CS107, Lecture 16
Wrap-Up / What’s Next?

Plan For Today

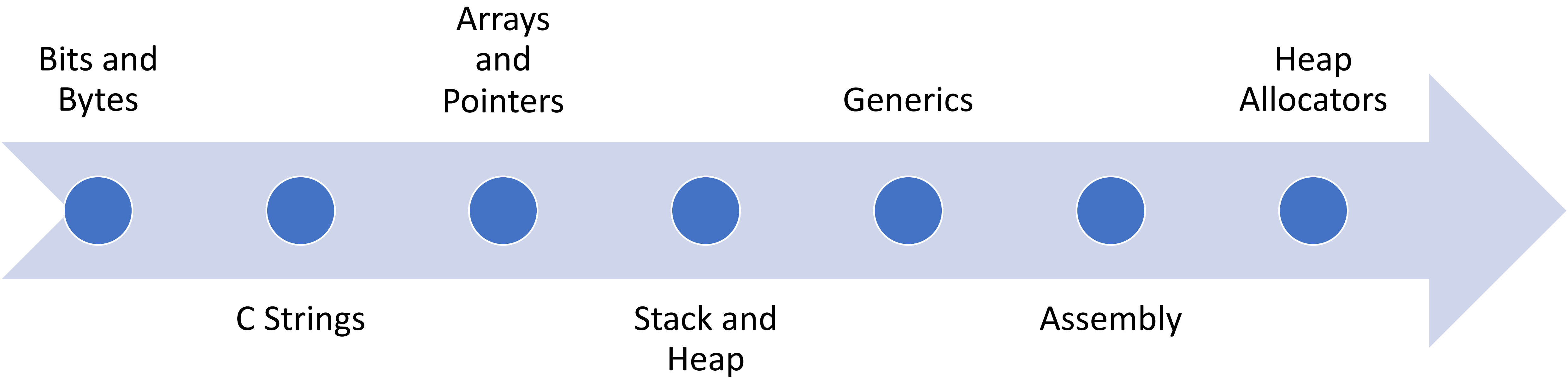
- **Recap**: Where We've Been
- Larger Applications
- What's Next?
- Q&A

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

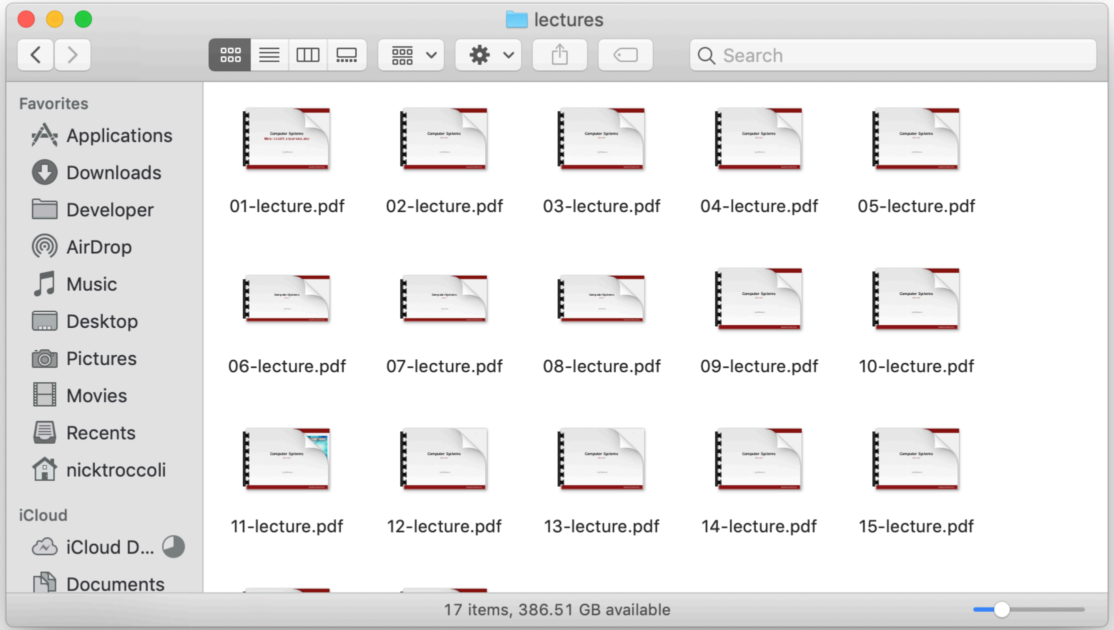
1. **Bits and Bytes -** *How can a computer represent integer numbers?*
2. **Chars and C-Strings -** *How can a computer represent and manipulate more complex data like text?*
