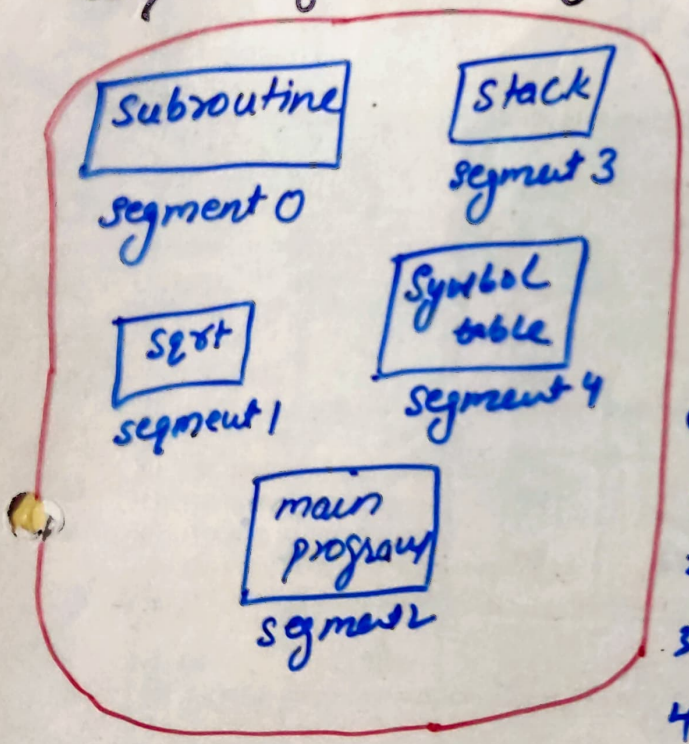
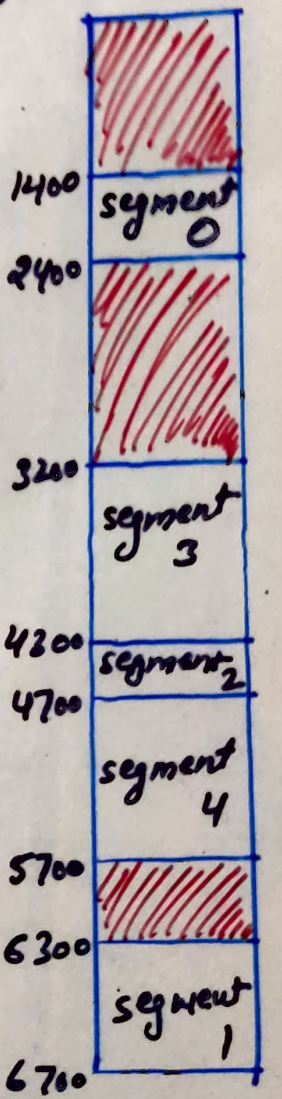
Hardware:- Although the user can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one-dimensional sequence of bytes. Each entry in the segment table has a segment base and segment limit. The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.



