Operating Systems: Internals and Design Principles

Chapter 7
Memory Management

Seventh Edition
William Stallings

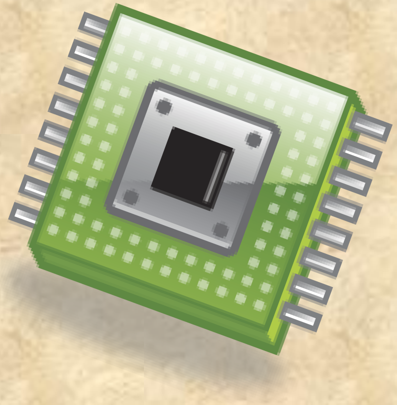
Definition

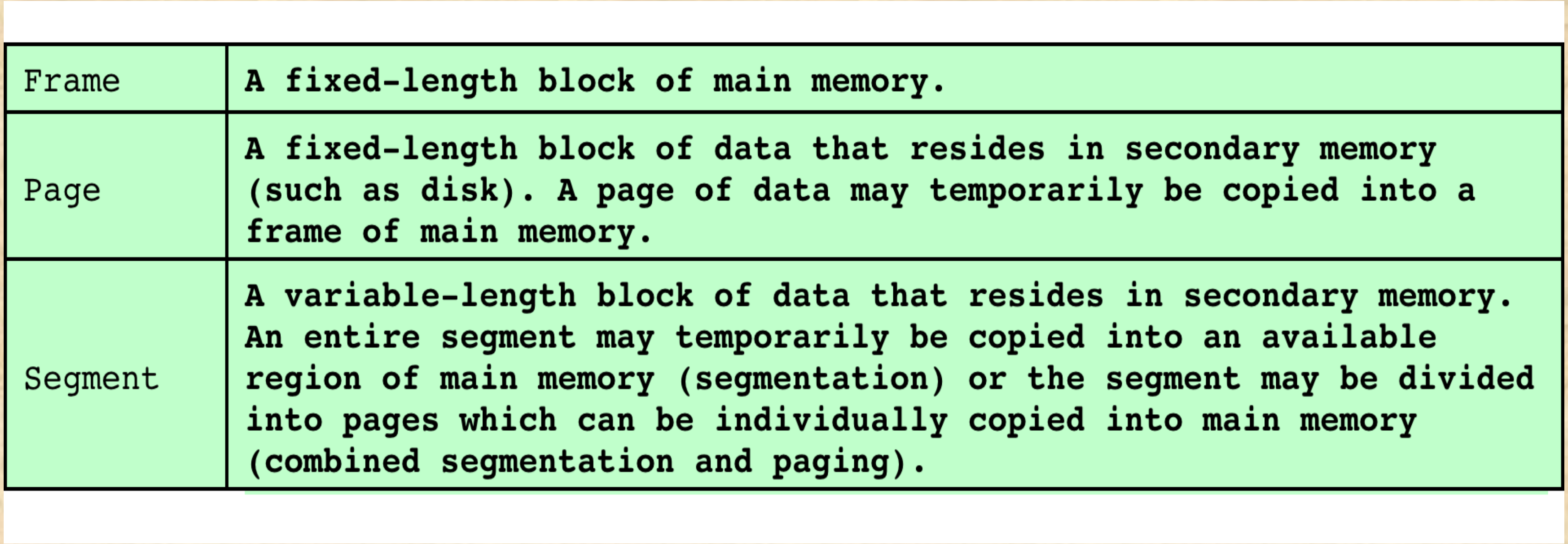
- Memory management is the process of
  - allocating primary memory to user programs
  - reclaiming that memory when it is no longer needed
  - protecting each user's memory area from other user programs; i.e., ensuring that each program only references memory locations that have been allocated to it.

Requirements

- In order to manage memory effectively the OS must have
  - Memory allocation policies
  - Methods to track the status of memory locations (free or allocated)
  - Policies for preempting memory from one process to allocate to another

Memory Management Terms

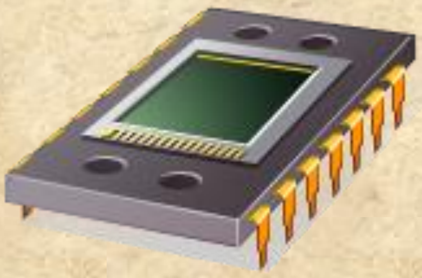




Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

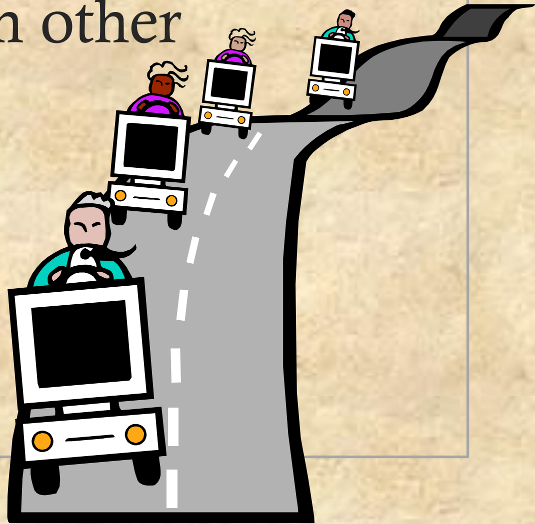
Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization



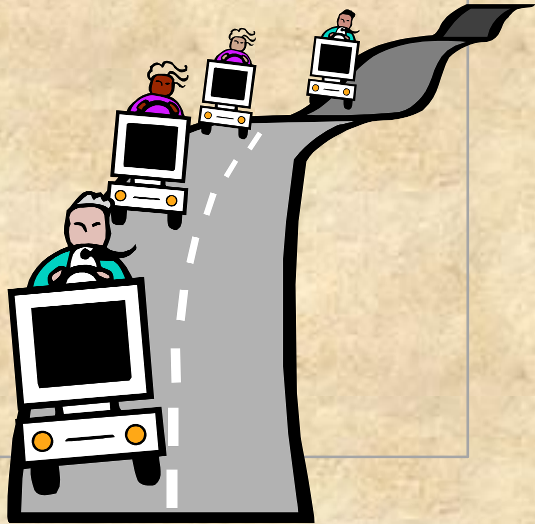
Relocation

- Relocation is the process of adjusting program addresses to match the actual physical addresses where the program resides when it executes
- Why is relocation needed?
  - Programmer/translator don't know which other programs will be memory resident when the program executes

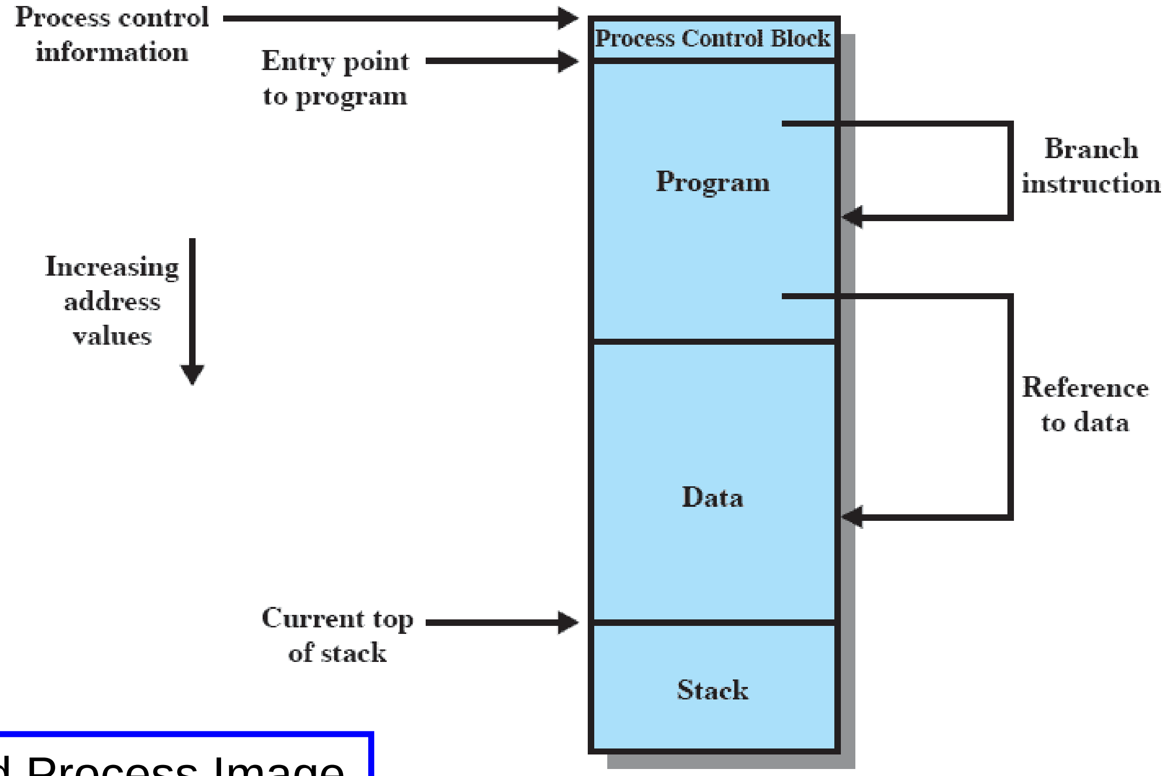


Relocation

- Why is relocation needed? (continued)
  - Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
  - Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
- Consequently it must be possible to adjust addresses whenever a program is loaded.



