Operating Systems: Internals and Design Principles

Chapter 7 Memory Management

Eighth Edition

William Stallings

Uni-Processing to Multi-Processing

Taking the step into multi-processing adds many challenges

- Sharing available resources across processes:
  - How to efficiently use the available resources?
  - How to allocate resources on the fly?
  - How to avoid deadlock and starvation?
- How to keep processes from interfering with one-another?
  - I/O activities
  - CPU resources
  - Memory

Multi-Processing and Memory Management

- Efficient allocation of memory to processes
- OS/Hardware support for quick access to memory resources

Memory Management Terminology

- Frame:
- Page:
- Segment:

Memory Management Terminology

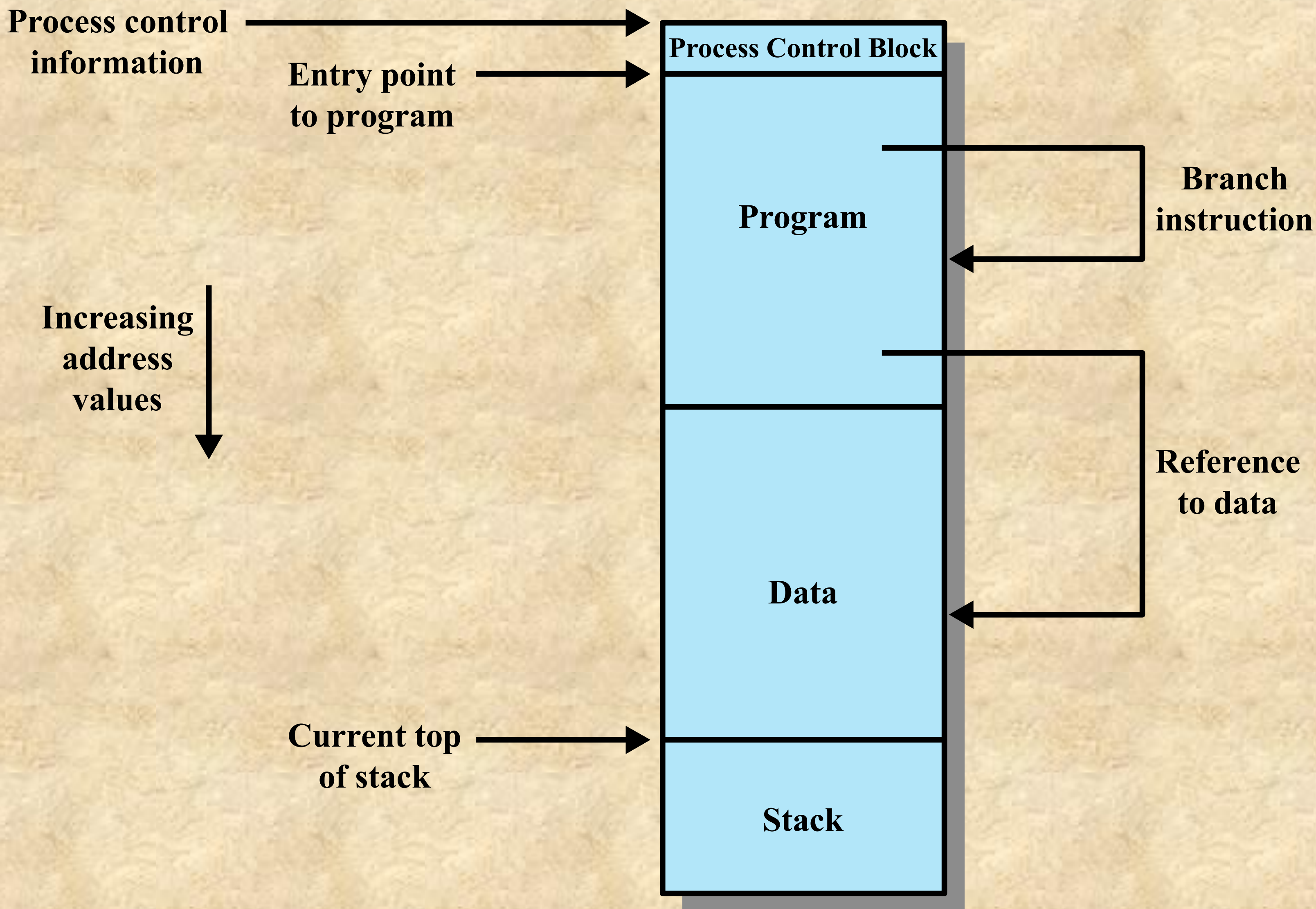
- Frame: a fixed-length block in main memory
- Page: a fixed-length block stored in secondary memory
- Segment: a non-fixed length block of memory

Memory Management Issues to be Addressed

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

Relocation

- Want to support many processes in main memory at once
  - This set may change over very short periods of time
- No way to guarantee that a process will be placed in the same region of physical memory from one instant to the next
- Relocation: OS and hardware work together to support placing a process at any location in main memory
  - Challenge: how to make this invisible to the process?


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

Protection

The relocation and protection mechanisms work together and require hardware support

- Processes need to have permission to reference memory locations for reading or writing purposes
- Memory references generated by a process must be checked for safety at run time

Sharing

- Advantageous to allow each process access to a single copy of the program rather than for each to have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation must also support sharing capabilities

Logical Organization

- Physical memory is organized as linear. We would like to preserve this abstraction at the program level
- But: programs are generally partitioned into modules
  - Example: a source file produces a single code module that can be compiled independently of the other source files
- Would like to preserve this notion of modules:
  - Different modules will have different lengths
  - Protection and sharing can be done at a module level
- We refer to this as ***segmentation***

Physical Organization

- Multi-layer organization to memory:
  - Primary memory is fast, but expensive
  - Secondary memories are slower, but less expensive
- Memory management is the process of allocating processes to primary and secondary memory
  - This must be coordinated with the process scheduler

Physical Organization

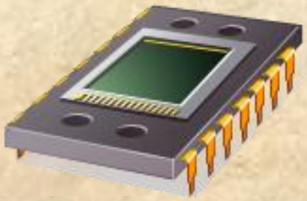
When main memory is too small to fit a program:

- User managed (overlays):
  - Program is split into multiple pieces, only one of which is in memory at once
  - Program triggers the copying of the next piece from secondary memory into main memory when it is needed
  - In modern OSes, we do not trust a program to do this level of memory management
- System managed: ***virtual memory*** (next lecture)

Memory Partitioning

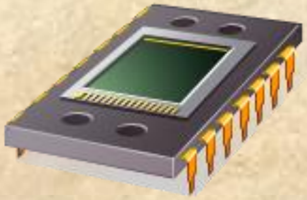
Memory Partitioning:

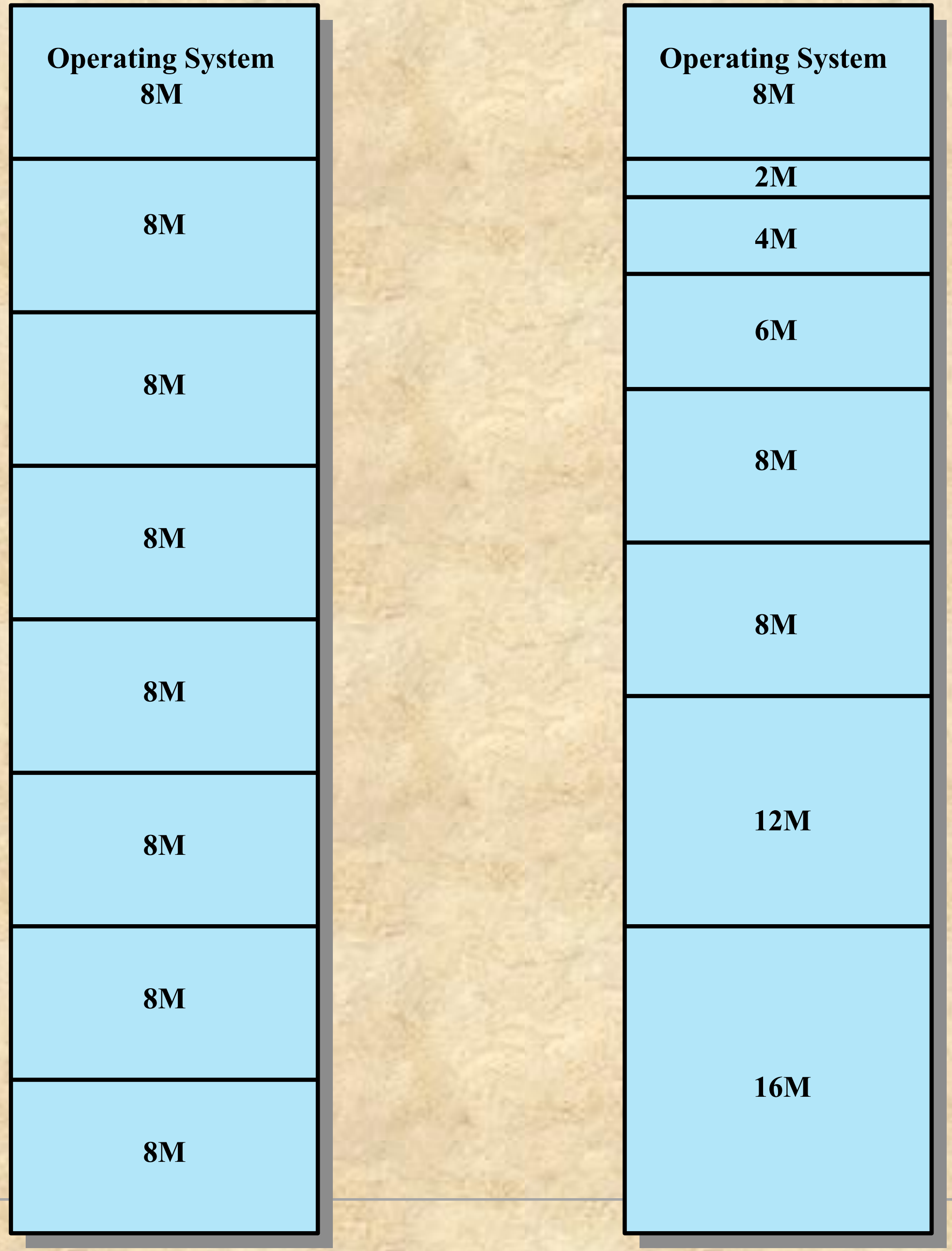
- Our first attempt in early OSes
- A process is brought into main memory as a monolithic unit
- Many different techniques for implementation



Fixed Partitioning

- Physical memory is permanently cut fixed-sized partitions
- Processes are allocated to free partitions when they are ready to execute





Operating System
8M
8M
8M
8M
8M
8M
8M
8M
Operating System
8M
2M
4M
6M
8M
8M
12M
16M

