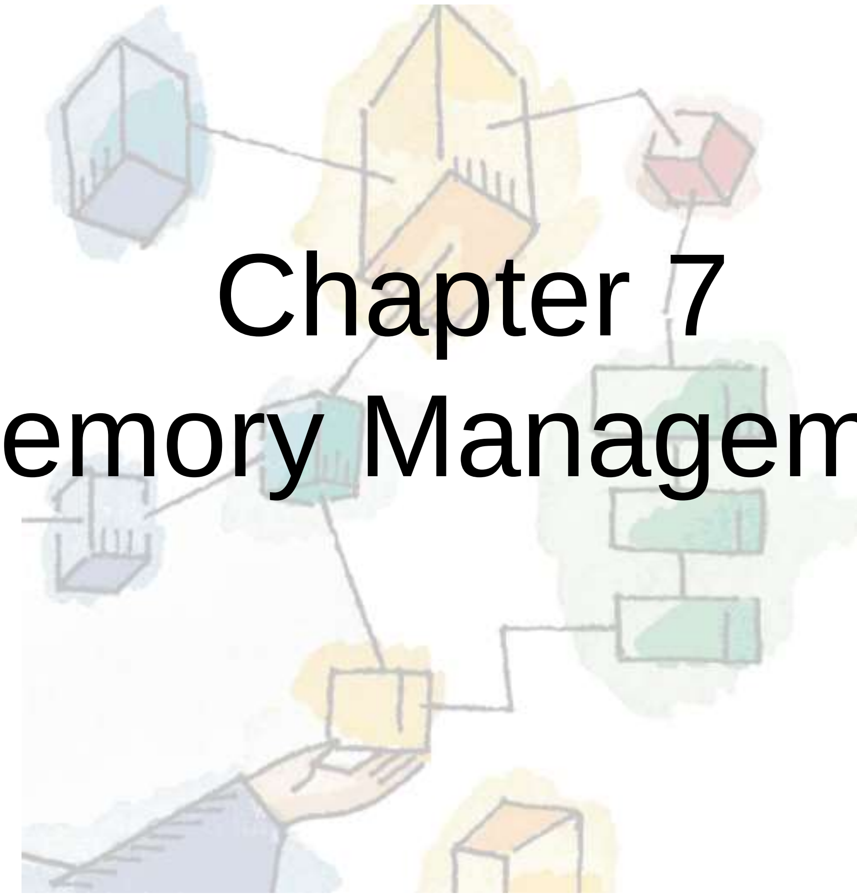
*Operating Systems:*
*Internals and Design Principles, 6/E*
William Stallings



# Chapter 7
# 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

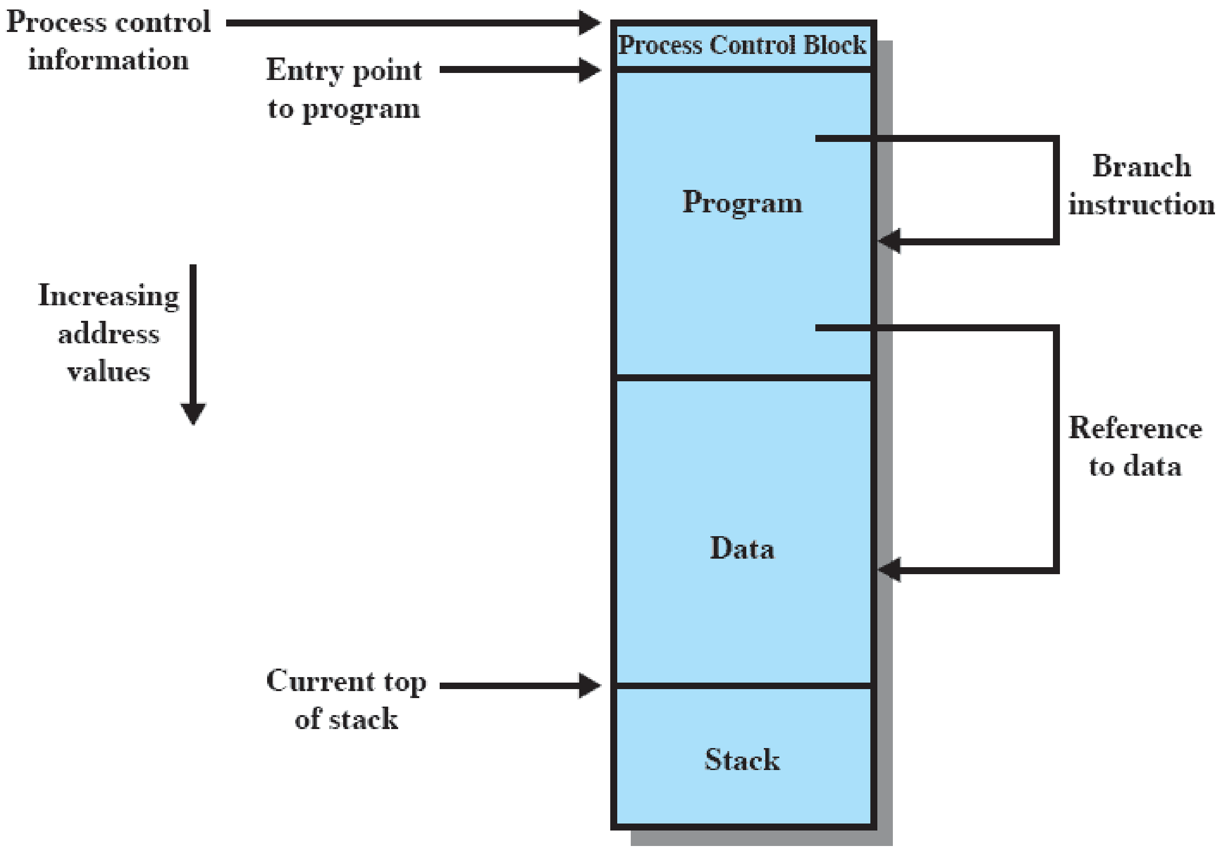



**Figure 7.1 Addressing Requirements for a Process**

# 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

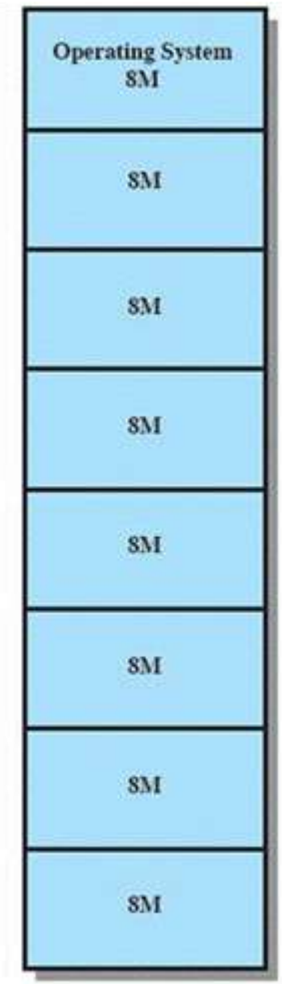
- An early method of managing memory
  - Pre-virtual memory
  - Not used much now
- But, it will clarify the later discussion of virtual memory if we look first at partitioning
  - Virtual Memory has evolved from the partitioning methods

