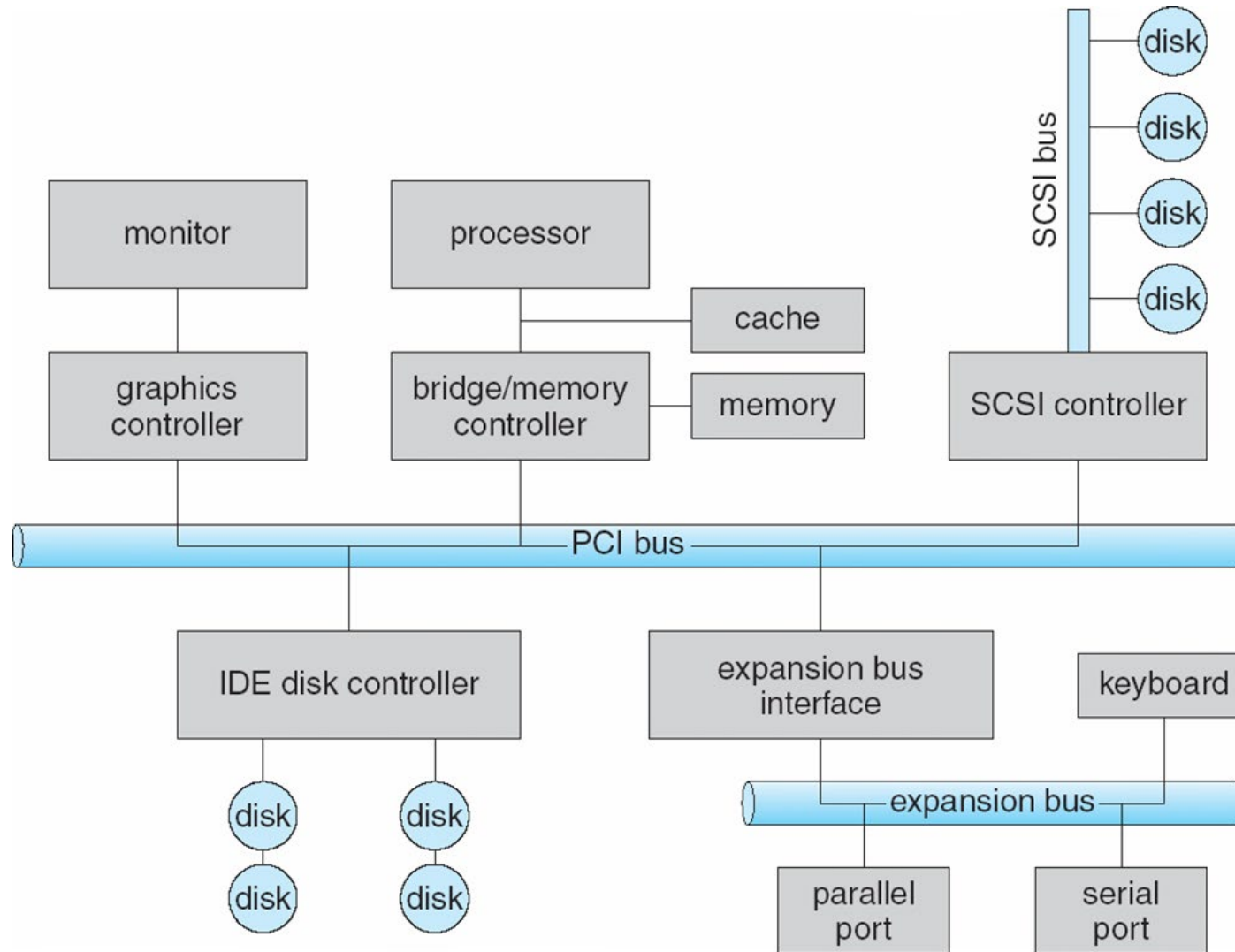


Part 9: I/O System & Disk

- I/O Hardware
- I/O System Design and Implementation
 - Design Objectives
 - Kernel I/O Structure
 - Kernel I/O Subsystem
 - Performance Consideration
- Disk
 - Disk Structure
 - Disk Scheduling
 - *FCFS, SSTF, SCAN, C-SCAN, C-LOOK*
 - Disk Management
 - Disk Reliability

