Understanding Operating Systems Seventh Edition

Chapter 3
Memory Management: Virtual Memory Systems

Learning Objectives

After completing this chapter, you should be able to describe:

- The basic functionality of the memory allocation methods covered in this chapter: paged, demand paging, segmented, and segmented/demand paged memory allocation
- The influence that these page allocation methods have had on virtual memory

Learning Objectives (cont'd.)

- The difference between a first-in first-out page replacement policy, a least-recently-used page replacement policy, and a clock page replacement policy
- The mechanics of paging and how a memory allocation scheme determines which pages should be swapped out of memory

Learning Objectives (cont'd.)

- The concept of the working set and how it is used in memory allocation schemes
- Cache memory and its role in improving system response time

Introduction

- Evolution of virtual memory
  - Paged, demand paging, segmented, segmented/demand paging
  - Foundation of current virtual memory methods
- Areas of improvement from the need for:
  - Continuous program storage
  - Placement of entire program in memory during execution
- Enhanced Memory Manager performance: cache memory

Paged Memory Allocation

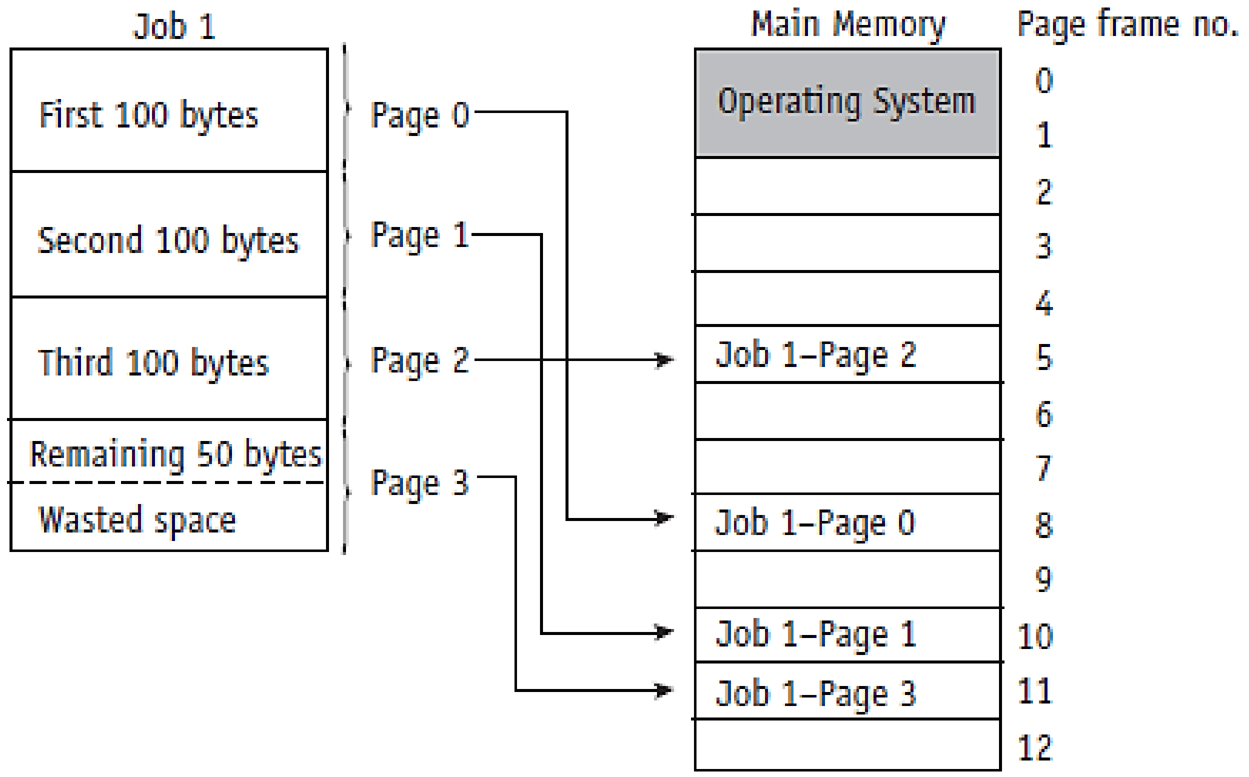
- Incoming job: divided into pages of equal size
- Best condition
  - Pages, sectors, and page frames: same size
    - Exact sizes: determined by disk’s sector size
- Memory manager tasks: prior to program execution
  - Determine number of pages in program
  - Locate enough empty page frames in main memory
  - Load all program pages into page frames

Paged Memory Allocation (cont'd.)

- Program: stored in noncontiguous page frames
  - Advantages: more efficient memory use; compaction scheme eliminated (no external fragmentation)
  - New problem: keeping track of job's pages (increased size, complexity, and operating system overhead)

Paged Memory Allocation (cont'd.)

(figure 3.1)
In this example, each page frame can hold 100 bytes. This job, at 350 bytes long, is divided among four page frames with internal fragmentation in the last page frame.
© Cengage Learning 2014



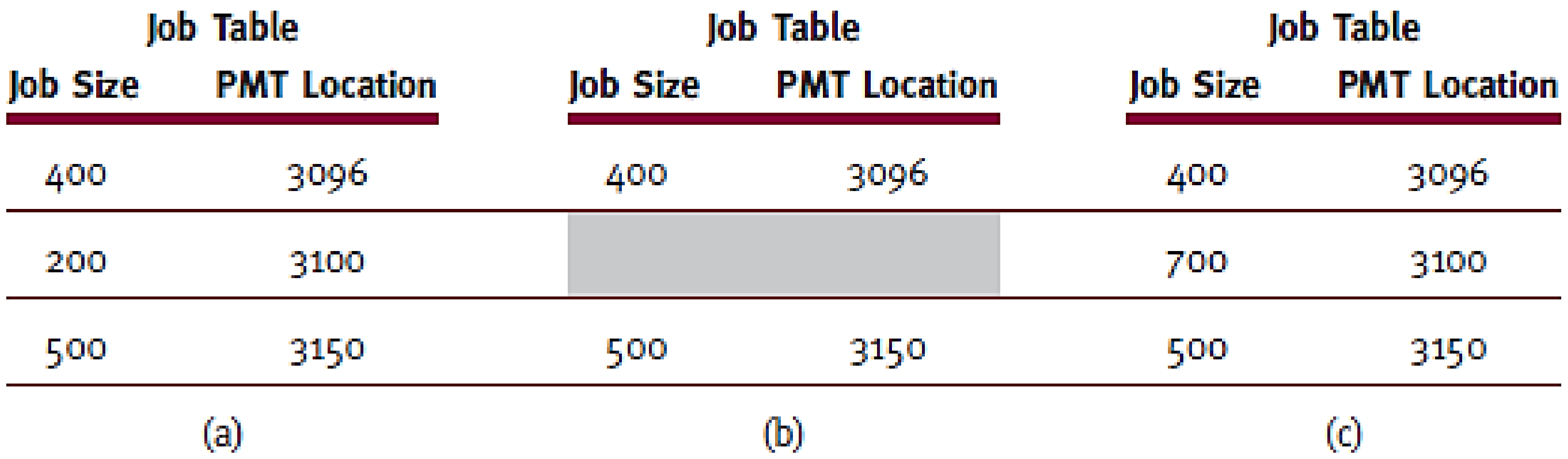
Paged Memory Allocation (cont'd.)

- Internal fragmentation: job's last page frame only
- Entire program: required in memory during its execution
- Three tables for tracking pages: Job Table (JT), Page Map Table (PMT), and Memory Map Table (MMT)
  - Stored in main memory: operating system area

Paged Memory Allocation (cont’d.)

- Job Table: information for each active job; is a dynamic list that grows and shrinks as jobs are loaded and unloaded into the system
  - Job size
  - Memory location: job’s PMT
- Page Map Table: information for each page; each job has its own PMT
  - Page number: beginning with Page 0
  - Page’s Memory address
- Memory Map Table: entry for each page frame
  - Location
  - Free/busy status

Paged Memory Allocation (cont'd.)



Job Table

Job Size	PMT Location
400	3096
200	3100
500	3150

(a)

Job Table

Job Size	PMT Location
400	3096
500	3150

(b)

Job Table

Job Size	PMT Location
400	3096
700	3100
500	3150

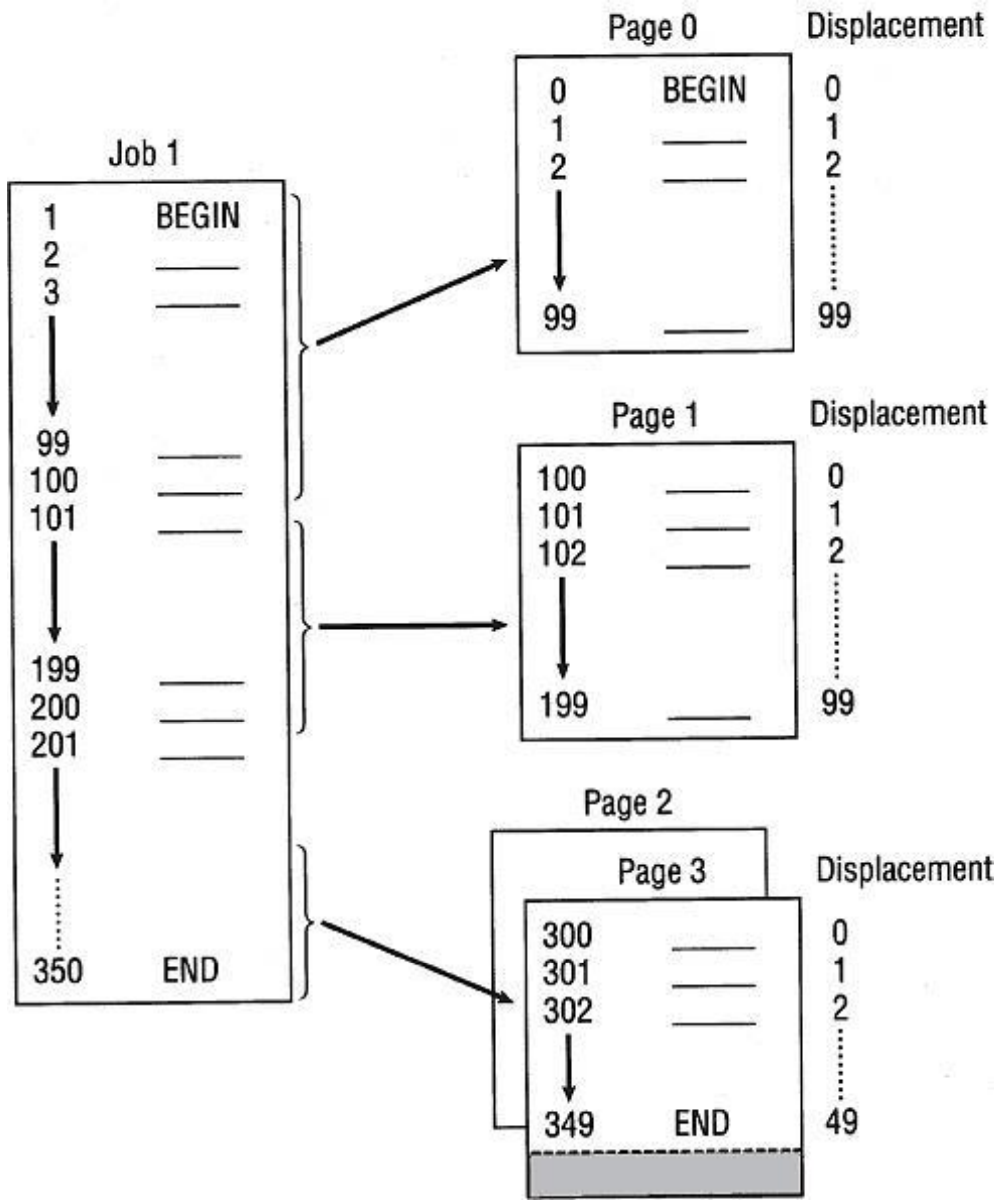
(c)