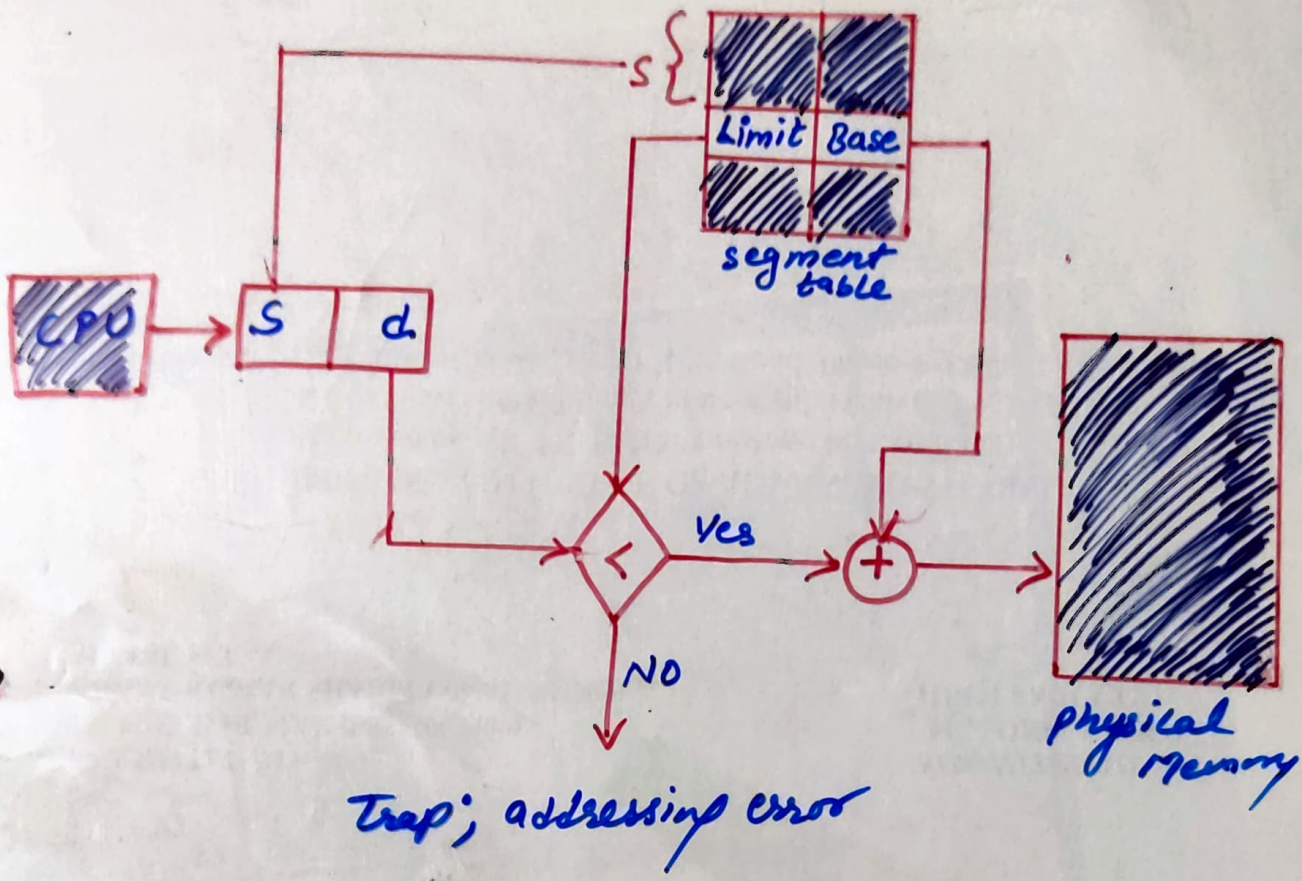
Logical address space

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

segment Table



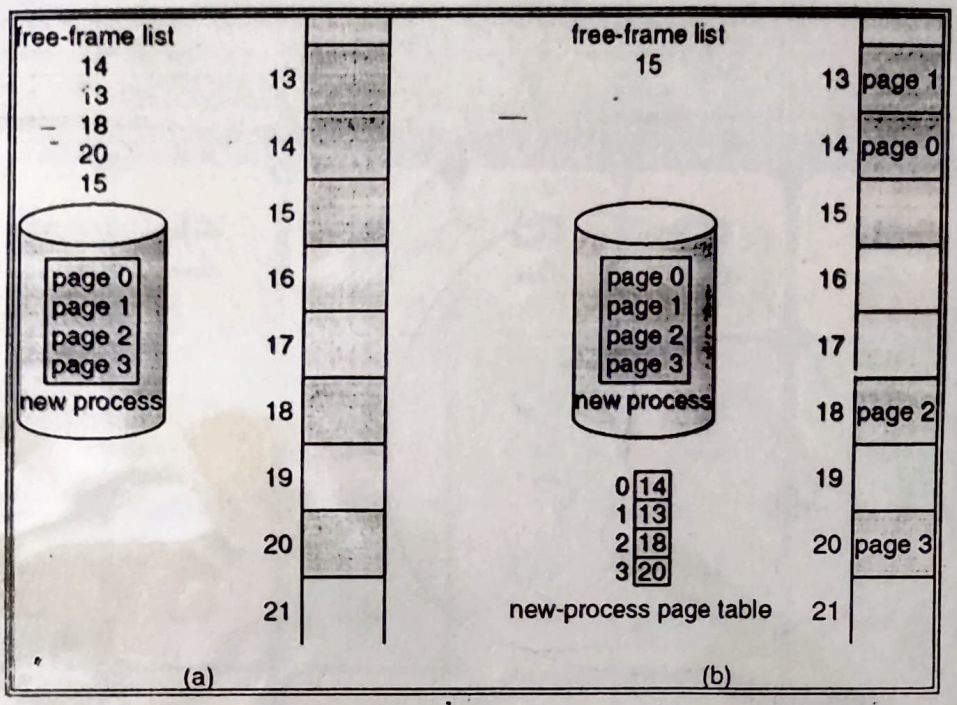
Physical Memory



Segmentation hardware

A logical address consists of two parts: a segment number, s, and an offset into that segment, d. The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system. When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.

FREE FRAMES



When a process arrives in the system to be executed, its size, expressed in pages, is examined. Each page of the process needs one frame. Thus, if the process requires n pages, at least n frames must be available in memory. If n frames are available, they are allocated to this arriving process.

An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory. The difference between the user's view of memory and the actual physical memory is reconciled by the address translation hardware. The logical addresses are translated into physical addresses.

Implementation of Page Table

Each operating system has its own methods for storing page tables. The hardware implementation of the page table can be done in several ways. In the simplest case, the page table is implemented as a set of dedicated registers.

