Lecture 1: Introduction

Prof. Seyed Majid Zahedi

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

Outline

- How do things work in SE 350?

- What is an operating system?

- What makes operating systems so exciting?

Useful Links

- Course webpage

  https://ece.uwaterloo.ca/~smzahedi/crs/se350

- Course on Piazza

  https://piazza.com/uwaterloo.ca/winter2022/se350

- Anonymous feedback form

  https://forms.gle/cR93sRrzTERZVtXN8

Class is Entirely Online!

- Lectures will be delivered on Teams
  - Links are provided on course webpage
  - Recordings will be available afterwards

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

- Lab tutorials will be delivered on Teams
  - Links are provided on course webpage

When you start hearing voices

But it's just your online class teacher

Who Are We?

Instructor: Prof. Seyed Majid Zahedi
zahedi@uwaterloo.ca
https://ece.uwaterloo.ca/~smzahedi

Lab instructor: Irene Huang
yqhuang@uwaterloo.ca

SE 350 GTAs

Karim Walid Abdelazim Elhammady
kwamelhammady@uwaterloo.ca
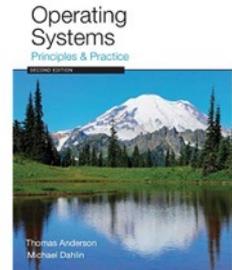
Sulav Shrestha
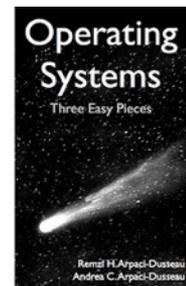slshrestha@uwaterloo.ca

Readings

- Main textbook

  Operating Systems: Principles and Practice (2nd Edition)
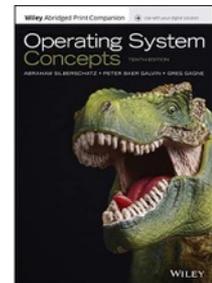

- Optional references

  Operating Systems: Three Easy Pieces (Freely Available)

  Operating System Concepts (10th Edition)

SE 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 SE 350?
Why Learn About OS?

- Some of you will design and build parts of 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

- In-person final: 30%

- Lab projects: 50%
    - 3 projects (more on this later)

- Quizzes: 20%
    - 9 quizzes (highest 7 will be considered)
    - 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 (you can email course instructors if you have any questions or need any clarification)
        - You may not discuss nor disclose quiz questions with anyone

Contingency Proviso

- Course outline presents intended weights, and due dates

- As best as possible, we will keep to the outline

- We reserve the right to modify topics and/or assessments and/or weight and/or deadlines with due and fair notice

- In the event of such challenges, we will work with the Department/Faculty to find reasonable and fair solutions

Lab Project

- You will design, implement, and test
  real-time executive (RTX) on Keil MCB1700 boards
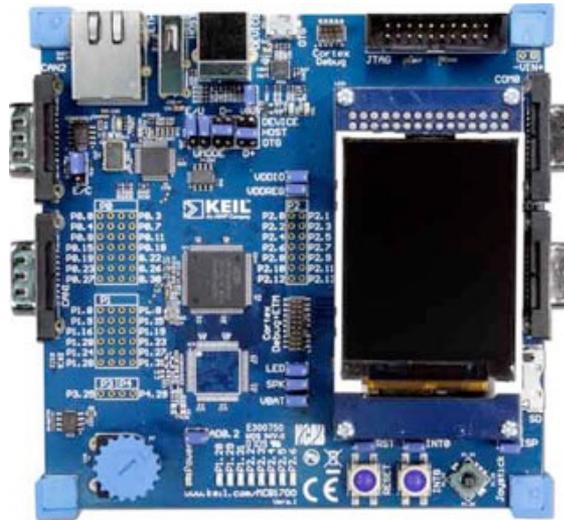
Groups

- Groups should have 4 members
    - Never 5! 3 requires serious justification
    - Sign up on LEARN by 23:00 on Jan. 11th EST

- Only one split-up is allowed
    - One-week notice in writing before nearest deadline
    - All students involved lose their group sign-up points

Milestones

Project	Weight	Due Date
Group sing-up	3%	8:30 Jan 11
Memory and task management (P1)	33%	8:30 Feb 1
Message passing and timing (P2)	32%	8:30 Mar 1
I/O and stress testing (P3)	32%	8:30 Mar 29

All times are Eastern Standard Time

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 $\Rightarrow$ 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)

BE NICE.
WE ALL MAKE MISTAKES

photo by Amanda Lowther

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

- 3 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) <u>even after this term</u>

- 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

Seeking Help

- Lab Q&A on Piazza discussion forum
  - Looking for group partners
  - Lab/Project administration
  - 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

Sign up for project groups on LEARN by

23:00 on January 11th , 2022 EST

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
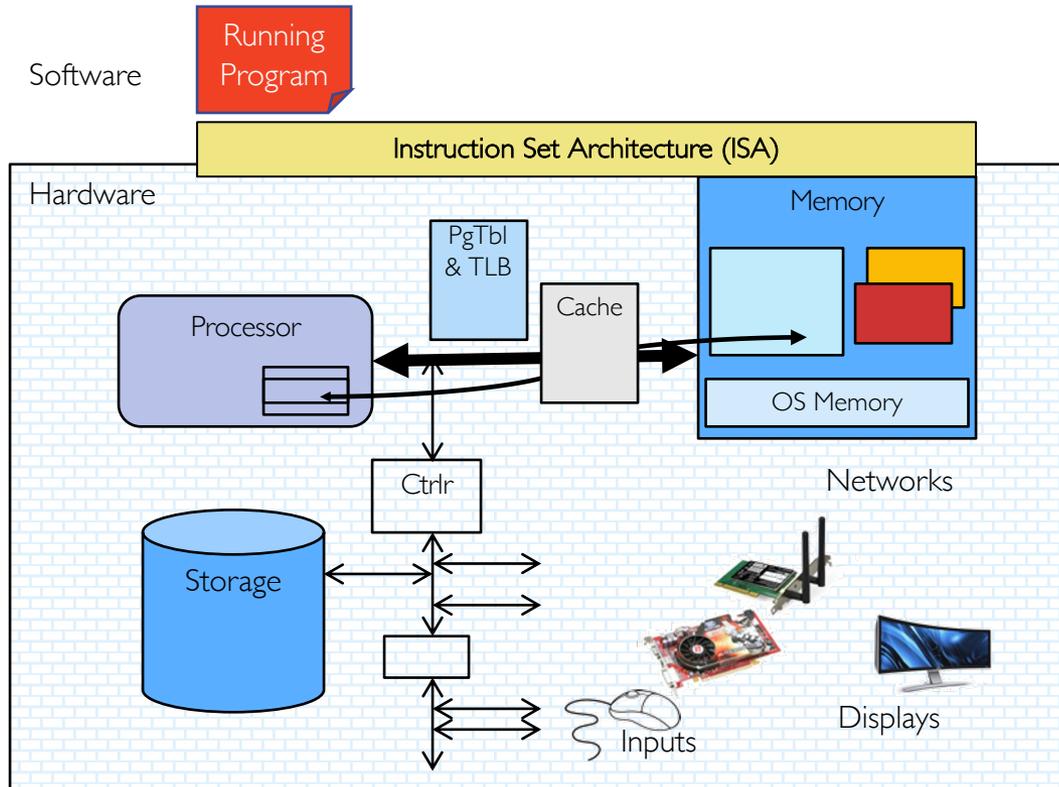
- ECE 222 and ECE 320: Machine structures (and C)

