

INTRODUCTION TO SOFTWARE DEVELOPMENT

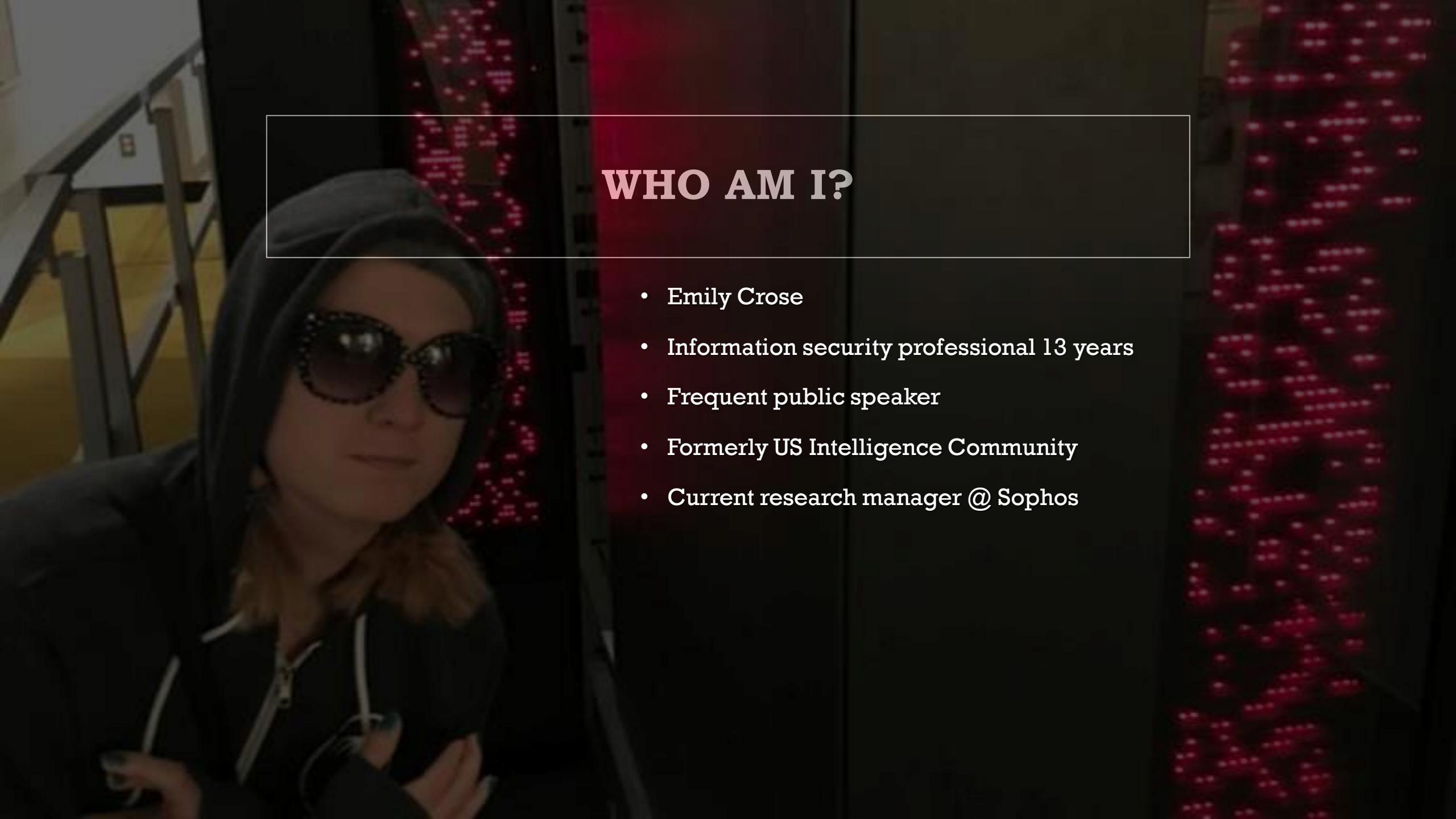
Emily Crose

For

Oakland University

FOR TODAY

- Introductions
- Tools for the class
- Overview of class material
- Hardware & Software introduction

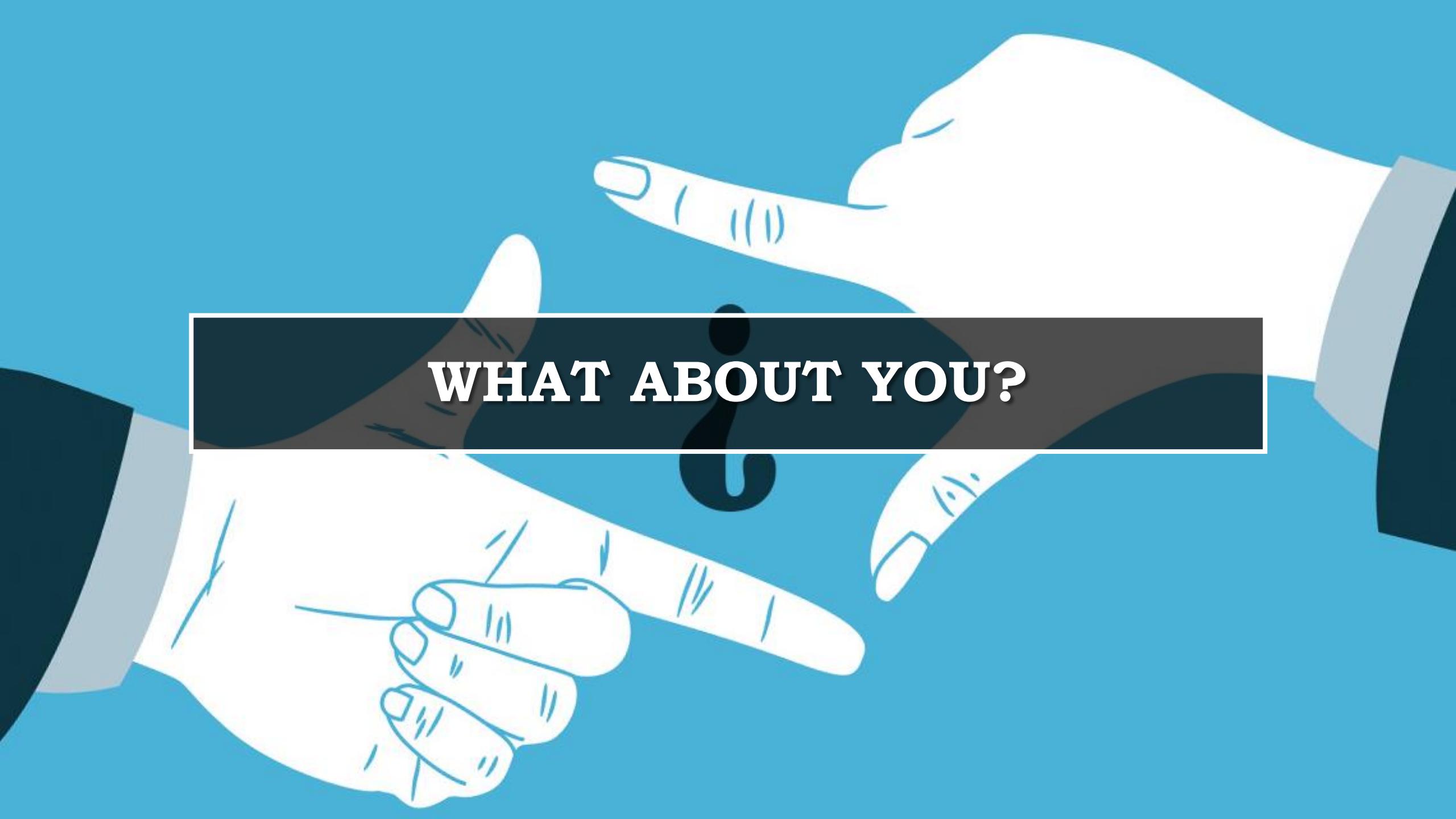


WHO AM I?

