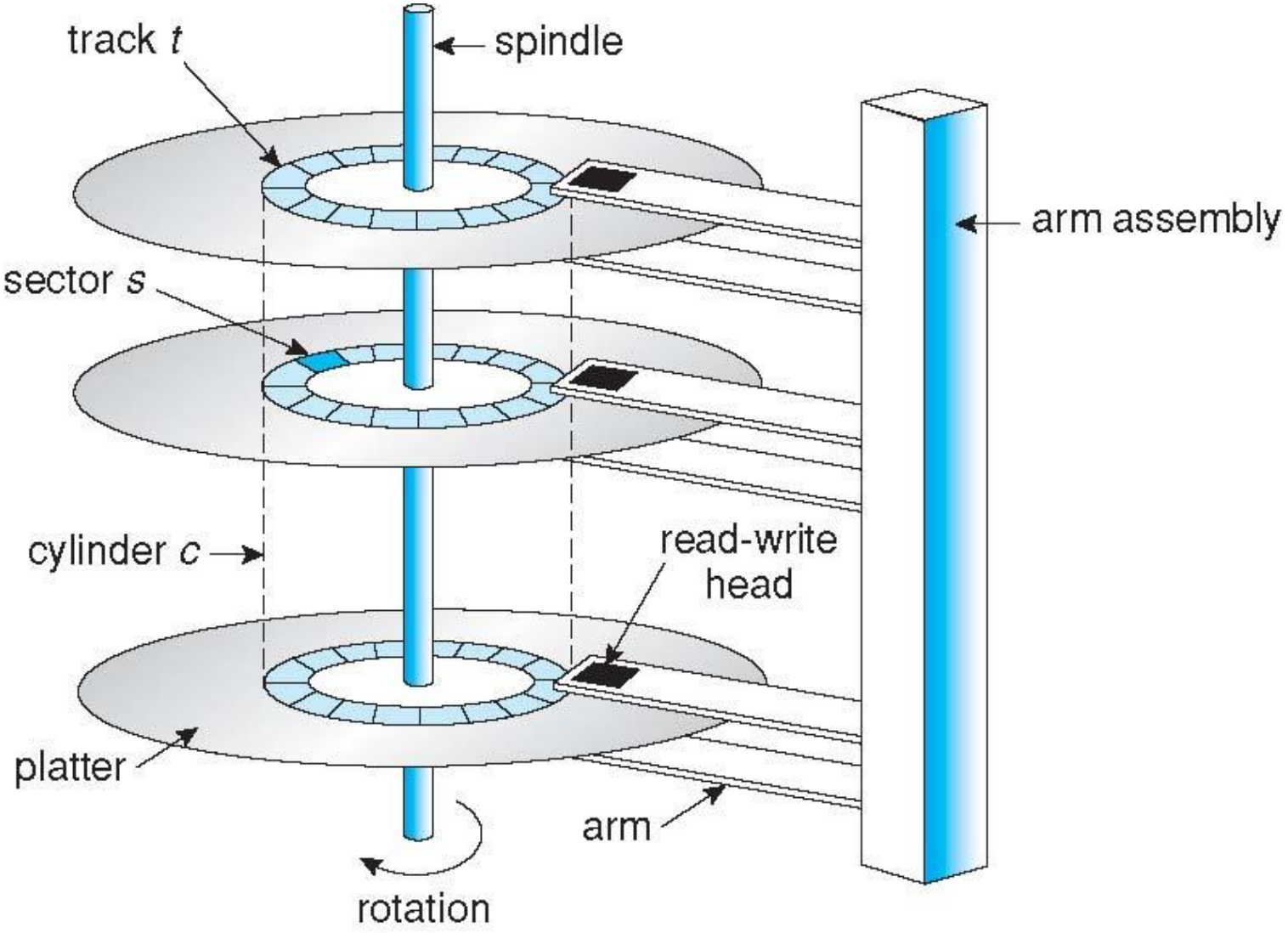
Disk and I/O Management

Moving-head Disk Mechanism



Magnetic Disk Performance

- **Average access time** = average seek time + average latency+ transfer time

- For example, to transfer a k bytes block on a 3600 RPM disk with a 10ms average seek time with a .1ms controller overhead
- Track size=32kB
- block size k

