Course Overview

CS-392-A Systems Programming

Instructors:

Georgios Portokalidis *(This section)*

Srinivas Sridharan (Other section)

Updated textbook and content!

Based on CMU's Computer Systems course

Overview

- Course theme
- Five realities
- Academic integrity
- Logistics

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

- **Most CS 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

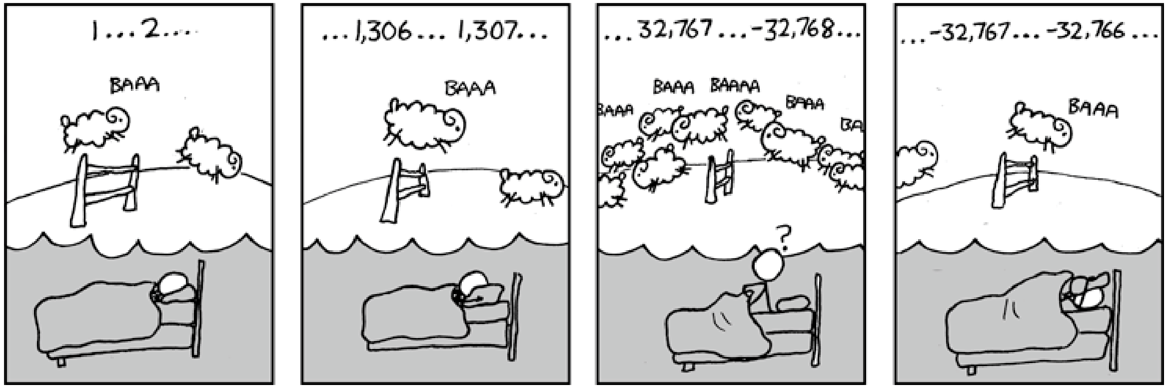
Course Theme:
Useful outcomes from this course

- **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**
  - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

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)?**
  - 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)  →   3.14
fun(6)  →   Segmentation fault
```

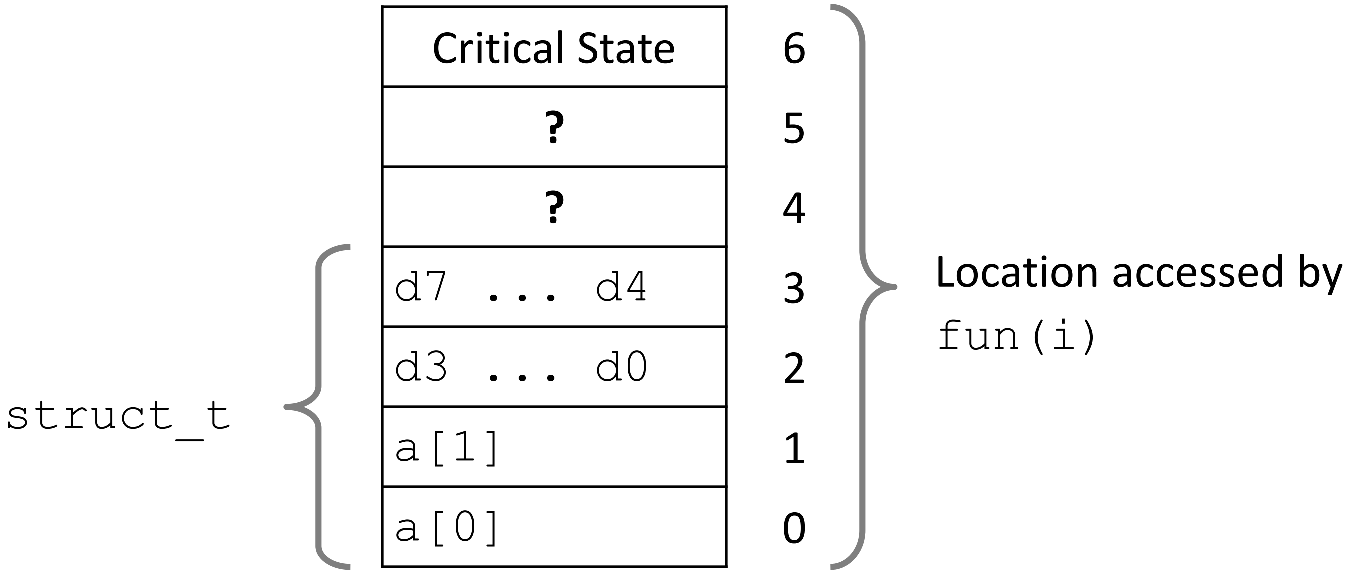
- Result is system specific

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

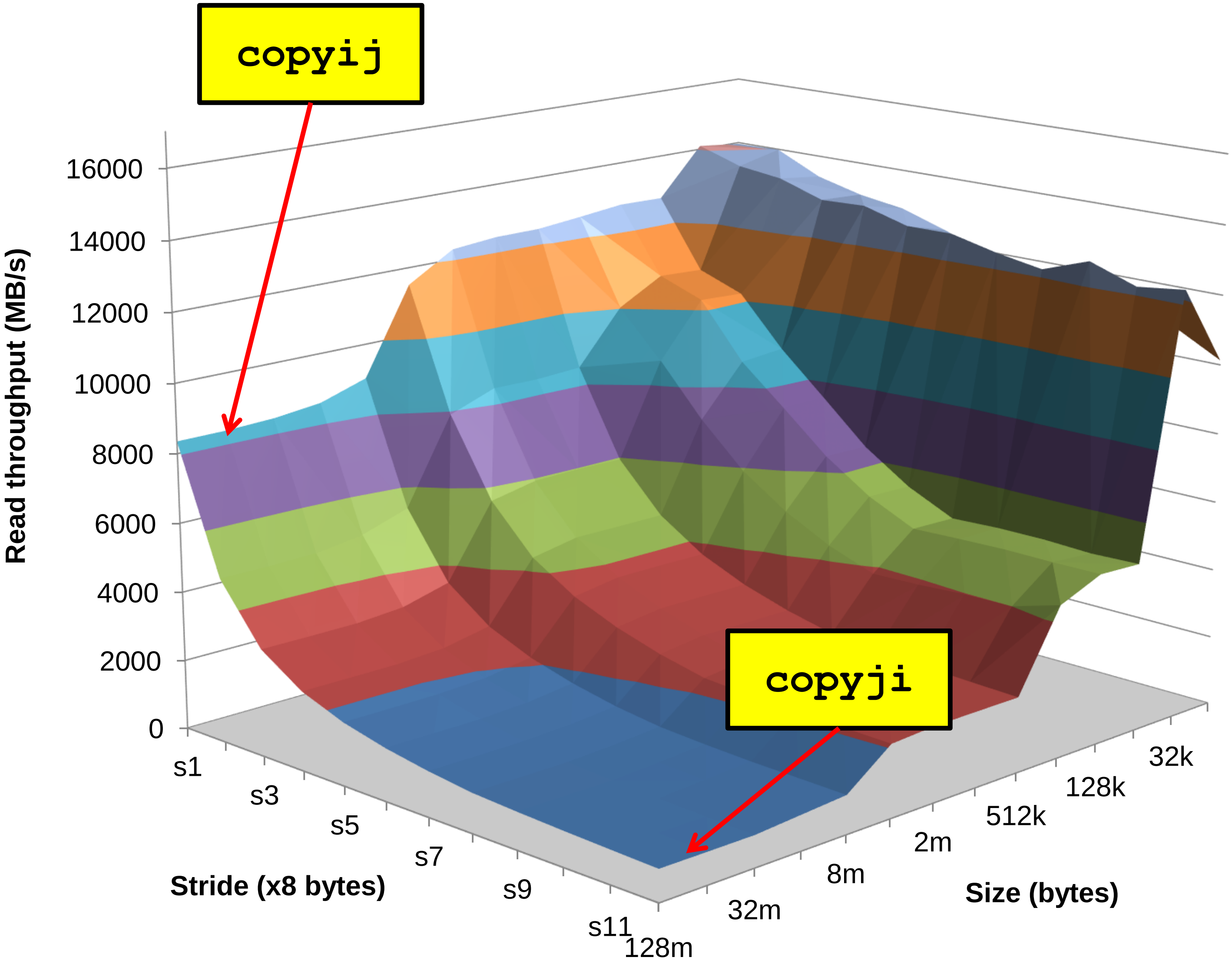
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
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

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.)

- **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!**

Staff for This Section

- **Instructor: Georgios Portokalidis (that's me)**
  - Email: Georgios.Portokalidis@stevens.edu

- **Course Assistants**
  - Nicholas Cyprus (ncyprus@stevens.edu)
  - Benjamin Iofel (biofel@stevens.edu)

Cheating: Description

- **Please pay close attention**
- **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 for solutions
  - Copying code from a previous course or online solution
    - 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
- **Ignorance is not an excuse**

Cheating: Consequences

- **Penalty for cheating:**
  - You will be reported to the Dean and the Honor board
  - Penalties may include suspension and expulsion
- **Detection of cheating:**
  - We have sophisticated tools for detecting code plagiarism
- **Don't do it!**
  - Start early
    - You wont, but really start early!
  - Ask us for help when you get stuck

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
- **Brian Kernighan and Dennis Ritchie,**
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators
- **Helpful but optional: W. Richard Stevens, Stephen A. Rago**
  - Advanced Programming in the UNIX Environment , 3rd ed., Addison-Wesley Professional, 2013

Course Components

- **Lectures (first 100 mins)**
  - Higher level concepts

- **Lectures (last 50 mins)**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
  - In-class programming exercises
    - Make sure you have enough battery left

- **Quizzes (2 midterms)**
  - Test your understanding of concepts & mathematical principles

Course Components (cont)

- **Take-home assignments or labs (5)**
  - The heart of the course
  - 2-3 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
  - Will be done individually
- **Final project**
  - Similar to labs
  - Done at the end
  - Small groups of maximum 2 students allowed
  - Small demo and presentation of measurement results

Getting Help

- **Canvas**
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures and assignments
- **Slack**
  - https://cs392-2017s.slack.com/x-132211270262-132212553142/signup
  - Join with your Stevens email
  - Discuss with your classmates
  - Get feedback from CAs and instructors
  - Do not post grading-related requests and grievances
- **Gradescope**
  - https://gradescope.com/courses/6111
  - Submission of assignments
  - Entry code 9X4BY9

Getting Help

- **Instructor Office hours:**
  - Tuesdays, 5:00pm-7:00pm, Lieb 214
  - Try emailing first
  - Appointments for meeting outside office hours may also possible
  - Contact through email NOT canvas

- **CA Office hours**
  - Wednesdays, 4:00pm-6:00pm, Lieb lounge (3^{rd} floor)
  - Fridays, 3:00pm-5:00pm, Lieb lounge (3^{rd} floor)
  - Check Canvas for announcements
  - Appointments for meeting outside office hours may also possible

Policies: Labs And Exams

- **Work groups**
  - You must work alone on all take-home assignments
- **Handins**
  - Labs due at 11:59pm on Tuesday
  - Electronic handins using through gradescope
    - See assignment description for exactly what to download
- **Quizzes**
  - Quizzes will be during first part of lecture
  - See syllabus for more info
- **Appealing grades**
  - By email me within 7 days from the posting of grades
  - Appeal can lead to **higher** or **lower** grade

Development Platform

- **Use Xubuntu 16.04 x86-64 VM provided**
  - https://www.dropbox.com/s/144ahwra1i6uaxk/cs392-xubuntu16.04.zip?dl=0
  - You can use your own platform but your submitted code must compile and execute correctly on this VM
  - Submitted files must be as described in assignment
  - You can run the VM using
    - Vmplayer – freely available from Vmware
    - VirtualBox – free software

Timeliness

- **Grace days**
  - **5 grace days for the semester**
  - **Limit of 2 grace days per assignment used automatically**
  - Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
  - Save them until late in the term!

- **Lateness penalties**
  - Once grace day(s) used up, get penalized **15% per day**
  - No handins later than **3 days after due date**

Timeliness (cont.)

- **Catastrophic events**
  - Major illness, death in family, …
  - Need to contact the office of undergraduate academics
    - They will contact me to make arrangements

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

Other Rules of the Lecture Hall

- **Laptops: permitted**
  - But you need to make sure to have enough power for any in-class assignments at the end of the lecture
- **Electronic communications:** ***forbidden***
  - No email, instant messaging, cell phone calls, etc
- **Presence in lectures: voluntary, recommended**
- **No recordings of ANY KIND**

Policies: Grading

- **Quizzes (30%): midterm I (15%), midterm II (15%)**
- **Take-home assignments / labs (50%): weighted according to effort**
- **Final project / lab (20%)**
- **In-class assignments (bonus 5%)**
- **Final (letter) grades will be based on a slight modification of a straight scale**
  - You pass if you score >50%

Topics

Week #	Topics
1	Introduction Bits and Bytes
2	Integers Machine Prog: Basics
3	Machine Prog: Control Machine Prog: Procedures
4	C language
5	More C language The Memory Hierarchy Cache Memories
6	Linking ECF: Exceptions
7	ECF: Processes & Signals
8	System Level I/O
9	Virtual Memory
10	Dynamic Memory Allocation
11	Network Programming
12	Concurrent Programming
13	The Future of Computing
14	F1 demo and short presentations

Welcome

and Enjoy!