Course Overview

CS-2011: Introduction to Computer Systems (Fall 2022)
Lecture 1

Outline

- **Big Picture**
  - Course theme
  - Five realities
  - How the course helps with other CS/ECE courses
- **Academic integrity**
- **Logistics and Policies**

The Big Picture

Course Theme:
(Systems) Knowledge is Power!

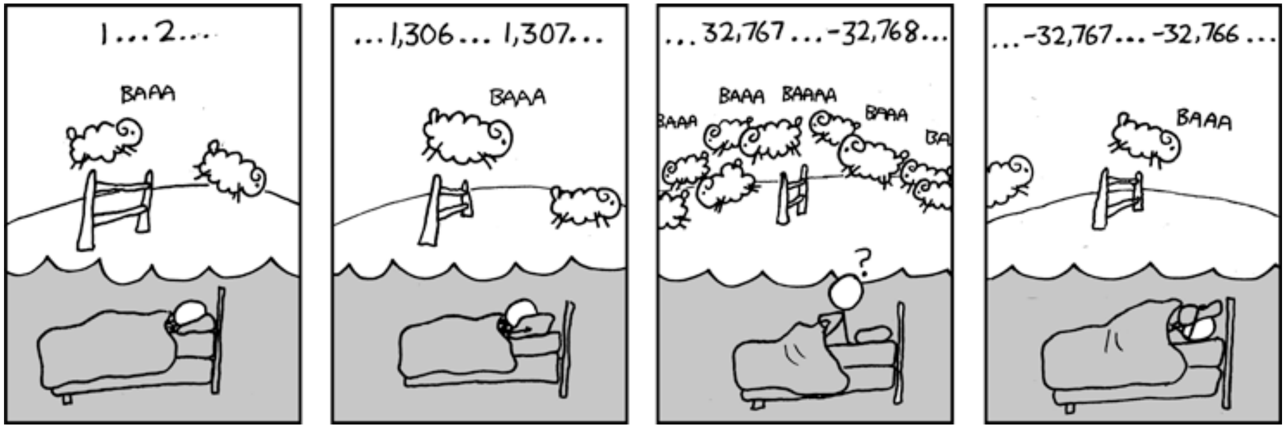
- **Systems Knowledge**
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you as a programmer can best use these resources
- **Useful outcomes from taking CS-2011**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS, ECE, ...
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Computer Security, Malware Analysis, etc.

It’s Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don’t forget reality
- **Most CS courses emphasize abstraction**
  - **(CE courses less so)**
  - **Abstract** data types
  - Asymptotic analysis
    - e.g., Big-O notation (best case, average case, and worst case scenario of an algorithm)
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying **implementations**
  - Sometimes the abstract interfaces don’t provide the level of control or performance you need

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 ➙ 1600000000
    - 50000 * 50000 ➙ ??

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

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 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

```
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)  ➙   2.00000061035156
fun(4)  ➙   Segmentation fault
```