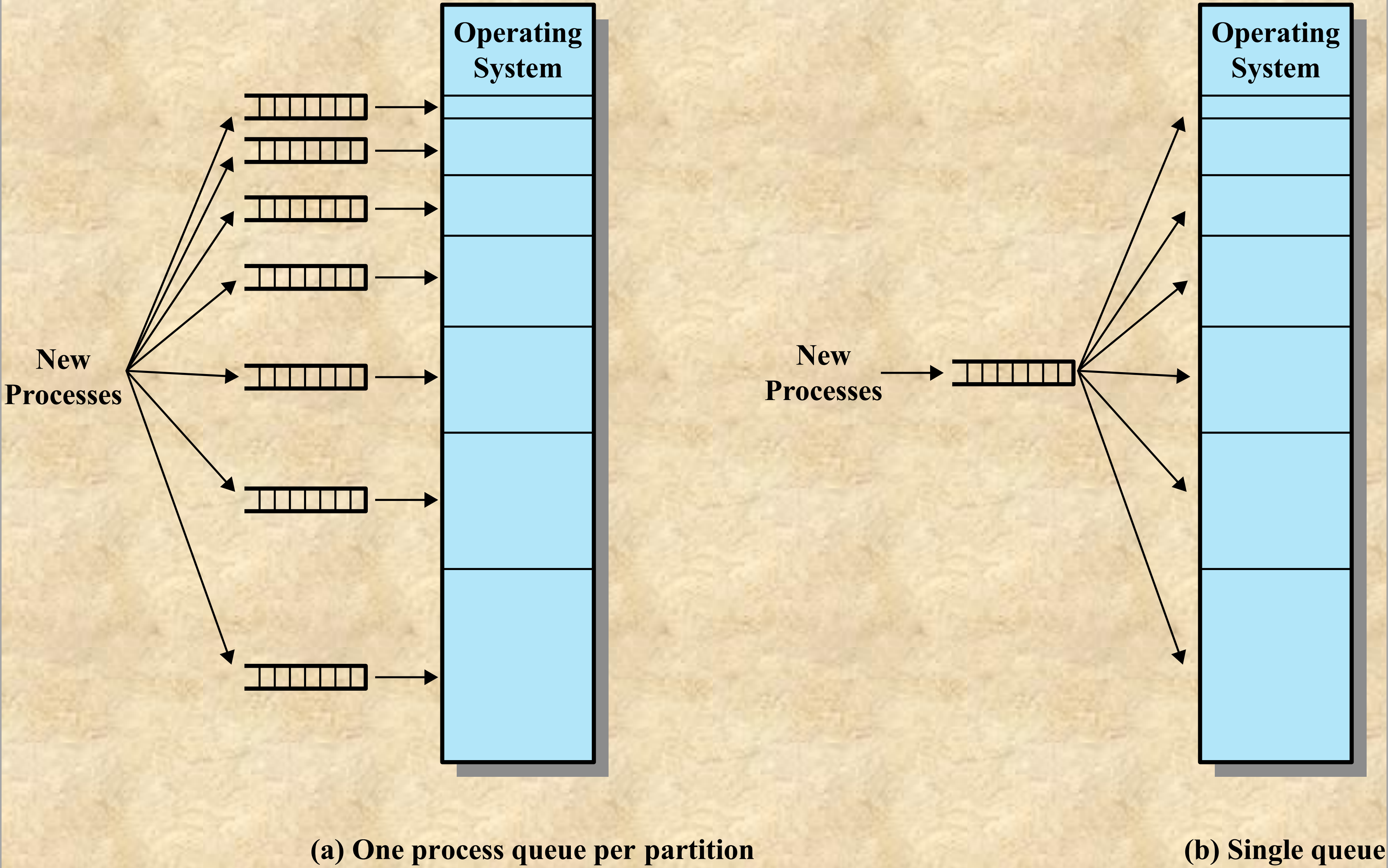


Figure 7.3 Memory Assignment for Fixed Partitioning

Fixed Partitioning: Disadvantages

- A program may be too big to fit in a partition
  - Program must drop back to using overlays
- Main memory utilization is inefficient
  - Any program, regardless of size, occupies an entire partition
  - ***Internal fragmentation:*** wasted space due to the block of data loaded being smaller than the partition

Fixed Partitioning: Disadvantages

- The number of partitions specified at system generation time
  - Limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently
  - And there are typically many of these processes

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as it requires
- This technique was used by IBM's mainframe operating system, OS/MVT

(a)

