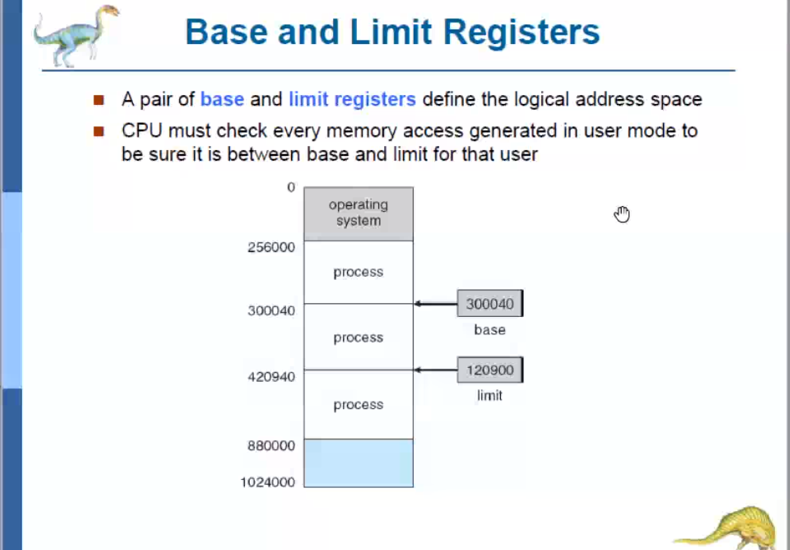
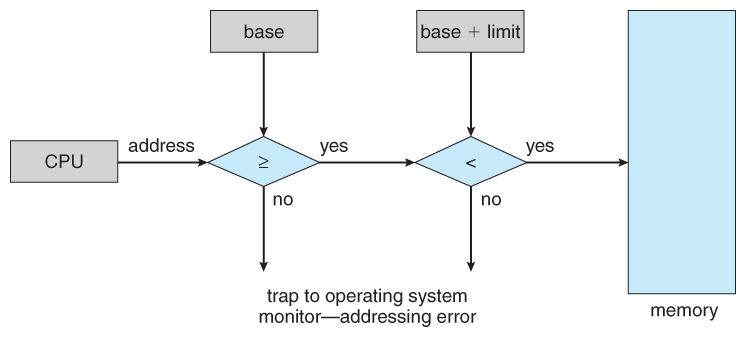
Processor Cache memory Main memory

Cache memory 🡪 Recently and repeatedly doing task are stored for the faster retrieval purpose.

**Base and Limit Registers**



Base register 🡪 Will contain the starting address of the process.  
Limit register 🡪 Will contain the process’s size.



Where the base and limit register of a particular process is present??  
Whether in PCB of that particular process???

**When the CPU wants to fetch a particular process by logical addressing, How it will correctly ensure which process must be taken in-order to execute???**

The address generated by the CPU must be >= base address and < base + limit.  
If < then some-other process not wanted by the CPU will come into action.

Logical address 🡪 It is the address which is envisioned by the programmer. But that is not the exact thing which is happening at the background.

Eg: A programmer writes a program and it occupies 100bytes. After compiling, it is in RAM and what the programmer thinks is that the CPU generates the address for the code to start executing from 1,2,3..100 bytes, **but** the compiled code is not present exactly in the way with the same physical address (1,2,3,….100 bytes) the programmer thought in the main memory.   
 

**Why the physical memory address is not visible/accessed by the user???**  
In-order to avoid confusion, if the user unknowingly access the OS’s portion / some other important process in the physical memory. It will become a total chaos and it is difficult to retrieve back.

Now our task is how to manage/link between the logical address(programmer envisioned) with the physical address(how the code is stored exactly in the main-memory) ????

**Compiling** : While compiling the source file, an object code will be generated. That time the linking can take place.   
 Loading

