

# Lecture 01: Introduction

CS343 – Operating Systems  
Branden Ghena – Spring 2024

Some slides borrowed from:

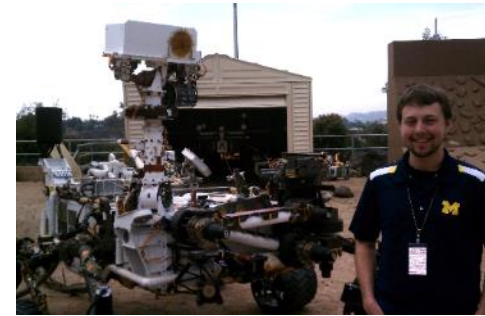
Stephen Tarzia (Northwestern), Jaswinder Pal Singh (Princeton), and UC Berkeley CS162

# Welcome to CS343!

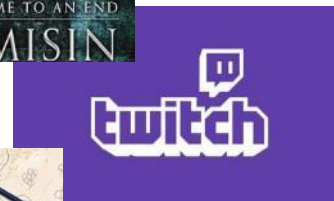
- In brief: how does the operating system work and why?
- Role of the Operating System
  - **Manages** hardware resources
  - Provides **abstractions** to support processes
- Major topics
  - Concurrency
  - Scheduling
  - Devices
  - Virtual Memory
  - File Systems

# Branden Ghena (he/him)

- Assistant Faculty of Instruction
- Education
  - Undergrad: Michigan Tech
  - Master's: University of Michigan
  - PhD: University of California, Berkeley
- Research
  - Resource-constrained sensing systems
  - Low-energy wireless networks
  - Embedded operating systems
- Teaching
  - Computer Systems
    - CS211: Fundamentals of Programming II
    - CS213: Intro to Computer Systems
    - CS343: Operating Systems
    - CE346: Microprocessor System Design
    - CS397: Wireless Protocols for the IoT



## Things I love



# Today's Goals

- Discuss the role of an Operating System
- Introduce theme and goals of the course
- Describe how this class is going to function
- Explore trends in OS history

