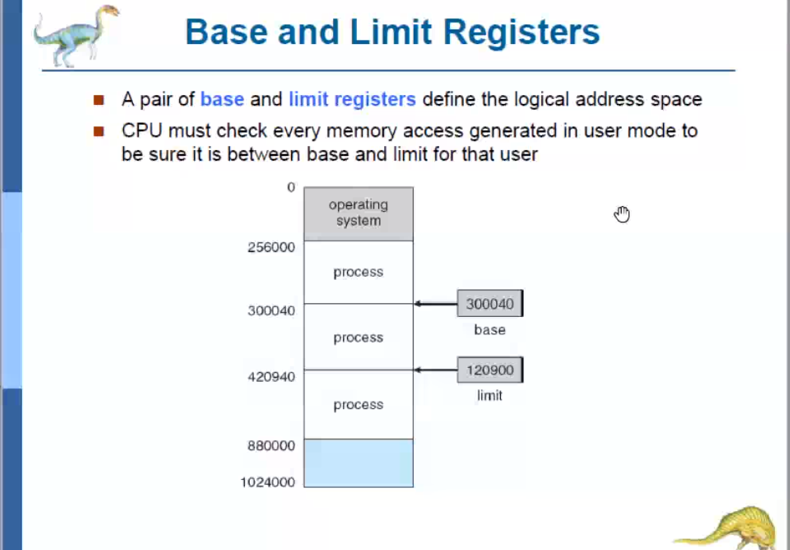
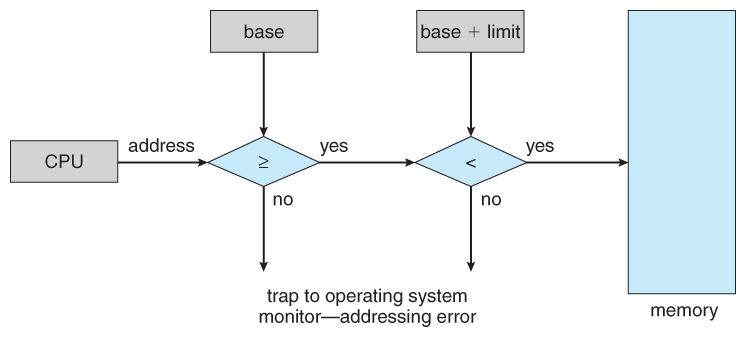
Processor Cache memory Main memory

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.

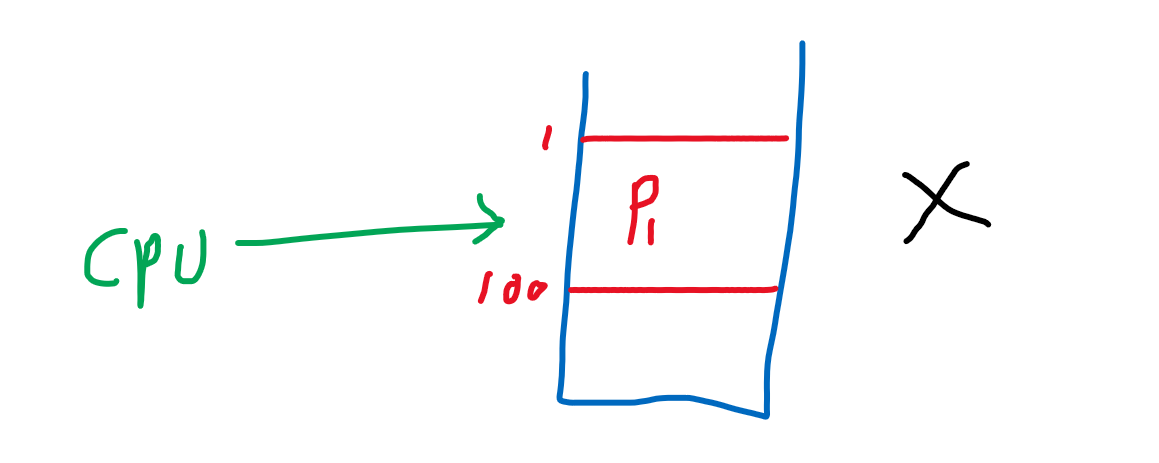


Where the base and limit register of a particular process is present??  
Whether in PCB of that particular process???