Addressing Requirements



Simplified Process Image

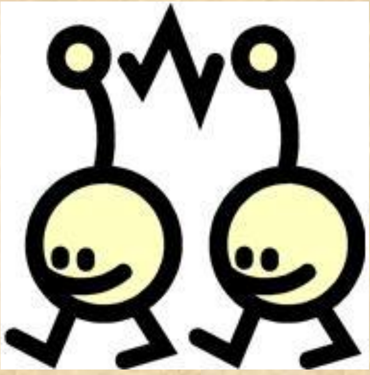
Figure 7.1 Addressing Requirements for a Process

Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
- Mechanisms that support relocation also support protection

Sharing

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



Logical Organization

- Main memory is organized as a linear (1-D) address space consisting of a sequence of bytes or words.
- Programs aren't necessarily organized this way
- Paging versus segmentation

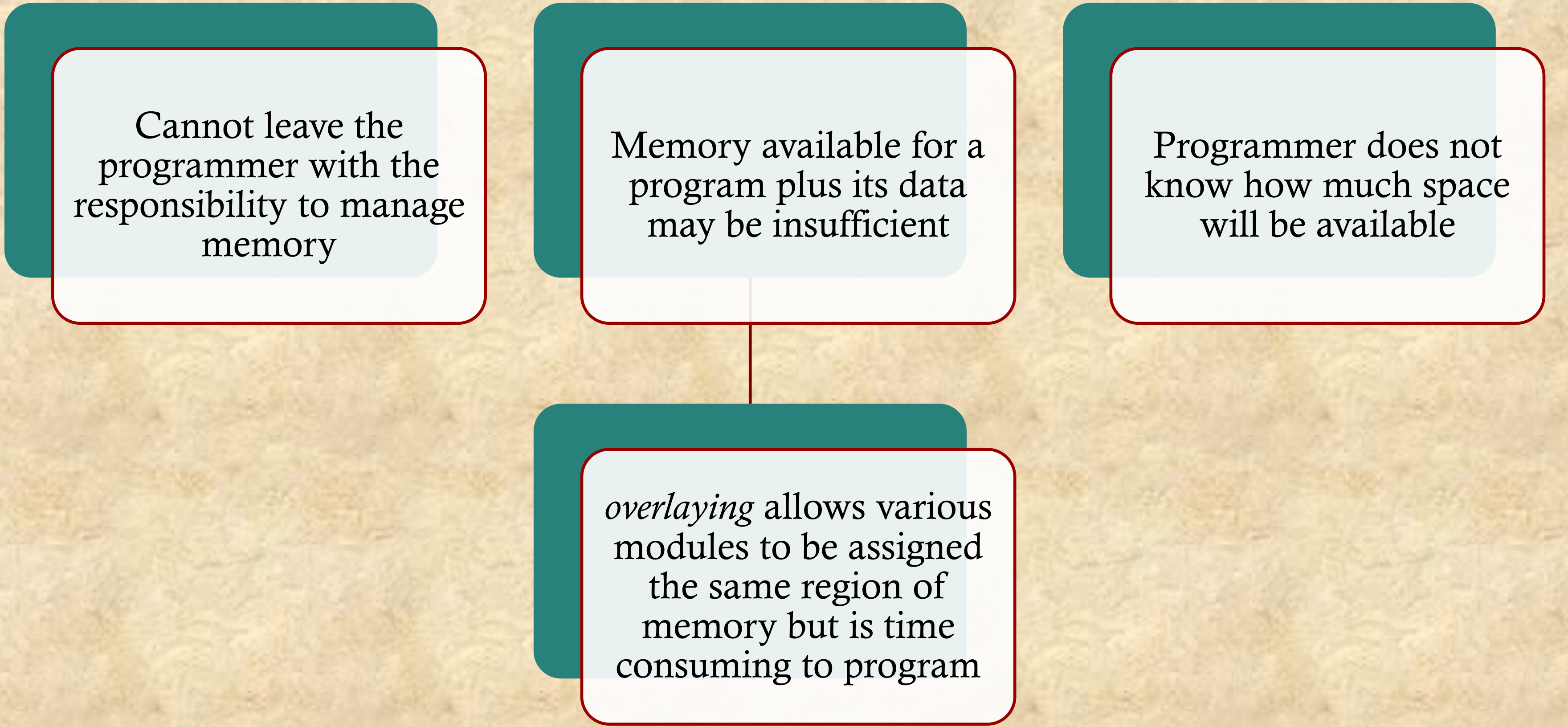
Programs are written in modules

- modules can be written and compiled independently
- different degrees of protection given to modules (read-only, execute-only)
- sharing on a module level corresponds to the user's way of viewing the problem

Physical Organization

- Two-level memory for program storage:
  - Disk (slow and cheap) & RAM (fast and more expensive)
  - Main memory is volatile, disk isn’t
- User should not have to be responsible for organizing movement of code/data between the two levels.

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

Memory Partitioning

- Virtual memory management brings processes into main memory for execution by the processor
  - involves virtual memory
  - based on segmentation and paging
- Partitioned memory management
  - used in several variations in some now-obsolete operating systems
  - does not involve virtual memory

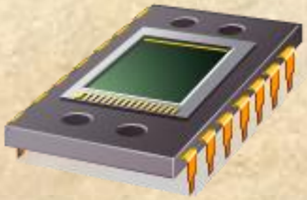
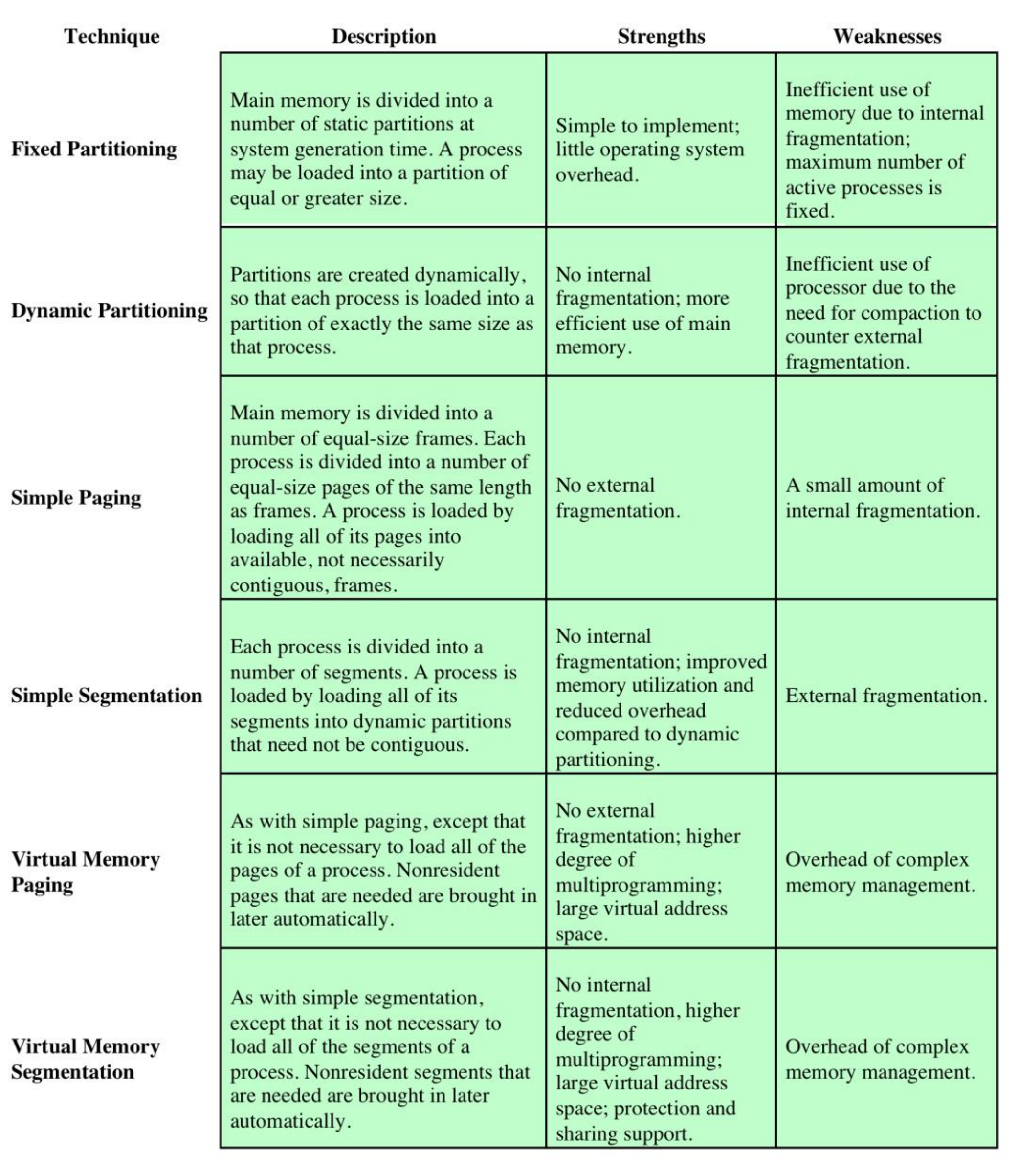


