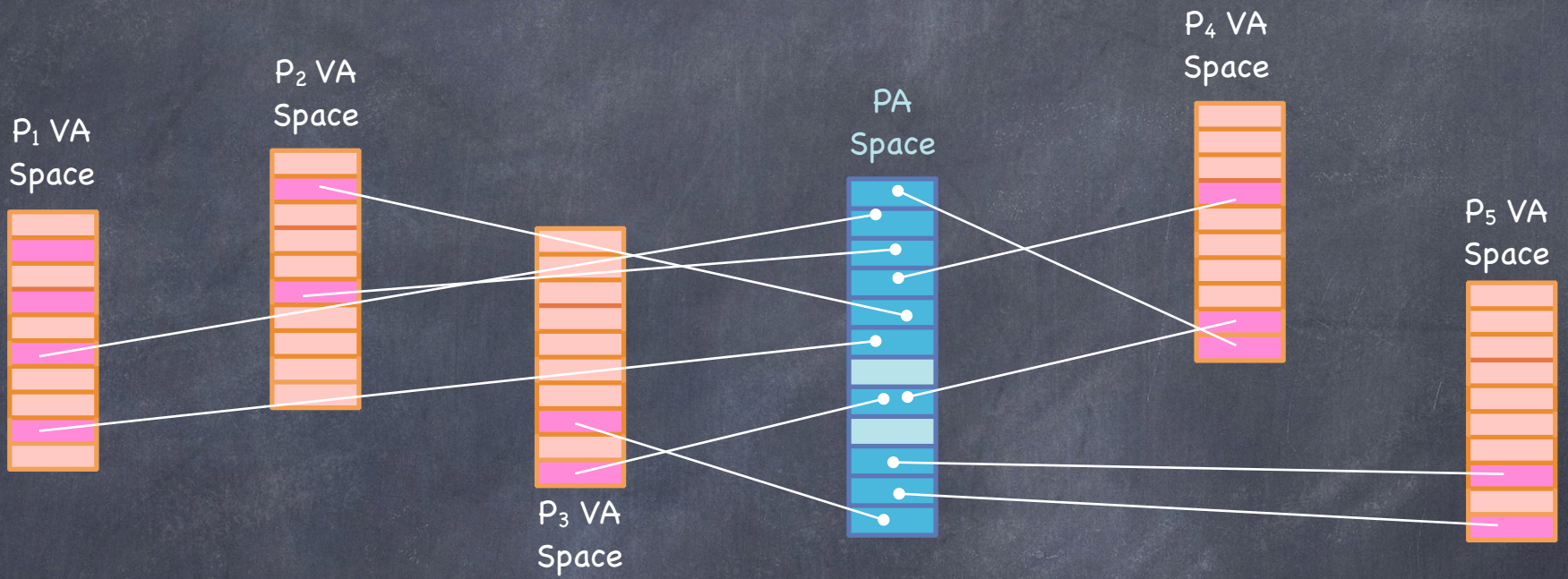
Translation and Locality

- A physically indexed cache ensures that nearby frames are not mapped to the same cache line...
- ...but adding a layer of indirection (through address translation) disrupts the spatial locality of physical caching
- OS may map adjacent virtual pages to physically far away frames, sharing the same entry in physical addressed cache
  - cache appears smaller
  - performance is unpredictable
- Solution: page coloring
  - frames colored according to the bucket where they will be cached
  - When mapping pages to frames, OS spreads each process' pages across frames with as many colors as possible

A different approach



- So many virtual pages…
- …comparatively few physical frames
- What if mapping size were proportional to the number of frames, not pages?

