Chapter 9: Virtual Memory, On-Demand Paging

CSCI 3753 Operating Systems
William Mortl & Prof. Rick Han

University of Colorado Boulder

On-Demand Paging

- Page tables may be large, consuming much RAM
- Key observation: not all pages in a logical address space need to be kept in memory
  - in the simplest case, just keep the current page where the PC is executing
  - all other pages could be on disk
  - when another page is needed, retrieve the page from disk and place in memory before executing
    - this would be costly and slow, because it would happen every time that a page different from the current one is needed

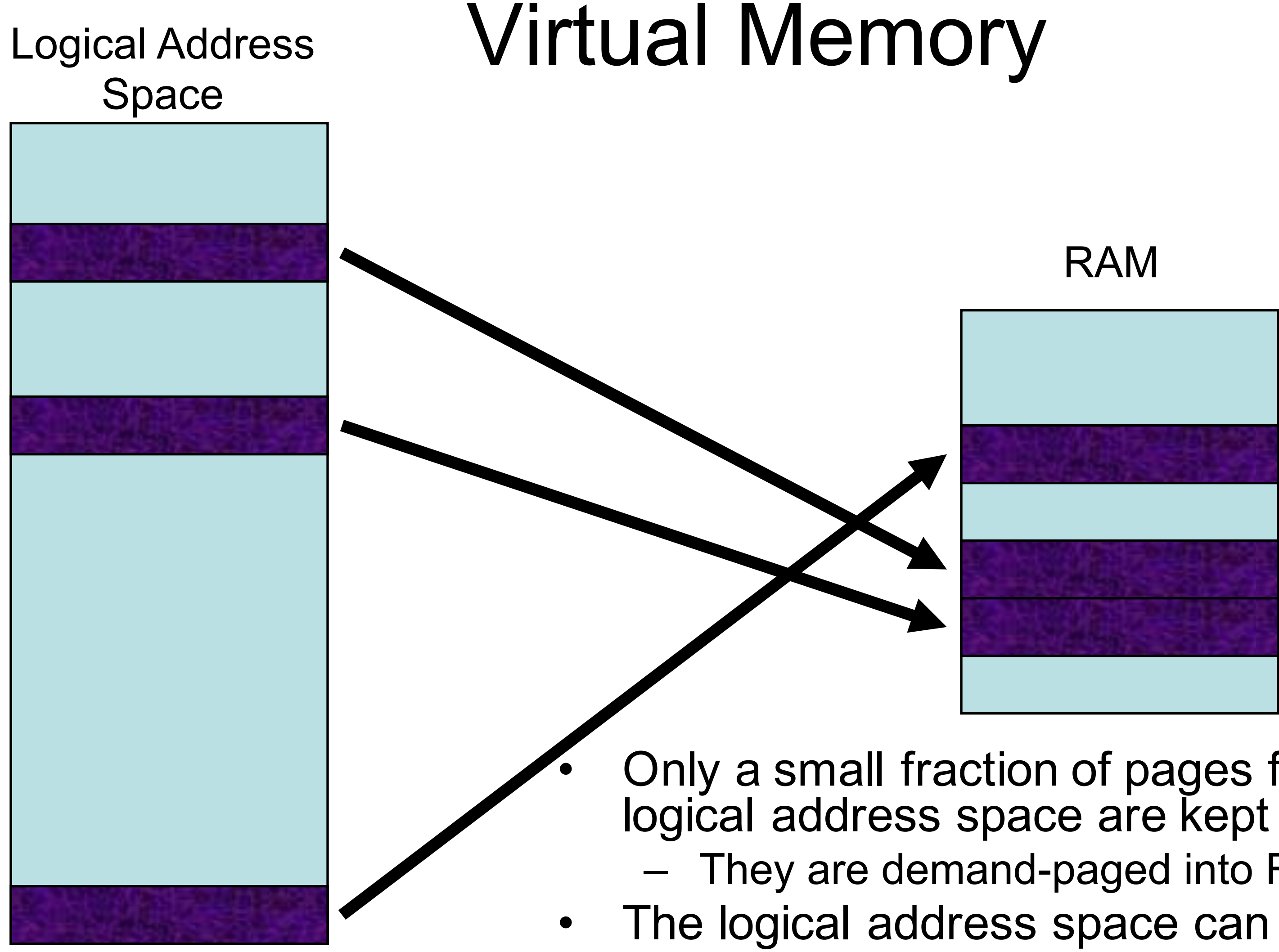
On-Demand Paging

- Instead of just keeping one page, keep a *subset* of a process's pages in memory
  - Load just what you need, not the entire address space
  - use memory as a cache of frequently or most recently used pages
  - rely on a program's behavior to exhibit locality of reference
