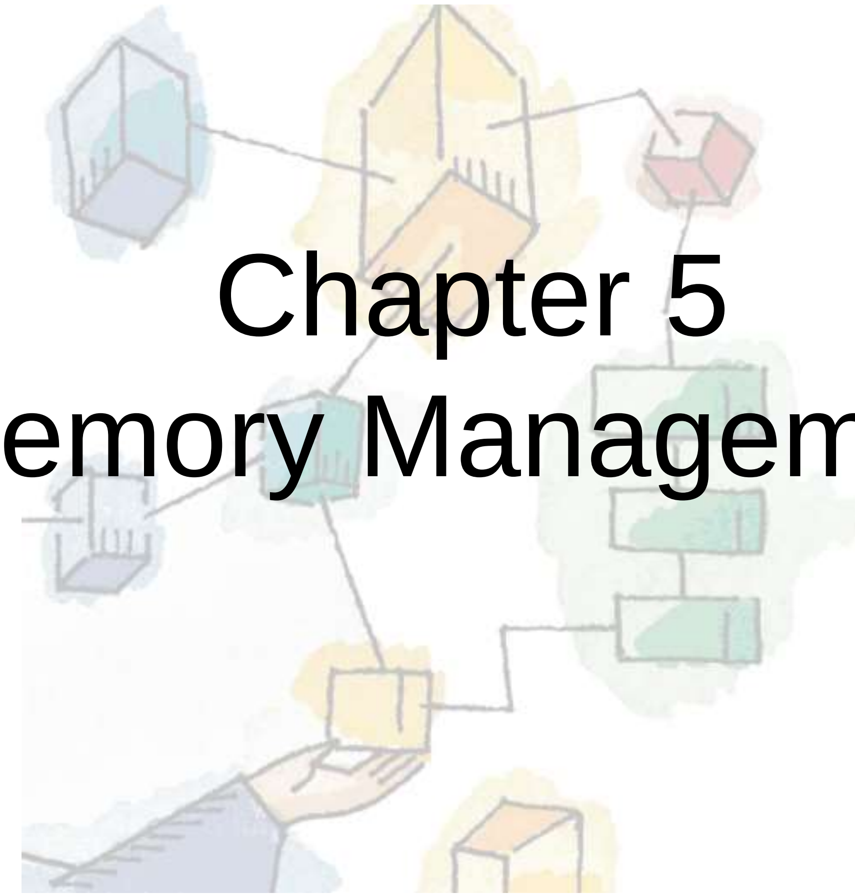
Operating Systems:
Internals and Design Principles, 6/E
William Stallings

Chapter 5
Memory Management



Roadmap

- Basic requirements of Memory Management
- Memory Partitioning
- Basic blocks of memory management
  - Paging
  - Segmentation

The need for memory management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never enough!
- 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

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

Memory Management Terms

Table 7.1 Memory Management Terms

Term	Description
Frame	***Fixed***-length block of main memory.
Page	***Fixed***-length block of data in secondary memory (e.g. on disk).
Segment	***Variable-length*** block of data that resides in secondary memory.