- Emily Crose
- Information security professional 13 years
- Frequent public speaker
- Formerly US Intelligence Community
- Current research manager @ Sophos

CONTACT INFO

ecrose@oakland.edu



WHAT ABOUT YOU?

WHEN WE MEET

- Meet 4 times/week
- 3 hours/day
- 4 weeks
- Mondays, Tuesdays, Wednesdays & Thursdays
 - 7AM Eastern – 10AM Eastern

TOOLS WE WILL USE

- Moodle
 - Class materials
- Github
 - Code repositories
 - Change management
- Jira
 - ticketing
- Confluence
 - For wikis
- More?



COURSE OVERVIEW

What will we be talking about in this course?

What do you want to learn?

MATERIAL OVERVIEW



**Understanding
modern software**

How it works

How it is maintained



**The Software
Development Life
Cycle (SDLC)**



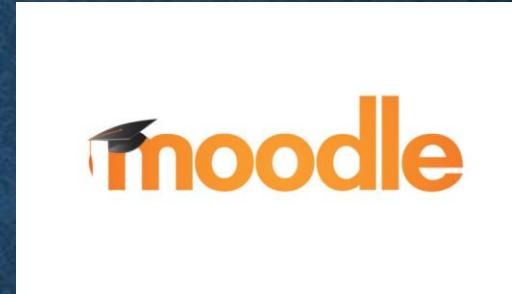
**Tools for managing
software
development**



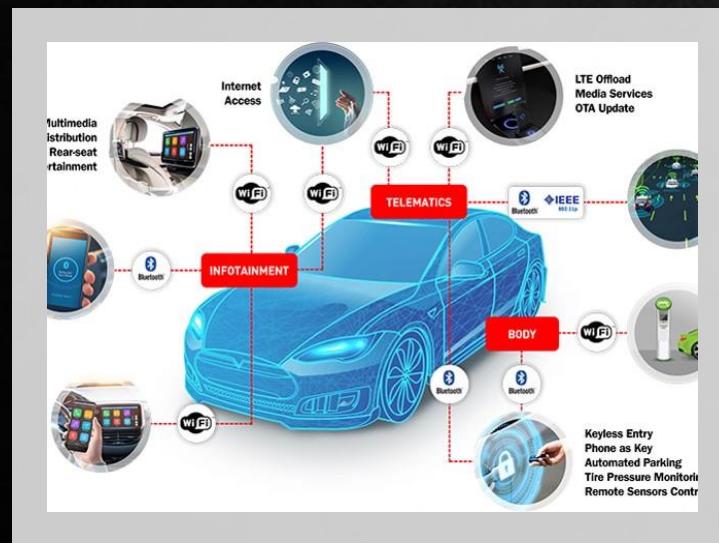
**Management
techniques for
maintaining
codebases**