Rotation time=60/3600 sec=16.67 ms

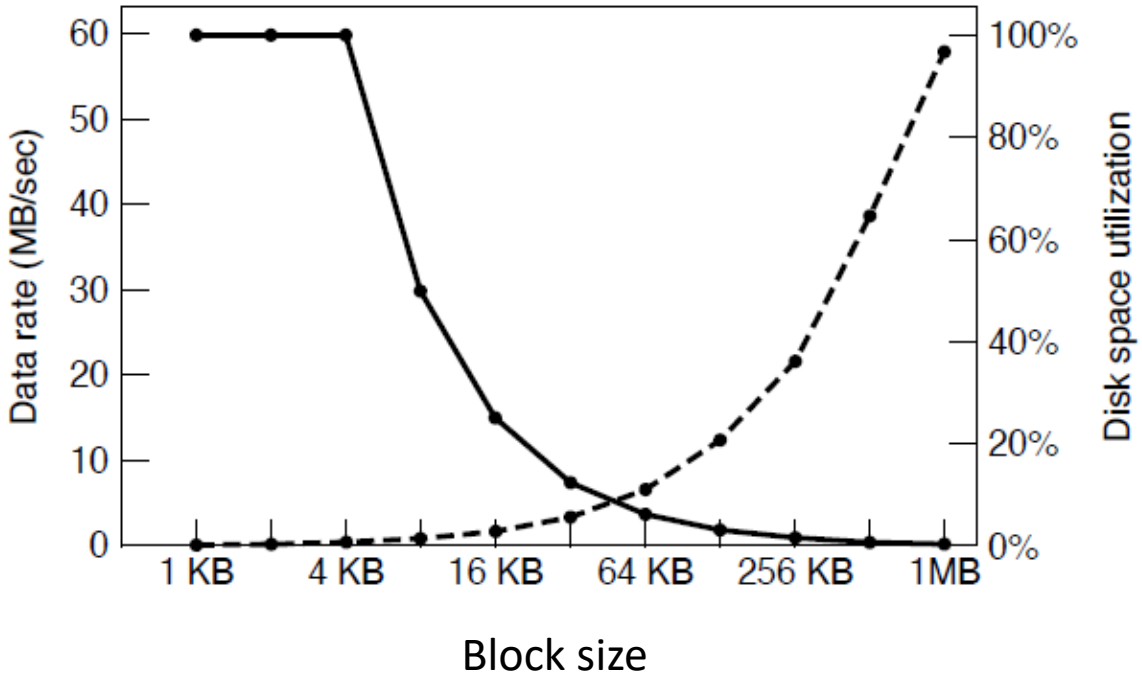
Average rotational latency=(0+r)/2=r/2

Avg. access time=seek time+ rotational delay +transfer time

  - 10ms + 8.3ms + (k / 32KB)×16.67 sec +0.1ms

Data rate vs storage efficiency

File size: 4KB

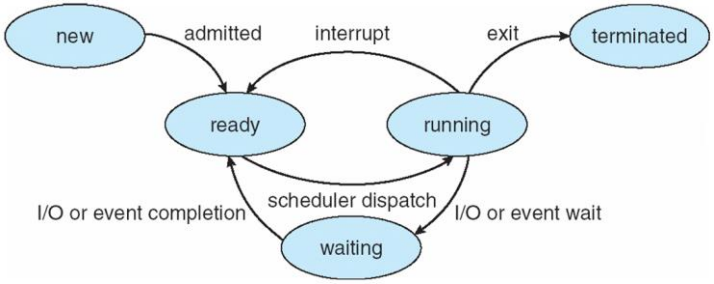


Disk Scheduling

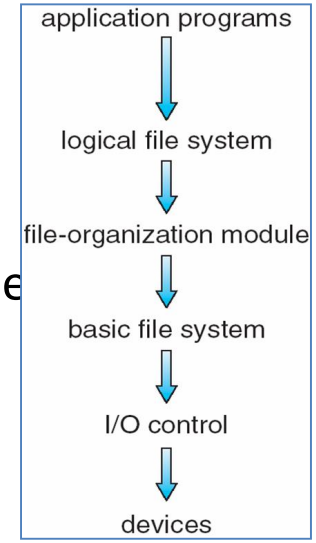
- The operating system is responsible for using hardware efficiently
  - for the disk drives, this means having a fast access time
- Minimize seek time
- Seek time $\approx$ seek distance (depends on track numbers)

Disk Scheduling

- There are many sources of disk I/O request
  - System processes
  - Users processes



- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request
  - busy disk means work must queue



- Note that drive controllers have small buffers and can manage a queue requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests

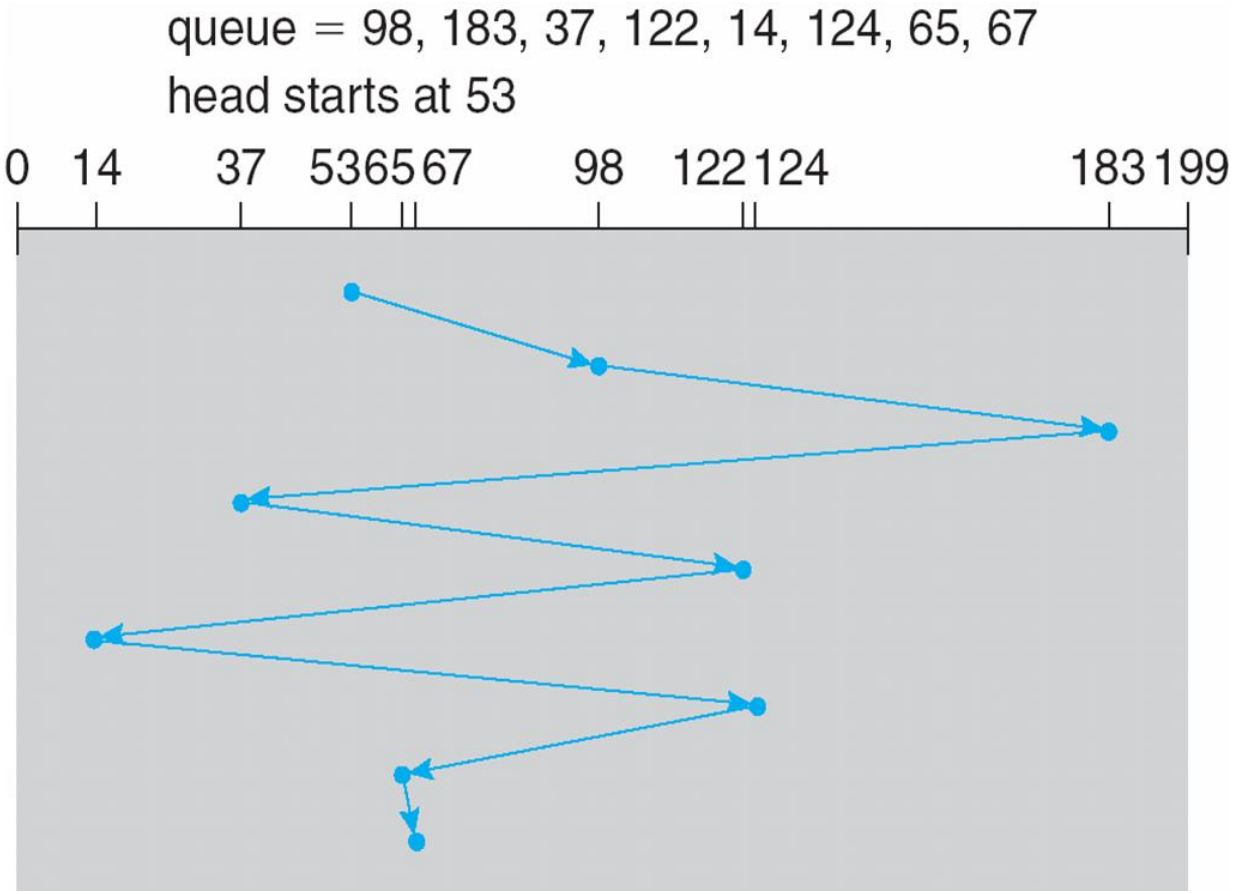
- We illustrate scheduling algorithms with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67 (track numbers)

Head pointer 53

FCFS

Illustration shows total head movement of 640 cylinders



SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
  - Close to the current disk head

