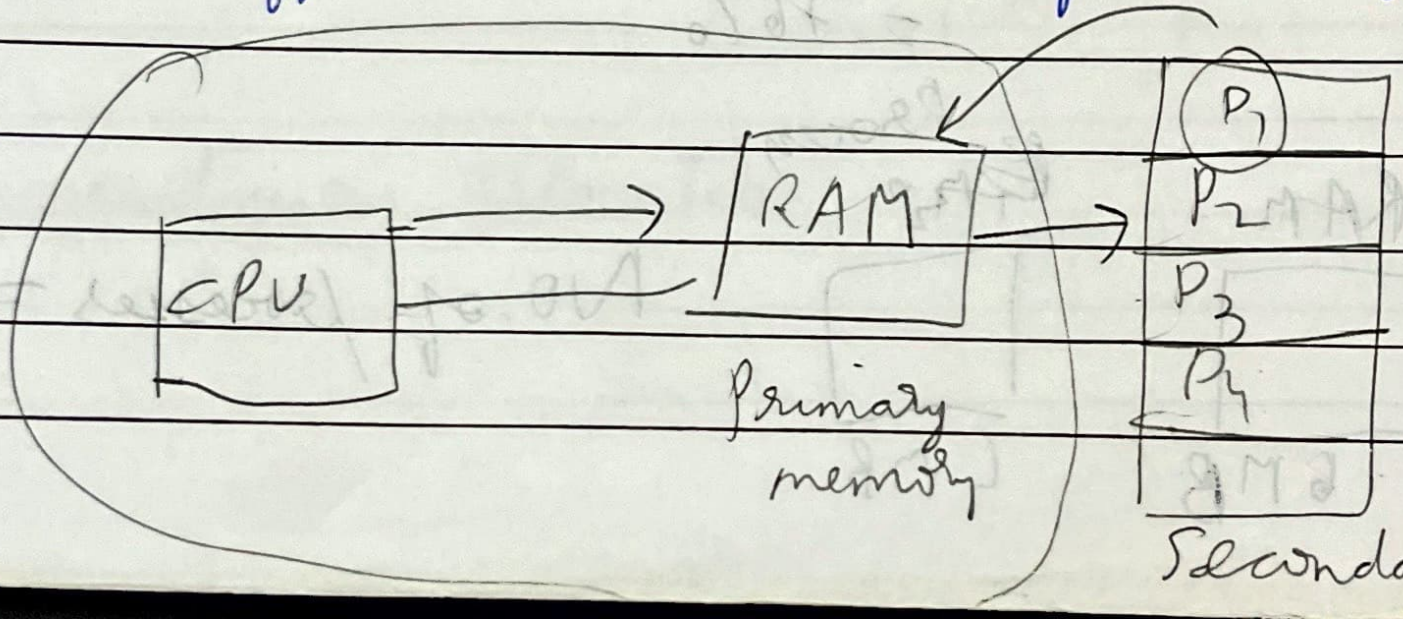
L-5.1 Memory management and degree of multiprogramming

Memory management → method of managing primary memory

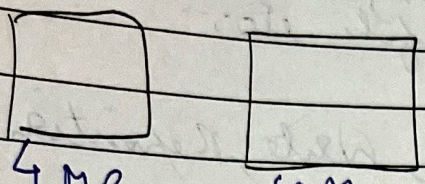
Goal: Efficient Utilisation of memory



Process

CPU I/O

RAM Process



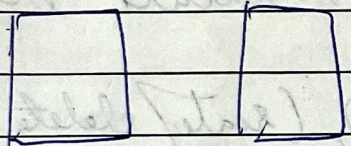
'k' = I/O operation (70%) ← Probability of

CPU = (1-k) = 30%

CPU utilisation

$\frac{4MB}{4MB} = 1$

RAM Process



No. of Processes

$= \frac{8MB}{4MB} = 2$

k^2

CPU utilisation $= (1-k)^2$

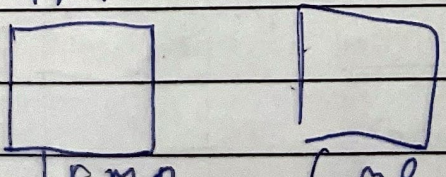
$k = 70\%$

CPU $= 1 - (0.7)^2$

$= 76\%$

Process

RAM



No. of processes = 4

R^4

CPU utilisation $= 1 - R^4 =$ 93%

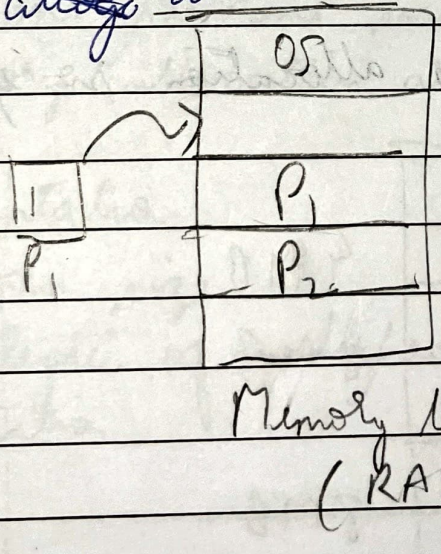
Observation :- The ~~no of~~ size of the ram is directly related to the CPU utilisation

