Chapter 3
Memory Management: Virtual Memory

Understanding Operating Systems, Fourth Edition

Objectives

You will 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
- The difference between a first-in first-out page replacement policy, a least-recently-used page replacement policy, and a clock page replacement policy

Objectives (continued)

You will be able to describe:

- The mechanics of paging and how a memory allocation scheme determines which pages should be swapped out of memory
- The concept of the working set and how it is used in memory allocation schemes
- The impact that virtual memory had on multiprogramming
- Cache memory and its role in improving system response time

Memory Management: Virtual Memory

- **Disadvantages of early schemes:**
  - Required storing entire program in memory
  - Fragmentation
  - Overhead due to relocation
- **Evolution of virtual memory helps to:**
  - Remove the restriction of storing programs contiguously
  - Eliminate the need for entire program to reside in memory during execution

Paged Memory Allocation

- Divides each incoming job into pages of equal size
- Works well if page size, memory block size (page frames), and size of disk section (sector, block) are all equal
- Before executing a program, Memory Manager:
  - Determines number of pages in program
  - Locates enough empty page frames in main memory
  - Loads all of the program’s pages into them

Paged Memory Allocation (continued)

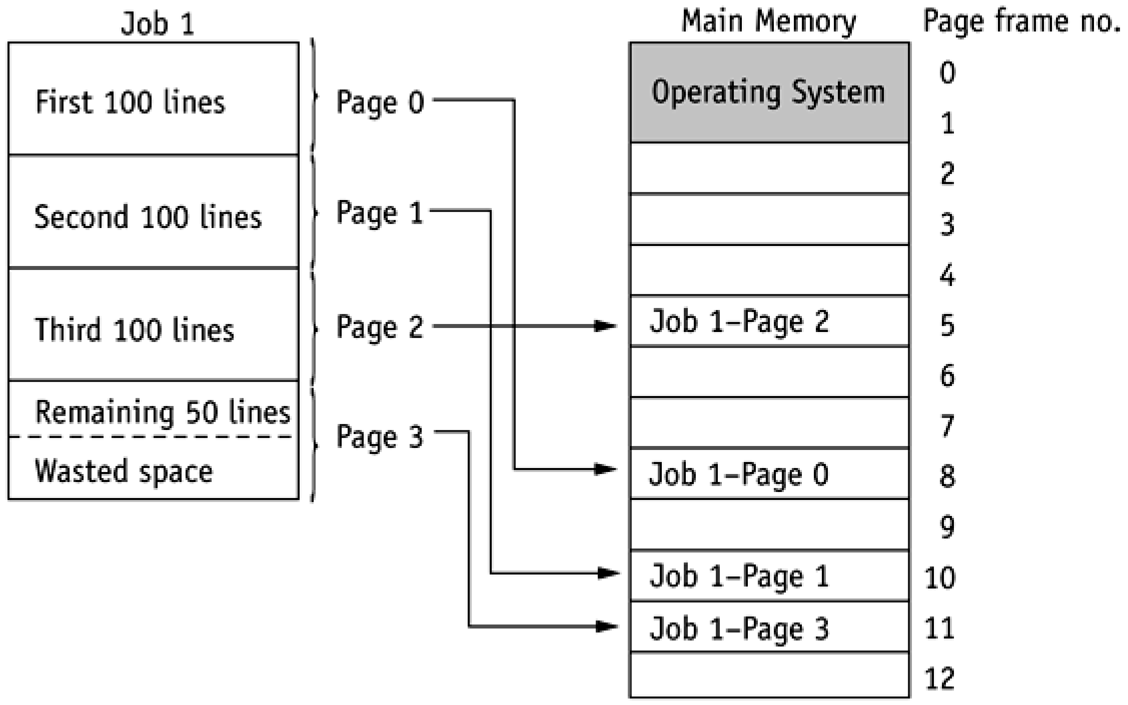
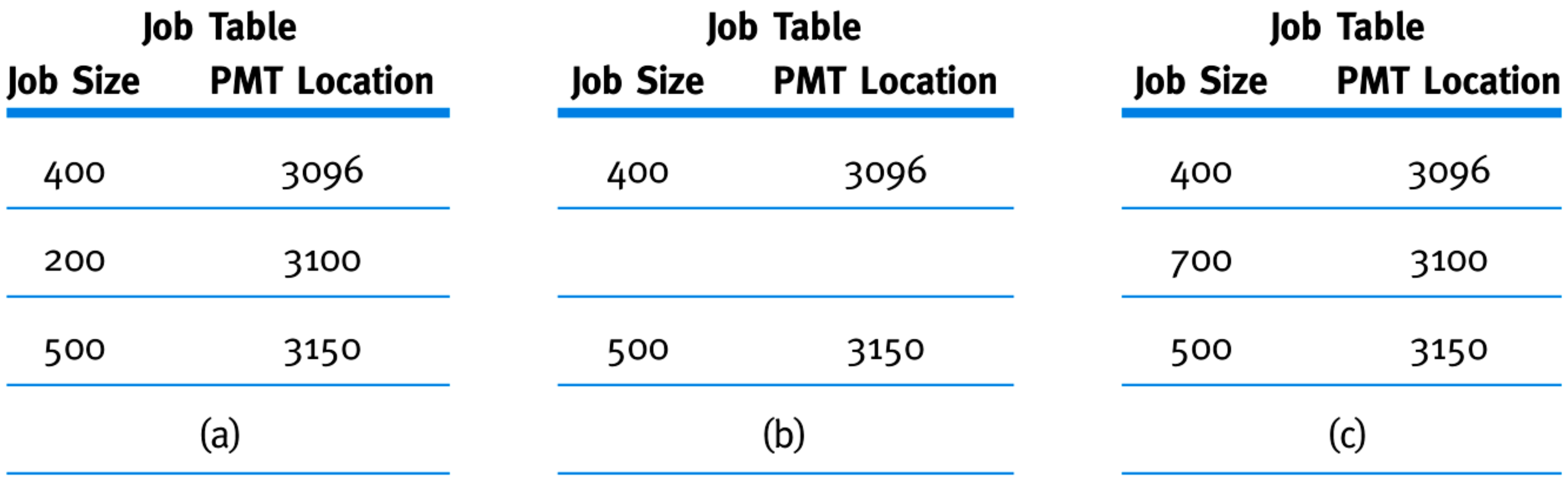


Figure 3.1: Paged memory allocation scheme for a job of 350 lines

Paged Memory Allocation (continued)

- Memory Manager requires three tables to keep track of the job’s pages:
  - **Job Table (JT)** contains information about
    - Size of the job
    - Memory location where its PMT is stored
  - **Page Map Table (PMT)** contains information about
    - Page number and its corresponding page frame memory address
  - **Memory Map Table (MMT) c**ontains
    - Location for each page frame
    - Free/busy status

Paged Memory Allocation (continued)



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

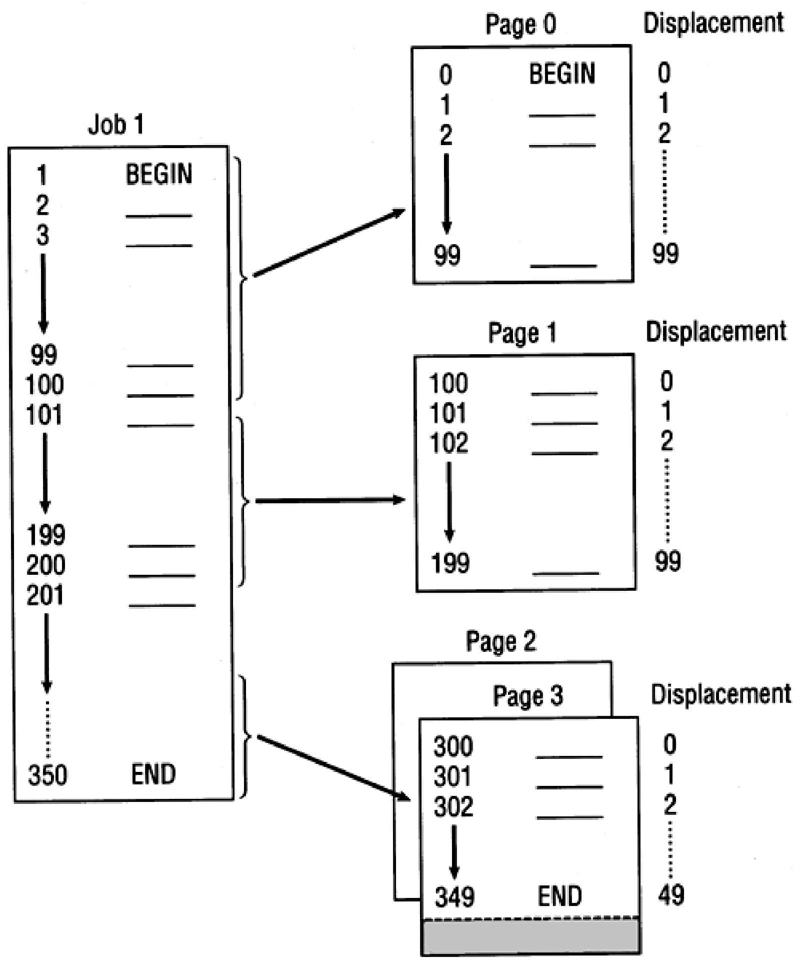
Job Size	PMT Location
400	3096
700	3100
500	3150

(c)

Table 3.1: A Typical Job Table

(a) initially has three entries, one for each job in process. When the second job (b) ends, its entry in the table is released and it is replaced by (c), information about the next job that is processed

Paged Memory Allocation (continued)



Job 1 is 350 lines long and is divided into four pages of 100 lines each.

