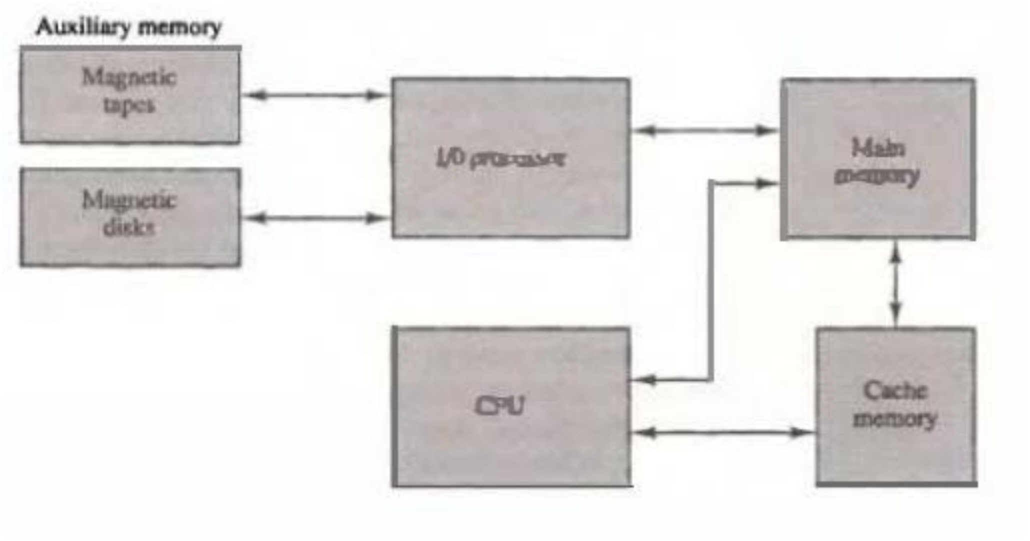
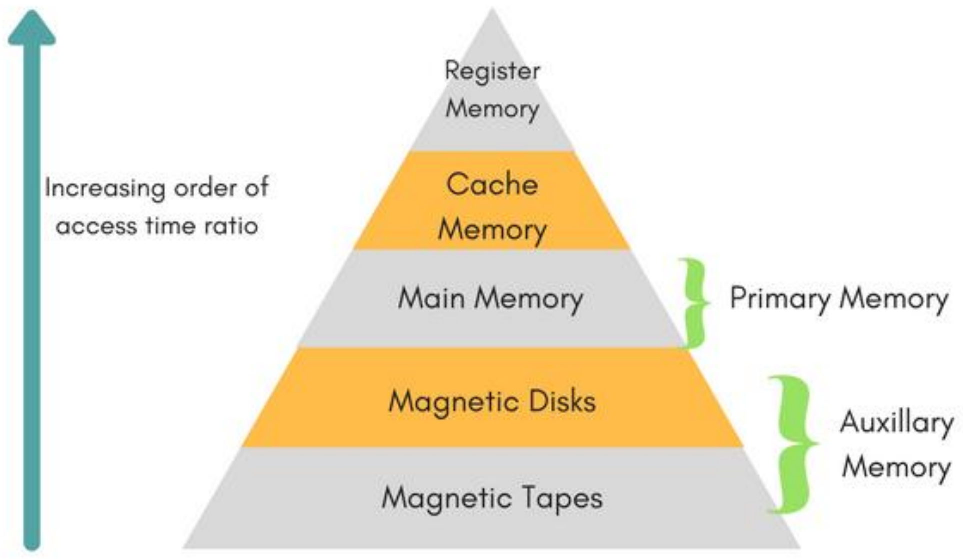
Memory Hierarchy

The memory unit is essential component in any digital computer since it is needed for storing instruction and data.



The memory unit that communicates directly with the CPU is called main memory. Devices that provides back up storage are called Auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data sets and other backup information.
Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

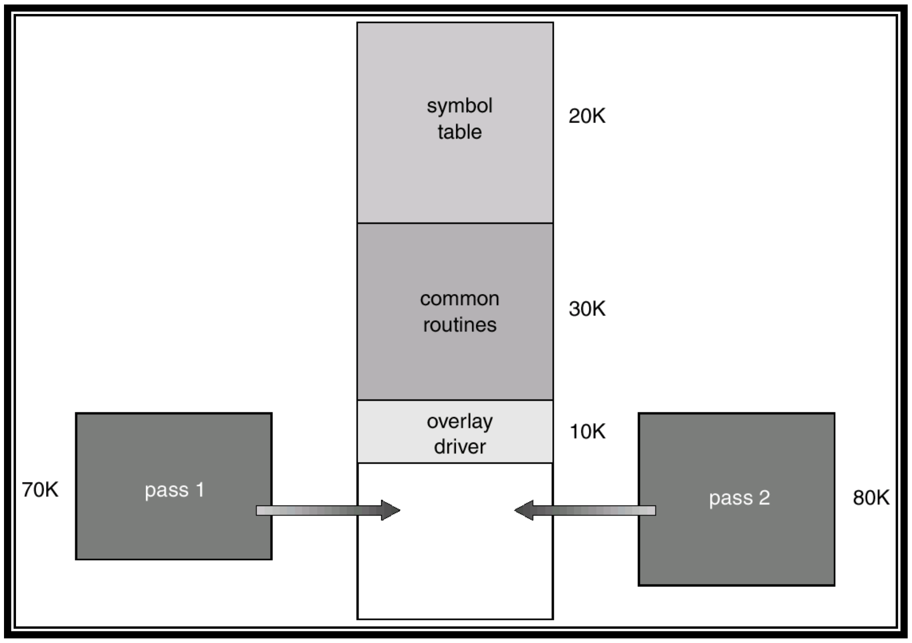


The total memory capacity of a computer can be visualized as being a hierarchy of components. The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic.

The part of the computer system that supervises the flow of information between auxiliary memory and main memory is called the ***memory management system****.*

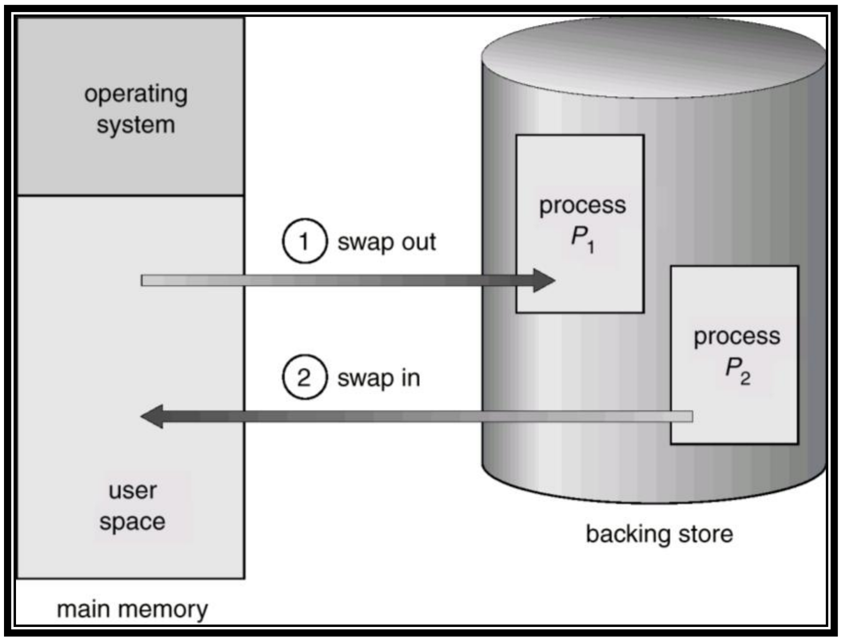
Overlays

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system, programming design of overlay structure is complex



Swapping

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- Backing store – 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.Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.



