**Ch-4 Contents:**

**Memory Management**

Main Memory management Requirement, memory partitioning techniques –static and dynamic,Free space management Techniques – Bitmap, linked list.
Paging: concept, structure of page table.
Swapping, Segmentation.
Virtual Memory: Background, Demand paging,
Page Replacement Algorithms: FIFO, LRU.

**Introduction:**

- Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.
- Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.
- In operating systems, Memory Management is the function responsible for allocating and managing computer's main memory.
- Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.
- The main memory must accommodate both the operating system and the various user processes. We therefore need to allocate main memory in the most efficient way possible. This section explains one common method, contiguous memory allocation.
- The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.
- We can place the operating system in either low memory or high memory. The major factor affecting this decision is the location of the interrupt vector. Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well. Thus, in this text, we discuss only the situation in which the operating system resides in low memory. The development of the other situation is similar.
- We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory.
-

**Basic Memory Management**

In operating systems, Memory Management is the function responsible for allocating and managing computer's main memory.

Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.

There are two Memory Management Techniques: Contiguous, and Non-Contiguous. In Contiguous Technique, executing process must be loaded entirely in main-memory.

Contiguous Technique can be divided into:

1. Fixed (or static) partitioning
2. Variable (or dynamic) partitioning

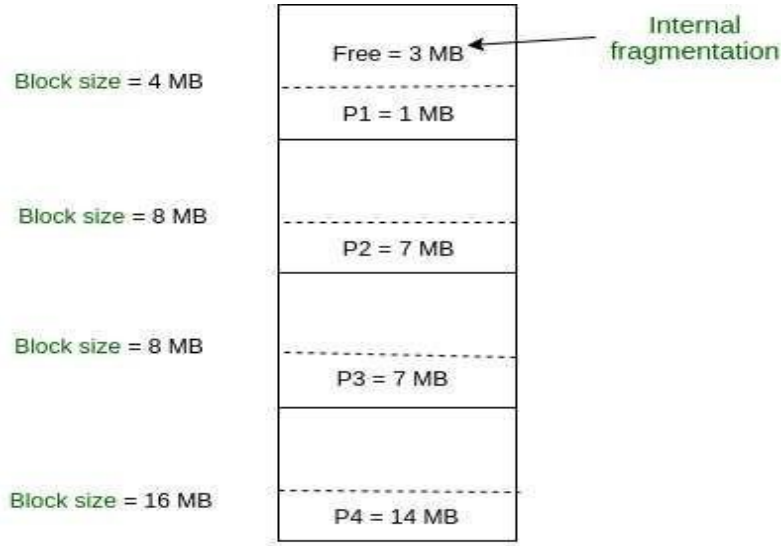
**Partitioning:**

Partitioning is the way of allocating memory. In this we can allocate memory in two ways

a. Fixed size (Static)
b. Variable size (Dynamic)

**a. Fixed size (Static)**

- This is the oldest and simplest technique used to put more than one processes in the main memory.
- One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions Each partition may contain exactly one process.
- Thus, the degree of multiprogramming is bound by the number of partitions.
- In this multiple partition method when a partition is free, a process is selected from the input queue and is loaded into the free partition.
- When the process terminates, the partition becomes available for another process.
- In this partitioning, numbers of partitions (non-overlapping) in RAM are fixed but size of each partition may or may not be same. As it is contiguous allocation, hence no spanning is allowed. Here partition is made before execution or during system configure.



Fixed size partition

- **Advantages of Fixed Partitioning –**
  - **Easy to implement:**
    Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into certain partition without focussing on the emergence of Internal and External Fragmentation.
  - **Little OS overhead:**
    Processing of Fixed Partitioning require lesser excess and indirect computational power.

- **Disadvantages of Fixed Partitioning** –
  - **Internal Fragmentation:**
    Main memory use is inefficient. Any program, no matter how small, occupies an

entire partition. This can cause internal fragmentation.

- **External Fragmentation:**

  The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).
- **Limit process size:**

  Process of size greater than size of partition in Main Memory cannot be accommodated. Partition size cannot be varied according to the size of incoming process's size. Hence, process size of 32MB in above stated example is invalid.
- **Limitation on Degree of Multiprogramming:**

  Partition in Main Memory are made before execution or during system configure. Main Memory is divided into fixed number of partition. Suppose if there are partitions in RAM and are the number of processes, then condition must be fulfilled. Number of processes greater than number of partitions in RAM is invalid in Fixed Partitioning.