**Loading:**After compiling Object code 🡪 executable file 🡪 Brought into main memory

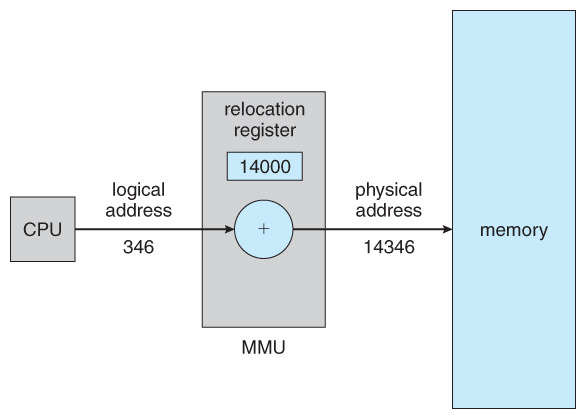


Linking

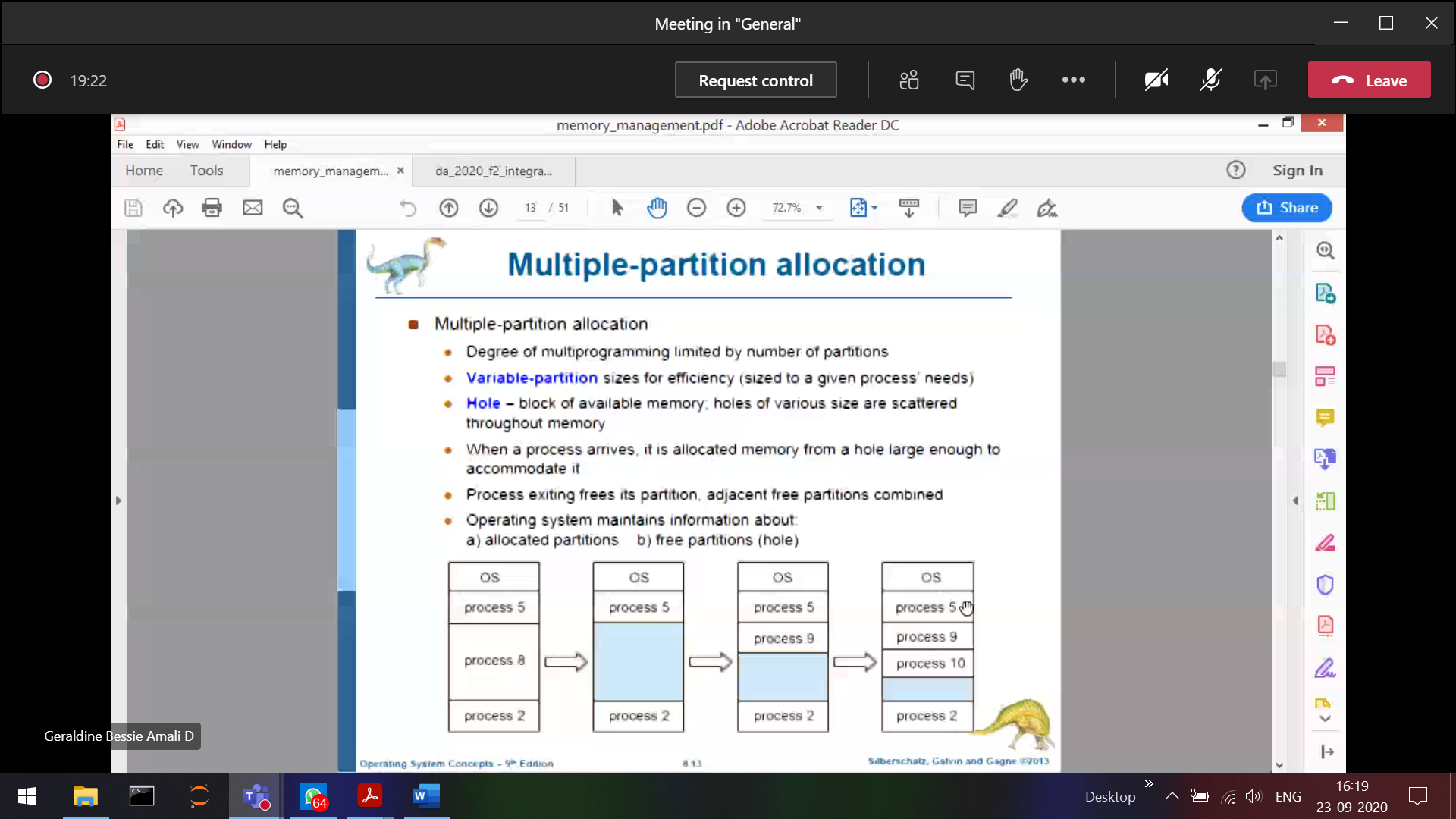
**Execution time:**Linking takes place during run time. Dynamic memory allocation. That time the linking can take place.

