Memory - consists large array of words or bytes, each having its own unique address.

Program generates sequence of memory address during its execution.

⇒ CPU only access data from memory or registers so if data is not present in these direct access storage devices than data should moved there befor CPU access.

⇒ Memory is slow than register access so to speed up a fast memory is is added between memory & CPU called cache.

⇒ (OS) we have to protect user process to access each other address space.
OS have two registers for each process -
① base register - Smallest physical address
② limit register - Specifies size of range

base address ≤ CPU generated address < base + limit

eg 30004 (base) & limit is 12030 means address can range from 30004 to 42034.

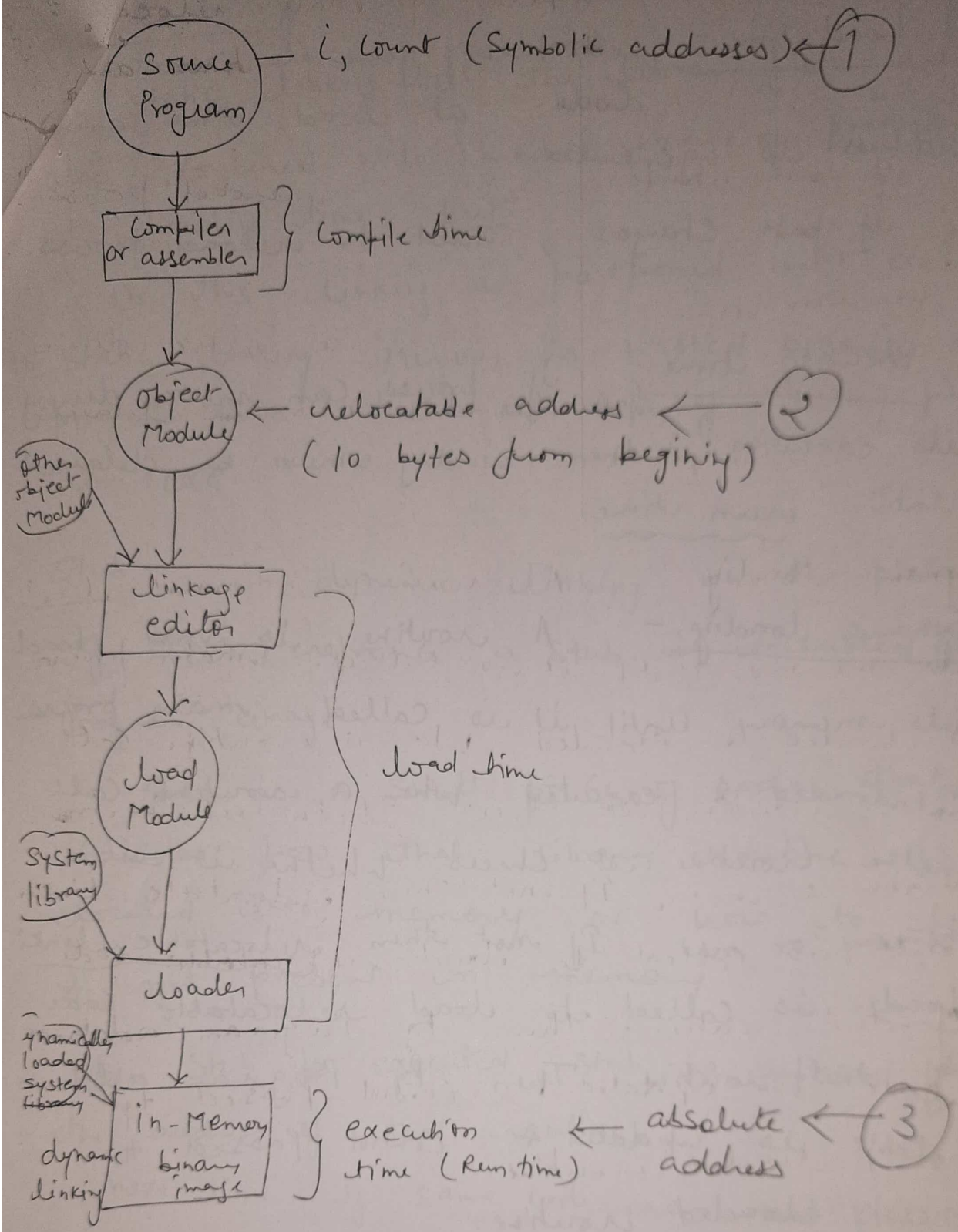
NOTE - Only OS can load access & change the value of these two registers

Address Binding –

disk → Program as binary executable file.
(Name.exe)

Input queue / Job queue – waiting processes on the disk.
they are loaded into main memory & form ready queue.

