

# CS107, Lecture 27

## Wrap-Up / What's Next?

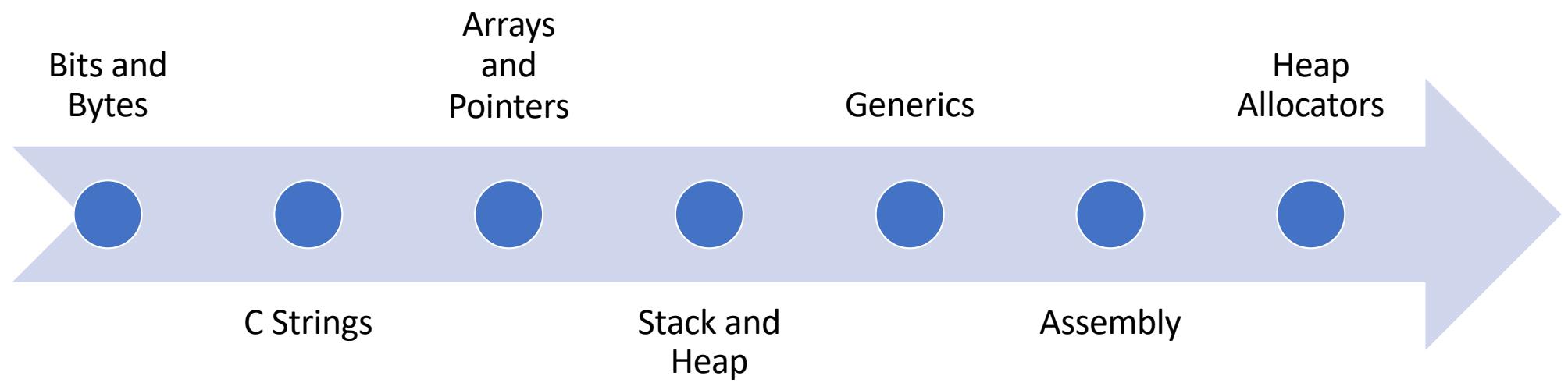


This document is copyright (C) Stanford Computer Science, licensed under Creative Commons Attribution 2.5 License. All rights reserved.  
Based on slides created by Cynthia Lee, Chris Gregg, Jerry Cain, Lisa Yan, Nick Troccoli, and others.



**We've covered a lot in just  
10 weeks! Let's was  
nostalgic.**

# Our CS107 Journey



# Course Overview

1. **Bits and Bytes** - *How can a computer represent integer numbers?*
2. **Characters and C Strings** - *How can a computer represent and manipulate more complex data types like text?*
3. **Pointers, Stack Memory and Heap Memory** – *How can we effectively manage all forms of memory in our programs?*
4. **Generics** - *How can we tap our knowledge of computer memory and data representation to write code that works with any data type?*
5. **Assembly** - *How does a computer compile, interpret, and execute C programs? And what does assembly code look like?*
6. **Heap Allocators** - *How do core memory-allocation operations like `malloc` and `free` work? Are the built-in versions always good enough?*

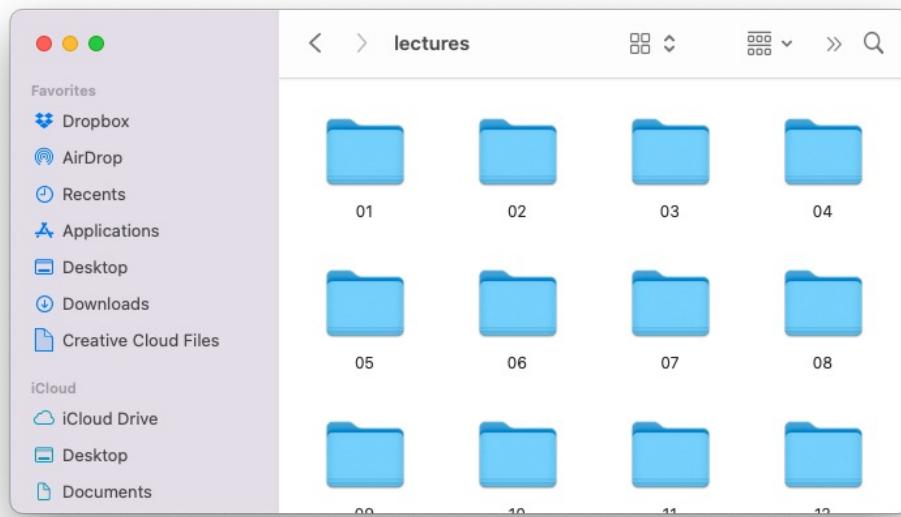
# First Day

```
/*
 * 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).

