KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Website: www.ktunotes.in

OPERATING SYSTEM

MODULE - 4

4.1 MEMORY MANAGEMENT:

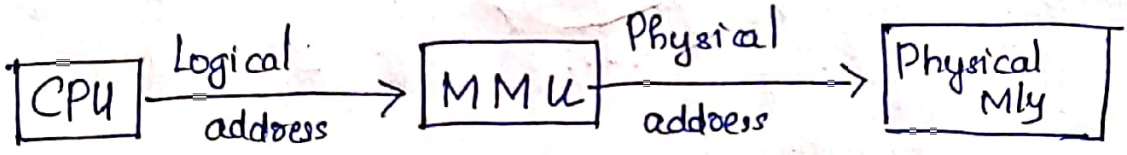
- Memory consists of a large array of words or bytes, each with its own address.
- The CPU fetches instrns from mly according to the value of the pgm counter (PC).
- Main mly & registers built into the processor itself are the only storage that the CPU can access directly.
- Each process has a separate memory space.
- There is a range of legal addresses that the process may access & to the process can access only these legal addresses.
- To provide this protection, two registers are used:
  - Base Register - Holds the smallest legal physical memory address.
  - Limit Register - Specifies the size of the range

Address Binding:

- Address Binding is the process of mapping the programs logical or virtual addresses to the corresponding physical or main mly addresses.
- The binding of instructions & data to mly addresses can be done at any of the foll. steps:
  - ✓ Compile Time
  - ✓ Load Time.
  - ✓ Execution Time

Logical Address Space	Physical Address Space
- Generated by the CPU; Also referred as virtual address.	- Addressed seen by the memory unit.
- Logical address space is the set of all logical addresses generated by the program.	- Physical address space is the set of all physical addresses generated by the pgm.

* Logical & Physical address are the same in compile time & load time address-binding scheme.
* Logical & physical address are different in execution time address binding scheme.

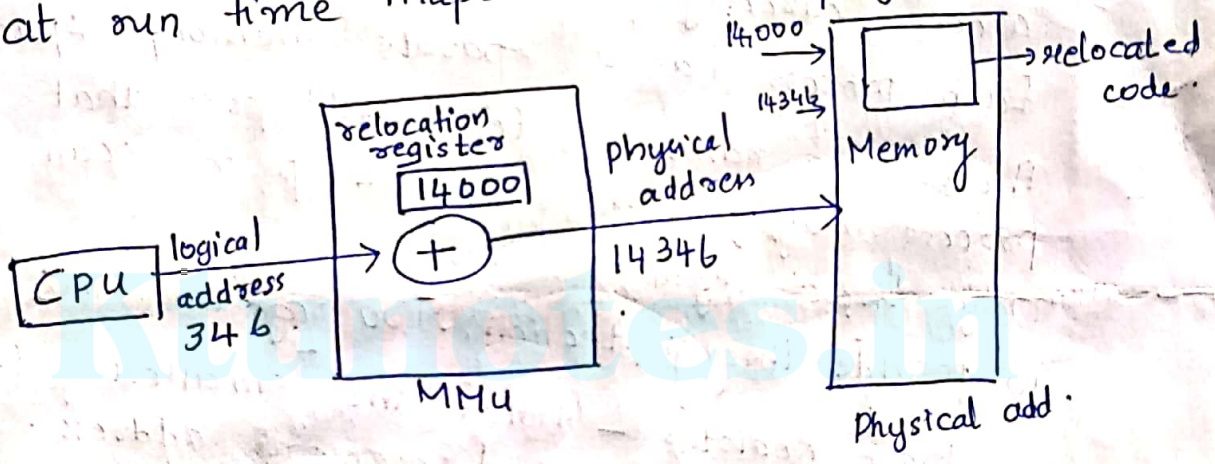


Memory Management Unit

- It is a hardware device that maps virtual address to physical address.
- In MMU scheme, the value in relocation reg is added to every address generated by the user process at the time it is sent to memory.
- The user program ~~always~~ deals with logical address. It never sees the real physical address.

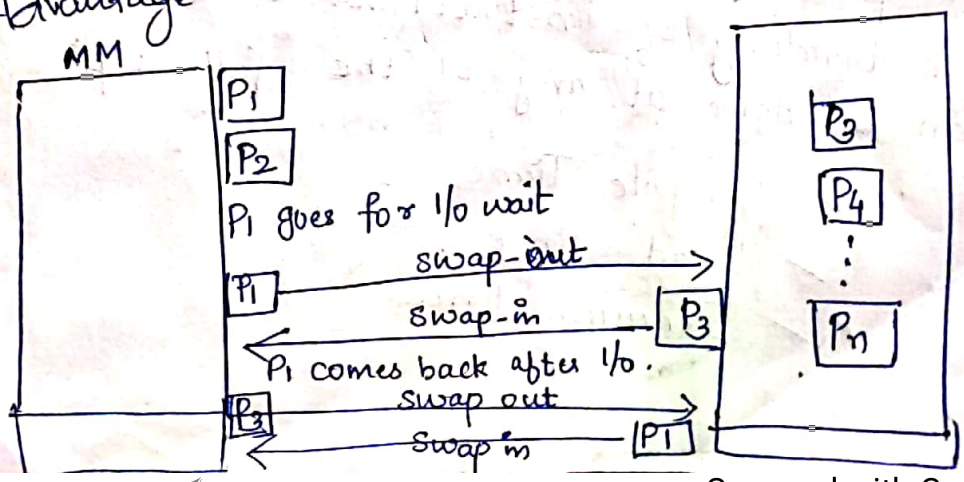
Logical versus Physical Address Space.

- **Logical address (Virtual address)** – Generated by the CPU. Also referred to as virtual address.
- **Physical address** – Address seen by the memory unit.
- Logical address space is the set of all logical addresses generated by a pgm.
- Physical address space is the set of all physical addresses generated by a pgm.
- Memory Management Unit (MMU) – H/w devices that at run time maps virtual to physical address.



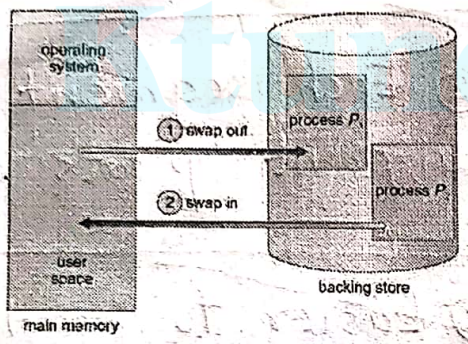
4.2 Swapping.

- Swapping used inorder to implement multi-programming.
- In multiprogramming, can store multiple process in main mly. So that CPU can execute all process in an ebbicient manner.
- Advantage – utilize CPU peformance in efficient manner



- 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
- Swapping variant used for priority-based scheduling algorithms are called roll out, roll in- lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Swapping requires a backing store.
- The **backing store** is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.

- 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.
- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.

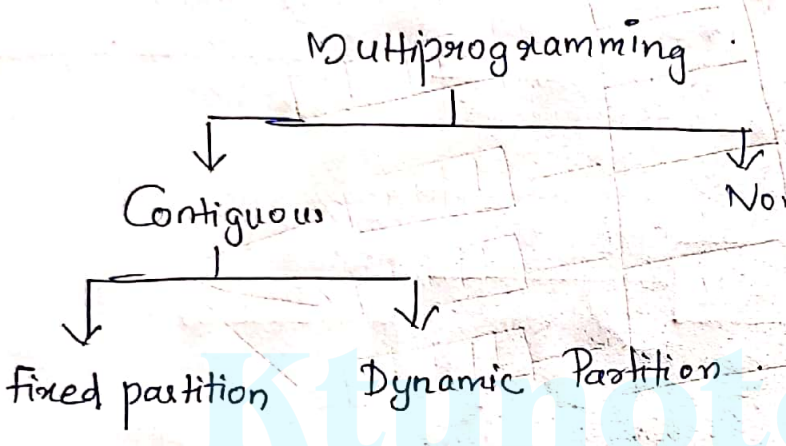


Swapping of two processes using a disk as a backing store.

Contiguous v/s Non-Contiguous Allocation.

- Way of organizing program in m/y

Contiguous	Non-Contiguous.
① Program must exist as a single block of contiguous address.	① Programs divided into chunks called segment.
② Sometimes it is impossible to find large enough block.	② Each segment can be placed in different parts of memory.
③ Low Overhead.	③ Easier to find "holes" (free space) in which segment it will fit



* Fixed Partitioning.

- Main m/y is divided into no. of static partitions at system generation time.
  - ☆ Equal size partition.
  - ★ Unequal size partition.

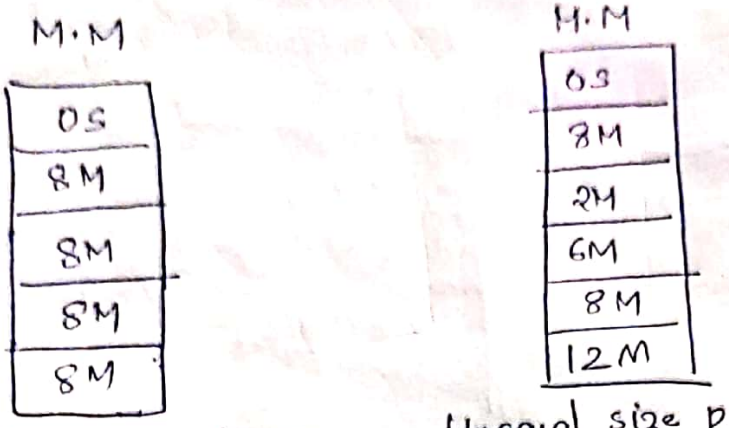
Pros.
- Simple to Implement.
- Little OS overhead.

Cons.
- Inefficient
- Maximum no. of active process - fixed

- Placement Algorithm.

