Week 3

DATS 6450 – FOUNDATIONS OF COMPUTER SCIENCE

Class Overview

- Review and Answer Questions on Computer Architecture
- Overview of Operating Systems and File Systems
- Introduce the Shell
  - Why learn the shell as a Data Scientist?
  - How can it optimize my workflow?
  - What capabilities does the Shell provide that other interfaces/frameworks don't?

Review of Computer Architecture

Computer Architecture

Intro to Computer Architecture

- When it comes to computer architecture, the purpose (for this course) is to show you how the computer works from a high level

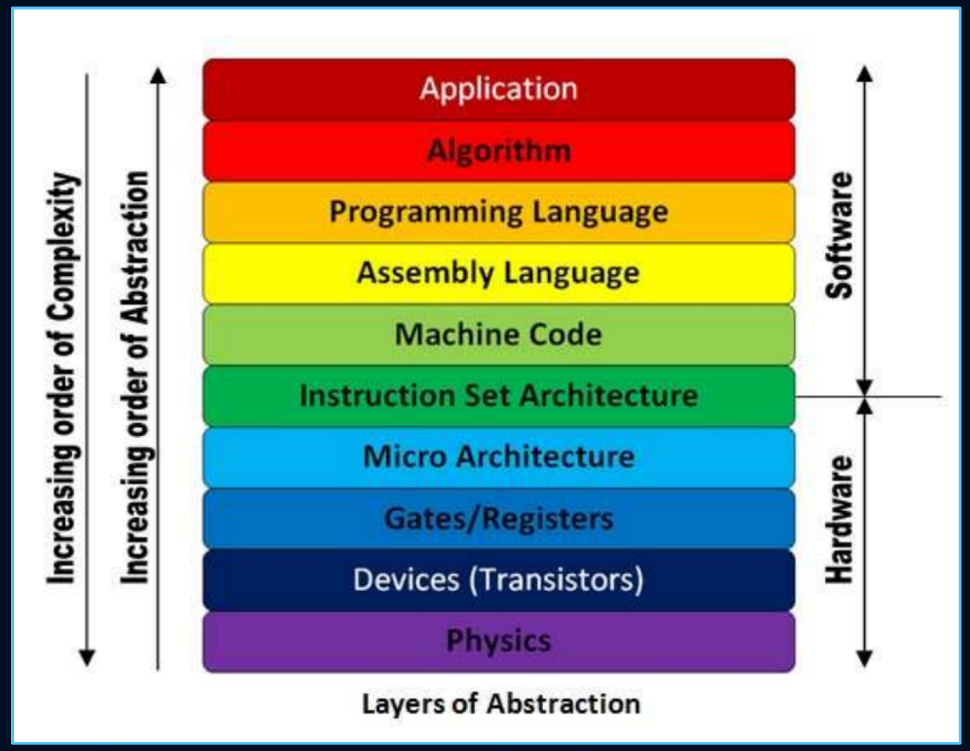
- The hope is that by knowing more about the design of the underlying system, you can be more effective as a programmer

- This should (in theory) enable you to:
  - Write programs that are more reliable and efficient
  - Troubleshoot your code and debug errors
  - Understand what (physically) is happening when you log-in to a computer remotely and navigate its internal filesystems

Intro to Computer Architecture

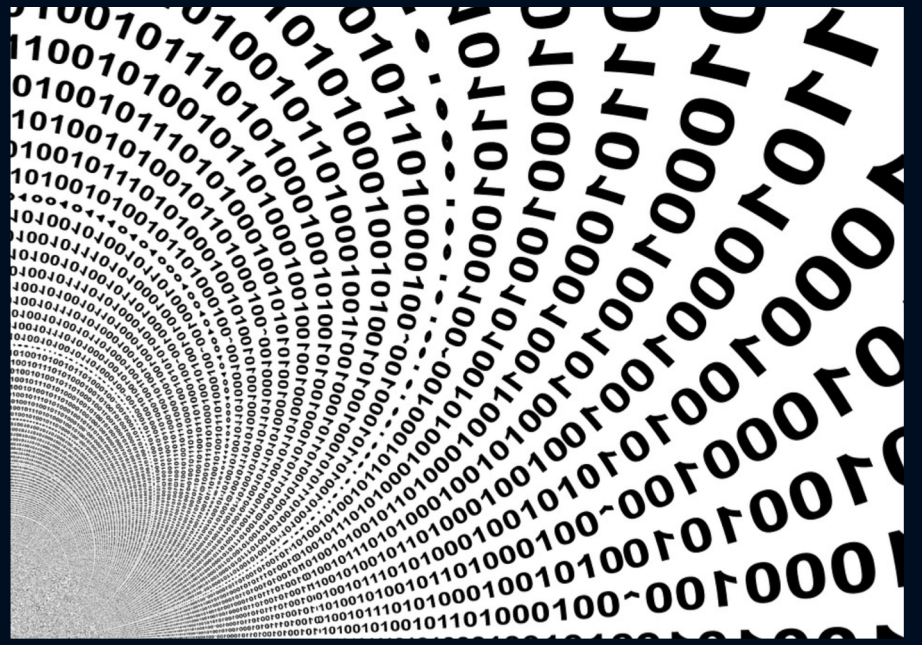
- When programming a computer, we interact with abstractions of abstractions of an abstractions

- At the end of the day it's all 1's an 0's



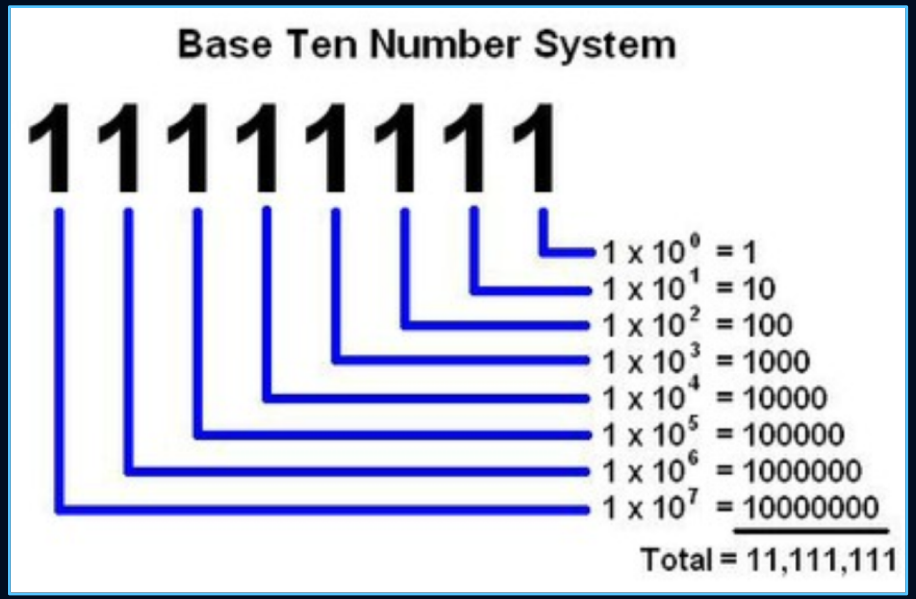
Binary

- "There are 10 kinds of people in the world, those than understand binary, and those who don't"
- Whatever is stored on a computer, has to ultimately be represented as a finite collection of bits
- This is true whether it's reals, ints, chars, strings, programs, images, videos, etc.



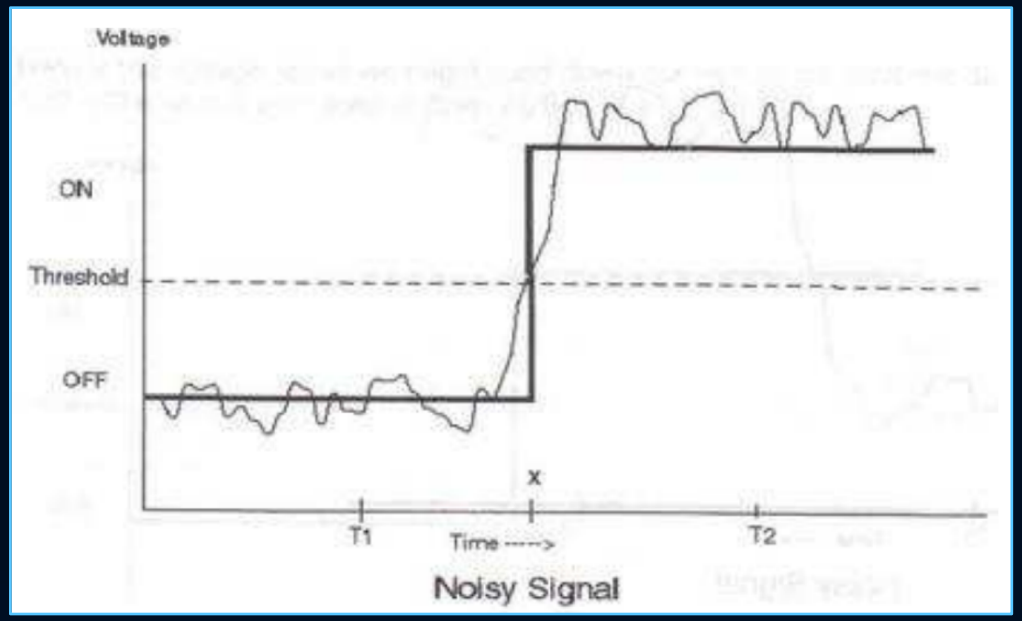
Why Not Base 10 (Decimal) System?

- Hard to store in memory
- Hard to transmit (need high precision to encode 10 different signals on a single wire)
- Messy to implement addition/multiplication/etc.



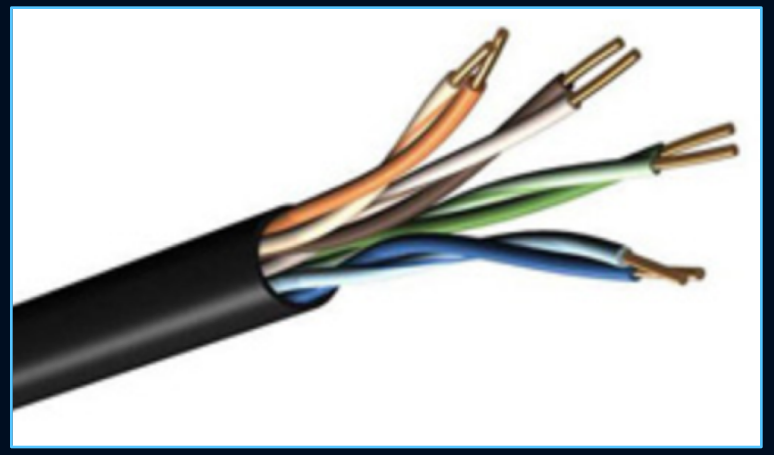
Binary to the rescue

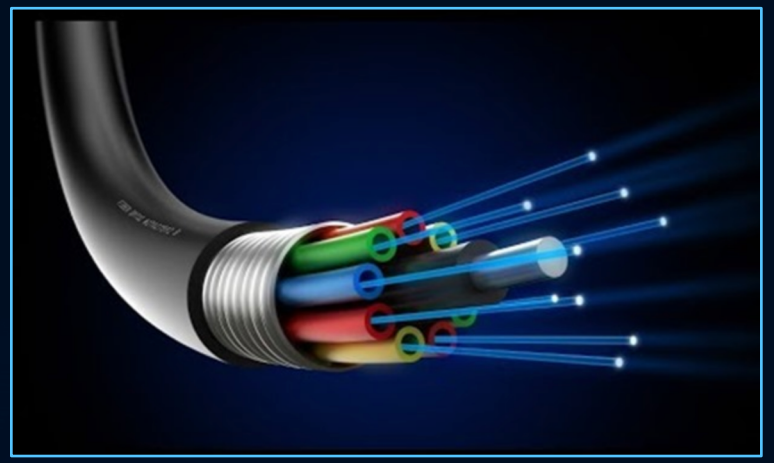
- Easy to store bits with bistable elements
- Easy to transmit signals (streams) on noisy and inaccurate wires



Binary Data

- Data streams are typically voltage changes (on copper wires)
  - Electrons flowing through a conductor
- Recently, with the advent of fiber-optics, data streams can be represented as light pulses (on/off)
  - Photons travelling through glass





Binary to the rescue

- So computers use binary numbers, and therefore use **binary digits** in place of decimal digits.

- The word **bit** is a shortening of the words "Binary digIT." Whereas decimal digits have 10 possible values ranging from 0 to 9, bits have only two possible values: 0 and 1.

- Therefore, a binary number is composed of only 0s and 1s, like this: 1011.

- How do you figure out what the value of the binary number 1011 is?

$$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) =$$
$$8 + 0 + 2 + 1 =$$
$$11$$

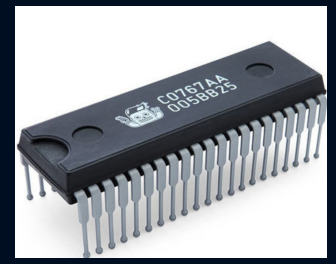
Bits/Bytes/…

Unit	Value	Size
bit (b)	0 or 1	1/8 of a byte
byte (B)	8 bits	1 byte
kilobyte (KB)	1000^1 bytes	1,000 bytes
megabyte (MB)	1000^2 bytes	1,000,000 bytes
gigabyte (GB)	1000^3 bytes	1,000,000,000 bytes
terabyte (TB)	1000^4 bytes	1,000,000,000,000 bytes
petabyte (PB)	1000^5 bytes	1,000,000,000,000,000 bytes
exabyte (EB)	1000^6 bytes	1,000,000,000,000,000,000 bytes
zettabyte (ZB)	1000^7 bytes	1,000,000,000,000,000,000,000 bytes
yottabyte (YB)	1000^8 bytes	1,000,000,000,000,000,000,000,000 bytes

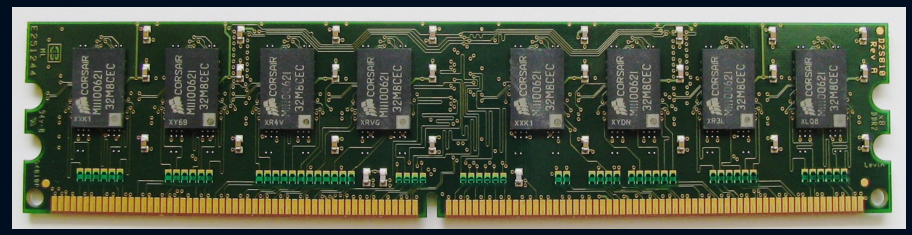
Internet Speed

- www.Speedtest.net

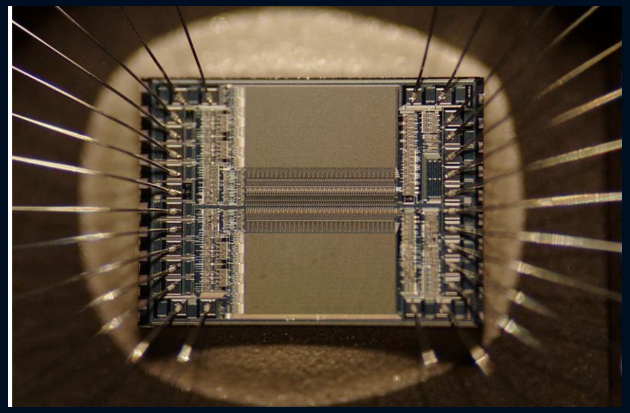
Computer Hardware



- Basic element
  - Solid-state transistor (i.e., electrical switch)
  - Building block of integrated circuits (ICs)



- Why IC's?
  - High Performance, High Reliability, Low Cost, Low Power
  - Easily mass-produced



- Modern IC's
  - Use tiny components that can be etched onto the surface of silicon chips
  - Easily mass-produced

- Moore's Law

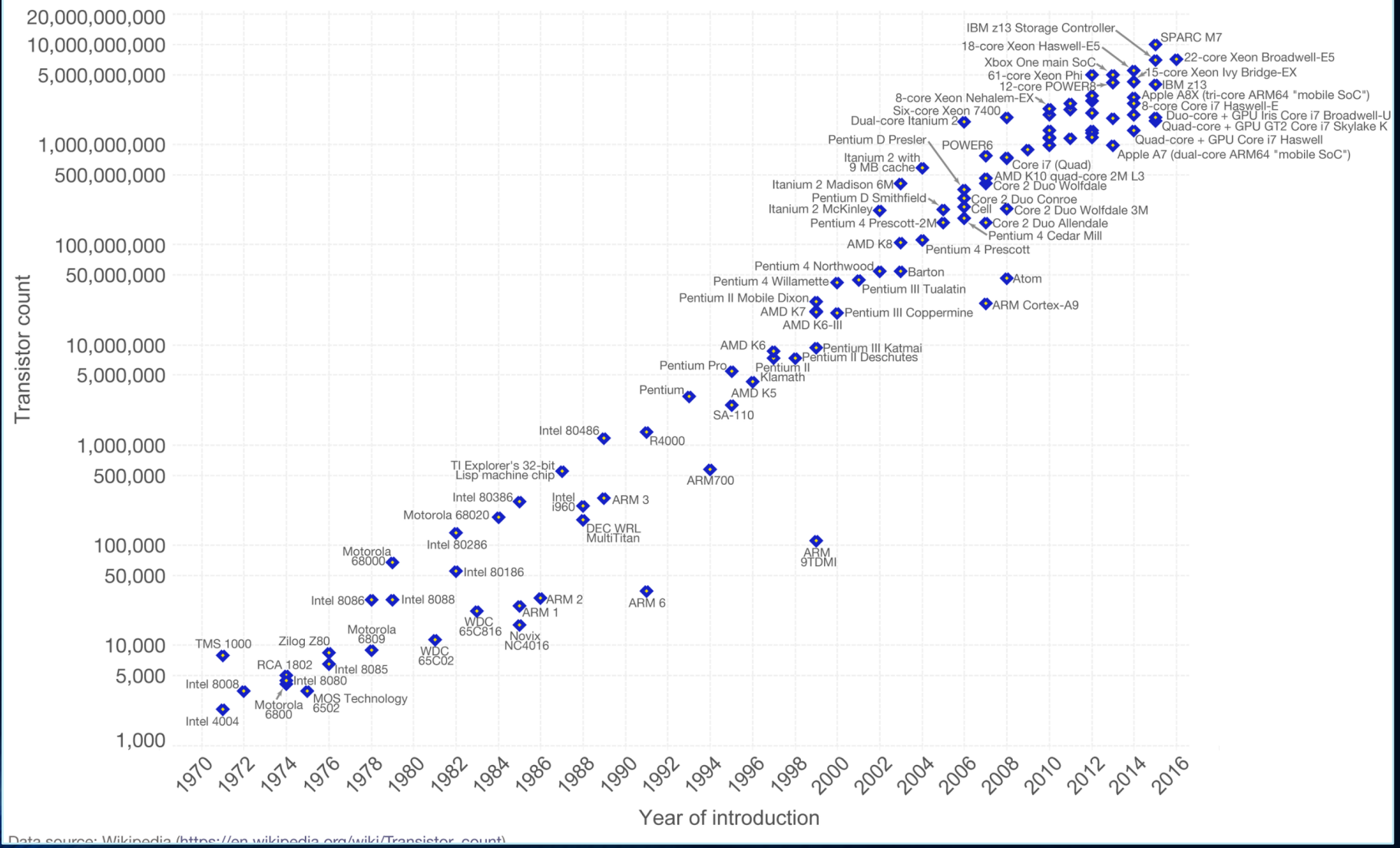
Moore’s Law

- Transistors get 2x smaller about every 2 years
  - sometimes listed as about 18 months
- Can fit twice as many transistors per chip
- Due to better chip etching technology
  (But a cutting edge chip factory costs more than 1 billion dollars)
- Moore's law (Gordon Moore, Intel co-founder) states that the density of transistors on a chip doubles about every 2 years or so
- It is not a scientific law, just a broad prediction that seems to keep working.

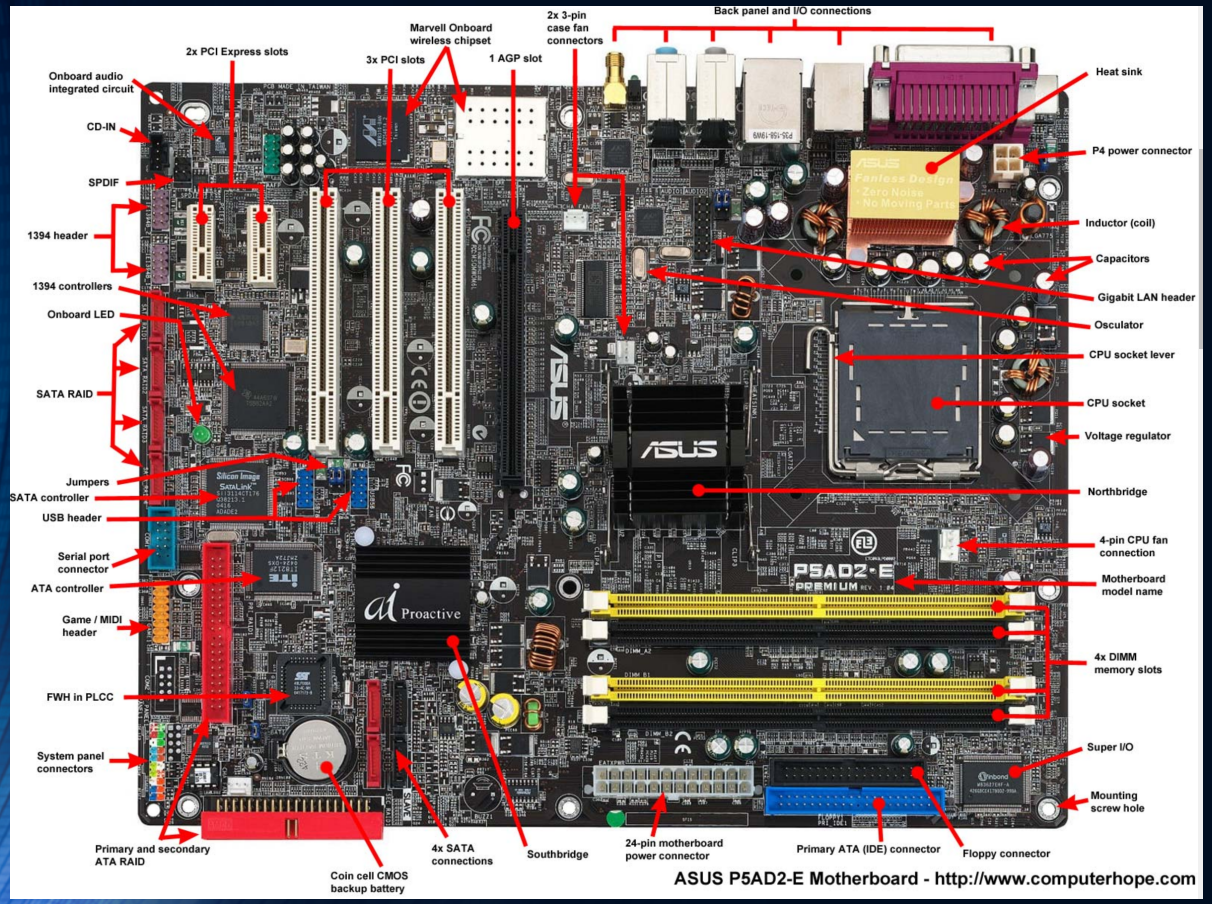
Moore’s Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

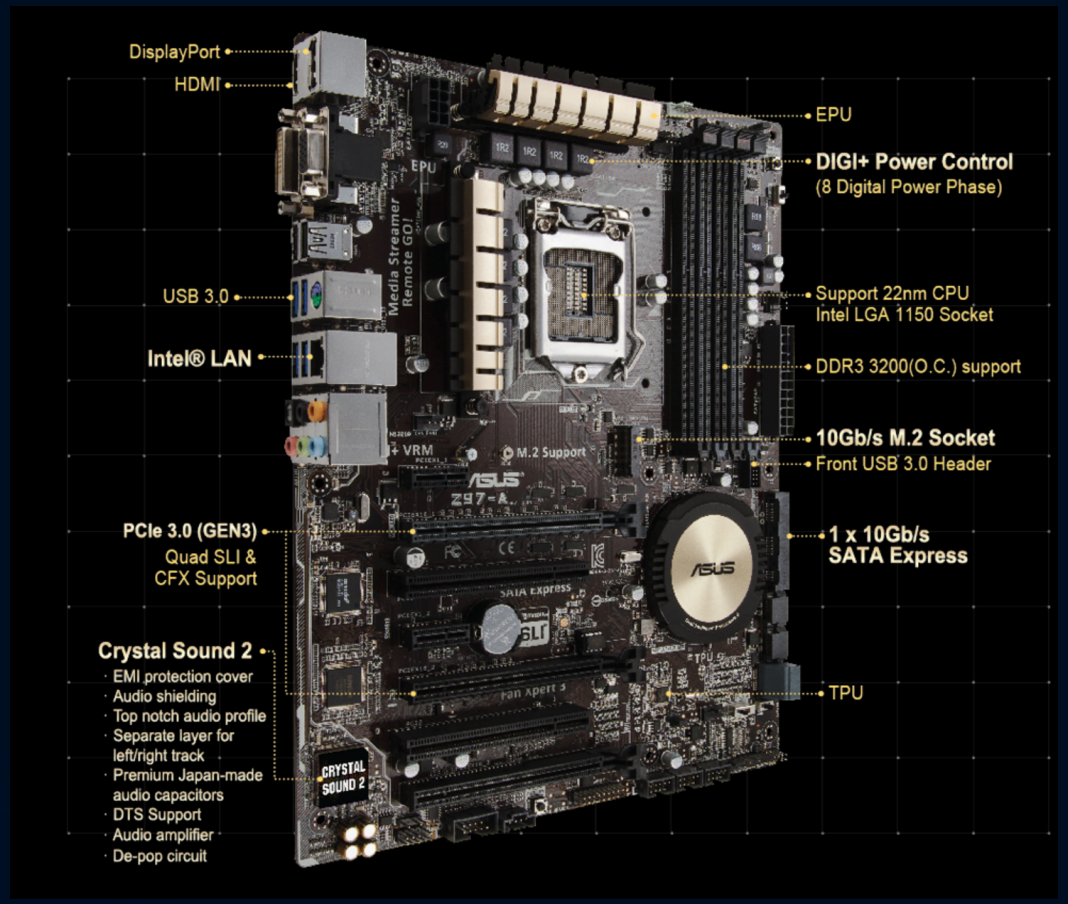
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)



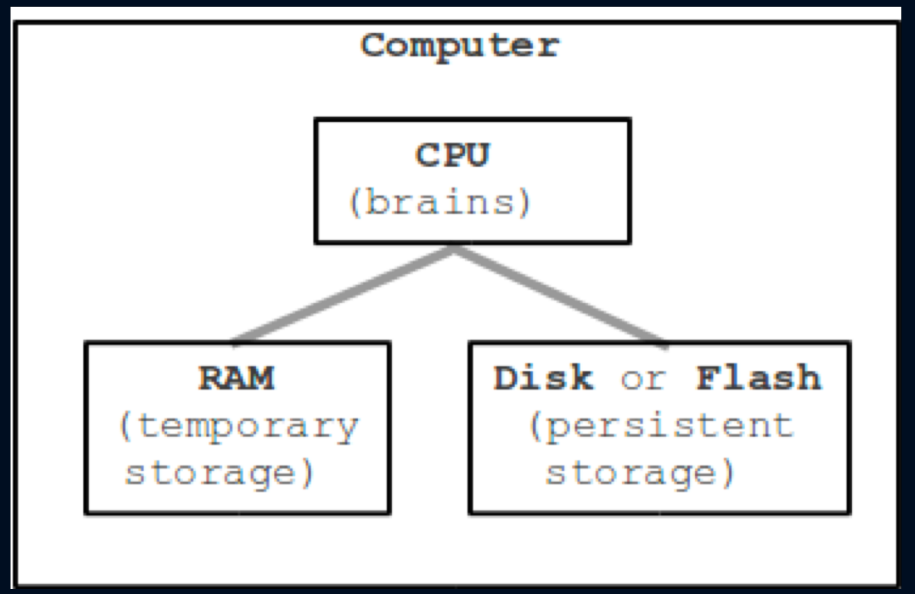
ASUS P5AD2-E Motherboard - http://www.computerhope.com



CPU/RAM

CPU - Central Processing Unit

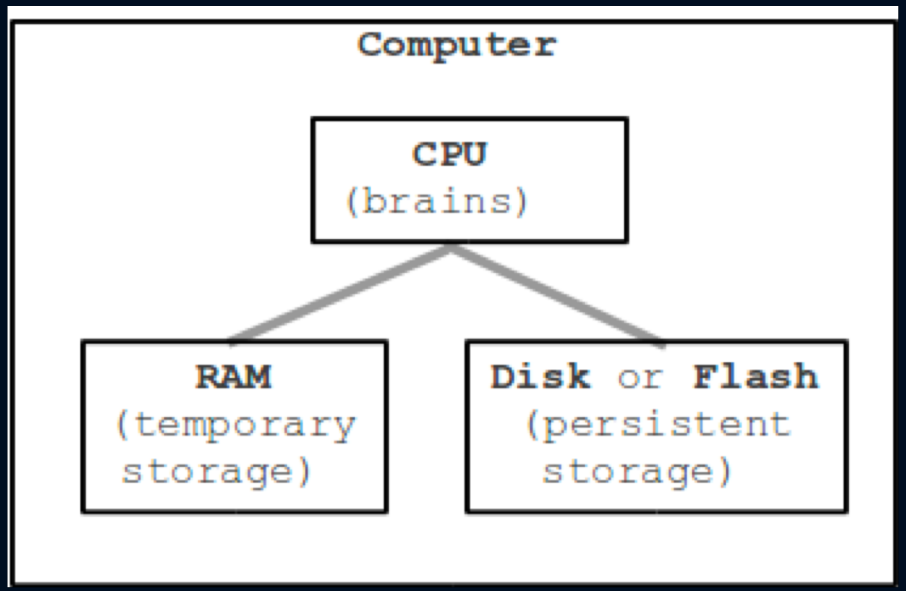
- Acts like a brain: follows the instructions in the code
- "general processing" - images, networking, math .. all on the CPU
- Performs computations, e.g. add two numbers
- vs. RAM and persistent storage which just store data
- "gigahertz" = 1 billion operations per second



RAM

RAM – Random Access Memory

- Acts like a whiteboard
- Temporary, working storage bytes
- RAM stores both code and data (temporarily)
- e.g. open an image in Photoshop
  - image data loaded into the bytes of RAM
- "persistent"
- - RAM is not persistent. State is gone when power turned off
  - e.g. You're working on a doc, then power goes out and you lose your work (vs. "Save")



BIOS

- BIOS (Basic Input/Output System)
  - ROM chip on motherboards that allows you to access and setup your computer system at the most basic level

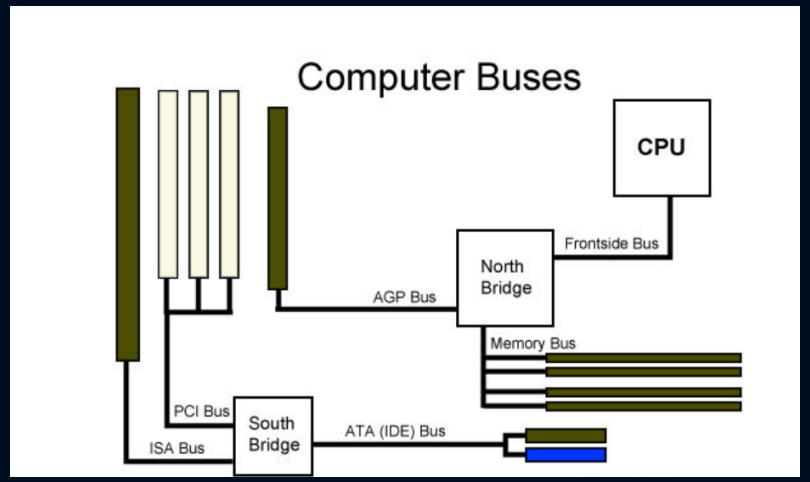
- The four main functions of a PC BIOS
  - POST - Test the computer hardware and make sure no errors exist before loading the operating system.
  - Bootstrap Loader - Locate the OS. If a capable operating system is located, the BIOS will pass control to it.
  - BIOS drivers - Low-level drivers that give the computer basic operational control over your computer's hardware.
  - BIOS or CMOS Setup - Configuration program that allows you to configure hardware settings including system settings such as computer passwords, time, and date.

BUS

- When referring to a computer, the **bus** also known as the **address bus**, **data bus**, or **local bus**, is a data connection between two or more devices connected to the computer.

- The bus contains multiple wires (signal lines) that contain addressing information that describes the memory location of where the data is being sent or where it is being retrieved.

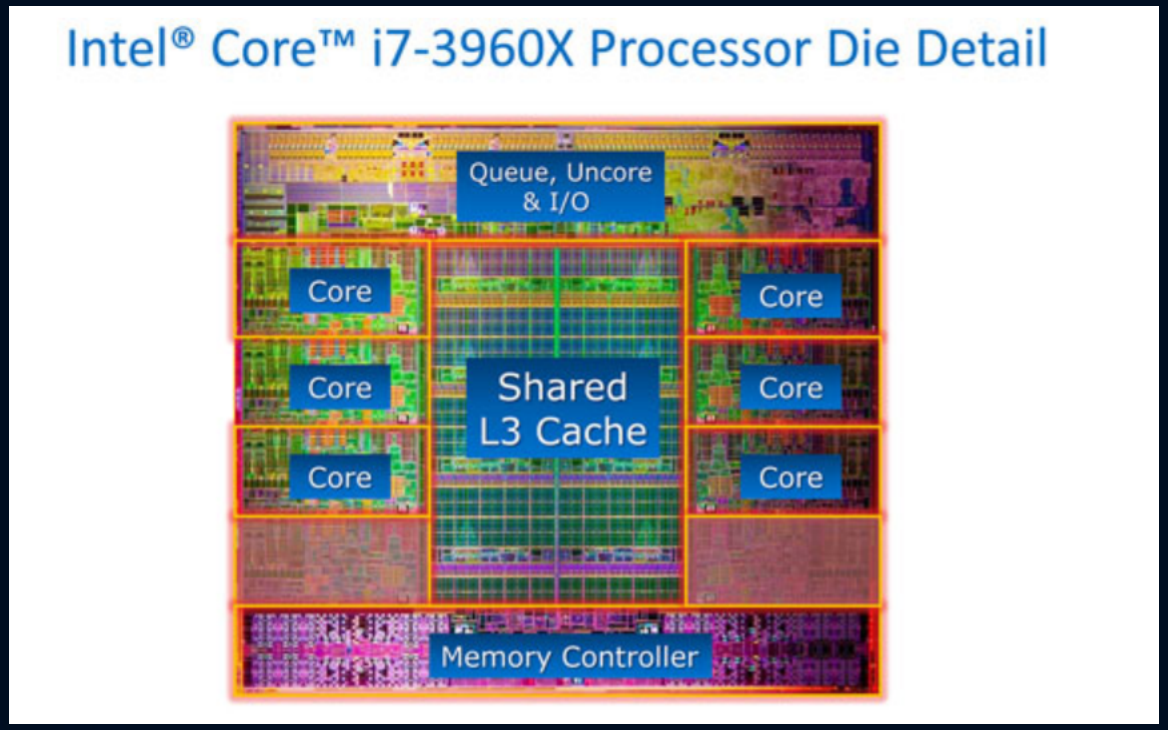
- Each wire in the bus carries a single bit of information, which means the more wires a bus has the more information it can address. For example, a computer with a 32-bit address bus can address 4 GB of memory, and a computer with a 36-bit bus can address 64 GB of memory.



Cache

Cache is a high-speed access area that can be a reserved section of main memory or on a storage device. The two main types of cache are memory cache and disk cache.

Memory cache is a portion of the high-speed static RAM (SRAM) and is effective because most programs access the same data or instructions repeatedly



Internet Cache

- Cache, Internet browser cache, or temporary Internet files with an Internet browser, is used to improve how fast data loads while browsing the Internet.

- In most cases, each time you open a web page, the page and all its files are sent to the browser's temporary cache on the hard drive.

- If the web page and its resources have not changed since the last time you viewed it, the browser loads the data from cache rather than downloading the files again.

Hard Drive Formats

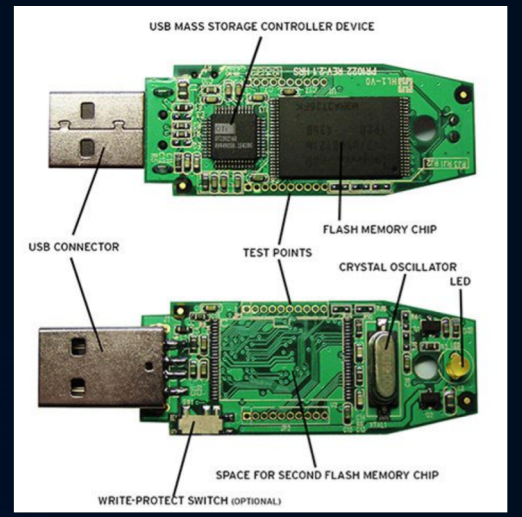
- **NTFS**: The NT File System (NTFS) is the file system that modern Windows versions use by default.
- **HFS+:** The Hierarchical File System (HFS+) is the file system modern macOS versions use by default.
- **APFS**: The proprietary Apple file system developed as a replacement for HFS+, with a focus on flash drives, SSDs, and encryption. APFS was released with iOS 10.3 and macOS 10.13, and will become the mandatory file system for those operating systems.
- **FAT32**: The File Allocation Table 32 (FAT32) was the standard Windows file system before NTFS.
- **exFAT**: The extended File Allocation Table (exFAT) builds on FAT32 and offers a lightweight system without all the overhead of NTFS.
- **EXT 2, 3, & 4**: The extended file system (EXT) was the first file system created specifically for the Linux kernel.

Persistent Storage: Hard Drive, Flash Drive

- Persistent storage of bytes
- "Persistent" means preserved even when not powered
- e.g. Hard drive - stores bytes as a magnetic pattern on a spinning disk
  - aka "hard disk"
  - High pitch spinning sound you may have heard
- Hard drives have been the main, persistent storage tech for a long time
- BUT now flash is getting more popular.

Persistent Storage, Newer Technology: Flash

- "Flash" is a transistor-like persistent storage technology
  "solid state" - no moving parts
- Flash is better than a hard drive in every way but cost - faster, more reliable, less power
- Flash is more expensive per byte
- Formats: usb key, SD card in camera, flash storage built into a phone or tablet or computer
- Flash used to be very expensive, so most computers used hard disks
- Flash is getting cheaper (Moore's law)
- However per-byte, hard drives are still substantially cheaper
- Not to be confused with "Adobe Flash", a proprietary media format

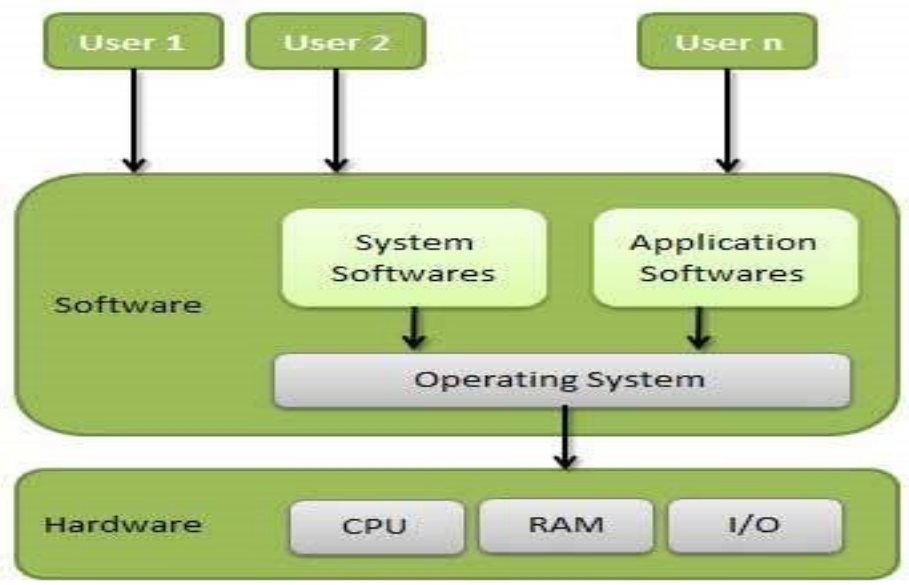


- **Flash does not persist forever. It may not hold the bits past 10 or 20 years. Nobody knows for sure

Operating Systems and File Systems

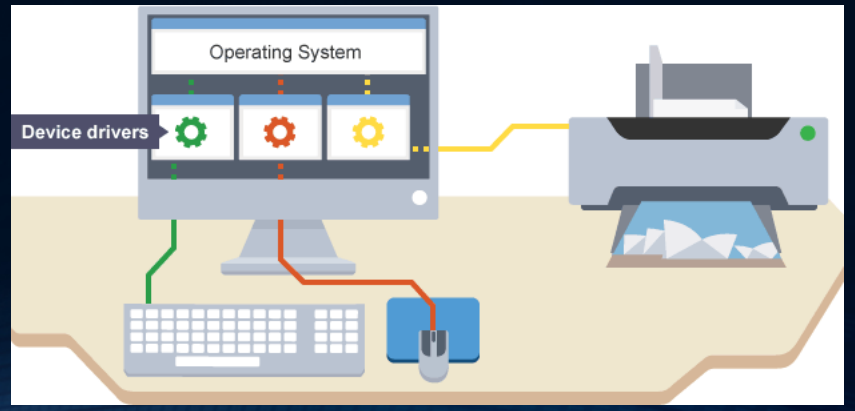
What is an Operating System?

- An *Operating System* is a collection of software that manages computer hardware resources
- The operating system acts as an interface between the user and computer hardware



Key Functions of the Operating System

1. **Interface** between the user and the computer hardware
   - This interface can be a graphical user interface (GUI) in which users click onscreen elements to interact with the OS or a command-line interface (CLI) in which users type commands at the CLI.
2. **Coordinate** hardware components
   - An OS enables coordination of hardware components. Each hardware device speaks a different language, but the operating system can talk to them through the specific translational software called *device drivers*.

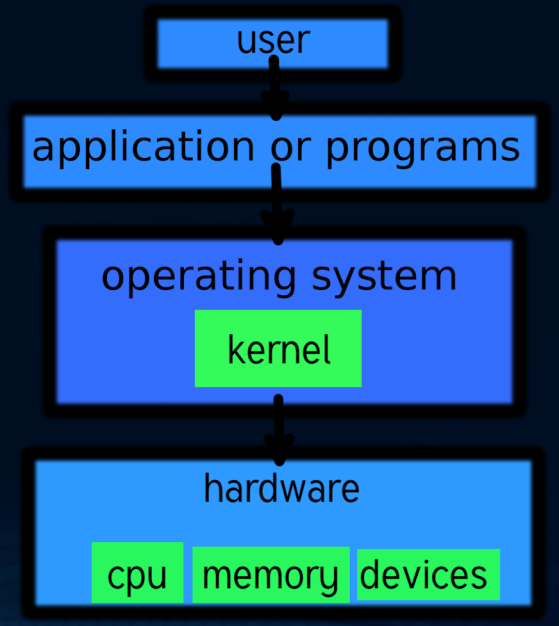


Key Functions of the Operating System

3. **Provide environment** for software to function
   - An OS provides an environment for software applications to function. An application software is a specific software which is used to perform a specific task.
4. **Provide structure** for data management
   - An OS displays structure/directories for data management. We can view file and folder listings and manipulate those files and folders (move, copy, rename, delete, etc.).
5. **Monitor** system health and functionality
   - OS monitors the health of our system's hardware, giving us an idea of how well it's performing. We can see how busy our CPU is, or how quickly our hard drives retrieve data, etc. and also monitor system activity for malware.

How does the operating system do all of this?

- The **Kernel**
  - Central part of the OS
  - It can be thought of as the program which controls all other programs on the computer



What is a File System?

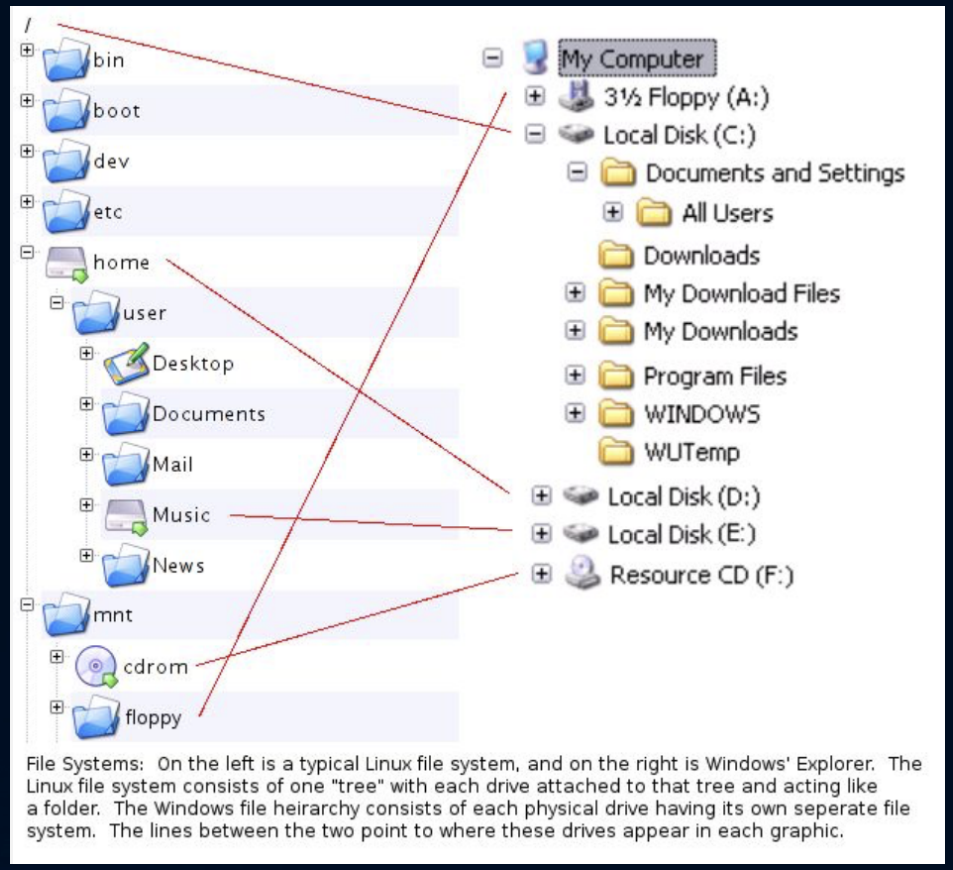
- A **file system** is the part of the operating system responsible for managing files and directories

- In this course we fill focus primarily on the Linux/Unix file system as this is the industry standard and preferred computing environment for Data Scientists

Difference between Linux/Unix & Windows

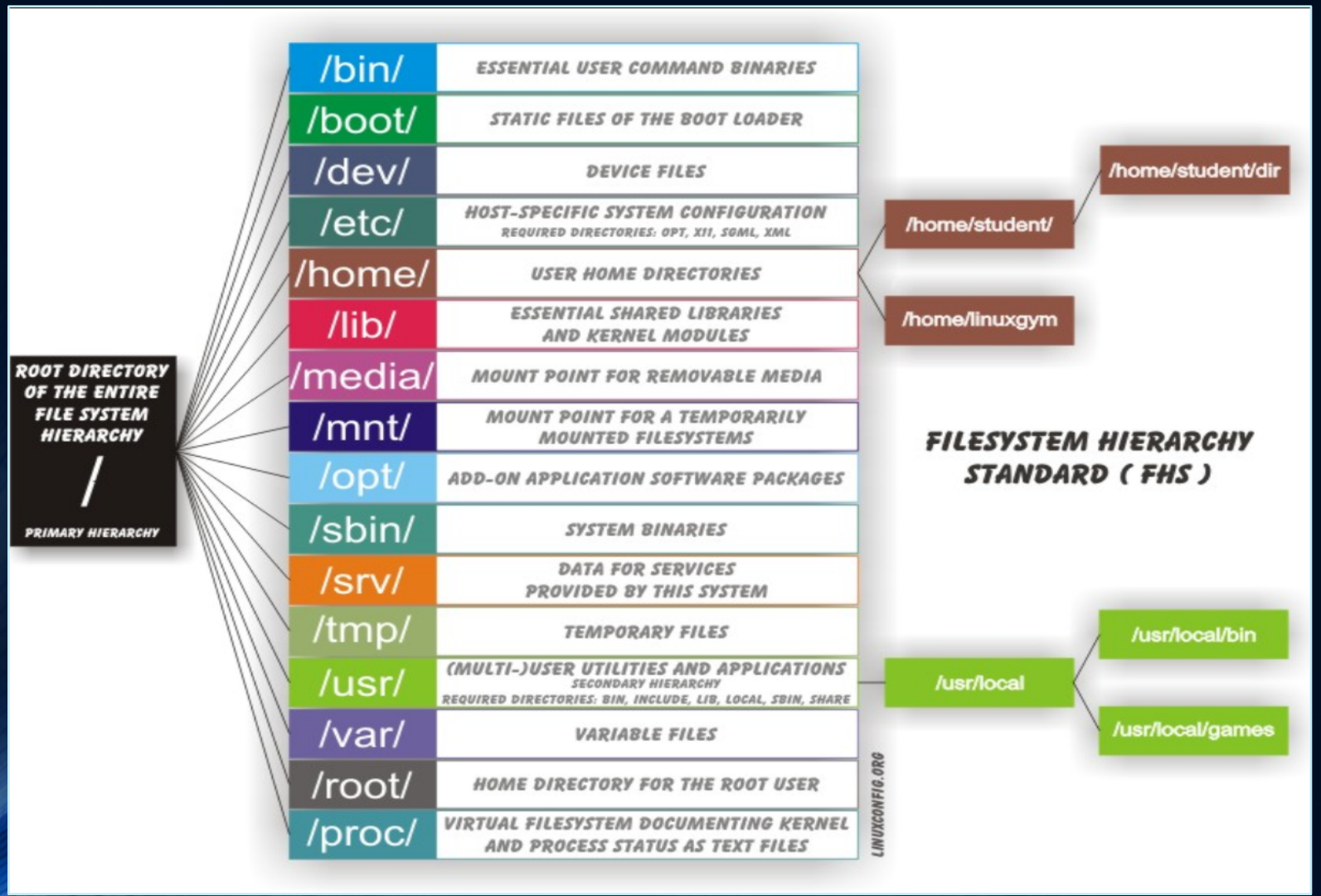
WINDOWS vs LINUX FILE SYSTEM

- In Linux there is a **single hierarchical directory structure.** In Windows, there are typically **many partitions with directories** under these partitions.
- In Linux, everything **starts from the root directory,** represented by '/', and then expands into sub-directories. In Windows, it had **various partitions and then directories under those partitions**
- unlike Windows, **Linux is case sensitive**



File Systems: On the left is a typical Linux file system, and on the right is Windows' Explorer. The Linux file system consists of one "tree" with each drive attached to that tree and acting like a folder. The Windows file heirarchy consists of each physical drive having its own seperate file system. The lines between the two point to where these drives appear in each graphic.

Linux File Systems



Paths Explained

- A file has *two* key properties:
  - Filename
    - usually written as one word + extension
  - Path
    - specifies the location of a file on the computer

- On Windows, paths are written using backslashes (\) as the separator between folder names.
- OS X and Linux use the forward slash (/) as their path separator.
  - *If you want your programs to work on all operating systems, you will have to write your Python scripts to handle both cases. (import os)*

- There are two ways to specify a file path.
  - An **absolute** path, which always begins with the root folder
  - A **relative** path, which is relative to the program's current working directory

Directory Terminology

- path: list of names separated by “/”
- Absolute Path
  - Traces a path from root to a file or a directory
  - Always begins with the root (/) directory

  Example: `/home/student/Desktop/assign1.txt`
- Relative Path
  - Traces a path from the current directory
  - No initial forward slash (/)
    - dot (.) refers to current directory
    - two dots (..) refers to one level up in directory hierarchy

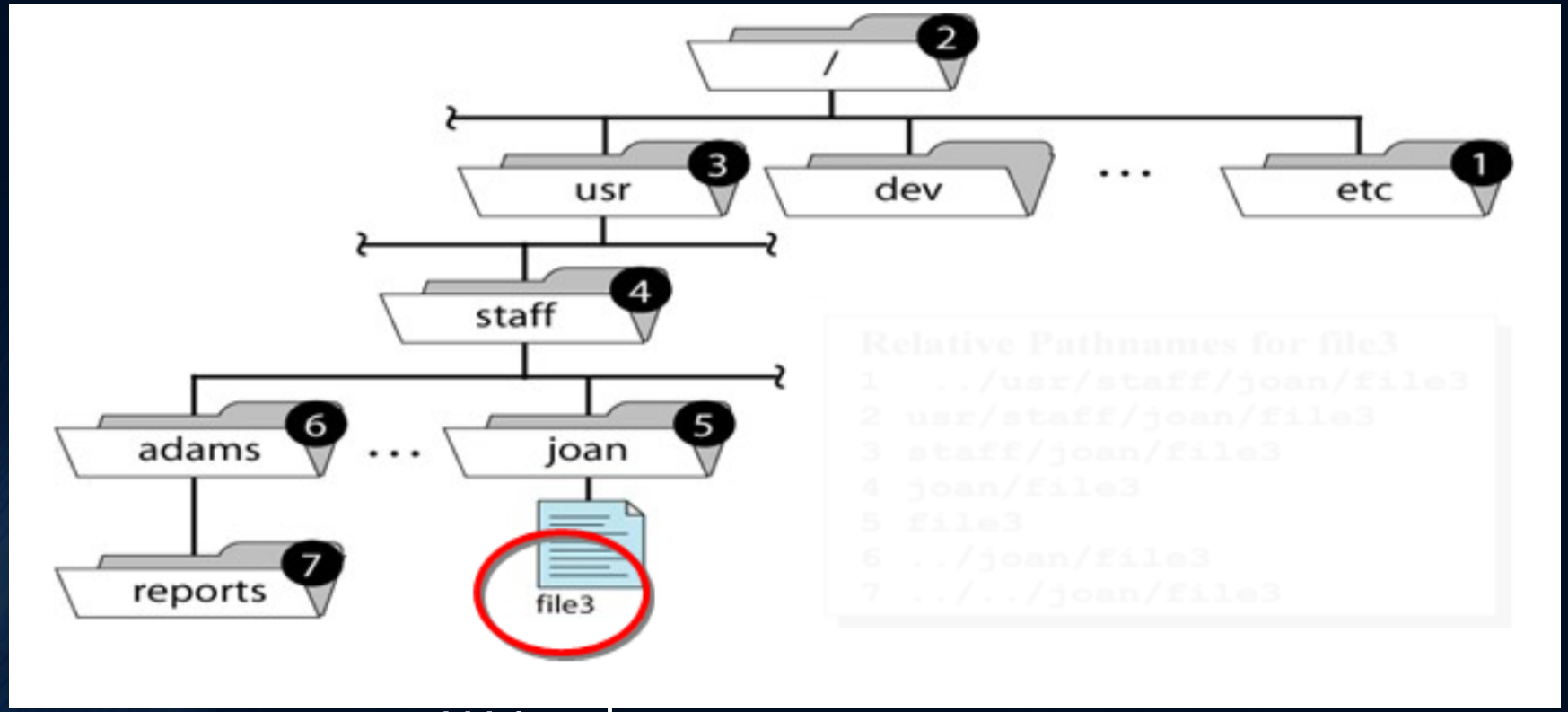
  Example: `Desktop/assign1.txt`

Directory Terminology

Directory terminology

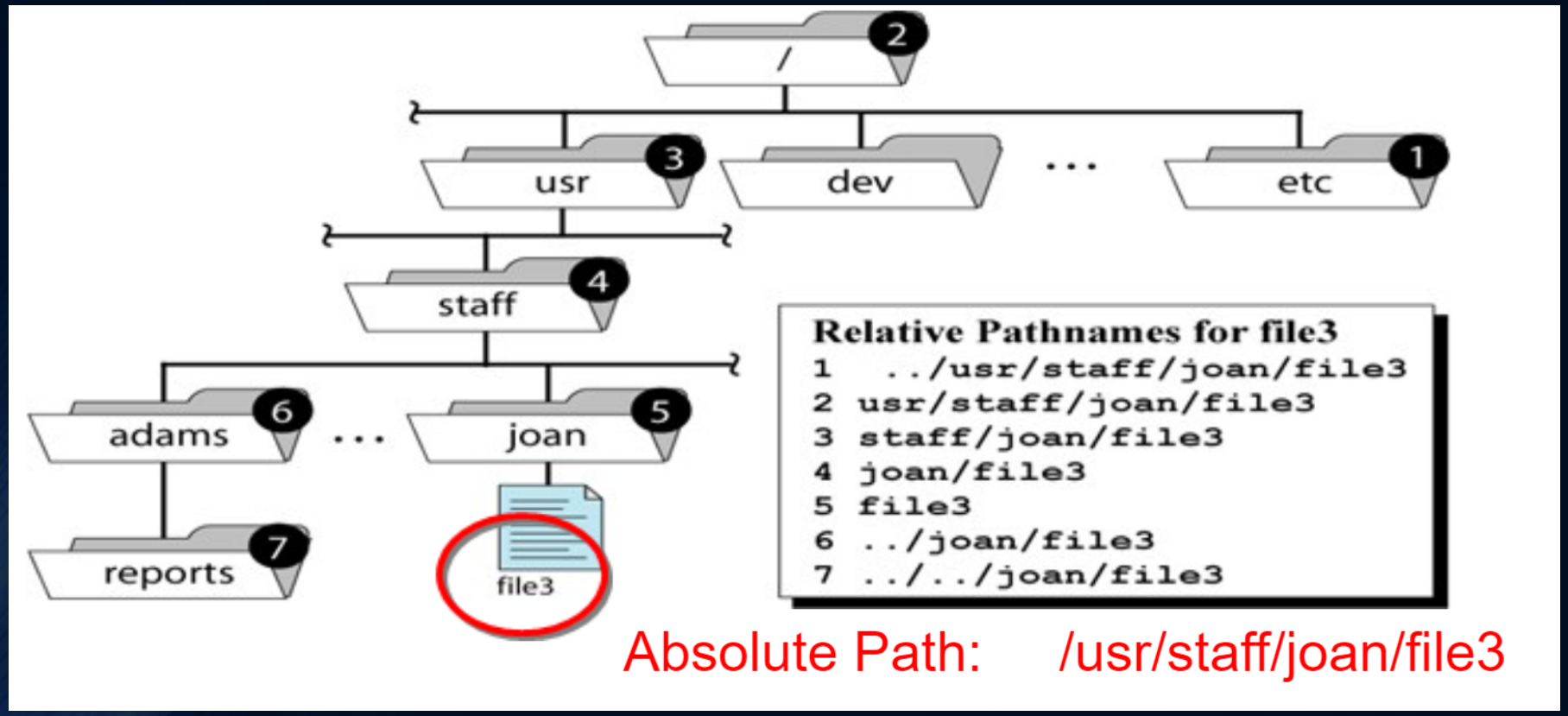
- Root Directory: /
  - top-most directory in any UNIX file structure
- Home Directory: ~
  - directory owned by a user
  - default location when user logs in
- Current Directory: .
  - default location for working with files
- Parent Directory: ..
  - directory immediately above the current directory

Exercise

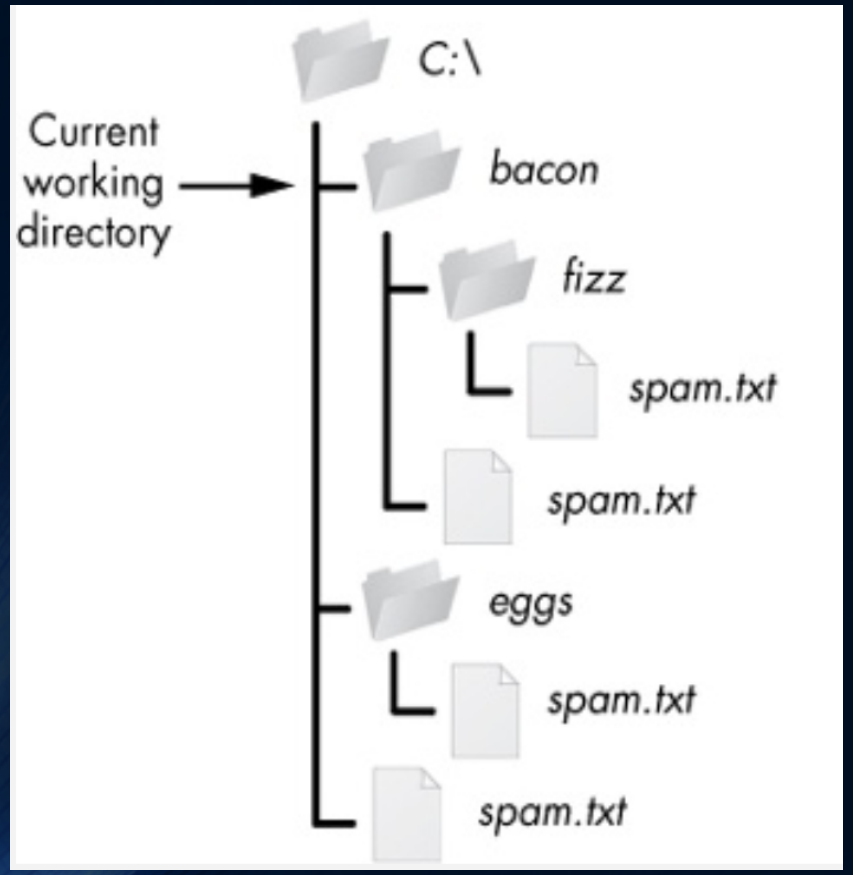


- Write down
  - The relative path to *file3* from numbers 1-7
  - The absolute path of *file3*

Exercise Answers



Directory Terminology

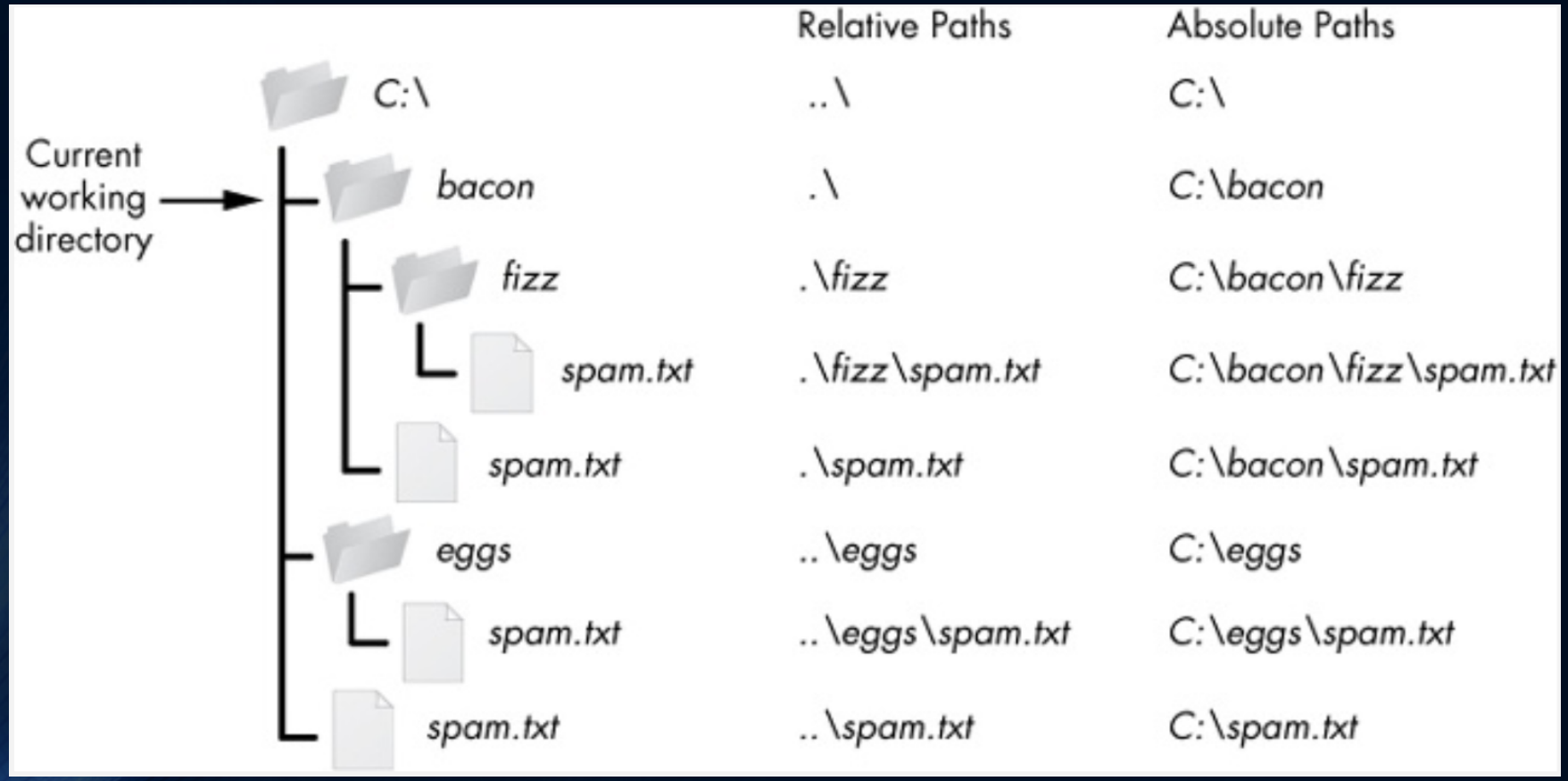


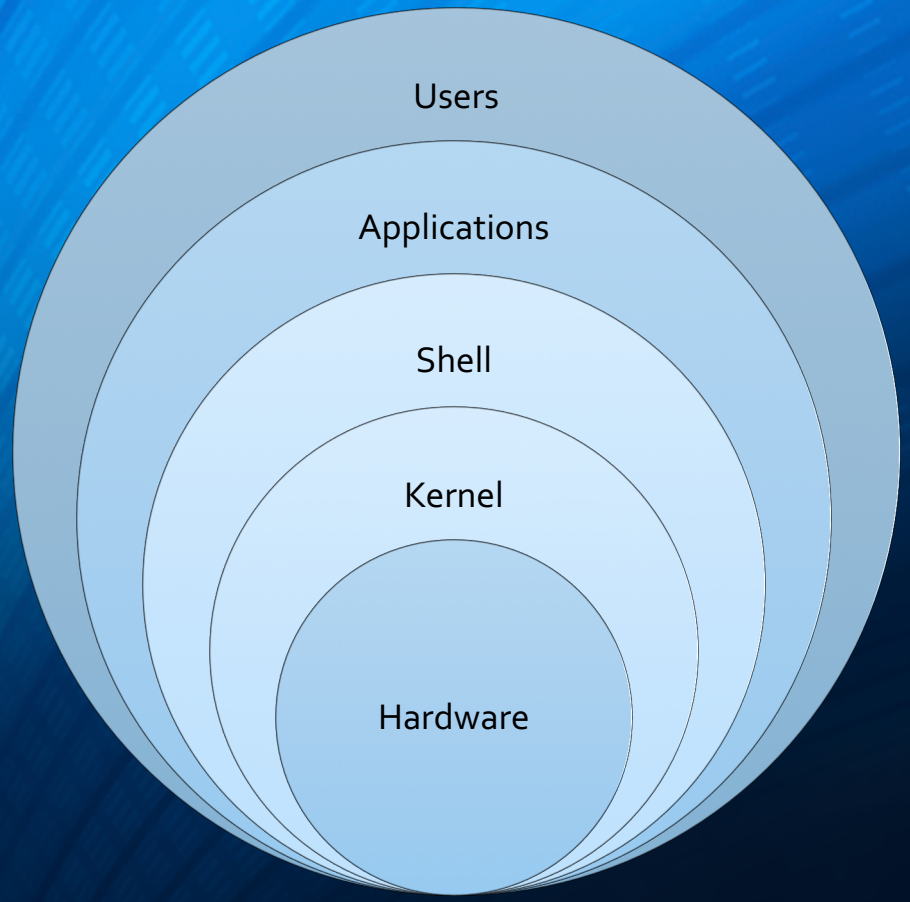
Relative Path

Absolute Path

Given your current working directory is *bacon*, provide the relative and absolute path for each file and directory. *Hint: this is a Windows file system*

Directory Terminology





Intro to Shell/Command Line

LEVELING UP AS A DATA SCIENTIST

What is the Shell/Command Line?

- The **terminal** or **command line** allows a user to interact with the shell
- The **shell** is the user interface for accessing OS services and is the outermost layer of the OS kernel

- In the terminal, each command is preceded by a **$**
  - This is called the **prompt**
  - Just like
    - **>>>** in Python
    - **>** in R

Why use the Command Line?

- The command line will make you a more efficient and productive Data Scientist
- The command line is:
  - *Agile*
  - *Augmenting*
  - *Scalable*
  - *Extensible*
  - *Ubiquitous*

Agile

- Data Science is largely interactive and explorative in nature
  - This is why platforms like jupyter notebook have taken off
- The command line is agile because:
  - It provides a REPL
    - Read-Eval-Print-Loop
    - Your commands are executed immediately, may be stopped at will, and can be changed quickly. This short iteration cycle really allows you to play with your data.
  - It's very close to the filesystem.
    - Because data is the main ingredient for doing data science, it is important to be able to easily work with the files that contain your data set. The command line offers many convenient tools for this.

Augmenting

- The command line is an augmenting technology in that it supplements rather than complements it
- The command line integrates well with other technologies
  - Python and R, for instance, allow you to run command-line tools and capture their output. On the other hand, you can turn your code (e.g., a Python or R function that you have already written) into a command-line tool.

Scalable

- Everything that you type manually on the command line, can also be automated through scripts and tools.
  - This makes it very easy to re-run your commands in case you
    - made a mistake
    - the data set changed
    - your colleague wants to perform the same analysis
  - Your commands can be run at specific intervals, on a remote server, and in parallel on many chunks of data
- Because the command line is automatable, it becomes scalable and repeatable.
  - It is not straightforward to automate pointing and clicking, which makes a GUI a less suitable environment for doing scalable and repeatable data science.

Extensible

- The command line is language agnostic.
  - Allows command-line tools to be written in many different programming languages.
  - These command-line tools can work together, which makes the command line very **flexible**.
  - You can also create your own tools which allows you to **extend** the effective functionality of the command line.

Ubiquitous

- The command line comes with any Unix-like operating system
- Linux/Unix runs on servers, laptops, and embedded systems.
  - Cloud computing is becoming more common and if you ever log in to such a virtual machine or server (you will), there's a good chance you'll have to use the command line.
- The command line has been around for over 40 years and is here to stay for the foreseeable future.
  - Learning how to use the command line (for data science) is therefore a worthwhile investment.

Terminal Navigation – Files and Directories

- How do I figure out where I am in the file system?
  - **$ pwd**
    - You always have a current working directory to which all your commands are in reference
    - The **pwd** command *prints* the *working directory* to the console

Terminal Navigation – Files and Directories

- While it's useful to know where you are, you often want to know what files are there too
- You can *list* the contents of the working directory with:
  - **$ ls**
- You can list other attributes of the files with options (often called *flags*), given after a single dash
  - Example: **$ ls –lahtr**
    - -l: means to list details or *long* format
    - -a: means to list *all* files (even hidden files, i.e., start with period)
    - -h: means to list file sizes as *human-readable* (default is # of bits)
    - -t: means to list in order of *modification time*
    - -r: means to *reverse* the list

Flags Detailed

- Flags can either be combined or separate
  - Combined: **$ ls –la**
  - Separate: **$ ls –l –a**

- There are also sometimes full text versions of flag arguments that you can use by specifying 2 dashes:
  - Example: **$ ls --all --long**

Terminal Navigation – Files and Directories

- Entered on it's own, **ls** lists the contents of the working directory, but you can also add the names of certain files or directories as *arguments*
  - Example: **$ ls file1 file2 file3**
  - Examples: **$ ls ~/Desktop**

Terminal Navigation – Files and Directories

- How do I move to another directory?
  - The **cd** command allows the user to *change directories*
    - The argument to cd is the directory you want to switch to
    - Example: **$ cd Desktop/**
    - Example: **$ cd /Users/michaelarango/Desktop**
      - Note: If the path begins with /, it is *absolute*; if it doesn't, it is *relative*.
  - You can give absolute paths to **cd**, but more often than not you will take advantage of the *special directory notations* we learned earlier:
    - To switch to the directory immediately above (parent):
      - **$ cd ..**
    - To switch to the home directory:
      - **$ cd ~**

Demo

Manipulating Files and Directories

- How can I make files?
  - You can make files with **touch**
    - Syntax: **$ touch <file_name>**
    - Example: **$ touch my-new-file.txt**
  - **touch** is a tool to change timestamps but will create a file if one with that name does not already exist

- How can I copy files?
  - You can *copy* files with **cp**
    - Syntax: **$ cp <file_to_copy> <name_of_copy>**
    - Example: **$ cp raw-data.csv raw-data-copy.csv**
      - If there is already a file with the name you assign to the copy, the file will be overwritten

Manipulating Files and Directories

- How can I move files?
  - You can us the **mv** command to *move* files
    - Syntax: **$ mv <file_to_move> <path_to_move_to>**
    - Example: **$ mv project-data.csv data/**

- How can I rename files?
  - You can also use **mv** to rename files
    - Syntax: **$ mv <file_to_rename> <new_file_name>**
    - Example: **$ mv raw-data.txt cleaned-data.txt**
      - Just like **cp**, if there is already a file with the name you assign to the copy, the file will be overwritten
      - You can add the **–i** flag to request confirmation that you want to overwrite *if* the file already exists

Manipulating Files and Directories

- How can I delete files?
  - You can us the **rm** command to *remove* files
    - Syntax: **$ rm <file_to_remove>**
    - Example: **$ rm week3-lecture**
  - CAUTION: Unlike graphical file browsers, the shell doesn't have a trash can, so when you delete files with **rm**, they are gone for good!

- You can pass the names of as many files to **touch**, **cp**, **mv**, and **rm** as you want

Manipulating Files and Directories

- How can I make directories?
  - You can us the **mkdir** command to *make a directory*
    - Syntax: **$ mkdir <directory_name>**
    - Example: **$ mkdir week4**

- How can I remove directories?
  - You can us the **rmdir** command to *remove a directory*
    - Syntax: **$ rmdir <directory_name>**
    - Example: **$ rmdir week4**
  - For extra precaution, **rmdir** only works if the directory is already empty, so you have to delete all the files in the directory first

Manipulating Files and Directories

- What if I'm feeling risky and want to delete a directory and all of it's contents at once?
  - You can us the **rm** command to *remove a directory* if you add the **–r** (*recursive*) flag
    - Syntax: **$ rm –r <directory_name>**
    - Example: **$ rm –r week4**
      - Note that flags *always* come before arguments
- How can I copy directories?
  - You can us the **cp** command to *copy a directory* if you add the **–r** (*recursive*) flag
    - Syntax: **$ cp –r <directory_to_copy> <name_of_copy>**
    - Example: **$ cp –r data more-data**
- Moving directories does note require the recursive flag

Demo

Extras

How do I interpret the output of ls –al?

```
% ls -al
total 126
drwxr-xr-x 13 ege  csci 1024 Apr 26 15:49 .
drwxr-xr-x 15 root root  512 Apr 24 15:18 ..
-rwxr--r--  1 ege  csci  885 Dec  2 13:07 .login
-rwx------  1 ege  csci  436 Apr 12 11:59 .profile
drwx------  7 ege  csci  512 May 17 14:11 330
drwx------  3 ege  csci  512 Mar 19 13:31 467
drwx------  2 ege  csci  512 Mar 31 10:16 Data
-rw-r--r--  1 ege  csci   80 Feb 27 12:23 quiz.txt
```