Virtual Memory & Caching

(Chapter 9)

CS 4410
Operating Systems

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

Last Time: Address Translation

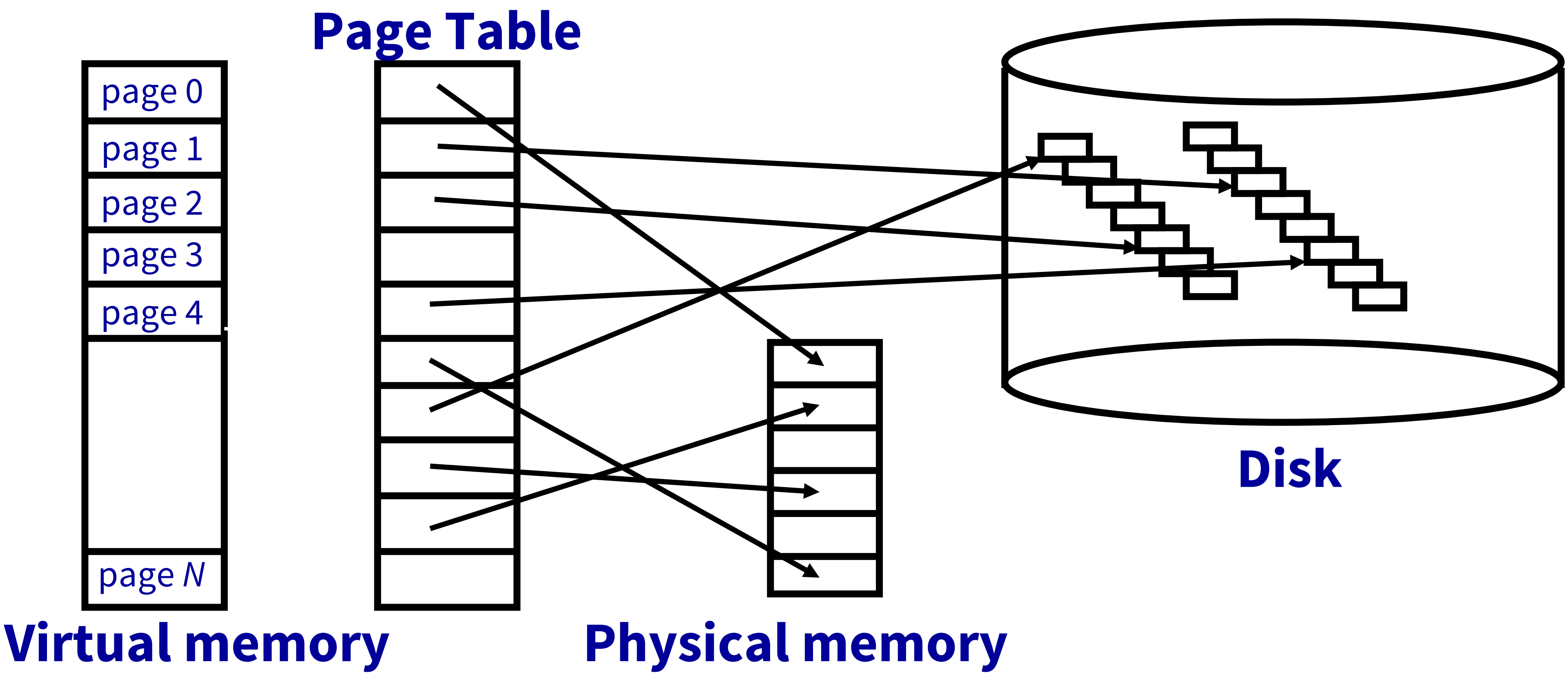
- Paged Translation
- Efficient Address Translation
  - Multi-Level Page Tables
  - Inverted Page Tables
  - TLBs

This time: **Virtual Memory & Caching**

- **Virtual Memory**
- Caching

What is Virtual Memory?