SSTF

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

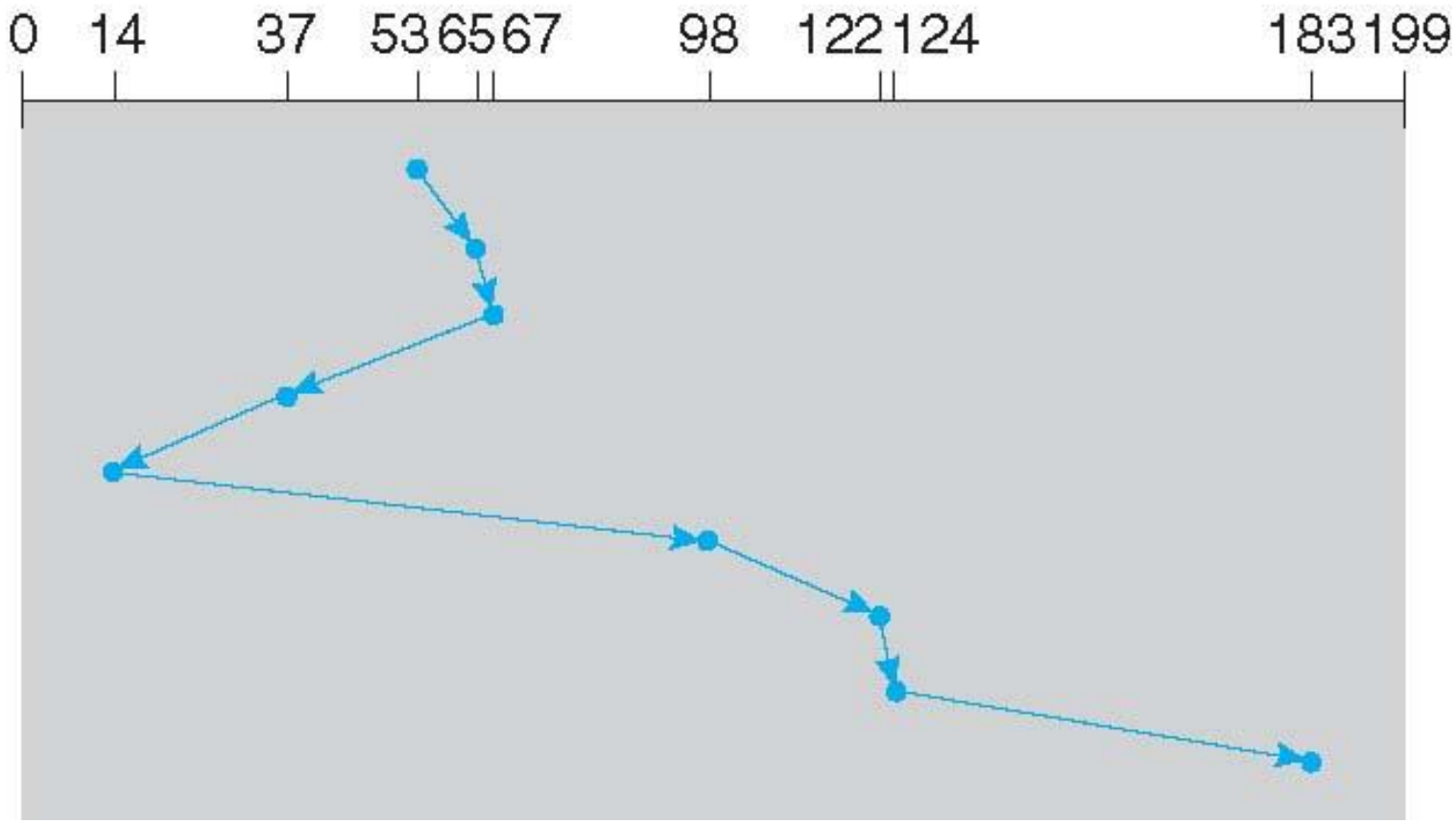


Illustration shows total head movement of 236 cylinders

SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
  - Close to the current disk head
- Greedy
  - Not necessarily optimal
- SSTF scheduling may cause starvation of some requests

