Finding an anomaly in the CLOCK algorithm with Harmony

```
def CLOCK(n):
    result = { .entries: [None,] * n, .recent: {}, .hand: 0, .misses: 0 }

def ref(ck, x):
    if x not in ck→entries:
        while ck→entries[ck→hand] in ck→recent:
            ck→recent −= {ck→entries[ck→hand]}
            ck→hand = (ck→hand + 1) % len(ck→entries)
        ck→entries[ck→hand] = x
        ck→hand = (ck→hand + 1) % len(ck→entries)
        ck→misses += 1
    ck→recent |= {x}

clock3, clock4, refs = CLOCK(3), CLOCK(4), [ ]

for i in {1..10}:
    let x = choose({ 1..5 }):
        refs += [x,]
        ref(?clock3, x); ref(?clock4, x)
        assert(clock4.misses <= clock3.misses)
```

CLOCK algorithm

find an anomaly

Harmony output

```
#states 746532
Safety Violation
T0: __init__() [0,1,125-132,2-19,133-137,2-19,138-157(choose 5),158-174,21-43,79-124,175-184,
21-43,79-124,185-201,152-157(choose 4),158-174,21-43,79-124,175-184,21-43,79-124,185-201,152-
157(choose 3),158-174,21-43,79-124,175-184,21-43,79-124,185-201,152-157(choose 2),158-174,21-
78,29-78,29-78,29-43,79-124,175-184,21-43,79-124,185-201,152-157(choose 4),158-174,21-28,114-
124,175-184,21-28,114-124,185-201,152-157(choose 5),158-174,21-78,29-43,79-124,175-184,21-28,
114-124,185-201,152-157(choose 4),158-174,21-28,114-124,175-184,21-28,114-124,185-201,152-157
(choose 1),158-174,21-78,29-78,29-78,29-43,79-124,175-184,21-78,29-78,29-78,29-78,29-43,79-12
4,185-201,152-157(choose 5),158-174,21-28,114-124,175-184,21-43,79-124,185-201,152-157(choose
 4),158-174,21-28,114-124,175-184,21-43,79-124,185-197] { clock3: { .entries: [ 1, 4, 5 ], .h
and: 1, .misses: 6, .recent: { 1, 4, 5 } }, clock4: { .entries: [ 1, 5, 4, 2 ], .hand: 3, .mi
sses: 7, .recent: { 1, 4, 5 } }, refs: [ 5, 4, 3, 2, 4, 5, 4, 1, 5, 4 ] }
Harmony assertion failed
```

