Course Wrap-Up

CSE 333 Autumn 2019

Instructor: Hannah C. Tang

Teaching Assistants:

Dao Yi
Farrell Fileas
Lukas Joswiak
Nathan Lipiarski
Renshu Gu
Travis McGaha
Yibo Cao
Yifan Bai
Yifan Xu

Course Evals

- Please do them as you file in!
  - If you're done, check pollev.com/cse333 for a little surprise ☺

- Lecture: https://uw.iasystem.org/survey/215106
  Section AA: https://uw.iasystem.org/survey/216645
  Section AB: https://uw.iasystem.org/survey/216653
  Section AC: https://uw.iasystem.org/survey/216627
  Section AD: https://uw.iasystem.org/survey/216638

- As of 11am, we're at 28 responses right now 😭

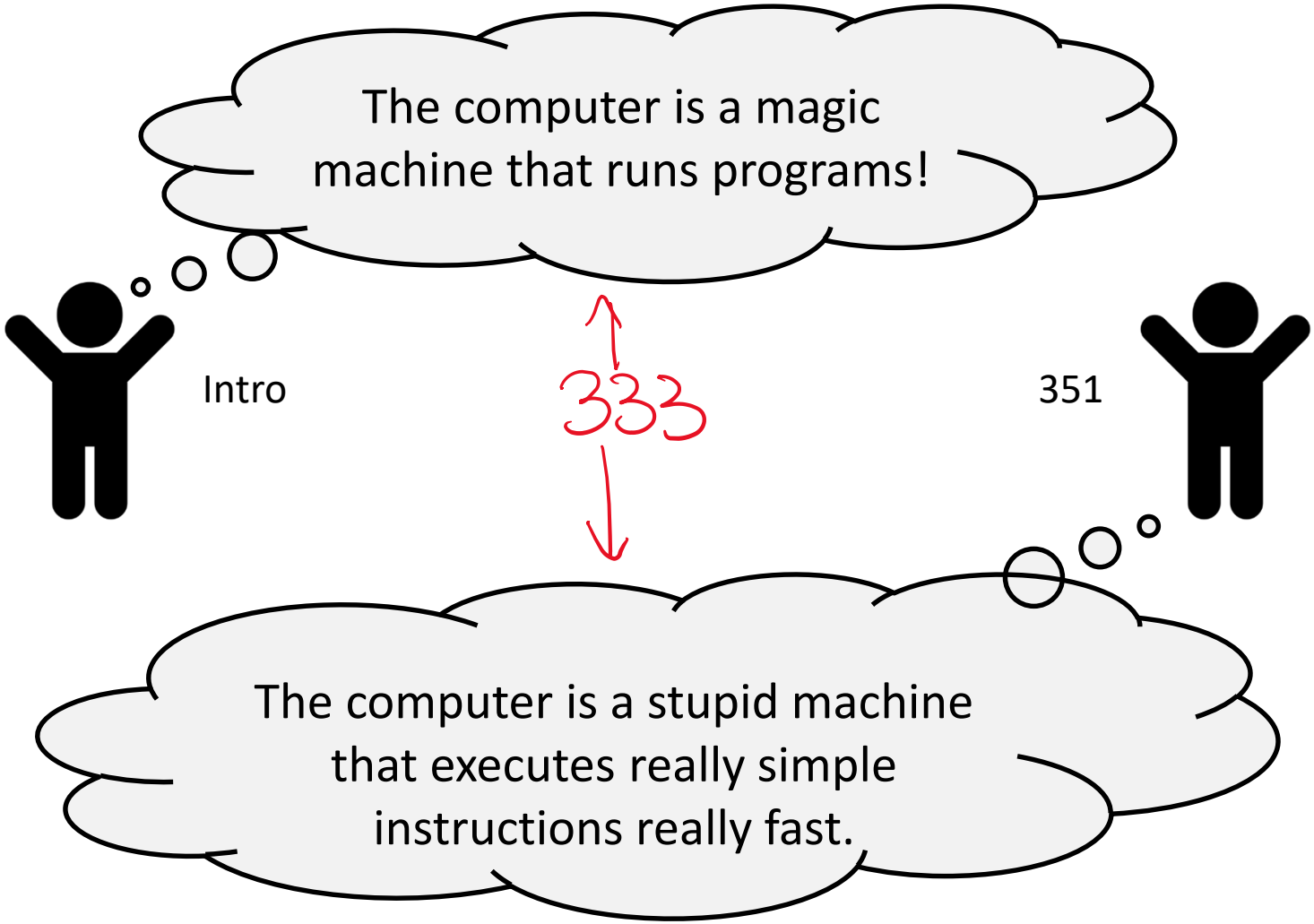
Administrivia

- Final in AND 223 on Wed, Dec 11 @ 2:30-4:20pm
  - Topics and review packet available on the course website
  - Review session in ECE 037 on Sun, Dec 8 @ 12-2pm

