# Operating systems

**Module-4**

M 4.1 Memory Management

**M 4.2 Virtual memory**

By: M Yogi Reddy
Assistant Professor
CSE Dept.
GITAM University, Hyd

# Virtual Memory

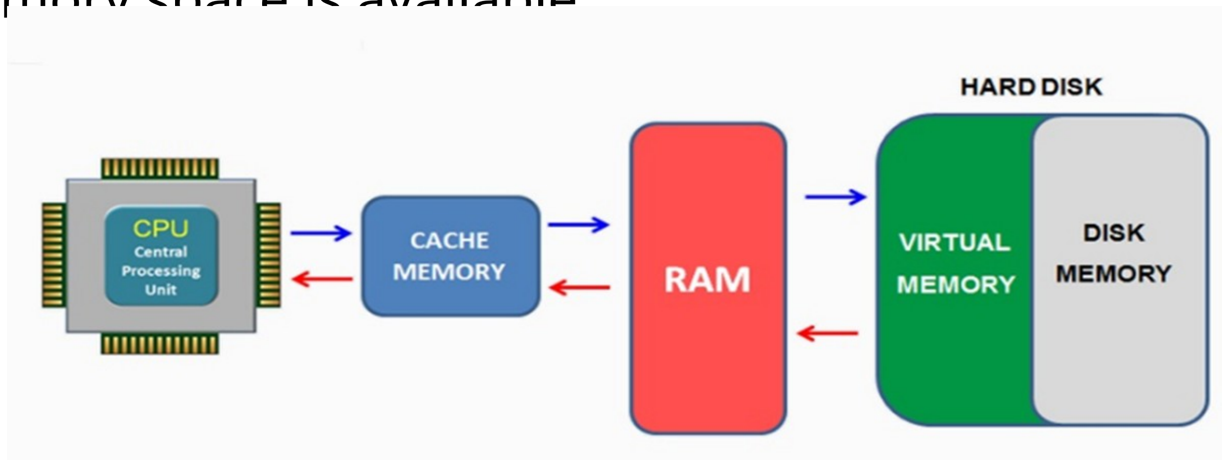
- Demand paging
- Copy-on-Write
- Page-replacement
- Allocation of frames
- Thrashing

# What is Virtual Memory

- **Virtual Memory** is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory. In Virtual memory, the user can store processes with a bigger size than the available main memory.
- Therefore, instead of loading one long process in the main memory, the OS loads the various parts of more than one process in the main memory.

**For example:**

- Let's assume that an OS requires 300 MB of memory to store all the running programs. However, there's currently only 50 MB of available physical memory stored on the RAM.
- The OS will then set up 250 MB of virtual memory and use a program called the Virtual Memory Manager(VMM) to manage that 250 MB.
- So, in this case, the VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required.
- The OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.



# Why Need Virtual Memory?

Here, are reasons for using virtual memory:

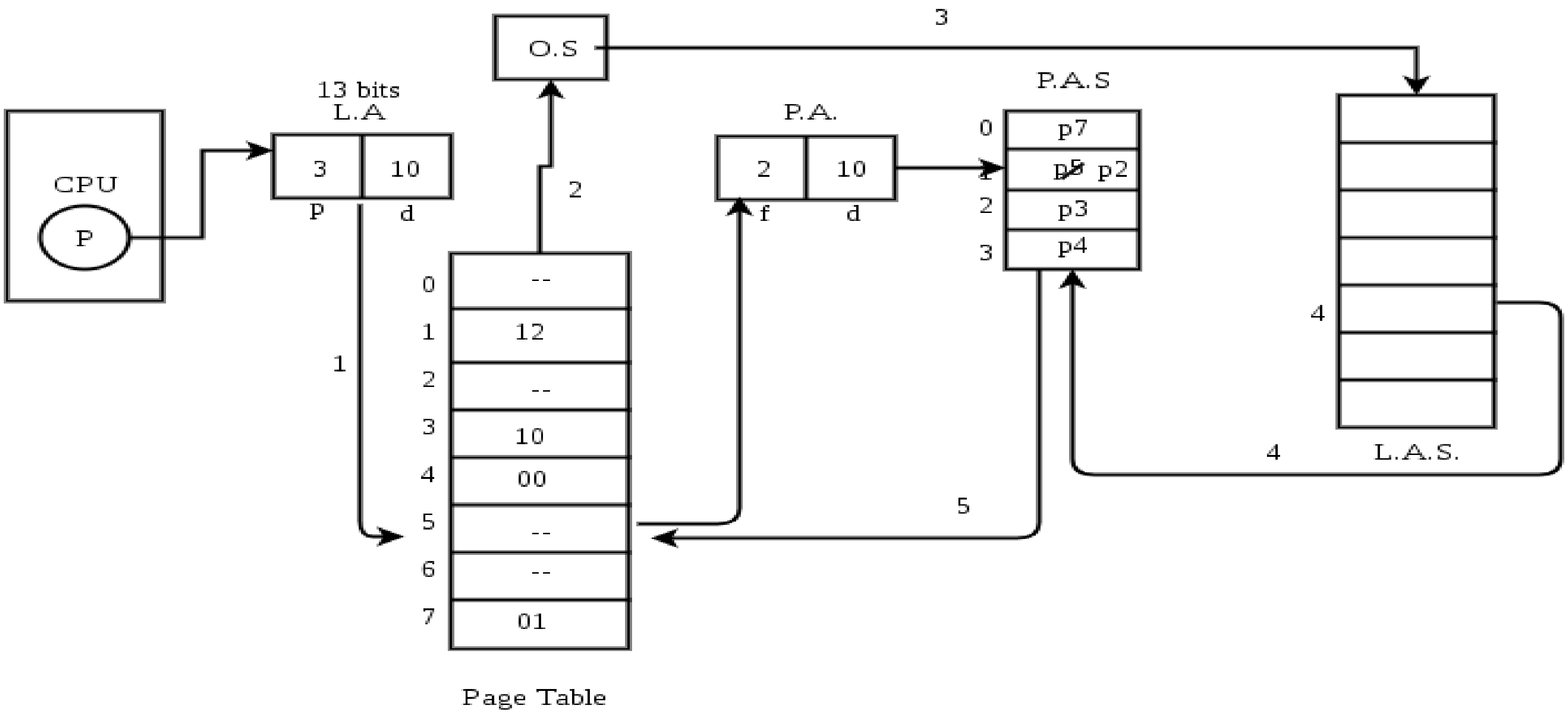
- Whenever your computer doesn't have space in the physical memory it writes what it needs to remember to the hard disk in a swap file as virtual memory.
- If a computer running Windows needs more memory/RAM, then installed in the system, it uses a small portion of the hard drive for this purpose.
- Virtual memory is mostly implemented with **demand paging** and **demand segmentation**.

# Benefits of having Virtual Memory

- Large programs can be written, as virtual space available is huge compared to physical memory.
- Less I/O required, leads to faster and easy swapping of processes.
- More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

# Demand paging

- The process of loading the page into memory on demand (whenever **page fault** occurs) is known as demand paging.
- The process includes the following steps :



- If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault.
- The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
- The OS will search for the required page in the logical address space.
- The required page will be brought from logical address space to physical address space. The **page replacement algorithms** are used for the decision making of replacing the page in physical address space.
- The page table will updated accordingly.
- The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.

Hence whenever a **page fault** occurs these steps are followed by the

# Page Fault Service Time

- Whenever a page fault occurs, the CPU takes considerable amount of time to perform the above mentioned steps. This time taken by the CPU is called as page fault service time.
  - Access time for Main memory : m
  - Service time for Page fault: s
  - Rate for Page fault Page fault rate is : p
  - **Effective memory access time** = (p*s) + (1-p)*m
- Demand Paging Example: Memory access time = 200 nanoseconds Average page-fault service time = 8 milliseconds and p is 0.8

