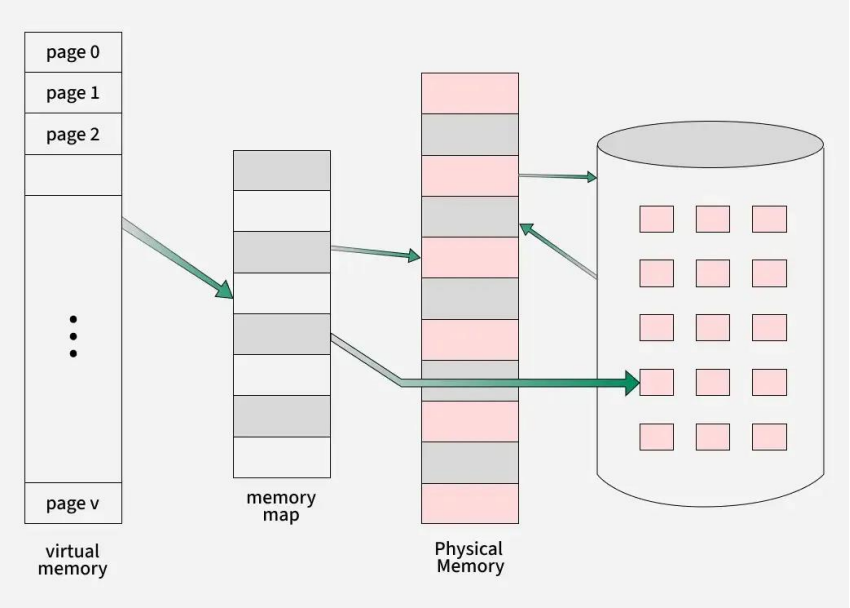
5. VIRTUAL MEMORY

Virtual Memory in Operating System

Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited. It allows larger applications to run on systems with less RAM.

- ✓ The main objective of virtual memory is to support multiprogramming, The main advantage that virtual memory provides is, a running process does not need to be entirely in memory.
- ✓ Programs can be larger than the available physical memory. Virtual Memory provides an abstraction of main memory, eliminating concerns about storage limitations.
- ✓ A memory hierarchy, consisting of a computer system's memory and a disk, enables a process to operate with only some portions of its address space in RAM to allow more processes to be in memory.



- ✓ A virtual memory is what its name indicates- it is an illusion of a memory that is larger than the real memory.
- ✓ We refer to the software component of virtual memory as a virtual memory manager. The basis of virtual memory is the noncontiguous memory allocation model.
- ✓ The virtual memory manager removes some components from memory to make room for other components.

- ✓ The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations.

How Virtual Memory Works?

Virtual Memory is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

- ✓ All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
- ✓ A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permit this.
- ✓ Virtual memory is implemented using Demand Paging or Demand Segmentation.

Types of Virtual Memory

In a computer, virtual memory is managed by the Memory Management Unit (MMU), which is often built into the CPU. The CPU generates virtual addresses that the MMU translates into physical addresses.

There are two main types of virtual memory:

- ✓ Paging
- ✓ Segmentation

Paging

- ✓ Paging divides memory into small fixed-size blocks called pages.
- ✓ When the computer runs out of RAM, pages that aren't currently in use are moved to the hard drive, into an area called a swap file. The swap file acts as an extension of RAM.
- ✓ When a page is needed again, it is swapped back into RAM, a process known as page swapping.
- ✓ This ensures that the operating system (OS) and applications have enough memory to run.

Segmentation :-

Segmentation is also a memory management technique where memory is partitioned into variable-sized blocks that are commonly known as **Segments**.

- Segments: Programs are divided into logical units, like code, data, stack, and heap, that are stored in segments. Each segment has a unique identifier and is of variable size, matching the needs of the program.
- Segment Table: The operating system utilizes a segment table to keep track of information about each segment, including its base address, size (limit), and access rights.

Logical vs. Physical Address: The CPU generates a logical address containing the segment number and an offset within that segment. This logical address is then translated into a physical address by the Memory Management Unit (MMU) using the segment table

Demand Paging:

DEMAND PAGING

Demand paging is a memory management technique where pages are loaded in main memory only when they are ***demanded*** during program execution.

- **Lazy swapper** – It is a concept that never swaps a page into memory unless page will be needed

- Swapper that deals with pages is a **pager**
  - While a process is executing, some pages will be in memory, and some will be in secondary storage. Thus, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk.