- Page table is kept in mair memory

- *Page-table base register* (PTBR) points to the page table
- *Page-table length register* (PRLR) indicates size of the page table

The problem with this approach is the time required to access a user memory location. If we want to access location i, we must first index into the page table, using the value in the PTBR offset by the page number of i. This task requires a memory access. It provides us with the frame number, which is combined with the page offset to produce the actual address. So, in this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs).** Each entry in the TLB consists of two parts: a key(or tag) and a value. When the associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned.

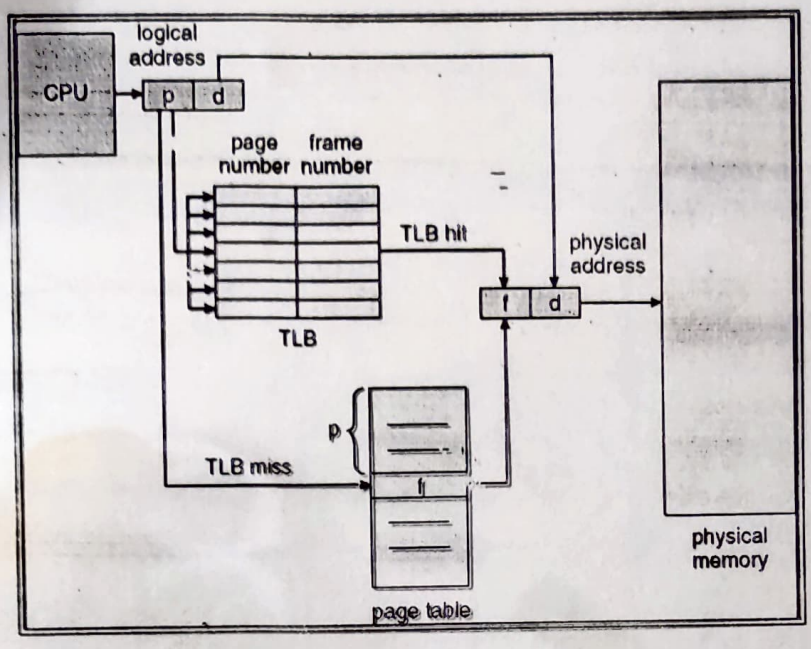
The TLB is used with page tables in the following way. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found its frame number is immediately available and is used to access memory. If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. If the TLB is already full of entries, the operating system must select one for the replacement.

