Lecture 15: memory management continued

- clock algorithm
- thrashing
- (time permitting) Disk

Rand, FIFO fast, easy, dumb

LRU, LFU require storage (timestamp / count per page)

estimating whether something will be needed soon.

OPT $\underbrace{\text{estimating whether}}_{\text{perfect}}$

---

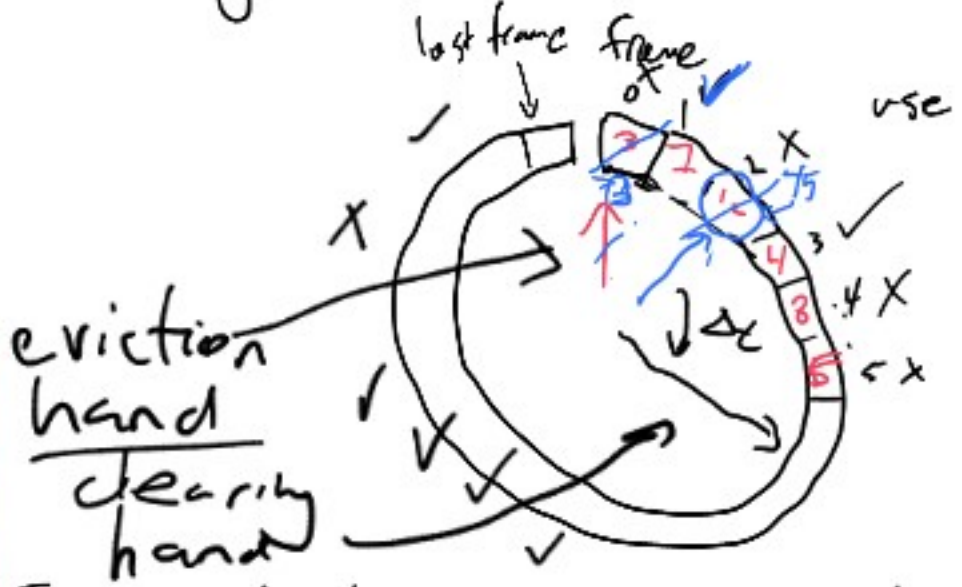
requires HW support : do something on every access.
(TLB)

instead, add a "use" bit:

- set when page accessed
- cleared by OS.

(dumb counter)

Clock algorithm:



use bit: ✓ (set, page accessed)

: X (cleared, not accessed).

access

12, 73, 45

(hit) (evict frame 0) (evict frame 2)

To evict: take next page, skipping over pages marked "in use"

- bits are set when page used (automatic TLB, copied into PT)
- clear bits "a little while" before we read them
  - clearing hand, set a fixed # of frames in the future, clear use bit when it passes a frame.

Note: Δt is a distance (# frames) not a time

---

simple, decent locality.

effect:

page doesn't get evicted if it is used between when it is cleared & when eviction hand arrives.

$\boxed{(\Delta t : \text{distance})}$

Δt is large: in limit, all pages (or most) will be marked "in use" : evicting takes a long time.

Δt is tiny (e.g. 1): use bits always clear: FIFO

Thrashing:

- page replacement / swapping work ok if don't swap often.
- lots of swapping: terrible
- too much contention for same frames: spend all time swapping, no useful progress: thrashing.
- thrashing caused by launching more processes than you have memory for.
- need to know how much memory a process is using (not allocated)
    - working set: set of pages that process has accessed recently (within some Δt)
- Compute total size of working sets of all processes, if bigger than # frames of physical memory: thrashing.
- if thrashing is detected: suspend a process until total WSS (working set size) goes down.
    - Which process? similar to CPU scheduling.

Disk

Very different from memory:

- Big
  memory: GB
  disk: TB, PB
  1000 - 1,000,000 x bigger.