    - if an instruction or data reference in a page was previously invoked, then that page is likely to be referenced again in the near future

On-Demand Paging

- most programs exhibit some form of locality
  - looping locally through the same set of instructions,
  - branching through the same code
  - executing linearly, the next instruction is typically the next one immediately after the previous instruction, rather than some random jump
- Thus process execution revisits pages already stored in memory
  - so you don't have to go to disk each time the program counter (PC) jumps to a different page

Virtual Memory



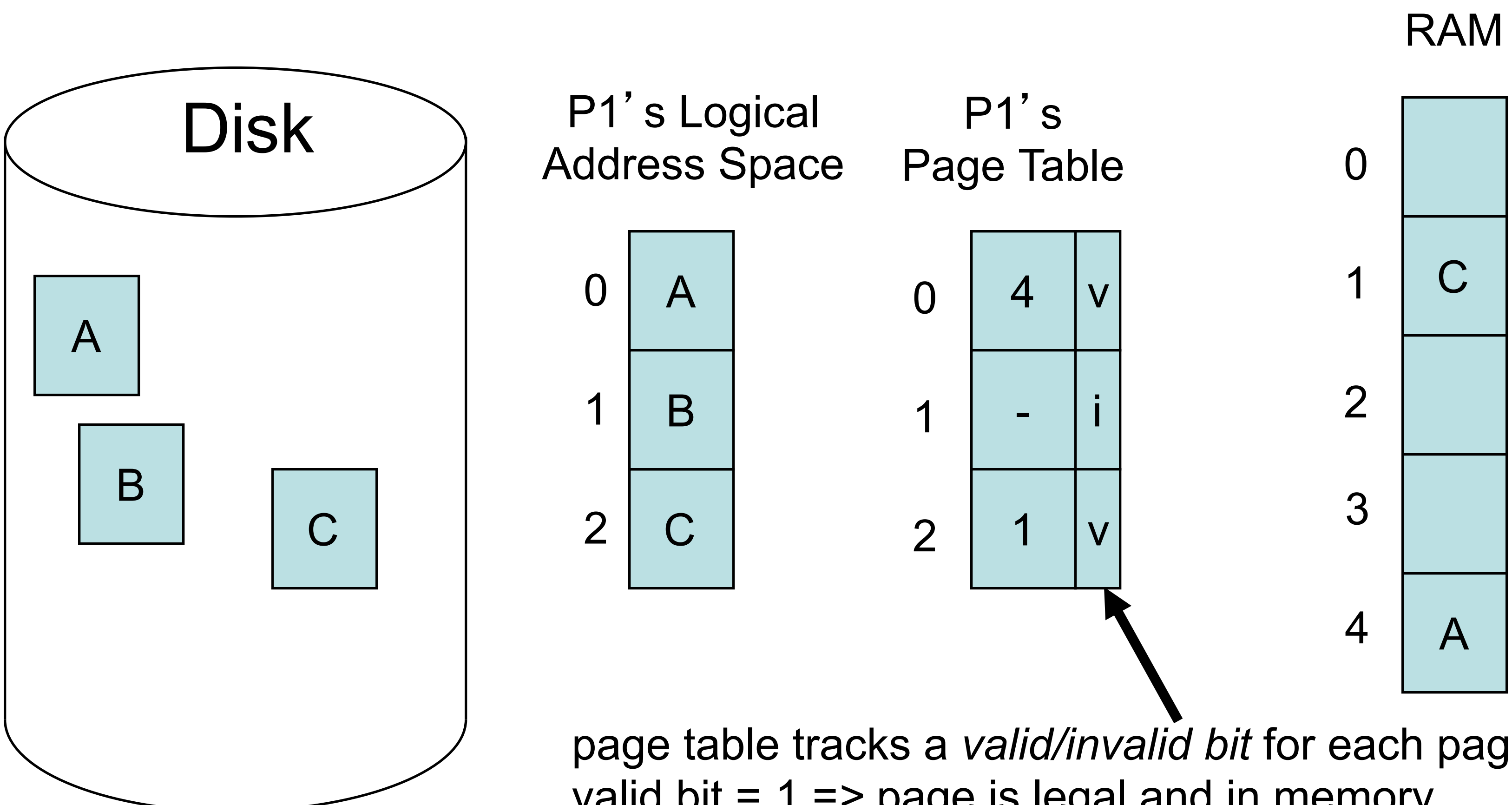
- Only a small fraction of pages from the logical address space are kept in RAM
  - They are demand-paged into RAM
- The logical address space can be much larger than physical RAM, hence the logical addresses form a *virtual memory*