# 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



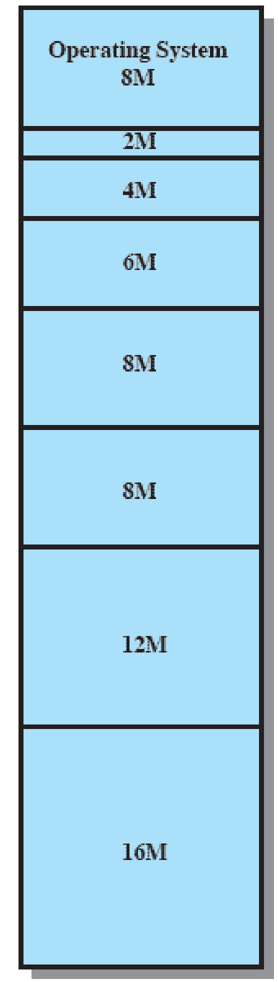

(a) Equal-size partitions

# 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

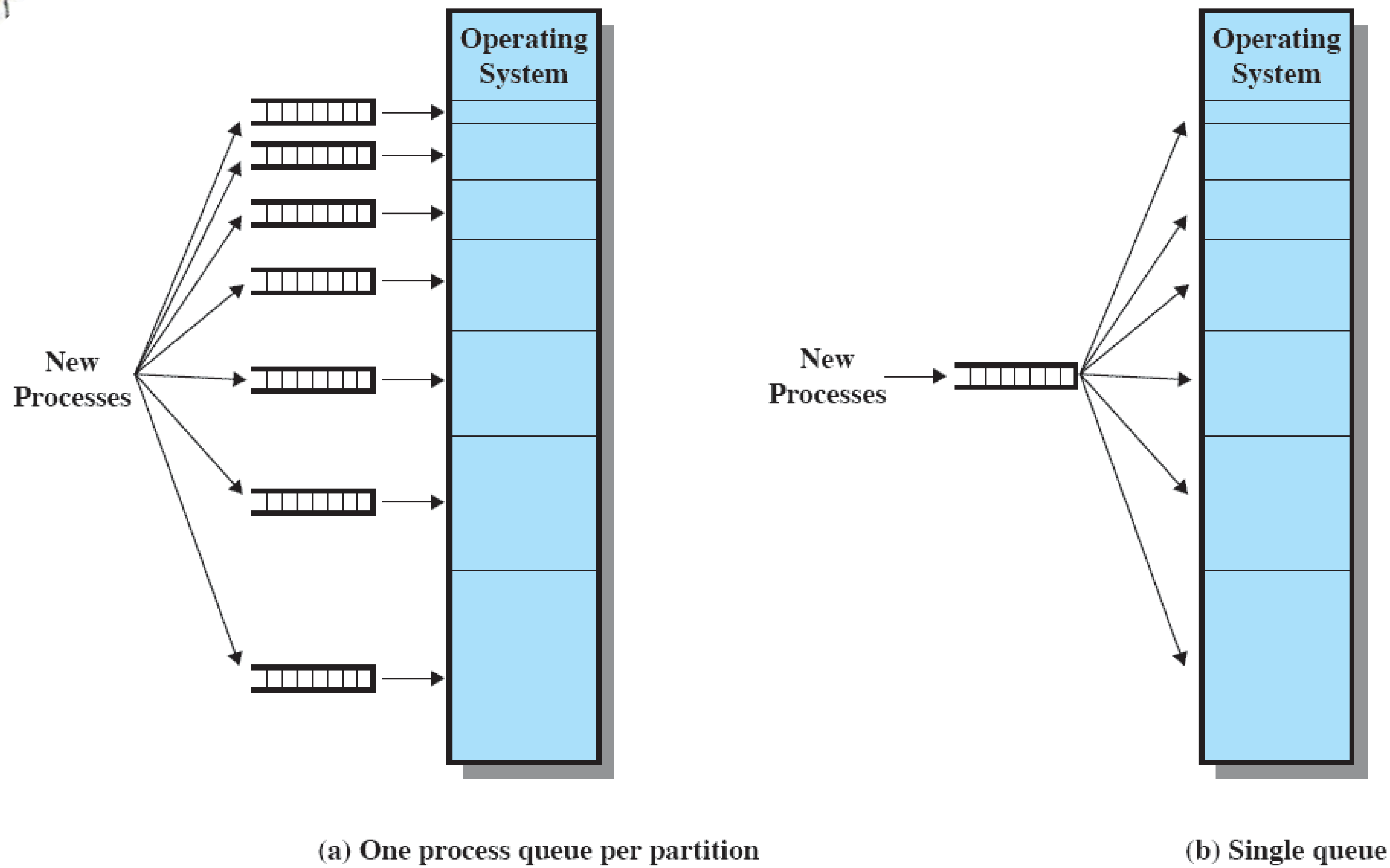



Figure 7.3 Memory Assignment for Fixed Partitioning

# 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


- ***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

