Chapter 9: Virtual Memory

Narzu Tarannum

CSE, BU

Background

- 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.
- Virtual memory is a **concept** that we use when we have processes that exceed the main memory.
- When computer runs out of physical memory, it writes its requirement to the hard disc in a swap file as “virtual memory”.

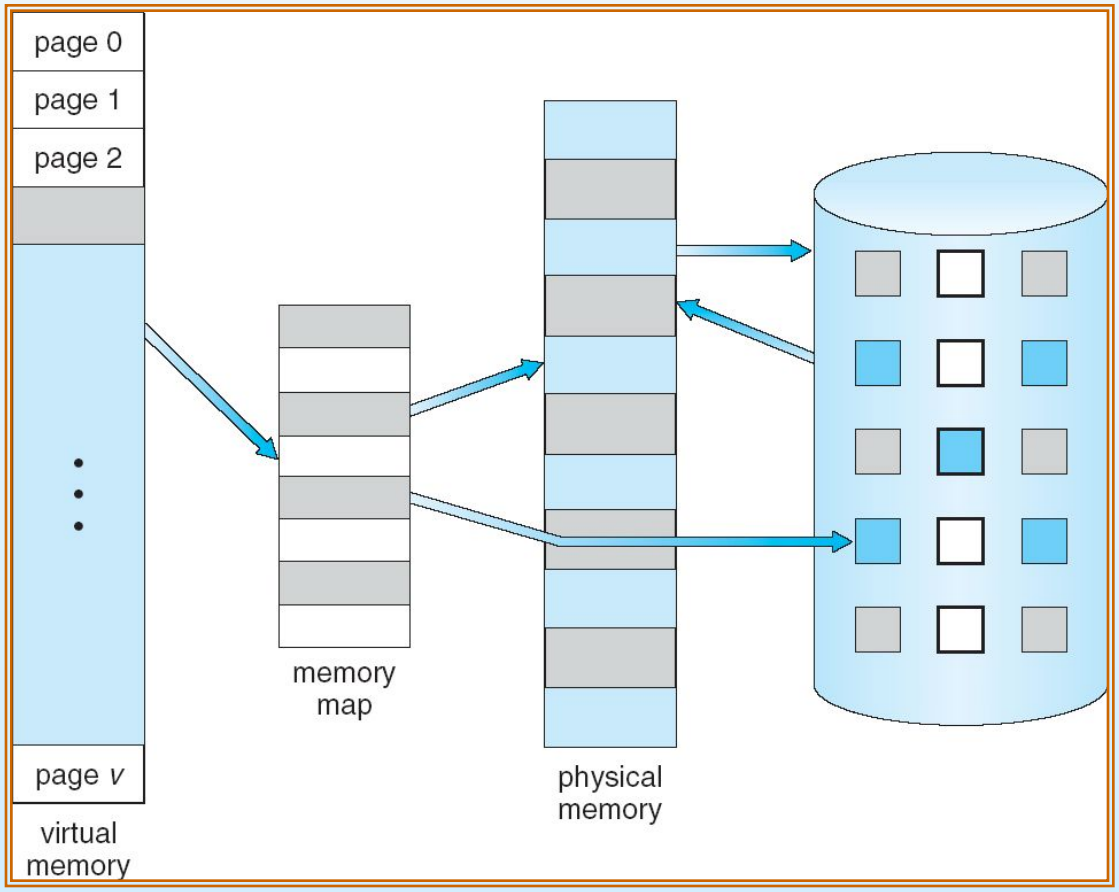
Intro. of virtual memory

- **Virtual memory** separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
- An examination of real programs show that: 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 is a storage scheme where secondary memory can be treated as though it is a part of main memory and large process can be stored in it. Only the part of the process that is actually needed for execution will be loaded on to the actual main memory.**

