Welcome to COMP 530

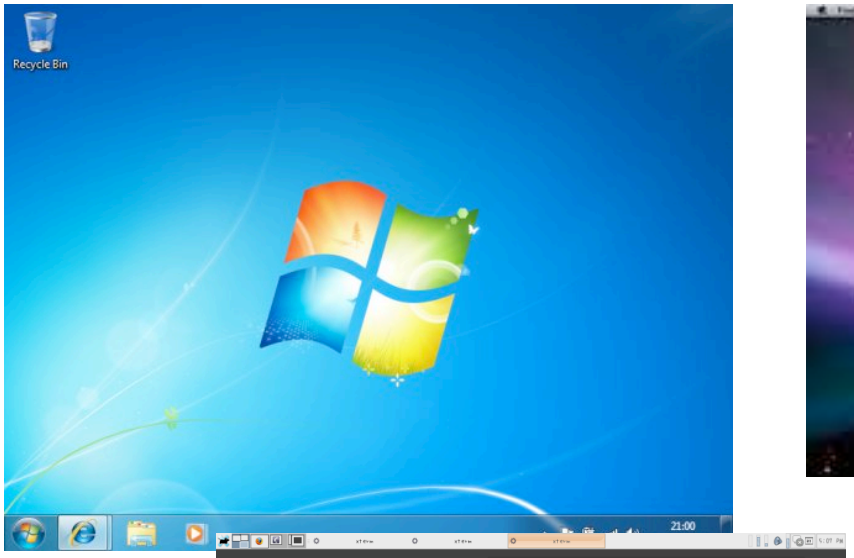
Don Porter

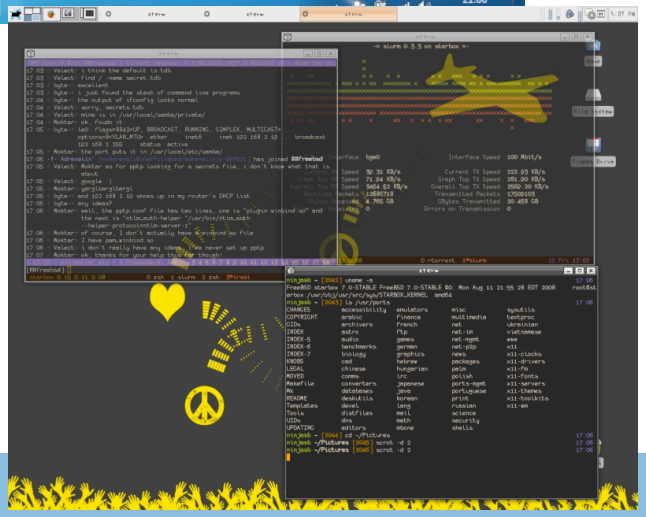
Welcome!

- I just moved here from Stony Brook University
  - I taught a comparable class at SBU regularly
- Today's goals:
  - Give you a flavor of my teaching style with a mini-lecture
  - Cover course organization
- My high-level goals for the class:
  - Demystify how computers work (No magic)
  - Learn core principles: secure multiplexing, scheduling, concurrency, performance analysis
  - **Challenging, but supportive, environment**

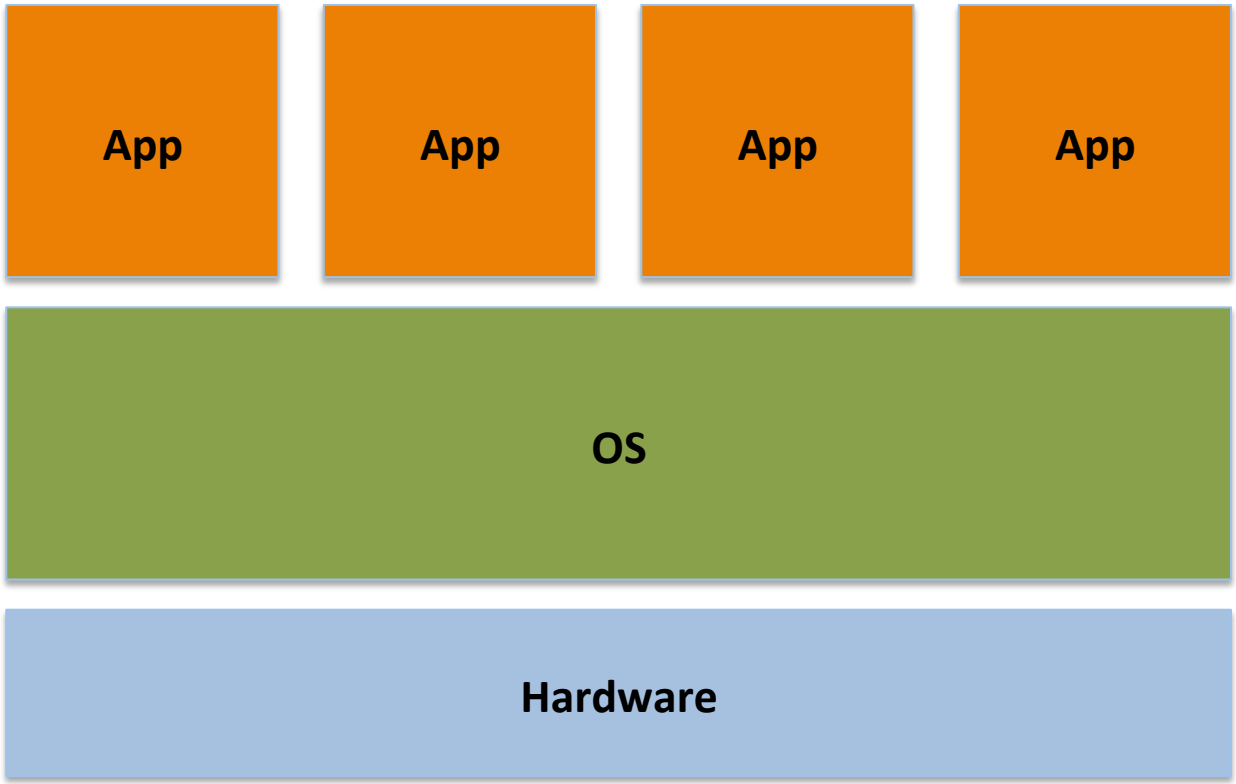
So what is an OS?

