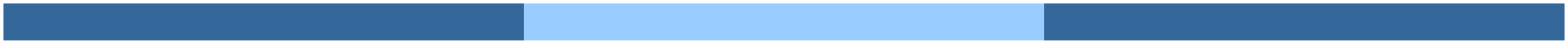
Virtual Memory

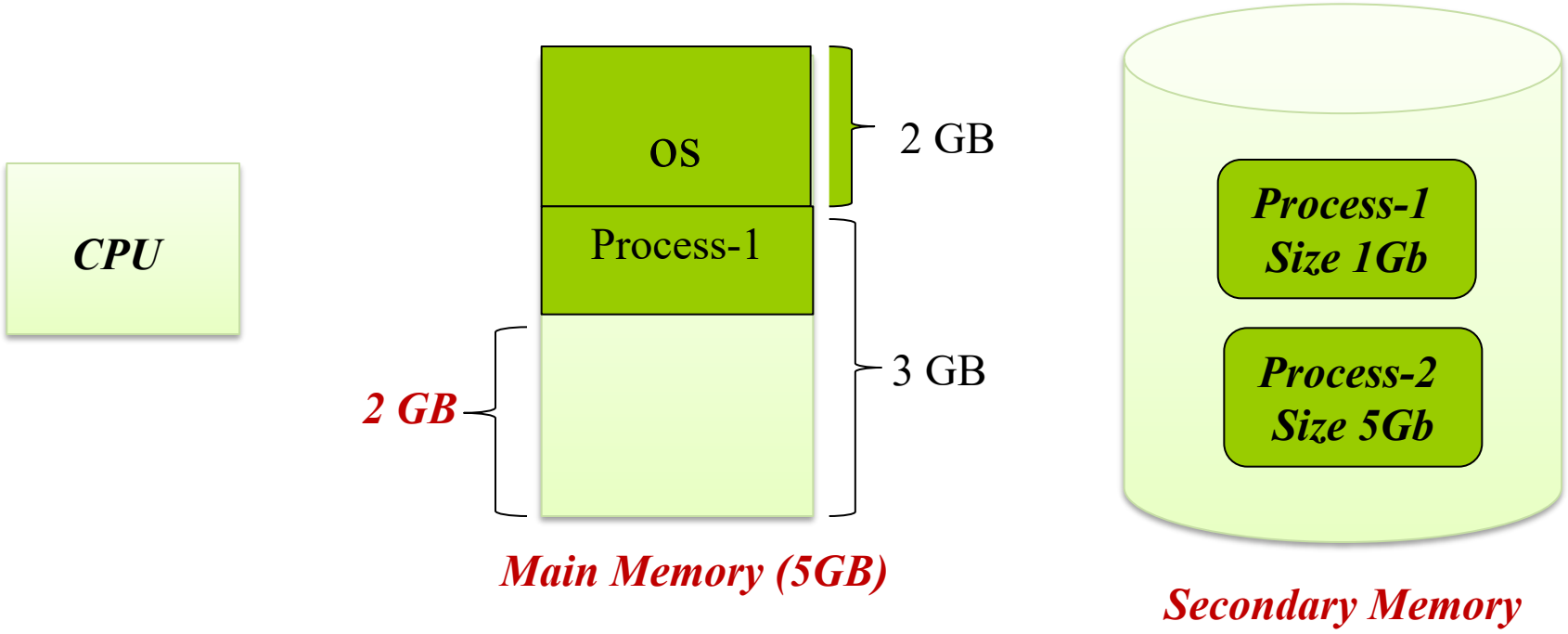


Contents

- ***Background***
- ***Demand Paging***
- ***Performance of Demand Paging***
- ***Page Replacement***
- ***Page-Replacement Algorithms***
- ***Allocation of Frames***
- ***Thrashing***

Requirement of Virtual Memory Technique

Problem1: *Process – 1 can execute because size of Process-1 is smaller than available main memory but there is a problem with Process-2, available main memory is not enough to store process 2.*



Problem2: To keep many processes in main memory simultaneously (***increase degree of multiprogramming.)***

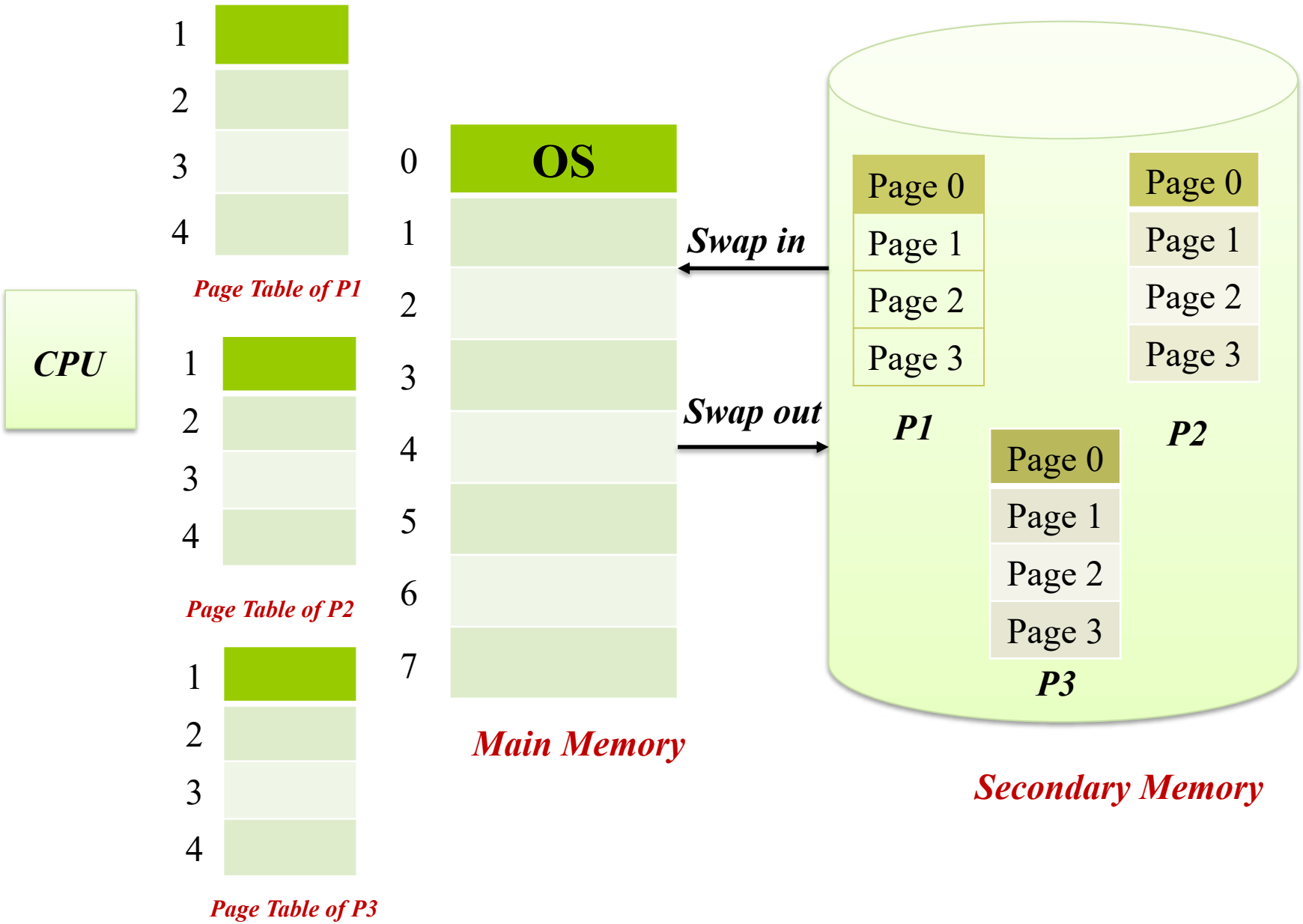
Virtual Memory

- ***Virtual Memory*** *Virtual memory is a technique that allows the execution of processes that are not completely in memory, and 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****.*
- *By virtual memory technique, 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.***

Virtual Memory Implementation

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

Virtual Memory



Virtual Memory: Advantages

- *A program would no longer be constrained by the **amount** of physical memory that is available. Users would be able to write programs for an extremely large **virtual address space**, simplifying the programming task.*
- *Because each user program could take less physical memory, **more programs could be run** at the same time, with a corresponding increase in **CPU utilization** and **throughput** but with no increase in response time or turnaround time.*
- *Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.*

Demand Paging

- ***Demand paging:*** *A technique to load pages only as they are needed. When a process is swapped in, **its pages** are not swapped in **all at once**. Rather they are swapped in only when the process needs them (on demand). Pages that are never accessed are thus never loaded into the physical memory.*

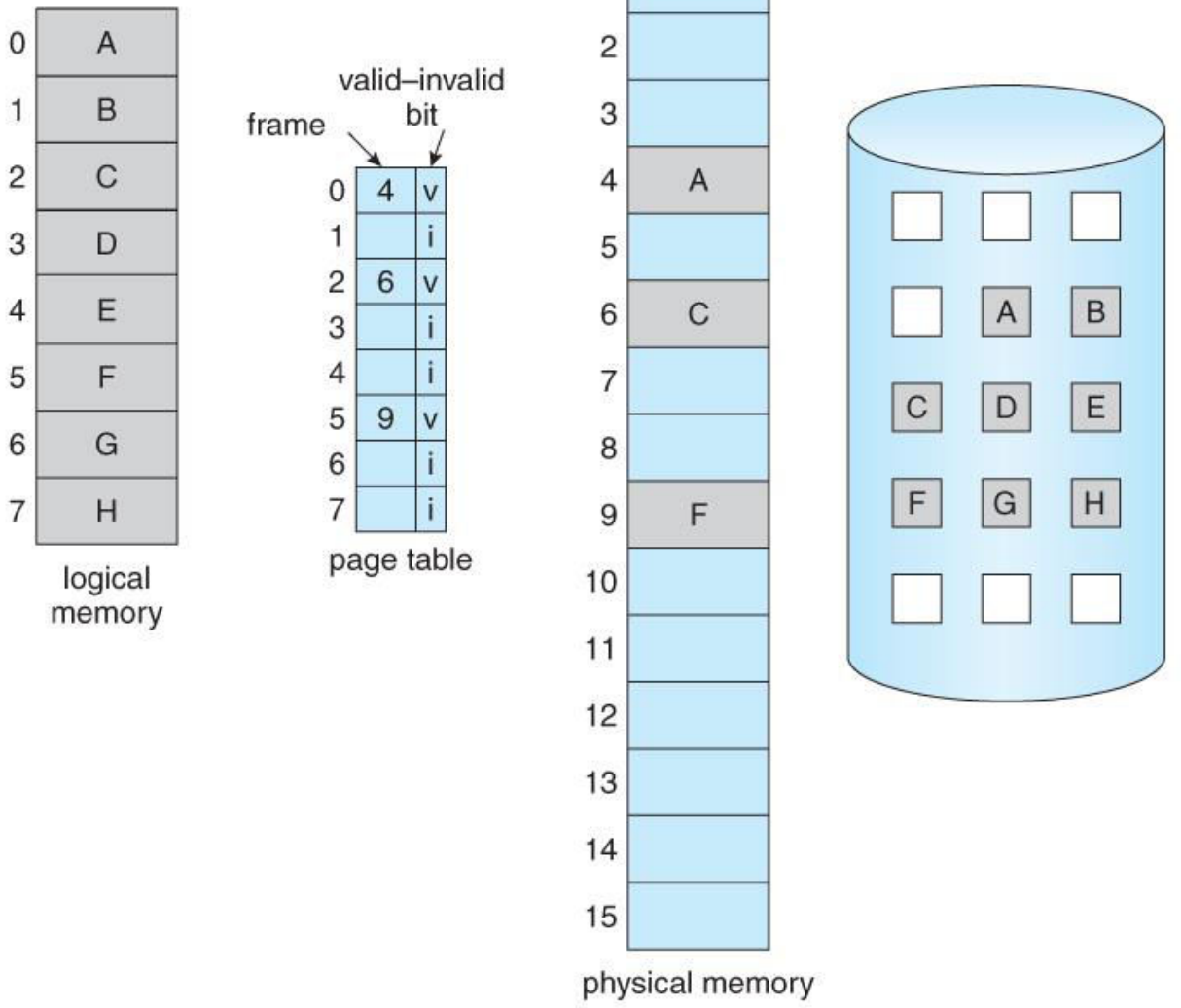
- ***Lazy swapper:*** *A lazy swapper never swaps a page into memory unless that page is needed, although a **pager** is a more accurate term.*

Page Table When Some Pages Are Not in Main Memory

Hardware support to distinguish pages on memory and on disk.