* One process queue per partitioning.
* Single Queue " "

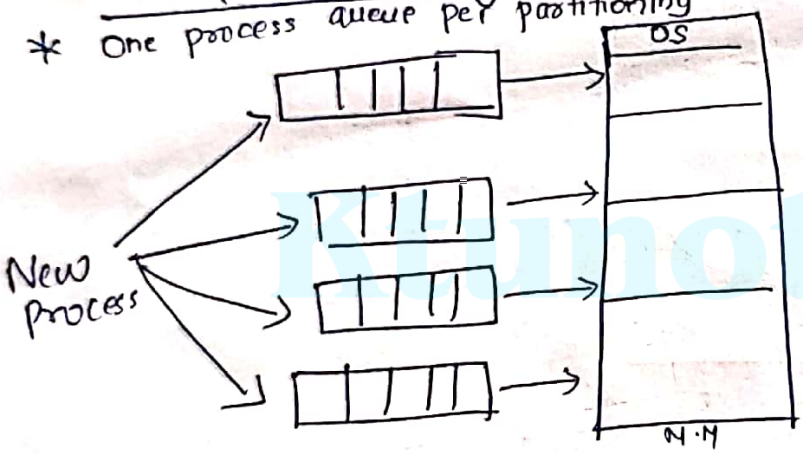
Eg:



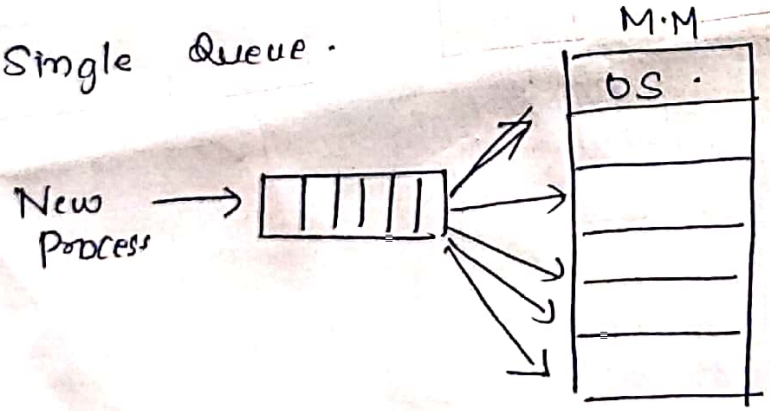
Equal size partition. Unequal size partition

Fixed Partitioning Placement Algorithm.

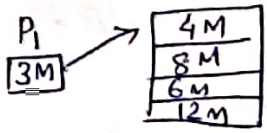
* One process queue per partitioning



* Single Queue.



* Dynamic Partitioning

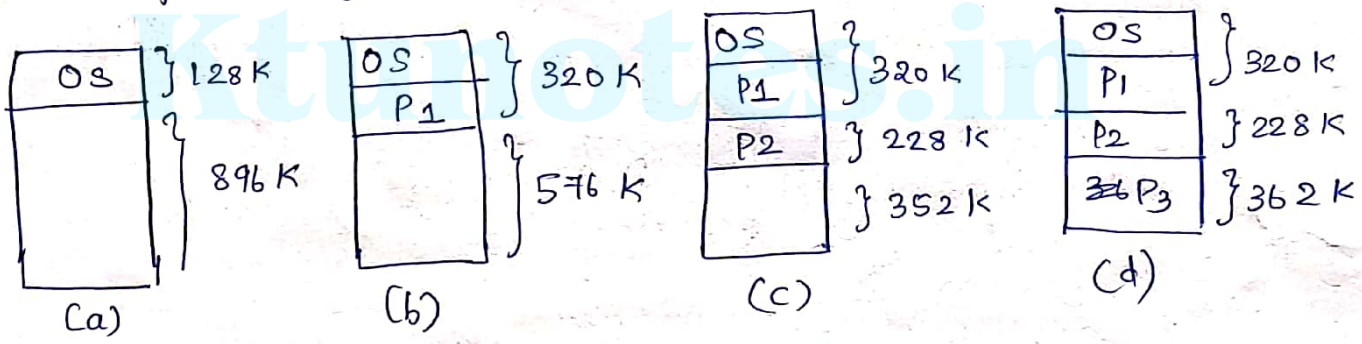


- Partitions are created dynamically.
- Each process is loaded into a partition of exactly the same size of that process.

Placement Algorithm

→ First fit : First hole that is big enough is allocated to progoam.
→ Best fit : Smallest hole that is big enough is allocated to pgm.
→ Worst fit : Largest hole that is big enough is allocated to pgm.

Eg:



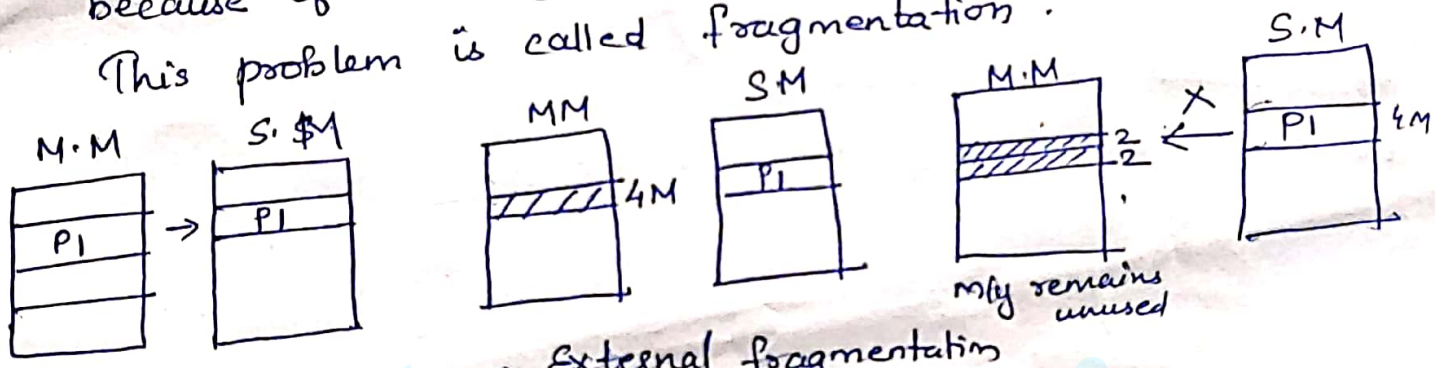
Memory Allocation

- A simplest method 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 partitioned 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 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, a **hole**.Eventually, memory contains a set of holes of various sizes.
- As processes enter the system, they are put into an input queue. The operating system takes into account the memory requirements of each process and the 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 CPU time. When a process terminates, it releases its memory which the operating system may then fill with another process from the input queue.
- In general, 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. At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes.
- **Dynamic storage allocation problem** concerns how to satisfy a request of size n from a list of free holes. Solutions for this problems are
  - **First fit.** Allocate the **first hole** that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
  - **Best fit.** Allocate the **smallest hole** that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
  - **Worst fit.** Allocate the **largest hole.** 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 the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- As processes are loaded and removed from memory, the free memory space is broken into little pieces.

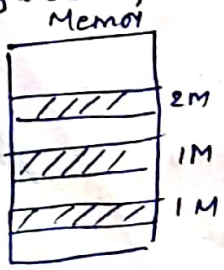
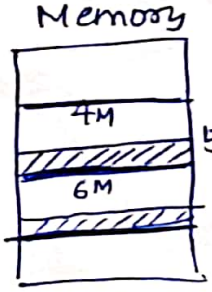
- **External fragmentation** exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous; storage is fragmented into a large number of small holes.
- In the worst case, it could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, it might be able to run several more processes.
- Memory fragmentation can be internal as well as external.
- 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.
- 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.
- **Compaction** is the solution for external fragmentation- shuffles the memory contents so as to place all free memory together in one large block.

FRAGMENTATION

- A process is loaded & removed from memory, the free memory space is broken into little pieces.
- After sometimes, that process cannot be allocated to memory because of small size & memory block remains unused. This problem is called fragmentation.



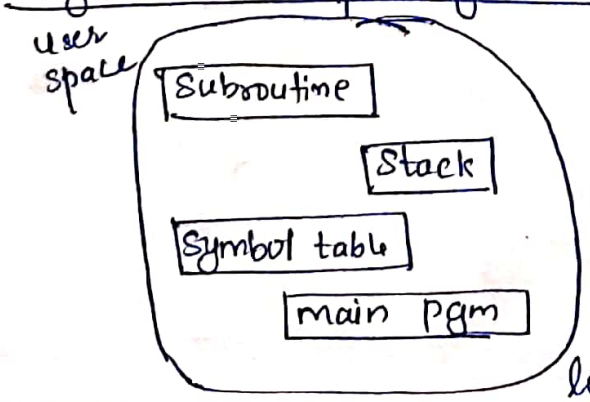
Fragmentation → External fragmentation
Fragmentation → Internal fragmentation

External fragmentation	Internal fragmentation
- Total memory space is enough to satisfy a request or reside a process in it, but it is not contiguous, so it cannot be used	- Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.
 $P_1 = 4M$	 $P_1 = 4M$ $P_2 = 6M$.

SEGMENTATION

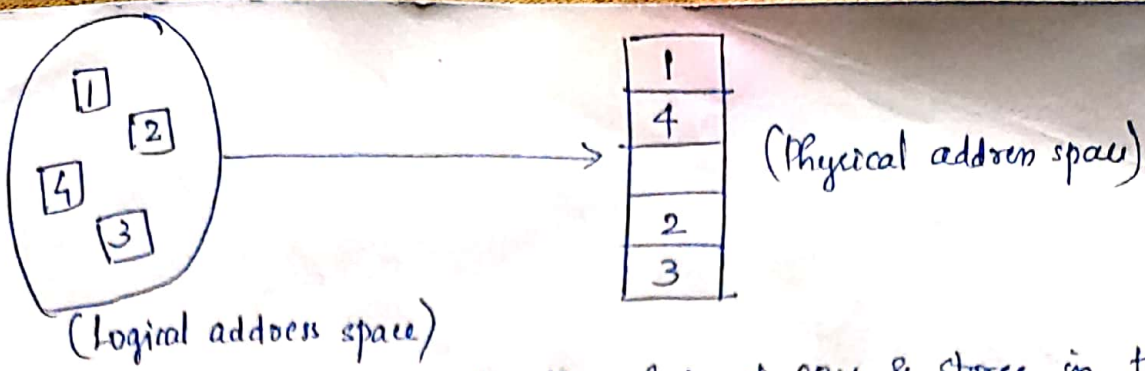
* A program is a collection of segments.
* A segment is a logical unit such as:
  - Main program.
  - Procedure.
  - Function.
  - Local variables, global variables
  - Common block.
  - Stack
  - Symbol Table
  - Array.