EAT = 0.8 * (8 ms) + (1 – 0.8) * 200 ns

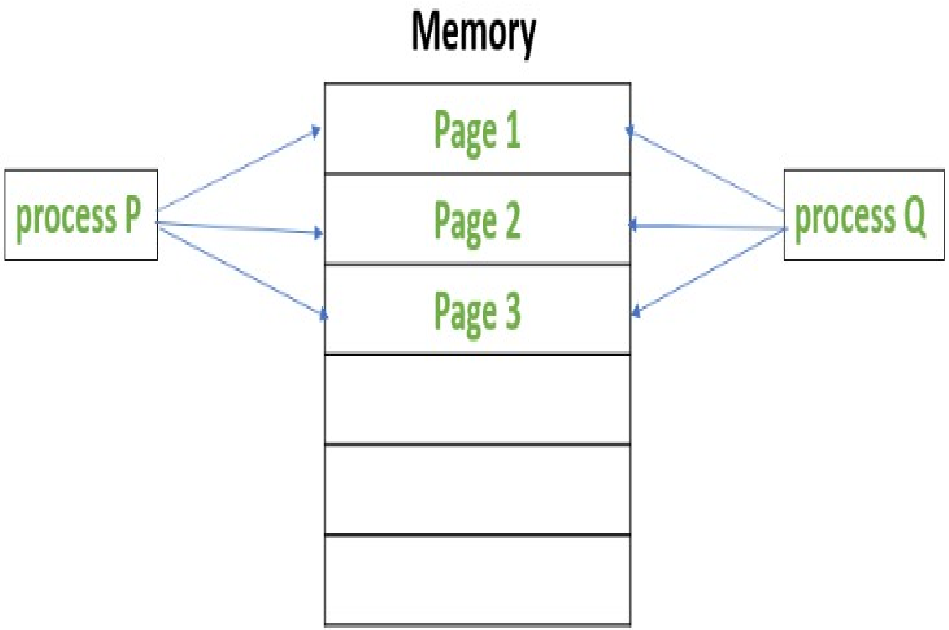
= 6.4ms + 0.2 * 0.2ms

= 6.4ms + 0.04ms

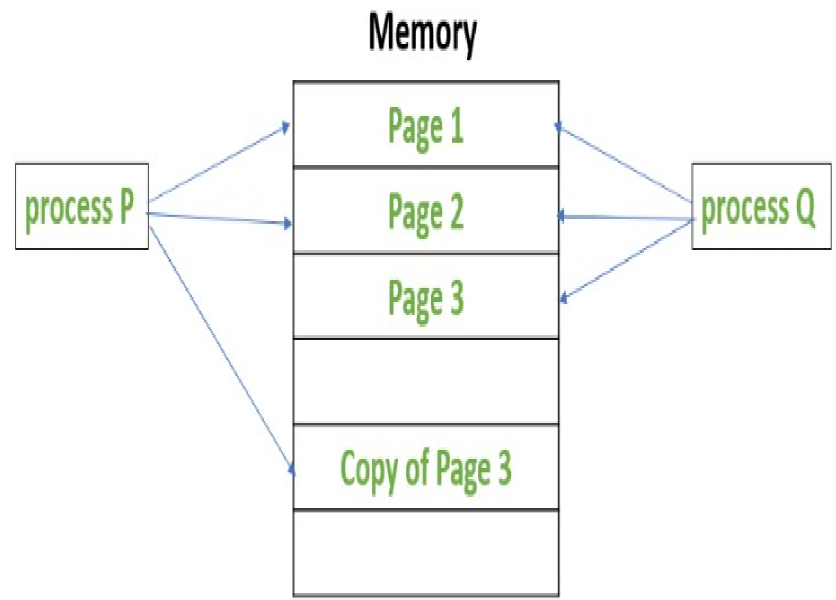
= 6.44ms

# Copy-on-Write (CoW)

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.
- Shared page is copied only if either process modifies it.
- COW allows more efficient process creation as only modified pages are copied.
- For example, in UNIX OS, fork() system call creates a duplicate process of the parent process which is called as the child process.