**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

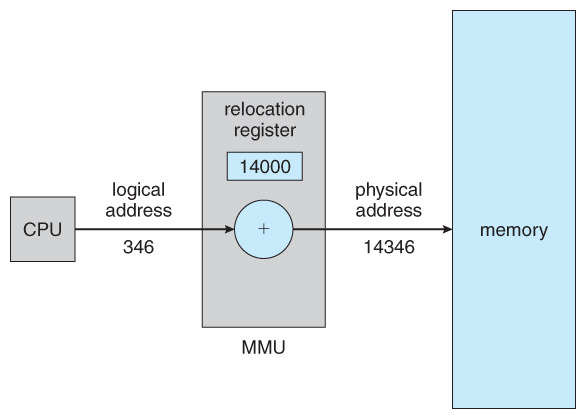


Linking

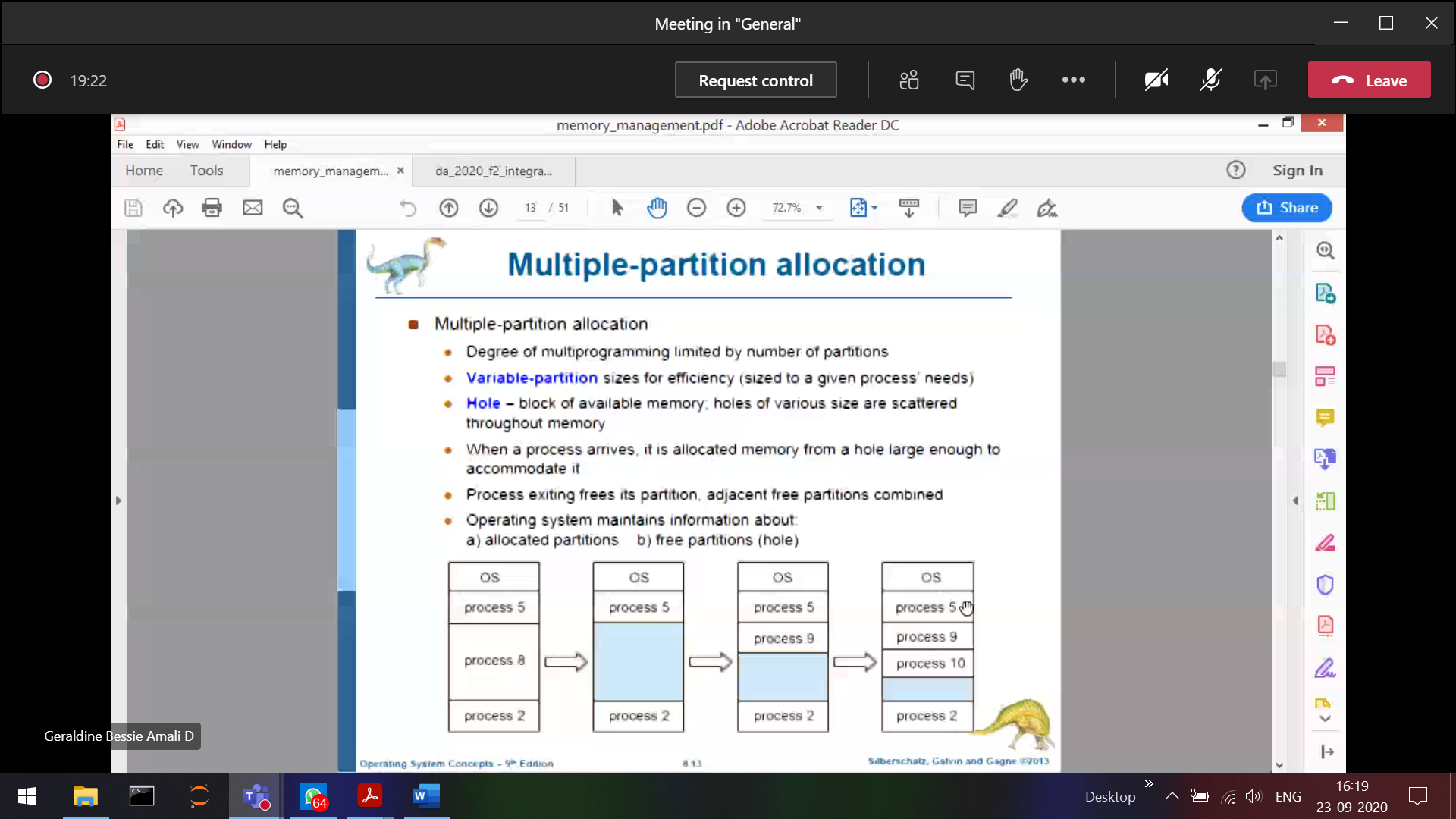
**Execution time:**Linking takes place during run time. Dynamic memory allocation. That time the linking can take place.