Reference string

Presenting… The Belady CLOCK Anomaly

3 frames
6 misses

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

4 frames
7 misses

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

red = miss

* is clockhand

(green) is recent bit

Presenting… The Belady CLOCK Anomaly

3 frames, 6 misses

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

4 frames, 7 misses

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

stack property first violated

red = miss

* is clockhand

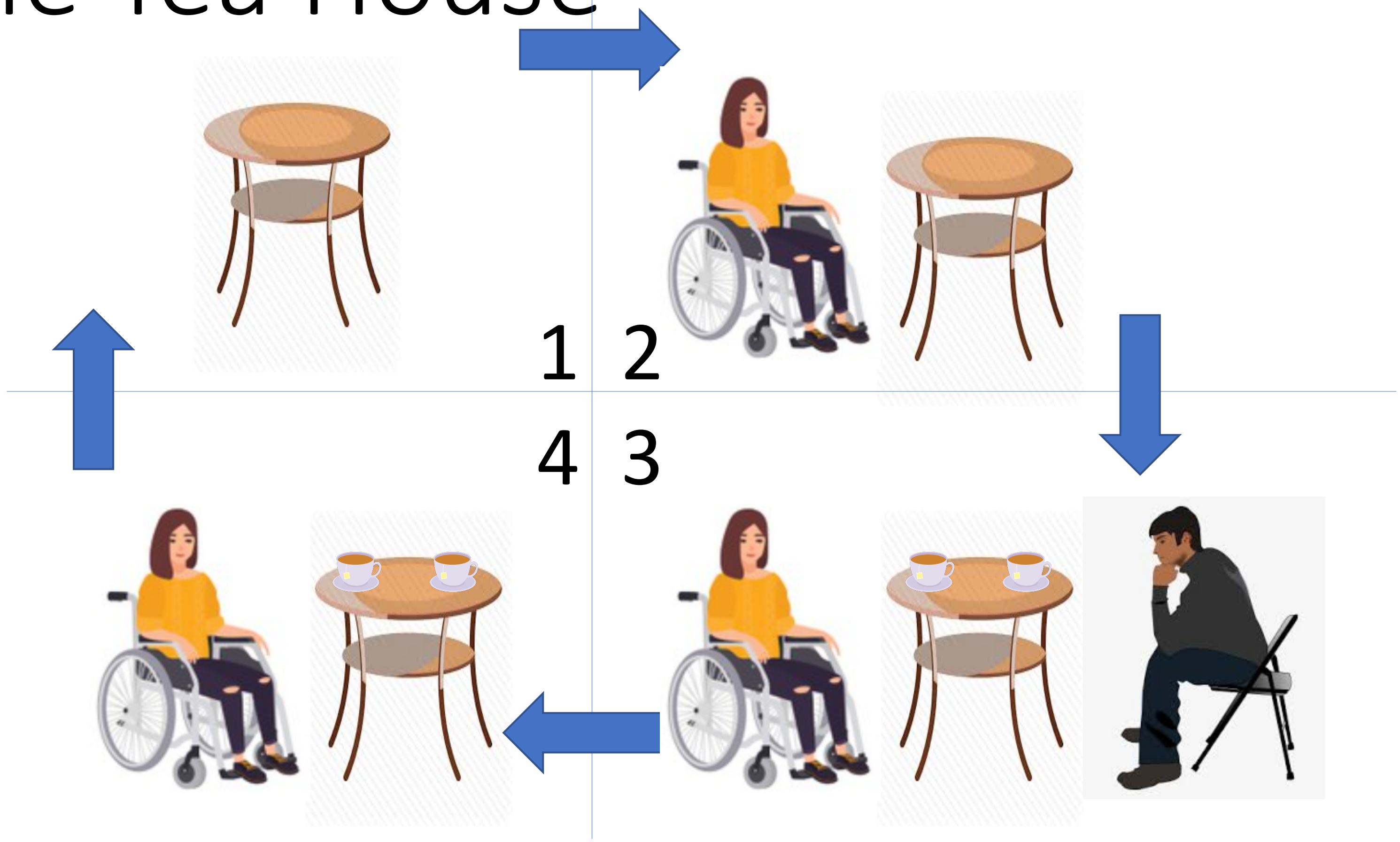
is recent bit

The Little Tea House

- The table can be in one of four states
  1. no one sitting at the table
  2. one person sat down, but is not yet allowed to drink while waiting for the second person
  3. two persons are sitting down, both allowed to drink
  4. one person has left after drinking

- State 2 and 4 both have one person sitting at the table, but they are very different states nonetheless

The Little Tea House

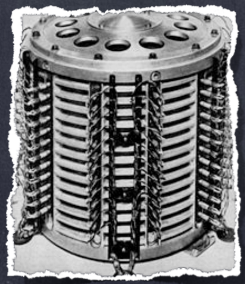


Persistent Storage

Storage Devices

- We focus on two types of persistent storage
  - magnetic disks
    - servers, workstations, laptops
  - flash memory
    - smart phones, tablets, cameras, laptops
- Other exist(ed)
  - tapes

