

UNIT IV

Memory Management: Logical and Physical address map, Memory allocation, Internal and External fragmentation and Compaction, Paging.

Virtual Memory: Demand paging, Page Replacement policies.

Background:

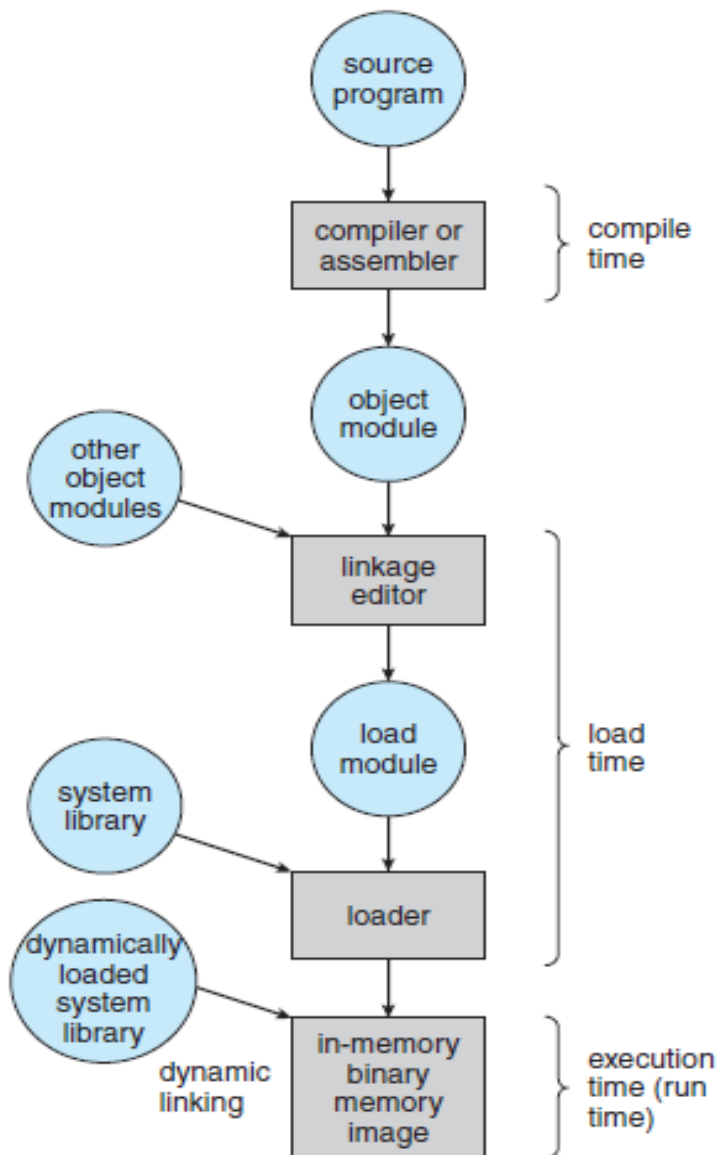
- Memory management is concerned with managing the primary memory.
- Memory consists of array of bytes or words each with their own address.
- The instructions are fetched from the memory by the CPU based on the value program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- Memory unit sees only a stream of memory addresses. It does not know how they are generated.
- Program must be brought into memory and placed within a process for it to be run.
- **Input queue** – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being run.

Functions of memory management:-

- Keeping track of status of each memory location.
- Determining the allocation policy - Memory allocation technique & De-allocation technique.

Address Binding:-

- Programs are stored on the secondary storage disks as binary executable files.
- When the programs are to be executed they are brought in to the main memory and placed within a process.
- The collection of processes on the disk waiting to enter the main memory forms the input queue.
- One of the processes which are to be executed is fetched from the queue and placed in the main memory.
- During the execution it fetches instruction and data from main memory. After the process terminates it returns back the memory space.
- During execution the process will go through different steps and in each step the address is represented in different ways.
- In source program the address is symbolic.
- The compiler converts the symbolic address to re-locatable address.
- The loader will convert this re-locatable address to absolute address.



Multistep processing of a user program.

Binding of instructions and data can be done at any step along the way:-

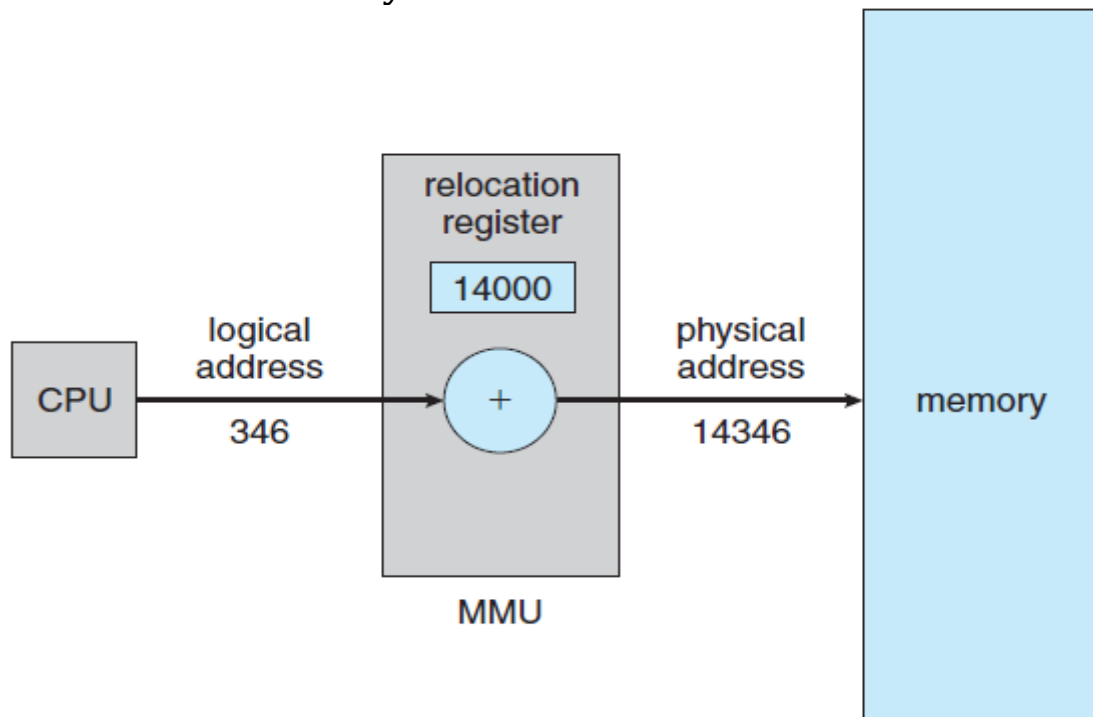
1. **Compile time:-** If we know whether the process resides in memory then absolute code can be generated. If the static address changes then it is necessary to re-compile the code from the beginning.
2. **Load time:-** If the compiler doesn't know whether the process resides in memory then it generates the re-locatable code. In this the binding is delayed until the load time.
3. **Execution time:-** If the process is moved during its execution from one memory segment to another then the binding is delayed until run time. Special hardware is used for this. Most of the general purpose operating system uses this method.

Logical and Physical address map

- The address generated by the CPU is called **logical** address or **virtual** address.

- The address seen by the memory unit i.e., the one loaded in to the memory register is called the **physical** address.
- Compile time and load time address binding methods generate some logical and physical address. The execution time address binding generate different logical and physical address.
- Set of logical address space generated by the programs is the **logical address space**.
- Set of physical address corresponding to these logical addresses is the **physical address space**.
- The mapping of virtual address to physical address during run time is done by the hardware device called memory management unit (MMU).
- The base register is also called re-location register. Value of the re-location register is added to every address generated by the user process at the time it is sent to memory.
- User program never sees the real physical address.

The figure below shows the dynamic relation. Dynamic relation implies mapping from virtual address space to physical address space and is performed at run time usually with some hardware assistance.



Dynamic relocation using a relocation register.

Relocation is performed by the hardware and is invisible to the user. Dynamic relocation makes it possible to move a partially executed process from one area of memory to another without affecting.

Dynamic re-location using a re-location registers The above figure shows that dynamic re-location which implies mapping from virtual

addresses space to physical address space and is performed by the hardware at run time.

Re-location is performed by the hardware and is invisible to the user dynamic relocation makes it possible to move a partially executed process from one area of memory to another without affecting.

Example:

- If the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location 14000; an access to location 346 is mapped to location 14346.
- The user program never sees the real physical addresses.
- The program can create a pointer to location 346, store it in memory, manipulate it, and compare it with other addresses—all as the number 346.
- Only when it is used as a memory address (in an indirect load or store, perhaps) is it relocated relative to the base register.
- The user program deals with logical addresses.
- The memory-mapping hardware converts logical addresses into physical addresses.
- We now have two different types of addresses: logical addresses (in the range 0 to max) and physical addresses (in the range $R + 0$ to $R + max$ for a base value R).
- The user program generates only logical addresses and thinks that the process runs in locations 0 to max . However, these logical addresses must be mapped to physical addresses before they are used.
- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.