That linking takes place in Memory management unit

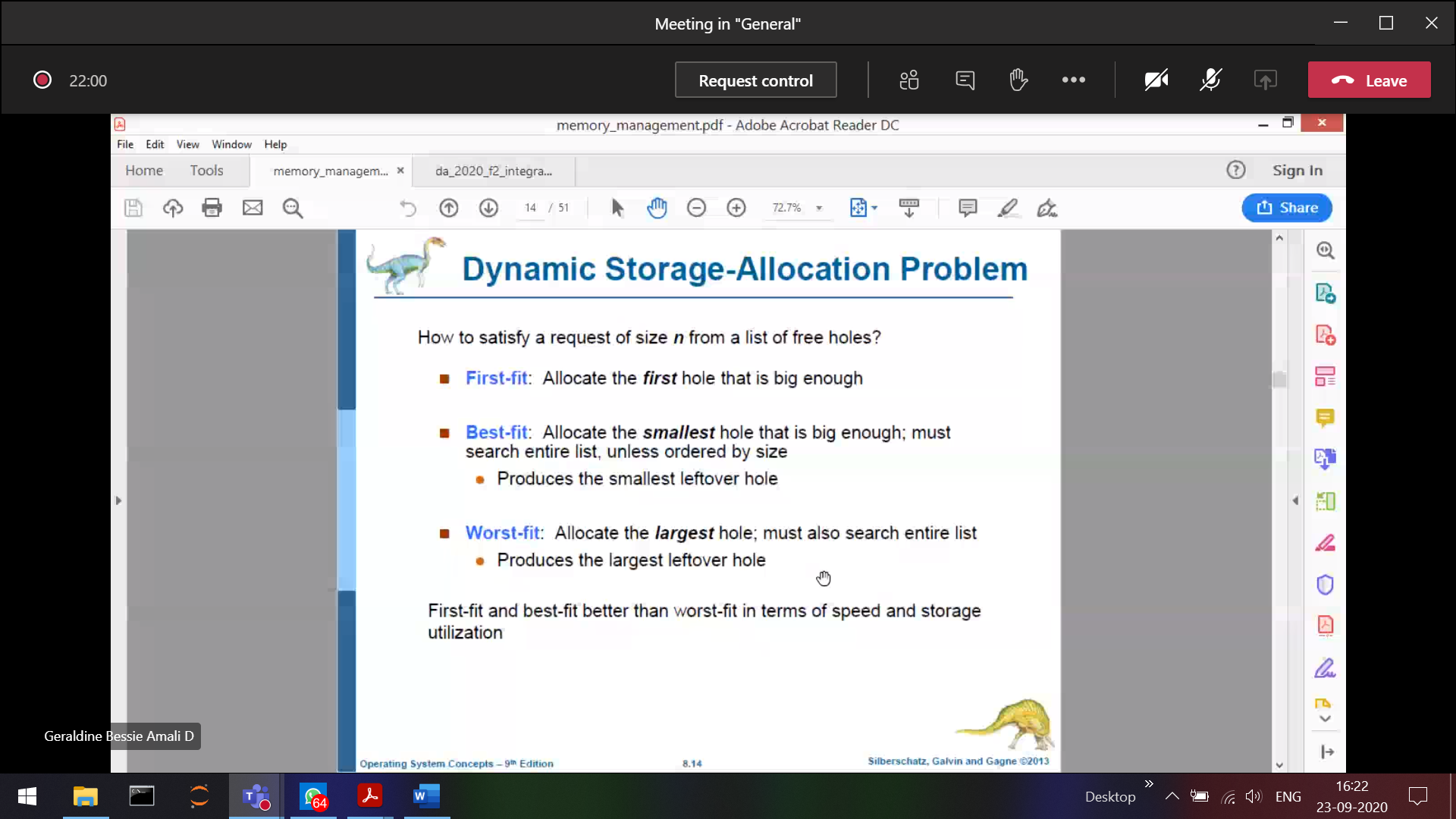
relocation register = base register

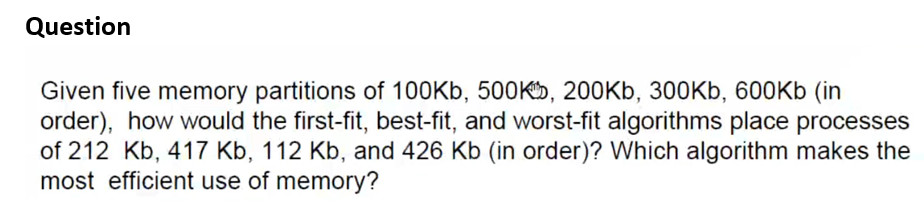


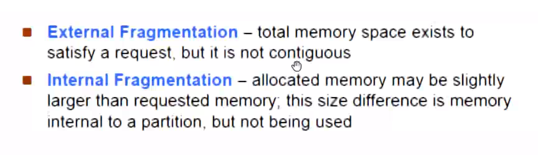
CPU wants the 346th line of the code(i.e instruction). This 346(logical address) will be added to the 14000 relocation register(base address register) and 14346 is the exact place where that instruction is stored and this instruction is fetched by the main-memory.



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.



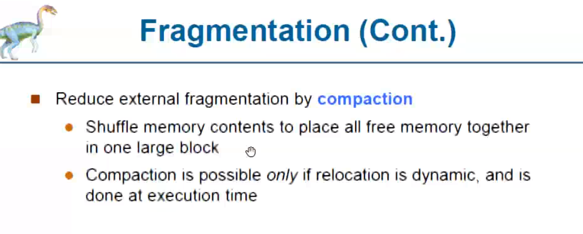




