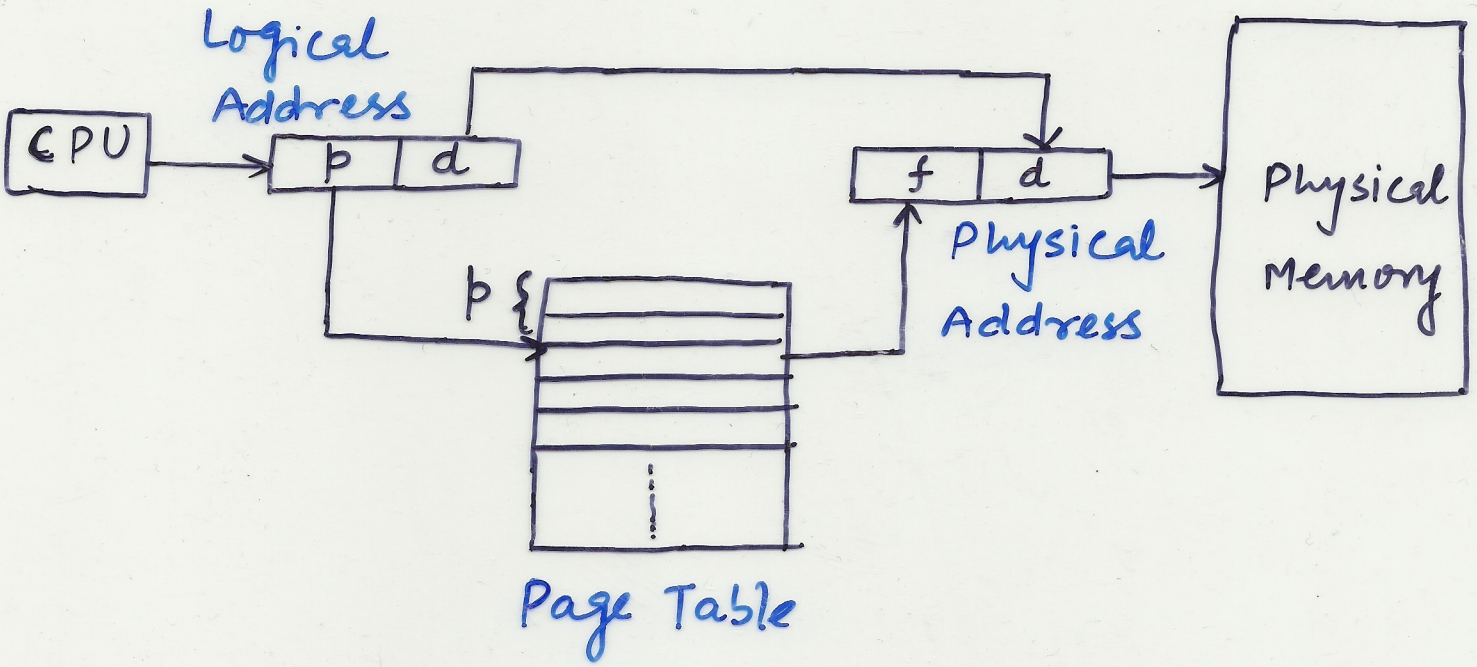
Hardware Support



→ Every address generated by CPU is divided into two parts :

- Page Number (p)
- Page offset (d)

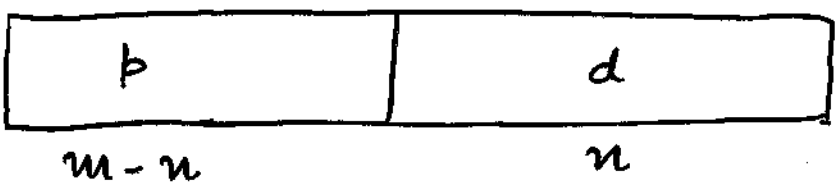
→ Page table uses page number as its index.

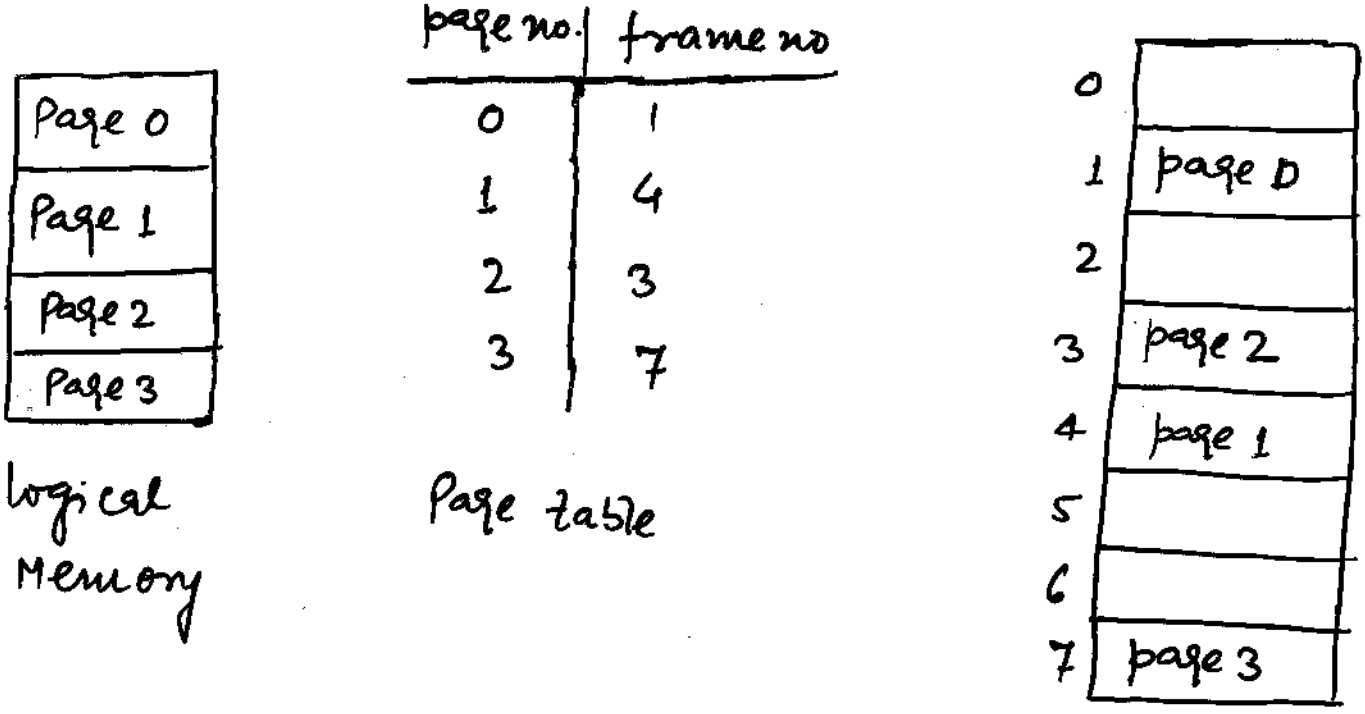
→ Page table has base address of each page in physical memory.

→ Base address is combined with page offset to define the physical memory address.

→ Size of a page is typically a power of 2.

→ If the size of logical address space is 2^m, and a page size is 2^n bytes (Assuming 1 byte = 1 addressing Unit) Higher order $m-n$ bits of logical address shows page no while lower n bits show page offset.





Logical Memory

Page table

Physical Memory

Paging Model of Logical & Physical Mem.

→ Paging completely removes external fragmentation.

→ Internal fragmentation is still a possibility.

Example:

Physical Memory = 32 bytes

Page Size = 4 bytes.

No. of pages = $32/4 = 8$

Page no.	Frame No
0	5
1	6
2	1
3	2

Page table

- logical address 0

  has page 0 & offset 0.