# Outline

- **What is an OS?**
- Logistics
- Operating Systems History
- CS343 Focus

# Computers come in incredible diversity

**Bell's Law:  
New computer  
class every 10  
years**

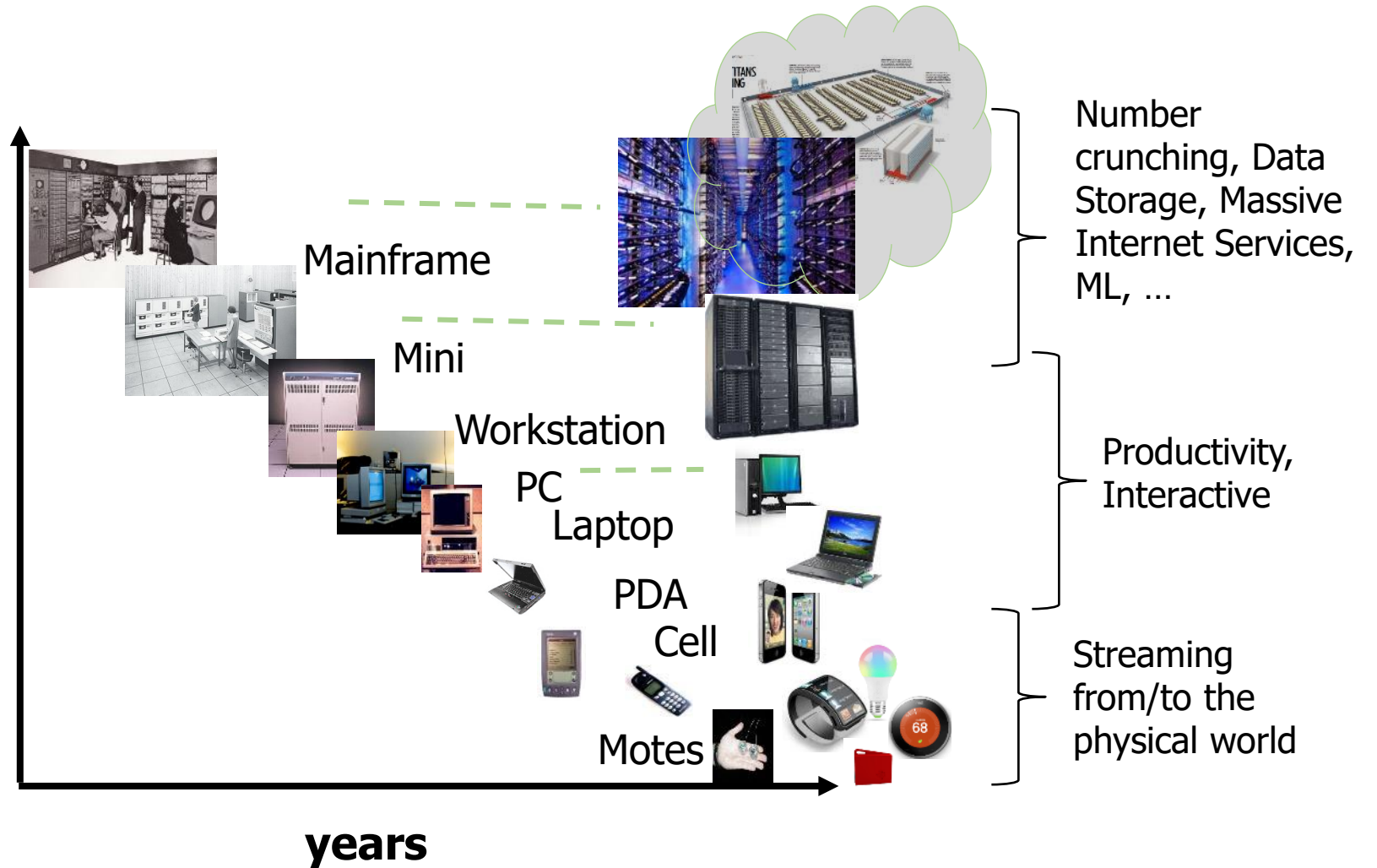
**Ratio of  
Computers  
to People**

**1:10<sup>6</sup>**

**1:10<sup>3</sup>**

**1:1**

**10<sup>3</sup>:1**



# Computing timescales are increasingly large

**Jeff Dean  
(Google AI):  
“Numbers Everyone  
Should Know”**

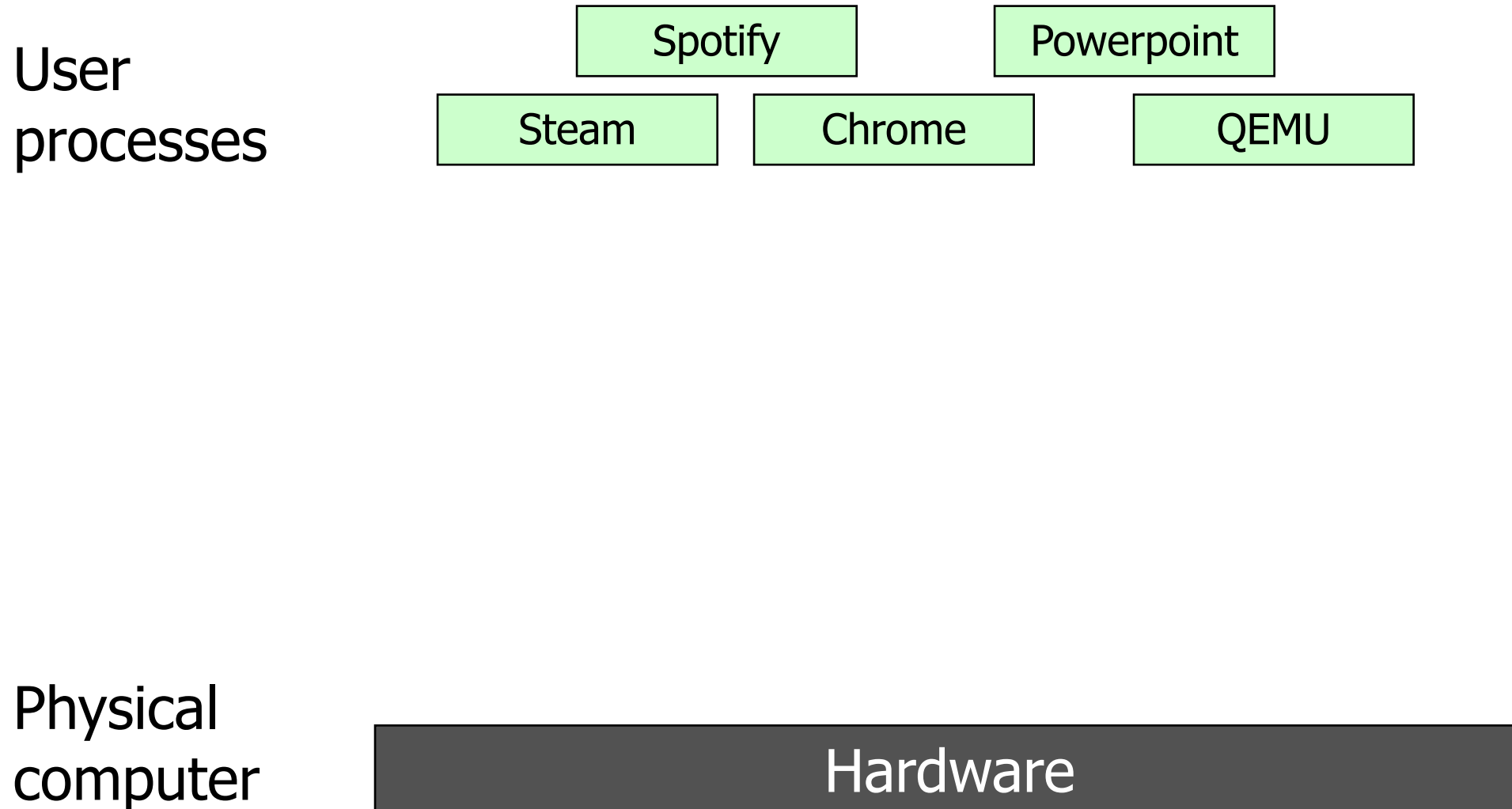
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

# Operating systems are at the heart of these challenges

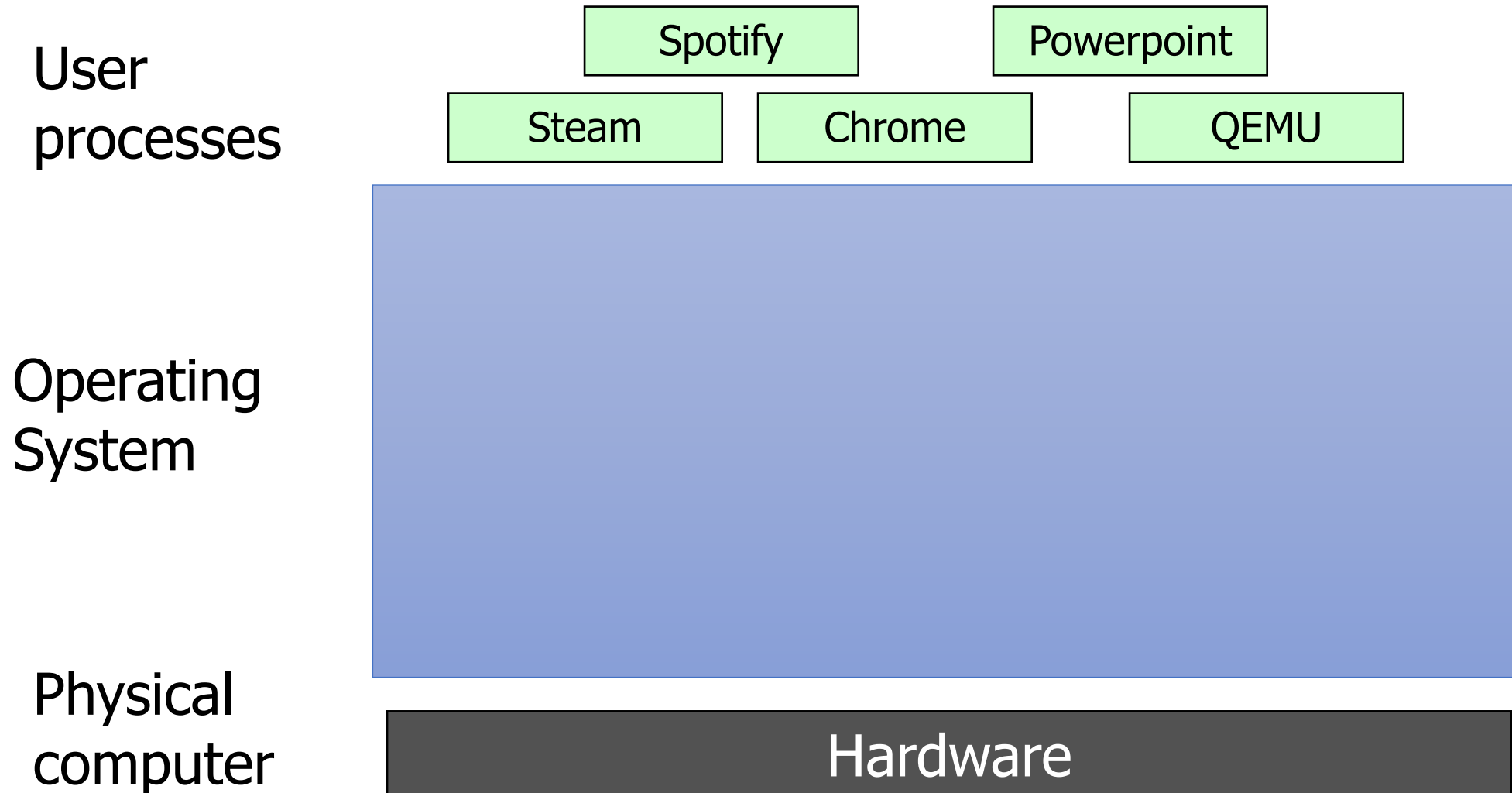
- OSes make advancing technology available to rapidly evolving applications. They do so with two major goals:
  1. Provide **abstractions** to applications to enable hardware compatibility
    - Why: allow reuse of common features, avoid low-level details
    - Challenges: What are the correct abstractions?
  2. Manage **sharing of resources** across many applications
    - Why: protect applications, enforce fair access
    - Challenges: What are the mechanisms and what are the policies?
- Good operating systems do these quickly, efficiently, and securely



