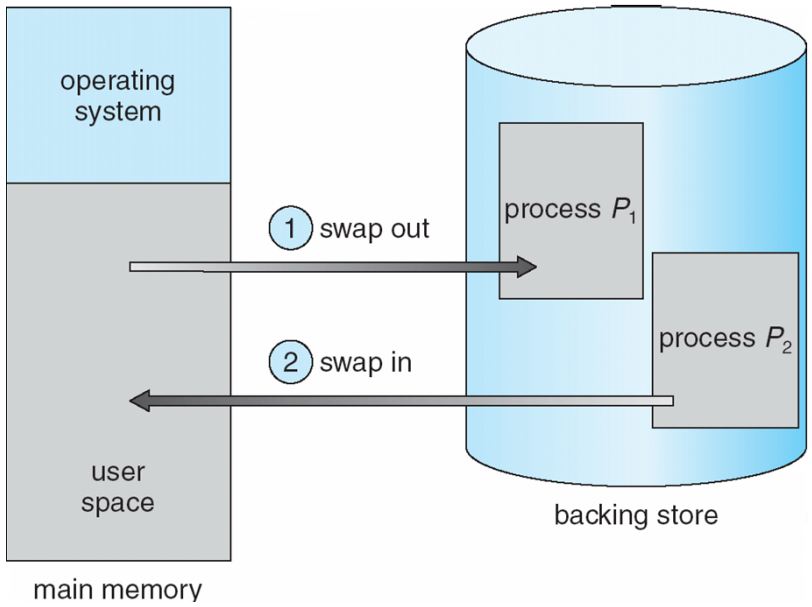
Swapping

- Allows the monitor to switch from one process to another
  - Current memory contents written to a backing store (disk)
  - Memory image for the next user process read in
  - Ready queue contains processes whose memory images are on disk (and ready to run)

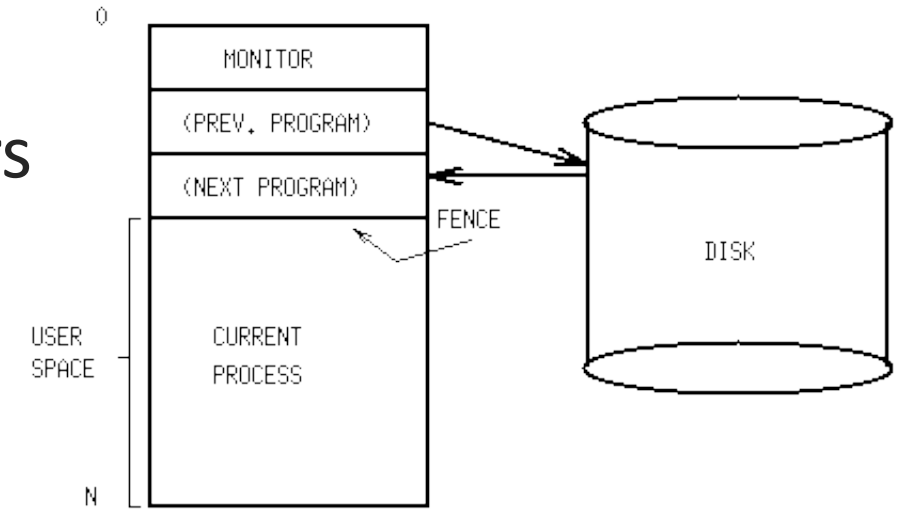


Swapping

- Fast device required (i.e., disk)
- context switch time is very high
  - Dependent upon the device performance
  - If we used a scheduling scheme like Round Robin, we would want a VERY large quantum

Background Swapping

- We need to set up two buffers: “incoming” and “outgoing”
- At context switch time we do two memory-to-memory copies
- During application execution, we do two disk-to-memory copies to update the buffers
  - requires a DMA device