Refer to Figure 7.4

# 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

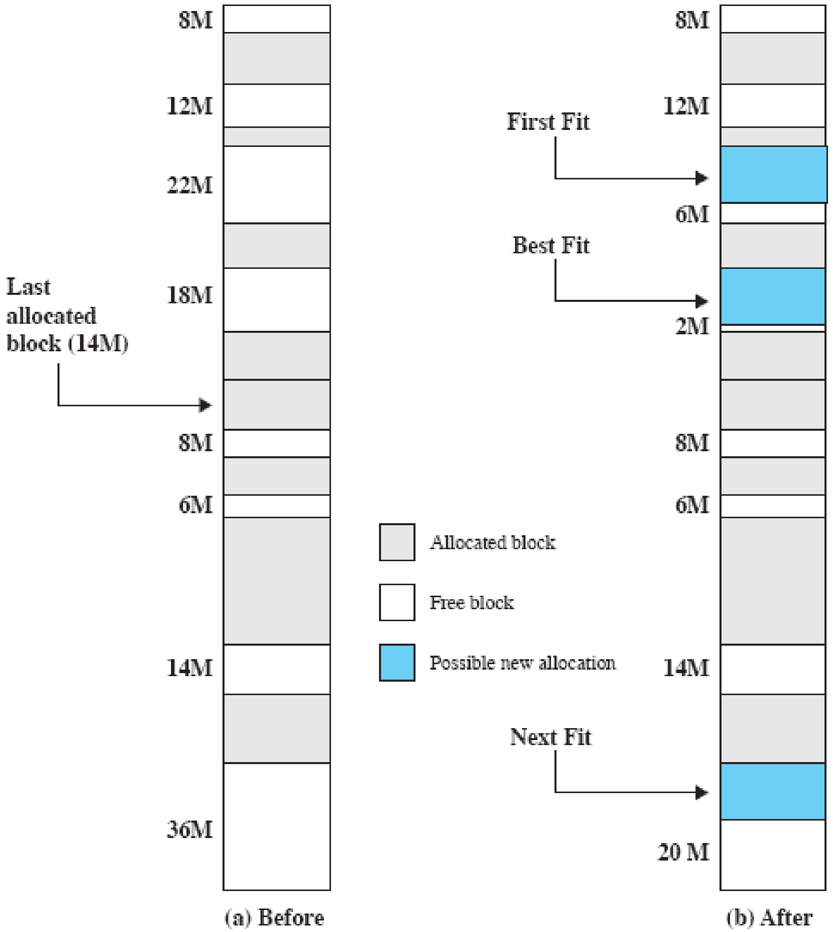



**Figure 7.5** Example Memory Configuration before and after Allocation of 16-Mbyte Block

# 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

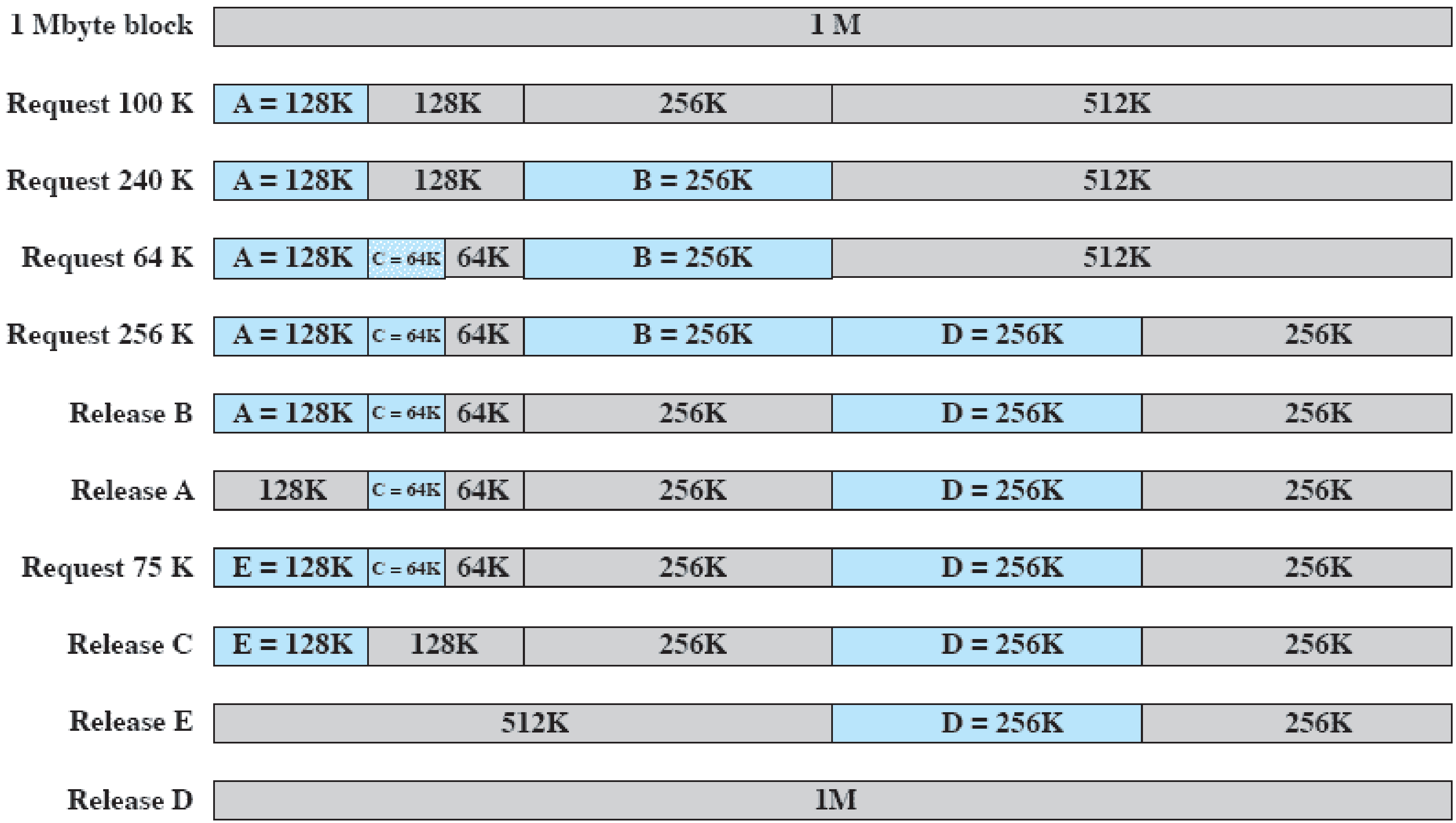



