Unit 5: Virtual Memory

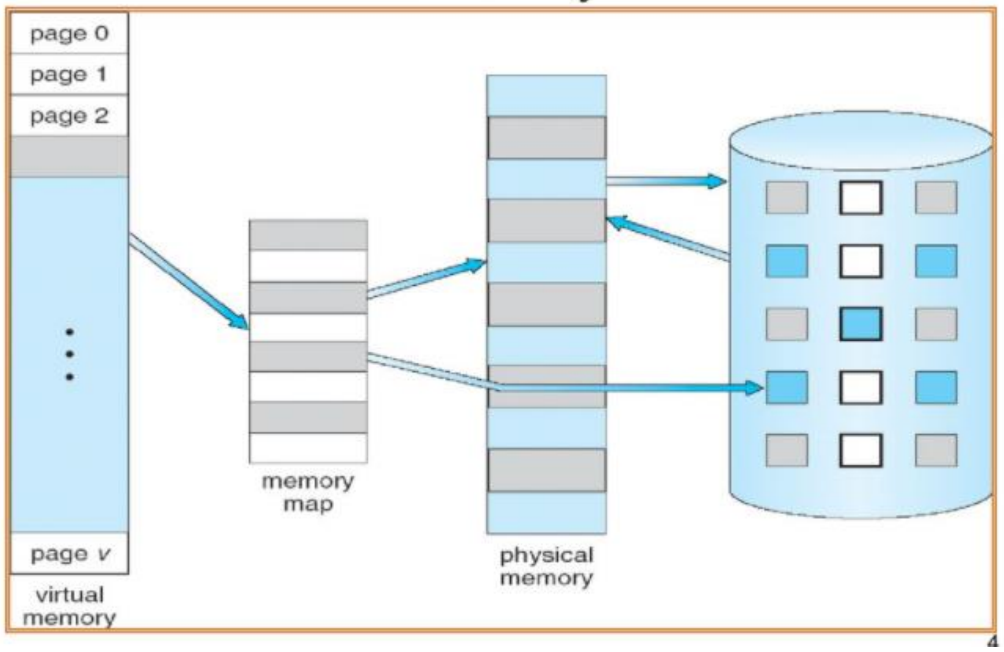
INTRODUCTION

- A virtual memory is a technique that allows a process to execute even though it is partially loaded in memory.
- A computer can address more memory than the amount physically installed on the system. This extra memory is called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.
- Virtual memory removes the requirement that an entire process should be in main memory for its execution.
- The main advantage of this is a process can be larger than that the main memory.
- Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available (Figure)

Following are the situations when entire program is not required to load fully.

1. User written error handling routines are used only when an error occurs in the data or computation.
2. Certain options and features of a program may be used rarely.
3. Many tables are assigned a fixed amount of address space even though only a small amount of the table is used.

Virtual Memory That is Larger Than Physical Memory



- The ability to execute a program that is only partially in memory would counter many benefits.

1. Less number of I/O would be needed to load or swap each user program into memory.
2. A program would no longer be constrained by the amount of physical memory that is available.
3. Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Motivations behind this technique:

For real time programs, it is observed that the entire program is not needed to be in memory for its execution. Consider the following observation.

1. Programs contain coded to handle unusual error conditions. Such error occur very rarely, and so such codes execute in rare situations.
2. Generally, arrays are allocated more memory than the actual requirement.
3. Programs contain certain operations and features which are used very rarely.
   So, such types of code and data part need not to be in memory all the time. Also even in cases where the entire program in needed, it may not be in memory
   These all-factors motivations the technique of Virtual Memory.

Advantages:

There are two main advantages of virtual memory.

1. Programs (and so processes) are not constrained by physical memory size. A process larger than the memory can also execute.
2. Degree of multiprogramming can be varied over a large range. As there is no need to keep an entire process in memory, more and more processes can be accommodated in memory.

Implementation:

The Virtual memory can be implemented in one of the following three ways.