**b. Variable size (Dynamic)**

- In the variable partition scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied.
- Initially, all memory is available for user processes and is considered one large block of available memory as a hole.
- In general as mentioned, the memory blocks available comprise a *set* of holes of various sizes scattered throughout memory.
- When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process.
- If the hole is too large, it is split into two parts. 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 hole 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 concerns how to satisfy a request of size *n* from a list of free holes.
- There are many solutions to this problem which can used to select a free hole from the set of available holes. These are:
  **First-fit**: Allocate the *first* hole that is big enough
  **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole
  **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization. Both the first-fit and best-fit strategies for memory allocation suffer from External fragmentation.
- **Advantages of Variable Partitioning –**
  - **No Internal Fragmentation:**

    In variable Partitioning, space in main memory is allocated strictly according to the need of process, hence there is no case of internal fragmentation. There will be no unused space left in the partition.
  - **No restriction on Degree of Multiprogramming:**

    More number of processes can be accommodated due to absence of internal fragmentation. A process can be loaded until the memory is not empty.
  - **No Limitation on the size of the process:**

    In Fixed partitioning, the process with the size greater than the size of the

largest partition could not be loaded and process cannot be divided as it is invalid

in contiguous allocation technique. Here, In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

- **Disadvantages of Variable Partitioning –**
  - **Difficult Implementation:**
    Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.
  - **External Fragmentation:**
    There will be external fragmentation inspite of absence of internal fragmentation.

**Free Space management Techniques**

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free space list.
- The free-space list records all *free* disk blocks-those not allocated to some file or directory.
- To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

**c. Bit Vector:**

- The main task of the operating system is to keep track of whether the partition is free or filled. For this purpose, the operating system also manages another data structure that is called bitmap.
- Frequently, the free-space list is implemented as a bit map or a bit vector.
- Each block is represented by 1 bit.
- If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- The process or the hole in Allocation units is represented by a flag bit of bitmap. In the image shown below, a flag bit is defined for every bit of allocation units
- A **string of 0s** in the bitmap shows that there is a **hole** in the relative Allocation unit while **the string of 1s** represents the **process** in the relative allocation unit.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free space bit map would be:
  00111100111111000110000011100000 ...
- **Advantages:**
  - The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or *n* consecutive free blocks on the disk.
- **Disadvantages**
  - The OS has to assign some memory for bitmap as well since it stores the details about allocation units. That much amount of memory cannot be used to load any process therefore that decreases the degree of multiprogramming as well as throughput
  - To identify any hole in the memory, the OS need to search the string of 1s in the bitmap. This searching takes a huge amount of time which makes the system inefficient to some extent.

**d. Linked List:**

- The better and the most popular approach to keep track the free or filled partitions is using Linked List
- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.
- For example: blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, and so on.

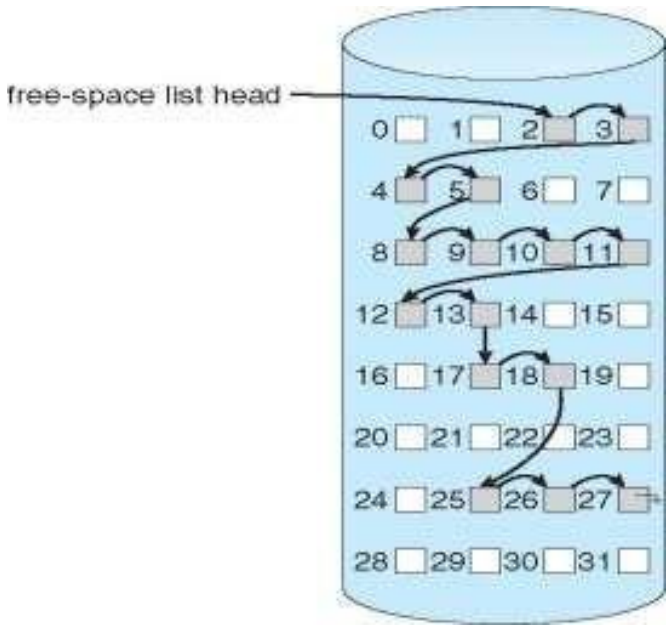



**Fig :- Linked free-space list on disk.**

**Virtual Memory –**
**Concept**

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased. Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

In practice, most real processes do not need all their pages, or at least not all at once, for several reasons:

1. Error handling code is not needed unless that specific error occurs, some of which are

quite rare.

2. Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
3. Certain features of certain programs are rarely used, such as the routine to balance the federal budget.

