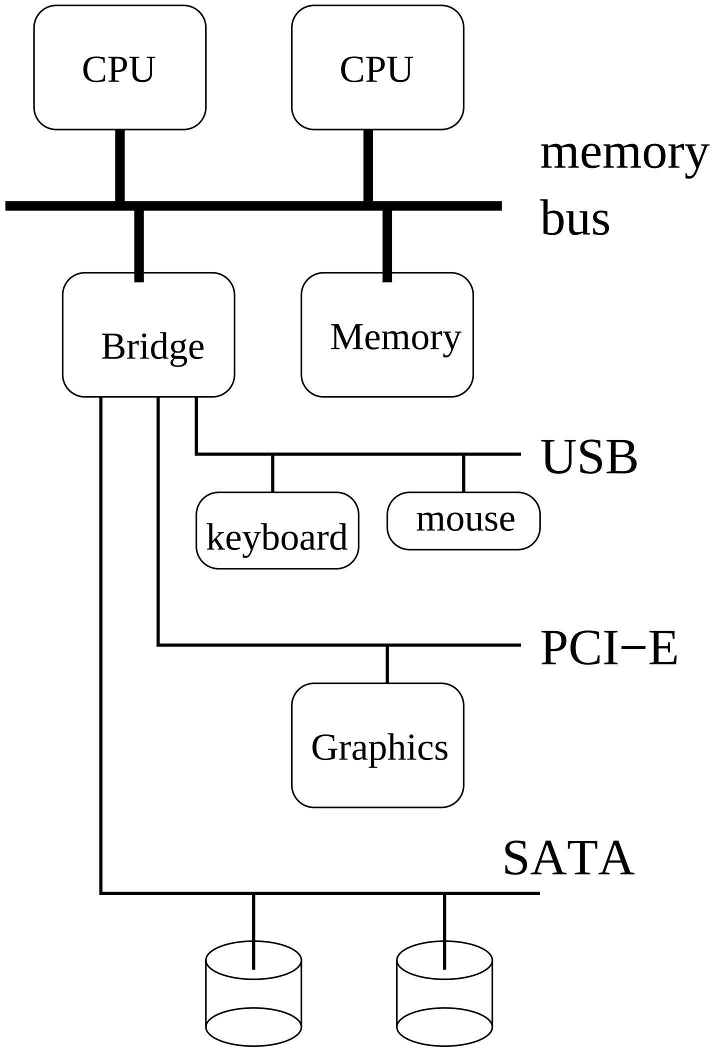
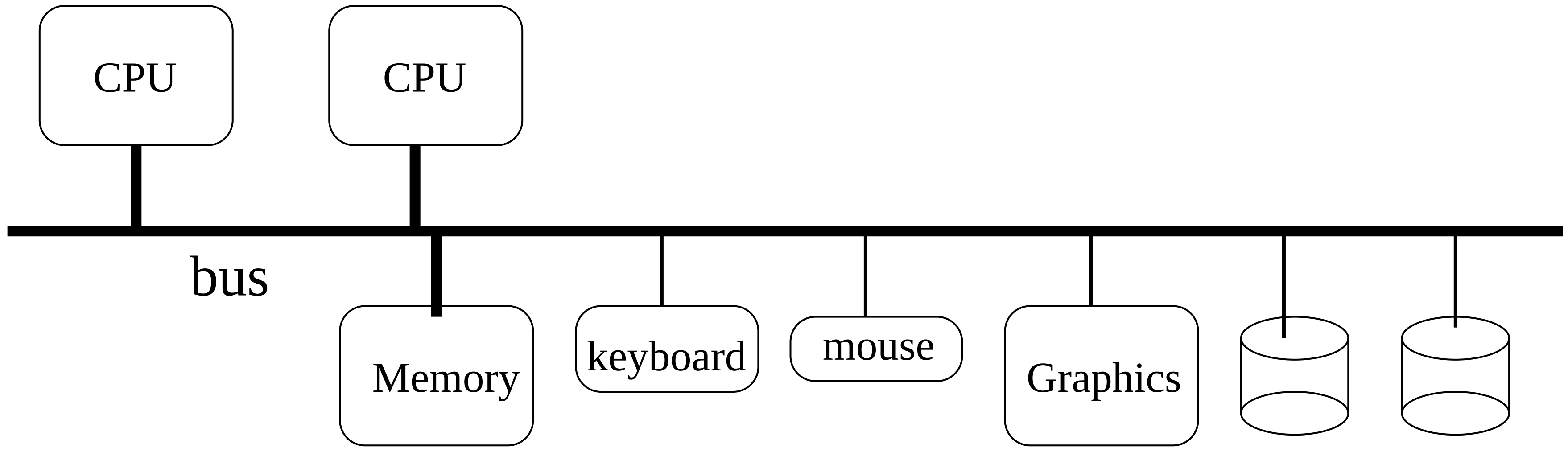
Bus Architecture Example



