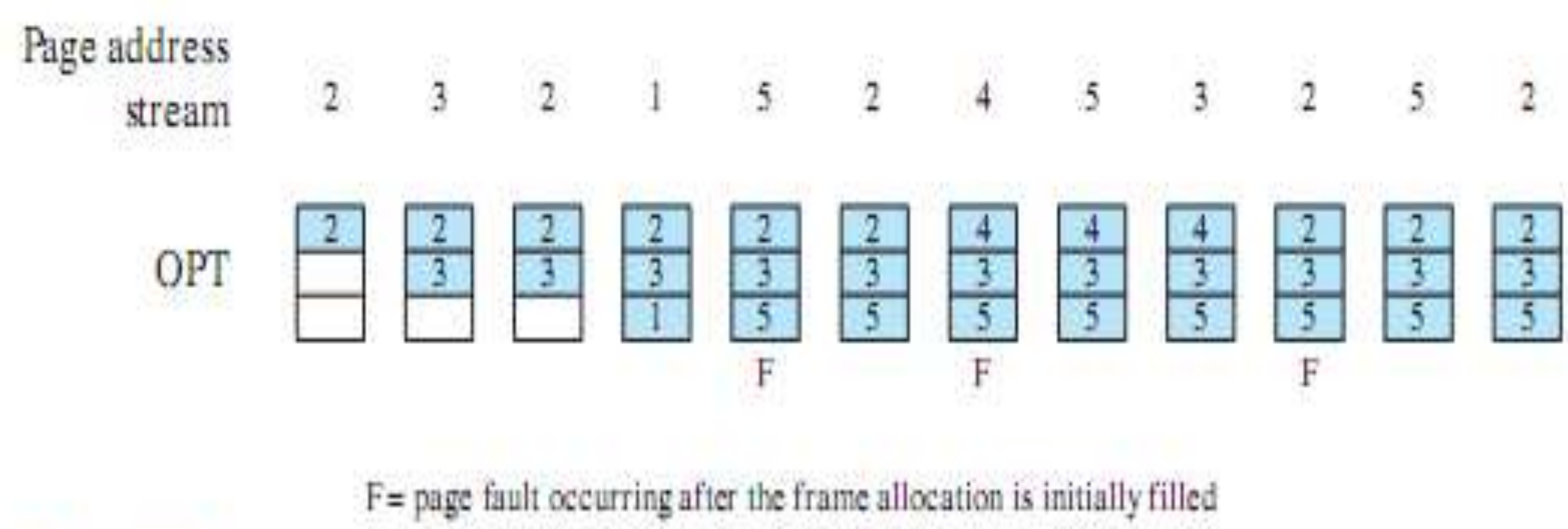


Figure 8.15 **Behavior of Four Page Replacement Algorithms**

- The optimal policy produces three page faults after the frame allocation has been filled.

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
- Difficult to implement
  - One approach is to tag each page with the time of last reference.
  - This requires a great deal of overhead.