L-5-2 : Memory management techniques - Contiguous and non-contiguous

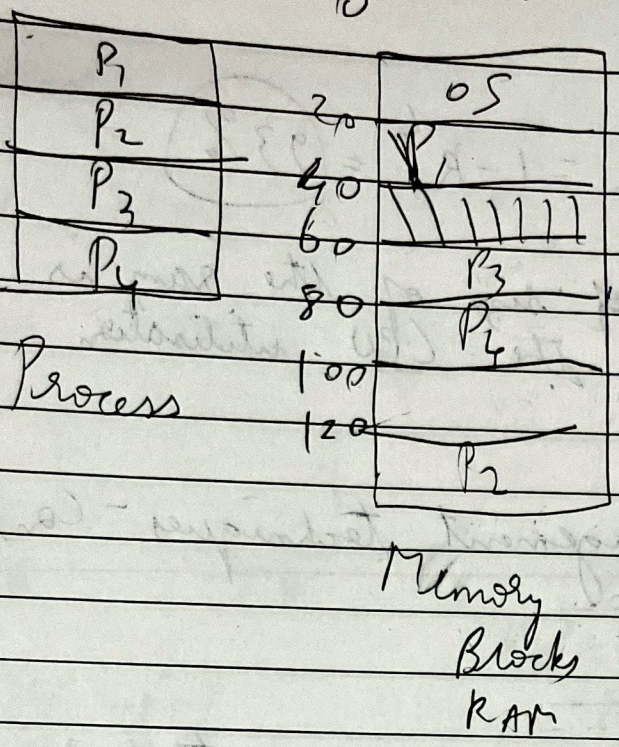
Memory management techniques

- Contiguous
  - Static
    - Fixed Partition
  - Dynamic
    - Variable partition
- Non-contiguous
  - Paging
  - Multilevel Paging
  - Inverted Paging
  - Segmentation
- Segmented paging

Contiguous ~~allocation~~ allocation



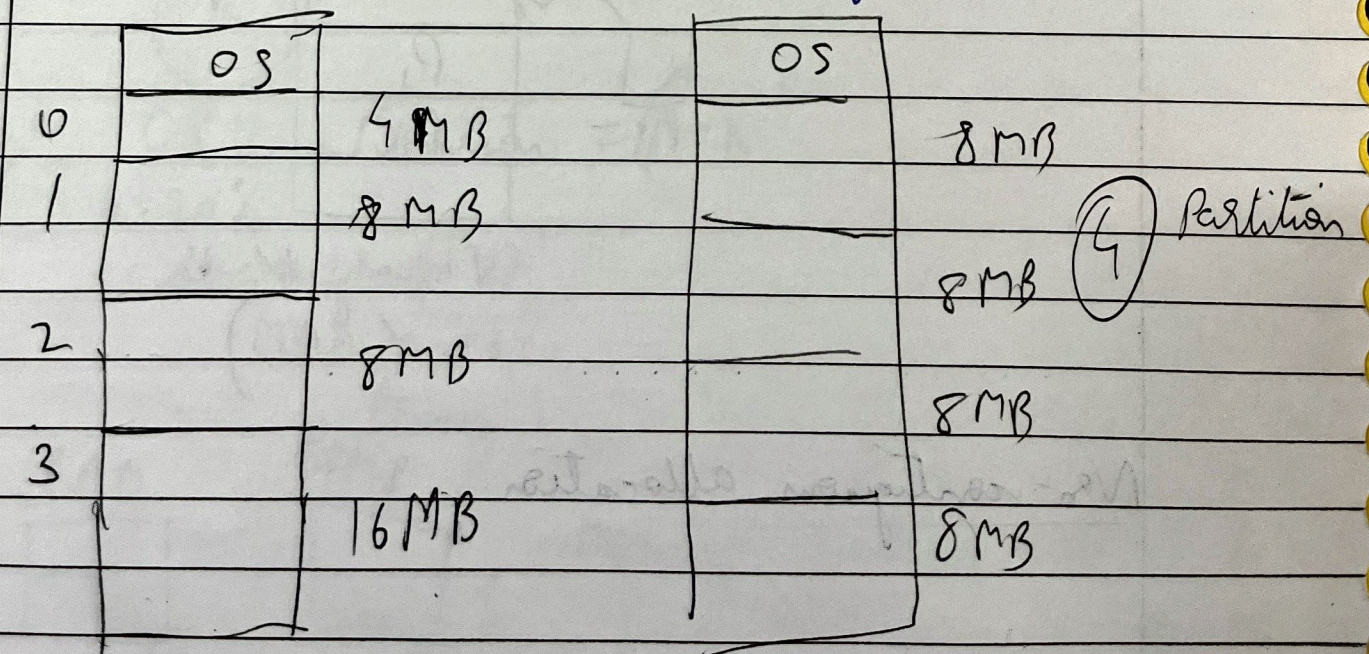
Non-contiguous allocation



L5-3 Internal Fragmentation – Fixed Size Partitioning

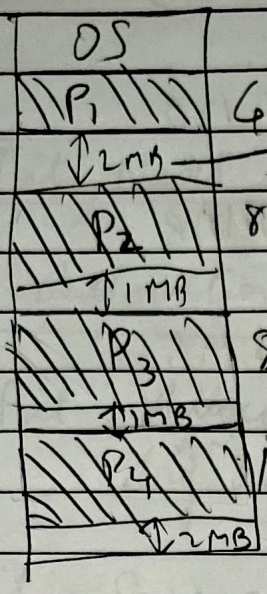
Fixed Partitioning (Static Partition)

