# Cache Replacement Policy

- On a cache miss, how do we choose which entry to replace?
  - Assuming the new entry is more likely to be used in the near future
  - In direct mapped caches, not an issue!

- Policy goal: reduce cache misses
  - Improve expected case performance
  - Also: reduce likelihood of very poor performance

# 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/MIN for Sequential Scan

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | **LRU** | | | | | | | | |
| **Reference** | **A** | **B** | **C** | **D** | **E** | **A** | **B** | **C** | **D** | **E** | **A** | **B** | **C** | **D** | **E** |
| **1** | **A** | | | | **E** | | | | **D** | | | | **C** | | |
| **2** | | **B** | | | | **A** | | | | **E** | | | | **D** | |
| **3** | | | **C** | | | | **B** | | | | **A** | | | | **E** |
| **4** | | | | **D** | | | | **C** | | | | **B** | | | |
| | | | | | | | **MIN** | | | | | | | | |
| **1** | **A** | | | | | **+** | | | | | **+** | | | **+** | |
| **2** | | **B** | | | | | **+** | | | | | **+** | **C** | | |
| **3** | | | **C** | | | | | **+** | **D** | | | | | **+** | |
| **4** | | | | **D** | **E** | | | | | **+** | | | | | **+** |

**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 | | + | | + | | | | | |

# Belady's Anomaly

| FIFO (3 slots) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reference | A | B | C | D | A | B | E | A | B | C | D | E |
| 1 | A | | | D | | | E | | | | | + |
| 2 | | B | | | A | | | + | | C | | |
| 3 | | | C | | | B | | | + | | D | |

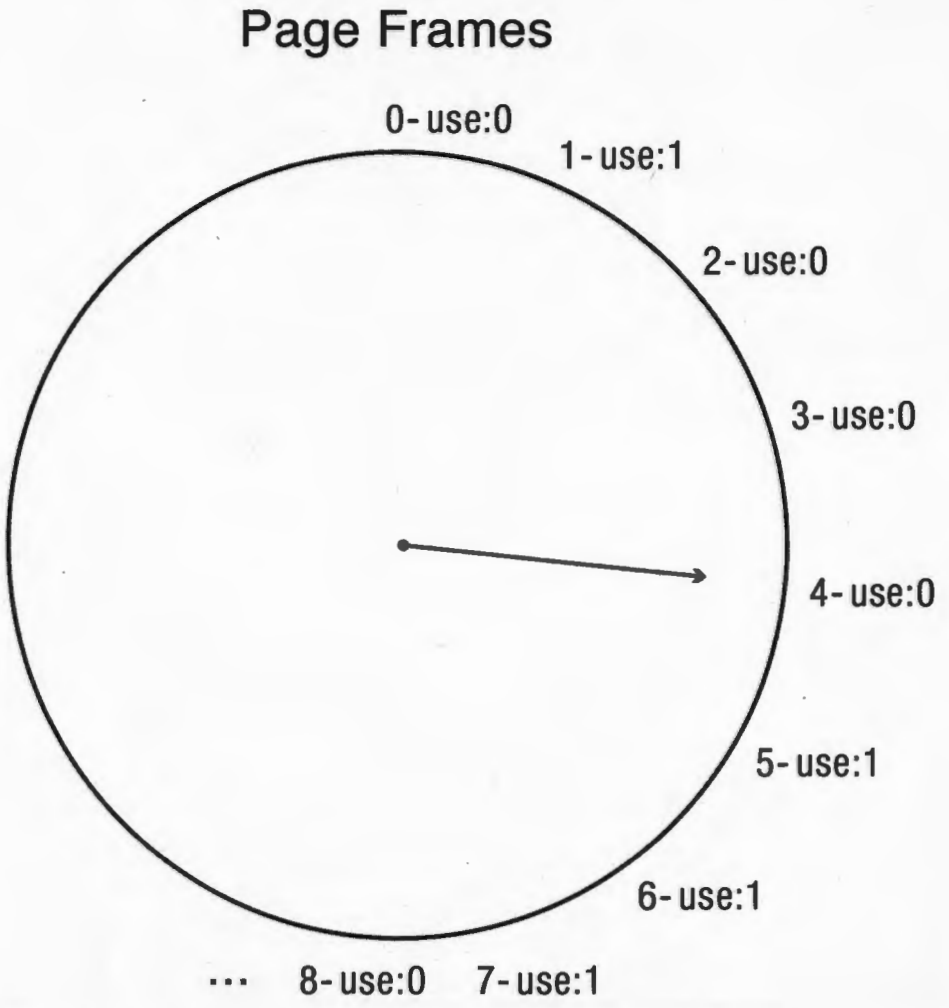
| FIFO (4 slots) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | | | | + | | E | | | | D | |
| 2 | | B | | | | + | | A | | | | E |
| 3 | | | C | | | | | | B | | | |
| 4 | | | | D | | | | | | C | | |

