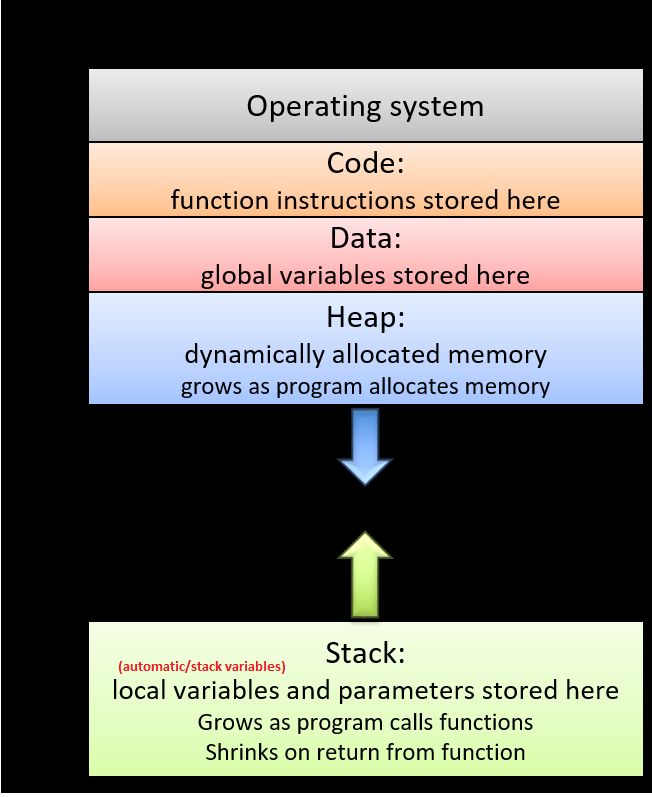
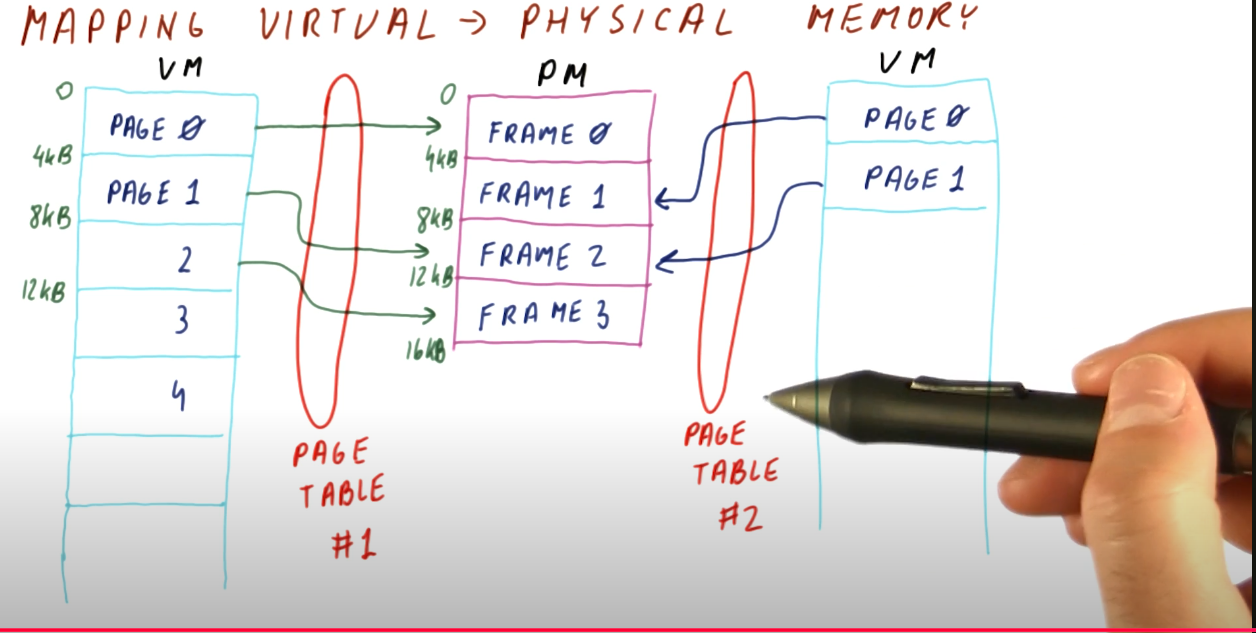
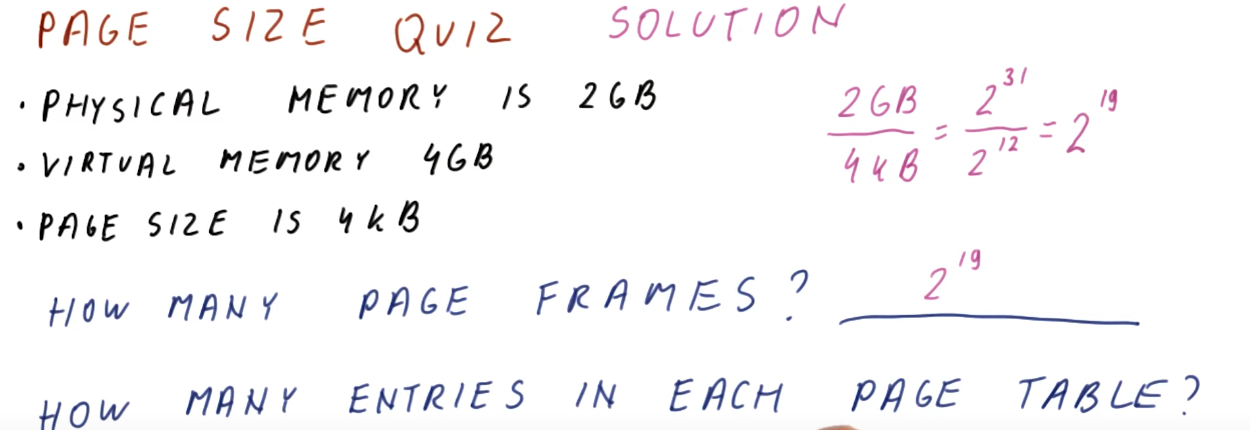
**Virtual Memory:**

