Welcome!
CS 223: Systems Programming

Instructor: Aline Normoyle

TA: Rebecca Lassman

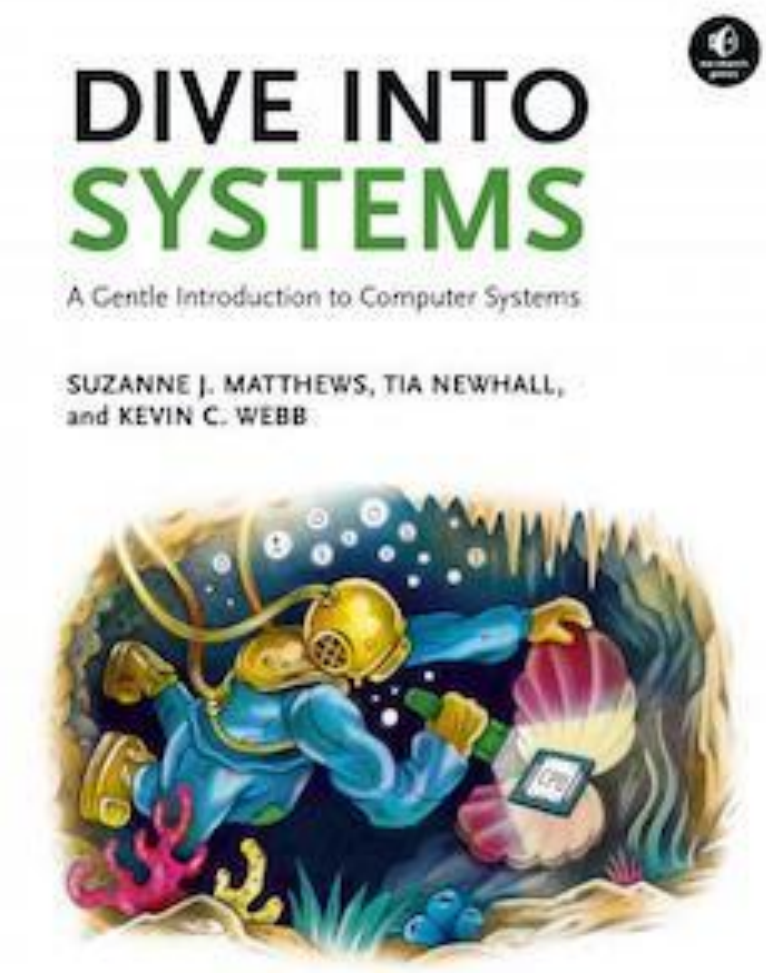
Textbook: Dive into Systems

Slack: Announcements, links, etc

Website: Policies, syllabus, etc

Github: Code repository

Lab: Park 231



Course Resources

Webpage

https://brynmawr-cs223-s26.github.io/website/

Github

https://github.com/BrynMawr-CS223-S26/

Slack

https://BrynMawr-CS223-S26.slack.com

Agenda

What is a computer system?

