Chapter 10: Mass Storage Structure
Hard Disk Drives

Chen Hao
haochen@aimlab.org
Hunan University

- The hard disk drive have been the main form of persistent data storage in computer systems for decades and much of the development of file system technology is predicated on their behavior.

- Thus, it is worth understanding the details of a disk's operation before building the file system software that manages it.

CRUX: HOW TO STORE AND ACCESS DATA ON DISK

How do modern hard-disk drives store data? What is the interface? How is the data actually laid out and accessed? How does disk scheduling improve performance?

The Interface

- The drive consists of a large number of sectors (512-byte blocks), each of which can be read or written. The sectors are numbered from 0 to n – 1 on a disk with n sectors.

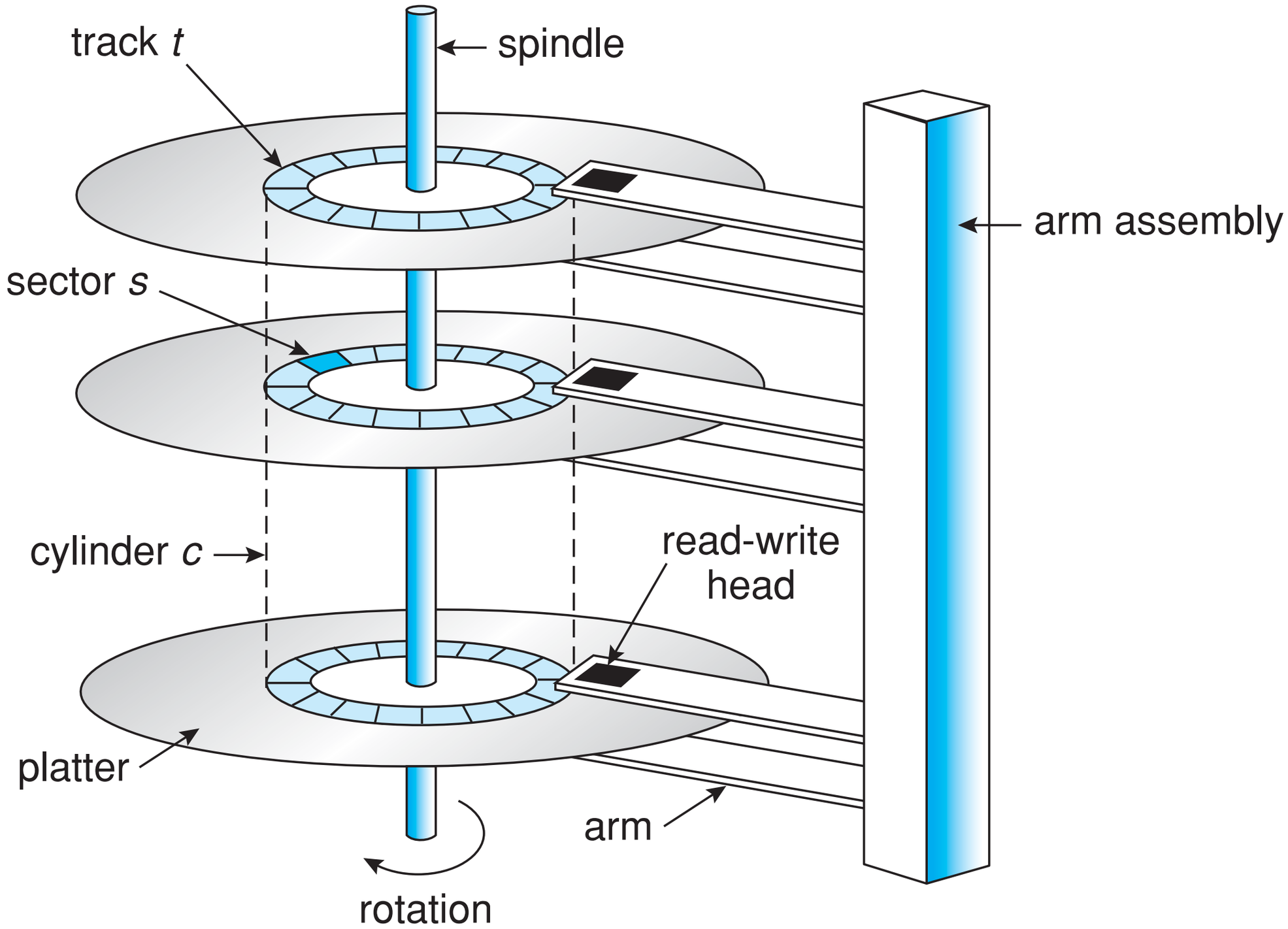
- Thus, we can view the disk as an array of sectors; 0 to n – 1 is thus the address space of the drive.

