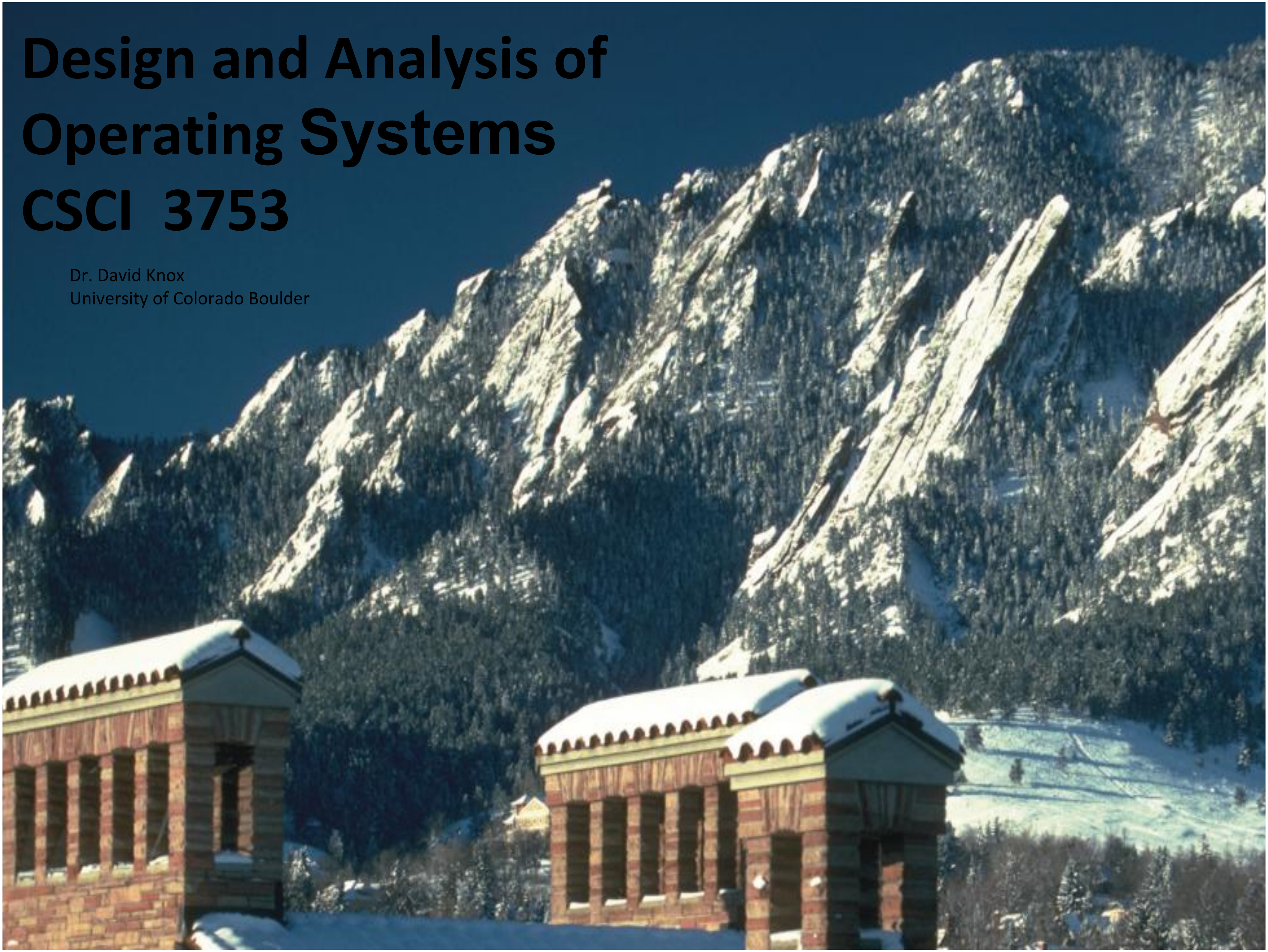
Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder



Design and Analysis of Operating Systems CSCI 3753

Page Replacement Policies

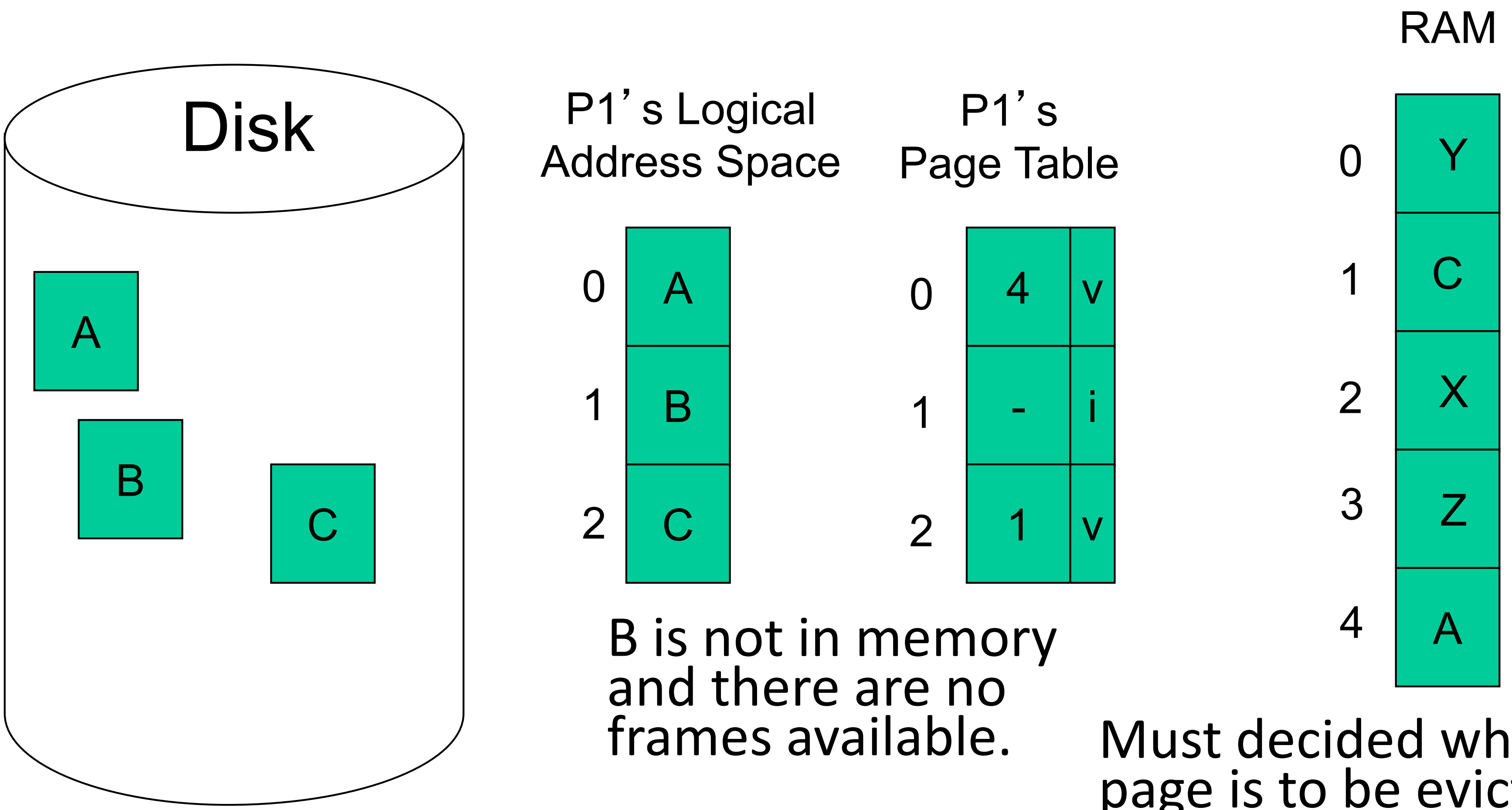
Dr. David Knox

University of Colorado Boulder

Memory Management
Page Replacement Policies

Demand Paging

- **Demand paging loads a page from disk into RAM only when needed**
  - in the example below, pages A and C in memory, but page B is not
  - How can we access B if all other frames are already used



B is not in memory and there are no frames available.

Must decided which page is to be evicted from memory.

