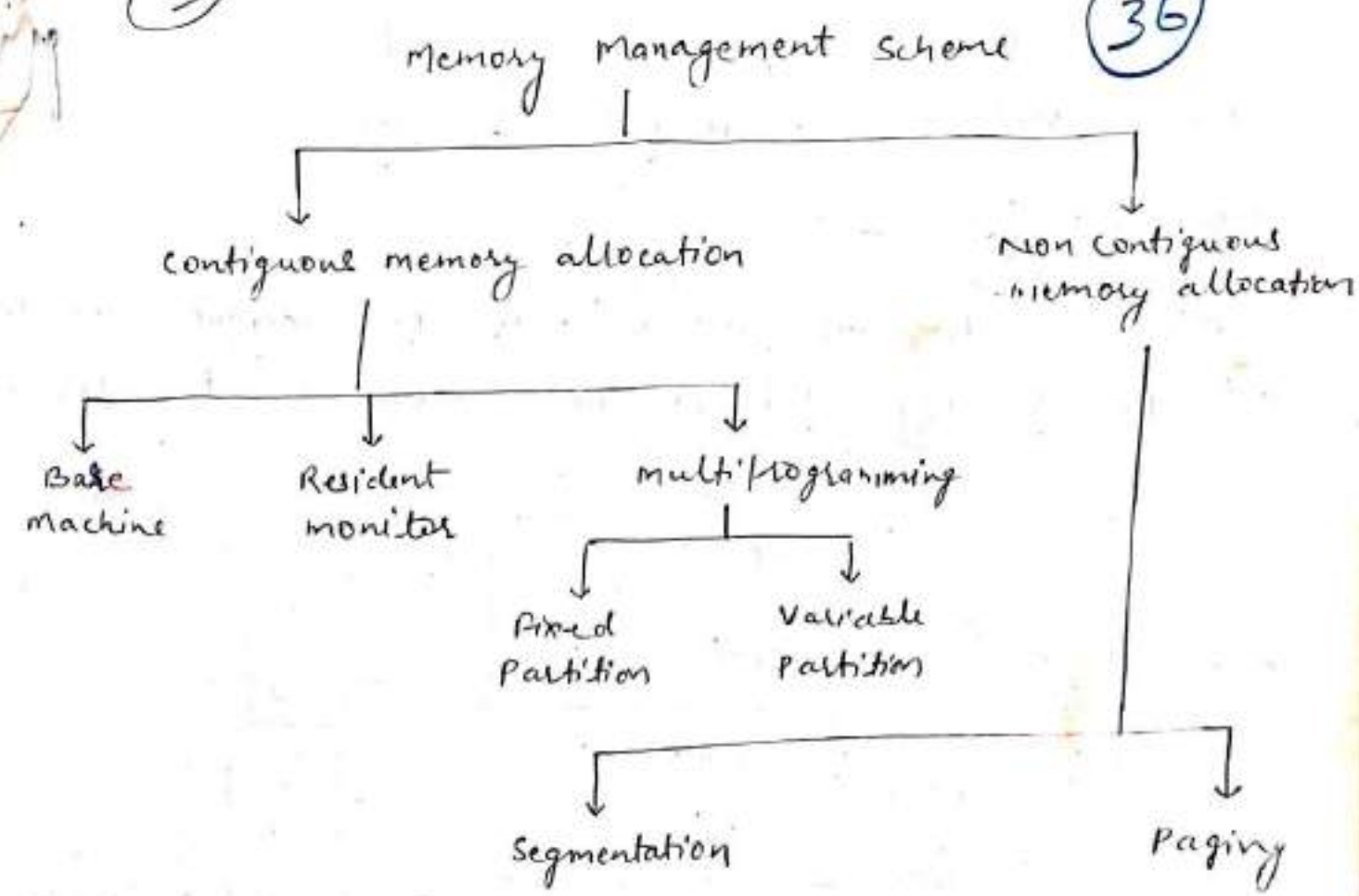
UNIT- 4

III rd

(3)

(36)



- Contiguous allocation means that each logical object is placed in a set of memory locations with strictly consecutive addresses.

- Non-contiguous allocation implies that a single logical object may be place in non consecutive sets of memory location.

Bare machine (Single User) :→

- In this method user has complete control over the memory while in other methods operating system has complete control over memory.

The advantages of using this method are:

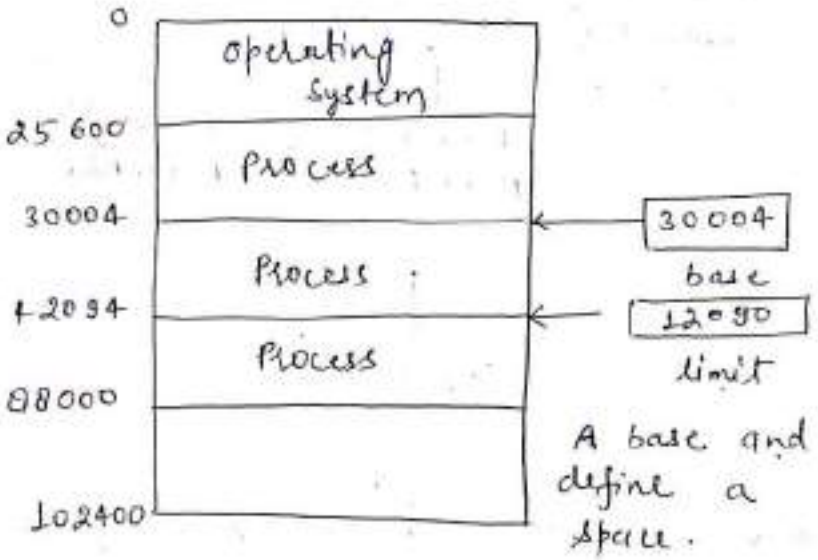
(i) Flexible

(ii) No hardware support is required

(iii) No software is needed for OS.

Disadvantages :-

- As OS has no control over the memory, it does not provide any services and resources to user.

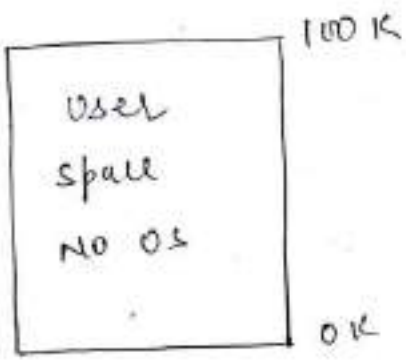


A base and a limit register define a logical address space.

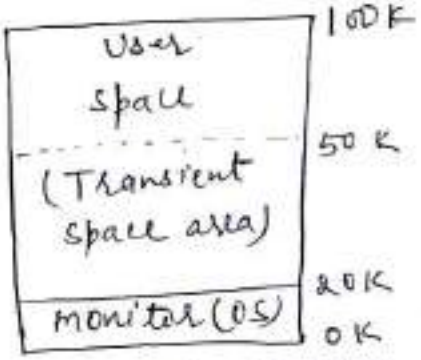
- The base register holds the smallest legal physical memory address.
- The limit register specifies the size of the range.
- For example if the base register holds 300040 and limit register is 120900 then program can legally access all addresses from 300040 through 420940 (inclusive).

2. Resident monitor (single process monitor) :→

- In this memory is divided into two contiguous areas.
- One of them is usually the permanently allocated to the resident portion of the O.S. (monitor).
- The remaining memory is allocated to the so-called transient processes.



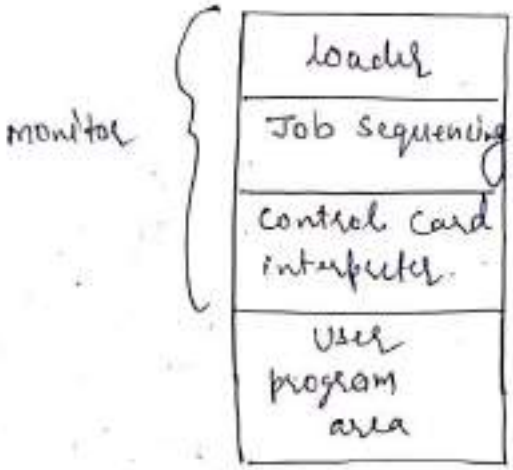
Bare machine



Resident Monitor

- Transient processes are loaded and executed one at a time in response to user commands.
- When a transient process is completed, the OS may load another one for execution.
- Resident monitor was introduced to overcome idle time (in finding one process as stopped and starting another process.)
- Resident monitor is a small program and was created to transfer control automatically from one job to the next.

- When the computer was turned on, the resident monitor was invoked, and it would transfer control to a program.
- When the program terminated it would return control to the resident monitor, which would then go on to the next program.



memory layout for a resident monitor

- But how would the resident monitor know which program to execute?
- control cards were introduced to provide this information directly to the monitor.
- ~~In addition to the resident monitor indicating what program to run.~~
- In addition to the program or data for a job, the programmer included the control cards, which contained directives to the resident monitor indicating what program to run.

for example a normal user program might require one of three programs to run -

(a) The FORTRAN compiler (FTN)
(b) The assembler (ASM)
(c) The user's program (RUN)

We would use a separate control card for each of these -

$ FTN - Execute the FORTRAN compiler
$ ASM - Execute the assembler
$ RUN - Execute the user program

These cards tell the resident monitor which program to run.

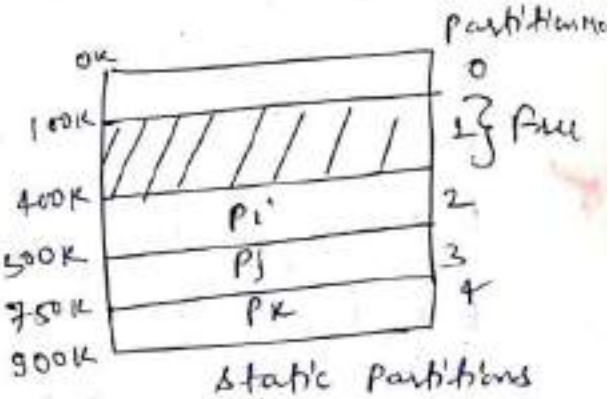
A resident monitor thus has several identifiable parts -

- The control card Interpreter :→
  - Responsible for reading and carrying out the instructions on the cards at the point of execution.
- The loader :— is invoked by the control card interpreter to load system programs and application programs into memory at intervals.
- The device drivers :→ are used by both the control card interpreter and the loader for the system's I/O devices to perform I/O.