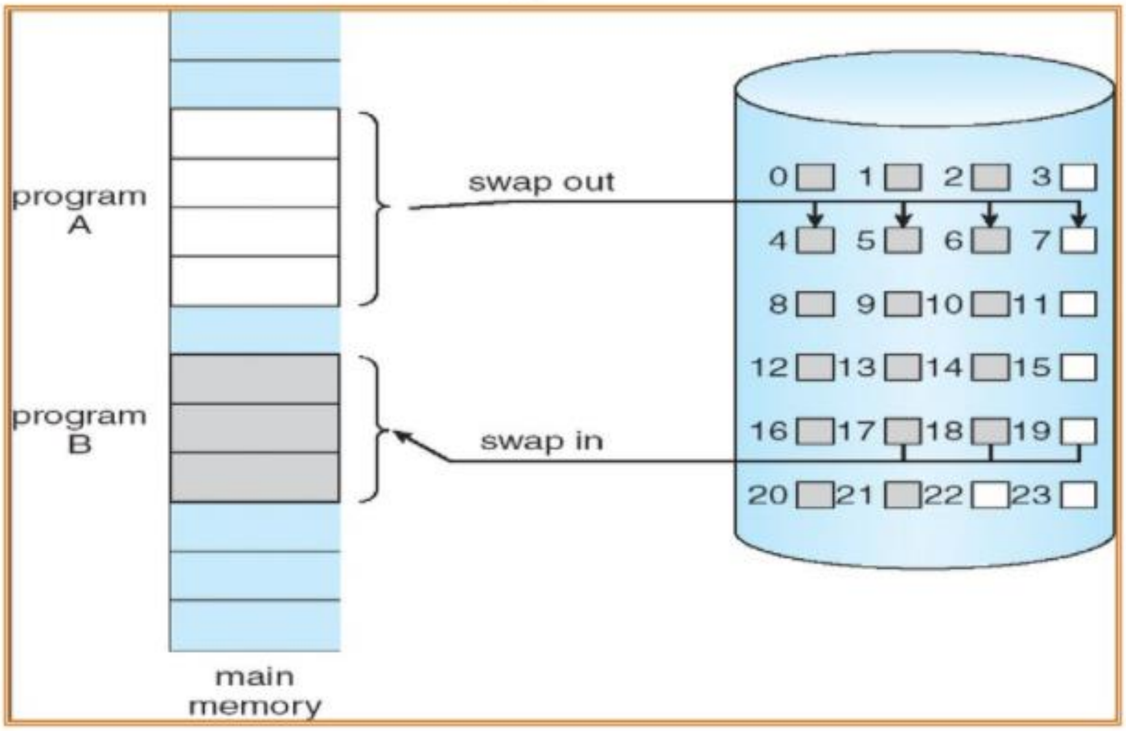
1. **Demand Paging**
2. **Demand Segmentation**
3. **Segmentation with Paging.**

5.1 DEMAND PAGING.

- Demand paging system is like a paging system with swapping processes reside on 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.

- A *lazy swapper* never swaps a page into memory unless that page will be needed.
- A *swapper* manipulates entire processes, whereas a *pager* is concerned with the individual pages of a process.
- The term pager is use frequently rather than swapper, in connection with demand paging.