The ability to load only the portions of processes that were actually needed ( and only when they were needed ) has several benefits:

- Programs could be written for a much larger address space ( virtual memory space ) than physically exists on the computer.
- Because each process is only using a fraction of their total address space, there is more memory left for other programs, improving CPU utilization and system throughput.
- Less I/O is needed for swapping processes in and out of RAM, speeding things up.

Following figure shows the general layout of virtual memory, which can be much larger than physical memory:

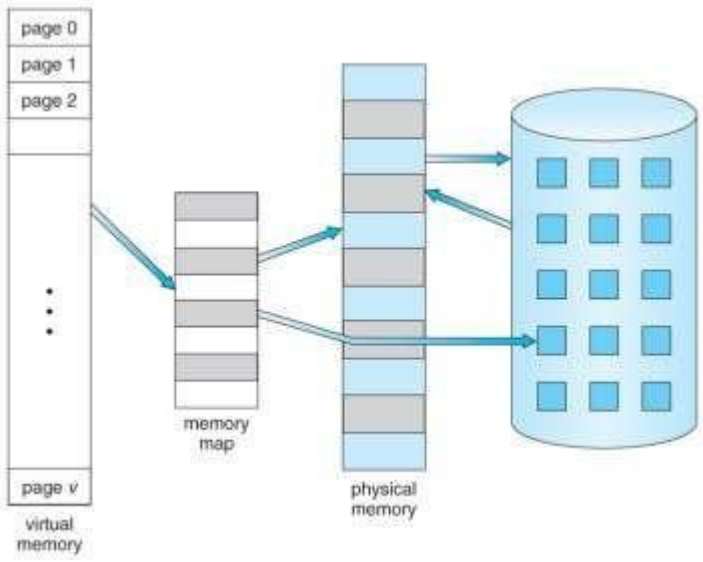



**Figure - Diagram showing virtual memory that is larger than physical memory**

**Fragmentation:**

- As processes are loaded and removed from memory, the free memory space is broken into little pieces
- All the memory allocation strategies suffer from external fragmentation, though first and best fits experience the problems more so than worst fit.
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as **Fragmentation.**
- Fragmentation is of two types −

- **External Fragmentation** –
  - External fragmentation means that the available memory is broken up into lots of little pieces, none of which is big enough to satisfy the next memory requirement, although the sum total could.
  - Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
- **Internal Fragmentation** –
  - allocated memory may be slightly larger than requested memory
  - this size difference is memory internal to a partition,
  - Some portion of memory is left unused

- **External fragmentation can be reduced by compaction** or shuffle memory contents to place all free memory together in one large block.
- To make compaction feasible, relocation should be dynamic.
- **The internal fragmentation can be reduced by** effectively **assigning the smallest partition but large enough for the process.**

**Segmentation:**

- Most users do not think of their programs as existing in one continuous linear address space.
- Rather they tend to think of their memory in multiple segments, each dedicated to a particular use, such as code, data, the stack, the heap, etc.
- Memory segmentation supports this view by providing addresses with a segment number and an offset from the beginning of that segment.
- For example, a C compiler might generate 5 segments for the user code, library code, global ( static ) variables, the stack, and the heap, as shown in following figure