* Why Virtual Memory?
  + Programmer’s view Vs H/W view of memory
* Processor’s View Of Memory:
  + What processor sees is called physical memory – memory contained in actual memory modules.
  + Addresses used by processor have 1:1 mapping to bytes/words in physical memory
* Program’s view of Memory:
  + Regions:- OS, Code, Data, Heap (grows downwards) & Stack (grows upwards)
  + 
  + Some contiguous regions of program’s view of memory are actually used by the program. And there is enormous region in the middle between the heap & stack that the program will never access unless the heap/stack grows downwards/upwards resp.
  + The above is virtual memory. Every process/program has its own virtual memory.
  + How can virtual memory (address of program’s view of memory) be mapped to a corr physical memory address?
* Virtual Memory 🡪 Physical Memory Mapping Introduction:
  + When a program generates a virtual address (by either doing a load or a store operation), the processor needs to address some physical address.
  + How does the processor map what the program is trying to access(Virtual memory address) TO what really should be accessed(Physical memory address)
  + Mapping every virtual memory address to physical memory address is not a feasible option.
  + Instead, virtual memory (program memory) can be divided into equal size chunks called PAGES. Each page is typically 4KB.
  + Physical Memory is divided into slots that can hold pages & these slots are called FRAMES.
  + Correlation:- Pretty much the physical memory behaves like a cache for the virtual memory in that it has certain number of slots/frames where we can put pages.
    - A page is kind of a memory block & frame is kind of a cache line
  + 
  + The OS decides when the VA to PA mapping needs to be done while actual mapping mechanism is called page table.
  + Page table says for each page in the process where is the page mapped to in its physical memory.