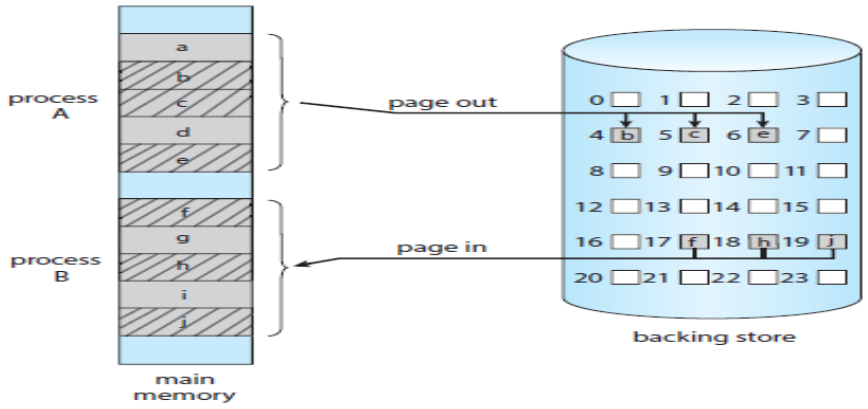


Figure Swapping with paging.

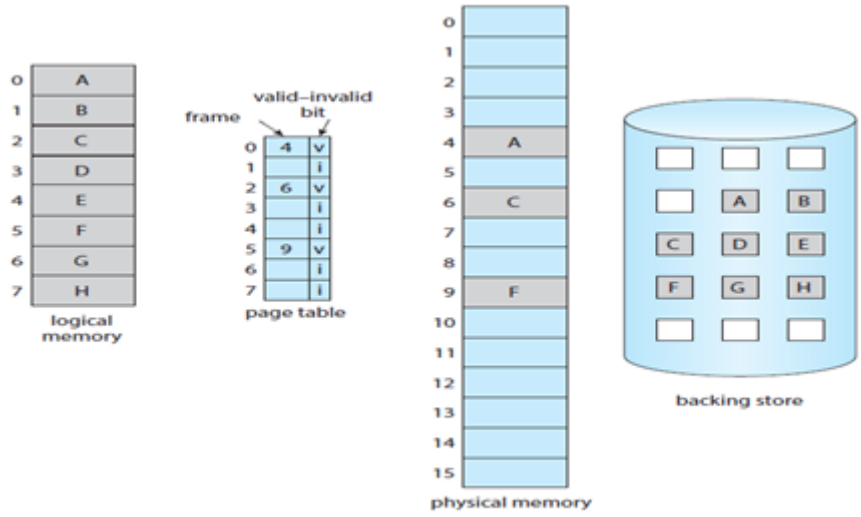


Figure Page table when some pages are not in main memory.

The valid/invalid bit scheme can be used for this purpose. Each entry in the page table has minimum two fields: page frame and valid-invalid bit.

- **If the bit is set to "valid," the page is legal and its present in memory. (vaild-1)**
- **If the bit is set to "invalid," the page is currently not in main memory. (invalid-0)**
    - Access to a page which is invalid causes a **page fault.**

➢ Page fault result causes a trap to the operating system, which indicate the failure and asked to bring the desired page into memory.

➢ When the page fault occur the execution is suspended until missing page is brought into memory.

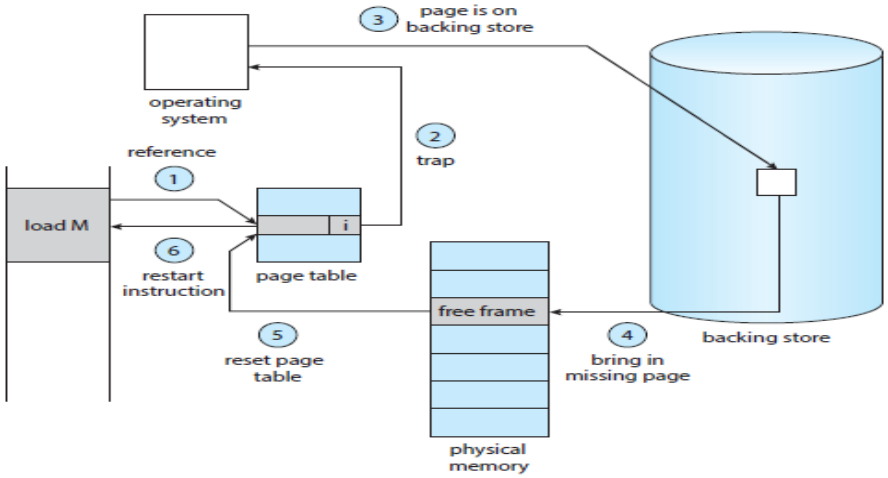


Figure Steps in handling a page fault.

