CS 4410
Operating Systems

Virtual Memory: Page Replacement

Summer 2013

Cornell University

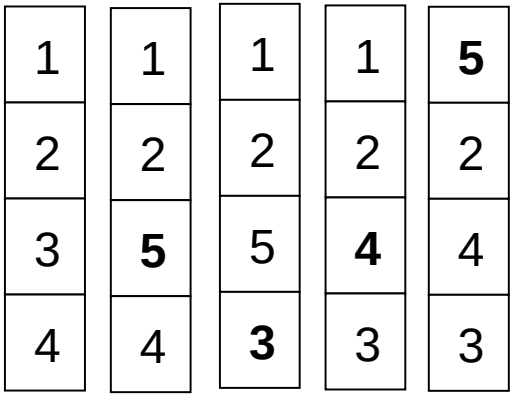
Today

- Is there any replacement algorithm that approximates OPT?
- LRU
- Other algorithms
- Thrashing
- Working Set
- Page Fault Frequency

LRU Page Replacement

- Replace the page that has not been used for the longest period of time.
- Use the recent past as an approximation of the near future.

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

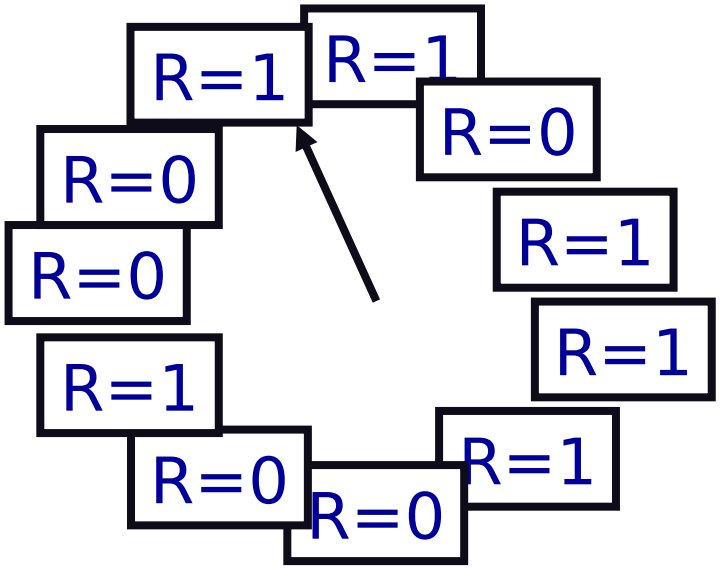


LRU Implementation

- Counters
  - Each page-table entry is associated with a time-of-use field.
  - CPU updates the field of a referenced page.
  - Scan the page table to find the LRU page.
- Stack
  - Whenever a page is referenced, it is removed from the stack and put on the top.
  - The LRU page is always at the bottom.
  - The update is expensive.

LRU: Clock Algorithm

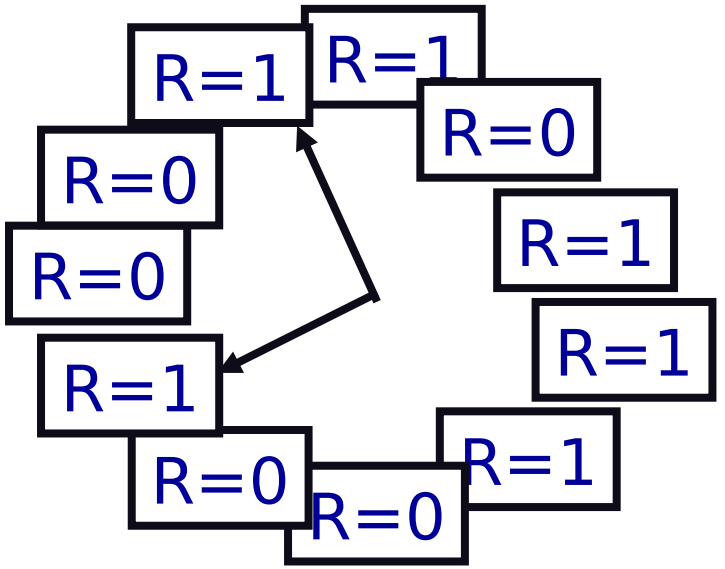
- Each page has a reference bit.
  - Set on use, reset periodically by the OS.
- Algorithm:
  - FIFO + reference bit (keep pages in circular list)
  - Scan: if ref bit is 1, set to 0, and proceed. If ref bit is 0, stop and evict.
- Problem:
  - Low accuracy for large memory



LRU: Clock Algorithm

- Solution: Add another hand
  - Leading edge clears ref bits
  - Trailing edge evicts pages with ref bit 0

- What if angle small?
- What if angle big?



Other Algorithms

- MRU: Remove the most recently touched page.
  - Works well for data accessed only once, e.g. a movie file.
  - Not a good fit for most other data, e.g. frequently accessed items.

- LFU: Remove page with lowest count.
  - No track of when the page was referenced.
  - Use multiple bits. Shift right by 1 at regular intervals.

- MFU: remove the most frequently used page

Global vs Local Allocation

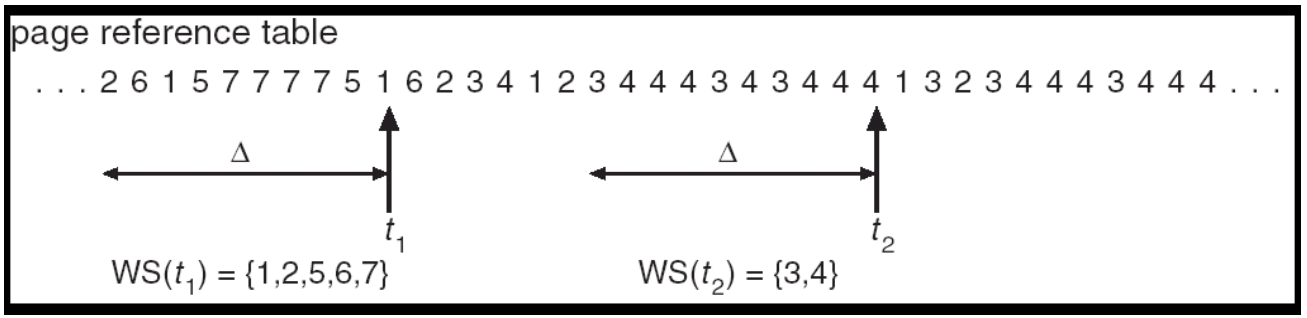
- Global replacement
  - Single memory pool for entire system.
  - On page fault, evict oldest page in the system.
  - Problem: lack of performance isolation.
- Local (per-process) replacement
  - Have a separate pool of pages for each process.
  - Page fault in one process can only replace pages from its own process.
  - Problem: might have idle resources.

Thrashing

- Def: Excessive rate of paging
  - May stem from lack of resources.
  - More likely, caused by bad choices of the eviction algorithm.
  - Keep throwing out page that will be referenced soon.
  - So, they keep accessing memory that is not there.
- Why does it occur?
  - Poor locality, past != future
  - There is reuse, but process does not fit model.
  - Too many processes in the system
- How can we solve this problem?

Working Set

- Estimate locality → Identify useful pages → Do not evict these pages.
- Working Set = An approximation of the program's **locality**.
- The set of pages in the most recent Δ page references.
- Example (Δ = 10):
  - t1 → WSS = {1,2,5,6,7}
  - t2 → WSS = {3,4}



Working Set

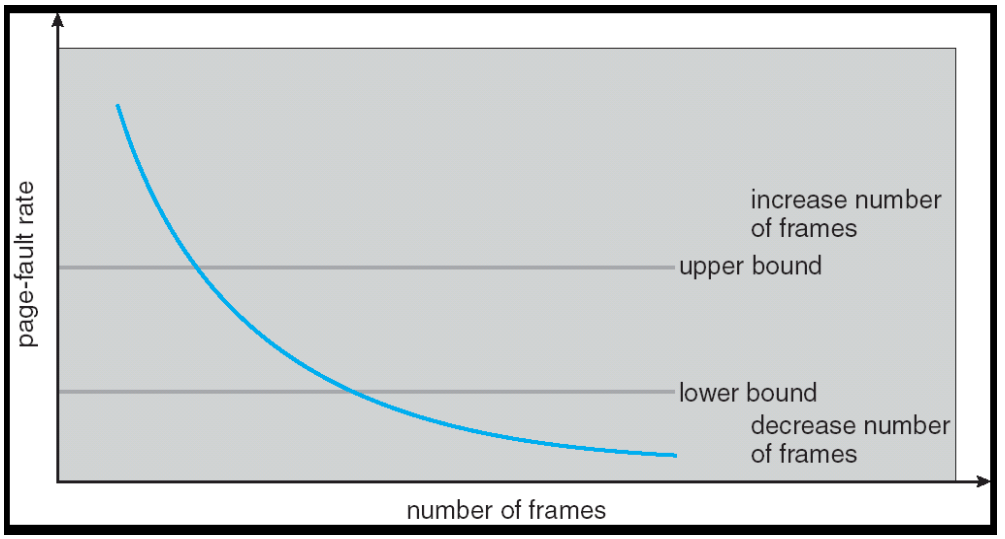
- How large the Δ should be?
- Total demand for frames:
  - D = Σ WSSi
  - If D > available memory frames:
    - Thrashing
    - The OS selects a process to suspend.
- Target:
  - No thrashing
  - High multiprogramming

Working Set Approximation

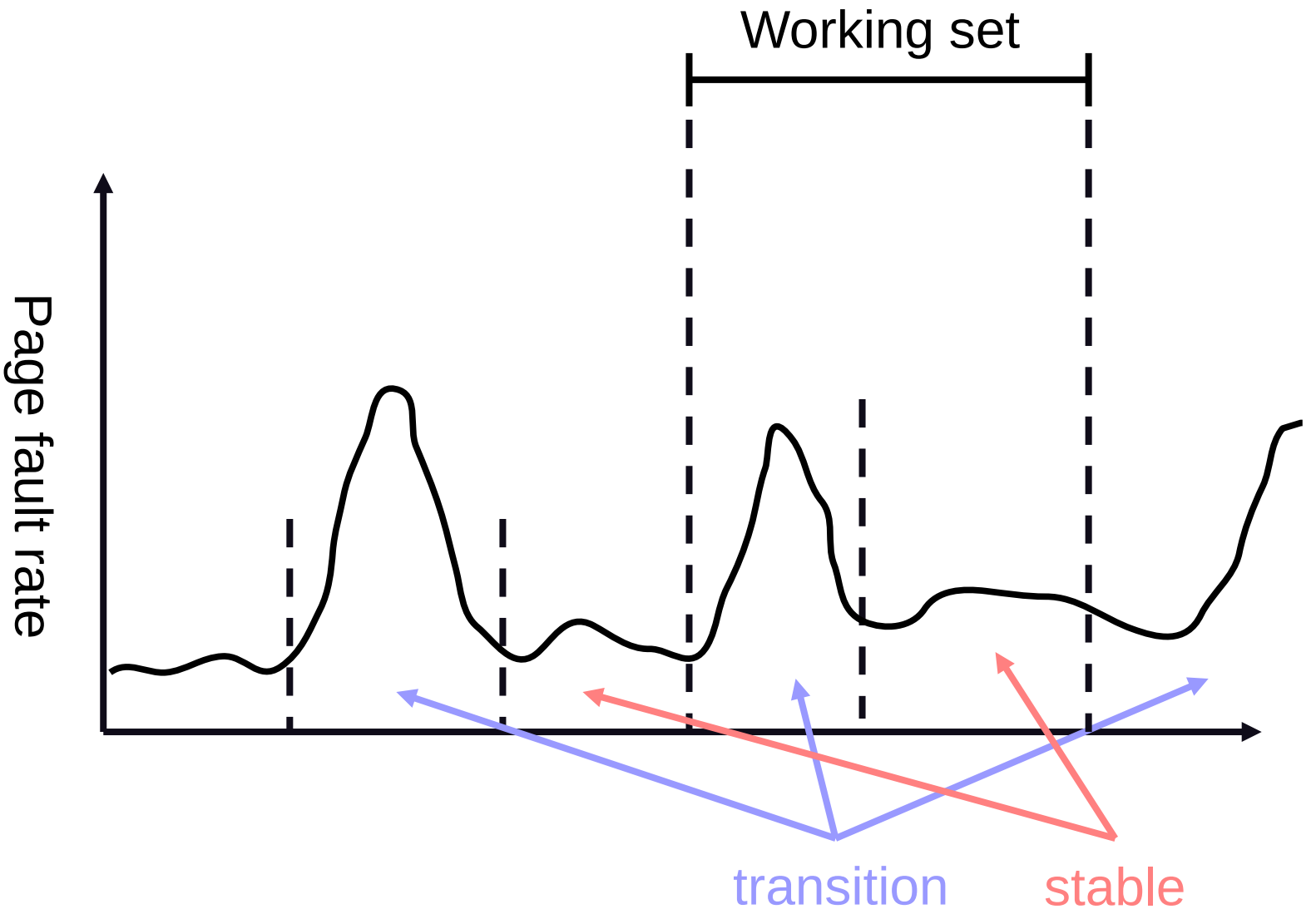
- Approximate with interval timer + a reference bit.
- Example: $\Delta = 10{,}000$
  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy and set the values of all reference bits to 0.
  - If one of the bits in memory = 1 → page in working set
- Why is this not completely accurate?
  - Cannot tell (within interval of 5000) where reference occurred.
- Improvement = 10 bits and interrupt every 1000 time units.

Page Fault Frequency

- Thrashing viewed as poor ratio of fetching to work.
- PFF = page faults / instructions executed.
- If PFF rises above threshold, process needs more memory.
  - Not enough memory on the system? → Swap out.
- If PFF sinks below threshold, memory can be taken away.



Working Sets and Page Fault Rates



Today

- Is there any replacement algorithm that approximates OPT?
- LRU
- Other algorithms
- Thrashing
- Working Set
- Page Fault Frequency