Today

- Roadmap:
  - TLBs are a cache of translations
  - Memory is really a cache of pages … and where there is a cache, there is a … replacement policy
- Learning Outcomes
  - Identify similarities between *file caching* and *memory management*.
  - Explain how the clock algorithm relates to LRU.
  - Explain why the OS cannot efficiently implement policies such as LRU, LFU, etc.
- Reading:
  - 9.7 for material presented in class

Memory can fill up with pages…

- On a busy system, it’s quite possible that all of memory will contain valuable pages.
- When that happens, what do we do when we need yet another page?
- We;ve got this! This is a page replacement policy, just like we learned about in our caching unit:
  - LRU
  - LFU
  - MRU
- Or not? Why might those not be practical?

Memory Eviction Policies: Practicality

- Why might our conventional replacement algorithms not be practical?
  - The OS does not *see* every access to a page.
  - Most accesses are handled by the MMU, so we don't have time to update counts (LFU) or recently (LRU/MRU).
  - What is an OS to do?

What does the hardware provide?

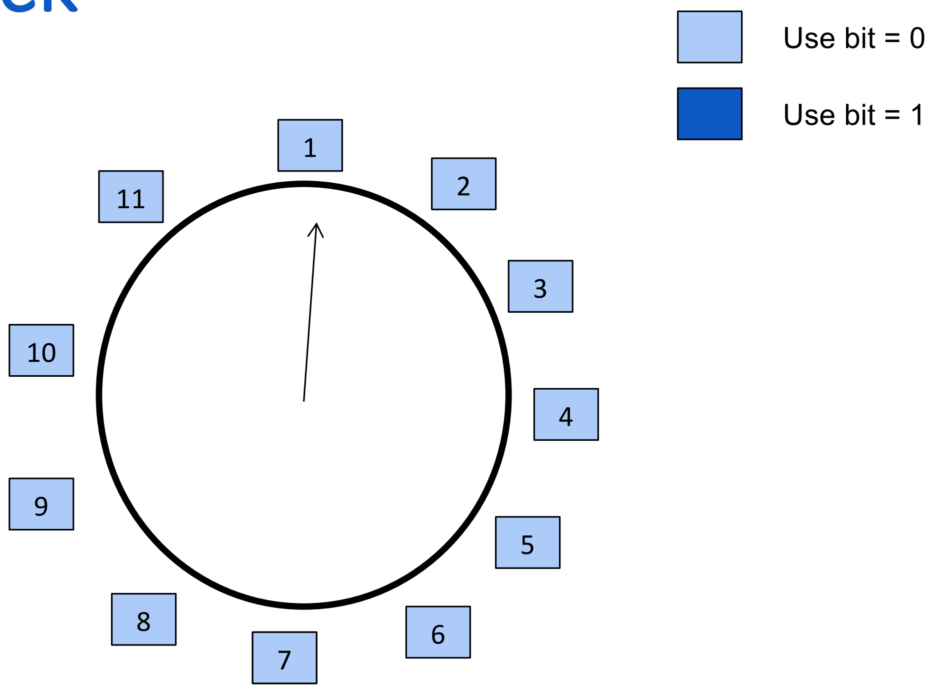
- Precisely two things:
  - Access bits per PTE (sometimes called use bits)
  - Dirty bits per PTE
- Ideally an algorithm will:
  - Do something that approximates LRU (after all LRU is really just an approximation for Belady's algorithm)
  - Take advantage of the information the HW does provide
  - Make sure there are lots of clean pages available, so we don't get stuck always doing a write before a read.

Intuition from LRU

- LRU is trying to select the page that hasn't been used for the longest time.
- Since we have a lot of main memory pages, maybe it's OK to pick a page that hasn't been used for a long time.
- How can we take advantage of use bits to determine pages that have not been used in a long time?

