고려대학교
KOREA UNIVERSITY

KU-The Future

Operating Systems (10th Ed., by A. Silberschatz)

Chapter 10 Virtual Memory

Heonchang Yu
Distributed and Cloud Computing Lab.

Contents

Objectives

- Define virtual memory and describe its benefits.
- Illustrate how pages are loaded into memory using demand paging.
- Apply the FIFO, optimal, and LRU page-replacement algorithms.
- Describe the working set of a process, and explain how it is related to program locality.
- Describe how Linux, Windows 10, and Solaris manage virtual memory.
- Design a virtual memory manager simulation in the C programming language.

Background

- Examples that the entire program is not needed 전체 프로그램이 로딩 될 필요가 없다
  - Code to handle unusual error conditions – since these errors seldom occur in practice, this code is almost never executed.
  - Arrays, lists, and tables are often allocated more memory than they actually need.
  - Certain options and features of a program may be used rarely.
- Benefits – ability to execute a program that only partially in memory 가상 메모리의 장점
  - No longer be constrained by the amount of physical memory that is available
  - More programs could be run at the same time
  - Less I/O would be needed to load or swap portions of programs into memory, so each program would run faster.

Background

- **Virtual memory** – separation of logical memory as perceived by developers from physical memory
  - 실제 *physical* 메모리보다 더 크게 사용할 수 있다는 점
  - This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. (Figure 10.1)
- The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.
  - Figure 10.2
    - ✓ Allow the heap to grow upward in memory as it is used for dynamic memory allocation
    - ✓ Allow for the stack to grow downward in memory through successive function calls

Background

Usei

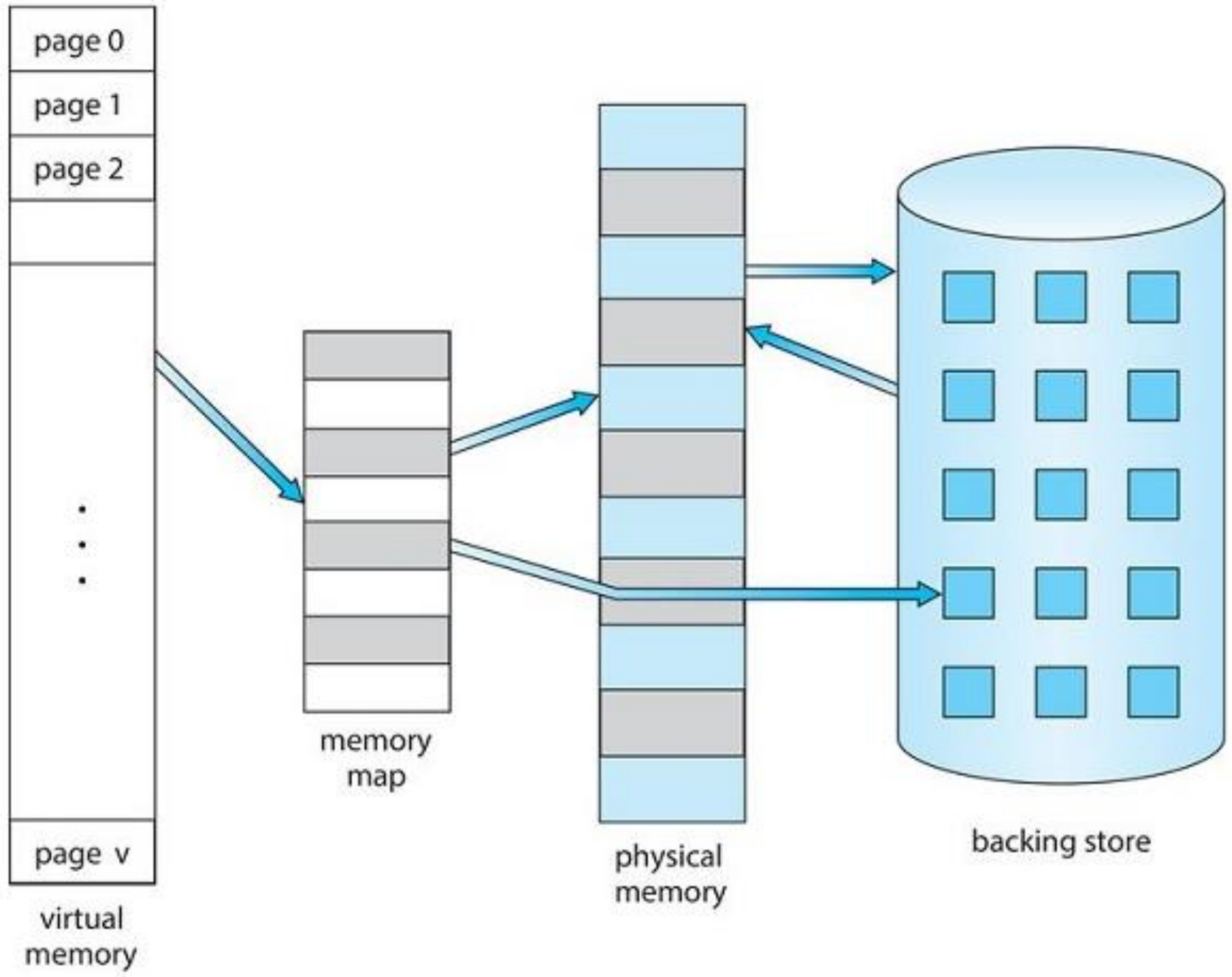


Figure 10.1 Diagram showing virtual memory that is larger than physical memory.

Background

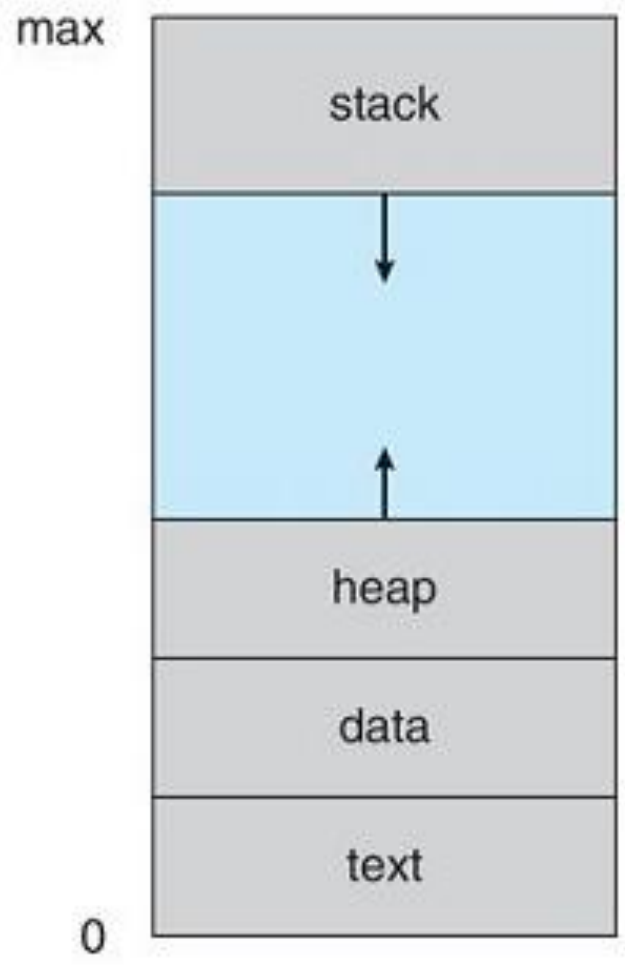


Figure 10.2 Virtual address space of a process in memory.

Background

- Benefits – virtual memory allows files and memory to be shared by two or more processes through page sharing
  - The actual pages where the libraries reside in physical memory are shared by all the processes – Figure 10.3
  - Processes can share memory.
    - ✓ Virtual memory allows one process to create a region of memory that it can share with another process.
  - Pages can be shared during process creation with the `fork()` system call, thus speeding up process creation.

