CS107, Lecture 26
Wrap-Up / What's Next?

😷 masks recommended

Plan For Today

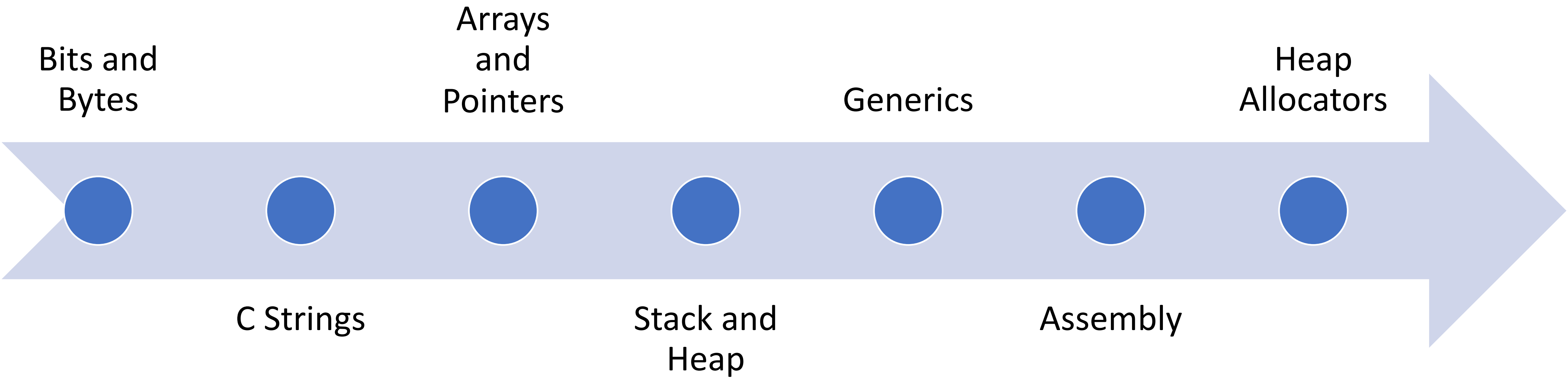
- **Recap**: Where We've Been
- Larger Applications
- What's Next?
- Q&A

Plan For Today

- **Recap: Where We've Been**
- Larger Applications
- What's Next?
- Q&A

We’ve covered a *lot* in just 10 weeks! Let’s take a look back.

Our CS107 Journey



Course Overview

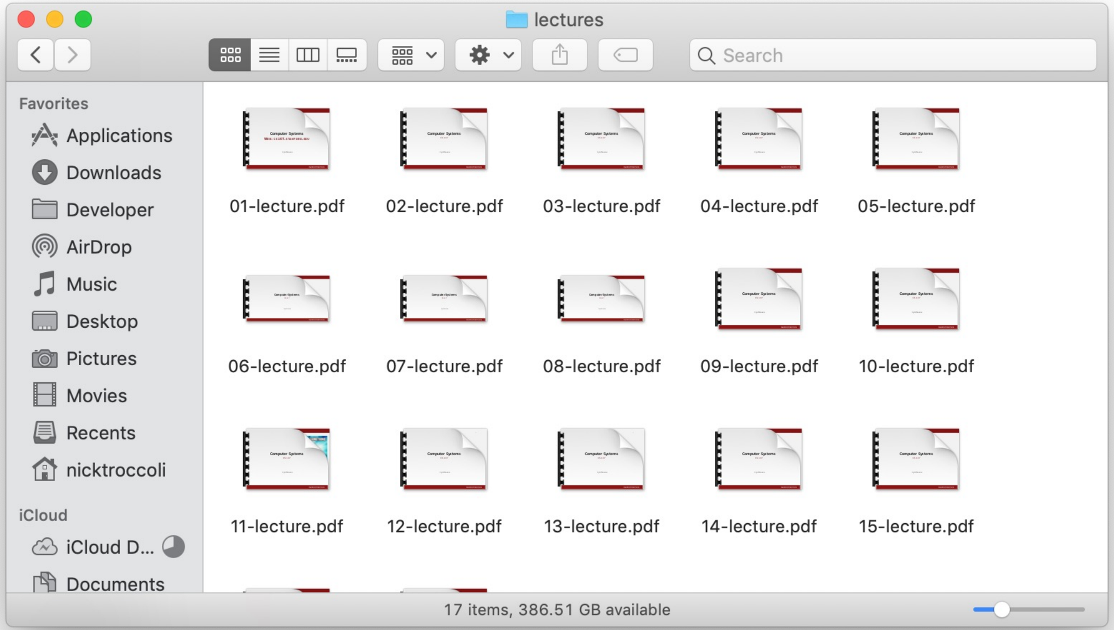
1. **Bits and Bytes -** *How can a computer represent integer numbers?*
2. **Chars and C-Strings -** *How can a computer represent and manipulate more complex data like text?*
3. **Pointers, Stack and Heap –** *How can we effectively manage all types of memory in our programs?*
4. **Generics -** *How can we use our knowledge of memory and data representation to write code that works with any data type?*
5. **Assembly -** *How does a computer interpret and execute C programs?*
