Memory Replacement Policies

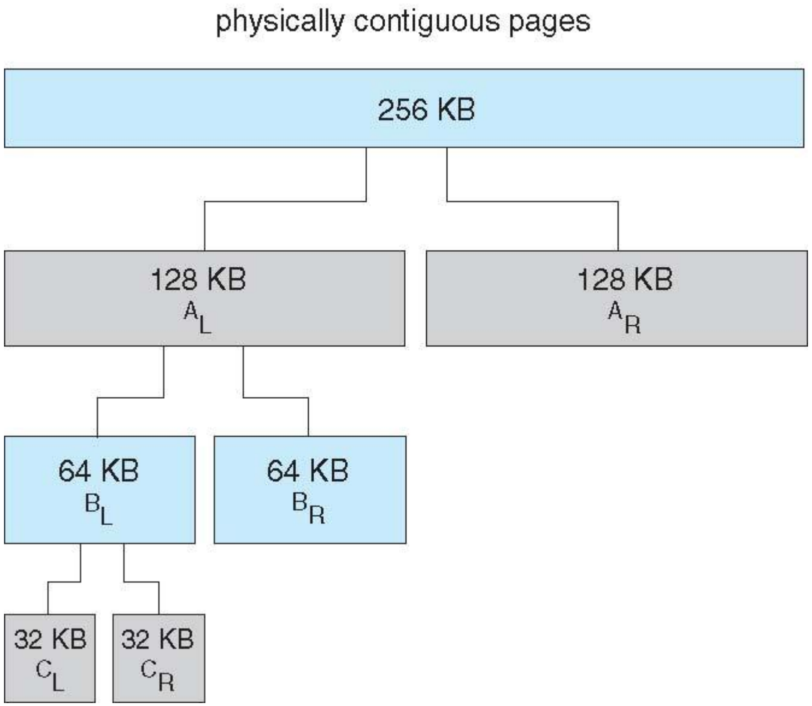
Storage Placement Strategies

- Best fit
  - Produces the smallest leftover hole
  - Creates small holes that cannot be used
- Worst Fit
  - Produces the largest leftover hole
  - Difficult to run large programs
- First Fit
  - Creates average size holes

- Buddy System
  - Used in Linux

Buddy System

- Memory allocated using power-of-2 allocator
  - Satisfy requests in units of size power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available



Buddy System

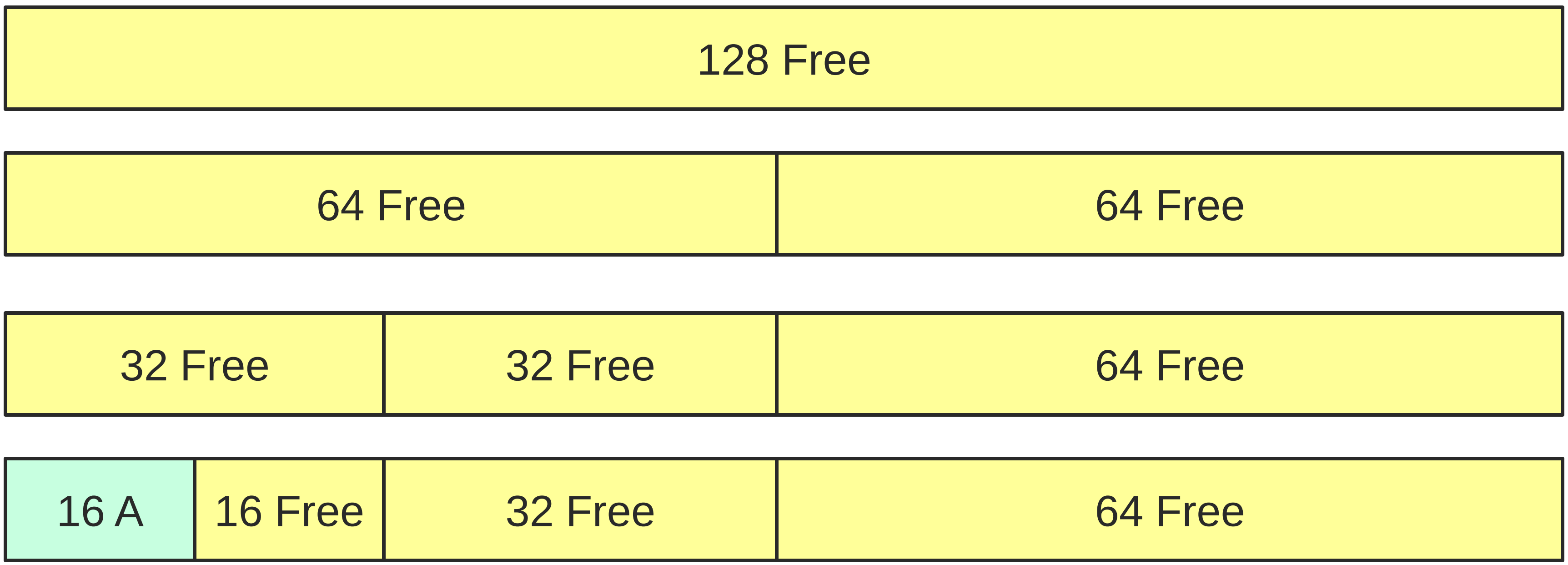
- Approach
  - Minimum allocation size = smallest frame
  - Use a bitmap to monitor frame use
  - Maintain freelist for each possible frame size
    - power of 2 frame sizes from min to max
  - Initially one block = entire buffer
  - If two neighboring frames (“buddies”) are free, combine them and add to next larger freelist

