Chapter 10: Input/Output and Stable Storage

10.1 Communication Between CPU and I/O Devices

A special piece of hardware known as a *device controller* acts as an intermediary between an I/O device and the computer

- Controller knows how to communicate with both the I/O device and with the computer

10.1.1 Example: Keyboard

A keyboard register needs to have a *data* register and a *status* register

- Data stores the character typed on the keyboard
- Status stores aggregation of the current state of information exchange
    - Includes ready bit (is character in the data register new?)
    - Interrupt Enable bit (Is the processor allowing the device to interrupt it?)
    - Interrupt Flag bit (Is the controller ready to interrupt the processor?)

10.1.2 Memory Mapped I/O

Registers in the controller appear as memory locations to the CPU

- Use load/store instructions

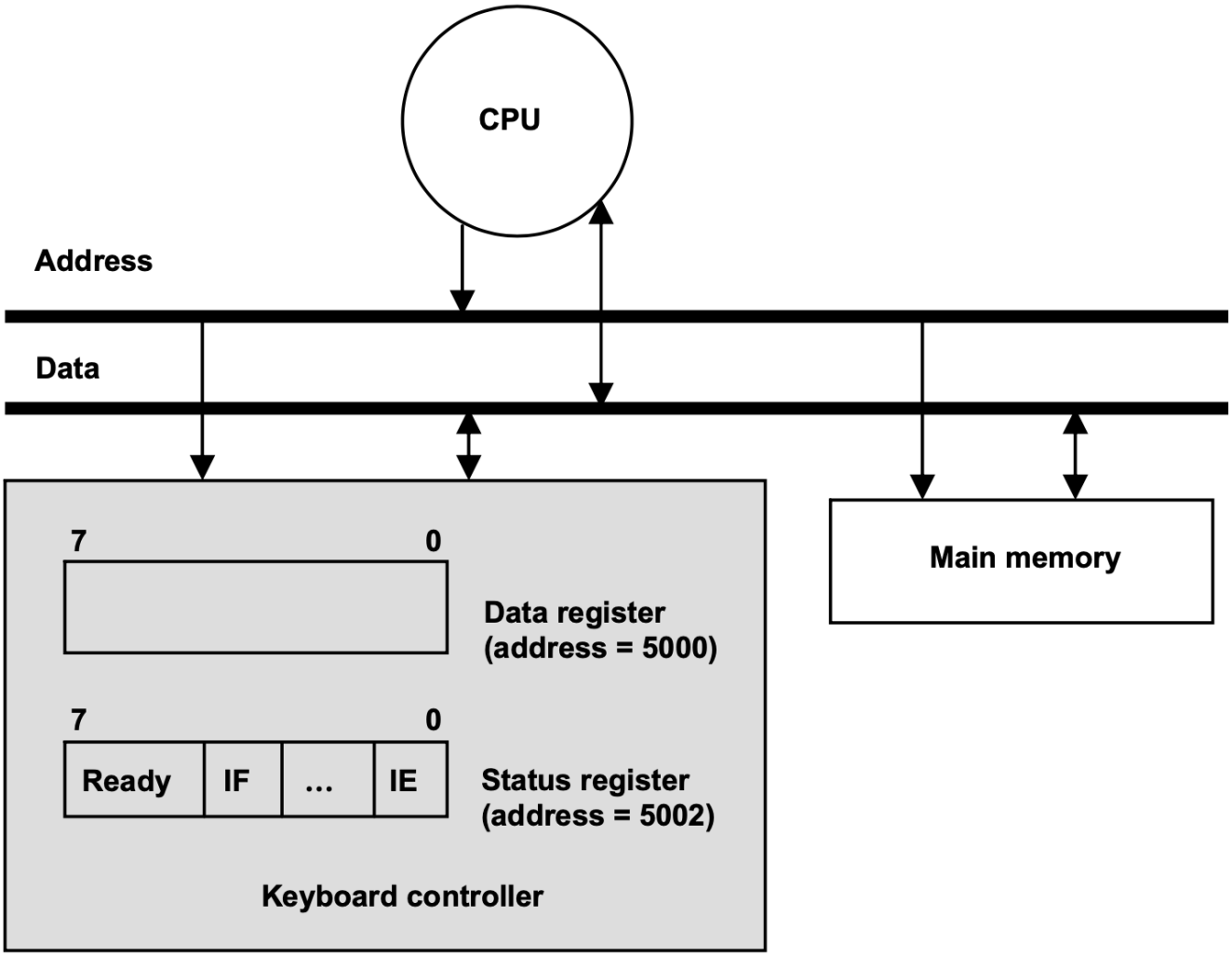


Figure 10.1: Connecting the device controller to the processor-memory bus

Why doesn't this cause confusion between normal memory and device controllers?

- Reserve a portion of the address space for device controllers
  - Ex: `0xFFFF0000` through `0xFFFFFFFF` for I/O devices
- Every device controller has circuitry to decode an address appearing on the bus. If an address appearing on the bus corresponds to one assigned to this controller, then it behaves like “memory” for the associated command on the bus

How does this work with cache?

- We mark certain areas of memory as uncacheable

Advantages

- no special I/O instructions are required

Disadvantages

- portion of the memory address space is lost to device registers

10.2 Programmed I/O

Programmed I/O (PIO) refers to writing a computer program that accomplishes such data transfer

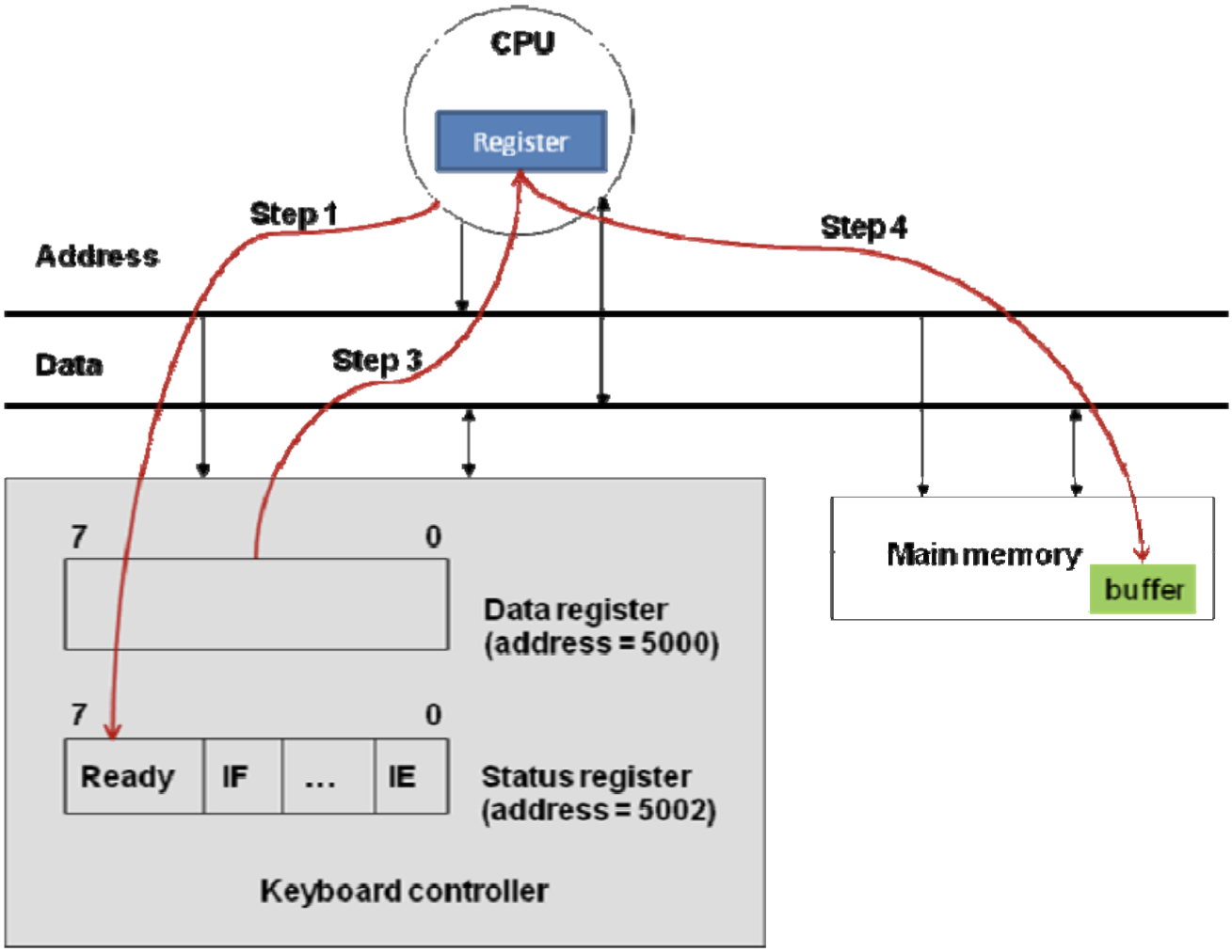


Figure 10.2: An example of PIO Data Transfer

Here is a simple program for the keyboard controller

1. Check the ready bit (step 1 in Figure 10.2).
2. If not set go to step1.
3. Read the contents of the data register (step 3 in Figure 10.2).
4. Store the character read into memory (step 4 in Figure 10.2).
5. Go to step 1.

Step 1 and 2 constitute the handshake between the processor and the device controller.

Programmed data transfer works for slow speed devices (for example, keyboard and mouse) that typically produce data *asynchronously* – data production is not rhythmic with any clock pulse

High-speed devices such as disks produce data synchronously – data follows a rhythmic clock pulse

10.3 DMA

Streaming devices – once data transfer starts in either direction ("from" or "to" the device), data moves in or out the device continuously until completion of the transfer

We need a *hardware buffer* the size of *unit of synchronous transfer* between the device and device controller

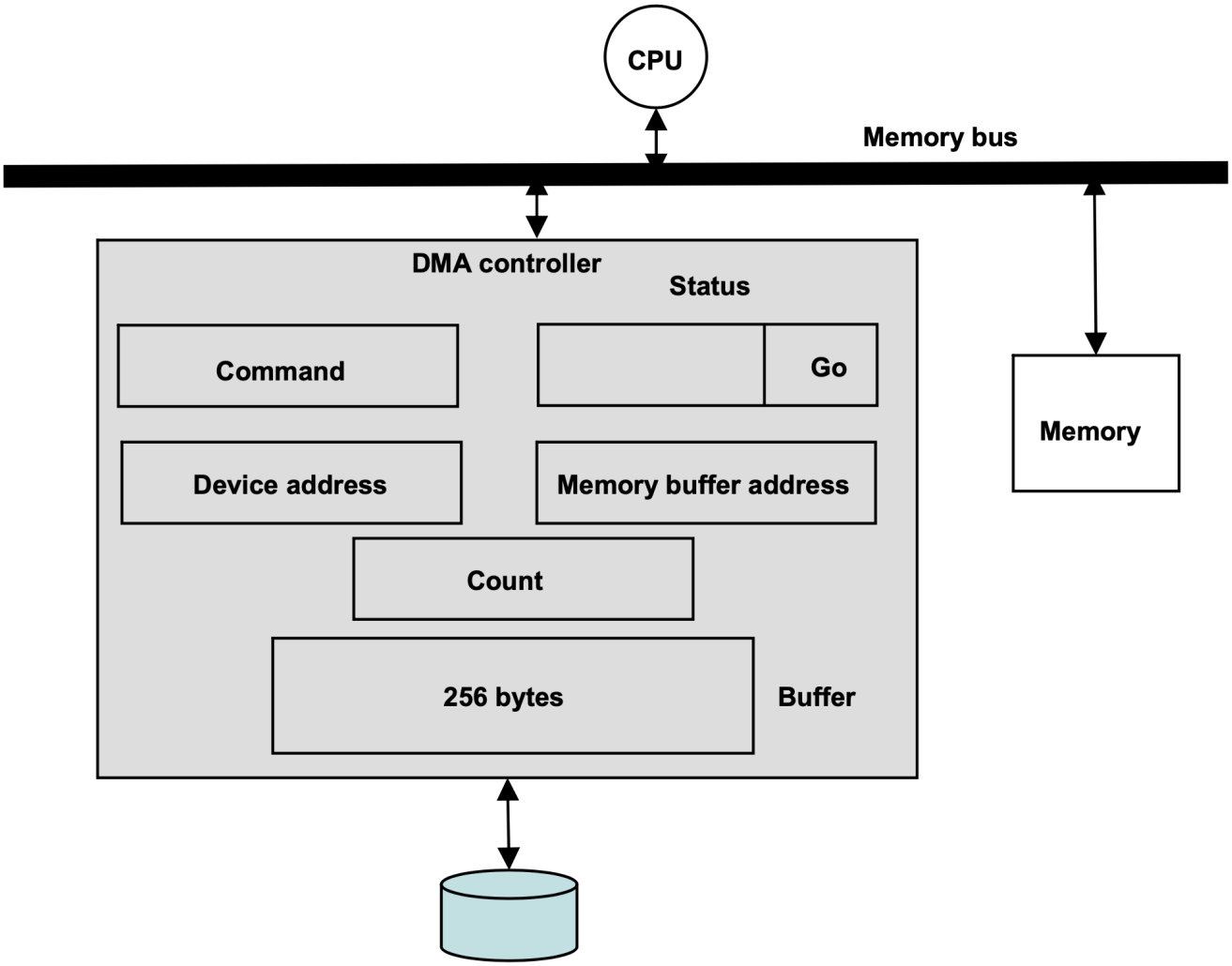


Figure 10.3: DMA controller

For example, to transfer `N` bytes of data from a memory buffer at address `M` to the device at address `D`, the CPU executes the following instructions:

1. Store `N` into the Count register
2. Store `M` into the Memory buffer address register
3. Store `D` into the Device address register
4. Store write to the device command into the Command register
5. Set the Go bit in the Status register

Note that all of the above are simple memory store instructions (so far as the CPU is concerned) since these registers are memory mapped

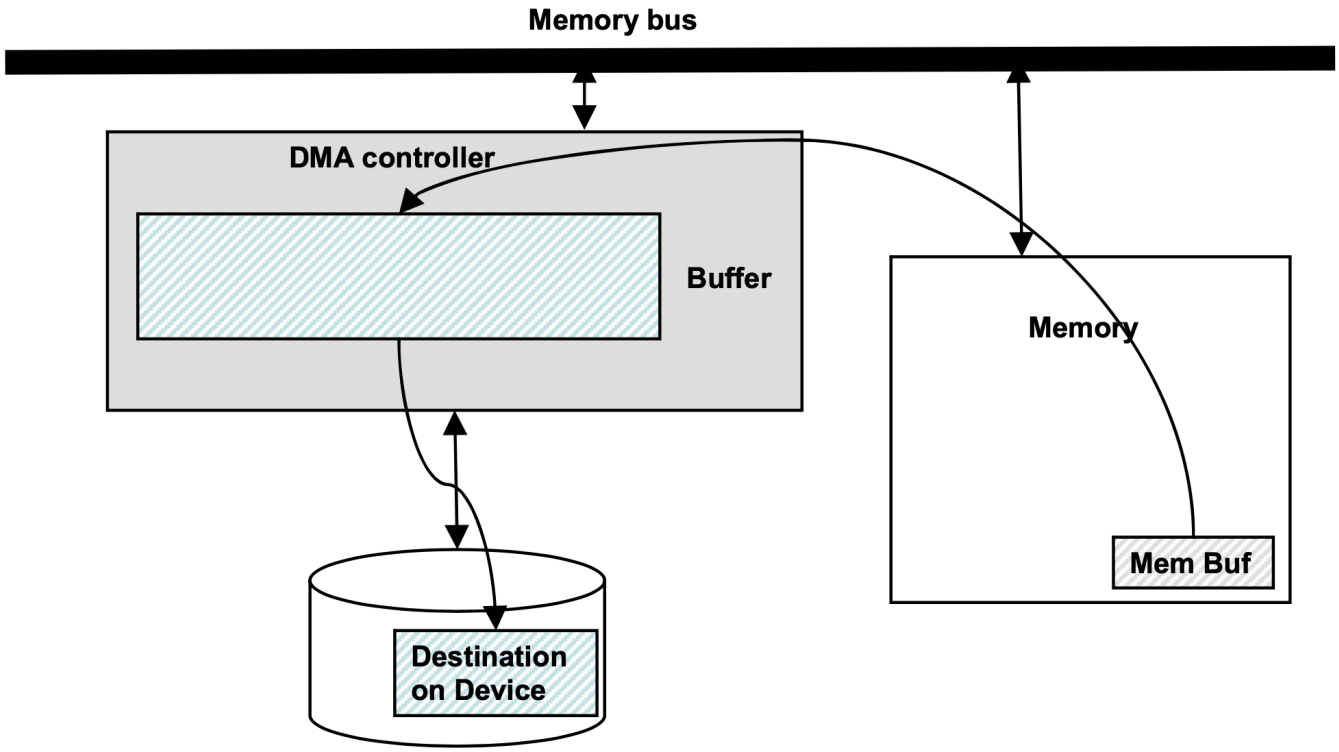


Figure 10.4: An Example of DMA Data Transfer

The device controller competes with the processor for memory bus cycles, referred to as *cycle stealing*

- Devices steal bus cycles when the processor does not need them

10.4 Buses

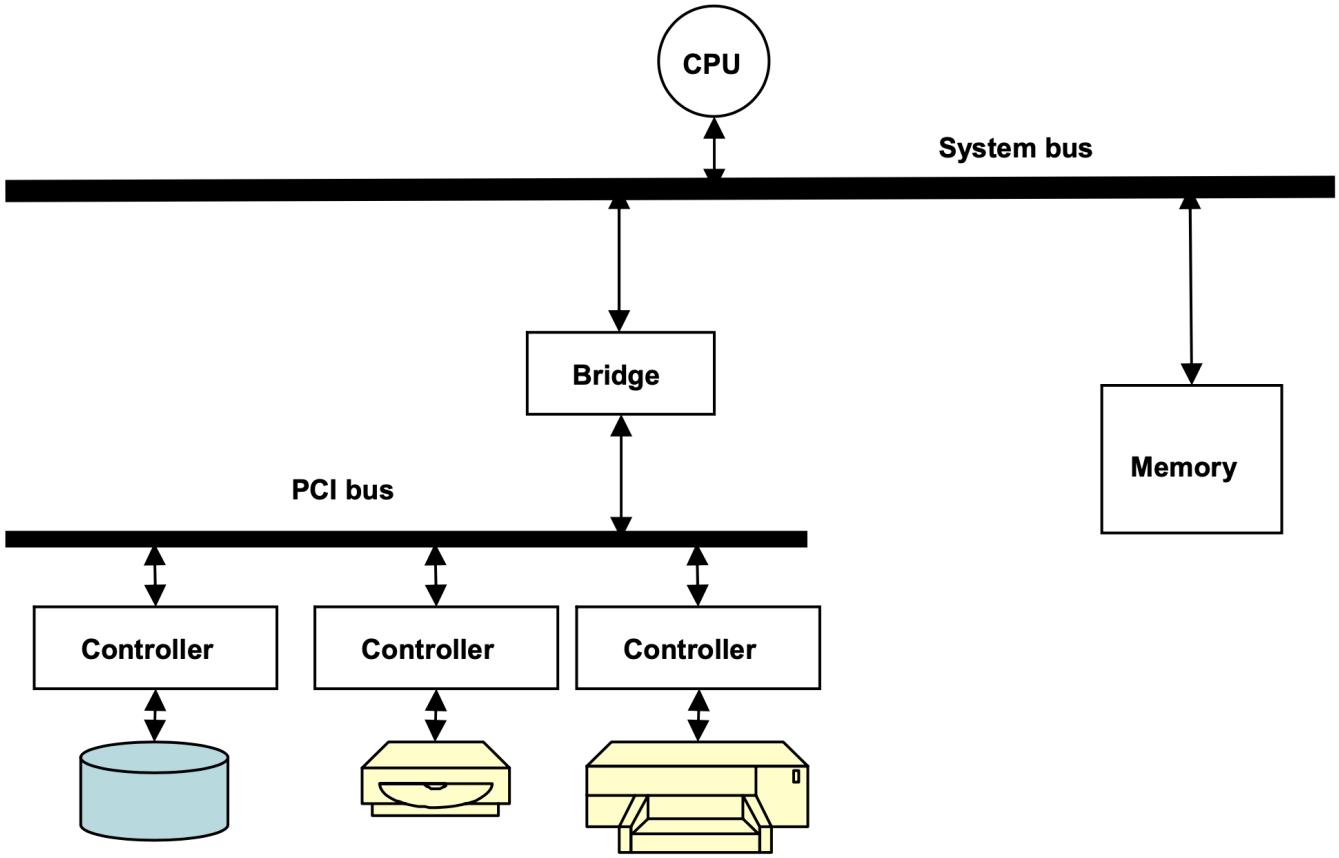


Figure 10.5: Coexistence of Standard Buses and System Buses

10.5 I/O Processor

I/O processors decouple the I/O chores from the main processor

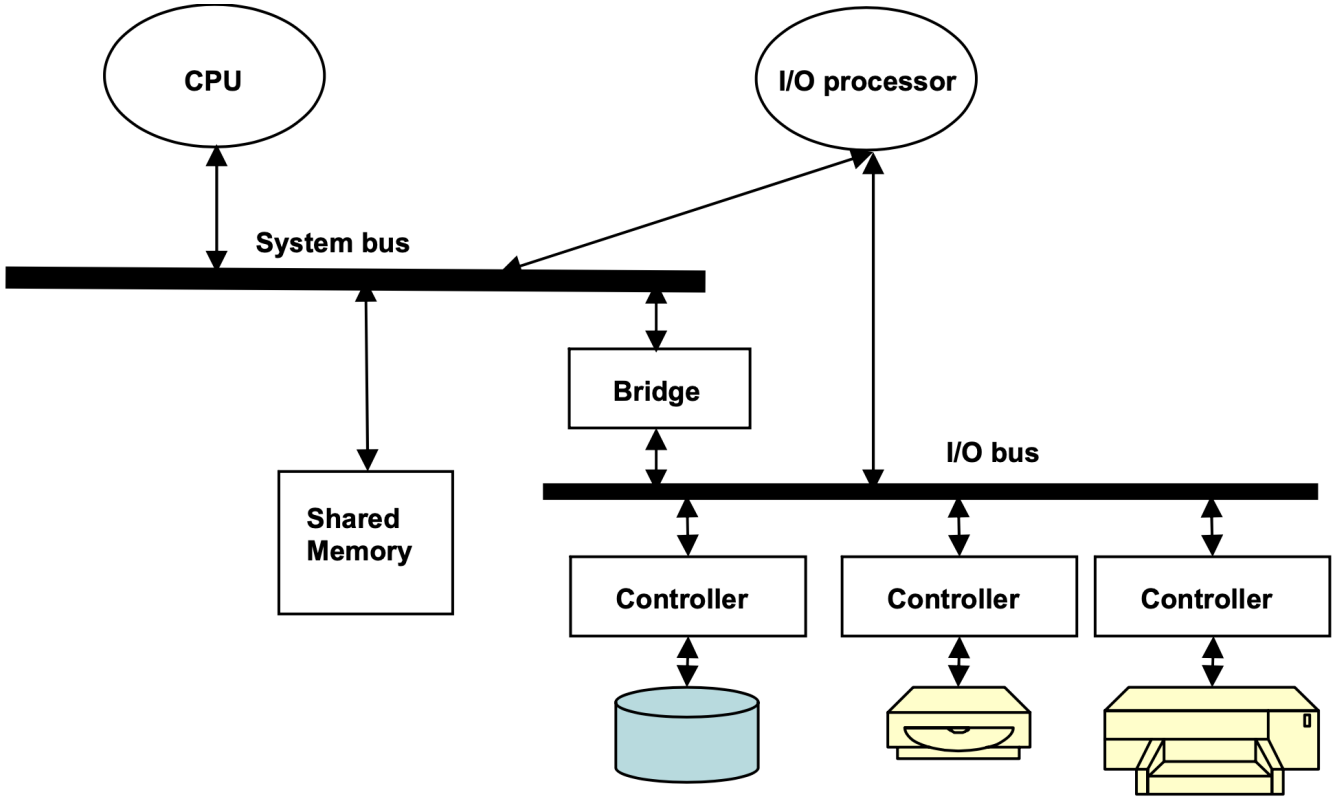


Figure 10.6: I/O Processor

10.6 Device Driver

There is some abstraction going on here – we don't really care how the device driver handles the device (e.g. interrupt driven vs DMA)

10.6.1 Example (Pan Tilt Camera)

Some of the situations may be totally unplanned for and could occur simply because of "human in the loop."

1. The device is unplugged from the computer while data transfer is going on.
2. The power chord for the device is unplugged while the data transfer is going on.
3. The device starts malfunctioning during data transfer (e.g., someone accidentally knocked the camera off its moorings; etc.)

10.7 Peripheral Devices

Character Oriented

- I/O happens one character at a time
- Devices are relatively slow
- Uses programmed I/O
- Example: CRT, teletypewriters

Block Oriented

- I/O happens with a group of data every time
- Example: Laser Printer
- Need DMA

10.8 Disk Storage

A disk consists of *platters* (ferromagnetic surfaces) that store data. A *central spindle* groups these together and spins them at a high speed. There is an array of magnetic *read/write heads*, one per surface (though the heads do not touch the surface!!). An *arm* connects each head to a common shaft which forms the *head assembly*. All the heads simultaneously line up on their respective surfaces at the same radial position.

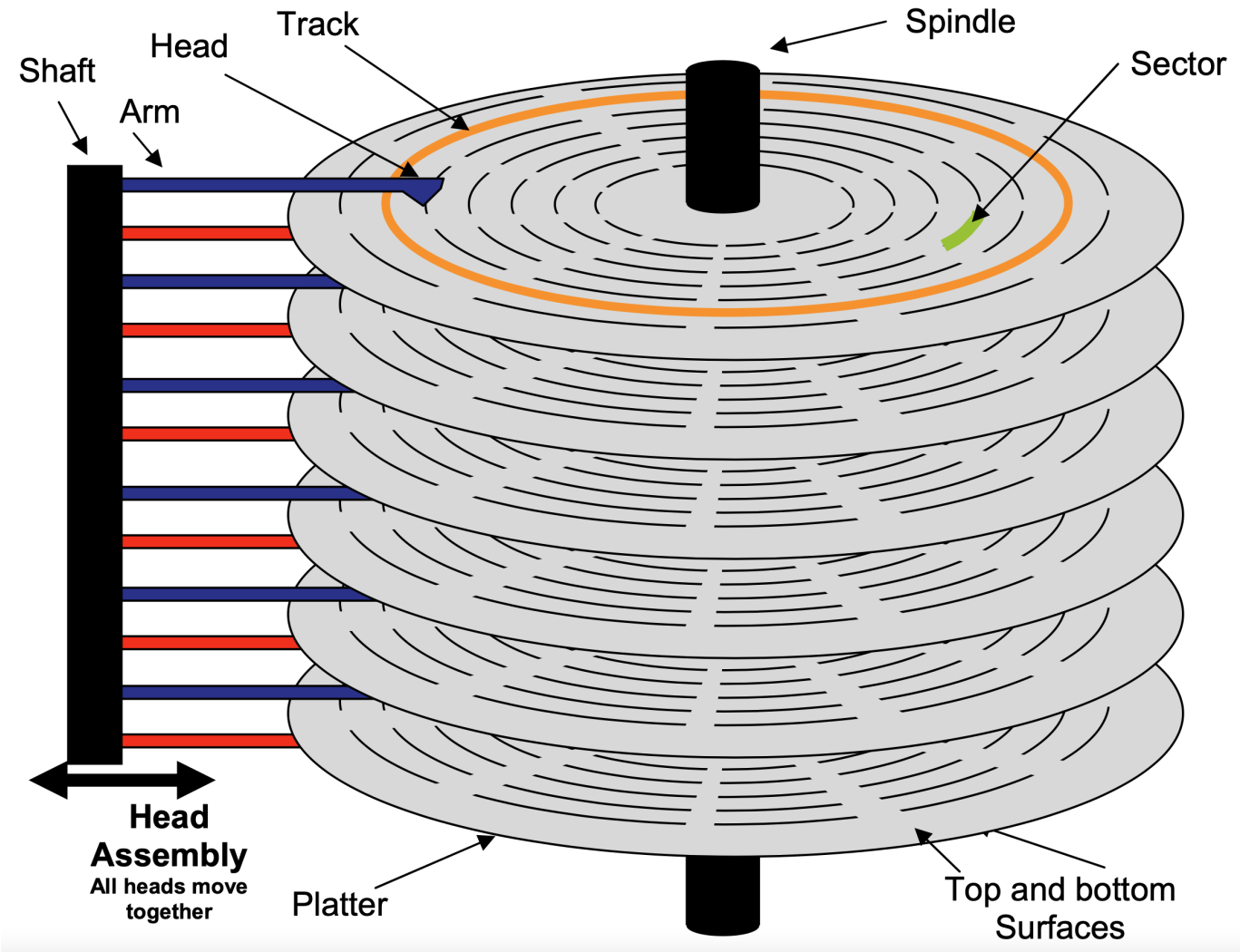


Figure 10.7: Magnetic Disk

A *track* is a predefined radial distance from the center of the disk that represents the recording surface

- Based on granularity of actuator
- Based on recording density

The set of corresponding tracks on all the surfaces form a *logical cylinder*

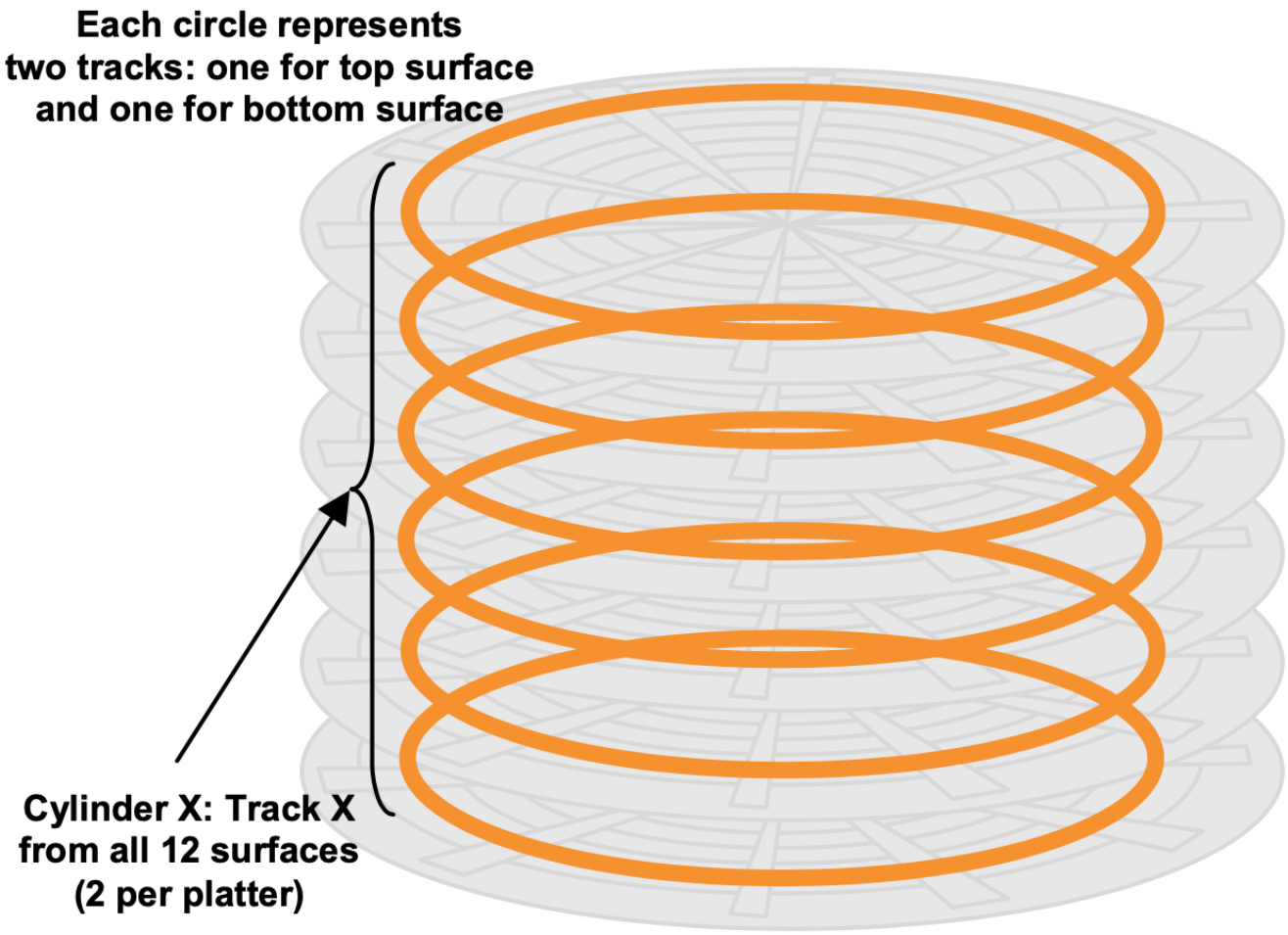
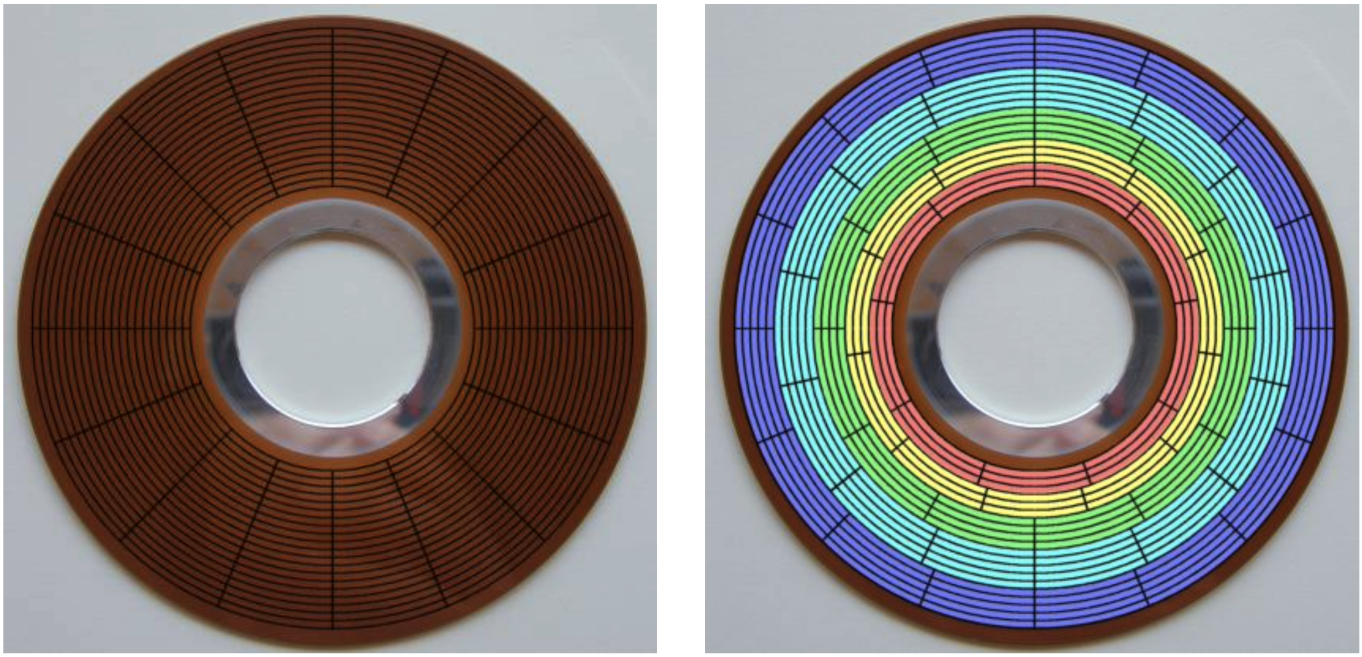


Figure 10.8: Logical Cylinder

Note how outer tracks with have a larger circumference and therefore have a larger footprint



(a) Normal (Non zoned) recording **(b) Zoned recording**

Figure 10.9: Difference between non-zoned and zoned recording

Modern disk drives use ***zoned bit recording (ZBR)***

Let

p – number of platters

n – number of surfaces per platter (1 or 2), t – number of tracks per surface

s – number of sectors per track

b – number of bytes per sector

The total capacity of the disk assuming non-zoned recording:

$\text{Capacity} = p * n * t * s * b \text{ bytes}$

With zoned recording,

z – number of zones

t_{zi} – number of tracks at zone z_i

s_{zi} – number of sectors per track at zone z_i

$\text{Capacity} = p * n * \sum_{1 \leq i \leq z} t_{zi} * s_{zi} \text{ bytes}$

Example

Given the following specifications for a disk drive:

- 256 bytes per sector
- 12 sectors per track
- 20 tracks per surface
- 3 platters

What is the total capacity?

$$\text{total capacity} = \text{number of platters} \cdot \text{surfaces/platter} \cdot \text{tracks/surface} \cdot \text{sectors/track} \cdot \text{bytes/sector}$$

$$\text{total capacity} = 3 \cdot 2 \cdot 20 \cdot 12 \cdot 256 \text{ bytes} = 360 \textbf{ KBytes}$$

Assume a zoned bit recording with the following specifications:

- 3 zones
  - Zone 3 (outermost): 8 tracks, 18 sectors per track
  - Zone 2: 7 tracks, 14 sectors per track
  - Zone 1: 5 tracks, 12 sectors per track

What is the total capacity with zoned-bit recording?

$$\text{Capacity of Zone 3} = \text{num of platters} \cdot \text{surfaces/platter} \cdot \text{tracks in zone 3} \cdot \text{sectors/track} * \text{bytes/sector} =$$

$$\textbf{Capacity of Zone 3} = 216 \textbf{ Kbytes}$$

$$\text{Capacity of Zone 2} = \text{num of platters} \cdot \text{surfaces/platter} \cdot \text{tracks in zone 2} \cdot \text{sectors/track} * \text{bytes/sector} =$$

$$\textbf{Capacity of Zone 3} = 147 \textbf{ Kbytes}$$

$$\text{Capacity of Zone 1} = \text{num of platters} \cdot \text{surfaces/platter} \cdot \text{tracks in zone 1} \cdot \text{sectors/track} * \text{bytes/sector} =$$

$$\textbf{Capacity of Zone 3} = 90 \textbf{ Kbytes}$$

$$\textbf{Total capacity} = 216 + 147 + 90 = 453 \textbf{ KBytes}$$

An important side effect of ZBR is the difference in transfer rates between outer and inner tracks

- Angular momentum remains the same but outer track has more sectors
- Head reads more sectors per revolution of the disk when it is over an outer track compared to an inner track
- You want to allocate outer tracks first before allocating inner tracks

Address is represented as `(cylinder #, surface #, sector #)`

Ranked Expensiveness Top to Bottom!!

Seek time – time to move to the specific cylinder

Rotational latency – time to spin to bring the required sector under the head

Data transfer time – time it takes for data from the selected surface is read and transferred to the controller as the sector moves under the head

Let

r – rotational speed of the disk in Revolutions Per Minute (RPM)

s – number of sectors per track

b – number of bytes per sector

$$\textbf{Data transfer rate} = \frac{\text{Amount of data in track}}{\text{time for one revolution}} = \frac{s * b * r}{60} \text{ bytes/second}$$

Let

a – average seek time in seconds

r – rotational speed of the disk in Revolutions Per Minute (RPM)

s – number of sectors per track

$$\textbf{Rotational latency} = \frac{60}{r} \text{ seconds}$$

$$\textbf{Average rotational latency} = \frac{60}{r * 2} \text{ seconds}$$

$$\textbf{Sector read time} = \frac{60}{r * s} \text{ seconds}$$

$$\text{Time to get to the desired track} = a$$

$$\text{Time to get the head over the desired sector} = \frac{\frac{60}{r}}{2}$$

$$\text{Time to read a sector} = \frac{60}{r \cdot s}$$

$$\textbf{Time to read a random sector on the disk} = a + \frac{\frac{60}{r}}{2} + \frac{60}{r \cdot s} \text{ seconds}$$

10.9 Disk Scheduling Algorithms

request_q

Figure 10.10: Disk request queue in the order of arrival

Remember the metrics we had in Chapter 6? They are back now

10.9.1 First-Come First Served (FCFS)

Incurs the least variance in waiting time regardless of the track from which it requests I/O

- Poor throughput
- Disk head may have to swing back and forth across the disk surface to satisfy the requests in an FCFS schedule

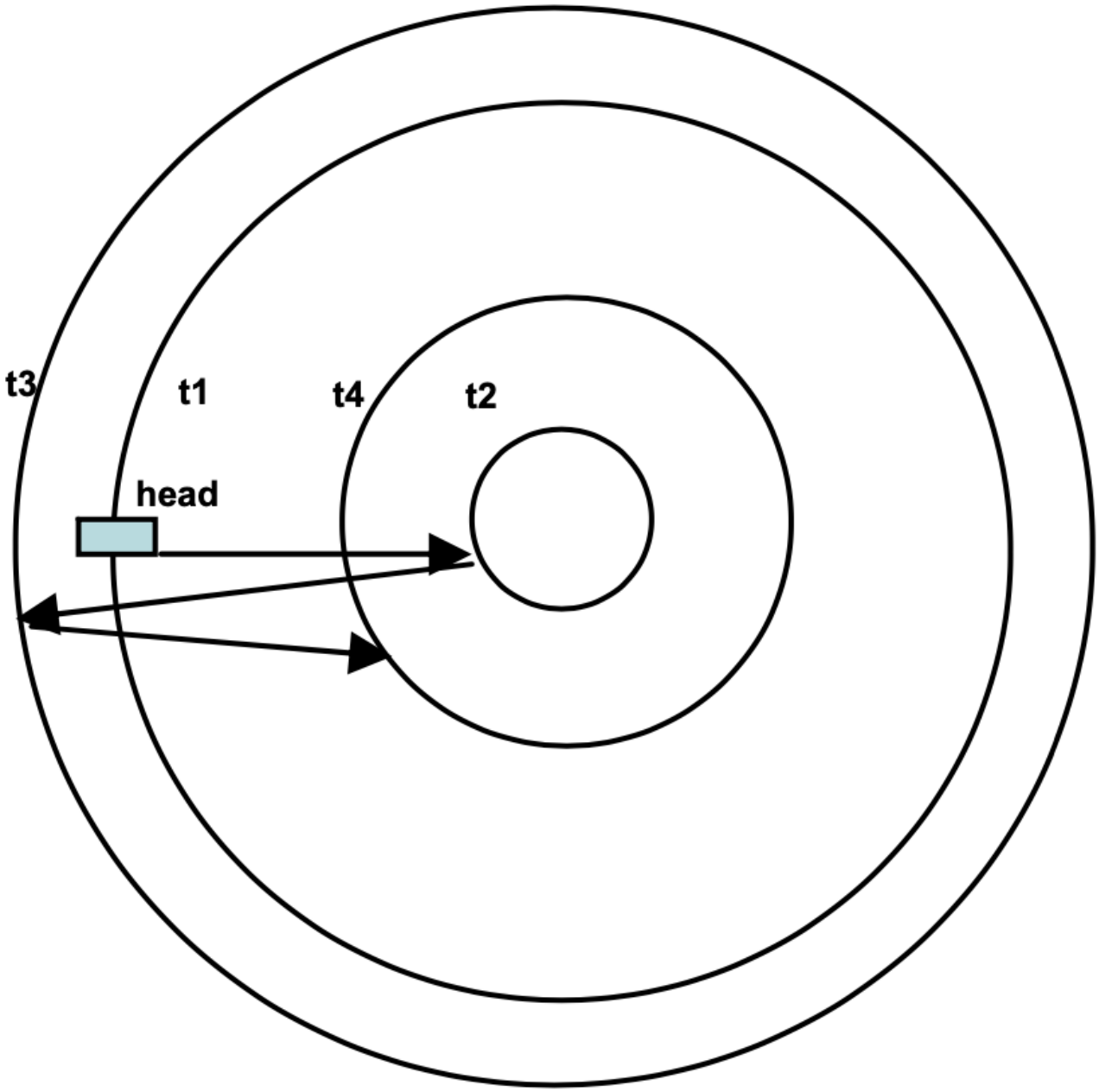


Figure 10.11: Movement of Disk Head With FCFS

10.9.2 Shortest Seek Time First (SSTF)

Similar to SJF processor scheduling

- Minimizes the average wait time for a given set of requests and results in good throughput
- Potential of starving requests
- High variance

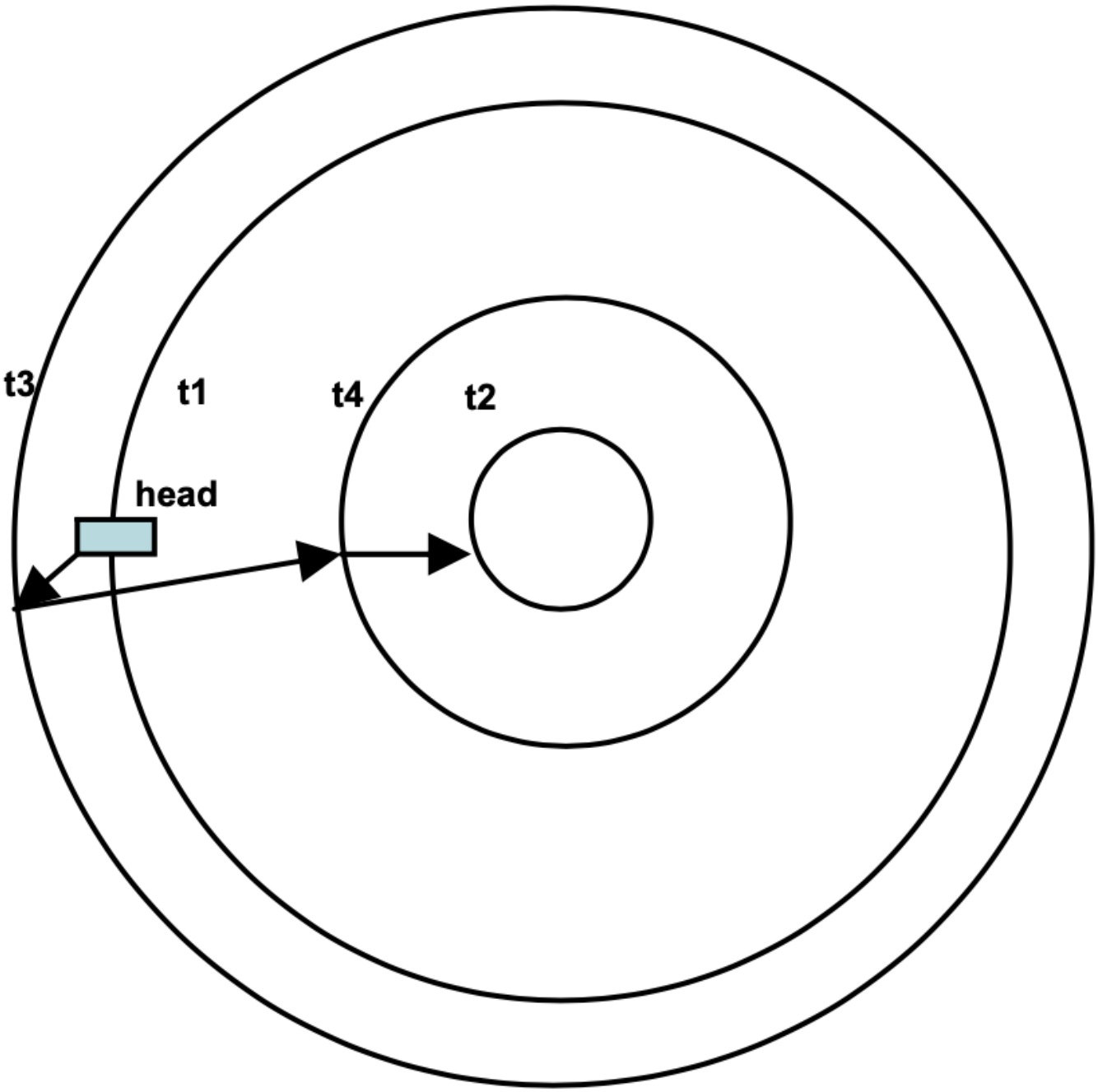


Figure 10.12: Movement of Disk Head With SSTF

10.9.3 Scan (elevator algorithm)

The head moves from its position of rest (track 0) towards the innermost track (track 199)

- algorithm services the requests that are en route from outermost to innermost track, regardless of the arrival order
- once reaching innermost track, reverse direction

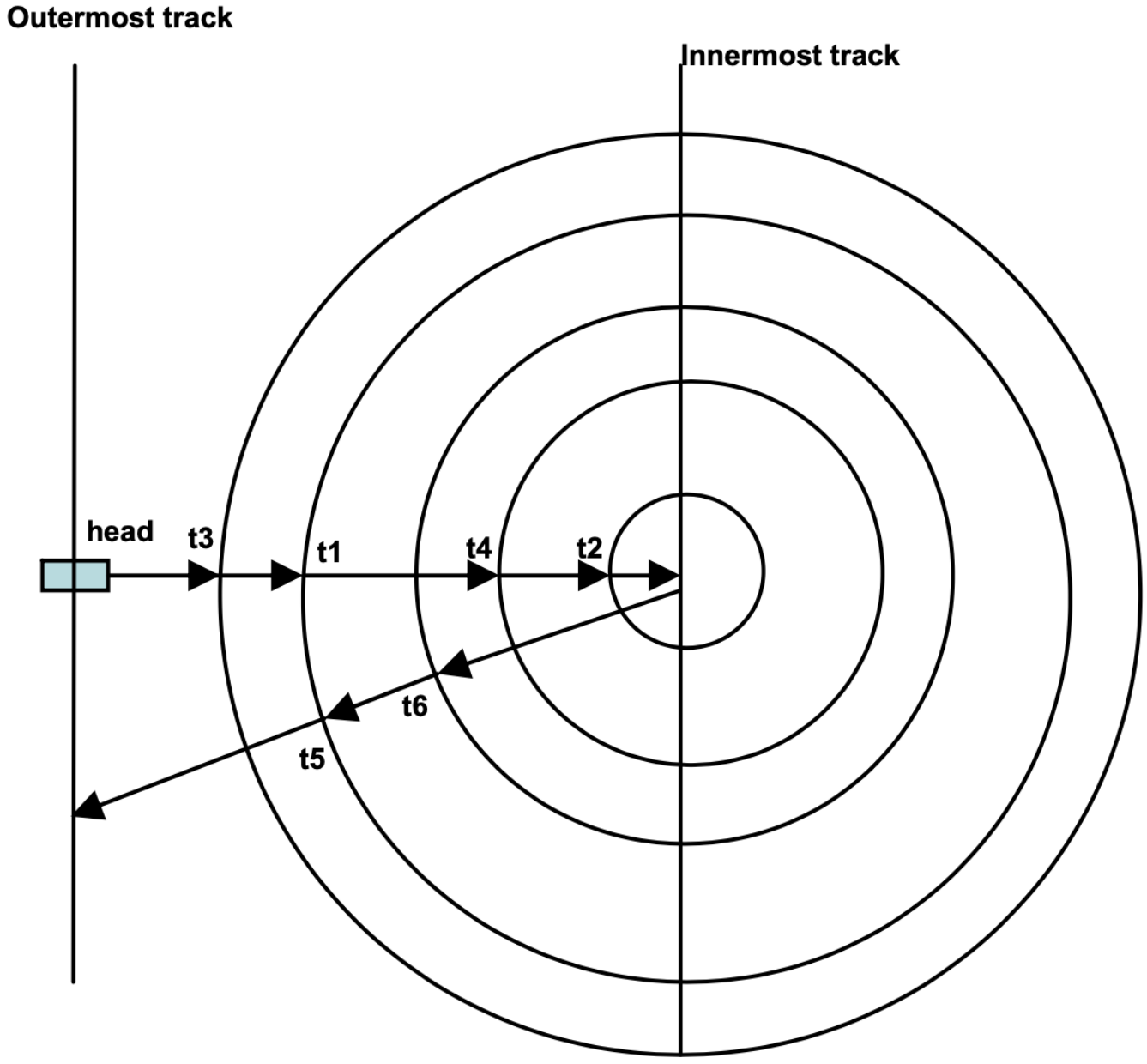


Figure 10.13: Movement of Disk Head with SCAN

Similar to how elevators work

- lower variance in wait time compared to SSTF, and overall has an average wait time that is similar to SSTF

- Limit on starvation

10.9.4 C-Scan (Circular Scan)

Similar to SCAN but on retracting to rest position, we do not service any requests

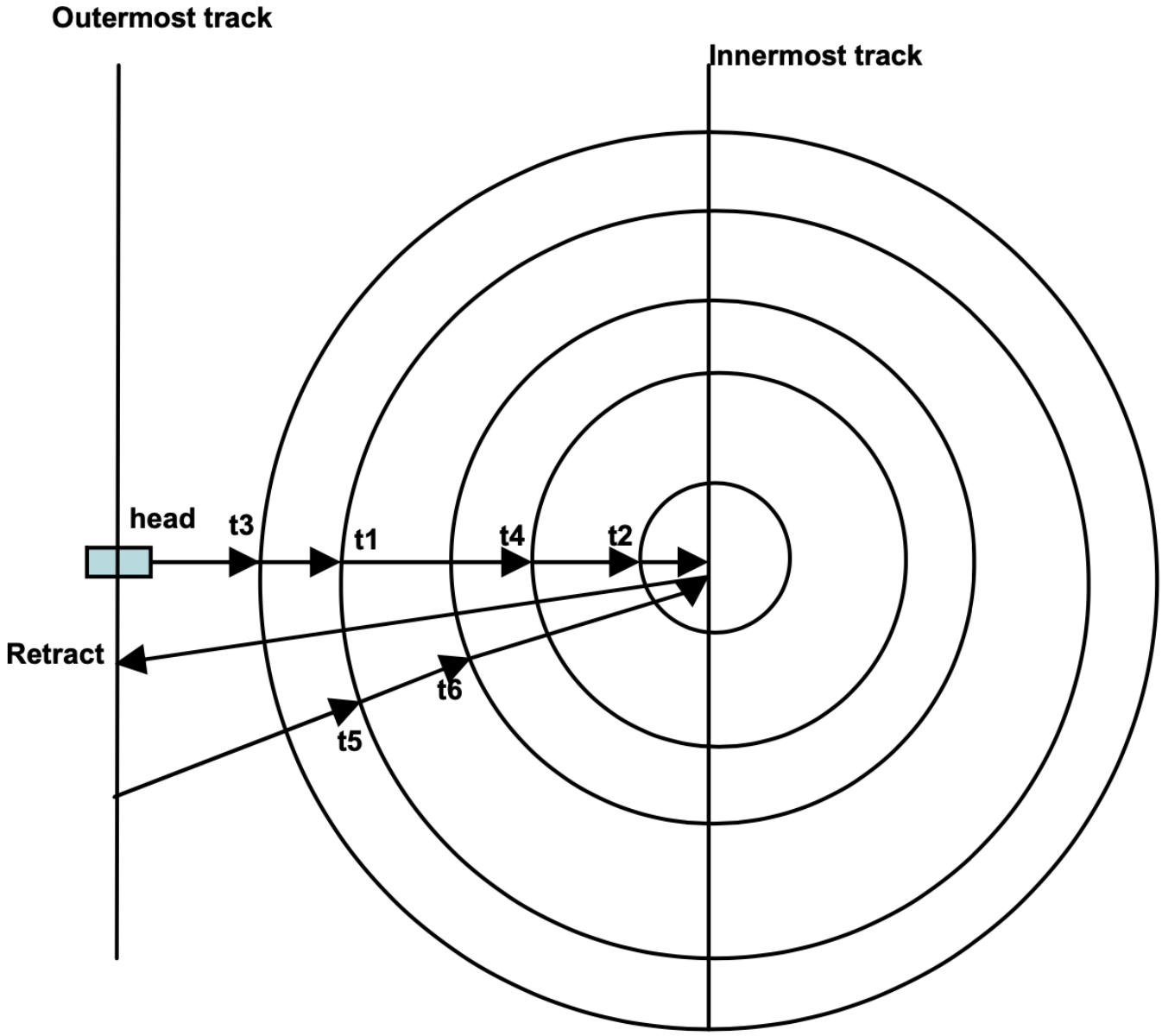


Figure 10.14: Movement of Disk Head with C-SCAN

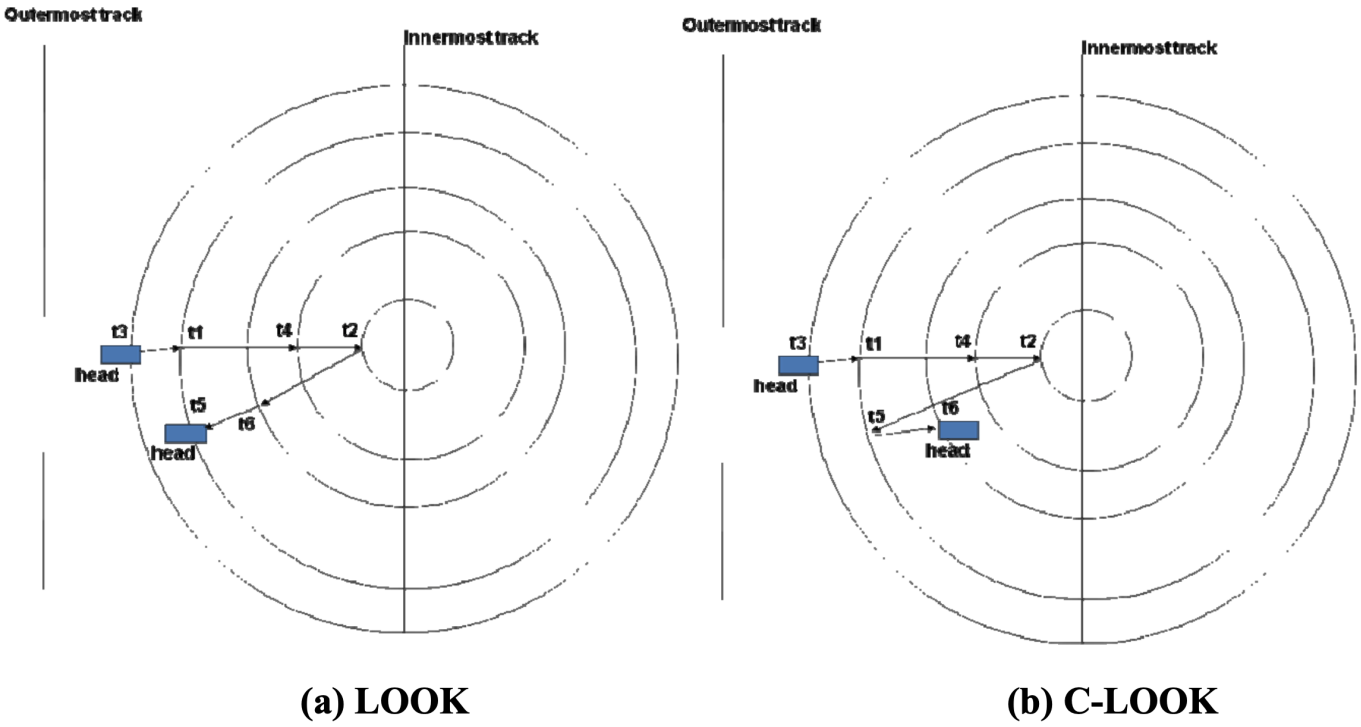
By ignoring requests in the reverse direction, C-SCAN removes the bias that the SCAN algorithm has for requests clustered around the middle tracks of the disk

- Reduces unfairness in servicing
- Lower variance in waiting time compared to SCAN

10.9.5 Look and C-Look

Same concepts as SCAN but instead of going all the way to either end we just go to deepest request

- Even better performance than SCANs



(a) LOOK **(b) C-LOOK**

Figure 10.15: Movement of Disk Head with LOOK and C-LOOK

10.10 Solid State Drive

Figure 10.22: A Railroad Switch[14]

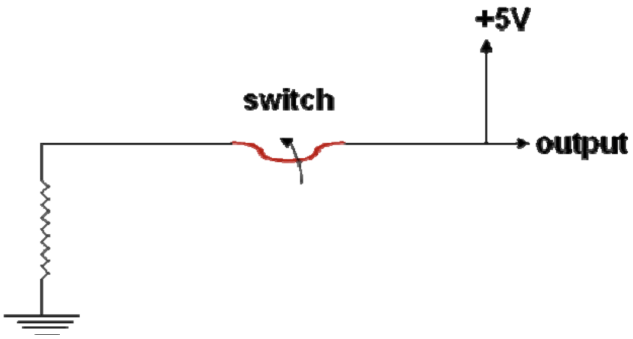


Figure 10.23: An Electrical Switch

Once the switch is thrown the incoming track (in the figure above) remains connected to the chosen fork. A ROM works in exactly the same manner

- We can do this with circuits
- Allow the bit patterns in the ROM to be electrically erased and re-programmed to contain a different bit pattern to form EEPROM

Problems with SSDs

- density of storage (higher cost per byte)
- Lower read/write bandwidth
- Shorter life (partly mitigated by wear leveling)

Kishore is a bit outdated and SSDs are the way to go honestly

10.11 Evolution of I/O Buses and Device Drivers

> In recent times, it is impossible for a computer enthusiast not to have heard of terminologies such as USB and Firewire

- I doubt anyone uses Firewire anymore now that USB 3 and 4 is a thing

Parallelism helps in overcoming latency issues and boosts the overall throughput by pumping the bits on parallel wires

- Suffers from cross talk at higher speeds without careful shielding of the parallel wires
- More expensive as there are more connectors

Upon power up, the processor automatically starts executing a *bootstrapping code* that is in a well-known fixed location in read-only memory (ROM)

- Initializes system including the peripheral devices before transferring the control to the upper layers of the system software
- This bootstrapping code is called the *BIOS*

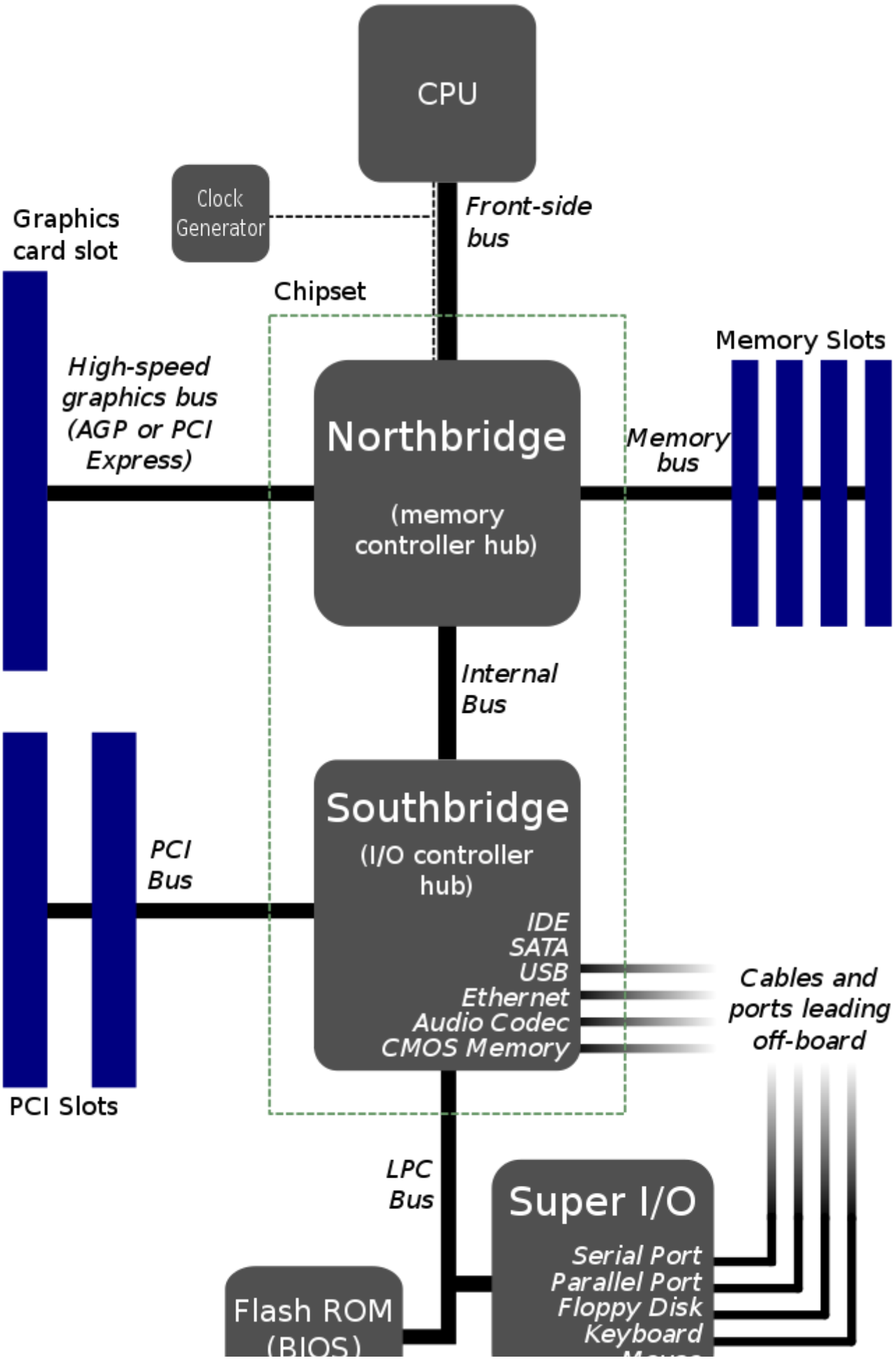

CPU
Clock Generator
Front-side bus
Graphics card slot
Chipset
Memory Slots
High-speed graphics bus (AGP or PCI Express)
Northbridge
(memory controller hub)
Memory bus
Internal Bus
Southbridge
(I/O controller hub)
PCI Bus
IDE
SATA
USB
Ethernet
Audio Codec
CMOS Memory
Cables and ports leading off-board
PCI Slots
LPC Bus
Super I/O
Serial Port
Parallel Port
Floppy Disk
Keyboard
Flash ROM (BIOS)

Figure 10.16: Block Diagram of a Motherboard