Fragmentation

In a computer storage system, as processes are loaded and removed from memory, the free memory space is broken into small pieces. 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. In this way memory space used inefficiently, so the capacity or performance of the system may degrade.

The conditions of the **fragmentation** depend on the system of memory allocation. In most of the cases, memory space is wasted.

Sometimes it happens that memory blocks cannot be allocated to processes due to their small size and memory blocks remain unused. This problem is known as **Fragmentation**.

Cause of fragmentation

User processes are loaded and removed from the main memory, processes are stored in the blocks of main memory. At the time of process loading and swapping there are many spaces left which are not capable to load any other process due to their size.

Due to the dynamical allocation of main memory processes, main memory is available but its space is not sufficient to load any other process.

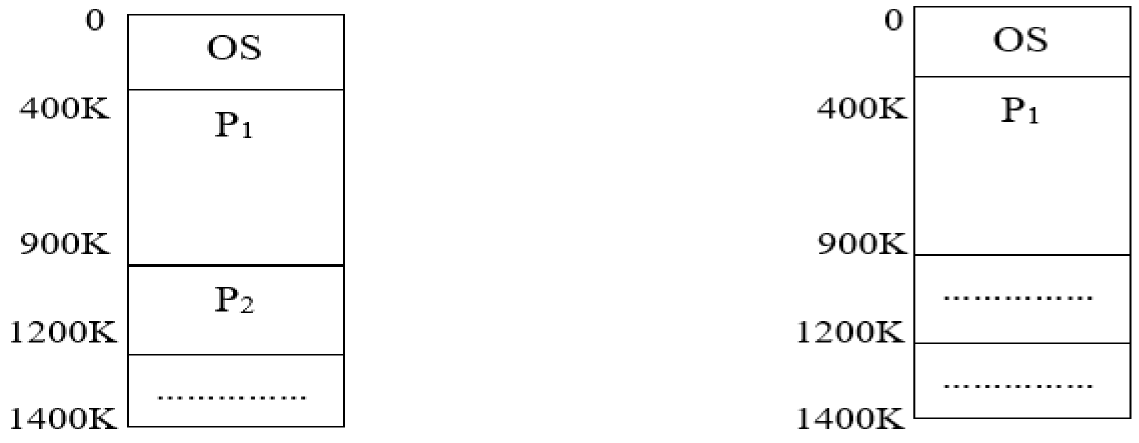
Types of fragmentation

1. External fragmentation
2. Internal fragmentation

External fragmentation

External fragmentation exists when adequate total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of fragments/partitions.

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.

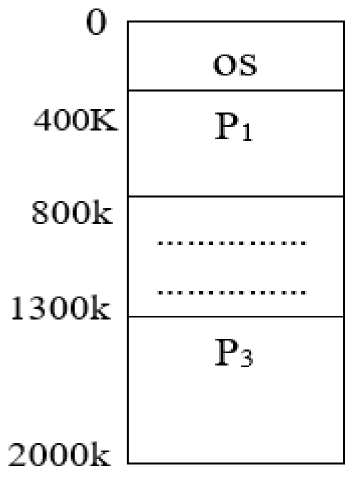


External fragmentation

Internal fragmentation

An approach is to allocate very small holes as part of the larger request. Thus the allocated memory may be larger than the requested memory.

The difference between these two numbers is **internal fragmentation -** the memory that is internal to any partition but is not being used.



Internal Fragmentation

There is a space of 500k. Suppose that next process request 450k. If we allocate the requested block, so there is a hole left which is 50k. This type of condition raises **internal fragmentation**.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

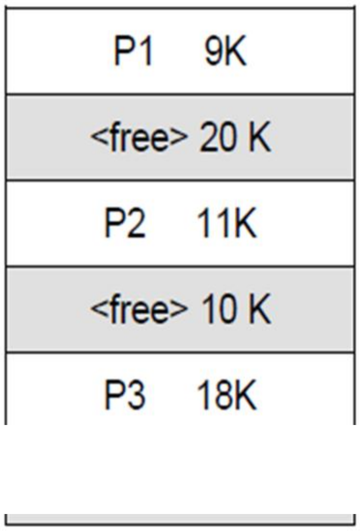
Difference between Internal fragmentation and External fragmentation:-

Internal fragmentation	External fragmentation
In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to method
Internal fragmentation happens when the method or process is larger than the memory.	External fragmentation happens when the method or process is removed.
The solution of internal fragmentation is best-fit block.	Solution of external fragmentation is compaction, paging and segmentation.
Internal fragmentation occurs when memory is divided into fixed sized partitions.	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, is called External fragmentation .

Q1. Consider the memory map given in the figure. If worst fit policy is to be applied, then what will be the memory map after arrival of the processes

P5=3K, P6=5K, P7=7K P8=6K.

Indicate if compaction is needed.



Q2. The following memory map is given for a computer system with variable partitioning memory management.

Event	**Required contiguous memory size**
P4 arrives	16K
P5 arrives	40K
P6 arrives	20K
P7 arrives	14K

Find and draw the resulting memory maps after the above job sequence is processed completely for **a.** first fit **b.** best fit **c.** worst fit. Indicate whenever a compaction is needed.

Solution for Dynamic Storage Allocation Problem:

- First Fit
- Best Fit
- Worst Fit

In **Partition Allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

Below are the various partition allocation schemes:

1. **First Fit**: In the first fit, the partition is allocated which is first sufficient block from the top of Main Memory.

This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory with space more than or equal to it's size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.

Job number	Memory requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	**Memory block size**	**Job number**	**Job size**	**Status**	**Internal Fragmentation**
10567	200 K	J1	20 K	Busy	180 K
30457	30 K			Free	30 K
300875	700 K	J2	200 K	Busy	500 K
809567	50 K	J4	50 K	Busy	None
Total available	980 K	Total used	270 K		710 K

As illustrated above, the system assigns J1 the nearest partition in the memory. As a result, there is no partition with sufficient space is available for J3 and it is placed in the waiting list.

