Operating System Concepts

Lecture 33: Disk Structure and Scheduling

Omid Ardakanian
oardakan@ualberta.ca
University of Alberta

MWF 12:00-12:50 VVC 2 215

Today’s class

- Storage structure
  - Disk
  - NVM
- Disk scheduling

What's secondary storage?

- secondary storage devices permanently store files/data
  - the slowest major component of computers

What’s secondary storage?

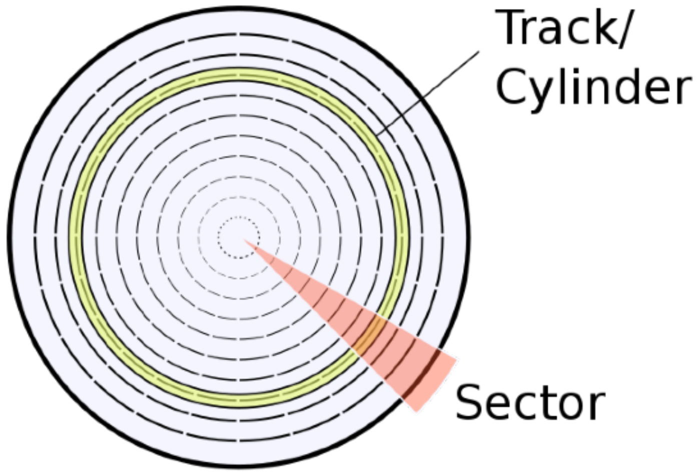
- secondary storage devices permanently store files/data
  - the slowest major component of computers
- two main types of secondary storage are hard disk drives (HDDs) and nonvolatile memory (NVM) devices
  - we use the term nonvolatile storage (NVS) to talk about all types of devices used for persistent data storage

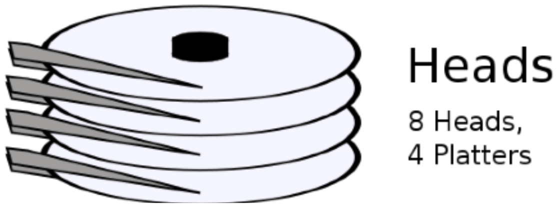
What's secondary storage?

- secondary storage devices permanently store files/data
  - the slowest major component of computers
- two main types of secondary storage are hard disk drives (HDDs) and nonvolatile memory (NVM) devices
  - we use the term nonvolatile storage (NVS) to talk about all types of devices used for persistent data storage
- to a file system, disk looks like a linear array of blocks that can be read from or written to

Basic geometry

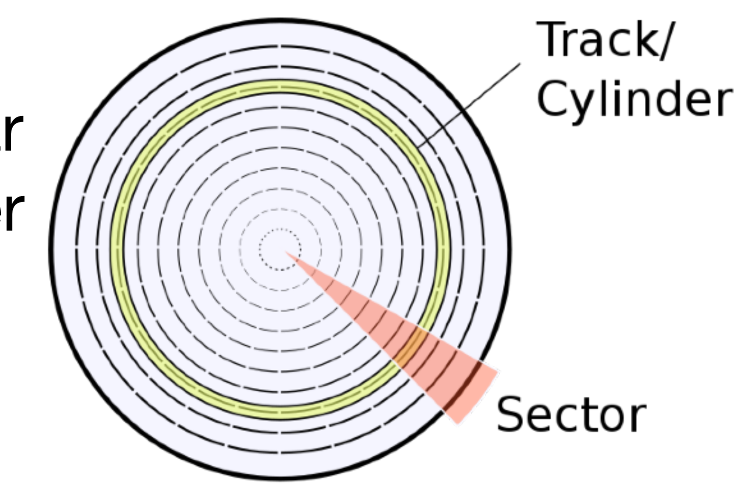
- disk is a sealed pack of platters, each having a flat circular shape and two **surfaces** coated with a thin magnetic layer
  - writing data: inducing magnetic changes on platters
  - reading data: detecting magnetic patterns on platters

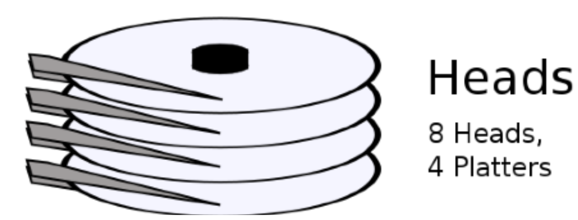




Basic geometry

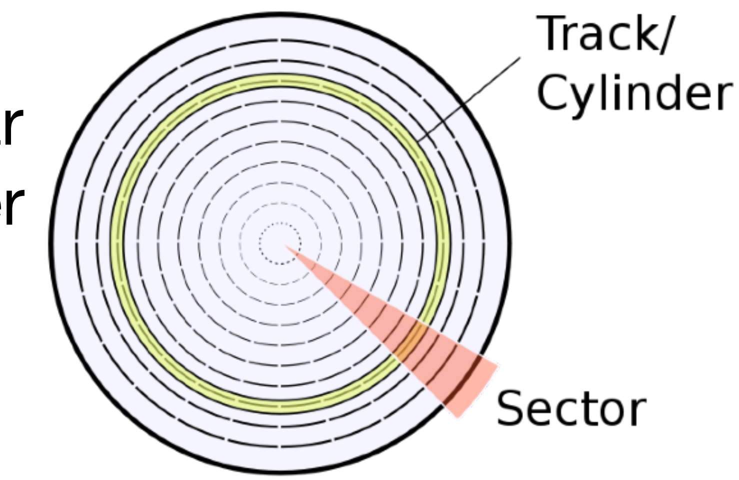
- disk is a sealed pack of platters, each having a flat circular shape and two **surfaces** coated with a thin magnetic layer
  - writing data: inducing magnetic changes on platters
  - reading data: detecting magnetic patterns on platters
- platter's surface is divided into many thousands of circular tracks
  - all tracks (on different platters) at a given arm position make up a cylinder; cylinders are equidistant from the disk centre

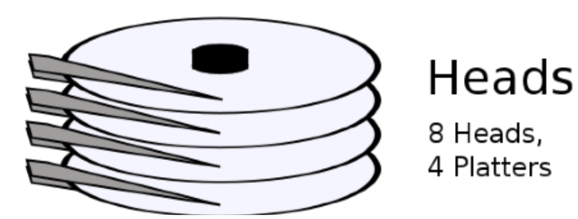




Basic geometry

- disk is a sealed pack of platters, each having a flat circular shape and two **surfaces** coated with a thin magnetic layer
  - writing data: inducing magnetic changes on platters
  - reading data: detecting magnetic patterns on platters
- platter's surface is divided into many thousands of circular tracks
  - all tracks (on different platters) at a given arm position make up a cylinder; cylinders are equidistant from the disk centre
- tracks are divided into several hundreds of sectors (aka blocks) which are fixed-size array of bytes
  - sectors are the smallest unit of transfer and only a single sector read/ write is atomic
  - sector size was 512 bytes until ~2010 and then increased to 4KB
  - outer tracks tend to have more sectors than inner tracks





Seek and rotation

- read-write heads are attached to disk arms; a disk arm moves across the surface of a platter to position the head over the desired track
  - there is one head per surface
  - head will sometimes damage the magnetic surface (**head crash**); resulting in defective/bad sectors and loss of data