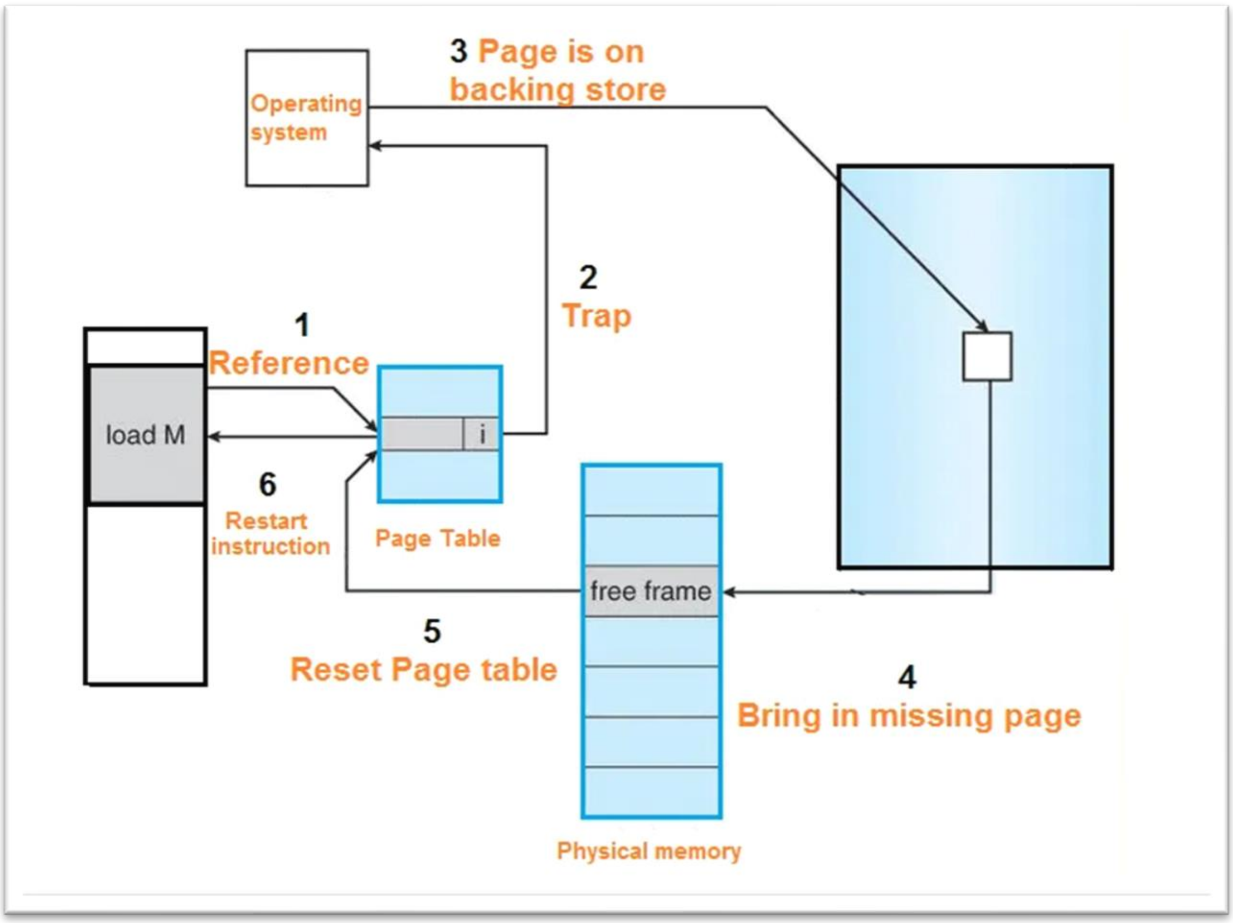
Transfer of a Paged Memory to Contiguous Disk Space



Valid and Invalid Bit

- With each page table entry, a valid-invalid bit is associated.
  (V = in memory, i= not- in- memory)
- Initially valid-invalid bit is set to *i* on all entries.
- While the process executes and accesses pages that are ***memory resident*** , execution proceeds normally.
- During address translation if valid-invalid bit in page table entry is ***i=page fault.***

Steps to Handle Page Fault:



1. Whenever any logical address is generated, page table is searched. If page validity bit is set to invalid, it means that required page is not available in memory. This is called page fault.
2. Operating system is requested to bring this page into memory from disk.
3. Operating system determines the location of that page on the disk.
4. That page is brought into free frame in memory. If free frame is not available, some replacement algorithm is used to replace an occupied frame.
5. Frame number is entered in the page table entry, and validity bit is set to valid.
6. Instruction referenced by that logical address is restarted.

This is how a virtual memory works. Today, most modern operating systems come up with support to virtual memory.