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

- **SCAN algorithm** Sometimes called the **elevator algorithm**

- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

SCAN

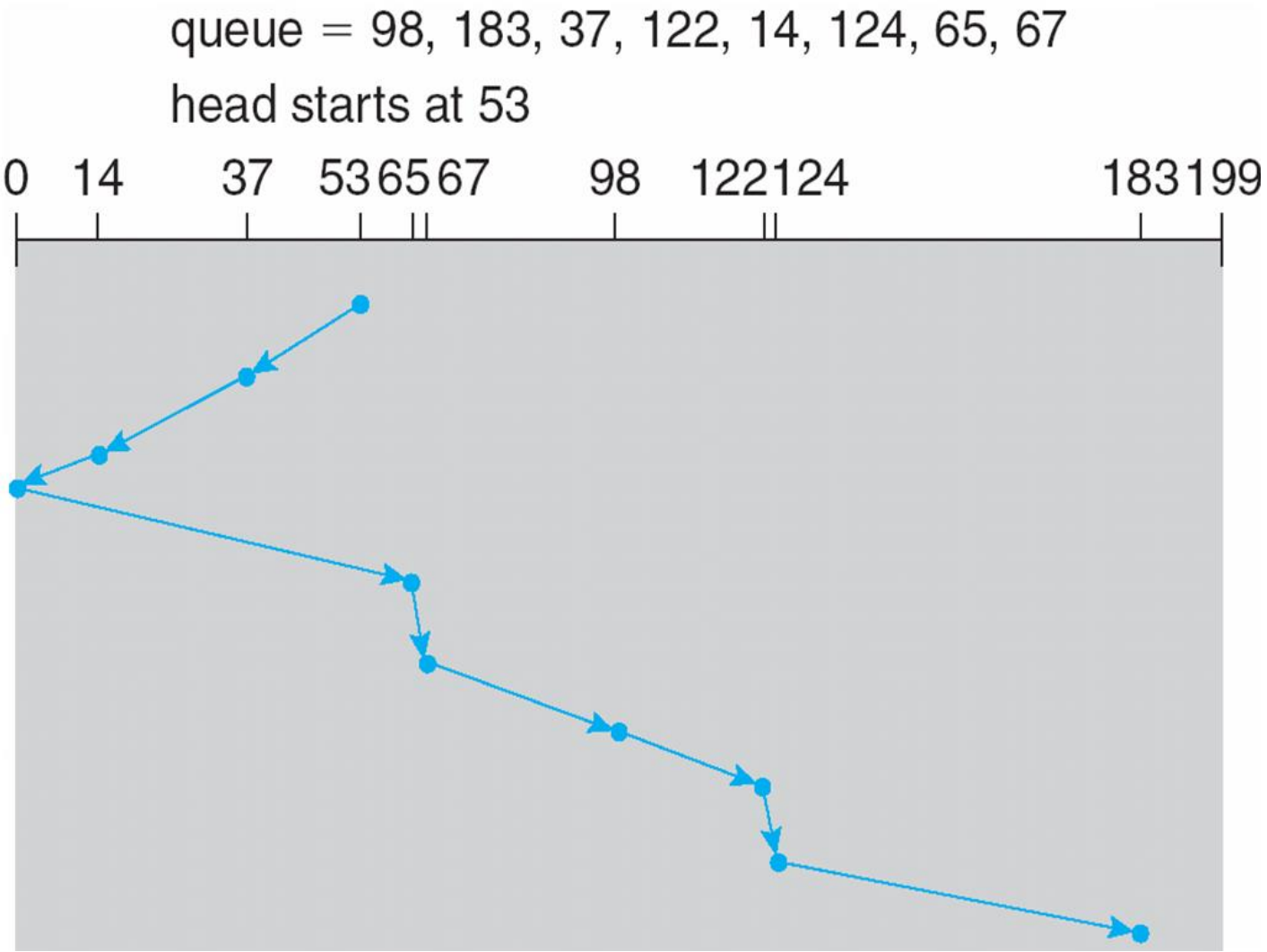


Illustration shows total head movement of 208 cylinders

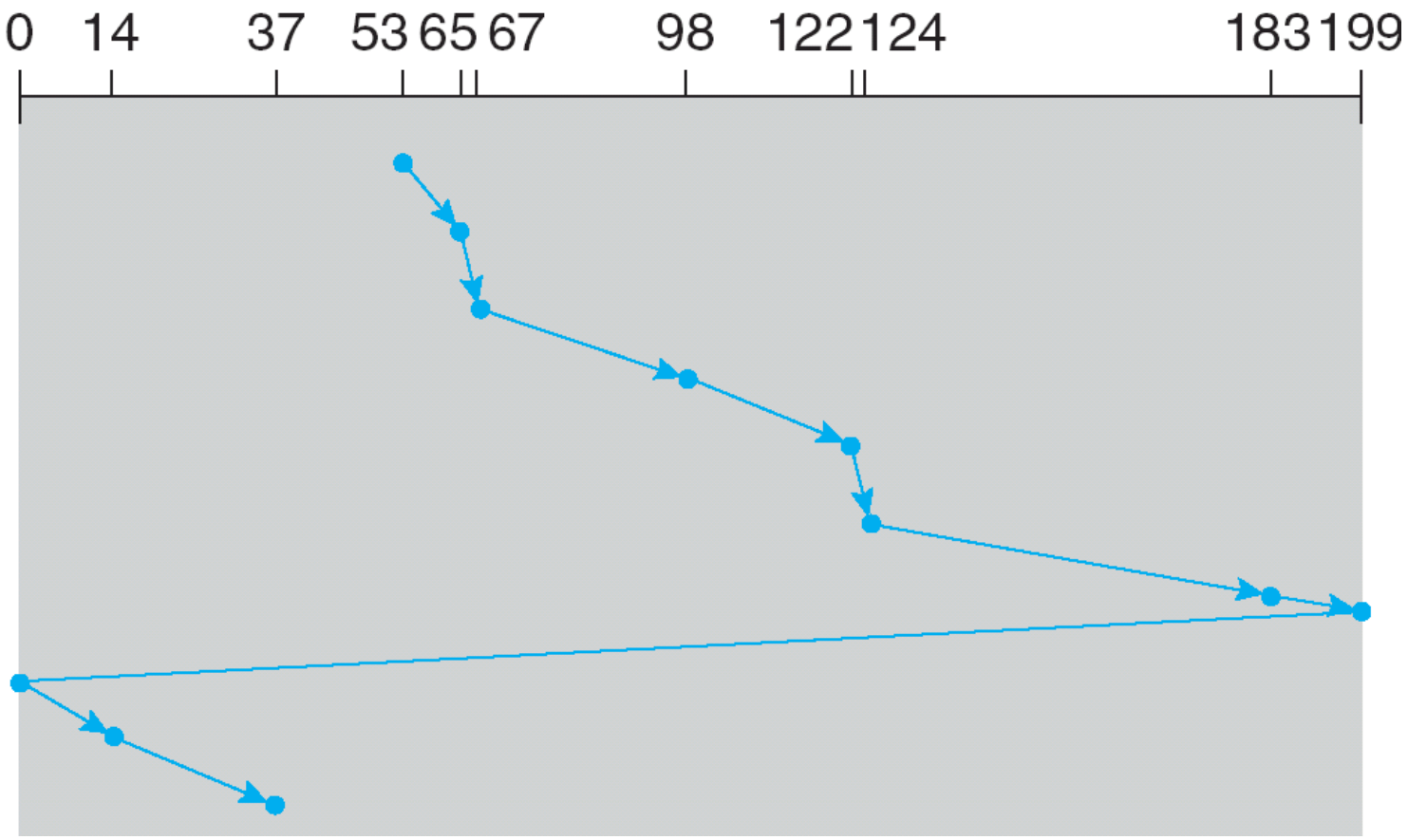
C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN (Cont.)