Multiprogramming with fixed Partitions

- In this method for allocating memory, memory is divide into several fixed sized partitions.
- Suppose the memory for allocation is of the size say 500K and is divided into 4 partitions of different but fixed sizes as -

Partition A (125K)
Partition B (150K)
Partition C (100K)
Partition D (125K)



Static Partitions

- Each partition may contain exactly one process.
- Thus the degree of multiprogramming is bound by the number of partitions.
- In this 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.
- It is primarily used in batch environments and is no longer in use.
- In the fixed partition scheme, the operating system keeps a table indicating which part of

Memory are available and which are occupied.

- This data structure is called as Partition Description Table (PDT).

Partition No.	Partition base	Partition size	Partition status
0	0K	100K	Allocated
1	100K	300K	Free
2	400K	100K	Allocated
3	500K	250K	Allocated
4	750K	150K	Allocated

Partition Description Table

- When a process arrives and needs memory, we search for hole large memory enough for this process.
- If we find one, we allocate only as much memory as is needed, keeping the rest available to satisfy future requests.
- As processes enter the system they are put into an input queue.
- The operating system takes into account the memory requirements of each processes and amount of available memory space in determining which processes are

allocated memory

- when a process is allocated space, it is loaded into memory and it can then compete for the CPU.
- At any given time we have 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 back and placed in the set of holes.
- If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole.
- At this point the system may need to check whether there are processes waiting for memory and newly freed and recombined memory could satisfy the demand of any of waiting processes.
- 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

(i) First fit :→ • Allocate the first hole that is big enough.
- Searching can start either at the begining of the set of holes or where the previous first fit search ended.

(ii) Best fit :→ • Allocate the smallest hole that is big enough.
- We must search the entire list unless the list is ordered by size.

(iii) Worst fit :→ • Allocate the largest hole
- We must search the entire list, unless it is sorted by size.
- This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best fit approach.

- Both first fit and best fit are better than worst fit
- Neither first fit nor best fit is clearly better than the other but first fit is generally faster.

Fragmentation

- Both the first fit and best fit strategies suffer from external fragmentation.

- External fragmentation exists when there is enough total memory space to satisfy a request, but the available space are not contiguous, storage is fragmented into a large number of small holes.

- Memory fragmentation can be internal as well as external.
- Consider a multiple partition allocation scheme with a hole of 18,464 bytes.
- Suppose that next process requests 18,462 bytes.
- If we allocate exactly the requested block, we are left with a hole of 2 bytes.
- The overhead to keep track of this hole will be substantially larger than the hole itself.
- The general approach to avoiding this problem is to break the physical memory into fixed sized blocks and allocate memory in units based on block size.
- With this approach the memory allocated to a process may be slightly larger than the requested memory.

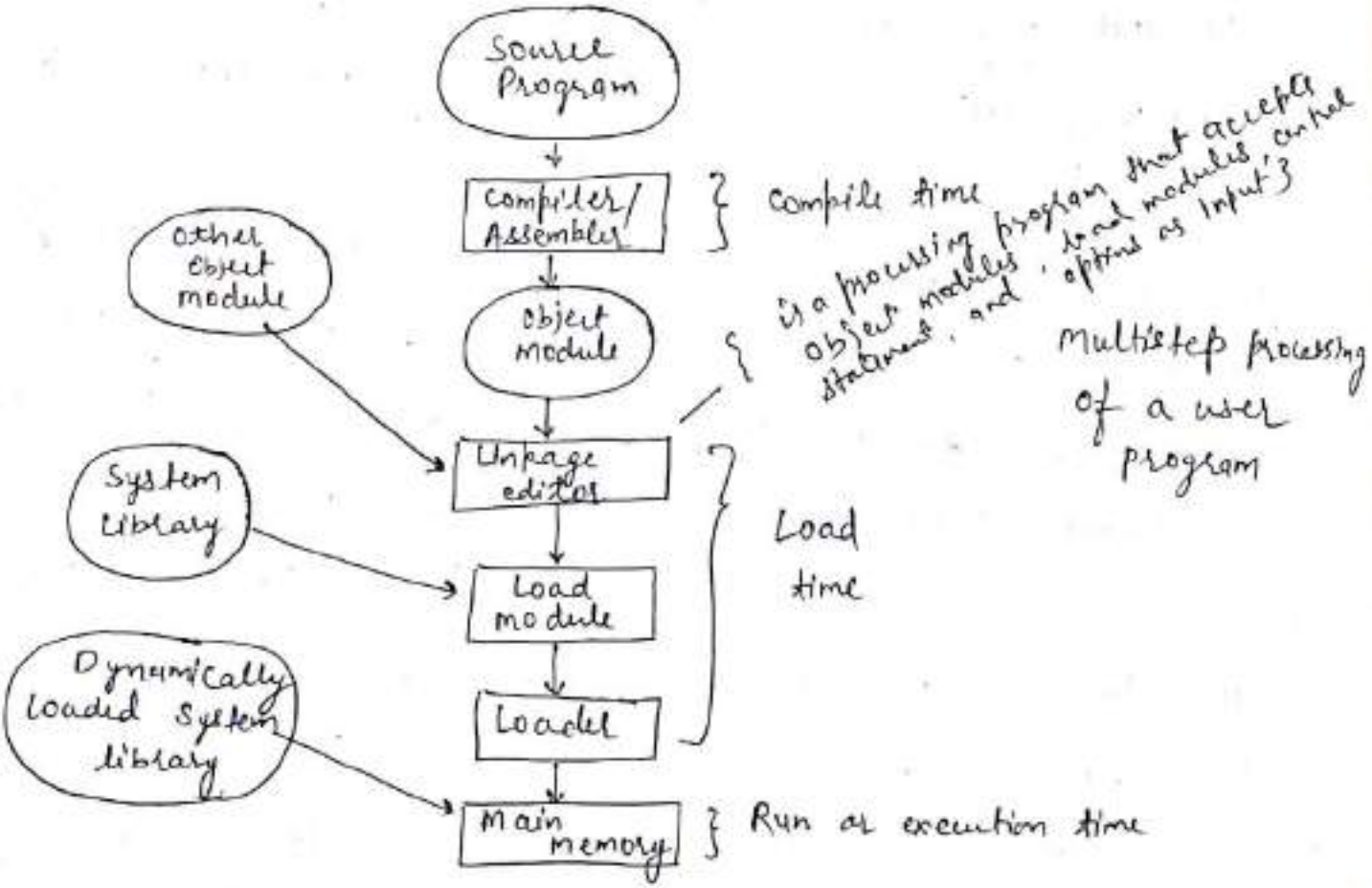
- The difference between these two numbers is internal fragmentation - memory that is internal to a partition but is not being used.

Solution to external fragmentation is compaction.
(Relocation is the process of assigning load addresses for position dependent (code and data))

- The goal is to shuffle the memory contents so as to place all free memory together in one large block.
- Compaction is not always possible, however.
- If relocation is static and is done at assembly or load time, compaction cannot be done.
- Compaction is possible only if relocation is dynamic and is done at execution time.
- If addresses are relocated dynamically, relocation requires only moving the program and data and then changing the base register to reflect the new base address.
- simplest compaction algorithm is to move all processes to one end of memory, all holes move in the other direction, producing one large hole of available memory.
- Another solution to external fragmentation is to permit the logical address space of the processes to be non contiguous.
- Techniques for this are - Paging & Segmentation

Address Binding Mechanism

- The program must be brought in main memory and placed within a process for its execution.
- The collection of process as on the disk waiting to be brought into memory forms the input queue.
- The procedure is to select one of the process from this queue and load it into memory.
- As the process is executed, it access data and program from the memory.
- When process completes its execution it sets free the memory space for the availability of next process for execution.



(ASU)

- Addresses in the source program are generally symbolic.
- A compiler will bind these address to relocatable addresses.
- The linkage editor or loader will bind these addresses to absolute addresses.
- Each binding is a mapping from one address space to another.

The binding of data and program to memory addresses can be done at any of the following

(i) Compile Time :→ • Binding at compile time generates absolute addresses, If you know at compile time where the process will reside in memory.

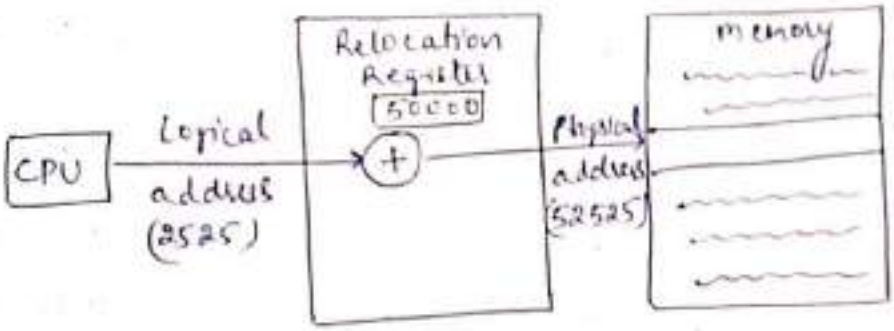
- After some time if the starting address of a process is changed, then the entire process must be recompiled to generate the absolute address again.

(ii) Load Time :→ • At compile time if it is not known that where a process will reside in memory then the compiler must generate relocatable address.

- If the starting address is changed then we need only to reload the user code to incorporate this changed value.

(iii) Execution time :→ . This method permits moving a process from one memory segment to another during run time.

- In this case final binding is delayed until run time.
- Special hardware is available for this scheme to work.
- most general-purpose operating system use this method.



Dynamic Relocation using relocation register.

Relocation is a method of shifting a user program from one memory location to another.

- There are two types of relocation: Static & Dynamic
- Static relocation takes place at load time and remains fixed once the program is relocated while the dynamic relocation takes place at run time.