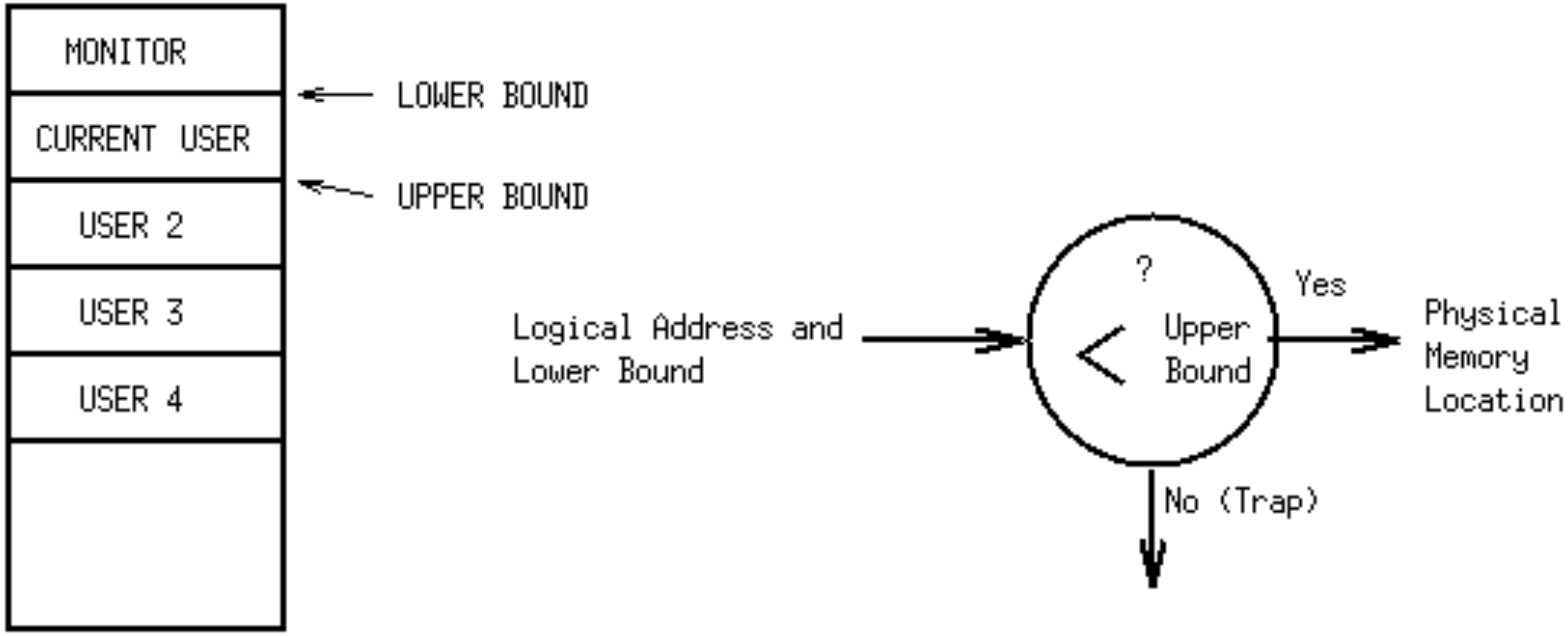


Swapping with Partitions

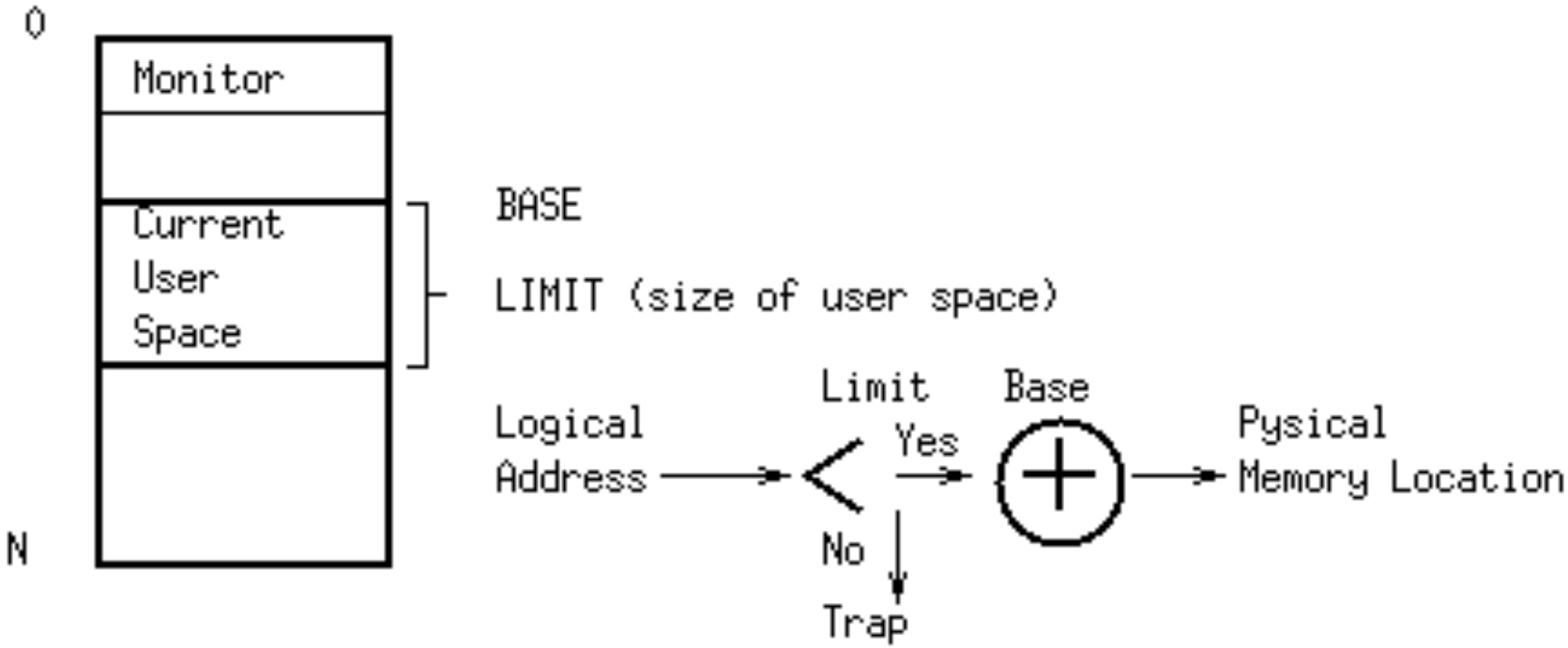
- Swapping with multiple programs (buffers)
  - a fence is inadequate

- one solution is to do some copying at context switch time - just more overhead
- need to protect access both before and beyond the user program

Upper/Lower Bound



Base/Limit Registers

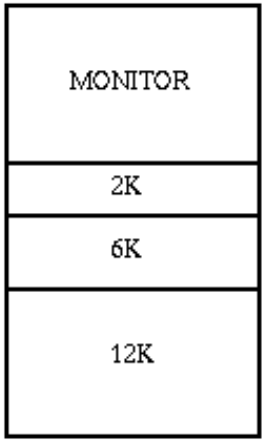


Swapping with Partitions

- Part of the context switching is the reloading of these registers
- Now, swapping can be done outside of the user space, with no copying
- Most significant: swapping can be eliminated in some cases
  - process can stay in memory between CPU bursts

Multiprogramming with a Fixed Number of Tasks

- Main memory is divided into a set of fixed partitions
- Each partition can be scheduled separately
  - possibly a separate ready queue for each memory partition
- Bounds are set just before and after the partition for the running job
- A process cannot span partition boundaries
- A partition cannot be shared
- When a process is done, another appropriately sized job will take its place