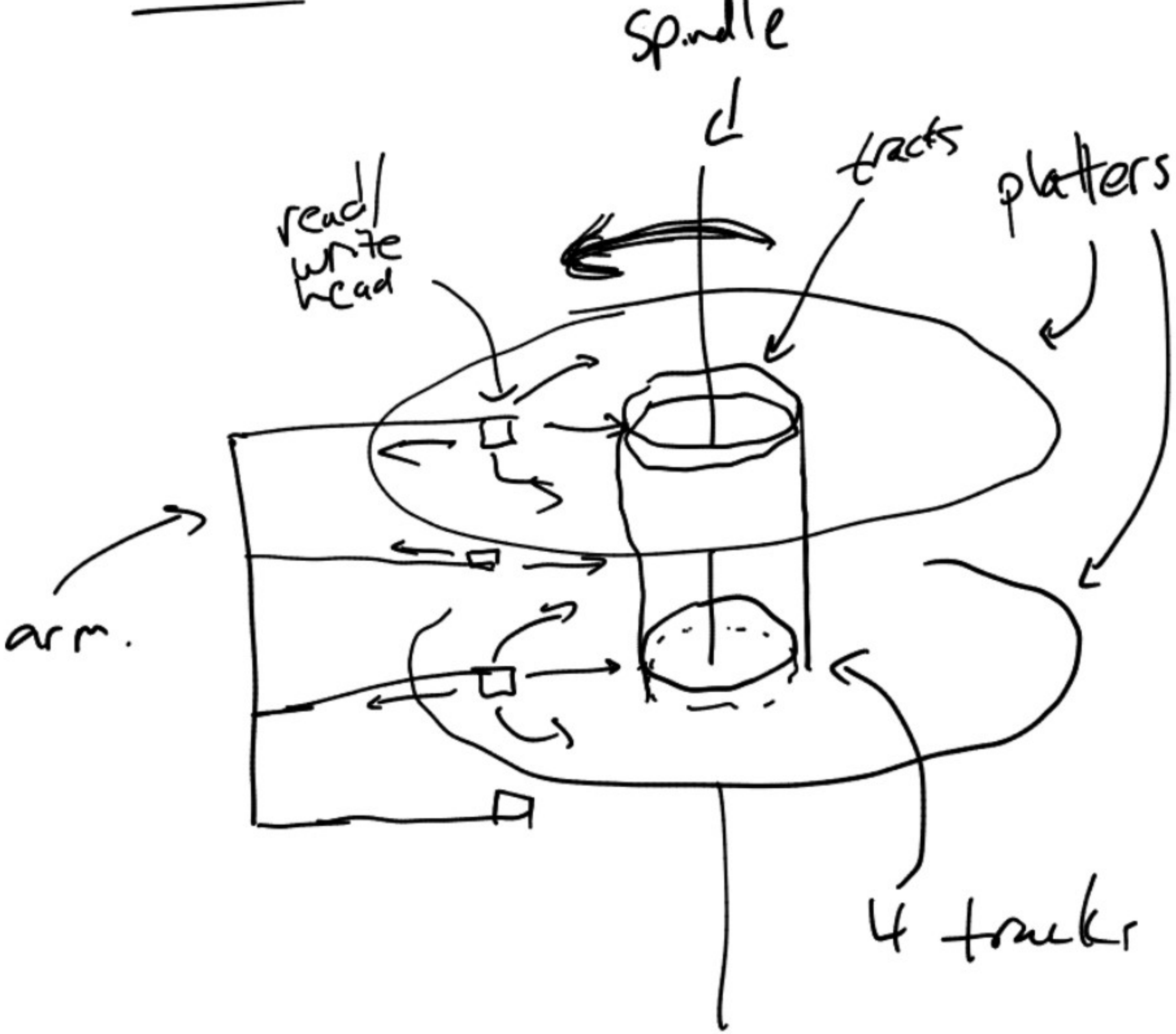
- Slow
  memory: ns
  disk: ms
  1000 - 1,000,000 x slower

