logical address spa

Paging (lect #09)

Whenever a process is created, it is divided into pages. When the process has to be executed, the whole process will be loaded into the memory. If the process requires n pages then there needs to be n frames available in memory that would be allocated to the process.
When process is loaded in the memory, the base address of frame of each page is added in the page table.

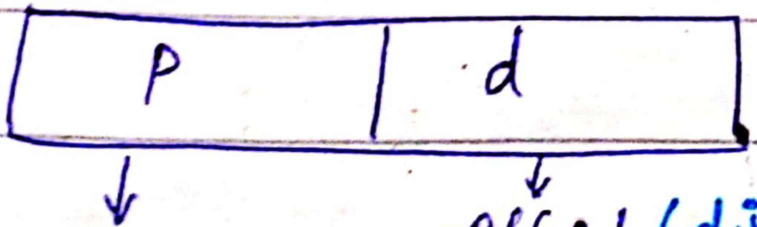
When you generates a logical address then it is translated into the physical address by MMU.

Scanned with CamScanner

→ Pages are of fixed size and each page ~~table~~ has associated frame number.

→ Paging has internal fragmantation.

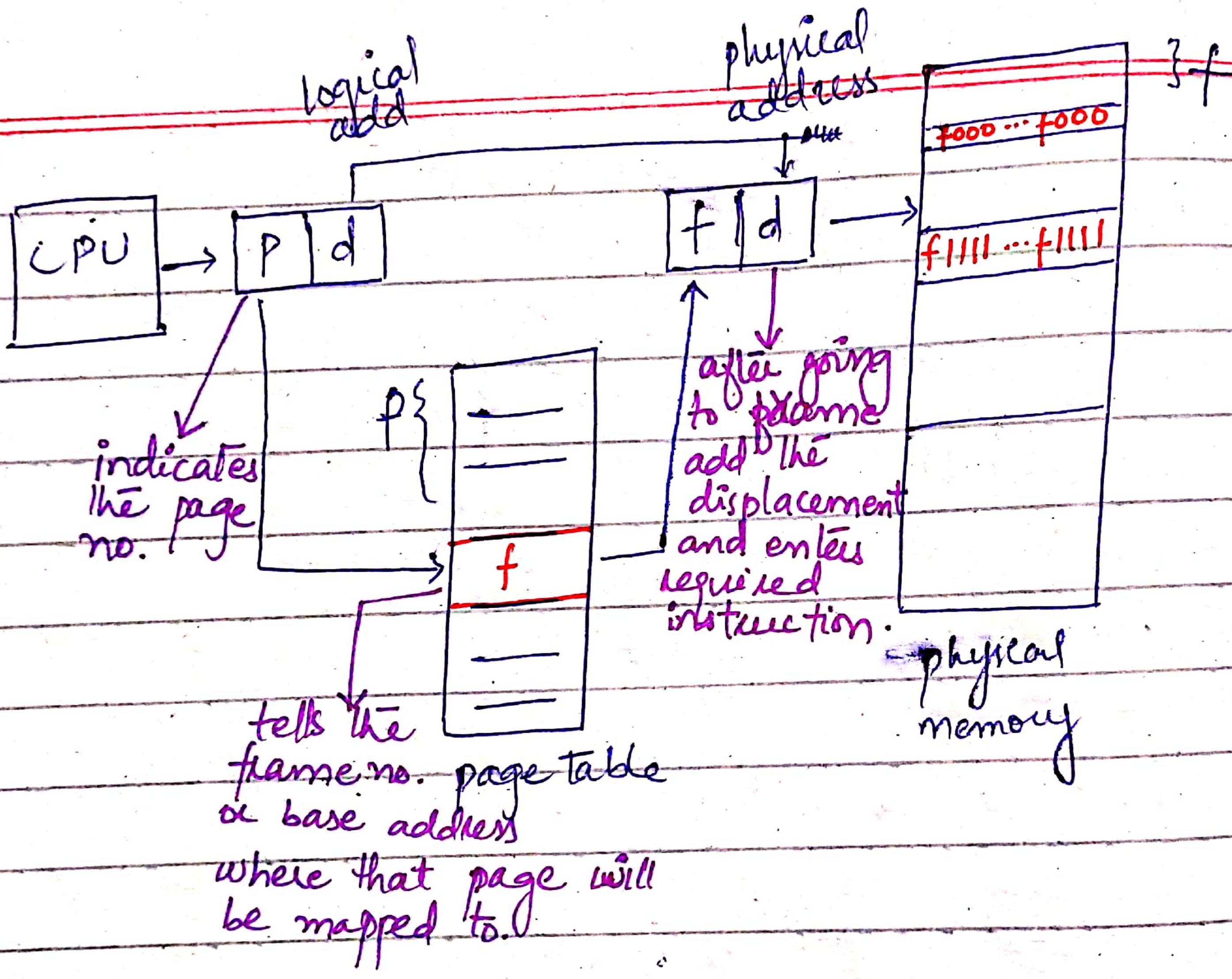
CPU generates the logical address, which is divided into:



page no. (index of page table which tells the frame (base add) where that page will be mapped)

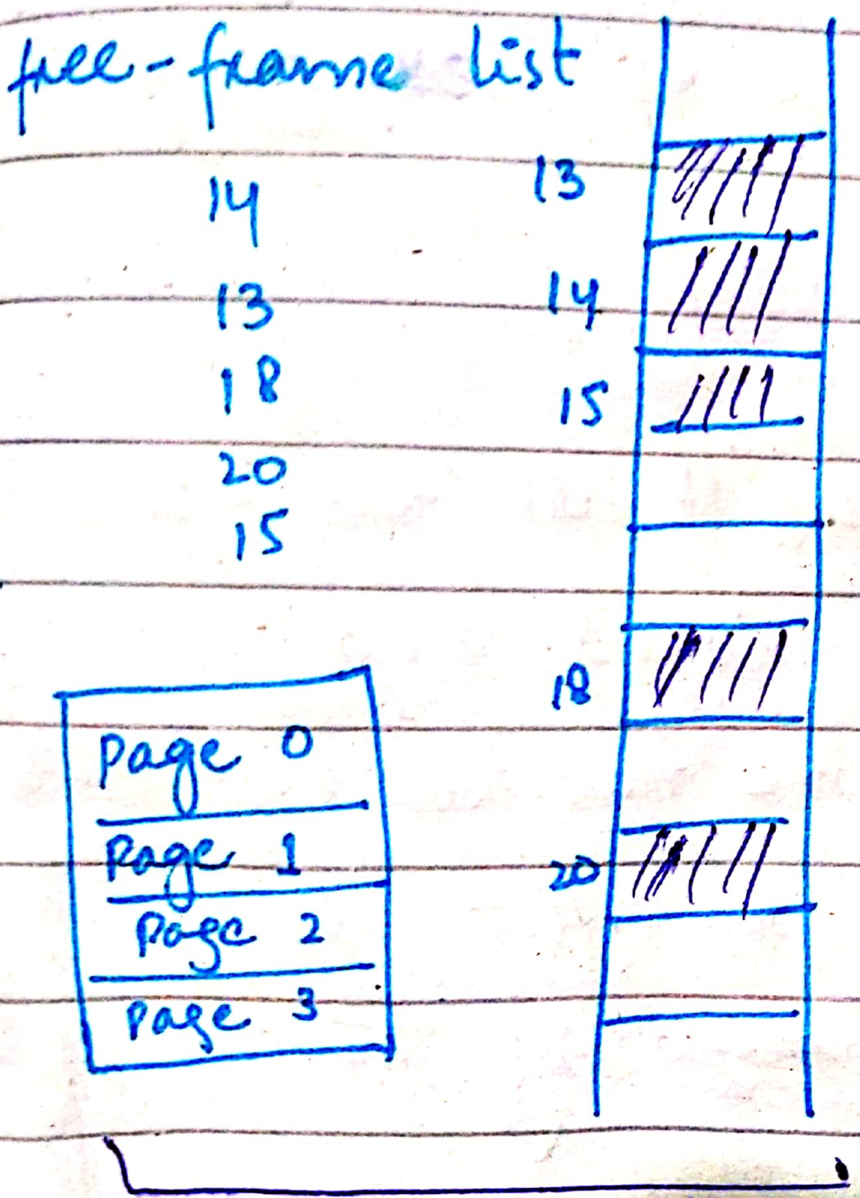
offset (displacement within page/frame, added to base address)