⇒ Addresses may be represented in different ways during coding to execution.



① Compile time address binding – if you know exact base address at compile time then absolute code can be generated.

if base changes, recompilation necessary

(2) Load time - Compiler generate relocatable Code. at load time absolute address is generated.

∴ if base changes, need to reload process.

(3) Execution time - if process can move during its execution, then binding must be delayed until run time.

Dynamic loading - A routine is not loaded into memory until it is called. main program is loaded & executed when a routine call another routine it checks whether it is in memory or not. If not then relocatable linking loader is called to load relocatable code of that routine. Then By Program address table is updated & Control Passed to newly loaded routine.

∴ Unused routine is never loaded.

Static linking -

In static linking all the library files are also combined with binary image of program at loading time. but

in this linking is postponed until execution in static linking memory is wasted because all program have their own copy of library files in exe.

Stub - in dynamic linking in the place of library refrence a stub is included in program image.

Stub is a small piece of code that indicates how that library routine is located into memory or how to load if not present in memory.

When Stub is executed Stub is replaced by the address of that routine.

next time if same code run no Stub is executed direct address is found.

in this scheme, all processes that use a language library execute only one copy of library code.

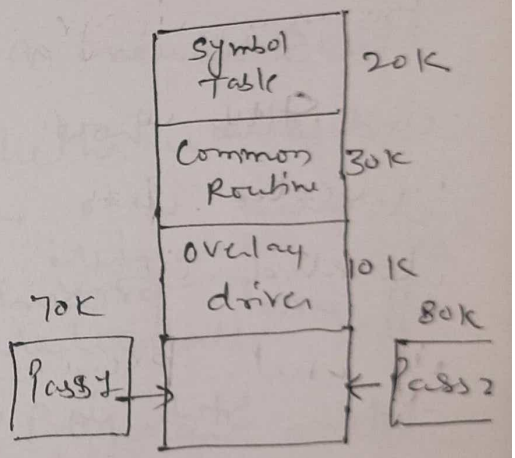
if library replaced by new version all the programs automatically use new version.

Overlays - A program to be executed must be in memory. if program is larger than allocated memory then How a program will execute ?

The idea of overlays is to keep only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded into space occupied by previously loaded instructions that are no longer needed.

example - two pass assembler

Pass 1	70K
Pass 2	80K
Symbol table	20K
Common Routine	30K
	200K



if only 150K is available then what? define two overlays -

overlay A - ST + CR + Pass 1 = 120K

overlay B - ST + CR + Pass 2 = 130K

∴ Speed becomes slower

Code on disk is kept as absolute memory image.

it requires complete knowledge of program

ical versus Physical address space -

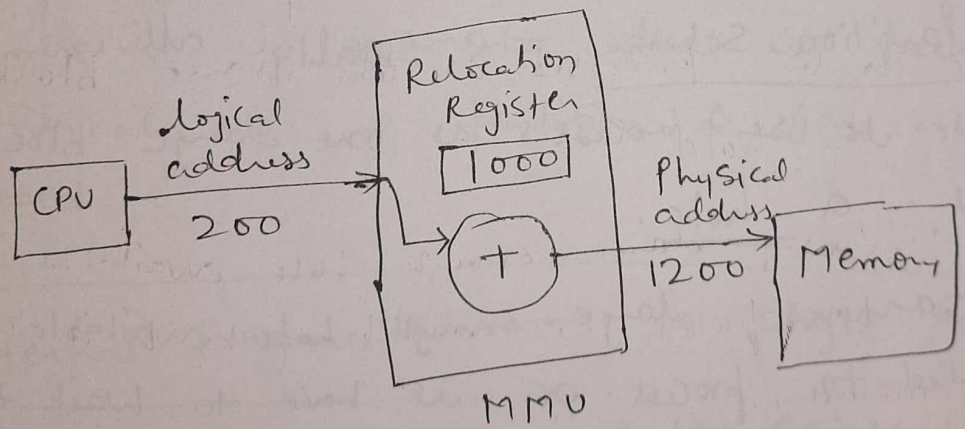
Logical Address – Address generated by CPU.
Physical Address - Address seen by memory unit.