(table 3.1)
This section of the Job Table initially has one entry for each job (a). When the second job ends (b), its entry in the table is released and then replaced by the entry for the next job (c).
© Cengage Learning 2014

Paged Memory Allocation (cont'd.)

- Line displacement (offset)
  - Line distance: from beginning of its page
  - Line location: within its page frame
  - Relative value
- Determining page number and displacement of a line
  - Divide job space address by the page size
  - Page number: integer quotient
  - Displacement: remainder

(figure 3.2)
This job is 350 bytes long and is divided into four pages of 100 bytes each that are loaded into four page frames in memory.
© Cengage Learning 2014

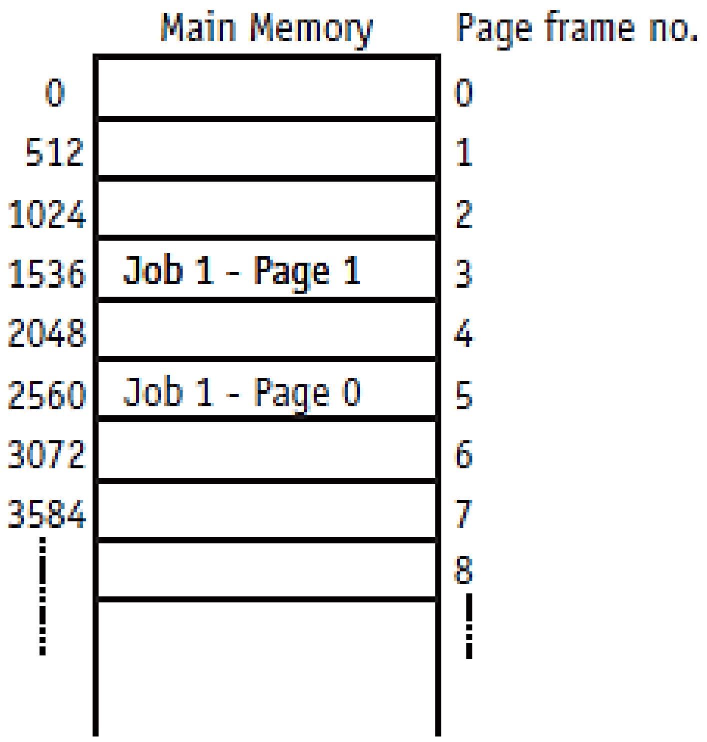


Job 1

Byte no.	Instruction/Data
000	BEGIN
025	LOAD R1, 518
518	3792

PMT for Job 1

Page no.	Page frame number
0	5
1	3

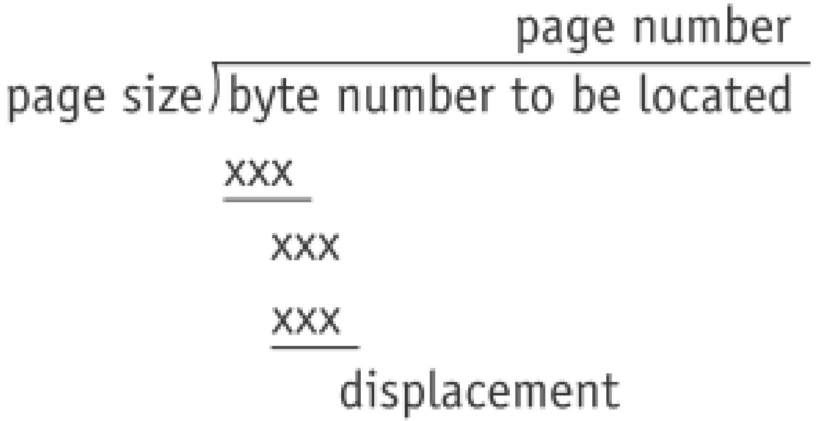


(figure 3.3)
This system has page frame and page sizes of 512 bytes each. The PMT shows where the job’s two pages are loaded into available page frames in main memory.
© Cengage Learning 2014

Paged Memory Allocation (cont'd.)

- Page displacement calculation: OS uses simple arithmetic calculation
  - BYTE_NUMBER_TO_BE_LOCATED / PAGE_SIZE = PAGE_NUMBER

  - For example: 276 / 100 = 2.76 ⇒ 2 (with a remainder of 76)
  - For example: 6144 / 4096 = 1 ⇒ (with a remainder of 2048)



Paged Memory Allocation (cont'd.)

- Instruction: determining exact location in memory

  Step1: Determine page number/displacement of line

  Step 2: Refer to the job’s PMT

    - Determine page frame containing required page

  Step 3: Obtain beginning address of page frame

    - Multiply page frame number by page frame size

  Step 4: Add the displacement (calculated in first step) to starting address of the page frame

- Address resolution (address translation)
  - Job space address (logical) → physical address (absolute)

Paged Memory Allocation (cont'd.)

- Advantages
  - Efficient memory use: job allocation in noncontiguous memory
- Disadvantages
  - Increased overhead: address resolution
  - Internal fragmentation: last page
- Page size: crucial
  - Too small: very long PMTs
  - Too large: excessive internal fragmentation

Paged Memory Allocation (cont'd.)

Using the concepts just presented, answer the following questions by assuming page size is 100.

1. Could the operating system (or the hardware) get a page number that is greater than 3 if the program was searching for Byte 276?
2. If it did, what should the operating system do?
3. Could the operating system get a remainder / displacement of more than 99?
4. What is the smallest remainder/displacement possible?

Demand Paging Memory Allocation

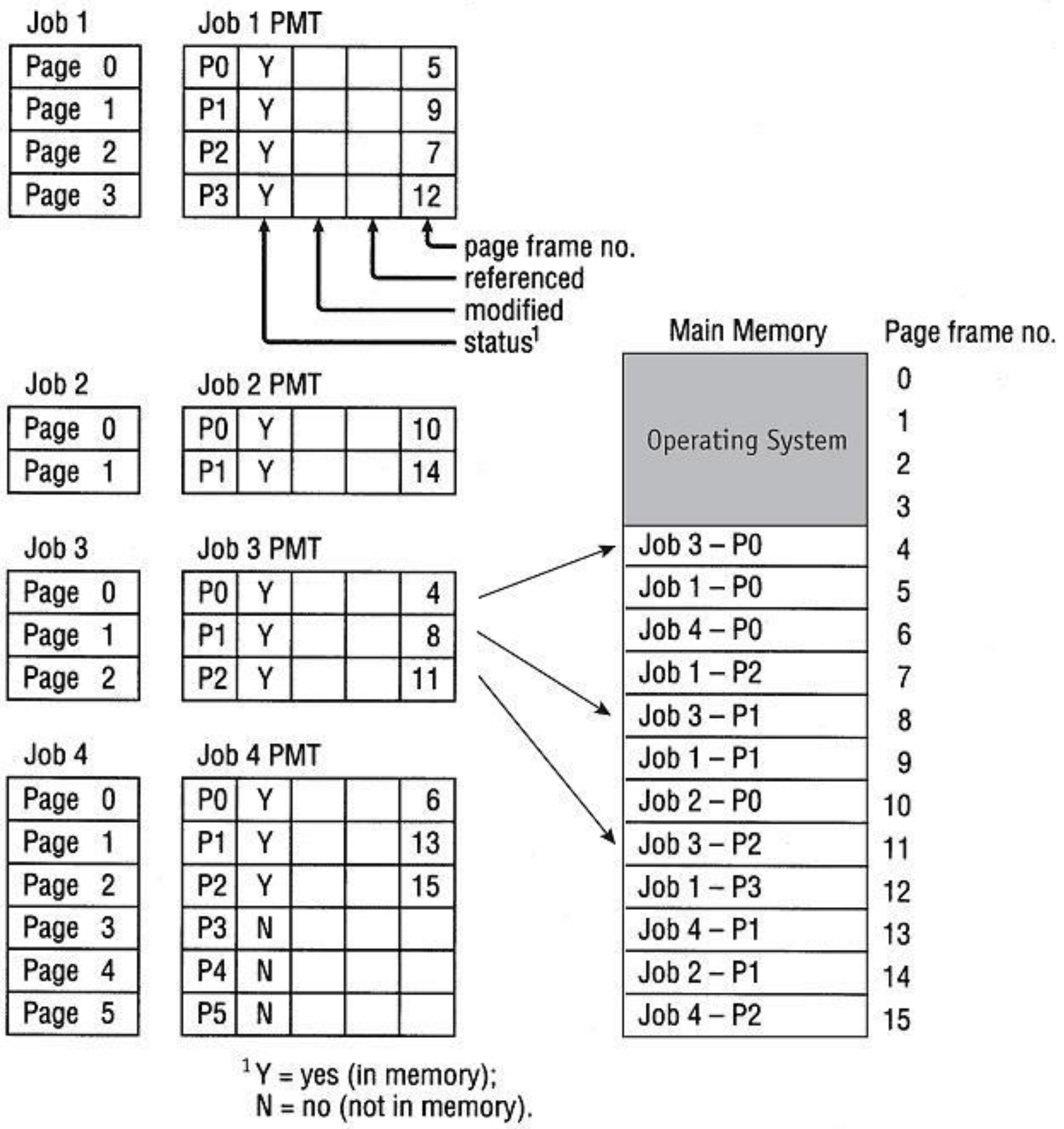
- Loads only a part of the program into memory
  - Removes restriction: entire program in memory
  - Requires high-speed page access
- Exploits programming techniques
  - Modules: written sequentially
    - All pages: not needed simultaneously
  - Examples
    - Error-handling modules instructions
    - Mutually exclusive modules
    - Certain program options: mutually exclusive or not always accessible

Demand Paging (cont'd.)

