IOS102

OPERATING SYSTEM
Module 4: VIRTUAL MEMORY

Objectives

At the end of the course, the student should be able to:

- *Define virtual memory;*
- *Discuss the demand paging;*
- *Define page replacement;*
- *Discuss different page-replacement algorithms such as FIFO, Optimal Algorithm, LRU Algorithm and Counting-Based Page Algorithm.*

Virtual Memory

Introduction

- The various memory management strategies that are used in computer systems have the same goal: to keep many processes in memory simultaneously to allow multiprogramming. However, they tend to require the entire process to be in memory before the process can execute.

Virtual Memory

- ***Virtual Memory*** is a technique that allows the execution of processes that may not be completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.
- Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This technique frees programmers from the concerns of memory-storage limitations.

Virtual Memory

- Requiring that the entire process to be in main memory before they can be executed limits the size of a program to the size of physical memory.
- However, an examination of real programs shows that, it need not be in main memory at the same time. For example:

  1. Programs often have code to handle unusual error conditions. Since these errors seldom, if ever, occur in practice, this code is almost never executed.

Virtual Memory

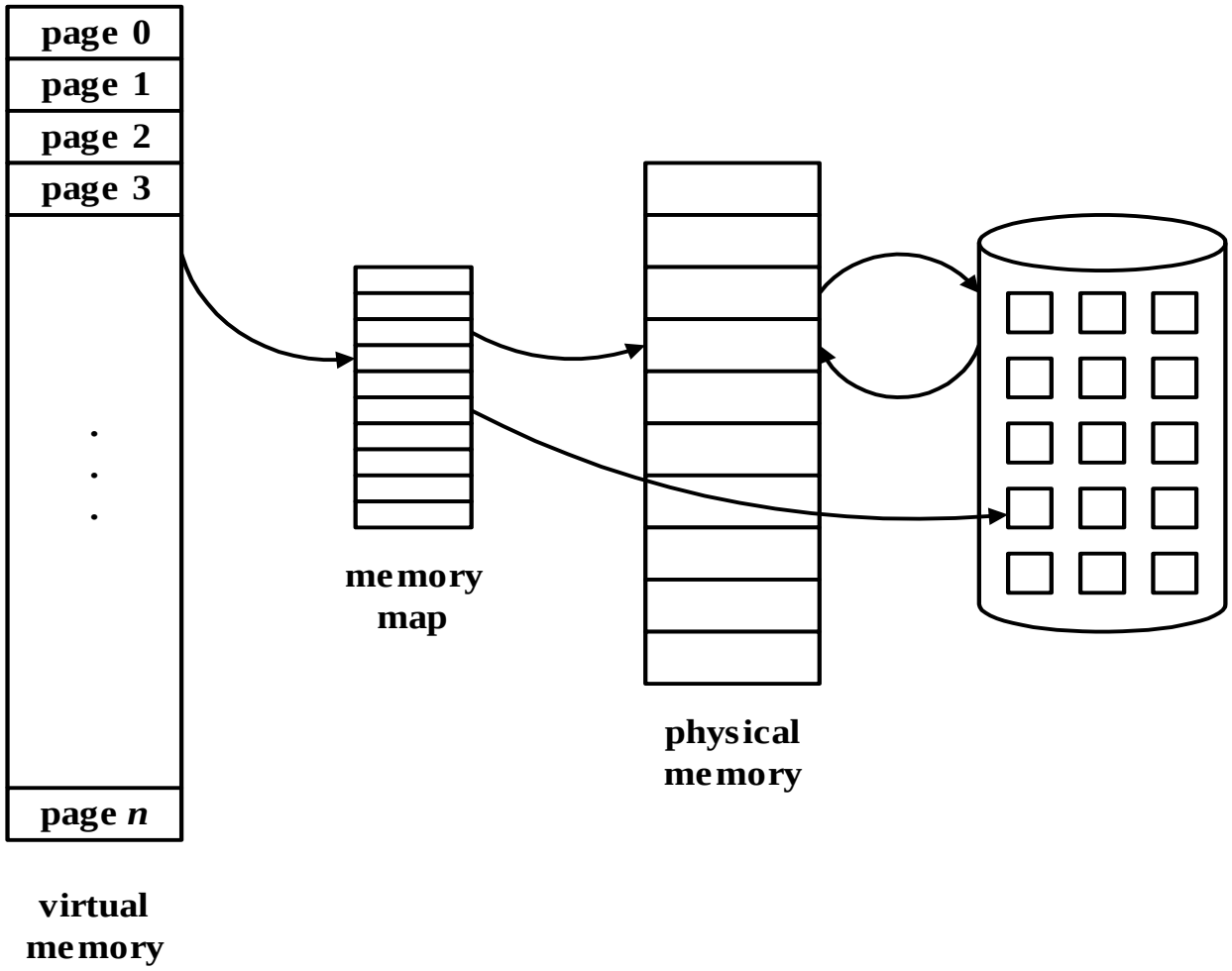
The ability of execute a program that is only partially in memory would confer many benefits:

1. A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual-address space.
2. Because each user program could take less physical memory, more programs could run at the same time.
3. Less I/O would be needed to load or swap each user program into memory, so each program would run faster.

Virtual Memory

- Virtual memory is the separation of user logical memory from physical memory. This separation allows programmers to have a very large virtual memory when only a small physical memory is available.

Virtual Memory



Virtual Memory

- Virtual memory makes the task of programming much easier, since the programmer no longer needs to worry about the amount of physical memory available but can concentrate instead on the programming problem.

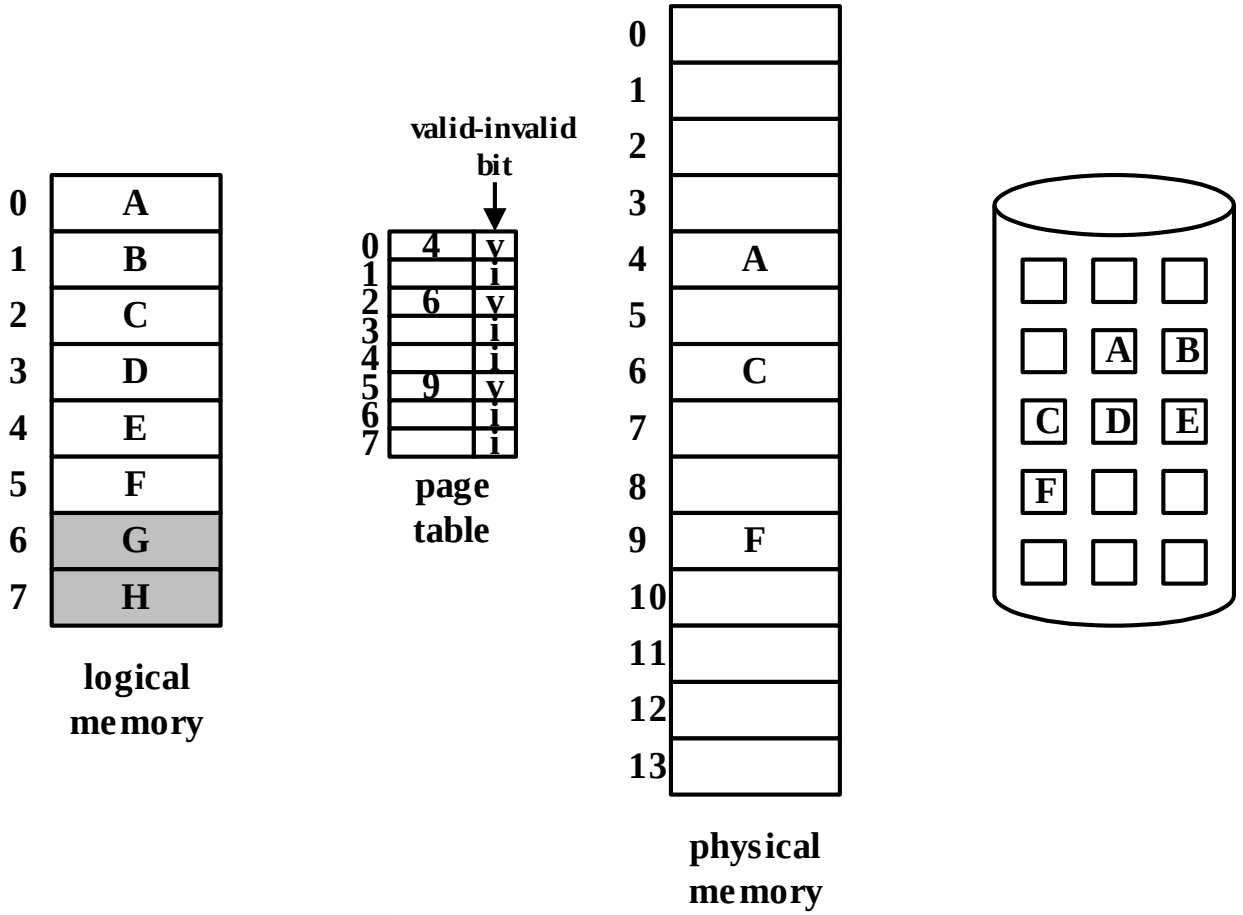
Virtual Memory

Demand Paging

- A demand-paging system is similar to a paging system with swapping. However, instead of swapping the entire process into memory, the OS (particular the ***pager***) swaps only the necessary pages into memory (***lazy swapping***).