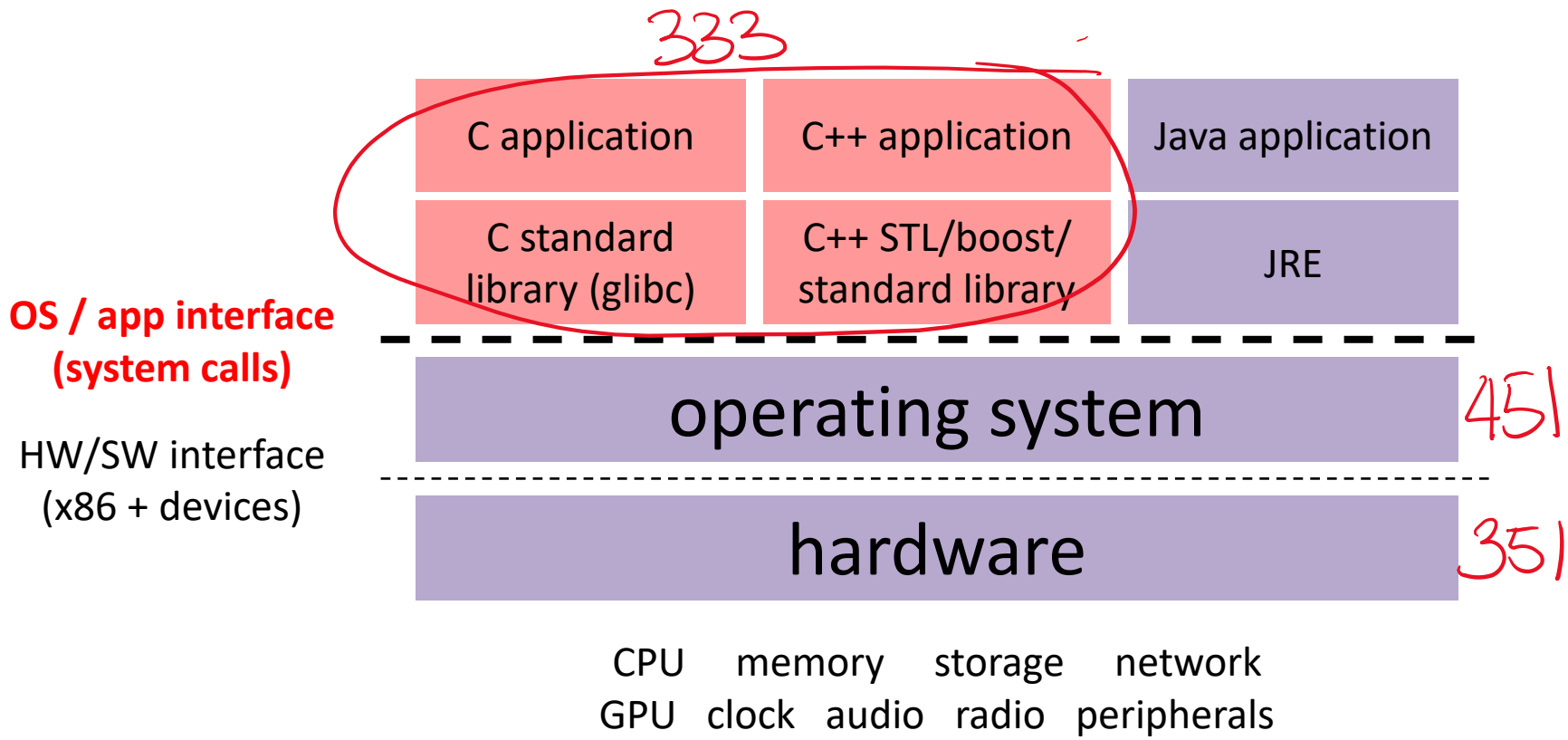
- Nominate your 😍AMAZING😍 TAs for the Bob Bandes Award

Course Goals

- Explore the gap between:



Course Map: 100,000 foot view



Systems Programming: What we just spent the quarter learning

- The programming skills, engineering discipline, and knowledge you need to build a system
  - **Programming:** C / C++
  - **Knowledge:** long list of interesting topics
    - Concurrency, OS interfaces and semantics, techniques for consistent data management, distributed systems algorithms, …
    - Most important: a deep(er) understanding of the "layer below"
  - **Discipline:** testing, debugging, performance analysis

Discipline?!?

the reason why we're studying ~50 year old ideas!