Scanned with CamScanner

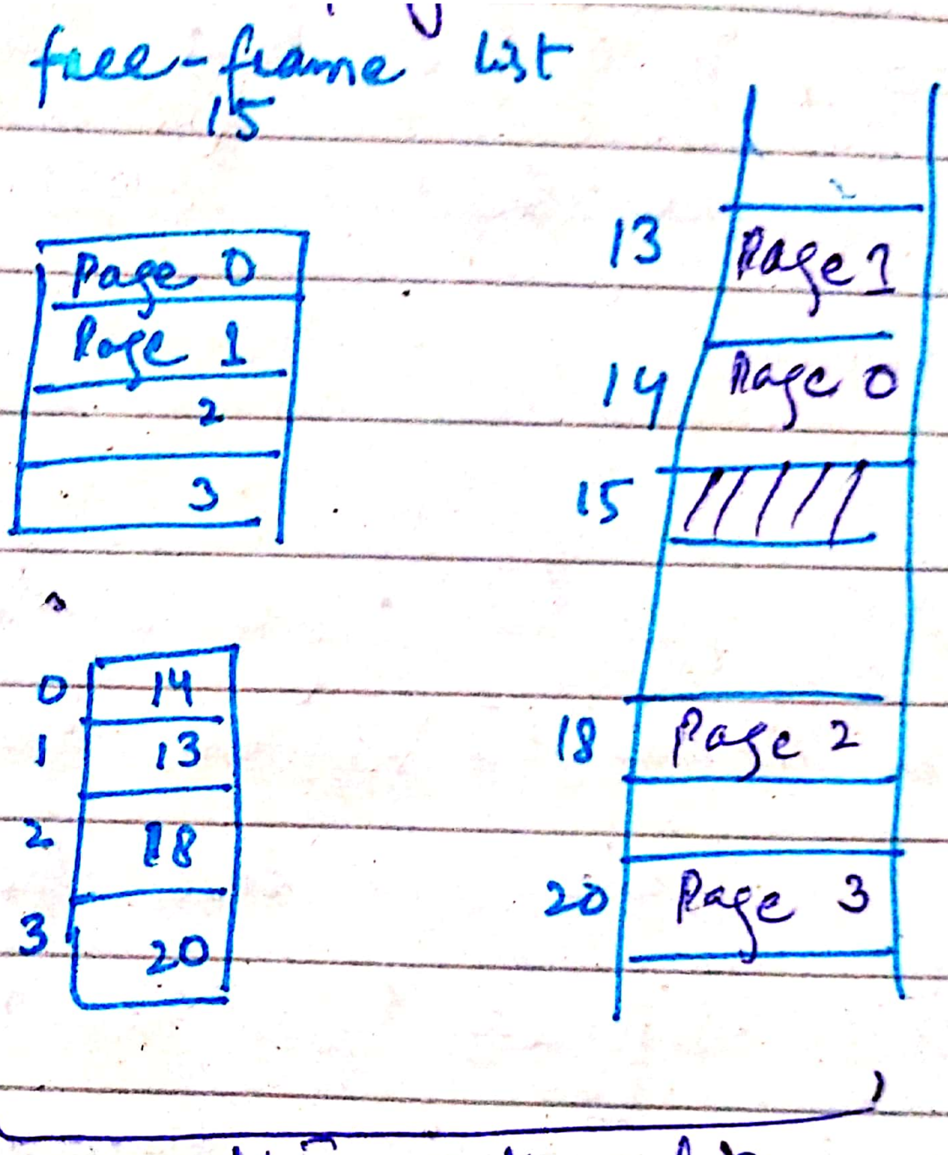


logical add
physical address
CPU
p
d
f
d
p{
f
f000 ··· f000
f1111 ··· f1111
}f
indicates the page no.
tells the frame no. or base address where that page will be mapped to.
page table
after going to frame add the displacement and enters required instruction.
physical memory

eg: $p = 1$ $\underbrace{f = 1000}_{\text{in page table}}$ $d = 15$ now go to 1015 in memory.



Befoe allocation



after allocation.

Scanned with CamScanner

There is a free ~~page table~~ frame list available to show which memory frames are available and when the pages of process are loaded into memory, these frames are removed from free list.

Scanned with CamScanner

During paging, for each process a page table is created which stores all the possible page number present in the logical address space. For each entry, the frame number is mentioned. If that page is not in process's logical address space then 0 is mentioned in place of frame no.