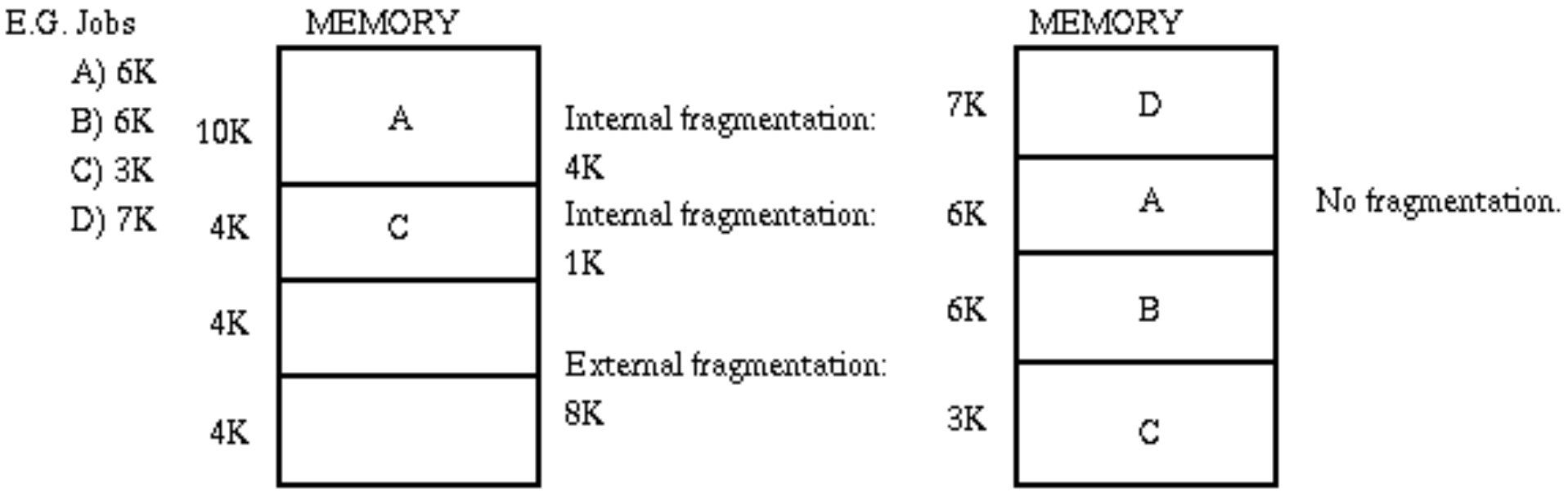
Multiprogramming with a Fixed Number of Tasks

- We could add swapping done separately for each partition
  - when a 6K job begins an I/O burst, we could swap it out and put another job in
- What if a 1.5K process in the 2K spot asks for another 1K of memory?
  - we can do any of the following:
    - abort the process
    - deny the request
    - swap it out, and put it on the queue for the 6k partition
- Region size selection is difficult

Multiprogramming with a Fixed Number of Tasks

- MFT suffers from fragmentation:
  - *internal fragmentation*: memory internal to one partition that is wasted because of the way we have set up the partitions
    - an X K job running in a YK partition
      => (y–x)K wasted space
  - *external fragmentation*: enough total free space exists to run a process, but the space can't be used for that process because no single partition can hold the entire process