- Cultivate good habits, encourage clean code
  - Coding style conventions
  - Unit testing, code coverage testing, regression testing
  - Documentation (code comments, design docs)
  - Code reviews

- Will take you a lifetime to learn
  - But oh-so-important, especially for systems code
    - Avoid write-once, read-never code

Systems Programming: Why we just spent the quarter learning it

We had two major "thesis statements" this quarter:

1. Learning to handle the unique challenges of low level programming allows you to work directly with the countless "systems" that take advantage of it.

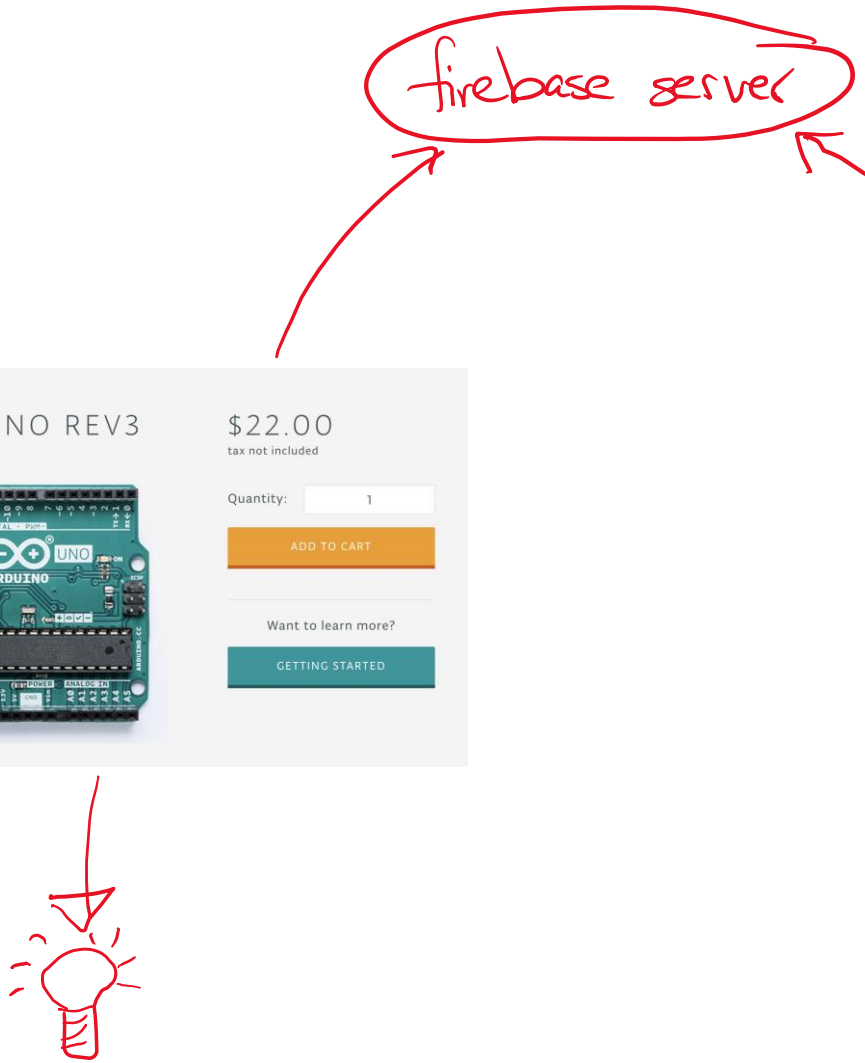
Case Study: An Internet-controlled lightbulb

Requirements:

(1) No custom network stacks. Off-the-shelf only!

(2) No open ports!