Addressing

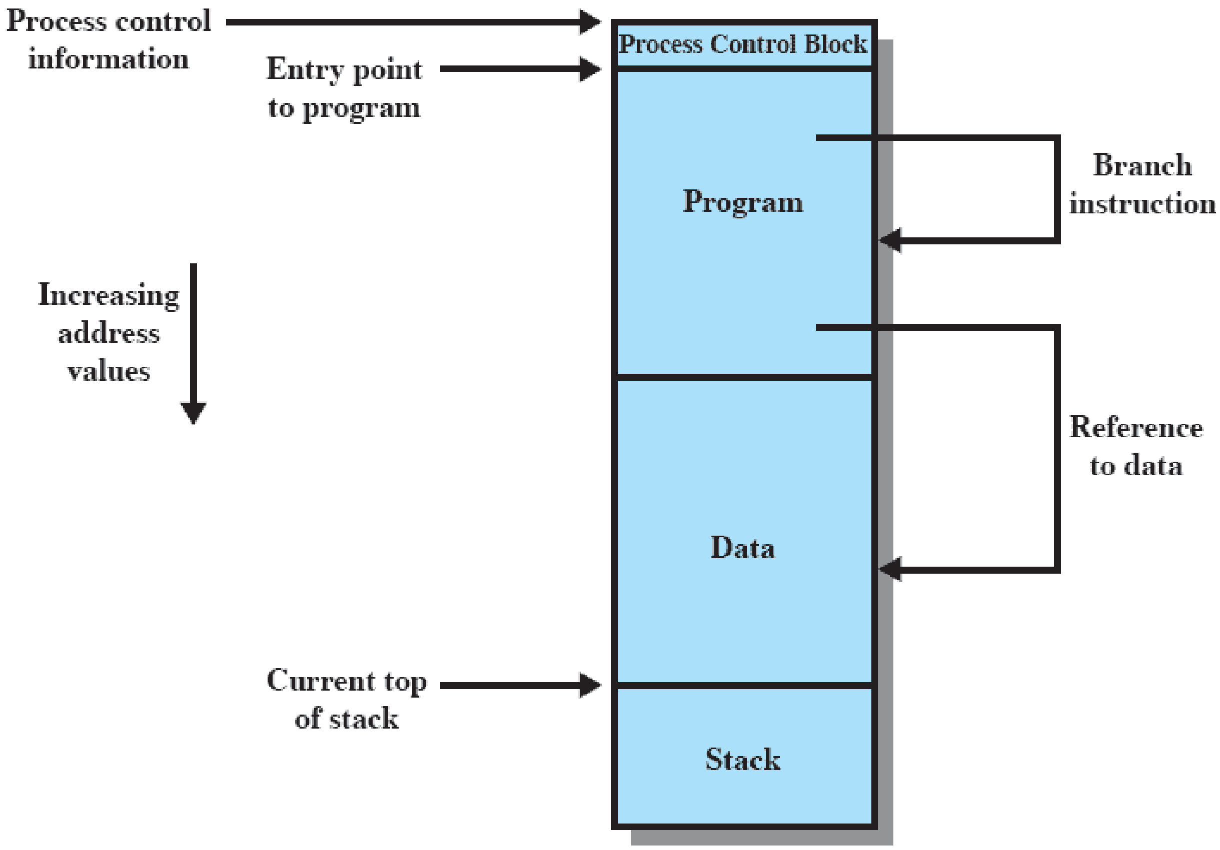


Figure 7.1 Addressing Requirements for a Process

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 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

Partitioning

- An early method of managing memory
  - Pre-virtual memory
  - Not used much now
- But, it will clarify the later discussion of virtual memory if we look first at partitioning
  - Virtual Memory has evolved from the partitioning methods

Types of Partitioning

- Fixed Partitioning
- Dynamic Partitioning
- Simple Paging
- Simple Segmentation
- Virtual Memory Paging
- Virtual Memory Segmentation

Fixed Partitioning

- Equal-size partitions (see fig 7.3a)
  - 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

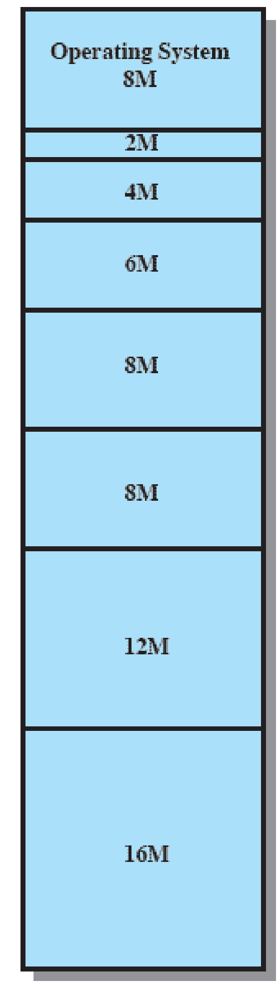
(a) Equal-size partitions

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

- Lessens both problems
  - but doesn’t solve completely
- In Fig 7.3b,
  - Programs up to 16M can be accommodated without overlay
  - Smaller programs can be placed in smaller partitions, reducing internal fragmentation