That linking takes place in Memory management unit

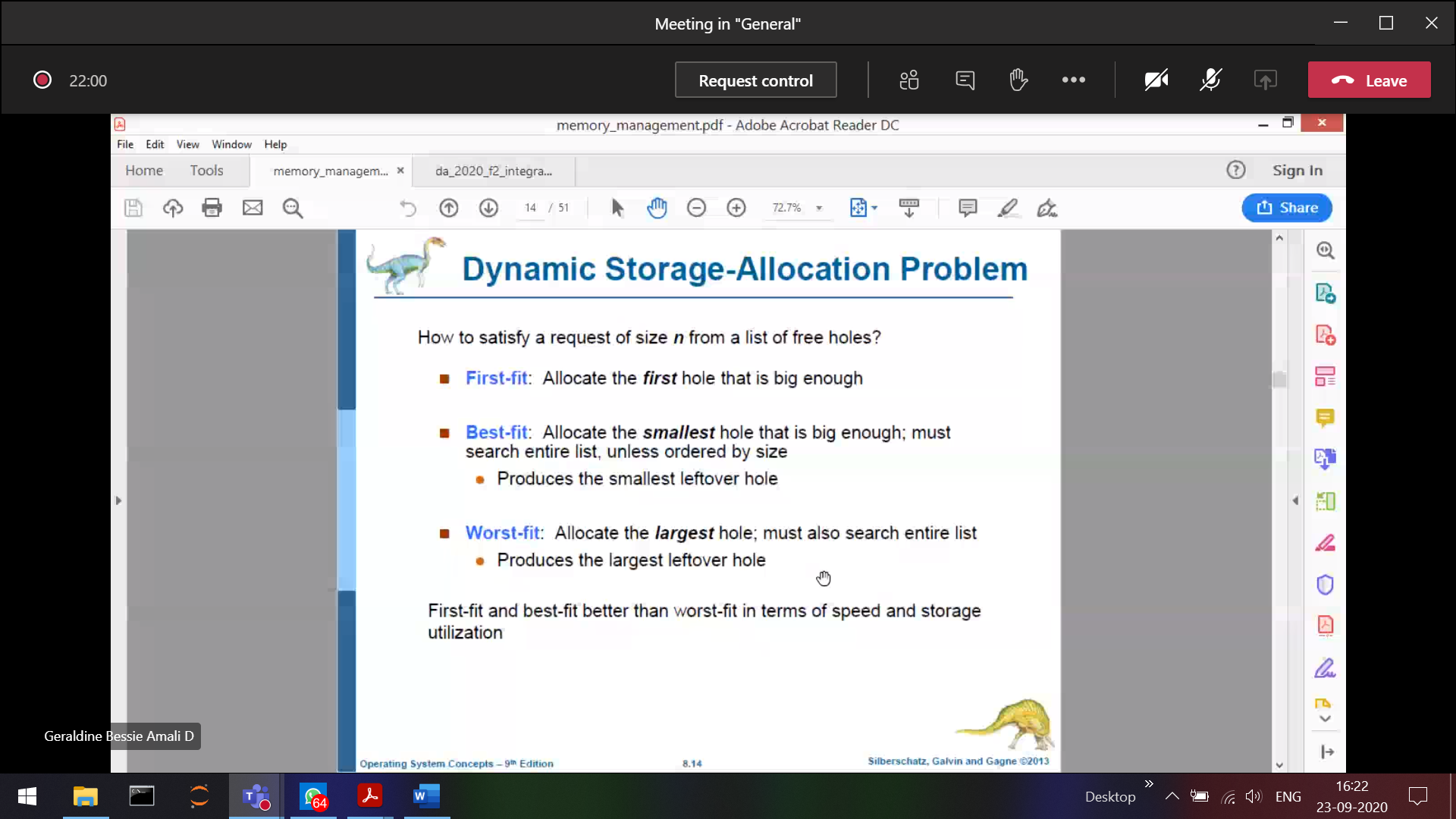
relocation register = base register



CPU wants the 346th line of the code(i.e instruction). This 346(logical address) will be added to the 14000 relocation register(base address register) and 14346 is the exact place where that instruction is stored and this instruction is fetched by the main-memory.



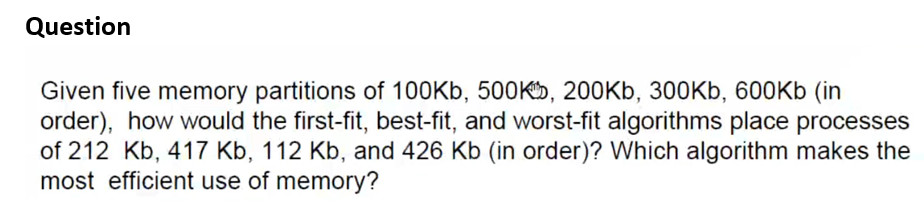
Since the process’s are of different size. So the hole size is also different.   
OS should decide in which hole the upcoming process can be allocated.  
In-order to achieve this, there are 3 different algorithms.



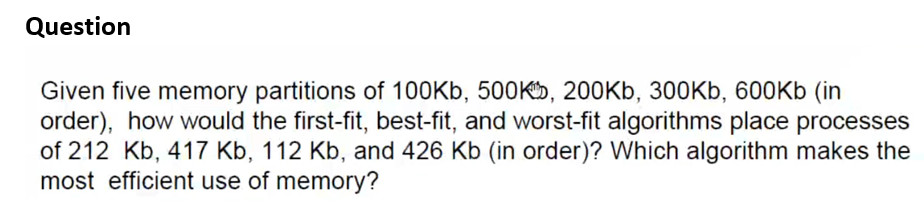
**First fit 🡪** When we scan the RAM, we can find many holes. Among that the first hole is taken into consideration. That first hole can be large/small.  
P1 comes and resides, While P1 residing there may be some small space left out or some large space left out 🡪 Uncertain for the other process to reside in the left over place.