I/O Hardware



I/O Hardware

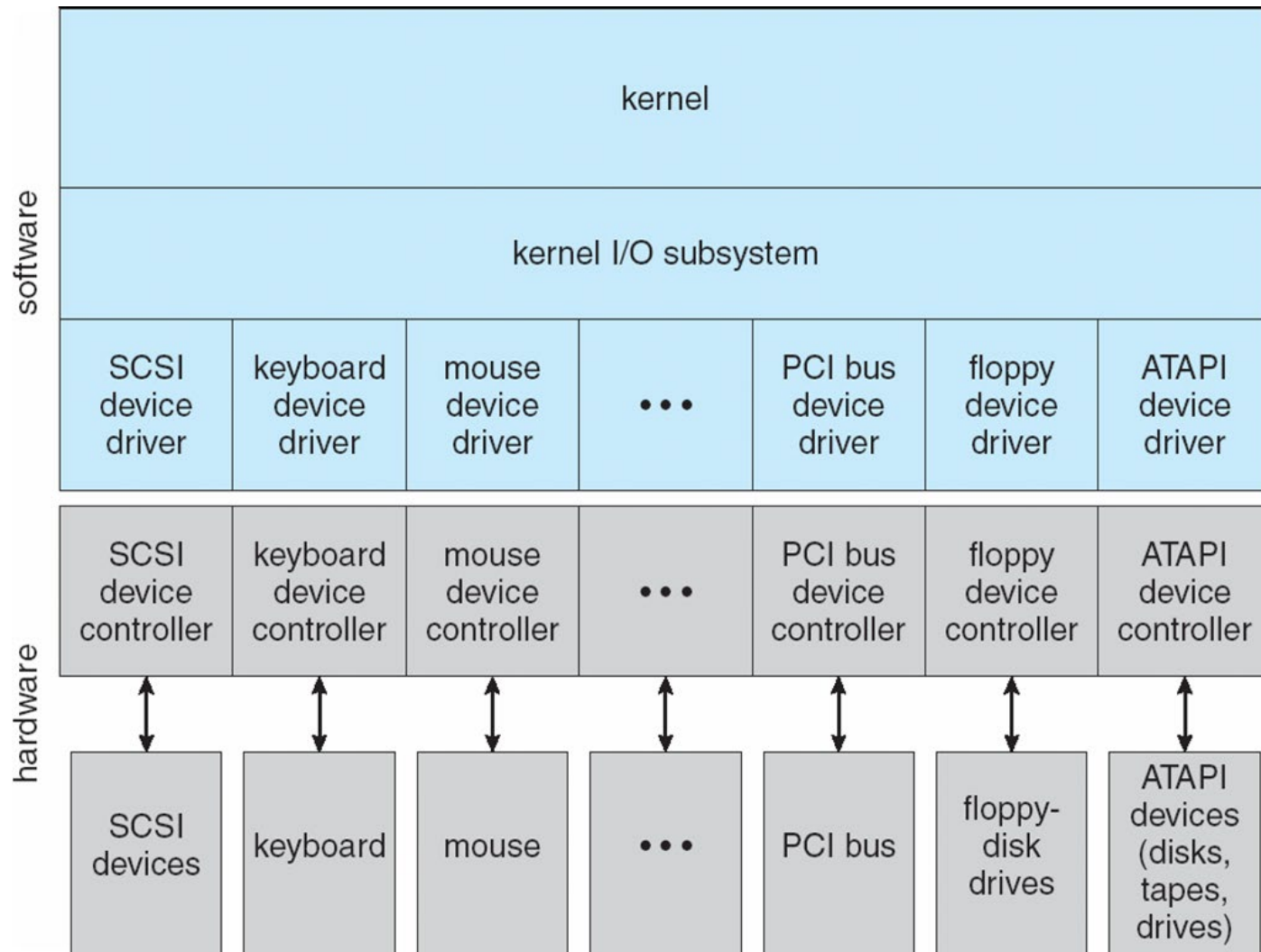
- Devices vary in many dimensions

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

I/O System Design Objectives

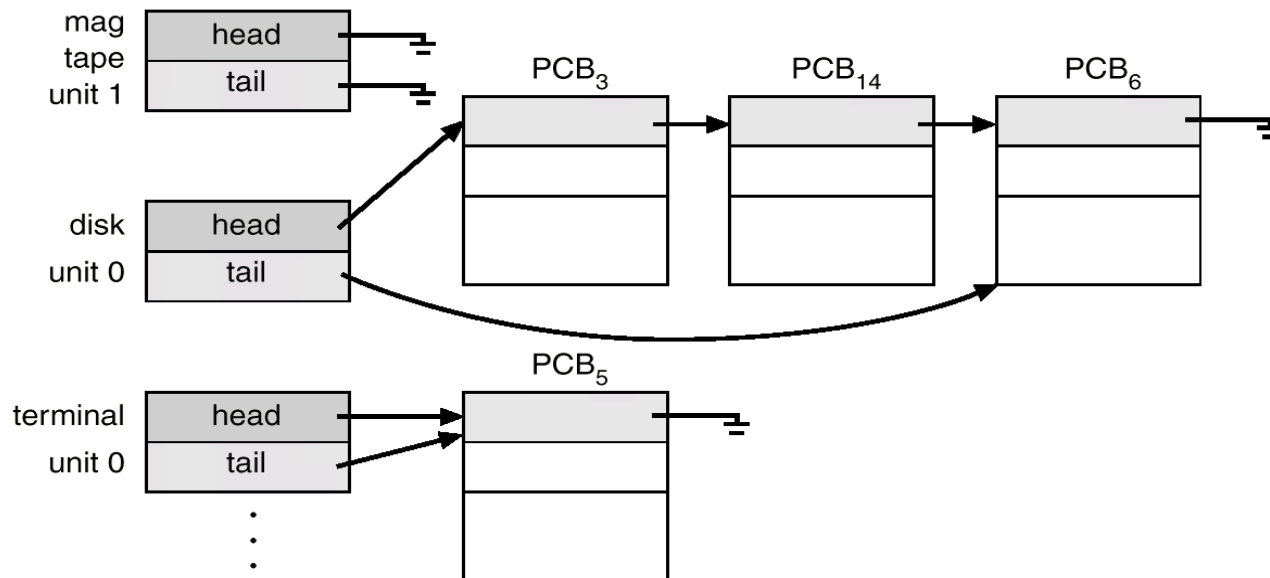
- **Efficiency**
 - Important because I/O operations often form a bottleneck
 - Most I/O devices are extremely slow compared with main memory and the processor
 - The area that has received the most attention is disk I/O
- **Generality**
 - Desirable to handle all devices in a uniform manner
 - Applies to the way processes view I/O devices and the way the operating system manages I/O devices and operations

Kernel I/O Structure



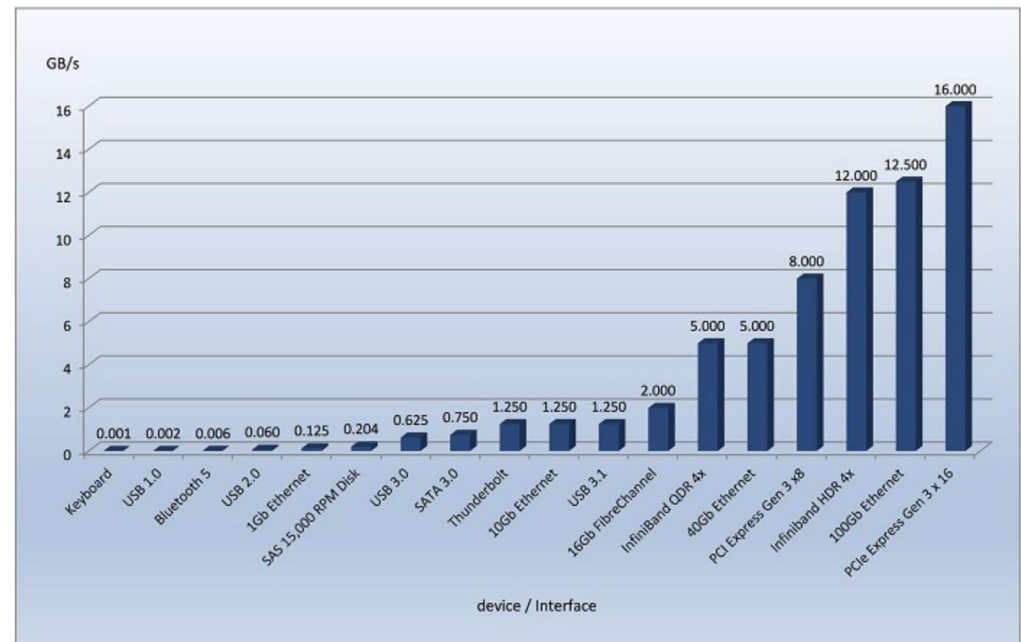
Kernel I/O Subsystem

- I/O Scheduling
 - Schedule I/O request through rearranging the order of service, e.g. disk scheduling
 - Can improve the system efficiency



Kernel I/O Subsystem

- Buffering
 - Store data in memory while transferring between devices
 - To cope with device speed mismatch
 - To cope with device transfer size mismatch, e.g. in networking

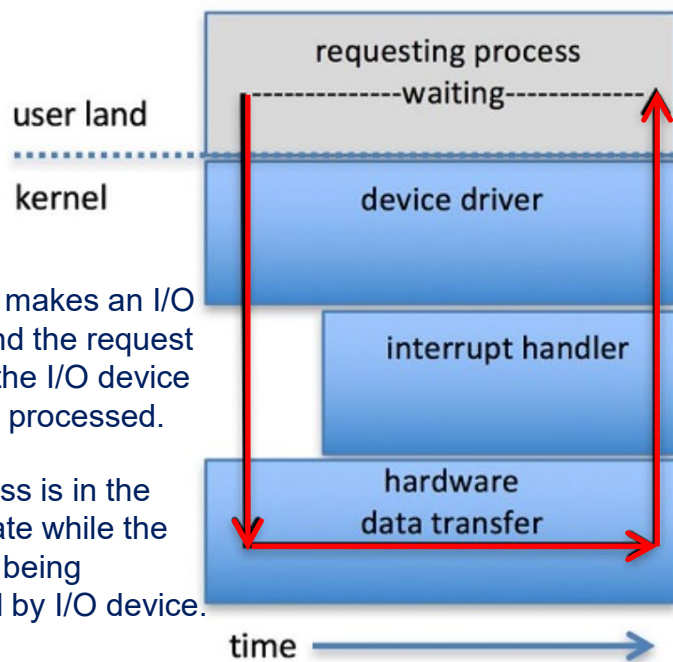


Kernel I/O Subsystem

- Caching
 - Fast memory holding copy of data
 - To improve I/O efficiency for files that are being written and reread rapidly.
- Spooling
 - Hold output for a device
 - Used in device that can serve only one request at a time, e.g. printing

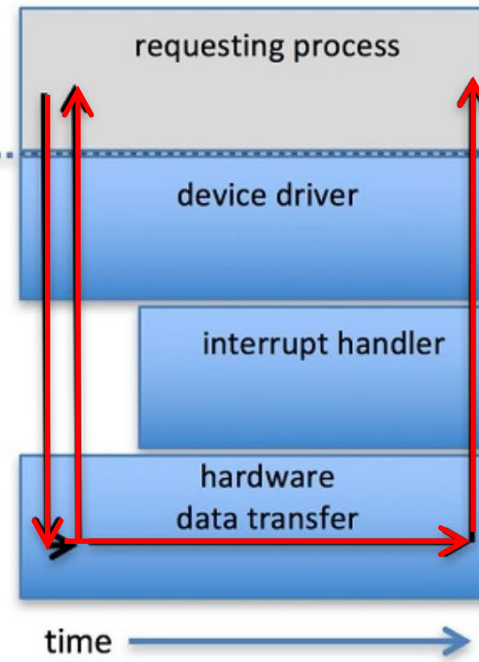
Performance Consideration

• Application I/O Interface



1. A process makes an I/O request and the request is sent to the I/O device where it is processed.
2. The process is in the waiting state while the request is being processed by I/O device.
3. The process is put back to the ready queue when the I/O request is completed.