Development environment overview

C and Java

Makefiles

Unix review: bash and working with paths

Editors

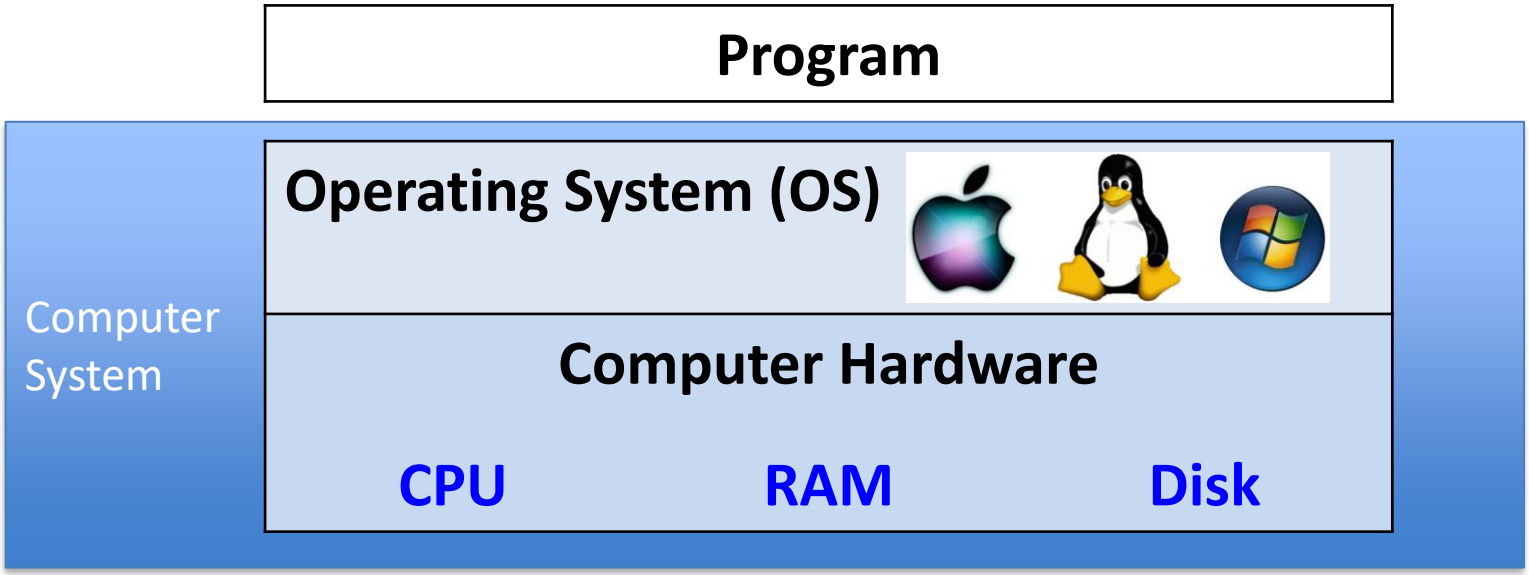
Administrivia

What is a computer system?

Hardware & Special Systems Software (OS)

Work together to run application programs

- HW executes program instructions
- OS implements interface to user/program and manages the computer HW



What is systems programming?

- Programs that interact with hardware
  - drivers, hardware management, embedded systems
- Programming of tools intended for use by *other* programmers or software
  - Compilers, operating systems, game engines, inter-process communication
- Typically performance or memory critical
  - Small optimizations have large payoff
  - Software runs in resource constrained environment

What you will learn

- C programming fundamentals
- Memory management
- Debugging Tools
- Data representations under-the-hood
- Neuman architecture, Assembly
- Core operating systems concepts
- Concurrent Programming
  - Processes
  - Threads
- Performance do’s and dont’s
- Working with UNIX, git, and terminal editors

Understanding binary representations and machine code

We will learn

- How to directly access and modify memory
- How data is stored in memory
- What low-level machine instructions look like and how to read it (x86_64 assembly language)

How hardware affects performance

The overhead associated with running programs

Q: why does my program run slowly?

A1: picked a bad algorithm (big O analysis)

A2: picked best algorithm, but program using system resources in inefficient way

Example: How a program uses memory can have huge effect on its performance

(ex) merge sort is O(nlogn) but it is not in-place: each merge pass requires moving elements from one list to another (requires 2n memory space)

(ex) program's access patterns and the Memory Hierarchy

Slides based on Tia Newhall, CS31, Swarthmore

Why study systems?