Multiprogramming with a Fixed Number of Tasks



End of midterm material

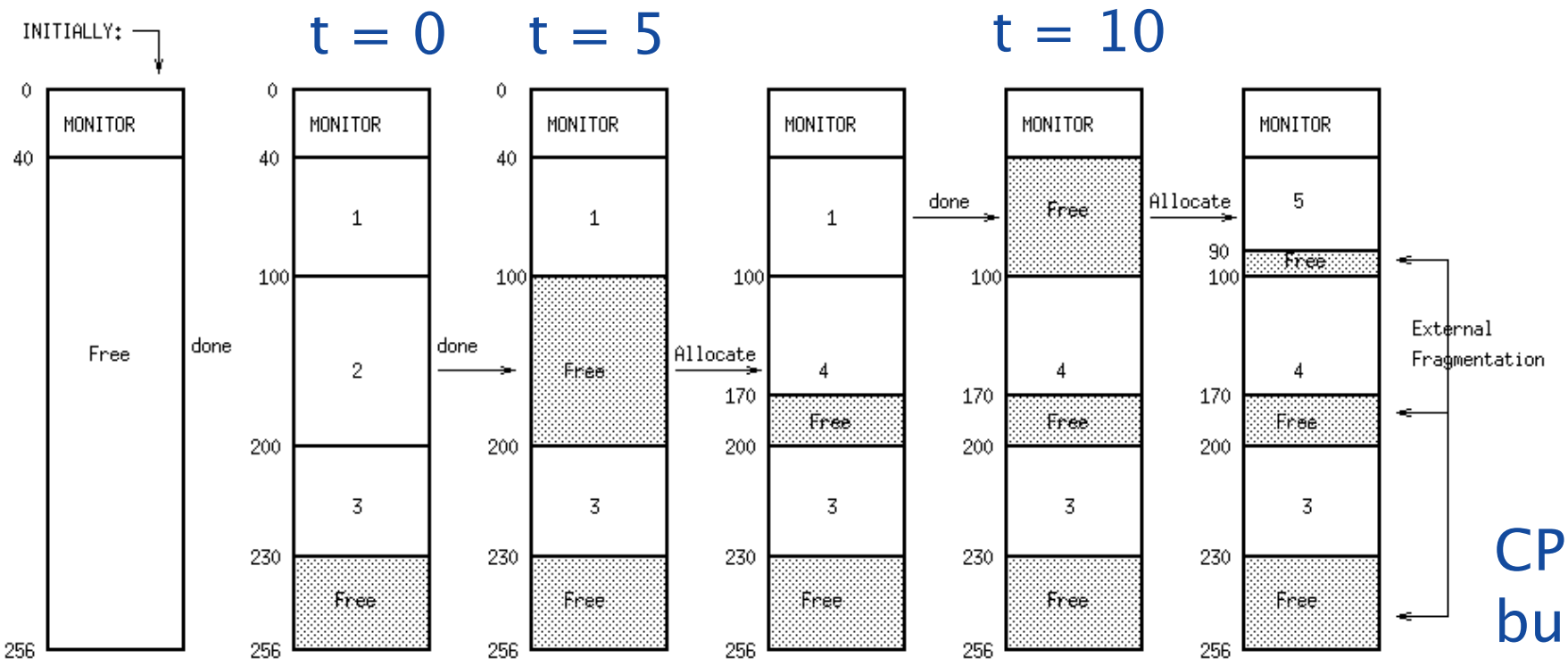
Multiprogramming with a Variable Number of Tasks

- MVT - “multiple, contiguous variable partition allocation”
- The O/S keeps track of which memory is in use
  - initially, all is free, and kept in one block
- A job arrives
  - O/S searches the free memory block(s)
  - If one is found that is big enough
    - we allocate the required portion to the job
    - mark the remainder of that block as a new, smaller, free block

Multiprogramming with a Variable Number of Tasks

- When a job finishes:
  - its memory is returned to the system
  - If the returned memory is next to another free block, the two are joined into one contiguous block

Multiprogramming with a Variable Number of Tasks



CPU burst

assuming FCFS scheduling

Jobs	Memory	Time
1	60	10
2	100	5
3	30	20
4	70	8
5	50	15

Multiprogramming with a Variable Number of Tasks

- Generally, free blocks can be kept in a list.
- When we want memory for a job, the list must be searched for a memory chunk that is big enough.
- Which do we choose?
  - This is an example of the general, dynamic storage allocation problem