UNIT-IV
Part-2

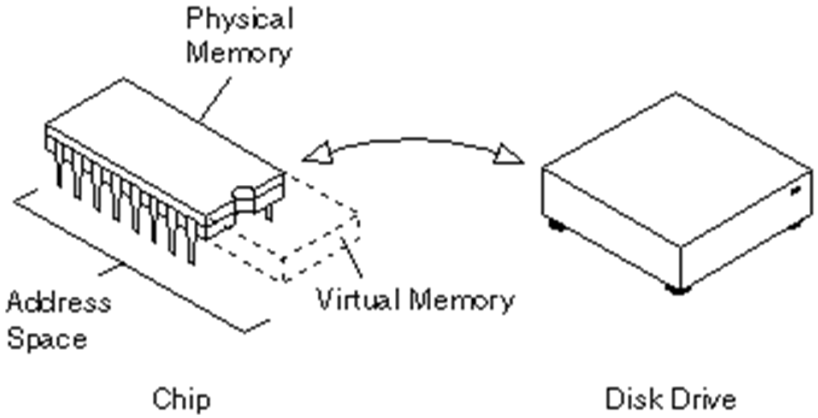
Syllabus:

Virtual Memory Management

1. Virtual Memory
2. Demand Paging
3. Page-Replacement Algorithms
4. Thrashing

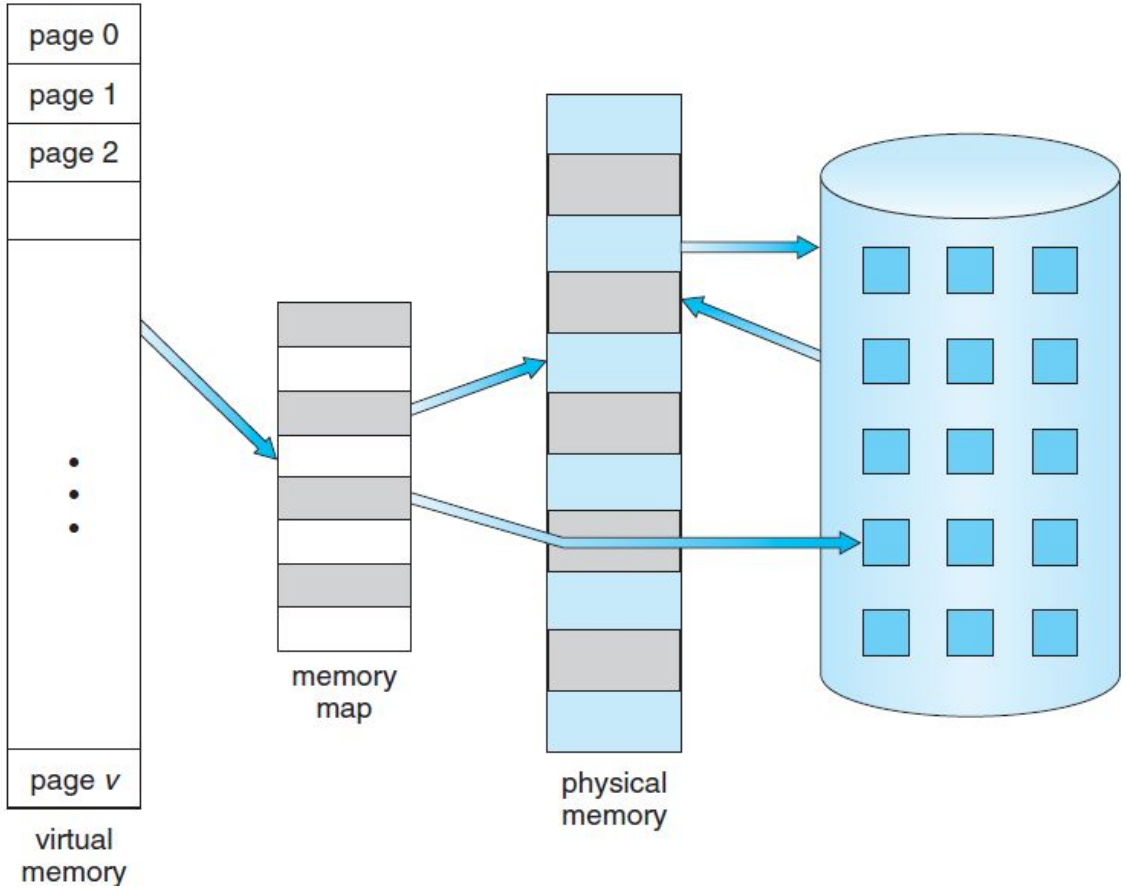
Virtual Memory

- The term *virtual memory* refers to something which appears to be present but actually it is not.
- The virtual memory technique allows users to use more memory for a program than the real memory of a computer.
- So, virtual memory is the concept that gives **the illusion to the user** that they will have main memory equal to the capacity of secondary storage media.