Logical View of Segmentation



* User specifies each address by 2 quantities:
  (a) Segment name.
  (b) Segment offset.

logical address ⇒ < segment name, offset)



* Log. add. is executed with the help of CPU & stores in the phy. mly space in the form of frames.

Segmentation Architecture.

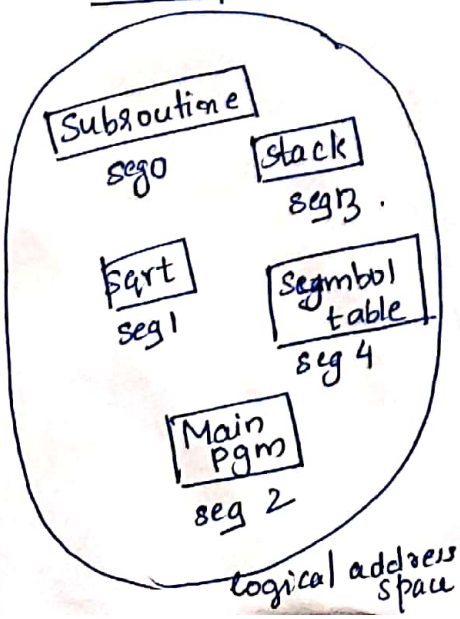
* Logical address contains 2 tuples :
  < seg no, offset >.
* Segment table maps 2 dimensional physical address.
* Each table entries have :
  * Base. — ~~Stor~~ contains starting physical address.
  * Limit — Specifies the length of segment.

seg. table

base	limit

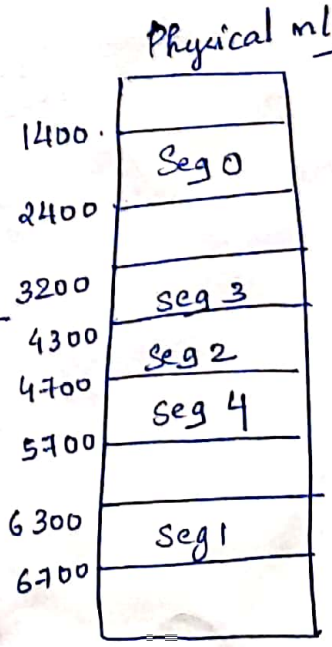
* Segment table base Register (STBR) —
  Points to the segment ~~table's~~ location in mly.
* Segment table Length Register (STLR) —
  Indicates no. of segments used by program.
* Segment no, S is legal if, S < STLR.

Example of Segmentation.

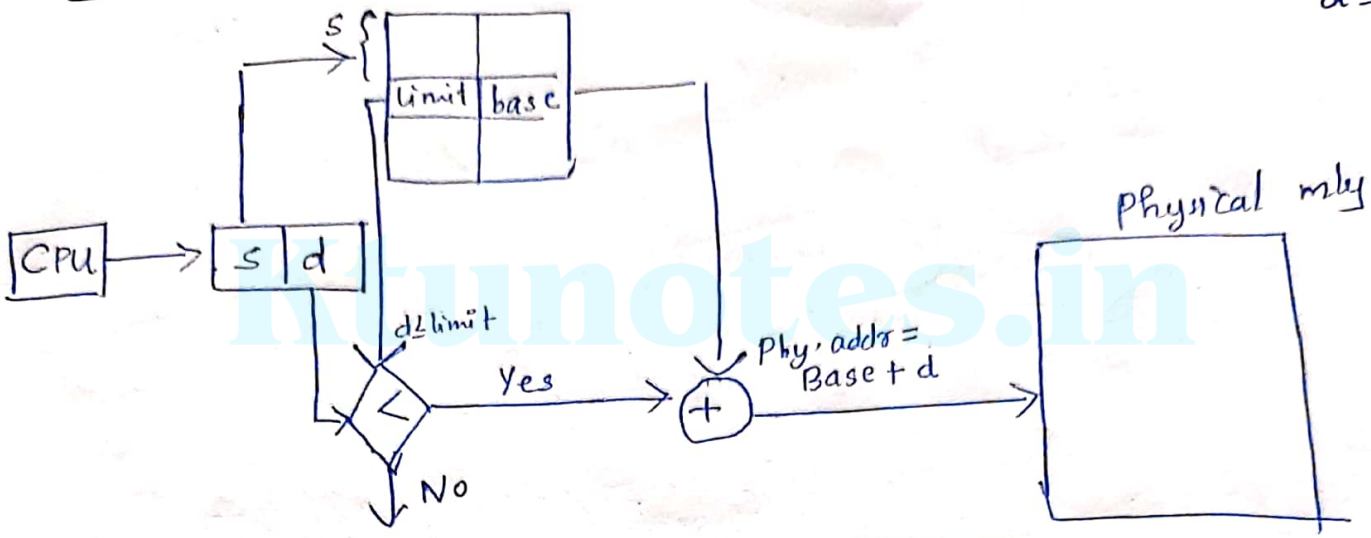


segment table

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



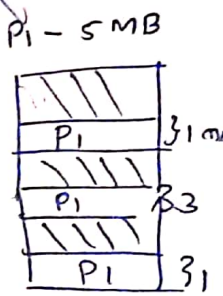
Segmentation Hardwase



trap: addressing error

PAGING

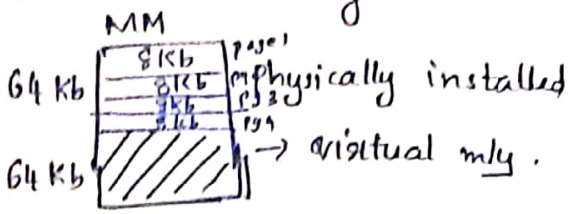
* Non-Contiguous allocation (Virtual Memory). → wherever free space, OS can make use of free space
  - To provide easy to the users.
  - To increase CPU utilization



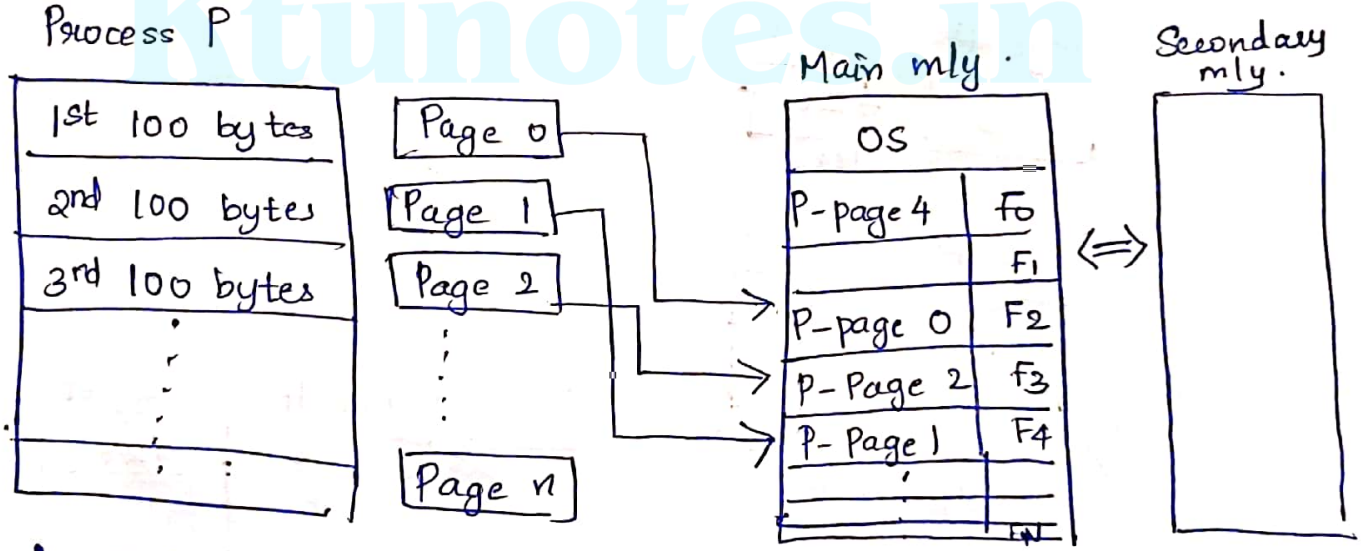
* OS giving illusion to the user such that
  - User can write a very big program
  - User thinks that its entire program is present in RAM.
  - All the space allocated to the user is contiguous.

In reality, only a small portion of the user program is in RAM, which may or maynot be contiguous. While the remaining pgm is in secondary memory.

* A computer can add more memory than the amount physically installed on system. This extra memory is called virtual memory.

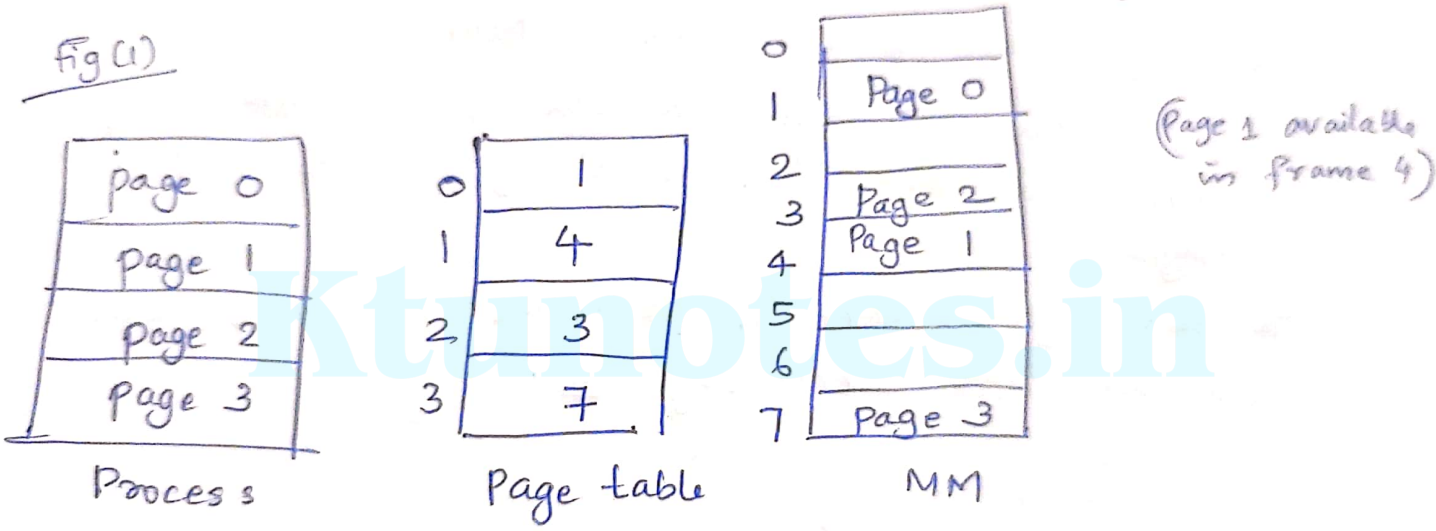


* Paging is a memory management technique in which process address space (virtual mly) is broken into blocks of same size called pages. Size of each page is same.
* ~~Page~~ Block size → power of 2 (2, 4, 6, 8, ...)
* The size of process can be measured in no. of pages. Similarly, main mly is divided into small fixed blocks (physically) of memory called frames. Size of each frame is same.
* Size of frame = Size of page.
  - We can avoid External fragmentation.