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

Virtual Memory That is Larger Than Physical Memory

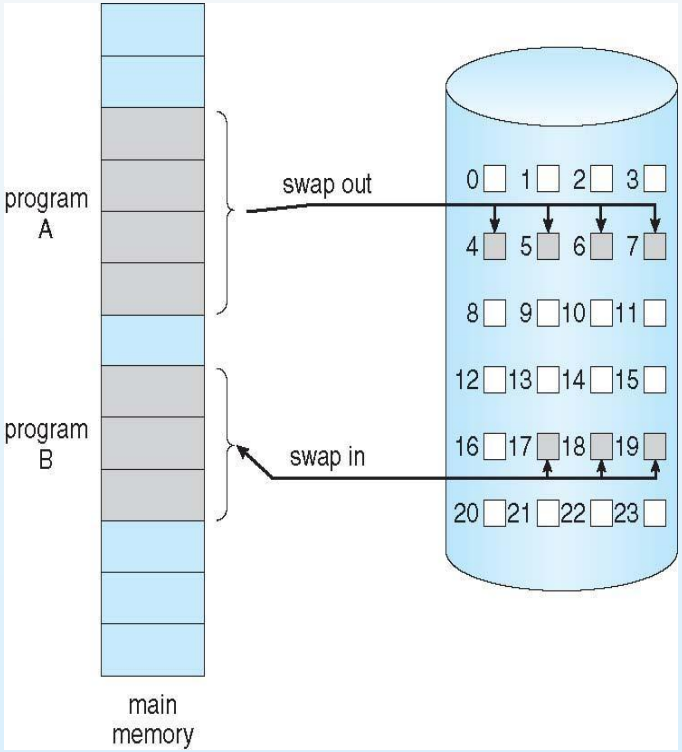


Demand Paging

- Bring pages into memory only when it is needed or pages are only loaded when they are demanded during program execution. Pages are never accessed are thus never loaded into physical memory.-------This is called Demand Paging.
- A demand paging system similar to a paging system with swapping where processes reside in secondary memory.
- In pure swapping/ pure demand paging, where entire program is swapped from secondary storage to main memory during the process startup.
- Commonly, to achieve this process a page table implementation is used. The page table maps logical memory to physical memory.
- Demand paging follows that pages should only be brought into memory if the executing process demands them. This is often referred to as lazy swapper as only those pages demanded by the process are swapped from secondary storage to main memory.
- Rather than swapping the entire process into memory, since we are now viewing a process as a sequence of pages, rather than as one large contagious address space, use the term SWAPPER is technically incorrect here- so in concern with demand paging we will use the term **pager.**

Demand Paging

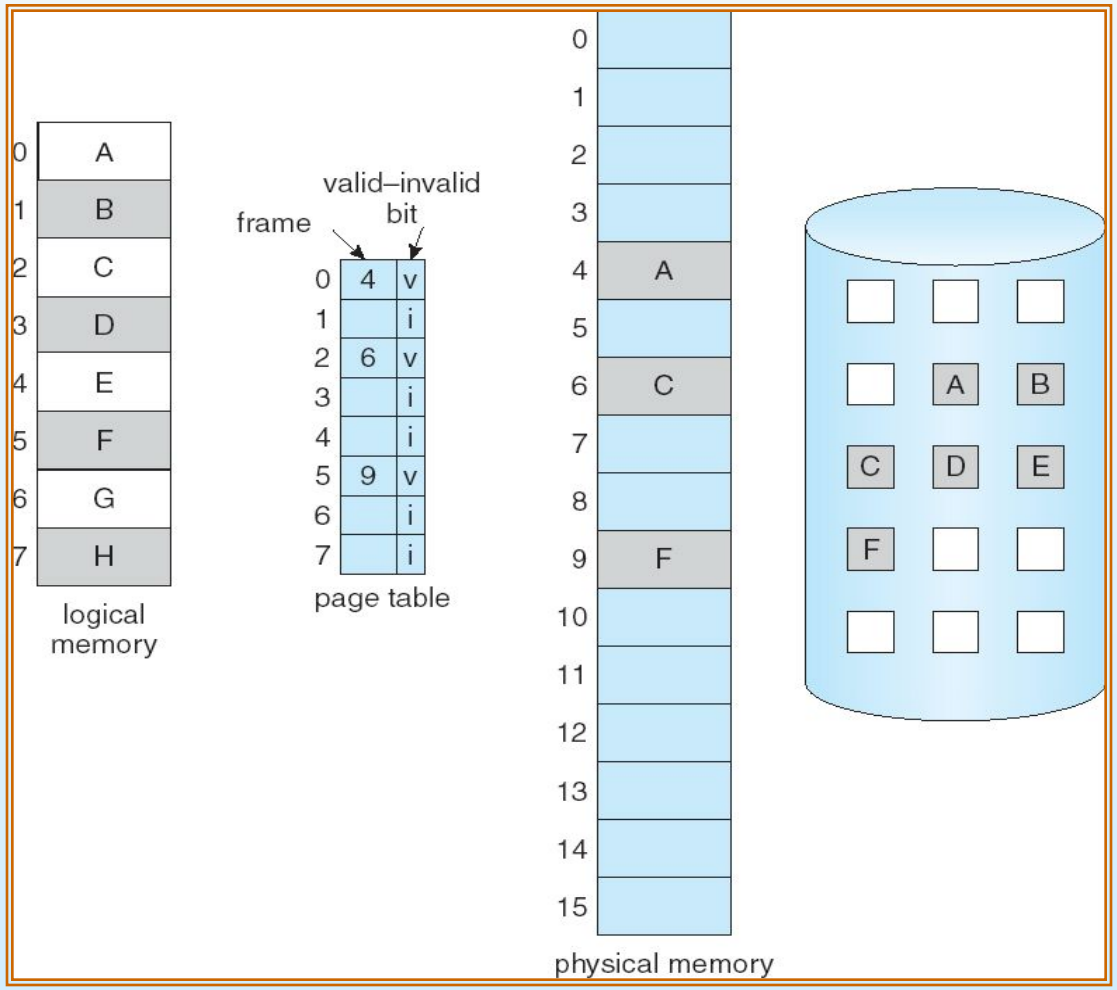
- Advantages
  - Less memory needed
  - Faster response
  - More users/ Degree of multiprogramming increase.
  - Less I/O needed
  - Reduce memory requirement
  - Swap time is also reduce
- Disadvantages
  - Page fault interrupt
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory



Valid Invalid bit

- The page table uses a bitwise operator to mark if a page is valid or invalid. A valid page is one that currently resides in main memory. An invalid page is one that currently resides in secondary memory. When a process tries to access a page, the following steps are generally followed:
  - Attempt to access page.
  - If page is valid (in memory) then continue processing instruction as normal.
  - If page is invalid then a **page-fault trap / page-fault interrupt** occurs.
    - Page is needed (is valid) ⇒ reference to it
    - invalid reference ⇒ abort
    - not-in-memory ⇒ bring to memory
  - Restart the instruction that was interrupted by the operating system trap.
- With each page table entry a valid–invalid bit is associated (1/ v ⇒ in-memory, 0 / i ⇒ not-in-memory)

Page Table When Some Pages Are Not in Main Memory

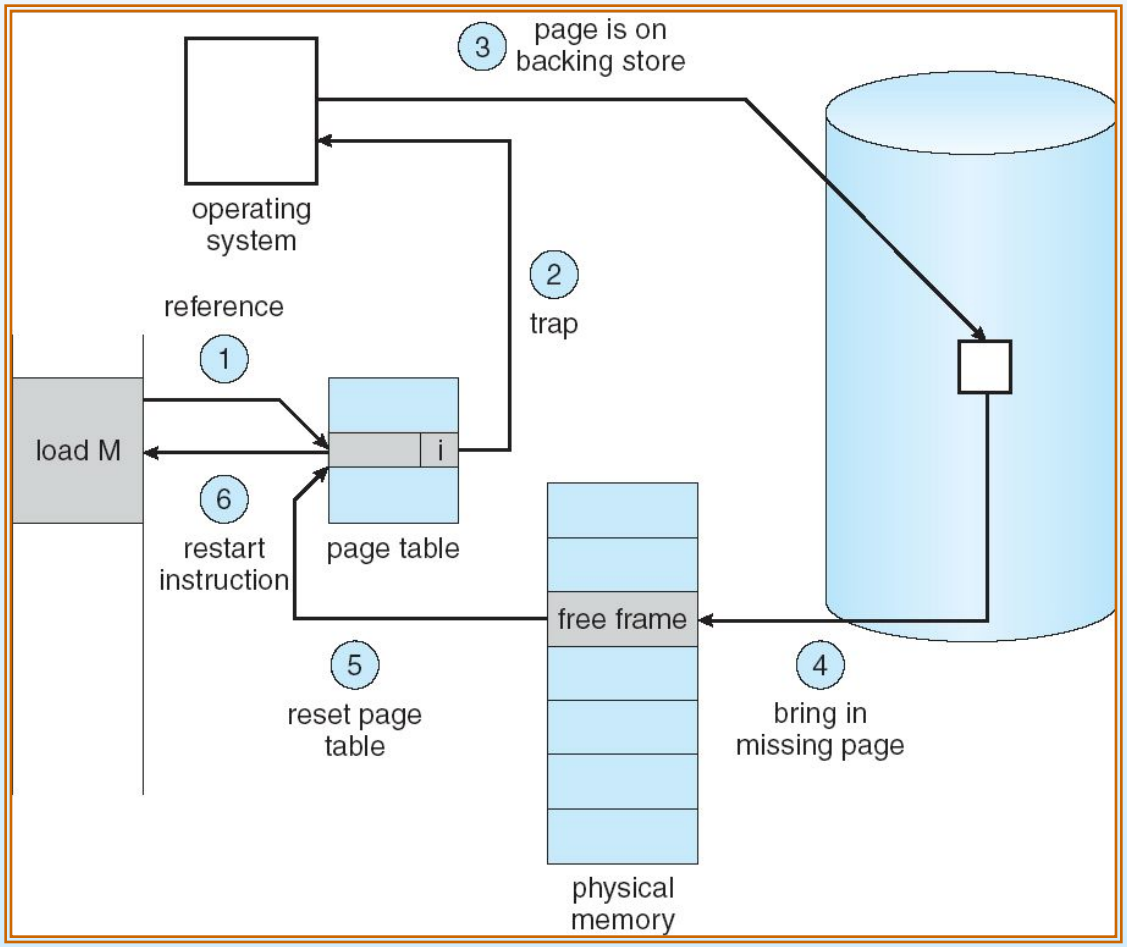


Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ **page fault**

1. OS looks at another table to decide:
   - Invalid reference ⇒ abort.
   - Just not in memory.
2. Find empty/ free frame.
3. Load page from disk into frame.
4. Reset tables, validation bit = 1.
5. Restart instruction that caused page fault

Steps in Handling a Page Fault



Problems of demand paging

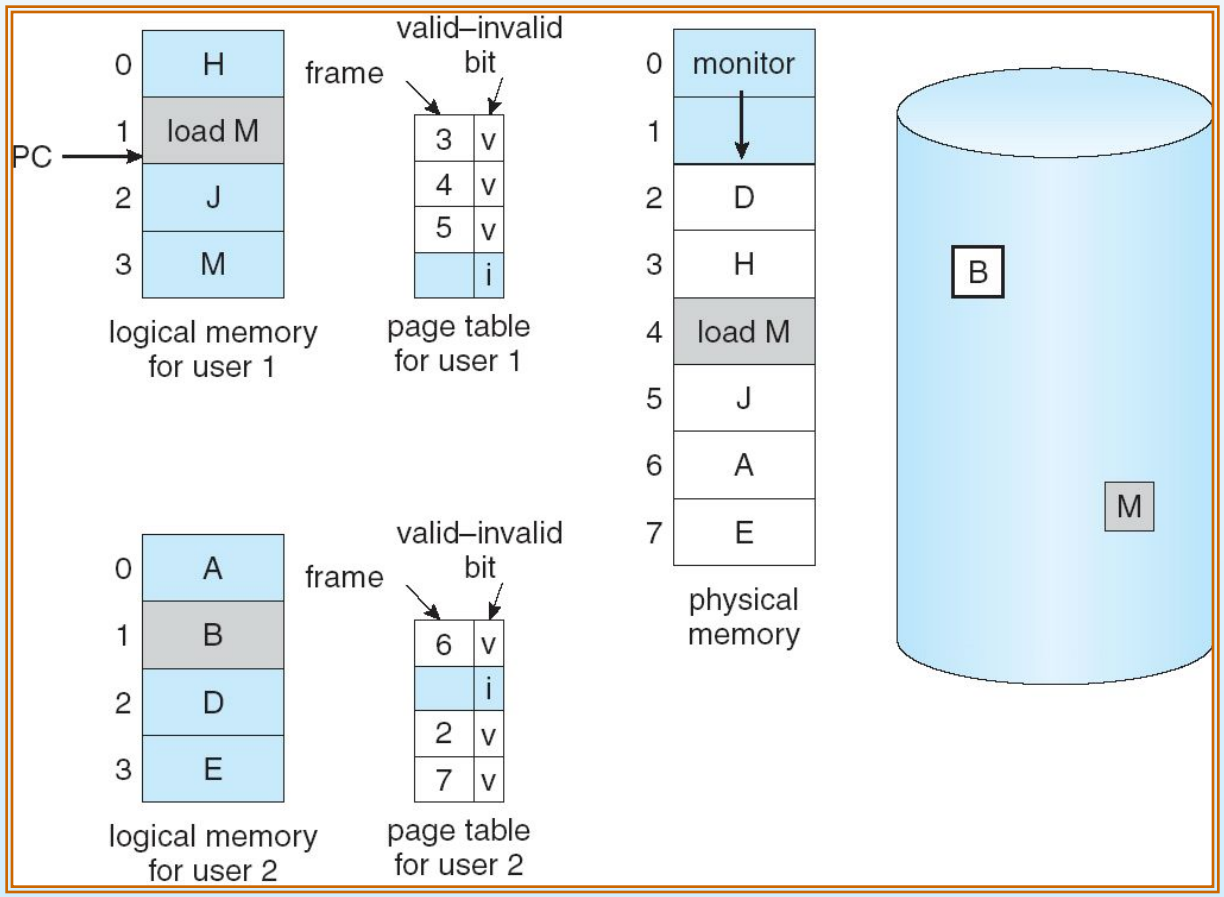
With the help of demand paging, we are increasing our degree of multi-programming by loading only the required pages into memory hence facilitating the possibility of having more processes loaded into memory--- this could lead to over allocation of memory.

