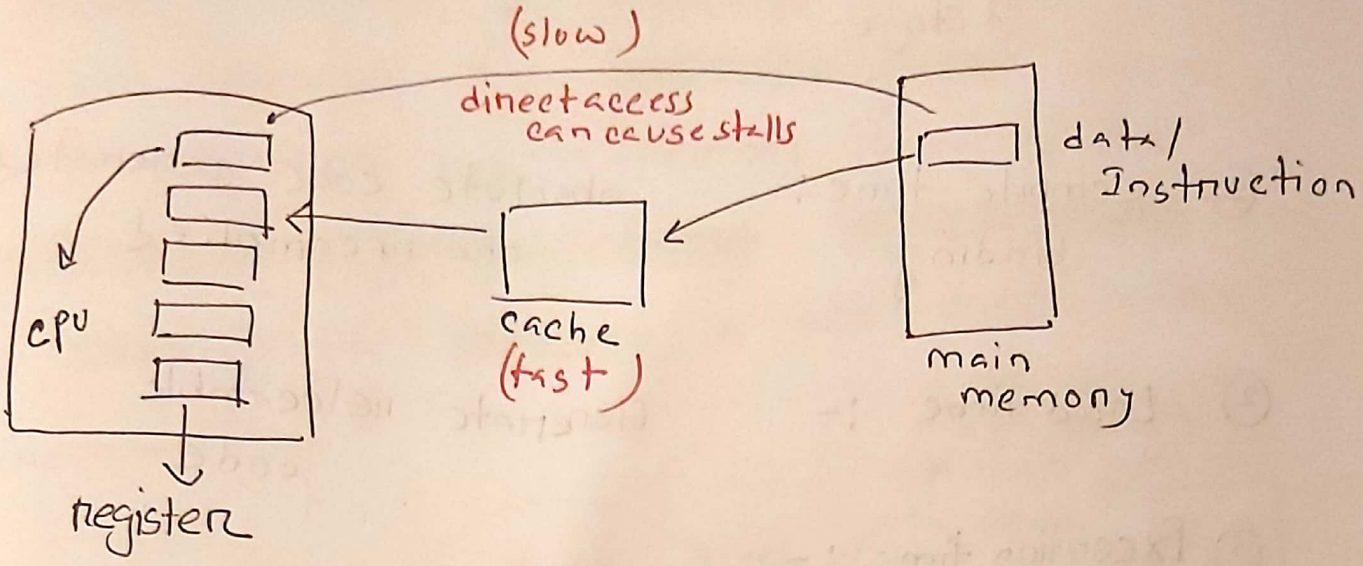
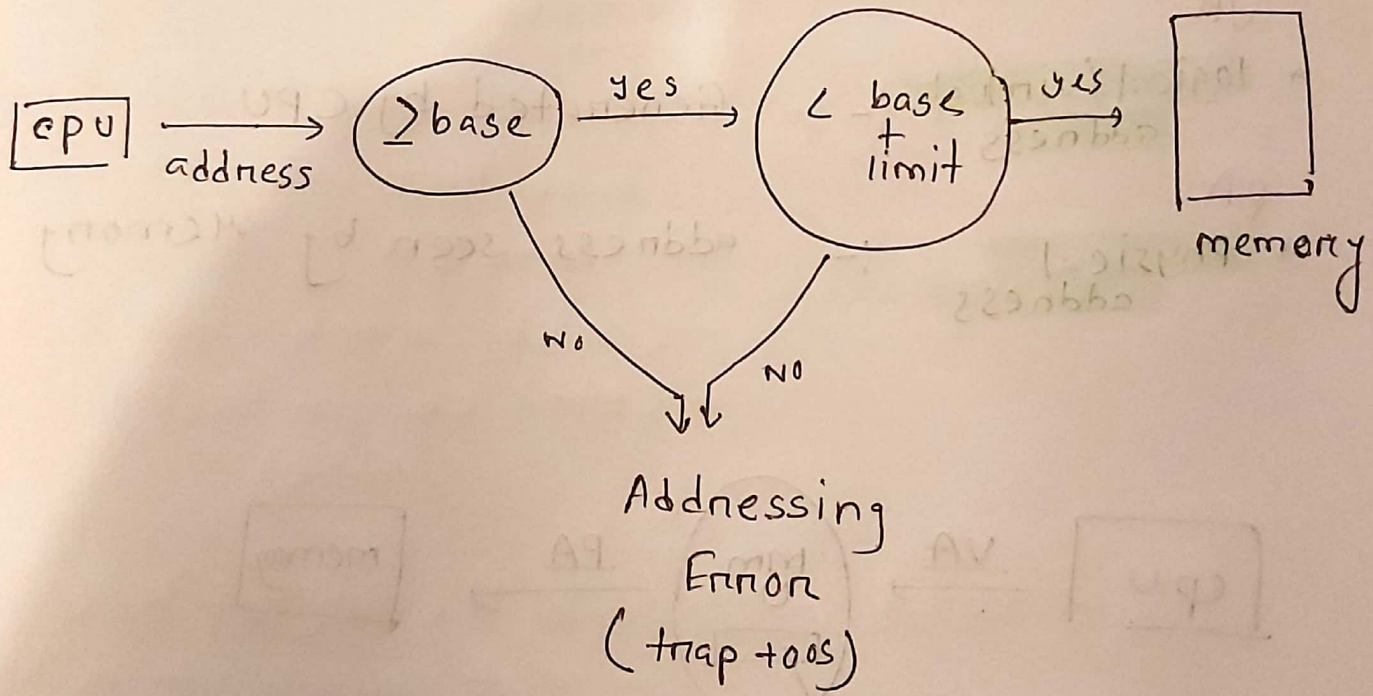
c-9

