Hard Disk Drives

CS333
Williams College

Prior Knowledge

- Memory Hierarchy
  - Often orders of magnitude difference between levels
    - Speed/Capacity/$
  - Variety of transfer sizes

- I/O Devices
  - Software Interface (Device Drivers)
  - Firmware (Program logic running on the devices themselves)

This Class

- HDD “guarantees”
  - Performance & correctness

- Physical components and Geometry
  - Breaking down an I/O

- The role of caching

- Scheduling requests
  - Who schedules requests?
  - How is schedule determined?

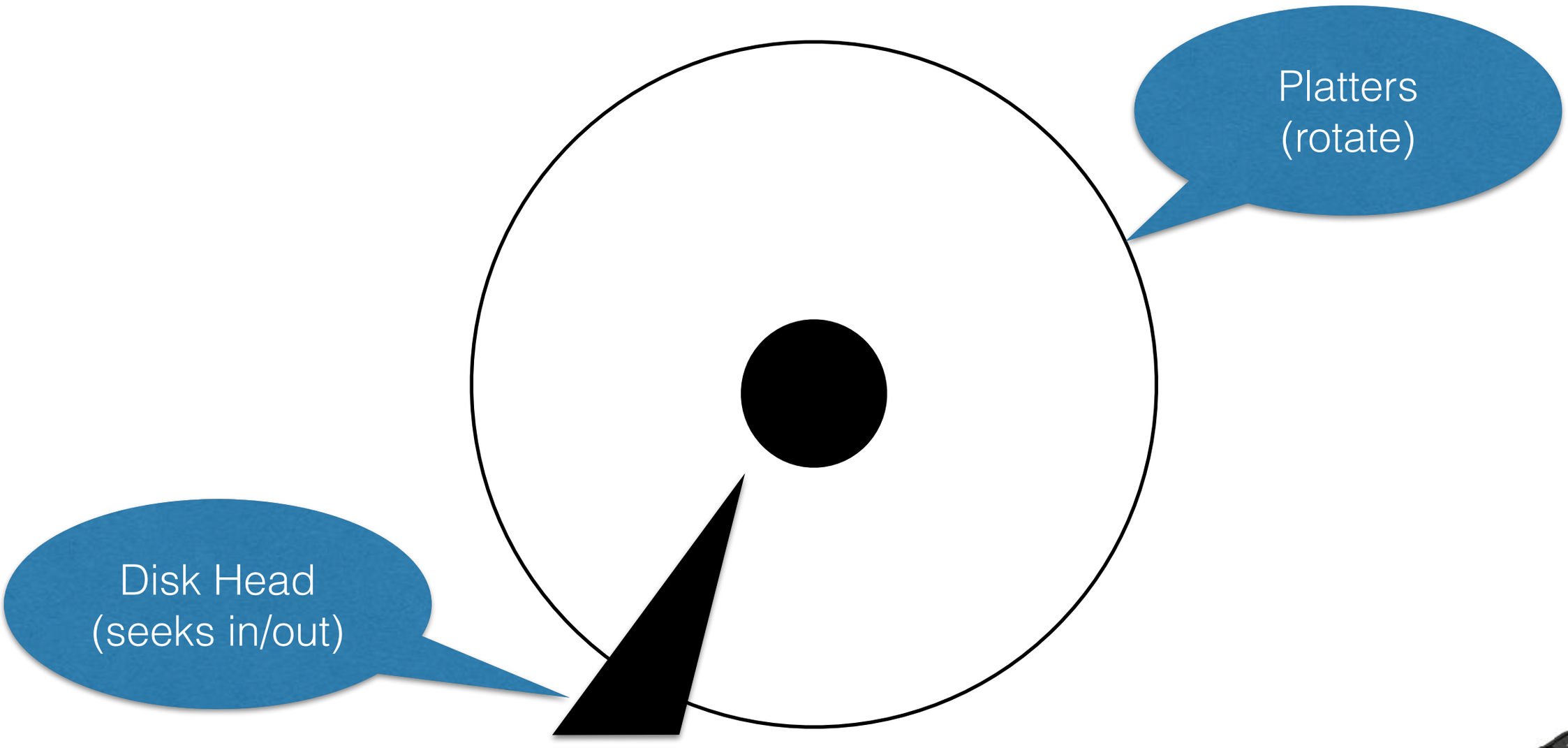
Hard Disk Drives (HDDs)

Despite the existence of newer, faster devices, HDDs have many benefits and use cases:

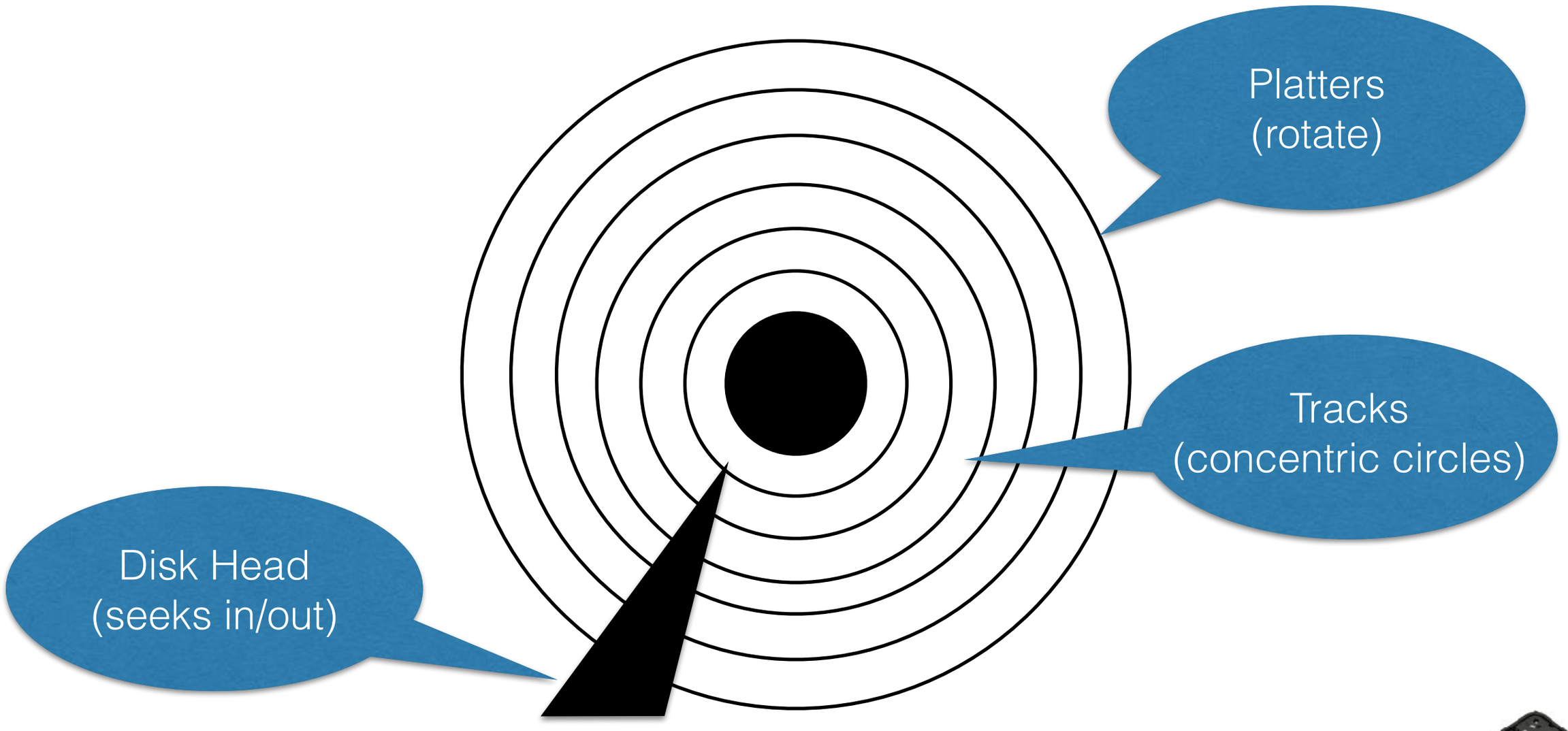
- High capacity, low cost ($/GiB)

