Course Wrap-Up

CSE 333 Spring 2025

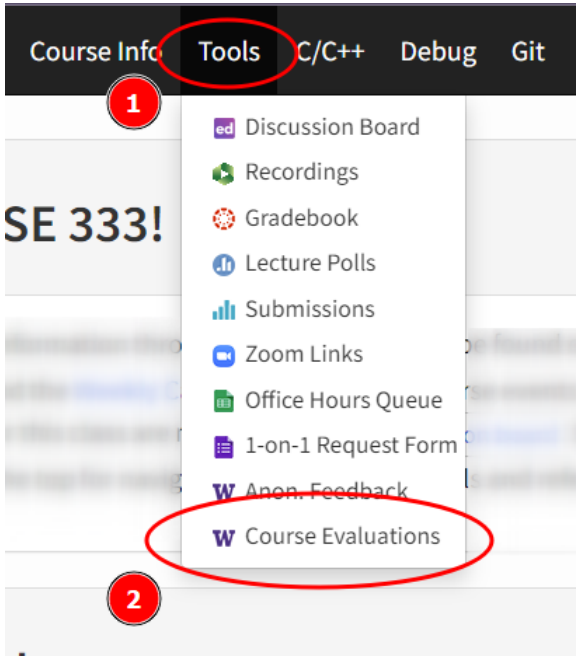
Instructors: Naomi Alterman, Chris Thachuk

Teaching Assistants:

Ann Baturytski	Derek de Leuw	Blake Diaz
Rishabh Jain	Chendur Jel Jayavelu	Lucas Kwan
Irene Xin Jie Lau	Nathan Li	Maya Odenheim
Advay Patil	Selim Saridede	Deeksha Vatwani
Angela Wu	Jiexiao Xu	

Final Administrivia

- Take 5 minutes to do your course evals please!
  - **[Constructive criticism]** helps us improve the course and our teaching (we're always trying)
  - **[Compliments]** about things you liked improve our ego (we're human beings!)
  - **[Righteous and impassioned rage]** is healthy to get out, but instead of doing that in course evals, try:
    - writing it out as an email to us
    - sitting on it overnight
    - returning the next day and deciding whether to hit 'send' or 'delete'