Table 7.2 Memory Management Techniques



Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

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 out a process if all partitions are full and no process is in the Ready or Running state

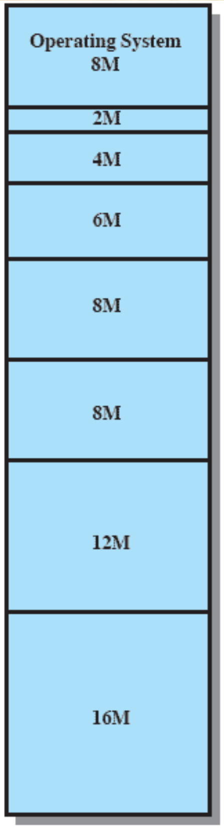
(a) Equal-size partitions

Disadvantages

- A program may be too big to fit in a partition
    - program needs to be designed with the use of 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

Unequal Size Partitions

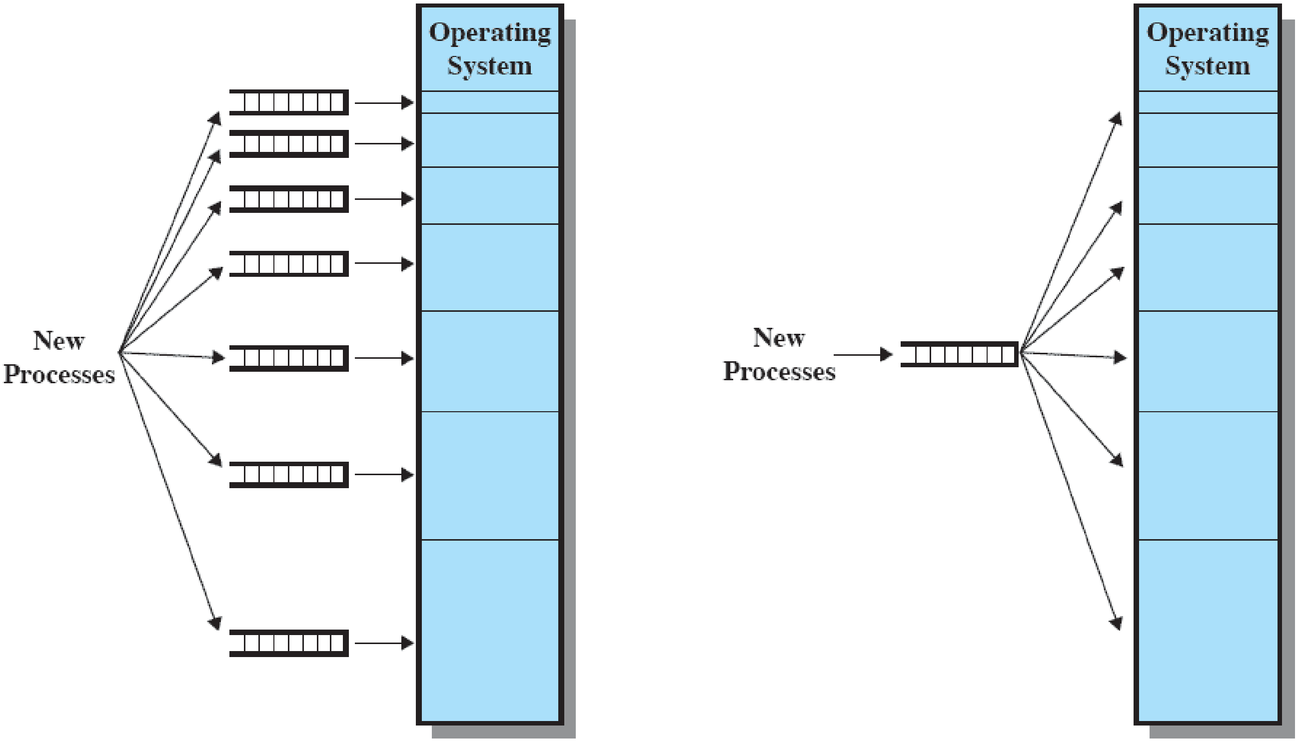
- Using unequal size partitions helps lessen the problems
  - programs up to 16M can be accommodated without overlays
  - partitions smaller than 8M allow smaller programs to be accommodated with less internal fragmentation



(b) Unequal-size partitions

Memory Assignment

Fixed Partitioning



(a) One process queue per partition

(b) Single queue

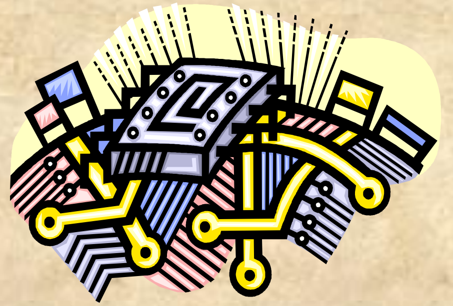
Figure 7.3 Memory Assignment for Fixed Partitioning