- Predictable performance
  - "Unwritten contract": Tracks (LBAs) near each other are more efficient to access than tracks (LBAs) that are far away

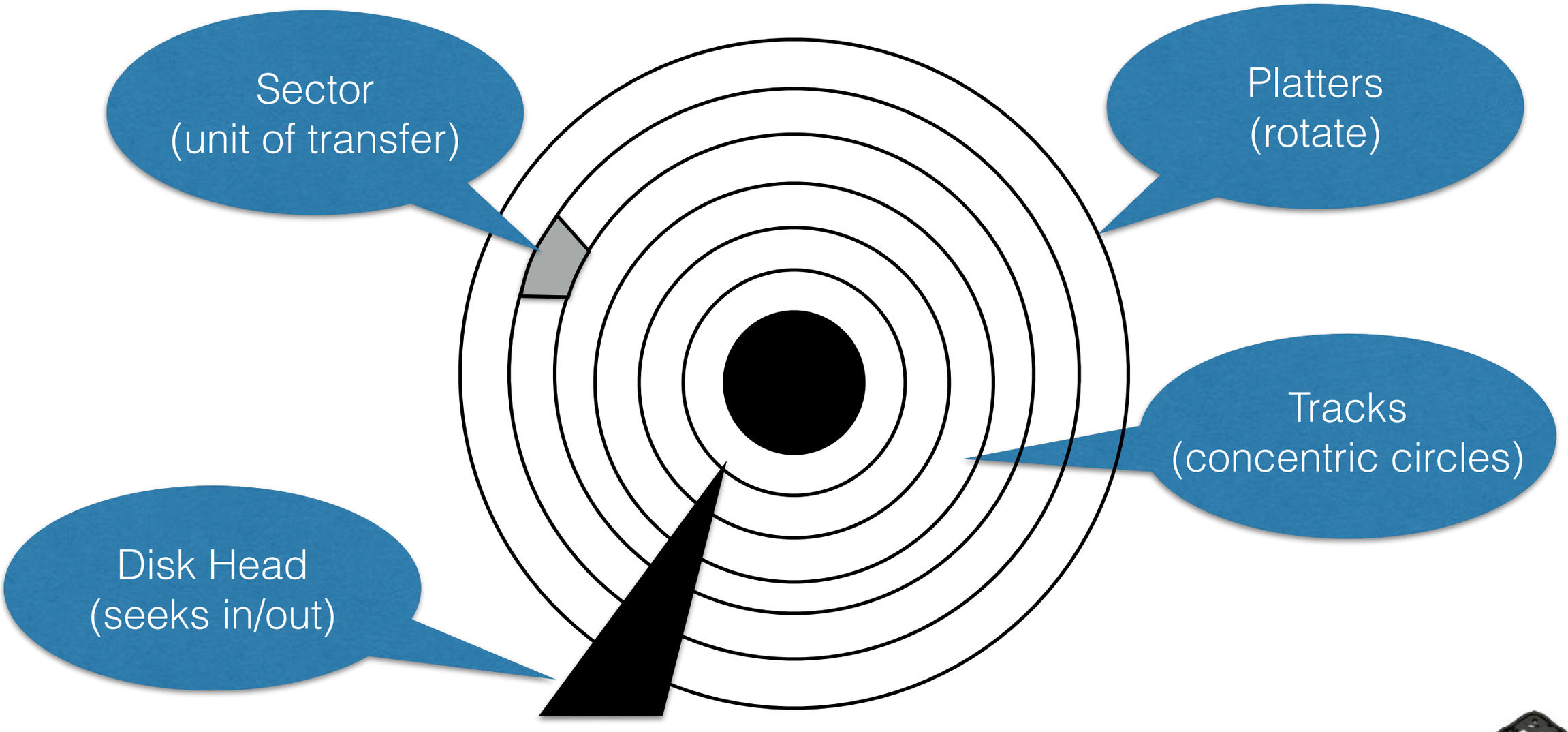
HDDs



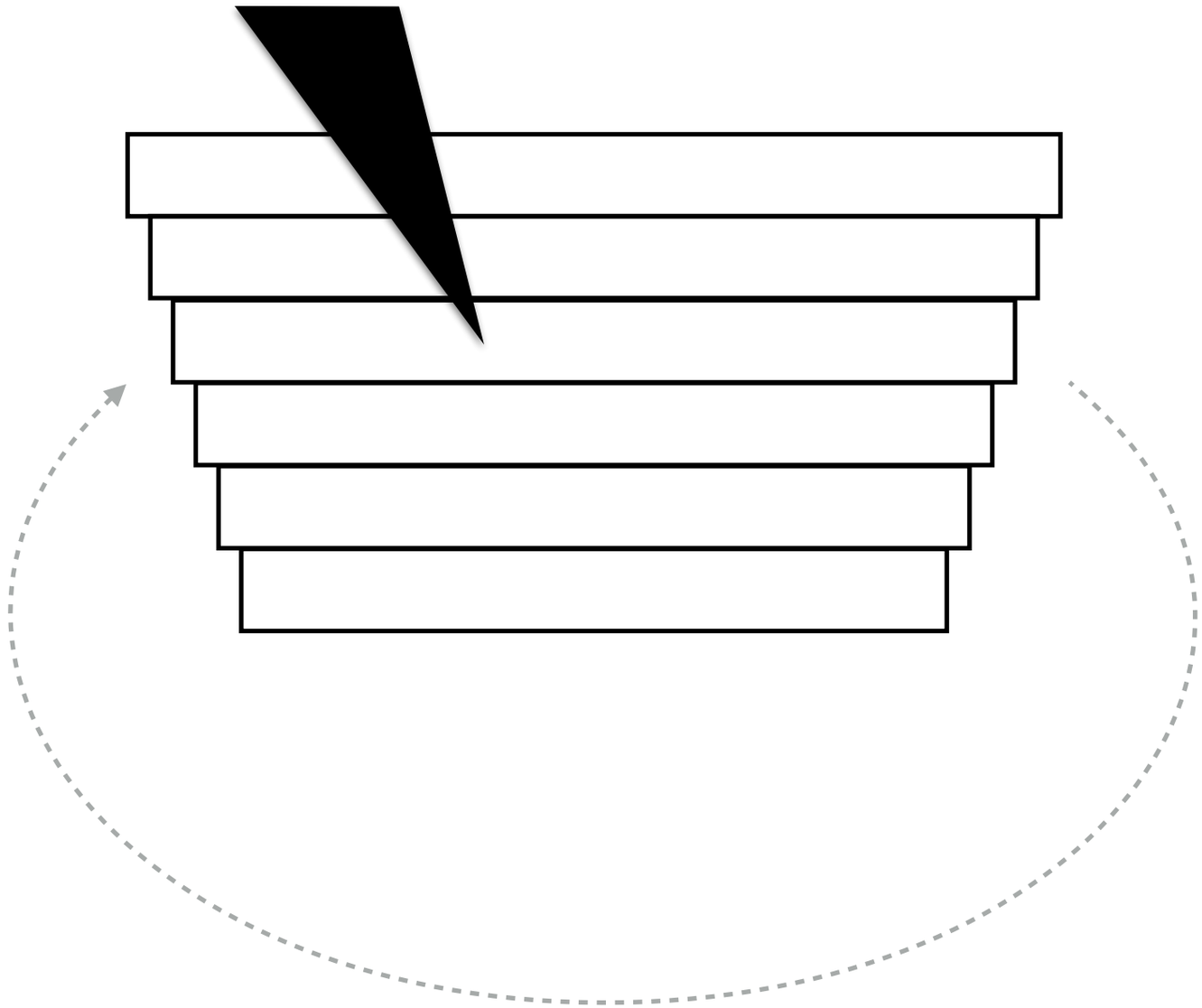
