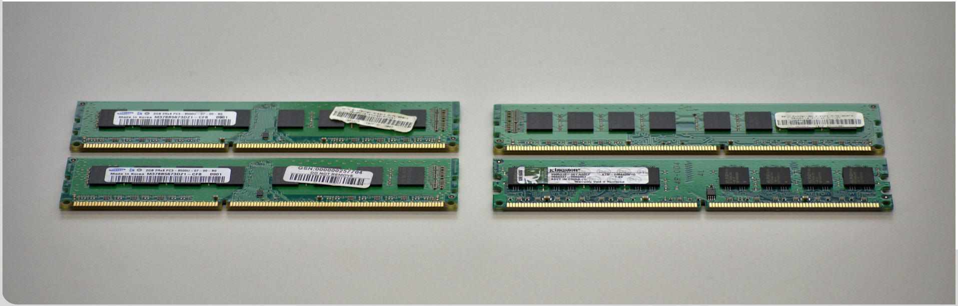
Operating Systems

09. Page Replacement

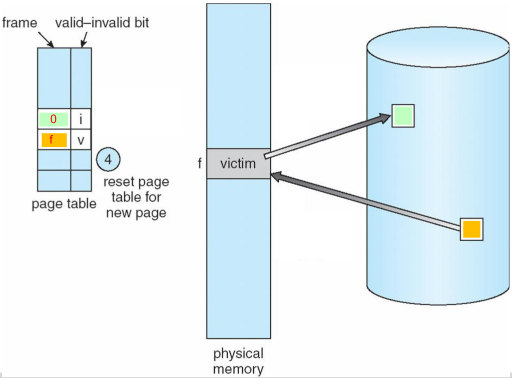
Prof. Dr. Frank Bellosa | WT 2020/2021

KARLSRUHE INSTITUTE OF TECHNOLOGY (KIT) – ITEC – OPERATING SYSTEMS



Naïve Page Replacement [SGG12]

- Save/clear victim page
  - Drop page if fetched from disk (e.g., code) and clean (PTE dirty bit)
  - Write back modifications if from disk and dirty
  - Write pagefile/swap partition otherwise (e.g., stack, heap memory)
- Unmap page from old AS
  - unset valid bit in PTE
  - flush TLB
- Prepare the new page
  - e.g., NULL page
  - e.g., load new contents
- Map the page frame into the new address space(s)
  - set valid bit in PTE
  - flush TLB

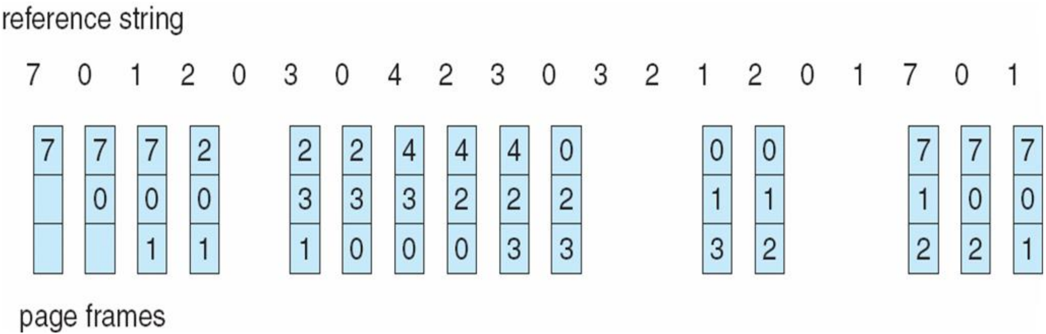


Page Buffering

- Problem: Naïve page replacement encompasses two I/O transfers swapping out (demand cleaning) and swapping the new page in
  - Both operations block the page fault from completing
- Goal: Reduce I/O from critical page fault path to speed up page faults
- Idea: Keep pool of free page frames (pre-cleaning)
  - On a page fault, use a page frame from the free pool
  - Run a daemon that cleans (write back changes), reclaims (unmap), and scrubs (zero out) pages for the free pool in the background
- Such a free pool smoothes out I/O and speeds up paging significantly
- Remaining problem: Which pages to select as victims?
  - Goal: Identify a page that has left the working set of its process to add it to the free pool
  - Success metric: Low overall page fault rate

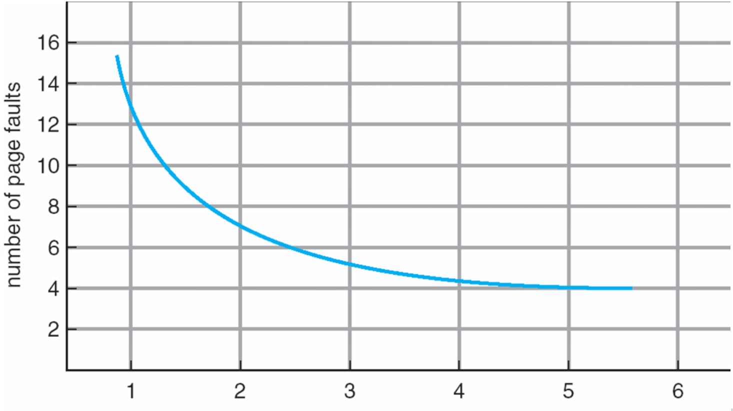
First-In-First-Out (FIFO) Page Replacement