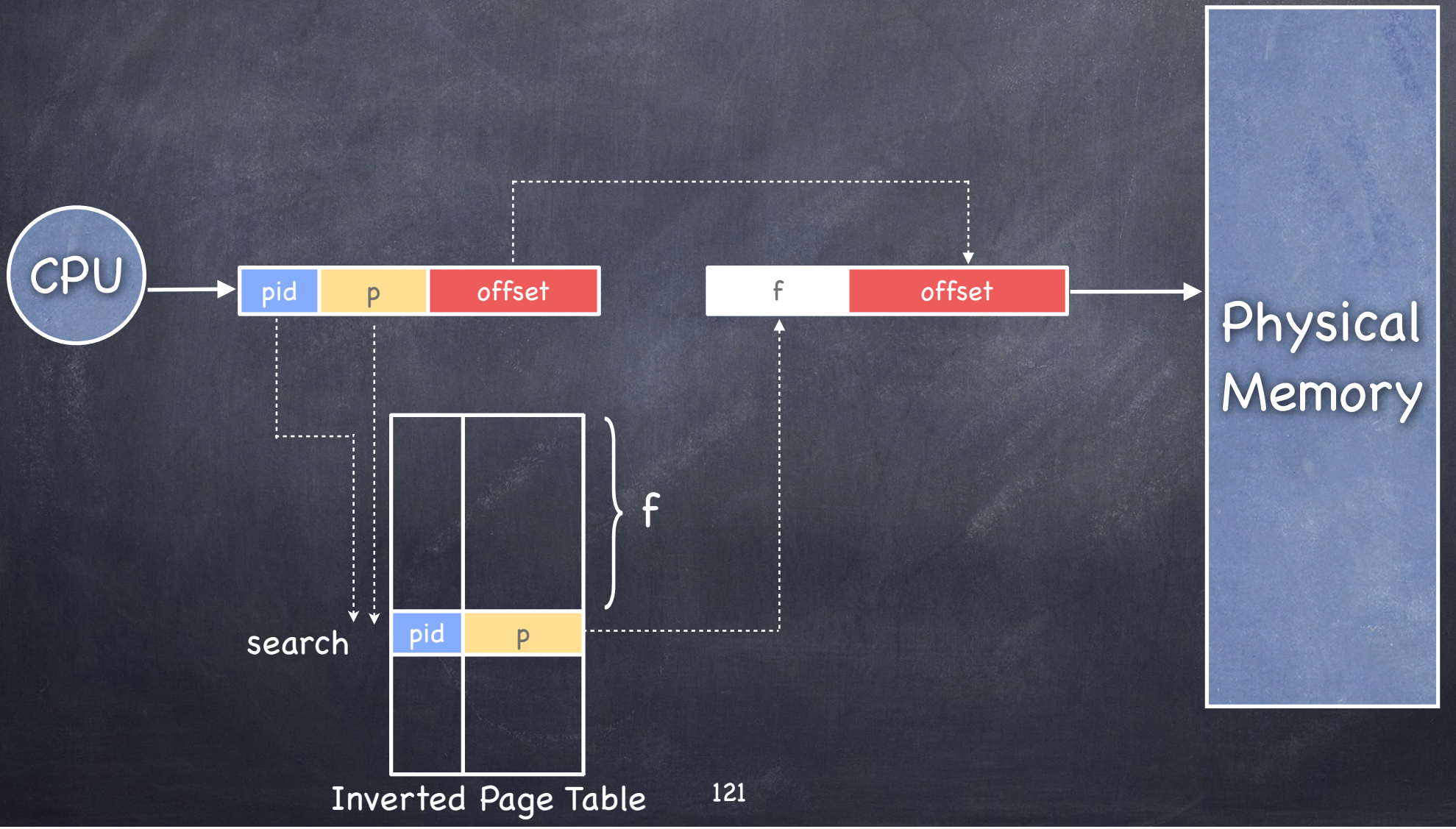
Inverted Page Table

- For each frame, a register containing
  - Residence bit
    - is the frame occupied?
  - Number of the occupying page
  - Protection bits
- Searched by page number

Catch?

- Multiple processes may map the same page number to different frames — which one is the right one?
  - add pid to IPT entry