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

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



Effect of Dynamic Partitioning

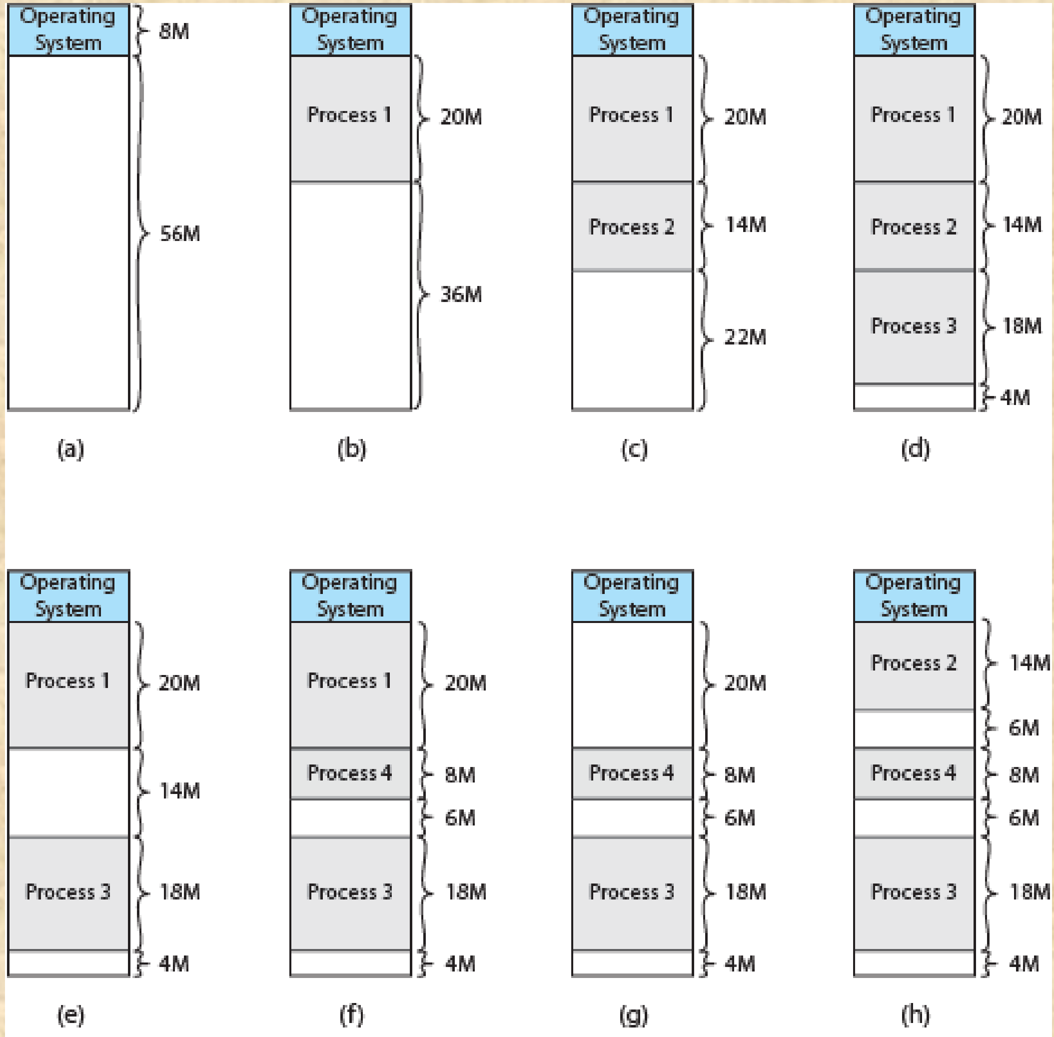


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

External Fragmentation

- memory becomes more and more fragmented
- memory utilization declines

Compaction

- technique for overcoming external fragmentation
- OS shifts processes so that they are contiguous
- free memory is together in one block
- time consuming and wastes CPU time

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

Memory Configuration Example

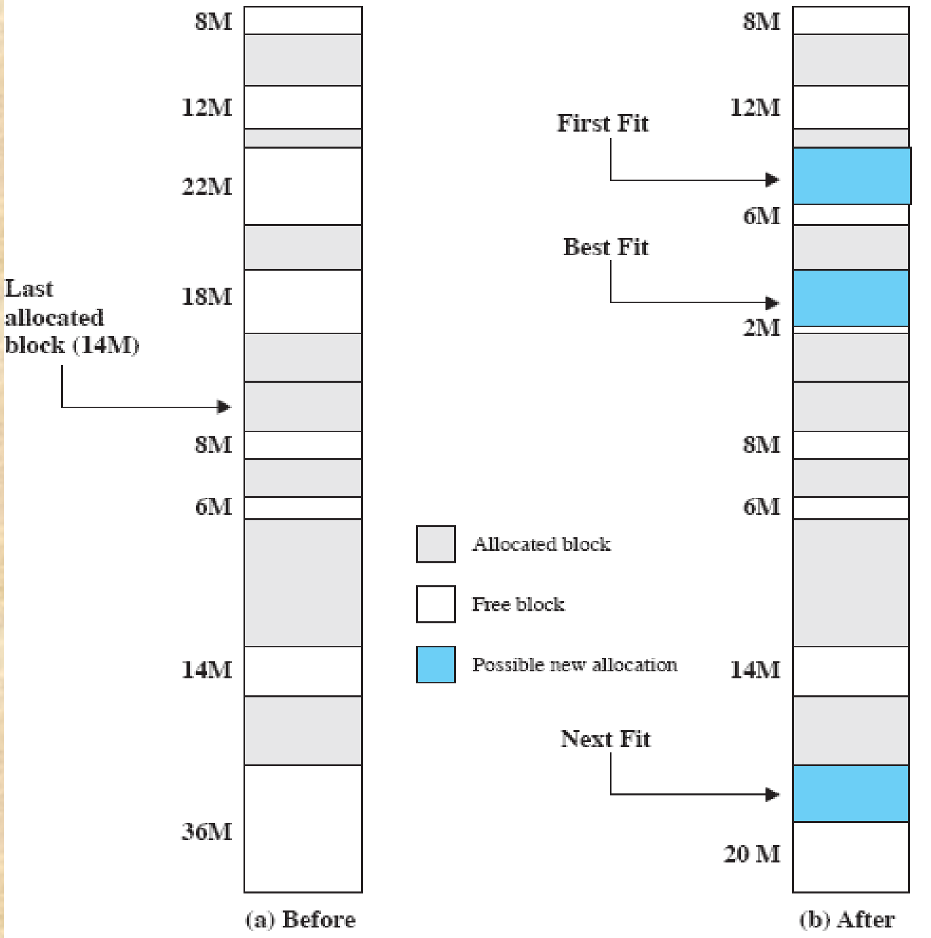
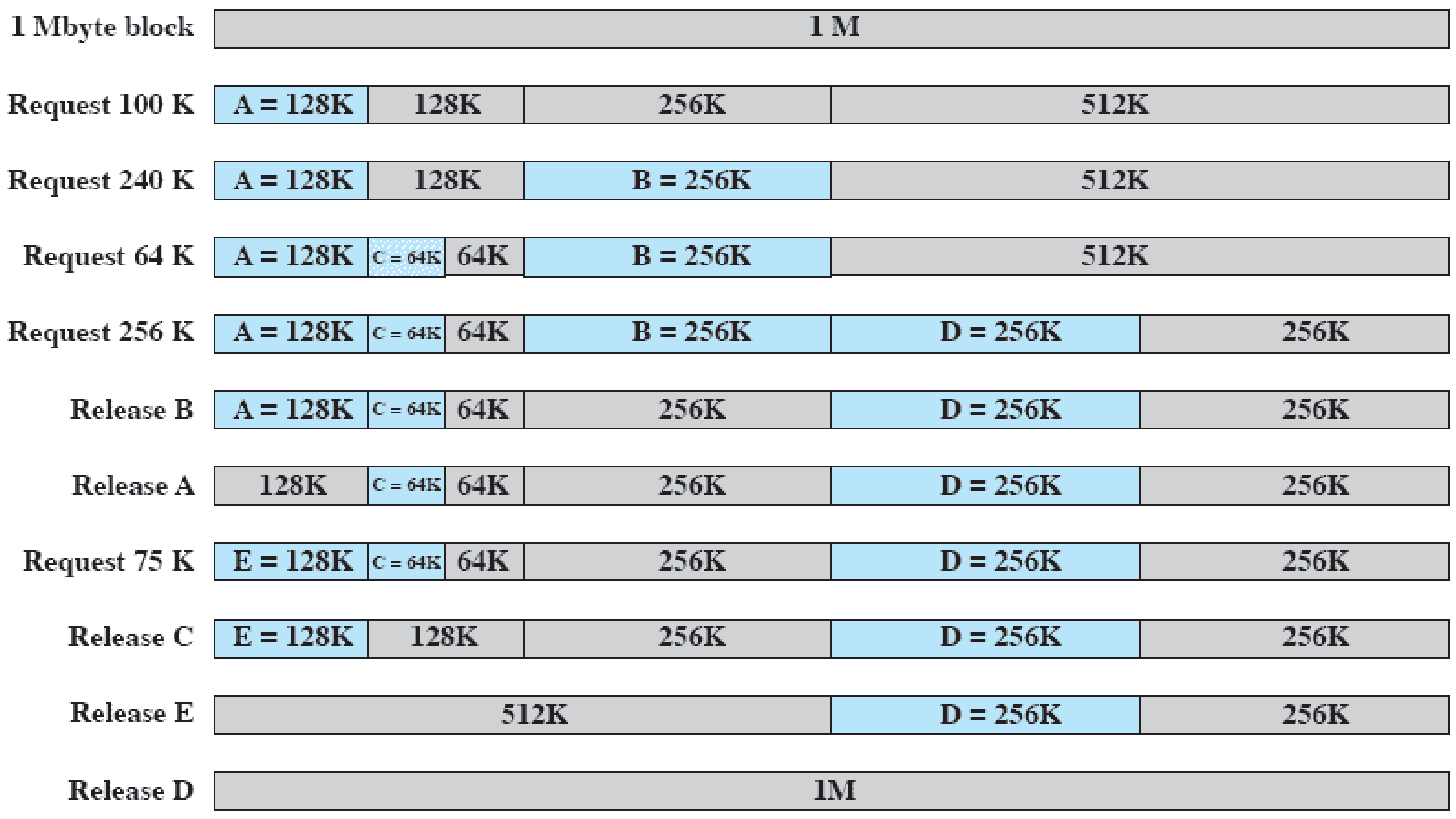