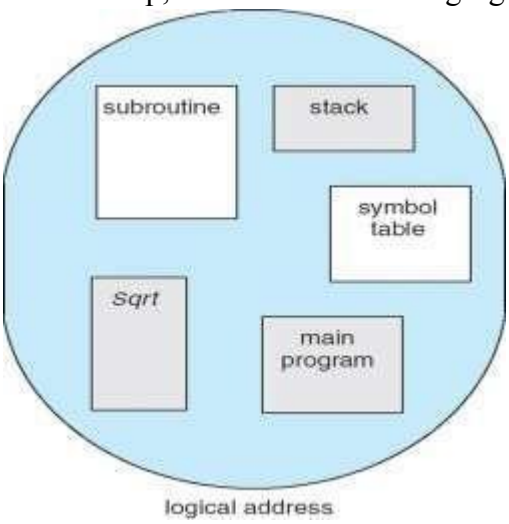


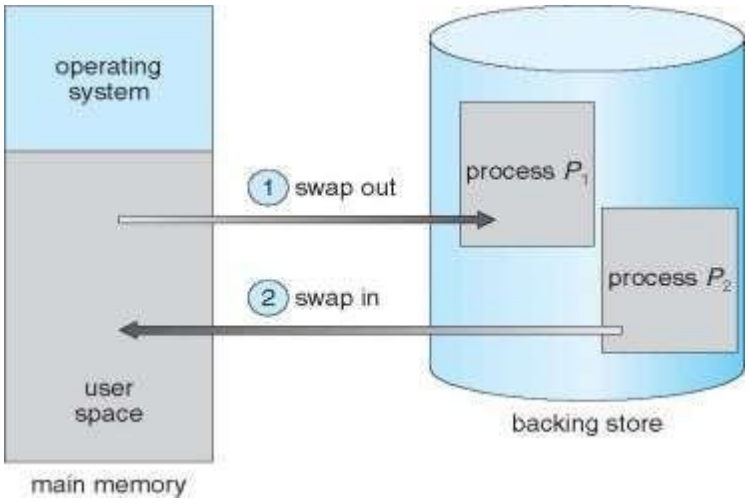

**Fig: User's/programmer's view of a program**

- It is a Memory-management scheme that supports user view of memory
- A program is a collection of segments where a segment is a logical unit such as:
  - main program
  - procedure
  - function
  - method
  - object
  - local variables, global variables

- common block
- stack
- symbol table
- arrays

### 5.2.2 Swapping:

- A process must be in memory to be executed.
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- Total physical memory space of processes can exceed physical memory
- For example, assume a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into the memory space that has been freed.
- **Backing store** is fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a ready queue of ready-to-run processes which have memory images on disk




**Paging and Page Table:**

Paging is a memory-management scheme that permits the physical address space of a process to be noncontiguous.

Paging avoids external fragmentation and the need for compaction.

Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available

Divide physical memory into fixed-sized blocks called **frames.**

- Size is power of 2, between 512 bytes and 16 Mbytes

Divide logical memory into blocks of same size called **pages**

Keep track of all free frames

To run a program of size *N* pages, need to find *N* free frames and load program

Set up a **page table** to translate logical to physical addresses

Backing store likewise split into pages

Still have Internal fragmentation

Address translation scheme:

Address generated by CPU(Logical address) is divided into:

- **Page number (*p*)** – used as an index into a **page table** which contains base address of each page in physical memory
- **Page offset (*d*)** – combined with base address to define the physical memory address that is sent to the memory unit
- For given logical address space $2^m$ and page size $2^n$