HDDs


Platters
(rotate)
Tracks
(concentric circles)
Disk Head
(seeks in/out)

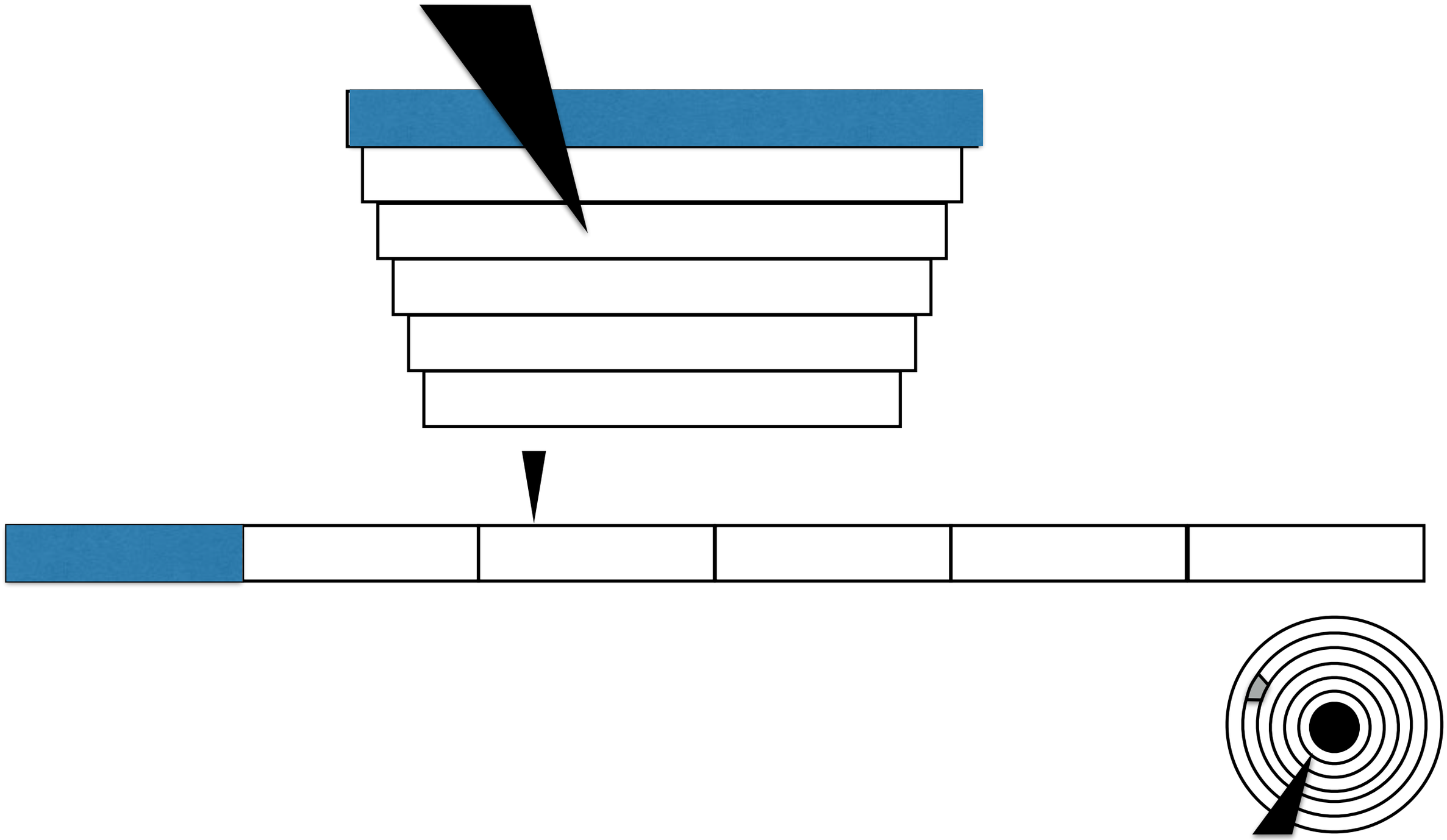
HDDs


Sector
(unit of transfer)
Platters
(rotate)
Tracks
(concentric circles)
Disk Head
(seeks in/out)