One view of an OS





Another simple view of an OS



A less happy view of an OS

So which one is right?

- They all are

An OS serves three masters

1. Give users a desktop environment
2. Give applications a more usable abstraction of the hardware
3. Give hardware manufacturers an abstraction of the applications

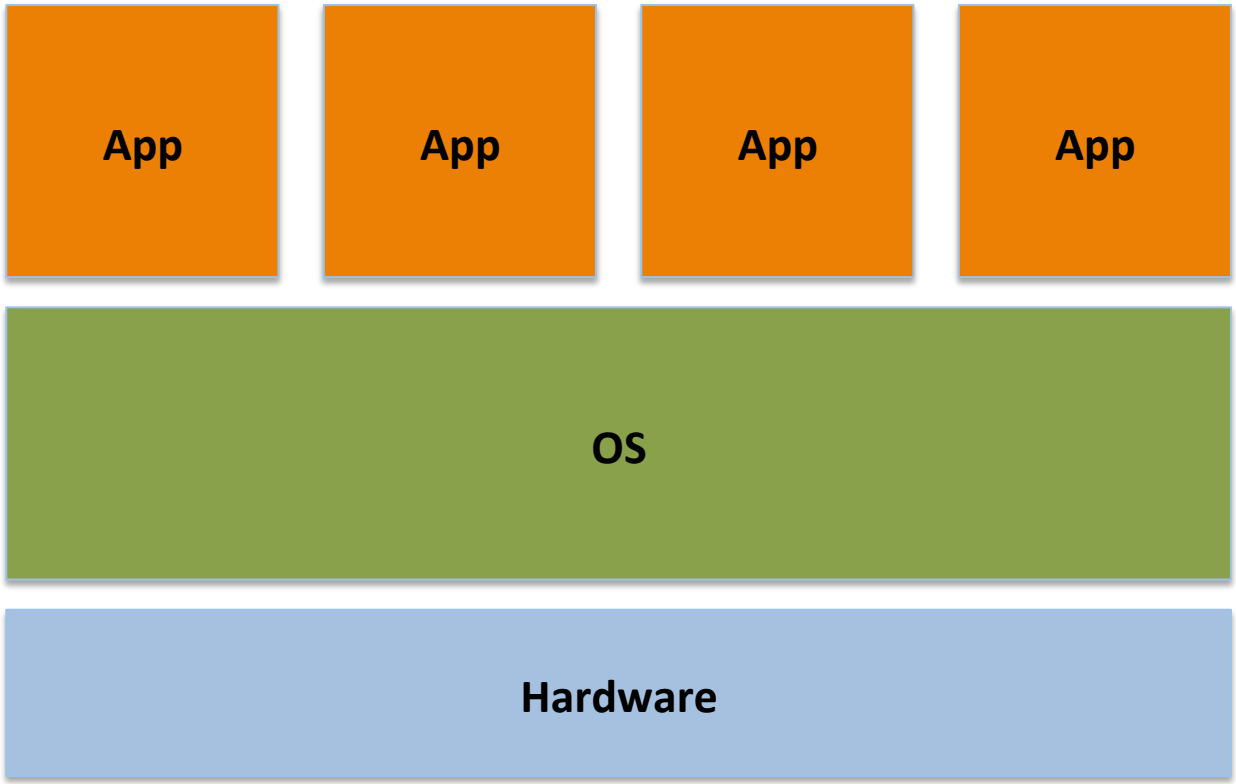
Why Study Operating Systems?

- Primary Goal: Demystify how computers work
  - Lots of abstractions and heuristics between your application and the hardware
  - A good computer scientist should understand what happens inside the system when one types a command
- Secondary: Learn how to write robust programs
  - OSes like Linux have many users and work on a wide range of hardware
  - Deal with subtle issues: concurrency, consistency, etc.