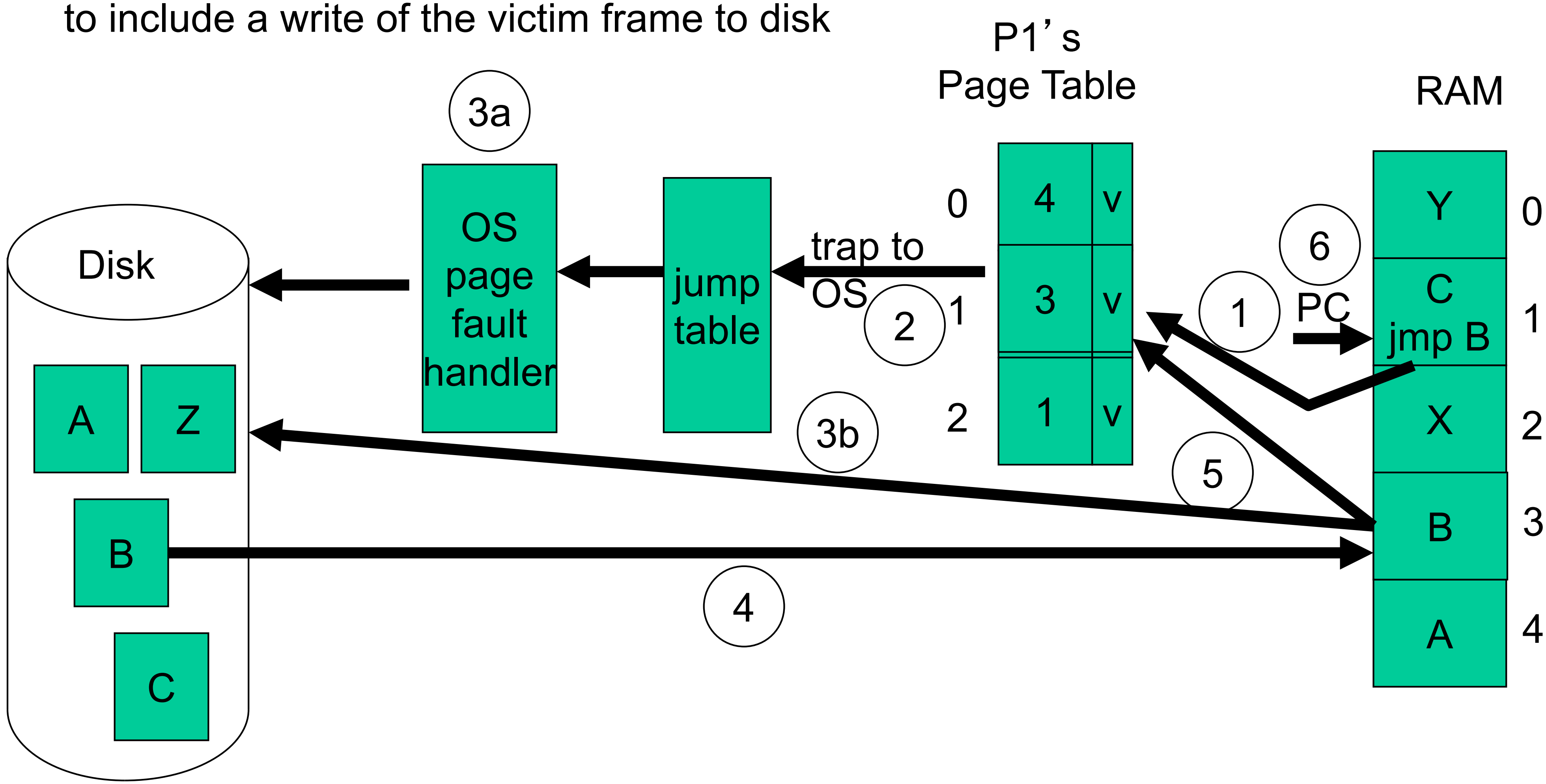
Page Replacement Policies

- **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, 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