  - drums



  - clay tablets

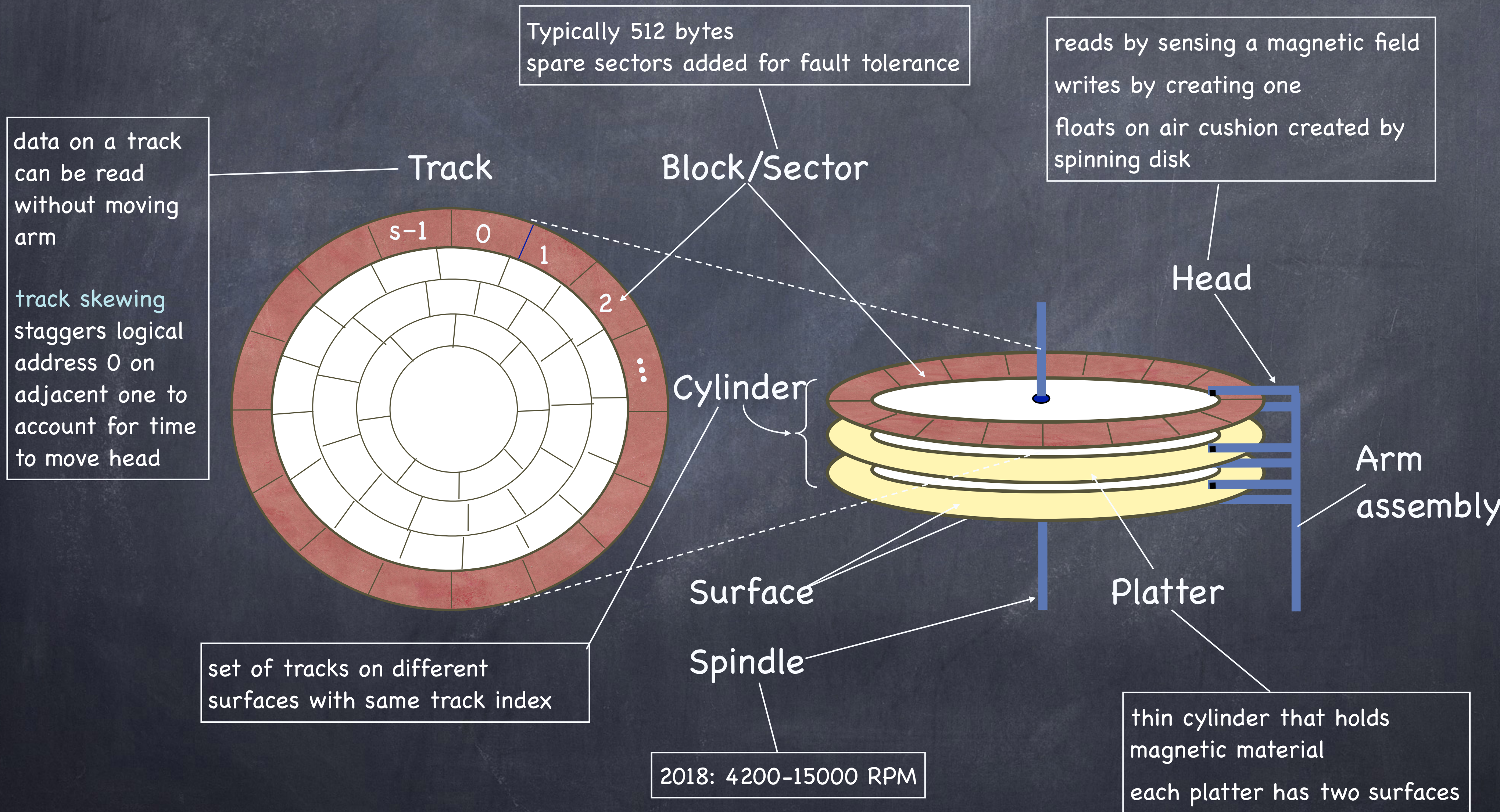
The Oldest Library?

