CS107, Lecture 17
Wrap-Up / What's Next?

Plan For Today

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

Plan For Today

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

Final Exam

- The final exam is **Fri. 3/22 12:15-3:15PM in Hewlett 200 and Hewlett 201**
  - **Last names A-G: Hewlett 201**
  - **Last Names H-Z: Hewlett 200**
- Covers all material from the quarter
- Closed-book, 1 2-sided page of notes permitted, C reference sheet provided, Assembly reference sheet provided
- Administered via BlueBook software (on your laptop)
- Practice materials will be available on the course website
- If you need exam accommodations, please do your best to let us know by **Friday 3/15.**
- If you need a laptop for the exam, you **must** let us know by **Friday 3/15**. Limited charging outlets will be available for those who need them.

Plan For Today

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

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

You'll be able to...

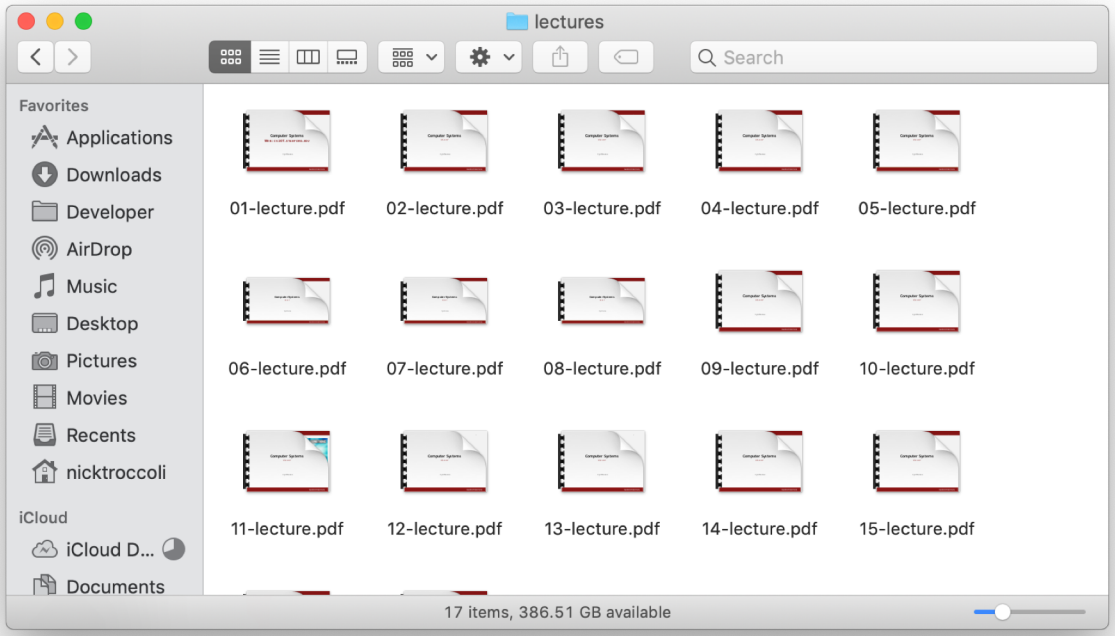
- Manipulate bits and bytes and work within the limits of computer arithmetic
- Implement complex algorithms using C-strings
- Re-implement existing Unix tools yourself
- Write generic C code that works with a variety of data types
- Defuse a "bomb" program without ever seeing its source code
- Find security vulnerabilities in programs
- Create your own memory allocator to manage heap memory

First Day

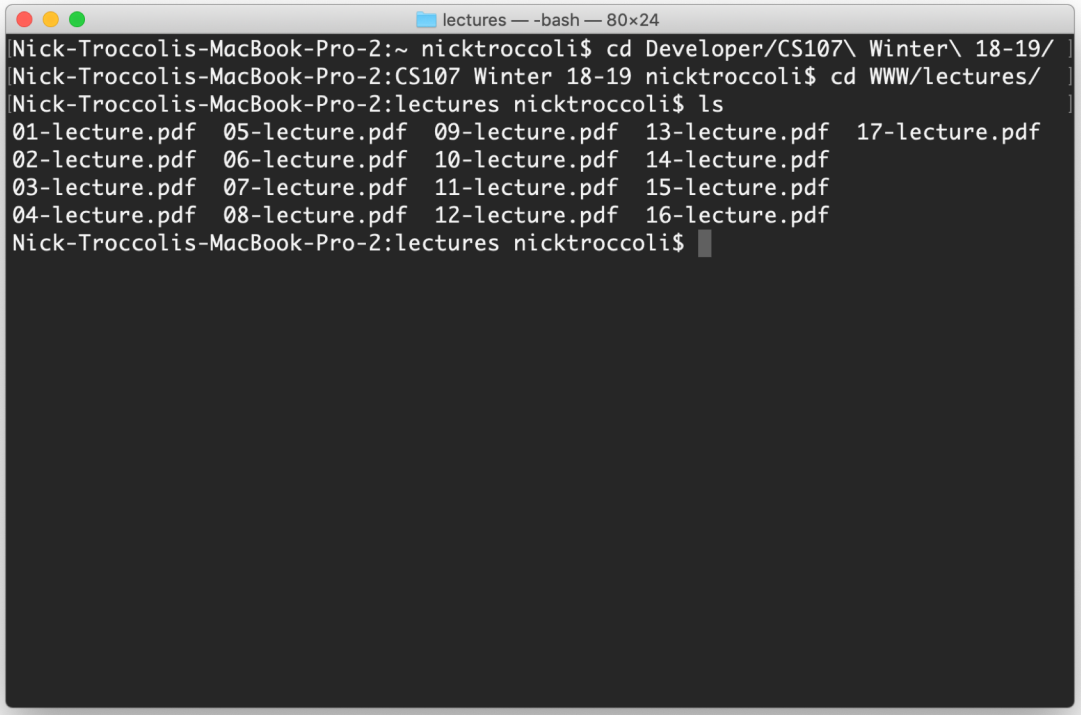
```c
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

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).