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single 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 Example



<table>
<tr><td>1 Mbyte block</td><td colspan="16">1 M</td></tr>
<tr><td>Request 100 K</td><td colspan="2">A = 128K</td><td colspan="2">128K</td><td colspan="4">256K</td><td colspan="8">512K</td></tr>
<tr><td>Request 240 K</td><td colspan="2">A = 128K</td><td colspan="2">128K</td><td colspan="4">B = 256K</td><td colspan="8">512K</td></tr>
<tr><td>Request 64 K</td><td colspan="2">A = 128K</td><td>C = 64K</td><td>64K</td><td colspan="4">B = 256K</td><td colspan="8">512K</td></tr>
<tr><td>Request 256 K</td><td colspan="2">A = 128K</td><td>C = 64K</td><td>64K</td><td colspan="4">B = 256K</td><td colspan="4">D = 256K</td><td colspan="4">256K</td></tr>
<tr><td>Release B</td><td colspan="2">A = 128K</td><td>C = 64K</td><td>64K</td><td colspan="4">256K</td><td colspan="4">D = 256K</td><td colspan="4">256K</td></tr>
<tr><td>Release A</td><td colspan="2">128K</td><td>C = 64K</td><td>64K</td><td colspan="4">256K</td><td colspan="4">D = 256K</td><td colspan="4">256K</td></tr>
<tr><td>Request 75 K</td><td colspan="2">E = 128K</td><td>C = 64K</td><td>64K</td><td colspan="4">256K</td><td colspan="4">D = 256K</td><td colspan="4">256K</td></tr>
<tr><td>Release C</td><td colspan="2">E = 128K</td><td colspan="2">128K</td><td colspan="4">256K</td><td colspan="4">D = 256K</td><td colspan="4">256K</td></tr>
<tr><td>Release E</td><td colspan="8">512K</td><td colspan="4">D = 256K</td><td colspan="4">256K</td></tr>
<tr><td>Release D</td><td colspan="16">1M</td></tr>
</table>