Concept of Virtual Memory

- A programmer can write a program which requires more memory space than the capacity of the main memory. Such a program is executed by virtual memory technique.
- The program is stored in the secondary memory. **The *memory management unit* (MMU)** transfers the currently needed part of the program from the secondary memory to the main memory for execution.
- This to and from movement of instructions and data (parts of a program) between the main memory and the secondary memory is called **Swapping.**



Implementation of Virtual Memory in Operating System

- Virtual memory enables efficient memory management, provides the flexibility to run programs **larger than the available physical memory**.

***Real-time examples that demonstrate the use of virtual memory**

Running Multiple Applications:

- Suppose you have several applications open on your computer, such as a web browser, a word processor, and a media player. Each application requires memory to store its code and data.
- Virtual memory allows these applications to coexist in memory, even if the physical memory is limited.
- The operating system allocates a portion of virtual memory to each application, swapping data between physical memory and disk as needed.

- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Demand Paging

- Consider how an executable program might be loaded from disk into memory.
- **One option** is to load the entire program in physical memory at program execution time.
- However, a problem with this approach is that we may not initially need the entire program in memory.
- An alternative strategy is to load pages only as they are needed. This technique is known as demand paging and is **commonly used in virtual memory systems.**
- With **demand-paged virtual memory**, pages are loaded only when they are demanded during program execution.
- A demand-paging system is similar to a paging system with swapping (Figure 9.4) where processes reside in secondary memory (usually a disk).
- When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, though, we use **a lazy swapper**.
- A lazy swapper **never swaps a page into memory unless that page will be needed**.

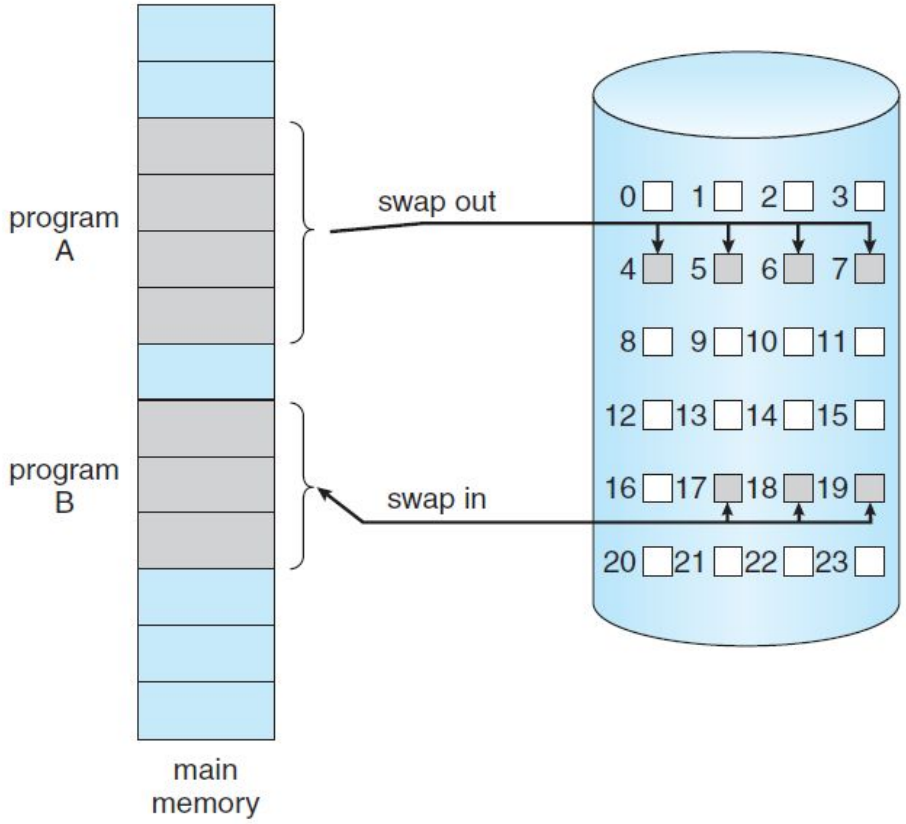


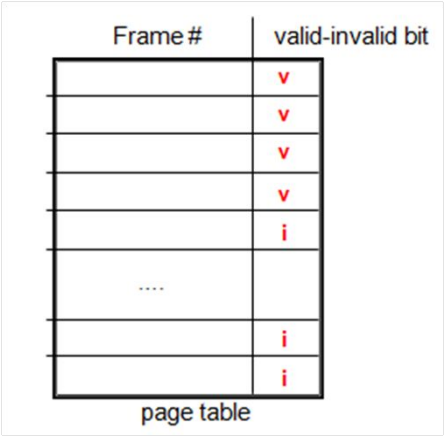
Figure 9.4 Transfer of a paged memory to contiguous disk space.

- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory.
- Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.
- Hardware support is required to distinguish between
    - Those pages that are in **memory**
    - Those pages that are **on the disk** using the **valid-invalid bit** scheme.
- Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages.
- While the process executes and accesses pages that are memory resident, execution proceeds normally.

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated

  (**v** → in-memory – **memory resident**, **i** → not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



- During MMU address translation, if valid–invalid bit in page table entry is **i** → **page fault**

Page Table When Some Pages Are Not in Main Memory

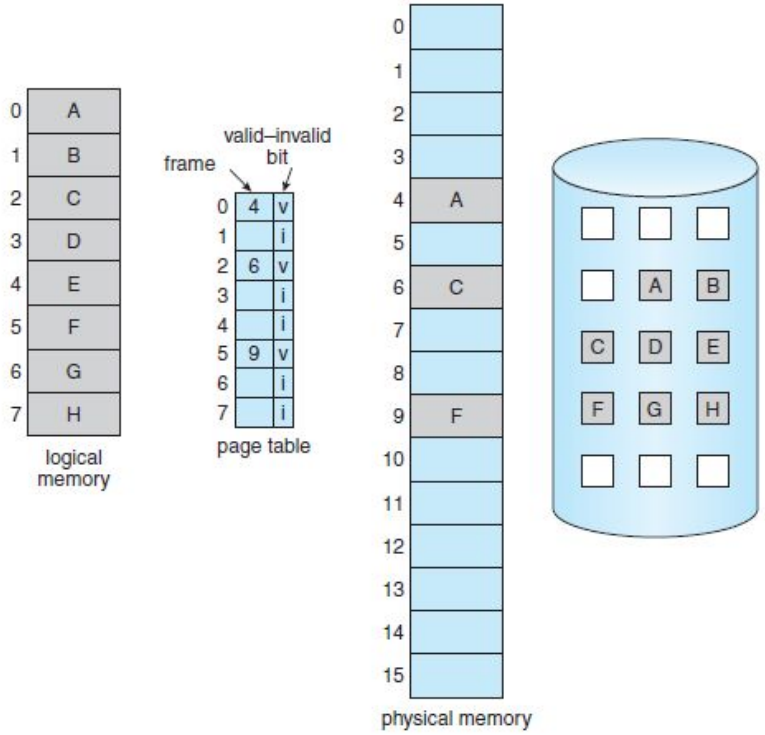


Figure 9.5 Page table when some pages are not in main memory.

When a process references a page that is not currently in memory, **a page fault occurs**, **triggering** the operating system to load the required page from secondary storage into an available page frame in main memory.

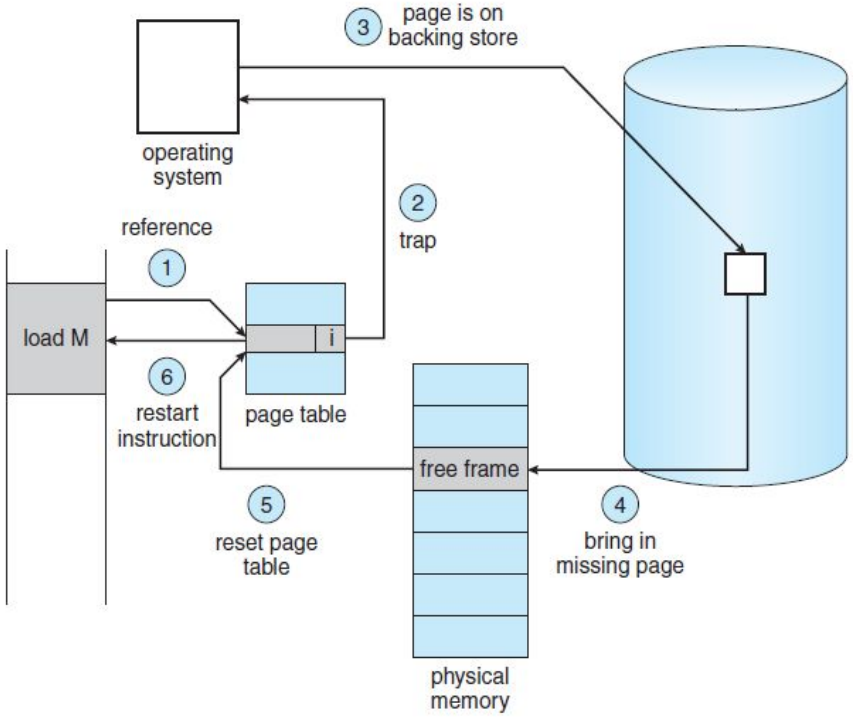


Figure 9.6 Steps in handling a page fault.