LRU Example

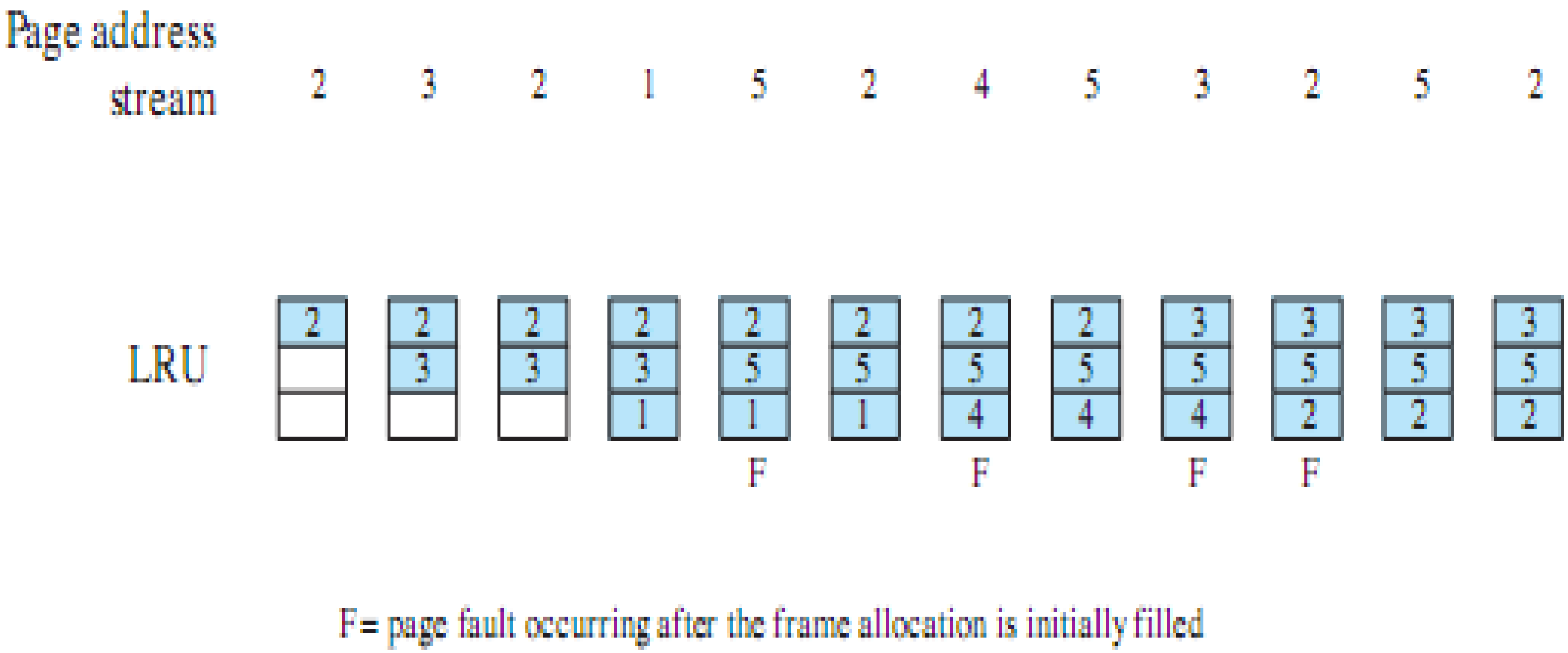


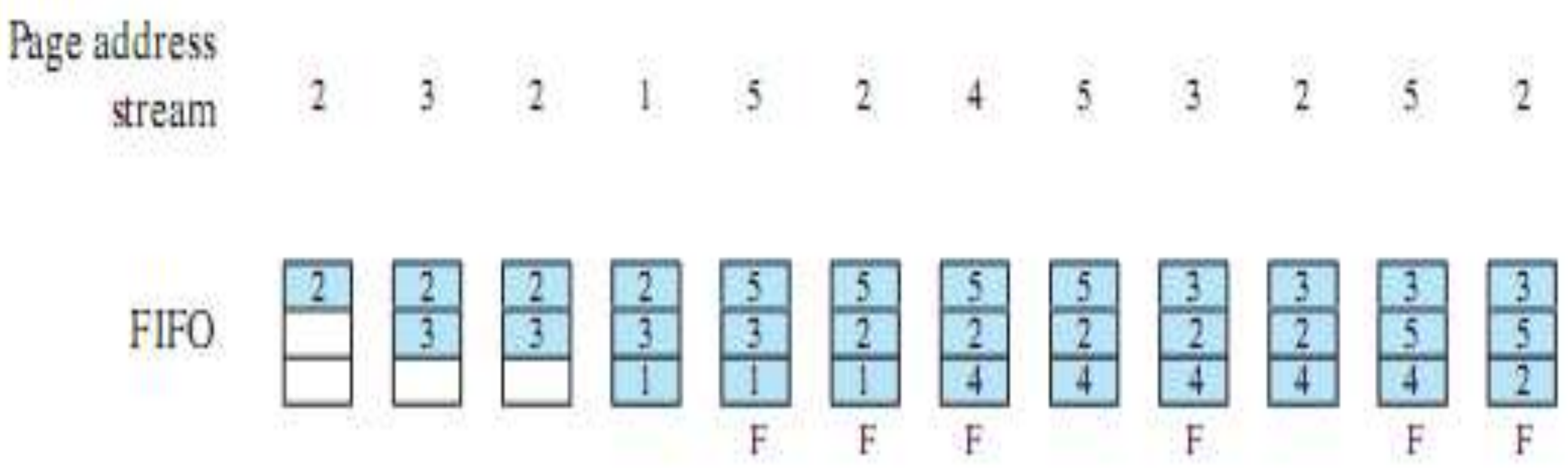
Figure 8.15 Behavior of Four Page Replacement Algorithms