Valid- Associated page is legal and in memory

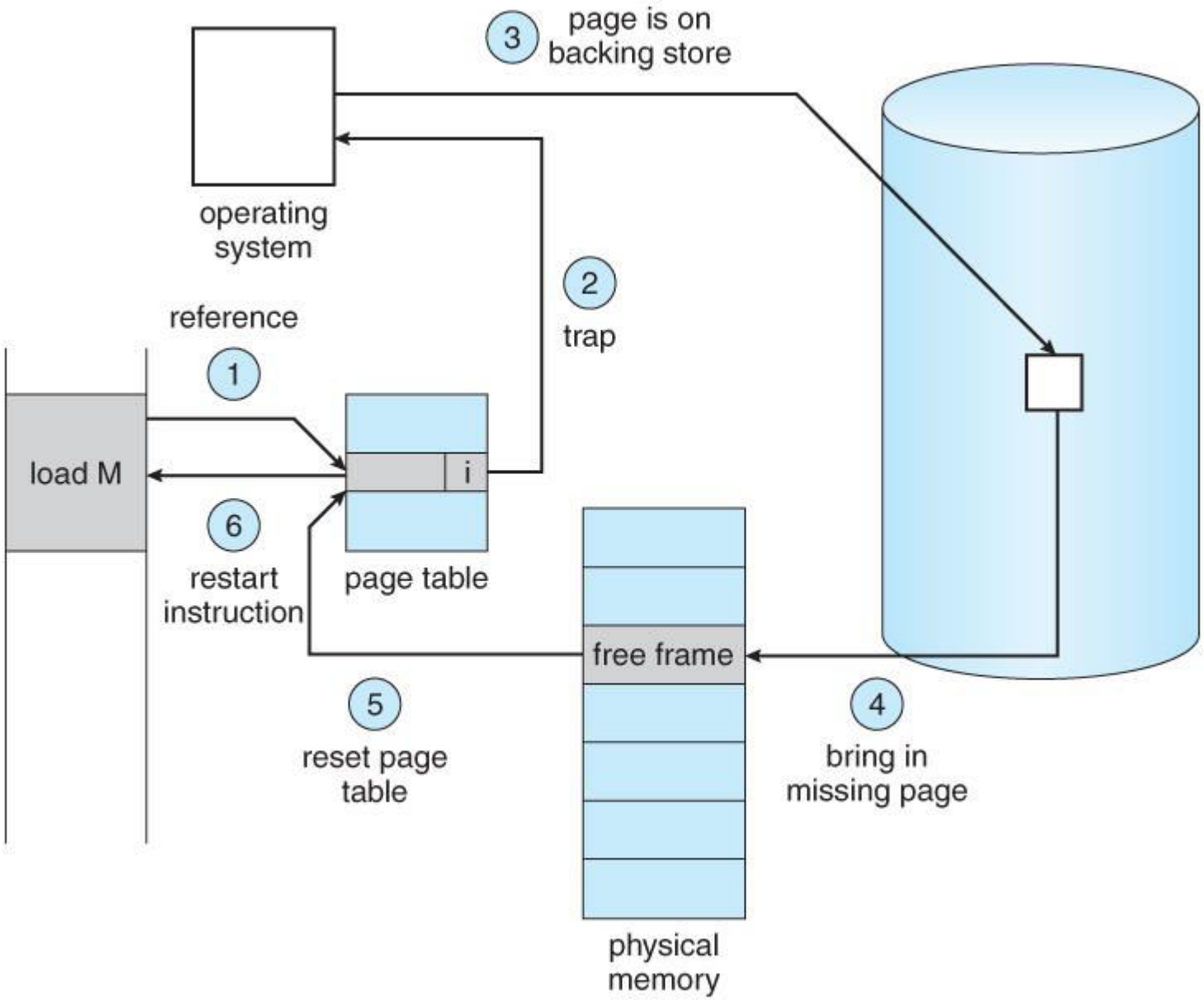
Invalid- Associated page either is not valid or valid but currently on disk.



Page Fault

- ***Page Fault:***
- When a ***page referenced by the CPU*** is not found in the main memory, it is called as a page fault.
- Access to a page marked ***invalid*** causes page fault.
- When a page fault occurs, the required page has to be fetched from the secondary memory into the main memory.

Steps to Handling a Page Fault



Steps to Handling a Page Fault

If a page is needed that was not originally loaded up, then a ***page fault trap*** is generated, which must be handled in a series of **steps**:

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)

Steps to Handling a Page Fault

5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU.)

Demand Paging

Pure demand paging: No pages are swapped in for a process until they are requested by page faults.

In theory each instruction could generate multiple page faults. In practice this is very rare, due to ***locality of reference.***

Performance of Demand Paging

- *Let **p** be the probability of a page fault ($0 \leq p \leq 1$).*
- ***ma** is memory access time*

Effective Access Time = p* page fault time + (1 - p) * (ma)

Performance of Demand Paging

Suppose that a normal memory access requires 200 nanoseconds, and that servicing a page fault takes 8 milliseconds. (8,000,000 nanoseconds), if only one access in 1000 causes a page fault, then effective access time will be-

$$= (1 - p) * (200) + p * 8000000$$

$$= 200 + 7,999,800 * p$$

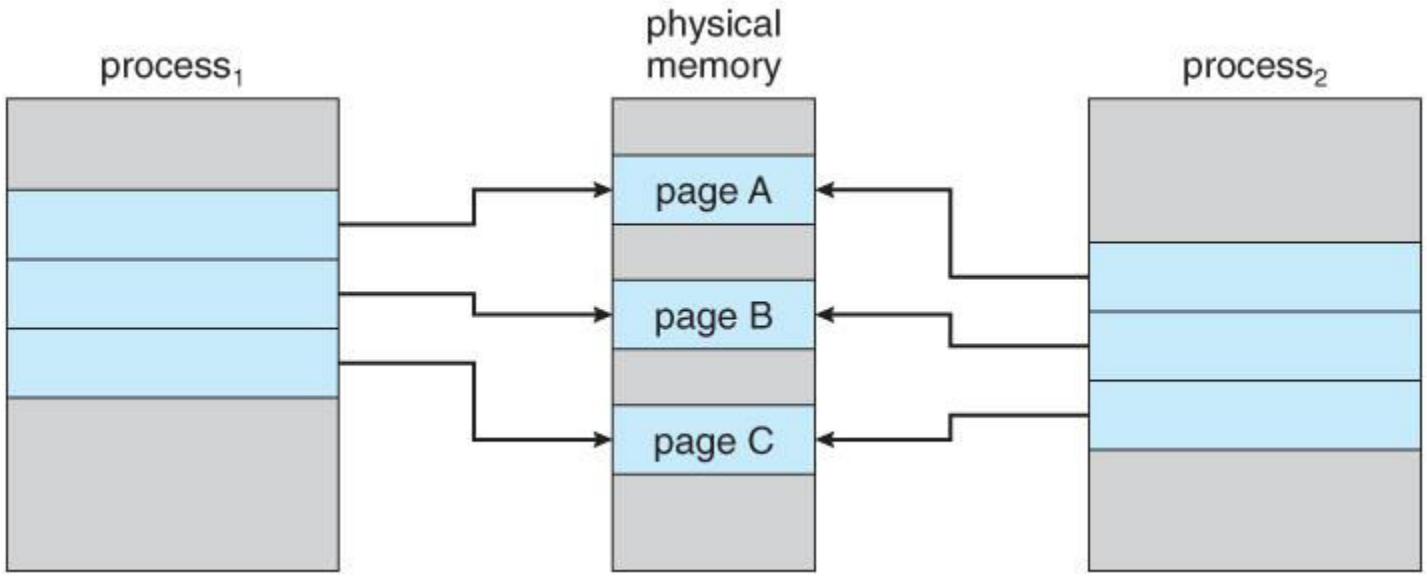
$$= 200 + 7,999,800 * (1/1000)$$

$$= 8.2 \text{ microseconds}$$

Copy on Write

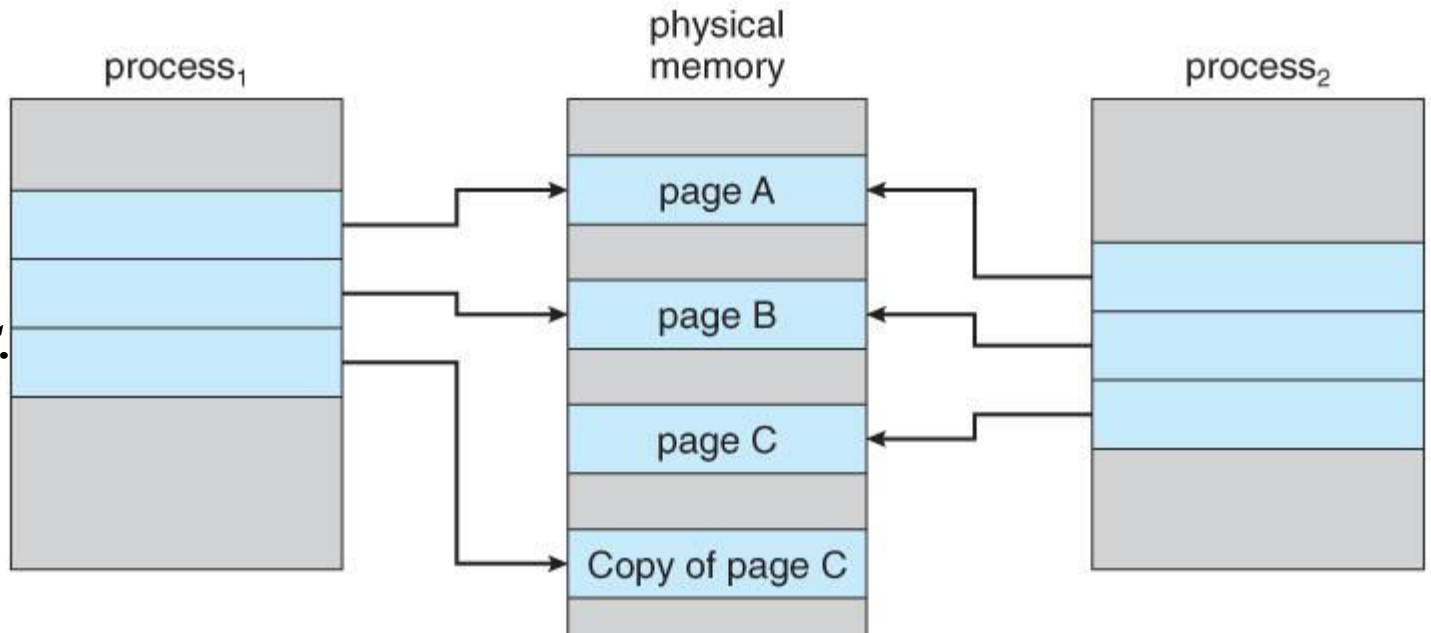
- ***Copy-on-write:*** *Allows parent and child processes initially share the same page, marked as copy on write pages, meaning that if any process will write to the shared page a copy of shared page is created.*

Copy on Write



Before process 1 modifies page C.

