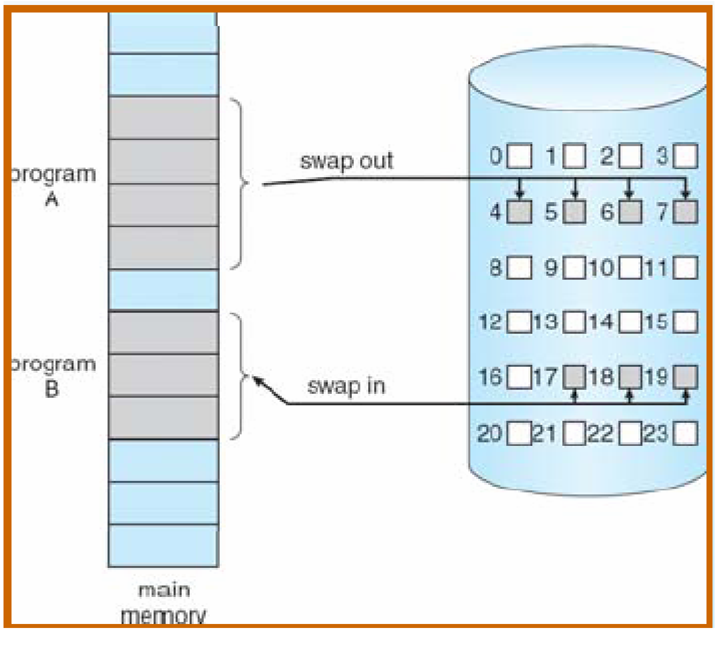
الملزمة الثامنة
Virtual Memory

- ***Virtual memory*** is a technique that allows the execution of processes that may not be completely in memory.
- **Benefits**:
  1. Program size not restricted to memory size.
  2. More programs could be run at the same time increasing CPU utilization and throughput.
  3. Less I/O needed to load programs, so user program run faster.

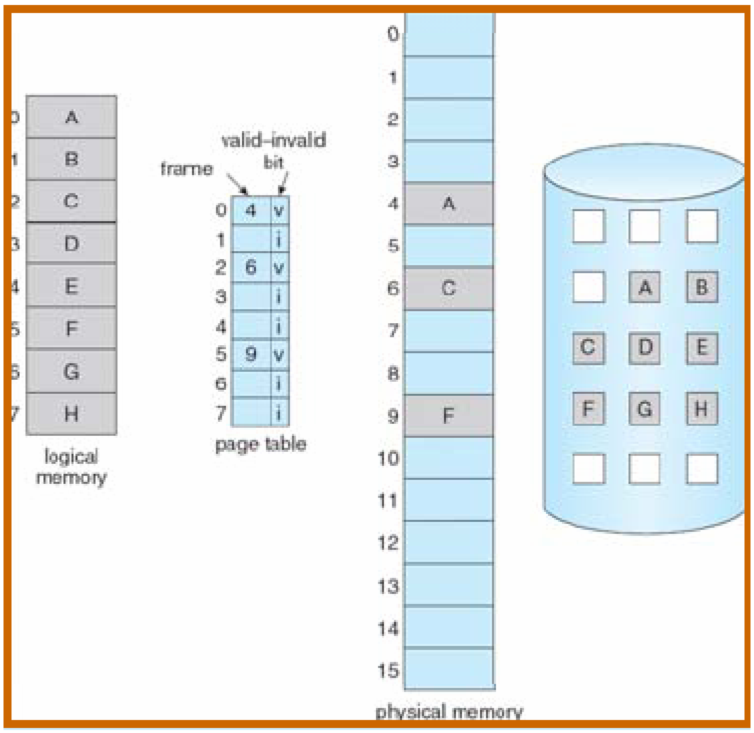
Demand Paging

A demand paging system is similar to paging system with swapping. Processes reside on secondary memory (Disk). When we want to execute a process, we swap it into memory.

- We use **a Lazy Swapper**.
- lazy swapper never swaps a page into memory unless that page will be needed.



- To distinguish between those pages that are in memory and those pages that are on the disk, we use ***valid-invalid bit***.
    - When this bit is set to valid, page in memory.
    - When this bit is set to invalid, page in disk.