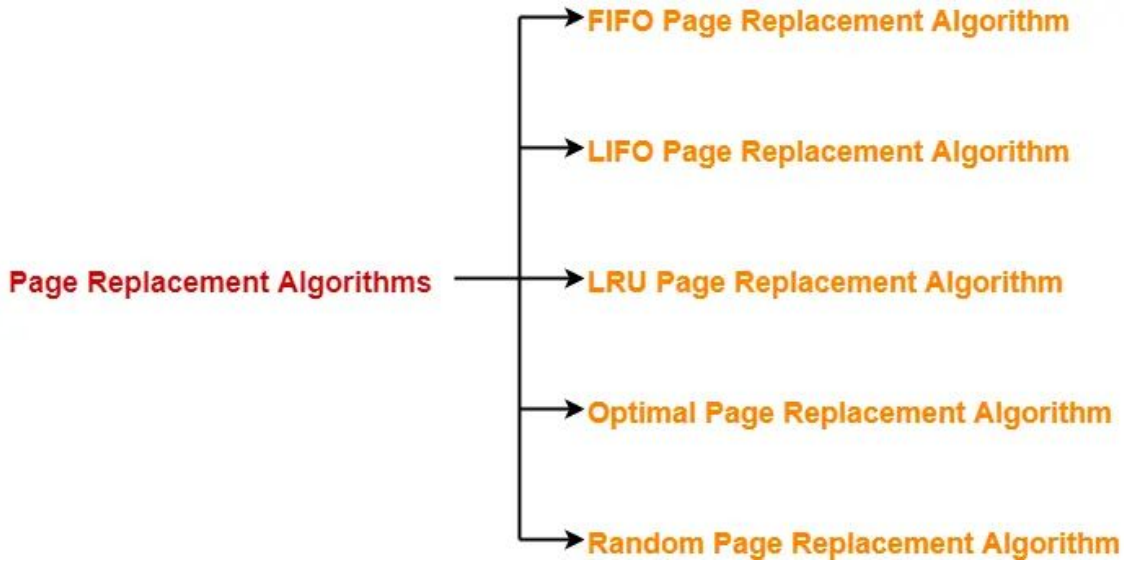
The procedure for handling this page fault is straightforward (Figure -Previous Slide):

1. We 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**, we **terminate the process**. If it was valid but we have not yet brought in that page, we now page it in.
3. We find a free frame (by taking one from the free-frame list, for example).
4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk 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. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

Page Replacement Algorithms

★ Page replacement algorithms are used in demand paging systems to determine which page should be remove from Main Memory when a page fault occurs and there are **no free page frames available.**

★ The goal of these algorithms is to minimize the number of page faults and optimize system performance.

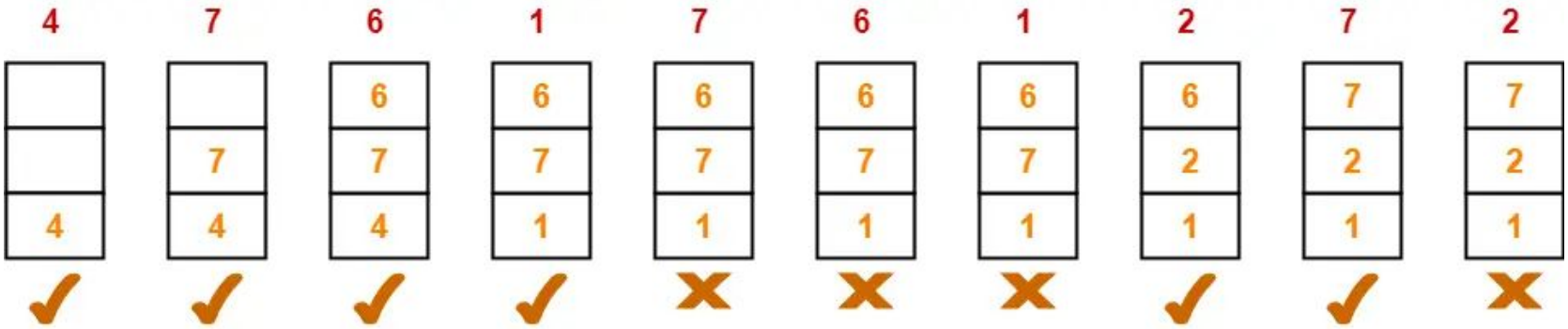


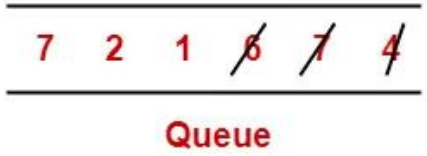
FIFO Page Replacement Algorithm(Belady's anomaly)