Figure 7.6 Example of Buddy System

Tree Representation

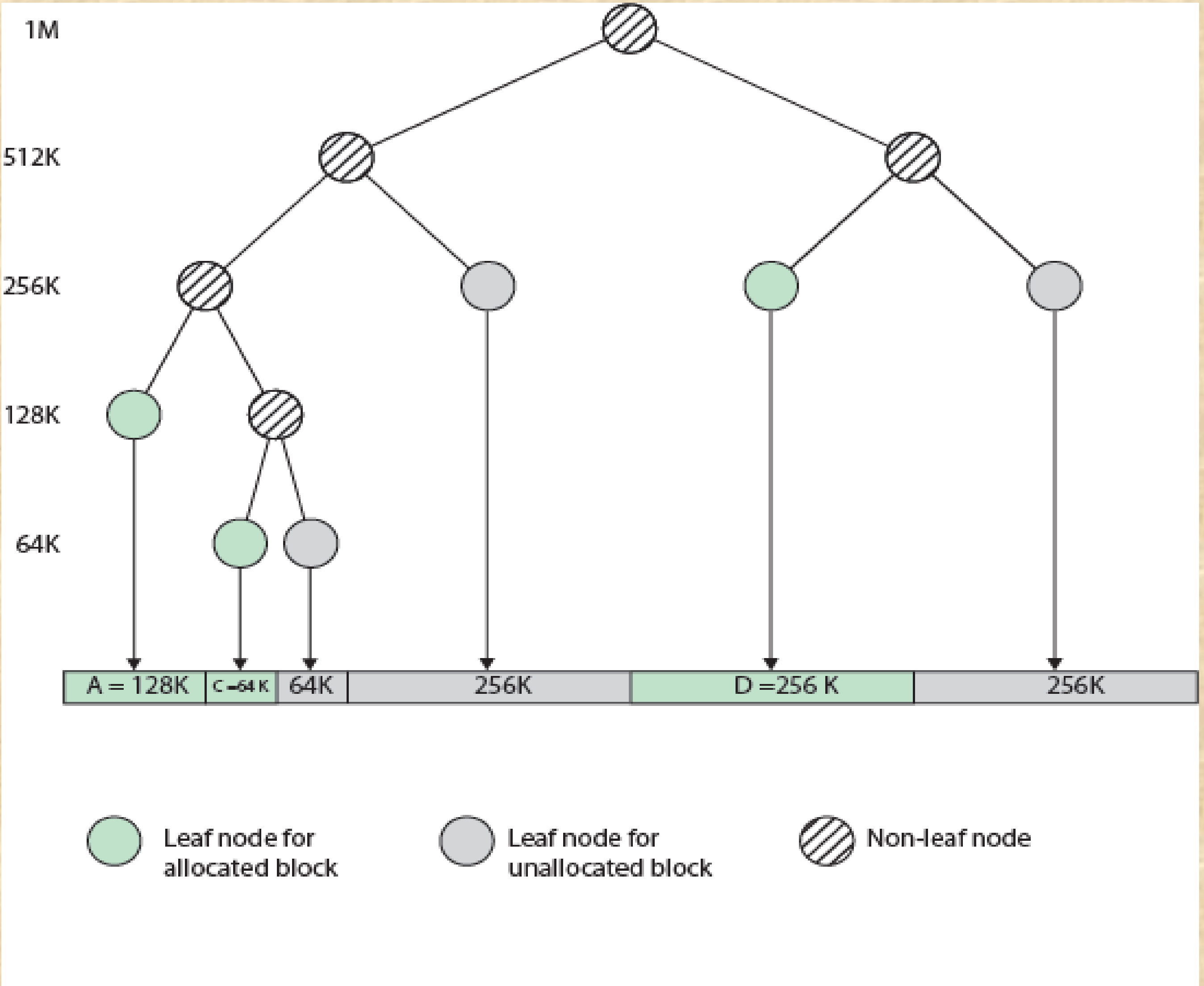


Figure 7.7 Tree Representation of Buddy System

Addresses

Logical

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

Relative

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

Physical or Absolute

- actual location in main memory

Relocation

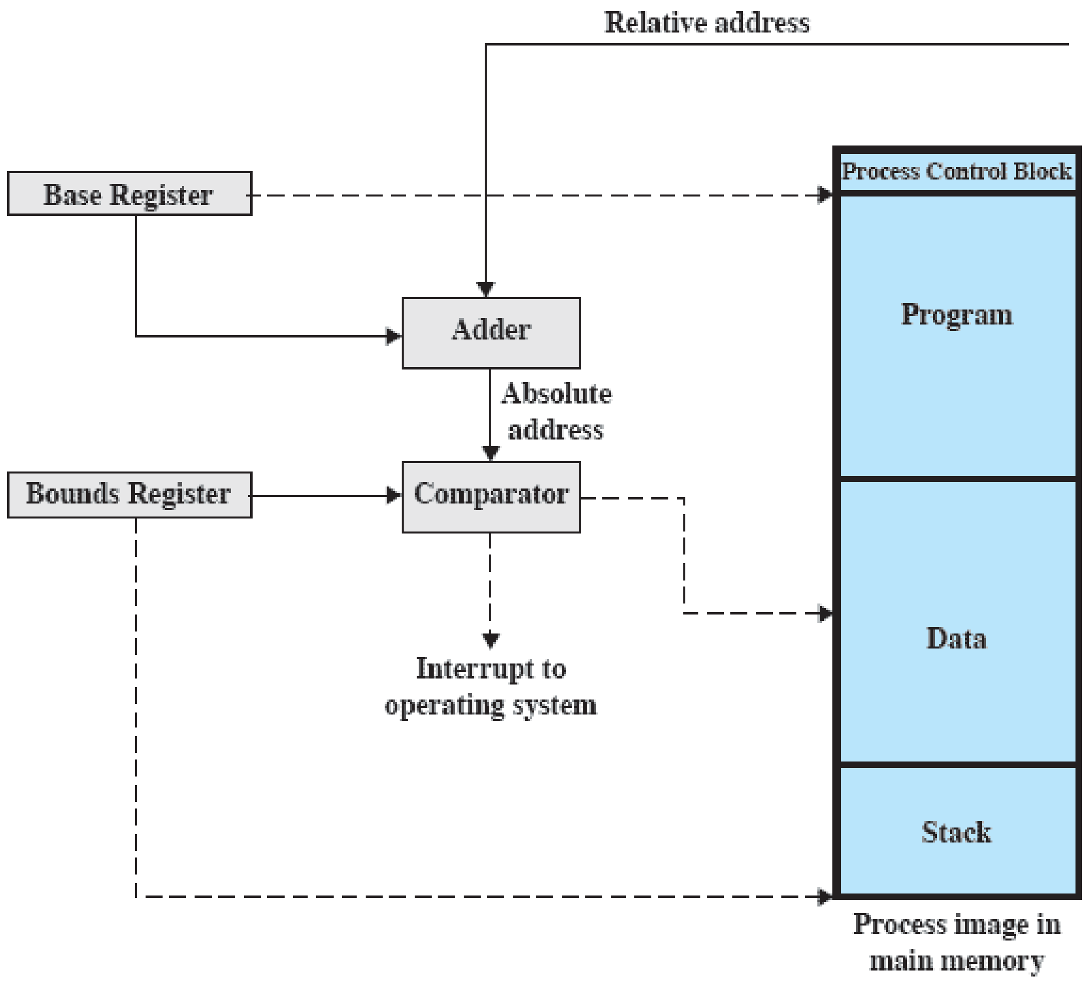


Figure 7.8 Hardware Support for Relocation

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

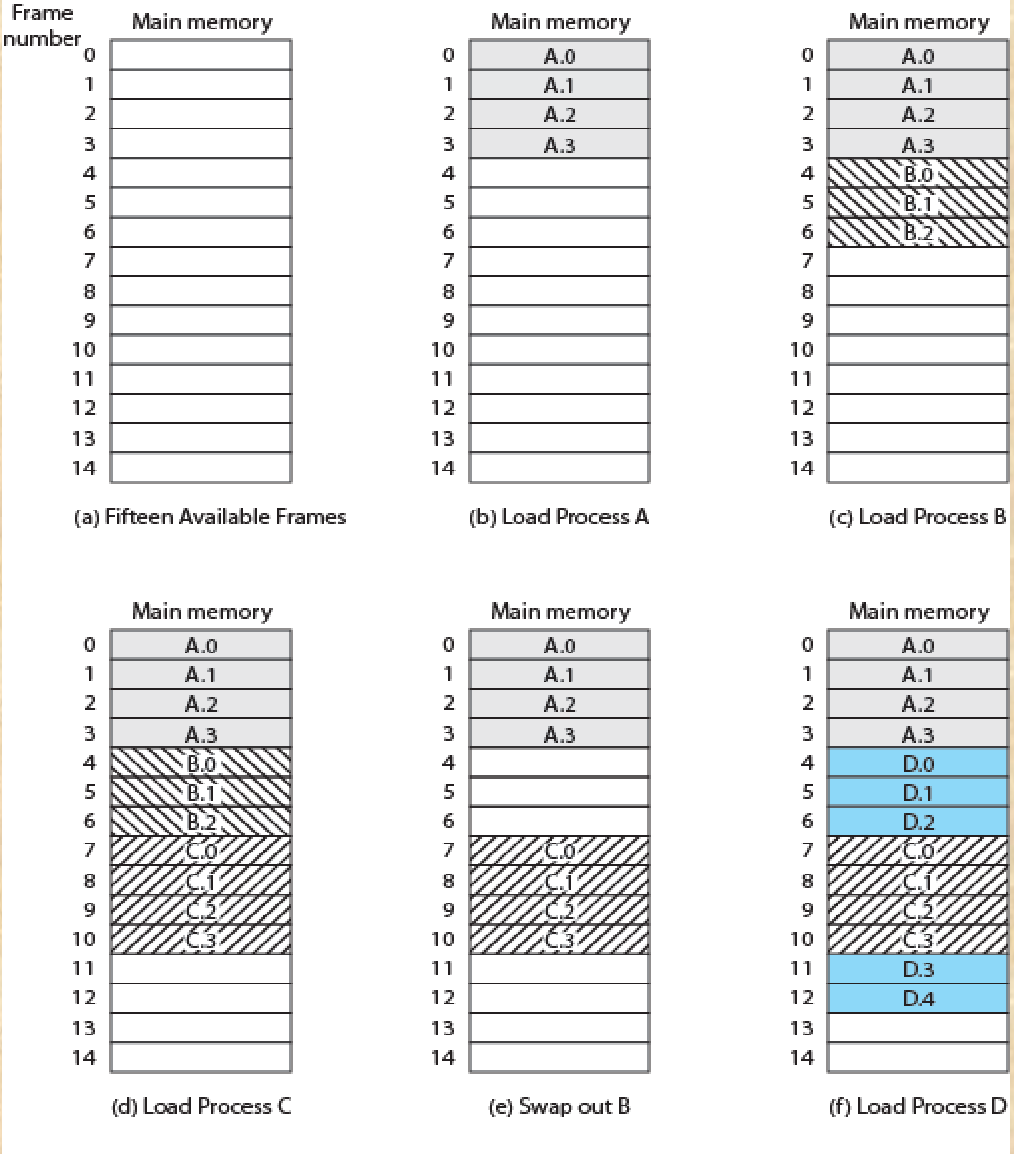
Pages

- chunks of a process

Frames

- available chunks of memory

Assignment of Process to Free Frames

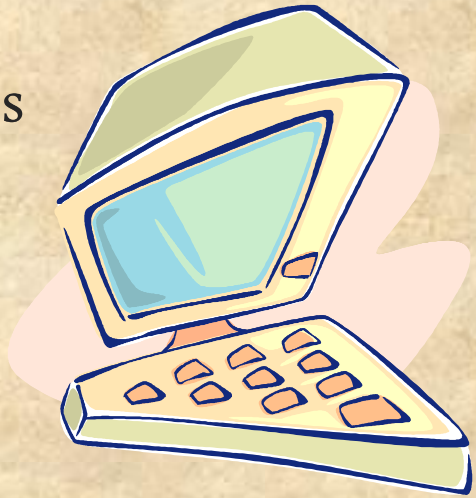


(a) Fifteen Available Frames
(b) Load Process A
(c) Load Process B
(d) Load Process C
(e) Swap out B
(f) Load Process D

Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

- Maintained by operating system for each process
- Contains the frame location for each page in the process
- Processor must know how to access the page table for the current process
- Used by processor to produce a physical address