- Each process has illusion of large address space
  - 2^{32} for 32-bit addressing
- However, physical memory is much smaller
- How do we give this illusion to multiple processes?
  - Virtual Memory: some addresses reside in disk



Swapping vs. Paging

Swapping

- Loads entire process in memory, runs it, exit
- “Swap in” or “Swap out” a process
- Slow (for big, long-lived processes)
- Wasteful (might not require everything)

Paging

- Runs all processes concurrently
- A few pages from each process live in memory
- Finer granularity, higher performance
- Large virtual mem supported by small physical mem

“to swap” (pushing contents out to disk in order to bring other content from disk) ≠ “swapping”

(the contents of) **A Virtual Page Can Be**

Mapped

- to a physical frame

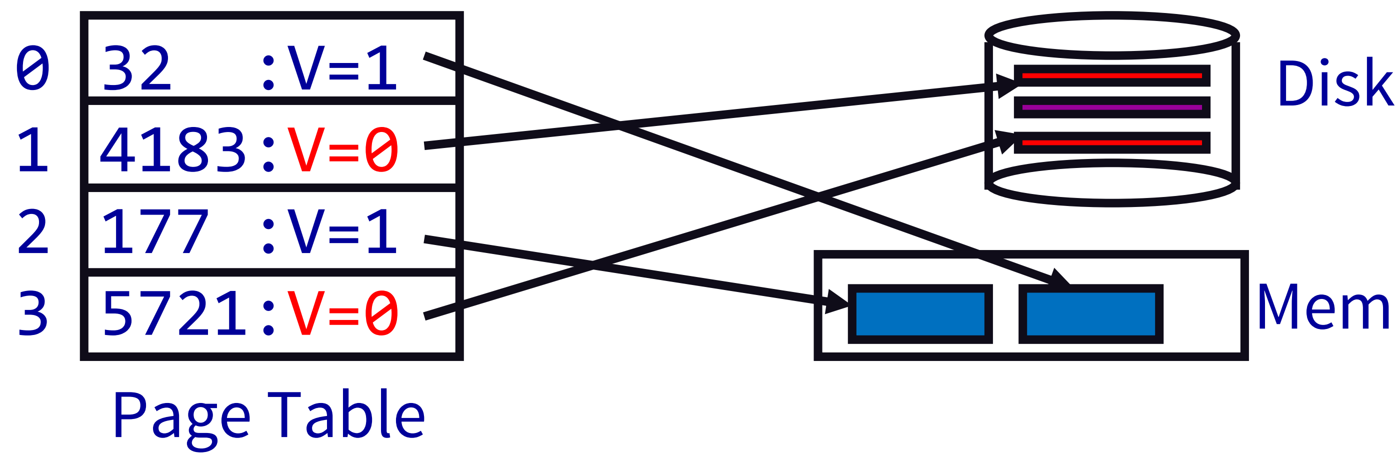
Not Mapped (→ Page Fault)

- in a physical frame, but not currently mapped
- still in the original program file
- zero-filled (heap/BSS, stack)
- on backing store (“paged or swapped out”)
- illegal: not part of a segment
  → Segmentation Fault

Supporting Virtual Memory

Modify Page Tables with a valid bit (= "present bit")

- Page in memory → *valid = 1*
- Page not in memory → PT lookup triggers **page fault**



Handling a Page Fault

Identify page and reason (r/w/x)

- access inconsistent w/ segment access rights
  → terminate process

- access of code or data segment:
  → does frame with the code/data already exist?
  No? Allocate a frame & bring page in (next slide)

- access of zero-initialized data (BSS) or stack
  - Allocate a frame, fill page with zero bytes

When a page needs to be brought in…

- Find a free frame
  - or evicts one from memory (next slide)
  - which one? (next lecture)
- Issue disk request to fetch data for page
  - what to fetch? (requested page or more?)
- Block current process
- Context switch to new process
- When disk completes, set valid bit to 1 (& other permission bits), put current process in ready queue

When a page is swapped out…

- Find all page table entries that refer to old page
  - Frame might be shared
  - Core Map (frames → pages)
- Set each page table entry to invalid
- Remove any TLB entries
  - Hardware copies of now invalid PTE
  - “TLB Shootdown”
- Write changes on page back to disk, if needed
  - Dirty/Modified bit in PTE indicates need
  - Text segments are (still) on program image on disk

Demand Paging, MIPS style

1. TLB miss
2. Trap to kernel
3. Page table walk
4. Find page is invalid
5. Convert virtual address to file + offset
6. Allocate frame
   - Evict if needed
7. Initiate disk block read into frame
8. Disk interrupt when DMA complete
9. Mark page valid
10. Update TLB
11. Resume process at faulting instruction
12. Execute instruction