**Figure 7.6 Example of Buddy System**

# Tree Representation of Buddy System

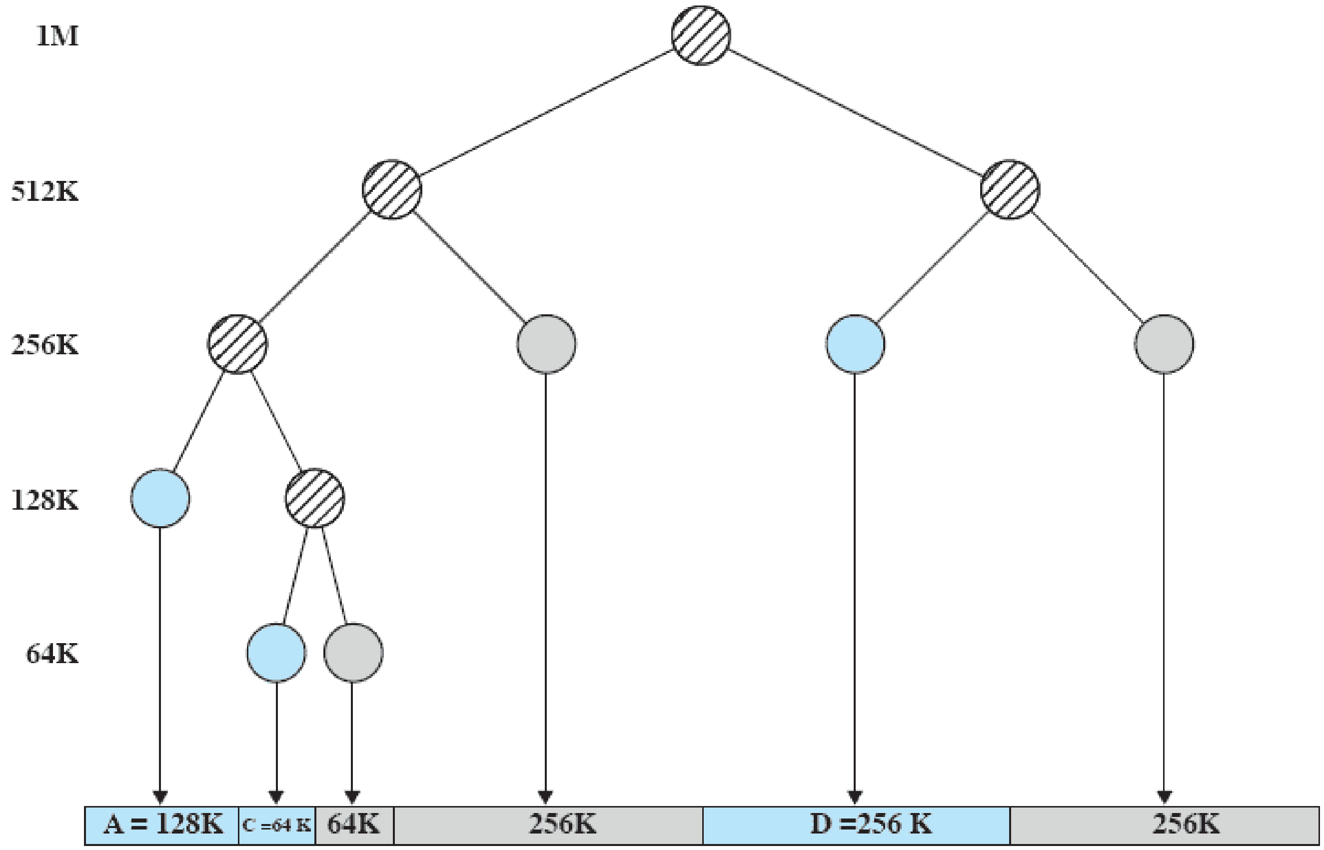



Figure 7.7 Tree Representation of Buddy System

# 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

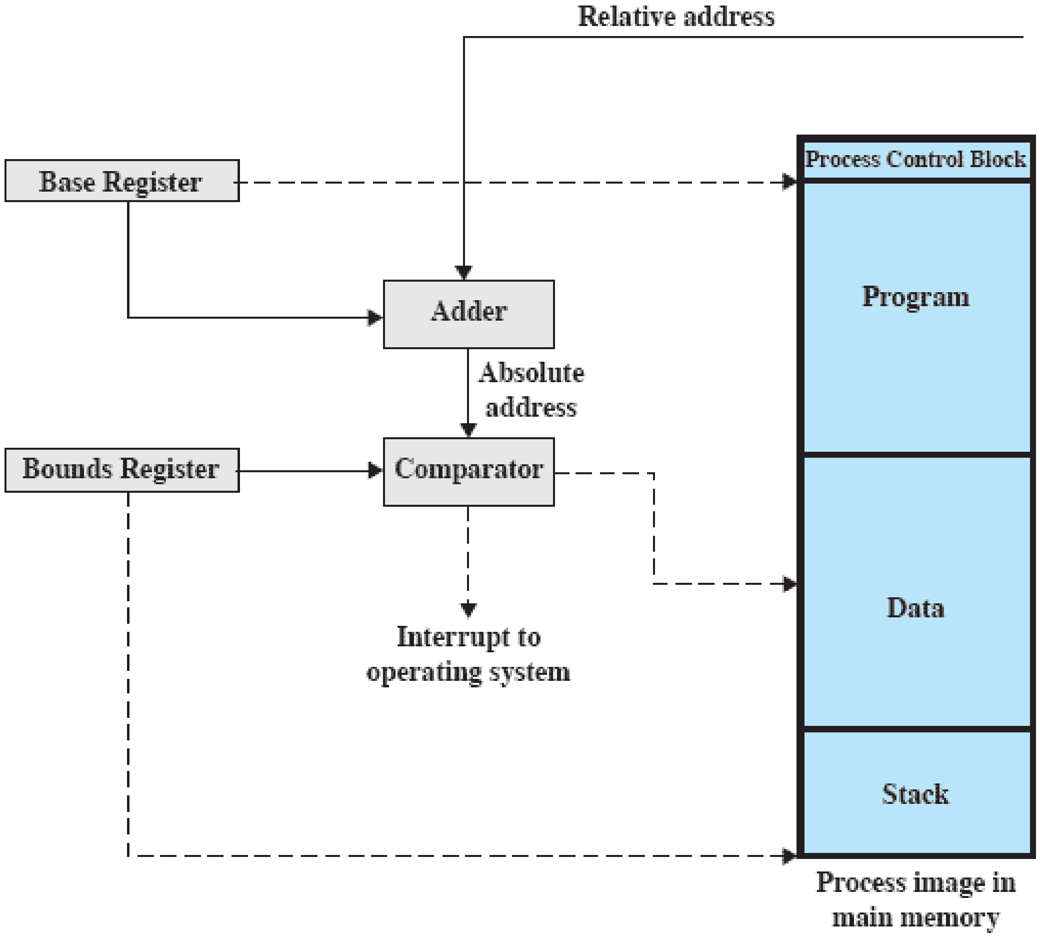



**Figure 7.8 Hardware Support for Relocation**

# 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

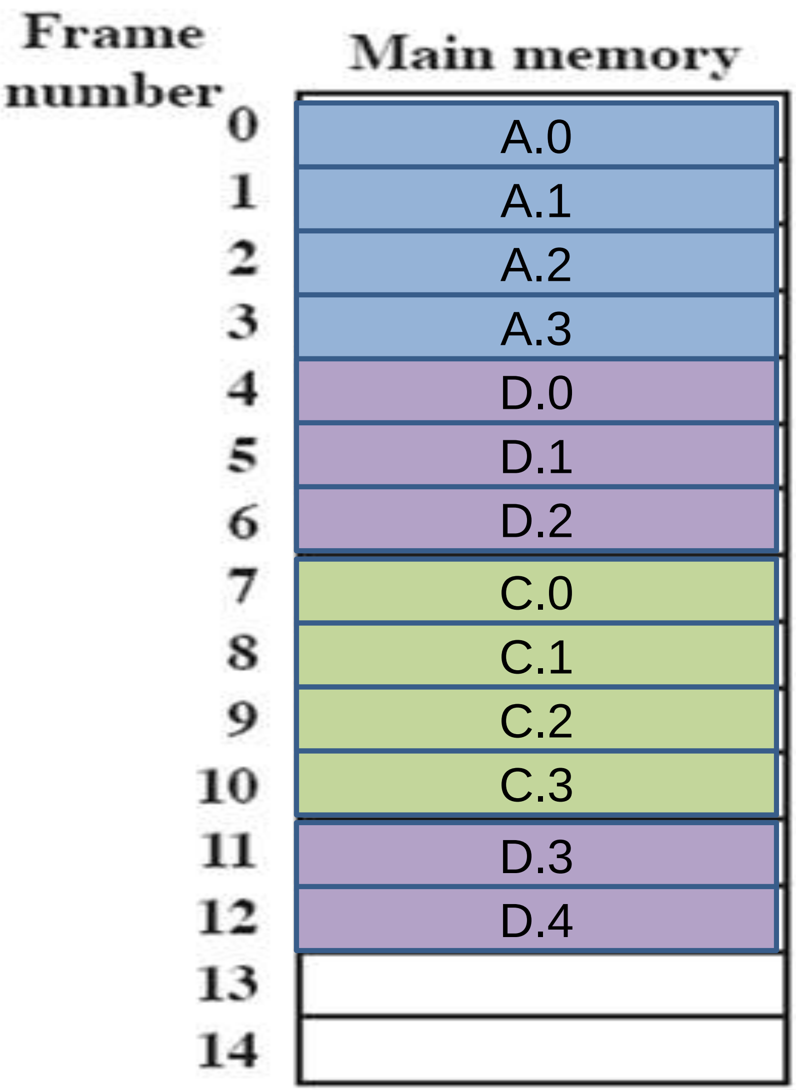