- A single process is divided into fixed slots called Pages. These pages need to be stored in MM.
- MM is divided into frames & virtual mly into pages. Whatever data send from virtual mly to MM is in form of pages.
- ~~To make~~ After execution, to make MM free, can swap data into secondary Mly.

Fig (1)



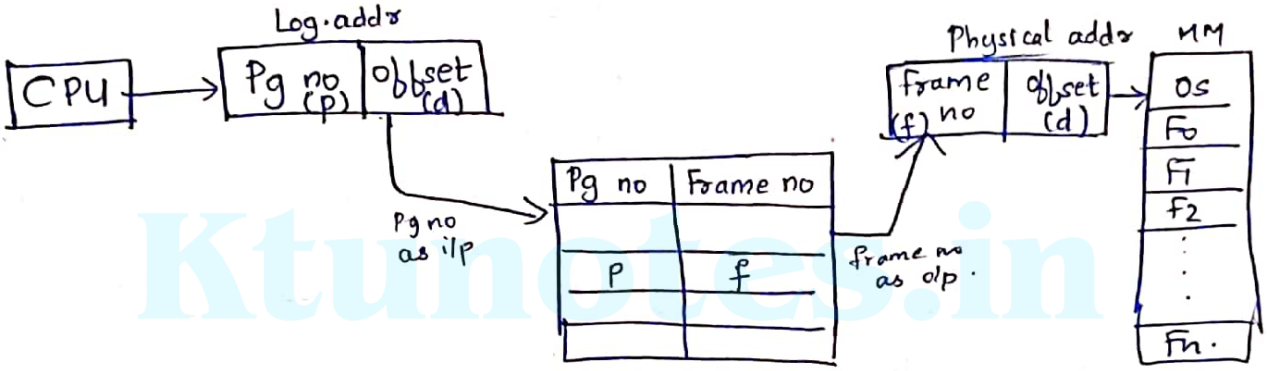
Address Translation (Paging Hardware)

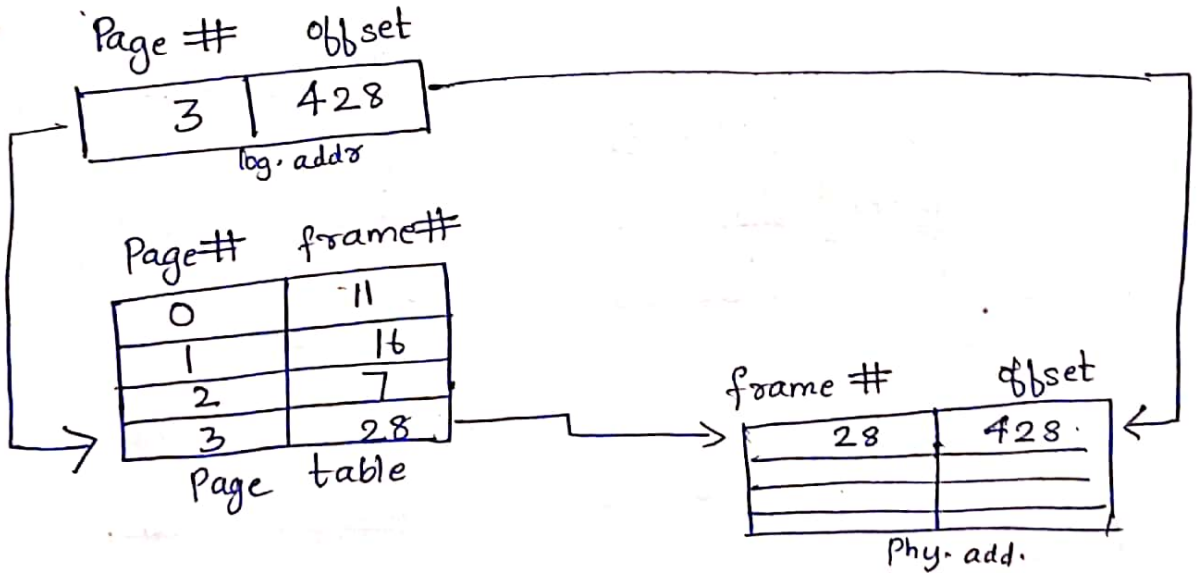
- Page address is logical address & represented by,

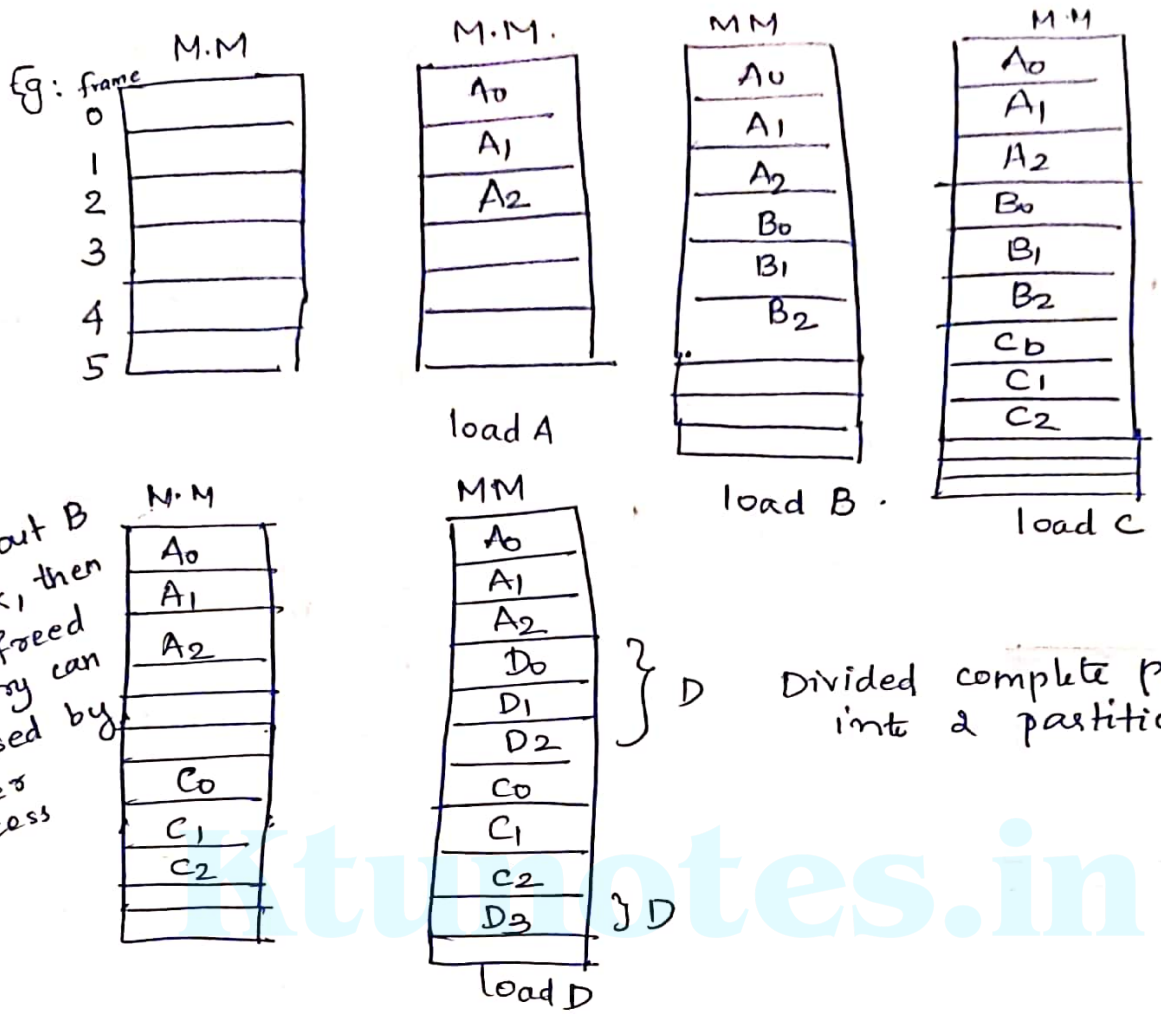
Logical addr = Pg no + page offset

- Frame address is called physical address & represented by,

Physical addr = Frame no + page offset







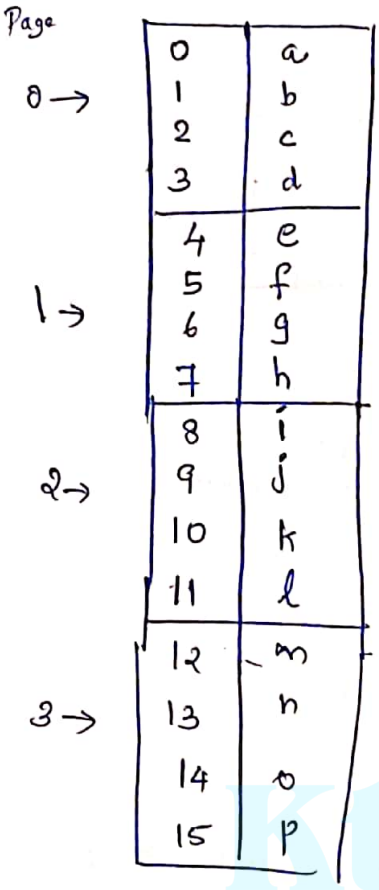
Advantages & Disadvantages

1. Paging reduces external fragmentation but still suffer from Internal fragmentation.
2. It is simple to implement.
3. Swapping becomes very easy because of ~~same~~ equal sizes of pages & frames.
4. It is not suitable for small RAM's.