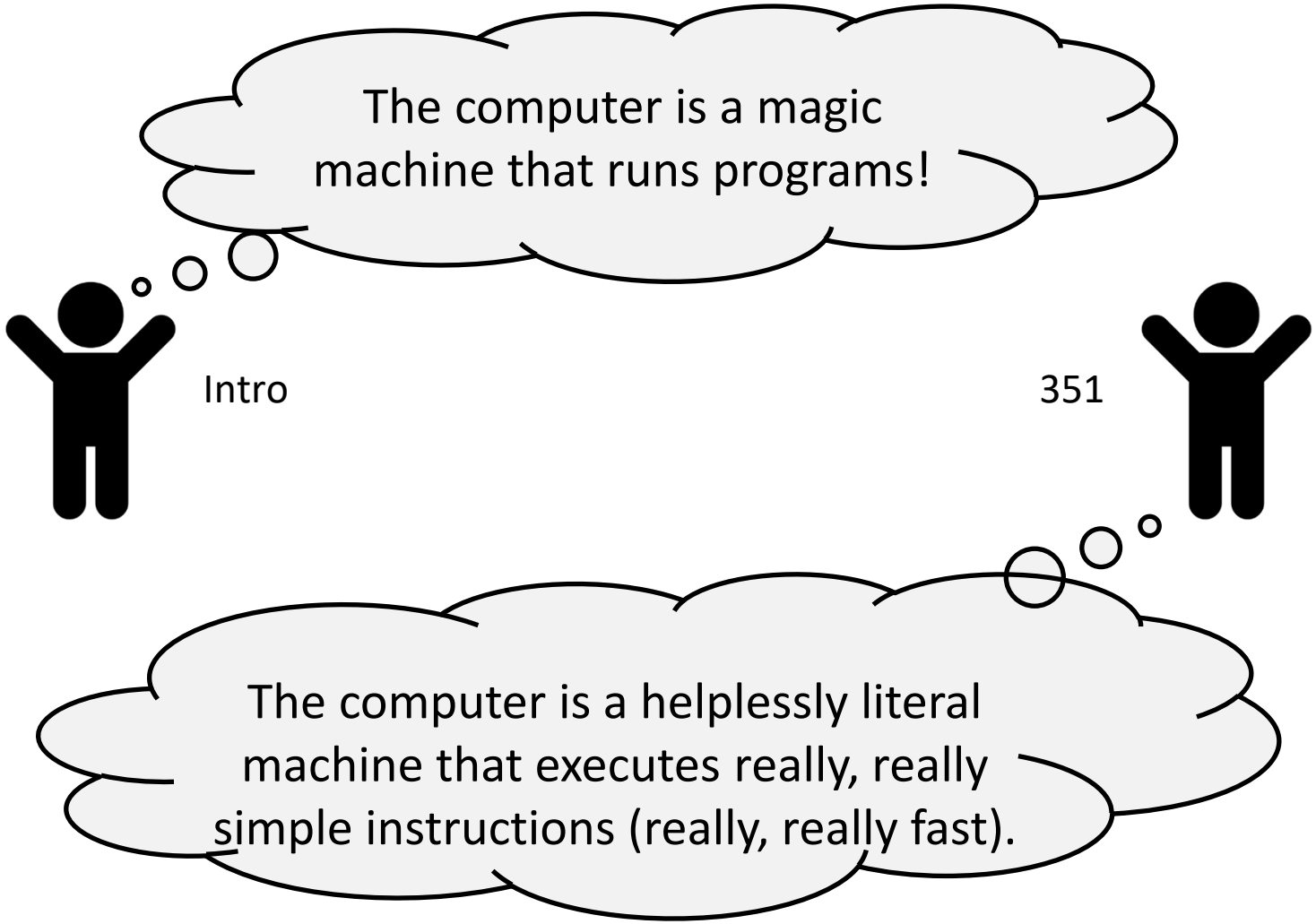
Final Administrivia

- Final exam Wed. Dec 10, 12:30-2:20, BAG 131
  - Topic list on the web now; somewhat weighted towards 2nd half of the quarter
  - Closed book but you may have **two 5x8 cards (or equivalent)** with handwritten notes (midterm card + new card or two new cards)
  - Finish strong, you're almost there!

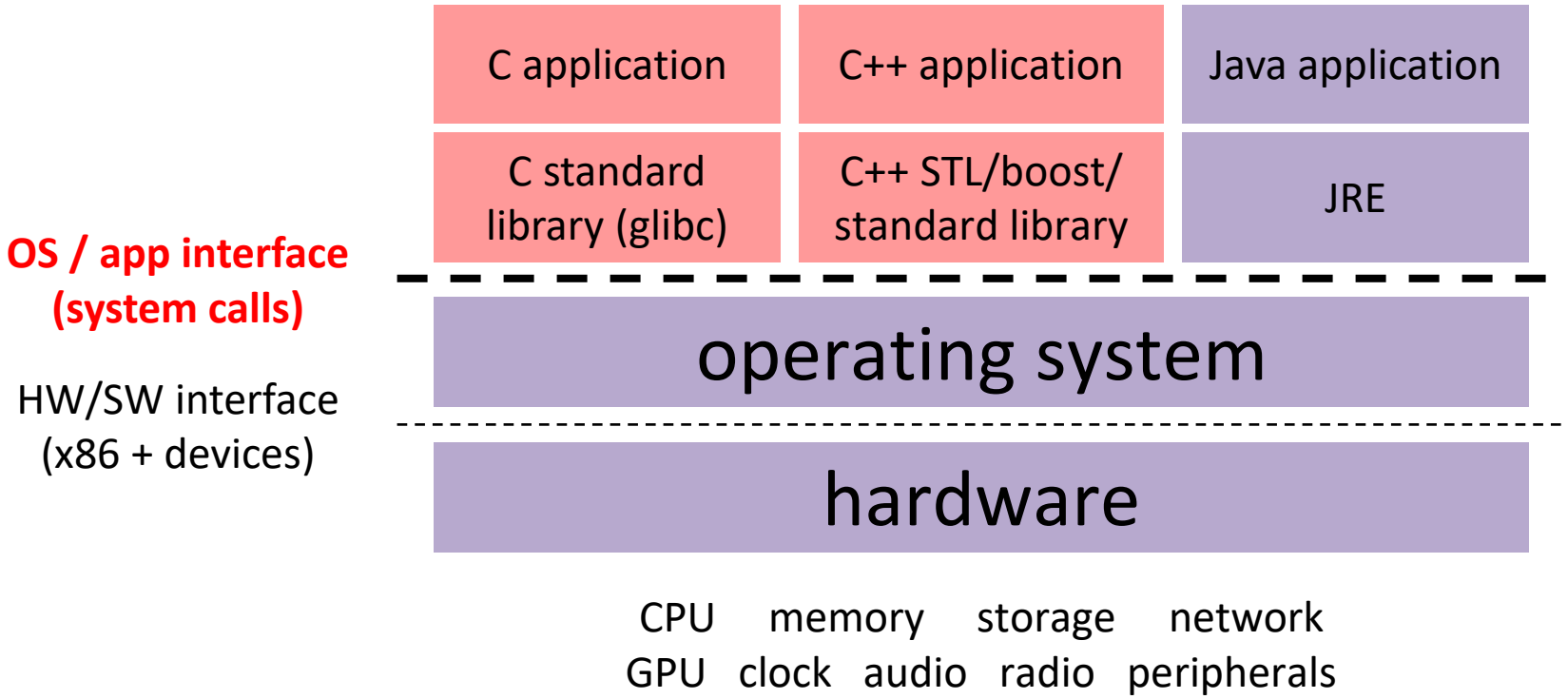
So what have we been doing for the last 11 weeks?

