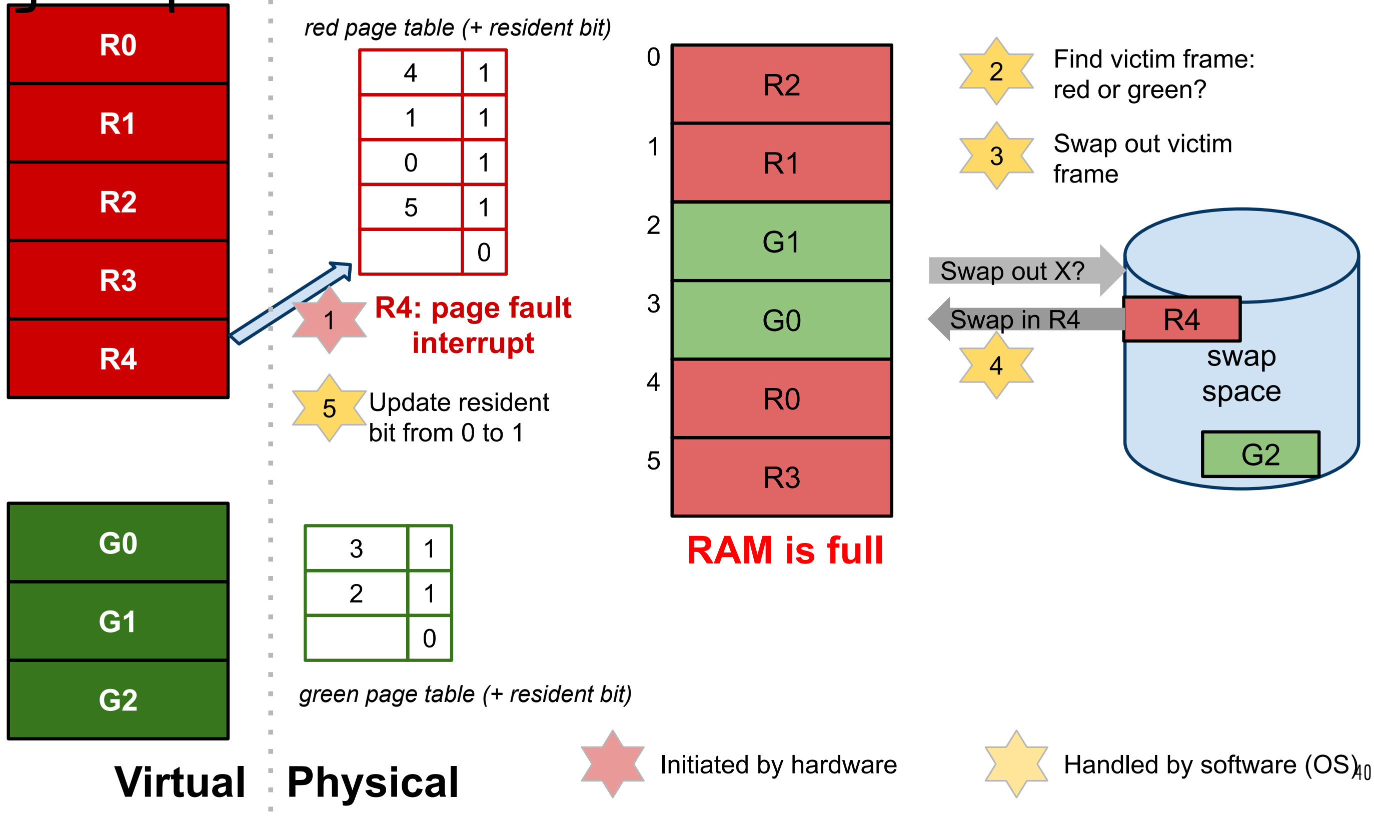
Page Replacement Algorithms

Preemptive CPU Scheduler: which process to run next when CPU is **vacant**

*Page Replacement: which process page to kick-out of RAM when RAM becomes **full***

Page Replacement: HW and SW coordination



Page Replacement

- Page Fault Interrupt triggers the OS to bring the missing page into RAM
  - But, no empty frame is available in RAM
- One of the frames (**victim frame**) must be overwritten
  - Whose frame?
  - Which frame?
  - Was the frame modified?
- Page Replacement = [Swap Out Victim Pg] + Swap In Missing Pg
- *Technicality:* **Page** Replacement or **Frame** Replacement?