External fragmentation 🡪 Even-though the memory space exists, the process P3 cannot able to enter since the blocks are not continuous and scattered everywhere.

External fragmentation can be avoided by using compaction.

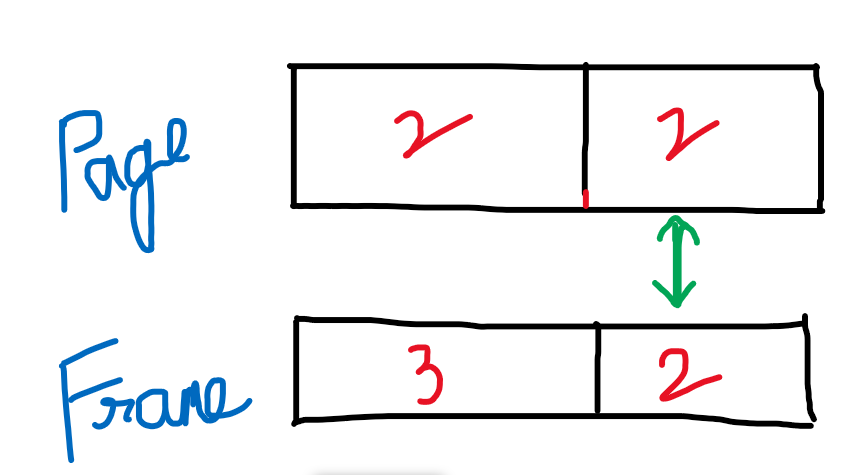
Defragmentation/Compaction 🡪 Collect all the wholes and push to one side, so that the whole size becomes bigger. Now P3 can easily enter into RAM.



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**





Page table base register 🡪 to point to the

Page table length register 🡪