Dynamic Loading:-

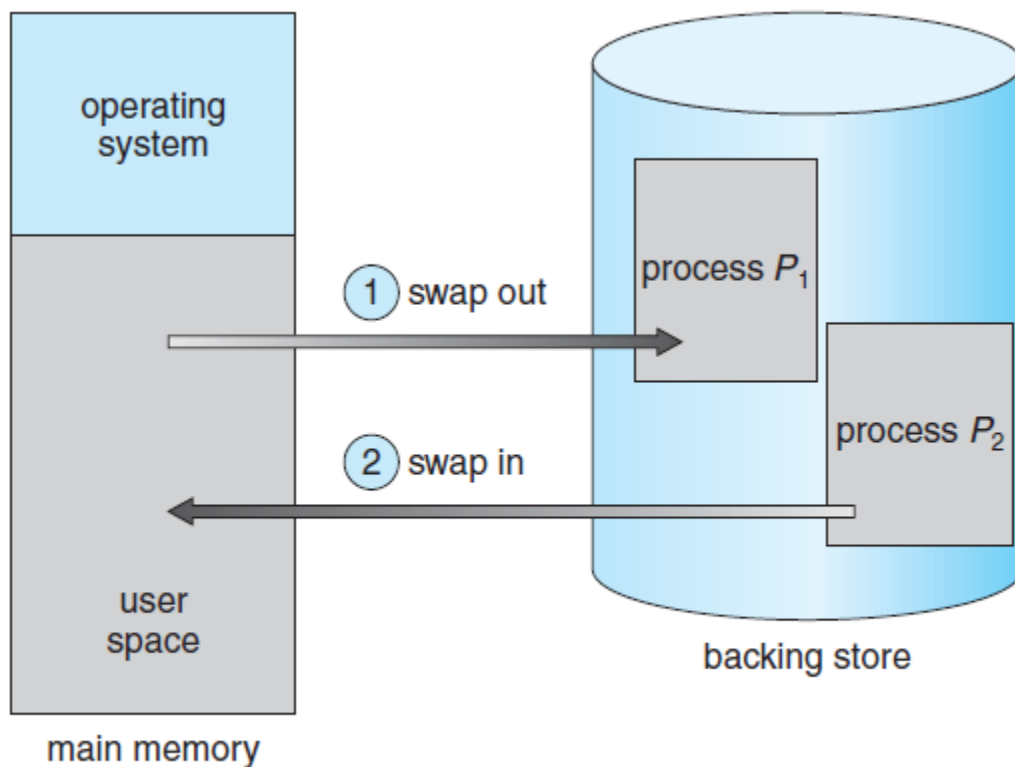
- For a process to be executed it should be loaded in to the physical memory. The size of the process is limited to the size of the physical memory.
- Dynamic loading is used to obtain better memory utilization.
- In dynamic loading the routine or procedure will not be loaded until it is called.
- Whenever a routine is called, the calling routine first checks whether the called routine is already loaded or not. If it is not loaded it cause the loader to load the desired program in to the memory and updates the programs address table to indicate the change and control is passed to newly called routine.

Advantage:-

- Gives better memory utilization.
- Unused routine is never loaded.
- Do not need special operating system support.
- This method is useful when large amount of codes are needed to handle in frequently occurring cases.

Swapping:

Swapping is a technique of temporarily removing inactive programs from the memory of the system. A process can be swapped temporarily out of the memory to a backing store and then brought back in to the memory for continuing the execution. This process is called swapping.



Swapping of two processes using a disk as a backing store.

Example:-

- In a multi-programming environment with a round robin CPU scheduling whenever the time quantum expires then the process that has just finished is swapped out and a new process swaps in to the memory for execution.
- A variation of swap is priority based scheduling. When a low priority is executing and if a high priority process arrives then a low priority will be swapped out and high priority is allowed for execution. This process is also called as **Roll out and Roll in**.

- Normally the process which is swapped out will be swapped back to the same memory space that is occupied previously. This depends upon address binding.
- If the binding is done at load time, then the process is moved to same memory location. If the binding is done at run time, then the process is moved to different memory location.
- This is because the physical address is computed during run time.
- Swapping requires backing store and it should be large enough to accommodate the copies of all memory images. The system maintains a ready queue consisting of all the processes whose memory images are on the backing store or in memory that are ready to run.

Swapping is constant by other factors:-

- To swap a process, it should be completely idle.
- A process may be waiting for an I/O operation. If the I/O is asynchronously accessing the user memory for I/O buffers, then the process cannot be swapped.

Sl. No	Logical Address	Physical Address
1	Address generated by the CPU	Address seen by the memory unit
2	The set of all logical addresses generated by a program is a logical address space.	The set of all physical addresses corresponding to these logical addresses is a physical address space.
3	Logical addresses (in the range 0 to max)	Physical addresses (in the range $R + 0$ to $R + \text{max}$ for a base value R)
4	User can view the logical address of a program	User can never view the Physical address of a program
5	User can use the logical address to access the physical address	User can indirectly access physical address but not directly
6	Logical address is variable hence will changing with the system	Physical address of that object always remains constant.

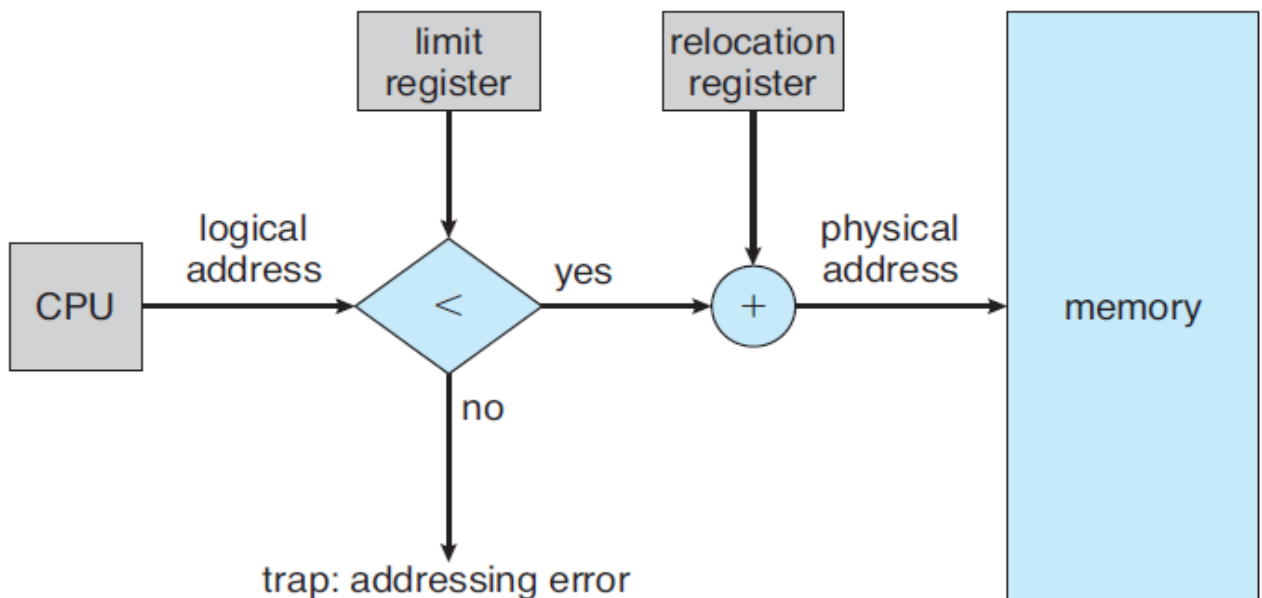
Contiguous Memory Allocation :

- One of the simplest method for memory allocation is to divide memory in to several fixed partition.

- Main memory is usually divided into two partitions:
 - Resident operating system, usually held in low memory with interrupt vector.
 - User processes, held in high memory.
- Each partition contains exactly one process.
- The degree of multi-programming depends on the number of partitions.

Memory mapping and Protection:-

- Memory protection means protecting the OS from user process and protecting process from one another.
- Memory protection is provided by using a re-location register, with a limit register.
- Re-location register contains the values of smallest physical address and limit register contains range of logical addresses. (Re-location = 100040 and limit = 74600).



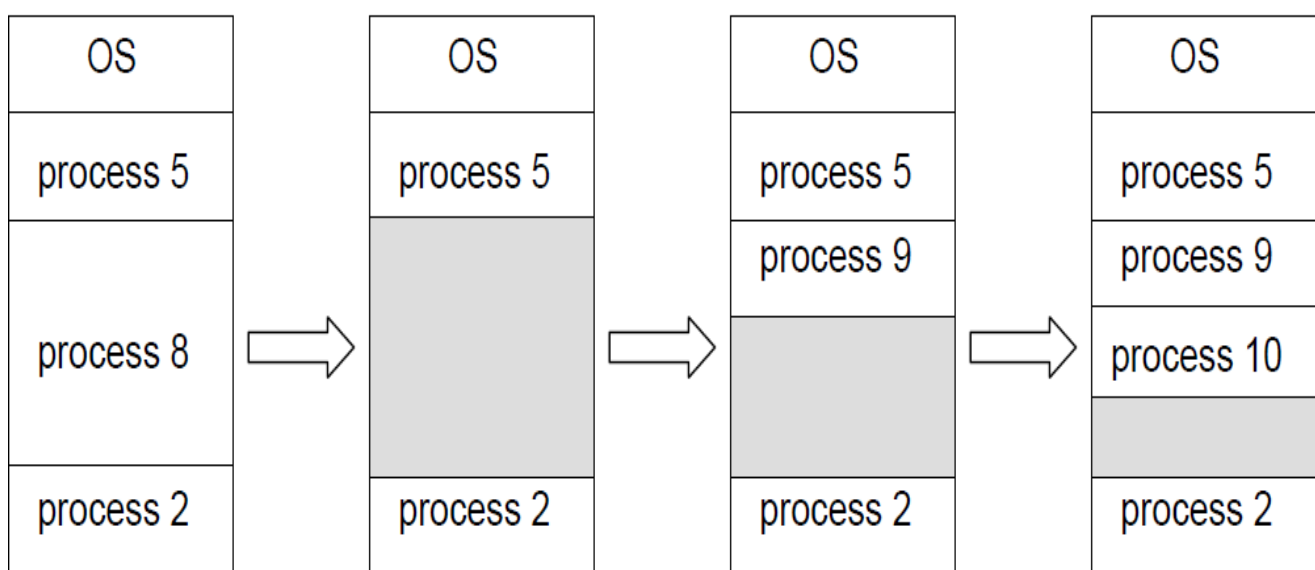
Hardware support for relocation and limit registers.