# Question

- How accurately do we need to track the least recently/least frequently used page?
  - If miss cost is low, any approximation will do
    - Hardware caches
  - If miss cost is high but number of pages is large, any not recently used page will do
    - Main memory paging with small pages
  - If miss cost is high and number of pages is small, need to be precise
    - Main memory paging with superpages

# Clock Algorithm: Estimating LRU

- Hardware sets use bit
- Periodically, OS sweeps through all pages
- If page is unused, reclaim
- If page is used, mark as unused




Recall: page frames in physical memory
use bits in page table (virtually indexed)

core map:
list of virtual pages
for each page fram

# Nth Chance: Not Recently Used

- Instead of one bit per page, keep an integer
  - notInUseSince: number of sweeps since last use
- Periodically sweep through all page frames

```
if (page is used) {
    notInUseForXSweeps = 0;
} else if (notInUseForXSweeps < N) {
    notInUseForXSweeps++;
} else {
    reclaim page; write modifications if needed
}
```

# Implementation Note

- Clock and Nth Chance can run synchronously
  - In page fault handler, run algorithm to find next page to evict
  - Might require writing changes back to disk first
- Or asynchronously
  - Create a thread to maintain a pool of recently unused, clean pages
  - Find recently unused dirty pages, write mods back to disk
  - Find recently unused clean pages, mark as invalid and move to pool
  - On page fault, check if requested page is in pool!
  - If not, evict page from the pool

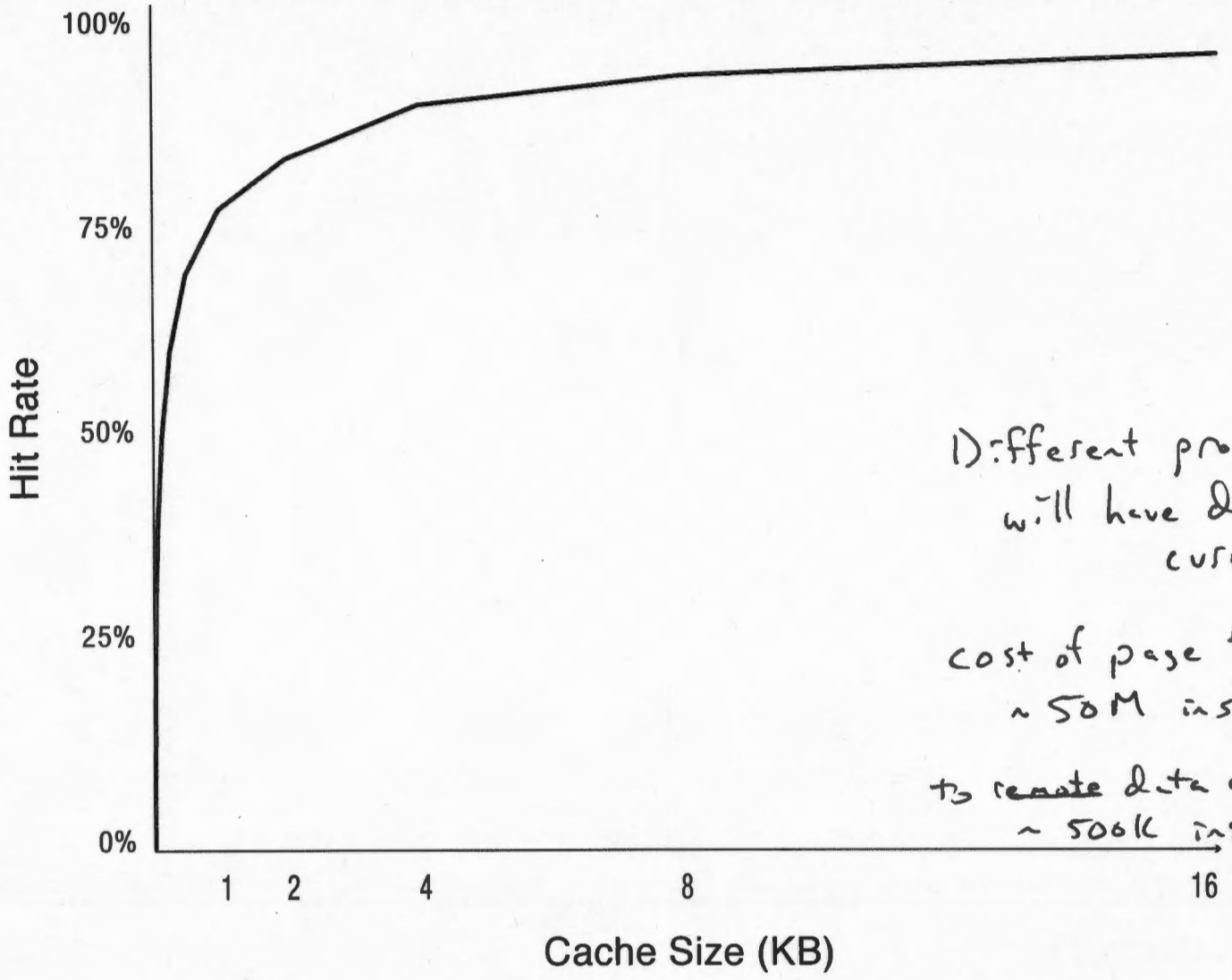
# Recap

- 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"