Figure 3.2: Paged Memory Allocation Scheme

Paged Memory Allocation (continued)

- **Displacement (offset)** of a line: Determines how far away a line is from the beginning of its page
  - Used to locate that line within its page frame
- How to determine page number and displacement of a line:
  - **Page number** = the integer quotient from the division of the job space address by the page size
  - **Displacement** = the remainder from the page number division

Paged Memory Allocation (continued)

- **Steps to determine exact location of a line in memory:**
  - Determine page number and displacement of a line
  - Refer to the job's PMT and find out which page frame contains the required page
  - Get the address of the beginning of the page frame by multiplying the page frame number by the page frame size
  - Add the displacement (calculated in step 1) to the starting address of the page frame

Paged Memory Allocation (continued)

- **Advantages:**
  - Allows jobs to be allocated in noncontiguous memory locations
    - Memory used more efficiently; more jobs can fit
- **Disadvantages:**
  - Address resolution causes increased overhead
  - Internal fragmentation still exists, though in last page
  - Requires the entire job to be stored in memory location
  - Size of page is crucial (not too small, not too large)

Demand Paging

- **Demand Paging:** Pages are brought into memory only as they are needed, allowing jobs to be run with less main memory
- Takes advantage that programs are written sequentially so not all pages are necessary at once. For example:
  - User-written error handling modules are processed only when a specific error is detected
  - Mutually exclusive modules
  - Certain program options are not always accessible

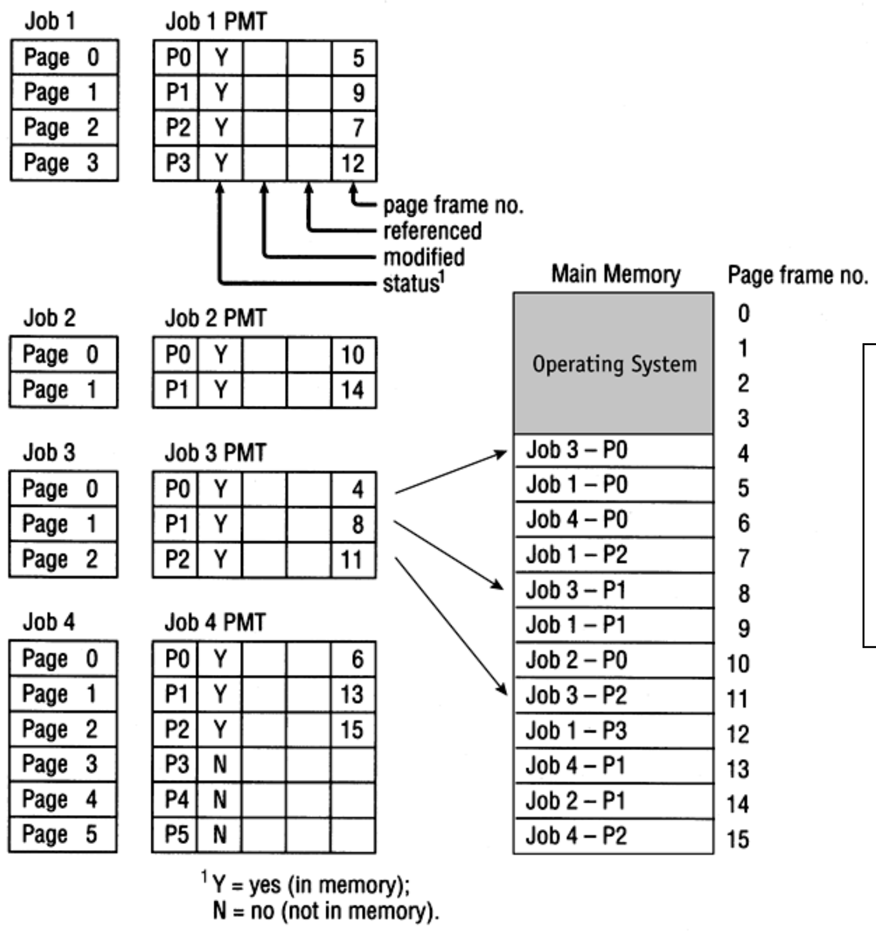
Demand Paging (continued)

- Demand paging made **virtual memory** widely available
  - Can give appearance of an almost infinite or nonfinite amount of physical memory
- Allows the user to run jobs with less main memory than required in paged memory allocation
- Requires use of a high-speed direct access storage device that can work directly with CPU
- How and when the pages are passed (or “swapped”) depends on predefined policies

Demand Paging (continued)

- The OS depends on following tables:
  - **Job Table**
  - **Page Map Table** with 3 new fields to determine
    - If requested page is already in memory
    - If page contents have been modified
    - If the page has been referenced recently
      - Used to determine which pages should remain in main memory and which should be swapped out
  - **Memory Map Table**

Demand Paging (continued)



Total job pages are 15, and the number of total available page frames is 12.

Figure 3.5: A typical demand paging scheme

Demand Paging (continued)

- **Swapping Process:**
  - To move in a new page, a resident page must be swapped back into secondary storage; involves
    - Copying the resident page to the disk (if it was modified)
    - Writing the new page into the empty page frame
  - Requires close interaction between hardware components, software algorithms, and policy schemes

Demand Paging (continued)

- **Page fault handler:** The section of the operating system that determines
  - Whether there are empty page frames in memory
    - If so, requested page is copied from secondary storage
  - Which page will be swapped out if all page frames are busy
    - Decision is directly dependent on the predefined policy for page removal

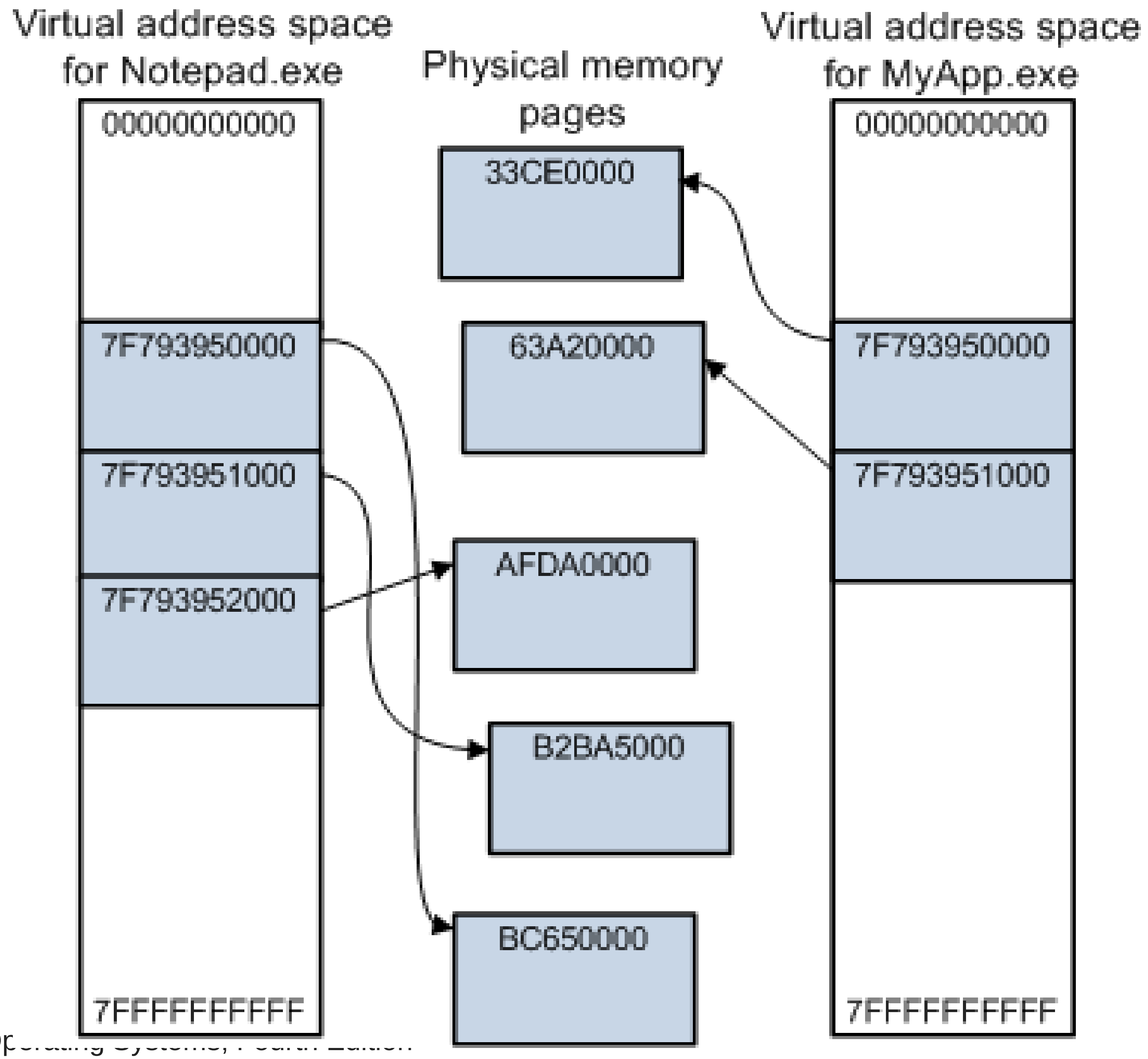
Demand Paging (continued)