# Paging

- 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



# 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)

# Logical Addresses

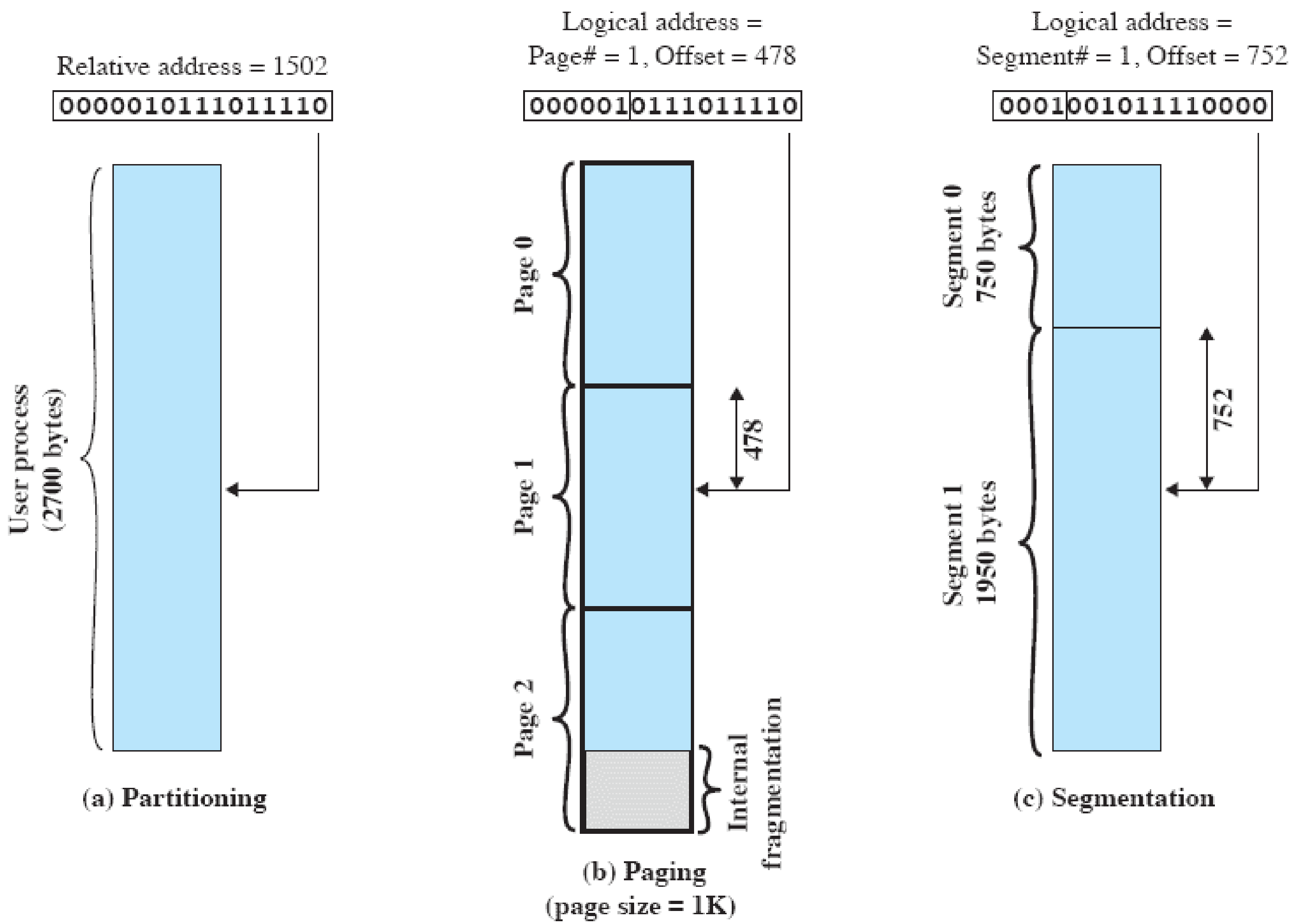
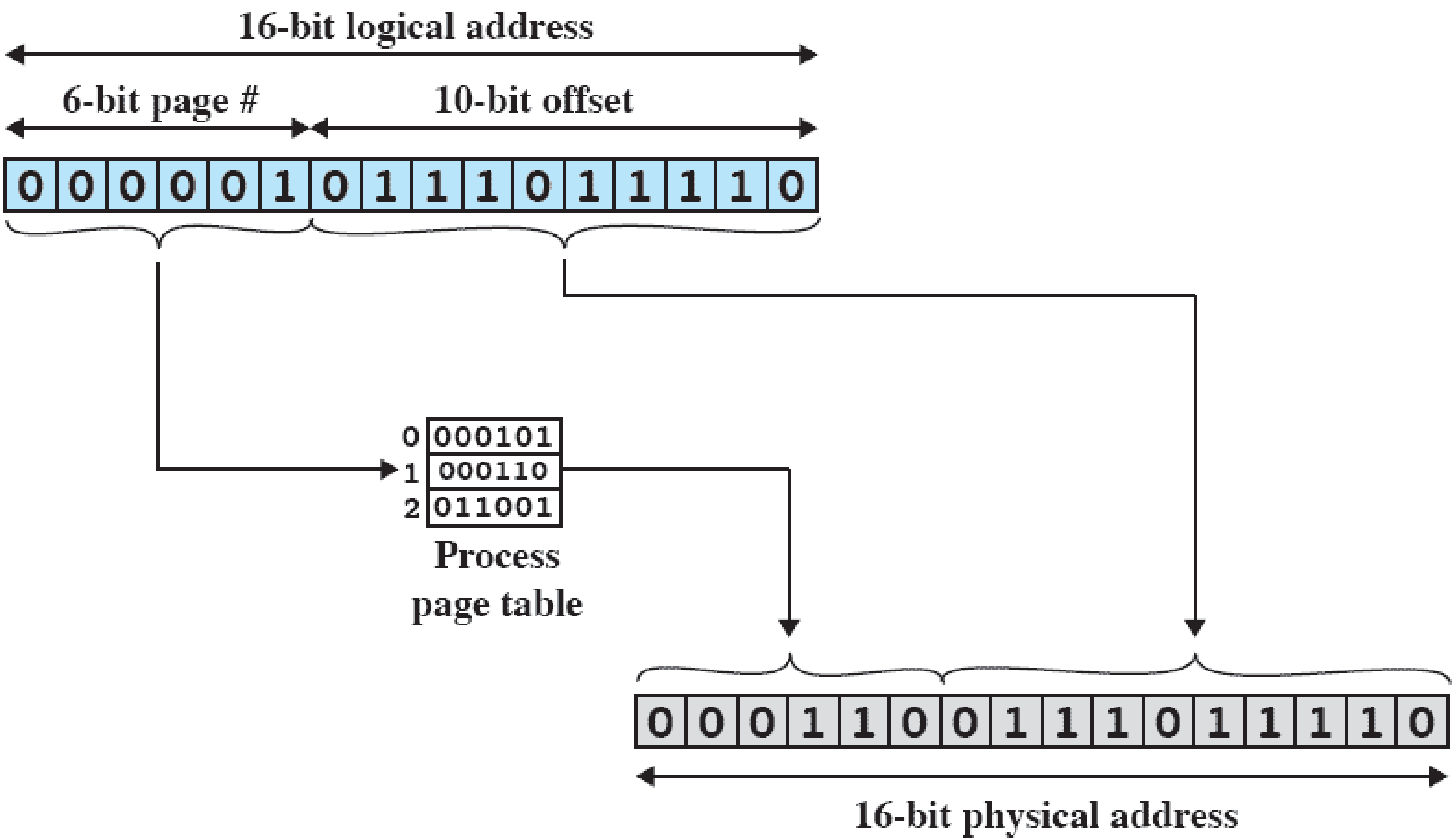



**Figure 7.11** Logical Addresses

# Paging



(a) Paging

# Paging

- Memory-management technique that permits the physical address space of a process to be non-contiguous.