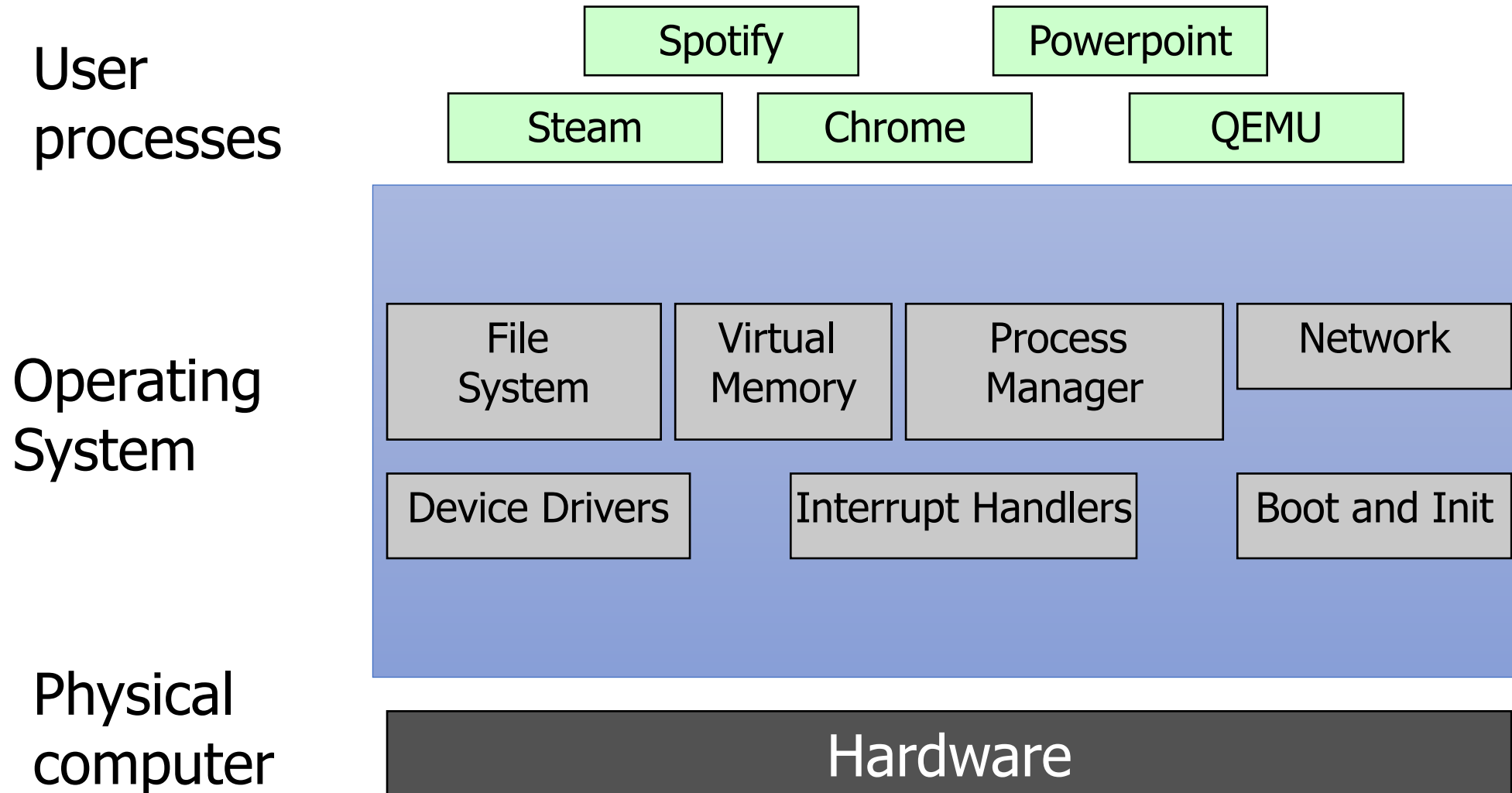
# What is an operating system?



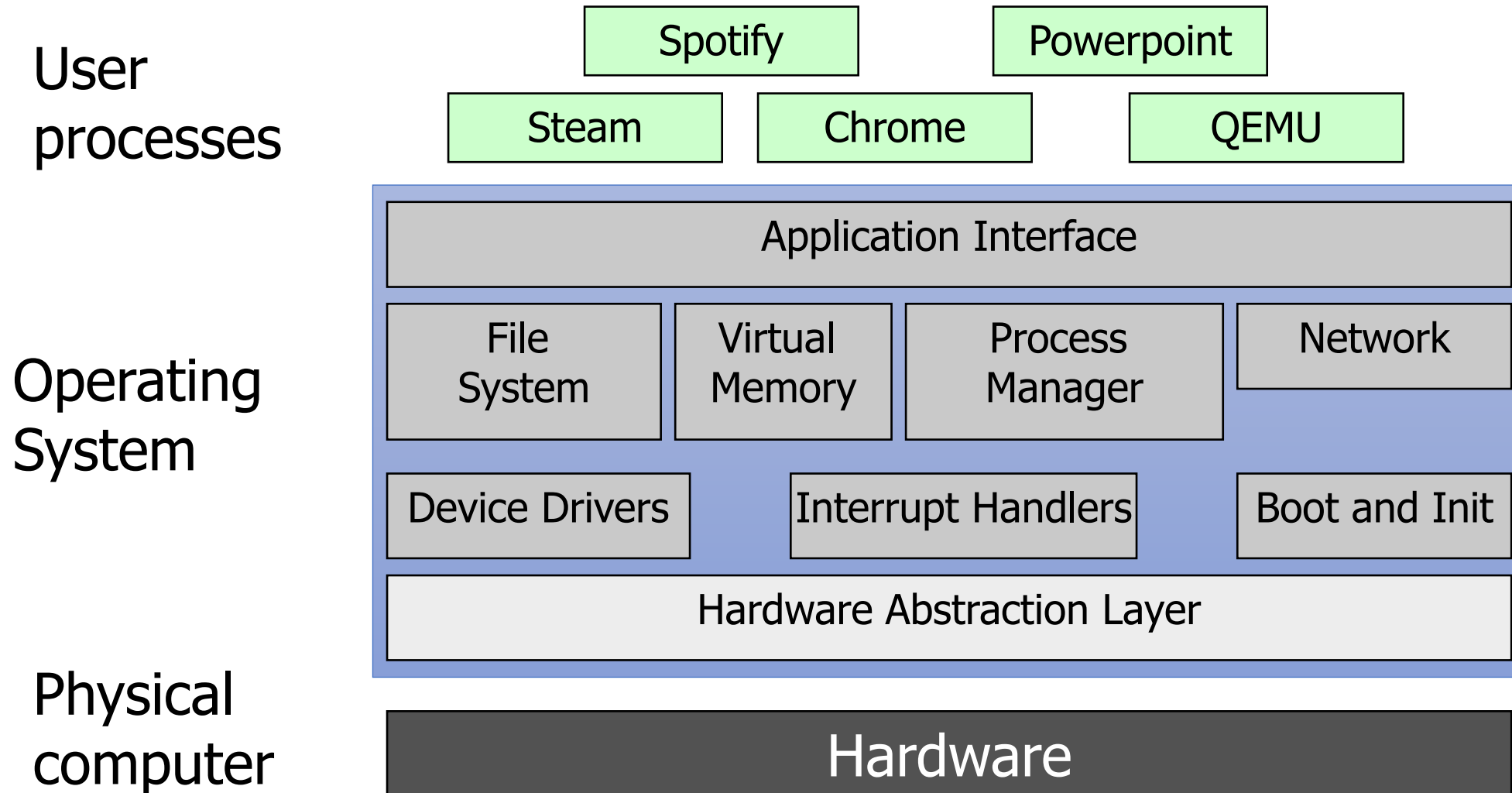
# What is an operating system?