Advantages/ Benefits of Demand Paging:

- Less I/O needed
- Less memory needed
- Faster response
- More users

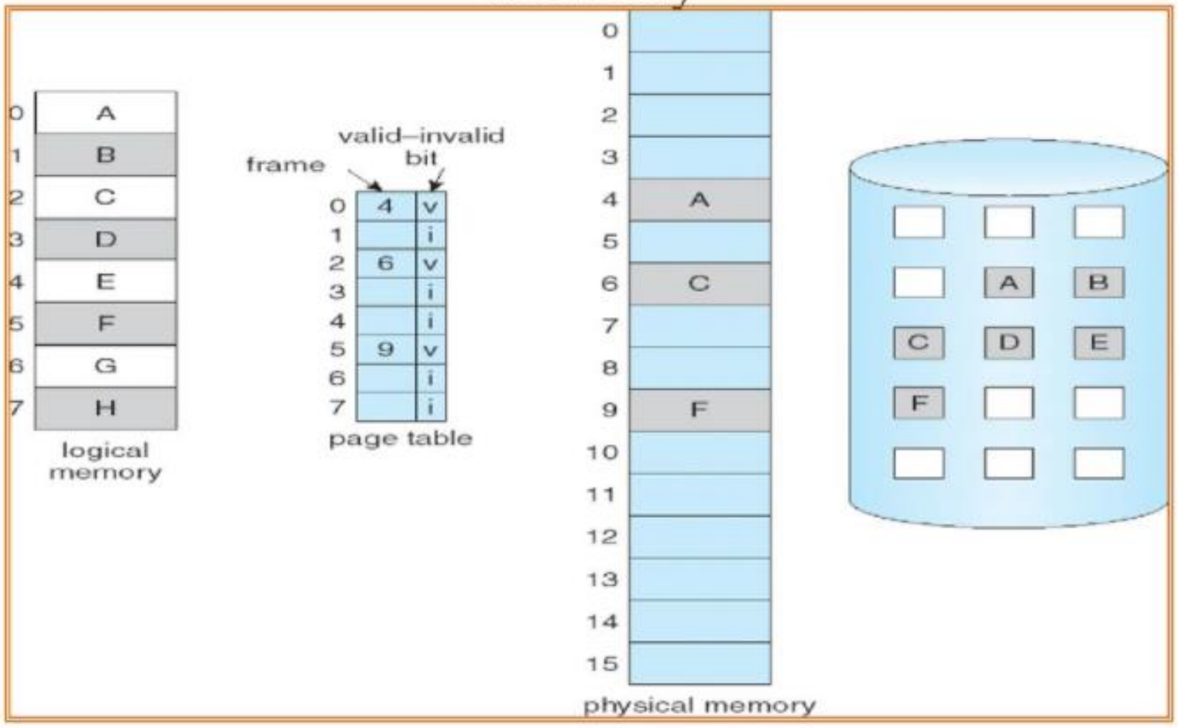
Disadvantages of Demand Paging:

- Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of explicit constraints on a jobs address space size.

Valid and Invalid Bit

- With each page table entry a valid-invalid bit is associated.
  (V = in memory, i= not- in- memory)
- Initially valid-invalid bit is set to *i* on all entries.

Page Table When Some Pages Are Not in Main Memory



- While the process executes and accesses pages that are ***memory resident*** , execution proceeds normally.
- During address translation if valid-invalid bit in page table entry is ***i=page fault.***

Page Fault

1. If there is a reference to a page, first reference to that page will trap to operating system:

page fault steps

1. Operating system looks at another table to decide:
   Invalid reference ⇒ abort
   Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = v
6. Restart the instruction that caused the page fault

What happens if there is no free frame?

Page replacement

- find some page in memory, but not really in use, swap it out.
- Apply algorithm
- Check performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

5.2 PAGE REPLACEMENT ALGORITHM

- There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults. The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data.
- Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with the required page.