- ★ As the name suggests, this algorithm works on the principle of "First in First out".
- ★ It replaces the oldest page that has been present in the main memory for the longest time.
- ★ It is implemented by keeping track of all the pages in a queue.

[Q] A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2





Total number of page faults occurred = 6

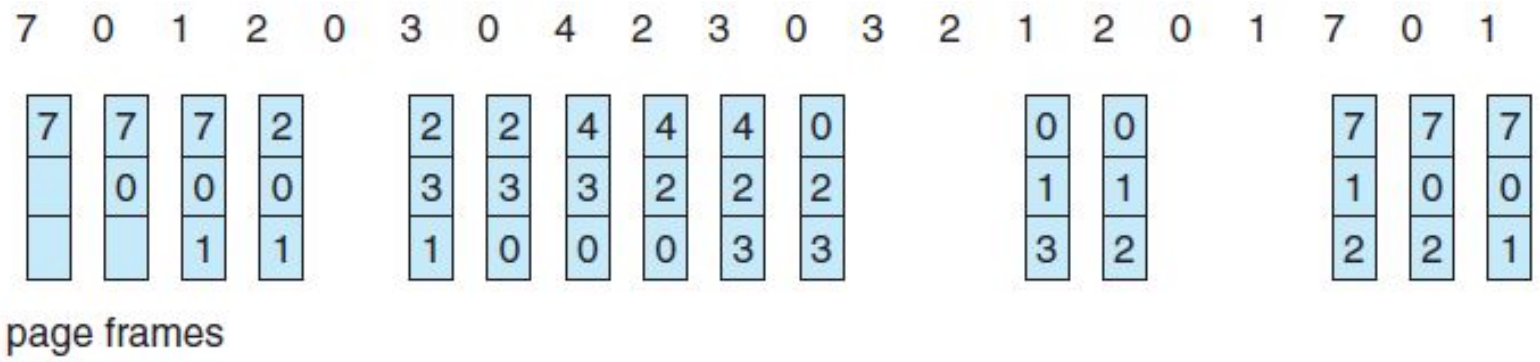
[Q]

A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

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

[Q] A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

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



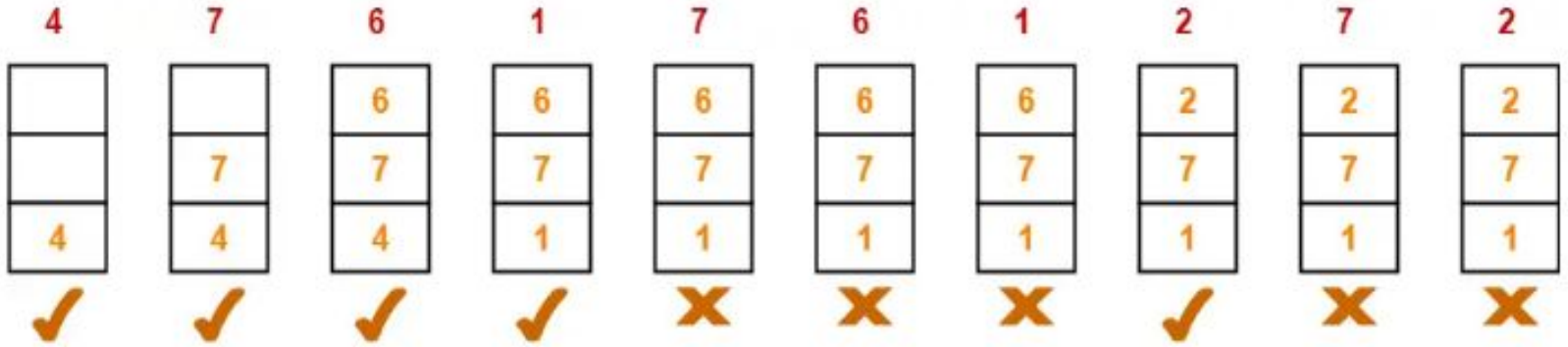
Total number of page faults occurred = 15

Optimal Page Replacement

- ★ This algorithm **replaces the page that will not be referred by the CPU in future for the longest time.**
- ★ It is practically impossible to implement this algorithm.
- ★ This is because the pages that will not be used in future for the longest time can not be predicted.
- ★ However, it is the best known algorithm and gives the **least number of page faults.**
- ★ Hence, it is used as a performance measure criterion for other algorithms.