- Each process is divided into a number of small, fixed-size partitions called pages or Divide logical memory into blocks of same size called **pages.**

- Physical memory is divided into a large number of small, fixed-size partitions called **frames.**

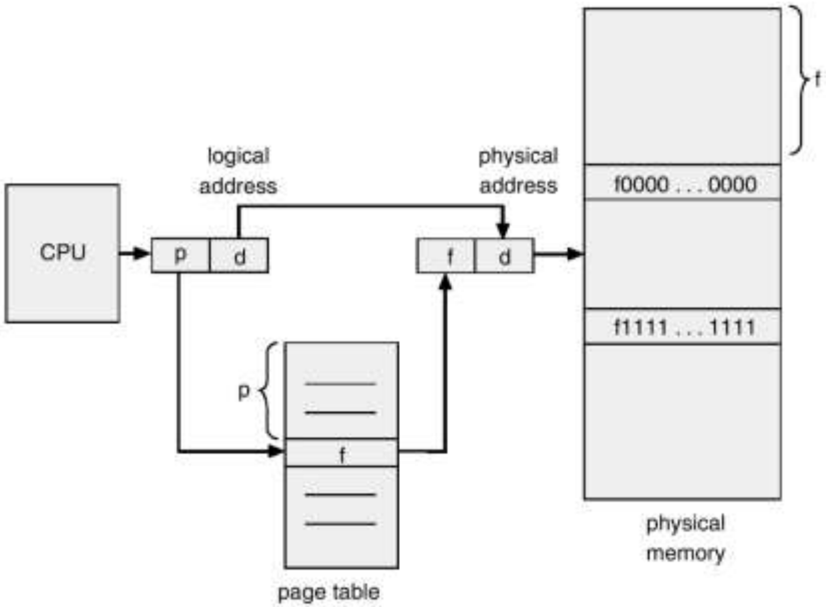
- When a process is to be executed, its pages are loaded into any available memory frames.

- Some internal fragmentation , but no external fragmentation.

# Address Translation Scheme

- Operating system maintains a page table for each process
  - Contains the frame number for each process page
  - To translate logical to physical addresses

- A logical address 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.



- Paging Model of memory is shown:

| | |
|---|---|
| Page 0 | |
| Page 1 | |
| Page 2 | |
| Page 3 | |

Logical memory

| | |
|---|---|
| 0 | 1 |
| 1 | 4 |
| 2 | 3 |
| 3 | 7 |

Page table

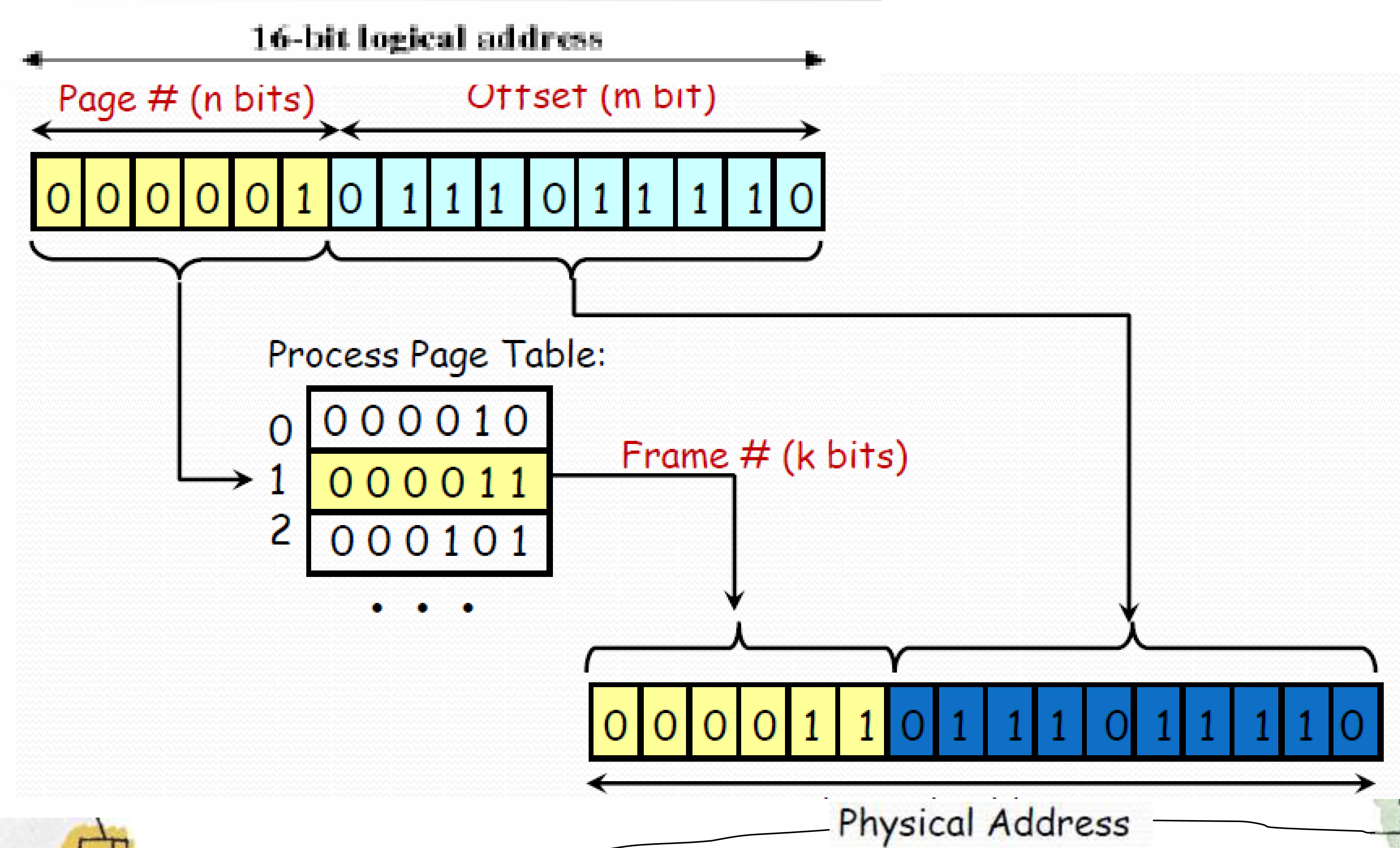
| Frame number | |
|---|---|
| 0 | |
| 1 | Page 0 |
| 2 | |
| 3 | Page 2 |
| 4 | Page 1 |
| 5 | |
| 6 | |
| 7 | Page 3 |

Physical memory

- To run a program of size ***n*** pages, need to find *n* free frames and load program.

- Set up a page table to translate logical to physical addresses

# Address translation in paging



# Segmentation

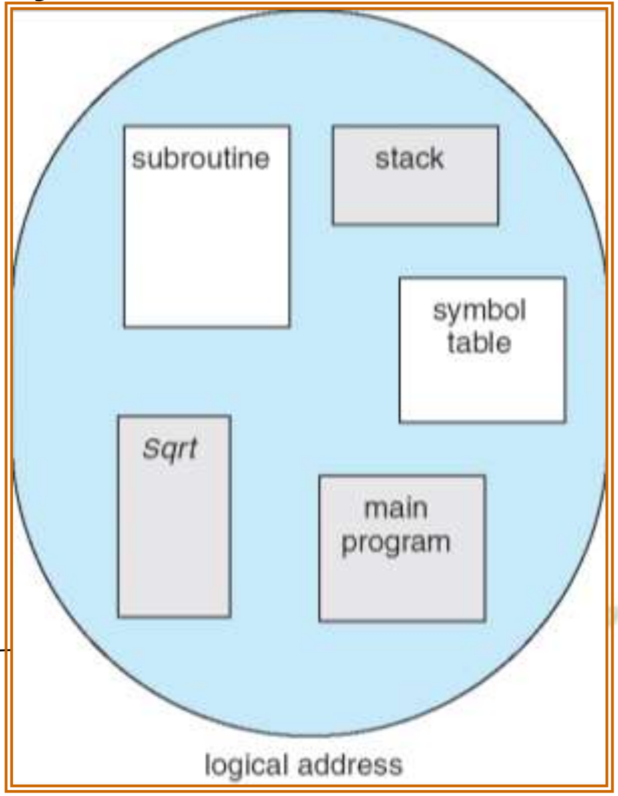
- A program can be subdivided into segments
  - Segments may vary in length
  - There is a maximum segment length
- Addressing consist of two parts
  - a segment number and
  - an offset
- Segmentation is similar to dynamic partitioning