**Standards that apply
to these topics**

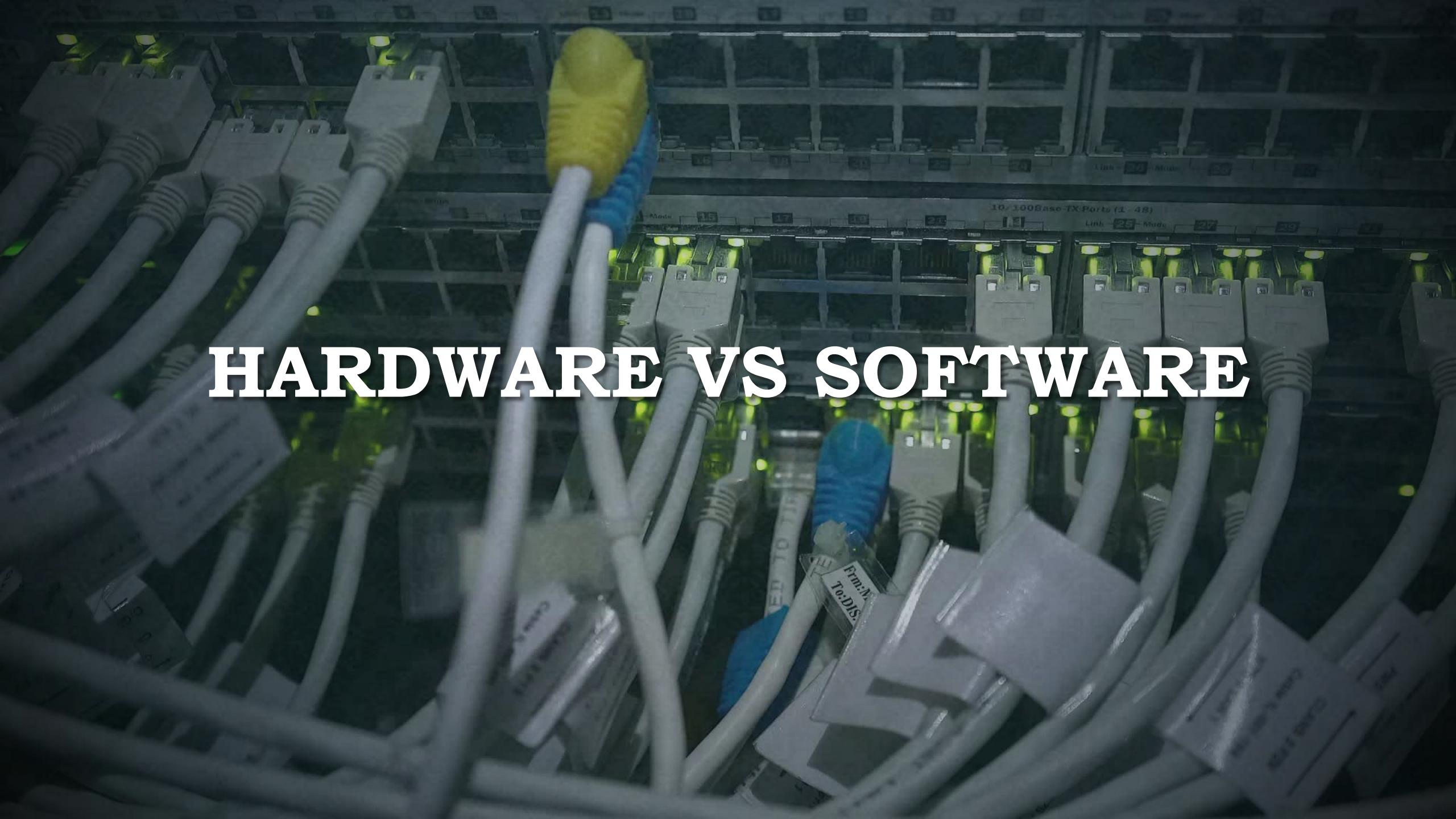


SOFTWARE WE CAN SEE



HIDDEN SOFTWARE

HARDWARE VS SOFTWARE

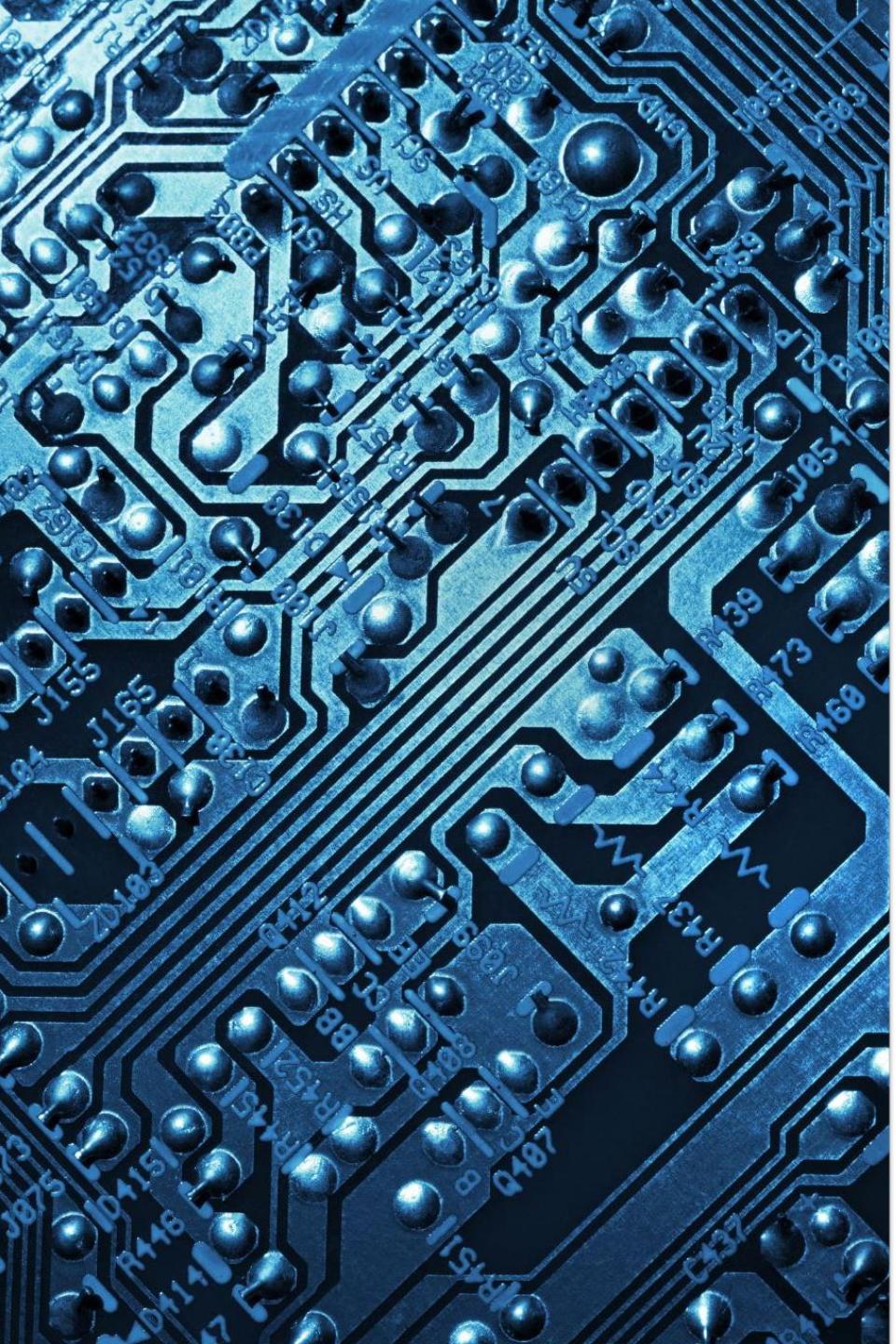


HARDWARE COMPONENTS

- Central Processing Unit (CPU)
- Graphics Processing Unit (GPU)
- Random Access Memory (RAM)
- I/O devices
 - Keyboard
 - Mouse
 - VR Headset

HARDWARE COMPONENTS CONT'

- Sensors
 - Temperature
 - Pressure
 - gyroscope



PURPOSE OF HARDWARE

- The physical half of computing (As opposed to logical)
- Does physical calculations
- Produces signals which are interpreted by software

PURPOSE OF SOFTWARE

- Provides interface between hardware and the user
- Software provides a “logical” layer of interpretation

```
mirror_mod = modifier_obj
# set mirror object to mirror
mirror_mod.mirror_object

if operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

# selection at the end - add
# mirror_mod.select= 1
# mirror_ob.select=1
context.scene.objects.active = eval("Selected" + str(modifier))
mirror_ob.select = 0
bpy.context.selected_objects.append(mirror_ob)
data.objects[one.name].select = 1

print("please select exactly one object")
print("press enter to continue")

# --- OPERATOR CLASSES ---
# --- OPERATOR CLASSES ---

@types.Operator:
def execute(self, context):
    if len(context.selected_objects) != 1:
        print("X mirror to the selected object.mirror_mirror_x")
        return {'FINISHED'}
    else:
        print("X")
        return {'FINISHED'}
```



MODERN SOFTWARE

OLD THINGS

- Old languages
 - Javascript
 - C
 - C++
 - Assembly
- Desktop software
- Mainframes?

OPERATING SYSTEMS

Windows

- 10
- 11
- Embedded

Mac OS

iOS

Linux

- Various flavors



10 MINUTE BREAK!

NEW THINGS

- Mobile & IoT
- “THE CLOUD”
- AI programming assistance

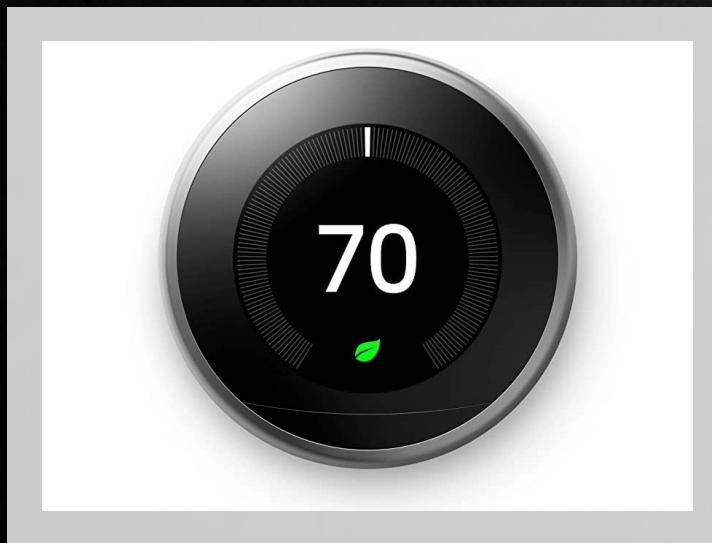
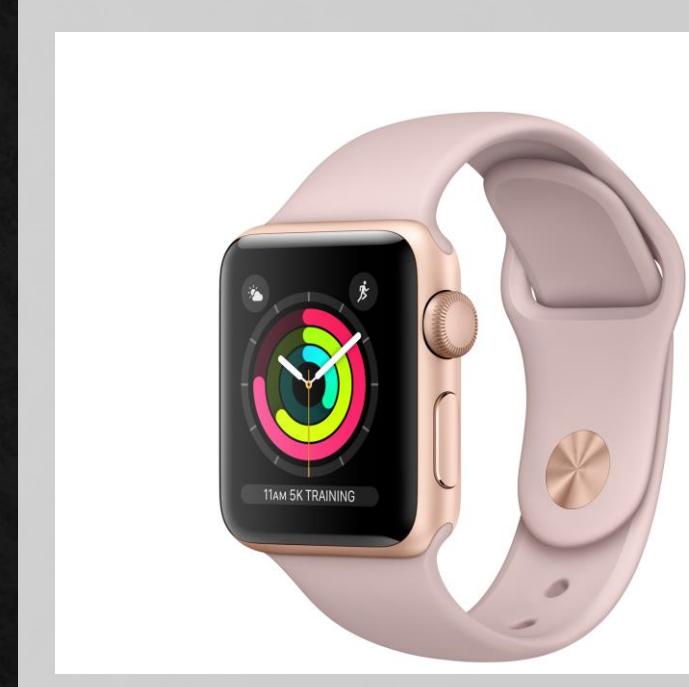


WHAT IS THE CLOUD?

Have we seen this before?

POPULAR CLOUD VENDORS

- Azure (Microsoft)
- Amazon Web Services (AWS)
- Google cloud computing
- Resellers



WHAT IS MOBILE/IOT?

FACTORY TECH



hardware + software!