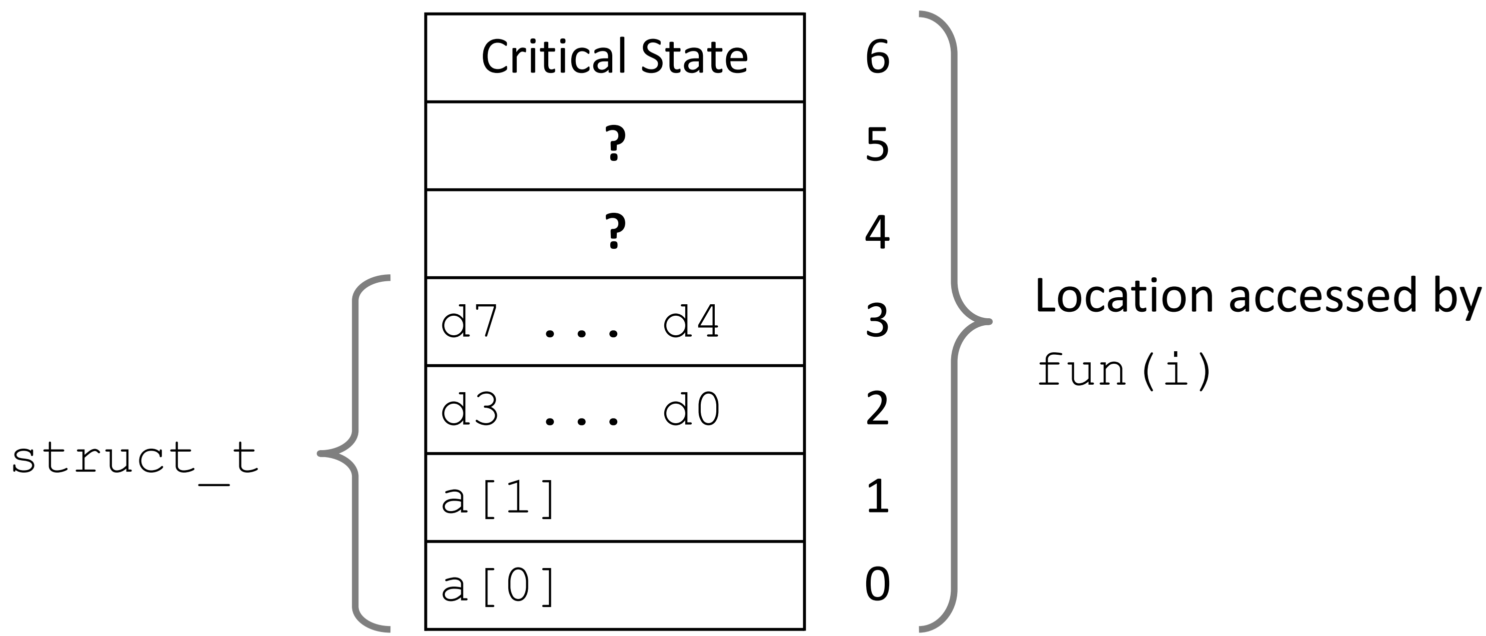
- Result is system specific

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)`	➙	`2.00000061035156`
`fun(4)`	➙	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, Ruby, Python, ML, …
  - 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 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];
}
```

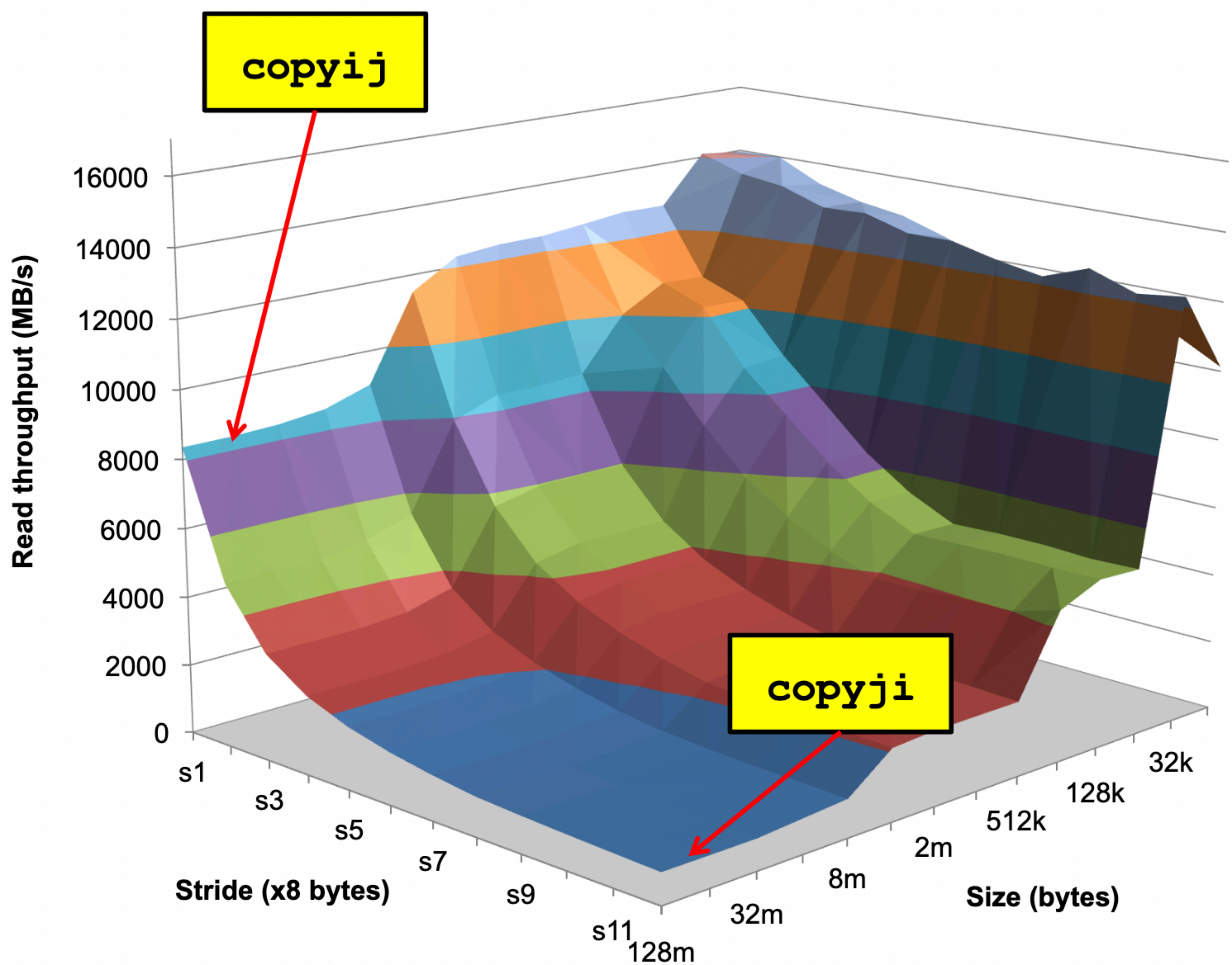
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms **81.8ms**

2.0 GHz Intel Core i7 Haswell

- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how 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

- Computer Architecture
  - Design pipelined processor in Verilog
- Operating Systems
  - Implement sample portions of operating system
- Compilers
  - Write compiler for simple language
- Networking
  - Implement and simulate network protocols

Course Perspective (Cont.)

This 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
  - **This course brings out the hidden hacker in everyone!**

Role within CS/ECE Curriculum

Imperative Programming

Foundation of Computer Systems
Underlying principles for hardware, software, and networking

CS2011

CS Systems
- Cloud Computing
- Computer Security
- Operating Systems
- Compiler Design
- Database Applications
- Parallel Computing
- Distributed Systems
- Computer Networks
- Database Systems

ECE Systems
- Computer Security
- Malware Analysis
- Intro to Embedded Systems
- Computer Networks
- Computer Architecture
- Wireless Networking
- Cyberphysical Systems

CS Graphics
- Computer Graphics
- Comp. Photography

Academic Integrity

- **DON'Ts**
  - 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** for solutions, discussions, tutorials, blogs, other universities' systems courses,... in English or any other language
  - Copying code from a previous course or online solution
    - You are only allowed to use code I supply, or from the book author's website

- **OK**
  - Explaining how to use systems or tools to others
  - Helping others with high-level design issues
    - Code / pseudo-code is **NOT** high level

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016, ISBN 0-13-409266-X
    - ***Hardcopy*** also available on Amazon (rent, used, or new "more expensive than Pearson")
    - Electronic edition available here (Rent 180 days *$44.99*, Lifetime *$74.99*)
    - Do not buy an *earlier edition (1st / 2nd)*
    - *DO NOT BUY A PAPERBACK/INTERNATIONAL EDITION (HAS MANY ISSUES)*
    - *DO NOT BUY KINDLE (BASED ON INTERNATIONAL VERSION)*
  - Book Website: https://csapp.cs.cmu.edu
  - This book really matters for the course! (**REQUIRED**)
    - How to solve labs
    - Practice problems typical of exam problems (or any other potential homework assignments)

- **Brian Kernighan and Dennis Ritchie,**
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators
  - Even though it does not cover more recent extensions of C

Note on Textbook Global Edition

- **Quote from textbook's website**

  Unfortunately, the publisher arranged for the generation of a different set of practice and homework problems in the global edition. The person doing this didn't do a very good job, and so these problems and their solutions have many errors. We have not created an errata for this edition.

Course Components (Subject to Change)

Lectures

- Higher level concepts
- May be utilized to teach you some applied concepts, important tools and skills for labs, deeper clarification of certain concepts, exam coverage, **demonstrations**, etc.
- **May** run random online Quizzes on canvas

Lab Assignments (~5-6)

- The heart of the course
- About 1-2 weeks each
- Provide in-depth understanding of an aspect of systems
- Programming and measurement

Homework Assignments (~2)

- Mainly arithmetic calculations, etc.. (no coding)

Exam(s)

- Current Options: 1) Midterm + Final OR, a 2) Comprehensive Final
- Test your understanding of concepts & mathematical principles, etc…
- Will **most likely** be held online through **canvas**

Getting Help

- **Instructor's office hours:**
  - Answering questions about theoretical concepts we did cover in lecture
  - Time/Location: Check Canvas

- **TA's office hours:**
  - Technical help with labs, homework assignments, etc.
  - Most likely will run tutoring hours (Details will be added to Canvas)
  - Time/Location: Check Canvas

- **CANVAS**
  - PLEASE CHECK CANVAS CONSTANTLY FOR UPDATES

Timeliness

Late Penalty

- You **may** submit your assignments up to 24 hours **late** without getting penalized **as long as this does not happen more than 2 times** throughout the semester.
- If you submit late within the 24-hour period **more than two times** throughout the semester, **OR** if you submit after the 24-hour period, you will risk receiving a late penalty of up to **15% per day**
- No late submission is allowed **3 days after due date**
  - Graders can easily miss your submission

Catastrophic events

- Major illness, death in family, …
- Formulate a plan (with your academic advisor) to get back on track

Advice

- Once you start running late, **it's really really hard to catch up**

During Lecture

- **Laptop use is encouraged**
  - No GAMING please :) !!!

- **Please no electronic communications**
  - No email, instant messaging, cell phone calls, etc

- **Be Present**
  - Attendance will not be taken but is very **strongly encouraged**!!!

- **Please NO recordings of ANY KIND**

Grading (Tentative)

- **Labs/Homeworks (70%)**
  - `Most likely weighted according to effort`
- **Exams (30%)**

Programs and Data

Topics

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

Possible Lab Assignments

- Lab0: Test/refresh your C programming abilities
- Lab1: Manipulating bits
- Lab2: The basics of Assembly/Disassembly
- Lab3: The basics of code injection attacks

The Memory Hierarchy

Topics

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

Possible Lab Assignments

- Lab4: 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

- **Possible Lab Assignments**
  - L5: Writing your own malloc package
    - Get a real feel for systems-level programming

Exceptional Control Flow

Topics

- Hardware exceptions, processes, process control, Unix signals, nonlocal jumps BN
- Includes aspects of compilers, OS, and architecture

Possible Lab Assignments

- Lab6: Writing your own Unix shell.
  - A first introduction to concurrency

Welcome and Enjoy!