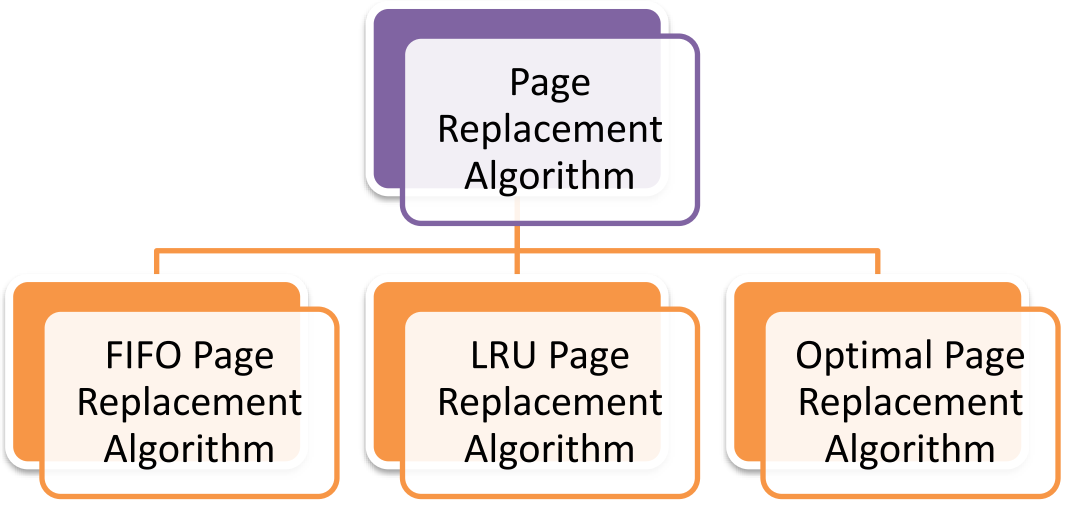
Page replacement is required when-

- All the frames of main memory are already occupied.
- Thus, a page has to be replaced to create a room for the required page.

Page Replacement Algorithms-

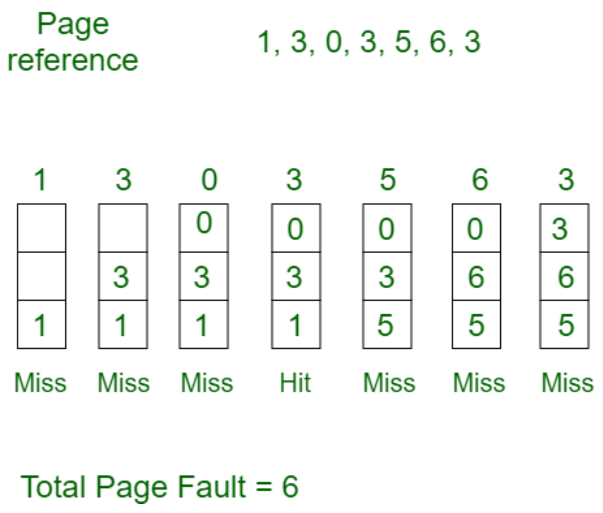
- Page replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.

Various page replacement algorithms are-

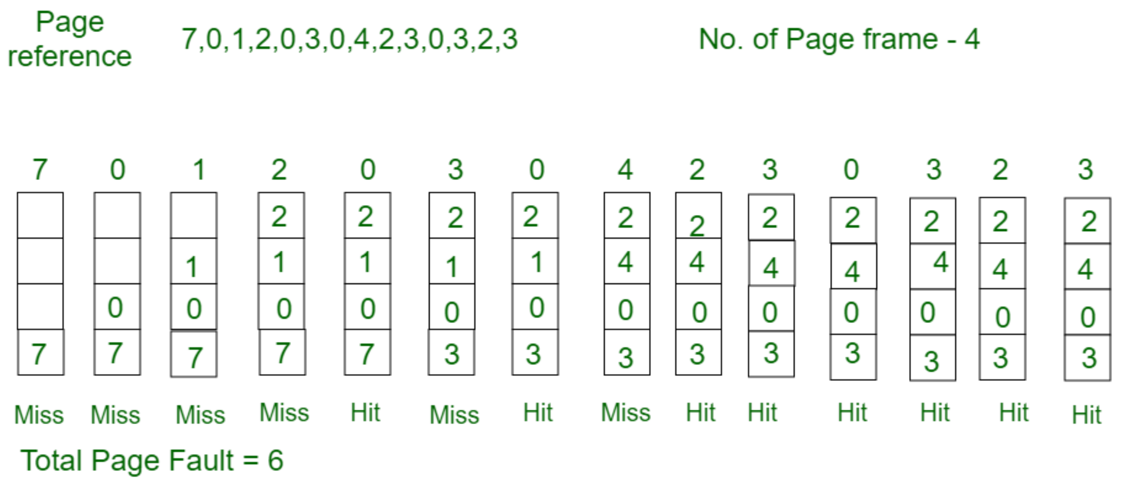

Page Replacement Algorithm
FIFO Page Replacement Algorithm
LRU Page Replacement Algorithm
Optimal Page Replacement Algorithm

A good page replacement algorithm is one that minimizes the number of page faults.