Basic Inverted Page Table Architecture

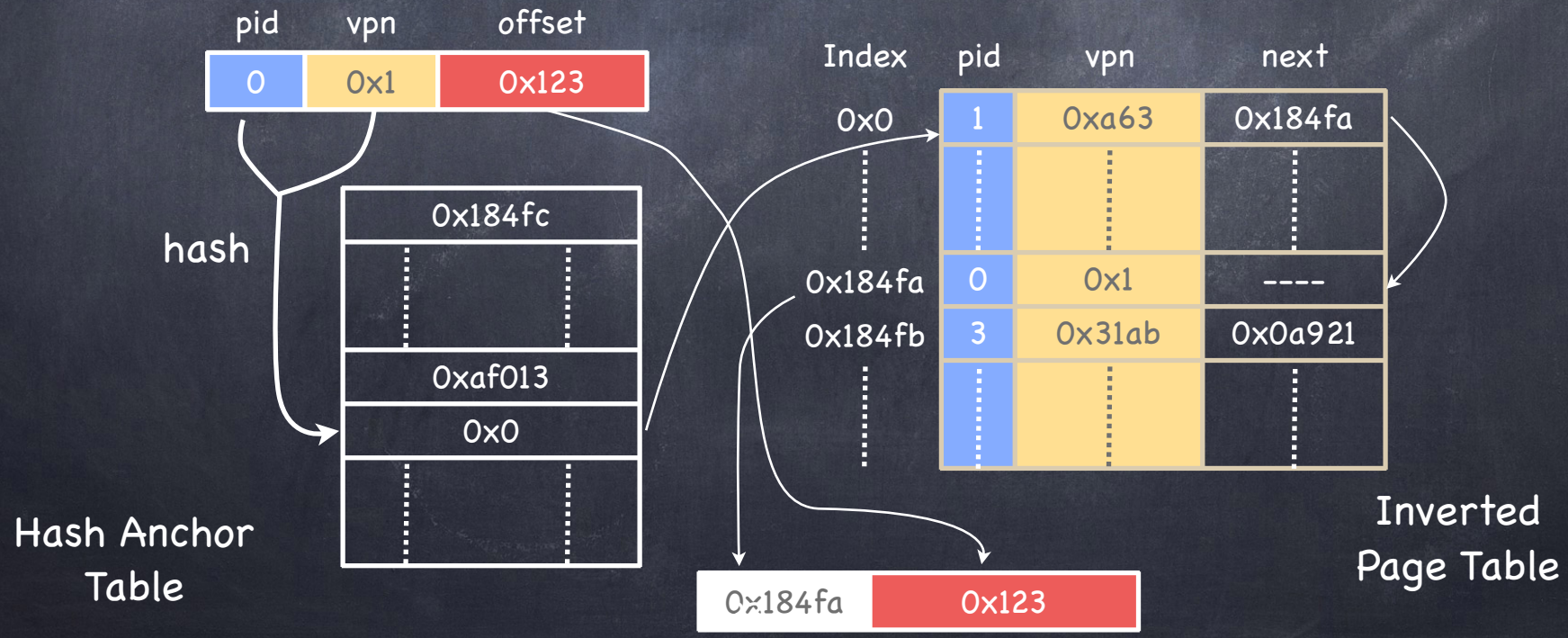


Discussion

- Less memory to store page tables
- More time to search page tables
  - searching linearly a long list of entries is no fun
  - and using associative memory is too expensive
- Solution: hashing

Hashed Inverted Page Tables

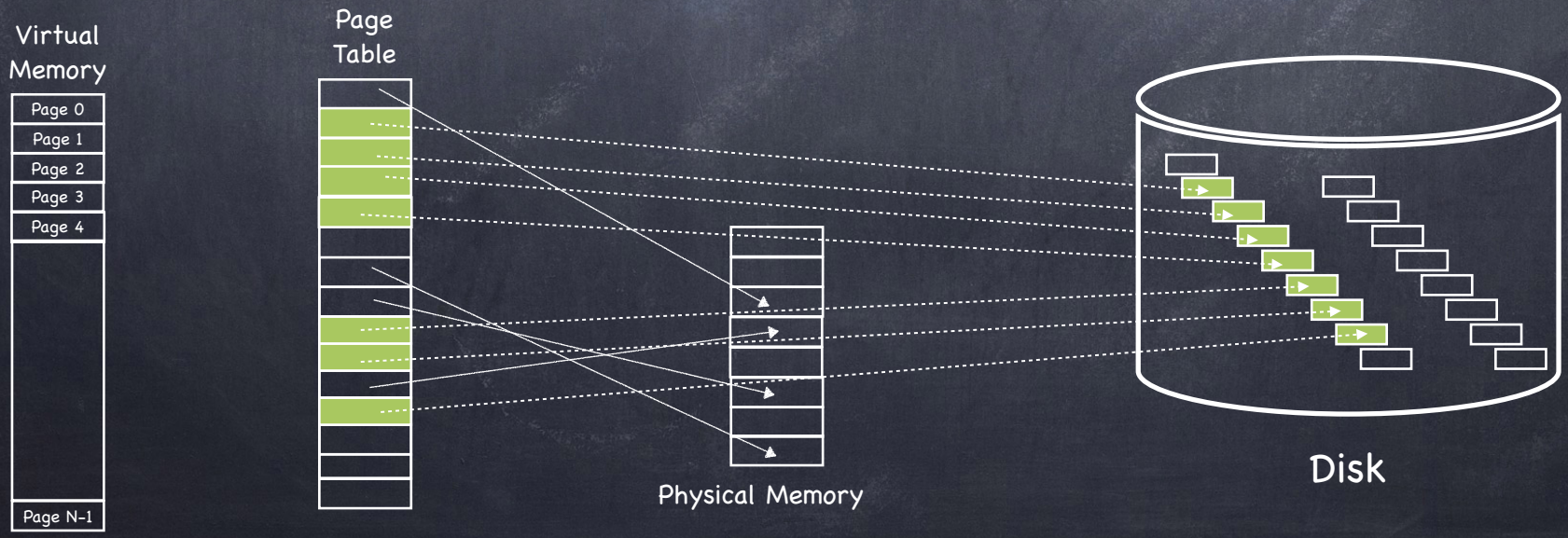
- Add a Hash Anchor Table, mapping <pid, VP#> to an entry of the Inverted Page Table
- Collisions handled by chaining