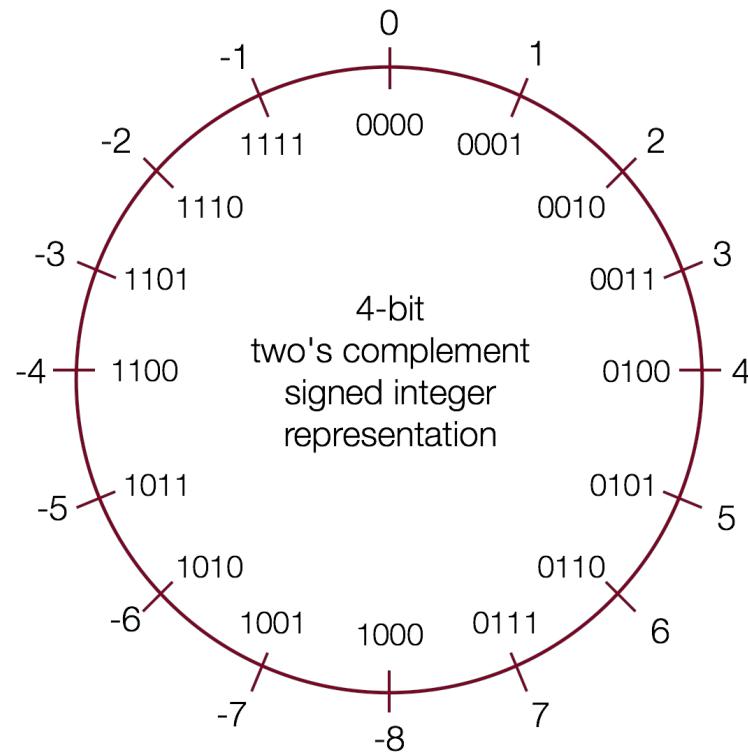


Graphical User Interface

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.

PSY - GANGNAM STYLE (강남스타일) M/V



officialpsy

Subscribe 7,600,830

Add to Share \*\*\* More

-2142584554

8,761,309 1,139,833



# 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 now).

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

# C Strings

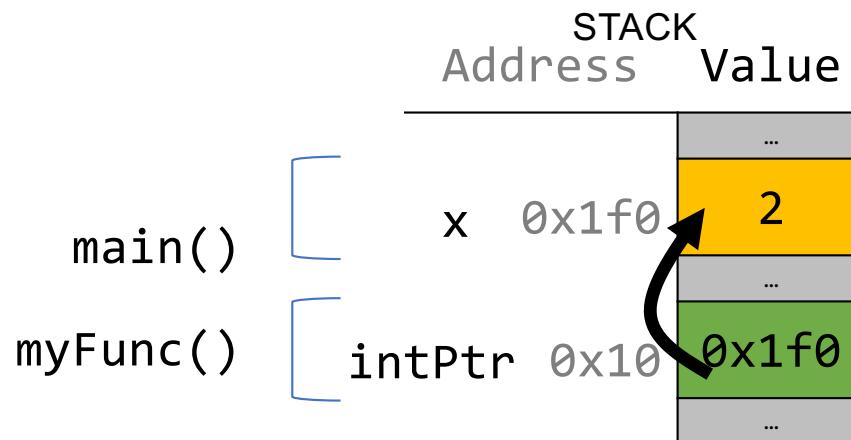
Why does this matter?