Simplified Bus Architecture



Sys/161 LAMEbus Device Examples

- timer/clock - current time, timer, beep
- disk drive - persistent storage
- serial console - character input/output
- text screen - character-oriented graphics
- network interface - packet input/output

Device Register Example: Sys/161 timer/clock

Offset	Size	Type	Description
0	4	status	current time (seconds)
4	4	status	current time (nanoseconds)
8	4	command	restart-on-expiry
12	4	status and command	interrupt (reading clears)
16	4	status and command	countdown time (microseconds)
20	4	command	speaker (causes beeps)

Device Register Example: Sys/161 disk controller

Offset	Size	Type	Description
0	4	status	number of sectors
4	4	status and command	status
8	4	command	sector number
12	4	status	rotational speed (RPM)
32768	512	data	transfer buffer

Device Drivers

- a device driver is a part of the kernel that interacts with a device
- example: write character to serial output device

```
write character to device data register
write output command to device command register
repeat {
    read device status register
} until device status is ‘‘completed’’
clear the device status register
```

- this example illustrates *polling*: the driver repeatedly checks whether the device is finished, until it is finished.

Another Polling Example

```
write target sector number into sector number register
write output data (512 bytes) into transfer buffer
write ‘‘write’’ command into status register
repeat {
   read status register
} until status is ‘‘completed’’ (or error)
clear the status register
```

Disk operations are slow. The device driver may have to poll for a long time.

Using Interrupts to Avoid Polling

- polling can be avoided if the device can use interrupts to indicate that it is finished
- example: disk write operation using interrupts:

```
write target sector number into sector number register
write output data (512 bytes) into transfer buffer
write ’’write’’ command into status register
block until device generates completion interrupt
read status register to check for errors
clear status register
```

- while thread running the driver is blocked, the CPU is free to run other threads
- kernel synchronization primitives (e.g., semaphores) can be used to implement blocking

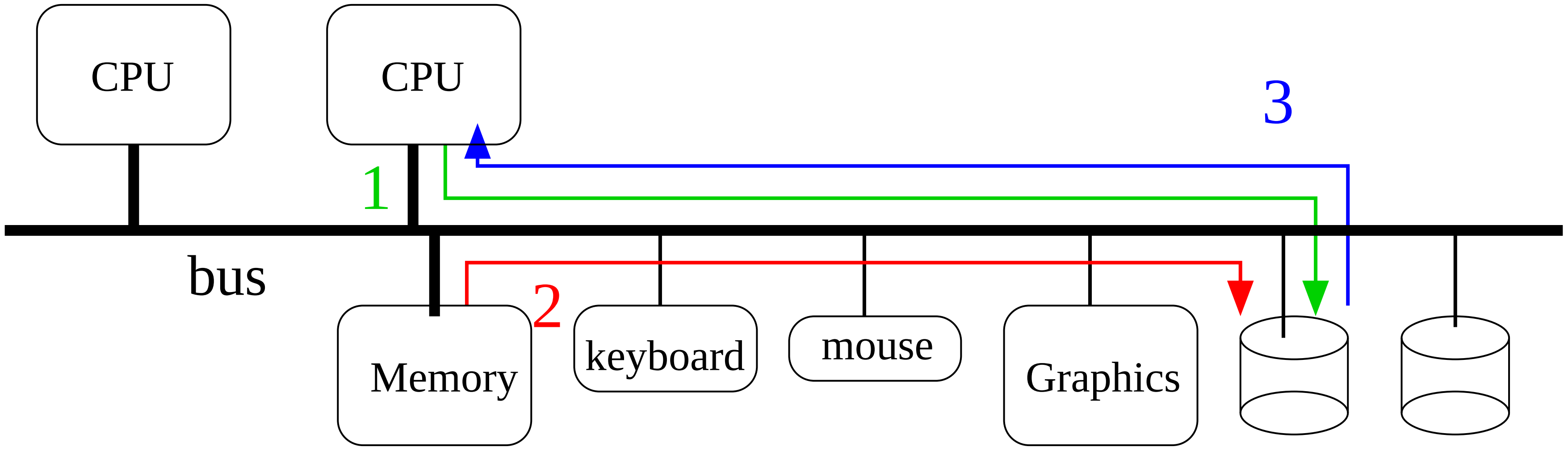
Device Data Transfer

- Sometimes, a device operation will involve a large chunk of data - much larger than can be moved with a single instruction.
  - example: disk read or write operation
- Devices may have data buffers for such data - but how to get the data between the device and memory?
  - Option 1: *program-controlled I/O*
    The device driver moves the data iteratively, one word at a time.
    - Simple, but the CPU is *busy* while the data is being transferred.
  - Option 2: *direct memory access (DMA)*
    - CPU is *not busy* during data transfer, and is free to do something else.