On-Demand Paging

- On-demand paging is used to page in new pages from disk to RAM
  - only when a page is needed is it loaded into memory from disk
- Can page in an entire process on demand:
  - starting with “zero” pages
  - the reference to the first instruction causes the first page of the process to be loaded on demand into RAM.
  - Subsequent pages are loaded on demand into RAM as the process executes.

On-Demand Paging

- On-demand paging loads a page from disk into RAM only when needed
  - in the example below, pages A and C are in memory, but page B is not



page table tracks a *valid/invalid bit* for each page
valid bit = 1 => page is legal and in memory

Virtual Memory Advantages

1. the virtual address space can now exceed physical RAM!
   – before, we had to worry about how to fit a logical A.S. into RAM
   – now only a subset of the (most demanded) pages are kept in RAM, so the logical A.S. can be almost arbitrarily large
     • example: if a physical memory has only 10 pages, and a subset of 3 pages of each logical A.S. is kept in memory, then it doesn't matter whether the size of my logical A.S. is 100 pages or 1000 pages!
   – So we have decoupled virtual memory from physical memory.

Virtual Memory Advantages

2. can fit many more processes in memory!
3. decreases swap time – there is less to swap
4. Can have large sparse address spaces, in which most of the address space is unused, without taking up lots of physical RAM
   – a large heap and stack that is mostly unused/empty won't take up any actual RAM until needed, i.e. when the stack or heap grows very large