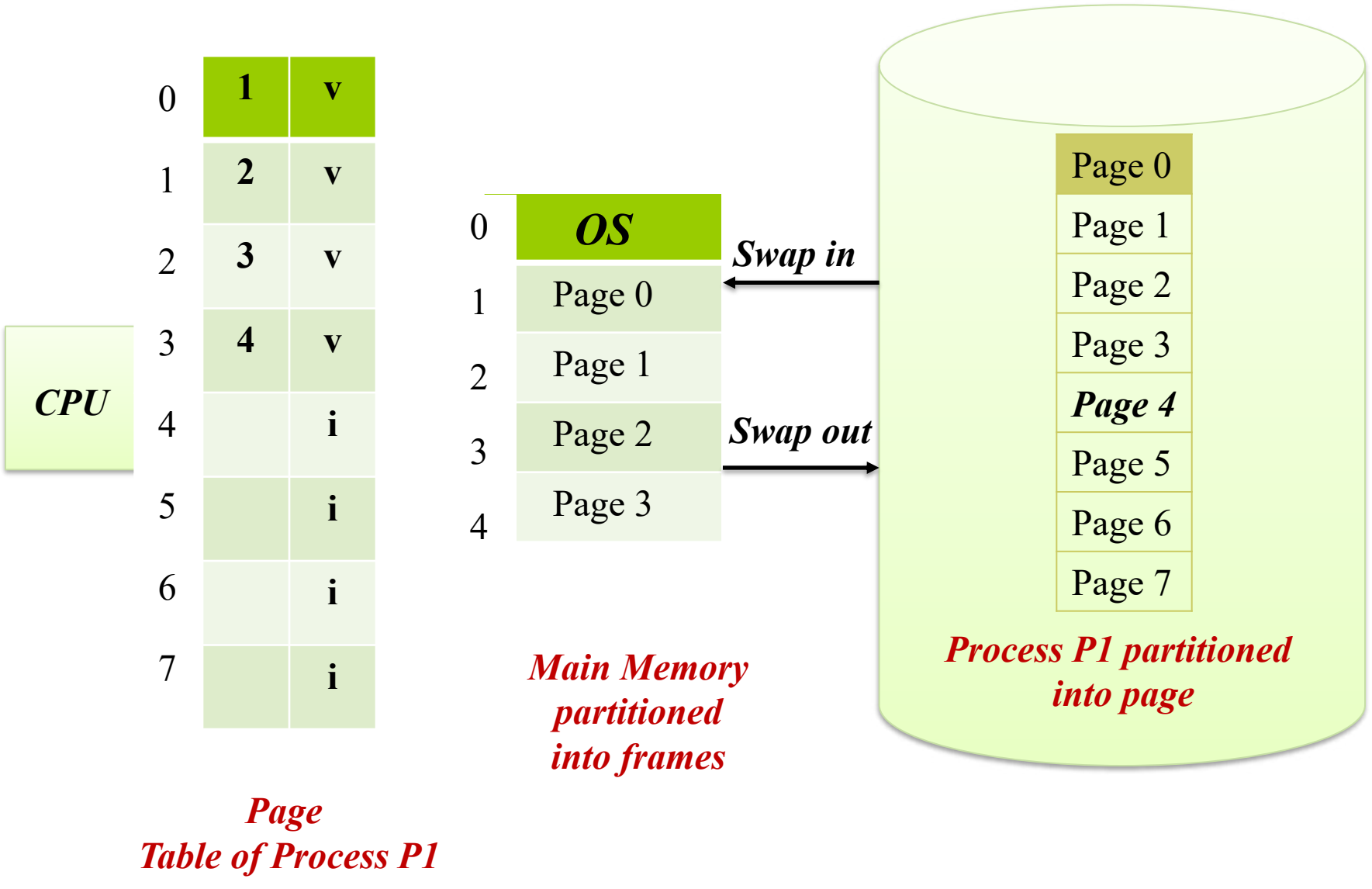
After process 1 modifies page C.



Copy on Write

- *Obviously only pages that can be modified even need to be labeled as **copy-on-write**. Code segments can simply be shared.*
- *Duplicated pages may get free pages from a pool of free pages maintained by OS. Pages used to satisfy copy-on-write duplications are typically allocated using **zero-fill-on-demand**, meaning that their previous contents are zeroed out before the copy proceeds.*
- *Some systems provide an alternative to the fork( ) system call called a **virtual memory fork, vfork( )**. In this case the parent is suspended, and the child uses the parent's memory pages. This is very fast for process creation, but requires that the child not modify any of the shared memory pages before performing the exec( ) system call.*

Page Replacement



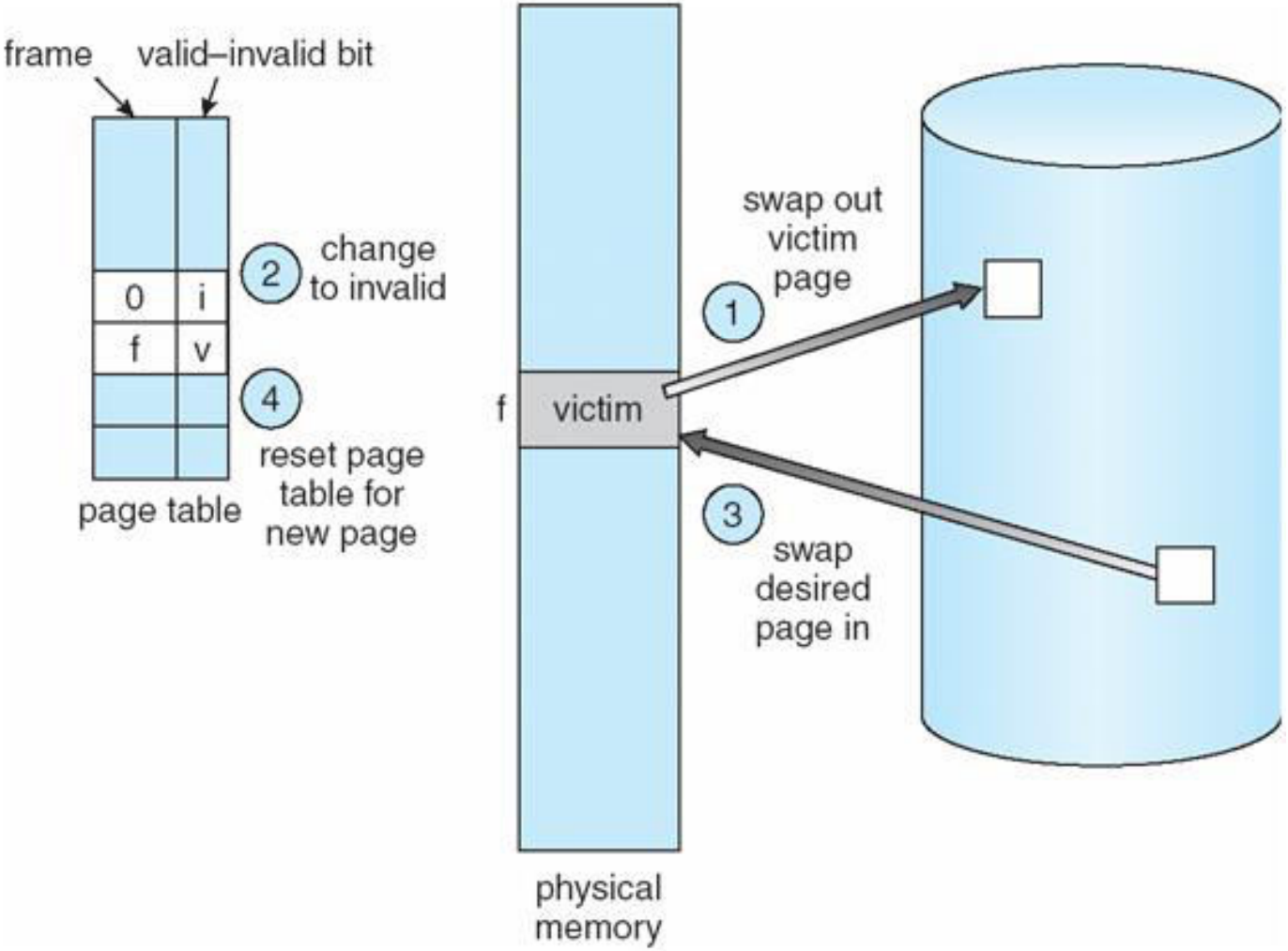
Page Replacement

Page replacement is required when-

- ***All the frames of main memory are already occupied.***

- *Thus, a page has to be replaced to create a room for the required page*
- *Find some page in memory that isn't being used right now, and swap that page only out to disk, freeing up a frame that can be allocated to the required page. This is known as* ***page replacement****.*
- *A* ***page replacement algorithm*** *is needed to decide which page needs to be replaced when new page comes in.*
- .

Page Replacement



Basic Page Replacement

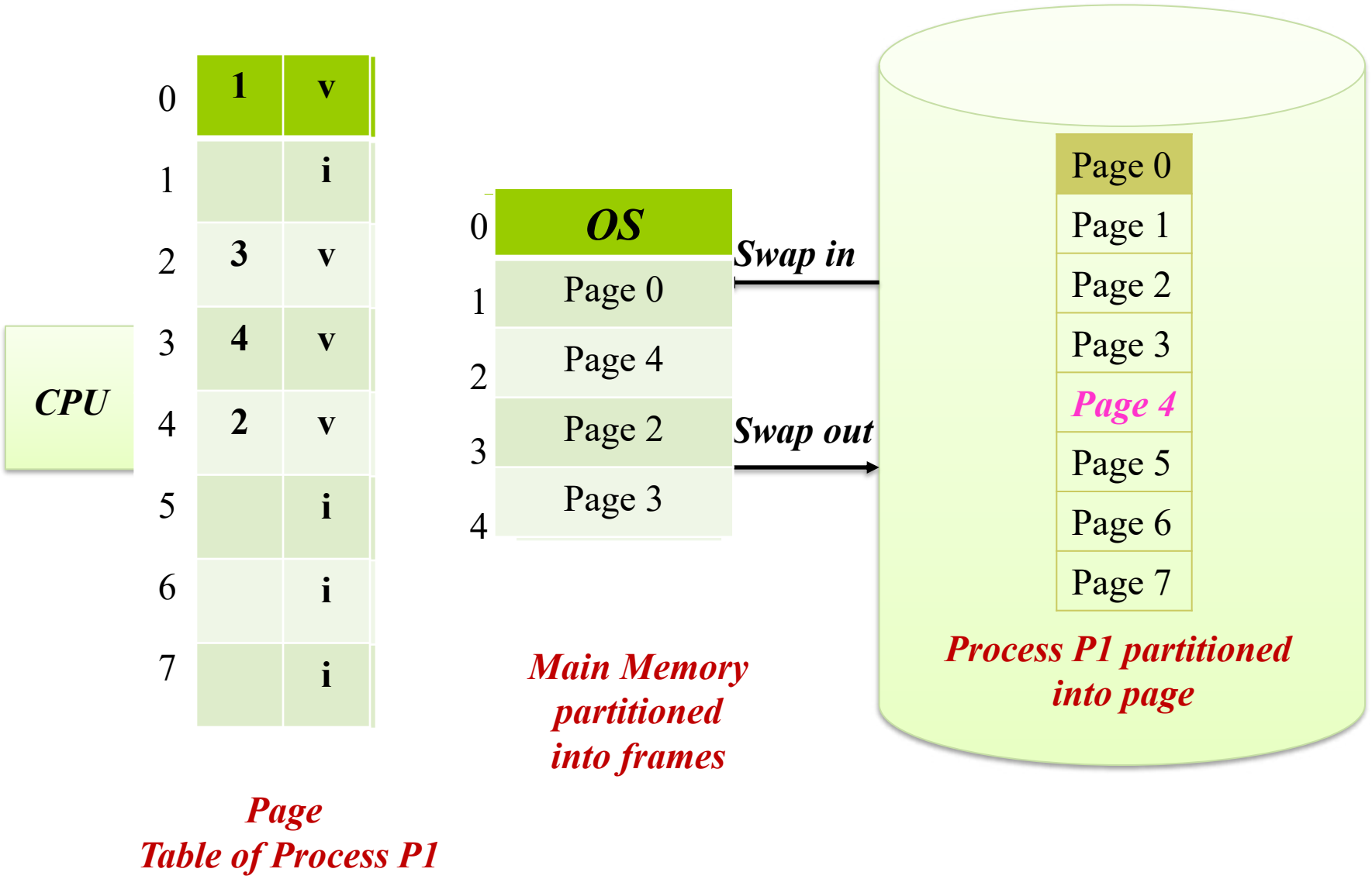
*The **page-fault** handling must be modified to free up a frame if necessary, as follows:*

- *Find the location of the desired page on the disk, either in swap space or in the file system.*
- *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 an existing frame to be replaced, known as the victim frame.*
  - *Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.*

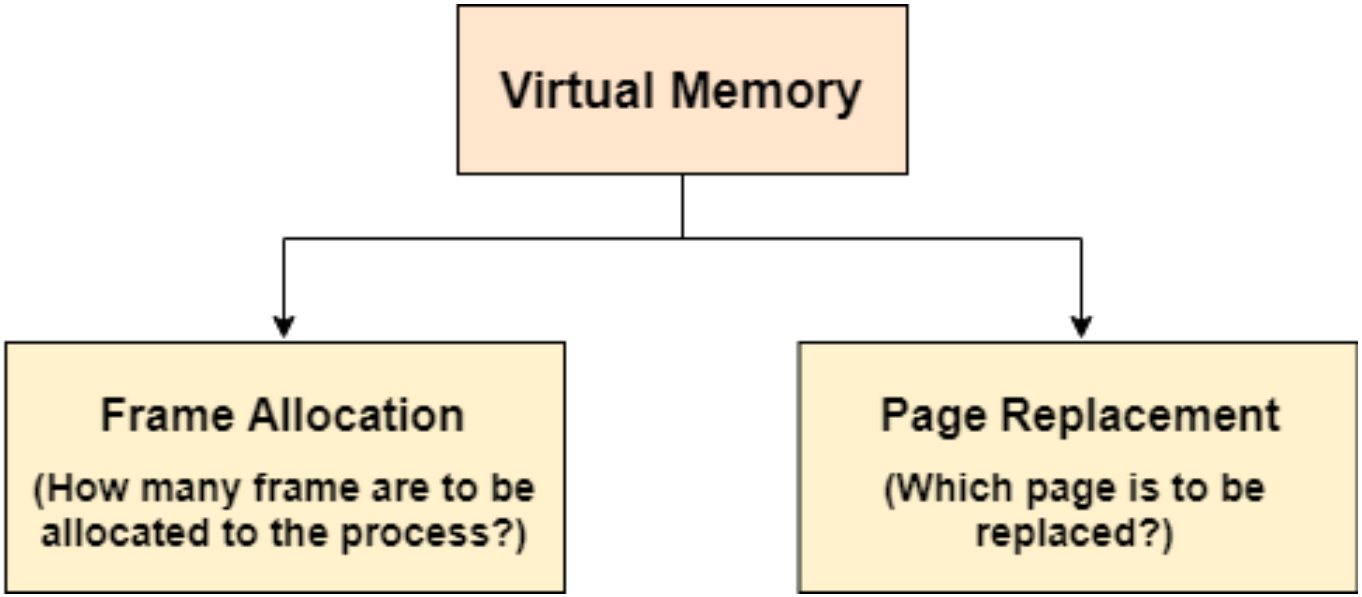
Basic Page Replacement

- *Read in the desired page and store it in the frame. Adjust all related page and frame tables to indicate the change.*
- *Restart the process that was waiting for this page.*