(a)
Blocking
I/O

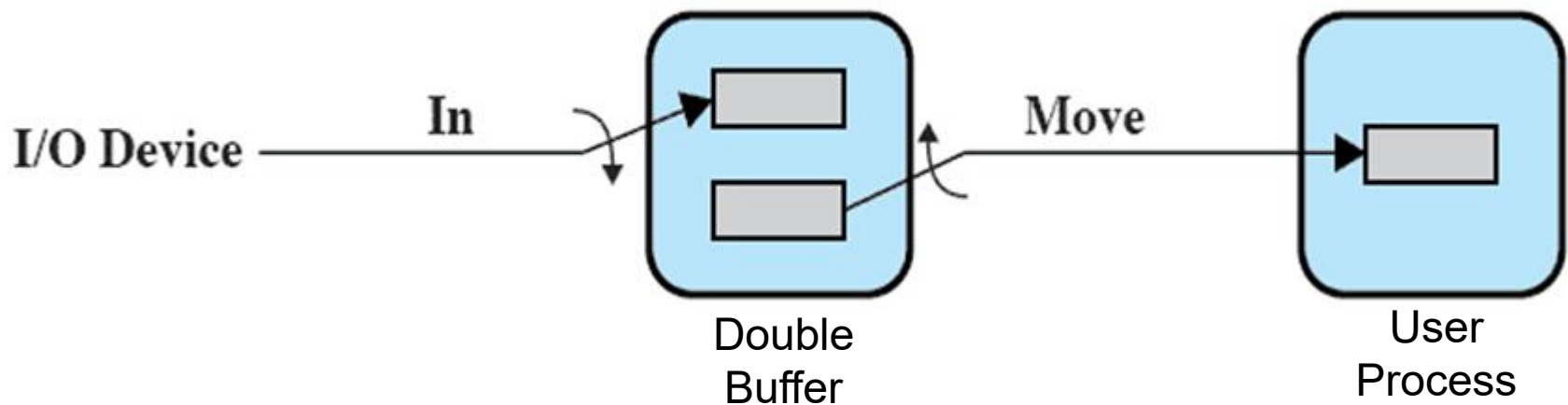


(b)
Asynchronous
I/O

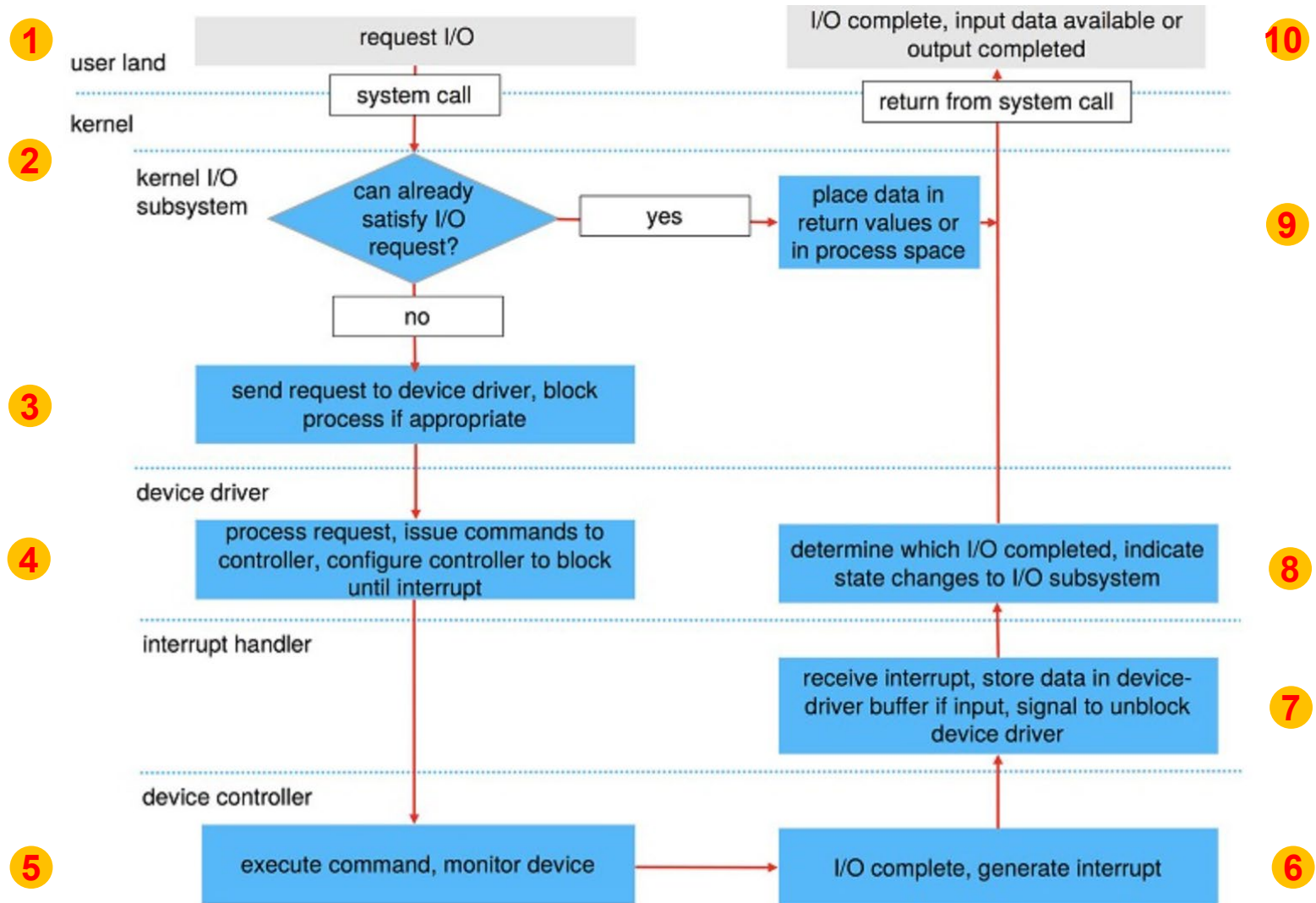
1. A process makes an I/O request and the request is sent to the I/O device where it is processed.
2. The system call returns after the request is queued at I/O device.
3. The requesting process continues to execute its code while the request is being processed by I/O device.
4. When the I/O completes, the requesting process is signaled.

Performance Consideration

- Using asynchronous I/O and double-buffer to improve performance



Performance Consideration



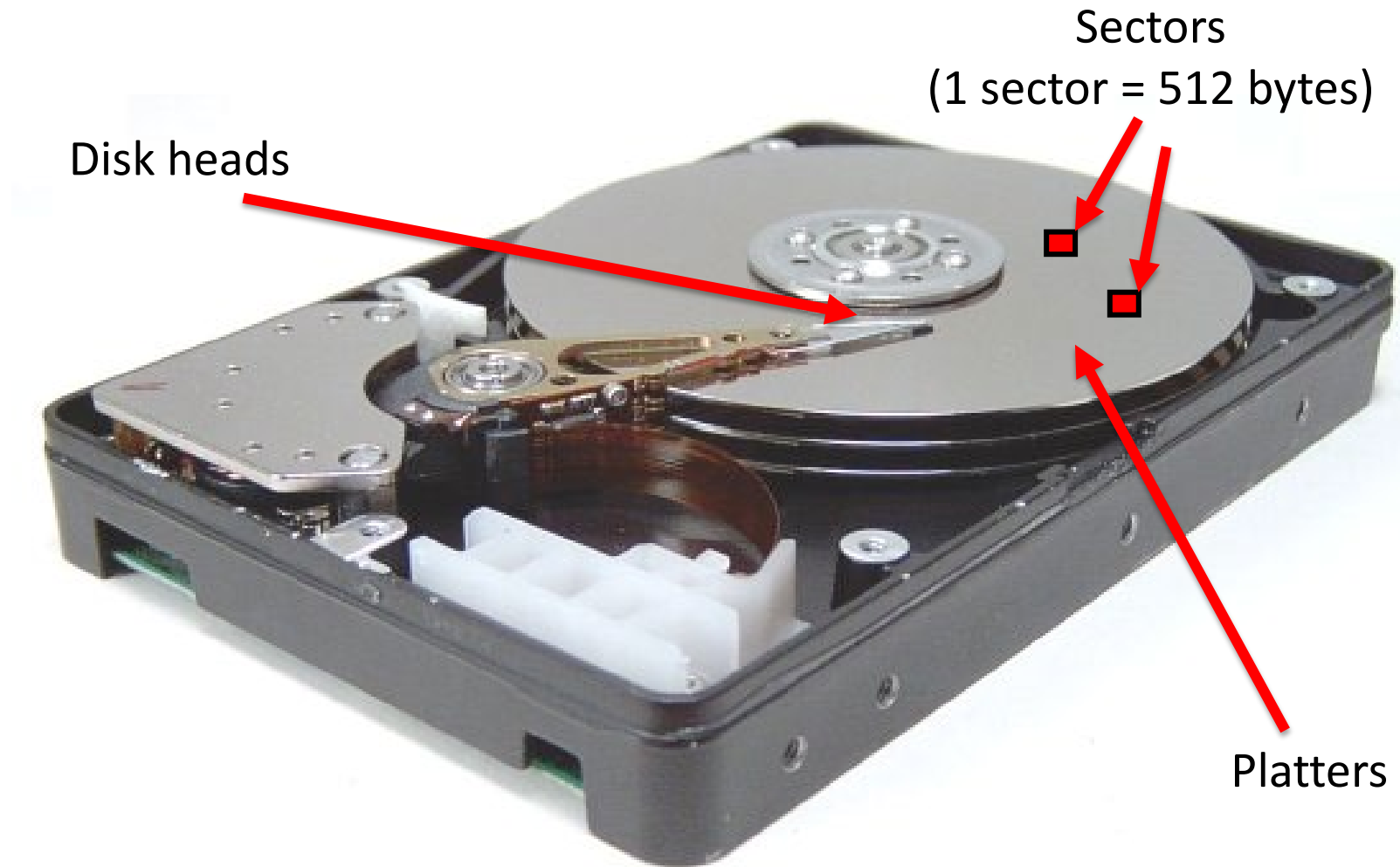
Performance Consideration

- I/O is a major factor in system performance
 - It demands CPU to execute device driver code
 - Context switches due to interrupts may stress CPU
 - Data copying between I/O controllers and physical memory also affect system performance.