Advantages of First-Fit Memory Allocation:

It is fast in processing. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.

Disadvantages of Fist-Fit Memory Allocation :

It wastes a lot of memory. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

2. Best Fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.

This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Job number	**Memory requested**
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	**Memory block size**	**Job number**	**Job size**	**Status**	**Internal Fragmentation**
10567	30 K	J1	20 K	Busy	10 K
30457	50 K	J2	50 K	Busy	None
300875	200 K	J3	200 K	Busy	None
809567	700 K	J4	500 K	Busy	200 K
Total available	980 K	Total used	770 K		210 K

As illustrated in above figure, the operating system first search throughout the memory and allocates the job to the minimum possible memory partition, making the memory allocation efficient.

Advantages of Best-Fit Allocation:
Memory Efficient. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To save memory from getting wasted, it is the best method.

Disadvantages of Best-Fit Allocation:
It is a Slow Process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.

3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.

4. Next Fit Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Although best fit minimizes the wastage space, it consumes a lot of processor time for searching the block which is close to the required size. Also, Best-fit may perform poorer than other algorithms in some cases. For example, see below example.

Example: Consider the requests from processes in given order 300KB, 25KB, 125KB and 50KB. Let there be two blocks of memory available of size 150KB followed by a block size 350KB.

Which of the following partition allocation schemes can satisfy above requests?

A) Best fit but not first fit.

B) First fit but not best fit.

C) Both First fit & Best fit.

D) Neither first fit nor best fit.

Solution: Let us try all options.

Best Fit:

300KB is allocated from block of size 350KB. 50 is left in the block.

25KB is allocated from the remaining 50KB block. 25KB is left in the block.

125KB is allocated from 150 KB block. 25KB is left in this block also.
50KB can’t be allocated even if there is 25KB + 25KB space available.

First Fit:

300KB request is allocated from 350KB block, 50KB is left out.

25KB is be allocated from 150KB block, 125KB is left out.
Then 125KB and 50KB are allocated to remaining left out partitions.
So, first fit can handle requests.

So option B is the correct choice.

Question:- Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Solution:

First-fit:

212KB is put in 500KB partition
417KB is put in 600KB partition
112KB is put in 288KB partition (new partition 288KB = 500KB - 212KB)
426KB must wait

Best-fit:

212KB is put in 300KB partition
417KB is put in 500KB partition
112KB is put in 200KB partition
426KB is put in 600KB partition

Worst-fit:

212KB is put in 600KB partition
417KB is put in 500KB partition
112KB is put in 388KB partition (new partition 388KB = 600KB - 212KB)
426KB must wait

In this example, best-fit turns out to be the best.

Question:- Given five memory partitions of 500 KB, 350 KB, 250 KB, 420 KB, and 450 KB (in order), how would the worst-fit algorithms place processes of 325 KB, 150 KB, 400 KB, and 375 KB (in order)?

Solution:

325KB is put in 500KB partition

150KB is put in 450KB partition

400KB is put in 420KB partition

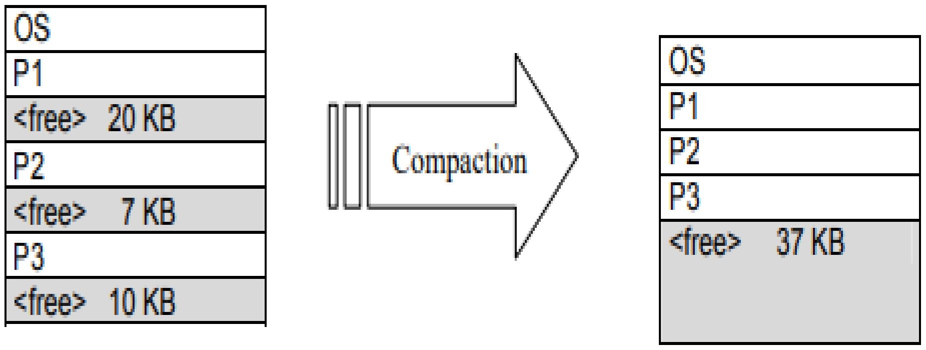
375KB must wait

Contiguous Memory Allocation generates external fragmentation. External Fragmentation is the problem where sufficient or adequate memory space is available to store data but this space is not in Contiguous Memory locations. This problem of External fragmentation can be removed by:

a. Compaction
b. Paging
c. Segmentation

Compaction

Compaction is a method to overcome the external fragmentation problem. All free blocks are brought together as one large block of free space. Compaction requires dynamic relocation. Certainly, compaction has a cost and selection of an optimal compaction strategy is difficult. One method for compaction is swapping out those processes that are to be moved within the memory, and swapping them into different memory locations.



Important points

- Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes.
- In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.
- Compaction refers to combining all the empty spaces together and processes.
- Compaction helps to solve the problem of external fragmentation, but it requires too much of CPU time.
- It moves all the occupied areas of store to one end and leaves one large free space for incoming jobs, instead of numerous small ones.

In compaction, the system also maintains relocation information and it must be performed on each new allocation of job to the memory or completion of job from memory.

Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non–contiguous.

Logical Address or Virtual Address (represented in bits): An address generated by the CPU.

Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program

Physical Address (represented in bits): An address actually available on memory unit.

Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses.

Example:

- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = $\log_2 2^{27}$ = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^4 * 2^{20}$ words, then Physical Address = $\log_2 2^{24}$ = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

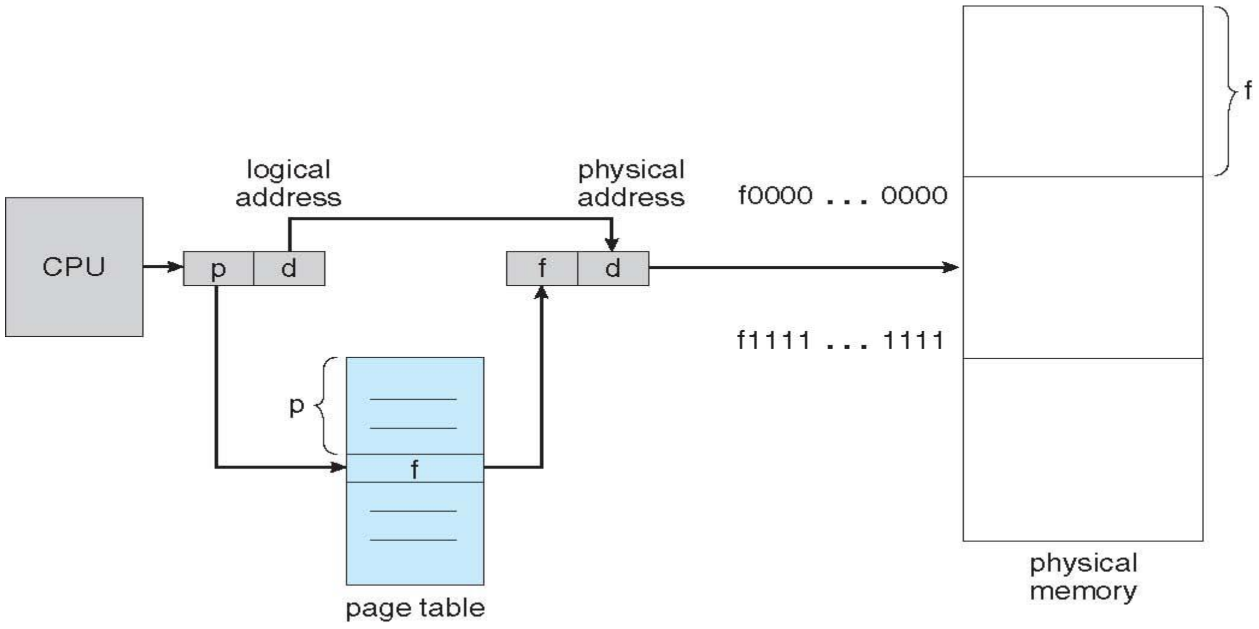
- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