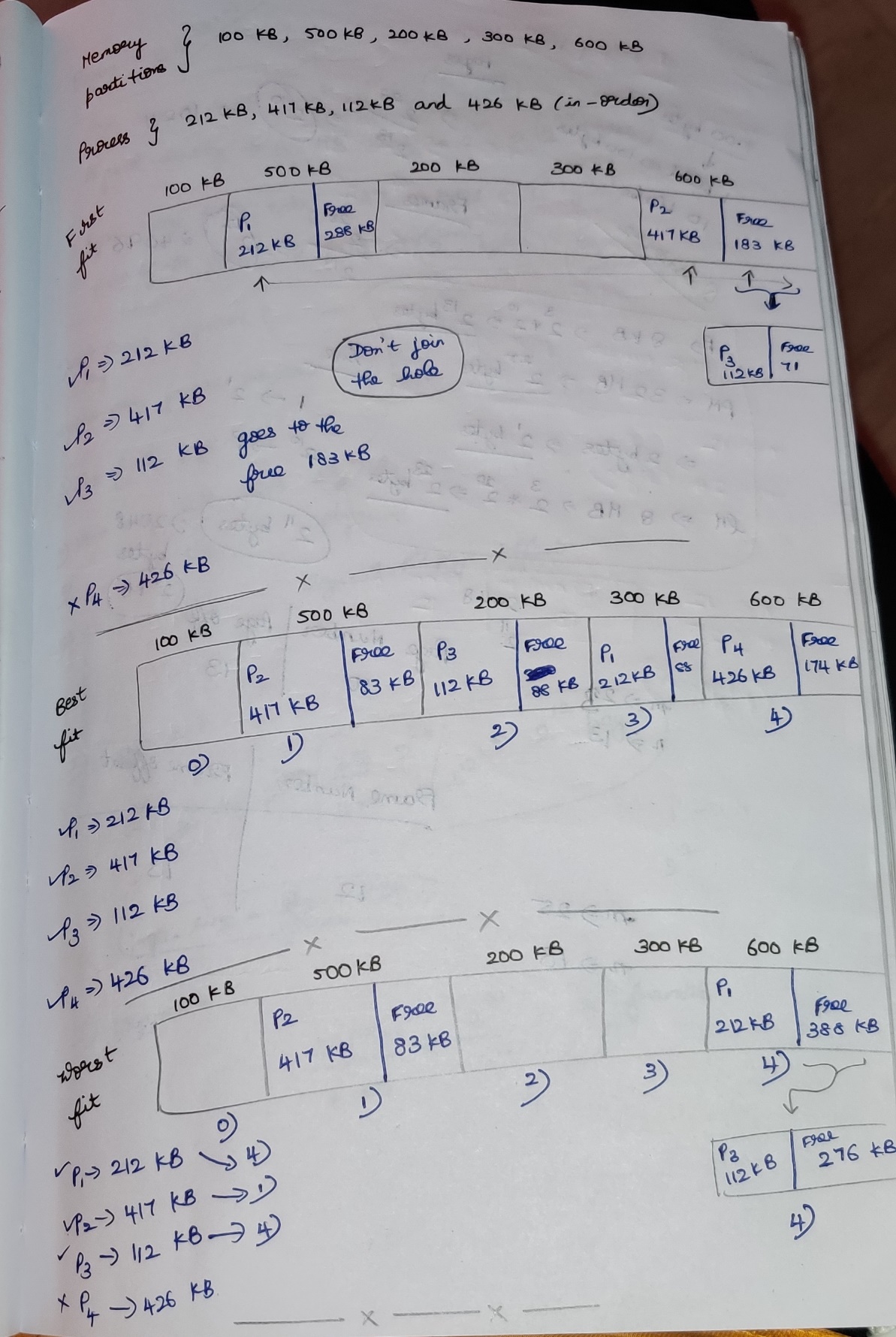
**Best fit 🡪** Will compare all the holes in the RAM, it will look for the best match which is large enough to accommodate that process.   
It will scan and find-out the best match 🡪 Takes more time.

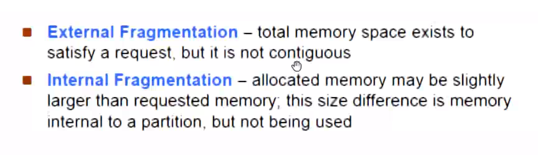
**Worst fit 🡪**  Will look for the largest hole to fit the process.  
P1 comes in and resides. Mean-while since worst fit has the largest space for P1 after P1 resides also there are some good enough space for other process to reside.



In first fit and in worst fit, P3 🡪 426KB was not having the space to enter into RAM, even-though as a whole there are some enough spaces to occupy P4 due to the partition of RAM P4 cannot able to occupy. 🡪 Fragmentation



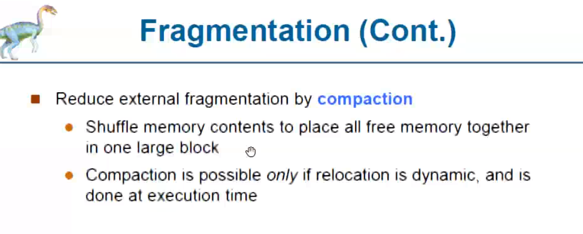




External fragmentation 🡪 Even-though the memory space exists, the process P3 cannot able to enter since the blocks are not continuous and scattered everywhere.

External fragmentation can be avoided by using compaction.

Defragmentation/Compaction 🡪 Collect all the wholes and push to one side, so that the whole size becomes bigger. Now P3 can easily enter into RAM.