- Virtual memory
  - Appearance of vast amounts of physical memory
- Less main memory required than paged memory allocation scheme
- Requires high-speed direct access storage device (DASDs): e.g., hard drives or flash memory
- Swapping: how and when pages passed between memory and secondary storage
  - Depends on predefined policies

Demand Paging Memory Allocation (cont'd.)

- Algorithm implementation: tables, e.g., Job Table, Page Map Table, and Memory Map Table
- Page Map Table
  - First field: page requested already in memory?
  - Second field: page contents modified?
  - Third field: page referenced recently?
  - Fourth field: frame number



(figure 3.5)
Demand paging requires that the Page Map Table for each job keep track of each page as it is loaded or removed from main memory. Each PMT tracks the status of the page, whether it has been modified, whether it has been recently referenced, and the page frame number for each page currently in main memory. (Note: For this illustration, the Page Map Tables have been simplified. See Table 3.3 for more detail.
© Cengage Learning 2014

Demand Paging Memory Allocation (cont'd.)

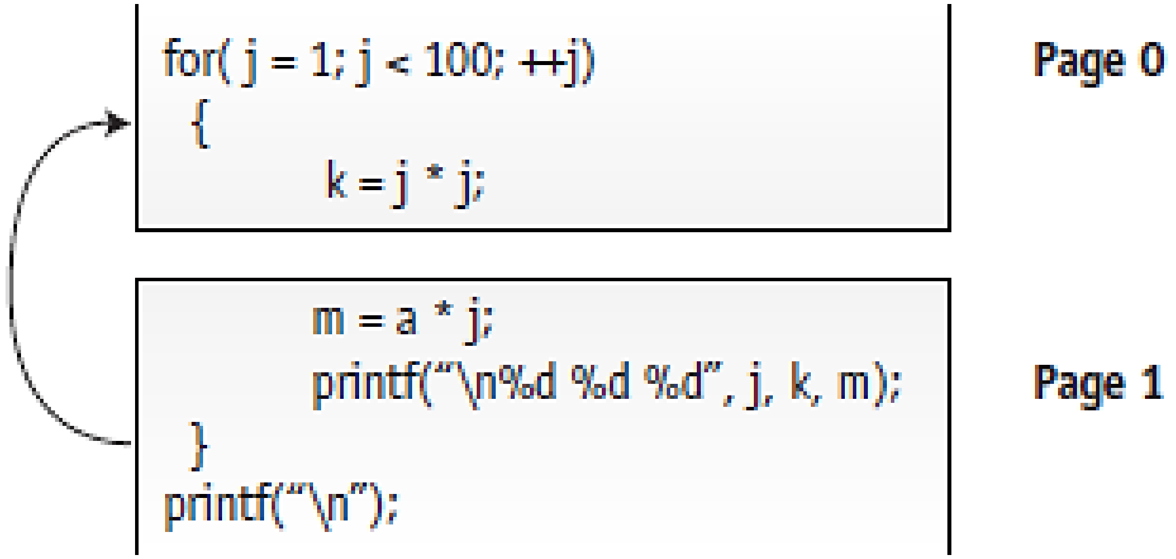
- Swapping process
  - Resident memory page: exchanged with secondary storage page
    - Resident page: copied to disk (if modified)
    - New page: written into available page frame
  - Requires close interaction between:
    - Hardware components
    - Software algorithms
    - Policy schemes

Demand Paging Memory Allocation (cont'd.)

- Hardware components:
  - Generate the address: required page
  - Find the page number
  - Determine page status: already in memory
- Page fault: failure to find page in memory
- Page fault handler: part of operating system
  - Determines if empty page frames in memory
    - Yes: requested page copied from secondary storage
    - No: swapping (dependent on the predefined policy)

Demand Paging Memory Allocation (cont'd.)

- Tables updated when page swap occurs
  - PMT for both jobs (page swapped out; page swapped in) and the MMT
- Thrashing
  - Excessive page swapping: inefficient operation
  - Main memory pages: removed frequently; called back soon thereafter
  - Occurs across jobs
    - Large number of jobs: limited free page frames
  - Occurs within a job
    - Loops crossing page boundaries


```
for( j = 1; j < 100; ++j)
  {
          k = j * j;
```

```
          m = a * j;
          printf("\n%d %d %d", j, k, m);
  }
printf("\n");
```

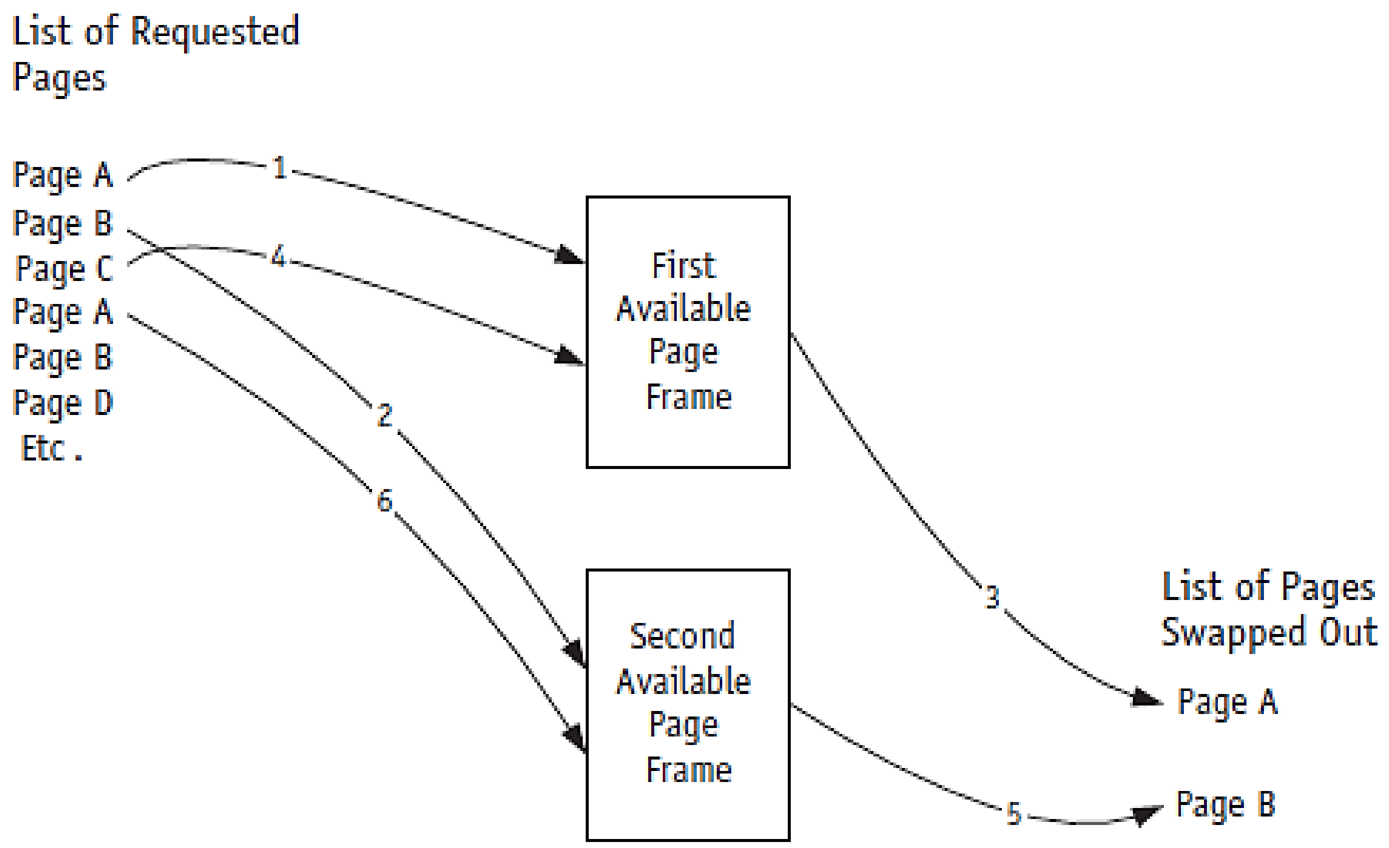
(figure 3.6)
An example of demand paging that causes a page swap each time the loop is executed and results in thrashing. If only a single page frame is available, this program will have one page fault each time the loop is executed.