Page Replacement



Major Problems of Demand Paging



- ***It is very important to have the optimal frame allocation and page replacement algorithm***

- ***We want page replacement algorithm with lowest page fault rate.***

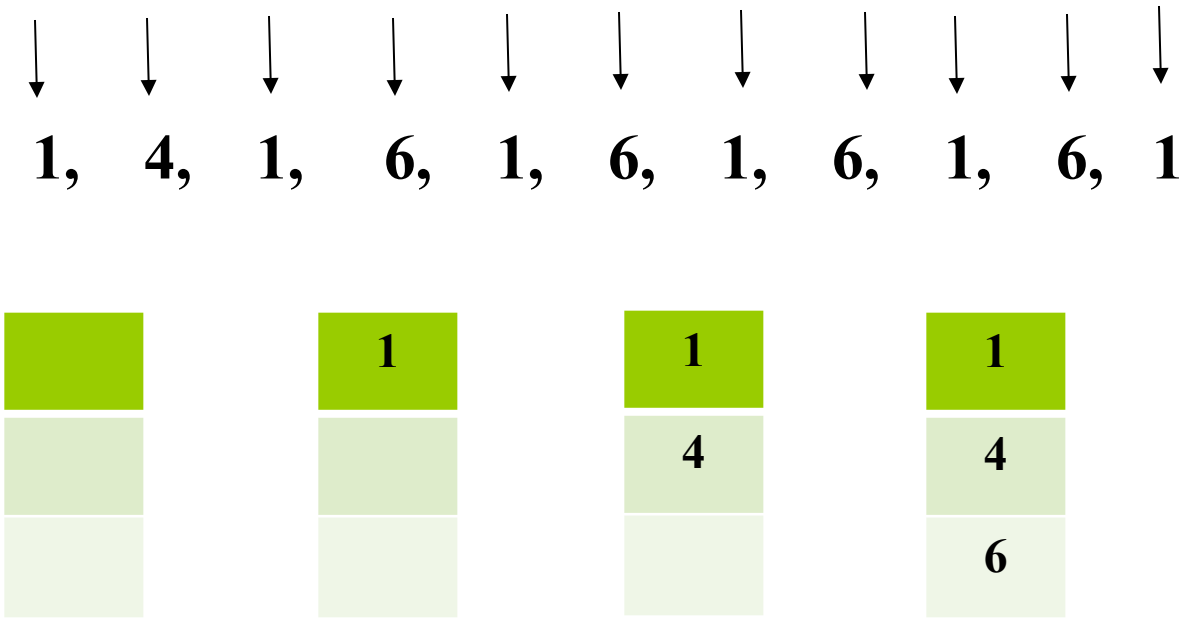
Page-Replacement Algorithms

- *Want **lowest page-fault rate**.*
- *Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.*
- *For example if the reference string is*

1, 4, 1, 6, 1, 6, 1, 6, 1, 6,

Page-Replacement Algorithms

- For example if there are ***3 frames*** and ***reference string*** is:



Total Page fault = 3.

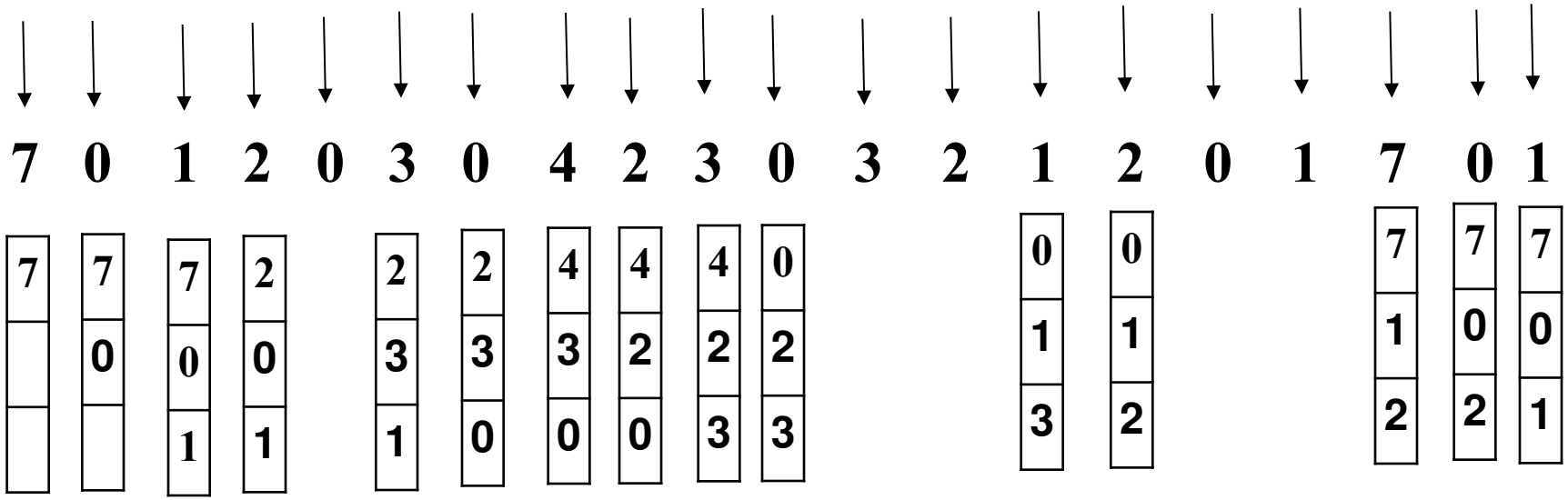
If number of frame is only 1:

Number of page fault: 11

FIFO Page Replacement Algorithms

- *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 **FIFO queue**.*
- *We replace page at the head of queue. When a page is brought into memory we insert it at the tail of queue.*

FIFO Page Replacement Algorithms



States of the FIFO queue:

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

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

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

Total Page Fault = 15

*Hit Ratio = (No. of Hits / total references)*100*

*Hit Ratio = (5/20)*100= 25%*

FIFO Page Replacement Algorithms

Consider a main memory with ***five*** page frames and the following sequence of page references: ***3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3.***

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	1	1	1
Frame 4					9	9	9	9	9	9	9	9	2	2	2
Frame 3			2	2	2	2	2	2	8	8	8	8	8	8	8
Frame 2		8	8	8	8	8	8	3	3	3	3	3	3	3	3
Frame 1	3	3	3	3	3	3	6	6	6	6	6	6	6	6	6
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults = 9

Number of hits = 6

Problem with FIFO Page Replacement Algorithms

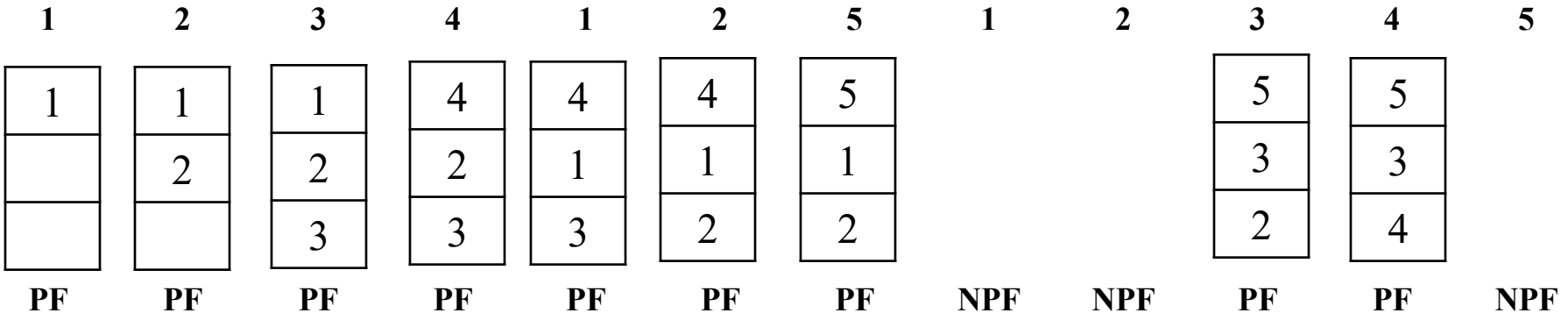
- *Although FIFO is simple and easy, it is not always optimal, or even efficient.*
- *The number of page faults should either decrease or remain constant on increasing the number of frames in main memory.*
- *But sometimes the unusual behavior is observed, on increasing the number of frames in main memory, the number of page faults also increase.*
- ***Belady's Anomaly:*** *Iincreasing the number of frames available can actually* ***increase*** *the number of page faults.*

Problem with FIFO Page Replacement Algorithms

Belady's Anomaly: *Iincreasing the number of frames available can actually* ***increase*** *the number of page faults.*

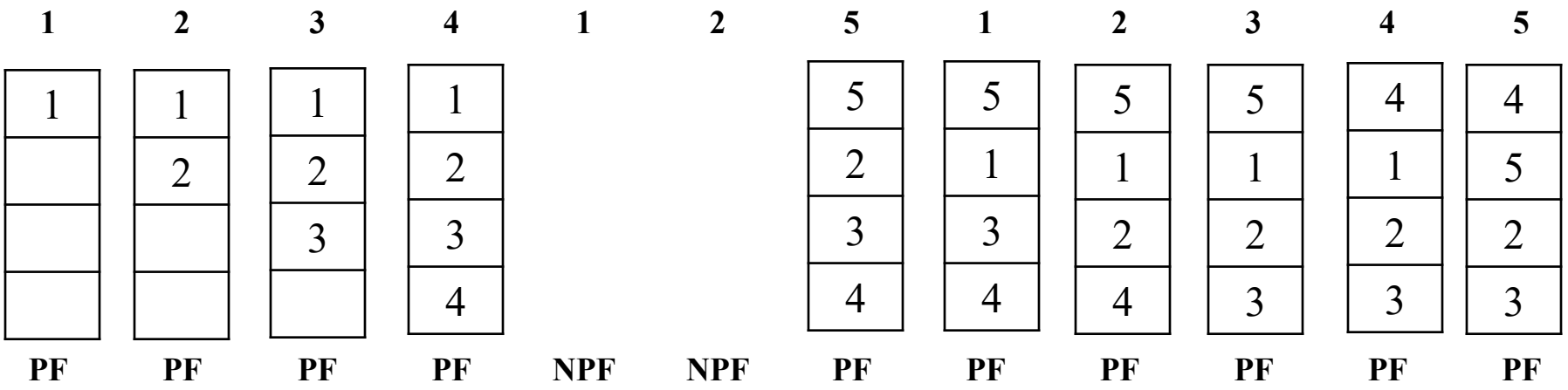
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Case-1: If the system has ***3 frames***, the given reference string on using FIFO page replacement algorithm yields a total of 9 page faults.



Problem with FIFO Page Replacement Algorithms

Case-2: If the system has ***4 frames,*** the given reference string on using FIFO page replacement algorithm yields a total of 10 page faults.