- The LRU policy does nearly as well as the optimal policy.
  - In this example, there are four page faults

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
  - But, these pages may be needed again very soon if it hasn’t truly fallen out of use

FIFO Example



F= page fault occurring after the frame allocation is initially filled

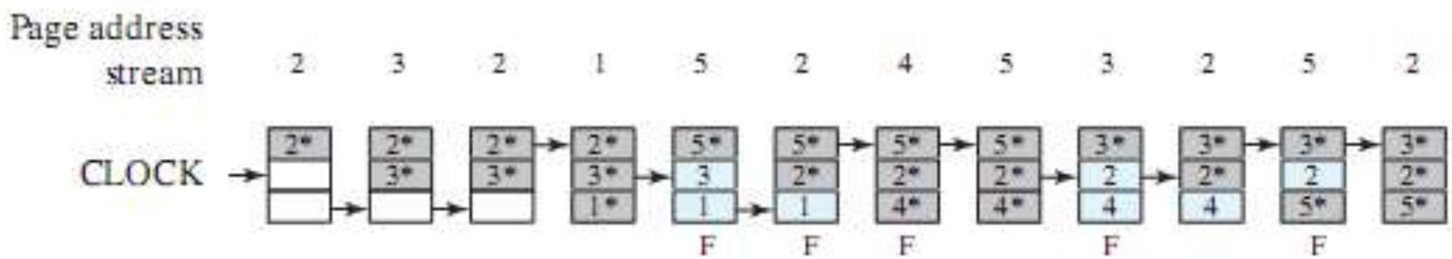
Figure 8.15 Behavior of Four Page Replacement Algorithms

- The FIFO policy results in six page faults.
  - Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.

Clock Policy

- Uses and additional bit called a “use bit”
- When a page is first loaded in memory or referenced, the use bit is set to 1
- When it is time to replace a page, the OS scans the set flipping all 1’s to 0
- The first frame encountered with the use bit already set to 0 is replaced.