- **Thrashing :** An excessive amount of page swapping between main memory and secondary storage
  - Operation becomes inefficient
  - Caused when a page is removed from memory but is called back shortly thereafter
  - Can occur across jobs, when a large number of jobs are vying for a relatively few number of free pages
  - Can happen within a job (e.g., in loops that cross page boundaries)
- **Page fault:** a failure to find a page in memory

Demand Paging (continued)

- **Advantages:**
  - Job no longer constrained by the size of physical memory (concept of virtual memory)
  - Utilizes memory more efficiently than the previous schemes
- **Disadvantages:**
  - Increased overhead caused by the tables and the page interrupts

Virtual address and physical address



Page Replacement Policies and Concepts

- Policy that selects the page to be removed; crucial to system efficiency. Types include:
  - **First-in first-out (FIFO) policy:** Removes page that has been in memory the longest
  - **Least-recently-used (LRU) policy:** Removes page that has been least recently accessed
  - **Most recently used (MRU) policy**
  - **Least frequently used (LFU) policy**

Page Replacement Policies and Concepts (continued)

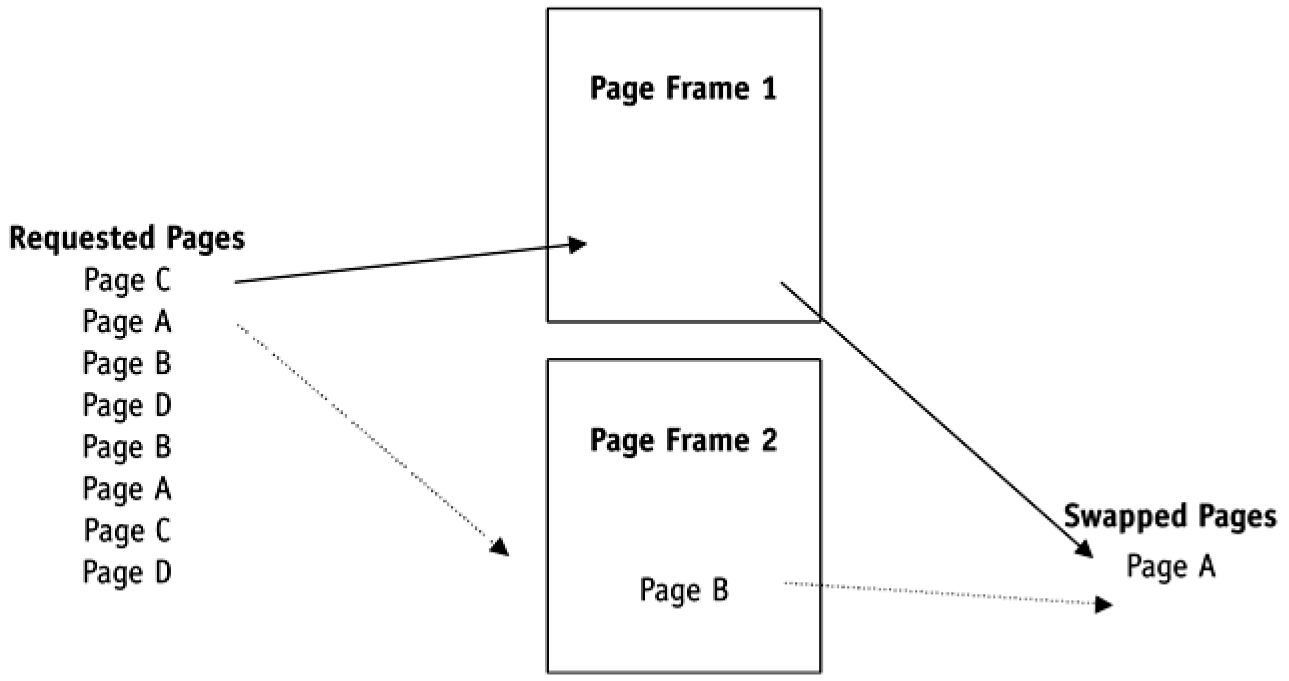


Figure 3.7: FIFO Policy

Page Replacement Policies and Concepts (continued)

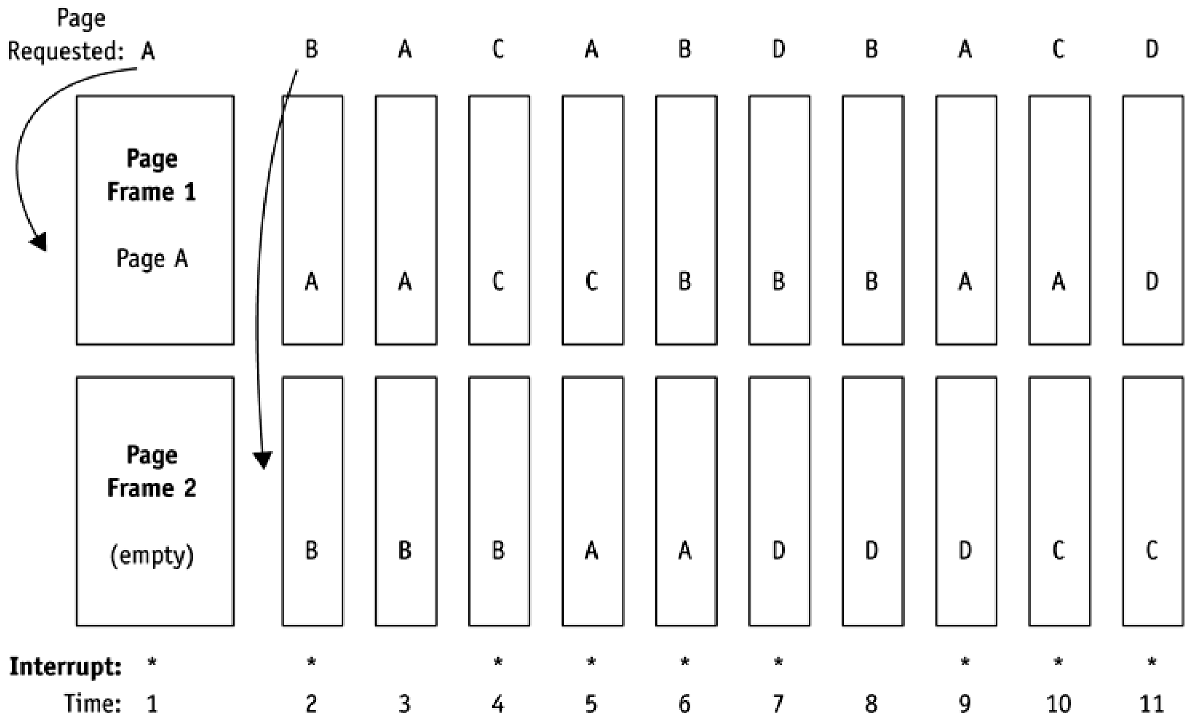

Page Requested: A B A C A B D B A C D
Page Frame 1
Page A
A A C C B B B A A D
Page Frame 2
(empty)
B B B A A D D D C C
Interrupt: * * * * * * * * *
Time: 1 2 3 4 5 6 7 8 9 10 11

Figure 3.8: Working of a FIFO algorithm for a job with four pages (A, B, C, D) as it's processed by a system with only two available page frames

Page Replacement Policies and Concepts (continued)