Virtual Memory

Demand Paging



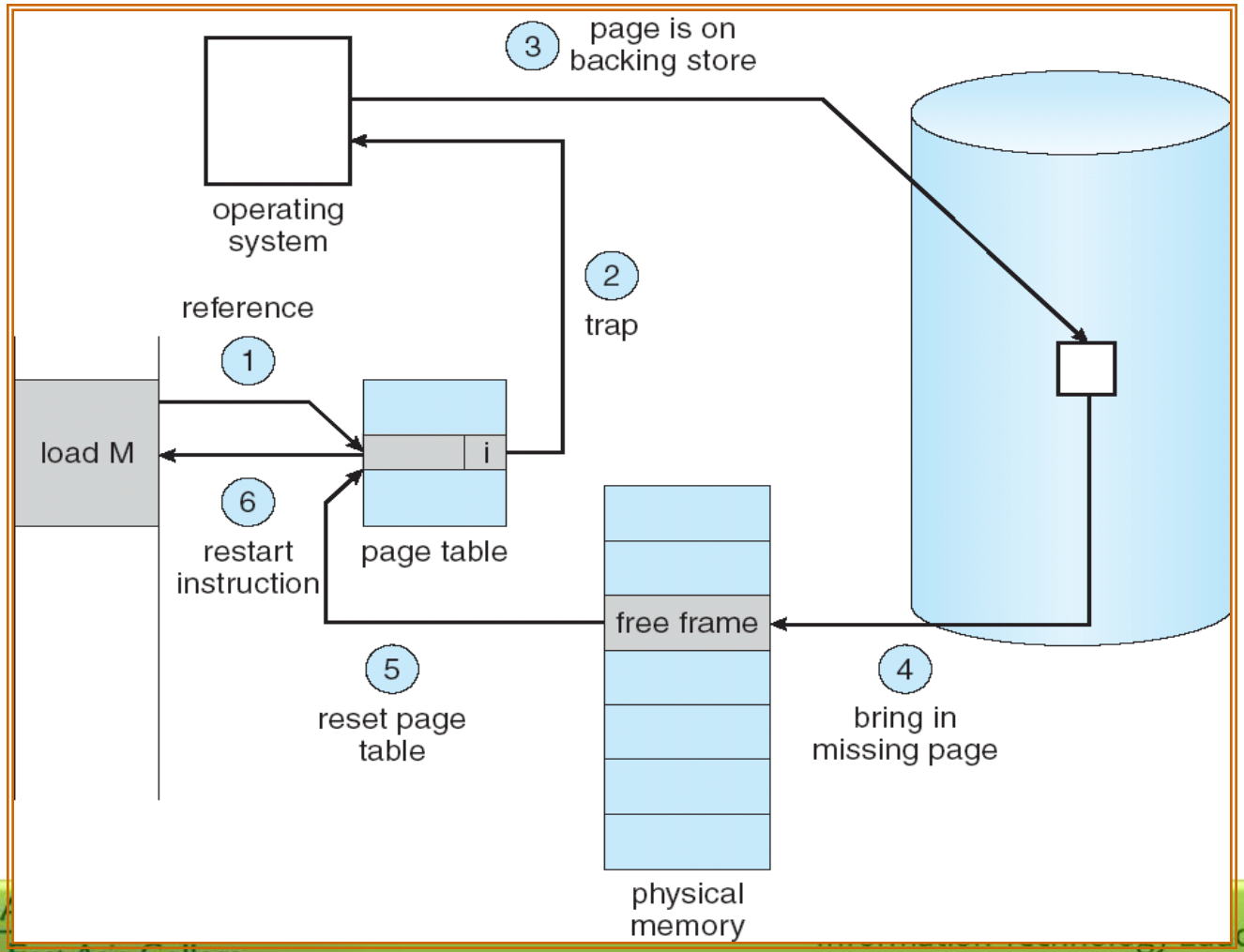
Virtual Memory

Demand Paging

- There is an additional bit in the page table which is the ***valid-invalid bit***. This bit is set to *valid* to indicate that a corresponding page is in memory. This bit is set to *invalid* to indicate that the corresponding page is in secondary storage.
- If a process tries to use a page that is not in physical memory, then a ***page-fault*** will occur. This will cause a trap to the operating system indicating an invalid address error.

Virtual Memory

- Steps in handling a page fault:



Virtual Memory

- Steps in handling a page fault:

  1. Check the internal page table of the process to determine whether the reference was a valid or invalid memory access.
  2. If the reference is invalid (the page is not in main memory), an illegal address trap is initiated and the process is temporarily terminated.
  3. Find a free frame in main memory and allocate this to the incoming page.

Virtual Memory

- Steps in handling a page fault:

  4. Schedule a disk operation to read the desired page into the newly allocated frame.
  5. When the disk read is complete, modify the page table of the process to indicate that the page is now in memory.
  6. Restart the instruction that was interrupted by the illegal address trap. The process can now access the page as if it had always been in memory.

Virtual Memory

- In the extreme case, the system could start executing a process with no pages in memory. The process would immediately fault for the page with the first instruction.

- After the first page is brought into memory, the process would continue to execute, faulting as necessary until every page that it needed was actually in memory. This is ***pure demand paging***: never bring a page into memory until it is required.

Virtual Memory

- The principle of ***locality of reference*** ensures that programs do not access a new page of memory with each instruction execution.

- The effectiveness of the demand paging is based on a property of computer programs called the locality of reference. Analysis of programs shows that most of their execution time is spent on routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures or functions that repeatedly call each other.

Virtual Memory

- If is important to keep the page-fault rate low in a demand-paging system. Otherwise, the effective access time increases, slowing down process execution dramatically.

Virtual Memory

Page Replacement

- A problem occurs if there is a need to transfer a page from disk to memory but there is no memory space available (there are no free frames). In other words, memory is ***over-allocated***.
- The operating system has several options at this point. It could terminate the user process. However, demand paging is the operating system's attempt to improve the computer system's utilization and throughput. Users should not be aware their processes are running on a paged system – paging should be logically transparent to the user.

Virtual Memory

Page Replacement

- Another possibility is ***page replacement***. In this scheme, the operating system removes or replaces one of the existing pages in memory to give way for the incoming page.

- A page replacement algorithm is necessary to select which among the pages currently residing in memory will be replaced.