[Q] A system uses 3 page frames for storing process pages in main memory. It uses the **Optimal page replacement policy.** Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below--

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2



Total number of page faults occurred = 5

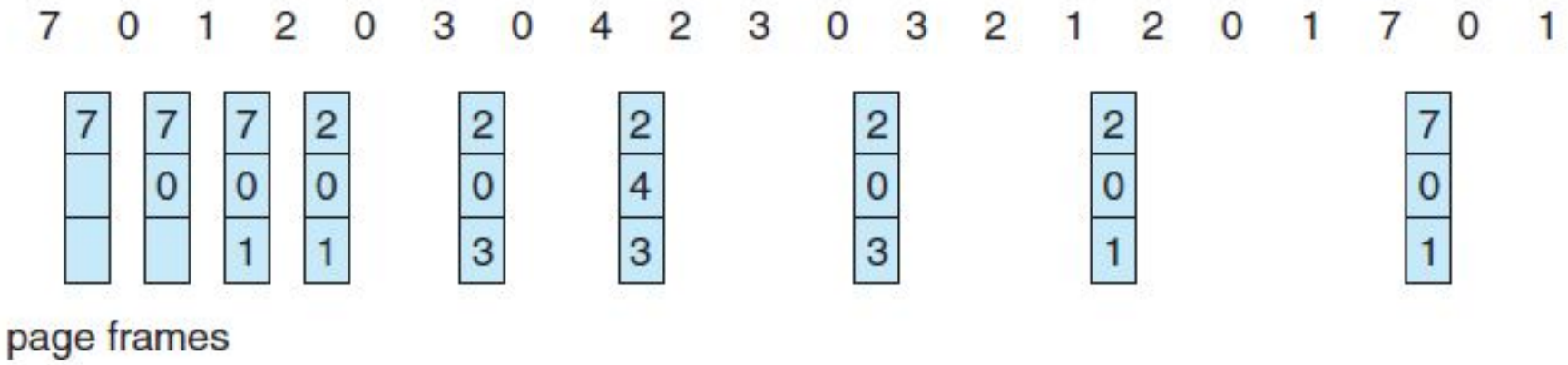
[Q]

A system uses 3 page frames for storing process pages in main memory. It uses the Optimal page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below--

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

[Q] A system uses 3 page frames for storing process pages in main memory. It uses the **Optimal page replacement policy.** Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below--

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



Total number of page faults occurred = 9

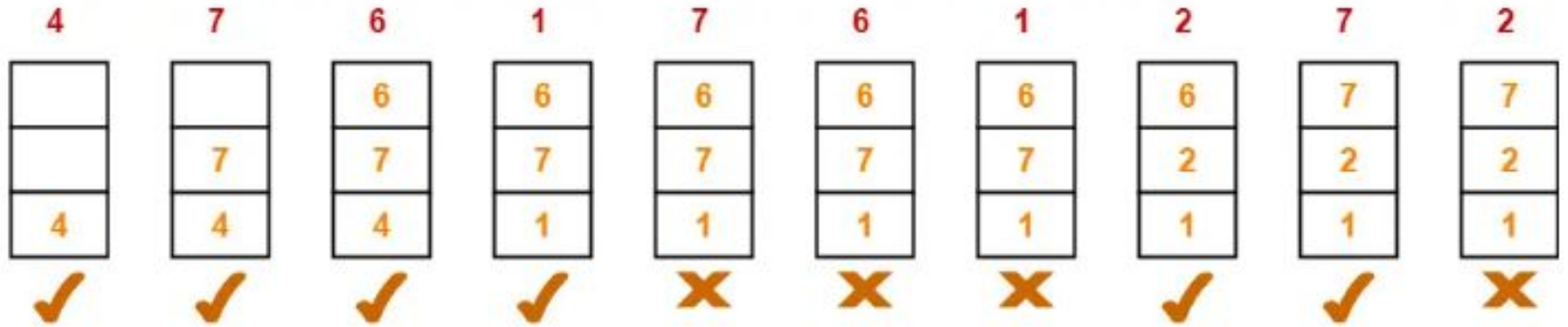
LRU(Least Recently Used) Page Replacement

★ As the name suggests, this algorithm works on the principle of **"Least Recently Used".**

★ It replaces the page that has not been referred by the CPU for the longest time.