Demand Paging, x86 style

1. TLB miss
2. Page table walk
3. Page fault (find page is invalid)
4. Trap to kernel
5. Convert virtual address to file + offset
6. Allocate frame
   - Evict if needed
7. Initiate disk block read into frame
8. Disk interrupt when DMA complete
9. Mark page valid
10. Resume process at faulting instruction
11. TLB miss
12. Page table walk to fetch translation
13. Execute instruction

Updated Context Switch

- Save current process' registers in PCB
  - Also Page Table Base Register (PTBR)
- ***Flush TLB*** *(if no pids)*
- Page Table itself is in main memory
- Restore registers of next process to run
- "Return from Interrupt"

OS Support for Paging

Process Creation

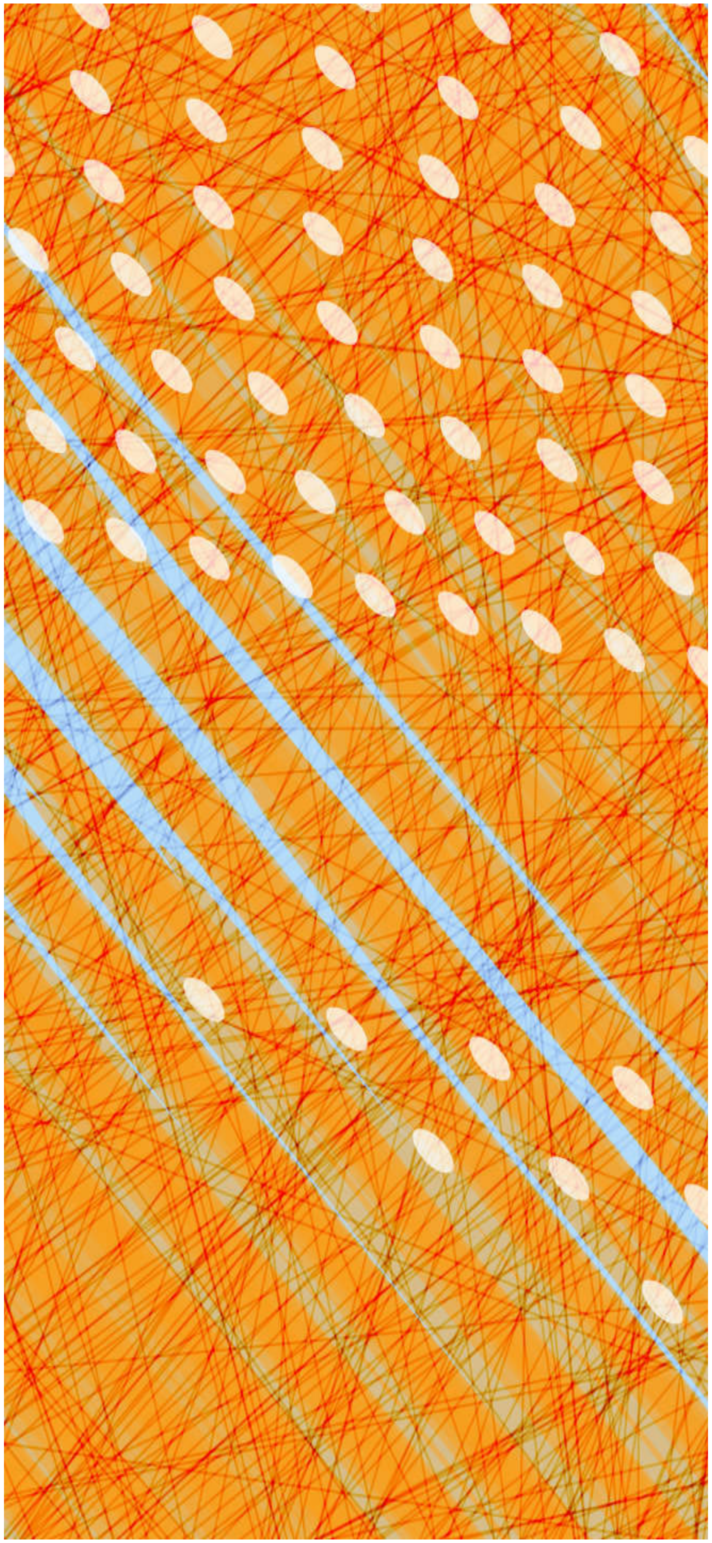
- Allocate frames, create & initialize page table & PCB

Process Execution

- Reset MMU (PTBR) for new process
- Context switch: flush TLB (or TLB has pids)
- Handle page faults

Process Termination

- Release pages



- Virtual Memory
- **Caching**

What are some examples of caching?