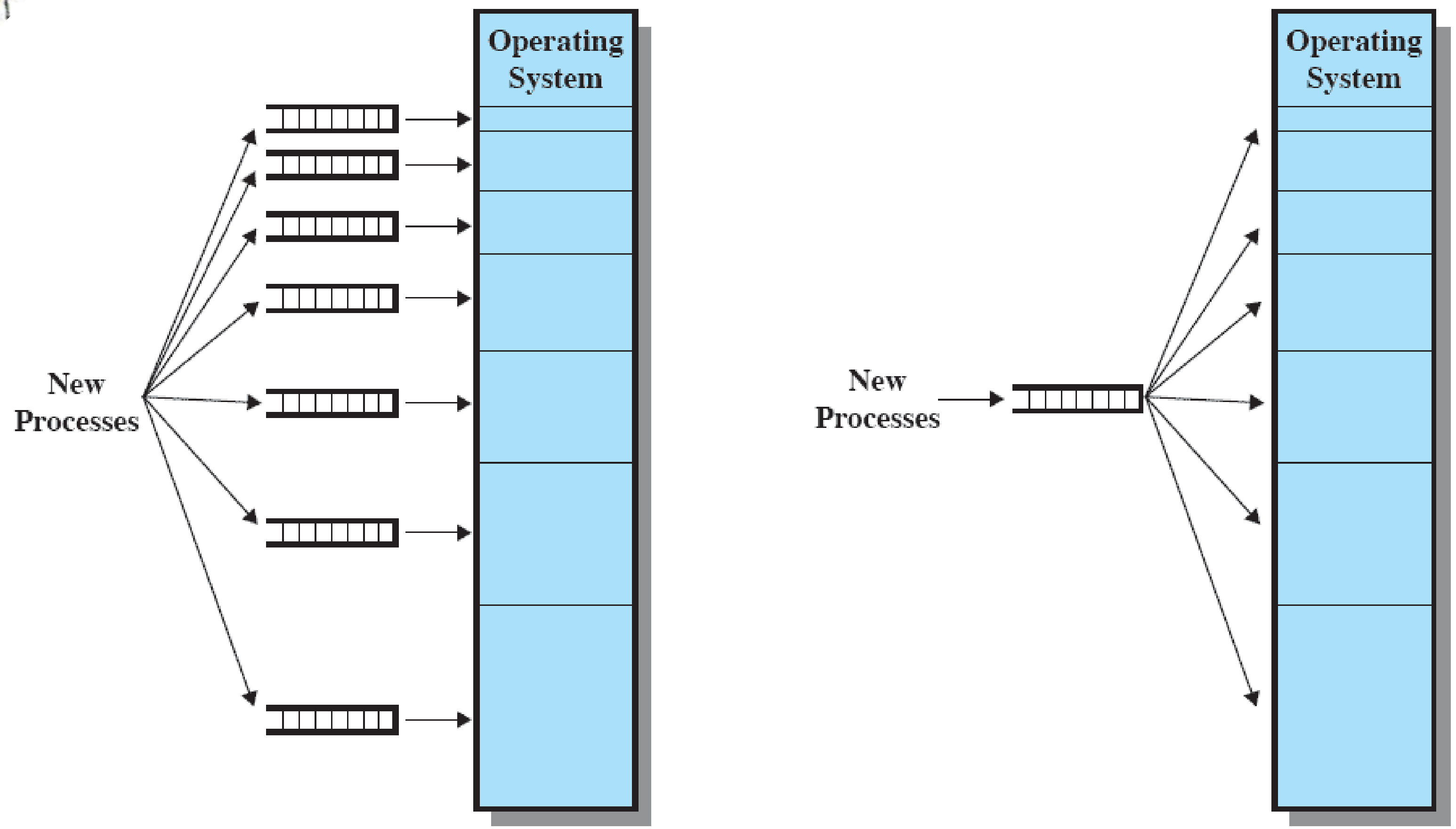


(b) Unequal-size partitions

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

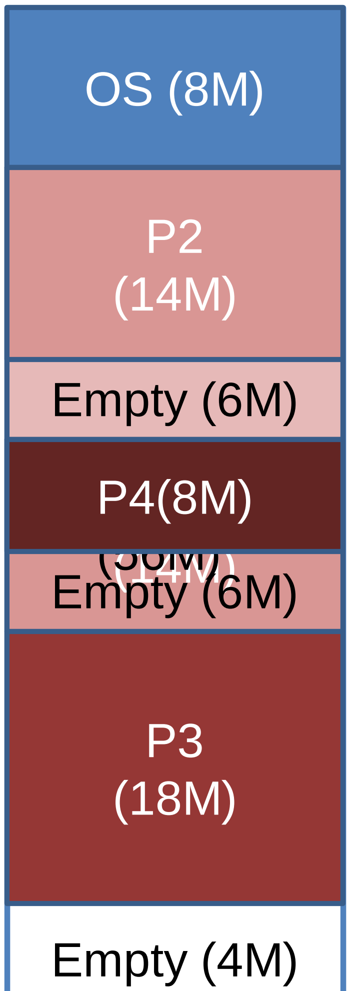
Remaining Problems with Fixed Partitions

- The number of active processes is limited by the system
  - I.E 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

Dynamic Partitioning

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

Dynamic Partitioning Example



Refer to Figure 7.4

- ***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
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often