Sys/161 LAMEbus devices do program-controlled I/O.

Direct Memory Access (DMA)

- DMA is used for block data transfers between devices (e.g., a disk controller) and memory
- Under DMA, the CPU initiates the data transfer and is notified when the transfer is finished. However, the device (not the CPU) controls the transfer itself.



1. CPU issues DMA request to controller
2. controller directs data transfer
3. controller interrupts CPU

Device Driver for Disk Write with DMA

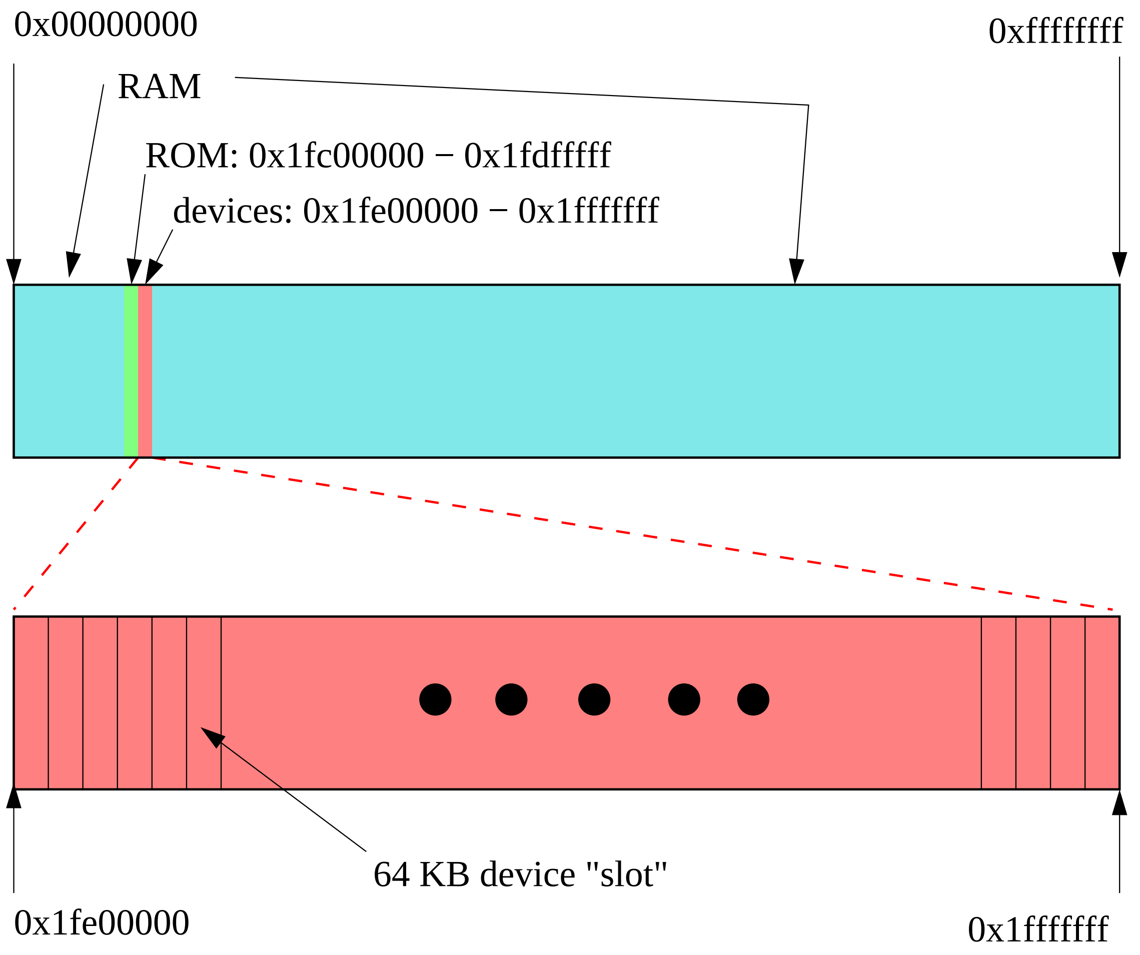
```
write target disk sector number into sector number register
write source memory address into address register
write ’’write’’ command into status register
block (sleep) until device generates completion interrupt
read status register to check for errors
clear status register
```