* No. of partitions are fixed
* Size of each partition may or may not be same
* Contiguous allocation so spanning is not allowed



is

If there are 2 MB process,



→ This part is left

↓

This is called internal fragmentation

$P_2 \rightarrow 7$ MB , $P_4 \rightarrow 14$ MB

$P_3 \rightarrow 7$ MB

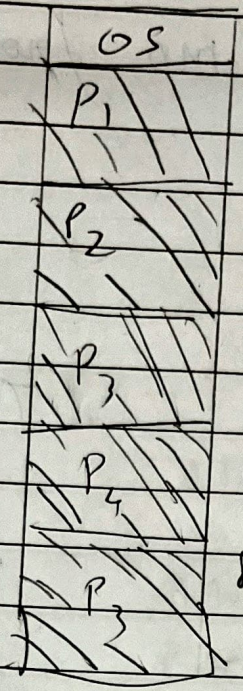
If $P_5 = 5$ MB, we cannot fit the process into any block, ~~since all the space~~ even though the leftover space is greater than or equal to the process size. ~~Therefore~~, as this is contiguous allocation. This condition is called external fragmentation

To summarise, the problems with ~~fixed size~~ contiguous memory allocation are

i) Internal fragmentation
ii) Limit in process size
iii) Limitation on degree of multiprogramming
iv) External fragmentation

L5.4 Variable size partitioning

$P_1 = 2MB$
$P_2 = 4MB$
$P_3 = 8MB$
$P_4 = 4MB$
$P_5 = 8MB$



Advantages

1) No internal fragmentation
2) No limitation on no. of processes
3) No limitation on process size
4)

Disadvantages 1) If P_2 & P_4 processes are removed, there are collectively 8 MB space left. If we want to add another process of 6 MB, we cannot add as this is contiguous allocation. This leads to external fragmentation.

- This problem can be solved through compaction, where the processes P_1, P_3, P_5 can be put in contiguous spaces through copy-paste. But, for this, we have to kill the processes

2) When holes are created, (hole means empty spaces between 2 already occupied process memories),

it is difficult to allocate/deallocate in those holes.

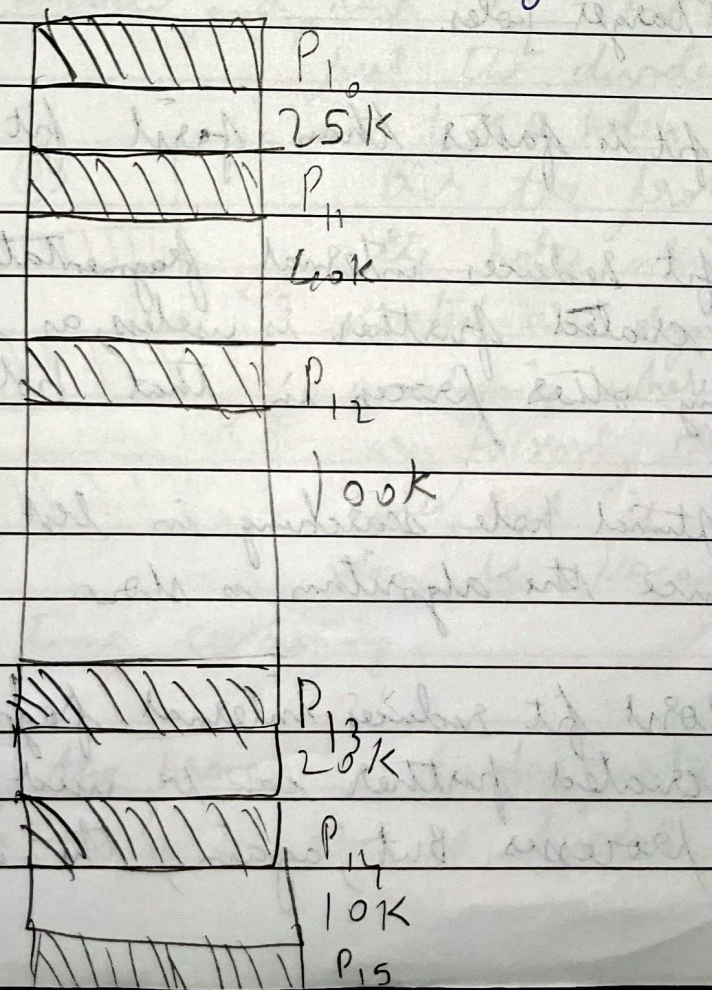
L5-5 **First Fit, Next Fit, Best Fit, Worst Fit for Memory Allocation**

First-Fit: Allocate the first hole that is big enough

Next-fit: Same as first fit start search always from last allocated file

Best-fit: Allocate the smallest hole that is big enough

Worst fit: Allocate the largest hole



Suppose $P_1 = 15K$

Through first fit, the P_1 process will be fit in to 25K hole

Next, we have $P_2 = 18K$. We are using next fit. ∴ Here, we know that P_1 was last allocated. There fore we start searching from P_1 process, and we fit P_2 in to 40K hole

If we start allocating $P_1 = 15K$ by removing P_2, then by best fit, P_1 will be allocated in 20K hole, as the difference is very small.

