

Lecture 13: Mass-Storage Systems

COMP362 Operating Systems
Prof. AJ Biesczad

Outline



- Location of Mass-Storage Media
- Magnetic Disks
 - Structure
 - Scheduling
- Solid State Drives
- Performance Issues

Classification of Storage



- Function
 - **primary** - computer memory directly accessed by CPU during computation
 - **secondary** - memory indirectly accessible to CPU by copying to and from the primary storage
 - **tertiary** - backup; accessible to CPU (still indirectly) after loading to secondary storage
- Access
 - **online** - available at once
 - **nearline** - available after some automated action executed
 - e.g. a robot arm inserts a tape into a reader
 - **offline** - available only after human intervention
- Location
 - **directly-attached (DAS)**
 - a.k.a. **host-attached**
 - **network-attached (NAS)**
 - **storage area network (SAN)**
 - attached through a switching fabric
 - **cloud**
 - managed at the application rather than OS level
 - some OS “hooks” needed for advanced features like automatic syncing

Directly Attached Storage (DAS)

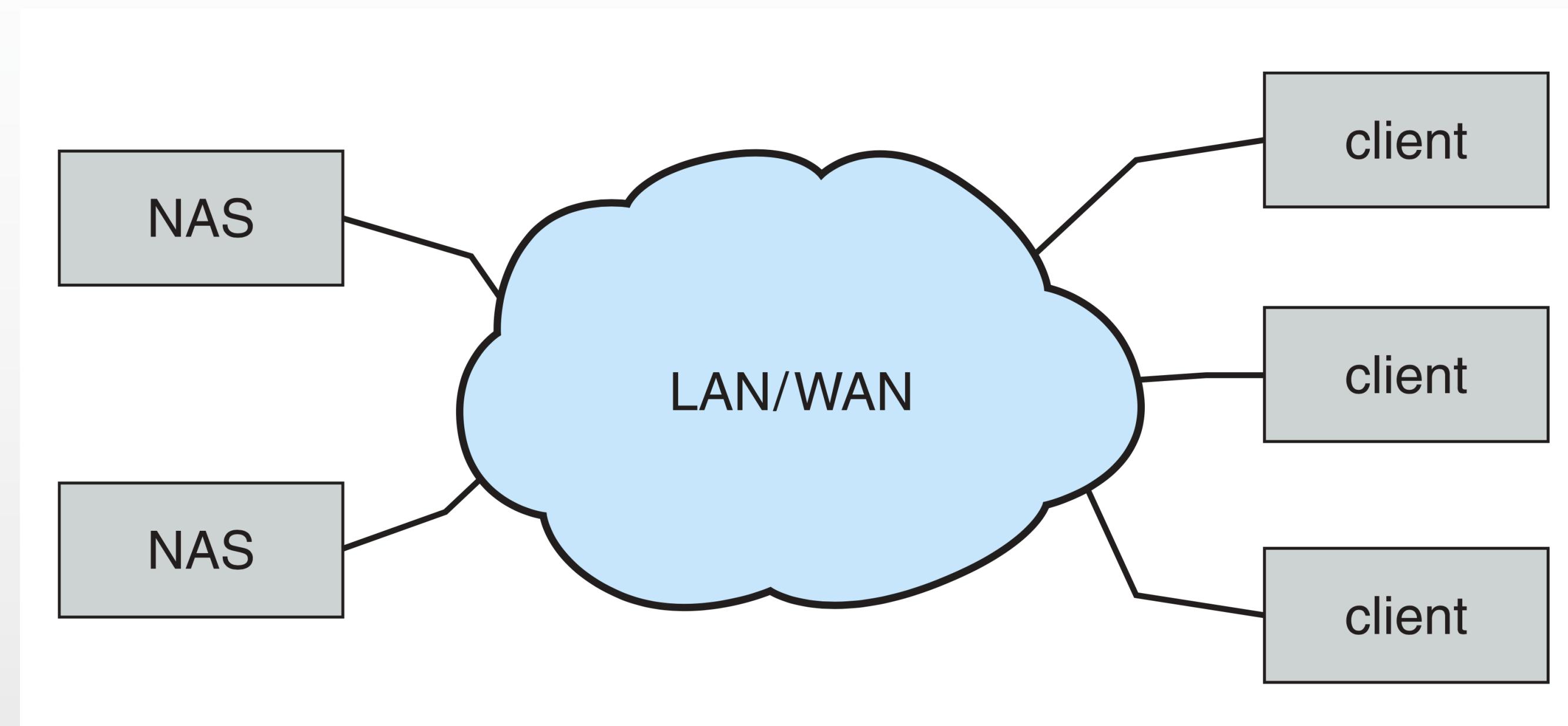


- Host-attached storage accessed through **I/O ports and I/O buses**
- **ATA (Advanced Technology Attachment)** is older parallel interface
 - also called **IDE** (Integrated Drive Electronics)
 - also **ATAPI (ATA Packet Interface)** for non-HD devices
 - e.g., add eject command
- **SATA** is newer serial interface (**Serial ATA**)
 - also external **eSATA**
- **USB (Universal Serial Bus)**
 - **USB-C** increasingly popular
 - v3.1 up to 10 Gbps
- **SCSI (pronounced scuzzy) Small Computer System Interface**
 - a bus, up to 16 devices on one cable (initially parallel; recently serial; iSCSI uses IP)
 - uniform interface
 - mostly on high end systems, servers
- **FC (fiber channel)** is high-speed serial architecture
 - can be attached to a switched fabric with 24-bit address space
 - the basis of storage area networks (SANs) in which many hosts attach to many storage units
 - can also be attached to an **arbitrated loop (FC-AL)** of 126 devices
- **Thunderbolt**
 - daisy-chain up to 6 devices
 - up to 3 m copper and 100 m optical
 - originally optical; switched to copper for cost and ability to supply power



Network-Attached Storage (NAS)

