Course Overview

~~18-600~~ **18-613: Foundations of Computer Systems**
1st Lecture, Jan 15, 2019

Instructors:

Franz Franchetti

Seth Goldstein

Brandon Lucia

Brian Railing

The course formerly known as 18-600

Overview

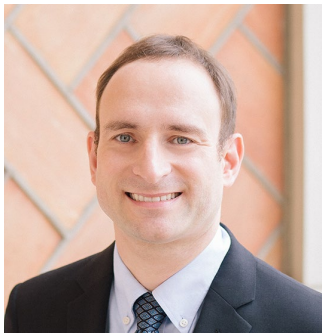
- ~~18-600~~ 18-613
- Big Picture
- Academic integrity
- Logistics and Policies

~~18-600~~ 18-613

~~18-600~~ 18-613

- **This semester: 18-613, NOT 18-600**
  - Full replacement in ECE curriculum w.r.t. prerequisites and requirements
  - Pittsburgh and Silicon Valley sections
  - No Silicon Valley instructor, but Silicon Valley TAs
  - Course will be officially renumbered in S3 soon
- **18-613 = 15-513 + 10 lectures + extra lab + extra exam questions**
  - 15-513 lectures are online in Canvas/Panopto (+ Rashid if space permits)
  - 18-613 specific additional lectures are here (DH A302/B23 211) and recorded in Canvas, schedule in Canvas
  - Some lecture slots will be discussion slots (faculty recitations)
  - Pittsburgh TA office hours are joint with 15-213/15-513/18-213
  - Recitations are 18-613 specific (Section 18-613A/B/C/SV)
  - Course web page: http://www.cs.cmu.edu/~213/
- **My office hours will be in my office + BlueJeans ID 4122688297**

Instructors: The 1x-x13 Team



18-613
Franz Franchetti

15-213
Seth C. Goldstein

18-213
Brandon Lucia

15-513
Brian Railing

I am your main contact and official instructor but you may hear from/interact with all of us

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 213/513**
  - 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.

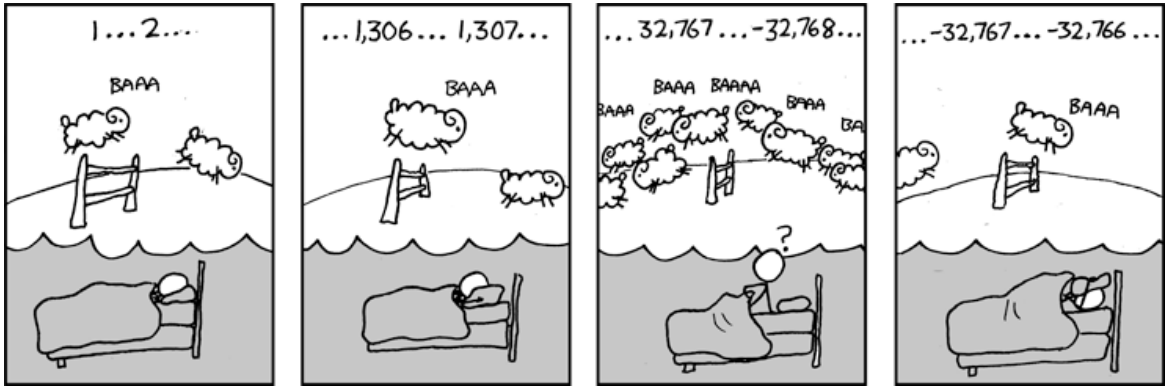
It's Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don't forget reality
- **Most CS 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
  - 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)?**
  - 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
```

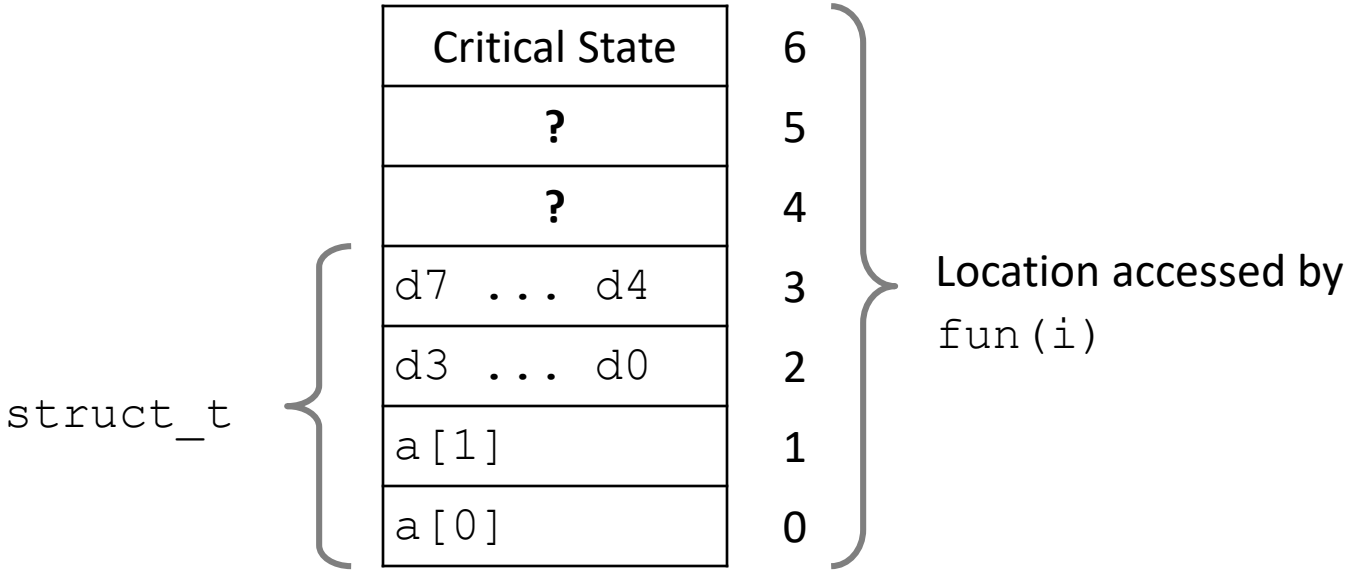
- Result is system specific

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) -->     2.0000061035156
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];
}
```

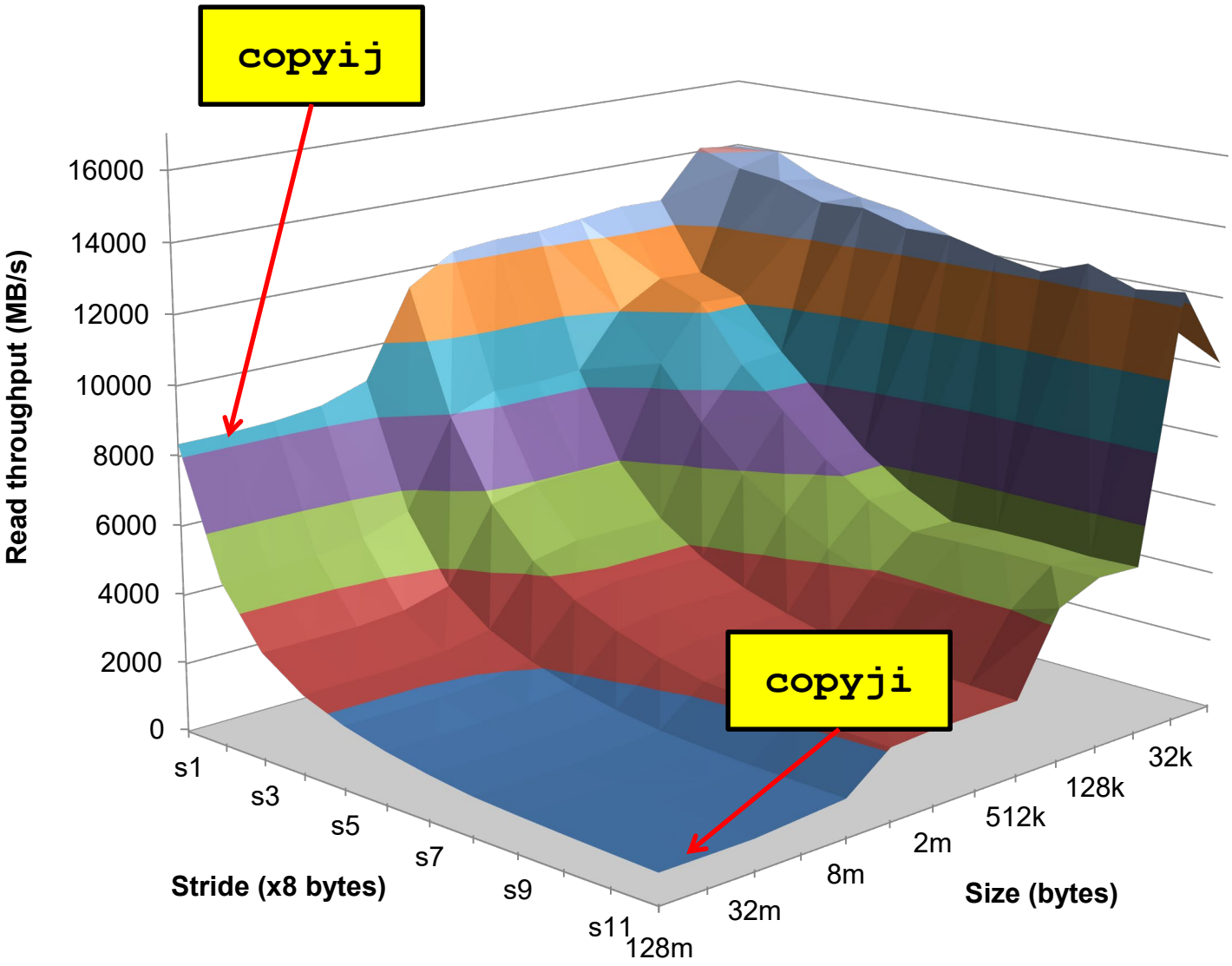
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms **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

613 Extra: Great Reality #6:
ECE MS Students in 18-613 learn extra stuff

Date	Topic
January 15	**Overview:** Scope and mechanics of 18-613 and relationship to 15-513
January 22	**State of computing:** The state-of-the-art of computer systems w.r.t. 18-613
January 31	**Data structures review:** Review of data structures needed in the 18-613 labs
February 19	**SIMD instructions:** floating-point and integer SIMD instructions in x86
February 21	**High performance CPU cores:** super-scalar, out-of-order, speculation, etc.
March 5	**Multicore CPUs:** Memory hierarchy, cache coherency, etc. (*Brandon Lucia*)
March 7	**Accelerators:** GPUs, FPGAs, and Xeon PHI, and AI/ML accelerators
April 4	**Debugging:** It is a science, not a black art. MallocLab and ProxyLab howto.
April 9	**Automatic Performance Tuning:** Expert performance engineering
April 25	**Parallel computing:** Concepts and frameworks, shared and distributed mem.

613: More hardware centric beyond 213/513

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**
  - By knowing more about the underlying system, you 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!**

18-613 broadens this a bit and digs a bit deeper

Academic Integrity

Please pay close attention, especially if this is your first semester at CMU

http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15213-s19/www/academicintegrity.html

Cheating/Plagiarism: Description

- **Unauthorized use of information**
  - **Borrowing code:** by copying, retyping, ***looking at*** a file
  - **Describing:** verbal description of code from one person to another
    - ***Even if you just describe/discuss how to put together CS:APP code snippets to solve the problem***
  - **Searching the Web** for solutions, discussions, tutorials, blogs, other universities' 213 instances,… in English or any other language
  - **Copying code** from a previous course or online solution
  - **Reusing *your* code** from a previous semester (here or elsewhere)
    - ***If you take the course this semester, all work has to be done this semester***
  - **Unattributed use of legal copies:** all code not written by the student must be attributed (book, handout, recitation examples,…)

Cheating/Plagiarism: Description (cont.)

- **Unauthorized supplying of information**
  - **Providing copy:** Giving a copy of a file to someone
  - **Providing access:**
    - Putting material in unprotected directory
    - Putting material in unprotected code repository (e.g., Github)
  - **Applies to this term and the future**
    - There is no statute of limitations for academic integrity violations

Cheating/Plagiarism: Description

- **What is NOT cheating?**
  - Explaining how to use systems or tools
  - Helping others with *high-level* design issues
  - Using code supplied by us ***(needs to be attributed)***
  - Using code from the CS:APP web site ***independently***

- **See the course syllabus for details.**
  - Ignorance is not an excuse

Cheating: Consequences

- **Penalty for cheating:**
  - **Best case:** -100% for assignment
    - ***You would be better off to turn in nothing***
  - **Worst case:** Removal from course with failing grade
    - ***This is the default***
  - Permanent mark on your record
  - Loss of respect by you, the instructors and your colleagues
  - ***If you do cheat – come clean asap!***
- **Detection of cheating:**
  - We have sophisticated tools for detecting code plagiarism
  - In Fall 2015, 20 students were caught cheating and failed the course.
    - Some were **expelled** from the University
  - In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.
  - In Spring 2017 we caught people looking at Chinese-language blogs; 30+ AIV letters
- **Don't do it!**
  - Manage your time carefully
  - Ask the staff for help when you get stuck

Some Concrete Examples:

- **This is Cheating:**
  - Searching the internet with the phrase 15-213, 18600, 213, 18213, malloclab,…
    - That's right, just entering it in a search engine
  - Looking at someone's code on the computer next to yours
  - Giving your code to someone else, now or in the future
  - Posting your code in a publicly on the Internet, now or in the future
  - Discussing with your friends how to solve a lab by putting together CS:APP code fragments
  - Hacking the course infrastructure
- **This is OK (and encouraged):**
  - Googling a man page for fputs
  - Asking a friend for help with gdb
  - Asking a TA or course instructor for help, showing them your code, …
  - Looking in the textbook for a code example
  - Talking about a (high-level) approach to the lab with a classmate

How it Feels: Student and Instructor

- **Fred is desperate.** He can't get his code to work and the deadline is drawing near. In panic and frustration, he searches the web and finds a solution posted by a student at U. Oklahoma on Github. He carefully strips out the comments and inserts his own. He changes the names of the variables and functions. Phew! Got it done!
- **The course staff** run checking tools that compare all submitted solutions to the solutions from this and other semesters, along with ones that are on the Web.
  - Remember: We are as good at web searching as you are
- **Meanwhile, Fred has had an uneasy feeling:** Will I get away with it? Why does my conscience bother me?
- **Fred gets email from an instructor:** "Please see me tomorrow at 9:30 am."
  - Fred does not sleep well that night

Why It's a Big Deal

- **This material is best learned by doing**
  - Even though that can, at times, be difficult and frustrating
  - Starting with a copy of a program and then tweaking it is very different from writing from scratch
    - Planning, designing, organizing a program are important skills
- **We are the gateway to other system courses**
  - Want to make sure everyone completing the course has mastered the material
- **Industry appreciates the value of this course**
  - We want to make sure anyone claiming to have taken the course is prepared for the real world
- **Working in Teams and Collaboration is an Important Skill**
  - But only if team members have solid foundations
  - This course is about foundations, not teamwork

How it Feels: Student and Instructor

- **The instructor feels frustrated.** His job is to help students learn, not to be police. Every hour he spends looking at code for cheating is time that he cannot spend providing help to students. But, these cases can't be overlooked
- **At the meeting:**
  - Instructor: "Explain why your code looks so much like the code on Github."
  - Fred: "Gee, I don't know. I guess all solutions look pretty much alike."
  - Instructor: "I don't believe you. I am going to file an academic integrity violation."
    - Fred will have the right to appeal, but the instructor does not need him to admit his guilt in order to penalize him.
- **Consequences**
  - Fred may (most likely will) be given a failing grade for the course
  - Fred will be reported to the university
  - A second AIV will lead to a disciplinary hearing
  - Fred will go through the rest of his life carrying a burden of shame
  - The instructor will experience a combination of betrayal and distress

A Scenario: Cheating or Not?

Alice is working on malloc lab and is just plain stuck. Her code is seg faulting and she doesn't know why. It is only 2 days until malloc lab is due and she has 3 other assignments due this same week. She is in the cluster.

Bob is sitting next to her. He is pretty much done.

Sitting next to Bob is Charlie. He is also stuck.

- **1. Charlie gets up for a break and Bob makes a printout of his own code and leaves it on Charlie's chair.**
  - Who cheated: Charlie? Bob?
- **2. Charlie finds the copy of Bob's malloc code, looks it over, and then copies one function, but changes the names of all the variables.**
  - Who cheated: Charlie? Bob?

Another Scenario

Alice is working on malloc lab and is just plain stuck. Her code is seg faulting and she doesn't know why. It is only 2 days until malloc lab is due and she has 3 other assignments due this same week. She is in the cluster.

Bob is sitting next to her. He is pretty much done.

Sitting next to Bob is Charlie. He is also stuck.

- **1. Bob offers to help Alice and they go over her code together.**
  - Who cheated: Bob? Alice?
- **2. Bob gets up to go to the bathroom and Charlie looks over at his screen to see how Bob implemented his free list.**
  - Who cheated: Charlie? Bob?

Another Scenario (cont.)

- **3. Alice is having trouble with GDB. She asks Bob how to set a breakpoint, and he shows her.**
  - Who cheated: Bob? Alice?
- **4. Charlie goes to a TA and asks for help**
  - Who cheated: Charlie?

- **If you are uncertain which of these constitutes cheating, and which do not, please read the syllabus carefully. If you're still uncertain, ask one of the staff**

Version Control: Your Good Friend

- **We will be using Github Education**
  - Assignment distribution
  - Your workspace
    - Use your course account, rather than a personal Github account
- **Use as you should a version server**
  - Commit early and often
  - Document your commits
  - Missing GIT history can count against you
- **How we use it**
  - If we suspect academic integrity issues, we can see if commit history looks reasonable.
    - Steady, consistent, and sustained work
    - It can serve as your character witness

How to Avoid AIVs

- **Start early**
- **Don't rely on marathon programming sessions**
  - Your brain works better in small bursts of activity
  - Ideas / solutions will come to mind while you're doing other things
- **Plan for stumbling blocks**
  - Assignment is harder than you expected
  - Code doesn't work
  - Bugs hard to track down
  - Life gets in the way
    - Minor health issues
    - Unanticipated events

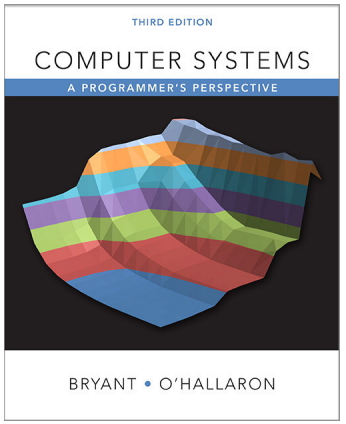
Logistics

18-613, 15-213/18-213 and 15-513

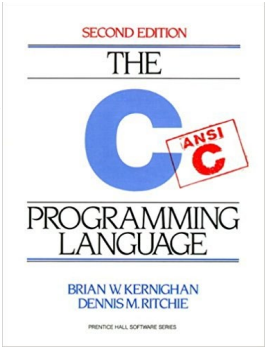
- **18-613**
  - Mainly ECE Masters students in Pittsburgh and Silicon Valley
  - 15-513 Lectures by video (on the website and Panopto)
    Pittsburgh students are welcome to attend 15-213/18-213 lectures in Rashid Auditorium if space is available
  - Additional live lectures according to schedule in Canvas (also recorded)
  - Additional "faculty recitations" in 18-613 lecture slots
  - Additional lab (extra credit)
  - Additional exam questions (covering extra material and extra lab)
- **15-213/18-213**
  - Only undergraduates
  - Live lectures
  - Recitations
- **15-513**
  - Only Masters students
  - Lectures by video (on the website and Panopto)
- **Everything else is the same for all the courses**

Textbooks

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



- **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



Course Components

- **Lectures**
  - Higher level concepts
- **Recitations**
  - More hands-on towards labs, ***start 1/30***
  - Discussions
- **Labs (8+1)**
  - The heart of the course
  - 1-2+ weeks each (except for extra lab—spaced out more)
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Exams (midterm + final)**
  - Test your understanding of concepts & mathematical principles

Getting Help

- **1x-x13 Class Web page: http://www.cs.cmu.edu/~213**
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions
  - Links to all infrastructure components (Canvas, Autolab, Piazza,…)
  - FAQ
- **1x-x13 Canvas and 18-613 Canvas**
  - 15-213/18-213/15-513 Canvas instance is gateway to all non-public course information
  - Additional 18-613 Canvas instance contains all 18-613 specific information
- **Piazza**
  - **Piazza 1x-x13:** Standard questions and course material related questions
  - **Piazza 18-613:** 18-613 specific questions (see Piazza post)
  - Best place for questions about assignments
  - By default, your posts will be private

Getting Help

- **Email**
  - Send email to individual TAs **only to schedule appointments**
  - **Special situations, emergencies, meeting requests: franzf@ece.cmu.edu**
- **Office hours (starting Tue Jan 22):**
  - **Pittsburgh TAs:** SMTWR(FS), 5:00–9:00pm, location TBD
  - **Silicon Valley TAs:** TBD
  - **Prof. Franchetti:** Tue 3:15pm EST/12:15pm PST
    Pittsburgh: HH A312, *Silicon Valley:* BlueJeans ID 4122688297
- **Recitations, start 1/30**
  - Section 18-613 A: Wed 4:30pm-5:50pm, HH B131 (TA: TBD)
    Section 18-613 B: Wed 4:30pm-5:50pm BH A53 (TA: TBD)
    Section 18-613 C: Wed 6:30pm-7:50pm HH B131 (TA: TBD)
    Section 18-613 SA: Wed 5pm-6:20pm B23 211 SV (TA: TBD)
- **Walk-in Tutoring (Pittsburgh)**
  - Details TBA. Will put information on class webpage.

18-613 Student HowTo

- Attend Lectures
- Attend Recitations and boot camps
- Start labs early (really) and use GIT properly
- TA office hours: we need to manage load and waiting time
  - lab-related concrete questions
  - must write them down before getting help
  - Time slots
- Faculty Office Hours
  - Grading, special cases, issues
  - Conceptual and longer questions
  - Open discussions
  - *Please stop by and have a coffee, espresso, tea or hot chocolate (Pittsburgh-only, sorry)*

Policies: Labs And Exams

- **Work groups**
  - You must work alone on all lab assignments
    no "programming parties" at coffee shops etc.
- **Handins**
  - Labs due at 11:59pm
  - Electronic handins using **Autolab** (no exceptions!)
    after a final GIT commit/push
- **Exams**
  - Exams will be online in network-isolated clusters (this may change)
  - Held over multiple days. Self-scheduled; just sign up! (Need to see re SV)
- **Appealing grades**
  - Exams & Problem Sets: Request via exam server
  - Labs: Via detailed private post to Piazza within 7 days of completion of grading
  - Follow formal procedure described in syllabus

Facilities

- **Labs will use the Intel Computer Systems Cluster**
  - The "shark machines"
  - `linux> ssh shark.ics.cs.cmu.edu`

  - 21 servers donated by Intel for 213/513
    - 10 student machines (for student logins)
    - 1 head node (for instructor logins)
    - 10 grading machines (for autograding)
  - Each server: Intel Core i7: 8 Nehalem cores, 32 GB DRAM, RHEL 7.5
  - Rack-mounted in Gates machine room
  - Login using your Andrew ID and password

Timeliness

- **Grace days**
  - **5 grace days for the semester**
  - **Limit of 0, 1, or 2 grace days per lab used automatically**
  - Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
- **Lateness penalties**
  - Once grace day(s) used up, get penalized **15% per day**
  - No handins later than **3 days after due date**
- **Catastrophic events**
  - Major illness, deat
  - Formulate a plan (
- **Advice**
  - Once you start running late, it's really hard to catch up
  - Try to save your grace days until the last few labs

Really, Really Hard!

Other Rules of the Lecture Hall

- **Laptops: permitted**
- **Electronic communications: forbidden**
  - No email, instant messaging, cell phone calls, etc
- **Presence in lectures (613): voluntary, recommended**
- **No recordings of ANY KIND**
- **Slides will be posted right before or soon after the lecture**
- **Videos post the next day the latest**

Policies: Grading

- **Exams (50%): midterm (20%), final (30%)**
  18-613 exams include extra material from lectures and ArchLab
- **Labs (50%): weighted according to effort**
- **ArchLab is extra credit**
- **Final grades based on a straight scale (90/80/70/60) with a small amount of curving**
  - Only upward

Programs and Data

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

- **Assignments**
  - L0 (C programming Lab): Test/refresh your C programming abilities
  - L1 (datalab): Manipulating bits
  - L2 (bomblab): Defusing a binary bomb
  - L3 (attacklab): The basics of code injection attacks

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.

Exceptional Control Flow

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

- **Assignments**
  - L5 (tshlab): Writing your own Unix shell.
    - A first introduction to concurrency

Virtual Memory

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

- **Assignments**
  - L6 (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**
  - L7 (proxylab): Writing your own Web proxy
    - Learn network programming and more about concurrency and synchronization.

18-613 Extra: Computer Architecture

- **Topics**
  - Microarchitecture implementation
  - Superscalar and out of order cores
  - Speculation
  - Multicore memory hierarchy
- **Assignments**
  - LX (ArchLab): Understand how assembly executes and use the GEM5 simulator
    - Learn about architecture level trade-offs and implementation issues
    - Learn about performance optimization both above and below the ISA abstraction level
  - Extra credit
  - Longer time active
  - Concepts tested in exams

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
  - Post intermediate results (anonymized) on Autolab scoreboard for glory!

Lab0: C Programming

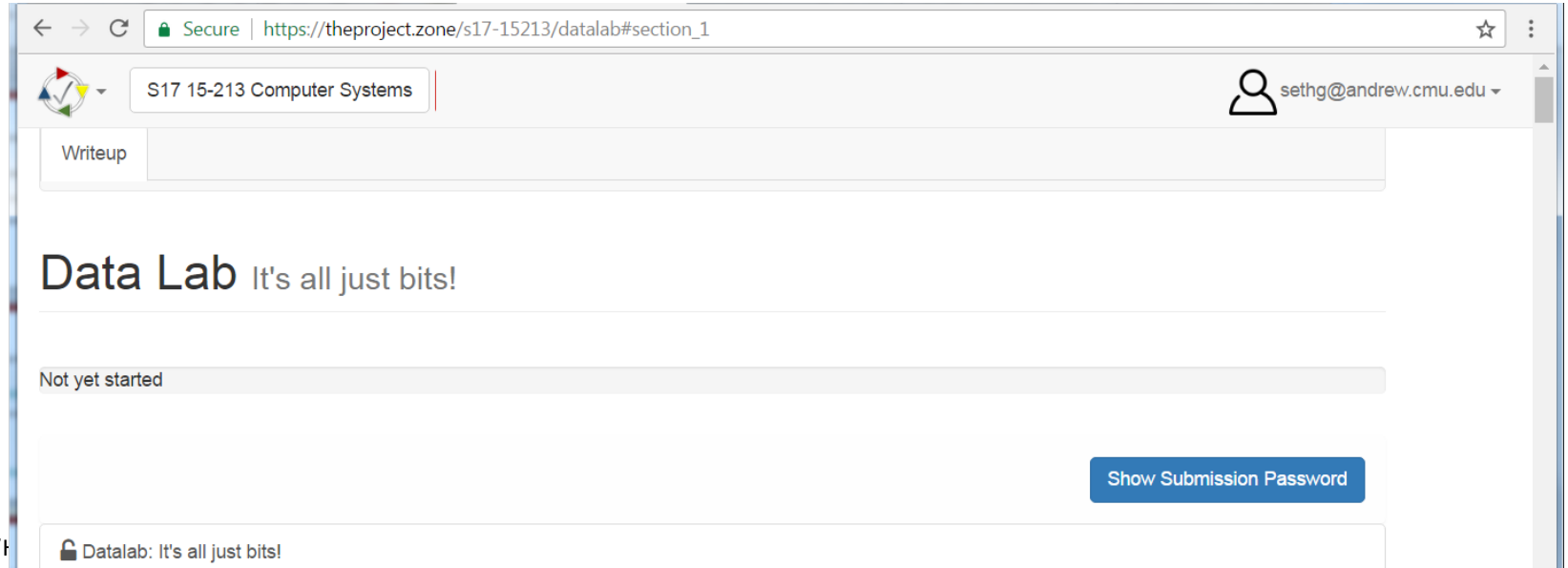
- **Due 1/22 11:59pm**
- **You can start now: download from 213 schedule page**
- **It should all be review:**
  - Basic C control flow, syntax, etc.
  - Explicit memory management, as required in C.
  - Creating and manipulating pointer-based data structures.
  - Implementing robust code that operates correctly with invalid arguments, including NULL pointers.
  - Creating rules in a Makefile
- **If this lab takes you more than 10 hours, please think hard about taking the course. And REALLY come to 18-613 Lecture 3**

Doing L1—L7 + LX

- https://autolab.andrew.cmu.edu/courses/15213-s19
  - Download the lab materials
  - (Usually as a tar file, so you will need to untar them in a new directory)
- **If you have questions**
  - Piazza (log in via Canvas)
  - Office hours
  - Walkin-tutoring

Theproject.zone

- Lab writeups (L1—L7) will be online
- As you read the writeup there will be assessment questions.
- When you have completed all the assessment questions you will get a submission code.
- Use the submission code when submitting your lab to autolab.
  - Not required for L0



Autolab (https://autolab.andrew.cmu.edu)

- **Labs are provided by the CMU Autolab system**
  - Project page: http://autolab.andrew.cmu.edu **– should have access on Fri**
  - Developed by CMU faculty and students
  - Key ideas: Autograding and Scoreboards
    - **Autograding:** Providing you with instant feedback.
    - **Scoreboards:** Real-time, rank-ordered, and anonymous summary.
  - Used by over 3,000 students each semester
- **With Autolab you can use your Web browser to:**
  - Download the lab materials
  - Handin your code for autograding by the Autolab server
  - View the class scoreboard
  - View the complete history of your code handins, autograded results, instructor's evaluations, and gradebook.
  - View the TA annotations of your code for Style points.

Version Control: Your Good Friend

- We use GIT classroom for version control
- Must be used by all students for lab 1—7
- Students must commit early and often
  at least at the end of each day working on a lab
- If a student is accused of cheating (plagiarism), we will consult the GIT server and look for a reasonable commit history
- Missing GIT history will count against you
- We may ask you to include a GIT hash on your submissions
- Learn how to use GIT now

Linux/Git bootcamp (Pittsburgh)

- How to tar and untar files
- How to set permissions on local and afs directories
- How to recover old files from git
- How to ssh to the lab machines
- How to use a make file
- And all the other things you were always afraid to ask …

- This Sunday 1/20 in Rashid Auditorium @ 7pm.
  (as with everything else, monitor web page for changes)

- Silicon Valley: Bootcamp slides will be available, and local TAs will be able to help

ECE Course Hub

- **Course Hub: coursehub@ece.cmu.edu**

More Information regarding 1x-x13

- Check Lecture 1 from 15-213/18-213/15-513 http://www.cs.cmu.edu/~213/schedule.html
- Infrastructure will be discussed there as well and is joint
- Their lecture is after our lecture…

Welcome
and Enjoy!