Page table when some pages are not in main memory

- What happens if the process tries to use a page that was not brought into memory? Access to a page marked invalid causes ***page-fault trap***.
- O.S do the following procedure for handling this page-fault:-

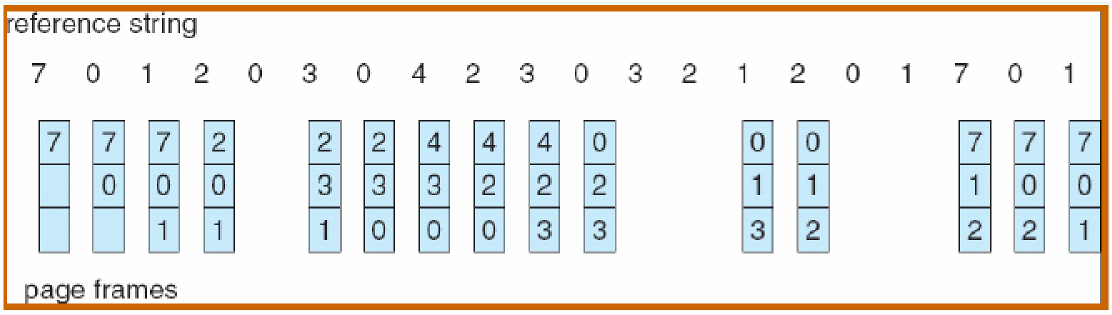
1. Terminate the process.
2. Find free frame (in M.M).
3. Read the desirable page into the newly allocated frame.
4. Modify page table (i→v).
5. Restart the process from the instruction that was caused the page-fault.

Page Replacement

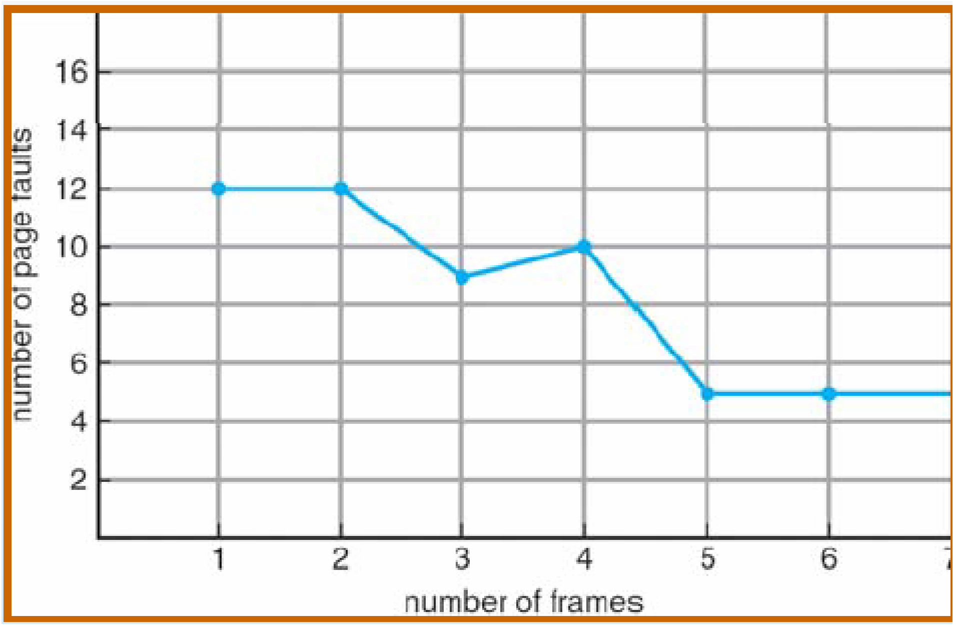
- If there is no free frame, then O.S :
    1. Use a ***page replacement algorithm*** to select a ***victim frame***.
    2. Write the victim page to the disk; change the page table accordingly.
    3. Read the desired page into the newly free frame; change the page table.

First-In-First-Out (FIFO) Algorithm

- The simplest algorithm.
- We use a FIFO queue and replace the page at the head of the queue, and insert these new page in the tail of the queue.
- Easy to program.
- Its performance is not always good.
- Example: Let's have the following reference string
  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1, and three frames only.



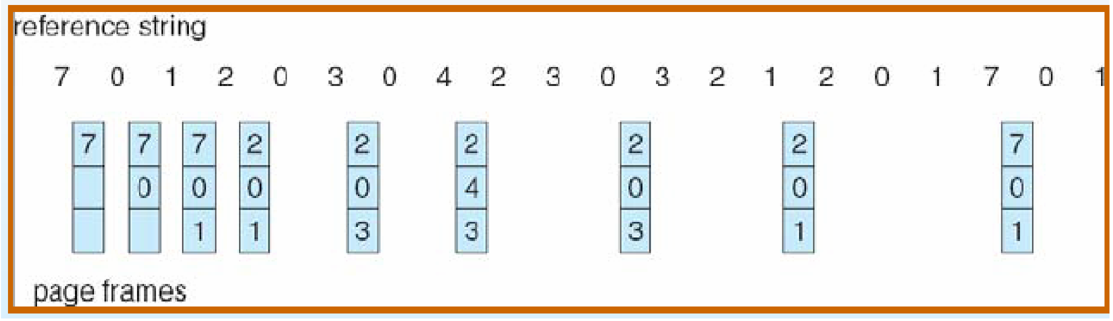
- There are 15 page-fault, which will slows process execution.
- FIFO algorithm suffer from (**Belady's anomaly**): The page-fault rate may increase as the number of allocation frame increases.
- **H.W**: Implement FIFO algorithm on the following string :
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, using (3, 4- frames).