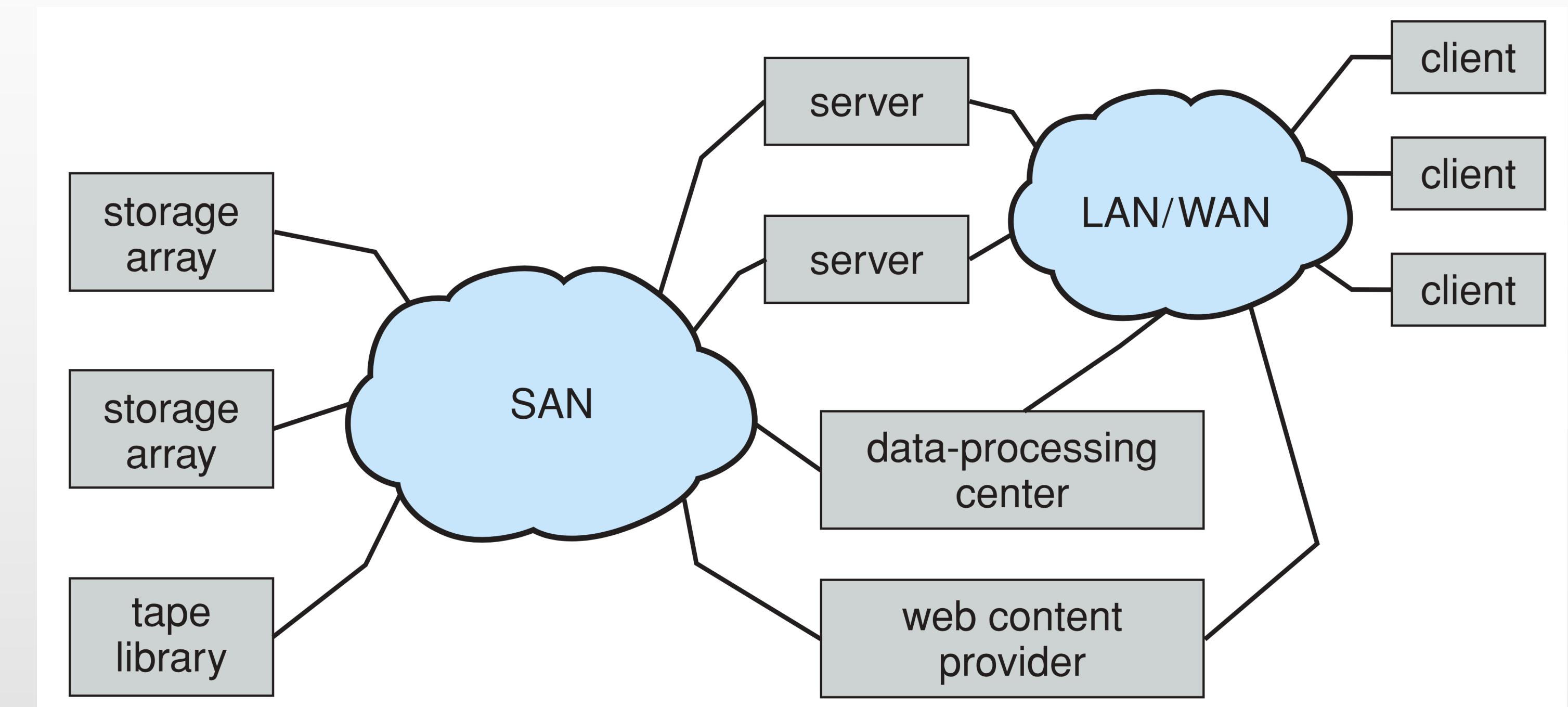
- File-level storage made available over a network rather than over a local connection (such as a bus)
 - clients read/write files on remote computers
- Uses network access protocols
 - **NFS** historical
 - **CIFS/SMB** are common protocols
 - **AFP** in Mac world
- Implemented via **remote procedure calls (RPCs)** between local host and storage host
 - RPC allows for calling functions on remote computers
 - precursor of Java's **RMI** (Remote Method Invocation), Microsoft's **.Net**
 - **iSCSI** protocol uses IP network to carry the SCSI protocol



Storage Area Network (SAN)



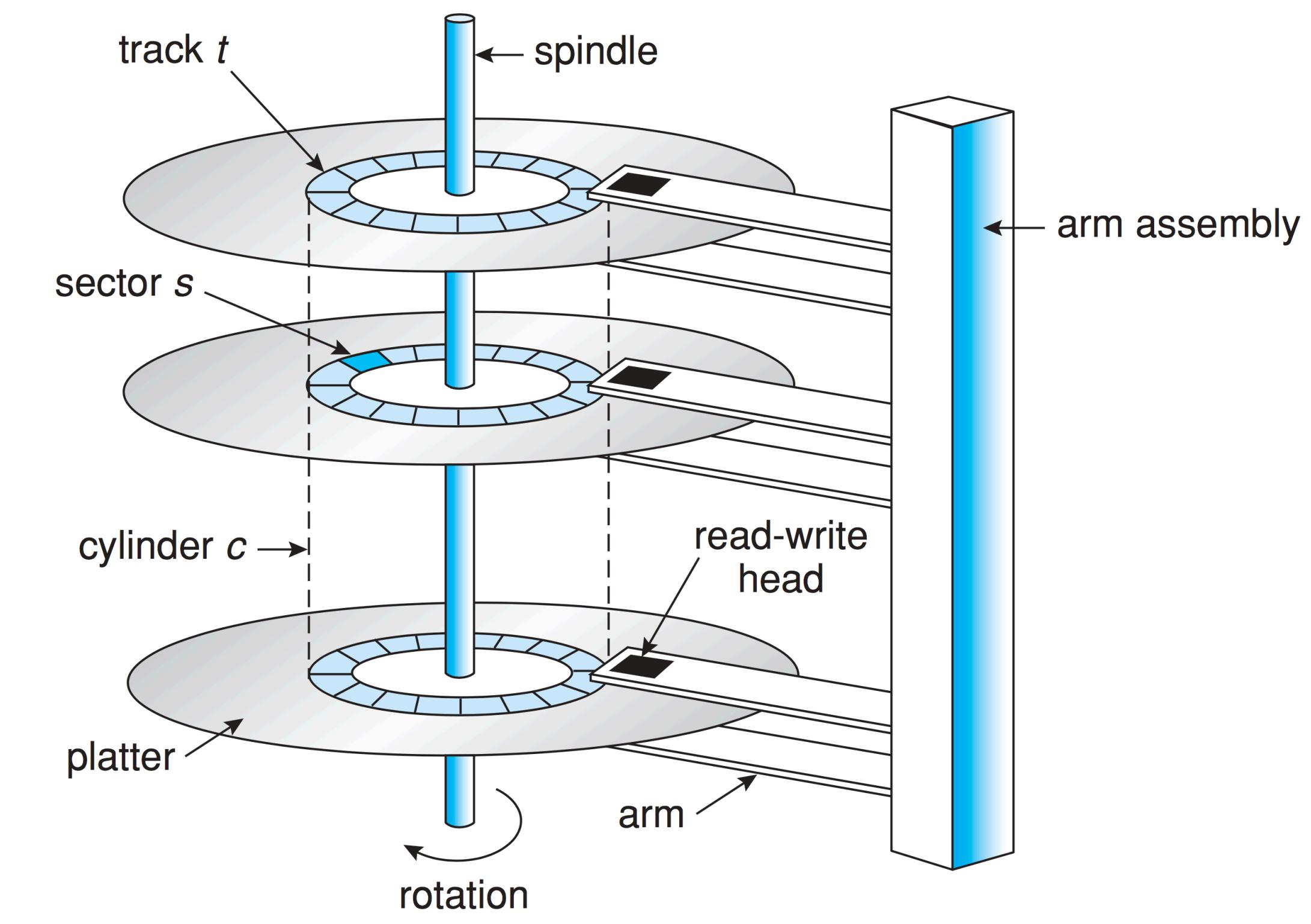
- Block-level storage common in large storage environments
 - blocks are read/written rather than files
- Multiple hosts attached to multiple storage arrays
 - very flexible
 - for example, can provide storage utility (storage on-demand)
- Attached devices appear as local
 - uses local rather than network access protocols