Logical Versus Physical Address space

- An address generated by the CPU is commonly referred to as a logical address.
- An address seen by the memory unit - that is the one loaded into the memory address register of the memory is referred to as a physical address.
- The compile time and load time address binding methods generate identical logical and physical addresses.
- The execution time address binding scheme results in differing logical and physical addresses.
- In this we can refer logical address as a virtual addr
- The run time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (MMU).
- In simple mmu scheme, the base register is now called a relocation register.
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

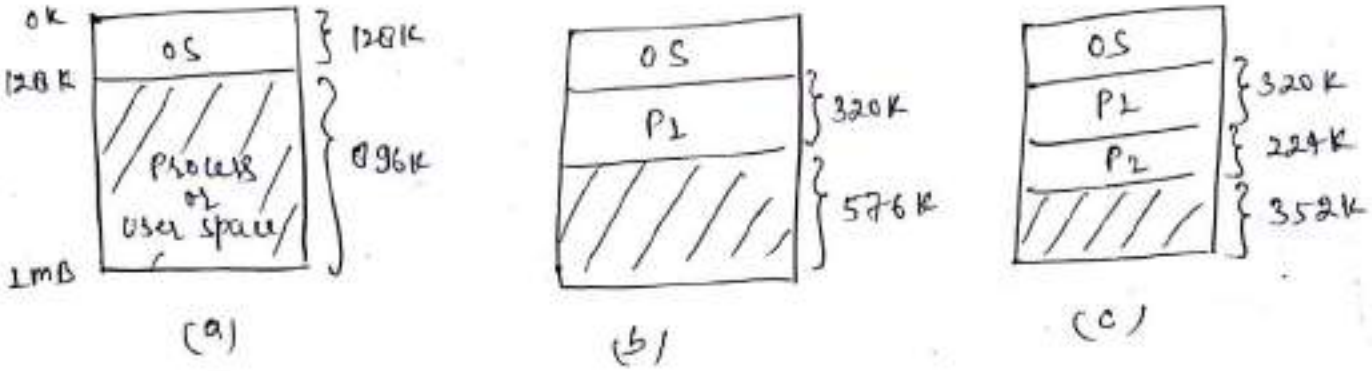
- For ex. if the base is at 50000, then an attempt by the user to address location 0 is dynamically relocated to location 50000.
- An access to location 2525 is mapped to location 52525
- The user program never sees the real physical addresses.
- The program can create a pointer to location 2525, store it in memory, manipulate it, and compare it with other addresses – all as the number 2525.
- Only when it is used as a memory address is it relocated relative to the base register.
- 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 supplies logical addresses, these addresses must be mapped to physical addresses before they are used.

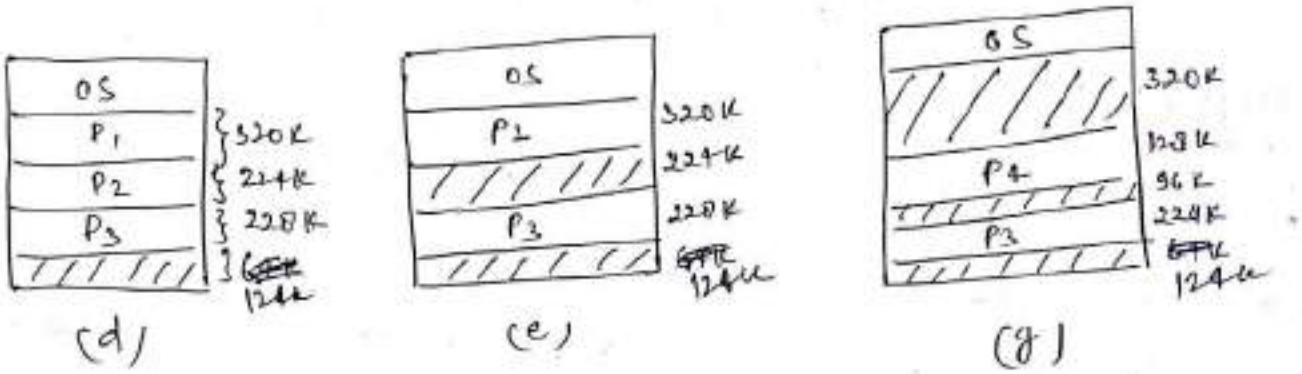
Multiprogramming With Variable Partition

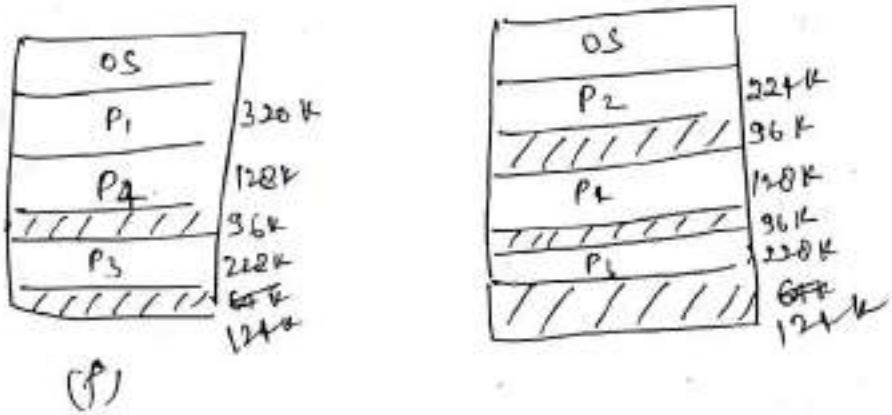
- Static partitioning of memory is generally suitable for static environment where, the workload is predictable
- For ex. stable production environment such as process control.
- Main disadvantage is its inflexibility and inability to adapt to changing system needs. One of its primary problem is internal fragmentation of memory.
- In dynamic partitioning the partitions used are of variable length and number.
- When a process is brought into memory, it is allocated exactly as that amount of memory that it needs not more than that it requires.
- In figure shown uses 1 MB memory for dynamic partition.
- The first three processes P_1, P_2, P_3 are loaded starting where OS ends and occupy just enough space for each process (figure b, c, d).
- This leaves a small 'hole' at the end of memory that is very small for the fourth process P_4.
- Then the operating system must swaps out any of the three process whose execution is computed or

the process which is not required presently.

- Say process P_2 is swapped out which leaves sufficient space to load the process P_4 (fig f)
- Since P_4 is smaller than P_2 so another hole is created





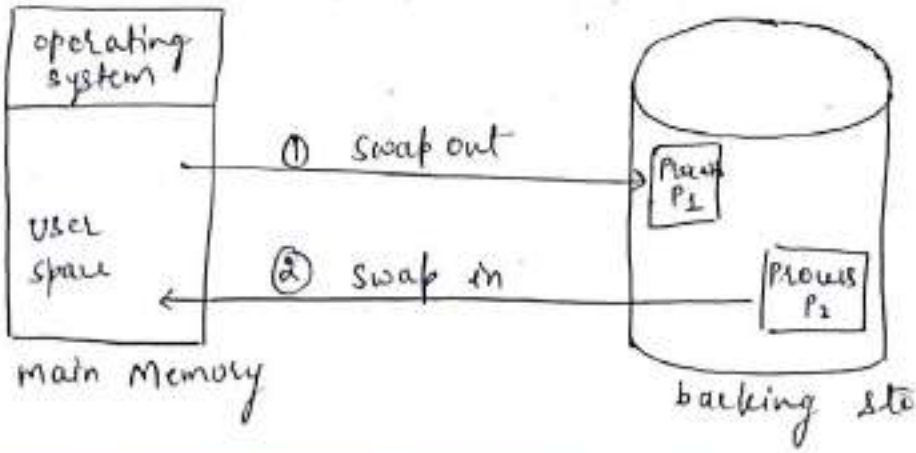


- After some time there may be a situation when no process is present in ready queue and P_2 wants to be executed from waiting queue.

- Then the OS swaps process P_1 out and process P_2 back (fig. h).
- As shown in this example this method is started well but after some time it result a lot of small holes in memory.
- And memory becomes more and more fragmented as time goes on. This is the problem with this method and called as external fragmentation.

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.
- For ex. in round robin scheduling algorithm when a time quantum expires, the memory manager will start to swap out the process and to swap another process into the memory space.



swapping of two processes using a disk as a backing store

- A variant of swapping policy is used for priority based scheduling algorithms.
- If a higher ~~two~~ priority process arrives then memory manager can swap out the lower priority process. and then load higher priority process.
- When higher priority process finishes, the lower priority process can be swapped back in and continued.
- This variant of swapping is sometime called roll out, roll in.

- If binding is done at assembly or load time, then the process cannot be easily moved to a different location.
- If execution time binding is being used, then a process can be swapped into a different memory space.
- Normally a process swapped back into the same memory space it occupied previously.
- Swapping requires a backing store which is fast disk. and should be large enough to accommodate copies of all memory images for all users.
- The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.

- The context-switch time in swapping is fairly high.
- For ex. assume that the user process is 10 MB in size and the backing store is a hard disk with a transfer rate of 40 MB per second.

The actual transfer from or to main memory

10 MB / 40 MB per second = 1/4 second
= 250 milsec.

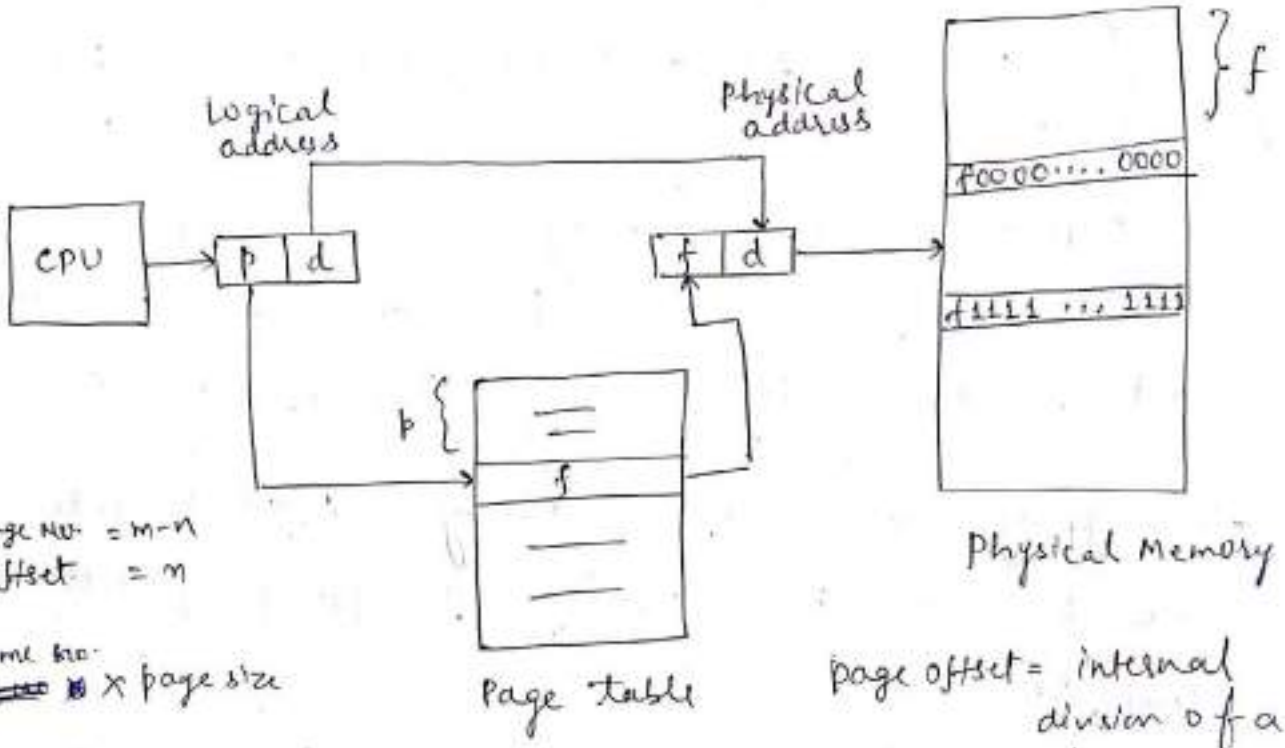
Suppose average latency is 8 milsec then time will be 258 MS for each swap in and swap out, total time will be 516 milisec.