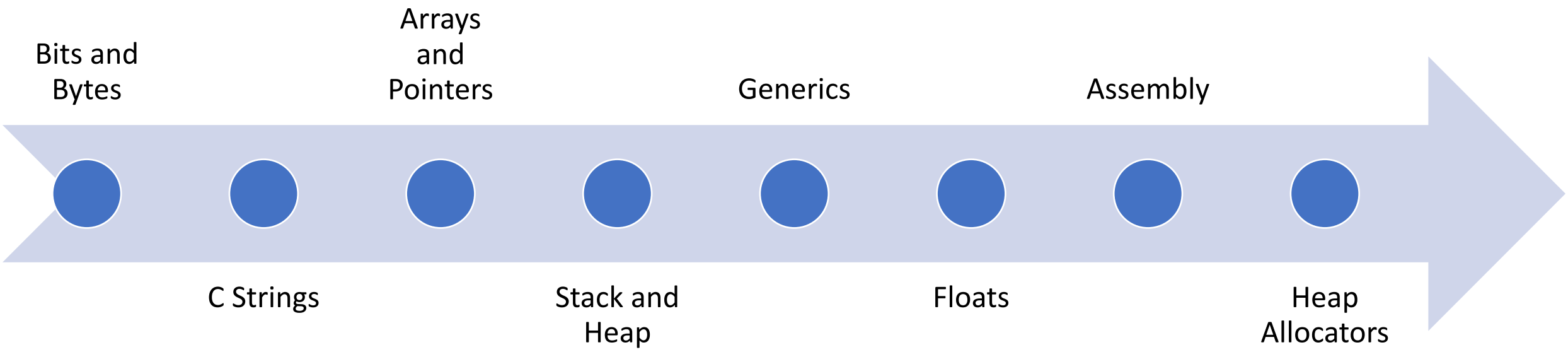


Graphical User Interface



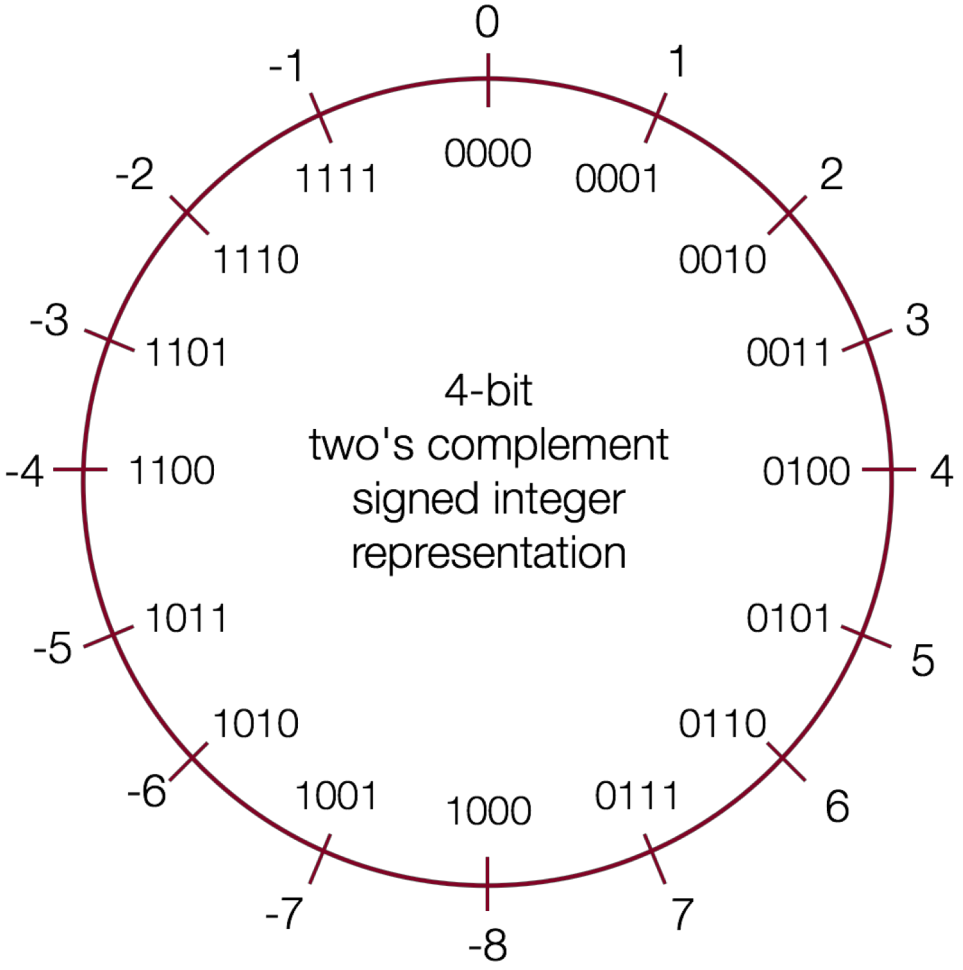
Text-based interface

Our CS107 Journey



Bits And Bytes

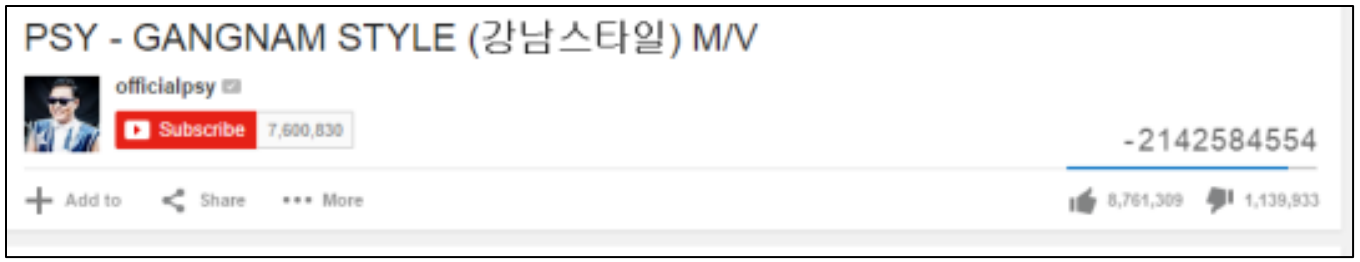