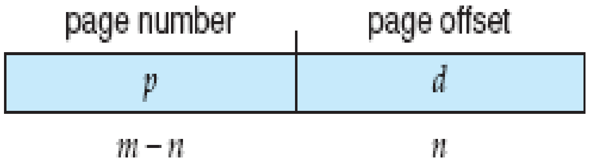
- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words

- Page size = frame size = 1 K words (assumption)

The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called ***frames*** and breaking logical memory into blocks of the **same** size called ***pages***. When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or backing store). The backing store is divided into fixed sized blocks that are of the same size as that of the memory frames.

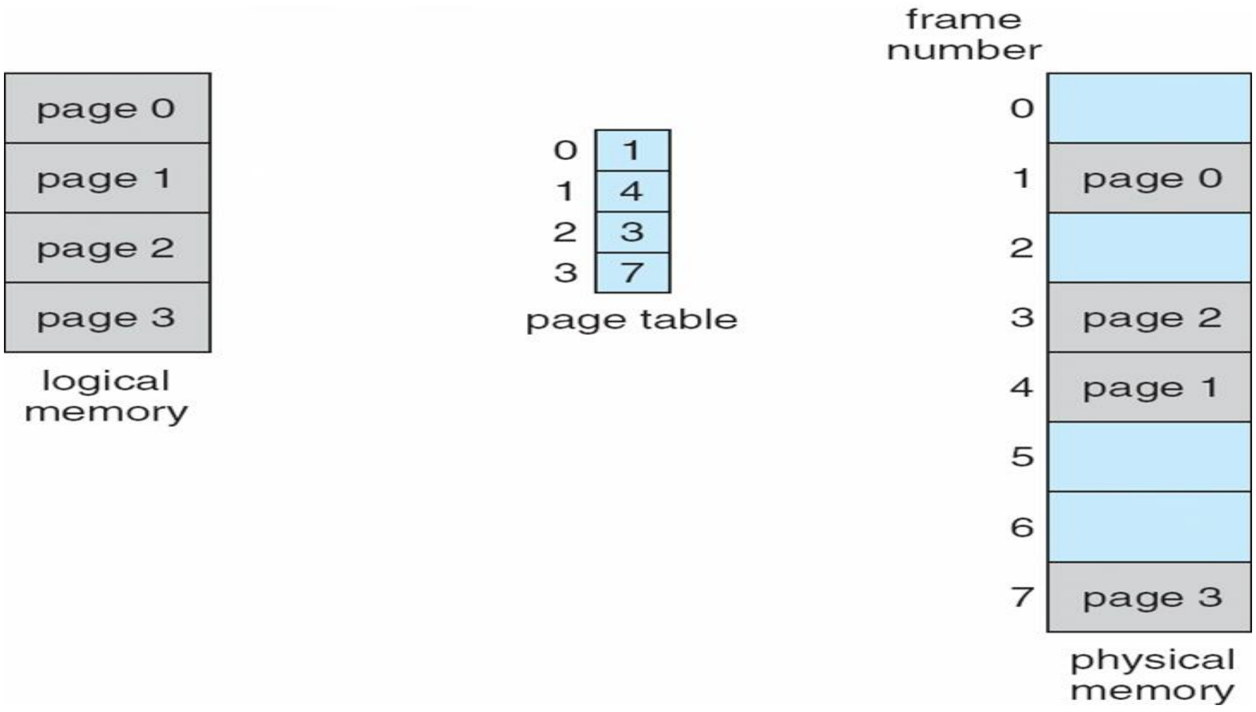


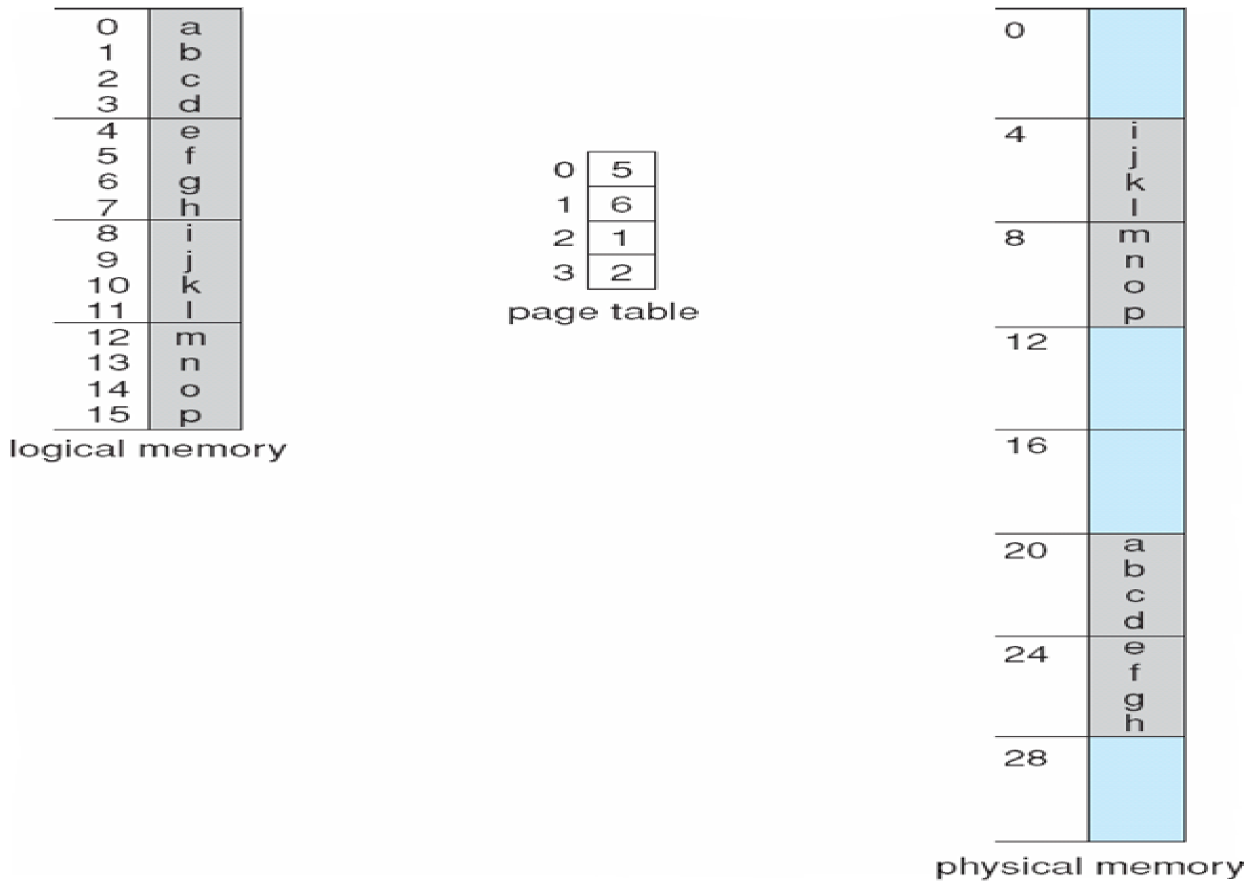
- Every address generated by the CPU is divided into 2 parts: a ***page number*** (p) and a ***page offset*** (d).The page number is used as an index into a ***page table***. The page table contains the base address of each page in physical memory. This page address is combined with page offset to define the physical memory address that is sent to the memory unit.
- The page size is defined by the hardware. The size of the page is typically a power of 2, varying between 512 bytes and 16MB per page, depending on the computer architecture. **The selection of power of 2 as a page size makes the translation of logical address into a page number and page offset particularly easy.** If the size of logical address space is 2^m, and page size is 2^n, then the high order "**m-n**" bits of a logical address space designate the page number, and the "**n**" low order bits designate the page offset. Thus the logical address is shown below where "**p**" is an index into the page table and "**d**" is the replacement within the page.