[Q] A system uses 3 page frames for storing process pages in main memory. It uses the **Least Recently Used (LRU) page replacement policy**. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2



Total number of page faults occurred = 6

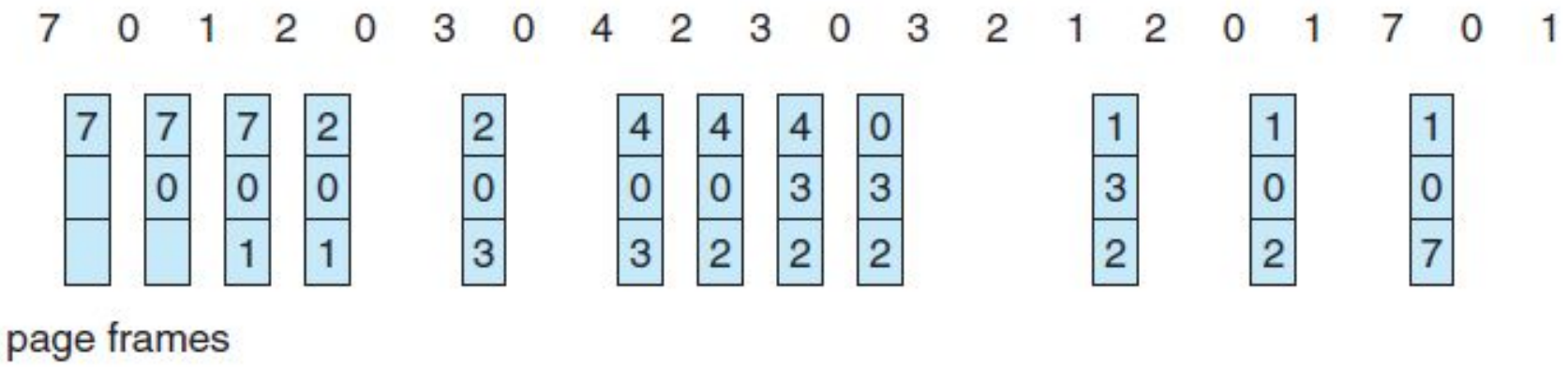
[Q]

A system uses 3 page frames for storing process pages in main memory. It uses the LRU replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below--

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

[Q] **A system uses 3 page frames for storing process pages in main memory. It uses the LRU replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below--**

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



Total number of page faults occurred = 12

Thrashing

- **Thrashing :** The processor spends most of its time swapping process (process page faults) rather than executing user instructions.
- Because of thrashing, the CPU utilization is going to be reduced or negligible.

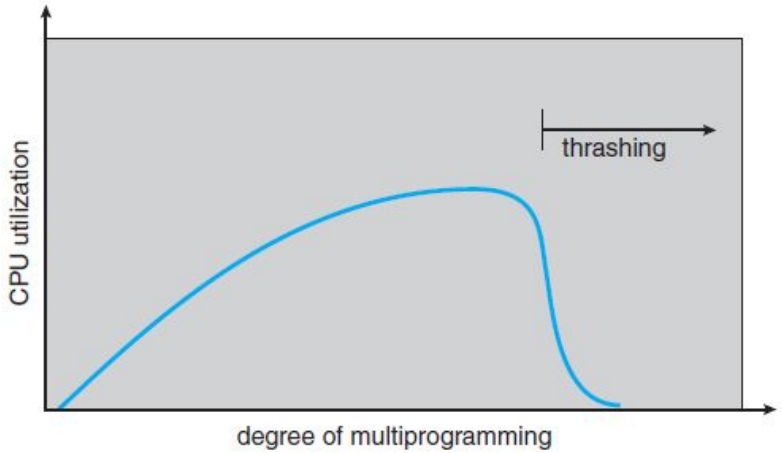


Figure 9.18 Thrashing.

Locality Model:

- ➔ A locality is a set of pages that are actively used together.
- ➔ The locality model states that as a process executes, it moves from one locality to another.
- ➔ A program is generally composed of several different localities which may overlap.
- ➔ For example, when a function is called, it defines a new locality where memory references are made to the instructions of the function call, it's local and global variables, etc. Similarly, when the function is exited, the process leaves this locality.

Techniques to handle:

1. Working Set Model

2. Page Fault Frequency

1. Working Set Model

- The working-set model is based on the assumption of locality.
- This model uses a parameter, Δ, to define the working-set window.
- The idea is to examine the most recent Δ page references.
- The set of pages in the most recent Δ page references is the working set (Figure 9.20).
- If a page is in active use, it will be in the working set.
- If it is no longer being used, it will drop from the working set Δ time units after its last reference.
- Thus, the working set is an approximation of the program's locality.

