CS 423
Operating System Design: Virtual Memory Mgmt

Professor Adam Bates
Spring 2018

Goals for Today

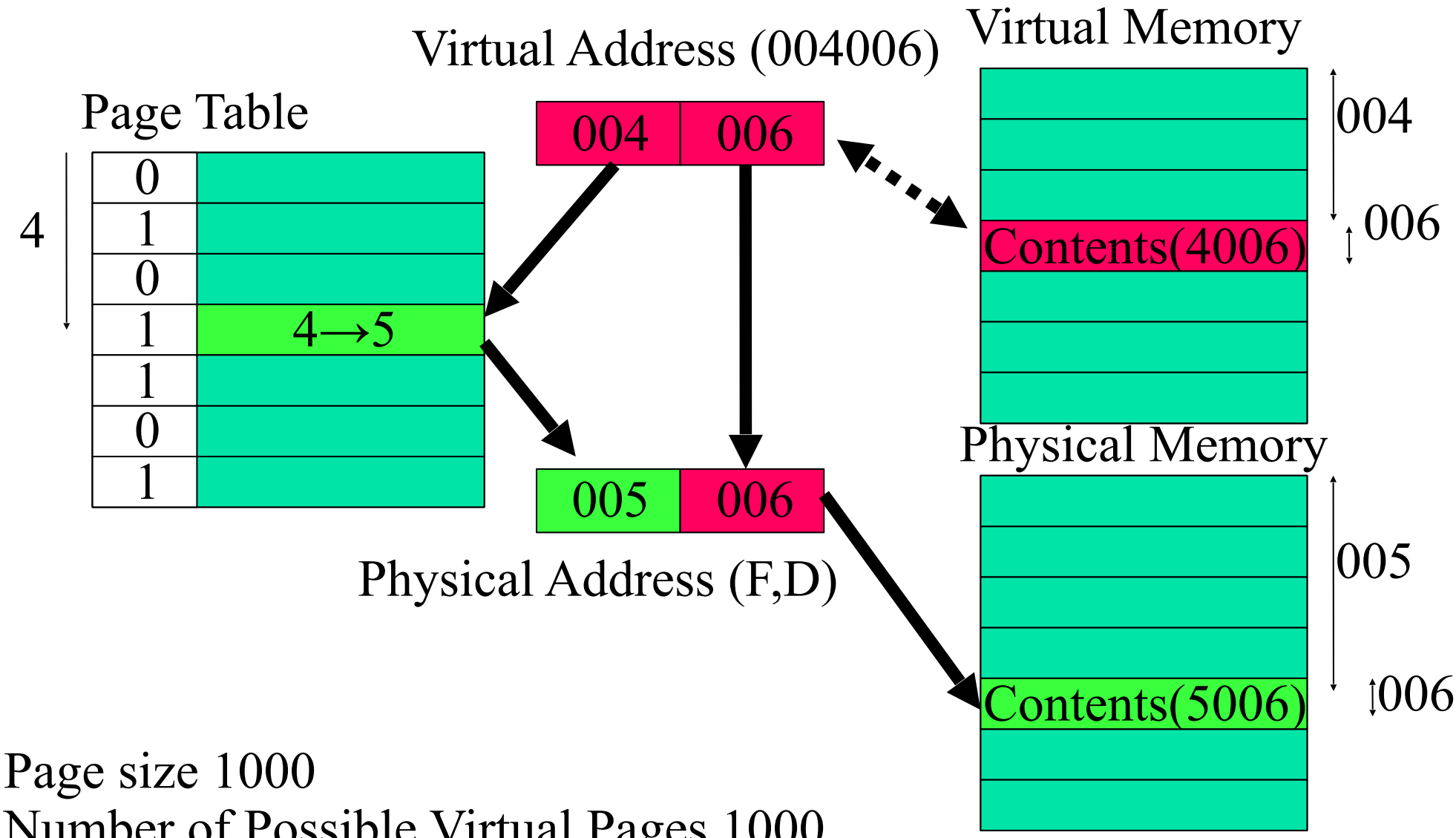
- Learning Objective:
  - Understand properties of virtual memory systems
- Announcements, etc:
  - Next C4 Summaries due March 2nd
  - Midterm March 7th!! Details on next slide
  - MP2 due March 16th
  - Strike Update: Classes continue as scheduled, assignment distribution/grading will likely be affected.

