Local vs. Global Page Replacement

- Local: processes select victim among allocated frames allocated to them
  - Can lead to under utilization
- Global: Select any free frame, even if allocated to another process
  - Processes lose control over their own page fault rate

Brother, can you spare a frame?

FIFO

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Trace	a	b	c	d	a	b	c	d	a	b	c	d	
0		a	a	a	d	d	d	c	c	c	b	b	b
1			b	b	b	a	a	a	d	d	d	c	c
2				c	c	c	b	b	b	a	a	a	d
Faults	X	X	X	X	X	X	X	X	X	X	X	X	X

Brother, can you spare a frame?

FIFO

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Trace	a	b	c	d	a	b	c	d	a	b	c	d	
0		a	a	a	a	a	a	a	a	a	a	a	a
1			b	b	b	b	b	b	b	b	b	b	b
2				c	c	c	c	c	c	c	c	c	c
3					d	d	d	d	d	d	d	d	d
Faults	X	X	X	X	X								

So, what's wrong with global replacement?

Sharing Memory as a Cache

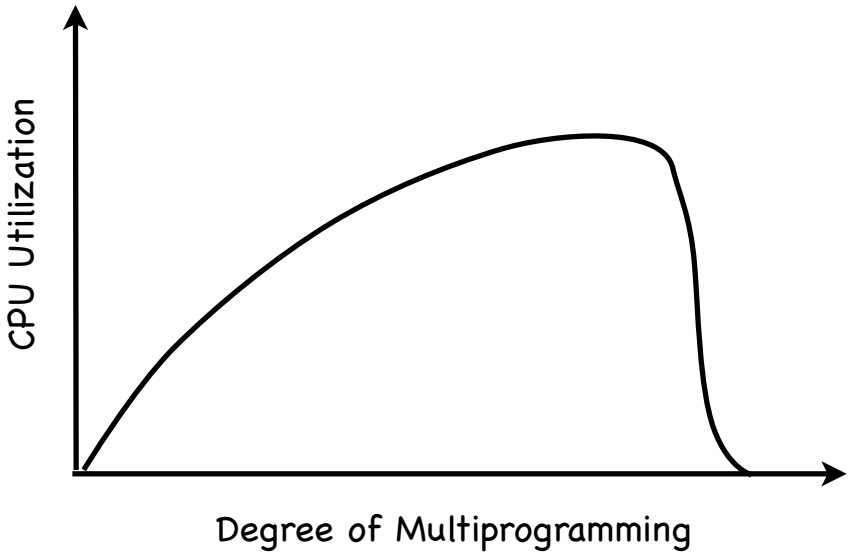
- Demand paging enables frames to cache part of a process VA space
- If the cache is large enough, hit ratio is high
  - few page faults
- What if there aren't enough frames to go around?
  - should decrease degree of multiprogramming
    - swapped out process can then release its frames

Instead...

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!

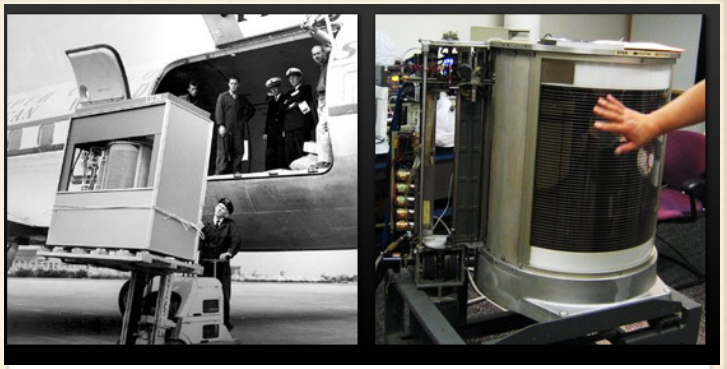
Instead...

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!
- Thrashing
  - process spends all its time swapping pages in and out



Why "thrashing"?

"Thrash" dates from the 1960's, when disk drives were as large as washing machines. If a program's working set did not fit in memory, the system would need to shuffle memory pages back and forth to disk. This burst of activity would violently shake the disk drive.



The first hard disk drive—the IBM Model 350 Disk File (came w/IBM 305 RAMAC, 1956).