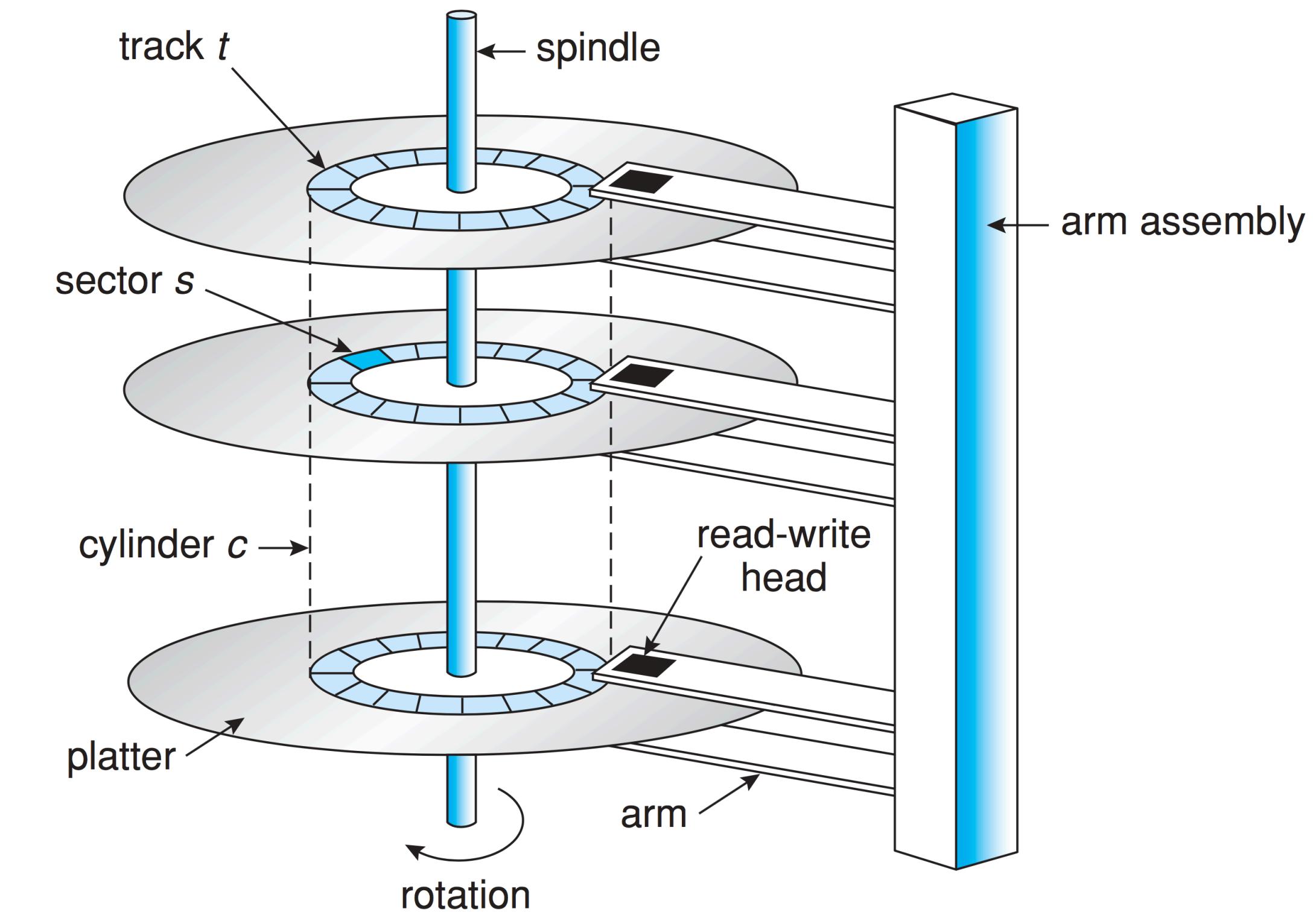
Magnetic Disks

- Although solid state memory is becoming more common, magnetic disks still provide bulk of secondary storage of modern computers
 - drives rotate at 60 to 200 times per second
 - 5400 rpm, 7200 rpm, 10,000 rpm, etc.
 - **transfer rate** is rate at which data flow between drive and computer
 - **positioning time** (random-access time)
 - time to move disk arm to desired cylinder (**seek time**) and
 - time for desired sector to rotate under the disk head (**rotational latency**)
 - head crash may occur as a result of disk head making contact with the disk surface
- Disks can be removable
- Drive attached to computer via I/O bus
 - buses vary, including EIDE, ATA, SATA, USB, Thunderbolt, Fiber Channel, SCSI
 - host controller in computer uses bus to talk to disk controller built into drive or storage array



Disk Structure

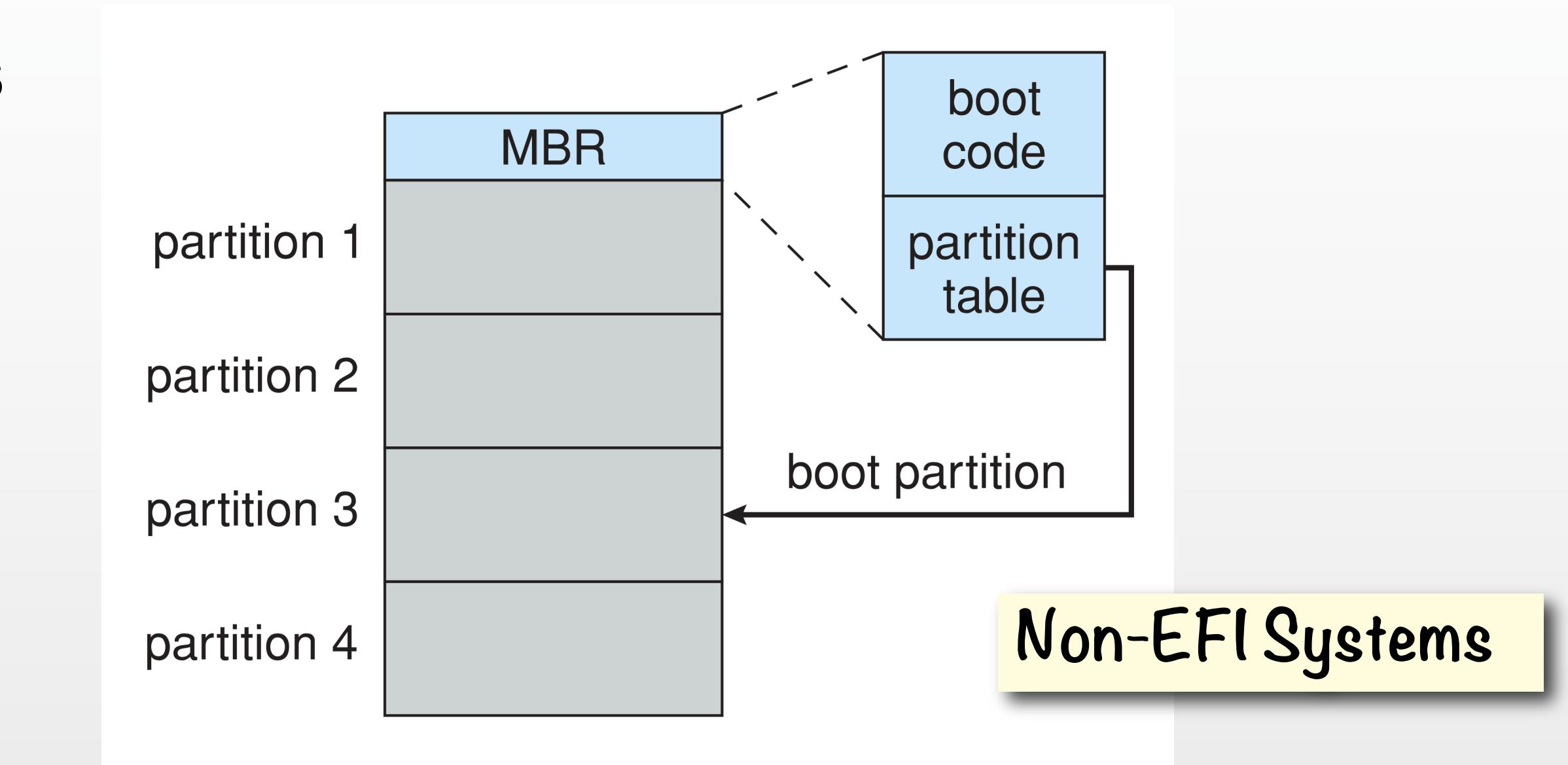
- **Cylinder-head-sector (CHS)**
 - historical organization of discs that reflected internal disk structure
 - originally built into system architecture (BIOS), and still used in some utilities
 - does not map well onto new media
- **Logical block addressing (LBA)**
 - since ATA-6 disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.
 - no guarantees, but the block-to-CHS mapping tends to be sequential