6. **Heap Allocators -** *How do core memory-allocation operations like malloc and free work?*
7. **Ethics, Privacy, Partiality and Trust -** *How do we act responsibly in maintaining security, protecting privacy, and ensuring warranted trust in the systems we build and maintain?*

First Day

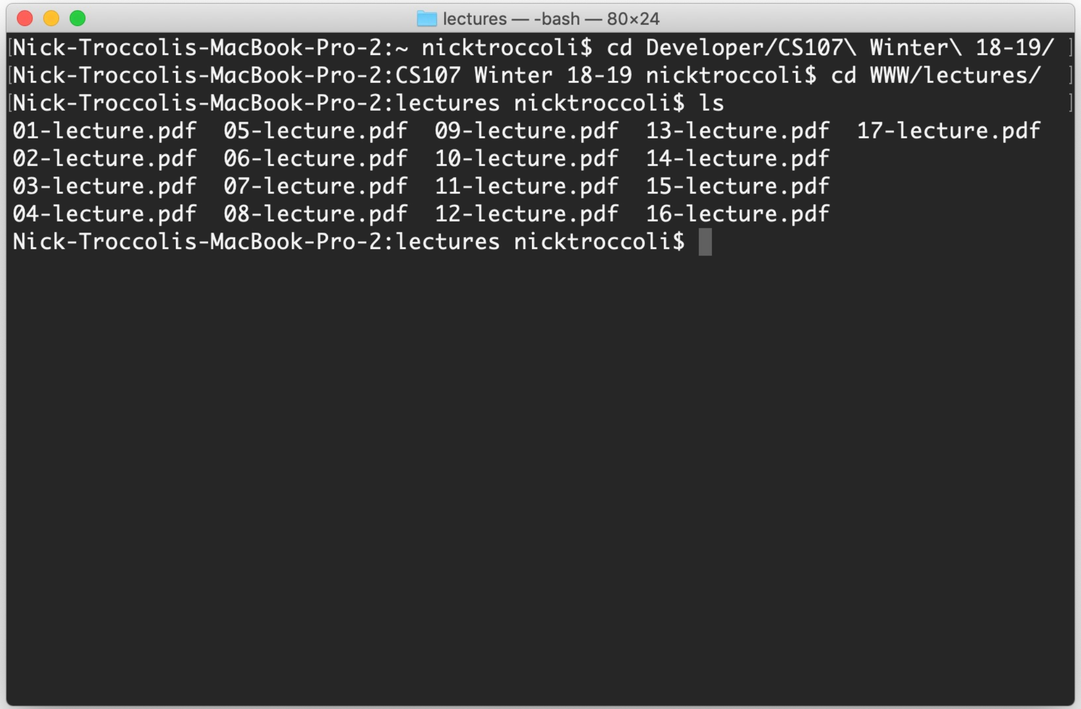
```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

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

First Day

The **command-line** is a text-based interface to navigate a computer, instead of a Graphical User Interface (GUI).



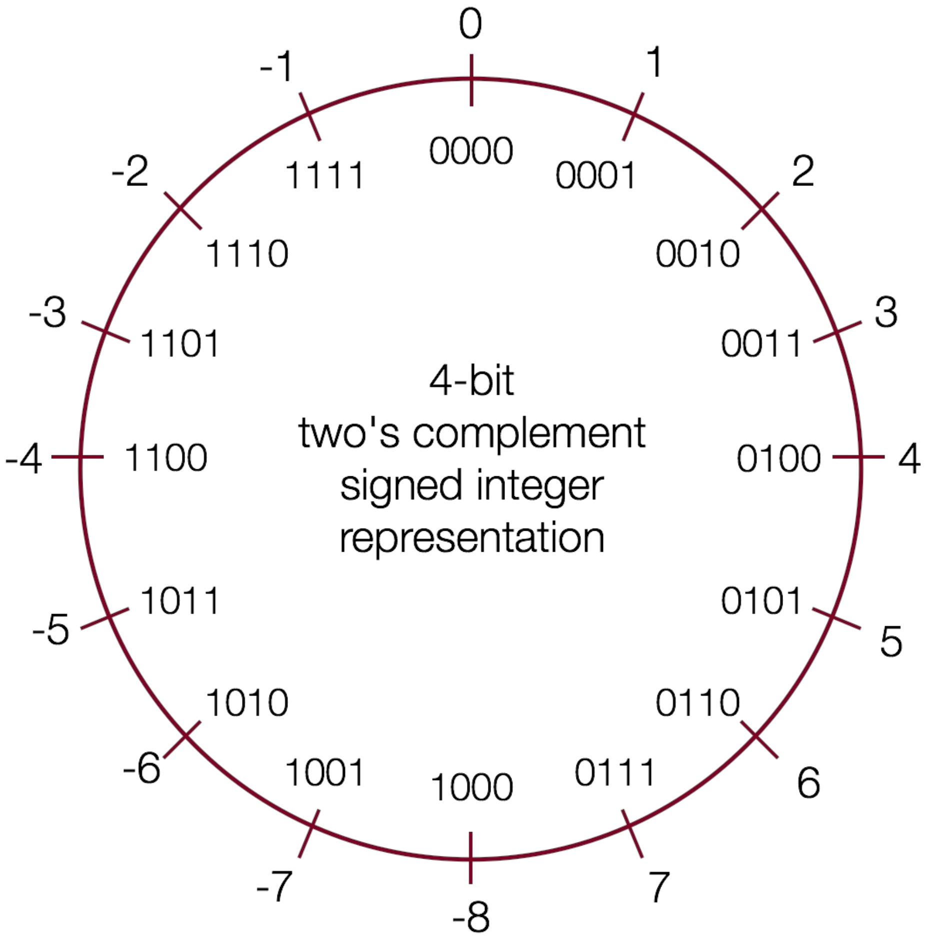
Graphical User Interface



Text-based interface

Bits And Bytes

Key Question: *How can a computer represent integer numbers?*



Bits And Bytes

Why does this matter?

- Limitations of representation and arithmetic impact programs!
- We can also efficiently manipulate data using bits.