- For example, given the sequence of memory references shown in Figure 9.20, if = 10 memory references, then the working set at time t1 is {1, 2, 5,6, 7}. By time 2, the working set has changed to {3, 4}.

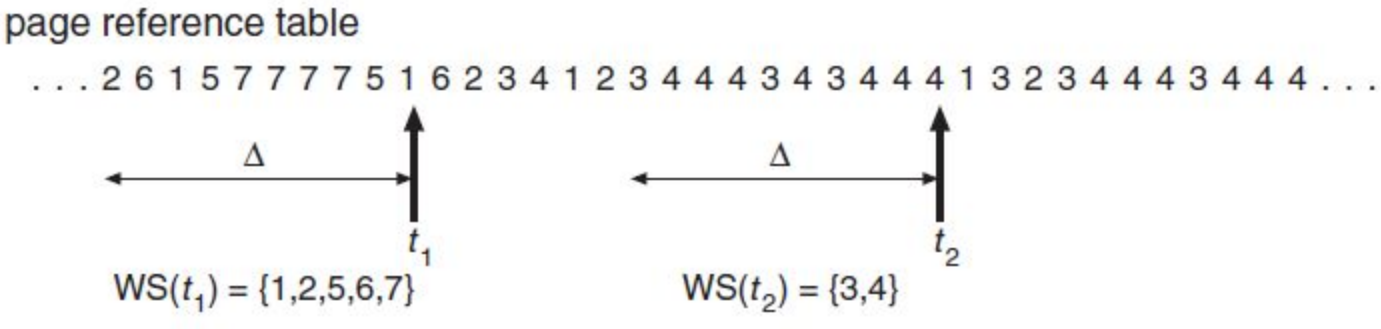


Figure 9.20 Working-set model.

- The most important property of the working set, then, is its size. If we compute the working-set size, WSSi, for each process in the system, we can then consider that

$$D = \sum WSSi$$, where D is the total demand for frames

- Each process is actively using the pages in its working set. Thus, process i needs WSSi frames.
- Now, if 'm' is the number of frames available in the memory, there are 2 possibilities:
  - *If the **total demand is greater than the total number of available frames (D> m)**, thrashing will occur, because some processes will not have enough frames.*
  - **D<=m,** then there would be no thrashing.

2. Page Fault Frequency :

A more direct approach to handling thrashing is the one that uses the Page-Fault Frequency concept.

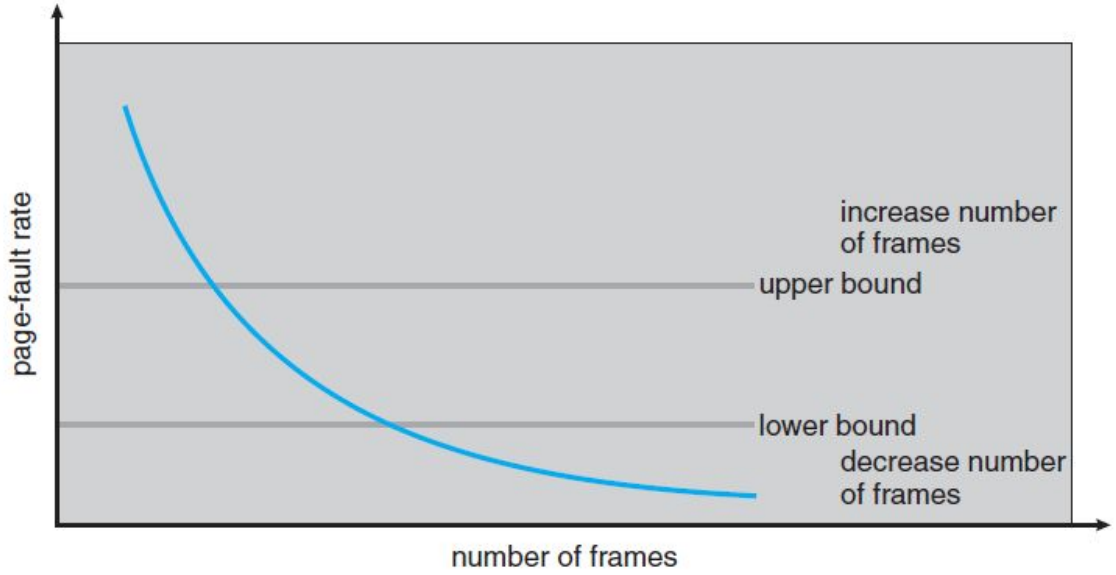


Figure 9.21 Page-fault frequency.