Disk Management

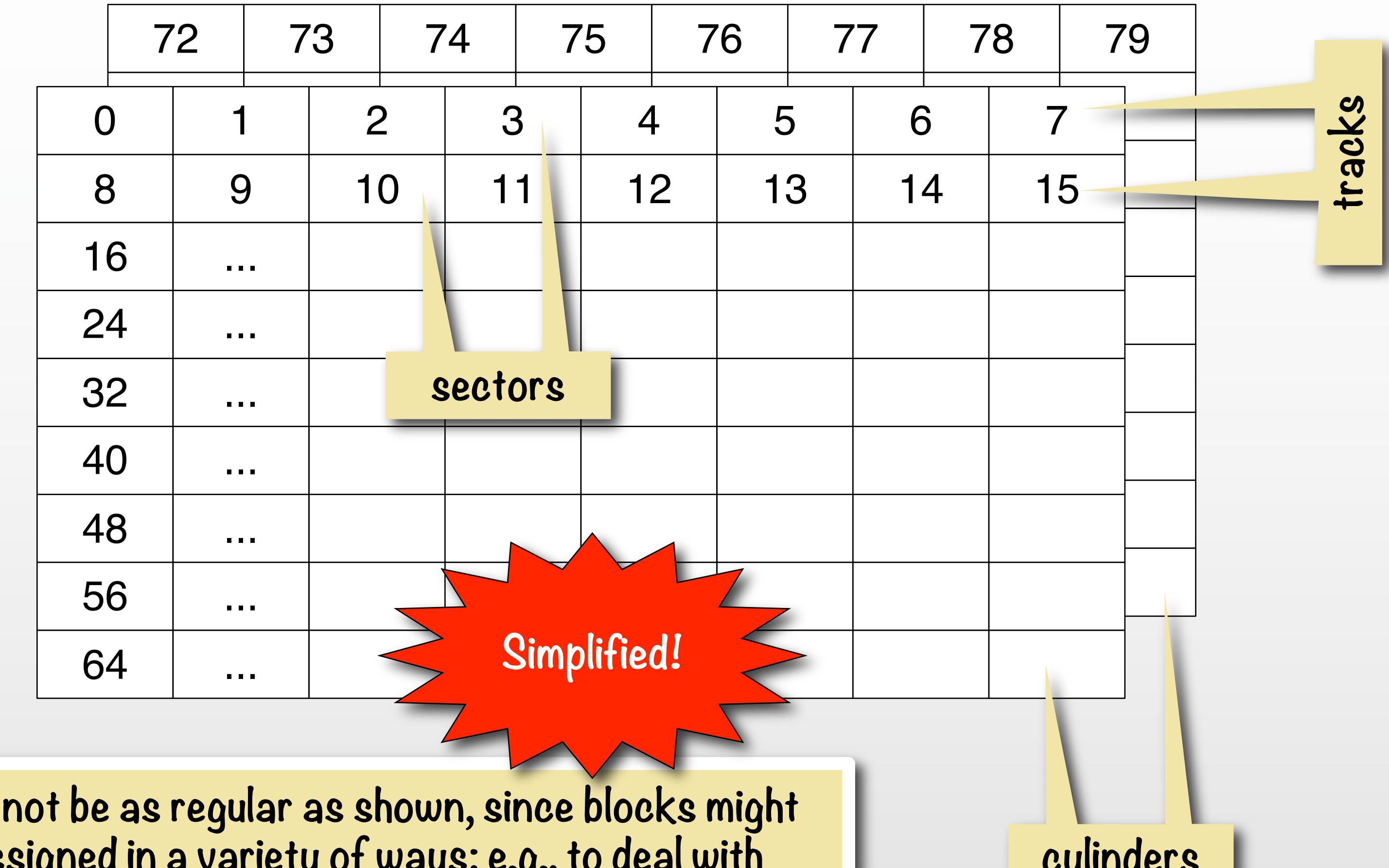


- Disk vendors create basic disk structure through low-level formatting (LLF)
 - dividing a disk into LLF blocks that the disk controller can read and write
 - usually a block is the size of a sector
 - often the names are used interchangeably
 - this cannot be done by end users nowadays
- The operating system creates its own data structures on the disk to construct a "map" of the disk
 - boot code
 - partition table
 - partitions
 - can be used individually or in combination as independent storage volumes
 - hold specific file systems
- Some LLF blocks might be bad, so the operating system fixes the problems by:
 - sparing that remaps logical blocks pointing to bad LLF blocks with pre-arranged spare LLF blocks
 - slipping that remaps logical blocks to jump over bad LLF blocks ("slipping" to adjacent good LLF blocks)



Disk Structure

- The one-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder
- Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
 - not necessarily in order, as tagging sequential numbers sparsely may speed up access



May not be as regular as shown, since blocks might be assigned in a variety of ways; e.g., to deal with bad sectors, sector density, etc. Also, sectors in outer cylinders might have more capacity.

Disk Scheduling



- The operating system is responsible for using hardware efficiently
 - for the disk drives, this means having a fast access time and high disk bandwidth
- Maximizing **disk bandwidth** is one of the objectives
 - this is the total amount of data transferred, divided by the total time between the first request for service and the completion of the last transfer.
- OS tries to minimize seek time
 - seek time \approx seek distance (is proportional)
 - OS aggregates requests and reorders them to optimize disk access
- A number of algorithms exist to schedule the servicing of disk I/O requests.
- On the following slides, we illustrate the algorithms with a request queue for a disk with 200 tracks (0-199).

As we already discussed, access time has two major components: seek time and rotational latency.

