Course Overview

CSCI 237: Computer Organization
1st Lecture, Feb 7, 2025

Jeannie Albrecht

Overview

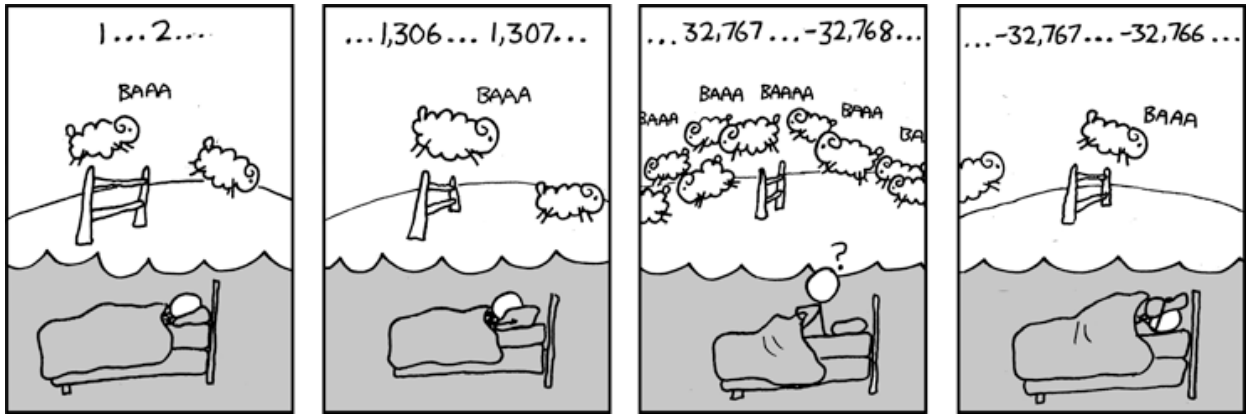
- Course theme
- Five realities
- How the course fits into the CS curriculum
- Course logistics and academic integrity

Course Theme: Abstraction Is Good, But Don't Forget Reality

- **Most CS courses emphasize abstraction**
  - Abstract data types
  - Asymptotic (Big O) analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying *implementations*
- **Useful outcomes from taking CS237**
  - Become more effective programmers and scientists
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS
    - Operating Systems (432), Distributed Systems (339), Security (331), Storage Systems (333), Parallel Computing (338), Robotics (345)

Great Reality #1:
Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 → 1,600,000,000
    - 50000 * 50000 → ??



- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

```
code — jeannie@sysr
rugger9:code jeannie$ cat test.c
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("%d\n",40000*40000);
  printf("%d\n",50000*50000);
  printf("%f\n",(1e20 + -1e20) + 3.14);
  printf("%f\n",1e20 + (-1e20 + 3.14));
}
rugger9:code jeannie$ gcc -o test test.c
rugger9:code jeannie$ ./test
1600000000
-1794967296
3.140000
0.000000
```

Source: xkcd.com/571

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all "usual" mathematical properties**
  - Due to *finiteness of representations*
  - Integer operations satisfy "ring" properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy "ordering" properties
    - Monotonicity, values of signs
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2:
You've Got to Know (a little) Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory is an *Unphysical* Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