Key Question: how are integer numbers represented by a computer? How do we manipulate this representation?

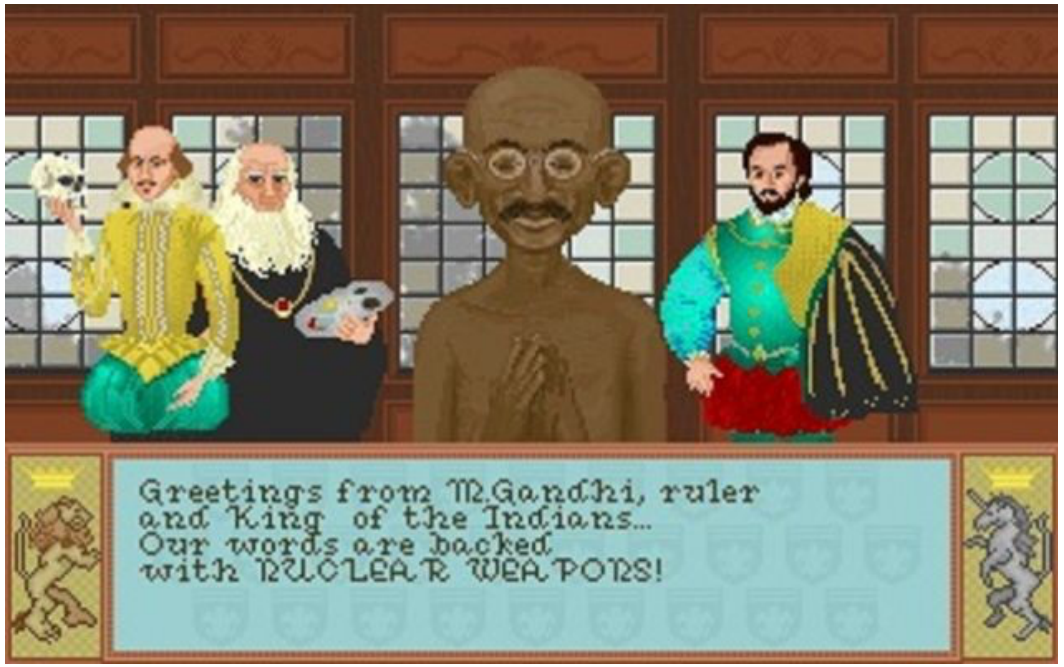


Bits And Bytes

Key Question: why does this matter?