Page Table: keeps track that which page (logical memory) is mapped to which fame (Physical Memory).

From the below Page Table : Page-0 is mapped to Frame-1, Page-1 is mapped to Frame-4, Page-2 is mapped to Frame-3 and Page-3 is mapped to Frame-7.





PHYSICAL ADDRESS = Frame Number * size of the frame + Displacement

- **Size of the frame: offset or size of the page (In above figure page size =4)**
- **Displacement : position of the word within page i.e**

Page-0 contains a, b, c, d having position (displacement) 0,1,2,3 respectively.

Page-1 contains e, f, g, h having position (displacement) 0,1,2,3 respectively.

And so on…..

Consider the memory as shown in previous slide. Here, in the logical address n=2 and m=4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the user's view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 [= (5*4) +0].Logical address 3 (page 0, offset 3) maps to physical address 23[= (5*4) +3]. Logical address 4 is page 1, offset 0; according to page table, page 1 is mapped to frame 6 .Thus, logical address 4 maps to physical address 24[= (6*4) + 0].Logical address 13 maps to physical address 9.

Usually a page table entry is 4 bytes long, but that size can vary as well. A 32 bit entry can point to one of the 2^{32} physical page frames. If the frame size is 4 KB, then a system with 4-byte entries can address 2^{44} bytes (16 TB) of physical memory.

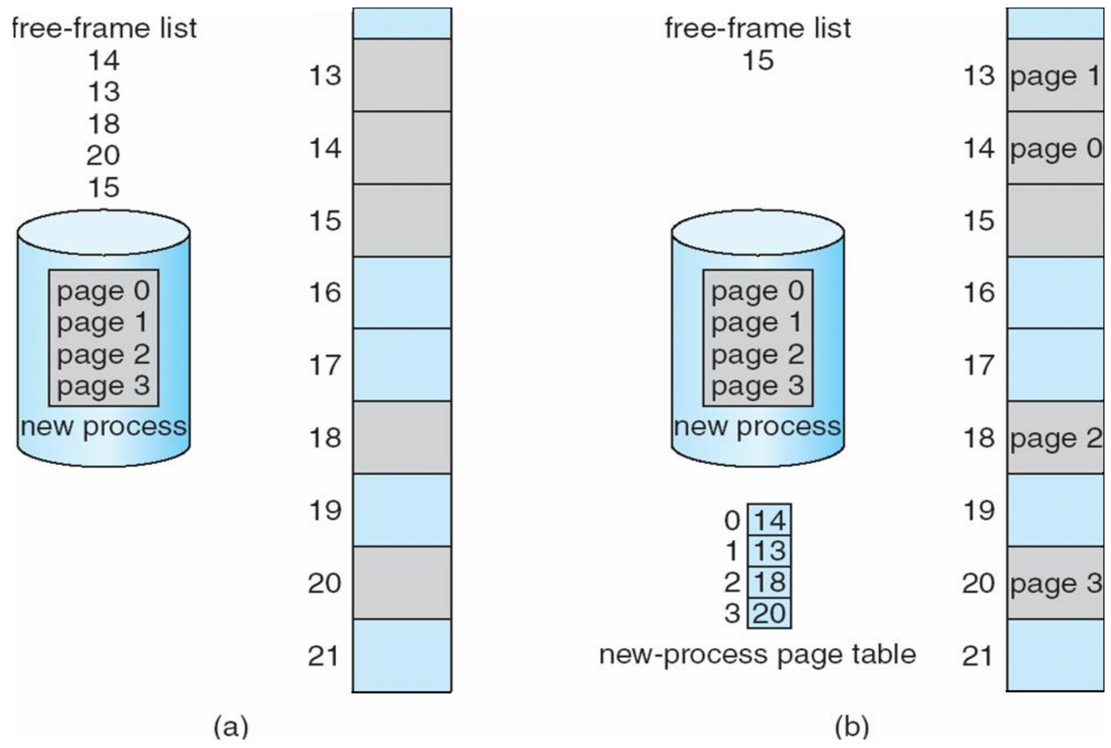
When we use a paging scheme, we have **no external fragmentation**: any free frame can be allocated to the process that needs it. However, we may have **some internal fragmentation[unused space in a fixed size block/partition]**. If the memory requirements of a process do not happen to coincide with page boundaries, the last frame allocated may not be completely full.

For example, if pages are 2048 bytes, a process of 72,766 bytes would need 35 pages plus 1086 bytes. It would be allocated 36 frames, resulting in an internal fragmentation of 2048 -1086 = 962 bytes.

In the worst case, a process would need "**n**" pages plus 1 byte. It would be allocated "**n+1**" frames, resulting in internal fragmentation of at most an entire frame. If process size is independent of page size, we expect **internal fragmentation to average one half page per process**. This consideration suggests that small page sizes are desirable. However, overhead is involved in each page table entry, and this overhead is reduced as the size of the page increases.

