CIS 5512 - Operating Systems
Memory Management – Cache Replacement & Review

Professor Qiang Zeng

Fall 2017

Previous class…

What is the relation between CoW and sharing page frames?

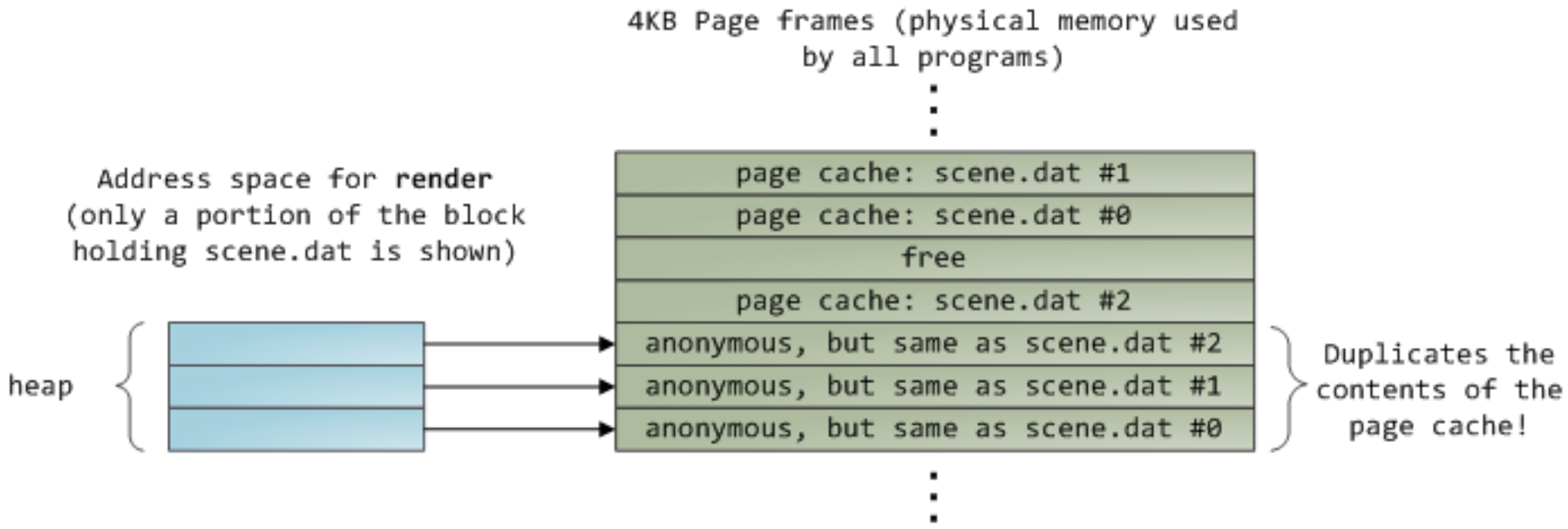
CoW is built on sharing page frames.

Previous class…

How does memory mapped file work?

