ECE 350 Real-time Operating Systems

Lecture 1: Introduction

Prof. Seyed Majid Zahedi

https://ece.uwaterloo.ca/~smzahedi

Outline

- How do things work in ECE 350?
- What is an operating system?
- What makes operating systems so exciting?

Class is Entirely Online!

- Lectures will be on Teams
  - Links will be provided on LEARN
  - Recordings will be available afterwards

- Office hours will be on Teams
  - Links will be provided on LEARN
  - There will be no recordings
  - No office hours during week 1 and reading week

- Lab tutorials will be on delivered asynchronously

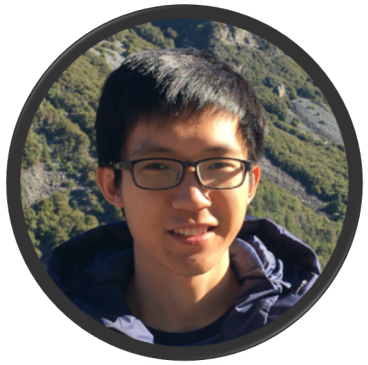
Who Are We?

Instructor: Prof. Seyed Majid Zahedi
zahedi@uwaterloo.ca
https://ece.uwaterloo.ca/~smzahedi
Office hours: M&F 14:45 to 15:45 (EST)

Lab instructor: Irene Huang
yqhuang@uwaterloo.ca
Office hours: Tu&Th 11:30 to 12:30,
F 16:30 to 17:30 (EST)

ECE 350 TAs

Ali Hossein Abbasi Abyaneh
a36hosse@uwaterloo.ca
Office hours: W 11:30 to 12:30 (EST)



Maizi Liao
m7liao@uwaterloo.ca

Weitian Xing
w23xing@uwaterloo.ca
Office hours: M 16:30 to 17:30 (EST)

Useful Links

- Course webpage

  https://ece.uwaterloo.ca/~smzahedi/crs/ece350/

- Course on Piazza

  https://piazza.com/uwaterloo.ca/fall2020/ece350

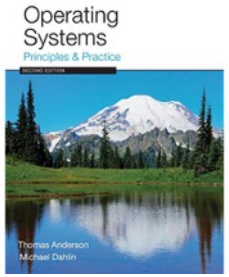
- Anonymous feedback form

  https://forms.gle/Ea7kbQhVWLJbphfJA

Readings

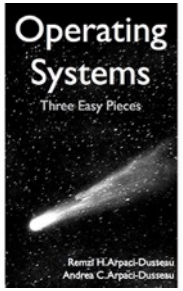
- Main textbook

  Operating Systems: Principles and Practice (2nd Edition)

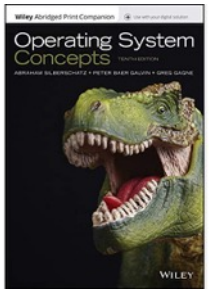


- Optional references

  Operating Systems: Three Easy Pieces (Freely Available)



  Operating System Concepts (10th Edition)



Prerequisite: ECE 252

- It is assumed that you know
  - File system API
    - File operations, directory structure
  - Processes and threads
    - PCB, TCB, process/thread life cycle, fork and exec, signals
  - Inter-process communication
    - File sharing, memory sharing, message passing, pipes
  - Networking
    - Sockets, ports, client and server programs
  - Concurrency
    - Mutual exclusion, mutex, semaphores, locks, condition variables, monitors
  - Deadlock
    - Avoidance, detection, recovery, Banker's algorithm

ECE 350 Is a Class About…

- Design of key systems abstractions that have emerged over time
  - Processes, threads, events, address spaces, file systems, sockets, transactions, key-value stores, etc.
- Tradeoffs surrounding these designs
- Their efficient implementation
  - Including hardware support that makes them possible and practical
- And how to use them effectively

Why Take ECE 350?
Why Learn About OS?

- Some of you will design and build parts of (real-time) operating systems
- Many of you will create systems that use OS concepts
  - Whether you build hardware or software
  - Concepts and design patterns appear at many levels
- All of you will write programs that use OS abstractions
  - The better you understand them, the better you use them

Evaluation

- Lab project: 70%
  - 5 deliverables (more on this later)
- Quizzes: 30%
  - 10-12 quizzes
  - All online on LEARN
  - Quizzes are open book
    - You may consult your textbook, course notes, and materials posted on course webpage
    - Use of any other resource (including online services such as stackexchange.com) is prohibited
    - You may not communicate directly or indirectly with any person except course instructors

Important disclaimer

We reserve the right to change these weights at any time during the term without prior notice

Lab Project

- You will design, implement, and test real-time executive (RTX) on Keil MCB1700 Cortex-M3 board

Groups

- Groups should have 4 members
  - Never 5, 3 requires serious justification
  - Signup in LEARN by 08:30 AM on Sep.14th EST
- Each group must have a project manager
  - Responsible for coordinating group
  - Can rotate between teammates
- Only one split-up is allowed
  - One-week notice in writing before nearest deadline
  - All students involved lose one grace day

Milestones

Deliverable	Weight	Due Date
P0 Group sign-up	2%	08:30 Sept 14
P1 Memory management	18%	20:30 Sept 25
P2 Task management	20%	20:30 Oct 09
P3 Synchronization & console I/O	25%	20:30 Nov 02
P4 Timing and real-time scheduling	20%	20:30 Nov 16
P5 Memory protection & stress tests	15%	20:30 Nov 30

All times are Eastern Standard Time

Important disclaimer

We reserve the right to change these weights at any time during the term without prior notice

Grades for P1 – P5

- Teammates anonymously evaluate each other for each deliverable

Peer rating	Contribution factor
[8, 10]	1
[7,8)	0.8
[6,7)	0.6
[4,6)	0.4
[0,4)	0

- Your grade = Your group's grade x your contribution factor

Start Early!

- Time/work estimation is hard
  - Programmers are eternal optimists (it will only take two days)!
    - This is why we bug you about starting the project early

- Can a project be efficiently partitioned?
  - Partitionable task decreases in time as you add people
  - But … what about communication?
    - Time reaches a minimum bound
    - With complex interactions, time increases!

Techniques for Partitioning Tasks

- Functional
  - Person A implements threads, Person B implements semaphores, Person C implements locks…
  - Problem: Lots of communication across APIs
    - If B changes the API, A may need to make changes
- Task
  - Person A designs, Person B writes code, Person C tests
  - May be difficult to find right balance, but can focus on each person's strengths (Theory vs systems hacker)
  - Since debugging is hard, Microsoft has two testers for each programmer

Communication

- More people means more communication
  - Changes have to be propagated to more people
  - Think about person writing code for most fundamental component of system: everyone depends on them!
- Miscommunication is common
  - ***"Index starts at 0? I thought you said 1!"***
- Who makes decisions?
  - Individual decisions are fast but trouble
  - Group decisions take time
  - Centralized decisions require a big picture view (someone who can be the "system architect")
- Often designating someone as system architect can be a good thing
  - Better not be clueless
  - Better have good people skills
  - Better let other people do work

Coordination

- Many are in different time zones ⇒ some cannot make all meetings!
  - They miss decisions and associated discussion
  - Why do we limit groups to 4 people?
    - You would never be able to schedule meetings otherwise
  - Why do we require 3 people minimum?
    - You need to experience groups to get ready for real world

- People have different work styles
  - Some people work in the morning, some at night
  - How do you decide when to meet or work together?
- What about project slippage?
  - Everyone busy but not talking, one is way behind, but no one will know until very end!
- Hard to add people to existing group
  - Members have already figured out how to work together

How to Make it Work?

- People are human … get over it!
  - People will make mistakes, miss meetings, miss deadlines, etc.
  - You need to live with it and adapt
  - It is better to anticipate problems than clean up afterwards
- Document, document, document
  - Why Document?
    - Expose decisions and communicate to others
    - Easier to spot mistakes early
    - Easier to estimate progress
  - What to document?
    - Everything (but don't overwhelm people or no one will read)

Suggested Documents for You to Maintain

- Project objectives: goals, constraints, and priorities
- Specifications
  - This should be the first document generated and the last one finished
- Meeting notes
  - Document all decisions
- Schedule
  - This document is critical!
- Organizational chart
  - Who is responsible for what task?

Use Software Tools

- Source revision control software (CVS, SVN, git)
  - Easy to go back and see history
  - Figure out where and why bugs got introduced
  - Communicates changes to everyone (use RCS's features)
- Use automated testing tools
  - Write scripts for non-interactive software
- Use E-mail and instant messaging consistently to leave history trail

Test Continuously

- Integration tests all the time, not at 8pm on due date!
  - Write dummy stubs with simple functionality
  - Schedule periodic integration tests
    - Get everyone code, build, and test … don't wait until it is too late!
- Testing types
  - Unit tests: white-/black-box check each module in isolation
  - Daemons: subject code to exceptional cases
  - Random testing: subject code to random timing changes
- Test early, test later, test again
  - What if something changes in some other part of code?

Late Submissions

- Five grace days (including weekends) without penalty
- 15% per day late submission penalty afterwards
  - 1-hour-late submission = 15-hour-late submission
- Lat submissions are not accepted after three days

Collaboration Policy

- Explaining concepts to someone in another group
- Discussing algorithms/testing strategies with other groups
- Helping debug someone else's code (in another group)
- Searching online for generic algorithms (e.g., hash table)

- Sharing code or test cases with another group
- Open-sourcing code (e.g., on GitHub) even after this term
- Copying OR reading another group's code or test cases
- Copying OR reading online code or test cases from prior years

- Zero tolerance policy for plagiarism
  - We use Moss and follow UW Policy 71 for any single incident

Lab Facilities

- No scheduled lab sessions
  - Tentative project demos: P3, P4, and P5
- Englab > ECE > ece-rtos remote desktop
  - Nexus computers
  - Keil MCB1700 LPC1768 (Cortex-M3) boards
  - MDK-ARM MDK-Lite ed. V5.x (32KB code size limit)
    - ARM compilation tools are included
    - Simulator is good for development work at home

⚠ Simulator may not perfectly match hardware behaviour
⚠ Test your code on hardware well before deadlines!
⚠ Code that only works on simulator loses 15%

Seeking Help

- Lab Q&A on Piazza discussion forum
  - Looking for group partners
  - Lab/Project administration
  - Keil IDE Q&A
  - Project Q&A
  - Target response time: one business day
  - Do not wait till the last minute to ask questions
- Individual emails
  - Only for questions containing confidential information
- Office hours
- Appointment

Important Near-term Task

Signup for project groups in Learn by

08:30 am on September 14th , 2020 EST

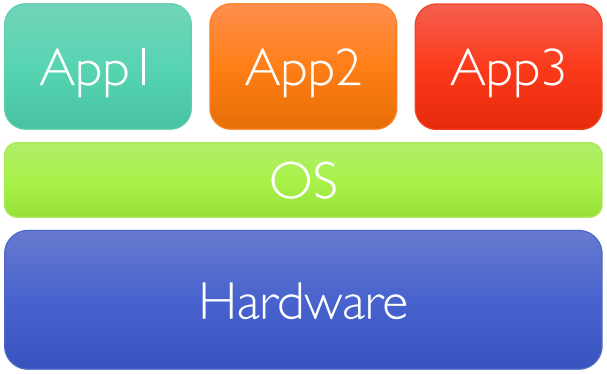

WORK HARD
RELAX HARDER
makeameme.org

What is an Operating System?