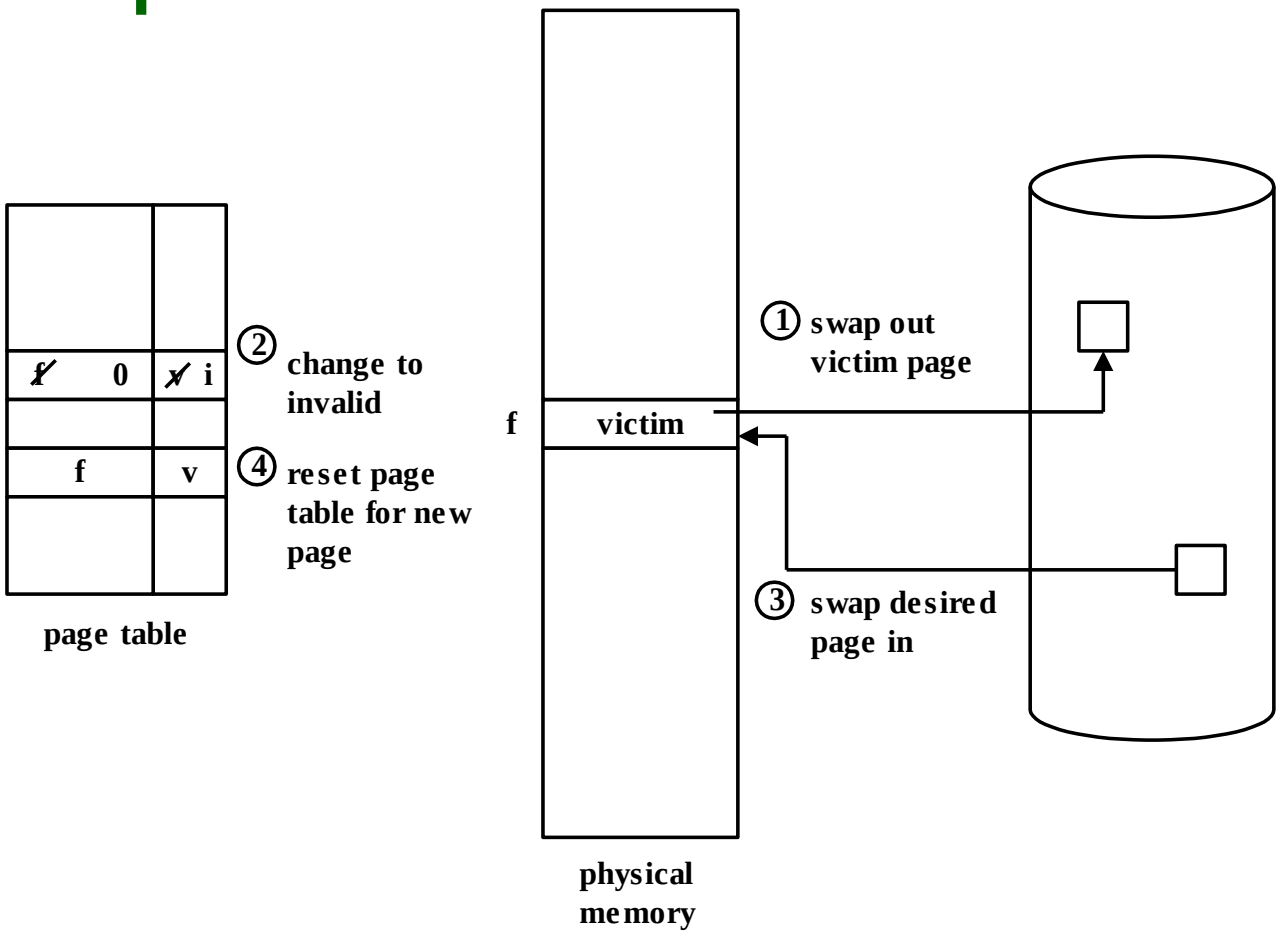
Virtual Memory

Page Replacement

- Page replacement takes the following approach. If no frame is free, the system finds one that is currently being used and frees it. Freeing a frame means transferring its contents to the disk and changing the page table (and all other tables) to indicate that the page is no longer in memory.

Virtual Memory

Page Replacement



Virtual Memory

The page fault service routine is now modified to include page replacement:

1. Find the location of the desired page on the disk.
2. Find a free frame:
   - 2.1. If there is a free frame, use it.
   - 2.2. Otherwise, use a page-replacement algorithm to select a *victim* frame.
   - 2.3. Write the victim page to the disk; change the page and frame tables accordingly.
3. Read the desired page into the (newly) free frame; change the page and frame tables.
4. Restart the user process.