Buddy System Example

128 Free

Buddy System Example

Process A requests 16



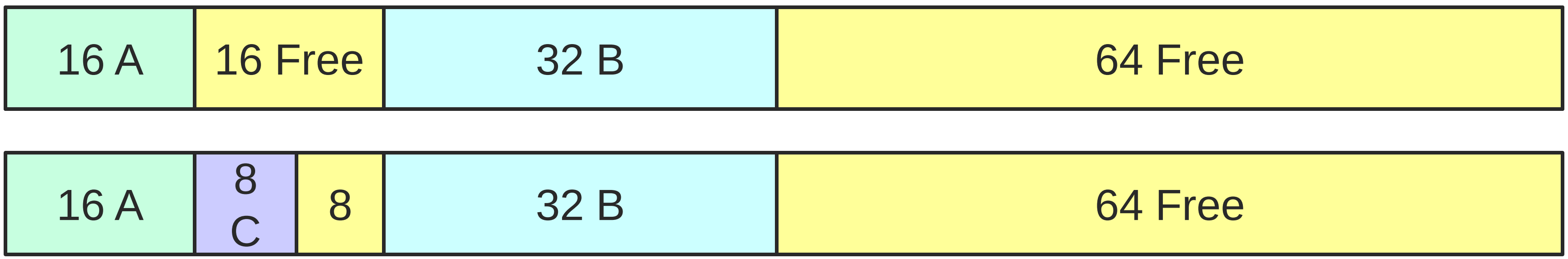
Buddy System Example

Process B requests 32

16 A	16 Free	32 B	64 Free

Buddy System Example

Process C requests 8



Buddy System Example

Process A exits

16 Free	8 C	8	32 B	64 Free

Buddy System Example

Process C exits

16 Free	8	8	32 B	64 Free

16 Free	16 Free	32 B	64 Free

32 Free	32 B	64 Free

- Advantage
  - Minimizes external fragmentation
- Disadvantage
  - Internal fragmentation when not 2^n request

