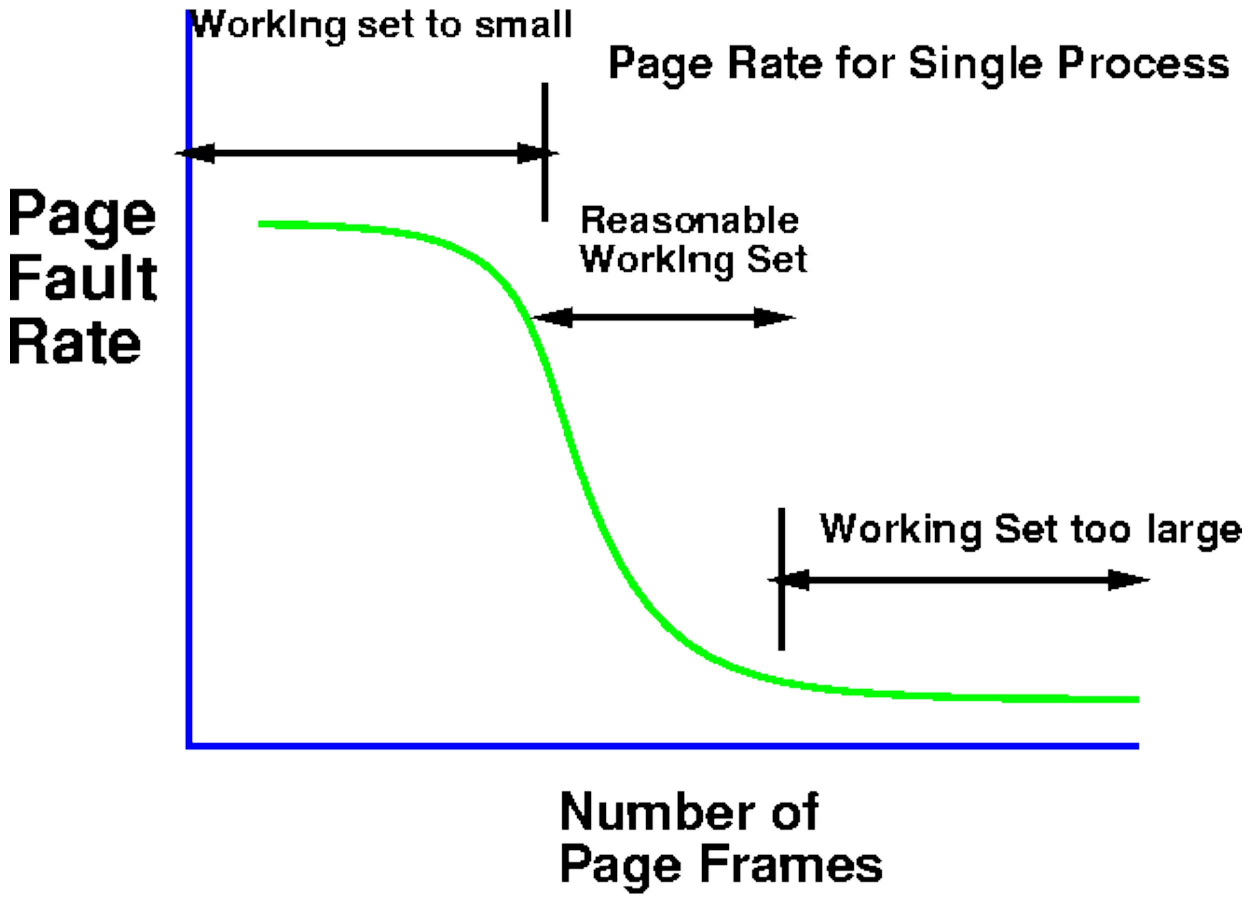
Virtual Memory: appendix

Page Fault Frequency (PFF) algorithm

- Approximation of pure Working Set
  - Assume that working set strategy is valid; hence, properly sizing the resident set will reduce page fault rate.
  - Let's focus on process fault rate rather than its exact page references
  - If process page fault rate increases beyond a maximum threshold, then increase its resident set size.
  - If page fault rate decreases below a minimum threshold, then decrease its resident set size

  ➔ Without harming the process, OS can free some frames and allocate them to other processes suffering higher PFF