3. **Pointers, Stack and Heap –** *How can we effectively manage all types of memory in our programs?*
4. **Generics -** *How 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?*

First Day

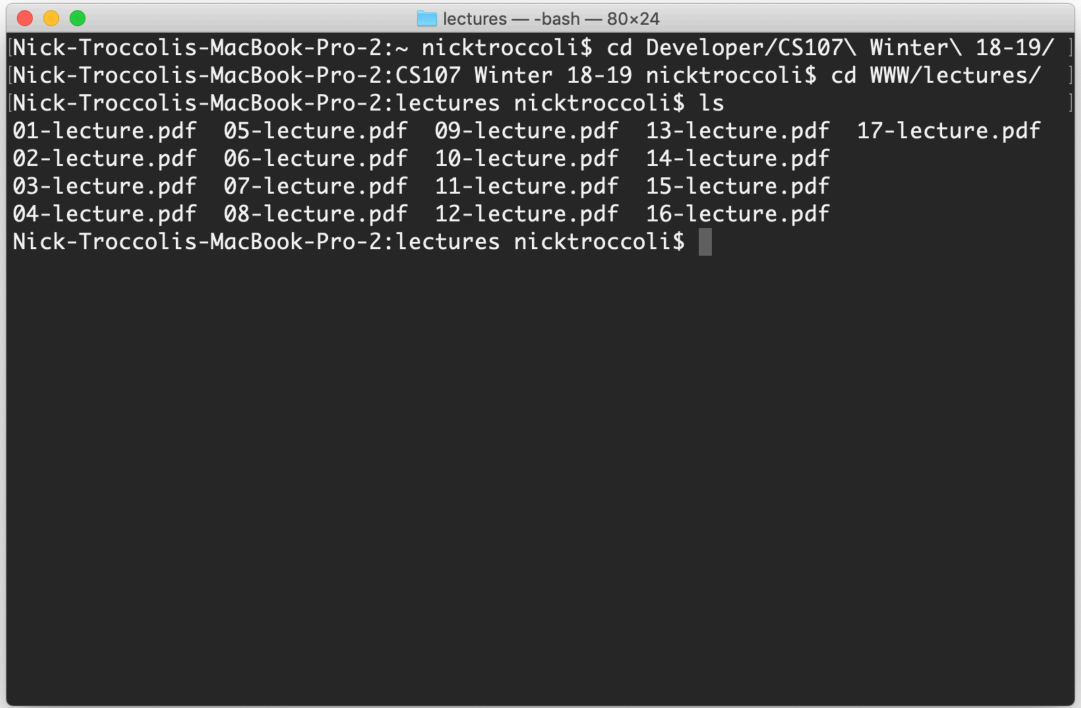
```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

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).