Clock Policy Example

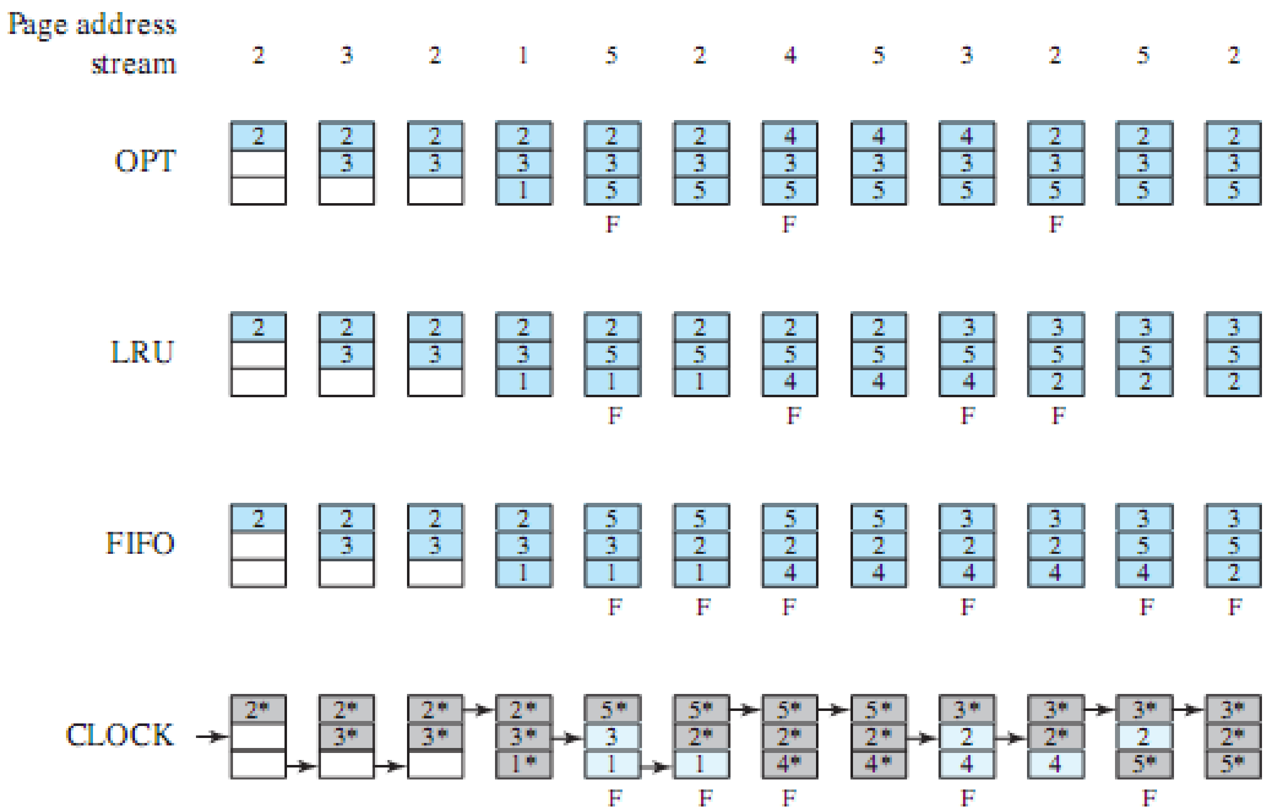


F= page fault occurring after the frame allocation is initially filled

Figure 8.15 **Behavior of Four Page Replacement Algorithms**

- Note that the clock policy is adept at protecting frames 2 and 5 from replacement.

Combined Examples



F= page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page Replacement Algorithms