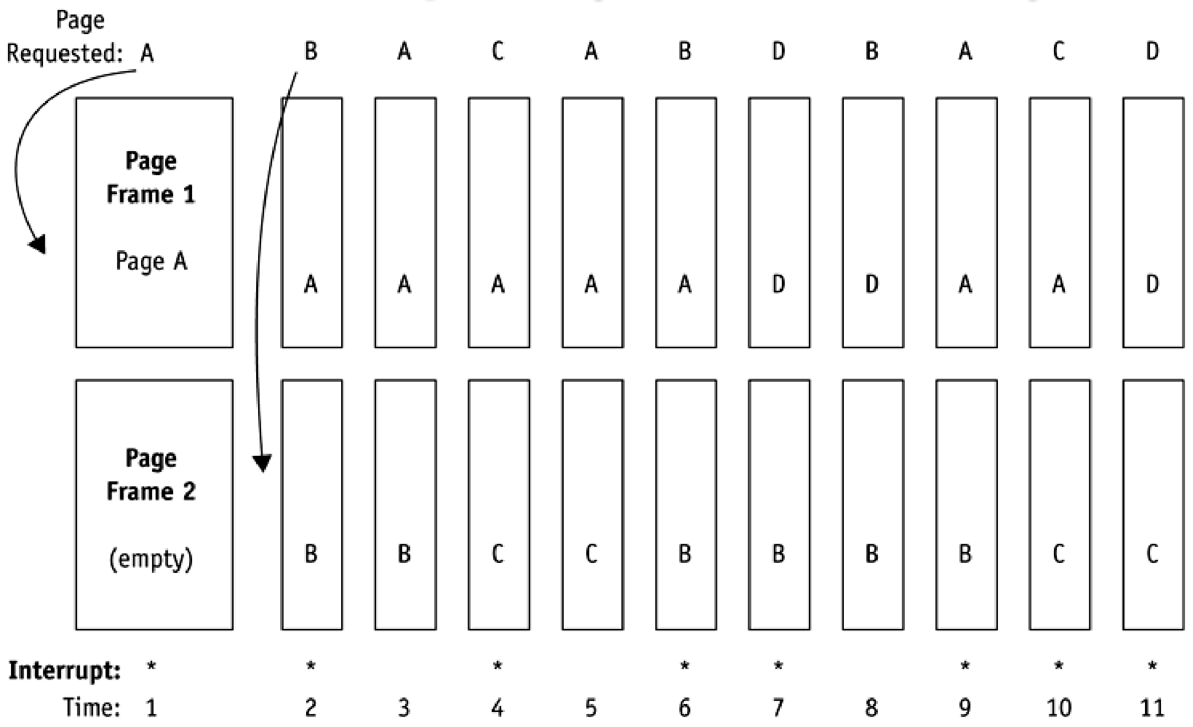


Figure 3.9: Working of an LRU algorithm for a job with four pages (A, B, C, D) as it's processed by a system with only two available page frames

Page Replacement Policies and Concepts (continued)

- Efficiency (ratio of page interrupts to page requests) is slightly better for LRU as compared to FIFO
- **FIFO anomaly:** No guarantee that buying more memory will always result in better performance
- In LRU case, increasing main memory will cause either decrease in or same number of interrupts
- LRU uses an 8-bit reference byte and a bit-shifting technique to track the usage of each page currently in memory

FIFO Belady’s Anomaly

1. Usually Allocating more main memory page frames to a process results in less page faults.

1. In few cases; instead of getting less page faults, more page faults happen when more page frames are allocated to the same job.

2. Belady’s anomaly is the phenomenon where increasing page frames using FIFO strategy results in increase in number of page faults (instead of decrease) for a given memory pattern.

FIFO Belady's Anomaly

Time	1	2	3	4	5	6	7	8	9	10	11	12
Page Requests	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Page frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Page frame 2		2	2	2	1	1	1	1	1	3	3	3
Page frame 3			3	3	3	2	2	2	2	2	4	4
Page Fault	*	*	*	*	*	*	*	**Hit**	**Hit**	*	*	**Hit**

Page allocation using FIFO strategy for the page request pattern: 1 2 3 4 1 2 5 1 2 3 4 5

And using: 3 page frames

Page faults: 9 page faults.

FIFO Belady's Anomaly

Time	1	2	3	4	5	6	7	8	9	10	11	12
Page Request	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Page frame 1	1	1	1	1	1	1	5	5	5	5	4	4
Page frame 2		2	2	2	2	2	2	1	1	1	1	5
Page frame 3			3	3	3	3	3	3	2	2	2	2
Page frame 4				4	4	4	4	4	4	3	3	3
Page Fault	*	*	*	*	**Hit**	**Hit**	*	*	*	*	*	*

Page allocation using FIFO strategy for the page request pattern: 1 2 3 4 1 2 5 1 2 3 4 5

And using: 4 page frames

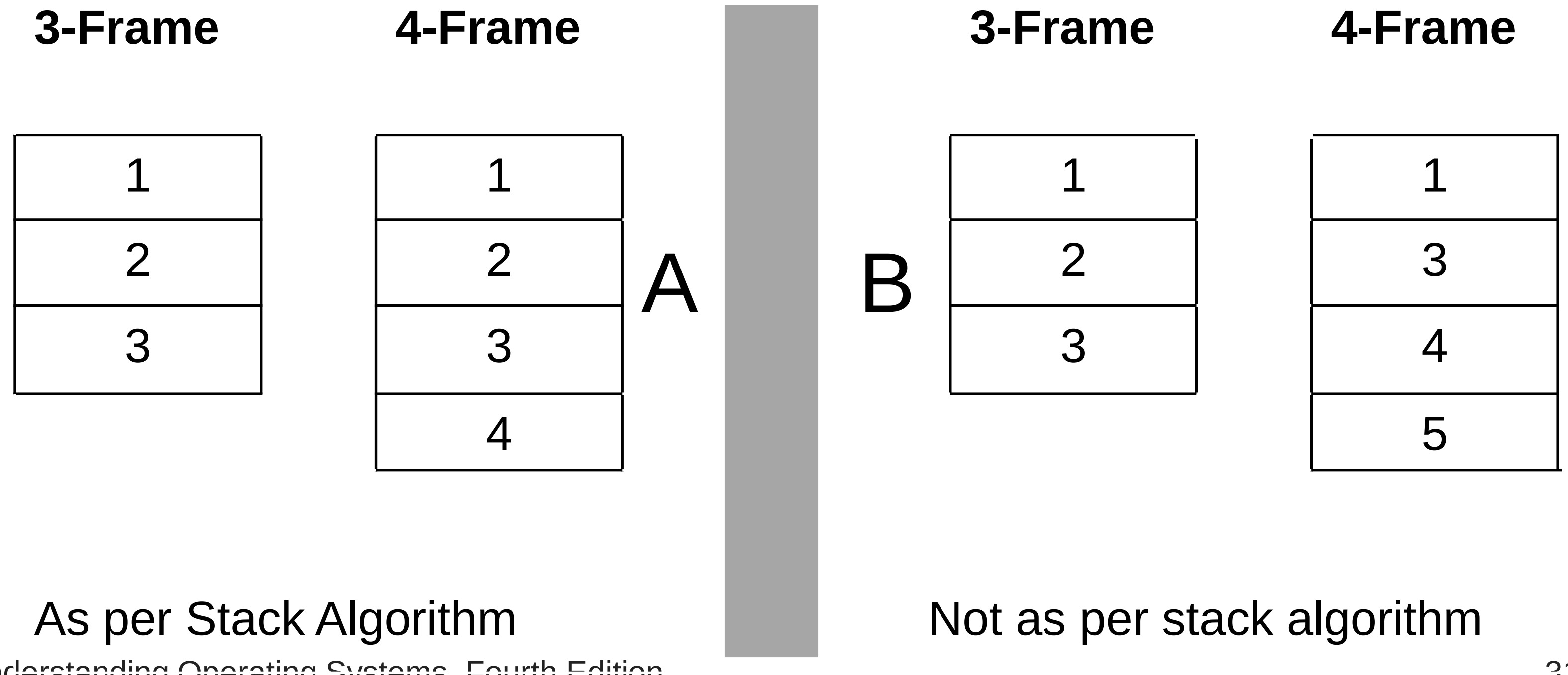
Page faults: 10 page faults.

Why FIFO Belady's Anomaly occurs?

- If at any point of time, the set of pages in the 3-frame memory is not a subset of the set in the 4-frame memory (if all pages in 3-fram are not in in pages in 4-frame) then this means the 4-frame memory will producing a page fault that does not occur in the 3-frame memory.
- If the pages in the frames of the smaller memory are also <u>all</u> in the frames of a larger memory, the algorithm is said to be a stack algorithm. Because a stack algorithm by definition prevents the discrepancy above, no stack algorithm can suffer from Belady's anomaly.

STACK ALGORITHM

- A page replacement policy/algorithm suffers from Belady's Anomaly's iff its does not follow "STACK ALGORITHM".
- LRU always follows the stack algorithm property. So it never suffers from Belady's Anamoly.