Eg.

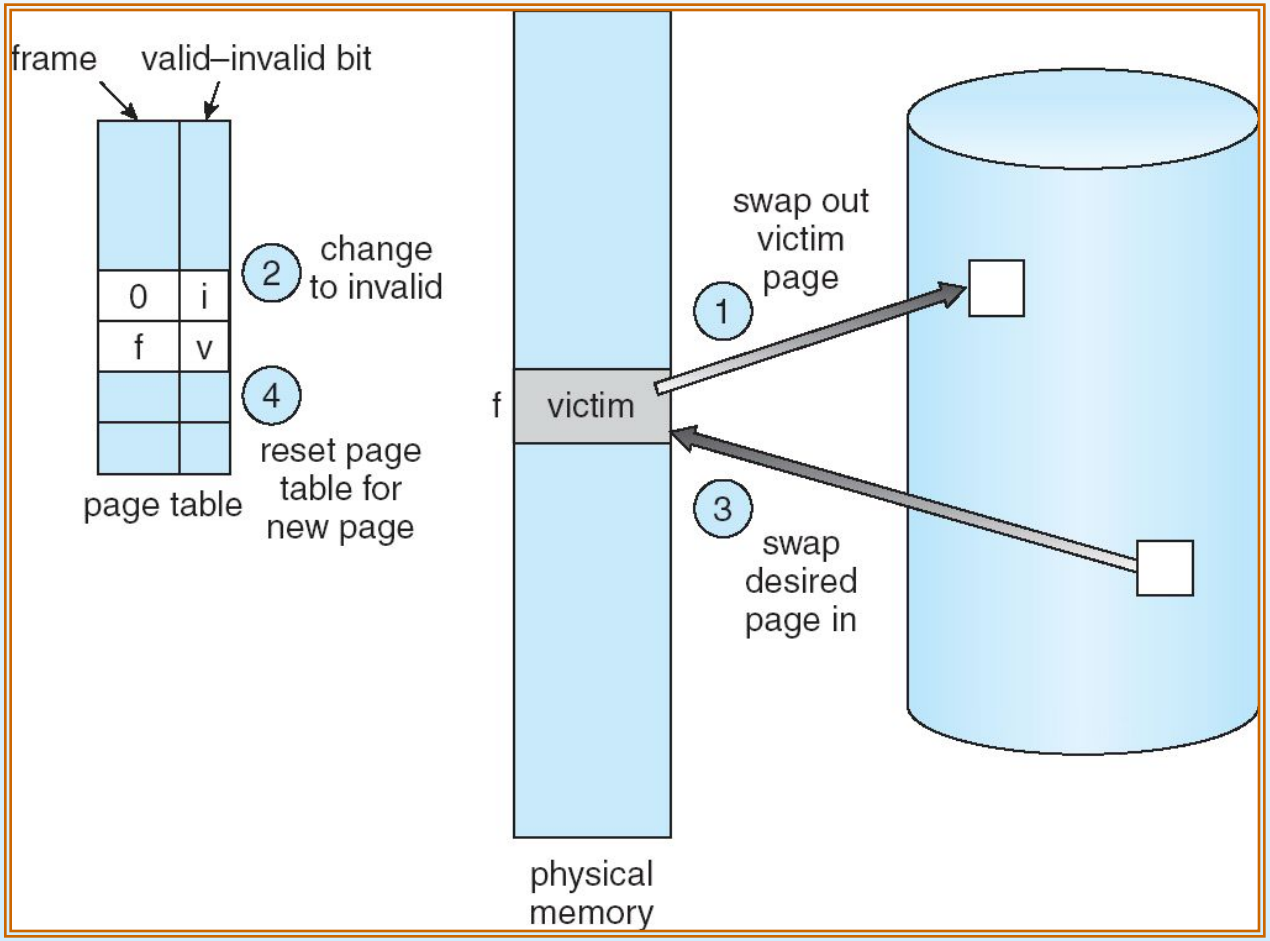
- Suppose we have 40 frames in memory.
- We have 6 processes each of which has 10 pages but uses only 5 at the moment.
- We can load this (6 X 5) = 30 pages into the memory
- So all the processes are executing simultaneously.
- Still we have 10 frames free.