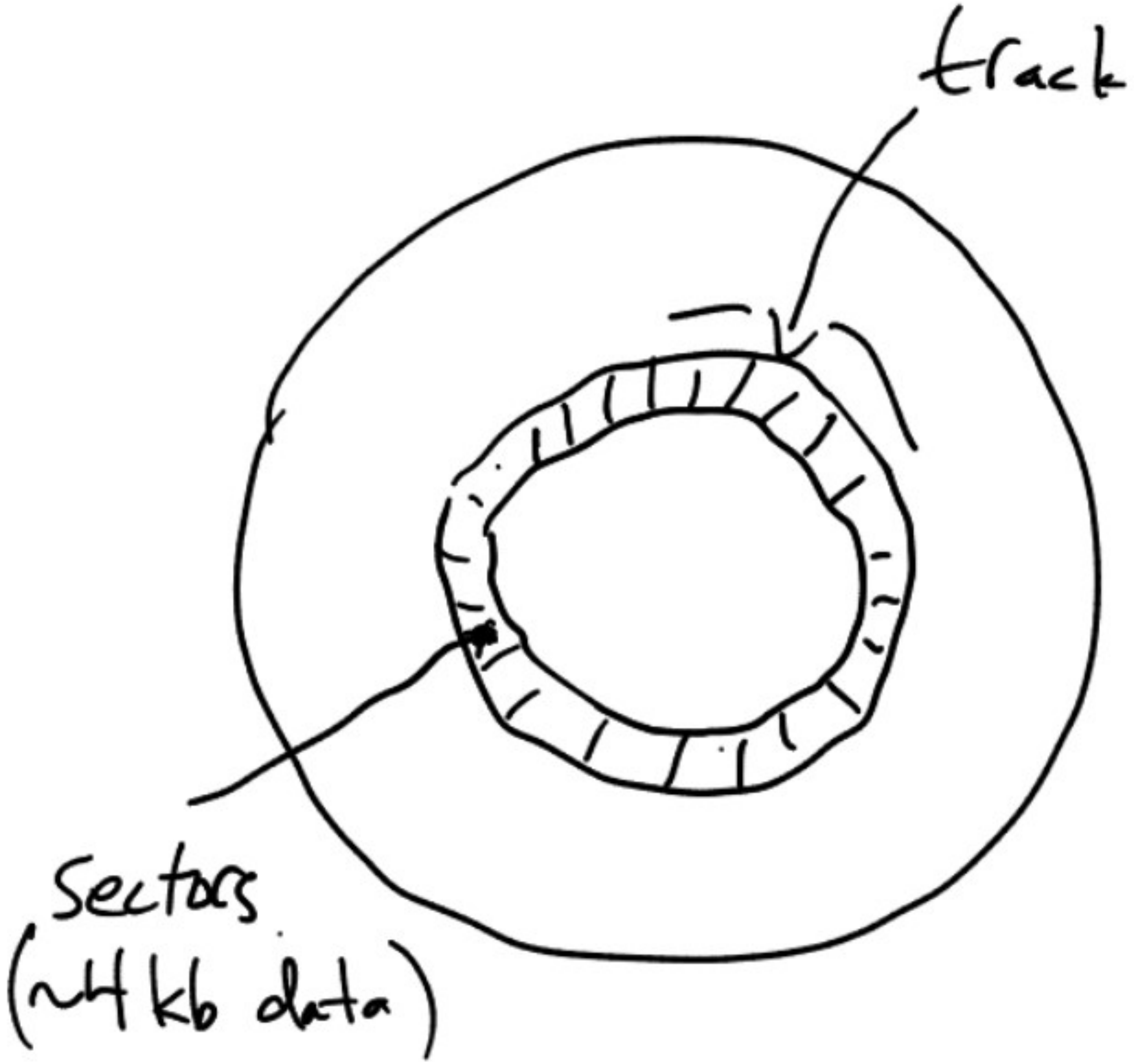
- Persistent: memory is volatile: gone when you end program

  disk is persistent: think about what to do when suddenly shut down.