- **FIFO Page Replacement Algorithm**
- 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.

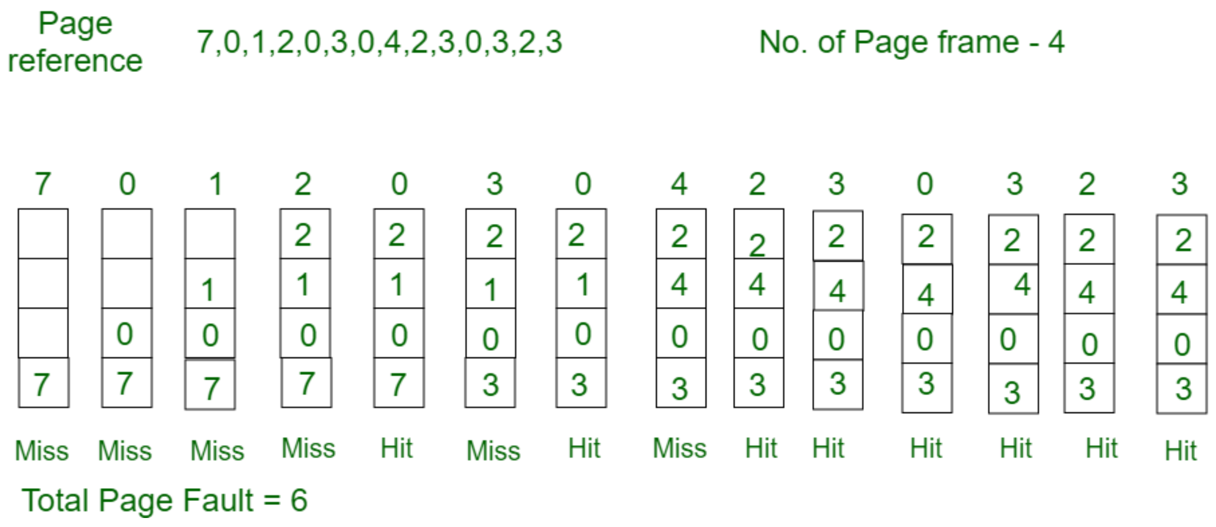


- **LRU Page Replacement Algorithm**
- As the name suggests, this algorithm works on the principle of "Least Recently Used".
- It replaces the page that has not been referred by the CPU for the longest time.



Here LRU has same number of page fault as optimal but it may differ according to question.

- **Optimal Page Replacement Algorithm**
- 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 cannot 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.



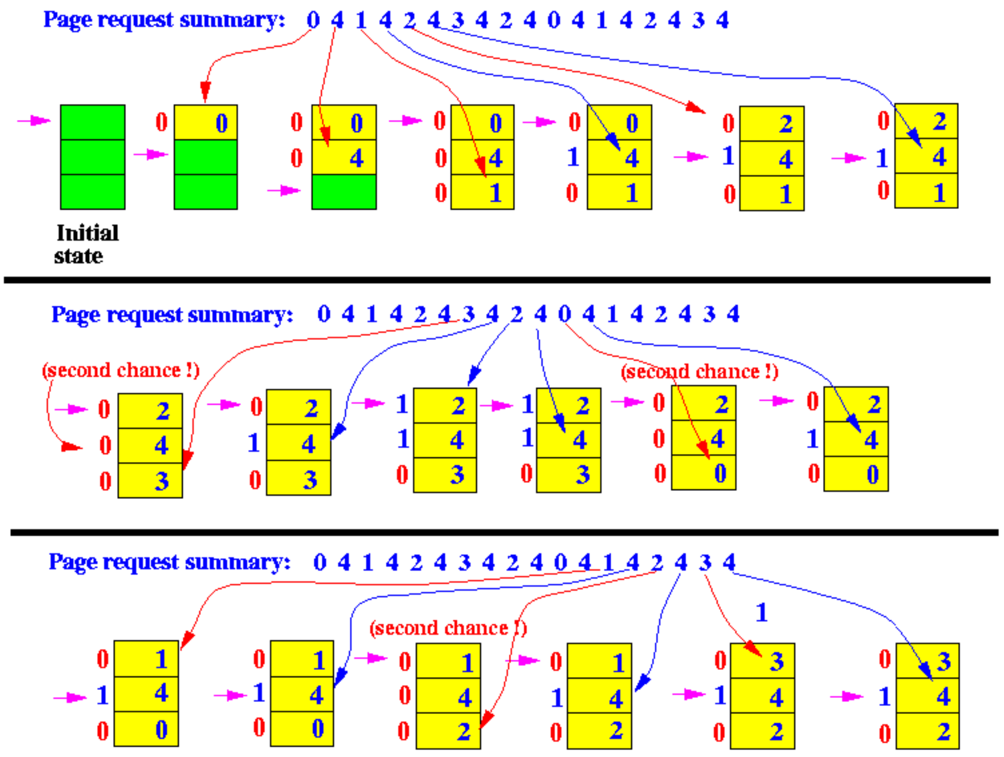
4. **Second chance Page Replacement Algorithm OR Clock Replacement Page Replacement Algorithm**