IMPORTANT TERMINOLOGY

HARDWARE TERMINOLOGY

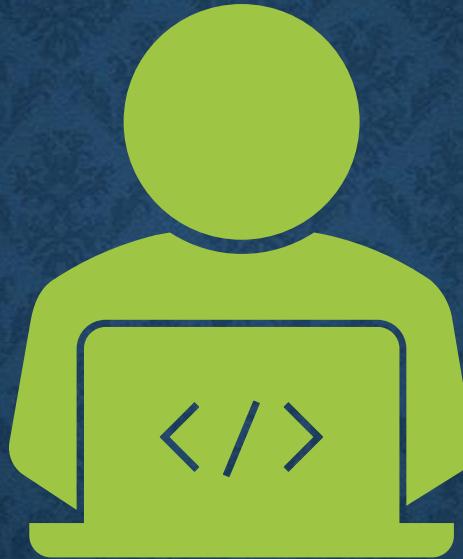
- CPU/processor
 - Architecture
- Server
- Infrastructure

SOFTWARE TERMINOLOGY

- Programming/coding
- Updates vs. Upgrades
- Code commenting
- Parsing
- Execution
- Environment
- Application Programming Interface (API)

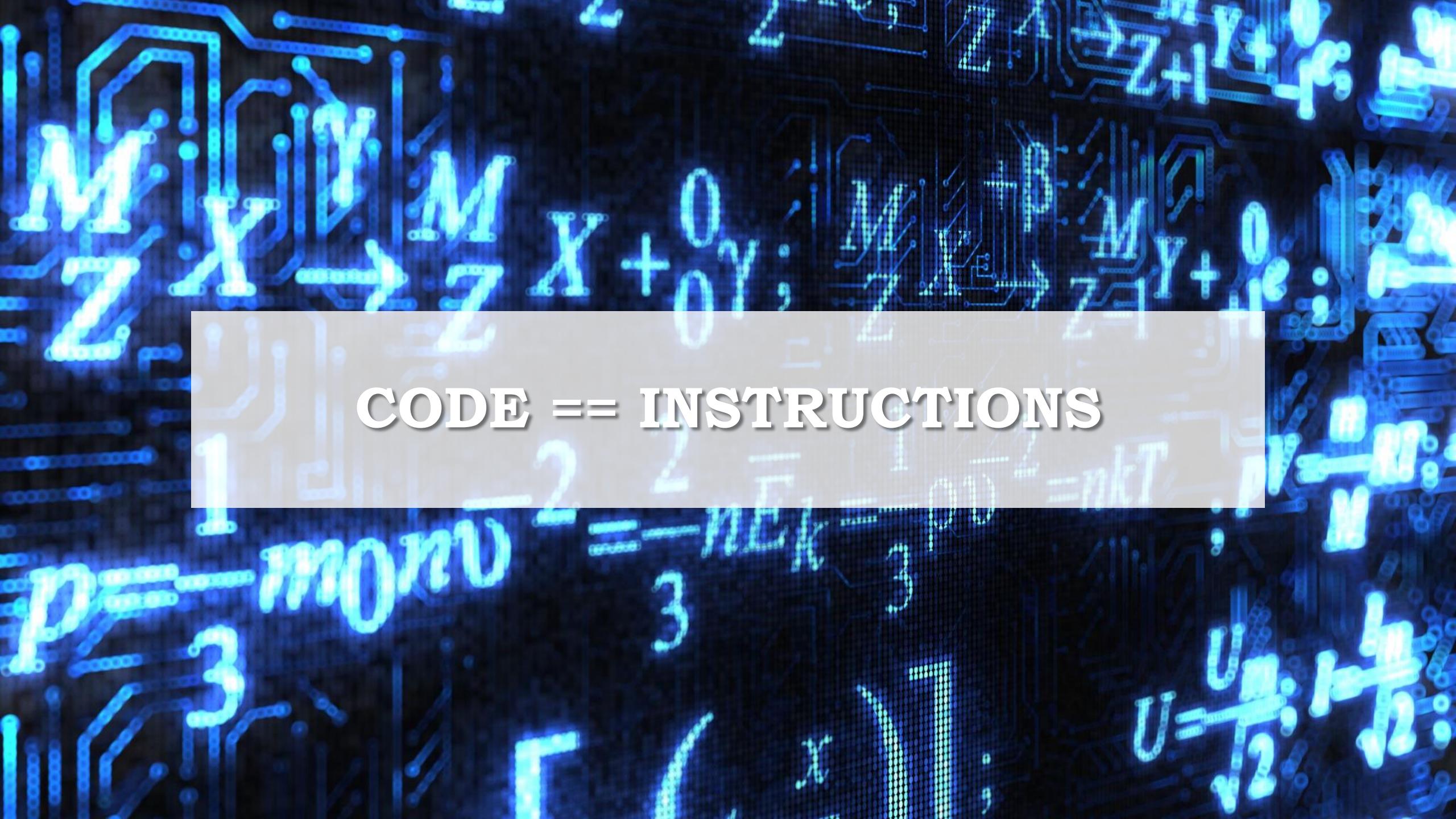
SECURITY TERMINOLOGY

- Vulnerability
- Exploit
- Payload



WHAT IS A PROGRAM *REALLY*?

PROGRAM == CODE



CODE == INSTRUCTIONS

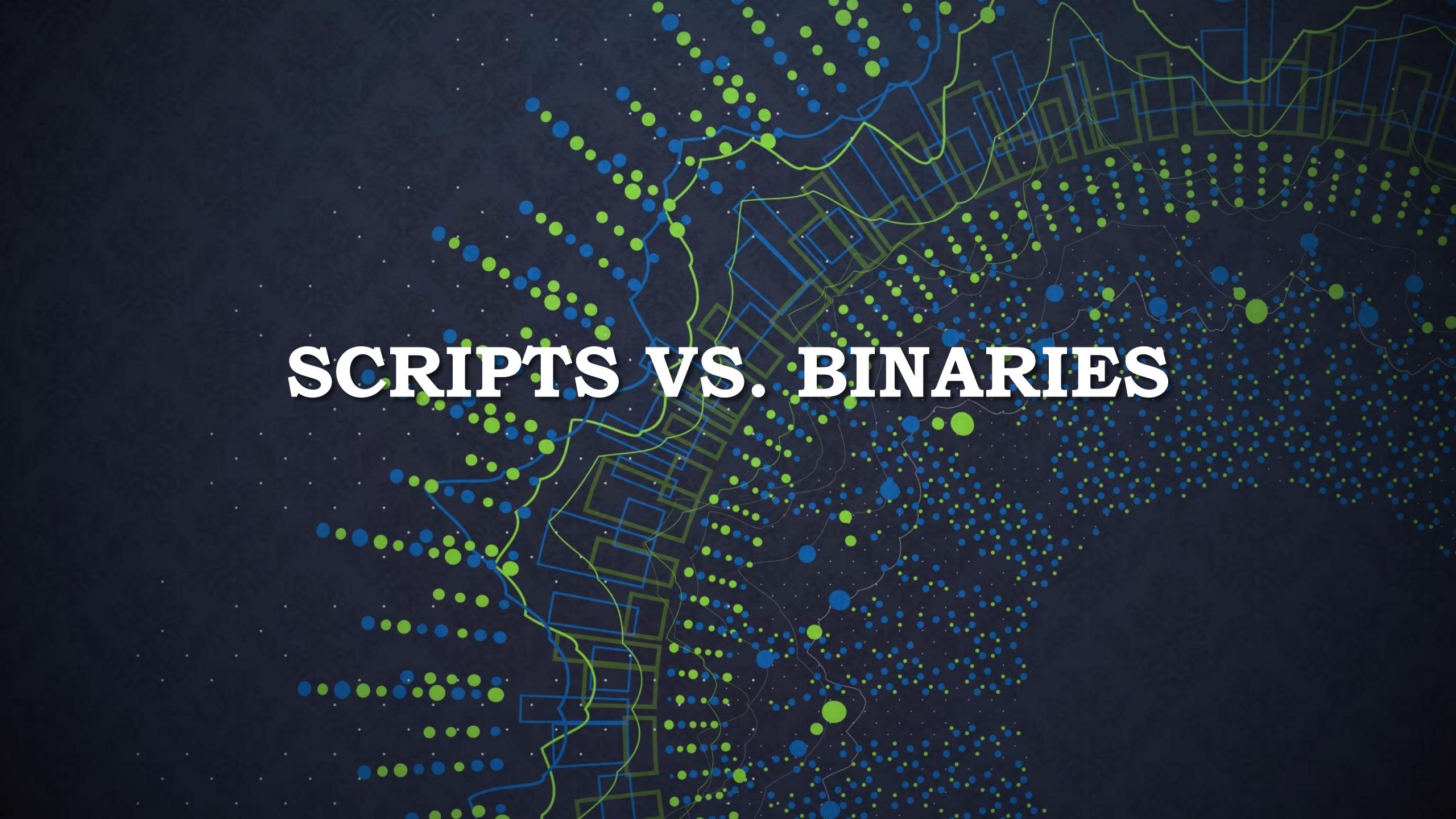
PARTS OF A PROGRAM

```
1  #include<stdio.h>                                Header Files
2  #include<conio.h>
3  int addNumbers(int a, int b); // function prototype ← Function Prototype
4
5  int main() ← Main Function
6 {
7     int n1,n2,sum; ← Variable Declaration
8
9
10    printf(" \n Enter First Number : ");
11    scanf("%d",&n1);
12    printf(" \n Enters Second Number : ");
13    scanf("%d",&n2);
14    sum = addNumbers(n1, n2); // function call ← User Defined Function Call
15
16    printf(" \n Sum of two number = %d",sum);
17    getch();
18    return 0;
19 }
20 int addNumbers(int a,int b) // function definition ← Function Declaration
21 {
22     int result;
23     result = a+b;
24     return result; // return statement
25 }
```