Figure 7.4 The Effect of Dynamic Partitioning

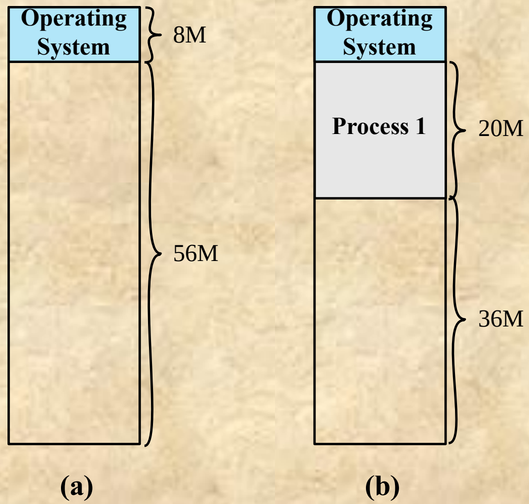


Figure 7.4 The Effect of Dynamic Partitioning

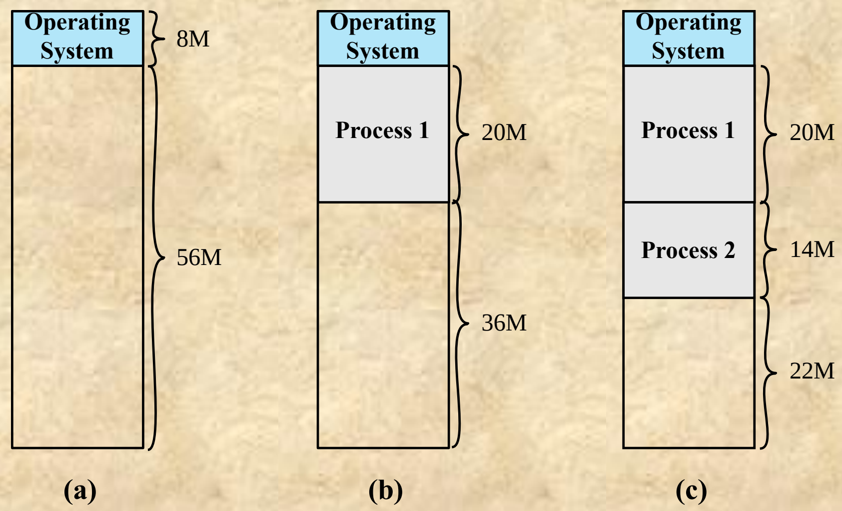


Figure 7.4 The Effect of Dynamic Partitioning

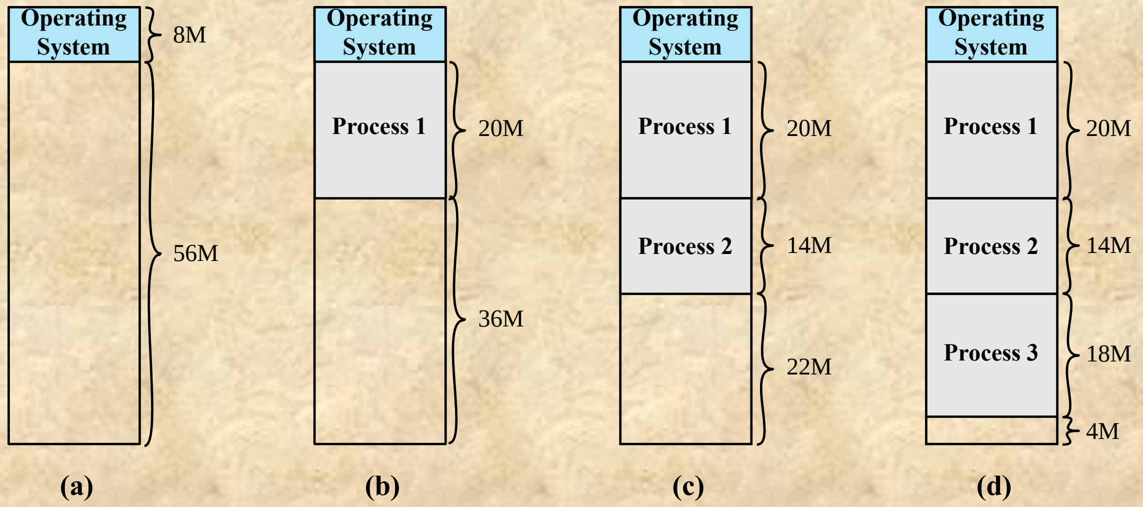


Figure 7.4 The Effect of Dynamic Partitioning

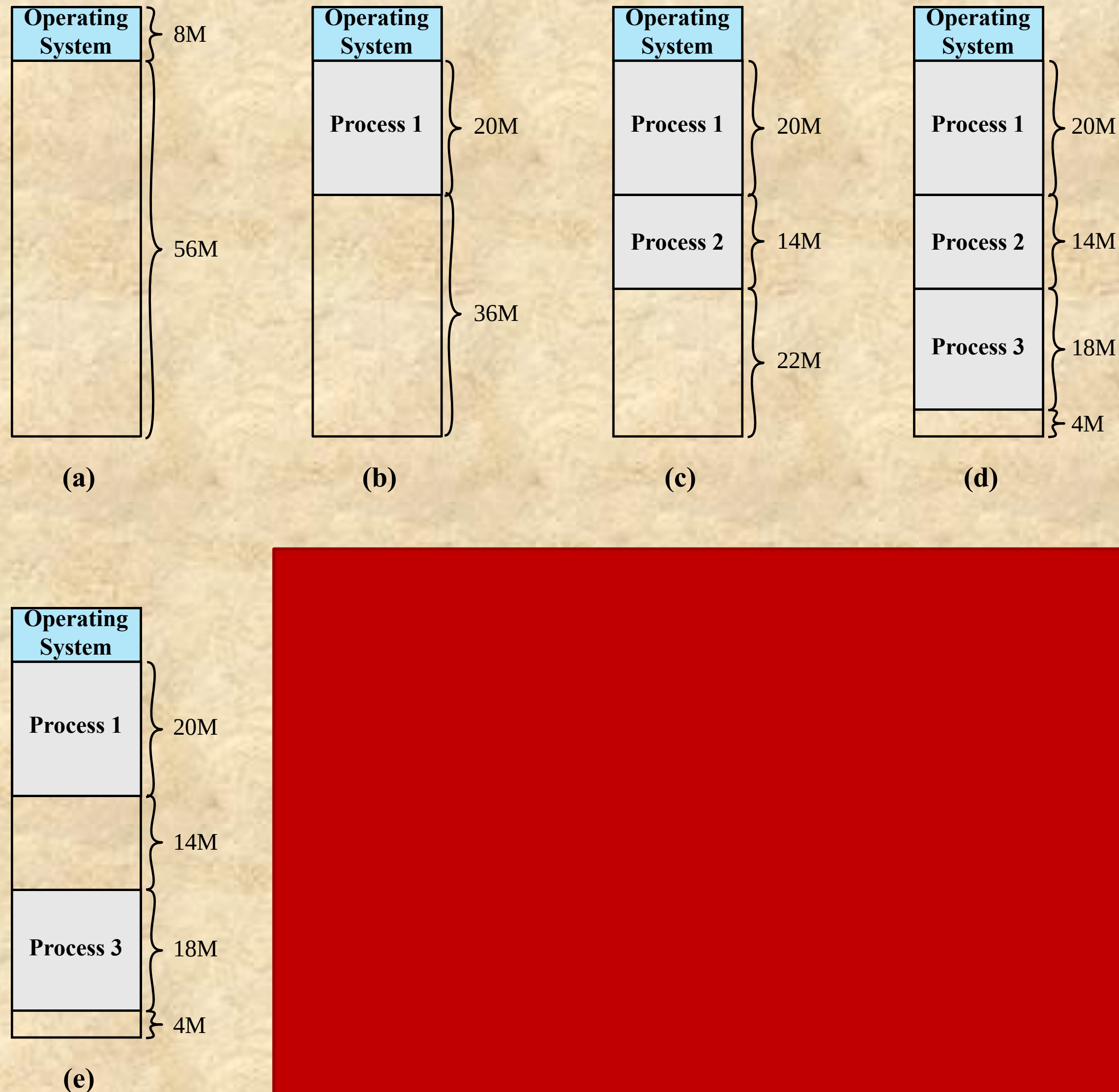


Figure 7.4 The Effect of Dynamic Partitioning

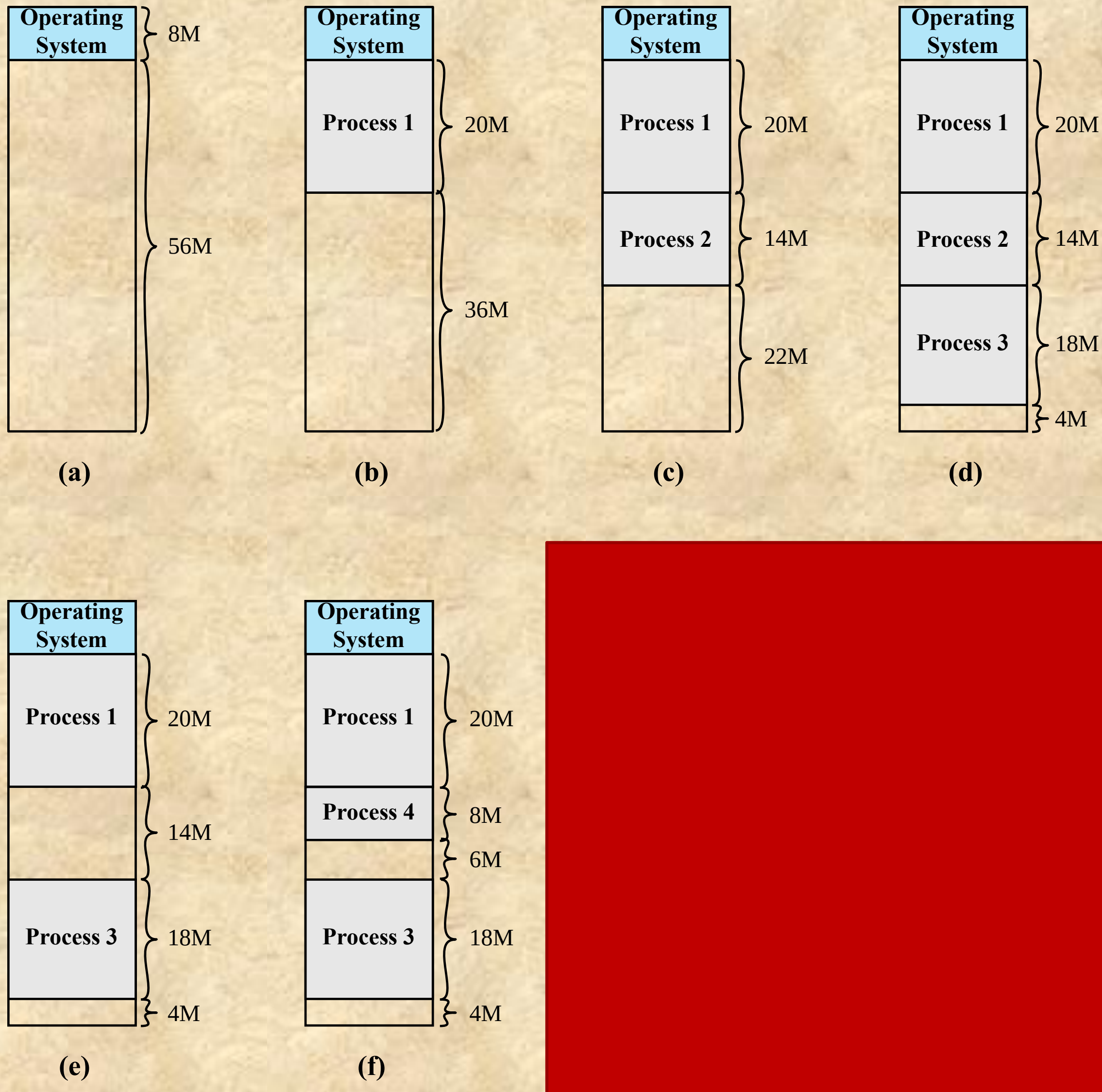


Figure 7.4 The Effect of Dynamic Partitioning

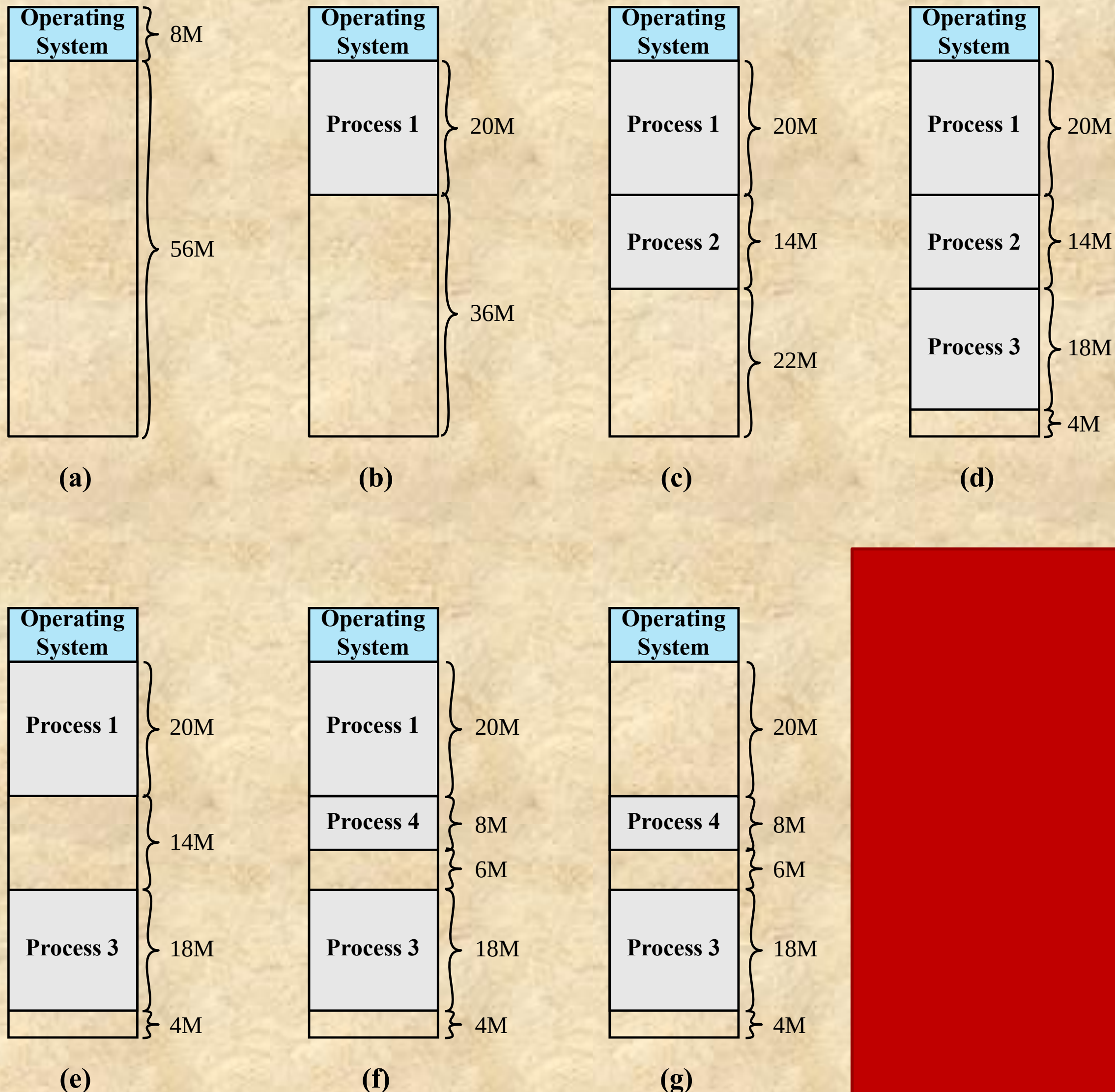


Figure 7.4 The Effect of Dynamic Partitioning

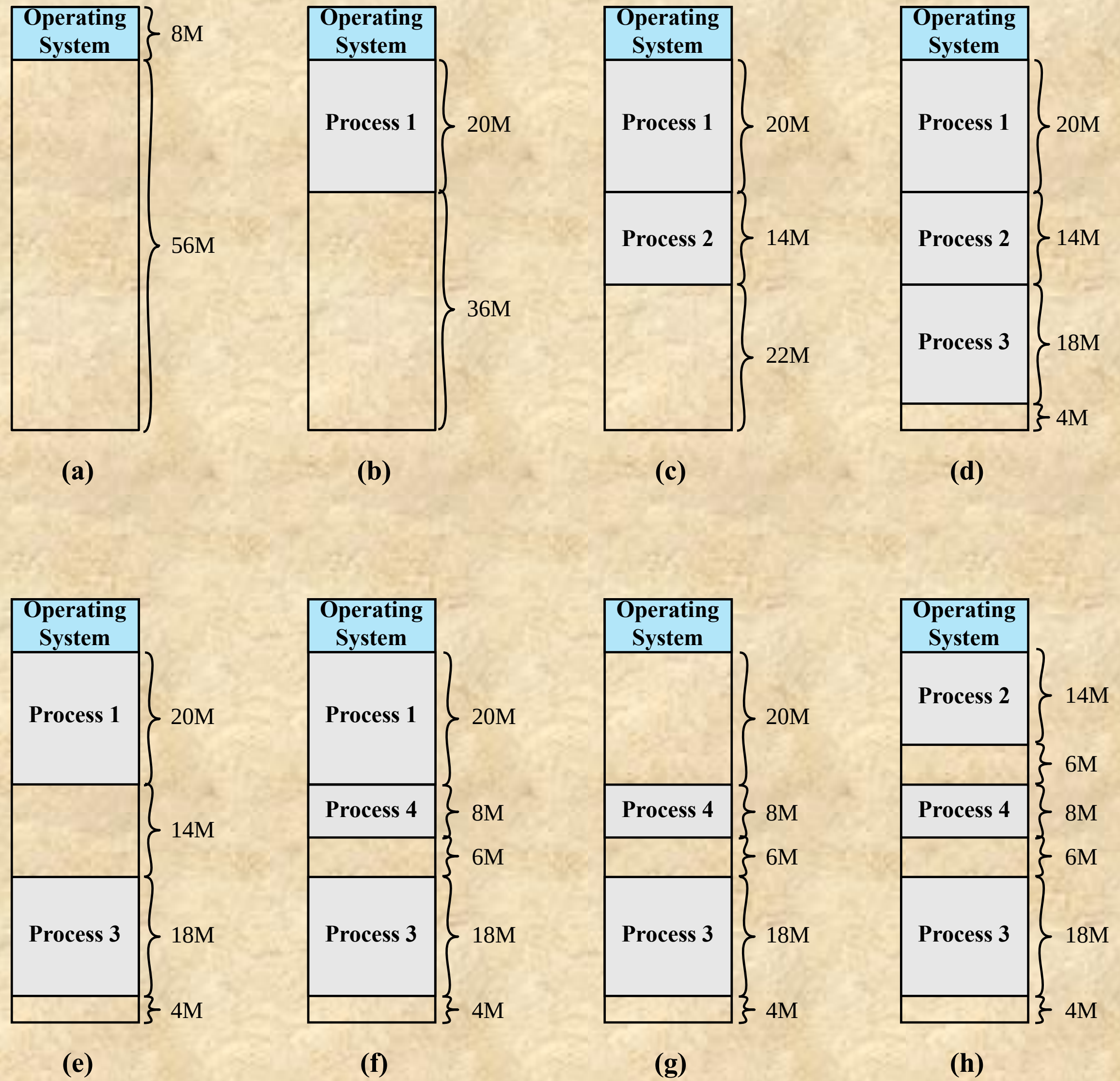


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning: Challenges

External Fragmentation

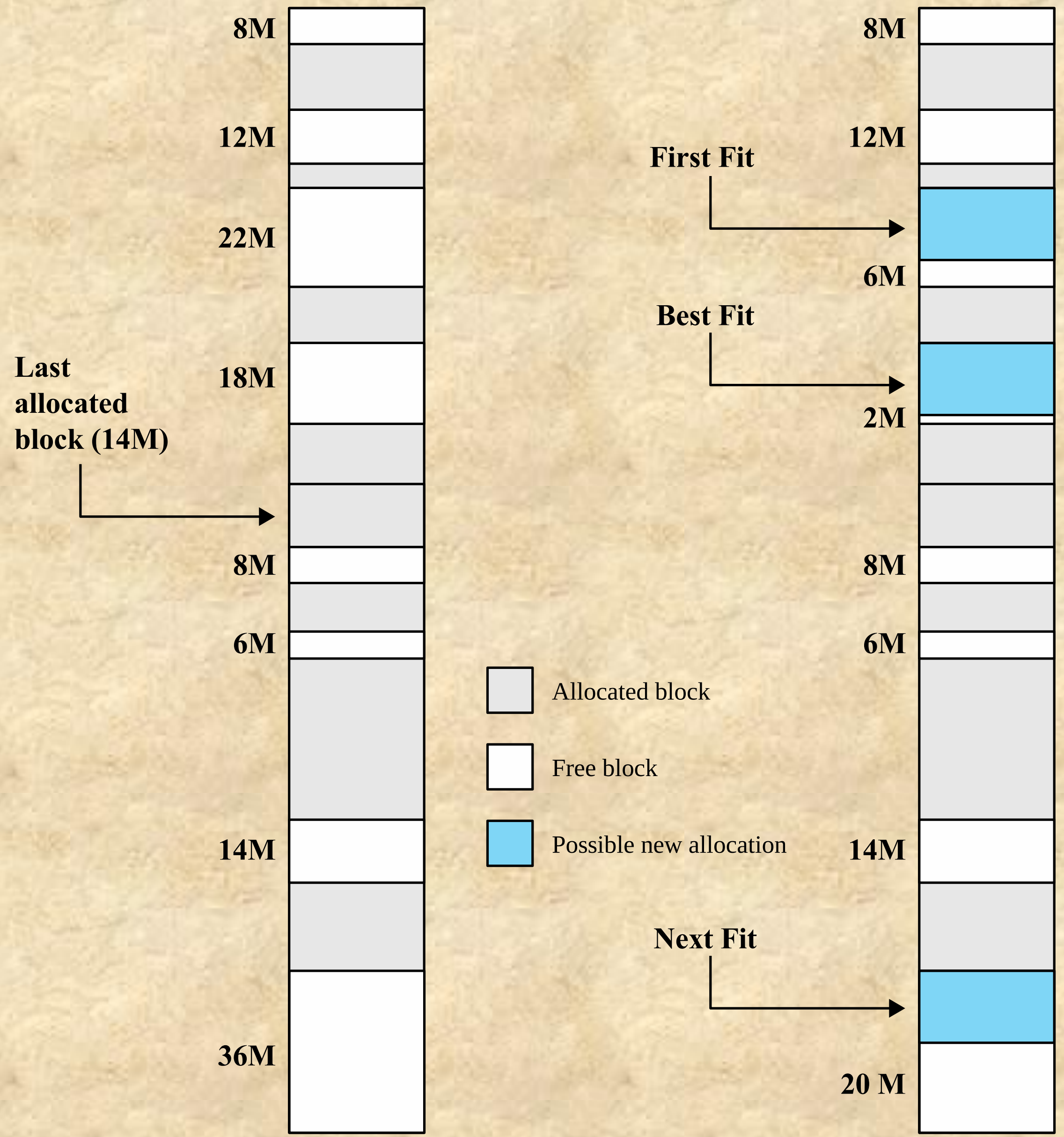
- Memory becomes more and more fragmented
- As a result, memory utilization declines

A Fix: Compaction

- OS shifts processes so that the group is contiguous
- Free memory is together in one block
- But: time consuming and wastes CPU time

Dynamic Allocation: Placement Algorithms

Best-fit	First-fit	Next-fit
• Chooses the block that is closest in size to the request	• Begins to scan memory from the beginning and chooses the first available block that is large enough	• Begins to scan memory from the location of the last placement and chooses the next available block that is large enough



(a) Before
(b) After

Dynamic Allocation: Placement Algorithms

- In practice: First Fit tends to perform best