Page Fault Frequency Working Set



Exploiting Locality

- Temporal locality
  - Memory accessed recently tends to be accessed again soon
- Spatial locality
  - Memory locations near recently-accessed memory is likely to be referenced soon

Exploiting Locality

- Locality helps reduce the frequency of page faults
  - Once something is in memory, it should be used many times
- Page fault rate depends on many things
  - The amount of locality and reference patterns in a program
  - The page replacement policy
  - The amount of physical memory and application memory footprint

Page Replacement Strategies

- OPT
  - Evict page that won't be used for the longest time in the future
- Random page replacement
  - Choose a page randomly
- FIFO - First in First Out
  - Replace the page that has been in primary memory the longest
- LRU - Least Recently Used
  - Replace the page that has not been used for the longest time
- LFU - Least Frequently Used
  - Replace the page that is used least often
- NRU - Not Recently Used
  - An approximation to LRU.
- Working Set
  - Keep in memory those pages that the process is actively using.

Page Replacement Strategies

- The Optimal Algorithm
  - Among all pages in frames, evict the one that has its next access farthest into the future
  - Can prove formally this does better than any other algorithm
  - OPT is useful as a “yardstick” to compare the performance of other (implementable) algorithms against
  - Realistic?

The Optimal Page Replacement Algorithm

- Idea
  - Select the page that will not be needed for the longest time <u>in the future</u>

Time		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Requests			**c**	**a**	**d**	**b**	**e**	**b**	**a**	**b**	**c**	**d**
Page	**0**	**a**	a	a	a	a						
Frames	**1**	**b**	b	b	b	b						
	2	**c**	c	c	c	c						
	3	**d**	d	d	d	d						
Page faults							x					

The Optimal Page Replacement Algorithm

- Idea:
  - Select the page that will not be needed for the longest time <u>in the future</u>

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a	
	1	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	e	e	e	e	e	
Page faults							x					x

The Optimal Page Replacement Algorithm

- Idea:
  - Select the page that will not be needed for the longest time in the future

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	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	c
	3	d	d	d	d	d	e	e	e	e	e	d
Page faults							x					x

The Optimal Page Replacement Algorithm

- Problems?
  - Can't know the future of a program
  - Can't know when a given page will be needed next
  - The optimal algorithm is unrealizable

FIFO Page Replacement Algorithm

- Always replace the oldest page
- Example: Memory system with 4 frames

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page	0	a		a	a	a						
Frames	1	b				b						
	2	c	c	c	c	c						
	3	d			d	d						
Page faults							x					

FIFO Page Replacement Algorithm

- Always replace the oldest page
- Example: Memory system with 4 frames

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page	0	a		a	a	a	a	a	a	a		
Frames	1	b				b	b	b	b	b		
	2	c	c	c	c	c	e	e	e	e		
	3	d			d	d	d	d	d	d		
Page faults							x				x	

FIFO Page Replacement Algorithm

- Always replace the oldest page
- Example: Memory system with 4 frames

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		a	a	a	a	a	a	a	c	
	1	b				b	b	b	b	b	b	
	2	c	c	c	c	c	e	e	e	e	e	
	3	d			d	d	d	d	d	d	d	
Page faults							x				x	x

FIFO Page Replacement Algorithm

- Always replace the oldest page
- Example: Memory system with 4 frames

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		a	a	a	a	a	a	a	c	c
	1	b				b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	e
	3	d			d	d	d	d	d	d	d	a
Page faults							x				x	x