Main Memory



Handware Address protection



Addres Binding

3 stage

① Compile time :- binding — absolute code generated and recomplied

② Load time :- Generate relocatable code

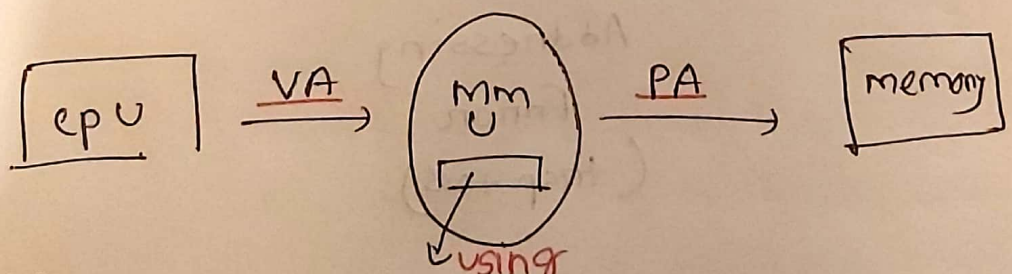
③ Execution time :- binding

VA

* logical/virtual address :- Generated by CPU

PA

* Physical address :- address seen by Memory



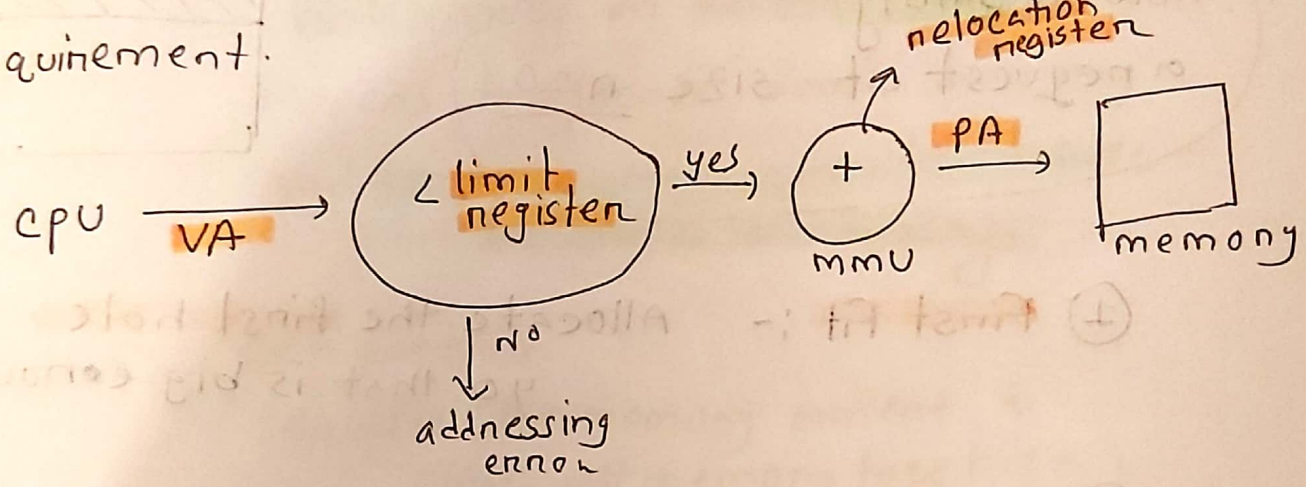
what is mmu?

mmu - Memory management Unit

- is hardware device that maps VA to PA at runtime

what is contigous Memory Allocation?

→ refers to memory management technique in which whenever there occurs a request by a user process for the memory, one of the sections of contigous memory block would be given to that process in accordance with its requirement.

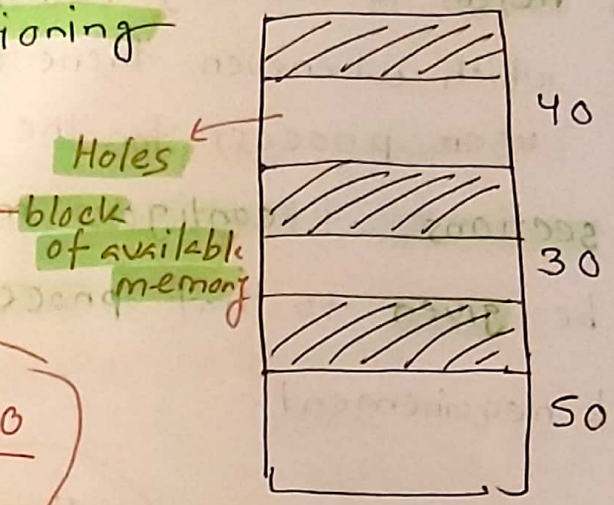


Variable Partition

— dynamic partitioning

(1) Free/hole panition

(11) Allocated panition



How to satisfy a request of size n=20

(1) First Fit :- Allocate the first hole that is big enough
40

(2) Best Fit :- Allocate the smallest hole that is big enough
here 30
- (must search whole list)
- (produce smallest leftoven hole)

(3) worse fit :- Allocate the largest hole
- must search whole list
- produce large leftoven hole

Speed, storage utilization :- First, Best > worse

Fragmentation

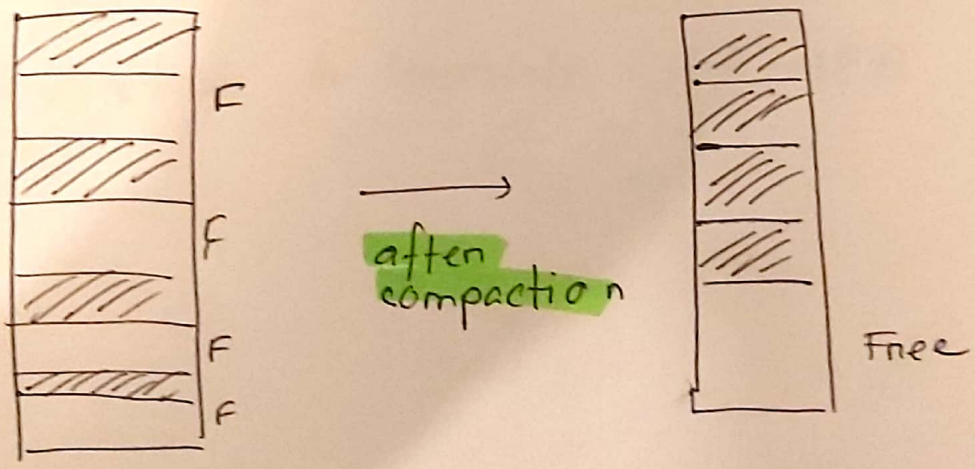
→ means small fragment on holes of memory which are located here, there in main memory

two types

① External Fragmentation :-

- occurs whenever a method dynamic partitioning on memory allocation happens to allocate some memory and leave a small amout of hole on unusable memory