During Compaction, the processes are moved from their original state in physical memory to another in-order to leave a big hole for the big process to come.   
As the process’s are moved the base address of the process should be kept a track

**Non-contiguous memory allocation**



Paging 🡪 linking the logical (pages) to physical (frames) to avoid internal fragmentation.



Pages 🡪 Dividing the logical memory into fixed size blocks  
Frames 🡪 Diving the physical memory into fixed size blocks



Page size and frame size are 🡪 equal



Eg: Size of Physical memory 🡪 1000 bytes  
1 block of logical memory 🡪 10 bytes. How many logical memory blocks are present????

Logical memory 🡪 10 blocks(pages). Since page size and frame size are equal, physical memory 🡪10 blocks(frames)

If we want to run a program/process of size N pages then there must be in N frames, These N frames are not continuous in RAM. Where-ever there are free frames the process will be allocated. This prevents external fragmentation.



Avoids varying size memory blocks.

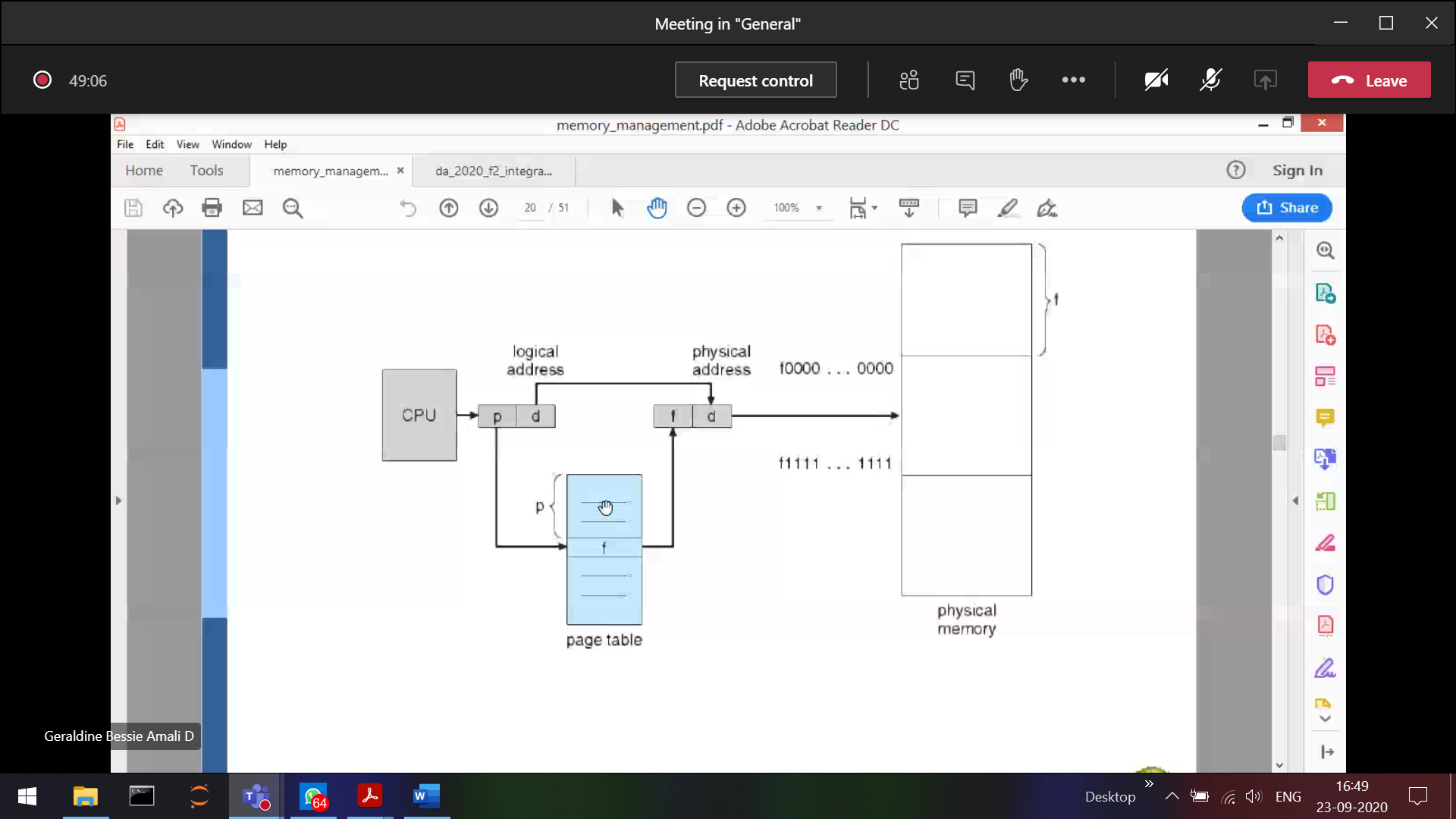


**Will the base register and limit register is enough to translate the logical address to physical address??? 🡪 NO**In continuous memory allocation, base and limit registers are used since we can easily find a process. Base + limit address



Non-contiguous memory allocation 🡪 Page table





Protection:

Contiguous memory allocation 🡪 Base and limit register  
Non-contiguous memory allocation 🡪 Page table

Page table (logical to physical address conversion)   
Let say there are 4 pages  
P0 🡪 base address of F15  
P1 🡪 base address of F12  
P2 🡪 base address of F59  
P3 🡪 base address of F45

Paging leads to internal fragmentation  
Non-contiguous memory allocation 🡪 External fragmentation

Eg: Suppose if there are 1000 bytes page, and the   
process requires 🡪999 bytes (1 byte will be wasted) [ best-case ]  
process requires 🡪990 bytes (100 bytes will be wasted) [worst-case]

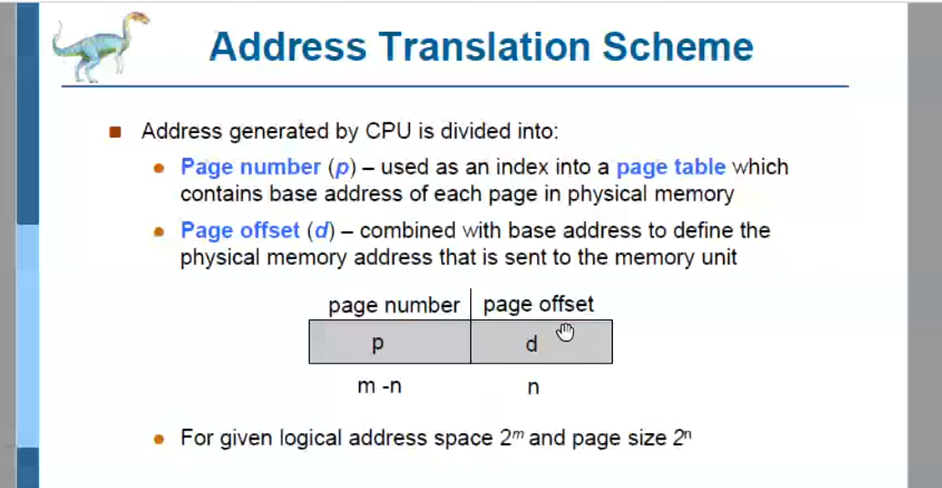


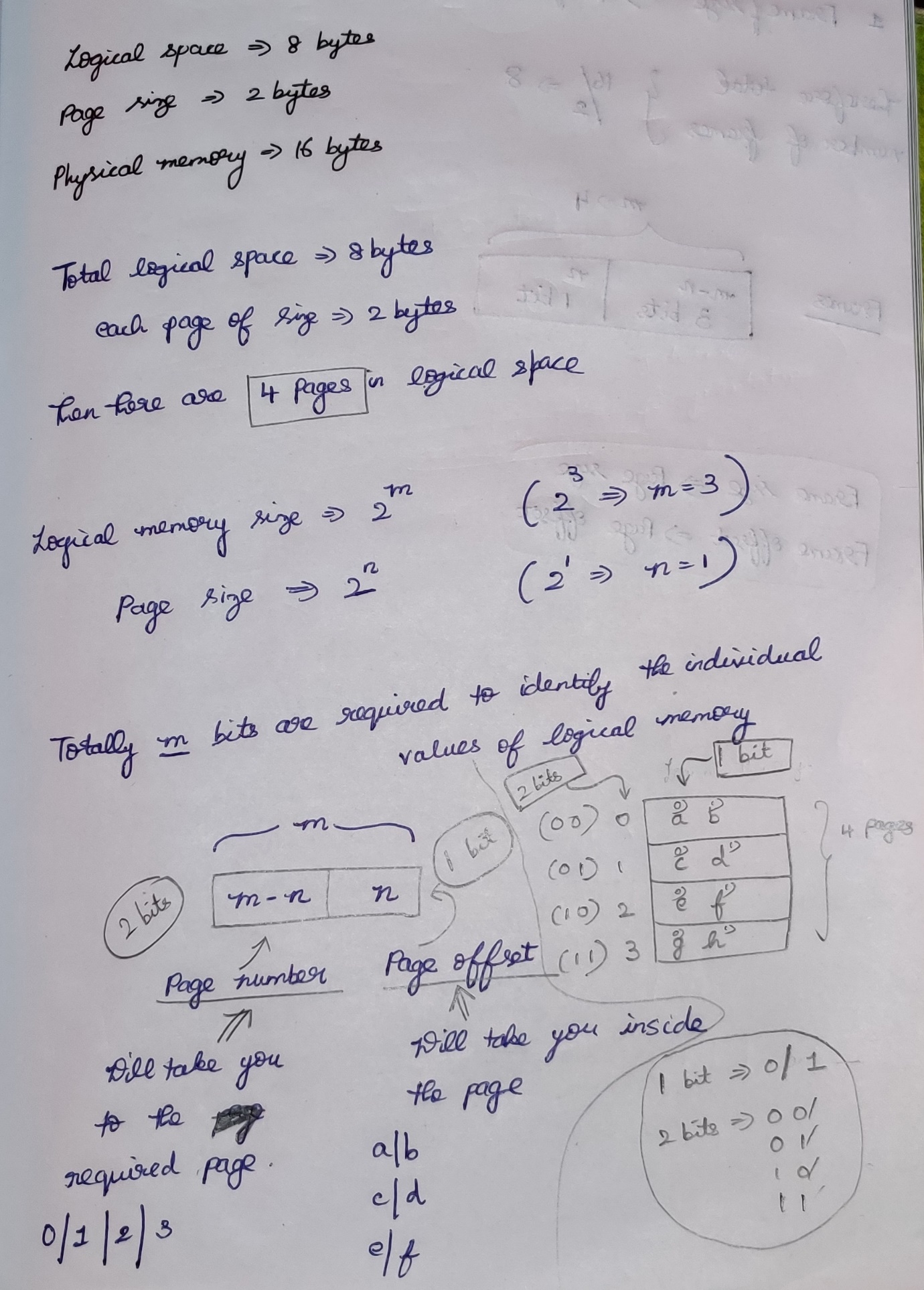
Since the pages and frame block are of fixed size.