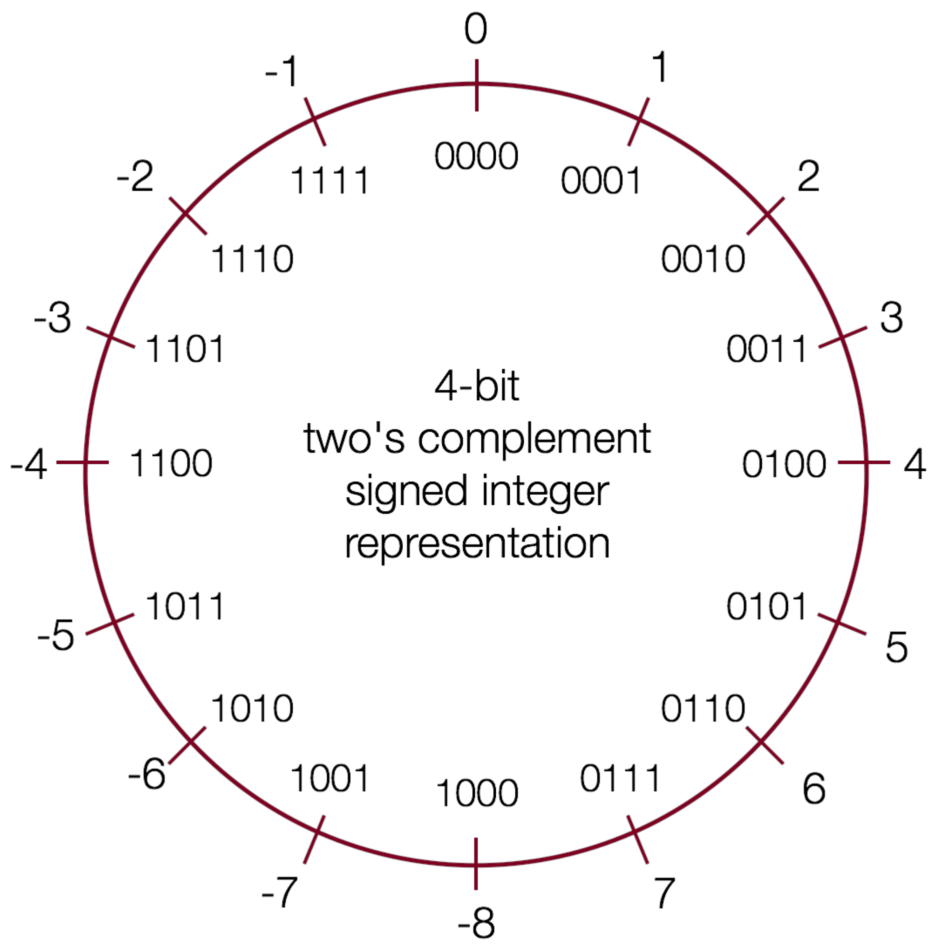
Graphical User Interface



Text-based interface

Bits And Bytes

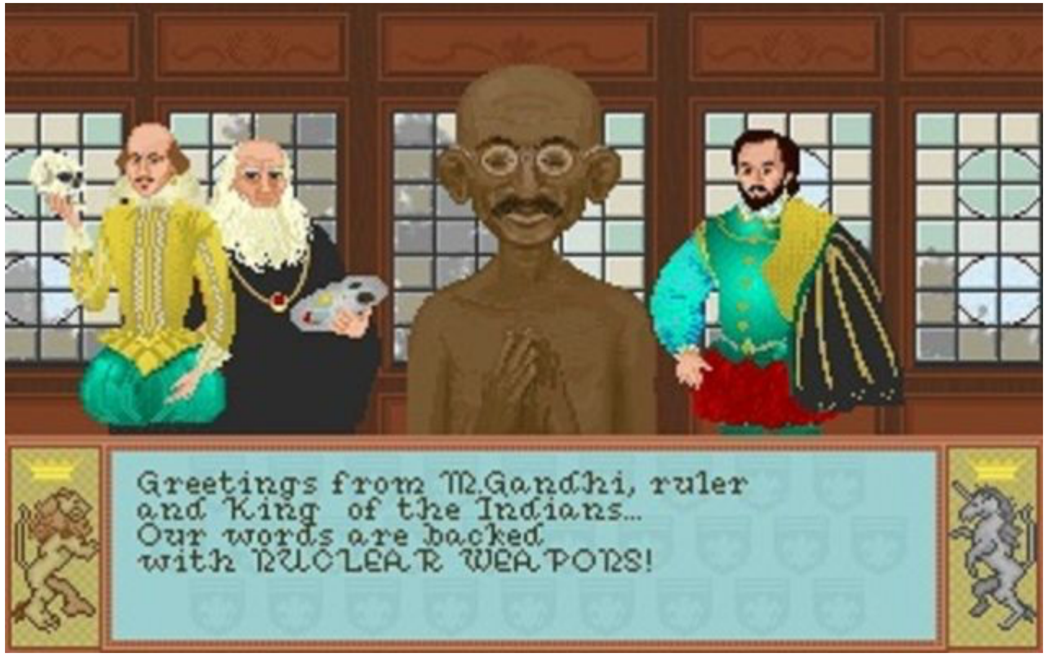
Key Question: *How can a computer represent integer numbers?*



Bits And Bytes

Why does this matter?