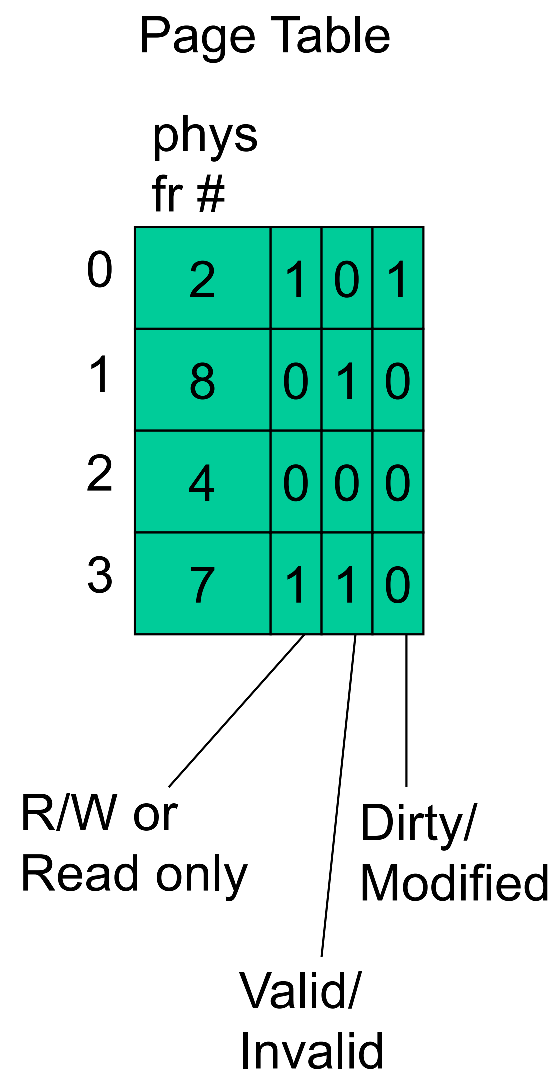


Page Replacement Policies

- **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)