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

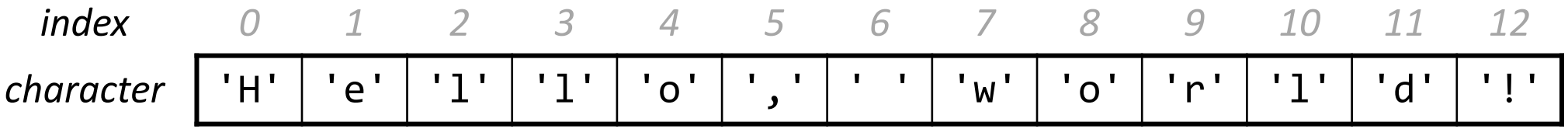




C Strings

Key Question: how can we manipulate text in C programs?

- 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).



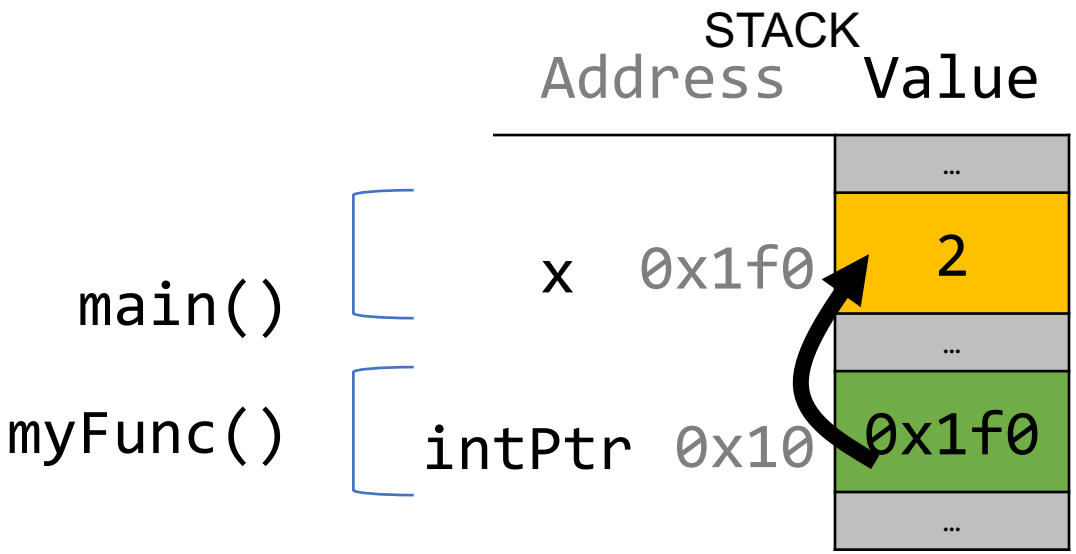
C Strings

Key Question: 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/

Arrays and Pointers

- Arrays let us store ordered lists of information.
- Pointers let us pass addresses of data instead of the data itself.

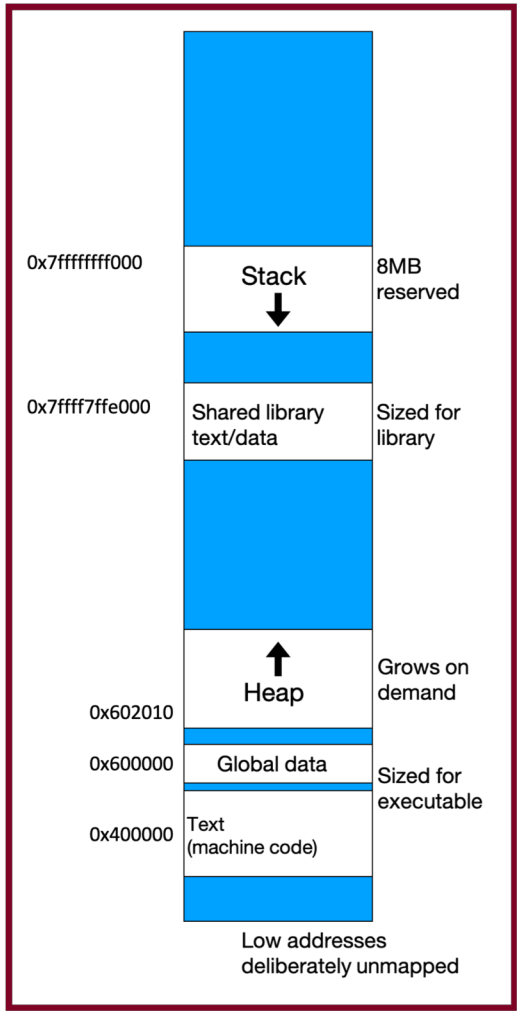


Stack And Heap

Key Question: How can we allocate memory with lifetimes that we want?