C code

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds! */
  return s.d;
}
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000006103516
fun(4)  →  3.14
fun(6)  →  3.14, Segmentation fault
```

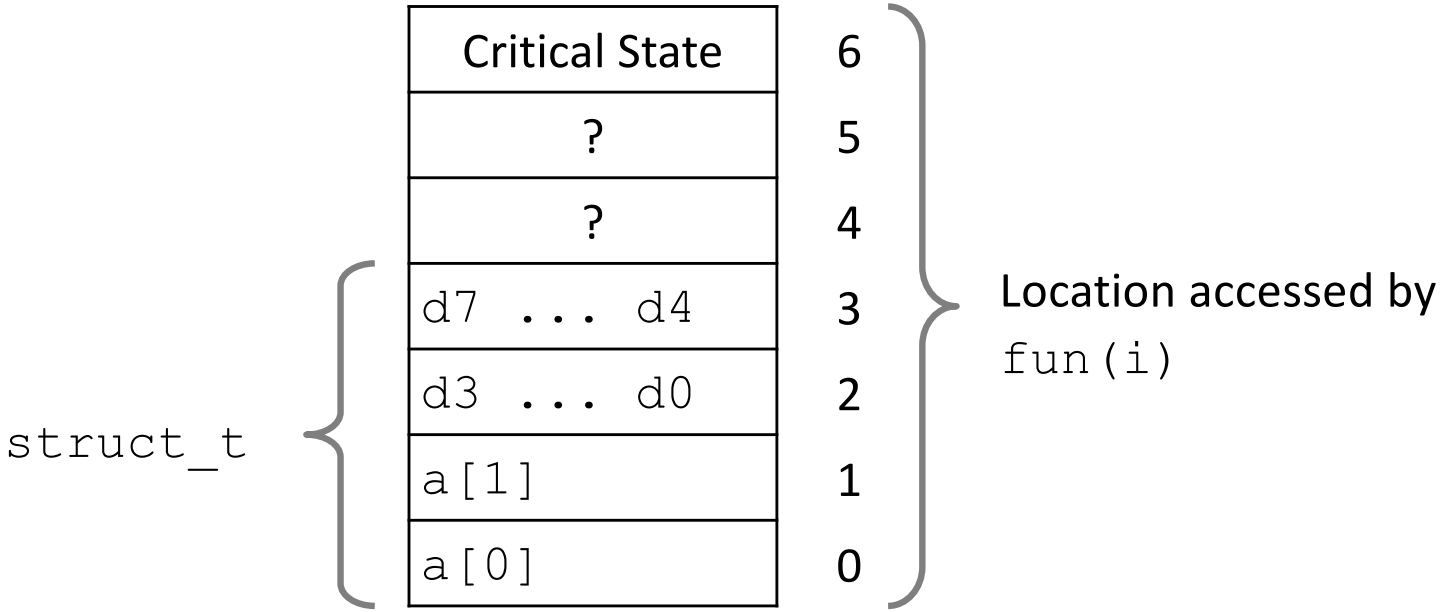
- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000006103515 6
fun(4)  →  3.14
fun(6)  →  3.14, Segmentation fault
```

Explanation:



Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs!**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Python, Go, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs are compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048]) {