Solution :- Compaction
(shuffle the memory content to place all memory together)



Internal fragmentation :-

- if allocated memory is slightly greater than requested memory ; a unused free space is created which cause Internal fragmentation

— Gave 1000 to 800 request process
200 wasted / unused

— reason first fit method

Paging

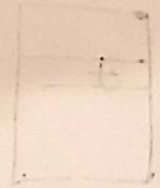
→ is a storage mechanism used in OS to retrieve process from secondary storage to main memory as pages

- avoid external fragmentation
- avoid problem of varying sized memory chunks

(1) Frames
- Divide physical memory into fixed sized blocks

(2) Pages
- Divide virtual memory into block of same size

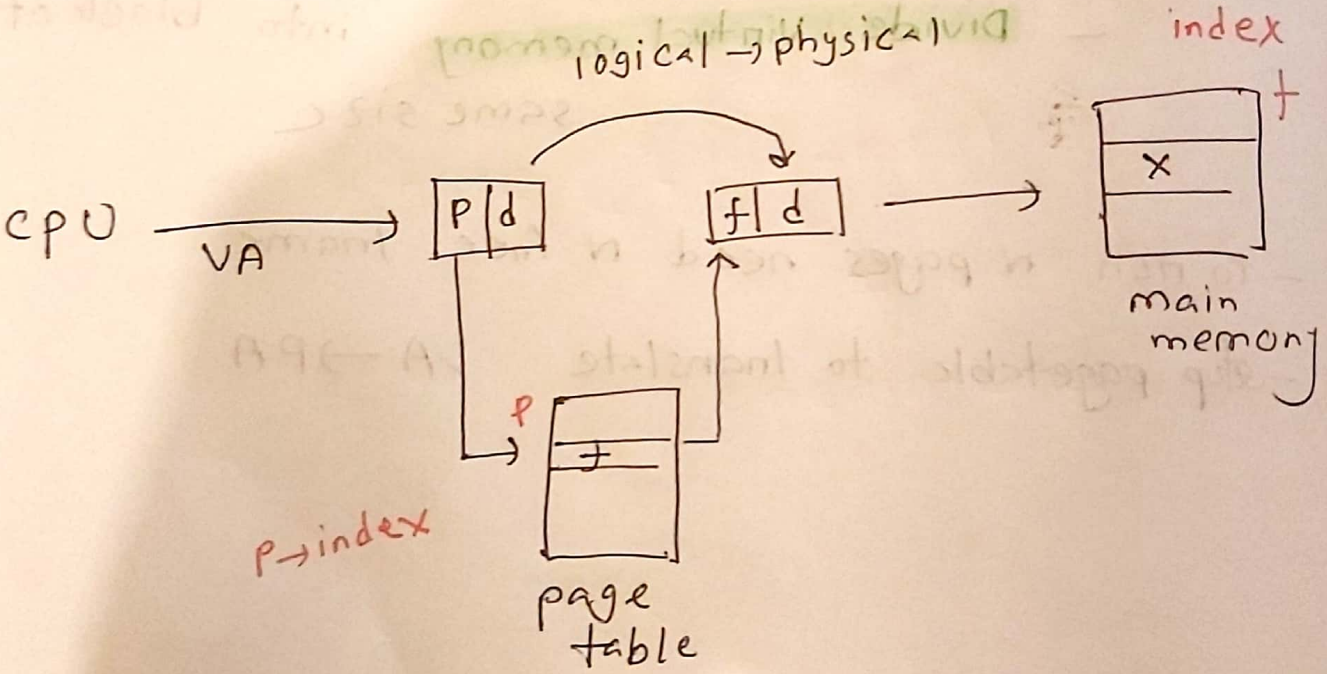
- To run N pages need N free frame
- setup pagetable to translate VA → PA



