Operating Systems
Lecture 13

disk and fs abstraction

Prof. Mengwei Xu

Goals for Today

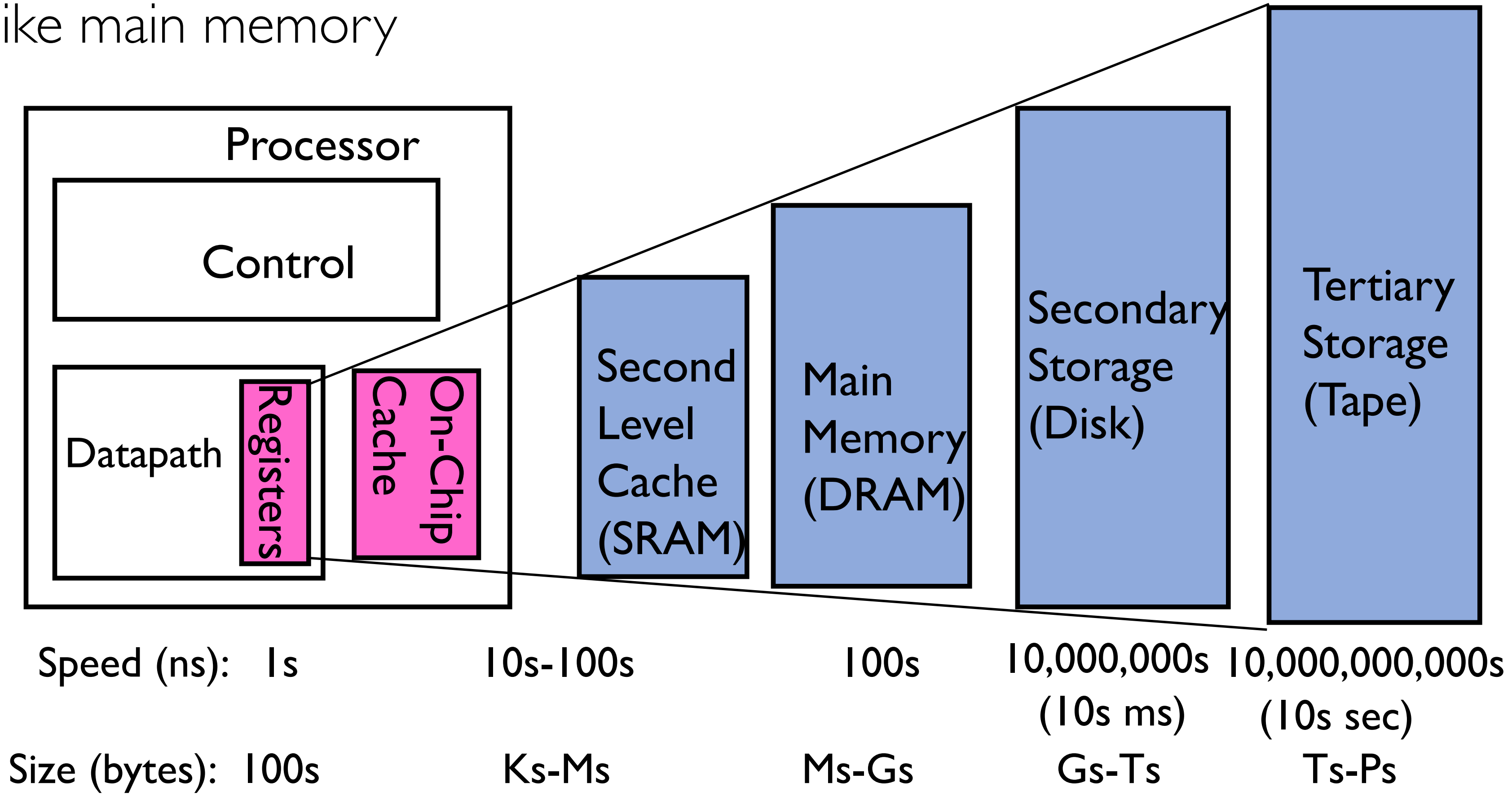
- Storage Devices
- File System Abstraction

Goals for Today

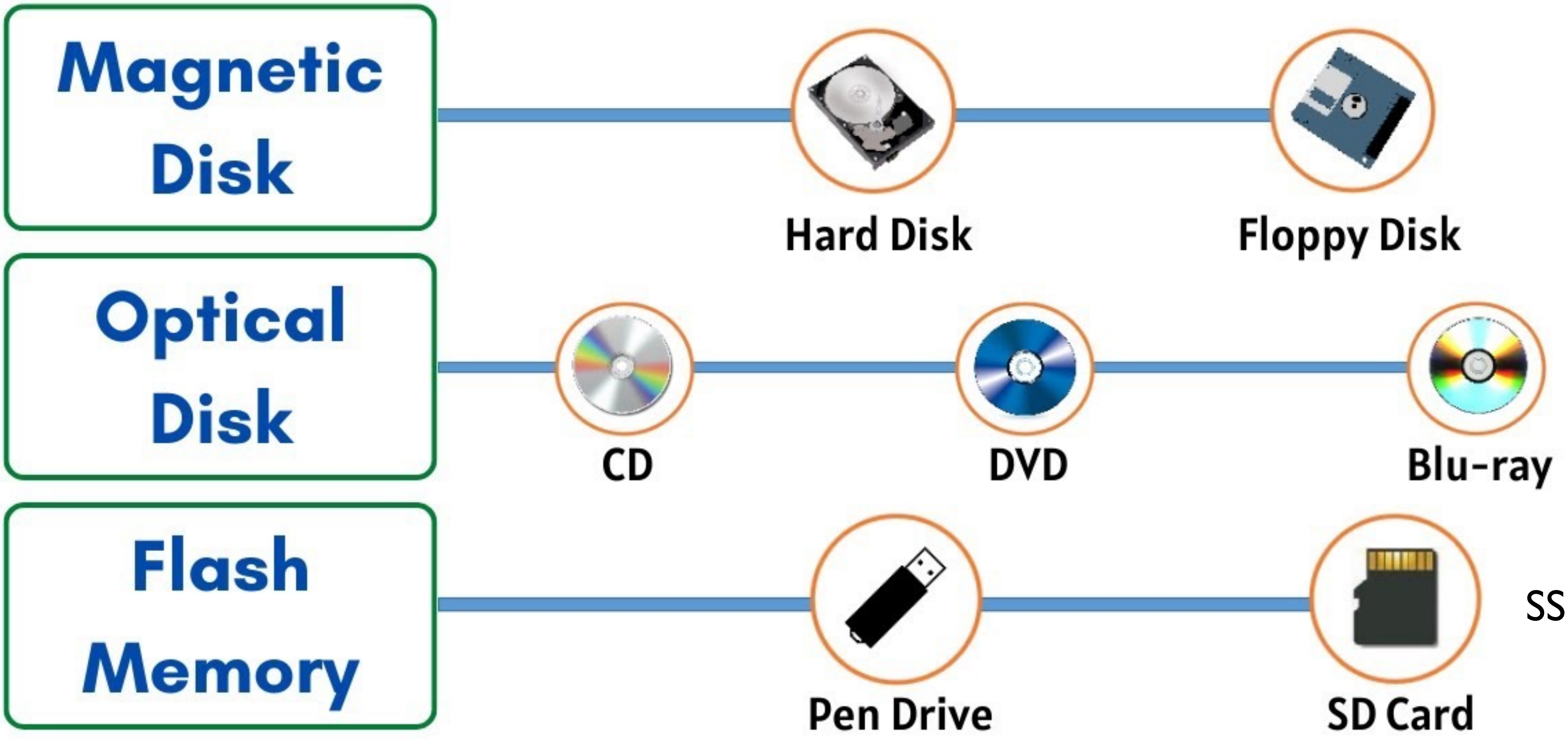
- Storage Devices
- File System Abstraction

Storage Devices

- Why we learn the hardware characteristics? Because they help us build better OSes and applications!
- As secondary storage to computers, storage devices are persistent.
  - Unlike main memory