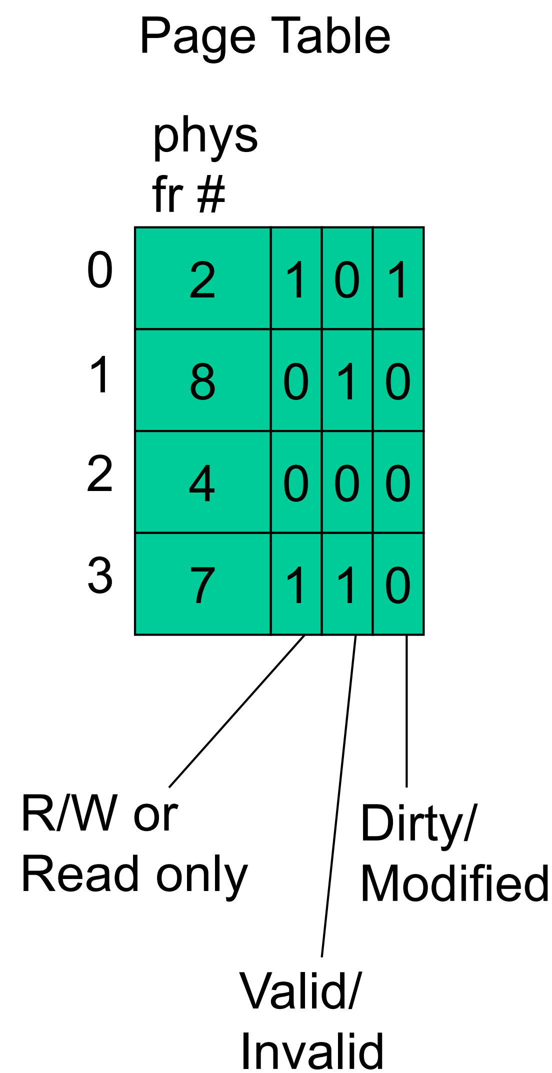
Page Table Status Bits

- **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



Recap …

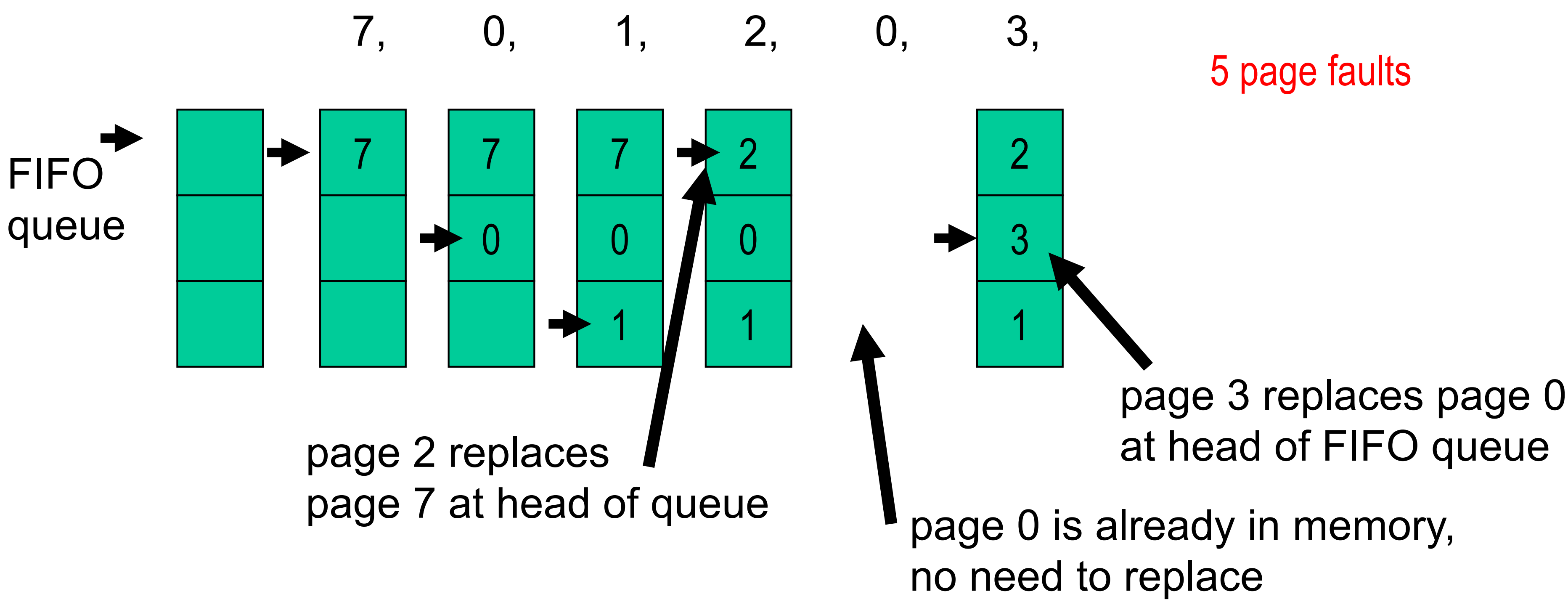
- **Virtual memory**
  - Keep only a few pages in memory, rest on disk
  - On-demand paging: retrieve a page when needed
  - Page fault
    - A referenced page is not loaded in memory
    - OS blocks the process and retrieves the referenced page
    - Significant performance overhead – need to keep page fault frequency low, e.g. less than 1 in 10^7 for overhead <10%
  - Page replacement algorithm
    - Principle of locality of reference
    - Dirty bit: choose a clean page before a dirty page

Page Replacement Policies

- **FIFO**
- **OPT**
- **LRU (least recently used)**
- **Evaluation**
  - Variables: algorithm, page reference string, # of memory frames
  - algorithm with lowest # of page faults is most desirable

FIFO Page Replacement

- **FIFO - create a FIFO queue of all pages in memory**
  - example reference string: 7, 0, 1, 2, 0, 3, ...
  - assume also that there are 3 frames of memory total