Data Structures

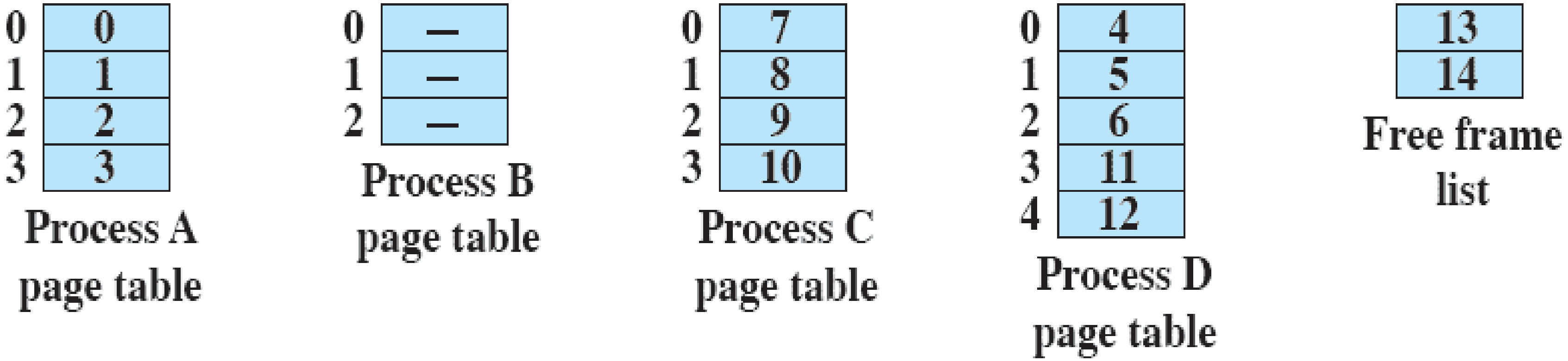
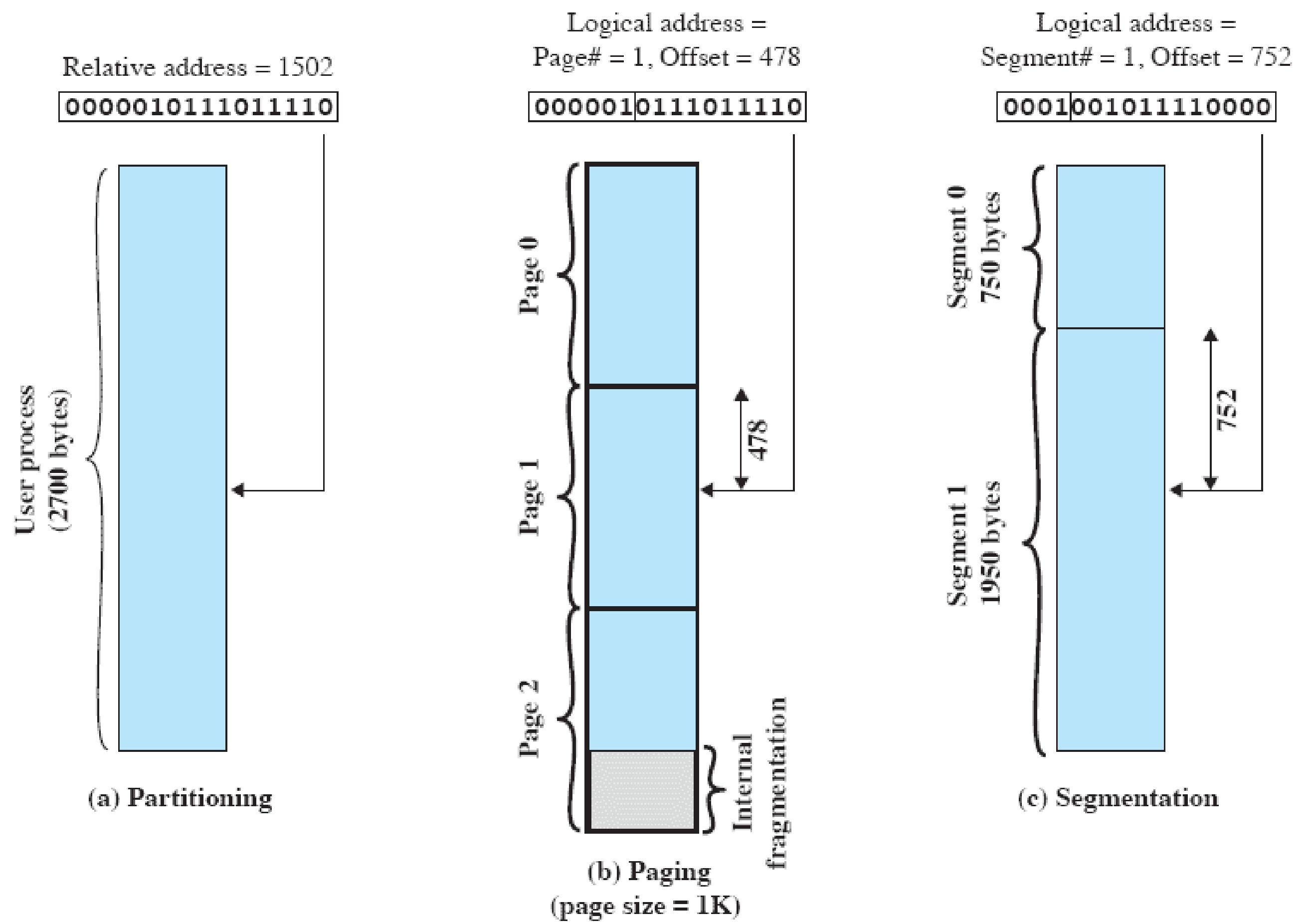


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Logical Addresses



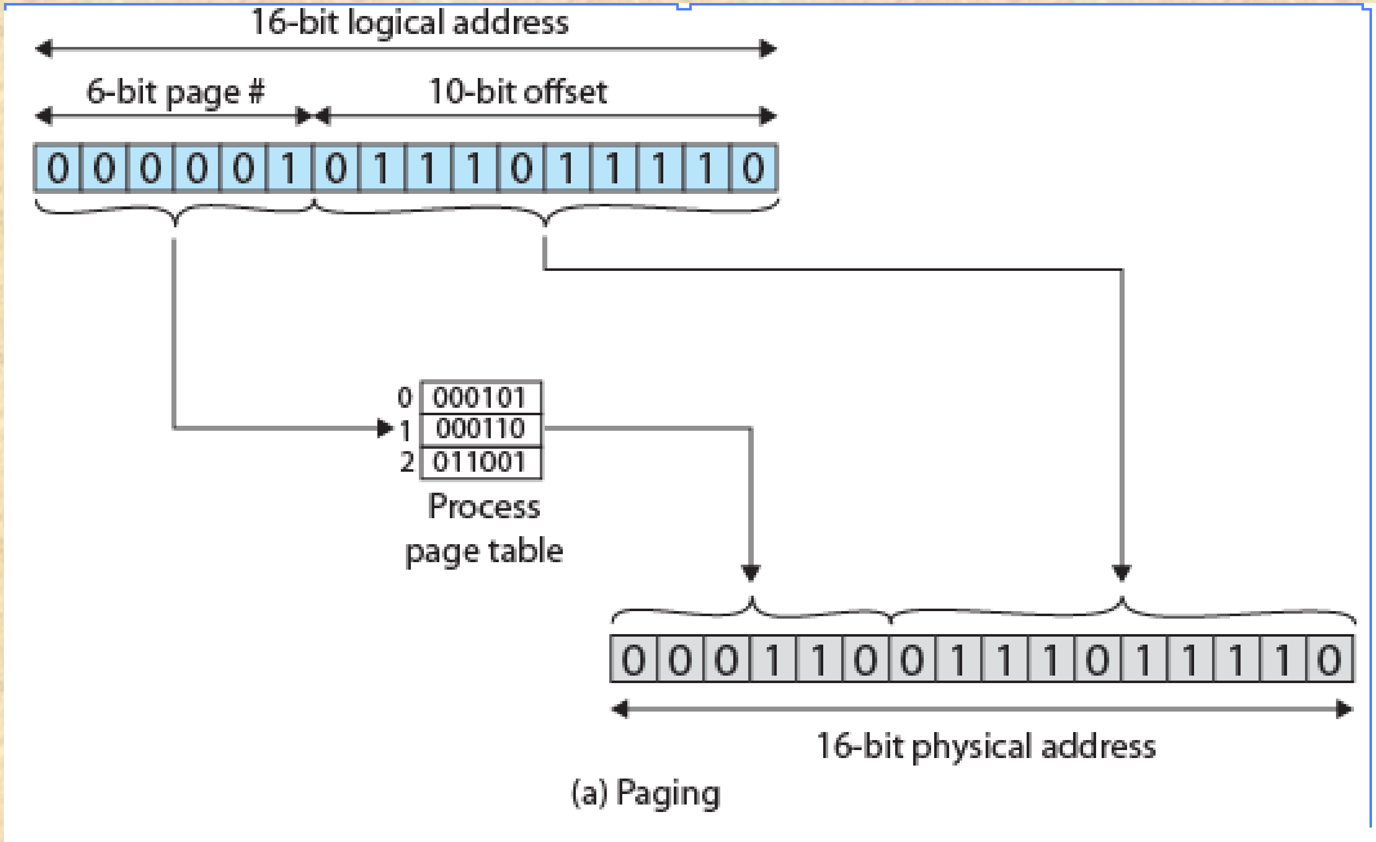
(a) Partitioning

(b) Paging
(page size = 1K)