**Paging Hardware:**

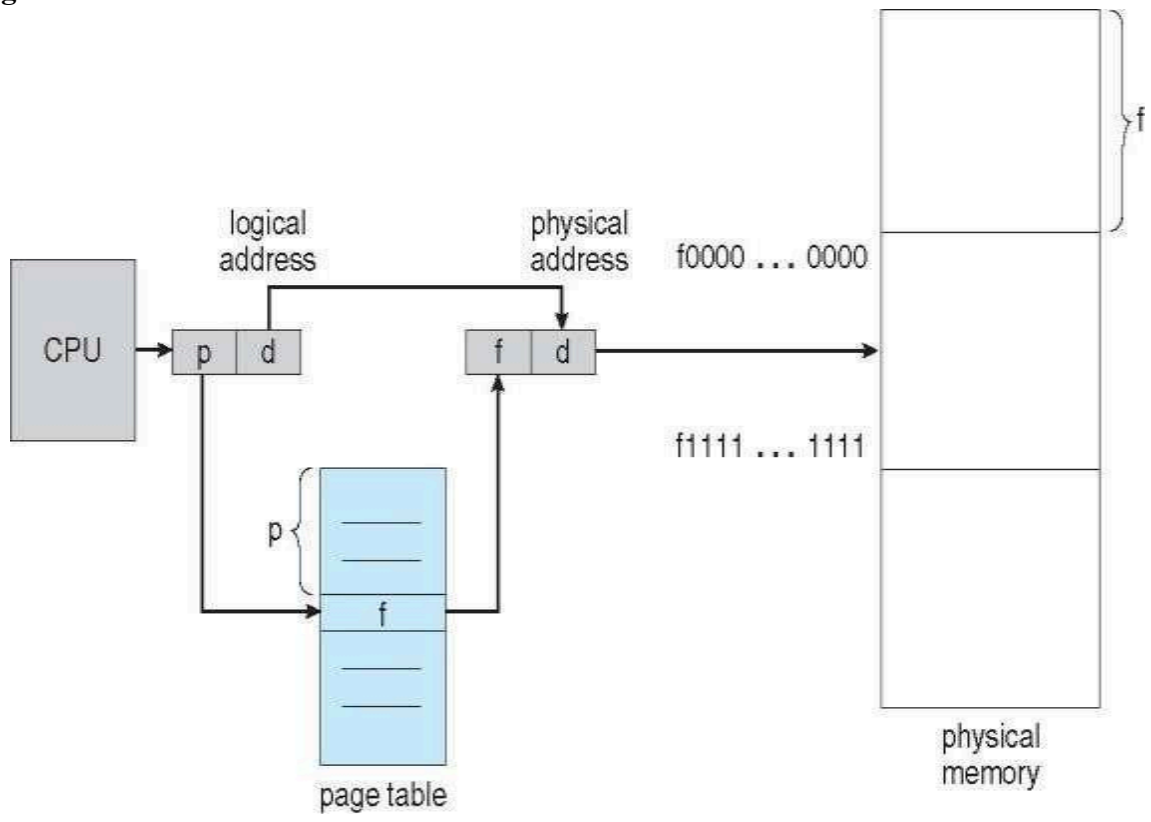



**Fig :- Paging hardware**

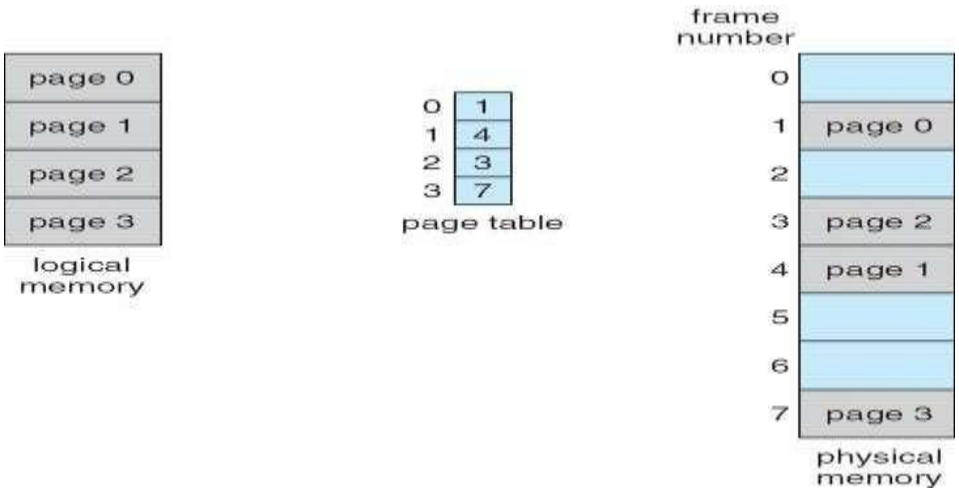



**Fig. :-Paging Model of Logical and Physical Memory:**

The page table maps the page number to a frame number, to yield a physical address which also has two parts: The frame number and the offset within that frame. The number of bits in the frame number determines how many frames the system can address, and the number of bits in the offset determines the size of each frame.

Page numbers, frame numbers, and frame sizes are determined by the architecture, but are typically powers of two, allowing addresses to be split at a certain number of bits. For example, if the logical address size is 2^m and the page size is 2^n, then the high-order m-n bits of a logical address designate the page number and the remaining n bits represent the offset.

Consider the following micro example, in which a process has 16 bytes of logical memory, mapped in 4 byte pages into 32 bytes of physical memory. ( Presumably some other processes would be consuming the remaining 16 bytes of physical memory. )