Background

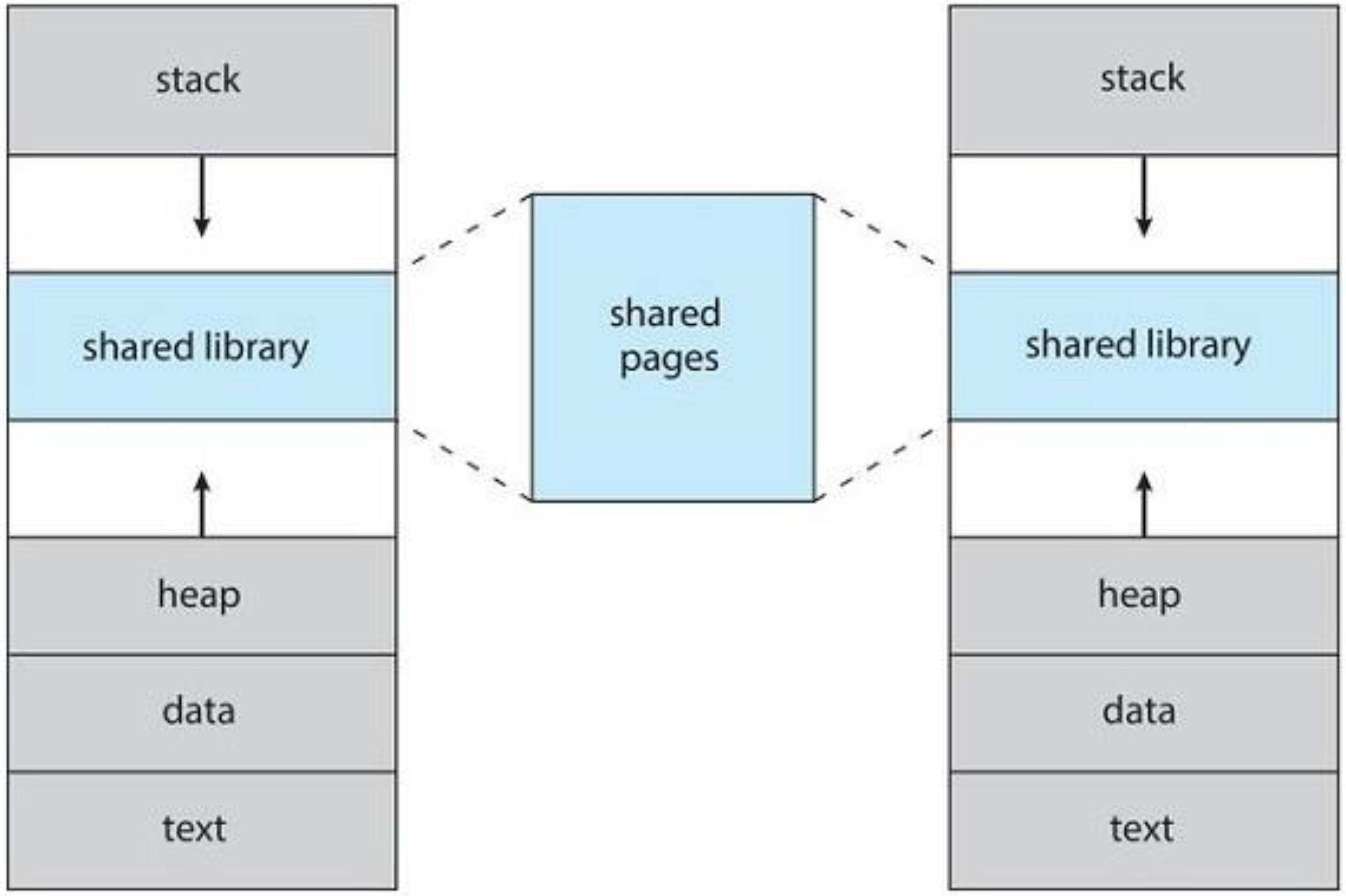


Figure 10.3 Shared library using virtual memory.

Demand Paging

– Consider how an executable program might be loaded from secondary storage into memory.
  - Load the entire program in physical memory at program execution time (regardless of whether an option is selected by the user or not)
  - Load pages only as they are needed

    ⇒ demand paging

    ✓ With demand-paged virtual memory, pages are loaded only when they are **demanded** during program execution.

    ✓ Pages that are never accessed are thus never loaded into physical memory.

Demand Paging

- We need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.
  - Valid-invalid bit (Figure 10.4)
    - ✓ When this bit is set to "valid", the associated page is both legal and in memory.
    - ✓ If the bit is set to "invalid", the page either is not valid or is valid but is currently in secondary storage.
  - Marking a page invalid will have no effect if the process never attempts to access that page.

페이지의 모든 정보는 페이지 테이블에서 관리하고 있음
페이지 테이블에 하나의 비트를 추가해서 *valid & invalid* 판단

Demand Paging

*A, C, F*만 현재 페이지 테이블에 로딩된 상황

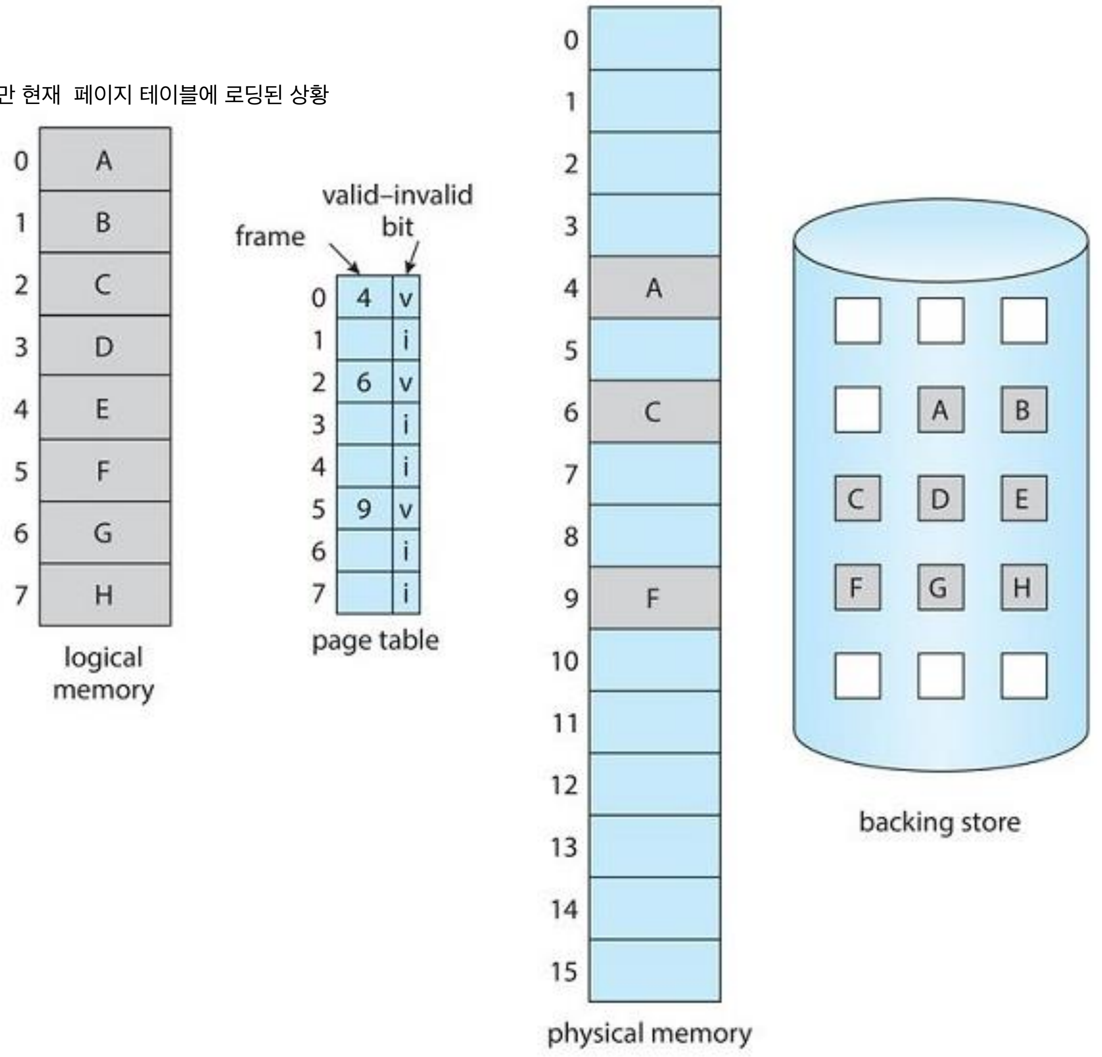


Figure 10.4 Page table when some pages are not in main memory.

Demand Paging

– What happens if the process tries to access a page that was not brought into memory?

invalid 값에 접근하는 상황

⇒ page fault : access to a page marked invalid

- Procedure for handling this page fault (Figure 10.5)
  - ✓ We check an internal table for this process to determine whether the reference was a valid or an invalid memory access.
  - ✓ If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.
  - ✓ We find a free frame.
  - ✓ We schedule a secondary storage operation to read the desired page into the newly allocated frame.
  - ✓ When the stroage read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
  - ✓ We restart the instruction that was interrupted by the trap. The process can access the page as though it had been in memory.

