CSCI 2400 – Computer Systems

Instructor:	**Nicholas V. Lewchenko (Nick)**	nile1033@colorado.edu
TA:	**Mayank Joshi**	Mayank.Joshi@colorado.edu
TA:	**Insoo Lee**	Insoo.Lee@colorado.edu
TA:	**Austin Pahl**	Austin.Pahl@colorado.edu
GSS:	**Amatullah Sethjiwala**	Amatullah.Sethjiwala@colorado.edu
GSS:	**Shruthi Sridharan**	Shruthi.Sridharan@colorado.edu

First things first

- **This is a challenging class!**
- **A full semester in 8 weeks.**
- **Start assignments as soon as they are posted!**

- **First big lab is due next Monday (2020-06-08)**

Recitations

- **Recitations:**
  - 301-REC: Wed, 9:15am-10:35am
  - 302-REC: Wed, 11:00am-12:00pm
  - 303-REC: Wed, 9:15am-10:35am
- **Check mycuinfo for your section, Moodle for Zoom link**
  - Can’t find your section? You’re probably in 303-REC.

Grading: Labs

- **5 Labs, total 60% of course grade**
  - Interview graded, lots of work!
  - First posted later today, due next Monday night
- **Turned in via GitHub Classroom**

In general, things due at 10:00pm

Nothing accepted late!*

Grading: Exams

- **2 Exams, total 32% of course grade**
  - Conducted through Moodle
  - 24-hour window, single 80-minute attempt
- **“Open Book”, using:**
  - Your notes
  - The lecture slides/video
  - Textbook
  - Simple calculator

Grading: Self-Study Quizzes

- **Weekly quizzes, total 8% of course grade**
  - These cover textbook readings and lecture
  - *Computer Systems*, *A Programmer's Perspective* (3rd Edition) by Bryant and O'Halloran
  - Allow multiple attempts
  - Released Monday morning, due Sunday night
  - This week, four quizzes covering Chapters 1,2

The Many Online Services

- **Moodle**
  - Exams, quizzes, lecture recordings
- **Piazza**
  - Weekly overviews, asynchronous office hours
- **Zoom**
- **coding.csel.io**
  - Environment for working on labs
  - Tutorial in this lecture, more in recitation
- **GitHub Classroom**
  - Turn-in system for labs, uses Git!
  - Invite later today, tutorial in recitation

Important Dates

- **Exam 1:** Thursday, 2020-07-02
- **Exam 2:** Friday, 2020-07-24 (last day of class)
- **Last day to drop:** Monday, 2020-06-08 (1 week from now!)

Course Overview

These slides adapted from materials provided by the textbook authors

Overview

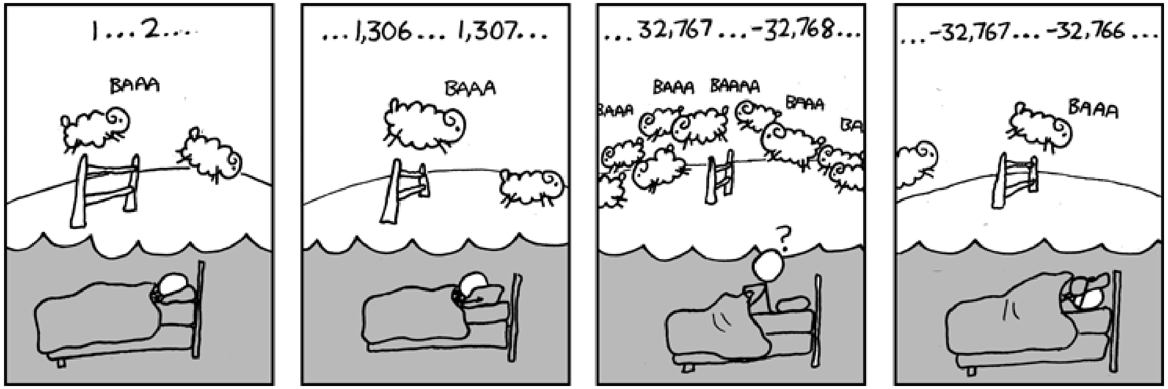
- Course theme
- Five realities

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

- **Most CS and CE courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes**
  - 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, Storage Systems, etc.

Great Reality #1:
`int` is not Integers, `float` is 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)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 3.14
    - 1e20 + (-1e20 + 3.14) ??

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all the *usual* properties**
  - Necessary due to finiteness of data representations
  - Integer operations satisfy “ring” properties
    - Commutativity, Associativity, Distributivity
  - Floating point operations satisfy “ordering” properties
    - Monotonicity, values of signs

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

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)    3.14
fun(6)    Segmentation fault
```

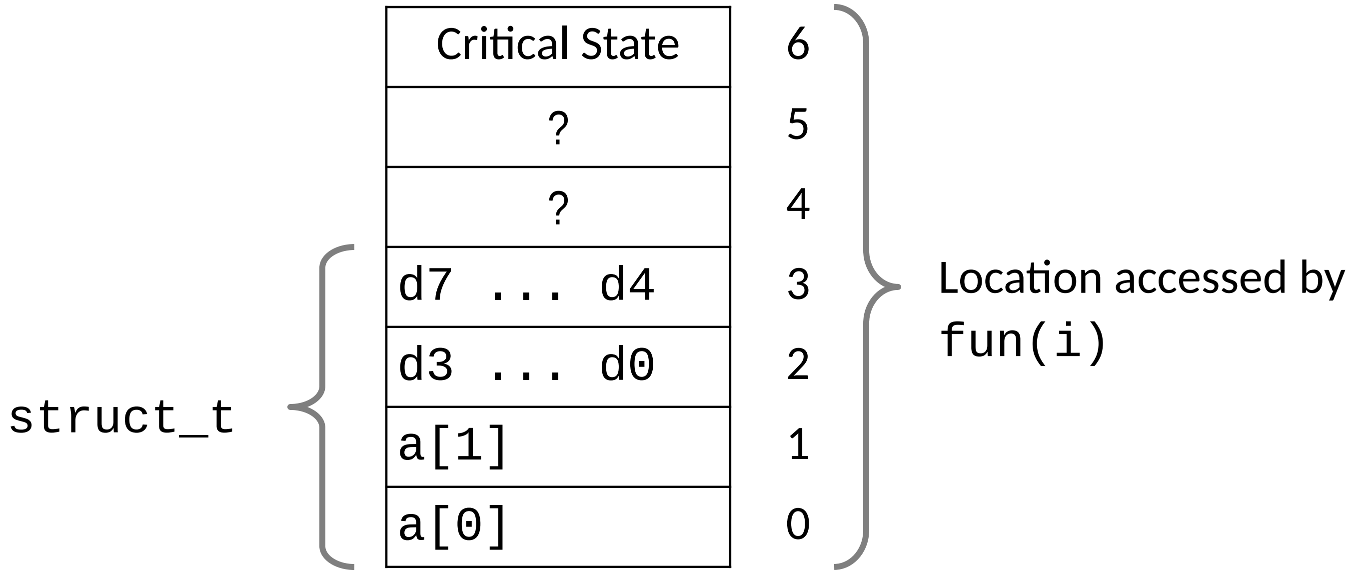
- Result is system specific

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)`	3.14
`fun(6)`	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];
}
```

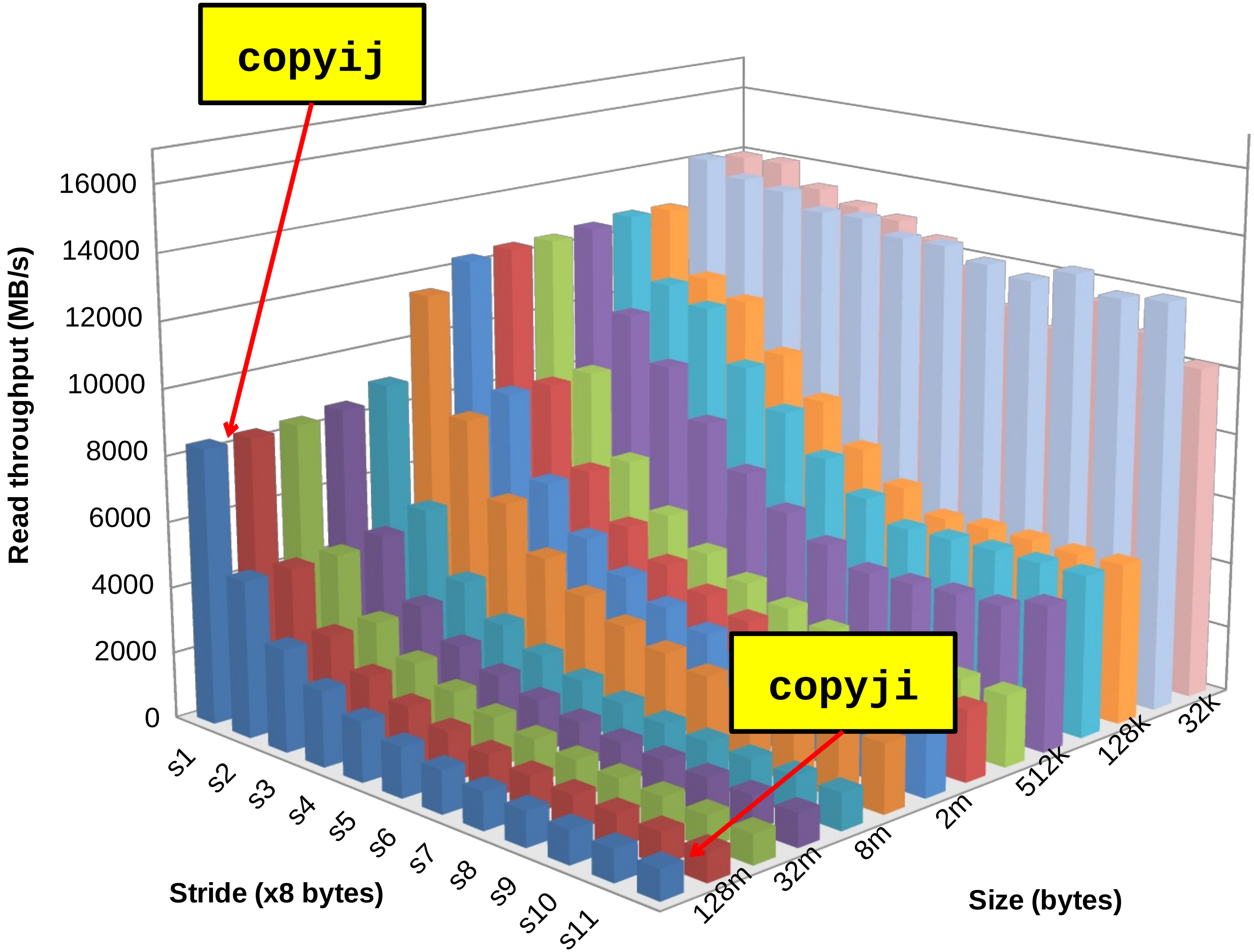
4.3ms

2.0 GHz Intel Core i7 Haswell

81.8ms

- **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

Welcome and Enjoy!