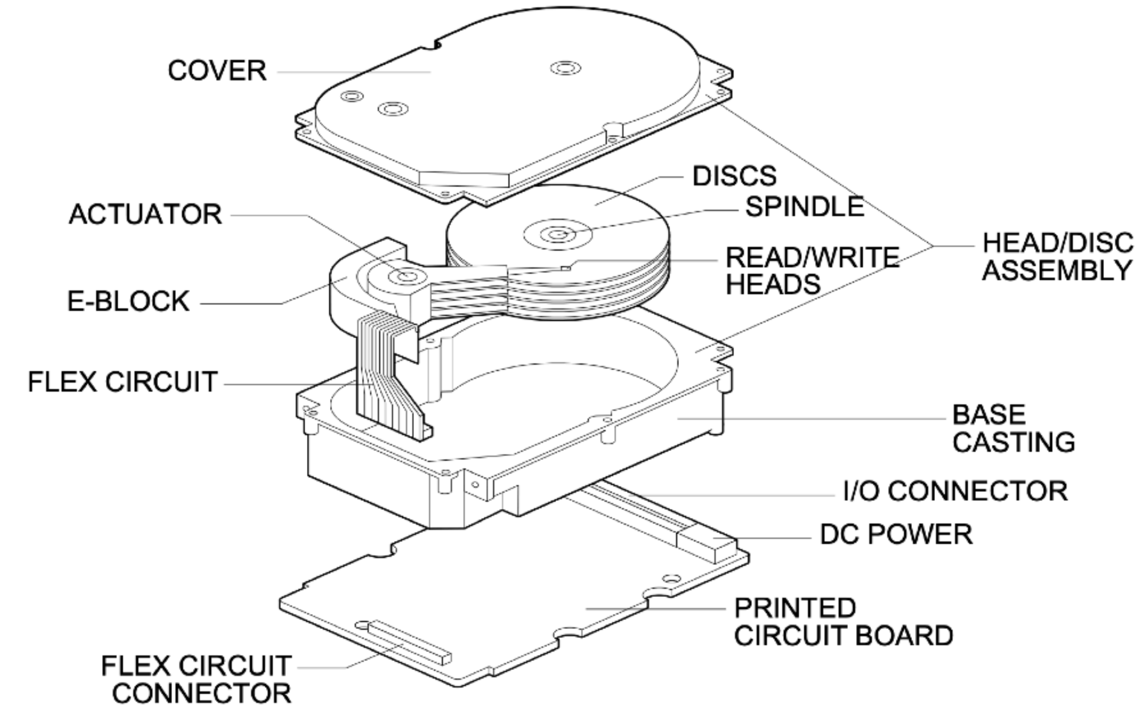


image from "Dave Anderson, Jim Dykes, and Erik Riedel. 2003. More Than an Interface-SCSI vs. ATA. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03). USENIX Association, Berkeley, CA, USA, 245-257."

Seek and rotation

- read-write heads are attached to disk arms; a disk arm moves across the surface of a platter to position the head over the desired track
  - there is one head per surface
  - head will sometimes damage the magnetic surface (**head crash**); resulting in defective/bad sectors and loss of data
- a spindle connected to a motor spins platters around at a constant rate
  - disk rotates at high speed: 5,400 to 15,000 times per minute (RPM) so a single rotation takes a few milliseconds
  - rotation speed determines the transfer rate

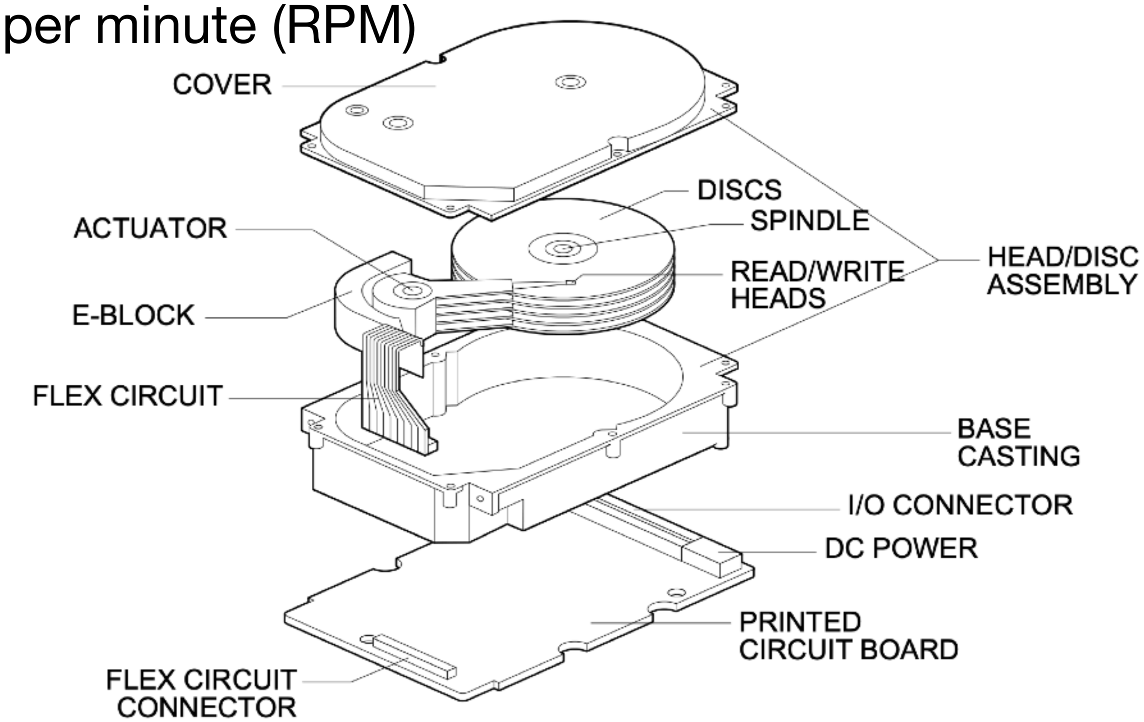


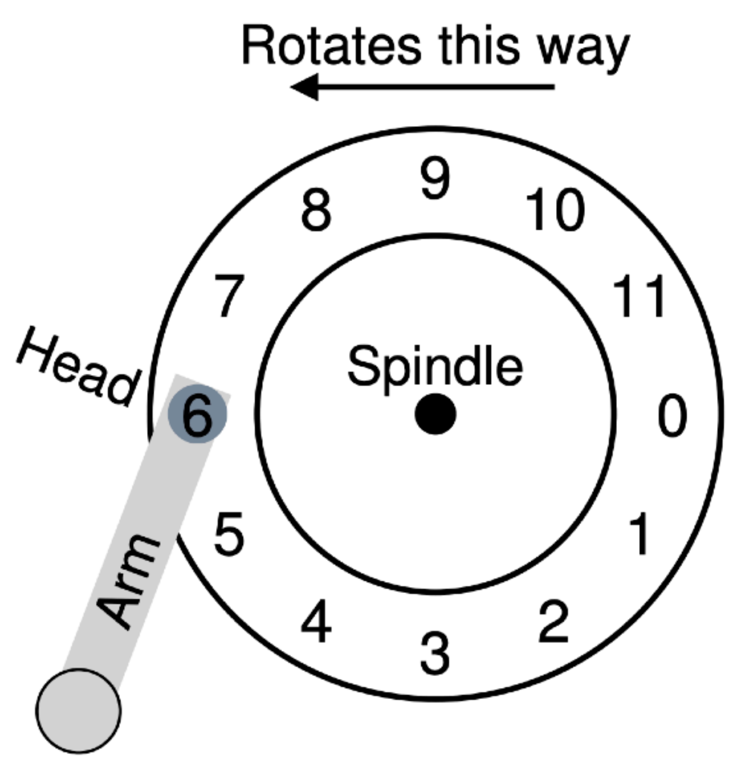
image from "Dave Anderson, Jim Dykes, and Erik Riedel. 2003. More Than an Interface-SCSI vs. ATA. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03). USENIX Association, Berkeley, CA, USA, 245-257."

Unwritten contract

- accessing two nearby disk blocks is usually faster than accessing two blocks that are far apart
  - may need fewer head movements
- accessing disk blocks sequentially is typically much faster than any random access pattern
  - may need fewer head movements

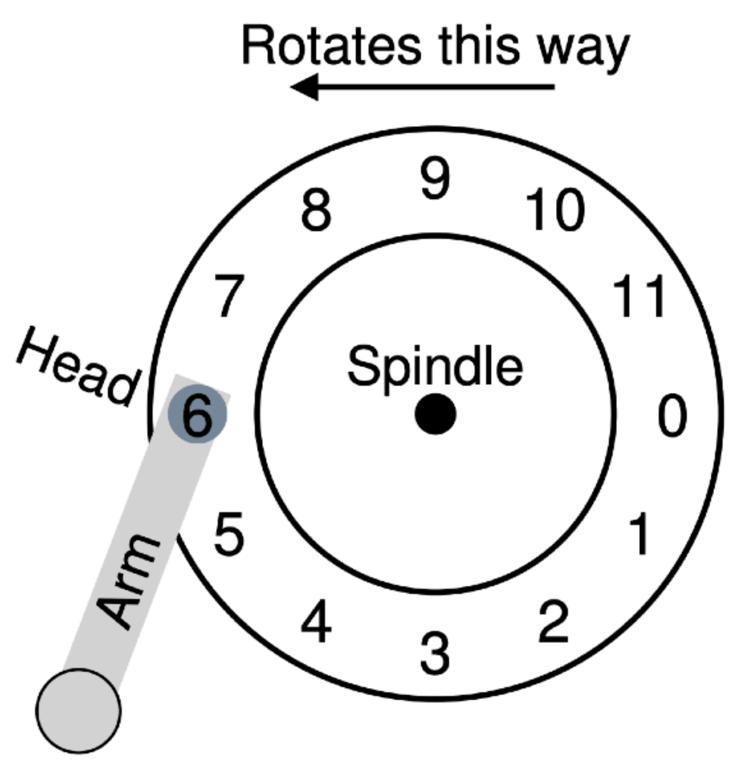
Understanding disk performance: rotation