- Multi-sector operations are possible; indeed, many file systems will read or write 4KB at a time (or more).

- However, when updating the disk, the only guarantee drive manufactures make is that a single 512-byte write is atomic (i.e., it will either complete in its entirety or it won't complete at all).

- Thus, if an untimely power loss occurs, only a portion of a larger write may complete (sometimes called a torn write).

Basic Geometry



- A platter is a circular hard surface on which data is stored persistently by inducing magnetic changes to it.

- A disk may have one or more platters; each platter has 2 sides, each of which is called a surface.

- These platters are usually made of some hard material (such as aluminum), and then coated with a thin magnetic layer that enables the drive to persistently store bits even when the drive is powered off.

- The platters are all bound together around the spindle, which is connected to a motor that spins the platters around (while the drive is powered on) at a constant (fixed) rate.

- The rate of rotation is often measured in rotations per minute (RPM), and typical modern values are in the 7,200 RPM to 15,000 RPM range.

- Note that we will often be interested in the time of a single rotation, e.g., a drive that rotates at 10,000 RPM means that a single rotation takes about 6 milliseconds (6 ms).

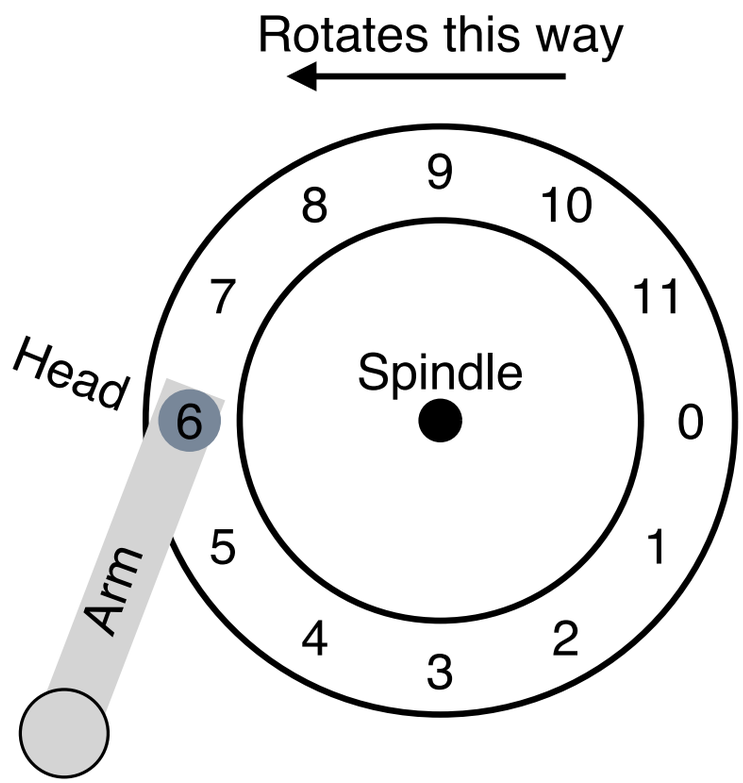
- Data is encoded on each surface in concentric circles of sectors; we call one such concentric circle a track.

- A single surface contains many thousands and thousands of tracks, tightly packed together, with hundreds of tracks fitting into the width of a human hair.

- This process of reading and writing is accomplished by the disk head; there is one such head per surface of the drive.

- The disk head is attached to a single disk arm, which moves across the surface to position the head over the desired track.