The rotational latency is determined by the disk manufacturer and cannot be controlled by the OS

Disk Scheduling: First Come First Serve (FCFS)

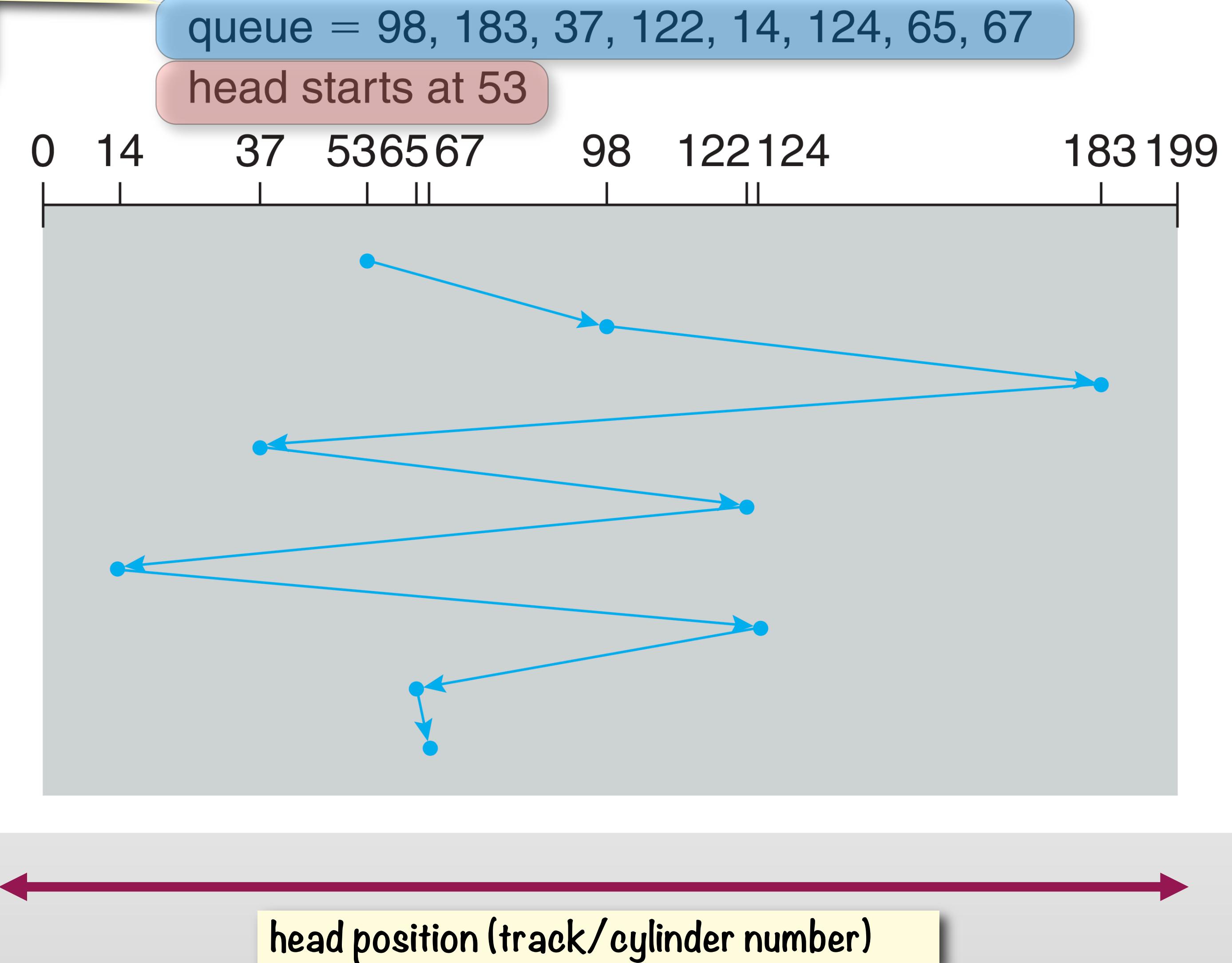


the OS received the following aggregated list of disk access requests

- Illustration shows total head movement of 640 cylinders.

a measure of movement efficiency

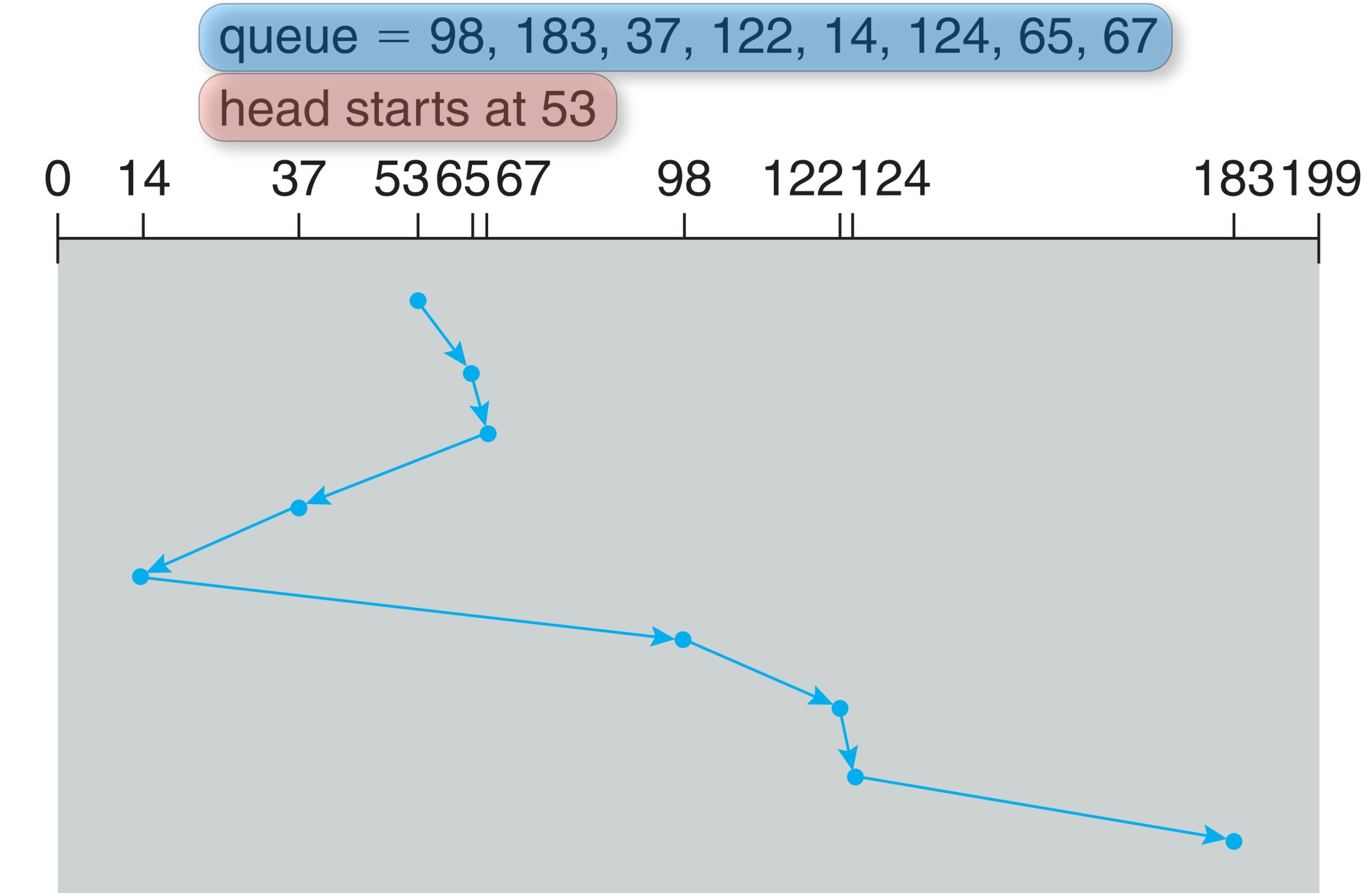
- a.k.a. FIFO
- a.k.a. NOOP



Disk Scheduling: Shortest Seek Time First (SSTF)