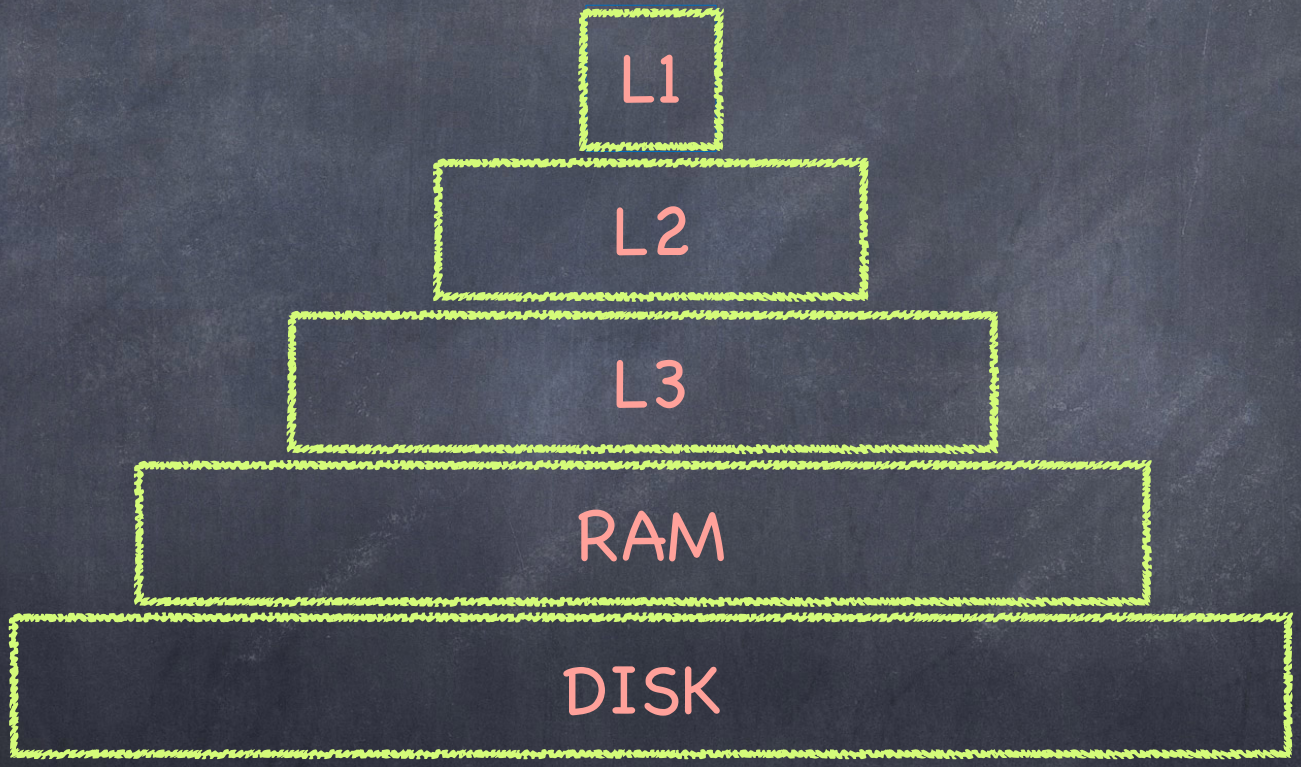
Virtual Memory

Virtual Memory

- Each process has the illusion of a large address space
  - 2^x bytes for x-bit addressing
- However, physical memory is usually much smaller
  - and we want to run multiple processes concurrently
- How do we give this illusion to multiple processes?
  - Virtual Memory: back every memory segment with a file on disk



Processes execute from disk!



RAM is just another layer of cache!

Swapping vs. Paging

- Swapping
  - Loads entire process in memory
  - "Swap in" (from disk) or "Swap out" (to disk) a process
  - Slow (for large processes)
  - Wasteful (might not require everything)
  - Does not support sharing of code segments
  - Virtual memory limited by size of physical memory
- Paging
  - Runs all processes concurrently
  - A few pages from each process live in memory (the rest is on disk)
  - Finer granularity, higher performance
  - Large virtual memory supported by small physical memory
  - Certain pages (e.g., read-only ones) can be shared among processes

A Virtual Page can be...

- Mapped (present bit set in PTE)
  - to a physical frame, with certain r/w/x permissions (may trigger Page Fault)
- Not mapped (present bit not set in PTE)
  - paired to a frame, but not currently mapped
  - or needing to be zero-filled (heap, BSS, stack)
  - or on backing store (paged or swapped out)

  (Page Fault)
  - or not part of one of the processes' segments
    - Segmentation Fault!

Handling a Page Fault

- Identify page and reason
  - access inconsistent with segment access rights
    - terminate process
  - access a page currently on disk
    - does frame with the code/data already exist?
      - if not, allocate a frame and load page in
  - access of page data (BSS) or stack to be 0-filled
    - allocate a frame, initialize all bytes to 0
  - access (for write) of a COW page
    - allocate a frame and copy

When a page must be brought in...

- Find a free frame
  - evict one if there are no free frames
- Issue disk request to fetch data for page
- Move "current process" to disk queue
- Context switch to new process
- Update PTE when disk completes
  - frame number, present bit, RWX bits, etc.
- Move "current process" to ready queue

When a page must be swapped out...

- Find all page table entries that refer to old page
  - Frame might be shared by multiple processes
  - Access Core Map (frames —> pages)
- Set each page table entry to not present (invalid)
- Remove any TLB entries
  - "TLB Shootdown": in multiprocessors, TLB entry must be eliminated from the TLB of all processors
- Write page back to disk, if needed
  - Dirty bit in PTE indicates need

Demand Paging

MIPS Style

1. TLB Miss
2. Exception to kernel
3. Page Table walk
4. Page fault (present bit not set in Page Table)
5. Convert VA to file offset
6. Allocate page frame (evict page if needed)
7. Initiate disk block read into page frame
8. Disk interrupt when DMA completes
9. Mark page as present
10. Update TLB
11. Resume process at faulting instruction
12. TLB hit
13. Execute instruction