  So corresponding physical address = $5 \times 4 + 0 = 20$

- logical address 3

  has page 0 & offset 3.

  ∴ Physical address = $5 \times 4 + 3 = 23$.

- logical address 13

  has page no 3. offset 1.

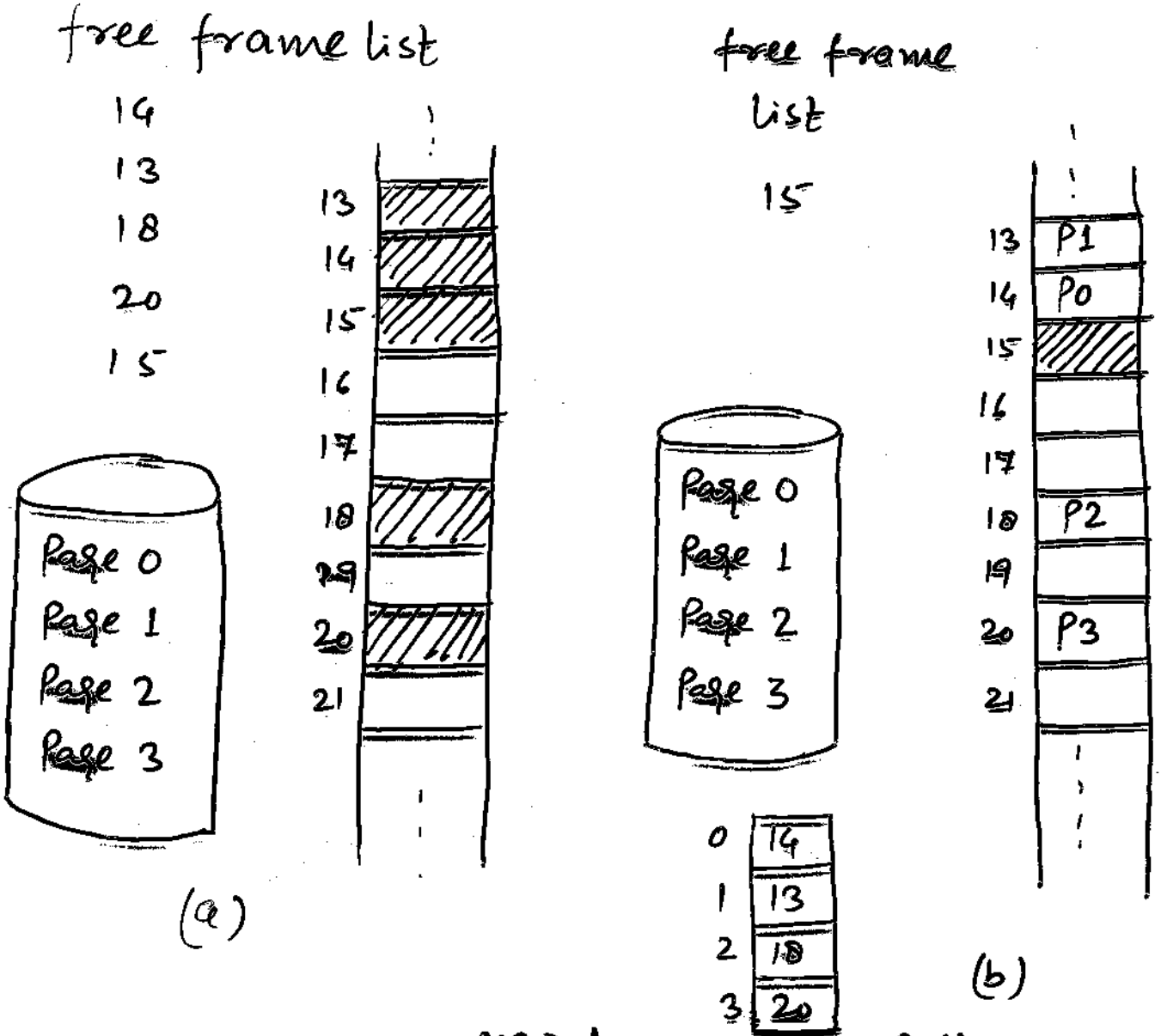
  ∴ Physical address = $2 \times 4 + 1 = 9$

- In paging, there is no external fragmentation
- Any free frame can be allocated to a process that needs it.

→ Internal fragmentation can not be ignored.

Free Frames:

→ When a process arrives, its size, expressed in pages is examined.



Frame Table:

→ A data structure, used to keep track of frames

→ Has one entry for each frame, indicating whether it is free or allocated.

→ If allocated, to which user process/ processes.

Implementation of page table

→ Page table is kept in main memory.

→ Page Table Base Register (PTBR) points to the page table.

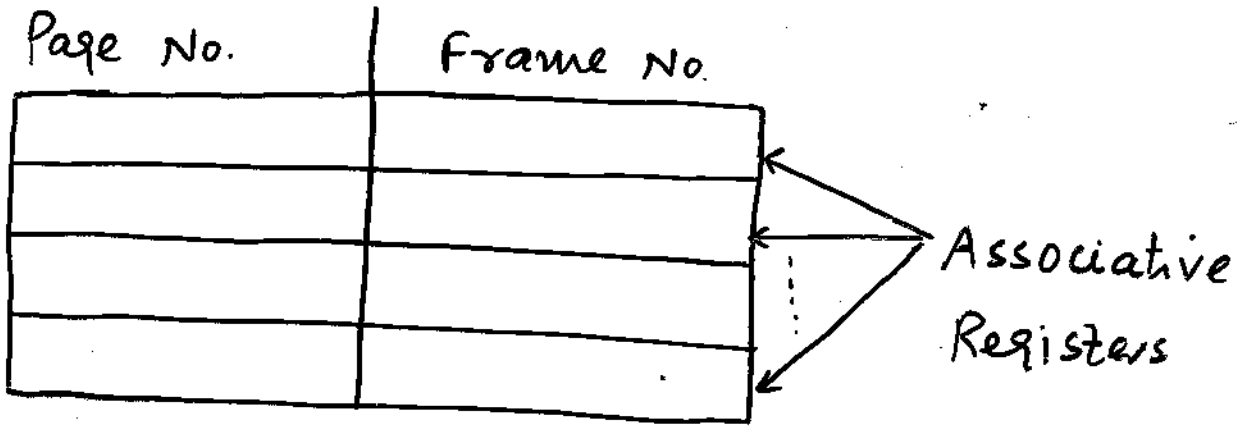
→ Page Table length Register (PTLR) indicates the size of the page table.

→ Data/ Instruction access requires two memory accesses. One for page table & one for data/ instruction.

→ This problem is solved by using special fast hardware cache - Associative Memory / Translation look-aside buffers (TLBs).

Associative Memory

→ Brings parallel search.