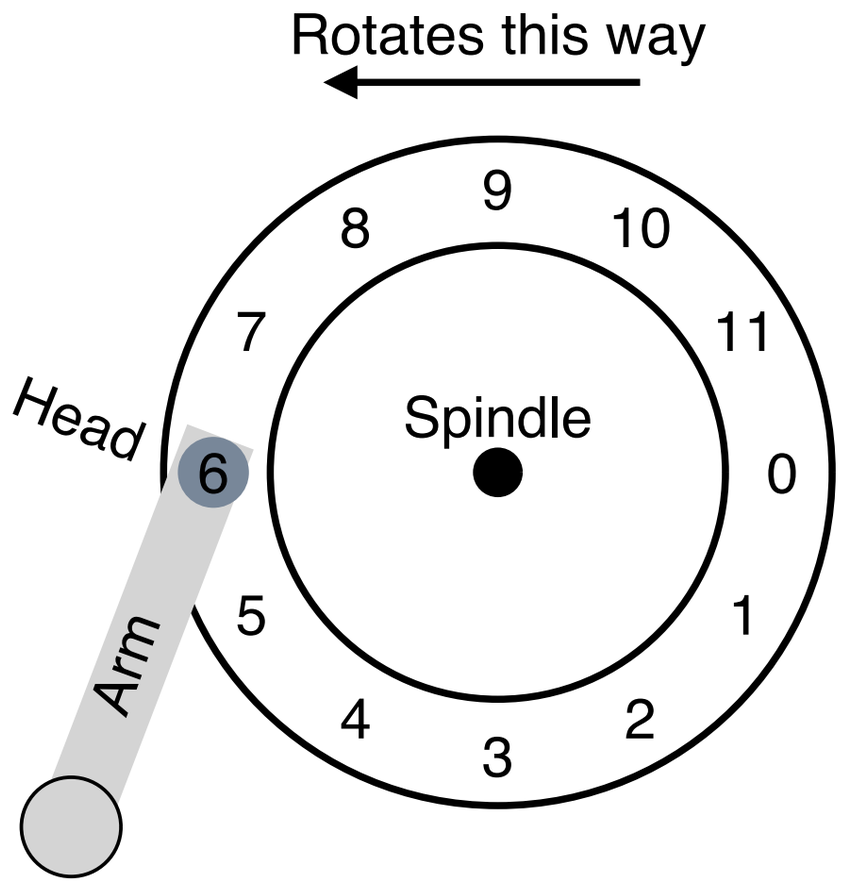
A Simple Disk Drive



- This track has just 12 sectors, each of which is 512 bytes in size (our typical sector size, recall) and addressed therefore by the numbers 0 through 11.

- In the figure, the disk head, attached to the end of the arm, is positioned over sector 6, and the surface is rotating counter-clockwise.