Course Goals

- Explore the gap between:



Course Map: 100,000 foot view



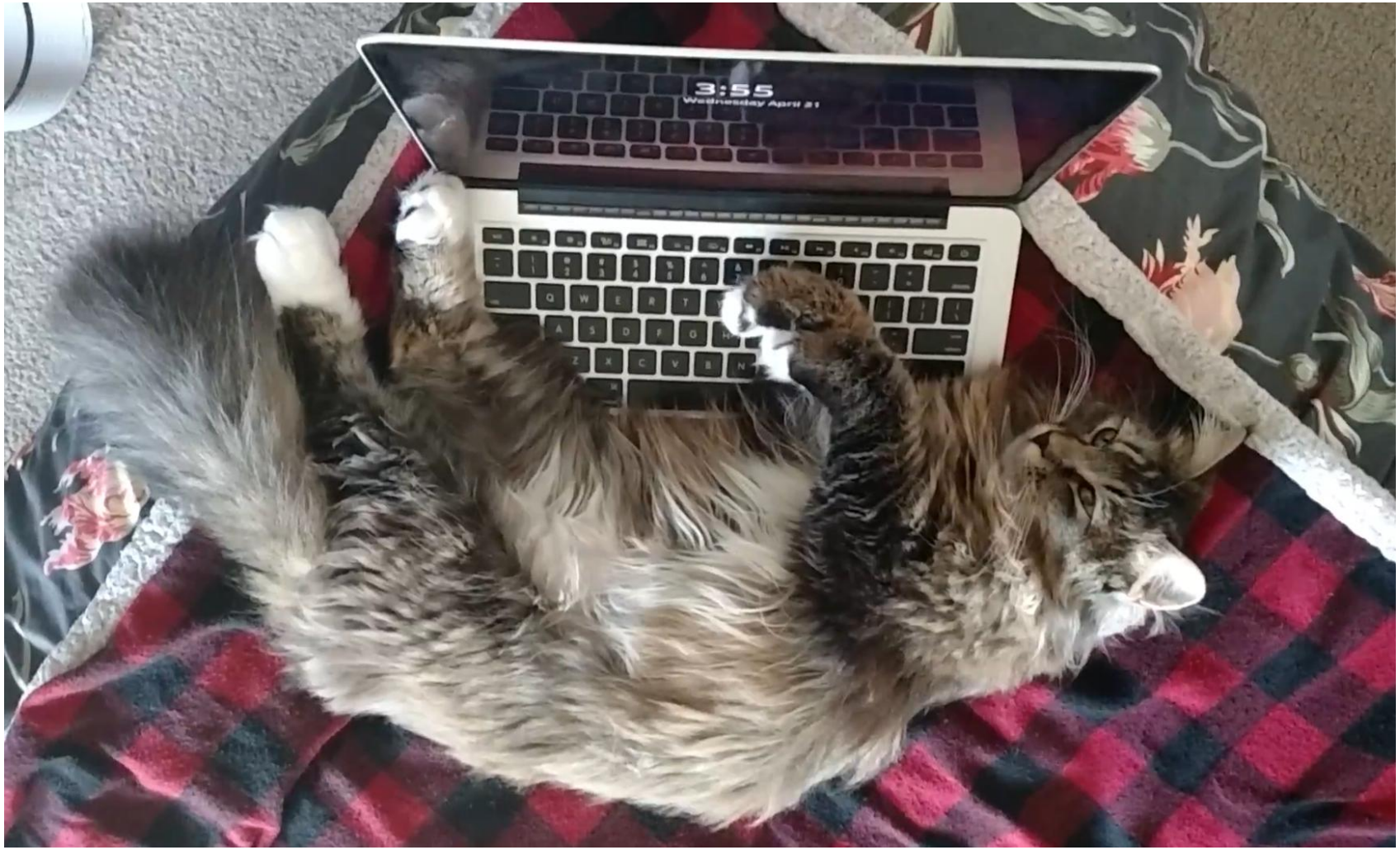
Systems Programming

- The programming skills, engineering discipline, and knowledge you need to build a system
  - **Programming:** C / C++
  - **Discipline:** design, testing, style, debugging, performance analysis
  - **Knowledge:** long list of topics
    - Concurrency, OS interfaces and semantics, techniques for consistent data management, distributed systems algorithms, …
    - Most important: **a willingness to engage with the "layer below," whatever that may be**

