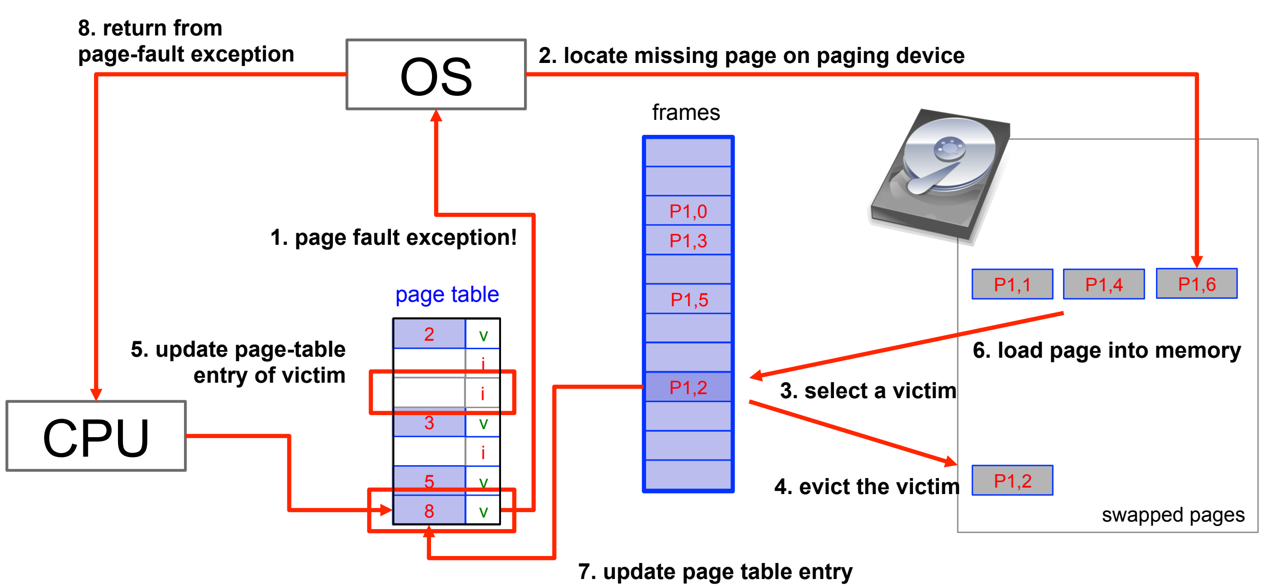
Virtual Memory: Policies (Part I)

- Recap: Page Fault Mechanics: Page faults are expensive!
- Memory Access Patterns: Locality of Reference
- Page Replacement Policies
  - FIFO, Optimal
  - Approximations to optimal: LRU
  - Approximations to LRU: 2nd Chance, Enh. 2nd Chance

- Coming Next: Working Sets, Page Caching, Case Studies

Recap: Steps of a Page Fault



Cost of Page Faults

Observation: Page faults are very expensive!

Effective Memory Access Time ema:

$$ema = (1-p) * ma + p * \text{“page fault time”}$$

where

- p = probability of a page fault
- ma = memory access time

Example:

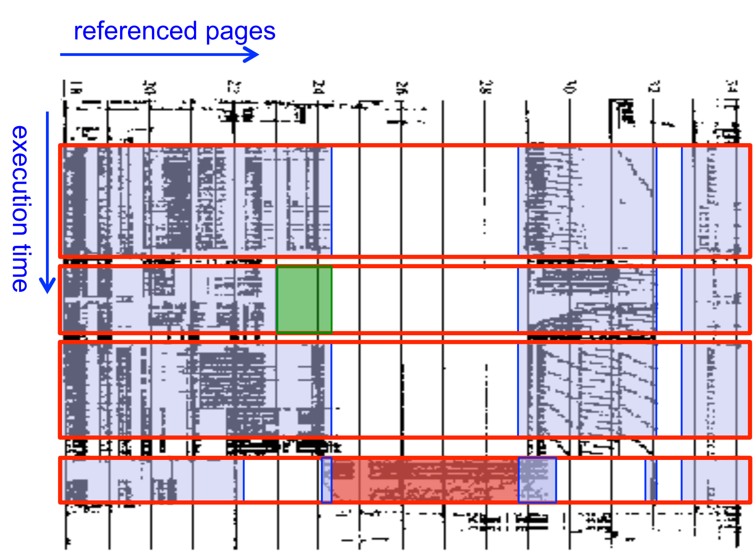
- ma = 50 ns
- pft = 10 msec
- p = 1% ==> ema = **0.1msec**
- p = 0.001% ==> ema = **150ns**

2000x !