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

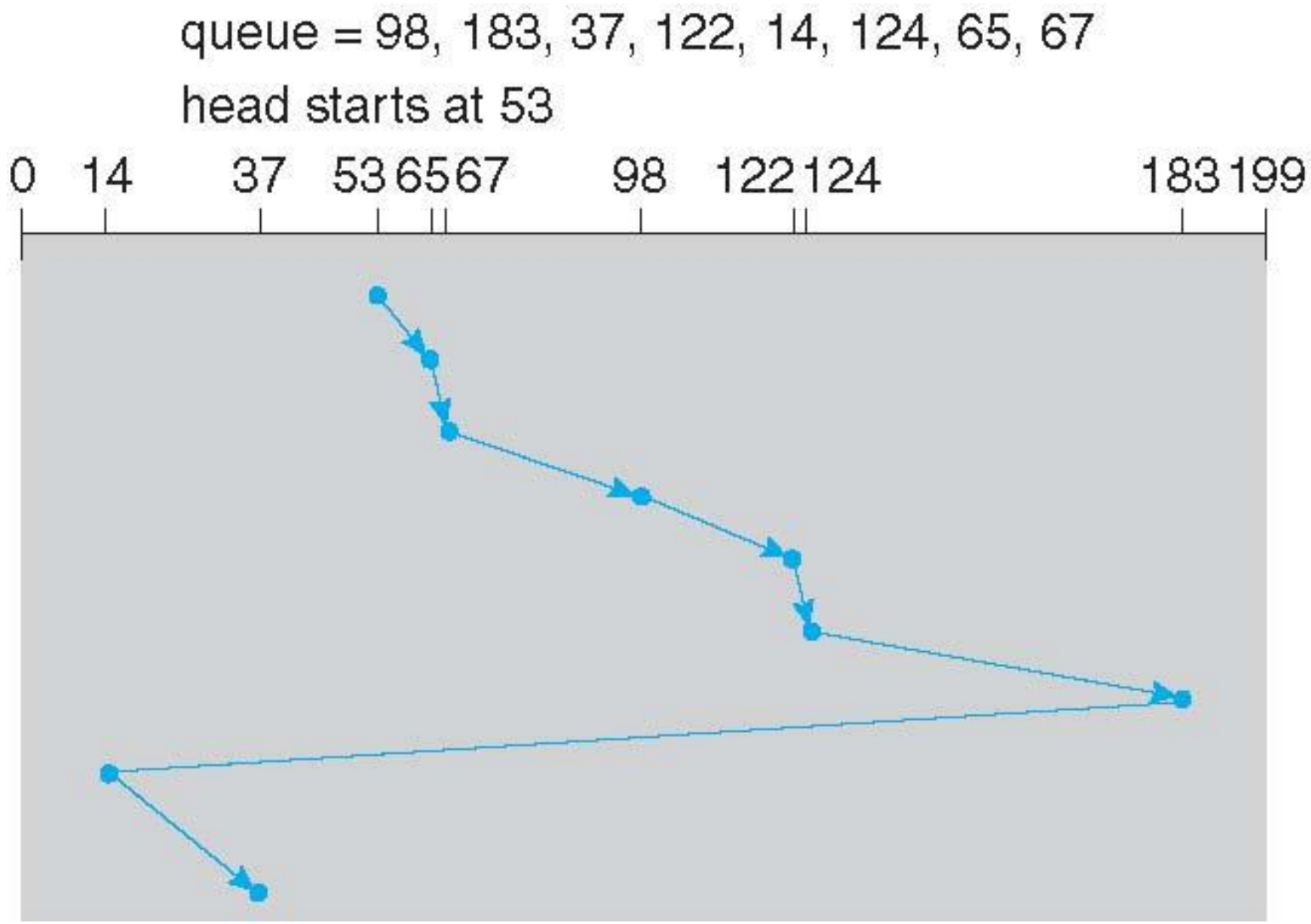


LOOK

- LOOK a version of SCAN, C-LOOK a version of C-SCAN

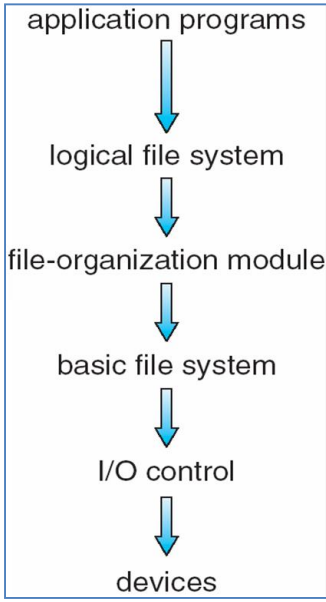
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk

C-LOOK (Cont.)