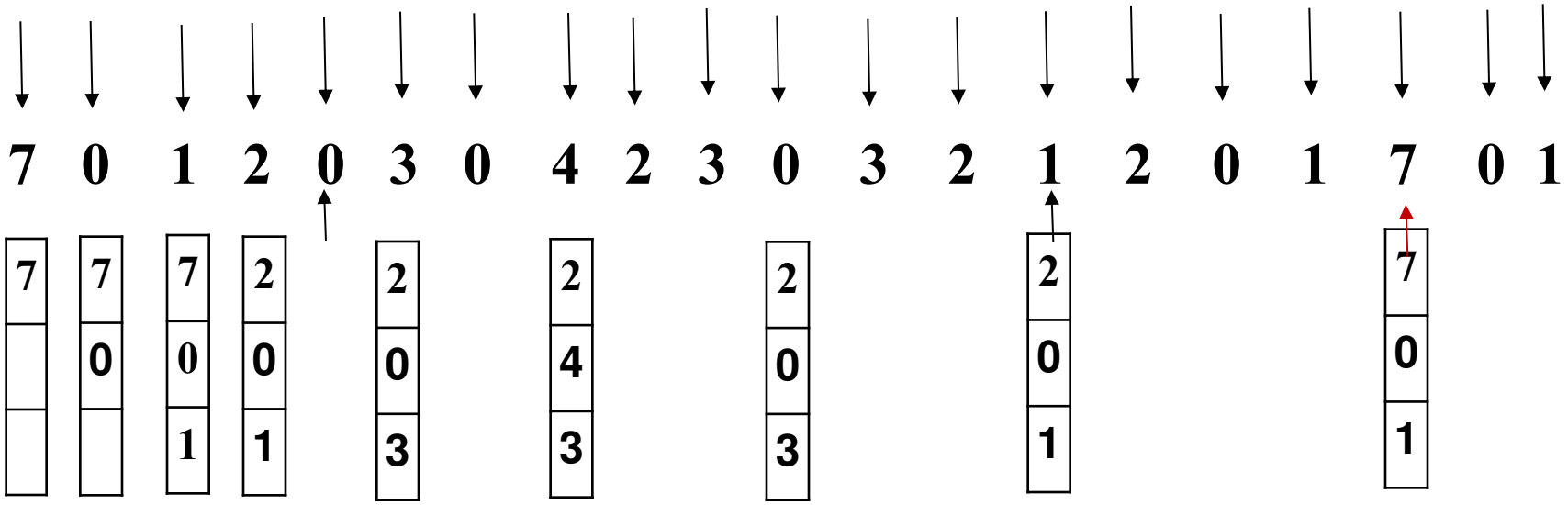


It can be seen from the above example that on increasing the number of frames while using the FIFO page replacement algorithm, the number of page faults increased from 9 to 10.

Optimal Page Replacement Algorithms

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

Optimal Page Replacement Algorithms



Total Page fault = 9.

Total hit = 11

Total miss = 9

Optimal Page Replacement Algorithms

- 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

Also calculate the ***hit ratio*** and ***miss ratio***.

Optimal Page Replacement Algorithms

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

Total page fault = 5

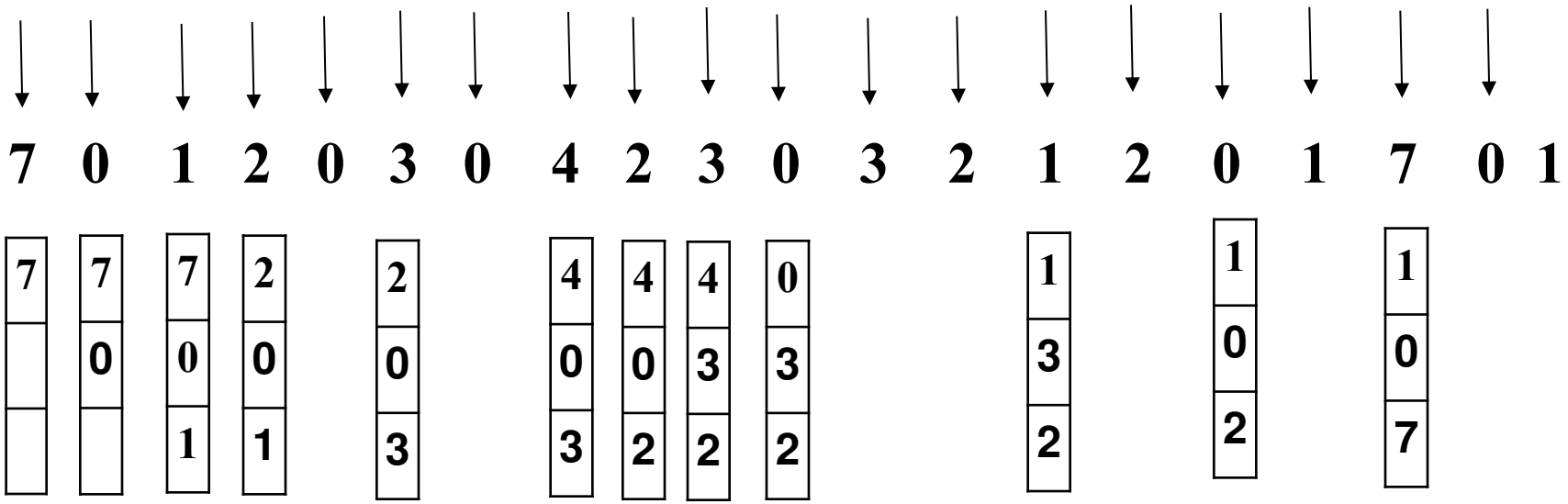
Hit ratio = 0.5 or 50%

Miss ratio = 0.5 or 50%

LRU Replacement Algorithms

- 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.***

LRU Replacement Algorithms



Total Page Fault: 12

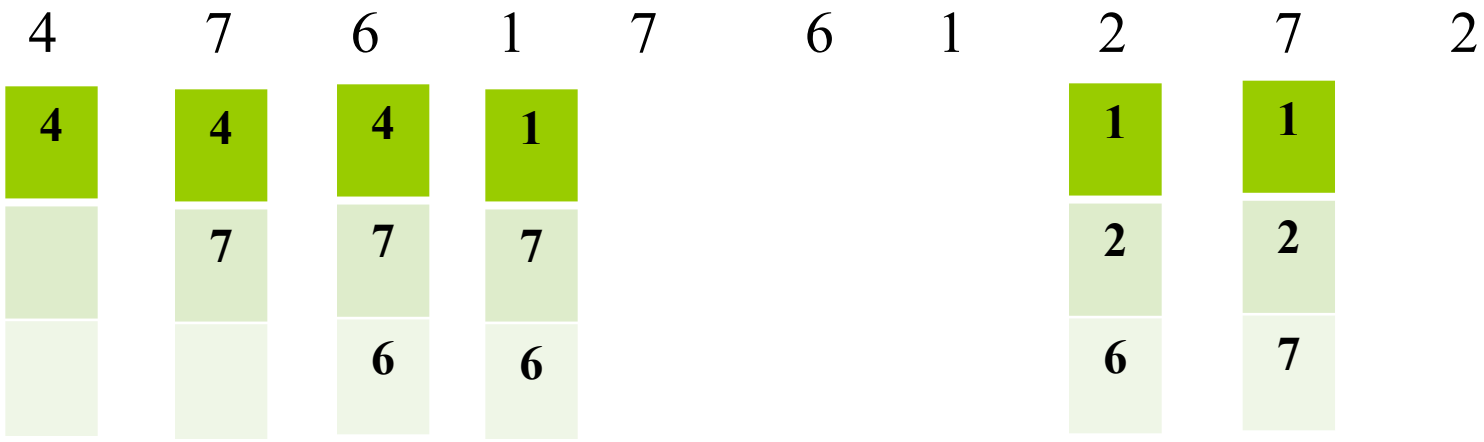
LRU Replacement Algorithms

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

Also calculate the hit ratio and miss ratio.

LRU Replacement Algorithms



Total number of page faults occurred = 6

Hit ratio = 40%
Miss ratio = 60%

Belady's Anomaly

Belady's Anomaly is the phenomenon of increasing the number of page faults on increasing the number of frames in main memory.

Following page replacement algorithms suffer from Belady's Anomaly-

- *FIFO Page Replacement Algorithm*
- *Random Page Replacement Algorithm*
- *Second Chance Algorithm*

Belady's Anomaly

Reason Behind Belady's Anomaly-

- An algorithm suffers from Belady's Anomaly if and only if it does not follow ***stack property.***
- Algorithms that follow stack property are called as ***stack based algorithms***.
- Stack based algorithms do not suffer from Belady's Anomaly.

Following page replacement algorithms are stack based algorithms-

1. LRU Page Replacement Algorithm
2. Optimal Page Replacement Algorithm

Stack Property-

Consider-

- Initially, we had ***'m'*** number of frames in the main memory.
- Now, the number of frames in the main memory is increased to ***'m+1'.***

According to stack property-

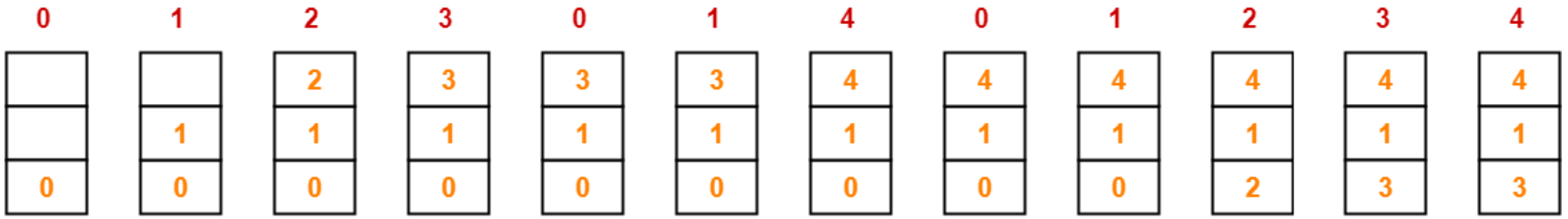
- At each stage, the set of pages that were present in the main memory when number of frames is ***'m'*** will be compulsorily present in the corresponding stages in main memory when the number of frames is increased to ***'m+1'.***
- ***the set of pages in memory for m frames is always a subset of the set of pages that would be in memory with m + 1 frames.***

OPTIMAL Page Replacement Algorithm

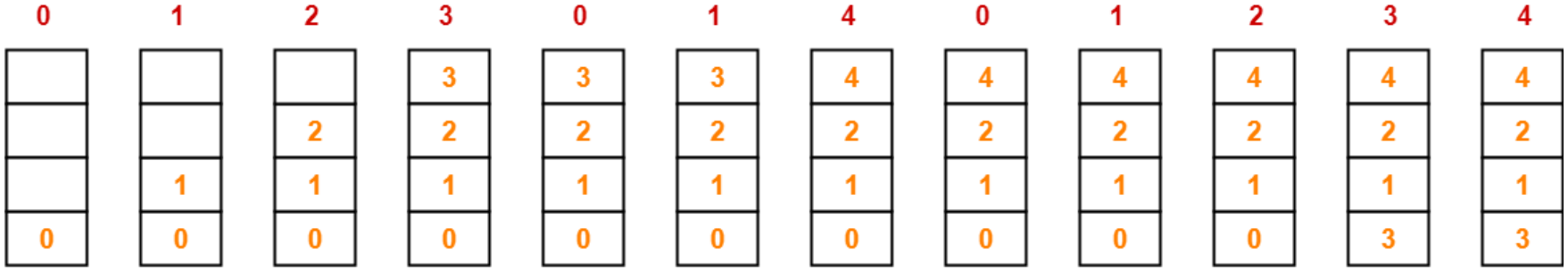
Consider the reference string is-

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

Now, observe the following two cases- Frame =3, page fault =7



Frame =4, page fault =6



Optimal Page Replacement Algorithm

From here, we can observe-

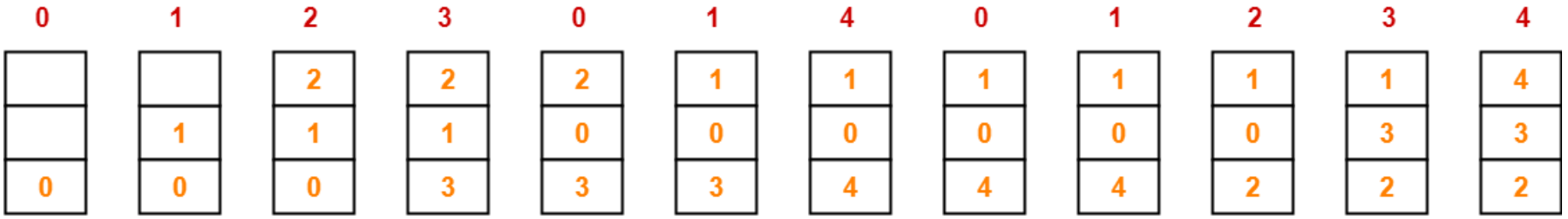
- At all the stages in case-02, main memory compulsorily contains the set of pages that are present in the corresponding stages in case-01.
- Thus, optimal page replacement algorithm follows the stack property.
- Hence, it does not suffer from Belady's Anomaly.
- As a proof, number of page faults decrease when the number of frames is increased from 3 to 4.