- The logical address must be less than the limit register, the MMU maps the logical address dynamically by adding the value in re-location register. When the CPU scheduler selects a process for execution, the dispatcher loads the re-location and limit register with correct values as a part of context switch.
- Since every address generated by the CPU is checked against these register we can protect the OS and other users programs and data from being modified.

Memory Allocation

Multiple-partition allocation:

- Hole – block of available memory; holes of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Operating system maintains information about:
 1. Allocated partitions
 2. Free partitions (hole)
- A set of holes of various sizes, is scattered throughout memory at any given time. When a process arrives and needs memory, the system searches this set for a hole that is large enough for this process.
- If the hole is too large, it is split into two:
 - One part is allocated to the arriving process;
 - The other is returned to the set of holes.
- When a process terminates, it releases its block of memory, which is then placed back in the set of holes.
- If the new hold is adjacent to other holes, these adjacent holes are merged to form one larger hole.
- This procedure is a particular instance of the general dynamic storage allocation problem, which is how to satisfy a request of size n from a list of free holes.



- There are many solutions to this problem. The set of holes is searched to determine which hole is best to allocate. The first-fit, best-fit and

worst-fit strategies are the most common ones used to select a free hole from the set of available holes.

The three strategies to select a free hole:-

- **First fit:-** Allocates first hole that is big enough. This algorithm scans memory from the beginning and selects the first available block that is large enough to hold the process.
 - **Best fit:-** It chooses the hole i.e., closest in size to the request. It allocates the smallest hole i.e., big enough to hold the process.
 - **Worst fit:-** It allocates the largest hole to the process request. It searches for the largest hole in the entire list.
- First fit and best fit are the most popular algorithms for dynamic memory allocation.
 - First fit is generally faster.
 - Best fit searches for the entire list to find the smallest hole i.e., large enough.
 - Worst fit reduces the rate of production of smallest holes.

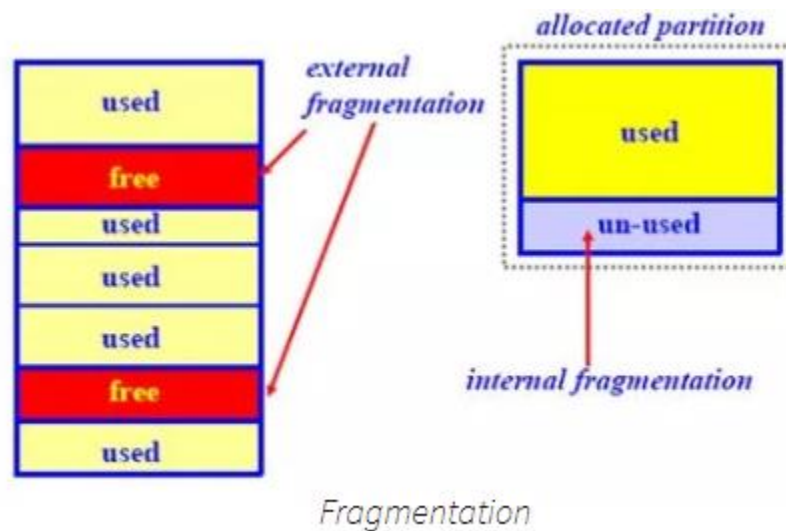
All these algorithms suffer from **fragmentation**.

Internal and External fragmentation and Compaction

As processes are loaded and removed from memory, the free memory space is broken into little pieces. After some time that process which we removed from memory cannot be allocated because of small pieces of memory and memory block remains unused. This problem is called Fragmentation.

(OR)

Fragmentation occurs when memory is allocated sequentially/contiguously in the system. Different files are stored as chunks in the memory. When these files are frequently modified or deleted, gaps/free spaces are left in between, this is called fragmentation.



Memory fragmentation can be of two types:-

- Internal Fragmentation
- External Fragmentation

Internal Fragmentation:

There is wasted space internal to a portion due to the fact that block of data loaded is smaller than the partition.

The scenario is where a free space is too small for any file to fit into. The overhead to keep a track of that free space is too much for the operating system. This is called **internal fragmentation**.

Example:- If there is a block of 50kb and if the process requests 40kb and if the block is allocated to the process then there will be 10kb of memory left.

External Fragmentation

Exists when there is enough memory space exists to satisfy the request, but it not contiguous i.e., storage is fragmented in to large number of small holes. External Fragmentation may be either minor or a major problem.

As shown in the diagram, if we want a file to fit into the memory which is equal to the sum of the freed space, we cannot as the space is not contiguous. Therefore even though there is enough memory to hold the file we cannot fit it due to the memory being scattered at different places. This is **external fragmentation**.

- One solution for over-coming external fragmentation is **compaction**.
 - The goal is to move all the free memory together to form a large block.
 - Compaction is not possible always. If the re-location is static and is done at load time then compaction is not possible.

- Compaction is possible if the re-location is dynamic and done at execution time.

Fragmented memory before compaction



Memory after compaction



- Another possible solution to the external fragmentation problem is to **permit the logical address space of a process to be non-contiguous**, thus allowing the process to be allocated physical memory whenever the latter is available.

INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
When a process is allocated more memory than required few space is left in the block which causes INTERNAL FRAGMENTATION.	After execution of processes when they are swapped out of memory and smaller processes replace them many small non contiguous blocks of empty spaces are formed which can serve a new request if put together but as they are non contiguous they cannot service a new request causing EXTERNAL FRAGMENTATION.
It occurs when memory is divided into fixed sized partitions.	It occurs when memory is divided into variable sized partitions based on the size of process.
It can be cured by allocating memory dynamically or having partitions of different sizes..	It can be cured by compaction,paging and segmentation.

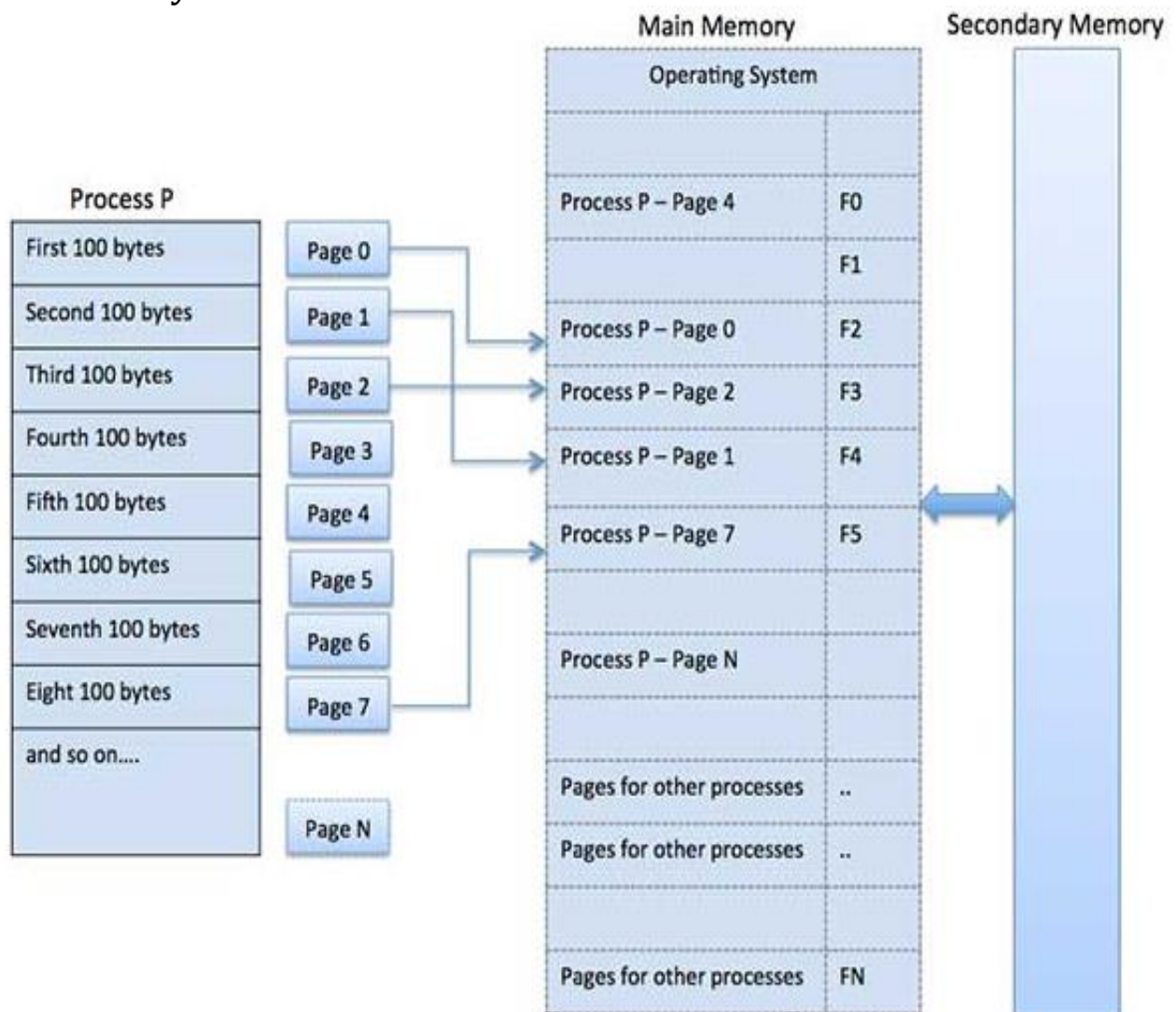
Paging

Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware. It is used to avoid external fragmentation.

Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing store. When some code or data residing in main memory need to be swapped out, space must be found on backing store.

