Course Wrap-Up
CSE 333 Spring 2020

Instructor: Hal Perkins

Teaching Assistants:

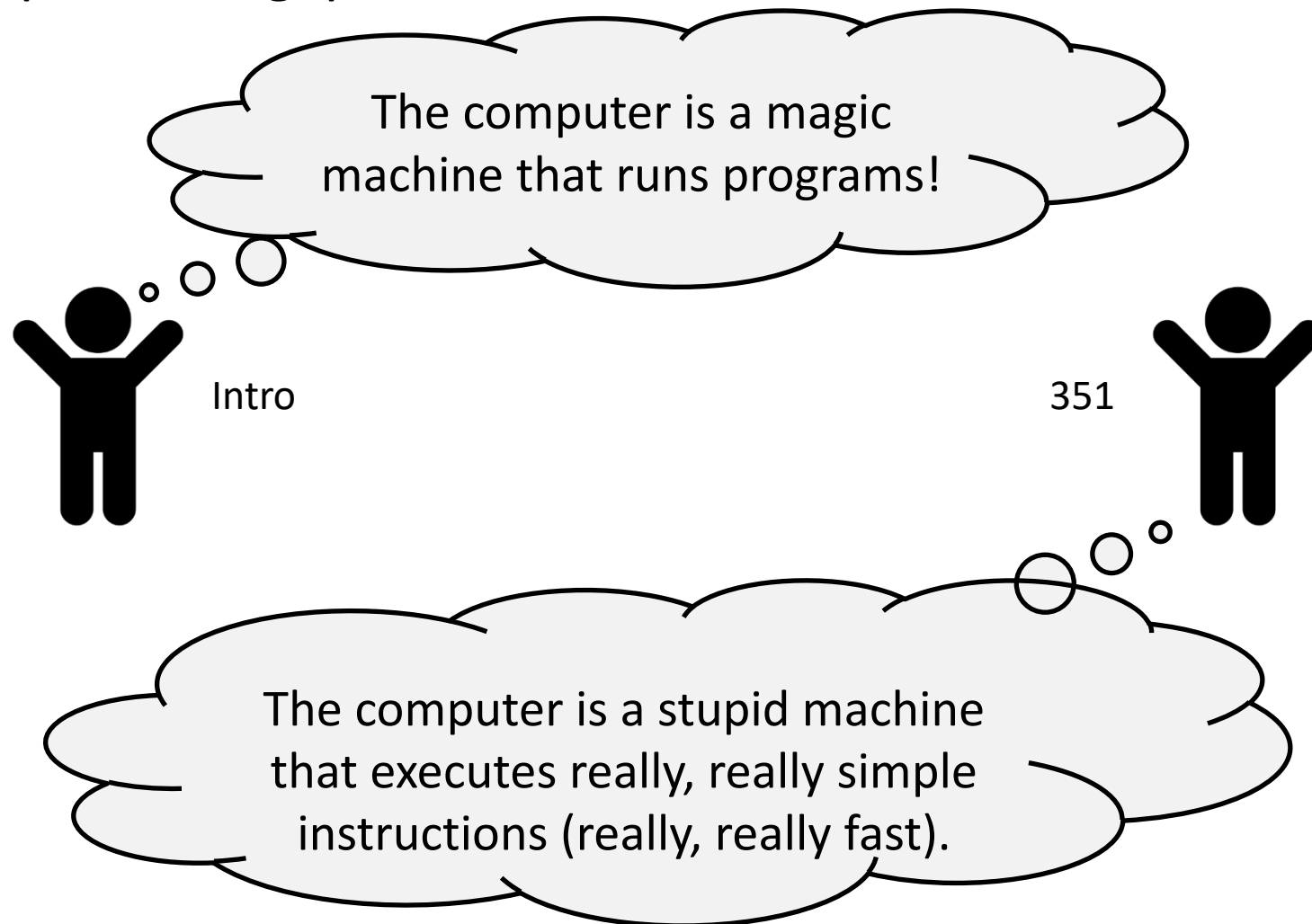|		
Ramya Challa	Mengqi Chen	John Depaszthory
Greg Guo	Zachary Keyes	CJ Lin
Travis McGaha	Arjun Singh	Guramrit Singh
Cosmo Wang	Yifan Xu	Robin Yang
Haoran Yu	Velocity Yu	

Administrivia

❖ hw4 "due" last night, but really due 11 pm Saturday because everyone has 2 free late days

  ▪ Give it the best effort you can under the circumstances; we'll do our best to award substantial partial credit if not done perfectly, and we'll also work to help individuals as needed given the stressful situation around us

❖ Please finish course evals while they are still available

❖ Please nominate great TAs for the Bandes award. Thanks.