- Evict the oldest fetched page in the system



Intuition: Page Fault Rate vs. Number of Frames

- Intuitively one would say that the page fault rate decreases when the amount of memory increases
- This is true most of the time, but not universally

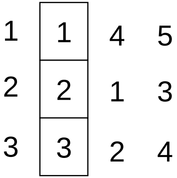


Belady's Anomaly [Bel66]

- Reference string for all our examples: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Belady's Anomaly
  - When using FIFO page replacement, for every number **N** of page frames you can construct a reference string that performs worse with **N+1** frames

→ With FIFO it is possible to get more page faults with more page frames

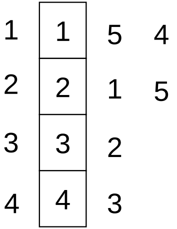
3 frames



1	1	4	5
2	2	1	3
3	3	2	4

9 page faults

4 frames

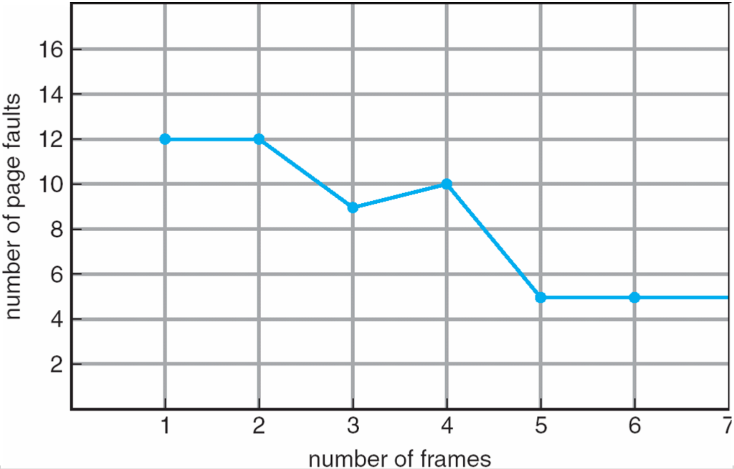


1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults

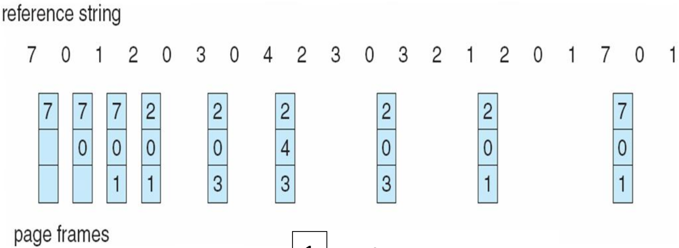
Belady’s Anomaly using FIFO page replacement

- More physical memory doesn’t always imply fewer faults [SGG12]

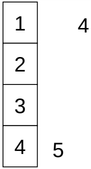


Oracle: Optimal Page Replacement (OPT)

- The optimal page replacement strategy is to replace the page whose next reference is furthest in the future [SGG12]



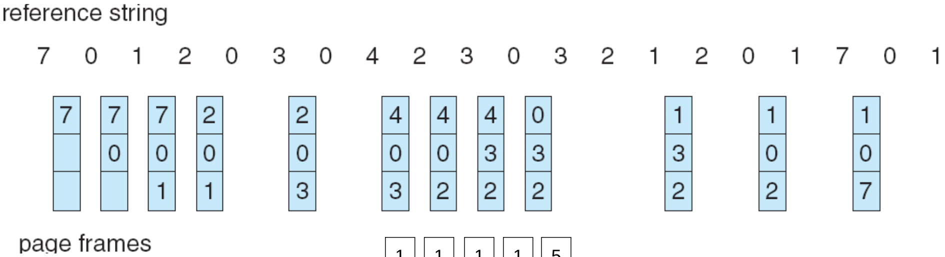
- Example with 4 frames:
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Cannot predict the future
  - Not suitable in practice
  - However: Good metric to check how well other algorithms perform


1 4
2
3
4 5

6 page faults

Least Recently Used (LRU) Page Replacement

- Goal: Approximate Oracle page replacement
- Idea: Past often predicts the future well
- Assumption: Page used furthest in past is used furthest in the future



- Reference string
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  8 page faults

1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3

LRU: Easy to understand, hard to implement well

- Cycle counter implementation
  - Have MMU write CPU’s time stamp counter to PTE on every access
  - On a page fault: Scan all PTEs to find oldest counter value
  - + Cheap at access if done in HW
  - − Memory traffic for scanning

- Stack implementation
  - Keep a doubly linked list of all page frames
  - Move each referenced page to tail of list
  - + Can find replacement victim in O(1)
  - − Need to change 6 pointers at every access

- No silver bullet
  - Observation: Predicting the future based on the past is not precise
  - Conclusion: Relax requirements – maybe perfect LRU is not needed?