- rotational delay: time necessary for the spindle to rotate the desired sector to the disk head
  - depends on how fast the disk spins



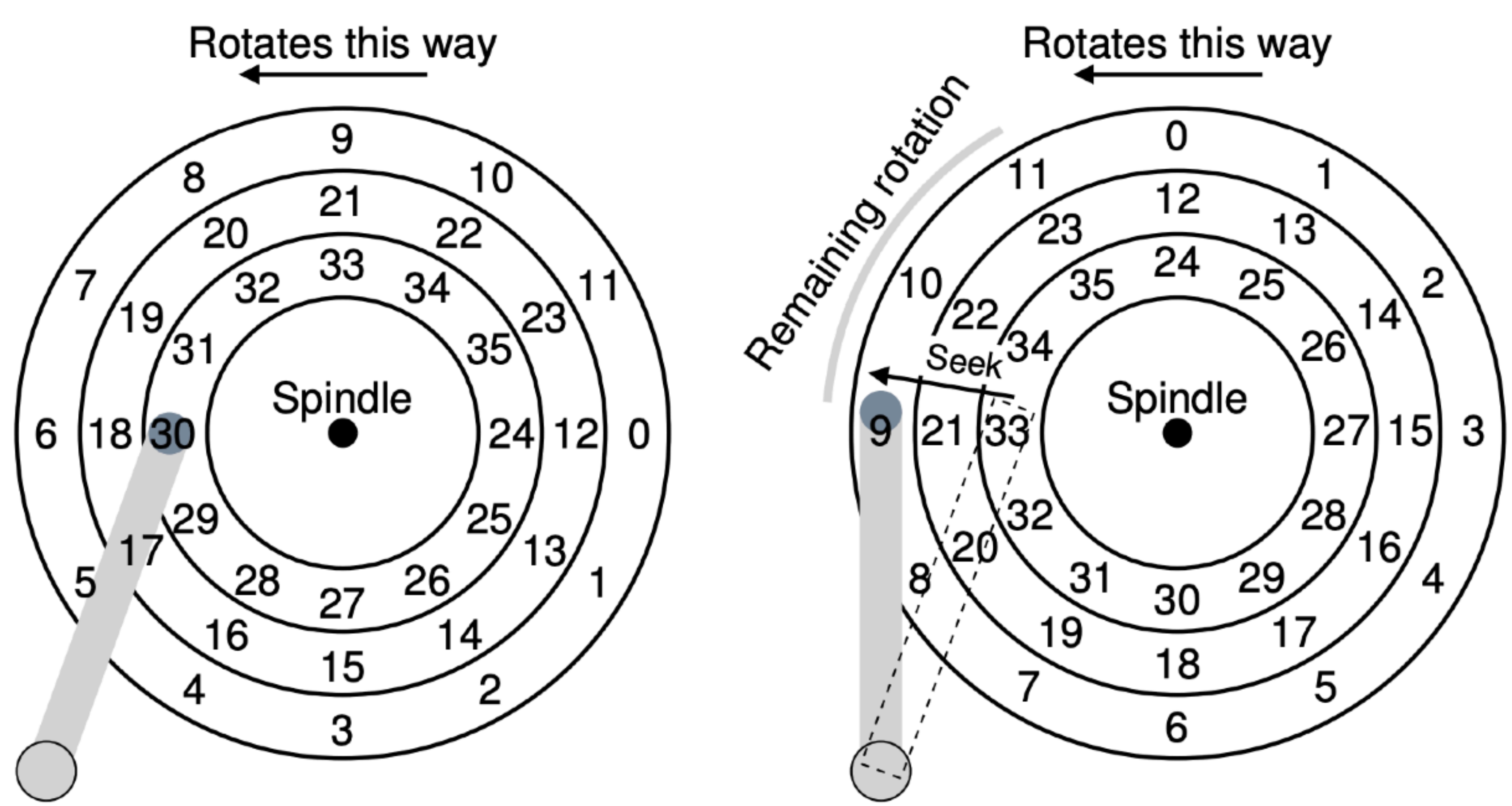
Understanding disk performance: rotation

- rotational delay: time necessary for the spindle to rotate the desired sector to the disk head
  - depends on how fast the disk spins
- example: disk head is currently positioned over sector 6
  - average case: if rotational delay is R, the disk has to incur a rotational delay of about R/2 to wait for sector 0 to come under the read/write head
  - worst case: wait for sector 5 to come under the read/write head (almost one full rotation)



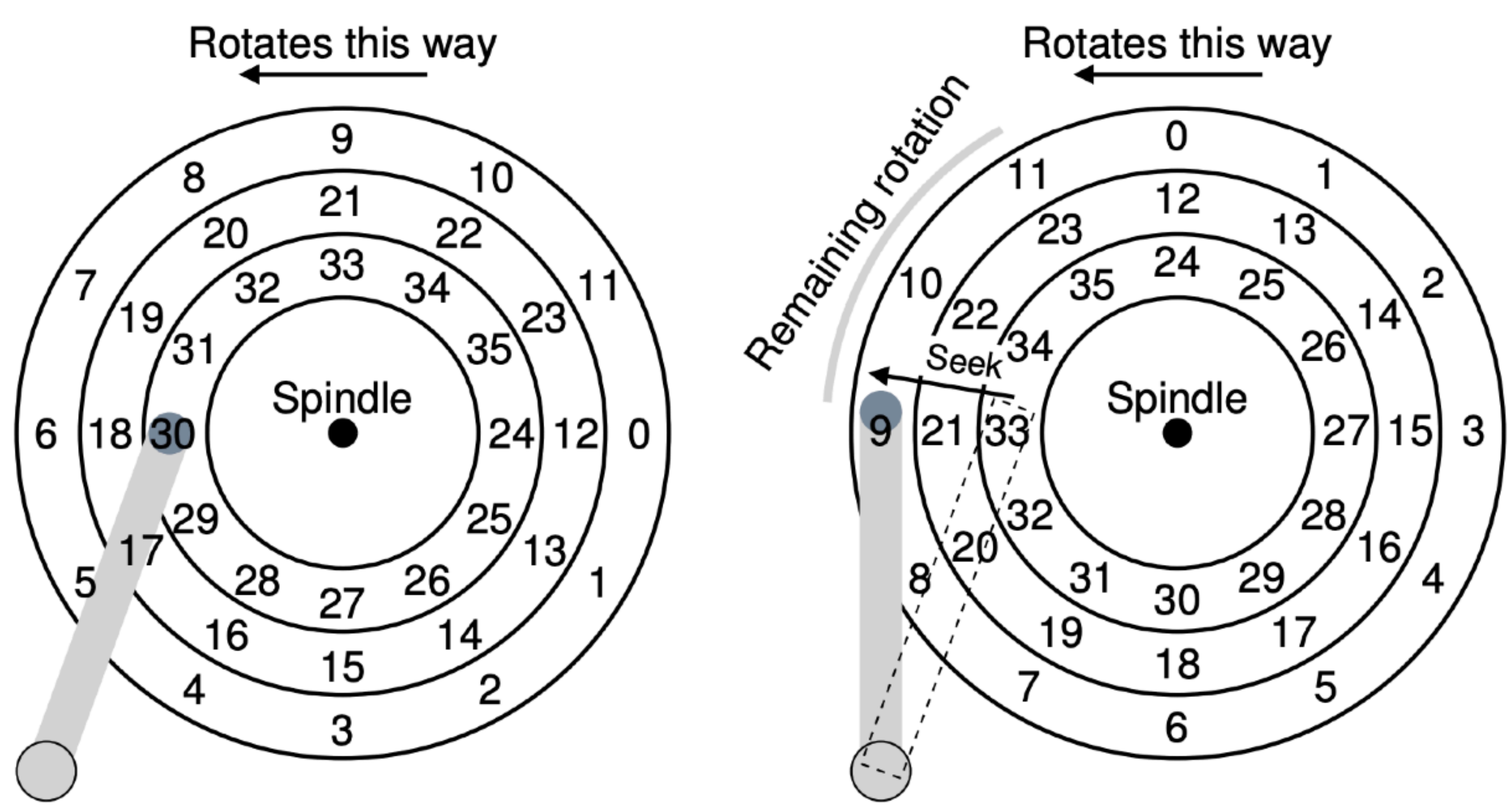
Understanding disk performance: disk seek