- But: all methods involve a lot of overhead to compute where to place a process
- And: we have the overhead of compaction

Moving Beyond Simple Allocation Schemes: The Buddy System

- A synthesis of the fixed and dynamic partitioning schemes
- Space available for allocation is initially treated as a single, large block
- Memory blocks are available of size 2^K *words,* $L \leq K \leq U$, where
  - 2^L = smallest size block that is allocated
  - 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation

Buddy System Algorithm

- New process allocation:
  - Find the smallest available block that fits the new process
  - Cut this block by factors of 2 until it just fits the process, leaving other parts as available for other processes (these pairs are the ***buddies***)
- Deallocation:
  - If the deallocated block has an unallocated buddy, then merge back together
  - Repeat recursively

1 Mbyte block

1 M

Request 100 K

Request 240 K

Request 64 K

Request 256 K

Release B

Release A

Request 75 K

Release C

Release E

Release D

Figure 7.6 Example of Buddy System

1 Mbyte block	1 M			
Request 100 K	A = 128K	128K	256K	512K
Request 240 K				
Request 64 K				
Request 256 K				
Release B				
Release A				
Request 75 K				
Release C				
Release E				
Release D				

Figure 7.6 Example of Buddy System

1 Mbyte block	1 M			
Request 100 K	A = 128K	128K	256K	512K
Request 240 K	A = 128K	128K	B = 256K	512K
Request 64 K				
Request 256 K				
Release B				
Release A				
Request 75 K				
Release C				
Release E				
Release D				