The procedure(steps) for handling this page fault is straightforward

1. Check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

2. If the reference was invalid, terminate the process. If it was valid but not yet brought in that page, now page it in.

3. Find a free frame.

4. We schedule a secondary storage operation to read the desired page into the newly allocated frame.

5. When the storage 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.

6. Restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Page vs Frame

- ✓ A page is a fixed size block of data in virtual memory and a frame is a fixed size block of physical memory in RAM where these pages are loaded.
- ✓ Think of a page as a piece of a puzzle (virtual memory) and a frame as the spot where it fits on the board (physical memory).
- ✓ When a program runs its pages are mapped to available frames so the program can run even if the program size is larger than physical memory.

Sometimes, both paging and segmentation are used together. In this case, memory is divided into pages, and segments are made up of multiple pages. The virtual address includes both a segment number and a page number.

Virtual Memory vs Physical Memory

Feature	Virtual Memory	Physical Memory (RAM)
Definition	An abstraction that extends the available memory by using disk storage	The actual hardware (RAM) that stores data and instructions currently being used by the CPU
Location	On the hard drive or SSD	On the computer's motherboard
Speed	Slower (due to disk I/O operations)	Faster (accessed directly by the CPU)
Capacity	Larger, limited by disk space	Smaller, limited by the amount of RAM installed
Cost	Lower (cost of additional disk storage)	Higher (cost of RAM modules)
Data Access	Indirect (via paging and swapping)	Direct (CPU can access data directly)
Volatility	Non-volatile (data persists on disk)	Volatile (data is lost when power is off)

Page Replacement Algorithms

- ✓ In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
- ✓ Page replacement becomes necessary when a page fault occurs and no free page frames are in memory.
- ✓ Page replacement algorithms are techniques used in operating systems to manage memory efficiently when the physical memory is full.
- ✓ When a new page needs to be loaded into physical memory**,** and there is no free space, these algorithms determine which existing page to replace.

- ✓ If no page frame is free, the virtual memory manager performs a page replacement operation to replace one of the pages existing in memory with the page whose reference caused the page fault.
- ✓ It is performed as follows: The virtual memory manager uses a page replacement algorithm to select one of the pages currently in memory for replacement, accesses the page table entry of the selected page to mark it as "not present" in memory, and initiates a page-out operation for it if the modified bit of its page table entry indicates that it is a dirty page.

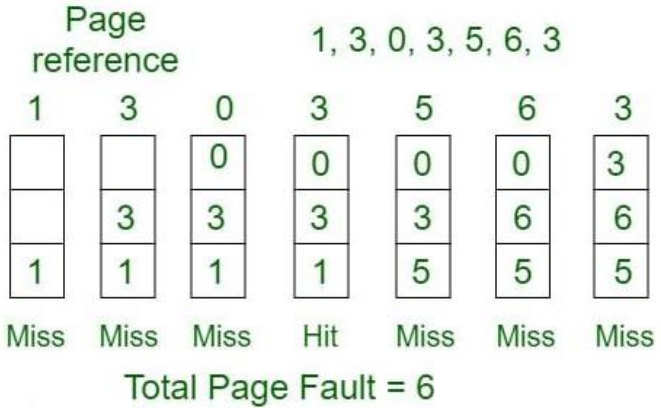
Common Page Replacement Techniques

- ✓ First In First Out (FIFO)
- ✓ Optimal Page replacement
- ✓ Least Recently Used (LRU)
- ✓ Most Recently Used (MRU)

First In First Out (FIFO)

- ✓ This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue.
- ✓ When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3-page frames. Find the number of page faults using FIFO Page Replacement Algorithm.