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

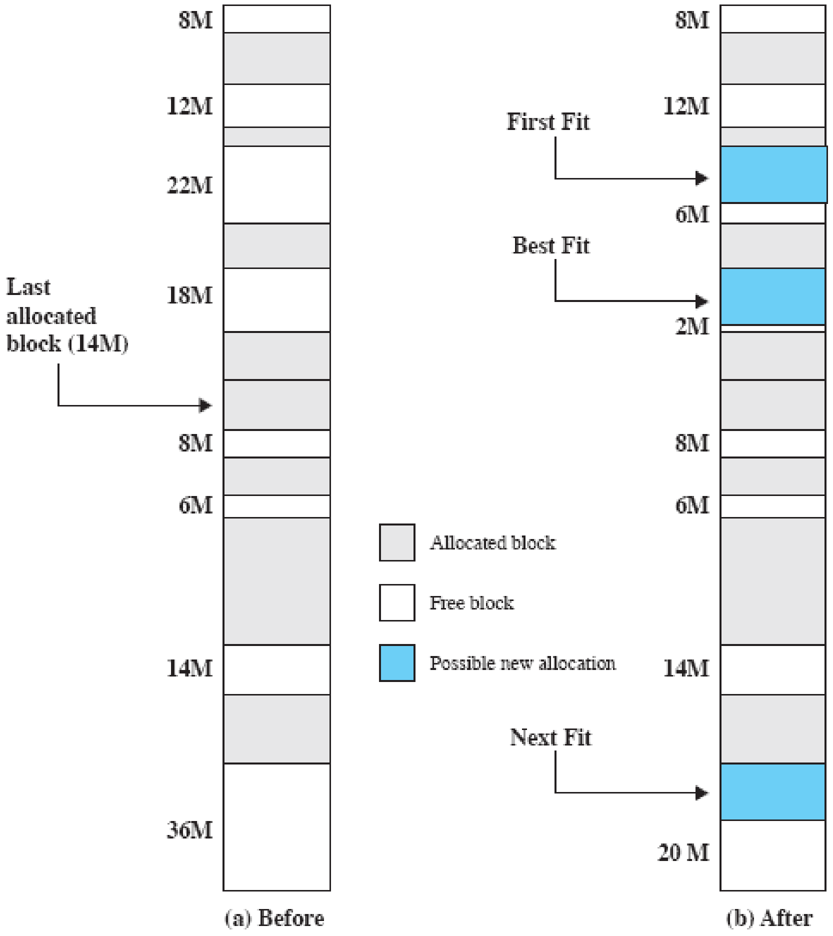


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

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

Example of Buddy System