Address translation scheme

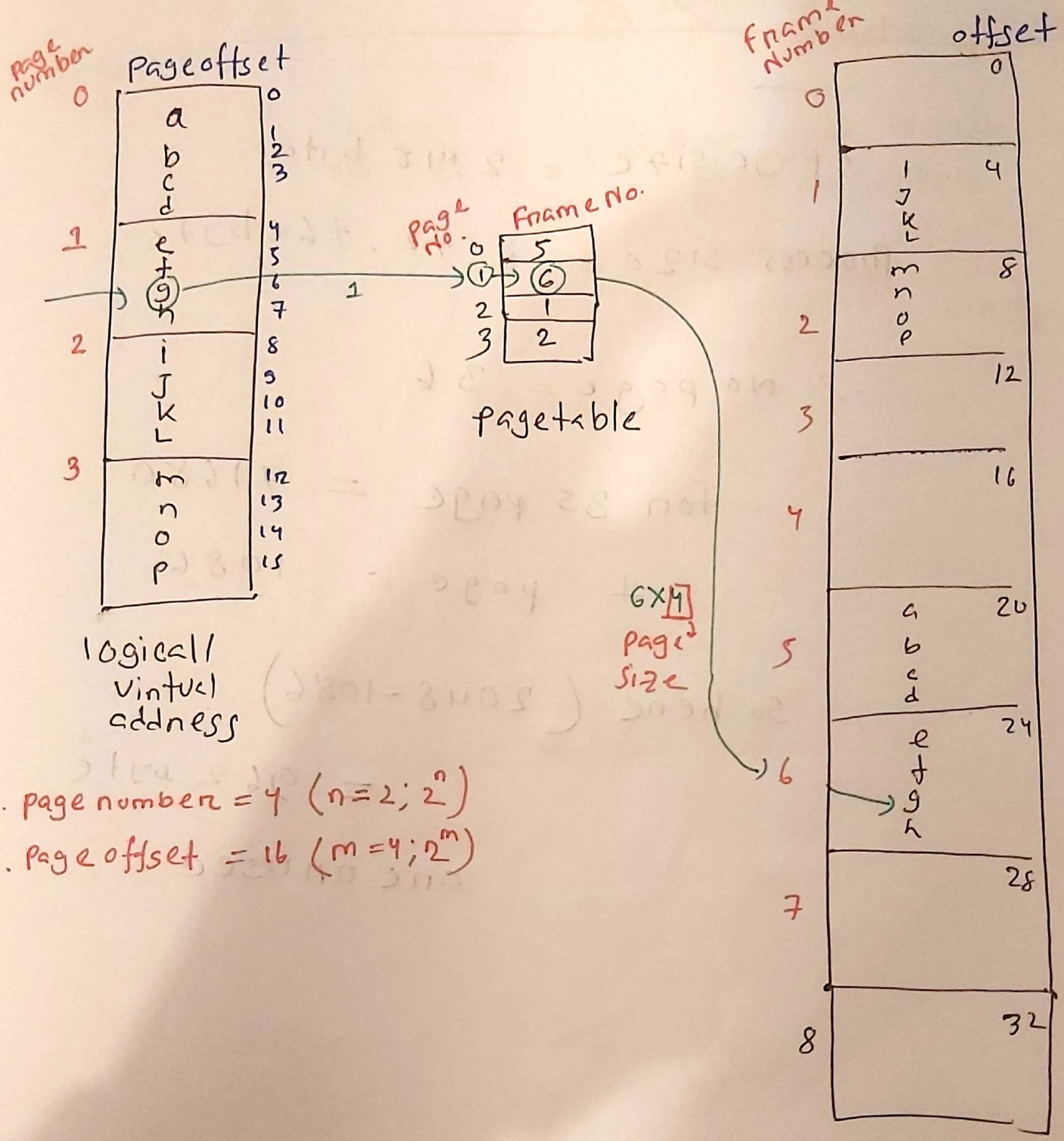
- Address generated by CPU divided into parts

1. Page Number (p) — used as index in pagetable — base address of page in physical memory
2. Page offset (d) — define the physical memory address that is to be sent



Paging handware

Paging



∴ page number = 4 ($n=2; 2^n$)

∴ page offset = 16 ($m=4; 2^m$)

Page size = 4 byte

Frame 8.

memory size = 32 byte

Paging can't reduce internal fragmention :-

∴ Page size = 2,048 bytes

∴ Process size = 72,766 byte

∴ No page = 36

∴ for 35 page = 71680

∴ 1 page = 1086

∴ So here (2048 - 1086)

= 962 byte

are unused

Implementation :-
of page table

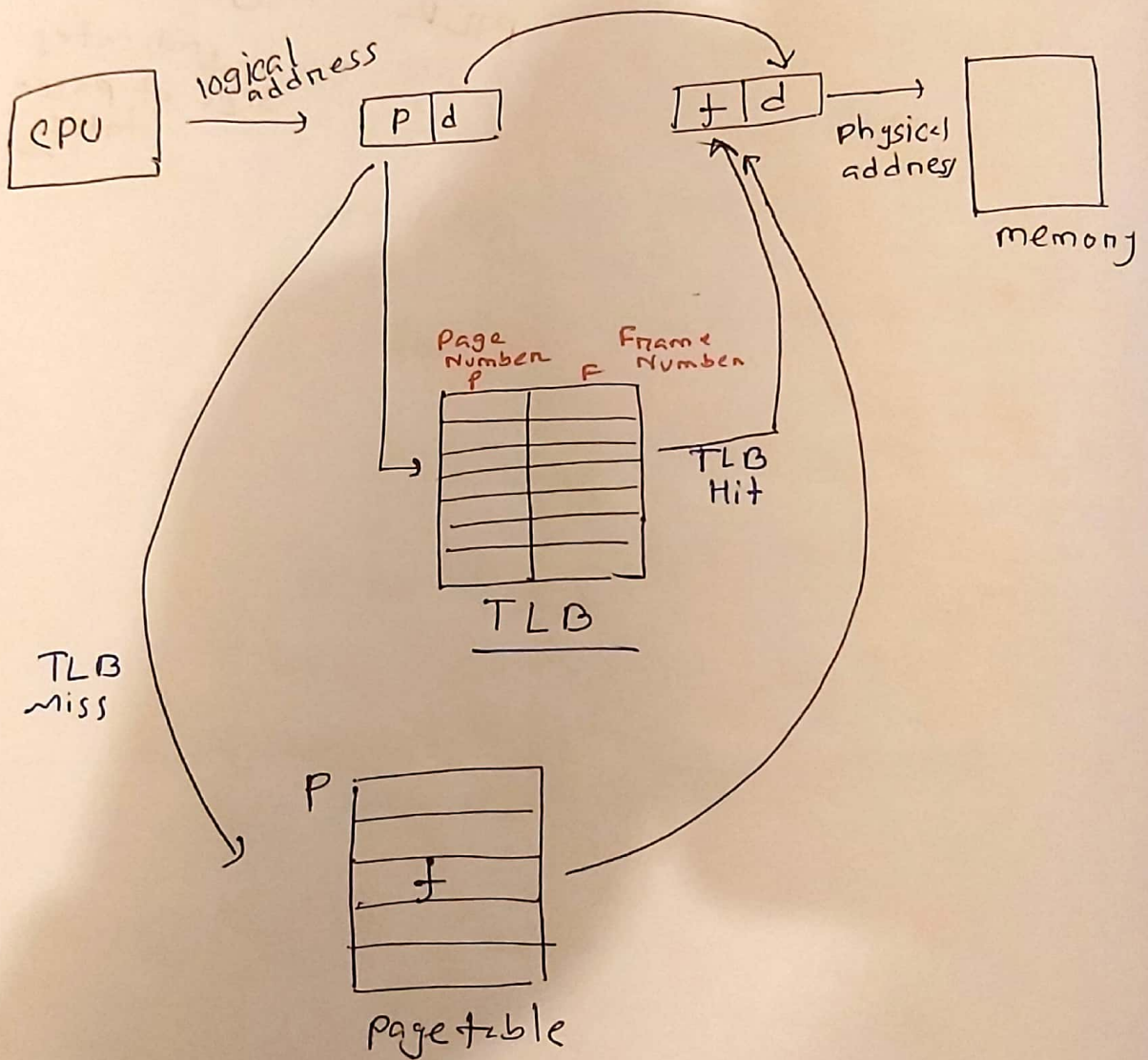
point to actual page table

Page table ⟶ 2 page table

① Page table base register PTBR

② Page table length register PTLR indicates size of page table

TLB → Translation look aside buffer is a memory cache that store recent translations of virtual memory to physical address for faster retrieval on lookup