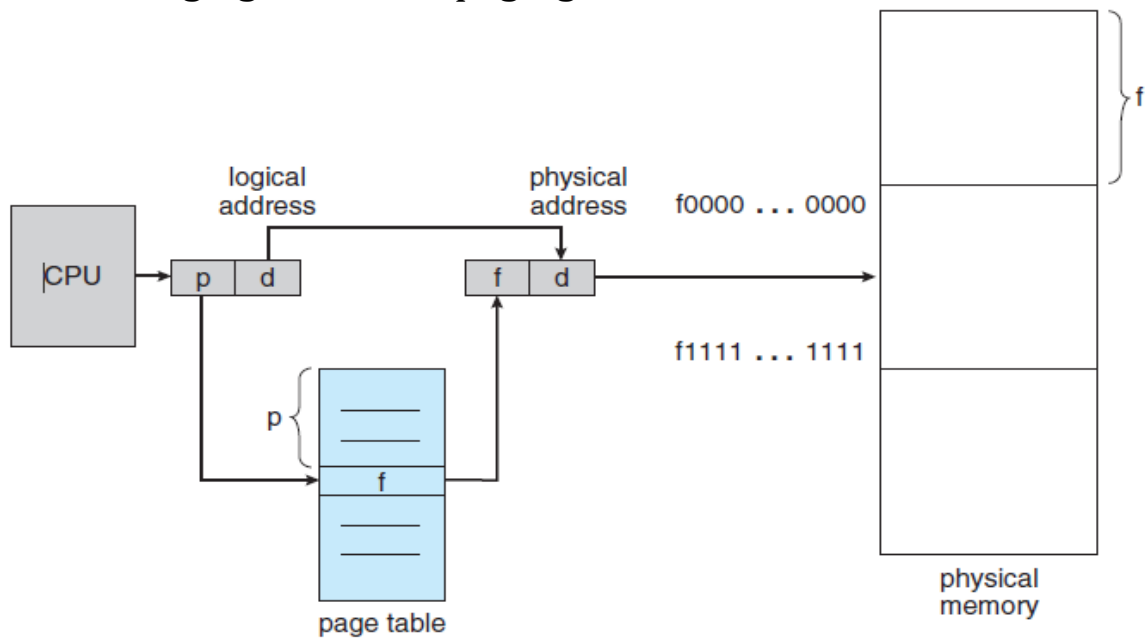
Basic Method:-

- Physical memory is broken in to fixed sized blocks called **frames (f)**.
- Logical memory is broken in to blocks of same size called **pages (p)**.
- When a process is to be executed its pages are loaded in to available frames from backing store.
- The blocking store is also divided in to fixed-sized blocks of same size as memory frames.



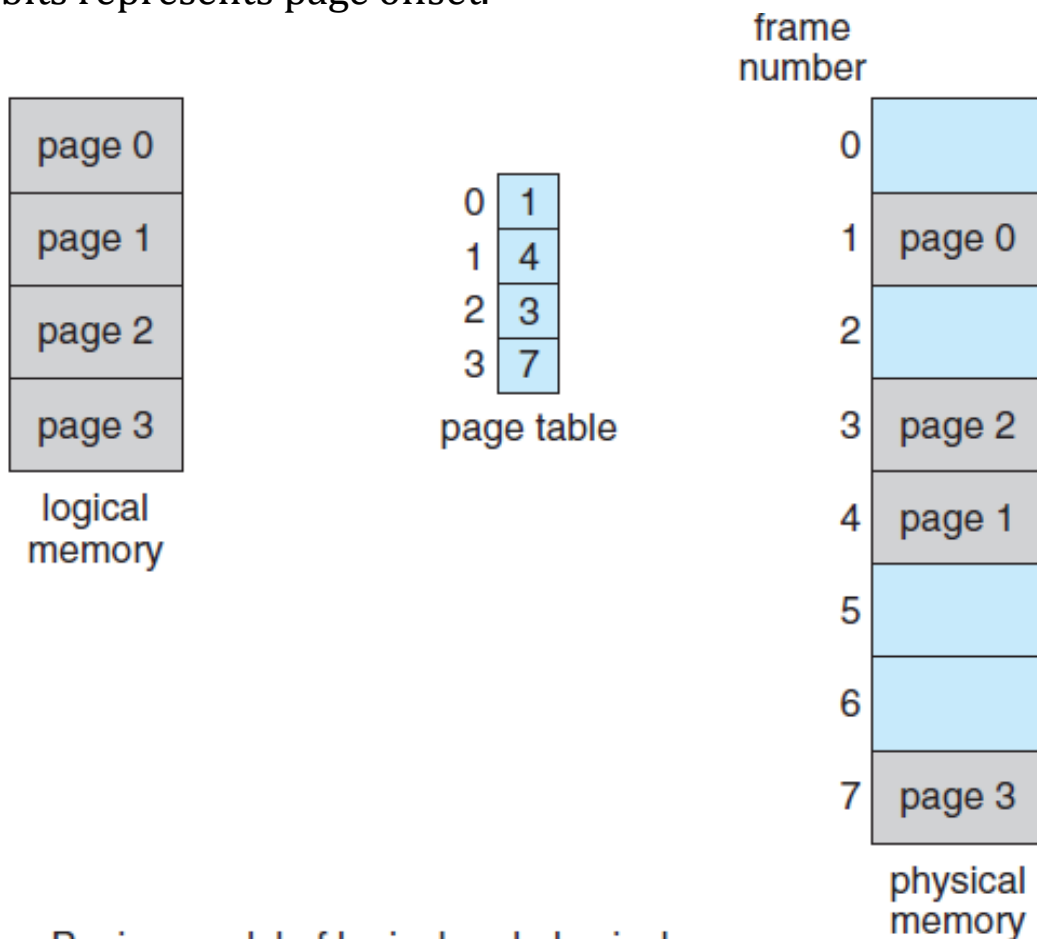
- Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d).
- The page number (p) is used as index to the page table. The page table contains base address of each page in physical memory. This base address is combined with the page offset to define the physical memory i.e., sent to the memory unit.

The following figure shows paging hardware:-



Paging hardware.

- The page size is defined by the hardware. The size of a power of 2, varying between 512 bytes and 10Mb per page.
- If the size of logical address space is 2^m address unit and page size is 2^n , then high order $m-n$ designates the page number and n low order bits represents page offset.

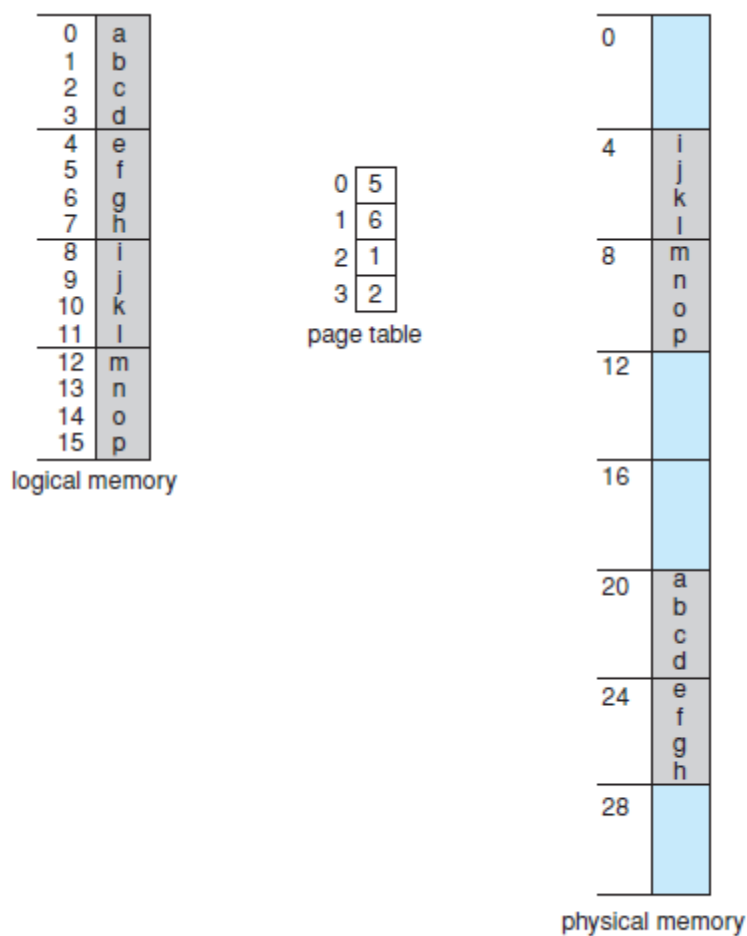


Paging model of logical and physical memory.