Set of logical addresses – logical address space.

For execution time binding these both are different.

A Hardware device, Memory management unit (MMU) used to translate logical address to physical address.

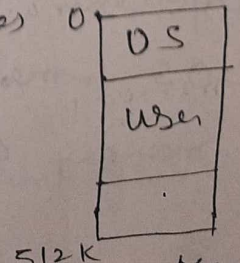
User program deals /sublies logical addres



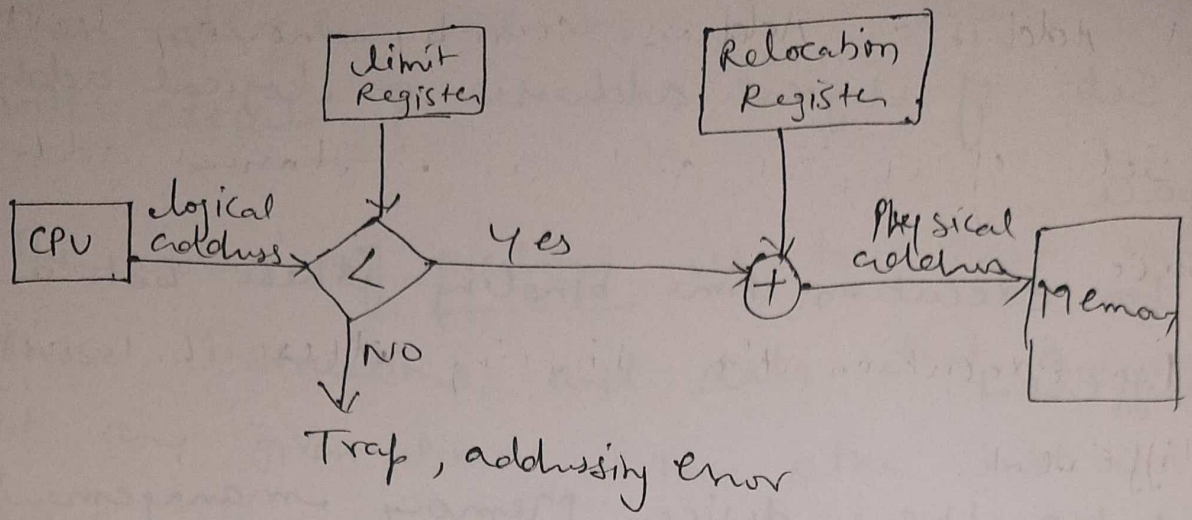
Contiguous Allocation –

Operating system & Interrupt vector resides in low memory & user process resides in high memory

Transient OS Code – it comes into memory & goes as needed.



it changes the size of operating syse
execution.



limit register used to protect processes to access each other code section.
limit register contains the range of logical addresses ie. 0 to 255

① Single Partition Scheme – Initially all memory is available to user processes as one large block called a hole.

Process arrives, large enough hole available is allocated to process. or it have to wait to free memory.
When process terminates its freed memory is added into hole list or merged with adjacent hole.
OS maintains a list indicating which part of memory are available and which are occupied.

allocate hole to process –

First Fit – Allocate the first hole that is big enough

Best Fit – Allocate the smallest hole that is big enough. This produce smallest leftover hole.

3) Worst - fit – Allocate largest hole. this produces largest leftover hole more usefule than Smallest

2) Multiple Partition Scheme – devide memory into a number of fixed size partitions. each Partition may contain one process. No. of Partitions define degree of multiprogramming

Drawback – Internal Fragmentation –

Allocated memory may be slightly larger than requested memory. difference between these two No. is internal fragmentation.

↳ it is memory that is free but internal to a partition & can not be used.

---

external fragmentation – in single Partition scheme, external fragmentation exists when enough total memory space exists to ~~stat~~ satisfy a request, but it is not contiguous Storage is fragmented into a large Number of small holes.