- TLBs
- hardware caches
- internet naming
- web content
- web search
- email clients
- incremental compilation
- just in time translation
- virtual memory
- file systems
- branch prediction

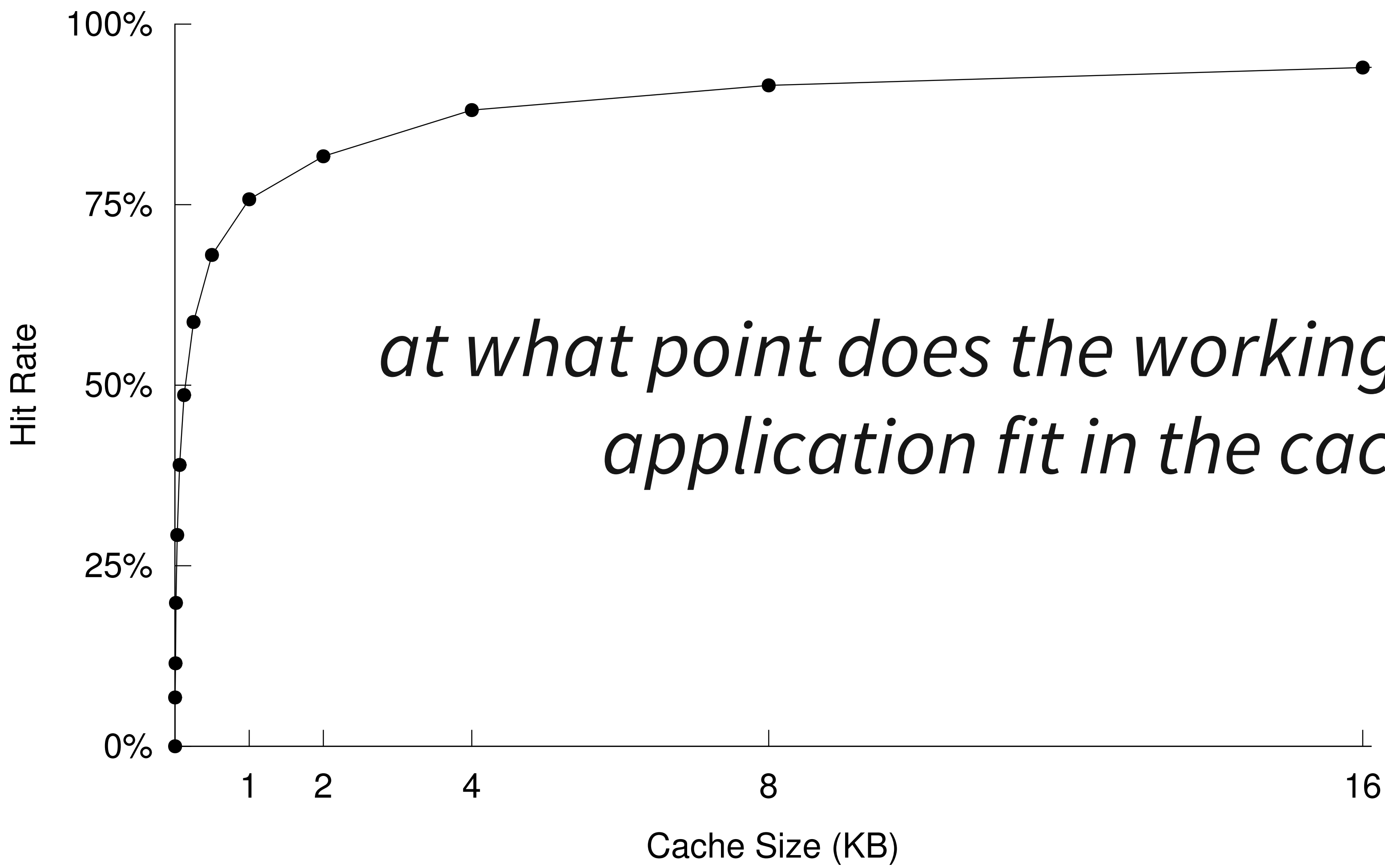
Memory Hierarchy

Cache	Hit Cost	Size
1st level cache / 1st level TLB	1 ns	64 KB
2nd level cache / 2nd level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

Every layer is a cache for the layer below it.