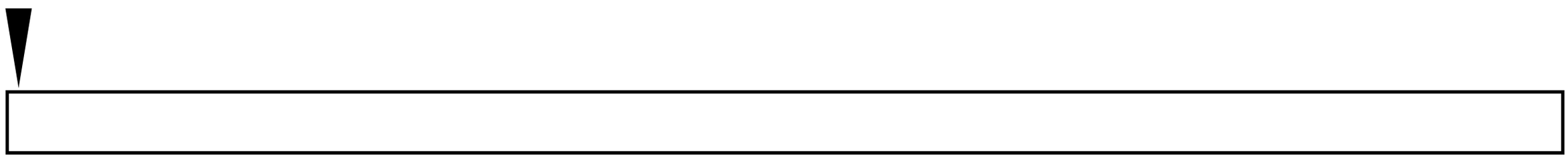
Visualizing HDDs: “Unwind” The Tracks



Seeking through the Linear Address Space

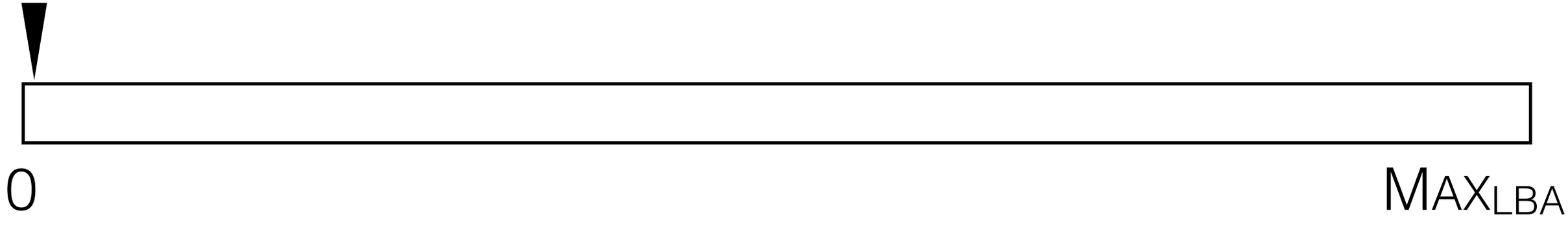


HDDs



HDDs

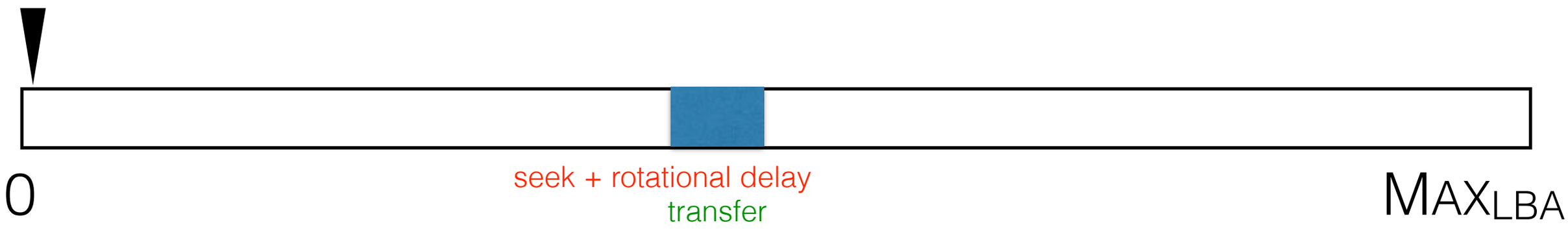
- Disks are addressed by **LBA**: [**0-MAX$_{LBA}$**)
- Transfer data in fixed-size units: "disk block" (~sector)
- "block interface" used for both reads and writes



Breaking Down an I/O

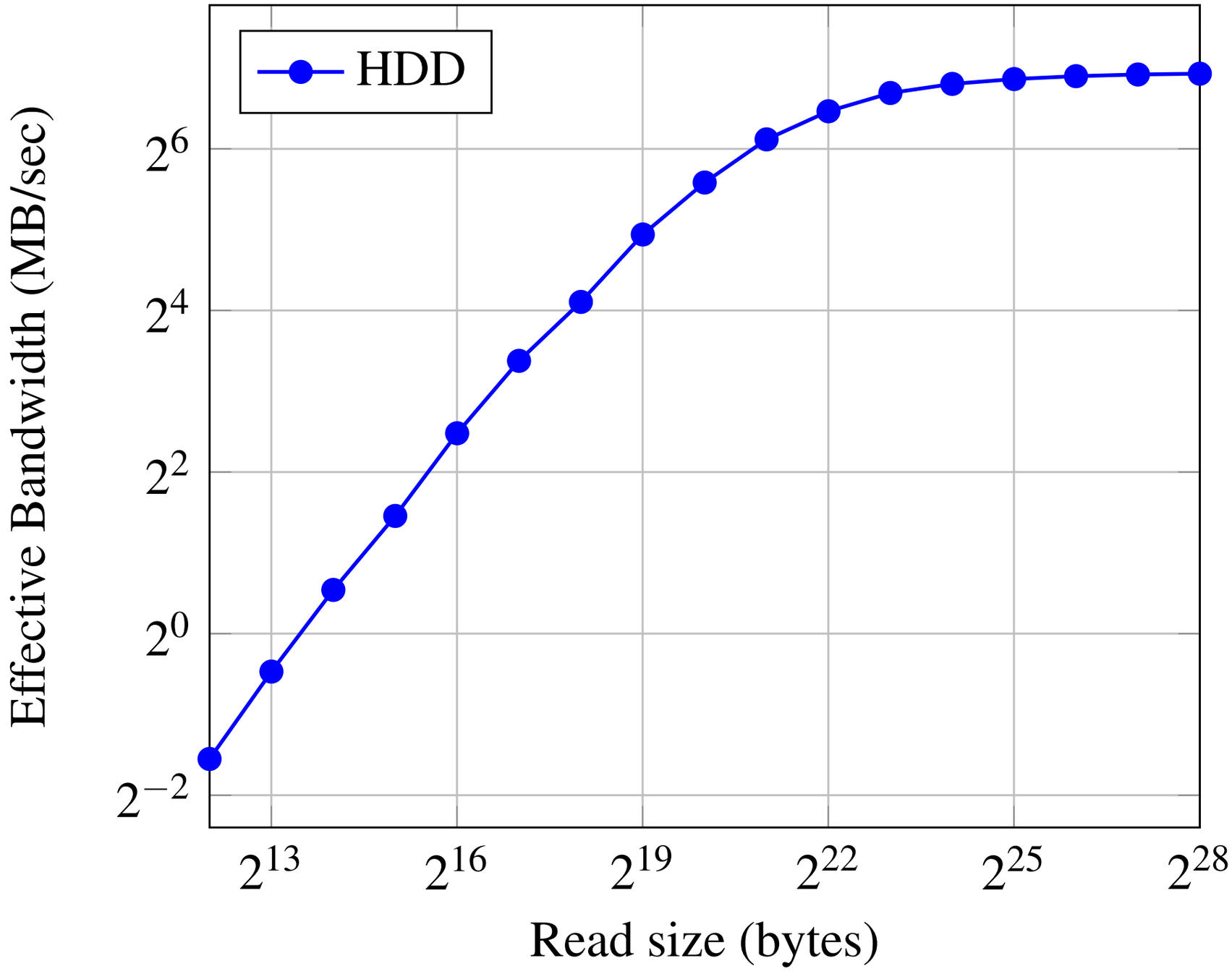
- Two costs to every operation:
  - **Setup:** Moving the disk head, rotating the platters
    - Everything that you need to do to position the disk's read/write head directly above the first byte of your target sector
  - **Transfer:** Reading/writing data while the disk rotates