On-Demand Paging

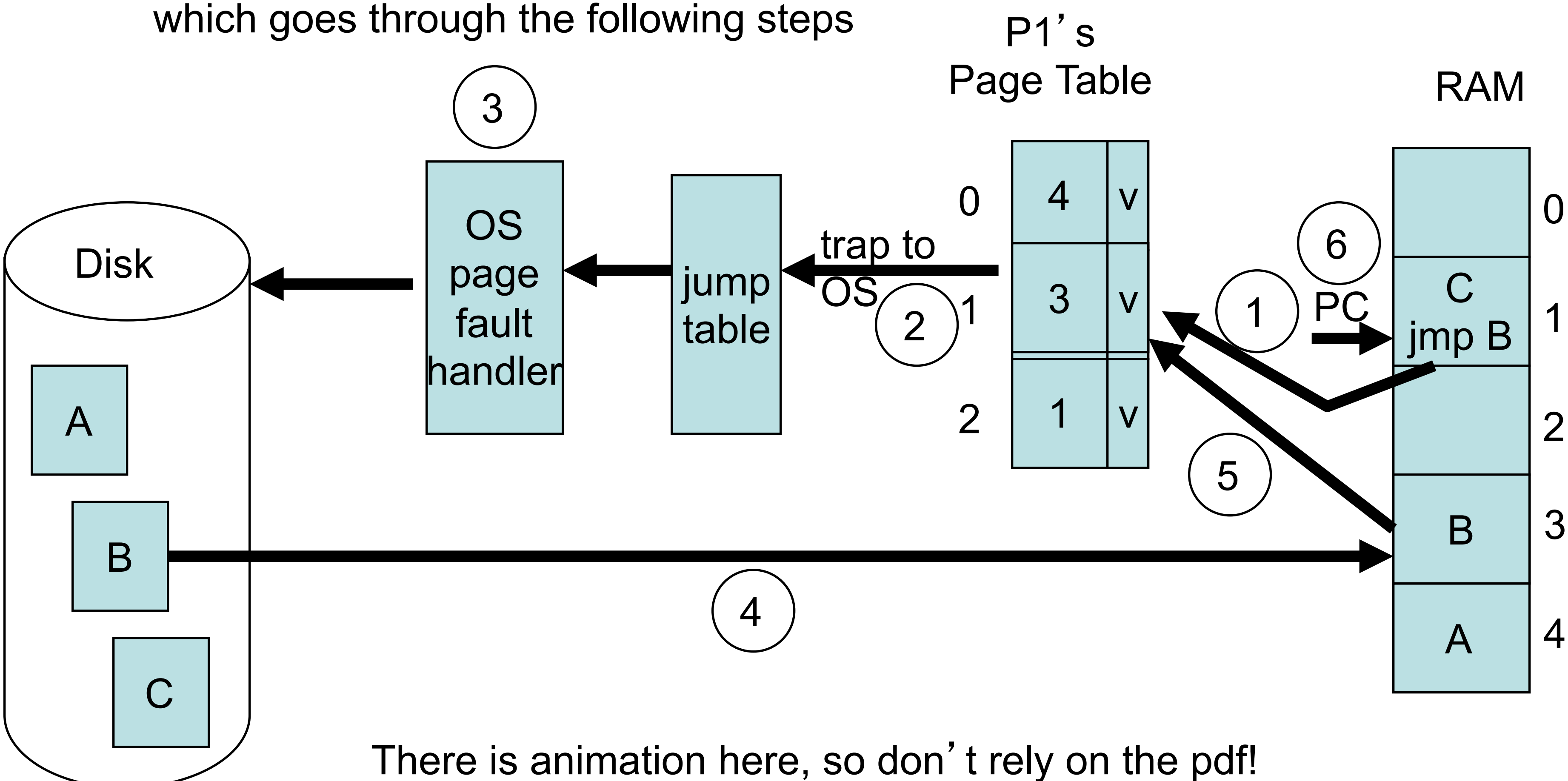
- Open questions to be answered:
  - how is a needed page loaded into memory from disk?
  - how many pages in memory should be allocated to a process?
  - if the # of pages allocated to a process is exceeded, i.e. the cache is full, then how do you choose which page to replace?

Virtual Memory

- Page-fault steps to load a page into RAM:
  1. MMU detects a page is not in memory (invalid bit set) which causes a *page-fault trap* to OS
  2. OS saves registers and process state. Determines that the fault was due to demand paging and jumps to page fault handler
  3. *Page fault handler*
     a) If reference to page not in logical A.S. then seg fault.
     b) Else if reference to page in logical A.S., but not in RAM, then load page
     c) OS finds a free frame
     d) OS schedules a disk read. Other processes may run in meantime.
  4. Disk returns with interrupt when done reading desired page. OS writes desired page into free frame
  5. OS updates page table, sets valid bit of page and its physical location
  6. Restart interrupted instruction that caused the page fault

Virtual Memory

On-demand paging causes a page fault, which goes through the following steps



There is animation here, so don't rely on the pdf!

Virtual Memory

- OS can retrieve the desired page either from the disk's
  - file system, or
  - from the swap space/backing store
    - faster, avoids overhead of file system lookup
- pages can be in swap space because:
  - the entire executable file was copied into swap space when the program was first started.
    - Avoids file system, but also allows the copied executable to be laid out contiguously on disk's swap space, for faster access to pages (no seek time)

Virtual Memory

- pages can be in swap space because:
  - as pages have to be replaced in RAM, they are written to swap space instead of the file system's portion of disk.
    - The next time they're needed, they're retrieved quickly from swap space, avoiding a file system lookup.