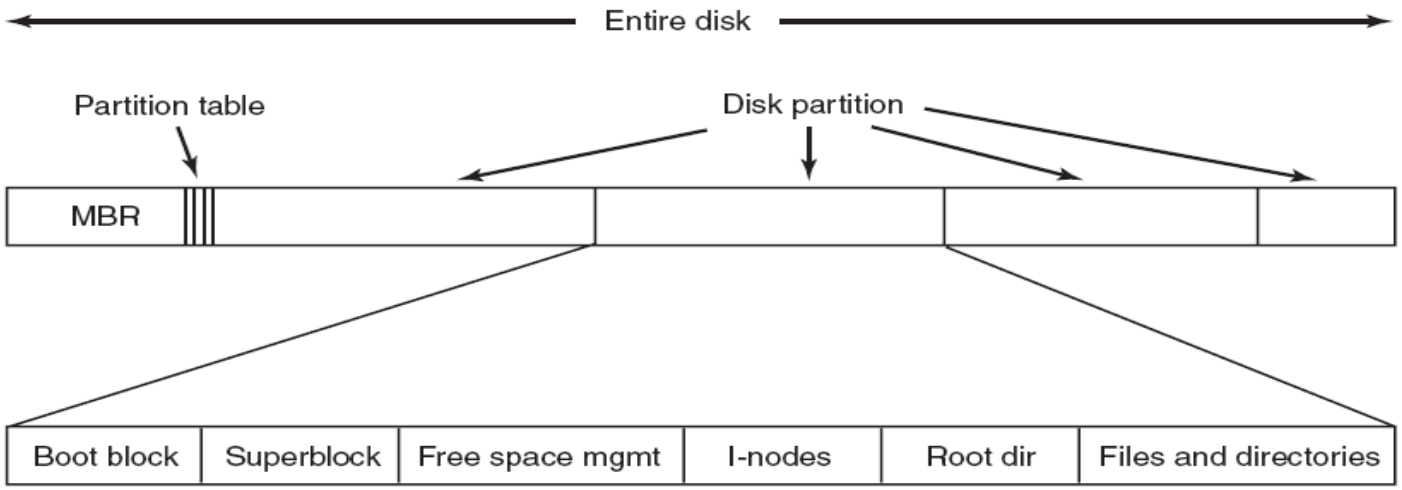

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Selecting a Disk-Scheduling Algorithm

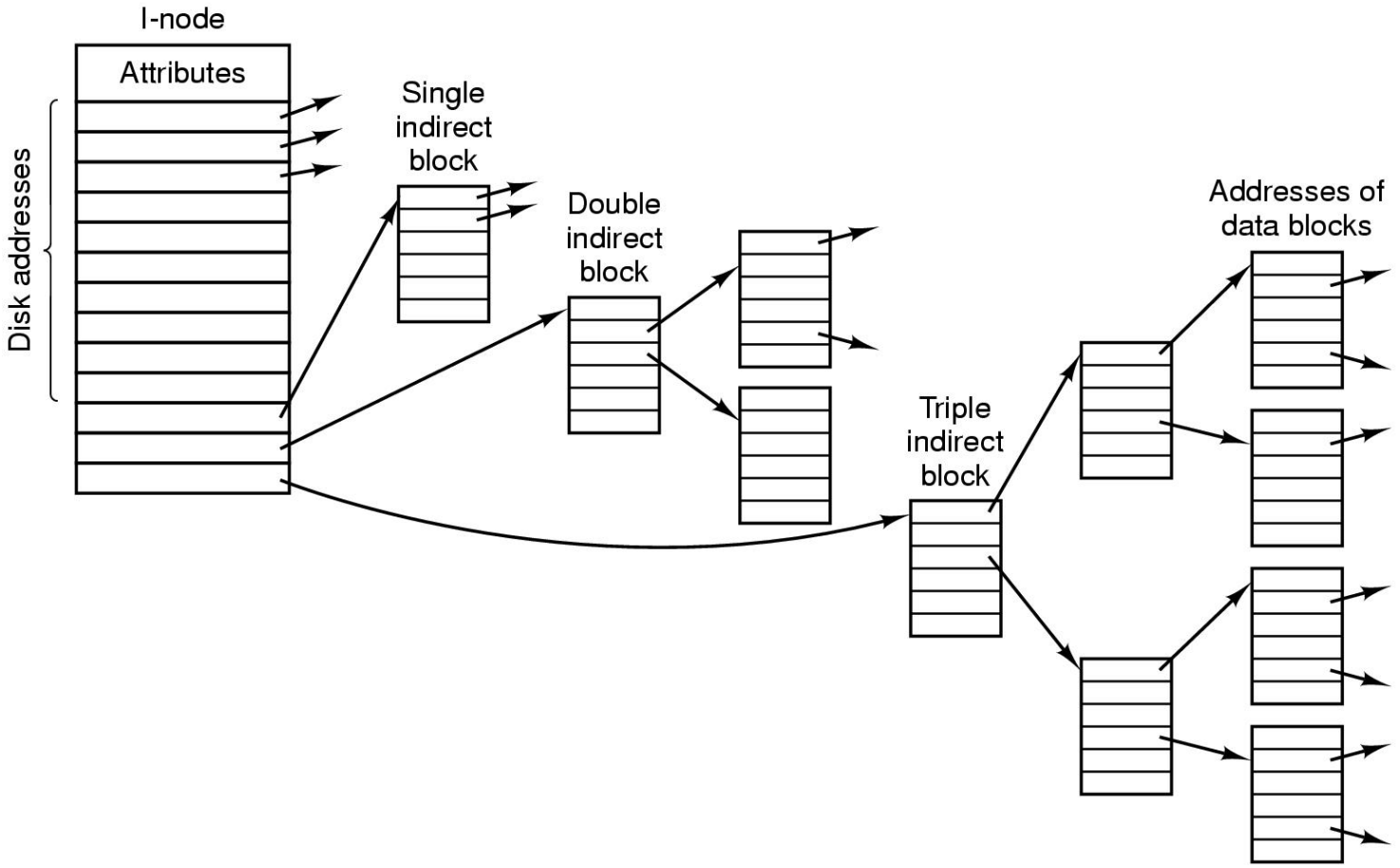
- Requests for disk service can be influenced by the file-allocation method
  - Contiguous, indexed etc
  - Location of Directory , i-node
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
  - Either SSTF or LOOK is a reasonable choice for the default algorithm
- But implemented inside disk controller
  - Physical block info is inside controller
- How does disk-based queuing effect OS queue ordering?
  - OS has other constraints
  - Demand paging over normal I/O
  - Controller is unaware of that
  - File organization module ensures the order