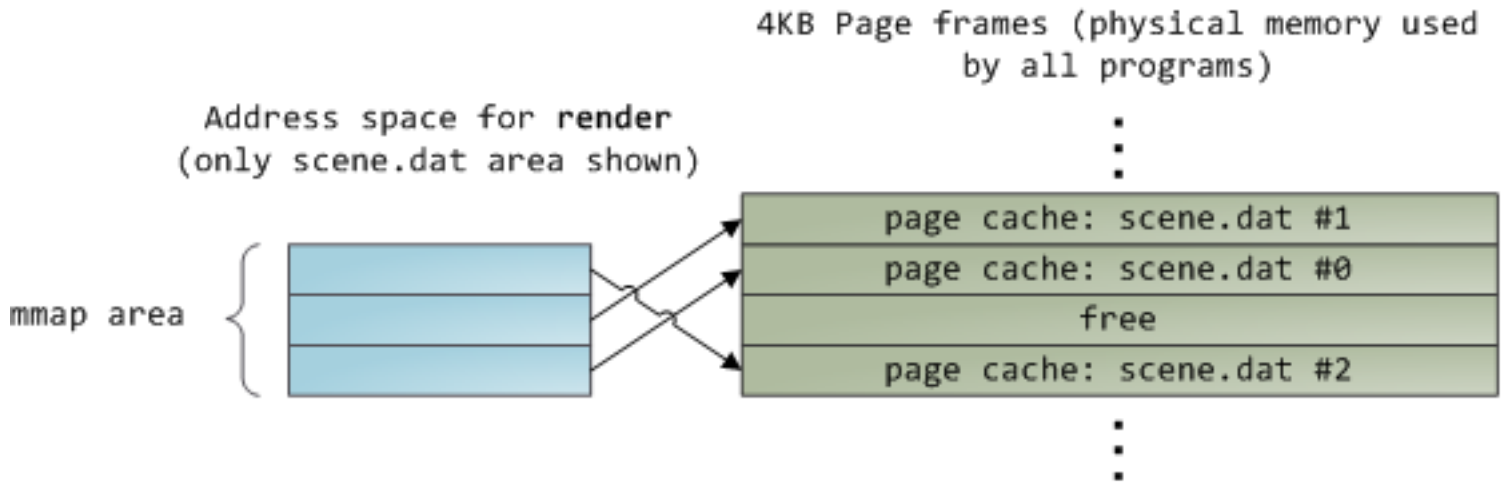
MMF works by ***mapping*** the file (or a portion of it) directly with a segment of the process's virtual address space, such that when your program accesses the file, the system will automatically load the file content into frames of page cache and manipulate the page table entries, which then point to the frames

Regular file I/O, e.g., reading 12kB



- Open the file
- Allocate a 12KB buffer
- Copy data from page cache to the buffer

Effect of memory mapped file I/O



- It is faster for two reasons:
  - No system calls such as read()/write() due to access
  - Zero-copy: operate on page cache directly
- It also saves memory for two reasons:
  - Zero-copy: no need to allocate memory for buffers
  - Lazy loading: page cache is only allocated for data being accessed (compared to allocating a big buffer for storing the data to be accessed)

Previous class…

What are the disadvantages of MMF I/O

Triggering a page fault when accessing each new page.
Probably consuming a large virtual memory area

Previous class…

How does your memory subsystem help you start a process quickly (i.e., your code and data don't need to be copied to the physical memory before execution)?

Your program loader makes use of MMF I/O + demand paging to speed up

Previous class…

Do you swap out code pages?

No, code pages are read-only, so no need

When to evict pages

- When the physical memory runs low, it is critical for the kernel to evict pages to satisfy physical memory requests smoothly

How to evict an old page

- Select an old page to evict
  - How to select? Replacement alg. (to be discussed)
- Find all pages that refer to the old page frame, if the page frame is shared; and set each of the corresponding page table entries to invalid
- Invalidate TLB entries
- Write changes on page back to disk, if it has been modified

Some slides are courtesy of Dr. Thomas Anderson

How do we know if a page has been modified/accessed?

- Every page table entry has some status bits
  - Dirty bit: has page been modified?
    - Set by hardware on store instruction
  - Accessed bit: has page been recently used?
    - Set by hardware on reference
- Status bits can be reset by the OS kernel
  - When changes to page are flushed to disk
  - When enforcing the clock algorithm (to be discussed)

Cache Replacement Policy

- On a cache miss and all the cache entries have been used, how do we choose which entry to replace?