Note: driver no longer copies data into device transfer buffer

Accessing Devices

- how can a device driver access device registers?
- Option 1: special I/O instructions
  - such as `in` and `out` instructions on x86
  - device registers are assigned "port" numbers
  - instructions transfer data between a specified port and a CPU register
  -
- Option 2: memory-mapped I/O
  - each device register has a physical memory address
  - device drivers can read from or write to device registers using normal load and store instructions, as though accessing memory

MIPS/OS161 Physical Address Space

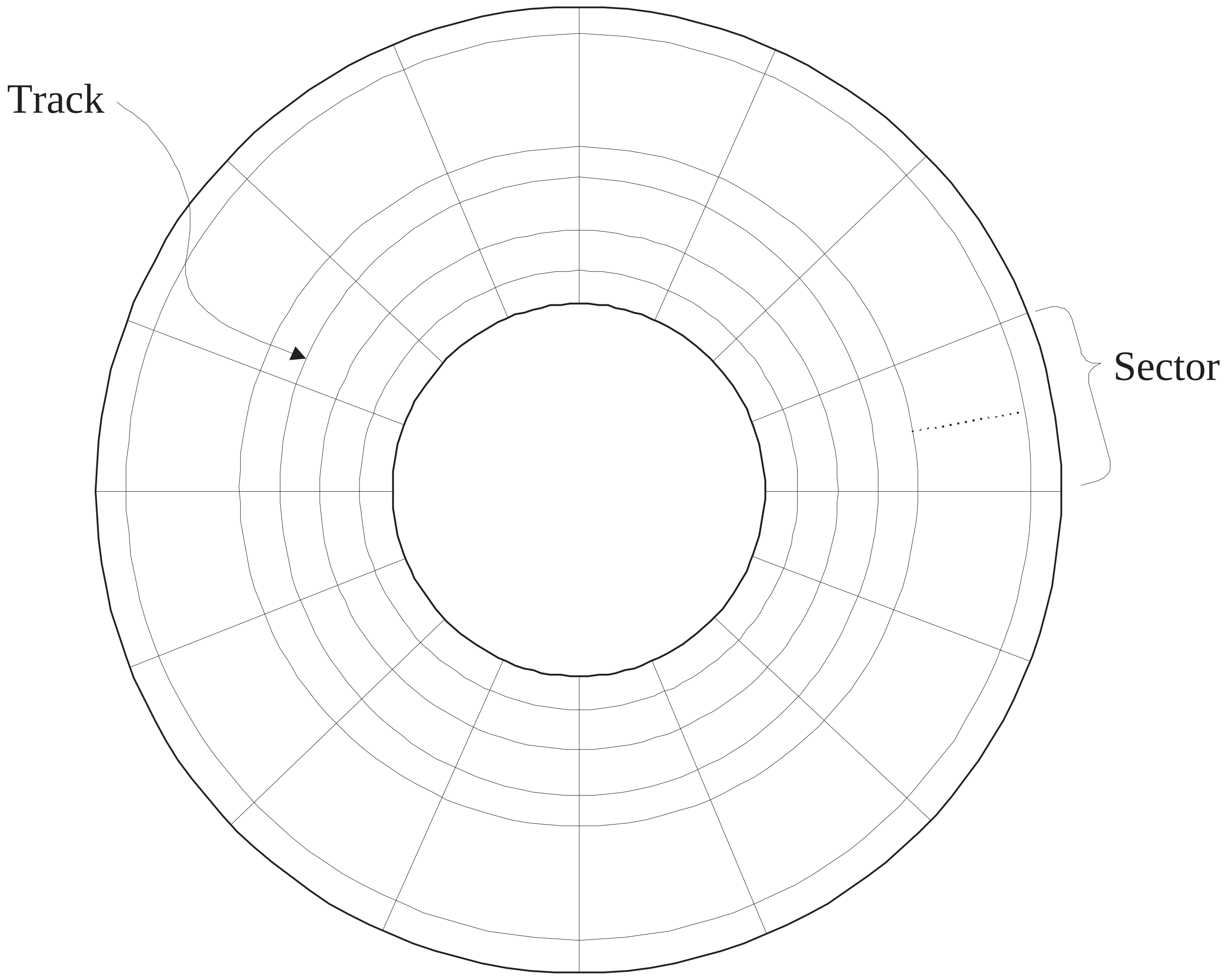


Each device is assigned to one of 32 64KB device “slots”. A device’s registers and data buffers are memory-mapped into its assigned slot.