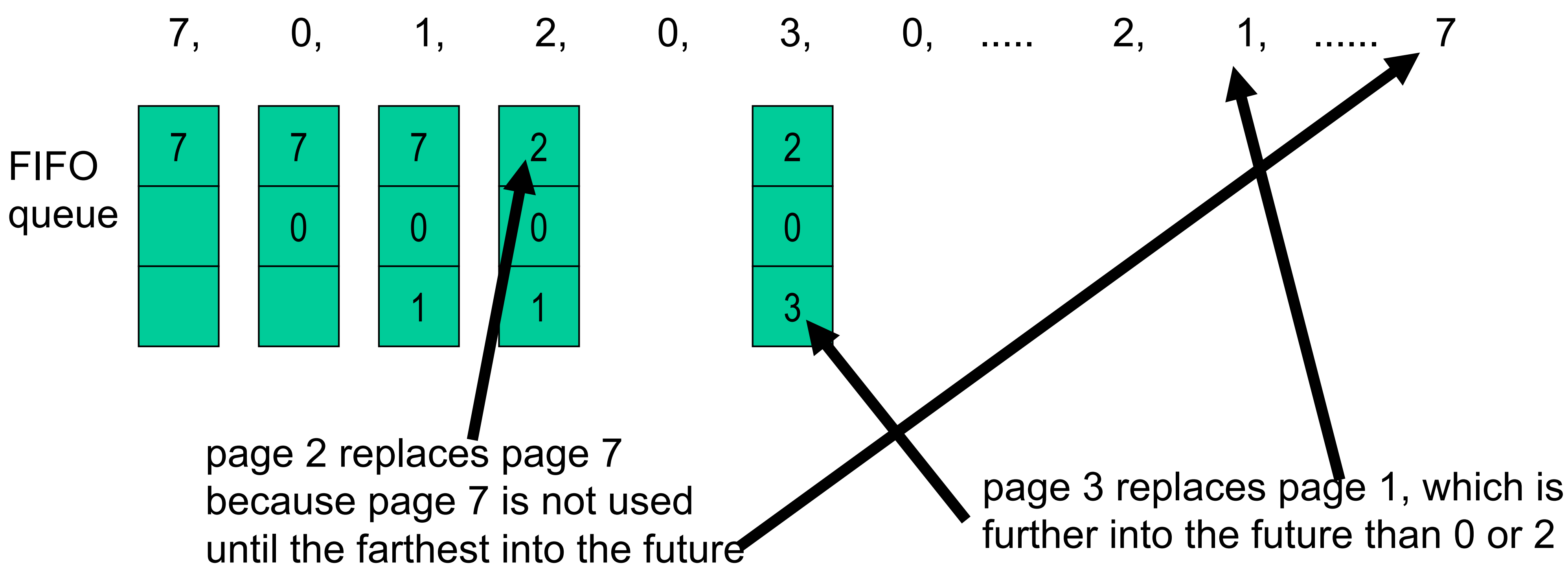


FIFO Page Replacement

- **FIFO is easy to understand and implement**
- **Performance can be poor**
  - Suppose page 7 that was replaced was a very active page that was frequently referenced, then page 7 will be referenced again very soon, causing a page fault because it's not in memory any more
  - In the worst case, each page that is paged out could be the one that is referenced next, leading to a high page fault rate
  - Ideally, keep around the pages that are about to be used next – this is the basis of the OPT algorithm in the next slide

OPT Page Replacement

- **OPT = Optimal**
  - Replace the page that will not be referenced for the longest time
  - Guarantees the lowest page-fault rate
  - Problem: requires future knowledge

