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)
# Harmony output

<table>
<thead>
<tr>
<th>states</th>
<th>746532</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety Violation</td>
<td></td>
</tr>
</tbody>
</table>


Harmony assertion failed

Reference string
Presenting... The Belady CLOCK Anomaly

<table>
<thead>
<tr>
<th>3 frames</th>
<th>6 misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>5</td>
</tr>
<tr>
<td>5*</td>
<td>2</td>
</tr>
<tr>
<td>2*</td>
<td>2*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 frames</th>
<th>7 misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>4</td>
</tr>
</tbody>
</table>
| 5*       | 5*       | 5 *
| 5*       | 5*       | 1 |
| 5         | 1        | 1 |
| *        | 3        | 3 |
| 7         | 5        |
| 4         | 3        |
| 3         | 3        |
| 3         | 3        | 3 |
| 4         | 5        |
| 3*        | 4        |
| *        | 2        | 2 |
| 7         | 4        |
| 5         | 4*       |
| 5         | 4*       |
| 5         | 5        |
| 5         | 5        |
| *        | 2        | 2 |
| 4*       | 2        |
| 7         | 4        |
| 5         | 4*       |
| 5         | 4*       |
| 5         | 5        |

red = miss
* is clockhand
□ is recent bit
Presenting... The Belady CLOCK Anomaly

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>5</td>
<td>5</td>
<td>5*</td>
<td>2</td>
<td>2</td>
<td>2*</td>
<td>2*</td>
</tr>
<tr>
<td>*</td>
<td>4</td>
<td>4</td>
<td>4*</td>
<td>4*</td>
<td>4</td>
<td>4</td>
<td>4*</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

red = miss
* is clockhand
☐ is recent bit

Stack property first violated
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

- **Track**: a set of tracks on different surfaces with the same track index.
- **Block/Sector**: typically 512 bytes, spare sectors added for fault tolerance.
- **Surface**: each platter has two surfaces.
- **Cylinder**: set of tracks on different surfaces with the same track index.
- **Head**: reads by sensing a magnetic field, writes by creating one.
- **Arm Assembly**: floats on air cushion created by spinning disk.
- **Spindle**: a thin cylinder that holds magnetic material; each platter has two surfaces.
- **Data on a track can be read without moving arm**
- **Track skewing**: staggering logical address 0 on adjacent one to account for time to move head.

---

**Typical 2018 RPMs**
- 4200–15000 RPM
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 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 adjustments)
- 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 adjustments)
- 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 adjustments)
- 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

  \[
  \text{sum of distances} = N \times (\sum_{x=0}^{N-1} x + \sum_{y=0}^{N-1} y) = N^2 \cdot \frac{N(N-1)}{2}
  \]

  which we compute as

  \[
  \frac{N^2 \cdot \frac{N(N-1)}{2}}{N^2} = \frac{N(N-1)}{2}
  \]
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, \( \binom{N}{2} \) pairs, and sum of distances is

which we compute as

The inner integral expands to

which evaluates to
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, \( \binom{N}{2} \) pairs, and sum of distances is

\[
\sum_{x=0}^{N} \sum_{y=0}^{N} |x-y|
\]

which we compute as

\[
\int_{-N/2}^{N/2} \int_{-N/2}^{N/2} |x-y| dy dx = \frac{N^3}{3}
\]

The inner integral expands to

which evaluates to

The outer integral becomes

which we divide by the number of pairs to obtain \( \frac{N}{3} \)
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 10 ms; up to over 20 ms

- **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 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 than 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); 25.6GB/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

The rate of I/O is computed as
Example: Toshiba MK3254GSY (2008)