Before process P modifies Page 3




After process P modifies Page 3

# Page-replacement Algorithm

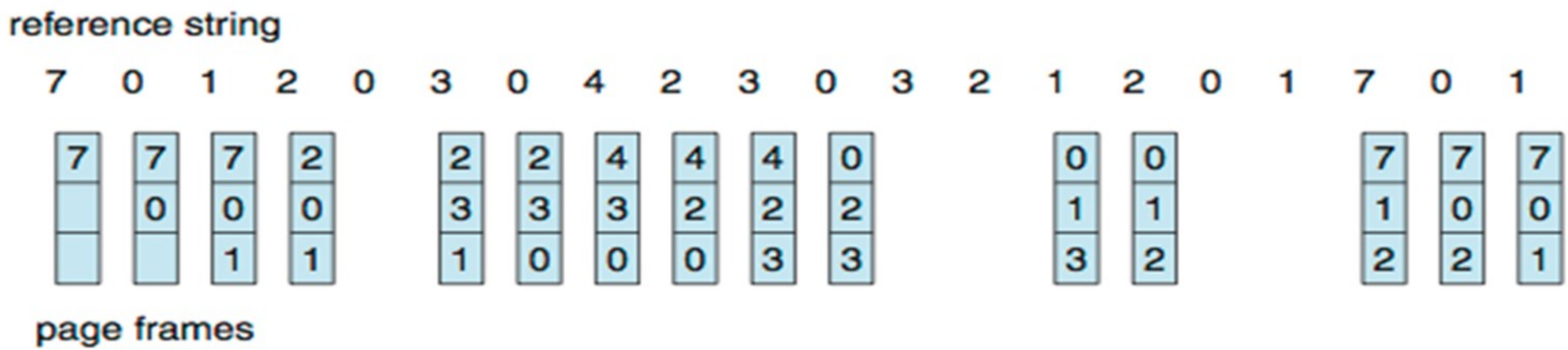
- Page Replacement algorithms in operating system are different operational logics to decide which page should be replaced when a new page comes in the system.
- Types of Page Replacement Algorithms
  - FIFO – First in First Out
  - Optimal Page Replacement Algorithm
  - LRU – Least Recently Used
  - MRU – Most Recently Used
  - **Counting Algorithms**
    - LFU – Least Frequently Used
    - MFU – Most Frequently Used

- 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**

# FIFO Algorithm

- FIFO stands for First In First Out and it 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.
- Consider page reference string: **7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1** with 3 page frames. Find number of page faults and page fault rate.




- Total page faults are 15
- Page fault rate = 15/21 = 0.71

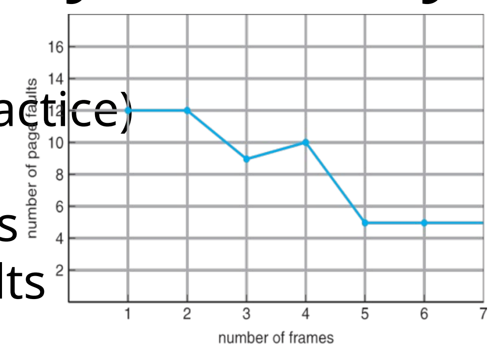
Some times by increasing the page size, page fault rather increases, this type of anomaly is called **belady's Anomaly**. But, thats not the case always.

Example consider the following(solve yourself for practice)