Ex: `data <- read(10024, 10048)`



Performance Observations

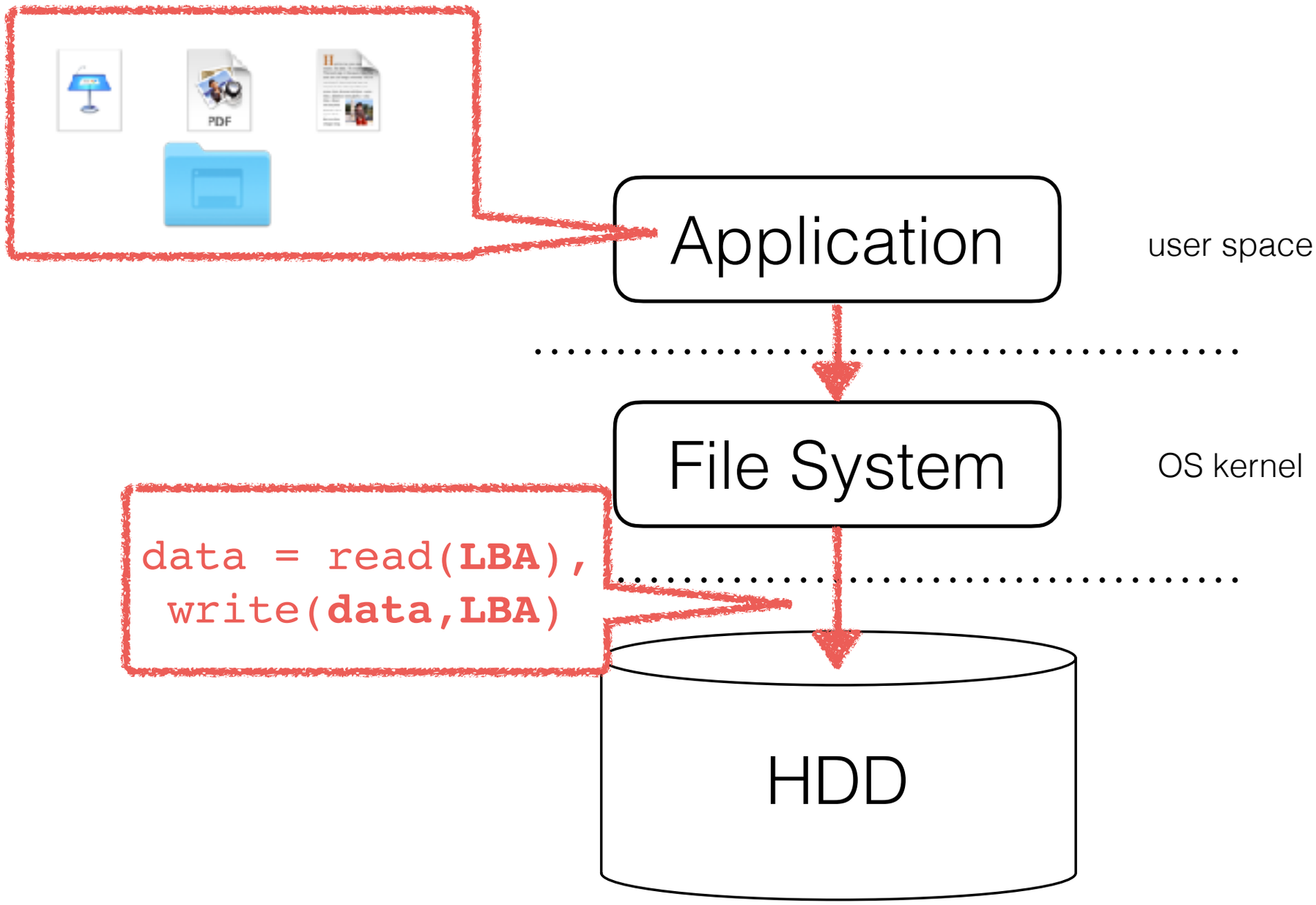
- **Setup** (placing the disk head) is expensive O(10 ms)
  - **seeking** to target track
    - Up to a full **rotational delay** to locate sector
- Once the disk head is in place, data **transfer** is quite fast O(100 MiB/s)



To maximize performance, minimize seeks and maximize the ratio of time spent transferring.

Why Does This Matter?

Simplified Storage Stack

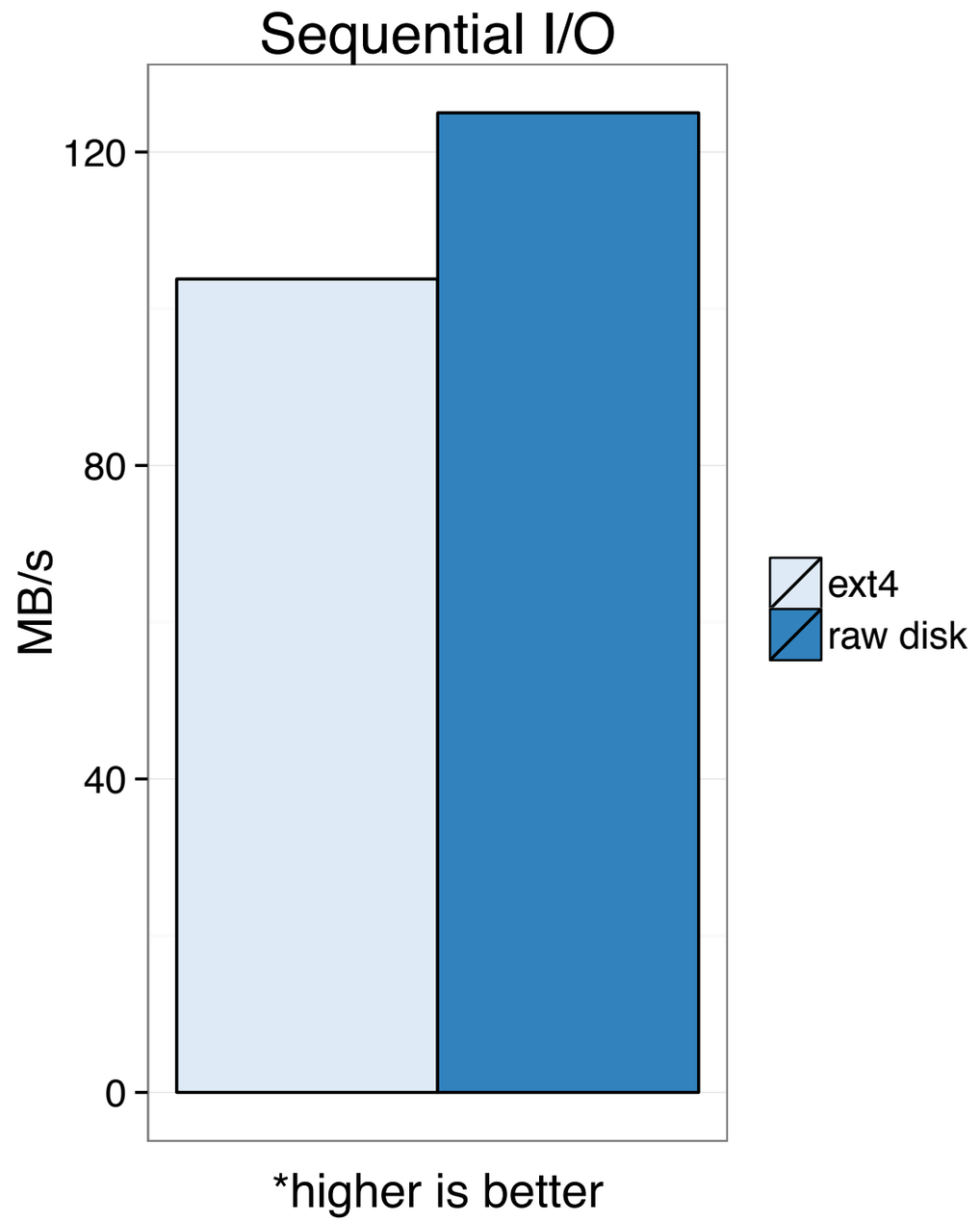


Our file systems transform application requests into block requests… how well are they doing?