- No universally accepted definition
- "Everything vendors ship when you order OS" is good approximation, but varies wildly
- "The one program running at all times on computer" is kernel
  - Everything else is either system program (ships with OS) or application program

What is an Operating System? (cont.)

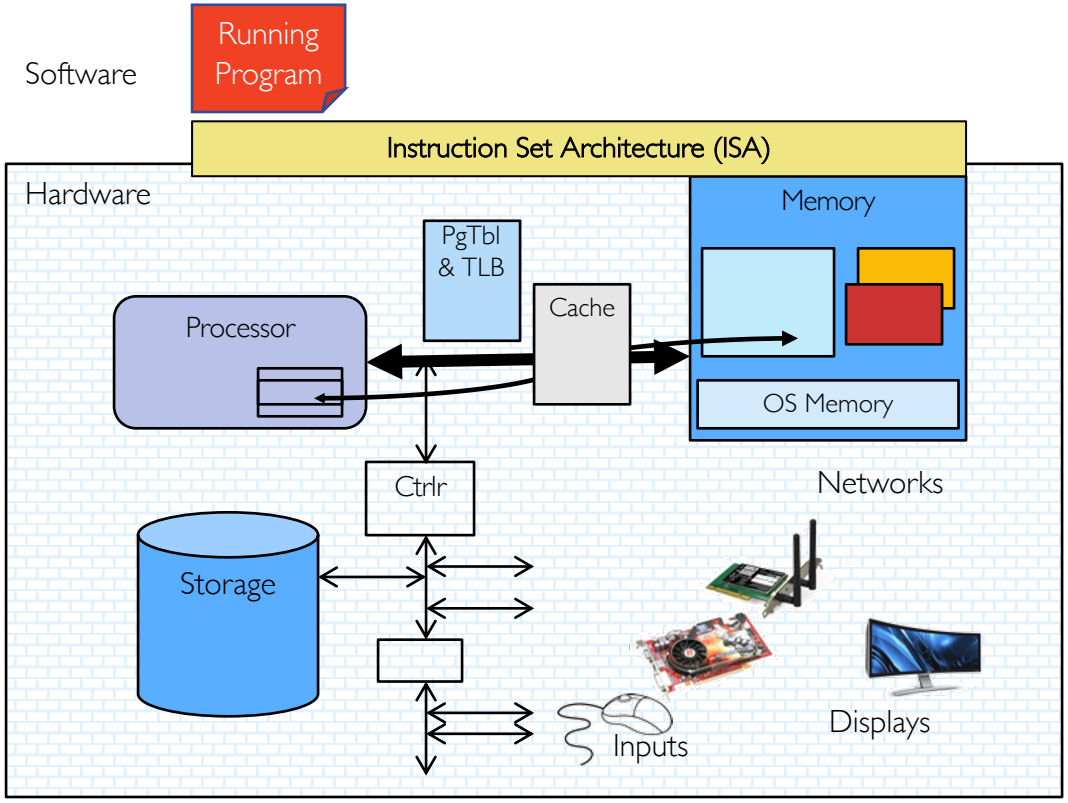
- Special layer of software that provides applications access to hardware resources
  - Abstract view of complex hardware devices
  - Protected access to shared resources
  - Security and authentication
  - Communication amongst logical entities



What is an Operating System? (cont.)

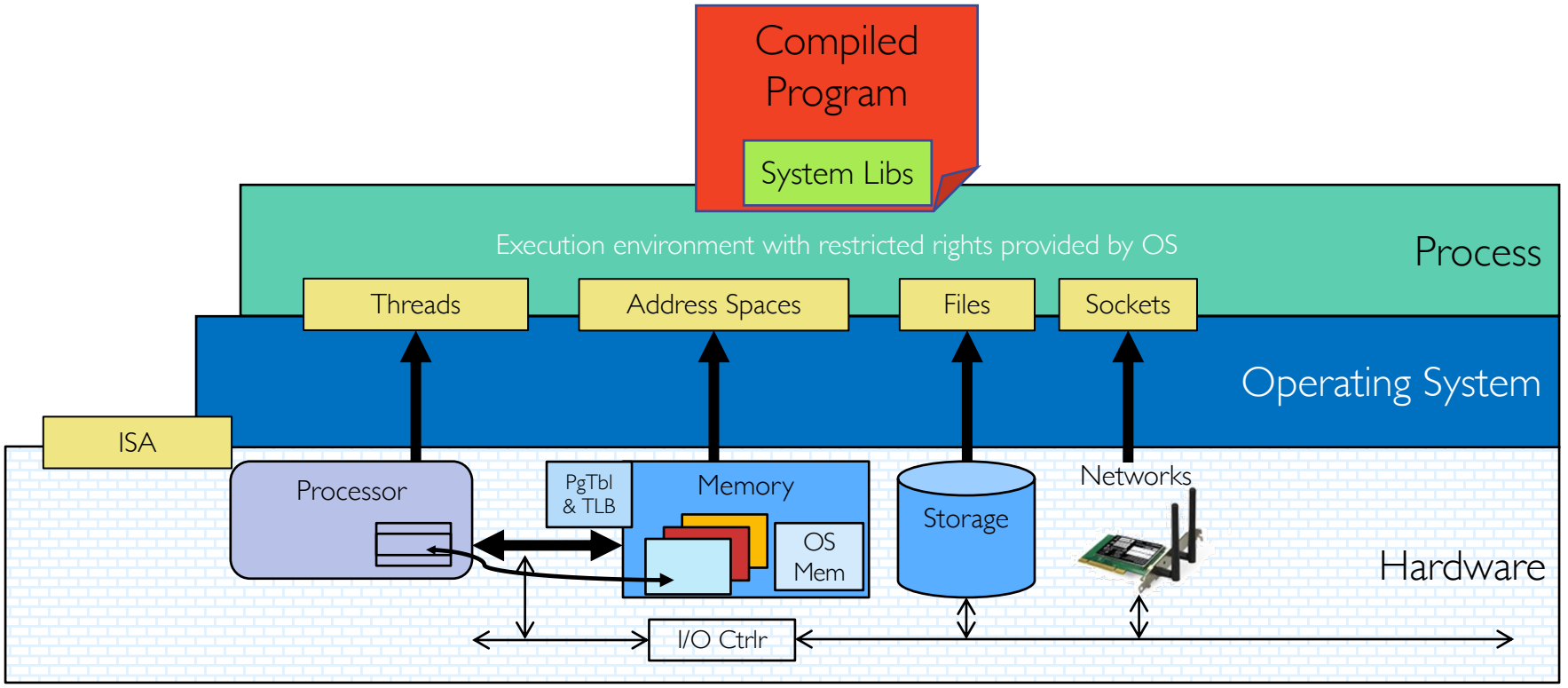
- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

Hardware/Software Interface



- ECE 222 and ECE 320: Machine structures (and C)
- OS *abstracts* these hardware details from the application

OS Basics: Virtualizing Hardware

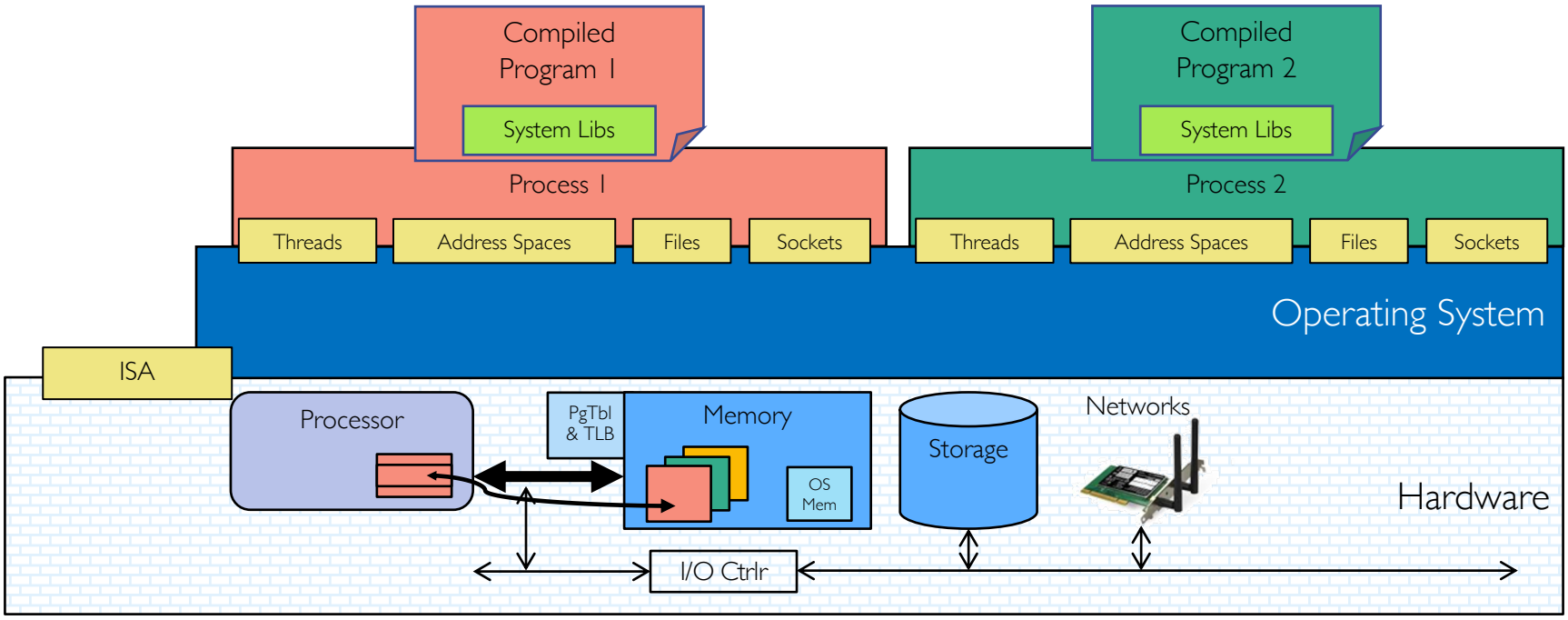


What is an Operating System? (cont.)