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

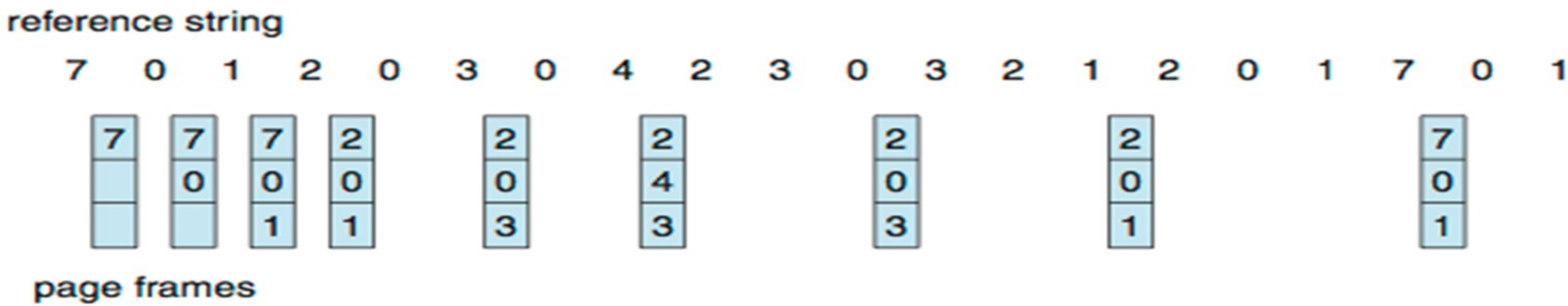
Using 3 page frames gives 9 page faults

Using 4 page frames gives 10 page faults



# Optimal Algorithm

- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
- Consider page reference string: **7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1** with 3 page frames. Find number of page faults and page fault rate.

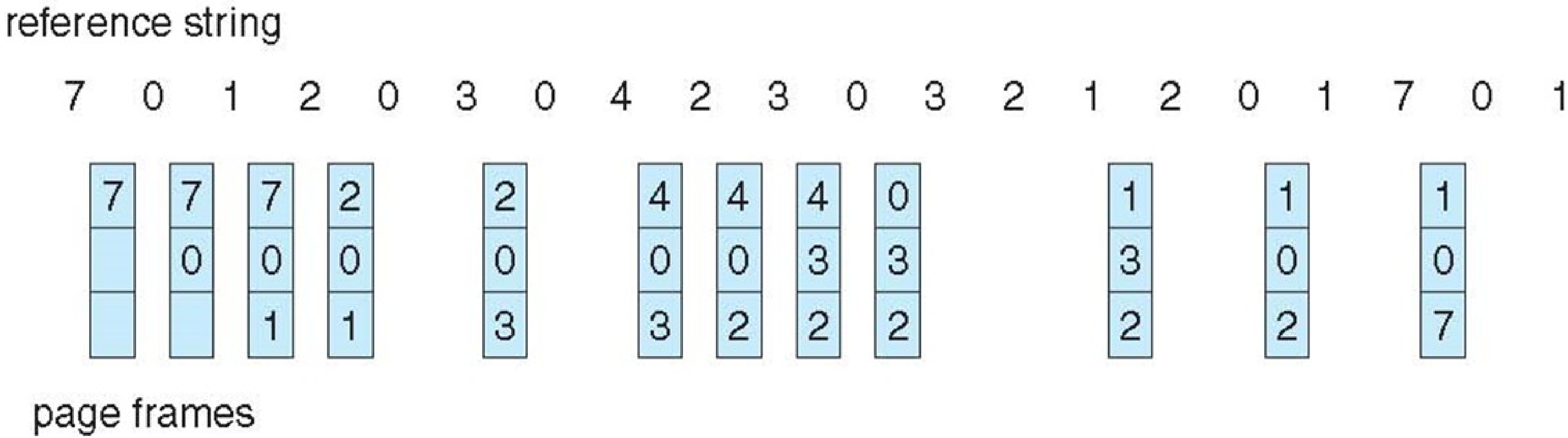



- Total page faults are 9
- Page fault rate = 9/21 = 0.42

The use of Optimal Page replacement is to set up a **benchmark** so that other replacement algorithms can be analyzed against it.