# What is an operating system?



# What is an operating system?

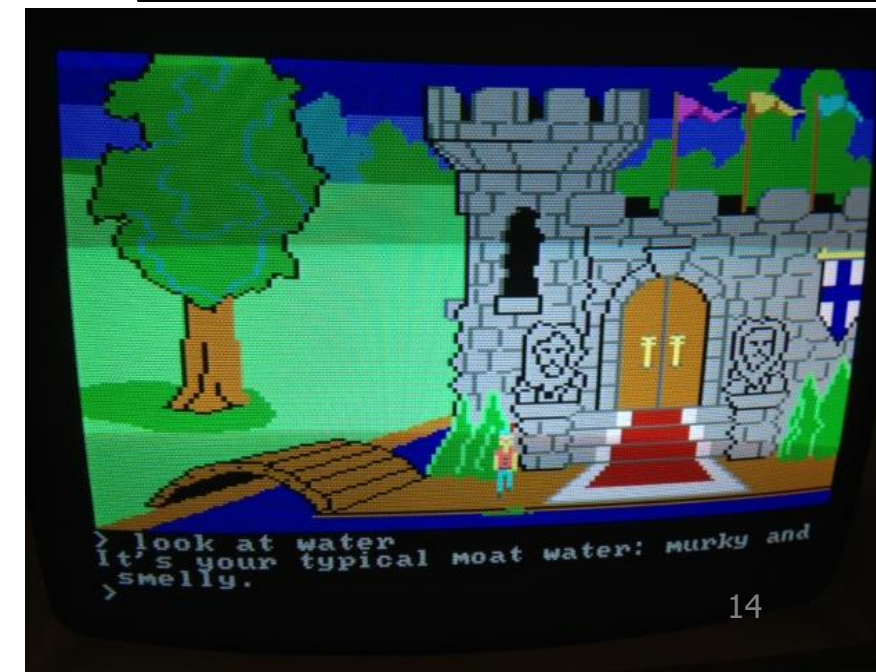


# What's part of the OS?

- **OS kernel** – the part of the OS code that has full control of the system
- Process scheduling (who uses CPU)
- Memory allocation (who uses RAM)
- Accesses hardware devices
  - Outputs graphics
  - Reads/writes to network
  - Read/write to disks
  - Handles boot-up and power-down
- **OS distribution** – the kernel + lots of other useful stuff
- GUI / Window manager
- Command shell
- Software package manager
  - “app store”, yum, apt, brew
- Common software libraries
- Useful apps:
  - Text editor, compilers, web browser, web server, SSH, anti-virus, file-sharing, media libraries,

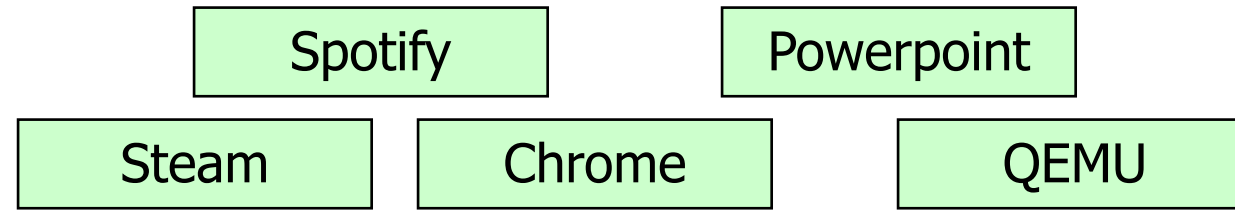
# Before operating systems

- User could only run one program at a time.
- Had to insert the program disk before booting the machine.
- Program had to control the hardware directly
  - This is a nuisance because hardware is complicated
  - Program will only be compatible with one set of hardware
- An example (at right): 1983 "King's Quest" game for IBM PC Jr.

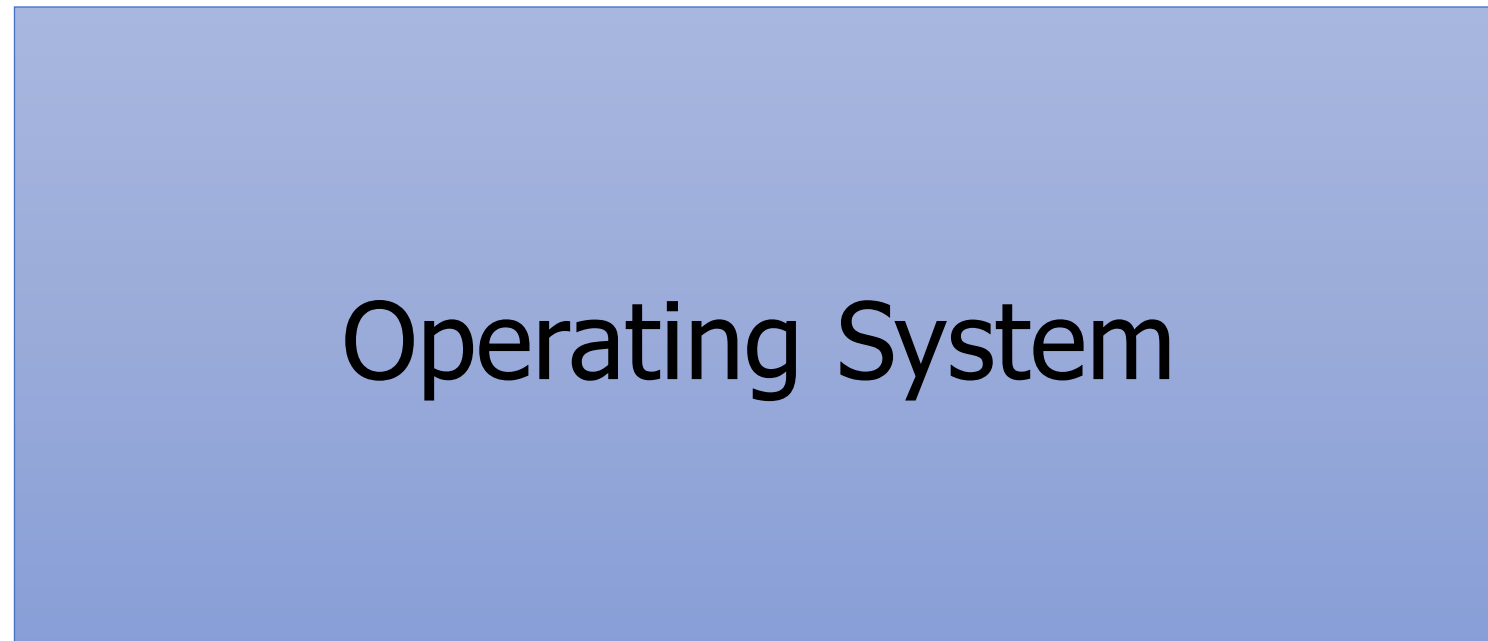


# Operating Systems are middle-management

Applications  
we care about



Overhead



Hardware to  
run things



# Embedded systems often still run without operating systems

- “Bare-metal” embedded systems
  - Many downsides:
    - Application must handle things for themselves
      - Boot and initialization
      - All hardware it wants to interact with
    - Applications are not portable
      - Rewrite, mostly from scratch, for new microcontroller
    - No malloc, no segfaults
      - Instead invalid memory accesses likely crash the whole system
      - Imagine if each CS211 segfault resulted in the EECS server rebooting...
- Upside: can be a VERY highly efficient system



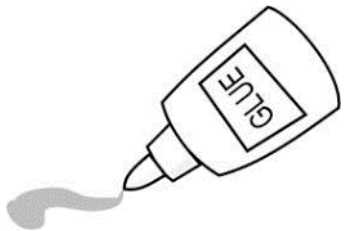
# Roles of an Operating System give the overhead value