- 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?*

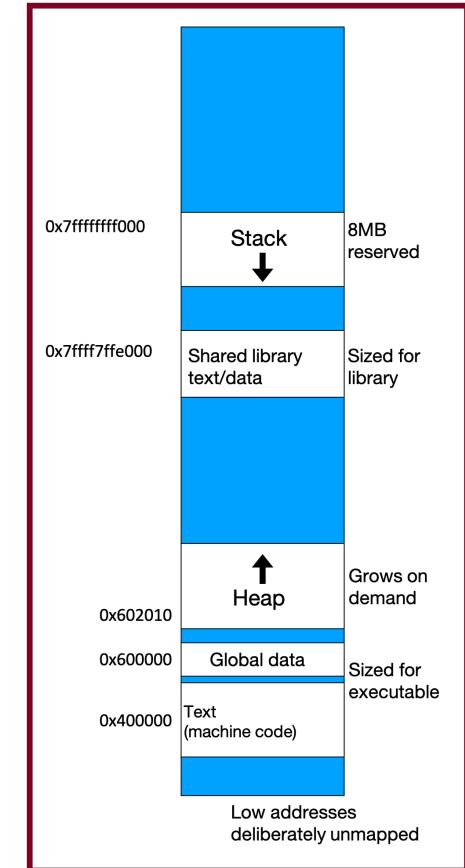
- Arrays let us store ordered lists of information.
- Pointers let us pass addresses of data instead of the data itself.
- We can use the stack, which cleans up memory for us, or the heap, which we must manually manage.



# Stack And Heap

Why does this matter?

- The stack and heap allow for two ways to store data, and each with its own pros and cons. It's crucial to understand the nuances of memory management in any program you write!
- Pointers let us efficiently pass around references to data.



# 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 sedate the type system, **memcpy**, etc. to copy raw bytes of data, and function pointers to pass around logic to be applied to generics.

Why does this matter?

- Working with any data type allows us to write more generic, reusable code.
- Generics are our opportunity to better understand C's type system and even the type systems of other languages. Generics: lots of pros, lots of cons!

# Assembly Language

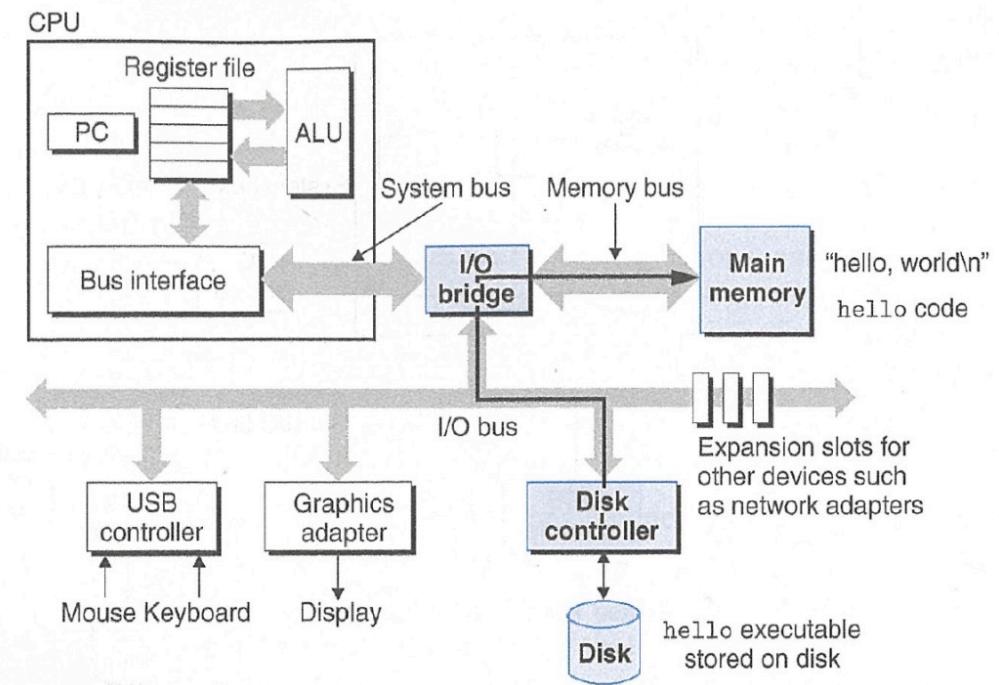
**Key Question:** *How does a computer interpret and execute C programs?*

- GCC compiles our code into *machine code instructions* that can be executed on the hardware.
- CPUs rely on registers and instructions like **mov**, **add**, and **xor** to manipulate data, and instructions like **jmp**, **jne**, **jae**, **call**, and **ret** to control execution flow.

# Assembly Language

## Why does this matter?

- We write C code because it is higher level and architecture-neutral. It is not, however, what's executed on the hardware!