- Ashurbanipal, King of Assyria (668-630 bc)

Magnetic disk

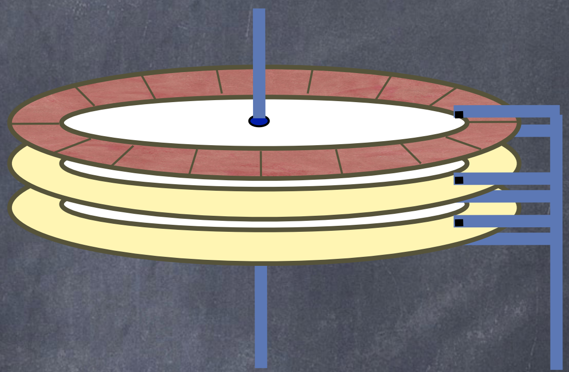
- Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum

Disk Drive Schematic



Disk Read/Write

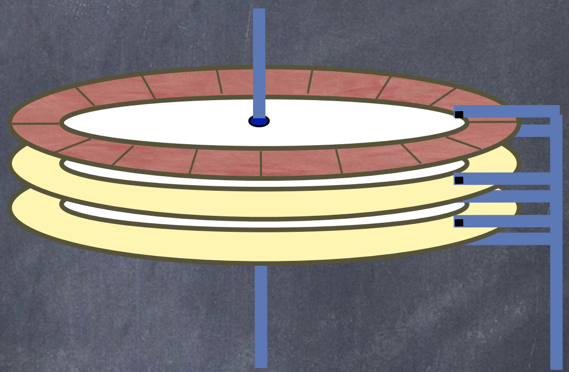
- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek
  - settle
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time



Disk access time:

Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek (and though shalt approximately find)
  - settle (fine adustments)
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time

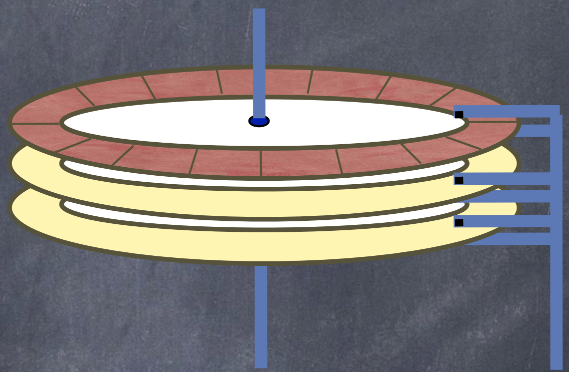


Disk access time:

seek time +

Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek (and though shalt approximately find)
  - settle (fine adustments)
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time



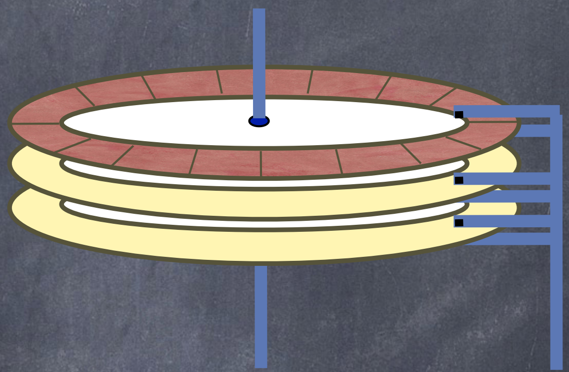
Disk access time:

seek time +

rotation time +

Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek (and though shalt approximately find)
  - settle (fine adustments)
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time



Disk access time:

seek time +

rotation time +

transfer time

Seek time: A closer look

- Minimum: time to go from one track to the next
  - 0.3-1.5 ms
- Maximum: time to go from innermost to outermost track
  - more than 10ms; up to over 20ms
- Average: average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across disk

How did we get that?

- To compute average seek time, add distance between every possible pair of tracks and divide by total number of pairs
  - assuming N tracks, N^2 pairs, and sum of distances is $\sum_{x=0}^{N}\sum_{y=0}^{N}|x-y|$ which we compute as $\int_{x=0}^{N}\int_{y=0}^{N}|x-y|dy\,dx$

How did we get that?