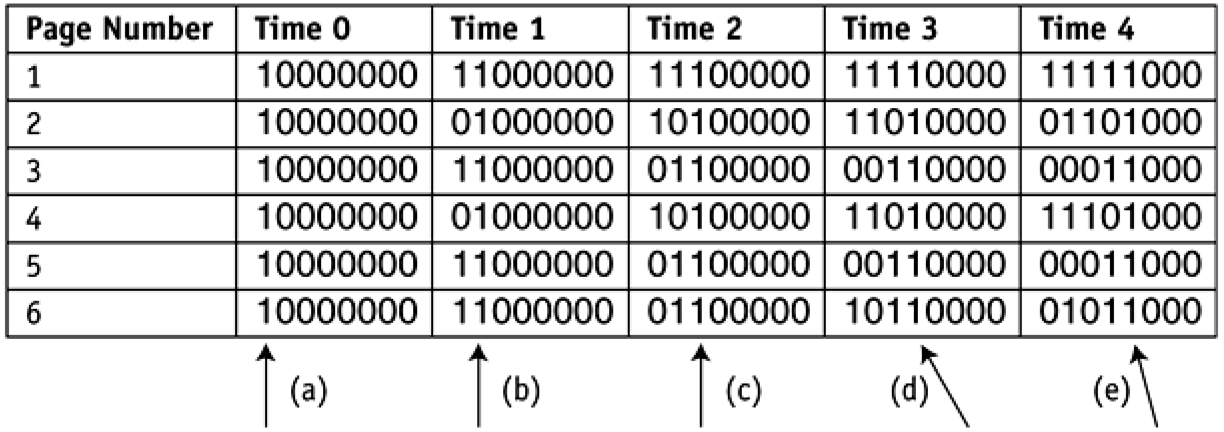


As per Stack Algorithm

Not as per stack algorithm

Page Replacement Policies and Concepts (continued)

- Initially, leftmost bit of its reference byte is set to 1, all bits to the right are set to zero
- Each time a page is referenced, the leftmost bit is set to 1
- Reference bit for each page is updated with every time tick



Page Number	Time 0	Time 1	Time 2	Time 3	Time 4
1	10000000	11000000	11100000	11110000	11111000
2	10000000	01000000	10100000	11010000	01101000
3	10000000	11000000	01100000	00110000	00011000
4	10000000	01000000	10100000	11010000	11101000
5	10000000	11000000	01100000	00110000	00011000
6	10000000	11000000	01100000	10110000	01011000

Figure 3.11: Bit-shifting technique in LRU policy

The Mechanics of Paging

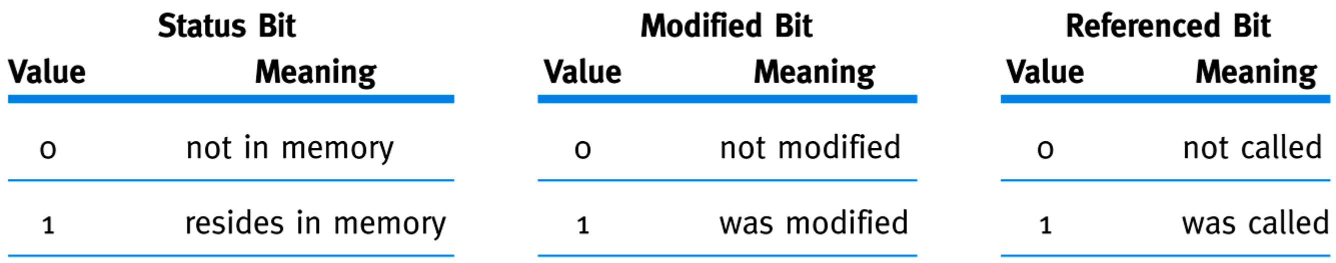
- **Status bit:** Indicates if page is currently in memory
- **Referenced bit:** Indicates if page has been referenced recently
  - Used by LRU to determine which pages should be swapped out
- **Modified bit:** Indicates if page contents have been altered
  - Used to determine if page must be rewritten to secondary storage when it’s swapped out

The Mechanics of Paging (continued)

Page	Status Bit	Referenced Bit	Modified Bit	Page Frame
0	1	1	1	5
1	1	0	0	9
2	1	0	0	7
3	1	1	0	12

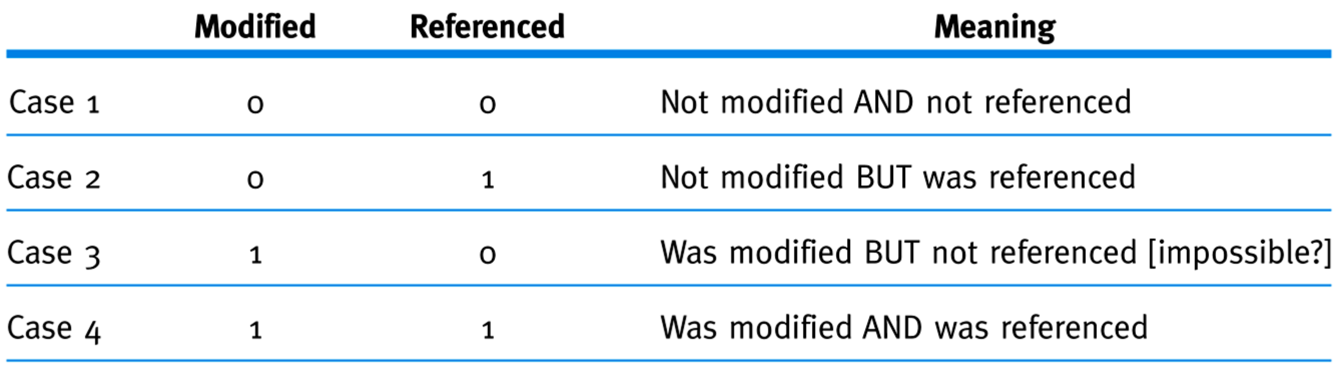
Table 3.3: Page Map Table for Job 1 shown in Figure 3.5.

The Mechanics of Paging (continued)



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: Meanings of bits used in PMT



	Modified	Referenced	Meaning
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: Possible combinations of modified and referenced bits

The Working Set

- **Working set:** Set of pages residing in memory that can be accessed directly without incurring a page fault
  - Improves performance of demand page schemes
  - Requires the concept of “locality of reference”
- **System must decide**
  - How many pages compose the working set
  - The maximum number of pages the operating system will allow for a working set

The Working Set (continued)

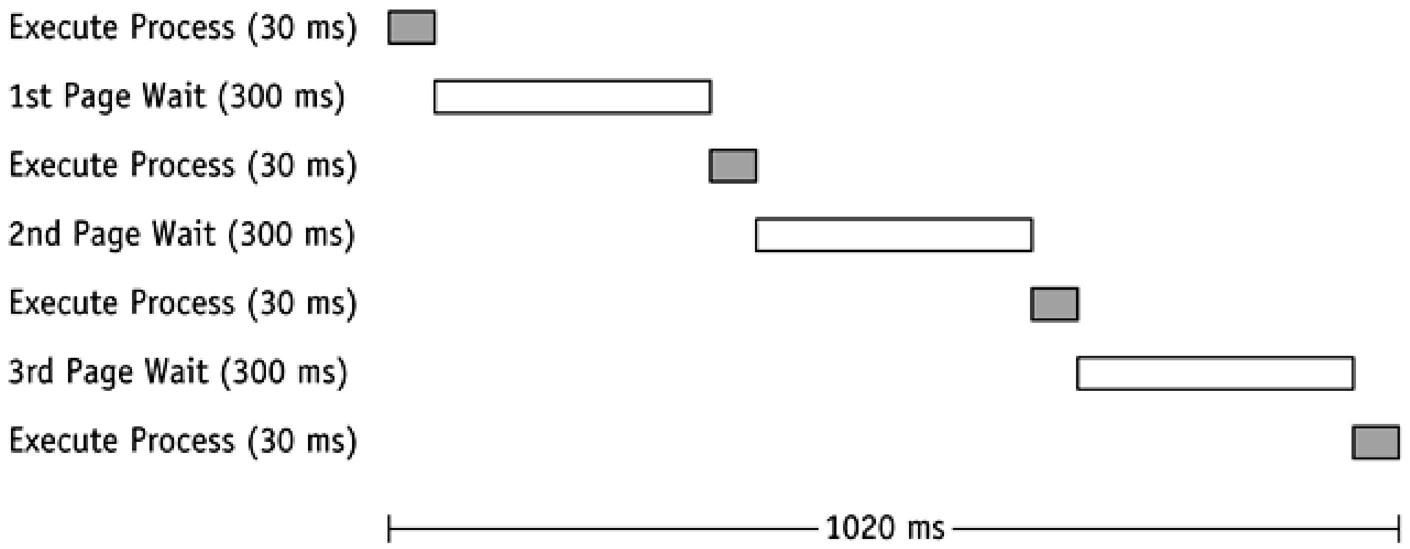


Figure 3.12: An example of a time line showing the amount of time required to process page faults

Segmented Memory Allocation

- Each job is divided into several segments of different sizes, one for each module that contains pieces to perform related functions
- Main memory is no longer divided into page frames, rather allocated in a dynamic manner
- Segments are set up according to the program's structural modules when a program is compiled or assembled
  - Each segment is numbered
  - Segment Map Table (SMT) is generated

Segmented Memory Allocation (continued)

