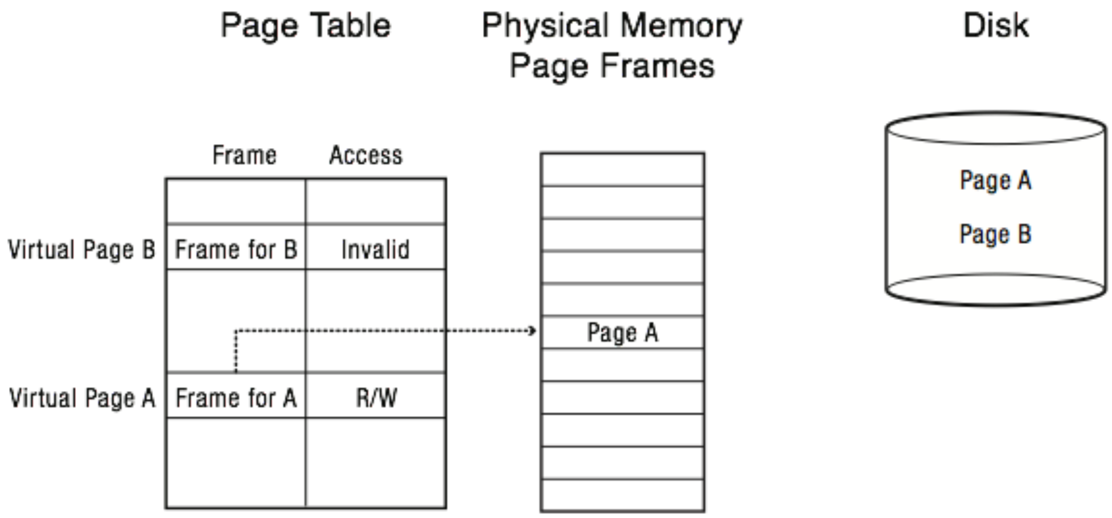
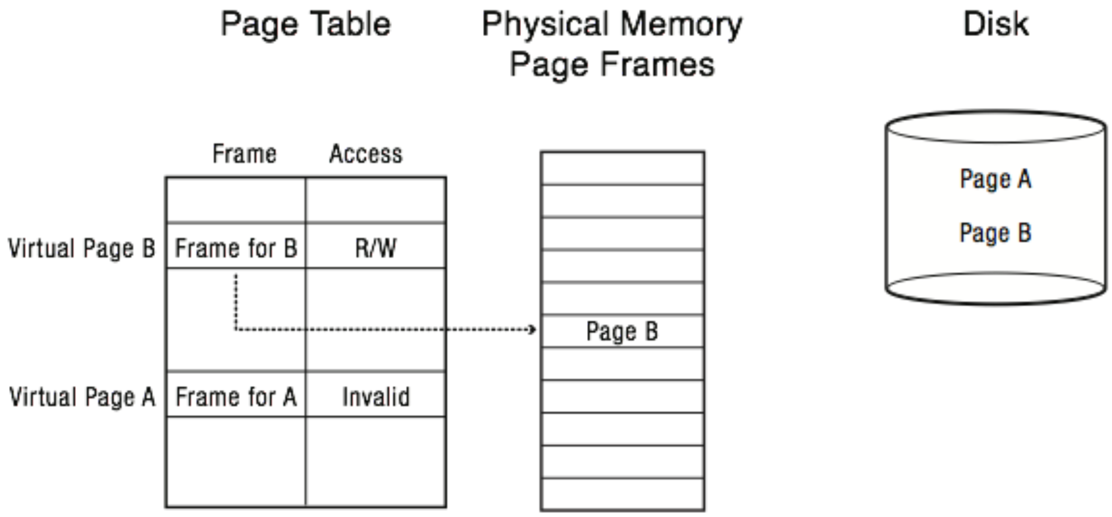
Topic 15: Demand Paging, Page Replacement

Reading: 9.1 - 9.6
Next reading: 11, 12.1, 13.3

- So far we have separated a process's view of memory from the operating system's view using a mapping mechanism. Each sees a different organization. This makes it easier for the OS to shuffle processes' memory around and simplifies memory sharing between processes.

- However, up to this point a process had to be completely loaded into memory before it could run. This is wasteful since a process only needs a small amount of its total memory at any one time (locality). *Virtual memory* permits a process to run when it is only partially loaded into memory.
    - A process's pages that are not in memory are stored on disk. The space on disk used to store pages is called the *backing store* or *swap*. (Figures 9.17, 9.18)





- ○ VAS can be larger than PAS.

- The idea is to produce the illusion of memory as large as a disk and as fast as main memory.

- The reason that this works is that most programs spend most of their time in only a small piece of the code.
    - 90/10 rule -- Knuth's estimate is of 90% of the time in 10% of the code.
- If not all of process's memory is loaded when it is running, what happens when it references a byte that is only in the backing store?
    - Hardware and OS cooperate to make things work anyway.
- First, need to keep track of where each page is stored on disk.
    - Use contiguous disk blocks, file, or table (e.g. extend page table to include disk location when page is not in memory).
- Second, extend the page tables to include a *present* bit. If present bit isn't set then a reference to the page results in a trap called a *page fault*. Present bit can be implemented via invalid access permissions.
    - Any page not in main memory has its present bit cleared in its page table entry.
    - When page fault occurs:
        - If it's a new page (never before accessed), fill it with zeros.
        - Otherwise, bring page into memory (context switch while I/O is taking place).
        - 2 Page table is updated, present bit is set.

3 ■ The process continues execution.

- ○ In USLOSS the present bit is called the *incore* bit.

- Continuing process is very tricky, since page fault may have occurred in the middle of an instruction. Don't want user process to be aware that the page fault even happened.
  - ○ Can the instruction just be skipped?
  - ○ Suppose the instruction is restarted from the beginning?
    - ■ How is the "beginning" located?
    - ■ Even if the beginning is found, what about instructions with side effects, like MOVE (SP)+, R2?
    - ■ Some hardware is not restartable in this way, and cannot easily support virtual memory, e.g. 68000. Apollo workstation had 2 CPUs; second CPU handled page fault while first was suspended in memory access.

- Once the hardware has provided basic capabilities for virtual memory, the OS must make two kinds of scheduling decisions:
  - when IN ○ *Page selection*: when to bring pages into memory.
  - when out who ○ *Page replacement*: which page(s) should be swapped out, and when.

- Page selection algorithms:
  - ○ *Demand paging*: start up process with no pages in memory, swap in a page when a page fault for it occurs, i.e. wait until