Disk Layout


Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories

The UNIX File System



A UNIX i-node.

Disk Management

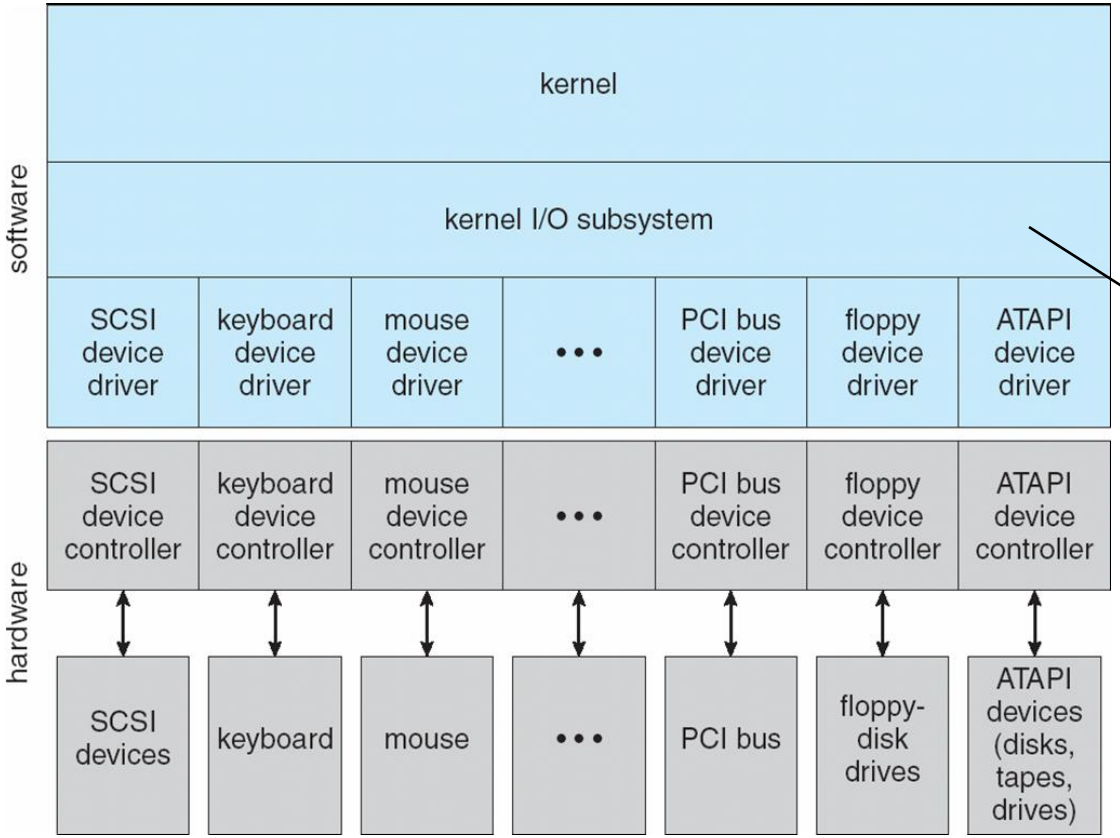
- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
  - Logical to physical address map

- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **Logical formatting** or “making a file system”
  - Create the data structures (FAT, directory, inode, free space)

Application I/O Interface

- OS structure enables the I/O devices to be treated in a standard/uniform way
  - Application can open a file on a disk without knowing the disk details
  - New devices can be added without disrupting the OS

Application I/O Interface



- Abstract away detailed differences in I/O devices identifying few general/standard syscall (read(), write())
  - Interface
- I/O system calls encapsulate device behaviors in generic classes
- Kernel program interacts with I/O system using standard system call

- Device differences are encapsulated within device drivers (custom tailored for that device) => provides a generic interface to interact
- Device-driver layer hides differences among I/O controllers from kernel
- Want to add new device?
  - New devices talking already-implemented controller need no extra work
- Otherwise write a driver for that device (specific to particular OS)

Thank you