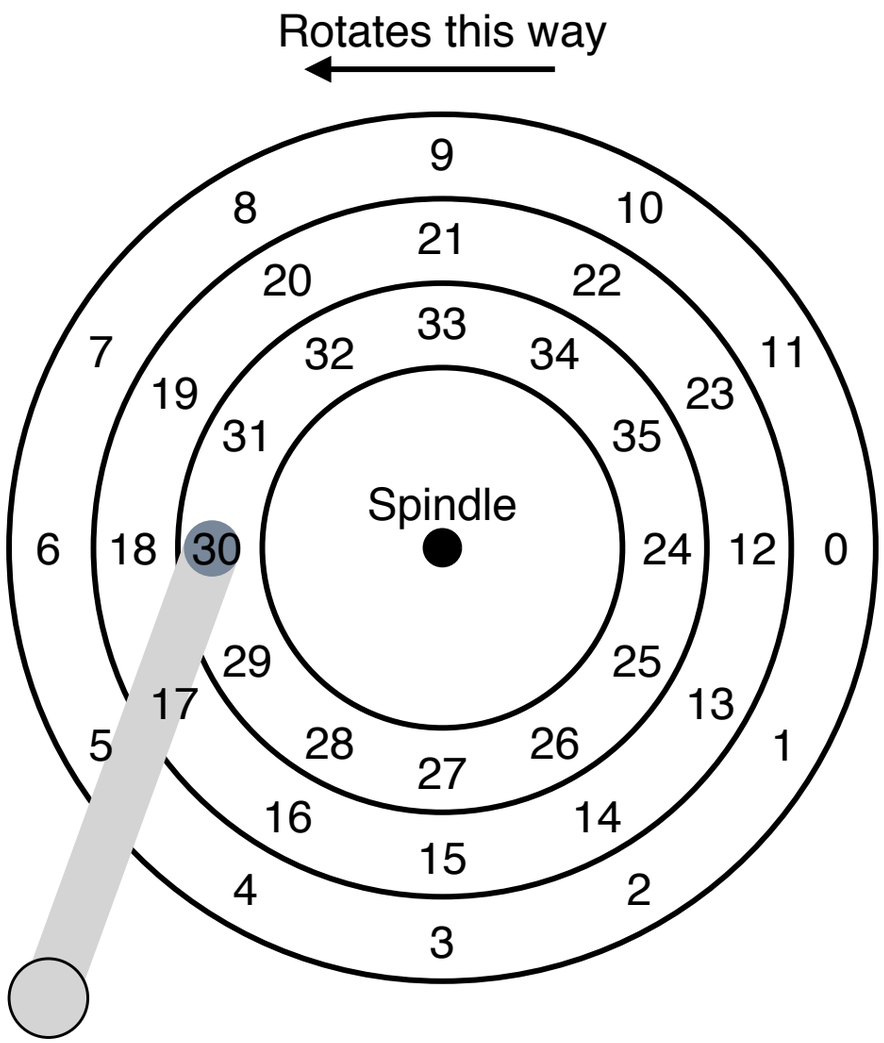
The Rotational Delay

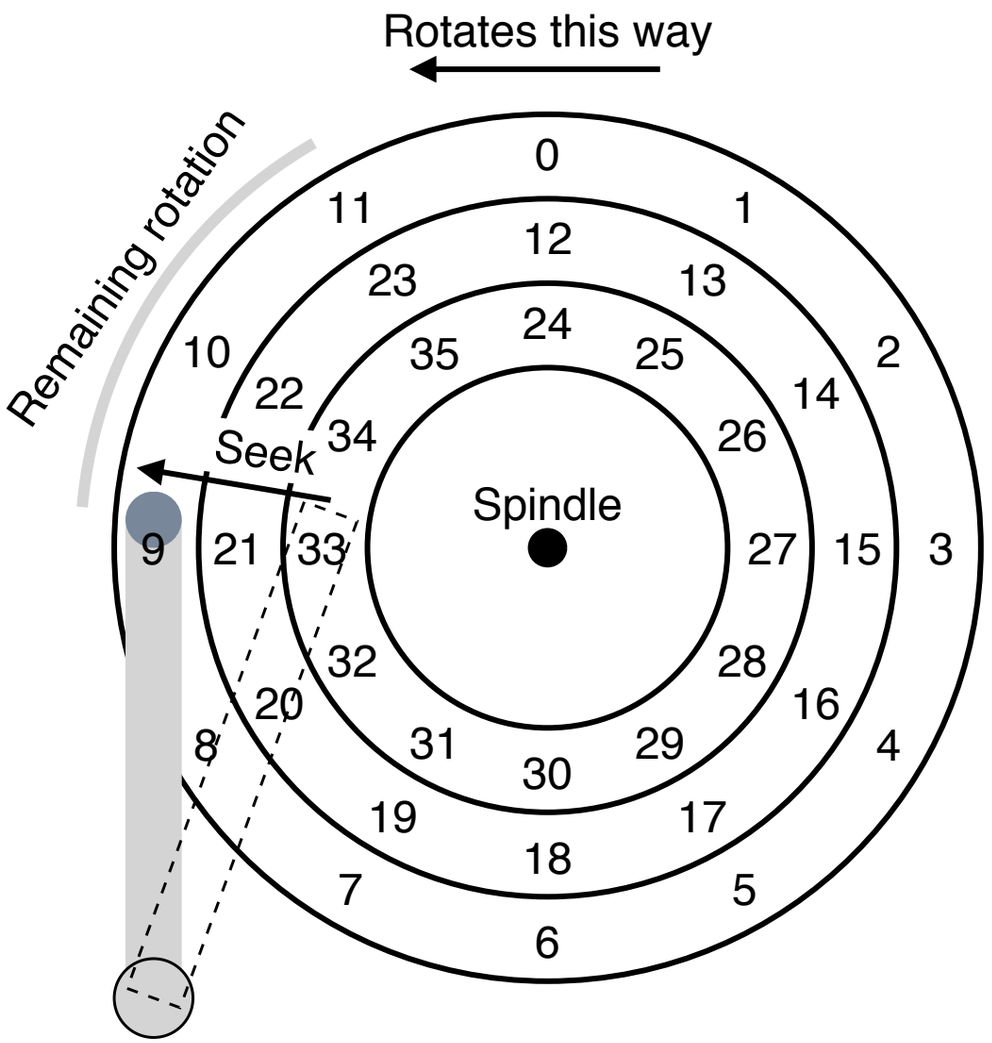


- In the example, if the full rotational delay is R, the disk has to incur a rotational delay of about R/2 to wait for 0 to come under the read/write head (if we start at 6).

- A worst-case request on this single track would be to sector 5, causing nearly a full rotational delay in order to service such a request.

Seek Time





- We can trace what would happen on a request to a distant sector, e.g., a read to sector 11.

- The seek, it should be noted, has many phases.
- First, an *acceleration* phase as the disk arm gets moving;
- Then, *coasting* as the arm is moving at full speed, then *deceleration* as the arm slows down;
- Finally, *settling* as the head is carefully positioned over the correct track. The settling time is often quite significant, e.g., 0.5 to 2 ms, as the drive must be certain to find the right track (imagine if it just got close instead!).

- And thus, we have a complete picture of I/O time: first a seek, then waiting for the rotational delay, and finally the transfer.

Some Other Details

- An important part of any modern disk drive is its cache, for historical reasons sometimes called a track buffer.

- This cache is just some small amount of memory (usually around 8 or 16 MB) which the drive can use to hold data read from or written to the disk.

- On writes, the drive has a choice: should it acknowledge the write has completed when it has put the data in its memory, or after the write has actually been written to disk? The former is called write back caching, and the latter write through.

I/O Time: Doing The Math

- We can now represent I/O time as the sum of three major components:

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