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

Danni built a chat program to slide into Professor Naomi's DMs

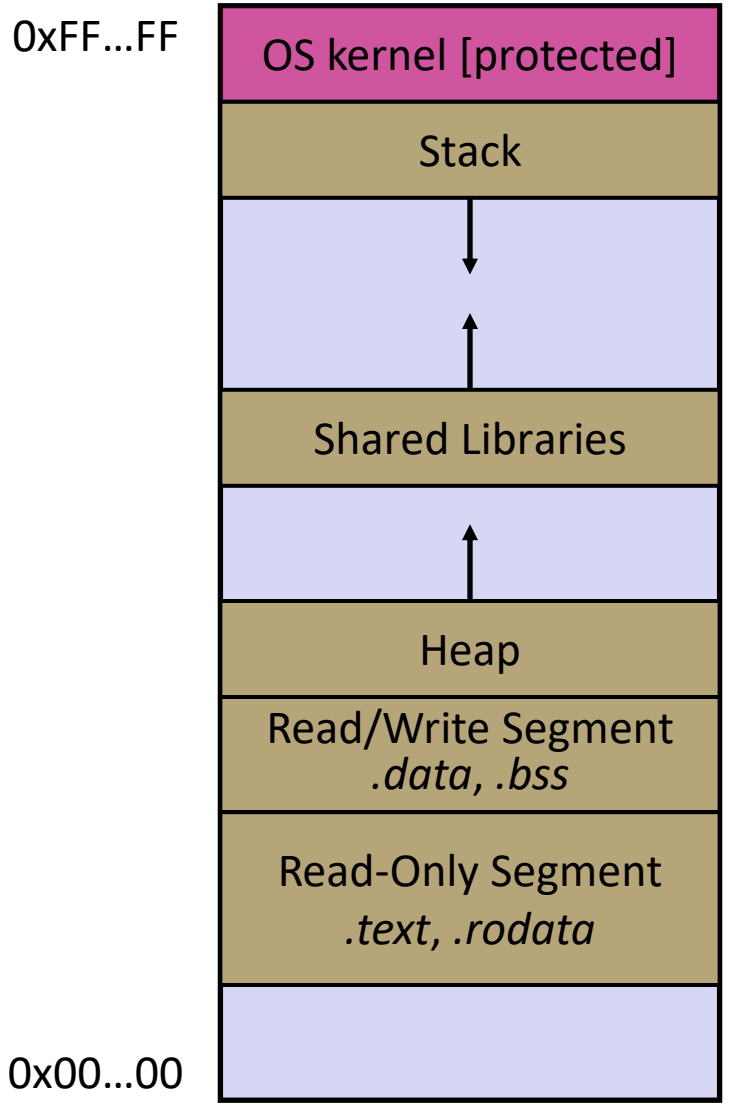


The C/C++ Ecosystem

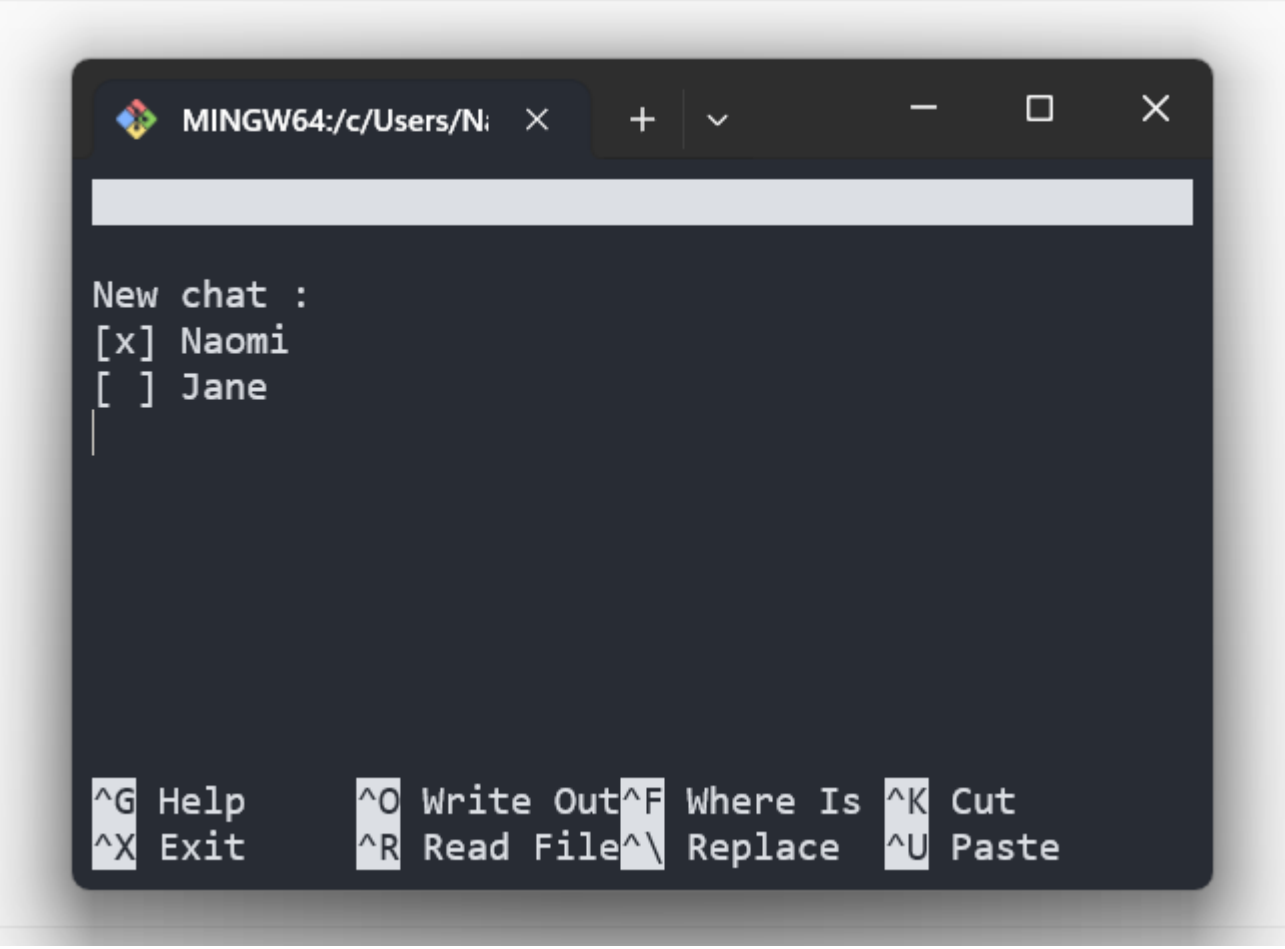
- System layers:
  - C/C++
  - Libraries
  - Operating system
- Building Programs:
  - Pre-processor (`cpp`, `#include`, `#ifndef`, …)
  - Compiler: source code → object file (`.o`)
  - Linker: object files + libraries → executable
- Build tools:
  - `make` and related tools
  - Dependency graphs

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.