Can we not swap out?

Can we just *overwrite* the victim frame?

Dirty Bit / Modify Bit

- A(nother) bit in the page table to indicate if the corresponding page has been altered since the last time it was swapped in
    - The modify/dirty bit is **automatically set by hardware (**i.e. data written to a page)
- The the modify/dirty bit is FALSE, there is no need to write a victim page to the swap space

Algorithms for Demand Paging

- Objective: **Minimize Page Fault Rate**
- Frame Allocation Algorithm
  - Determine **how many** frames to allocate to each process
  - Easier task to solve
  - Consequence of poor decision?
- Page Replacement Algorithm
  - **Select a victim frame** (when handling page fault and memory is full)
  - Harder task to solve
  - Consequence of poor decision?

Page Replacement Algorithms

- Static Algorithms assume the **number of frames** allocated to a process is **fixed**
  - Local scope: victim page is selected from the process experiencing the fault
- Dynamic Algorithms **adjust number of frame allocations** as a process runs
  - Global scope: victim page is selected system wide

Static Page Replacement Algorithms

- FIFO
  - Belady's Anomaly
- Optimal Page Replacement
- LRU
  - Stack Algorithms
- LRU approximation
  - Several techniques for timestamp approximation, clock/second-chance algorithms
- Counting-Based
  - LFU: Least Frequently Used
  - MFU: Most Frequently Used

Replacement Policy

- Which page to replace?
- The page replaced should (ideally) by the page least likely to be referenced in the **near future**.
  - **Impossible** to know future behavior of our programs!
- Most algorithms predict future behavior based on past behavior

FIFO Page Replacement

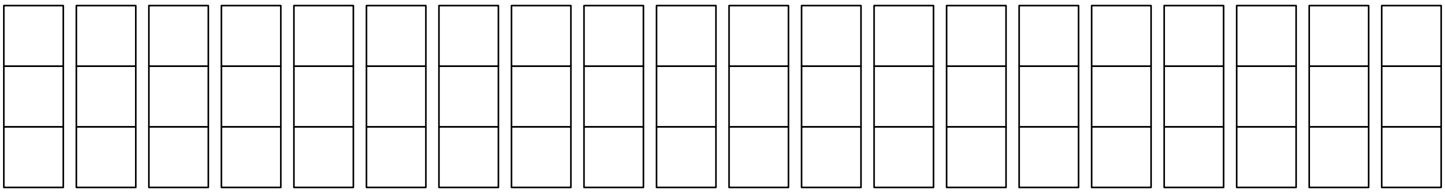
- Selection of victim: max age in RAM / oldest frame in RAM
  - Replace the page the has been in RAM the longest

retire long-time employees

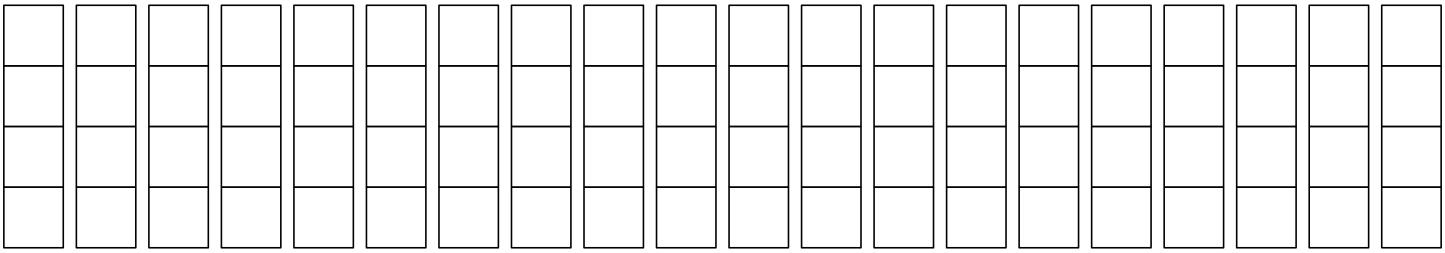
- Implementation: use a circular buffer to keep the resident **page numbers**
- Belady's Anomaly: number of page faults may increase when a process is allocated more frames
- Consequence of poor choice: the victim page may be referenced again very soon