Paging Example for a 32-byte memory with 4-byte pages

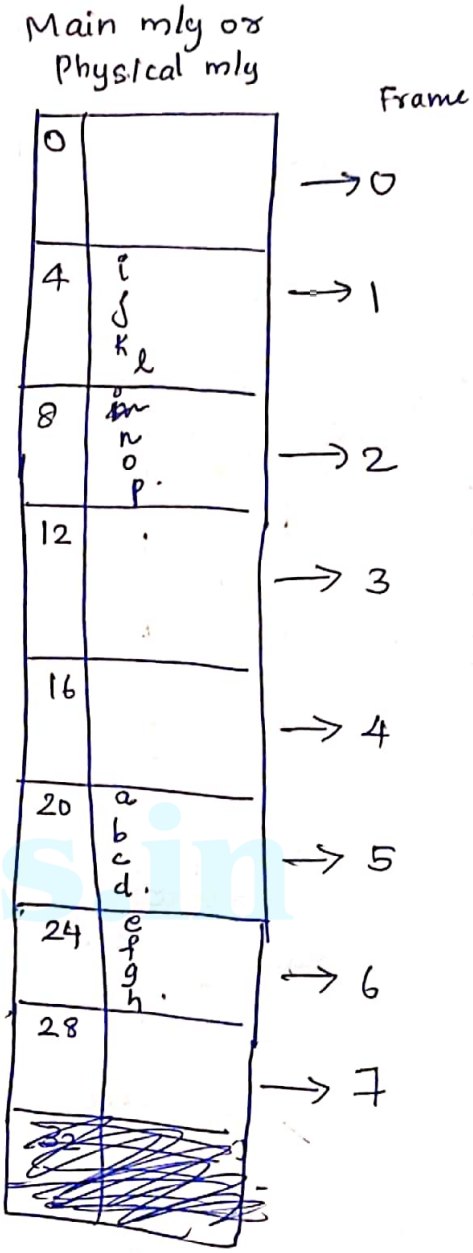
Process

Logical mly.



Page table

0	5
1	6
2	1
3	2



- Size of page (4 bytes) = Size of frame (4 bytes)

Log addr = 0 (a)
Page = 0.
offset = 0.
Phy. addr = fr. no × size + offset
= 5 × 4 + 0 = 20.

L.A = 1.
Pg = 0.
offset = 1
Phy addr = 5 × 4 + 1
= 21 (b)

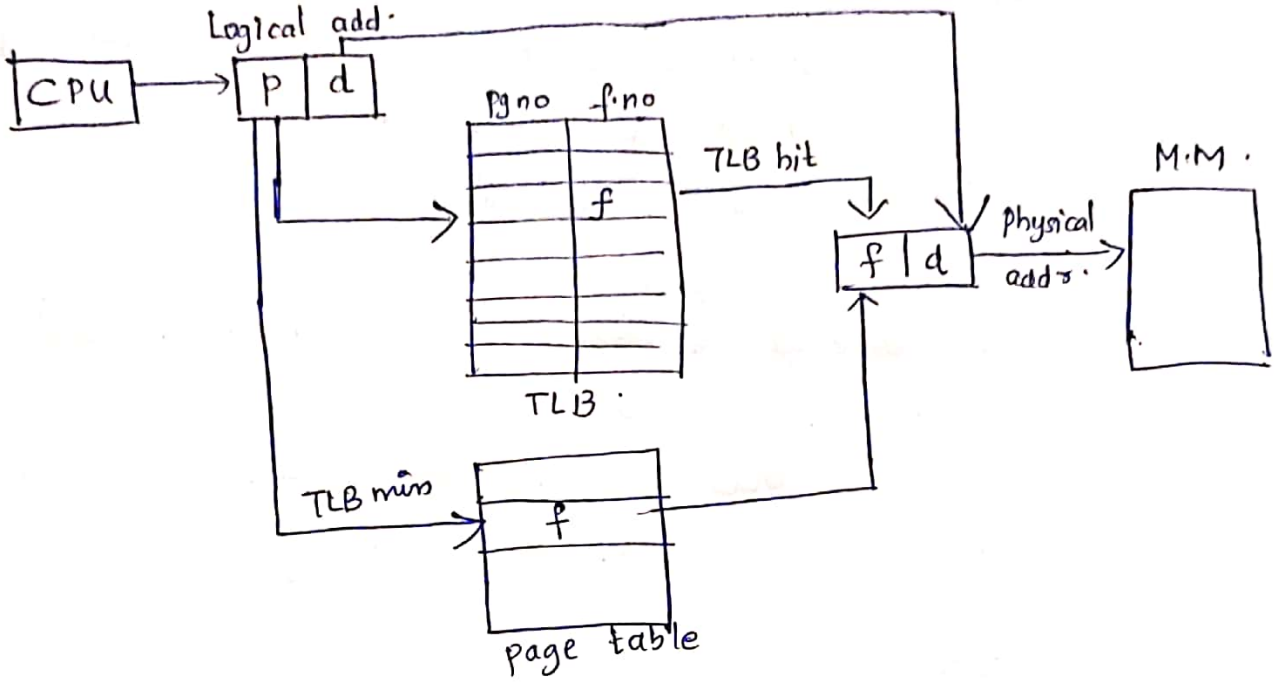
LA = 4.
Pg = 1
offset = 0
P.A = 5 × 4 + 0
= 24 (e)

LA = 15
Pg = 3
offset = 3
PA = 2 × 4 + 3
11 (p)

Translation Look-aside Buffer (TLB)

(or)

Paging Hardware with TLB

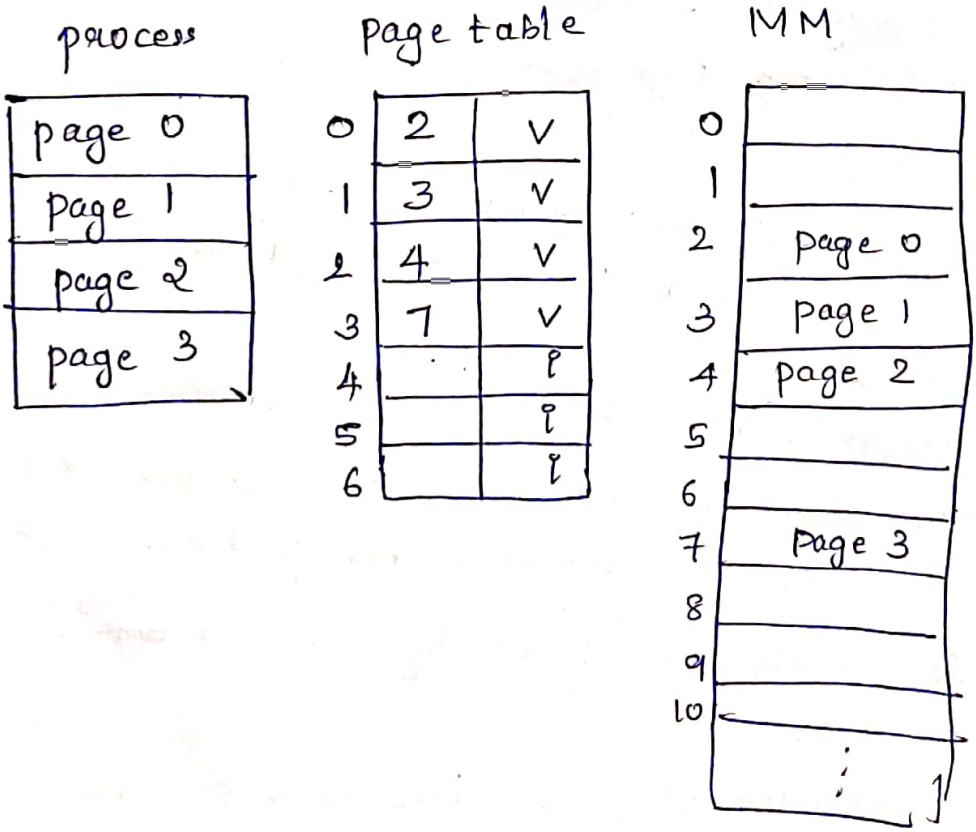


* Page table is stored inside MM.
* 2 memory access required:
  - Accessing page table
  - Accessing MM.
* CPU generate LA (pg no, offset). Pg no given as i/p to page table. So inorder to access pg table, requir 1 m/y access. Frame no as o/p — Phy. add. (2nd m/y. access).
* ~~To overcome this~~ 2 m/y access required for executing a single instn. To overcome this problem, TLB is used.
* TLB contains the most frequently accessed ~~entries~~ pages in page table.
* CPU generated LA [p, d]. Page no is compared with every page no. in TLB.

- If match found, it is TLB hit & gives frame no. as o/p.
  & the frame no is combined with offset will give physical address (1 mly access required)
- If match not found, it is TLB miss.
  Page no given as i/p to page table. Produces frame no as o/p & the frame no is combined with offset will give physical address.
  After getting frame no, the corresponding entry ie, page no & frame no information will be updated in TLB.
  Suppose TLB is full, one of the row will be replaced with this new entry.

Protection.