LRU Page Replacement Algorithm

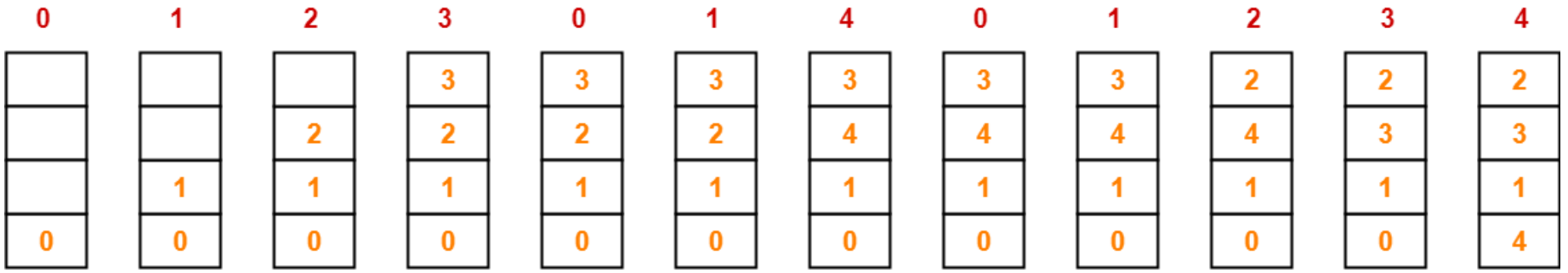
Consider the reference string is-

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

Now, observe the following two cases- Frame = 3, page fault = 10



Frame = 4, page fault = 8

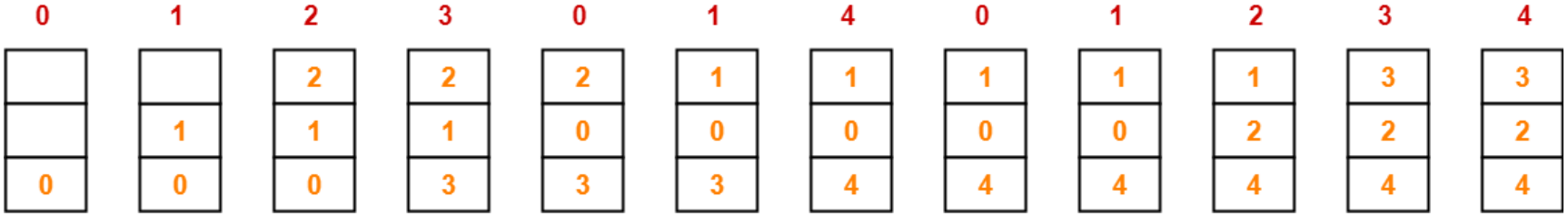


FIFO Page Replacement Algorithm

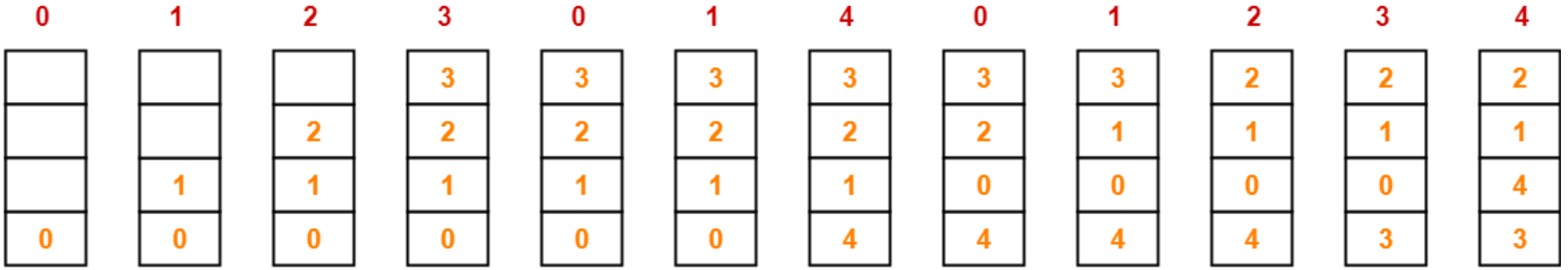
Consider the reference string is-

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

Now, observe the following two cases- Frame = 3, page fault = 9



Frame = 4, page fault = 10

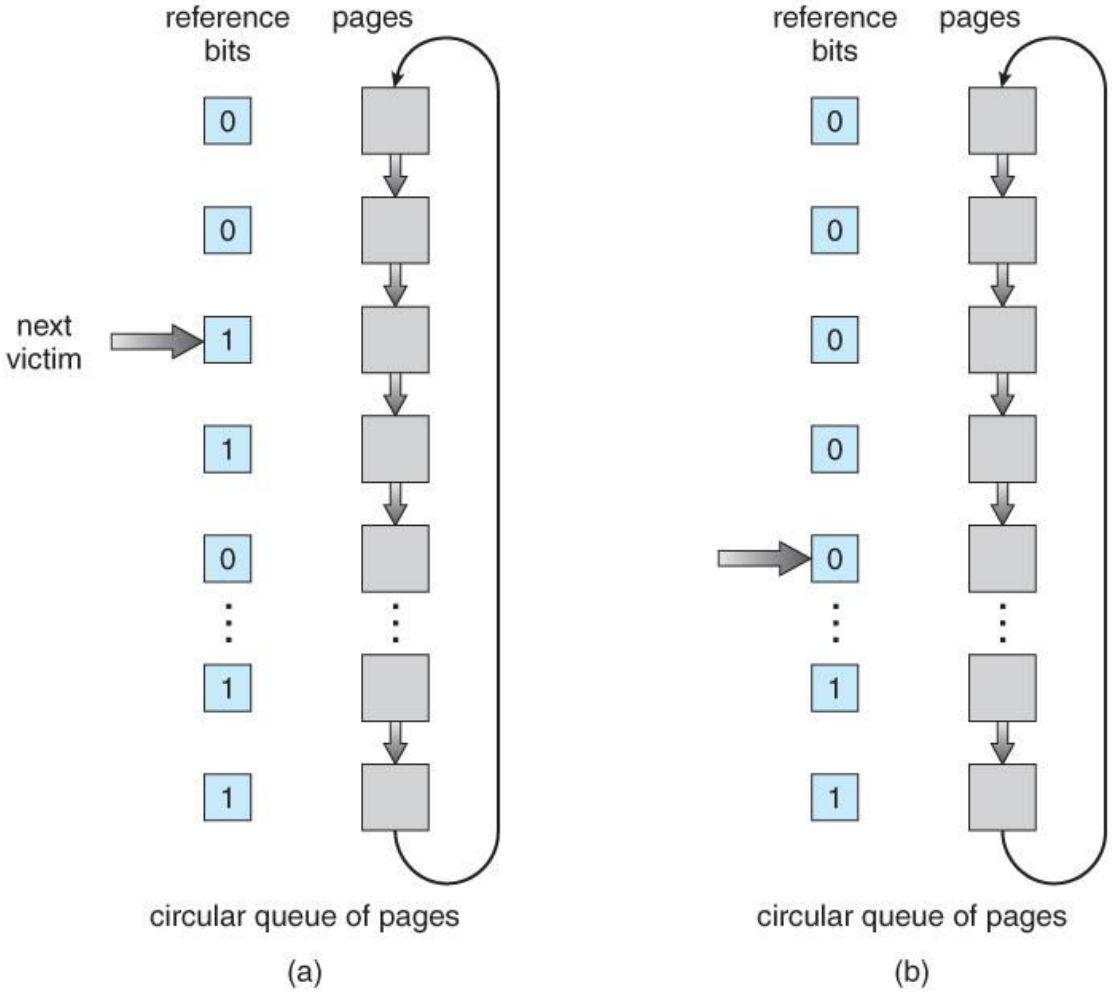


Second Chance Algorithm (Clock Algorithm):

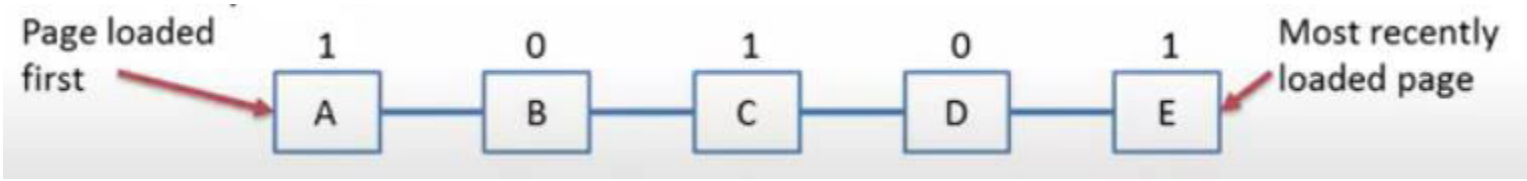
- *The **second chance** algorithm is a modified FIFO algorithm, A reference bit is used to give pages a second chance at staying in the page table.*
- *A new page read into a memory frame has the **reference bit** set to 0.*
- *Each time a memory frame is referenced, set the **reference bit** to 1 - this will give the frame a second chance.*
- *When a page has been selected to be replace, we inspect its **reference bit**.*
- *If value of reference bit is **0**, proceed to replace this page.*
- *If reference bit is set to **1,** give a second chance to this page and move on to select **next FIFO Page**.*
- *On second chance reference bit of the page is cleared and arrival time is reset to the **current time**.*

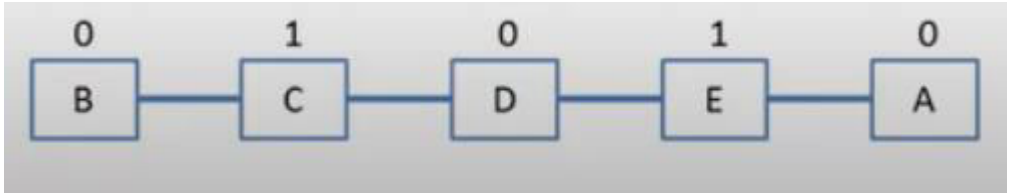
Second Chance Algorithm (Clock Algorithm):

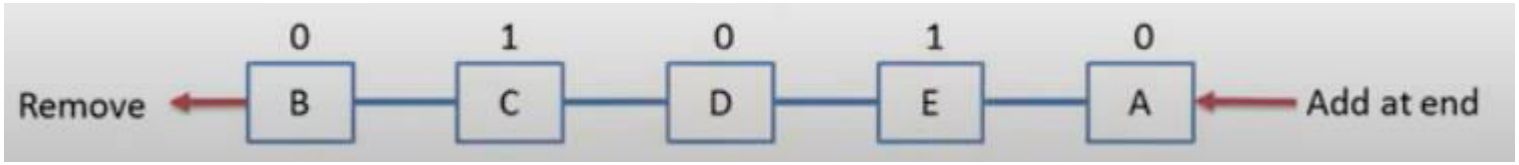
- Thus, the page given second chance will not be replaced until all other page will be replaced or given the second chance.

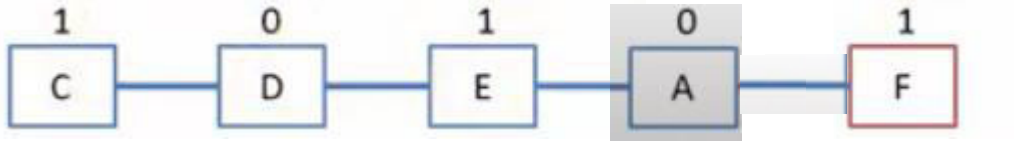


Second Chance Algorithm (Clock Algorithm):