Figure 7.6 Example of Buddy System

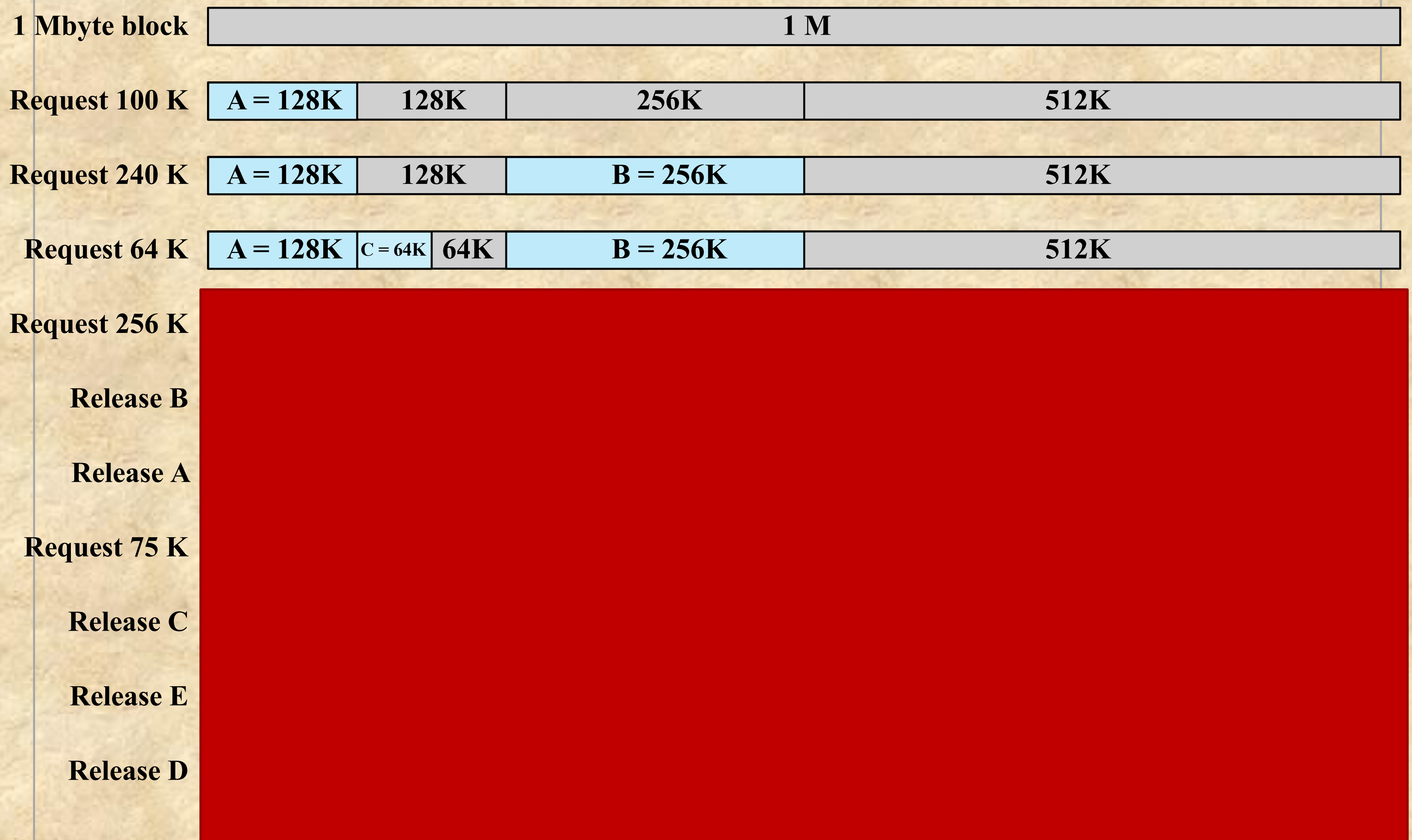


Figure 7.6 Example of Buddy System

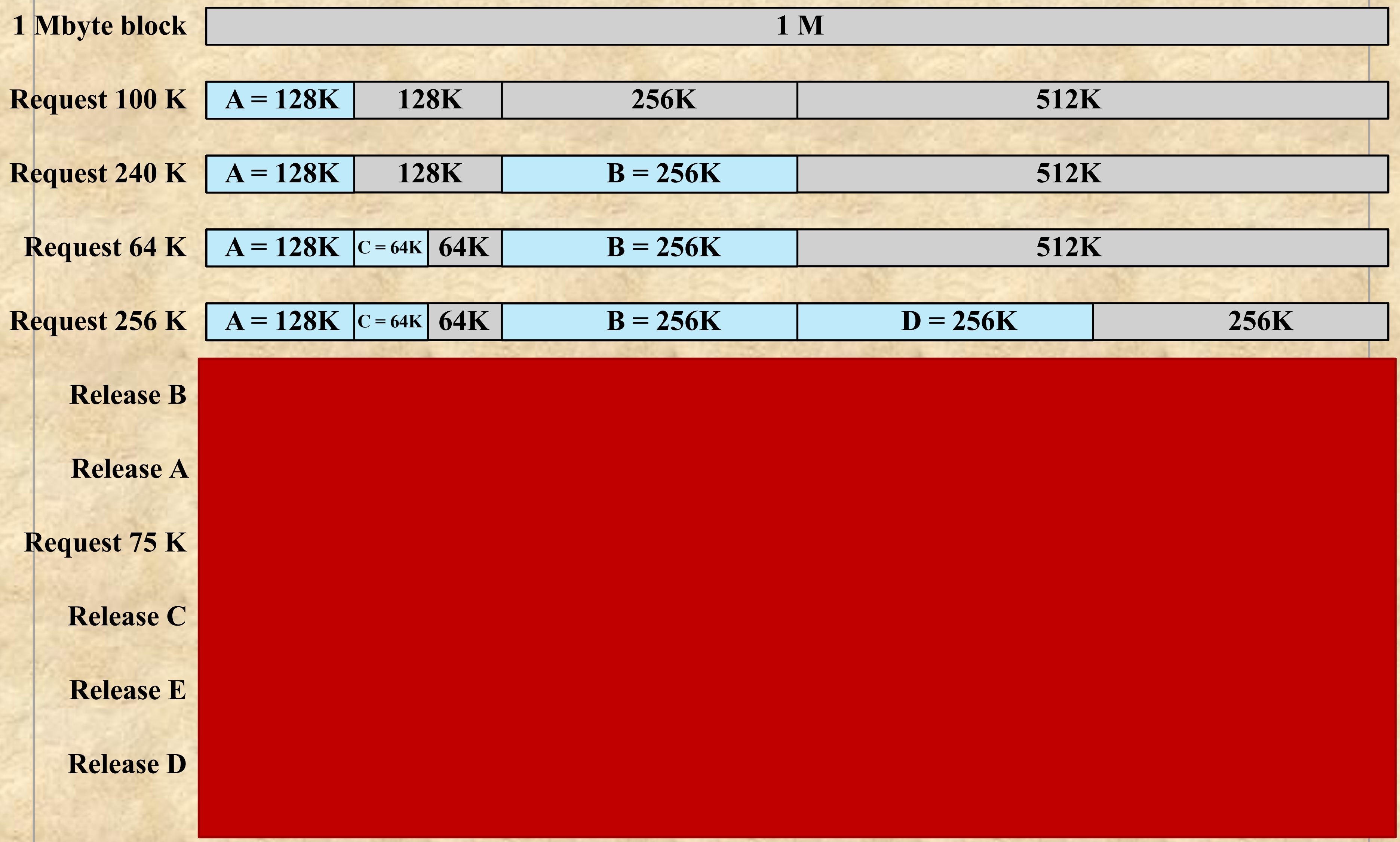


Figure 7.6 Example of Buddy System

1 Mbyte block	1 M						
Request 100 K	A = 128K	128K		256K	512K		
Request 240 K	A = 128K	128K		B = 256K	512K		
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K		
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K	
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K	
Release A							
Request 75 K							
Release C							
Release E							
Release D							

Figure 7.6 Example of Buddy System

1 Mbyte block	1 M					
Request 100 K	A = 128K	128K		256K	512K	
Request 240 K	A = 128K	128K		B = 256K	512K	
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K	
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75 K						
Release C						
Release E						
Release D						

Figure 7.6 Example of Buddy System

1 Mbyte block	1 M					
Request 100 K	A = 128K	128K		256K	512K	
Request 240 K	A = 128K	128K		B = 256K	512K	
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K	
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75 K	E = 128K	C = 64K	64K	256K	D = 256K	256K
Release C						
Release E						
Release D						

Figure 7.6 Example of Buddy System

1 Mbyte block	1 M					
Request 100 K	A = 128K	128K		256K	512K	
Request 240 K	A = 128K	128K		B = 256K	512K	
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K	
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75 K	E = 128K	C = 64K	64K	256K	D = 256K	256K
Release C	E = 128K	128K		256K	D = 256K	256K
Release E						
Release D						