- Sequential access is much faster than random access.

Disk (magnetic)

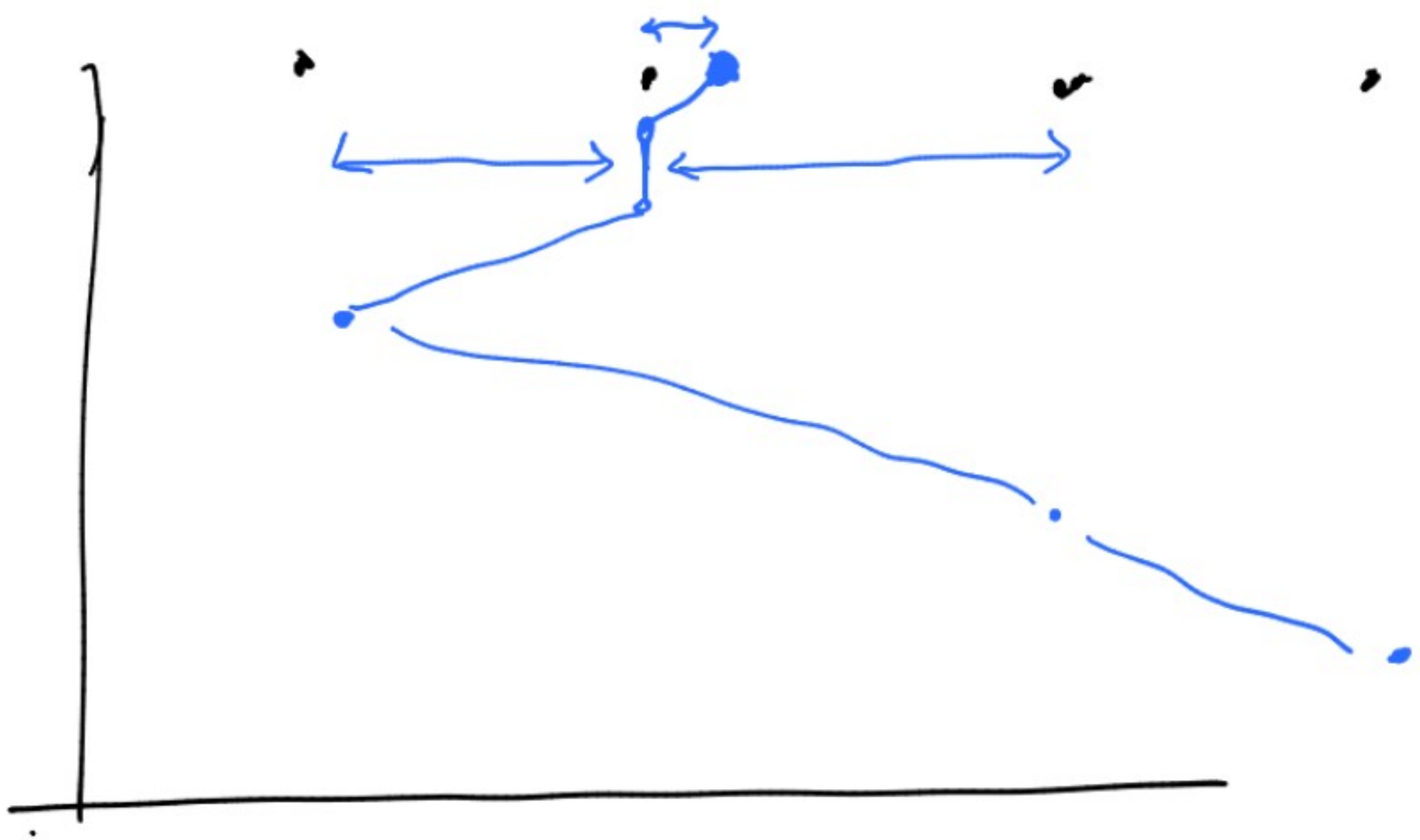




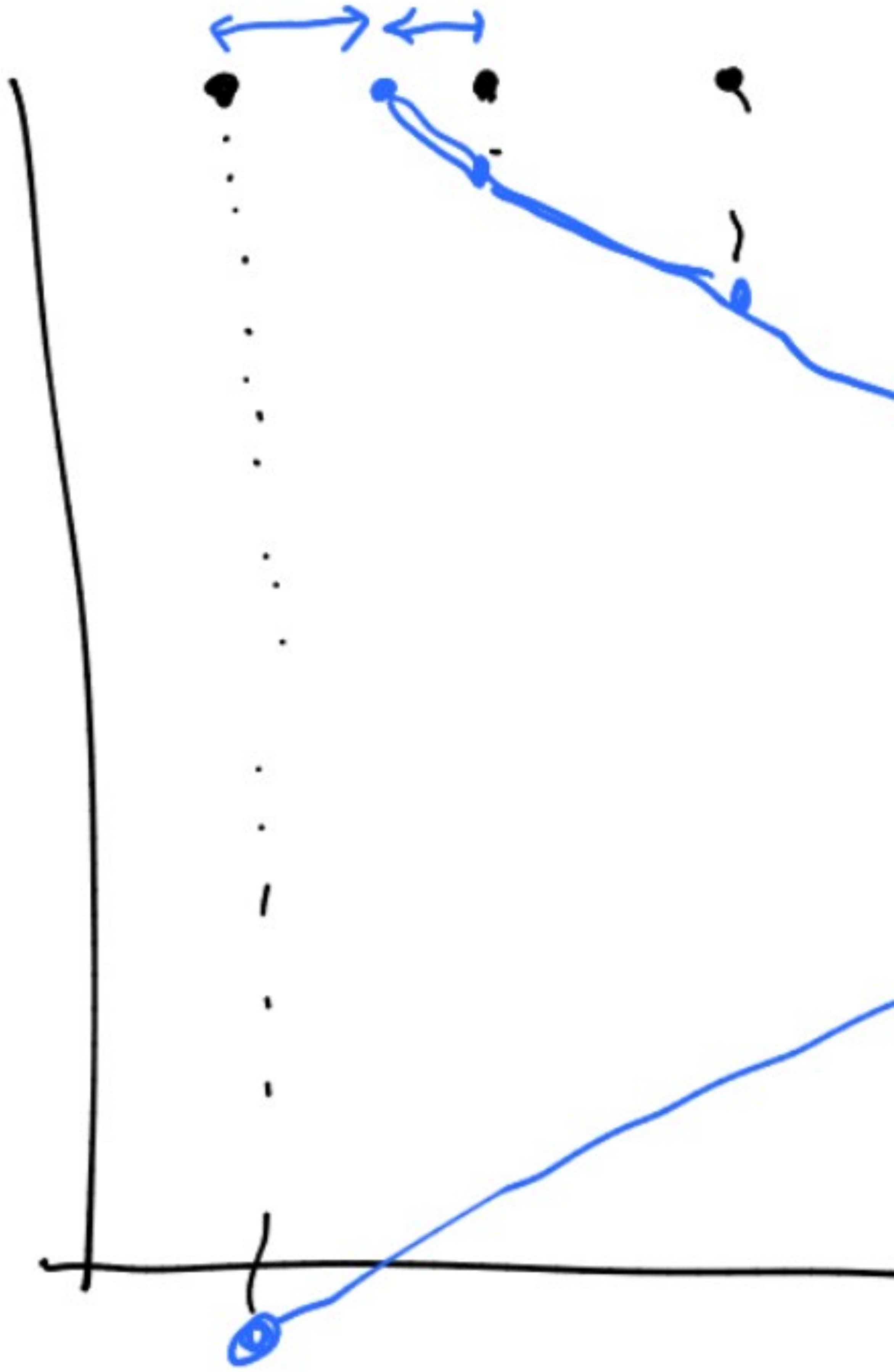
OOPS! SSTF not actually optimal

~~Optimal algorithm~~: Shortest seek time first

- at any point, consider seek time to all requests, service the closest one.