Performance of On-Demand Paging

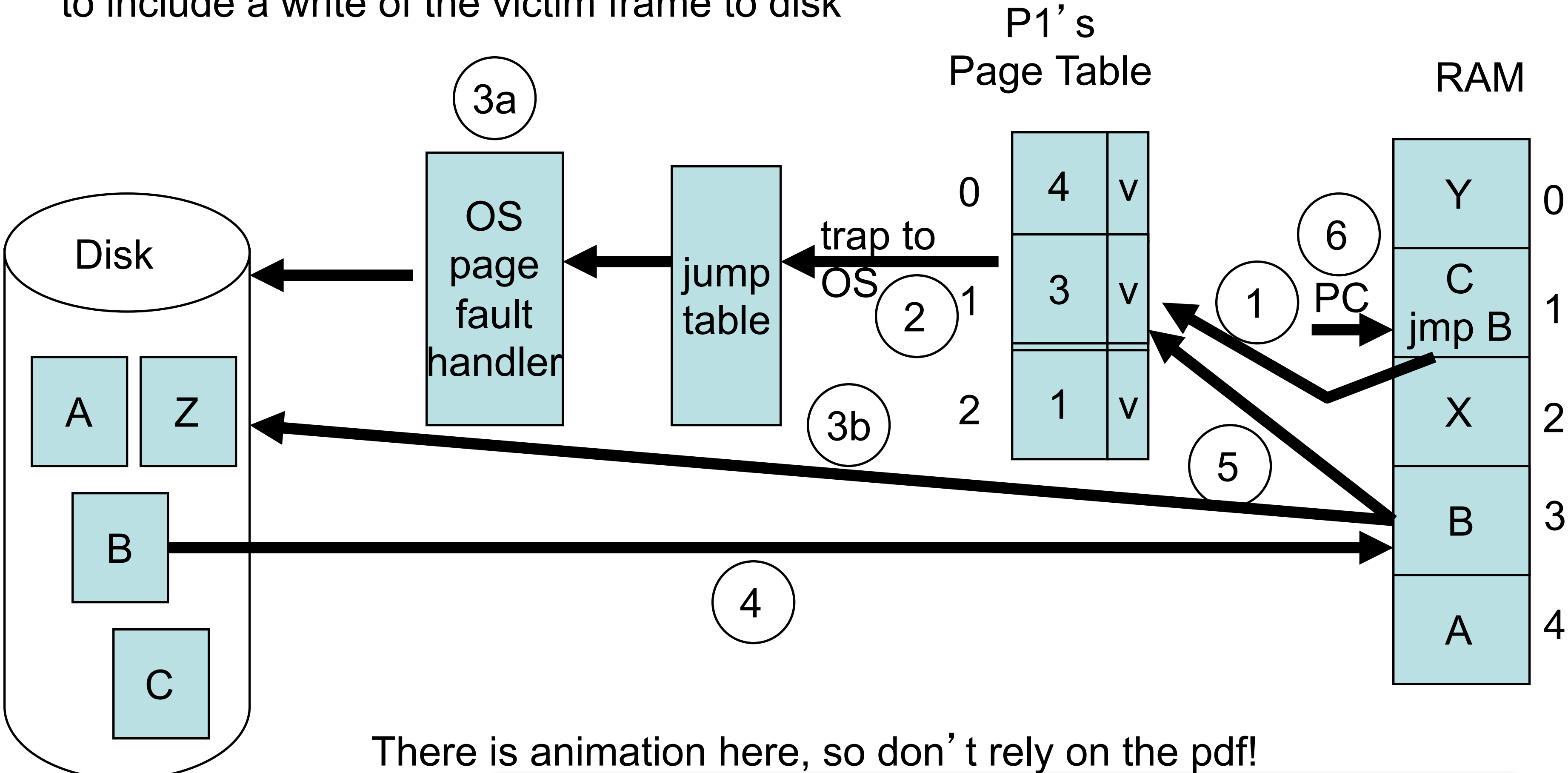
- want to limit the number/frequency of page faults, which cause a read from disk, which slows performance
  - disk read is about 10 ms
  - memory read is about 10 ns
- What is the average memory access time?
  - average access time = p*10 ms + (1-p)*10 ns, where p = probability of a page fault.
  - if p=.001, then average access time = 10 μs >> 10 ns (1000 X greater!)
  - to keep average access time within 10% of 10 ns, would need a page fault rate lower than $p<10^{-7}$
  - Reducing page fault frequency improves performance in a big way

