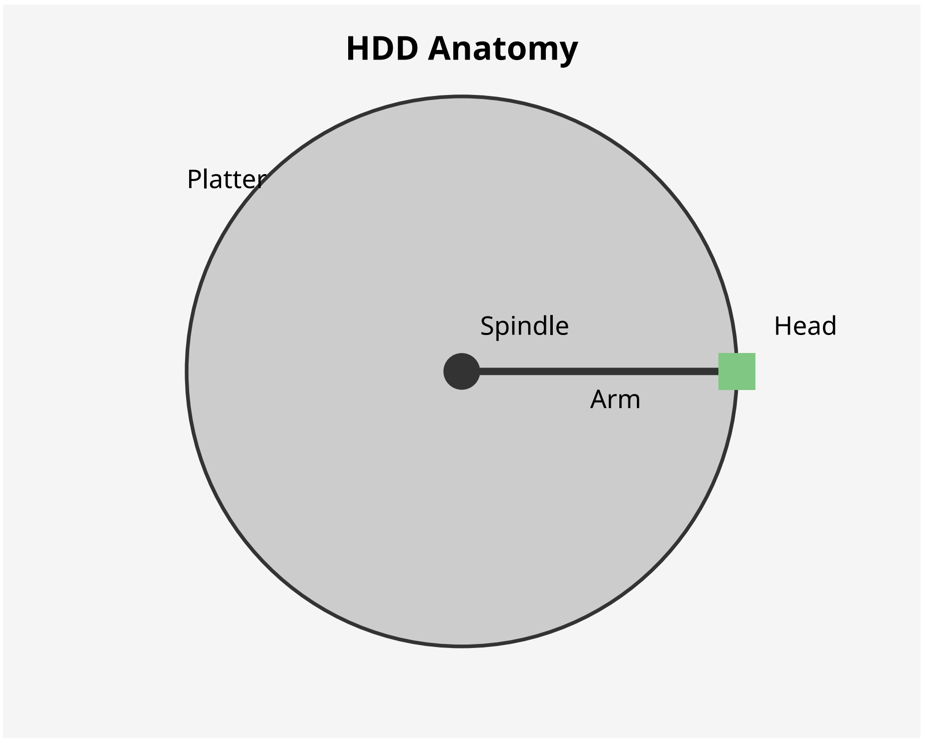
Chapter 37: Hard Disk Drives

1. The Anatomy of a Hard Disk Drive

A hard disk drive (HDD) is a complex piece of engineering. To understand how it works, we need to look at its key components:

- **Platters:** These are circular disks coated with a magnetic material where data is stored.
- **Spindle:** The platters are mounted on a spindle, which rotates them at a constant speed (e.g., 7200 RPM).
- **Disk Head:** A disk head is responsible for reading and writing data to a platter. There is one head per platter surface.
- **Disk Arm:** The heads are attached to a disk arm, which can move across the platters to position the heads over the correct track.



2. The Abstraction: A Linear Array of Blocks

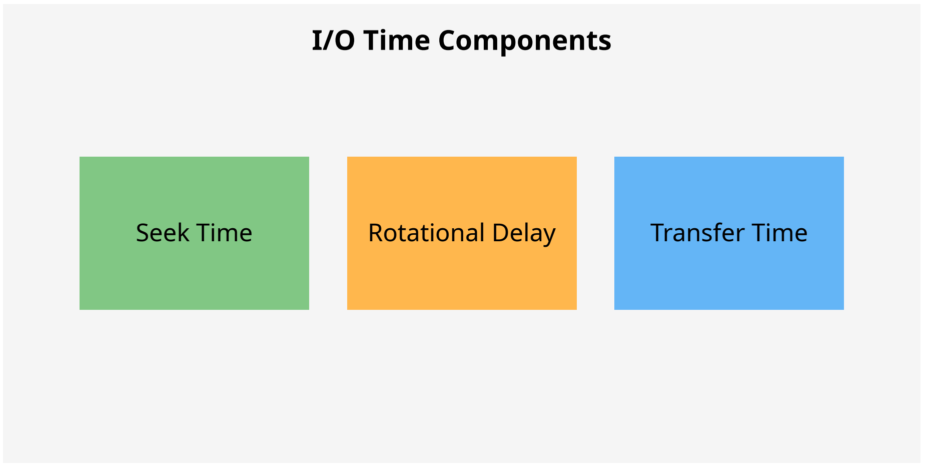
While the physical structure of an HDD is complex, the operating system sees it as a much simpler abstraction: a **linear array of blocks**. Each block is a fixed-size unit of storage (e.g., 512 bytes).

This abstraction allows the OS to interact with the disk in a uniform way, regardless of the underlying hardware details.

3. I/O Time: The Three Components

The time it takes to complete a disk I/O request can be broken down into three main components:

1. **Seek Time:** The time it takes to move the disk arm to the correct track.
2. **Rotational Delay:** The time it takes for the desired sector to rotate under the disk head.
3. **Transfer Time:** The time it takes to transfer the data from the disk to memory.