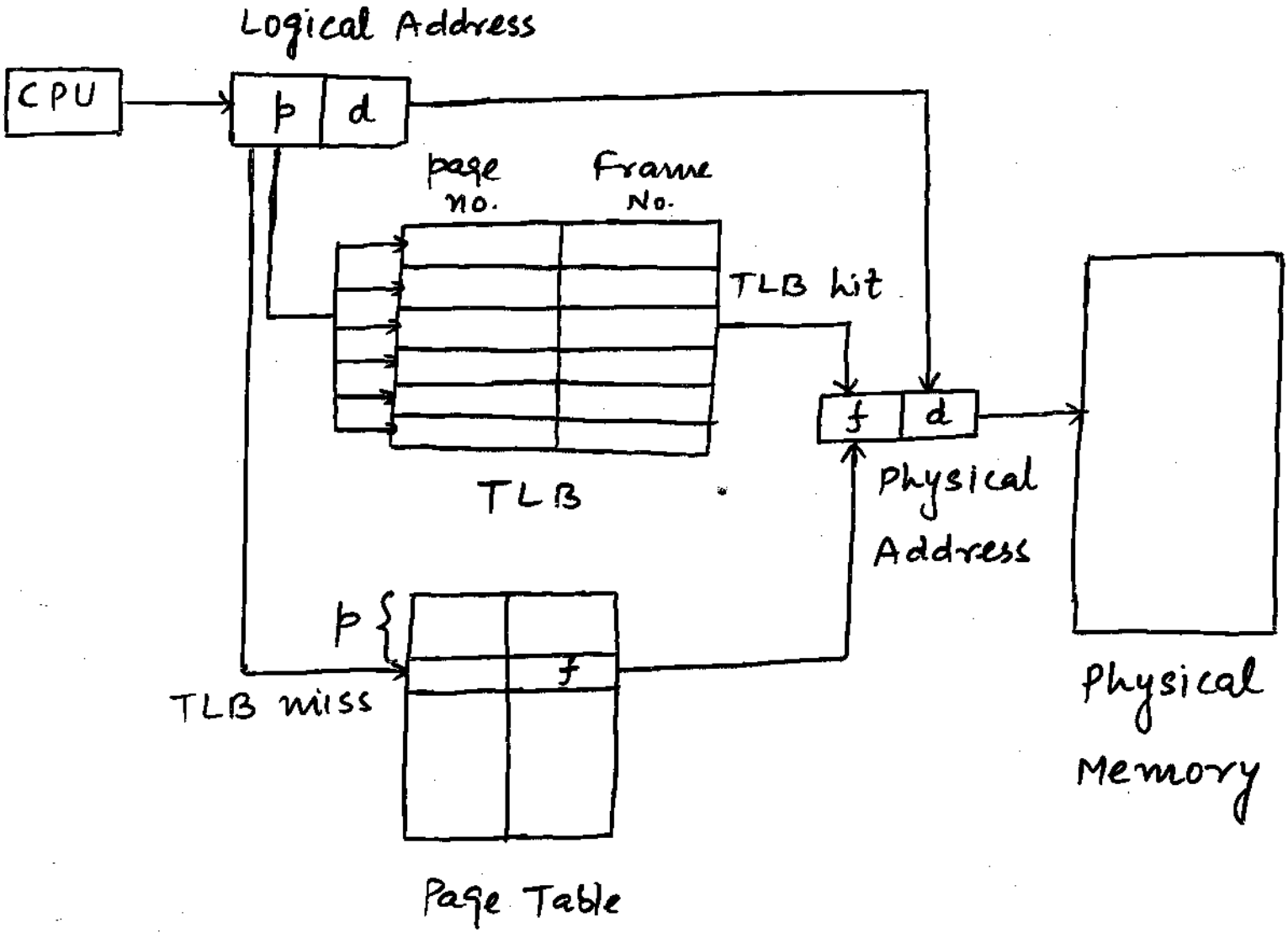
→ Address Translation (p, d):

- If p is in associative register, get frame no. out.
- Otherwise, get frame no from page table in main memory.

→ Each time a new page table is selected, TLB is flushed.

→ Otherwise there will be old entries in TLB that contain valid virtual address but incorrect physical address.

Paging Hardware with TLB



Effective Access Time

Hit Ratio: Percentage of times that a page no. is found in the associative memory/TLB.

ε → Associative look up time

Let memory cycle time is 1 millisecond.

Hit ratio is α.

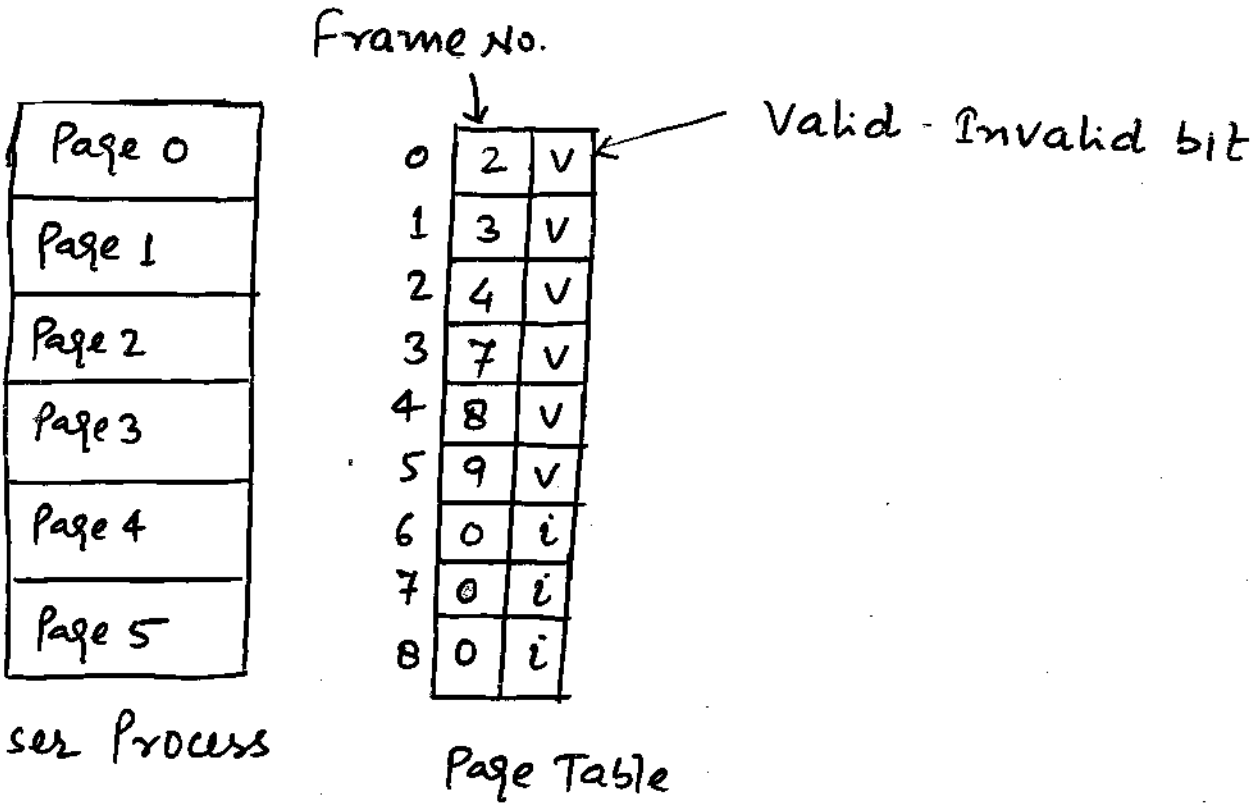
- Effective Access Time =

$$(1+\varepsilon)\alpha + (2+\varepsilon)(1-\alpha)$$

$$\boxed{EAT = 2+\varepsilon-\alpha}$$

Memory Protection in Paging

→ Memory protection is implemented by associat-ing a protection bit with each page.



User Process

Page Table

- Valid-Invalid bit attached to each entry in the page table :
  - Valid indicates that associated page is in the proc-ess logical address space & is thus a legal page.
  - Invalid indicates that page is not in the process legal address space.

Page Table structure

- Hierarchical Paging
- Inverted Page Table

→ For systems with large logical address space, page table size can be excessively large.

→ Problem can be solved by breaking logical address space into multiple page tables.

Hierarchical Page Table

→ Breaks up the logical address space into multiple page tables.

→ A simple technique is two level page table.