Paging Hardware with TLB :- PTO

Q Memory takes 200 ns to access a memory & 100 ns to access the TLB, calculate effective access time if the hit ratio is 80%.

(i) If the hit ratio is being increased from 80% to 90%.

(ii) If the hit ratio is being decreased from 80% to 70%.



The percentage of times that a particular page number is found in the TLB is called the **hit ration.** An 80% hit ratio means that we find the desired page number in the TLB 80% of the time. If it takes 20 nanoseconds to search the TLB, and 100 nanoseconds to access memory, then a mapped memory access takes 120 nanoseconds when the page number is in the TLB. If we fail to find the page number in the TLB(20 nanoseconds), then we must first access the desired byte in memory (100 nanoseconds), for a total of 220. To find the effective memory access time, we must weigh each case by its probability:

Effective access time=0.80 * 120 + 0.20 * 220

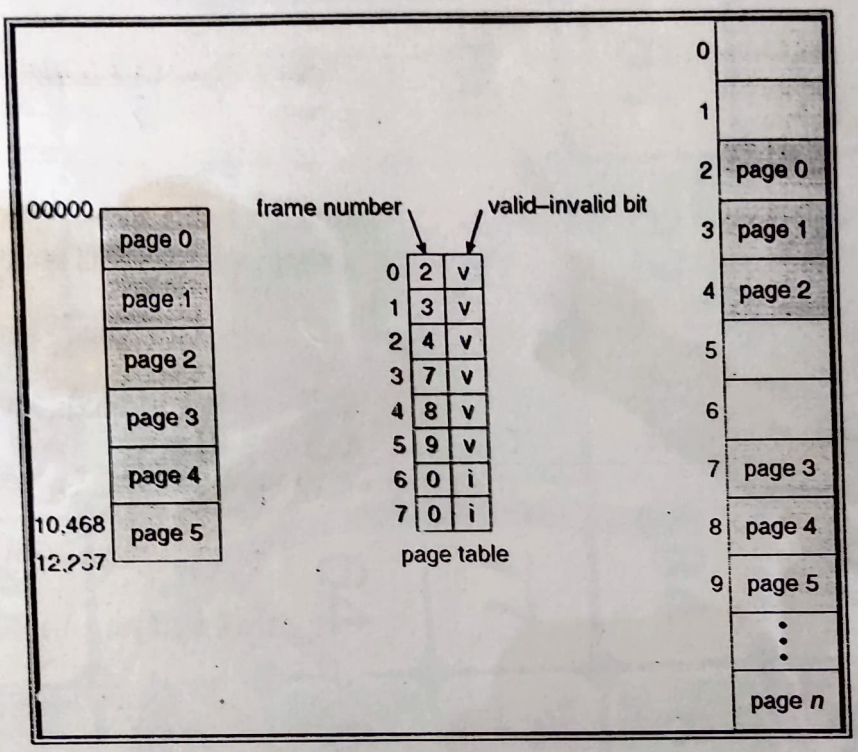
= 140 nanoseconds.

Memory Protection

- Memory protection implemented by associating protection bit with each frame. Normally, these bits are kept in the page table. One bit can define a page to be read/write or read only.
- Valid-invalid bit attached to each entry in the page table:

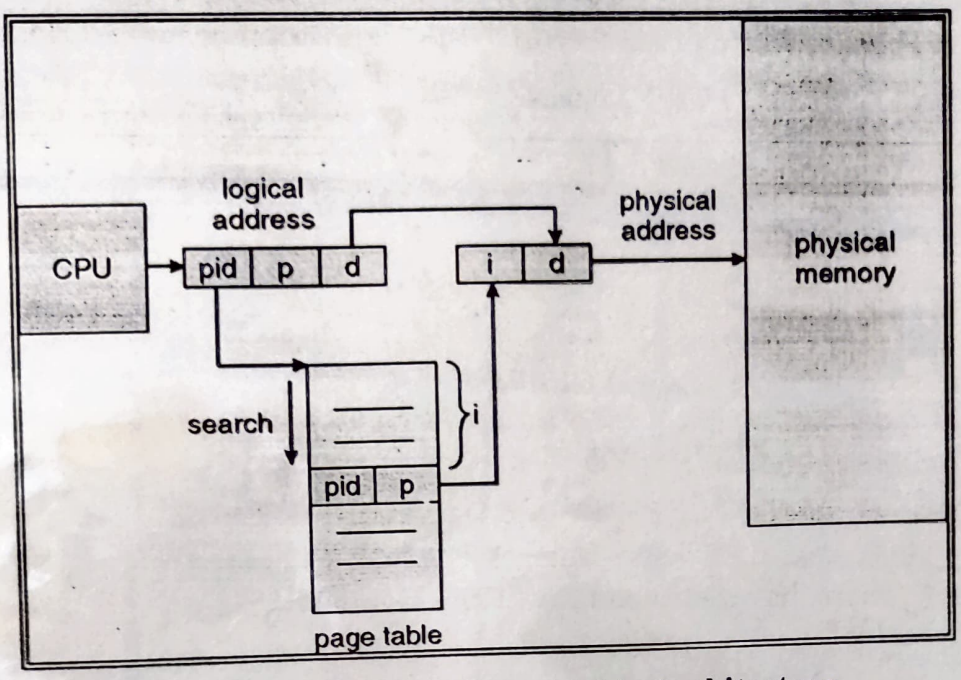
"valid" indicates that the associated page is in the process' logical address space, and is thus a legal page & "invalid" indicates that the page is not in the process' logical address space

Valid (v) or Invalid (i) Bit In A Page Table



Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries



Inverted Page Table Architecture

Each inverted page-table entry is a pair <process-id, page-number> where the process-id assumes the role of the address-space identifier. When a memory reference occurs, part of the virtual address, consisting of <process-id, page-number>, is presented to the memory subsystem. The inverted page table is then searched for a match. If a match is found-say, at entry i-then the physical address <i,offset> is generated. If no match is found, then an illegal address access has been attempted.

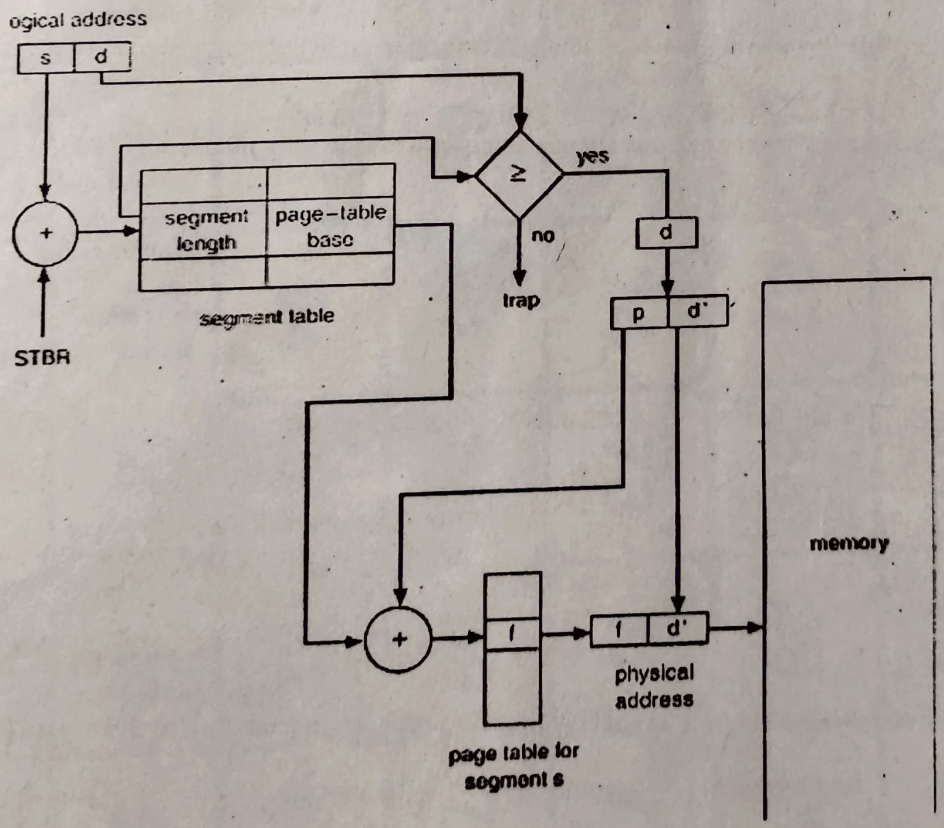
Segmentation with paging:- Both paging and segmentation have advantages and disadvantages. The Intel 80X86 and Pentium family is base on segmentation. Combination of both is the best architecture of the Intel 80X86. The maximum number of segments per process is 16 KB, and each segment can be as large as 4 gigabytes. The page size is 4KB.