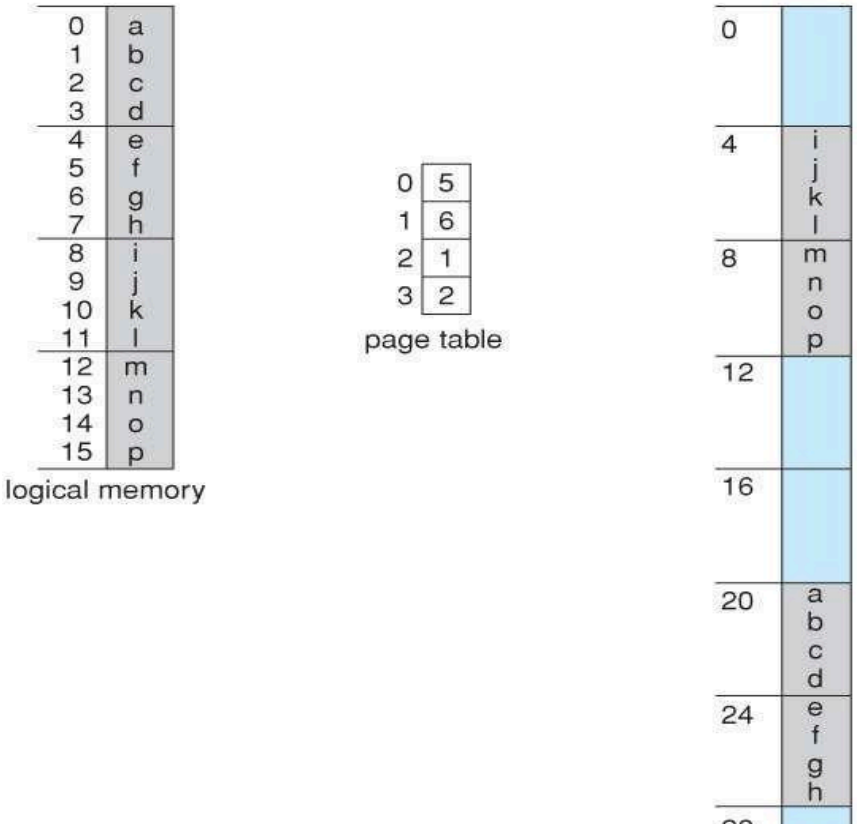


Figure :- Paging example for a 32-byte memory with 4-byte pages

### 5.2.3 Demand paging :

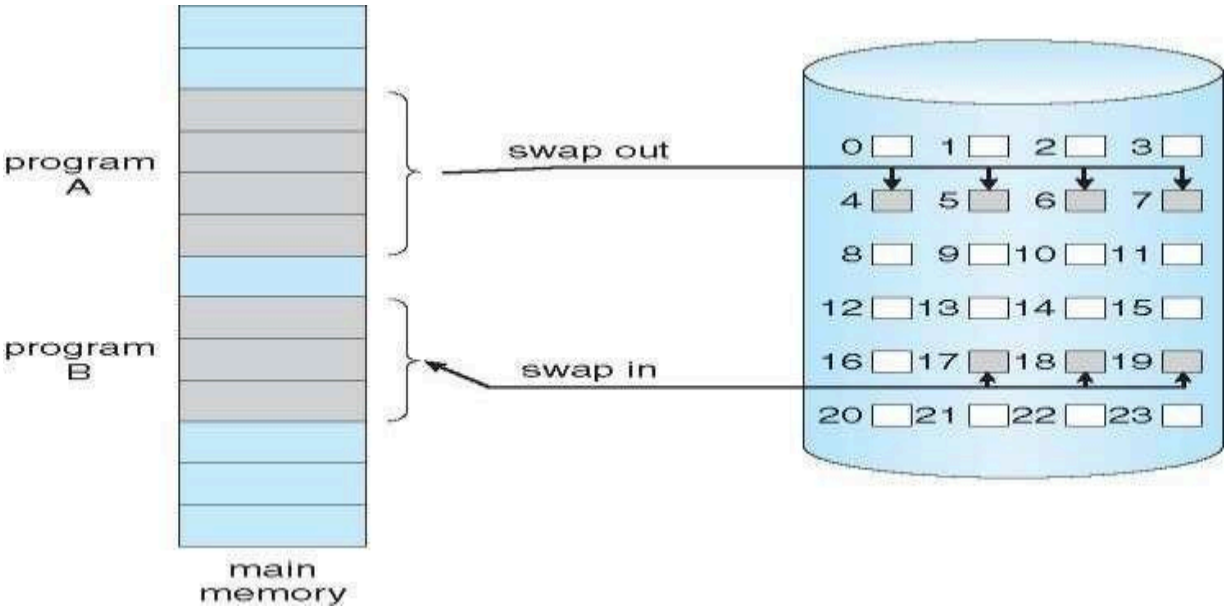
Consider how an executable program might be loaded from disk into memory.

One option is to load the entire program in physical memory at program execution time. However, a problem with this approach is that we may not initially *need* the entire program in memory. An alternative strategy is to load pages only as they are needed. This technique is known as demand paging and is commonly used in virtual memory systems.

With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

A demand-paging system is similar to a paging system with swapping where processes reside in 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, however, we use a lazy swapper.

A lazy swapper never swaps a page into memory unless that page will be needed.




With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk. The valid -invalid bit scheme can be used for this purpose. When this bit is set to "valid/' the associated page is both legal and in memory. If the bit is set to "invalid/' the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.