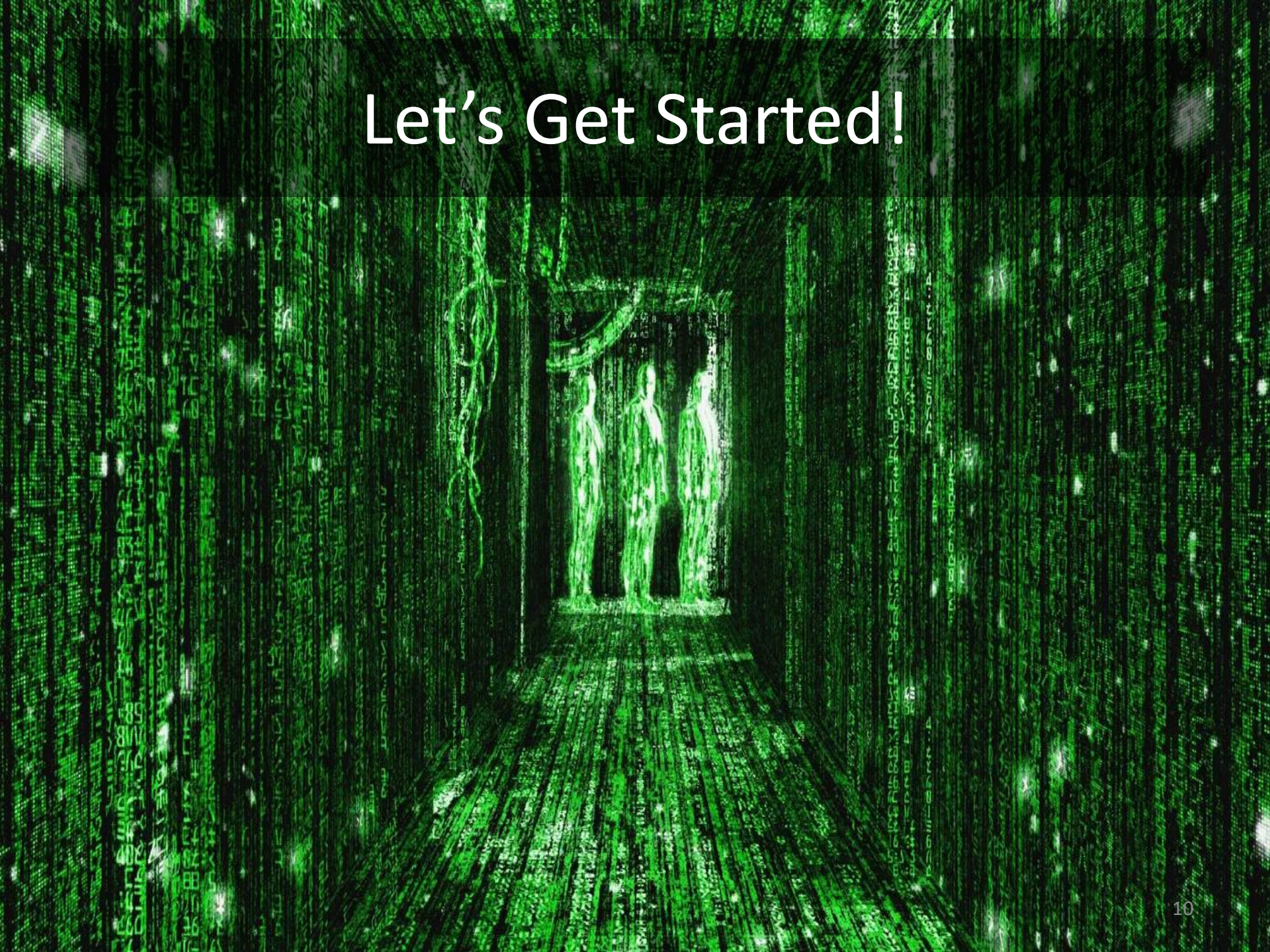
To understand the systems that form the basis of all modern computing

To become better software engineers and programmers

- write more efficient code
- understand the limitations of computing in terms of security, performance, energy, etc

Skills to design, build, and customize your own software and devices

Let’s Get Started!



Development Environment

A **development environment** consists of the platform and tools that you use to write software

Systems programmers need to be able to

- work from terminal using shell commands
- program in low-level languages
- use debugging and profiling tools

This class:

- Operating system: Ubuntu (Linux)
- Programming languages: C, x86_64 assembly language
- Editor: nano, vim, or emacs
- Makefiles for compiling and linking
- git for source control

C

- High-level programming language
  - Java, python, ruby, Javascript, C++, etc
  - Imperative (sequence of statements)
  - Procedural (structured using functions)
  - No classes, built-in types such as strings, lists

- Less abstracted than other languages
  - easier to see relationship between code and the computer's running of it
  - capable of more efficient code

From Java to C: Hello World

```
class Hello {
  public static void main(String[] args) {
    System.out.println(“Hello World”);
  }
}
```