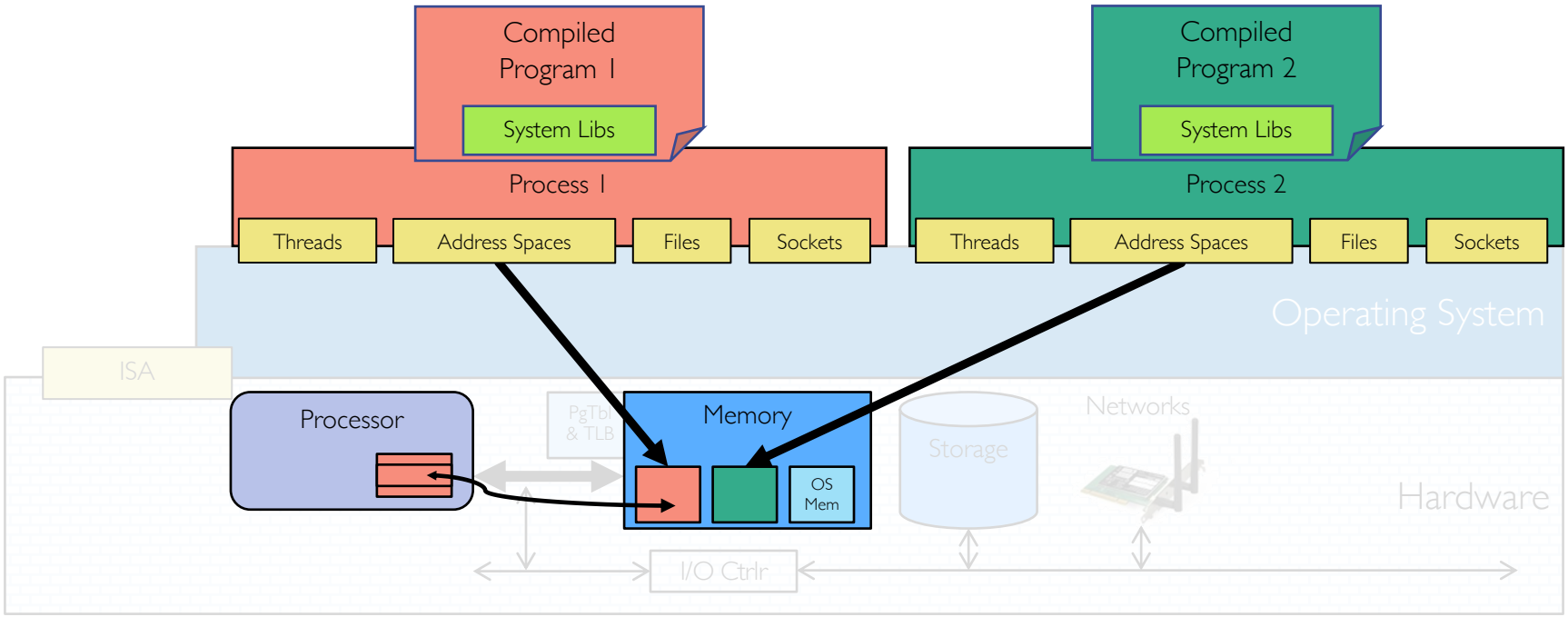
- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