Diagram illustrating the parts of a C program:

- Header Files:** #include<stdio.h>, #include<conio.h>
- Function Prototype:** int addNumbers(int a, int b);
- Main Function:** int main()
- Variable Declaration:** int n1,n2,sum;
- Pre Defined Function Call:** printf(), scanf()
- User Defined Function Call:** addNumbers(n1, n2);
- Function Declaration:** int addNumbers(int a,int b)
- Function Body:** The block of code between the function definition and its corresponding closing brace.

SCRIPTS VS. BINARIES

The background of the slide features a dark navy blue gradient with a subtle texture. Overlaid on this are numerous abstract elements: a dense field of small, semi-transparent blue and light green circular dots; several thick, wavy lines in the same color palette; and a series of larger, overlapping rectangles in a darker shade of blue-green. These geometric shapes are scattered across the right side of the slide, creating a sense of digital data or network activity.

COMPILING



What is compiling?



A set of instructions that has been compiled becomes an executable 'binary'

We may just call this an 'executable'



Scripts do not compile!

Compiled Language

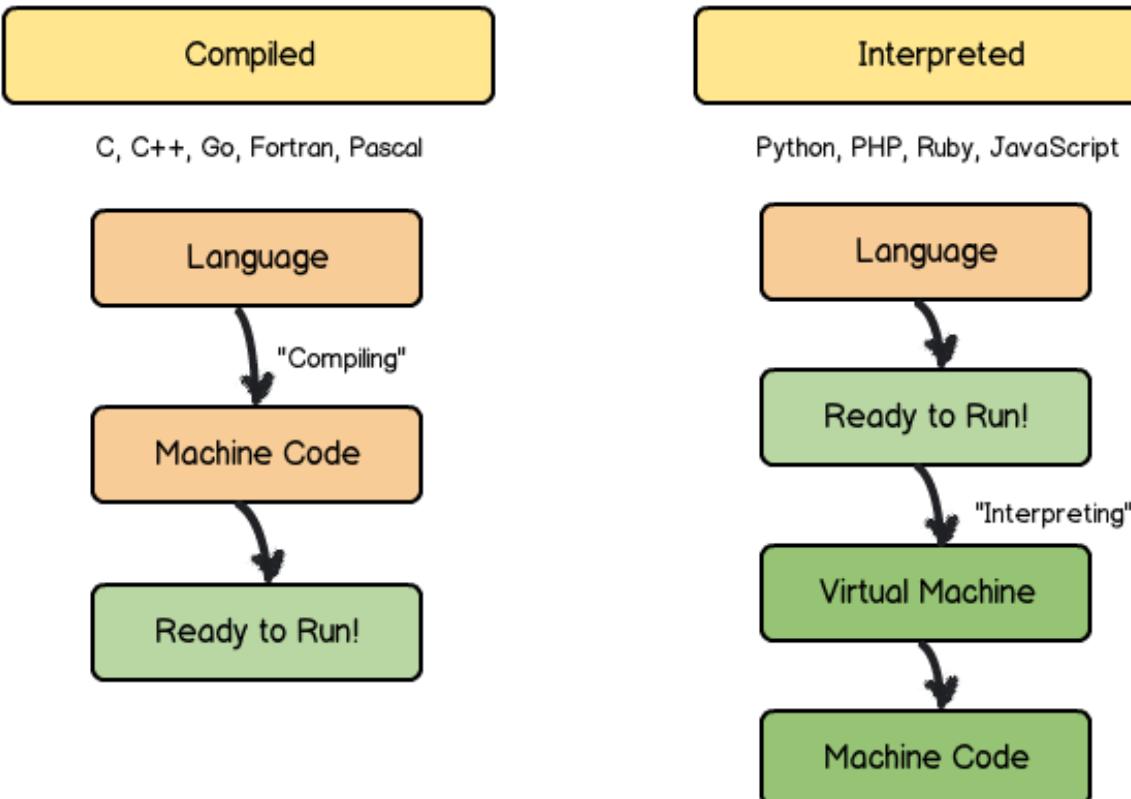