Part 7: I/O System & Disk

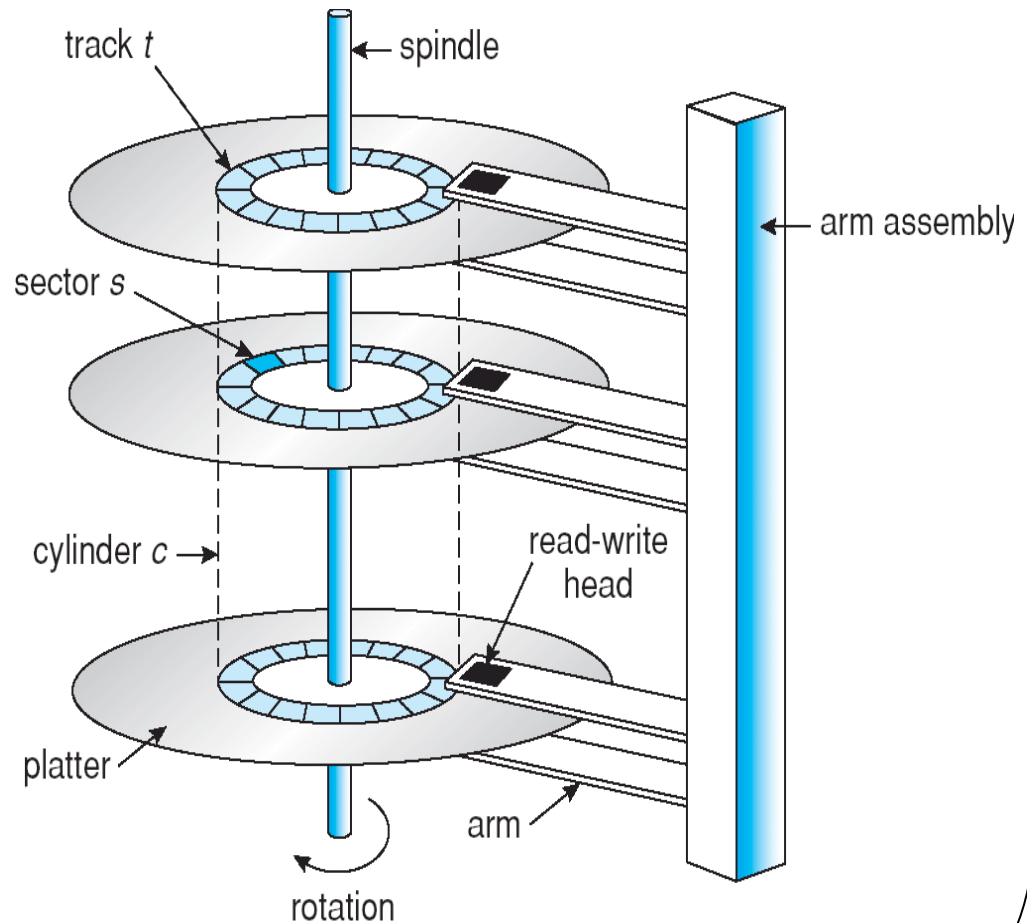
- I/O Hardware
- I/O System Design and Implementation
 - Design Objectives
 - Kernel I/O Structure
 - Kernel I/O Subsystem
 - Performance Consideration
- Disk
 - Disk Structure
 - Disk Scheduling
 - *FCFS, SSTF, SCAN, C-SCAN, C-LOOK*
 - Disk Management
 - Disk Reliability

Disk Structure



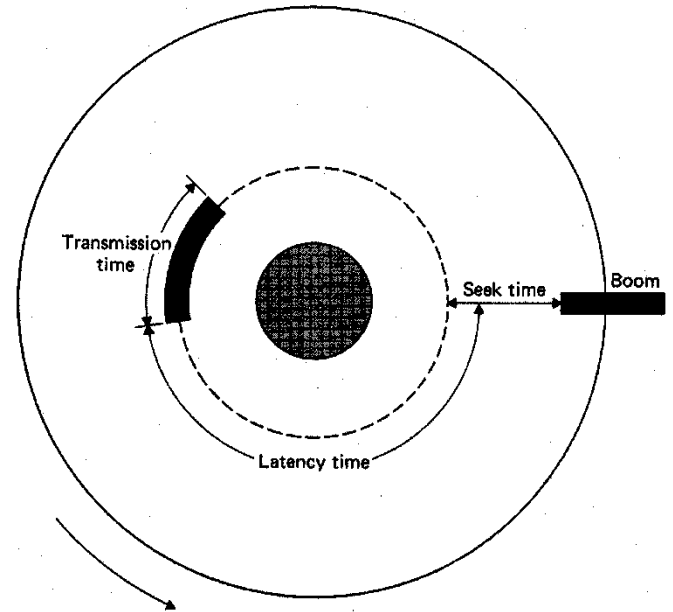
Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of *physical blocks*, where the physical block is the smallest unit of transfer (usually 512 bytes)
- The array of physical blocks is mapped into the sectors of the disk sequentially



Disk I/O

- To read a requested sector, disk drive needs to position the disk head and perform data transfer.
- Position time has two major components
 - *Seek time* for the disk to move the heads to the track containing the desired sector (5 -25 milliseconds)
 - *Rotational latency* for the disk to rotate the desired sector to the disk head (8 milliseconds)

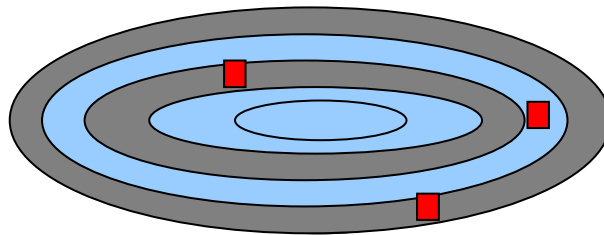


Disk I/O (Cont.)

- How long to read or write N sectors?
 - Positioning Time + Data Transfer Time (N)
 - Positioning time: *Seek Time + Rotational Latency*
 - Transfer time: $(N / \text{Number of Sectors per Track}) \times \text{Full Rotation Time}$
- Some parameters:
 - *Seek time*: 10 ms
 - *Full rotational time*: 8 ms (*Rotational latency*: 4 ms)
 - *Sector size*: 512 Bytes
 - *Track size*: 1 K sectors per track

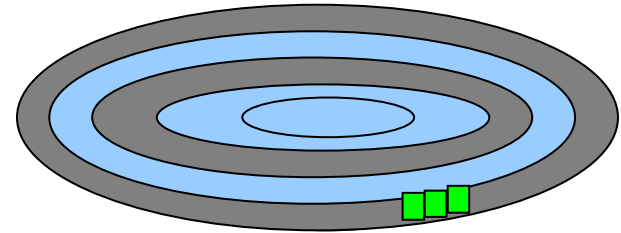
Disk I/O (Cont.)

- Random Request
 - Seek Time + Rotational Latency + Transfer Time
 $10\text{ ms} + 4\text{ ms} + (1/1024) \times 8\text{ms} = \sim 14\text{ ms}$
- Time and bandwidth for random request of size:
 - 4 KB = 8 sectors = $8 * 14\text{ ms} = 112\text{ ms}$
 - 128 KB = 256 sectors = $256 * 14\text{ ms} = 3.5\text{ secs}$
 - 1 MB = 2048 sectors = $2048 * 14\text{ ms} = 28\text{ seconds !!}$



Disk I/O (Cont.)

- Sequential Request



- 4KB? 8 sectors

- 1 positioning time = 14 ms
 - Transfer time = $(8 / 1024) \times \text{full rotational time} = 0.0625 \text{ ms}$
 - Total = ~14 ms

- 128 KB? 256 sectors

- Still fits in a track, hence 1 positioning time = 14 ms
 - Transfer time = $(256 / 1024) \times \text{full rotational time} = 2 \text{ ms}$
 - Total = 16 ms

- 1 MB? 2048 sectors

- Does not fit in a track, fits in 2 tracks (at best) or 3 tracks (at worst).
 - Read one track: positioning time + full rotation time = 14 ms + 8 ms = 22 ms
 - Read two tracks: $2 * 22 \text{ ms} = 44 \text{ ms}$ (vs. 28 seconds of random I/Os)