Internal fragmentation 🡪 Allocated memory might slightly larger than the requested memory. The size difference is internal to the partition, but not being be used.

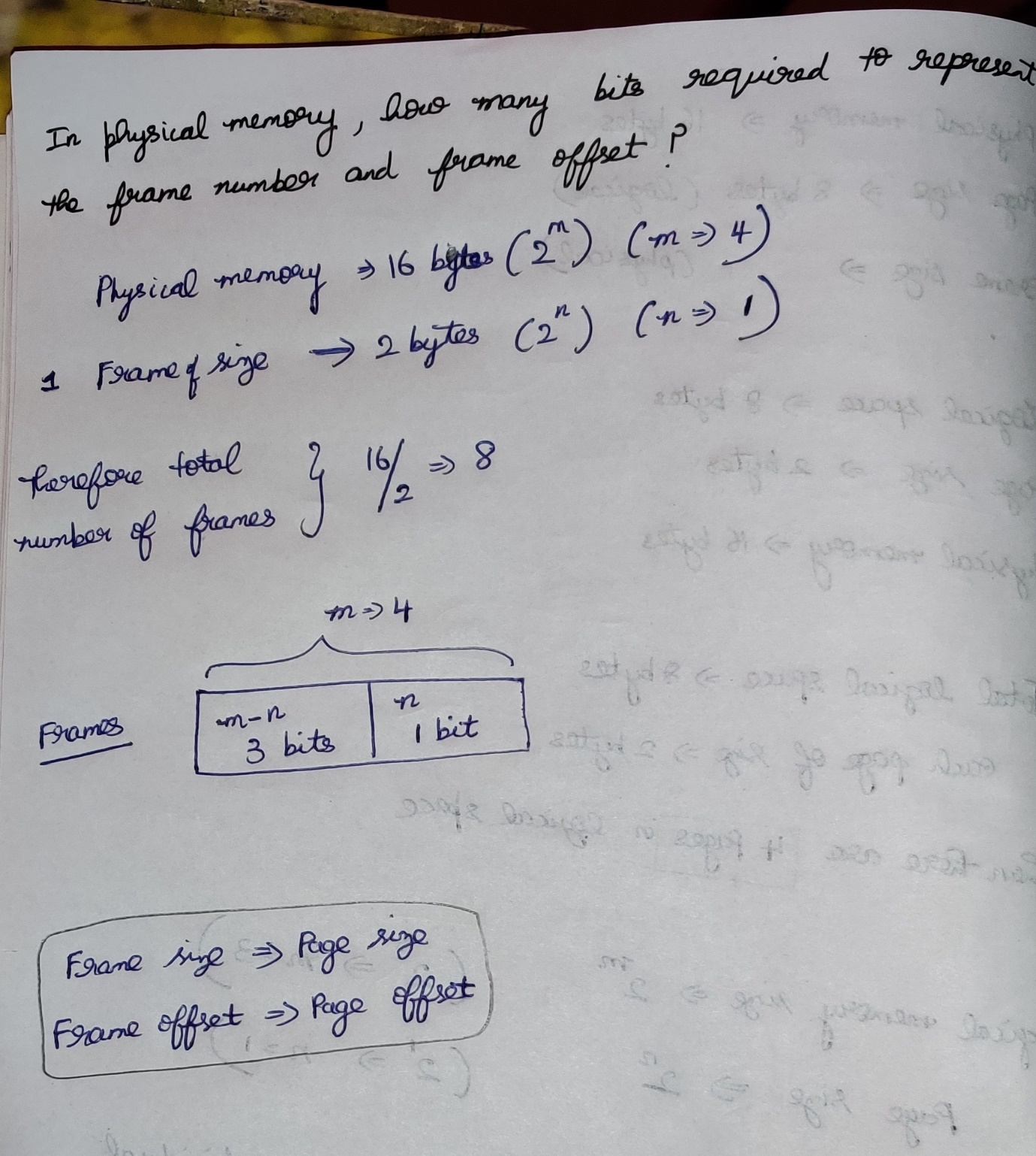
In External fragmentation, empty holes can be pushed together as a big hole but in internal fragmentation It is Within the process, so waste spaces cannot be reused again.