# LRU Algorithm

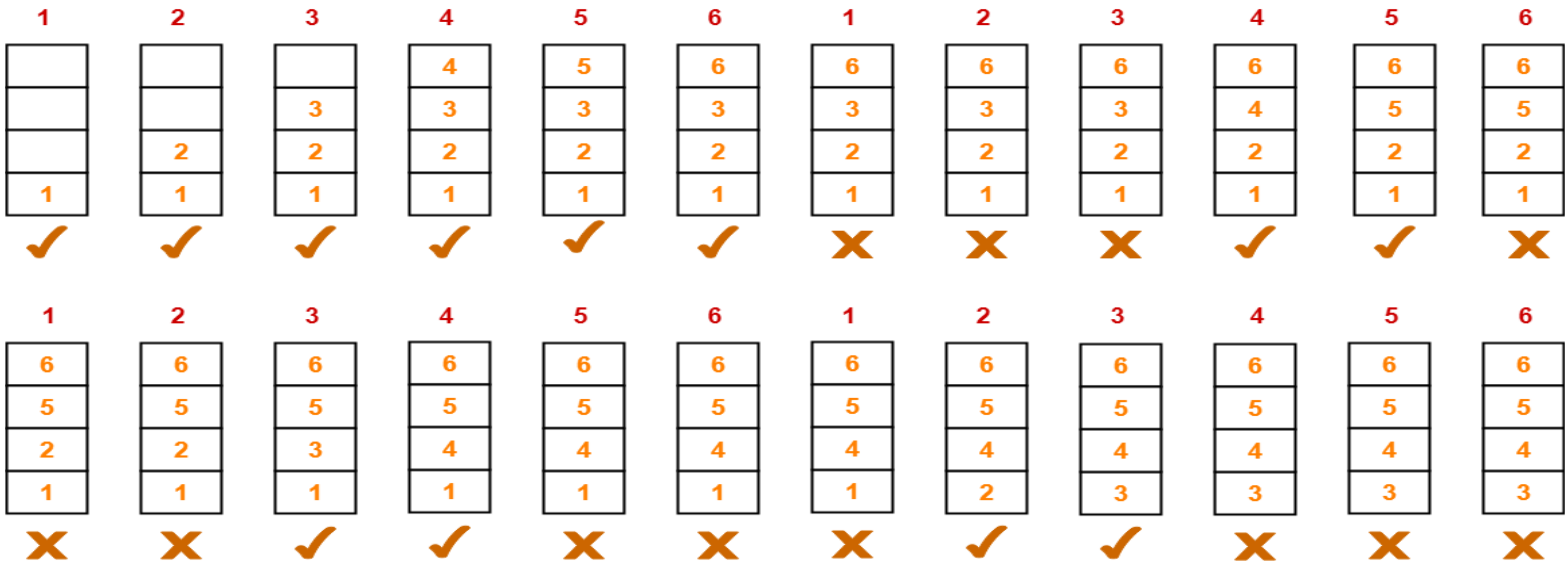
- In this algorithm page will be replaced which is least recently used.
- Consider page reference string: **7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1** with 3 page frames. Find number of page faults and page fault rate.




- Total page faults are 12
- Page fault rate = 12/21 = 0.57

# MRU Algorithm

- In this algorithm page will be replaced which is most recently used.
- Consider page reference string: **1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6** with 4 page frames. Find number of page faults and page fault rate.




- Total page faults are 12
- Page fault rate = 12/24 = 0.5

# Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with **Smallest count**.

- MFU Algorithm: replaces page with **Highest count**.

# LFU Algorithm

| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $1_1$ | $1_1$ | $1_1$ | $1_1$ | $1_2$ | $1_2$ | $1_2$ | $1_2$ | $1_2$ | $1_2$ | $1_2$ | $1_2$ |
| | $2_1$ | $2_1$ | $2_1$ | $2_1$ | $2_2$ | $2_2$ | $2_2$ | $2_2$ | $2_2$ | $2_2$ | $2_2$ |
| | | $3_1$ | $3_1$ | $3_1$ | $3_1$ | $5_1$ | $5_1$ | $5_1$ | $5_1$ | $4_1$ | $5_1$ |
| | | | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $3_1$ | $3_1$ | $3_1$ |
| * | * | * | * | | | * | | | * | * | * |

***NOTE:***

- *The subscripted number in red gives the counter number.*
- *In case of tie of counter numbers, the page with oldest arrival time is replaced.*
- ** : Page Replacement occurs*
- ***8 page replacements** are done in this case.*

# MFU Algorithm

| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $1_1$ | $1_1$ | $1_1$ | $1_1$ | $1_2$ | $1_2$ | $5_1$ | $5_1$ | $5_1$ | $5_1$ | $4_1$ | $4_1$ |
| | $2_1$ | $2_1$ | $2_1$ | $2_1$ | $2_2$ | $2_2$ | $1_1$ | $1_1$ | $1_1$ | $1_1$ | $5_1$ |
| | | $3_1$ | $3_1$ | $3_1$ | $3_1$ | $3_1$ | $3_1$ | $2_1$ | $2_1$ | $2_1$ | $2_1$ |
| | | | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $3_1$ | $3_1$ | $3_1$ |
| * | * | * | * | | | * | * | * | * | * | * |