  ▪ Both for CSE 333 and for other courses

So what have we been doing for the last 10 weeks?
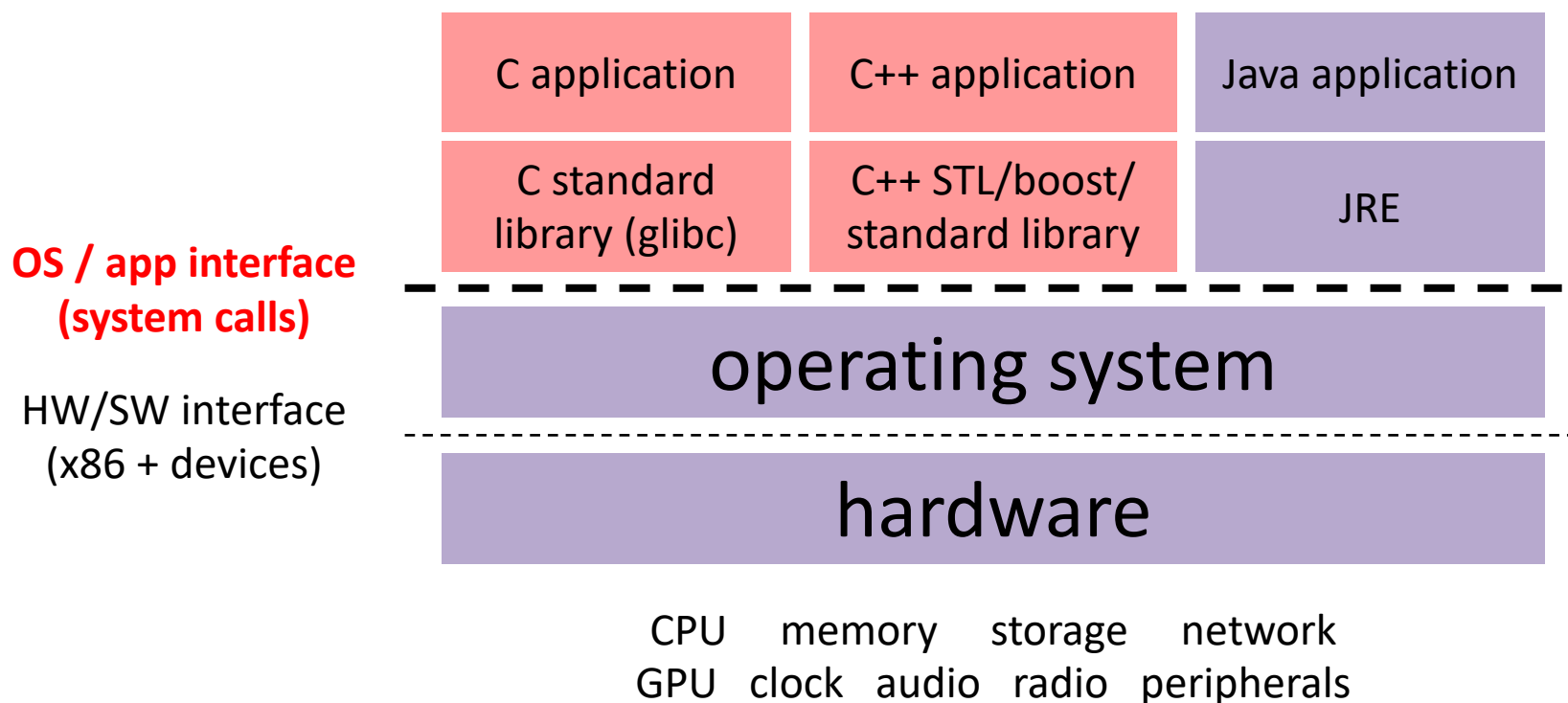
?

Course Goals

❖ Explore the gap between:

The computer is a magic machine that runs programs!

Intro   351

The computer is a stupid machine that executes really, really simple instructions (really, really fast).

