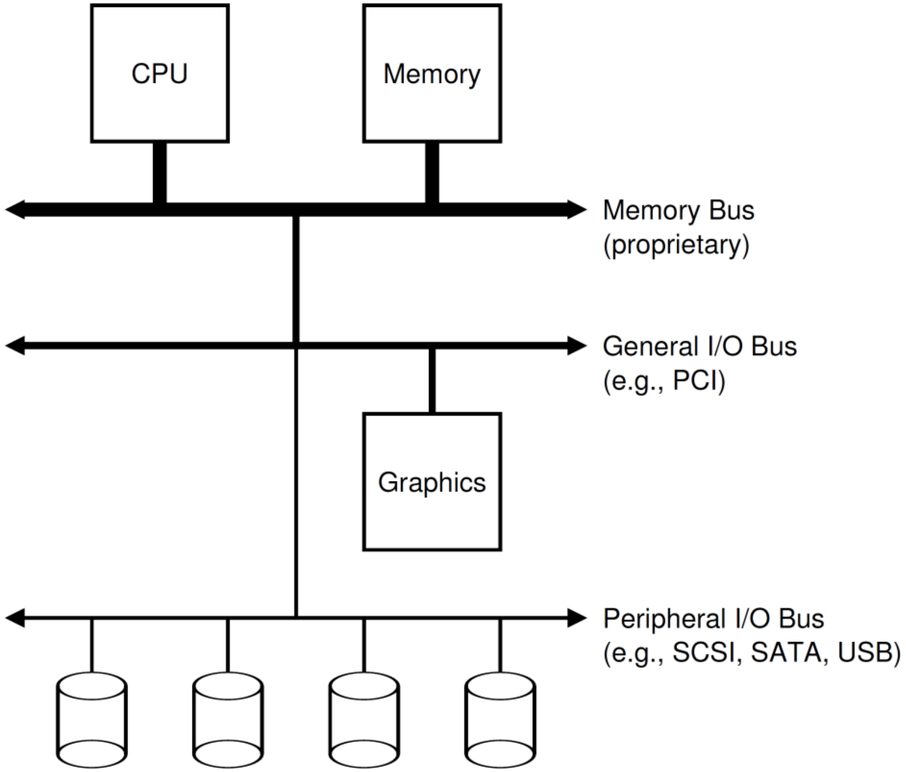
5118020-03 Operating System

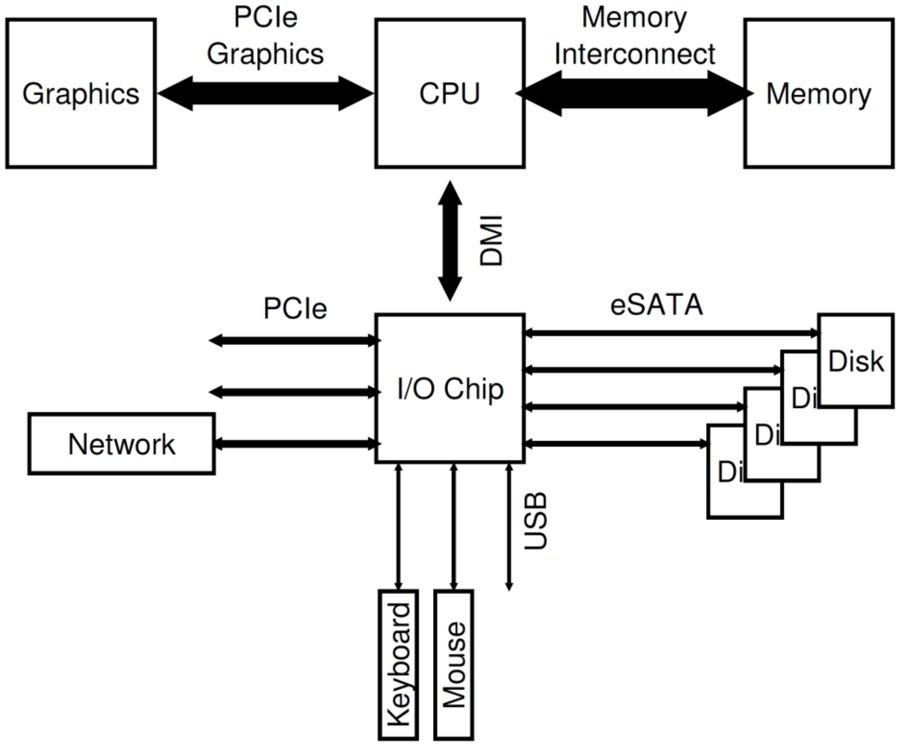
Input/Output Devices & Hard Disk Device

Chapters 36 & 37

Shin Hong

How I/O devices are attached to system architecture





- nearer memory bus, fasterer devices are connected

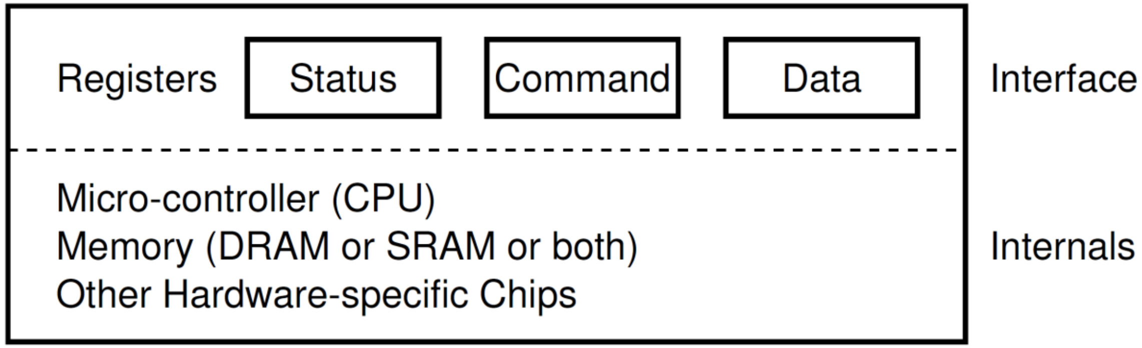
- Memory bus and graphic Bus
- I/O chip
  - Direct Media Interface

Method of Device Interaction

- via I/O instructions
  - CPU has previlaged instructions to send data to a specific device addressed by a port number
  - OS uses these instructions to send data and command to each device

- via memory instructions
  - Device registeres are mapped to specific memory addresses
  - OS uses memory read and write instructions to operate on the device

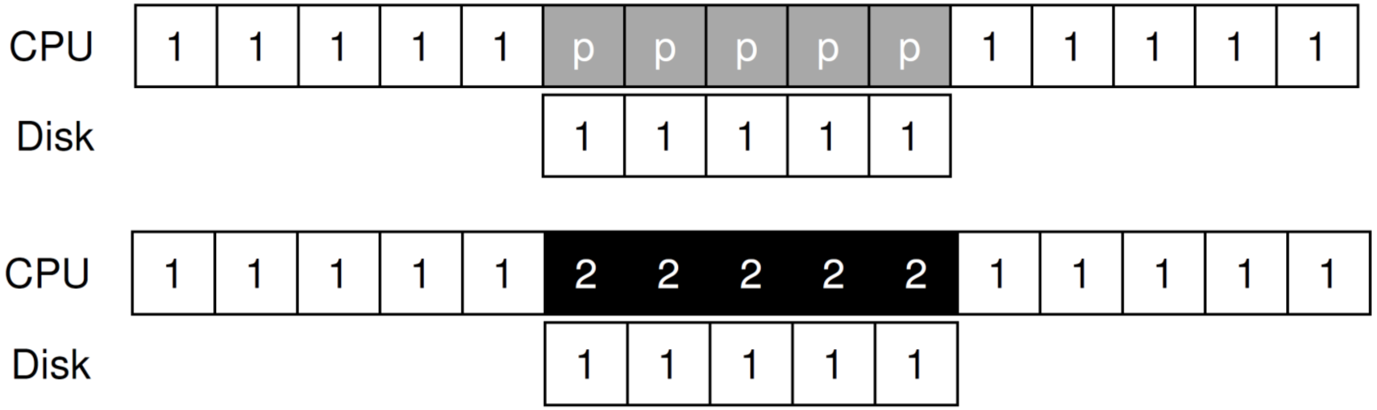
Interaction with Hardware Device - Polling




```
While (STATUS == BUSY)
    ; // wait until device is not busy          polling
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request   polling
```

Interaction with Hardware Device - Interrupt

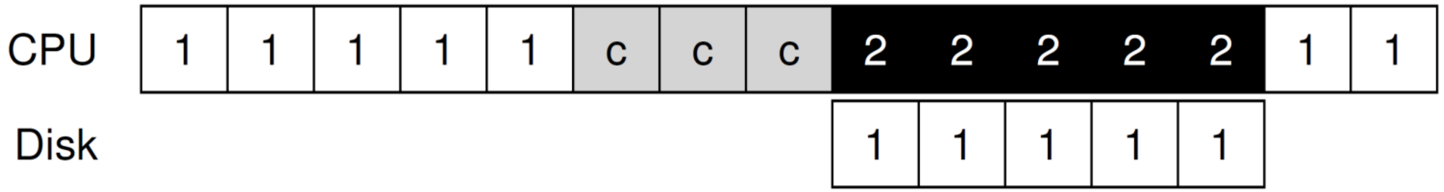
- polling vs. interrupt



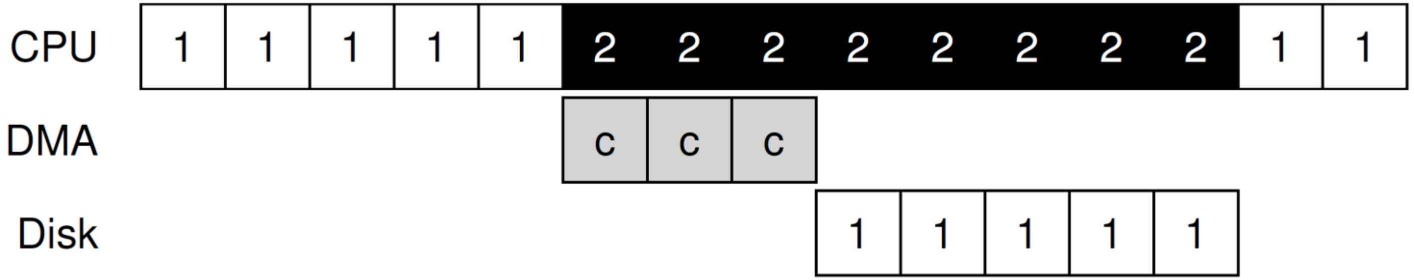
- efficient use of interrupt
  - two-phase approach: conduct polling for a short time period first, and then use interrupt
  - coalescing: merge multiple messages and deliver them once with a single interrupt to limit the number of interrupts in a time unit

Moving Data

- Programmed I/O
  - CPU moves each value one-by-one to the data register
  - Device register access time is far longer than memory access time
  - Example. writing data to disk



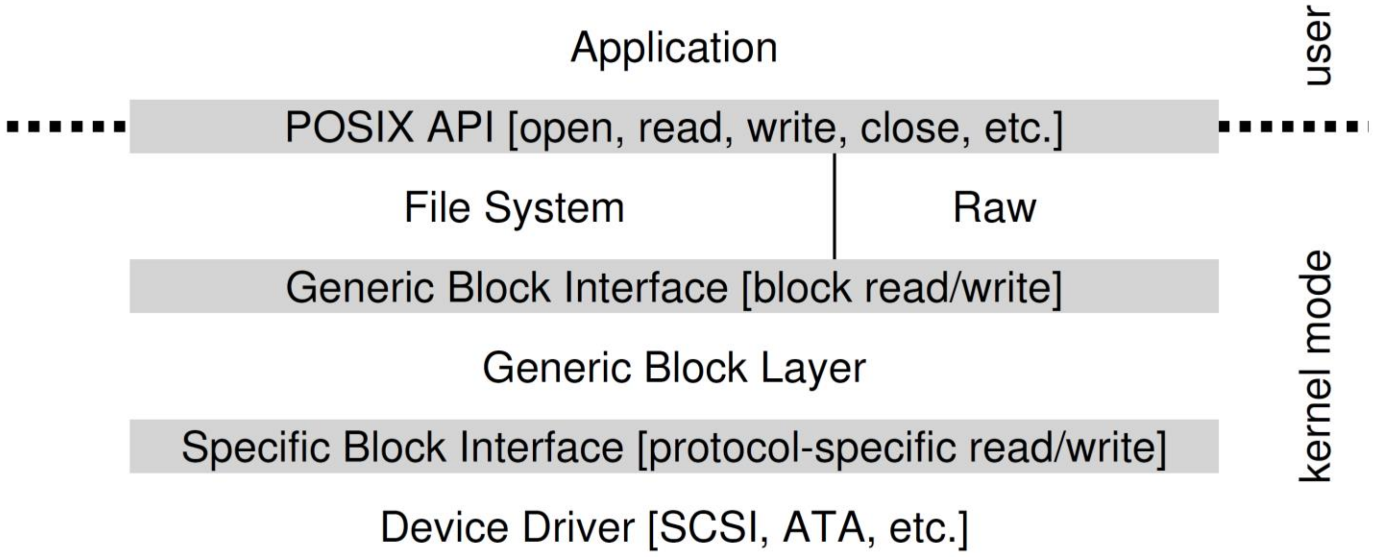
- Direct Memory Access (DMA)
  - CPU commands DMA to transfer data from memory to device registers
  - DMA raises an interrupt when it accomplishes the data transfer
  - Example. writing data to disk



Device Driver

- A device driver is a kernel module that encapsulates hardware details and provices interface for OS to control the device
  - the devices of the same kind share the same interface
  - for Linux, device driver modules take around 70% of kernel code

- E.g. disk device drivers and file system stack



Example – IDE Disk Driver

- Four interface registers
  - Control
  - Command block
  - Status
  - Error

- Each register can be read and written by the I/O instructions by its I/O address

```
Control Register:
  Address 0x3F6 = 0x08 (0000 1RE0): R=reset,
                  E=0 means "enable interrupt"

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7      6     5     4    3    2    1     0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when ERROR==1)
    7      6     5     4    3    2    1    0
   BBK    UNC   MC    IDNF MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```

Example – IDE Disk Driver

- Protocol
  - Wait for device to be ready: wait until Status becomes Ready, not Busy
  - Write Sector Count, Logical Block Address of the sector to access, and Drive number
  - Write Read, or Write to Command block
  - Wait until Status is Ready and DRQ; write data to Data Port
  - Handle an interrupt for each sector transferred
  - After each operation, read Status and if its error bit is on, read Error

```
Control Register:
  Address 0x3F6 = 0x08 (0000 1RE0): R=reset,
                  E=0 means "enable interrupt"

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7      6     5      4    3    2    1     0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when ERROR==1)
    7      6     5      4    3    2    1    0
   BBK    UNC   MC    IDNF  MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```

System

Hard Disk Drives (HDD)

- HDD has been the main form of persistent data storage
- Most of the existing file systems have been predicated on the characteristics of HDD
- Interface
  - a drive consists of an array of **sectors**
    - each sector is 512-byte blocks
    - the sectors are numbered from 0 to $n - 1$
    - a single 512-byte write is guranateed to be atomic
  - a disk operation may involve multiple sectors (e.g., read 4 KB)
  - accessing two sectors near one-another is faster than accessing two sectors far apart

Physical Components

- multiple **platter** with two **surfaces**
  - a platter is an aluminum wafer coated with magnetic layer
  - each surface has a **disk head** attached to a **disk arm**

- multiple **tracks** on a surface
  - many thousands of tracks on a surface
  - a disk arm places the disk head to a desired track
  - a track comprises of multiple **sectors**

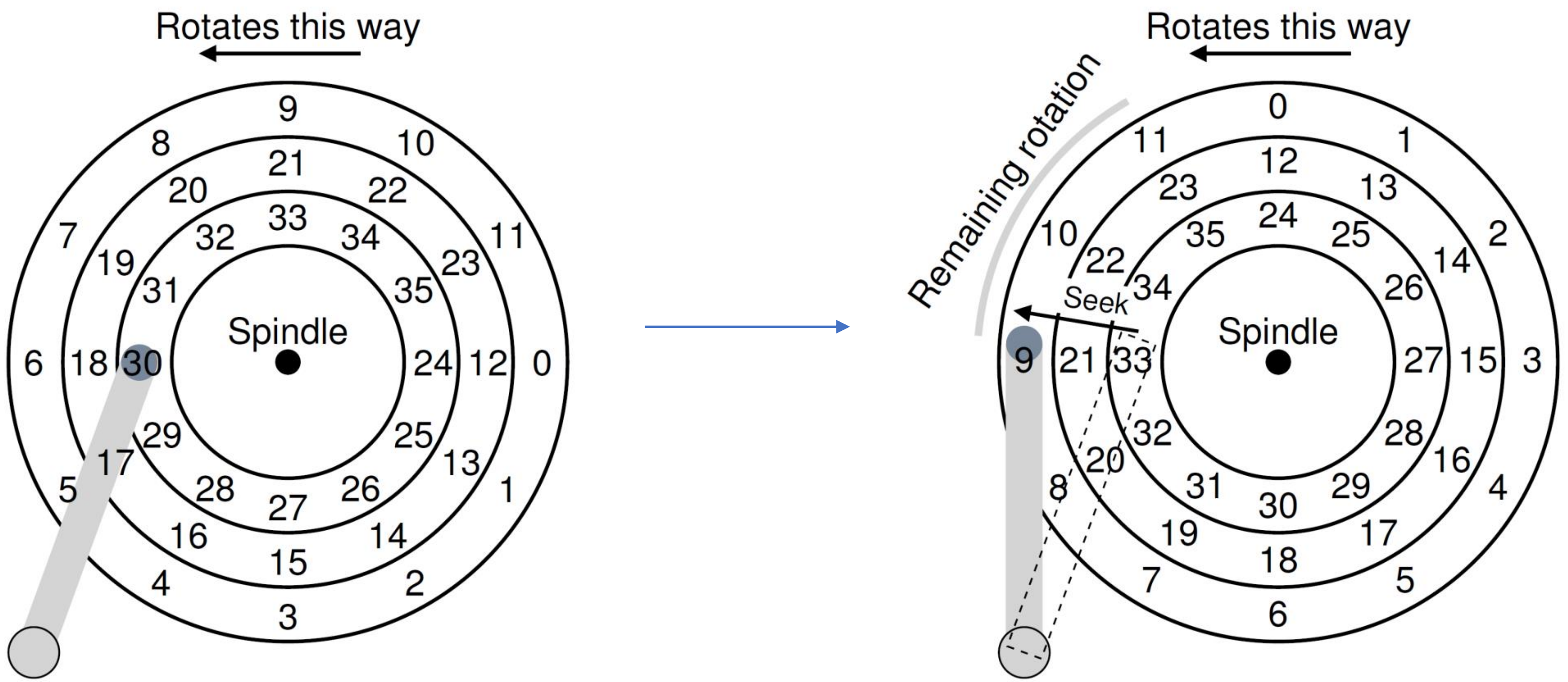
- **spindle**
  - the platterns are spinning at a constant rate
    (typically, 7200 to 15000 RPM)
  - a disk header can read/write data when the surface is spinning

How do hard drives work? - Kanawat Senanan

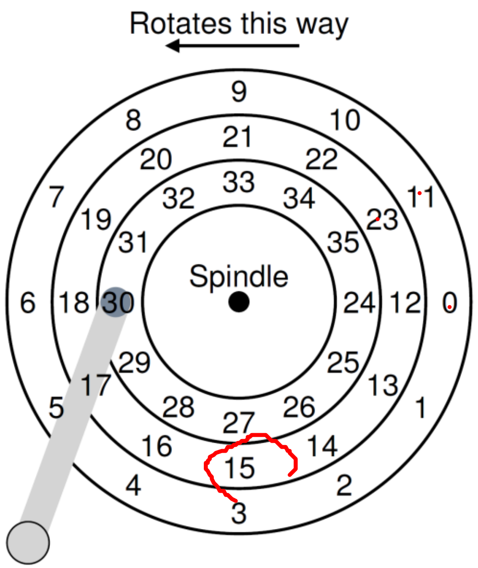
How Disks Work

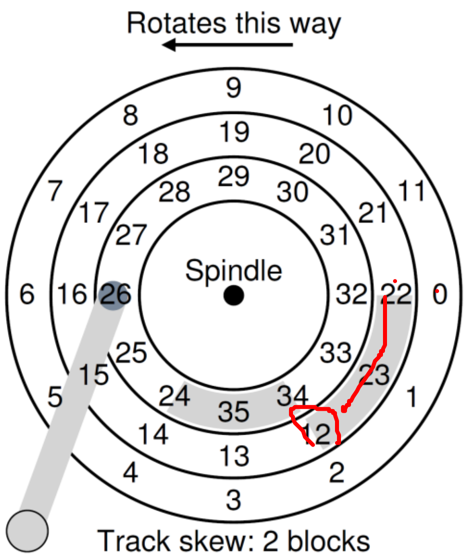
- for a given request of accessing a sector
  - identify the target surface and the target track
  - move the disk arm to the corresponding track position (seek)
    - acceleration -> coasting -> deceleration -> settling
    - seek time
  - wait for the desired sector to read to the head
    - rotational delay
  - read the magnetic signal or write data on the surface (transfer)

Example


Rotates this way
Spindle
Rotates this way
Remaining rotation
Seek
Spindle

Some Other Details





Track skew: 2 blocks

- track skew
  - considering disk arm replacement in consecutive sector accesses
- multi-zoned disk
  - tracks in an outer zone has more sectors than tracks in an inner zone
- track buffer
  - write back caching vs. write through

I/O Time

- I/O time and the rate of I/O

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

- High-performance disk and large-capacity disk

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

e.g., reading 4KB at a random location

Cheetah:

T_{seek} = 4ms

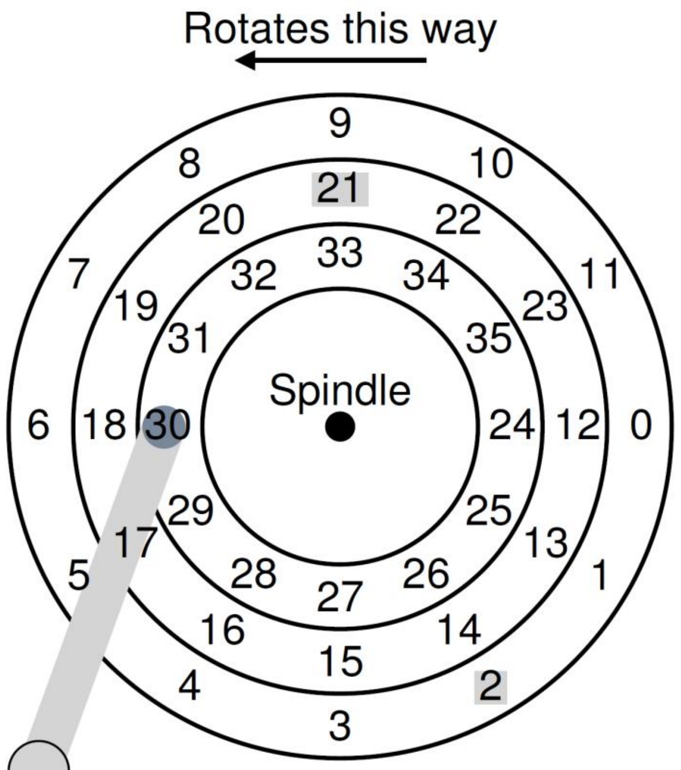
$T_{rotation}$ = (0 + 60000/15000) / 2 = 2 ms

$T_{transfer}$ = 4 / 125 ~= 0.03 ms

Disk Scheduling

- Given a set of I/O requests, the OS decides in which order the I/O requests are to be issued to the disk
- Basically, disk schedulers follow the principle of shortest job first

- Approach 1. Shortest Seek Time First (SSTF)
  - pick requests on the nearest track to the disk header first
  - estimate the distance between the header and the target sector by the difference of the sector numbers
  - have the starvation problem
  - example. sector 2 vs. sector 21



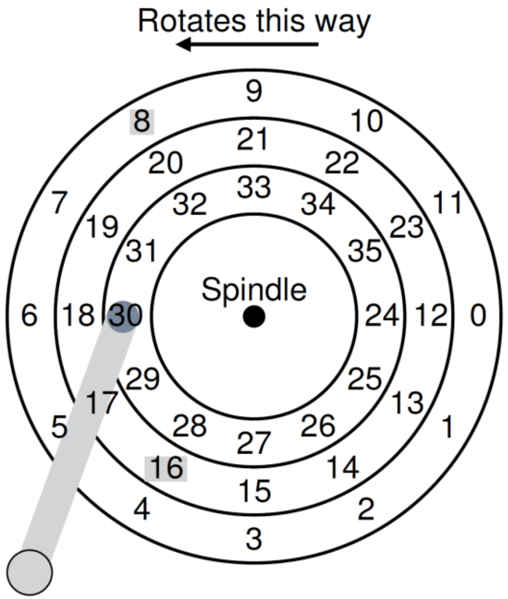
Elevator (or SCAN)

- Move the disk head back and forth across tracks to serve the requests for each track
  - a **sweep** is a single pass across the disk
  - avoid the disk starvation problem

- Variants
  - F-SCAN: freezes the request queue when it starts a new sweep
  - C-SCAN: sweeps from outer-to-inner only for fair scheduling

- Elevator algorithms are limited to optimize seeking time, but do not count the rotational delay

Shortest Positioning Time First (SPTF)



- Motivating example
  - which one is closer, sector 8 or sector 16?

- On modern devices, the seek time and the rotational delay are roughly equivalent, thus, both of these must be considered together at scheduling

- To implement SPTF, the OS side picks and issues best few I/O requests, then the disk controller finds the best SPTF order based on the internal information on the disk drive

Other Scheduling Issues

- merging
  - cluster requests for accessing consecutive blocks

- anticipatory disk scheduling
  - instead of serving given requests immediately, wait for a while to receive more requests and then schedule them together (i.e., non-work-conserving approach)