Working Set

1. Collection of a process' most recently used pages
   (The Working Set Model for Program Behavior, Denning,'68)
2. Pages referenced by process in last Δ time-units



at what point does the working set of this application fit in the cache?

Thrashing

Excessive rate of paging
Cache lines evicted before they can be reused

Causes:

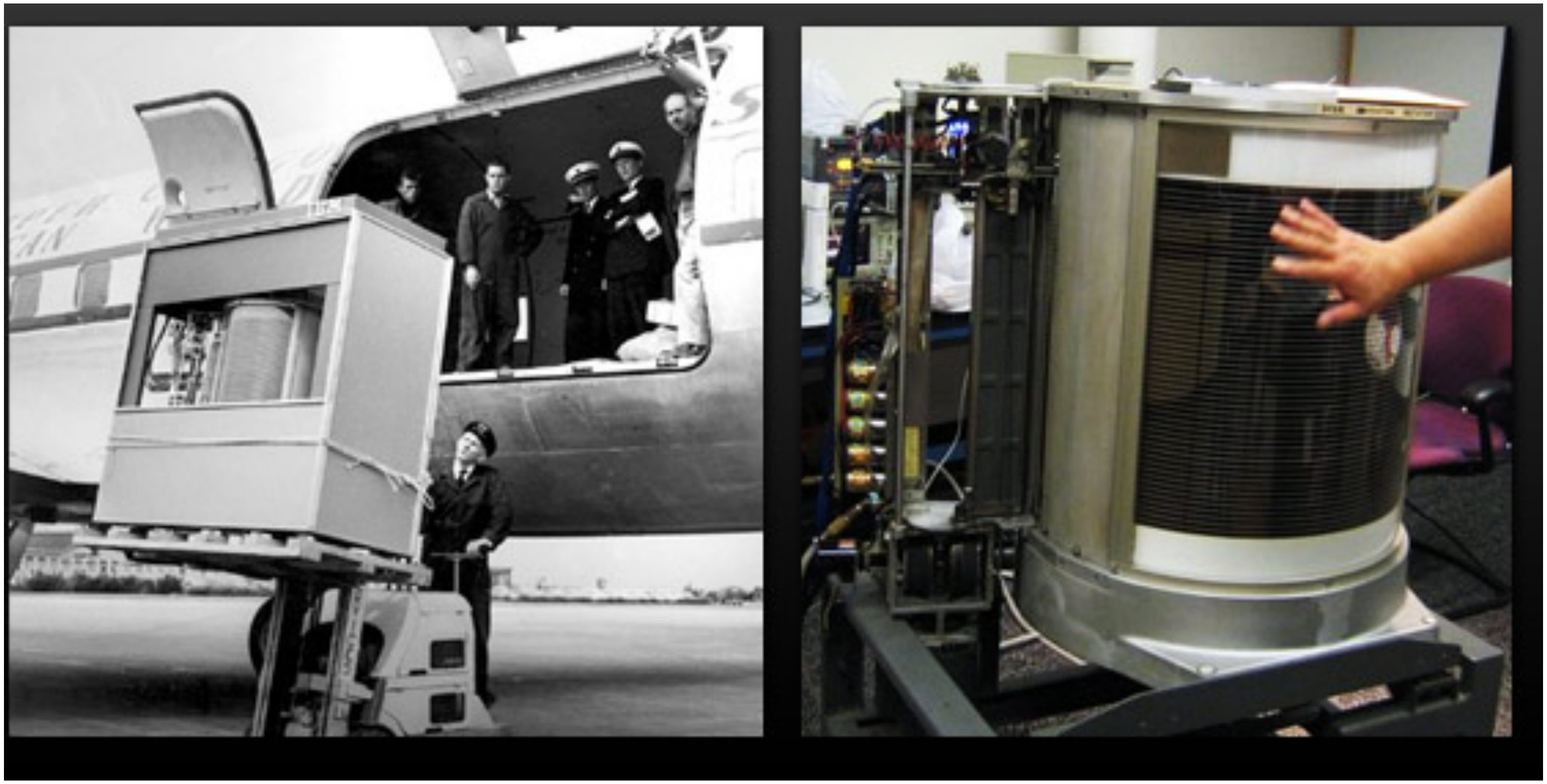
- Too many processes in the system
- Cache not big enough to fit working set
- Bad luck (conflicts)
- Bad eviction policies (later)

Prevention:

- restructure your code

  (smaller working set, shift data around)
- restructure your cache (↑ capacity, ↑ associativity)

Why "thrashing"?



The first hard disk drive—the IBM Model 350 Disk File (came w/IBM 305 RAMAC, 1956).

Total storage = 5 million characters (just under 5 MB).

http://royal.pingdom.com/2008/04/08/the-history-of-computer-data-storage-in-pictures/

"Thrash" dates from the 1960's, when disk drives were as large as washing machines. If a program's working set did not fit in memory, the system would need to shuffle memory pages back and forth to disk. This burst of activity would violently shake the disk drive.

Caching

- Assignment: where do you put the data?
- Replacement: who do you kick out?

Caching

- Assignment: where do you put the data?
  - Which entry in the cache? **— not much choice**
  - Which frame in memory? **— lots of freedom**
- Replacement: who do you kick out?

Address Translation Problem

- Adding a layer of indirection disrupts the spatial locality of caching

- What if virtual pages are assigned to physical pages that are n cache sizes apart?

→**BIG PROBLEM:**

cache effectively smaller

Solution: Cache Coloring (Page Coloring)

1. Color frames according to cache configuration.

2. Spread each process' pages across as many colors as possible.

Caching

- Assignment: where do you put the data?
- **Replacement: who do you kick out?**

What do you do when Memory is full?

Caching

- Assignment: where do you put the data?
- **Replacement: who do you kick out?**
  - Random: pros? cons?
  - FIFO
  - MIN
  - LRU
  - LFU
  - Approximating LRU

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
- **OPT:** Belady's algorithm
  - Select page not used for longest time
- **LRU:** Evict page that hasn't been used for the longest
  - Past could be a good predictor of the future
- **MRU:** Evict the most recently used page
- **LFU:** Evict least frequently used page

First-In-First-Out (FIFO) Algorithm

- *Reference string*: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- **3 frames** (3 pages in memory at a time per process):

frames			reference
			1
		1	2
	2	1	3
3	2	1	4
3	2	**4**	1
3	**1**	4	2
2	1	4	5
2	1	**5**	1 (hit)
2	1	5	2 (hit)
2	1	5	3
2	**3**	5	4
4	3	5	5 (hit)
4	3	5	

← contents of frames at time of reference

page fault

hit

4 marks arrival time

9 page faults

First-In-First-Out (FIFO) Algorithm

- *Reference string*: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- **4 frames** (4 pages in memory at a time per process):

frames				reference
				1
			1	2
		2	1	3
	3	2	1	4
4	3	2	1	1
4	3	2	1	2
4	3	2	1	5
4	3	2	**5**	1
4	3	**1**	5	2
4	**2**	1	5	3
3	2	1	5	4
3	2	3	**4**	5
3	2	**5**	4	

← contents of frames at time of reference

page fault

hit

4 marks arrival time

10 page faults

more frames → more page faults?

Belady's Anomaly

Optimal Algorithm (OPT)

- Replace page that will not be used for the longest
- 4 frames example

				1
			1	2
		2	1	3
	3	2	1	4
4	3	2	1	1
4	3	2	1	2
4	3	2	1	5
5	3	2	1	1
5	3	2	1	2
5	3	2	1	3
5	3	2	1	4
5	3	2	4	5
5	3	2	4	

6 page faults

Question: How do we tell the future?
Answer: We can't

OPT used as upper-bound in measuring how well your algorithm performs

OPT Approximation

In real life, we do not have access to the future page request stream of a program

- No crystal ball
- no way to know which pages a program will access

→ Need to make a best guess at which pages will not be used for the longest time

Least Recently Used (LRU) Algorithm

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

				1
			1	2
		2	1	3
	3	2	1	4
4	3	2	1	1
4	3	2	**1**	2
4	3	**2**	1	5
4	**5**	2	1	1
4	5	2	**1**	2
4	5	**2**	1	3
3	5	2	1	4
3	**4**	2	1	5
3	4	2	**5**	

page fault

hit

4 marks most recent use

8 page faults

Implementing* Perfect LRU

- On reference: Timestamp each page
- On eviction: Scan for oldest frame

Problems:

- Large page lists
- Timestamps are costly

Solution: **approximate LRU**

Q: “I thought LRU was already an approximation…”

A: “It is... Oh well…”