Now at some point let's say all the 6 processes want to use all their 10 pages. So we need to load the remaining (6 X 5) = 30 pages into the memory, while we have only 10 free frames.

Need For Page Replacement



Page Replacement



What happens if there is no free frame?

- **Page replacement** – is the technique of swapping out pages from physical memory when there are no more free frames available, in order to make room for other pages which has to be loaded to the physical memory.
- If a process wants to use/access a page which is not present in physical memory, it will cause a page fault.—So, that page has to be now loaded into memory.
- But if there are no free frames available to load that page, what should we do than?
  - We look for an occupied frame that is not being used currently.
  - We free that frame by writing it's contents to the swap space (disk)
  - We update the page tables to indicate that the page is now no more in memory.
  - We load the page of the process that caused the page fault to occur into the freed frame

Basic Page Replacement

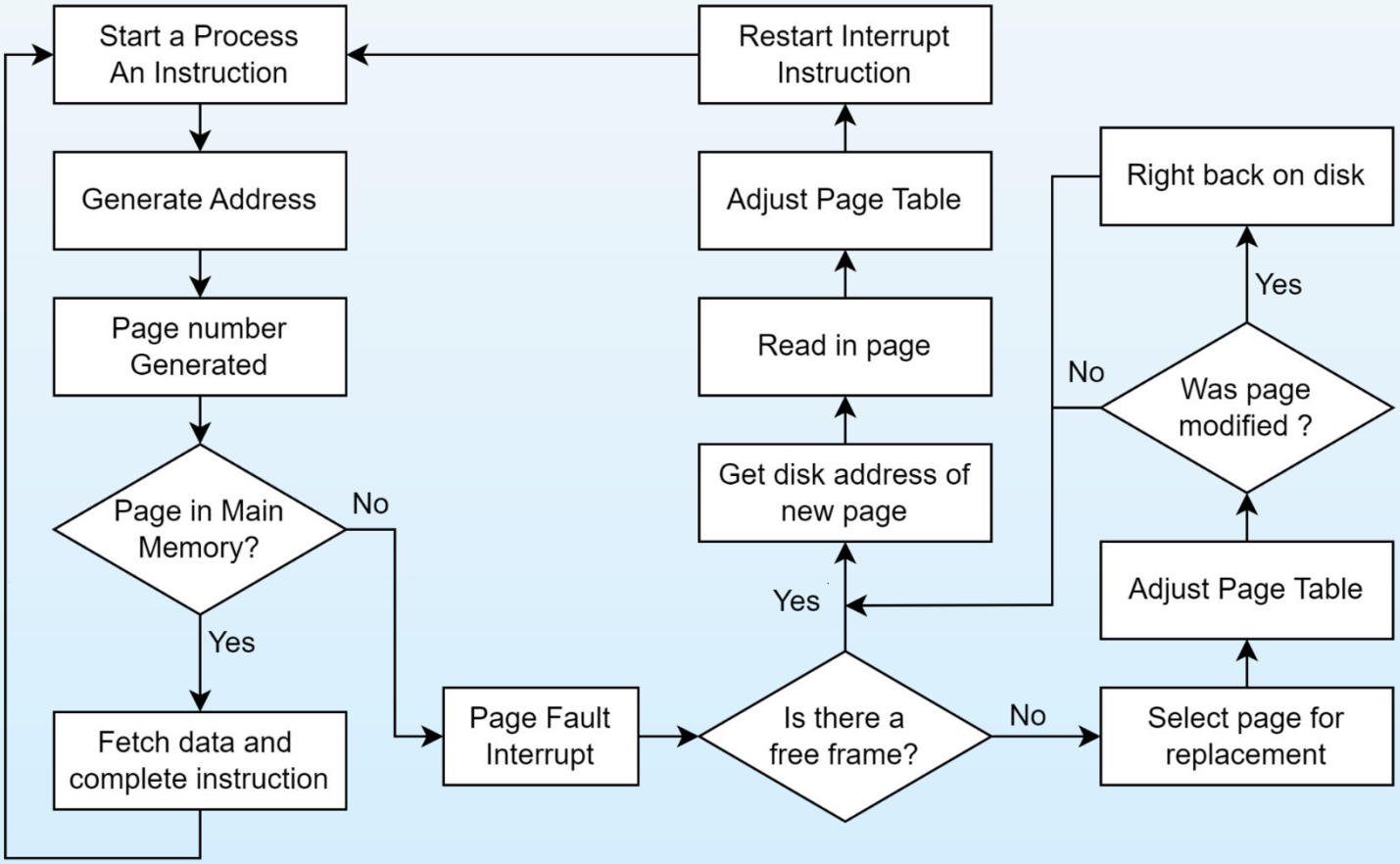
1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame

3. Bring the desired page into the (newly) free frame; Update the page and frame tables.

4. Continue the process by restarting the instruction that caused the trap

Flowchart of Demand Paging and page replacement



Page Replacement

- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

- Solve two problems in demand paging implementation:
- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access

Page Replacement Algorithms

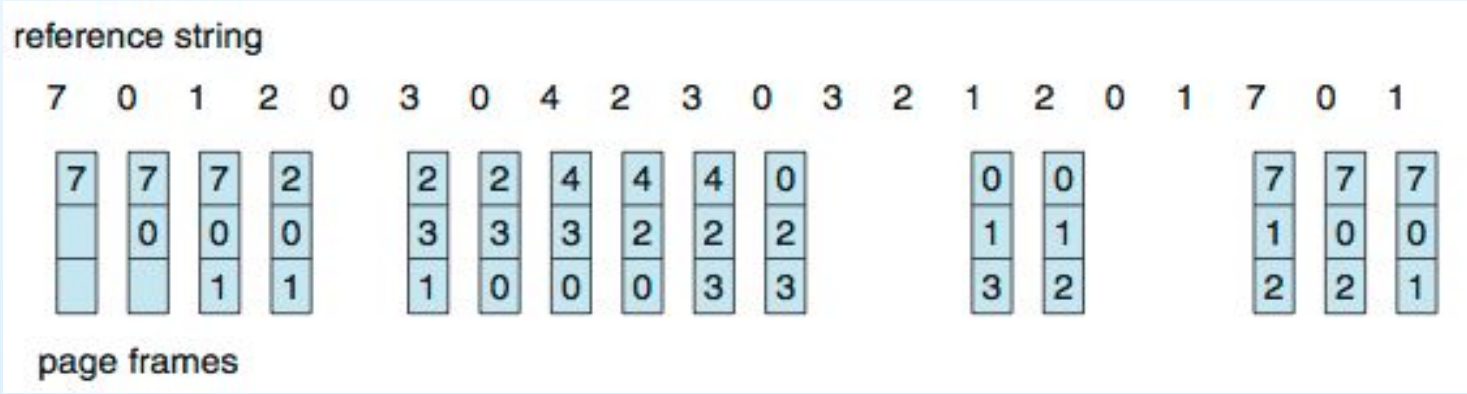
- Page Replacement Algorithms:
  - FIFO (First In First Out)
  - LRU (Least Recently Used)
  - OPT (Optimal)

- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

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

First-In-First-Out (FIFO) Algorithm

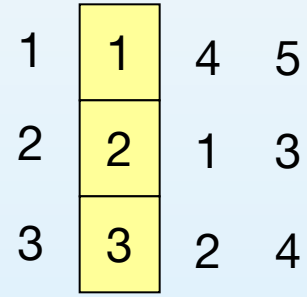
- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



15 page faults

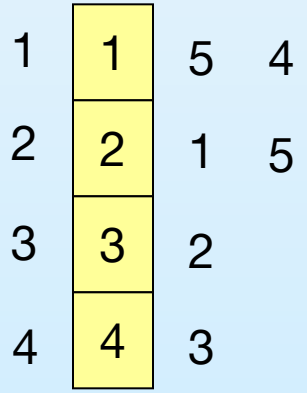
First-In-First-Out (FIFO) Algorithm

- The page which brought first will be replace first
- Reference string: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- 3 frames (3 pages can be in memory at a time per process)



9 page faults

- 4 frames

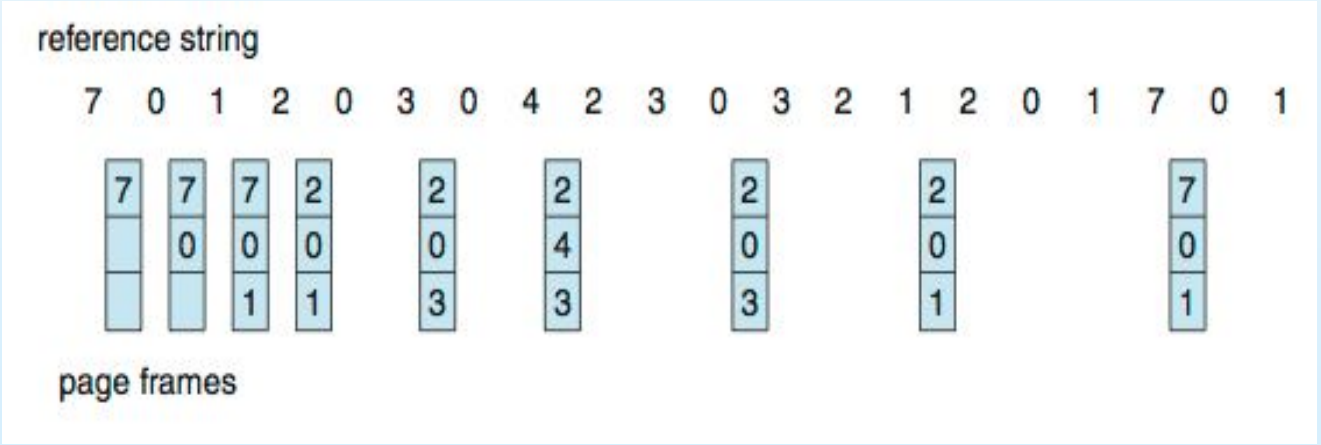


10 page faults

- FIFO Replacement – Belady's Anomaly

Optimal Algorithm

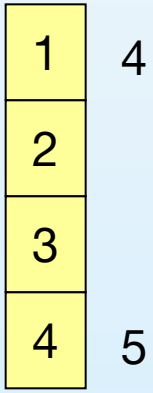
- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs



Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

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

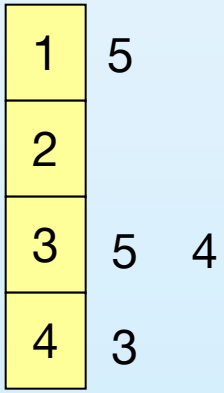


6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

Least Recently Used (LRU) Algorithm

- LRU replaces page that has not been used for the longest time
- Use the recent past to predict the future
- Reference string: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**



8 page faults

End of Chapter 9