To compile: javac hello.java
To run: java Hello

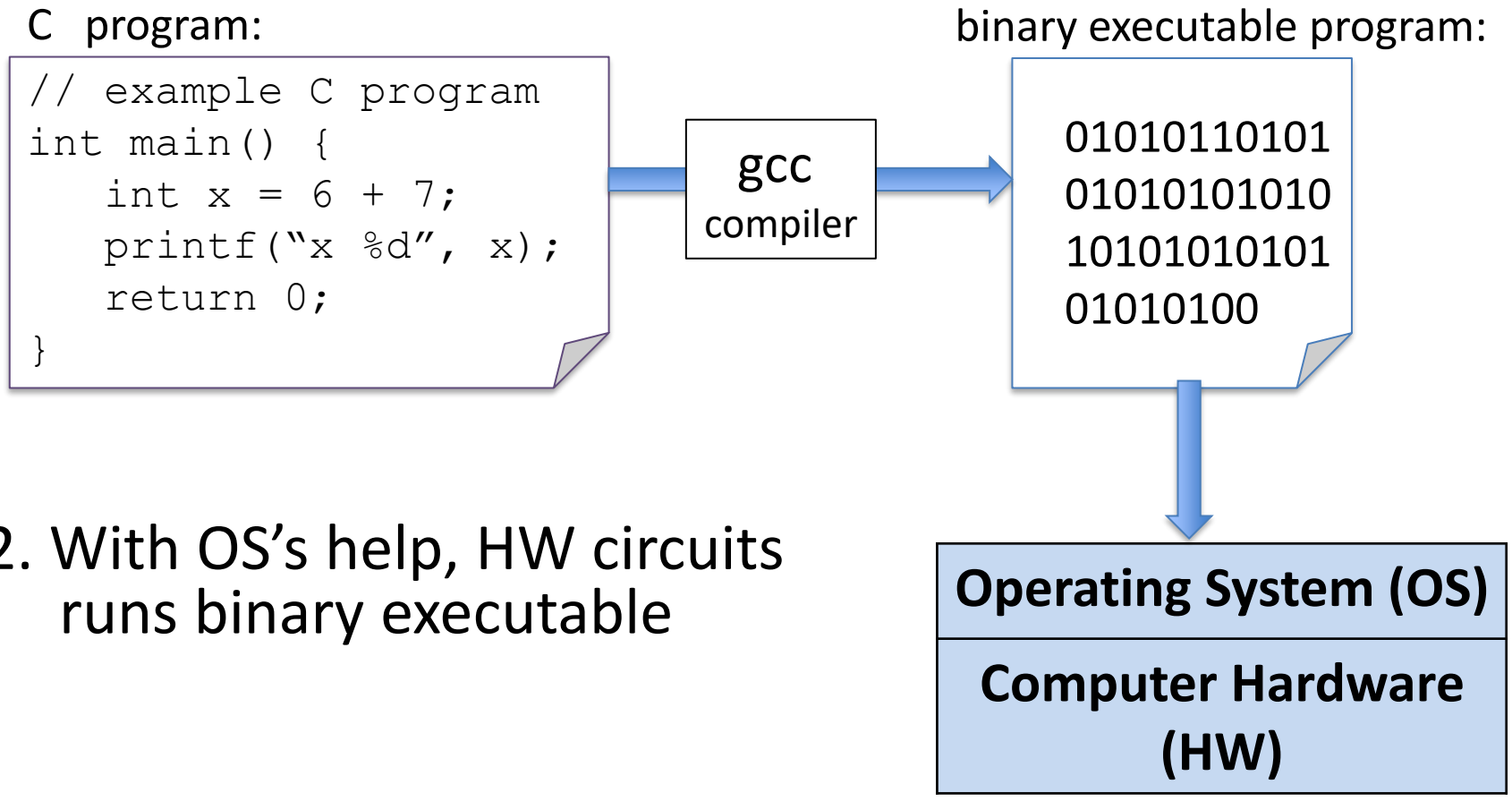
```
#include <stdio.h>

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

To compile: gcc hello.c
To run: ./a.out

Building and Running a C program

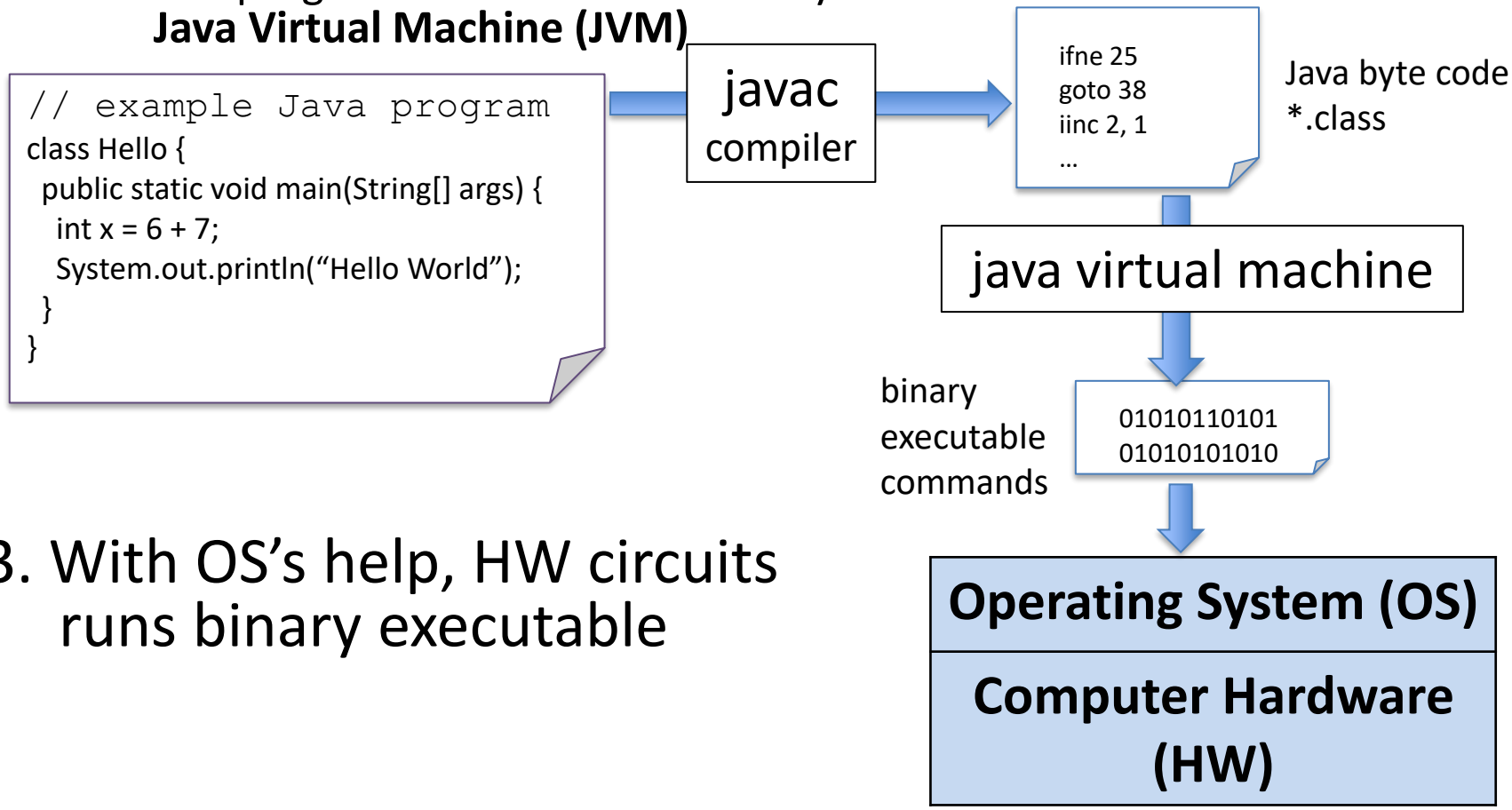
1. **Compiling** a C program translates it to binary (0's and 1's)
   - The binary file is an **executable**, meaning "we can run it"



2. With OS's help, HW circuits runs binary executable

Building and Running a Java program

1. **Compiling** (javac) a Java program translate it to Java byte code
2. **Running** (java) translates the program to binary (0's and 1's)
   - The program that translate from byte code to machine code is called the **Java Virtual Machine (JVM)**



3. With OS's help, HW circuits runs binary executable

Slide based on those by Tia Newhall, CS31, Swarthmore

All programs must eventually become binary (0's and 1's) to run on a computer

- The binary code is specific to the hardware
- Higher-level languages (e.g. Java) have more **layers of abstraction** between the programmer's code and the binary code
  - higher-level languages are **cross-platform**, e.g. the same program can run on different hardware
    - ex. Our C and Java programs run on mac, windows, and linux

Makefiles

Idea: Put all build commands into a file

