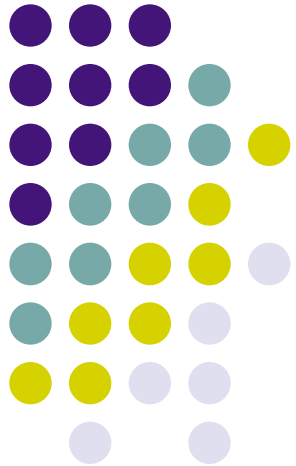
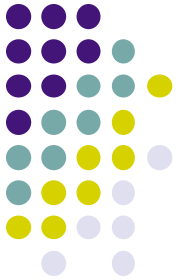
Page Replacement, Midterm Review 2

ECE469, March 24

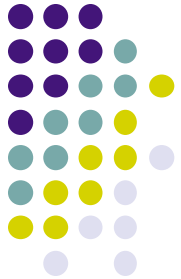
Yiying Zhang



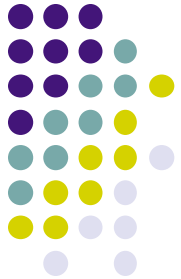
Practice Exam



break

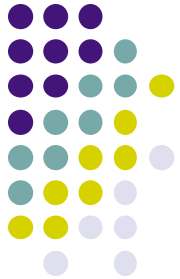


Linus Torvalds



- Linux
- git

Andrew Tanenbaum

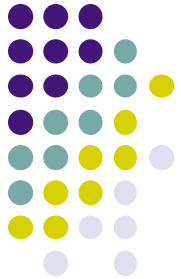


- Minix
- OS textbooks

Reading

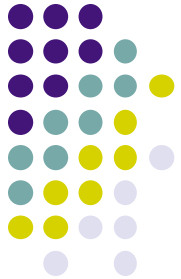
- Chapter 9

[lec16] The Policy Issue: Page selection and replacement



- Once the hardware has provided basic capabilities for VM, the OS must make two kinds of decisions
  - Page selection: *when* to bring pages into memory
  - Page replacement: *which* page(s) should be thrown out
    - Try to kick out the least useful page

[lec 17] Page replacement problem



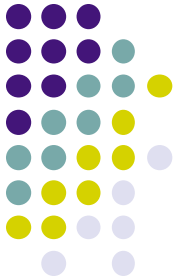
Definition:

- Expect to run with all physical pages in use
- Every "page-in" requires an eviction to swap space
- How to select the victim page?

- Performance goal:
  - Give a sequence of memory accesses, minimize the # of page faults
  - → given a sequence of virtual page references, minimize the # of page faults

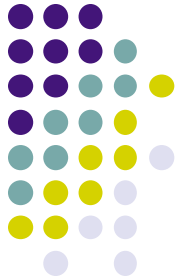
- Intuition: kick out the page that is least useful
- Challenge: how do you determine "least useful"?
  - Even if you know future?

Least Recently Used (LRU)



- Algorithm
  - Replace page that hasn't been used for the longest time
- Advantage: with locality, LRU approximates Optimal
- Question
  - What hardware mechanisms are required to implement *exact* LRU?