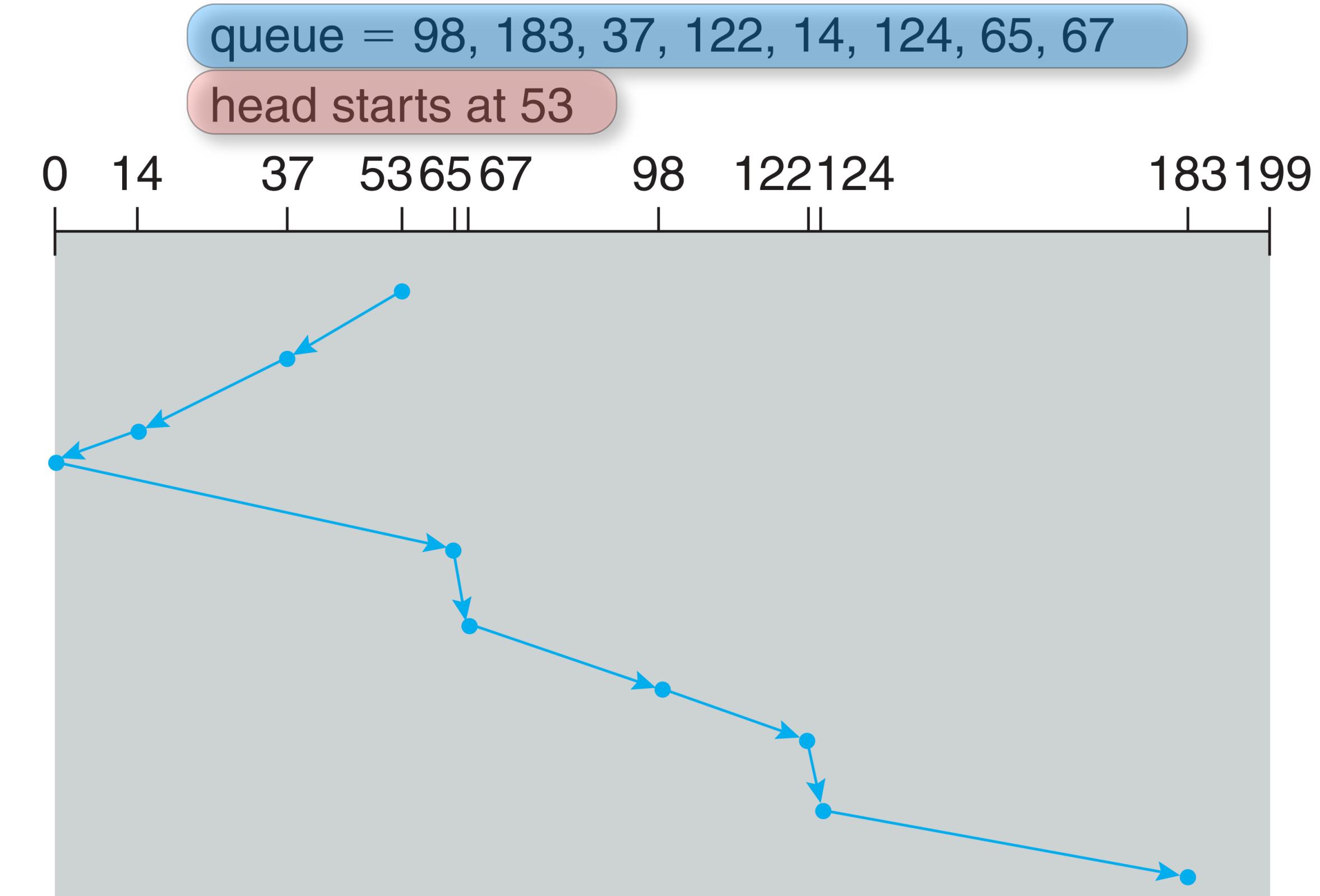
- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of shortest job first (SJF) scheduling
 - this is a greedy algorithm, so it may cause starvation of some requests
 - consider that the queue is dynamic in a live system
- Illustration shows total head movement of 236 cylinders.





Disk Scheduling: SCAN (a.k.a. Elevator Algorithm)

- SCAN -- acronym from scanning tracks
- The idea is to minimize changes of direction
 - sweep one way, then the other, and so on
- 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.
- A.k.a. elevator algorithm.
 - why?
- Illustration shows total head movement of 236 cylinders