```
$ nano Makefile
$ make hello
```

```
CC=gcc
% :: %.c
  $(CC) -g -Wall -Wvla -Werror -Wno-unused-variable $< -o $@

all: hello

clean :
  rm hello
```

Review: UNIX basics

Ubuntu Desktop has a window manager (lab machines) but we will mostly be using **command-line interfaces (CLI)**

terminal – text-based interface for the OS

command line – current line in the terminal; where we issue a command

command prompt – prefix text at the beginning of the command line

shell – program that executes commands from terminal

- **bash –** the shell we will use in this class!
- **zsh** – mac shell
- **powershell** – windows shell

Exercise: Connect to a server

On a laptop or home desktop computer, open a terminal and ssh to comet

```
$ ssh <username>@comet.cs.brynmawr.edu
```

Exercise: Edit a file

Write and compile a program, `hello.c`, that prints “Hello World”

```
$ nano hello.c

$ gcc hello.c
$ ./a.out

$ gcc hello.c –o hello
$ ./hello
```

Reference: Some useful commands

- ls – list all directories
- cd, mkdir, mv, cp, rm – change directory, make directory, move, copy, remove
- cat, less, more – showing files
- javac, gcc, make – compiling programs
- vi, nano, emacs – editing files
- grep, find – searching files
- man – read documentation (RTFM: “Read the fine manual”)
- ssh <username>@goldengate.cs.brynmawr.edu – log into CS server
- git – source control

Working with paths from terminal

- What are files? What are directories?

- path - full name of a file or directory that indicates the file/directory location within the file system
  - Absolute paths: path from the root of the file system to the file
  - Relative paths: path from **current working directory** to the file
- File extension: Tells the OS what type of data is in the file (ex: *.txt, *.jpg, etc)

Special directories

.. ← the parent directory (two dots)

. ← the current directory (one dot)

/ ← the root directory

/home/<username> ← your home directory

~ ← your home directory

Exercise

```
root
-- A
---- hello.txt
-- B
```

What is the absolute path of hello.txt?

What is the absolute path of hello.txt from the A directory?

What is the relative path of `hello.txt` from

- the root directory?
- the A directory?
- the B directory?

Exercise: Draw the directory hierarchy after the following commands