- 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**.

- In addition, the operating system must be aware that user processes operate in user space, and all logical addresses must be mapped to produce physical addresses. **The operating system maintains a copy of the page table for each process, just as it maintains a copy of the instruction counter and register contents**. This copy is used to translate logical addresses to physical addresses whenever the operating system must map a logical address to a physical address manually. It is also used by the CPU dispatcher to define the hardware page table when a process is to be allocated the CPU. Paging therefore increases the context-switch time.



Example: Consider a logical address space of 64 pages each of 1024 words mapped onto physical memory of 32 frames.

a. How many bits are there in Logical address Space?
b. How many bits are there in physical address space?

Solution:

Logical Address		Physical Address	
Page No. (p)	Offset(d)	Frame No.(f)	Offset(d)

a. 64 pages = 2^6 pages that is 6 bits are required for Page no.
   Page Size is 1024 = 2^{10} that is 10 bits are required for Page size

   Logical Address = no of bits in page no + no of bits in Page size
   = 6+10 = 16 bits

b. 32 Frame = 2^5 Frames that is 5 bits are required for Frame no.
   Frame Size = Page Size =1024 = 2^{10} that is 10 bits are required for Frame size

   Physical Address Space = no of bits in Frame no + no of bits in Frame size
   = 5+10 = 15 bits

Paging Hardware with TLB

*(*Translation Look aside Buffer)

In Operating System (Memory Management Technique : Paging), for each process page table will be created, which will contain Page Table Entry (PTE). This PTE will contain information like frame number (The address of main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit etc). This page table entry (PTE) will tell where in the main memory the actual page is residing.
Now the question is where to place the page table, such that overall access time (or reference time) will be less.

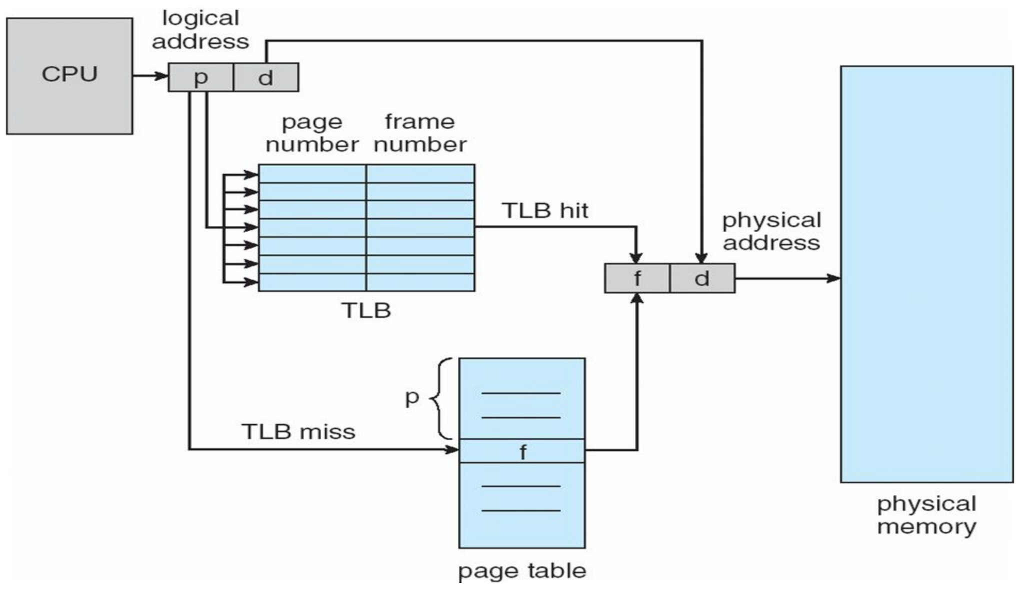
The problem initially was to fast access the main memory content based on address generated by CPU (i.e logical/virtual address). Initially, some people thought of using registers to store page table, as they are high-speed memory so access time will be less.

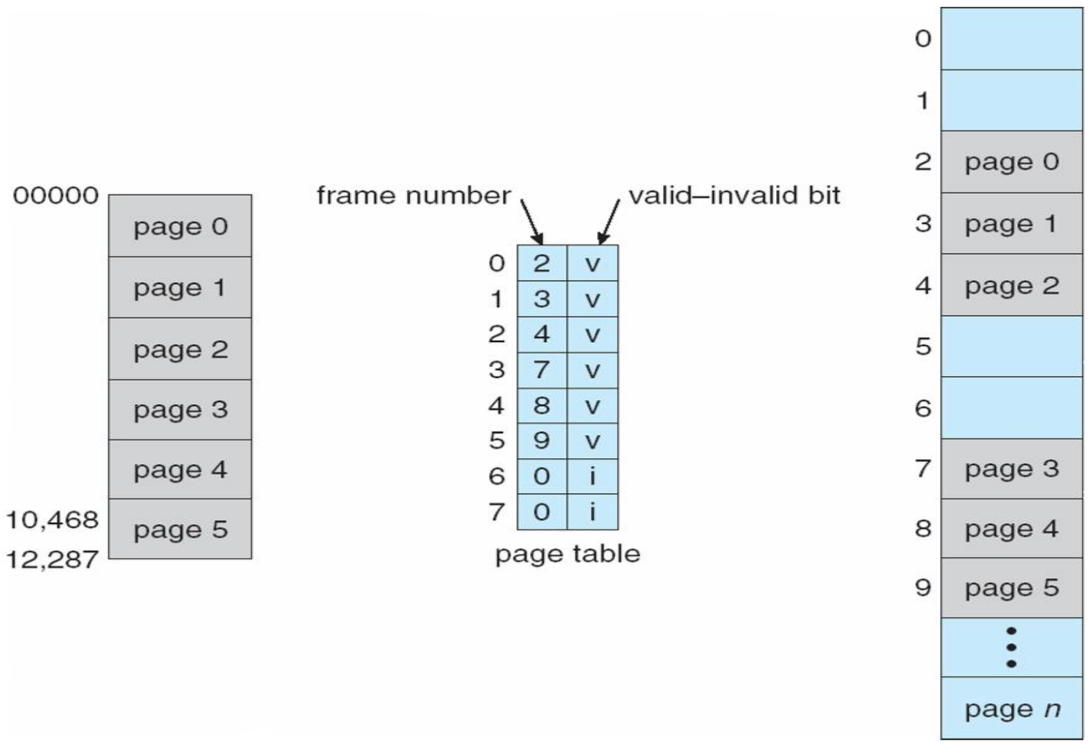
The idea used here is, place the page table entries in registers, for each request generated from CPU (virtual address), it will be matched to the appropriate page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big (lets say this page table contains 1M entries), so registers may not hold all the PTE's of Page table. So this is not a practical approach.