Thus, With each page table entry a valid–invalid bit is associated

(v ⇒ in-memory – memory resident, i ⇒ not-in-memory)

Initially valid–invalid bit is set to i on all entries

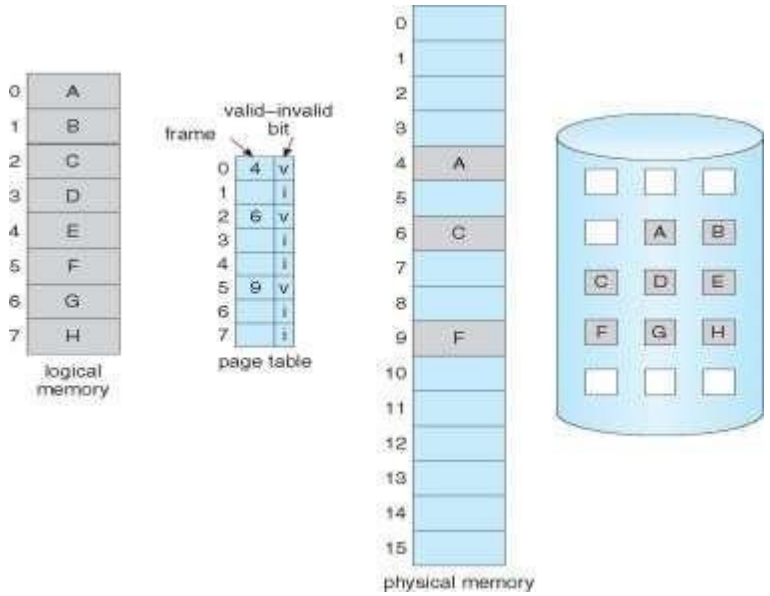



Fig: Page table when some pages are not in memory(valid-invalid bit)

### 5.2.4 Page Fault:

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. But what happens if the process tries to access a page that was not brought into memory? Access to a page marked invalid causes a Page fault. The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.

The procedure for handling this page fault is straightforward (following Figure ):

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
3. We find a free frame (by taking one from the free-frame list, for example).
4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

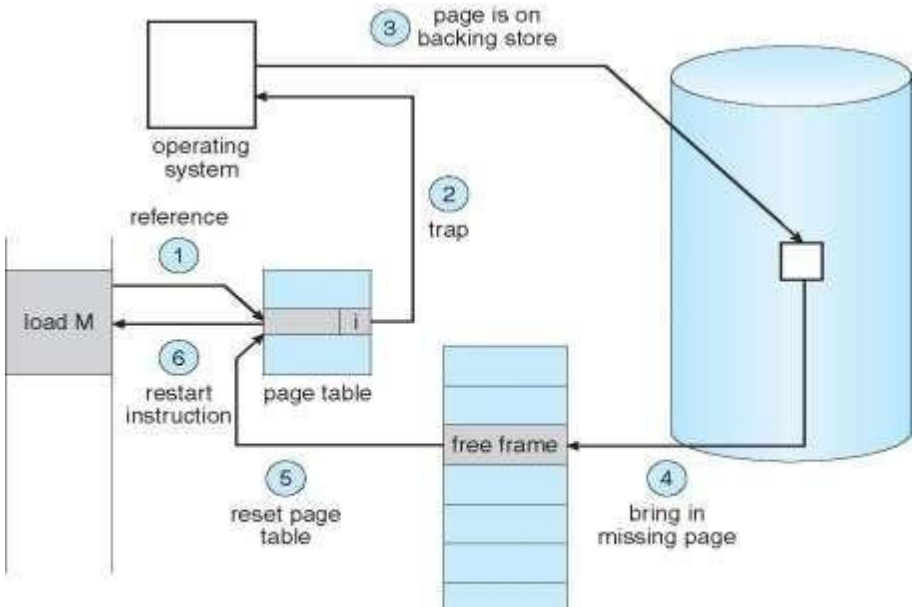


Fig : Steps to handle Page fault.

## 5.2 Page Replacement :

- Over-allocation of memory itself causes problem as : While a user process is executing, a page fault occurs. The operating system determines where the desired page is residing on the disk but then finds that there are *no* free frames on the free-frame list; all memory is in use.

**Then the question arises as :**

What Happens if There is no Free Frame and still another page is to be brought into memory?
**The solution to this problem** : OS could swap out unused pages on disk and swap in required pages in main memory.