50-Percent rule – First Fit analysis reveals that

Swapping –

① in case of round robin scheduling quantum expired process is swap out

② in case priority scheduling low priority process is roll out & roll in

⇒ if address binding is done at the compile time or load time then swapped out process should be swap in at the same location.

⇒ Swapping requires a backing store i.e a fast disk.

Swapped process must be completely idle.

---

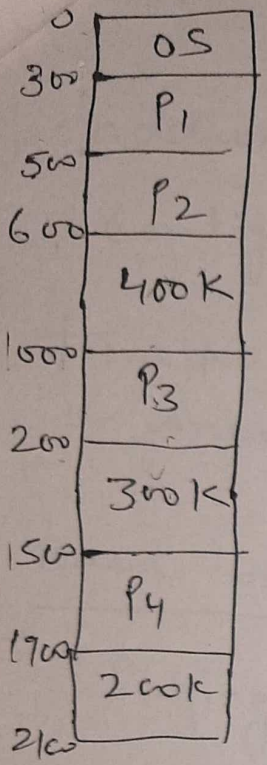
1 ✱ Continue –

Given N allocated blocks, another 0.5 N blocks will be lost due to fragmentation. That is 1/3 of memory is unusable.

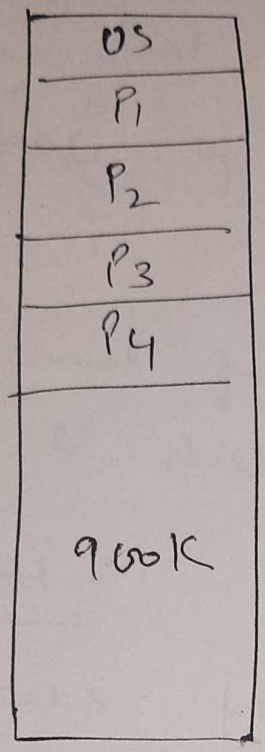
Solution – Compaction → Shuffle the memory holes to place all free memory together in one large block.

Only relocatable code can be compacted by changing base register value.

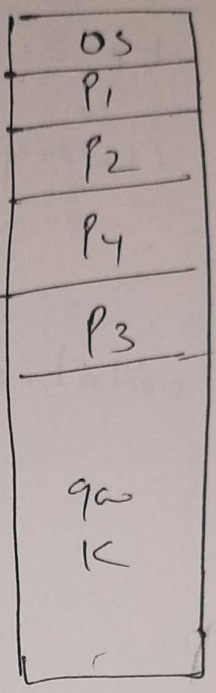
Compaction can also be combined with swapping & other ways may be used to compact memory.



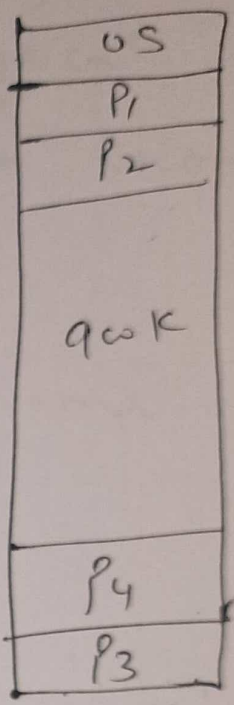
Original allocation



Moved 600K
P3 + P4
200 400



Moved 400K only P4 is Moved



Moved 200K only P3 is moved

Different ways to Compact memory

Ans - it is a memory management scheme that allows the physical address space of a process to be non-contiguous.

No external fragmentation in main memory & disk.