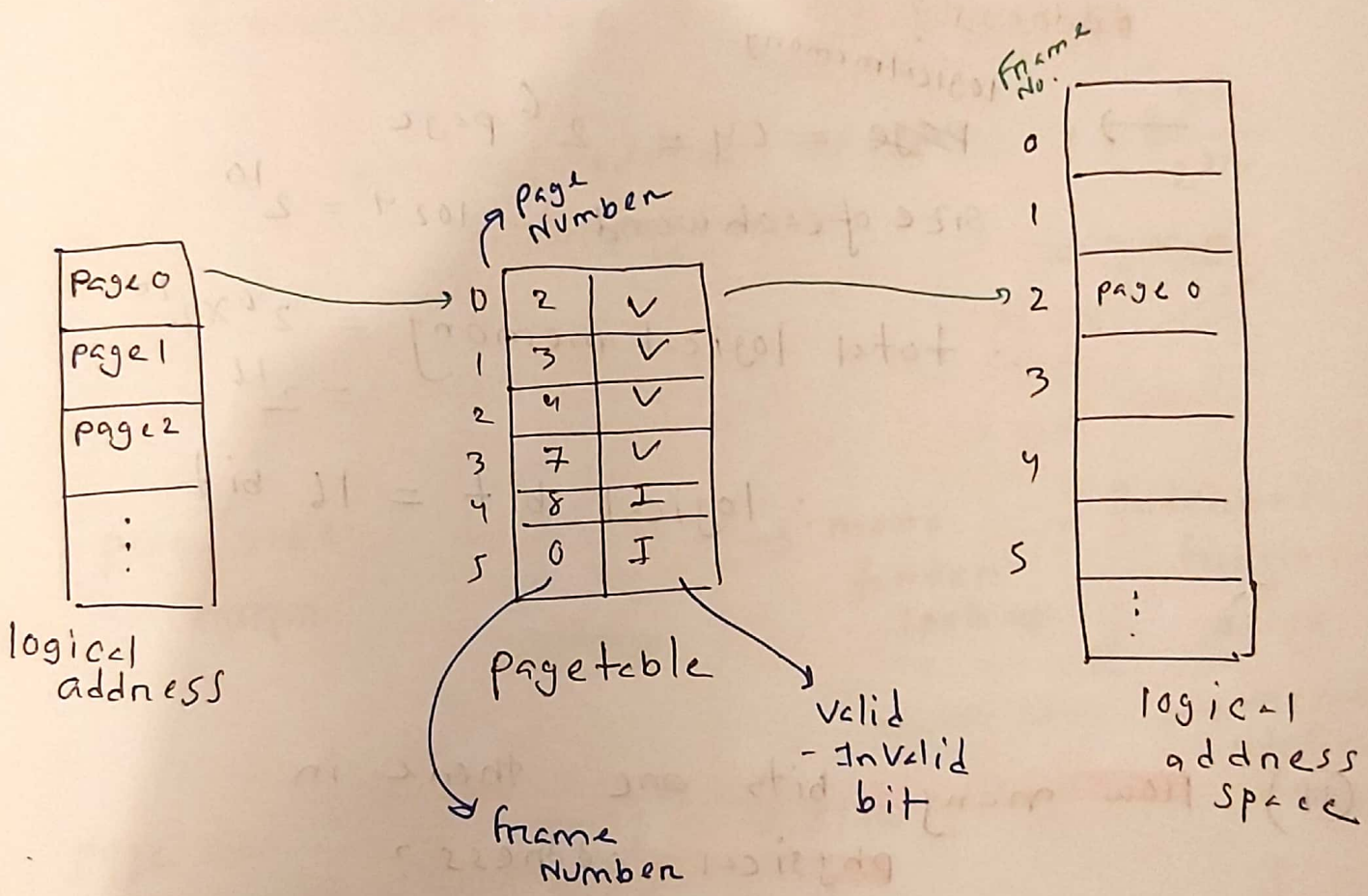
Memory Protection

→ implemented by associating protection bit with each frame to read only or write ~~only~~ access is allowed

Valid - InValid bit

→ indicated the page not process logical address space

indicates legal page



consider a logical address space of 64 pages of 1024 each word and mapped into a physical memory of 32 frames?

(I) How many bits are there in logical address?

→ logical memory

page = 64 = 2^6 page

size of each word = 1024 = 2^{10}

∴ total logical memory = $2^6 \times 2^{10}$

$= 2^{16}$

∴ logical bit = 16 bit

(II) How many bits are there in physical address?

→ physical memory = 32 frames

$= 2^5$ "

∵ size of each word = $1024 = 2^{10}$

∴ total physical memory = $2^{10} \times 2^5$

$= 2^{15}$

∴ physical bit = 15

Given,

logical address bit = 32

∴ page size = 4KB = 2^{12}

∴ page table entry = $\frac{2^{32}}{2^{12}}$

∴ if each entry size 4 byte

physical address space for page table = $4 \times \frac{2^{32}}{2^{12}}$

= 4MB

page size বাড়লে → page entry কমবে → more fanten look up ∴ Internal fragmentation বাড়বে → memory utilize কমবে → searching time কম