Good Cases

Sequential I/O

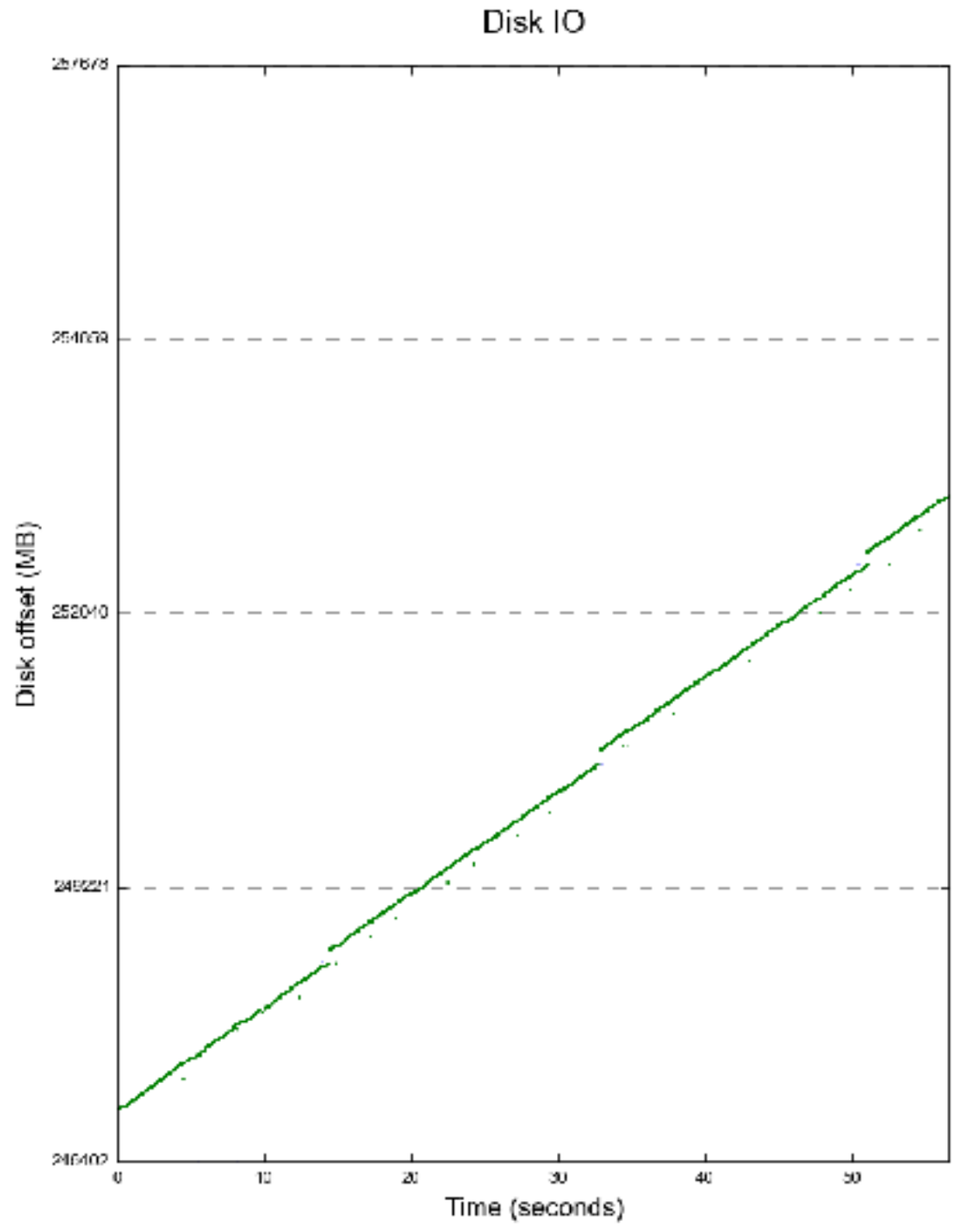
- Write a large file to an empty file system.
- Read an existing file in order



Good Cases

- Write a large file to an empty file system

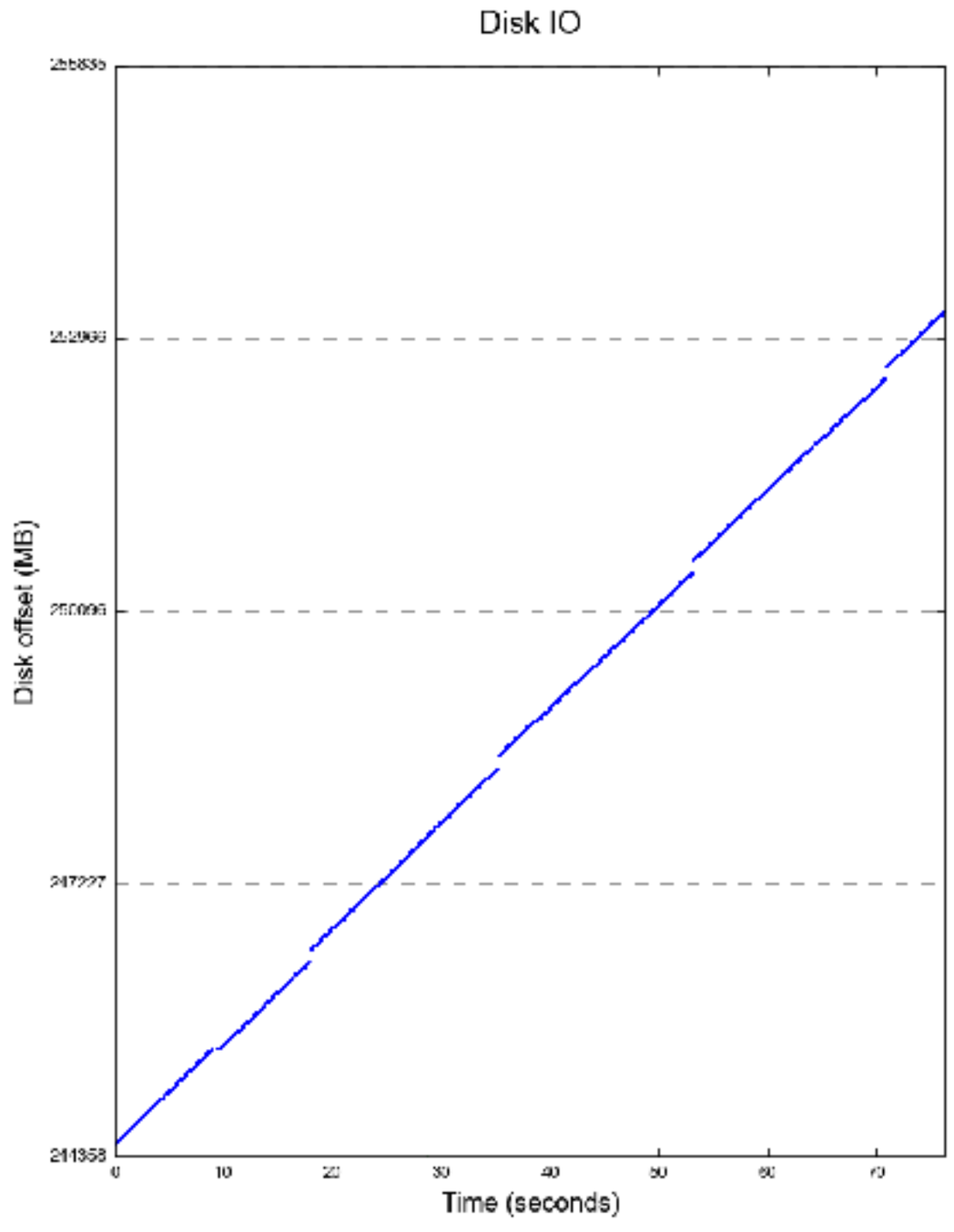
y-axis = offset, so a “straight line” means consecutive block addresses.