- OS *abstracts* these hardware details from the application

OS Basics: Virtualizing Hardware
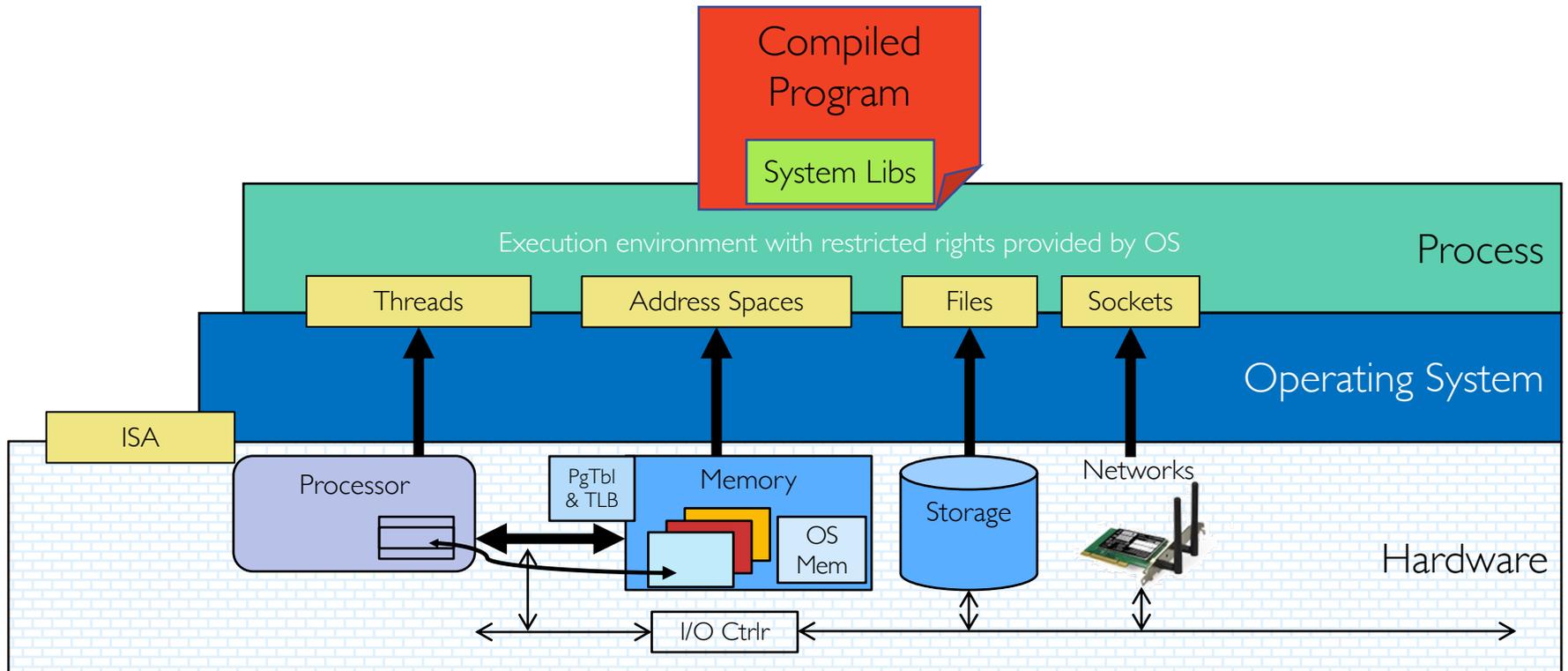
What is an Operating System? (cont.)