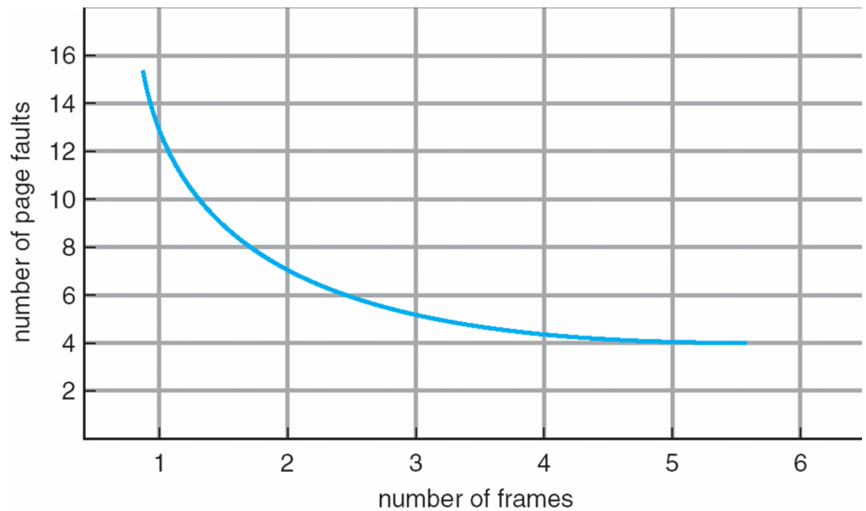
FIFO Page Replacement Algorithm

- Why might FIFO be good?
  - Maybe the page allocated very long ago isn't used anymore
- Why might FIFO not be so good?
  - Doesn't consider locality of reference!
  - The oldest page may be needed again soon
  - Some page may be important throughout execution

Belady's anomaly: Performance of an application might get worse as physical memory increases!!!

Belady's Anomaly

- Given a reference string, it would be natural to assume that
  - The more the total number of frames in main memory, the fewer the number of page faults



- Not true for some algorithms!
  - E.g., for FIFO

Belady's Anomaly

- Consider FIFO page replacement
  - Look at this reference string
    - 012301401234
  - Case 1:
    - 3 frames available
  - Case 2:
    - 4 frames available

Belady's Anomaly

FIFO with 3 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page													
Oldest Page													

FIFO with 4 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page													
Oldest Page													

Belady's Anomaly

FIFO with 3 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest Page				0	1	2	3	0	0	0	1	4	4
		P	**P**	**P**	**P**	**P**	**P**	**P**			**P**	**P**	

Belady's Anomaly

FIFO with 3 page frames

		0	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
Youngest Page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest Page				0	1	2	3	0	0	0	1	4	4
		P	**P**	**P**	**P**	**P**	**P**	**P**			**P**	**P**	

FIFO with 4 page frames

		0	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
Youngest Page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	2	2	3	4	0	1	2	3
Oldest Page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		P	**P**	**P**	**P**			**P**	**P**	**P**	**P**	**P**	**P**

Belady's Anomaly

FIFO with 3 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	9 page faults				1	4	2	2
Oldest Page				0	1	2	3	0	0	0	1	4	4
		P	**P**	**P**	**P**	**P**	**P**	**P**			**P**	**P**	

FIFO with 4 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	10 page faults				0	1	2	3
Oldest Page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		P	**P**	**P**	**P**			**P**	**P**	**P**	**P**	**P**	**P**

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently
  - Keep track of when pages are referenced to make a better decision
  - Use past behavior to predict future behavior
    - LRU uses past information
    - OPT uses future information
- Not optimal
- Does not suffer from Belady's anomaly

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page	0	a										
Frames	1	b										
	2	c										
	3	d										

Page faults

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a						
	1	b	b	b	b	b						
	2	c	c	c	c	c						
	3	d	d	d	d	d						
Page faults							x					

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a		
	1	b	b	b	b	b	b	b	b	b		
	2	c	c	c	c	c	e	e	e	e		
	3	d	d	d	d	d	d	d	d	d		
Page faults							x				x	

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a	
	1	b	b	b	b	b	b	b	b	b	b	
	2	c	c	c	c	c	e	e	e	e	e	
	3	d	d	d	d	d	d	d	d	d	c	
Page faults							x				x	x

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	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	e	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	d	c	c
Page faults							x				x	x

Least Recently Used Issues

- Implementation
  - Use time of last reference
    - Update every time page accessed (use system clock)
    - Page replacement - search for oldest time
  - Use a stack
    - On page access : remove from stack, push on top
    - Victim selection: select page at bottom of stack
- Problems or limitations?

Least Recently Used Issues

- Implementation
  - Use time of last reference
    - Update every time page accessed (use system clock)
    - Page replacement - search for smallest time
  - Use a stack
    - On page access : remove from stack, push on top
    - Victim selection: select page at bottom of stack
- Problems or limitations?
  - Both approaches require large processing overhead, more space, and hardware support
  - 32-bit timestamp would double size of PTE

Least Recently Used Issues

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

- Q1: What fraction of page accesses are faults?
  - None or almost none
  - About 1 in 4
  - About 2 in 4
  - About 3 in 4
  - All or almost all
- Q2: How well does OPT do?