- Swapping is constrained by transfer rate and we have to sure that swapping process must be completely idle.
- Never swap a process with pending I/O.

Paging

- Paging is a memory management scheme that permits the physical address space of a process to be noncontiguous.
- Paging avoids the considerable problem of fitting memory chunks of varying sizes onto the backing store.
- The problem arises because when some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store.

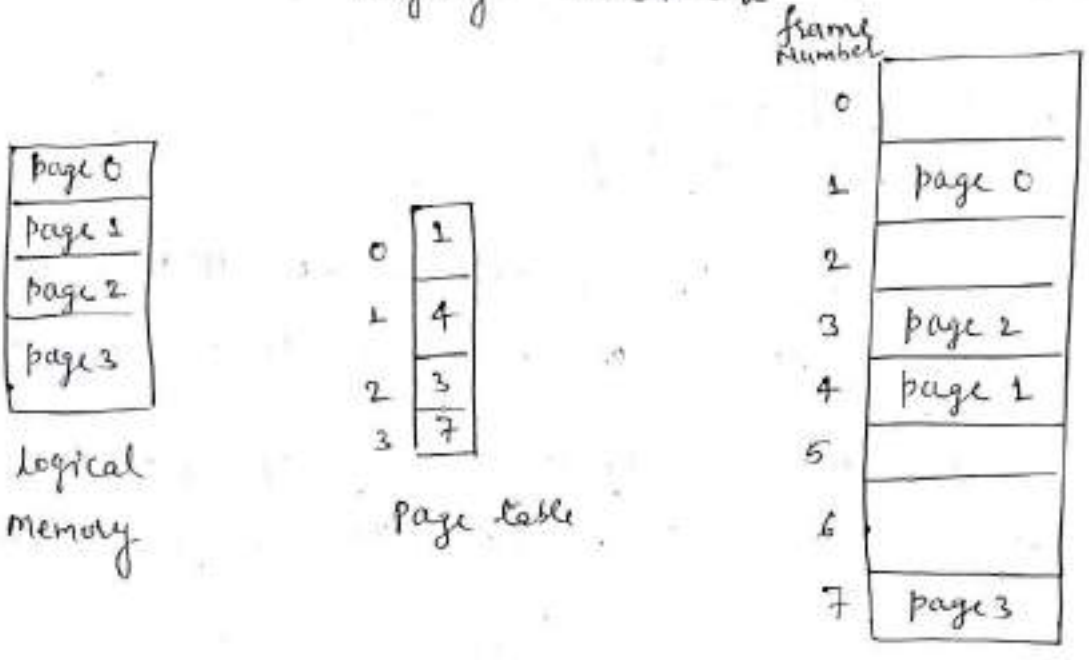
- The backing store also has the fragmentation problem.
- Traditionally support for paging has been handled by hardware.
- Recent designs have implemented paging by closely integrating the hardware and operating system.



f = frame no. × page size

page offset = internal division of a page

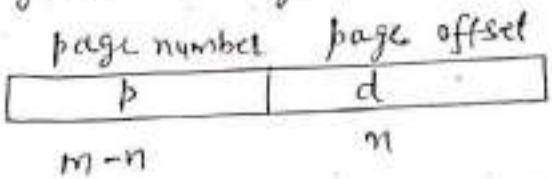
Paging hardware



Paging model of logical and physical memory

Basic Method :→

- 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 the backing store.
- The backing store is divided into fixed sized blocks that are of the same size as the memory frames.
- Every address generated by the CPU is divided into two parts : 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 base address is combined with the page offset to define the physical memory address that is sent to the memory unit.
- The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.

- The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.
- If the size of logical address space is 2^m and a page size is 2^n addressing unit (bytes or words) then the high order m-n bits of a logical address designate the page number and the n low order bits designate the page offset.
- where p is an index into the page table and d is the displacement within the page.
- for ex: Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)
- 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 the 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.

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

Logical memory

0	5
1	6
2	1
3	2

Page Table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

Physical memory

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

- When we use a paging scheme, we have no external fragmentation: Any free frame can be allocated to a process that needs it.
- However, we may have some internal fragmentation.
- For ex: If page size is 2048 bytes, a process of 72,766 bytes would need 35 pages plus 1,086 bytes.
- Resulting in an internal fragmentation of 2,048 - 1086 = 962 bytes.

- If process size is independent of page size, we expect internal fragmentation to average one-half page per process.
- Hence small page sizes are desirable but overhead is involved in each page table entry.
- An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory.
- The user program views memory as one single space, containing only this one program but it is scattered throughout physical memory which also holds other programs.
- The difference between the user's view of memory and the actual physical memory is reconciled by the address translator hardware.
- The logical addresses are translated into physical addresses and this mapping is hidden from the user and is controlled by the operating system.

~~Segmentation~~

Hardware support for paging :→

- The hardware implementation of page table can be done in many ways.
- One approach is ~~through~~ through by using dedicated registers. But the usage of register for the page table is satisfactory only if the page table is small.

- most of the computers requires more entries in their page table. then this approach is not feasible.
- solution to this problem is to use a special, small fast lookup hardware cache called translation look aside buffer (TLB). and is high speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used then an item is compared with all tags simultaneously.
- If the item is found then corresponding value is returned.
- When a logical address is generated by CPU then its page number is presented in TLB.
- If page number is found in TLB then it is called TLB hit.
- If the page number is not in TLB then it is called TLB miss.
- The percentage of times that a particular page number is found in TLB is called the hit ratio.
- If a computer system takes t_1 time to access TLB and t_2 time to access memory then mapped memory access will take $t_1 + t_2$ time when page number is present in TLB.

- When page number is not present in TLB then we require t_1 time to access TLB in which page number is not present and t_1' time to access the page number in page table and then t_2 time to access memory.
- So we require total $t_1 + t_1' + t_2$ time when page no is not present in TLB.

So effective access time will be

$$T_{eff} = \text{hit ratio} * (t_1 + t_2) + \text{miss ratio} * (t_1 + t_1' + t_2)$$

$t_1' = t_2$ for a particular computer hardware

Segmentation

- Segmentation is another technique for the noncontiguous storage allocation.
- It is different from paging as pages are physical in nature and hence are of fixed size, whereas segments are logical divisions of a program and hence are of variable size.
- In segmentation, we divide the logical address space into different segments.
- For simplicity, the segments are referred by a segment number, rather than segment name.

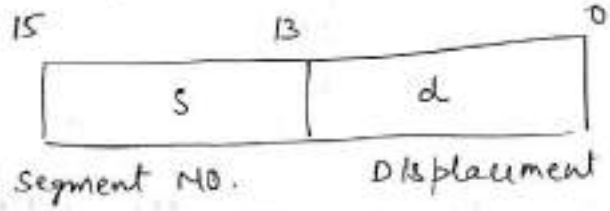
< segment_number , offset >

- The size of a segment varies according to the data stored in it or the nature of operation performed on that segment.
- Each segment consists of linear sequence (0, 1, 2, 3, ...) of address starting from 0 to maximum value depending upon the size of segment.

Segment NO	0	1	2	3
Limit Add.	0-999	1000-1699	1700-2139	2500-3399

Virtual address segmentation

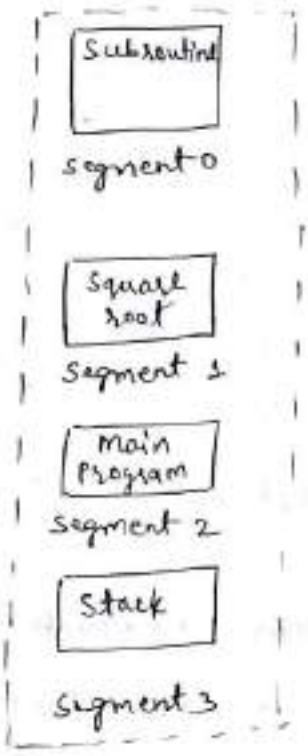
- This is different from the paging system in which a single dimensional virtual address and a two dimensional address would be exactly same in the binary form as the page size is an integral power of 2.
- In segmentation it is not possible as the segment size is unpredictable.



Logical address space (16 bit).

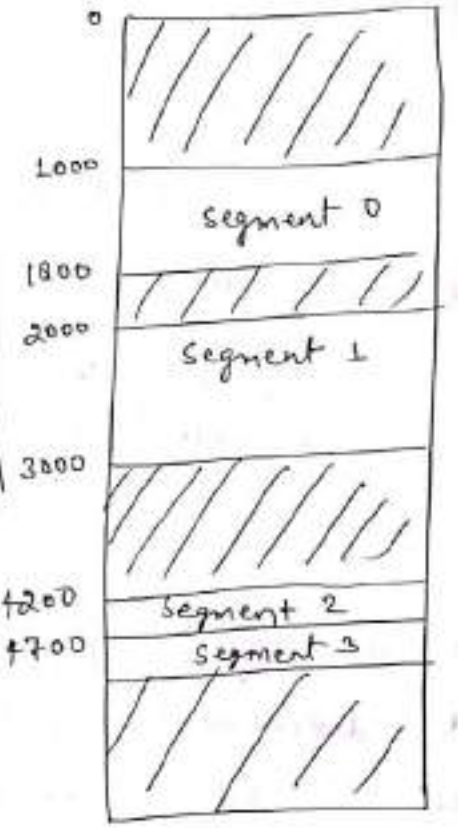
- Virtual address space is divided into two parts in which high order units refers to segment number and lower order units refer to displacement (limit value

Ex - consider the example in which logical address space is divided into four segments numbered from 0 through 3.