- Understanding how programs are compiled and executed is key to writing performant programs (e.g., fewer lines of code is not necessarily better).
- We can reverse engineer and ethically 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 special segment of memory (the heap, obvi) and completes requests to allocate or free up slices of it.
- We can manage the data in a heap allocator using headers, pointers to free blocks, or other more advanced directives.

Why does this matter?

- Designing a heap allocator requires all allocation and deallocation requests work and work efficiently. There is no single best design!
- All languages implement a heap. Some expose it, and others conceal it.

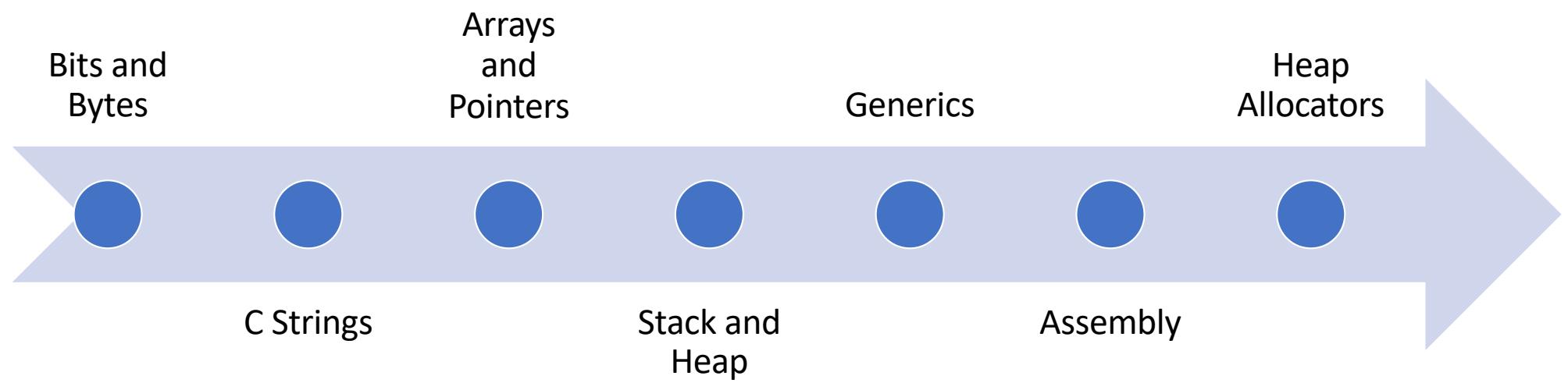
# Ethics, Privacy, Partiality and Trust

**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 it and assess risks.
- Responsible system design requires we choose a trust model, consider data privacy, and decide who we’re partial to when vulnerabilities are exposed.

# Our CS107 Journey



# CS107 Learning Goals

The goals for CS107 are for students:

to acquire a **fluency** with

- pointers and memory and how to make use of them when writing C code
- an executable's address space and runtime behavior