Reminder: Please put away devices at the start of class

Midterm Details

- In-Class on March 7th.
  - i.e., *50 minutes*
- Multiple choice
- 20-30 Questions
- **Openbook**: Textbooks, paper notes, printed sheets allowed. *No electronic devices permitted (or necessary)!*
- **Content**: All lecture and text material covered prior to March 5th (i.e., up to and including memory).
- We will have a review session, Q&A on March 5th.

Page Mapping Hardware



Page size 1000
Number of Possible Virtual Pages 1000
Number of Page Frames 8

Reasoning about Page Tables

- On a 32 bit system we have 2^32 B virtual address space
  - i.e., a 32 bit register can store 2^32 values
- # of pages are 2^n (e.g., 512 B, 1 KB, 2 KB, 4 KB…)
- Given a page size, how many pages are needed?
  - e.g., If 4 KB pages (2^12 B), then 2^32/2^12=…
    - 2^20 pages required to represent the address space
- **But**! each page entry takes more than 1 Byte of space to represent.
  - suppose page size is 4 bytes (Why?)
  - (2*2) * 2^ 20 = 4 MB of space required to represent our page table in physical memory.

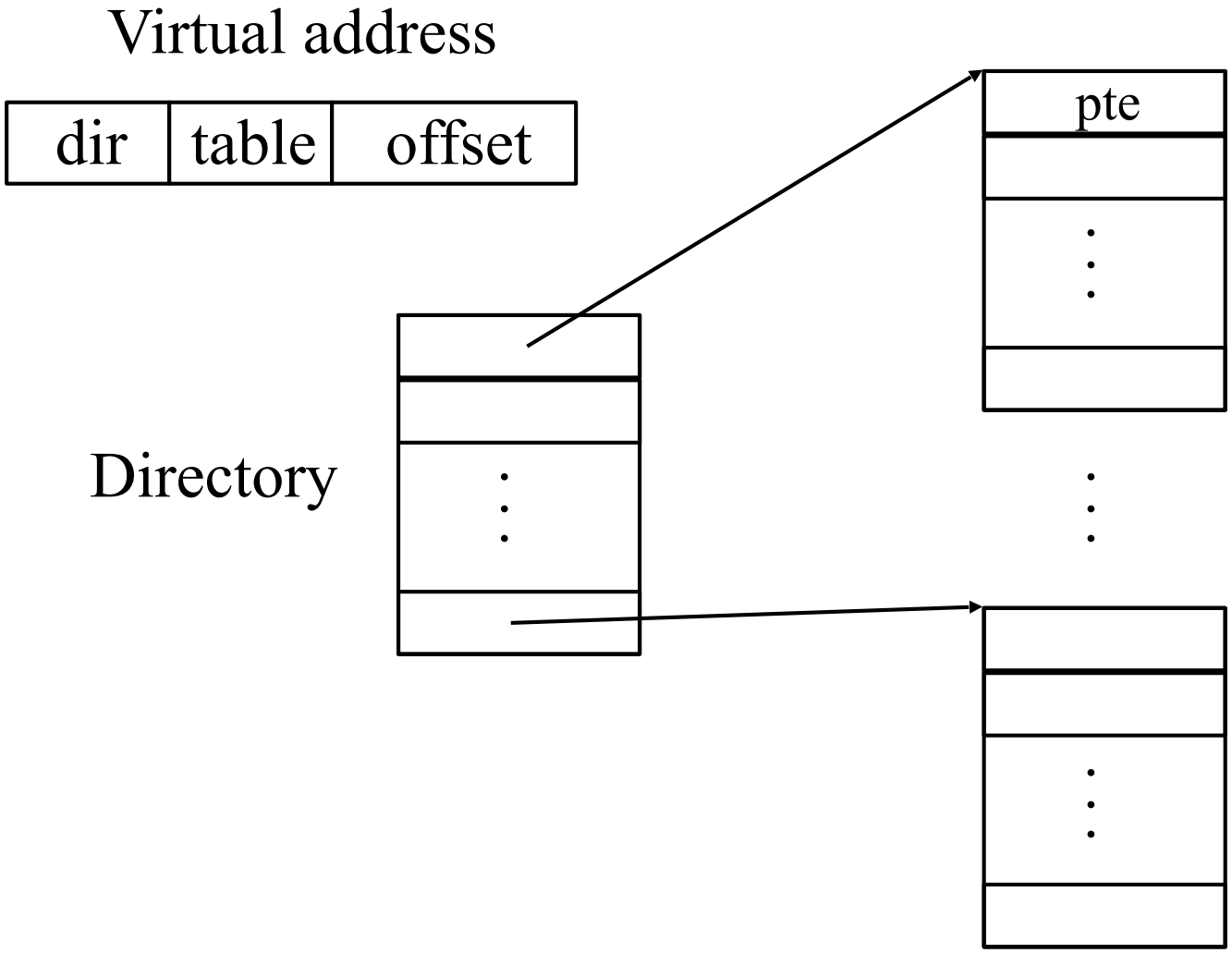
Question

Applications might make sparse use of their virtual address space. How can we make our page tables more efficient?