Disk Scheduling

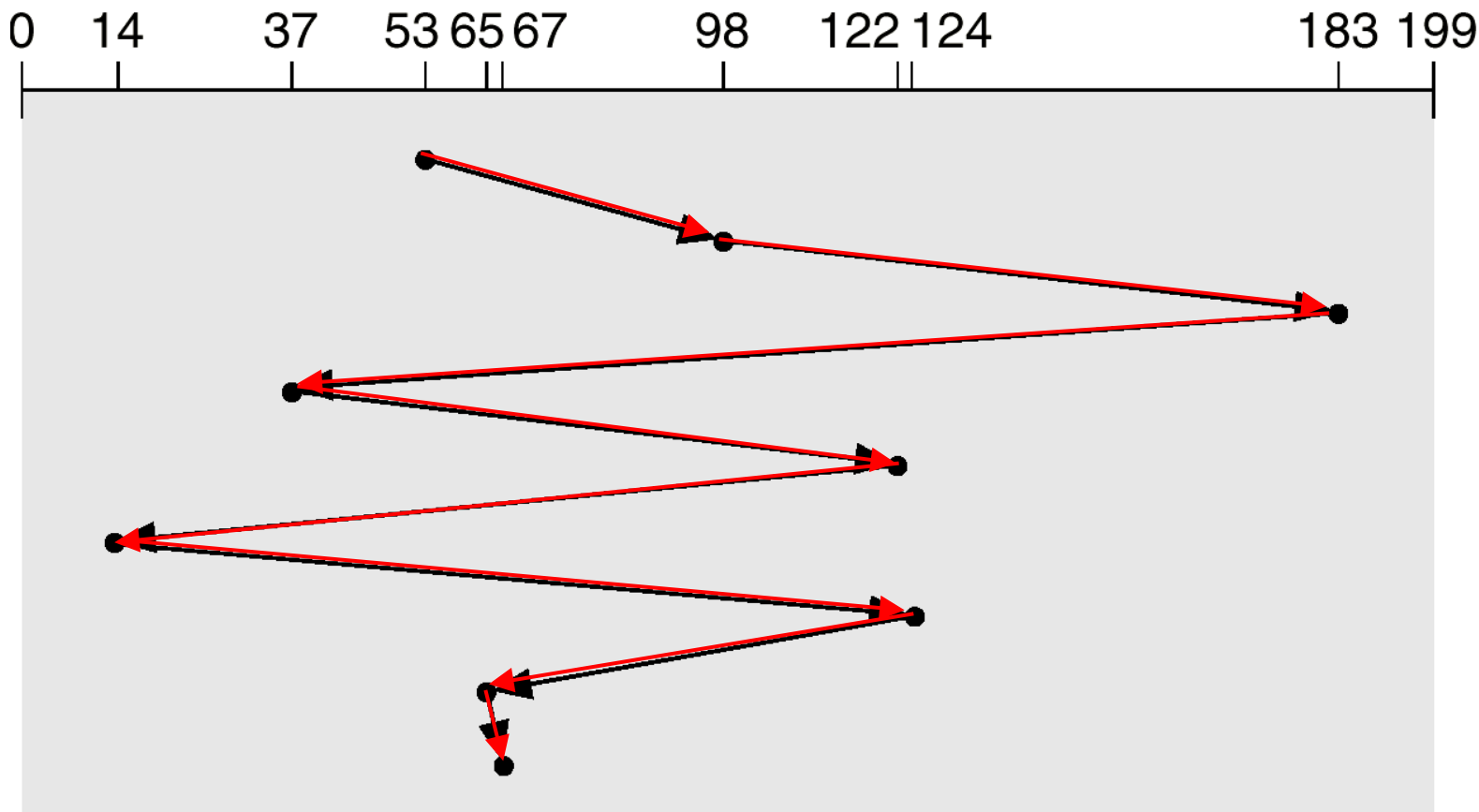
- The OS is responsible for using hardware efficiently — for disk drives this means having a fast access time and high disk bandwidth
- Minimize seek time
- Seek time \approx seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the 1st request and completion of the last transfer

Disk Scheduling (Cont.)

- Several algorithms exist to schedule the servicing of disk I/O requests
- We illustrate them with a request queue (0 -199)
98, 183, 37, 122, 14, 124, 65, 67
- Assume that head is initially at cylinder 53

First-Come-First-Served (FCFS)

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



FCFS (Cont.)

Total head movement =

$$\begin{aligned} &|98 - 53| + |183 - 98| + |37 - 183| + |122 - 37| + \\ &|14 - 122| + |124 - 14| + |65 - 124| + |67 - 65| \\ &= 640 \end{aligned}$$

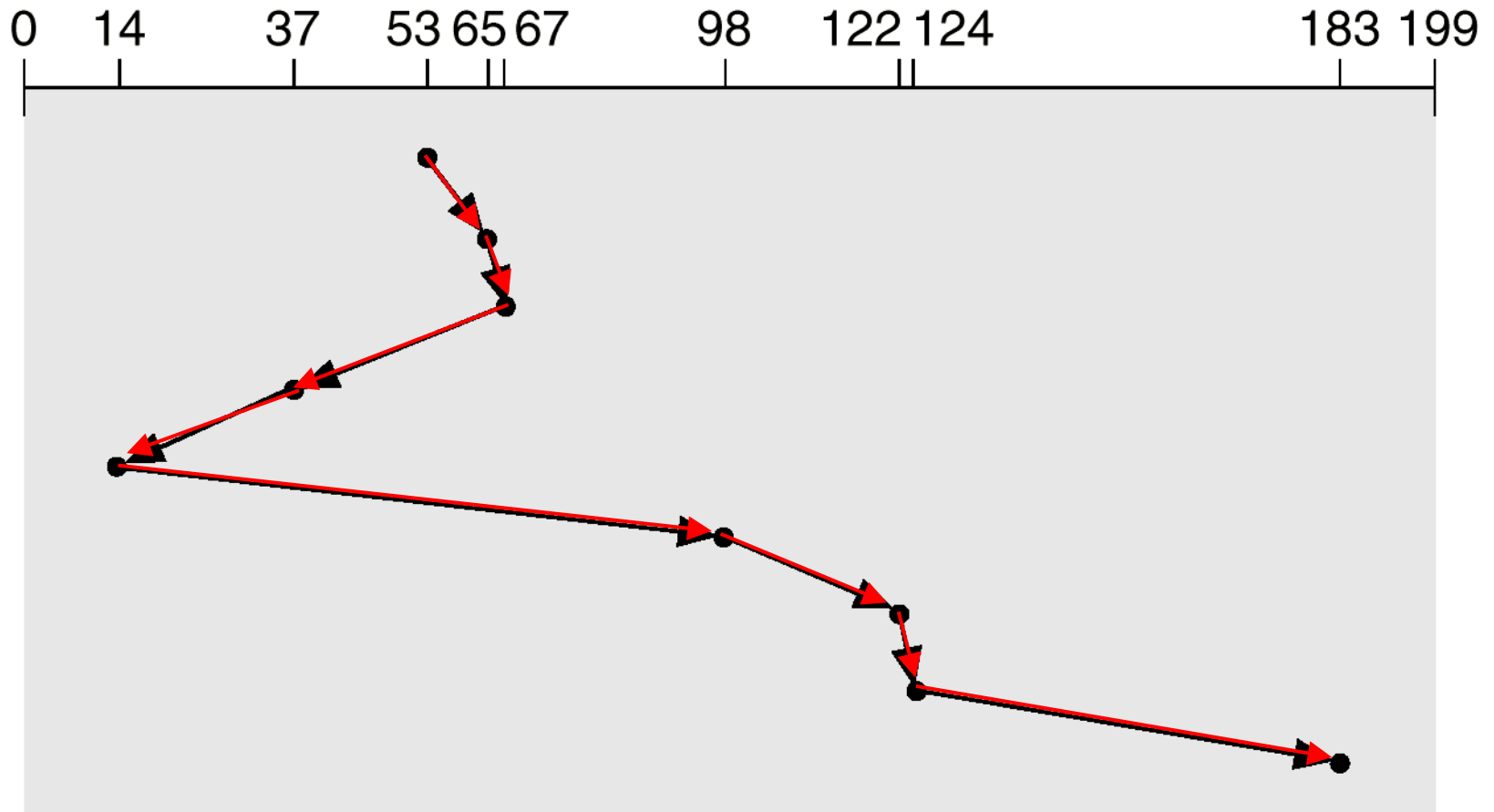
Problem: *FCFS may have wild swings back and forth!*

Shortest-Seek-Time-First (SSTF)

- Selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may have starvation under heavy load, since distant requests may never be serviced
- **Note:** requests are from processes executing concurrently

SSTF (Cont.)

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



SSTF (Cont.)