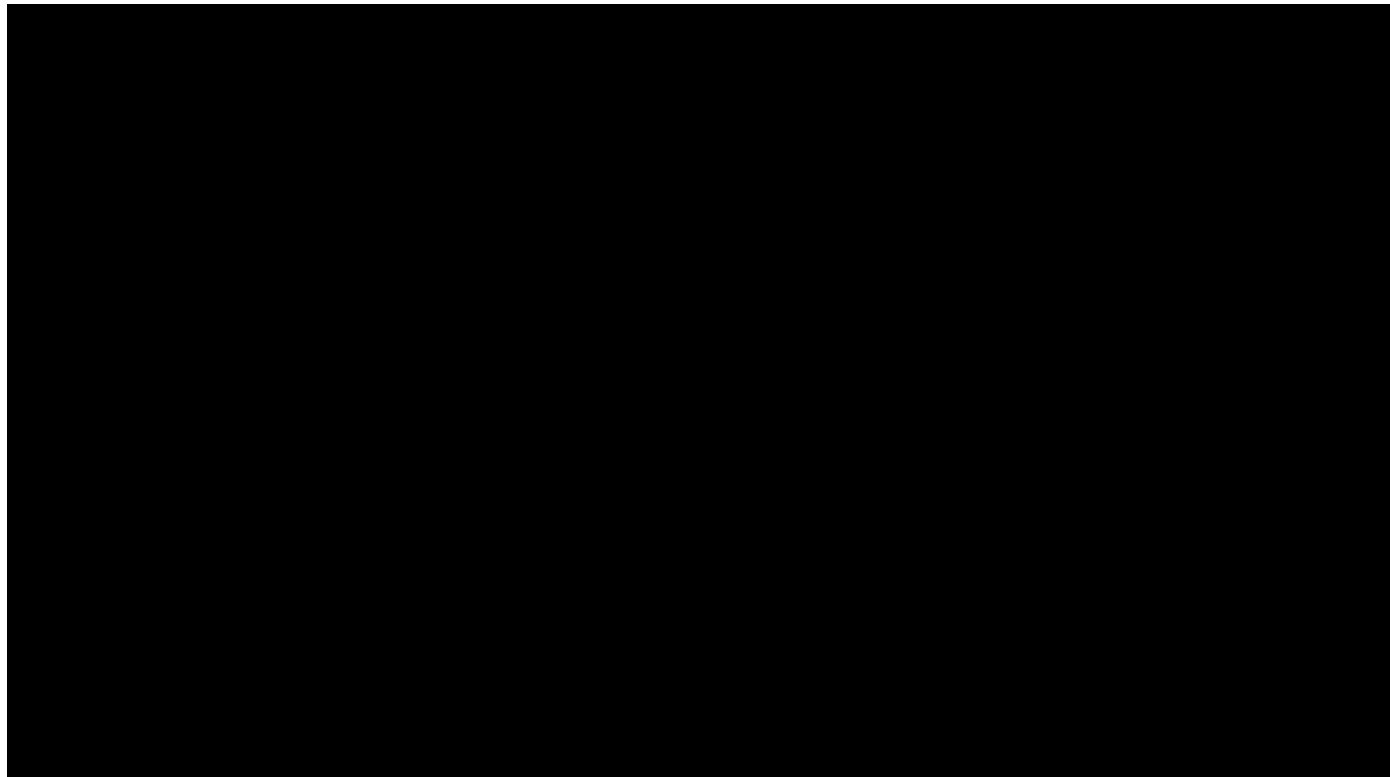
to acquire **competency** with

- the translation of C to and from assembly code
- the implementation of programs that respect the limits of computer arithmetic
- the ability to identify bottlenecks and improve runtime performance
- the ability to navigate your own Unix development environment
- the ethical frameworks you need to better design and implement software

to have some **exposure** to

- the basics of computer architecture

# The C Coding Experience



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

# Tools and Techniques

- Unix and the command line
- Coding Style
- Debugging (**gdb**)
- Testing (**sanitycheck**)
- 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.)

Our goal for CS107 was to help you become proficient in navigating Unix

# Coding Style

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

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

# Debugging (GDB)

- Debugging is a crucial skill for any computer scientist
- Our goal for CS107 was to help you become a better debugger
  - 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, XCode, Visual Studio)
  - Web development: in-browser debugger

# Testing (sanitycheck)

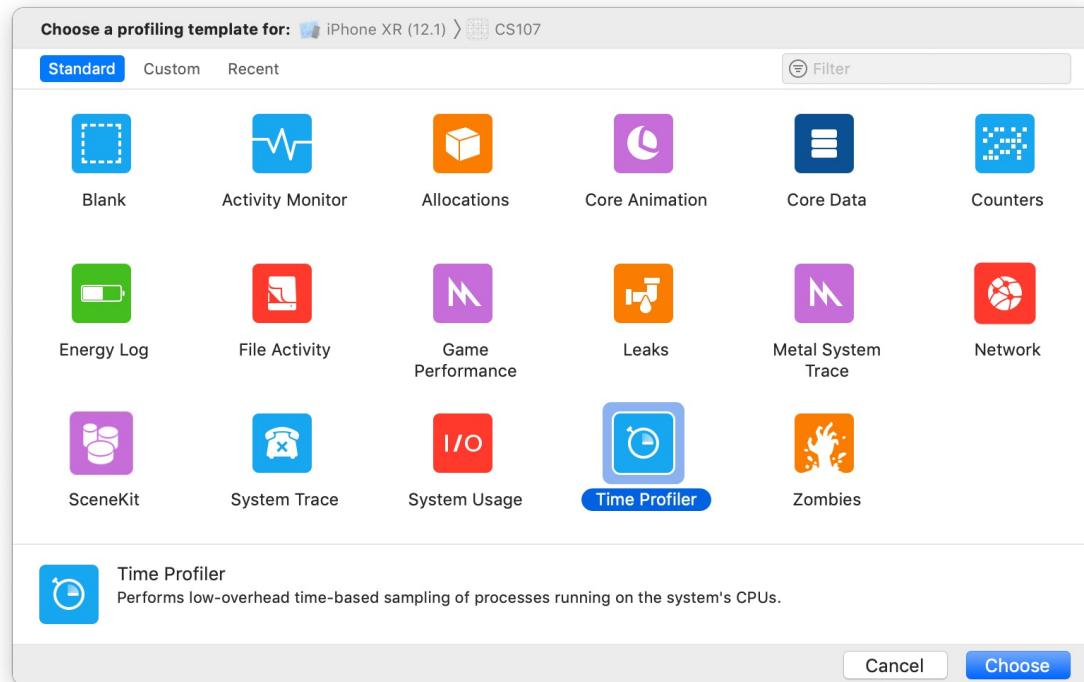