Virtual Memory Recap

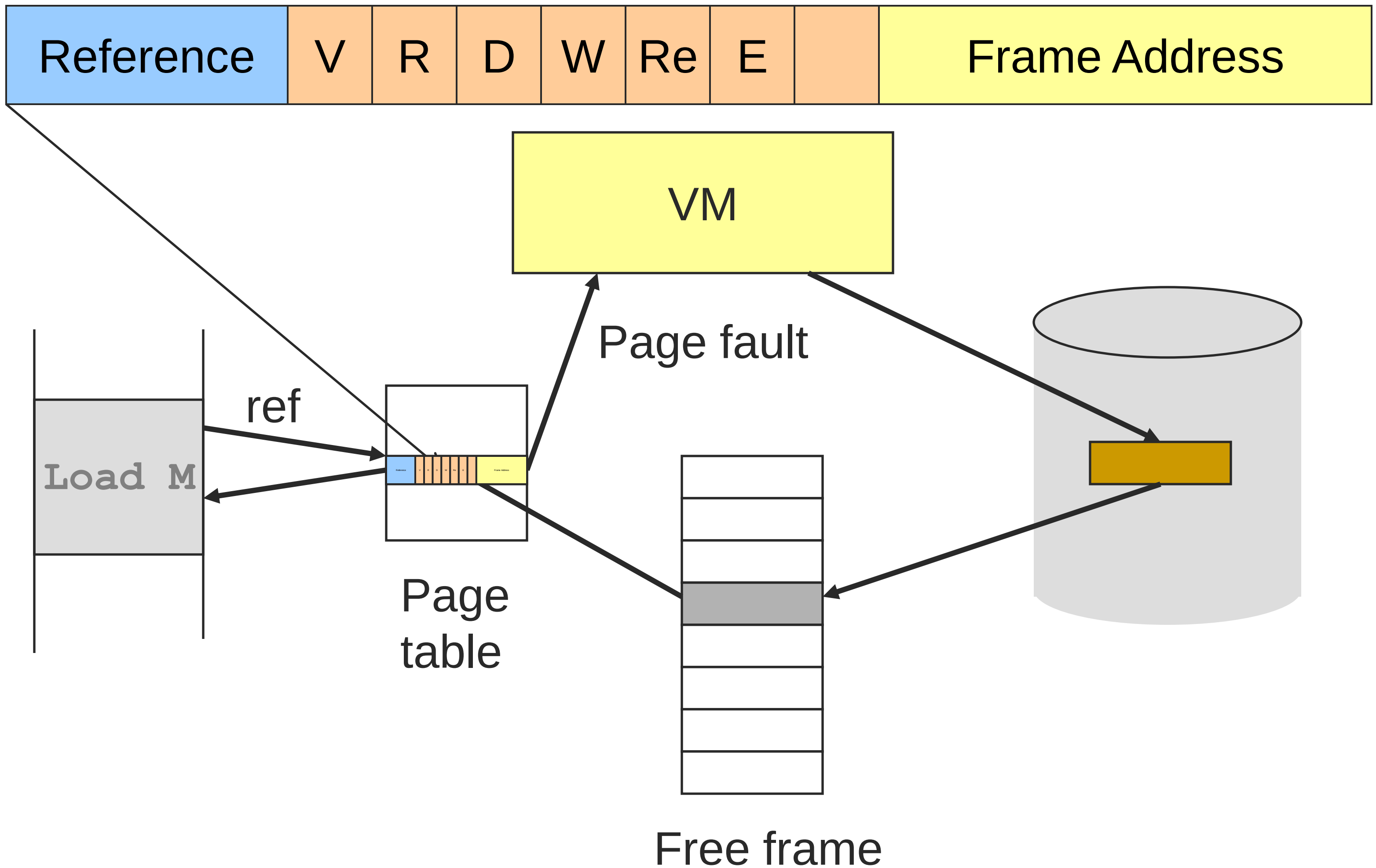
- Main memory
  - Organized into fixed sized frames
- Virtual address space
  - Per process
  - Split into fixed-sized pages
- Page
  - Size of frame = size of page
  - May be brought from disk into a frame in main memory
- Page fault
  - Process accesses a page that is not in main memory
  - Trap/interrupt occurs to OS and VM system is invoked

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

- Page is needed when
  - Process references it
  - invalid reference -> abort
  - not-in-memory -> bring to memory

Demand Paging Example



Demand Paging Policies

- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
    - Demand Paging = only when demanded by a process (default)
    - Pre-paging = before request by process
- Placement Strategies
  - When a page is brought into primary storage, which frame should it be placed in?
- Replacement Strategies
  - Which page now in primary storage should be removed from primary storage when some other page needs to be brought in and there is no free frame