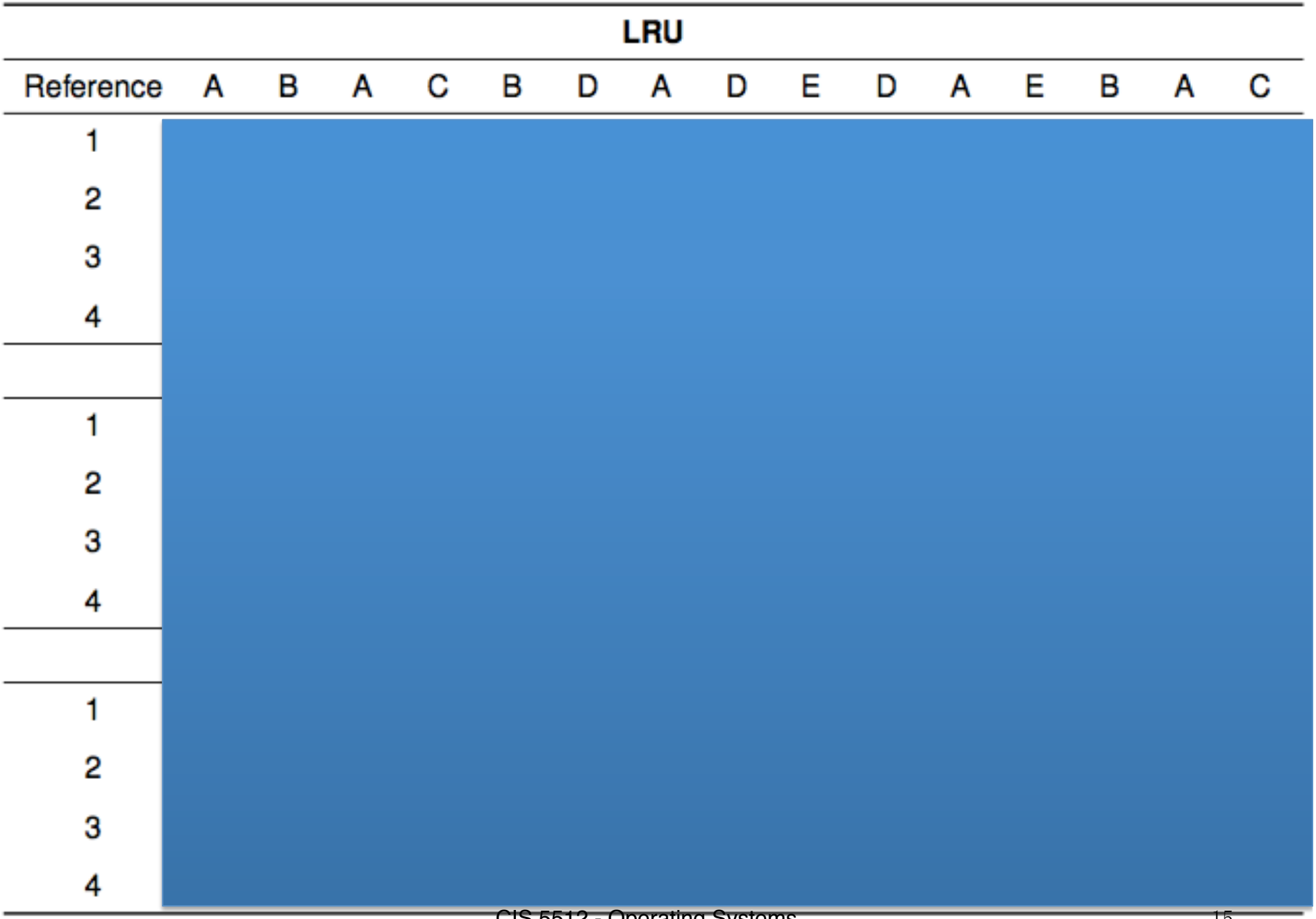
- Policy goal: reduce cache misses

A Simple Policy

- Random?
  - Replace a random entry
- FIFO?
  - Replace the entry that has been in the cache the longest time
  - What could go wrong?

MIN, LRU, LFU

- MIN
  - Replace the cache entry that will not be used for the longest time into the future
  - Optimality proof based on exchange: if evict an entry used sooner, that will trigger an earlier cache miss
- Least Recently Used (LRU)
  - Replace the cache entry that has not been used for the longest time in the past
  - Approximation of MIN
- Least Frequently Used (LFU)
  - Replace the cache entry used the least often (in the recent past)


LRU
Reference
A B A C B D A D E D A E B A C
1
2
3
4
1
2
3
4
1
2
3
4

LRU

Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+				+			+	
2		B			+								+		
3				C					E			+			
4						D		+		+					C