Case Study: An Internet-controlled lightbulb



Case Study: An Internet-controlled lightbulb

- Turns out … a physical timer also meets these requirements
- If programmer discipline is interesting to you, consider CSE 331!

Systems Programming: Why we just spent the quarter learning it

We had two major "thesis statements" this quarter:

1. Learning to handle the unique challenges of low level programming allows you to work directly with the countless "systems" that take advantage of it.

2. Understanding the "layer below" makes you a better programmer at the layer above.

2. Understanding the "layer below" makes you a better programmer at the layer above.

Common Heap Related Switches

There are many different command line switches that can be used with Java. This section describes some of the more commonly used switches that are also used in this OBE.

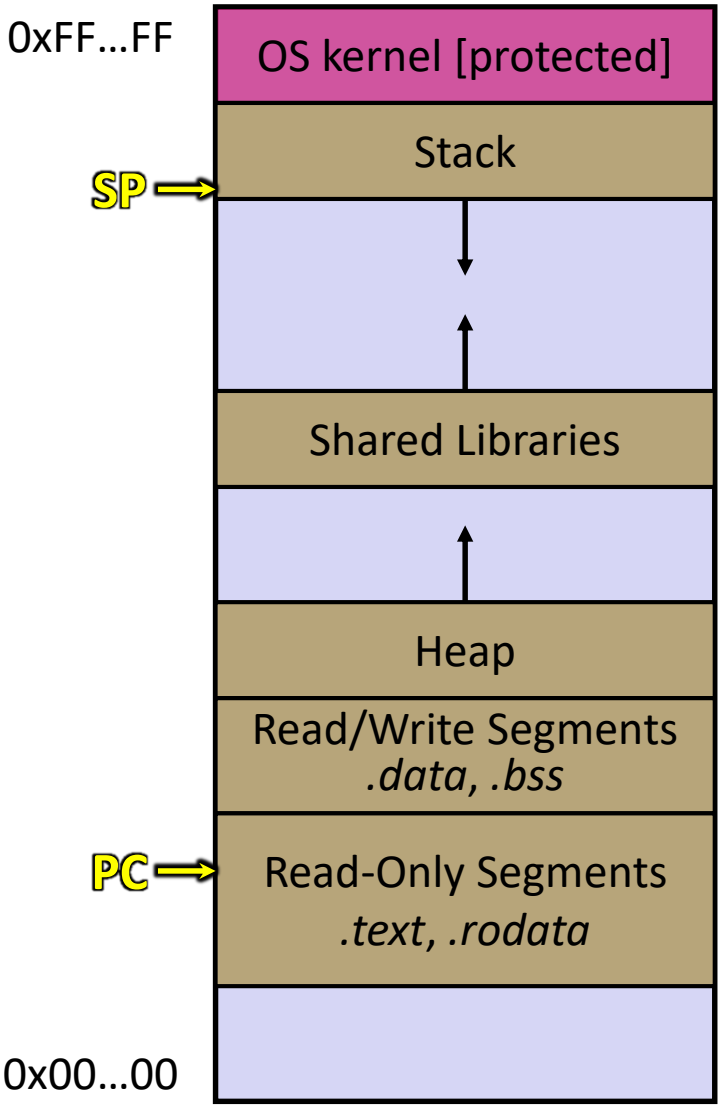
Switch	Description
`-Xms`	Sets the initial heap size for when the JVM starts.
`-Xmx`	Sets the maximum heap size.
`-Xmn`	Sets the size of the Young Generation.
-XX:PermSize	Sets the starting size of the Permanent Generation.
-XX:MaxPermSize	Sets the maximum size of the Permanent Generation

Main Topics

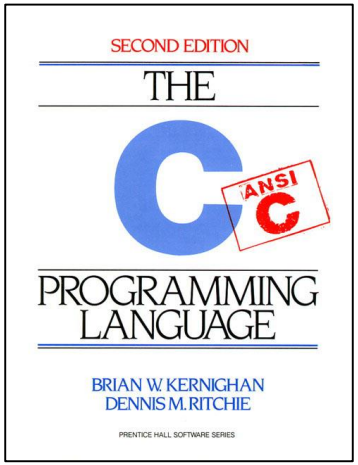
- Program Execution
- C
  - Low-level programming language
- C++
  - The 800-lb gorilla of programming languages
  - A "better C": classes + STL + smart pointers + …
- Memory management
- System interfaces and services
- Networking basics
  - TCP/IP, sockets, …
- Concurrency basics
  - POSIX threads, synchronization

Program Execution

- What's in a process?
  - Address space
  - Current state
    - Stack, SP, PC, register values, etc.
  - Thread(s) of execution
  - Environment
    - Arguments, open files, etc.