- To compute average seek time, add distance between every possible pair of tracks and divide by total number of pairs
  - assuming N tracks, N^2 pairs, and sum of distances is $\sum_{x=0}^{N}\sum_{y=0}^{N}|x-y|$ which we compute as $\int_{x=0}^{N}\int_{y=0}^{N}|x-y|dy\,dx$
  - The inner integral expands to $\int_{y=0}^{x}(x-y)dy + \int_{y=x}^{N}(y-x)dy$

    which evaluates to $x^2/2 + (N^2/2 - xN + x^2/2)$

How did we get that?

- To compute average seek time, add distance between every possible pair of tracks and divide by total number of pairs
  - assuming N tracks, N^2 pairs, and sum of distances is $\sum_{x=0}^{N}\sum_{y=0}^{N}|x-y|$ which we compute as $\int_{x=0}^{N}\int_{y=0}^{N}|x-y|dy\,dx$
  - The inner integral expands to $\int_{y=0}^{x}(x-y)dy + \int_{y=x}^{N}(y-x)dy$

    which evaluates to $x^2/2 + (N^2/2 - xn + x^2/2)$
  - The outer integral becomes $\int_{x=0}^{N}(x^2 + N^2/2 - xN)dx = N^3/3$

    which we divide by the number of pairs to obtain $N/3$

Seek time: A closer look

- Minimum: time to go from one track to the next
  - 0.3-1.5 ms
- Maximum: time to go from innermost to outermost track
  - more than 10ms; up to over 20ms
- Average: average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across disk
- Head switch time: time to move from track i on one surface to the same track on a different surface
  - range similar to minimum seek time

Rotation time: A closer look

- Today most disk rotate at 4200 to 15,000 RPM
  - ≈15ms to 4ms per rotation
  - good estimate for rotational latency is half that amount
- Head starts reading as soon as it settles on a track
  - track buffering to avoid "shoulda coulda" if any of the sectors flying under the head turn out to be needed

Transfer time: A closer look

- Surface transfer time
  - Time to transfer one or more sequential sectors to/from surface after head reads/writes first sector
  - Much smaller that seek time or rotational latency
    - 512 bytes at 100MB/s ≈ 5μs (0.005 ms)
  - Lower for outer tracks than inner ones
    - same RPM, but more sectors/track: higher bandwidth!
- Host transfer time
  - time to transfer data between host memory and disk buffer
    - 60MB/s (USB 2.0); 640 MB/s (USB 3.0); 25.GB/s (Fibre Channel 256GFC)

Buffer Memory

- Small cache [``Track buffer", 8 to 16 MB] holds data
  - read from disk
  - about to be written to disk
- On write
  - write back (return from write as soon as data is cached)
  - write through (return once it is on disk)

Computing I/O time

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

- The rate of I/O is computed as

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/0}}$$

Example:

Toshiba MK3254GSY (2008)

Size	
Platters/Heads	2/4
Capacity	320GB
Performance	
Spindle speed	7200 RPM
Avg. seek time R/W	10.5/12.0 ms
Max. seek time R/W	19 ms
Track-to-track	1 ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer memory	16MB
Power	
Typical	16.35 W
Idle	11.68 W

500 Random Reads

Size	
Platters/Heads	2/4
Capacity	320GB
Performance	
Spindle speed	7200 RPM
Avg. seek time R/W	10.5/12.0 ms
Max. seek time R/W	19 ms
Track-to-track	1 ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer memory	16MB
Power	
Typical	16.35 W
Idle	11.68 W

- Workload
  - 500 read requests, randomly chosen sector
  - served in FIFO order
- How long to service them?
  - 500 times (seek + rotation + transfer)
  - seek time: 10.5 ms (avg)
  - rotation time:
    - 7200 RPM = 120 RPS
    - rotation time 8.3 ms
    - on average, half of that: 4.15 ms
  - transfer time
    - at least 54 MB/s
    - 512 bytes transferred in (.5/54,000) seconds = 9.26μs
  - Total time:
    - 500 x (10.5 + 4.15 + 0.009) ≈ 7.33 sec

$$R_{I/O} = \frac{500 \times .5 \times 10^{-3} MB}{7.33\, s} = 0.034\, MB/s$$

500 Sequential Reads

Size	
Platters/Heads	2/4
Capacity	320GB
Performance	
Spindle speed	7200 RPM
Avg. seek time R/W	10.5/12.0 ms
Max. seek time R/W	19 ms
Track-to-track	1 ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer memory	16MB
Power	
Typical	16.35 W
Idle	11.68 W