Software handling page fault

Demand Paging:
x86 Style

1. TLB Miss
2. Page Table walk
3. Page fault (page not present in Page Table)
4. Exception to kernel
5. Convert VA to file offset
6. Allocate page frame (evict page if needed)
7. Initiate disk block read into page frame
8. Disk interrupt when DMA completes
9. Mark page as present
10. Resume process at faulting instruction
11. TLB miss
12. Page Table walk – success!
13. TLB updated
14. Execute instruction

Software handling page fault

OS Support for Demand Paging

- Process Creation
  - Create and initialize page table and PCB
- Process Execution
  - Reset MMU (PTBR) for new process
  - Context switch: flush TLB
    - unless TLB is tagged with PIDs
  - Handle page faults
- Process Termination
  - Release pages
- Page Daemon
  - proactively prepares pages for eviction

Page Replacement

- When physical memory is full, we need to choose a "victim" to evict
- Local vs Global replacement
  - Local: victim chosen from frames of process experiencing page fault
    - fixed allocation of frames per process
  - Global: victim chosen from frames allocated to any process
    - variable allocation of frames per process
- Goal: minimizing number of page faults

Page Replacement Algorithms

- Random: Pick any page to eject at random
  - Used mainly for comparison
- FIFO: The page brought in earliest is evicted
  - Ignores usage
- LRU: Evict page not been used the longest
  - Assumes past is good predictor of the future
- MRU: Evict most recently used page
  - Good for data accessed only once, e.g., a movie
- LFU: Evict least frequently used page
- OPT: Belady's algorithm

How do we pick a victim?

- We want:
  - low page fault-rate
  - page faults as inexpensive as possible
- We need:
  - a way to compare the relative performance of different page replacement algorithms
  - some absolute notion of what a "good" page replacement algorithm should accomplish

Comparing Page Replacement Algorithms

- Record a trace of the pages accessed by a process
  - E.g. 3,1,4,2,5,2,1,2,3,4 (or c,a,d,b,e,b,a,b,c,b)
- Simulate behavior of page replacement algorithm on trace
- Record number of page faults generated

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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									
	1			b	b									
	2				c									
Faults		X	X	X	X									

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames (3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames (3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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						
	1			b	b	b	b	b						
	2				c	d	d	d						
Faults		X	X	X	X	✓	✓	X						

Process can use 3 frames (3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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	a	a	a			
	1			b	b	b	b	b	b	b	b			
	2				c	d	d	d	e	e	e			
Faults		X	X	X	X	✓	✓	X	✓	✓	X			

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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	a	a	a	c		
	1			b	b	b	b	b	b	b	b	b		
	2				c	d	d	d	e	e	e	e		
Faults		X	X	X	X	✓	✓	X	✓	✓	X	X		

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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	a	a	a	c	c	
	1			b	b	b	b	b	b	b	b	b	d	
	2				c	d	d	d	e	e	e	e	e	
Faults		X	X	X	X	✓	✓	X	✓	✓	X	X	✓	

Process can use 3 frames
(3 pages in memory)

Page loaded

Optimal Page Replacement

- Replace page needed furthest in future

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	a	a	a	c	c	c
	1			b	b	b	b	b	b	b	b	b	d	d
	2				c	d	d	d	e	e	e	e	e	e
Faults		X	X	X	X	✓	✓	X	✓	✓	X	X	✓	

7 page faults

Process can use 3 frames (3 pages in memory)

Page loaded

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	a	a	a	a	d	d
	1			b	b	b	b	b	b	b	b	b	a	e
	2				c	c	c	c	c	c	c	c	b	b
	3					d	d	d	e	e	e	e	e	c
Faults		X	X	X	X	✓	✓	X	✓	✓	✓	X	✓	

6 page faults

Process can use 4 frames (4 pages in memory)

FIFO Replacement

- Replace pages in the order they come into memory

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	✓	

Process can use 3 frames (3 pages in memory)

Page loaded

9 page faults

FIFO Replacement

- Replace pages in the order they come into memory

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	b	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	

Process can use 4 frames (4 pages in memory)

Page loaded

10 page faults

More frames —> more page faults?

Belady's Anomaly

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

LRU: Least Recently Used

- Replace page not referenced for the longest time

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

Process can use 4 frames
(4 pages in memory)

LRU: Least Recently Used

- Replace page not referenced for the longest time

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

Process can use 4 frames
(4 pages in memory)

LRU: Least Recently Used

- Replace page not referenced for the longest time

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

Process can use 4 frames
(4 pages in memory)

LRU: Least Recently Used

- Replace page not referenced for the longest time

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	a	a	a	a	a	e
	1			b	b	b	b	b	b	b	b	b	b	b
	2				c	c	c	c	e	e	e	e	d	d
	3					d	d	d	d	d	d	c	c	c
Faults		X	X	X	X	✓	✓	X	✓	✓	X	X	X	