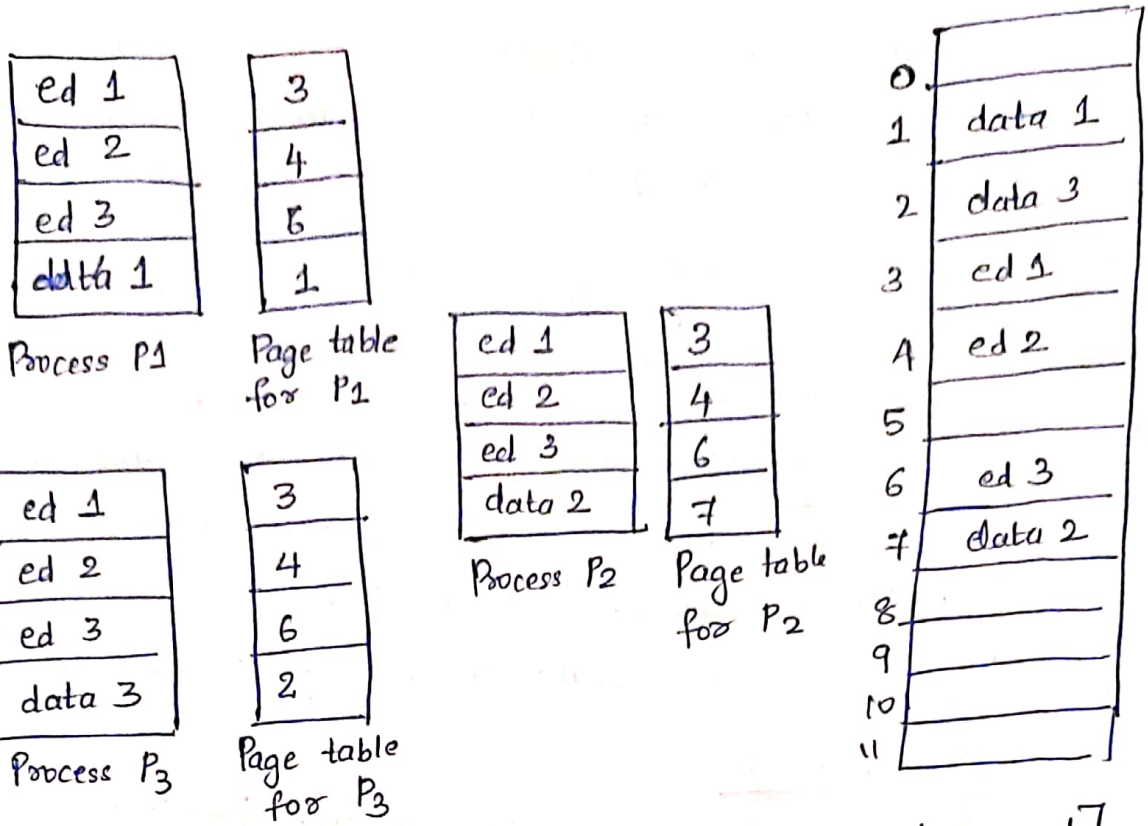
* ~~process~~ How to provide protection in paging.



* Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.
* Valid-Invalid bit attached to each entry in the page table.
  - "Valid" (v) indicates that the associated page is in the process's Logical address space & is thus a legal page.
  - "Invalid" (i) indicates that the page is not in the process's logical address space.
* Any violations, result in a trap to the kernel.

Shared Pages.

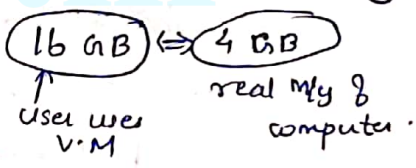
* An advantage of paging is the possibility of sharing common code.
* Shared code
  - One copy of read-only code shared among processes (ie, text editors)
  - Similar to multiple threads sharing the same process space.
* ~~P Ato~~.
* Private code & data.
  - Each process keeps a separate copy of the code & data.
  - The pages for the private code & data can appear anywhere in the logical address space.



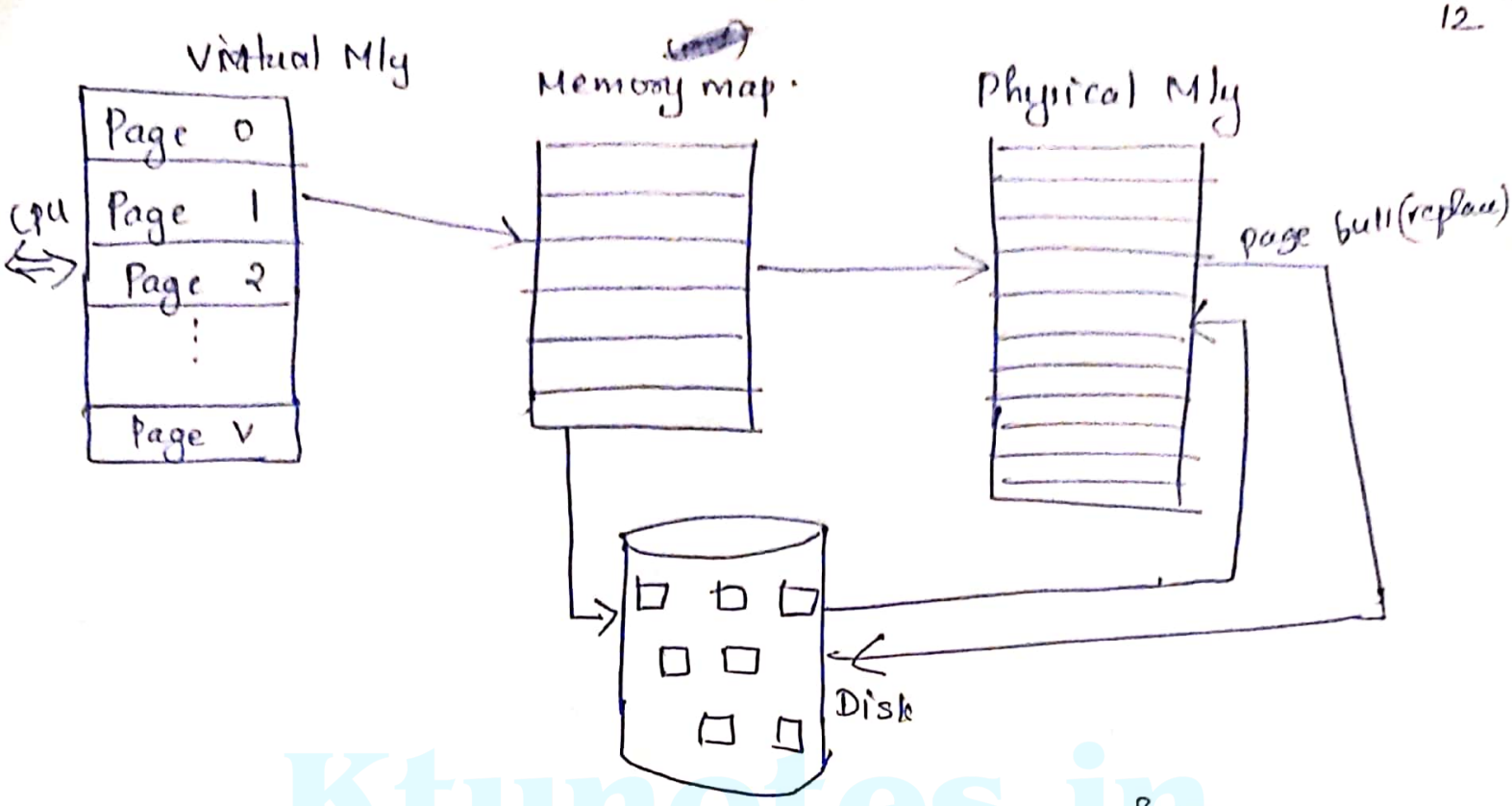
[Sharing of code in a paging environment]

VIRTUAL MEMORY.

* Something which appear to be present but actually it is not.
* It just gives illussion to users.



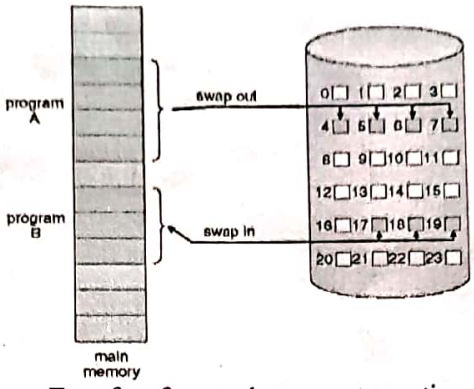
* A programmer can write a pgm which requires more memory than the capacity of MM. Such a pgm is executed by virtual m/y technique.
* Only part of pgm needs to be in MM for execution.
* Logical address space can therefore be much larger than Physical address space.
* Allows address spaces to be shared by several processes.
* More pgms running concurrently.
* V.M requires less I/o needed to load or swap the process.



* Pages in VM are stored in mly map & their information from Send to physical mly.
* If MM doesn't contain the page that CPU is demanding, if page not present in M.M, it will get from disk & store in physical mly in form of disk.
* If physical mly is full, an unused page will be swapped to the disk.
* VM can be implemented via demand paging & demand segmentation.

5.6 Demand paging

- Loading the entire program in physical memory at program execution time is not so good because the user doesn't *need* the entire program in memory at a time.
- An alternative strategy is to load pages only as they are needed. This technique is known as **demand paging.**
- 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.

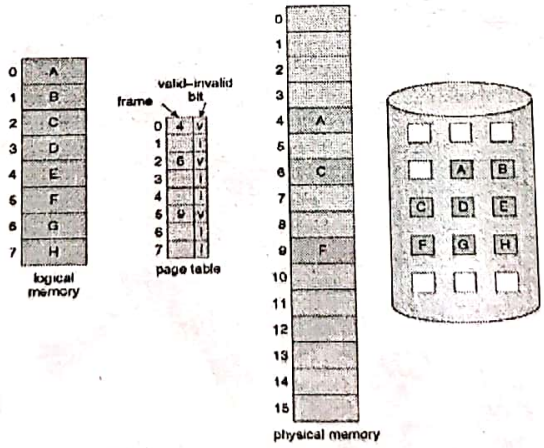


Transfer of a paged memory to contiguous disk space.