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



https://kotaku.com/why-gandhi-is-such-an-asshole-in-civilization-1653818245

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

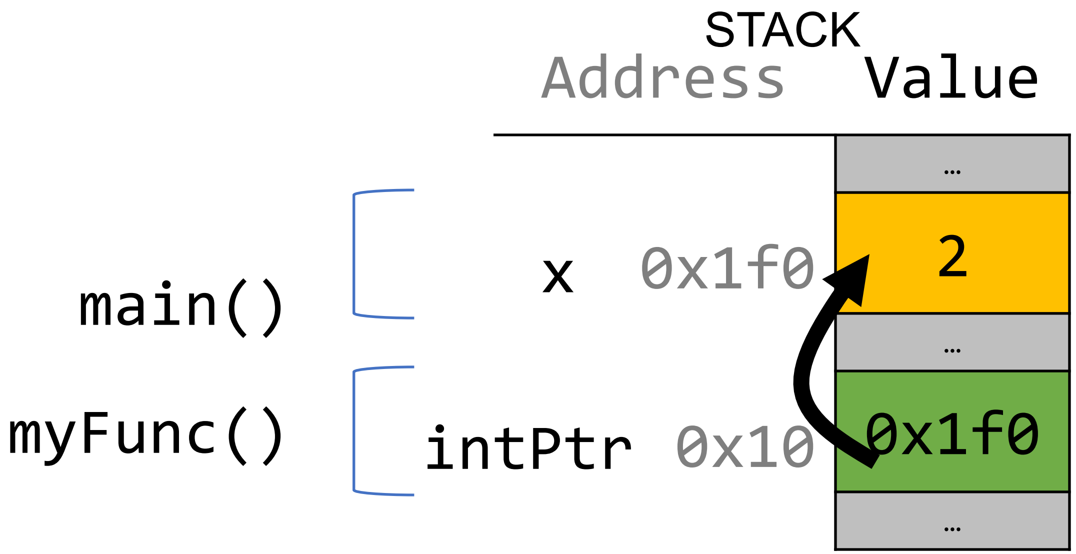
Why does this matter?

- 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?*

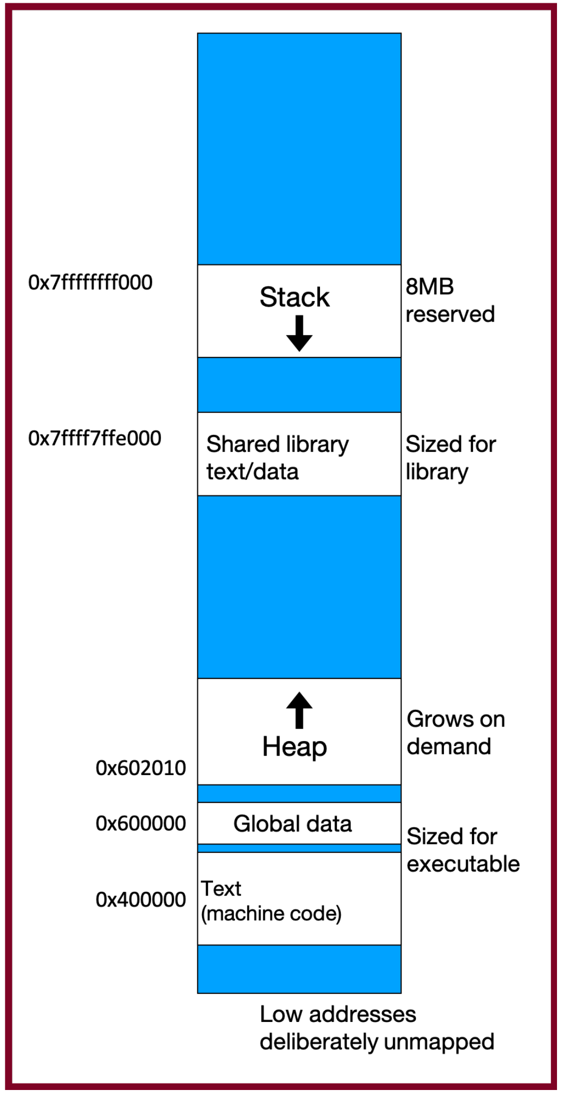
- Arrays let us store ordered lists of information.
- Pointers let us pass addresses of data instead of the data itself.
- We can use the stack, which cleans up memory for us, or the heap, which we must manually manage.



Stack And Heap

Why does this matter?