**Again here question is** : Which page to replace ( swap out )

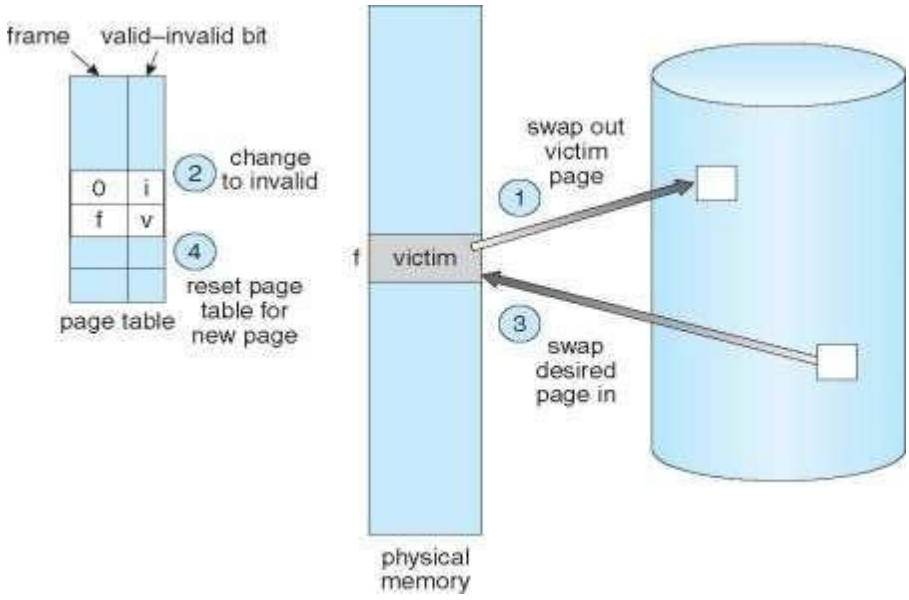
**Solution is** : it is decided by the Page Replacement Algorithm.

**Basic Page Replacement:**

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

   b. If there is no free frame, use a page-replacement algorithm to select

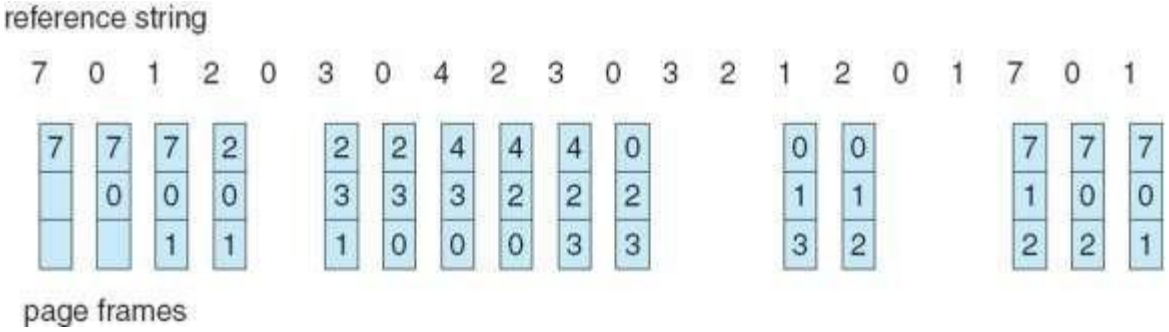
   c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.




**Page Replacement Algorithms:**

**A. FIFO Page Replacement:**

- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory.
- We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.
- For our example reference string, our three frames are initially empty. The first three references (7, 0, 1) cause page faults and are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since 0 is the next reference and 0 is already in memory, we have no fault for this reference and so on.



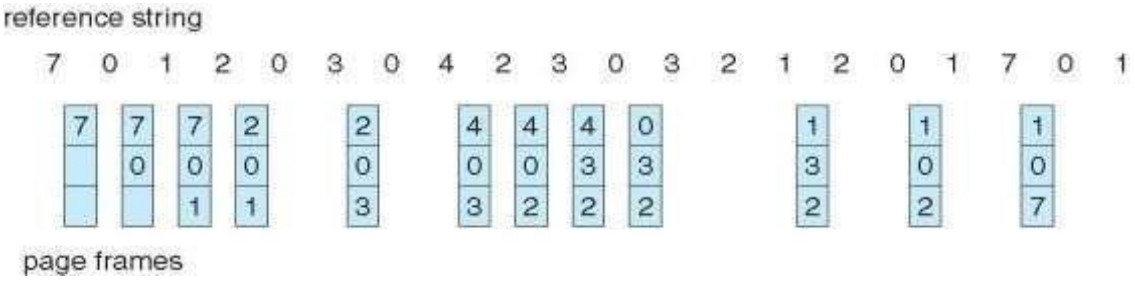

FIFO on above string with 3 frame size causes 15 page faults.

**Disadvantage:**

Adding more frames can cause more page faults! Thus suffers from **Belady's Anomaly**

### B. LRU (Least Recently Used) Page Replacement:

- LRU is : Replace the page that *has not been used* for the longest period of time.
  Use past knowledge rather than future. Associate time of last use with each page. Again here, it is assumed that main memory has three frames to solve following sum.




- Here LRU replacement causes 12 page faults which is much better than FIFO replacement with 15 page faults.