Segmentation with Paging – MULTICS

The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.

Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a page table for this segment.

MULTICS Address Translation Scheme



Virtual Memory

1. **Virtual memory** – separation of user logical memory from physical memory.
   a. Only part of the program needs to be in memory for execution some can be on disk
   b. Disk address can be stored in place of frame number
2. A valid–invalid bit is associated with each page table entry (1 ⇒ in-memory, 0 ⇒ not-in-memory)
   a. During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault ⇒ bring in to memory

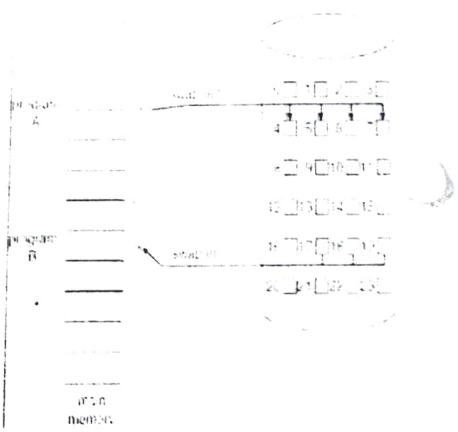
Virtual Memory is being implemented by two techniques:-

a) Demand Paging b) Demand Segmentation

Demand Paging :-

When a process needs to be executed then we take that process from secondary memory and we put it into memory. Normally, a disk is used to represent secondary memory. Pages are only loaded into memory when they are demanded during execution

- Less I/O needed
- Less memory needed
- Higher degree of multiprogramming
- Faster response



- Pager (lazy swapper) never swaps a page into memory unless that page will be needed.
- An extreme case: Pure demand paging starts a process with no pages in memory …

Demand Segmentation :-

- Same idea as demand paging applied to segments.
- If a segment is loaded, base and limit are stored in the STE and the valid bit is set in the PTE.
- The PTE is accessed for each memory reference (not really, TLB).
- If the segment is not loaded, the valid bit is unset. The base and limit as well as the disk address of the segment is stored in the an OS table.
- A reference to a non-loaded segment generate a segment fault (analogous to page fault).
- To load a segment, we must solve both the placement question and the replacement question (for demand paging, there is no placement question).
- I believe demand segmentation was once implemented by Burroughs, but am not sure. It is not used in modern systems.

Page Faults:- whenever we are trying to access that page that have not being loaded in memory or you are trying to access a page that is invalid. An invalid page fault or page fault error occurs when the operating system cannot find the data in virtual memory. This usually happens when the virtual memory area, or the table that maps virtual addresses to real addresses, becomes corrupt.