Secondary Storage



Magnetic Disk

Hard Disk

Floppy Disk

Optical Disk

CD

DVD

Blu-ray

Flash Memory

Pen Drive

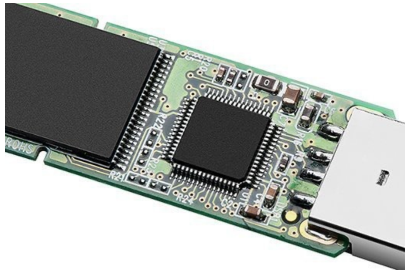
SD Card

SSDs

Storage Devices

1. Magnetic disks (磁盘)
   - Storage that rarely becomes corrupted
   - Large capacity at low cost
   - Block level random access
   - Slow performance for random access
   - Better performance for sequential access
2. Flash memory (闪存)
   - Storage that rarely becomes corrupted
   - Capacity at intermediate cost (5-20x disk)
   - Block level random access
   - Good performance for reads; worse for random writes
   - Erasure requirement in large blocks
   - Wear patterns issue

Servers, workstations, and labtops



Smartphones and tablets

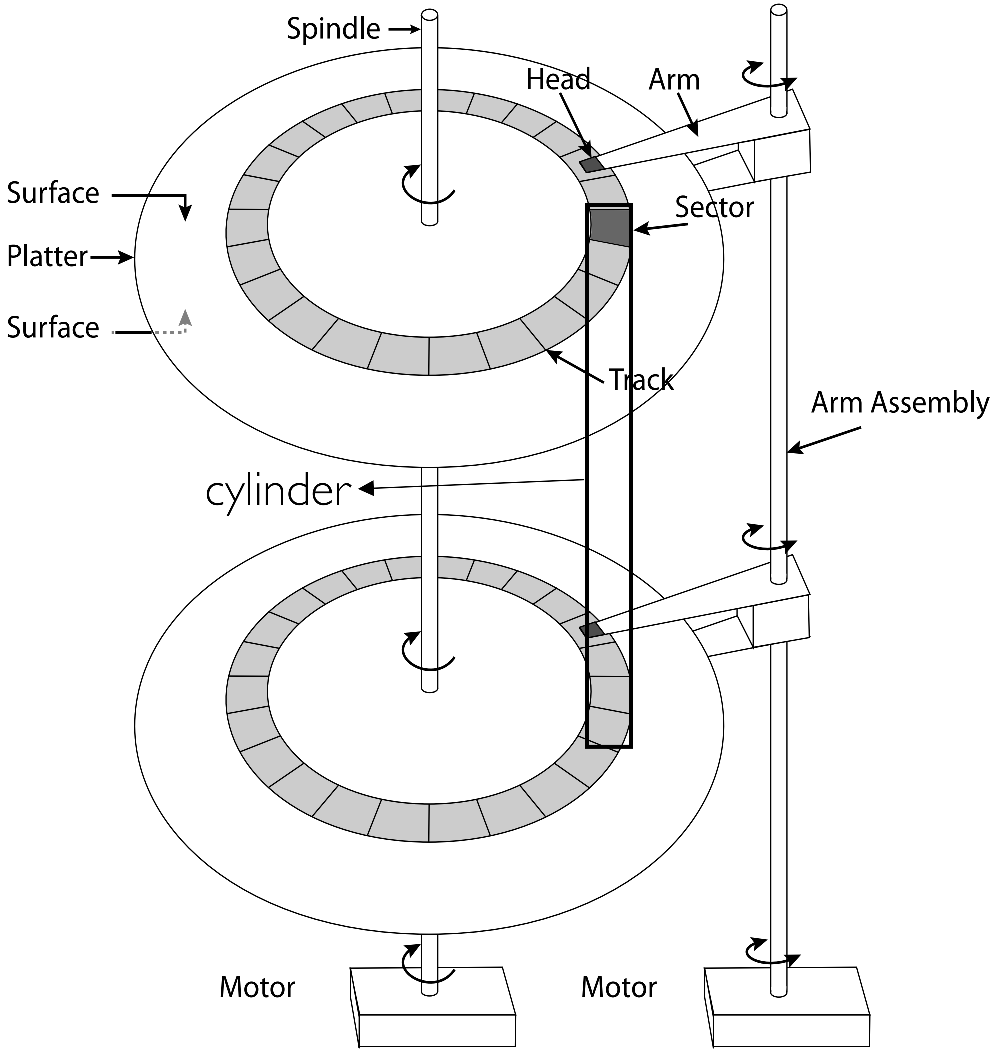
The Magnetic Disk

- Sector (扇区): the unit of transfer
- Track (磁道): ring of sectors
  - ~ 1um (10^{-6}m) wide
    - ❑ Resolution of human eye: 50um
    - ❑ Wavelength of light is ~0.5um
- Cylinder (柱面): stacked tracks
- Head (磁头): attached to movable arms to read data
  - 2 per each platter (磁片) for each surfaces

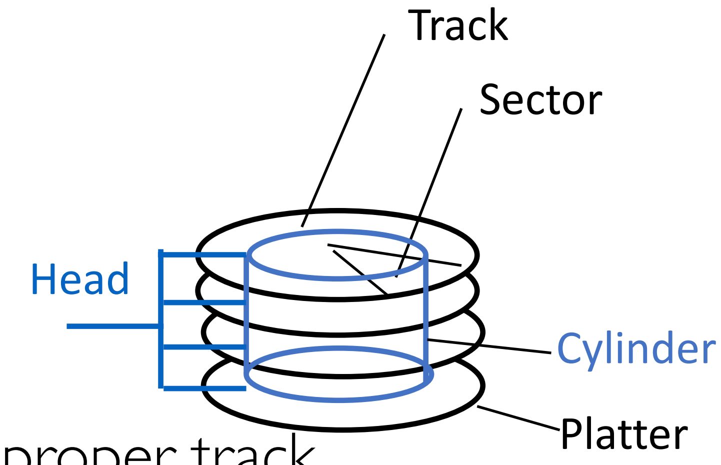
- Storage capacity =

(head #) * (cylinder #) * (sector #) * (sector size)

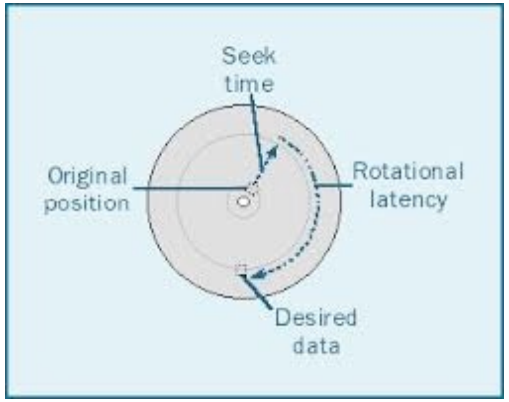
Often 512 bytes



The Magnetic Disk



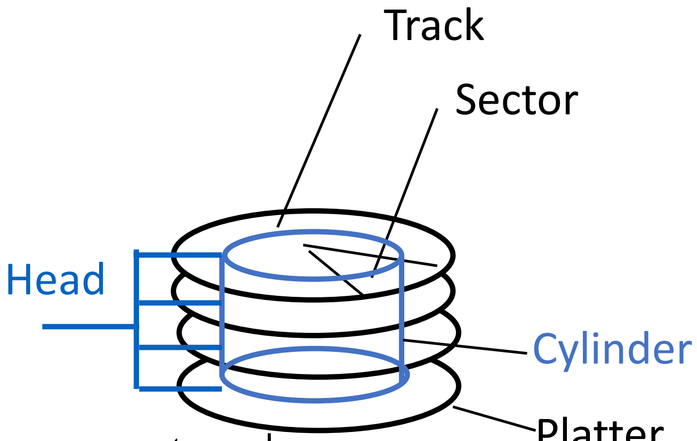
- Cylinders: all the tracks under the head at a given point on all surface
- Read/write data is a three-stage process:
  - Seek time (寻道时间): position the head/arm over the proper track
  - Rotational latency (延迟时间): wait for desired sector to rotate under r/w head
  - Transfer time (传输时间): transfer a block of bits (sector) under r/w head



Seek time = 4-8ms
One rotation = 1-2ms
(3600-7200 RPM)

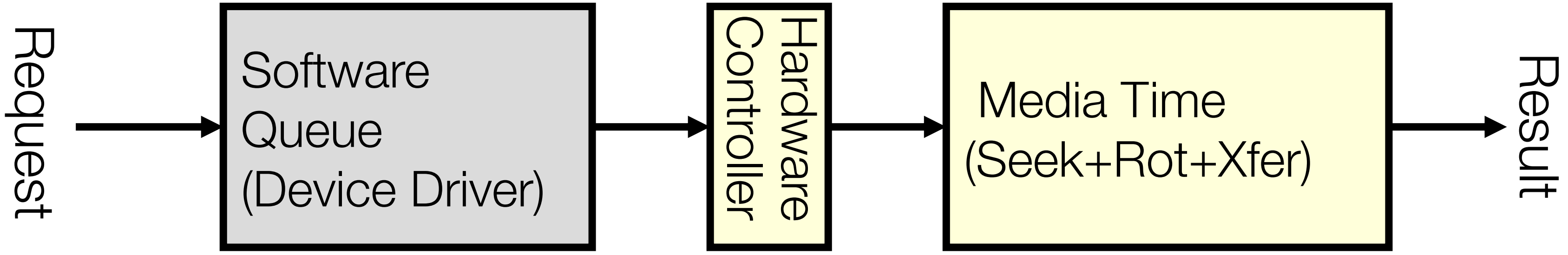
The Magnetic Disk

- Cylinders: all the tracks under the head at a given point on all surface



- Read/write data is a three-stage process:
  - Seek time (寻道时间): position the head/arm over the proper track
  - Rotational latency (延迟时间): wait for desired sector to rotate under r/w head
  - Transfer time (传输时间): transfer a block of bits (sector) under r/w head

Disk Latency = Queuing Time + Controller time +
Seek Time + Rotation Time + Transfer Time



Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/minute) / 7200(rev/min) ~= 8ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte ⇒
    1024 bytes/4×10^6 (bytes/s) = 256×10^{-6} sec ≅ .26 ms
- Read sector from random place on disk:

Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/minute) / 7200(rev/min) ~= 8ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte ⇒ 1024 bytes/4×10^{6} (bytes/s) = 256×10^{-6} sec ≅ .26 ms
- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.26ms) = 9.26ms
  - *Approx* 10ms to fetch/put data: **100 KByte/sec**
- Read sector from random place in same cylinder:

Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/minute) / 7200(rev/min) ~= 8ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte ⇒ 1024 bytes/4×10^6 (bytes/s) = 256×10^{-6} sec ≅ .26 ms
- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.26ms) = 9.26ms
  - *Approx* 10ms to fetch/put data: **100 KByte/sec**
- Read sector from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.26ms) = 4.26ms
  - *Approx* 5ms to fetch/put data: **200 KByte/sec**
- Read next sector on same track:

Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/minute) / 7200(rev/min) ~= 8ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte ⇒ 1024 bytes/4×10^6 (bytes/s) = 256×10^{-6} sec ≅ .26 ms
- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.26ms) = 9.26ms
  - *Approx* 10ms to fetch/put data: **100 KByte/sec**
- Read sector from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.26ms) = 4.26ms
  - *Approx* 5ms to fetch/put data: **200 KByte/sec**
- Read next sector on same track:
  - Transfer (0.26ms): **4 MByte/sec**

Key to using disk effectively (especially for file systems) is to *minimize seek and rotational delays*

(Lots of) Intelligence in the Controller

- Sectors contain sophisticated error correcting codes
  - Disk head magnet has a field wider than track
  - Hide corruptions due to neighboring track writes

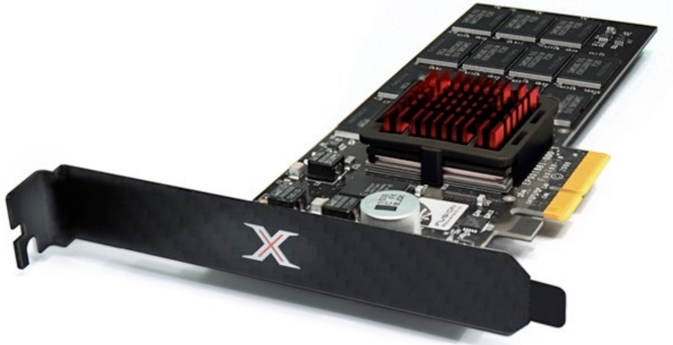
- Sector sparing
  - Remap bad sectors transparently to spare sectors on the same surface

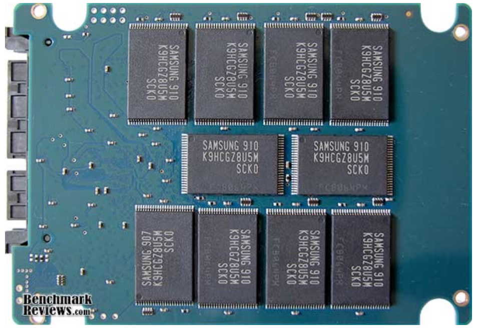
- Slip sparing
  - Remap all sectors (when there is a bad sector) to preserve sequential behavior

- Track skewing
  - Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

- …

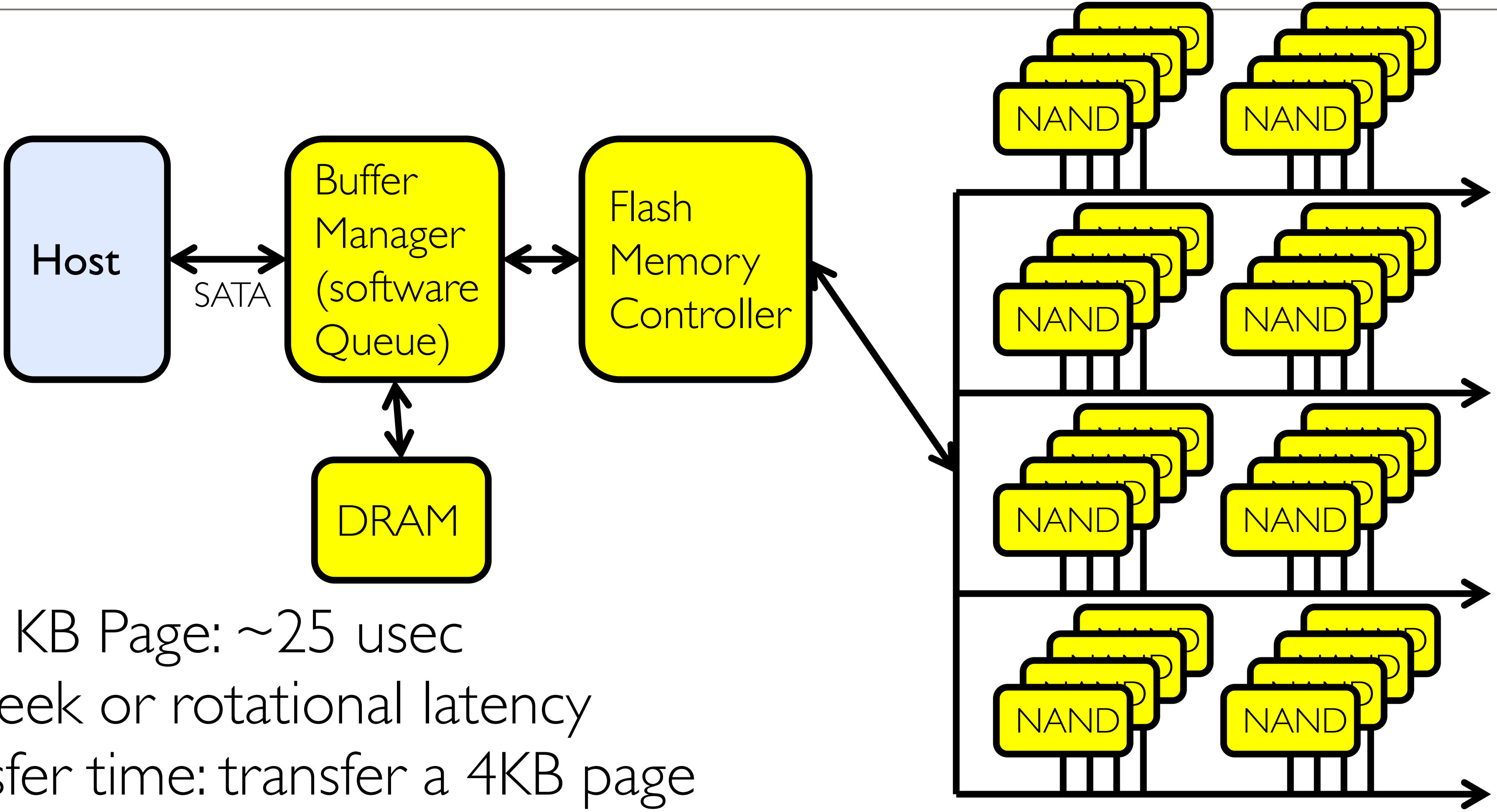
Solid State Disks (SSDs)





- 1995 – Replace magnetic media with non-volatile memory (battery backed DRAM)
- 2009 – Use NAND Multi-Level Cell (2 or 3-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 ”pages” per memory block
  - Trapped electrons distinguish between 1 and 0
- No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight
  - Limited “write cycles”
- Rapid advances in capacity and cost ever since!
- A 5-min video on SSD: https://www.bilibili.com/video/BV1644y157mB

SSD Architecture – Reads

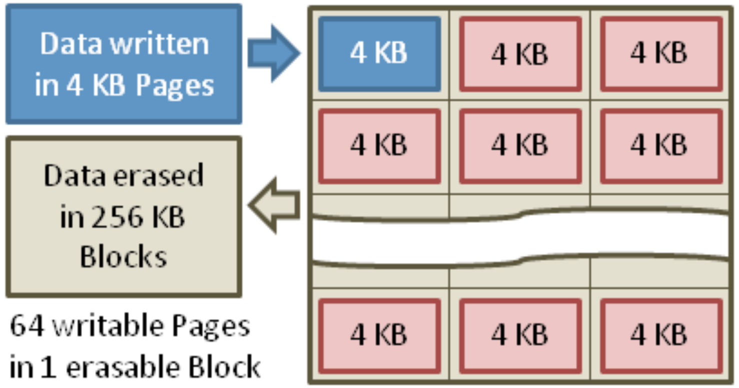


Read 4 KB Page: ~25 usec

- No seek or rotational latency
- Transfer time: transfer a 4KB page
  - ❑ SATA: 300-600MB/s => ~4 x10^3 b / 400 x 10^6 bps => 10 us
- Latency = Queuing Time + Controller Time + Xfer Time
- Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Writes

- Writing data is complex! (~200µs – 1.7ms)
  - Can only write empty pages in a block
  - Erasing a block takes ~1.5ms
  - Controller maintains pool of empty blocks by coalescing used pages (read, erase, write), also reserves some % of capacity
- Rule of thumb: writes 10x reads, erasure 10x writes


Data written in 4 KB Pages
Data erased in 256 KB Blocks
64 writable Pages in 1 erasable Block
4 KB
4 KB
4 KB
4 KB
4 KB
4 KB
4 KB
4 KB
4 KB

Typical NAND Flash Pages and Blocks

https://en.wikipedia.org/wiki/Solid-state_drive

Amusing calculation: is a full Kindle heavier than an empty one?

- Actually, "Yes", but not by much
- Flash works by trapping electrons:
  - So, erased state lower energy than written state
- Assuming that:
  - Kindle has 4GB flash
  - ½ of all bits in full Kindle are in high-energy state
  - High-energy state about 10^{-15} joules higher
  - Then: Full Kindle is 1 attogram (10^{-18}gram) heavier (Using $E = mc^2$)
- Of course, this is less than most sensitive scale can measure (it can measure 10^{-9} grams)
- Of course, this weight difference overwhelmed by battery discharge, weight from getting warm, ….
- According to John Kubiatowicz (New York Times, Oct 24, 2011)

SSD Summary

- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - ❑ Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - Small storage (0.1-0.5x disk), expensive (3-20x disk)
    - ❑ Hybrid alternative: combine small SSD with large HDD

SSD Summary

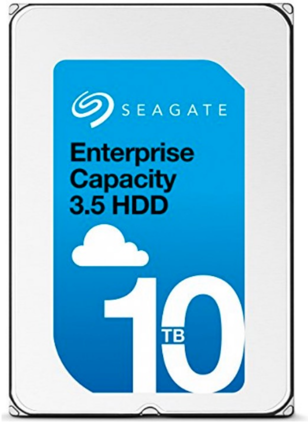
- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - ❑ Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - ~~Small storage (0.1-0.5x disk)~~, expensive (3-20x disk)
    - ❑ Hybrid alternative: combine small SSD with large HDD
  - Asymmetric block write performance: read pg/erase/write pg
    - ❑ Controller garbage collection (GC) algorithms have major effect on performance
  - Limited drive lifetime
    - ❑ 1-10K writes/page for MLC NAND
    - ❑ Avg failure rate is 6 years, life expectancy is 9–11 years
- These are changing rapidly!

No longer true!

Enterprise

10TB (2016)

- 7 platters, 14 heads
- 7200 RPMs
- 6 Gbps SATA /12Gbps SAS interface
- 220MB/s transfer rate, cache size: 256MB
- Helium filled: reduce friction and power usage
- Price: $500 ($0.05/GB)



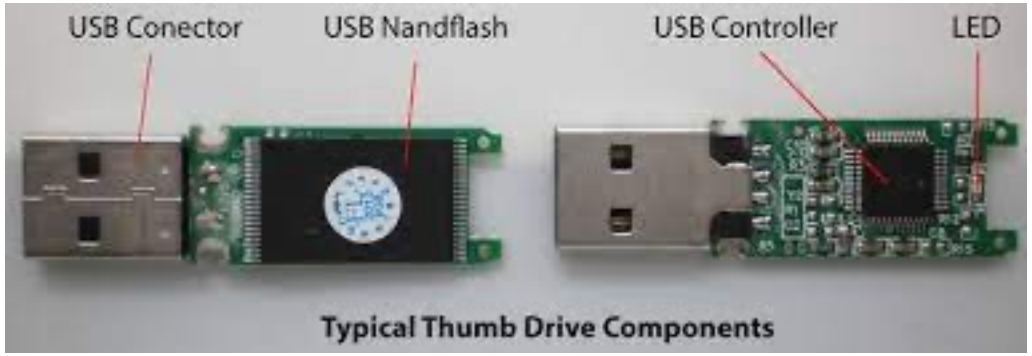
IBM Personal Computer/AT (1986)

- 30 MB hard disk
- 30-40ms seek time
- 0.7-1 MB/s (est.)
- Price: $500 ($17K/GB, 340,000x more expensive !!)

Largest SSDs

- 60TB (2016)
- Dual port: 16Gbs
- Seq reads: 1.5GB/s
- Seq writes: 1GB/s
- Random Read Ops (IOPS): 150K
- Price: ~ $20K ($0.33/GB)

USB Drive

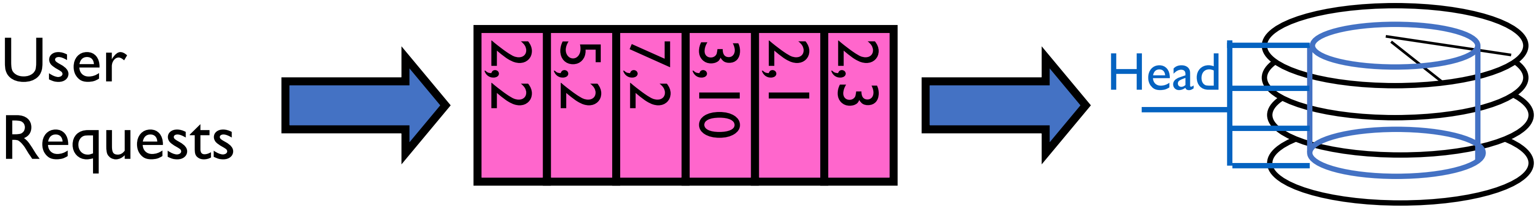


1GB~8GB, 2010

Up to 1TB, 2023

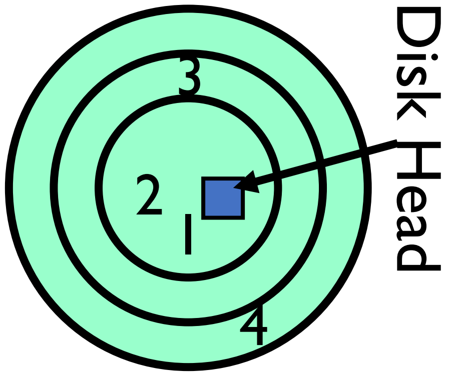
Disk Scheduling

- Disk can do only one request at a time; What order do you choose to do queued requests?
  - The scheduling can be done in OS, firmware, or both.



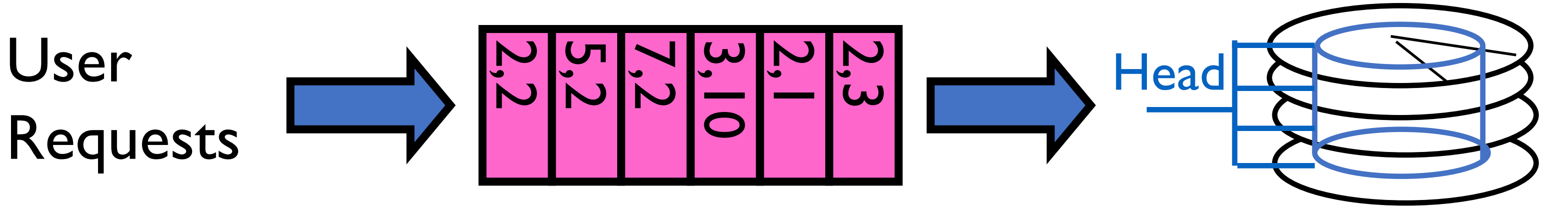
User Requests

- FIFO Order
  - Fair among requesters, but order of arrival may be to random spots on the disk ⇒ Very long seeks

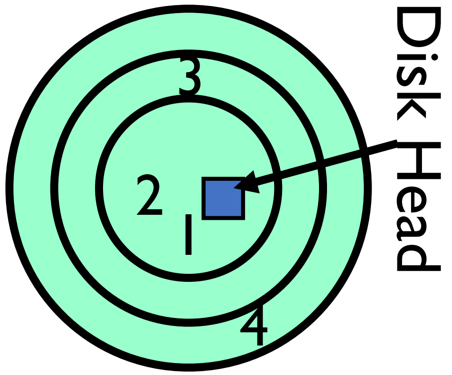


Disk Scheduling

- Disk can do only one request at a time; What order do you choose to do queued requests?
  - The scheduling can be done in OS, firmware, or both.

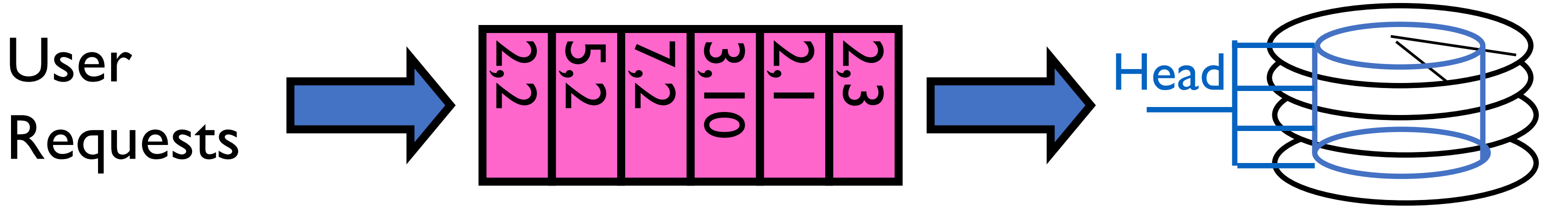


- SSTF: Shortest seek time first
  - Pick the request that's closest on the disk
  - Although called SSTF, today must include rotational delay in calculation, since rotation can be as long as seek
  - Con: SSTF good at reducing seeks, but may lead to starvation

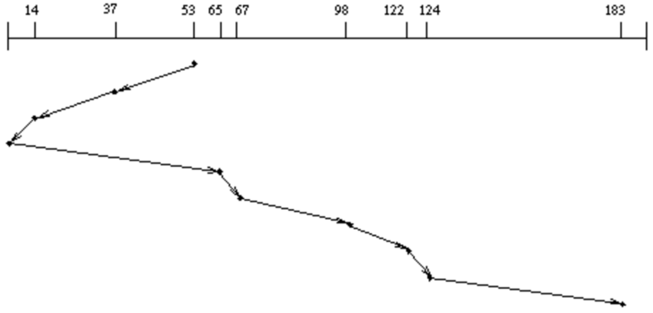


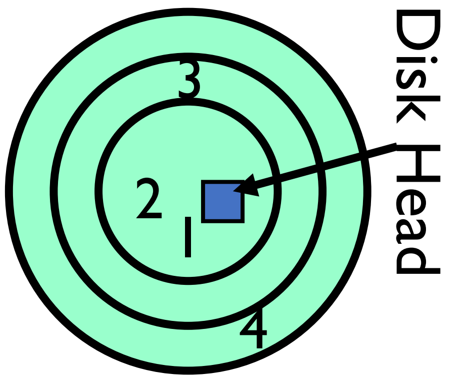
Disk Scheduling

- Disk can do only one request at a time; What order do you choose to do queued requests?
  - The scheduling can be done in OS, firmware, or both.



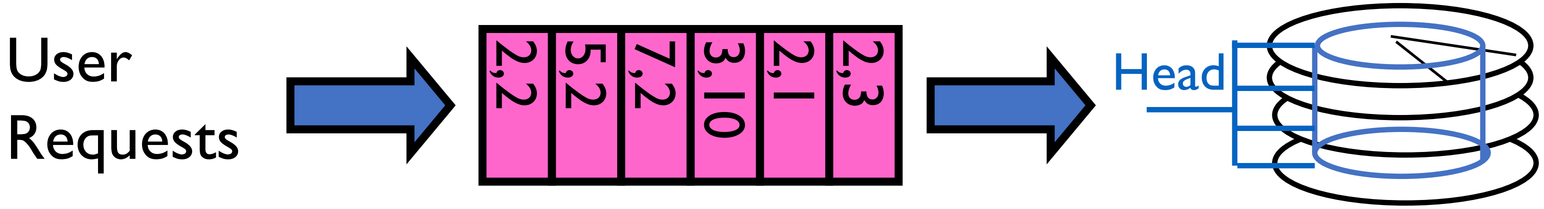
- SCAN: Implements an Elevator Algorithm (**电梯算法**): take the closest request in a fixed direction of travel (reversed at the end)
  - No starvation, but retains flavor of SSTF



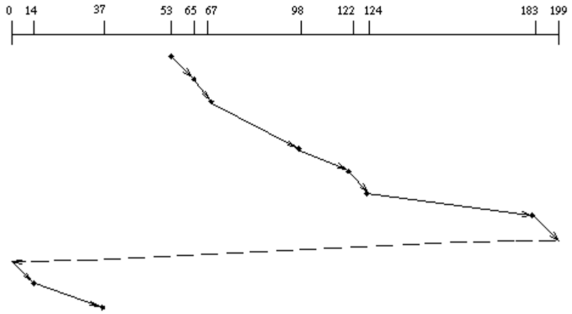


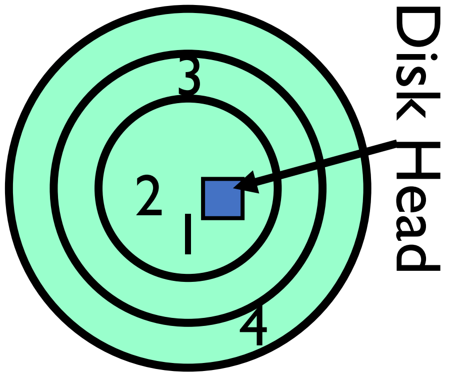
Disk Scheduling

- Disk can do only one request at a time; What order do you choose to do queued requests?
  - The scheduling can be done in OS, firmware, or both.



- C-SCAN: Circular-Scan: only goes in one direction
  - Skips any requests on the way back
  - Fairer than SCAN, not biased towards pages in middle





A Simple Read() Lifecycle

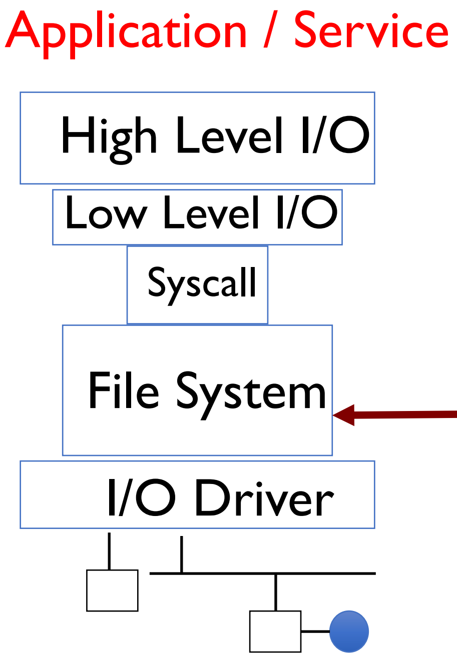
- A process issues a syscall read()
- OS moves the calling thread to a wait queue (state=WAITING)
- OS uses memory-mapped I/O to tell the disk to read the requested data and set up DMA so the disk can place the data in kernel's memory
- Disk reads the data and DMAs it into main memory
- Disk triggers an interrupt
- OS's interrupt handler copies the data from the kernel's buffer into the process's address space
- OS moves the thread to the ready list
- The thread is scheduled on CPU, and returns from the read()

Goals for Today

- Storage Devices
- File System Abstraction

I/O & Storage Layers

Operations, Entities and Interface



Application / Service

High Level I/O

Low Level I/O

Syscall

File System

I/O Driver

streams

handles

registers

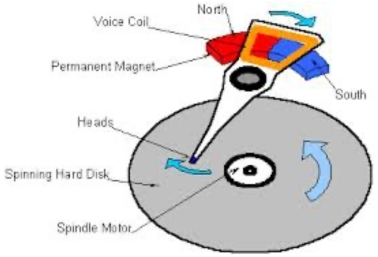
`file_open, file_read, … on struct file * & void *`

descriptors

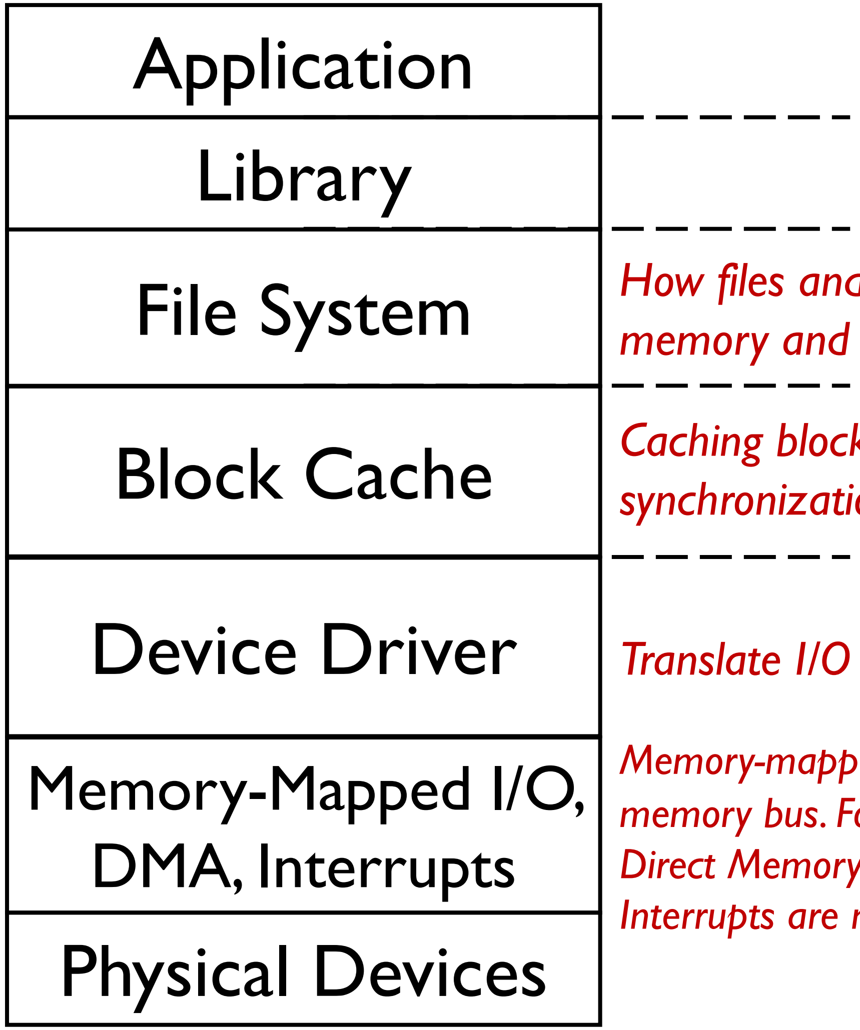
we are here …

Commands and Data Transfers

Disks, Flash, Controllers, DMA



Layered abstractions of I/O and storage



Application

Library

File System

Block Cache

Device Driver

Memory-Mapped I/O, DMA, Interrupts

Physical Devices

stdio: fopen(), fclose(), fread(), fwrite()

syscall: open(), close(), read(), write()

How files and directories are organized in memory and disk

Data block ops between storage and memory

Caching blocks in memory; write buffering, synchronization

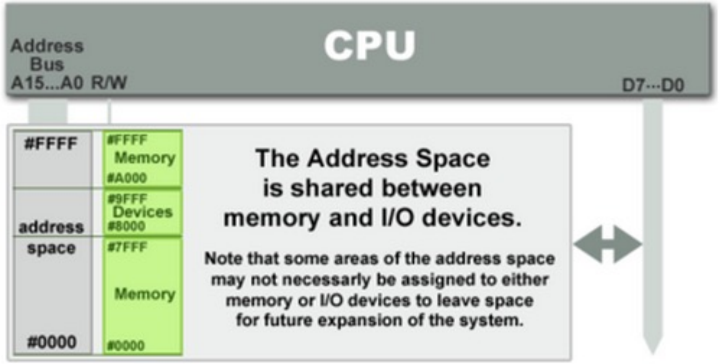
Block device interface: a standard interface for different I/O devices to R/W in fixed-sized blocks (e.g., 512 bytes).

Translate I/O abstractionsinto device-specific I/O operations

Memory-mapped I/O: maps each device's control registers to a range of physical addresses on the memory bus. For example, the OS knows last key pressed by keyboard in a physical address.
Direct Memory Access: copy a block of data between storage and memory.
Interrupts are needed so OS knows when I/O device completes its request (otherwise use polling).

Memory-mapped I/O vs. Port-mapped I/O

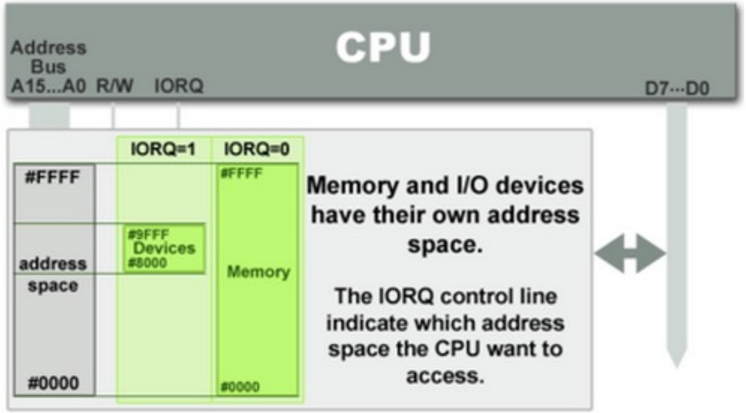
- Two complementary ways for CPU to access I/O devices
  - I/O devices have their own registers (or memory)
- Memory-mapped I/O (MMIO): let memory and devices share the physical address space.
  - Most widely adopted
  - Shared address bus



The Address Space is shared between memory and I/O devices.

Note that some areas of the address space may not necessarly be assigned to either memory or I/O devices to leave space for future expansion of the system.

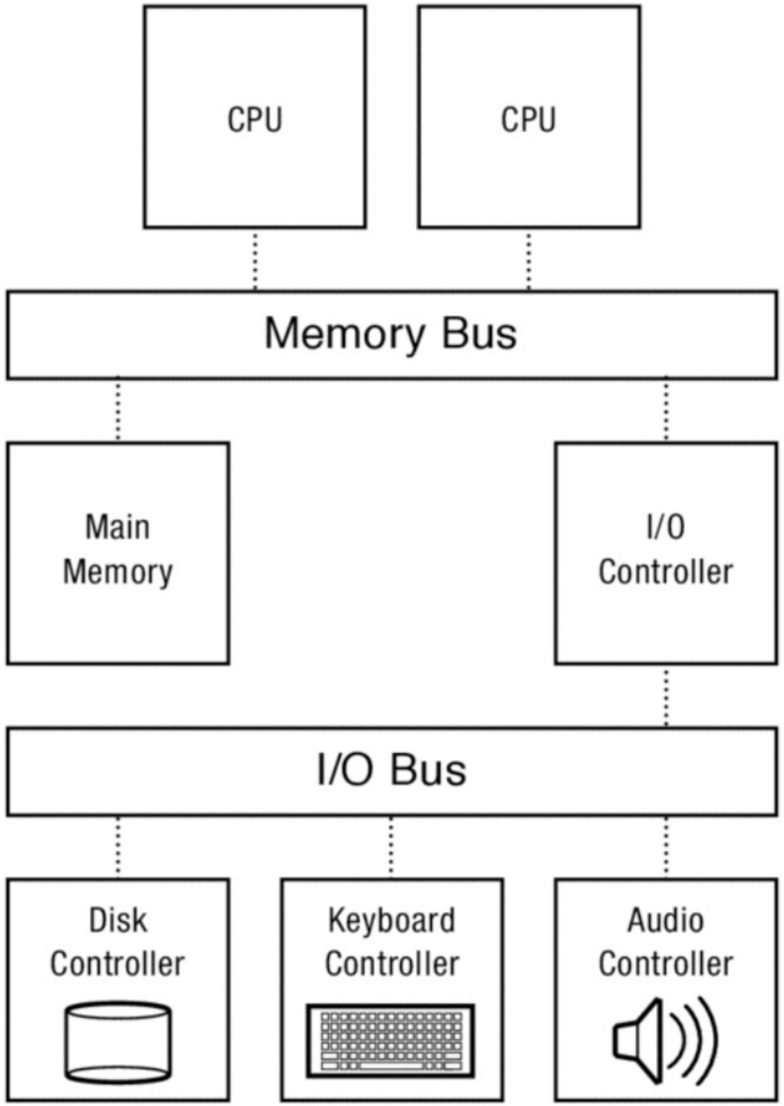
- Port-mapped I/O (PMIO), or isolated I/O: use specialized instructions to R/W I/O devices
  - In Intel: outb, outw, etc.

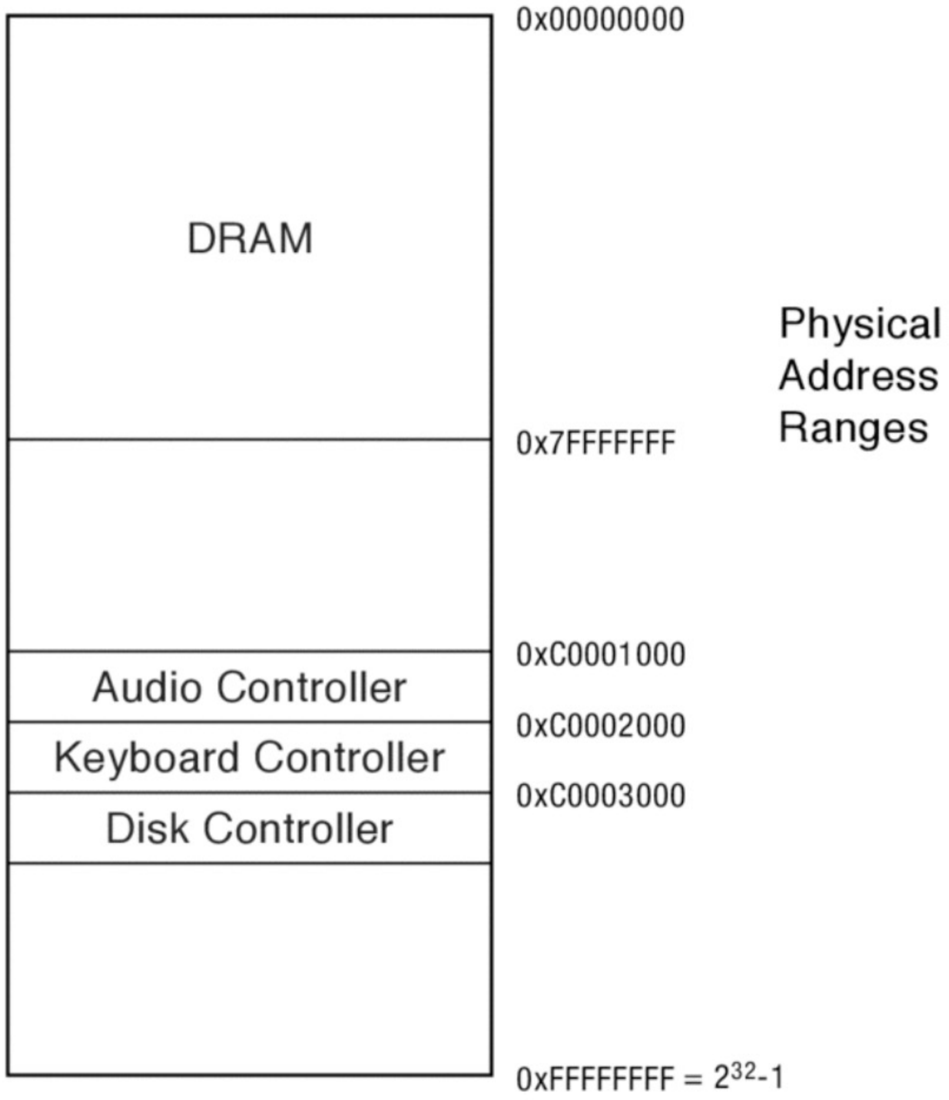


Memory and I/O devices have their own address space.

The IORQ control line indicate which address space the CPU want to access.

Storage Stack





Recall: C Low level I/O

- File Descriptors – as OS object representing the state of a file
  - User has a "handle" on the descriptor

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int open (const char *filename, int flags [, mode_t mode])
int create (const char *filename, mode_t mode)
int close (int filedes)
```

Bit vector of:
- Access modes (Rd, Wr, …)
- Open Flags (Create, …)
- Operating modes (Appends, …)

Bit vector of Permission Bits:
- User|Group|Other X R|W|X

http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html

Recall: C Low level I/O

- File Descriptors – as OS object representing the state of a file
  - User has a "handle" on the descriptor

```
ssize_t read (int filedes, void *buffer, size_t maxsize)
 - returns bytes read, 0 => EOF, -1 => error
ssize_t write (int filedes, const void *buffer, size_t size)
 - returns bytes written
off_t lseek (int filedes, off_t offset, int whence)
 - set the file offset
   * if whence == SEEK_SET: set file offset to "offset"
   * if whence == SEEK_CRT: set file offset to crt location + "offset"
   * if whence == SEEK_END: set file offset to file size + "offset"
int fsync (int fildes)
 – wait for i/o of filedes to finish and commit to disk
void sync (void) – wait for ALL to finish and commit to disk
```

- When write returns, data is on its way to disk and can be read, but it may not actually be permanent!

Building a File System

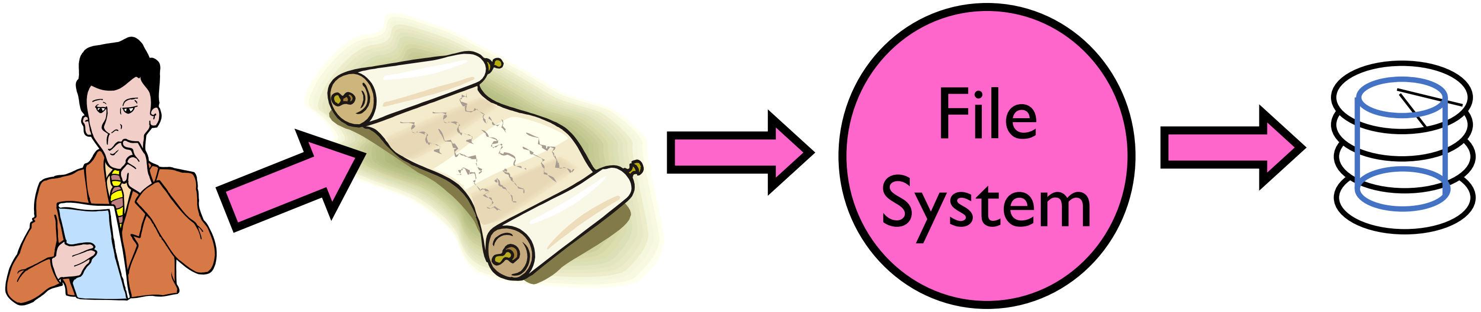
- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.

- File System Components
  - **Naming**: Interface to find files by name, not by blocks
  - **Disk Management**: collecting disk blocks into files
  - **Protection**: Layers to keep data secure
  - **Reliability/Durability**: Keeping of files durable despite crashes, media failures, attacks, etc.

User vs. System View of a File

- User's view:
  - Durable Data Structures
- System's view (system call interface):
  - Collection of Bytes (UNIX)
  - Doesn't matter to system what kind of data structures you want to store on disk!
- System's view (inside OS):
  - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
  - Block size ≥ sector size; in UNIX, block size is 4KB

Translating from User to System View



- What happens if user says: give me bytes 2—12?
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about: write bytes 2—12?
  - Fetch block
  - Modify portion
  - Write out Block
- Everything inside File System is in whole size blocks
  - For example, **`getc()`**, **`putc()`** ⇒ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, file is a collection of blocks

Disk Management Policies (1/2)

- Basic entities on a disk:
  - File: user-visible group of blocks arranged sequentially in logical space
  - Directory: user-visible index mapping names to files

- Access disk as linear array of sectors. Two Options:
  - Identify sectors as vectors [cylinder, surface, sector], sort in cylinder-major order
    - ❑ Used in BIOS, but not in OSes anymore
  - Logical Block Addressing (LBA, 逻辑块寻址): Every sector has integer address from zero up to max number of sectors
  - Controller translates from address ⇒ physical position
    - ❑First case: OS/BIOS must deal with bad sectors
    - ❑Second case: hardware shields OS from structure of disk

Disk Management Policies (2/2)

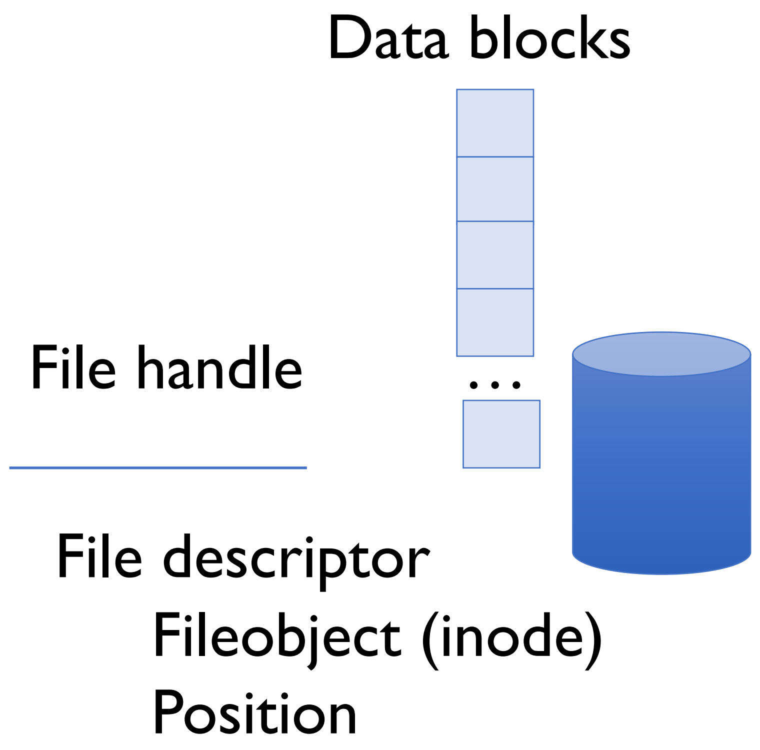
- Need way to track free disk blocks
  - Link free blocks together ⇒ too slow today
  - Use bitmap to represent free space on disk

- Need way to structure files: File Header
  - Track which blocks belong at which offsets within the logical file structure
  - Optimize placement of files' disk blocks to match access and usage patterns

File

- Named permanent storage

- Contains
  - Data
    - ❑Blocks on disk somewhere
  - Metadata (Attributes)
    - ❑Owner, size, last opened, …
    - ❑Access rights
      - R, W, X
      - Owner, Group, Other (in Unix systems)
      - Access control list in Windows system



Data blocks

…

File handle

File descriptor

Fileobject (inode)

Position

Directory

- Basically a hierarchical structure

- Each directory entry is a collection of
  - Files
  - Directories
    - ❑A link to another entries

- Each has a name and attributes
  - Files have data

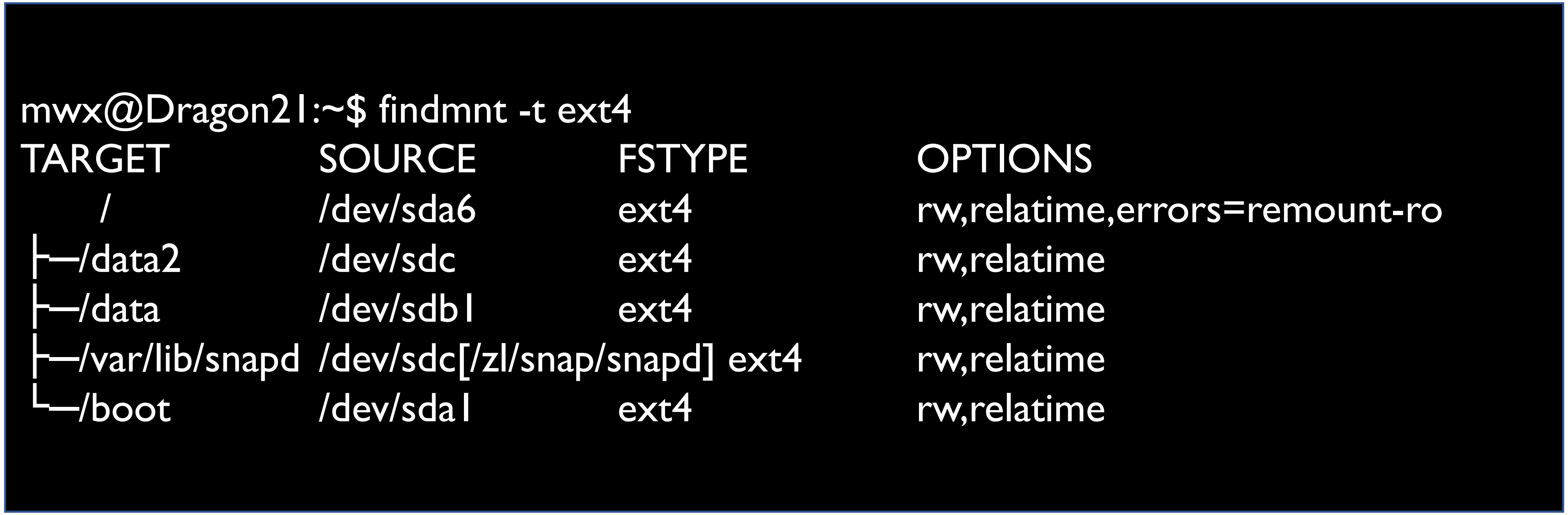
- Links (hard links) make it a DAG, not just a tree
  - Softlinks (aliases) are another name for an entry

Directory

- Conventions of directory
  - Root directory (根目录):“/”
  - Home directory (主目录):“~/cur_dir/file.txt”
  - Absolute path (绝对路径):“/home/mwx/cur_dir/file.txt”
  - Relative path (相对路径):“file.txt”
- Volume (卷): a collection of physical storage resources that form a logical storage device. Could be a part of or many physical devices.
- Mount (挂载): an operation that creates a mapping from some path in the existing file system to the root directory of the mounted volume’s file system

```
mount –t type device dir
```

Directory



```
mwx@Dragon21:~$ findmnt -t ext4
TARGET            SOURCE             FSTYPE         OPTIONS
     /            /dev/sda6          ext4           rw,relatime,errors=remount-ro
├─/data2          /dev/sdc           ext4           rw,relatime
├─/data           /dev/sdb1          ext4           rw,relatime
├─/var/lib/snapd  /dev/sdc[/zl/snap/snapd] ext4     rw,relatime
└─/boot           /dev/sda1          ext4           rw,relatime
```

Designing a File System …

- What factors are critical to the design choices?
- Durable data store => it's all on disk
- (Hard) Disks Performance !!!
  - Maximize sequential access, minimize seeks
- Open before Read/Write
  - Can perform protection checks and look up where the actual file resource are, in advance
- Size is determined as they are used !!!
  - Can write (or read zeros) to expand the file
  - Start small and grow, need to make room
- Organized into directories
  - What data structure (on disk) for that?
- Need to allocate / free blocks
  - Such that access remains efficient

Reminder

- Easy_lab 3 is available
- Don’t forget the first homework (LLM-powered command line helper)