Demand Paging

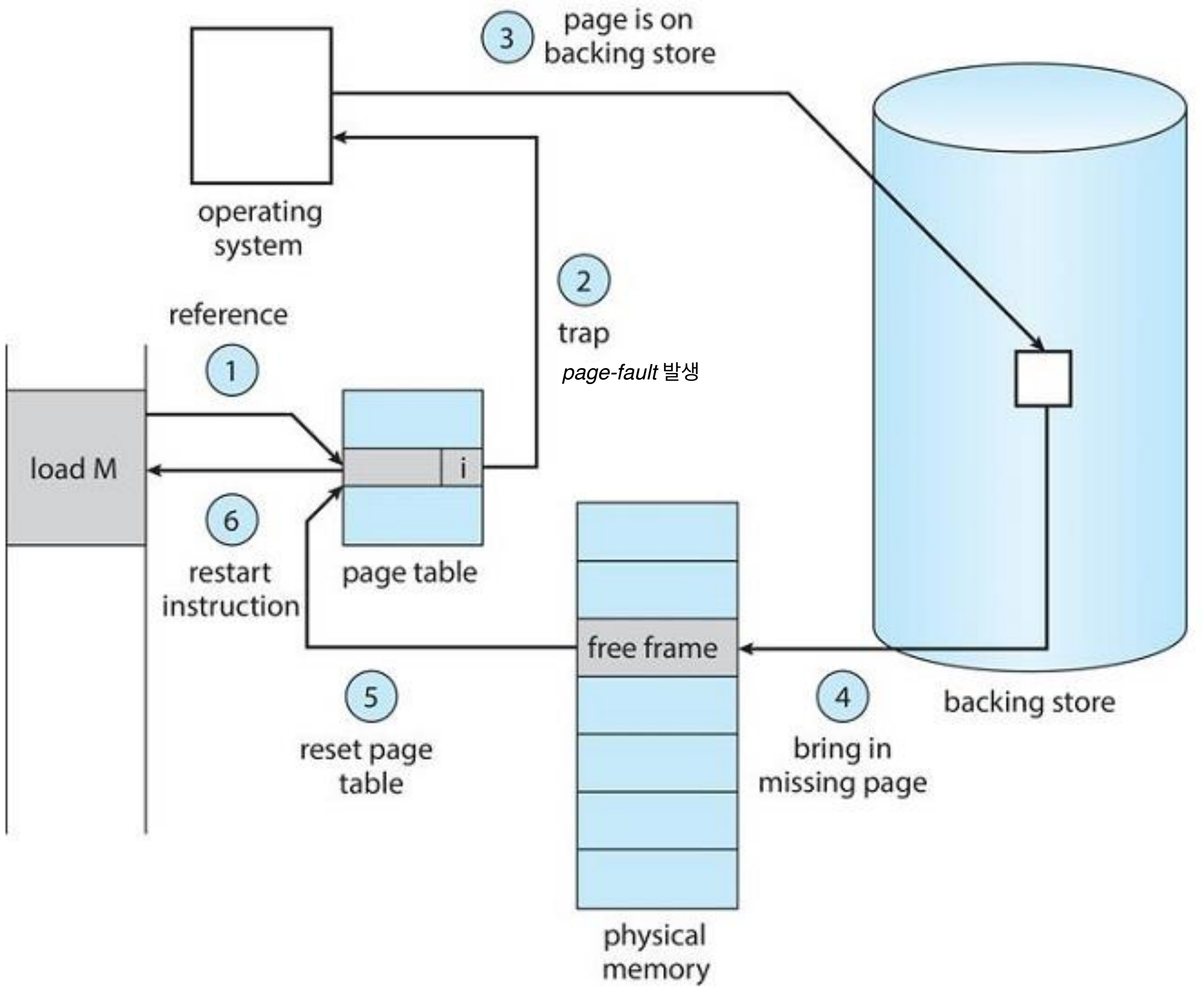


Figure 10.5 Steps in handling a page fault.

Demand Paging

- Pure demand paging
  - ✓ Never bring a page into memory until it is required.
- Some programs could access several new pages of memory with each instruction execution (one page for the instruction and many for data), possibly causing multiple page faults per instruction.
- Hardware to support demand paging
  - ✓ Page table
  - ✓ Secondary memory – a high-speed disk (swap device)
    - ❖ This memory holds those pages that are not present in main memory. (swap space)

Demand Paging

- Performance of Demand Paging
  - The probability of Page Fault $0 \le p \le 1$
    - ✓ if $p = 0$ no page faults
    - ✓ if $p = 1$, every reference is a fault
  - Effective Access Time (EAT)

$$\text{EAT} = (1 - p) \times \text{memory access time (ma)} + p \times \text{page fault time}$$

- ✓ With an average page-fault service time of 8ms and a memory-access time of 200ns

effective access time $= (1-p) \times (200\text{ns}) + p \times (8\text{ms})$

$= (1-p) \times 200 + p \times 8{,}000{,}000$

$= 200 + 7{,}999{,}800 \times p$

Demand Paging

- A page fault causes the following sequence to occur.
  1. Trap to the operating system.
  2. Save the registers and process state. PCB
  3. Determine that the interrupt was a page fault. ISR
  4. Check that the page reference was legal and determine the location of the page in secondary storage.
  5. Issue a read from the stroage to a free frame:
     a. Wait in a queue until the read request is serviced.
     b. Wait for the device seek and/or latency time.
     c. Begin the transfer of the page to a free frame.
  6. While waiting, allocate the CPU to some other process
  7. Receive an interrupt from the storage I/O subsystem (I/O completed).
  8. Save the registers and process state for the other process (if step 6 is executed).

Demand Paging

- A page fault causes the following sequence to occur.
  9. Determine that the interrupt was from the secondary storage device.
  10. Correct the page table and other tables to show that the desired page is now in memory.
  11. Wait for the CPU core to be allocated to this process again.
  12. Restore the registers, process state, and new page table, and then resume the interrupted instruction.

Copy-on-Write

- Works by allowing the parent and child processes initially to share the same pages
- These shared pages are marked as copy-on-write pages.
  - ✓ If either process writes to a shared page, a copy of the shared page is created. (Figure 10.7 & 10.8)
- Example – Assume that the child process attempts to modify a page.
  - ✓ The operating system will create a copy of this page, mapping it to the address space of the child process.
  - ✓ The child process will modify its copied page and not the page belonging to the parent process.
  - ✓ Only pages that can be modified need be marked as copy-on-write.
- OS allocate free frames using a technique known as "zero-fill-on-demand"
  - ✓ Zero-fill-on-demand frames are "zeroed-out" before being allocated, thus erasing the previous contents.