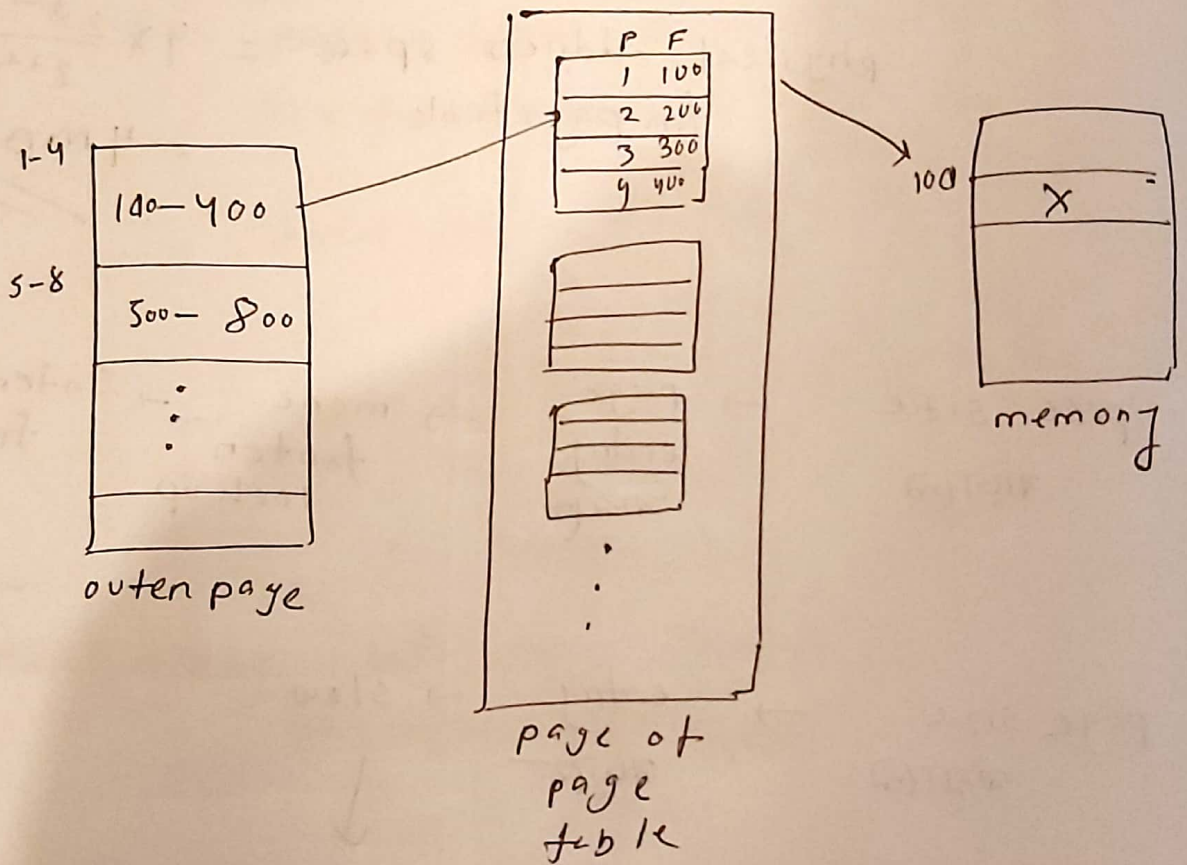
page size কমলে → entry বাড়বে → slow → Internal frag. কমবে → search time বাড়বে