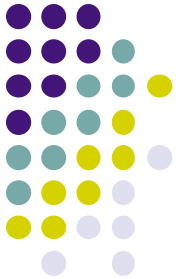
Implementing LRU: hardware

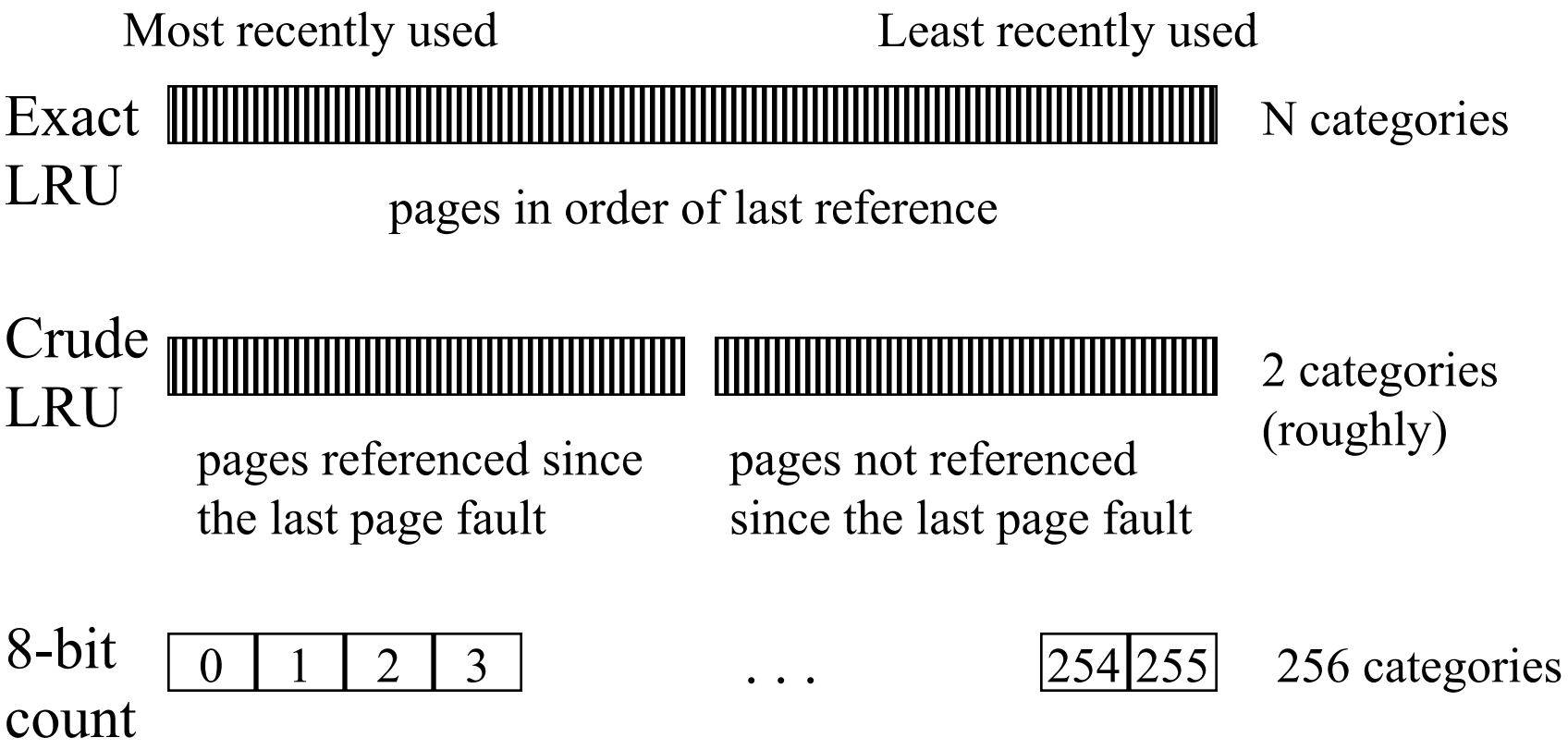


- A counter for each page
- Every time page is referenced, save system clock into the counter of the page
- Page replacement: scan through pages to find the one with the oldest clock

- Problem: have to search all pages/counters!

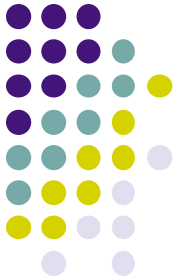
Approximate LRU





Keep 8-bit counter for each page in a table in memory

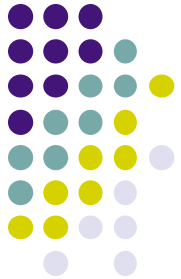
Approximate LRU



Interval 1	Interval 2	Interval 3	Interval 4	Interval 5
00000000	00000000	10000000	01000000	10100000
00000000	10000000	01000000	10100000	01010000
10000000	11000000	11100000	01110000	00111000
00000000	00000000	00000000	10000000	01000000

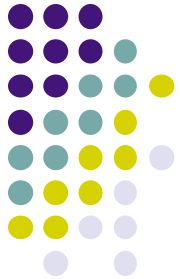
- Algorithm
  - At regular interval, OS shifts reference bits (in PTE) into counters (and clear reference bits)
  - Replacement: Pick the page with the "smallest counter"
- How many bits are enough?
  - In practice 8 bits are quite good
- Pros: Require one reference bit, small counter/page
- Cons: Require looking at many counters (or sorting)

Implementing LRU: software

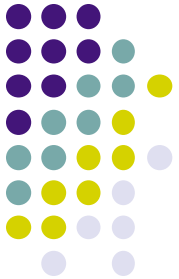


- A doubly linked list of pages
- Every time page is referenced, move it to the front of the list
- Page replacement: remove the page from back of list
  - Avoid scanning of all pages
- Problem: too expensive
  - Requires 6 pointer updates for each page reference info
  - High contention on multiprocessor

Not Recently Used (NRU)



- Algorithm
  - Randomly pick a page from the following (in this order)
    - Not referenced and not modified
    - Not referenced and modified
    - Referenced and not modified
    - Referenced and modified
- Pros
  - Easy to implement
- Cons
  - No frequency → Not very good performance
  - Takes time to classify



Thinking

- Performance design philosophy:
- Make common case fast!

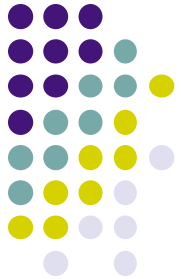
- Amadhl's Law

$$\text{Overall Speedup} = \frac{1}{(1-F)+\frac{F}{S}}$$

F = The fraction enhanced

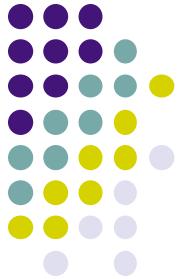
S = The speedup of the enhanced fraction

Factors that affect paging performance

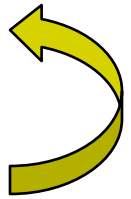


- # of memory misses
  - So far, have talked about algorithms to reduce misses (increase hit rate)
- Cost of a memory miss (replacement)

[lec17] Enhanced FIFO with 2nd-Chance Algorithm (used in Macintosh VM)

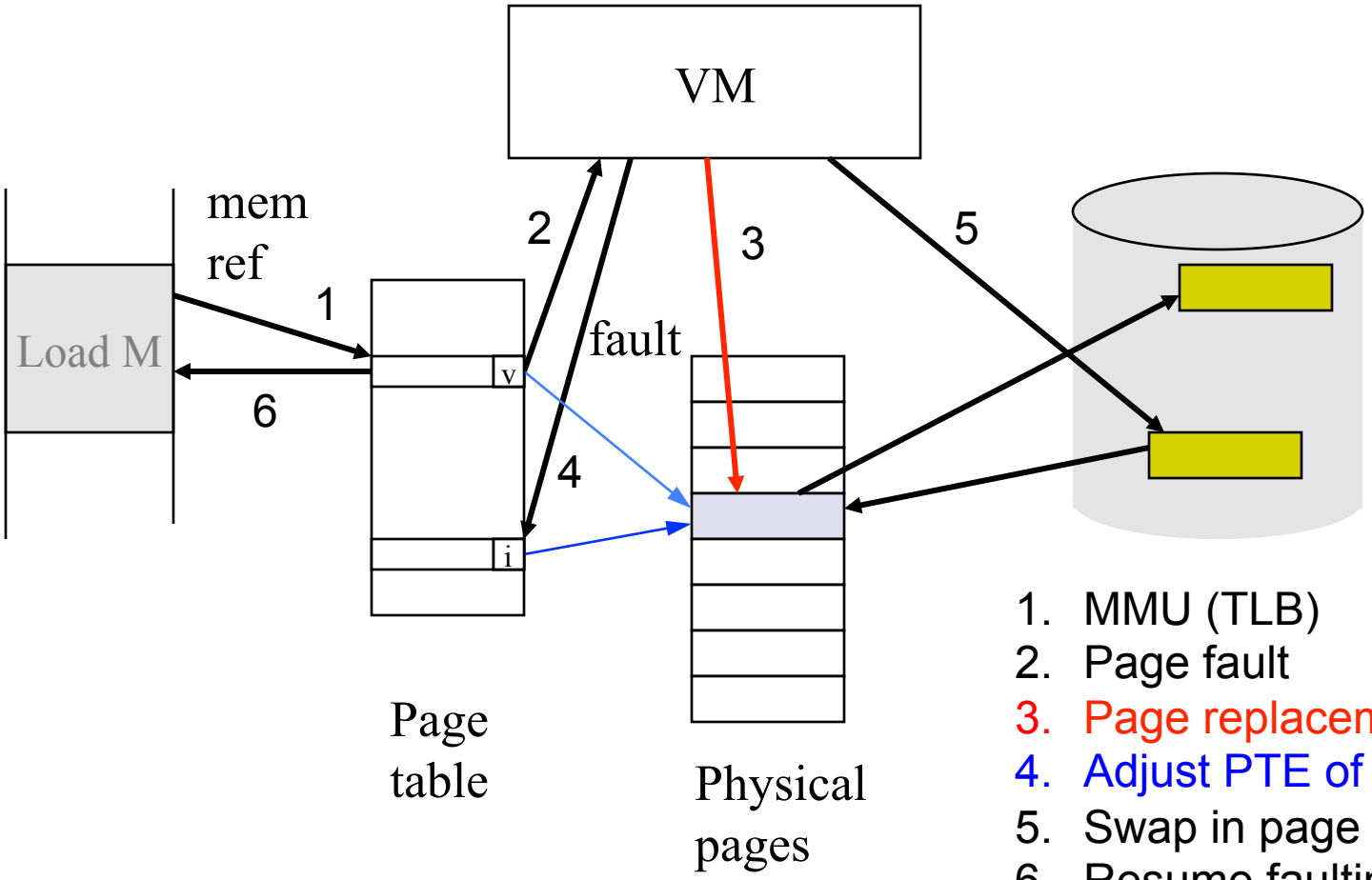


- Same as the basic FIFO with 2nd chance, except that it considers both (reference bit, modified bit)
  - (0,0): neither recently used nor modified (good)
  - (0,1): not recently used but dirty (not as good)
  - (1,0): recently used but clean (not good)
  - (1,1): recently used and dirty (bad)
  - When giving second chance, only clear reference bit



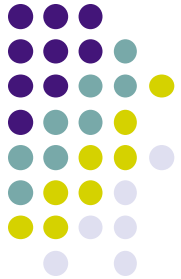
- Pros
  - Avoid write back
- Cons
  - More complicated, worse case scans multiple rounds

[lec16] Page Fault Handling in demand paging



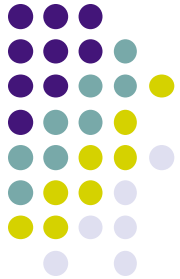
1. MMU (TLB)
2. Page fault
3. Page replacement
4. Adjust PTE of victim pg
5. Swap in page from swap
6. Resume faulting intr

Page out on critical path?



- If no free page pool, page in has to wait till page out is finished
  - Page fault handling time = proc. overhead + 2 * I/Os
- There is a chance of paged out page being referenced soon

Page buffering techniques



1. System maintains a pool of free pages
   - When a page fault occurs, victim page chosen as before
   - But desired page paged into a free page right away before victim page paged out
   - Dirty victim page written out when disk is idle

2. Also remember what was in free pages - maybe reused before reallocated