Through worst case, P_1 will be allocated in to 100K hole, because it has the largest space

* First fit is simple and fast, but it can create larger holes.

* Next fit is fastes than first fit.

* Best fit reduces internal fragmentation, but the holes created further is useless, as we cannot put any other process in that hole.

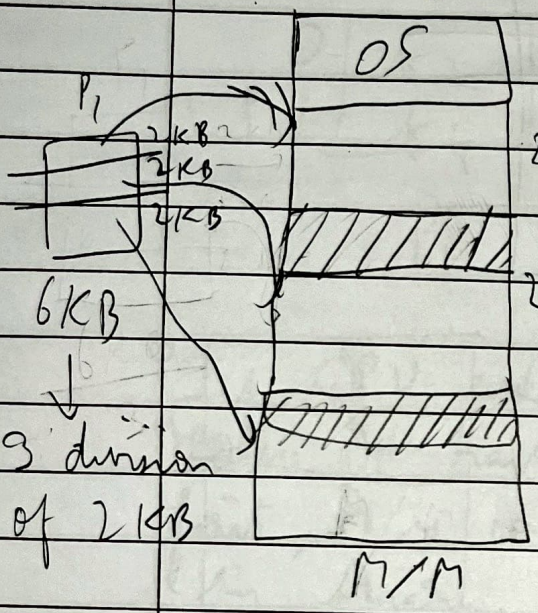
** The optimal hole searching in best fit will take time, hence the algorithm is slow

* The Worst fit reduces internal fragmentation, and holes created further can be used to allocate more processes. But, again, the algorithm is slow

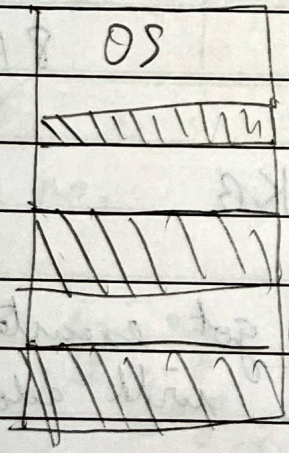
In real life, first fit is the most convenient method as it takes less time