# Segmentation

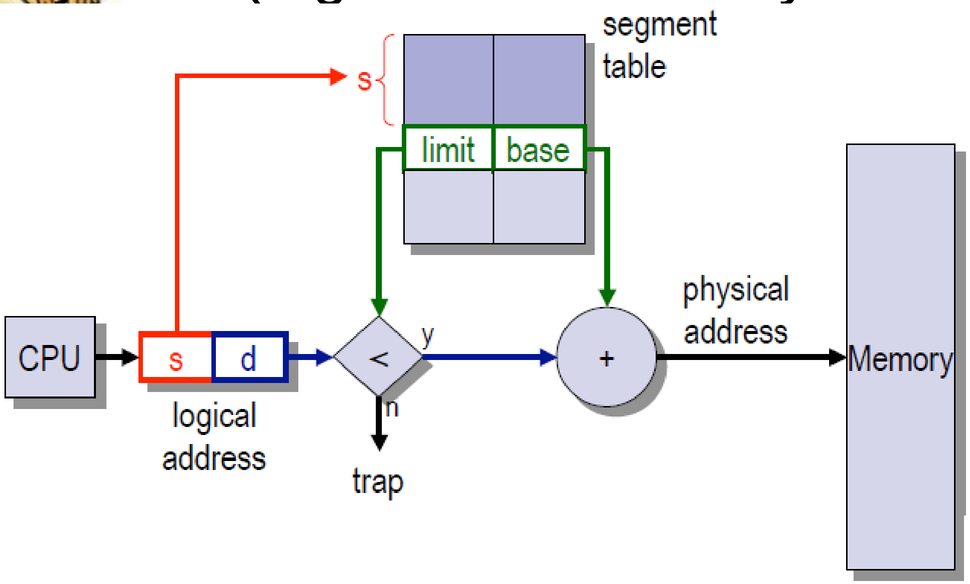
- A segment can be defined as a logical grouping of instructions, such as a subroutine, array, or a data area.
- A program is a collection of segments.
- Segmentation is a technique for managing these segments.
- Divide a process into unequal size blocks called segments.
- Each segment has a name and a length.
- Memory Management Scheme that supports user view of memory.

- A segment is a logical unit such as
  - main program, procedure, function
  - local variables, global variables, common block
  - stack, symbol table, arrays
- Protect each entity independently
- Allow each segment to grow independently
- Share each segment independently

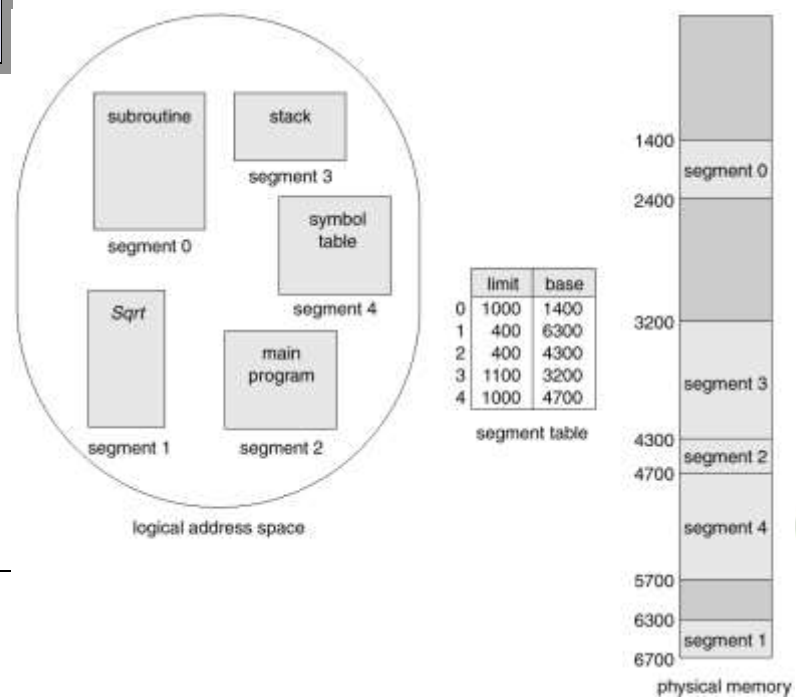
- Each of these segments are of variable length and length will be defined in the program.



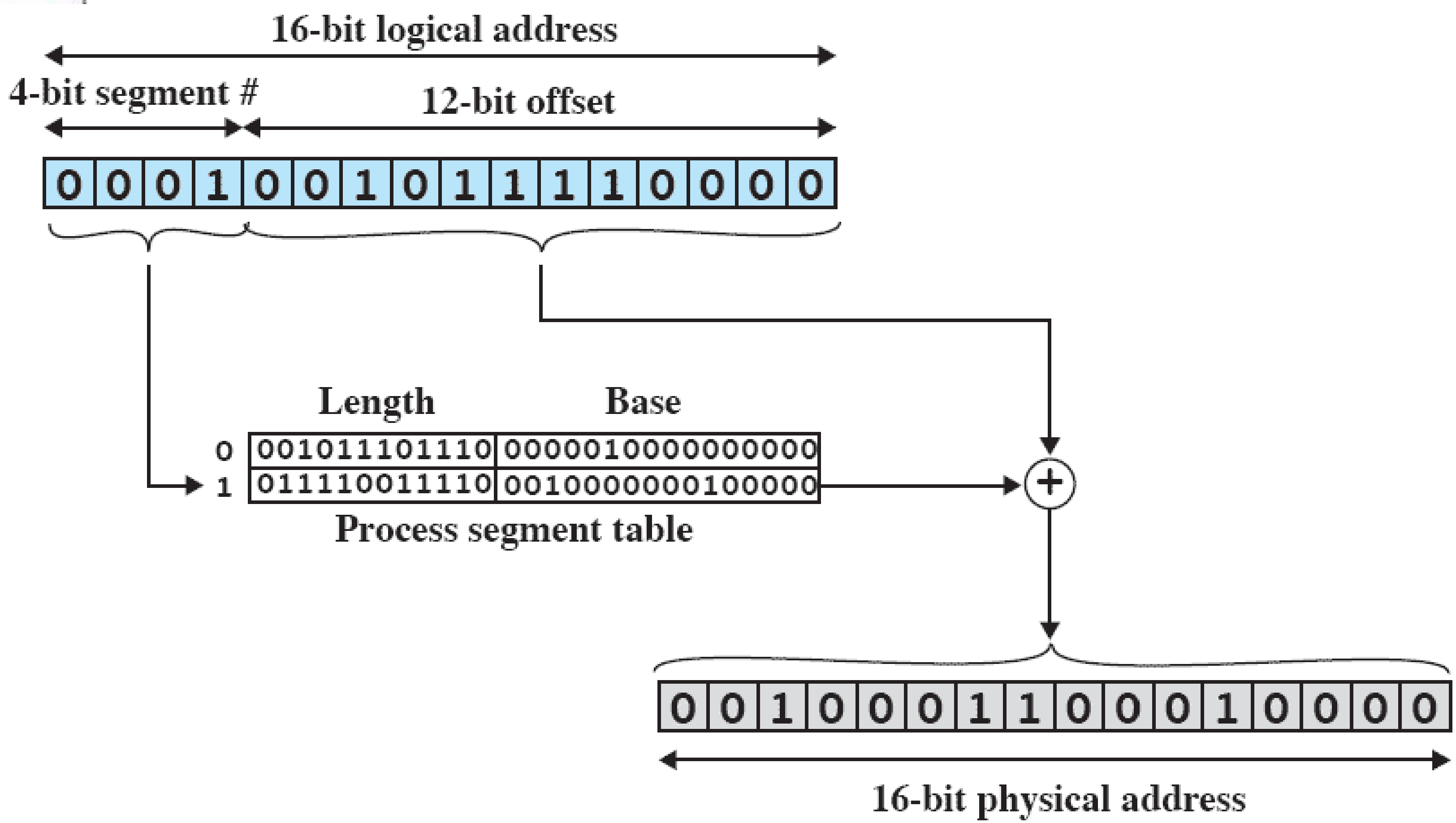
- If the offset is not within the limit, trap to the operating system.(logical address beyond end of segment).