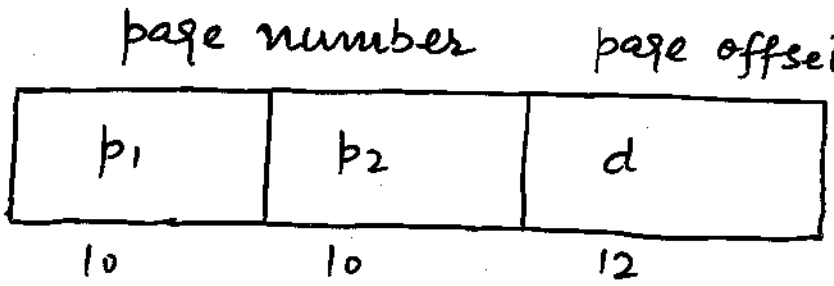
→ A logical address (on 32 bit machine with 4K page size) is divided into:

- a page number consisting of 20 bits.
- a page offset " " 12 " .

→ Page table is further paged, so 20 bit page no. is further divided into:

- 10 bit page no.
- 10 bit page offset.

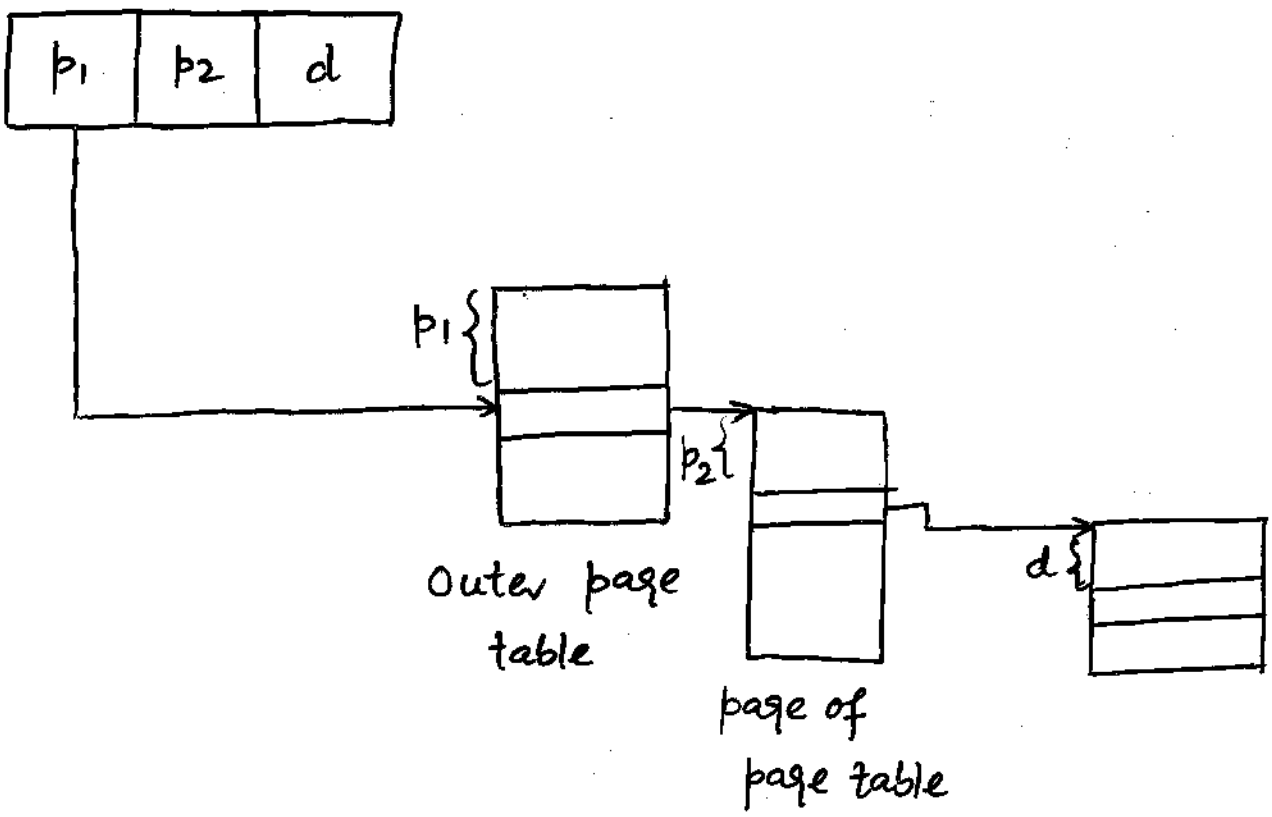
→ Thus, a logical address is as follows:



p_1 : index into outer page table

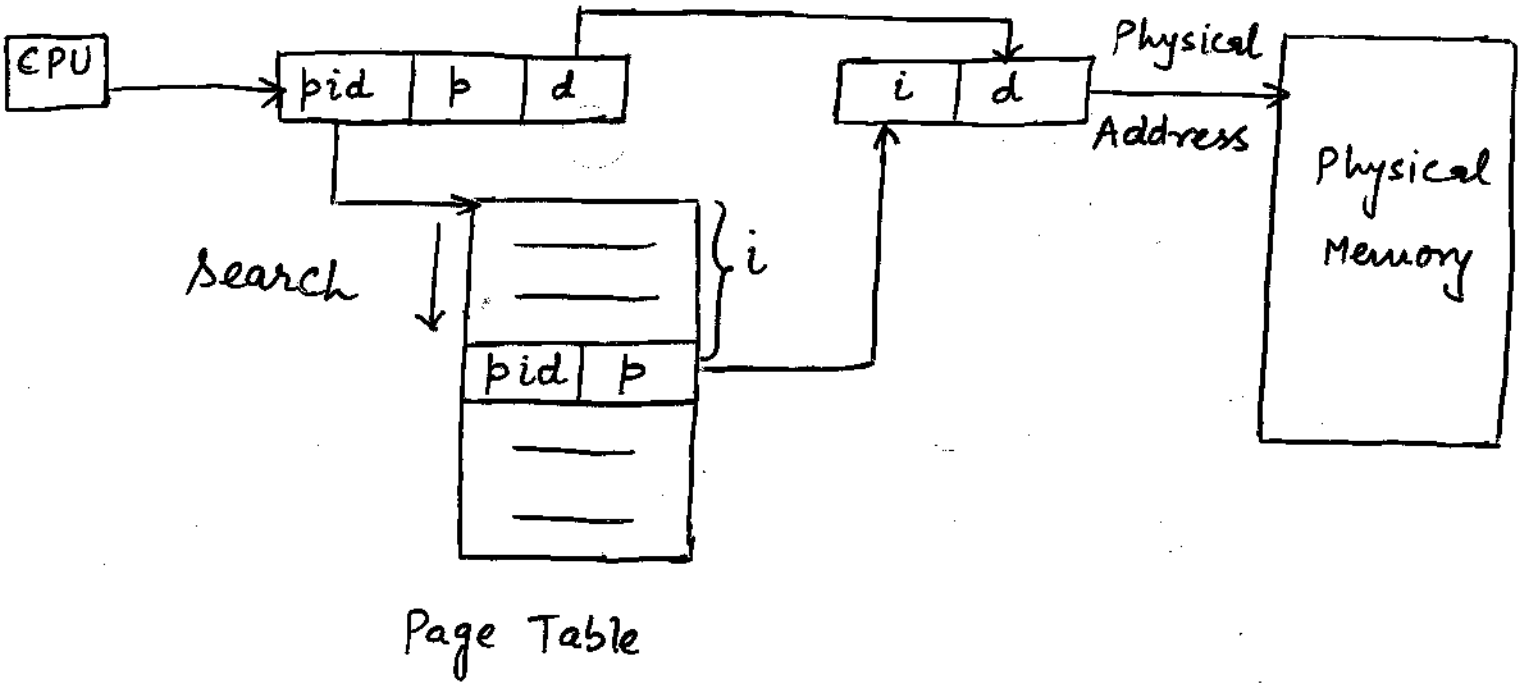
p_2 : displacement within the page of outer page table

→ Address translation scheme for a 2-level 32 bit paging Architecture is:



Inverted Page Table

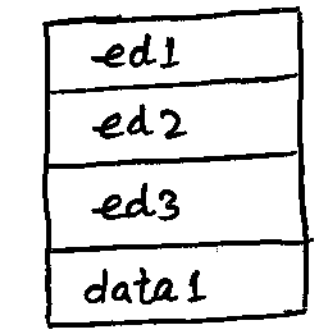
- One entry for each real page (frame) of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Decreases memory needed to store each page table but increases time needed to search the table when a page reference occurs.