- 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
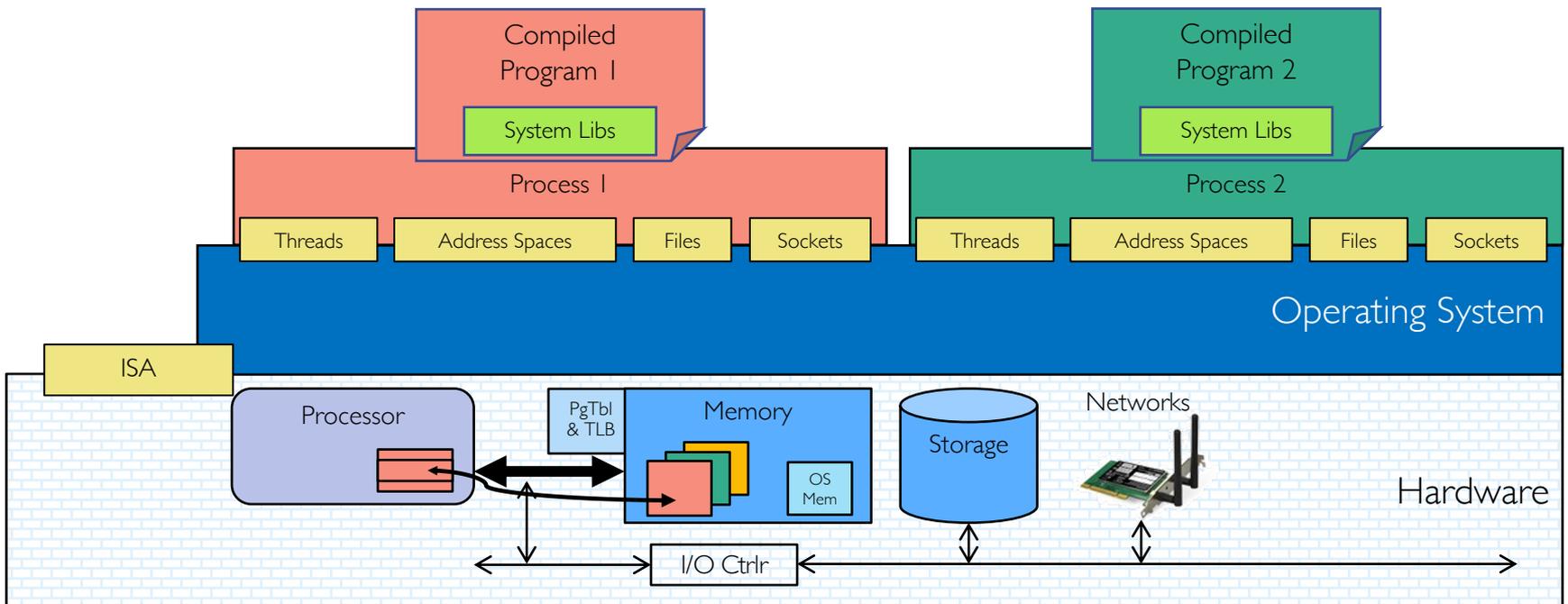
OS Basics: Switching Processes
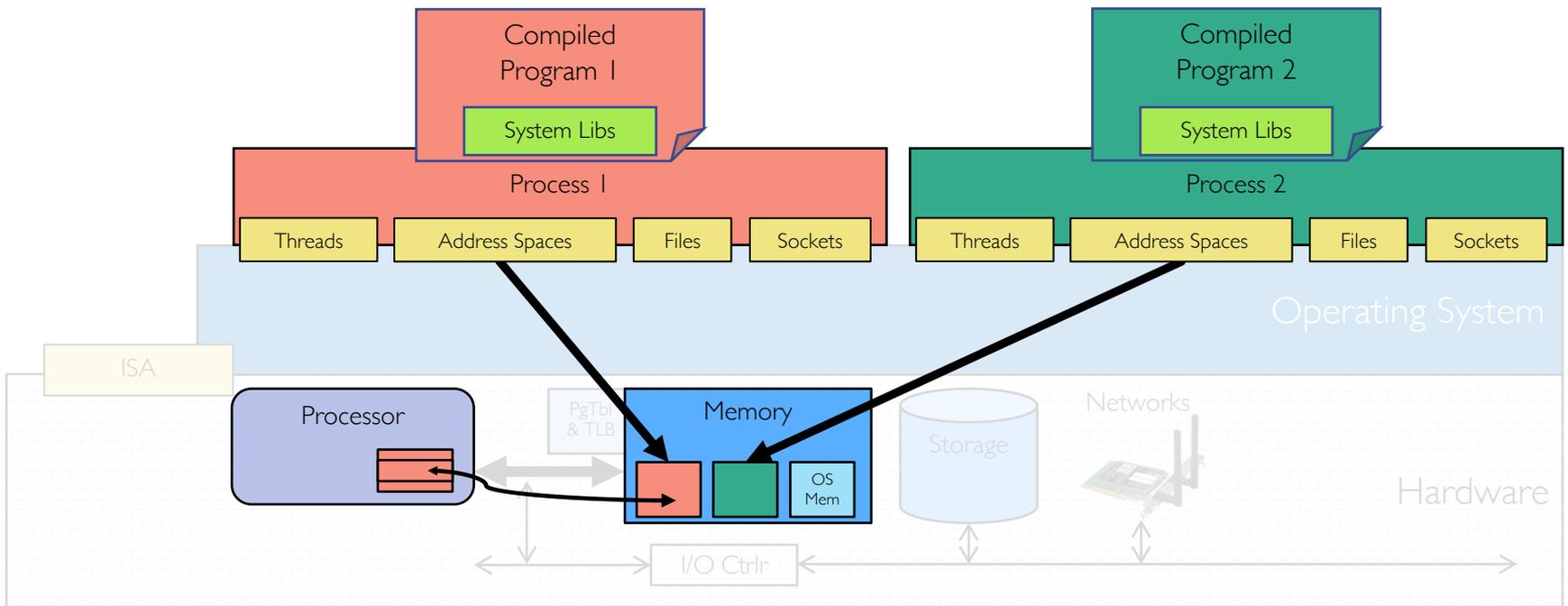
OS Basics: Switching Processes (cont.)
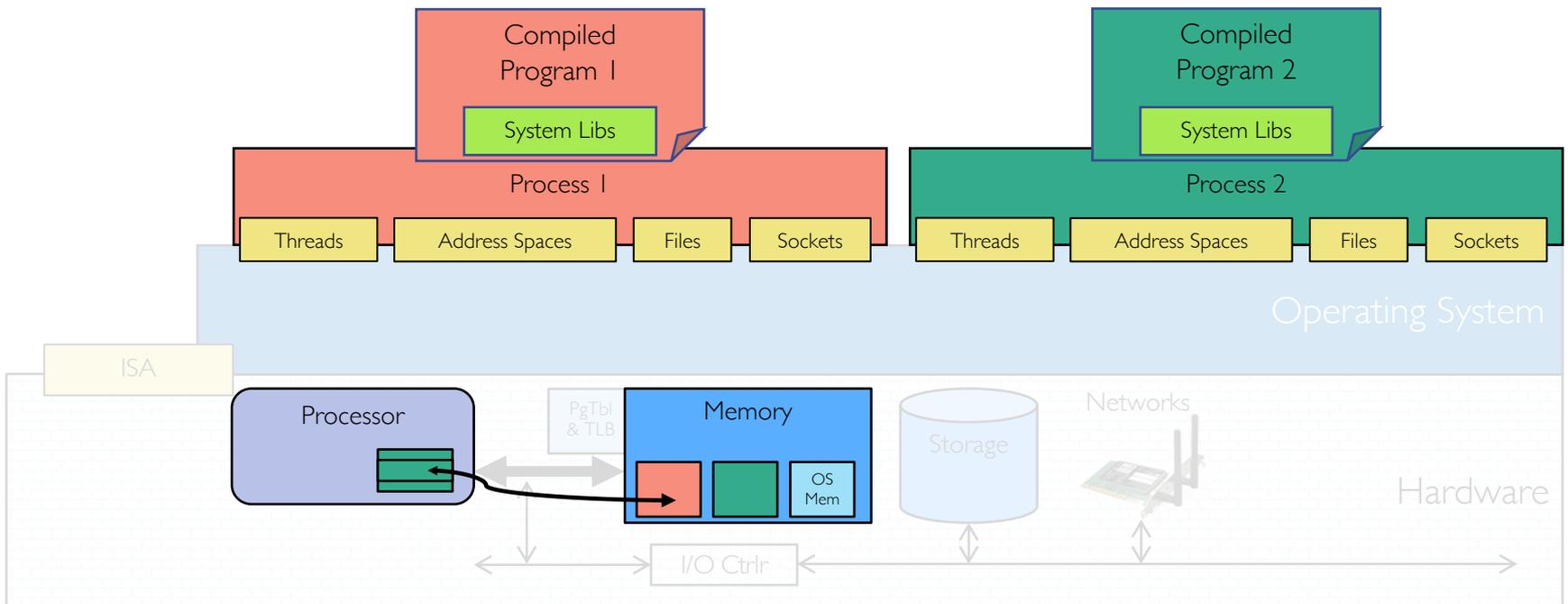
OS Basics: Switching Processes (cont.)
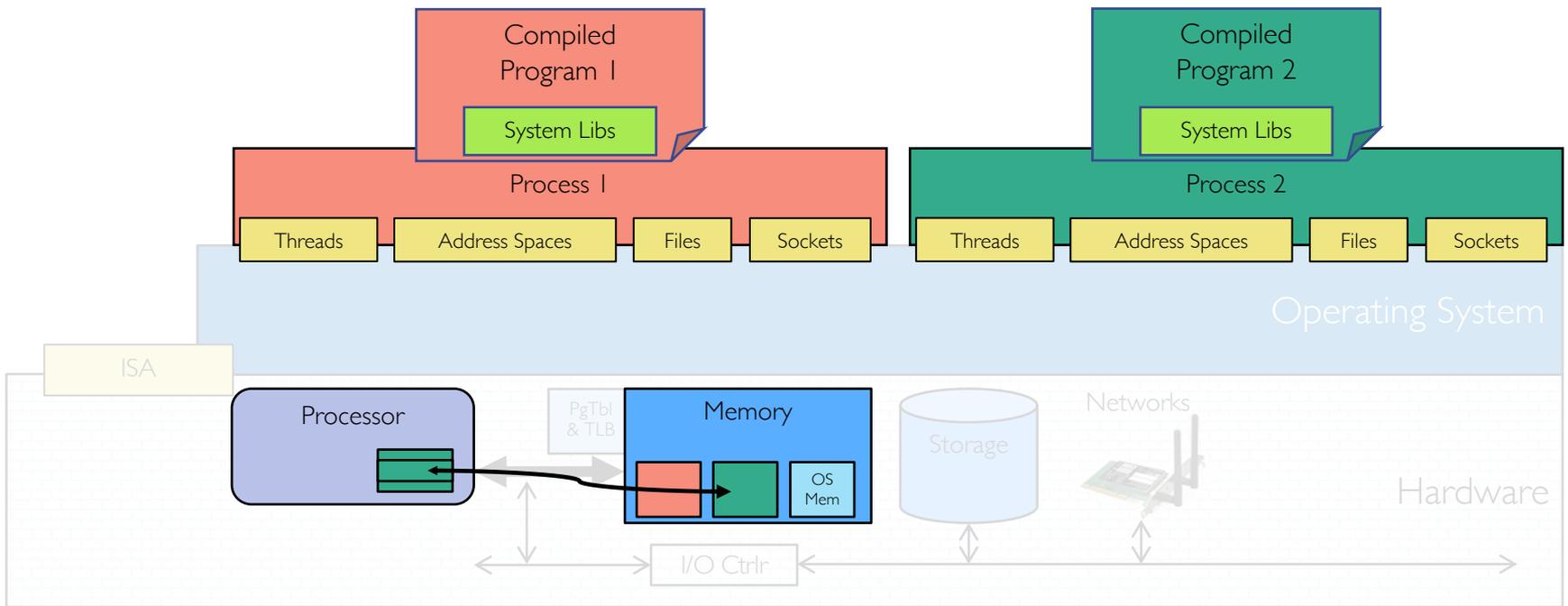
OS Basics: Protection
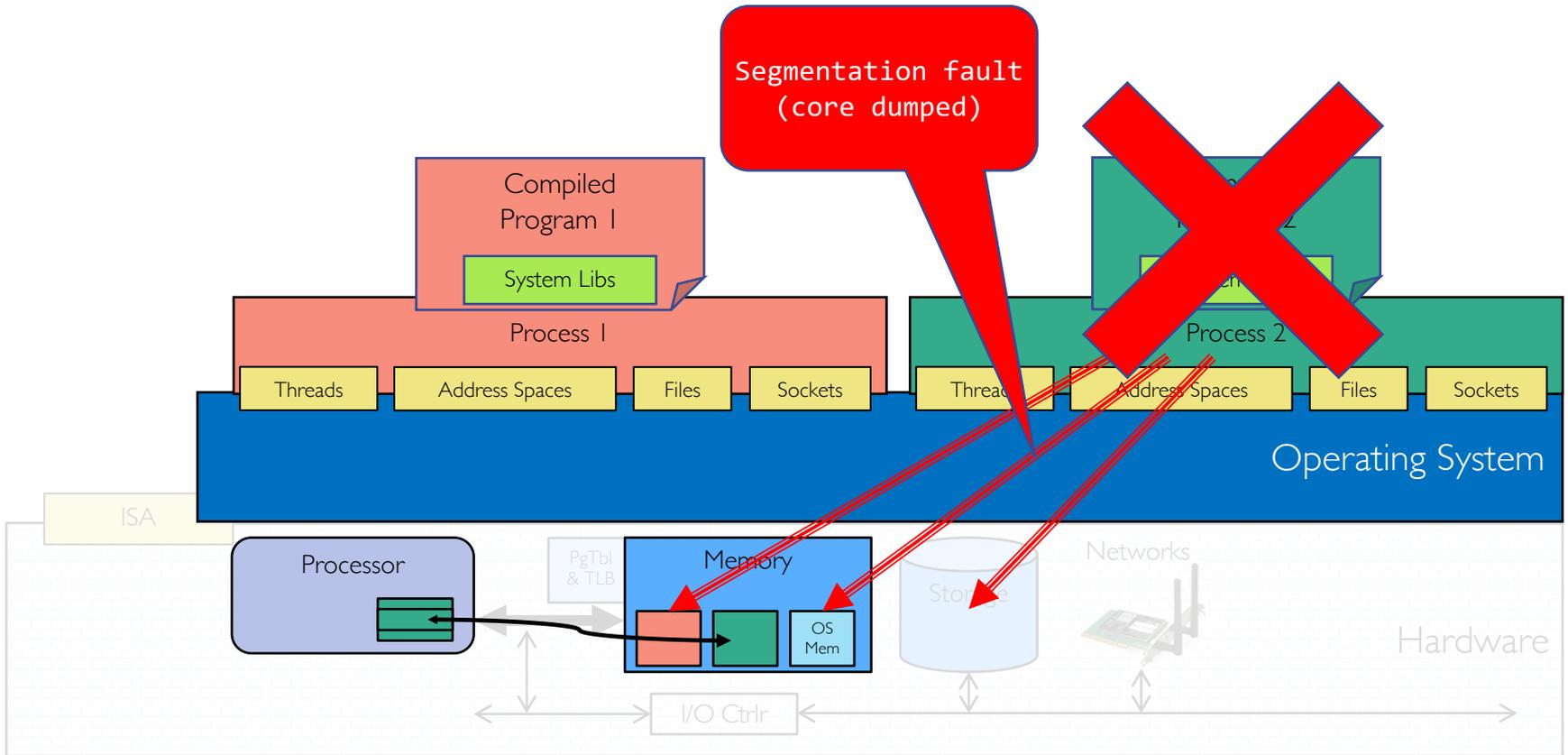
OS Basics: Protection (cont.)
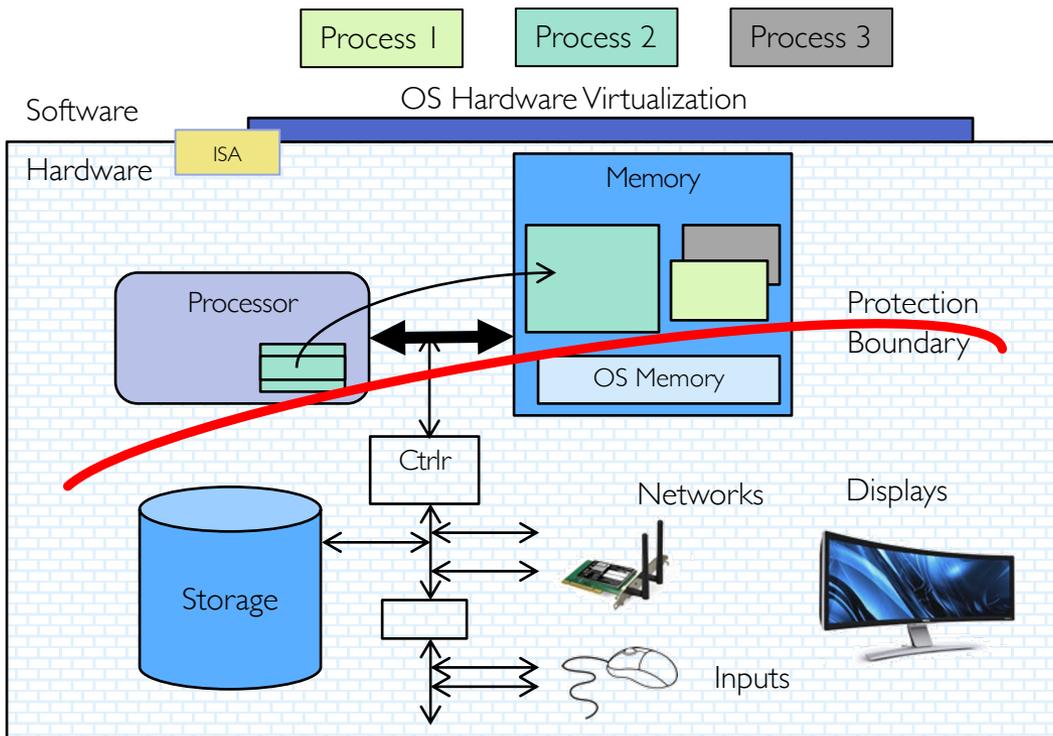
- OS isolates processes from each other