Key Question: 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!



Generics

Key Question: how can we write code that works with data of any type?

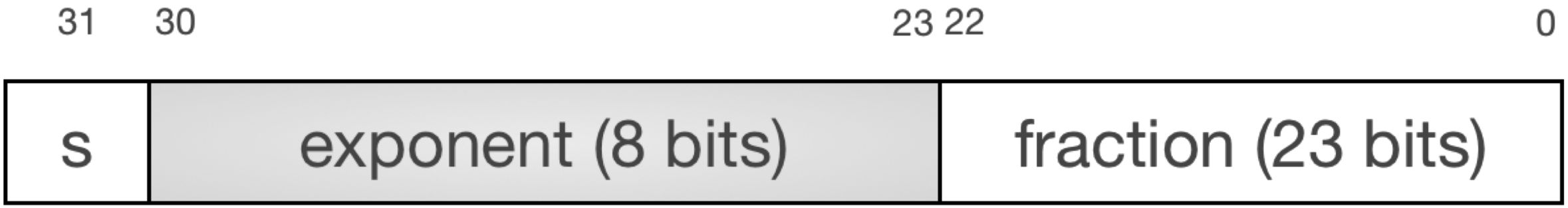
- We can use **void *** to circumvent the type system, **memcpy**, etc. to copy generic data, and function pointers to pass logic around.

Key Question: 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.

Floating Point

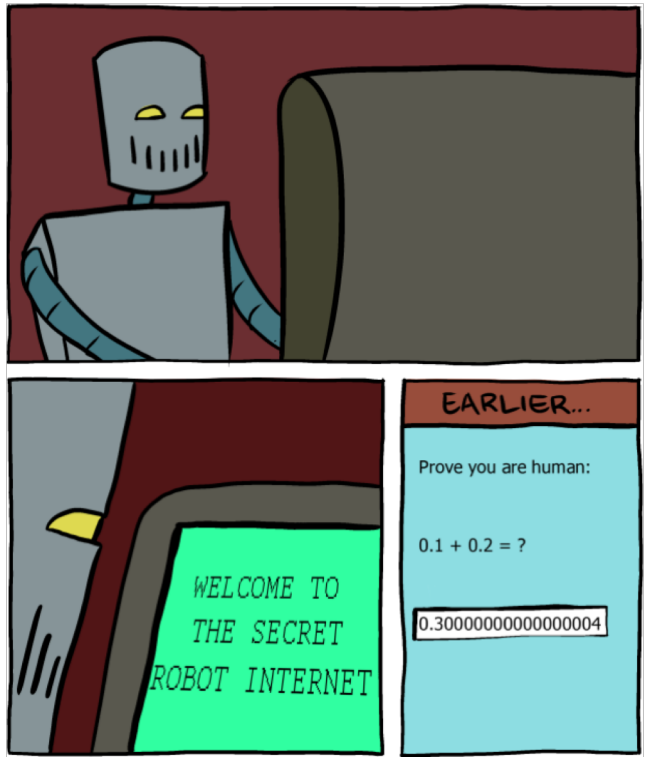
Key Question: how are floating point numbers represented by a computer?



Floating Point

Key Question: why does this matter?

- IEEE floating point represents tradeoffs in representing floats with high precision in a large range. These tradeoffs impact programs!