- The stack and heap allow for two ways to store data 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

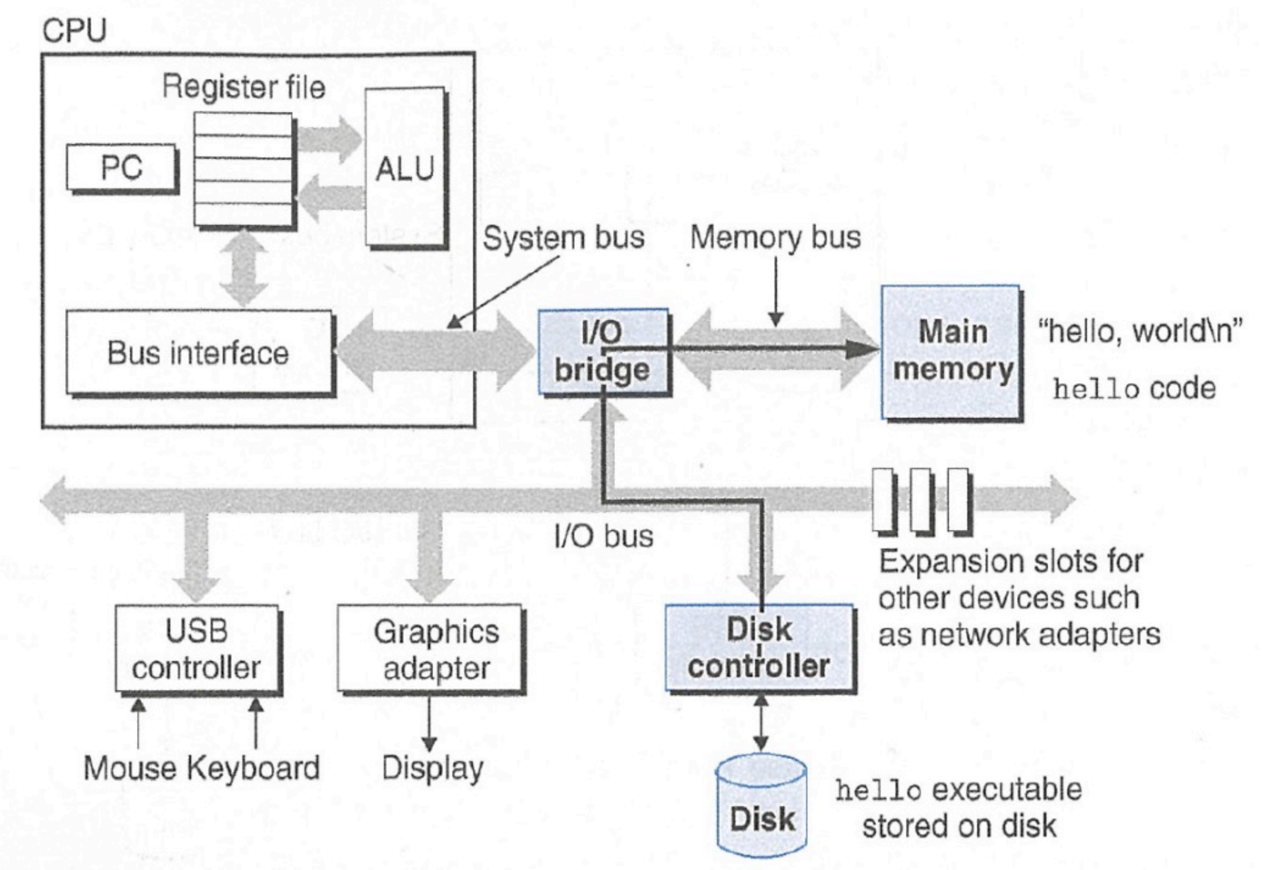
Key Question: *How does a computer interpret and execute C programs?*

- 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

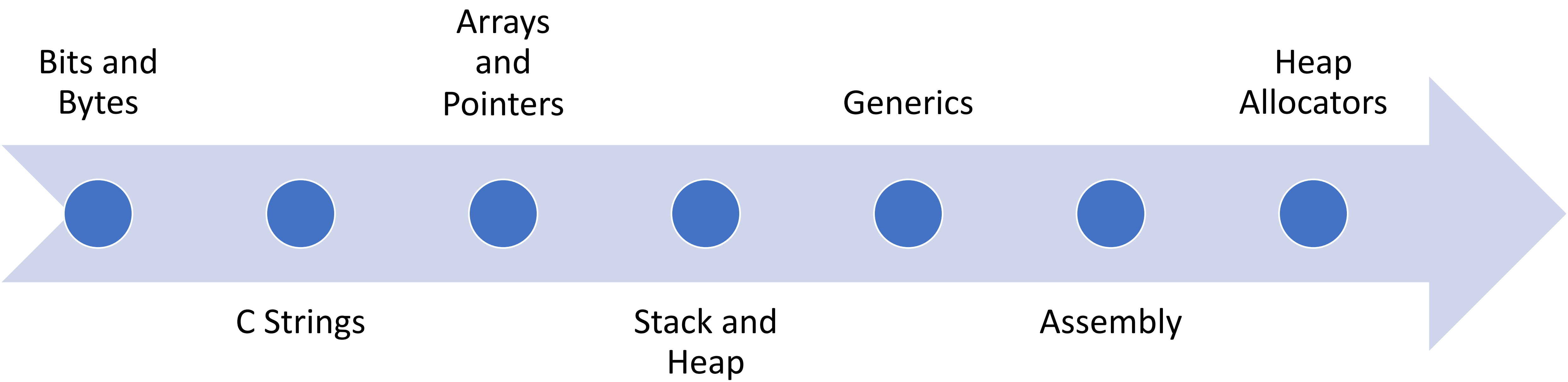
Key Question: *How do core memory-allocation operations like malloc and free work?*

- 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.

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

and have **exposure** to

- a working understanding of the basics of computer architecture

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
  - Bonus: *How does an X86 processor handle floating point numbers?*
- What's Next?
- Q&A

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
  - **CS107 Tools and Techniques**
  - CS107 Concepts
  - Bonus: *How does an X86 processor handle floating point numbers?*
- 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
- 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)