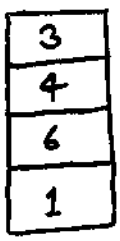
Page Table

Shared Pages:

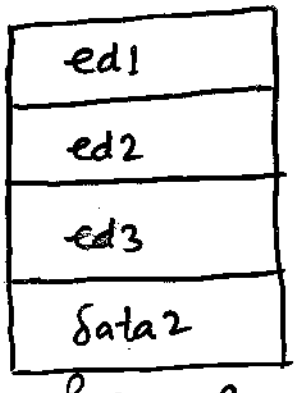
- Supports possibility of sharing common code.
- One copy of shared code (reentrant code) is shared among processes.
- Text editors, compilers etc are shared code.
- Shared code must appear in same location in the logical address space of all processes.
- Each process keeps a separate copy of the code & data.
- The pages for private code & data can appear anywhere in the logical address space.



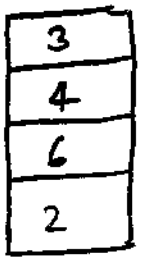
Process P_1



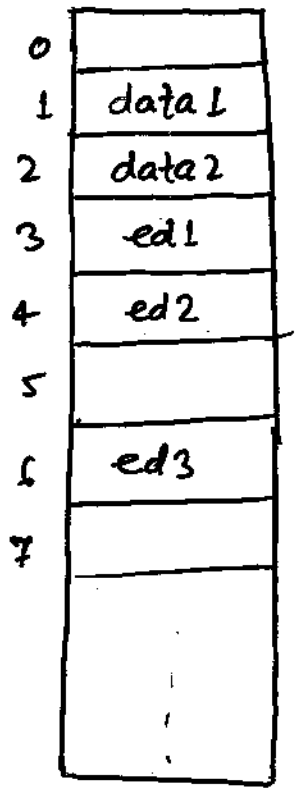
Page table for P_1



Process P_2



Page table for P_2



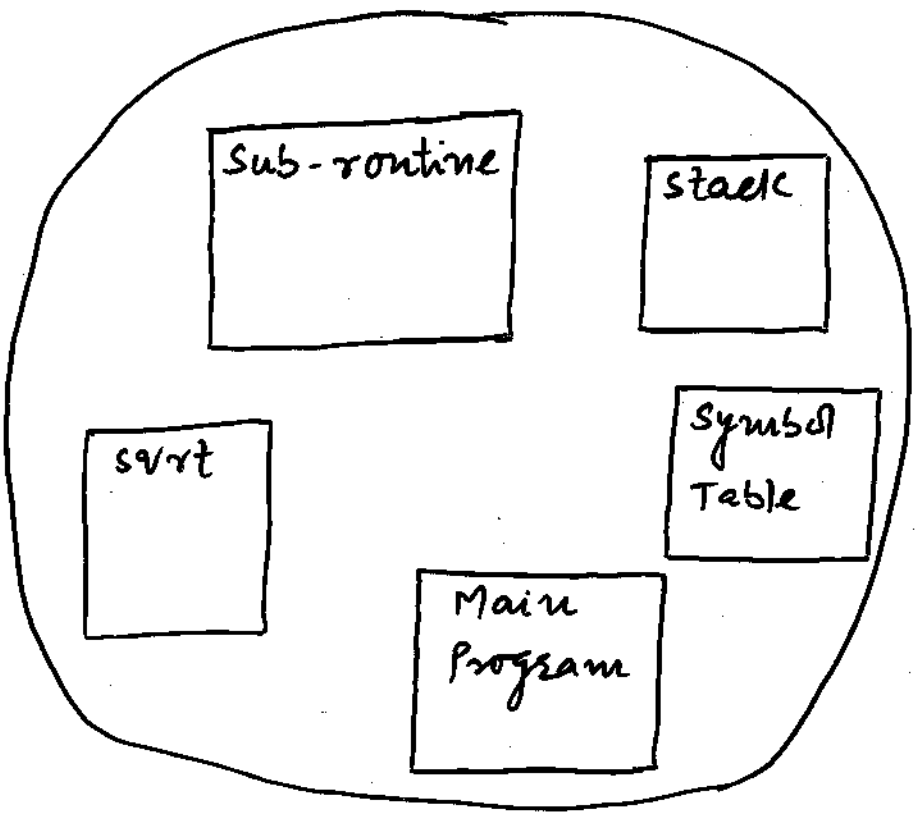
Physical Memory.

Segmentation

→ A memory management scheme that supports user view of memory.

→ A program is a collection of segments:

* main program
* function
* local variables/ global variables
* symbol table.



Logical Address space.

→ Logical Address space is a collection of segments.

→ Unlike pages, segments are variable length chunks in memory.

Segment Hardware :

- Logical address consists of two tuples:

  < segment-number, offset >

- Physical address is one dimensional sequence of bytes.
- Mapping between Logical address & physical address is done by segment table.
- Each segment table entry has :

  Base - Contains the starting physical address where the segments reside in memory.

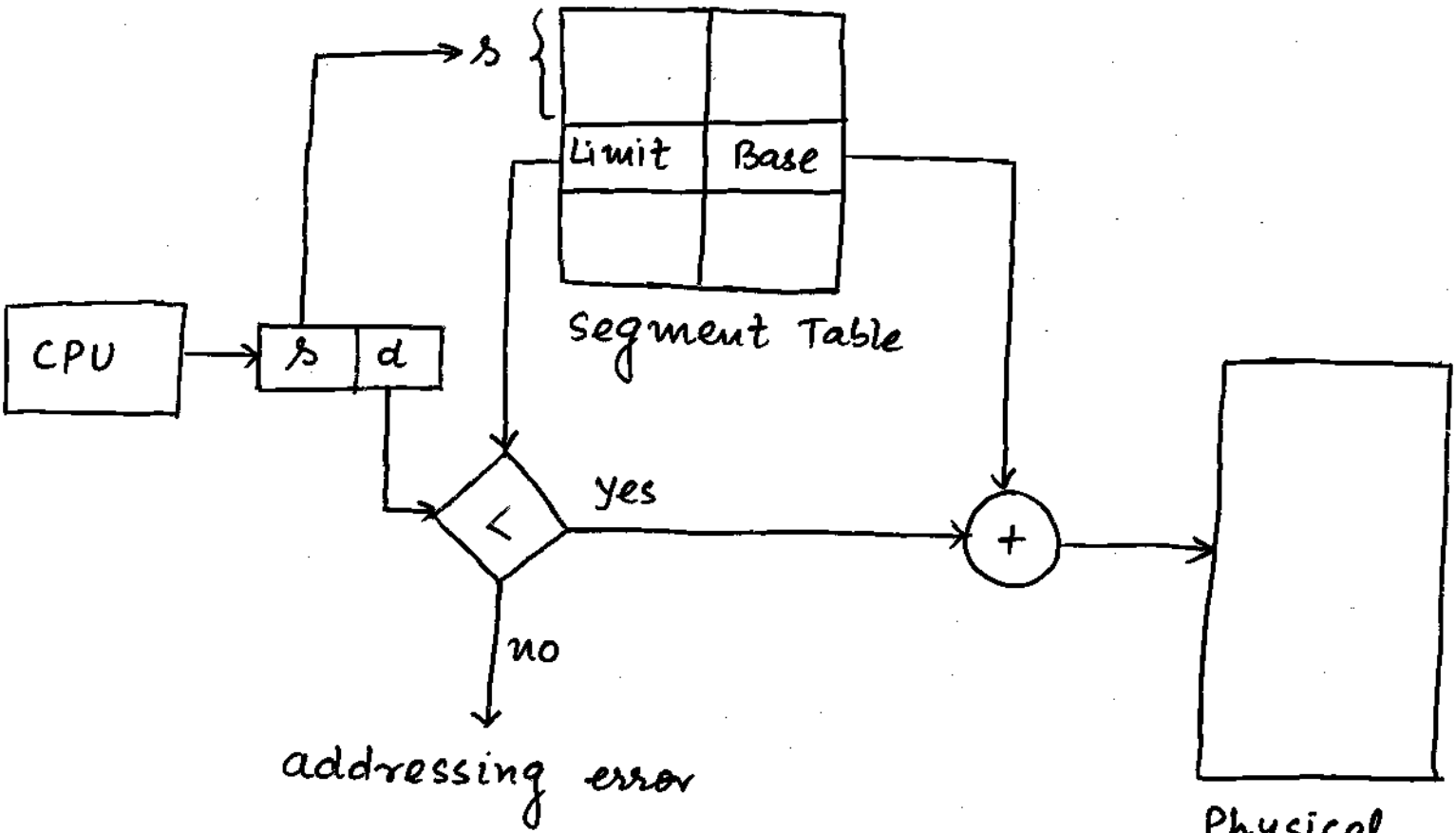
  Limit - Specifies the length of the segment.