Logical address space

	Base	Limit
0	1000	800
1	2000	1000
2	4200	500
3	4700	600



physical memory

Ex -

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses - (a) 0430 (b) 110 (c) 2500 (d) 3400 (e) 4112

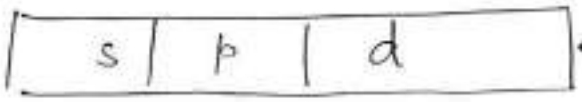
⇒ For the given logical addresses the first digit refers to the segment number (s) while remaining digits refers to the offset value (d).

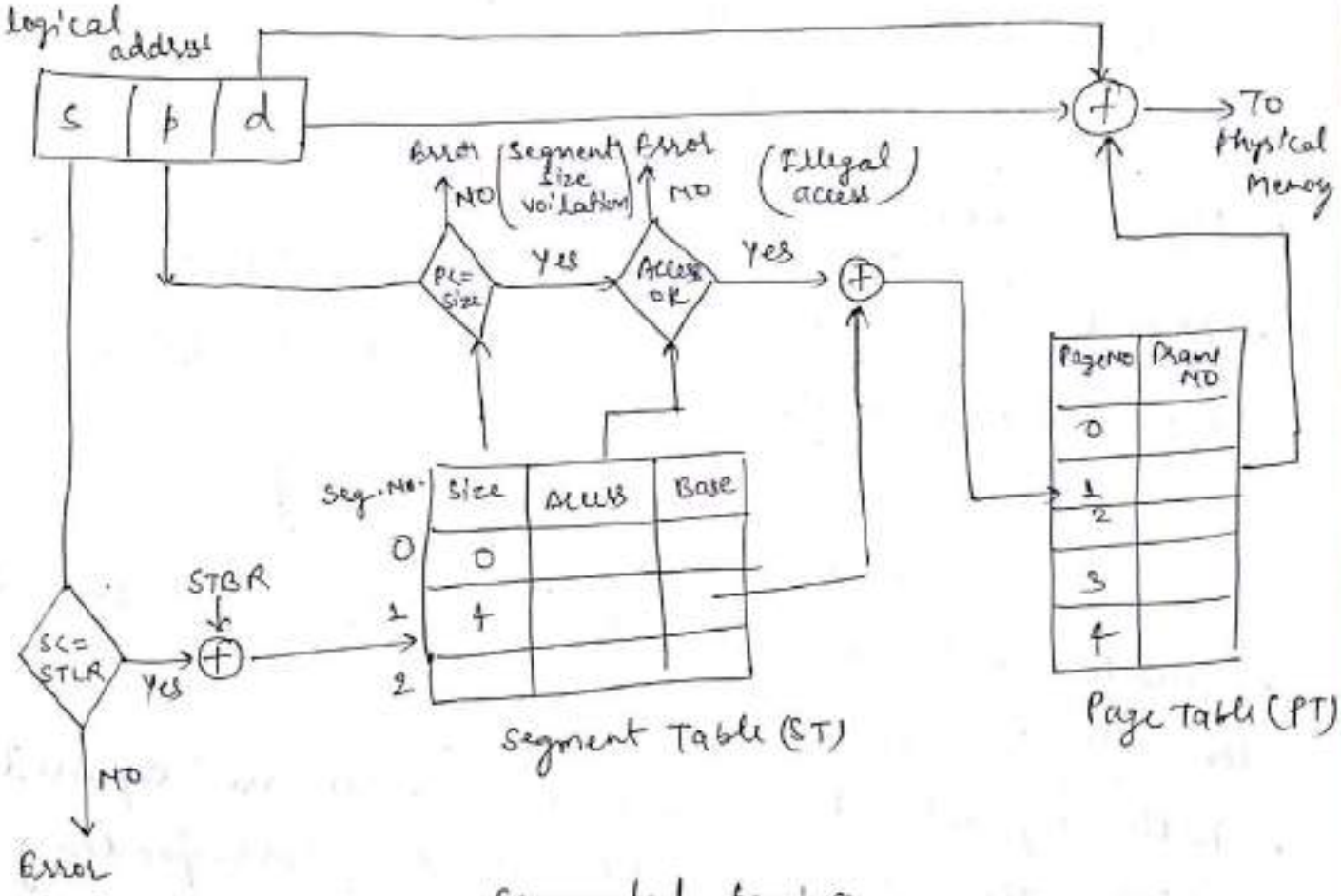
- Physical address for logical address 0430 will be
  = base + offset (430) = 219 + 430 = 649
- Physical address for 110 = 2300 + 10 = 2310
- Physical address for 2500 = 90 + 500 = 590
  but it is impossible since the segment size is 100 so it is illegal address.
- Physical address for 3400 = 1327 + 400 = 1727
- Physical address for 4112 = illegal address as the size of segment four (96) < offset value (112).

Segmentation With Paging

- Divide each segment into pages of fixed size for the allocation purposes.
- Both segment table and page table are required for mapping the logical addresses to corresponding physical addresses.
- Instead of containing the base and limit (size) of a particular segment, each entry of segment table contains the base address and size of the page table to be used for mapping of the related pages of segments.

- Each logical address in a combined system consists of three fields: the segment number (s), page no (p) and the offset (d) within the page as:





Segmented paging

- When a virtual address is presented to the mapping hardware, the segment number is used to locate the corresponding page table.
- Provided that the issuing process is authorized to make the result to the target segment, the page no is used to index the page table.
- If target page is absent from the memory, the mapping

hardware generate an error.

- The program consists of various segments which is given by a segment table (ST) and each segment is divided into a no of fixed sized (equal size) pages whose information is maintained into a page table (PT).
- So there are as many page tables as there are enters in segment table
- If a process has 3 segment then there will be 3 page tables for that process one for each segment.

This mapping scheme works as follows:

(1) The system has a pair of registers, STLR (Limit Register) and STBR (base register) whose value will be different for each process.

(2) The value of segment number (s) is checked with the value of limit register and then added with the value of base register to form the index of the segment table.

(3) An entry of segment table is chosen and its size is compared with the value of page no (p) which indirectly gives, the maximum no of pages in that segment

(4) Now access rights are validated against the chosen entry of segment for instance. If the current instruction is for writing into that segment and the access rights for that user process is mentioned as 'Read only' in ST then an error will be generated.

(5) If everything is OK, then the base (b) of the chosen entry is added with the page no. (p) to result the index of the page table (PT).

(6) Then the frame no. (f) is extracted from the selected PT entry.

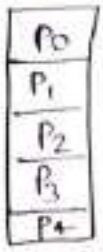
(7) The offset value (d) is concatenated to this from (f) to get the actual physical address.

- In segment table the field size gives the total no. of pages and therefore ~~maximum~~ page no in that segment starting from 0.
- For ex. In figure PT ~~entire~~ entry are from 0 to 4 so the size field corresponding to this segment is 4.

Protection in Paging

- Protection is produced by means of protection bit, which is associated with each page.
- This bit is kept in page table as every reference to the memory, goes through paage table.
- This bit is called valid (v) or invalid (i) bit.
- If the access to the page present in the memory is valid, then the particular bit of the frame is set to 'v' otherwise it is set to 'i'.

- For eg: in which there are two users A and B.
- Suppose the total memory space for both processes is of 8 frames.
- Out of which 6 frames are for user A and 2 frames are for user B.
- Suppose user B uses pages 5, 6 and 7 and frames 0, 3 and 7 respectively.

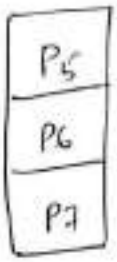


User A

page no | frame no

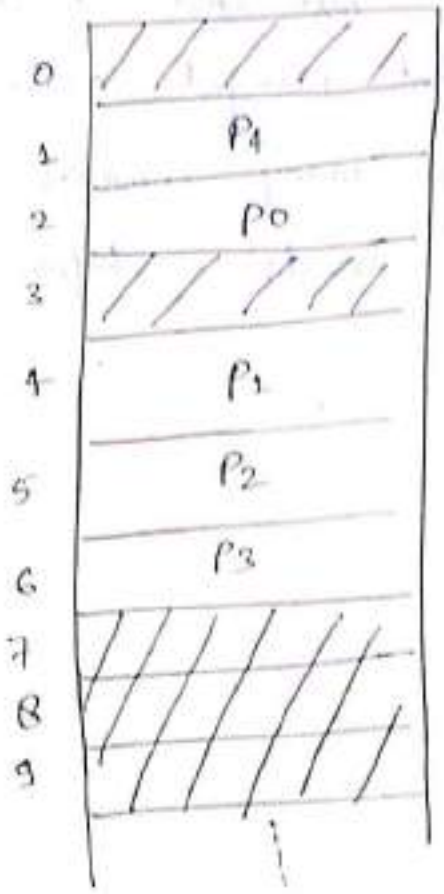
0	2	V
1	4	V
2	5	V
3	6	V
4	1	V
5	0	i
6	3	i
7	7	i

Page table for user A



User B

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



Physical memory for user A

Protection in Paging System

Protection in Segmentation

- An advantage of segmentation is the association of protection with the segments.
- Some segments are instructions while other are data.
- Since instructions are non self modifying so they are defined as read only or execute only.
- Memory mapping hardware will check the protection bits associated with each segment table entry to prevent illegal acces to memory.
- Thus many common program errors can be detected by the hardware before they cause any damage.