- seek time: time to move the disk arm to the desired cylinder/track (between 3ms and 12ms)
  - depends on how fast the arm moves



Understanding disk performance: disk seek

- seek time: time to move the disk arm to the desired cylinder/track (between 3ms and 12ms)
  - depends on how fast the arm moves
- example: servicing a request for logical block 11
  - disk arm has to move to the outermost cylinder/track
  - during the seek, the arm has been moved to the desired track, and the platter of course has rotated (about 3 sectors in this example)
  - still need to wait until the disk spins and the desired sector comes under the disk head



Disk I/O time

- positioning time (aka random-access time): seek time + rotational latency
  - usually several milliseconds

Disk I/O time

- positioning time (aka random-access time): seek time + rotational latency
  - usually several milliseconds
- transfer rate (bandwidth): data flow rate between disk and computer
  - typically hundreds of megabytes per second

Disk I/O time

- positioning time (aka random-access time): seek time + rotational latency
  - usually several milliseconds
- transfer rate (bandwidth): data flow rate between disk and computer
  - typically hundreds of megabytes per second
- transfer time = (amount of data to transfer)/transfer rate
  - for example if transfer rate is 100MB/second, it would take 5ms to transfer 512KB of data; hence 5ms is the transfer time

Disk I/O time

- positioning time (aka random-access time): seek time + rotational latency
  - usually several milliseconds
- transfer rate (bandwidth): data flow rate between disk and computer
  - typically hundreds of megabytes per second
- transfer time = (amount of data to transfer)/transfer rate
  - for example if transfer rate is 100MB/second, it would take 5ms to transfer 512KB of data; hence 5ms is the transfer time
- avg. disk I/O time = avg. positioning time + transfer time (+ controller overhead)

Disk I/O time

- positioning time (aka random-access time): seek time + rotational latency
  - usually several milliseconds
- transfer rate (bandwidth): data flow rate between disk and computer
  - typically hundreds of megabytes per second
- transfer time = (amount of data to transfer)/transfer rate
  - for example if transfer rate is 100MB/second, it would take 5ms to transfer 512KB of data; hence 5ms is the transfer time
- avg. disk I/O time = avg. positioning time + transfer time (+ controller overhead)
- disk I/O rate is the amount of data to be transferred divided by the disk I/O time

Example

what is the average I/O time to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, a 0.1ms controller overhead, and 1Gb/s transfer rate?

$$avg.\ rotational\ delay = \frac{1}{2 \times \frac{7200\ rotation}{1\ min} \times \frac{1\ min}{60{,}000\ ms}} = 4.17\ ms$$

$$avg.\ seek\ delay = 5\ ms$$

$$controller\ overhead = 0.1\ ms$$

$$transfer\ time = \frac{4KB}{\frac{1\ Gb}{s} \times \frac{1\ GB}{8\ Gb} \times \frac{1{,}024 \times 1{,}024\ KB}{1\ GB} \times \frac{1\ s}{1{,}000\ ms}} = 0.03\ ms$$

$$avg.\ I/O\ time = 4.17\ ms + 5\ ms + 0.1\ ms + 0.03\ ms = 9.30\ ms$$

Other considerations

- HDDs usually have some small amount of memory (8 to 16 MB) referred to as cache or track buffer which holds data read from or written to the disk
  - e.g., the drive may read more than one sector on a given track and cache it in its memory (faster response to subsequent requests for sectors on the same track)

Other considerations

- HDDs usually have some small amount of memory (8 to 16 MB) referred to as cache or track buffer which holds data read from or written to the disk
  - e.g., the drive may read more than one sector on a given track and cache it in its memory (faster response to subsequent requests for sectors on the same track)
- **write back**: the request to write data can be acknowledged right away by putting the data in cache and writing to specific sectors at a later time
  - response appears to be faster

Other considerations

- HDDs usually have some small amount of memory (8 to 16 MB) referred to as cache or track buffer which holds data read from or written to the disk
  - e.g., the drive may read more than one sector on a given track and cache it in its memory (faster response to subsequent requests for sectors on the same track)
- **write back**: the request to write data can be acknowledged right away by putting the data in cache and writing to specific sectors at a later time
  - response appears to be faster
- **write through**: the request to write data is acknowledged only after the data is actually written to disk

How to reduce disk access time?

- sequential I/O is optimal on HDD
  - disk seek is necessary only when reaching the end of a track and have to move head to the beginning of the next track
  - random access I/O causes disk head movement and is much slower than sequential I/O

How to reduce disk access time?

- sequential I/O is optimal on HDD
  - disk seek is necessary only when reaching the end of a track and have to move head to the beginning of the next track
  - random access I/O causes disk head movement and is much slower than sequential I/O
- how to minimize disk I/O time?
  1. make disks smaller and spin faster
  2. deciding the order in which I/O requests are issued to disk (minimize head movement)
     - disk scheduler can make a good guess at how long servicing a specific I/O request will take
  3. lay out data on disk so that related data are on nearby tracks
     - e.g., put file contents near its metadata
  4. choose sector size more carefully
     - smaller sectors means more disk seeks for the same amount of data

Nonvolatile memory

- nonvolatile memory (NVM) devices are electrical rather than mechanical
  - so there is no disk head
  - random-access I/O is much faster on NVM compared to HDD
  - writing to NVM is slower than reading (blocks have to be erased first)

Nonvolatile memory

- nonvolatile memory (NVM) devices are electrical rather than mechanical
  - so there is no disk head
  - random-access I/O is much faster on NVM compared to HDD
  - writing to NVM is slower than reading (blocks have to be erased first)
- two main types of NVM are SSD and NAND semiconductors

Nonvolatile memory

- nonvolatile memory (NVM) devices are electrical rather than mechanical
  - so there is no disk head
  - random-access I/O is much faster on NVM compared to HDD
  - writing to NVM is slower than reading (blocks have to be erased first)
- two main types of NVM are SSD and NAND semiconductors
- solid-state disks (SSD) are built out of transistors
  - USB drives (aka thumb/flash drive) are similar
  - more reliable than HDDs because they have no moving parts
  - faster than HDDs because they have no seek time to rotational latency
  - more energy efficient than HDDs

NAND semiconductors

- NAND semiconductors are organized into banks
  - within each bank there are a large number of blocks; within each block there are a large number of pages

NAND semiconductors

- NAND semiconductors are organized into banks
  - within each bank there are a large number of blocks; within each block there are a large number of pages
- data cannot be overwritten so to write to a page within a block, you first have to erase the entire block first
  - invalid pages: NAND pages (like disk blocks) containing old data that must be erased before they can be rewritten
  - flash translation layer tracks which physical pages contain valid or invalid data at a given time
  - erasure time >> program (write) time > read time

NAND semiconductors

- NAND semiconductors are organized into banks
  - within each bank there are a large number of blocks; within each block there are a large number of pages
- data cannot be overwritten so to write to a page within a block, you first have to erase the entire block first
  - invalid pages: NAND pages (like disk blocks) containing old data that must be erased before they can be rewritten
  - flash translation layer tracks which physical pages contain valid or invalid data at a given time
  - erasure time >> program (write) time > read time
- lifetime is approx. 100,000 program erase cycles (due to deterioration)
  - lifespan is measured in Drive Writes Per Day (DWPD)
  - a 1TB NAND drive rated at 5 DWPD is expected to have 5TB per day data written for it for the warranty period without failure
  - frequently erased block wear faster, hence controller does wear levelling by erasing and overwriting less erased pages

Disk Scheduling

HDD scheduling

- if the desired drive and controller are available, the I/O request (made by a system call) can be serviced immediately

HDD scheduling

- if the desired drive and controller are available, the I/O request (made by a system call) can be serviced immediately
- otherwise, the request is added to a queue of pending requests for that drive