- Segment Table Base Register (STBR) -

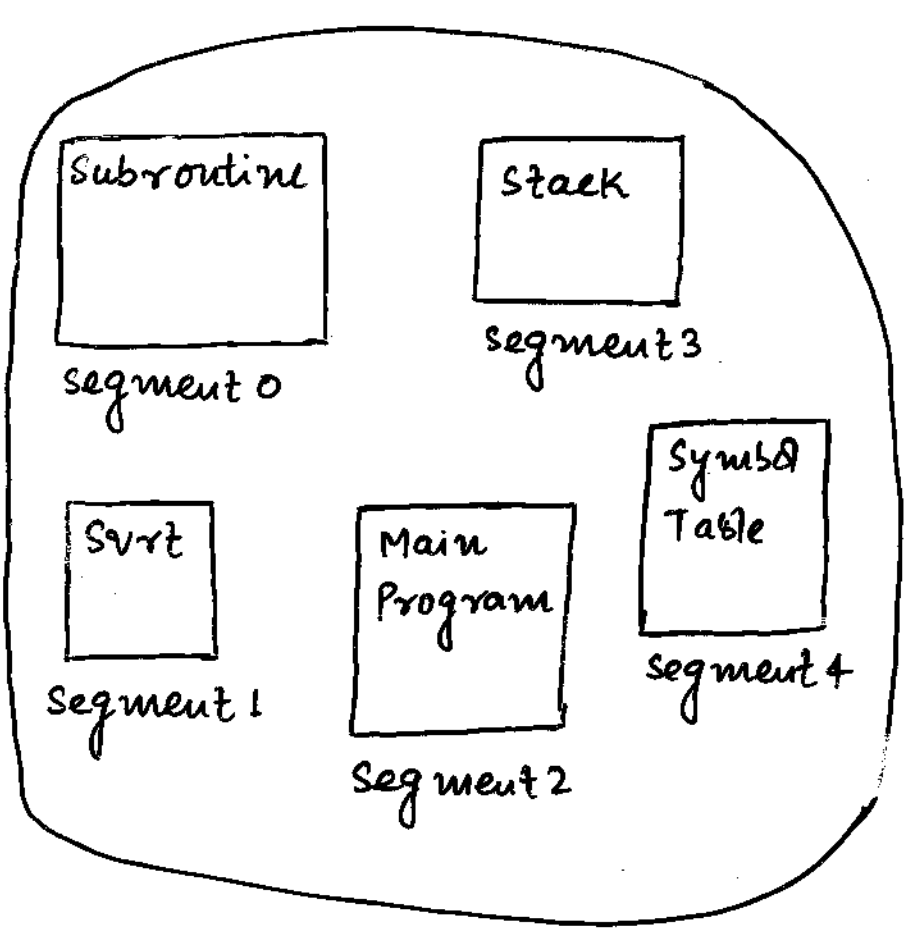
  points to the segment table's location in memory.

- Segment Table Length Register (STLR)

  indicates no. of segments used by a program.



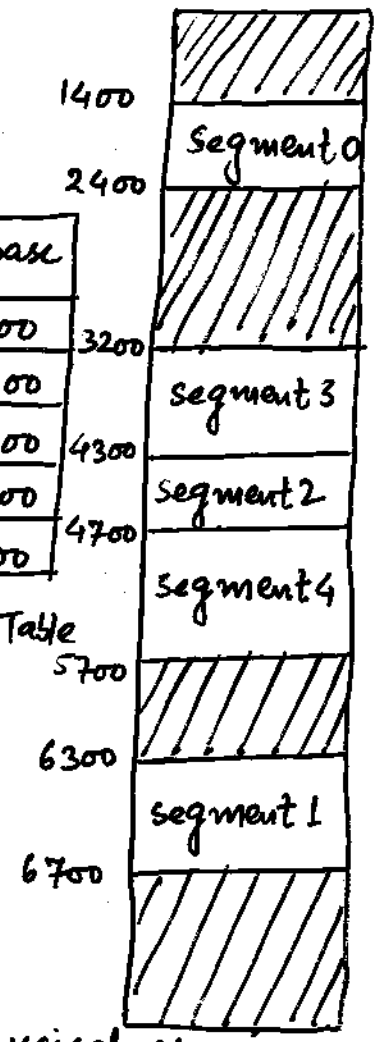
Example of Segmentation



Logical Address space

	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment Table

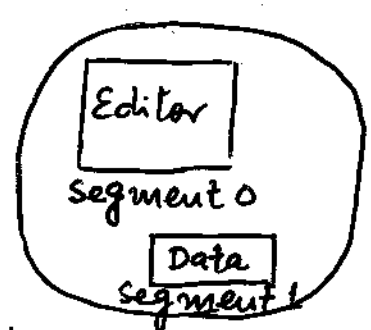


Physical Memory

- As with paging, this mapping requires two memory references.
- Effectively slows down the system by a factor of 2.
- Concept of associative memory can be used to increase the effective Access Time.

Protection & sharing:

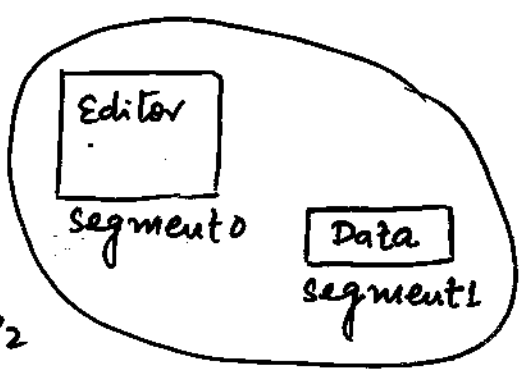
- Segments support protection, by associating :
  - * Validation bit = 0 ⇒ Illegal segment.
  - * Read / write / Execute previleges.
- Segments can be shared by a no. of user proce-sses.