Virtual Memory

- Notice that, if no frames are free, two page transfers (one out and one in) are required. This situation effectively doubles the page-fault service time and will increase the effective access time accordingly.
- To reduce this overhead, a ***modify*** or ***dirty bit*** is necessary for each page or frame. The modify bit for a page is set whenever any word or byte is written into, indicating that the page has been modified.

Virtual Memory

- It is no longer necessary to swap out pages whose modify bit is 0 (there was no modification). An incoming page may simply overwrite an unchanged page.

- There are two major problems in the implementation of demand paging:

  1. Frame Allocation

  How many frames will the operating system allocate to a process, particularly if there are multiple processes?

Virtual Memory

2. Page Replacement

How will the operating system select pages that are to be removed from memory to give way for incoming pages?

Virtual Memory

Page-Replacement Algorithms

- A good page-replacement algorithm is one with a low page-fault rate.

- An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called the ***reference string***.

Virtual Memory

First-In First-Out (FIFO) Algorithm

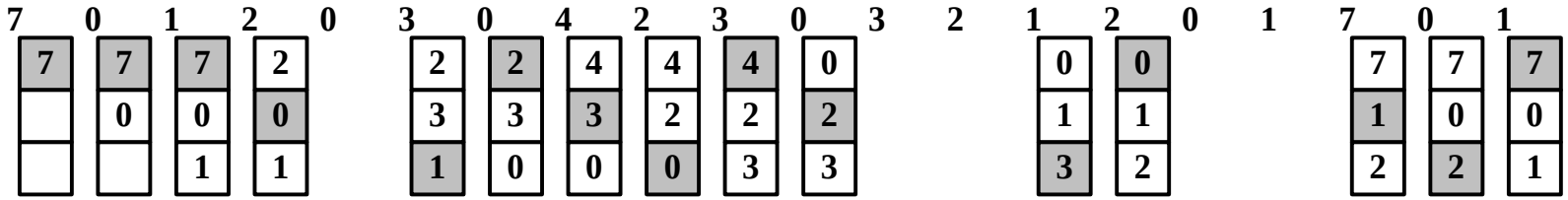
•This is the simplest page-replacement algorithm. When a page must be replaced, the oldest page is chosen.

Example:

Assume that there are three frames in memory and that the following is the page reference string:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Virtual Memory



- There are 15 faults altogether.

Virtual Memory

First-In First-Out (FIFO) Algorithm

•It is not strictly necessary to record the time when a page is brought in. The operating system simply creates a FIFO queue to hold all pages in memory. The page at the head of the queue is replaced if a frame is needed. When a page is brought into memory, it is inserted at the tail of the queue.

Virtual Memory

First-In First-Out (FIFO) Algorithm

•The FIFO page-replacement algorithm is easy to understand and program. However, its performance of is not always good. The page replaced may be an initialization module that was used a long time ago and is no longer needed. On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

Virtual Memory

First-In First-Out (FIFO) Algorithm

•Notice that, even if algorithm selects for replacement a page that is in active use, everything still works correctly. After an active page is removed to bring in a new one, a fault occurs almost immediately to retrieve the active page. Thus, a bad replacement choice increases the page-fault rate and slow process execution, but does not cause incorrect execution.

Virtual Memory

First-In First-Out (FIFO) Algorithm

- As a general rule, the more frames available in physical memory, the lower the page-fault rate.

- However, there are some instances where the page-fault rate may increase as the number of physical memory frames increases. This is known as ***Belady's Anomaly***.

Virtual Memory

First-In First-Out (FIFO) Algorithm

- The FIFO algorithm suffers from Belady's Anomaly.

Example:

Consider the reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

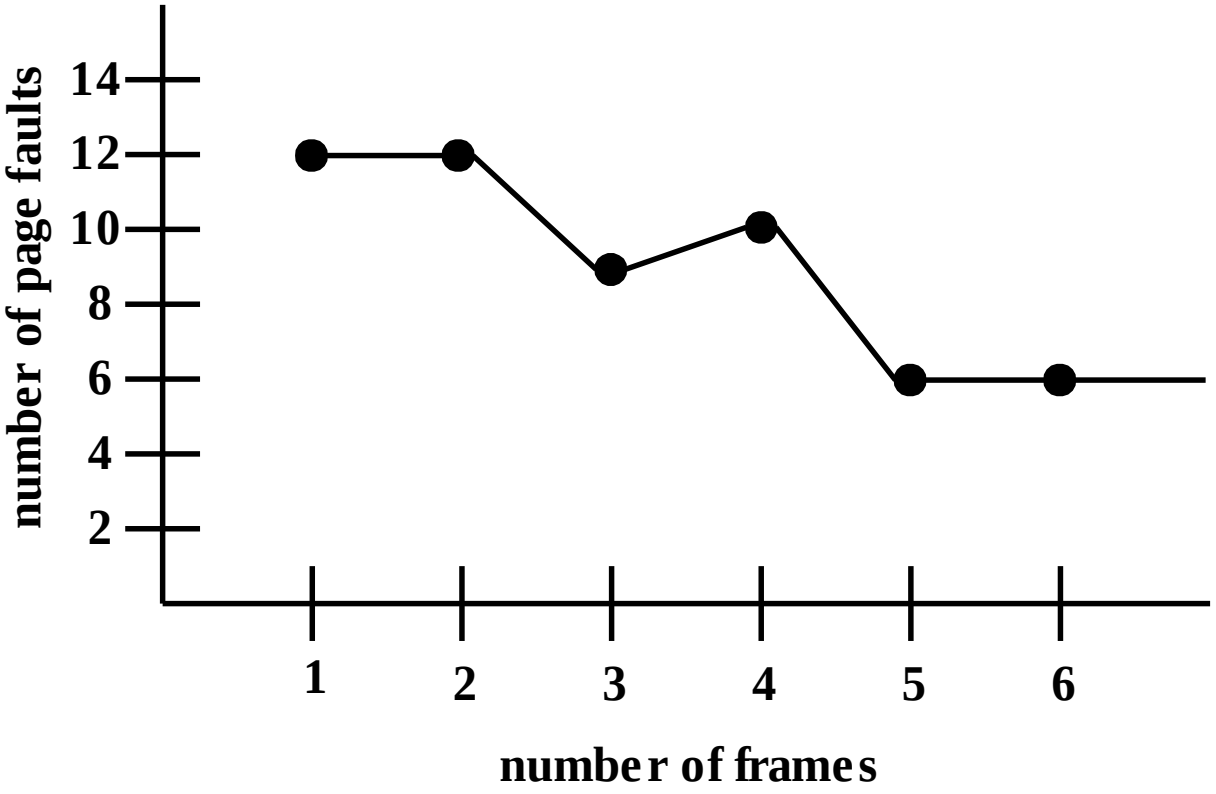
Virtual Memory

First-In First-Out (FIFO) Algorithm

Number of Frames	Page Faults
1	12
2	12
3	9
4	10
5	5
6	5

Virtual Memory

First-In First-Out (FIFO) Algorithm



Virtual Memory

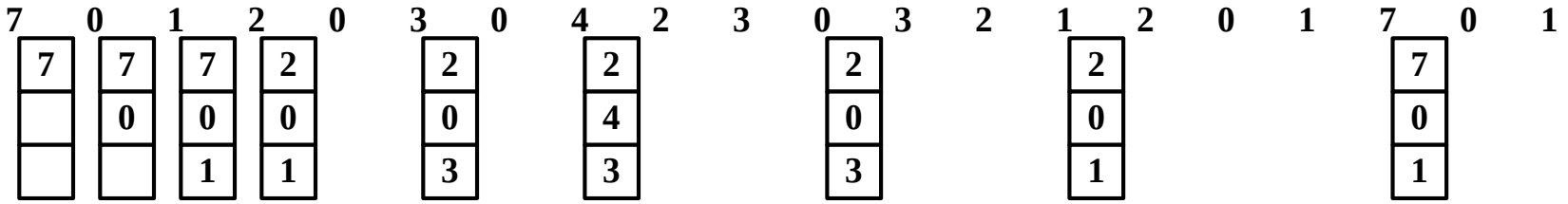
Optimal Algorithm

- An optimal algorithm has the lowest page-fault rate of all algorithms.
- In an optimal algorithm, the page that will be replaced is the one that will not be used for the longest period of time.