- Testing is a crucial skill for any computer scientist
- Our goal for CS107 was to help you become a better tester
  - Writing targeted tests to validate correctness
  - Use tests to prevent regressions
  - Use tests to develop incrementally

# Memory Validation 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!



# Concepts

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

# Systems

**How can we write programs to execute many tasks simultaneously? (take CS111!)**

- Threads of execution
- Locks to prevent simultaneous access

**How is a compiler implemented? (take CS143!)**

- 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?

# Systems

**How is an operating system implemented? (take CS111/CS112/CS212!)**

- Threads
- User Programs
- Virtual Memory
- Filesystem

# 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 **nodejs** 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>
- 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'
- Rust and resource ownership

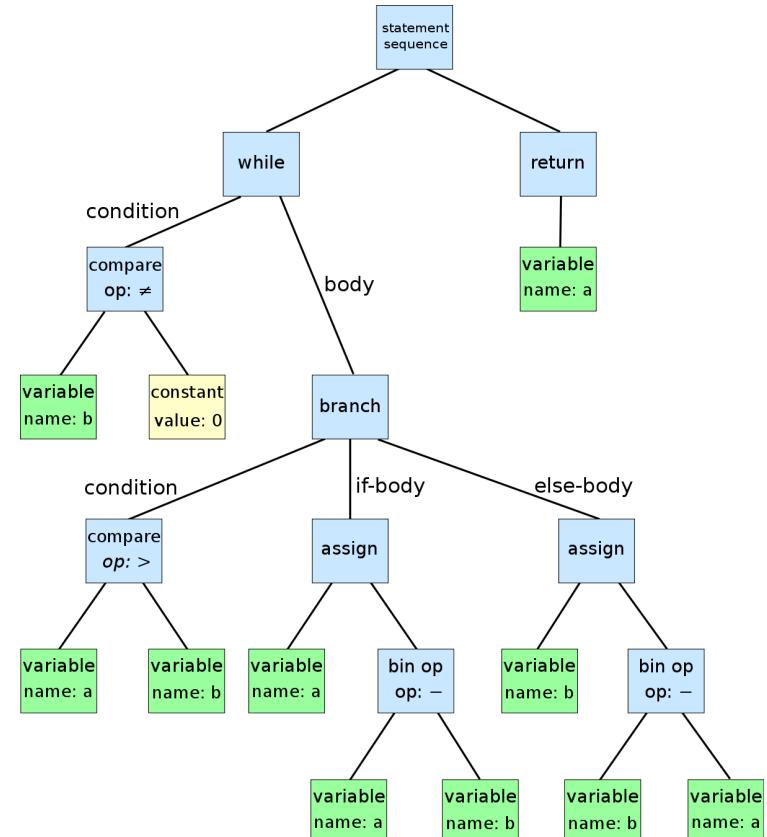
**How can we design portable programming languages?**

- Java Bytecode: [https://en.wikipedia.org/wiki/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, and programs can be represented as tree structures.