Example:-

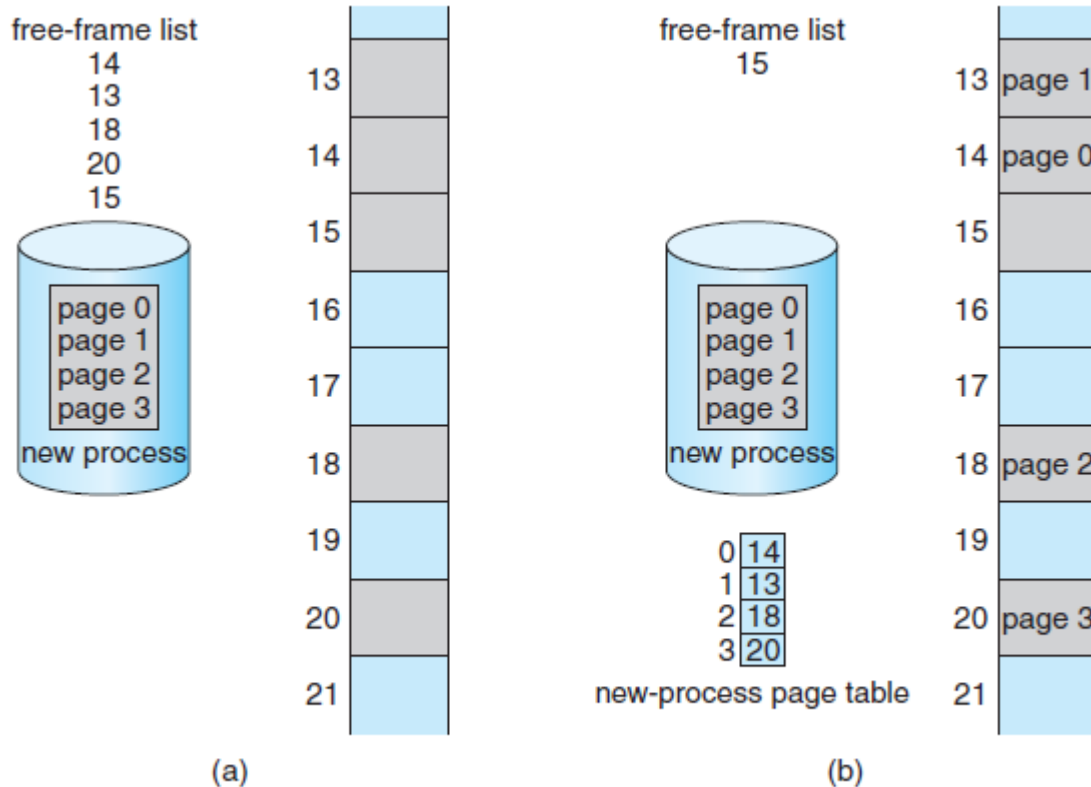
To show how to map logical memory in to physical memory consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).

- Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. $[(5 \times 4) + 0]$.
- Logical address 3 is page 0 and offset 3 maps to physical address 23 $[(5 \times 4) + 3]$.
- Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 $[(6 \times 4) + 0]$.
- Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 $[(2 \times 4) + 1]$.



Paging example for a 32-byte memory with 4-byte pages.

When a process arrives in the system to be executed, its size, expressed in pages, is examined. Each page of the process needs one frame. Thus, if the process requires n pages, at least n frames must be available in memory. If n frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame, its frame number is put into the page table, and so on



Free frames (a) before allocation and (b) after allocation.

Since the operating system is managing physical memory, it must be aware of the allocation details of physical memory—which frames are allocated, which frames are available, how many total frames there are, and so on. This information is generally kept in a data structure called a **frame table**. The frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process or processes.

Address Translation:

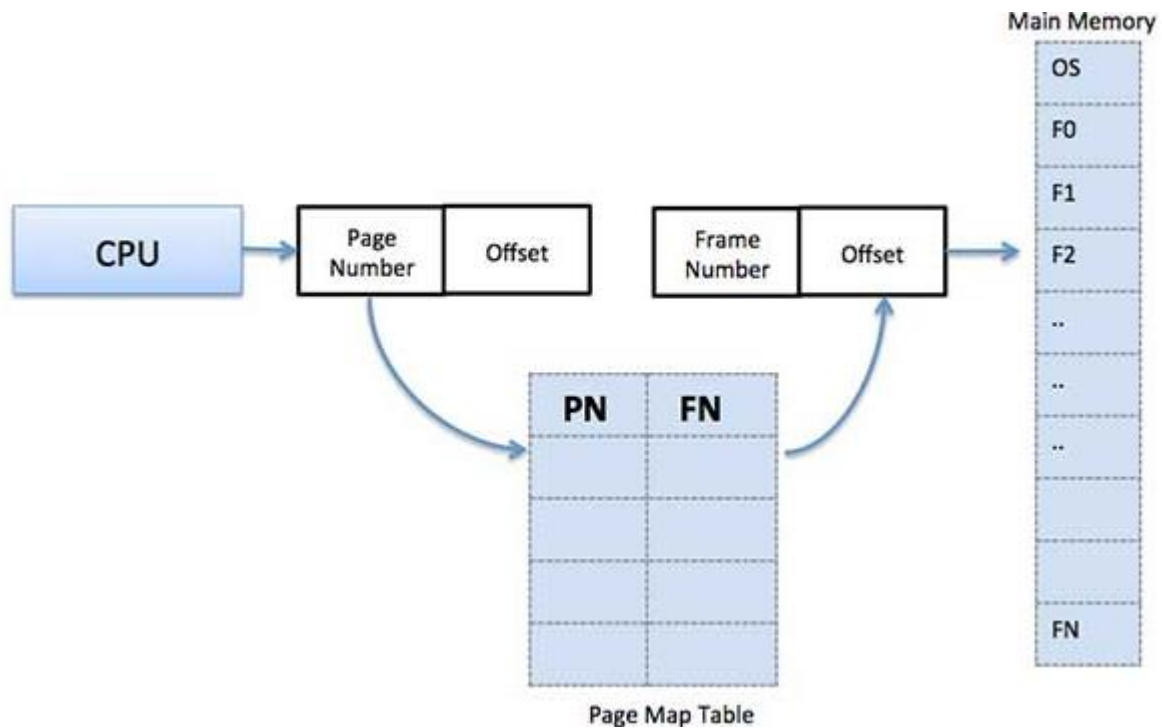
Page address is called **logical address** and represented by **page number** and the **offset**.

$$\text{Logical Address} = \text{Page number} + \text{page offset}$$

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

$$\text{Physical Address} = \text{Frame number} + \text{page offset}$$

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Virtual Memory

It is a technique which allows execution of process that may not be compiled within the primary memory.

- It separates the user logical memory from the physical memory.
- This separation allows an extremely large memory to be provided for program when only a small physical memory is available.
- Virtual memory makes the task of programming much easier because the programmer no longer needs to working about the amount of the physical memory is available or not.
- The virtual memory allows files and memory to be shared by different processes by page sharing.
- It is most commonly implemented by demand paging.

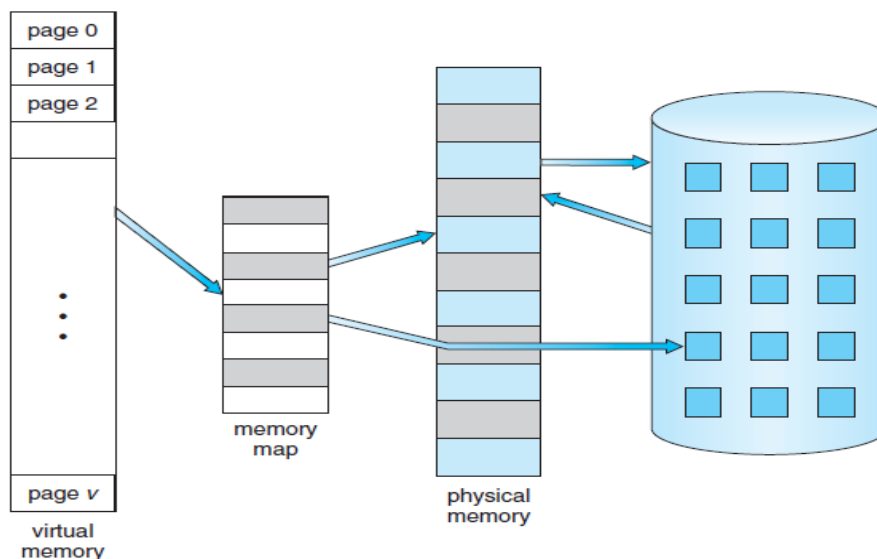
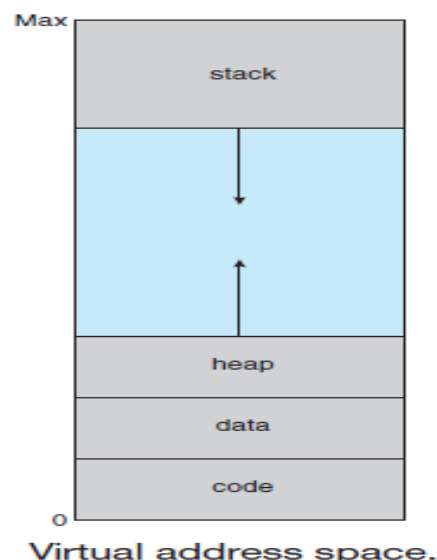


Diagram showing virtual memory that is larger than physical memory.

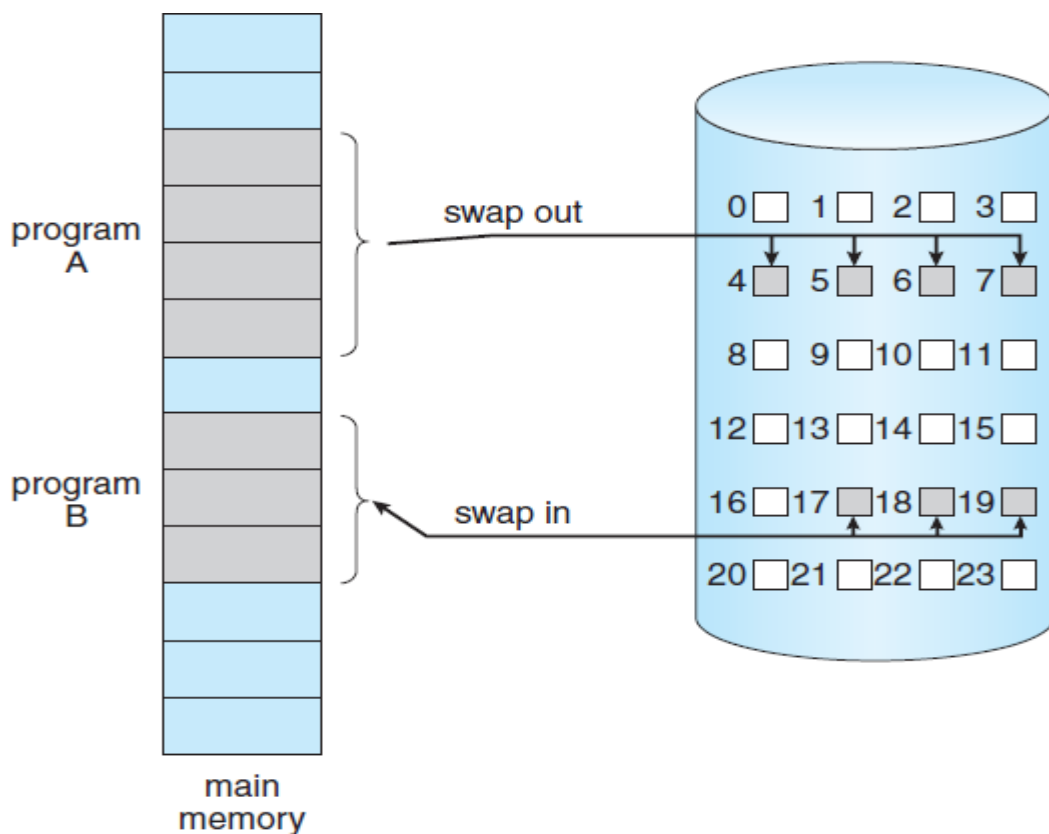
The **virtual address space** of a process refers to the logical (or virtual) view of how a process is stored in memory. Typically, this view is that a process begins at a certain logical address—say, address 0—and exists in contiguous memory