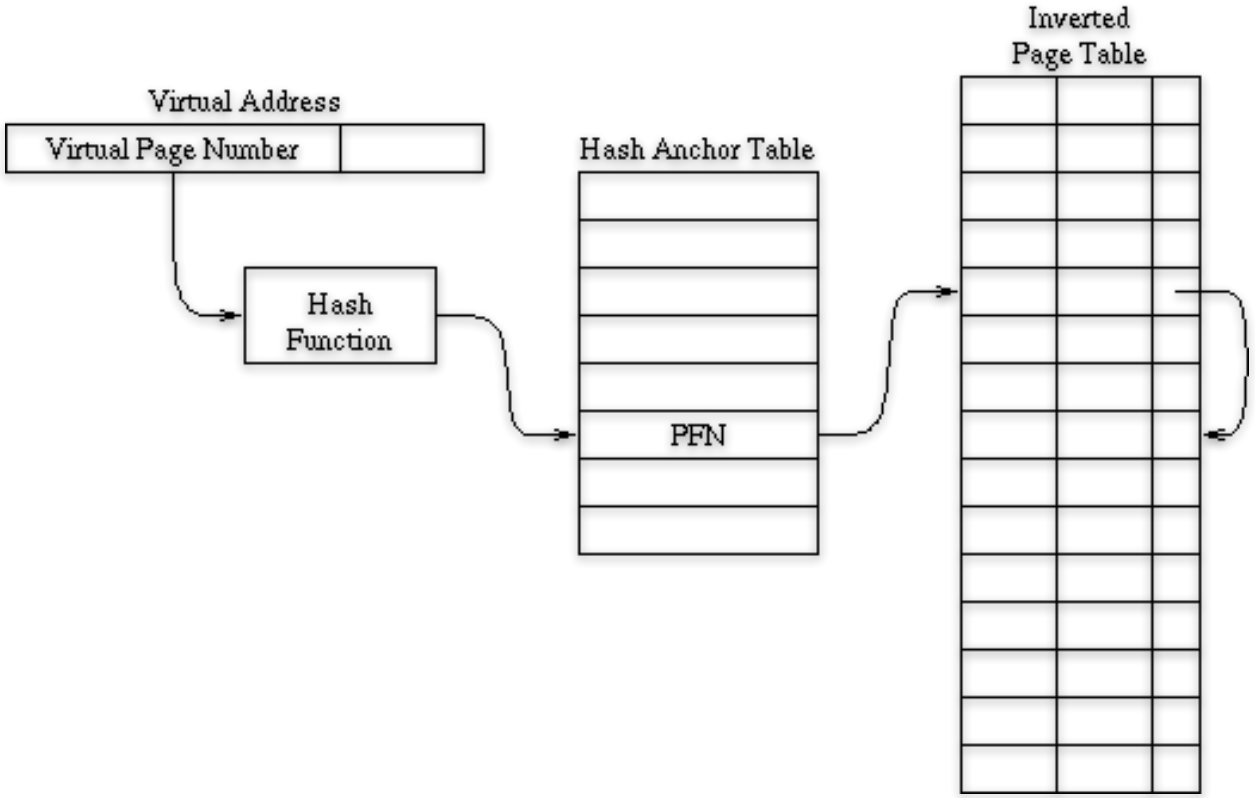
Multi-level Page Tables

What does this buy us?

Answer: Sparse address spaces, and easier paging



Inverted Page Tables



- Hash the process ID and virtual page number to get an index into the HAT.
- Look up a Physical Frame Number in the HAT.
- Look at the inverted page table entry, to see if it is the right process ID and virtual page number. If it is, you're done.
- If the PID or VPN does not match, follow the pointer to the next link in the hash chain. Again, if you get a match then you're done; if you don't, then you continue. Eventually, you will either get a match or you will find a pointer that is marked invalid. If you get a match, then you've got the translation; if you get the invalid pointer, then you have a miss.

Page Faults

- Access a virtual page that is not mapped into any physical page
  - A fault is triggered by hardware
- Page fault handler (in OS's VM subsystem)
  - Find if there is any free physical page available
    - If no, evict some resident page to disk (swapping space)
  - Allocate a free physical page
  - Load the faulted virtual page to the prepared physical page
  - Modify the page table

Paging Terminology

- **Reference string**: the memory reference sequence generated by a program.
- **Paging** – moving pages to (from) disk
- **Optimal** – the best (theoretical) strategy
- **Eviction** – throwing something out
- **Pollution** – bringing in useless pages/lines

Page Replacement

1. Find location of page on disk
2. Find a free page frame
    1. If free page frame use it
    2. Otherwise, select a page frame using the page replacement algorithm
    3. Write the selected page to the disk and update any necessary tables
3. Read the requested page from the disk.
4. Restart instruction.

Issue: Eviction

- Hopefully, kick out a less-useful page
  - Dirty pages require writing, clean pages don't
    - Hardware has a dirty bit for each page frame indicating this page has been updated or not
  - Where do you write? To "swap space" on disk.
- Goal: kick out the page that's least useful
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren't likely to be used again

Page Replacement Strategies

- **The Principle of Optimality**
  - Replace the page that will not be used the most 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
- **Second Chance**
  - An approximation to LRU.

Principle of Optimality

- Description:
  - Assume that each page can be labeled with the number of instructions that will be executed before that page is first referenced, i.e., we would know the future reference string for a program.
  - Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.
- Impractical because it needs to know future references