Assembly Language

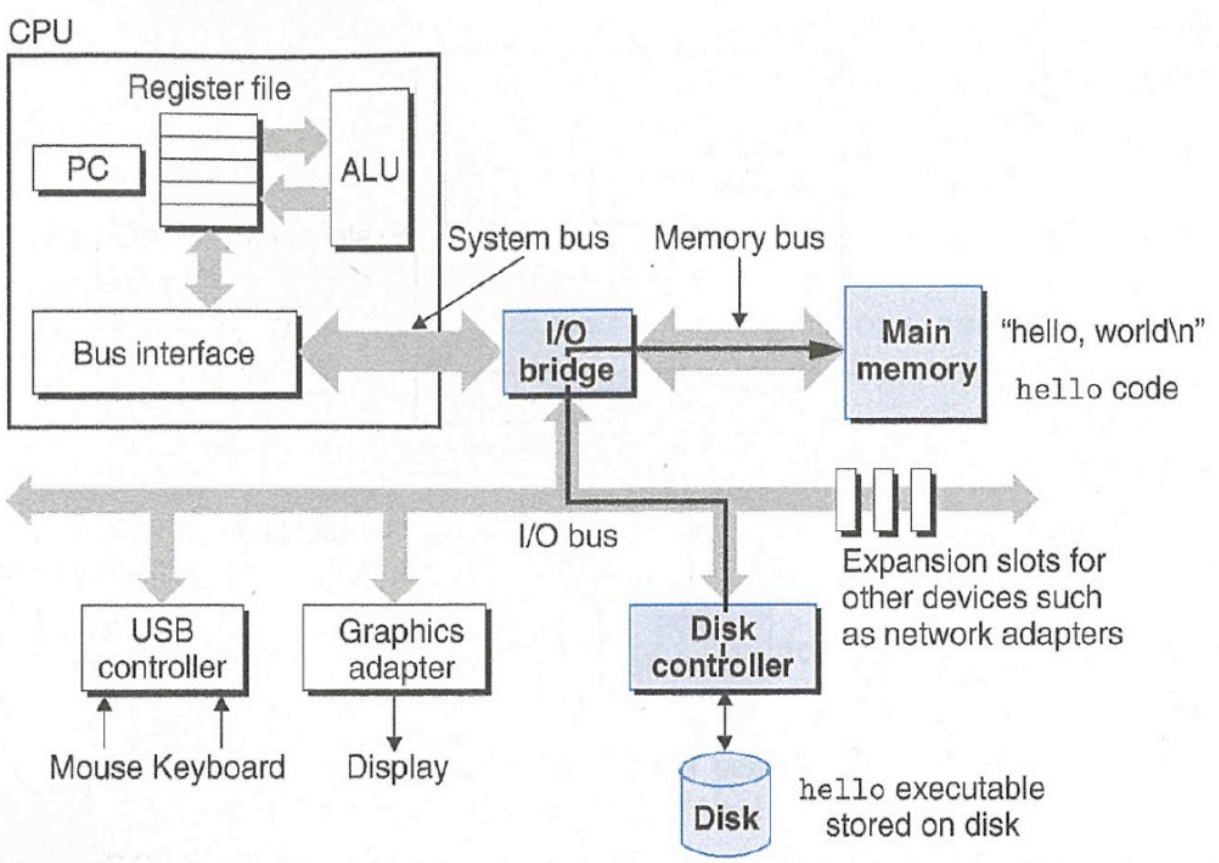
Key Question: how does the computer execute our programs?

- 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

Key Question: 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

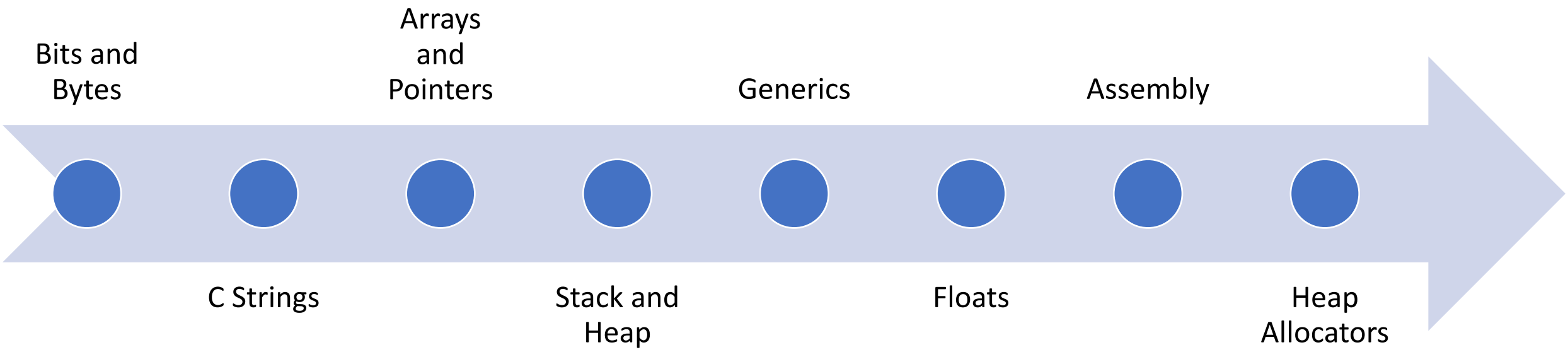
Key Question: how is the heap managed? How do malloc/realloc/free work?

- A *heap allocator* manages a block of memory for the heap and completes requests to use or give up memory space.

Key Question: why does this matter?

- Designing a heap allocator requires making many design decisions to optimize it as much as possible.
- All languages have a “heap”, but manipulate it in different ways.