- In some books, the **Second Chance** replacement policy is called the **Clock** replacement policy...
- In the Second Chance page replacement policy, the candidate pages for removal are **consider** in a round robin matter, and a page that has been **accessed between consecutive considerations** will not be replaced.

The page replaced is the one that - considered in a round robin matter - has not been accessed since its last consideration.

Implementation:

- Add a "second chance" bit to each memory frame.
- Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
- A new page read into a memory frame has the second chance bit set to ZERO (0)
- IHJ0When you need to find a page for removal, look in a round robin manner in the memory frames:
  i. If the second chance bit is ONE, reset its second chance bit (to ZERO)



Page request summary: 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4
Initial state
Page request summary: 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4
(second chance !)
(second chance !)
Page request summary: 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4
(second chance !)

  and continue.
  ii. If the second chance bit is ZERO, replace the page in that memory frame.

The following figure shows the behaviour of the program in paging using the Second Chance page replacement policy:

- We can see notably that the **bad** replacement decisions made by FIFO **is not present** in Second chance !!!

- There are a total of **9 page read operations** to satisfy the total of 18 page requests - just as good as the more computationally expensive LRU method !!!
- Note: in this example, the Second Chance method resulted in the same number of page faults as LRU. That is **not true in general** !

5.3 ALLOCATION OF FRAMES

An important aspect of operating systems, virtual memory is implemented using demand paging. Demand paging necessitates the development of a page-replacement algorithm and a frame allocation algorithm. Frame allocation algorithms are used if you have multiple processes; it helps decide how many frames to allocate to each process.

There are various constraints to the strategies for the allocation of frames:

1. *You cannot allocate more than the total number of available frames.*
2. At least a minimum number of frames should be allocated to each process.

This constraint is supported by two reasons. The first reason is, as less number of frames are allocated, there is an increase in the page fault ratio, decreasing the performance of the execution of the process. Secondly, there should be enough frames to hold all the different pages that any single instruction can reference.

Frame allocation algorithms –

The three algorithms commonly used to allocate frames to a process are:

1. **Global replacement allocation**
2. **Local replacement allocation**
3. **Equal allocation**
4. **Proportional allocation**
5. **Priority allocation**

1. **Global replacement allocation**

The global replacement allocation takes care of the frame that is being allocated in pages. The global replacement algorithm tells us that the process having a low priority can give frames to the process with higher priority so that fewer number of page faults occur.

1. **Local replacement allocation**

As the global replacement allocation tells us that the frames can be stored in any priority process the local replacement allocation tells us that the frames of the pages will be stored on the same page, unlike global. This also does not influence the behaviour of the process behaviour as it did in the global replacement allocation.

2. **Equal allocation**

- Each process gets the same number of frames. Divide the frames equally between the processes.

- **Disadvantage:** In systems with processes of varying sizes, it does not make much sense to give each process equal frames. Allocation of a large number of frames to a small process will eventually lead to the wastage of a large number of allocated unused frames.