VS

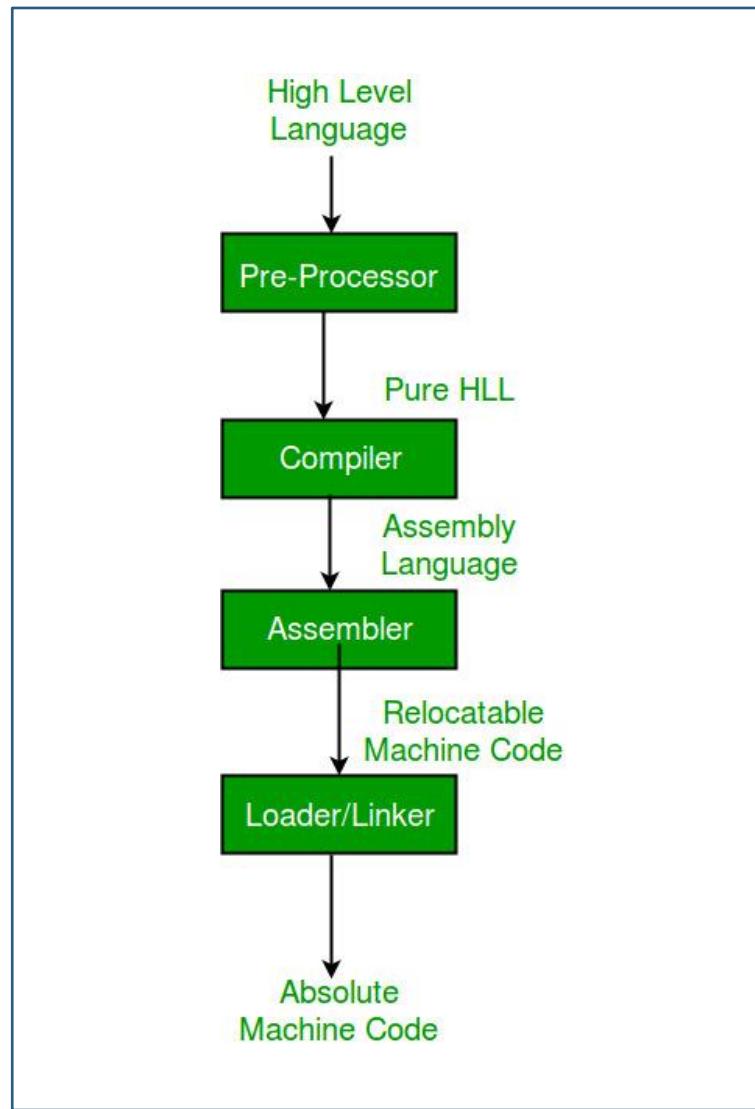
Interpreted Language

Comparison Chart

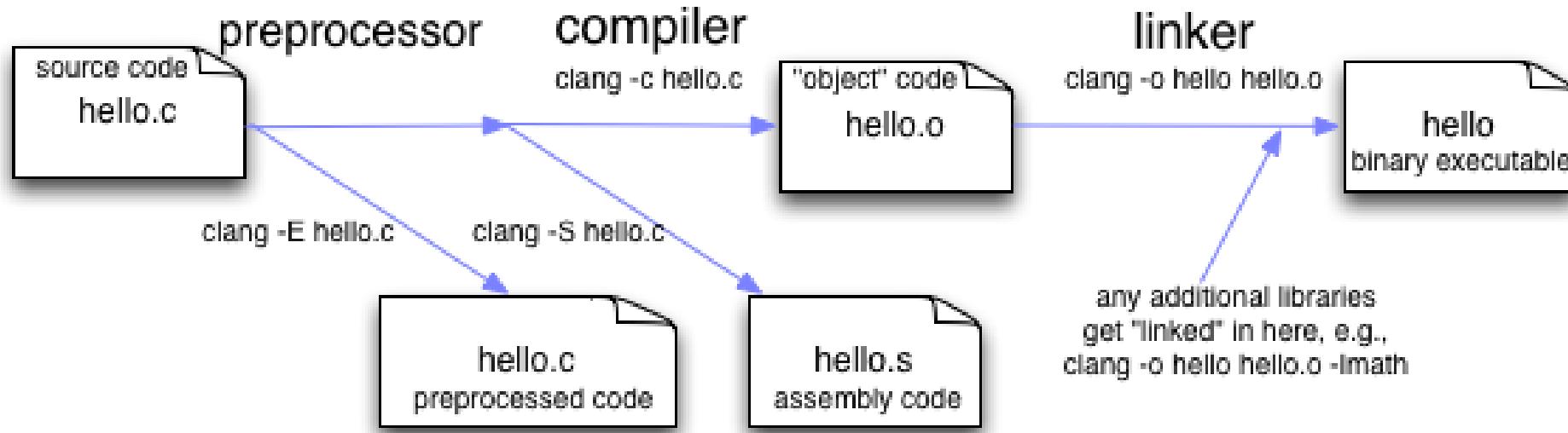
Compiled Language	Interpreted Language
The code of compiled languages can be executed directly by the computer's CPU.	A program written in an interpreted language is not compiled, it is interpreted.
The source code must be transformed into machine readable instructions prior to execution.	It does not compile the source code into machine language prior to running the program.
Compiled programs run faster than interpreted programs.	Interpreted programs can be modified while the program is running.
Delivers better performance.	Delivers relatively slower performance.
C, Fortran, and COBOL are languages used to produce compiled programs.	Java and C# are compiled into bytecode, the virtual interpreted language.

COMPILED VS INTERPRETED SIMPLIFIED

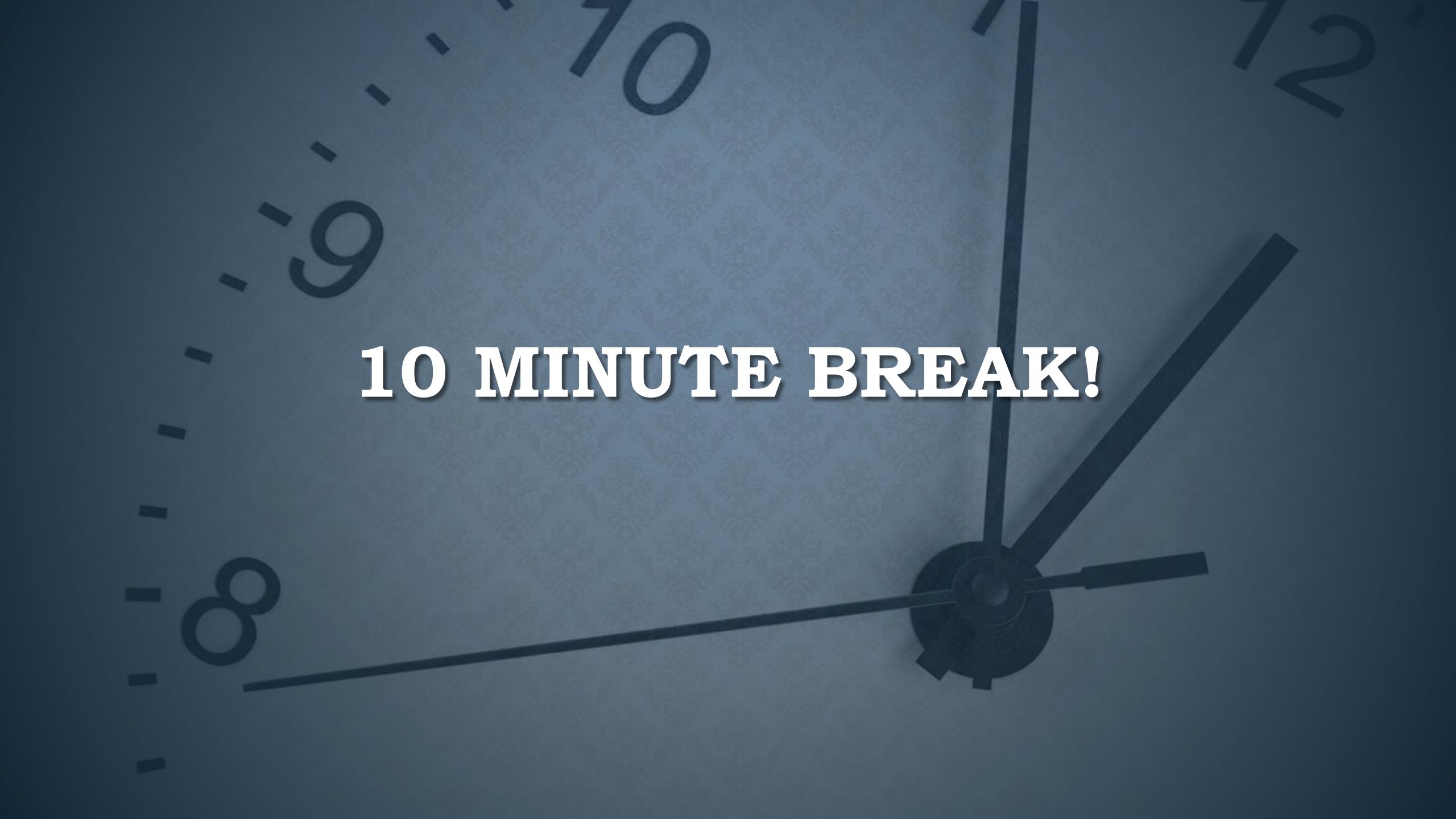




COMPILE PROCESS



COMPILING C IN PRACTICE



10 MINUTE BREAK!

SOFTWARE SNAFUS

- What can go wrong if we don't get software right?
 - Program crashes
 - System stability compromised
 - Exploitation
 - Network breach
 - Theft of IP/customer data
 - Loss of reputation
 - Risk of injury?