Good Cases

- Read an existing file in order

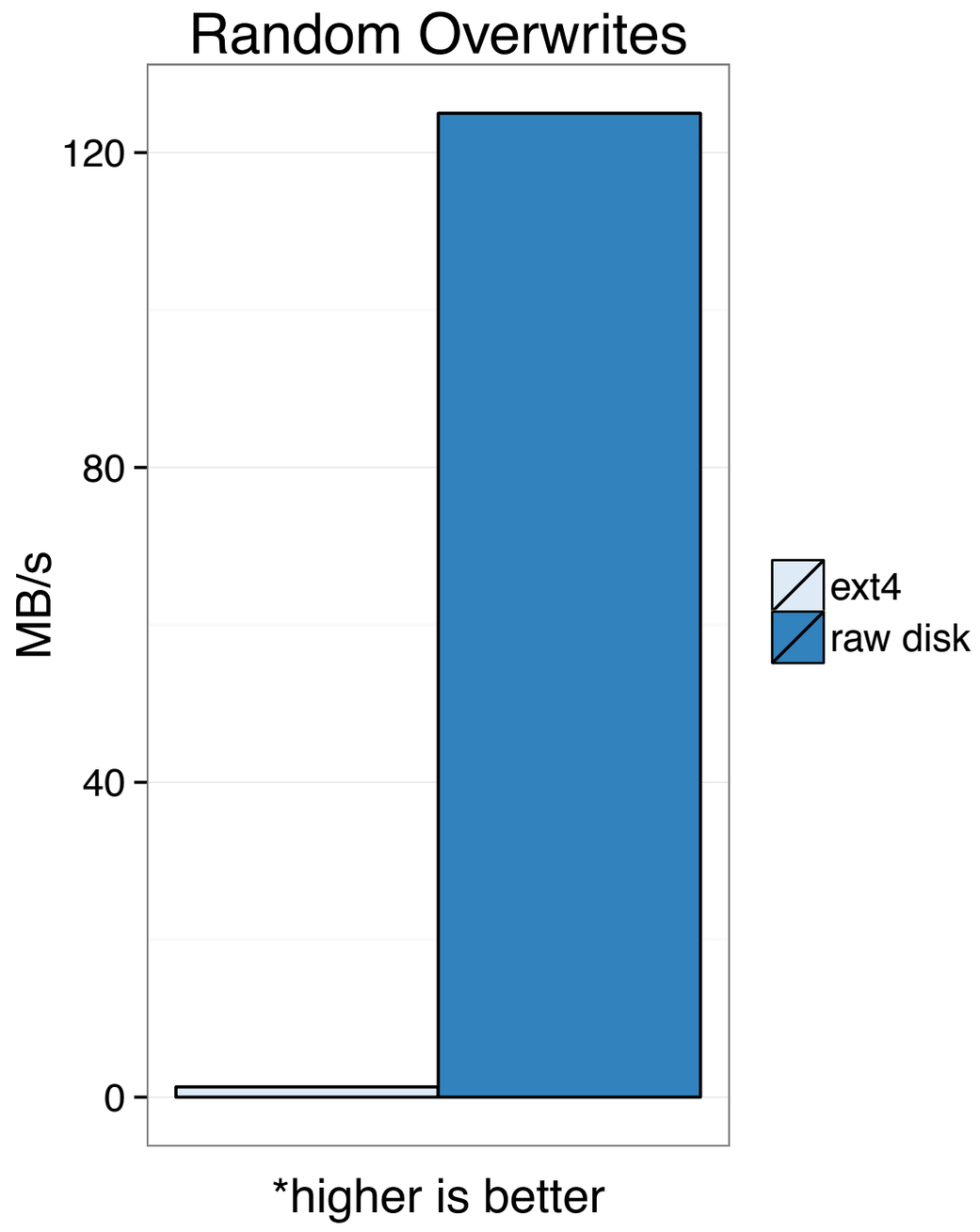
y-axis = offset, so a “straight line” means consecutive block addresses.



Bad Cases

Random I/O

- Randomly update an existing file
- Randomly reading an existing file
- Reading data from many independent files

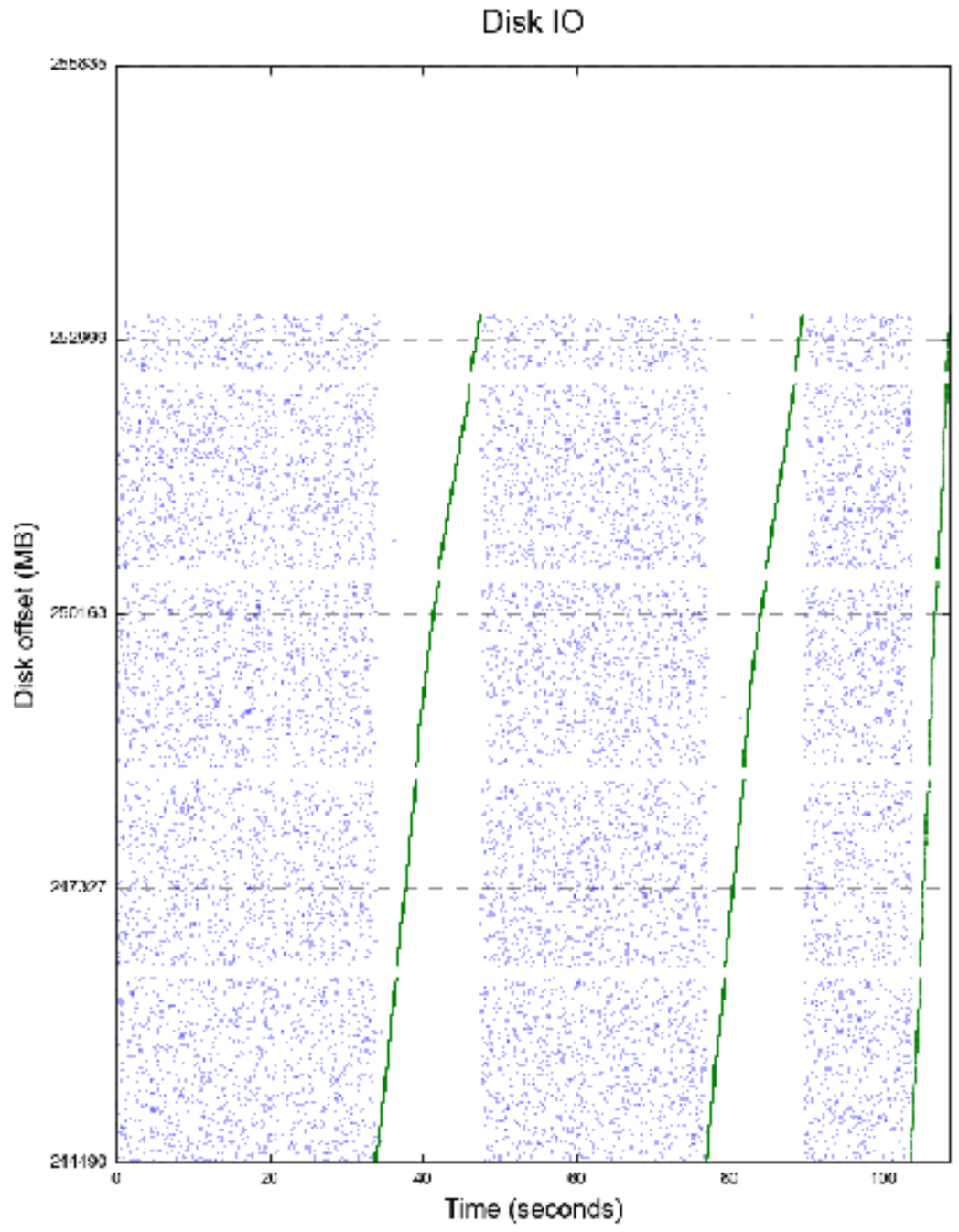


Bad Cases

- Randomly update an existing file

Note: these requests are 4-byte updates, so the old data is first read, then the block is modified in memory, and later the whole block is written back to disk.