To overcome this size issue, the entire page table was kept in main memory. but the problem here is two main memory references are required:

1. To find the frame number
2. To go to the address specified by frame number

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.





Hit Ratio: The percentage of time that particular Page is found in the TLB (Translation Look Aside Buffer) is called hit Ratio.

Miss Ratio: If specific Page is not found in Cache is called Miss.

TLB (Translation Look Aside Buffer): is small, fast and special Hardware Cache.

Effective Access Time = hit ratio * (TLB access time + Main memory access time) + (1 – hit ratio) * (TLB access time + 2 * main memory time)

https://www.youtube.com/watch?v=kHhZUphoANI

Example-1 : Find the Average Effective Memory Access Time when 80 percent is the hit ratio and it take 20 nanoseconds to search TLB and 100 nanoseconds to access memory (**Hint :** then take 120 nanoseconds to access mapped memory). In Case of Miss, It also takes 100 nanosecond to access frame number. The Total time taken in case of Miss is (20 + 100 + 100 = 220 nanosecond)

Solution:

Average Access Time = 0.80 * (20+ 100) + 0.20 * (20 + 100 + 100)

= 0.80 * 120 + 0.20 * 220

= 96 + 44 = **140 nanoseconds**

Example:-2

Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is _________.

Solution:

Effective Access Time = hit ratio * (TLB access time + Main memory access time) +
(1 – hit ratio) * (TLB access time + 2 * main memory time)

= 0.6*(10+80) + (1-0.6)*(10+2*80)
= 0.6 * (90) + 0.4 * (170)
= 122

Example:-3

Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 50 milliseconds to search the TLB and 400 milliseconds to access the physical memory. If the TLB hit ratio is 0.9, the effective memory access time (in milliseconds) is _________.

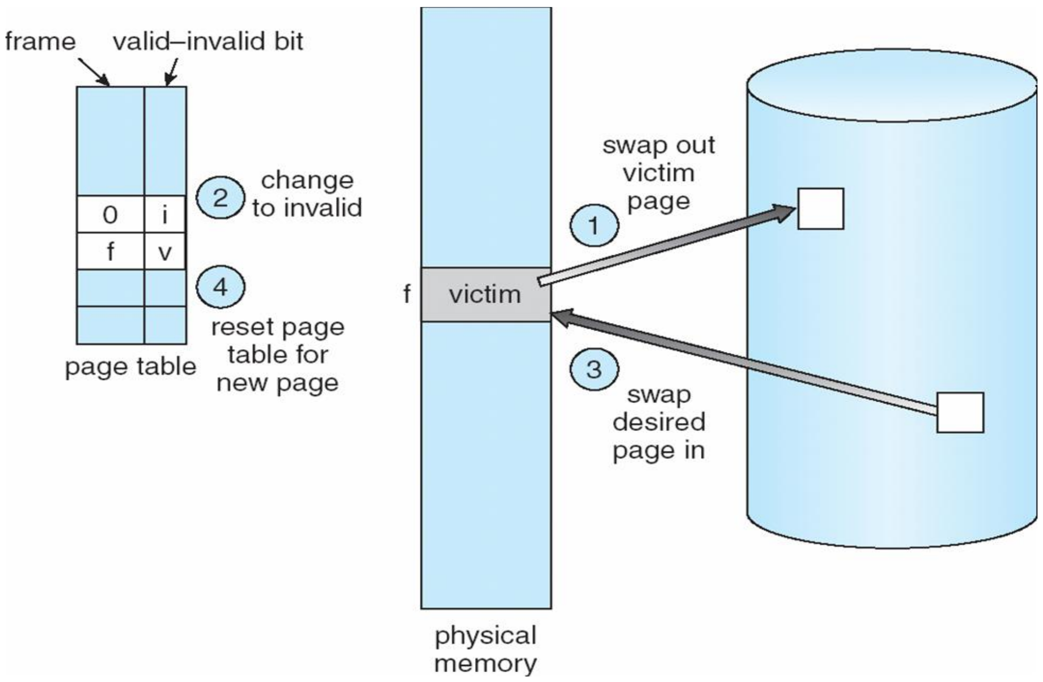
TLB Access Time= 50 milliseconds

Main Memory Access Time = 400 milliseconds

Solution:

????

Page Replacement



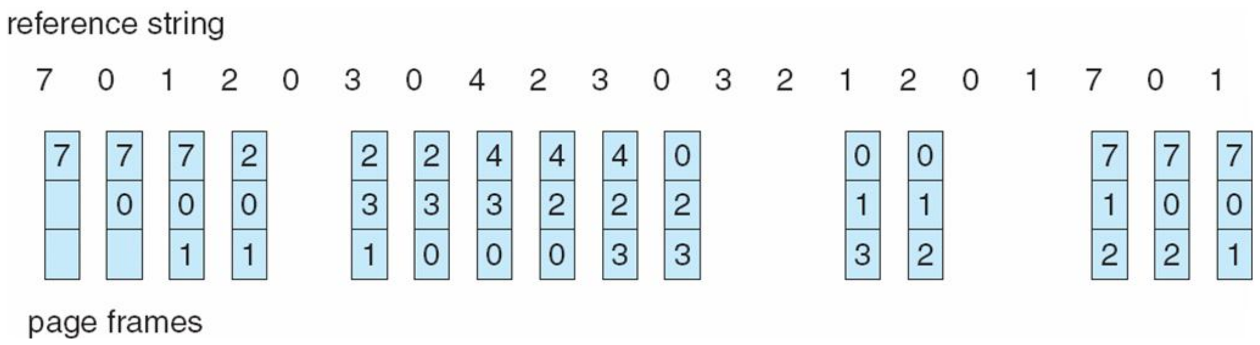
- ➢ Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- ➢ Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk
- ➢ Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory
- ➢ To choose the victim page[right now discussion is on **local replacement** only]
- ➢ Want lowest page-fault rate[means choosing a victim that should not give rise to page fault itself.]
- ➢ Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- ➢ In all our examples, the reference string is