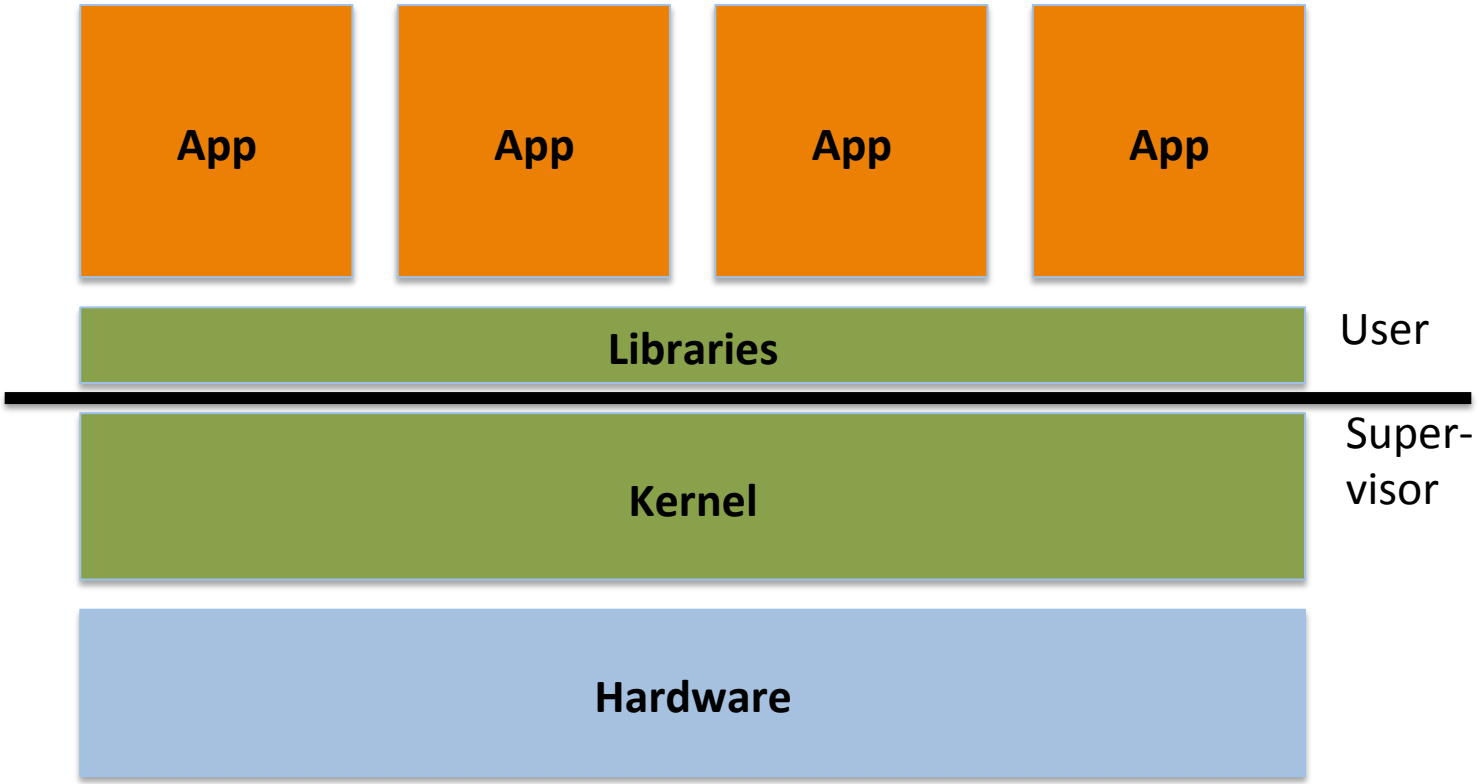
Background (1)

- CPUs have 2 modes: user and supervisor
  - Sometimes more, but whatevs
- Supervisor mode:
  - Issue commands to hardware devices
  - Power off, Reboot, Suspend
  - Launch missiles, Do awesome stuff
- User mode:
  - Run other code, hardware tattles if you try anything reserved for the supervisor