  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

48ms

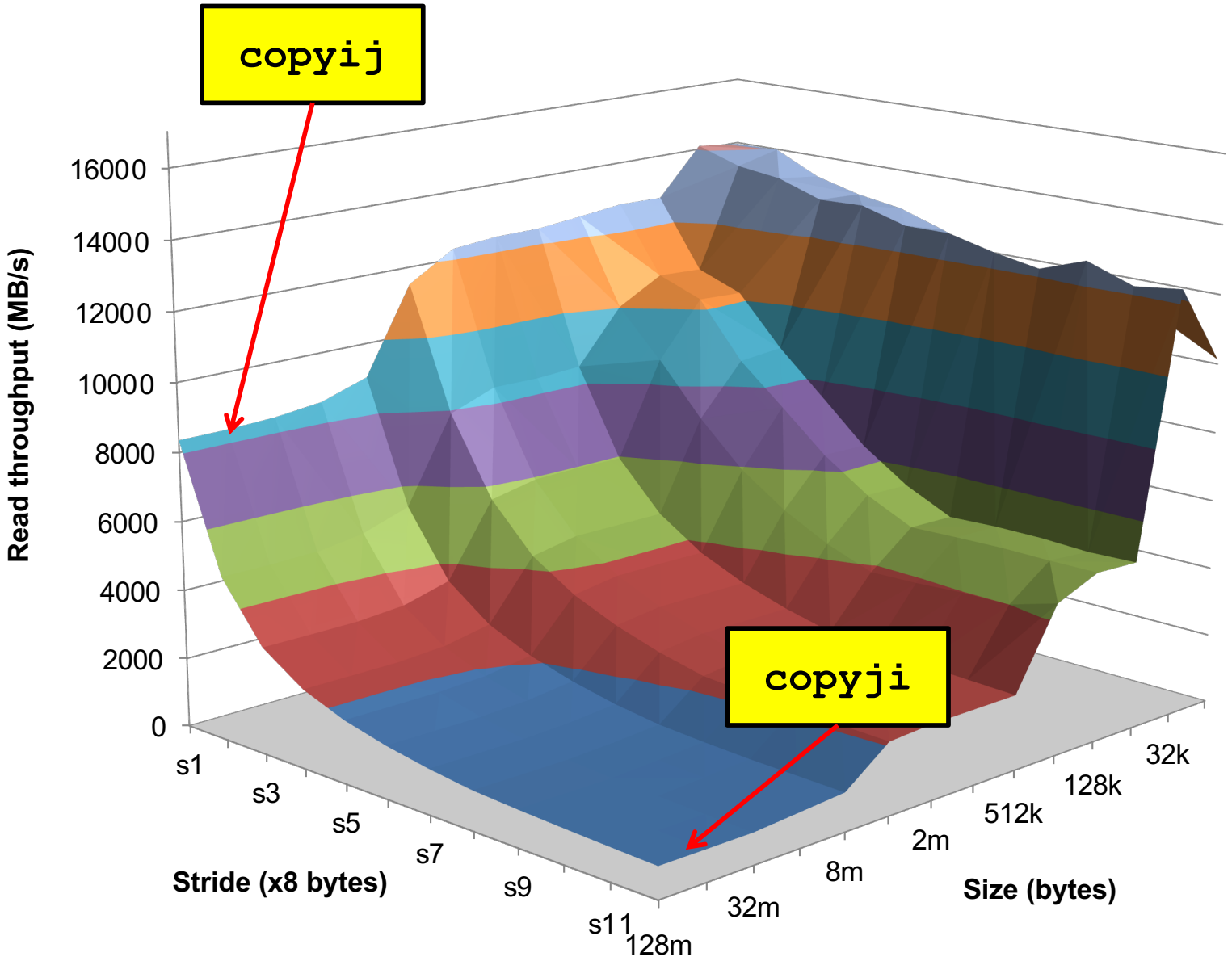
```
void copyji(int src[2048][2048],
            int dst[2048][2048]) {

  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

167ms

2.9 GHz Intel Core i5

- Hierarchical memory organization
- Performance depends on access patterns
  - Including how we step through multi-dimensional array

Why The Performance Differs



Great Reality #5: Computers do more than execute programs

- **They need to get data in and out**
  - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

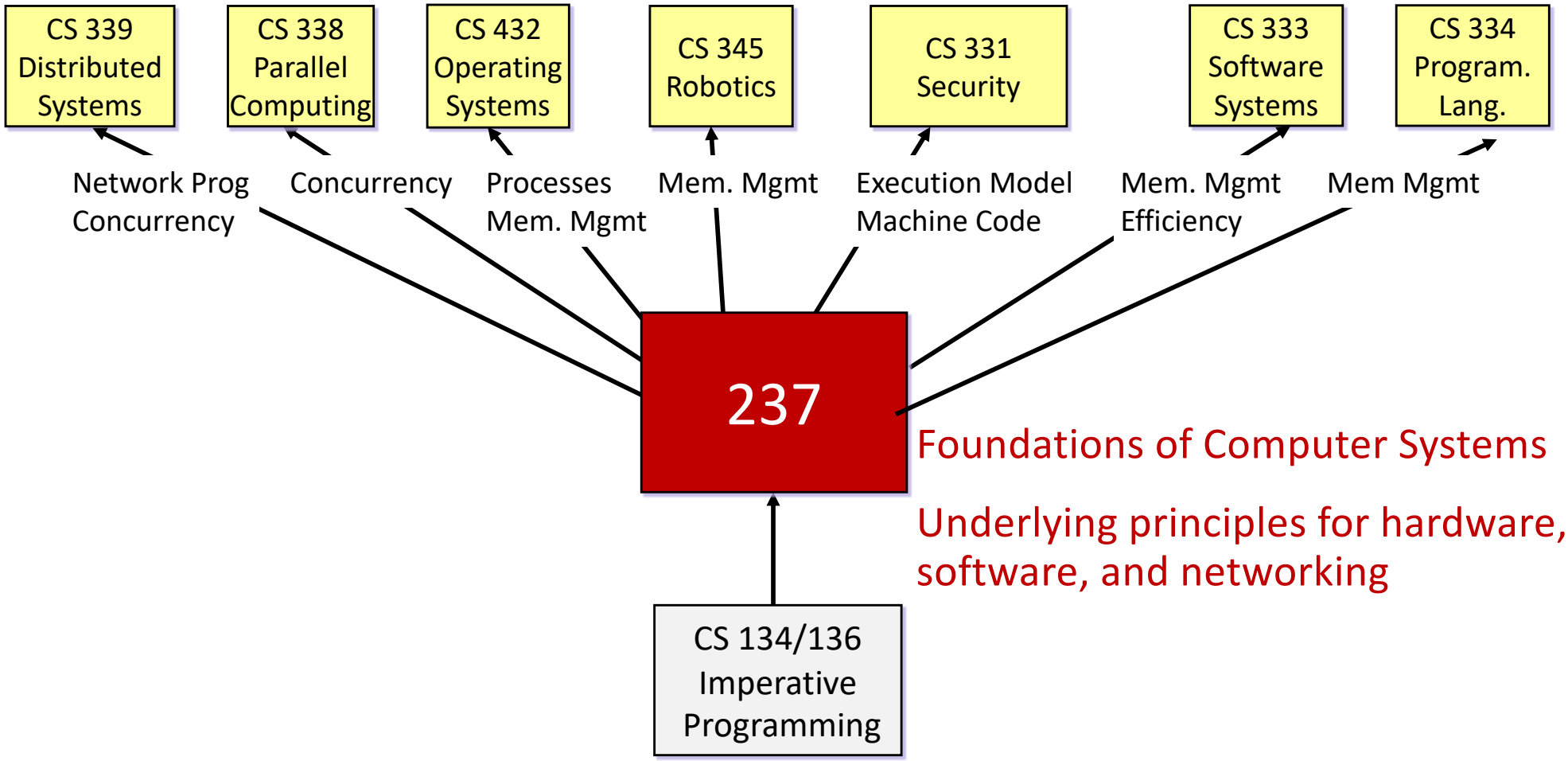
Course Perspective

- Most Systems Courses are Builder-Centric
  - Distributed Systems
    - Design programs that run on many machines at once
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

Course Perspective (Cont.)

- Our Course is Programmer-Centric
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - **We bring out the hidden hacker in everyone!**

Role within CS Curriculum



Administrative Details

- Attendance
- Course syllabus (see PDF on webpage)
- Lab: Wed and Thur 1-2:30 in Ward Lab (TBL 301)
  - Any volunteers to swap FROM Wed TO Thur lab?
- Instructor: jeannie@cs.willams.edu, TCL 305
- TAs (evening hours Sun-Thur, specific hours will be posted soon)
  - Michael Faulkner
  - Natalia Nolan
  - Nathan Vosburg
  - Charlie Tharas
  - Yanni Kakouris
  - Niklas Obermüller

Textbooks

- Randal E. Bryant and David R. O'Hallaron,
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- (optional) Brian Kernighan and Dennis Ritchie,
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators

Course Components

- Lectures
  - Mostly higher level concepts
  - Some info about important tools and skills for labs
- Weekly homework
  - On Glow, low stakes, test conceptual understanding, due Friday at noon
- Labs
  - The heart of the course
  - Challenging, but very rewarding and (hopefully) fun
  - 1-2 weeks each, mostly due on Tuesdays/Wednesdays
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- Exams (midterm + final)
  - Test your understanding of concepts & mathematical principles
  - Midterm: in lab on March 19/20
  - Final: scheduled

Getting Help

- Class Web page: **http://www.cs.williams.edu/~jeannie/cs237**
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, labs, sample exams, etc
  - Clarifications to labs

- Glow
  - Use Glow for weekly homework
  - Occasional announcements

Getting Help

- Office hours
  - TBD – I will finalize by next week!
- 1:1 appointments with me
  - Open door policy (although I have lots of meetings…)
  - Email is best way to contact me
- TA hours
  - TBD
  - Probably every evening from ~7-10 (except Fri and Sat)

Policies: Labs And Exams

- Groupwork
  - You must work alone on all assignments unless otherwise noted
  - When in doubt, just ask!
- Submissions
  - Labs due at 11pm on date specified
  - Electronic handins using autograder **submit script** (details in lab next week)
- Exams
  - Closed book
  - Details to come later
- Discussing grades
  - We all make mistakes! Come see me asap to discuss.

Cheating: Description

- **Please pay close attention! I take this VERY seriously.**
- What is cheating?
  - Sharing code: by copying, retyping, **looking at**, or supplying a file
  - Describing: verbal description of code from one person to another
  - Coaching: helping your friend to write a lab, line by line
  - **Searching the web or using ChatGPT for full or partial solutions**
  - Copying code from a previous course or online solution (autograder checks for this)
    - You are only allowed to use code we supply, or from the CS:APP website
- What is NOT cheating?
  - Explaining how to use systems or tools
  - Helping others with high-level design issues (be sure to give credit)
  - Searching web/ChatGPT for specific error messages info
  - Using web/ChatGPT for finding data structure documentation
- See the course syllabus for more details.
  - Ignorance is not an excuse. When in doubt, ask!

Cheating: Consequences

- Penalty for cheating:
  - According to our Honor Code, if I suspect cheating, I must notify committee
  - If found guilty, probably will fail course
  - Permanent mark on your record
- Detection of cheating:
  - We have sophisticated tools for detecting code plagiarism
  - Some tools are automatically run via autograder
- Just don't do it!
  - Start early and don't fall behind
  - Ask for help when you get stuck

Facilities

- Labs will be held in Ward lab (TBL 301)
- Can also use the Linux machines in TCL 312

springer	primrose	pepper	shiloh
ruth	finch	dolores	antoinette
rita	buttercup	cashew	dinah
dolly	pippa	kal	molly
pika	radish	shadow	pantalaimon

- `brownswiss (off campus access)`
- `limia (off campus access)`
- `lohani (off campus access)`
- `angus (off campus access)`

Timeliness

- Lateness penalties
  - Late lab submissions get penalized **20% per day**
  - No submissions later than **4 days after due date**
- Catastrophic events
  - Major illness, death in family, …
  - Come talk to me (and probably get help from a dean)
- Advice
  - Once you start running late/falling behind, it's really hard to catch up

Other Rules of the Classroom

- Laptops permitted for note taking
- Electronic communications
  - No email, instant messaging, texting, cell phone calls, etc, during class
  - Distracting to everyone around you
- Presence in lectures and labs is **required**
  - Let me know if you need to miss for any reason
- I don’t mind if you eat in class
  - Just don’t distract others
  - Avoid bringing food to lab if possible

Policies: Grading

- Exams (60%): midterm (30%), final (30%)
- Labs (30%): weighted (slightly) according to effort
- Homework (10%): weekly low stakes assignment on Glow

Programs and Data

- Topics
  - Bits operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

- Assignments (probably)
  - L1 (datalab): Manipulating bits
  - L2 (bomblab): Defusing a binary bomb
  - L3 (archlab): Closer look at assembly code

The Memory Hierarchy

- Topics
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- Assignments
  - L4 (cachelab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.

Virtual Memory

- Topics
  - Virtual memory, address translation, dynamic storage allocation
  - Includes aspects of architecture and OS

- Assignments
  - L5 (malloclab): Writing your own malloc package
    - Get a real feel for systems-level programming

Networking, and Concurrency

- Topics
  - High level and low-level I/O, network programming
  - Internet services, Web servers
  - concurrency, concurrent server design, threads
  - I/O multiplexing with select
  - Includes aspects of networking, OS, and architecture

- Assignments
  - L6 (echoserver): Brief intro to threads, processes, and sockets

Lab Rationale

- Each lab has a well-defined goal such as solving a puzzle or winning a contest
- Doing the lab should result in new skills and concepts
- We try to use competition in a fun and healthy way
  - Set a reasonable threshold for full credit
  - Optionally post intermediate results (anonymized) on scoreboard for glory!

Welcome and enjoy!