L-5-8 Need of Paging

Non-contiguous memory allocation

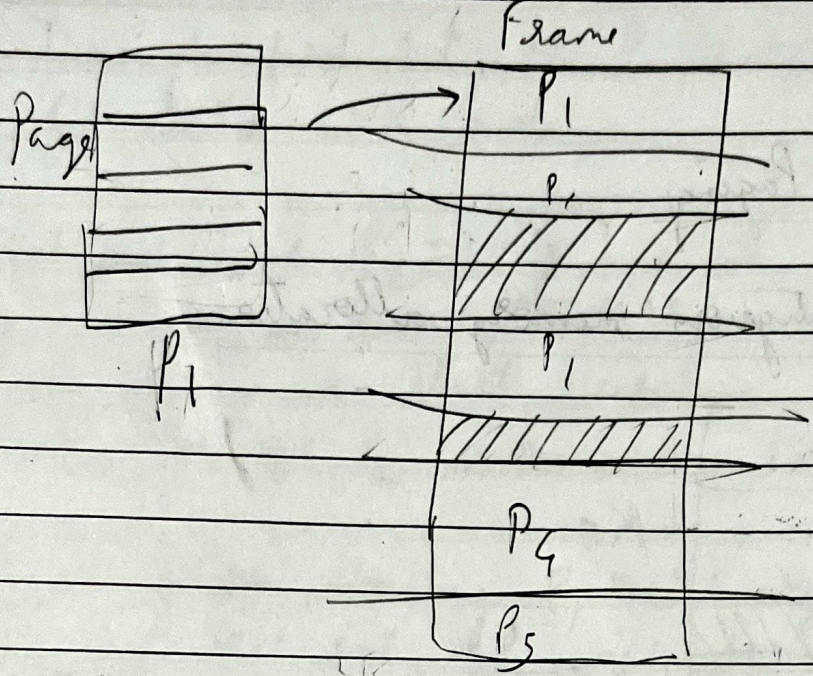


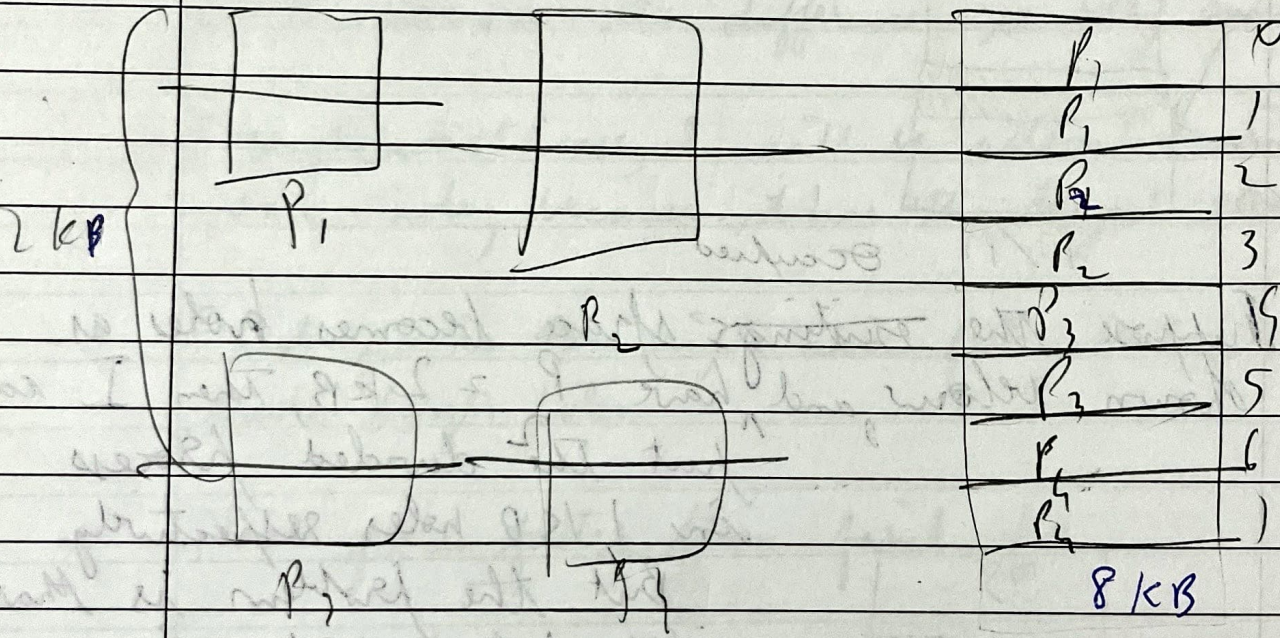
Suppose the ~~existing~~ space becomes hole as shown below, and have P = 2KB, then I can put the divided process in 1KB holes respectively. But the problem is that the holes which are created are dynamic, and analysing where the holes are present, the size of holes, and if the process can be allocated in these holes is time consuming.



Therefore, the process is divided into pages and the main memory is divided into frames

Page size = Frame size *

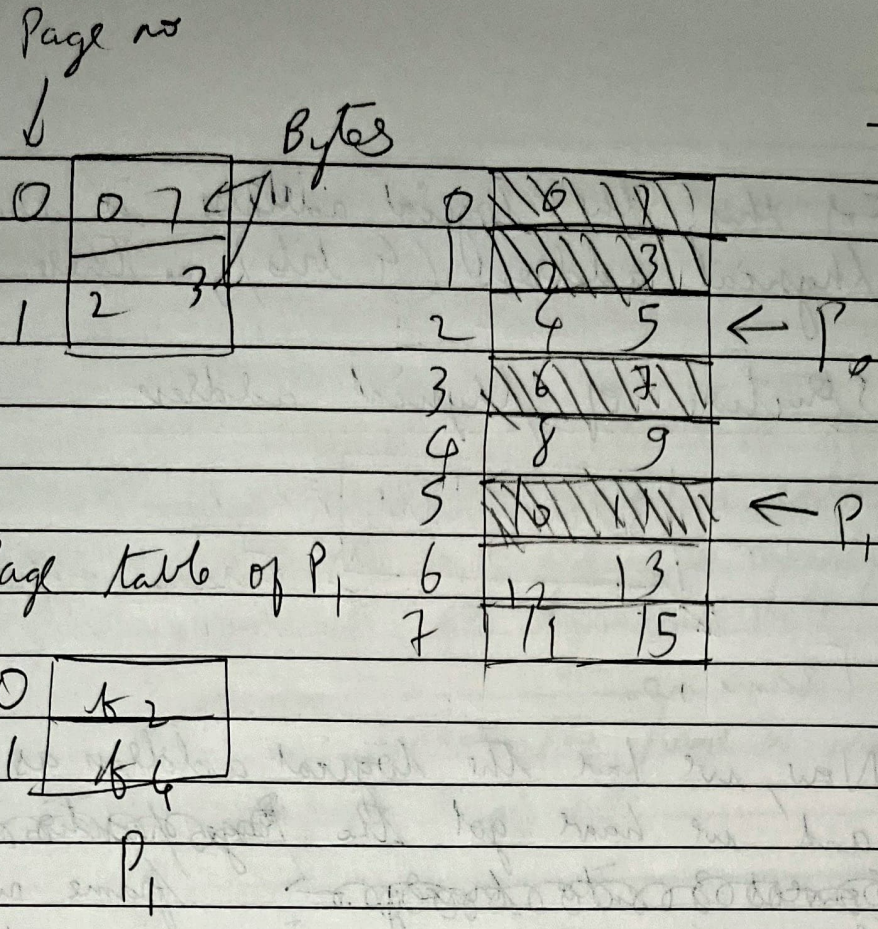




Page size = Frame size = 1KB

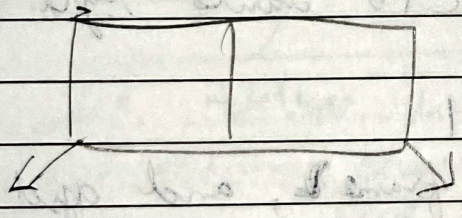
Now if P_2 & P_4 instructions got executed, there will be 4 KB space - I will add P_5 = 1KB easily to the main memory

L-9-9 What is paging?