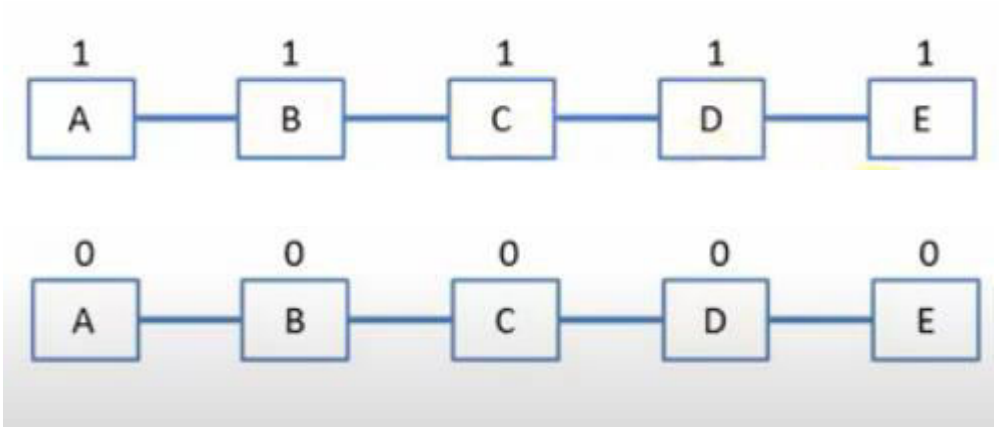




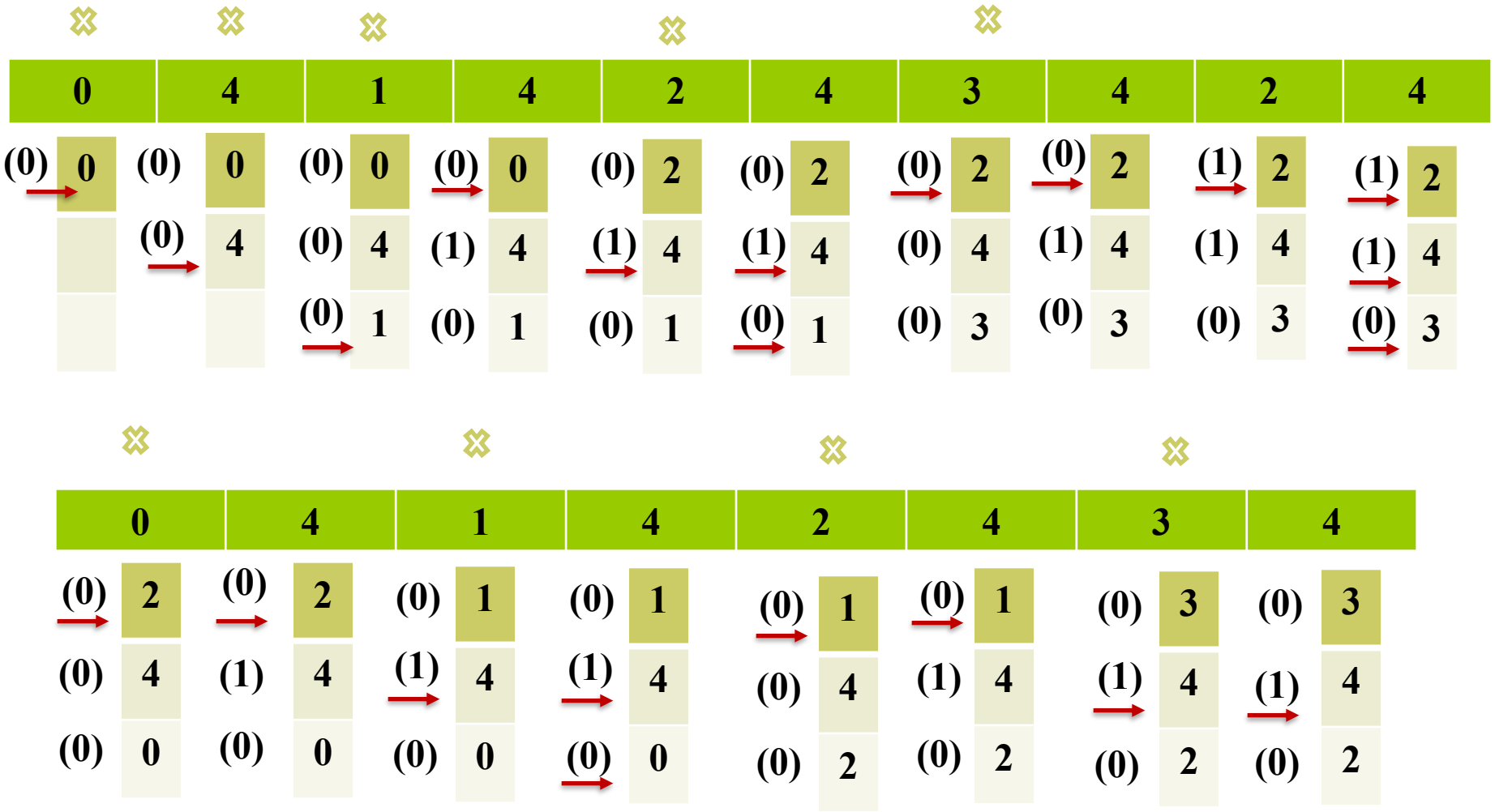


Second Chance Algorithm (Clock Algorithm):

If all the pages have their referenced bit set, on the second encounter of the first page in the list, that page will be swapped out, as it now has its referenced bit cleared.



Second Chance Algorithm (Clock Algorithm):

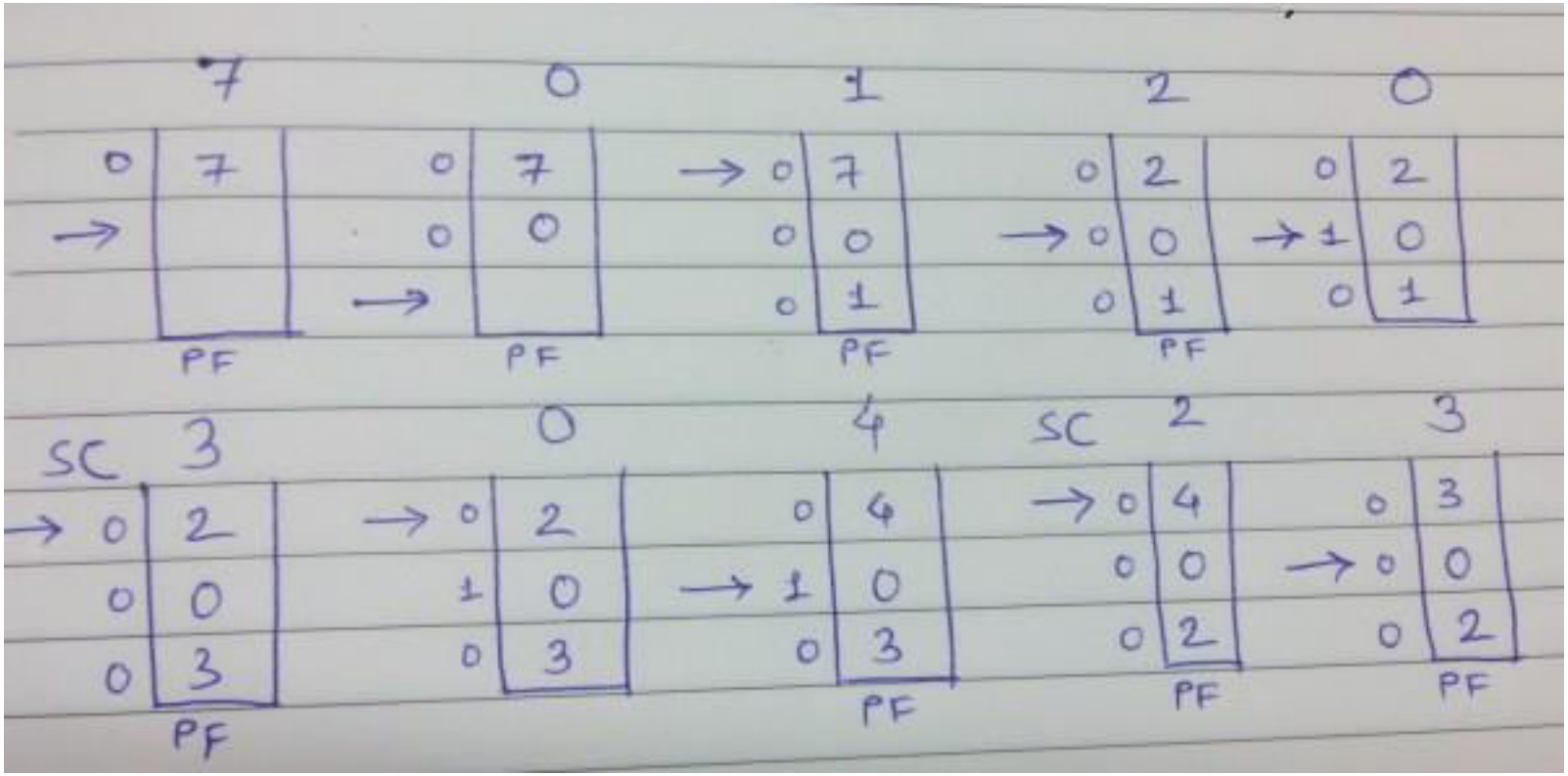


Page Fault = 9

Second Chance Algorithm (Clock Algorithm)

Assume that the system has 3 page frames. Consider the following page reference stream in the given order. 7, 0, 1, 2, 0, 3, 0, 4, 2, 3

The number of page faults occur using clock algorithm are



Second Chance Algorithm (Clock Algorithm):

Input: 2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1

Free frames: 3

Output: 13

Counting Based Page Replacement

There are several algorithms based on counting the number of references that have been made to a given page, such as:

- ***Least Frequently Used, LFU:*** *Replace the page with the* ***lowest reference count****. A counter is assigned to every page that is loaded into memory.*
- ***Most Frequently Used, MFU:*** *Replace the page with the* ***highest reference count****.*

Counting based page replacement

- *In general counting-based algorithms are not commonly used, as their implementation is expensive and they do not approximate OPT well.*

Least Frequently Used, LFU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	7	7	2	2	2	2	4	4	3	3	3	3	3	3
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	1	2
PF	PF	PF	PF	NPF	PF	NPF	PF	PF	PF	NPF	NPF	NPF	PF	PF

Frequency:

7 = 1

7 = 0

0 = 1

0 = 2

0 = 3

0 = 4

1 = 1

1 = 0

1 = 1

1 = 0

2 = 1

2 = 0

2 = 1

2 = 2

2 = 0

2 = 1

3 = 1

3 = 0

3 = 1

3 = 2

4 = 1

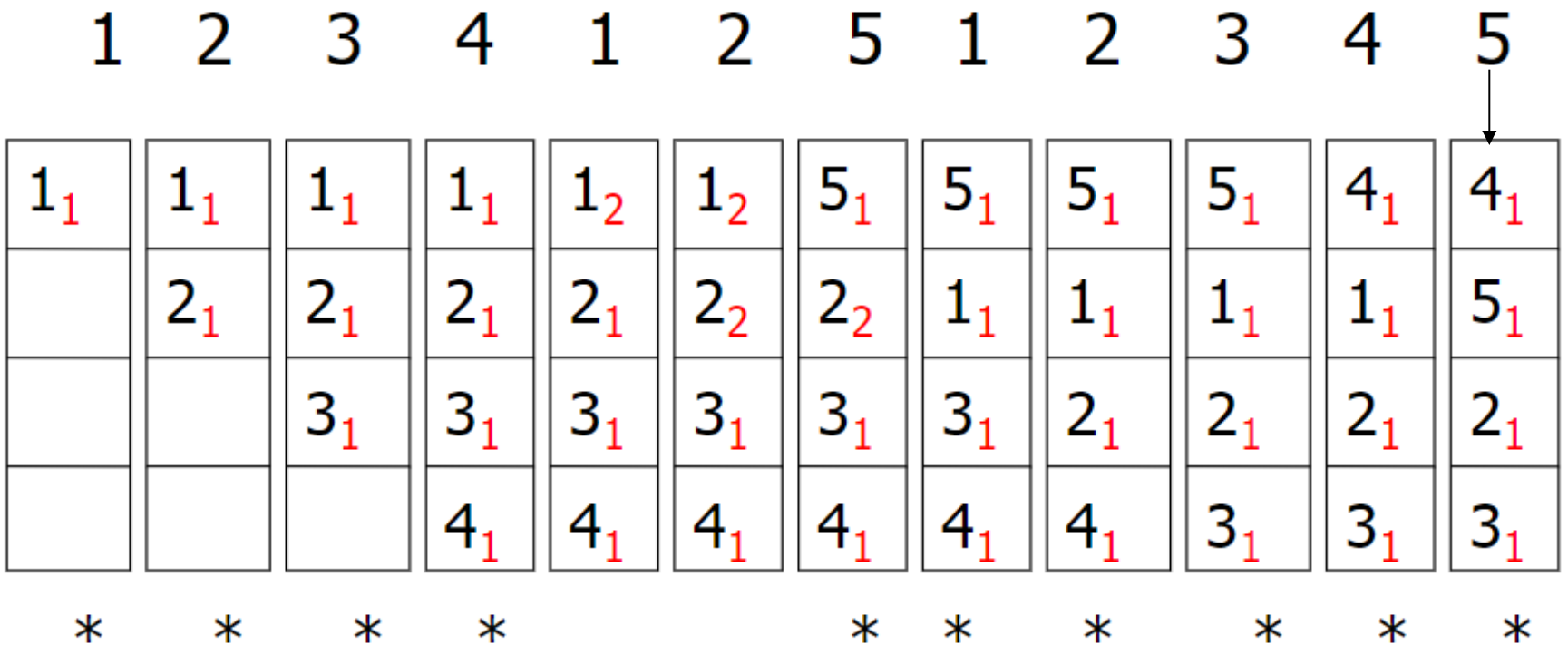
4 = 0

Total page fault =10

Most Frequently Used

- ***Most Frequently Used, MFU:***
- Replace the page with the ***highest reference count***.
- The reason for this selection is that the page with the smallest count was probably just brought in and has yet to be used.