Logical address space for P_1

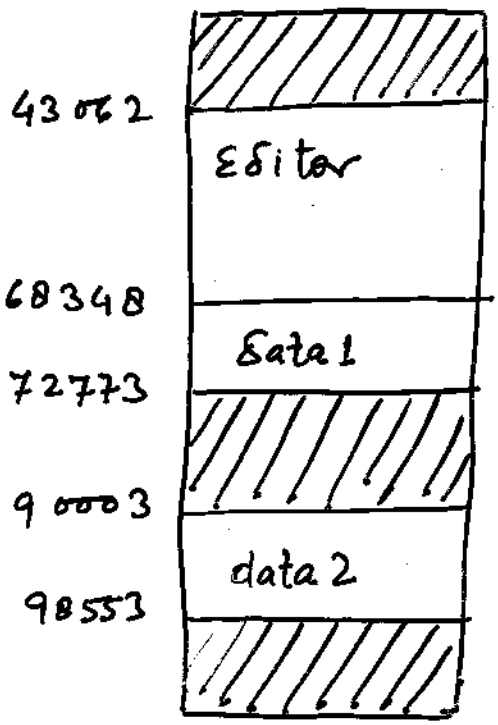
	limit	base
0	25286	43062
1	4425	68348

segment table for P_1



	limit	base
0	25286	43062
1	8850	90003

segment table for P_2



Physical Memory

Fragmentation:

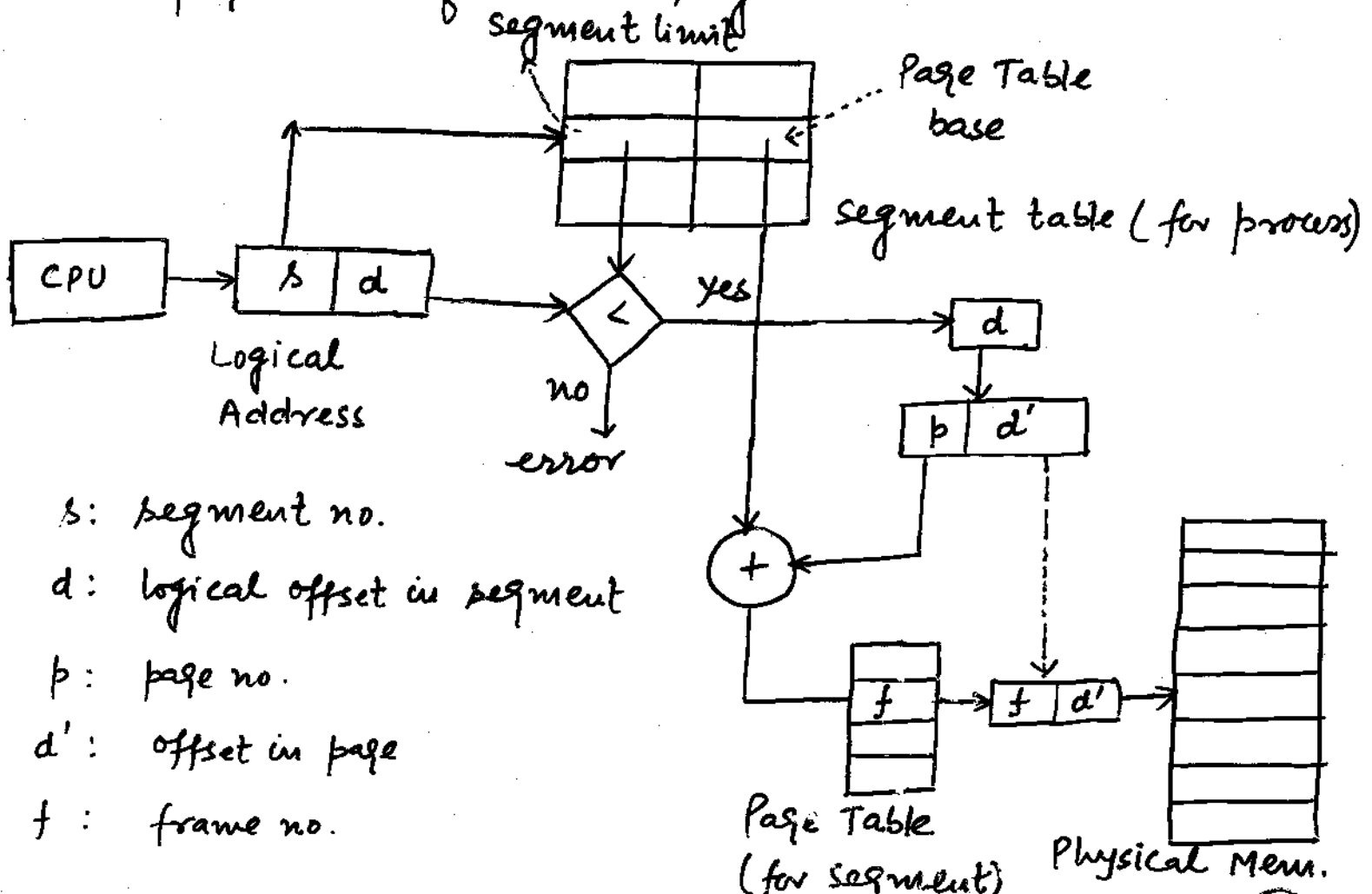
→ Segmentation has variable length segments

→ External fragmentation is possible.

Segmentation with Paging

→ Problem of external fragmentation is solved.

→ Differs from pure segmentation in that the segment table entry contains not the base address of the segment but rather the base address of the page table for this segment.



s: segment no.

d: logical offset in segment

p: page no.

d': offset in page

f: frame no.

Virtual Memory

→ Allows execution of processes that may not be completely in memory.

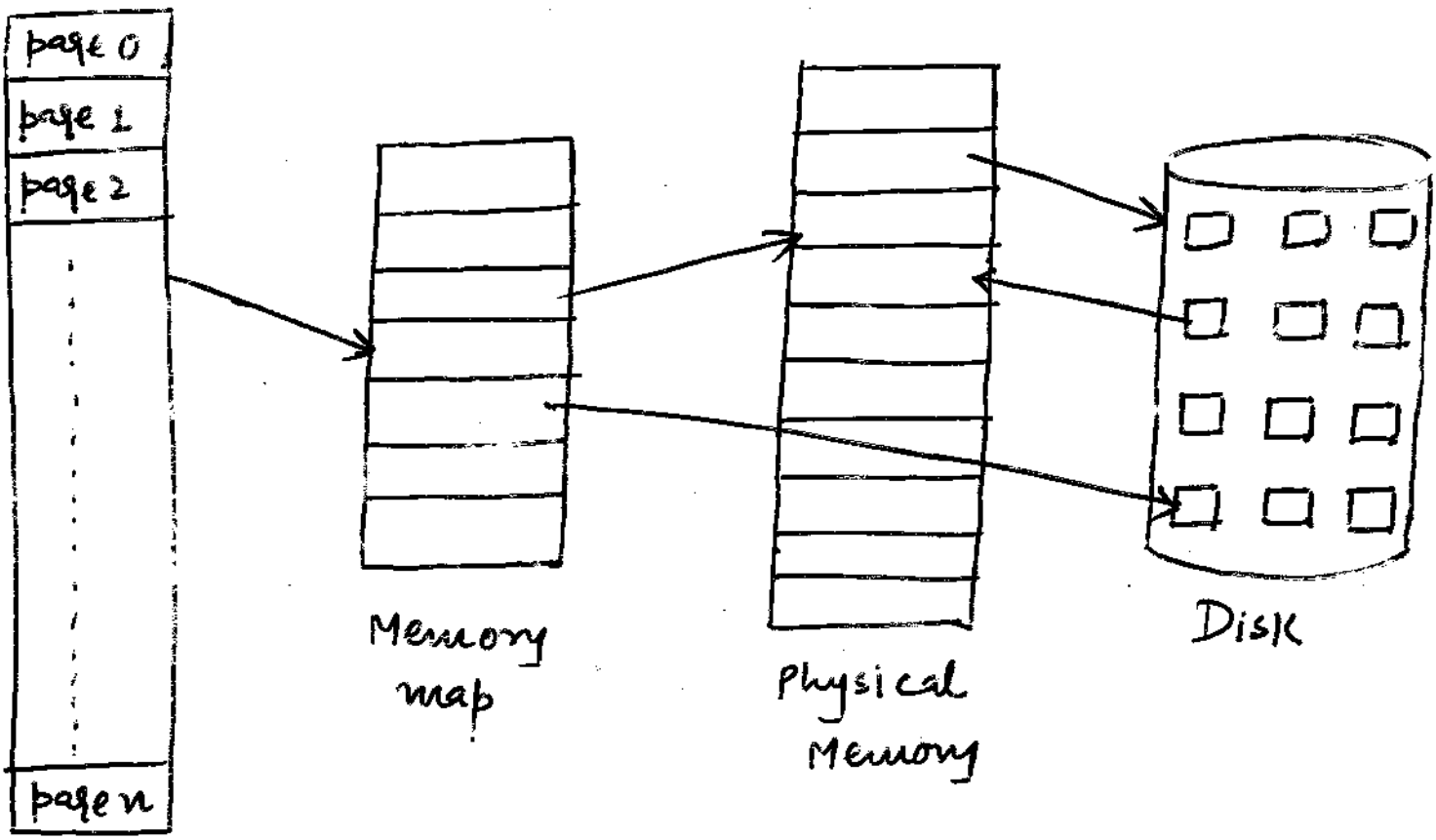
→ Separation of logical memory from physical memory.