Page Replacement Policies and Concepts

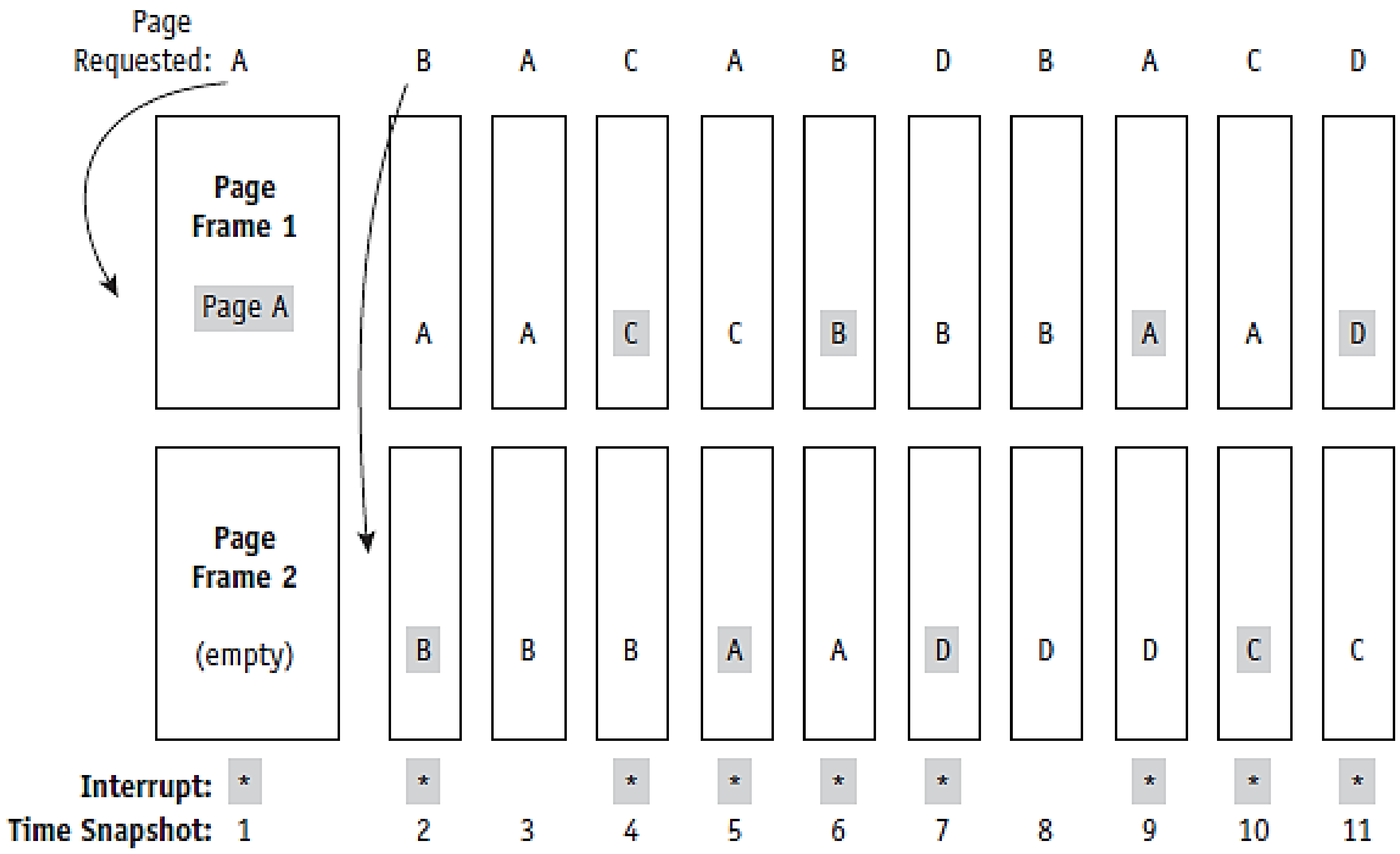
- Page replacement policy
  - Crucial to system efficiency
- Two well-known algorithms
  - First-in first-out (FIFO) policy
    - Best page to remove: page in memory longest
  - Least Recently Used (LRU) policy
    - Best page to remove: page least recently accessed

First-In First-Out

- Removes page: longest in memory
- Failure rate
  - Ratio of page interrupts to page requests
- More memory: does not guarantee better performance



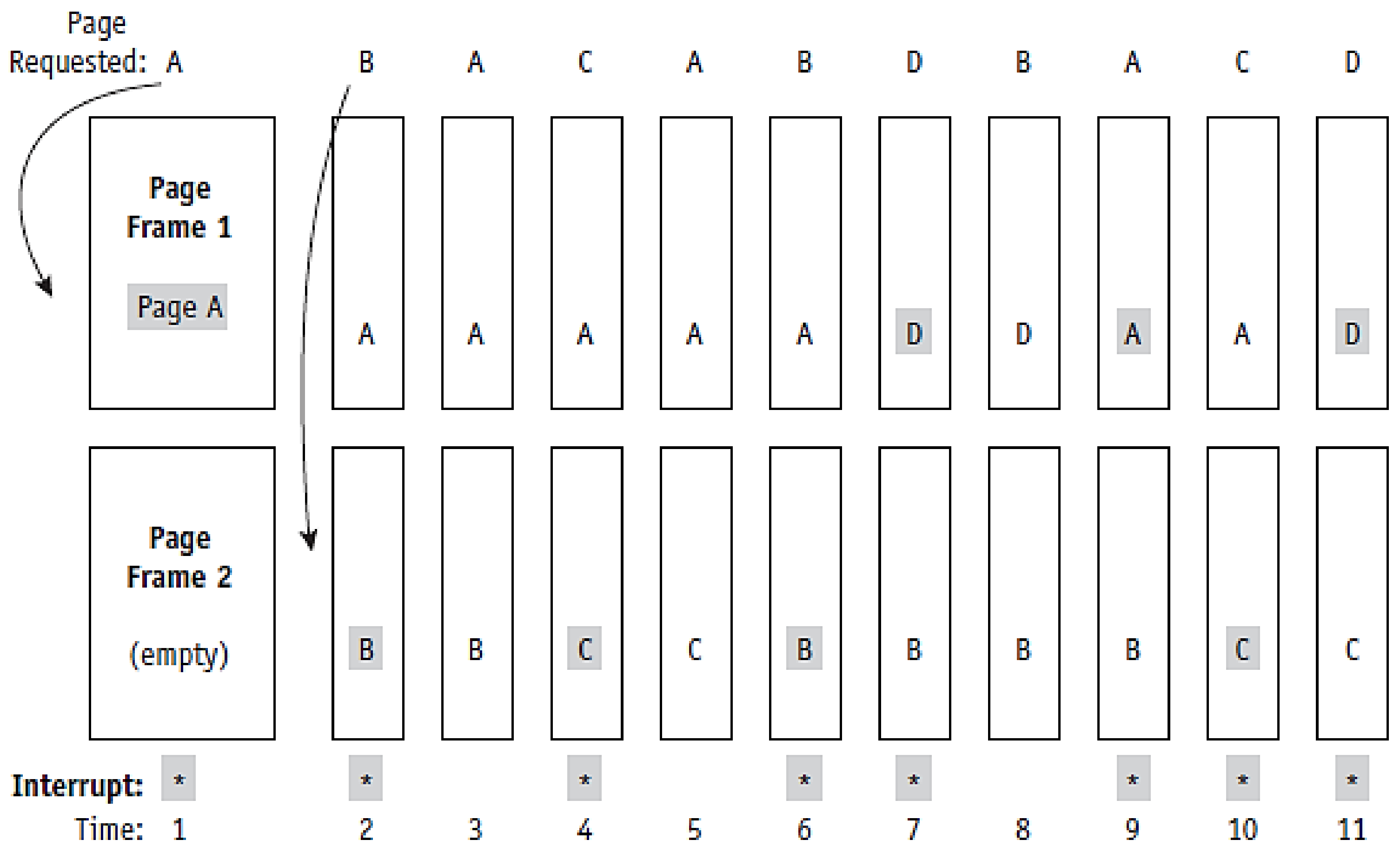
(figure 3.7)
First, Pages A and B are loaded into the two available page frames. When Page C is needed, the first page frame is emptied so C can be placed there. Then Page B is swapped out so Page A can be loaded there.
© Cengage Learning 2014


Page Requested: A B A C A B D B A C D
Page Frame 1
Page A
A A C C B B B A A D
Page Frame 2
(empty)
B B B A A D D D C C
Interrupt: * * * * * * * * *
Time Snapshot: 1 2 3 4 5 6 7 8 9 10 11

(figure 3.8)
Using a FIFO policy, this page trace analysis shows how each page requested is swapped into the two available page frames. When the program is ready to be processed, all four pages are in secondary storage. When the program calls a page that isn't already in memory, a page interrupt is issued, as shown by the gray boxes and asterisks. This program resulted in nine page interrupts.
© Cengage Learning 2014

Least Recently Used

- Removes page: least recent activity
  - Theory of locality
- Efficiency
  - Additional main memory: causes either decrease in or same number of interrupts
  - Does not experience FIFO Anomaly (Belady Anomaly)


Page Requested: A B A C A B D B A C D
Page Frame 1
Page A
A A A A A D D A A D
Page Frame 2
(empty)
B B C C B B B B C C
Interrupt: * * * * * * * *
Time: 1 2 3 4 5 6 7 8 9 10 11

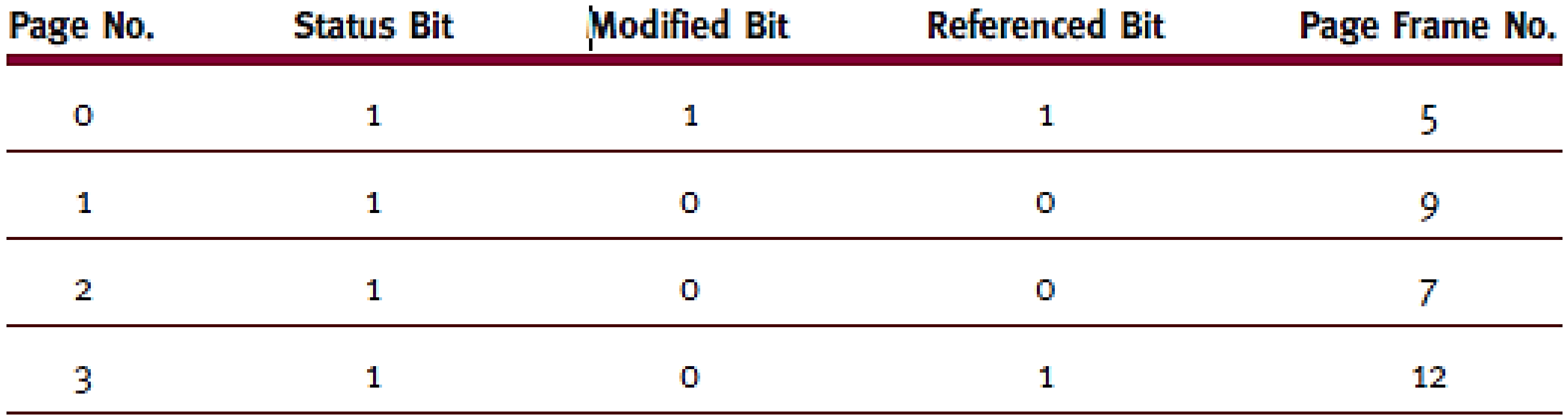
(figure 3.9)
Memory management using an LRU page removal policy for the program shown in Figure 3.8. Throughout the program, 11 page requests are issued, but they cause only 8 page interrupts.
© Cengage Learning 2014

Least Recently Used (cont'd.)