(c) Segmentation

Figure 7.11 Logical Addresses

Logical-to-Physical Address Translation - Paging



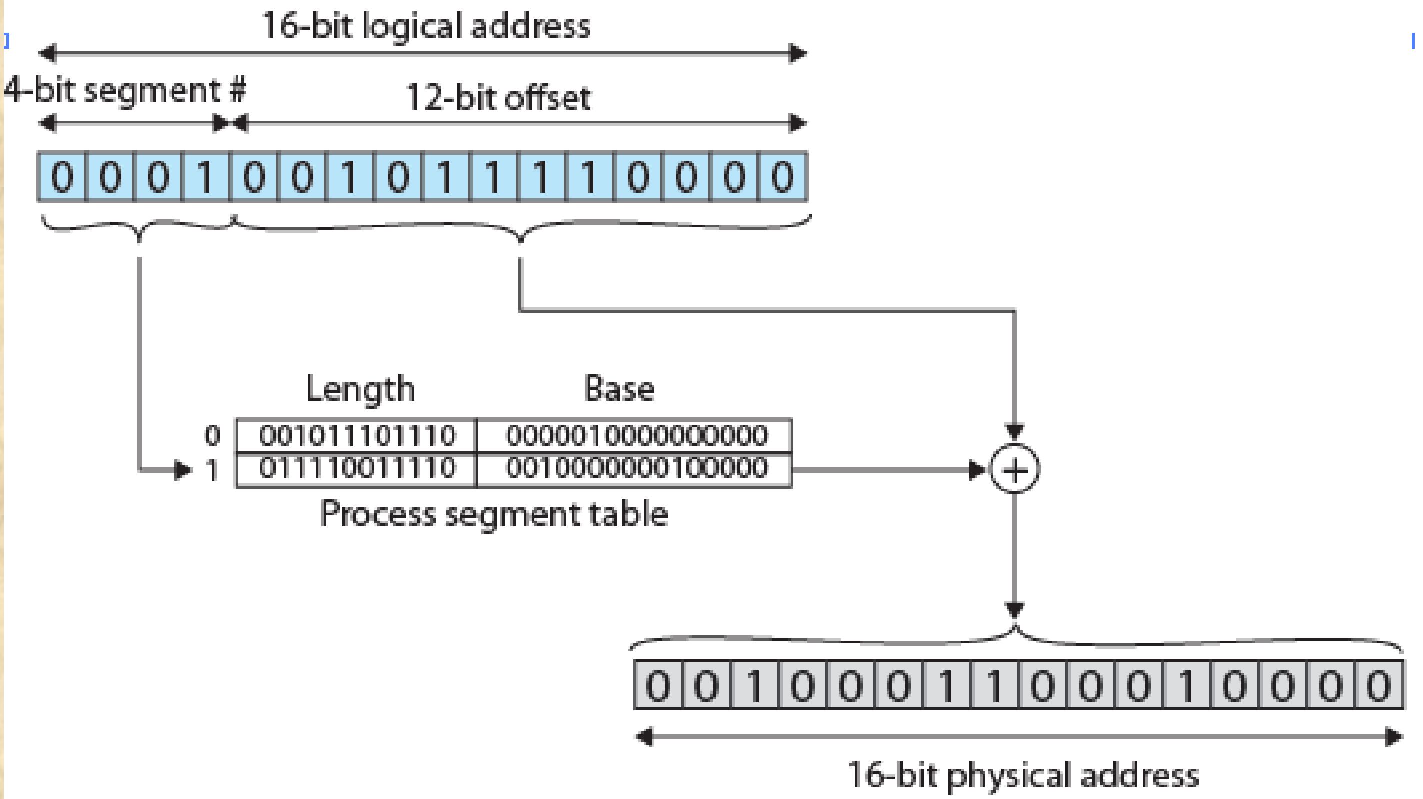
(a) Paging

Segmentation

- A program can be subdivided into segments
  - may vary in length
  - there is a maximum length
- Addressing consists of two parts:
  - segment number
  - an offset

- Similar to dynamic partitioning
- Eliminates internal fragmentation

Logical-to-Physical Address Translation - Segmentation



	Length	Base
0	001011101110	0000010000000000
1	011110011110	0010000000100000

Process segment table

(b) Segmentation

Security Issues

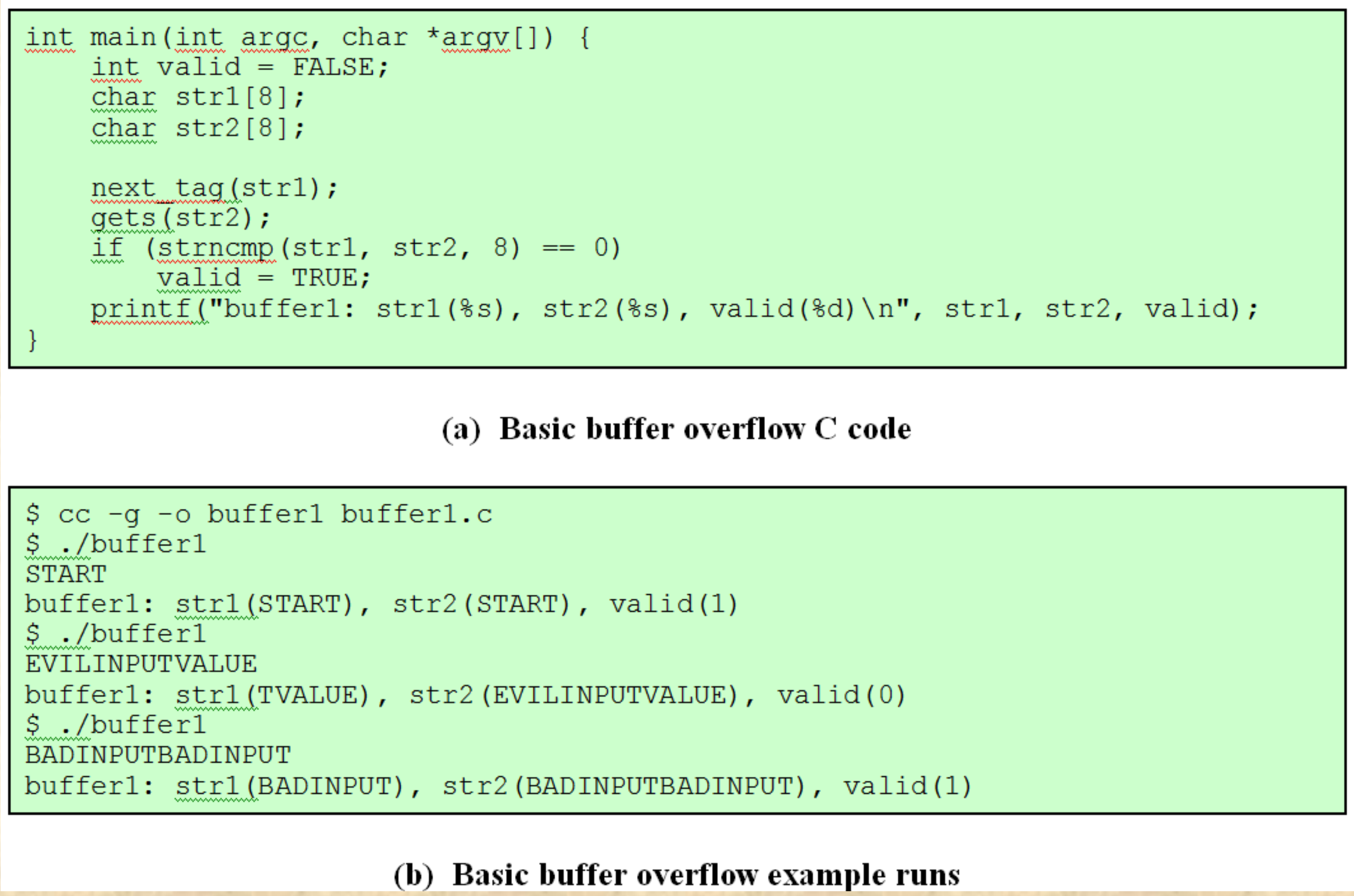
If a process has not declared a portion of its memory to be sharable, then no other process should have access to the contents of that portion of memory

If a process declares that a portion of memory may be shared by other designated processes then the security service of the OS must ensure that only the designated processes have access

Buffer Overflow Attacks

- Security threat related to memory management
- Also known as a buffer overrun
- Can occur when a process attempts to store data beyond the limits of a fixed-sized buffer
- One of the most prevalent and dangerous types of security attacks

Buffer Overflow Example



```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

(a) Basic buffer overflow C code

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

(b) Basic buffer overflow example runs

Buffer Overflow Stack Values

Memory Address	Before gets(str2)	After gets(str2)	Contains Value of
. . . .	. . . .	. . . .	
bffffbf4	34fcffbf 4 . . .	34fcffbf 3 . . .	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 . . . @	c6bd0340 . . . @	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 V . @	42414449 B A D I	str2[0-3]
. . . .	. . . .	. . . .	

Defending Against Buffer Overflows

- Prevention
- Detecting and aborting
- Countermeasure categories:

Compile-time Defenses

- aim to harden programs to resist attacks in new programs

Run-time Defenses

- aim to detect and abort attacks in existing programs

Summary

- Memory Management
    - one of the most important and complex tasks of an operating system
    - needs to be treated as a resource to be allocated to and shared among a number of active processes
    - desirable to maintain as many processes in main memory as possible
    - desirable to free programmers from size restriction in program development
    - basic tools are paging and segmentation (possible to combine)
        - paging – small fixed-sized pages
        - segmentation – pieces of varying size