C Strings

Key Question: *How can a computer represent and manipulate more complex data like text?*

- Strings in C are arrays of characters ending with a null terminator!
- We can manipulate them using pointers and C library functions (many of which you could probably implement).

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
value	`'H'`	`'e'`	`'l'`	`'l'`	`'o'`	`','`	`' '`	`'w'`	`'o'`	`'r'`	`'l'`	`'d'`	`'!'`	`'\0'`

C Strings

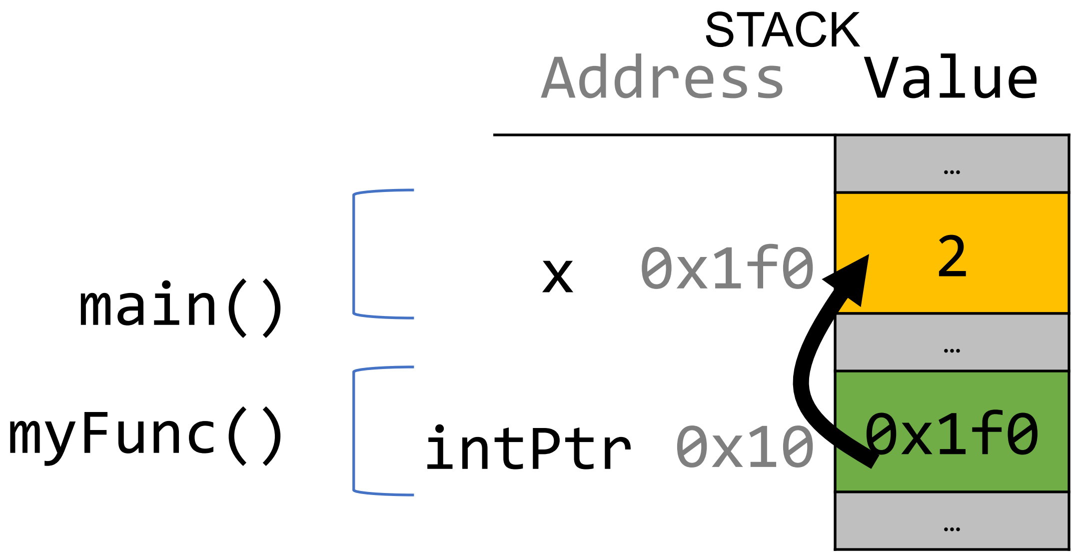
Why does this matter?

- Understanding this representation is key to efficient string manipulation.
- This is how strings are represented in both low- and high-level languages!
  - C++: https://www.quora.com/How-does-C++-implement-a-string
  - Python: https://www.laurentluce.com/posts/python-string-objects-implementation/

Pointers, Stack and Heap

Key Question: *How can we effectively manage all types of memory in our programs?*

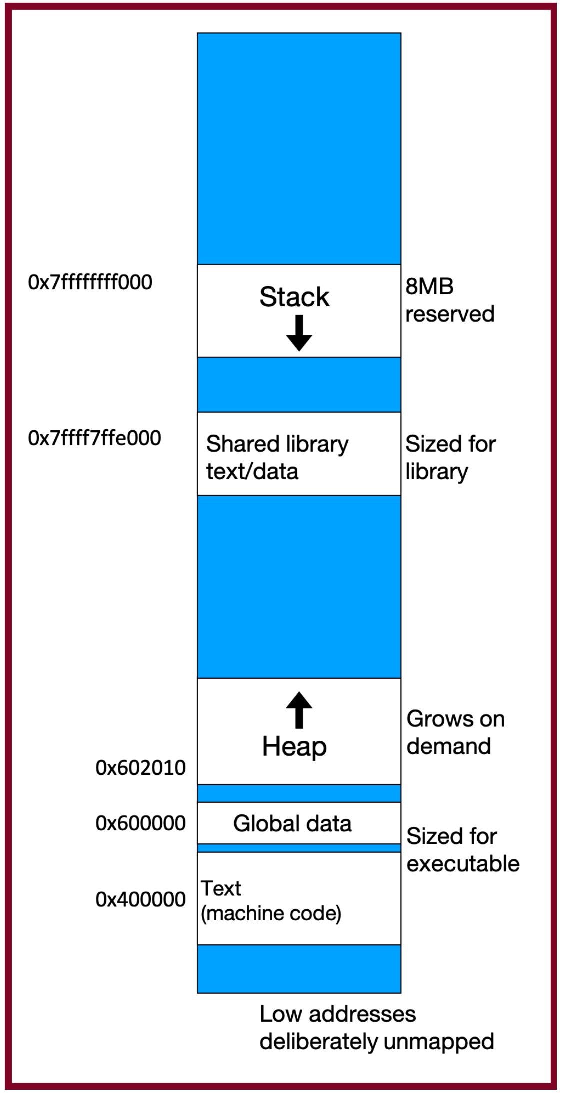
- Arrays let us store ordered lists of information.
- Pointers let us pass addresses of data instead of data itself.
- We can use the stack, which cleans up memory for us, or the heap, which we must manually manage.



Stack And Heap

Why does this matter?

- The stack and heap allow for two ways to store data in our programs, each with their own tradeoffs, and it's crucial to understand the nuances of managing memory in any program you write!
- Pointers let us pass around references to data efficiency



Generics

Key Question: *How can we use our knowledge of memory and data representation to write code that works with any data type?*

- We can use **void *** to circumvent the type system, **memcpy**, etc. to copy generic data, and function pointers to pass logic around.

Why does this matter?

- Working with any data type lets us write more generic, reusable code.