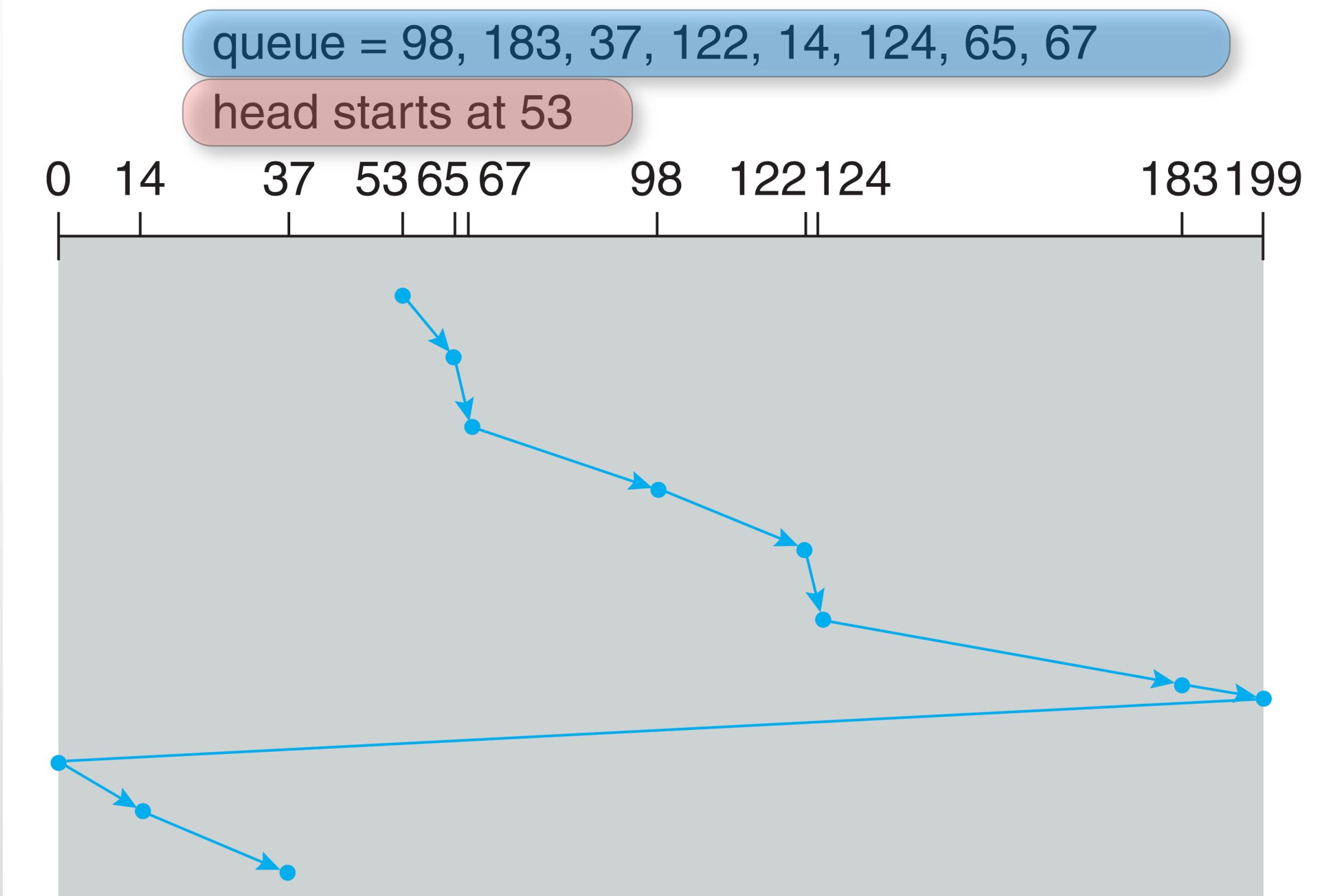


- Favors recently “visited” sectors
 - usually not the best approach as it checks for the requests at the end that was just recently checked



Disk Scheduling: Circular SCAN (C-SCAN)

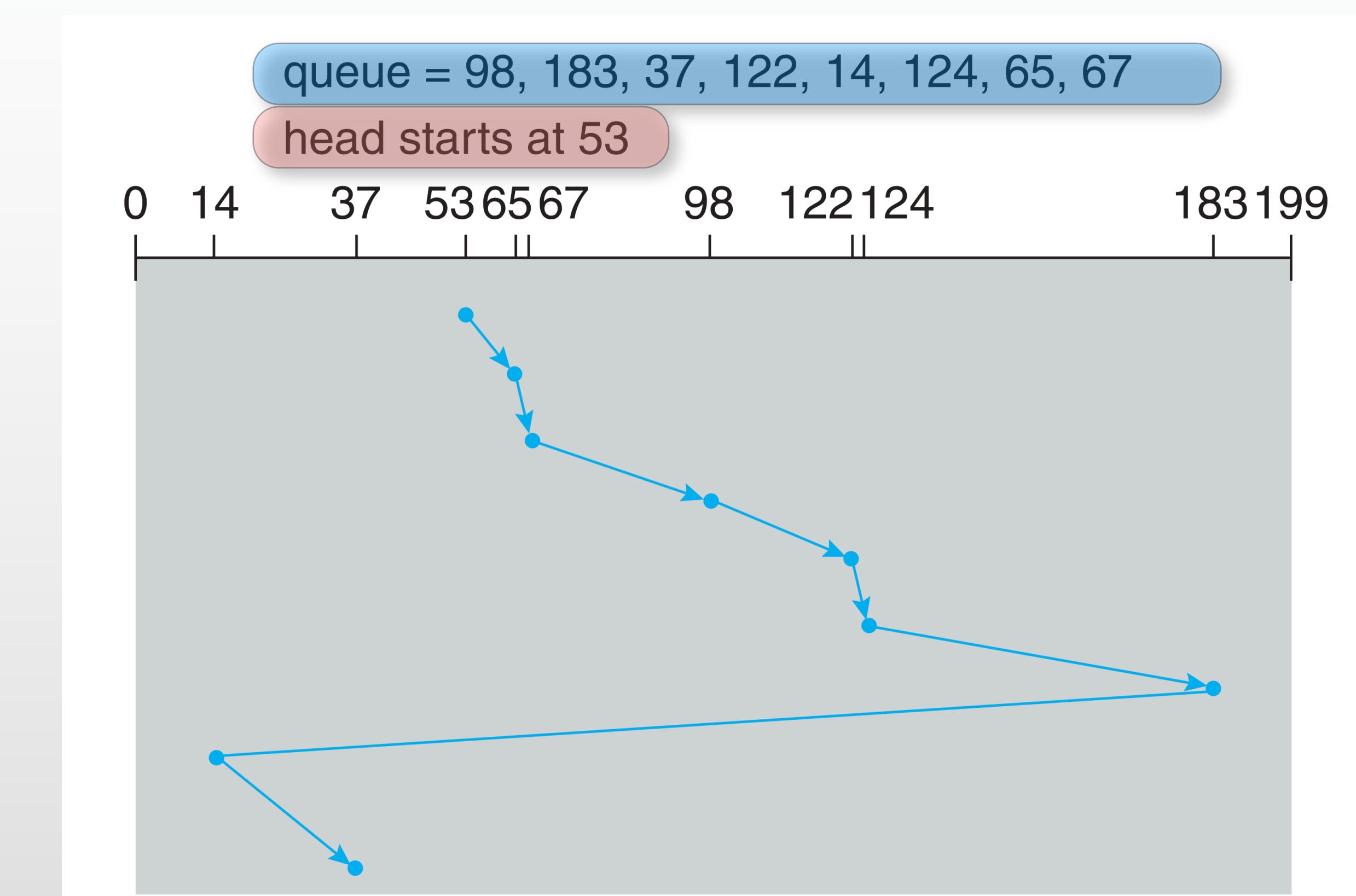
- The head moves from one end of the disk to the other servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
 - this technique provides a more uniform wait time than SCAN
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.
- Illustration shows total head movement of 183 cylinders.
 - without counting the return



Disk Scheduling: LOOK and C-LOOK



- LOOK is a version of SCAN in which the 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.
 - improvement from 236 to 208 cylinders
- C-LOOK is a version of C-SCAN with similar scanning border policy
 - Illustration shows total head movement of 153 cylinders (C-LOOK).