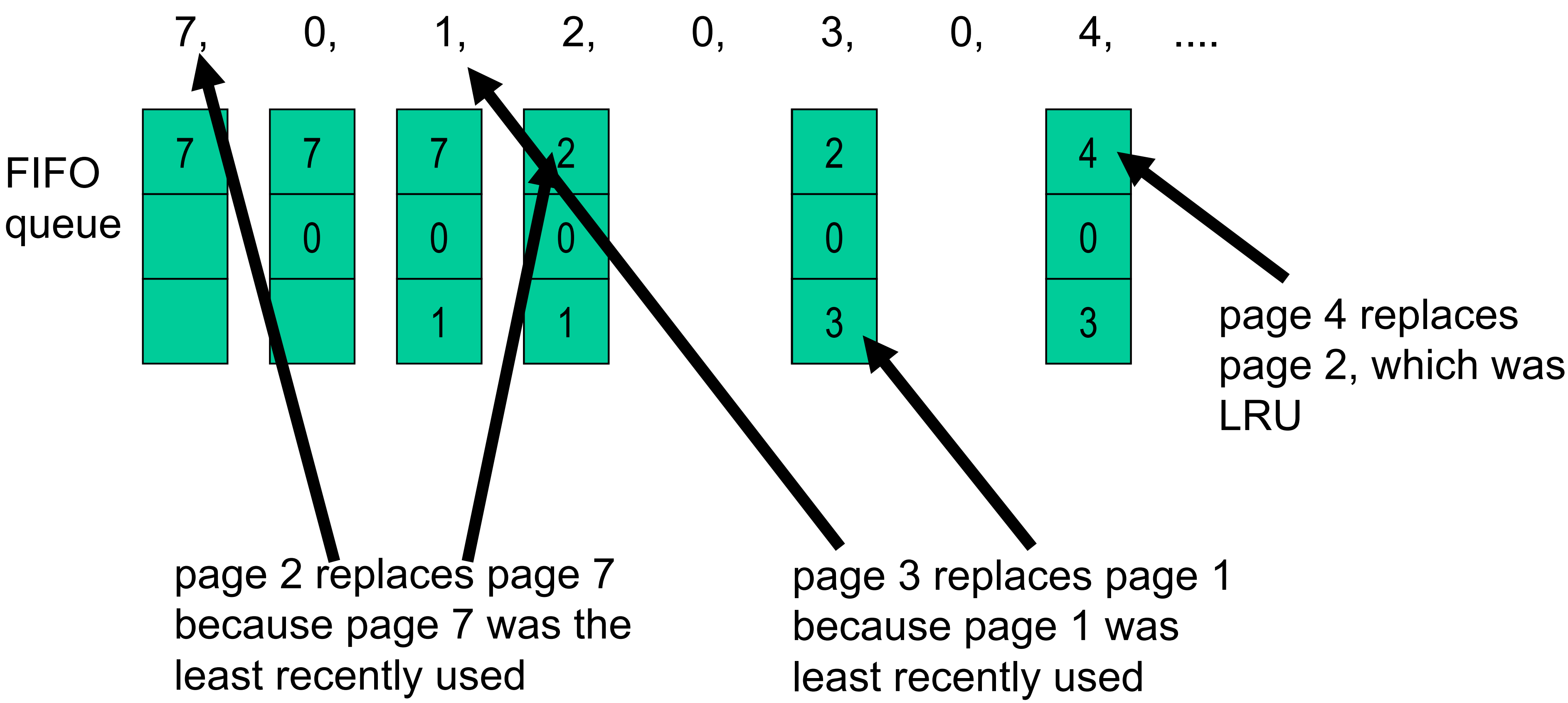


LRU Page Replacement

- **LRU = Least Recently Used**
  - Use the past to predict the future
    - if a page wasn’t used recently, then it is unlikely to be used again in the near future
    - if a page was used recently, then it is likely to be used again in the near future
    - so select a victim that was least recently used
  - Approximation of OPT
    - page fault rate LRU > OPT, but LRU < FIFO

LRU Page Replacement

- **LRU example**



Stack Algorithms

- **Key property:** **Set of pages in memory for *n* frames is always a subset of the set of pages that would be in memory with *n+1* frames, irrespective of the page reference string**

- **OPT and LRU are stack algorithms, while FIFO is not a stack algorithm**
  - Stack algorithms are a class of page replacement algorithms that do not suffer from Belady's anomaly
  - See textbook for explanation (Page 417)

LRU Implementation Options

- **Keep a history of past page accesses**
  - the entire history (lots of memory)
  - a sliding window

  - Complicated and slow

- Variations of LRU are popular

LRU Implementation Options

- **Timers**
  - keep an actual time stamp for each page as to when it was last used
  - Problem: expensive in delay (consult system clock on each page reference), storage (at least 64 bits per absolute time stamp), and search (find the page with the oldest time stamp)

LRU Implementation Options

- **Counters**
  - Approximate time stamp in the form of a counter that is incremented with any page reference, i.e. each page's counter must be incremented on each page reference
  - Counter is stored with that entry in the page table. Counter is reset to 0 on a reference to a page.
  - Problem: expensive
    - Must update each page's counter (on each reference)
    - Must search list

LRU Implementation Options

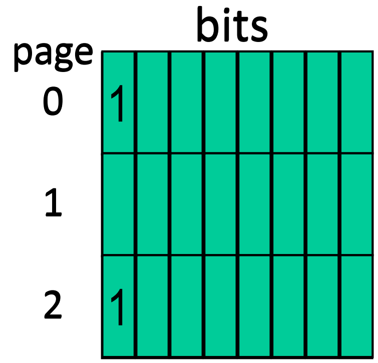
- **Linked List**
  - whenever a page is referenced, put it on the end of the linked list, removing if it from within the linked list if already present in list
  - Front of linked list is LRU
  - Problem: managing a (doubly) linked list and rearranging pointers becomes expensive
- **Similar problems with a *Stack* data structure**

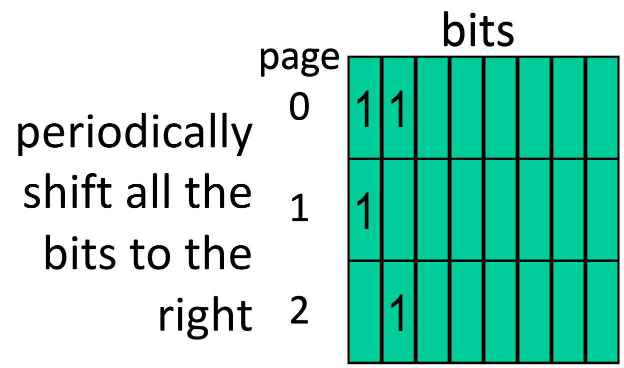
LRU approximation algorithms

- **Add an extra HW bit called a *reference bit***
  - This is set any time a page is referenced (read or write)
  - Allows OS to see what pages have been used, though not fine-grained detail on the order of use
  - Reference-bit based algorithms only *approximate* LRU, i.e. they do not seek to exactly implement LRU
  - 3 types of reference-bit LRU approximation algorithms:
    - Additional Reference-Bits Algorithm
    - Second-Chance (Clock) Algorithm
    - Enhanced Clock Algorithm with Dirty/Modify Bit

LRU approximation algorithms

- **Additional Reference-Bits Algorithm**
  - Record the last 8 reference bits for each page
  - Periodically a timer interrupt shifts right the bits in the record and puts the reference bit into the MSB
  - Easy to compare times:
    - 11000100 > 01110111
    - first has been used more recently than second
  - So LRU = lowest valued record





Reference-bit based LRU approximation algorithms

- **Second-Chance Algorithm**
  - In-memory pages + reference bits conceptually form a *circular* queue; a hand points to a page.
  - If page pointed to has R = 0, it is replaced and the hand moves forward.
  - Otherwise, set R = 0; hand moves forward and checks the next page.