Total head movement =

$$\begin{aligned} &|65 - 53| + |67 - 65| + |37 - 67| + |14 - 37| + \\ &|98 - 14| + |122 - 98| + |124 - 122| + |183 - 124| \\ &= 236 \end{aligned}$$

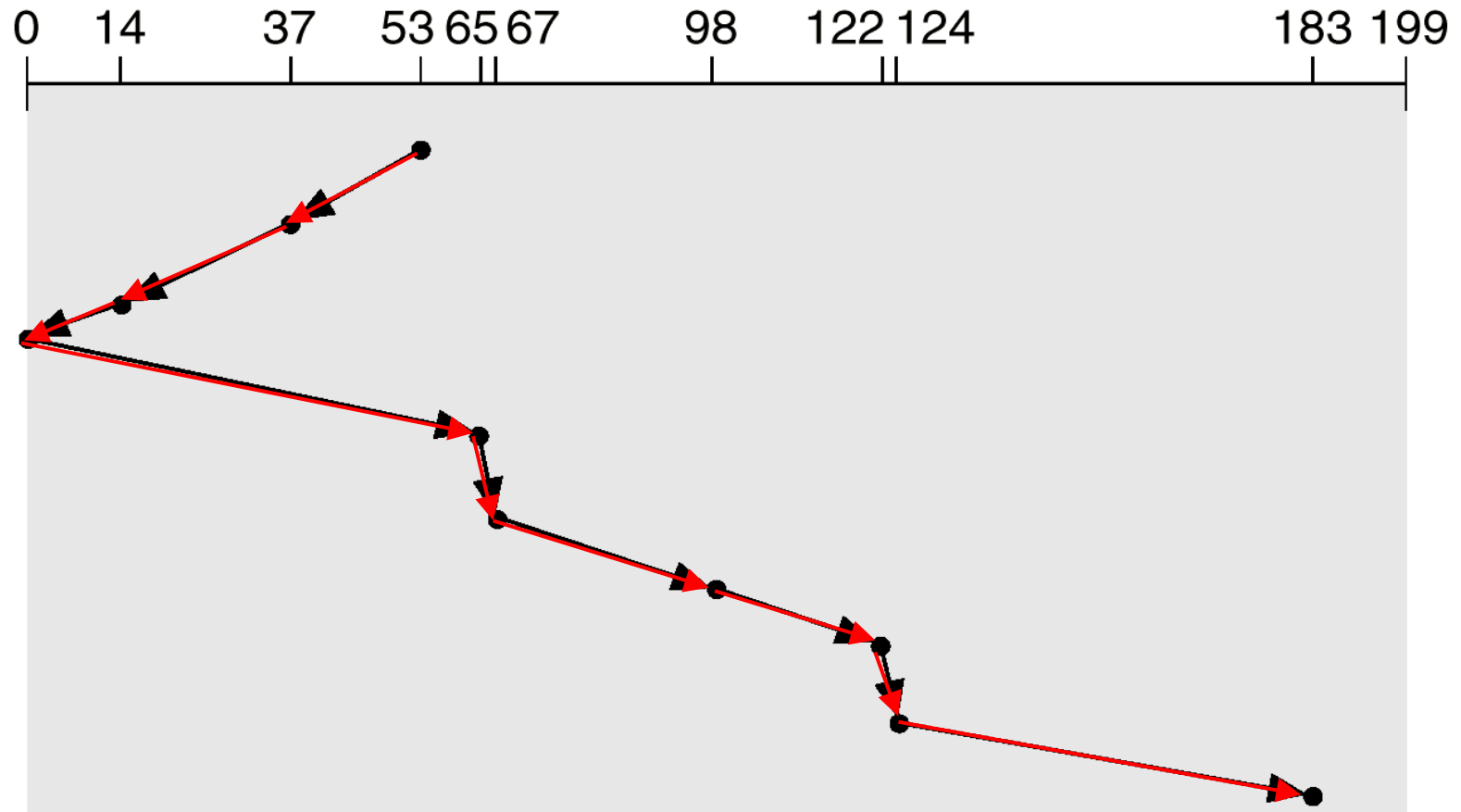
Problem: *SSTF may cause starvation!*

SCAN

- 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
- Also called the *elevator algorithm*
- Assuming here the head is moving toward 0
- Starvation unlikely but servicing requests may be delayed

SCAN (Cont.)

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



SCAN (Cont.)

Total head movement =

$$\begin{aligned} &|37 - 53| + |14 - 37| + |0 - 14| + |65 - 0| + |67 - 65| \\ &+ |98 - 67| + |122 - 98| + |124 - 122| + |183 - 124| \\ &= 236 \end{aligned}$$

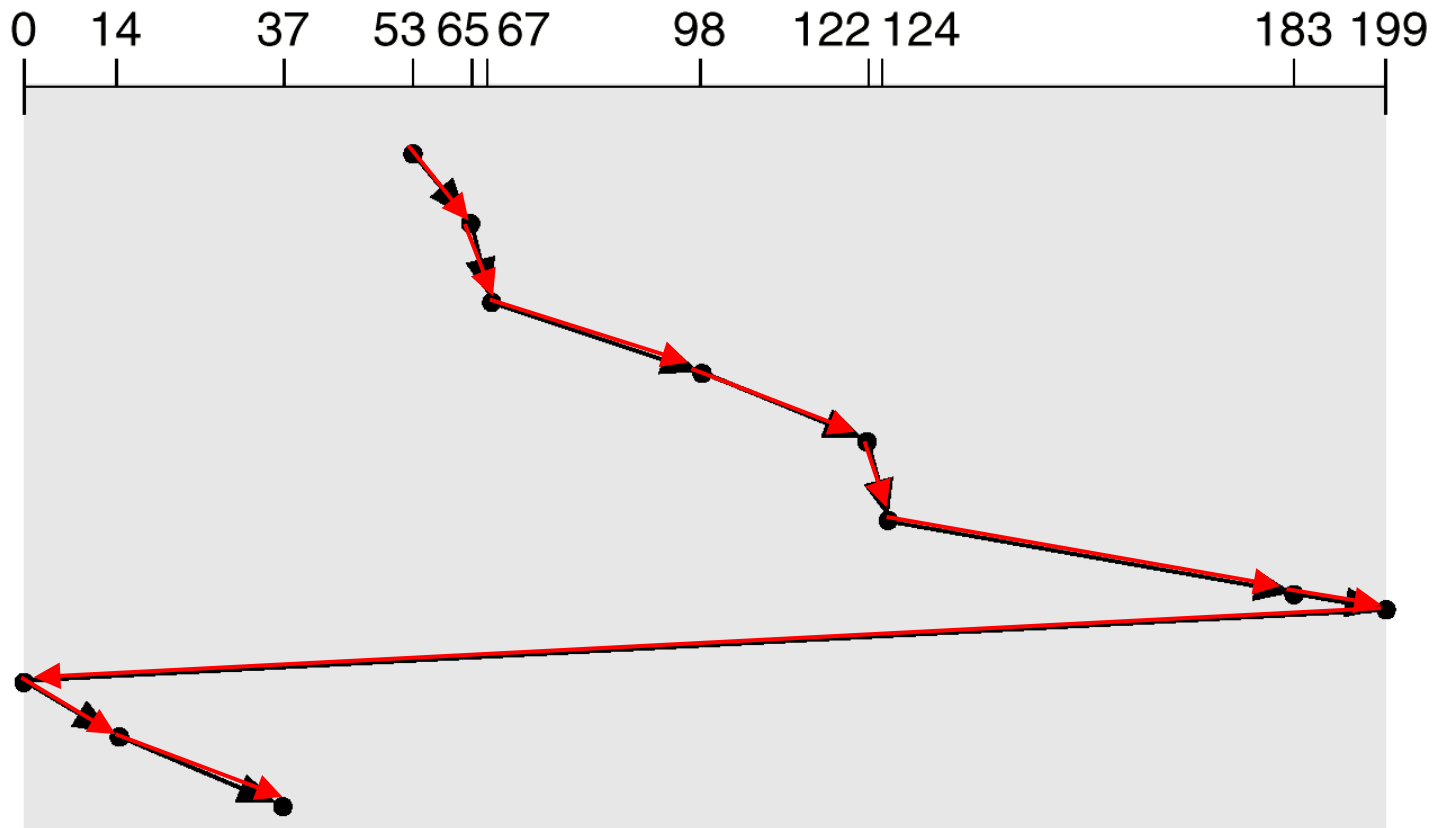
After the head reverses direction, does it need to serve requests in the reversed direction immediately?

Circular-Scan (C-SCAN)

- Head moves from low end to high end of the disk, servicing requests along
- When reaching the high end, head immediately returns to the beginning of the disk, without servicing any requests

C-SCAN (Cont.)

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



C-SCAN (Cont.)