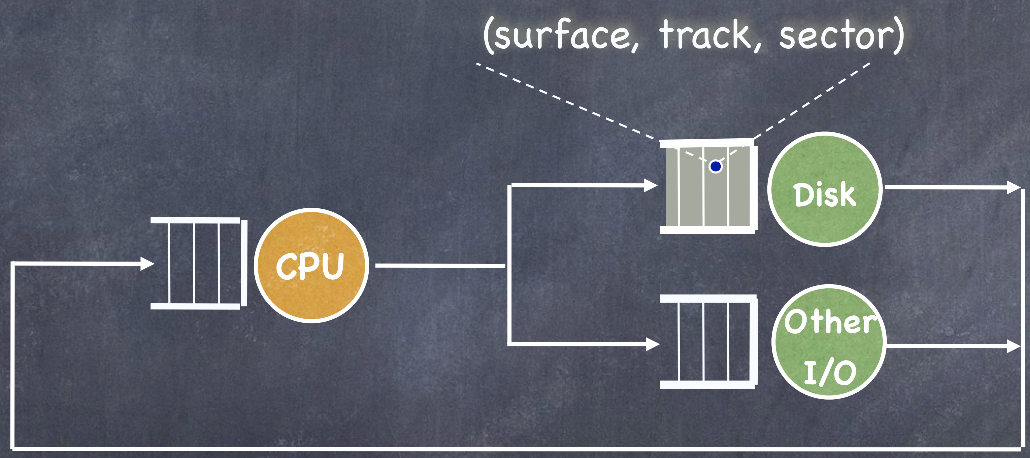
- Workload
  - 500 read requests for sequential sectors on the same track
  - served in FIFO order
- How long to service them?
  - seek + rotation + 500 times transfer
  - seek time: 10.5 ms (avg)
  - rotation time:
    - 4.15 ms, as before
  - transfer time
    - outer track: 500 x (.5/128000) ≈ 2ms
    - inner track: 500 x (.5/54000) seconds ≈ 4.6ms
  - Total time is between:
    - outer track: (2 + 4.15 + 10.5) ms ≈ 16.65 ms

      $R_{I/O} = \frac{500 \times .5 \times 10^{-3} MB}{16.65\,ms} = 15.02\,MB/s$
    - inner track: (4.6 + 4.15 + 10.5) ms ≈ 19.25 ms