8 page faults

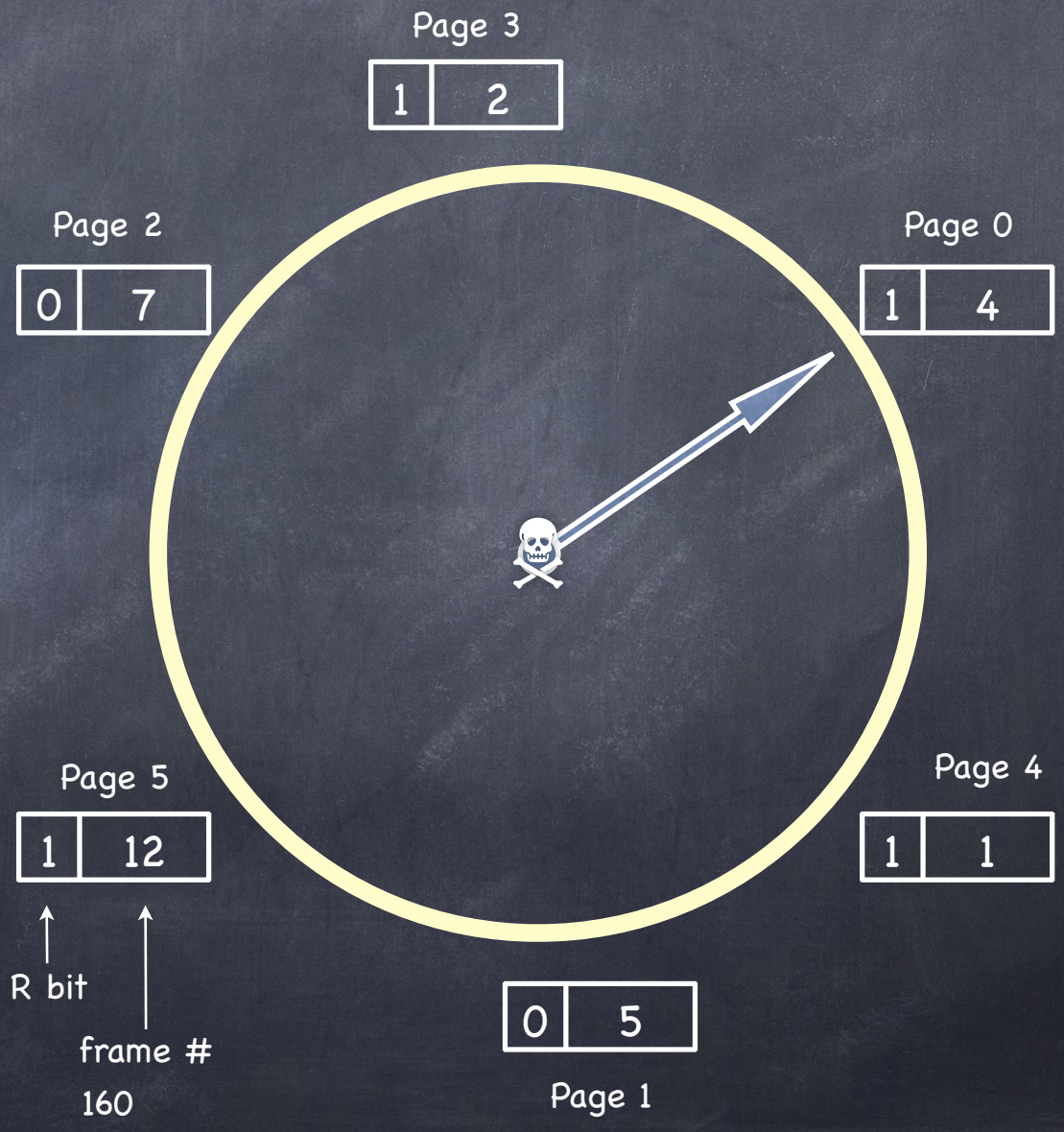
Process can use 4 frames
(4 pages in memory)

Implementing LRU

- On reference: timestamp each page
- On eviction: scan for oldest page
- Problems:
  - Large page lists
  - Timestamps are costly
- Solution: approximate LRU
  - after all, LRU is already an approximation! (of OPT)
  - exploit Reference (R) bit

The Clock Algorithm

- Organize pages in memory as a circular list
- When page is referenced, set its reference bit R to 1
- On page fault
  - if R = 1: clear R
  - else if R = 0:
    - evict page pointed to
    - load new page
    - set R to 1
  - advance hand



Clock Page Replacement

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

Page table entries for resident pages

1	a
1	b
1	c
1	d

Hand clock:

Clock Page Replacement

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

Page table entries for resident pages

1	a
1	b
1	c
1	d

Hand clock:

Clock Page Replacement

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

Page table entries for resident pages

1	a
1	b
1	c
1	d

1	e
0	b
0	c
0	d

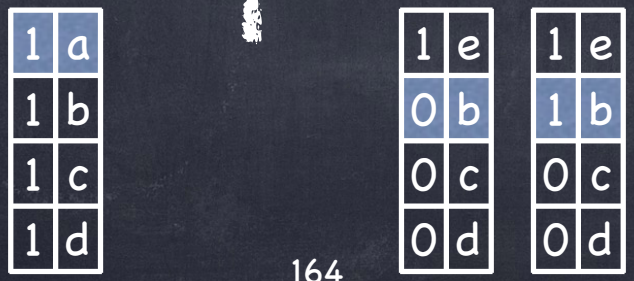
Hand clock:

Clock Page Replacement

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

Page table entries for resident pages

Hand clock:

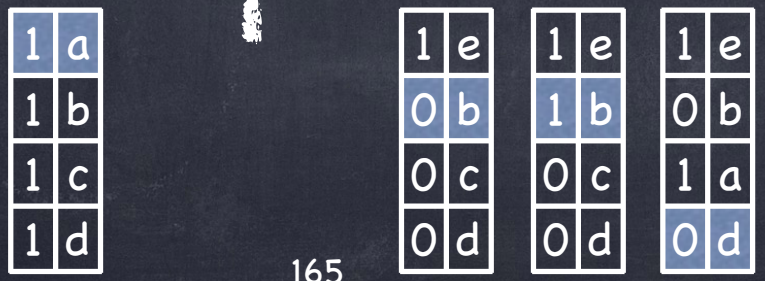


Clock Page Replacement

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

Page table entries for resident pages

Hand clock:

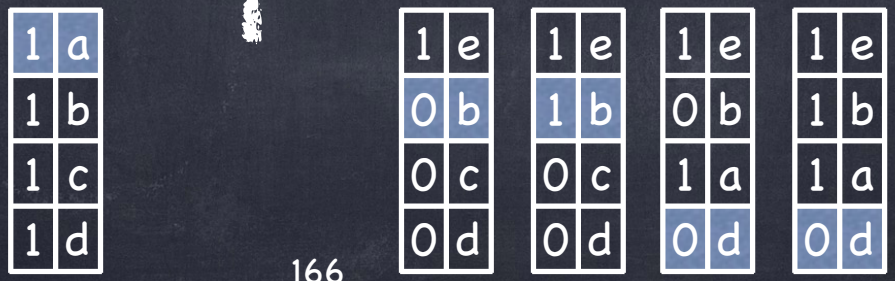


Clock Page Replacement

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

Page table entries for resident pages

Hand clock:



Clock Page Replacement

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

Page table entries for resident pages

1	a
1	b
1	c
1	d

1	e
0	b
0	c
0	d

1	e
1	b
0	c
0	d

1	e
0	b
1	a
0	d

1	e
1	b
1	a
0	d

1	e
1	b
1	a
1	c

Hand clock: ■

Clock Page Replacement

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

Page table entries for resident pages

1	a
1	b
1	c
1	d

1	e
0	b
0	c
0	d

1	e
1	b
0	c
0	d

1	e
0	b
1	a
0	d

1	e
1	b
1	a
0	d

1	e
1	b
1	a
1	c

1	d
0	b
0	a
0	c

Hand clock:

The Second Chance Algorithm

- Dirty pages get "second chance" before eviction
  - replacing dirty pages is expensive!