3x!

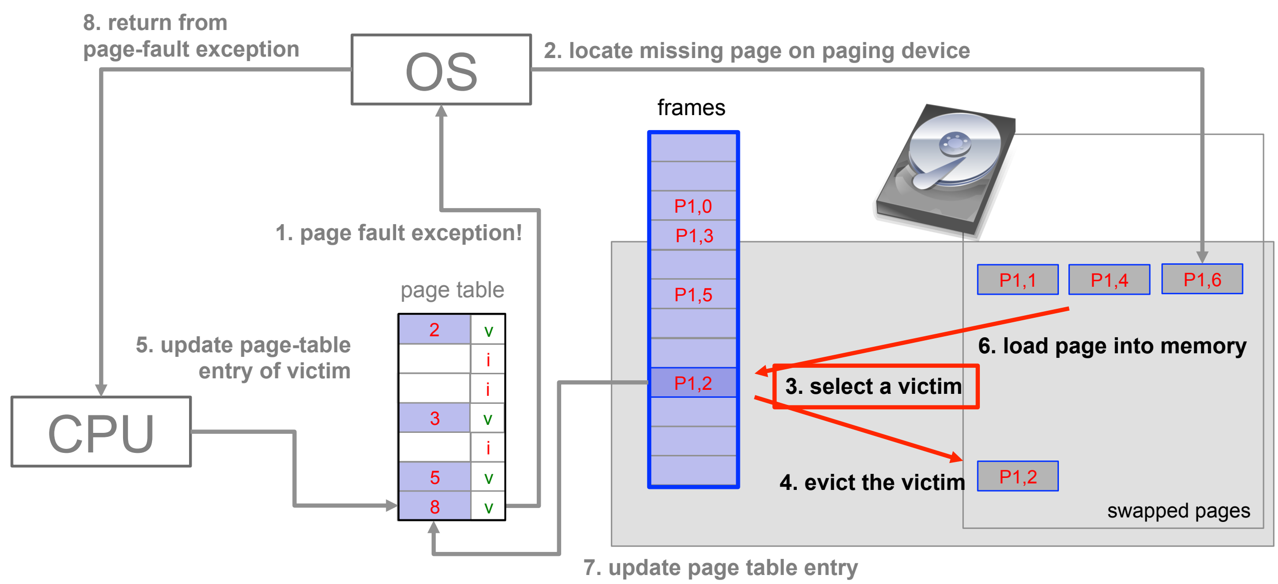
Objective: **Minimize** page fault rate.

Locality of Reference



Locality of Reference: A program that references a location n at some point in time is likely to reference the same location n and locations in the immediate vicinity of n in the near future.

Page Replacement



Page Replacement: Why bother?

Example Program:

```
for(int i=0; i<10; i++) {
   a = x * a;
}
```

Machine (pseudo-) Code:

```
FETCH i, R0
STORE R0++, i
FETCH a, R1
FETCH x, R2
MULT R1, R2
STORE R1, a
```

repeat 10 times!

