CSCI 330: Operating Systems

Sara Sprenkle

Today's Plan

- Introduce and motivate course themes
- Course organization and logistics

Discussion

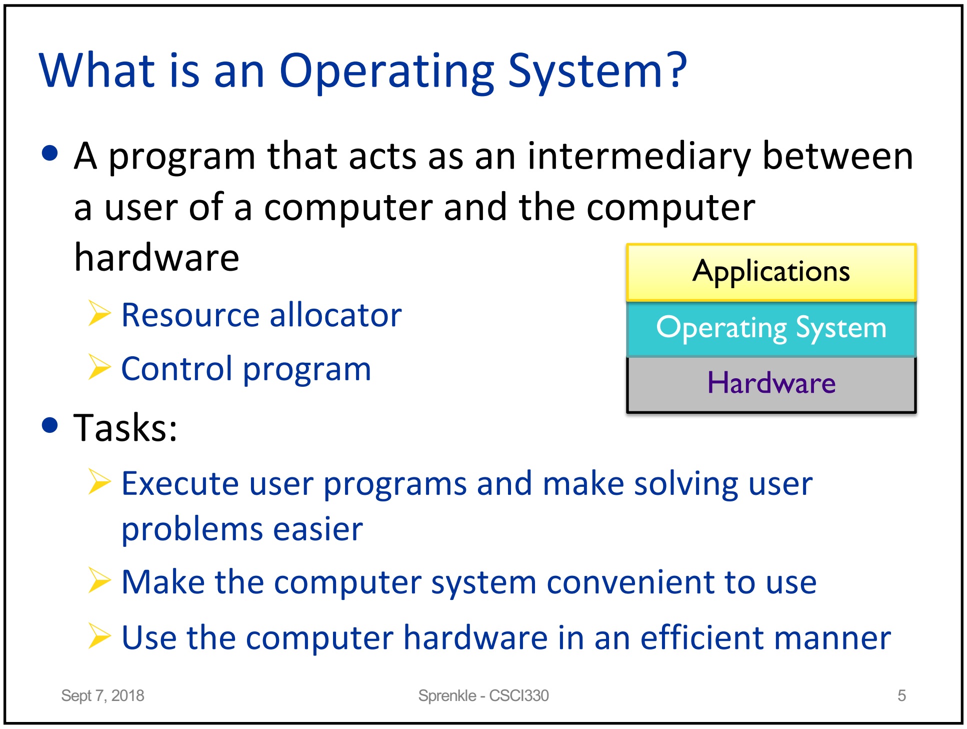
- What is an Operating System (OS)?
  - Why is defining them difficult?
  - From whose perspective should we answer this question?
- What are examples of OSs?
- Why do we need them?

OS Goals

- Make computers easier to use
  - Abstraction!
  - Bridge gap between hardware and user experience
- Use computer hardware efficiently

What is a "computer"?

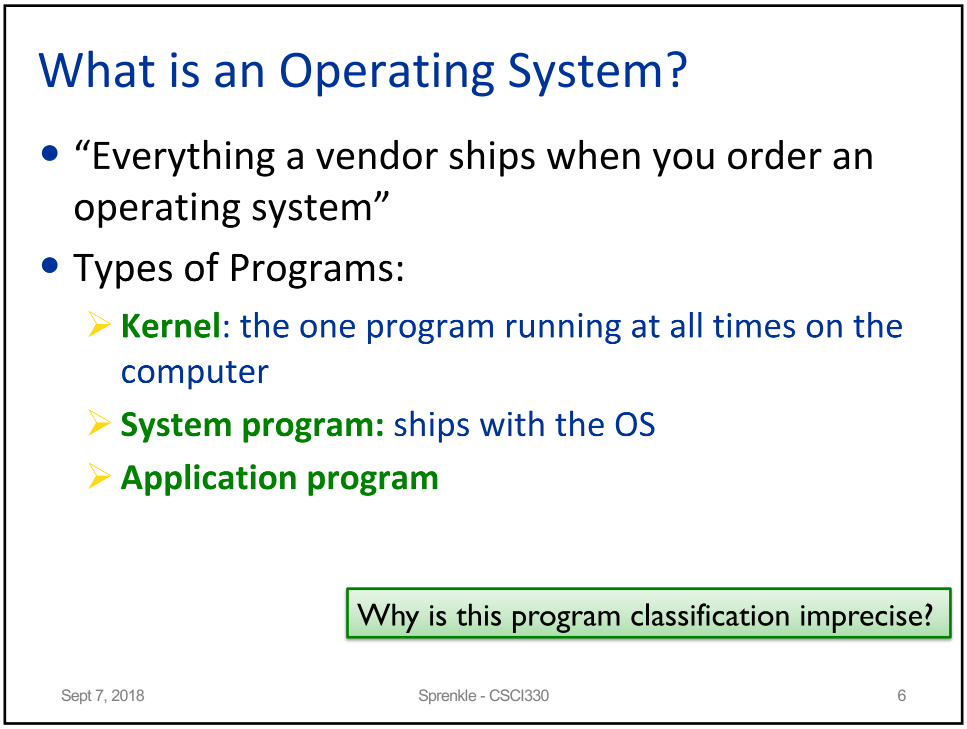
Why are these two separate goals?



What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
  - Resource allocator
  - Control program
- Tasks:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

Applications
Operating System
Hardware



What is an Operating System?

- “Everything a vendor ships when you order an operating system”
- Types of Programs:
  - **Kernel**: the one program running at all times on the computer
  - **System program:** ships with the OS
  - **Application program**

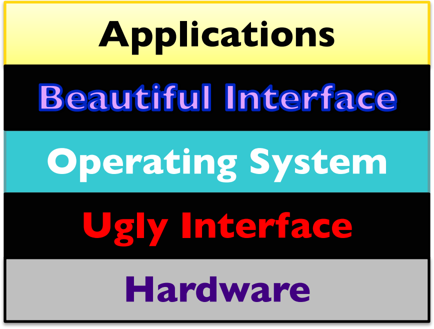
Why is this program classification imprecise?

What is an Operating System?

- “Everything a vendor ships when you order an operating system”
  - US v. Microsoft, 1998
- Types of Programs:
  - **Kernel**: the one program running at all times on the computer
  - **System program:** ships with the OS
  - **Application program**

What is an Operating System?

- An interface:



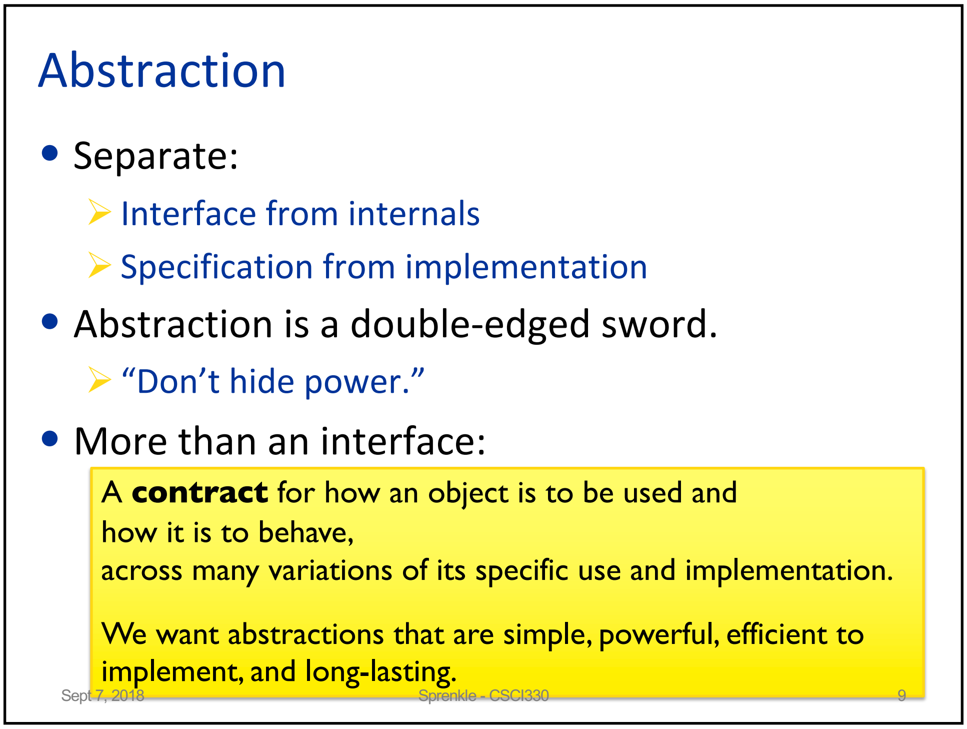
Applications

Beautiful Interface

Operating System

Ugly Interface

Hardware

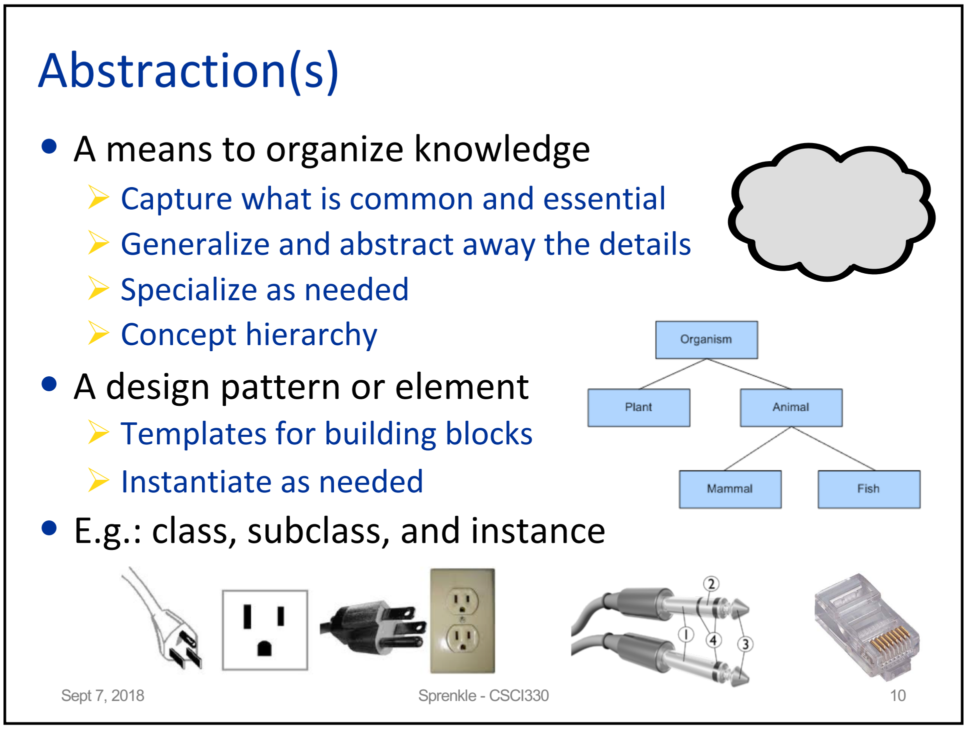


Abstraction

- Separate:
  - Interface from internals
  - Specification from implementation
- Abstraction is a double-edged sword.
  - "Don't hide power."
- More than an interface:

A **contract** for how an object is to be used and
how it is to behave,
across many variations of its specific use and implementation.

We want abstractions that are simple, powerful, efficient to implement, and long-lasting.

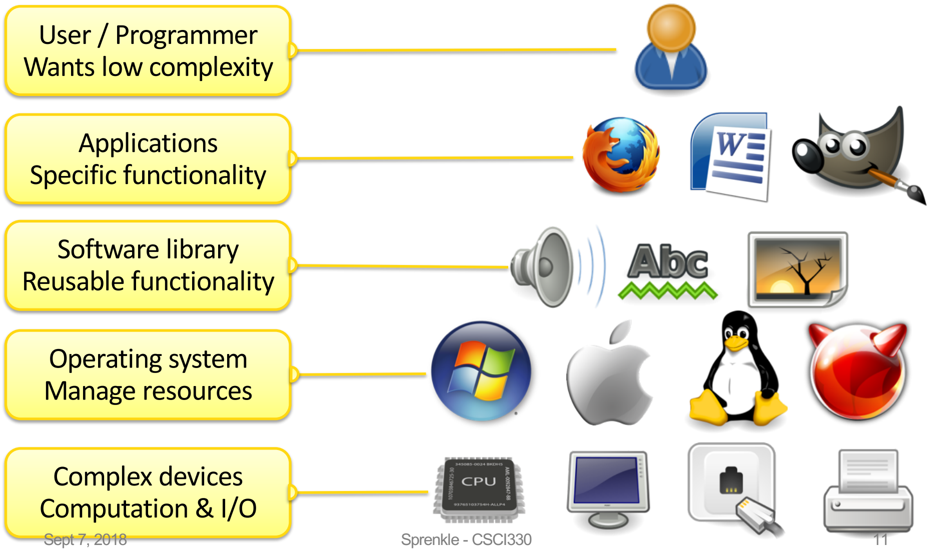


Abstraction(s)

- A means to organize knowledge
  - Capture what is common and essential
  - Generalize and abstract away the details
  - Specialize as needed
  - Concept hierarchy
- A design pattern or element
  - Templates for building blocks
  - Instantiate as needed
- E.g.: class, subclass, and instance

Organism
Plant
Animal
Mammal
Fish
2
1
4
3

Levels of Abstraction



User / Programmer
Wants low complexity

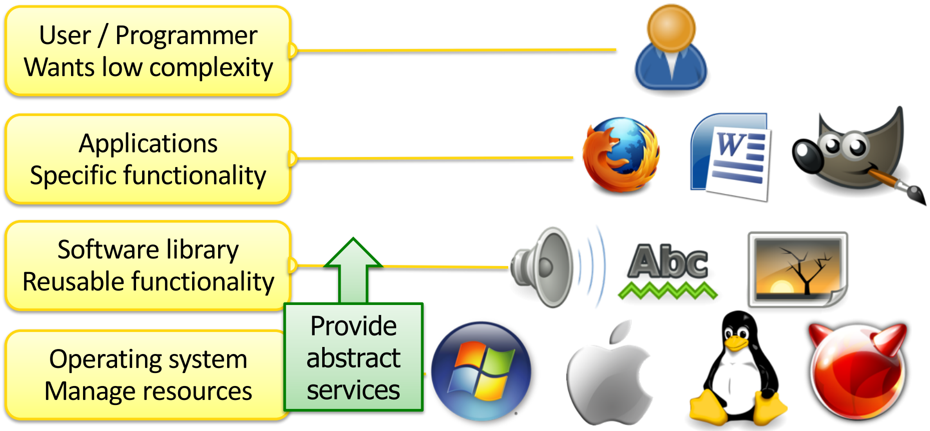
Applications
Specific functionality

Software library
Reusable functionality

Operating system
Manage resources

Complex devices
Computation & I/O

Levels of Abstraction



User / Programmer
Wants low complexity

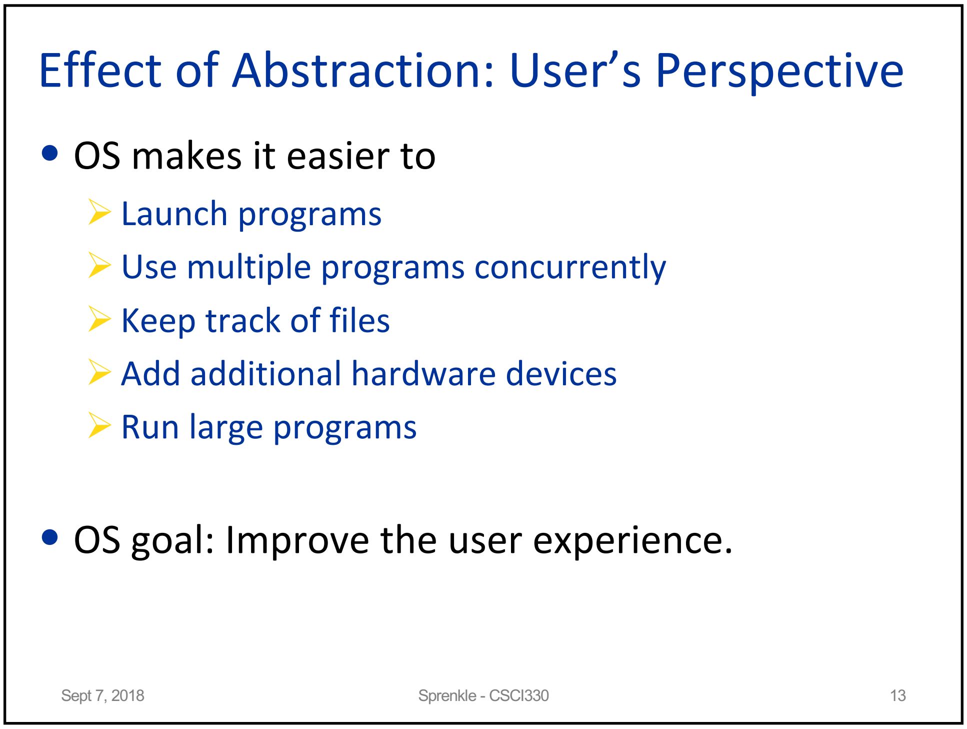
Applications
Specific functionality

Software library
Reusable functionality

Operating system
Manage resources

Provide
abstract
services

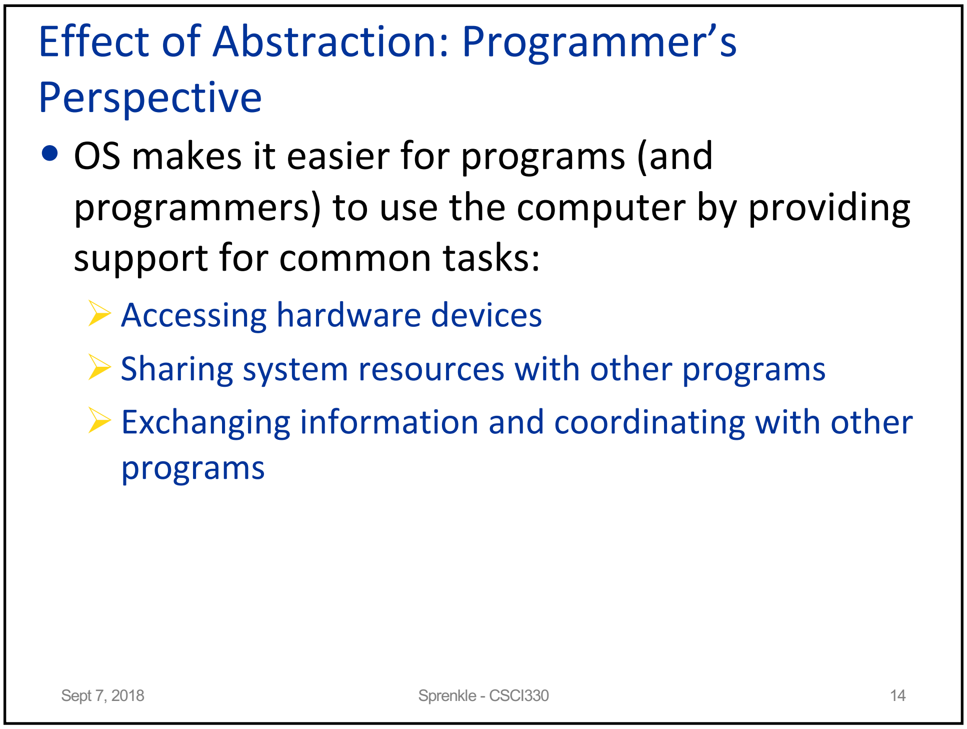
Hide the details of complex hardware
from the software above.



Effect of Abstraction: User's Perspective

- OS makes it easier to
  - ➢ Launch programs
  - ➢ Use multiple programs concurrently
  - ➢ Keep track of files
  - ➢ Add additional hardware devices
  - ➢ Run large programs

- OS goal: Improve the user experience.



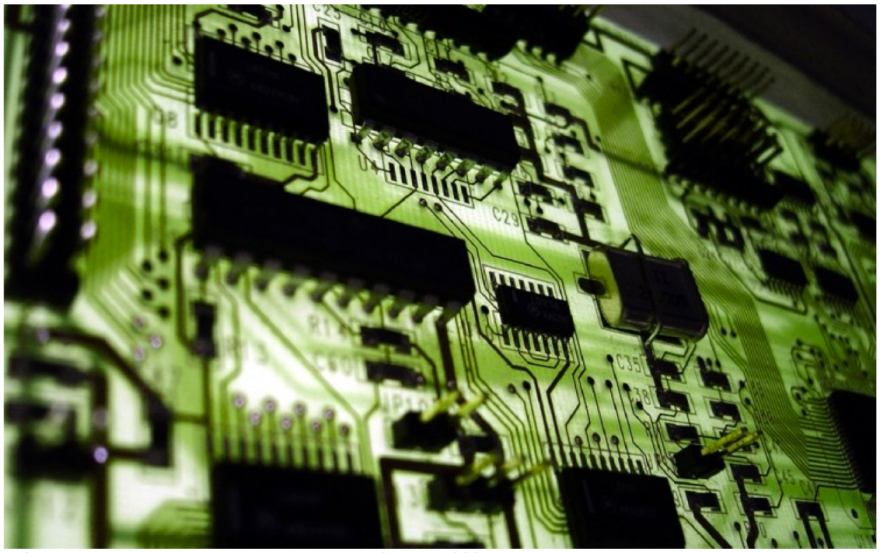
Effect of Abstraction: Programmer's Perspective

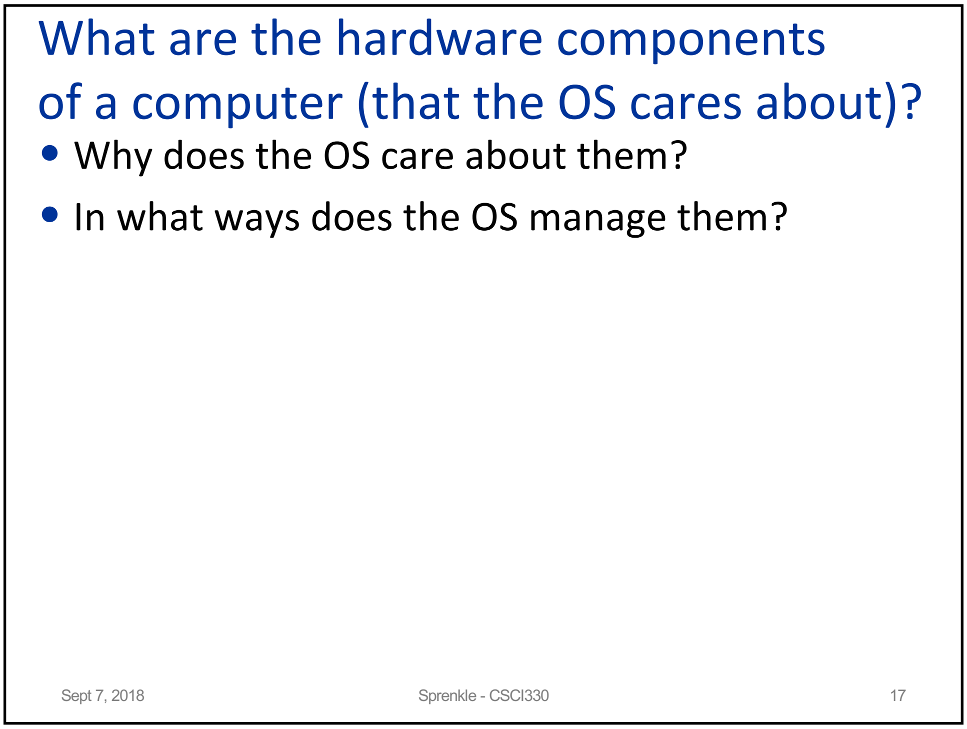
- OS makes it easier for programs (and programmers) to use the computer by providing support for common tasks:
  - ➢ Accessing hardware devices
  - ➢ Sharing system resources with other programs
  - ➢ Exchanging information and coordinating with other programs

What is an Operating System?

- **Formally**: A program that acts as an intermediary between the computer user and the computer hardware
- **Goals**:
  - Make the computer system easy to use.
  - Use the computer hardware efficiently.
- It is an extended machine
  - Hides the messy details that must be performed
  - Presents user with a virtual machine, easier to use
- It is a resource manager
  - Each program gets time with the resource
  - Each program gets space on the resource

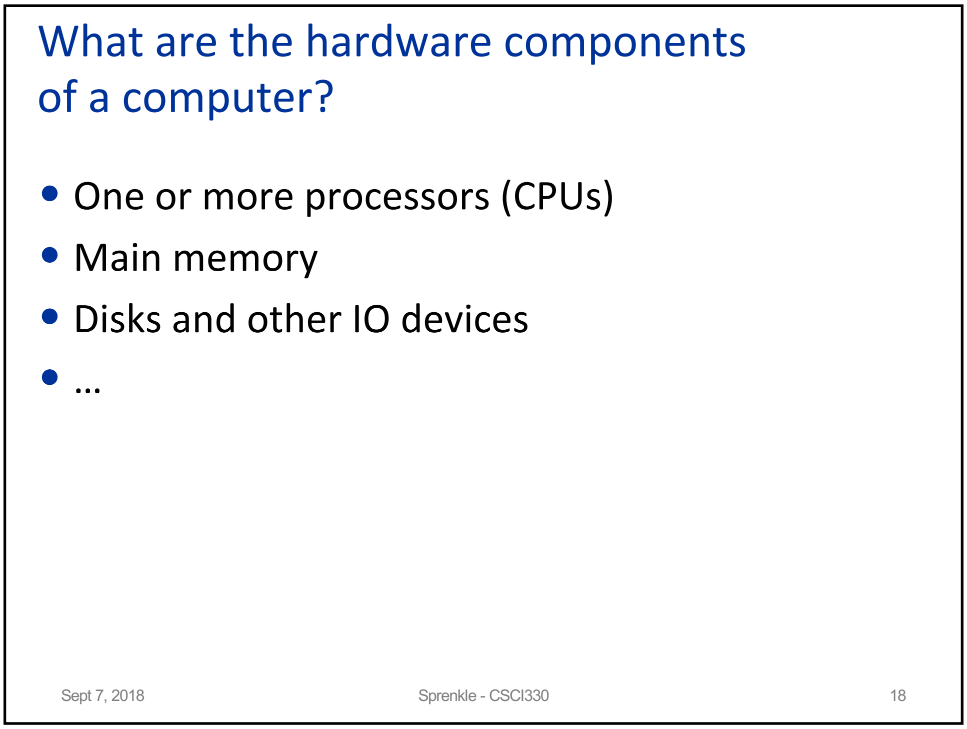
What are the hardware components of a computer (that the OS cares about)?





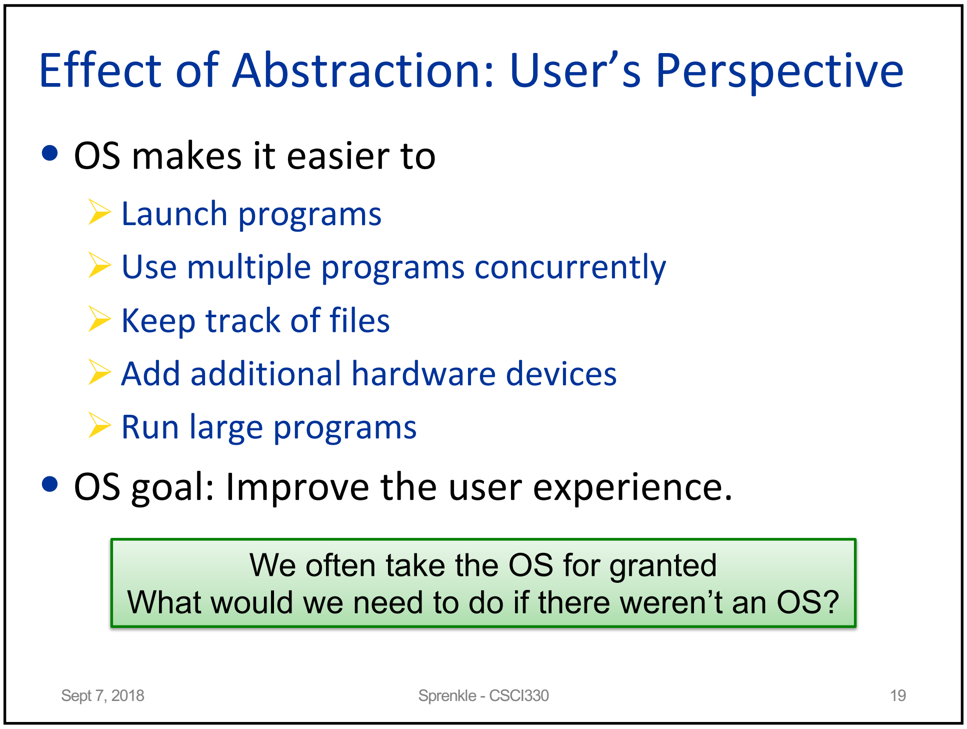
What are the hardware components of a computer (that the OS cares about)?

- Why does the OS care about them?
- In what ways does the OS manage them?



What are the hardware components of a computer?

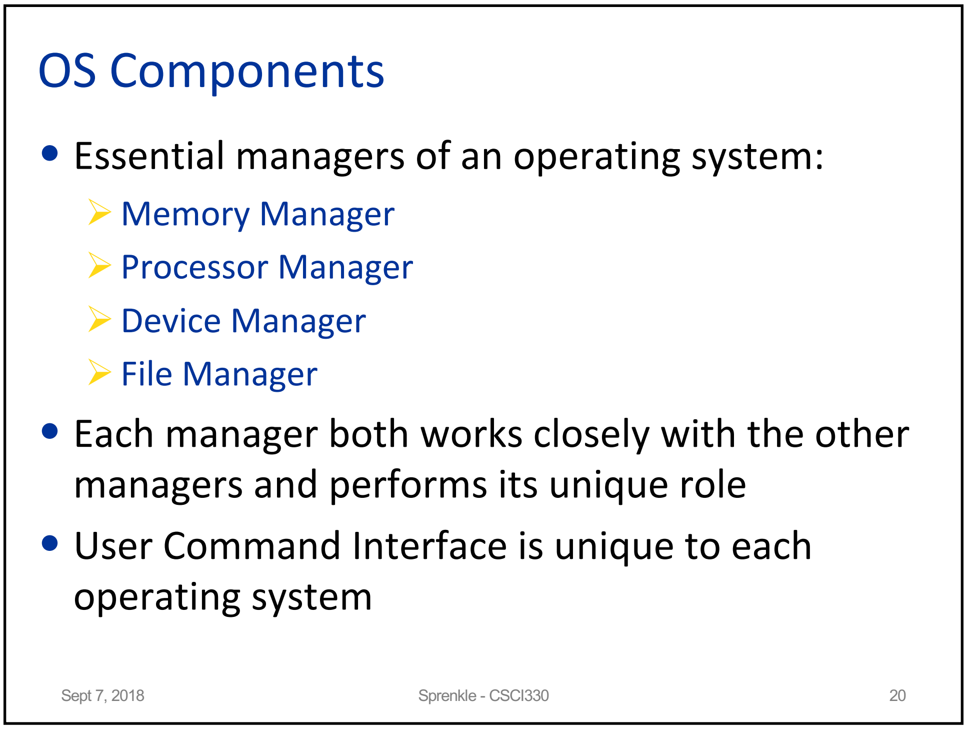
- One or more processors (CPUs)
- Main memory
- Disks and other IO devices
- …



Effect of Abstraction: User’s Perspective

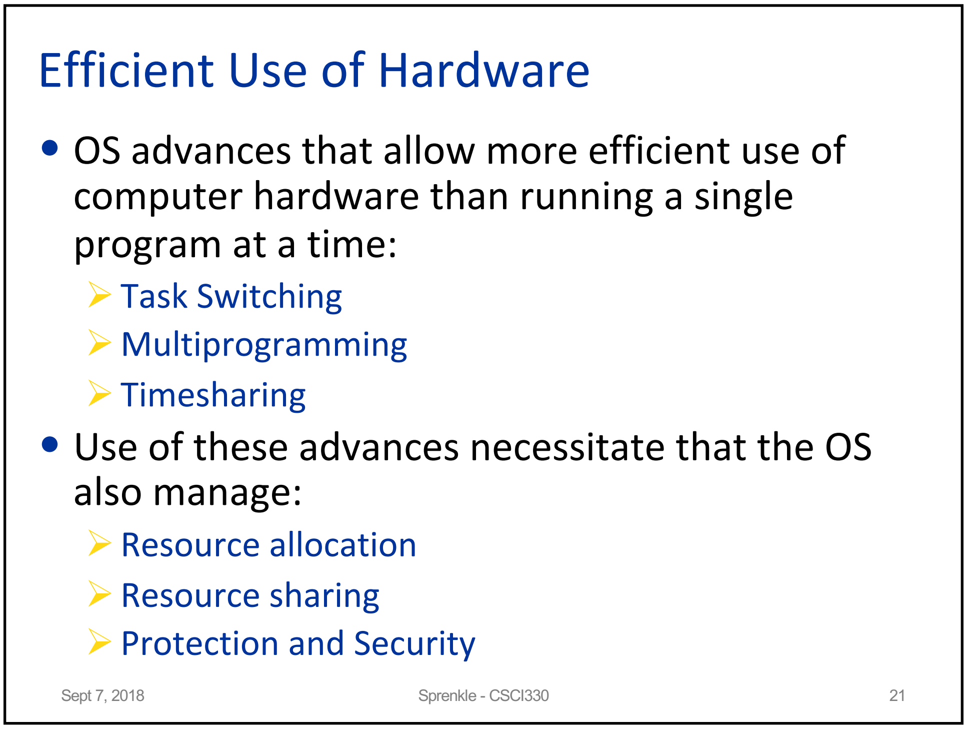
- OS makes it easier to
  - Launch programs
  - Use multiple programs concurrently
  - Keep track of files
  - Add additional hardware devices
  - Run large programs
- OS goal: Improve the user experience.

We often take the OS for granted
What would we need to do if there weren’t an OS?



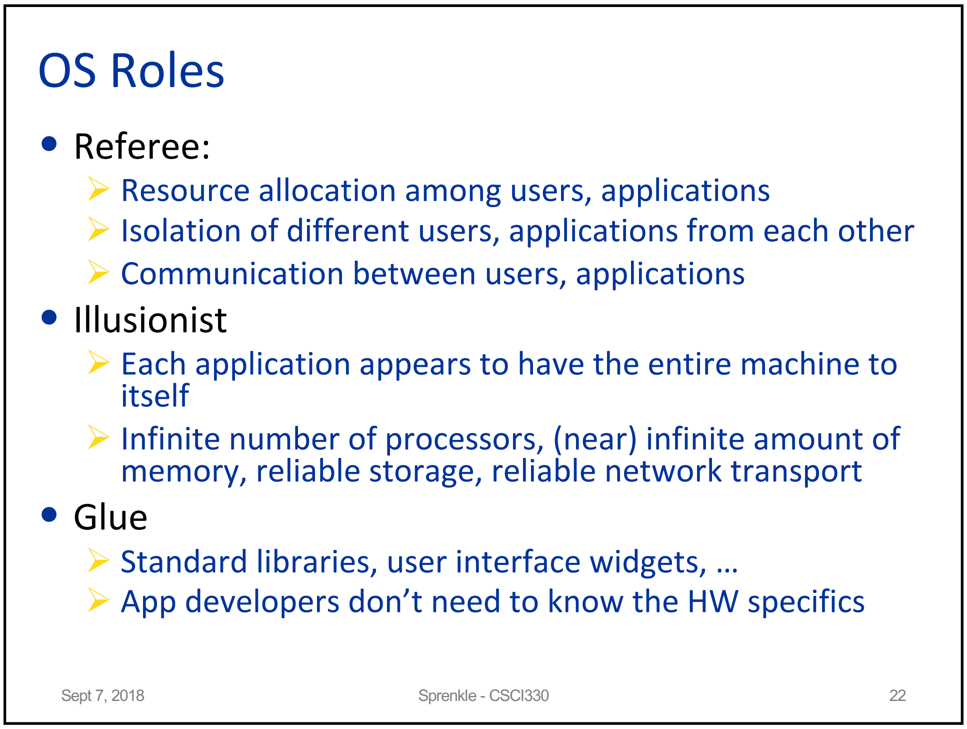
OS Components

- Essential managers of an operating system:
  - Memory Manager
  - Processor Manager
  - Device Manager
  - File Manager
- Each manager both works closely with the other managers and performs its unique role
- User Command Interface is unique to each operating system



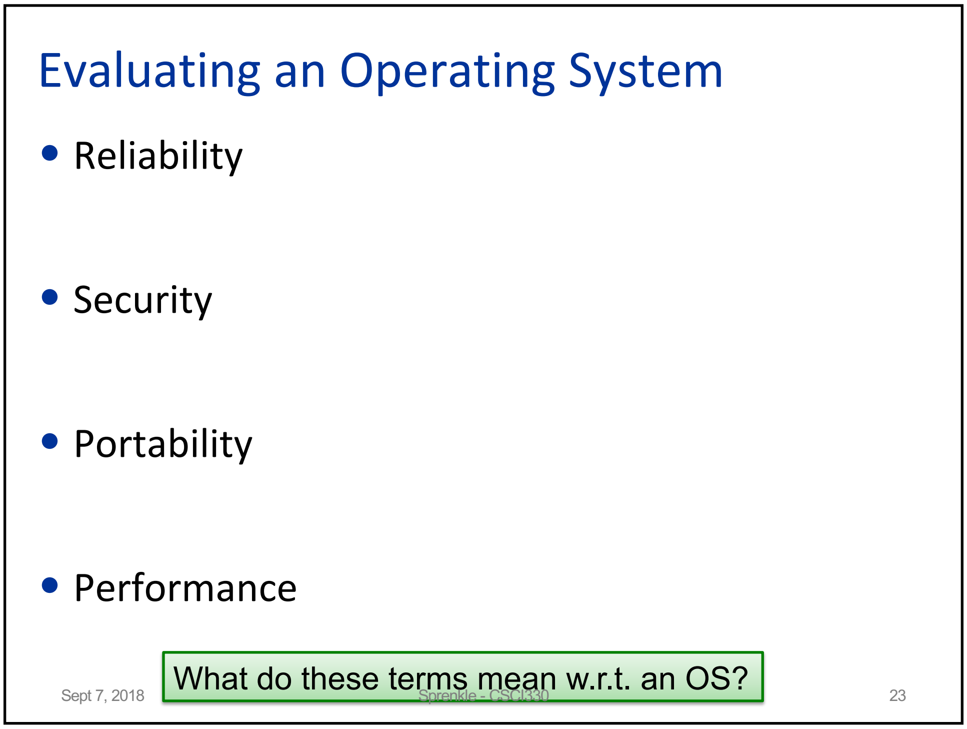
Efficient Use of Hardware

- OS advances that allow more efficient use of computer hardware than running a single program at a time:
  - Task Switching
  - Multiprogramming
  - Timesharing
- Use of these advances necessitate that the OS also manage:
  - Resource allocation
  - Resource sharing
  - Protection and Security



OS Roles

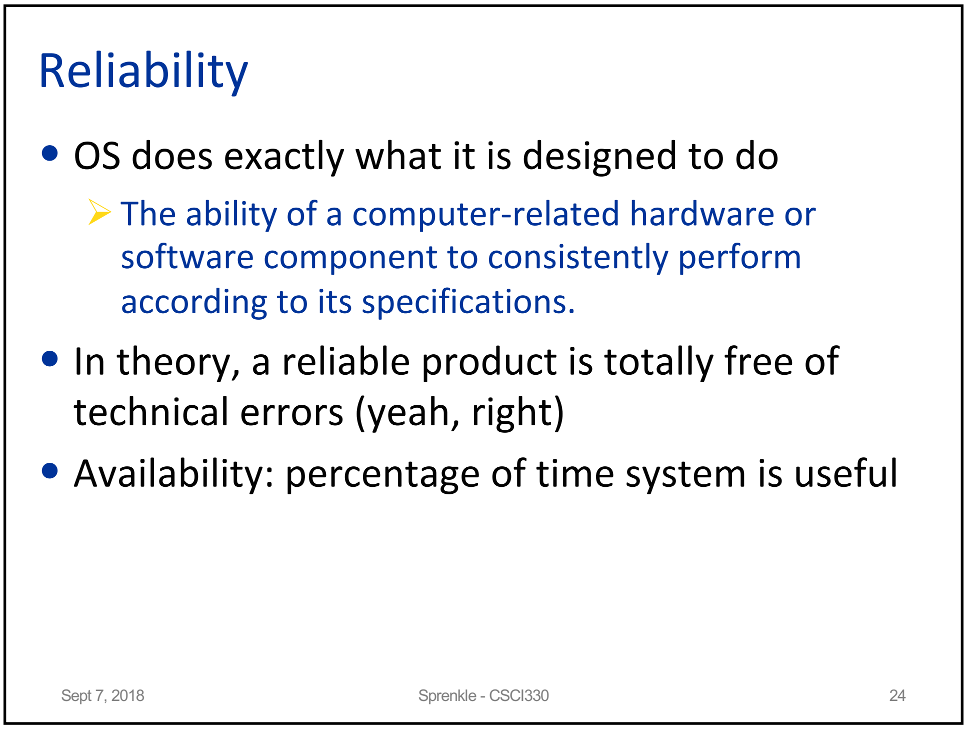
- Referee:
  - Resource allocation among users, applications
  - Isolation of different users, applications from each other
  - Communication between users, applications
- Illusionist
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport
- Glue
  - Standard libraries, user interface widgets, …
  - App developers don’t need to know the HW specifics



Evaluating an Operating System

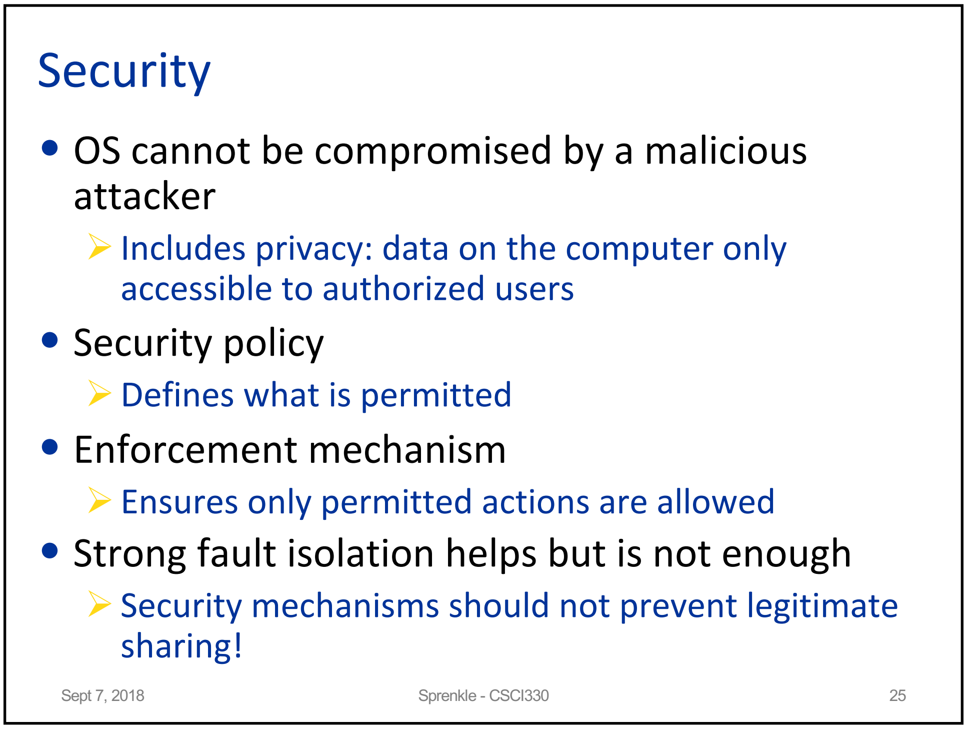
- Reliability
- Security
- Portability
- Performance

What do these terms mean w.r.t. an OS?



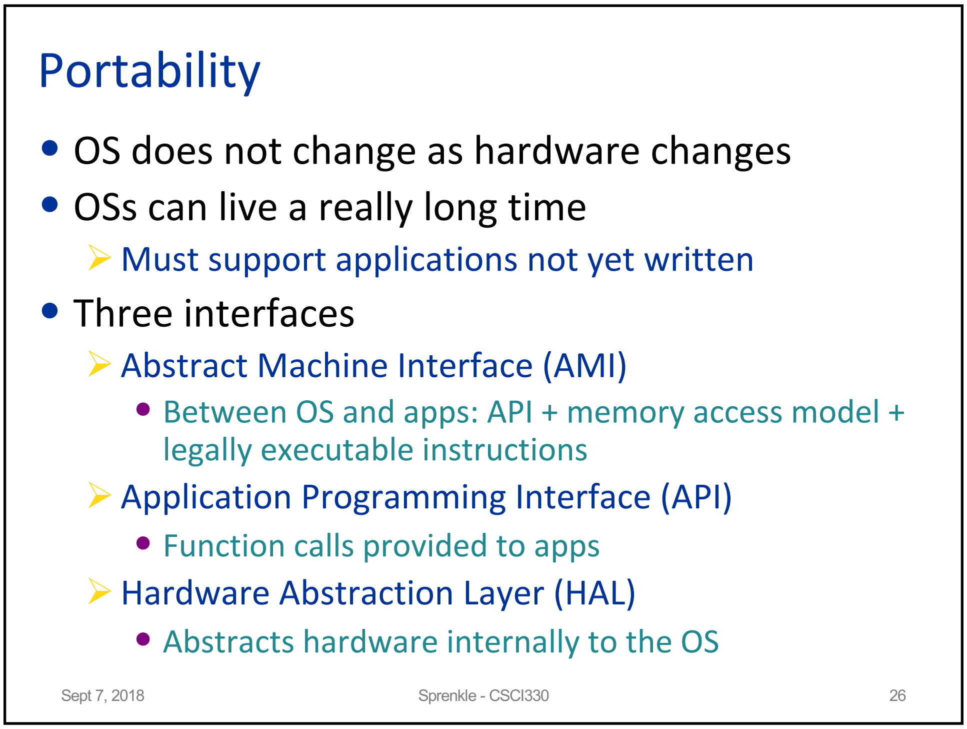
Reliability

- OS does exactly what it is designed to do
  - The ability of a computer-related hardware or software component to consistently perform according to its specifications.
- In theory, a reliable product is totally free of technical errors (yeah, right)
- Availability: percentage of time system is useful



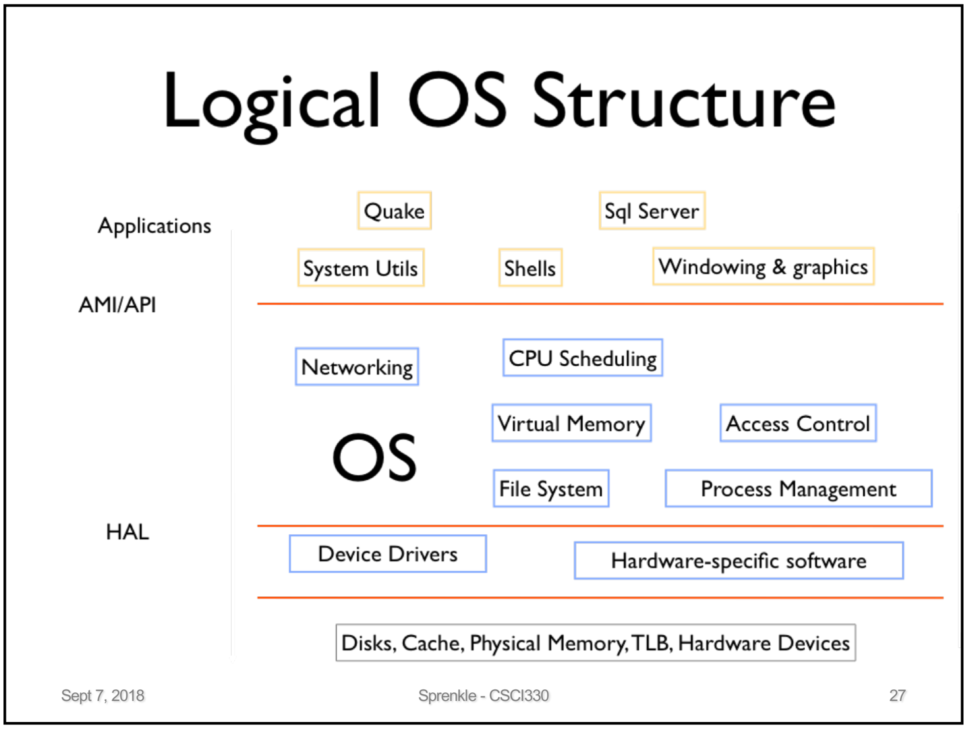
Security

- OS cannot be compromised by a malicious attacker
  - Includes privacy: data on the computer only accessible to authorized users
- Security policy
  - Defines what is permitted
- Enforcement mechanism
  - Ensures only permitted actions are allowed
- Strong fault isolation helps but is not enough
  - Security mechanisms should not prevent legitimate sharing!

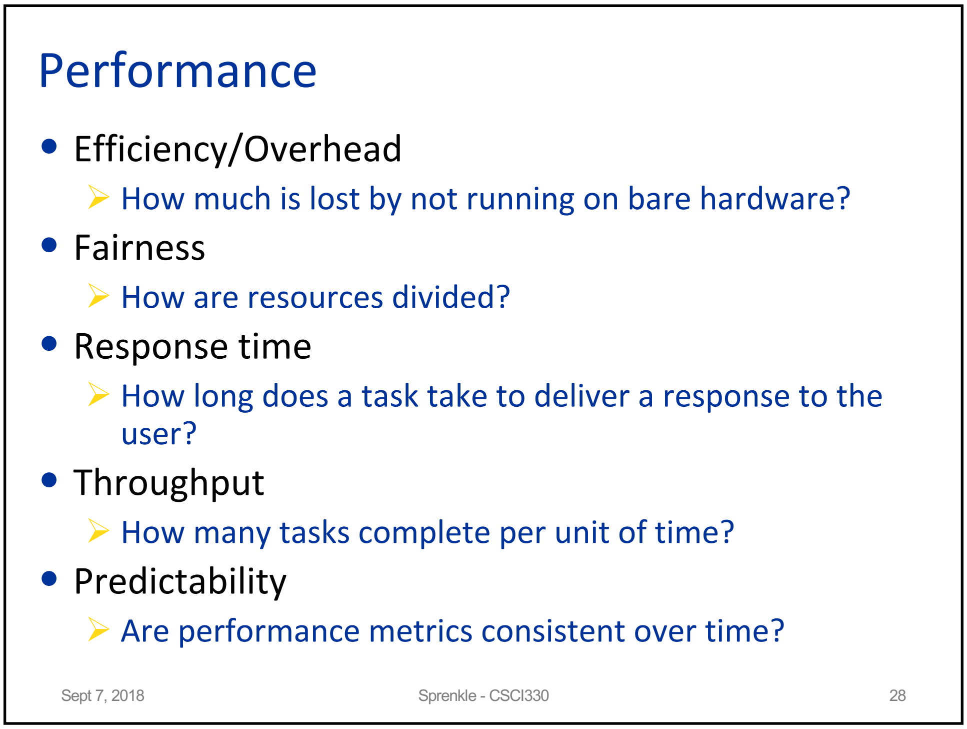


Portability

- OS does not change as hardware changes
- OSs can live a really long time
  - Must support applications not yet written
- Three interfaces
  - Abstract Machine Interface (AMI)
    - Between OS and apps: API + memory access model + legally executable instructions
  - Application Programming Interface (API)
    - Function calls provided to apps
  - Hardware Abstraction Layer (HAL)
    - Abstracts hardware internally to the OS



Logical OS Structure



Performance

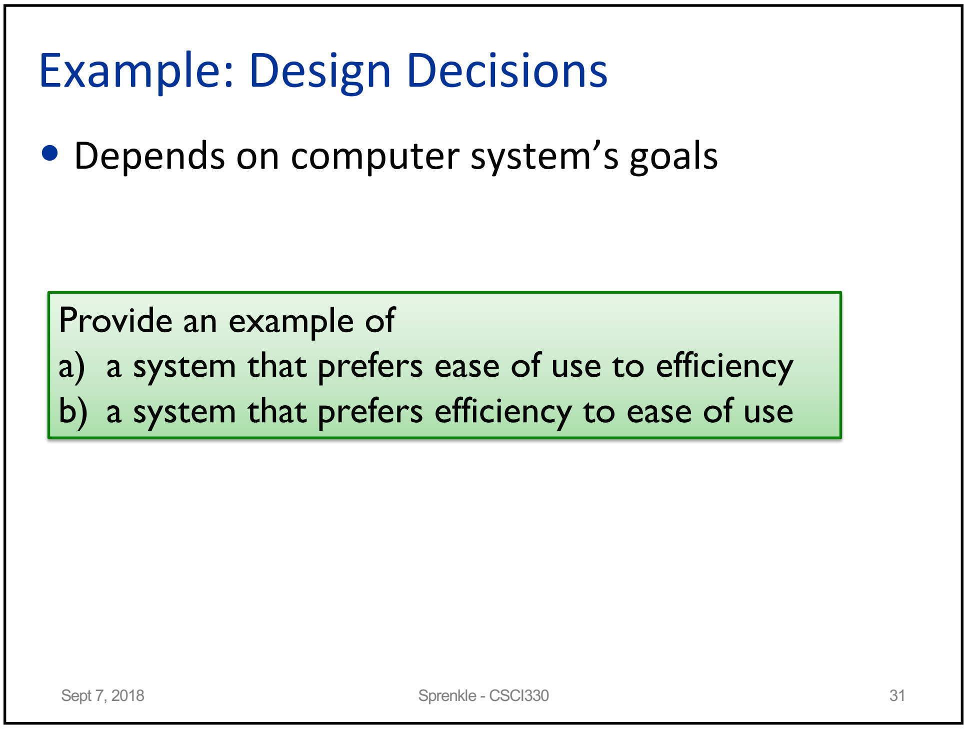
- Efficiency/Overhead
  - How much is lost by not running on bare hardware?
- Fairness
  - How are resources divided?
- Response time
  - How long does a task take to deliver a response to the user?
- Throughput
  - How many tasks complete per unit of time?
- Predictability
  - Are performance metrics consistent over time?

Why Should We Study OS?

- Are we likely to write our own OS?

Why Study Operating Systems?

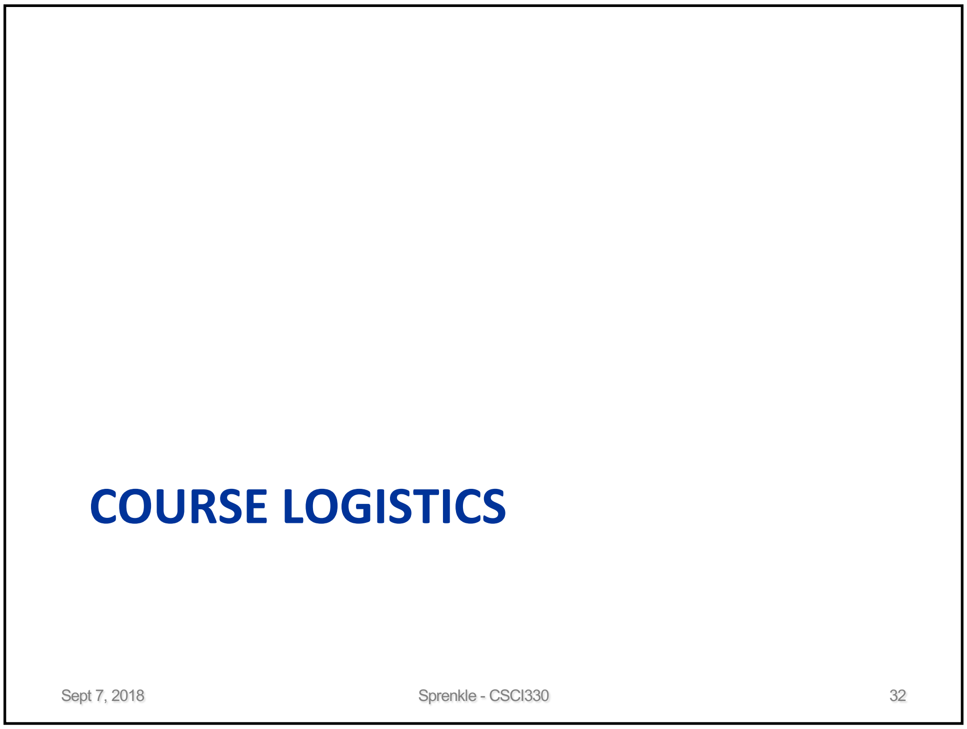
- Understanding the OS helps you write better code
  - Learn how to manage complexity through appropriate abstractions
- Understand a wide range of system designs and tradeoffs of those designs
  - Performance vs. simplicity, HW vs. SW, etc
    - What should be in the hardware? In the OS? In the user applications?
      - What are the tradeoffs of these decisions?
  - Design trade-offs made in the past do not necessarily apply now
  - Those made now will not necessarily apply in the future
- Operating Systems are everywhere!



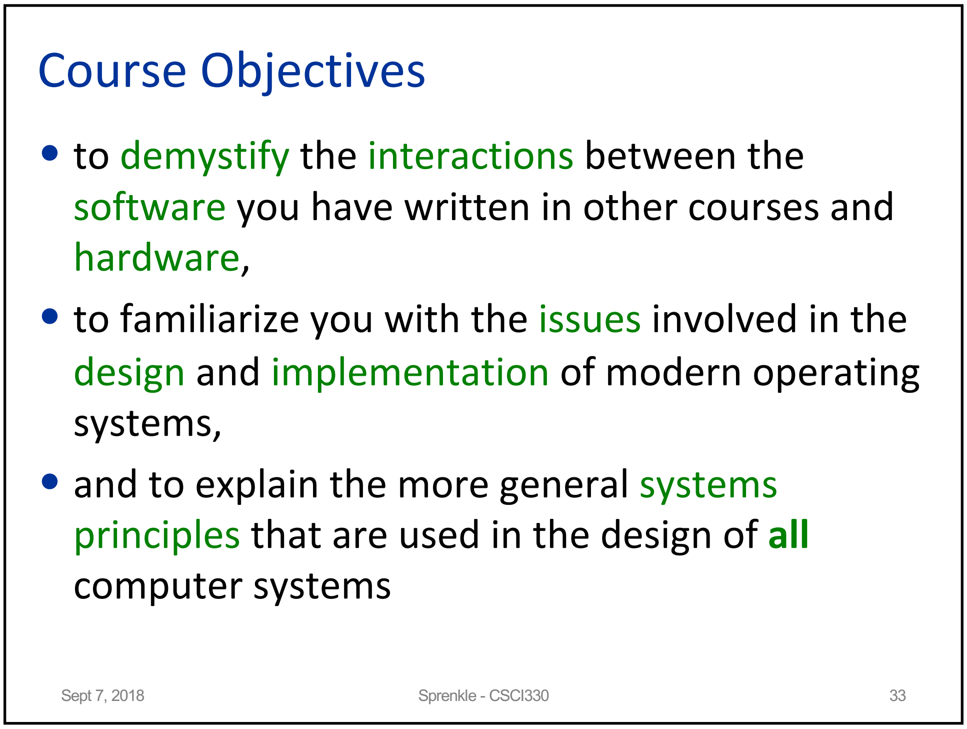
Example: Design Decisions

- Depends on computer system's goals

Provide an example of
a) a system that prefers ease of use to efficiency
b) a system that prefers efficiency to ease of use

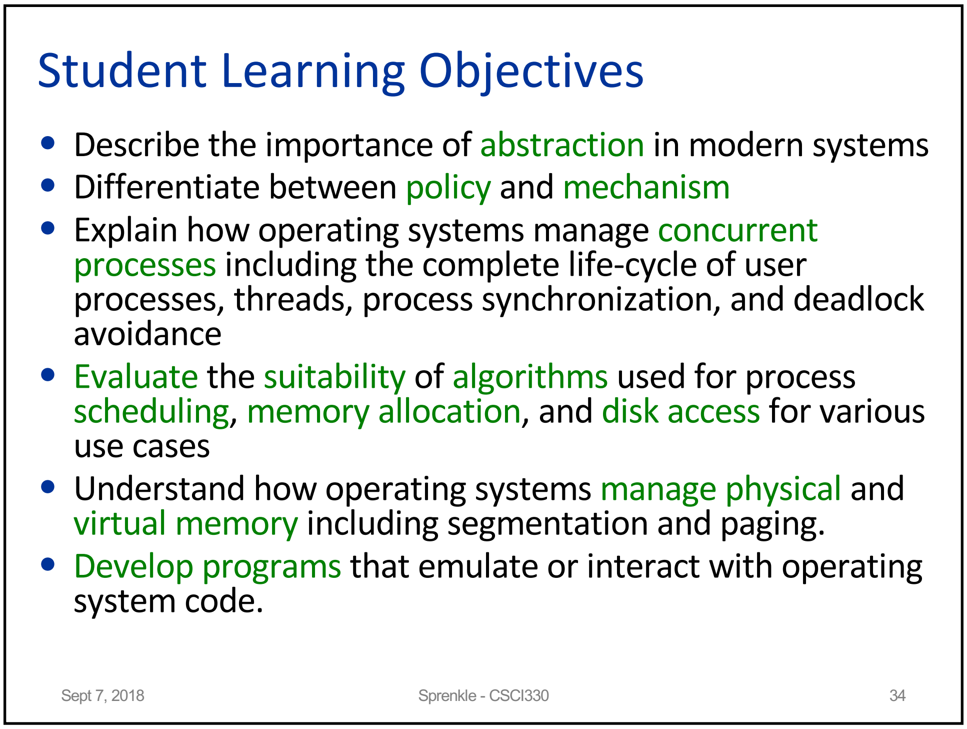


COURSE LOGISTICS



Course Objectives

- to demystify the interactions between the software you have written in other courses and hardware,
- to familiarize you with the issues involved in the design and implementation of modern operating systems,
- and to explain the more general systems principles that are used in the design of **all** computer systems



Student Learning Objectives

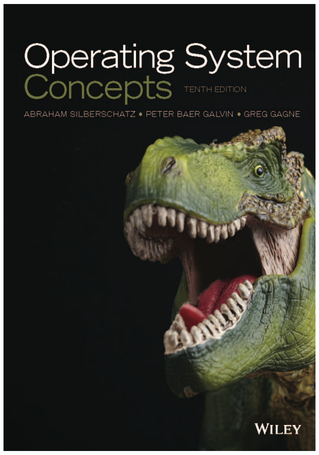
- Describe the importance of abstraction in modern systems
- Differentiate between policy and mechanism
- Explain how operating systems manage concurrent processes including the complete life-cycle of user processes, threads, process synchronization, and deadlock avoidance
- Evaluate the suitability of algorithms used for process scheduling, memory allocation, and disk access for various use cases
- Understand how operating systems manage physical and virtual memory including segmentation and paging.
- Develop programs that emulate or interact with operating system code.

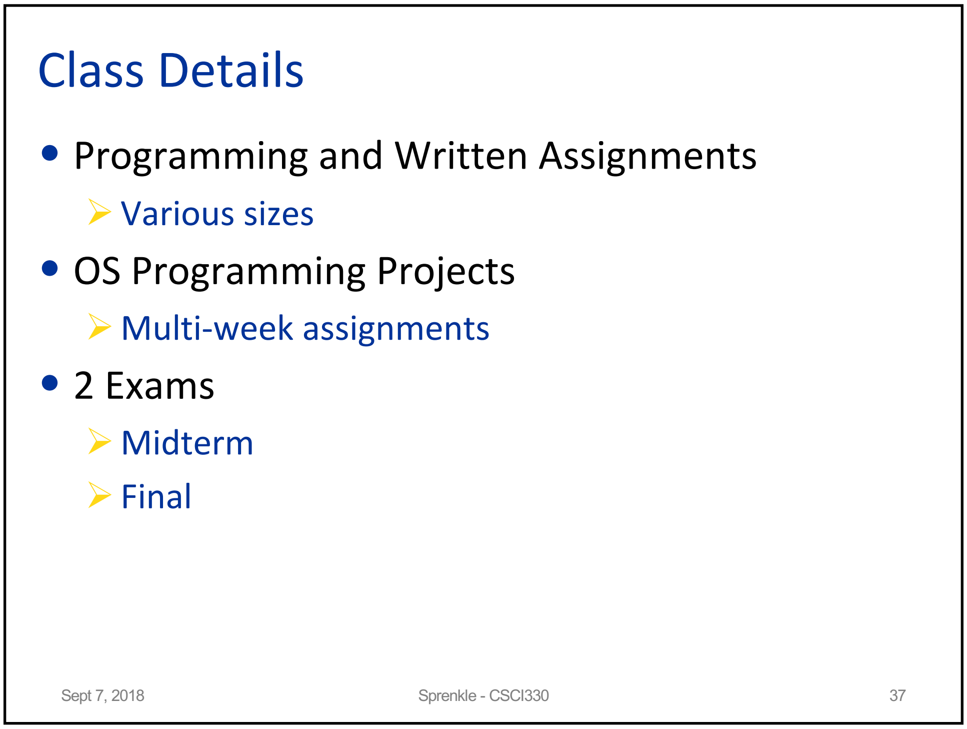
What this Course is NOT

- Deep dive into details of one particular OS
  - We will use Linux in the lab though!
- System administration / configuration

Class Details

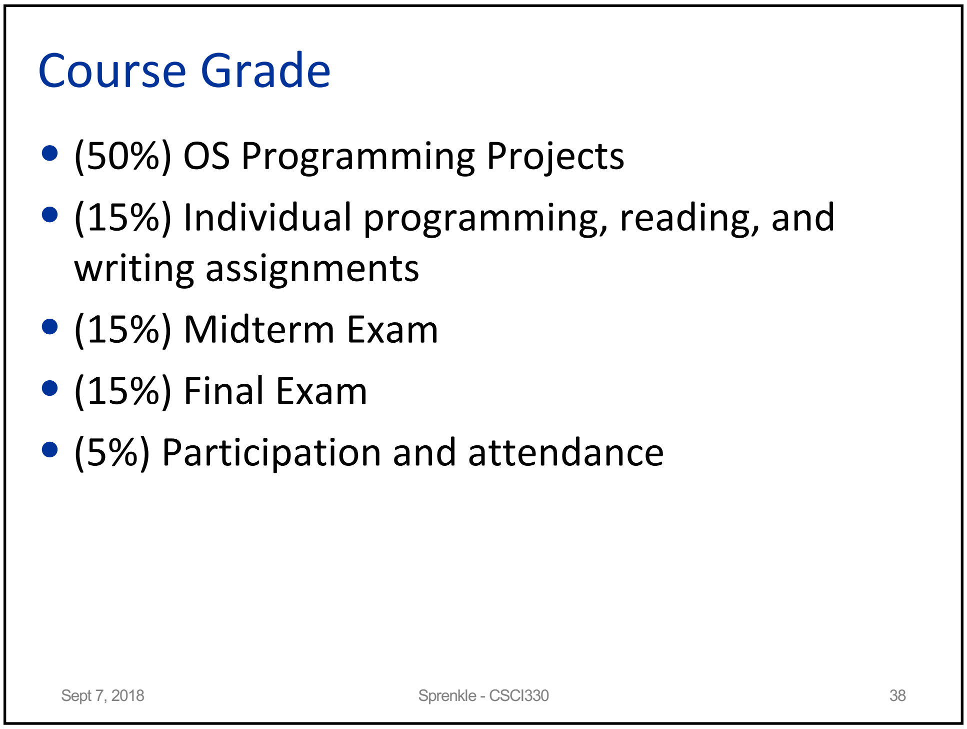
- Course Web Site `http://www.cs.wlu.edu/~sprenkle/cs330`
  - Slides, readings, assignments, resources
- Textbook
  - Operating System Concepts, Siberschatz, Galvin, and Gagne
  - eText
- Participation
  - Class discussions


Operating System Concepts
TENTH EDITION
ABRAHAM SILBERSCHATZ • PETER BAER GALVIN • GREG GAGNE
WILEY



Class Details

- Programming and Written Assignments
  - Various sizes
- OS Programming Projects
  - Multi-week assignments
- 2 Exams
  - Midterm
  - Final



Course Grade

- (50%) OS Programming Projects
- (15%) Individual programming, reading, and writing assignments
- (15%) Midterm Exam
- (15%) Final Exam
- (5%) Participation and attendance

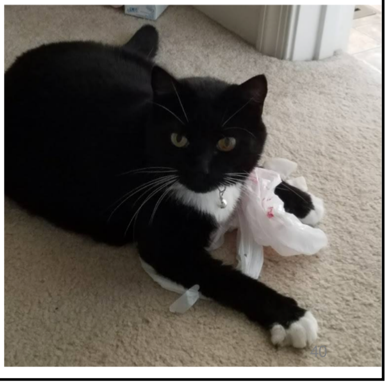
Course Dynamics

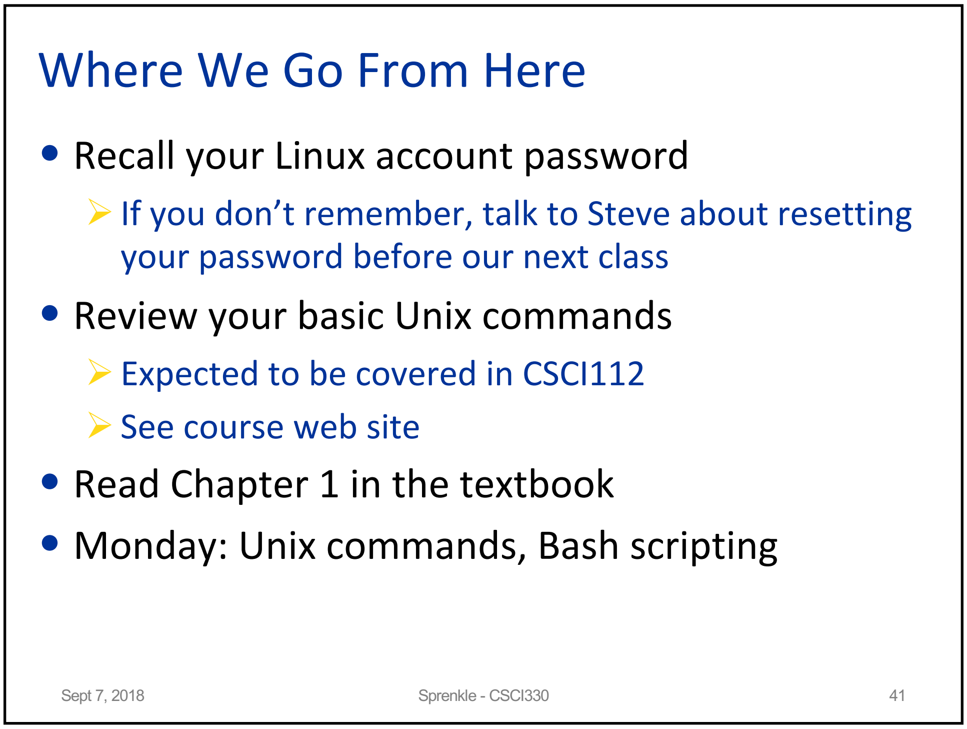
- Professor's Responsibilities:
  - Be prepared for class
  - Provide constructive feedback to students
  - Treat students with respect
  - Challenge and encourage students
  - Make material clear

- Student's Responsibilities
  - Be prepared for class (do readings and homework)
  - Give attention and effort in class to learning
  - Ask questions (during class and via email/Piazza)
  - Use professor's office hours
  - Let professor know if something is going wrong
  - Treat other students and professor with respect

My Bio

- From Dallastown, PA
- B.S., Gettysburg College
- M.S., Duke University
- Ph.D., University of Delaware
- For fun: pop culture, gardening
- Volunteer with Rockbridge Animal Alliance





Where We Go From Here

- Recall your Linux account password
  - If you don't remember, talk to Steve about resetting your password before our next class
- Review your basic Unix commands
  - Expected to be covered in CSCI112
  - See course web site
- Read Chapter 1 in the textbook
- Monday: Unix commands, Bash scripting