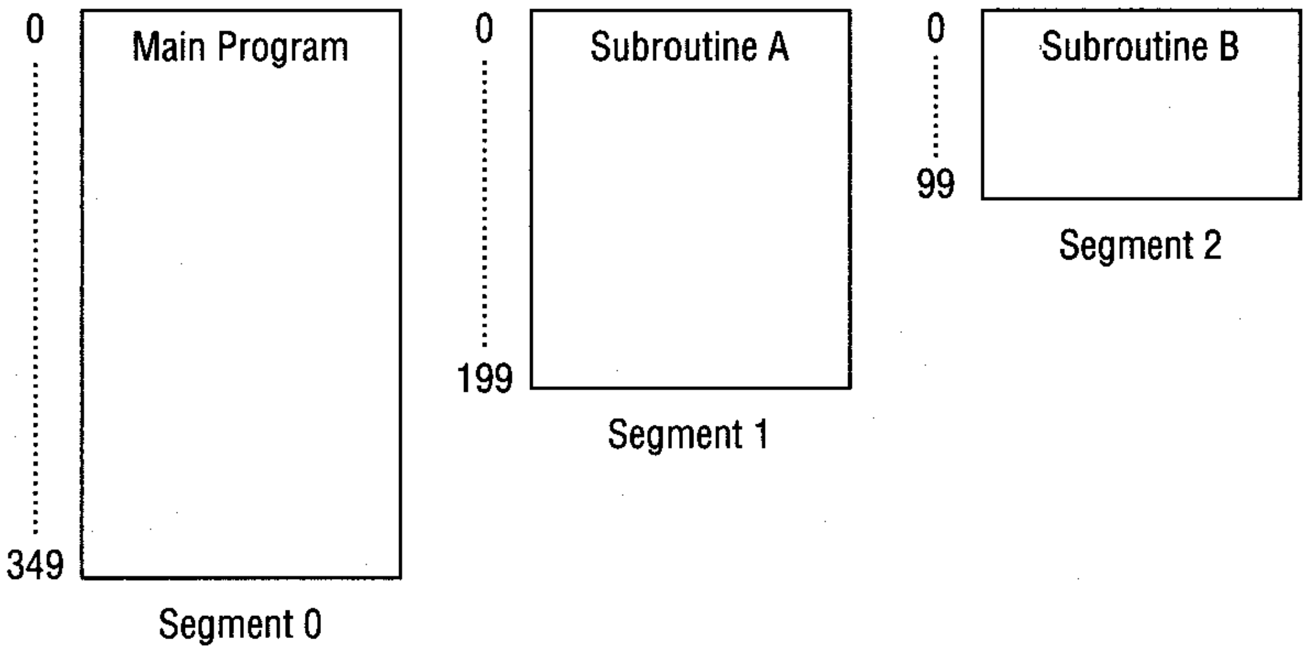
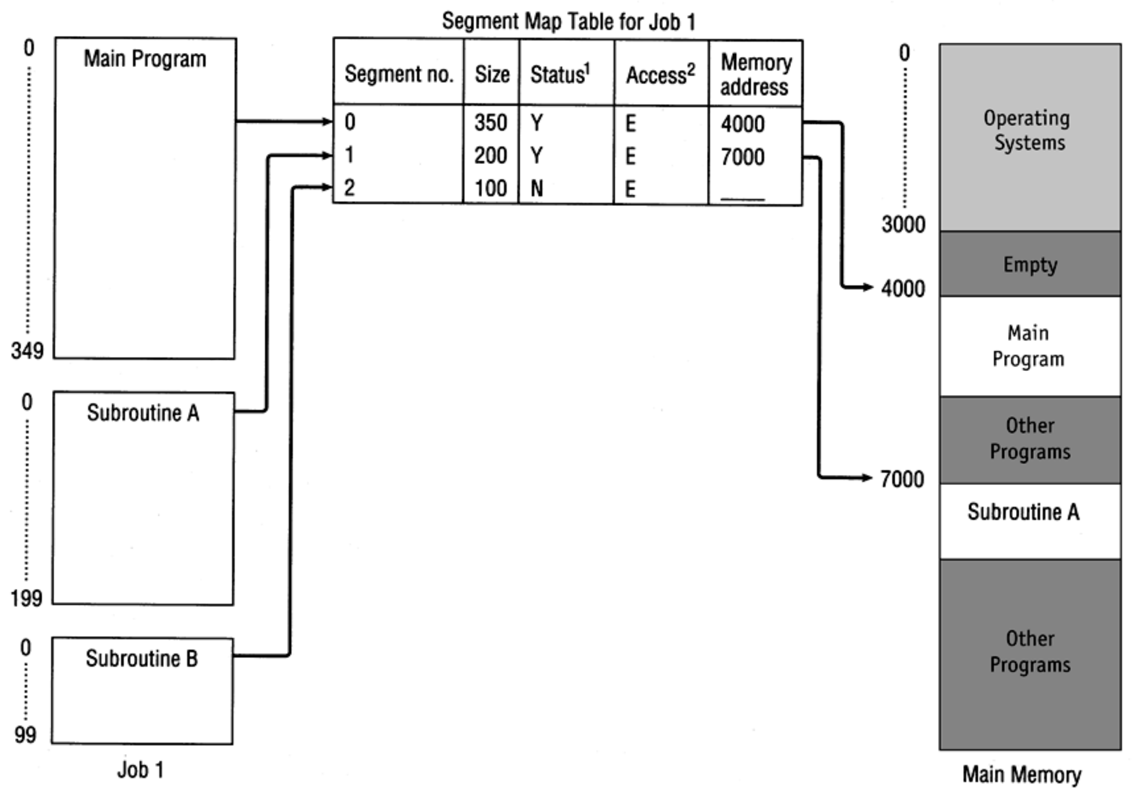


Figure 3.13: Segmented memory allocation. Job 1 includes a main program, Subroutine A, and Subroutine B. It’s one job divided into three segments.

Segmented Memory Allocation (continued)



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	____

Figure 3.14: The Segment Map Table tracks each segment for Job 1

Segmented Memory Allocation (continued)

- Memory Manager tracks segments in memory using following three tables:
  - **Job Table** lists every job in process (one for whole system)
  - **Segment Map Table** lists details about each segment (one for each job)
  - **Memory Map Table** monitors allocation of main memory (one for whole system)
- Segments don’t need to be stored contiguously
- The addressing scheme requires segment number and displacement

Segmented Memory Allocation (continued)

- **Advantages:**
  - Internal fragmentation is removed
- **Disadvantages:**
  - Difficulty managing variable-length segments in secondary storage
  - External fragmentation

Segmented/Demand Paged Memory Allocation

- Subdivides segments into pages of equal size, smaller than most segments, and more easily manipulated than whole segments. It offers:
  - Logical benefits of segmentation
  - Physical benefits of paging
- Removes the problems of compaction, external fragmentation, and secondary storage handling
- The addressing scheme requires segment number, page number within that segment, and displacement within that page

Segmented/Demand Paged Memory Allocation (continued)

- This scheme requires following four tables:
  - **Job Table** lists every job in process (one for the whole system)
  - **Segment Map Table** lists details about each segment (one for each job)
  - **Page Map Table** lists details about every page (one for each segment)
  - **Memory Map Table** monitors the allocation of the page frames in main memory (one for the whole system)

Segmented/Demand Paged Memory Allocation (continued)

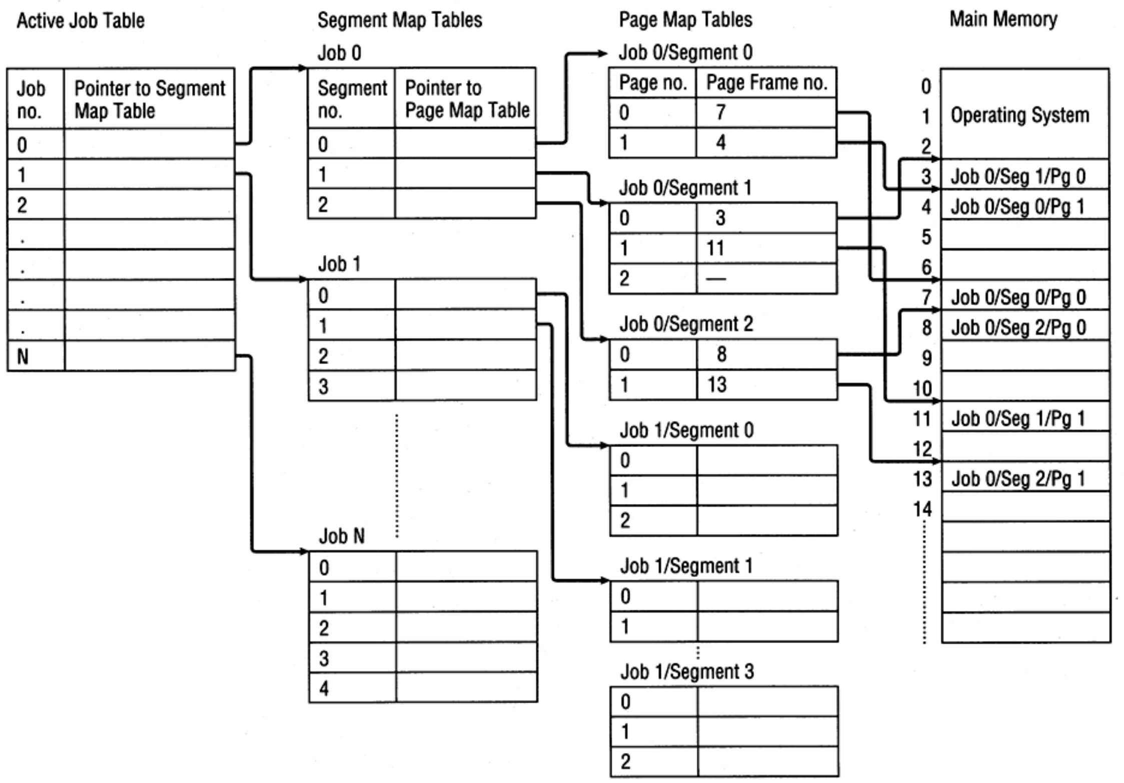


Figure 3.16: Interaction of JT, SMT, PMT, and main memory in a segment/paging scheme