HDD scheduling

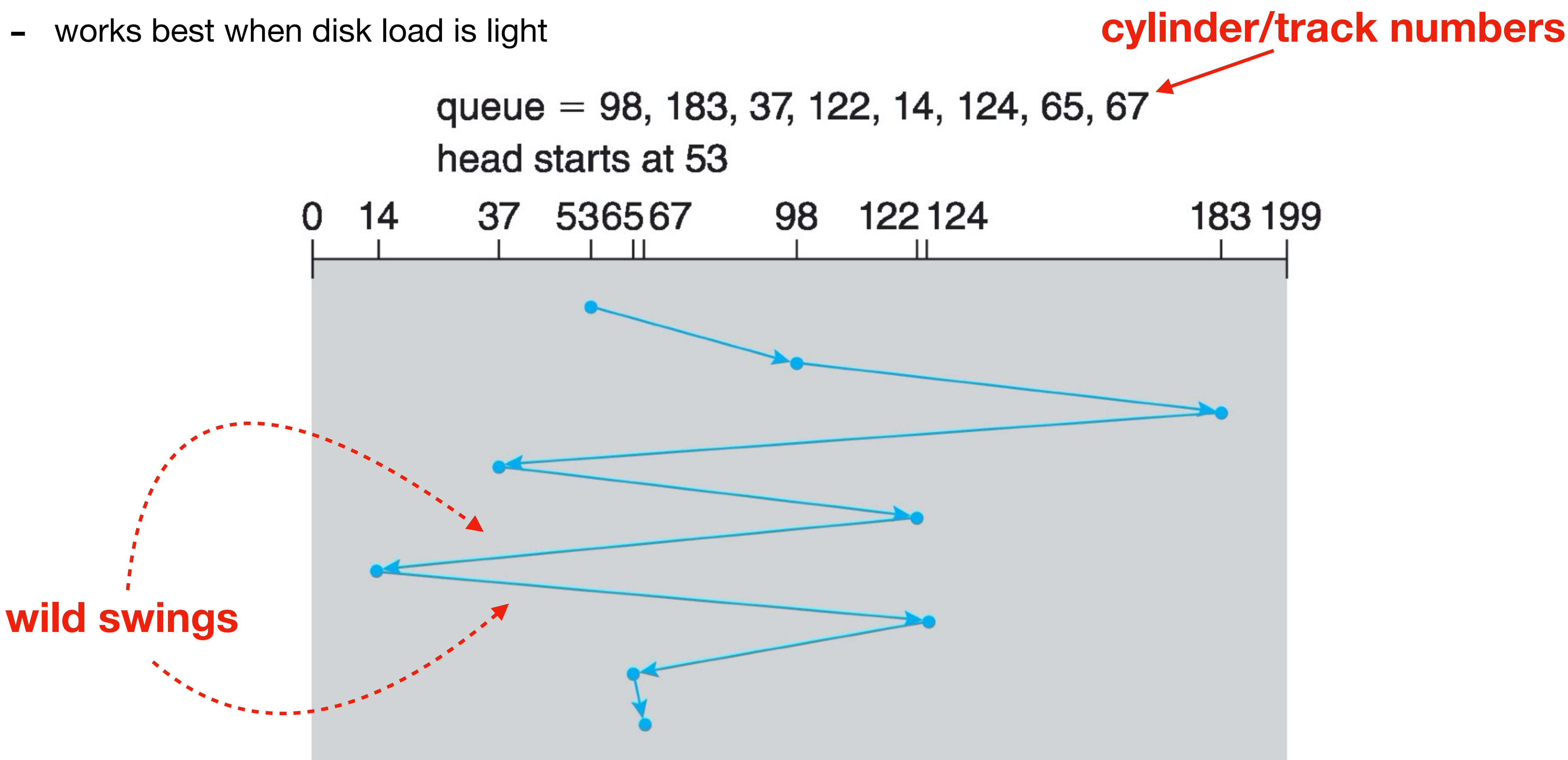
- if the desired drive and controller are available, the I/O request (made by a system call) can be serviced immediately
- otherwise, the request is added to a queue of pending requests for that drive
- the order of requests in the queue can be changed to avoid disk-head seeks
  - **Goal:** optimize/balance fairness, timeliness, and performance

HDD scheduling

- if the desired drive and controller are available, the I/O request (made by a system call) can be serviced immediately
- otherwise, the request is added to a queue of pending requests for that drive
- the order of requests in the queue can be changed to avoid disk-head seeks
  - **Goal:** optimize/balance fairness, timeliness, and performance
- **Definition of disk scheduling:** permute the order of servicing disk I/O requests from the order they arrive to an order that reduces the length and number of seeks

FCFS scheduling

- First-come first-served (FCFS): order the queue of I/O requests by arrival time
  - intrinsically fair, but does not guarantee fastest service
  - works best when disk load is light



total head movement (seek distance) of 640 cylinders

SSTF scheduling

- shortest seek time first (SSTF): order the queue of I/O requests by track, picking requests on the nearest track to service first
  - always go to the next closest request (greedy algorithm)
  - it's not optimal; minimizes head movement in each step but not the total head movement!

SSTF scheduling

- shortest seek time first (SSTF): order the queue of I/O requests by track, picking requests on the nearest track to service first
  - always go to the next closest request (greedy algorithm)
  - it's not optimal; minimizes head movement in each step but not the total head movement!
- problems:
  - drive geometry is not available to OS (e.g., how many sectors are there per track)

SSTF scheduling

- shortest seek time first (SSTF): order the queue of I/O requests by track, picking requests on the nearest track to service first
  - always go to the next closest request (greedy algorithm)
  - it's not optimal; minimizes head movement in each step but not the total head movement!
- problems:
  - drive geometry is not available to OS (e.g., how many sectors are there per track)
    - **can simply implement nearest-block-first (NBF), which schedules the request with the nearest block address next**

SSTF scheduling

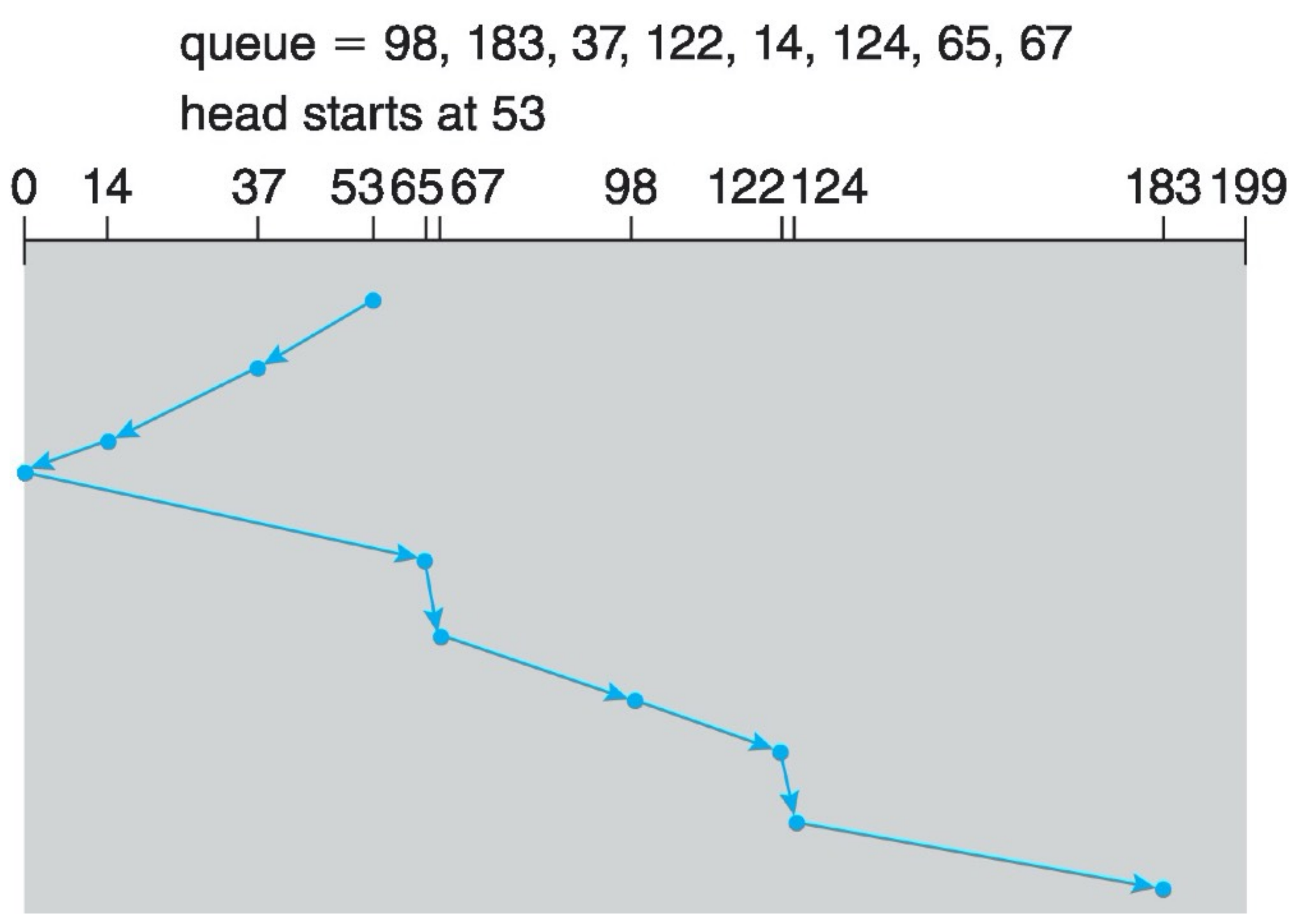
- shortest seek time first (SSTF): order the queue of I/O requests by track, picking requests on the nearest track to service first
  - always go to the next closest request (greedy algorithm)
  - it's not optimal; minimizes head movement in each step but not the total head movement!
- problems:
  - drive geometry is not available to OS (e.g., how many sectors are there per track)
    - **can simply implement nearest-block-first (NBF), which schedules the request with the nearest block address next**
  - starvation is possible: what if there is a steady stream of requests to the inner track, where the head currently is positioned