FIFO (victim selection: **oldest** page)

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



FIFO: Belady's Anomaly

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

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

FIFO: Belady's Anomaly

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

9 Page Faults

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

10 Page Faults

Too few coat hangers
@
end of winter cleanup

Which coats to store away? ***rain coats*** *or* ***winter coats****?*

Optimal Page Replacement Algorithm

- Selection of Victim: **farthest next** (near future) reference
  - Replace the page that WILL NOT be used for the longest period of time
  - Impossible to implement, use only for theoretical analysis
- Does not suffer from Belady's anomaly
- Gives the lowest page fault rate

Optimal (victim: **farthest next** ref)

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Optimal but can’t predict *future* refs.
Impossible to implement

We know only *past* refs…

Optimal: farthest *next* reference (forward dist)

LRU: farthest *previous* reference (backward dist)

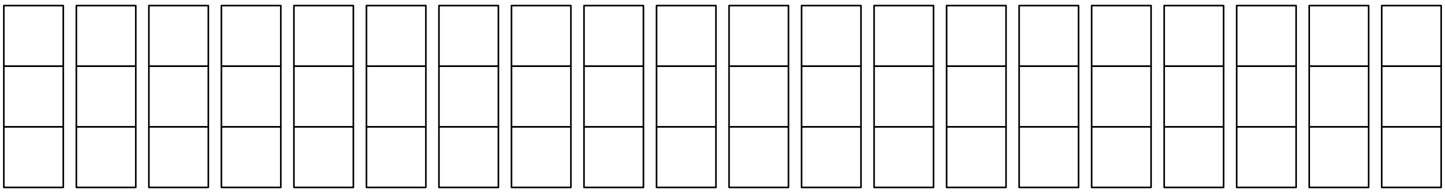
Least-Recently Used (LRU)

- Selection of Victim: **farthest previous** reference
  - Replace the page that HAS NOT been used for the longest period of time

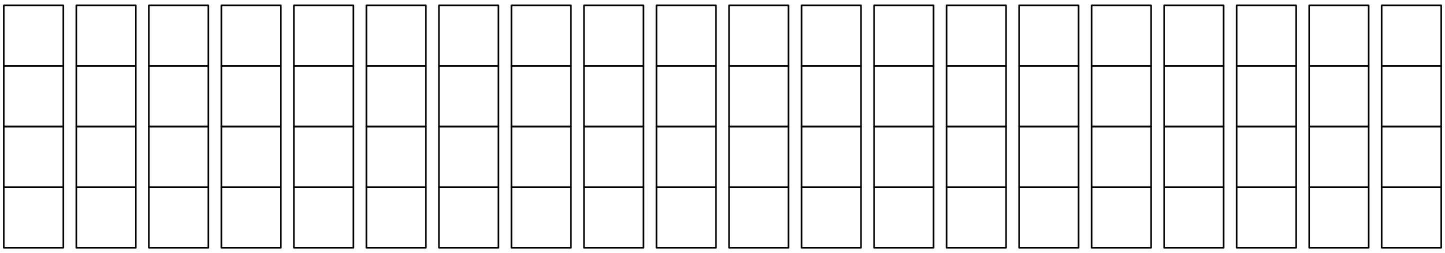
    Bookshelf clean up: C, C++, COBOL, Python, Ruby?
- Approximation to the Optimal Algorithm
  - **Backward distance is a good estimate for forward distance**
  - LRU does not suffer from Belady's Anomaly
- Implementation
  - What we need: timestamp to last access/reference (***updated by hardware***). Replace the page with the smallest timestamp

LRU (victim: **farthest past ref**)