Copy-on-Write

서로 다른 프로세스들이 공유하는 자원들이 있음

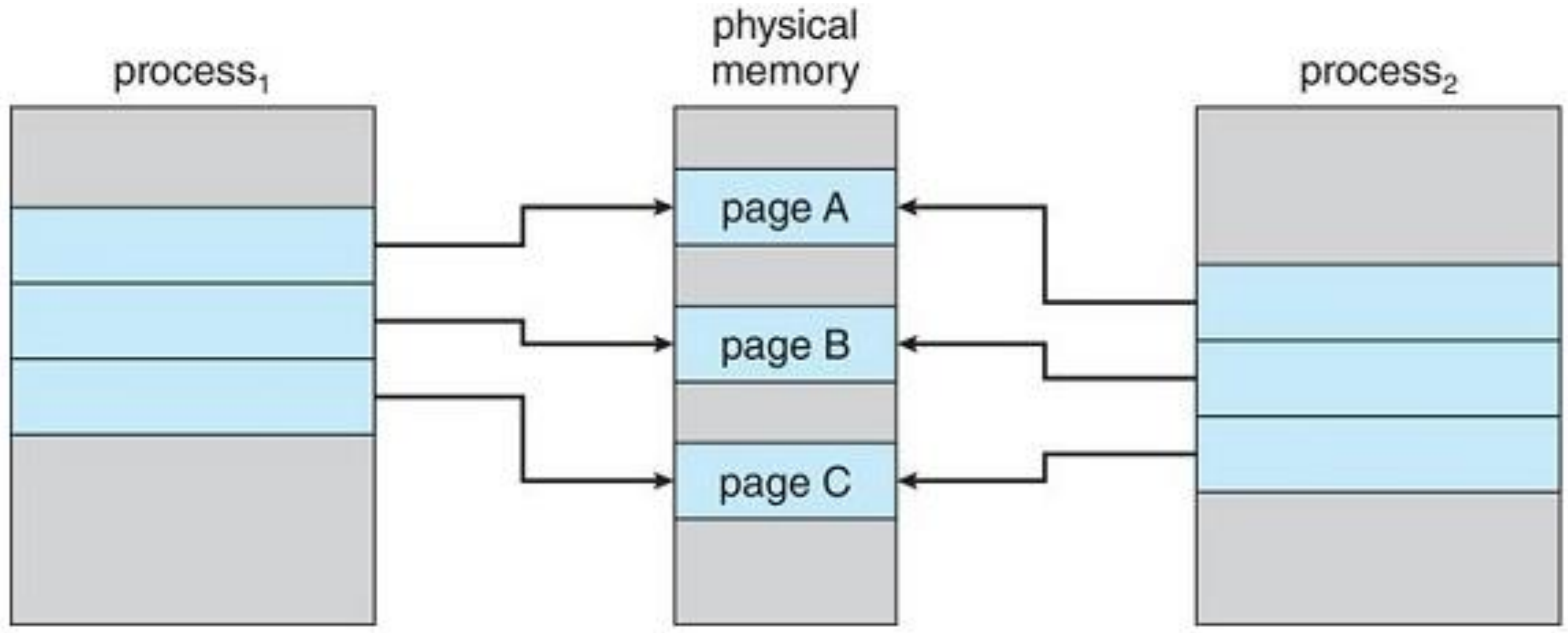


Figure 10.7 Before process 1 modifies page C.

Copy-on-Write

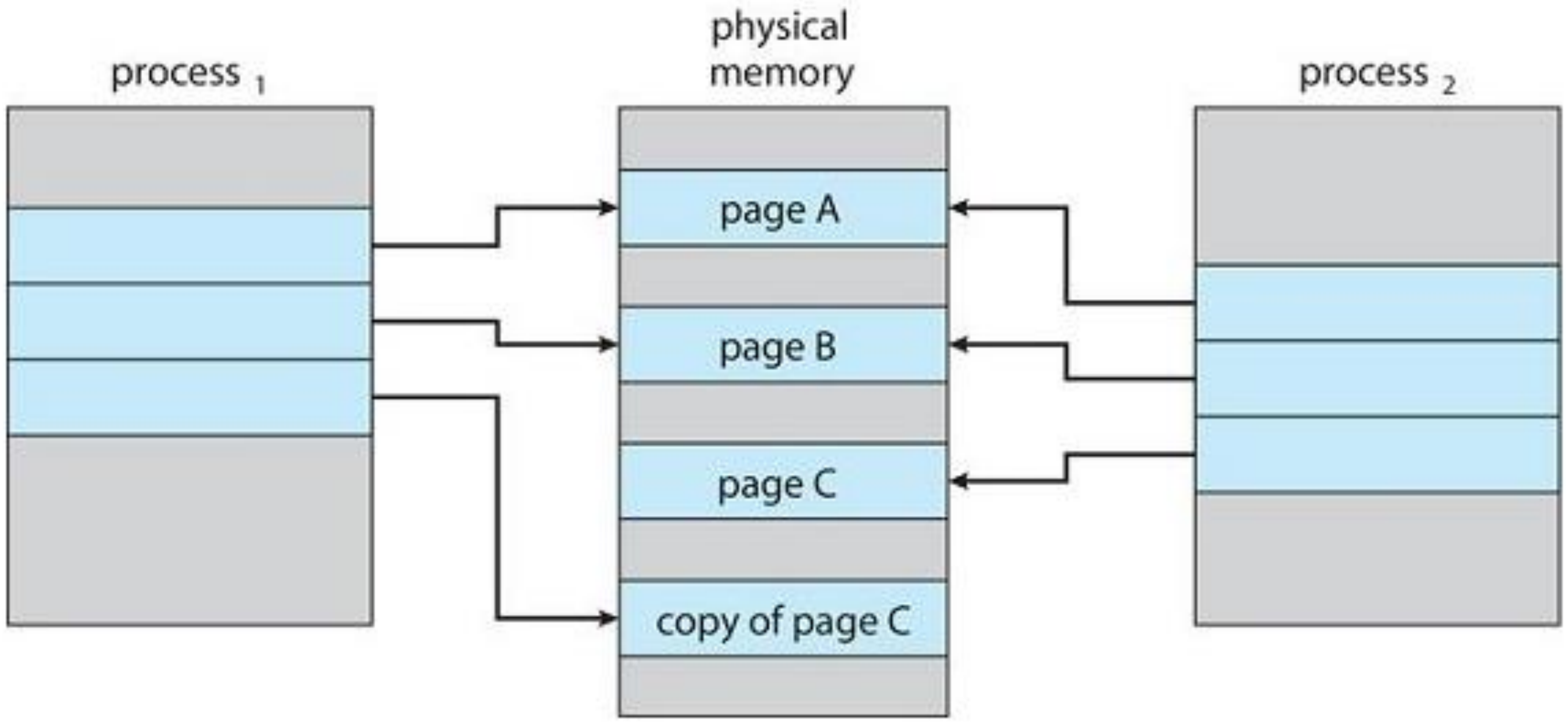


Figure 10.8 After process 1 modifies page C.

Page Replacement

- If we increase our degree of multiprogramming, we are over-allocating memory.

  메인 메모리에 로딩된 프로세스의 수가 여러 개일 때
- Over-allocation of memory manifests itself as follows.
  - ✓ While a user process is executing, a page fault occurs.
  - ✓ The operating system determines where the desired page is residing on secondary storage but then finds that there are no free frames on the free-frame list; all memory is in use (Figure 10.9).
- Several options on over-allocation of memory
  - ✓ It could terminate the process.
  - ✓ Page replacement : the most common solution