Memory Layout

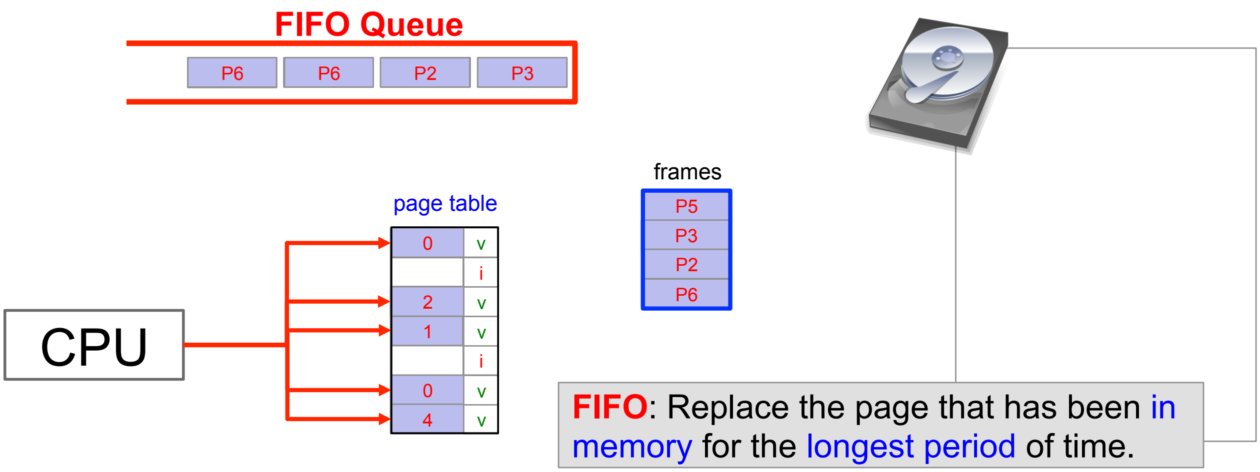
P0	i
P1	x
P2	
P3	a
P4	

Physical Memory

F0

6 I/O Operations

FIFO Page Replacement



FIFO Page Replacement

time						1	2	3	4	5	6	7	8	9	10
reference string		a	b	c	d	c	a	d	b	e	b	a	b	c	d
frames		a	a	a	a	a	a	a	a	e	e	e	e	e	d
			b	b	b	b	b	b	b	b	b	a	a	a	a
				c	c	c	c	c	c	c	c	c	b	b	b
					d	d	d	d	d	d	d	d	d	c	c

FIFO Page Replacement

time						1	2	3	4	5	6	7	8	9	10
reference string		a	b	c	d	c	a	d	b	e	b	a	b	c	d
frames		a	a	a	a	a	a	a	a	e	e	e	e	e	d
			b	b	b	b	b	b	b	b	b	a	a	a	a
				c	c	c	c	c	c	c	c	c	b	b	b
					d	d	d	d	d	d	d	d	d	c	c

Pros:
- Simplicity!

Cons:
- Assumes that pages residing the longest in memory are the least likely to be referenced in the future.
- Thus, does not exploit *principle of locality of reference*.

Optimal Replacement Algorithm

Algorithm with **provably** lowest page fault rate of all algorithms:

Replace that page which will not be used for the longest period of time (in the future).

time						1	2	3	4	5	6	7	8	9	10
future string		a	b	c	d	c	a	d	b	e	b	a	b	c	d
reference string		a	b	c	d	c	a	d	b	e	b	a	b	c	d
frames		a	a	a	a	a	a	a	a	a	a	a	a	a	
			b	b	b	b	b	b	b	b	b	b	b	b	
				c	c	c	c	c	c	c	c	c	c	c	
					d	d	d	d	d	e	e	e	e	e	

Optimal Replacement Algorithm

Algorithm with **provably** lowest page fault rate of all algorithms:

Replace that page which will not be used
for the longest period of time (in the future).

Approximate!

reference string	a	b	c	d	c	a	d	b	e	b	a	b	c	d
frames														

Pros:
- great performance!

Cons:
- needs clairvoyance!

Approximation to Optimal: LRU

Least Recently Used: replace the page that has not been accessed for longest period of time (in the past).

time		1	2	3	4	5	6	7	8	9	10
reference string		c	a	d	b	e	b	a	b	c	d
frames	a	a	a	a	a	a	a	a	a	a	a
	b	b	b	b	b	b	b	b	b	b	b
	c	c	c	c	c	e	e	e	e	e	d
	d	d	d	d	d	d	d	d	d	c	c
						!				!	!