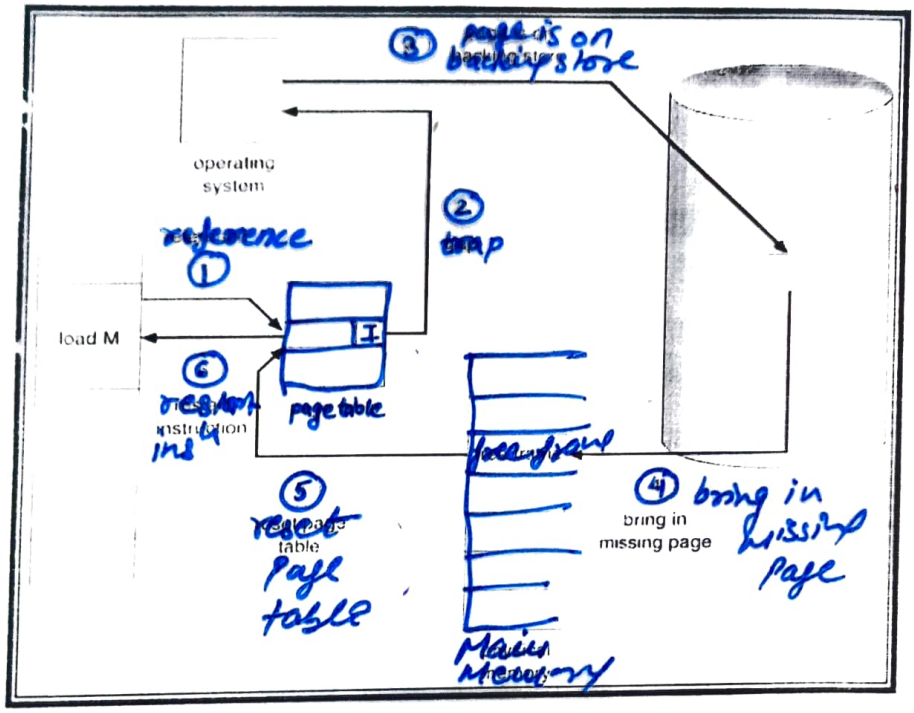
Steps in Handling a Page Fault

1. If there is ever a reference to a page, first reference will trap to

OS -> page fault

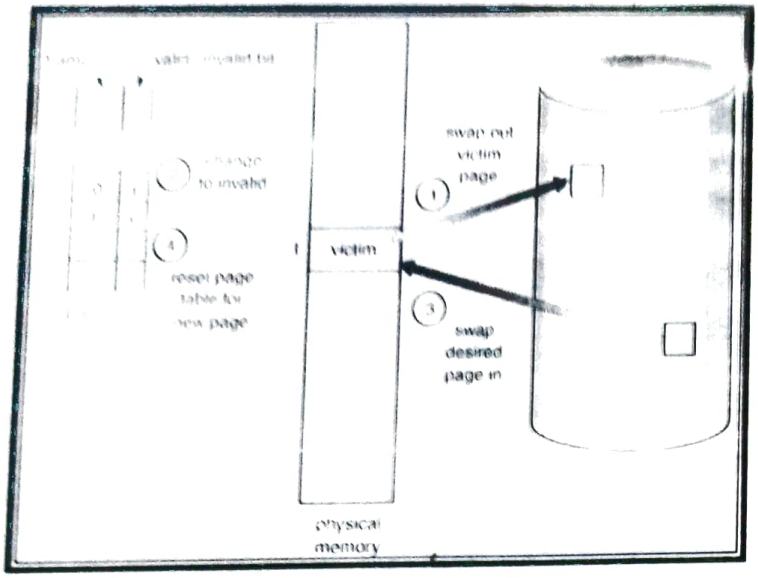
2. OS looks at another table to decide:
3. Invalid reference -> abort.

4. Just not in memory.
5. Get empty frame.
6. Swap page into frame.
7. Reset tables, validation bit = 1.
8. Restart instruction: Least Recently Used
9. block move
10. auto increment/decrement location



What happens if there is no free frame?

1. Page replacement – find some page in memory, but not really in use, swap it out.
2. Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.