Figure 7.6 Example of Buddy System

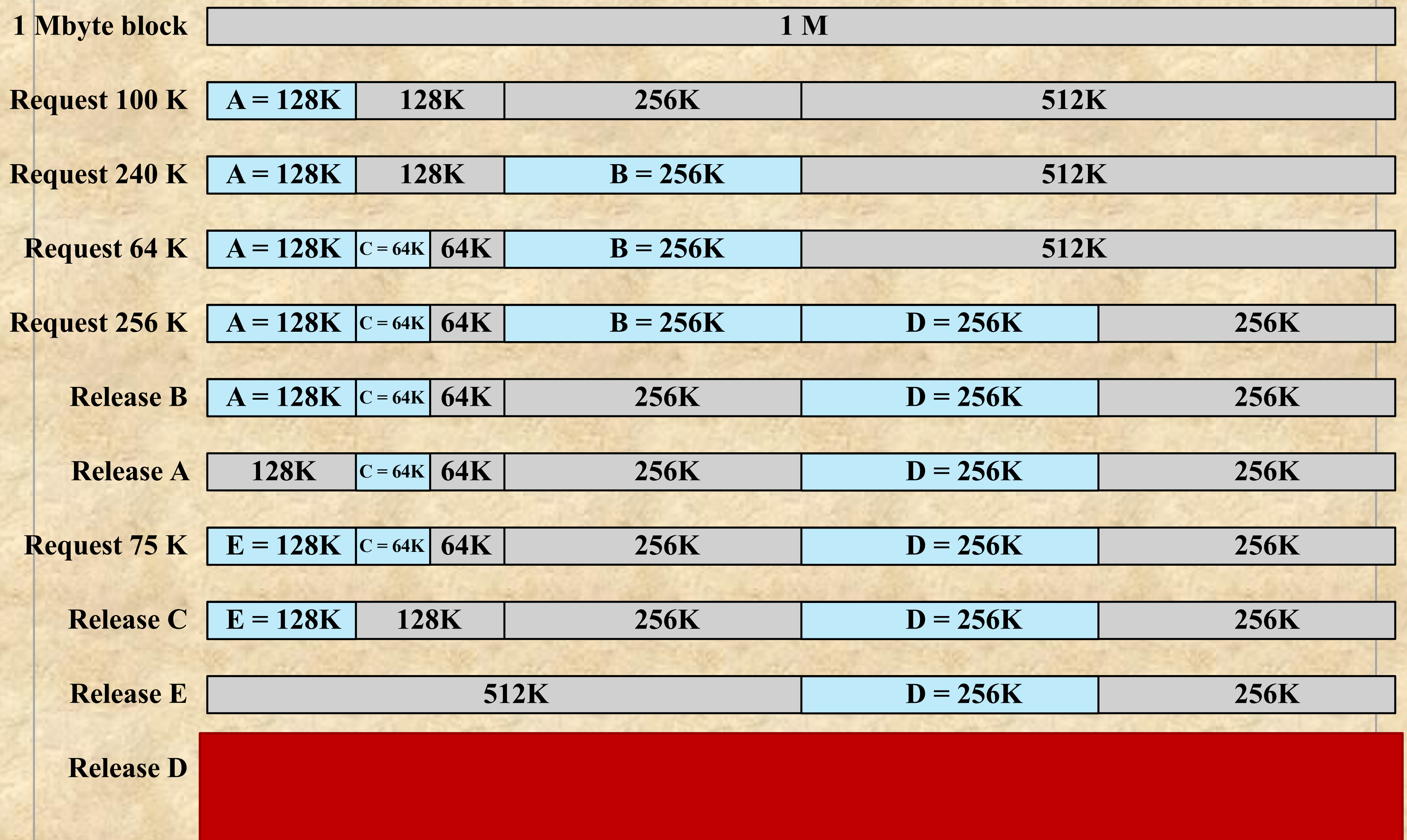


Figure 7.6 Example of Buddy System

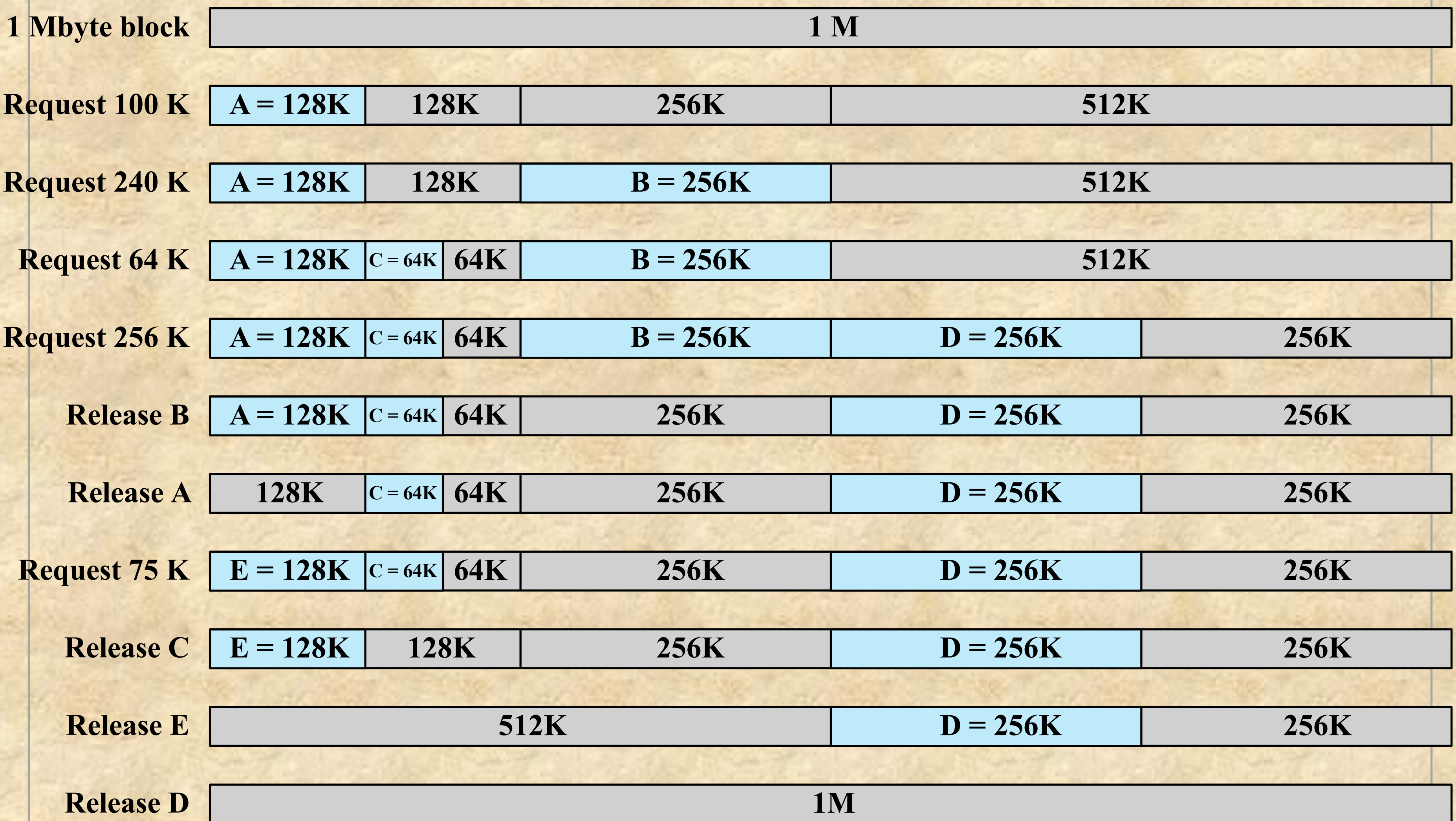
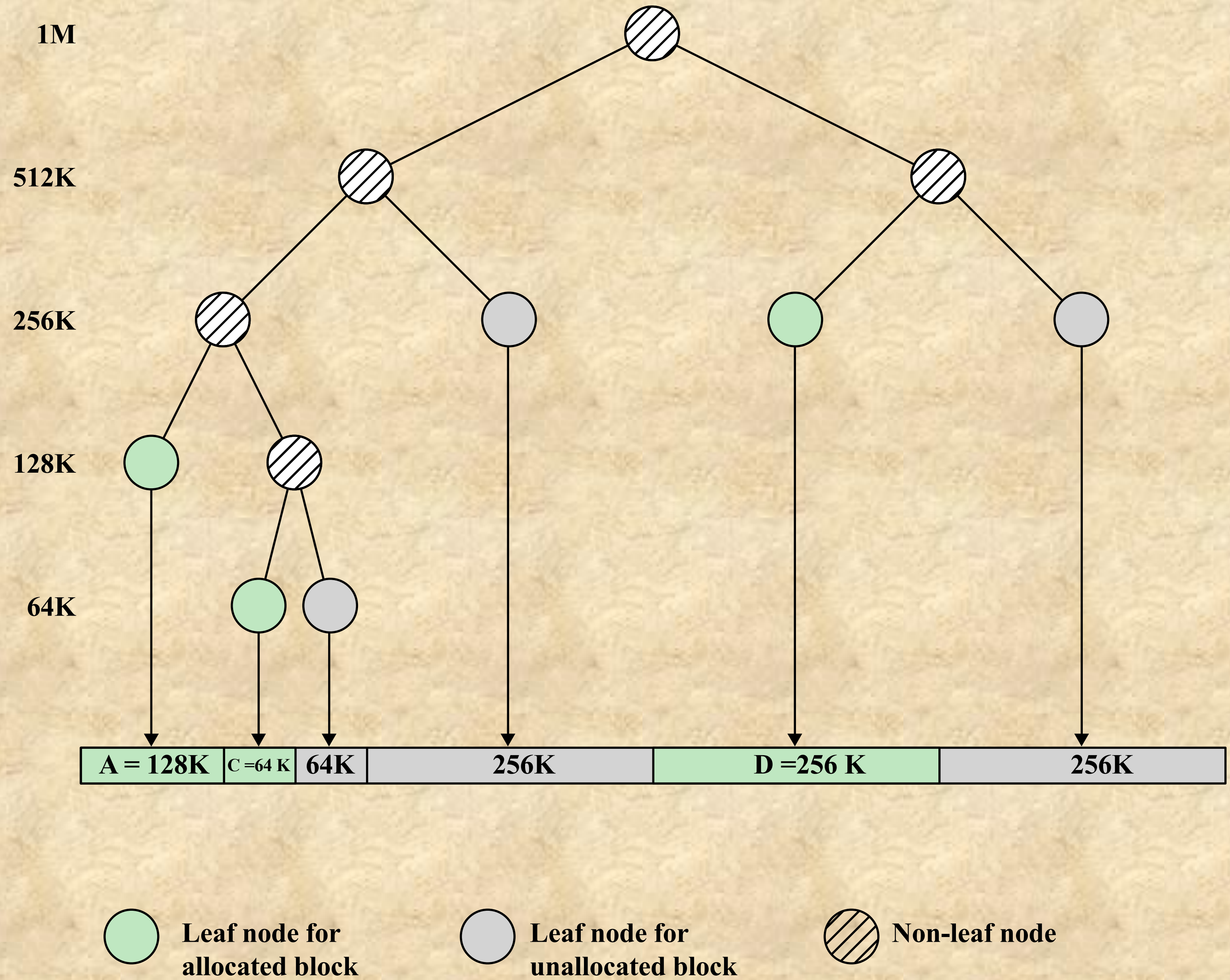


Figure 7.6 Example of Buddy System


1M
512K
256K
128K
64K
A = 128K
C =64 K
64K
256K
D =256 K
256K
Leaf node for allocated block
Leaf node for unallocated block
Non-leaf node

Addresses

Logical

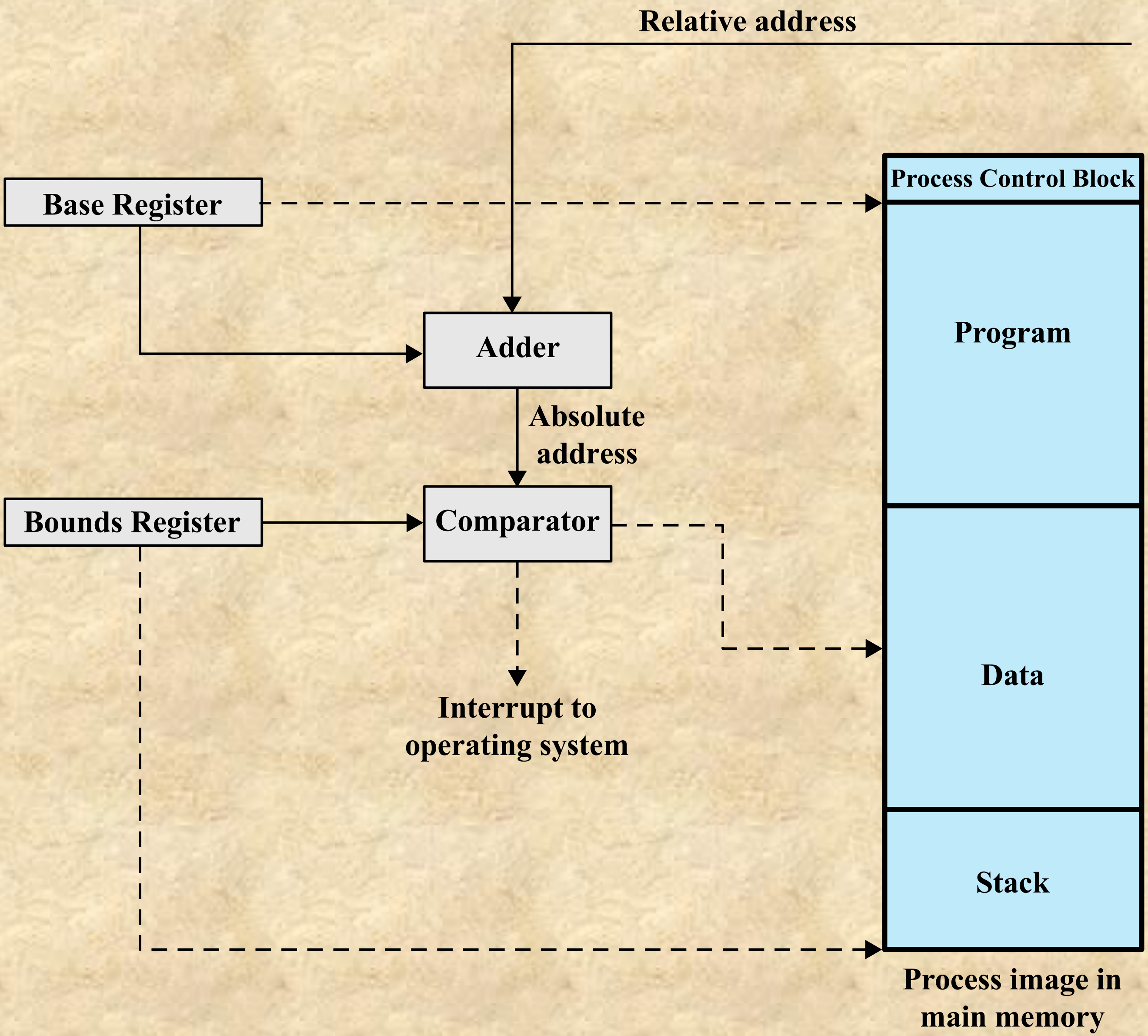
- Reference to a memory location independent of the current assignment of data to memory

Relative (one type of Logical)

- Address is expressed as a location relative to some known point

Physical or Absolute

- Actual location in main memory

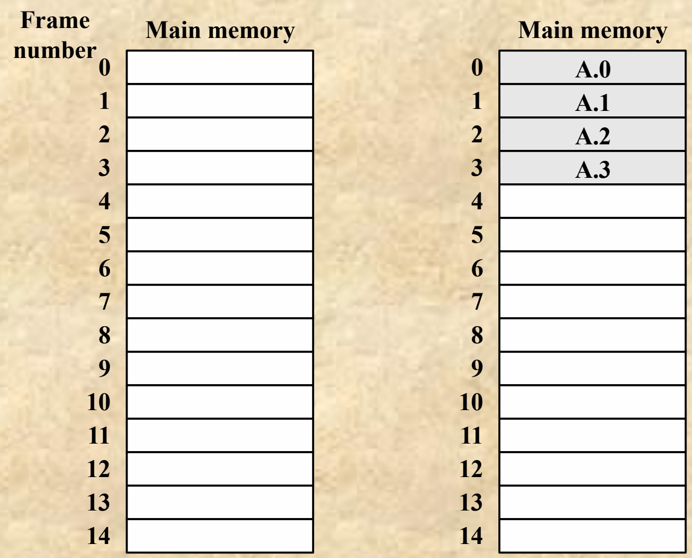

Relative address
Base Register
Adder
Absolute address
Bounds Register
Comparator
Interrupt to operating system
Process Control Block
Program
Data
Stack
Process image in main memory

Beyond Partitioning: Paging

- Partition memory into equal fixed-size chunks that are relatively small
- Process is also divided into small fixed-size chunks of the same size

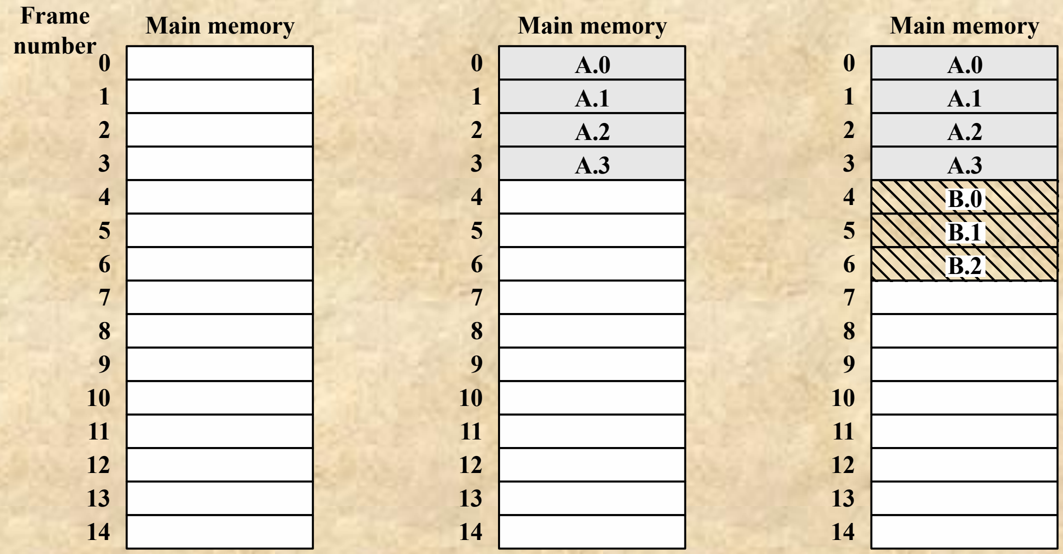
Pages	Frames
• Chunks of a process	• Available chunks of memory

(a) Fifteen Available Frames



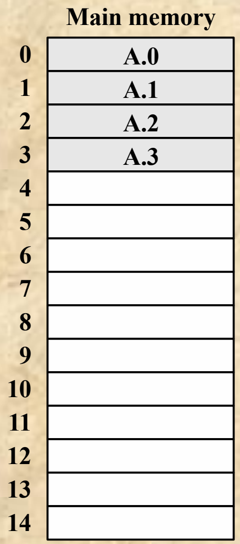
(a) Fifteen Available Frames

(b) Load Process A

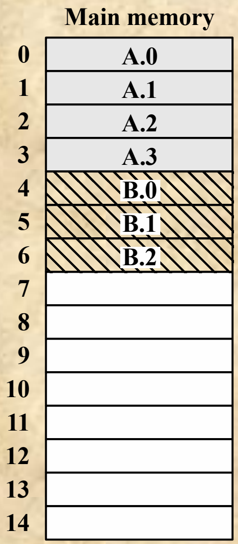


(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

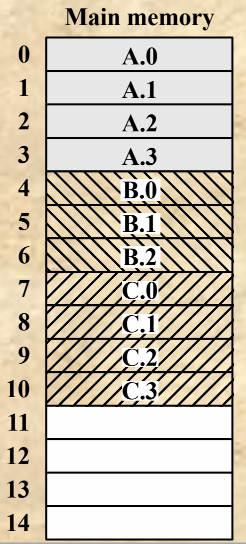
(a) Fifteen Available Frames



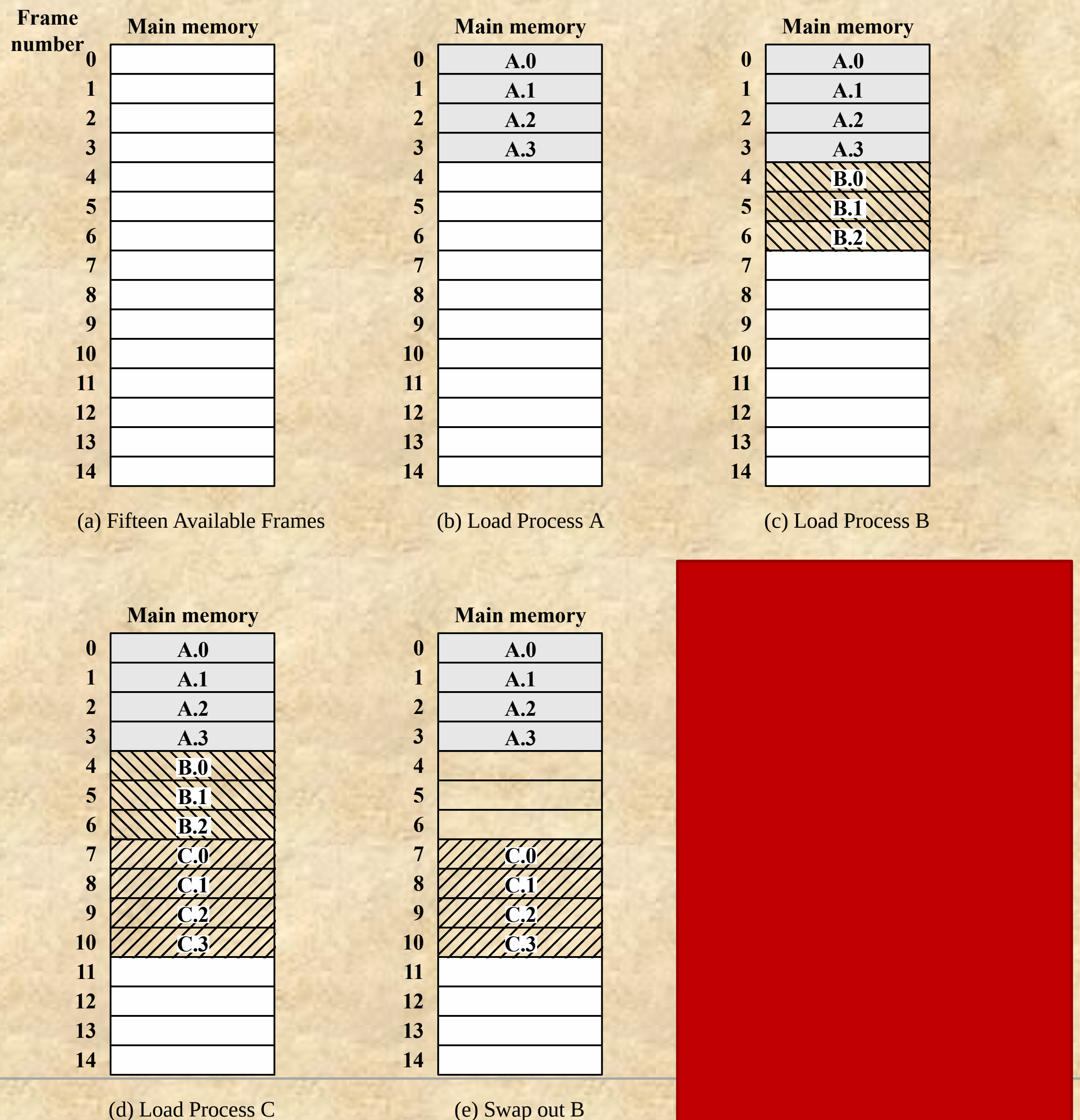
(b) Load Process A



(c) Load Process B



(d) Load Process C



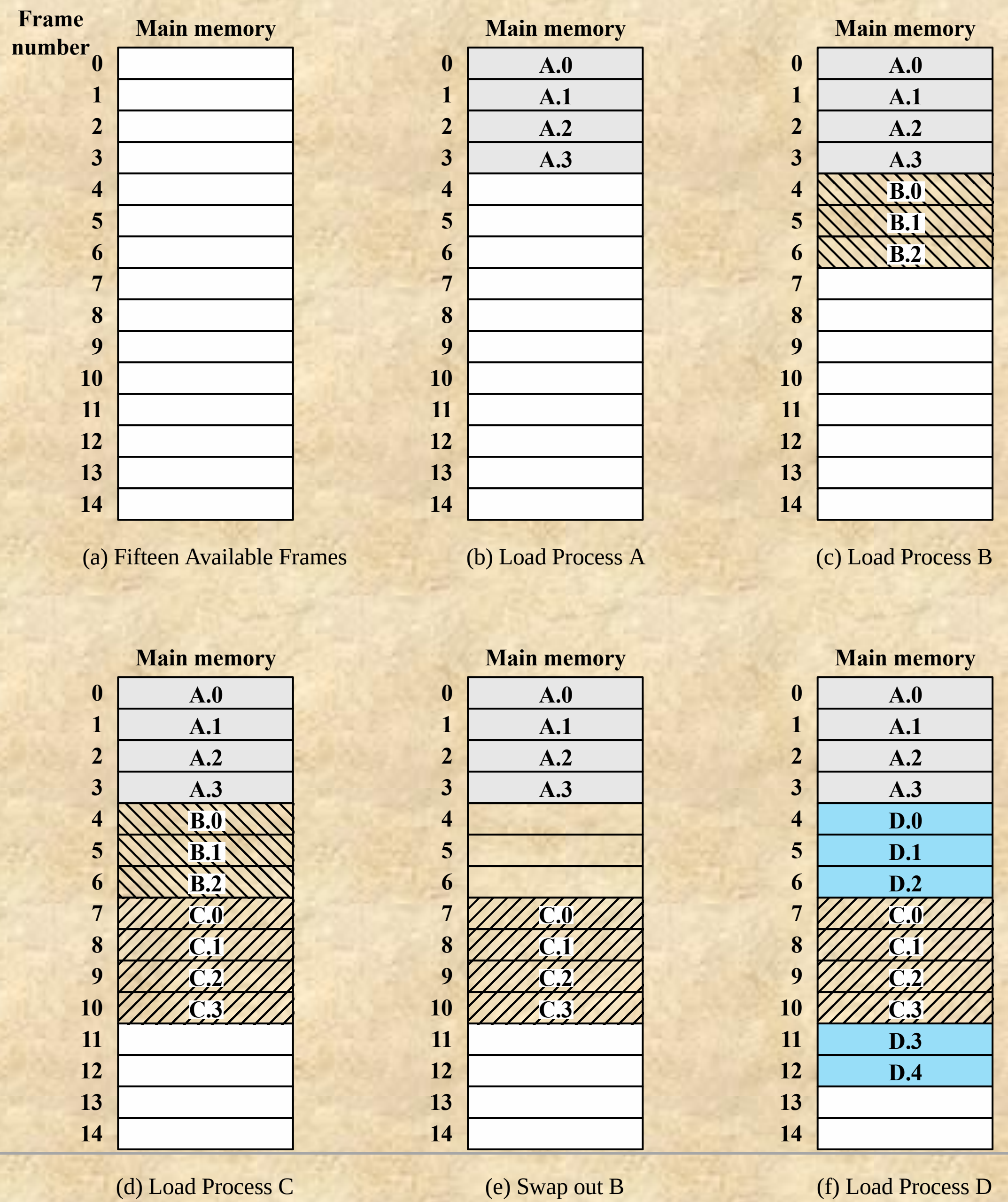
(a) Fifteen Available Frames

(b) Load Process A

(c) Load Process B

(d) Load Process C

(e) Swap out B



Frame number
Main memory
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
(a) Fifteen Available Frames
A.0 A.1 A.2 A.3
(b) Load Process A
A.0 A.1 A.2 A.3 B.0 B.1 B.2
(c) Load Process B
A.0 A.1 A.2 A.3 B.0 B.1 B.2 C.0 C.1 C.2 C.3
(d) Load Process C
A.0 A.1 A.2 A.3 C.0 C.1 C.2 C.3
(e) Swap out B
A.0 A.1 A.2 A.3 D.0 D.1 D.2 C.0 C.1 C.2 C.3 D.3 D.4
(f) Load Process D

Page Table

- Used by processor to produce a physical address from a logical one
- Contains the frame location for each page in the process
- Maintained by the operating system for each process

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

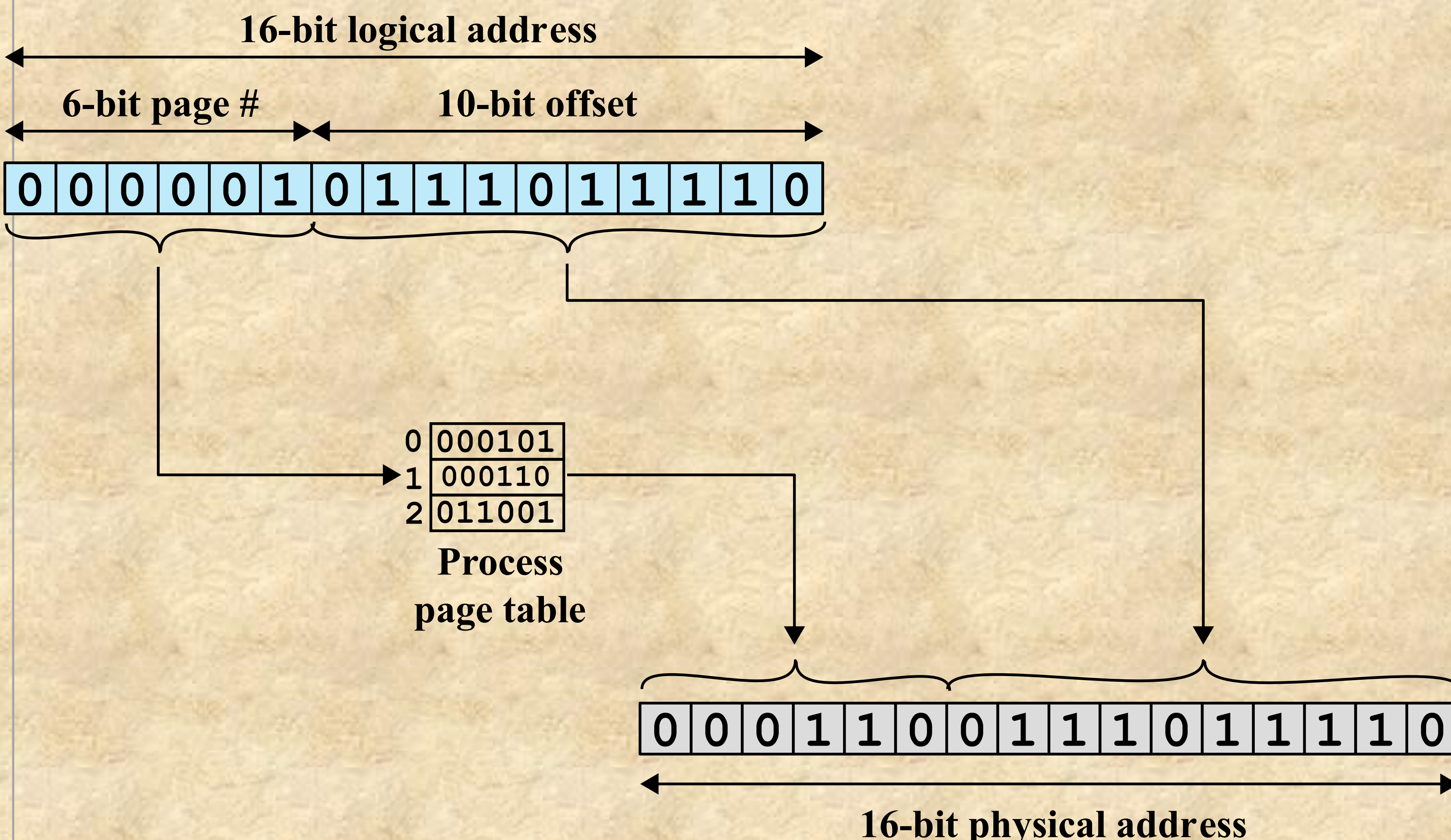
0	4
1	5
2	6
3	11
4	12

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)



(a) Paging

Figure 7.12 Examples of Logical-to-Physical Address Translation

Segmentation

- A program can be subdivided into segments
  - May be of different lengths
  - But, there is a maximum length
- Addressing consists of two parts:
  - segment number
  - an offset
- Similar to dynamic partitioning
- Eliminates internal fragmentation

Segmentation

- Usually visible to the programmer
- Typically, the programmer will assign programs and data to different segments
- Modular programming: the program or data may be further broken down into multiple segments
  - But: the programmer must be aware of the maximum segment size limitation

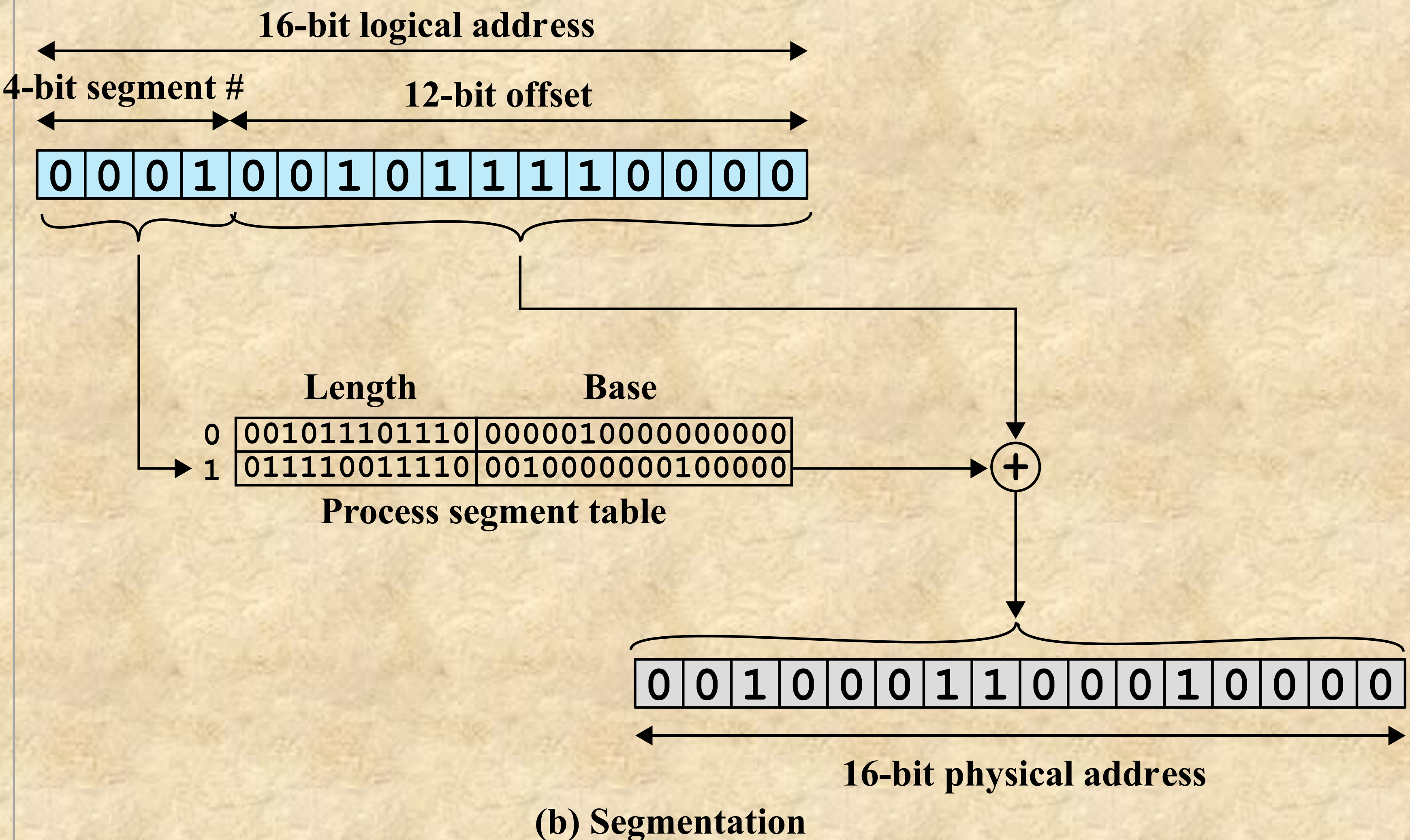
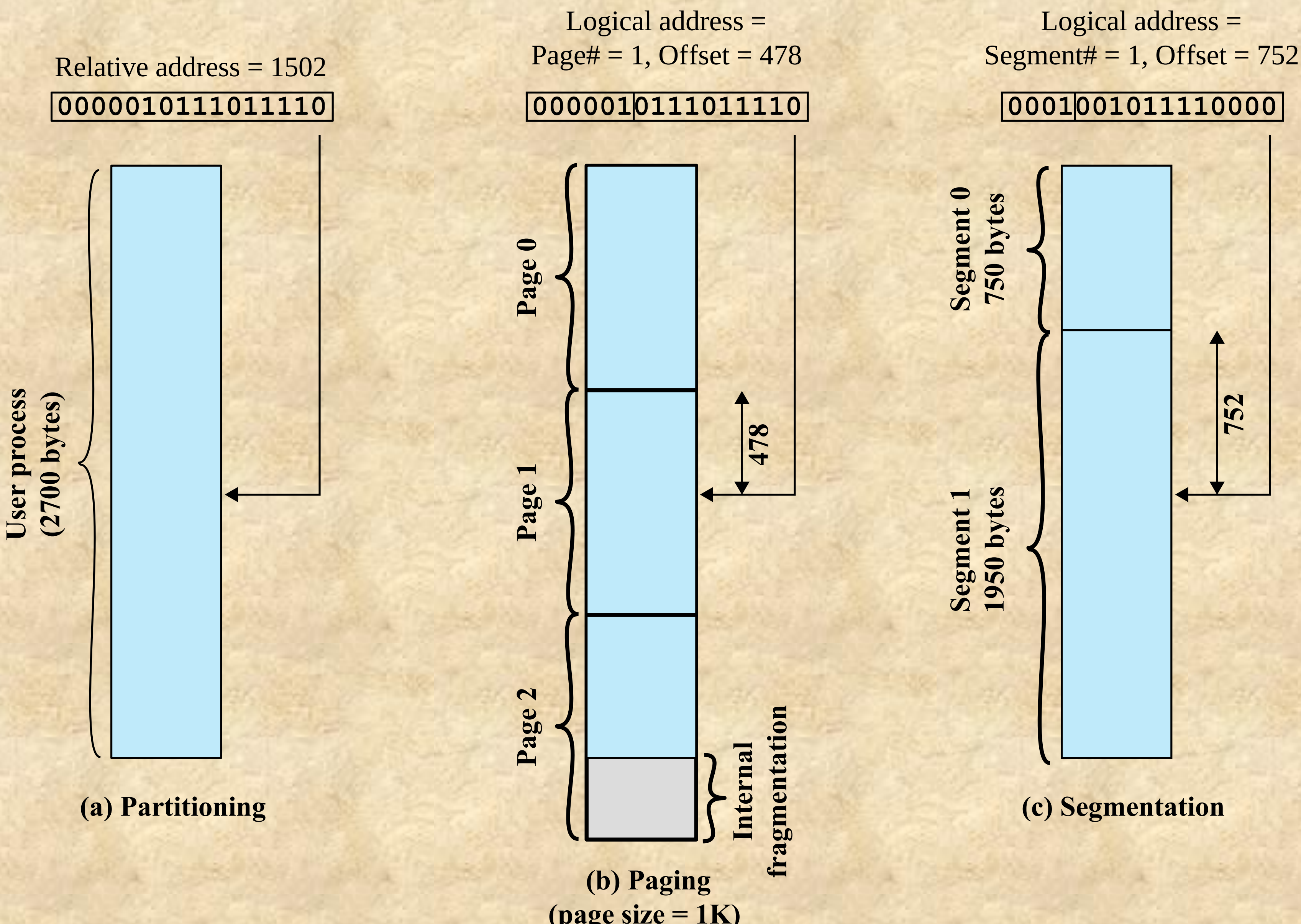


Figure 7.12 Examples of Logical-to-Physical Address Translation

Summary: Segmentation

- Eliminates internal fragmentation
- But: computation of addresses is more involved


Relative address = 1502
0000010111011110
User process
(2700 bytes)
(a) Partitioning
Logical address =
Page# = 1, Offset = 478
0000010111011110
Page 0
Page 1
Page 2
478
Internal
fragmentation
(b) Paging
(page size = 1K)
Logical address =
Segment# = 1, Offset = 752
0001001011110000
Segment 0
750 bytes
Segment 1
1950 bytes
752
(c) Segmentation

Summary

- Memory management issues
  - relocation
  - protection
  - sharing
  - logical organization
  - physical organization
- Paging
- Memory partitioning
  - fixed partitioning
  - dynamic partitioning
  - buddy system
  - relocation
- Segmentation