Basic method -

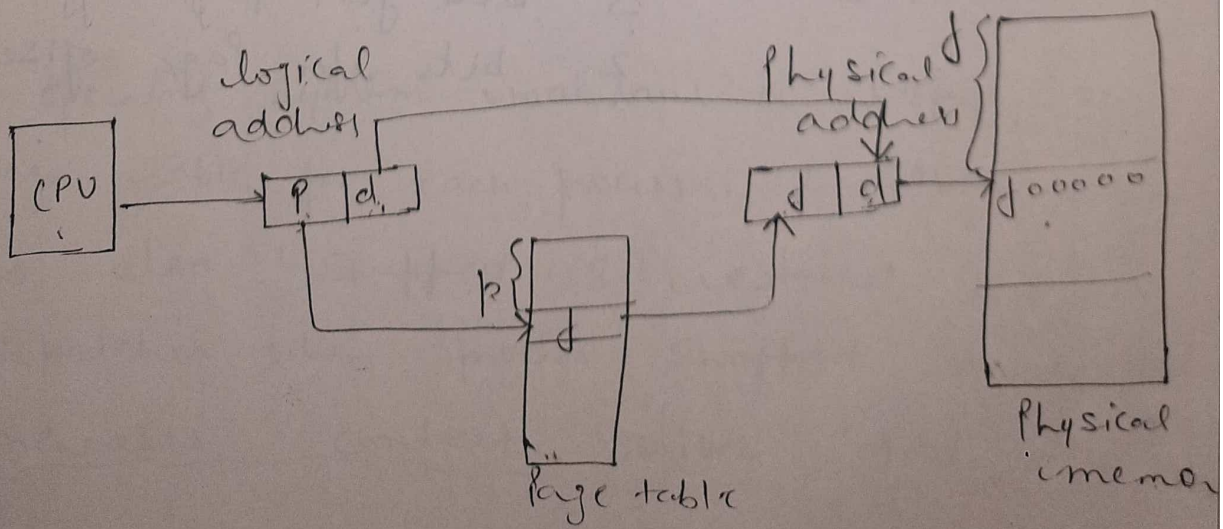
Paging breaks physical memory into fixed size blocks called **FRAMES** & disk into same size blocks called **PAGES**.

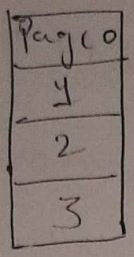
→ Every address generated by CPU is divided into two parts.

Page No. (P)	offset (d)

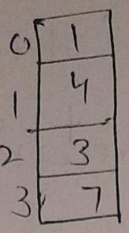
↳ Index in Page table (where base address for this page is stored)

↳ displacement within that page.

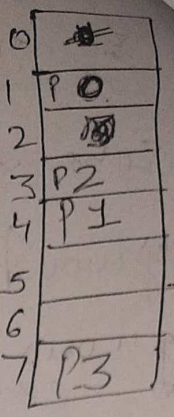




logical memory (disk)



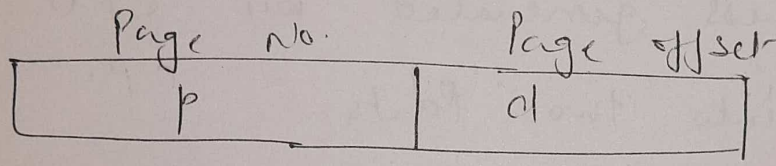
Page table



Page size is defined as power of 2, it makes translation of LA to PA easy.

→ Size of logical address space $\rightarrow 2^m$

Page size $\rightarrow 2^n$



$m-n$ bits — n bits

3 — 2

eg. 32 bytes address space $- 2^{5=m}$

4 bytes per page $- 2^{2=n}$

(8 Pages)

5-2 = 3 bits for page No.

2 bits for Page offset

no external fragmentation but internal fragmentation may appear.

eg: 4 bytes per page if process is 9 bytes long than 3rd page will have 3 bytes free.

Small Page Size can reduce this problem but Page table entries increase & disk I/O will be less efficient. (smaller data transfer)

A process to be executed of n pages should have n frames available in memory.

User View of memory & actual physical memory view is separate

frame table by OS is maintained in which frame is allocated or free, & to which process it is allocated is written.

operating system maintains a copy of page table for each process. & this copy is also swapped & restored by CPU dispatcher when process swapped so it increases context switch time

Hardware support - Hardware implem

of the page table can be done in seve ways -

(1) Page table implemented as set of dedicated registers, these registers are reload by CPU dispatcher as other registers.

→ registers should be built with high speed logic to speed up translation process.

→ but costly if Page table is very large

(2) Page table is stored in main memo & Page table Base register (PTBR) points to the page table.

⇒ Changing page table require to change contents of this register, reducing contex switch time.

⇒ With this scheme two memory access are needed to access a byte (one for page table, other for byte itself)

→ Translation look-aside buffer (TLB) — high speed, Associative, cache memory is used to speed up Page table look up time.