Our CS107 Journey



Course 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
- writing code that correctly ports to other architectures
- working effectively in a Unix development environment

and have exposure to

- a working understanding of the basics of computer architecture

Plan For Today

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

Plan For Today

- **Info**: Final Exam
- **Recap**: Where We've Been
- Larger Applications
  - CS107 Tools and Techniques
  - CS107 Concepts
- What's Next?
- **Break**
- 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.)

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 project
- 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)
  - Web development: in-browser debugger

Testing (Sanity Check)

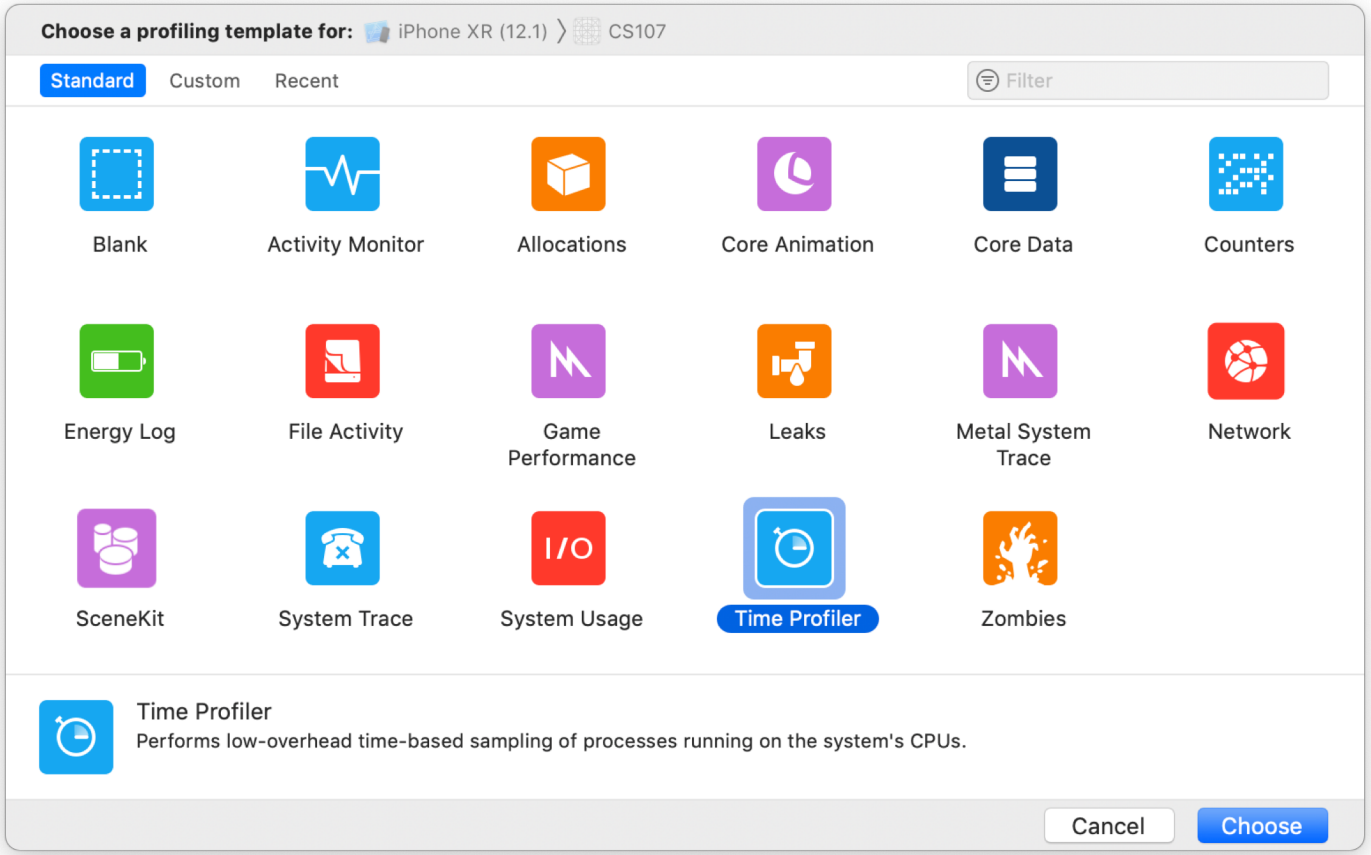
- Testing is a crucial skill for any computer scientist
- Our goal for CS107 was to help you become a better tester
  - Writing targeted tests to validate correctness
  - Use tests to prevent regressions
  - Use tests to develop incrementally

Memory 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!