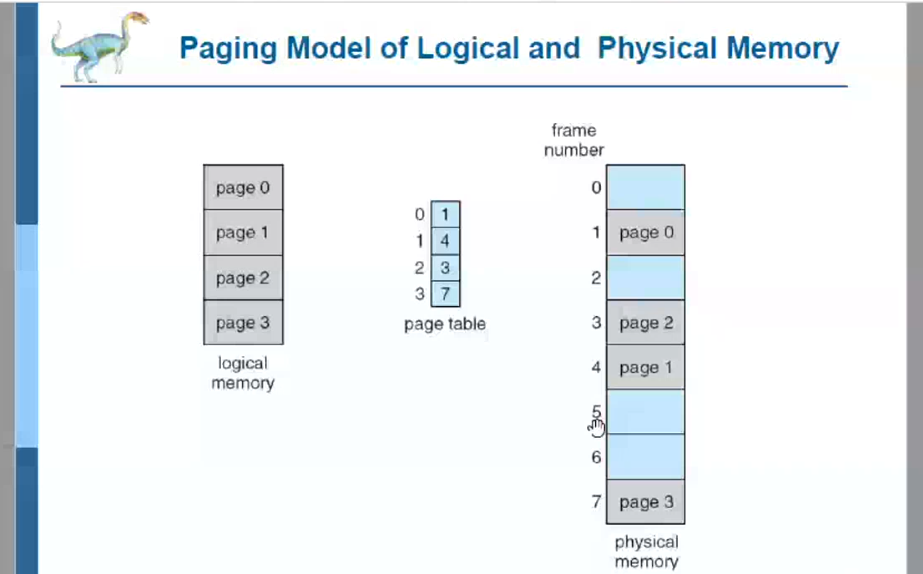
Size(i.e page size and frame size) should always be in the power of 2

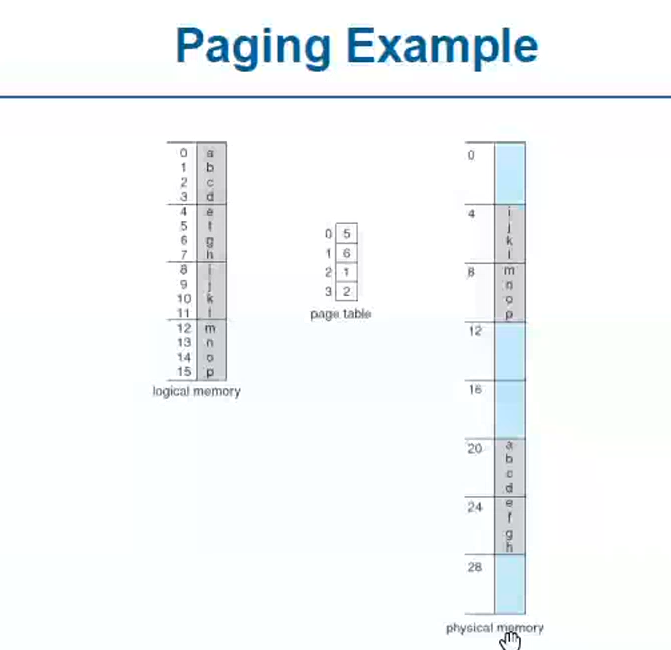










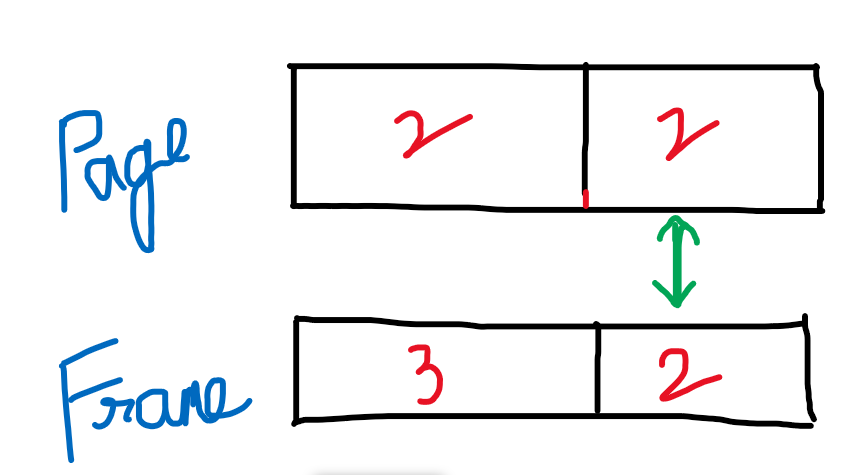




4 pages

2^4 🡪 16 bytes (m=4) 2^5 🡪 32 bytes (m=5)  
 ( 0 to 15) ( 0 to 31 )

1 Page 🡪 4 bytes (2^n where n=2) . Total 🡪 4 pages



Page table base register 🡪 to point to the

Page table length register 🡪