- Using generics helps us better understand the type system in C and other languages, and where it can help and hinder our program.

Assembly Language

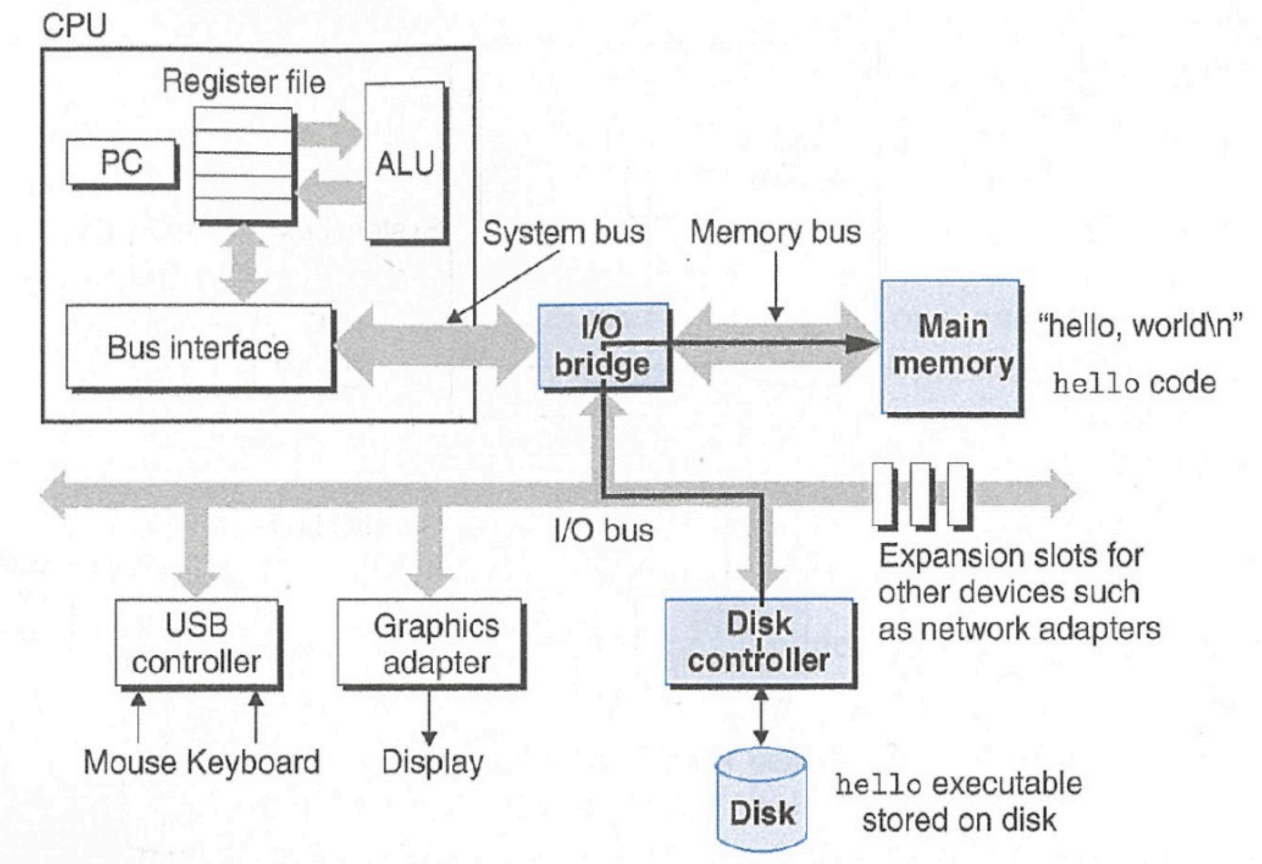
Key Question: *How does a computer interpret and execute C programs?*

- GCC compiles our code into *machine code instructions* executable by our processor.
- Our processor uses registers and instructions like **mov** to manipulate data.

Assembly Language

Why does this matter?

- We write C code because it is higher level and transferrable across machines. But it is not the representation executed by the computer!
- Understanding how programs are compiled and executed, as well as computer architecture, is key to writing performant programs (e.g. fewer lines of code is not necessarily better).
- We can reverse engineer and exploit programs at the assembly level!



Heap Allocators

Key Question: *How do core memory-allocation operations like malloc and free work?*

- A *heap allocator* manages a block of memory for the heap and completes requests to use or give up memory space.
- We can manage the data in a heap allocator using headers, pointers to free blocks, or other designs

Why does this matter?

- Designing a heap allocator requires making many design decisions to optimize it as much as possible. There is no perfect design!
- All languages have a “heap” but manipulate it in different ways.