- 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](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 worthy of our trust? (take CS152!)**

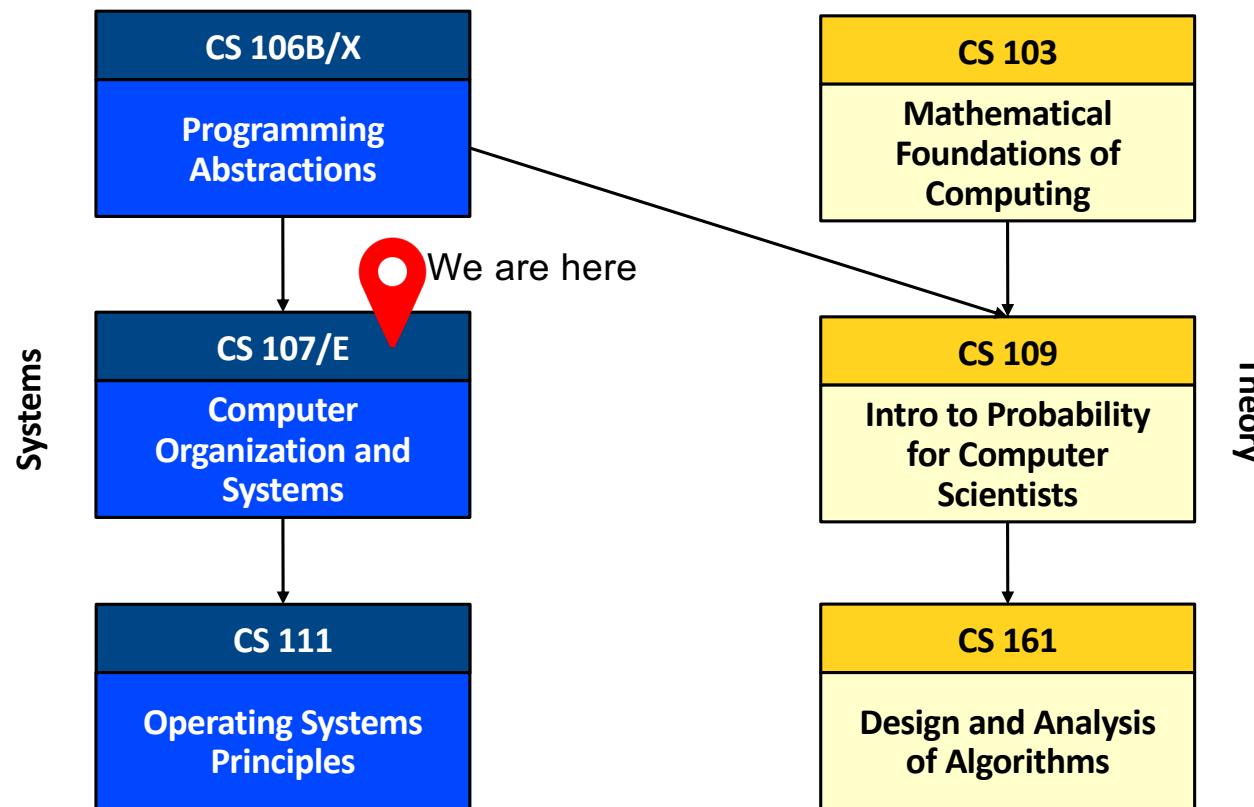
# **Ethics, Privacy, Partiality and Trust**

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

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

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

# Where Are We?



# CS 111

- How can programs perform multiple tasks concurrently and share resources between those tasks?
- How does every program think it has access to all memory addresses if it needs them?
- How can we implement a filesystem to store persistent data?



Jerry Cain



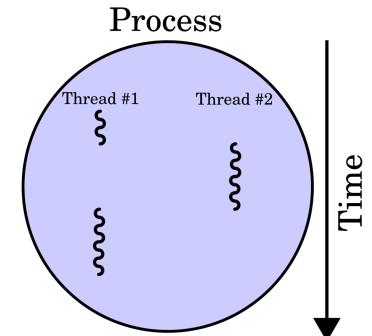
David Mazieres



Nick Troccoli



John Ousterhout



[https://en.wikipedia.org/wiki/Multithreading\\_\(computer\\_architecture\)#/media/File:Multithreaded\\_process.svg](https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)#/media/File:Multithreaded_process.svg)

# 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



**Thank you!**