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

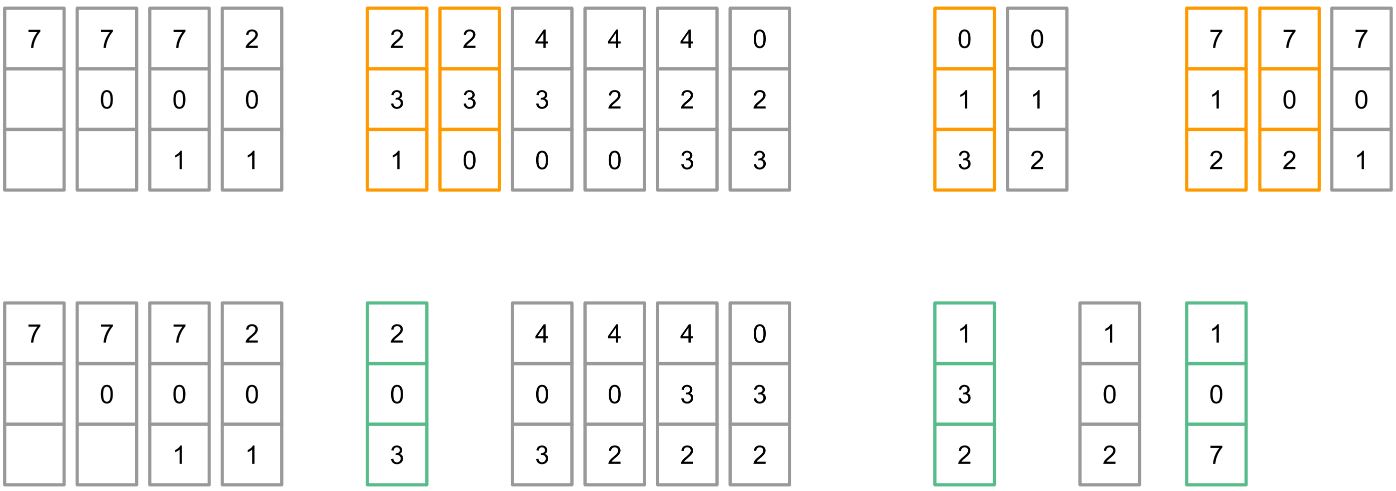


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



FIFO vs. LRU

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

LRU on Reversed Reference String

Original Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

Reversed Reference String

1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	0	2	1	0	7

LRU on a Reversed Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

Reversed Reference String

1	0	7	1	0	2	1	2	3	0	3	2	4	0	3	0	2	1	0	7
1	1	1			1			1	0			4	4	4		2	2		7
	0	0			0			3	3			3	0	0		0	0		0
		7			2			2	2			2	2	3		3	1		1

LRU vs Optimal

1. pf(**LRU**(S)): total number of page faults of LRU on reference string S
2. pf(**Opt**(Sr)): total number of page faults of Optimal on **reverse** of S
3. pf(**LRU**(S)) = pf(**Opt**(Sr))
4. For any ref string M pf(**LRU**(M)) ≥ pf(**Opt**(M)) Optimal is the best

Therefore

- pf(**LRU**(S)) ≥ pf(**Opt**(S)) = pf(**LRU**(Sr)) ⇒ pf(**LRU**(S)) ≥ pf(**LRU**(Sr))
- pf(**LRU**(Sr)) ≥ pf(**Opt**(Sr)) = pf(**LRU**(S)) ⇒ pf(**LRU**(Sr)) ≥ pf(**LRU**(S))

Consequently: pf(**LRU**(Sr)) = pf(**LRU**(S)) or **reversing the ref string** yields the **same page fault count** on LRU (and Optimal)

Page Replacement Algorithms: Implementation

Algorithm	Pros	Cons
FIFO	Easy to implement (queue of page numbers)	Belady's Anomaly
Optimal	No Belady's Anomaly	Impossible to implement
LRU	No Belady's Anomaly	

(Optimal and LRU →) **Stack Algorithm**

P(N) = set of pages resident in RAM when process is allocated N frames

Property of Stack Algorithm: P(N) ⊆ P(N + k) where k > 1

As the number of frames goes up, you never lose pages!

LRU as a Stack Algorithm