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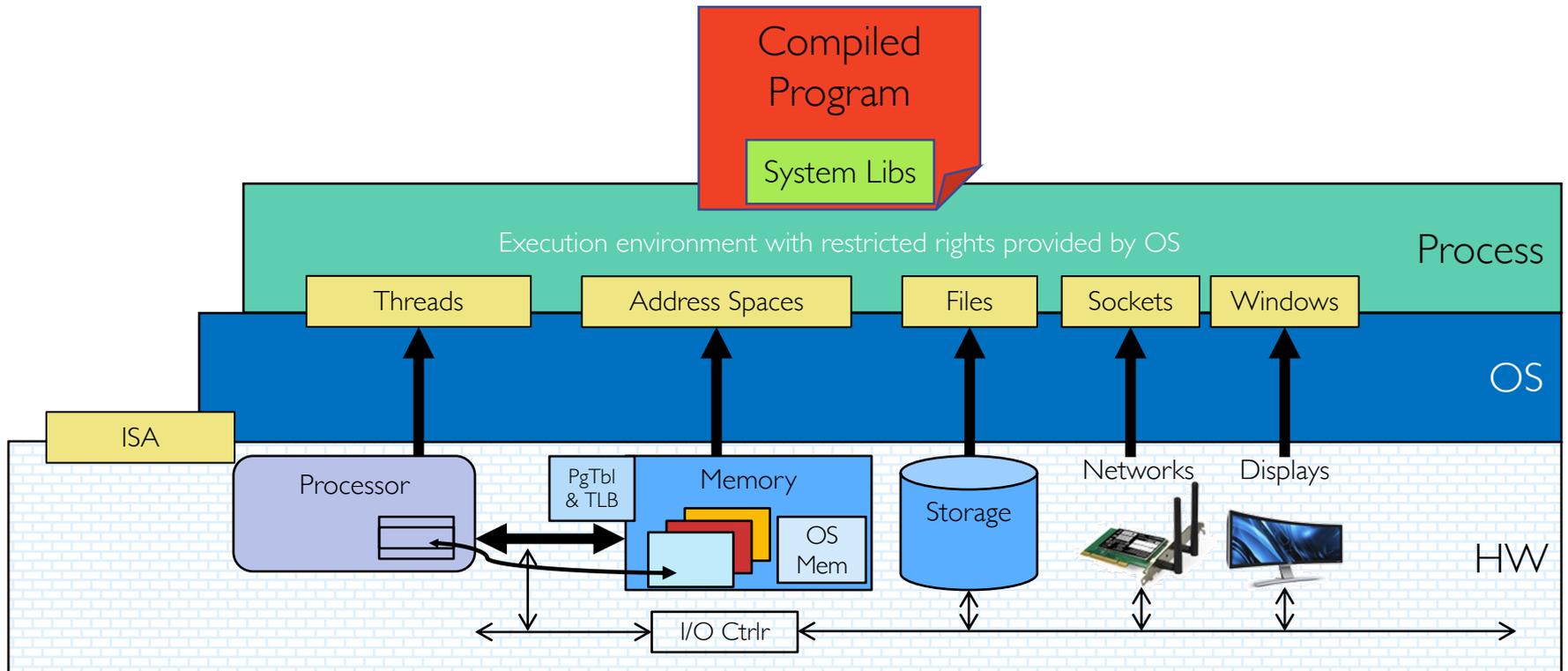   - Resource allocation and communication
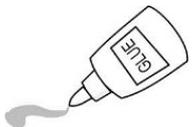
- Glue
  - Provide common services
    - Storage, window system, networking, sharing, authorization
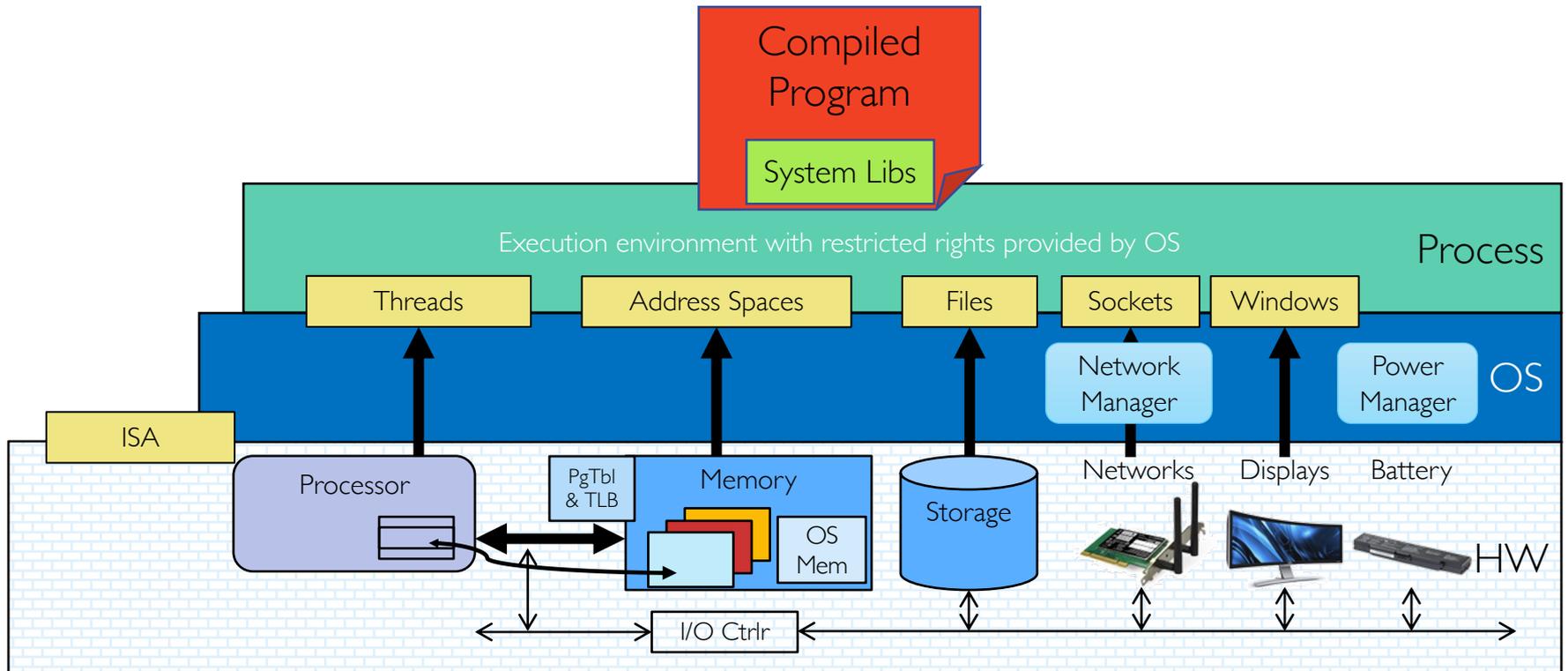   - Look and feel

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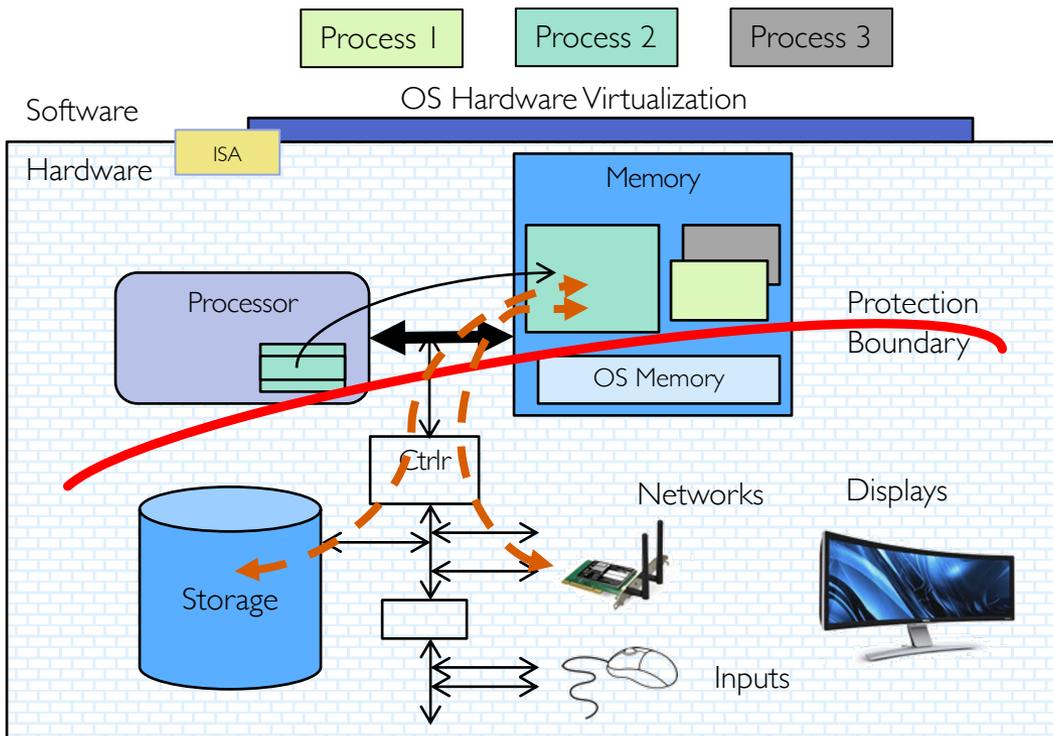
- OS provides common services in the form of I/O

# OS Basics: Look and Feel

OS Basics: Background Management

Compiled Program

System Libs

Execution environment with restricted rights provided by OS

Process

Threads

Address Spaces

Files

Sockets

Windows

Network Manager

Power Manager

OS

ISA

Processor

PgTbl & TLB

Memory

OS Mem

Storage

Networks

Displays

Battery

HW

I/O Ctrlr

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, …

Portable
OS Library

System Call
Interface

Portable Operating
System Kernel
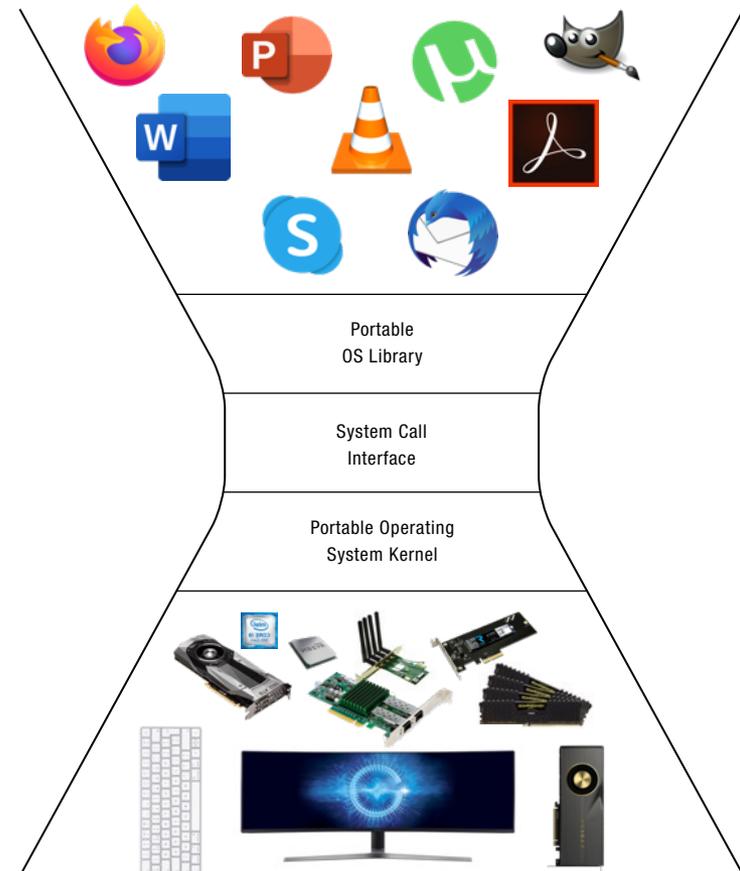
What do Operating Systems do? (cont.)

- 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)
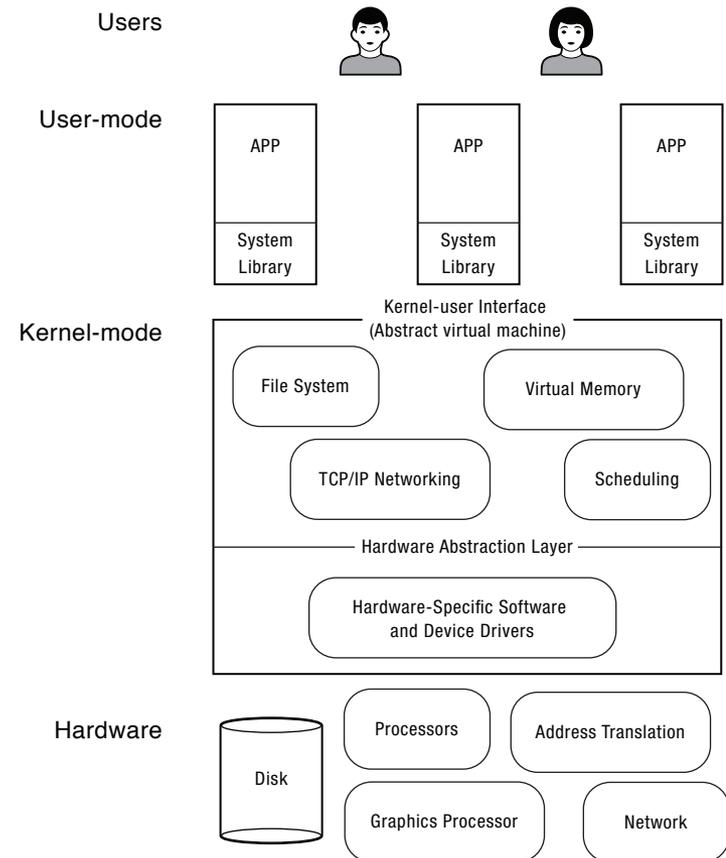
Virtual Machines (VMs)

- Software that emulates physical machine
  - Gives programs illusion that they run on physical machine
  - Provides platform that is independent of actual underlying hardware
  - 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)
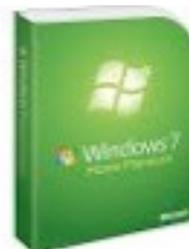
Process VMs

- Runs processes
  - Abstracts underlying OS and hardware
  - Provides platform-independent environment
  - E.g., Java virtual machine, .NET framework

Proc.	Proc.	Proc.	Proc.

Virtual ABI & ISA

Virtual Machine

Actual ABI

Host OS

Actual ISA

Hardware
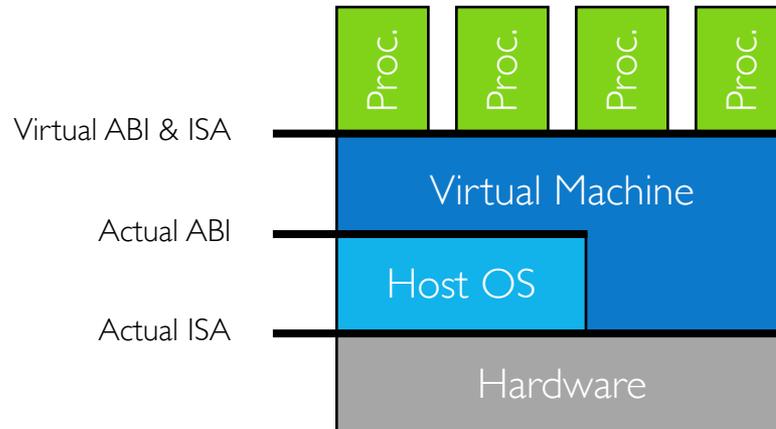
System Virtual Machines: Layers of OSes

- Runs OSes
  - Useful for OS development and testing programs on other OSes

- Hypervisors create and run virtual machines

- Type-1 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

Type 1 Hypervisor (Bare-Metal Architecture)

Type 2 Hypervisor (Hosted Architecture)
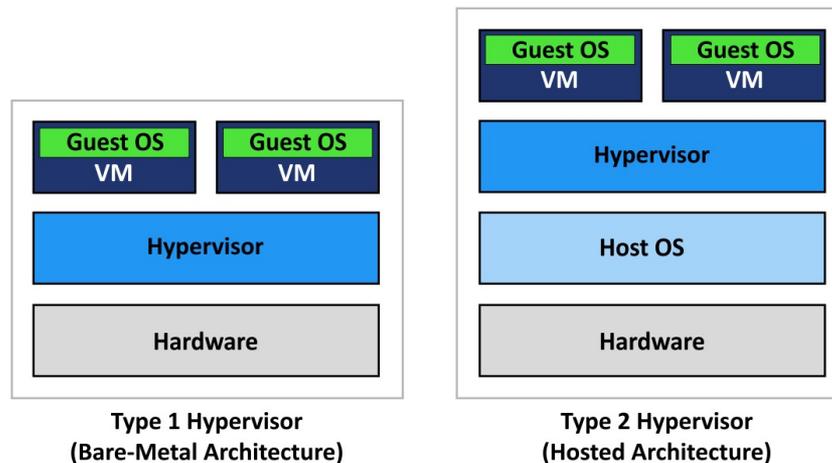
Containers: Low-weight Alternatives to Full-system Virtualization

- Provide OS virtualization above single shared kernel

    - Do not provide *full-machine* virtualization

    - Each VM has illusion of running on isolated machine

    - Each container has illusion of running on isolated OS

- Use OS constructs to provide *sand boxes* for execution

    - E.g., Linux cgroups, namespaces, etc.

- Can run on bare metal OS, or atop of OS running in VM

- OS containers: multiple applications run in same container

    - E.g., LXC, OpenVZ, FreeBSD Jail

- Application containers: each application has its own container

    - E.g., Docker, rkt

What Makes Operating Systems so Exciting and Challenging?
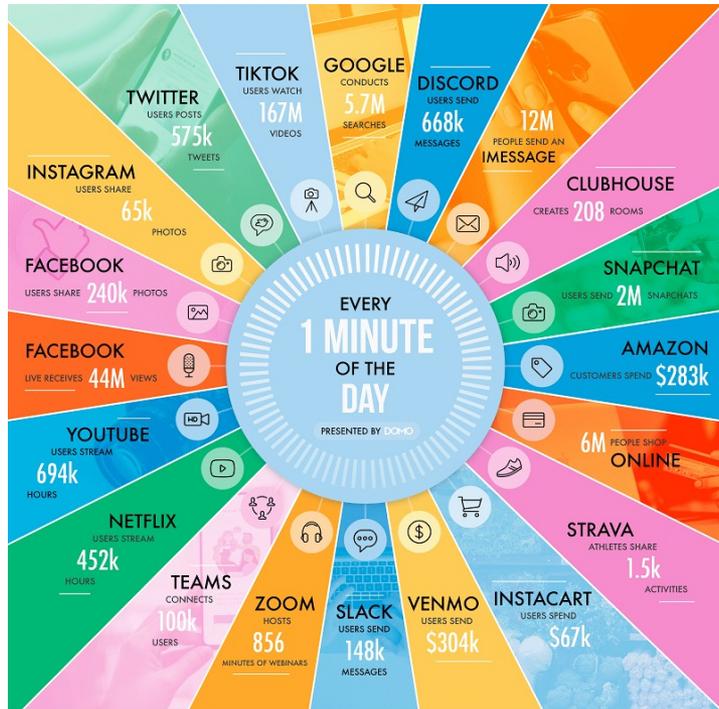
Operating systems are at the heart of it all …

- 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

[Moore's Law 1965]
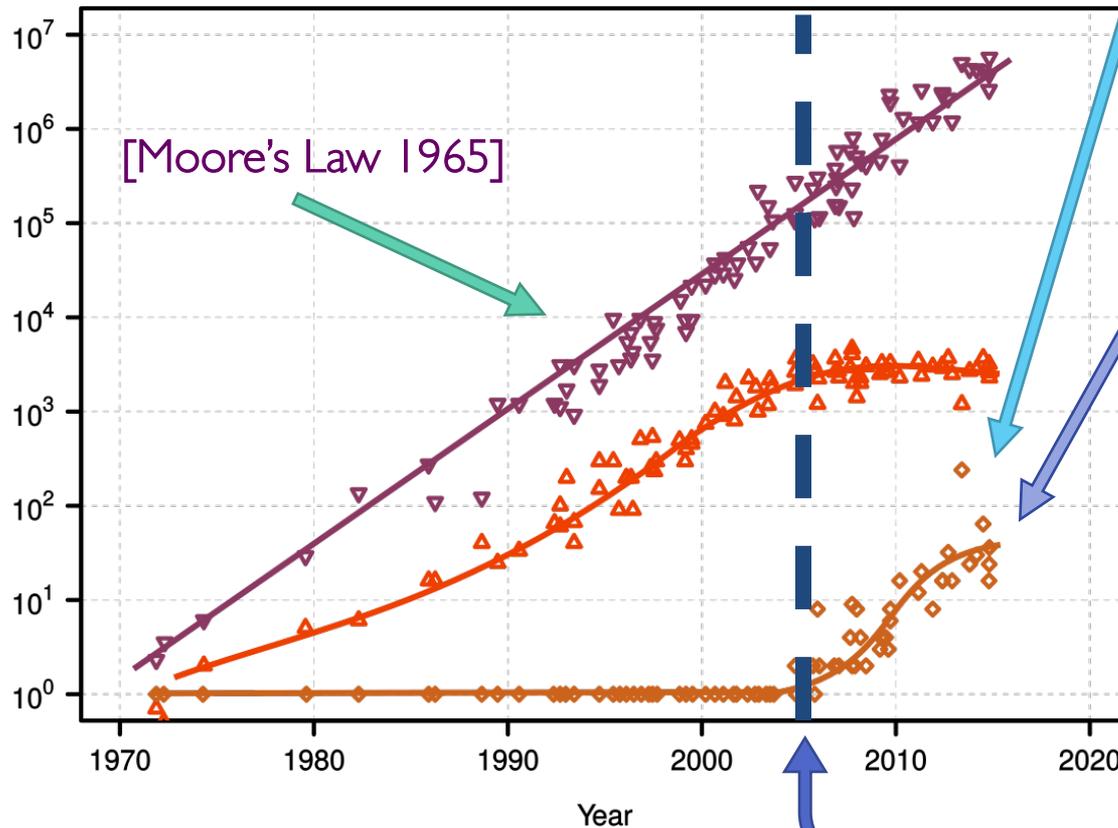
Transistors

Dark Silicon [Esmaeilzadeh et al. 2011]
Multiprocessor melts if all cores
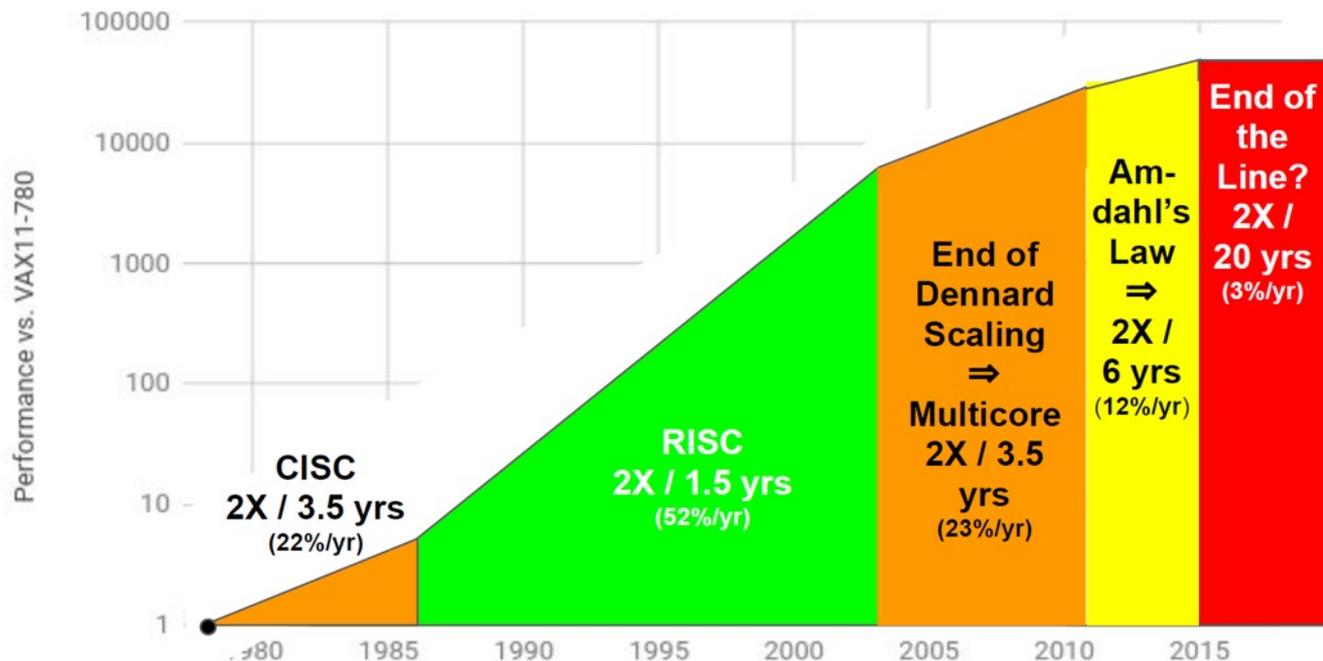simultaneously run at full capacity

Frequency (MHz)

Number of Cores

End of Dennard Scaling
[R. Dennard et al. 1974]

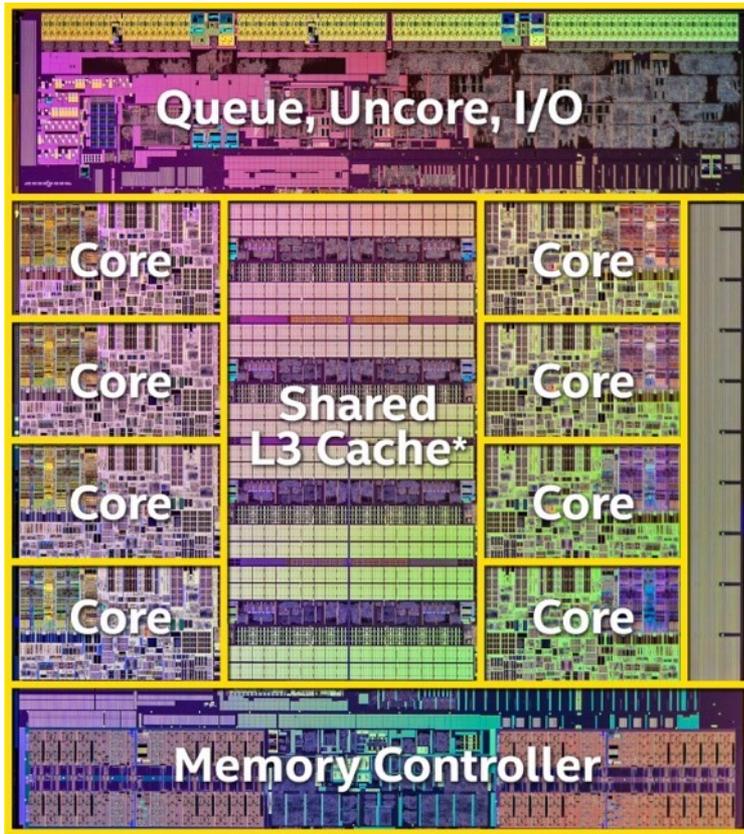# End of Growth of Single Program Speed

40 years of Processor Performance

Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

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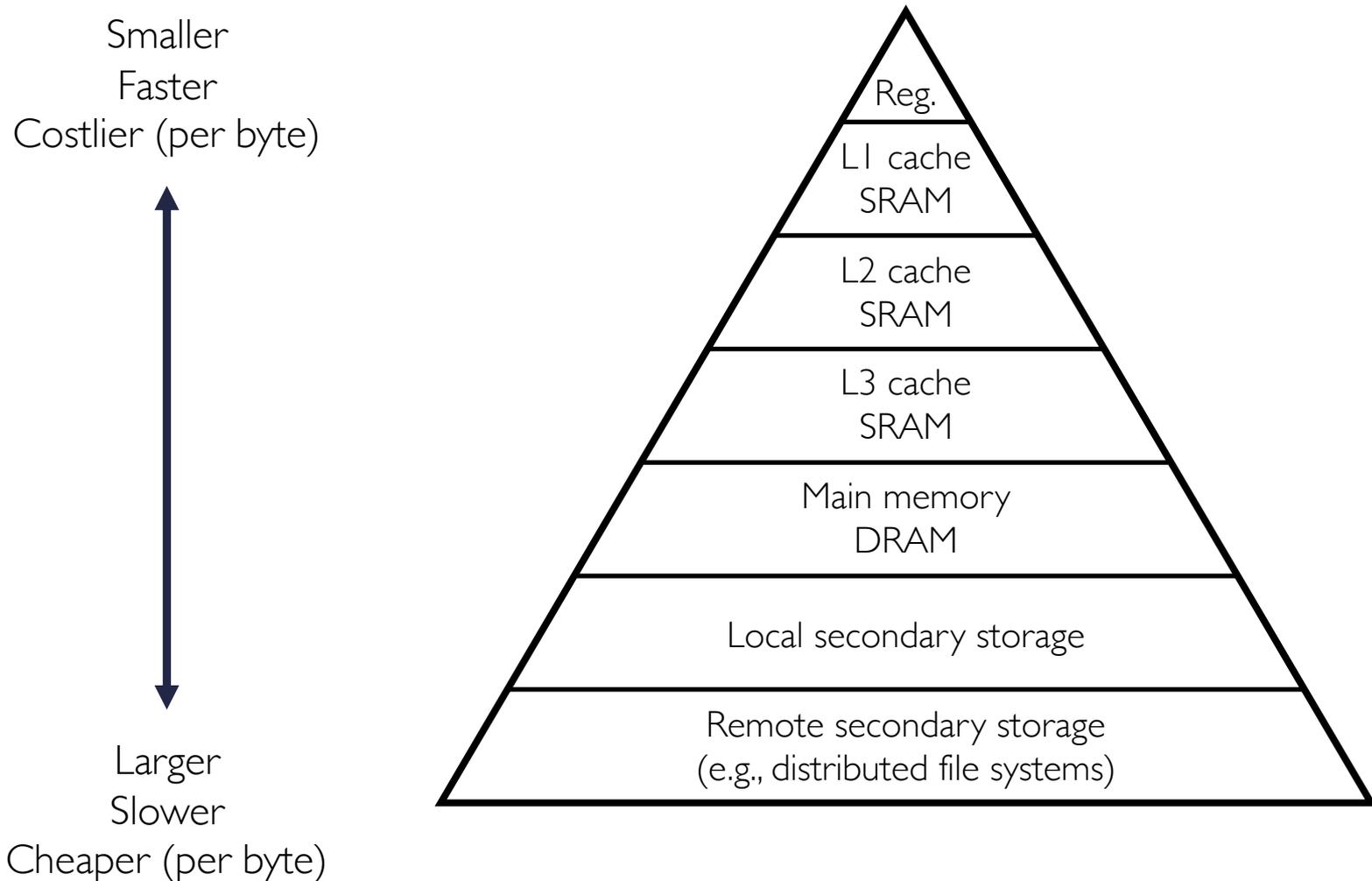
Modern Processors

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)

Reg.

L1 cache
SRAM

L2 cache
SRAM

L3 cache
SRAM

Main memory
DRAM

Local secondary storage

Remote secondary storage
(e.g., distributed file systems)

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
```
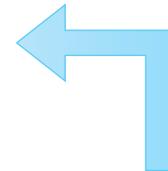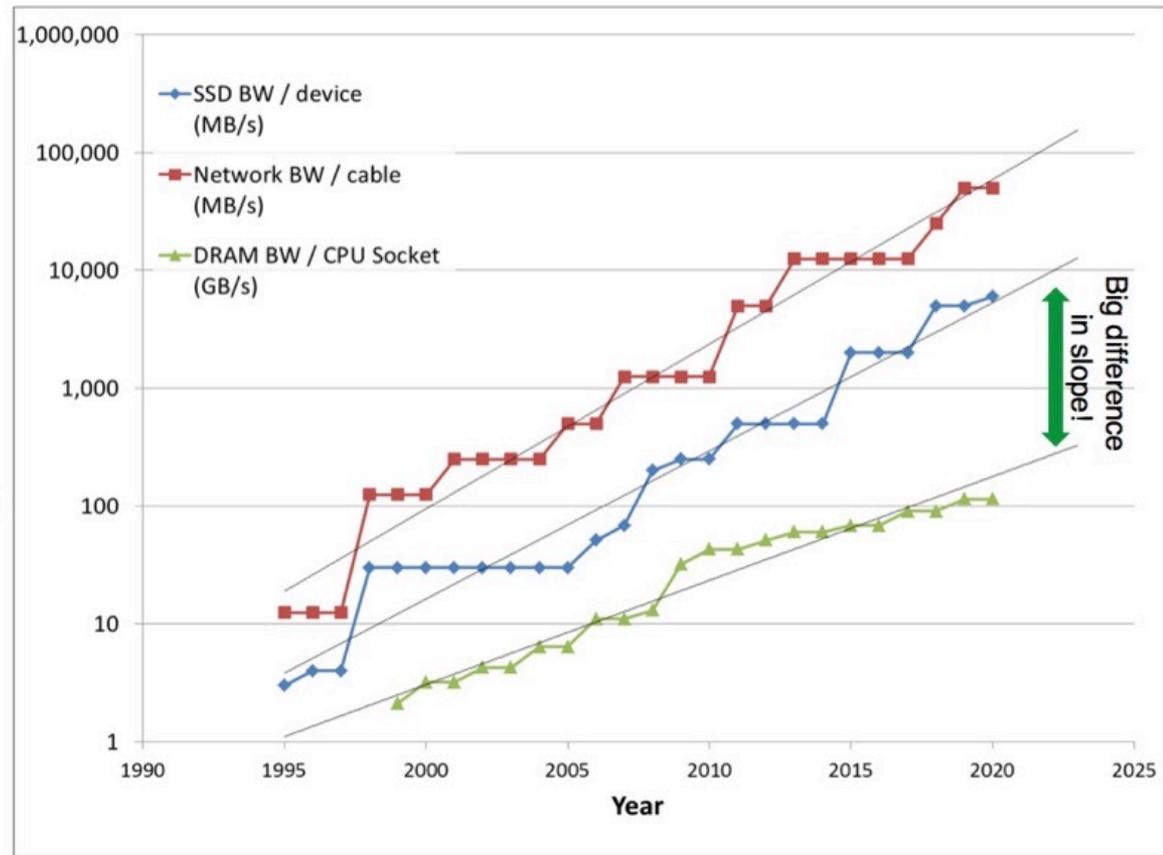
Key stroke ~100 ms

Network, IO, and Memory Bandwidth Trends

People to Computer Ratio Trend

Computers Per Person

01:10^6

01:10^3

01:01

10^3:01

Mainframe

Mini

Workstation

PC

Laptop

PDA

Cell

years

Number crunching, Data Storage, Massive Inet Services, ML, …

Productivity, Interactive

Streaming from/to physical world

Internet of Things!
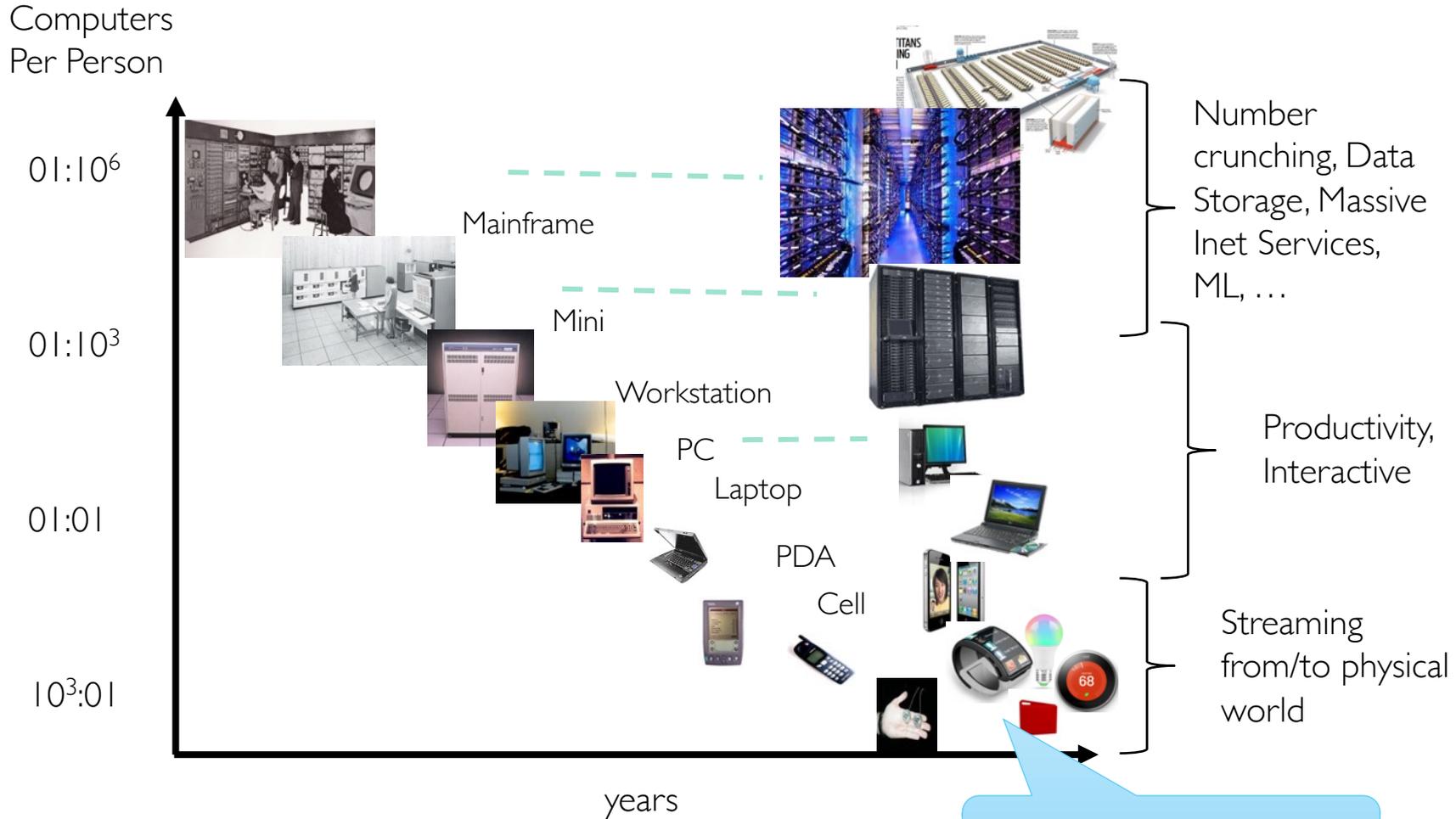
Bell's Law: new computer class per 10 years

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

Generated using https://github.com/udoprog/kernelstats

Legend:
- arch/i386
- arch/other
- arch/x86
- crypto, mm, sound
- drivers/gpu
- drivers/media
- drivers/net
- drivers/other
- fs
- net
- other

Y-axis: Lines of Code in Linux Kernel (05M, 10M, 15M, 20M)
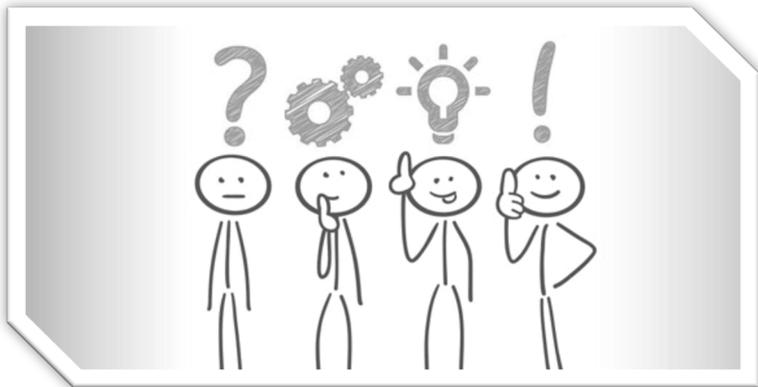
X-axis: Kernel Versions

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

- SE 350 combines ideas and concepts from many other areas of computer science and engineering
  - Languages, data structures, hardware, and algorithms

Questions?

Acknowledgment