- Clock replacement variation
  - Circular queue: pointer steps through active pages' reference bits; simulates a clockwise motion
  - Pace: computer's clock cycle
- Bit-shifting variation
  - 8-bit reference byte and bit-shifting technique: tracks pages' usage (currently in memory)

The Mechanics of Paging

- Page swapping
  - Memory manager requires specific information: Page Map Table



Page No.	Status Bit	Modified Bit	Referenced Bit	Page Frame No.
0	1	1	1	5
1	1	0	0	9
2	1	0	0	7
3	1	0	1	12

(table 3.3)
Page Map Table for Job 1 shown in Figure 3.5.
A 1 = Yes and 0 = No.
© Cengage Learning 2014

The Mechanics of Paging (cont'd.)

- Page Map Table: bit meaning
  - Status bit: page currently in memory
  - Referenced bit: page referenced recently
    - Determines page to swap: LRU algorithm
  - Modified bit: page contents altered
    - Determines if page must be rewritten to secondary storage when swapped out
- Bits checked when swapping
  - FIFO: modified and status bits
  - LRU: all bits (status, modified, and reference bits)

Status Bit		Modified Bit		Referenced Bit	
Value	Meaning	Value	Meaning	Value	Meaning
0	*not* in memory	0	*not* modified	0	*not* called
1	*resides* in memory	1	*was* modified	1	*was* called

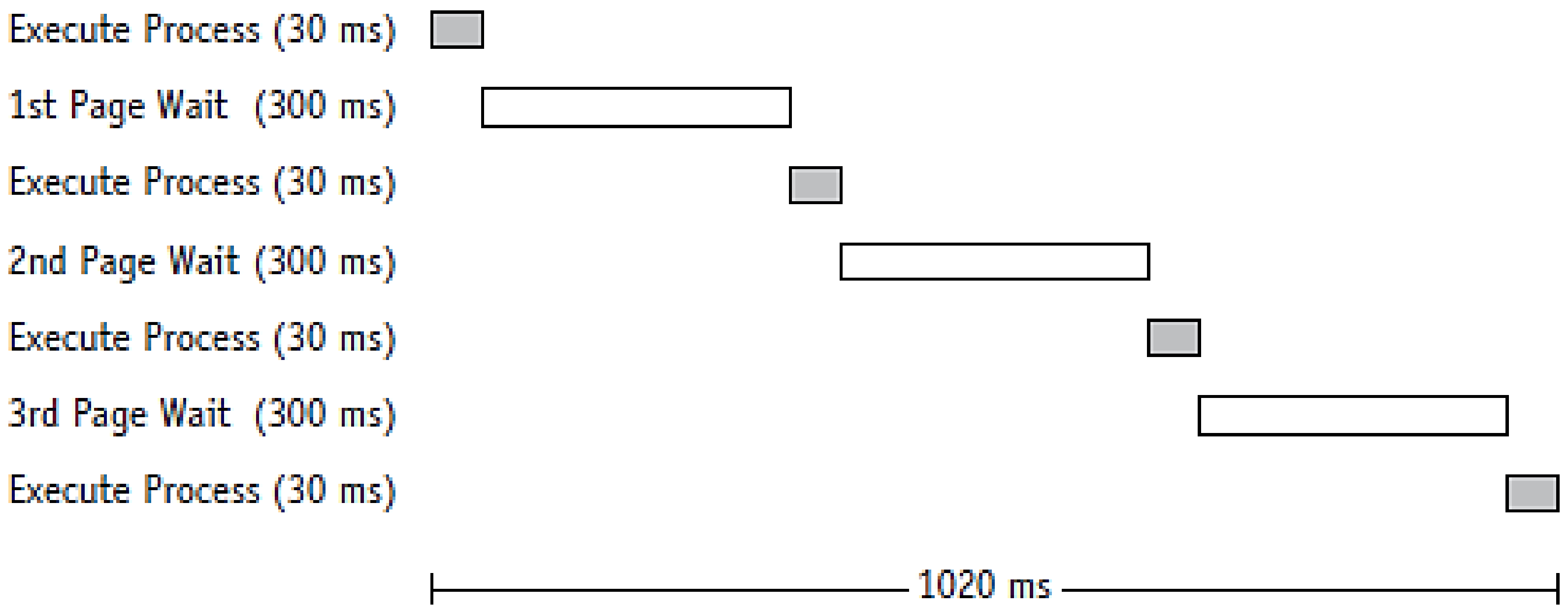
(table 3.4)
The meaning of these bits used in the Page Map Table.
© Cengage Learning 2014

	Modified?	Referenced?	What it Means
Case 1	0	0	Not modified AND not referenced
Case 2	0	1	Not modified BUT was referenced
Case 3	1	0	Was modified BUT not referenced [Impossible?]
Case 4	1	1	Was modified AND was referenced

(table 3.5)
Four possible combinations of modified and referenced bits and the meaning of each.
© Cengage Learning 2014

The Working Set

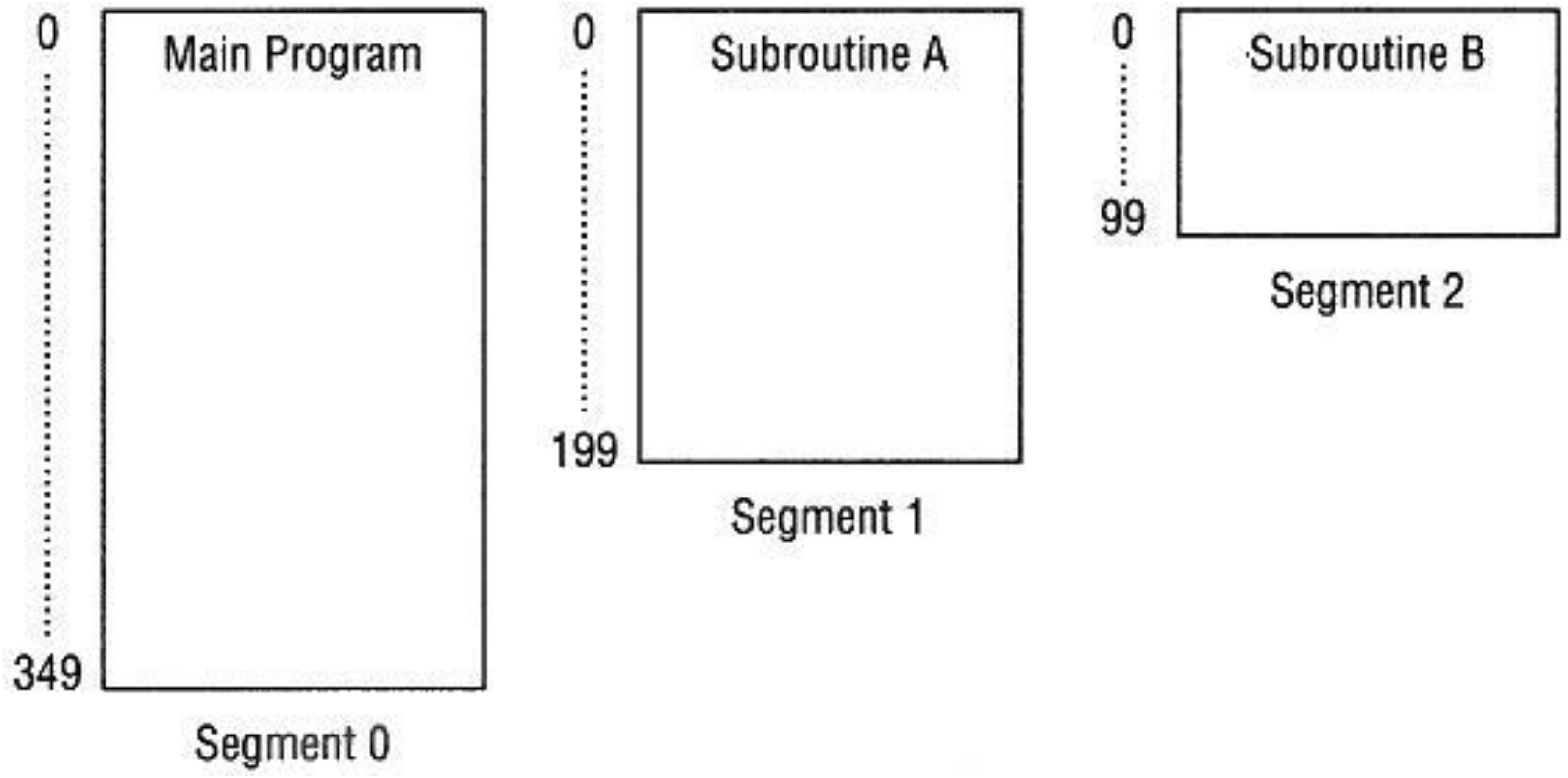
- Set of pages residing in memory: accessed directly without incurring a page fault
  - Demand paging schemes: improves performance
- Requires “locality of reference” concept
  - Structured programs: only small fraction of pages needed during any execution phase
- System needs definitive values:
  - Number of pages comprising working set
  - Maximum number of pages allowed for a working set
- Time-sharing and network systems
  - Must track every working set’s size and identity