OS Disk Scheduling vs. Disk Controller Scheduling



- Q: Why not have driver controller do all the scheduling?
 - e.g., could be more efficient; some controllers do exactly that
 - however...
- A: OS knows about constraints that the disk doesn't
 - for example:
 - demand paging takes priority over application I/O
 - writing is more important than reading if the cache is almost full
 - OS may need guaranteed write ordering
 - e.g. journaling, data flushing
- Hard to address rotational latency from the OS
 - disk controllers perform some optimization
 - e.g., request ordering, coalescing (aggregating), sector numbering schemes

Linux I/O Schedulers



- **NOOP**
 - no optimization (**none**); all is FCFS
 - good for non-rotational media
- **Linus Elevator** (default in 2.4 kernel)
 - merges adjacent requests and sorts request queue by block (sector) number
 - may lead to starvation of requests with large block numbers
- **Deadline**
 - employs multiple queues to minimize seeks while ensuring requests don't starve
 - adds expiration timer to requests
 - expired requests move from elevator to deadline queue
 - process requests from the deadline queue sorted by expiration time
 - if deadline queue empty, then use the elevator queue sorted by block numbers
- **Anticipatory**
 - acts like deadline scheduler, but pauses for a few milliseconds after a read request to see if another request is made from nearby blocks
 - addresses the deceptive idleness syndrome
 - a program may be processing the read data before asking for more reads from following sectors
- if prediction is correct, the gain in efficiency is substantial, while the loss is small
- **Completely Fair Queueing (CFQ)**
 - maintains a queue for each process
 - queues are serviced round-robin, usually picking 4 requests from each queue at a time
- **mq-deadline**
 - deadline specialized for multi-core
 - the only no-noop option in Linux kernel 5.x

```
root@aj:/home/aj# cd /sys/block/sda/queue/
root@aj:/sys/block/sda/queue# cat scheduler
[mq-deadline] none
```

- When to use which?
 - Deadline - good for lots of seeks, critical workloads
 - Anticipatory - good for servers (lots of reading)
 - CFQ - good for desktops, multimedia (e.g., ensuring audio buffers are full)
- Used to be fairly easy to change a scheduler, but it is quite complicated now:

<https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers>

Sorting I/O Requests



- Sorting can be done by:
 - name/path
 - usually, blocks (sectors) allocated for a file are close to each other
 - fast, but approximate
 - inode (a.k.a. file descriptor)
 - usually: for $i < j$, data block numbers of a file with inode i are smaller than data block numbers of a file with inode j
 - faster; still approximate
 - block number
 - obtain all blocks of a file, and then group I/O operations for contiguous sequences
 - accurate, but slowest

```
fstat (fd, &buf);
// use buf.st_ino for sorting

fstat (fd, &buf);
// number of file blocks buf.st_blocks
// get block number for each subsequent
// block in the file
for (int i = 0; i < buf.st_blocks; i++) {
    int block_num = i;
    // block_num is an in-out parameter
    // FIBMAP requests FIle Block MAPping
    ioctl (fd, FIBMAP, &block_num);
    // then, use the use block_num for sorting
}
```

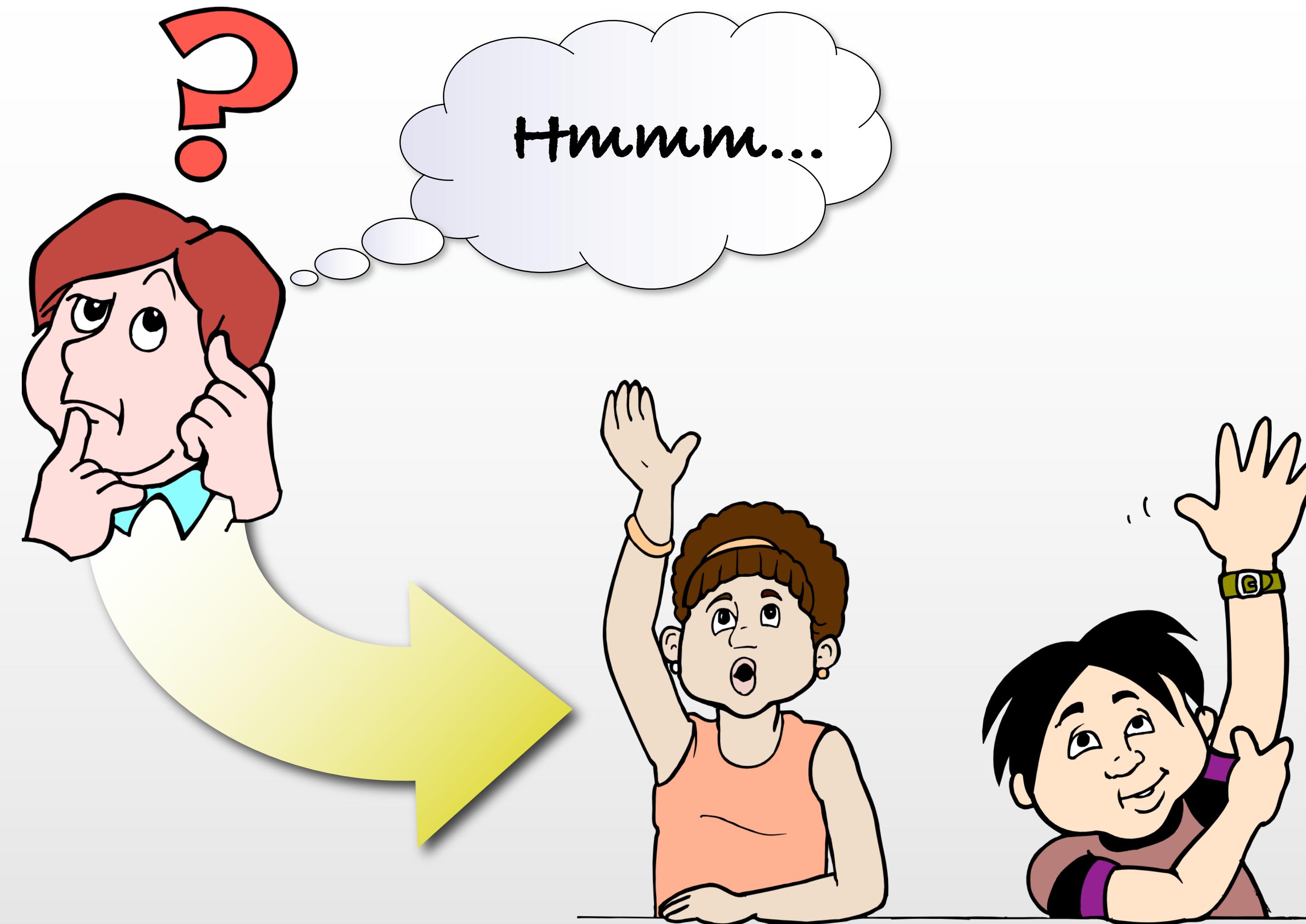
Solid State Drives



- Utilize the solid-state memory
 - use non-volatile flash memory
 - flash drives have usually some small DRAM that is used for caching, swap file, etc.
- Emulate interface to hard drives
 - can be accessed through “standard” hard drive interfaces (e.g., SATA)
- Use similar logical organizational structures to hard drives
 - e.g., paging
- Good
 - faster to start, read and write
 - with DRAM cache
 - no noise, spin
 - mechanical reliability (SLC better than MLC; single- vs. multi-level cell)
 - file fragmentation not a problem due to the direct access to any location
 - problems usually happen on writes, so writes can be repeated in good cells
 - less power usually
 - higher power consumption on SSDs with DRAM caches
- Bad
 - still expensive
 - still relatively low capacity
 - limited write cycles (up to 10,000 for MLC, and 100,000 for SLC)

one bit per cell
⇒ 0 or 1

usually 4 levels
⇒ equivalent of two bits
⇒ lower cost per bit
LESS RELIABLE!



COMP362 Operating Systems
Prof. AJ Biesczad