Total storage = 5 million characters (just under 5 MB).

http://royal.pingdom.com/2008/04/08/the-history-of-computer-data-storage-in-pictures/

Locality of Reference

- If a process access a memory location, then it is likely that
  - the same memory location is going to be accessed again in the near future (temporal locality)
  - nearby memory locations are going to be accessed in the future (spatial locality)
- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions

Tracking Locality

- When a process executes it moves from locality (set of pages used together) to locality

  the size of the process' locality (a.k.a. its working set) can change over time

- Goal: track the size of the process' working set, dynamically acquiring and releasing frames as necessary

The Working Set Model

- Choose Δ page references as WS sliding window

  track WS for the last Δ page references

- WSS_i = # of distinct pages referenced by in latest Δ references

  Δ too small does not cover locality

  Δ too large covers many localities

- Thrashing if WSS_i > # frames

  if so, swap out one of the processes; free its frames

- If enough free frames, increase degree of multiprogramming

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X										

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓									

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓								

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓	X							

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓	X	✓						

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓	X	✓	X					

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓	X	✓	X	✓				

WS Page Replacement

$\Delta = 4$

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace	e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory: Page a														
Page b														
Page c														
Page d														
Page e														
Faults	X	X	X	X	✓	✓	X	✓	X	✓	✓			

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓	X	✓	X	✓	✓	X		

WS Page Replacement

$\Delta = 4$

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		e	d	a	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X	✓	✓	X	✓	X	✓	✓	X	X	

Approximating the Working Set

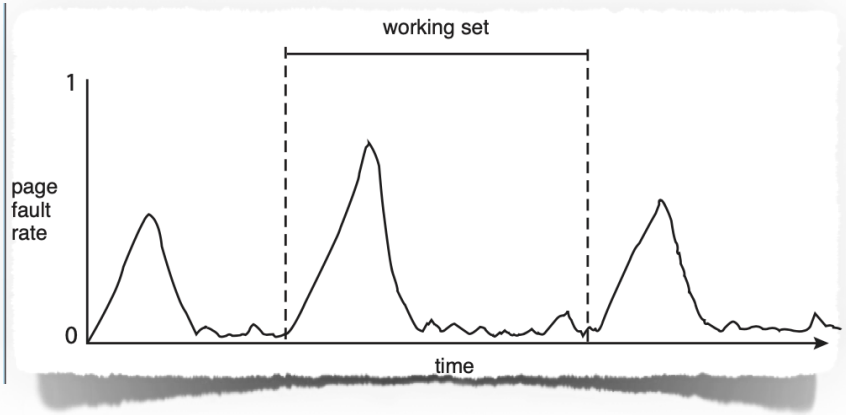
- Keep a k-bit tag in each table entry
- Set a timer interrupt to fire every Δ/k page references
  - granularity at which the window shifts
- On interrupt
  - Shift tag right one bit
  - Copy REF bit in tag's leftmost bit and clear REF
  - Add to a free list any page whose tag is zero

  Note: Must scan all frames!
- When a frame is needed, use the free list
  - if empty, pick any frame

Working Sets and Page Fault Rates

- As the working set changes, the page fault rate increases

  a steep increase in the page fault rate indicates a shift in locality, which may require a different WS



- Idea: Change the number of frames allocated to a process in response to changes to its PF rate

  as long as the working sets of all processes that are currently in memory does not exceed the size of physical memory, no thrashing

PFF (Page Fault Frequency) Algorithm

Keep time of last page fault

On page fault:

threshold

if , then unmap all pages not referenced in

else add faulting page to the working set

PFF Page Replacement

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Trace		d	a	e	c	c	d	b	c	e	c	e	a	d	
Pages in Memory	Page a														
	Page b														
	Page c														
	Page d														
	Page e														
Faults		X	X	X	X			X		X			X	X	
		0	1	1	1			3		2			3	1	

Taming Belady:
Stack Page Replacement Policies

- Let M(m,r) be the set of virtual pages in physical memory given that there there are m frames for trace r
- A stack page replacement policy has the property that, for all number of frames m and for all traces r

M(m,r) is a subset of M(m+1, r)

Stack page replacement policies do
not suffer from Belady's anomaly:
more frames —> not more misses

FIFO: m=3 vs m=4

Time		0	1	2	3	4	5	6	7	8	9	10	11	12
Trace		a	b	c	d	a	b	e	a	b	c	d	e	
Page Frames	0		a	a	a	d	d	d	e	e	e	e	e	e
	1			b	b	b	a	a	a	a	a	c	c	c
	2				c	c	c	b	b	b	b	b	d	d
Faults		X	X	X	X	X	X	X			X	X	X	

Time		0	1	2	3	4	5	6	7	8	9	10	11	12
Trace		a	b	c	d	a	b	e	a	b	c	d	e	
Page Frames	0		a	a	a	a	a	a	e	e	e	e	d	d
	1			b	b	b	b	b	a	a	a	a	a	e
	2				c	c	c	c	c	c	b	b	b	b
	3					d	d	d	d	d	d	c	c	c
Faults			X	X	X	X			X	X	X	X	X	X

Stack "subset property" violated

Theorem: Stack Property holds for LRU and MRU

- By definition:

  For LRU: M(m+1,r) contains m most frequently used frames, so M(m,r) is a subset of M(m+1,r)

  Similar argument holds for MRU, LFU

Theorem: Stack Property holds for OPT

- Proof is non-trivial!

  You can find more in the paper that introduced the notion of stock replacement policies

  R.L. Mattson, J. Gecsei, D.R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies" in IBM Systems Journl, vol. 9, no. 2, pp. 78-117, 1970