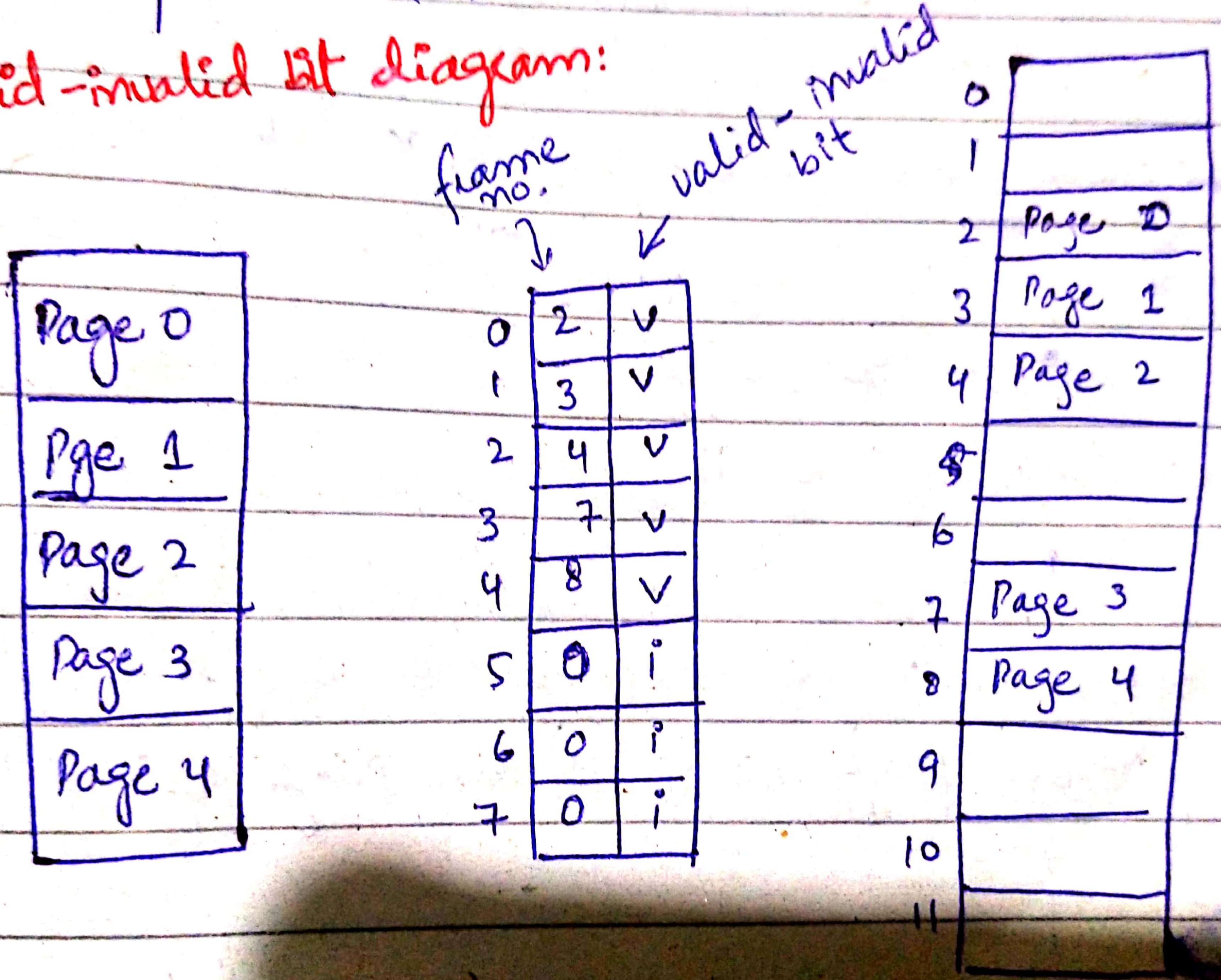
There is a valid-invalid bit written next to each page. If a ~~bit~~ page exists in process's logical address space then bit is set to valid otherwise bit is set to invalid.

If CPU generates an address for those pages whose bit is invalid then a trap is generated. In this way, CPU can not access the wrong address space.

Example: in a system with a 14-bit address space (0 to 16383), there is a program that holds (0 to 10468) addresses. Each page is of 2KB. This means process can access page 0, 1, 2, 3, 4, 5 but the page 6 and 7 are not in process address space thus it can not be accessed.

Scanned with CamScanner

valid-invalid bit diagram:



Scanned with CamScanner