Approximation to Optimal: LRU

Least Recently Used: replace the page that has not been accessed for longest period of time (in the past).

reference string	a	b	c	d	c	a	d	b	e	b	a	b	c	d

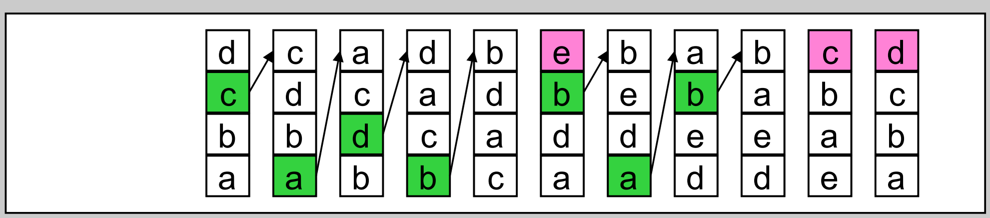
Pros:
- good performance!

Cons:
- difficult to implement!

LRU: Implementation

Problem: We need to keep chronological history of page references; need to be reordered upon each reference.

Stack:



Capacitors: Associate a capacitor with each memory frame. Capacitor is charged with every reference to the frame. The subsequent exponential decay of the charge can be directly converted into a time interval.

Aging registers: Associate aging register of n bits $(R_{n-1}, \ldots, R_0)$ with each frame in memory. Set R_{n-1} to 1 for each reference. Periodically shift registers to the right.

Approximation to LRU: 2nd Chance Algorithm

2nd Chance Algorithm: Associate a *use_bit* with every frame in memory.

– Upon each reference, set *use_bit* to 1.
– Keep a pointer to next "victim candidate" page.
– To select victim: If current frame's *use_bit* is 0, select frame and increment pointer. Otherwise delete *use_bit* and increment pointer.

time		1	2	3	4	5	6	7	8	9	10
reference string		c	a	d	b	e	b	a	b	c	d
frames	a/1	a/1	a/1	a/1	a/1	e/1	e/1	e/1	e/1	e/1	d/1
	b/1	b/1	b/1	b/1	b/1	b/0	b/1	b/0	b/1	b/1	b/0
	c/1	c/1	c/1	c/1	c/1	c/0	c/0	a/1	a/1	a/1	a/0
	d/1	d/1	d/1	d/1	d/1	d/0	d/0	d/0	d/0	c/1	c/0
						!		!		!	!

Improvement on 2nd Chance Algorithm

- Consider read/write activity of page: *dirty_bit*
- Algorithm same as 2nd Chance Algorithm, except that we scan for pages with **both** *use_bit* and *dirty_bit* equal to 0.
- Each time the pointer advances, the *use_bit* and *dirty_bit* are updated as follows:

	ud	ud	ud	ud
before	11	10	01	00
after	01	00	00*	(select)

- A dirty page is not selected until two full scans of the list later.
- Note: Other authors (e.g., Stallings) describe a slightly different algorithm!

Improved 2nd Chance Algorithm

	ud	ud	ud	ud
before	11	10	01	00
after	01	00	00*	(select)

time	1	2	3	4	5	6	7	8	9	10	
reference string	c	a^w	d	b^w	e	b	a^w	b	c	d	
frames	a/10	a/10	a/11	a/11	a/11	a/00*	a/00*	a/11	a/11	a/11	
	b/10	b/10	b/10	b/10	b/11	b/00*	b/10*	b/10*	b/10*	b/10*	
	c/10	c/10	c/10	c/10	c/10	e/10	e/10	e/10	e/10	e/10	
	d/10	d/10	d/10	d/10	d/10	d/00	d/00	d/00	d/00	c/10	
						!				!	!

Virtual Memory: Policies (Part I)

- Recap: Page Fault Mechanics: Page faults are expensive!
- Memory Access Patterns: Locality of Reference
- Page Replacement Policies
  - FIFO, Optimal
  - Approximations to optimal: LRU
  - Approximations to LRU: 2nd Chance, Enh. 2nd Chance
- Coming Next: Working Sets, Page Caching, Case Studies