Page Replacement Policies

- As processes execute and bring in more pages on demand into memory, eventually the system runs out of free frames
  - need a *page replacement policy*
    1. select a victim frame that is not currently being used
    2. save or write the victim frame to disk, update the page table (page now invalid)
    3. load in the new desired page from disk
  - If out of free frames then each page fault causes 2 disk operations, one to write the victim, and one to read the desired page - this is a big performance penalty

Page Replacement Policies

In Step 3b, we modify traditional on-demand paging to include a write of the victim frame to disk



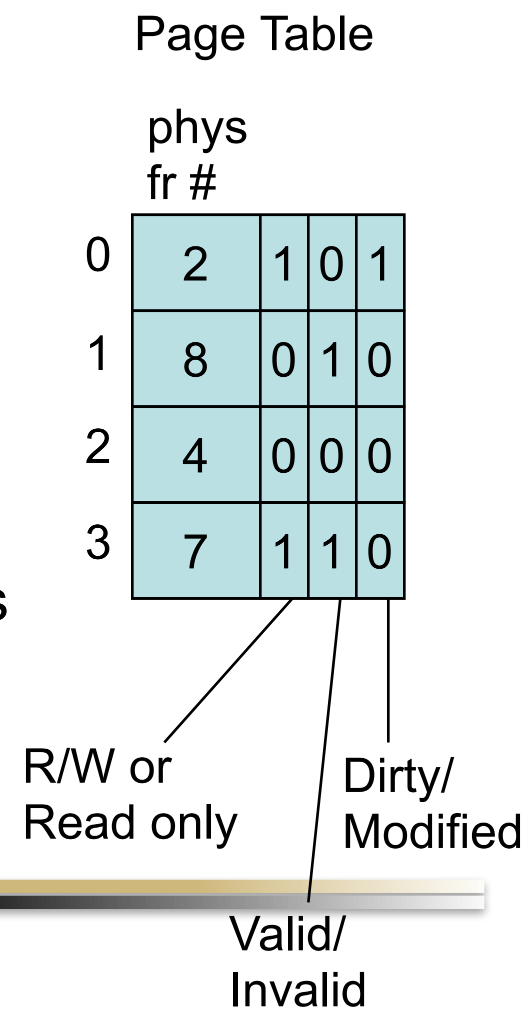
There is animation here, so don't rely on the pdf!

Page Replacement Policies

- To reduce the performance penalty of 2 disk operations, systems can employ a *dirty/modify bit*
  - modify bit = 0 initially
  - when a page in memory is written to, set the bit = 1
  - when a victim page is needed, select a page that has not been modified (dirty bit = 0)
    - such an unmodified page need not be written to disk, because its mirror image is already on disk!
    - this saves on disk I/O - reduces to only 1 disk operation (read of desired page)

Page Table Status Bits

- Each entry in the page table can conceptually store several extra bits of metadata information along with the physical frame # f
  - *Valid/invalid bits* - for memory protection, accessing an invalid page causes a page fault
    - Is the logical page in the logical address space?
    - If there is virtual memory (we'll see this later), is the page in memory or not?



Page Table Status Bits

- *dirty bits* - has the page been modified for page replacement?
- *R/W or Read-only bits* - for memory protection, writing to a read-only page causes a fault and a trap to the OS
- *Reference bit* – useful for Clock page replacement algorithm (next lecture)