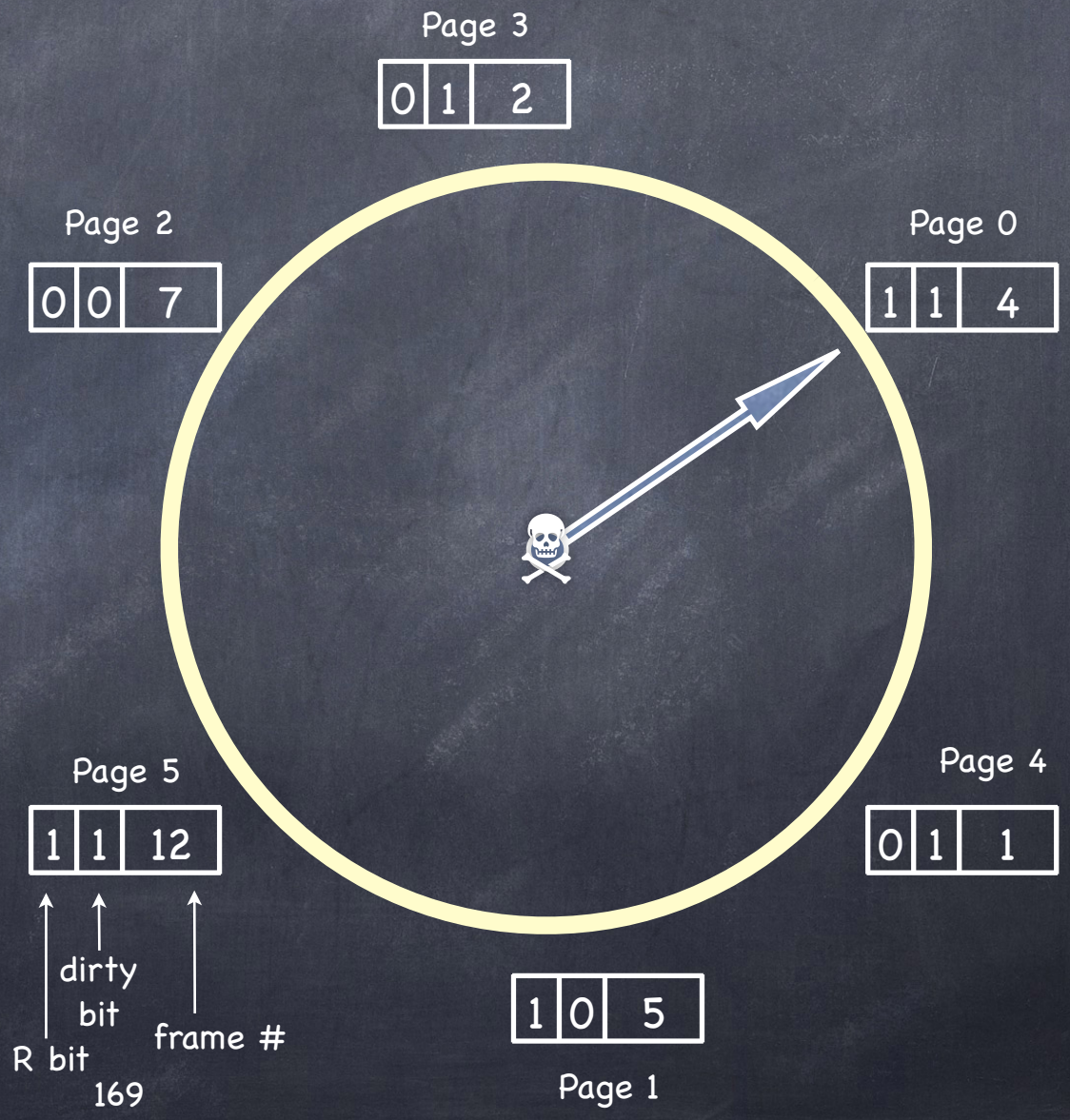
Before Clock sweep

R	dirty
0	0
1	0
0	1
1	1

After Clock sweep

R	dirty
replace page	
0	0
0	0*
0	1

* = remember when evicted must be saved to disk!



Second Chance Page Replacement

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

Page table entries for resident pages

10	a
10	b
10	c
10	d

Hand clock:

Second Chance Page Replacement

Time		0	1	2	3	4	5	6	7	8	9	10
Requests		c	a^w	d	b^w	e						
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						
Faults						X						

Page table entries for resident pages

10	a
10	b
10	c
10	d

11	a
11	b
10	c
10	d

Hand clock:

Second Chance Page Replacement

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

Page table entries for resident pages

10	a
10	b
10	c
10	d

11	a
11	b
10	c
10	d

00	a
00	b
10	e
00	d

Hand clock: 

Second Chance Page Replacement

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

Page table entries for resident pages

10	a
10	b
10	c
10	d

11	a
11	b
10	c
10	d

00	a
00	b
10	e
00	d

00	a
10	b
10	e
00	d

Hand clock:

Second Chance Page Replacement

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

Page table entries for resident pages

10	a
10	b
10	c
10	d

11	a
11	b
10	c
10	d

00	a
00	b
10	e
00	d

00	a
10	b
10	e
00	d

Hand clock:

Second Chance Page Replacement

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

Page table entries for resident pages

10	a
10	b
10	c
10	d

11	a
11	b
10	c
10	d

00	a
00	b
10	e
00	d

00	a
10	b
10	e
00	d

11	a
10	b
10	e
00	d

Hand clock:

Second Chance Page Replacement

Time		0	1	2	3	4	5	6	7	8	9	10
Requests		c	a^w	d	b^w	e	b	a^w	b	c		
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		
Faults						X				X		

Page table entries for resident pages

10	a	11	a	00	a	00	a	11	a
10	b	11	b	00	b	10	b	10	b
10	c	10	c	10	e	10	e	10	e
10	d	10	d	00	d	00	d	00	d

Hand clock:

Second Chance Page Replacement

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

Page table entries for resident pages

10	a	11	a	00	a	00	a	11	a	11	a				
10	b	11	b	00	b	10	b	10	b	10	b				
10	c	10	c	10	e	10	e	10	e	10	e				
10	d	10	d	00	d	00	d	00	d	10	c				

Hand clock: 

Second Chance Page Replacement

Time	0	1	2	3	4	5	6	7	8	9	10
Requests	c	a^w	d	b^w	e	b	a^w	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	d
2	c	c	c	c	c	e	e	e	e	e	e
3	d	d	d	d	d	d	d	d	d	c	c
Faults					X				X	X	

Page table entries for resident pages

10	a	11	a	00	a	00	a	11	a	11	a	00	a
10	b	11	b	00	b	10	b	10	b	10	b	10	d
10	c	10	c	10	e	10	e	10	e	10	e	00	e
10	d	10	d	00	d	00	d	00	d	10	c	00	c

Hand clock:

Back to Belady's Anomaly

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	

3 frames:
9 page faults

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

4 frames:
10 page faults!

Taming Belady:
Stack Page Replacement Policies

- Given m frames for trace r, let M(m,r) be the set of virtual pages in physical memory
- 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)
  - A 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

Local vs. Global Page Replacement

- Local: processes select victim among frames allocated to them
  - Can lead to under utilization
- Global: Select any frame, even if allocated to another process