Several

Value (Associative memory)

Page No.	Frame No.

Page No. from logical address is matched with key value of TLB if it found Corresponding value (frame No.) is returned if Not found (TLB Miss), memory reference of page table is generated & that entry is also made in cache (TLB).

if TLB is full LRU policy is generally used.

⇒ Wired down entries can not be removed from TLB. eg TLB entry for Kernel code

⇒ Address space identifier (ASID) is just a unique No. for each process as process id.

⇒ With the page No. ASID value stored also be matched to be correct frame No.

if ASID DO not match means TLB miss

⇒ ASID allows the TLB to contain entries for several different processes simultaneously

Page No.	ASID	Value (Frame No.)
0	1	4
1	1 (Process 1)	5
2	1	6
1	2 (Process 2)	1
2	2	7

Page No. are same i.e ASID

⇒ if TLB does not support ASID every time a new page table is selected TLB must be flushed.

Hit ratio - The percentage of times that a particular page No. is found in the TLB is called hit ratio.

Hit ratio = 80%

TLB access time = 20 ns

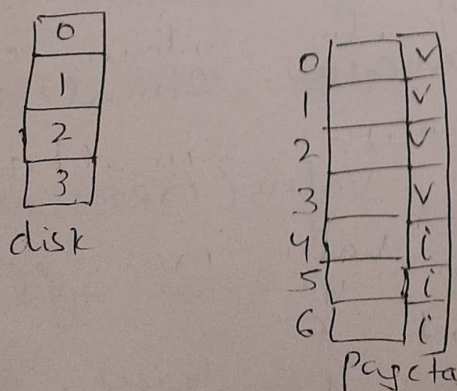
Memory access time = 100 ns

effective access time = .80 X 120 + .20 X 220

= 140 nsec.

Protection - A protection bit in page table is associated with each page like

→ read-write
→ read only
→ execute only
→ valid
→ invalid → Page is in the process's logical address space
→ Not in process's logical address space



← if Page table is of fix size like 7.

table length register (PTLR) –

define length of page table.

every logical address checked against PTLR to verify that address is in process range.

Shared Pages –

if the code is reentrant code or Pure code (it never changes during execution) then it can be shared by many processes at a time.

eg. text editor code – 150 KB
data – 50 KB

40 user want to run text editor then

200 × 40 = 8000 KB space is needed

but if page size is 50 KB then Page table of each user maps onto same physical copy of the editor. data is different for each user

Now

40 × 50 + 150
= 2150 KB is needed.

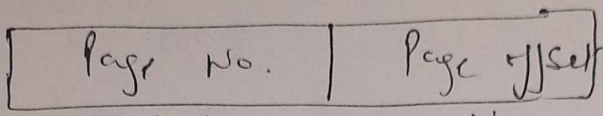
Multilevel Paging -

if logical address space is very large then Page table itself becomes excessively large.

example - 32 bit logical address space

Page size 4K (2^{12})

Means

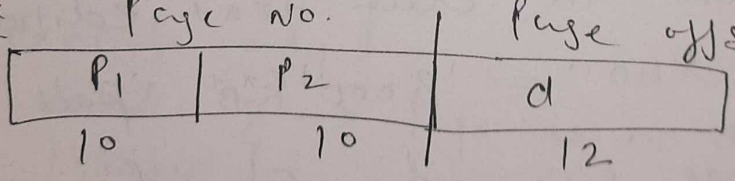


here Page table itself larger than one Page.

Page table also ~~is~~ may not be granted Contiguous memory allocation.

Soln - divide Page table into pieces (Paging of Page table)