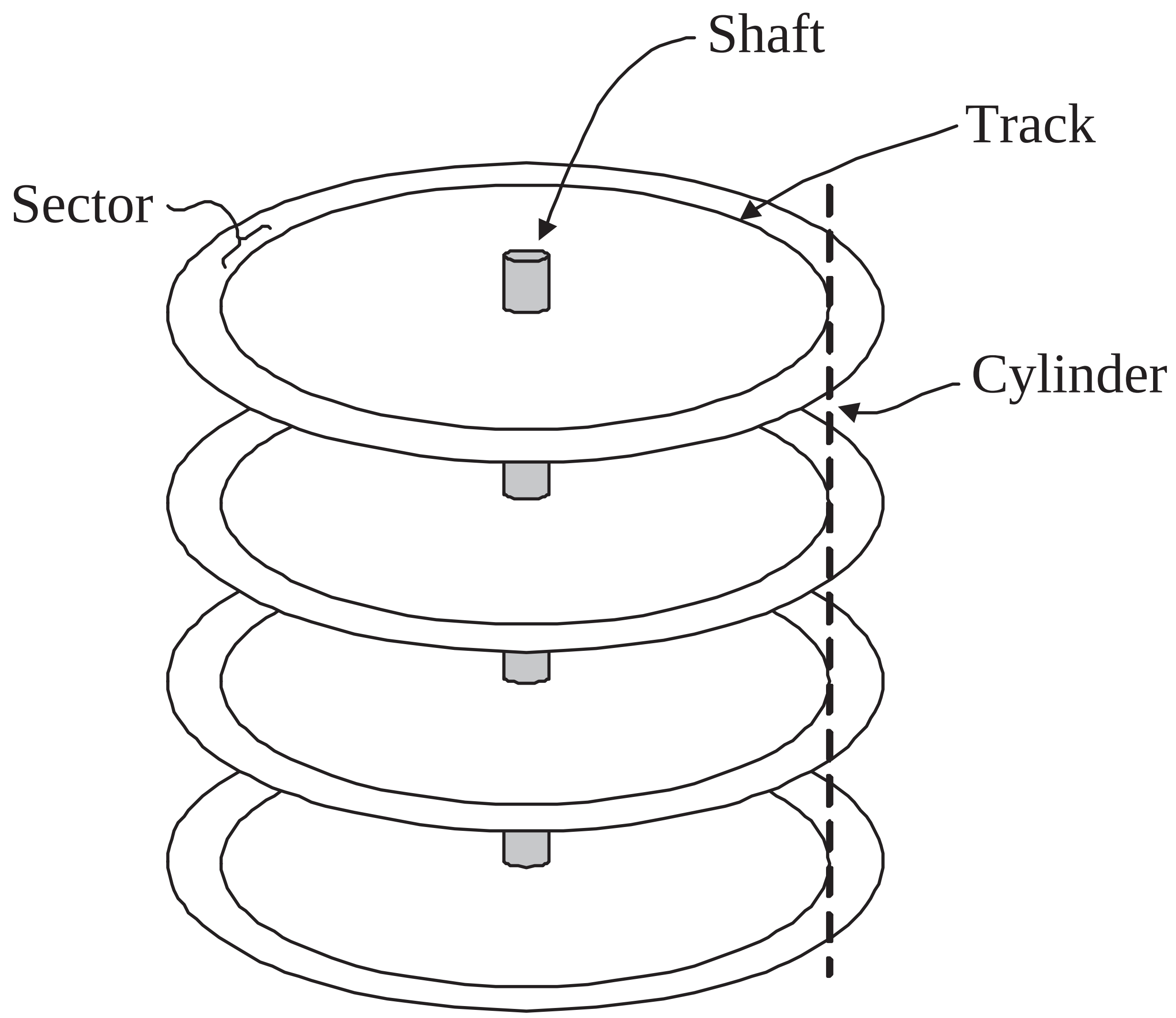
Logical View of a Disk Drive

- disk is an array of numbered blocks (or sectors)
- each block is the same size (e.g., 512 bytes)
- blocks are the unit of transfer between the disk and memory
  - typically, one or more contiguous blocks can be transferred in a single operation
- storage is *non-volatile*, i.e., data persists even when the device is without power

A Disk Platter's Surface



Physical Structure of a Disk Drive



Simplified Cost Model for Disk Block Transfer

- moving data to/from a disk involves:

  seek time: move the read/write heads to the appropriate cylinder

  rotational latency: wait until the desired sectors spin to the read/write heads

  transfer time: wait while the desired sectors spin past the read/write heads

- request service time is the sum of seek time, rotational latency, and transfer time

$$t_{service} = t_{seek} + t_{rot} + t_{transfer}$$

- note that there are other overheads but they are typically small relative to these three