Virtual Memory

- virtual memory is a technique that allows the execution of processes that are not completely in memory.
- One major advantage of this scheme is that program can be larger than physical memory.
- Virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concern of memory storage limitations.
- Also allows processes to share files easily and to implement shared memory.
- Virtual memory is not easy to implement and may substantially decrease performance if it is used carelessly

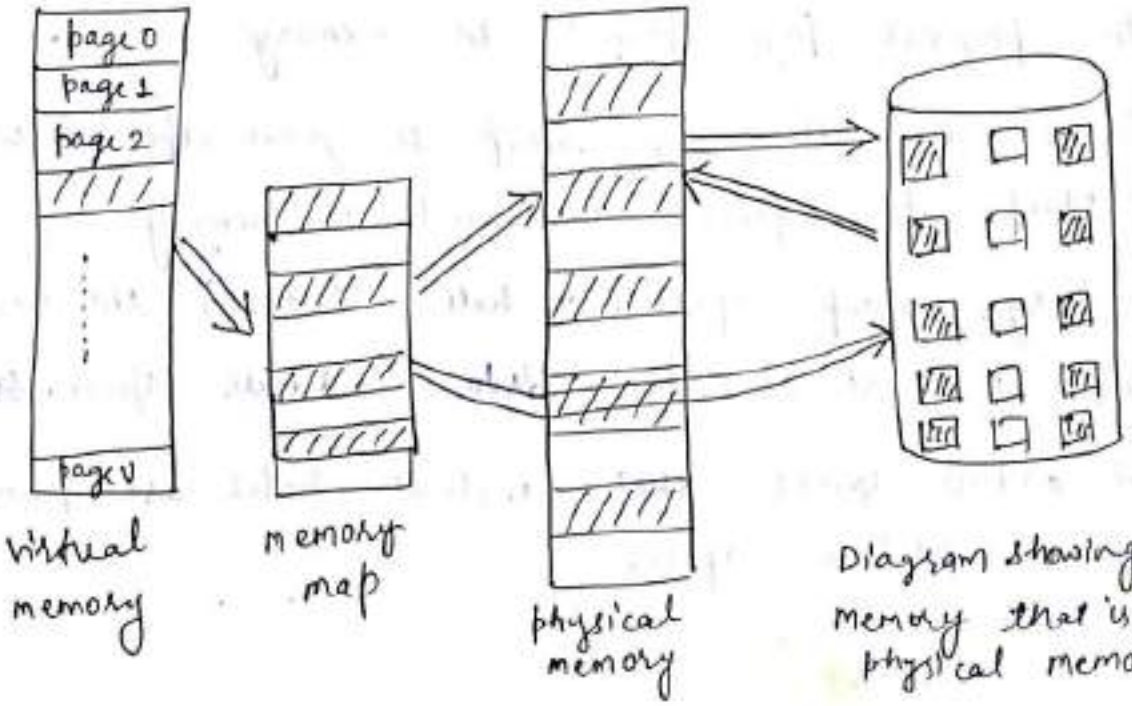
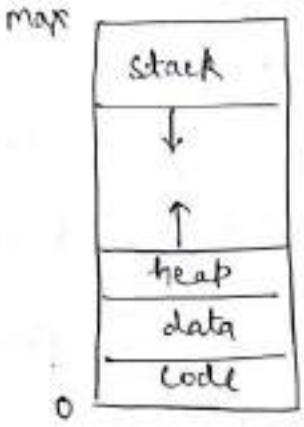


Diagram showing virtual memory that is longer tha physical memory

- The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.
- This view is that a process begins at a certain logical address -- say address 0 and exists in contiguous memory as shown in figure.



virtual address space

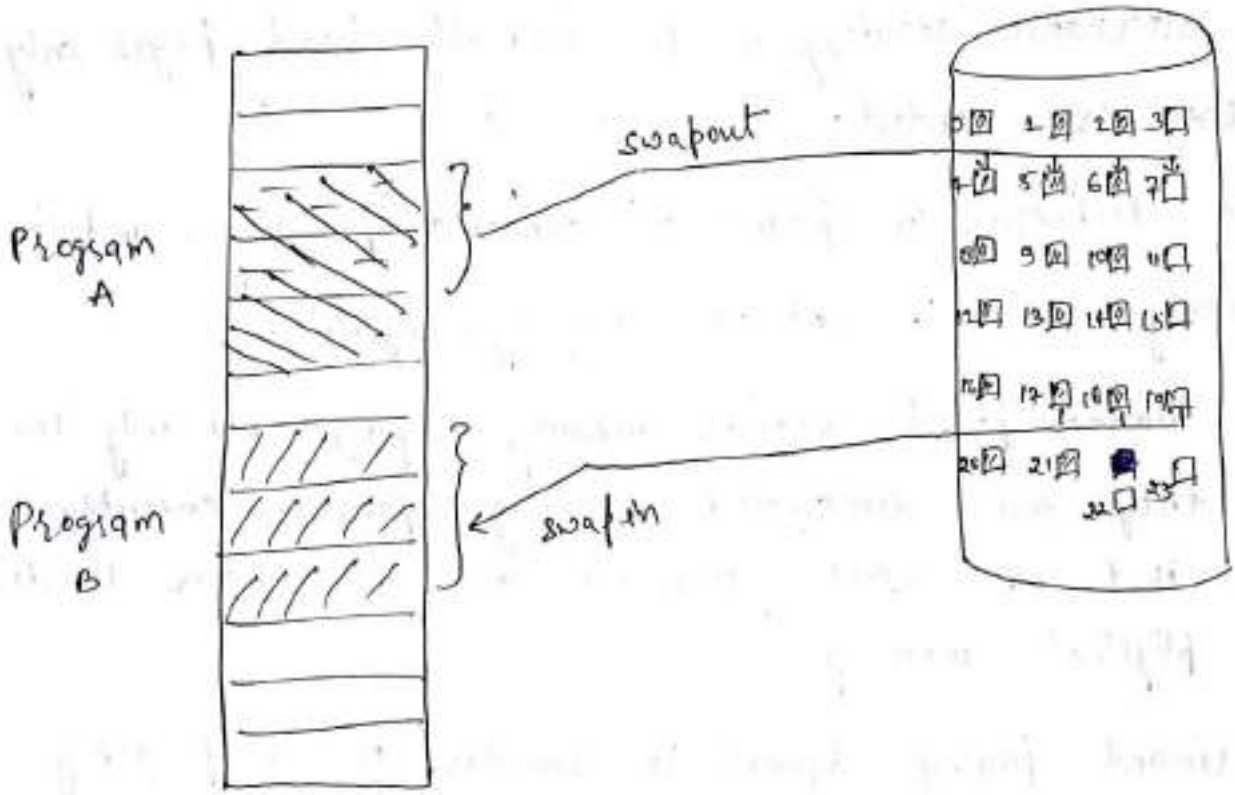
- In fact physical memory may be organized in page frames and that the physical page frames assigned to a process may not be contiguous.
- It is upto memory management Unit (MMU) to map logical pages to physical page frames in memory.
- In figure we allow for heap to grow upward in memory and stack to grow downward in memory.
- The large blank space (or hole) between the heap and the stack is part of the virtual address space.
- Virtual address spaces that include holes are known as <u>Sparse address</u> spaces.

- Using a sparse address space is beneficial because the holes can be filled as the stack or heap segment grows.

Demand Paging

- Suppose a program that starts with a list of available option from which the user is to select.
- Loading the entire program into memory results in loading the executable code for all options, regardless of whether an option is ultimately selected by the user or not.
- An alternative strategy is to initially load pages only as they are needed.
- This technique is known as demand paging and is commonly used in virtual memory system.
- 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 (next figure) 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 we use a lazy swapper which never swaps a page into memory unless that page will be needed.
- Since we are now viewing a process as a sequence of pages, rather than as one large contiguous address space, use of the term swapper is technically incorrect.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process.
- We thus use pager, rather than swapper, in connection with demand paging.



Transfer of a paged memory to contiguous disk space

Advantages :-

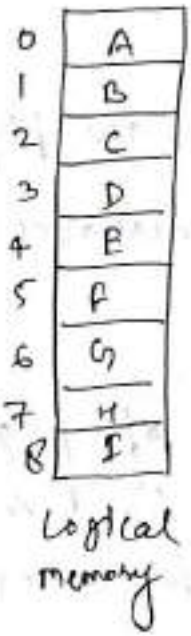
(i) Reduced memory requirement

(ii) Swap time is also reduced

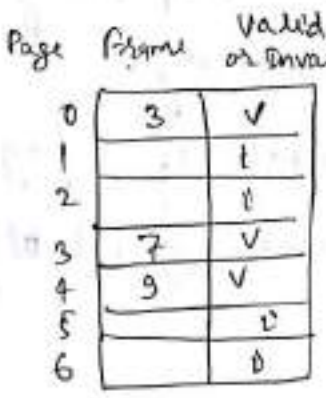
(iii) Increase degree of multiprogramming

Disadvantages :-

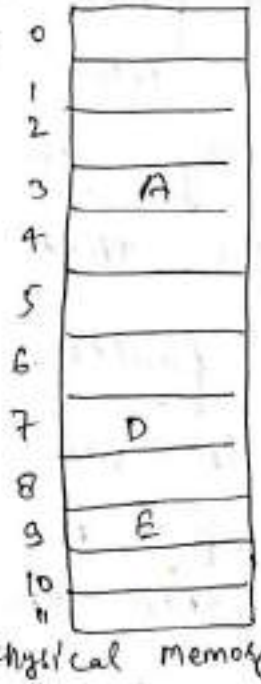
- Using this method swapping is done using hit and trial method.
- For eg. If a program is having 5 pages: 0, 1, 2, 3, 4 and out of these only pages 0, 3, 4 are loaded into memory at the time of execution start.
- If the program tries to access the address of page number 2, then there will be an error.
- This error is called as page fault error.
- This error is noticed by paging hardware while translating the address using the page table.



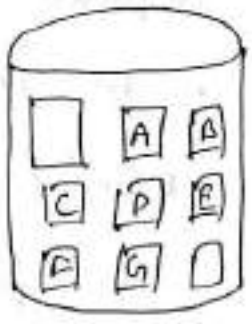
Logical memory



Page table



Physical memory



Back storage (Disk)

Pure Demand Paging :→

- We can start executing a process with no pages in memory
- When the operating system sets the instruction pointer to the first instruction of the process, which is on a non memory resident page, the process immediately faults for the page.
- After this page is brought into memory, the process continues to execute.
- Faulting as necessary until every page that it needs is in memory.
- At that point, it can execute with no more faults.
- This scheme is pure demand paging: Never bring a page into memory until it is required.

Performance of demand Paging

- When page fault occurs, operating system must bring required page into the memory.
- In this way the number of pages in the physical memory for that process gradually increases with the page faults.
- After while when a sufficient number of pages are build up, the pages are normally found in the memory and then the frequency of page fault reduces.