SCAN scheduling (elevator algorithm)

- disk head continuously scans back and forth across the disk (preventing starvation)
  - each pass across the disk is called a **sweep**
  - disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder
  - the direction of head movement is then reversed and servicing requests continues

a request arriving in the queue just in front of the head is serviced immediately

a request arriving in the queue just behind the head will have to wait

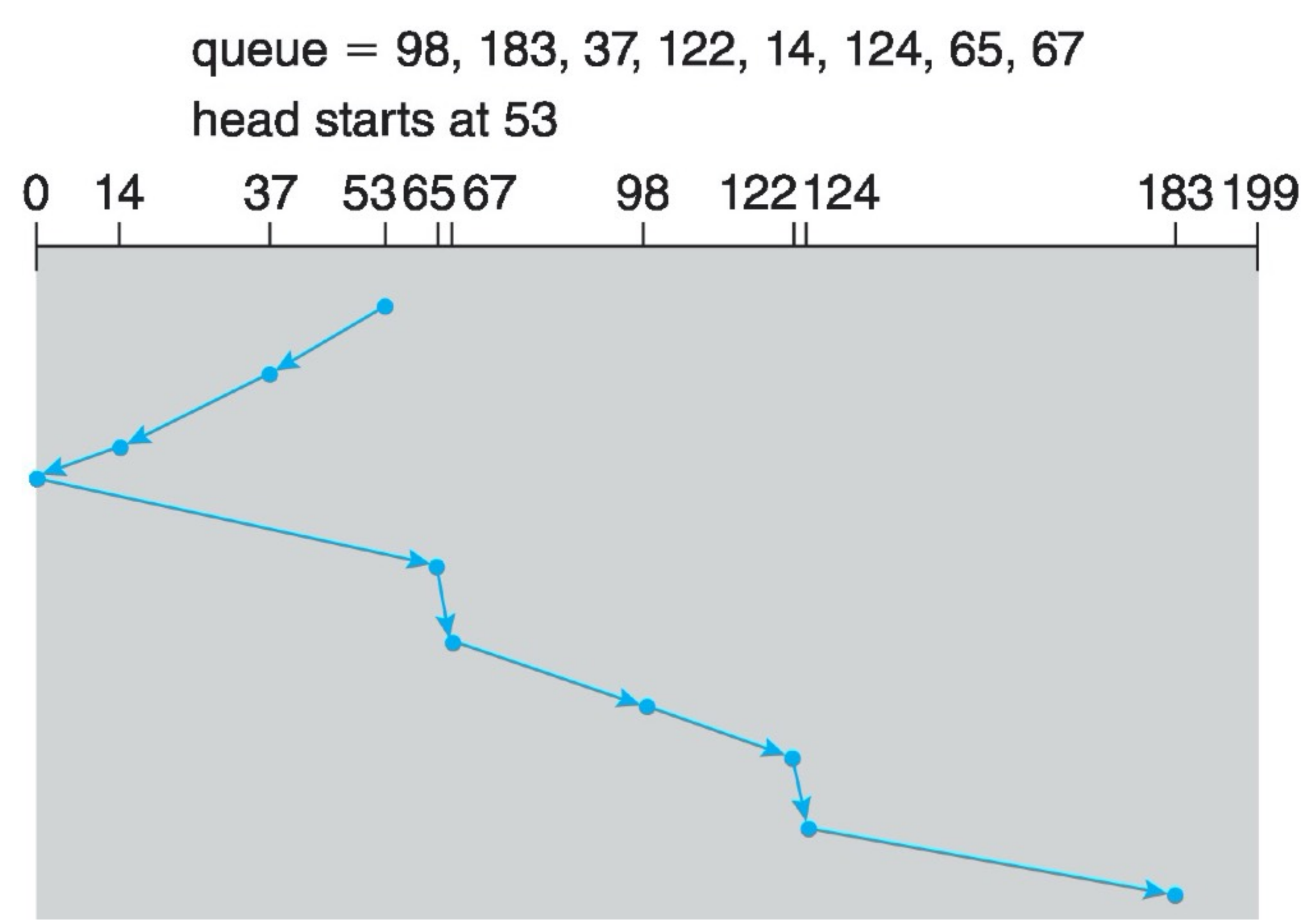


SCAN scheduling (elevator algorithm)

- disk head continuously scans back and forth across the disk (preventing starvation)
  - each pass across the disk is called a **sweep**
  - disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder
  - the direction of head movement is then reversed and servicing requests continues
- need to know the head initial position and its initial direction of movement

a request arriving in the queue just in front of the head is serviced immediately

a request arriving in the queue just behind the head will have to wait

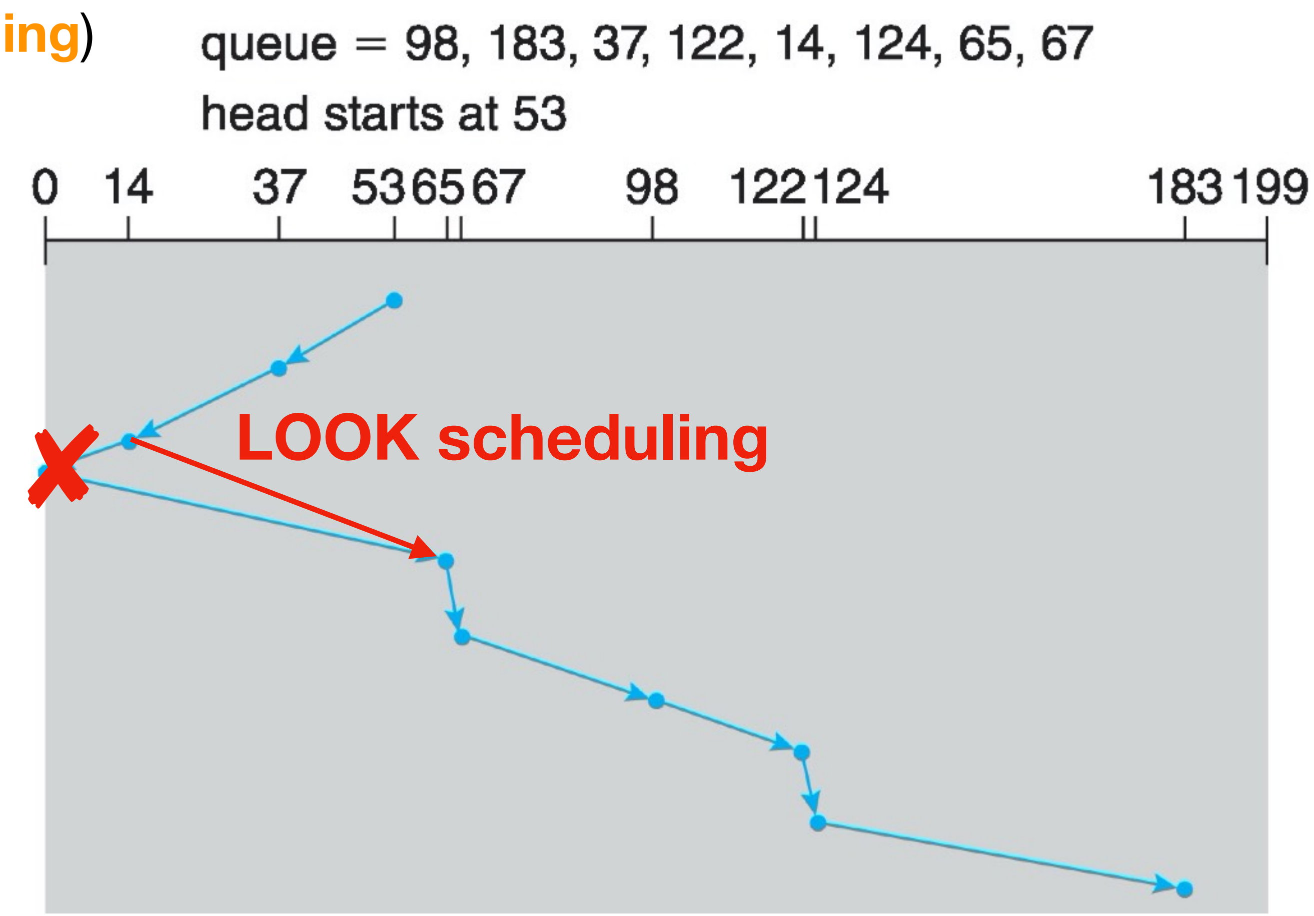


SCAN scheduling (elevator algorithm)