Comparison

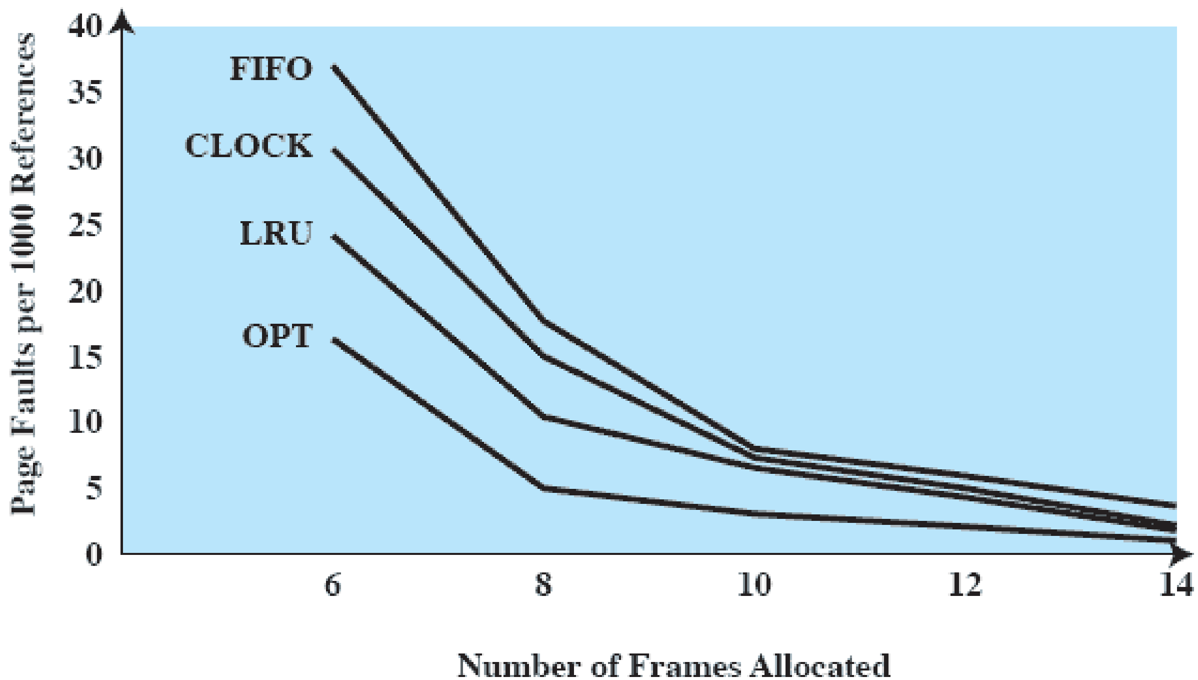
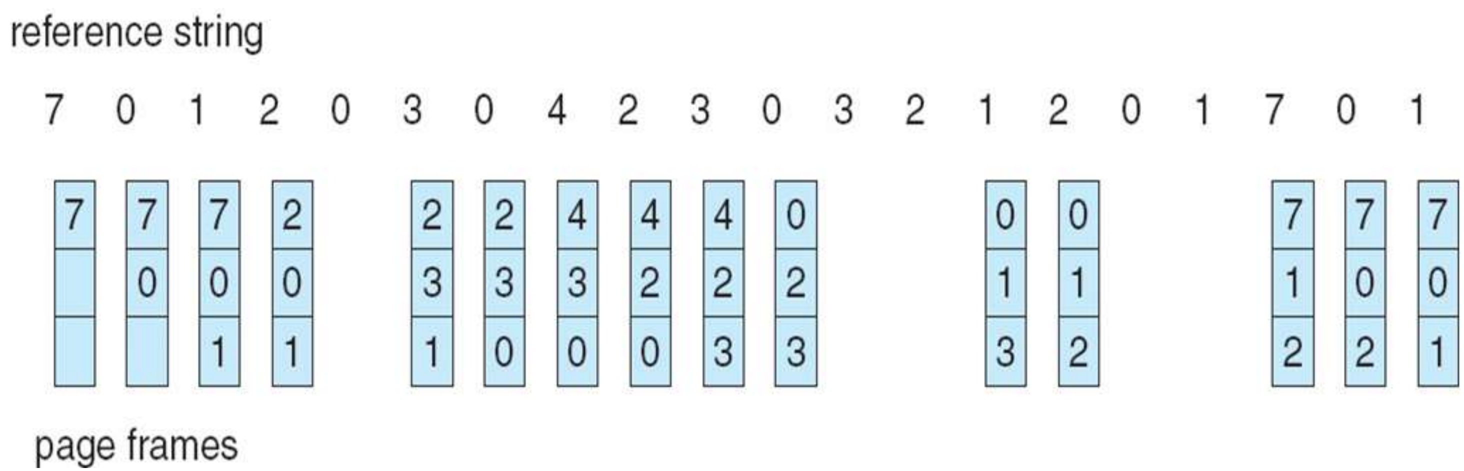


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

FIFO Page Replacement



LRU Page Replacement