Segmented/Demand Paged Memory Allocation (continued)

- **Advantages:**
  - Large virtual memory
  - Segment loaded on demand
- **Disadvantages:**
  - Table handling overhead
  - Memory needed for page and segment tables
- To minimize number of references, many systems use **associative memory** to speed up the process
  - Its disadvantage is the high cost of the complex hardware required to perform the parallel searches

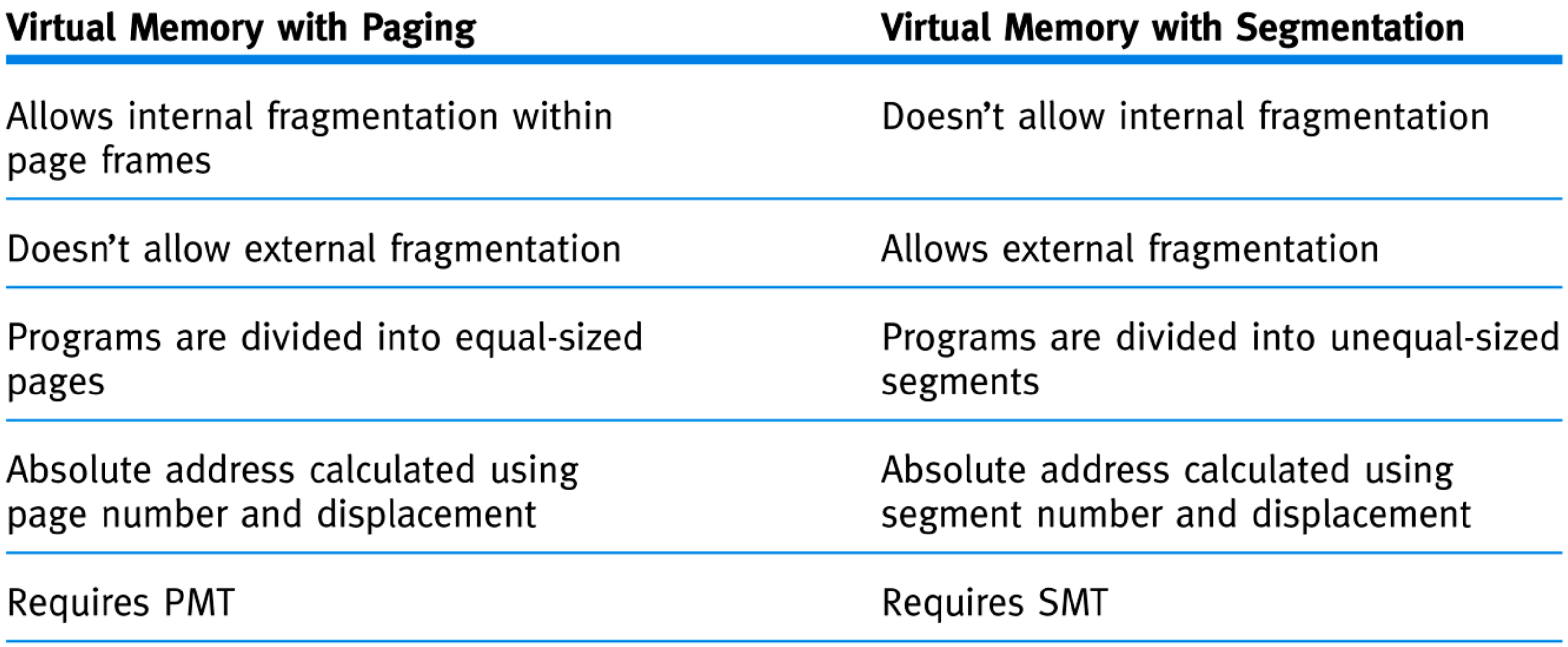
Virtual Memory

- Allows programs to be executed even though they are not stored entirely in memory
- Requires cooperation between the Memory Manager and the processor hardware
- **Advantages of virtual memory management:**
  - Job size is not restricted to the size of main memory
  - Memory is used more efficiently
  - Allows an unlimited amount of multiprogramming

Virtual Memory (continued)

- **Advantages (continued):**
  - Eliminates external fragmentation and minimizes internal fragmentation
  - Allows the sharing of code and data
  - Facilitates dynamic linking of program segments
- **Disadvantages:**
  - Increased processor hardware costs
  - Increased overhead for handling paging interrupts
  - Increased software complexity to prevent thrashing

Virtual Memory (continued)



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
Absolute address calculated using page number and displacement	Absolute address calculated using segment number and displacement
Requires PMT	Requires SMT

Table 3.6: Comparison of virtual memory with paging and segmentation

Cache Memory

- A small high-speed memory unit that a processor can access more rapidly than main memory
- Used to store frequently used data, or instructions
- Movement of data, or instructions, from main memory to cache memory uses a method similar to that used in paging algorithms
- Factors to consider in designing cache memory:
  - Cache size, block size, block replacement algorithm and rewrite policy

Cache Memory (continued)

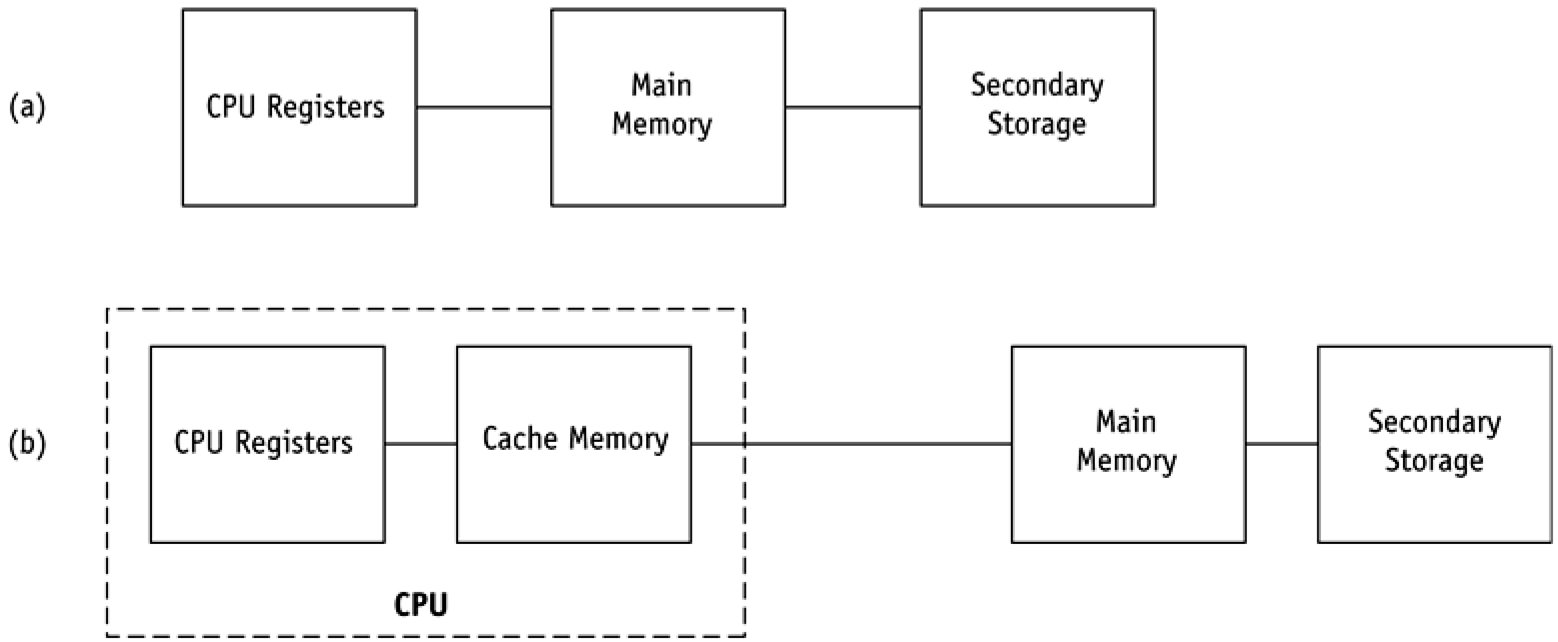
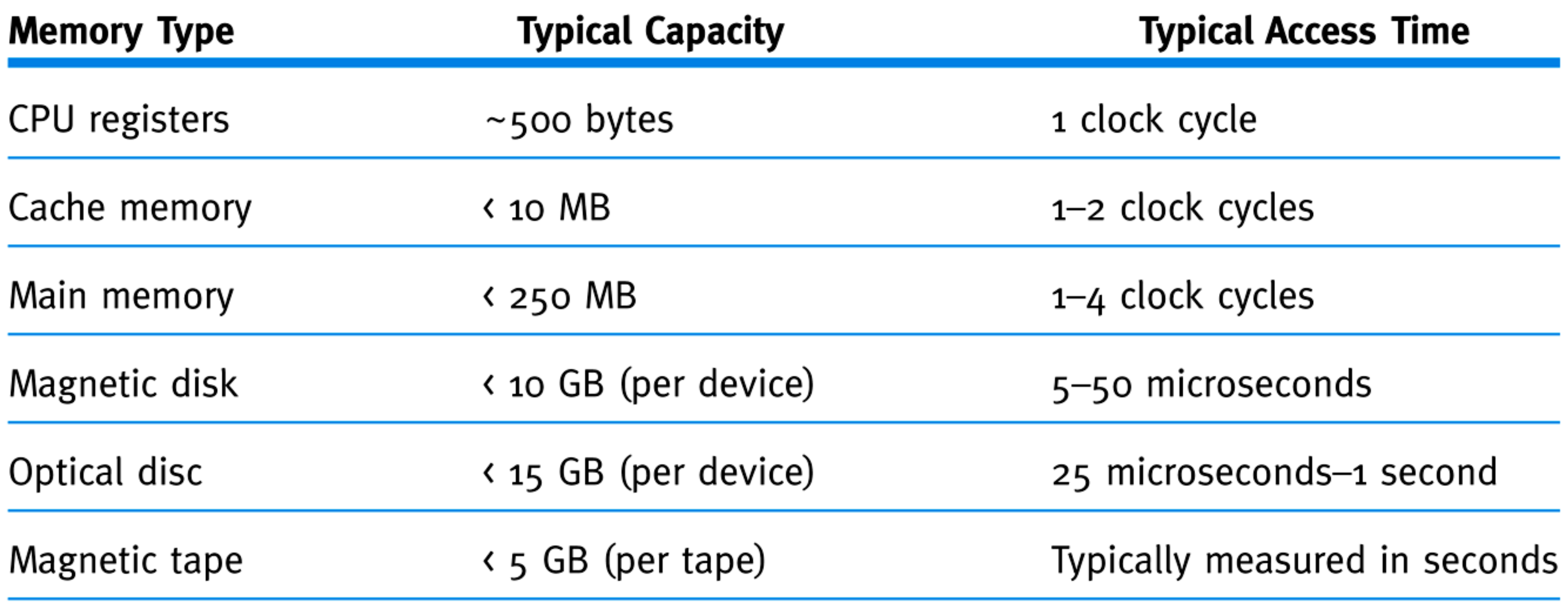