- Note that the rate of I/O ($R_{I/O}$), which is often more easily used for comparison between drives (as we will do below), is easily computed from the time. Simply divide the size of the transfer by the time it took:

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

- Assume there are two workloads we are interested in.

- The first, known as the random workload, issues small (e.g., 4KB) reads to random locations on the disk. Random workloads are common in many important applications, including database management systems.

- The second, known as the sequential workload, simply reads a large number of sectors consecutively from the disk (e.g. 100MB), without jumping around.

- To understand the difference in performance between random and sequential workloads, we need to make a few assumptions about the disk drive first.

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Disk Drive Specs: SCSI Versus SATA

- From these numbers, we can start to calculate how well the drives would do under our two workloads outlined above. (4 KB random read, 100 MB sequential read)

	Cheetah	Barracuda
$R_{I/O}$ Random	0.66 MB/s	0.31 MB/s
$R_{I/O}$ Sequential	125 MB/s	105 MB/s

- The table shows us a number of important things. First, and most importantly, there is a huge gap in drive performance between random and sequential workloads, almost a factor of 200 or so for the Cheetah and more than a factor 300 difference for the Barracuda.

Tip: Use Disks Sequentially

When at all possible, transfer data to and from disks in a sequential manner. If sequential is not possible, at least think about transferring data in large chunks: the bigger, the better. If I/O is done in little random pieces, I/O performance will suffer dramatically. Also, users will suffer. Also, you will suffer, knowing what suffering you have wrought with your careless random I/Os.