- Segmentation Example:



# Segmentation




(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation

# Logical Addresses

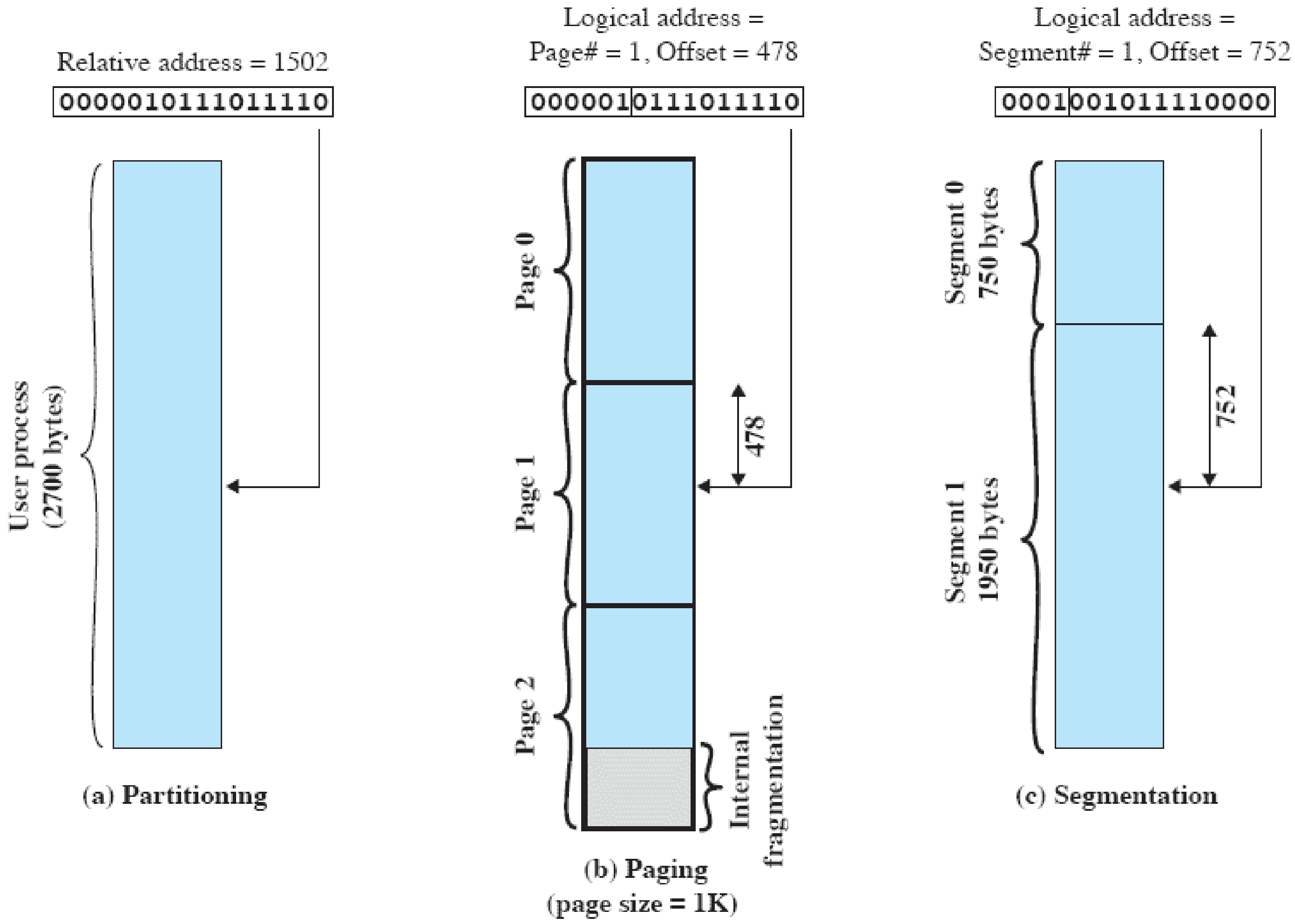



Figure 7.11 Logical Addresses

# Advantages and Disadvantages

- Advantages:
  - Each segment can be
    - located independently
    - separately protected
    - grow independently
  - Segments can be shared between processes.
  - Eliminate the fragmentation.

- Disadvantages:
  - Allocation algorithms as for memory partitions
  - External fragmentation, back to compaction problem.
    - Solution: combine segmentation and paging.

# Segmentation vs. Paging

- Segment is good logical unit of information
  - sharing, protection
- Segmentation
  - Break process up into logical segments
- Segments are generally of different sizes

- Page is good physical unit of information
  - simple memory management
- Paging
  - Break process up into physical pages
  - Pages are the same size

| Paging | Segmentation |
| --- | --- |
| The main memory partitioned into frames or blocks. | The main memory partitioned into segments. |
| The logical address space divided into pages by compiler or MMU | The logical address space divided into segment specified by the programmer. |
| Suffers from internal fragmentation or page break | Suffers from external fragmentation. |
| Operating system maintains the page table. | Operating system maintains the segment table. |
| Does not supports the user view of memory | Supports the user view of memory |
| Processor uses the page number and displacement to calculate absolute address. | Processor uses the segment number and displacement to calculate absolute address. |
| Multilevel paging is possible | Multilevel segmentation is possible. But no use. |

| Consideration | Paging | Segmentation |
|---|---|---|
| Need the programmer be aware that this technique is being used? | No | Yes |
| How many linear address spaces are there? | 1 | Many |
| Can the total address space exceed the size of physical memory? | Yes | Yes |
| Can procedures and data be distinguished and separately protected? | No | Yes |
| Can tables whose size fluctuates be accommodated easily? | No | Yes |
| Is sharing of procedures between users facilitated? | No | Yes |
| Why was this technique invented? | To get a large linear address space without having to buy more physical memory | To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection |