CS211: Computer Architecture

Summer 2020

Instructors and TAs

- Instructor:
  - David Domingo
    - Email: david.domingo@rutgers.edu
    - Office Hours: Fridays 2pm-3pm
- Teaching Assistants:
  - Yujie Ren
    - Email: yujie.ren@rutgers.edu
    - Office Hours: Mondays 1pm-2pm
  - Shaleen Garg
    - Email: shaleen.garg@rutgers.edu
    - Office Hours: Thursdays 11am-12pm
  - Chengguizi Han
    - Email: chengguizi.han@rutgers.edu
    - Office Hours: Thursdays 2pm-3pm

Course Logistics

- Sakai will be the primary source of course information
  - https://sakai.rutgers.edu
  - Announcements, course material, assignments, etc.
- Lectures
  - Held synchronously online via Sakai Meetings
  - Slides will be uploaded after lecture
  - Lectures will be recorded and available*
- Office Hours
  - Held online via Sakai Meetings
- Piazza
  - Main forum for questions and discussion on lecture materials and assignments.

Textbooks

- Required
  - **Computer Systems: A Programmer's Perspective**, by R.E.Bryant and D.R.O'Hallaron
- Recommended:
  - **Computer Organization and Design: The Hardware/Software Interface** by D.A.Patterson and J.L.Hennessy
  - **The C Programming Language** by B.W.Kernighan and D.M.Ritchie
  - Any book about C similar to (https://publications.gbdirect.co.uk/c_book/)

What You Should Know

- Prerequisite: CS 112 Data Structures
- Programming Languages
- Algorithms
- Data Structures
- Basics of how to write, run, and test programs

Goal

- Understand how programs run on hardware
- Write programs with good performance on modern architecture computers.

What you will learn

- How to program in C and Assembly
- The major hardware components in computer systems
- Trends in technology and computer architecture
- How hardware components are built from digital logic
- How programs written in a high-level language (e.g., C) is actually executed by the hardware
- How to understand and improve the performance of programs

Course Expectations

- Fun part:
  - 3+ Projects (programming assignments)
- Not So Fun Part:
  - 1 Midterm and 1 Final*
- What we expect from you
  - Attend Lectures
  - Ask Questions
  - Ask for help if you feel lost
  - Start programming assignments early
  - Do not copy or cheat

No Late Assignments

- We will not accept late assignments
- Emails with assignment attach will be discarded
- Assignments must be handed-in on Sakai
  - Deadline enforces by Sakai
  - Can submit unlimited number of times

Collaboration vs. Cheating

- Collaboration is encouraged!
  - Learn by discussing and helping each other
  - But, you must not cheat and copy
- Cheating will not be tolerated
  - We will look for cheating
  - Once, found everyone involved will be punished
  - https://www.cs.rutgers.edu/academics/undergraduate/academic-integrity-policy
- If you having any trouble with the course seek help
  - Email me or the TAs

List of topics

- Hardware trends
- C programming
- Data representation
- Assembly program
- Memory Hierarchy/Caching
- Digital Logic

Programming Assignments

- 3 Programming Assignments
- Program in C and/or Assembly
  - Start early, do not wait until the last minute
- Programming and Grading done using iLab Machines
  - https://resources.cs.rutgers.edu/docs/instructional-lab/
  - Sign up for account if you haven't already https://services.cs.rutgers.edu/accounts/
  - Programming in Linux Environment

Grading

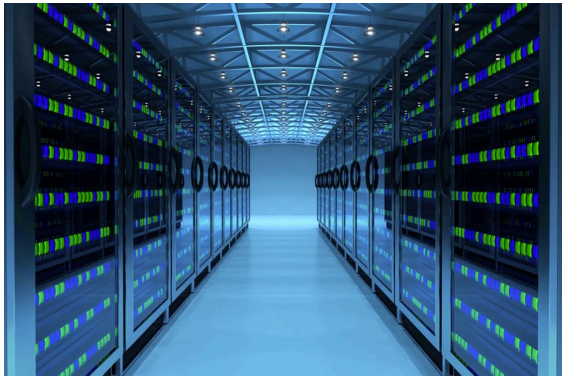
- Grading:
  - 45%: Programming Assignments
  - 20%: Midterm
  - 30%: Final Exam
  - 5%: Class Participation
    - Asking Questions in Class
    - Asking/answering questions in Piazza
  - All exams are cumulative
  - No make-up exams except for university sanctioned reasons
    - Must inform me before the exam

Tentative Schedule

- June 22nd – July 1st : C Programming
- July 6th : Data Representation
- July 8th – July 20th : Assembly Language
- July 22nd : Midterm
- July 29 : Memory Hierarchy/Cache
- July 31st – August 10 : Digital Logic
- August 12th : Final Exam

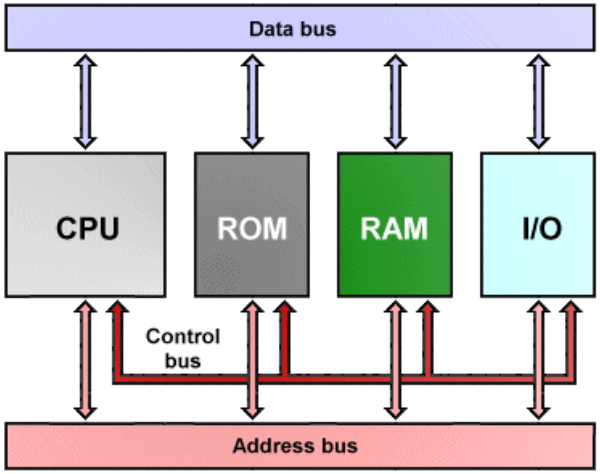
Computer Architecture: An Introduction

Computer Architecture



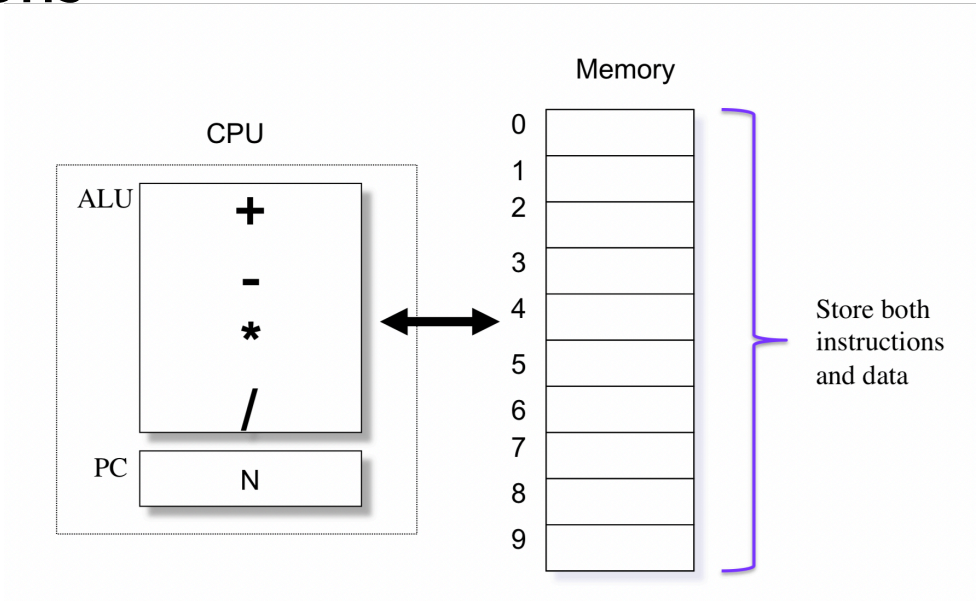
Main Components

- CPU
  - Central Processing Unit
  - Performs computation
- Memory
  - Stores data
- BUS
  - Used for carrying information
  - Communication and data transfer
- I/O Devices
  - Human Interface (mouse, keyboard, screen)
  - Networking
  - Graphics
  - Storage

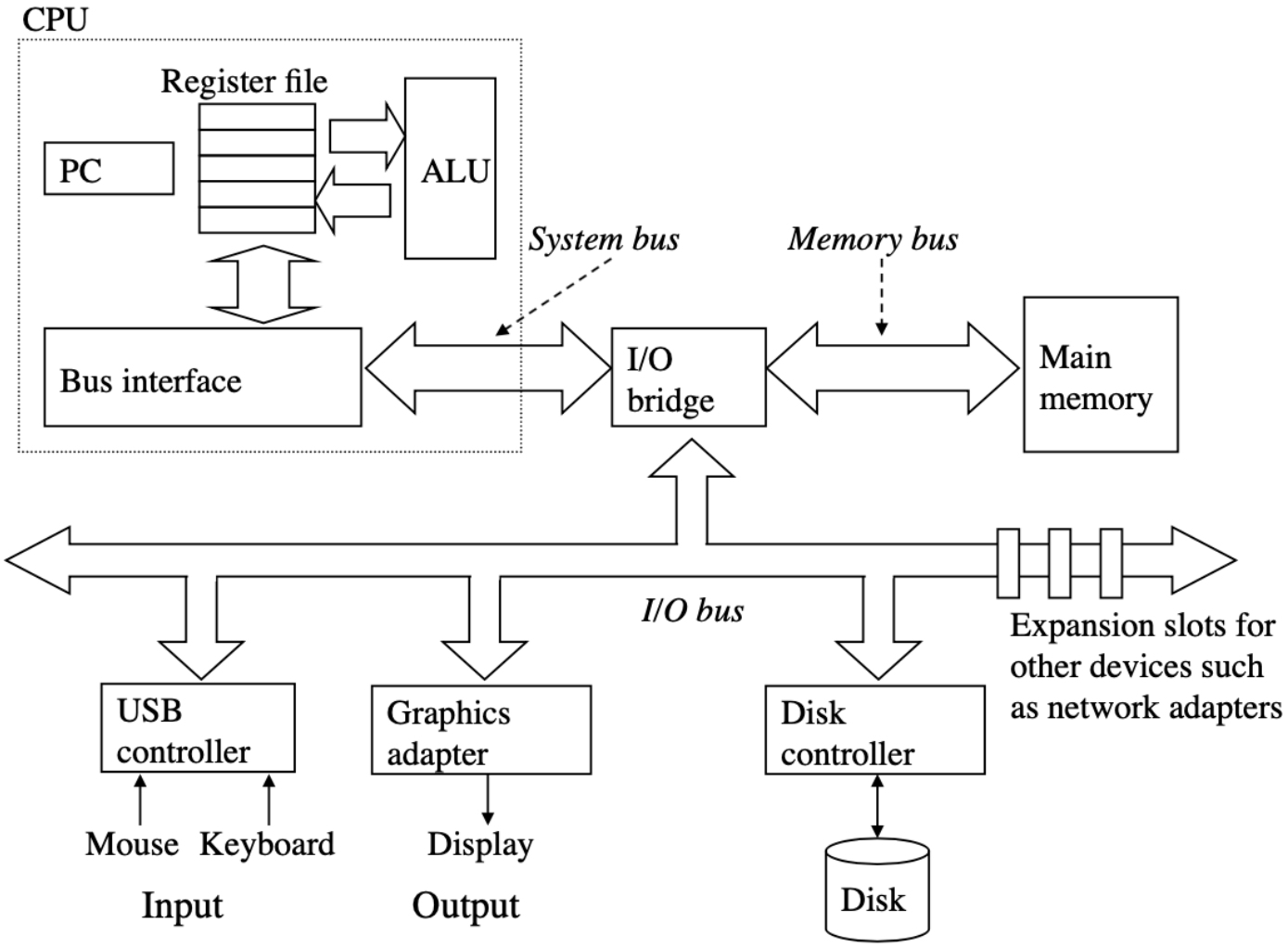


Von Neuman Model

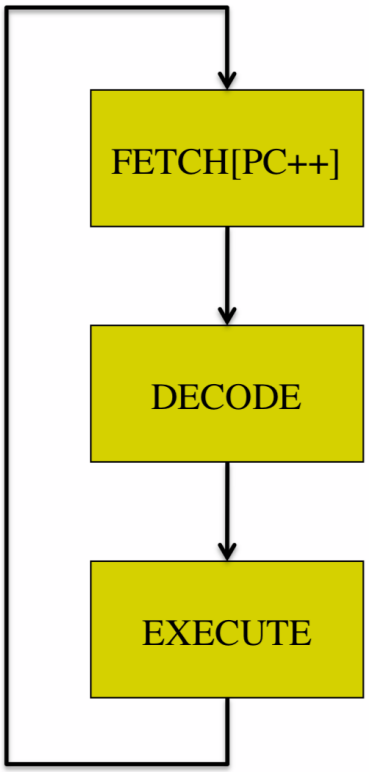
- Computer Architecture Model
- Central processing unit
  - Arithmetic Processing Unit (ALU)
  - Control Unit (instruction register and program Counter
- Memory Component that stores both data and instructions



Von Neumann Model: In Practice



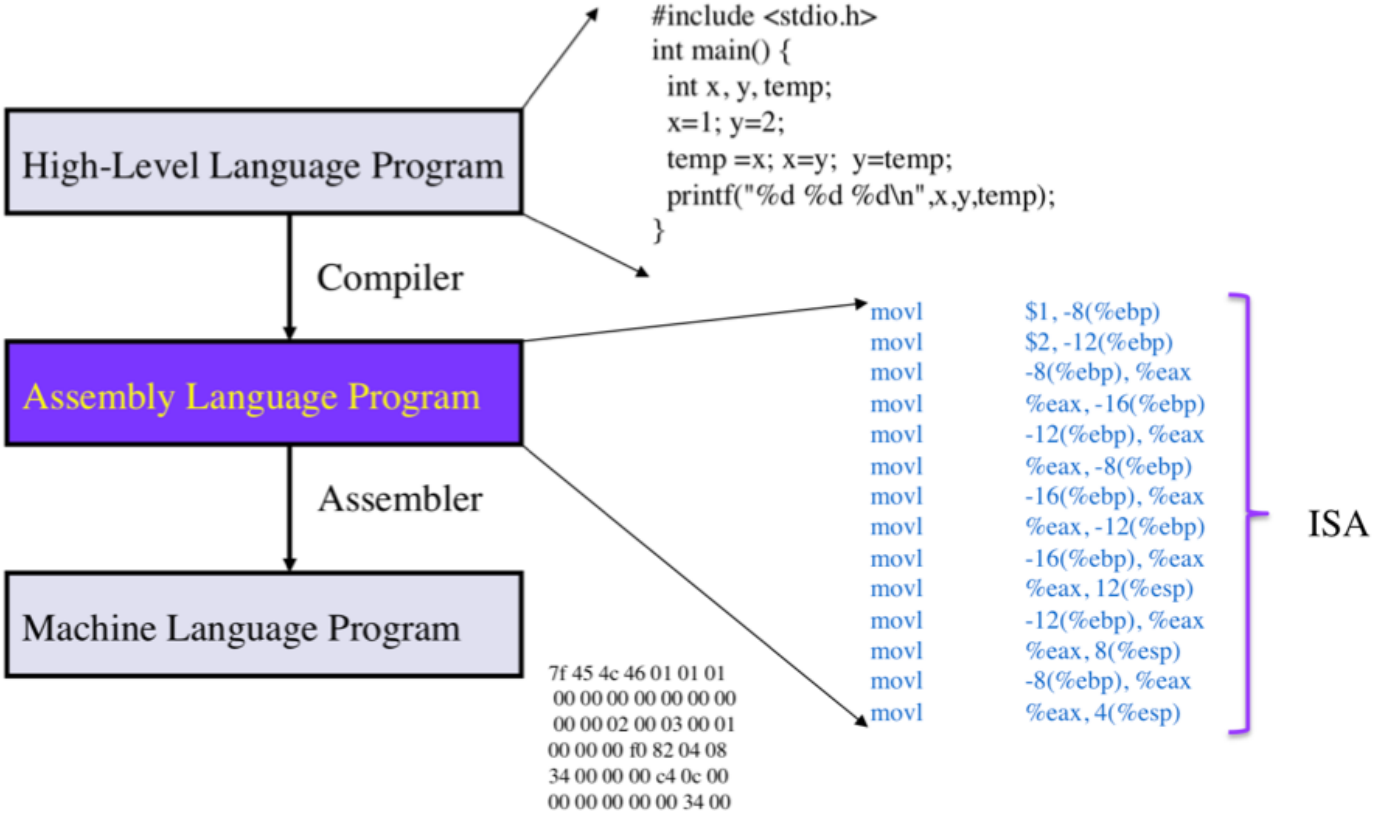
Basic CPU Function



Arithmetic: +, -, *, /
Logic: bre, jmp

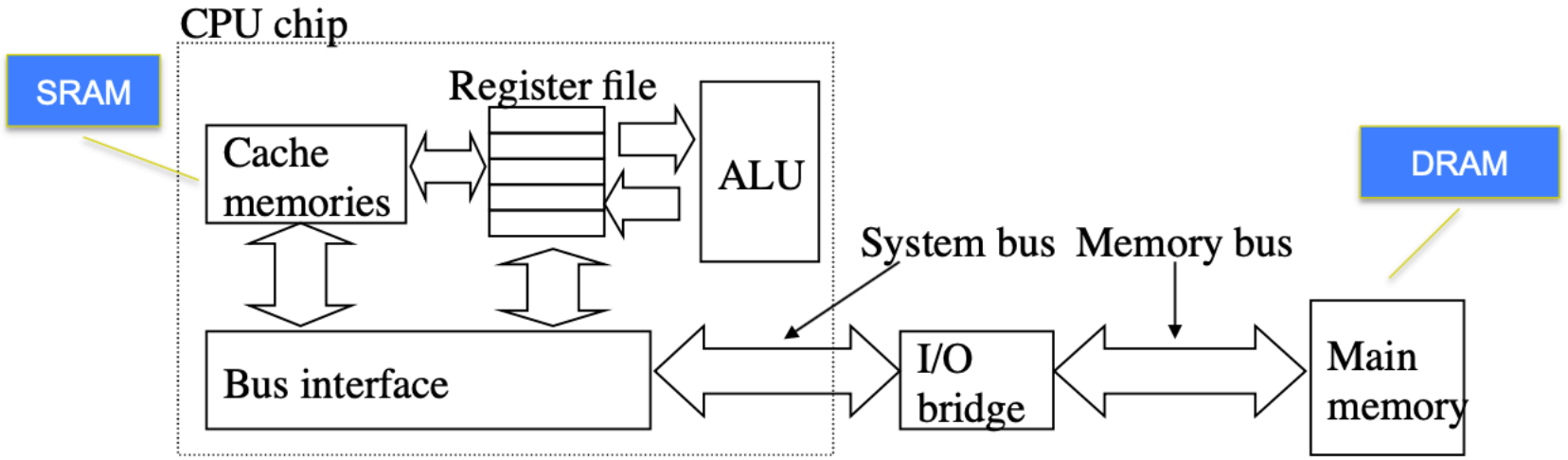
Program to Hardware

- How logic turns into something machines can understand



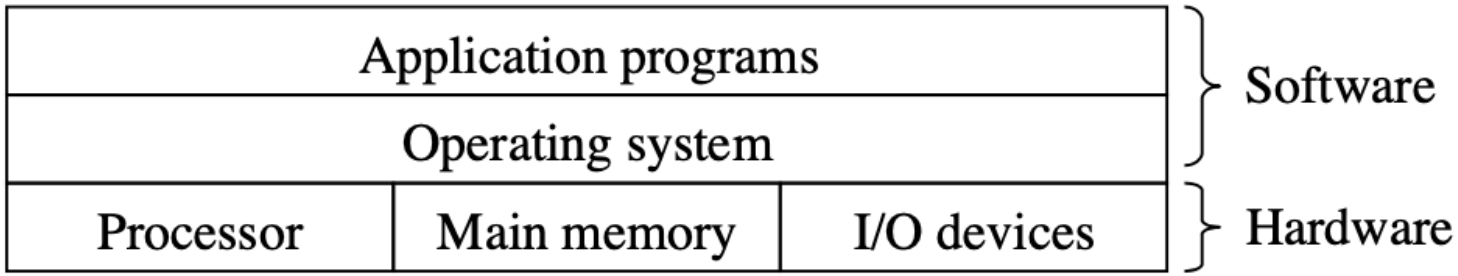
Memory Hierarchy

- A program that resides on disk must be loaded into memory for execution
- Why do we have levels of memory?
  - Larger devices are slower
  - Faster devices are more expensive
    - Typically less storage per dollar

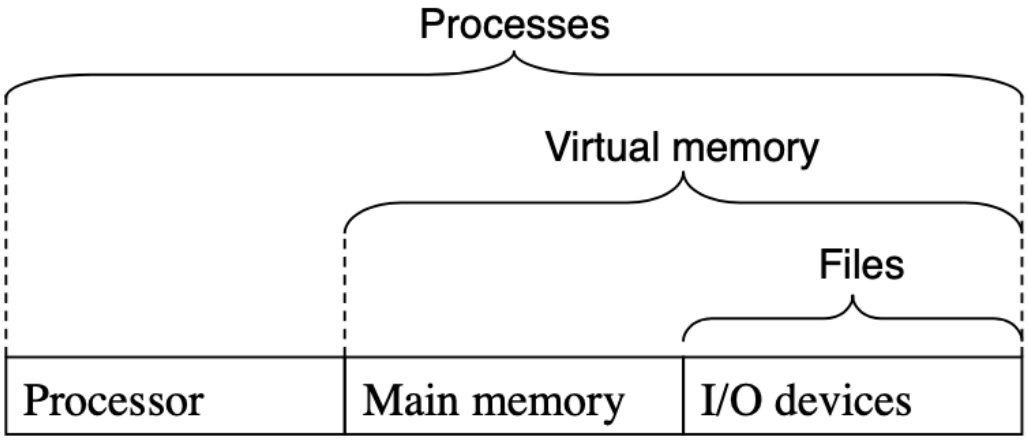


The Operating System

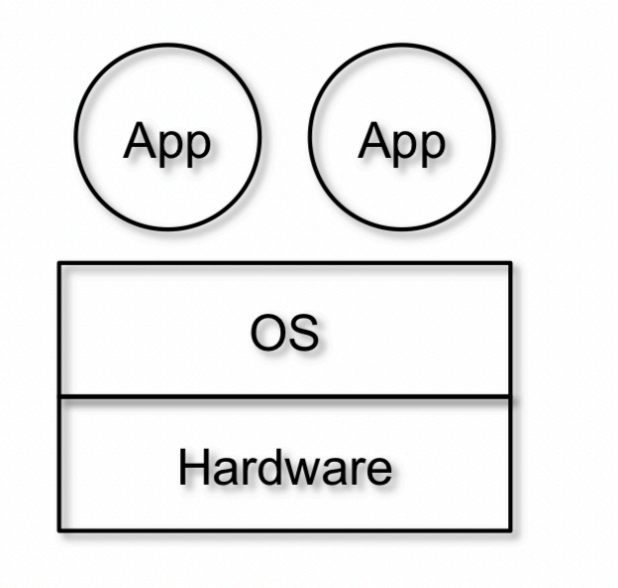
- An operating system manages the hardware

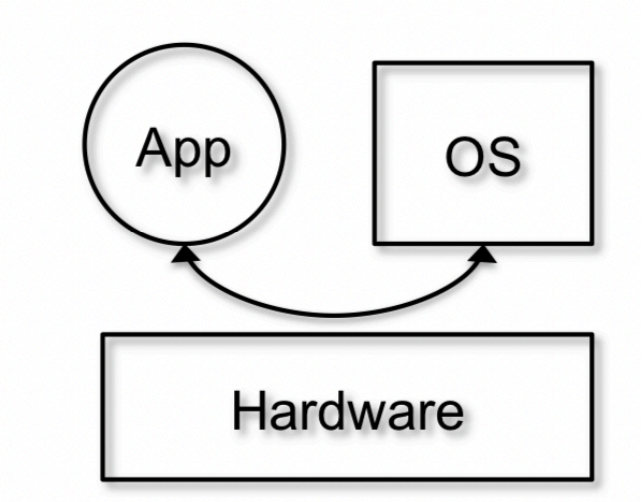


- Manages hardware through abstractions



Operating System Reality





- Main Idea: OS sits on top of hardware facilitate applications applications

- Reality: Apps typically run directly on hardware. Will switch over when OS when needed

Computer Architecture

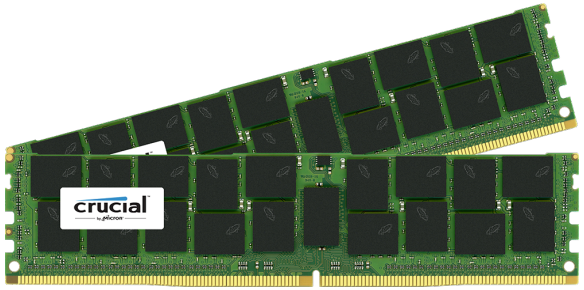
- How to design and build components
- In this class
  - Understand how programs run on current system
  - Understand how current architecture affect programming languages
- Additionally you'll be able to think about
  - How can I make programs faster
  - How can I make programs more power efficient

Architecture Trends

- How have Computer Architecture changed over time?

CPU

Storage



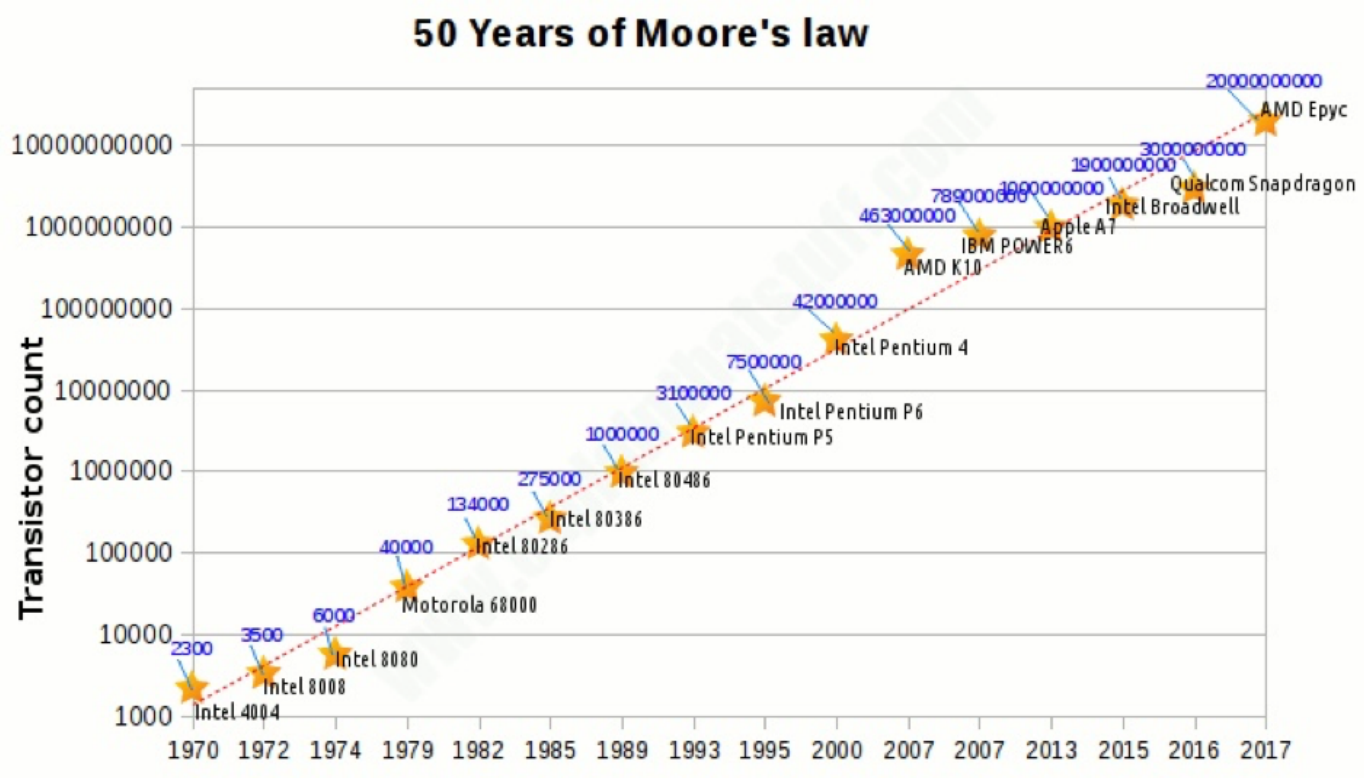
Memory

Moore's Law

- Gordon Moore, an Intel Engineer
- Observed architectural improvement trend
  - # of transistors on a chip doubles every 18 months
- General exponential growth in hardware:
  - Process speed x2 every 18 months
  - Memory capacity x2 every 2 years
  - Disk capacity x2 every year

Transistor Count Trend

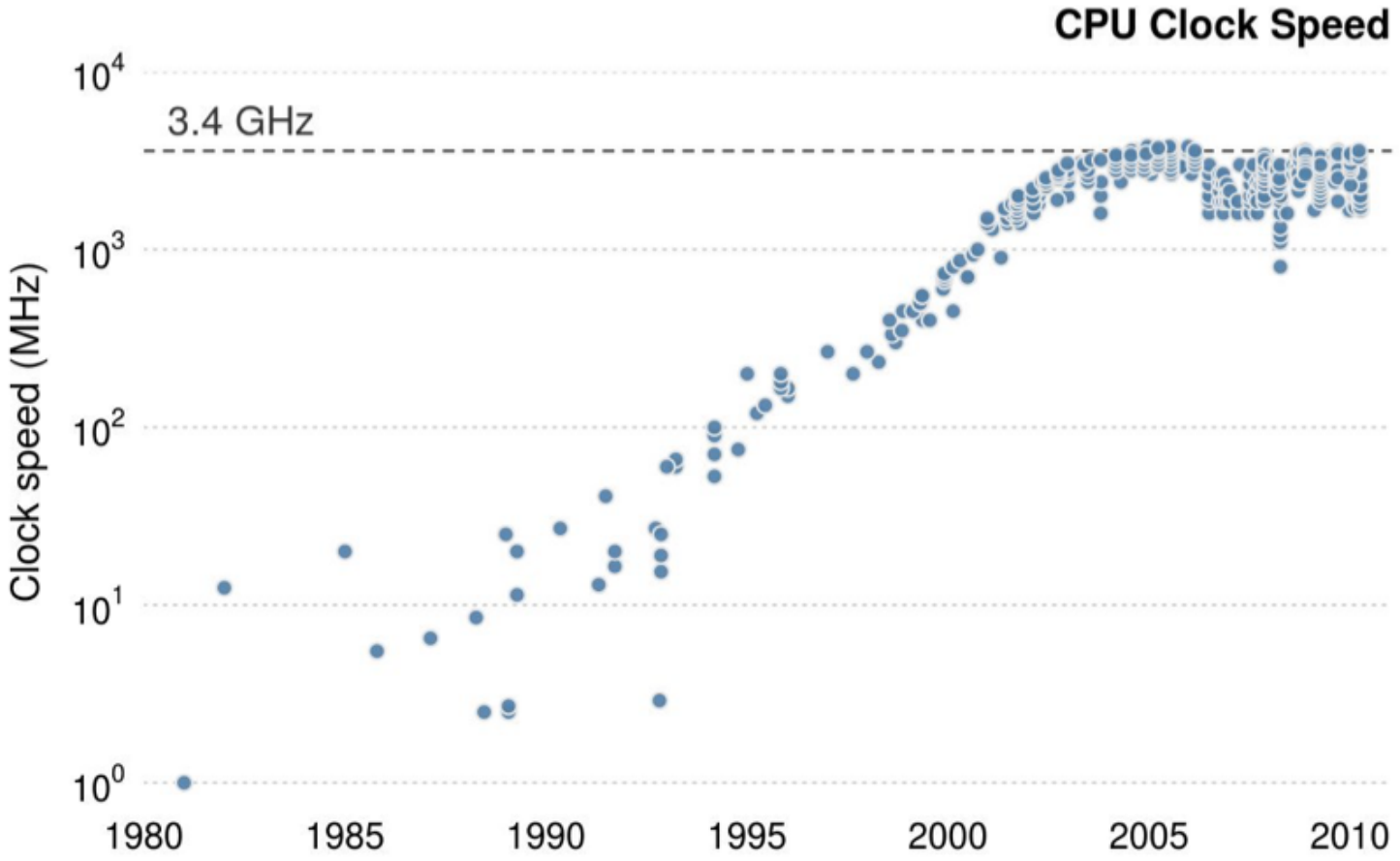
- Transistors are needed to implement logical gates within a CPU



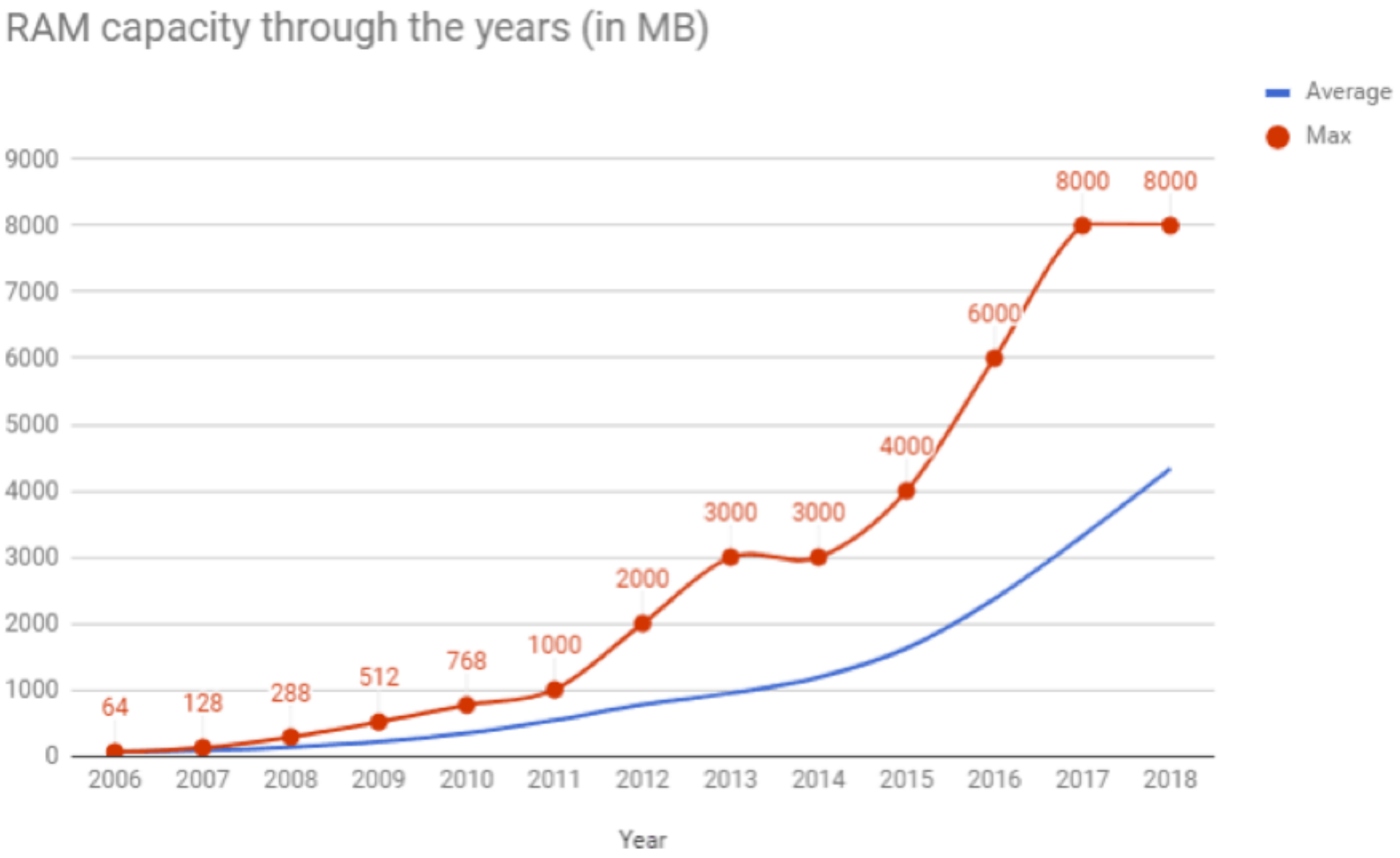
- 32-core AMD Epyc has 19,200,000,000 transistors

Trend: Clock Speed

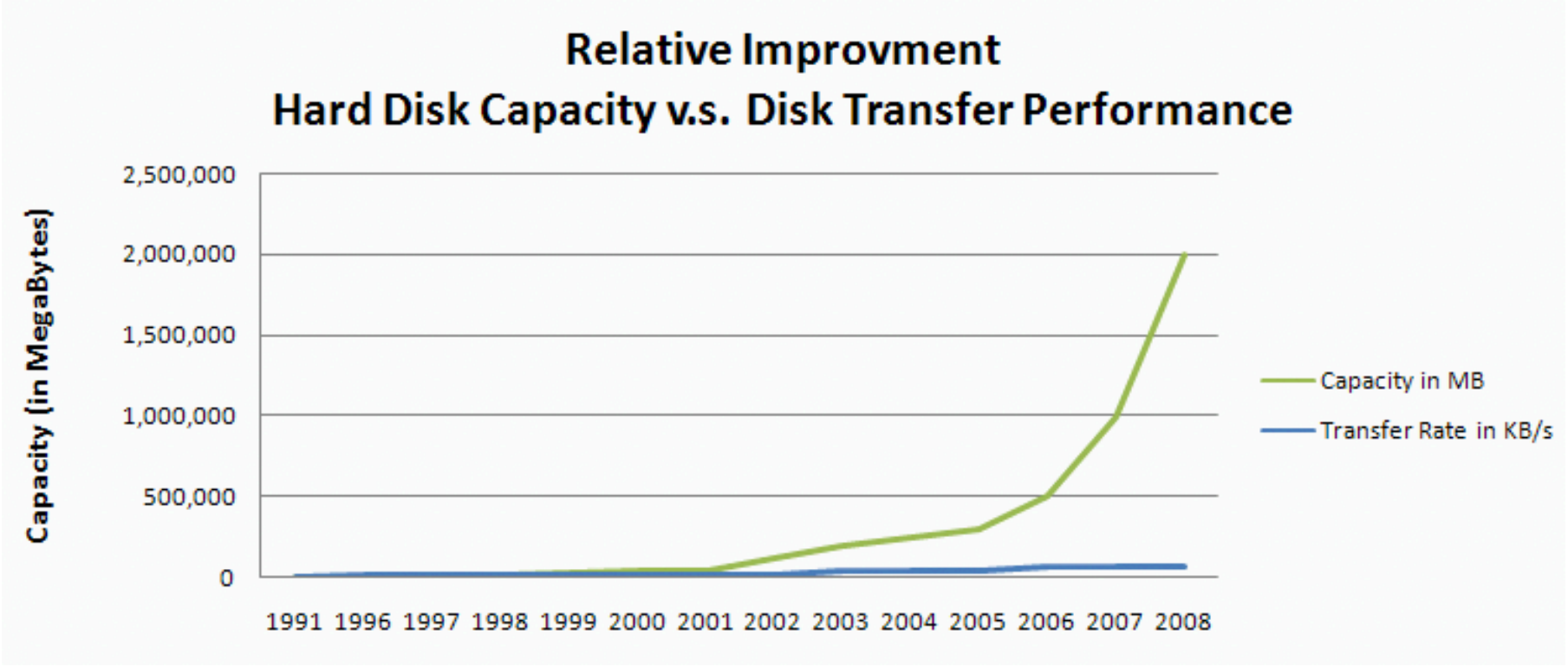
- Clock speed is the rate at which a processer completes a processing cycle



Trend: Memory Capacity

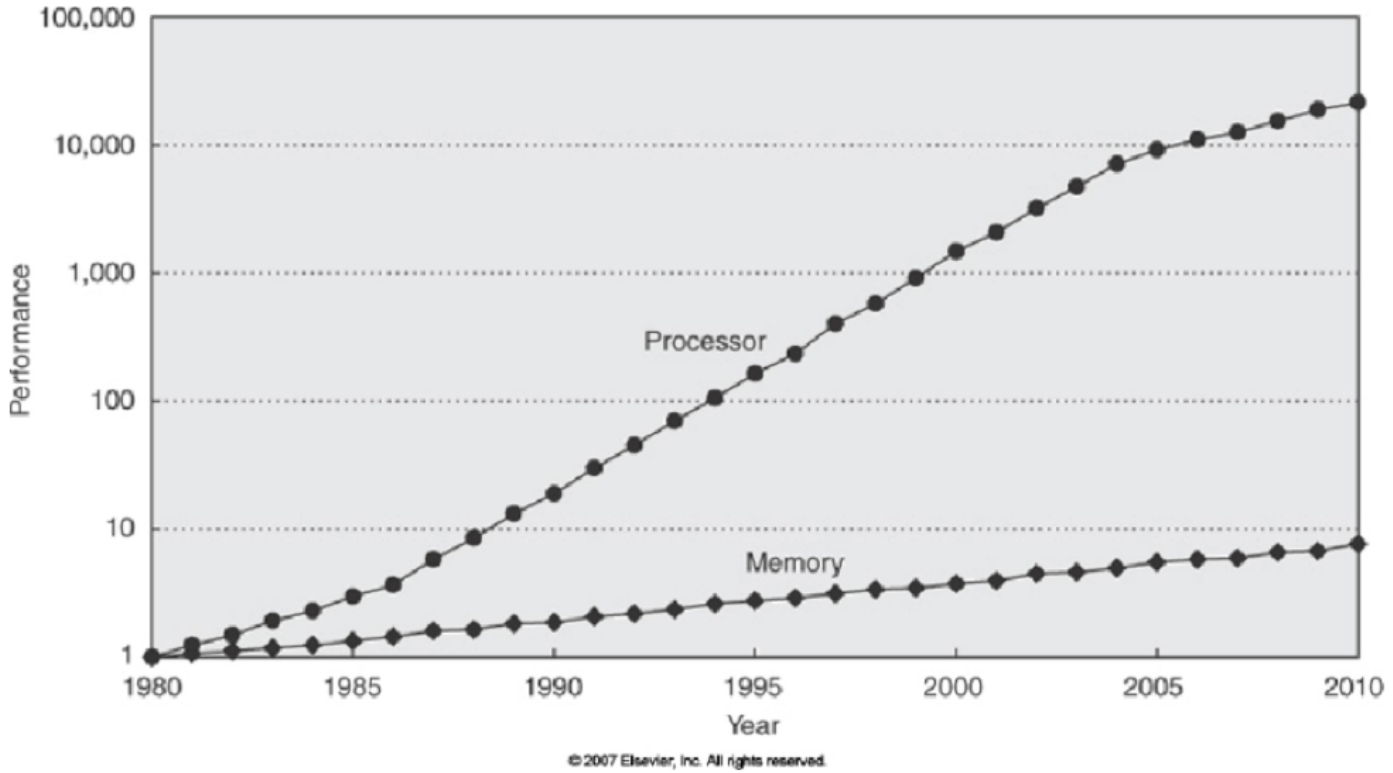


Trend: Disk Capacity



Why are these trends important when it comes to designing system components?

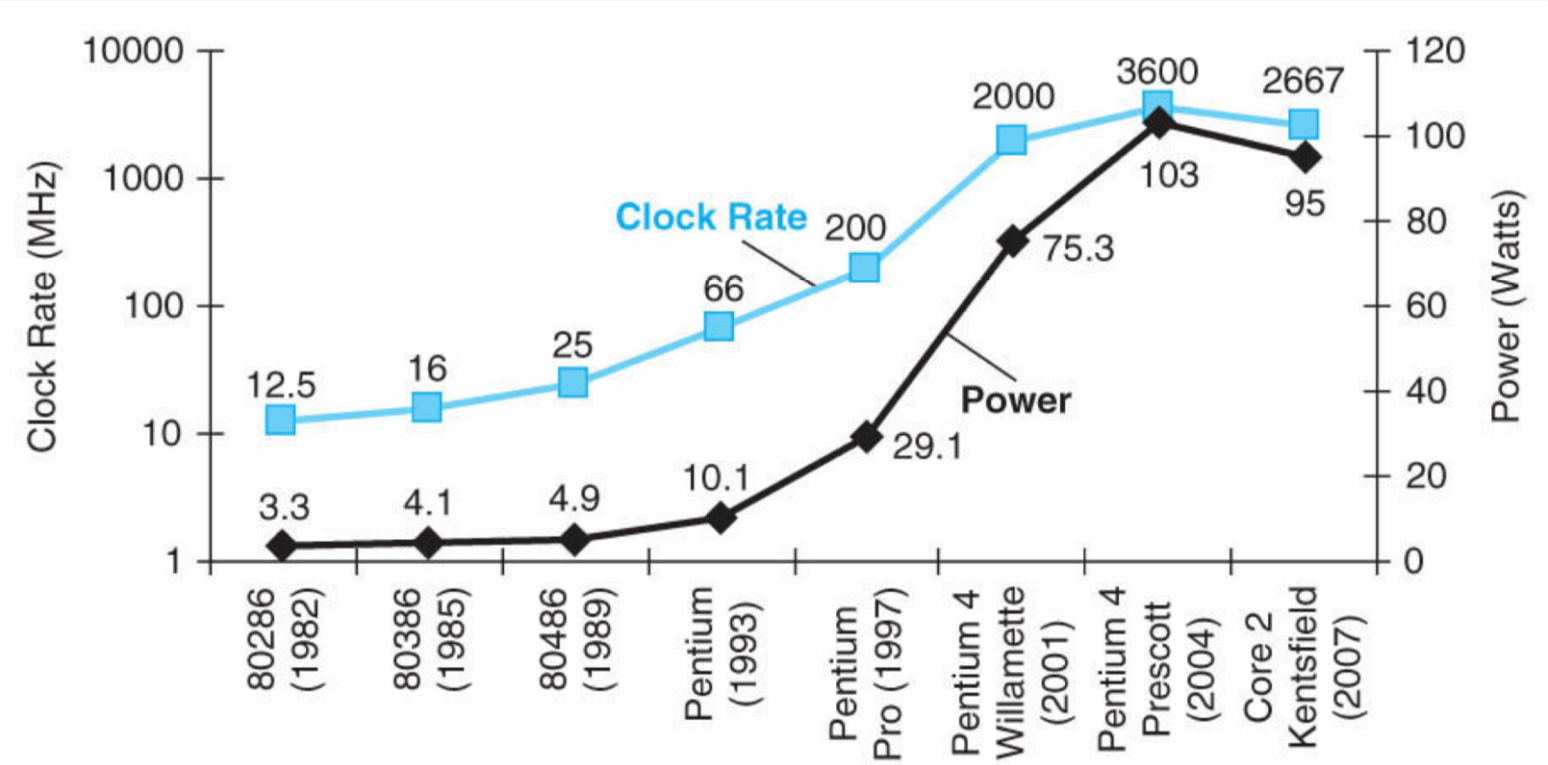
CPU vs Memory Performance Gap



- CPU performance grows faster than Memory
- Implication:
  - Need to make efficient use of the memory hierarchy
  - Registers/Caches

Power Wall

- More transistors -> higher power densities -> higher temperature
- Scaling performance limited by fundamental constraints
  - Power delivery
  - Cooling
- Goal is to maximize performance per watt



In Summary

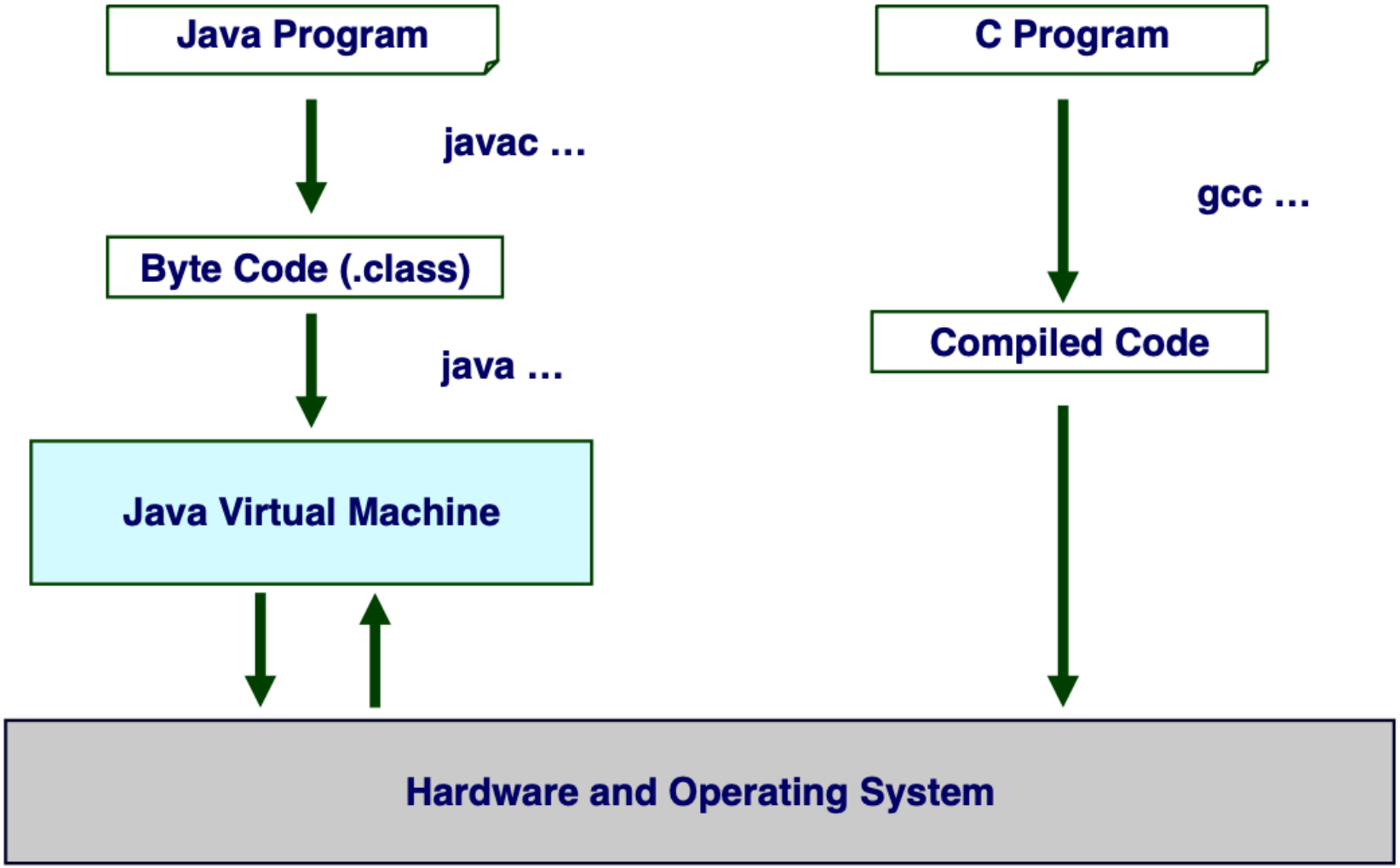
- Todays systems are complex
- Faster processor and systems enable new domains of applications
- Understanding the system is crucial to
  - Develop efficient applications
  - Ensure good tradeoffs between performance/power etc.
  - Address various constraints (memory, power, etc.)

Introduction to C Programming

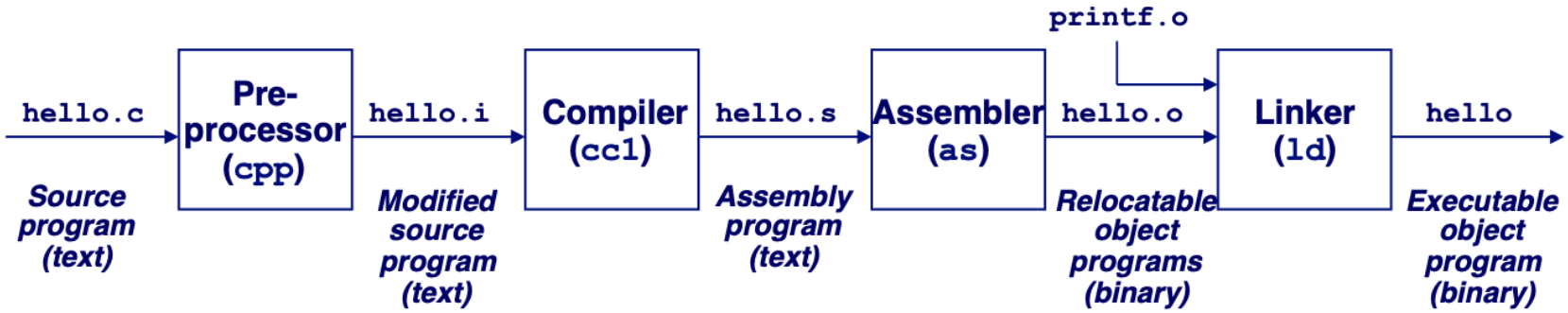
Why Learn C?

- You are learning to be a computer scientist
  - Languages are just tools
  - Choose the tools appropriate for the task
- Current task: Learn how programs written in high-level languages run on computers
- C is closer to machine compared to higher level languages such as Java

Comparison with Java



C Program to Machine Execution



- Compilation System Phases:
  1. Preprocessing: process directives
  2. Compilation: translated into assembly code
  3. Assembly: translates into machine language
  4. Linking: merges libraries

C vs Java Comparison

C

- Procedural
- Compiled
- Low-Level
- User manages memory

Java

- Object Oriented
- Interpreted
- High-Level
- Garbage collector

Anatomy of a C Program

```
#include <stdio.h>
#include <stdlib.h>

char cMessage[] = "Hello\n";

/* Execution will start here */
int main (int argc, char **argv)
{
    int i, count;

    count = atoi(argv[1]);
    for (i = 0; i < count; i++) {
            printf("Hello %d\n", i);
    }
}
```

include files

declaration of global variables

comment

one or more function; each program starts execution at “main”

declaration of local variables

code implementing function

Commenting

- Two ways to comment
- Start comment with //
  - comments until end of line
  - Example:

```
// This is a comment
// This is a second comment
```

- Start comment with /* and end with */
  - comments can span multiple lines
  - Example:

```
/*  This is a comment
that spans multiple lines */
```

- Comments are critical for good development

Variable Declarations

- Each variable has a type, which tells how the compiler how the data is to be interpreted (and how much space it needs, etc)
  - Examples:

    ```
    int counter;
    char letter;
    ```

- Variables can be global or local
  - Global : declare outside scope of any function and can be accessed anywhere
  - Local: declare inside the scope of a function, only accessible from inside of the function

Basic Data Types

Keyword	Data Type	Examples
char	individual character	‘a’, ‘b’, ‘\t’, ’\n’
int	integers	-10, 15, 0, 1324
float	real numbers	-24.4, 0, 4.23
double	real numbers with double precision	-24.4, 0, 4.23

- Modifiers
  - short, long : control size/range of numbers
  - signed, unsigned: include negative numbers or not

Arithmetic Operators

Symbol	Operation	Usage
+	addition	X + Y
-	subtraction	X - Y
*	multiplication	X * Y
/	division	X / Y
%	modulus	X % Y

Special Operators

- Changes value of variable before (or after) its value is used in an expression
- ++ increments a variable
- -- decrements a variable
- Can be used before or after a value

Use	Operation
X++	post increment
++X	pre increment
X--	post decrement
--X	pre decrement

- pre: increment/decrement before using its value
- post: increment/decrement after using its value

Relational Operators

Symbol	Operation	Usage
>	greater than	X > Y
>=	greater or equal to	X >= Y
<	less than	X < Y
<=	less thank or equal to	X <= Y
==	equal	X == Y
!=	not equal	X != Y

- Result is 1 (true) or 0 (false)
- Remember not to confuse equality (==) with assignment (=)

Logical Operators

Symbol	Operation	Usage
!	logical NOT	!X
&&	logical AND	X && Y
||	logical OR	X || Y

- Treats entire variable (or value) as True or False
- Result is either 0 (false) or non-zero (true)

Bit-wise Operators

Symbol	Operation	Usage
~	complement	~X
&	bit-wise AND	X & Y
|	bit-wise OR	X | Y
>>	bit-wise shift right	X >> 2
<<	bit-wise shift left	X << 2

- Operate on bits of variables
- For example:
  - ~0101 = 1010
  - 0101 & 1010 = 0000
  - 0101 | 1010 = 1111
  - 0101 >> 2 = 0001
  - 0101 << 3 = 1000

Control Statements

- Conditional
  - if else
  - Switch
- Iteration (loops)
  - for
  - while
  - do while
- Specialized “go-to”
  - break
  - continue

If Else Statements

```
if (expression){
        statement
}
```

- If an expression is true, then run the statement

```
if (expression A){
        statement 1;
} else {
        statement 2;
}
```

- If an expression is true, then run the statement 1, else run another statement 2

```
if (expression A){
        statement 1;
} else if (expression B){
        statement 2;
} else {
        statement 3;
}
```

- Evaluates all expressions until finds one with non-zero result
  - Executes corresponding statements
  - If all expessions evaluate to zero, executes statements for "else" branch

The Switch Statement

```
switch (expression) {
        case const-1:
                statement 1;
        case const-2:
                statement 2;
        case const-3:
                statement 3;
        default:
                statement-n;
}
```

- Evaluates expression and skips to case corresponding the the result.
- Result must be an integer
- Executes statements corresponding to the result and continues executing ultil encountering a break or end of switch statements
- default always matches

Switch Statement (example)

```
int fork;
…
switch (fork) {
      case 1:
            printf(“take left”);
      case 2:
            printf(“take right”);
            break;
      case 3:
            printf(“make U turn”);
            break;
      default:
            printf(“go straight”);
}
```

Iterations (Loops)

Format	Description

for (start-expression;
condition;
update-expression){
…
}
	- runs zero or more times
	- keeps iterating while cond-expression != 0
	- compute start-expr before first iteration
	- compute update-expr after each iteration

while (expression) {
…
}
	- runs zero or more times
	- keeps iterating while expression != 0
	- compute expression before each iteration

do { … } while (expression)	- runs one or more times - keeps iterating while expression != 0 - computes expression after each iteration

Specialized Go-to's

- break;
  - force immediate exit from switch or loop
  - goes to statement immediately following switch/loop
- continue;
  - skip the rest of the computation in the current iteration of the loop
  - goes to evaluation of conditionals expression for execution of the next iteration

Specialized Go-to's (example)

```
int index = 0;
int sum = 0;

while ( (index >= 0) && (index <= 20) ){
    index = index +1;
    if (index == 11){
        break;
    }
    if ( (index % 2 ) == 1){
        continue;
    }
    sum = sum + index;
}
```

Functions

- Similar to Java Method
- Components
  - Return Type
    - void if no return value
  - Function name
  - Parameters
  - Body
    - statements to be executed
    - return forces exit from function, resuming at statement immediately after function call

```
int Factorial(int n) {
        int i;
        int result = 1;
        for (i = 1; i <= n; i++){
                result = result * i;
        }
        return result;
}
```

Function Calls

- Function call as part of an expression
  - Example: z = x + Factorial(y);
  - Arguments evaluated before function call
  - Return value is used to compute expression
  - Cannot have a void return type
- Function Call as a statement
  - Example: Factorial(y);
  - Can have a void return type
  - Returned value is discared (if there is one)

Basic Input and Output

- Variety of I/O functions in C standard library
- Input:
  - int printf(const char *format, ….);
  - Writes to standard output based on the format provided
  - Examples:
    - printf(“Hello world\n”);
    - printf(“My number grade is %d\n”, grade);
- Input:
  - int scanf(const char *format, …);
  - Reads input from standard input and scans it based on formatting provided
  - Examples:
    - scanf(“%d”, &grade);
    - scanf(“%d %c”, &grade, &lettergrade)
  - Note: Don’t worry about the “&” yet

Basic Input and Output

Common format specifiers

Specifier	Type
%c	character
%d	integer
%u	unsigned int
%f	float
%lf	double
%s	string
%p	pointer

Common escape sequences

Escape Sequence	Character
\n	newline
\t	tab
\\	backslash
\’	single quotation
\”	double quotation

Basic C Program (HelloWorld.c)

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char** argv){
    printf("Hello World!\n");
    return 0;
}
```

Compiling and Running a C Program

- Use gcc to compile your programs
  - Example: gcc HelloWorld.c
  - Will compile into executable named "a.out"
  - Run the program by running ./a.out
- Use the "-o" flag to specify the output name
  - Example: gcc HelloWorld.c -o HelloWorld
  - Will compile program into executable named "HelloWorld"
  - Run the program by running ./HelloWorld

For Next Lecture

- Start practicing C Programming
  - Connect to iLab Machines
  - Start coding, compiling, and running short C programs
  - Get used to the iLab Machine
- For those who want to get ahead
  - Arrays
  - Pointers
  - Structs

iLab Machines
and
Linux

iLab Machines

- Sign up for a iLab account if you haven't already
  - https://services.cs.rutgers.edu/accounts/
- Two Ways to use it
  - Via Secure Shell (SSH)
    - terminal (Mac/Linux)
    - Putty (windows)
  - Via remote desktop
    - X2GO (Windows/Mac/Linux)
    - Microsoft Remote Desktop (Windows/Mac/Linux)
- List of available iLab Machines
  - https://report.cs.rutgers.edu/nagiosnotes/iLab-machines.html

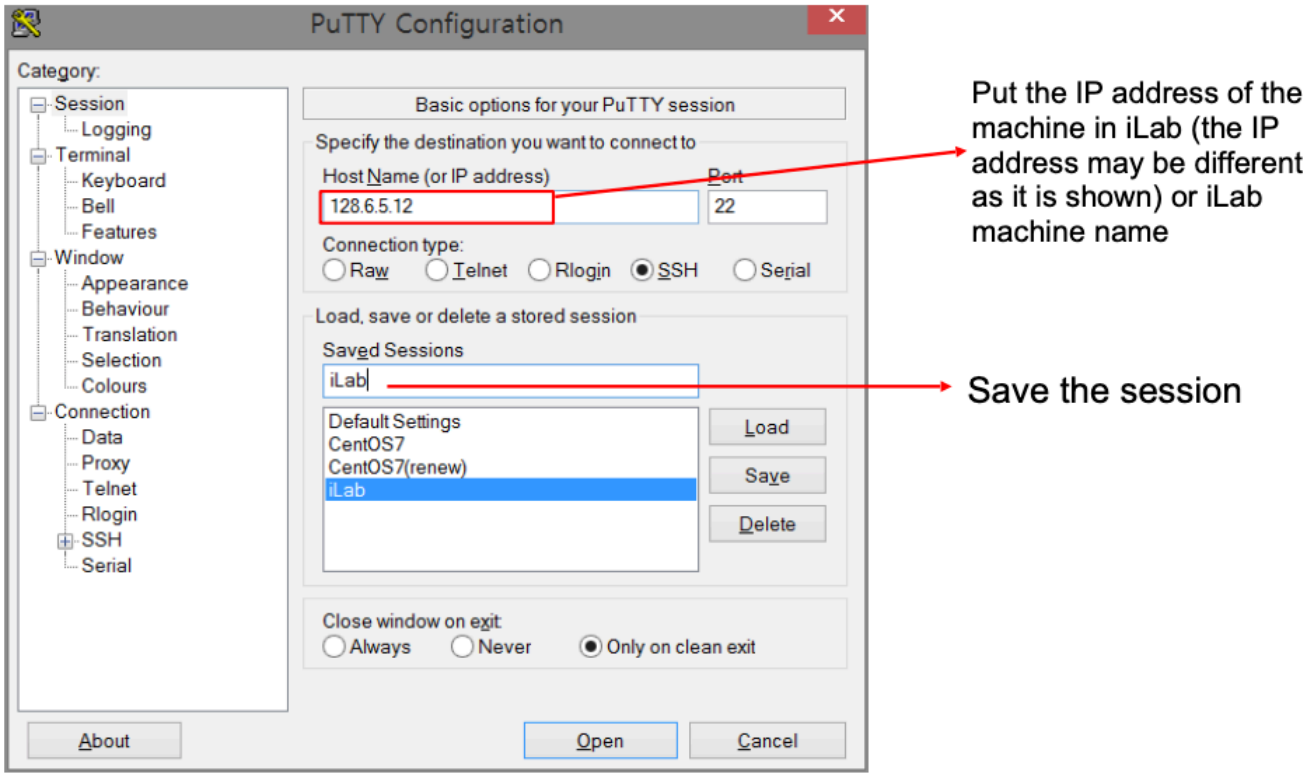
Connecting via SSH: Linux/Mac

- Open a new terminal
- Run “ssh <netid>@<ilabmachine>”
- Type in password to login

Guide: https://www.howtogeek.com/311287/how-to-connect-to-an-ssh-server-from-windows-macos-or-linux/

Connecting via SSH (Putty): Windows

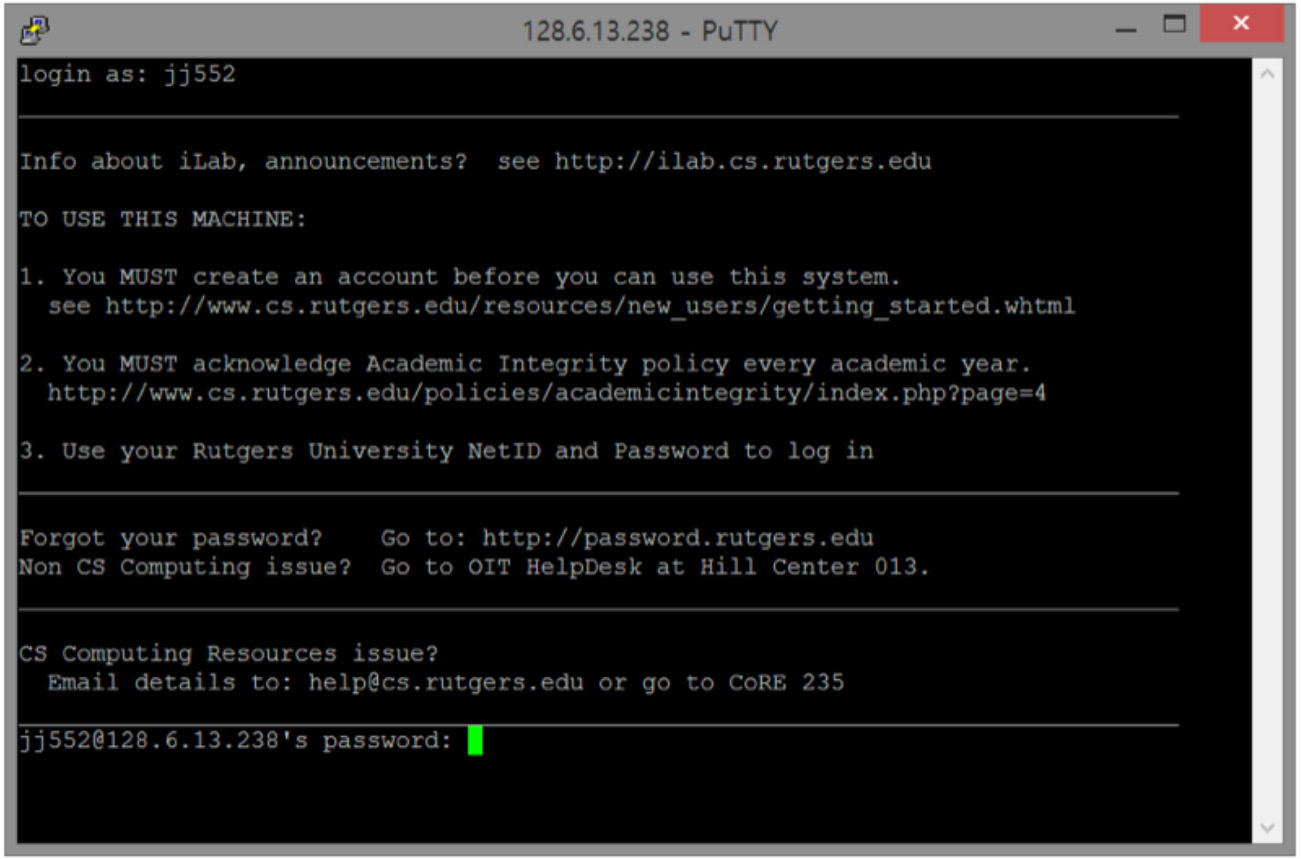
- Install Putty and configure Putty



Guide: https://www.howtogeek.com/311287/how-to-connect-to-an-ssh-server-from-windows-macos-or-linux/

Connecting via SSH (Putty): Windows

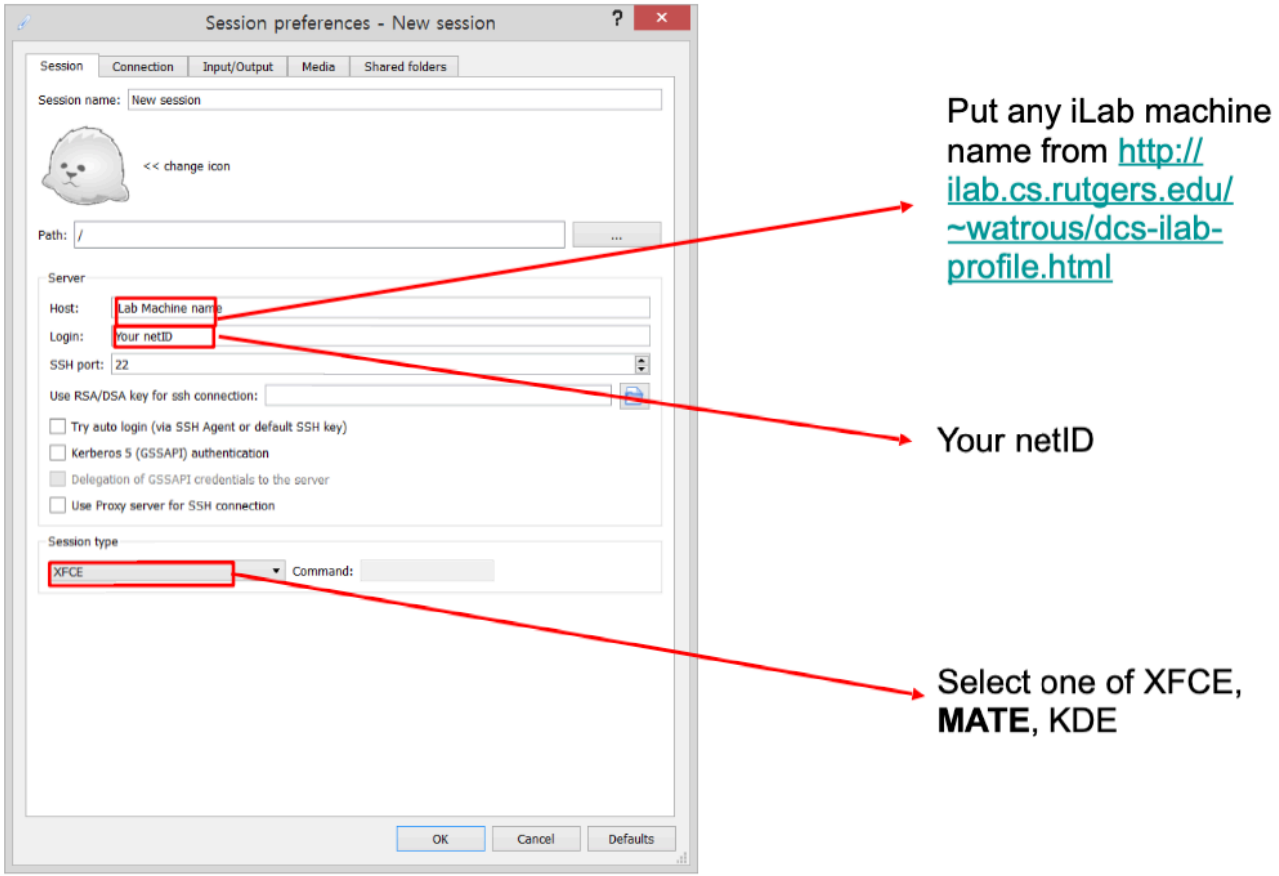
- Save session and open connection



Guide: https://www.howtogeek.com/311287/how-to-connect-to-an-ssh-server-from-windows-macos-or-linux/

Connecting to iLab Machines: X2GO Remote Desktop (Windows/Mac/Linux)

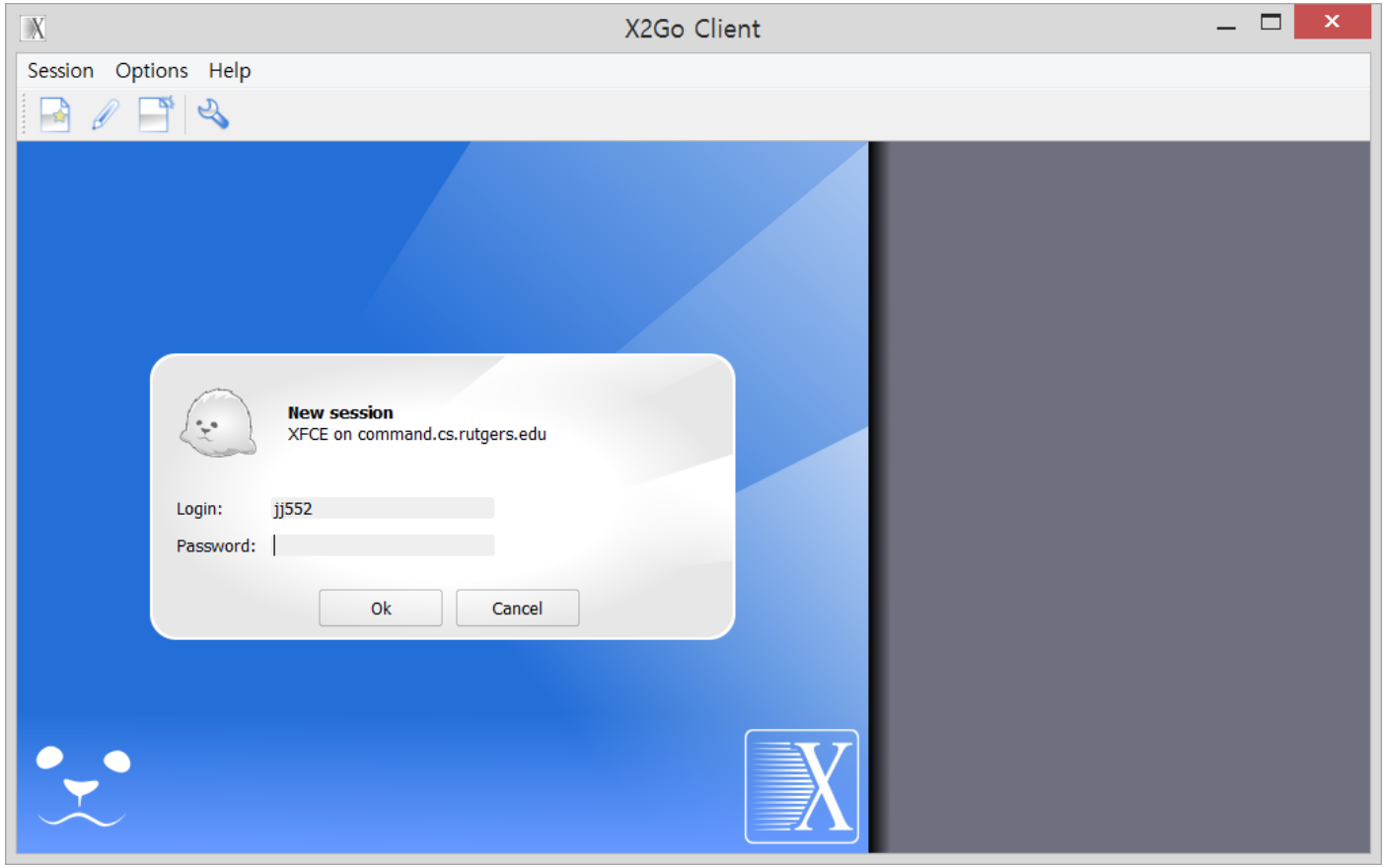
- Create new session



Guide: https://resources.cs.rutgers.edu/docs/other/x2go/

Connecting to iLab Machines: X2GO Remote Desktop (Windows/Mac/Linux)

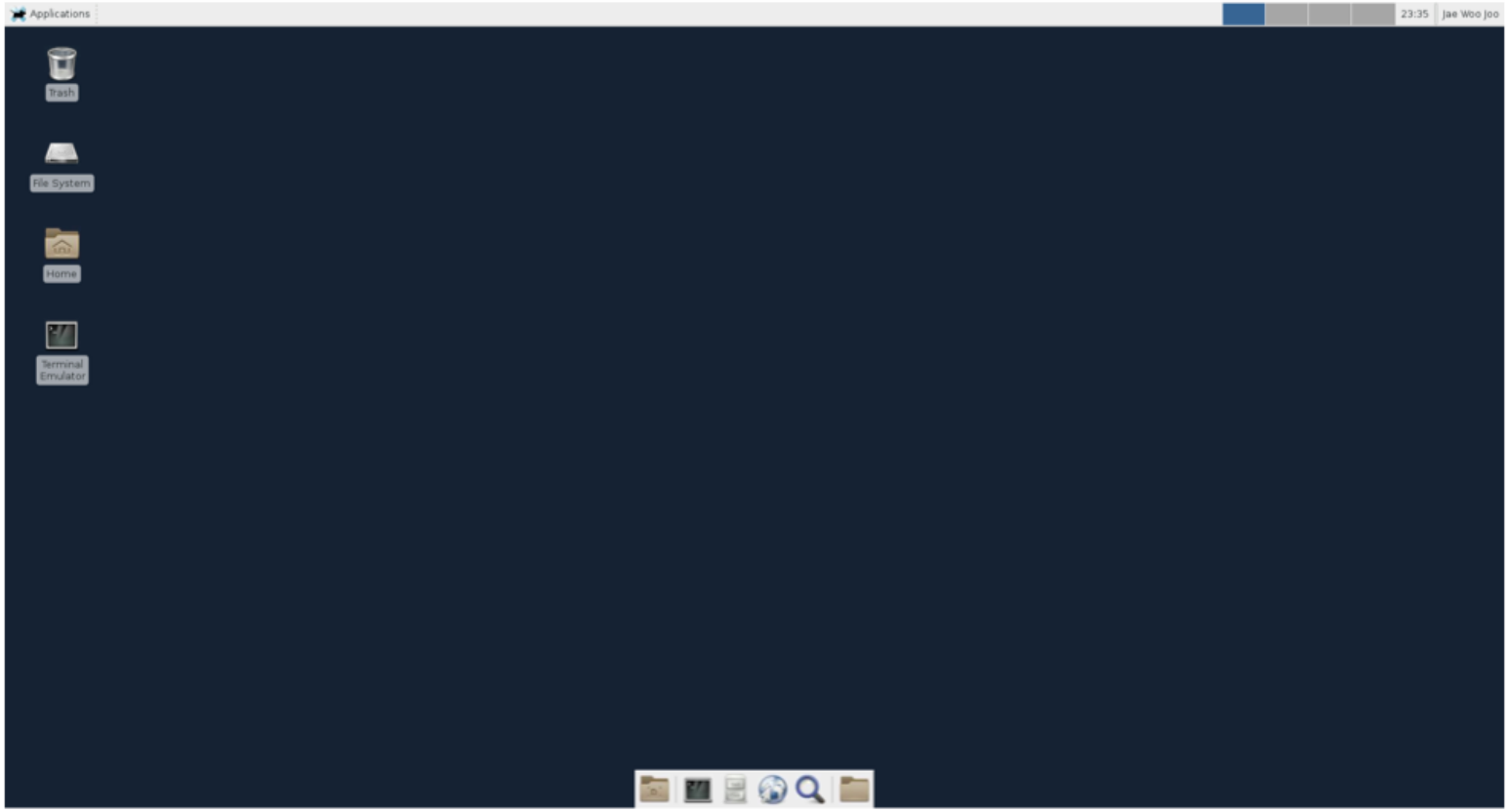
- Start session and login



Guide: https://resources.cs.rutgers.edu/docs/other/x2go/

Connecting to iLab Machines: X2GO Remote Desktop

- Start using Desktop Environment



Guide: https://resources.cs.rutgers.edu/docs/other/x2go/

Connecting to iLab Machines: Microsoft Remote Desktop (Windows/Mac/Linux)

- Necessary programs differ between systems
- Refer to guide linked below to see how to do it on your particular OS

Guide: https://resources.cs.rutgers.edu/docs/computer-systems/windows-remote-desktop/

Linux Shell Basics

- pwd : show name of current working directory
- cd : change directory
- cd .. : change to parent directory
- ls : list files
- mkdir : make a directory
- rmdir : remove a directory
- touch : create a file
- cat : print file contents
- mv : rename/move a file
- rm : remove a file
- grep : search texts in files

Special Directory Entries

- Every directory have two hidden directories:
  - “.” and “..”
- ”.” refers to the current directory
- “..” refers to the parent directory
- Examples
  - cd .. (navigate to parent directory)
- https://superuser.com/questions/37449/what-are-and-in-a-directory

Linux Text Editors

- Vi / Vim
- Nano
- Gedit
- Emacs
- ….
- Pick anyone you would like

Vim Tutorial: https://linuxconfig.org/vim-tutorial

Nano Beginners Guide: https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/

Transferring Files Between iLab and Local Machine

- Naïve way: Email files to yourself
- via scp (secure copy) on Mac/Linux
  - SCP tutorial: https://kb.iu.edu/d/agye
  - Examples:
    - (To iLab machine)
      scp localfile.txt djd240@ilab.cs.rutgers.edu:folder/file.txt
    - (From iLab machine)
      scp djd240@ilab.cs.rutgers.edu:folder/remotefile.txt file.txt
- via pscp (PuTTY secure copy) on Windows
  - pscp tutorial: https://www.ssh.com/ssh/putty/putty-manuals/0.68/Chapter5.html
  - Examples:
    - (To iLab machine)
      pscp localfile.txt djd240@ilab.cs.rutgers.edu:folder/file.txt
    - (From iLab machine)
      pscp djd240@ilab.cs.rutgers.edu:folder/remotefile.txt file.txt
- via SFTP client
  - drop and drag files between machines via GUI
  - WINSCP (Windows) https://winscp.net/eng/index.php
  - FileZilla Client (Windows/Mac/Linux) https://filezilla-project.org/