Disk Scheduling

- Because of the high cost of I/O, the OS has historically played a role in deciding the order of I/Os issued to the disk.

- More specifically, given a set of I/O requests, the disk scheduler examines the requests and decides which one to schedule next.

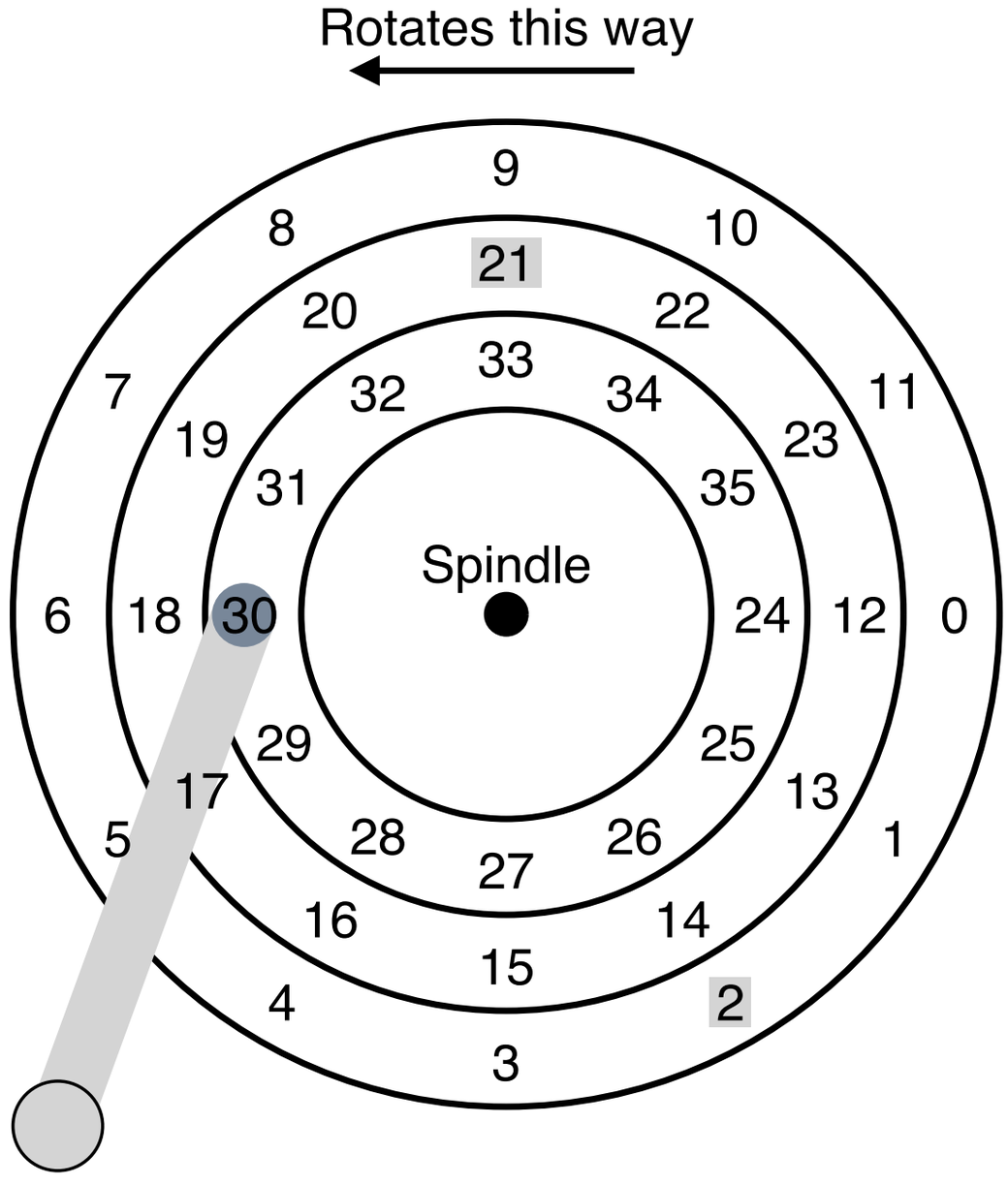
- Unlike job scheduling, where the length of each job is usually unknown, with disk scheduling, we can make a good guess at how long a "job" (i.e., disk request) will take.