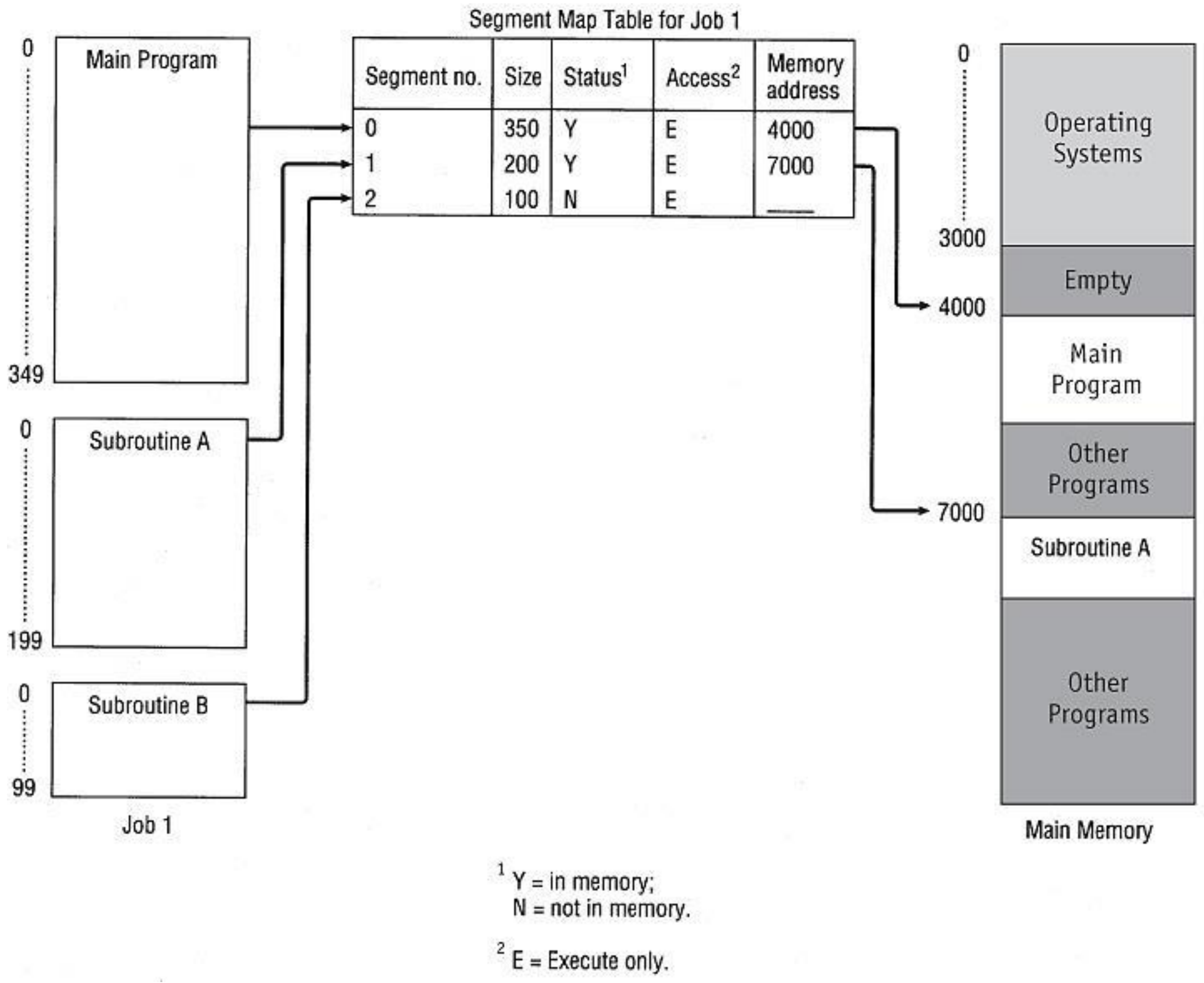
(figure 3.13)
Time line showing the amount of time required to process page faults for a single program. The program in this example takes 120 milliseconds (ms) to execute but an additional 900 ms to load the necessary pages into memory. Therefore, job turnaround is 1020 ms.
© Cengage Learning 2014

Segmented Memory Allocation

- Each job divided into several segments: different sizes
  - One segment for each module: related functions
- Reduces page faults
  - Loops: not split over two or more pages
- Main memory: allocated dynamically
- Program’s structural modules: determine segments
  - Each segment numbered when program compiled/assembled
  - Segment Map Table (SMT) generated



(figure 3.14)
Segmented memory allocation. Job 1 includes a main program and two subroutines. It is a single job that is structurally divided into three segments of different sizes.
© Cengage Learning 2014



Segment Map Table for Job 1

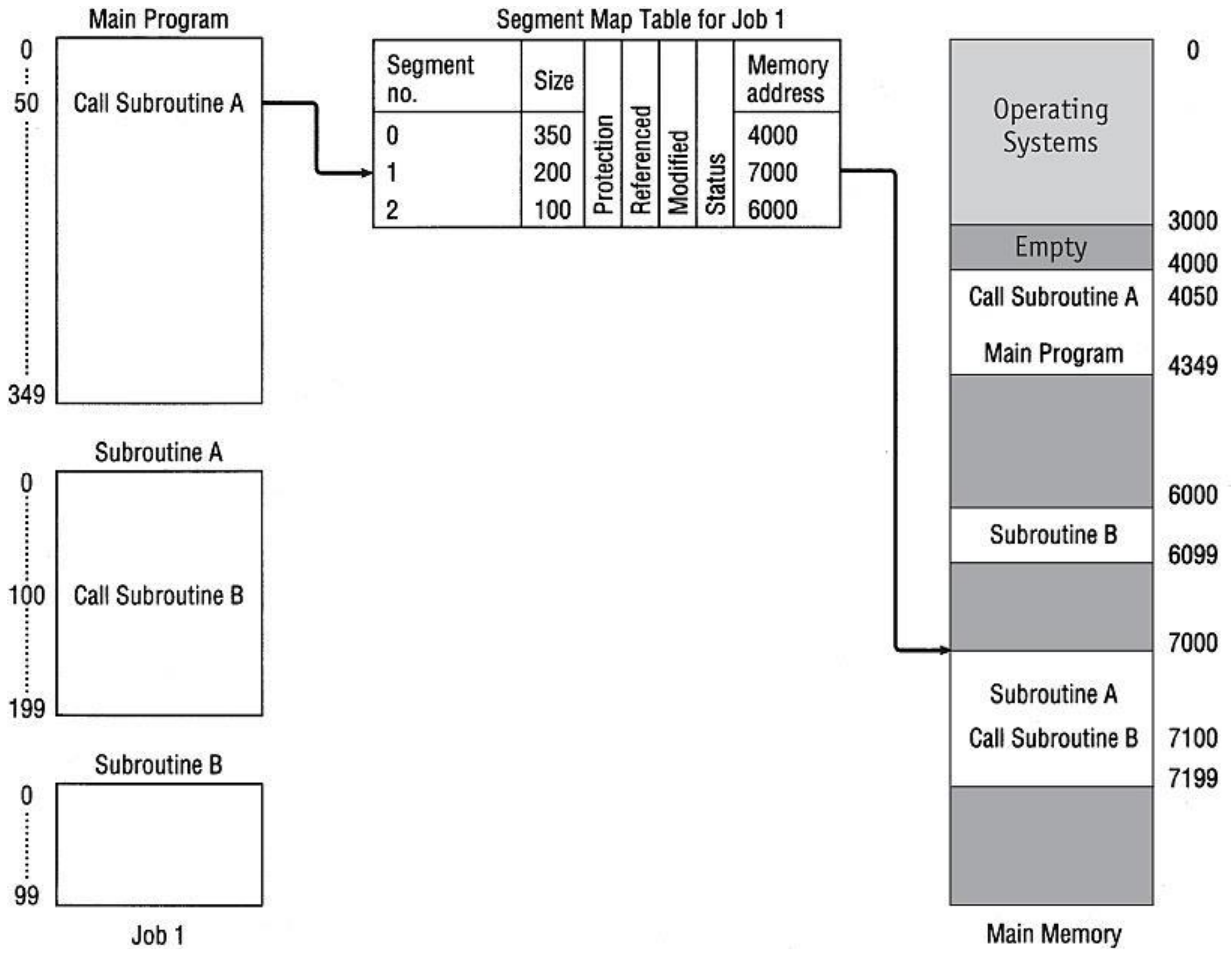
Segment no.	Size	Status[1]	Access[2]	Memory address
0	350	Y	E	4000
1	200	Y	E	7000
2	100	N	E	___

[1] Y = in memory;
N = not in memory.

[2] E = Execute only.

(figure 3.15)
The Segment Map Table tracks each segment for this job. Notice that Subroutine B has not yet been loaded into memory.
© Cengage Learning 2014

(figure 3.16)
During execution, the main program calls Subroutine A, which triggers the SMT to look up its location in memory.
© Cengage Learning 2014



Segment Map Table for Job 1

Segment no.	Size	Protection	Referenced	Modified	Status	Memory address
0	350					4000
1	200					7000
2	100					6000

Segmented Memory Allocation (cont'd.)

- Memory Manager: tracks segments in memory
  - Job Table: one for whole system
    - Every job in process
  - Segment Map Table: one for each job
    - Details about each segment
  - Memory Map Table: one for whole system
    - Main memory allocation
- Instructions within each segment: ordered sequentially
- Segments: not necessarily stored contiguously

Segmented Memory Allocation (cont'd.)

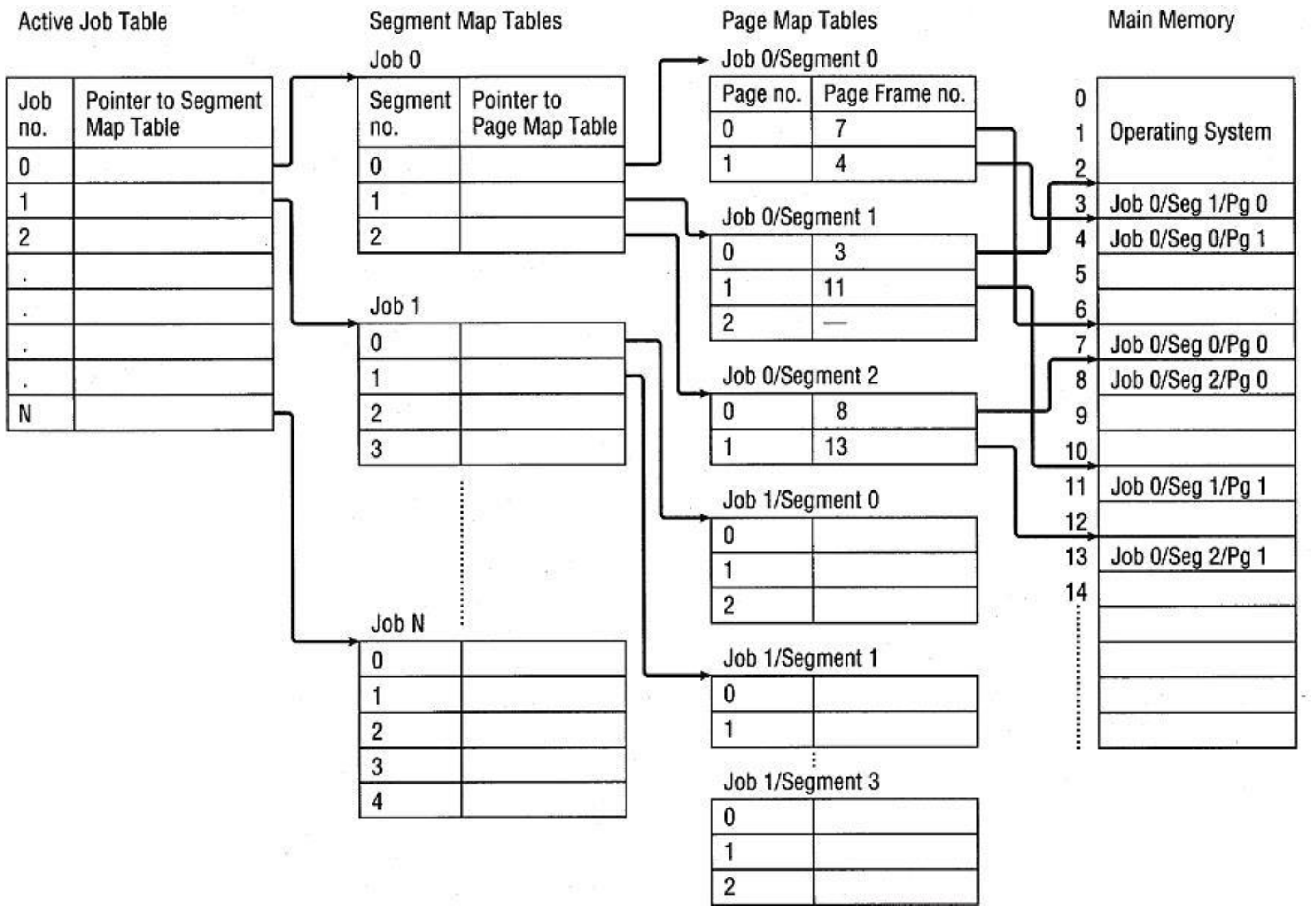
- Two-dimensional addressing scheme
  - Segment number and displacement
- Disadvantage
  - External fragmentation
- Major difference between paging and segmentation
  - Pages: physical units; invisible to the program
  - Segments: logical units; visible to the program; variable sizes