We see that the file system benefits from caching.



(blue = reads; green = writes)

Takeaway:

Locality Matters

Disk Geometry

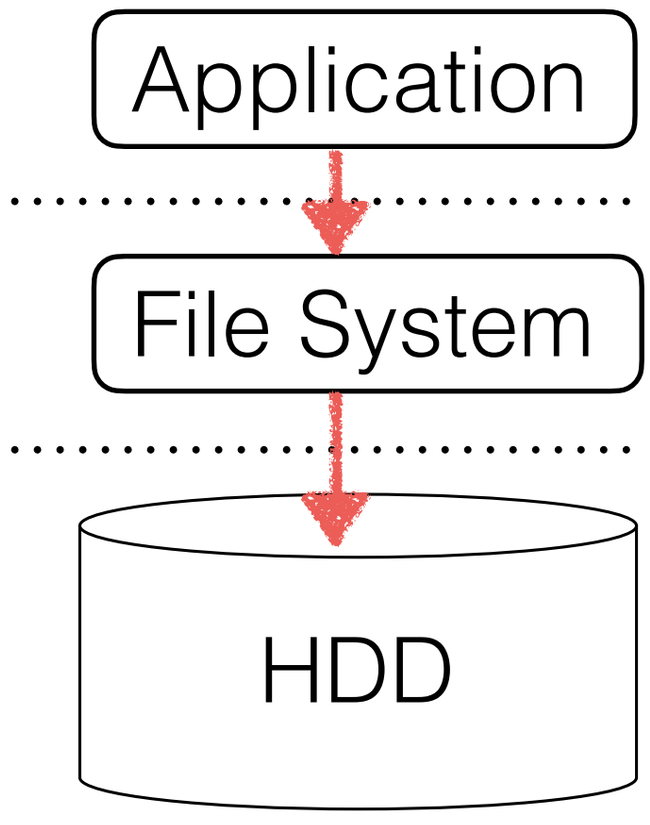
- High level idea gets us most of the way, but disk geometry adds complications (opportunities?)
  - **Multi-zoned disks**
    - The inside/outside of the disk might have different densities (and therefore speeds)
  - **Track Skew**
    - Locate LBAs on consecutive tracks so that seeking from last LBA on one track to first on next does not incur rotational delay

Takeaway:

Sometimes it pays to “open the black box”. Abstractions are important, but they sometimes hide useful details.

Scheduling

- High Level Question: given a set of requests that must be completed (LBAs), what order should we schedule the requests?
  - Who does the scheduling?
  - Obstacles?
    - Can we predict the future?
    - Crashes: what if requests are dependent on others & written out of order?
      - Applications: fsync
      - OS: “barriers”

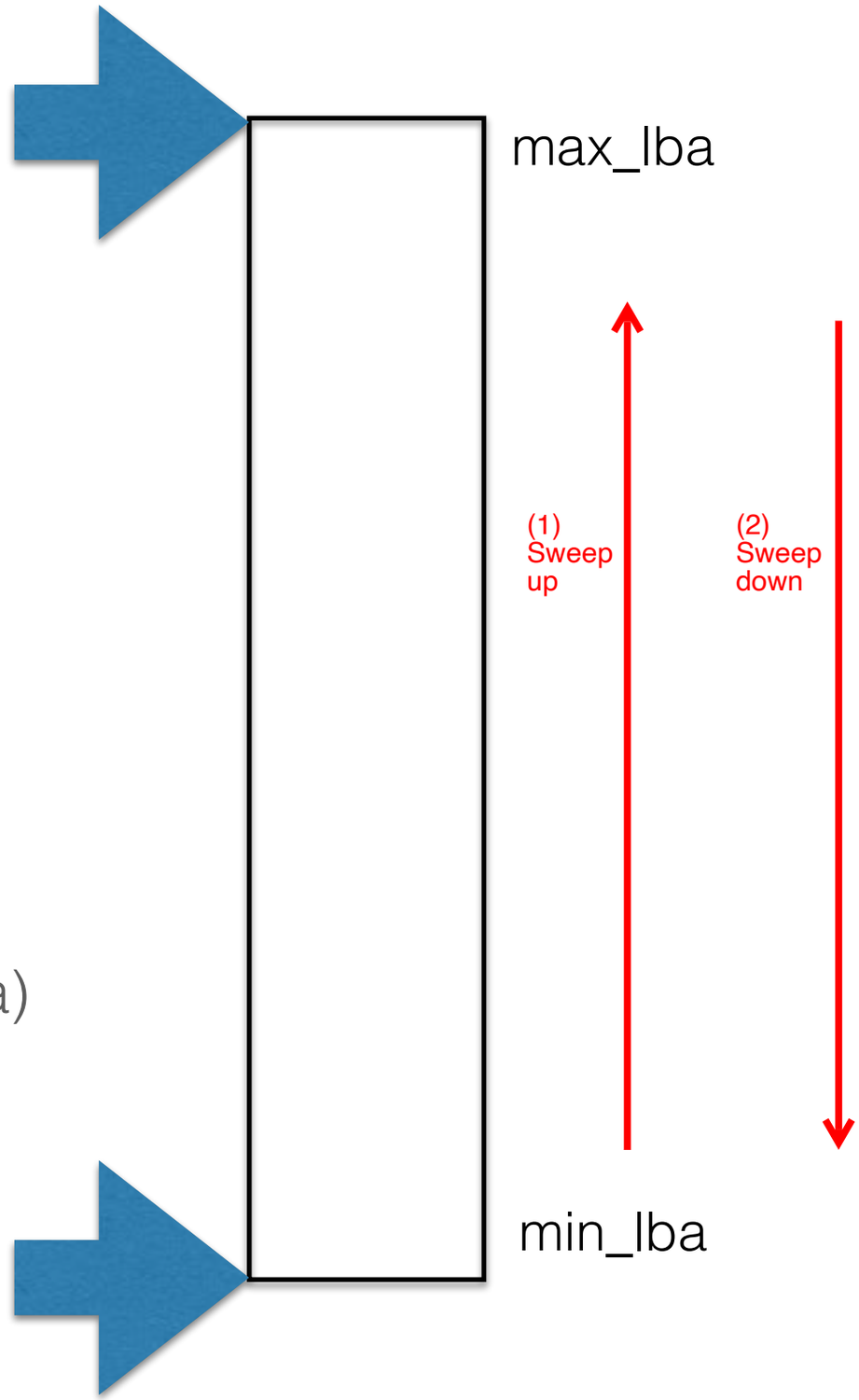


Disk Scheduling

- Greedy: **Shortest job first**
  - Shortest-seek-time-first (SSTF)
  - Nearest-block-first (NBF)
- Problems?
  - **Starvation**: one (or more) requests never receive access to the resources they need to complete

Disk Scheduling

- Elevator (SCAN)
  - Sweep back and forth along the disk tracks, writing blocks as you go
  - (Several variants of this general idea)



Hard Disk Drive Recap

- HDD mechanical behaviors suggest a rough performance model: $t_{req} = t_{setup} + t_{transfer}$
  - t_{setup} = seek + rotational delay
  - $t_{transfer}$ = reading bytes from the platter as it rotates
    - The "unwritten contract": nearby LBAs can be more efficiently accessed than distant LBAs
      - Small and random I/O patterns are expensive
      - Locality matters

- **Goal:** Given what we know about HDDs, we should design software that takes advantage of the HDD best cases and minimizes HDD worst cases.