Rotational Latency and Transfer Time

- rotational latency depends on the rotational speed of the disk
- if the disk spins at ω rotations per second:

$$0 \le t_{rot} \le \frac{1}{\omega}$$

- expected rotational latency:

$$\bar{t}_{rot} = \frac{1}{2\omega}$$

- transfer time depends on the rotational speed and on the amount of data transferred
- if k sectors are to be transferred and there are T sectors per track:

$$t_{transfer} = \frac{k}{T\omega}$$

Seek Time

- seek time depends on the speed of the arm on which the read/write heads are mounted.
- a simple linear seek time model:
  - $t_{maxseek}$ is the time required to move the read/write heads from the innermost cylinder to the outermost cylinder
  - C is the total number of cylinders
- if k is the required *seek distance* ($k > 0$):

$$t_{seek}(k) = \frac{k}{C} t_{maxseek}$$

Performance Implications of Disk Characteristics

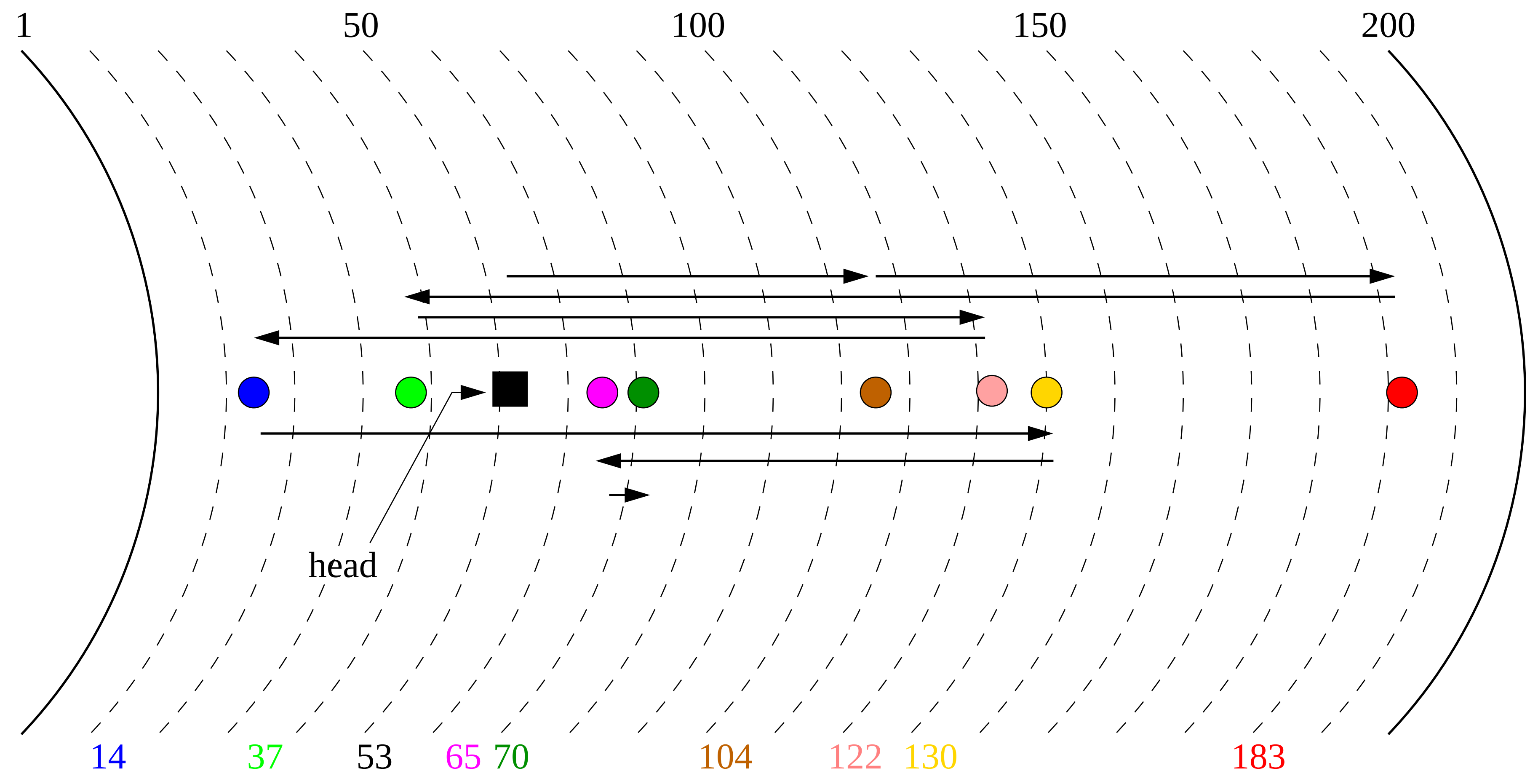
- larger transfers to/from a disk device are *more efficient* than smaller ones. That is, the cost (time) per byte is smaller for larger transfers. (Why?)
- sequential I/O is faster than non-sequential I/O
  - sequential I/O operations eliminate the need for (most) seeks
  - disks use other techniques, like *track buffering*, to reduce the cost of sequential I/O even more