FIFO Belady's anomaly

Optimal Algorithm

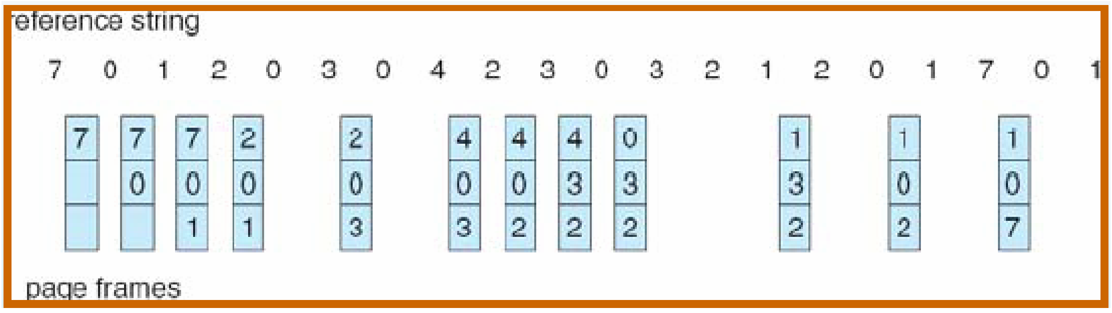
- Has the lowest page-fault rate of all algorithms.
- Will never suffer from Belady's anomaly.
- Its idea *"Replace the page that <u>will</u> not be used for the longest period of time".*
- Difficult to implement, and used for comparison studies only.
- Example:



- The page-fault is **9 only**.

Least Recently Used (LRU) Algorithm

- *"replace the page that has not been used for the longest period of time".*
- *Example:*



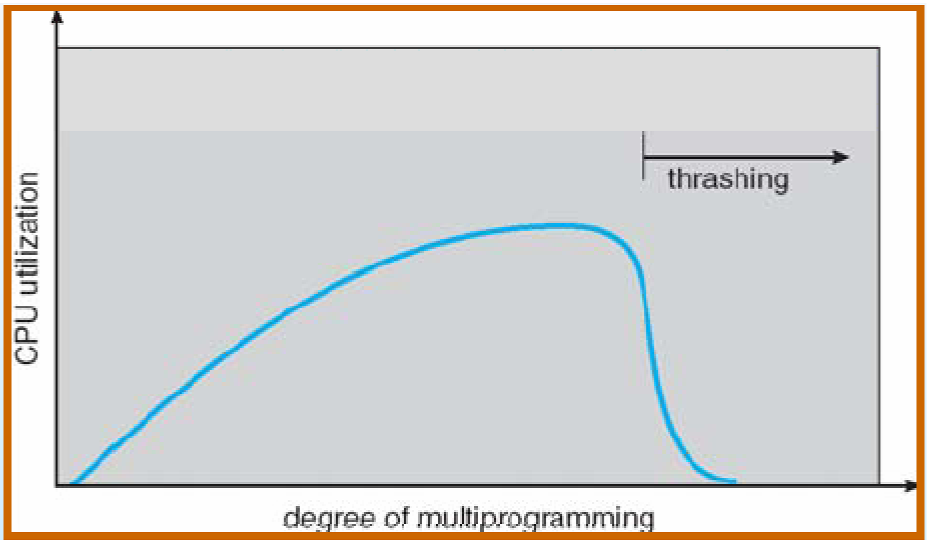
- The page-fault is **<u>12</u>**.

- This algorithm implemented by:
  1. <u>counter</u>: we associate with each page-table entry a time-of-use field.
  2. <u>stack</u>: whenever a page is referenced, it is removed from the stack and put on the top. So the top of the stack is the most recently used page and the bottom is the RLU page.
- RLU never suffer from Belady's anomaly.

Thrashing

It is a high paging activity. The processor is ***thrashing*** if it is spending more time paging than executing.

- If CPU utilization is too low, then degree of multiprogramming is increased by introducing a new process to the system. As the degree of multiprogramming increased, CPU utilization also increased, but more slowly until a maximum is reached. After that thrashing occurred and CPU utilization drops sharply.



- To stop thrashing, we must degrease the degree of multiprogramming.
- To prevent thrashing, we must provide a process as many fames as it needs.

Program Structure & Page Fault

Int[128,128] data;

Each row is stored in one page

<u>Program 1</u>

```
   for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
       data[i,j] = 0;
```

→ 128 x 128 = 16,384 page faults

<u>Program 2</u>

```
 for (i = 0; i < 128; i++)
  for (j = 0; j < 128; j++)
     data[i,j] = 0;
```

→128 page faults