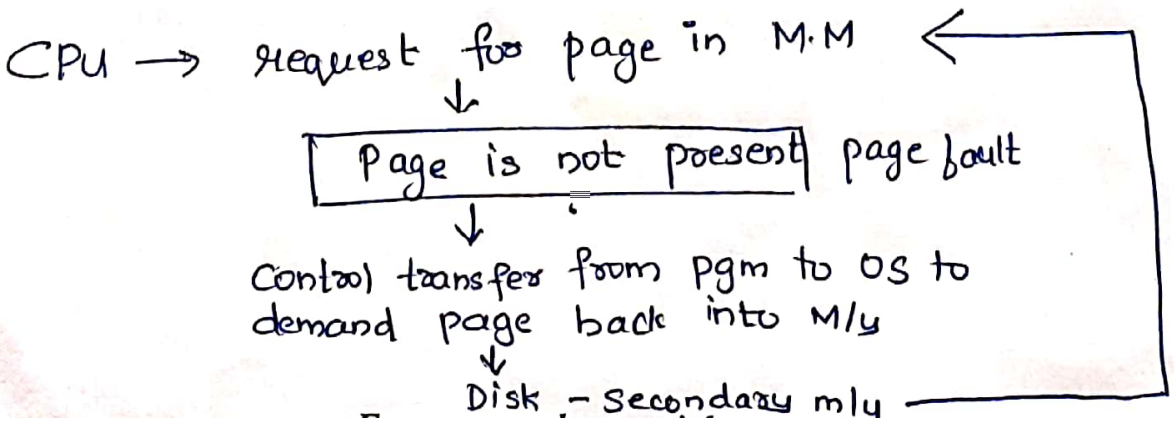
- A demand-paging system is similar to a paging system with swapping.
- But here it uses a lazy swapper; where it never swaps a page into memory unless that page will be needed.
- In Demand paging, **pager** is more suitable than swapper .

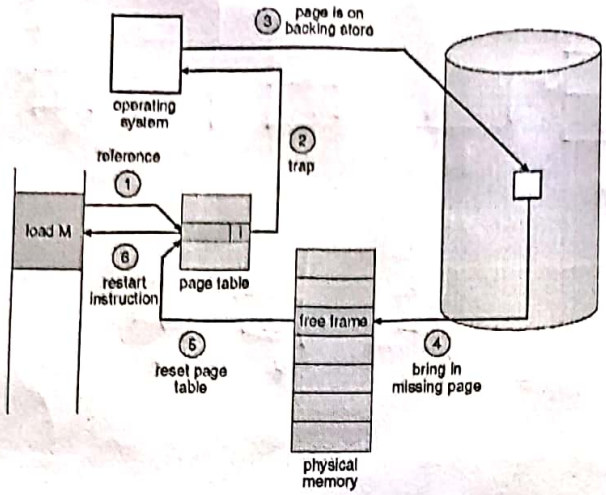
Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- some form of hardware support is needed 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.
- 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
- During MMU address translation, if valid–invalid bit in page table entry is i then a page fault occurs.



Page table when some pages are not in main memory





Steps in handling a page fault.

- If there is a reference to a page, first reference to that page will trap to operating system:
  - page fault
- The procedure for handling this page fault is
  1. 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, terminate the process. If it was valid, but have not yet brought in that page, page it in.
  3. Find a free frame.
  4. Schedule a disk operation to read the desired page into the newly allocated frame.
  5. When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.
  6. Restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.
- In the extreme case, we can start executing a process with no pages in memory. When the operating system sets the instruction pointer to the first 404 Chapter 9 Virtual Memory 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
- Theoretically, some programs could access several new pages of memory with each instruction execution, possibly causing multiple page faults per instruction. This situation would result in unacceptable system performance. Locality of reference can be used for such situations.
- Hardware support needed for demand paging

  • **Page table.** This table has the ability to mark an entry invalid through a valid–invalid bit or a special value of protection bits.

  • **Secondary memory.** This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known as swap space

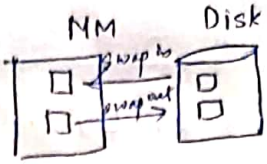
Performance of Demand Paging

- Demand paging can significantly affect the performance of a computer system.
- the effective access time is

Effective access time= (1 - p) x ma + p x page fault time.

- where,
  - p be the probability of a page fault (0 <=p <=1).
  - ma be the memory-access time

If p=0, no page fault
p=1, every reference is a fault.



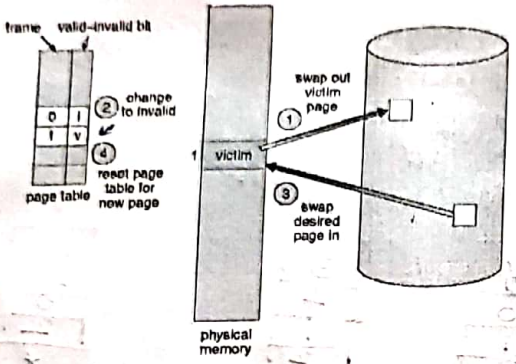
Page Replacement

- It prevents over-allocation of memory by modifying page-fault service routine to include page replacement
- Use modify (dirty) bit to reduce overhead of page transfers.

Basic Page Replacement

- Page replacement takes the following approach.

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 a victim frame
   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.



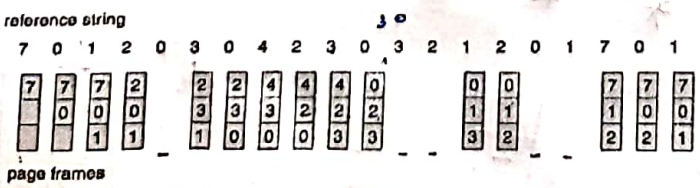
page replacement

- 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.
- two major problems to implement demand paging
  - Frame allocation algorithm
  - page replacement algorithm
- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available

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.
- Example:
  - Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
  - 3 frames (3 pages can be in memory at a time per process)

Page Hit = 7.
Page fault = 15
Hit ratio = no.of hit / Total no.of reference
= 7/22 x 100.



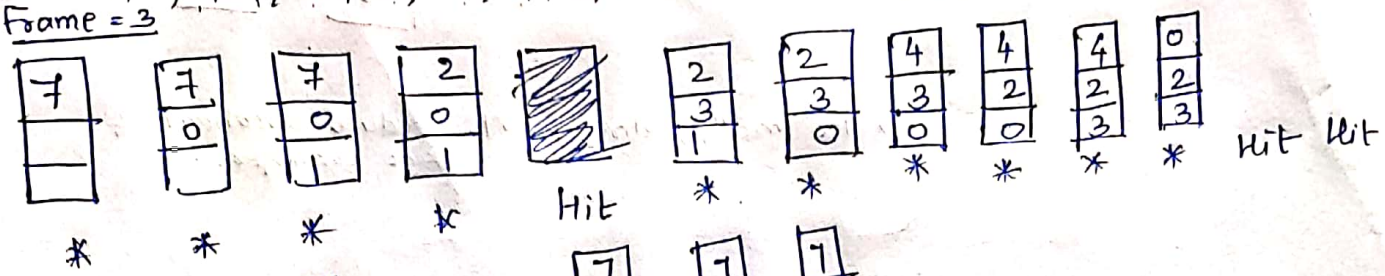
- **Belady's Anomaly**-for some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

Page fault ratio = no.of Page faults / Total no.of ref
= 15/22 x 100.

First In First Out (FIFO)

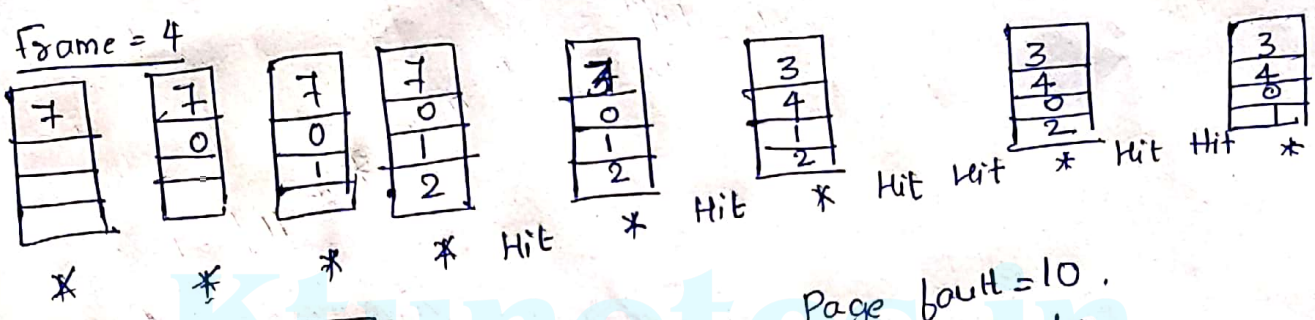
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Frame = 3



Page fault = 15
Page Hit = 5.

Frame = 4



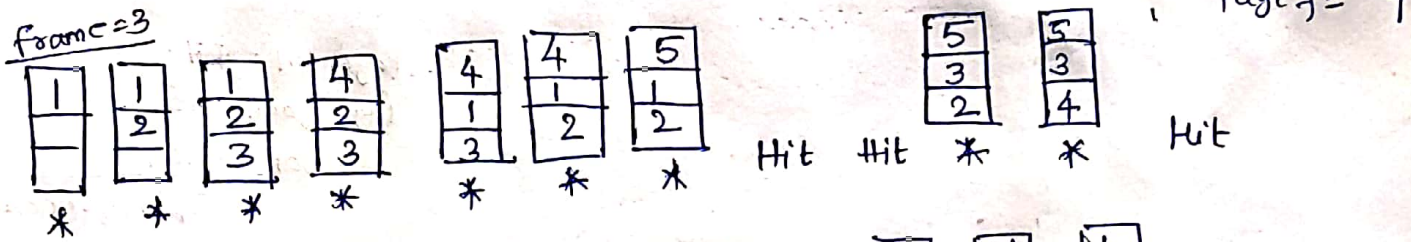
Page fault = 10.
Page Hit = 10.

As no. of frames ↑, page fault ↓.

Belady Anomaly in FIFO.

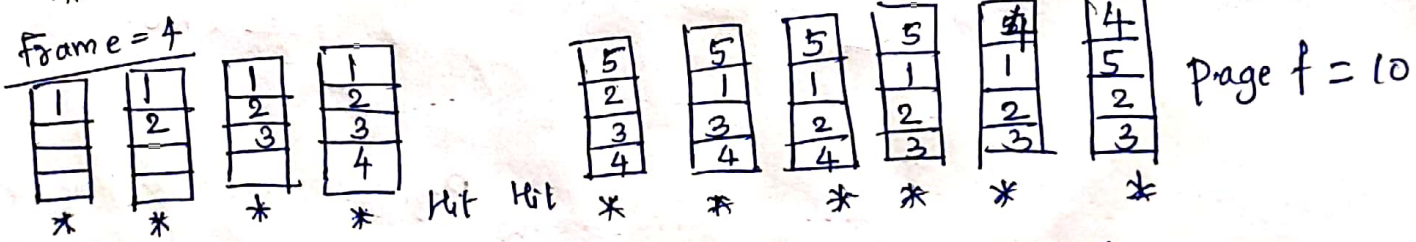
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Frame = 3



Page f = 9

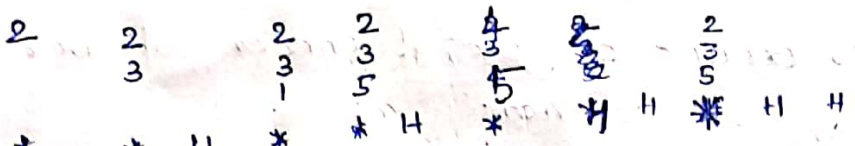
Frame = 4



Page f = 10

As no of frames ↑, page fault also ↑. This unexpected condition is called belady's ~~Algo~~ Anomaly.

