Page Replacement Algorithms

How do we choose a frame to swap out?

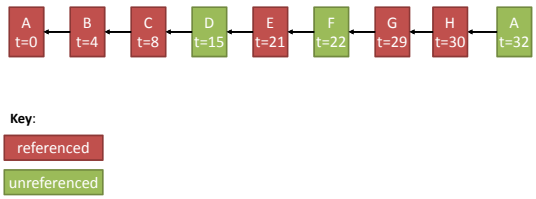
Page Replacement Algorithms

- Optimal
  - Evict the page that won't be needed until furthest in the future
- Not Recently Used (NRU)
  - Evict the page that is the oldest, preferring pages that are not dirty:

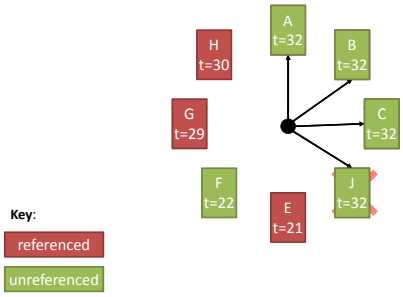
Preference	Referenced	Dirty
First Choice	0	0
	0	1
	1	0
Last Choice	1	1

- FIFO
  - First in, First out

Second Chance Page Replacement



Clock Algorithm



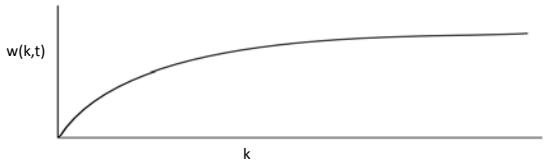
Least Recently Used (LRU)

Look to the past to predict the future

Aging Scheme

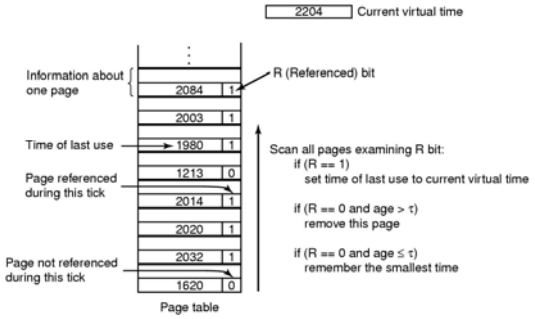
Referenced this tick	Tick 0	Tick 1	Tick 2	Tick 3	Tick 4
Page 0	10000000	11000000	11100000	01110000	10111000
Page 1	00000000	10000000	01000000	00100000	00010000
Page 2	10000000	01000000	00100000	10010000	01001000
Page 3	00000000	00000000	00000000	10000000	01000000
Page 4	10000000	01000000	10100000	11010000	01101000
Page 5	10000000	11000000	01100000	10110000	11011000

Working Set



- Working set is the set of pages used by the k most recent memory references
- w(k,t) is the size of the working set at time t

Working Set Page Replacement



Summary

Algorithm	Comment
OPT (Optimal)	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Crude
FIFO (First-In, First Out)	Might throw out useful pages
Second chance	Big improvement over FIFO
Clock	Better implementation of second chance
LRU (Least Recently Used)	Excellent, but hard to implement exactly
NFU (Not Frequently Used)	Poor approximation to LRU
Aging	Good approximation to LRU, efficient to implement
Working Set	Somewhat expensive to implement
WSClock	Implementable version of Working Set

Modeling Page Replacement

Youngest page

Oldest page

Page:	0	1	2	3	0	1	4	0	1	2	3	4
	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
			0	1	2	3	0	0	0	1	4	4

- FIFO replacement on reference string

  `0 1 2 3 0 1 4 0 1 2 3 4`

- Page replacements highlighted in orange

Belady's Anomaly

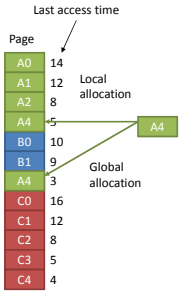
Youngest page

Oldest page

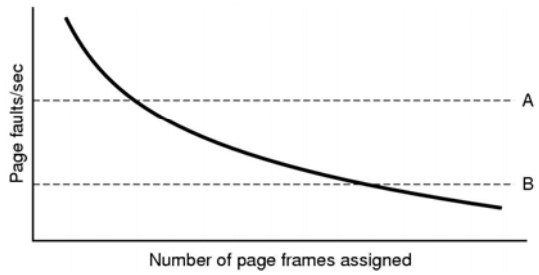
Page:	0	1	2	3	0	1	4	0	1	2	3	4
	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1

- Try to reduce the number of page faults by supplying more memory
  - Use previous reference string and FIFO algorithm
  - Add another page to physical memory (total 4 pages)

Local vs. Global Allocation



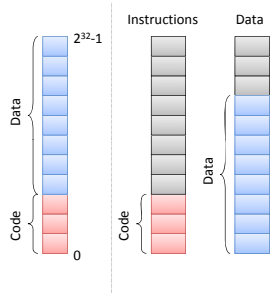
Page Fault Frequency



Page Size

- For larger pages
  - Smaller page tables
  - Less frames in memory (smaller degree of multiprogramming?)
  - Internal fragmentation
- For smaller pages
  - Bigger page table
  - More levels of page tables
  - Less wasted space

I- and D-Spaces



Page Sharing

Map multiple pages to a single frame

When to Write to Disk

Now or later?

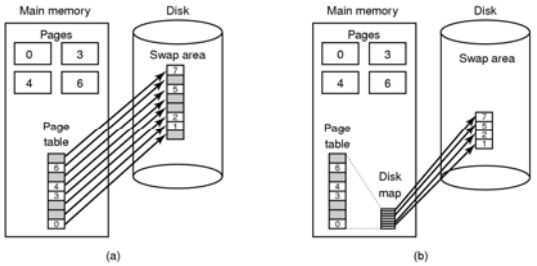
Implementation

- Process creation
- During process execution
- Page fault time
- Process termination time

Handling a Page Fault

1. Determine faulting virtual address
2. If the page is invalid, grow stack or heap, alternatively SEGFAULT on error
3. If physical memory is full, choose a frame to evict
4. Write frame to disk if dirty
5. Load requested page into now empty frame

Backing Store



Segmentation