- The performance of demand paging is measured by computing the effective access time for demand paging system.
- The general memory access time 'ma' is within the range of 500 nanoseconds to 2 microseconds.
- When there is no page fault i.e all the pages required for execution of a program is present in memory then effective access time is similar to memory access time.
- If the page fault occurs then it is necessary to bring the required page into memory from secondary storage.
- This will increase the effective access time.
- Hence the effective access time depends upon the probability [redacted] of page fault 'p' and hence it is computed as:

$$Ea = [(p \times \text{'page fault time'}) + ((1-p) \times ma)]$$

where page fault time is the time required to service the page which consists of:

(i) service the page fault interrupt
(ii) Swap the page
(iii) Restart the process

Page Replacement

Page replacement take the following approach.

- If no frame is free, we find one that is not currently being used and free it.
- We can free a frame by writing its contents to swap space and changing the page table to indicate that the page is no longer in memory.
- We can now use the freed frame to hold the page for which the process faulted.

Procedure includes -

(I) Find the location of the desired page on the disk.

(II) Find a free frame:

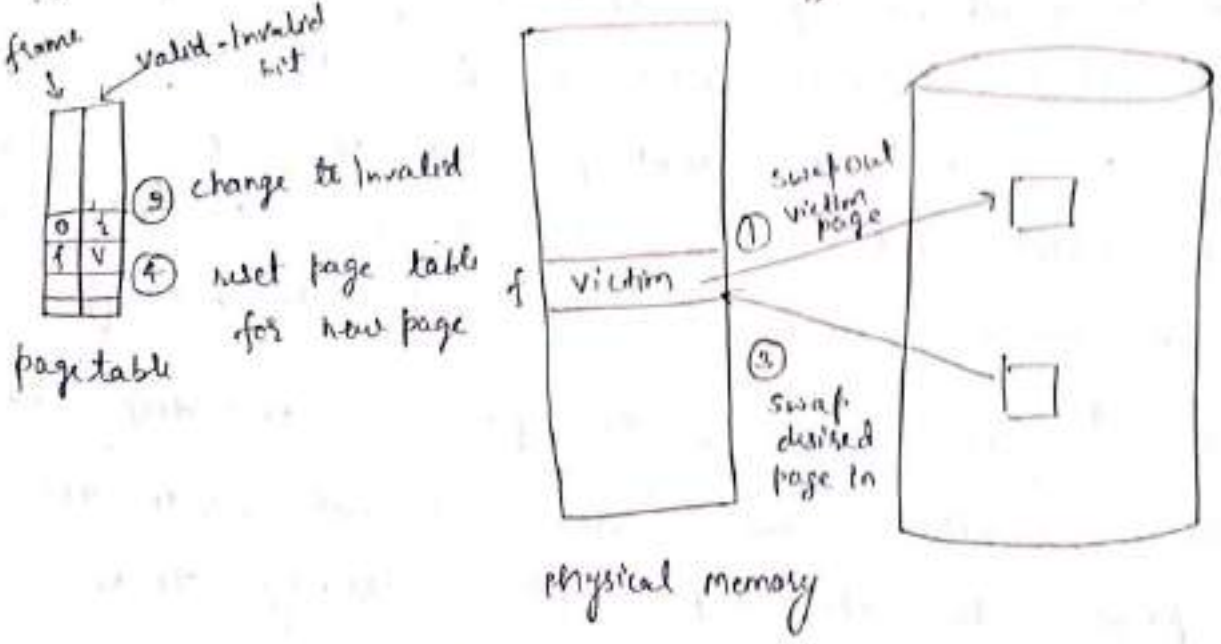
(a) If there is a free frame, use it

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

(C) Write victim frame to the disk, change the page and frame tables accordingly.

(III) Read the desired page into the newly freed frame; change the page and frame tables.

(IV) Restart the user process.



Page Replacement

- If no frames are free, two page transfers (one out and one in) are required.
- This effectively doubles the page fault service time and increases the effective access time accordingly.
- This overhead can be reduced by using a modify bit (or dirty bit).
- In this scheme each page or frame has a modify bit associated with it in 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 a page for replacement, we examine its modify bit.

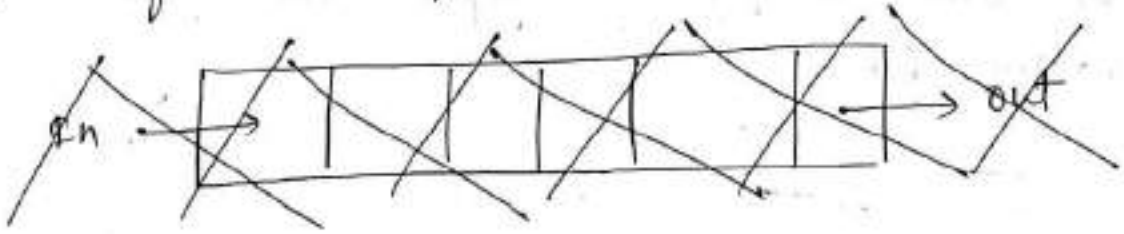
- If the bit is set, we know that the page has been modified since it was read in from the disk.
- Therefore we must write that page to the disk.
- If the modify bit is not set, however the page has not been modified since it was read into memory.
- Therefore if the copy of the page on the disk has not been overwritten then we need not write the memory page to the disk: It is already there.

Page Replacement Algorithms

(i) FIFO Page Replacement :→

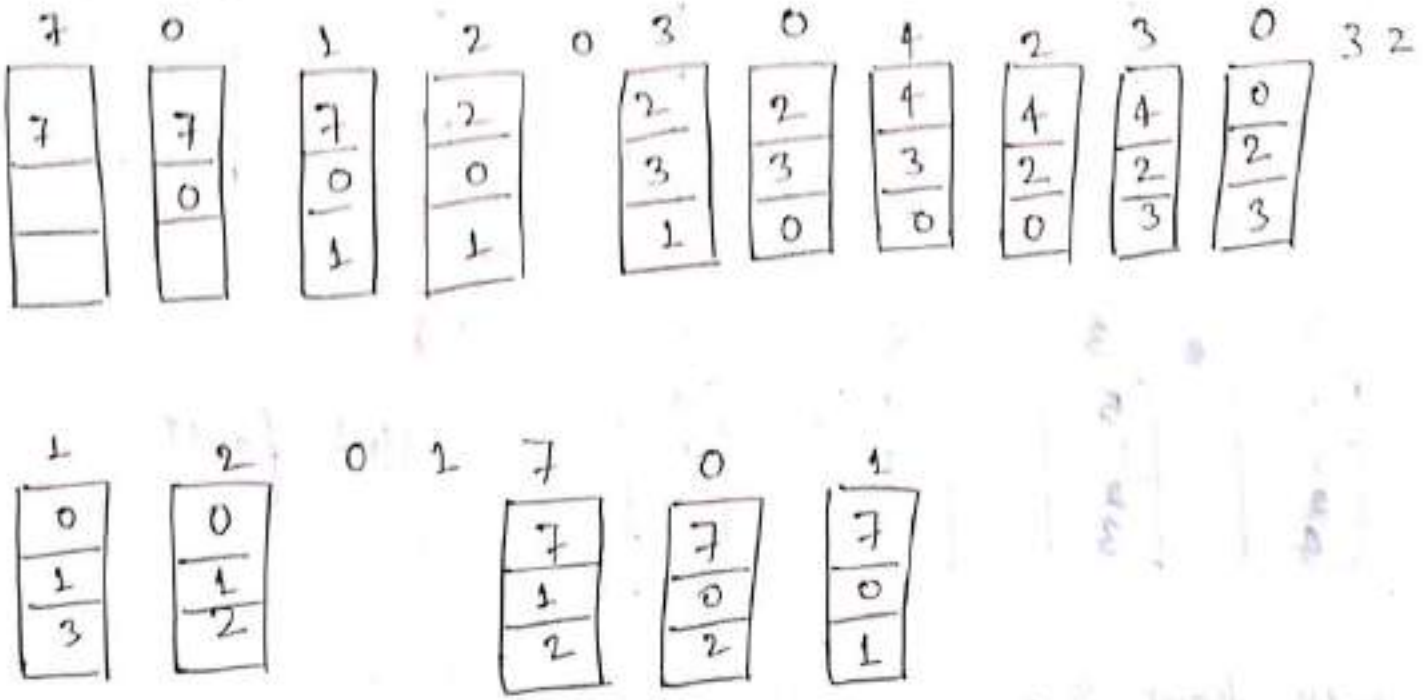
The string of mememory references is called as reference string.

- In this method the oldest page present in the memory is selected for replacement.
- For this the page at the head of the queue is replaced. ~~And when the page is brought into the main memory then it is placed at the end of the queue.~~



consider following reference string —

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



- some time page fault for more frames is greater than the number of faults for less no. of frames. This unexpected result is known as Belady's anomaly.
- means for some page replacement algorithms the page fault rate may increase as the number of allocated frames increases.