Virtual Memory

Optimal Algorithm

Example:



Virtual Memory

Optimal Algorithm

- In the example, the optimal page-replacement algorithm produced only 9 page faults as compared to the FIFO algorithm which produced 15 page faults. If the first three page faults were ignored (since all algorithms suffer from these initialization page faults), then the optimal algorithm is twice as good as FIFO.

Virtual Memory

Optimal Algorithm

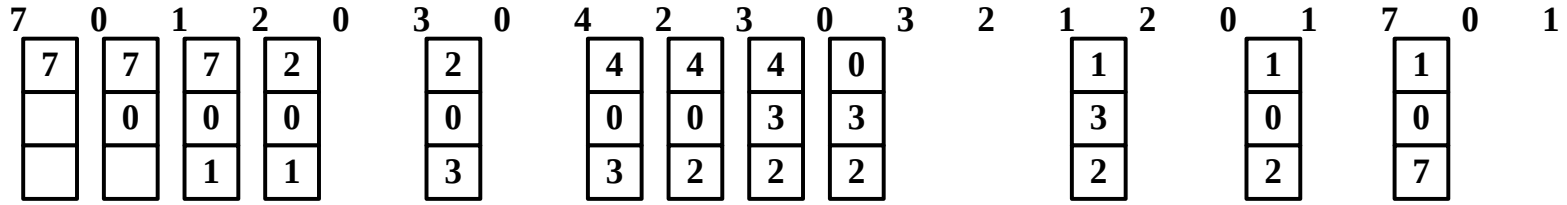
- Unfortunately, this algorithm is difficult to implement, since it requires future knowledge of the reference string. The optimal algorithm is used mainly for comparison studies.

Virtual Memory

Least Recently Used (LRU) Algorithm

- The LRU algorithm uses the recent past to approximate the near future. It simply replaces the page that has not been used for the longest period of time.

Example:



Virtual Memory

Least Recently Used (LRU) Algorithm

- The LRU algorithm produced 12 page faults. Although it is not as good as the 9 of the optimal algorithm (in fact, no other algorithm will produce less than 9 page faults for this example), it is much better than the 15 of the FIFO algorithm.
- The LRU policy is often used as a page replacement algorithm and is considered to be quite good. The major problem is how to implement LRU replacement.

Virtual Memory

Least Recently Used (LRU) Algorithm

- The LRU algorithm produced 12 page faults. Although it is not as good as the 9 of the optimal algorithm (in fact, no other algorithm will produce less than 9 page faults for this example), it is much better than the 15 of the FIFO algorithm.
- The LRU policy is often used as a page replacement algorithm and is considered to be quite good. The major problem is how to implement LRU replacement.

Virtual Memory

Least Recently Used (LRU) Algorithm

- An LRU page-replacement algorithm may require substantial hardware assistance. The problem is to determine an order for the frames defined by the time of last use. Two implementations are feasible:

Virtual Memory

Least Recently Used (LRU) Algorithm

1. ***Counters***. In the simplest case, the operating system associates with each page-entry table a ***time-of-use field***, and add to the CPU a logical clock or counter. Whenever a process references a page, the system copies the contents of CPU's clock register to the time-of-use field. In this way, the system has the "time" of the last reference to each page. The system replaces the page with the smallest time value. This solution obviously requires hardware support.

Virtual Memory

Least Recently Used (LRU) Algorithm

2. ***Stack***. Another implementation method is through the use of a stack. The system maintains a stack of page numbers. Whenever a process references a page, the system places the page number on top of the stack. This ensures that the page number at the bottom of the stack is for the least recently used page. This approach is particularly appropriate for software or microcode implementations of the LRU algorithm.

Virtual Memory

Counting-Based Page Replacement

- Counting-based Page Replacement keeps a counter of the number of references that have been made to each page.

- Two examples of Counter-Based Page Replacement algorithms:

Virtual Memory

Counting-Based Page Replacement

1. ***The Least Frequently Used (LFU) Algorithm*** requires that 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.

Virtual Memory

Counting-Based Page Replacement

- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.

Virtual Memory

Counting-Based Page Replacement

2. ***The Most Frequently Used (LFU) Algorithm*** is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.