      $R_{I/O} = \frac{500 \times .5 \times 10^{-3} MB}{19.25\,ms} = 12.99\,MB/s$

Disk Head Scheduling

- In a multiprogramming/time sharing environment, a queue of disk I/Os can form



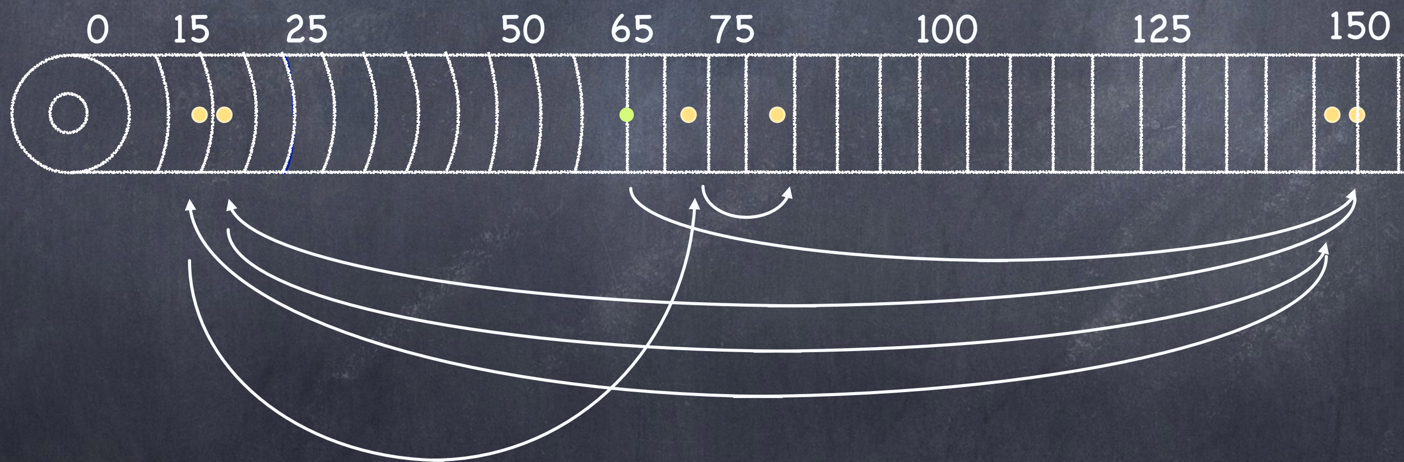
- OS maximizes disk I/O throughput by minimizing head movement through disk head scheduling
  - and this time we have a good sense of the length of the task!

FCFS

- Assume a queue of request exists to read/write tracks

....	83	72	14	147	16	150

and the head is on track 65



FCFS scheduling results in disk head moving 550 tracks

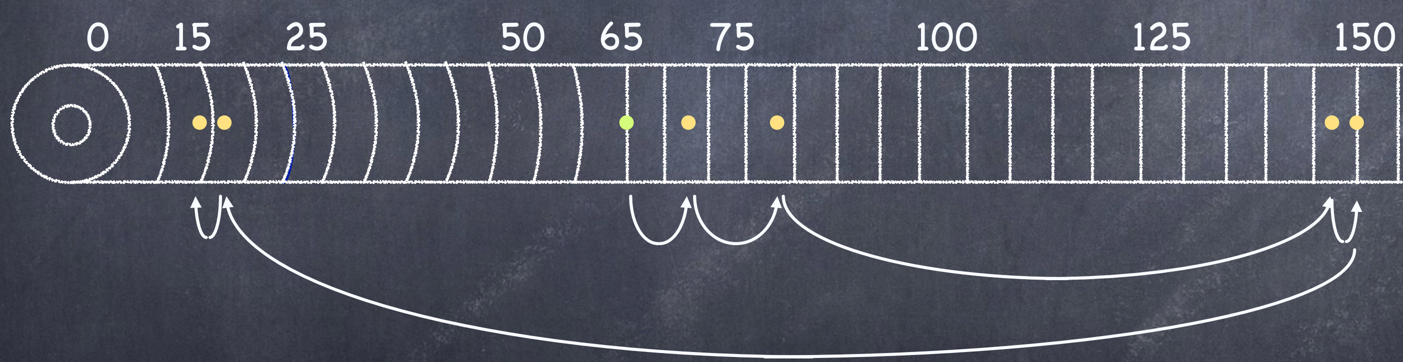
and makes no use of what we know about the length of the tasks!

SSTF:
Shortest Seek Time First

- Greedy scheduling

Rearrange queue from: ···· | 83 | 72 | 14 | 147 | 16 | 150 |

to: ···· | 14 | 16 | 150 | 147 | 83 | 72 |



Head moves 221 tracks BUT

- OS knows blocks, not tracks (easily fixed)
- starvation

SCAN Scheduling
"Elevator"