    기존에 메인 메모리에 존재하는 것을 빈 공간에 대체하는 것

Page Replacement

여기서 *page replacement*를 사용하자

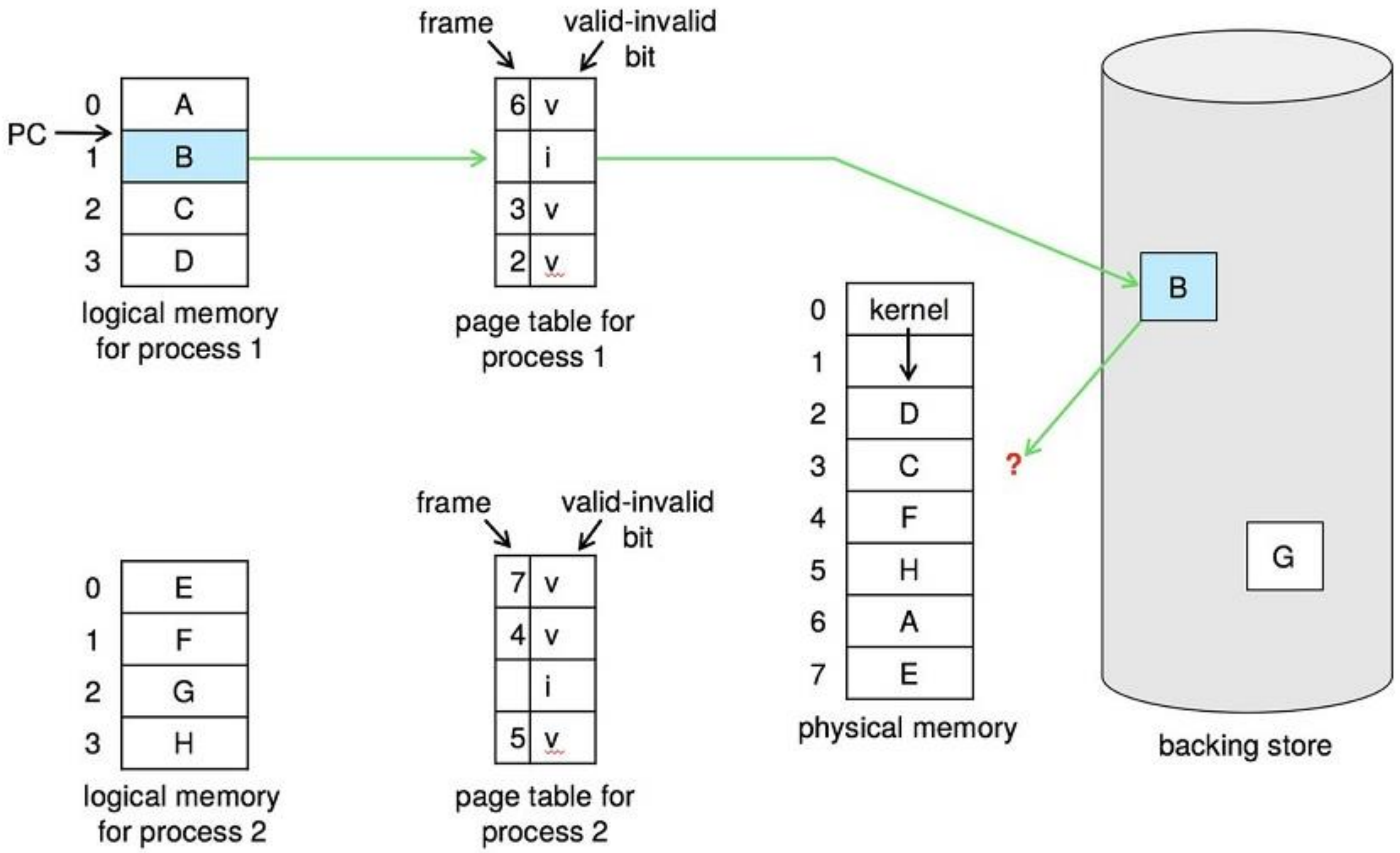


Figure 10.9 Need for page replacement.

Page Replacement

– Basic Page Replacement

- Page-fault service routine – Figure 10.10
  1. Find the location of the desired page on secondary storage.
  2. Find a free frame:
     a. If there is a free frame, use it
     b. If there is no free frame, use a page-replacement algorithm to select a **victim** frame.
     c. Write the victim frame to secondary storage (if necessary); change the page and frame tables accordingly.
  3. Read the desired page into the newly freed frame; change the page and frame tables.
  4. Continue the user process from where the page fault occurred.

Page Replacement

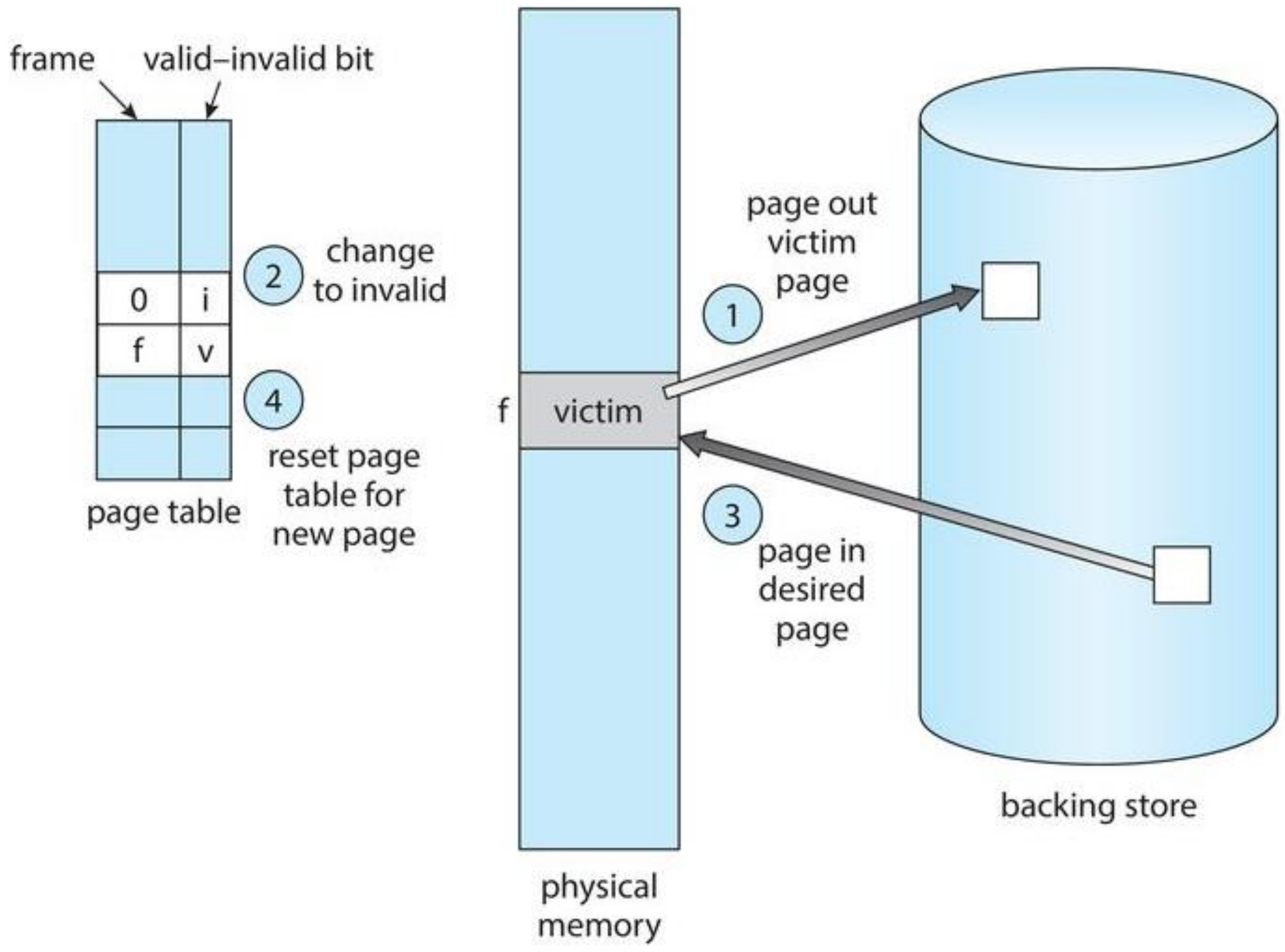


Figure 10.10 Page replacement.

Page Replacement

– Modify bit (or Dirty bit)
  - If no frames are free, two page transfers (one for the page-out and one for the page-in) are required.
    - ✓ This situation doubles the page-fault service time and increases the effective access time accordingly.
  - Reduce this overhead by using a **modify bit** (or **dirty bit**)
  - The modify bit for a page is set by the hardware whenever any byte in the page is written into, indicating that the page has been modified.
  - This scheme can significantly reduce the time required to service a page fault, since it reduces I/O time by one-half if the page has not been modified.

Page Replacement

- Page replacement is basic to demand paging.
- Frame-allocation algorithm
  - ✓ If we have multiple processes in memory, we must decide how many frames to allocate to each process.
- Page-replacement algorithm
  - ✓ When page replacement is required, we must select the frames that are to be replaced.
- Reference string
  - ✓ The string of memory references
  - ✓ Address sequence

    0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
    0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

    0 ~ 99 : 한 블록
    100 ~ 199 : 두 블록

  - ✓ Reference string

    1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

    *100*번은 *1*
    *432*번은 *4*
    요런 형태로 *reference*

- Figure 10.11 – As the number of frames increases, the number of page faults drops to some minimal level.

