



Course Overview

18-213/18-613: Computer Systems

1st Lecture, Spring 2026

Overview

- **Introductions**
- **Big Picture**
 - Course theme
 - Five realities
 - How the course fits into the CS/ECE/INI curriculum
- **Academic integrity**
- **Logistics and Policies**

Instructors (18-213/613)

Swarun Kumar



Greg Kesden



Lectures & course content coordinated with
15-213, 14-513 & 15-513 teams

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 213/613

- Become more effective programmers
 - Able to find and eliminate bugs efficiently
 - Able to understand and tune for program performance
- Prepare for later “systems” classes across departments ...
 - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, Computer Security, 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

■ These abstractions have limits

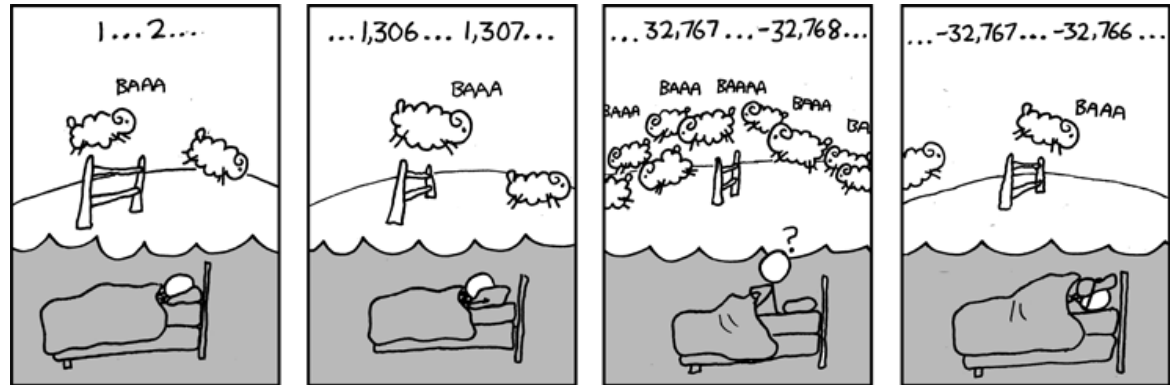
- Especially in the presence of bugs
- Need to understand details of underlying implementations
- Sometimes the abstract interfaces don't provide the level of control or performance you need

Great Reality #1:

Ints are not Integers, Floats are not Reals

■ Example 1: Is $x^2 \geq 0$?

- Float's: Yes!



- Int's:

- $40000 * 40000 \rightarrow 1600000000$
- $50000 * 50000 \rightarrow ?$

■ Example 2: Is $(x + y) + z = x + (y + z)$?

- Unsigned & Signed Int's: Yes!
- Float's:

- $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
- $1e20 + (-1e20 + 3.14) \rightarrow ??$

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(5) --> Segmentation fault
```

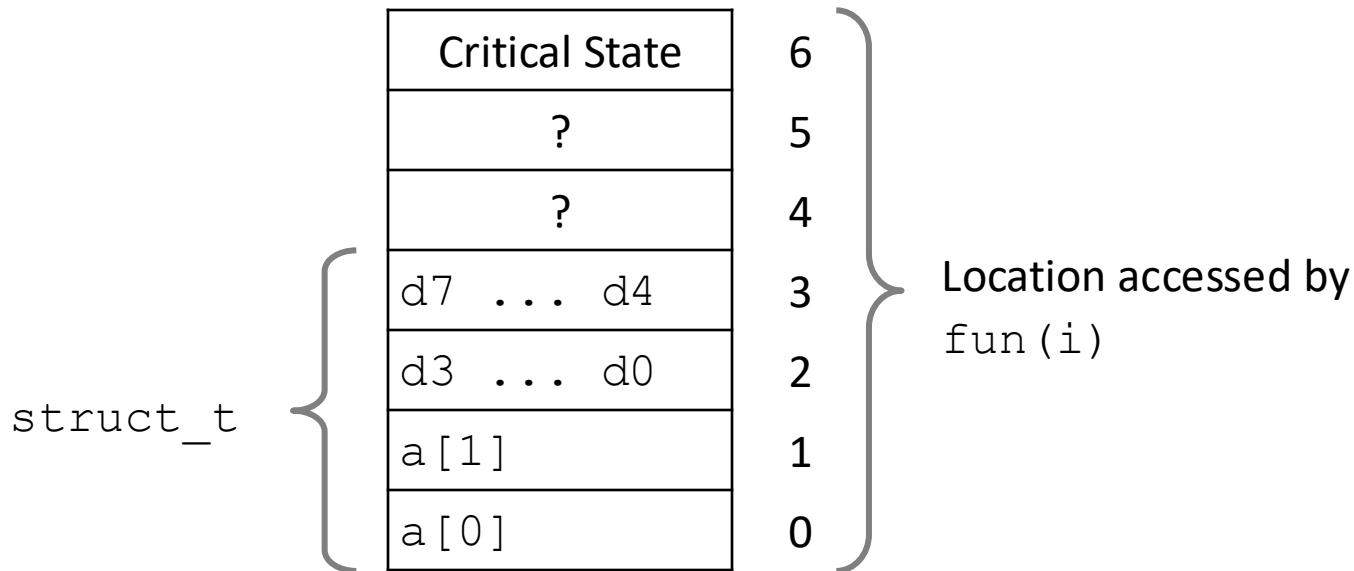
- Result is system specific

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];
}
```

4.3ms

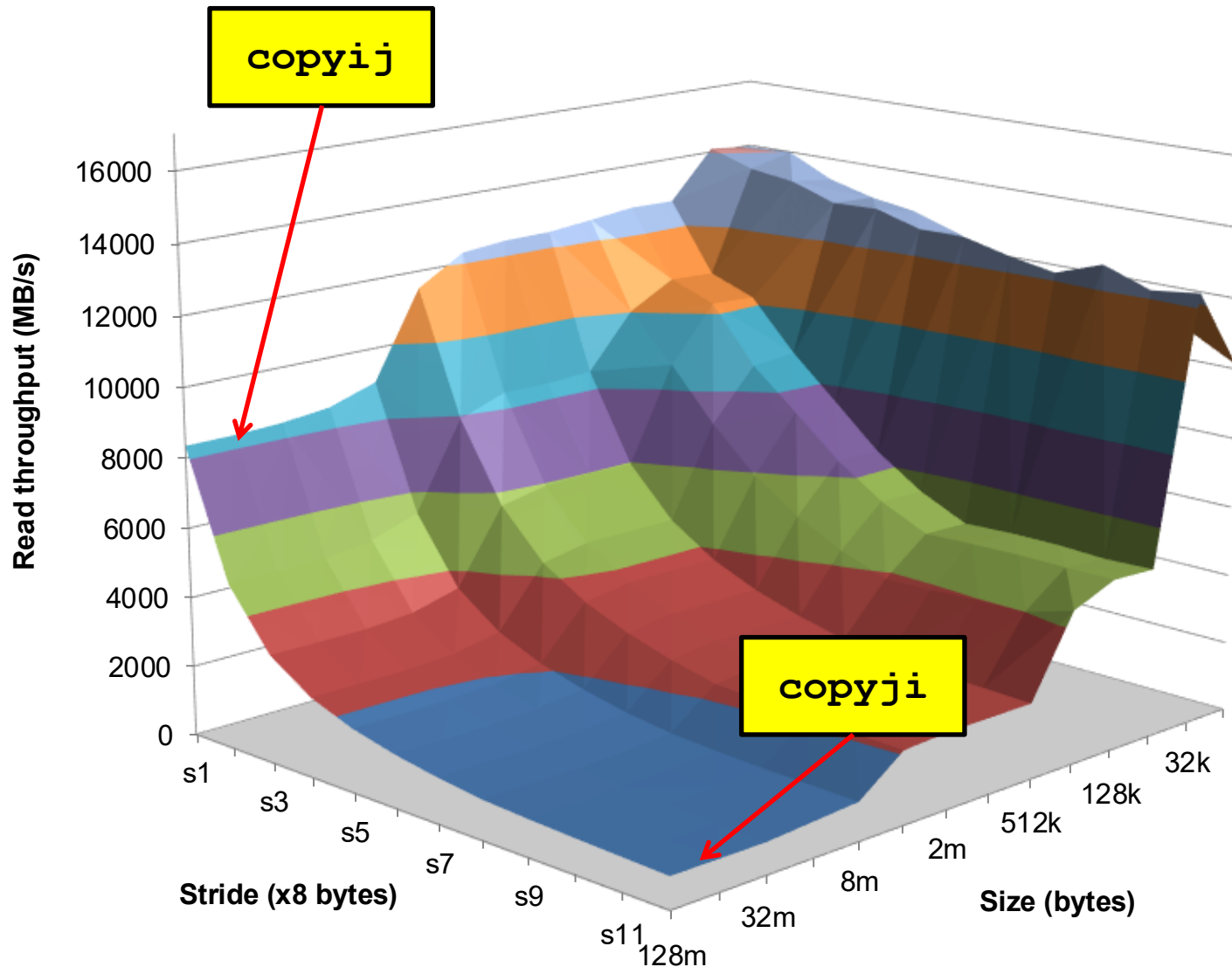
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

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.)

■ 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!**

Role within CS/ECE Curriculum

CS 122
Imperative
Programming

**213/513
/613**

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

CS Systems

- 15-319 Cloud Computing
- 15-330 Computer Security
- 15-410 Operating Systems
- 15-411 Compiler Design
- 15-415 Database Applications
- 15-418 Parallel Computing
- 15-440 Distributed Systems
- 15-441 Computer Networks
- 15-445 Database Systems

ECE Systems

- 18-349 Computer Security
- 18-349 Intro to Embedded Systems
- 18-441 Computer Networks
- 18-447 Computer Architecture
- 18-452 Wireless Networking
- 18-451 Cyberphysical Systems

CS Graphics

- 15-462 Computer Graphics
- 15-463 Comp. Photography

Academic Integrity

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

Carefully review policy:

<http://www.cs.cmu.edu/~18213/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.
- Searching the Web for solutions
- Copying code from a previous course or online solution
- Reusing your code from a previous semester (here or elsewhere)
 - Arrange meeting with instructor before reusing your old solutions

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)
 - Or, letting protections expire
- Applies to this term and the future
 - There is no statute of limitations for academic integrity violations

■ Collaborations beyond high-level, strategic advice

- Anything more than block diagram or a few words
- Code / pseudo-code is NOT high level
- Coaching, arranging blocks of allowed code is NOT high level
- Code-level debugging is NOT high level

Cheating/Plagiarism: Description

■ What is NOT cheating?

- Explaining how to use systems or tools
- Helping others with *high-level* design issues
 - High means very high
- Using code supplied by us
 - Starter code, class examples
- Using code from the CS:APP web site

■ Attribution Requirements

- Starter code: No
- Other allowed code (course, CS:APP): Yes
- Indicate source, beginning and end

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 May 2019, we gave an AIV to a student who took the course in Fall 2018 for unauthorized coaching of a Spring 2019 student. His grade was changed retroactively.

■ 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, 15213, 213, 18213, malloclab, etc.
 - 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 accessible place on the Internet, now or in the future
- Hacking the course infrastructure

■ This is OK (and encouraged):

- Googling a man page for fputs
- Asking a friend for help with gdb (but not with your code)
- Asking a TA or course instructor for help, showing them your code, ...
- Using code examples from book (with attribution)
- Talking about a (high-level) approach to the lab with a classmate

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

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

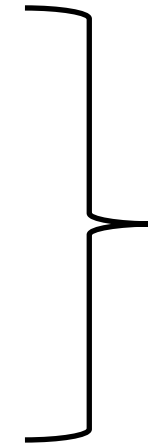
- **Start early**
- **Don't rely on marathon programming sessions**
 - Your brain works better in small bursts of activity
 - Ideas / solutions will come to mind while you're doing other things
- **Plan for stumbling blocks**
 - Assignment is harder than you expected
 - Code doesn't work
 - Bugs hard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

Logistics

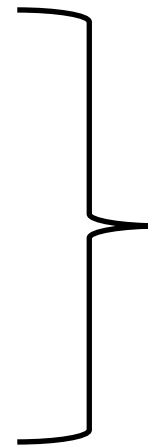
15-213, 14-513, 15-513, 18-213, 18-613

- 15-213
 - CS Undergraduates and other Undergraduates
- 14-513
 - INI Masters students
- 15-513
 - CS Masters and other Masters students

- **18-213**
 - ECE Undergraduates
 - In-class lectures in DH A302
- **18-613**
 - ECE Masters students
 - In-class lectures in HOA 160 / B23 110



15-cohort
(for TAs, office
hours, etc)



18-cohort
(for TAs, office
hours, etc)

- **Same material & labs for all the courses**
- **Only go to 18-cohort TAs and OHs**

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
- Electronic editions available (*Don't get paperback version!*)
- On reserve in Sorrells Library

■ 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
- On reserve in Sorrells Library

Course Components

■ Lectures

- Higher level concepts
- In-class quizzes could tilt you to a higher grade if borderline

■ Labs (8)

- 1-2+ weeks each
- Provide in-depth understanding of an aspect of systems
- Programming and measurement

■ Weekly Assignments (best 10 of 12)

- Reinforce concepts

■ In-class Midterm Exam & Final Exam (15% each)

- Test your understanding of concepts & mathematical principles
- Final exam will not be cumulative

■ Small student groups (weekly)

Programs and Data

■ Topics

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

■ Labs

- 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

■ Labs

- 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

■ Labs

- L5 (malloclab): 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
- Includes aspects of compilers, OS, and architecture

■ Labs

- L6 (tshlab): Writing your own Unix shell.
 - A first introduction to concurrency

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

■ Labs

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

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!

Policies: Lab

■ Work groups

- You must work **alone** on all lab assignments
- Instructors and TAs are here to help!

■ Handins

- Labs due at 11:59pm ET
- Electronic handins using **Autolab** (no exceptions!)

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**
- **Some assignments allow fewer late days (check labs web page)**

■ 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 hard to catch up
- Try to save your grace days until the last few labs

Facilities

■ Labs will use the Intel Computer Systems Cluster

- A subset of the “shark machines”, e.g.
- `linux> ssh angelshark.ics.cs.cmu.edu`

- Total set: 21 servers donated by Intel for 213/513/613
 - 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 6.1
- Rack-mounted in Gates machine room

- Login using your Andrew ID and password

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

■ Labs are provided by the CMU Autolab system

- Project page: <http://autolab.andrew.cmu.edu>
- 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.

Autolab accounts

- **Lab 0 (C Lab) does not require an account to start**
 - It is available via the schedule page of the course web site
- **Accounts will be created before datalab**
 - This is also before Lab 0 (C Lab) is due.
- **You must be enrolled to get an account**
 - Autolab is not tied into the Hub's rosters
 - Beginning this Thursday, we'll update accounts approximately daily.
- **If you join late, please chat with your group TA about catching up**

Weekly Assignments

- **12 weekly assignments starting this week**
- **Released on Tuesdays, due 9 days later at 11:59 pm ET**
 - Exception: Assignment #5/6 (released Mon 10/10, due Fri 10/14)
 - No grace days (15% penalty/day for up to 3 days)
 - Last turn-in date is 3 days after due date
- **Released and collected on Canvas**
 - Upload solutions & put final answers into a quiz
- **Drop 2 lowest out of 12**
 - Remaining 10 are each worth 2% of your grade
- **Reinforce concepts**
 - Typically, concepts from that week's lectures

Small Student Groups

- **Replaces recitation**
- **Goal: Descale the course, making it more personal and personally supportive**
 - Also taming office hours, which could get crazy at times in the past.
- **Meet Wednesdays at scheduled time**
 - Groups begin tomorrow. Your TA will contact you late today.
 - This week, and this week only, may be via Zoom (We are normally given room assignments before the 2nd week)
- **Groups of 5 students + 1 TA facilitator**
 - Meet for 1 hour each week, in-person (Mandatory and graded)
 - Maintain a group chat (Slack, GroupMe, Hangouts, WeChat, whatever)
 - Try to develop a good social bond, like 5 friends going through class
 - And a TA who really knows you and how to support you
 - TAs have time reserved for helping their group members, hopefully reducing dependency upon global office hours

Policies: Grading

- **Labs (50%): weighted according to effort**
- **Final Exam (25%)**
- **Written Assignments (20%): drop lowest 2 out of 12**
- **Small group participation (5%)**
- **Final grades based on a straight scale (90/80/70/60) with a small amount of curving**
 - Only upward

Getting Help

■ Class Web pages:

<http://www.cs.cmu.edu/~18213> for 18-213/18-613

- Complete schedule of lectures, exams, and assignments
- Copies of lectures, assignments, exams, solutions
- FAQ

■ Piazza

- Best place for questions about assignments
- We will fill the FAQ and Piazza with answers to common questions
- Be careful about public posts: Remember the AIV policy

■ Canvas

- Recorded lectures
- In-class quizzes
- Written assignments

Getting Help

■ Email

- Send email to individual instructors or TAs only to schedule appointments
 - (Kesden is the exception, and you can feel free to email or call, he is sometimes hard to reach otherwise)

■ TA Office Hours (starting today)

- Sundays - Thursdays, 6-8pm ET, Ansys A050 or via Zoom
- Sundays - Thursdays, 8-10pm ET, Zoom-Only
- CMU SV Only: Additional in-person hours will be announced for you

■ Walk-in Tutoring

- Details TBA. Will put information on class webpage.

■ 1:1 Appointments

- You can schedule 1:1 appointments with any of the teaching staff

Instructor Office Hours

- Swarun Kumar, CIC 4113,
<https://calendar.google.com/calendar/u/1?cid=Y183MzU3MjkxNTE2OTRiYWE2N2MzMWNjZDM0ZjY0OTM1NGZiOGM1YTcyNTVkNTcwOGYwYzY2MzBIMGRhM2JlZmM5QGdyb3VwLmNhbGVuZGFyLmdvb2dsZS5jb20>
- Greg Kesden, HH A205,
<https://www.cs.cmu.edu/~gkesden/schedule.html>

Waitlist questions

- ECE Academic services (ece-asc@andrew.cmu.edu)
- Please don't contact the instructors with waitlist questions.

*Welcome
and Enjoy!*