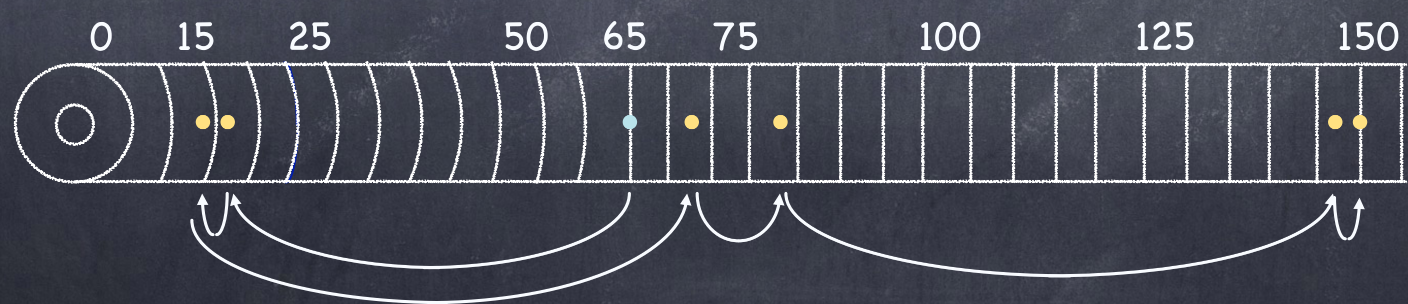
- Move the head in one direction until all requests have been serviced, and then reverse
  - sweeps disk back and forth

Rearrange queue from:

....	83	72	14	147	16	150

to:

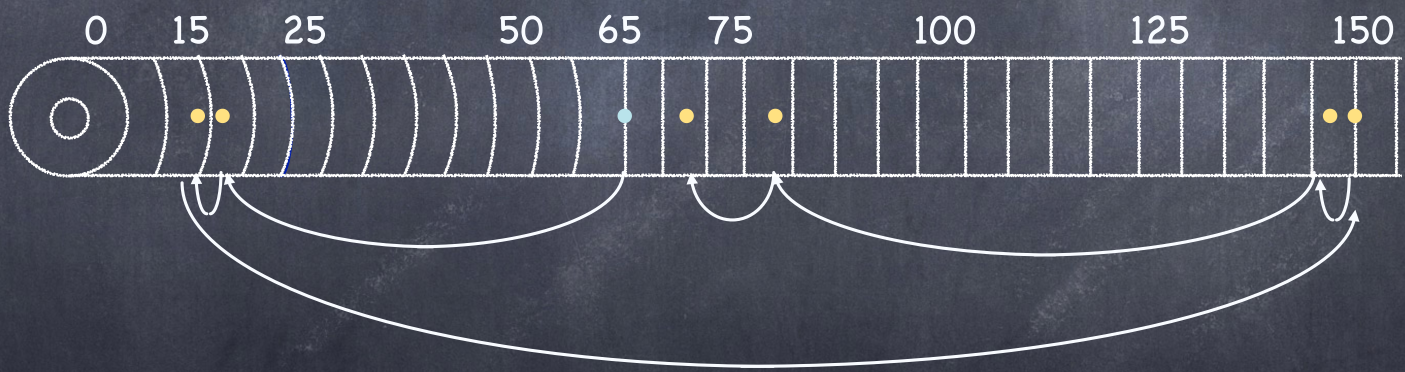
....	150	147	83	72	14	16



Head moves 187 tracks.

C-SCAN scheduling

- Circular SCAN
  - sweeps disk in one direction (from outer to inner track), then resets to outer track and repeats



- More uniform wait time than SCAN
  - moves head to serve requests that are likely to have waited longer

OS Outsources Scheduling Decisions

- Selecting which track to serve next should include rotation time (not just seek time!)
  - SPTF: Shortest Positioning Time First
- Hard for the OS to estimate rotation time accurately
  - Hierarchical decision process
    - OS sends disk controller a batch of "reasonable" requests
    - disk controller makes final scheduling decisions

Back to Storage…

What qualities we want from storage?

- Reliable: It returns the data you stored
- Fast: It returns the data you stored promptly
- Affordable: It does not break the bank
- Plenty: It holds everything you need

What we may instead get is a SLED!

- Single, Large, Expensive Disk