C



- Created in 1972 by Dennis Ritchie
  - Designed for creating system software
  - Portable across machine architectures
  - Most recently updated in 1999 (C99) and 2011 (C11)

- Characteristics
  - "Low-level" language that allows us to exploit underlying features of the architecture – but easy to fail spectacularly (!)
  - Procedural (not object-oriented)
  - "Weakly-typed" or "type-unsafe"
  - Small, basic library compared to Java, C++, most others….

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

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

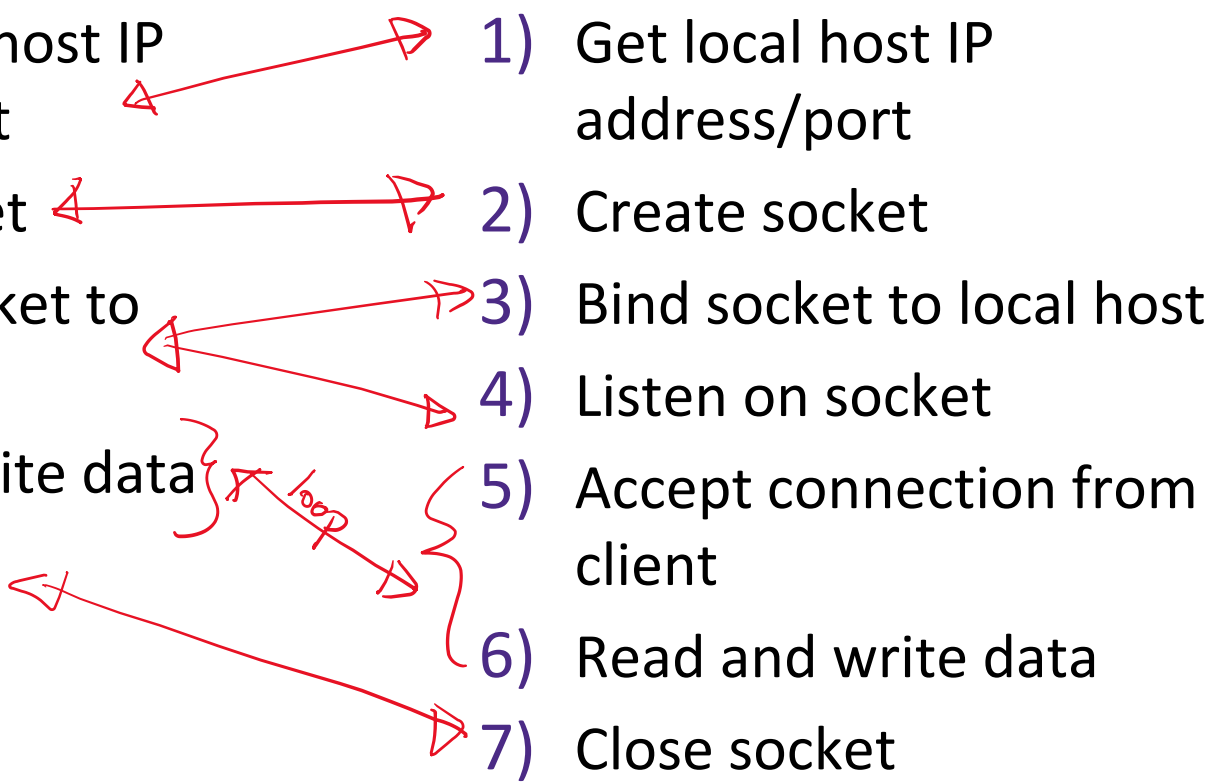
Memory Management

- 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! (and drawing!)

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

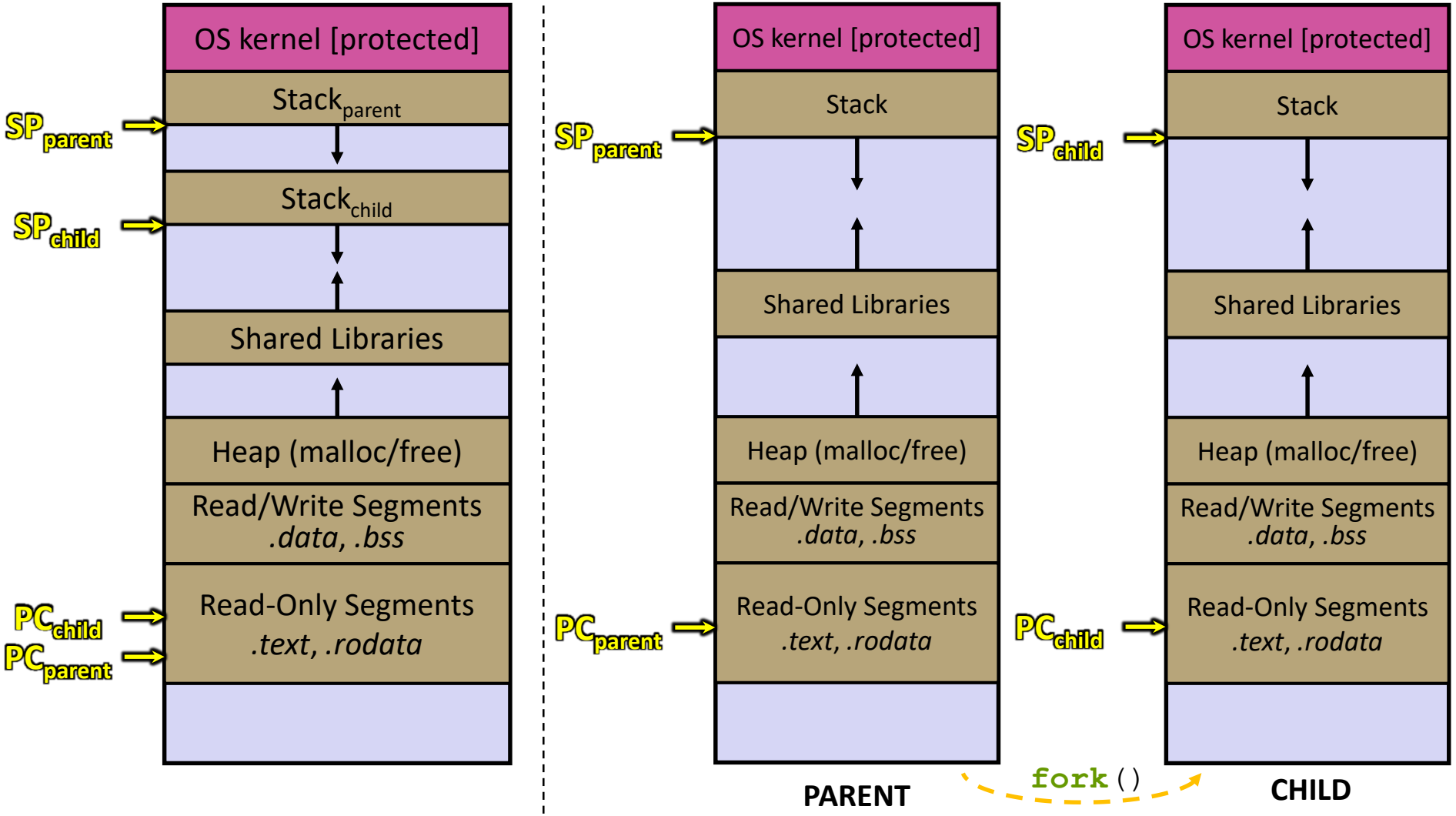
Network Programming



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, when/how much to lock…

Threads vs Processes … on One Slide!



Congratulations!

- Look how much we learned!

- Studying for the exam: (your mileage may vary)
  - Make cheat sheets *first* (that's how I wrote your final!)
    - 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

Thanks for a great quarter!

- Special thanks to the course content creators!!!


Steve Gribble

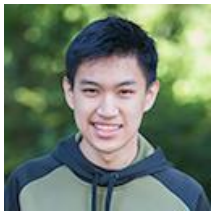

Hal Perkins


John Zahorjan


Justin Hsia

- Huge thanks to your awesome TAs!


Dao Yi


Farrell Fileas


Lukas Joswiak


Nathan Lipiarski


Renshu Gu


Travis McGaha


Yibo Cao


Yifan Bai


Yifan Xu

Ask Me Anything


That's all Folks!
Thank you for a great quarter
Good luck on the final!