- disk head continuously scans back and forth across the disk (preventing starvation)
  - each pass across the disk is called a **sweep**
  - disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder
  - the direction of head movement is then reversed and servicing requests continues
- need to know the head initial position and its initial direction of movement
- if there is no request between current position and the disk end, we can change the direction of movement right away (**LOOK scheduling**)

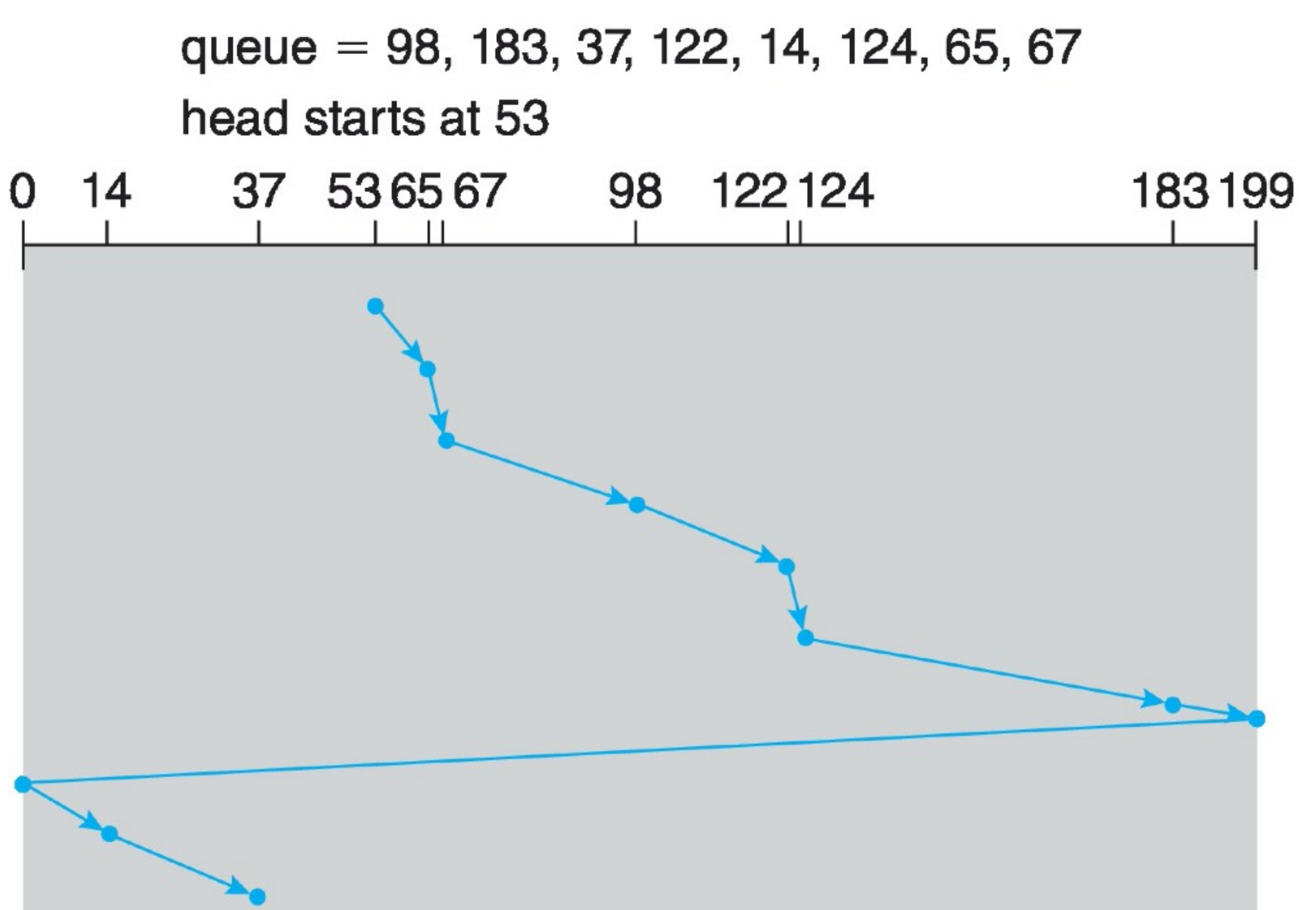
a request arriving in the queue just in front of the head is serviced immediately

a request arriving in the queue just behind the head will have to wait



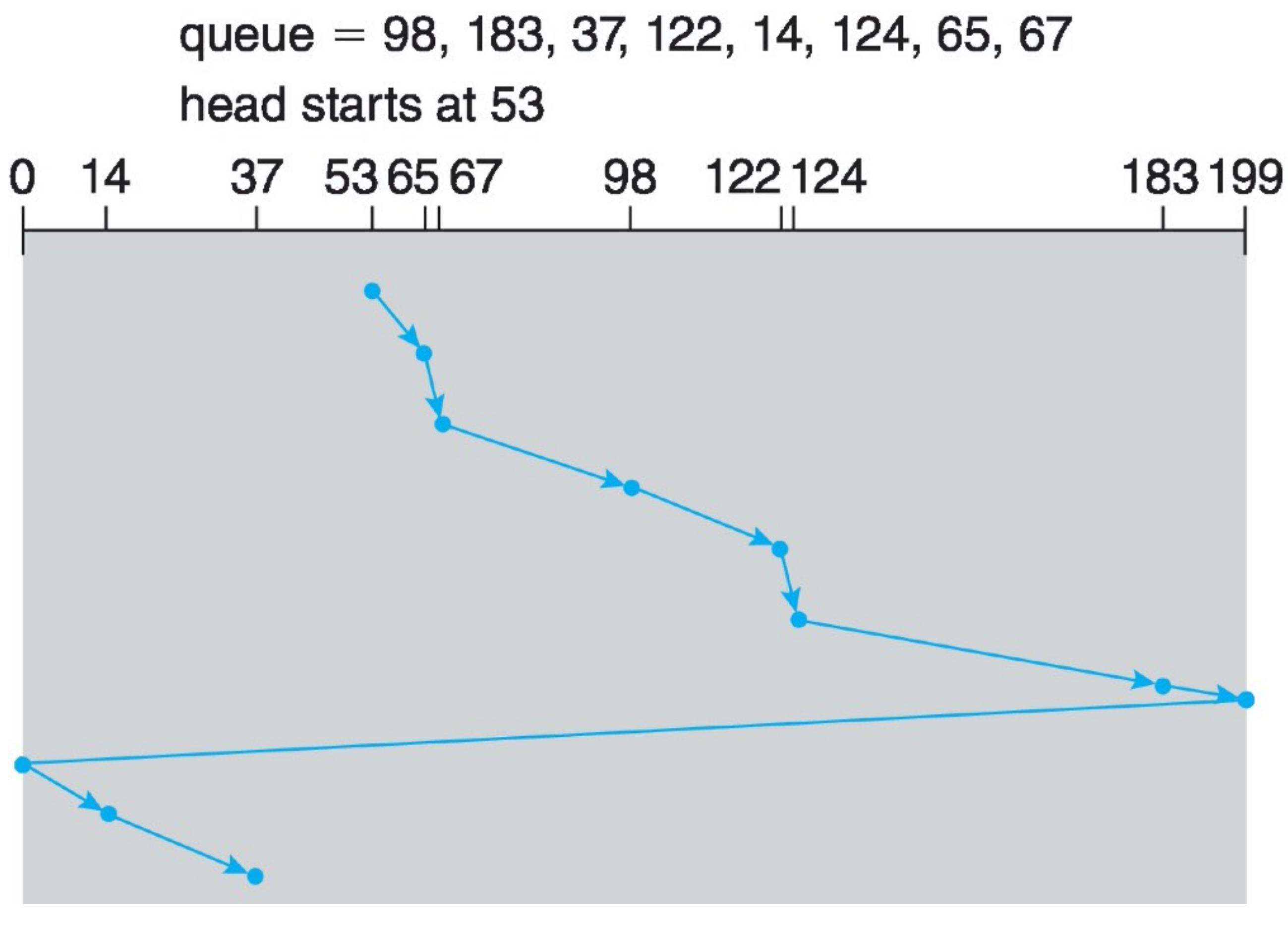
Circular SCAN (C-SCAN) scheduling

- variant of SCAN scheduling with one difference
  - when the head reaches one end of the disk it immediately returns to the other end without servicing any requests on the return path
  - why? assuming uniform distribution of requests, the chance of having requests ahead of the head on the return trip is low as these cylinders have recently been serviced