Example



two level Paging

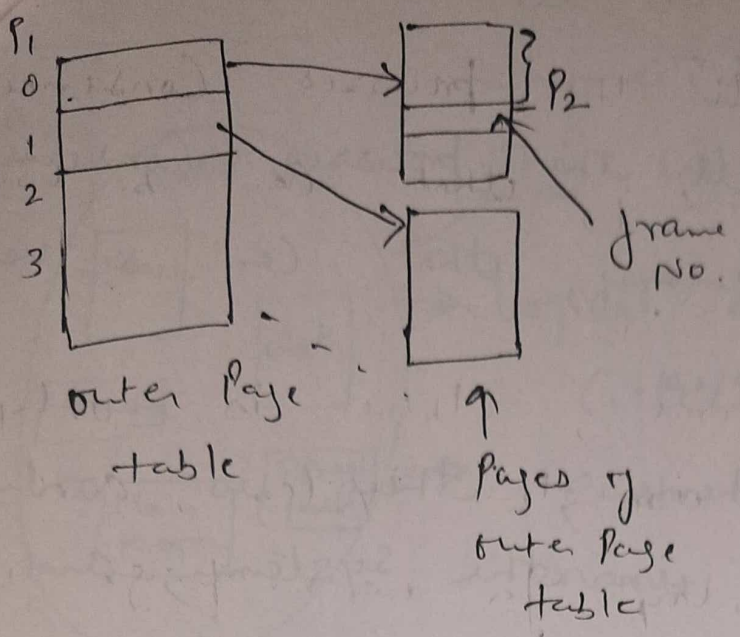
P_1 → Index in outer Page table

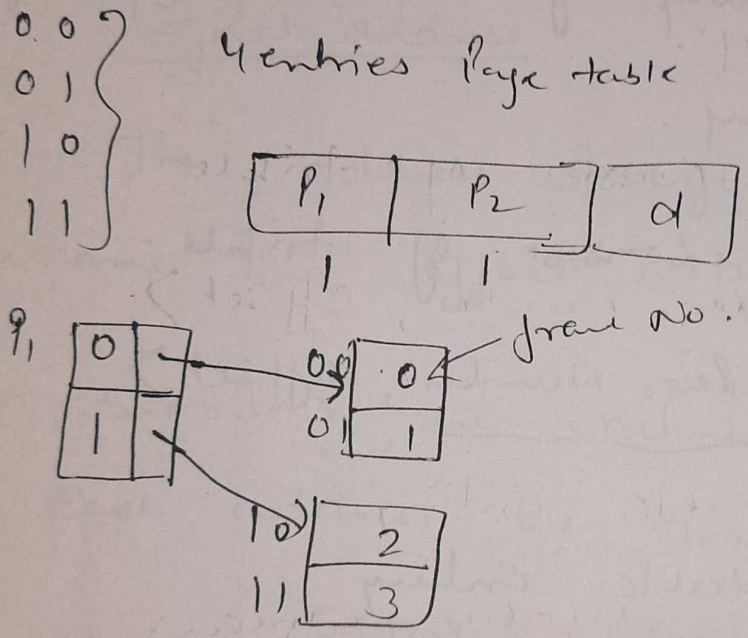
it gives Page Index into inner Page table

→ base address of page containing frame no. for that address

P_2 → Index into inner Page table whose address

Provided by P_1





Memory access increased.

Inverted Page table –

Page tables for all the processes consume more physical memory, that can be used in other processes.

in this scheme, there is only one page table in the system, and it has only one entry for each page of physical memory.

Virtual address consist of tribe-

< Process id, Page Number, offset >

↓

Page table entry

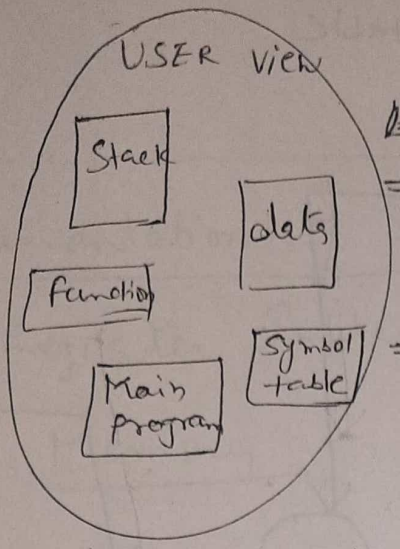
if found at i location means + this Virtual Page it at i^{th} frame and physical address

< i, d > is generated.

∴ Search time increased because Search is based on virtual address while table is sorted according to physical address.

Hash table + associative Look up may be used to fast Search process

nentation – it is a memory mngt. Scheme that supports this user view of memory.



⇒ logical address space is a collection of segments.