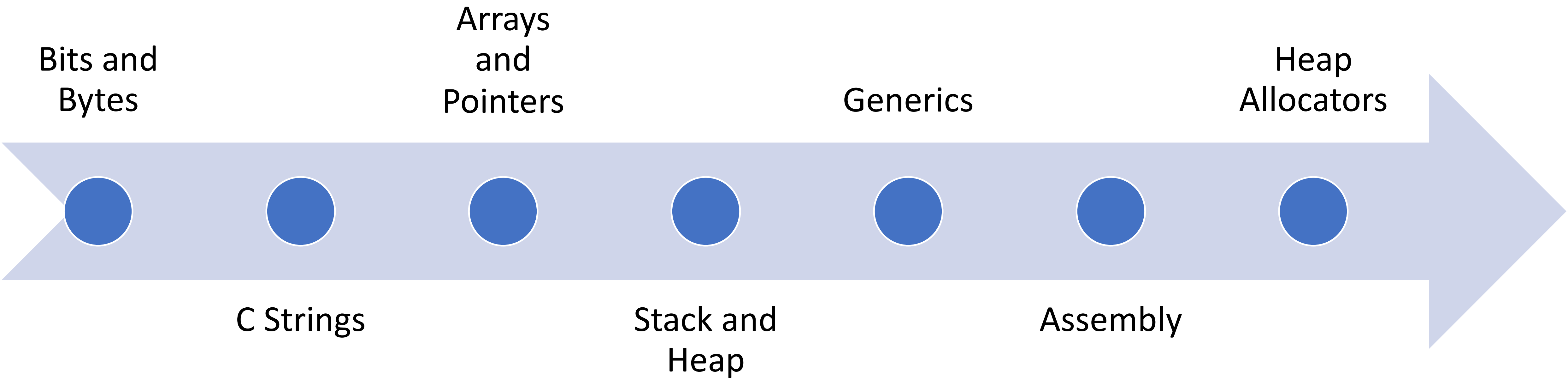
Ethics, Privacy, Partiality and Trust

Key Question: *How do we act responsibly in maintaining security, protecting privacy, and ensuring warranted trust in the systems we build and maintain?*

Why does this matter?

- Responsible programming involves documentation -- including informative error messages! -- that allows others to evaluate the reliability of your code.
- Responsible system and program design also requires choosing a trust model and considering data privacy. This might also require deciding to whom to be partial.

Our CS107 Journey



CS107 Learning Goals

The goals for CS107 are for students to gain **mastery** of

- writing C programs with complex use of memory and pointers
- an accurate model of the address space and compile/runtime behavior of C programs

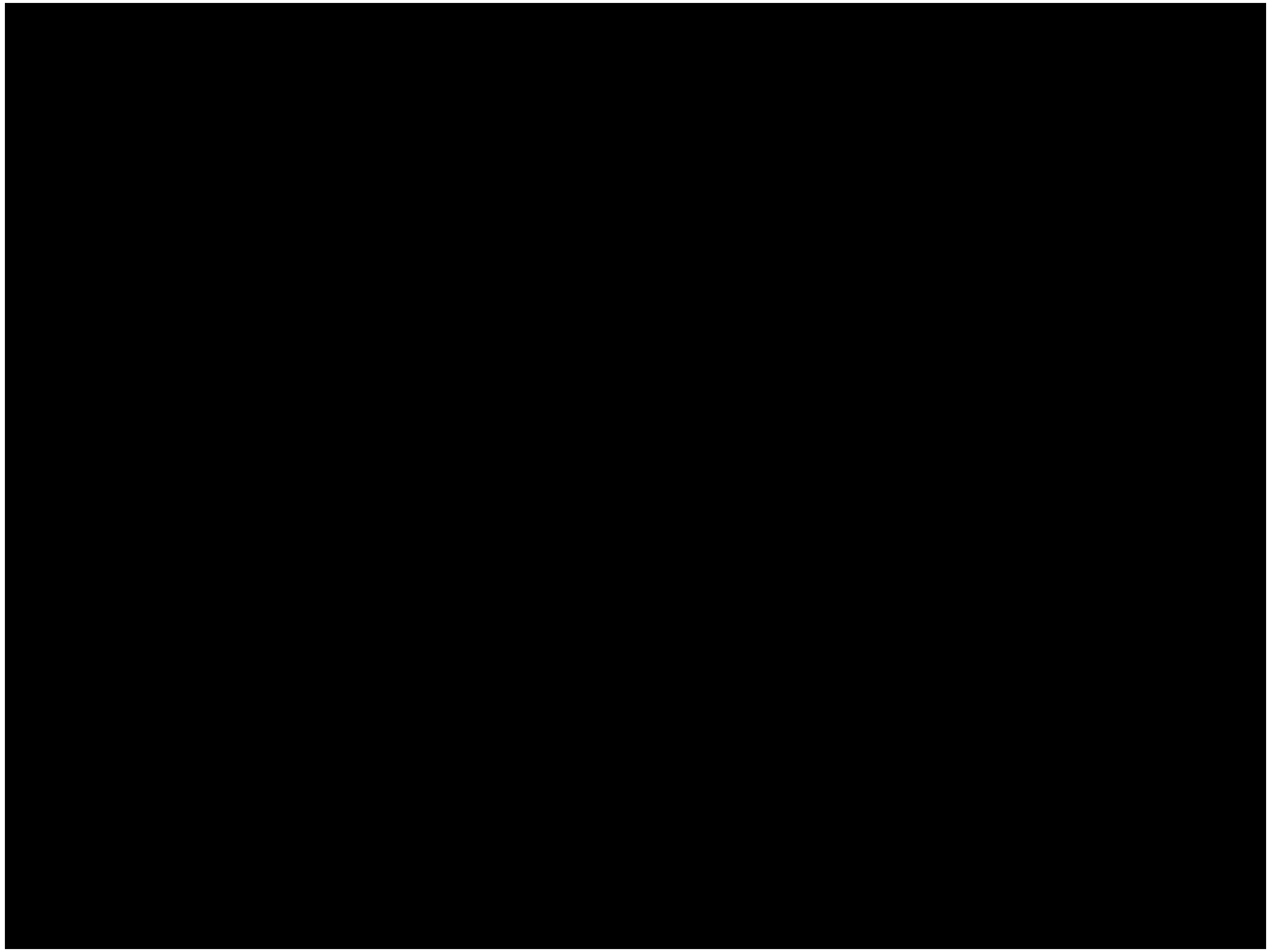
to achieve **competence** in

- translating C to/from assembly
- writing programs that respect the limitations of computer arithmetic
- identifying bottlenecks and improving runtime performance
- working effectively in a Unix development environment
- using ethical frameworks and case studies to inform decision-making

and have **exposure** to

- a working understanding of the basics of computer architecture

The C Coding Experience



https://www.youtube.com/watch?v=G7LJC9vJluU

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
- What's Next?