  - Web development: in-browser debugger

Testing (Sanity Check)

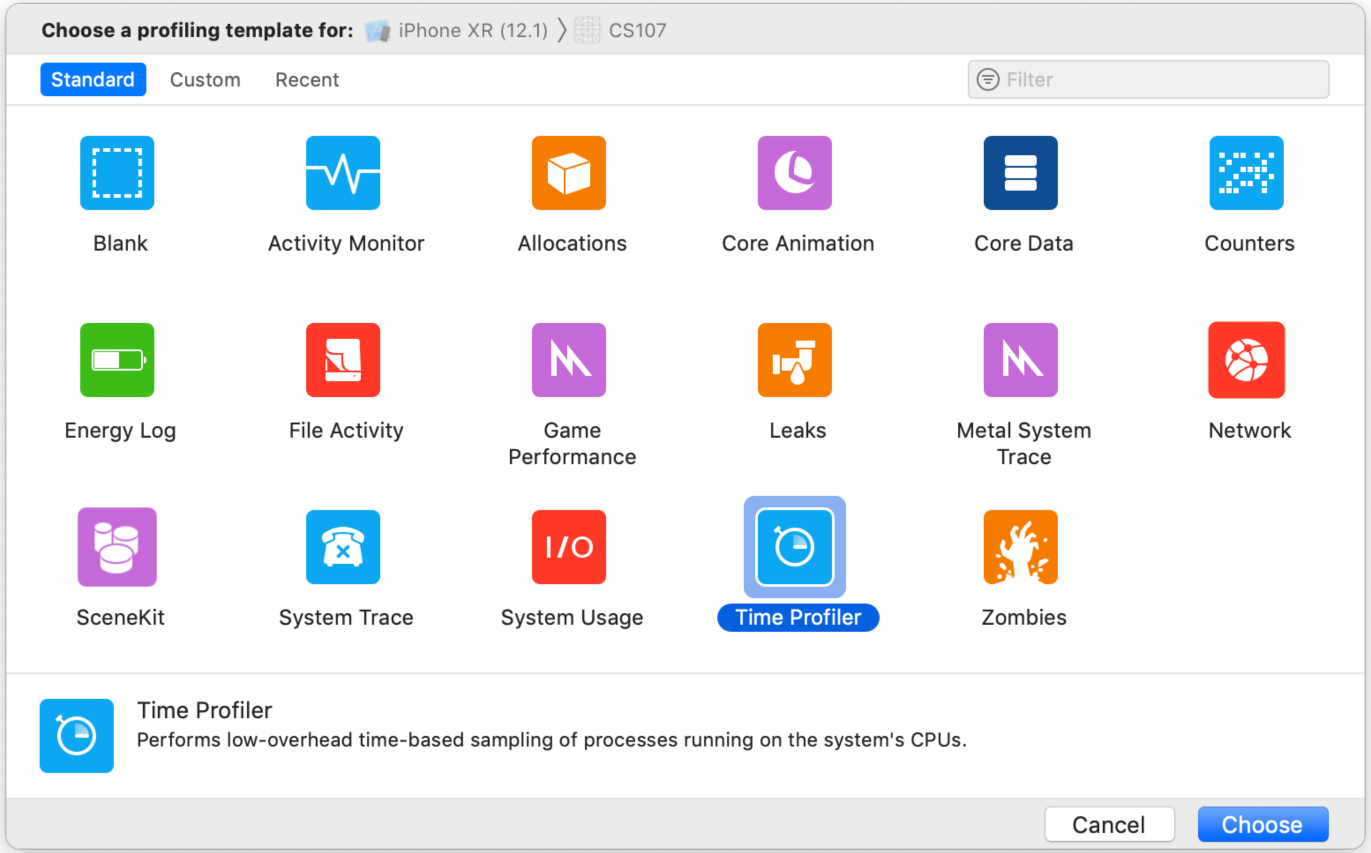
- Testing is a crucial skill for any computer scientist
- <u>Our goal for CS107 was to help you become a better tester</u>
  - Writing targeted tests to validate correctness
  - Use tests to prevent regressions
  - Use tests to develop incrementally

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**
  - Bonus: *How does an X86 processor handle floating point numbers?*
- What's Next?
- Q&A

Concepts

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

Systems

How is an operating system implemented? (take CS140!)
- Threads
- User Programs
- Virtual Memory
- Filesystem

How is a compiler implemented? (take CS143!)
- Lexical analysis
- Parsing
- Semantic Analysis
- Code Generation

How can applications communicate over a network? (take CS110/CS144!)
- How can we weigh different tradeoffs of network architecture design?
- How can we effectively transmit bits across a network?

Systems