# Working Set Model

- Working Set: set of memory locations that need to be cached for reasonable cache hit rate
- Thrashing: when system has too small a cache

# Cache Working Set

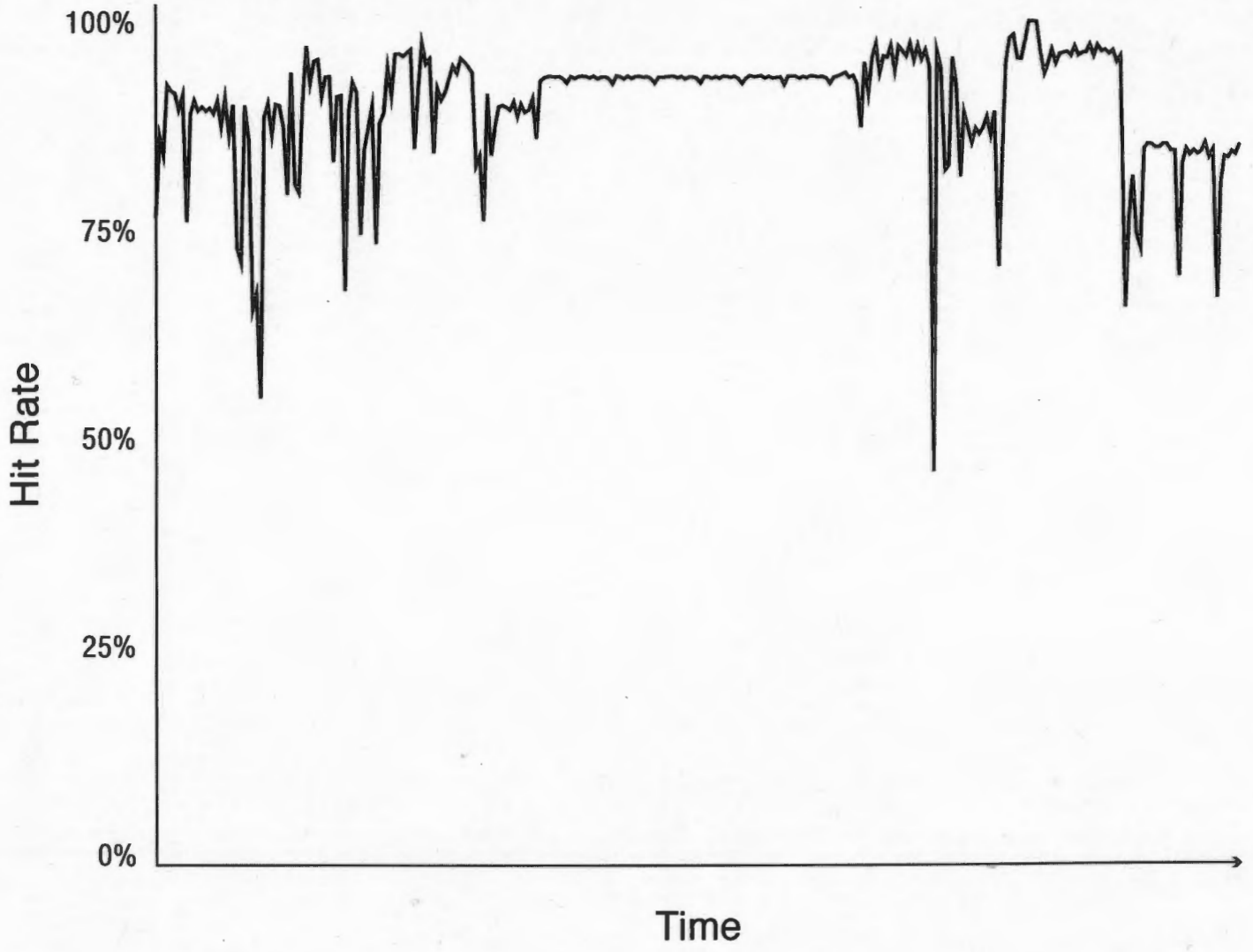



Different programs will have different curves

cost of page fault to disk ~ 50M instructions

to remote data center memory ~ 500K instructions

# Phase Change Behavior



# Question

- What happens to system performance as we increase the number of processes?
  - If the sum of the working sets > physical memory?