- Q&A

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
  - CS107 Tools and Techniques
  - CS107 Concepts
- What's Next?
- Q&A

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
  - **CS107 Tools and Techniques**
  - CS107 Concepts
- What's Next?
- Q&A

Tools and Techniques

- Unix and the command line
- Coding Style
- Debugging (GDB)
- Testing (Sanity Check)
- Memory Checking (Valgrind)
- Profiling (Callgrind)

Unix And The Command Line

Unix and command line tools are extremely popular tools outside of CS107 for:

- Running programs (web servers, python programs, remote programs…)
- Accessing remote servers (Amazon Web Services, Microsoft Azure, Heroku…)
- Programming embedded devices (Raspberry Pi, etc.)

<u>Our goal for CS107 was to help you become proficient in navigating Unix</u>

Coding Style

- Writing clean, readable code is crucial for any computer science project
- Unfortunately, a fair amount of existing code is poorly-designed/documented

<u>Our goal for CS107 was to help you write with good coding style, and read/understand/comment provided code.</u>

Debugging (GDB)

- Debugging is a crucial skill for any computer scientist
- <u>Our goal for CS107 was to help you become a better debugger</u>
  - narrow in on bugs
  - diagnose the issue
  - implement a fix
- Practically every project you work on will have debugging facilities
  - Python: “PDB”
  - IDEs: built-in debuggers (e.g. QT Creator, Eclipse)
  - Web development: in-browser debugger

Testing (Sanity Check)

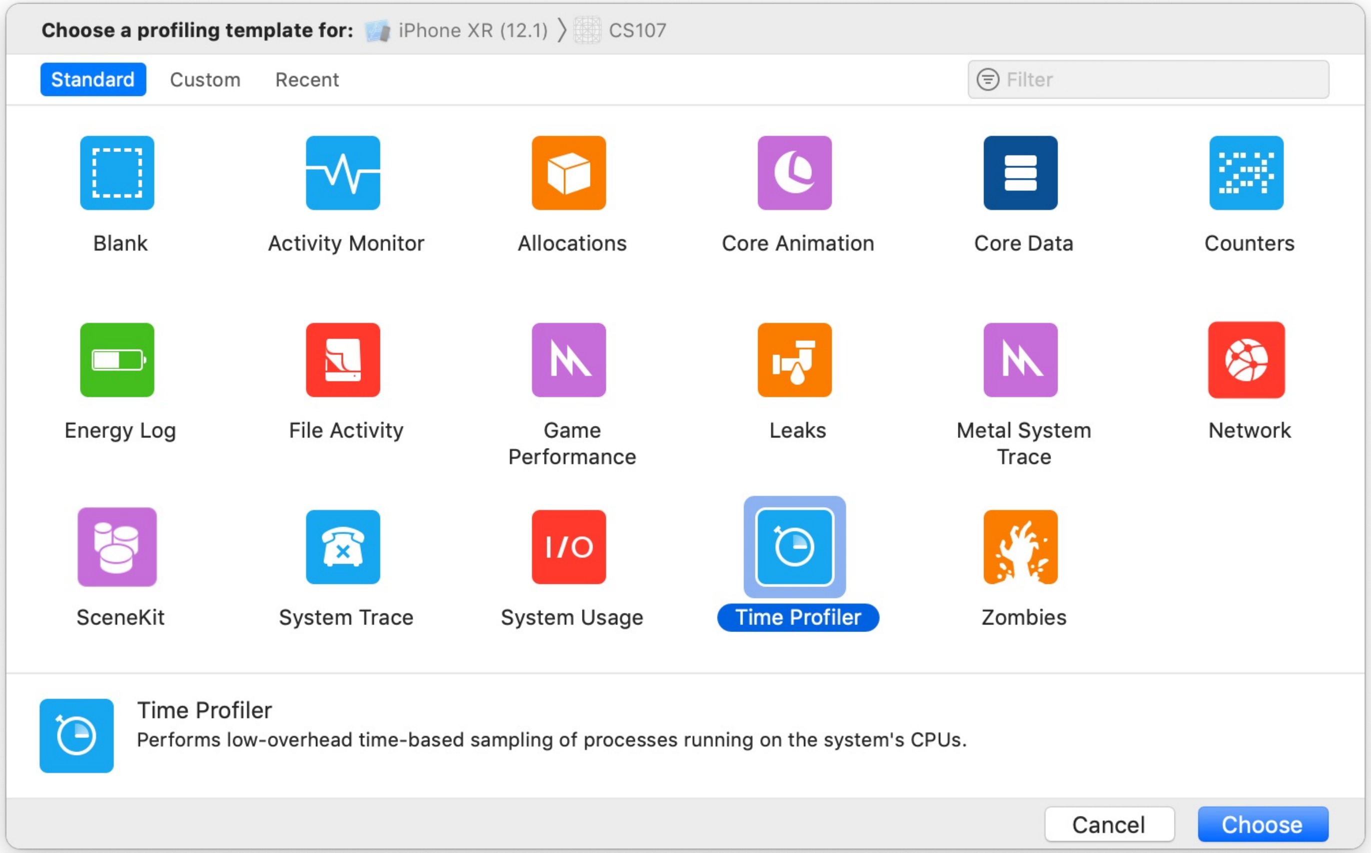
- Testing is a crucial skill for any computer scientist
- <u>Our goal for CS107 was to help you become a better tester</u>
  - Writing targeted tests to validate correctness
  - Use tests to prevent regressions
  - Use tests to develop incrementally

Memory Checking and Profiling

- Memory checking and profiling are crucial for any computer scientist to analyze program performance and increase efficiency.