How can we write programs that execute multiple tasks simultaneously? (take CS110!)

- Threads of execution
- ”Locks” to prevent simultaneous access

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
- 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?

- 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?

- 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?

- 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

Plan For Today

- **Recap**: Where We've Been
- Larger Applications
  - CS107 Tools and Techniques
  - CS107 Concepts
  - **Bonus:** ***How does an X86 processor handle floating point numbers?***
- What's Next?
- Q&A

Floats and Assembly

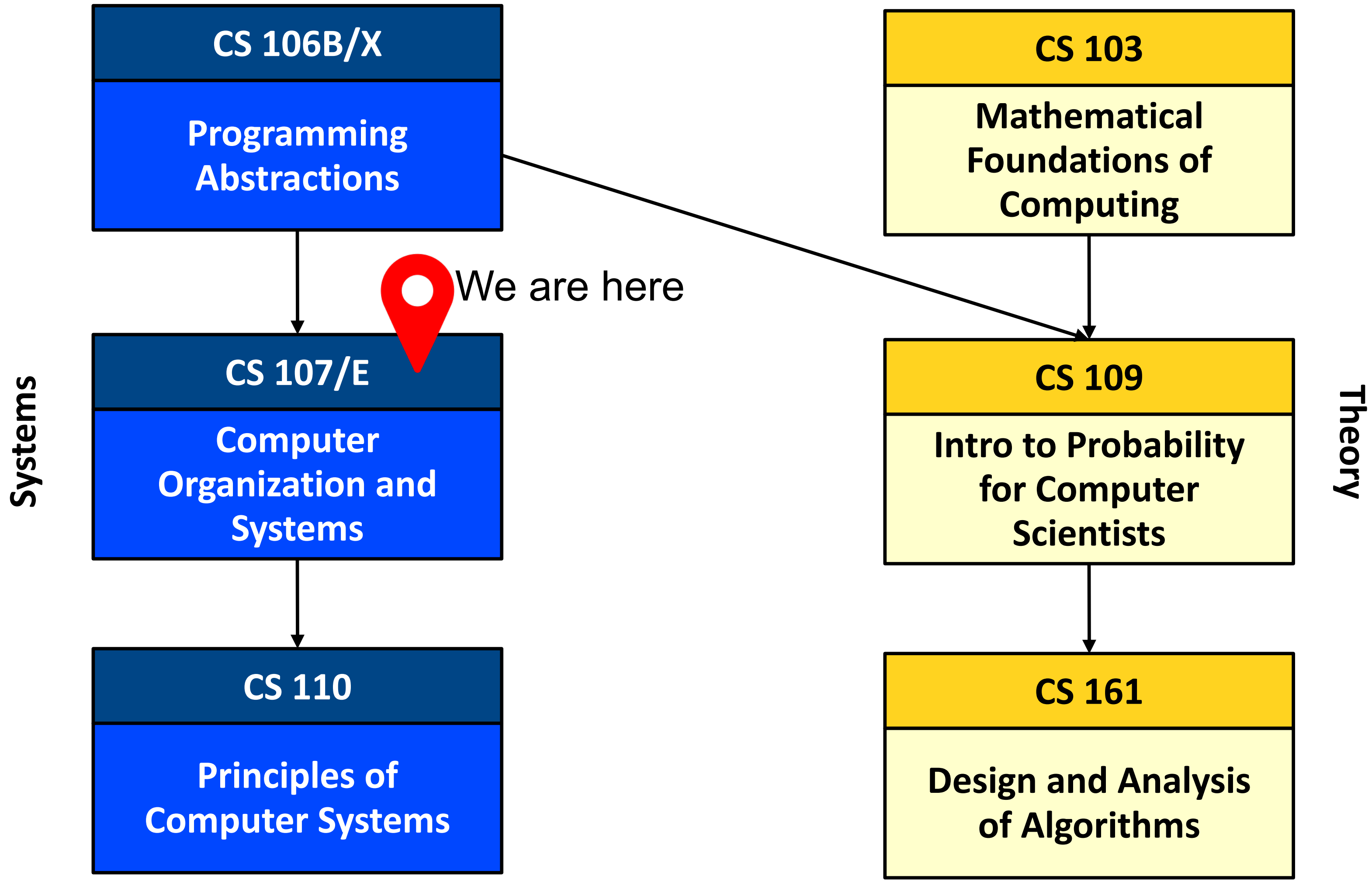
- Unfortunately, we couldn’t cover floating point numbers this quarter
  - Lecture video from past quarter here!
- We wanted to give you a taste of floating point; specifically how we can use our existing knowledge of assembly to better understand it

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?

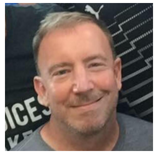
Where Are We?