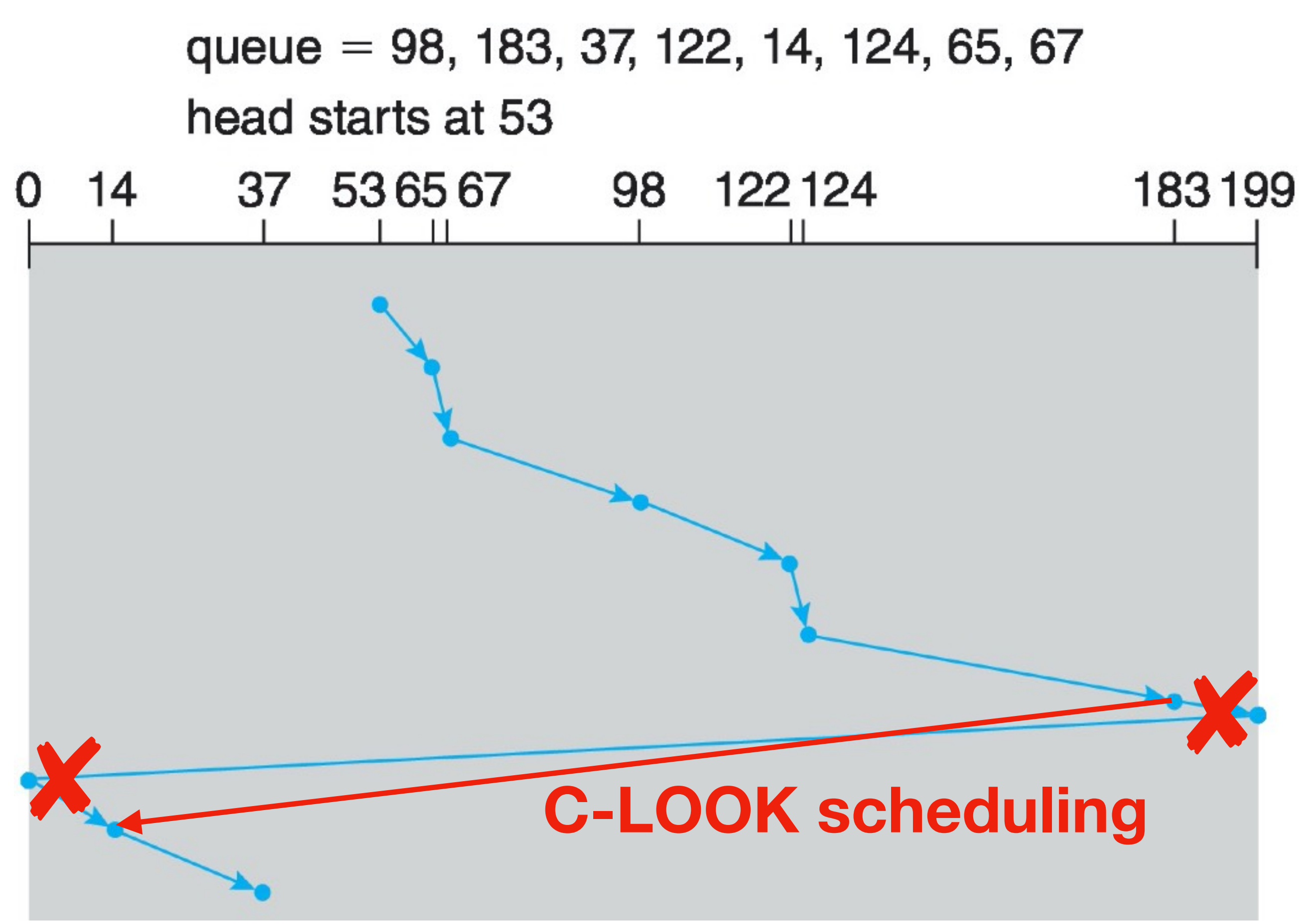
Circular SCAN (C-SCAN) scheduling

- variant of SCAN scheduling with one difference
  - when the head reaches one end of the disk it immediately returns to the other end without servicing any requests on the return path
  - why? assuming uniform distribution of requests, the chance of having requests ahead of the head on the return trip is low as these cylinders have recently been serviced
- this algorithm treats disk cylinders as a circular list (wrapping around from the final cylinder to the first one)



Circular SCAN (C-SCAN) scheduling

- variant of SCAN scheduling with one difference
  - when the head reaches one end of the disk it immediately returns to the other end without servicing any requests on the return path
  - why? assuming uniform distribution of requests, the chance of having requests ahead of the head on the return trip is low as these cylinders have recently been serviced
- this algorithm treats disk cylinders as a circular list (wrapping around from the final cylinder to the first one)



Comparison

- SCAN and C-SCAN (and LOOK and C-LOOK) perform better when there is heavy load on the disk (i.e., the disk service queue is almost full most of the time)
  - starvation is less likely compared to SSTF under these scheduling algorithm

Comparison

- SCAN and C-SCAN (and LOOK and C-LOOK) perform better when there is heavy load on the disk (i.e., the disk service queue is almost full most of the time)
  - starvation is less likely compared to SSTF under these scheduling algorithm
- SCAN favors the middle tracks compared to C-SCAN because it passes through the middle twice before coming back to the outer track

Comparison

- SCAN and C-SCAN (and LOOK and C-LOOK) perform better when there is heavy load on the disk (i.e., the disk service queue is almost full most of the time)
  - starvation is less likely compared to SSTF under these scheduling algorithm
- SCAN favors the middle tracks compared to C-SCAN because it passes through the middle twice before coming back to the outer track
- all these algorithms ignore rotation delay and optimize for seek delay only
  - makes sense if seek delay is much higher than rotational delay
  - but what if it is not the case for some storage technology?

Considering both seek time and rotational latency

- to approximate SJF policy, it is necessary to consider both seek time and rotational latency

Considering both seek time and rotational latency

- to approximate SJF policy, it is necessary to consider both seek time and rotational latency
- shortest access time first (SATF)
  - service the I/O request with the least positioning time (seek time + rotational latency)
  - need to know the relative time of seeking compared to rotation (OS doesn't usually have this information)

Considering both seek time and rotational latency

- to approximate SJF policy, it is necessary to consider both seek time and rotational latency
- shortest access time first (SATF)
  - service the I/O request with the least positioning time (seek time + rotational latency)
  - need to know the relative time of seeking compared to rotation (OS doesn't usually have this information)
- analogous to Traveling Salesman Problem (TSP)

Considering both seek time and rotational latency

- to approximate SJF policy, it is necessary to consider both seek time and rotational latency
- shortest access time first (SATF)
  - service the I/O request with the least positioning time (seek time + rotational latency)
  - need to know the relative time of seeking compared to rotation (OS doesn't usually have this information)
- analogous to Traveling Salesman Problem (TSP)
- in modern systems, disks have sophisticated internal schedulers themselves
  - so the OS scheduler usually issues a small batch of I/O requests that thinks are the best
  - internal disk scheduler services these requests in the SATF order

Improving disk performance using read ahead

- to reduce the number of seeks, we can read blocks that will probably be used while you have them under the head
- read blocks from the disk ahead of user's request and place in buffer on disk controller
- can we predict which disk blocks will be needed? with sequential access?

Deadline scheduler in Linux

- maintain separate read and write queues (specifically two read queues and two write queues) and give read requests a higher priority
  - because processes are more likely to block on read than write (write-back is often used)
  - requests are submitted to queues in batch
- one read queue and one write queue are kept sorted by logical block address order
  - it is similar to C-SCAN as both queues are kept sorted in logical block address order
- one read queue and one write queue are kept sorted by FCFS
- for each batch, if there are requests in FCFS queues older than some predefined age, they are serviced first (**fairness**)
  - otherwise, requests in queues sorted by logical block address are serviced