Hierarchical page table

→ two level page table

→ is a paging scheme that consists two or more level page in hierarchy manner

→ break up the logical address space into multiple page table

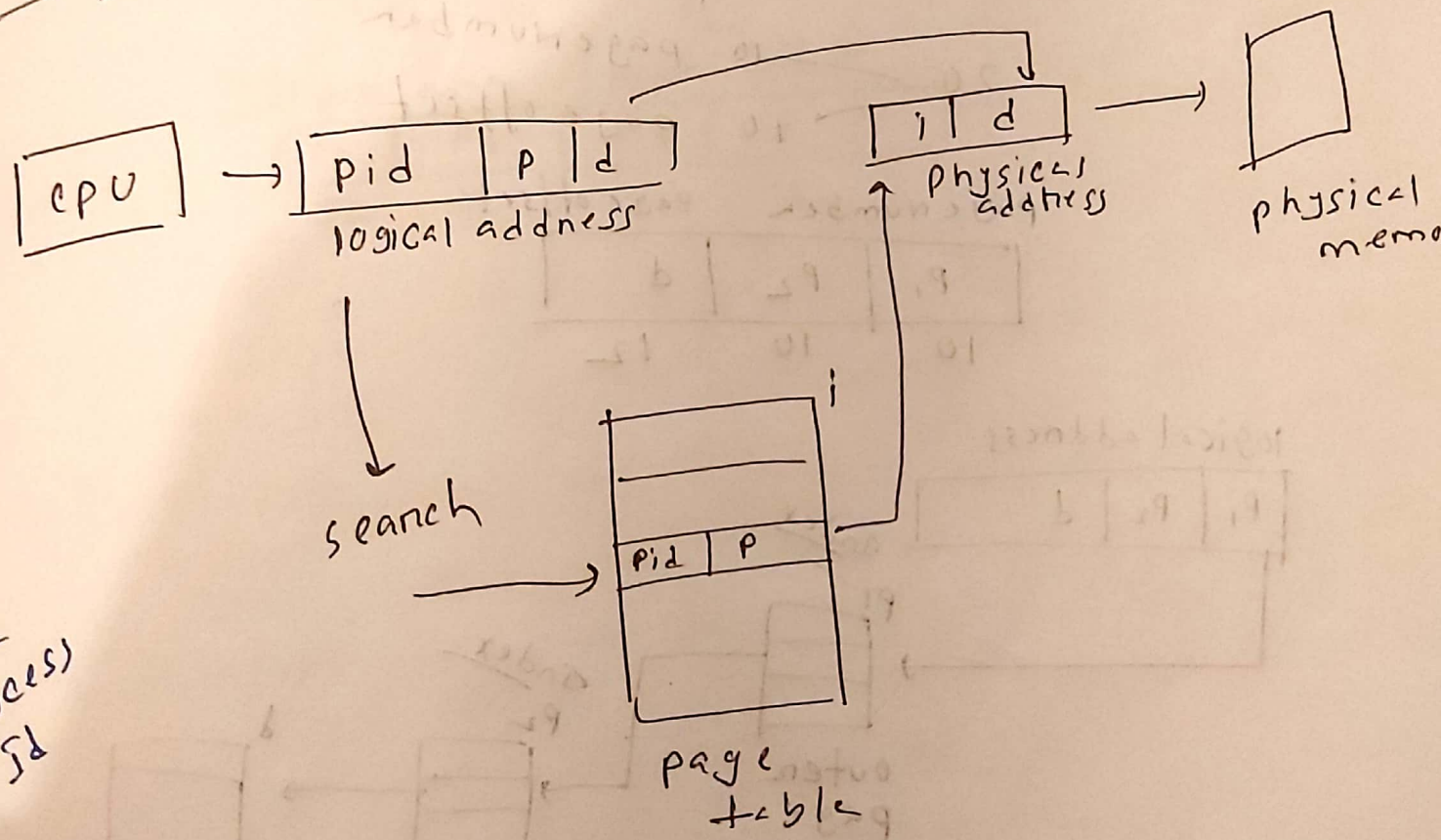


Help to solve if the size of page table is langer than frame size in memory?

Structure of page table

1. Hierarchical / multilevel page table
2. Hashed page table
3. Inverted page table

Inverted page table



pid
process id

forward mapped page table

Two level paging

A logical address (32 bit machine and 4kB page)