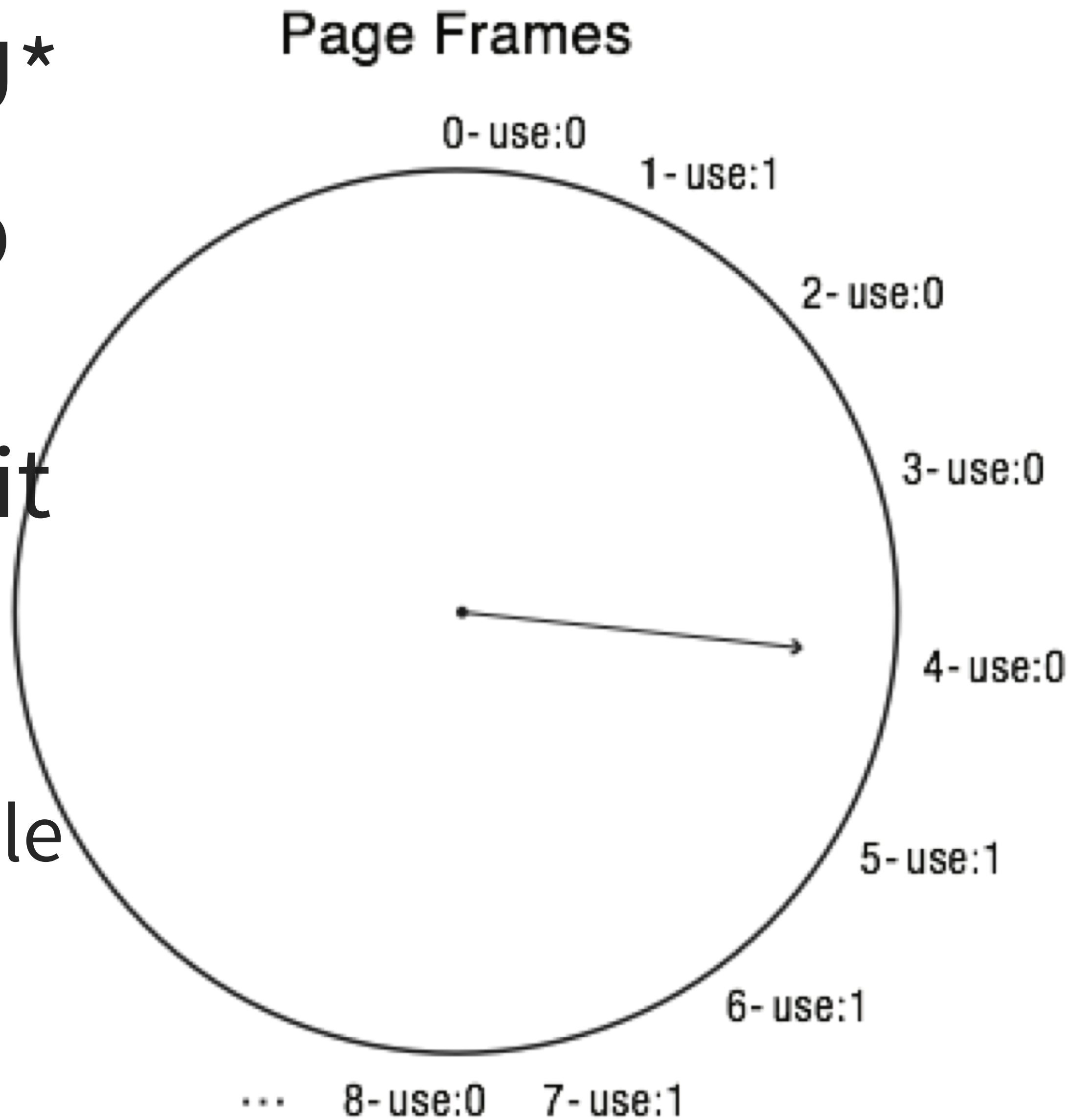
* the blue shading in the previous frame diagram