Demand paging

- A demand paging system is similar to the paging system with swapping feature.
- When we want to execute a process we swap it into the memory. A swapper manipulates entire process where as a pager is concerned with the individual pages of a process.
- The demand paging concept is using pager rather than swapper.
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again.
- Instead of swapping in a whole process, the pager brings only those necessary pages into memory.

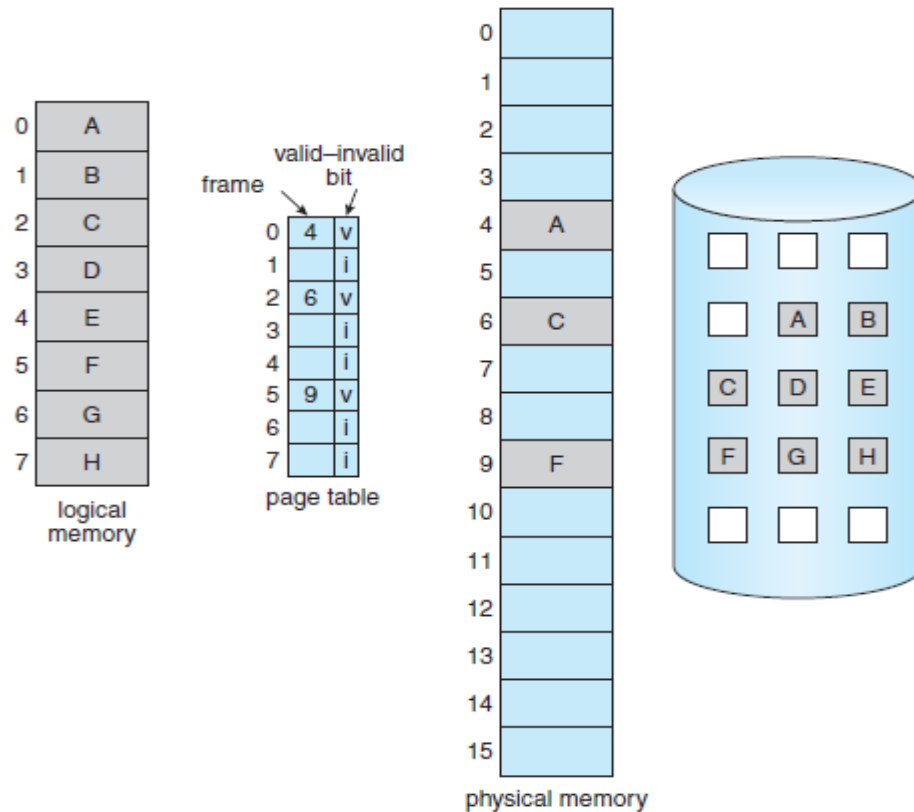
The transfer of a paged memory to contiguous disk space is shown in below figure.



Transfer of a paged memory to contiguous disk space.

- Thus it avoids reading into memory pages that will not be used any way decreasing the swap time and the amount of physical memory needed.
- In this technique we need some hardware support to distinguish between the pages that are in memory and those that are on the disk.
- A **valid and invalid bit** is used for this purpose.
- When this bit is set to valid it indicates that the associated page is in memory.

- If the bit is set to invalid it indicates that the page is either not valid or is valid but currently not in the disk.



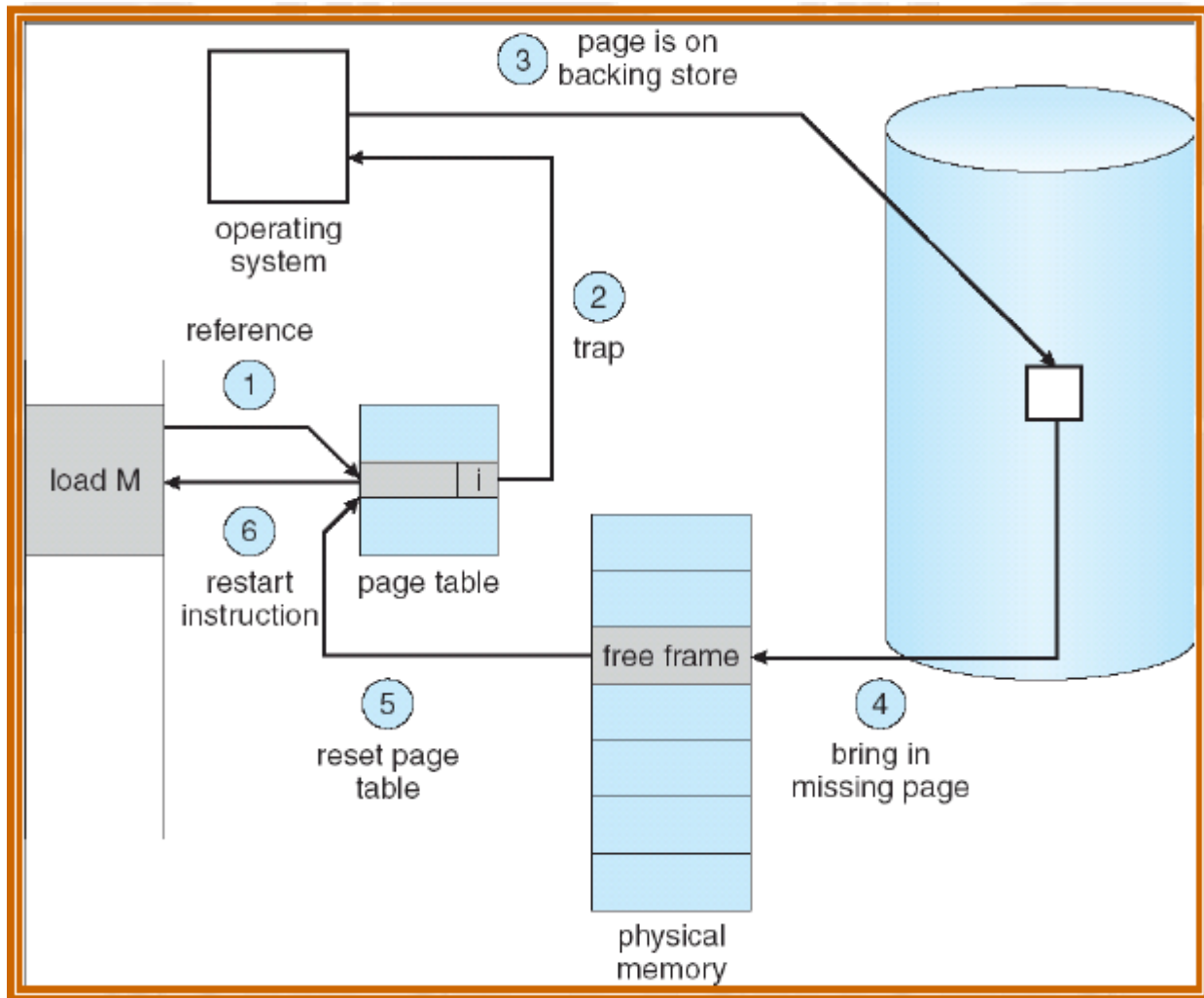
Page table when some pages are not in main memory.

- Marking a page invalid will have no effect if the process never attempts to access that page.
- So while a process executes and access pages that are memory resident, execution proceeds normally.
- Access to a page marked invalid causes a **page fault** trap.
- It is the result of the OS's failure to bring the desired page into memory.

Procedure to handle page fault

If a process refers to a page that is not in physical memory then

- We check an internal table (page table) for this process to determine whether the reference was valid or invalid.
- If the reference was invalid, we terminate the process, if it was valid but not yet brought in, we have to bring that from main memory.
- Now we find a free frame in memory.
- Then we read the desired page into the newly allocated frame.
- When the disk read is complete, we modify the internal table to indicate that the page is now in memory.
- We restart the instruction that was interrupted by the illegal address trap. Now the process can access the page as if it had always been in memory.



Hardware support:-

For demand paging the same hardware is required as paging and swapping.

- 1. Page table:-** Has the ability to mark an entry invalid through valid-invalid bit.
- 2. Secondary memory:-** This holds the pages that are not present in main memory. It's a high speed disk.

Performance of demand paging:-

- Demand paging can have significant effect on the performance of the computer system.
- Let P be the probability of the page fault ($0 \leq P \leq 1$) Effective access time = $(1-P) * ma + P * \text{page fault}$. Where P = page fault and ma = memory access time.
- Effective access time is directly proportional to page fault rate. It is important to keep page fault rate low in demand paging.