Of these three, seek time and rotational delay are the most significant, especially for random I/O patterns.

4. Disk Scheduling

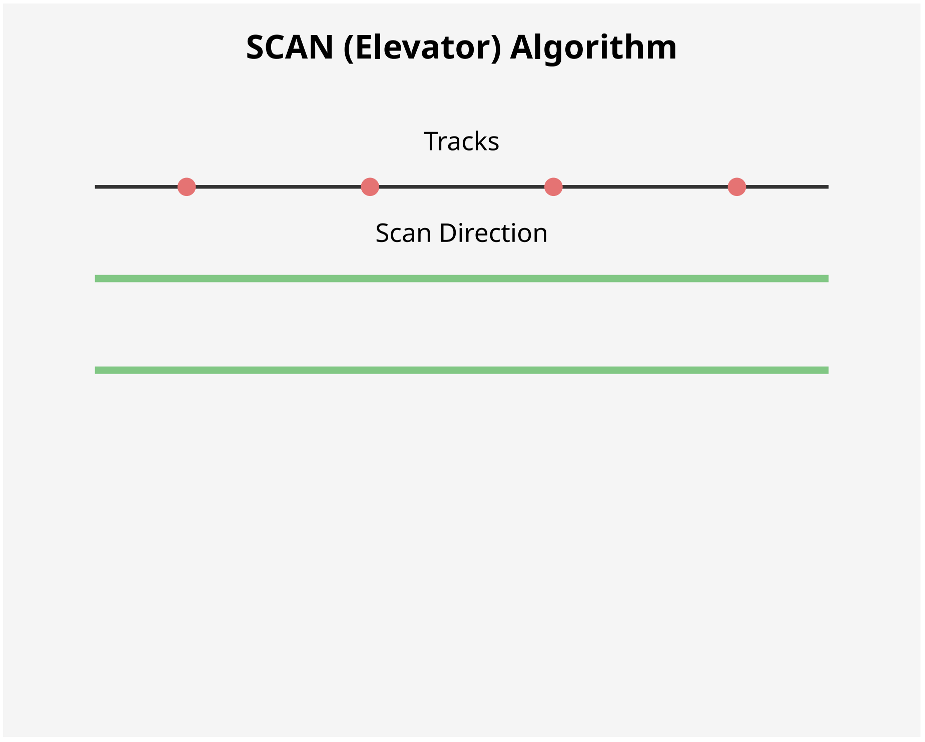
To minimize the time spent on seeks and rotations, the operating system employs **disk scheduling algorithms**. These algorithms reorder the queue of pending I/O requests to improve performance.

4.1. Shortest Seek Time First (SSTF)

- The **Shortest Seek Time First (SSTF)** algorithm selects the request that is closest to the current head position. This can be very effective at reducing seek times, but it can also lead to **starvation**, where some requests may never be serviced if there is a continuous stream of requests closer to the head.

4.2. SCAN (Elevator)

- The **SCAN** algorithm, also known as the **elevator algorithm**, addresses the starvation problem of SSTF. It works by moving the disk arm in one direction, servicing all requests in its path, until it reaches the end of the disk. Then, it reverses direction and repeats the process.



There are variations of SCAN, such as C-SCAN (Circular SCAN), which only moves in one direction and then jumps back to the beginning, providing more uniform wait times.

5. The Importance of Sequential I/O

- One of the key takeaways from this chapter is the importance of **sequential I/O**. When data is accessed sequentially, the seek time and rotational delay are minimized, leading to much higher performance compared to random I/O.

This is why operating systems and file systems go to great lengths to arrange data on disk in a way that promotes sequential access.

6. Advanced Disk Scheduling

6.1. Shortest Positioning Time First (SPTF)

- Modern disk schedulers take into account not just seek time but also rotational delay. The **Shortest Positioning Time First (SPTF)** algorithm, also known as Shortest Access Time First (SATF), is a more advanced scheduling discipline that considers both seek and rotational delay. The scheduler chooses the request that has the shortest positioning time, which is the sum of seek time and rotational delay. This can lead to significant performance improvements over SSTF and SCAN.

6.2. Internal Disk Scheduling

- In modern drives, the disk scheduler is often implemented within the disk controller itself. This is because the controller has the most accurate information about the geometry of the platters and the current position of the head. This allows for more precise scheduling decisions, such as implementing SPTF effectively.

6.3. I/O Merging

- Another optimization that modern disk schedulers perform is **I/O merging**. If there are multiple requests to the same track, the scheduler can merge them into a single request, reducing the overhead of processing multiple requests.

6.4. Anticipatory Scheduling

- **Anticipatory scheduling** is a non-work-conserving scheduling algorithm. This means that the scheduler may decide to wait for a short period of time, even if there are requests in the queue. The idea is to “anticipate” that a better request might arrive soon. For example, if the scheduler has just serviced a request at a particular location on the disk, it might wait for a moment to see if another request for a nearby location arrives. This can be beneficial in situations where there are multiple processes accessing the disk, as it can help to reduce the number of seeks. “”