Course Map: 100,000 foot view

C application	C++ application	Java application
C standard library (glibc)	C++ STL/boost/ standard library	JRE

OS / app interface (system calls)

- -

operating system

HW/SW interface (x86 + devices)

· ·

hardware

CPU   memory   storage   network
GPU clock audio radio peripherals

Systems Programming

❖ The programming skills, engineering discipline, and knowledge you need to build a system

- **Programming:** C / C++

- **Discipline:** design, testing, debugging, performance analysis

- **Knowledge:** long list of interesting topics
  - Concurrency, OS interfaces and semantics, techniques for consistent data management, distributed systems algorithms, …
  - Most important: a deep understanding of the "layer below"

Main Topics

❖ C

- Low-level programming language

❖ C++

- The 800-lb gorilla of programming languages
- "better C" + classes + STL + smart pointers + …

❖ Memory management

❖ System interfaces and services

❖ Networking basics – TCP/IP, sockets, …

❖ Concurrency basics – POSIX threads, synchronization

The C/C++ Ecosystem

❖ System layers:

 ▪ C/C++

 ▪ Libraries

 ▪ Operating system

❖ Building Programs:

 ▪ Pre-processor (`cpp`, `#include`, `#ifndef`, …)

 ▪ Compiler: source code → object file (`.o`)

 ▪ Linker: object files + libraries → executable
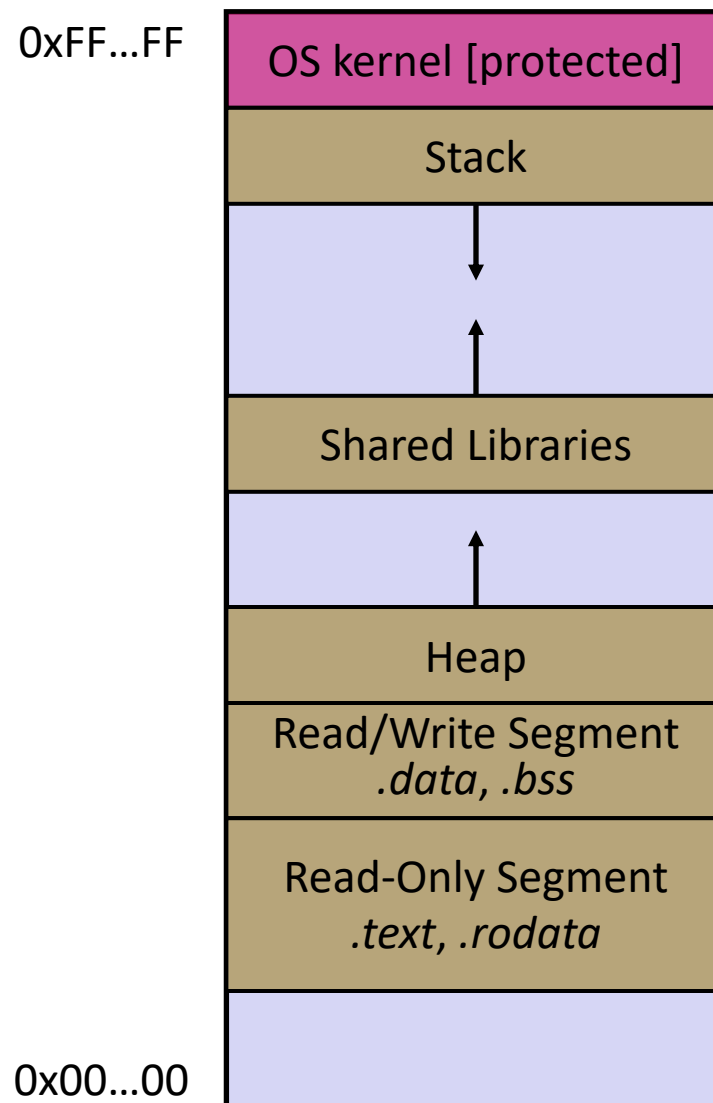
❖ Build tools:

 ▪ `make` and related tools

 ▪ Dependency graphs

Program Execution

❖ What's in a process?

- Address space

- Current state

  - SP, PC, register values, etc.

- Thread(s) of execution

- Environment

  - Arguments, open files, etc.

0xFF...FF

OS kernel [protected]
Stack
↓   ↑
Shared Libraries
↑
Heap
Read/Write Segment *.data*, *.bss*
Read-Only Segment *.text*, *.rodata*

0x00...00

Structure of C Programs

❖ Standard types and operators

  ▪ Primitives, extended types, structs, arrays, typedef, etc.

❖ Functions

  ▪ Defining, invoking, execution model

❖ Standard libraries and data structures

  ▪ Strings, streams, etc.

  ▪ C standard library and system calls, how they are related

❖ Modularization

  ▪ Declaration vs. definition

  ▪ Header files and implementations

  ▪ Internal vs. external linkage

❖ Handling errors without exception handling

  ▪ `errno` and return codes

C++ (and C++11)

❖ A "better C"

- More type safety, stream objects, memory management, etc.

❖ References and const

❖ Classes and objects!

- So much (too much?) control: constructor, copy constructor, assignment, destructor, operator overloading

- Inheritance and subclassing

    - Dynamic vs. static dispatch, virtual functions, vtables and vptrs
    - Pure virtual functions and abstract classes
    - Subobjects and slicing on assignment

❖ Copy semantics vs. move semantics

C++ (and C++11)

- ❖ C++ Casting
  - ▪ What are they and why do we distinguish between them?
  - ▪ Implicit conversion/construction and `explicit`

- ❖ Templates – parameterized classes and functions
  - ▪ Similarities and differences from Java generics
  - ▪ Template implementations via expansion

- ❖ STL – containers, iterators, and algorithms
  - ▪ `vector`, `list`, `map`, `set`, etc.
  - ▪ Copying and types

- ❖ Smart Pointers
  - ▪ `unique_ptr`, `shared_ptr`, `weak_ptr`
  - ▪ Reference counting and resource management

Memory

❖ Object scope and lifetime

  ▪ *Static*, *automatic*, and *dynamic* allocation / lifetime

❖ Pointers and associated operators (`&`, `*`, `->`, `[]`)

  ▪ Can be used to link data or fake "call-by-reference"

❖ Dynamic memory allocation

  ▪ **malloc**/**free** (C), **new**/**delete** (C++)

  ▪ Who is responsible? Who owns the data? What happens when (not if) you mess this up? (dangling pointers, memory leaks, …)

❖ Tools

  ▪ Debuggers (`gdb`), monitors (`valgrind`)

  ▪ Most important tool: thinking!

Networking

❖ **Conceptual abstraction layers**

  ▪ Physical, data link, network, transport, session, presentation, application

  ▪ Layered *protocol* model

    • We focused on IP (network), TCP (transport), and HTTP (application)

❖ **Network addressing**

  ▪ MAC addresses, IP addresses (IPv4/IPv6), DNS (name servers)

❖ **Routing**

  ▪ Layered packet payloads, security, and reliability

Network Programming

Client side

1) Get remote host IP address/port

2) Create socket

3) Connect socket to remote host

4) Read and write data

5) Close socket

Server side

1) Get local host IP address/port

2) Create socket

3) Bind socket to local host

4) Listen on socket

5) Accept connection from client

6) Read and write data

7) Close socket

Concurrency

❖ Why or why not?
  - Better throughput, resource utilization (CPU, I/O controllers)
  - Tricky to get right – harder to code and debug

❖ Threads – "lightweight"
  - Address space sharing; separate stacks for each thread
  - Standard C/C++ library: pthreads

❖ Processes – "heavyweight"
  - Isolated address spaces
  - Forking functionality provided by OS

❖ Synchronization
  - Data races, locks/mutexes, how much to lock…

Processes vs Threads on One Slide
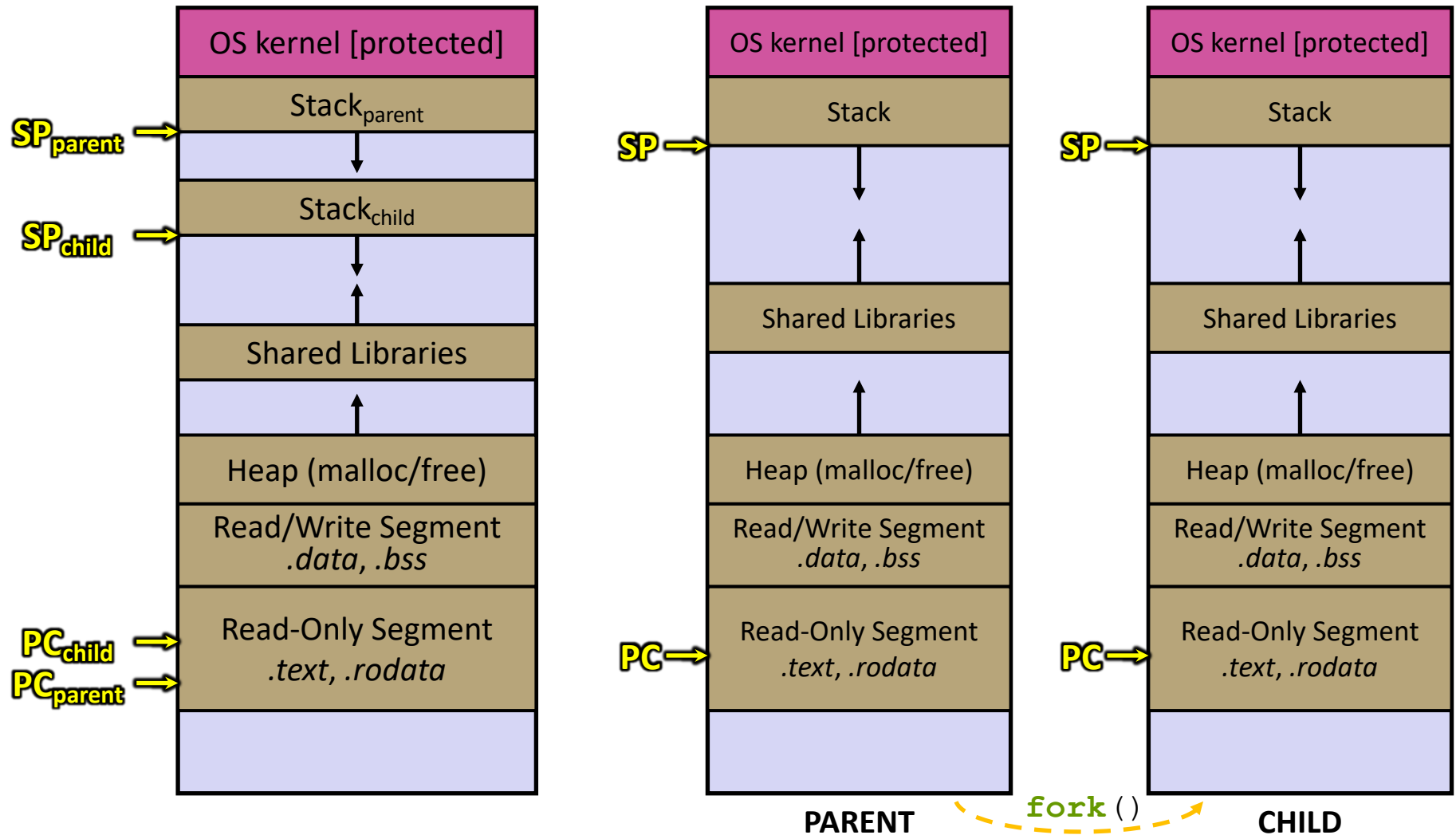
Phew! That's it!

❖ But that's a lot!!

❖ Take a look back and congratulate yourself on what you've accomplished – particularly during this difficult quarter

Courses: What's Next?

❖ **CSE401:** Compilers (pre-reqs: 332, 351)

  ▪ *Finally* understand why a compiler does what it does

❖ **CSE451:** Operating Systems (pre-reqs: 332, 333)

  ▪ How do you manage all of the computer's resources?

❖ **CSE452:** Distributed Systems (pre-reqs: 332, 333)

  ▪ How do you get large collections of computers to collaborate (correctly!)?

❖ **CSE461:** Networks (pre-reqs: 332, 333)

  ▪ The networking nitty-gritty: encoding, transmission, routing, security

❖ **CSE455:** Computer Vision

❖ **CSE457:** Computer Graphics

This doesn't happen without lots of help…

❖ Thanks to a fantastic staff – it can't work without them!!

Ramya Challa	Mengqi Chen	John Depaszthory
Greg Guo	Zachary Keyes	CJ Lin
Travis McGaha	Arjun Singh	Guramrit Singh
Cosmo Wang	Yifan Xu	Robin Yang
Haoran Yu	Velocity Yu	

❖ And thanks to the folks who put the course together:

▪ Steve Gribble, John Zahorjan, me, Justin Hsia, Hannah Tang

And thanks to…

You

It has been a tough quarter – first a pandemic and remote classes, and then recent events, which have affected all of us. You should be proud of your resilience and what you've done. Please take care of yourself, stay active, help yourself, your friends, your community. There are a lot of problems that need work, and we all need to be part of the solution.

Congratulations and best wishes!

You've learned a *lot* – go out and build great things!

Come by and say hello in the future – I'd love to know what you've been up to after CSE 333!

(and I'd actually like to meet everyone in person when we can!)