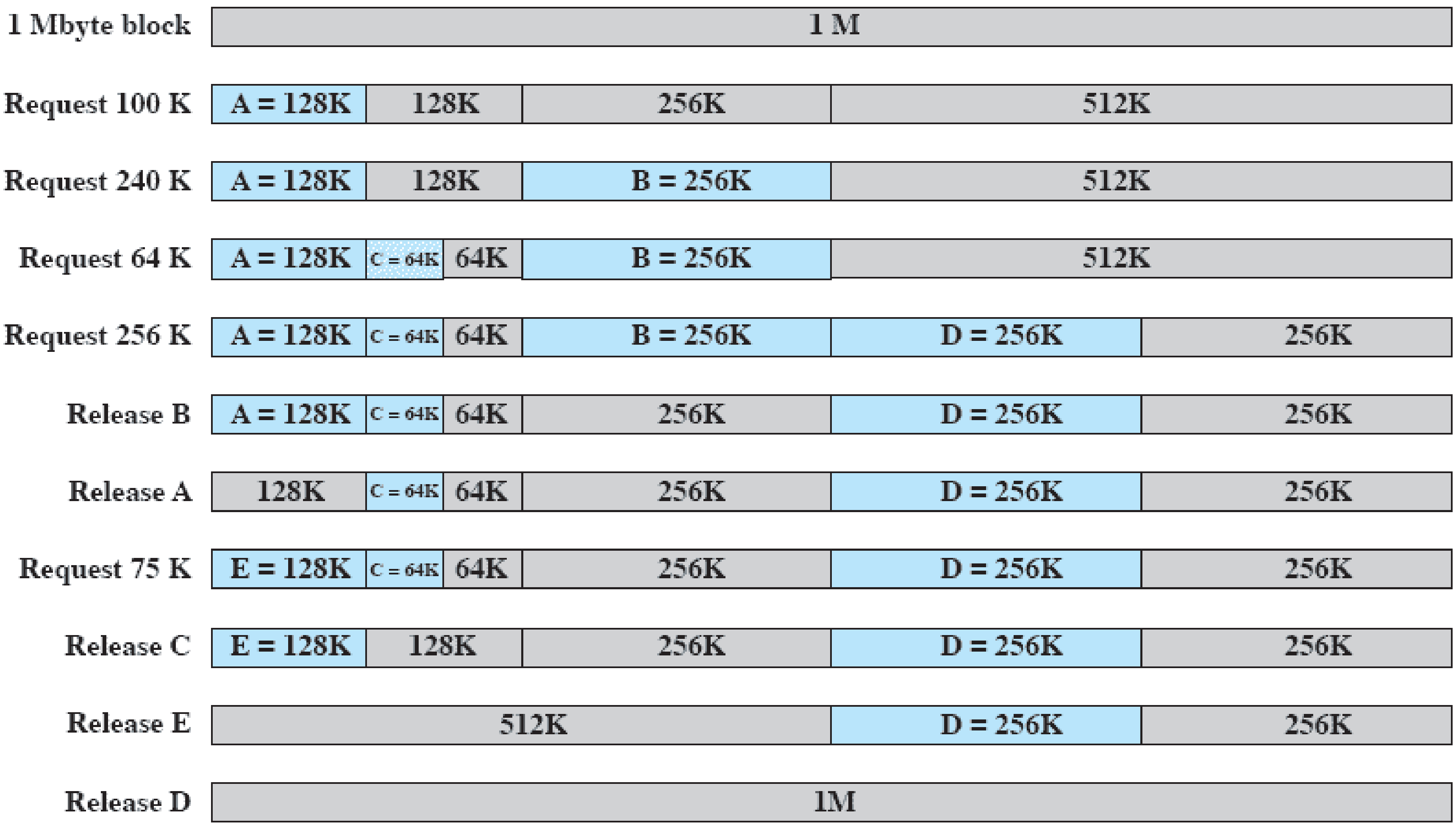


Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

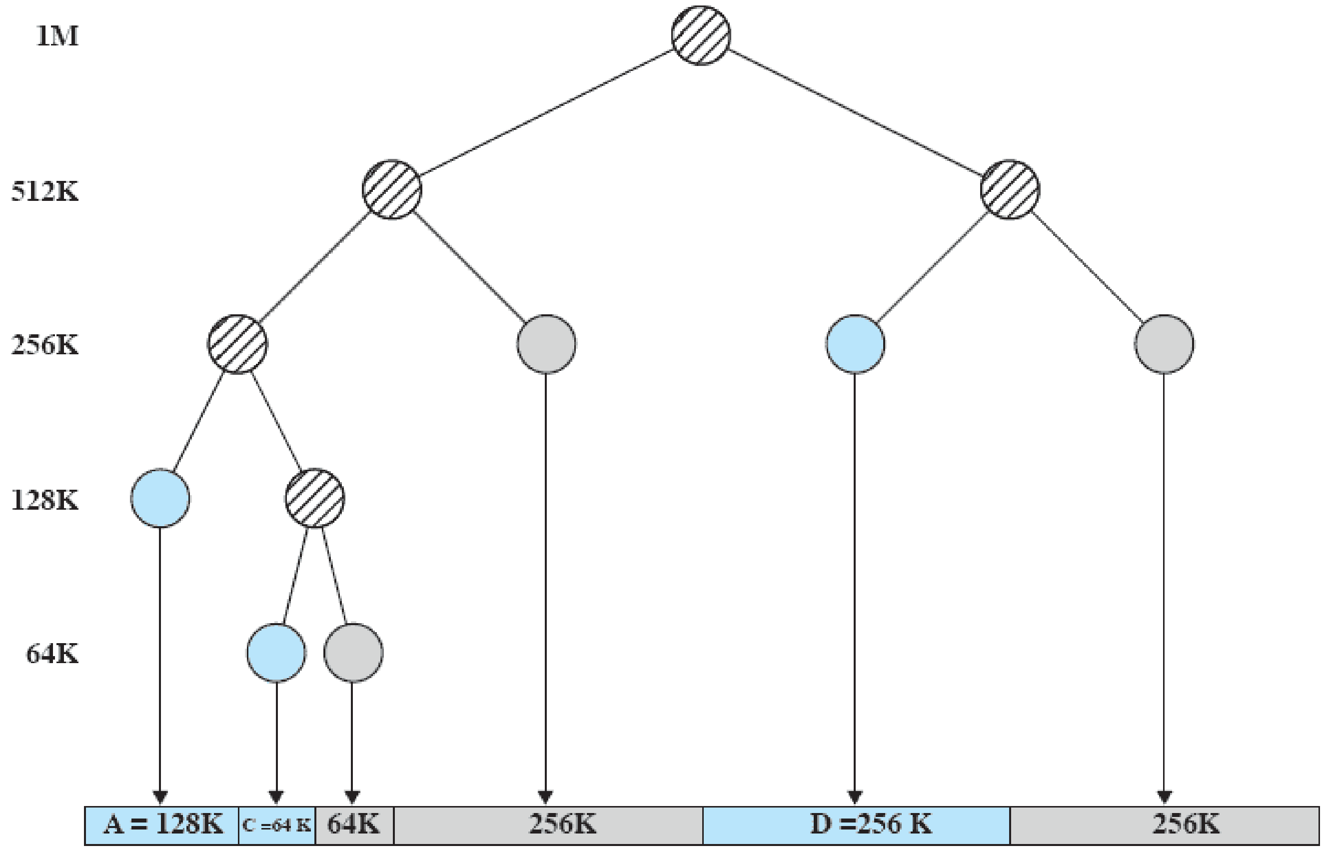


Figure 7.7 Tree Representation of Buddy System

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

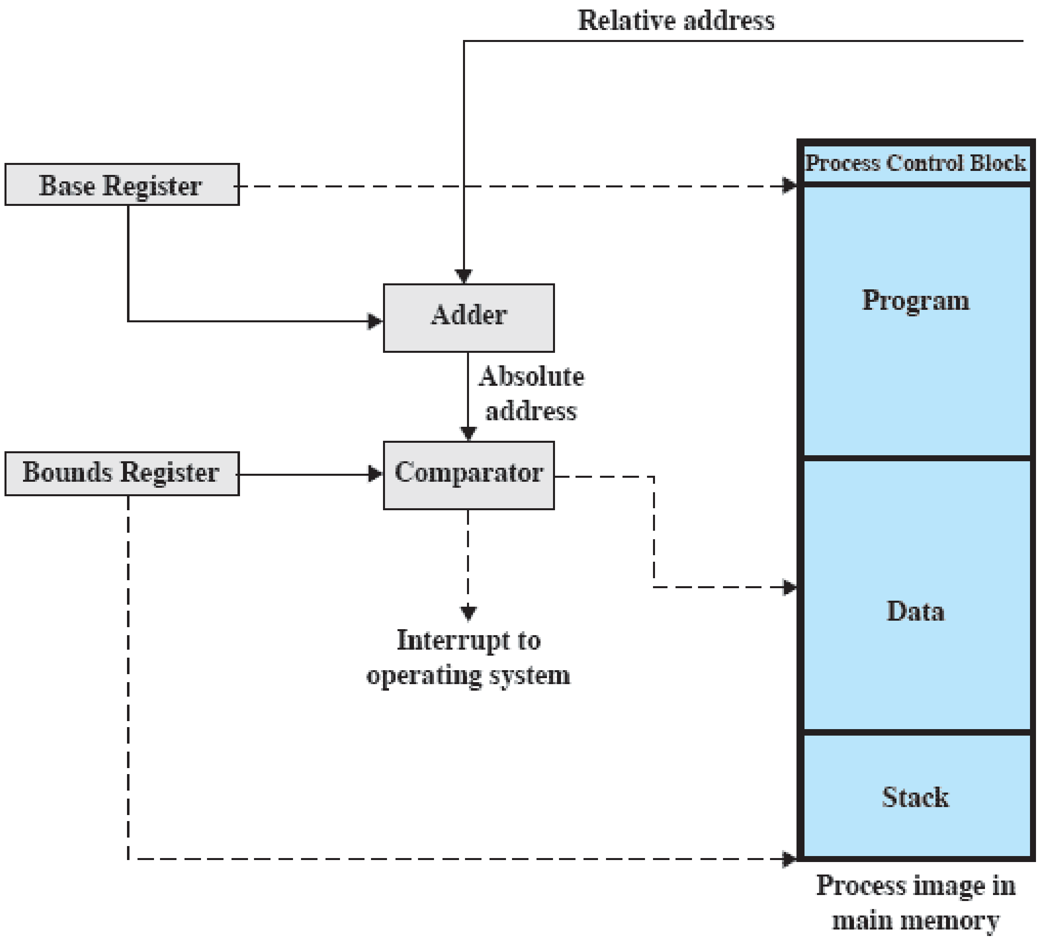