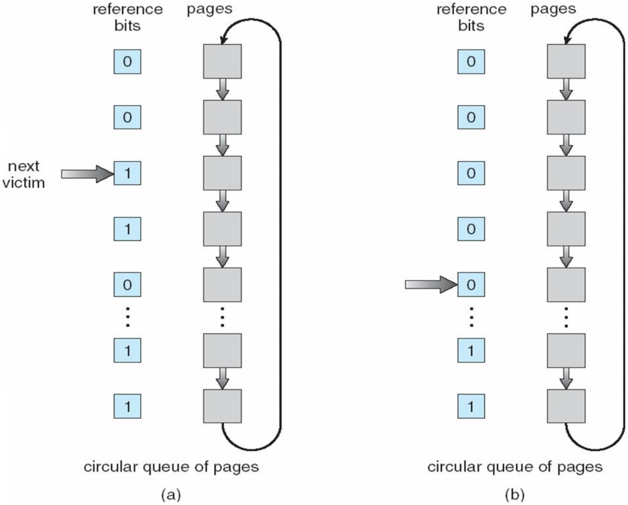
    → Approximate LRU

LRU Approximation: Clock Page Replacement

- Clock page replacement a.k.a. second chance page replacement
- Precondition: MMU sets reference bit in PTE at access [1]
  - Supported natively by most hardware (e.g., IA-32, x86-64, . . .)
  - Can easily be emulated in systems with software managed TLB (e.g., MIPS)
- Keep all pages in circular FIFO list
- When searching for a victim scan pages in FIFO's order
  - If reference bit is **0** → use page as victim and advance hand
  - If reference bit is **1** → set to **0** and continue scanning
- Large memory → most pages referenced before scanned
  - Use 2 arms: Leading arm clears reference bit, trailing arm selects victim

[1] Reference bit will be set for the first time by MMU after page fault when retrying access

Clock Page Replacement



Other replacement strategies

- Random eviction
  - Just pick a victim at random
  - Dirt simple and in reality not overly horrible

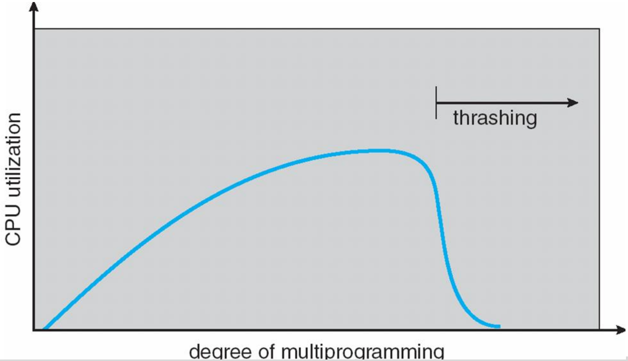
- Use larger counter: Use n-Bit reference counter instead of reference bit
  - Least frequently used (LFU)
    - Idea: Rarely used page is not in a working set

    → Replace page with smallest count
    - Problem: How to forget references of the past
  - Most frequently used (MFU)
    - Idea: The page with the smallest count was probably just brought in and will be used soon

    → Replace page with the largest count
  - Neither LFU nor MFU are used (no such hardware + far away from OPT)

Thrashing

- Thrashing: The system is busy swapping pages in and out
  - Each time one page is brought in, another page, whose contents will soon be referenced, is thrown out
  - Effect: Low CPU utilization, processes wait for pages to be fetched from disk
  - Consequence: OS “thinks” that it needs higher degree of multiprogramming [SGG12]



Reasons for Thrashing

- Memory too small to hold hot memory of a single process (the 10%)

- Access pattern has no temporal locality
  - Process doesn't follow the pareto principle
  - Past $\neq$ future

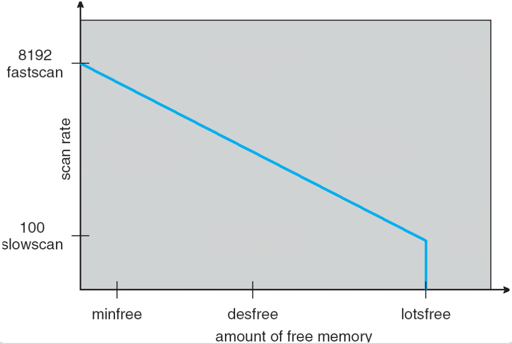
- Each process fits individually, but too many for system
  - Degree of multiprogramming too high

- Page replacement policy doesn't work well

Solaris

- Maintains a list of free pages to assign to faulting processes
- Paging is performed by *pageout* process
  - Scans pages using modified clock algorithm
  - *Scanrate* ranges from *slowscan* to *fastscan*
- Free memory thresholds determine the behavior of *pageout*
  - *Lotsfree*: Begin paging
  - *Desfree*: Increase paging
  - *Minfree*: Begin swapping

[SGG12]



References I

[Bel66] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[SGG12] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 9th edition, 2012.