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 (Python/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

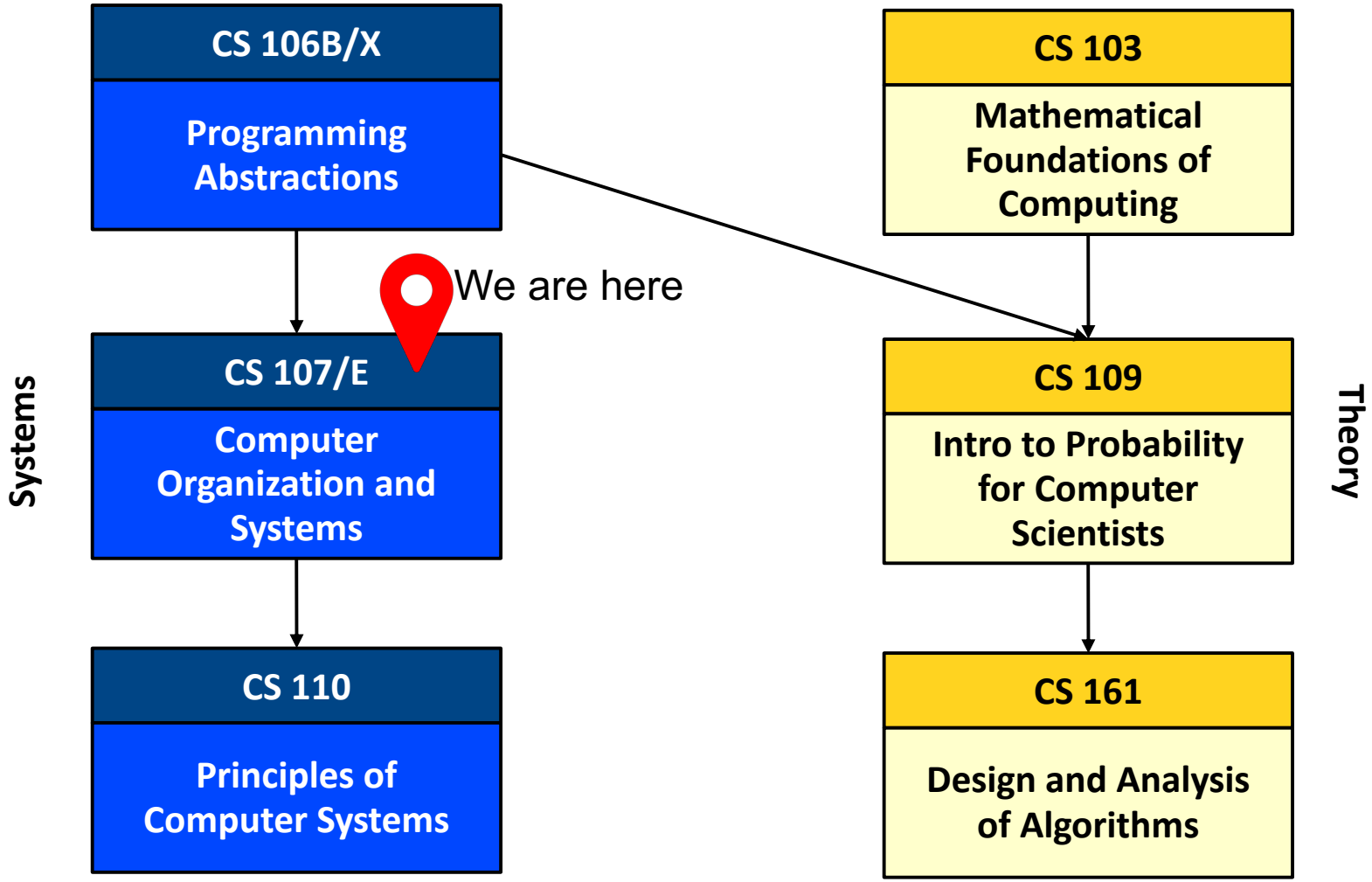
Plan For Today

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

What's Next?

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

Where Are We?

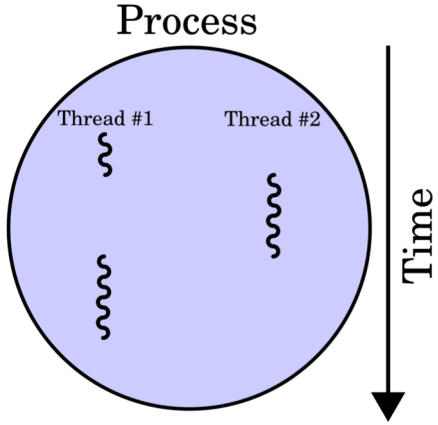

CS 106B/X
Programming Abstractions
CS 103
Mathematical Foundations of Computing
We are here
CS 107/E
Computer Organization and Systems
CS 109
Intro to Probability for Computer Scientists
CS 110
Principles of Computer Systems
CS 161
Design and Analysis of Algorithms
Systems
Theory

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



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

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

Plan For Today

- **Info**: Final Exam
- **Recap**: Where We've Been
- Larger Applications
- What's Next?
- **Break**
- 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!