- Many projects you work on will have profiling and memory analysis facilities:
  - Mobile development: integrated tools (XCode Instruments, Android Profiler, etc.)
  - Web development: in-browser tools

Tools

You'll see manifestations of these tools throughout projects you work on. We hope you can use your CS107 knowledge to take advantage of them!



Plan For Today

- **Recap**: Where We've Been
- Larger Applications
  - CS107 Tools and Techniques
  - **CS107 Concepts**
- What's Next?
- Q&A

Concepts

- C Language
- Bit-Level Representations
- Arrays and Pointers
- Memory Management
- Generics
- Assembly

Systems

How do operating systems work? (take CS111!)

- Storing files in filesystems
- Running user programs
- Sharing memory between programs
- Running multiple tasks concurrently with multithreading and multiprocessing

How is a compiler implemented? (take CS143!) [Demo]

- Lexical analysis
- Parsing
- Semantic Analysis
- Code Generation

How can applications communicate over a network? (take CS144!)

- How can we weigh different tradeoffs of network architecture design?
- How can we effectively transmit bits across a network?

Machine Learning

Can we speed up machine learning training with reduced precision computation?

- https://www.top500.org/news/ibm-takes-aim-at-reduced-precision-for-new-generation-of-ai-chips/
- https://devblogs.nvidia.com/mixed-precision-training-deep-neural-networks/

How can we implement performant machine learning libraries?

- Popular tools such as TensorFlow and PyTorch are implemented using C!
- https://pytorch.org/blog/a-tour-of-pytorch-internals-1/
- https://www.tensorflow.org/guide/extend/architecture

Web Development

How can we efficiently translate Javascript code to machine code?

- The Chrome V8 JavaScript engine converts Javascript into machine code for computers to execute: https://medium.freecodecamp.org/understanding-the-core-of-nodejs-the-powerful-chrome-v8-engine-79e7eb8af964
- The popular Node.js web server tool is built on Chrome V8

How can we compile programs into an efficient binary instruction format that runs in a web browser?

- WebAssembly is an emerging standard instruction format that runs in browsers: https://webassembly.org
- [DEMO] You can compile C/C++/other languages into WebAssembly for web execution

Programming Languages / Runtimes

How can programming languages and runtimes efficiently manage memory?

- Manual memory management (C/C++)
- Reference Counting (Swift)
- Garbage Collection (Java)

How can we design programming languages to reduce the potential for programmer error? (take CS242!)

- Haskell/Swift ”Optionals”

How can we design portable programming languages?

- Java Bytecode: https://en.wikipedia.org/wiki/Java_bytecode

Theory

How can compilers output efficient machine code instructions for programs? (take CS143!)

- Languages can be represented as regular expressions and context-free grammars
- We can model programs as control-flow graphs for additional optimization

Security

How can we find / fix vulnerabilities at various levels in our programs? (take CS155!)

- Understand machine-level representation and data manipulation
- Understand how a computer executes programs
- macOS High Sierra Root Login Bug: https://objective-see.com/blog/blog_0x24.html

How can we ensure that our systems and networks are secure? (take CS155!)

How can we design internet services that are worthy of trust? (take CS152!)

Ethics, Privacy, Partiality and Trust

Why is privacy important? What technical and policy standards should we strive for in protecting privacy? (take CS 182!)

How can we recognize ethically important decisions as they arise? What policies ought we to adopt to address these issues? (take CS 181!)

Floats and Assembly

- Unfortunately, we couldn’t cover floating point numbers this quarter
  - Lecture video from past quarter here!
- An example of tradeoffs in design decisions.
- Importance of thinking through priorities and pros/cons when designing systems.