Figure 3.17: Comparison of (a) traditional path used by early computers and (b) path used by modern computers to connect main memory and CPU via cache memory

Cache Memory (continued)



Memory Type	Typical Capacity	Typical Access Time
CPU registers	~500 bytes	1 clock cycle
Cache memory	< 10 MB	1–2 clock cycles
Main memory	< 250 MB	1–4 clock cycles
Magnetic disk	< 10 GB (per device)	5–50 microseconds
Optical disc	< 15 GB (per device)	25 microseconds–1 second
Magnetic tape	< 5 GB (per tape)	Typically measured in seconds

Table 3.7: A list of relative speeds and sizes for all types of memory. A clock cycle is the smallest unit of time for a processor.

Case Study: Memory Management in Linux

Virtual memory in Linux is managed using a three-level table hierarchy

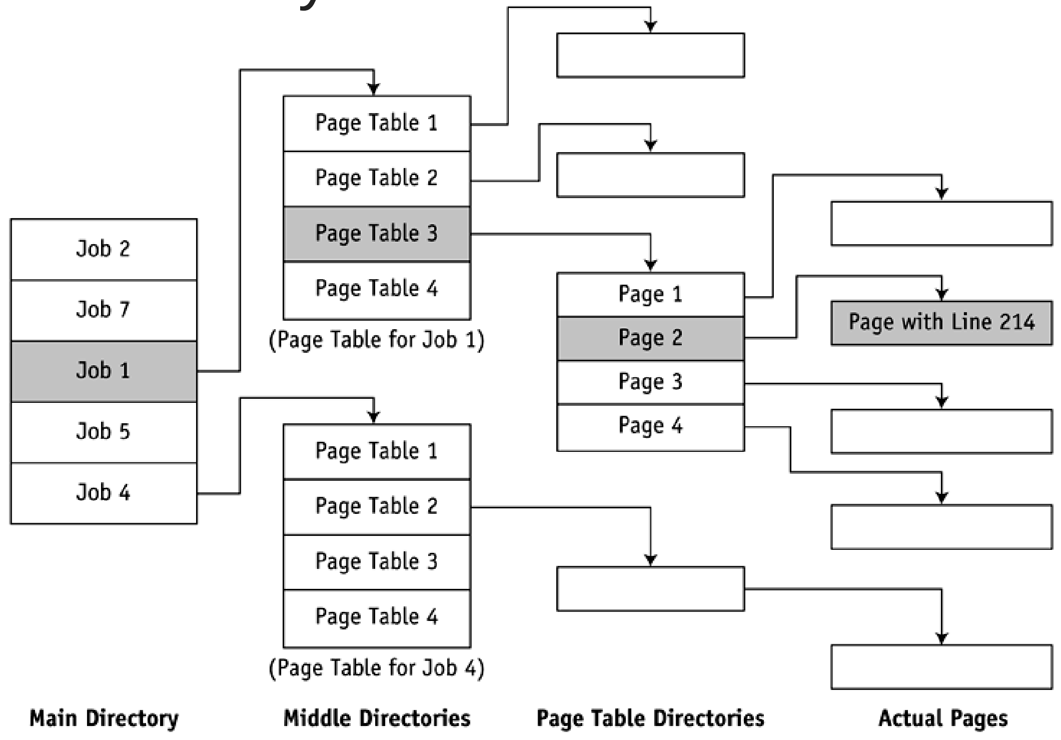


Figure 3.18: Virtual memory management in Linux

Case Study: Memory Management in Linux (continued)

Case: Main memory consists of 64 page frames, and Job 1 requests 15 page frames, Job 2 requests 8 page frames

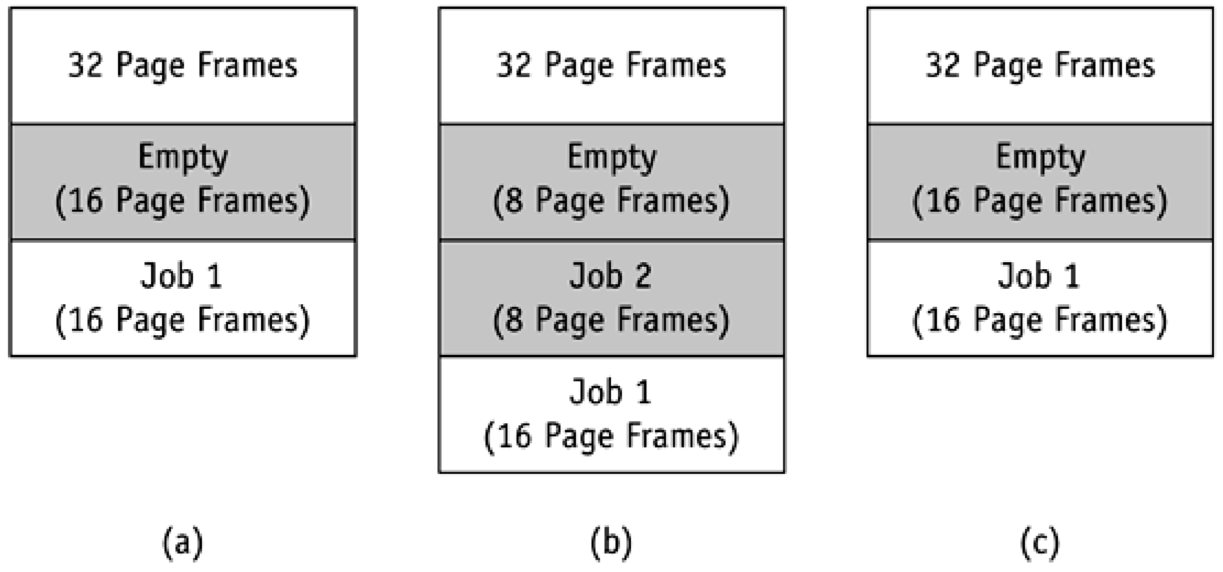


Figure 3.19: An example of Buddy algorithm in Linux

Summary

- Paged memory allocations allow efficient use of memory by allocating jobs in noncontiguous memory locations
- Increased overhead and internal fragmentation are problems in paged memory allocations
- Job no longer constrained by the size of physical memory in demand paging scheme
- LRU scheme results in slightly better efficiency as compared to FIFO scheme
- Segmented memory allocation scheme solves internal fragmentation problem

Summary (continued)

- Segmented/demand paged memory allocation removes the problems of compaction, external fragmentation, and secondary storage handling
- Associative memory can be used to speed up the process
- Virtual memory allows programs to be executed even though they are not stored entirely in memory
- Job’s size is no longer restricted to the size of main memory by using the concept of virtual memory
- CPU can execute instruction faster with the use of cache memory