- Referee
  - Manage protection, isolation, and sharing of resources
  - Resource allocation and communication



- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

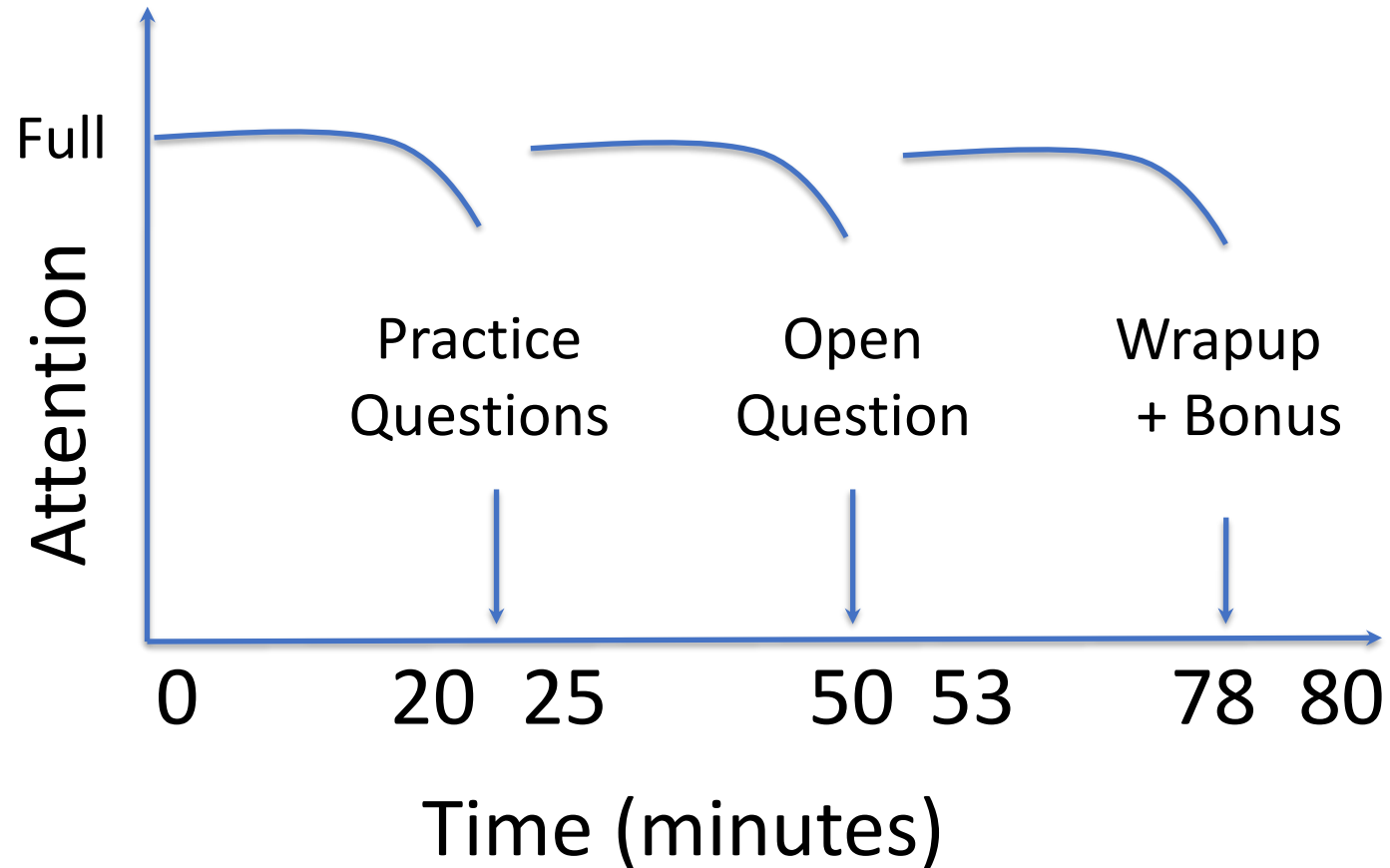


- Glue
  - Common services
    - Storage, Window system, Networking
  - Sharing, Authorization
  - Look and feel

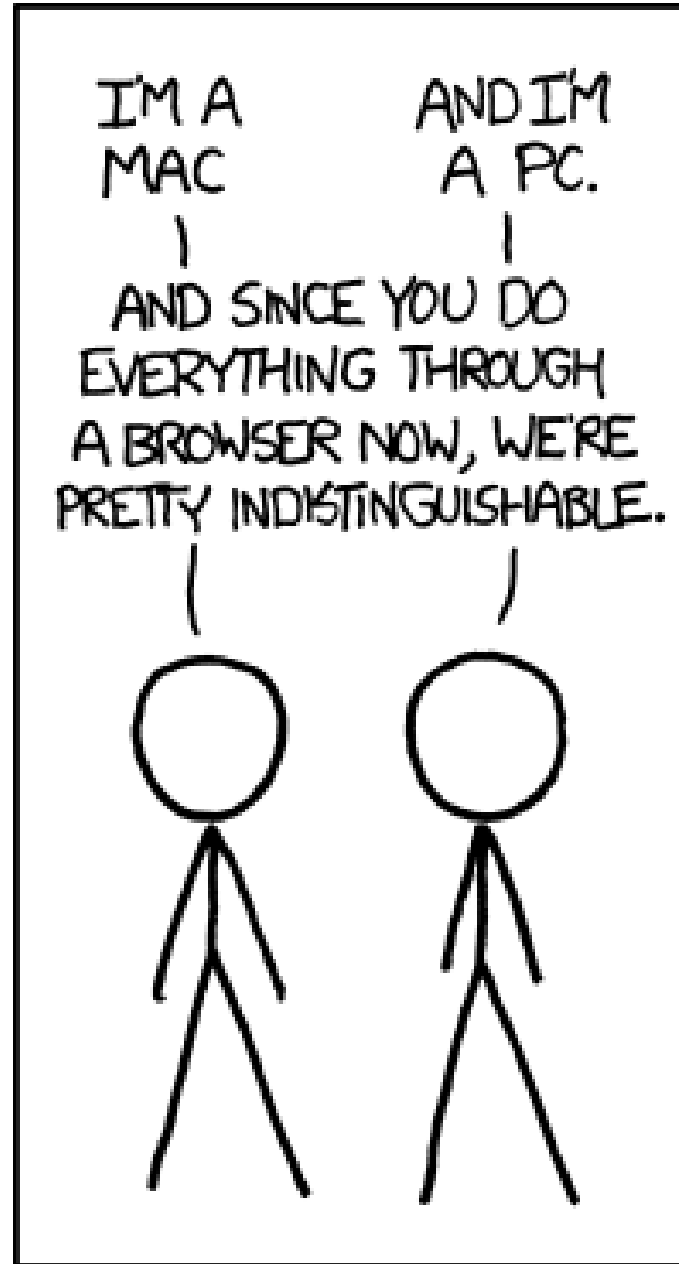
# Example: File Systems

- Referee
  - Prevent users from accessing other's files without permission
- Illusionist
  - Files can grow infinitely large
  - Where a file exists in memory or disk isn't important!
- Glue
  - Default file system types, named directories, file explorer

# Architecture of a lecture



# Break + xkcd



<https://xkcd.com/934/>

Open question:

Are modern web browsers  
basically operating systems?

# Outline

- What is an OS?
- **Logistics**
- Operating Systems History
- CS343 Focus

# Course staff

- Teaching Assistant
  - Michael Polinski
    - PhD student in systems
- Peer Mentors (10)
  - Eli Barlow
  - Emily Wei
  - Grigorii Kalin
  - Joseph Grantham
  - Justin Dong
  - Kellen Bryant
  - Kevin Hayes
  - Max Ward
  - Steve Ewald
  - Tim Sinaga

Their role: support student questions via office hours and Piazza

We'll have lots of weekly office hours blocks for you to get help from them

# Course Details – how to learn stuff

- Lectures: here Tuesdays and Thursdays
  - Provides background on materials
  - Attend and ask questions!
  - Panopto tab on Canvas should have best-effort recordings (a few hours later)
  - I also post the slides right before class to the Canvas homepage
- Textbook:
  - Modern Operating Systems (4<sup>th</sup> Edition), Tanenbaum and Bos
  - Very useful reference. Lecture will be relatively in sync with it
  - Other references are in the syllabus
- Office Hours: will start next week

# Asking questions

- Class and office hours are always an option!
  - I usually have time after class to answer a bunch of questions
  - Right after a break is a great time for questions too
- Piazza: (similar to Campuswire)
  - Post questions
  - Answer each other's questions
  - Find lab partners
  - Information from the course staff
  - Post private info just to course staff
- Please do not email me! Post to Piazza instead!
  - Let me know if you don't have access. I'll update the roster



# Course grade

- 20% Midterm (first half of the course)
- 20% Final (second half of the course)
- 60% Labs
  - 05% Getting Started Lab (individual)
  - 15% Queuing/Scheduling Lab
  - 10% Producer-Consumer Lab
  - 15% Device Driver Lab
  - 15% Paging Lab
- Exact number to letter mapping is a little flexible
  - But this course is NOT curved
  - Only flexible in your favor

# Midterm exams

- Test on your knowledge of course material (like a 213 exam)
  - In-person, on paper
  - I'll allow a notes sheet
- Not cumulative. Two midterms on two halves of the class
- First midterm will be during class time: April 25<sup>th</sup> (Thursday)
  - Week before the drop deadline
- Second midterm will be during exam week: June 5<sup>th</sup> (Wednesday)
  - Make sure you can be here in-person for it!

# Labs

- These are a significant amount of the learning in this class
  - Hands-on experience with the topics we're talking about
  - Labs primarily involve written code in C
  - Can be quite a bit of work
- Work on these in groups of up to three students
  - Preferably two or three
  - Goal: collaboration, not splitting labs
    - If you don't work on it, you're not going to learn from it
    - Pair programming more often results in code written right the first time

# Lab logistics

- Getting Started Lab
  - Learn how everything works
- Queuing/Scheduling Lab
  - OS application scheduling
- Producer-Consumer Lab
  - Concurrency and locks
- Device Driver Lab
  - Driver for a GPU
- Paging Lab
  - Memory management
- Getting started lab is special
  - One week deadline (due 04/04)
  - Must do alone
  - All-or-nothing grading
- Normally teams of 2 or 3 students
  - Find partners now!
- We'll put out a survey soon for those who don't know anyone

# Late policy

- You can submit labs late
- 20% penalty to maximum grade per day late
  - Example: three days late means maximum grade is 40%
- We will be flexible with deadlines for problems outside of your control
  - Sick, family emergency, broken computer
  - Contact me! (via Piazza)

# Slip days

- Slip days let you turn in a homework late and receive no penalty
- Each student gets **3 slip days**
  - Apply to **labs**
  - You don't need to tell us you're using them, we'll just automatically apply them at the end of the year
  - Be sure to coordinate about them on partner assignments
- Examples:
  - Turn in Scheduling Lab three days late
  - Turn in Scheduling Lab two days late and Paging lab one day late
  - Turn in Paging Lab five days late with only a two-day penalty

# Academic integrity

- This is something I take very seriously
- Collaboration good; plagiarism bad
  - You should know where that line is, and be nowhere near it
  - When in doubt, ask the instructor *before* you do something you're not sure about
- At no point should you see someone else's solutions
  - Not your colleagues', not your friends', not your cousin's, not something you found online
- I report everything suspicious to the dean

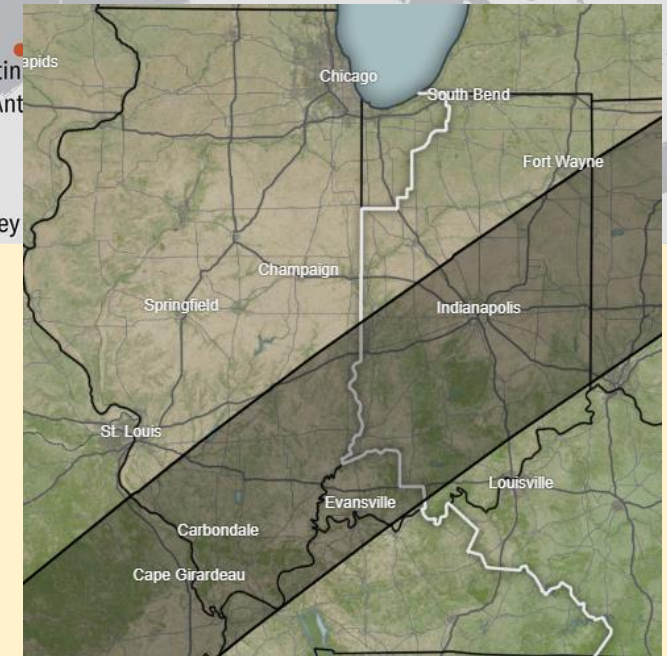
# Expectations

- Give yourself time to complete labs
  - Dealing with C code
  - Handling a large code base
  - Dealing with concurrency!!
  - You'll learn a lot through the challenge
- Don't fall behind on lecture materials
  - Material builds on itself, like in CS213
- Use course staff to help you out
  - Office hours & Piazza are for your benefit



# Great American Total Solar Eclipse

- Roughly 2:00 pm Central in southern Illinois/Indiana
  - 3.5 minutes of total solar eclipse
- Get yourself to 100% eclipse
  - 99% was “oh that’s kind of neat”
  - 100% was “the coolest thing I’ve ever seen in my entire life”
- Skip class, find a car, go see it (4-5 hour drive)
  - Next opportunity is 2044



<https://science.nasa.gov/eclipses/future-eclipses/eclipse-2024/where-when/>

# Break + First Tasks

## 1. Getting Started Lab

- Makes sure you've got everything set up to do all the labs
- Should be available right now
- Get this done on time

## 2. Find partner(s) for assignments

- We'll put out a form in the next few days if you don't know people in the class

## 3. Take a break, chat with your neighbors, look at your phone, reset your brain for a minute

# Outline

- What is an OS?
- Logistics
- **Operating Systems History**
- CS343 Focus

# Computer History

- Check out the textbook!
  - In-depth history
  - Entertaining writing with *just* the right amount of sarcasm
- This isn't a computer history course
  - But there is a good reason to understand the lineage of the techniques we explore in this course

# Early evolution of computing systems – Batch

- 1955: Batch systems
  - Collect a bunch of program punch cards and write them all one magnetic tape.
  - Run the tape through the mainframe to execute all the jobs in sequence.
- OS responsibility
  - Libraries for managing Input/Output (I/O)
- Problems
  - I/O is VERY slow. 80-90% of total time just waiting.



# Early evolution of computing systems – Multiprogramming

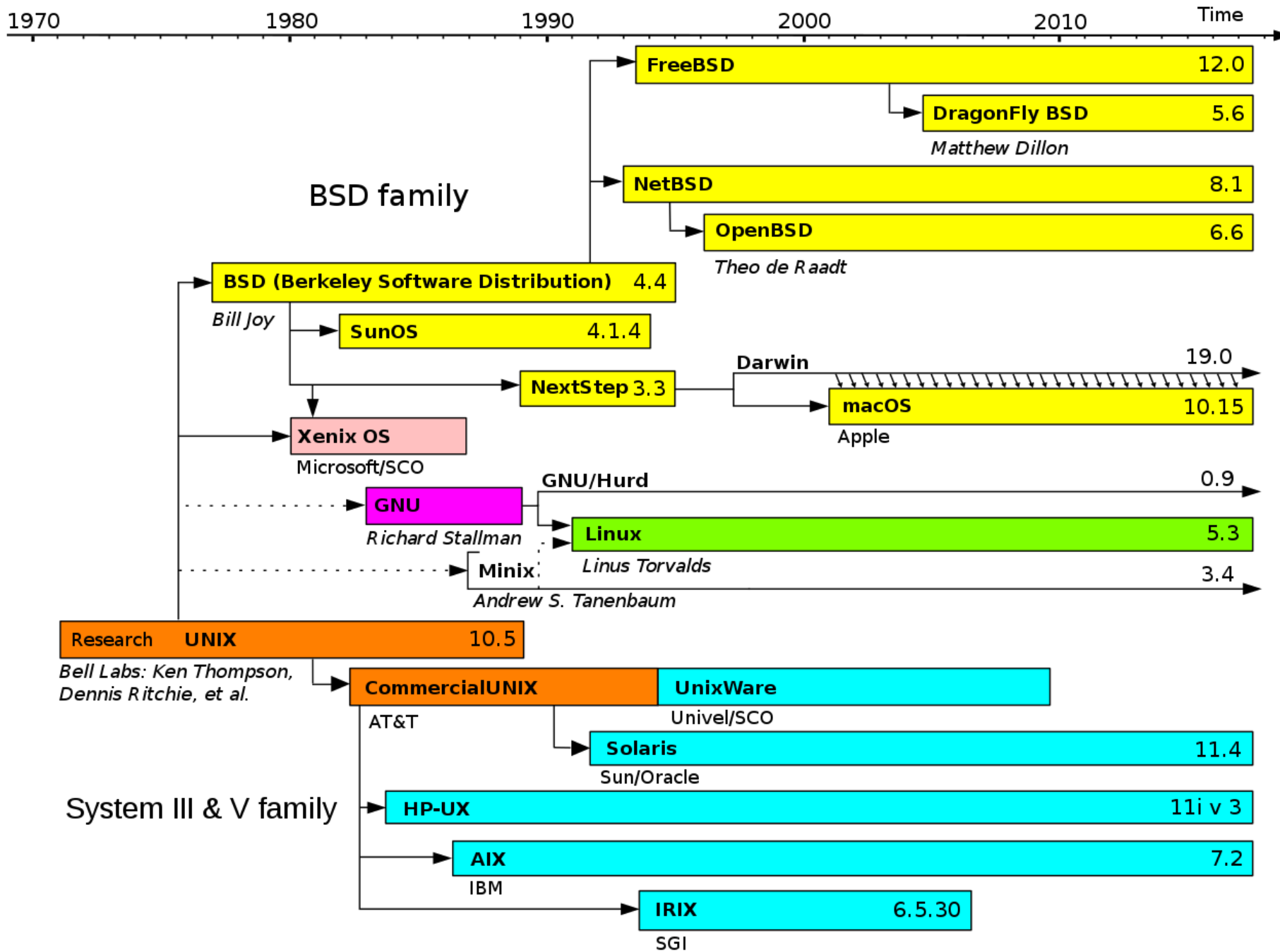
- 1960s: Multiprogramming (IBM OS/360)
  - Keep multiple runnable jobs in memory at once.
  - Allows overlap I/O of one job with computing of another.
    - Uses asynchronous I/O and interrupts or polling to detect I/O completion
- OS responsibility
  - Schedule jobs
  - Monitor I/O
- Problems
  - Still need to submit all jobs in advance



# Early evolution of computing systems – Timesharing

- 1960s-70s: Timesharing (MULTICS, Unix)
  - Multiple user terminals connected to one machine
  - Allows *interactive* use of machine to be efficient (because another user's job can run while you're thinking).
- OS responsibility
  - Multiple users (with permissions!)
  - Scheduling processes
  - Application interface
  - Shell tools





# Simplified History of Unix-like Operating Systems

Operating systems are very interconnected



# Later evolution of computer systems – PC

- 1980s-00s: Personal Computers (IBM PC, Macintosh)
  - Graphical user interfaces were developed
  - Mainframe OS concepts (like networking) were applied to PCs
  - Magnetic disk (hard drive) capacity becomes huge, but still slow
- OS responsibility
  - Look and feel of a system, particularly for non-experts
  - Tools that were distributed with the OS had significant business results
    - Computers are bought for Excel or for Lotus 1-2-3

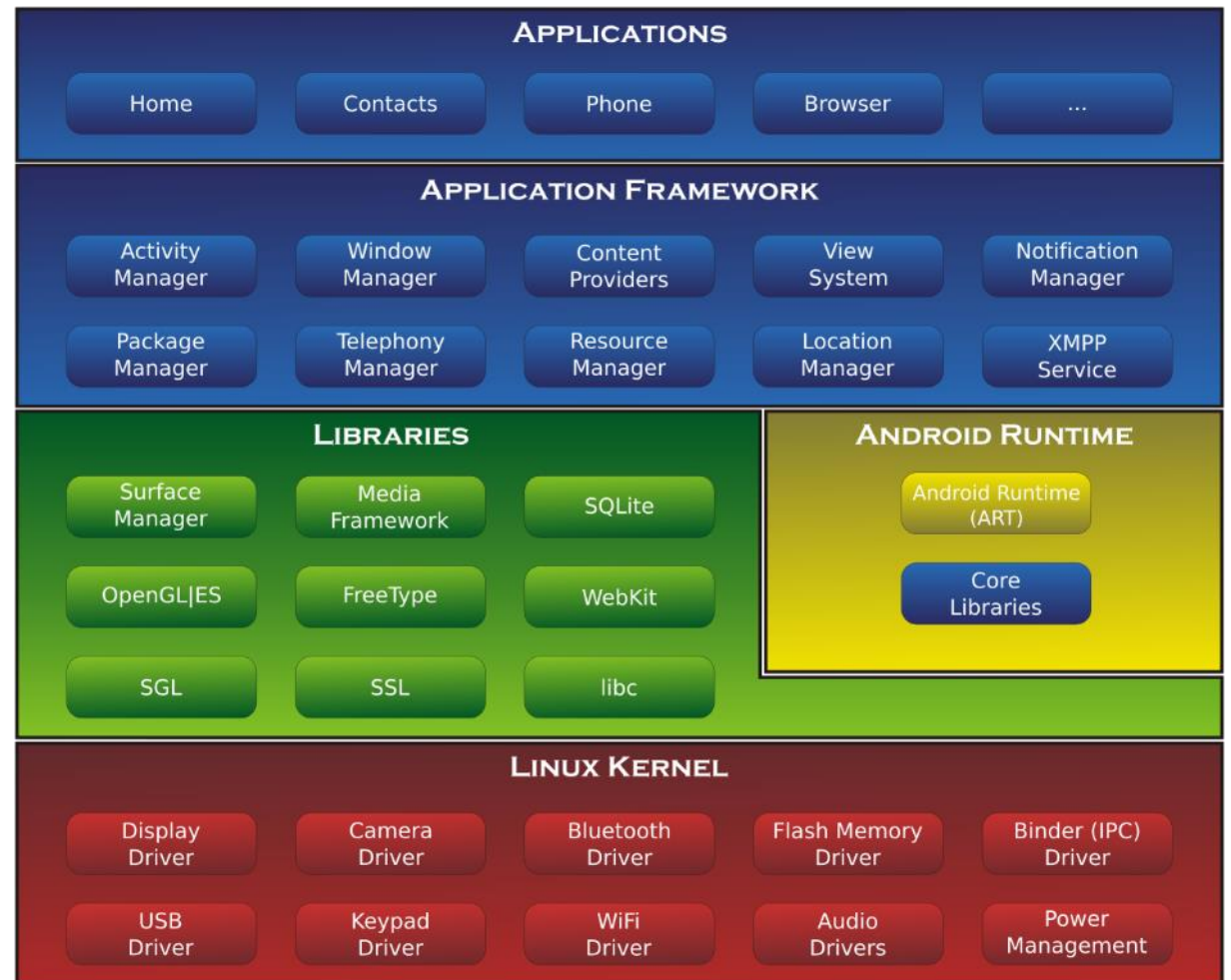
# Later evolution of computer systems – Mobile and Cloud

- 2000s-20s: Mobile and pervasive computing, Cloud Computing
  - Slow hardware is once again common (phones & wearables)
  - OS manages sensitive information like location and internet behavior
  - Fast flash storage is common.
  - Server hardware is shared by many different cloud computing customers
- OS responsibility
  - Diverse hardware drivers
  - Security
  - Massive parallelism

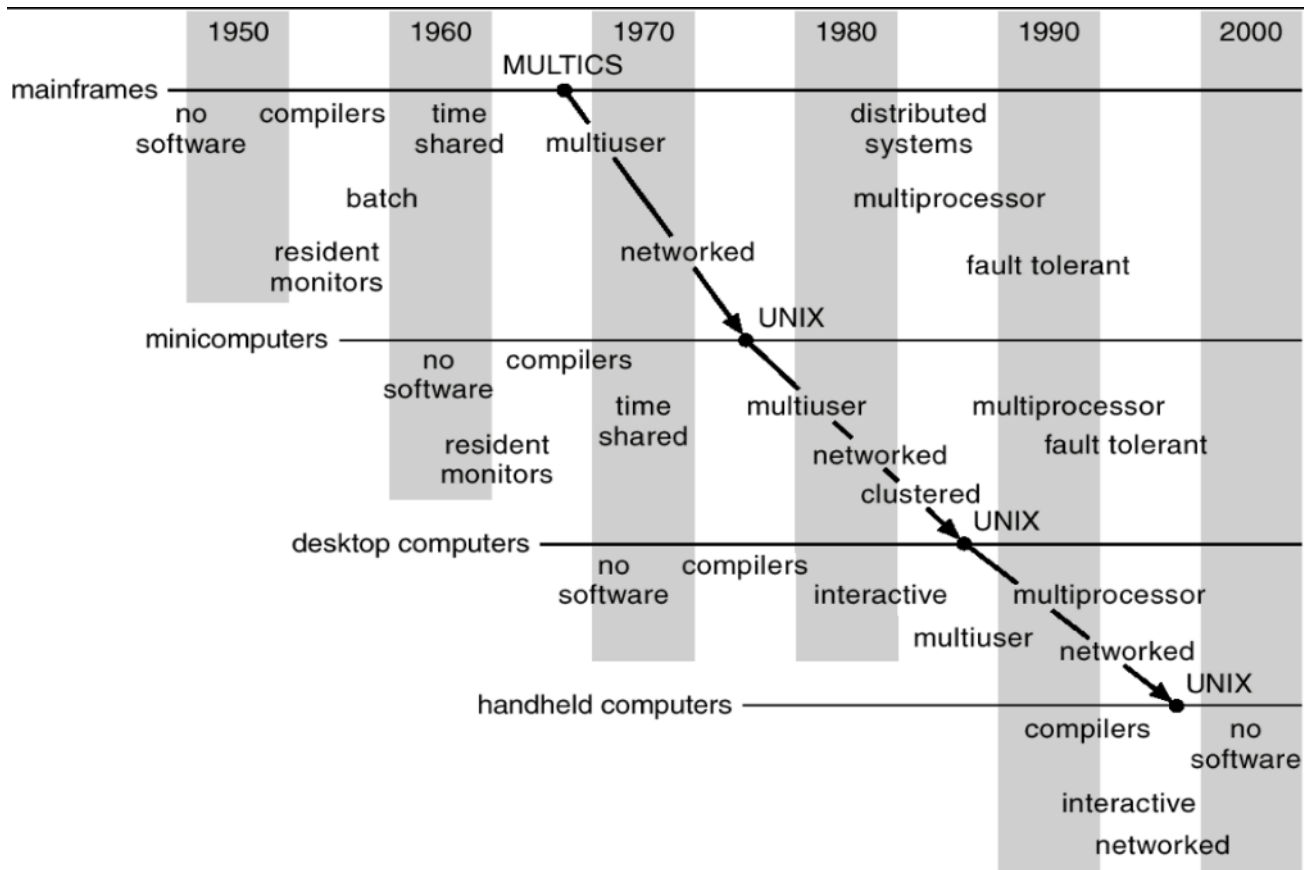


# An example: Android Operating System

- Kernel - Linux
  - With modifications particularly in power management
  - And additional drivers
- Distribution
  - Look and feel of “Android”
  - App framework
  - Some of this changes per vendor (Samsung vs Google)



# Operating systems have evolved with hardware in a cycle



- Sophisticated operating systems first arose on mainframes.
- OS ideas migrated to smaller machines as those machines became more powerful.
- In 2024, a **smart watch** has 1 GB RAM, 32 GB SSD storage, two CPU cores, and a real OS.

# Some future OS directions

1. Manage increasingly specialized hardware
  - Post-Moore's law, general-purpose CPUs lose out to special-purpose chips
  - OS must maintain abstractions while enabling capabilities
2. Energy as another resource
  - Already considered in laptop/smartphone worlds
  - Increasingly important to data center operations as well
3. Very small-scale, ubiquitous devices
  - Computers are becoming part of everything around us
  - How do we develop applications for those devices and coordinate them?

# Outline

- What is an OS?
- Logistics
- Operating Systems History
- **CS343 Focus**

# Schedule for first half of the course

## 1. Scheduling

- Managing CPU utilization
- Workload, Queuing, Real-time

## 2. Concurrency

- Dealing with the realities of modern-day computing
- Sources, Control, Challenges

# Schedule for second half of the course

## 3. Device Drivers

- Management and abstraction of devices
- Interrupts, DMA, Abstractions

## 4. Virtual Memory

- Management and abstraction of memory
- Paging, Allocation, Security

## 5. File Systems

- Management and abstraction of data
- Principles, Examples



# Why do we care about OS?

- Performance
  - Speed is influenced by
    - Parallelism, resource contention, memory management
    - Generally OS overhead
- Security
  - Process and data isolation when actually all running together
  - The biggest security vulnerabilities break abstractions
    - Meltdown and Spectre
- Systems Design and Implementation
  - Maybe you won't ever build an OS (maybe you will!)
  - But you might work on other large, backend systems that will deal with similar concerns. This is good practice!

# Outline

- What is an OS?
- Logistics
- Operating Systems History
- CS343 Focus