NetCatChat 1.0



Structure of C Programs

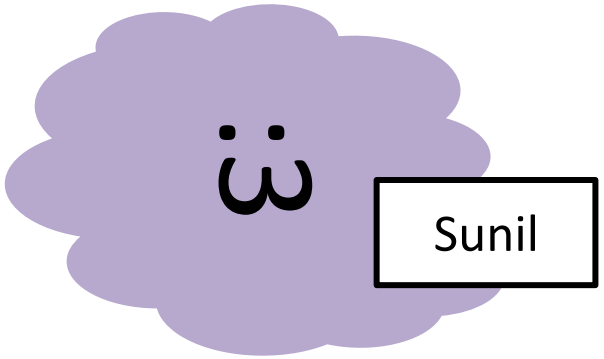
- Standard types and operators
  - Primitives, extended types, structs, arrays, typedef, etc.
- Functions
  - Defining, invoking, execution model
- Standard libraries and data structures
  - Strings, streams, etc.
  - C standard library and system calls, how they are related
- Modularization
  - Declaration vs. definition
  - Header files and implementations
  - Internal vs. external linkage
- Handling errors without exception handling
  - `errno` and return codes

C++ (and C++11 and later)

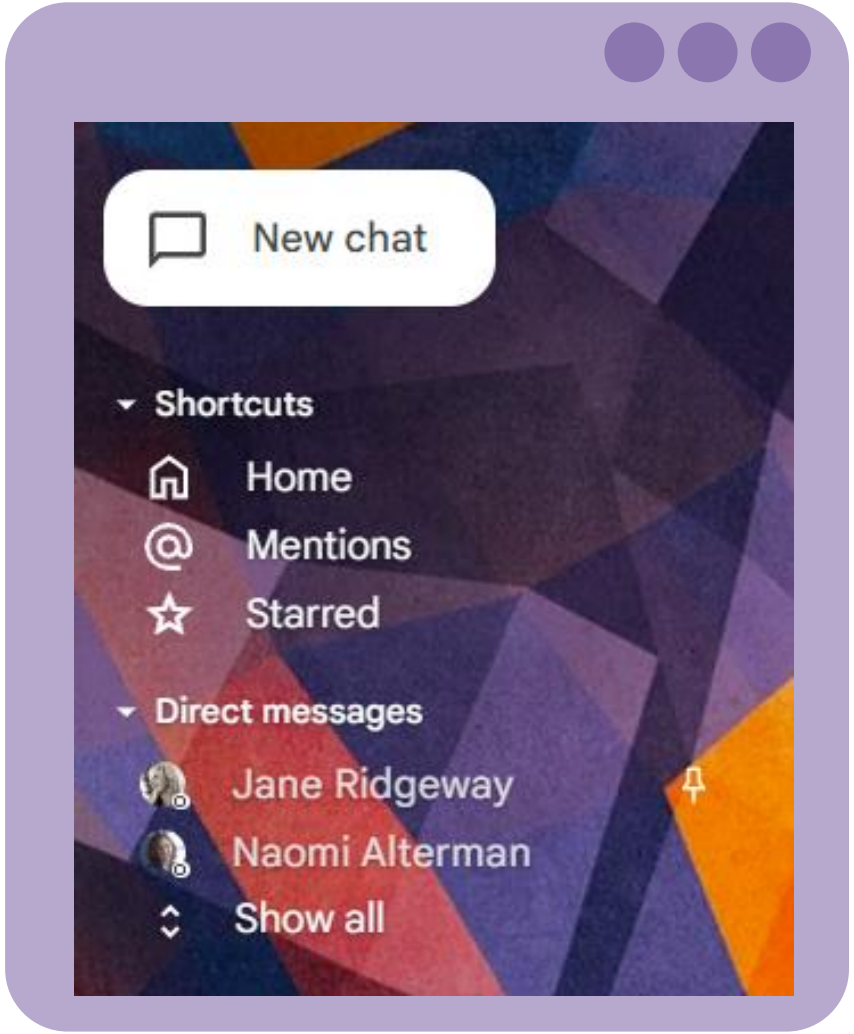
- A "better C"
  - More type safety, stream objects, memory management, etc.
- References and const
- Classes and objects!
  - So much (too much?) control: constructor, copy constructor, assignment, destructor, operator overloading
  - Inheritance and subclassing
    - Dynamic vs. static dispatch, virtual functions, vtables and vptrs
    - Pure virtual functions and abstract classes
    - Subobjects and slicing on assignment
- Copy semantics vs. move semantics

C++ (and C++11 and later)

- 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 implementation 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