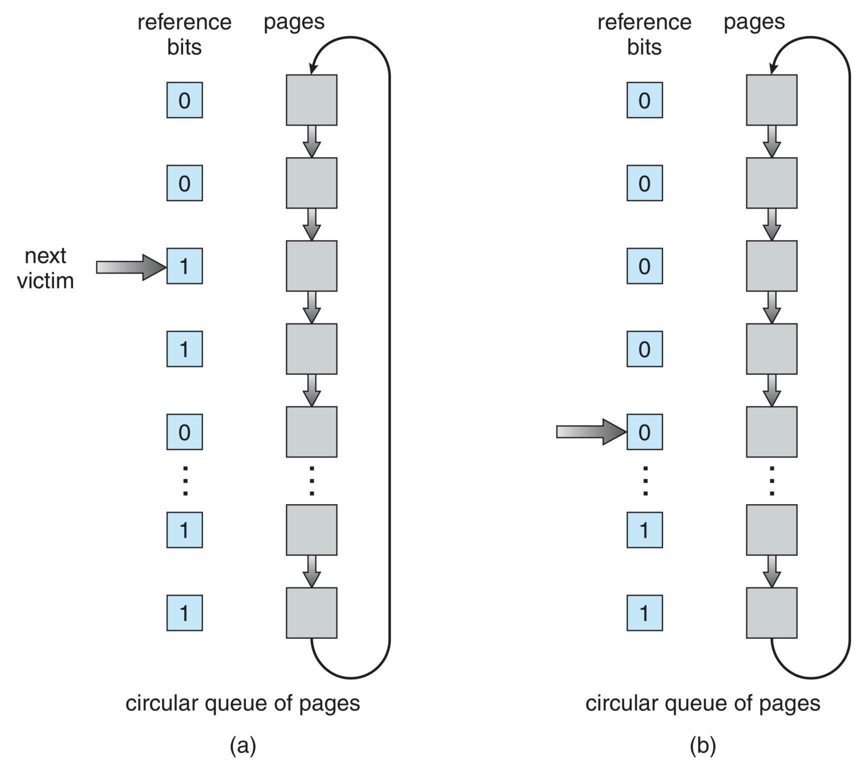


Figure 9.17 Second-chance (clock) page-replacement algorithm.

Second-Chance (Clock) Algorithm

- Advantages:
  - simple to implement (one pointer for the clock hand + 1 ref bit/page)
    - Note the circular buffer is actually just the page table with entries where the valid bit is set, so no new circular queue data structure need be created)
  - fast to check reference bit
  - usually fast to find first page with a 0 reference bit
  - approximates LRU
- Disadvantages:
  - in the worst case, have to rotate through the entire circular buffer once before finding the first victim frame

Reference-bit based LRU approximation algorithms

- **Enhanced Second-Chance (Clock) Algorithm**
  - Add a dirty/modify bit to the reference bit and consider them as a pair
  - Reference bit is cleared periodically
  - When selecting a victim, rotate a current pointer or clock hand through the queue as in the clock algorithm, and replace the first page encountered in the lowest nonempty class
  - Four classes are formed
    - Class 0: R = 0; M = 0. (Least heavily used class)
    - Class 1: R = 0; M = 1.
    - Class 2: R = 1; M = 0.
    - Class 3: R = 1; M = 1. (Most heavily used class)

Counting-Based Page Replacement

- **Keep a counter of number of page accesses for each page since its introduction, which is an activity or popularity index**
- **Most Frequently Used**
  - Replace page with highest count
  - Assumes the smallest counts are most recently loaded
- **Least Frequently Used**
  - Replace page with lowest count
  - What if a page was heavily used in the beginning, but not recently?
    - Age the count by shifting its value right by 1 bit periodically - this is exponential decay of the count.
- **These methods are not commonly used**

Approximation of OPT Algorithm

- **We use the past to approximate the future**

- **We can be smarter about looking at the past**
  - Not just a sequence, but a pattern of use
    - use this to create a better approximation of page use
  - Take advantage of the locality
  - Create a ***working set*** of pages

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of
Colorado Boulder