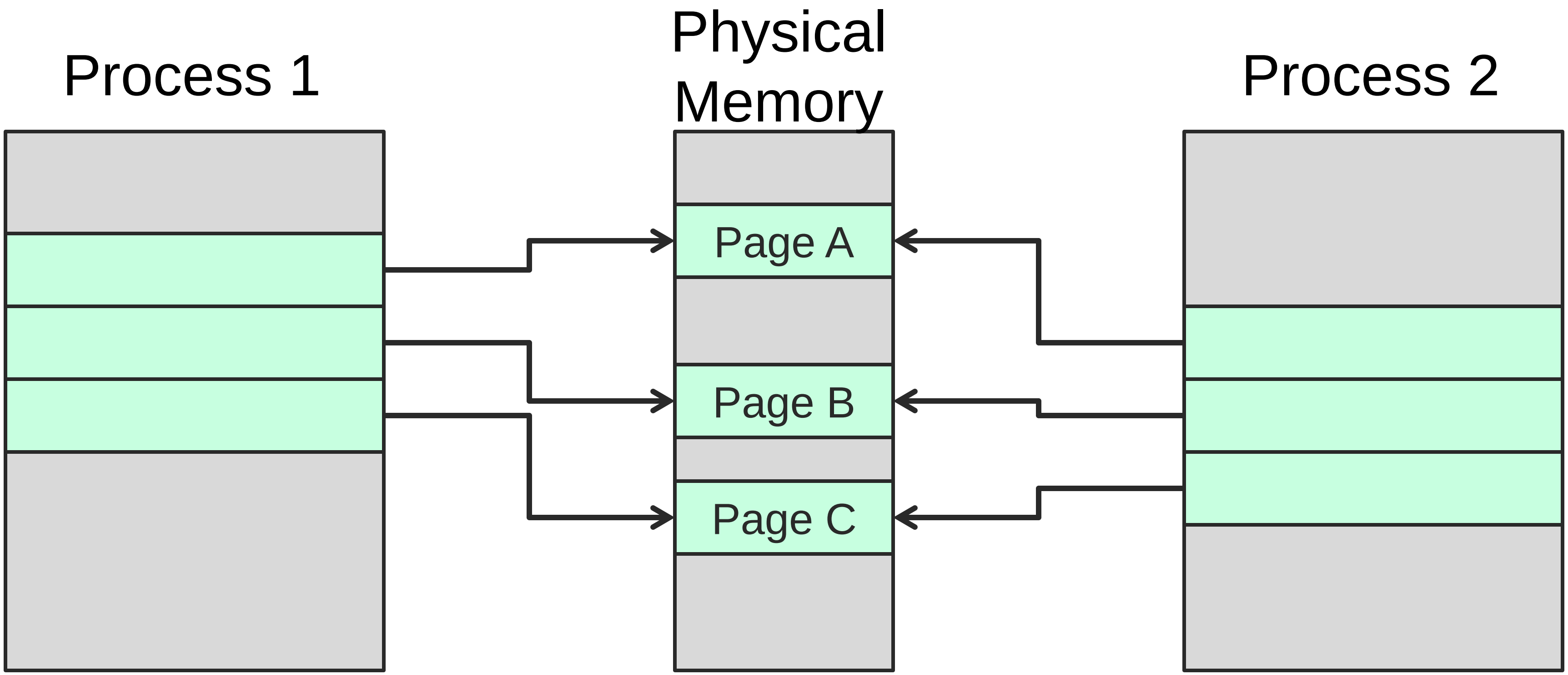
Page Fault Handler

- Find a free frame
  - If a free frame exists, use it
  - Otherwise, select a victim frame using a page replacement algorithm
  - Write the page in the victim frame to disk and update any necessary page tables
- Find location of page on disk
- Read the requested page from the disk to the selected frame
- Return from page fault handler to process