Most Frequently Used



- *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*

Total page fault = 10

Allocation of Frames

- ***Problem :*** *How do we allocate the fixed amount of free memory among the various processes.*
- ***Minimum Number of Frames:***
- *To increase performance* ***allocate at least minimum number of frames*** *to hold all the different pages that any single instruction can reference.*
- *The absolute minimum number of frames that a process must be allocated is dependent on system architecture, and corresponds to the worst-case scenario of the number of pages that could be touched by a single (machine) instruction.*
- *If an instruction (and its operands) spans a page boundary, then multiple pages could be needed just for the instruction fetch.*

Allocation Algorithms

- ***Equal allocation:*** *The easiest way to split m frames among n processes is to give everyone an equal share, m/n frames. This is known as equal allocation.*
- *e.g., if 100 frames and 5 processes, give each 20 pages.*
- *If there are 103 frames and 5 processes, then each process may be given 20 pages and the remaining 3 frames may be added to the pool of free frames.*

Disadvantage:

- *Due to the processes varying in size, It doesn't make sense to allocate equal number of frames to all processes.*

eg;. If a small student process of 10 KB and an interactive database of 127 KB are the only two processes running in a system with 62 free frames, it does not make much sense to give each process 31 frames. The student process does not need more than 10 frames, so the other 21 are, strictly speaking, wasted.

Allocation Algorithms

Proportional allocation:

- *Here, it allocates available memory to each process according to its size.*

If the size of the virtual memory for process $p_i = S_i$

Then define $S = \sum S_i$

if the total number of available frames is = m,

we allocate a_i *frames to process* p_i, *where* a_i *is approximately* $a_i = (S_i / S) * m$.

$$m = 64$$
$$s_i = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Allocation Algorithms

- ***Global Versus Local Allocation:***

- *We can classify page-replacement algorithms into two broad categories:* ***global replacement*** *and* ***local replacement.***

- ***Global replacement*** *allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; one process can take a frame from another.*

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

- ***Ex:*** *High priority process may select a frame for replacement from a process with lower priority number.*

Thrashing

- *If the process does not have the number of frames it needs to support pages in active use, it will **quickly page-fault**.*
- *At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it **quickly faults again, and again**, and again, replacing pages that it must back in immediately.*
- *This high paging activity is called **"Thrashing".** A process is thrashing if it is spending more time paging than executing.*

A process is thrashing if it is spending more time paging than executing.

Cause of Thrashing

- *A process is thrashing if it is spending more time in paging than executing. This leads to: low CPU utilization and the operating system thinks that it needs to increase the degree of multiprogramming.*

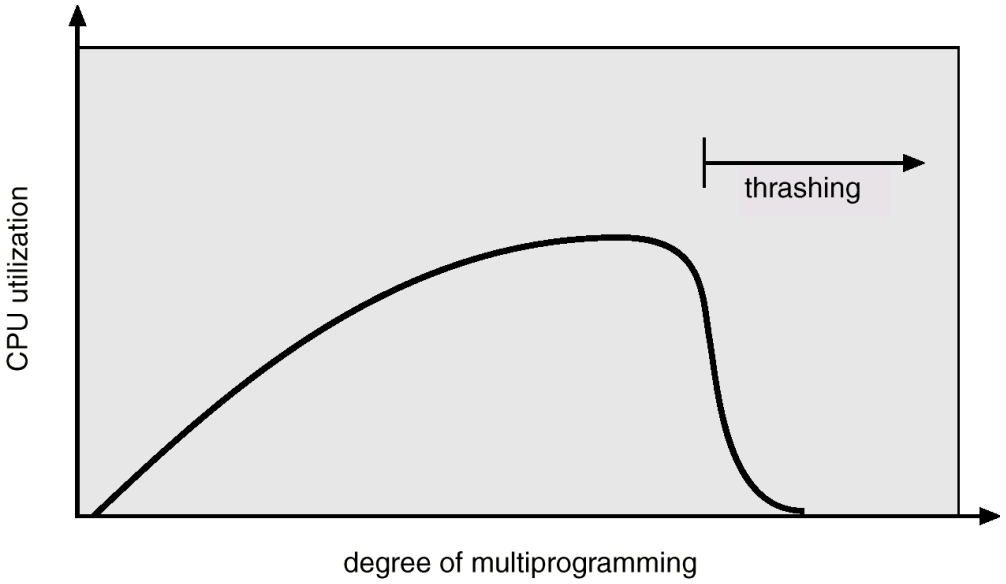


Figure shows that the CPU utilization increases with the increase in the degree of multiprogramming to a certain extent. If the degree of multiprogramming is increased still further, it results in the decrease of CPU utilization. This worsens the thrashing.

Cause of Thrashing

- *A thrashing process can cause other processes to thrash if a **global page replacement strategy** is used.*
- ***Local page replacement policies** can prevent one thrashing process from taking pages away from other processes, but it still tends to **clog up the I/O queue**, thereby slowing down any other process that needs to do even a little bit of paging (or any other I/O for that matter.)*
- *To prevent thrashing we must provide processes with as many frames as they really need "right now**", but how do we know what that is**?*

Locality model

- *It refers to the tendency of a program to access certain memory locations more frequently than others within a particular time frame.*
- *The locality model states that as a process executes it moves from locality to locality.*
- *The locality of a process refers to the set of pages that are used together by a process at a particular time.*
- *The minimum number of **frames to be allocated** for a process may be decided based on the **number of pages in the locality** of the process at a particular time.*

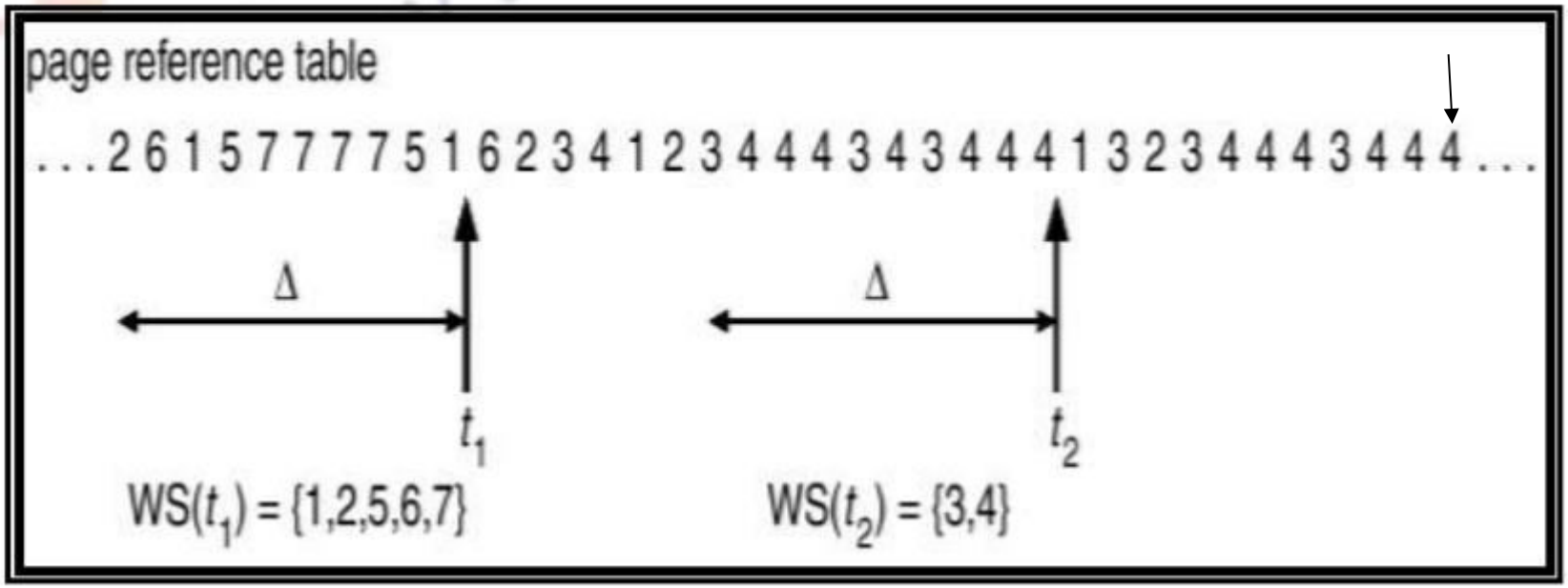
Locality model

- *We now see two methods used to avoid thrashing –*
- ***Working set model***
- ***Page-fault frequency scheme***.

Working-Set Model

- *The working set model is based on the concept of **locality,** and defines a **working set window**, of length delta Δ (size of working set, size of locality) it is a fixed number of page references.*
- *The **working set** refers to the set of pages in the most recent Δ page references.*
- *If a page of a process is active, then that page is present in the working set of that process. If a page is not used for some time, then that page will be dropped from the working set.*

Working-Set Model



Let ***Δ = 10*** memory references. At time t_1, the working set comprises of the (unique) pages 1,2,5,6 and 7. The working set size at t_1 is 5. At time t_2, the working set has changed and it comprises of the pages 3 and 4. The working set size at t_2 is 2.

Working-Set Model

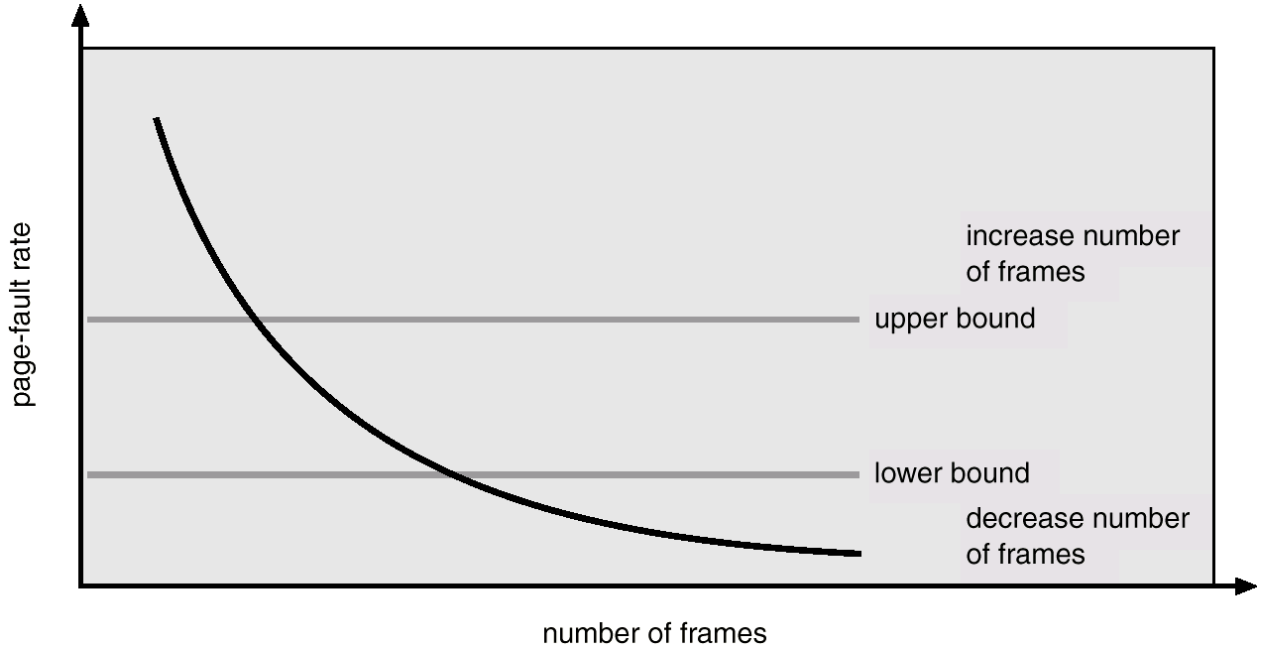
- The accuracy of the working set depends on the selection of Δ. If Δ is too small, the working set will not encompass the entire locality. If Δ is too large, the working set will encompass several localities. If $\Delta = \infty$, the working set will encompass the entire program.
- Let WSS_i denote the working set size of process P_i.
- Let $D = \Sigma\ WSS_i$ the total demand of all the processes. If $D > m$, that is, if the sum of the working set sizes of all the processes becomes greater than the total number of frames ***m***, ***thrashing occurs***. Therefore, if D > m, then one of the processes must be suspended.

Page-Fault Frequency Scheme

- *If the **page-fault** rate is **too low**, then it means that the process has more frames than needed and hence, it must lose frames.*
- *If the **page-fault rate too high**, it means that the process has fewer frames than needed and the process must gain frames.*
- *Figure shows how the page-fault rate varies with the increase in the number of frames. An **upper bound and a lower bound are maintained**. If the page-fault rate goes beyond the upper bound, the number of frames allocated to each process is increased. If the page-fault rate goes below the lower bound, the number of frames allocated to each process is decreased.*

Page-Fault Frequency Scheme

- ***OS monitors page-fault rate*** of each process to decide if it has ***too many*** or ***not enough*** page frames allocated



- Additional, free frames could be allocated from the free-frame list or removed from processes that have too many pages. If the free-frame list is empty and no process has any free frames, then a process might be swapped out.