Basic Page Replacement

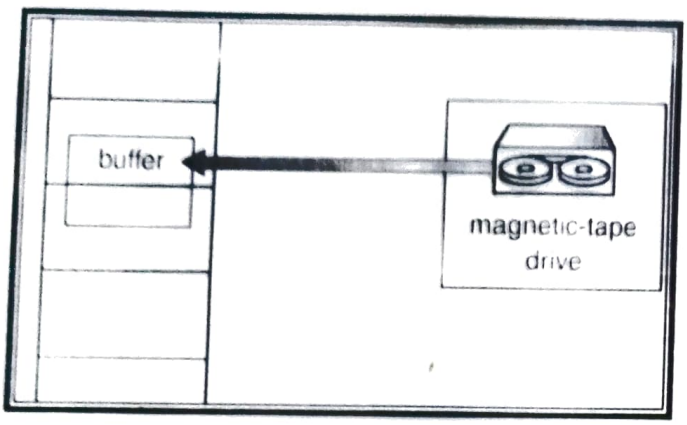
1. Find the location of the desired page on disk.
2. Find a free frame:
    1. If there is a free frame, use it.
    2. If there is no free frame, use a page replacement algorithm to select a *victim* frame.
    3. If victim is dirty, write it out to disk
3. Read the desired page into the (newly) free frame. Update the page and frame tables.
4. Restart the process

Page Replacement Policies

1. Global vs. Local Allocation
    a. Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another - only for crucial OS processes
    b. Local replacement – each process selects from only its own set of allocated frames - the normal case

2. I/O interlock

   a. Pages that are used for copying a file from a device must be locked



Thrashing

1. VM works because of locality

   a. Process migrates from one locality to another.

   b. Localities may overlap

2. What happens if

   Σ size of localities > total memory size

3. If a process does not have "enough" pages for its locality, the page-fault rate is very high.

   a. Thrashing ≡ a process is busy swapping pages in and out.

4. Thrashing may lead to:

   a. low CPU utilization.

   b. operating system thinks that it needs to increase the degree of multiprogramming.

   c. another process added to the system

5. It is important to allocate enough frames to each process

   a. If that's not possible, the process must be swapped out

Page Replacement Algorithms

Optimal Algorithm:- Replace page that will not be used for longest period of time in future.

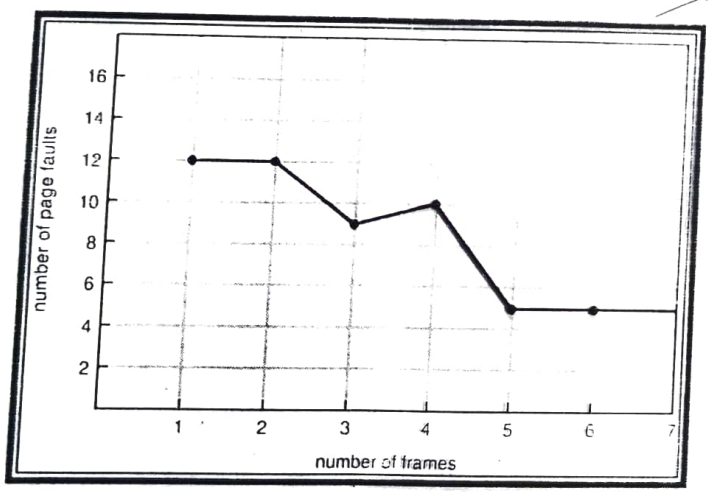
3 frames example: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 (9)

First-In-First-Out (FIFO) Algorithm

3 frames example: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 (15)

FIFO Illustrating Belady's Anamoly

1) 4 frames example: 1 2 3 4 1 2 5 1 2 3 4 5 (10)
2) 3 frames example: 1 2 3 4 1 2 5 1 2 3 4 5 (9)
   a) Belady's Anomaly - more frames does not imply less page faults



Least Recently Used (LRU) Algorithm

1) 3 frames example: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 (11)
2) Counter implementation
   a) Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
   b) When a page needs to be changed, look at the counters to determine which are to change.
3) Stack implementation – keep a stack of page numbers in a double link form: