

Course Wrap-Up

CSE 333 Winter 2019

Instructor: Hal Perkins

Teaching Assistants:

Alexey Beall

Renshu Gu

Harshita Neti

David Porter

Forrest Timour

Soumya Vasisht

Yifan Xu

Sujie Zhou

Administrivia

- ❖ hw4 due last night
 - (late days – max 2 – expire Saturday; if you have them)

- ❖ Please finish course evals while they are still available

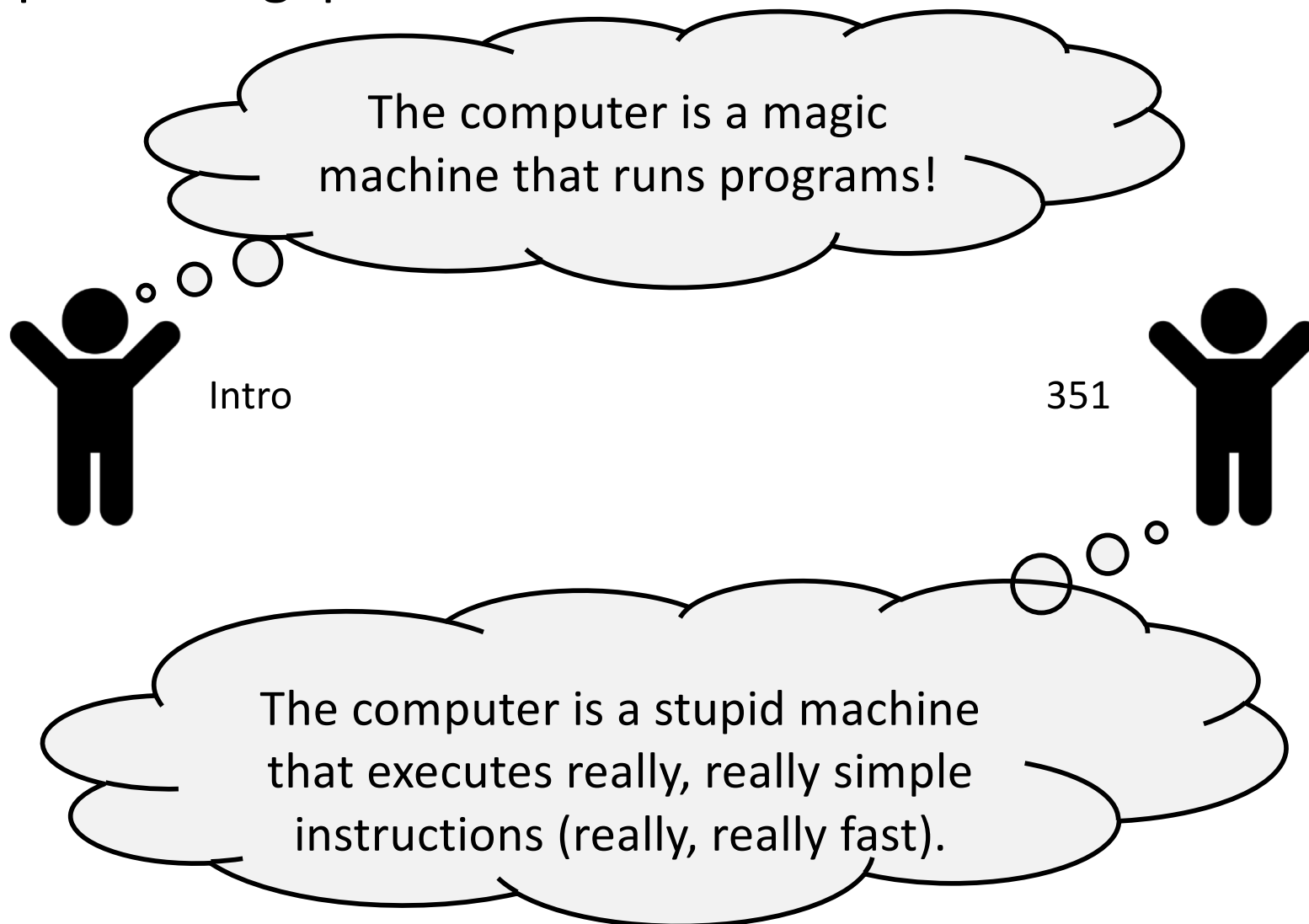
- ❖ Final exam Wed. 3/20, 2:30-4:20
 - Review Q&A Tue. 3/19, 4:30, ECE 125
 - Topic list and old finals on Exams page
 - Summer final exams are 1 hour; regular quarters are usual 2 hours

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

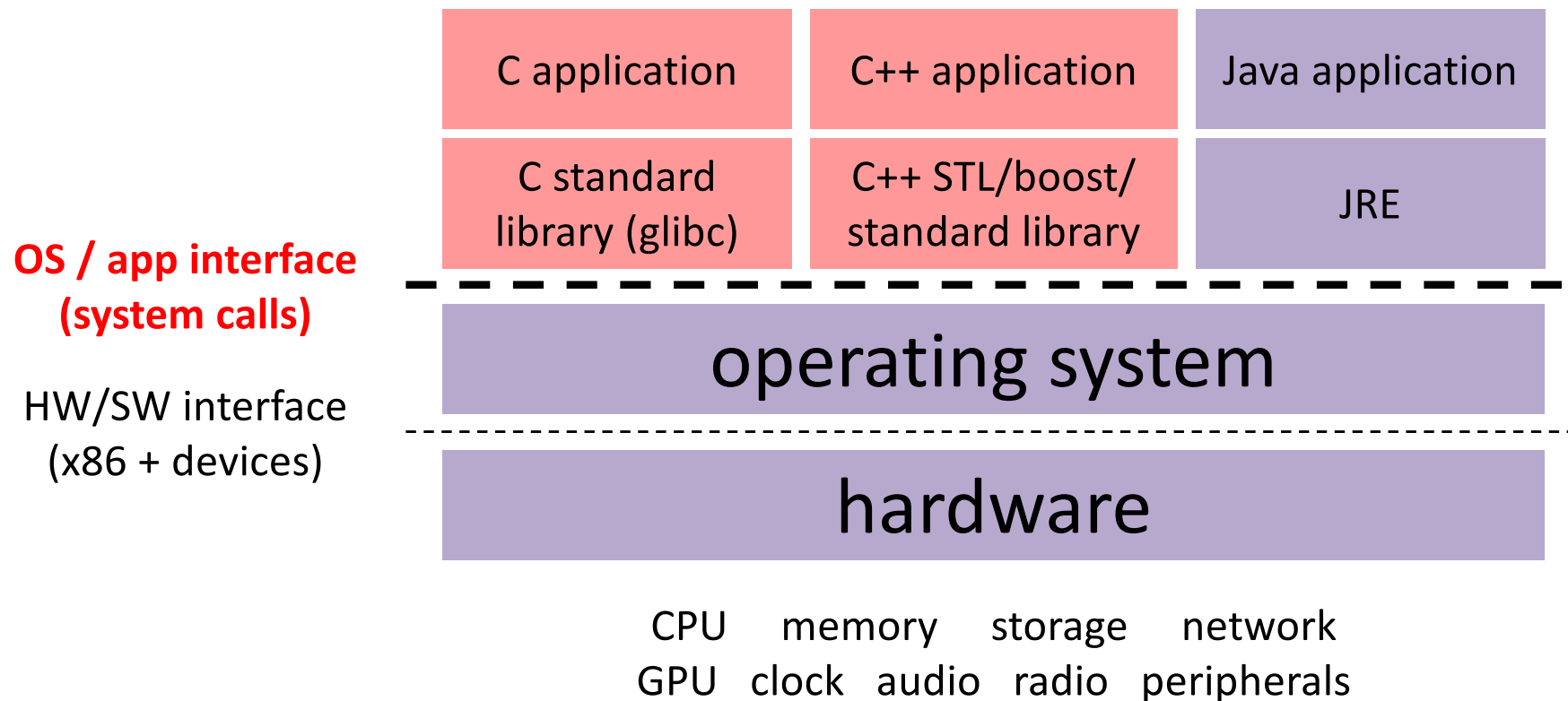


Course Goals

- ❖ Explore the gap between:



Course Map: 100,000 foot view



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

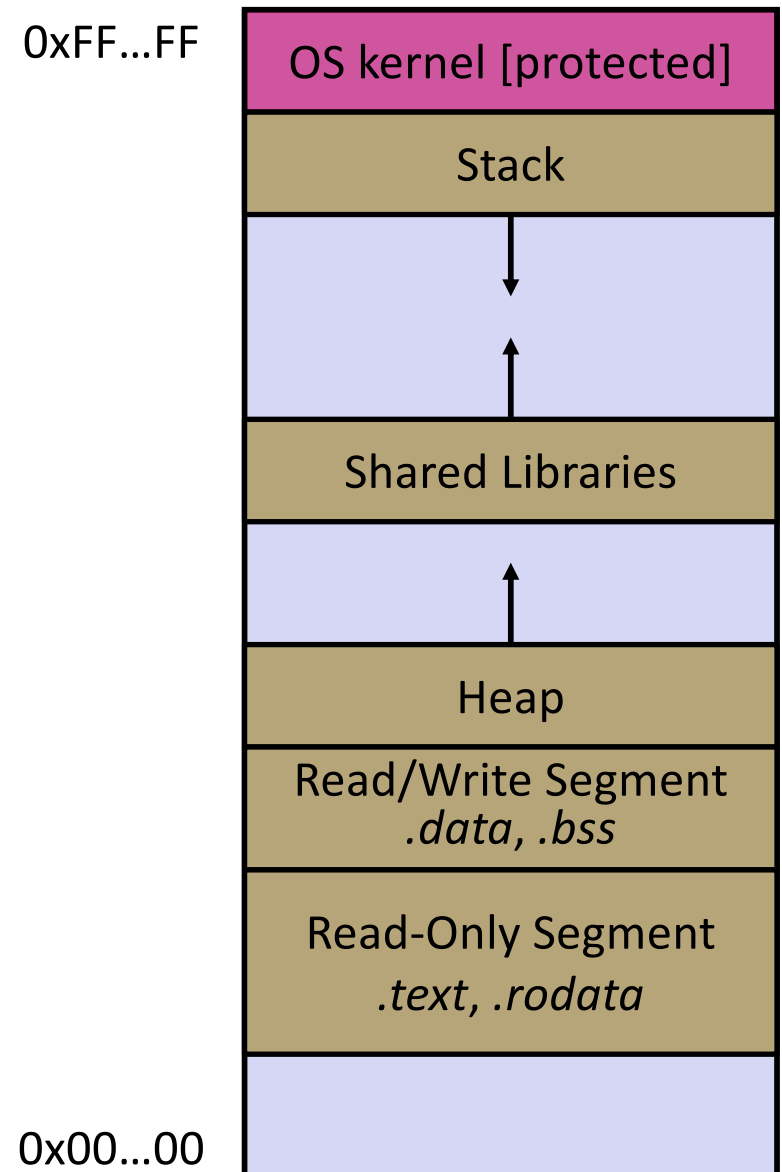
- ❖ C
 - Low-level programming language
- ❖ C++
 - The 800-lb gorilla of programming languages
 - “better C” + classes + STL + smart pointers + ...
- ❖ Memory management
- ❖ System interfaces and services
- ❖ Networking basics – TCP/IP, sockets, ...
- ❖ Concurrency basics – POSIX threads, synchronization

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.



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

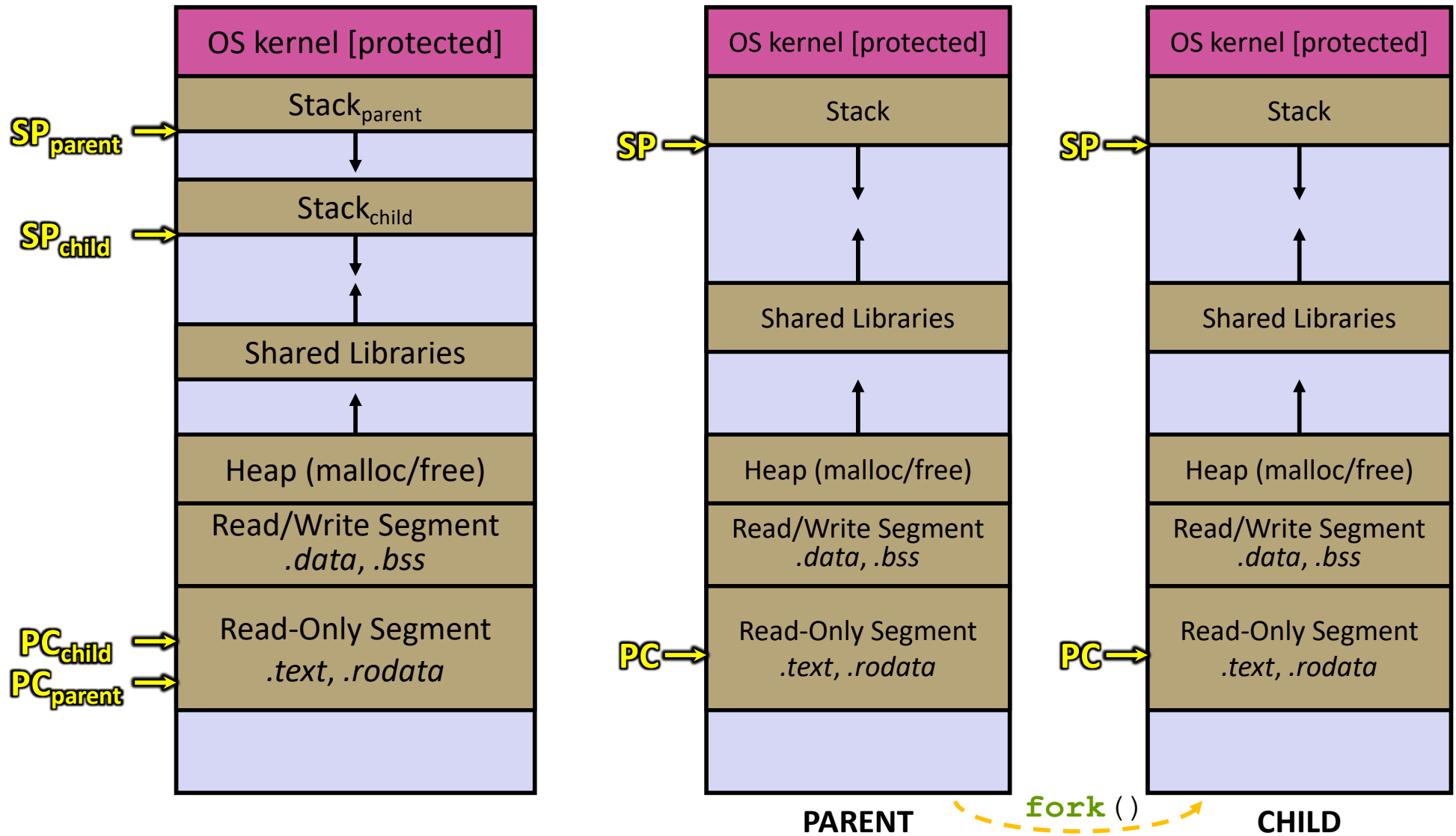
Server side

- 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!!
- ❖ Studying for the exam: (your mileage may vary)
 - Review *first*, make notes
 - Review lecture slides, exercises, sections, end-of-lecture problems
 - Look at topic list on website to check your coverage and help organize
 - Brainstorm and trade ideas with other students
 - “Simulate” an old exam
 - Do it in one timed sitting
 - Working problems is far more important than reading old answers!
 - “Grade” yourself, then go back and review problems
 - If still unsure why, ask the staff or your fellow students
 - Rinse and repeat!

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 great staff – it can't work without them!!

Alexey Beall Renshu Gu Harshita Neti

David Porter Forrest Timour Soumya Vasisht

Yifan Xu Sujie Zhou

- ❖ And thanks to the folks who put the course together:
 - Steve Gribble, John Zahorjan, me, Justin Hsia

Congratulations! Good luck on the exam!

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 333!

See you on Wednesday!



That's all Folks!