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

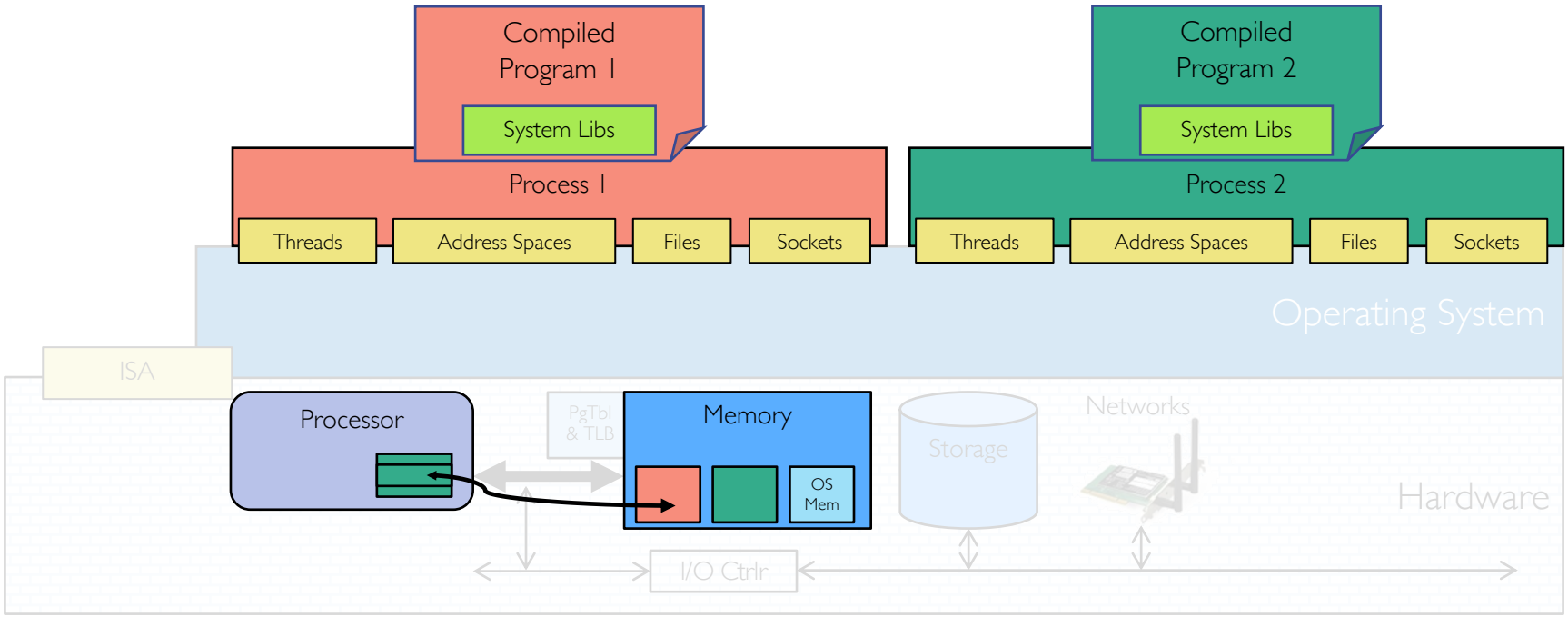
OS Basics: Switching Processes



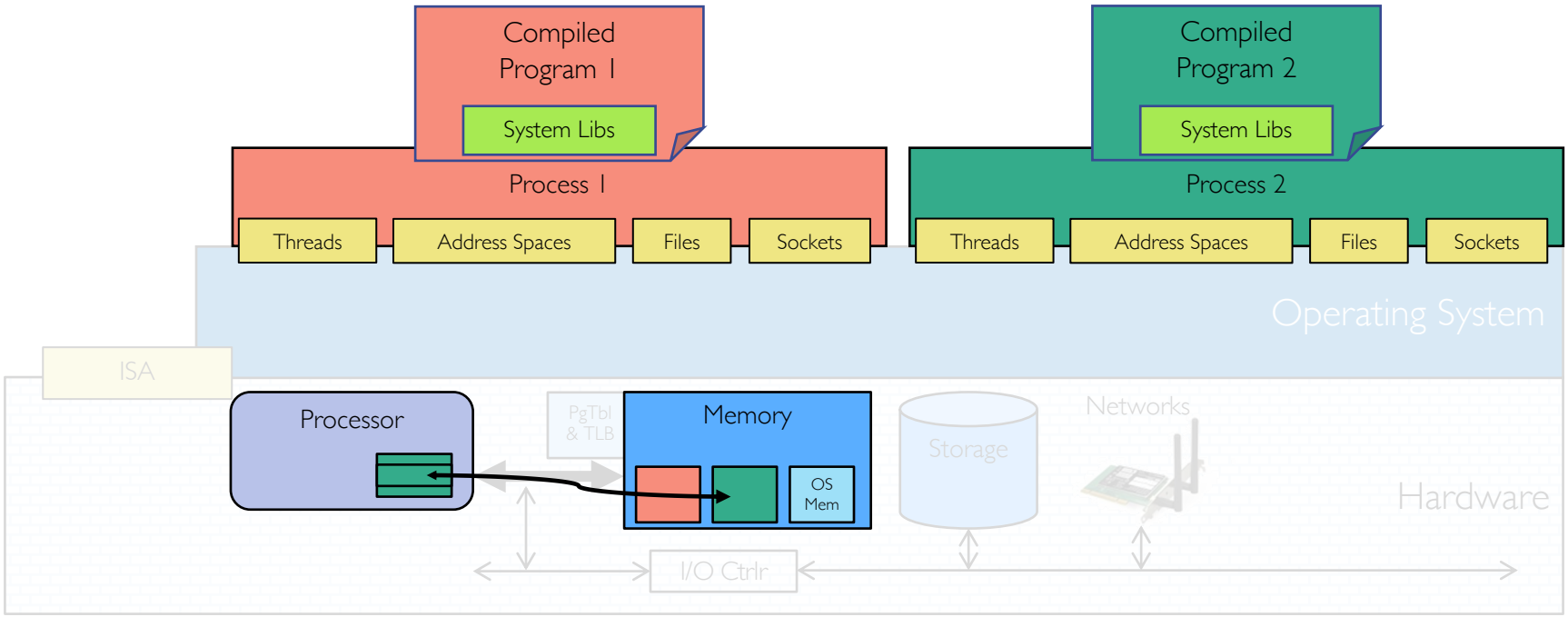
OS Basics: Switching Processes



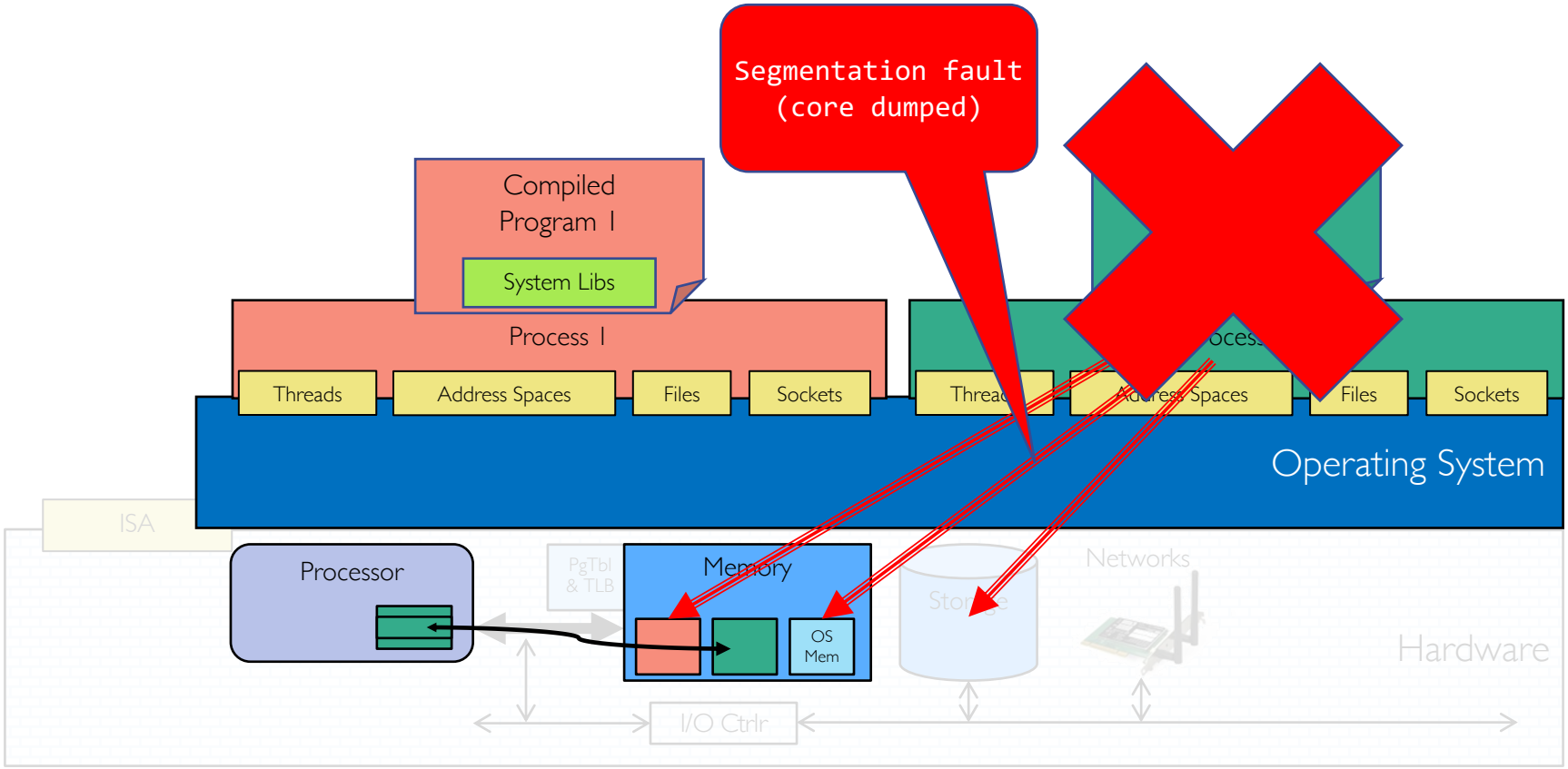
OS Basics: Switching Processes (cont.)



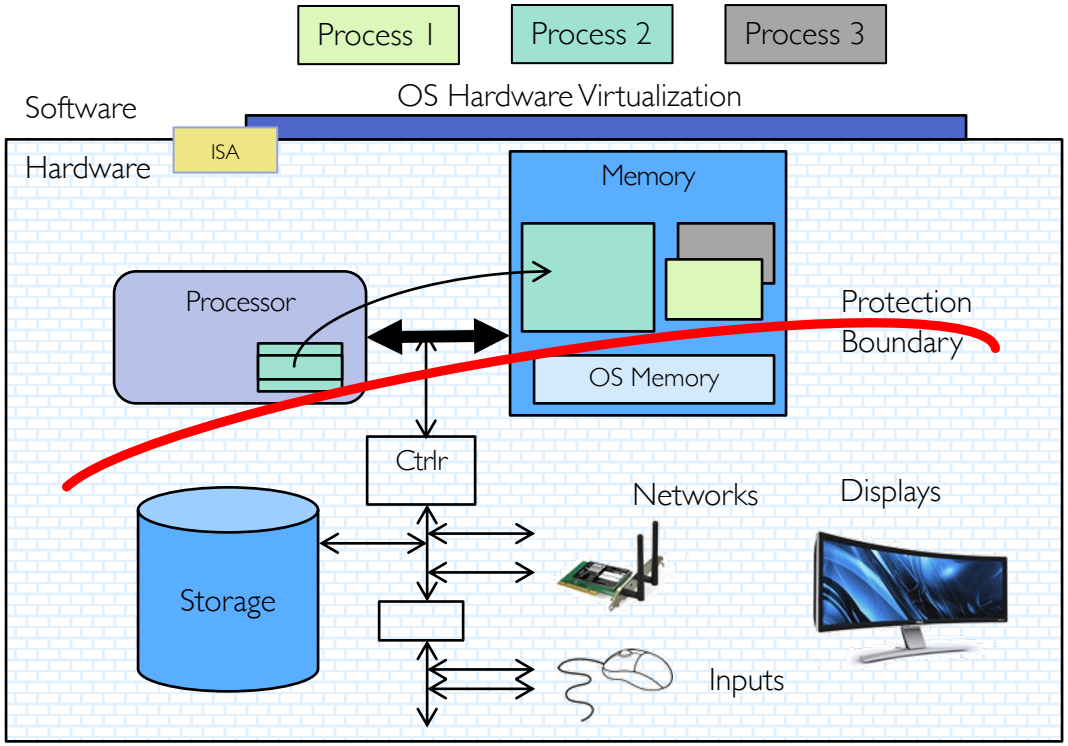
OS Basics: Switching Processes (cont.)



OS Basics: Protection



OS Basics: Protection (cont.)



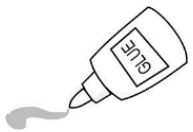
- OS isolates processes from each other
- OS isolates itself from other processes
- … even though they run on the same HW!

What is an Operating System? (cont.)

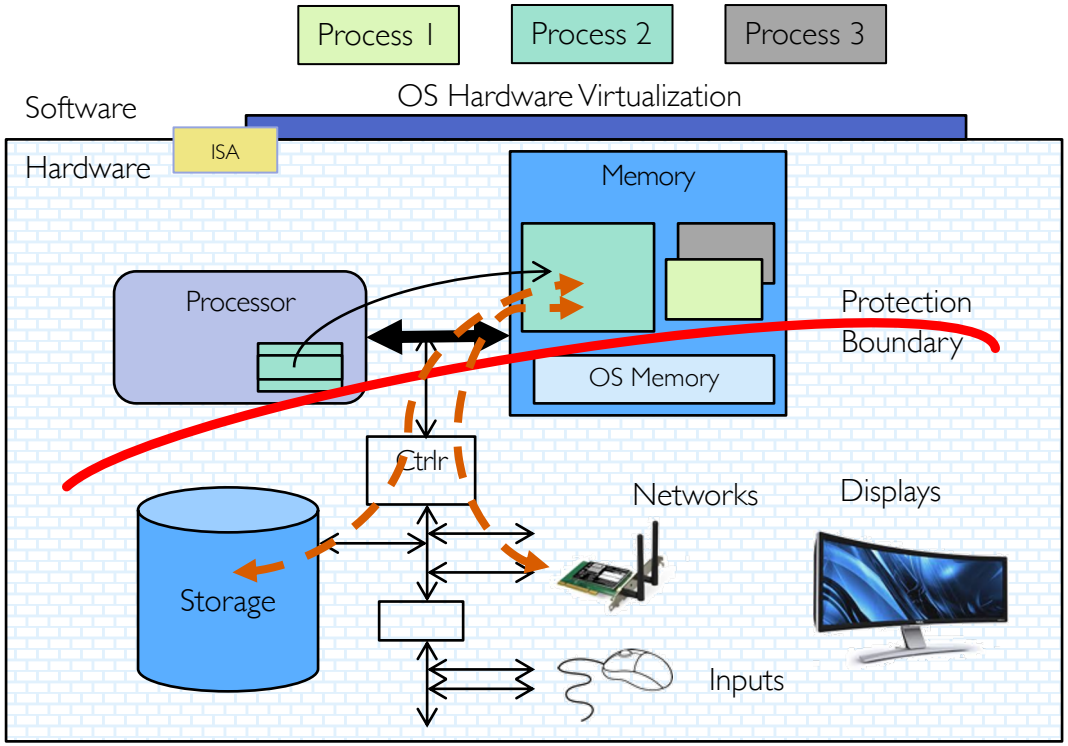
- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

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

- Glue
  - Provide common services
    - Storage, window system, networking, sharing, authorization
    - Look and feel

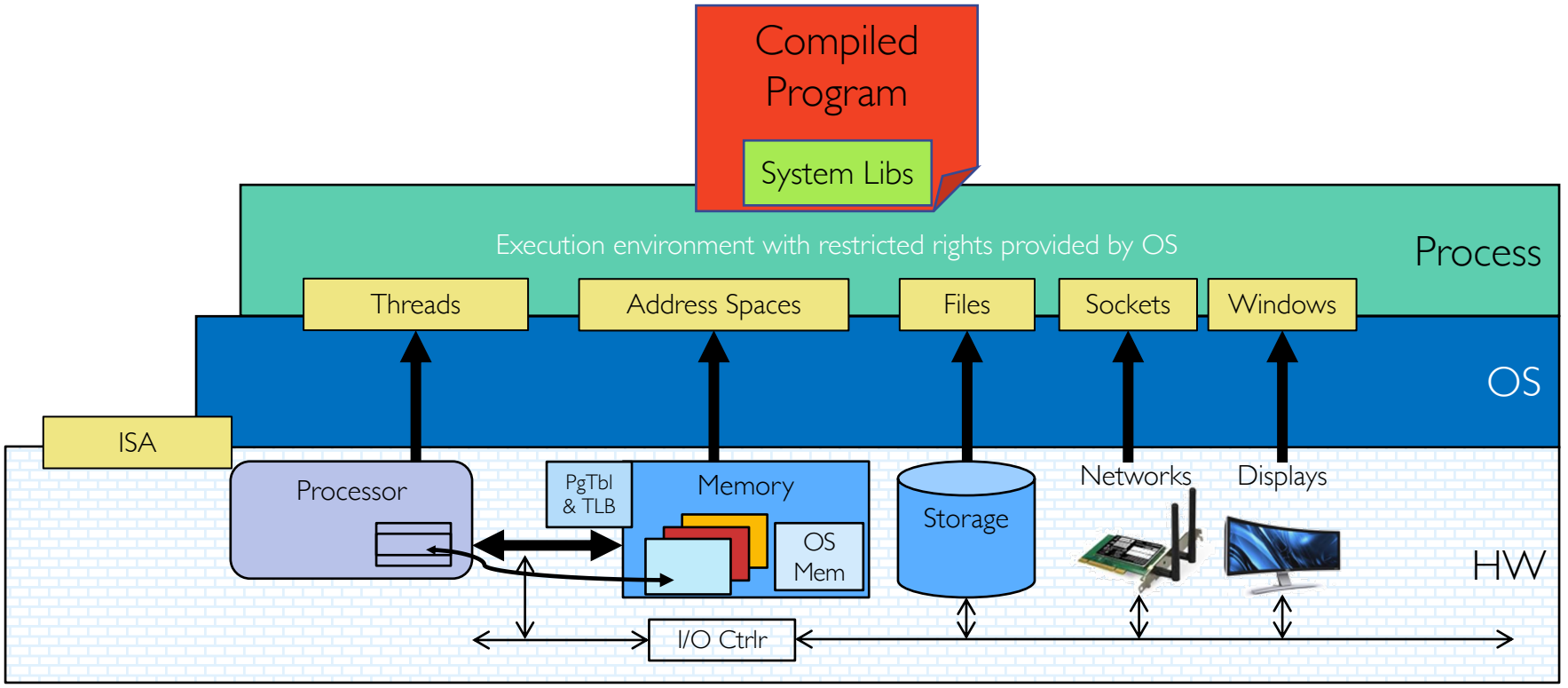


OS Basics: I/O

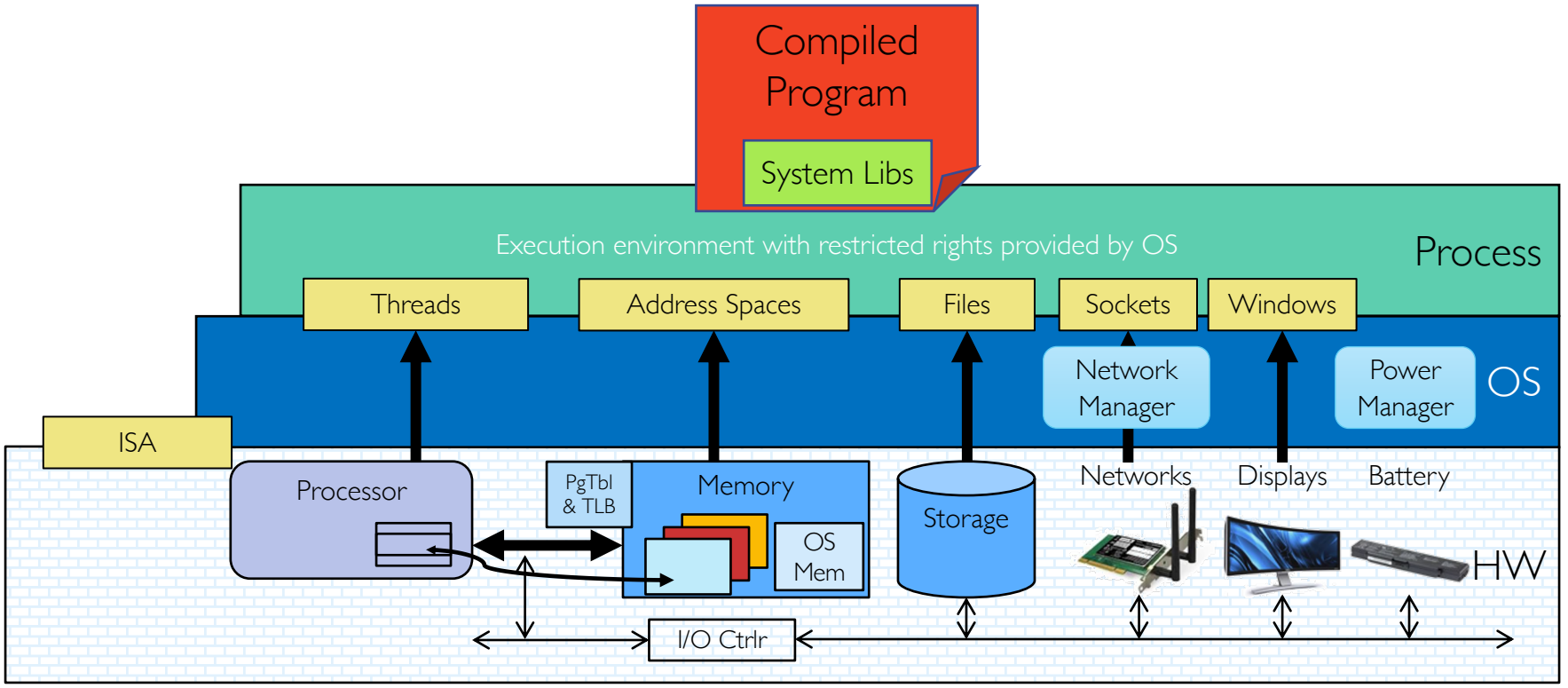


- OS provides common services in the form of I/O

OS Basics: Look and Feel



OS Basics: Background Management

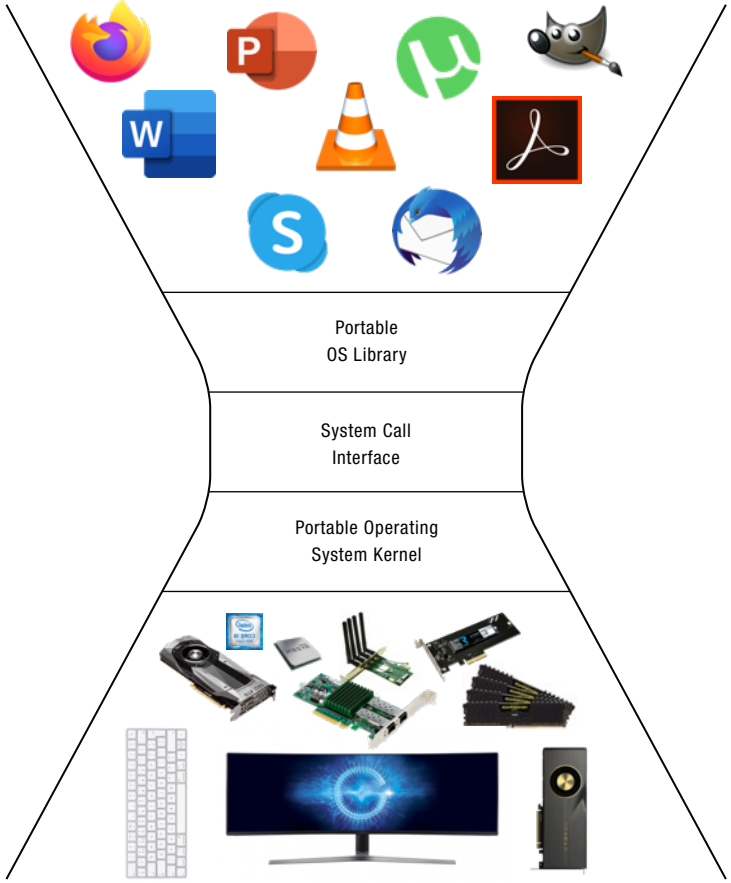


OS Basics: Hardware Support

- OS bottom line is to support applications!
  - OS itself is incidental
  - Ideally, OS should have very low performance overhead over raw hardware
- OS relies on HW support to provide abstractions efficiently
  - Dual-mode operation, interrupts, traps, precise exceptions, memory management unit, translation lookaside buffer, etc.
- HW support and OS design continue to co-evolve…
  - … as hardware performance improves (e.g., faster storage/network), …
  - … and application requirements change
  - What we study in this class is result of decades of co-evolution!

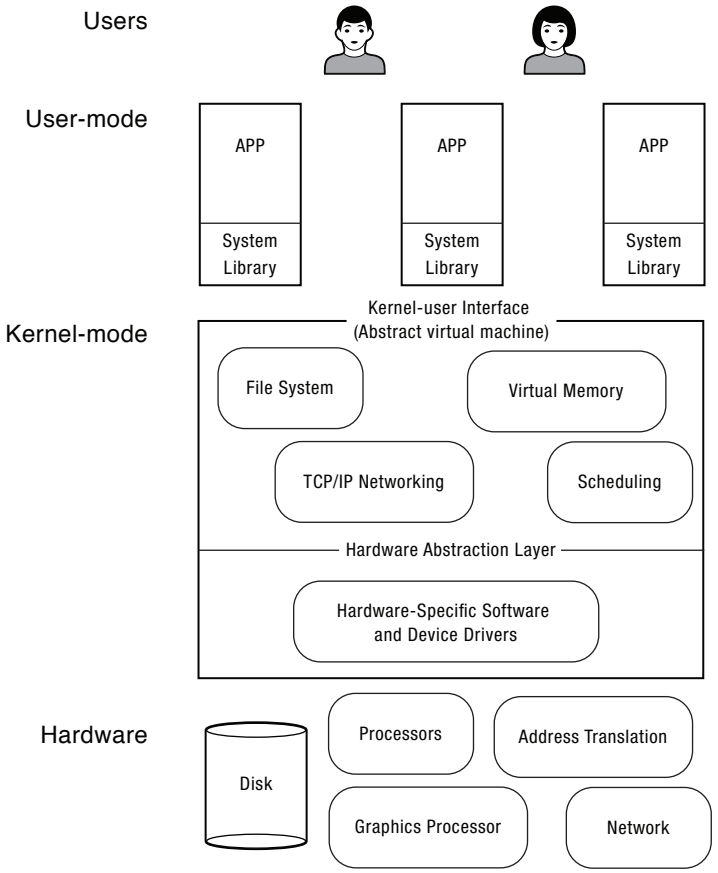
What Do Operating Systems Do?

- Provide abstractions to applications
  - File systems
  - Processes, threads
  - Virtual memory
  - Naming system, …
- Manage diverse resources
  - Memory, CPU, storage, …
- Achieves above by implementing specific algorithms and techniques
  - Scheduling
  - Concurrency
  - Transactions
  - Security, …



What Do Operating Systems Do? (cont.)

- Manage hardware resources for users and applications
- Convert what hardware gives into something that application programmers want
- For any OS component, begin by asking two questions
  - What is hardware interface? (physical reality)
  - What is application interface? (virtual machine)



Virtual Machines (VMs)

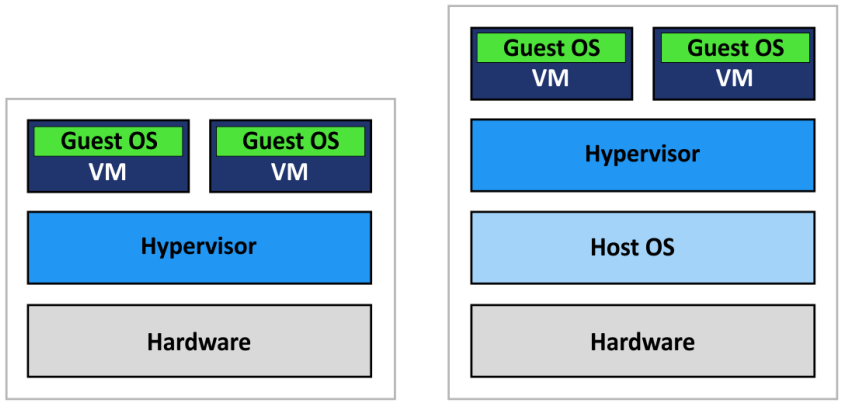
- Software emulation of abstract machine
  - Gives programs illusion that they own entire machine
  - Makes it look like hardware has features programs want
- Two types of virtual machines
  - Process VM: supports execution of single program (e.g., Java)
  - System VM: supports execution of entire OS (e.g., VMWare Fusion, Virtual box, Parallels Desktop, Xen)

Process VMs

- Programming simplicity
  - Each process thinks it owns all devices
  - Each process thinks it has all memory/CPU time
  - Device interfaces more powerful than raw hardware
    - Bitmapped display ⇒ windowing system
    - Ethernet card ⇒ reliable, ordered, networking (TCP/IP)
- Fault isolation
  - Processes unable to directly impact other processes
  - Bugs cannot crash whole machine
- Protection and portability
  - Java interface safe and stable across many platforms

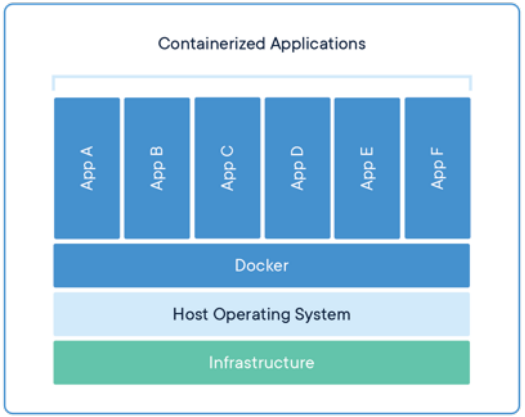
System Virtual Machines: Layers of OSes

- VMs are useful for OS development and testing programs on other OSes
- Hypervisors create and run virtual machines
- Type-I hypervisors allocate HW to VMs in addition to managing them
  - E.g., Xen, VMWare ESXi
- Type-II hypervisors rely on host OS for HW management
  - E.g., Virtual Box, VMWare Workstation, KVM

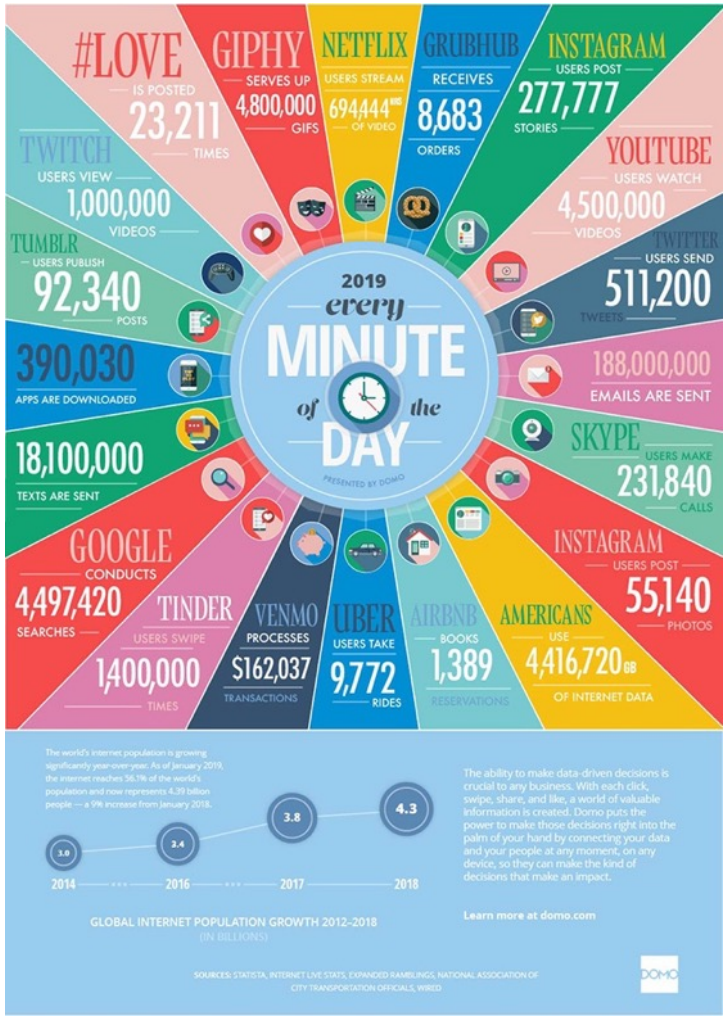


Containers: Low-weight Alternatives to Full-system Virtualization

- Do not provide *full-machine* abstraction
- Use OS constructs to provide *sand boxes* for execution
  - E.g., Linux cgroups, namespaces, etc.
- Can run on bare metal, or atop of VMs
- OS containers: provide virtualized OSes above single shared kernel
  - E.g., LXC, OpenVZ, FreeBSD Jail
- Application containers: expect single application to execute within container
  - E.g., Docker, rkt



What Makes Operating Systems so Exciting and Challenging?



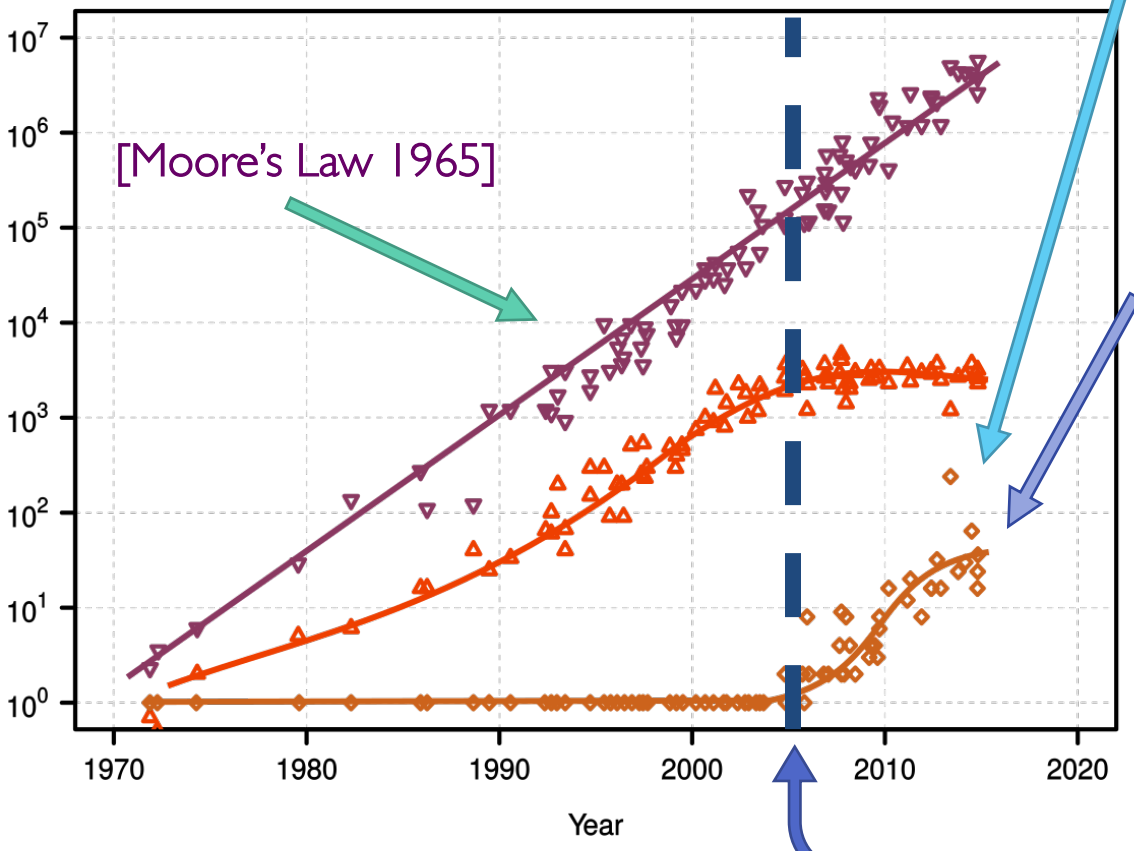
Operating systems are at the heart of it all …

- Challenges
  - Keeping up with evolving HW
  - Managing ever-growing complexity of SW

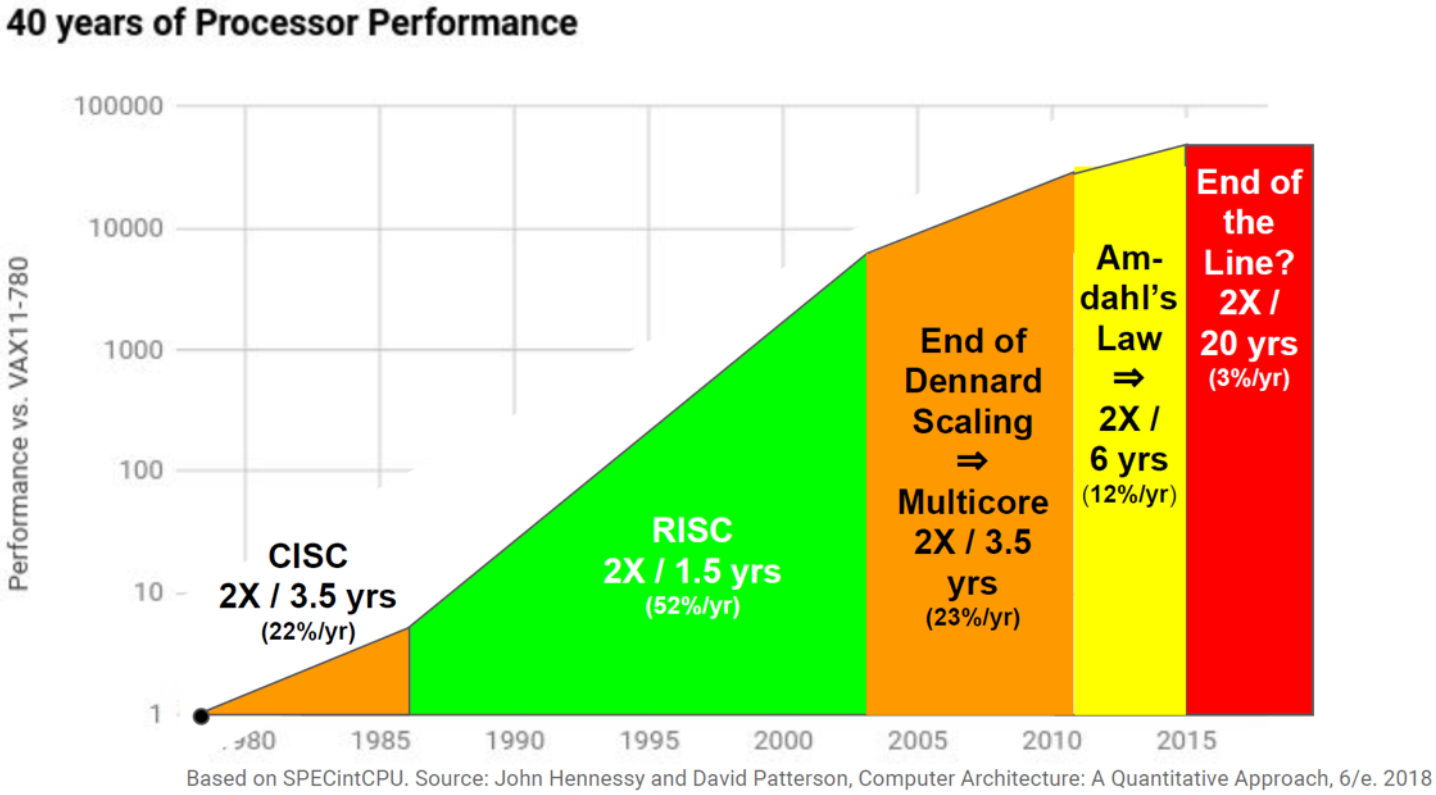
Technology Trends

How do we program these?

Parallelism must be exploited at all levels



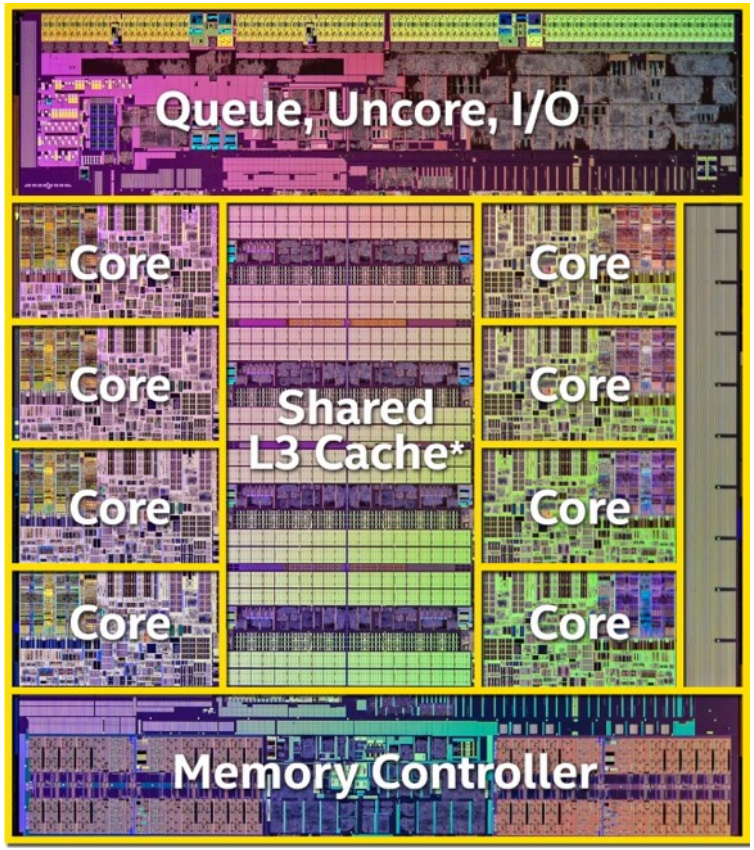
End of Growth of Single Program Speed



From John Hennessy and David Patterson's 2017 ACM A.M. Turing Award Lecture

Modern Processors

Intel Haswell E

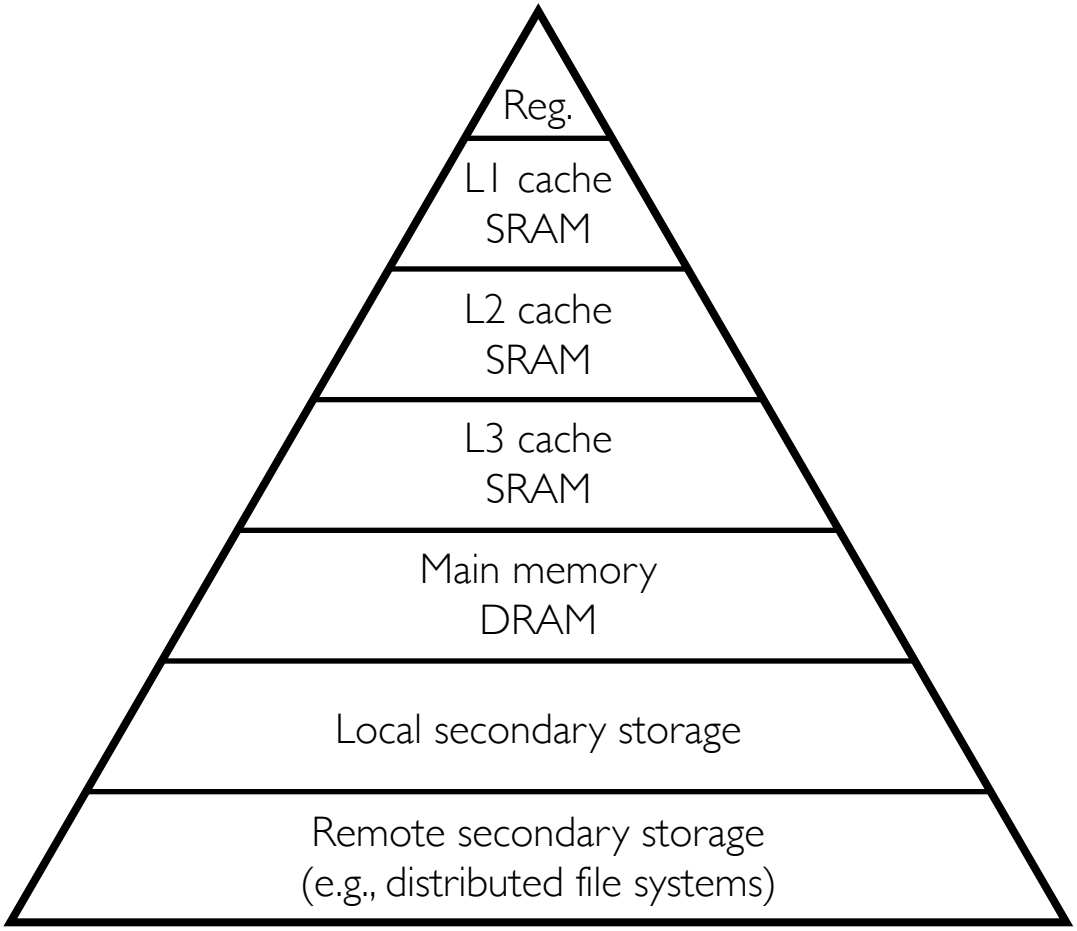


- Intel Xeon Platinum 9282
  - 14nm processor
  - 56 cores, 112 threads
  - 1.75MB data and ins. L1 cache
  - 56MB L2 cache
  - 77MB shared L3 cache
  - 8B transistors
- AMD EPYC 7H12
  - 7nm processor
  - 64 cores, 128 threads
  - 2MB data and ins. L1 cache
  - 32MB L2 cache
  - 256MB shared L3 cache
  - 4.8B transistors

Memory Hierarchy

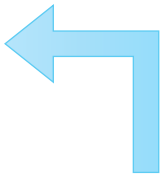
Smaller
Faster
Costlier (per byte)

Larger
Slower
Cheaper (per byte)



Numbers Everyone Should Know [Jeff Dean, 2009]

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



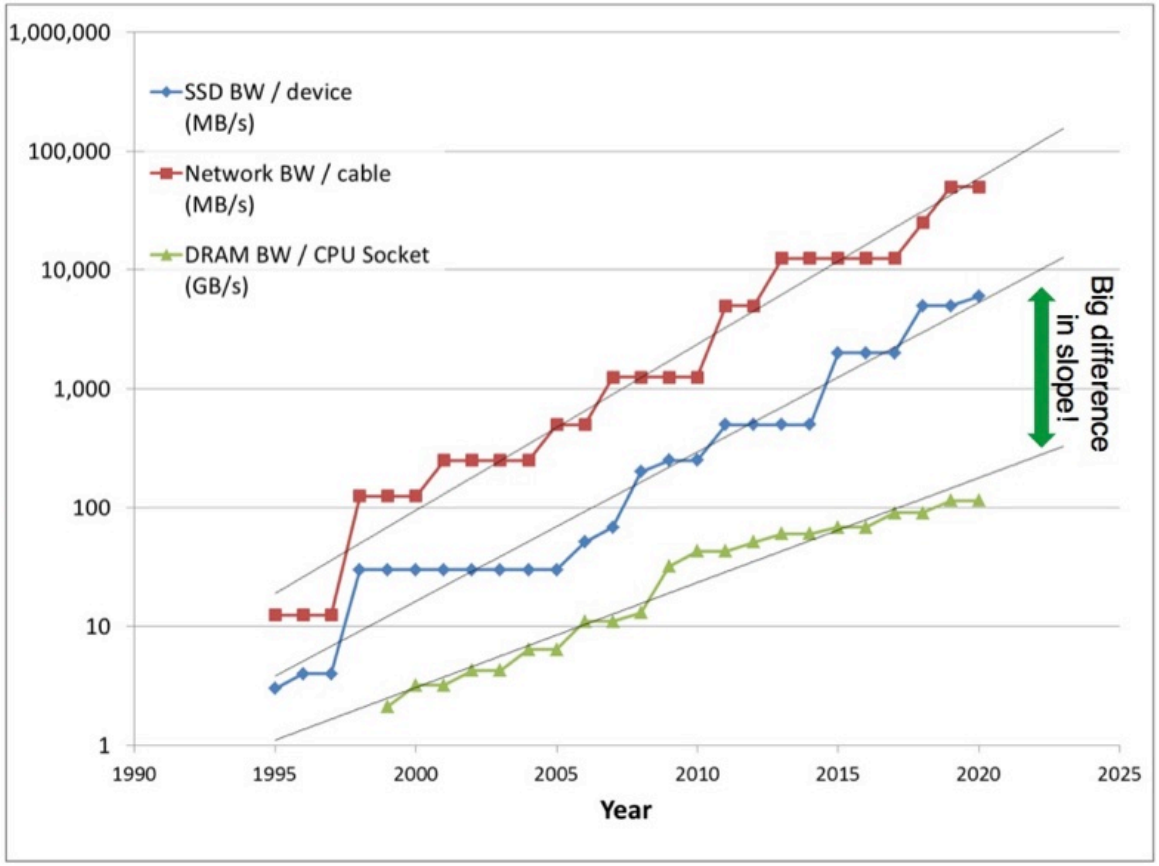
Key stroke ~100 ms

Network, IO, and Memory Bandwidth Trends

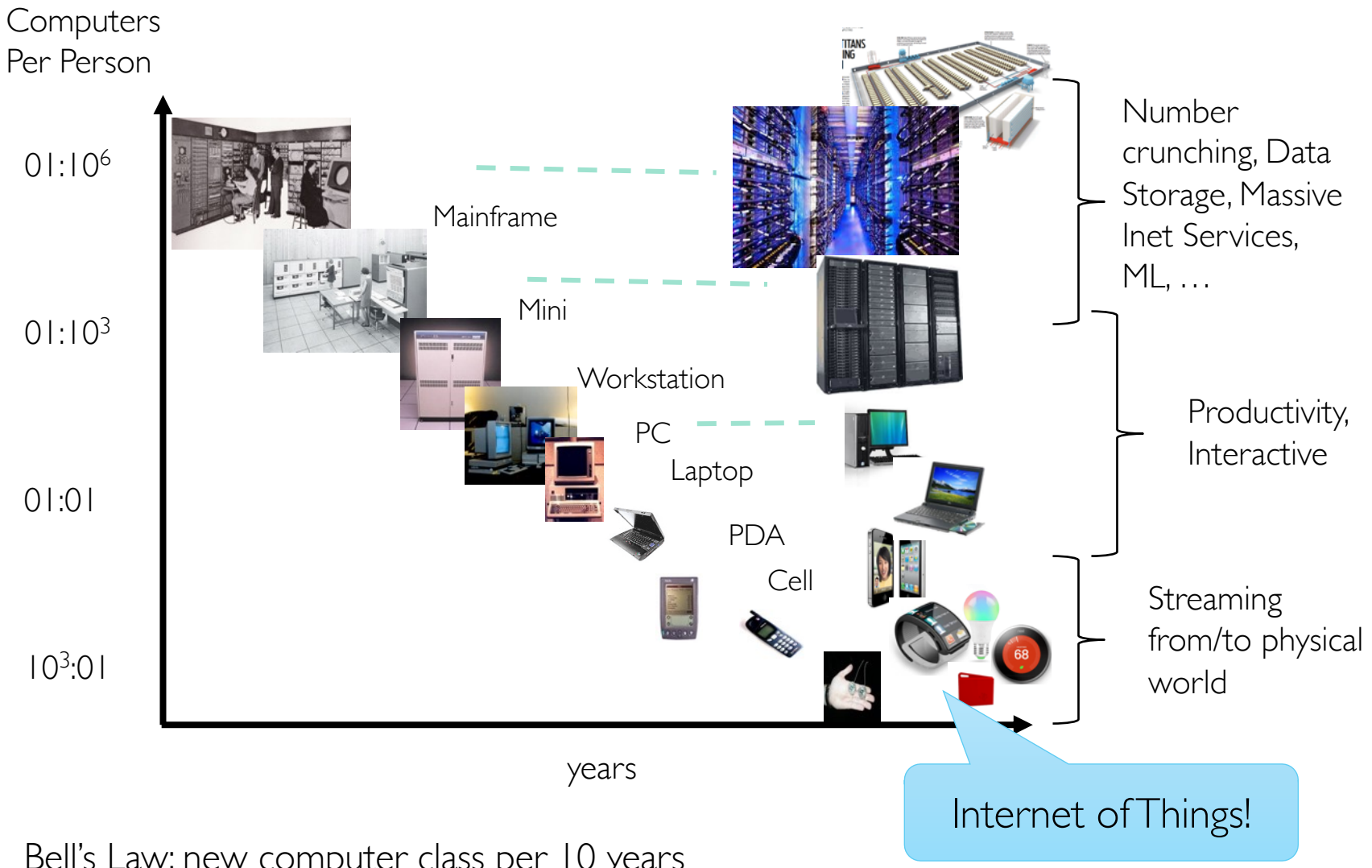
Network, Storage, & DRAM trends

Log scale

- Use DRAM Bandwidth as a proxy for CPU throughput
- Reasonable approximation for DMA and poor cache performance workloads (e.g. Storage)



People to Computer Ratio Trend



Bell's Law: new computer class per 10 years

Early vs. Current Operating Systems

- One user/application at any given time
  - Had complete control of hardware
  - OS was runtime library
  - Users would stand in line to use the computer

- Batch systems
  - Keep CPU busy by having a queue of jobs
  - OS would load next job while current one runs
  - Users would submit jobs, and wait, and wait, and wait …

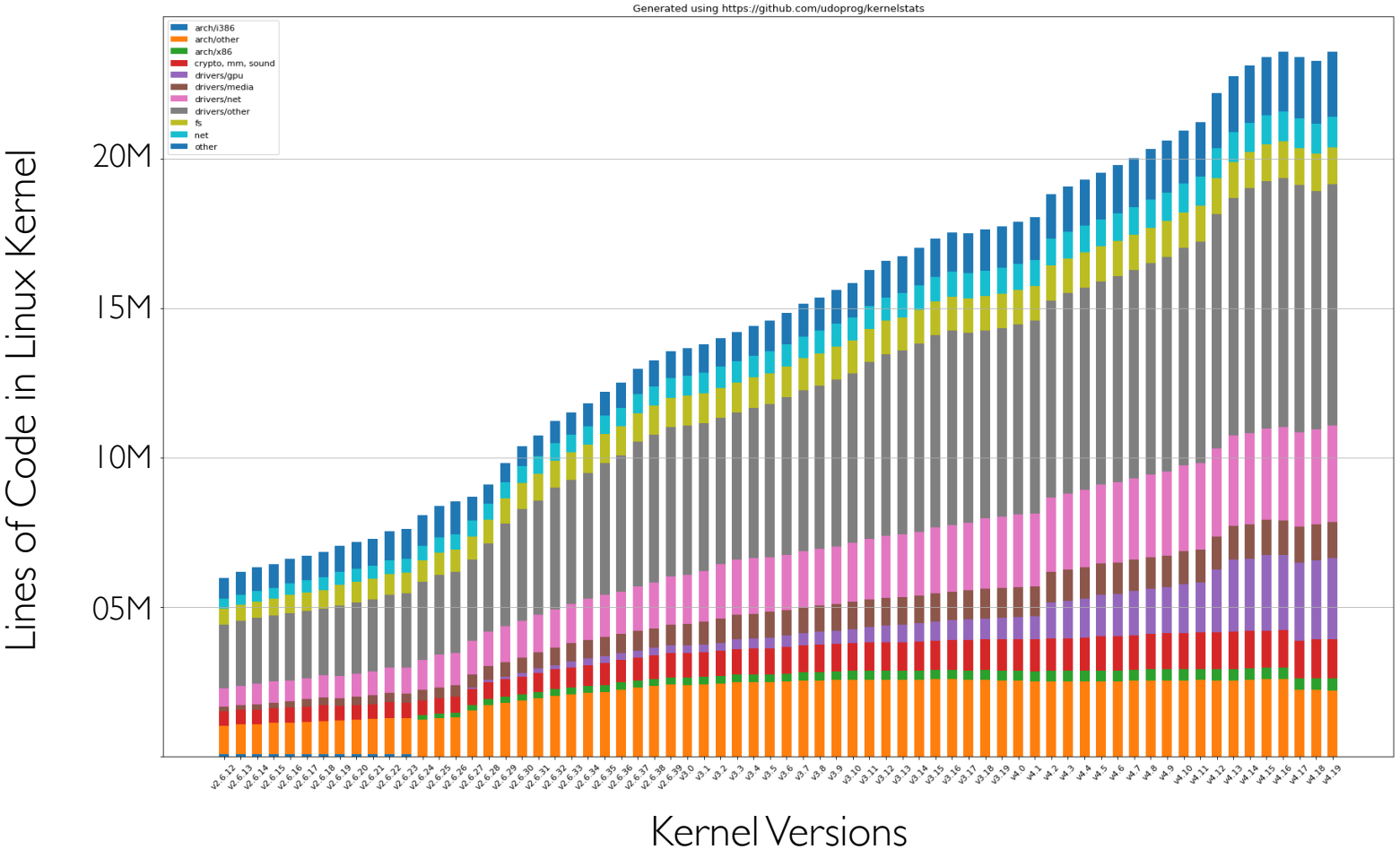
- Multiple users on computer at the same time
  - Multiprogramming: run multiple programs at the same time
  - Interactive performance: try to complete everyone's tasks quickly
  - As computers became cheaper, more important to optimize for user time, not computer time

Complexity

- Applications consisting of…
  - … a variety of software modules that …
  - … run on a variety of devices (machines) that
    - … implement different hardware architectures
    - … run competing applications
    - … fail in unexpected ways
    - … can be under a variety of attacks

- Not feasible to test software for all possible environments and combinations of components and devices
  - Question is not whether there are bugs but how serious are bugs!

Kernel Complexity



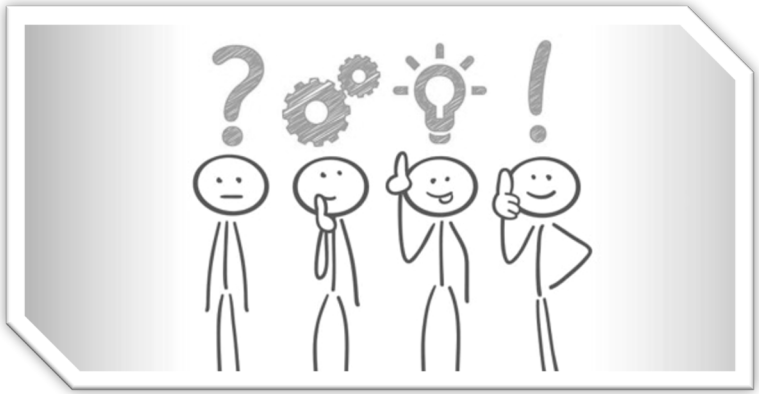
How do We Tame Complexity?

- Every piece of computer hardware different
  - Different CPUs
    - Pentium, ARM, PowerPC, ColdFire
  - Different amounts of memory, disk, …
  - Different types of devices
    - Mice, keyboards, sensors, cameras, fingerprint readers, touch screen
  - Different networking environment
    - Cable, DSL, Wireless, …
- Questions
  - Does programmer need to write single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does one faulty program crash everything?
  - Does every program have access to all hardware?

Summary

- OS provides VM abstraction to handle diverse HW
  - OS simplifies application development by providing standard services
- OS coordinates resources and protect users from each other
  - OS can provide fault containment, fault tolerance, and fault recovery
- ECE 350 combines ideas and concepts from many other areas of computer science and engineering
  - Languages, data structures, hardware, and algorithms

Questions?



Acknowledgment

- Slides by courtesy of Anderson, Culler, Stoica, Silberschatz, Joseph, Canny, and Kumar (Sam)