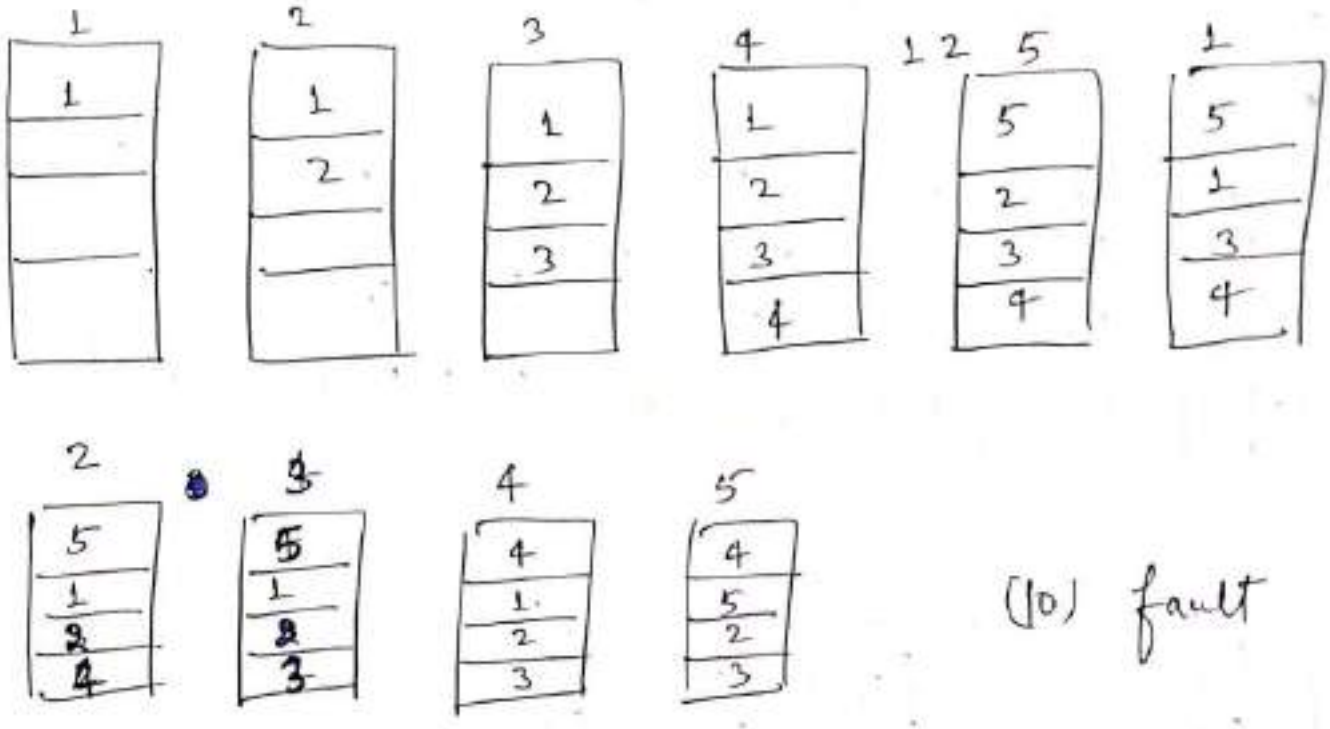
for ex. Consider reference string

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

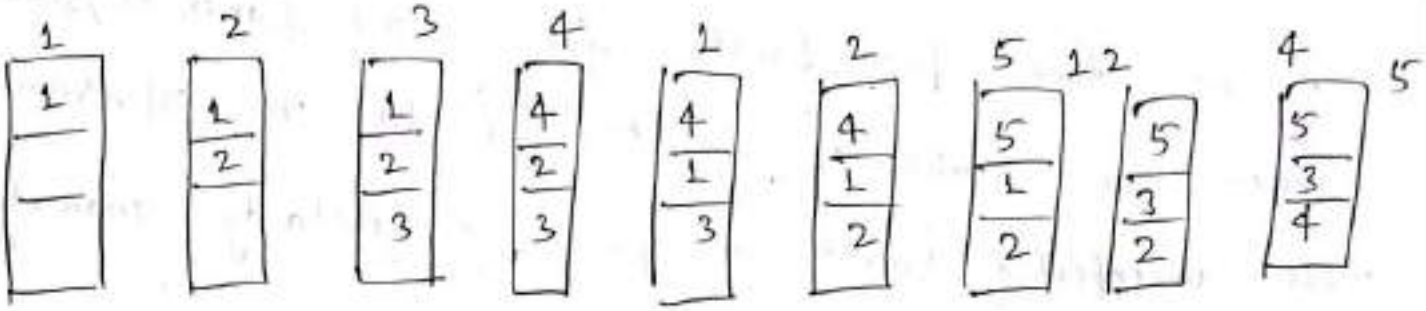
with frame 3 fault will be 9 and with 4 frame

fault will be 10.

with frame - 4



with frame 3 -



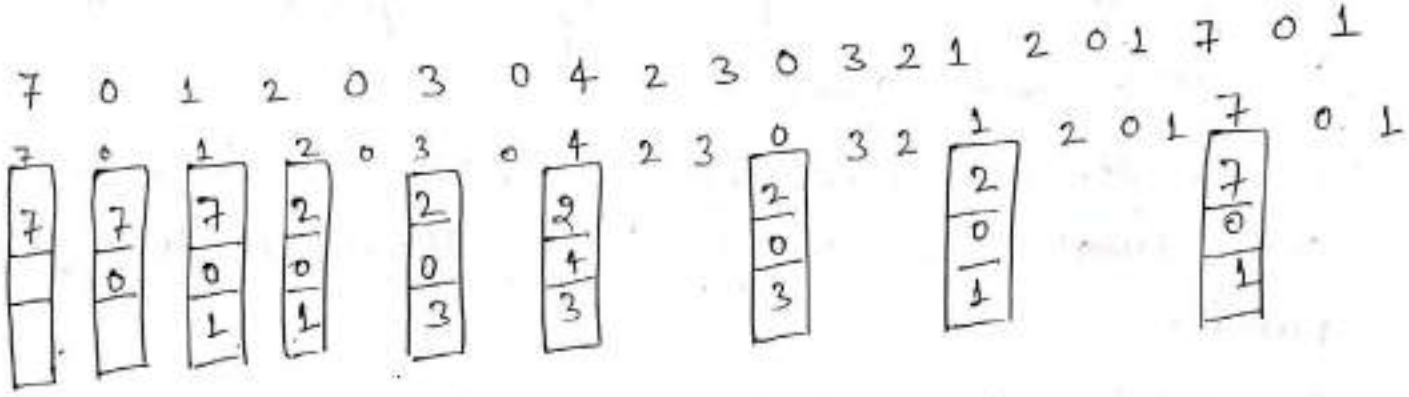
(ii) Optimal Page Replacement :→

. one result of the discovery of Belady's anomaly was the search for an optimal page replacement algorithm.

∴ An optimal page replacement algorithm has the lowest page fault rate of all algorithms and will never suffer from Belady's anomaly.

Procedure is as follows -

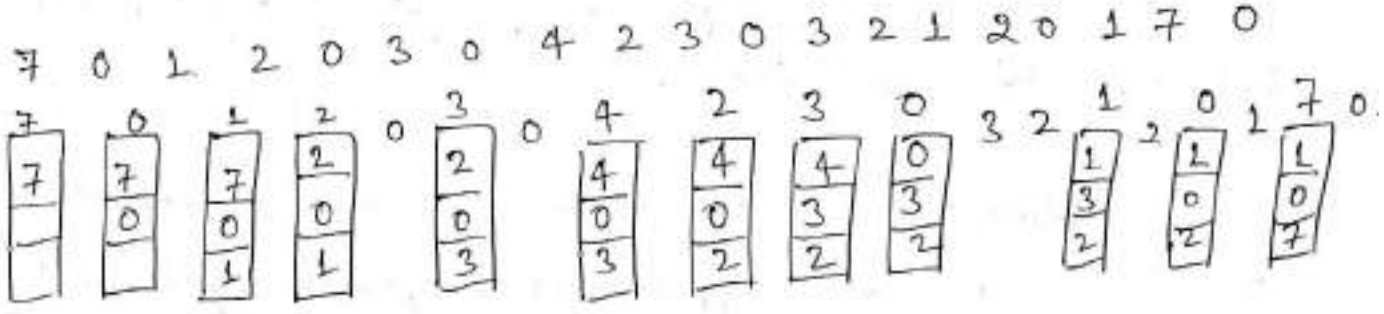
- Replace the page that will not be used for the longest period of time.



optimal page replacement.

(iii) LRV (Least Recently Used) Algorithm :→

- In this algorithm we can replace the page that has not been used for the longest period of time.



LRV Page Replacement algo.

(iv) Counting Based Page Replacement :→

(a) The least frequently Used (LFU) :→

- Requires that the page with the smallest count be replaced.
- The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, when a page is used heavily during the initial phase of a process but then is never used again.
- Since it was used heavily it has a large count and remains in memory even though it is no longer needed.
- One solution is to shift the counts right by 1 bit at regular intervals forming an exponentially decaying average usage count.

(b) The most frequently used (MFU) :→

- Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

→ The implementation of these algorithms is expensive and they do not approximate optimal replacement well.

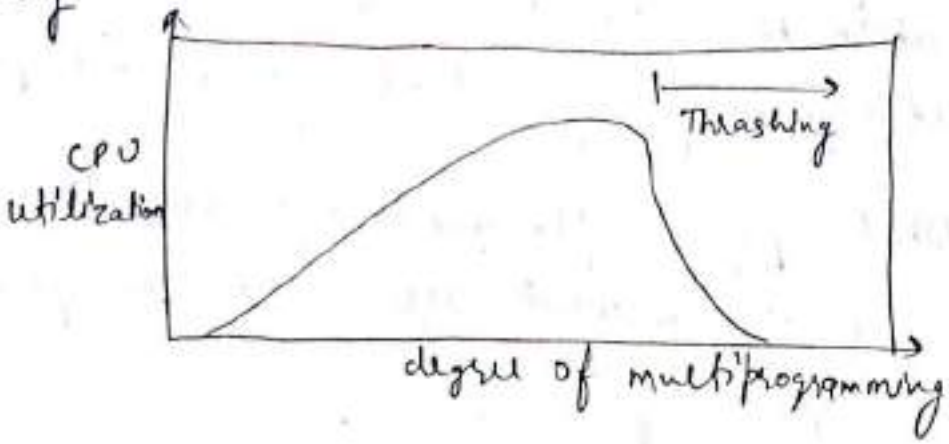
Thrashing

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

Cause of Thrashing :→

- The operating system monitors CPU utilization. If the CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process.
- A global page replacement algorithm is used; it replace pages without regard to the process to which they belongs.

- Now suppose that a process enters a new phase in its execution and needs more frames.
- It starts faulting and taking frames away from other processes.
- These processes need those pages and so they also fault, taking frames from other processes.
- These faulting processes must use the paging device to swap pages in and out.
- As they queue up for the paging device, the ready queue empties.
- As processes wait for the paging device, CPU utilization decreases.
- The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
- The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device.
- Process continues and the page fault rate increases tremendously



lution ⇒

- We can limit the effects of thrashing by using a local replacement algorithm (or priority replacement algorithm)
- In this if one process starts thrashing it cannot steal frames from other process and cause the latter to thrash as well.
- when more than one processes are in thrashing state then they will be in the queue for the paging device.
- Thus the effective access time will increase even for a process that is not thrashing.
- To prevent thrashing we must provide a process with as many frames as it needs.
- But how do we know how many frame it needs?
- In locality model, it says as a process executes, it moves from locality to locality.
- A locality is a set of pages that are actively used together.
- A program is generally composed of several different localities which may overlap.

Locality of Reference

- If a process's program is well structured, the references will follow some pattern in small neighbourhoods in the address space.
- And the system will be able to ensure low page fault rate with only a limited fraction of the process address space in the main memory.
- This is known as locality of reference.
- The better the locality, the better the expected performance from the demand paging system.
- A program ~~section~~ exhibits many different localities based on program structures.
- A program section consists of a set of functions that cause concentration of references in certain localities in the process address space.