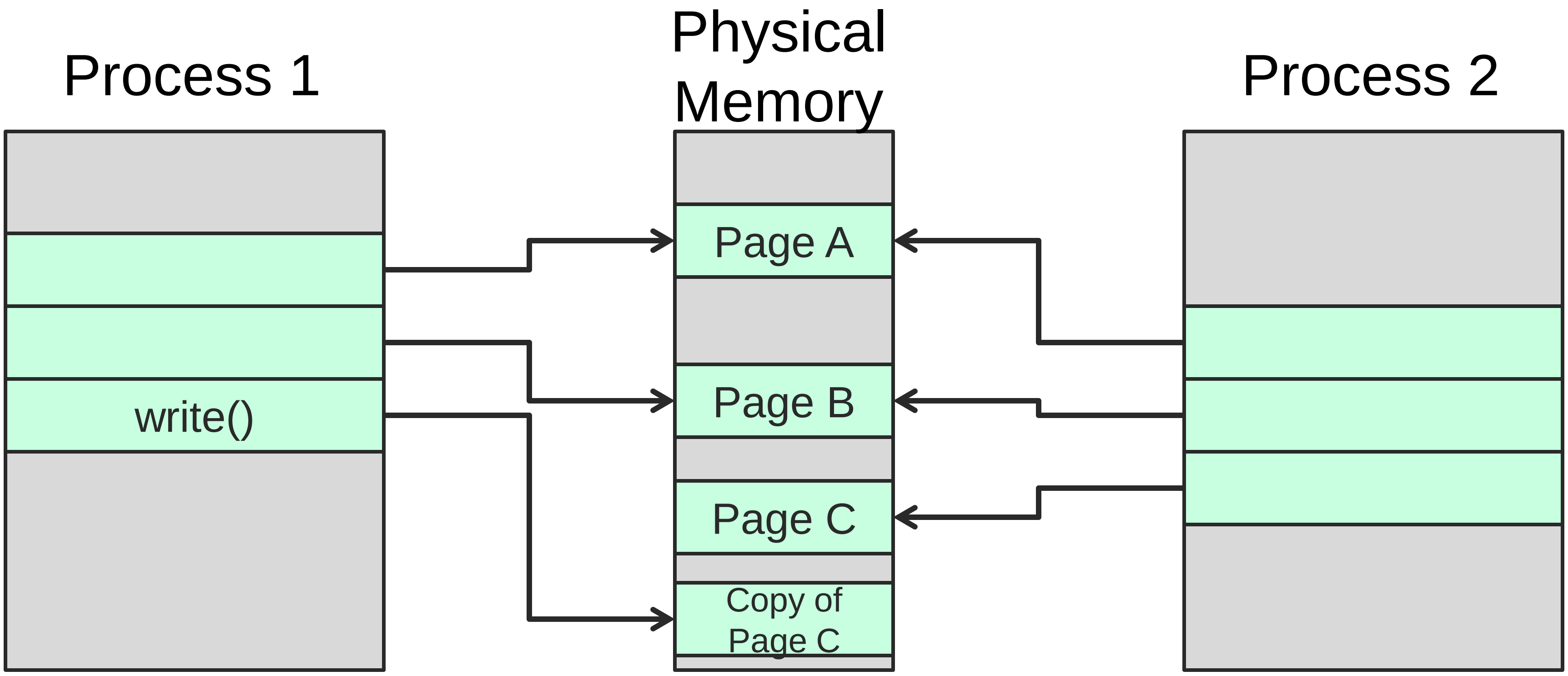
Copy-on-Write

- Copy-on-Write (COW)
  - Allows parent and child processes to initially share the same pages in memory
  - If either process modifies a shared page, only then is the page copied
  - More efficient process creation since only modified pages are copied

Before: Copy on Write



After: Copy on Write



Page Replacement Issues

- No free frames
  - Disk read (and possibly read) of pages is required
- Page to be replaced has not been changed since it was read in
  - Page can be overwritten and does not need to be copied out to disk
- Be careful before evicting!
  - Read-only pages are never written but may be shared!

Page Replacement Issues

- Dirty bit in each page table entry
  - When page brought in from disk, bit reset
  - First write to page => bit set
  - Bit checked when this frame selected as victim (if set, save to page before overwriting)
- Reference string
  - Set of page numbers generated by a process (via its page fault handler) over time, e.g., 1, 3, 4, 3, 2, …
- Goal
  - Come up with page replace algorithms that minimize number of page faults (why?)

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
  - Realistic?