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Platters/Heads</td>
<td>2/4</td>
</tr>
<tr>
<td>Capacity</td>
<td>320GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spindle speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Avg. seek time R/W</td>
<td>10.5/12.0 ms</td>
</tr>
<tr>
<td>Max. seek time R/W</td>
<td>19 ms</td>
</tr>
<tr>
<td>Track-to-track</td>
<td>1 ms</td>
</tr>
<tr>
<td>Surface transfer time</td>
<td>54-128 MB/s</td>
</tr>
<tr>
<td>Host transfer time</td>
<td>375 MB/s</td>
</tr>
<tr>
<td>Buffer memory</td>
<td>16MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical</td>
<td>16.35 W</td>
</tr>
<tr>
<td>Idle</td>
<td>11.68 W</td>
</tr>
</tbody>
</table>
500 Random Reads

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Platters/Heads</td>
<td>2/4</td>
</tr>
<tr>
<td>Capacity</td>
<td>320GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spindle speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Avg. seek time R/W</td>
<td>10.5/12.0 ms</td>
</tr>
<tr>
<td>Max. seek time R/W</td>
<td>19 ms</td>
</tr>
<tr>
<td>Track-to-track</td>
<td>1 ms</td>
</tr>
<tr>
<td>Surface transfer time</td>
<td>54-128 MB/s</td>
</tr>
<tr>
<td>Host transfer time</td>
<td>375 MB/s</td>
</tr>
<tr>
<td>Buffer memory</td>
<td>16MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical</td>
<td>16.35 W</td>
</tr>
<tr>
<td>Idle</td>
<td>11.68 W</td>
</tr>
</tbody>
</table>

**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

\[
R_{I/O} = \frac{500 \times 0.05 \times 10^{-3} MB}{7.33 \text{ s}} = 0.034 \text{ MB/s}
\]
500 Sequential Reads

### 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:
\[ 4.15 \text{ ms}, \text{ as before} \]
transfer time
\[ \text{outer track: } 500 \times (0.5/128000) \approx 2\text{ms} \]
\[ \text{inner track: } 500 \times (0.5/54000) \text{ seconds} \approx 4.6\text{ms} \]
Total time is between:
\[ \text{outer track: } (2 + 4.15 + 10.5) \text{ ms} \approx 16.65 \text{ ms} \]
\[ R_{I/O} = \frac{500 \times 5 \times 10^{-3} \text{MB}}{16.65 \text{ ms}} \approx 15.02 \text{MB/s} \]
\[ \text{inner track: } (4.6 + 4.15 + 10.5) \text{ ms} \approx 19.25 \text{ ms} \]
\[ R_{I/O} = \frac{500 \times 5 \times 10^{-3} \text{MB}}{19.25 \text{ ms}} \approx 12.99 \text{MB/s} \]

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Platters/Heads</td>
<td>2/4</td>
</tr>
<tr>
<td>Capacity</td>
<td>320GB</td>
</tr>
</tbody>
</table>

### Performance

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spindle speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Avg. seek time R/W</td>
<td>10.5/12.0 ms</td>
</tr>
<tr>
<td>Max. seek time R/W</td>
<td>19 ms</td>
</tr>
<tr>
<td>Track-to-track</td>
<td>1 ms</td>
</tr>
<tr>
<td>Surface transfer time</td>
<td>54–128 MB/s</td>
</tr>
<tr>
<td>Host transfer time</td>
<td>375 MB/s</td>
</tr>
<tr>
<td>Buffer memory</td>
<td>16MB</td>
</tr>
</tbody>
</table>

### Power

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical</td>
<td>16.35 W</td>
</tr>
<tr>
<td>Idle</td>
<td>11.68 W</td>
</tr>
</tbody>
</table>
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 requests exists to read/write tracks

\[
\begin{array}{cccccc}
\ldots & 83 & 72 & 14 & 147 & 16 & 150 \\
\end{array}
\]

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.

Rearrange queue from:

\[
\begin{align*}
83 & \\
72 & \\
14 & \\
147 & \\
16 & \\
150 & \\
\end{align*}
\]

to:

\[
\begin{align*}
150 & \\
147 & \\
83 & \\
72 & \\
14 & \\
16 & \\
\end{align*}
\]

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