CS 106B/X
Programming Abstractions
We are here
CS 107/E
Computer Organization and Systems
CS 110
Principles of Computer Systems
Systems
CS 103
Mathematical Foundations of Computing
CS 109
Intro to Probability for Computer Scientists
CS 161
Design and Analysis of Algorithms
Theory

CS 110

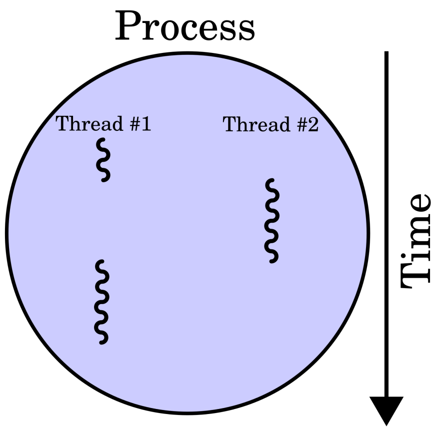
- How can we implement multithreading in our programs?
- How can multiple programs communicate with each other?
- How can we implement distributed software systems to do things like process petabytes of data?
- How can we maximally take advantage of the hardware and operating system software available to us?



Jerry Cain

Chris Gregg

Nick Troccoli



https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)#/media/File:Multithreaded_process.svg

Other Courses

- **CS140**: Operating Systems
- **CS143:** Compilers
- **CS144:** Networking
- **CS145:** Databases
- **CS166:** Data Structures
- **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. 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!