- By estimating the seek and possible the rotational delay of a request, the disk scheduler can know how long each request will take, and thus (greedily) pick the one that will take the least time to service first.

- Thus, the disk scheduler will try to follow the principle of SJF (shortest job first) in its operation.

SSTF: Shortest Seek Time First

- SSTF orders the queue of I/O requests by track, picking requests on the nearest track to complete first.
- For example, assuming the current position of the head is over the inner track, and we have requests for sectors 21 (middle track) and 2 (outer track), we would then issue the request to 21 first, wait for it to complete, and then issue the request to 2



SSTF: Scheduling Requests 21 And 2

Problems with SSTF

Problems with SSTF

- First, the drive geometry is not available to the host OS; rather, it sees an array of blocks. Fortunately, this problem is rather easily fixed. Instead of SSTF, an OS can simply implement nearest-block-first (NBF), which schedules the request with the nearest block address next.

Problems with SSTF

- First, the drive geometry is not available to the host OS; rather, it sees an array of blocks. Fortunately, this problem is rather easily fixed. Instead of SSTF, an OS can simply implement nearest-block-first (NBF), which schedules the request with the nearest block address next.

- The second problem is more fundamental: starvation. Imagine in our example above if there were a steady stream of requests to the inner track, where the head currently is positioned. Requests to any other tracks would then be ignored completely by a pure SSTF approach.

Crux: How To Handle Disk Starvation

How can we implement SSTF-like scheduling but avoid starvation?

Elevator (a.k.a. SCAN or C-SCAN)

- The algorithm, originally called SCAN, simply moves across the disk servicing requests in order across the tracks.

- Let us call a single pass across the disk a *sweep*. Thus, if a request comes for a block on a track that has already been serviced on this sweep of the disk, it is not handled immediately, but rather queued until the next sweep.

- SCAN has a number of variants.

- F-SCAN, which freezes the queue to be serviced when it is doing a sweep; this action places requests that come in during the sweep into a queue to be serviced later.

- Doing so avoids starvation of far-away requests, by delaying the servicing of late-arriving (but nearer by) requests.

- C-SCAN is another common variant, short for Circular SCAN.
- Instead of sweeping in one direction across the disk, the algorithm sweeps from outer-to-inner, and then inner-to-outer, etc.

- This algorithm (and its variants) is sometimes referred to as the elevator algorithm, because it behaves like an elevator which is either going up or down and not just servicing requests to floors based on which floor is closer.

- Imagine how annoying it would be if you were going down from floor 10 to 1, and somebody got on at 3 and pressed 4, and the elevator went up to 4 because it was "closer" than 1!

- As you can see, the elevator algorithm, when used in real life, prevents fights from taking place on elevators. In disks, it just prevents starvation.