OS architecture



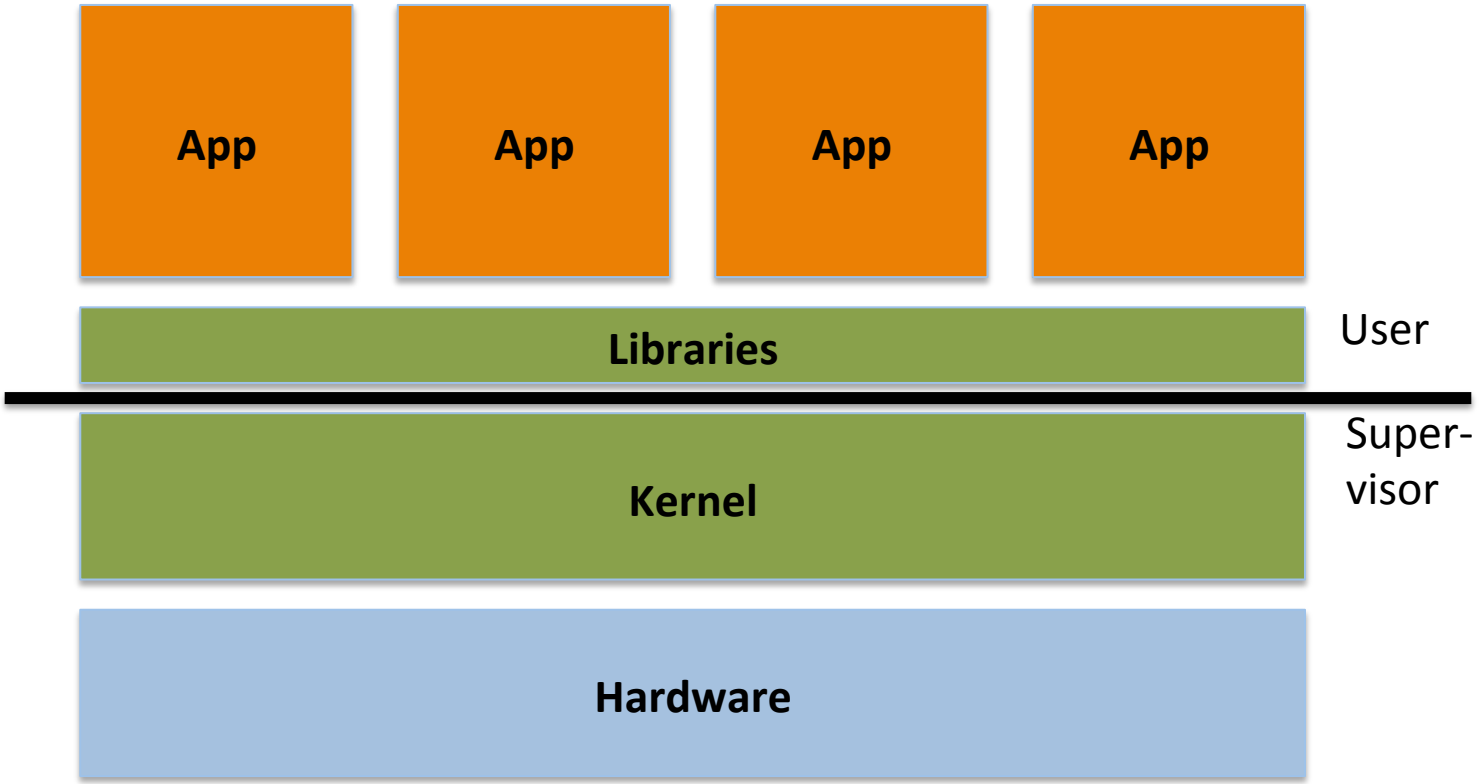
OS architecture



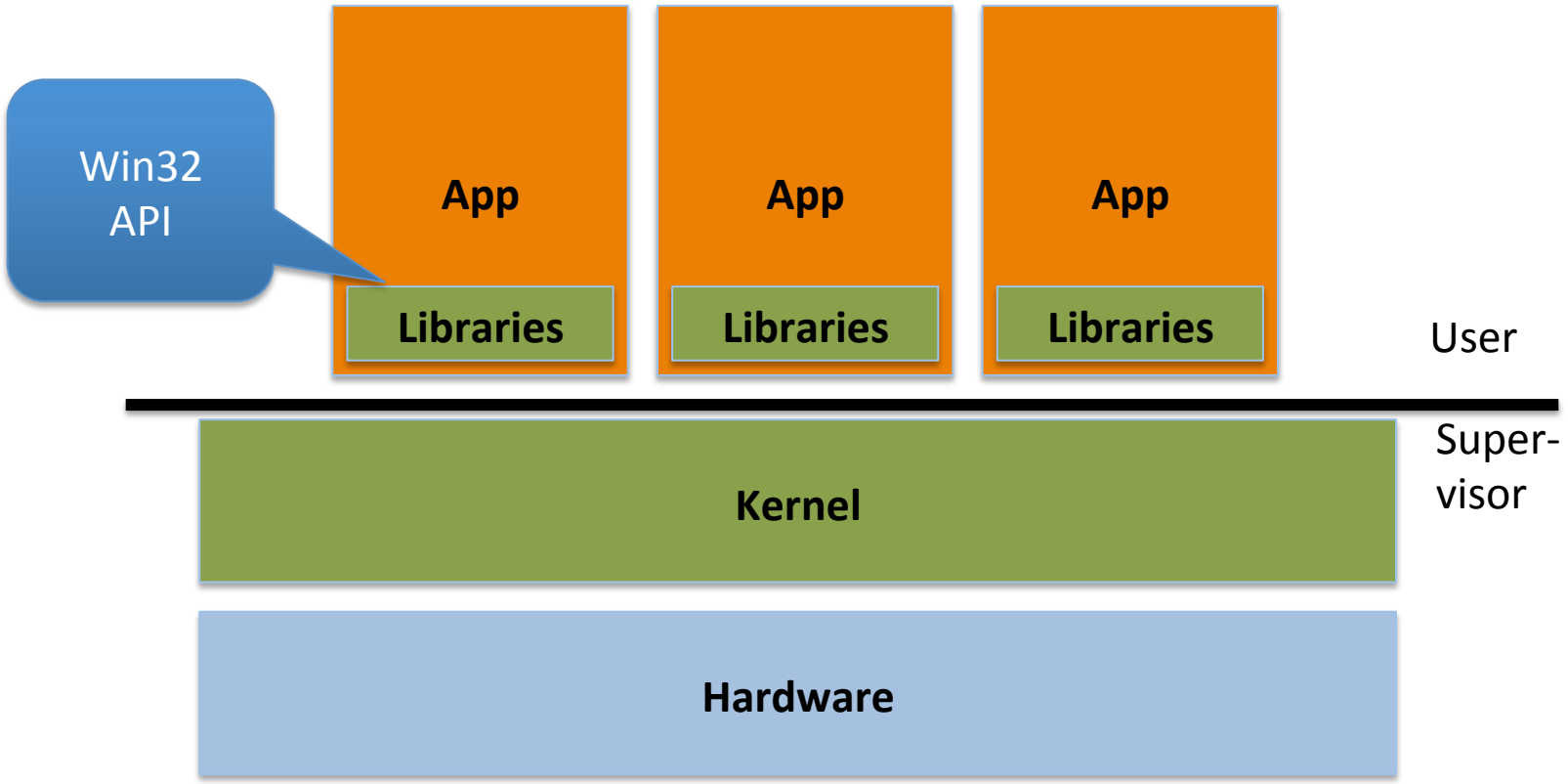
Master #2: Applications

- Application Programming Interface (API)
  - Win32 (Windows)
  - POSIX (Unix/Linux)
  - Cocoa/Cocoa Touch (Mac OS/iOS)
- Application-facing functions provided by libraries
  - Injected by the OS into each application

OS architecture



OS architecture



Famous libraries, anyone?

- Windows: ntdll.dll, kernel32.dll, user32.dll, gdi32.dll
- Linux/Unix: libc.so, ld.so, libpthread.so, libm.so

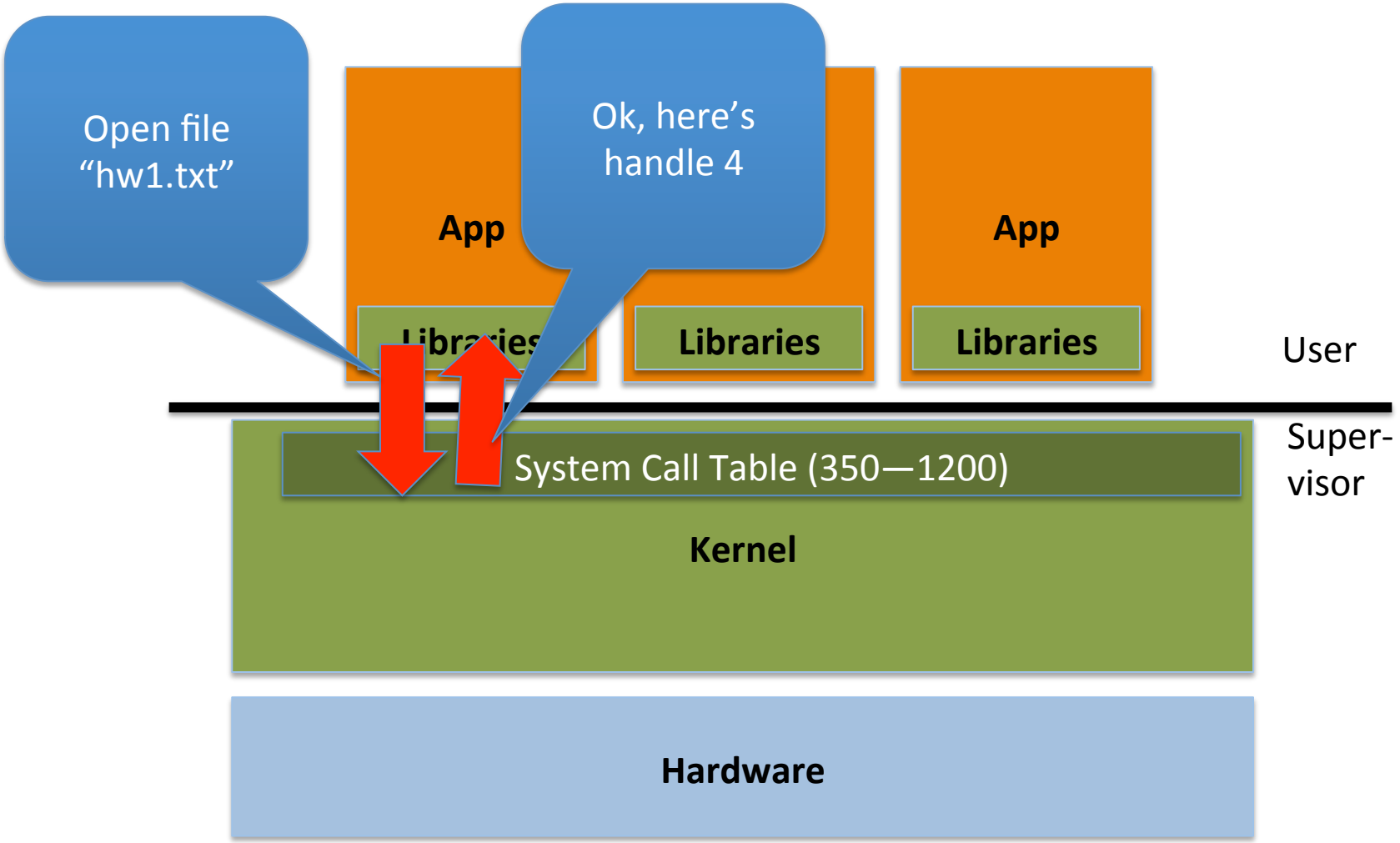
Caveat 1

- Libraries include a lot of code for common functions
  - Why bother reimplementing sqrt?
- They also give high-level abstractions of hardware
  - Files, printer, dancing Homer Simpson USB doll
- How does this work?

System Call

- Special instruction to switch from user to supervisor mode
- Transfers CPU control to the kernel
  - One of a small-ish number of well-defined functions
- How many system calls does Windows or Linux have?
  - Windows ~1200
  - Linux ~350

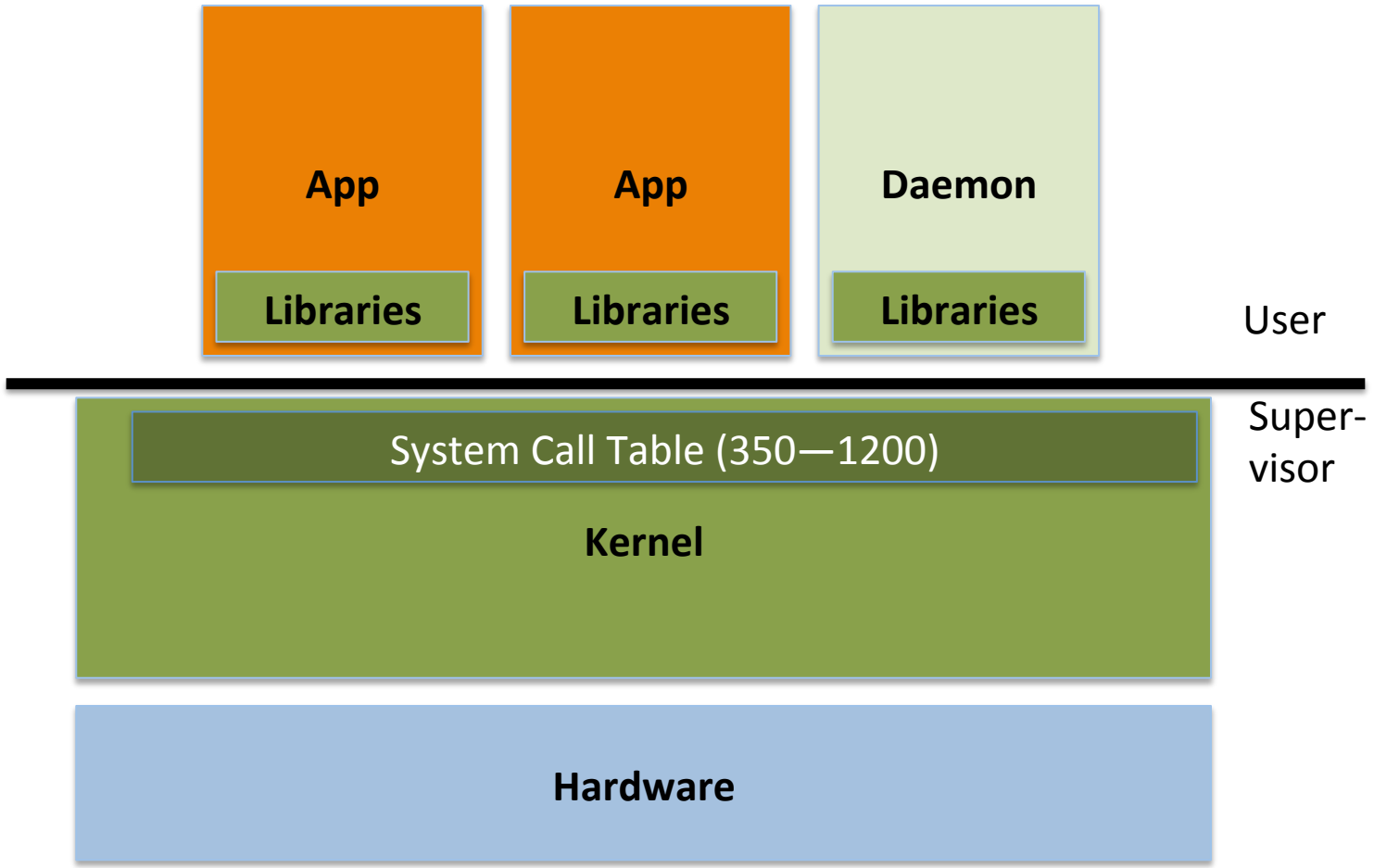
OS architecture



Caveat 2

- Some libraries also call special apps provided by the OS, called a ***daemon (or service)***
  - Communicate through kernel-provided API
- Example: Print spooler
  - App sends pdf to spooler
  - Spooler checks quotas, etc.
  - Turns pdf into printer-specific format
  - Sends reformatted document to device via OS kernel

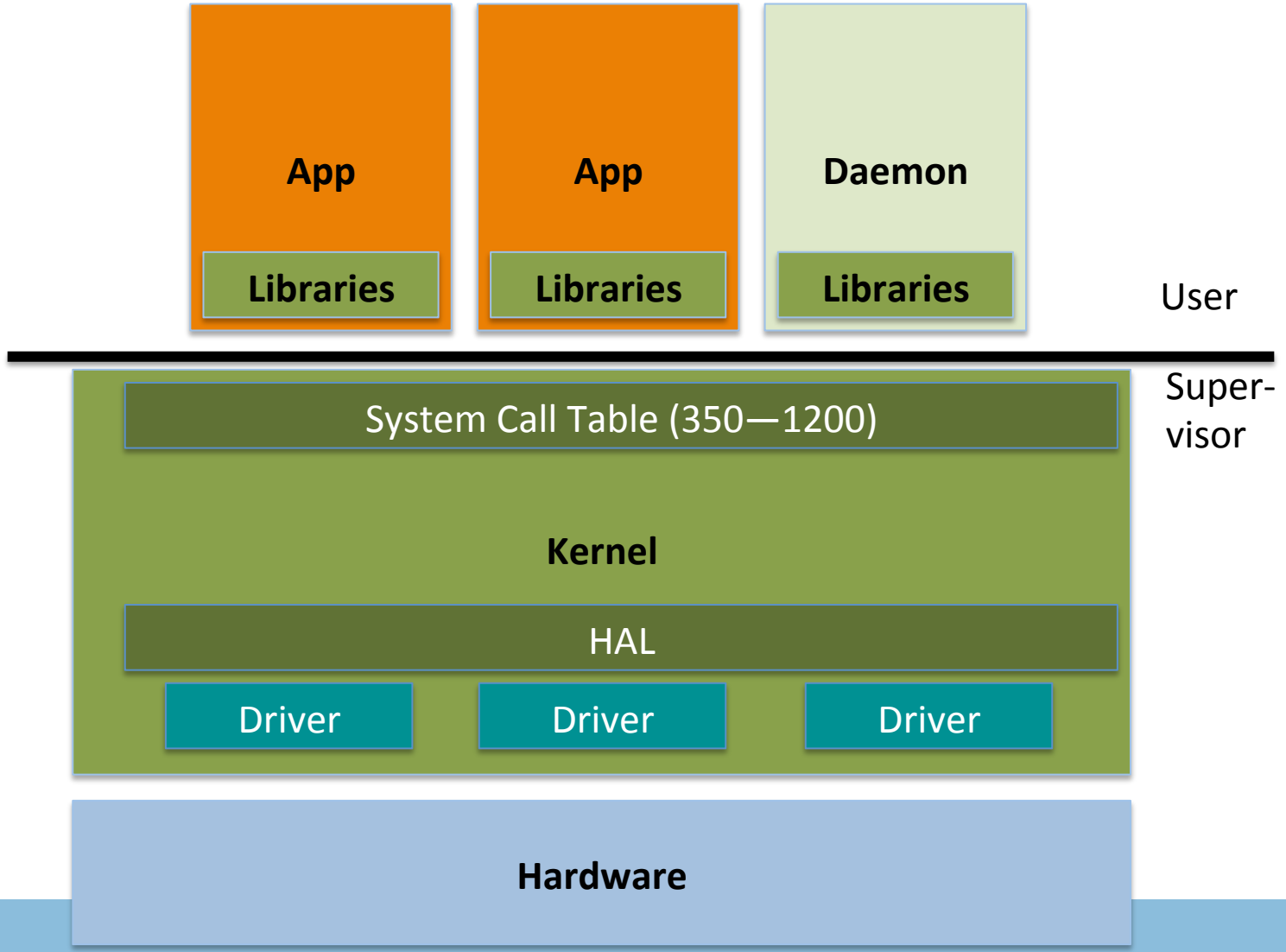
OS architecture



Master 3: Hardware

- OS kernels are programmed at a higher low level of abstraction
  - Disk blocks vs. specific types of disks
- For most types of hardware, the kernel has a "lowest common denominator" interface
  - E.g., Disks, video cards, network cards, keyboard
  - Think Java abstract class
  - Sometimes called a hardware abstraction layer (HAL)
- Each specific device (Nvidia GeForce 600) needs to implement the abstract class
  - Each implementation is called a **device driver**

OS architecture



What about Master 1

- What is the desktop?
- Really just a special daemon that interacts closely with keyboard, mouse, and display drivers
  - Launches programs when you double click, etc.
  - Some program libraries call desktop daemon to render content, etc.

An OS serves three masters

1. Give users a desktop environment
   - Desktop, or window manager, or GUI
2. Give applications a more usable abstraction of the hardware
   - Libraries (+ system calls and daemons)
3. Give hardware manufacturers an abstraction of the applications
   - Device Driver API (or HAL)

Multiplexing Resources

- Many applications may need to share the hardware
- Different strategies based on the device:
  - Time sharing: CPUs, disk arm
    - Each app gets the resource for a while and passes it on
  - Space sharing: RAM, disk space
    - Each app gets part of the resource all the time
  - Exclusive use: mouse, keyboard, video card
    - One app has exclusive use for an indefinite period

So what is Linux?

- Really just an OS kernel
  - Including lots of device drivers
- Conflated with environment consisting of:
  - Linux kernel
  - Gnu libc
  - X window manager daemon
  - CUPS printer manager
  - Etc.

So what is Ubuntu? Centos?

- A **distribution:** bundles all of that stuff together
  - Pick versions that are tested to work together
  - Usually also includes a software update system

OSX vs iOS?

- Same basic kernel (a few different compile options)
- Different window manager and libraries

What is Unix?

- A very old OS (1970s), innovative, still in use
- Innovations:
  - Kernel written in C (first one not in assembly)
    - Co-designed C language with Unix
  - Several nice API abstractions
    - Fork, pipes, everything a file
- Several implementations: *BSDs, Solaris, etc.
  - Linux is a Unix-like kernel

What is POSIX?

- A standard for Unix compatibility
- Even Windows is POSIX compliant!

Administrative

- Syllabus, schedule, homework, etc. posted on course website
- www.cs.unc.edu/~porter/courses/comp530/f16

Prerequisites

- COMP 410 – Data Structures
- COMP 411 – Computer Organization
- The background courses are necessary
- In some cases, industry experience is ok
- C programming
- Basic Unix command-line proficiency

C Programming

- You should have learned C in the prerequisite courses
  - Ok if you are not a C guru (you will be)
- A very good resource is "The C Programming Language" by Kernighan and Ritchie
  - Relatively short, and lots of useful exercises
- If you find yourself struggling with C, consider adding some work from this book to be able to complete this course on schedule

Labs: Learn by doing

- This course is **coding intensive**
  - You should know C, or be prepared to remediate quickly
  - You must learn on your own/with lab partner
- You will write several user-level utilities that exercise OS functionality
  - Challenging work, but a very marketable skill

Productive Frustration

- One of the "meta skills" that distinguishes an excellent programmer is the ability to get un-stuck
  - Fixing a "heisenbug" has this property
- How do you learn this skill?
  - Get stuck on a hard, but solvable problem
  - Learn which strategies will get you moving again
- If you take a quick cheat, you won't learn the skills to solve truly hard problems

Academic Integrity

- I take cheating very seriously. It can end your career.
- In a gray area, it is your job to stay on right side of line
- Never show your code to anyone except your partner and course staff
- Never look at anyone else's code (incl. other universities)
- Do not discuss code; do not debug each other's code
- Acknowledge students that give you good ideas

Why do we care?

- Analogy: This is the programming dojo
  - If you don't do your exercises, you will be unprepared for battle
  - You've wasted your money and both of our time
  - It brings dishonor on the dojo when you lose every battle
- Similarly, a lot of what I have to teach (and what will make you a valuable employee when you graduate) has no short cut
  - How do you learn to punch through a board?
  - You punch a board over and over until your fist goes through it

Integrity Homework

- Exercises applying course policies and ethics to several situations
- Due in class Wed 9/7, or by email

Lateness

- Each student gets 72 late hours for programming hw
  - List how many you use in slack.txt
  - Each day after these are gone costs a full letter grade on the assignment
  - If you work in a team, each member loses 1 hour for each hour late
- It is your responsibility to use these to manage:
  - Holidays, weddings, research deadlines, conference travel, Buffy marathons, release of the next Zelda game, etc.
- 3 Exceptions: illness (need doctor's note), death in immediate family, accommodation for disability

Lab Teams

- Can work alone, but better with help
  - Especially in a large class
  - No need to be a hero
- Choose your own partners
  - Piazza a list good for finding them
- Ok to change between assignments

Challenge Problems

- Each lab may include challenge problems, which you may complete for bonus points (generally 5—10 points out of 100)
  - Unwise to turn in a lab late to do challenge problems
  - Can complete challenge problems at any point in the semester---even on old labs
- Indicate any challenge problems completed in challenge.txt file

Lectures

- Discuss and supplement reading material
- An important chance to clarify issues
  - Questions are encouraged!
- I expect you to arrive prepared to answer and ask questions about the reading material
- Everything in lectures may appear on the exams, even if not in the book

- I need you here: Digressions are common to fill in "gaps" and to integrate material from other classes
  - And as I get familiar with UNC students

Recordings

- I usually record lectures for students to review later
  - NB: This room not equipped for capture, so there will be some trial and error on my laptop
  - I will share on youtube with cs.unc.edu emails; if you don't have one, please send me your youtube account
- Recordings are **best effort**
  - Recordings may fail, be unwatchable, or get deleted by accident
  - Or be discontinued if too many students stop attending
    - I need your facial expressions and questions to know if lectures make sense
- Do not use this as a substitute for class attendance

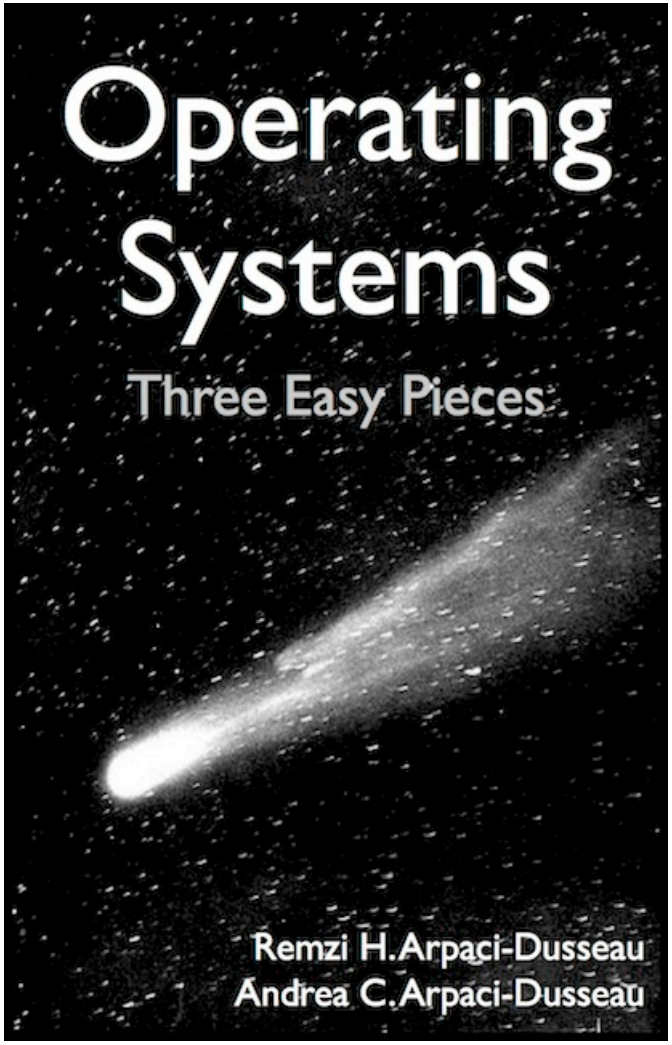
Textbook

- Free online at:

http://pages.cs.wisc.edu/~remzi/OSTEP/

- You can buy a hard-copy or ebook format online if you want

- Other optional references, definitely not required



Readings

- My lectures aren't perfect; some concepts are subtle
  - Reading other words can be helpful for reinforcement and clarification
- You will learn more in class if you read before class
  - Can't ask the textbook questions
- 1—2 papers will be posted and discussed; these you should definitely read before class

Course email list

- We will use Piazza this semester. Link on course website
- Will help scale up to a large class
- This is the primary announcement medium
- And for discussions about course work
  - Do not post code here or other solutions
  - Goal: Everyone can learn from general questions
- Material discussed on the mailing list can be an exam question

Worksheets

- You will get worksheets throughout the semester
  - And randomly assigned teams
- These will not be graded, except for participation
- But are valuable practice for the exams

- Do not save these until right before the exam
  - A lot of work
  - The material is cumulative

Other administrative notes

- Read syllabus completely
- Subscribe to the class piazza forum
- 3 exams cover: lectures, labs, mailing list
- All staff email goes to comp530ta-f16@cs.unc.edu
  - Except private issues for instructor only

Special Offer!

- You can write your own exam questions
  - Send them to me in advance of the test, if I like them, I will use them
  - Do NOT share with anyone else

Getting help

- TA's will keep office hours (TBD)
- Instructor keeps office hours
  - Note that "by appointment" means more time available on demand

Questions?

- Remember:
  - Do academic honesty homework
  - Lab 0 out