If CPU asks for a random bit (CPU does not have the idea of paging), say 3, first, 3 is converted to logical address in binary

Structure of logical address (2 bits)

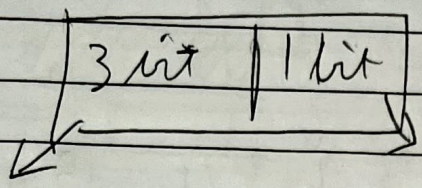


The binary of 3 is 11
∴ 1st page, and 1's bit of the page is accessed. ∴ 3 is accessed.

Now, the same byte is stored ~~as another~~ in another position of the main memory, as another location representation. How to recognise that?

For this, the logical address is converted to physical address (4 bits), as there are 16 bits.

Structure of physical address



Now, we have the logical address as 9 1, and we have got the ~~page position~~ frame number as 4. Converting this into binary gives the frame number → 100. The frame offset and the page offset remains the same. The resultant physical address is 1001 → $9_{(10)}$.

* The page table is stored and managed by memory management unit (MMU)

Another example :- CPU wants byte 1

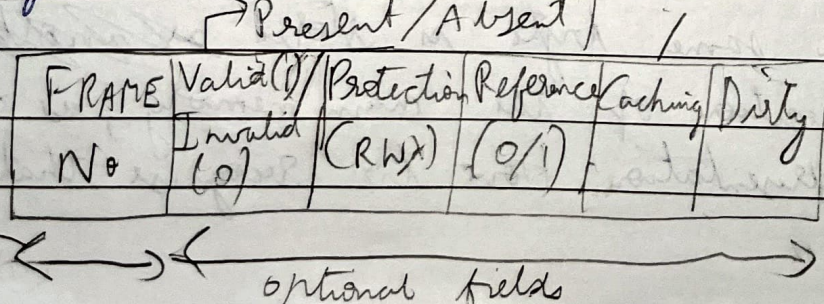
Logical address :- 01

1 is present in frame 2, and offset is 1 the physical address is 0101, which is $5_{(10)}$. ∴ The ~~bit representation~~ byte is in position 5 in the main memory.

LRU → Least recently used

Enable / Disable

L-5-12 Page table entries



R - Read
W - Write
X - Execute

Caching is done to store the data in the cache memory, if that particular data is called multiple number of times. Caching is mainly for static data, and not for dynamic data.

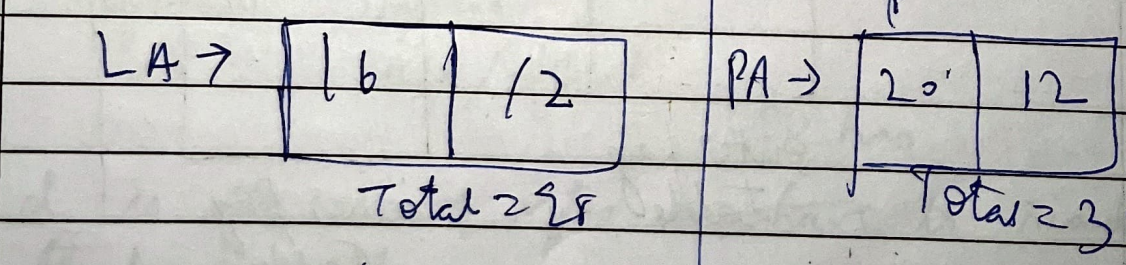
Dirty bit tells us whether the page is modified or not, 1 if modified, 0 if not.

6-5-13 **2-Level Paging in OS**

If the size of the page table is greater than the size of main memory, we have to divide the page table

Eg:-

Physical Address Space = 256 MB → 2^{28} B
Logical Address space = 4GB → 2^{32} B
Frame size = 4KB
Page Table Entry = 2B



Page table size = $2B \times 2^{20} = 2MB$

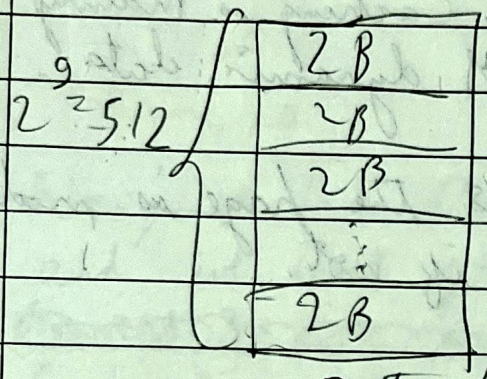
This is the inner page table

inner

Now we have to divide the page table into divisions

No. of divisions = $\frac{2^{32}\text{B}}{4\text{KB}} = \frac{2^{32}}{2^{12}} = 2^{20}$