**Quiz:**

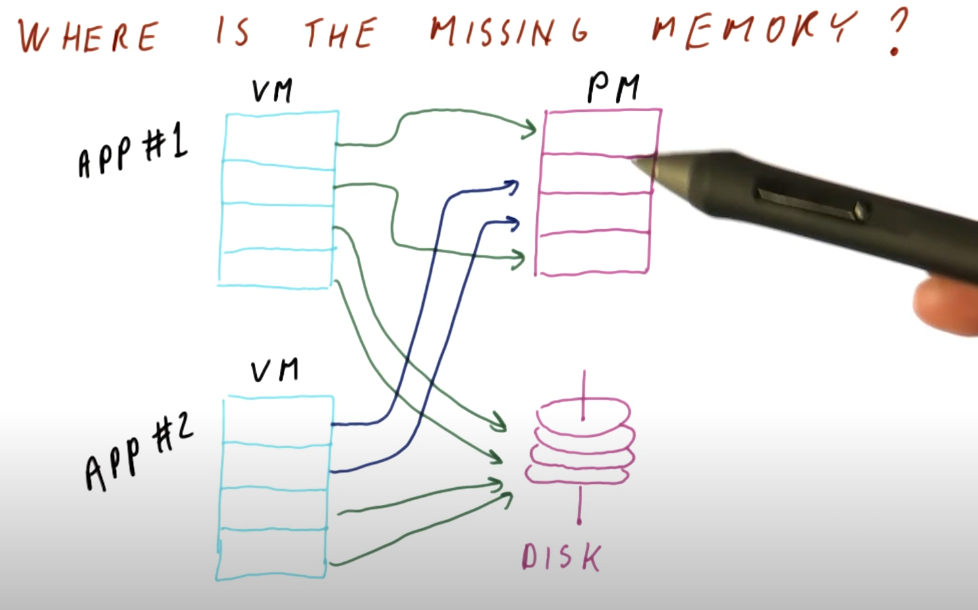


Ans:- frame size <=> page size

1)Page frames = Phy mem/page size => 2GB/4KB = 2^19 page frames.

2)We need one entry in page table for each page in virtual memory:

=> number of pages = virtual mem/page size = 4GB/4KB = 2^20 pages => 1 M entries in page table.



* Virtual memory of all the applications can significantly exceed the size of physical memory, some of the **virtual** memory that the applications think they have is on the DISK & not in physical memory.

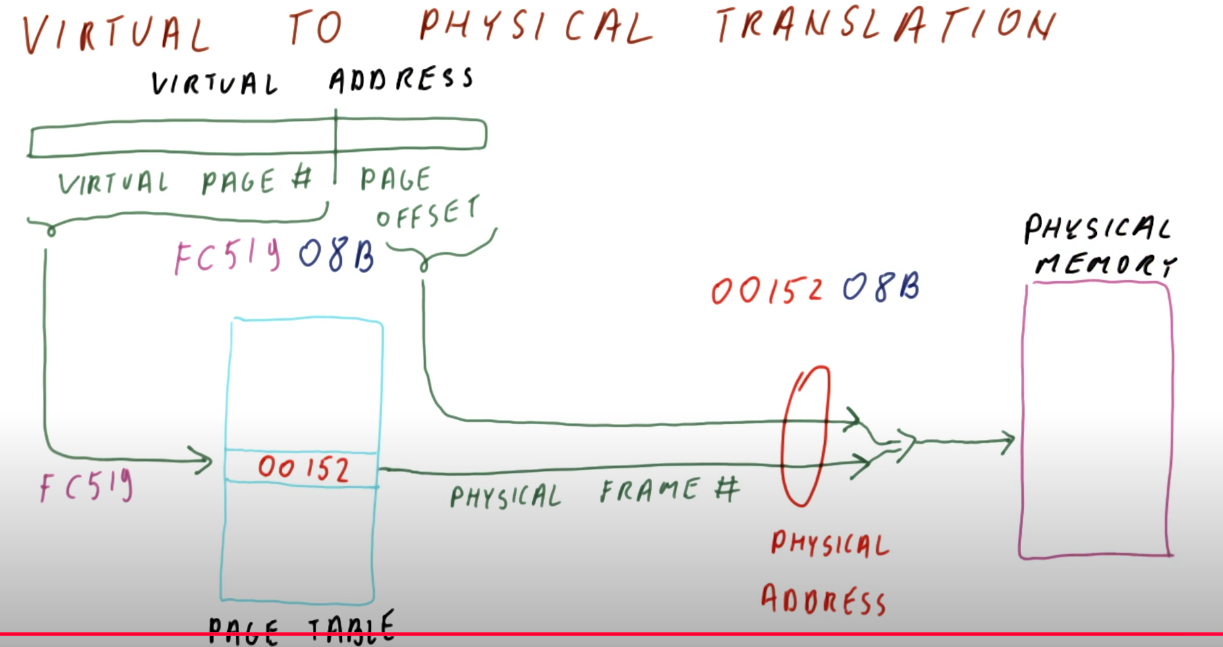
**Virtual to Physical Address Translation:**

1)Virtual Address will be generated by a program instruction/code, lets say load instruction.

2)Virtual Address is broken down into 2 parts:- least significant 12 bits represent the ***page offset*** (tells us where in the page we are-also assuming typical page size is 4KB) & ***Virtual page number*** (which tells us which page in the process we are talking about).

3)We take the virtual page number(VPN) from VA & use it to index into the page table. The entry corresponding to VPN in the page table tells us the corresponding ***physical frame number*** in physical memory.

4)The physical frame number is put together with page offset to generate a physical address which is then used access the physical memory.



a)Multi-level page tables/Hierarchical page tables (Where is savings)?

b)TLB