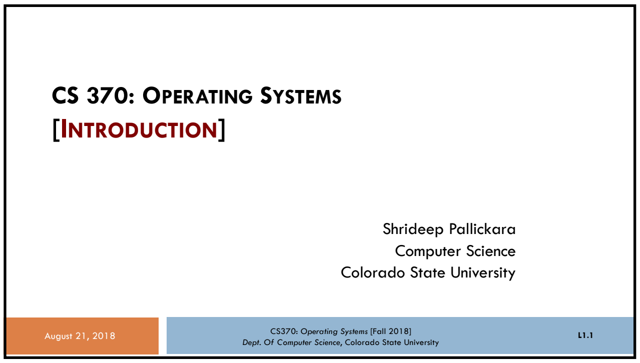
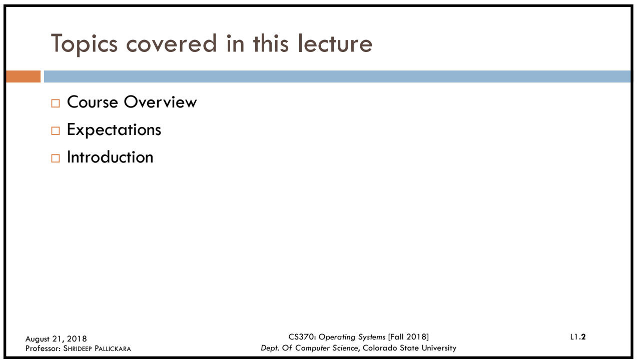


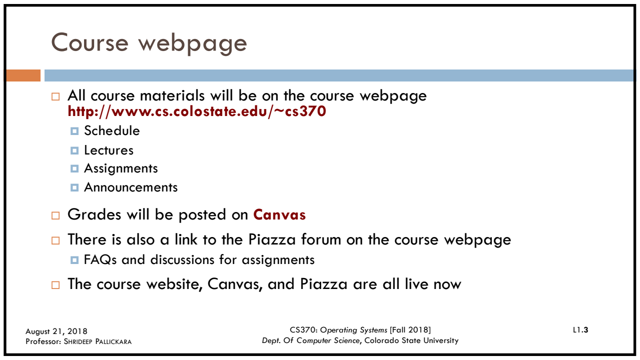
CS 370: Operating Systems
[Introduction]

Shrideep Pallickara
Computer Science
Colorado State University



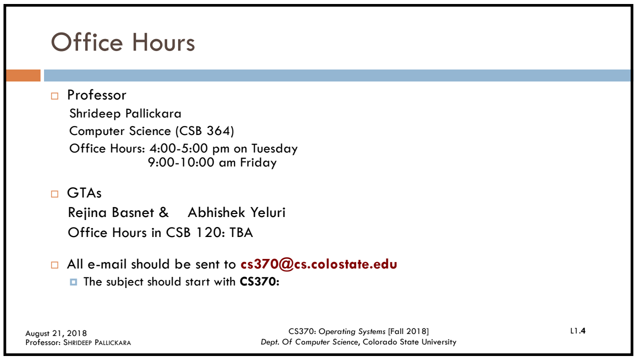
Topics covered in this lecture

- Course Overview
- Expectations
- Introduction



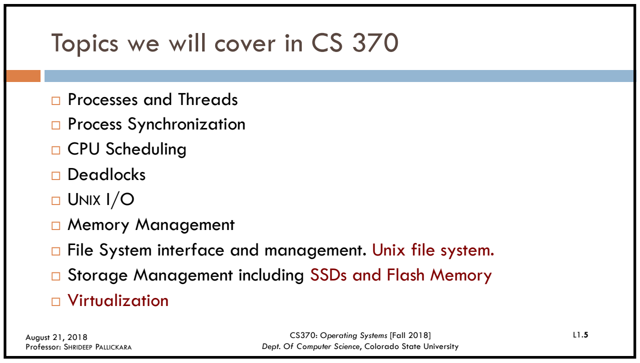
Course webpage

- All course materials will be on the course webpage **http://www.cs.colostate.edu/~cs370**
  - Schedule
  - Lectures
  - Assignments
  - Announcements
- Grades will be posted on **Canvas**
- There is also a link to the Piazza forum on the course webpage
  - FAQs and discussions for assignments
- The course website, Canvas, and Piazza are all live now



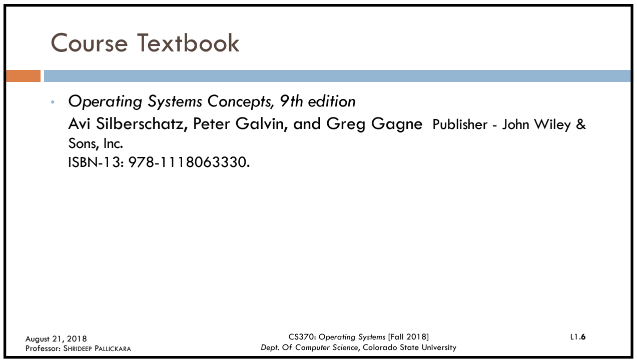
Office Hours

- Professor
  Shrideep Pallickara
  Computer Science (CSB 364)
  Office Hours: 4:00-5:00 pm on Tuesday
  9:00-10:00 am Friday
- GTAs
  Rejina Basnet & Abhishek Yeluri
  Office Hours in CSB 120: TBA
- All e-mail should be sent to **cs370@cs.colostate.edu**
  - The subject should start with **CS370:**



Topics we will cover in CS 370

- Processes and Threads
- Process Synchronization
- CPU Scheduling
- Deadlocks
- Unix I/O
- Memory Management
- File System interface and management. Unix file system.
- Storage Management including SSDs and Flash Memory
- Virtualization



Course Textbook

- *Operating Systems Concepts, 9th edition*
  Avi Silberschatz, Peter Galvin, and Greg Gagne Publisher - John Wiley & Sons, Inc.
  ISBN-13: 978-1118063330.

When I make slides, I usually refer to several texts. These include …

- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620*
- *Thomas Anderson and Michael Dahlin. Operating Systems: Principles and Practice, 2nd Edition. Recursive Books. ISBN: 0985673524/978-0985673529*
- *Kay Robbins & Steve Robbins. Unix Systems Programming, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2*
- I always list my references at the end of every slide set

On the schedule page

- You will see the **topics** that will be covered and the **order** in which I will cover them
- The readings section describe the chapters that I will cover
  - You will see chapters from other books besides the textbook; you are *not required* to read these
- You will also see the schedule for when the **assignments** will be posted and when they are due

GRADING

Grading breakdown

- Assignments: 45%
- Quizzes: 10%
- Mid Term: 20%
- Comprehensive Final Exam: 25%

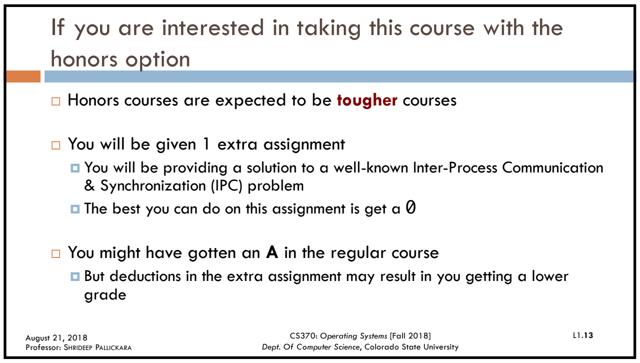
- 50 points before the 8th week
  - 50 points after
  - Midcourse grade will be posted before the course drop deadline

Grading Policy I

- Letter grades will be based on the following standard breakpoints:
  - >= 90 is an A, >= 88 is an A-,
  - >=86 is a B+, >=80 is a B, >=78 is a B-,
  - >=76 is a C+, >=70 is a C,
  - >=60 is a D, and <60 is an F.
- I will not cut higher than this, but I *may* cut lower.
- There is **no extra credit**
  - Any credit you earn, you must do so on a level-playing field with your peers
- There will be **no make-up exams**

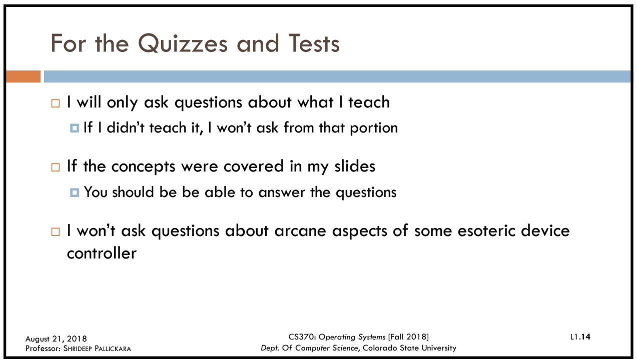
Grading Policy II

- Every assignment will be posted at least 2 weeks before the due date.
  - Every assignment will include information about how much it will count towards the course grade, and how it will be graded.
- Late submission penalty: 10% per-day for the first 2 days and a ZERO thereafter.
  - Detailed submission instructions posted on course website.
  - Assignments will be graded within 2 weeks of submission



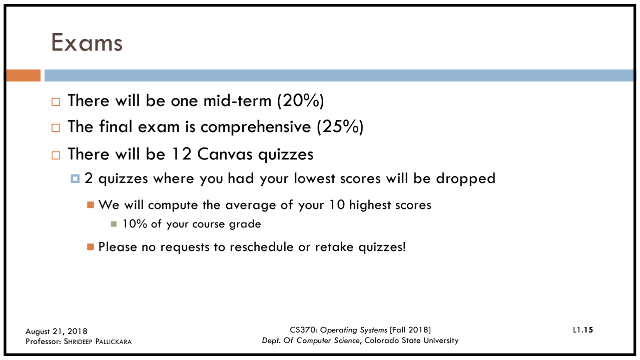
If you are interested in taking this course with the honors option

- Honors courses are expected to be **tougher** courses
- You will be given 1 extra assignment
  - You will be providing a solution to a well-known Inter-Process Communication & Synchronization (IPC) problem
  - The best you can do on this assignment is get a 0
- You might have gotten an **A** in the regular course
  - But deductions in the extra assignment may result in you getting a lower grade



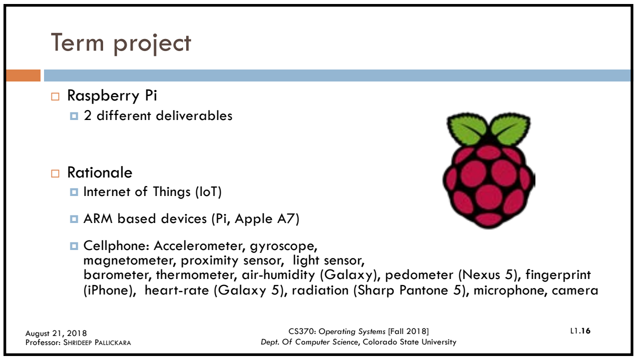
For the Quizzes and Tests

- I will only ask questions about what I teach
  - If I didn't teach it, I won't ask from that portion
- If the concepts were covered in my slides
  - You should be be able to answer the questions
- I won't ask questions about arcane aspects of some esoteric device controller



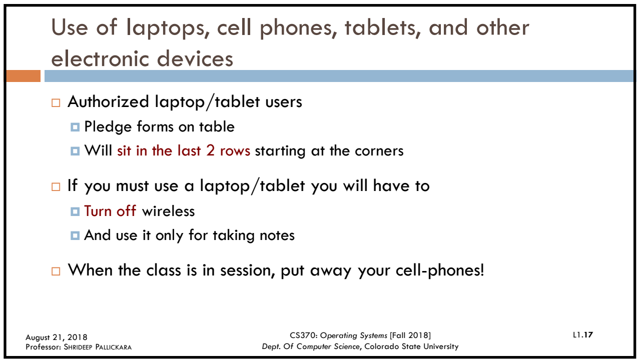
Exams

- There will be one mid-term (20%)
- The final exam is comprehensive (25%)
- There will be 12 Canvas quizzes
  - 2 quizzes where you had your lowest scores will be dropped
    - We will compute the average of your 10 highest scores
      - 10% of your course grade
    - Please no requests to reschedule or retake quizzes!



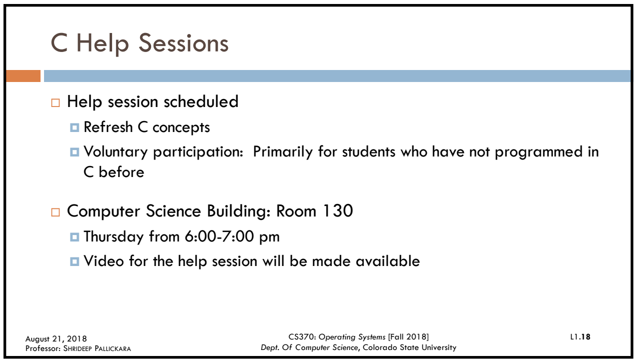
Term project

- Raspberry Pi
  - 2 different deliverables
- Rationale
  - Internet of Things (IoT)
  - ARM based devices (Pi, Apple A7)
  - Cellphone: Accelerometer, gyroscope, magnetometer, proximity sensor, light sensor, barometer, thermometer, air-humidity (Galaxy), pedometer (Nexus 5), fingerprint (iPhone), heart-rate (Galaxy 5), radiation (Sharp Pantone 5), microphone, camera



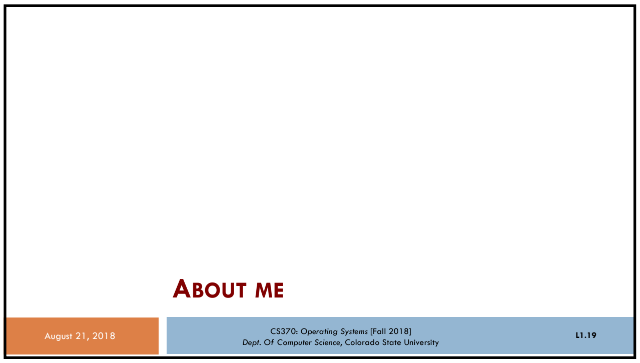
Use of laptops, cell phones, tablets, and other electronic devices

- Authorized laptop/tablet users
  - Pledge forms on table
  - Will sit in the last 2 rows starting at the corners
- If you must use a laptop/tablet you will have to
  - Turn off wireless
  - And use it only for taking notes
- When the class is in session, put away your cell-phones!

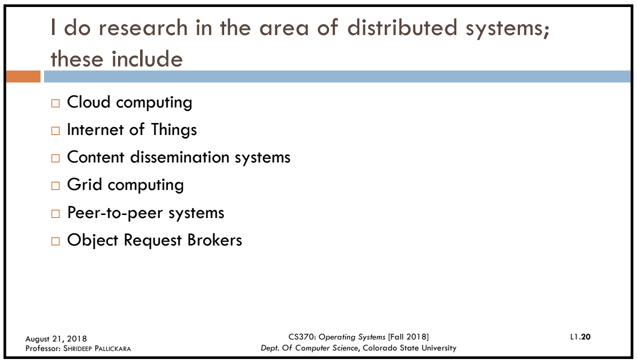


C Help Sessions

- Help session scheduled
  - Refresh C concepts
  - Voluntary participation: Primarily for students who have not programmed in C before
- Computer Science Building: Room 130
  - Thursday from 6:00-7:00 pm
  - Video for the help session will be made available

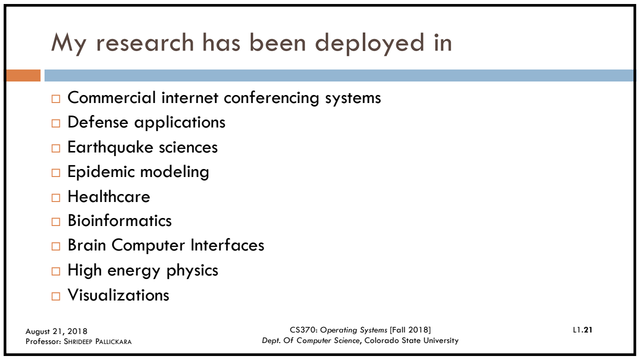


About me



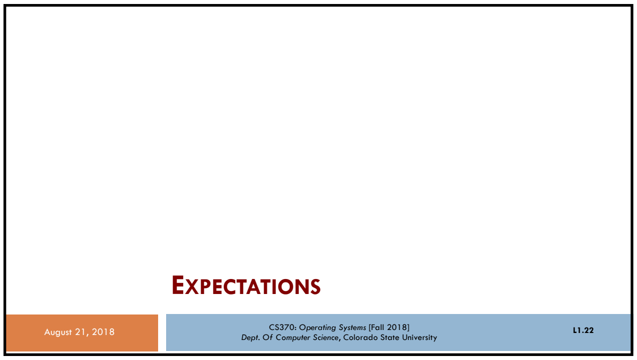
I do research in the area of distributed systems; these include

- Cloud computing
- Internet of Things
- Content dissemination systems
- Grid computing
- Peer-to-peer systems
- Object Request Brokers

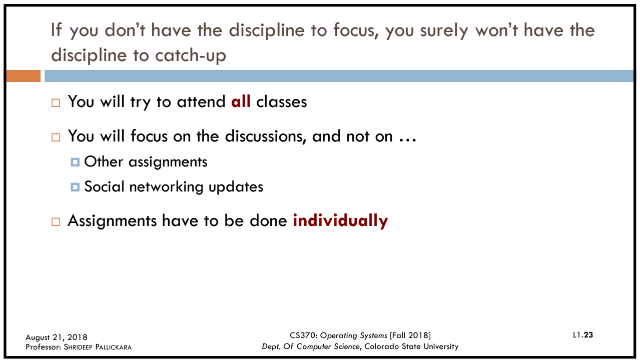


My research has been deployed in

- Commercial internet conferencing systems
- Defense applications
- Earthquake sciences
- Epidemic modeling
- Healthcare
- Bioinformatics
- Brain Computer Interfaces
- High energy physics
- Visualizations

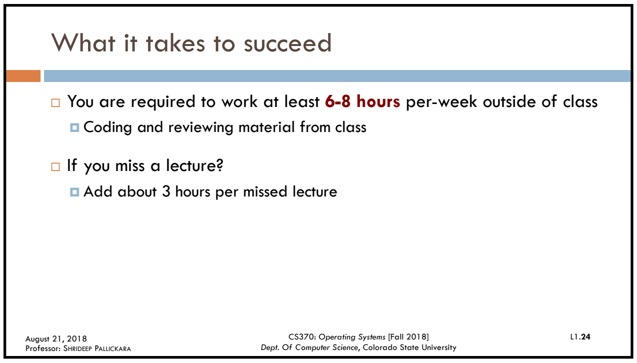


Expectations



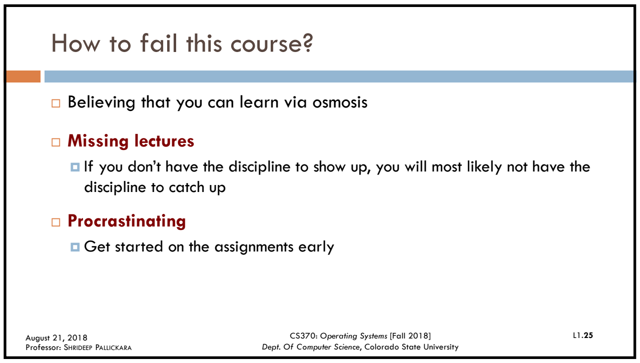
If you don't have the discipline to focus, you surely won't have the discipline to catch-up

- You will try to attend **all** classes
- You will focus on the discussions, and not on …
  - Other assignments
  - Social networking updates
- Assignments have to be done **individually**



What it takes to succeed

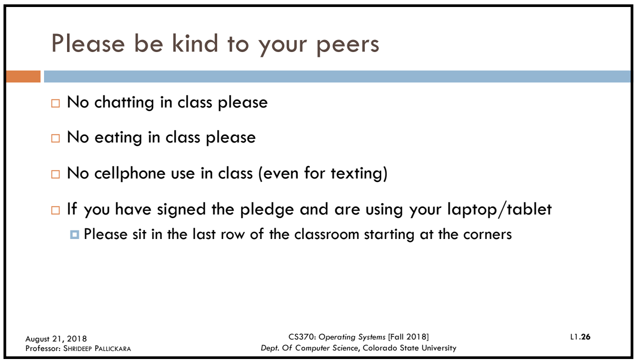
- You are required to work at least **6-8 hours** per-week outside of class
  - Coding and reviewing material from class
- If you miss a lecture?
  - Add about 3 hours per missed lecture



How to fail this course?

- Believing that you can learn via osmosis
- **Missing lectures**
  - If you don't have the discipline to show up, you will most likely not have the discipline to catch up
- **Procrastinating**
  - Get started on the assignments early

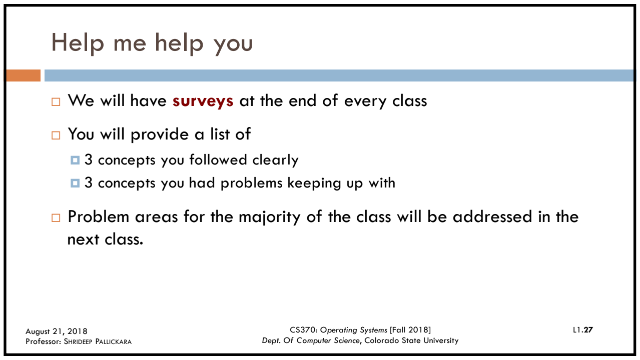
August 21, 2018
Professor: Shrideep Pallickara
CS370: *Operating Systems* [Fall 2018]
Dept. Of Computer Science, Colorado State University
L1.**25**



Please be kind to your peers

- No chatting in class please
- No eating in class please
- No cellphone use in class (even for texting)
- If you have signed the pledge and are using your laptop/tablet
  - Please sit in the last row of the classroom starting at the corners

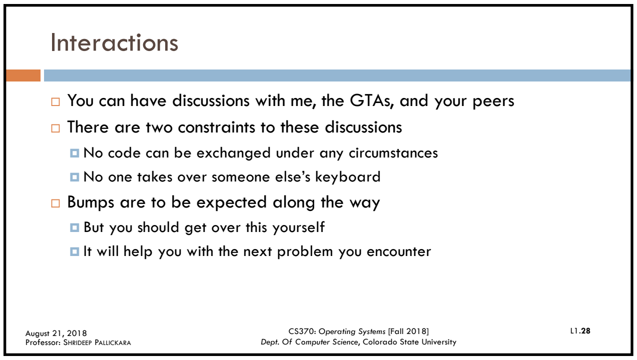
August 21, 2018
Professor: Shrideep Pallickara
CS370: *Operating Systems* [Fall 2018]
Dept. Of Computer Science, Colorado State University
L1.**26**



Help me help you

- We will have **surveys** at the end of every class
- You will provide a list of
  - 3 concepts you followed clearly
  - 3 concepts you had problems keeping up with
- Problem areas for the majority of the class will be addressed in the next class.

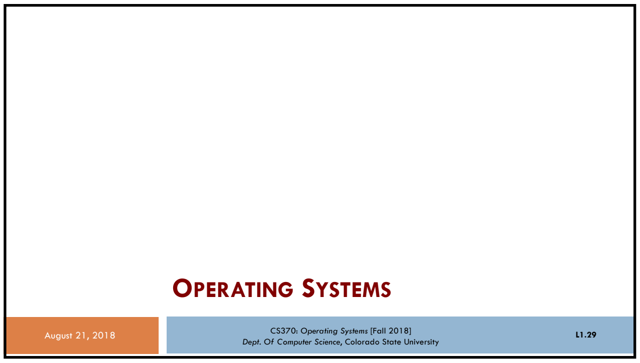
August 21, 2018
Professor: Shrideep Pallickara
CS370: *Operating Systems* [Fall 2018]
Dept. Of Computer Science, Colorado State University
L1.**27**



Interactions

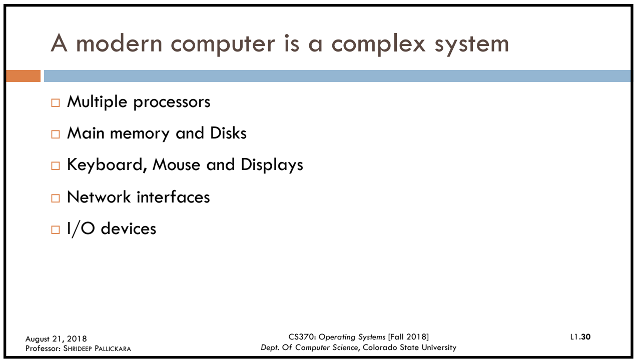
- You can have discussions with me, the GTAs, and your peers
- There are two constraints to these discussions
  - No code can be exchanged under any circumstances
  - No one takes over someone else's keyboard
- Bumps are to be expected along the way
  - But you should get over this yourself
  - It will help you with the next problem you encounter

August 21, 2018
Professor: Shrideep Pallickara
CS370: *Operating Systems* [Fall 2018]
Dept. Of Computer Science, Colorado State University
L1.**28**



Operating Systems

August 21, 2018
CS370: *Operating Systems* [Fall 2018]
Dept. Of Computer Science, Colorado State University
L1.29



A modern computer is a complex system

- Multiple processors
- Main memory and Disks
- Keyboard, Mouse and Displays
- Network interfaces
- I/O devices

August 21, 2018
Professor: Shrideep Pallickara
CS370: *Operating Systems* [Fall 2018]
Dept. Of Computer Science, Colorado State University
L1.**30**

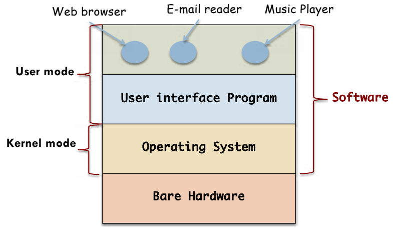
Why do we need Operating Systems?

- If every programmer had to understand how *all* these components work?
  - Software development would be arduous
- Managing all components and using them optimally is a challenge

Computers are equipped with a layer of software

- Called the **Operating System**
- Functionality:
  - Provide user programs with a better, simpler, cleaner model of the computer
  - Manage resources efficiently

Where the operating system fits in [1/3]


Web browser
E-mail reader
Music Player
User mode
User interface Program
Software
Kernel mode
Operating System
Bare Hardware

Where the operating system fits in [2/3]

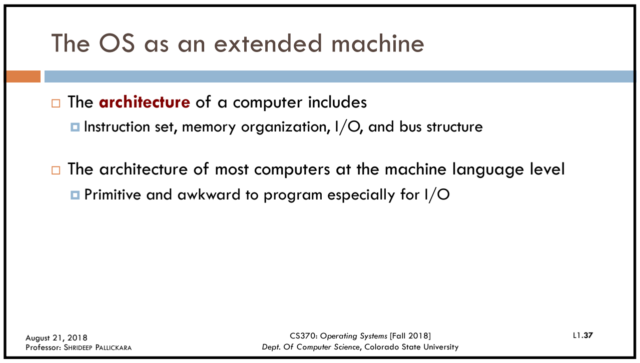
- The OS runs on bare hardware in **kernel mode**
  - *Complete access* to all hardware
  - Can execute *any* instruction that the machine is capable of executing
- Provides the base for all software
  - Rest of the software runs in **user-mode**
    - Only a **subset** of machine instructions is available

Where the operating system fits in [3/3]

- Users interact with applications
  - Applications execute in an environment provided by the operating system
    - And the operating system mediates access to the underlying hardware

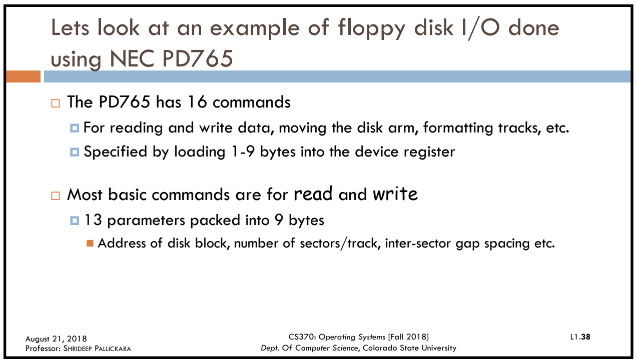
The application context is much more than a simple abstraction on top of hardware devices

- Applications execute in a virtual environment that is more **constrained** (to prevent harm)
- More **powerful** (to mask hardware limitations), and …
- More **useful** (via common services) than the underlying hardware



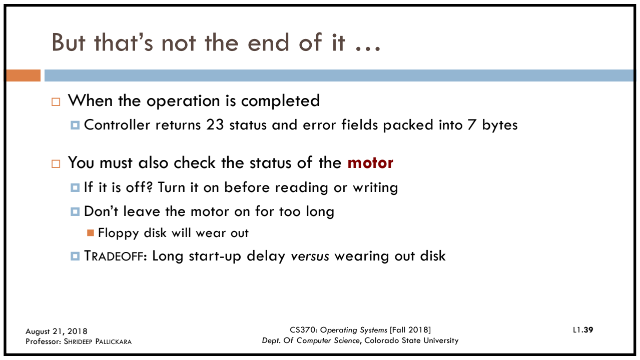
The OS as an extended machine

- The **architecture** of a computer includes
  - Instruction set, memory organization, I/O, and bus structure
- The architecture of most computers at the machine language level
  - Primitive and awkward to program especially for I/O



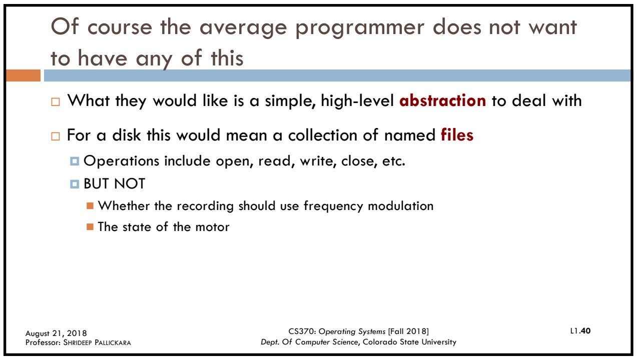
Lets look at an example of floppy disk I/O done using NEC PD765

- The PD765 has 16 commands
  - For reading and write data, moving the disk arm, formatting tracks, etc.
  - Specified by loading 1-9 bytes into the device register
- Most basic commands are for read and write
  - 13 parameters packed into 9 bytes
    - Address of disk block, number of sectors/track, inter-sector gap spacing etc.



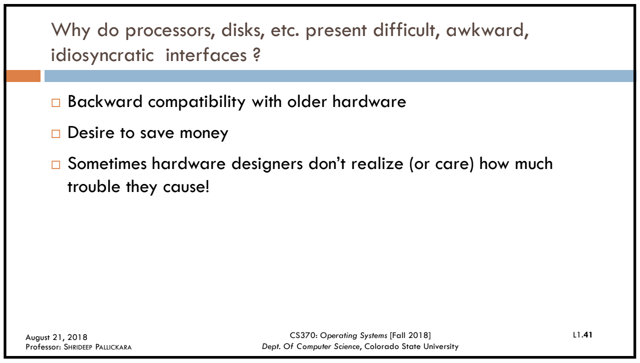
But that's not the end of it …

- When the operation is completed
  - Controller returns 23 status and error fields packed into 7 bytes
- You must also check the status of the **motor**
  - If it is off? Turn it on before reading or writing
  - Don't leave the motor on for too long
    - Floppy disk will wear out
  - Tradeoff: Long start-up delay *versus* wearing out disk



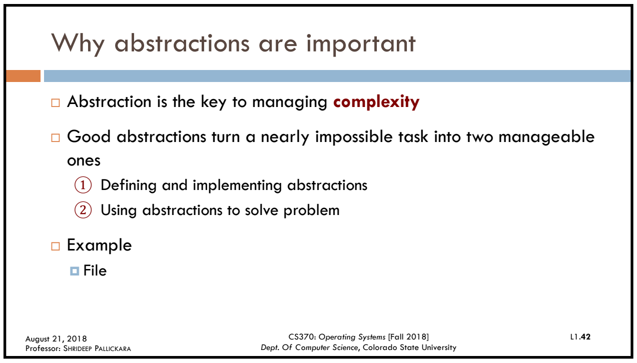
Of course the average programmer does not want to have any of this

- What they would like is a simple, high-level **abstraction** to deal with
- For a disk this would mean a collection of named **files**
  - Operations include open, read, write, close, etc.
  - BUT NOT
    - Whether the recording should use frequency modulation
    - The state of the motor



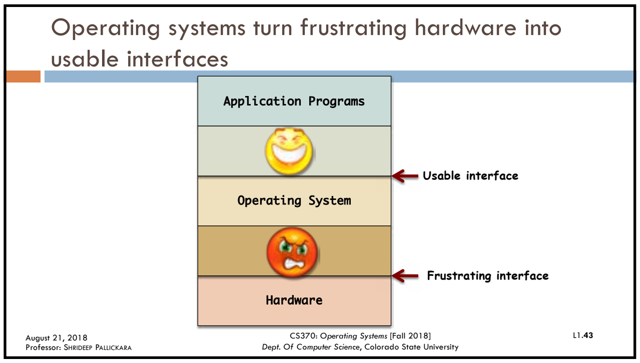
Why do processors, disks, etc. present difficult, awkward, idiosyncratic interfaces ?

- Backward compatibility with older hardware
- Desire to save money
- Sometimes hardware designers don't realize (or care) how much trouble they cause!

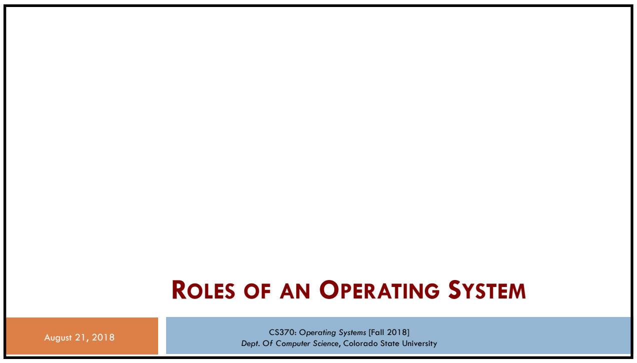


Why abstractions are important

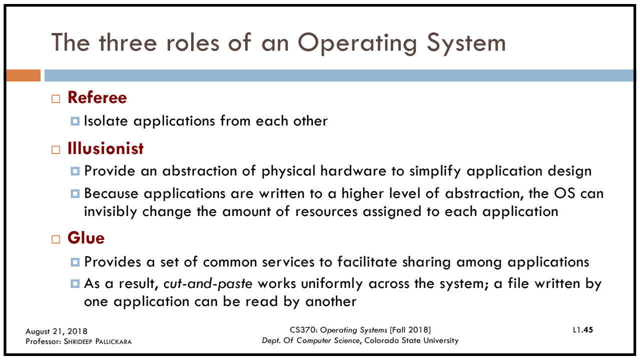
- Abstraction is the key to managing **complexity**
- Good abstractions turn a nearly impossible task into two manageable ones
  1. Defining and implementing abstractions
  2. Using abstractions to solve problem
- Example
  - File



Operating systems turn frustrating hardware into usable interfaces

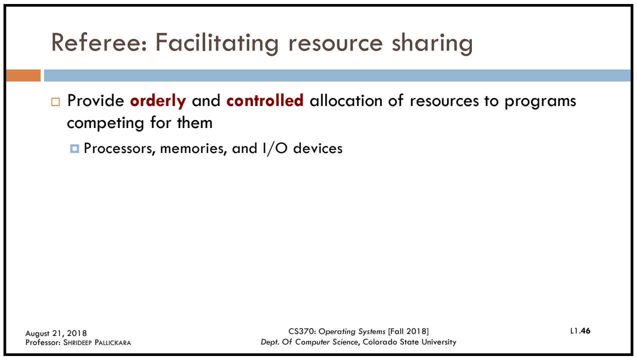


Roles of an Operating System



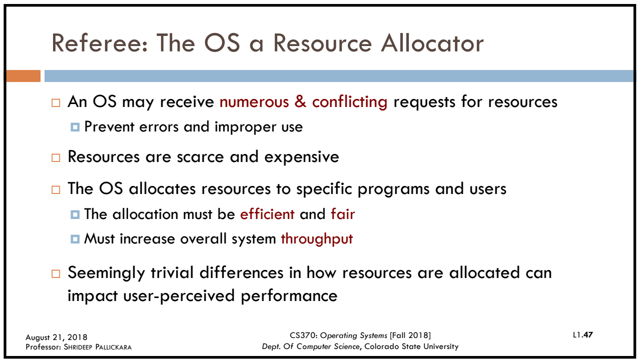
The three roles of an Operating System

- **Referee**
  - Isolate applications from each other
- **Illusionist**
  - Provide an abstraction of physical hardware to simplify application design
  - Because applications are written to a higher level of abstraction, the OS can invisibly change the amount of resources assigned to each application
- **Glue**
  - Provides a set of common services to facilitate sharing among applications
  - As a result, *cut-and-paste* works uniformly across the system; a file written by one application can be read by another



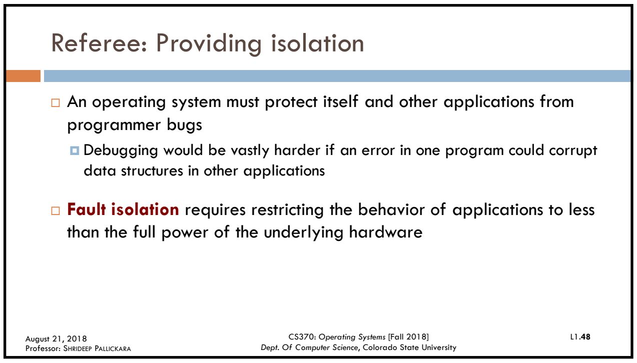
Referee: Facilitating resource sharing

- Provide **orderly** and **controlled** allocation of resources to programs competing for them
  - Processors, memories, and I/O devices



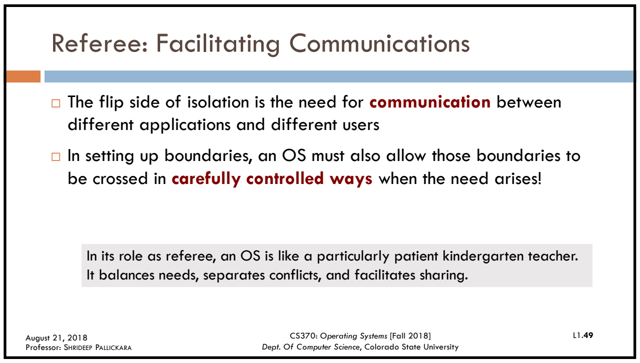
Referee: The OS a Resource Allocator

- An OS may receive numerous & conflicting requests for resources
  - Prevent errors and improper use
- Resources are scarce and expensive
- The OS allocates resources to specific programs and users
  - The allocation must be efficient and fair
  - Must increase overall system throughput
- Seemingly trivial differences in how resources are allocated can impact user-perceived performance



Referee: Providing isolation

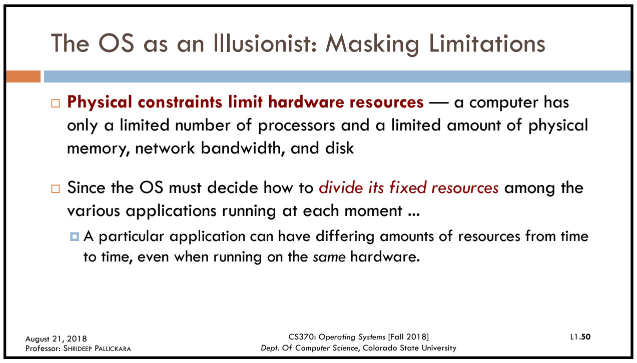
- An operating system must protect itself and other applications from programmer bugs
  - Debugging would be vastly harder if an error in one program could corrupt data structures in other applications
- **Fault isolation** requires restricting the behavior of applications to less than the full power of the underlying hardware



Referee: Facilitating Communications

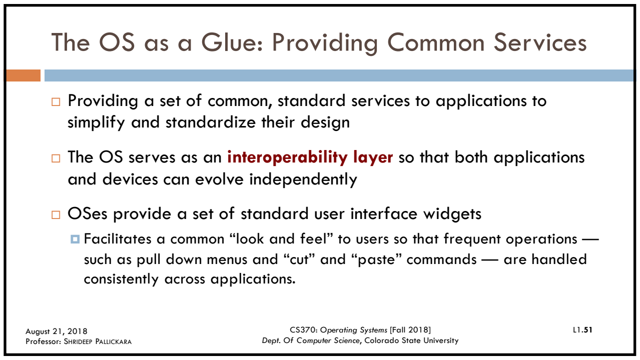
- The flip side of isolation is the need for **communication** between different applications and different users
- In setting up boundaries, an OS must also allow those boundaries to be crossed in **carefully controlled ways** when the need arises!

In its role as referee, an OS is like a particularly patient kindergarten teacher. It balances needs, separates conflicts, and facilitates sharing.



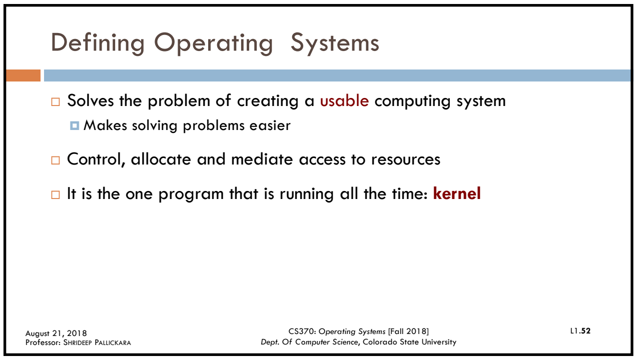
The OS as an Illusionist: Masking Limitations

- **Physical constraints limit hardware resources** — a computer has only a limited number of processors and a limited amount of physical memory, network bandwidth, and disk
- Since the OS must decide how to *divide its fixed resources* among the various applications running at each moment …
  - A particular application can have differing amounts of resources from time to time, even when running on the *same* hardware.



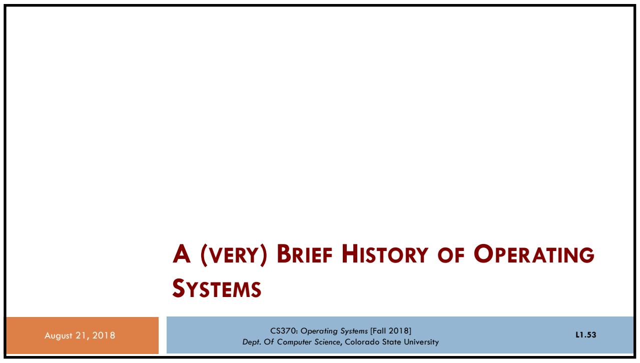
The OS as a Glue: Providing Common Services

- Providing a set of common, standard services to applications to simplify and standardize their design
- The OS serves as an **interoperability layer** so that both applications and devices can evolve independently
- OSes provide a set of standard user interface widgets
  - Facilitates a common "look and feel" to users so that frequent operations — such as pull down menus and "cut" and "paste" commands — are handled consistently across applications.

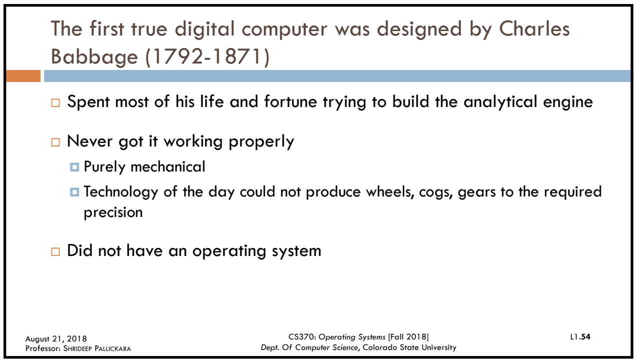


Defining Operating Systems

- Solves the problem of creating a usable computing system
  - Makes solving problems easier
- Control, allocate and mediate access to resources
- It is the one program that is running all the time: **kernel**

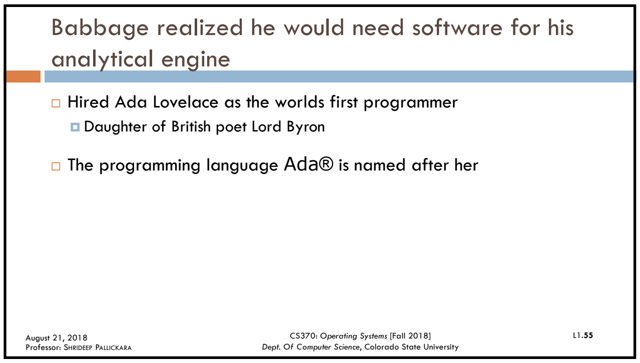


A (Very) Brief History of Operating Systems



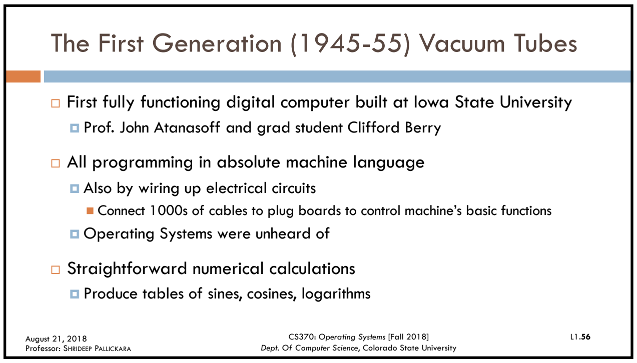
The first true digital computer was designed by Charles Babbage (1792-1871)

- Spent most of his life and fortune trying to build the analytical engine
- Never got it working properly
  - Purely mechanical
  - Technology of the day could not produce wheels, cogs, gears to the required precision
- Did not have an operating system



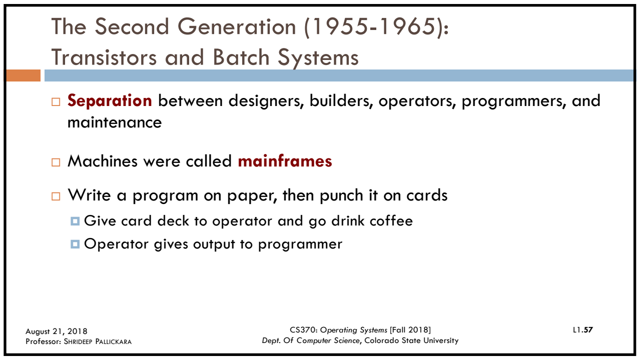
Babbage realized he would need software for his analytical engine

- Hired Ada Lovelace as the worlds first programmer
  - Daughter of British poet Lord Byron
- The programming language Ada® is named after her



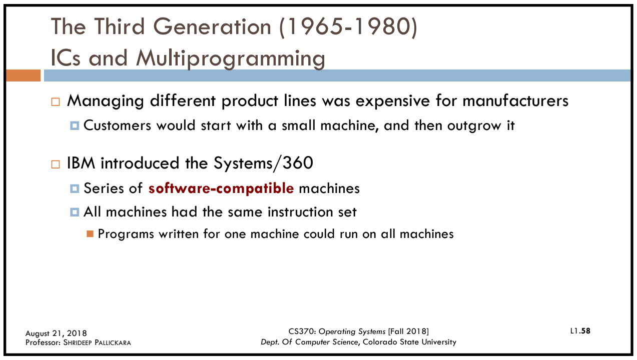
The First Generation (1945-55) Vacuum Tubes

- First fully functioning digital computer built at Iowa State University
  - Prof. John Atanasoff and grad student Clifford Berry
- All programming in absolute machine language
  - Also by wiring up electrical circuits
    - Connect 1000s of cables to plug boards to control machine's basic functions
  - Operating Systems were unheard of
- Straightforward numerical calculations
  - Produce tables of sines, cosines, logarithms



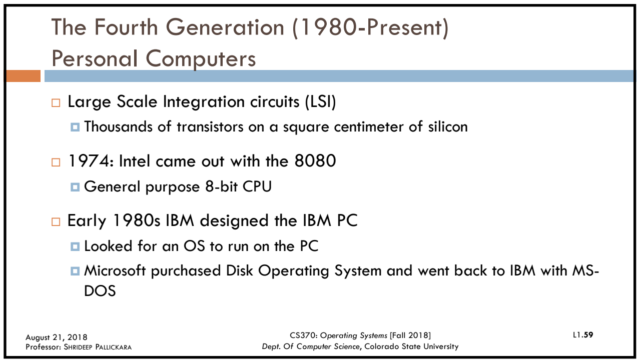
The Second Generation (1955-1965): Transistors and Batch Systems

- **Separation** between designers, builders, operators, programmers, and maintenance
- Machines were called **mainframes**
- Write a program on paper, then punch it on cards
  - Give card deck to operator and go drink coffee
  - Operator gives output to programmer



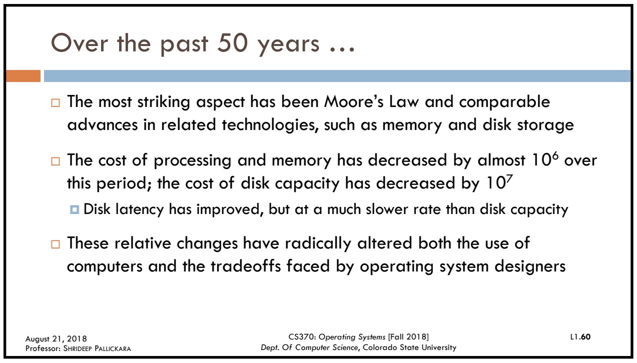
The Third Generation (1965-1980) ICs and Multiprogramming

- Managing different product lines was expensive for manufacturers
  - Customers would start with a small machine, and then outgrow it
- IBM introduced the Systems/360
  - Series of **software-compatible** machines
  - All machines had the same instruction set
    - Programs written for one machine could run on all machines



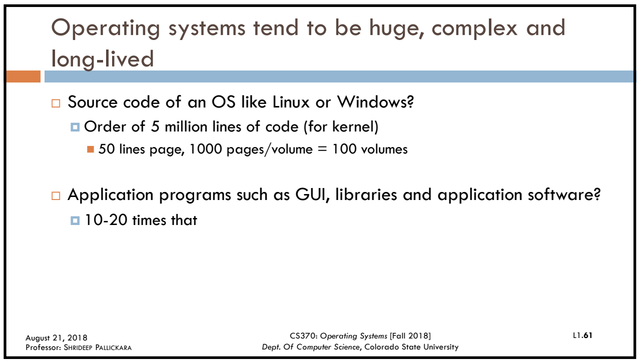
The Fourth Generation (1980-Present) Personal Computers

- Large Scale Integration circuits (LSI)
  - Thousands of transistors on a square centimeter of silicon
- 1974: Intel came out with the 8080
  - General purpose 8-bit CPU
- Early 1980s IBM designed the IBM PC
  - Looked for an OS to run on the PC
  - Microsoft purchased Disk Operating System and went back to IBM with MS-DOS



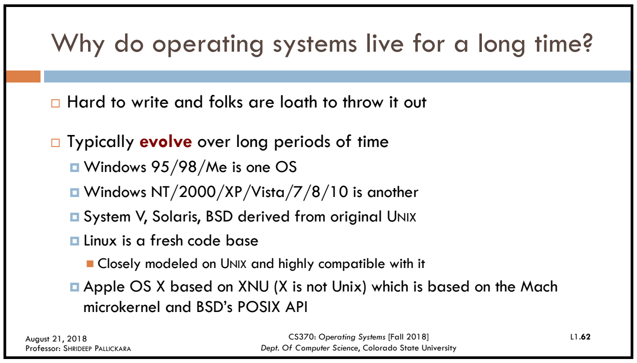
Over the past 50 years …

- The most striking aspect has been Moore's Law and comparable advances in related technologies, such as memory and disk storage
- The cost of processing and memory has decreased by almost 10^6 over this period; the cost of disk capacity has decreased by 10^7
  - Disk latency has improved, but at a much slower rate than disk capacity
- These relative changes have radically altered both the use of computers and the tradeoffs faced by operating system designers



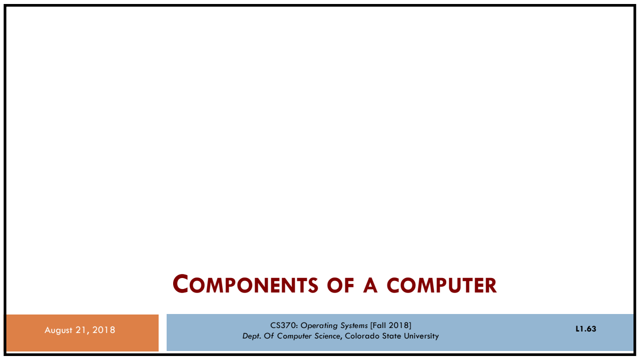
Operating systems tend to be huge, complex and long-lived

- Source code of an OS like Linux or Windows?
  - Order of 5 million lines of code (for kernel)
    - 50 lines page, 1000 pages/volume = 100 volumes
- Application programs such as GUI, libraries and application software?
  - 10-20 times that

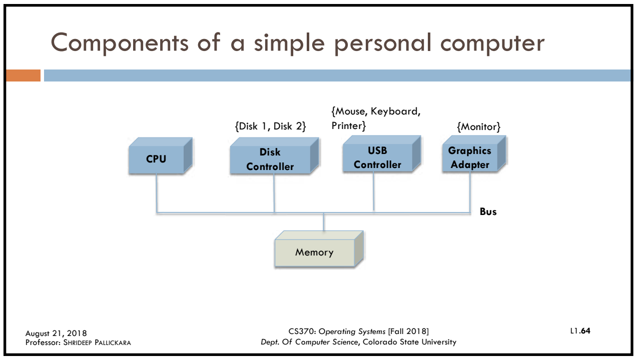


Why do operating systems live for a long time?

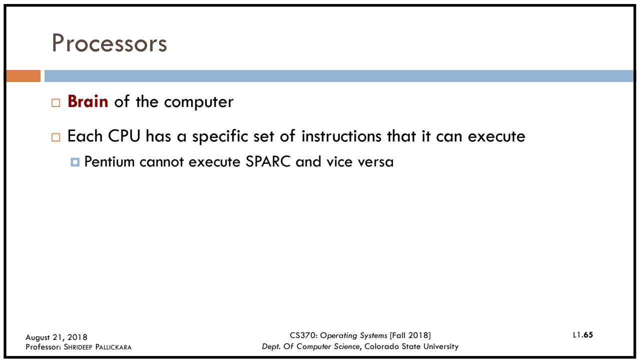
- Hard to write and folks are loath to throw it out
- Typically **evolve** over long periods of time
  - Windows 95/98/Me is one OS
  - Windows NT/2000/XP/Vista/7/8/10 is another
  - System V, Solaris, BSD derived from original Unix
  - Linux is a fresh code base
    - Closely modeled on Unix and highly compatible with it
  - Apple OS X based on XNU (X is not Unix) which is based on the Mach microkernel and BSD's POSIX API



Components of a computer

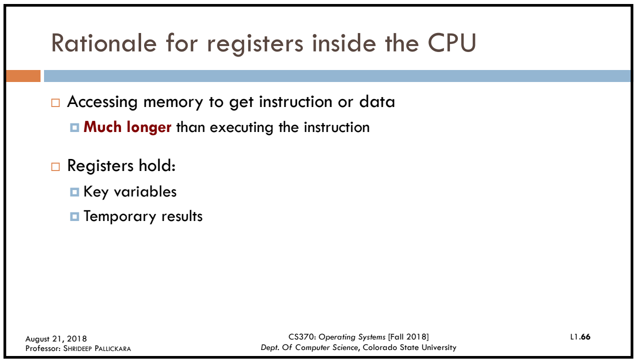


Components of a simple personal computer



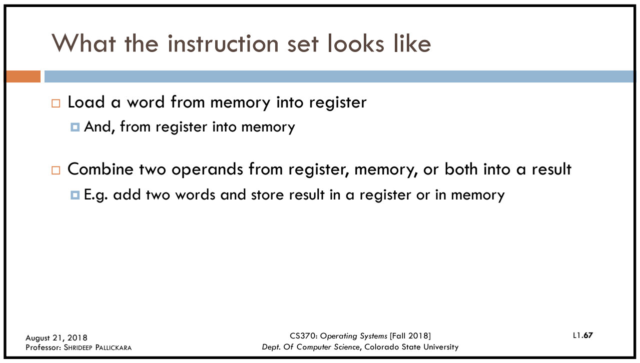
Processors

- **Brain** of the computer
- Each CPU has a specific set of instructions that it can execute
  - Pentium cannot execute SPARC and vice versa



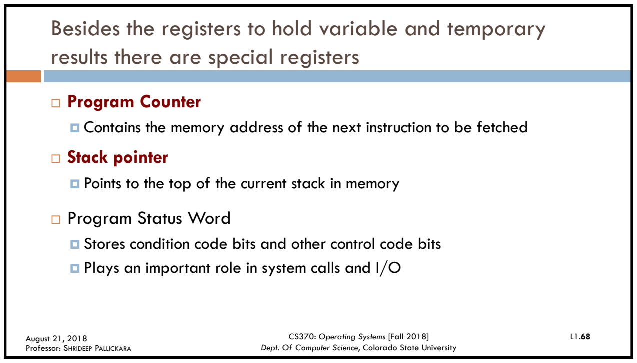
Rationale for registers inside the CPU

- Accessing memory to get instruction or data
  - **Much longer** than executing the instruction
- Registers hold:
  - Key variables
  - Temporary results



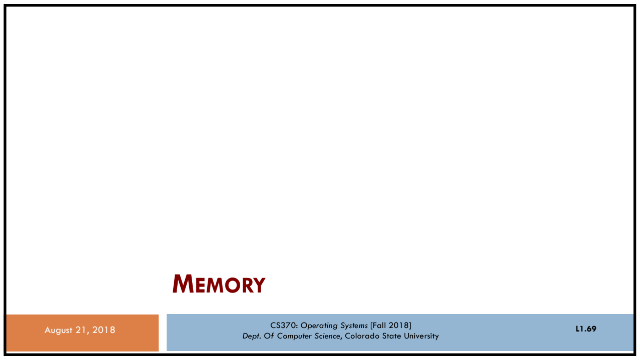
What the instruction set looks like

- Load a word from memory into register
  - And, from register into memory
- Combine two operands from register, memory, or both into a result
  - E.g. add two words and store result in a register or in memory

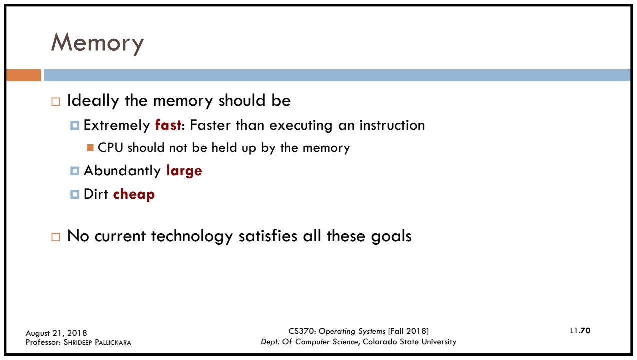


Besides the registers to hold variable and temporary results there are special registers

- **Program Counter**
  - Contains the memory address of the next instruction to be fetched
- **Stack pointer**
  - Points to the top of the current stack in memory
- Program Status Word
  - Stores condition code bits and other control code bits
  - Plays an important role in system calls and I/O

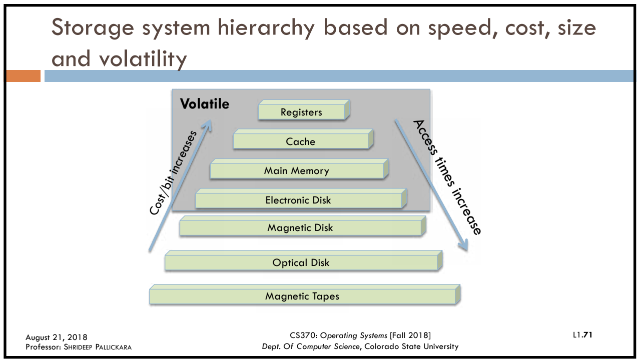


Memory

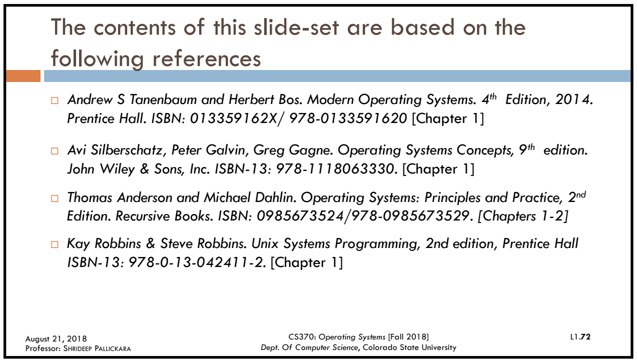


Memory

- Ideally the memory should be
  - Extremely **fast**: Faster than executing an instruction
    - CPU should not be held up by the memory
  - Abundantly **large**
  - Dirt **cheap**
- No current technology satisfies all these goals



Storage system hierarchy based on speed, cost, size and volatility



The contents of this slide-set are based on the following references

- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620* [Chapter 1]
- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 1]
- *Thomas Anderson and Michael Dahlin. Operating Systems: Principles and Practice, 2nd Edition. Recursive Books. ISBN: 0985673524/978-0985673529. [Chapters 1-2]*
- *Kay Robbins & Steve Robbins. Unix Systems Programming, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2.* [Chapter 1]