Disk Head Scheduling

- goal: reduce seek times by controlling the order in which requests are serviced
- disk head scheduling may be performed by the controller, by the operating system, or both
- for disk head scheduling to be effective, there must be a queue of outstanding disk requests (otherwise there is nothing to reorder)
- an on-line approach is required: the disk request queue is not static

FCFS Disk Head Scheduling

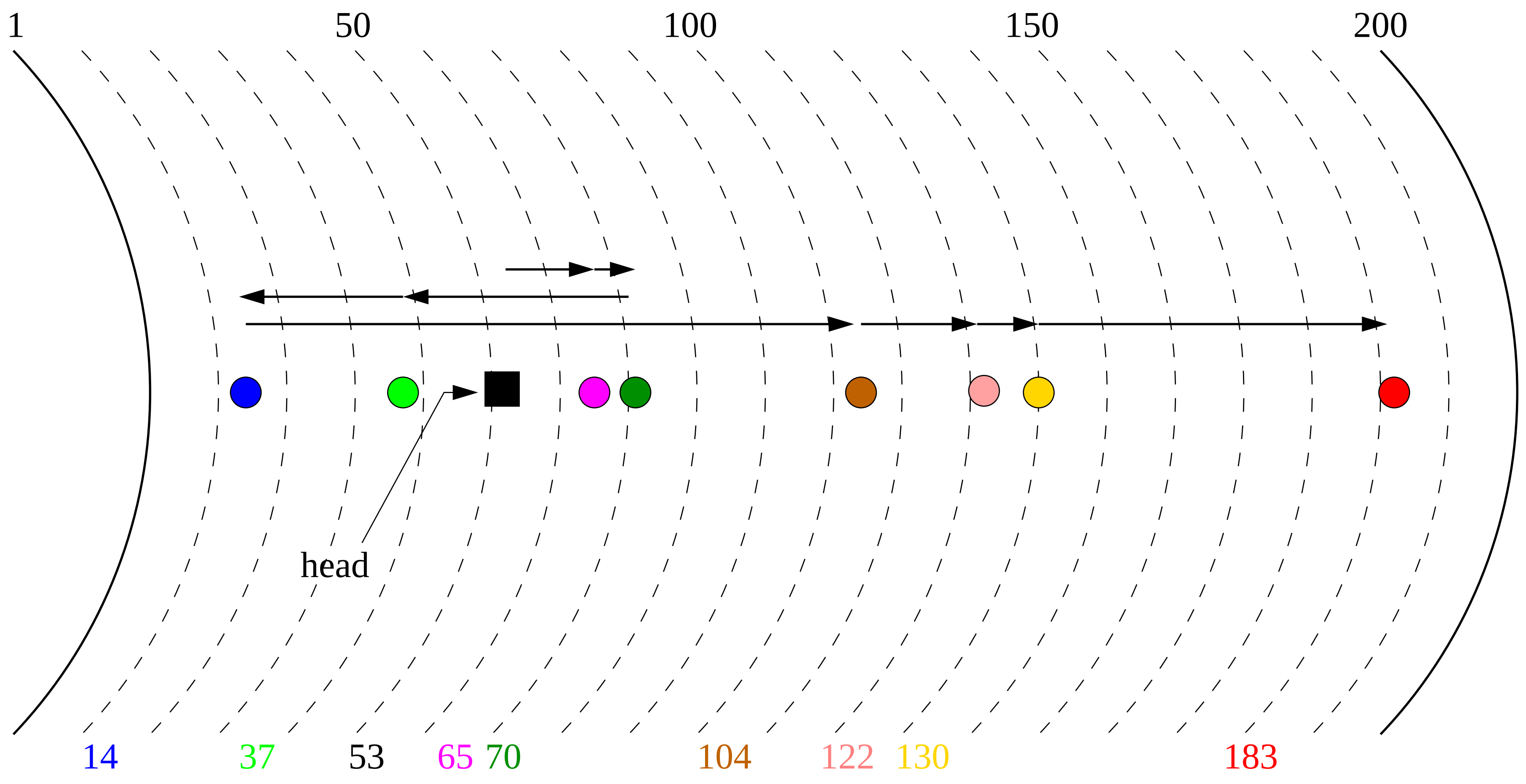
- handle requests in the order in which they arrive
- fair and simple, but no optimization of seek times



arrival order: 104 183 37 122 14 130 65 70

Shortest Seek Time First (SSTF)

- choose closest request (a greedy approach)
- seek times are reduced, but requests may starve

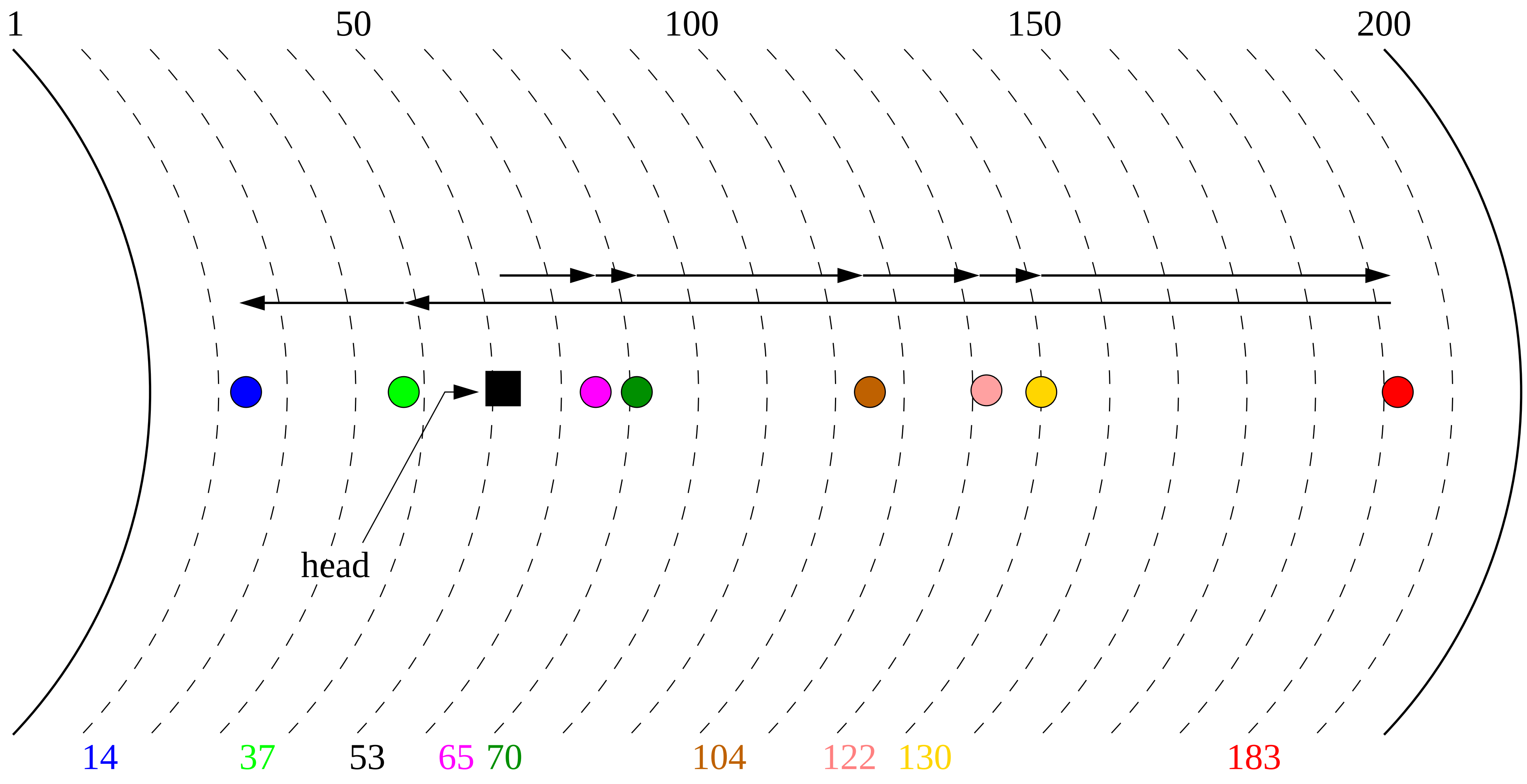


arrival order: 104 183 37 122 14 130 65 70

Elevator Algorithms (SCAN)

- Under SCAN, aka the elevator algorithm, the disk head moves in one direction until there are no more requests in front of it, then reverses direction.
- there are many variations on this idea
- SCAN reduces seek times (relative to FCFS), while avoiding starvation

SCAN Example



arrival order: 104 183 37 122 14 130 65 70