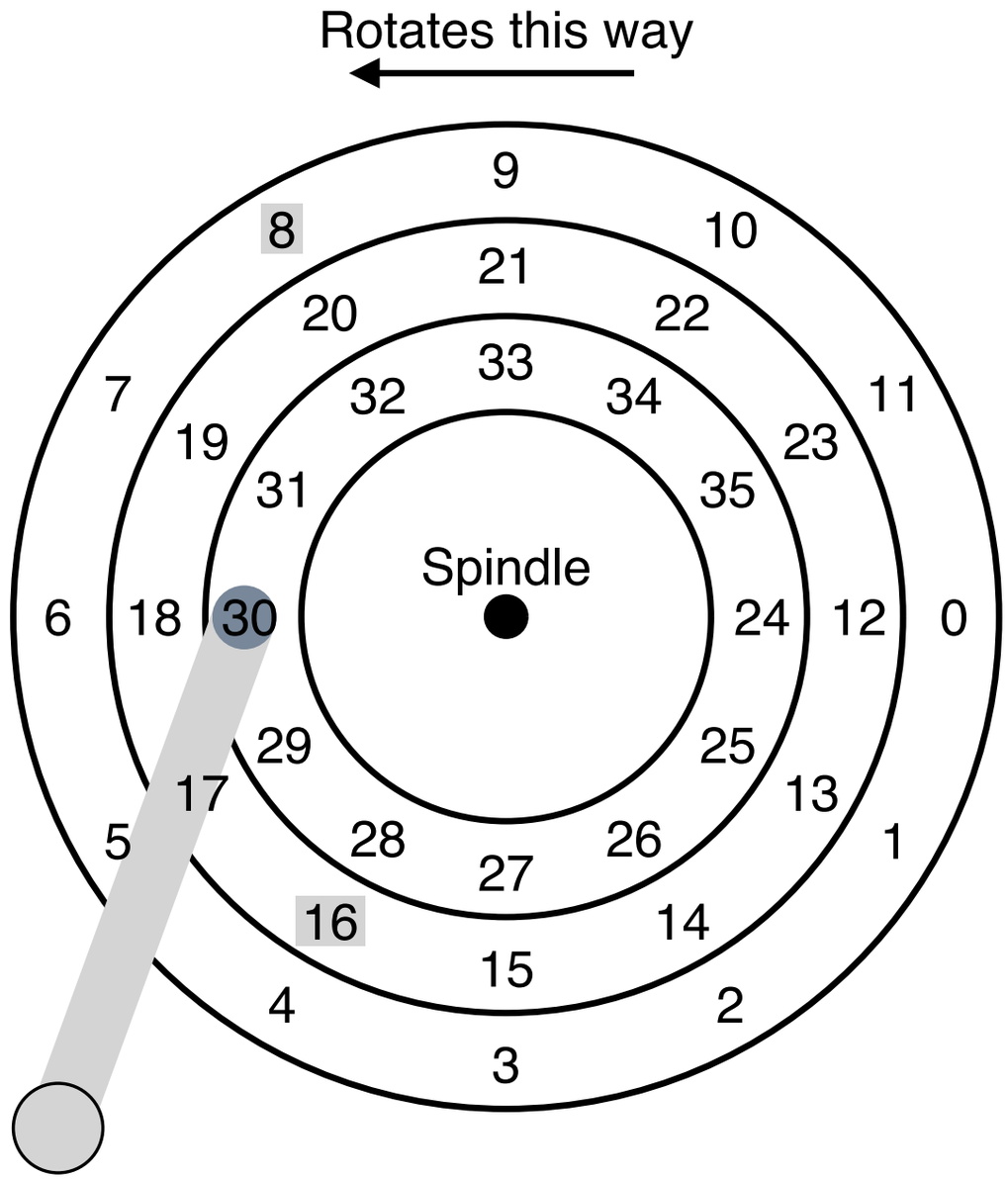
- Unfortunately, SCAN and its variants do not represent the best scheduling technology.
- In particular, SCAN (or SSTF even) do not actually adhere as closely to the principle of SJF as they could. In particular, they ignore rotation.

Crux: How To Account For Disk Rotation Costs

How can we implement an algorithm that more closely approximates SJF by taking *both* seek and rotation into account?

SPTF: Shortest Positioning Time First

- Before discussing shortest positioning time first or SPTF scheduling (sometimes also called shortest access time first or SATF), which is the solution to our problem, let us make sure we understand the problem in more detail.



- If seek time is much higher than rotational delay, then SSTF (and variants) are just fine.

- However, on modern drives, both seek and rotation are roughly equivalent (depending on the exact requests), and thus SPTF is useful and improves performance.

- It is difficult to implement in an OS, which generally does not have a good idea where track boundaries are or where the disk head currently is. Thus, SPTF is usually performed inside a drive.

Other Scheduling Issues

- In older systems, the operating system did all the scheduling.
- In modern systems, disks have sophisticated internal schedulers themselves. Thus, the OS scheduler usually picks what it thinks the best few requests are and issues them all to disk; the disk then uses its internal knowledge to service said requests in the best possible (SPTF) order.