NetCatChat 2.0



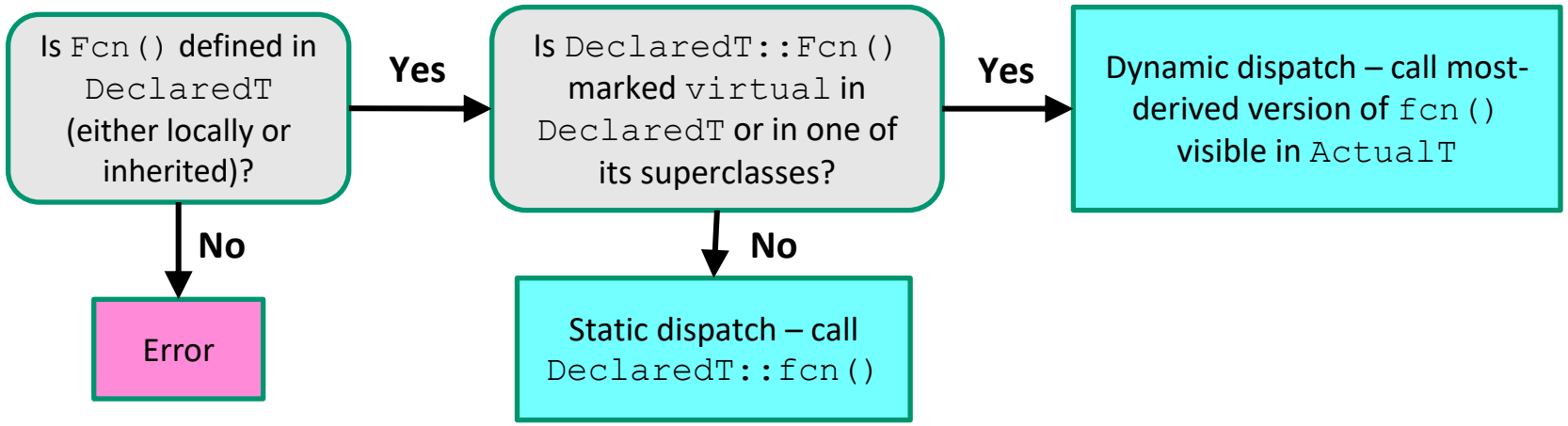
Dynamic Dispatch, Virtual Functions, &c

- *The* most frequent question on ed as the exam approaches, based on past experience.
- How to solve it? Understand the difference between static compile-time types (declared types) and actual type of the object referenced by a pointer.
- Understand which functions are virtual and which aren't
  - And remember that virtual is sticky, applies to all inherited / overridden functions in subclasses
- Then follow the "Mixed Dispatch" chart (next slide)

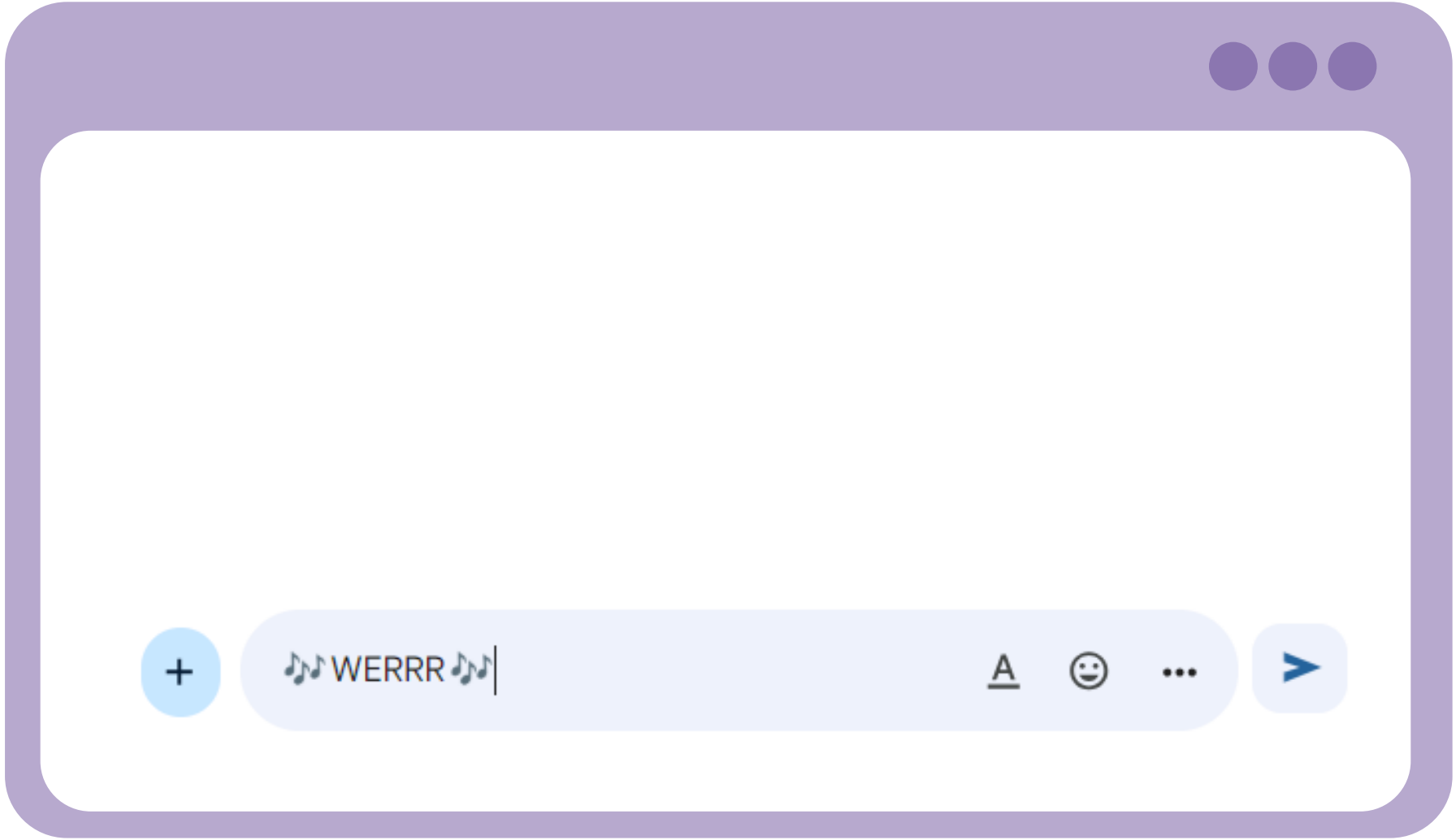
Mixed Dispatch

- Which function is called is a mix of both compile time and runtime decisions as well as *how* you call the function
  - If called on an object (*e.g.* `obj.Fcn()`), usually optimized into a hard-coded function call at compile time
  - If called via a pointer or reference:

```
DeclaredT *ptr = new ActualT;
ptr->Fcn();  // which version is called?
```



Ok, she's sending a message



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`), paper/whiteboards(!)
  - 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

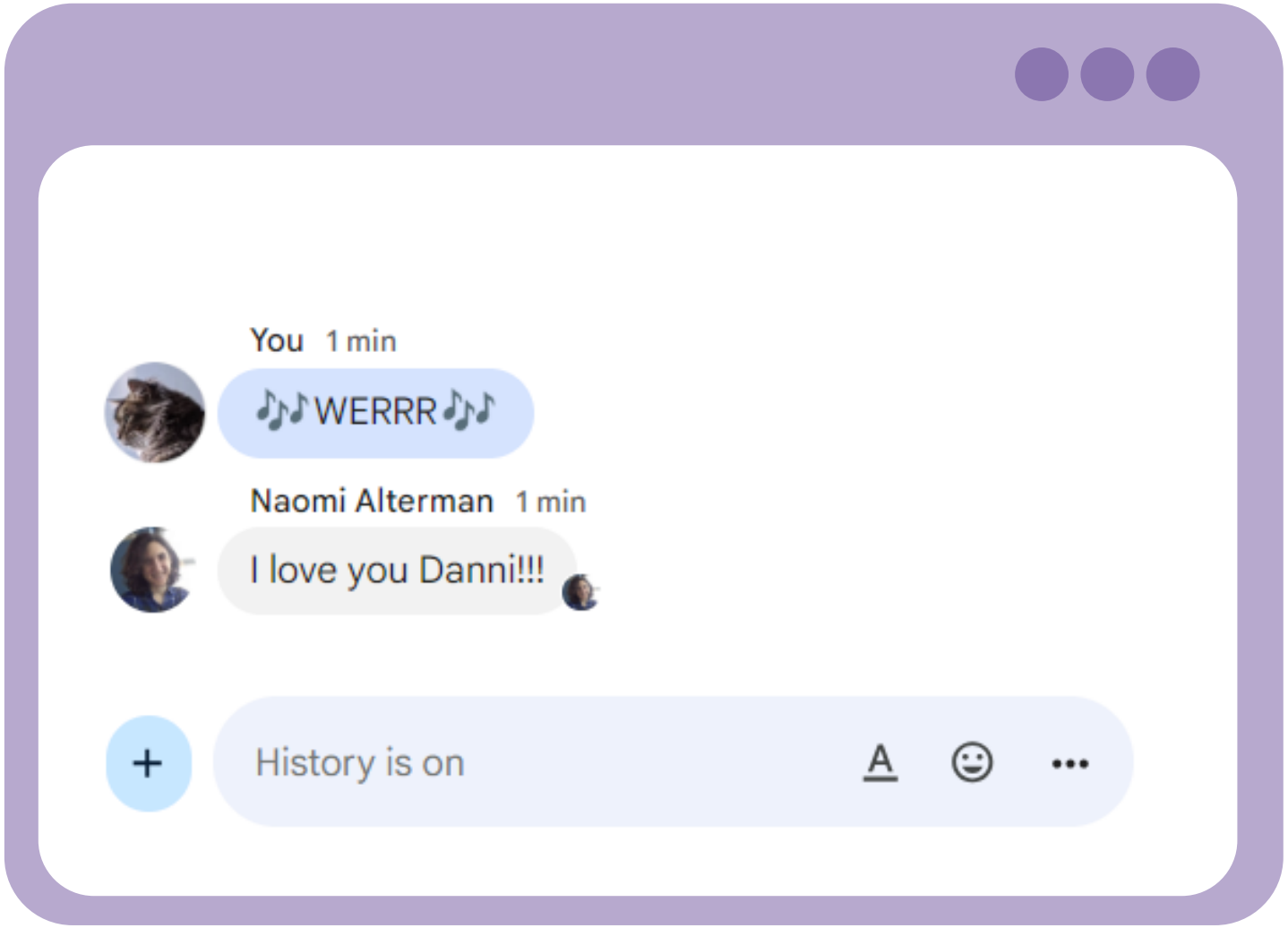
Client side

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

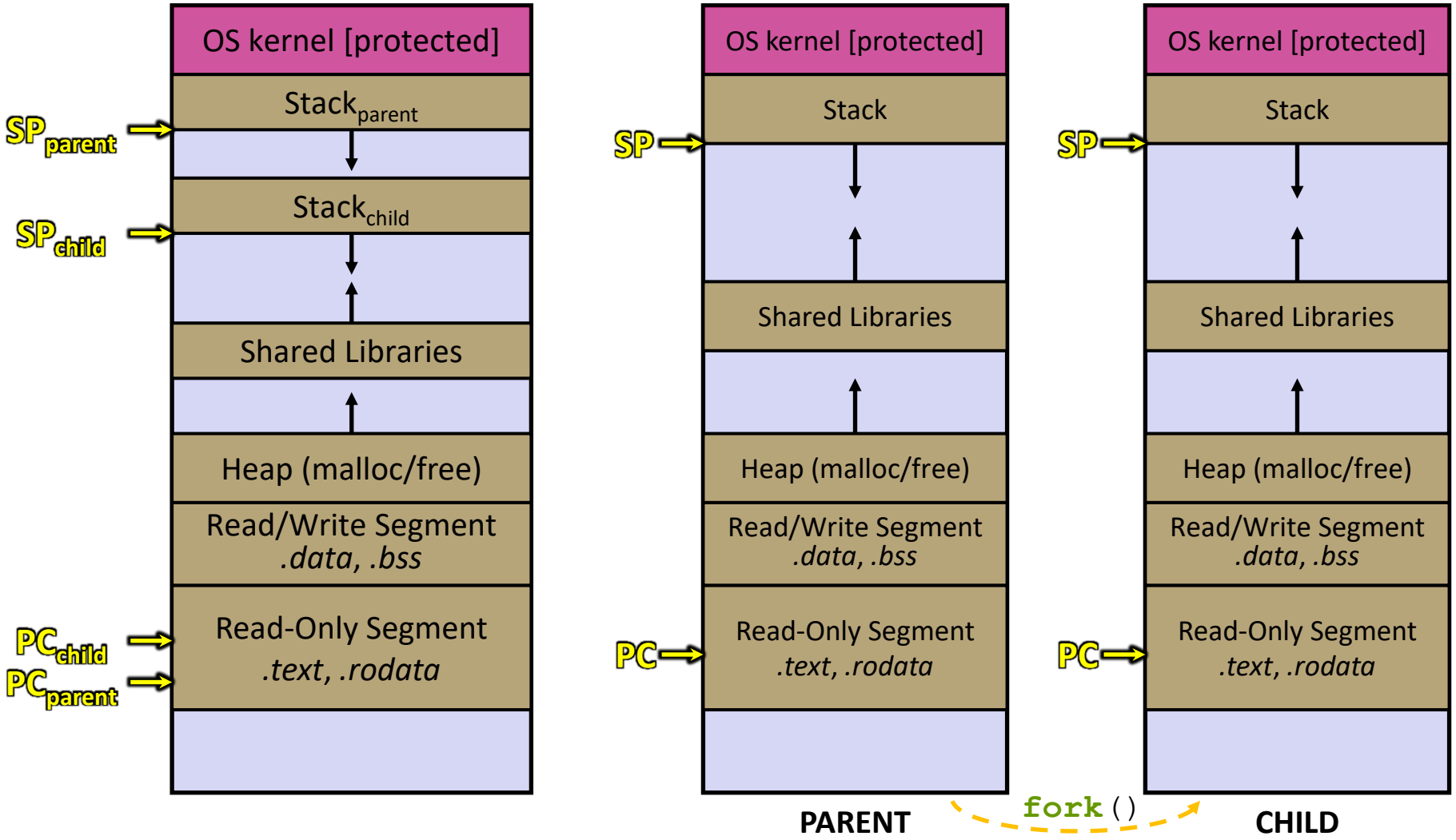
What's this!?


You 1 min
♪♪WERRR♪♪
Naomi Alterman 1 min
I love you Danni!!!
History is on

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



And a little bit of coding…

- Exercises
  - Small(ish) programs to try out new ideas and learn new things
  - Real Programmers® do this all the time
    - Super useful to try new ideas and use them in a small setting before relying on them in larger projects
    - Important habit to acquire – can save huge amounts of time in the end
- Projects
  - A chance to pull ideas together and get experience building bigger things
- Great programmers get that way because of the time, effort, and practice from writing lots of great code. With luck, CSE333 gave you a useful push in that direction.

Phew! That's it!

- But that's a lot!!

- Take a look back and congratulate yourself on what you've accomplished in a 10-week quarter!

One last thing…

- 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 ("ideate"?) and trade ideas with colleagues
  - "Simulate" an old exam
    - Do it in one timed sitting
    - Working problems is far more helpful than reading old answers!
  - "Grade" yourself, then go back and review problems
    - If still unsure why, ask staff or your fellow students (study groups!)
    - 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
- And many more….

This doesn't happen without lots of help…

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

Ann Baturytski	Derek de Leuw	Blake Diaz
Rishabh Jain	Chendur Jel Jayavelu	Lucas Kwan
Irene Xin Jie Lau	Nathan Li	Maya Odenheim
Advay Patil	Selim Saridede	Deeksha Vatwani
Angela Wu	Jiexiao Xu	

- And thanks to the folks who put the course together:
  - Steve Gribble, John Zahorjan, Hal Perkins, Justin Hsia, Hannah Tang, Aaron Johnston, Travis McGaha, many others

And thanks to…

You!

It's been great to share new ideas and skills with everyone. You should be proud of what you've done. Please take care of yourself, watch your health, stay active, and help yourself, your friends, your community.

Congratulations and best wishes!

You've learned a *lot* – go out and be good stewards of society's collective technical debt!

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


That's all Folks!