I SAT GeoStar 45
22-15 EAST 14 AUGUST 2003

MORE & MORE FAILS

- “Spaghetti” code
 - Difficult to add new features
- Poor documentation/code commenting
 - “Legacy” code maintenance

LET'S LOOK AT SOME CODE

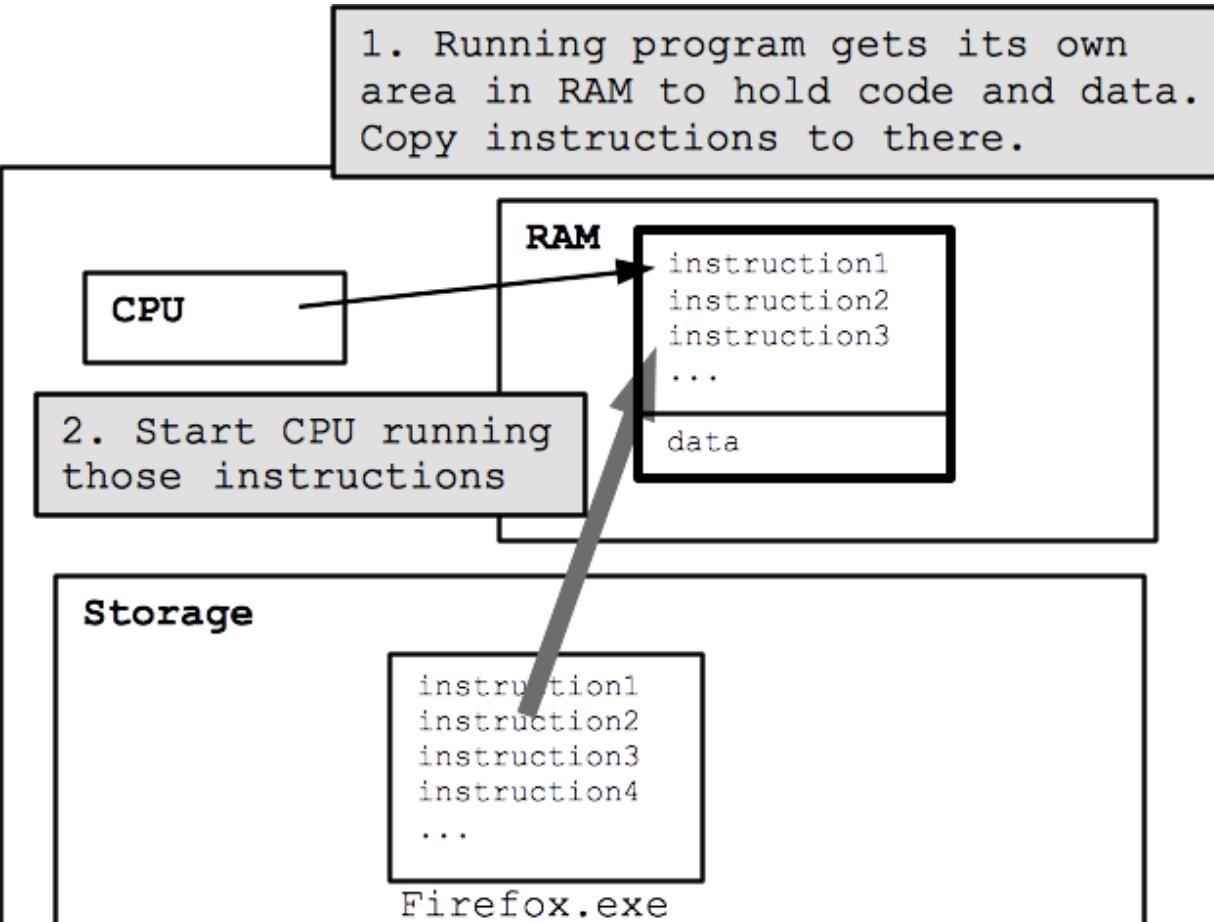
<https://github.com/hexadecim8/girltalk/blob/master/girltalk.sh>