Segmented/Demand Paged Memory Allocation

- Subdivides segments: equal-sized pages
  - Smaller than most segments
  - More easily manipulated than whole segments
  - Segmentation's logical benefits
  - Paging's physical benefits
- Segmentation problems removed
  - Compaction, external fragmentation, secondary storage handling
- Three-dimensional addressing scheme
  - Segment number, page number (within segment), and displacement (within page)

Segmented/Demand Paged Memory Allocation (cont'd.)

- Scheme requires four tables
  - Job Table: one for the whole system
    - Every job in process
  - Segment Map Table: one for each job
    - Details about each segment
  - Page Map Table: one for each segment
    - Details about every page
  - Memory Map Table: one for the whole system
    - Monitors main memory allocation: page frames



(figure 3.17)
How the Job Table, Segment Map Table, Page Map Table, and main memory interact in a segment/paging scheme.
© Cengage Learning 2014

Segmented/Demand Paged Memory Allocation (cont'd.)

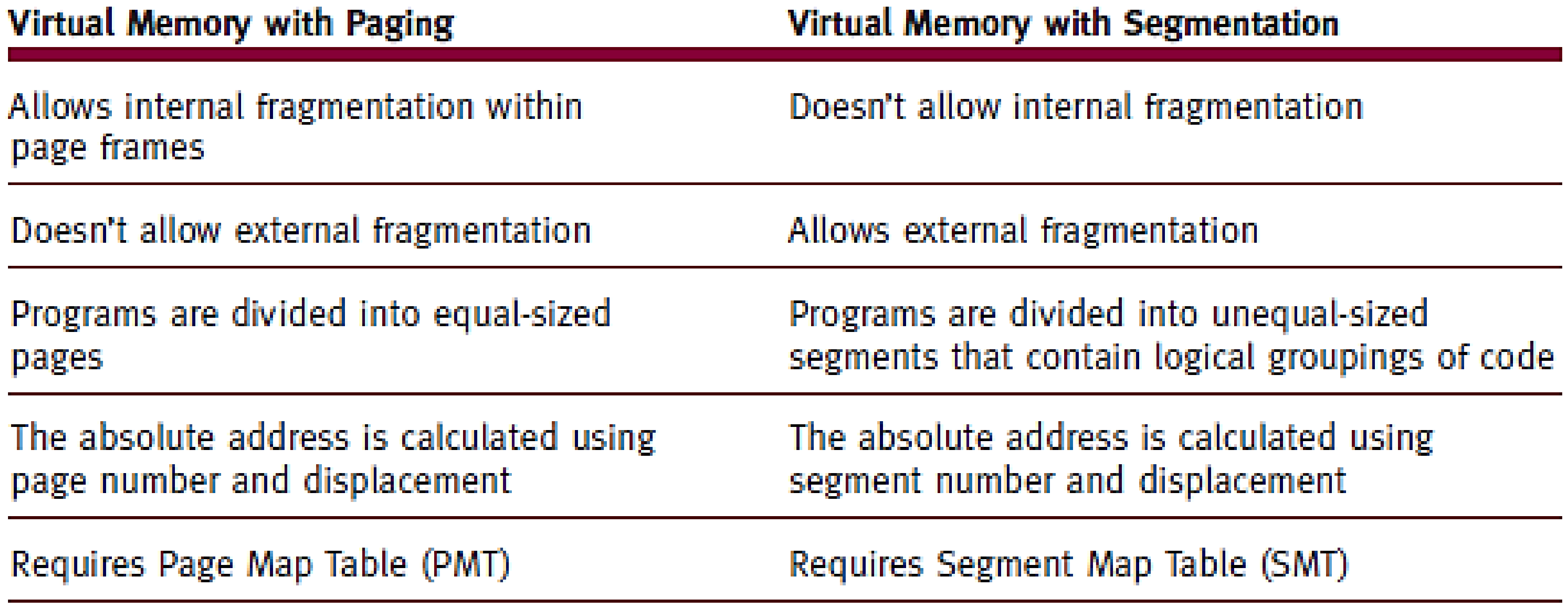
- Disadvantages
  - Overhead: managing the tables
  - Time required: referencing tables
- Associative memory
  - Several registers allocated to each job
    - Segment and page numbers: associated with main memory
  - Page request: initiates two simultaneous searches
    - Associative registers
    - SMT and PMT

Segmented/Demand Paged Memory Allocation (cont'd.)

- Associative memory
  - Primary advantage (large associative memory)
    - Increased speed
  - Disadvantage
    - High cost of complex hardware

Virtual Memory

- Made possible by swapping pages in/out of memory
- Program execution: only a portion of the program in memory at any given moment
- Requires cooperation between:
  - Memory Manager: tracks each page or segment
  - Processor hardware: issues the interrupt and resolves the virtual address



Virtual Memory with Paging	Virtual Memory with Segmentation
Allows internal fragmentation within page frames	Doesn't allow internal fragmentation
Doesn't allow external fragmentation	Allows external fragmentation
Programs are divided into equal-sized pages	Programs are divided into unequal-sized segments that contain logical groupings of code
The absolute address is calculated using page number and displacement	The absolute address is calculated using segment number and displacement
Requires Page Map Table (PMT)	Requires Segment Map Table (SMT)

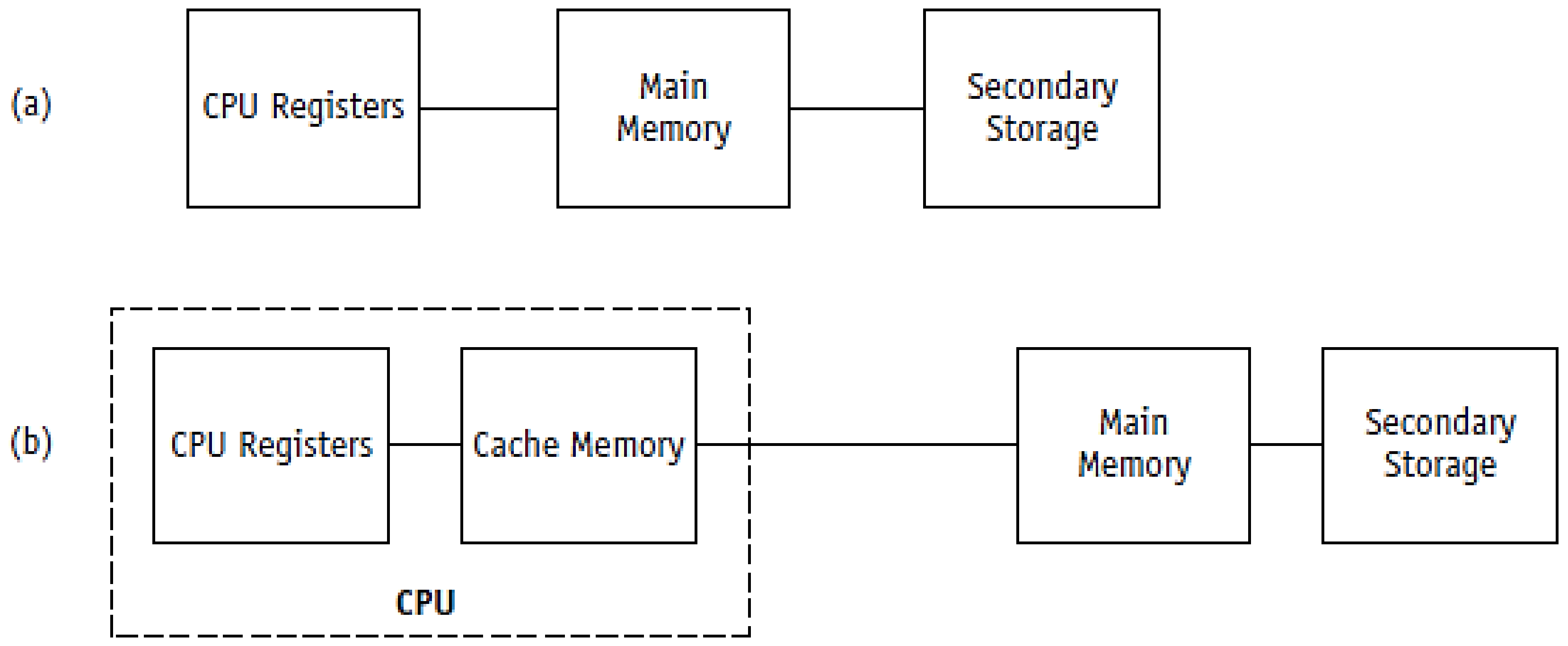
(table 3.6)
Comparison of the advantages and disadvantages of virtual memory with paging and segmentation.
© Cengage Learning 2014

Virtual Memory (cont'd.)