Page Replacement

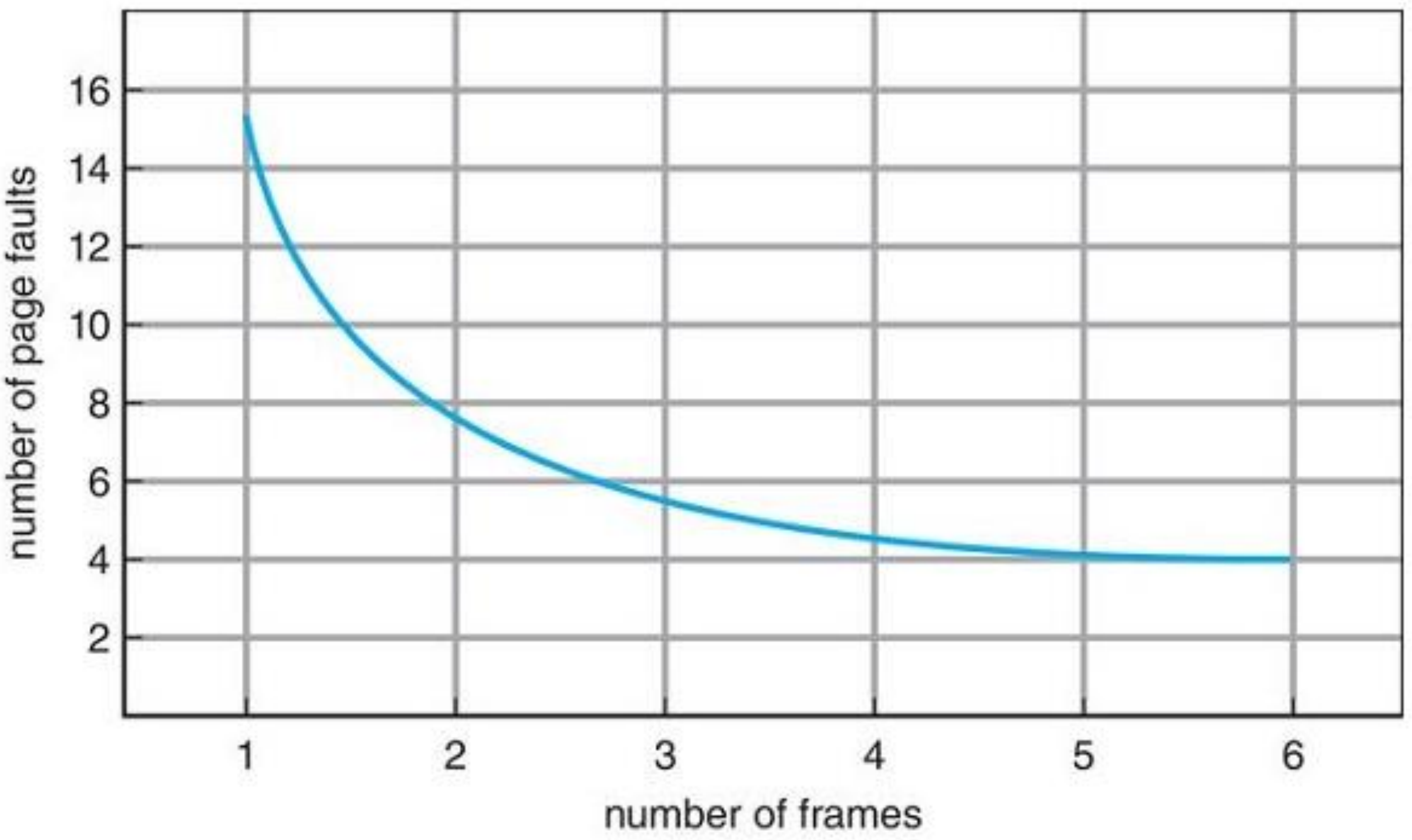
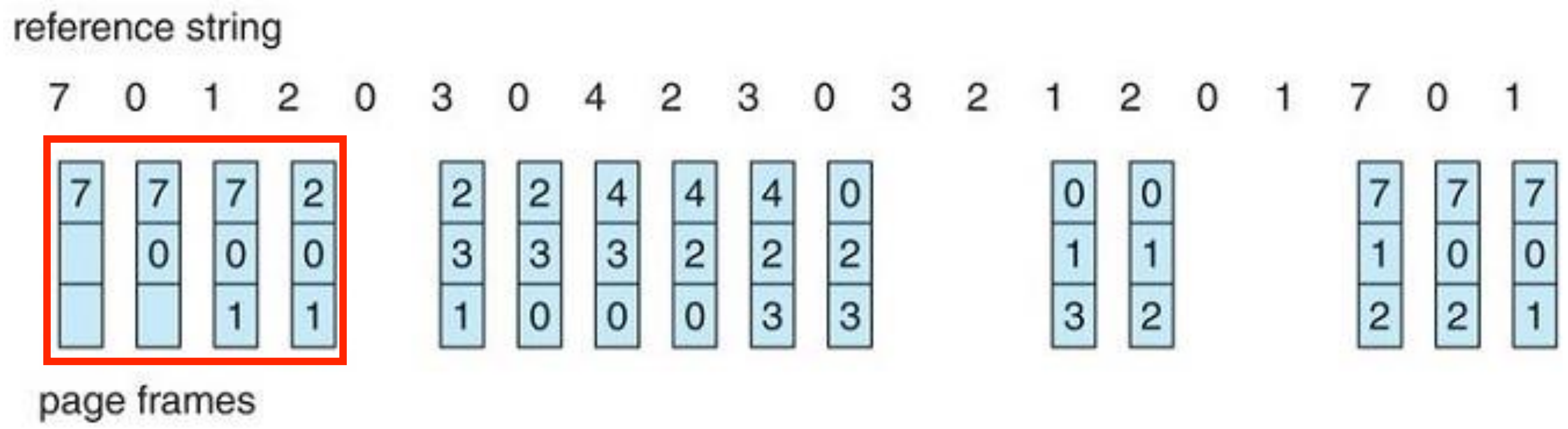


Figure 10.11 Graph of page faults versus number of frames.

Page Replacement

- FIFO Page replacement
  - The simplest page-replacement algorithm
  - When a page must be replaced, the oldest page is chosen.
  - FIFO queue to hold all pages in memory
    - ✓ Replace the page at the head of the queue
  - Figure 10.12
    - ✓ The first three references (7, 0, 1) cause page faults.
  - Belady's anomaly
    - ✓ The page-fault rate may increase as the number of allocated frames increases.
    - ✓ Figure 10.13

Page Replacement



*free frame*이 3개 있음

Figure 10.12 FIFO page-replacement algorithm.

먼저 들어온 값이 *victim frame*이 된다.