FIFO – Page Replacement

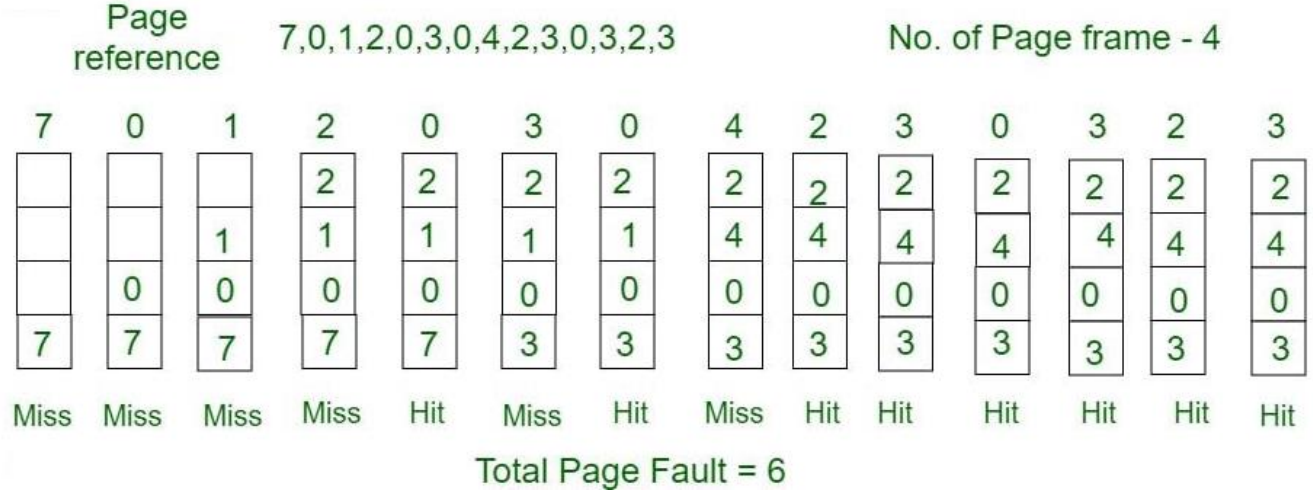
- ✓ *Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.***
- ✓ ***W**hen 3 comes, it is already in memory so —> 0 **Page Faults.***
- ✓ *Then 5 comes, it is not available in memory, so it replaces the oldest page slot i.e 1. —> 1 **Page Fault.***

- ✓ *6 comes, it is also not available in memory, so it replaces the oldest page slot i.e 3 —> 1* ***Page Fault.***
- ✓ *Finally, when 3 come it is not available, so it replaces 0 1-page* ***fault.***

Optimal Page Replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frame. Find number of page fault using Optimal Page Replacement Algorithm.



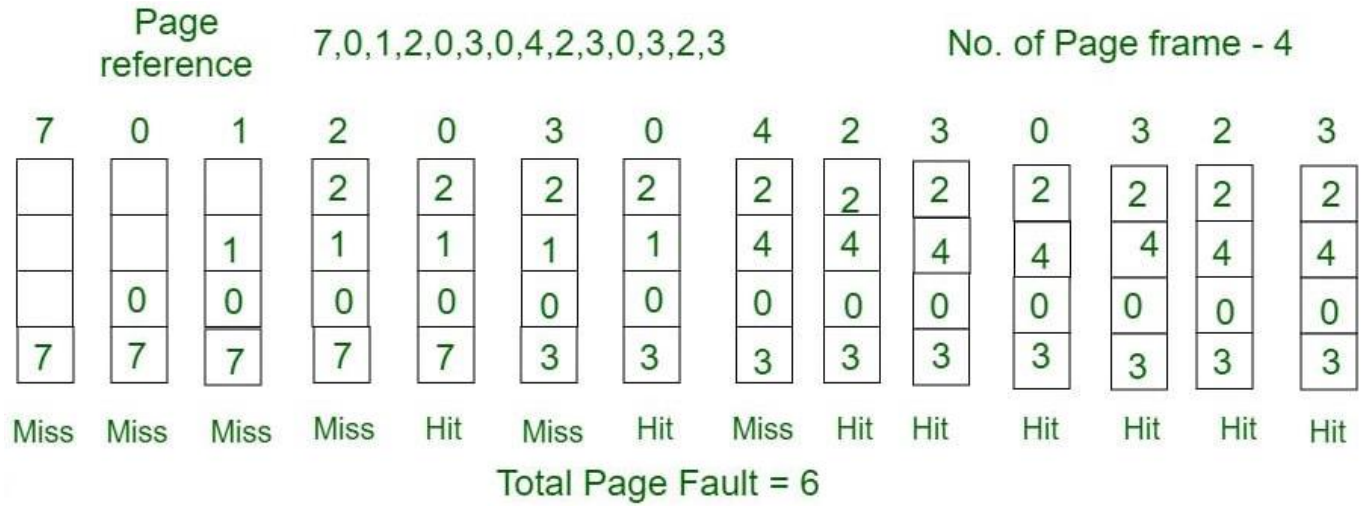
Optimal Page Replacement

- ✓ *Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>* ***4 Page faults .***
- ✓ *0 is already there so —>* ***0 Page fault.***
- ✓ *when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future—>* ***1 Page fault.***
- ✓ *0 is already there so —>* ***0 Page fault.***
- ✓ *4 will takes place of 1 —>* ***1 Page Fault.***
- ✓ *Now for the further page reference string —> 0* ***Page fault because*** *they are already available in the memory.*

Least Recently Used

In this algorithm, page will be replaced which is least recently used.