- Advantages
  - Job size: not restricted to size of main memory
  - More efficient memory use
  - Unlimited amount of multiprogramming possible
  - Code and data sharing allowed
  - Dynamic linking of program segments facilitated
- Disadvantages
  - Higher processor hardware costs
  - More overhead: handling paging interrupts
  - Increased software complexity: prevent thrashing

Cache Memory

- Small, high-speed intermediate memory unit
- Computer system's performance increased
  - Faster processor access compared to main memory
  - Stores frequently used data and instructions
- Cache levels
  - L2: connected to CPU; contains copy of bus data
  - L1: pair built into CPU; stores instructions and data
- Data/instructions: move between main memory and cache
  - Methods similar to paging algorithms



(figure 3.19)
Comparison of (a) the traditional path used by early computers between main memory and the CPU and (b) the path used by modern computers to connect the main memory and the CPU via cache memory.
© Cengage Learning 2014

Cache Memory (cont'd.)

- Four cache memory design factors
  - Cache size, block size, block replacement algorithm, and rewrite policy
- Optimal cache and replacement algorithm
  - 80-90% of all requests in cache possible

Cache Memory (cont'd.)

- Cache hit ratio

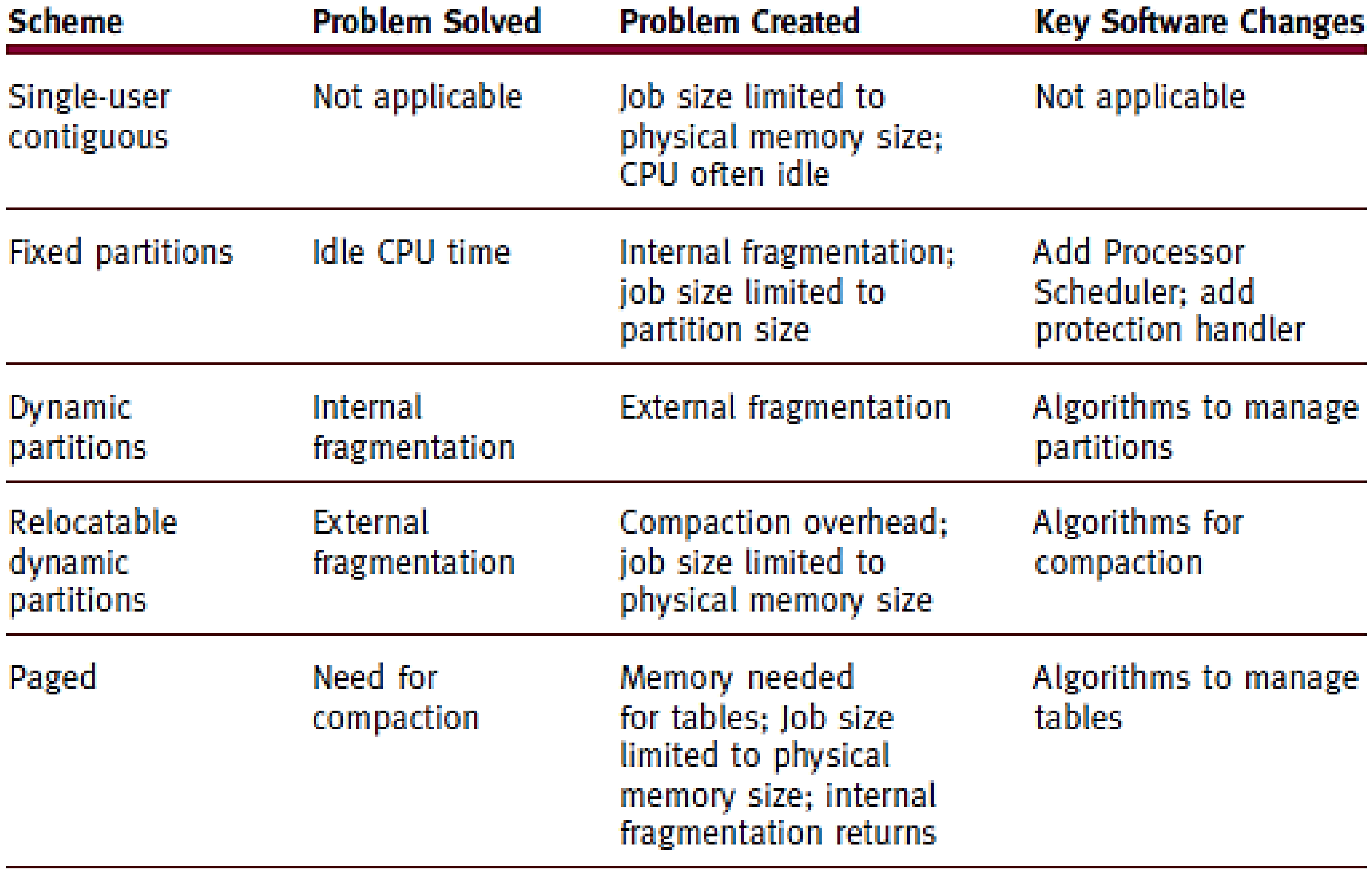
  $$HitRatio = \frac{number\ of\ requests\ found\ in\ the\ cache}{total\ number\ of\ requests} * 100$$

- Average memory access time

  $$Avg\_Mem\_AccTime = Avg\_Cache\_AccessTime + (1 - HitRatio) * Avg\_MainMem\_AccTime$$

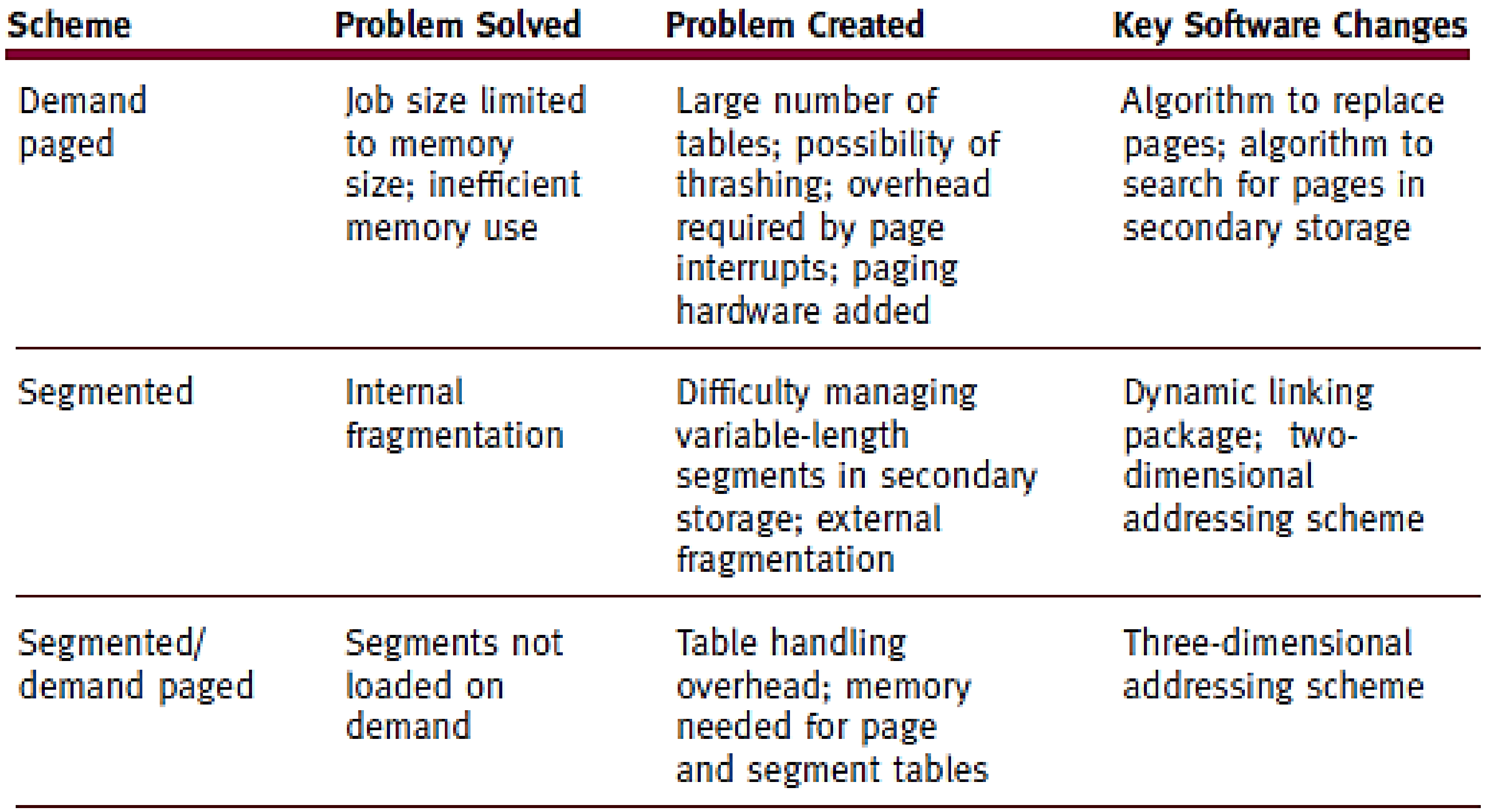
Summary

- Operating system: Memory Manager
  - Allocating memory storage: main memory, cache memory, and registers
  - Deallocating memory: execution completed



Scheme	Problem Solved	Problem Created	Key Software Changes
Single-user contiguous	Not applicable	Job size limited to physical memory size; CPU often idle	Not applicable
Fixed partitions	Idle CPU time	Internal fragmentation; job size limited to partition size	Add Processor Scheduler; add protection handler
Dynamic partitions	Internal fragmentation	External fragmentation	Algorithms to manage partitions
Relocatable dynamic partitions	External fragmentation	Compaction overhead; job size limited to physical memory size	Algorithms for compaction
Paged	Need for compaction	Memory needed for tables; Job size limited to physical memory size; internal fragmentation returns	Algorithms to manage tables

(table 3.7)
The big picture. Comparison of the memory allocation schemes discussed in Chapters 2 and 3.
© Cengage Learning 2014



Scheme	Problem Solved	Problem Created	Key Software Changes
Demand paged	Job size limited to memory size; inefficient memory use	Large number of tables; possibility of thrashing; overhead required by page interrupts; paging hardware added	Algorithm to replace pages; algorithm to search for pages in secondary storage
Segmented	Internal fragmentation	Difficulty managing variable-length segments in secondary storage; external fragmentation	Dynamic linking package; two-dimensional addressing scheme
Segmented/ demand paged	Segments not loaded on demand	Table handling overhead; memory needed for page and segment tables	Three-dimensional addressing scheme

(table 3.7) (cont'd.)
The big picture. Comparison of the memory allocation schemes discussed in Chapters 2 and 3.
© Cengage Learning 2014