A page fault causes the following sequence to occur:-

1. Trap to the OS.
2. Save the user registers and process state.

3. Determine that the interrupt was a page fault.
4. Checks the page references were legal and determine the location of page on disk.
5. Issue a read from disk to a free frame.
6. If waiting, allocate the CPU to some other user.
7. Interrupt from the disk.
8. Save the registers and process states of other users.
9. Determine that the interrupt was from the disk.
10. Correct the page table and other table to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user register process state and new page table then resume the interrupted instruction.

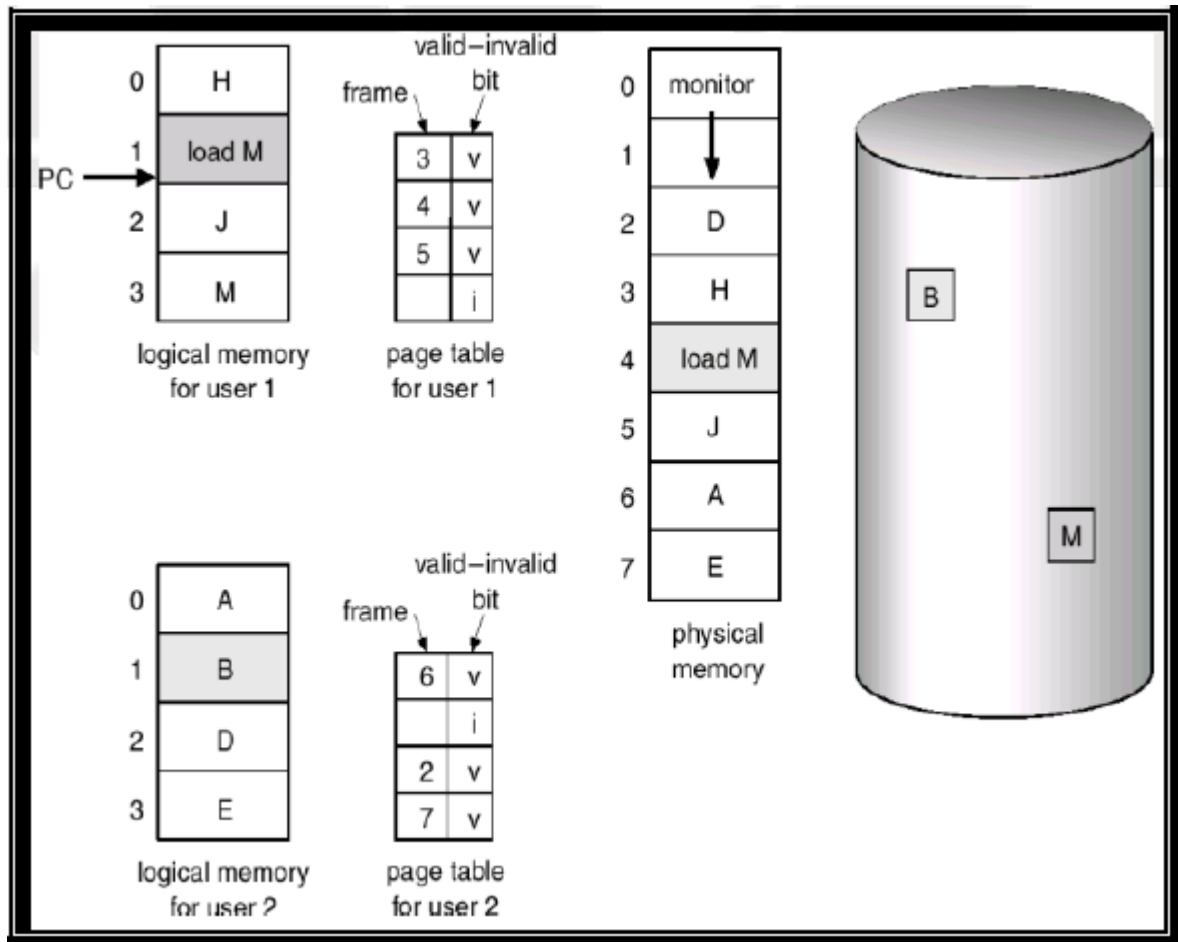
Page Replacement policies

- Each process is allocated frames (memory) which hold the process's pages (data)
- Frames are filled with pages as needed – this is called **demand paging**
- Over-allocation of memory is prevented by modifying the page-fault service routine to replace pages
- The job of the page replacement algorithm is to decide which page gets victimized to make room for a new page
- Page replacement completes separation of logical and physical memory

Page Replacement

- ✓ Demand paging shares the I/O by not loading the pages that are never used.
- ✓ Demand paging also improves the degree of multiprogramming by allowing more process to run at the same time.
- ✓ Page replacement policy deals with the solution of pages in memory to be replaced by a new page that must be brought in. When a user process is executing a page fault occurs.
- ✓ The hardware traps to the operating system, which checks the internal table to see that this is a page fault and not an illegal memory access.

- ✓ The operating system determines where the derived page is residing on the disk, and this finds that there are no free frames on the list of free frames.
- ✓ When all the frames are in main memory, it is necessary to bring a new page to satisfy the page fault, replacement policy is concerned with selecting a page currently in memory to be replaced.
- ✓ The page i.e to be removed should be the page i.e least likely to be referenced in future.

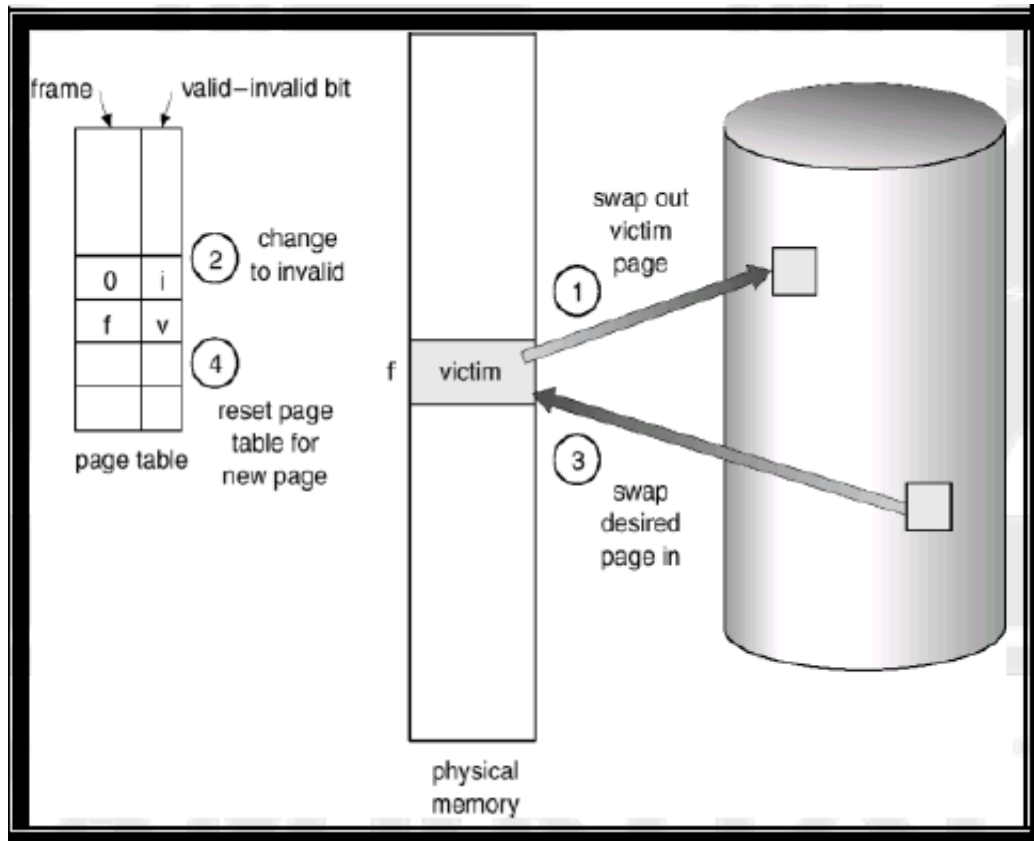


- ✓ When a user process is executing a page fault occurs. The OS determines where the desired pages is residing on disk but finds that there are no free frames on the free frame list i.e. all memory is in use as shown in the above figure.
- ✓ The OS has several options at this point. It can terminate the process.
- ✓ The OS could instead swap out a process by freeing all its frames and reducing the degree of multiprogramming.

Working of Page Replacement Algorithm

1. Find the location of derived page on the disk.
2. Find a free frame
 - a. If there is a free frame, use it.

- b. Otherwise, use a replacement algorithm to select a victim.
- c. Write the victim page to the disk; change the page and frame tables accordingly.
3. Read the desired page into the free frame; change the page and frame tables.
4. Restart the user process.



Victim Page

The page that is supported out of physical memory is called victim page.

- ✓ If no frames are free, the two page transforms come (out and one in) are read. This will see the effective access time.
- ✓ Each page or frame may have a dirty (modify) bit associated with the hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.
- ✓ When we select the page for replacement, we check its modify bit. If the bit is set, then the page is modified since it was read from the disk.
- ✓ If the bit was not set, the page has not been modified since it was read into memory. Therefore, if the copy of the page has not been modified we can avoid writing the memory page to the disk, if it is already there. Some pages cannot be modified.

We must solve two major problems to implement demand paging:

- ✓ We must develop a frame allocation algorithm and a page replacement algorithm.
- ✓ If we have multiple processors in memory, we must decide how many frames to allocate and page replacement is needed.

Page Replacement Algorithm

It decides which memory page to page will be swapped out when a page of memory needs to be allocated.

Page Fault: The CPU demanded page is not present in memory.

Page Hit: The CPU demanded page is present in memory.