Page Replacement

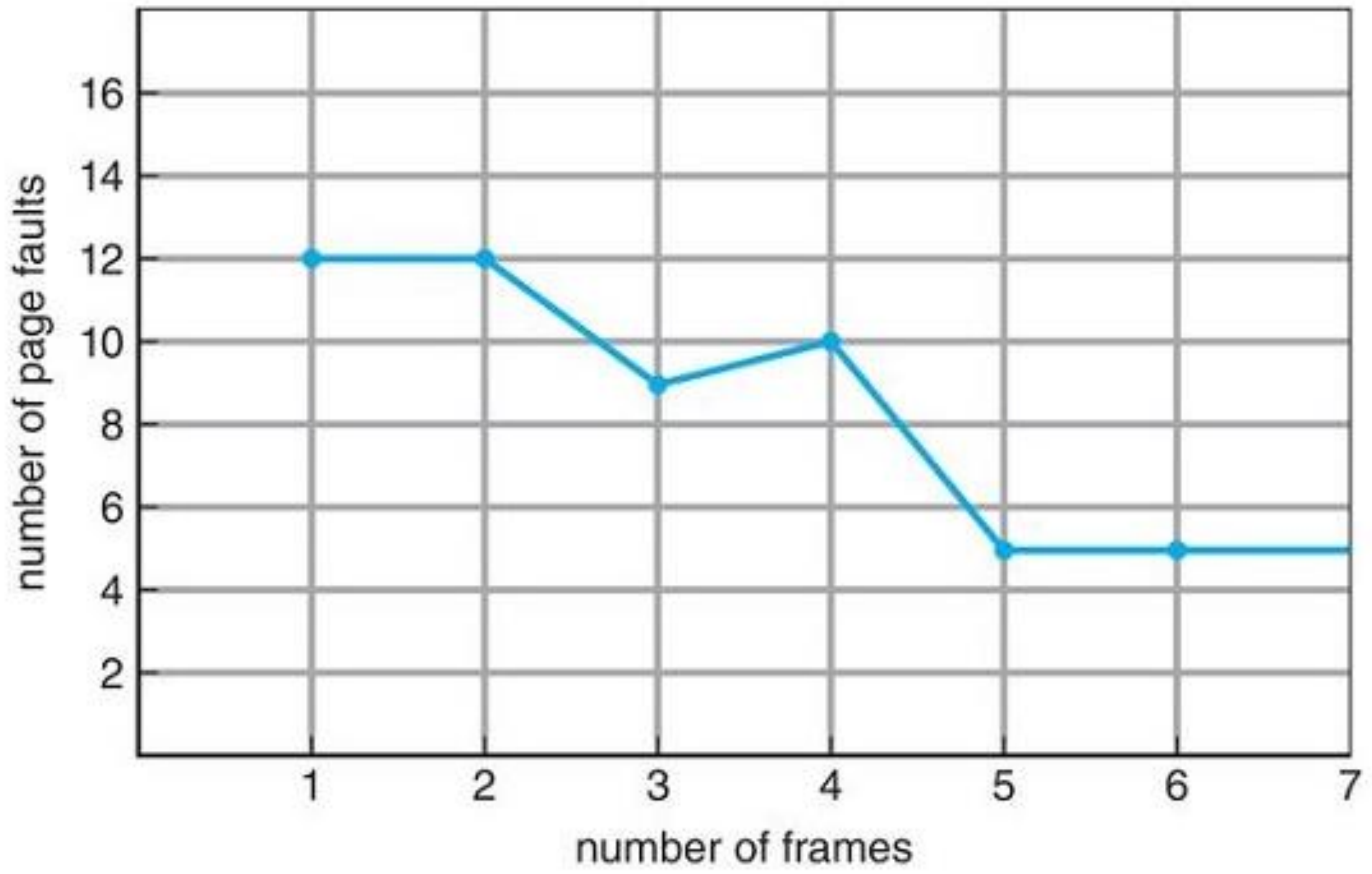


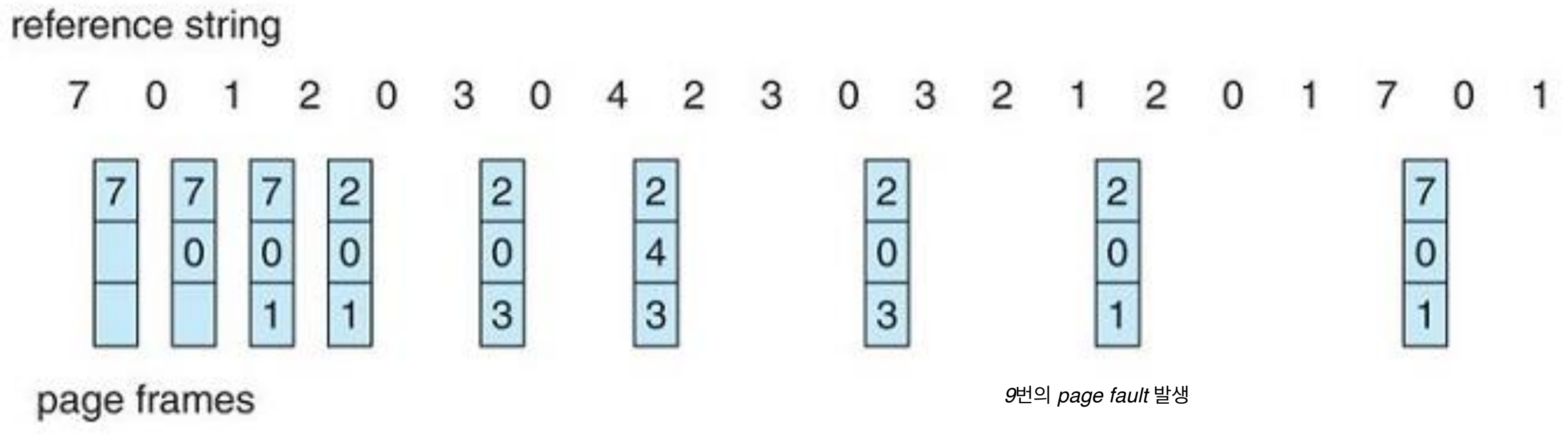
Figure 10.13 Page-fault curve for FIFO replacement on a reference string.

Page Replacement

- Optimal Page replacement
  - The lowest page-fault rate of all algorithms
  - Never suffer from Belady's anomaly
  - Replace the page that will not used for the longest period of time
  - The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.
  - Figure 10.14

Page Replacement

가장 오랫동안 사용되지 않을 값을 변경



9번의 *page fault* 발생

Figure 10.14 Optimal page-replacement algorithm.

Page Replacement

- LRU(least-recently-used) Page Replacement
  - An approximation of the optimal algorithm
  - The FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be used.
  - Replace the page that has not been used for the longest period of time
  - LRU replacement associates with each page the time of that page’s last use.
  - Figure 10.15

Page Replacement

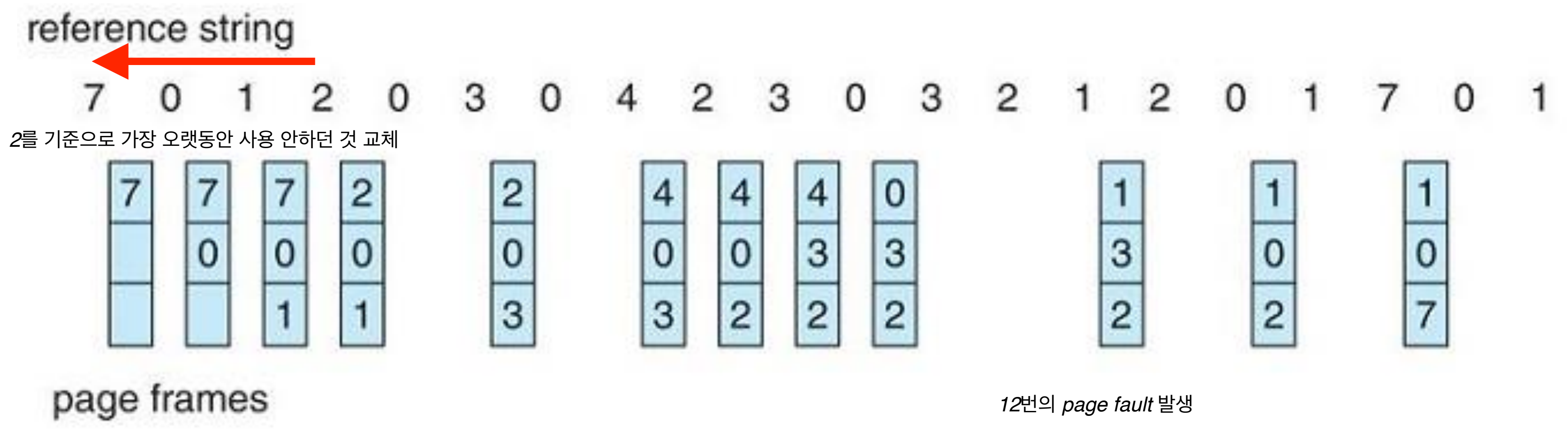


Figure 10.15 LRU page-replacement algorithm.

Page Replacement

- LRU(least-recently-used) Page Replacement
  - The major problem is how to implement LRU replacement.

가장 오랫동안 사용되지 않았던 *(과거 정보)*를 *replacement*

✓ Counters

❖ Associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter

❖ The clock is incremented for every memory reference.

❖ Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.

❖ We replace the page with the smallest time value.