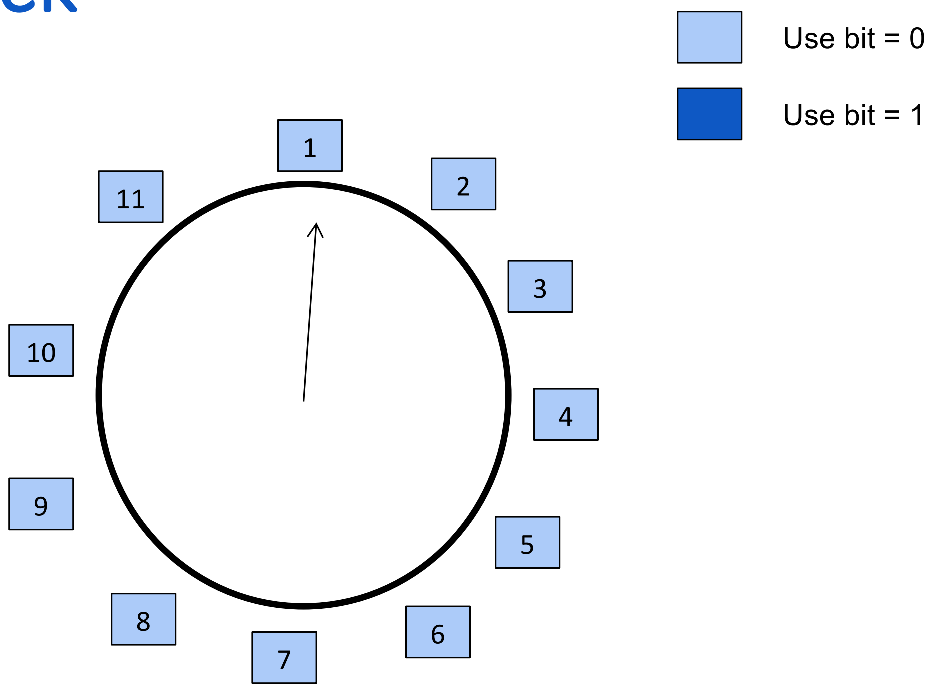
Introducing Clock

1. Imagine that all your physical pages are arranged around a clock.
2. Right now, we have virtual pages 1-11 in memory and their use bits are all 0.
3. *We have a clock hand that suggests the next page for eviction.*
4. Each time a page is accessed, the HW will set its use bit to be 1.
5. When we need to evict a page, we look at the page under the hand:
   a) If its use bit = 1, clear it and move the hand, repeat step 5
   b) If its use bit = 0, evict it.



Introducing Clock

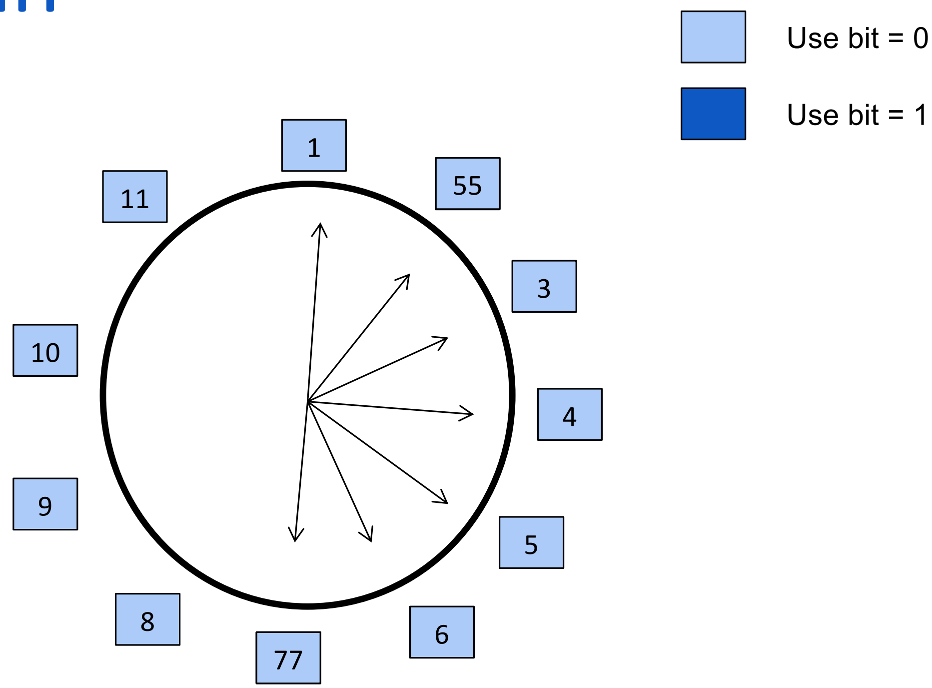
1. Imagine that all your physical pages are arranged around a clock.
2. Right now, we have virtual pages 1-11 in memory and their use bits are all 0.
3. *We have a clock hand that suggests the next page for eviction.*
4. Each time a page is accessed, the HW will set its use bit to be 1.
5. When we need to evict a page, we look at the page under the hand:
   a) If its use bit = 1, clear it and move the hand, repeat step 5
   b) If its use bit = 0, evict it.



Page reference stream

Watch Clock Run

1. Imagine that all your physical pages are arranged around a clock.
2. Each time a page is accessed, the HW will set its use bit to be 1.
3. Right now, we have virtual pages 1-11 in memory and their use bits are all 0.



Page reference stream 1 3 1 1 6 55 4 5 77

Clock and Beyond

- Why clock and not LRU?
  - The vast majority of accesses are handled in hardware and are invisible to the operating system (which is responsible for replacement).
- The OS may want to overlay policies on top of pure clock:
  - Should your processes get to evict my pages?
  - Should I let a single process consume all of memory?
  - What is 'fair' page allocation?