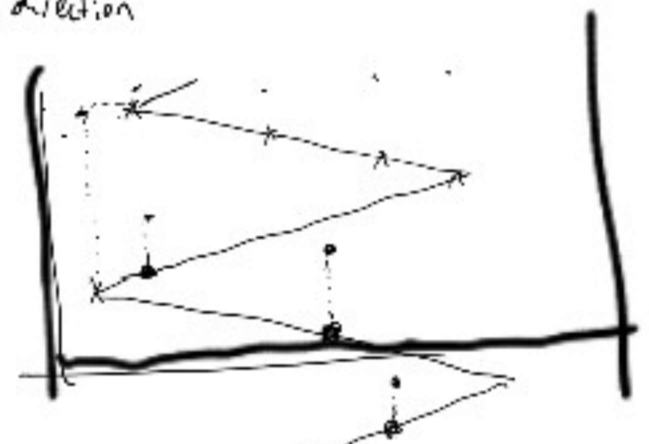


SSTF can be unfair



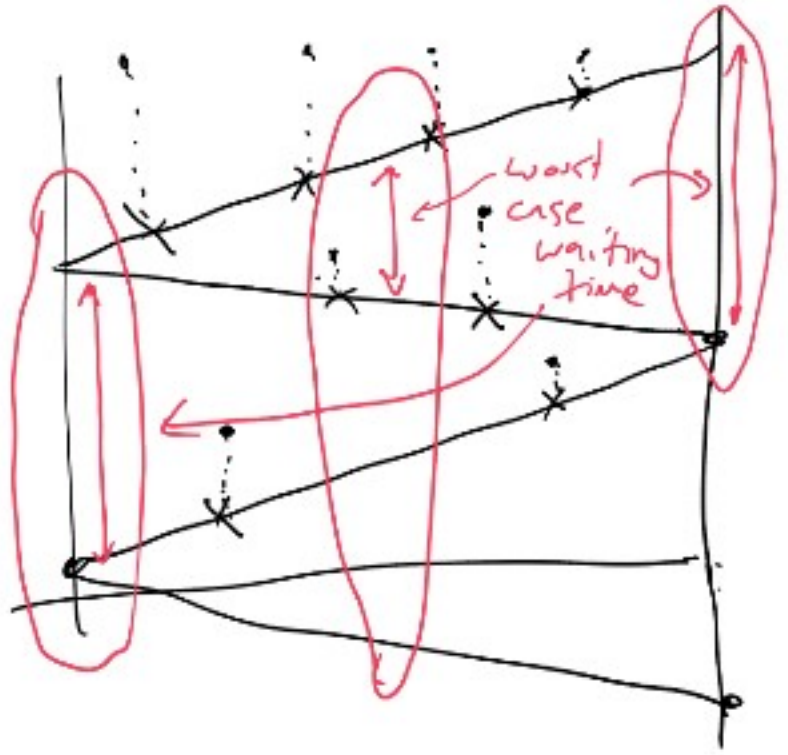
Fairer than SSTF: (LOOK)

<u>Elevator algorithm</u>: pick a direction, service all requests in that direction, turn around service all requests in other direction



Still unfair to locations right near end: miss the elevator.