Optimal Example

12 references,
7 faults

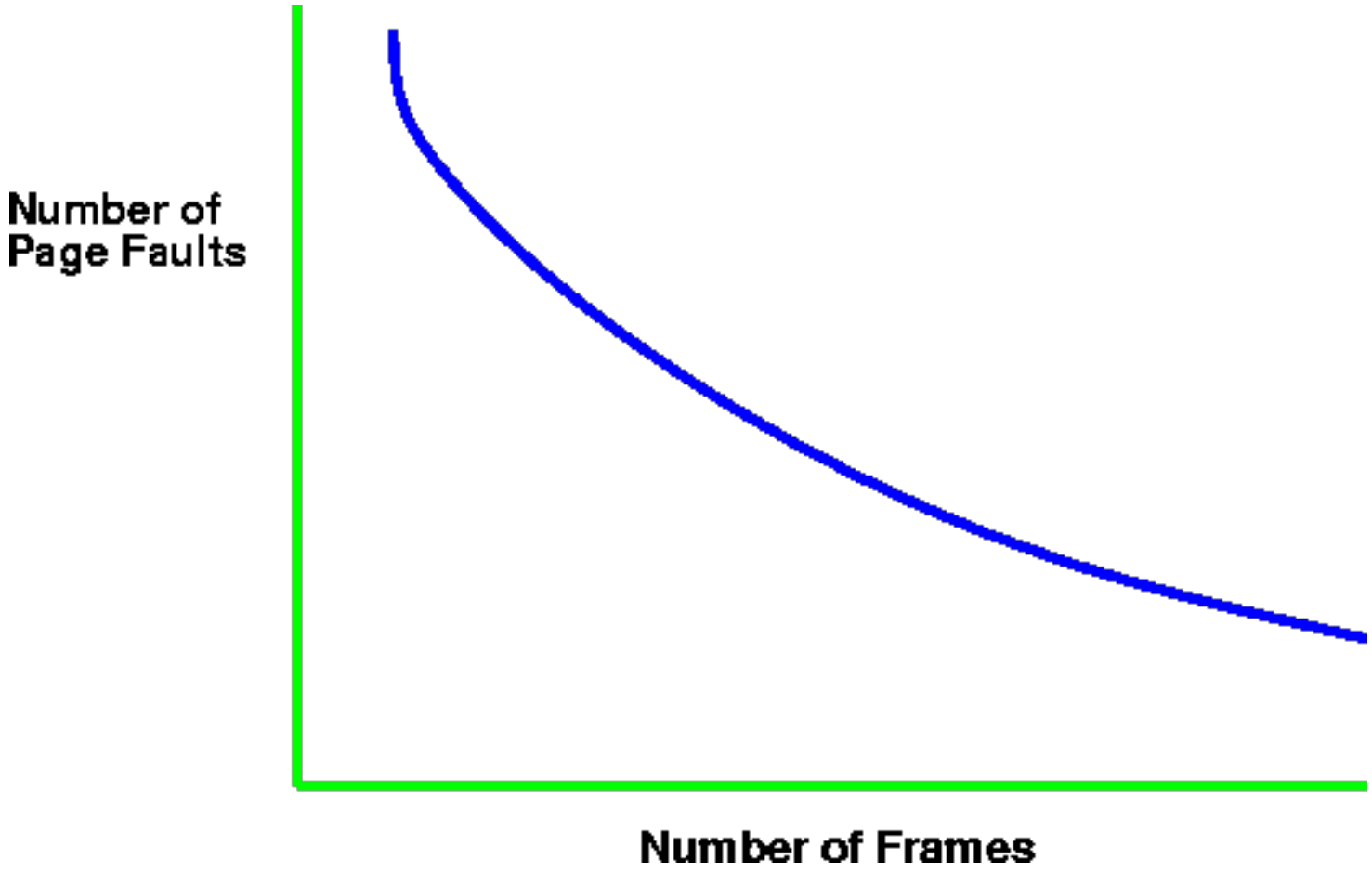
Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

FIFO

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Average Paging Behavior



As number of page frames increases, we can expect the number of page faults to decrease.

Belady's Anomaly (FIFO)

FIFO with 4 physical pages

12 references, 10 faults

As the number of page frames increase, so does the fault rate.

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

LRU

12 references,
10 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

Least Recently Used (LRU) Issues

- How to track "recency"?
  - use time
    - record time of reference with page table entry
    - use counter as clock
    - search for smallest time.
  - use stack
    - remove reference of page from stack (linked list)
    - push it on top of stack
- both approaches require large processing overhead, more space, and hardware support.

Second Chance

- Only one reference bit in the page table entry.
  - 0 initially
  - 1 When a page is referenced
- pages are kept in FIFO order using a circular list.
- Choose "victim" to evict
  - Select head of FIFO
  - If page has reference bit set, reset bit and select next page in FIFO list.
  - keep processing until you reach page with zero reference bit and page that one out.
- System V uses a variant of second chance

Second Chance Example

12 references
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A*		
B	yes	B*	A*	
C	yes	C*	B*	A*
D	yes	D*	C	B
A	yes	A*	D*	C
B	yes	B*	A*	D*
E	yes	E*	B	A
A	no	E*	B	A*
B	no	E*	B*	A*
C	yes	C*	E	B
D	yes	D*	C*	E
E	no	D*	C*	E*