3. **Proportional allocation**
    - Allocate frames according to size of process.

    Frames are allocated to each process according to the process size. For a process pi of size si, the number of allocated frames is ai = (si/S)*m, where S is the sum of the sizes of all the processes and m is the number of frames in the system. For instance, in a system with 62 frames, if there is a process of 10KB and another process of 127KB, then the first process will be allocated (10/137)*62 = 4 frames and the other process will get (127/137)*62 = 57 frames.

    Advantage: All the processes share the available frames according to their needs, rather than equally.

4. **Priority allocation**
    - Higher priority processes get more frames.
    - If the process is of high priority and has more number of frames the process will be allotted that many frames in the main memory and the process with low priority will be allocated next. The frame allocation can occur based on both priority and the size of the frames required by the process.

Global versus Local Frame Allocation

- **Global replacement**
    - Allows a process to select a victim frame from the set of all frames.

- **Local replacement**
    - A victim frame is selected from the set of frames allocated to this process.

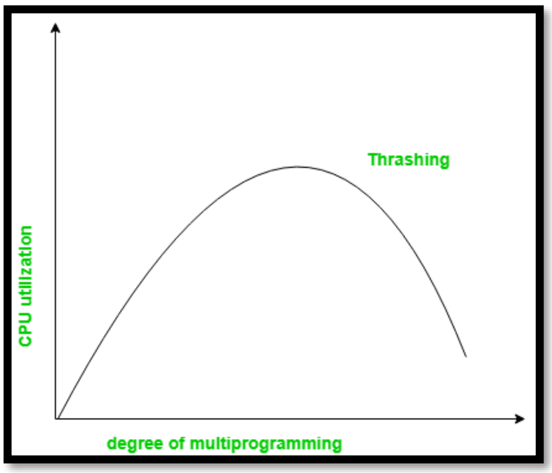
- **Global replacement is more efficient overall and therefore is commonly used. Also easier to implement.**

- **Local replacement prevents external influences on the page fault rate of this process as long as the same number of frames is allocated.**

5.4 THRASHING

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.

The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilisation would fall drastically. The long-term scheduler would then try to improve the CPU utilisation by loading some more processes into the memory thereby increasing the degree of multiprogramming. This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing.



5.4.1 Locality Reference

Locality of Reference refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by loops and subroutine calls in a program.

It is based on the principle of locality of reference. There are two ways with which data or instruction is fetched from main memory and get stored in cache memory. These two ways are the following:

1. **Temporal locality**

   Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.

2. **Spatial Locality**

   Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.

Belay's Anomaly

The rate of page faults varies directly with the number of frames allocated to the individual process. The increase in the number of frames considerably decreases the number of page faults. However, sometimes reverse action occurs when the increased number of frames results in increased page faults. This exception is known as the Belady's Anomaly. The occurrence of this exception depends on the page replacement algorithm, which governs the demand paging process. The page replacement algorithms in which Belady's Anomaly occurs the most includes:

1. **First In First Out (FIFO)**
2. Second Chance Algorithm
3. Random Page Replacement Algorithm

Belady's Anomaly in FIFO –
Assuming a system that has no pages loaded in the memory and uses the FIFO Page replacement algorithm.

Consider the following reference string: **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. The diagram below illustrates the pattern of the page faults occurring in the example.

1	1	1	2	3	4	1	1	1	2	5	5
	2	2	3	4	1	2	2	2	5	3	3
		3	4	1	2	5	5	5	3	4	4
PF	PF	PF	PF	PF	PF	PF	X	X	PF	PF	X

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. The diagram below illustrates the pattern of the page faults occurring in the example.

1	1	1	1	1	1	2	3	4	5	1	2
	2	2	2	2	2	3	4	5	1	2	3
		3	3	3	3	4	5	1	2	3	4
			4	4	4	5	1	2	3	4	5
PF	PF	PF	PF	X	X	PF	PF	PF	PF	PF	PF

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.

Note – It is not necessary that every string reference pattern cause Belady anomaly in FIFO but there are certain kind of string references that worsen the FIFO performance on increasing the number of frames.