Example Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using LRU Page Replacement Algorithm.

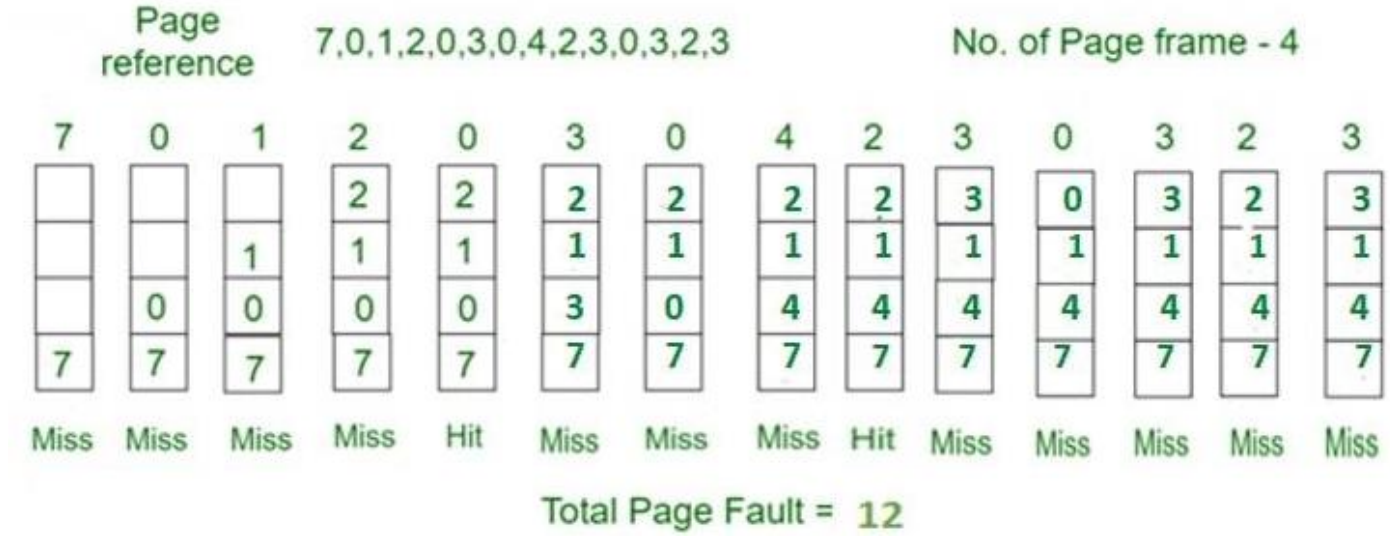


- ✓ *Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults.***
- ✓ *0 is already there so —> **0 Page fault.***
- ✓ ***W**hen 3 came it will take the place of 7 because it is least recently used —>**1Page fault***
- ✓ *0 is already in memory so —> **0 Page fault**.*
- ✓ *4 will takes place of 1 —> **1 Page Fault.***
- ✓ *Now for the further page reference string —> **0 Page fault** because they are already available in the memory.*

Most Recently Used (MRU)

In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

Example 4: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using MRU Page Replacement Algorithm.



Most Recently Used – Page Replacement

- ✓ *Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults***
- ✓ *0 is already there so–> **0 page fault***
- ✓ *when 3 comes it will take place of 0 because it is most recently used —> **1 Page fault***
- ✓ *when 0 comes it will take place of 3 —> **1 Page fault***
- ✓ *when 4 comes it will take place of 0 —> **1 Page fault***
- ✓ *2 is already in memory so —> **0 Page fault***
- ✓ *when 3 comes it will take place of 2 —> **1 Page fault***
- ✓ *when 0 comes it will take place of 3 —> **1 Page fault***
- ✓ *when 3 comes it will take place of 0 —> **1 Page fault***
- ✓ *when 2 comes it will take place of 3 —> **1 Page fault***
- ✓ *when 3 comes it will take place of 2 —> **1 Page fault***