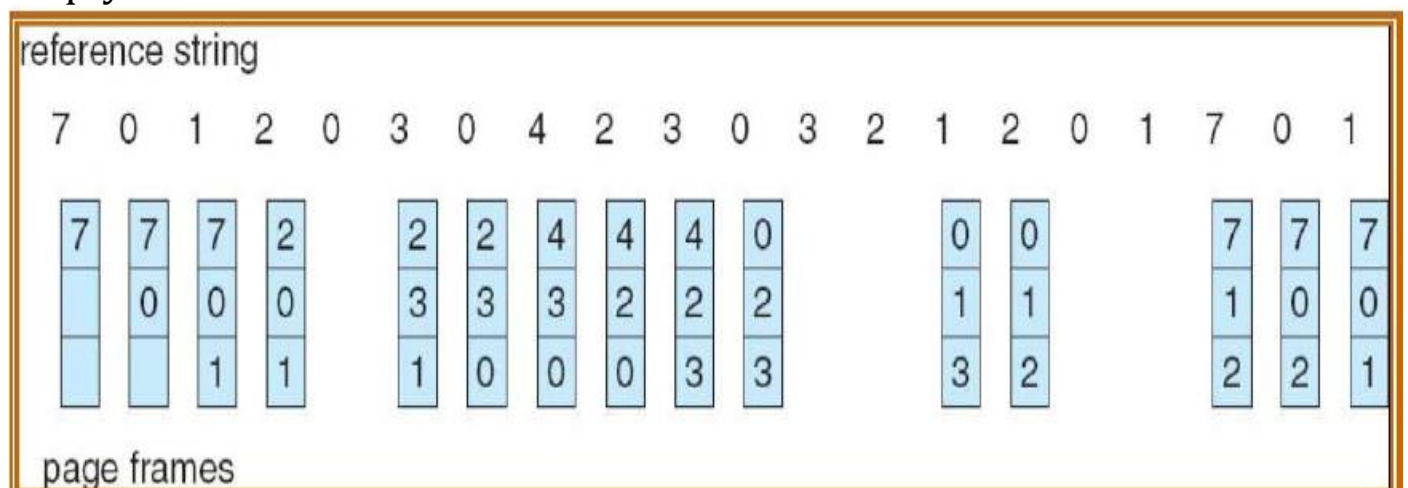
There are 3 different types:

1. FIFO (First In First Out)
2. Optimal
3. LRU (Least Recently Used)

FIFO (First In First Out) algorithm:

- This is the simplest page replacement algorithm. A FIFO replacement algorithm associates each page the time when that page was brought into memory.
- When a Page is to be replaced the oldest one is selected.
- We replace the queue at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

Example: Consider the following references string with frames initially empty.



- The first three references (7,0,1) causes page faults and are brought into the empty frames.
- The next reference 2 replaces page 7 because the page 7 was brought in first.
- Since 0 is the next reference and 0 is already in memory, it has no page faults.
- The next reference 3 results in page 0 being replaced so that the next reference to 0 causes a page fault. This will continue till the end of string. There are 15 faults all together.

Analysis:

- Number of Page references = 20
- Number of Page faults = 15
- Number of Page Hits = 5

$$\begin{aligned}\text{Page Hit Ratio} &= \text{Number of Page Hits} / \text{Total Number of Page references} \\ &= (5 / 20) * 100 \\ &= 25\%\end{aligned}$$

$$\begin{aligned}\text{Page Fault Ratio} &= \text{Number of Page Faults} / \text{Total Number of Page references} \\ &= (15 / 20) * 100 \\ &= 75\%\end{aligned}$$

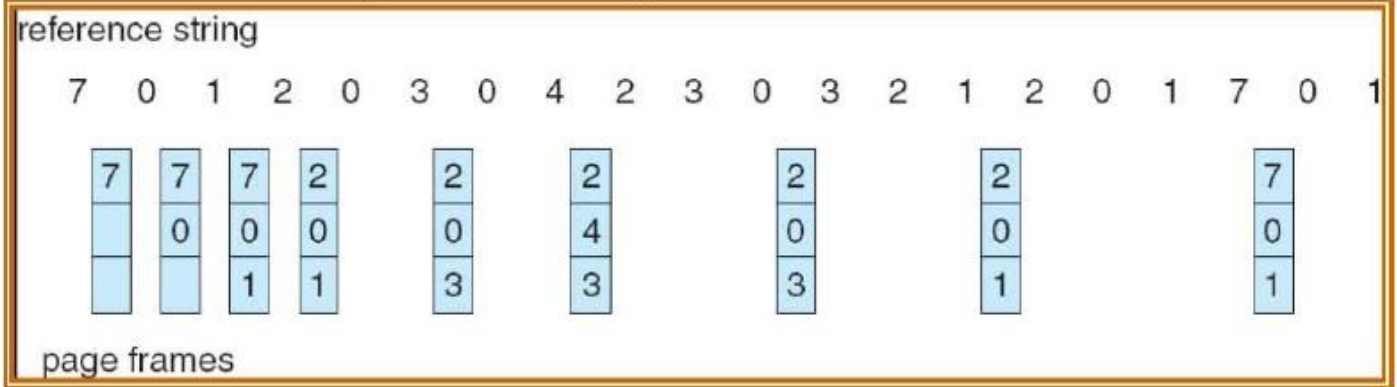
Belady's Anomaly:

For some page replacement algorithm, the page fault may increase as the number of allocated frames increases. FIFO replacement algorithm may face this problem.

Optimal Algorithm:

- Optimal page replacement algorithm is mainly to solve the problem of Belady's Anomaly.
- Ideally we want to select an algorithm with the lowest page-fault rate
- Such an algorithm exists, and is called (unsurprisingly) the optimal algorithm:
- Procedure: replace the page that will not be used for the longest time (or at all) – i.e. replace the page with the greatest forward distance in the reference string

Example: Consider the following reference string with frames initially empty.



- The first three references cause faults that fill the three empty frames.
- The references to page 2 replaces page 7, because 7 will not be used until reference 18.
- The page 0 will be used at 5 and page 1 at 14.
- With only 9 page faults, optimal replacement is much better than a FIFO, which had 15 faults.
- This algorithm is difficult to implement because it requires future knowledge of reference strings.

Analysis:

- Number of Page references = 20
- Number of Page faults = 09
- Number of Page Hits = 11

$$\begin{aligned}\text{Page Hit Ratio} &= \text{Number of Page Hits} / \text{Total Number of Page references} \\ &= (11 / 20) * 100 \\ &= 55\%\end{aligned}$$

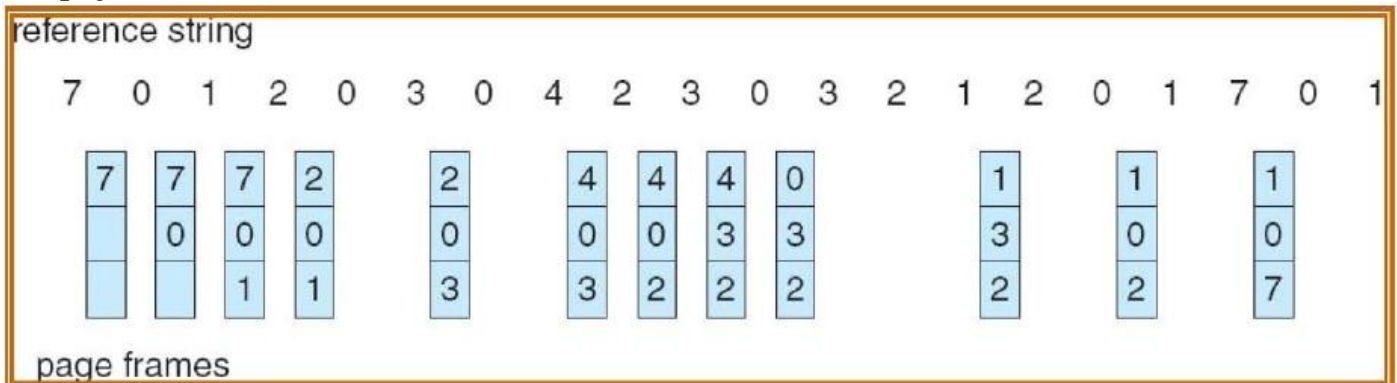
$$\begin{aligned}\text{Page Fault Ratio} &= \text{Number of Page Faults} / \text{Total Number of Page references} \\ &= (9 / 20) * 100 \\ &= 45\%\end{aligned}$$

Least Recently Used (LRU) Algorithm:

- If the optimal algorithm is not feasible, an approximation to the optimal algorithm is possible.
- The main difference b/w OPTS and FIFO is that; FIFO algorithm uses the time when the pages was built in and OPT uses the time when a page is to be used.
- The LRU algorithm replaces the pages that have not been used for longest period of time.
- The LRU associated its pages with the time of that pages last use.

- This strategy is the optimal page replacement algorithm looking backward in time rather than forward.

Example: Consider the following references string with frames initially empty.



- The first 5 faults are similar to optimal replacement.
- When reference to page 4 occurs, LRU sees that of the three frames, page 2 as used least recently.
- The most recently used page is page 0 and just before page 3 was used.
- The LRU policy is often used as a page replacement algorithm and considered to be good.

Analysis:

- Number of Page references = 20
- Number of Page faults = 12
- Number of Page Hits = 08

$$\begin{aligned}
 \text{Page Hit Ratio} &= \text{Number of Page Hits} / \text{Total Number of Page references} \\
 &= (8 / 20) * 100 \\
 &= \mathbf{40\%}
 \end{aligned}$$

$$\begin{aligned}
 \text{Page Fault Ratio} &= \text{Number of Page Faults} / \text{Total Number of Page references} \\
 &= (12 / 20) * 100 \\
 &= \mathbf{60\%}
 \end{aligned}$$