OPT → 2 3 2 1 5 2 4 5 3 2 5 2 P.f = 6. Hit = 6. Hit r = 6/12 Miss r = 6/12.



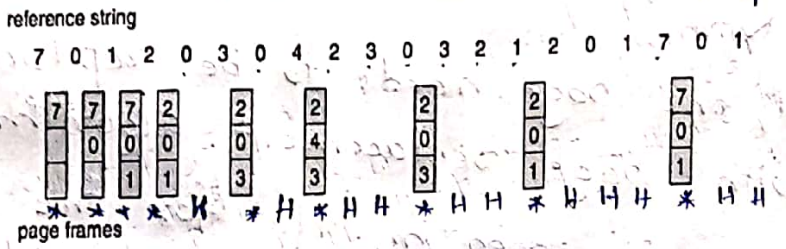
- Example: For the reference string 1,2,3,4,1,2,5,1,2,3,4,5

  page fault is 9 with frames 3 and page fault is 10 with frames 4.

Optimal Page Replacement

- Replace page that will not be used for longest period of time
- **It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly-hence known as OPT or MIN.**

P.f = 9, Hit = 11, Hit ratio = 11/20, Miss ratio = 9/20.



- It is difficult to implement, because it requires future knowledge of the reference string.

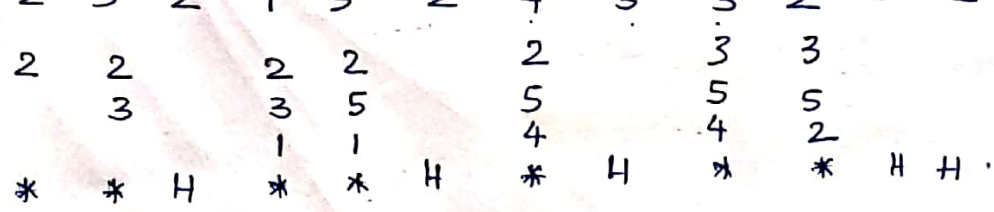
LRU Page Replacement (Least Recently Used Alg (LRU))

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page

P.f = 12.

- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- Two methods of implementations:
  - Counter implementation
    - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
      - When a page needs to be changed, look at the counters to find smallest value (Less frequently used)
        - Search through entire table needed
  - Stack implementation
    - Keep a stack of page numbers in a double link form:
    - Page referenced:
      - move it to the top
      - requires 6 pointers to be changed
    - But each update more expensive
    - No search for replacement
  - LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly

LRU → 2 3 2 1 5 2 4 5 3 2 5 2 P.f = 7.



FIFO:

Pages:- 3 2 1 3 4 1 6 2 4 3 4 2 1 4 5 2 1 3 4

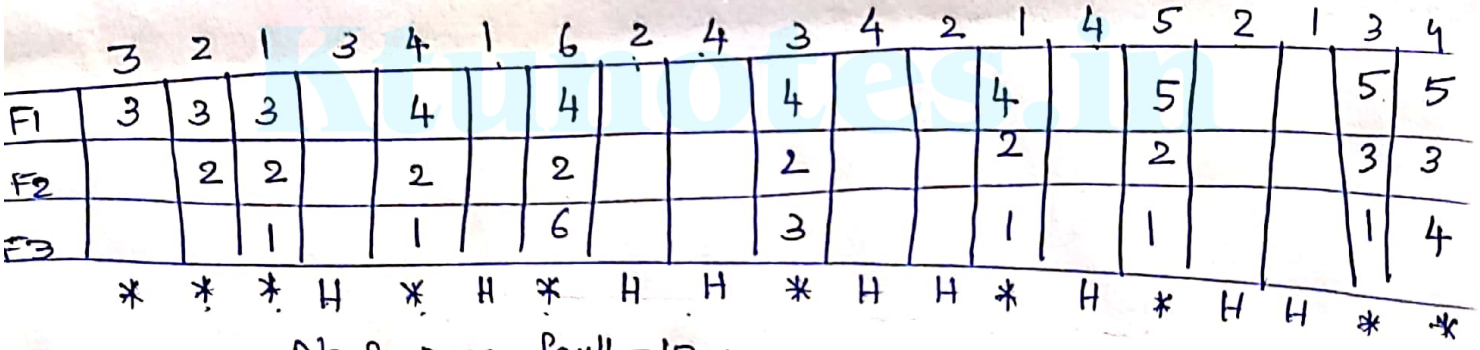
No. of page faults = 13.

Hit ratio = No. of hit / Total = 6/19

Miss ratio = No. of miss / Total = 13/19

OPT

Replace a page which is not in used in near future for a long time.



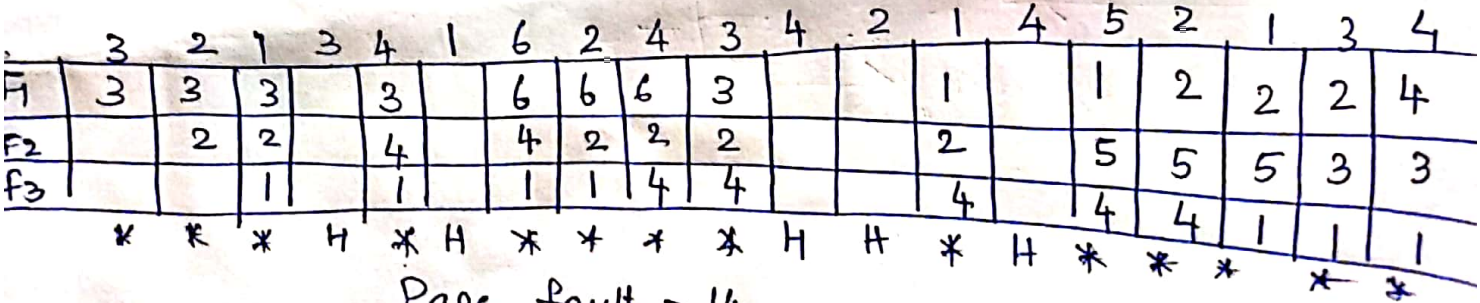
No. of page fault = 10.

Hit = 9

Hit ratio = 9/19

LRU

Replace a page which is not in use for a long time



Page fault = 14.

hit ratio = 5/19.

Compare FIFO, OPT, LRU.

* Whenever CPU needs to execute a page, it checks whether that page is present in MM or not.
If page present in MM, then CPU executes that page.
If page not present in MM, then it has page fault.
Whenever a page fault occurs, OS loads page from Hard disk to MM.
* If MM is full, one page needs to be replaced. That is decided with page replacement algorithm.

FIFO - Page that has been in mly for the longest period of time will be replaced ie, oldest page-replac

OPT - Page that will not be used for longest period of time in the future will be replaced

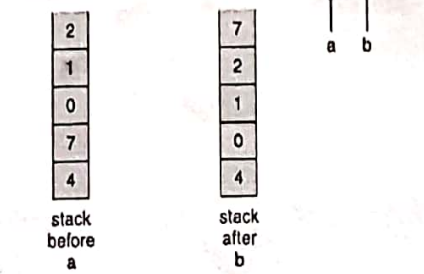
LRU - Page that is least recently used in the past will be replaced

* OPT is the best alg. bcoz OPT produces less no. of page faults than FIFO & LRU. But it is impossible to implement OPT alg practically bcoz OPT needs to know future page infoomation also.
* LRU is best compared to FIFO as it produces less page fault than FIFO.
* In increasing frame size, no. of page faults must be less.
* In FIFO, it has belady's anomaly. (ie, eventhough by increasing the no. of frames, the no. of page faults was also more)

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

reference string

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



Top → most recently used page
bottom → less "
updation → expensive.

Use of a stack to record the most recent page references.

LRU-Approximation Page Replacement

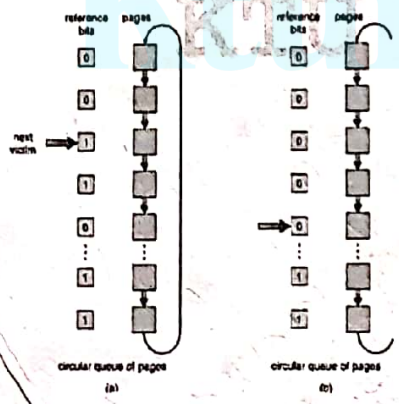
Additional-Reference-Bits Algorithm

Disadv → Cannot know the order which is most recently or least recently referenced.

- With each page associate a bit, initially = 0
- When page is referenced bit set to 1
- Replace any with reference bit = 0 (if one exists)

Second-Chance Algorithm

- The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page has been selected, inspect its reference bit.
- If the value is 0, proceed to replace this page; but if the reference bit is set to 1, give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced.
- One way to implement the second-chance algorithm (the clock algorithm) is as a circular queue. A pointer indicates which page is to be replaced next.
- When a frame is needed, the pointer advances until it finds a page with a 0 reference bit



oldest page will be replaced from the m/y first.

Each page given a second chance, that means if that page is most frequently used page it will be in m/y

Second-chance (clock) page-replacement algorithm

Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
- Take ordered pair (reference, modify)
  1. (0, 0) neither recently used not modified – best page to replace
  2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
  3. (1, 0) recently used but clean – probably will be used again soon
  4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement

High priority ← should not be replaced

- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non- empty class
  - Might need to search circular queue several times

Page - 1, 2, 3 – Class 1 → must selected for replacement

Page - 2, 3 – Class 2 → " "