Least Recently Used

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Requests		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2
Page Frames	0															
	1															
	2															

Page faults

Least Recently Used

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Requests		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2
Page Frames	0	0	0	0	3	3	3	2	2	2	1	1	1	0	0	0
	1		1	1	1	0	0	0	3	3	3	2	2	2	1	1
	2			2	2	2	1	1	1	0	0	0	3	3	3	2
Page faults		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Least Recently Used Issues

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

- Q1: What fraction of page accesses are faults?
  - None or almost none
  - About 1 in 4
  - About 2 in 4
  - About 3 in 4
  - All or almost all
- Q2: How well does OPT do?

OPT

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Requests		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2
Page Frames	0															
	1															
	2															

Page faults

OPT

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Time		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Requests		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2
Page Frames	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
	1		1	1	1	1	1	2	2	2	2	2	2	2	2	2
	2			2	3	3	3	3	3	3	3	3	3	0	0	0
Page faults		x	x	x	x			x			x			x		

LRU Approximation Algorithms

- Not used recently/Not recently used (NUR/ NRU)
- Accessed Bit in each page table entry
  - With each page, associate a bit, initially = 0
  - When page is accessed, bit is set to 1
  - Victim Selection
    - Any page with reference bit == 0, if one exists.
    - BUT: we do not know order of use

LRU Approximation Algorithms

- Additional Accessed Bits Algorithm
  - Use the PTE accessed bit and a small counter per page (2 or 3 bits in PTE)
  - Periodically (say every 100 msec), scan all physical pages. For each page:
    - If not accessed recently, (PTE accessed bit == 0), `counter++`
    - If accessed recently (PTE accessed bit == 1), `counter = 0`
    - Clear the PTE accessed bit in either case!

LRU Approximation Algorithms

- Additional Accessed Bits Algorithm
  - Counter will contain the number of scans since the last reference to this page
    - PTE that contains the highest counter value is the least recently used
    - So, evict the page with the highest counter

Approximate LRU

time

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Accessed pages in blue

0	1	1	1	0	0	1	1	0	1	0	1	1	0	0

Increment counter for untouched pages

0	1	1	1	0	0	1	1	0	1	0	1	1	0	0

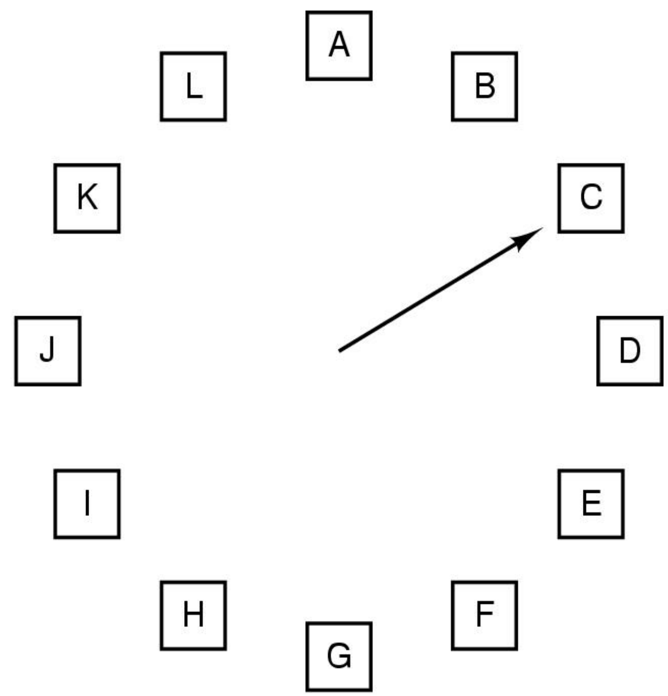
Accessed pages in blue

0	2	0	0	0	1	2	2	0	0	1	0	2	1	0

Highlighted pages have the highest counter value and can be evicted.

Clock Algorithm

- Treats page frames allocated to a process as a circular buffer
- Set accessed bit on access
- Pointer (clock) sweeps over page frames
  - Look for victim page with accessed bit unset
  - If bit is set, clear it and move on to next page
  - Replace pages that haven't been referenced for one complete clock revolution



Clock Algorithm

- "Clock pointer" scans over page frames
  - Clock pointer loops around when it gets to end of circular buffer
- If PTE accessed bit == 1, clear bit and advance pointer to give it a second-chance
- If PTE accessed bit == 0, evict this page
  - No need for a counter in the PTE!