Total head movement =

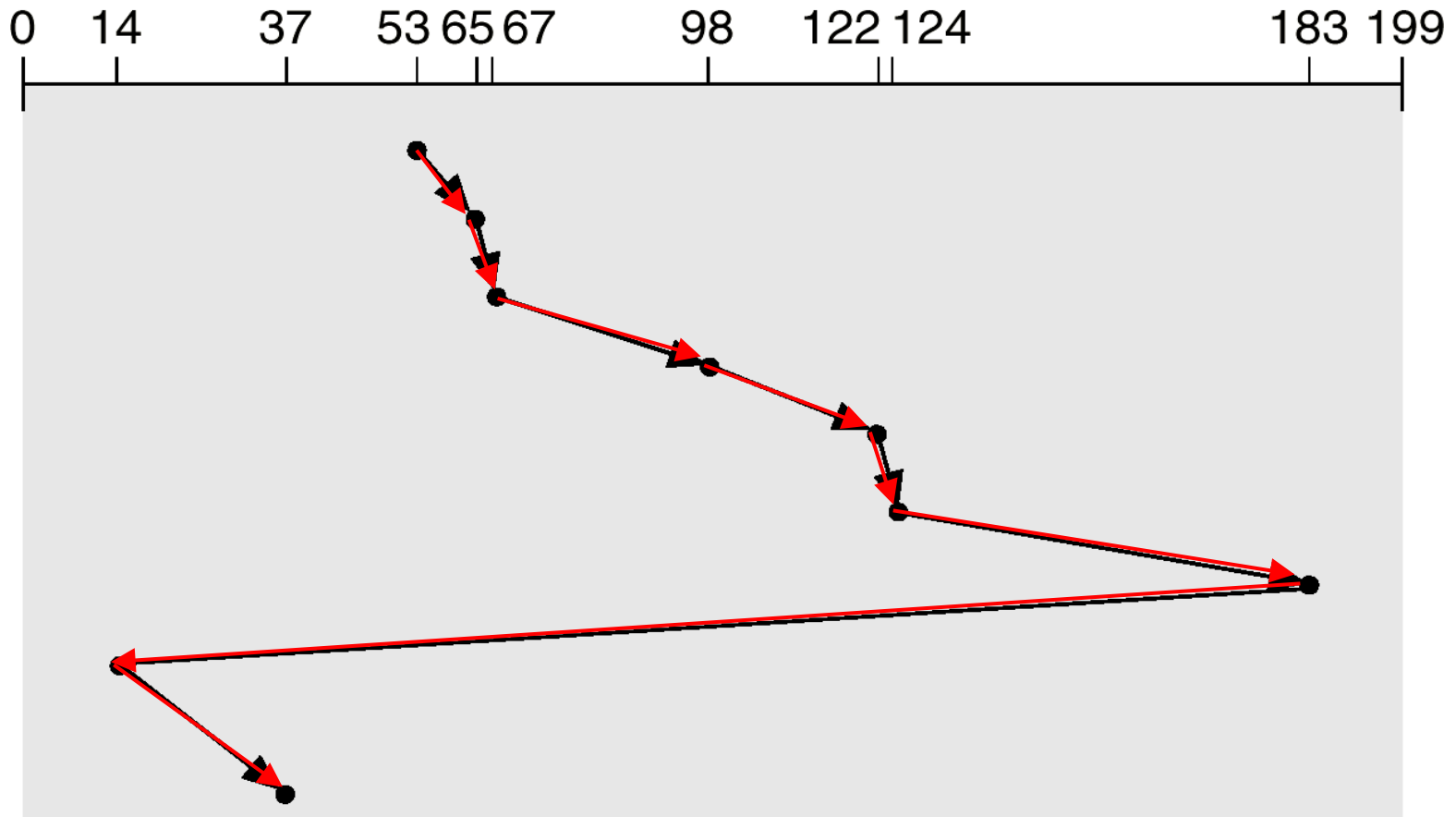
$$\begin{aligned} &|65 - 53| + |67 - 65| + |98 - 67| + |122 - 98| + \\ &|124 - 122| + |183 - 124| + |199 - 183| + \\ &|0 - 199| + |14 - 0| + |37 - 14| \\ &= 382 \end{aligned}$$

C-LOOK

- Version of C-SCAN
- Head moves from low end to high end of the disk, servicing requests along
- When reaching the last request, head returns immediately to the lowest cylinder # request
- From there it services the requests towards the high end of the disk

C-LOOK (Cont.)

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



C-LOOK (Cont.)

Total head movement =

$$\begin{aligned} &|65 - 53| + |67 - 65| + |98 - 67| + |122 - 98| + \\ &|124 - 122| + |183 - 124| + |14 - 183| + |37 - 14| \\ &= 322 \end{aligned}$$

Does SCAN, C-SCAN and C-LOOK really solve the starvation problem?

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN (or LOOK and C-LOOK) perform better for systems that place a heavy load on the disk (since starvation is unlikely)
- Performance depends on the number and types of requests

Selecting a Disk-Scheduling Algorithm (Cont.)

- Requests for disk service can be influenced by the file-allocation method:
 - A program reading a contiguously allocated file will generate requests that are close together
 - A linked or indexed file, may generate requests that are widely apart, resulting in greater head movement

Disk Management

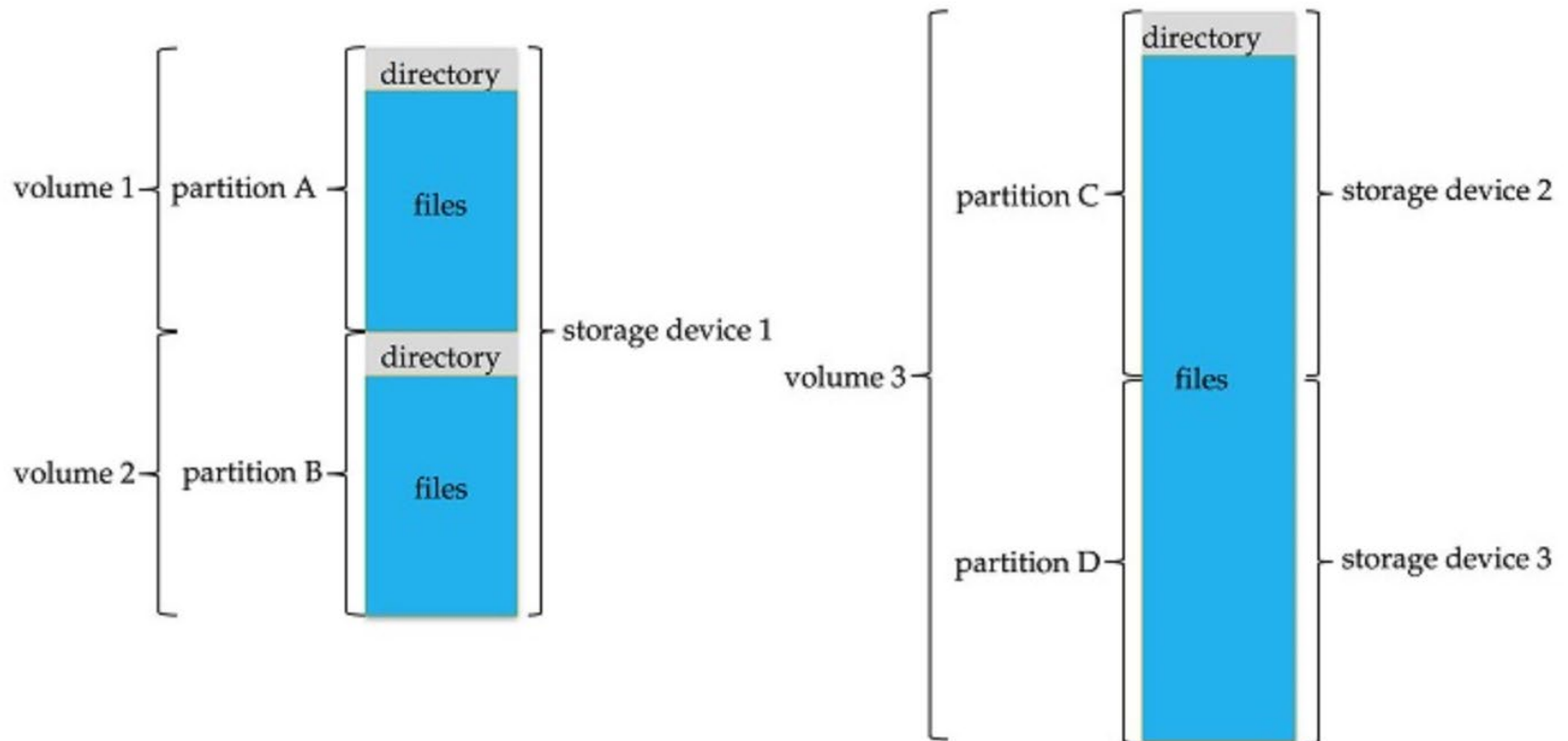
- *Low-level formatting (physical formatting)* —
Dividing a disk into sectors that the disk controller can read and write

- To use a disk to hold files, the OS needs to record its own data structures on the disk.