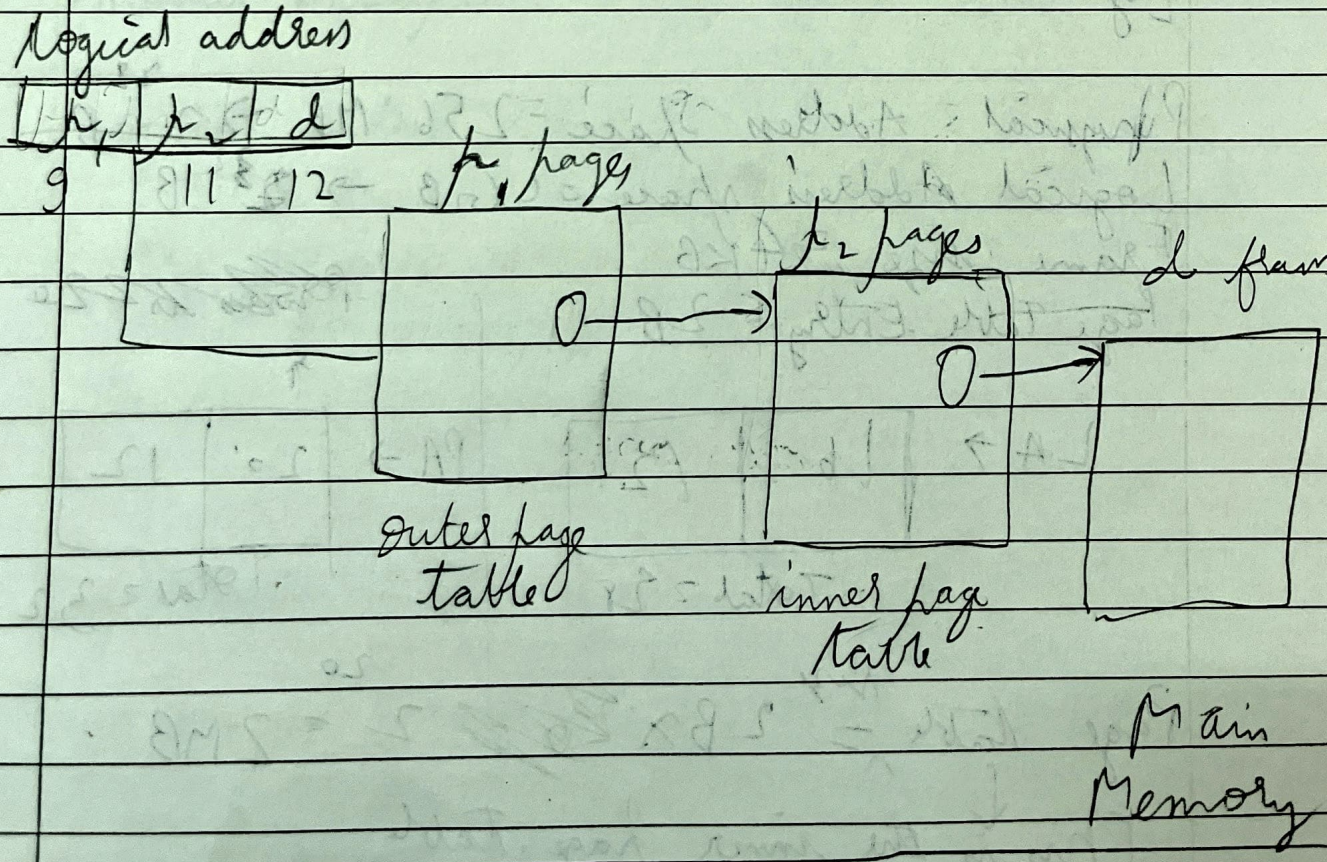
Now, we create an outer page table



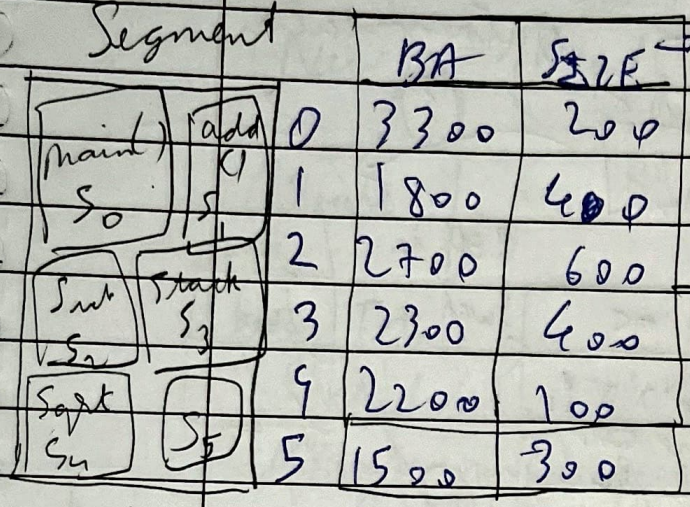
∴ The total outer page table size = $2^9 \times 2 = 1$KB