```
$ pwd
/home/alinen
$ mkdir A
$ cd A
$ mkdir Z
$ touch talk.c
$ cd ..
$ touch listen.c
$ cd
$ touch sing.c
```

Your editor and you!

You must learn a terminal editor this semester

- Nano
- Emacs
- Vim

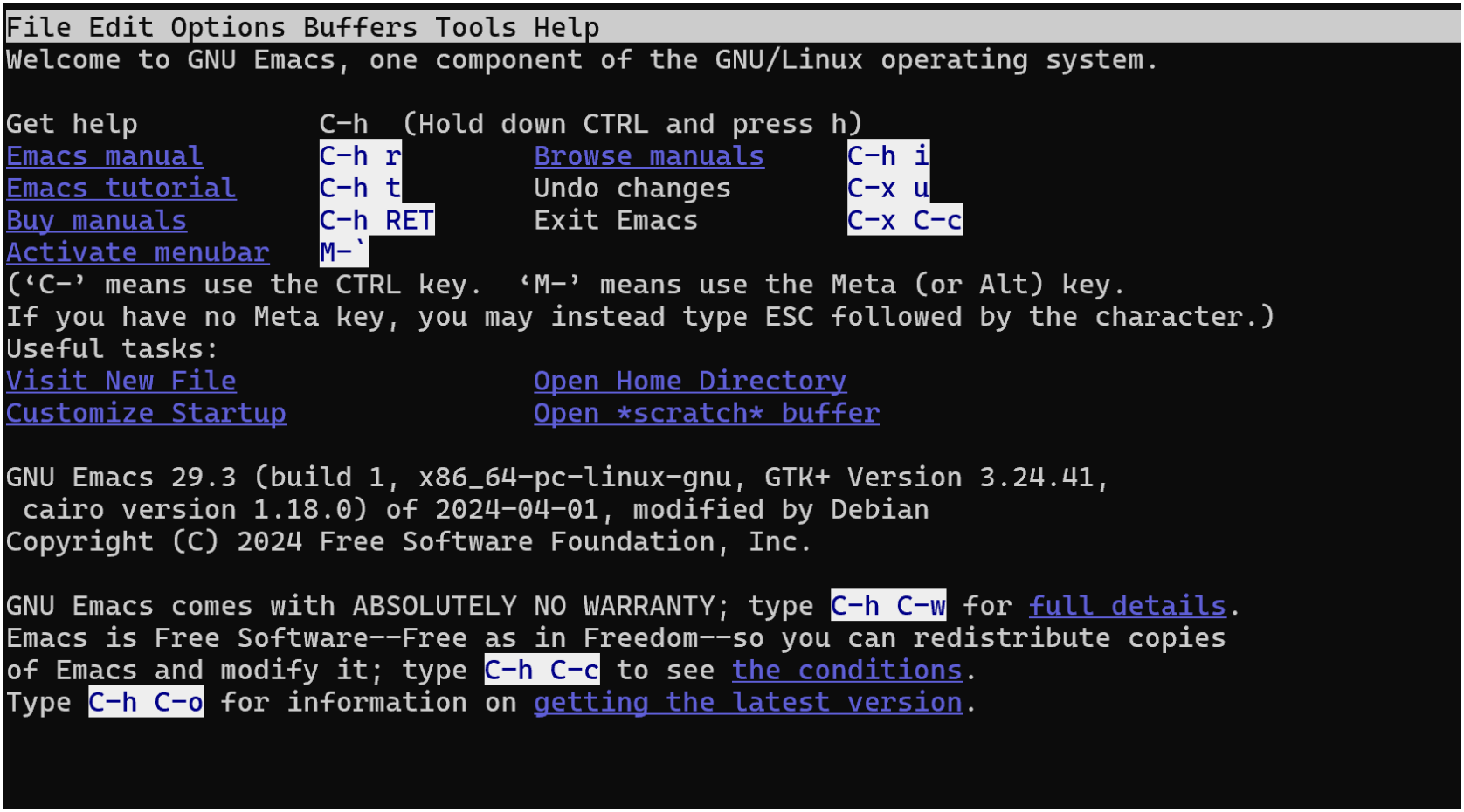
Learning a good editor will help you write code faster

You will need to use one of these editors for coding activities in lab

Nano


alinen@comet: ~
GNU nano 7.2 New Buffer *
Hello Nano
^G Help
^X Exit
^O Write Out
^R Read File
^W Where Is
^\ Replace
^K Cut
^U Paste
^T Execute
^J Justify
^C Location
^/ Go To Line

Emacs



NOTE: F10 to use the menu

Vim

- To open: `vi <filename>`
- To quit: Press escape, then `:q!`
- To save: Press escape, then `:w`
- Two modes: **insert** and **command mode**
  - insert mode: type text in the usual way: ‘i’ enters **insert mode** at current cursor position
  - Escape enters **command mode**: search, navigate, copy/paste/delete, etc

ADMINISTRIVIA

Lecture/Lab Format

- Lectures
  - Slides with integrated activities
  - Will share recordings after class
  - Quizzes (30 minutes, closed book, 1 cheat sheet)
- Labs
  - Coding Practice
  - Worksheets (in teams)
- Assignments
  - Weekly (due on Fridays)

Policies

- Accommodations
  - Need at least 2 weeks prior notice for extensions on quizzes/exams
- Covid policy: mask friendly
- Late policy: up to 1 day late

Programming Assignments

- Submissions MUST compile using **make** on our UNIX systems
  - Test on our servers to check your work
  - Do not change basecode!
- Full credit submissions must also
  - Follow the class coding style: **especially consistent indentation**
    - You may need to configure your editor to ensure this works correctly and we will help you with that!
  - not have memory errors (leaks, corruption)
    - Run using valgrind to test
- Assignment 01 has more information

How to succeed

- Read the textbook!

- Do the work each week
  - Approx 10 hour week commitment (4.5 hrs + 5 hrs)

- Attend lectures and labs
  - Lecture attendance is not mandatory
  - But better grades are correlated with attendance
  - Take hand-written notes

- Asking questions
  - Labs/Lectures/Office Hours are the best time
  - Reach out to me and TAs on Slack
  - Slack is great for questions. Responses within 24 hrs, Mon-Fri
  - Asking questions is a good way to network

Strategies

- Building focus
  - Work in silence on a specific task for a short period
    - Pomodoro Method
  - Turn off phone notifications, ring tones
  - Close browsers, mail, etc

- Building understanding
  - Alternate short focused periods of study with rest

- Building problem solving skills
  - Use assignments and quizzes as practice
  - Fix and understand errors
  - Start early

Strategies: Building programming skills

You're effectively using assignments to learn if you can:

- Explain how your code works to someone else, without looking at your code
- Explain the system calls necessary to complete your assignment without looking them up online
- Write a similar program within a few minutes, without looking up help online

What worked for me:

- Writing my own programs from scratch
- Checking my work by stepping line by line using a debugger (gdb)
- When stuck, debugging the program with the help of a more experienced developer
- Comparing my solutions with others *after I finished my own solution*

Lab this week

Checking your UNIX account

Signing into slack

Setting up SSH and Github

Pulling and pushing to your code repository for this course