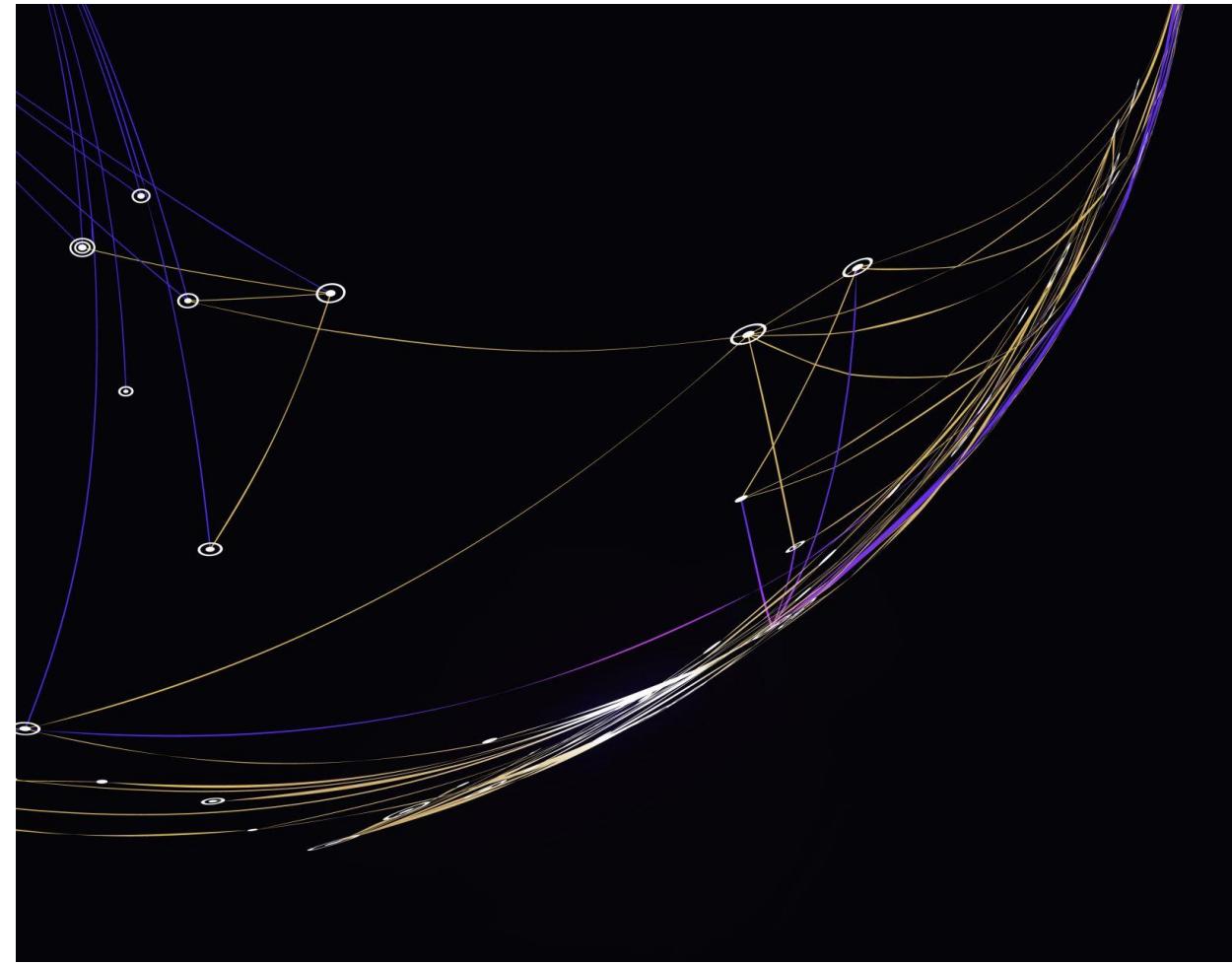




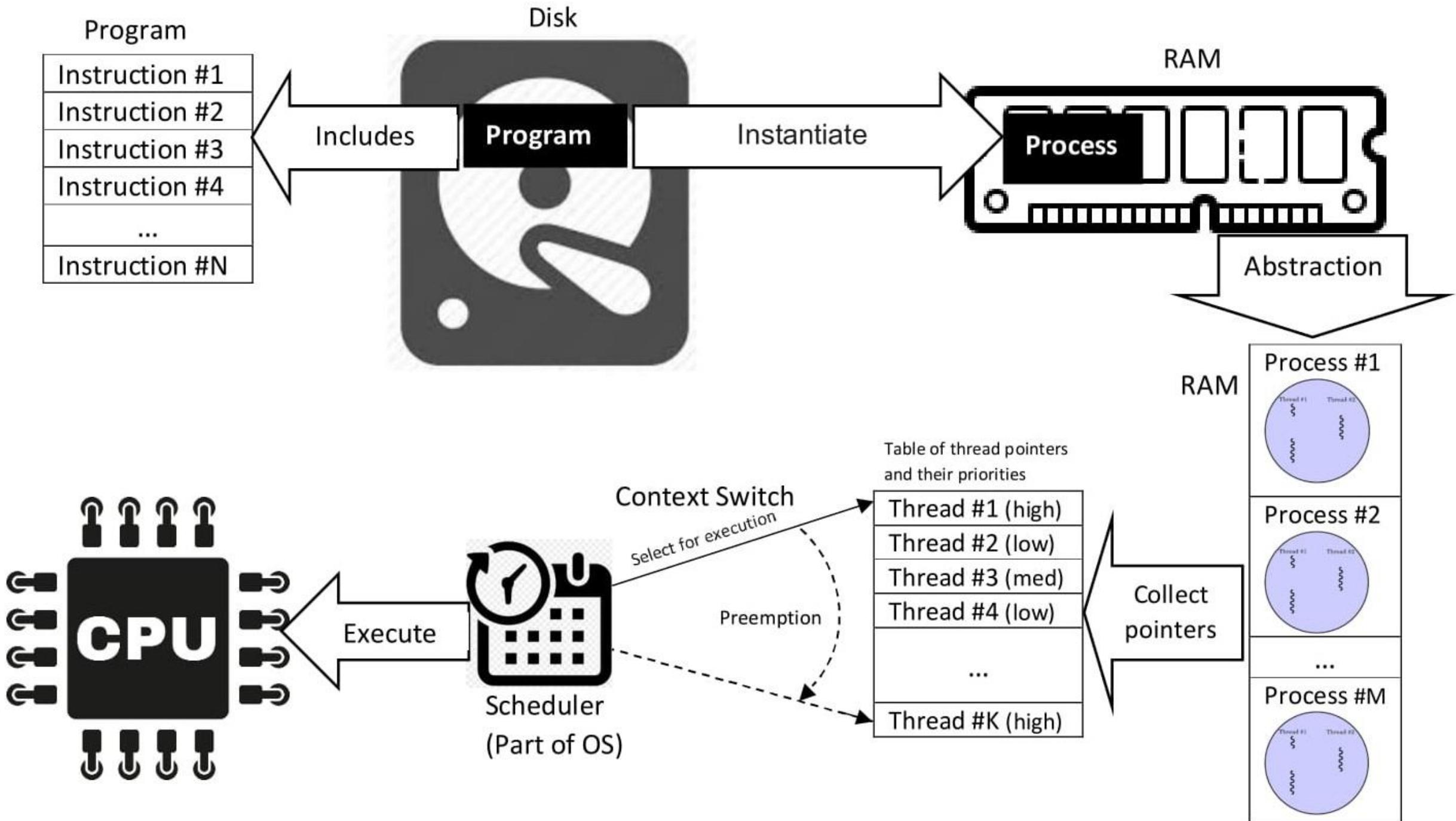
HOW A PROGRAM WORKS

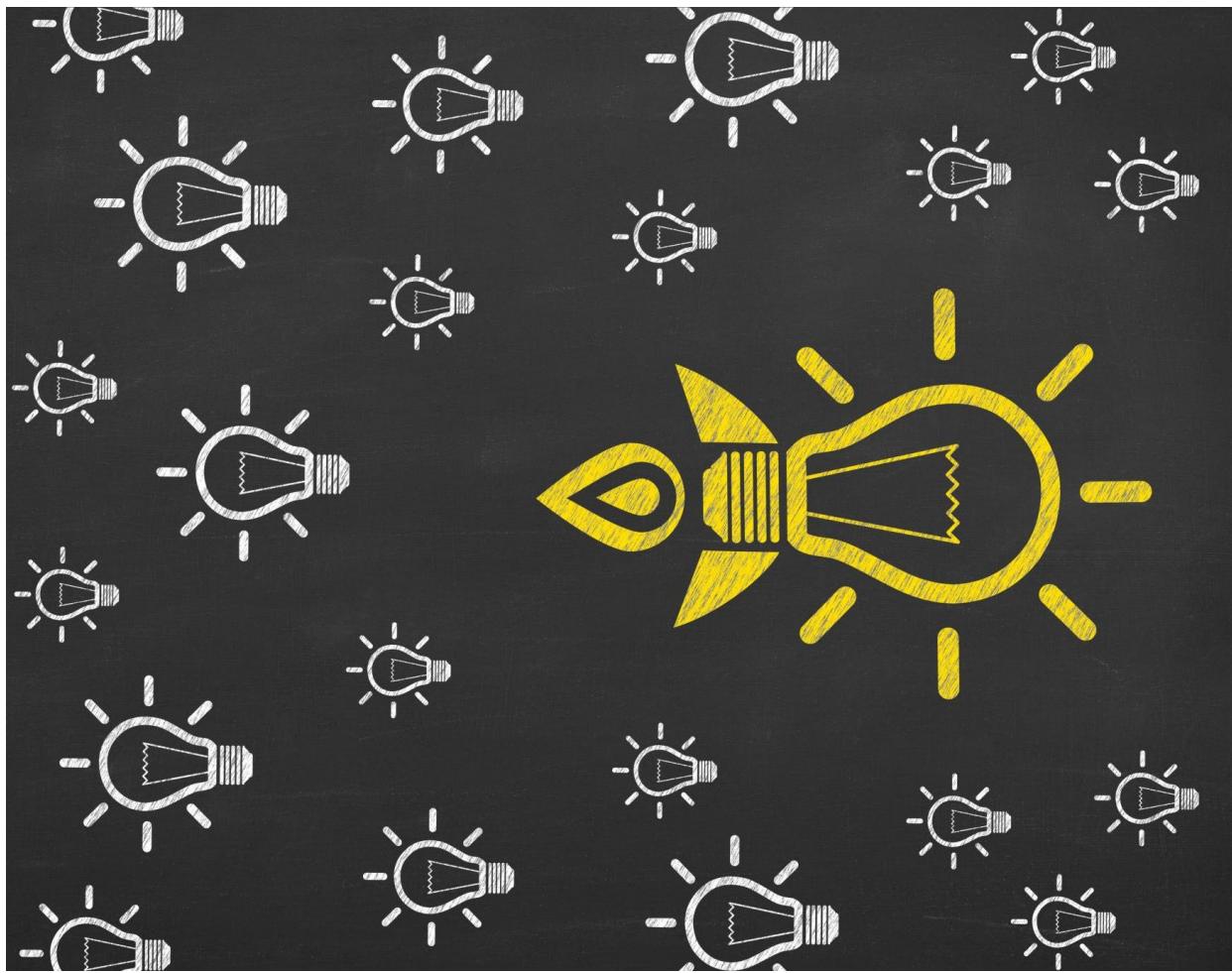
HOW FIREFOX WORKS





DETAILED EXECUTION





**IMAGINE AN
OUTSTANDING
APP**

TYING IT ALL TOGETHER

- Hardware is becoming much more diverse these days
 - Hardware can go places it didn't used to be able to
 - Software follows hardware
 - Making good software is complicated!
 - Managing these projects is complicated



QUESTIONS?

DAY 1 REVIEW

DAY 2 PREVIEW

CONTACT INFO

ecrose@oakland.edu