***NOTE:***

- *The subscripted number in red gives the counter number.*
- *In case of tie of counter numbers, the page with oldest arrival time is replaced.*
- ** : Page Replacement occurs*
- ***10 page replacements** are done in this case.*

# Allocation of frames

- Frame allocation algorithms are used if you have multiple processes; it helps decide how many frames to allocate to each process.
- Each process needs ***minimum*** number of frames
- **Major allocation schemes**
  - Equal fixed allocation
  - Proportional fixed allocation
  - Priority allocation
- **Major replacement policies**
  - Global replacement
  - Local replacement

- **Equal fixed allocation**: frames divided evenly among processes. For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
- **proportional fixed allocation**: partition weighted by process size.

$$s_i = \text{size of process } p_i$$
$$S = \sum s_i$$
$$m = \text{total number of frames}$$
$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

$$m = 64$$
$$s_1 = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 62 \approx 4$$
$$a_2 = \frac{127}{137} \times 62 \approx 57$$

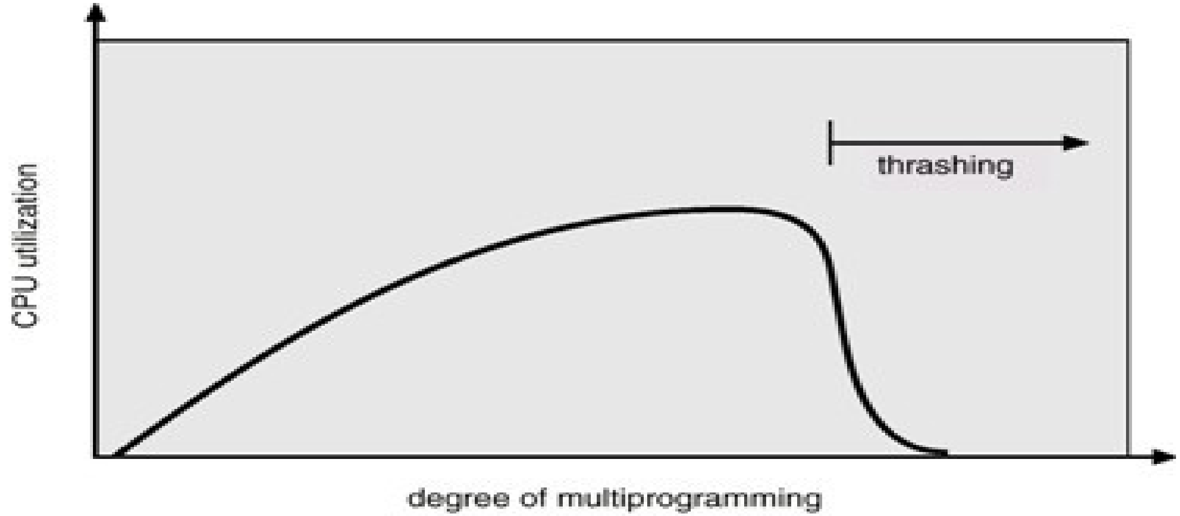
- **Priority Allocation**:
  - Use a proportional allocation scheme using priorities rather than size
  - If process Pi generates a page fault,
    - Select for replacement one of its frames
    - Select for replacement a frame from a process with lower priority number

- **Global replacement**: process selects a replacement frame from the set of all frames; one process can take a frame from another.

- **Local replacement**: each process selects from only its own set of allocated frames.

# Thrashing

- If a process does not have enough frames to hold its current working set, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - Operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system.




- Thrashing: a process is busy swapping pages in and out
- Why does paging work?
  - Locality model
    - Process migrates from one locality to another.
    - Localities may overlap.
- Why does thrashing occur?
  - $\sum$ size of locality > total memory size
- What should we do?
  - suspend one or more processes!


THANK YOU