Try Yourself: Float Arithmetic

Try it yourself:

```
./bank 100 1          # deposit
./bank 100 -1         # withdraw
./bank 100000000 -1 # make bank
./bank 16777216 1    # lose bank
```

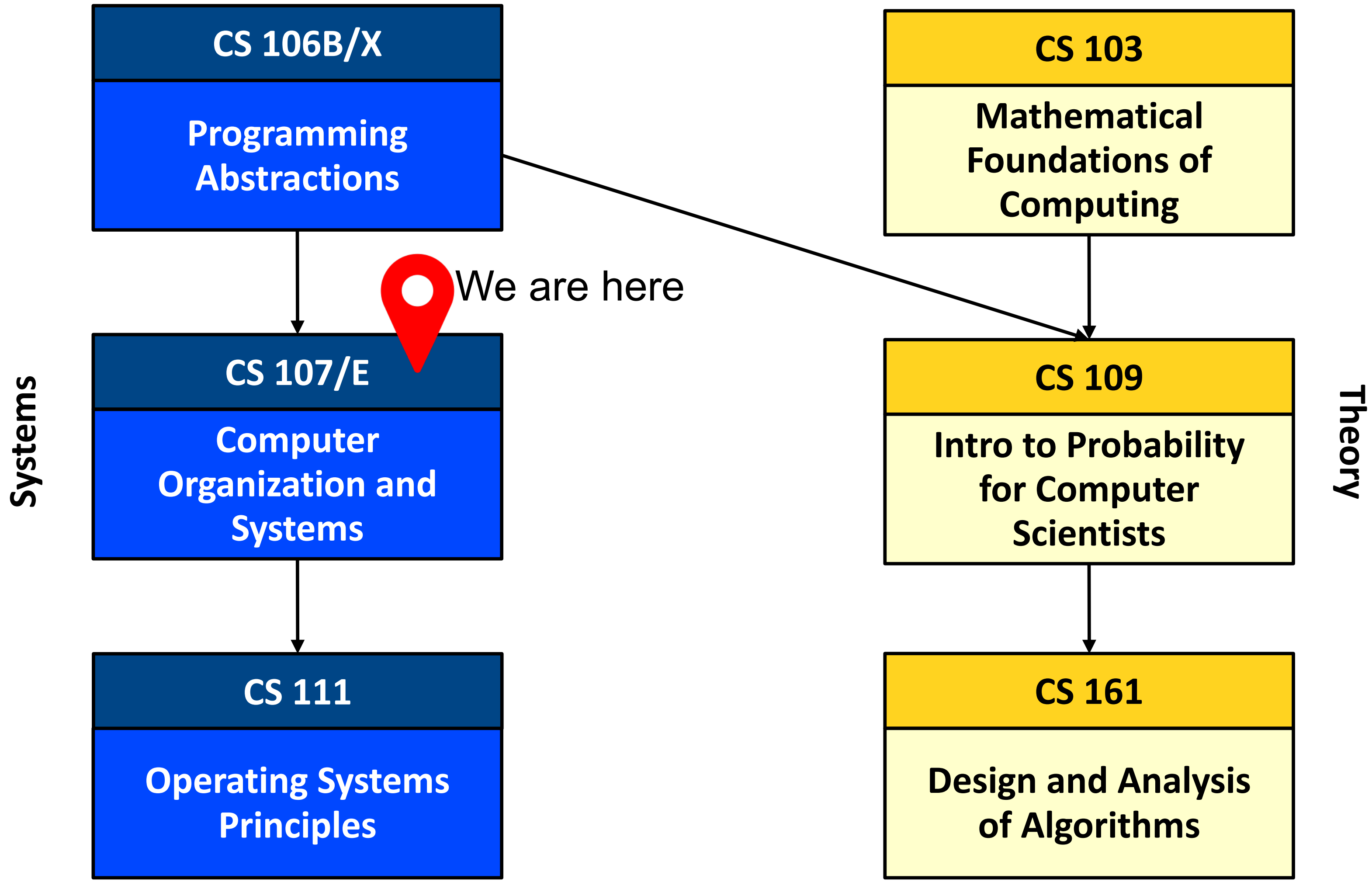
bank.c

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
- What's Next?
- Q&A

After CS107, you are prepared to take a variety of classes in various areas. What are some options?

Where Are We?


CS 106B/X
Programming Abstractions
We are here
CS 107/E
Computer Organization and Systems
CS 111
Operating Systems Principles
Systems
CS 103
Mathematical Foundations of Computing
CS 109
Intro to Probability for Computer Scientists
CS 161
Design and Analysis of Algorithms
Theory

CS111

CS107 (or equivalent) builds up and expanded your breadth and depth of programming experience and techniques and shows you how machines really work.

CS111 leverages this programming experience to introduce operating systems and how they work.

What is an operating system?

Jerry Cain

Nick Troccoli

John Ousterhout

What is an Operating System?

An operating system (“OS”) is software that allows people to run programs on a computer.

- Examples: iOS, Android, Windows, macOS, Linux

You may think mostly of the *user interface* of the operating system, but an operating system does so much more!

What is an Operating System?

The operating system sits between the hardware and user programs. It manages shared resources and provides functionality for programs to run.

It manages things like:

- Processor (CPU): decides what program gets to do work and for how long
- Memory (RAM): decides what programs get to use what areas of memory
- Hard Drive: decides how the disk is used to store files

User Programs

Operating System

Hardware (memory, hard drive, processor, etc.)

What is an Operating System?

- So far, when you've written programs, you haven't had to think about any of this. That's the point! The OS is doing its job – it abstracts away complexity from programs.
  - Don't have to coordinate with other programs for who gets to use what memory
  - Don't have to coordinate with other programs for who gets to run when
- OSes work behind the scenes, but are extremely powerful
  - **Example:** devices with 1 CPU core (common through early 2000s) could really only execute 1 program at a time! OSes switch *very* quickly between different tasks to simulate appearance of multitasking.
  - **Example:** how can every program think it can use every address from NULL to 0xfff…? OSes tell programs *fake ("virtual") addresses* and behind the scenes it maps them to the actual ("physical") addresses that it organizes itself.

Other Courses

- **CS112**: Operating Systems Project
- **CS140/CS212**: Operating Systems
- **CS143:** Compilers
- **CS144:** Networking
- **CS145:** Databases
- **CS152**: Trust and Safety Engineering
- **CS155:** Computer and Network Security
- **CS166:** Data Structures
- **CS181:** Computers, Ethics, and Public Policy
- **CS182:** Ethics, Public Policy, and Technological Change
- **CS221:** Artificial Intelligence
- **CS246**: Mining Massive Datasets
- **EE108**: Digital Systems Design
- **EE180:** Digital Systems Architecture

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
- What's Next?
- Q&A

Thank you!

Course Evaluations

We hope you can take the time to fill out the end-quarter CS 107 course evaluation once it’s available. We sincerely appreciate any feedback you have about the course and read every piece of feedback we receive. We are always looking for ways to improve!

Thank you!

Q&A: What questions do you have?

Respond on PollEV