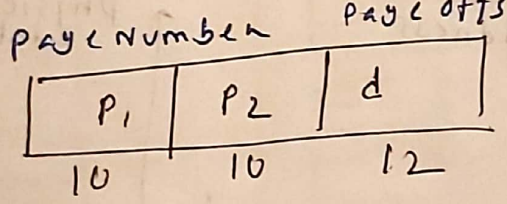
page size = 4KB = 2^{12}

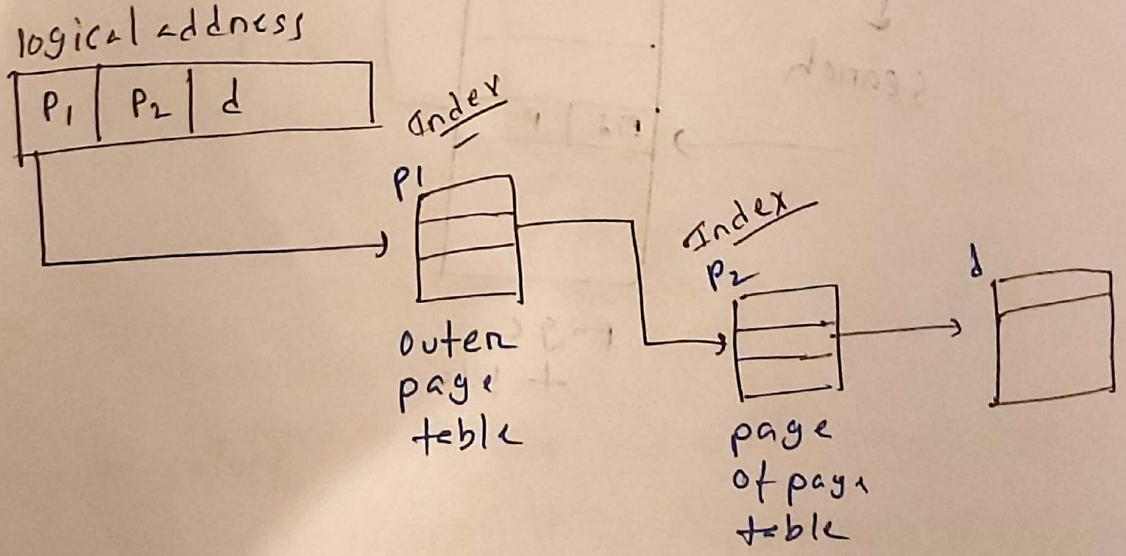
∴ a page offset = 12 bit

∴ page number = (32-12) = 20 bit

For two level paging → the page number is divided into 2 part

20 — 10 page number
20 — 10 page offset





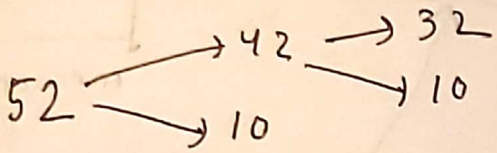
Three level paging

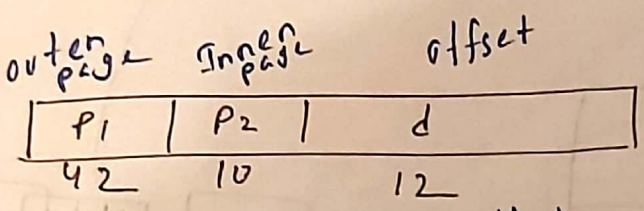
64Bit machine, 4KB page

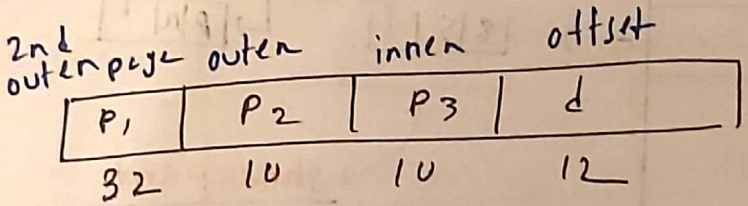
∴ page size = 4KB = 2^{12}

∴ page offset, d = 12

∴ page number, p = 64 − 12 = 52 bit

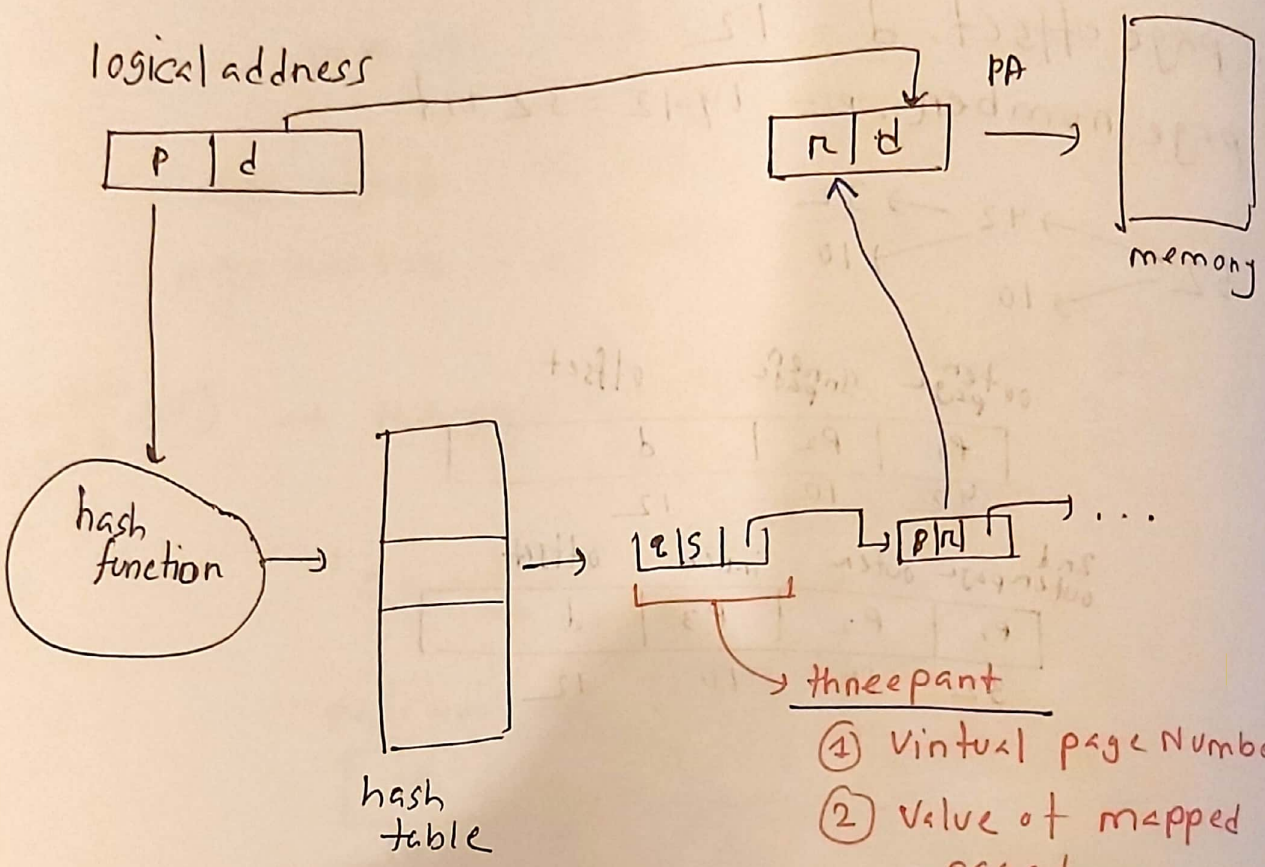






Hashed Page table

logical address space > 32 bits



threepant

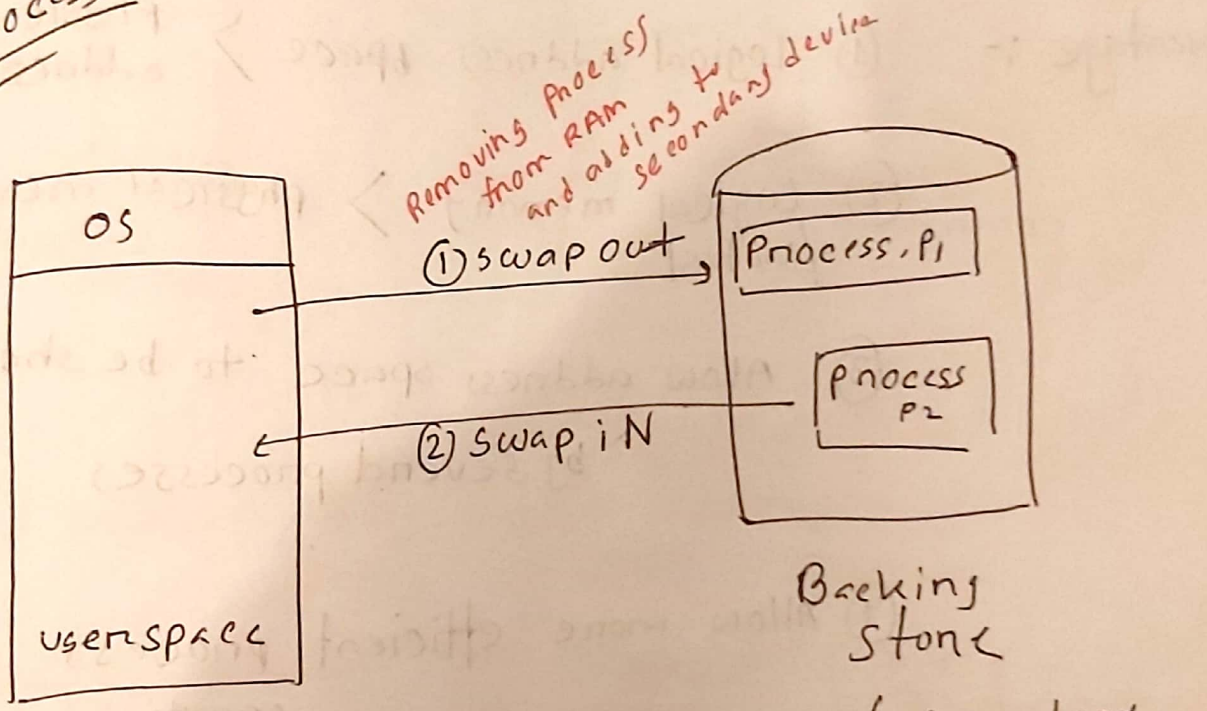
1. Virtual page Number
2. Value of mapped page frame
3. pointer to next element

works like hashing

Swapping

Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that main memory can be available for other process.

Swapping two process



Schematic view of swapping