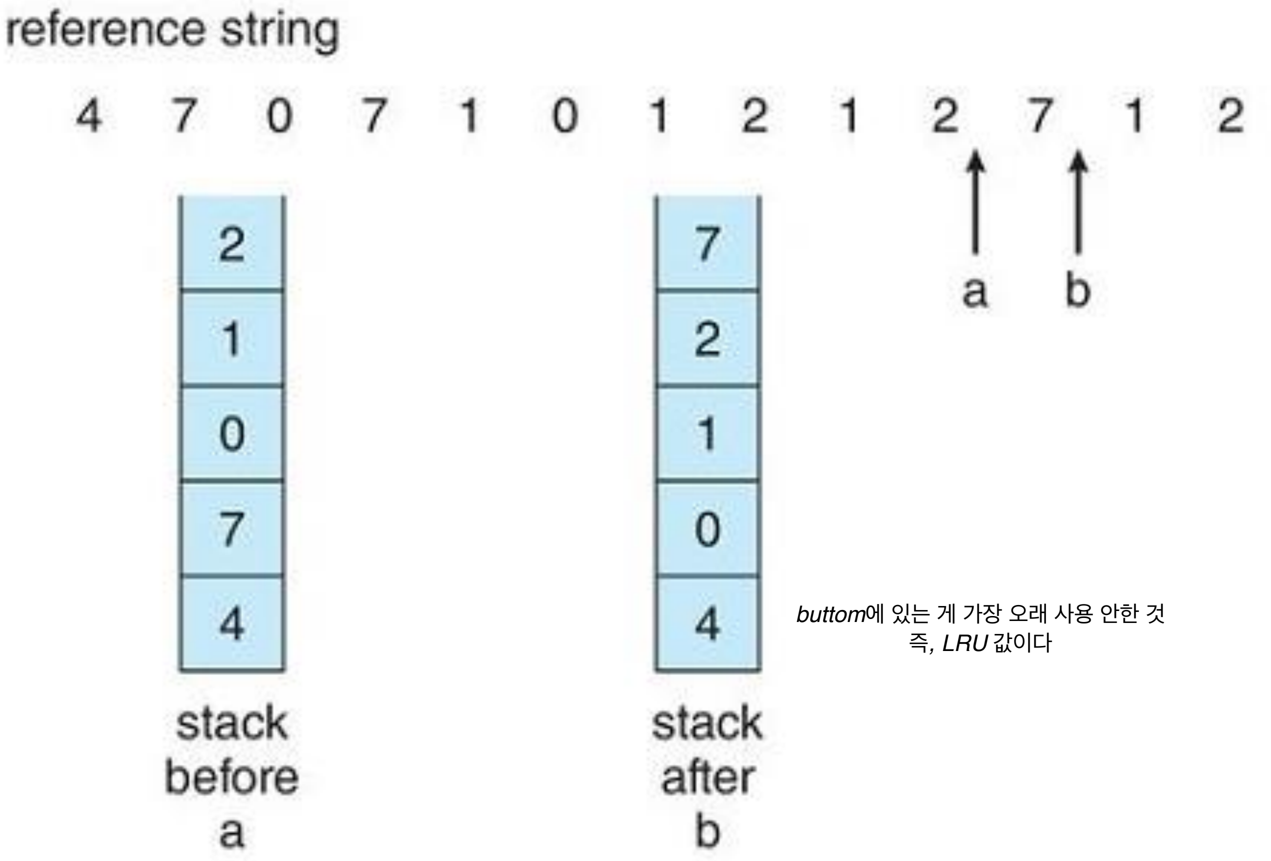
✓ Stack

❖ Whenever a page is referenced, it is removed from the stack and put on the top.

❖ The most recently used page is always at the top of the stack, and the least recently used page is always at the bottom.

❖ Figure 10.16

Page Replacement



*buttom*에 있는 게 가장 오래 사용 안한 것
즉, *LRU* 값이다

Figure 10.16 Use of a stack to record the most recent page references.

Page Replacement

- LRU-Approximation Page Replacement
  - The reference bit for a page is set by the hardware whenever that page is referenced.
  - We can determine which pages have been used and which have not been used by examining the reference bits, although we do not know the order of use.
  - Additional-Reference-Bits Algorithm
    - ✓ We can keep an 8-bit byte for each page in a table in memory.
    - ✓ The OS shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit and discarding the low-order bit.
    - ✓ If the shift register contains 00000000, for example, then the page has not been used for eight time periods; A page that is used at least once in each period has a shift register value of 11111111.
    - ✓ A page with a history register value of 11000100 has been used more recently than one with a value of 01110111. 큰 값이 가장 최근에 *reference* 그리고 작은 값은 나중에 *reference*

Page Replacement

- LRU-Approximation Page Replacement
  - Second-Chance Algorithm
    - ✓ The basic algorithm of second-chance replacement is a FIFO replacement algorithm.
    - ✓ If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page.
    - ✓ When a page gets a second chance, its reference bit is cleared.
      - ❖ A page that is given a second chance will not be replaced until all other pages have been replaced.
    - ✓ One way to implement the second-chance algorithm is as a circular queue.
    - ✓ Figure 10.17
      - ❖ When a frame is needed, the pointer advances until it finds a page with a 0 reference bit.

Page Replacement

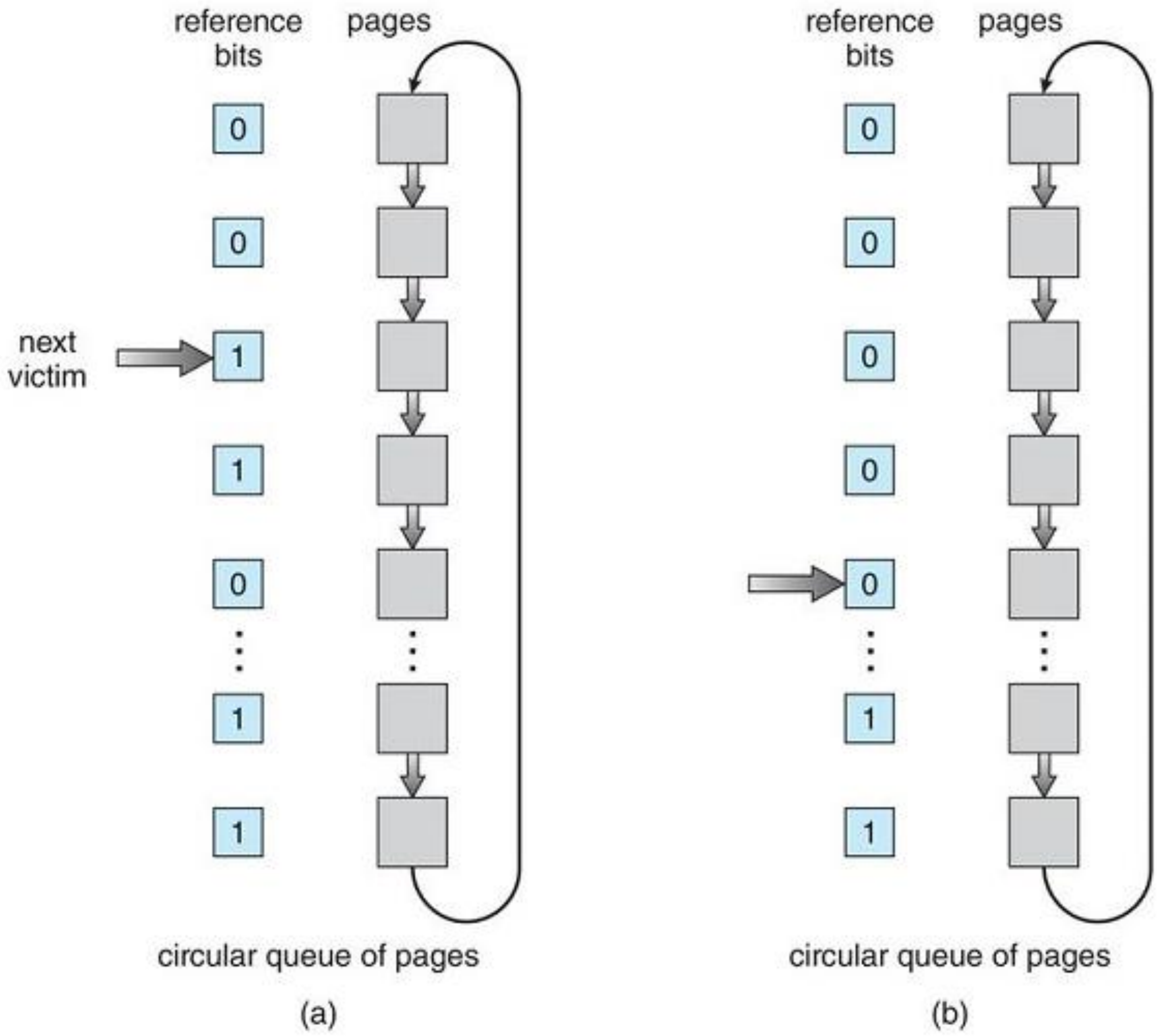


Figure 10.17 Second-chance (clock) page-replacement algorithm.

Page Replacement

- LRU-Approximation Page Replacement
  - Enhanced Second-Chance Algorithm
    - ✓ We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair.
    - ✓ The following four possible classes
      - ❖ (0, 0) neither recently used nor modified – best page to replace
      - ❖ (0, 1) not recently used but modified – not quite as good, because the page will need to be written out before replacement
      - ❖ (1, 0) recently used but clean – probably will be used again soon
      - ❖ (1, 1) recently used and modified – probably will be used again soon, and the page will be need to be written out to secondary storage before it can be replaced

Page Replacement

- Counting-Based Page Replacement
  - We can keep a counter of the number of references that have been made to each page
    - ✓ least frequently used (LFU) page-replacement algorithm
      - ❖ The page with the smallest count be replaced
      - ❖ The reason for this selection is that an actively used page should have a large reference count.
      - ❖ A problem arises when a page is used heavily during the initial phase of a process but then is never used again
    - ✓ most frequently used (MFU) page-replacement algorithm
      - ❖ based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Page Replacement

- Page-Buffering Algorithms
  - Keep a pool of free frames but to remember which page was in each frame
    - ✓ Since the frame contents are not modified when a frame is written to secondary storage, the old page can be reused directly from the free-frame pool if it is needed before that frame is reused.
    - ✓ When a page fault occurs, we first check whether the desired page is in the free-frame pool. If it is not, we must select a free frame and read into it.