Two steps:

- *Partition* the disk into one or more groups of cylinders
- *Logical formatting* or “making a file system”

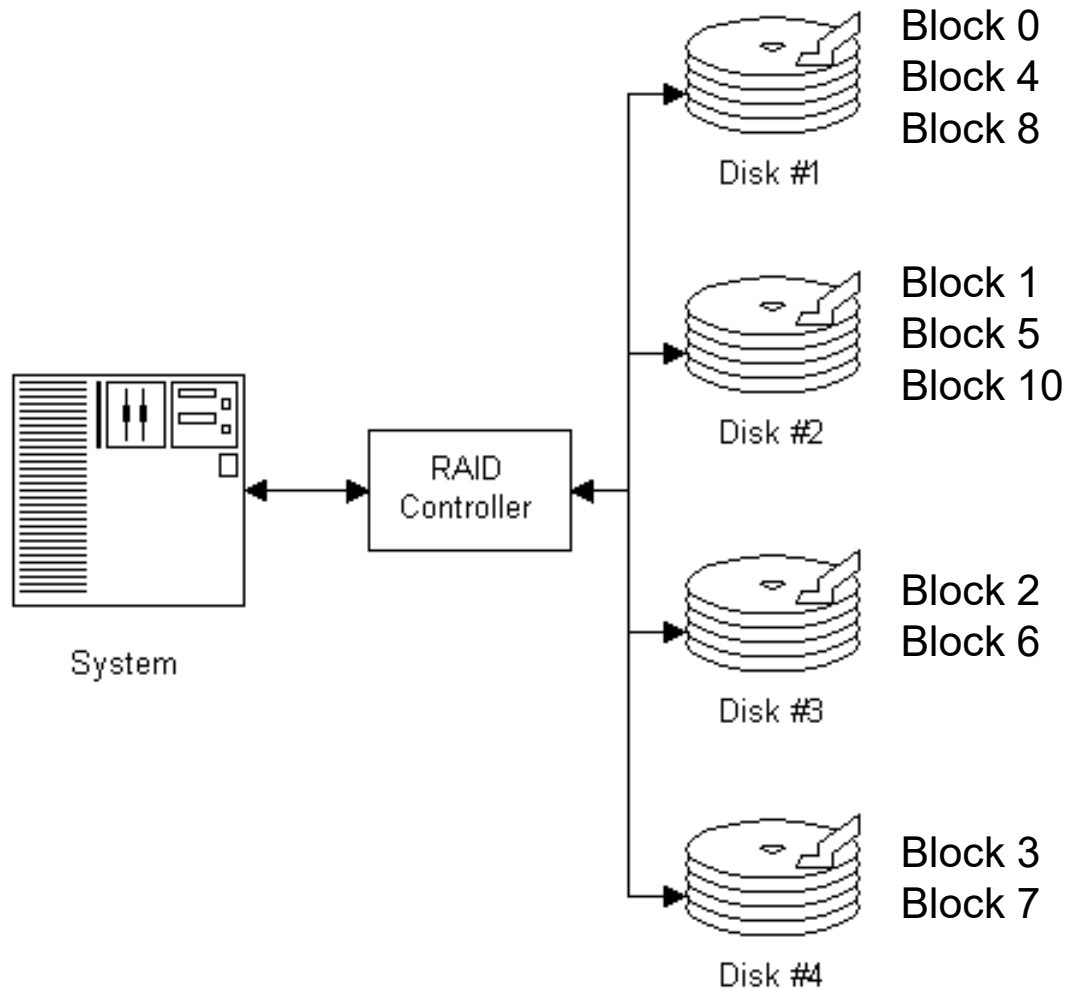
Disk Management (Cont.)



Disk Reliability

- Several improvements in disk-use techniques involve multiple disks working cooperatively
- *Disk striping* uses a group of disks as one storage unit
 - Each block is broken into several sub-blocks, with one sub-block stored on each disk
 - Time to transfer a block into memory is faster because all sub-blocks are transferred in parallel

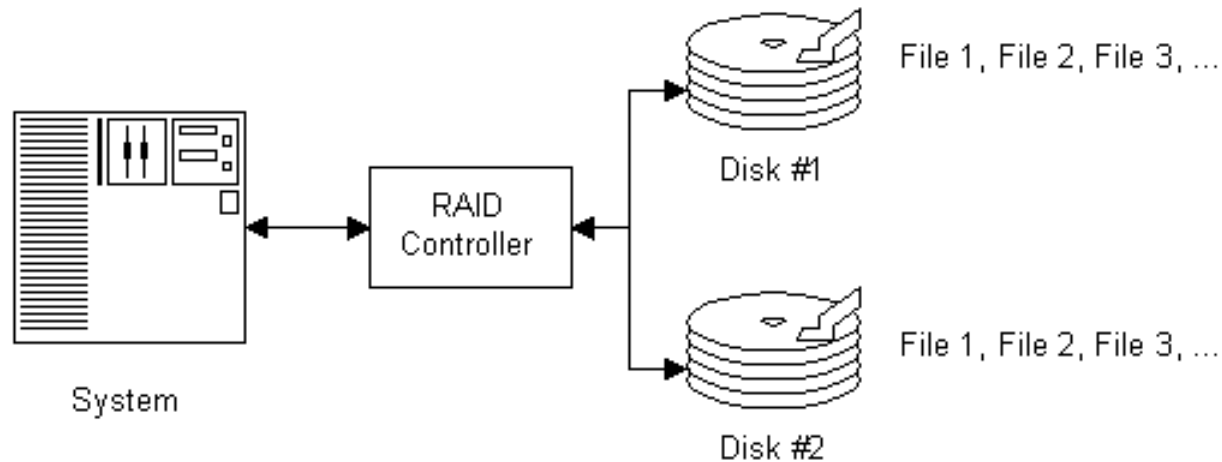
Disk Reliability (cont.)



with n drives, block i of a file stores on drive $(i \bmod n) + 1$

Disk Reliability (cont.)

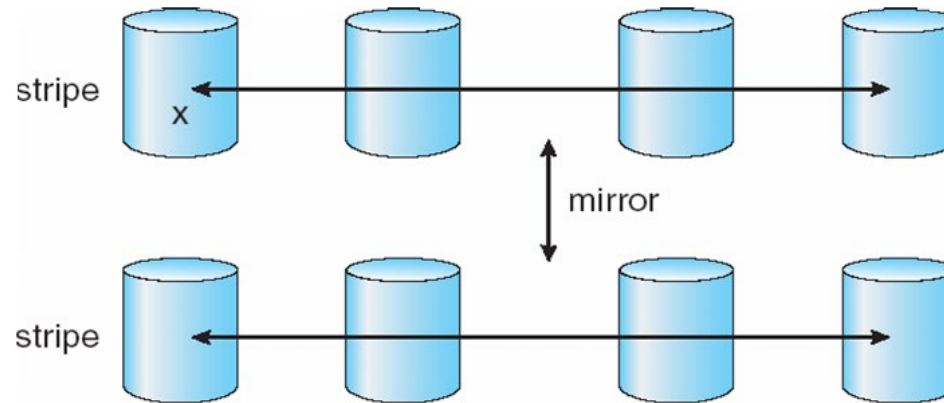
- *Mirroring* keeps duplicate of each disk
 - A logical disk consists of two physical disks
 - If one fails, the data can be read from the other



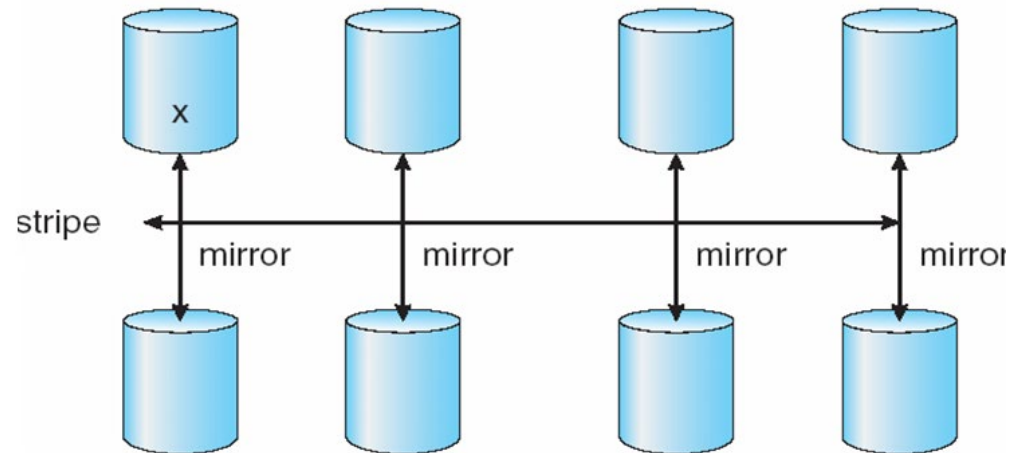
Disk Reliability (cont.)

- RAID (Redundant-Array-of-Independent-Disks) improves performance and reliability of the storage system by storing redundant data
 - **RAID 0**: non-redundant striping
 - Mirroring (**RAID 1**) keeps duplicate of each disk
 - Mirror of Strip (**RAID 0+1**) or Strip of Mirrors (**RAID 1+0**) provides high performance and high reliability

Disk Reliability (cont.)



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

Summary

- Device-driver layer hides the differences among device controllers from the I/O subsystem of the kernel.
- The I/O subsystem provides functions for: I/O scheduling, buffering, spooling, error handling, device reservation, and device name translation.
- I/O system calls are costly in terms of CPU consumption, because of the many layers of software between a physical device and the application.

Summary

- Disk scheduling algorithms optimize seek time:
 - FCFS, SSTF, SCAN, C-SCAN, C-LOOK
- Disk management requires disk partitioning and formatting
- Disk striping improve performance
- Mirroring improves reliability