FIFO

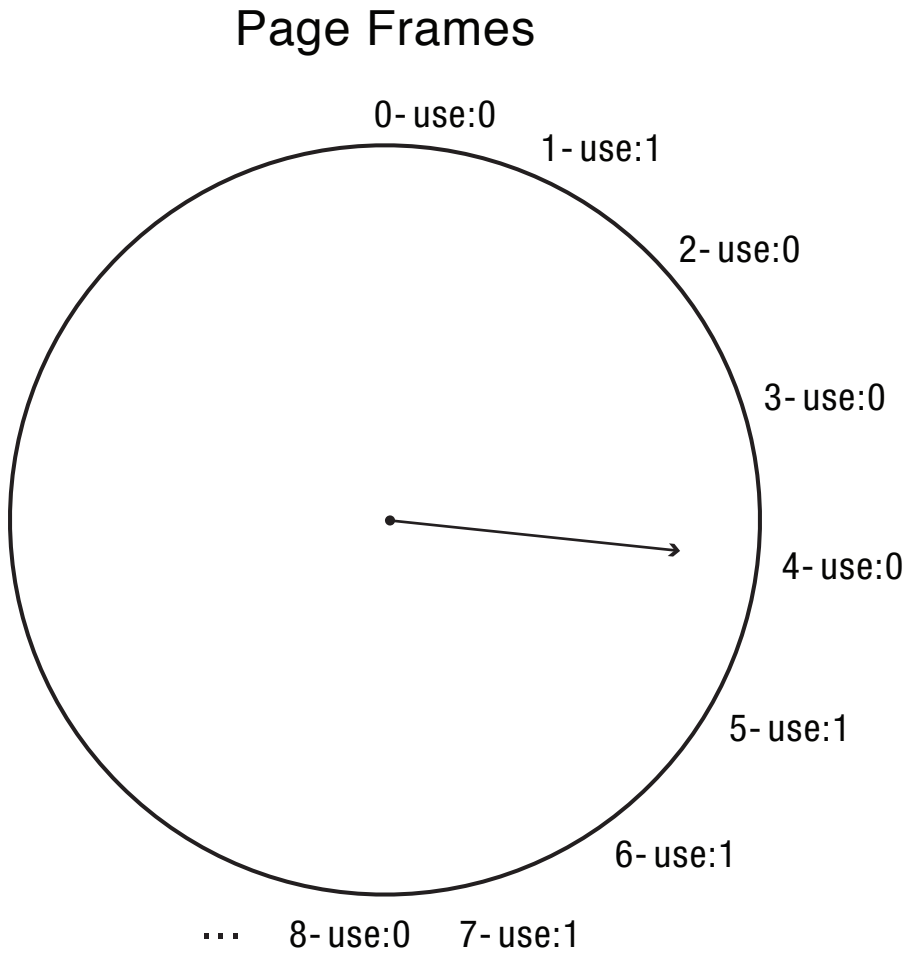
1	A		+				+		E						
2		B			+						A			+	
3				C								+	B		
4						D		+		+					C

MIN

1	A		+				+				+			+	
2		B			+								+		C
3				C					E			+			
4						D		+		+					

Clock Algorithm: Estimating LRU

- Periodically, sweep through all pages
- If page is unused, it is reclaimed
- If page is used, mark as unused
- The name of the algorithm is because, logically, all the pages form a circle during sweeps



Nth Chance: Not Recently Used

- Keep an integer for each page
  - notInUseSince: number of sweeps since last use
- Periodically sweep through all page frames

```
if (page is used) {
    notInUseSince = 0;
} else if (notInUseSince < N) {
    notInUseSince++;
} else {
    reclaim page;
}
```

Recap

- MIN is optimal
  - replace the page or cache entry that will be used farthest into the future
- LRU is an approximation of MIN
  - For programs that exhibit spatial and temporal locality
- Clock/Nth Chance is an approximation of LRU
  - Bin pages into sets of “not recently used”

Review

- How to allocate physical memory?
  - Contiguous allocations
    - Fixed
    - Dynamic
  - Non-contiguous allocations
    - Segmentation
    - Paging
      - Each process has a huge uniform virtual address space
      - Internal fragmentation
      - External fragmentation

Review

- How to do address translation?
  - Page tables: each process has its own page table
  - Why multi-level?
  - How to speed up?
    - TLB

Review

- How to be lazy?
  - Copy-on-write
  - Demand paging
  - Memory-mapped file
- Others
  - Memory hierarchy
  - Locality
  - Cache replacement: LRU, Clock algorithm

Thrashing

- Thrashing: when system has too small a cache, most of the time is spent on evicting cache entries and copying data to cache
- Resident Set: the pages of a process that are in memory (rather than in disk)

Question

- What happens to system performance as we increase the number of active processes?
  - If the sum of the working sets > physical memory?
  - In this case thrashing will occur, and the the system performance will degrade dramatically