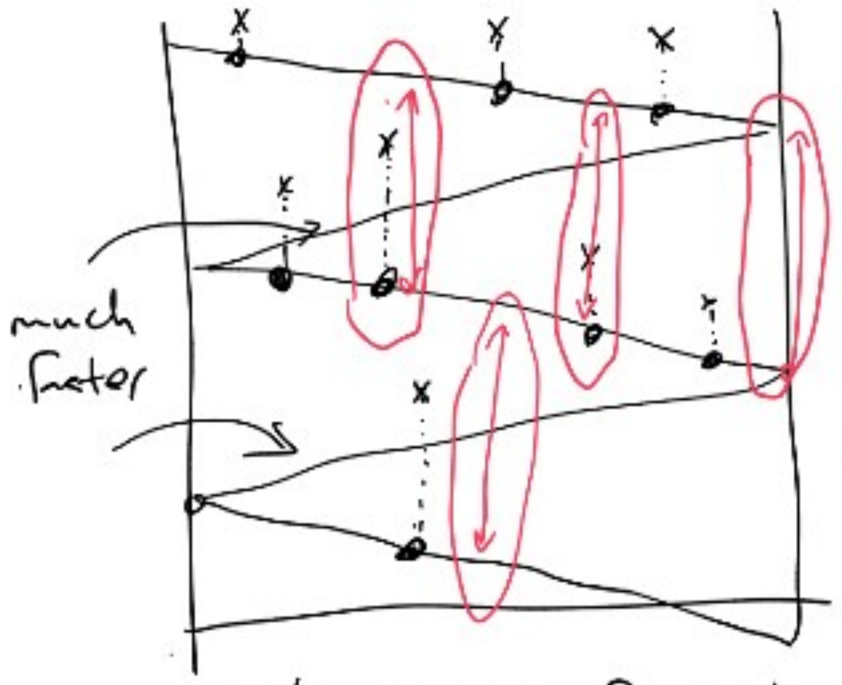
SCAN algorithm: (trades efficiency for fairness)



worst-case at end is 2x as big as worst case in middle.

C-SCAN algorithm

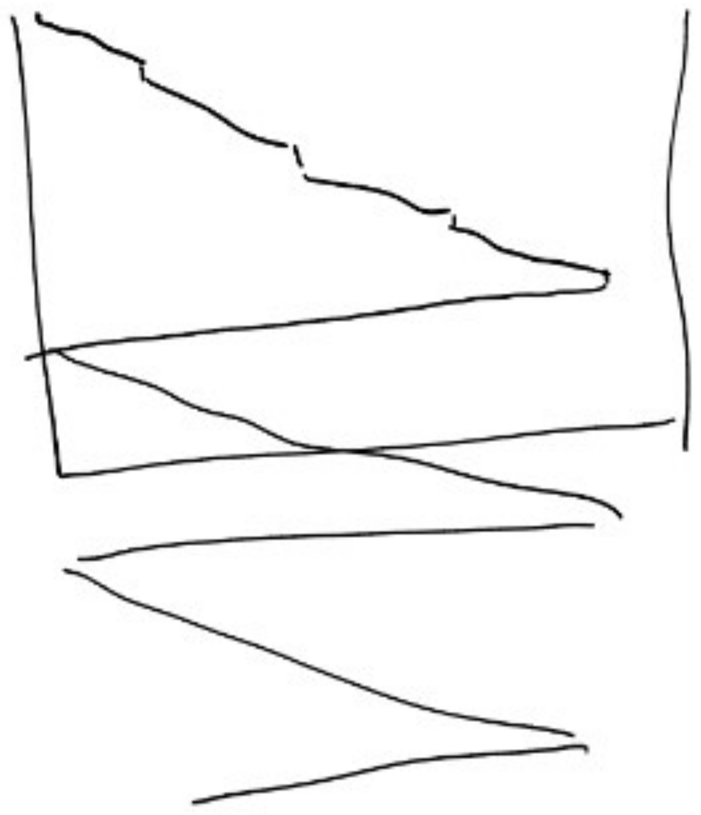
→ circular



only service requests in one direction

worst-case wait is same for all locations.

in practice, seek from one end to other quickly if don't stop



C-LOOK

like elevator algorithm, only go as far as next request instead of to end of disk.