Clock Algorithm: Not Recently Used

Approximating LRU*

Periodically, sweep through all pages

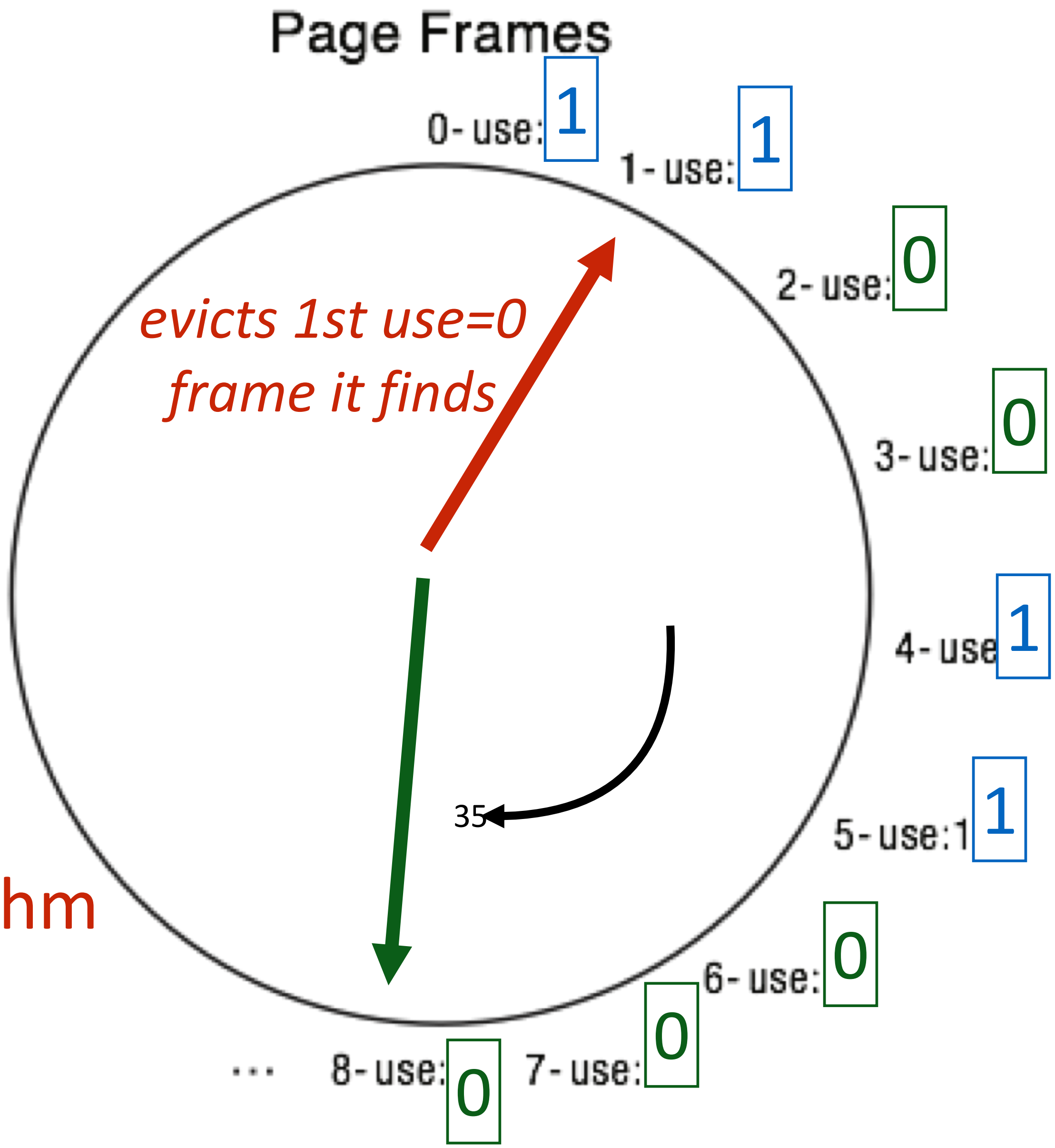
- Used? Clear use bit
- Unused? reclaim
  - update core map
  - invalidate page table
  - write back if dirty
  - TLB shootdown
  - add to free list



*(*yes, LRU was already an approximation...)*

Clock Algorithm Problems

blue 1's were used after use bit was cleared by green hand



What if Memory is Large?

Leading edge clears use bit
- slowly clears history
- finds victim candidates

Trailing edge evicts pages with use bit set to 0
- fast: original clock algorithm
- slow: all pages look used

Big angle? Small angle?

Other Algorithms

MRU: Remove the most recently touched page

- Good for data accessed only once, *e.g.* a movie file
- Not a good fit for most other data, *e.g.* frequently accessed items

LFU: Remove page with lowest usage count

- No record of *when* the page was referenced
- Use multiple bits. Shift right by 1 at regular intervals.

MFU: remove the most frequently used page

LFU and MFU do not approximate OPT well

A4: You will build a disk cache

How do you know:

- if your cache is caching?
- how well your cache is caching?