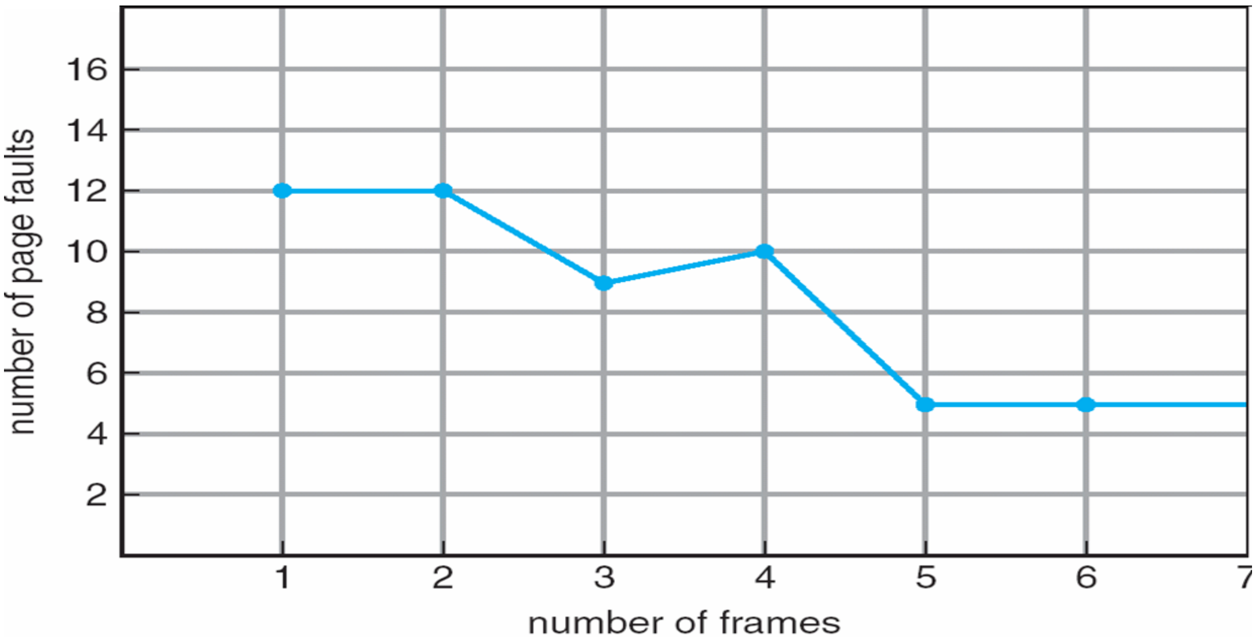
1, 2, 3, 4, 5, 3, 4 , 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

Number of frames allocated to this corresponding process =4

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

FIFO Illustrating Belady's Anomaly





Questions 1: A virtual memory has an address space 18K words, a memory space has 9K words and block sizes of 3K words. The following reference generated by CPU

3 4 2 6 4 7 1 3 2 6 3 5 1 2 3

Show each page reference change if the replacement algorithm used is (a) FIFO (b)LRU © Optimal. Also state which algorithm is better on the basis of page fault?

Question 2: A virtual memory has a page size of 1K words. There are eight pagesand four blocks. The memory page table contains the following entries:

Page	**Block/Frame**
0	3
1	1
4	2
6	0

Make a list of all virtual addresses (In decimal) that will cause a page fault if used by the CPU.

Question 3: A virtual memory system has an address space of 8K words, a memory space of 4K words, and page and block sizes of 1K words The following page reference changes occur during given time interval.

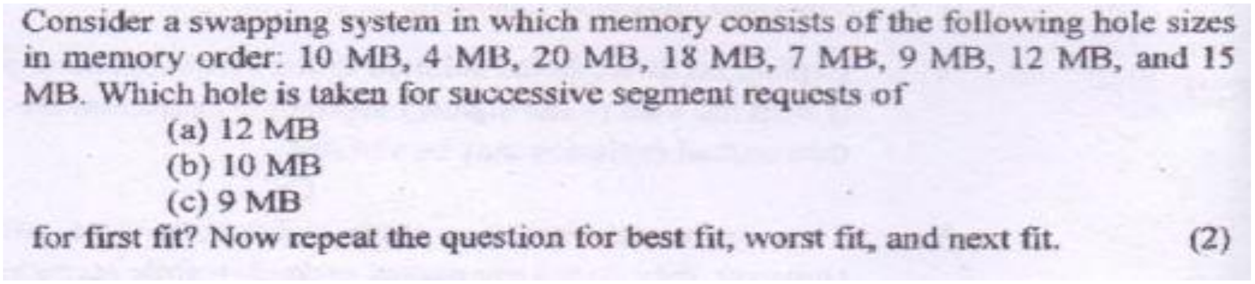
4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

Determine the four pages that are resident In main memory after each page reference change if the replacement algorithm used Is (a) FIFO; (b) LRU.

Question 4: An address space is specified by 24 bits and the corresponding memory space by 16 bits. How many words are there in the address space. How many words are there in the memory space. If a page consists of 2K words, how many pages and blocks are there in the system?

Question 5: The logical address space in a computer system consists of 128 segments each segment can have up to 32 pages of 4K words In each. Physical memory consists of 4K block of 4K words In each. Formulate the logical and physical address formats.

Question 6:

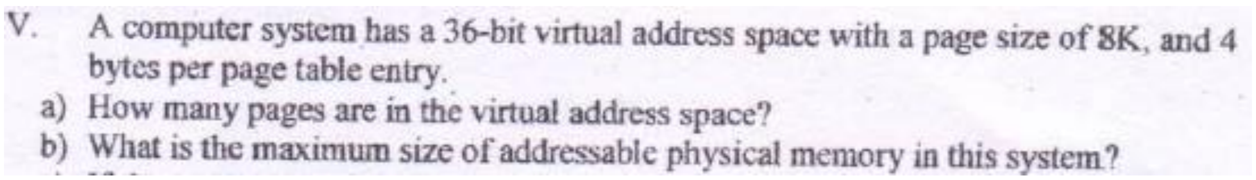


Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 MB, 4 MB, 20 MB, 18 MB, 7 MB, 9 MB, 12 MB, and 15 MB. Which hole is taken for successive segment requests of

(a) 12 MB
(b) 10 MB
(c) 9 MB

for first fit? Now repeat the question for best fit, worst fit, and next fit. (2)

Question7:



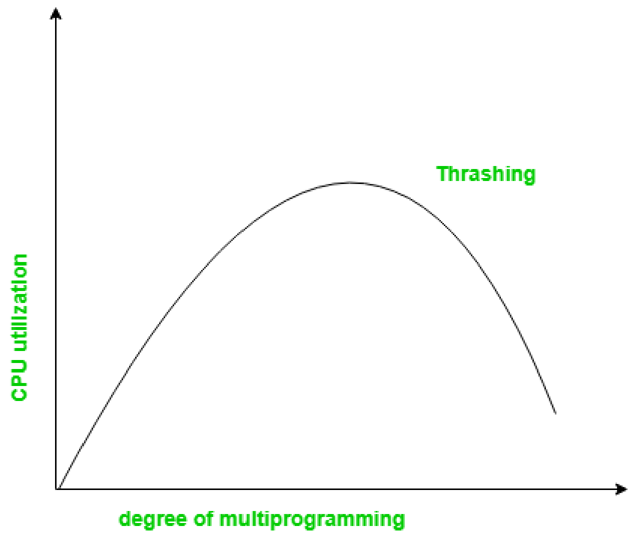
V. A computer system has a 36-bit virtual address space with a page size of 8K, and 4 bytes per page table entry.

a) How many pages are in the virtual address space?

b) What is the maximum size of addressable physical memory in this system?

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 utilization would fall drastically. The long-term scheduler would then try to improve the CPU utilization 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.