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						
	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 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	
	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 <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	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

- Idea:
  - Select the page that will not be needed for the longest time <u>in the future</u>
- Problem:
  - Can’t know the future of a program
  - Can’t know when a given page will be needed next
  - The optimal algorithm is unrealizable

Principal of Optimality

- If the reference string can be predicted accurately
  - Don't use demand paging; use pre-paging
  - Allows paging activity of pages needed in the future to be overlapped with computation
- Optimal provides a basis for comparison with other schemes
  - Is difficult to implement but compilers may help by providing hints (for a later course)
  - For now: try to approximate the optimal strategy with other page replacement algorithms

Page Replacement Algorithms

- FIFO - first in first out
  - Evict the page that has been in primary memory the longest
- Random
  - Choose a victim page randomly
- LRU - least recently used
  - Evict the page not used for the longest time in the past
  - Intended as an approximation to the optimal

Page Replacement Algorithms

- LFU - least frequently used
  - Evict the page that is used least often
- NUR/NRU - not used recently/not recently used
  - An approximation to LRU
- Working set
  - Keep in memory those pages that the process is actively using

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

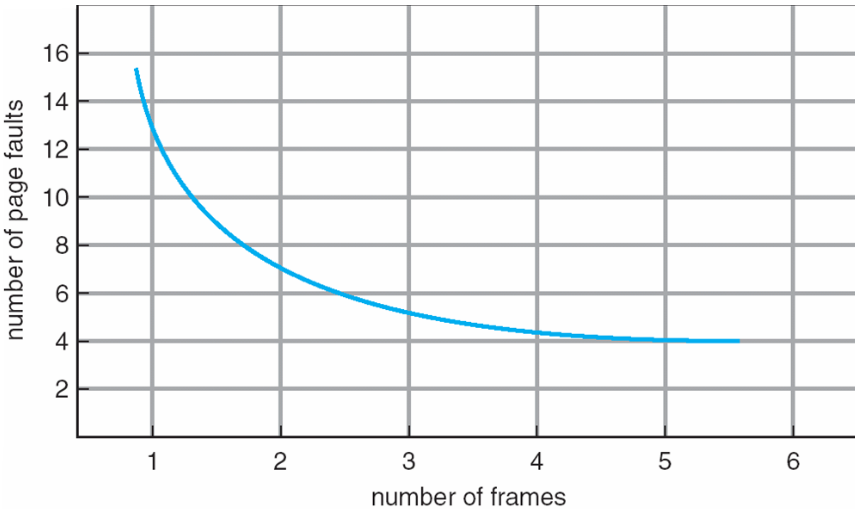
Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		a	a	a	a	a	a	a	c	c
	1	b				b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	e
	3	d			d	d	d	d	d	d	d	a
Page faults							x				x	x

FIFO Page Replacement Algorithm

- Always replace the oldest page
- Disadvantage
  - The oldest page may be needed again soon
  - Some page may be important throughout execution
    - Example?
    - It will get old, but replacing it will cause an immediate page fault

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		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

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										
	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	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					

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

- Not optimal
- Does not suffer from Belady's anomaly
- 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
- Both approaches require large processing overhead, more space, and hardware support.

LRU Approximation Algorithms

- Not used recently/Not recently used (NUR/NRU)
- Reference Bit in each page table entry
  - With each page, associate a bit, initially = 0
  - When page is referenced, bit is set to 1
  - Victim Selection:
    - Any page with reference bit == 0 (if one exists, otherwise FIFO)
    - BUT: Do not know order

LRU Approximation Algorithms

- Additional Reference Bits Algorithm
  - Keep n bits for each page in a table in memory
  - Each reference sets highest order bit
  - Periodically, shift bits right dropping the lowest bit)
  - Use value as 8 bit unsigned integer
  - Victim Selection:
    - Page with the lowest value of reference counter
    - Value may not be unique, use FIFO to resolve conflicts

Second Chance Page Replacement

- Modification to FIFO
- Pages kept in a linked list
  - Oldest is at the front of the list
- Look at the oldest page
  - If referenced bit == 0
    - Select for replacement
  - Else
    - Page was recently used -> don't replace it
    - Clear referenced bit
    - Move to the end of list
- What if every page was used in last clock tick?
  - Select a page at random

Clock Algorithm (Same effect as Second Chance)

- Maintain a circular list of pages in memory
- Set reference bit on access
- Clock sweeps over memory
  - Look for victim page with referenced 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