⇒ each segment has a name and length

Logical address

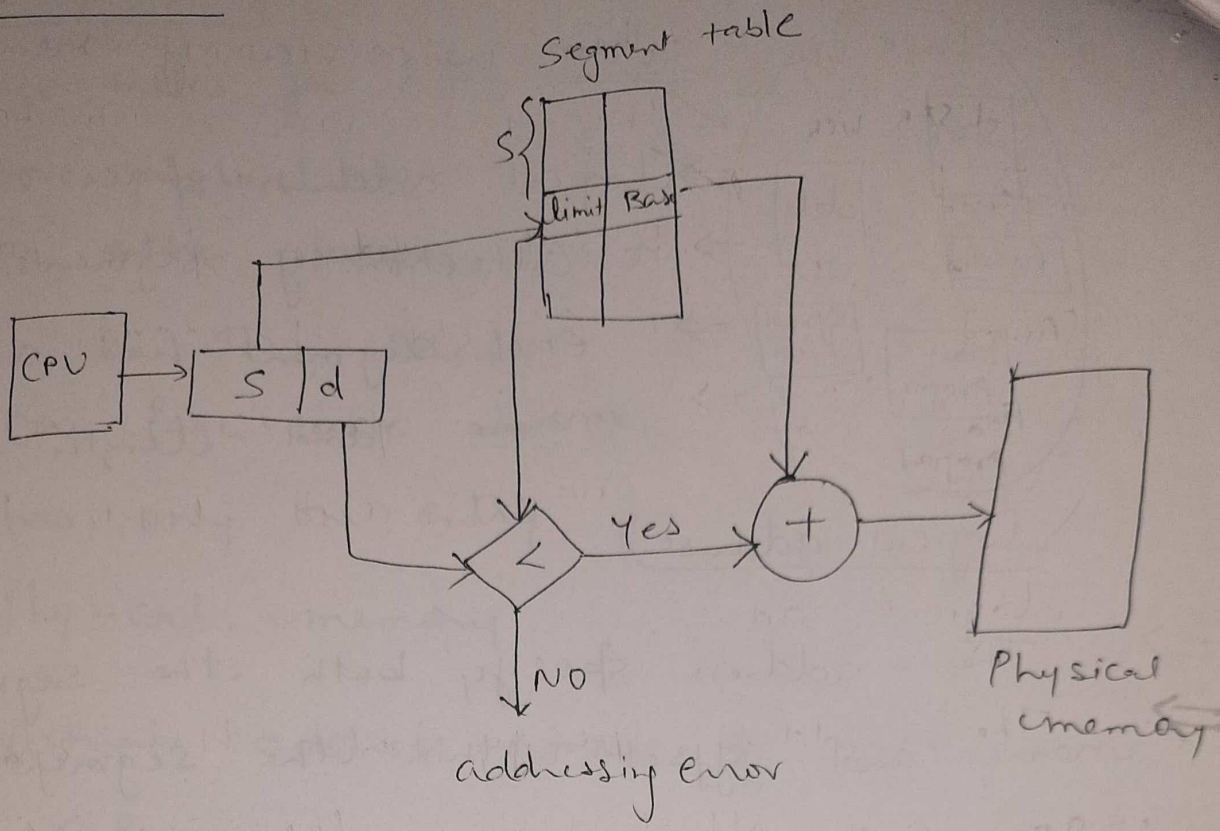
⇒ The address specify both the segment name and offset within the segment.

⇒ Segments are numbered & user gives each address by two tuples –

< segment-Numbe , offset >

⇒ ① Compiler construct segments
② loader assign segment Numbers.

Hardware -



User see two dimensional address but actual physical address is one dimensional to translate logical address to physical address segment table is used.

Base → Starting address of segment

limit → length of segment.

if d < limit then S is used to index into segment table to find base address of that segment. base address is added into offset to generate physical address.

if d > limit then addressing error is generated.

Implementation of segment table -

(1) Register Implementation → fast but expensive

(2) Memory - STBR, STLR } Registers Maintained

(S, d) ← logical address pair

if S < STLR then

STBR + S gives base address of that segment

Protection & sharing - Protection bit associated to check if segment is read only or execute only.

Shared segment → all segment table point same physical copy of shared segment.

Fragmentation - segments are of variable length. variable size partition scheme is used and best fit or first fit algo used to allocate memory.

External fragmentation may cause.