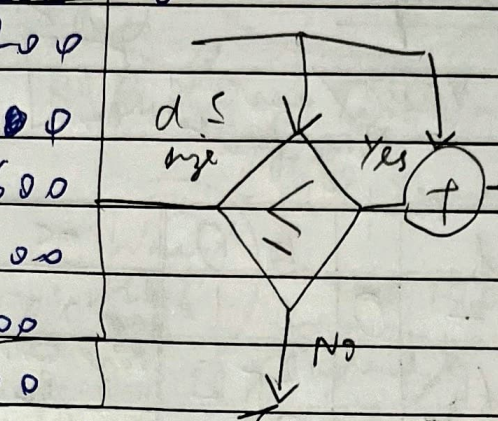
~~Now how do we~~

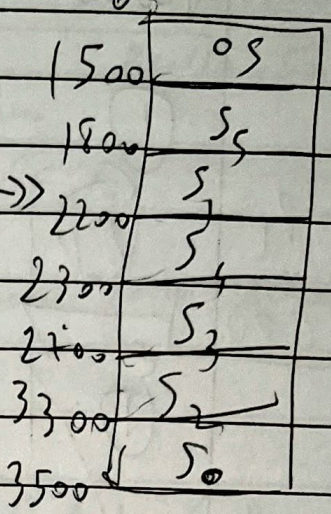
Address translation scheme



L-5-17 **Segmentation VS Paging**



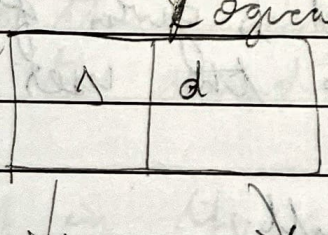




→ All are of various sizes, unlike pages

* Just like paging, logical address tells the CPU the location of the byte in the process, and physical address tells the CPU the location of byte in the main memory.

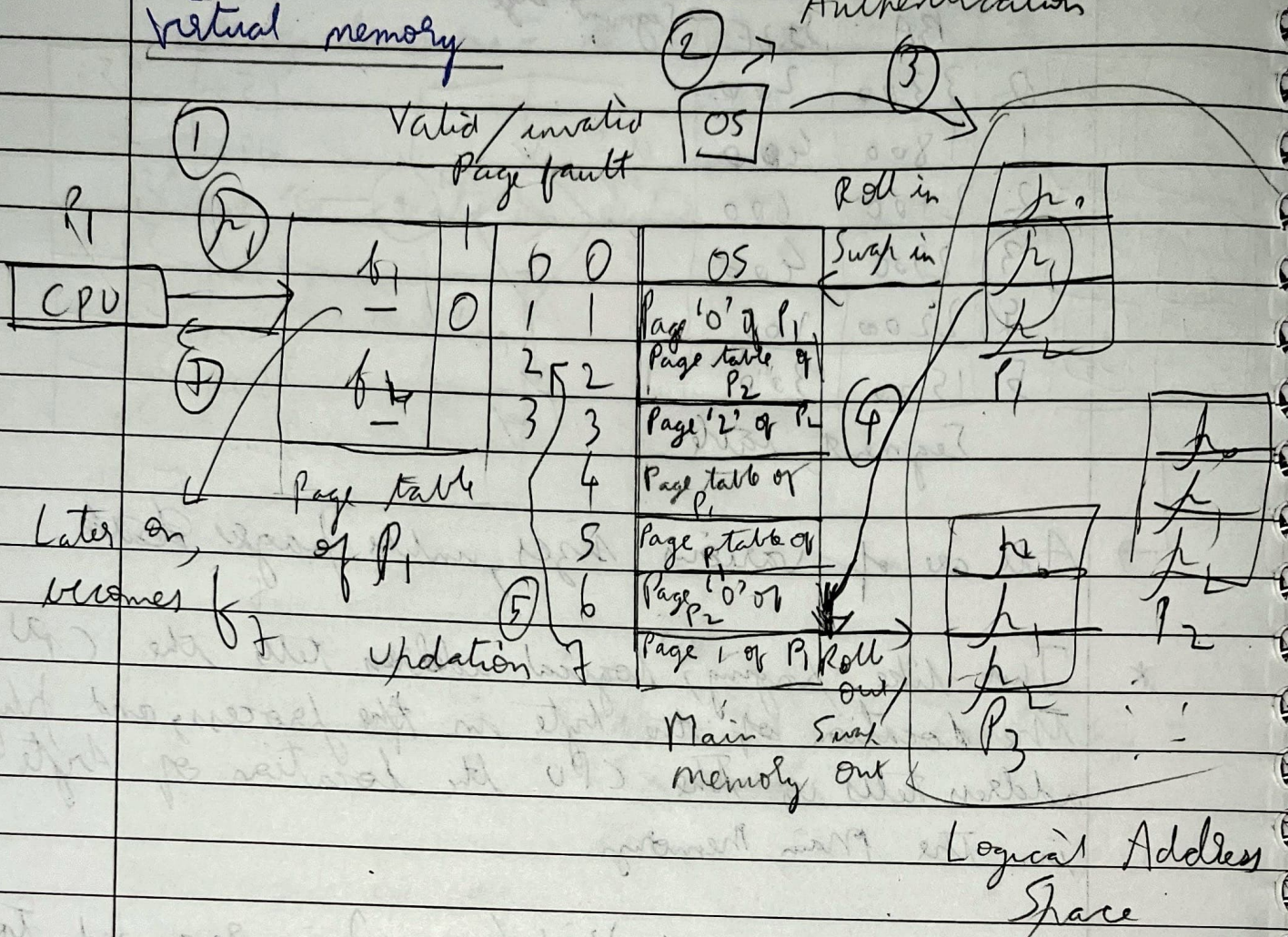
* Memory Management Unit (MMU) is required to convert from logical address to physical address.



- CPU has

d is the no of bytes ~~to~~ have to read in that segment

L-5.19 **Virtual Memory - Page fault - Significance of virtual memory**



If page fault is 0, Trap is generated

Through this method control is given from user to OS, and then to the user

$h \rightarrow$ Probability of page fault.

Effective memory access time, EMAT =

h (page fault service time) + $(1-h)$ (main memory access time)

(in ns) (in ns)

* We are not considering the main memory access time during page fault service time.