→ Only part of the program needs to be in memory for execution.

→ A program therefore, can be larger than the physical memory.

→ Can be implemented via:

- Demand Paging
- Demand segmentation.



Virtual Memory

Demand Paging

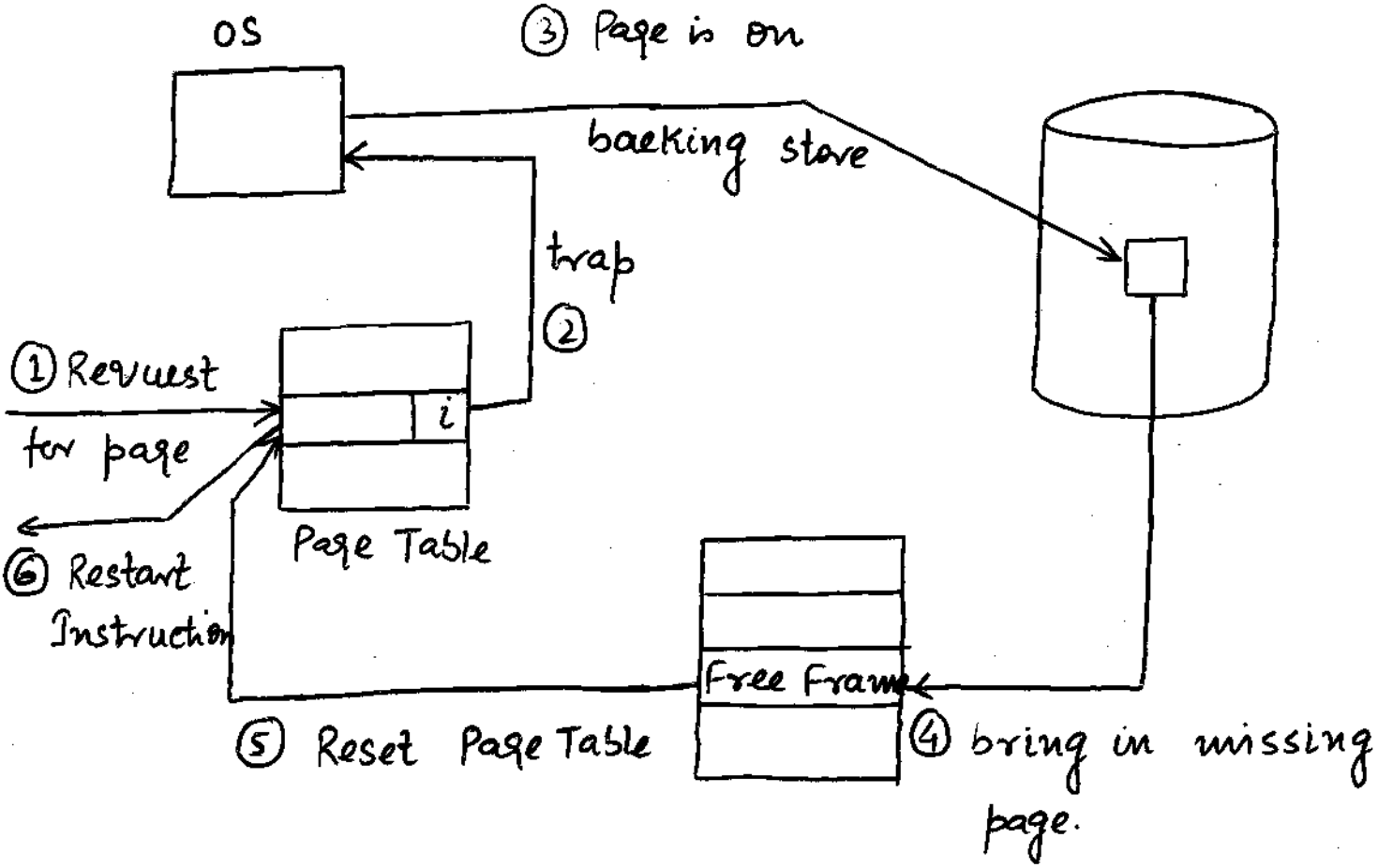
- Brings a page into memory only when it is needed.
  - * Less I/O needed.
  - * Less Memory needed.
  - * Faster Response.
  - * More users.
- Mechanism to distinguish between pages which are in main memory & which are on disk is required.
- This is done by using Valid-Invalid bit with each page, in the page table.

page #	frame #	Valid Invalid bit
		1
		1
		0
		1

Page Table

- When this bit is 1 – page is valid & present in main memory.

- When valid-Invalid bit is 0 – page is either not valid or not in main memory.
- Access to a page marked invalid causes page fault.



①: User process requests a page

②: page found invalid in page table. A trap is generated to OS.

③: Page is found on the baeking store.

④: Paged is swapped into a free frame.

⑤: Page table is updated.

⑥: Process resumes execution from that instruction which lead page fault.

Pure Demand Paging:

- start executing a process with no pages in memory
- when 1st instruction is executed, 1st page fault occurs.
- Page faults keeps on occurring, until required pages are brought into memory.
- Moto is never bring a page into memory until it is required.

Performance of Demand Paging:

→ Measured in terms of Effective Access Time of a page.

→ p : probability of a page fault ($0 < p < 1$)

if $p = 0$, no page fault

if $p = 1$, every reference is a fault.

Effective access time = $(1-p)$ * memory access time
+ p * page fault time.

→ Major components of page fault service time are:

- Service page fault interrupt.
- Read out / swap out a page (if needed).
- Swap in the required page.
- Restart the process.

Example:

Average page fault service time = 25 milliseconds.

Memory access time = 100 microseconds.

$$EAT = (1 - p) * 100 + p \times 25000$$

$$\boxed{EAT = 100 + 24,900p}$$

If we want less than 10% of degradation, then

$$100 + 24900p < 110$$

$$24900p < 10$$

$$\approx p < .00040$$

that is, percentage of page fault rate should be less than 0.04.