Figure 7.8 Hardware Support for Relocation

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

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

Page Table

0	0
1	1
2	2
3	3

Process A page table

0	—
1	—
2	—

Process B page table

0	7
1	8
2	9
3	10

Process C page table

0	4
1	5
2	6
3	11
4	12

Process D page table

13
14

Free frame list

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

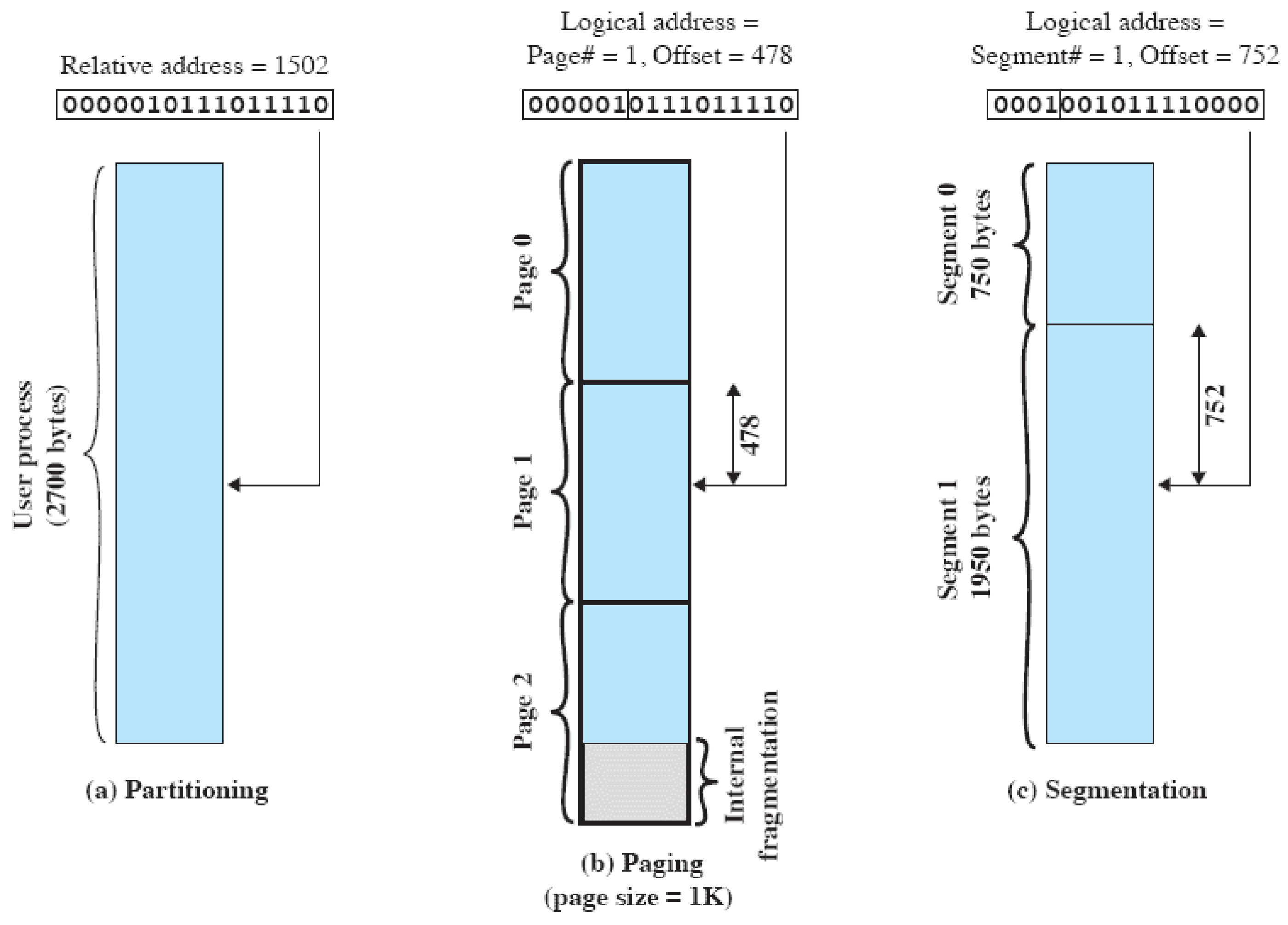
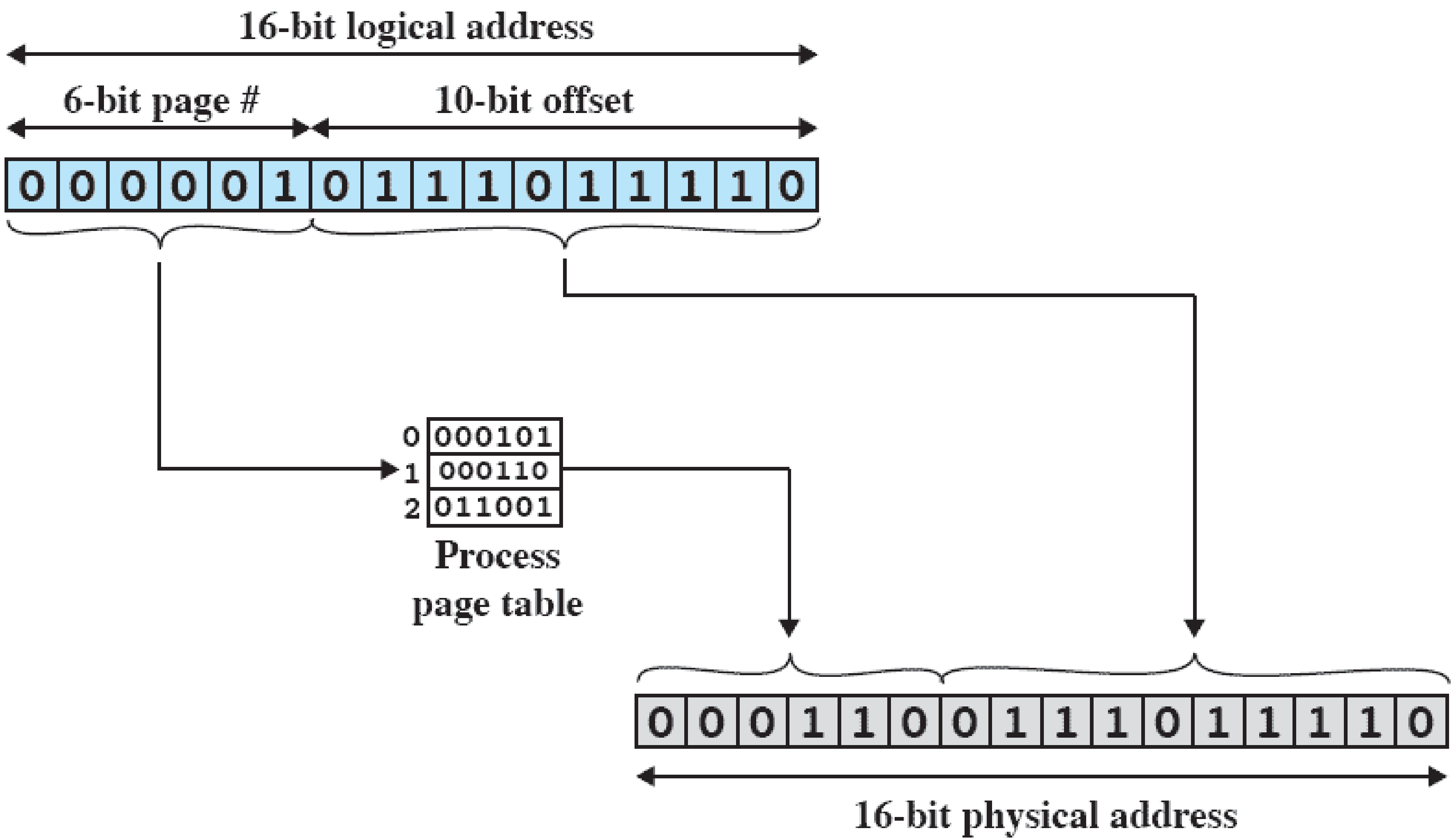


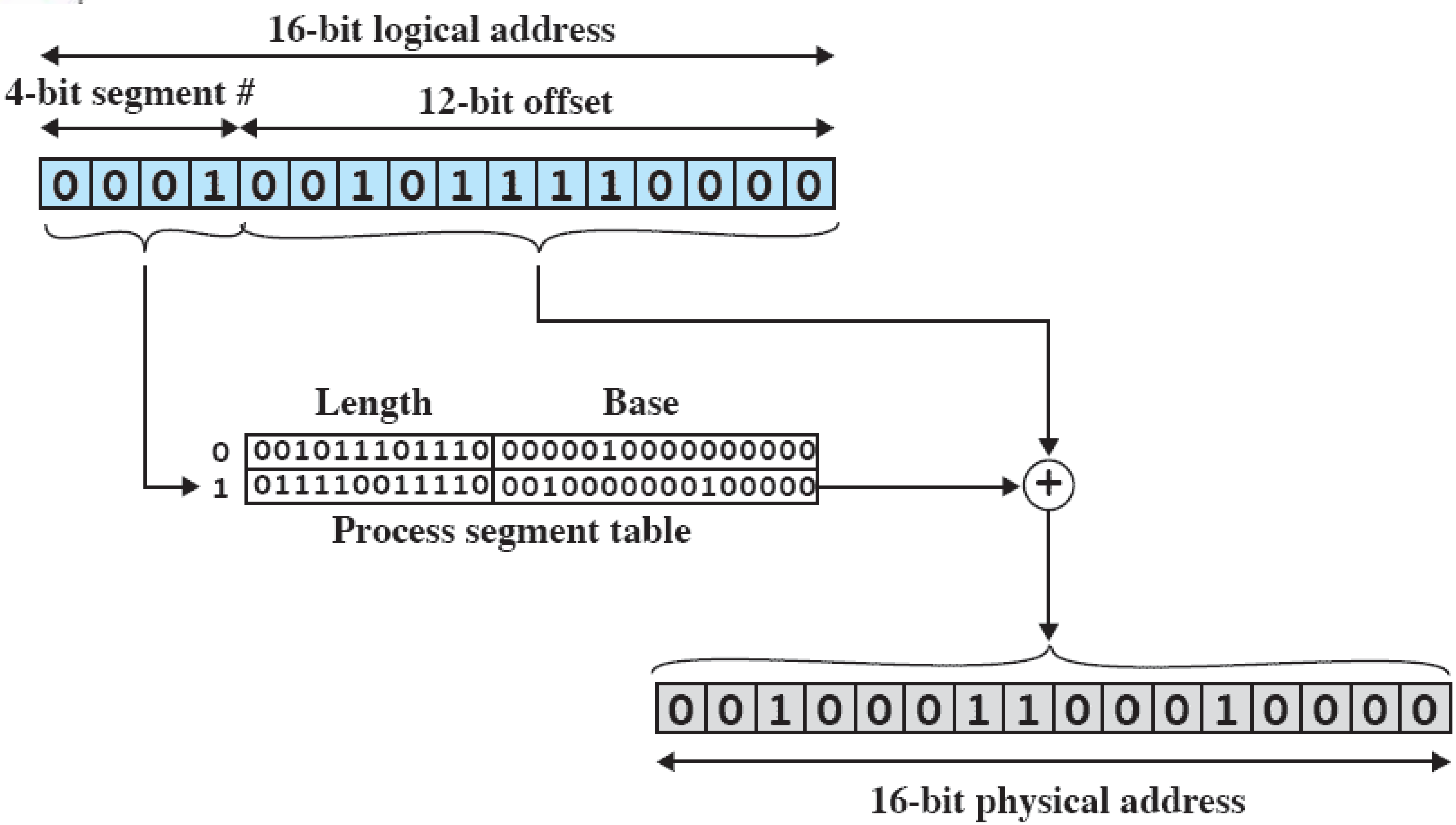
Figure 7.11 Logical Addresses

Paging



(a) Paging

Segmentation



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation