I/O Devices
Input/Output

- Mechanism to enable CPU to communicate with “outside” world
  - Memory
  - Persistent Storage
  - Remote resources/Network
Simplified I/O Architecture

- CPU
- MEM
- SSD
- Graphics
- Memory Bus
- General I/O Bus (PCI)
- Peripheral I/O Bus (SCSI, SATA USB)
- I/O CHIP
- PCIe
- eSATA
- USB
- Graph
- CPU
- MEM
Network within a server

- Memory bus, PCI bus, Peripheral I/O bus...

- What is a bus?
  - Common set of wires for communication among hardware devices
  - Allows connecting multiple devices over a singleton set of wires

- Split into three parts
  - Address bus: used to specify a physical address (e.g., memory location)
  - Control bus: used to carry commands from CPU, & signals from devices
  - Data bus: used for actual transfer of data

- Protocol:
  - Each bus has its own “protocol” for carrying out data transfer
Network within a server

- Memory bus, PCI bus, Peripheral I/O bus...

- Memory bus
  - Connects the main memory (DRAM) to the memory controller
  - Enables communication between CPU and DRAM

- General I/O (e.g., PCI express) bus
  - Connects variety of other devices to the root complex (host)
  - E.g., GPUs, SSDs, NICs (network interface cards), etc.

- Peripheral bus:
  - Connects variety of peripheral devices (and mass storage devices)
  - E.g., printers, mouse, keyboard, etc.
Interacting with a Device

Abstraction
(what the user sees)
Interacting with a Device

**Interface** (what the OS sees)

**Internals** (what is needed to implement the abstraction)
# Interacting with a Device

<table>
<thead>
<tr>
<th>Registers</th>
<th>Status</th>
<th>Command</th>
<th>Data</th>
</tr>
</thead>
</table>

**Internals**

- Microcontroller
- Memory
- Other device specific chips

*(what is needed to implement the abstraction)*
Interacting with a Device

- OS controls device by reading/writing registers

```plaintext
while (STATUS == BUSY)
    ; // wait until device is not busy
write data to DATA register
write command to COMMAND register
    // starts device and executes command
while (STATUS == BUSY)
    ; // wait until device is done with request
```

<table>
<thead>
<tr>
<th>Registers</th>
<th>Status</th>
<th>Command</th>
<th>Data</th>
</tr>
</thead>
</table>

Internals
(what is needed to implement the abstraction)

- Microcontroller
- Memory
- Other device specific chips
Tuning It Up

CPU is polling

- use interrupts
- run another process while device is busy
- what if device returns very quickly?

CPU is copying all the data to and from DATA

- use Direct Memory Access (DMA)

```c
while (STATUS == BUSY) {
    // wait until device is not busy
    write data to DATA register
    write command to COMMAND register
    // starts device and executes command
    while (STATUS == BUSY) {
        // wait until device is done with request
    }
}
```
From interrupt-driven I/O to DMA

Interrupt driven I/O

- Device ↔ CPU ↔ RAM

for \( i = 1 \ldots n \)
  - CPU issues read request
  - device interrupts CPU with data
  - CPU writes data to memory
From interrupt-driven I/O to DMA

- **Interrupt driven I/O**
  - Device ↔ CPU ↔ RAM
  - for \( i = 1 \ldots n \)
    - CPU issues read request
    - device interrupts CPU with data
    - CPU writes data to memory

- **+ Direct Memory Access**
  - Device ↔ RAM
  - CPU sets up DMA request
  - Device puts data on bus & RAM accepts it
  - Device interrupts CPU when done
How can the OS handle a multitude of devices?

- Abstraction!
  - Encapsulate device specific interactions in a **device driver**
  - Implement device neutral interfaces above device drivers

- Humans are about 70% water...
  - ...OSs are about 70% device drivers!

<table>
<thead>
<tr>
<th>File System Stack (simplified)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
</tr>
<tr>
<td>POSIX API [open, read, write, close, etc]</td>
</tr>
<tr>
<td><strong>File System</strong></td>
</tr>
<tr>
<td>Generic Block Interface [block read/write]</td>
</tr>
<tr>
<td><strong>Block Cache</strong></td>
</tr>
<tr>
<td><strong>Generic Block Layer</strong></td>
</tr>
<tr>
<td>Protocol-specific Block Interface</td>
</tr>
<tr>
<td><strong>Device Driver [SCSI, ATA, etc]</strong></td>
</tr>
<tr>
<td>Memory-mapped I/O, DMA, Interrupts</td>
</tr>
<tr>
<td><strong>Physical Device</strong></td>
</tr>
</tbody>
</table>
Persistent Storage
Storage Devices

We focus on two types of persistent storage:
- magnetic disks
  - servers, workstations, laptops
- flash memory
  - smart phones, tablets, cameras, laptops

Other exist(ed):
- tapes
- drums
- clay tablets
Magnetic disks vs Flash

- Magnetic disks
  - Large capacity at low cost
  - Block-level random access
  - Poor random access performance
  - Better sequential access performance

- Flash memory
  - Capacity at intermediate cost
  - Block-level random access
  - Good performance for reads
  - Worse performance for writes
  - Wear issues
The SSD Storage Hierarchy

- **Cell**: 1 to 4 bits
- **Page**: 2 to 8 KB, not to be confused with a VM page
- **Block**: 64 to 256 pages, not to be confused with a disk block
- **Plane/Bank**: Many blocks (Several Ks)
- **Flash Chip**: Several banks that can be accessed in parallel
Basic Flash Operations

- **Read (a page)**
  - 10s of $\mu$s, independent of the previously read page
  - great for random access!

- **Erase (a block)**
  - sets the entire block (with all its pages) to 1 (!)
  - very coarse way to write 1s...
  - 1.5 to 2 ms

- **Program (a page)**
  - can change some bits in a page of an erased block to 0
  - 100s of $\mu$s
  - changing a 0 bit back to 1 requires erasing the entire block!
Banks

Each bank contains many blocks

Bank 0

Bank 2

Bank 3
After an Erase, all cells are discharged (i.e., store 1s)
Block

Program

1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 0 0 1
Block

Program

Program
If now we want to set this bit to 1, we need to erase the entire block!

Modified pages must be copied elsewhere, or lost!
Every erase/program cycle adds some charge to a block; over time, hard to distinguish 1 from 0!
Using Flash Memory

Need to map reads and writes to logical blocks to **read**, **program**, and **erase** operations on flash

Flash Translation Layer (FTL)
From Flash to SSD

Flash Translation Layer

- tries to minimize
  - write amplification: \[ \frac{\text{write traffic (bytes) to flash chips}}{\text{write traffic (bytes) from client to SSD}} \]
  - wear out: practices wear leveling
  - disturbance: writes pages in a block in order, low to high
Log Structured FTL

Think of flash storage as implementing a log.
Think of flash storage as implementing a log

- On a write, program next available page of physical block being currently written
  - i.e., “append” the write to your log

- On a read, find in the log the page storing the logical block
  - don’t want to scan the whole log...
  - keep an in-memory map from logical blocks to pages!
Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

**A logical block maps to a physical page**

Client operations

1) **Erase(00)**
SSD’s clients read/write 4KB logical blocks

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Client operations

\[
\text{Write (a1, 100)}
\]

2) Program(00)
Example

- SSD’s clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

<table>
<thead>
<tr>
<th>Block</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
</tbody>
</table>

Flash Chip

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>00</td>
</tr>
</tbody>
</table>

Client operations: Write (a1, 100)
Example

- SSD's clients read/write 4KB **logical blocks**
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

---

<table>
<thead>
<tr>
<th>Block</th>
<th>00</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>10</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
</tbody>
</table>

**Client operations**

- Write (a1, 100)
- Write (a2, 101)

3) **Program(01)**
Example

SSD’s clients read/write 4KB logical blocks

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Client operations

- Write (a1, 100)
- Write (a2, 101)
Example

SSD’s clients read/write 4KB **logical blocks**

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

<table>
<thead>
<tr>
<th>Table</th>
<th>100</th>
<th>00</th>
<th>101</th>
<th>01</th>
<th>2000</th>
<th>02</th>
<th>2001</th>
<th>03</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td>a2</td>
<td>b1</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
</tbody>
</table>

**Client operations**

- Write (a1, 100)
- Write (a2, 101)
- Write (b1, 2000)
- Write (b2, 2001)
SSD's clients read/write 4KB logical blocks

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table: 100 ➔ 00 101 ➔ 01 2000 ➔ 02 2001 ➔ 03

Memory

<table>
<thead>
<tr>
<th>Block</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td>a2</td>
<td>b1</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
</tbody>
</table>

Flash Chip

Client operations

- Write (c1, 100)
- Erase(01)
Example

- SSD’s clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

<table>
<thead>
<tr>
<th>Table</th>
<th>100</th>
<th>00</th>
<th>101</th>
<th>01</th>
<th>2000</th>
<th>02</th>
<th>2001</th>
<th>03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td></td>
<td>00</td>
<td></td>
<td>01</td>
<td>2000</td>
<td>02</td>
<td>2001</td>
<td>03</td>
</tr>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td>a2</td>
<td>b1</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

Flash Chip

Program(04)

Client operations

Write (c1, 100)
Example

SSD's clients read/write 4KB **logical blocks**

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

<table>
<thead>
<tr>
<th>Table</th>
<th>100</th>
<th>00</th>
<th>101</th>
<th>01</th>
<th>2000</th>
<th>02</th>
<th>2001</th>
<th>03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td></td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td>a2</td>
<td>b1</td>
<td>b2</td>
<td>c1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

Flash Chip

Client operations

Write (c1, 100)
**Example**

- SSD’s clients read/write 4KB **logical blocks**

- Many physical SSD blocks; each holds 4 pages, each 4KB

*A logical block maps to a physical page*

<table>
<thead>
<tr>
<th>Table</th>
<th>100</th>
<th>04</th>
<th>101</th>
<th>01</th>
<th>2000</th>
<th>02</th>
<th>2001</th>
<th>03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td></td>
<td>00</td>
<td></td>
<td>01</td>
<td></td>
<td>02</td>
<td></td>
<td>03</td>
</tr>
<tr>
<td>Page</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
</tr>
<tr>
<td>Content</td>
<td>a1</td>
<td>a2</td>
<td>b1</td>
<td>b2</td>
<td>c1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

**Client operations**

- Write (c1, 100)
SSD’s clients read/write 4KB logical blocks

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

**Client operations**

- Write (c1, 100)
- Write (c2, 101)
SSD’s clients read/write 4KB logical blocks

Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Client operations

Write (c1, 100)
Write (c2, 101)
## APIs

<table>
<thead>
<tr>
<th></th>
<th>HDD</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>read</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sector</td>
<td></td>
<td>read page</td>
</tr>
<tr>
<td><strong>write</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sector</td>
<td></td>
<td>program page (0’s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>erase block (1’s)</td>
</tr>
</tbody>
</table>

## Performance

<table>
<thead>
<tr>
<th></th>
<th>HDD</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≈130MB/s (sequential)</td>
<td>≈200MB/s (random or sequential)</td>
<td></td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≈10ms</td>
<td></td>
<td>read 25μs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program 200–300μs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>erase 1.5–2 ms</td>
</tr>
</tbody>
</table>
Input/Output

- But, the OS must support all hardware devices ....

- Devices have different internal architecture/design:
  - OS: how can we standardize the interfaces to these devices?
  - Answer: OS File system, OS block layer (1 lecture)

- Devices are unreliable:
  - OS: how can we give the illusion of a reliable device?
  - Answer: RAID design (1 lecture)

- Devices have unpredictable performance:
  - OS: how can we manage them if we don’t know device behavior
  - Answer: OS schedulers (1 lecture)
Magnetic disk

- Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum
Disk Drive Schematic

- Data on a track can be read without moving the arm.
- Track skewing staggers logical address 0 on adjacent one to account for time to move head.
- The set of tracks on different surfaces with the same track index is as follows:
  
- **Track**
  - Typically 512 bytes
  - Spare sectors added for fault tolerance

- **Block/Sector**
  - Reads by sensing a magnetic field.
  - Writes by creating one.

- **Platter**
  - Thin cylinder that holds magnetic material.

- **Cylinder**
  - 2018: 4200-15000 RPM

- **Surface**

- **Spindle**
  - Floats on air cushion created by spinning disk.

- **Head**
  - Arm assembly
Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0...N-1
- Heads move to appropriate track
  - seek
  - settle
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time

Disk access time:
Disk Read/Write

Present disk with a sector address
- Old: CHS = (cylinder, head, sector)
- New abstraction: Logical Block Address (LBA)
  - linear addressing 0...N-1

Heads move to appropriate track
- seek (and though shalt approximately find)
- settle (fine adjustments)

Appropriate head is enabled

Wait for sector to appear under head
- rotational latency

Read/Write sector
- transfer time

Disk access time:

\[
\text{seek time} + \text{transfer time}
\]
Disk Read/Write

Present disk with a sector address
- Old: CHS = (cylinder, head, sector)
- New abstraction: Logical Block Address (LBA)
  - linear addressing 0...N-1

Heads move to appropriate track
- seek (and though shalt approximately find)
- settle (fine adjustments)

Appropriate head is enabled

Wait for sector to appear under head
- rotational latency

Read/Write sector
- transfer time

Disk access time:

seek time +
rotation time +
Disk Read/Write

Present disk with a sector address
- Old: CHS = (cylinder, head, sector)
- New abstraction: Logical Block Address (LBA)
  - linear addressing 0...N-1

Heads move to appropriate track
- seek (and though shalt approximately find)
- settle (fine adjustments)

Appropriate head is enabled

Wait for sector to appear under head
- rotational latency

Read/Write sector
- transfer time

Disk access time:
- seek time +
- rotation time +
- transfer time
Seek time:
A closer look

- **Minimum:** time to go from one track to the next
  - 0.3-1.5 ms
- **Maximum:** time to go from innermost to outermost track
  - more than 10ms; up to over 20ms
- **Average:** average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across disk
Seek time: A closer look

- **Minimum**: time to go from one track to the next
  - 0.3-1.5 ms

- **Maximum**: time to go from innermost to outermost track
  - more than 10ms; up to over 20ms

- **Average**: average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across disk

- **Head switch time**: time to move from track on one surface to the same track on a different surface
  - range similar to minimum seek time
Rotation time: A closer look

- Today most disk rotate at 4200 to 15,000 RPM
  - ≈15ms to 4ms per rotation
  - good estimate for rotational latency is half that amount
- Head starts reading as soon as it settles on a track
  - track buffering to avoid “shoulda coulda” if any of the sectors flying under the head turn out to be needed
Transfer time: A closer look

Surface transfer time

- Time to transfer one or more sequential sectors to/from surface after head reads/writes first sector
- Much smaller than seek time or rotational latency
  - 512 bytes at 100MB/s ≈ 5µs (0.005 ms)
- Lower for outer tracks than inner ones
  - same RPM, but more sectors/track: higher bandwidth!

Host transfer time

- Time to transfer data between host memory and disk buffer
  - 60MB/s (USB 2.0); 640 MB/s (USB 3.0); 25.6GB/s (Fibre Channel 256GFC)
Buffer Memory

- Small cache ["Track buffer", 8 to 16 MB]
  - holds data
    - read from disk
    - about to be written to disk

- On write
  - write back (return from write as soon as data is cached)
  - write through (return once it is on disk)
Computing I/O time

\[ T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} \]

The rate of I/O is computed as

\[ R_{I/O} = \frac{\text{Size}_{\text{Transfer}}}{T_{I/O}} \]
**Example:**

**Toshiba MK3254GSY** (2008)

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Platters/Heads</td>
<td>2/4</td>
</tr>
<tr>
<td>Capacity</td>
<td>320GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spindle speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Avg. seek time R/W</td>
<td>10.5/12.0 ms</td>
</tr>
<tr>
<td>Max. seek time R/W</td>
<td>19 ms</td>
</tr>
<tr>
<td>Track-to-track</td>
<td>1 ms</td>
</tr>
<tr>
<td>Surface transfer time</td>
<td>54-128 MB/s</td>
</tr>
<tr>
<td>Host transfer time</td>
<td>375 MB/s</td>
</tr>
<tr>
<td>Buffer memory</td>
<td>16MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical</td>
<td>16.35 W</td>
</tr>
<tr>
<td>Idle</td>
<td>11.68 W</td>
</tr>
</tbody>
</table>
500 Random Reads

### Workload
- 500 read requests, randomly chosen sectors
- Served in FIFO order

### How long to service them?
- 500 times (seek + rotation + transfer)
- Seek time: 10.5 ms (avg)
- Rotation time:
  - 7200 RPM = 120 RPS
  - Rotation time: 8.3 ms
  - On average, half of that: 4.15 ms
- Transfer time:
  - At least 54 MB/s
  - 512 bytes transferred in \( \frac{0.5}{54,000} \) seconds = 9.26\( \mu \)s

### Total time:
\[
R_{I/O} = \frac{500 \times 0.5 \times 10^{-3} \text{MB}}{7.33 \text{ s}} = 0.034 \text{ MB/s}
\]
500 Sequential Reads

**Workload**
- 500 read requests for sequential sectors on the same track
- served in FIFO order

**How long to service them?**
- seek + rotation + 500 times transfer
- seek time: 10.5 ms (avg)
- rotation time:
  - 4.15 ms, as before
- transfer time
  - outer track: 500 x (.5/128000) ≈ 2ms
  - inner track: 500 x (.5/54000) seconds ≈ 4.6ms

**Total time is between:**
- outer track: (2 + 4.15 + 10.5) ms ≈ 16.65 ms
- inner track: (4.6 + 4.15 + 10.5) ms ≈ 19.25 ms

- $R_{I/O} = \frac{500 \times 5 \times 10^{-3} MB}{16.65 \text{ ms}} = 15.02 \text{ MB/s}$
- $R_{I/O} = \frac{500 \times 5 \times 10^{-3} MB}{19.25 \text{ ms}} = 12.99 \text{ MB/s}$

---

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td></td>
</tr>
<tr>
<td>Platters/Heads</td>
<td>2/4</td>
</tr>
<tr>
<td>Capacity</td>
<td>320GB</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td></td>
</tr>
<tr>
<td>Spindle speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Avg. seek time R/W</td>
<td>10.5/12.0 ms</td>
</tr>
<tr>
<td>Max. seek time R/W</td>
<td>19 ms</td>
</tr>
<tr>
<td>Track-to-track</td>
<td>1 ms</td>
</tr>
<tr>
<td>Surface transfer time</td>
<td>54-128 MB/s</td>
</tr>
<tr>
<td>Host transfer time</td>
<td>375 MB/s</td>
</tr>
<tr>
<td>Buffer memory</td>
<td>16MB</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>16.35 W</td>
</tr>
<tr>
<td>Idle</td>
<td>11.68 W</td>
</tr>
</tbody>
</table>
In a multiprogramming/time sharing environment, a queue of disk I/Os can form.

OS maximizes disk I/O throughput by minimizing head movement through disk head scheduling.

And this time we have a good sense of the length of the task!
FCFS

Assume a queue of request exists to read/write tracks

and the head is on track 65

FCFS scheduling results in disk head moving 550 tracks

and makes no use of what we know about the length of the tasks!
SSTF: Shortest Seek Time First

Greedy scheduling

Rearrange queue from: 83 72 14 147 16 150
to: 14 16 150 147 83 72

Head moves 221 tracks

BUT

- OS knows blocks, not tracks (easily fixed)
- starvation
SCAN Scheduling “Elevator”

- Move the head in one direction until all requests have been serviced, and then reverse
- sweeps disk back and forth

Rearrange queue from:  
83  72  14  147  16  150

...  to:  
150  147  83  72  14  16

Head moves 187 tracks.
C-SCAN scheduling

Circular SCAN

- sweeps disk in one direction (from outer to inner track), then resets to outer track and repeats

More uniform wait time than SCAN

- moves head to serve requests that are likely to have waited longer
OS Outsouces

Scheduling Decisions

Selecting which track to serve next should include rotation time (not just seek time!)

- SPTF: Shortest Positioning Time First

Hard for the OS to estimate rotation time accurately

- Hierarchical decision process
  - OS sends disk controller a batch of “reasonable” requests
  - disk controller makes final scheduling decisions