

CS162  
Operating Systems and  
Systems Programming  
Lecture I

What is an Operating System?

August 29<sup>th</sup>, 2024

Prof. Ion Stoica

<http://cs162.eecs.Berkeley.edu>

# Instructor: Ion Stoica

---



- Web: <http://www.cs.berkeley.edu/~istoica/>
  - 465 Soda Hall (Sky Computing Lab)
  - Office hour: Tuesdays, 11-12pm
- Research areas:
  - ML/LLM systems (vLLM, Ray, ChatBot Arena, Alpa, ...)
  - Cloud computing (SkyPilot, SkyStorage)
  - Previous: Cluster computing (Apache Mesos, Apache Spark, Alluxio), Peer-to-Peer networking (Chord), Networking QoS
- Industry experience: co-founded
  - Conviva (2006 - ): Video distribution and analytics
  - Databricks (2013 - ): Company behind Apache ~~Spark~~ databricks
  - Ray (2019 - ): Company behind Ray



# CS162 TAs: Sections TBA

---



Diana Poplacenel  
(Head TA)



Jacob Laurence High  
(Head TA)



Sean Yang



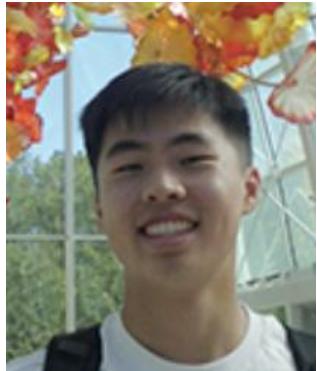
Ryan Almeddine



Sriram Srivatsan



Tushar Goyal



Ethan Zhang



Aarin Kalpeshkumar  
Salot



Ashwin Chugh

# CS162 Readers

---



Aditya Balasubramanian



Rohan Penmatcha



Jason Guo

# Enrollment

---

- This is an Early Drop Deadline course (September 6th)
  - If you are not serious about taking, please drop early
  - Department will continue to admit students as other students drop
  - Really hard to drop afterwards!
    - » Don't forget to keep up with work if you are still on the waitlist!
- On the waitlist/Concurrent enrollment ?
  - All decisions concerning appeals for enrolling in the course are made by CS departmental staff; the course staff have no say in the matter. So please do not e-mail us about waitlist or concurrent enrollment.
  - If people drop, others will be moved off the waitlist

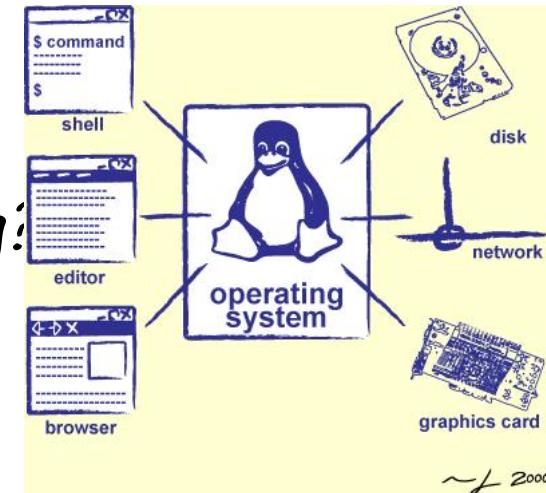
# Goals for Today

---

- What is an Operating System?  
-And - what is it not?
- What makes Operating Systems so exciting:
- “How does this class operate?”

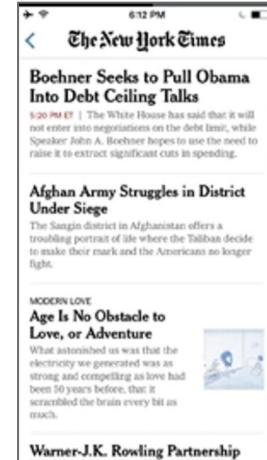
Interactive is important!

Ask Questions!

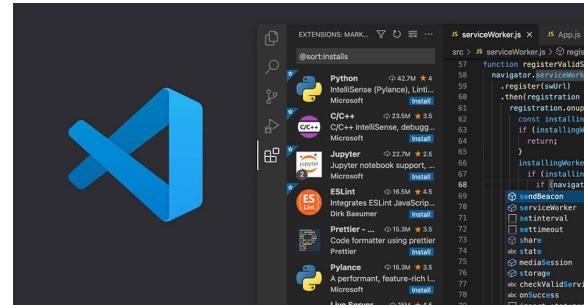


Slides courtesy of David Culler, Natacha Crooks, Anthony D. Joseph, John Kubiatowicz, AJ Shankar, Alex Aiken, Eric Brewer, Ras Bodik, Doug Tygar, and David Wagner.

# What is an operating system?



OS (e.g., iOS, Android)



OS (e.g., MacOS, Linux, Windows)

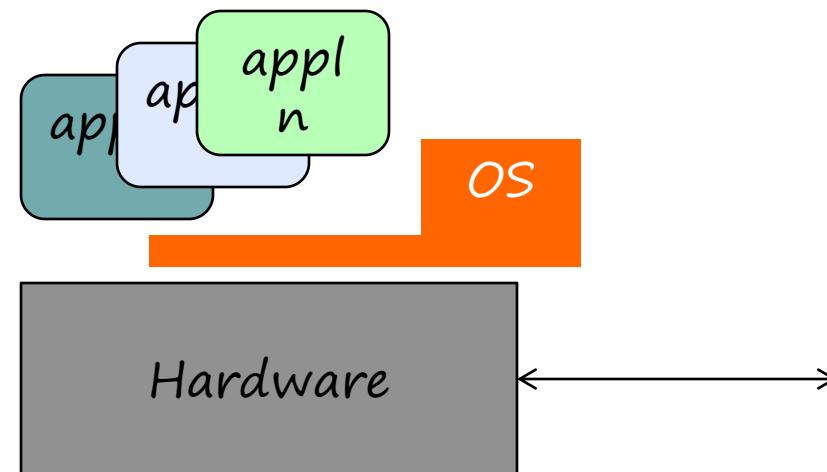


# What is an operating system?

---

Special layer of software that provides application software access to hardware resources

- Convenient abstraction of complex hardware devices
- Protected access to shared resources
- Security and authentication
- Communication amongst logical entities



# What Does an OS do?

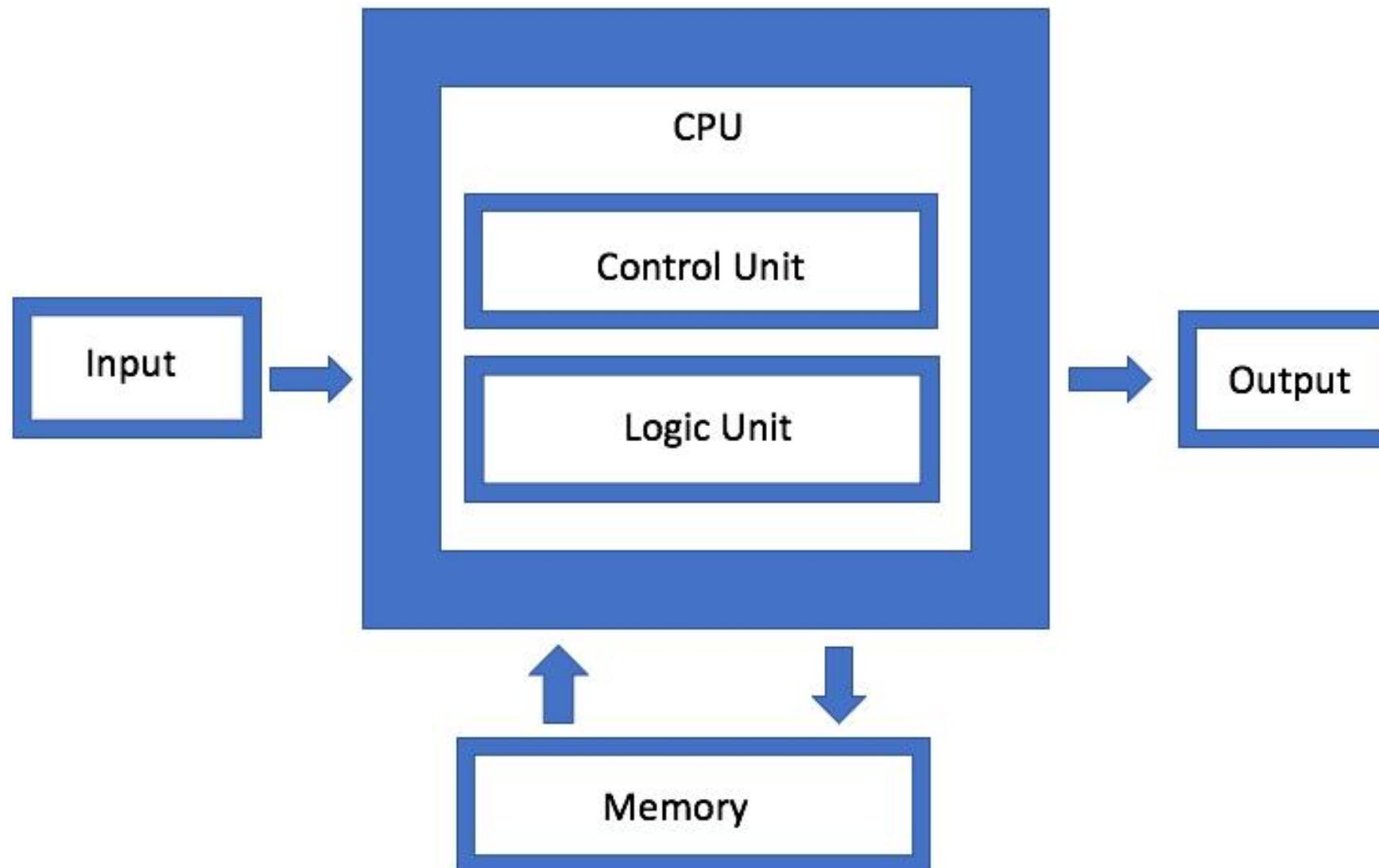
---

- Provide abstractions to apps
  - File systems
  - Processes, threads
  - VM, containers
  - Naming system
  - ...
- Manage resources:
  - Memory, CPU, storage, ...
- Achieves the above by implementing specific algos and techniques:
  - Scheduling
  - Concurrency
  - Transactions
  - Security
  - ....

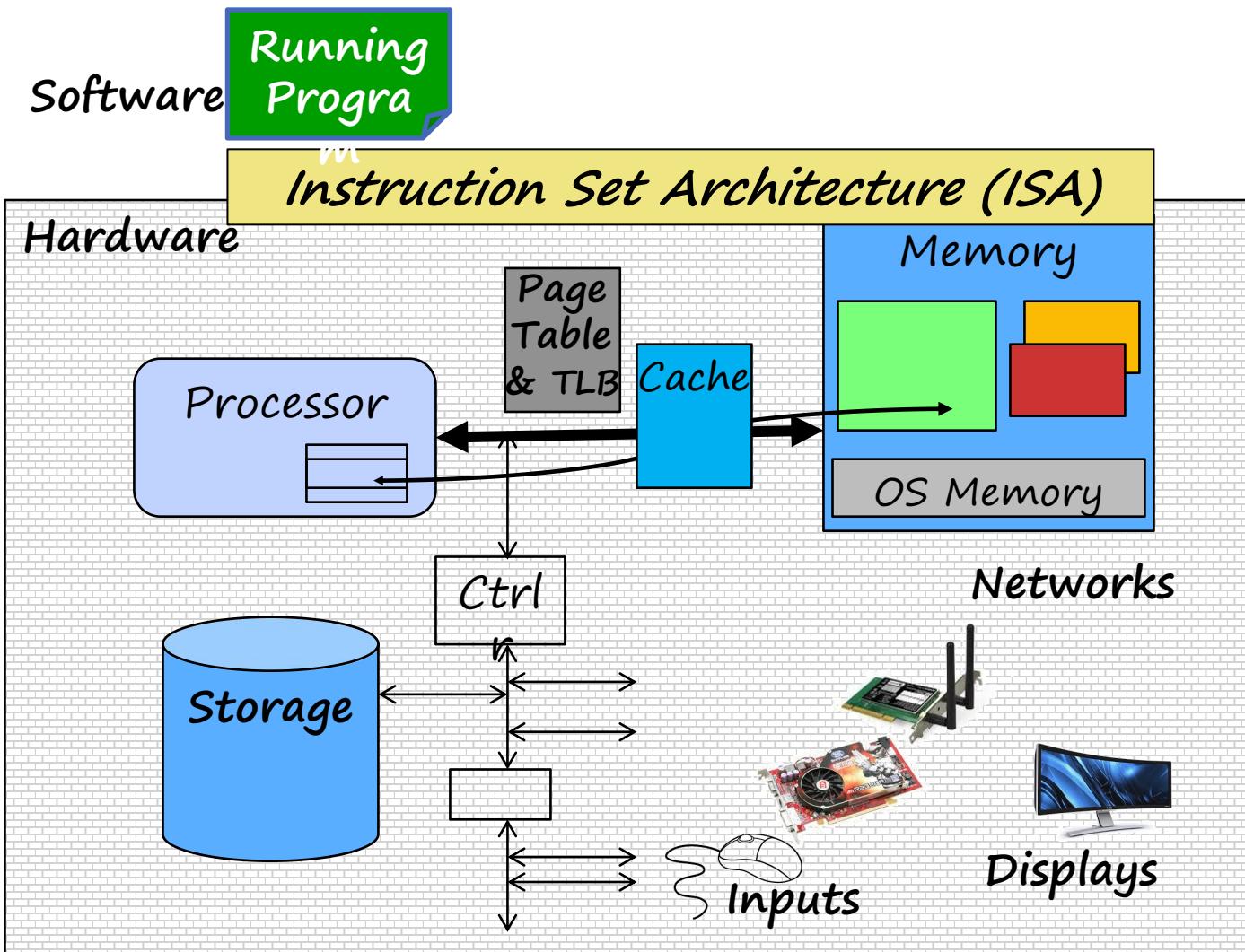
# Von Neumann Architecture



John von Neuman  
(1903 – 1957)



# Hardware/Software Interface



What you learned in  
CS 61C – Machine  
Structures (and C)

The OS abstracts  
these hardware  
details from the  
application

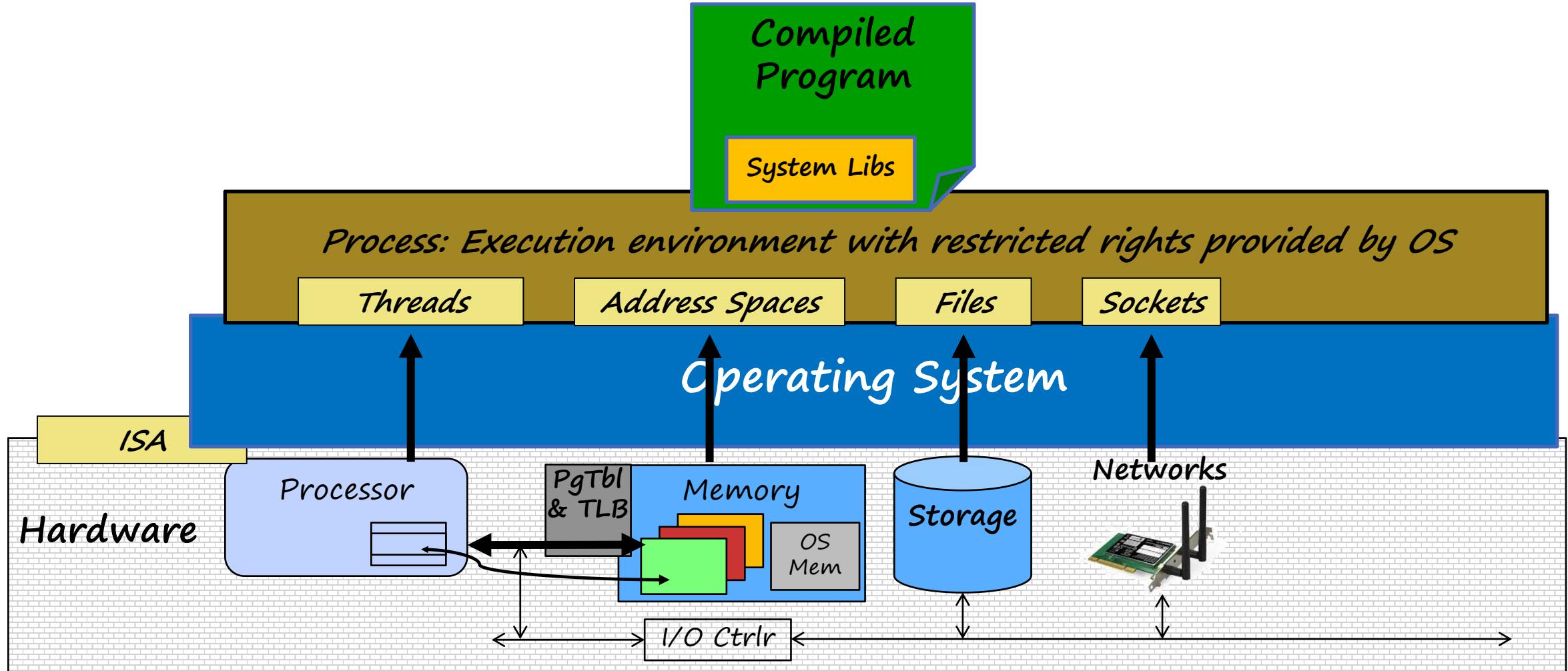
# What is an Operating System?

---

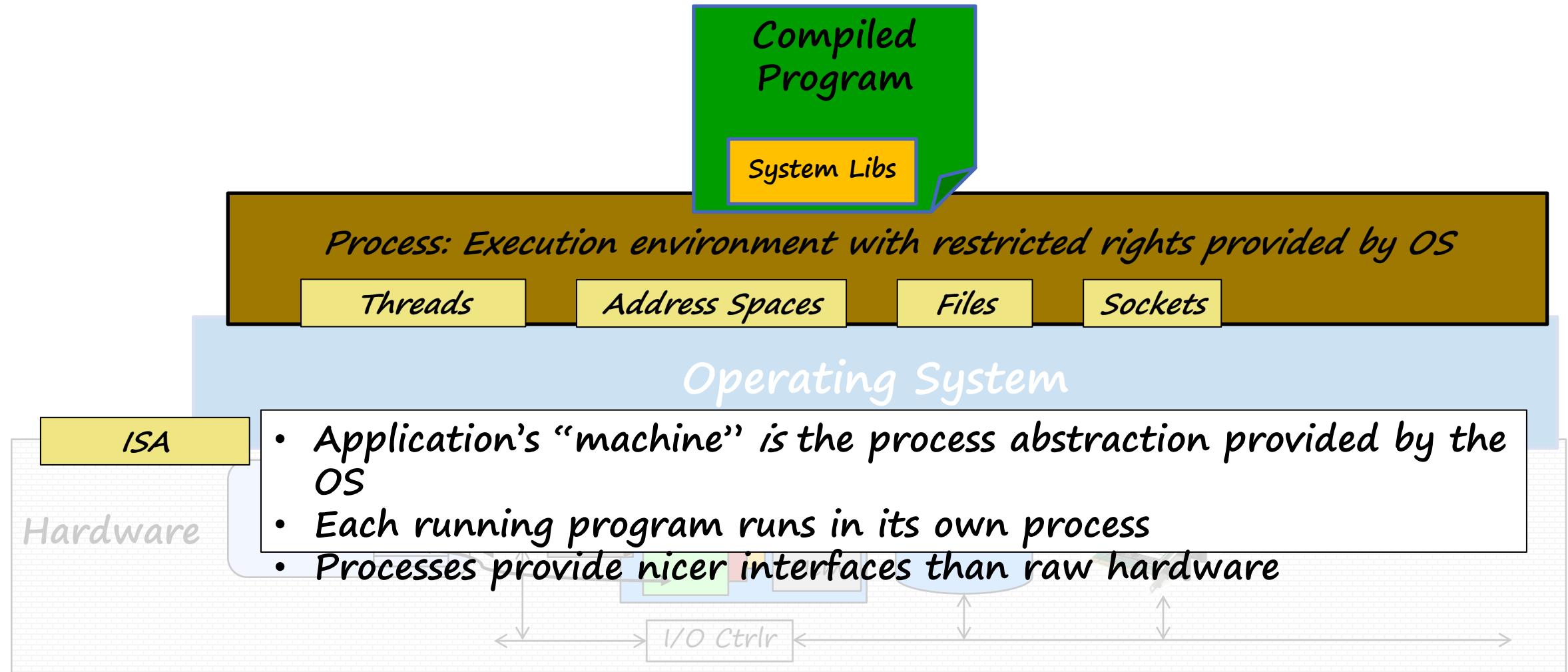


- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - >> Infinite memory, dedicated machine
    - >> Higher level objects: files, users, messages
    - >> Masking limitations, virtualization

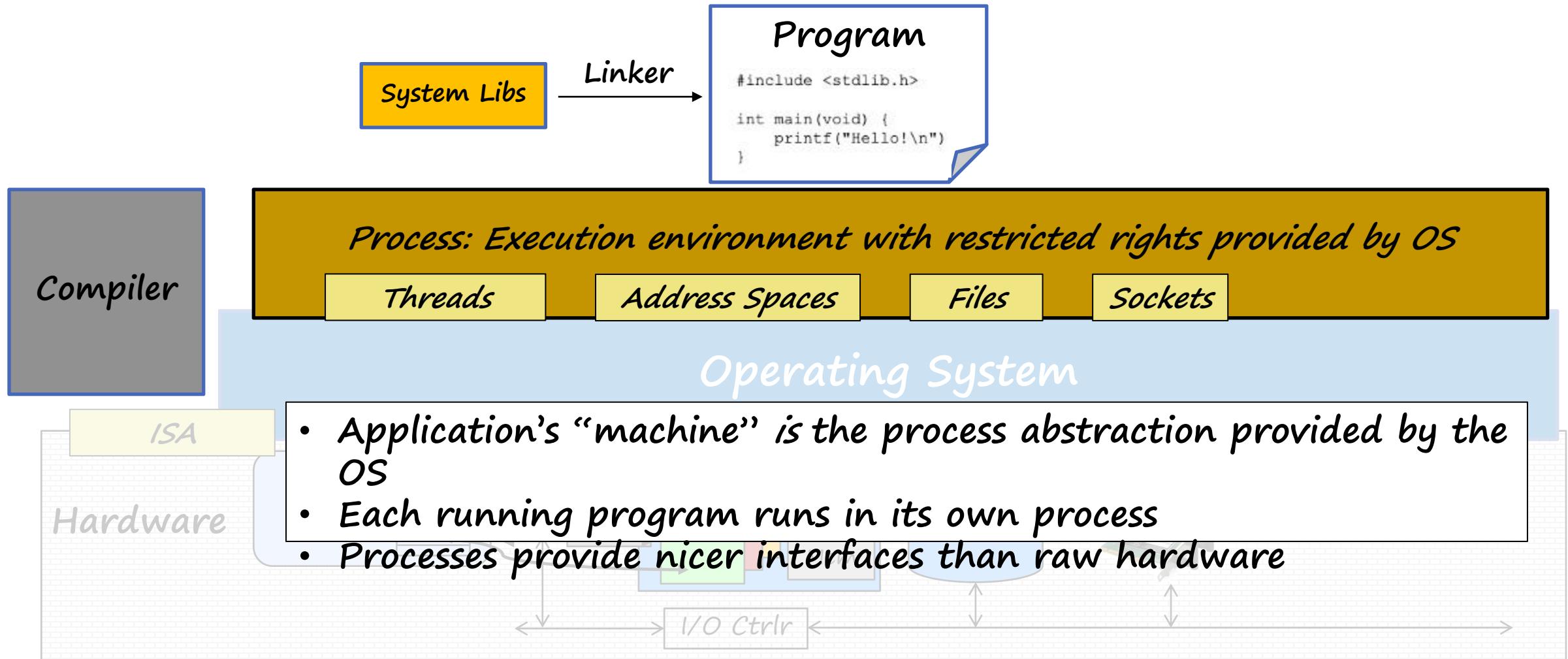
# OS Basics: Virtualizing the Machine



# Compiled Program's View of the World



# System Programmer's View of the World



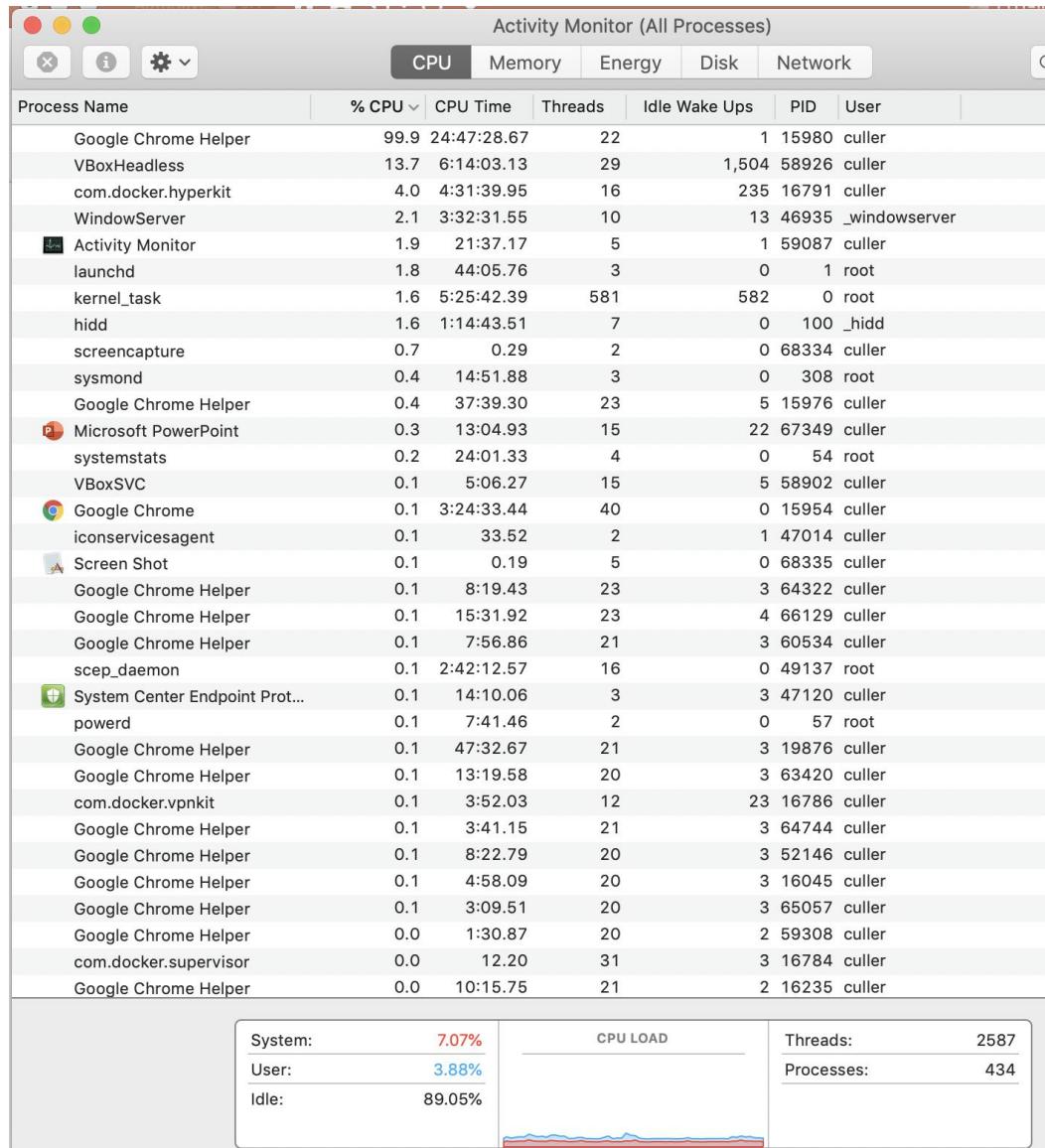
# What's in a Process?

---

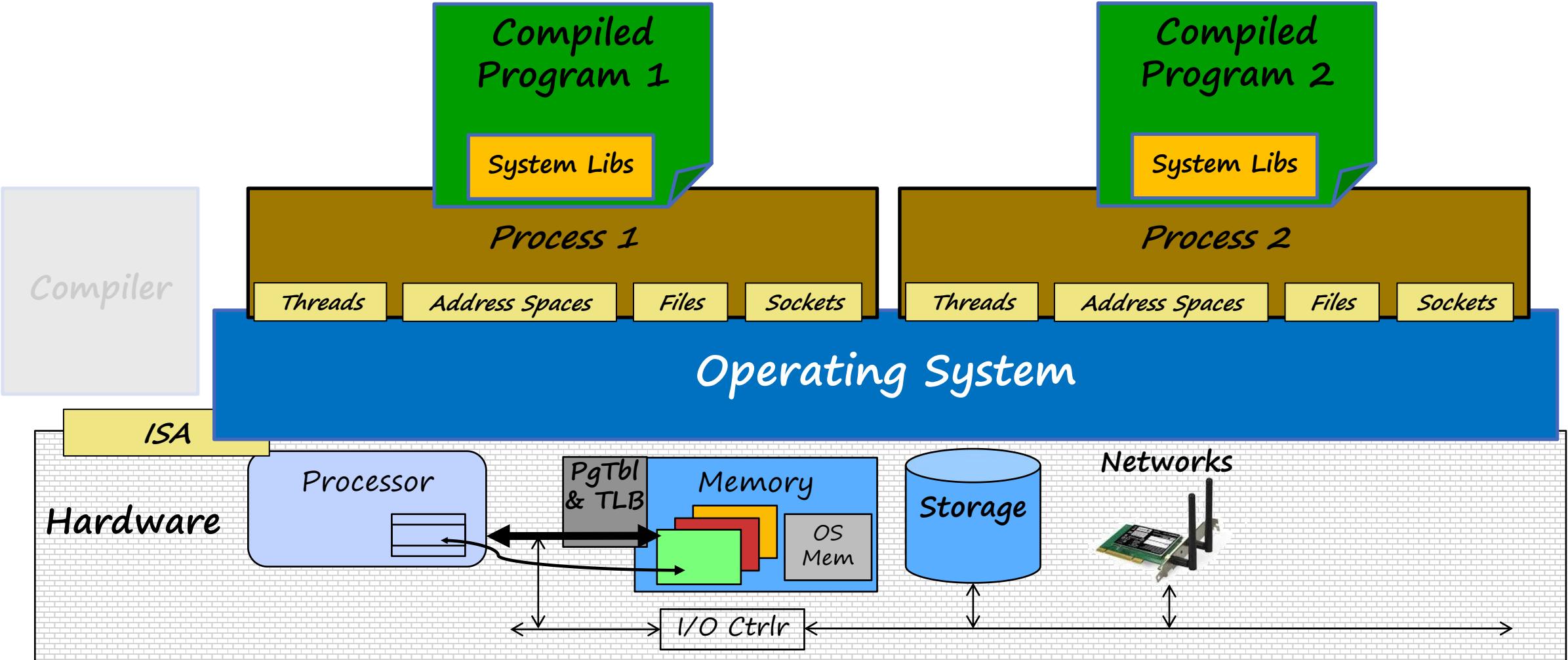
A process consists of:

- Address Space
- One or more threads of control executing in that address space
- Additional system state associated with it
  - Open files
  - Open sockets (network connections)
  - ...

# For Example...

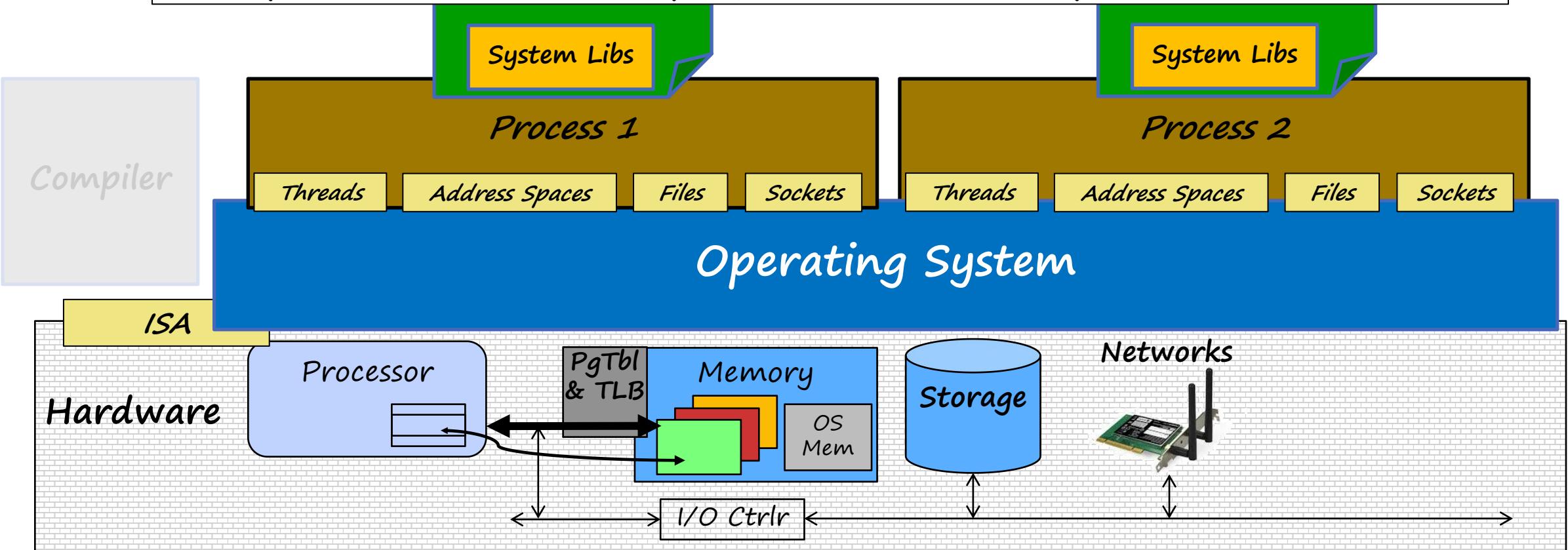


# Operating System's View of the World



# Operating System's View of the World

- OS translates from hardware interface to application interface
- OS provides each running program with its own process



# What is an Operating System?

---

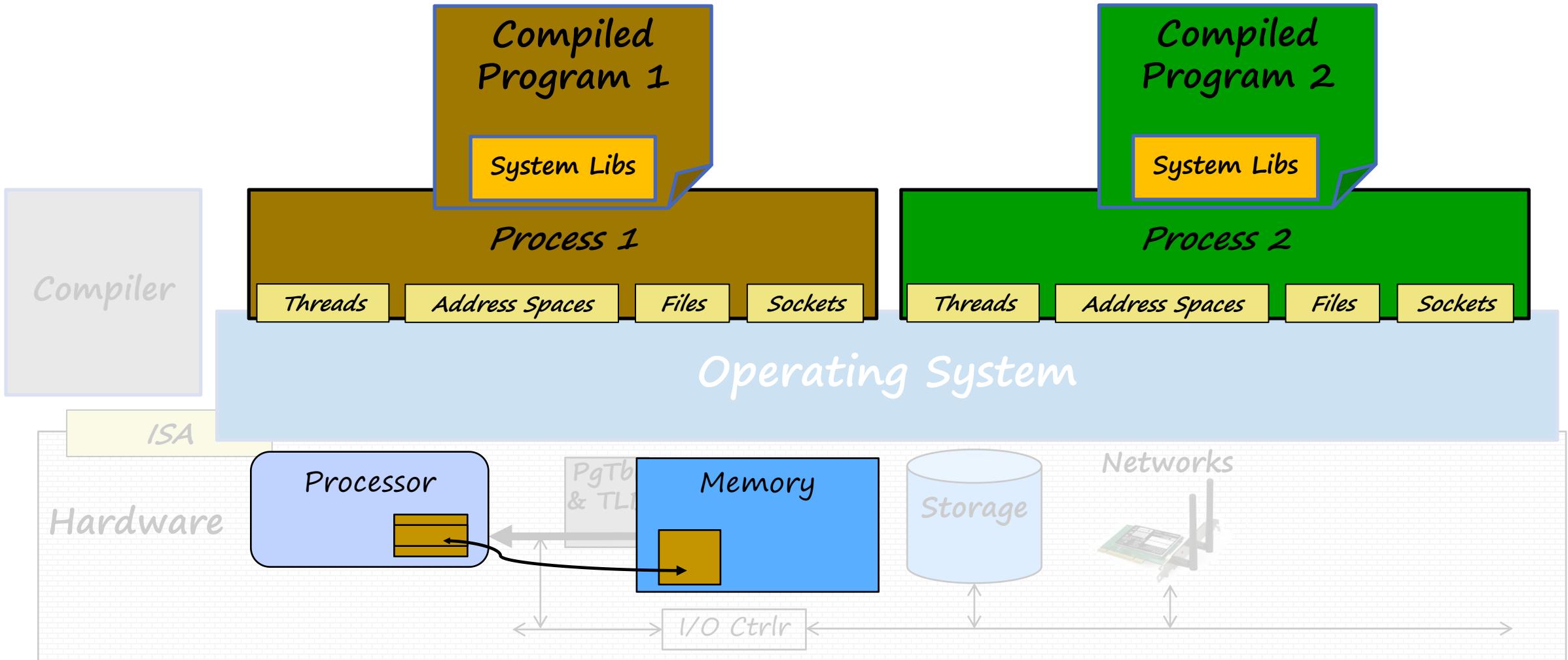


- Referee
  - Manage protection, isolation, and sharing of resources
    - >> Resource allocation and communication

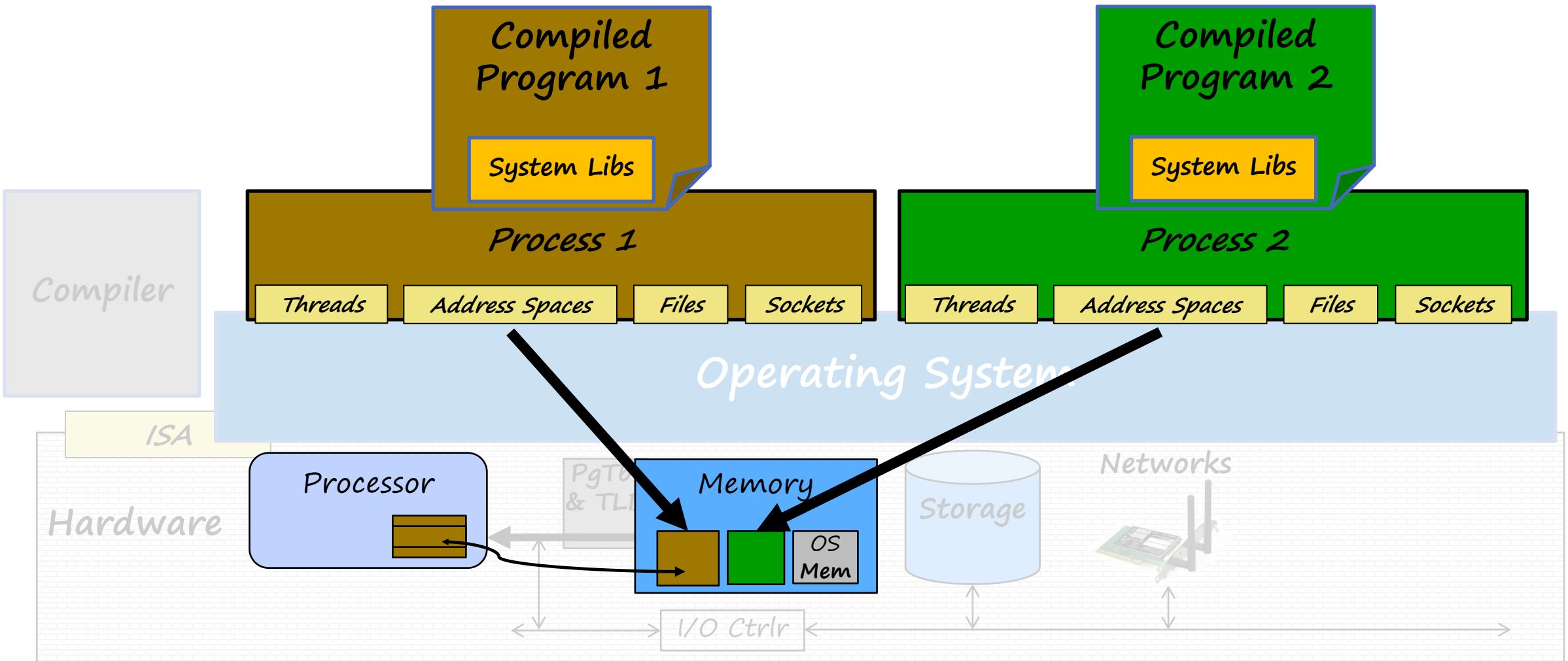


- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - >> Infinite memory, dedicated machine
    - >> Higher level objects: files, users, messages
    - >> Masking limitations, virtualization

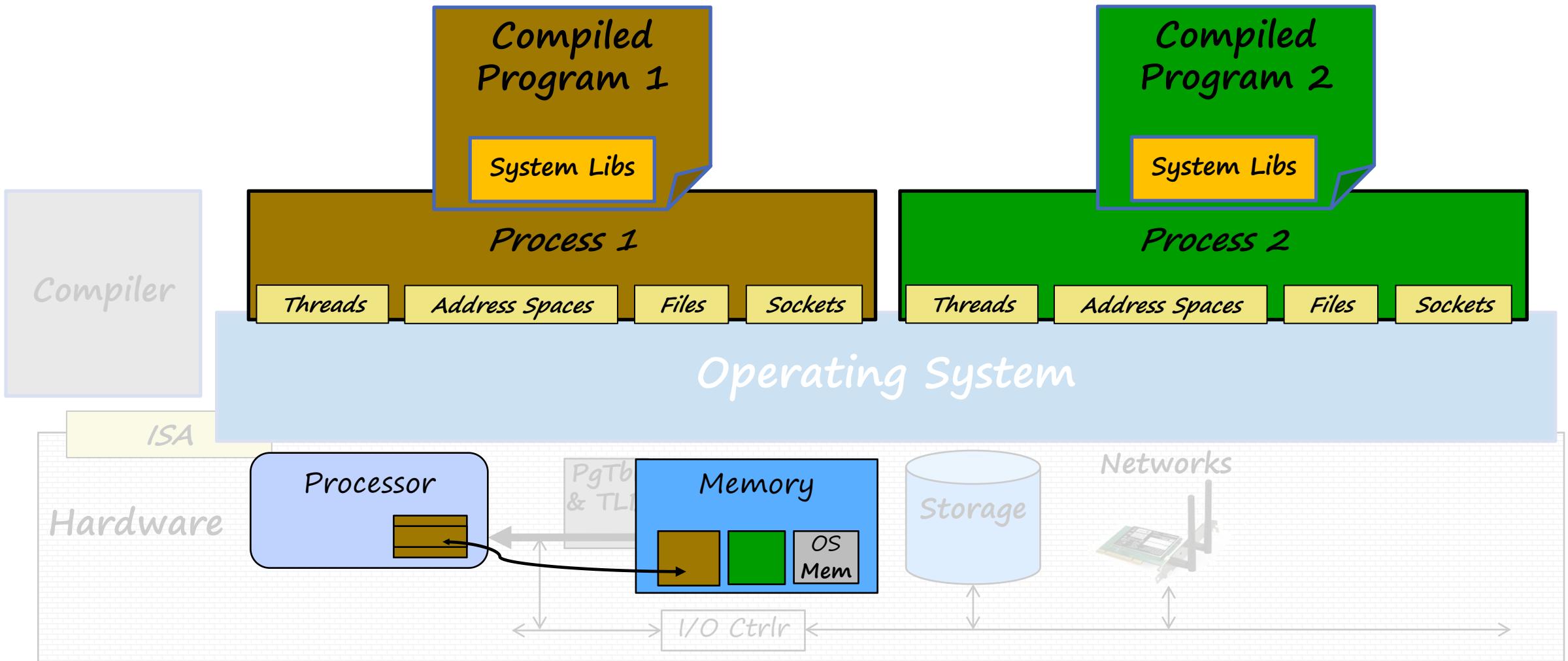
# OS Basics: Running a Process



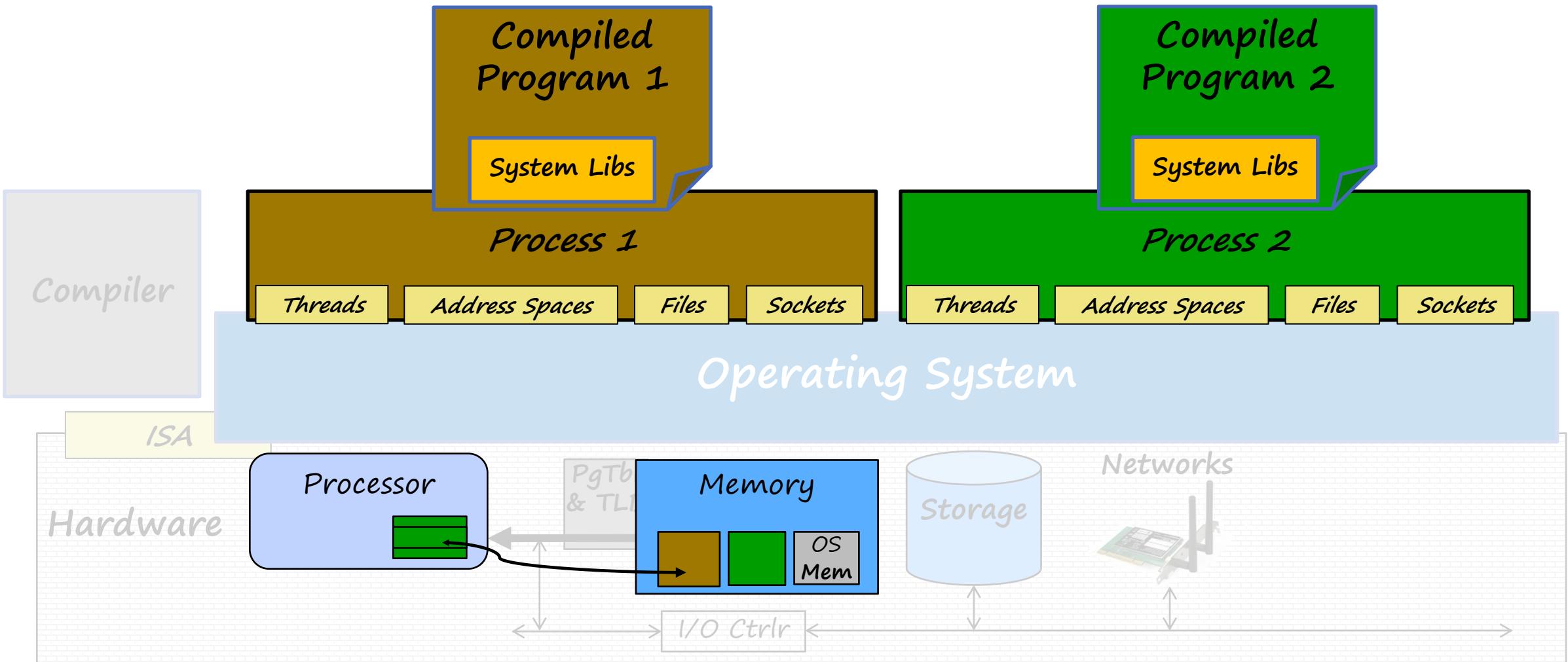
# OS Basics: Switching Processes



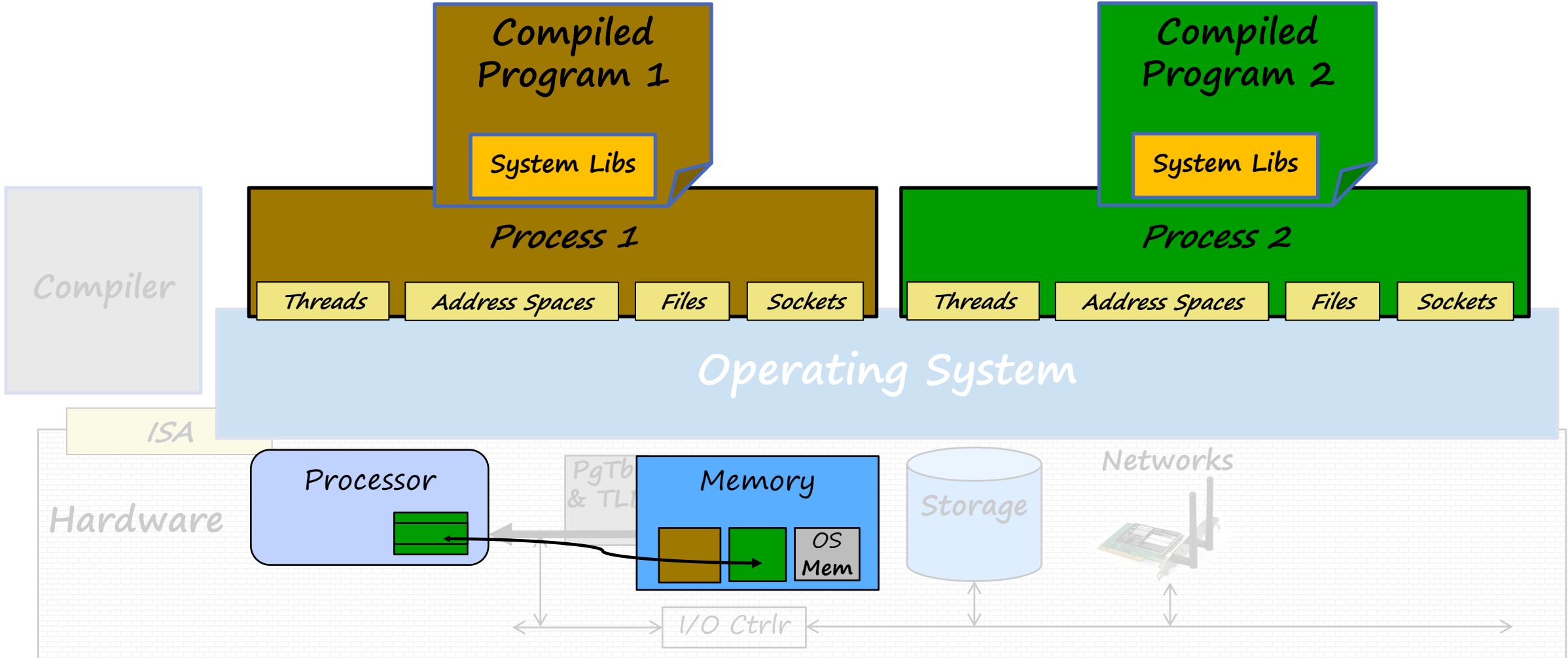
# OS Basics: Switching Processes



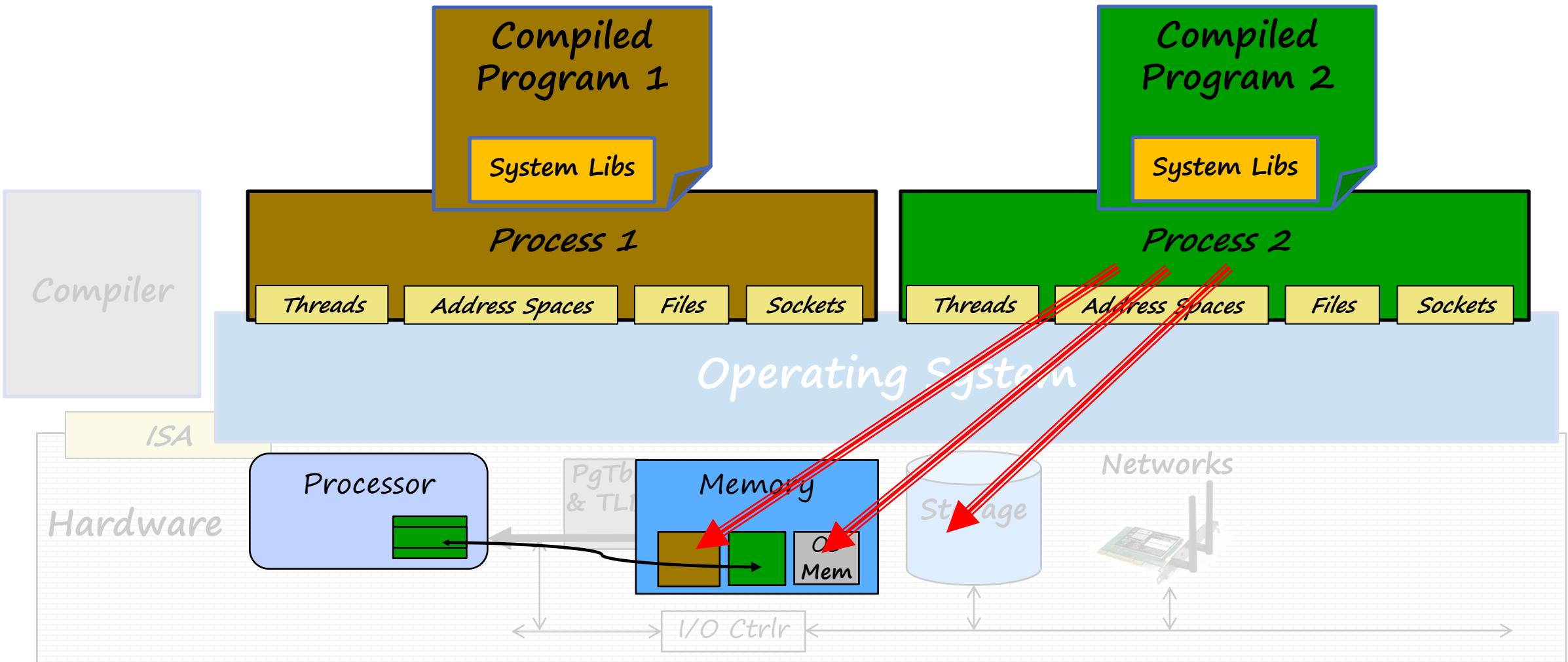
# OS Basics: Switching Processes



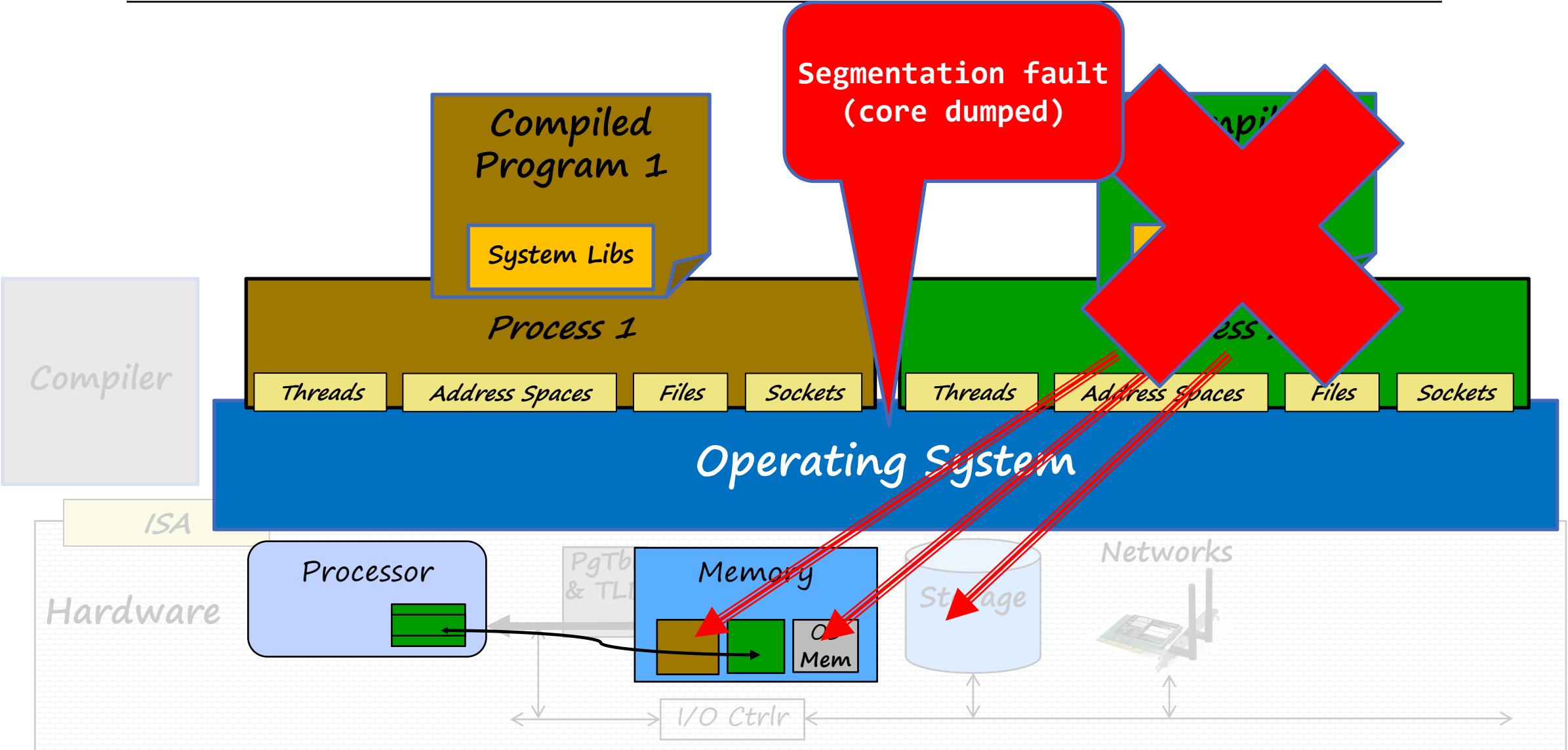
# OS Basics: Switching Processes



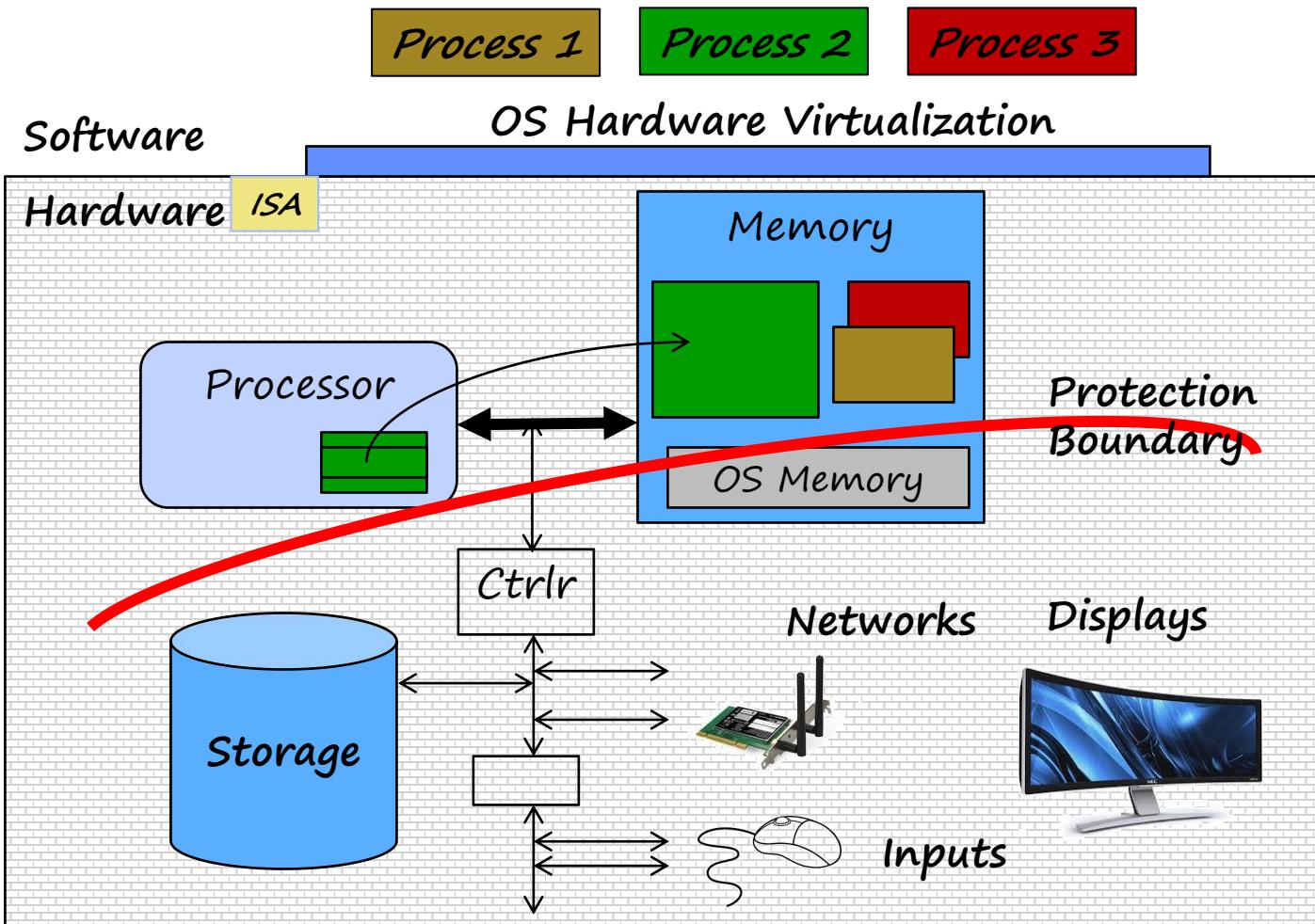
# OS Basics: Protection



# OS Basics: Protection



# OS Basics: Protection



- OS isolates processes from each other
- OS isolates itself from other processes
- ... even though they are actually running on the same hardware!

# What is an Operating System?

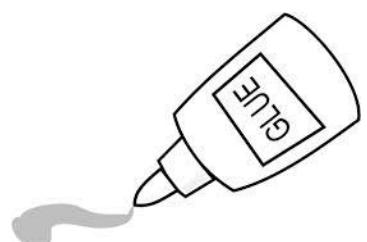
---



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

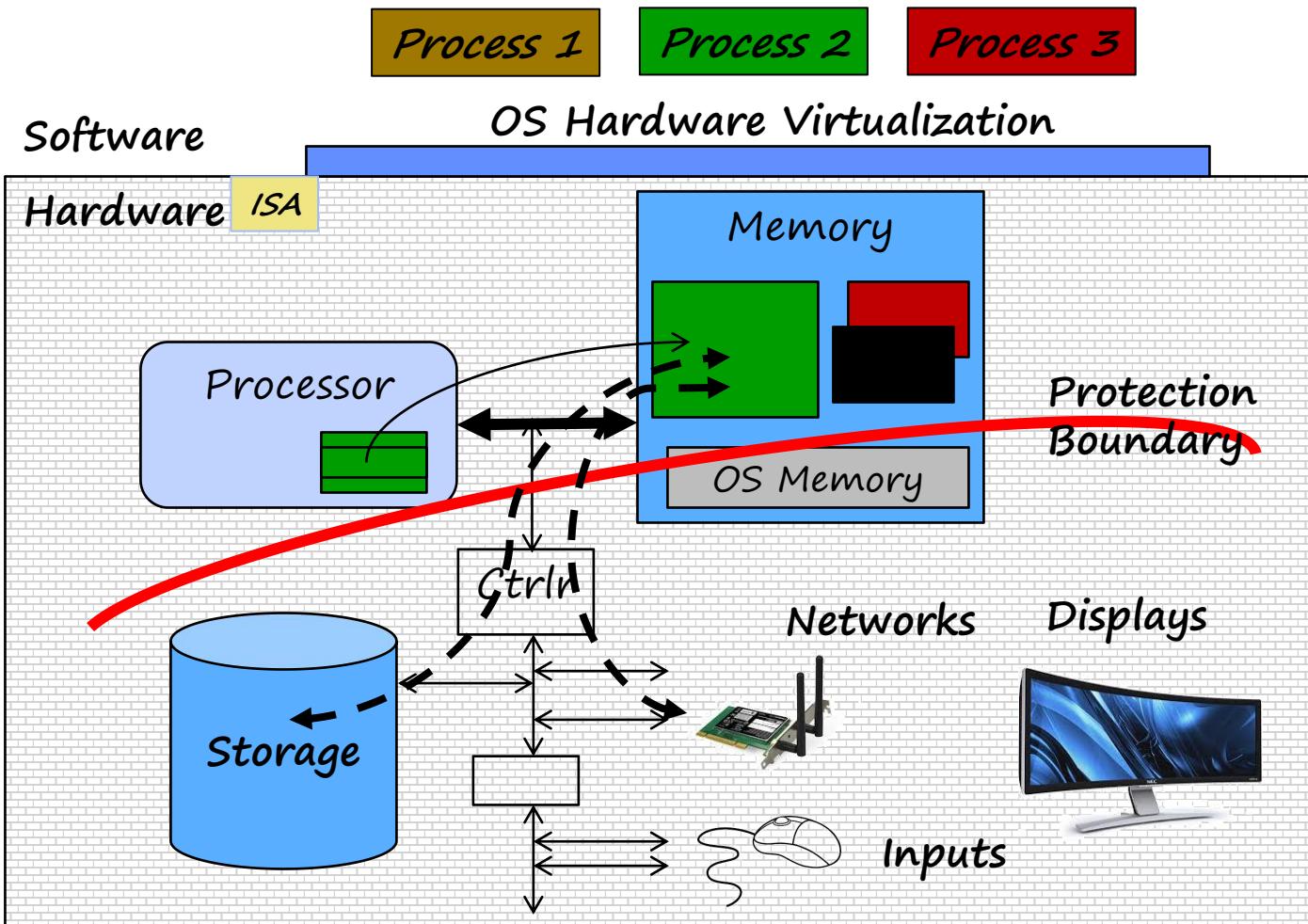


- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    - >> Infinite memory, dedicated machine
    - >> Higher level objects: files, users, messages
    - >> Masking limitations, virtualization



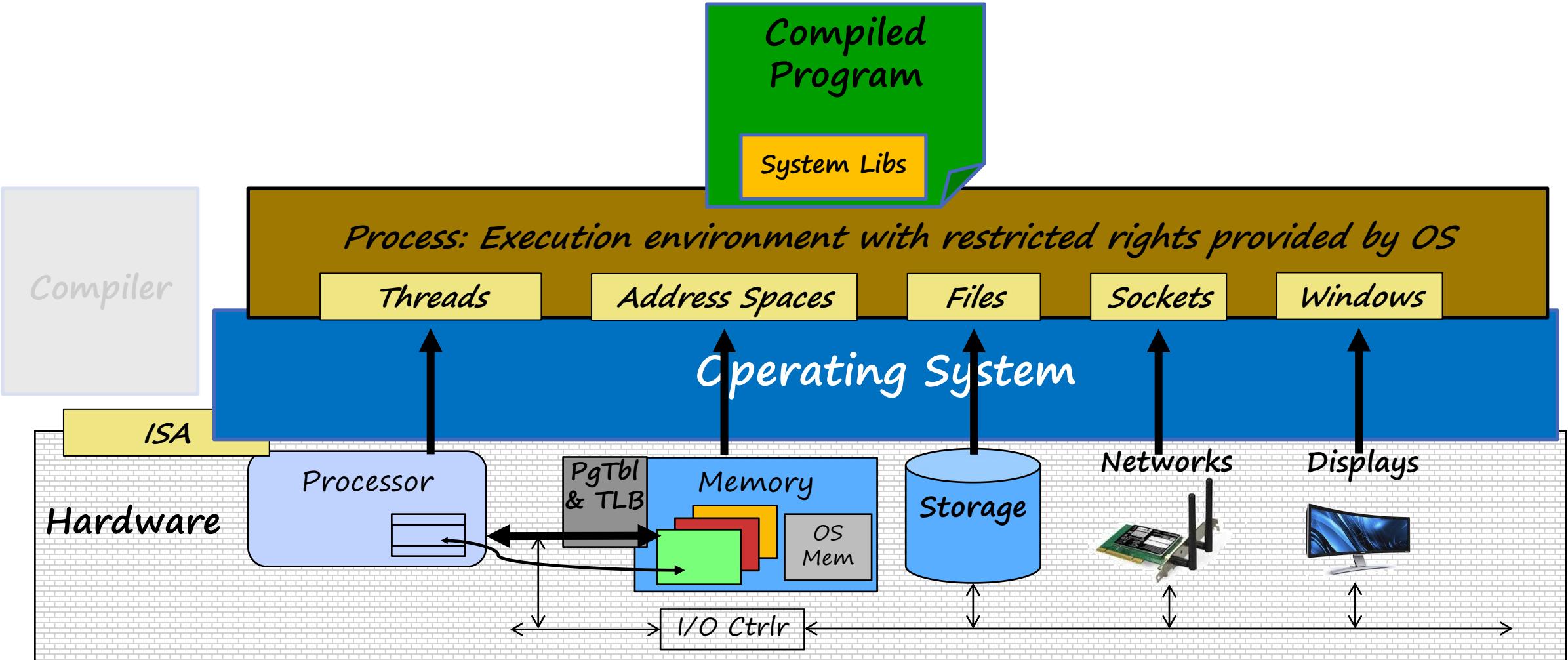
- Glue
  - Common services
    - >> Storage, Window system, Networking
    - >> Sharing, Authorization
    - >> Look and feel

# OS Basics: I/O

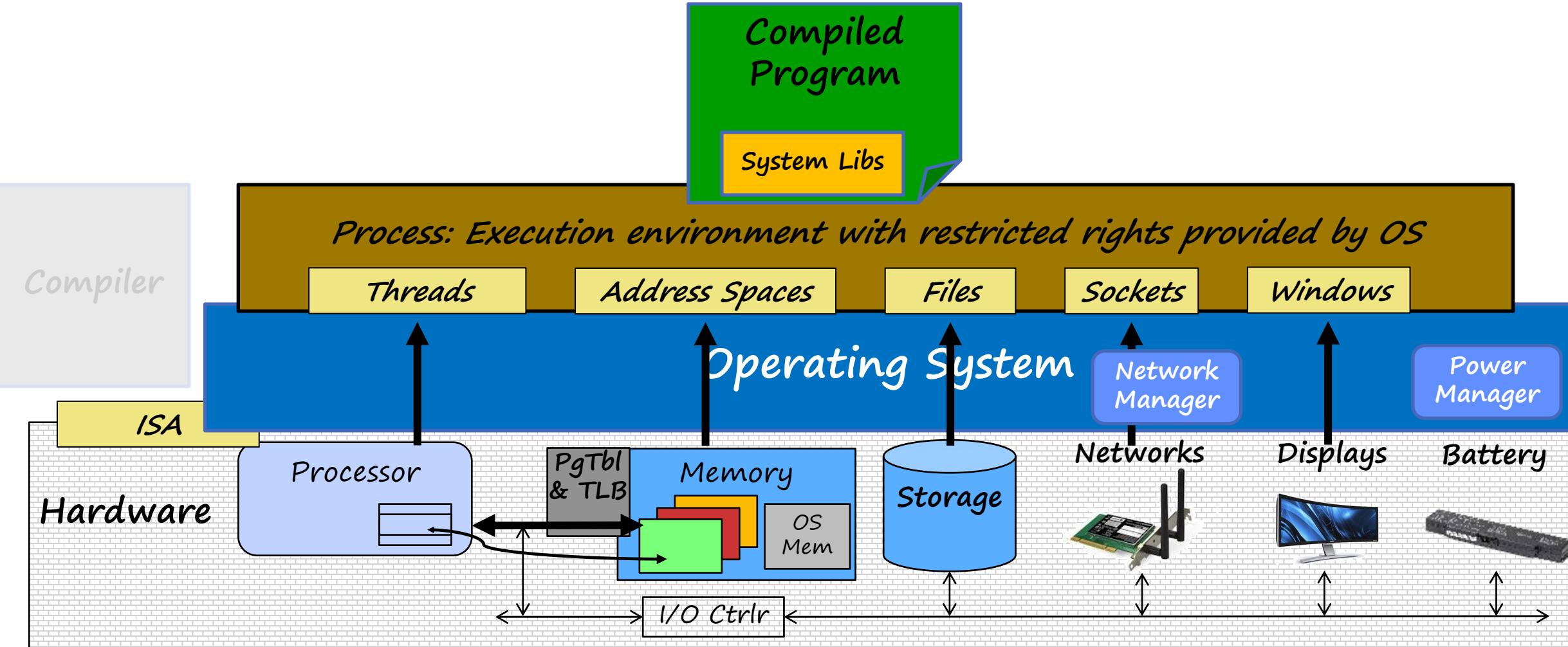


- *OS provides common services in the form of I/O*

# OS Basics: Look and Feel



# OS Basics: Background Management



# What is an Operating System?

---



- 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

# Why take CSI62?

---

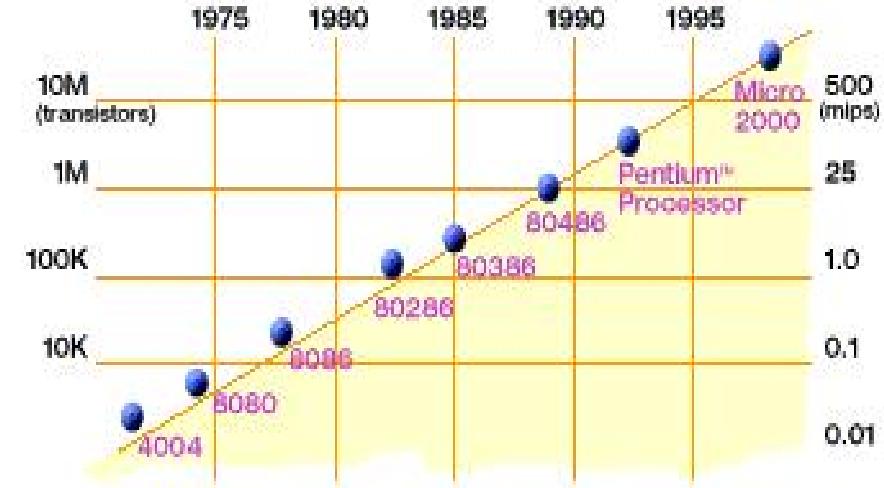
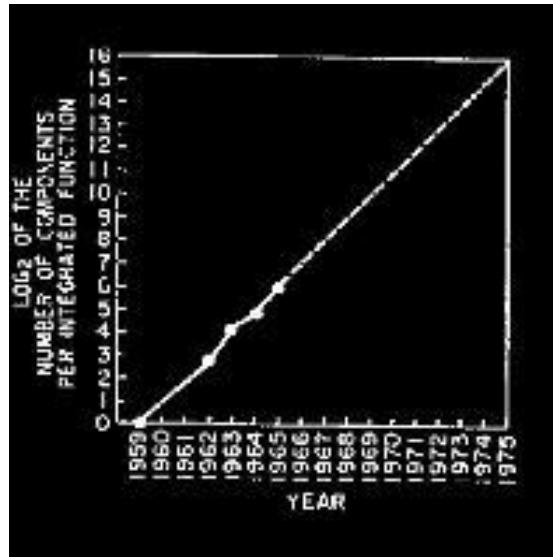
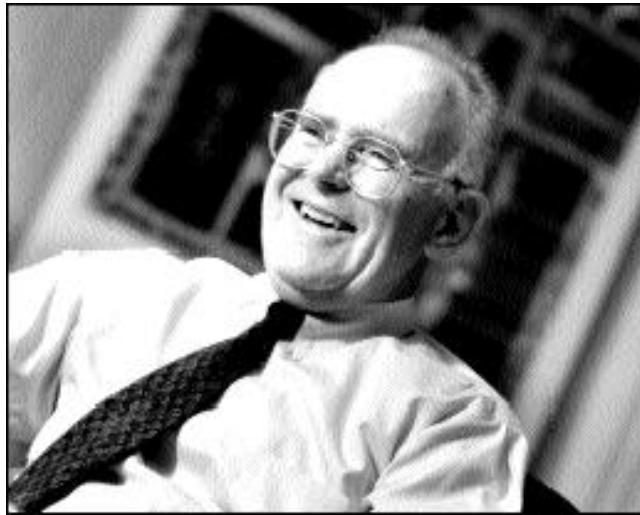
- Some of you will actually design and build operating systems or components of them
- Many of you will create systems that utilize the core concepts in operating systems.
  - Whether you build software or hardware
  - The concepts and design patterns appear at many levels
- All of you will build applications, etc. that utilize operating systems
  - The better you understand their design and implementation, the better use you'll make of them.

---

---

What makes Operating  
Systems Exciting and  
Challenging?

# Technology Trends: Moore's Law



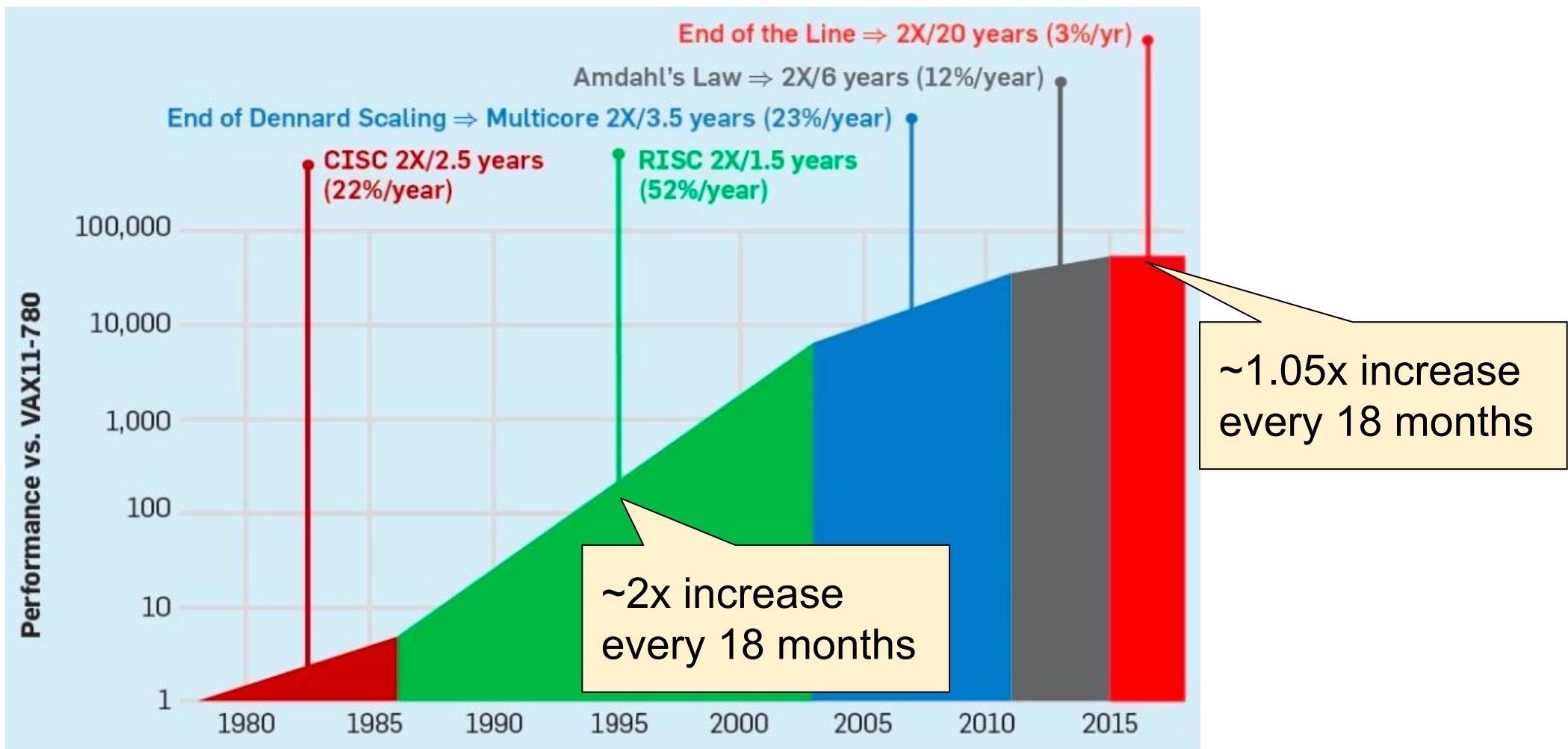
2X transistors/Chip Every 2 years

Called "Moore's Law"

Gordon Moore (co-founder of Intel) predicted in 1965 that the **transistor density** of semiconductor chips **would double roughly every 2 years**

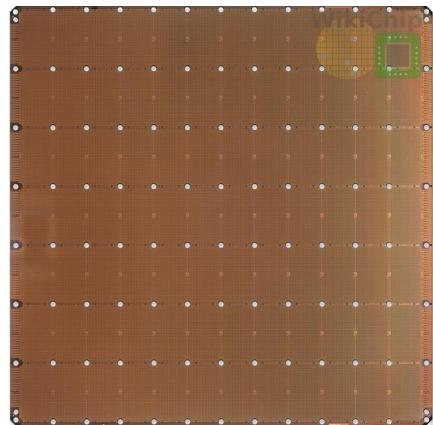
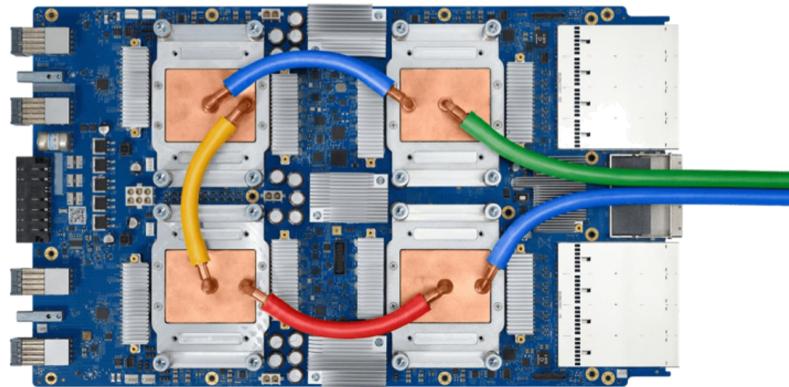
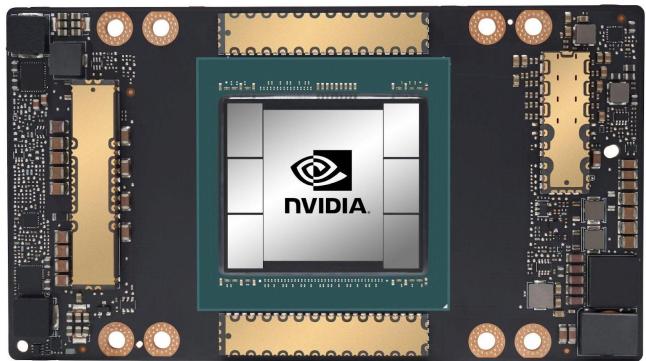
- Microprocessors have become smaller, denser, and more powerful
- Corollary: Performance double roughly every 1.5 years** (18 months)
- Faster growth because transistors are closer to each other so electrical signals travel faster

# Moore's Law coming to an end



# Specialized processors to the rescue

---



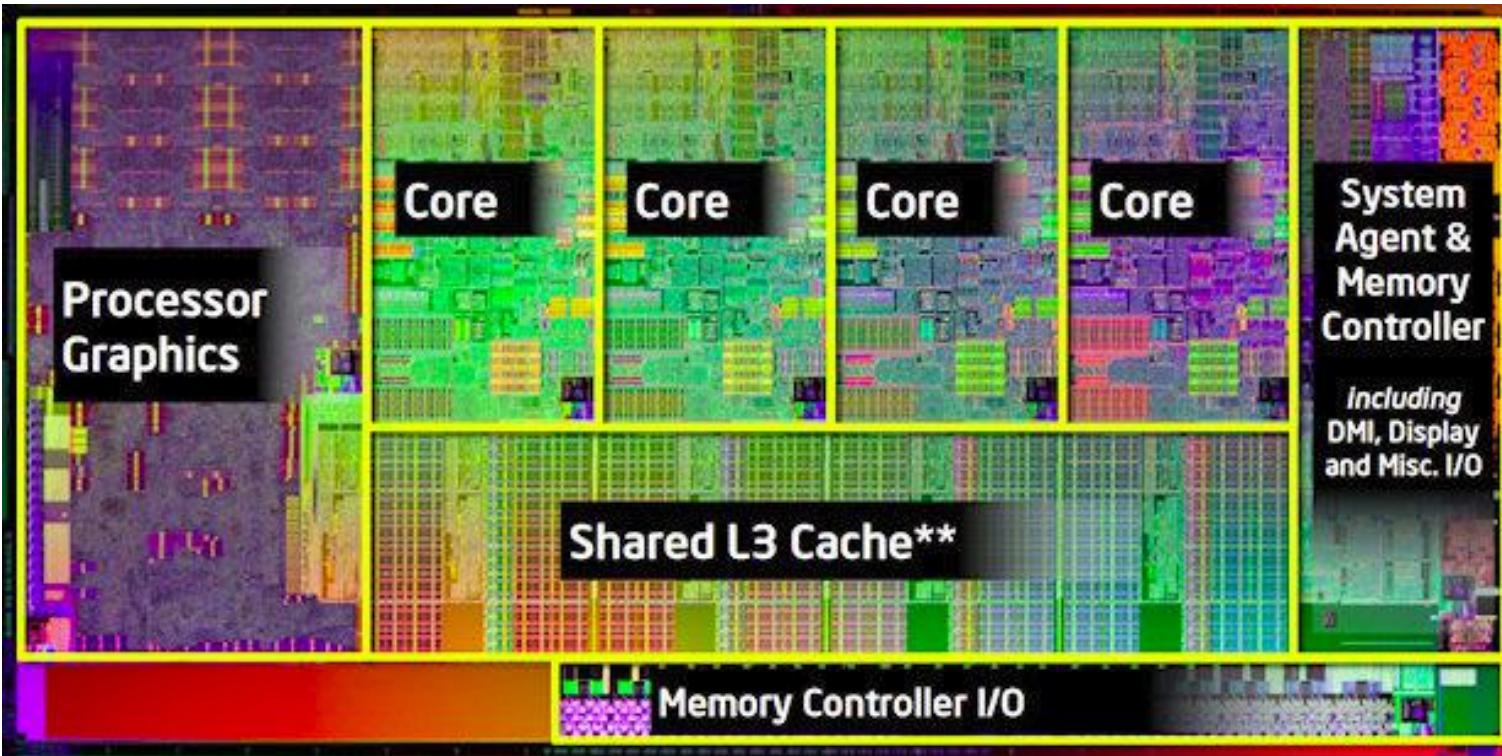
# Not only specialized processors...

---

- Multi-core processors
  - Intel Xeon Sierra Forest:  
144 cores; planned 288
  - AMD Epyc  
128 cores; planned 192



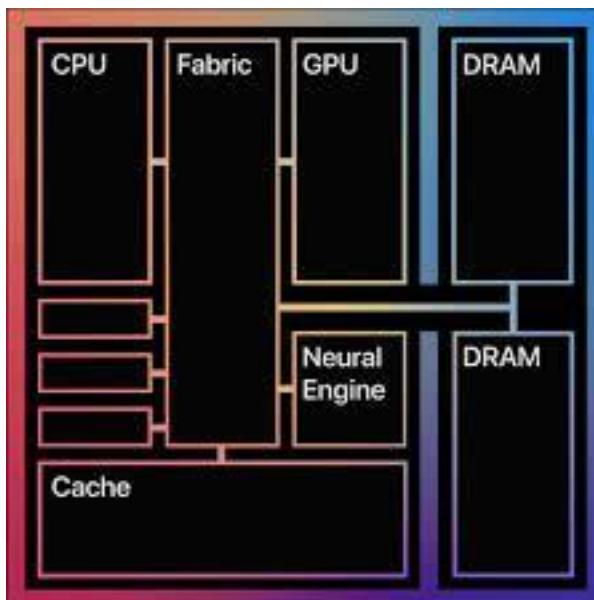
# A Modern Processor: Intel Sandy Bridge



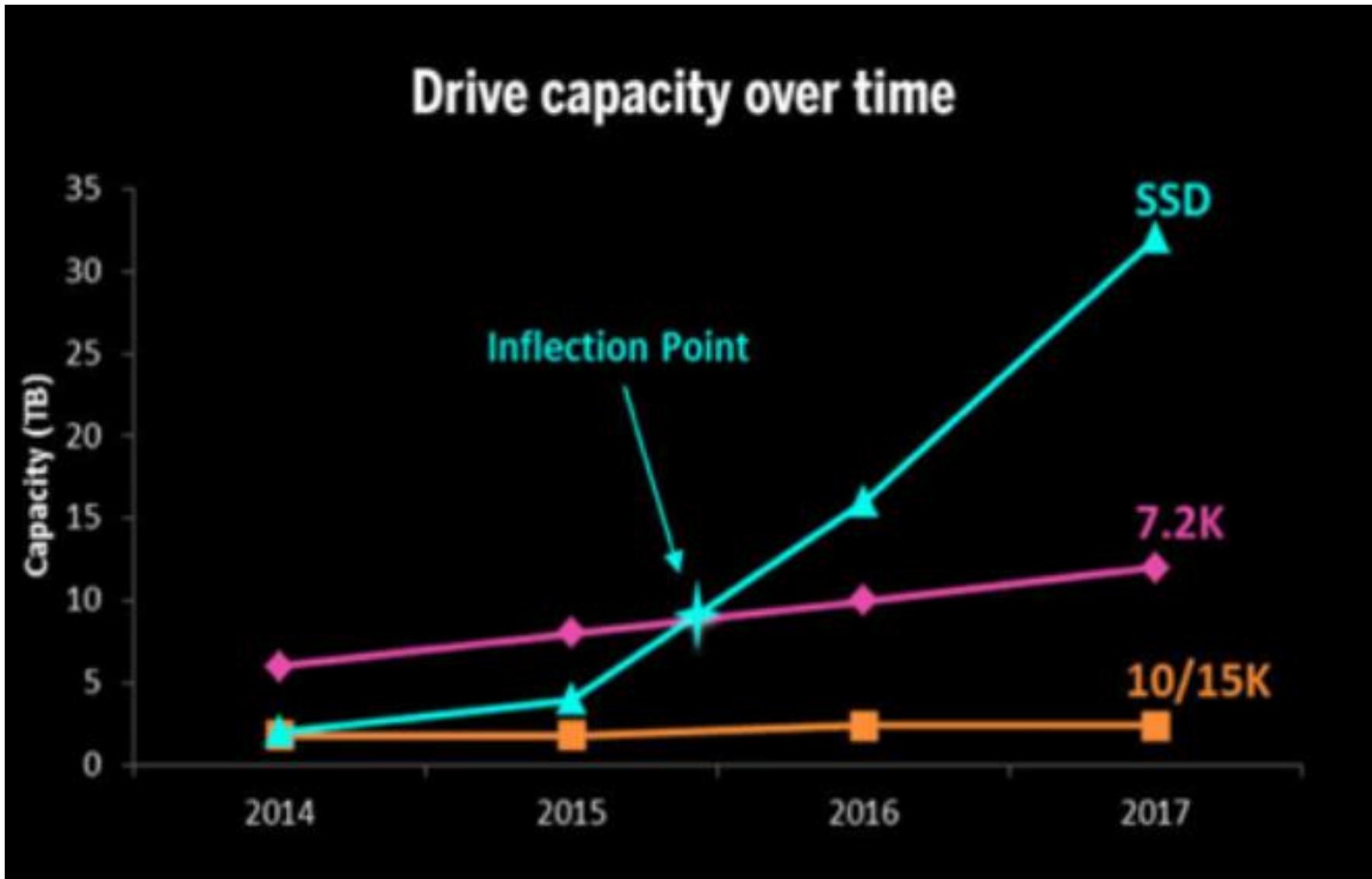
- Package: LGA 1155
  - 1155 pins
  - 95W design envelope
- Cache:
  - L1: 32K Inst, 32K Data (3 clock access)
  - L2: 256K (8 clock access)
  - Shared L3: 3MB – 20MB
- Transistor count:
  - 504 Million (2 cores, 3MB L3)
  - 2.27 Billion (8 cores, 20MB L3)
- Note that ring bus is on high metal layers – above the Shared L3 Cache

# Not only specialized processors...

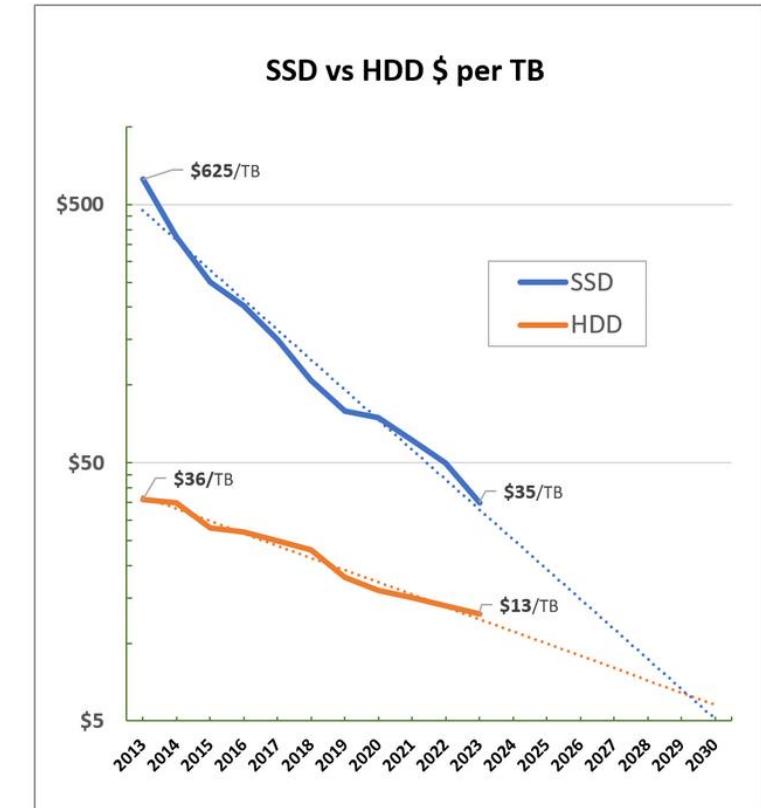
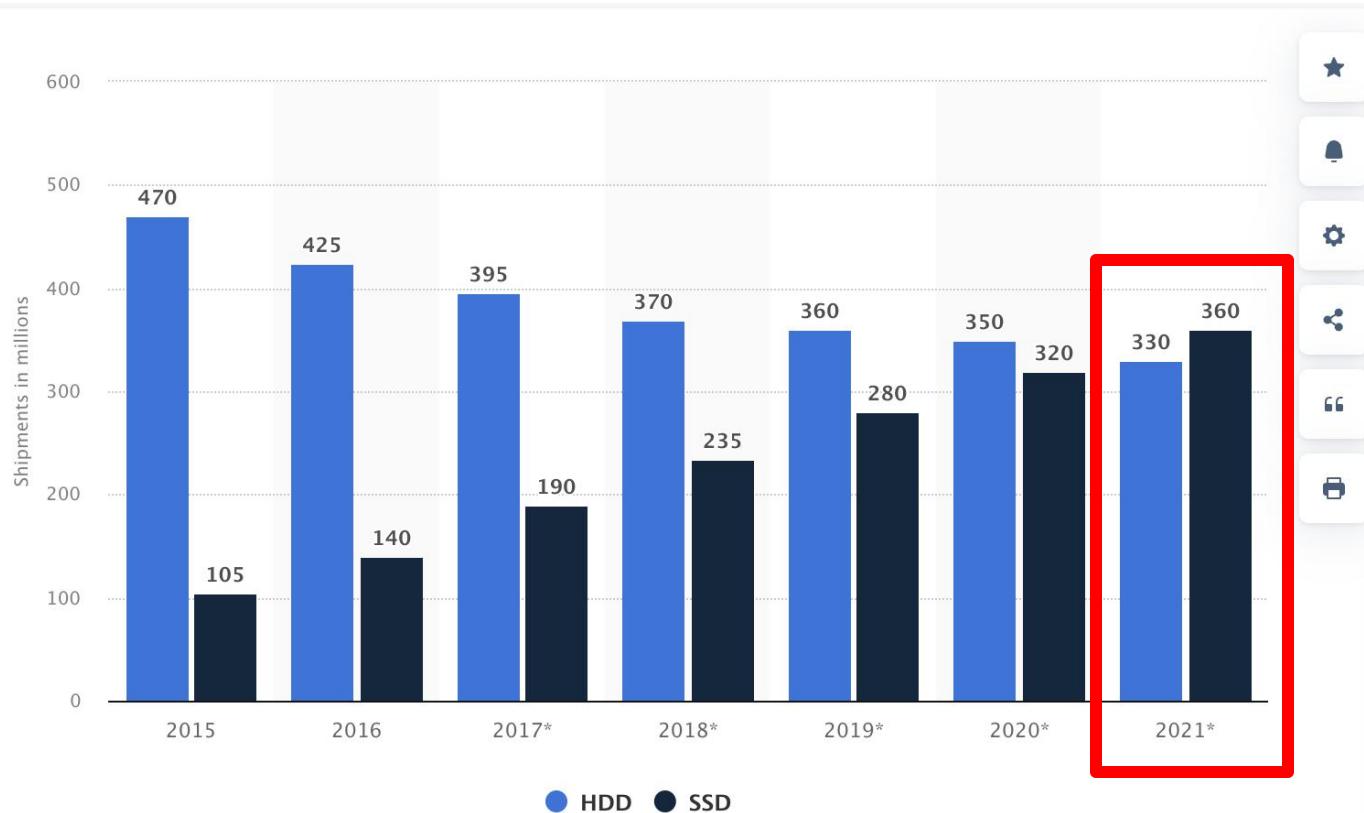
- Multi-core processors
  - Intel Xeon Sierra Forest:  
144 cores; planned 288
  - AMD Epyc  
128 cores; planned 192
- System On a Chip (SOC):  
accelerating trend



# SSD overtaking HDD



# SSD/Flash Dominating



# Storage Capacity

---



Largest SSD 3.5-inch drive:  
100 TB @ \$40K (\$400/TB)



Largest HDD 3.5-inch drive:  
24 TB @ \$480 (\$20/TB)

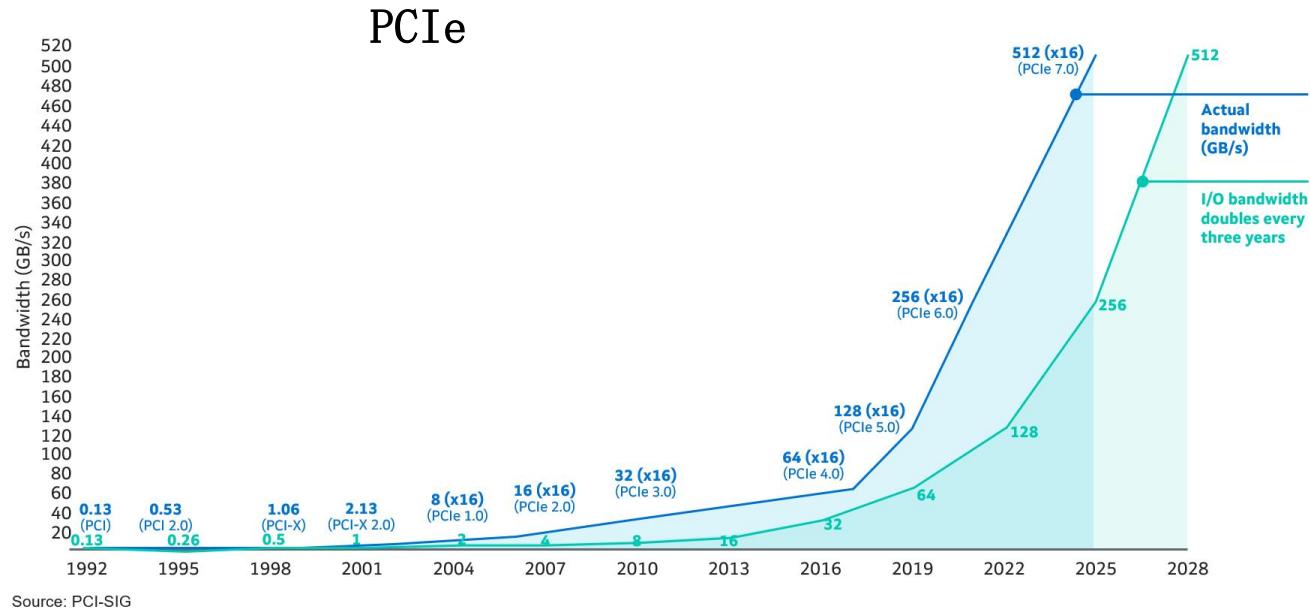
Leven SSD 2.5-inch drive:  
4 TB @ \$180 (\$45/TB)

Cheaper, but

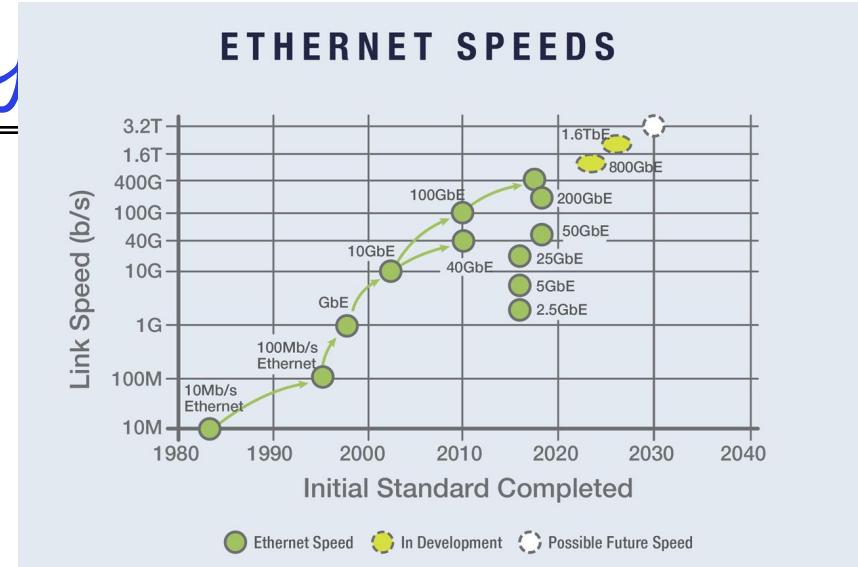
- Slower (10x-100x)
- Consumes more power
- Less reliable

# Network Capacity

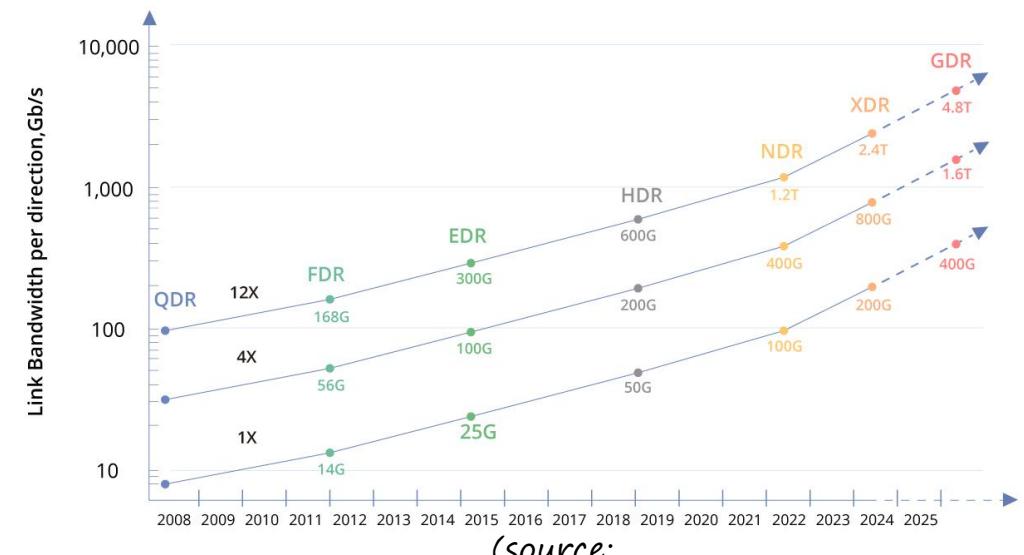
I/O bandwidth doubles every three years



(source: <https://www.marvell.com/blogs/how-pcie-interconnect-is-critical-for-the-emerging-ai-era.html>)

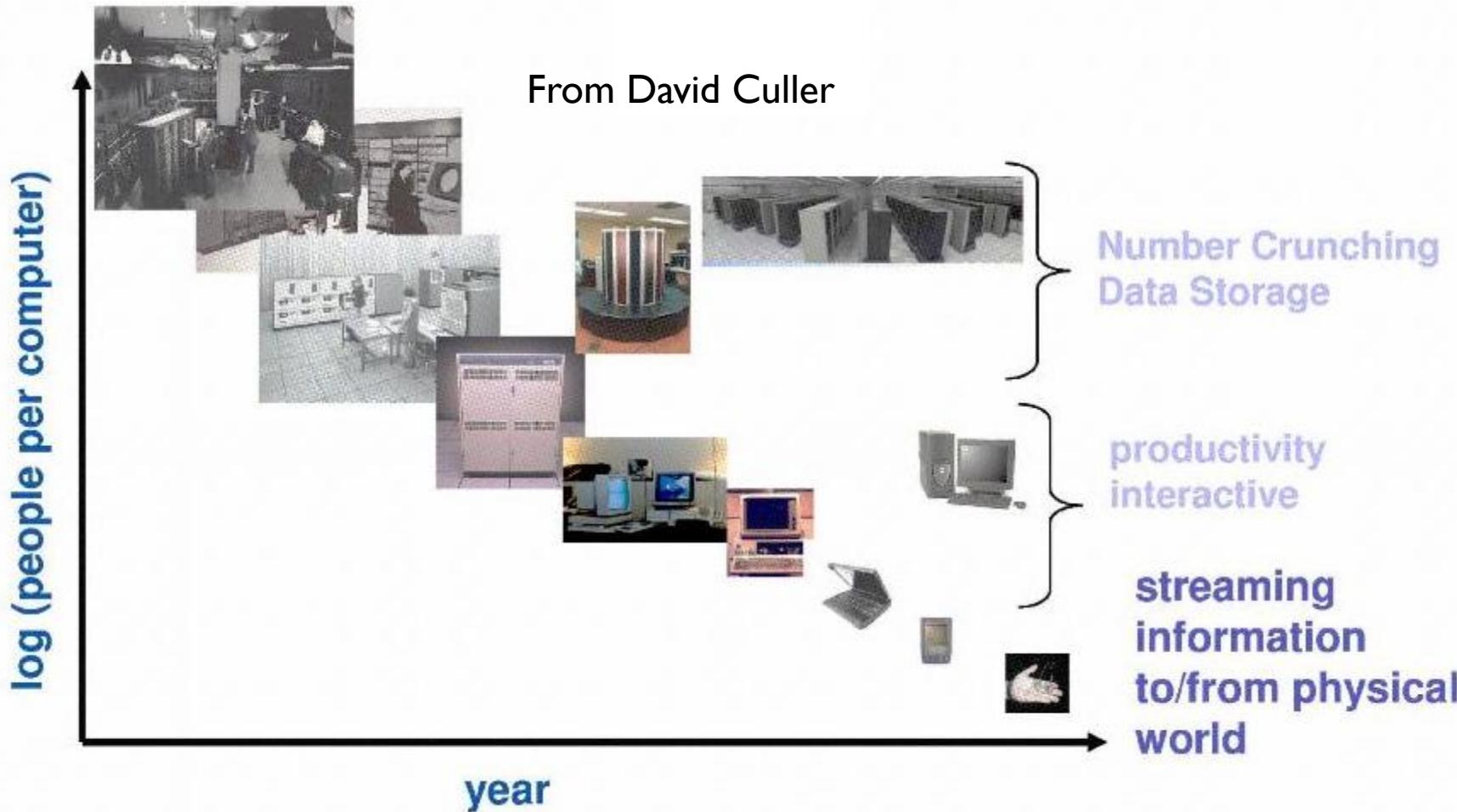


(source:  
<https://iebmedia.com/technology/industrial-ethernet/ethernet-celebrates-50-years-with-2023-roadmap-and-demo/>)  
 InfiniBand Roadmap



(source:  
<https://community.fs.com/article/need-for-speed-%E2%80%93-infiniband-network-bandwidth-evolution.html>)

# People-to-Computer Ratio Over Time



- Today: multiple CPUs/person!
  - Approaching 100s?

# And Range of Timescales

---

**Jeff Dean:**  
**“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

# Challenge: 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
  - The question is not whether there are bugs but how serious are the bugs!

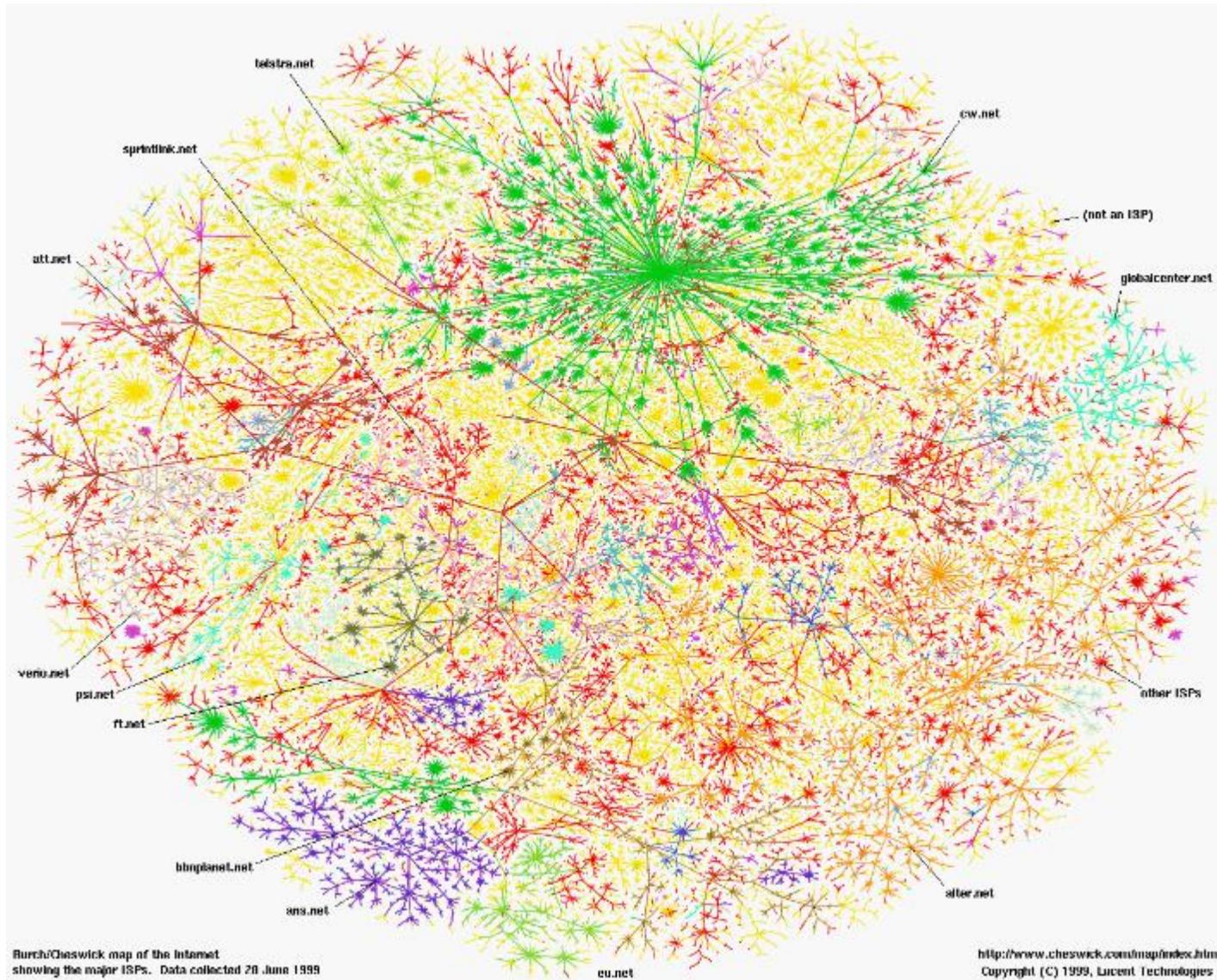
# *Everything going distributed and heterogeneous*

---

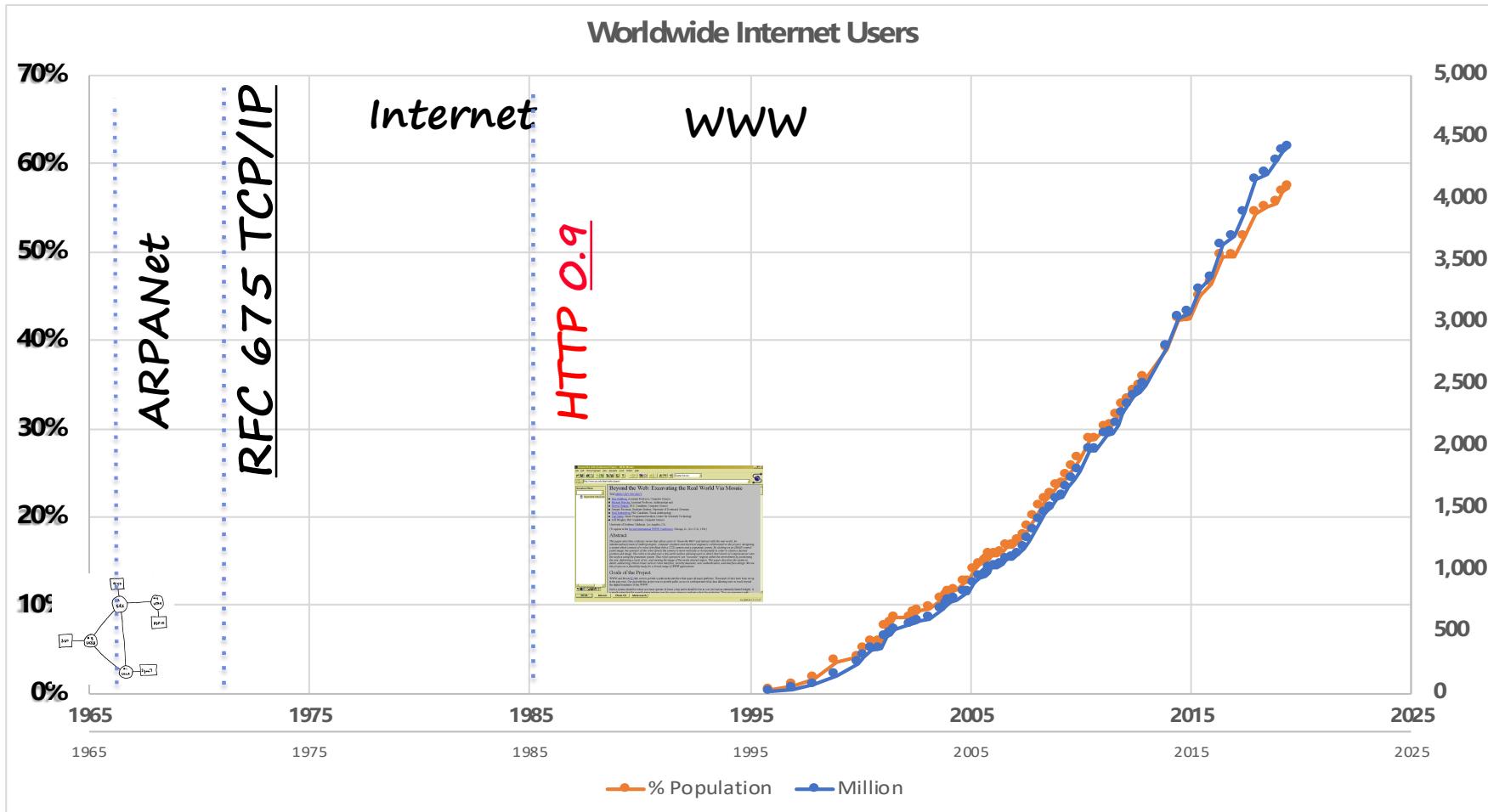
- Need to scale
  - Machine learning (AI) workloads
  - Big Data analytics
  - Scientific computing
  - ...
- *Everything is connected!*

# Greatest Artifact of Human Civilization...

---



# Running Systems at Internet Scale



196 1974  
9

199  
0

# Not Only PCs connected to the Internet

- Smartphone shipments exceed PC shipments in 2011

1.17B in 2023

- 2011 shipments:

- 487M smartphones

- 414M PC clients

>> 210M notebooks

>> 112M desktops

>> 63M tablets

- 25M smart TVs

241.9M in

2023

128.5M in

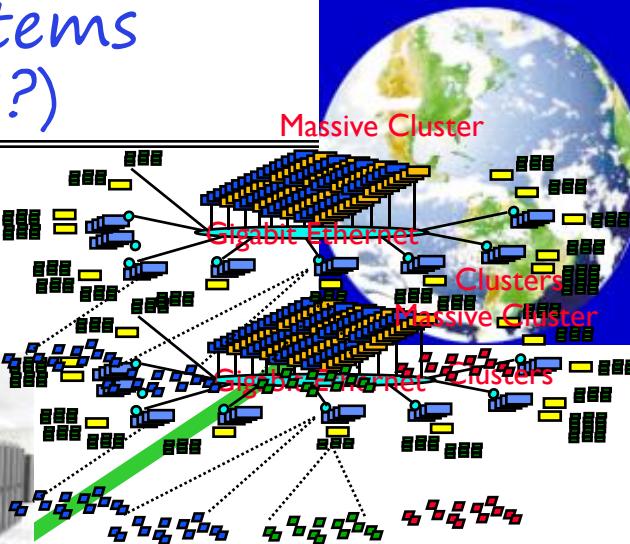
2023

200M in  
2023

- Now 7 billion smartphone in the world

# Societal Scale Information Systems (Or the “Internet of Things”?)

- The world is a large distributed system
  - Microprocessors in everything
  - Vast infrastructure behind them



Internet  
Connectivity



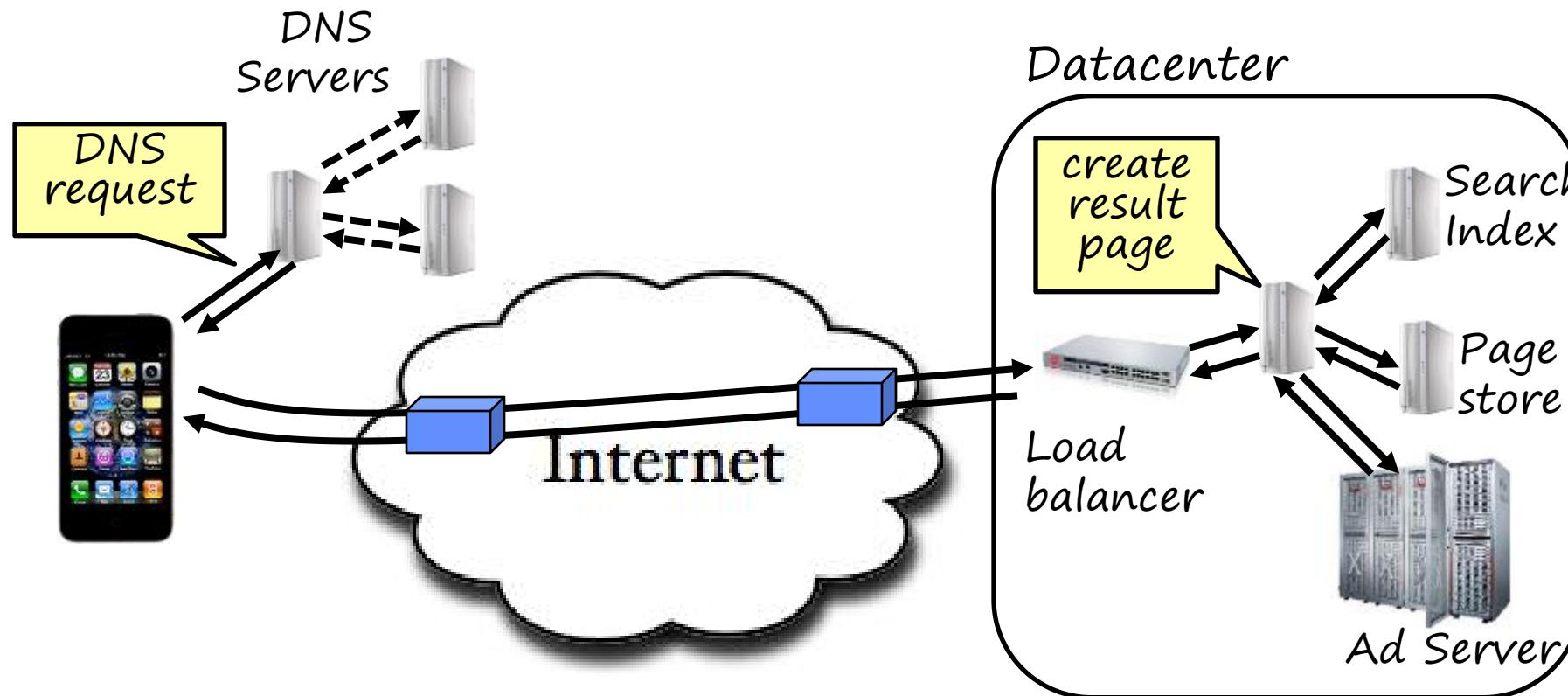
MEMS for  
Sensor Nets

Scalable, Reliable,  
Secure Services

Databases  
Information Collection  
Remote Storage  
Online Games  
Commerce

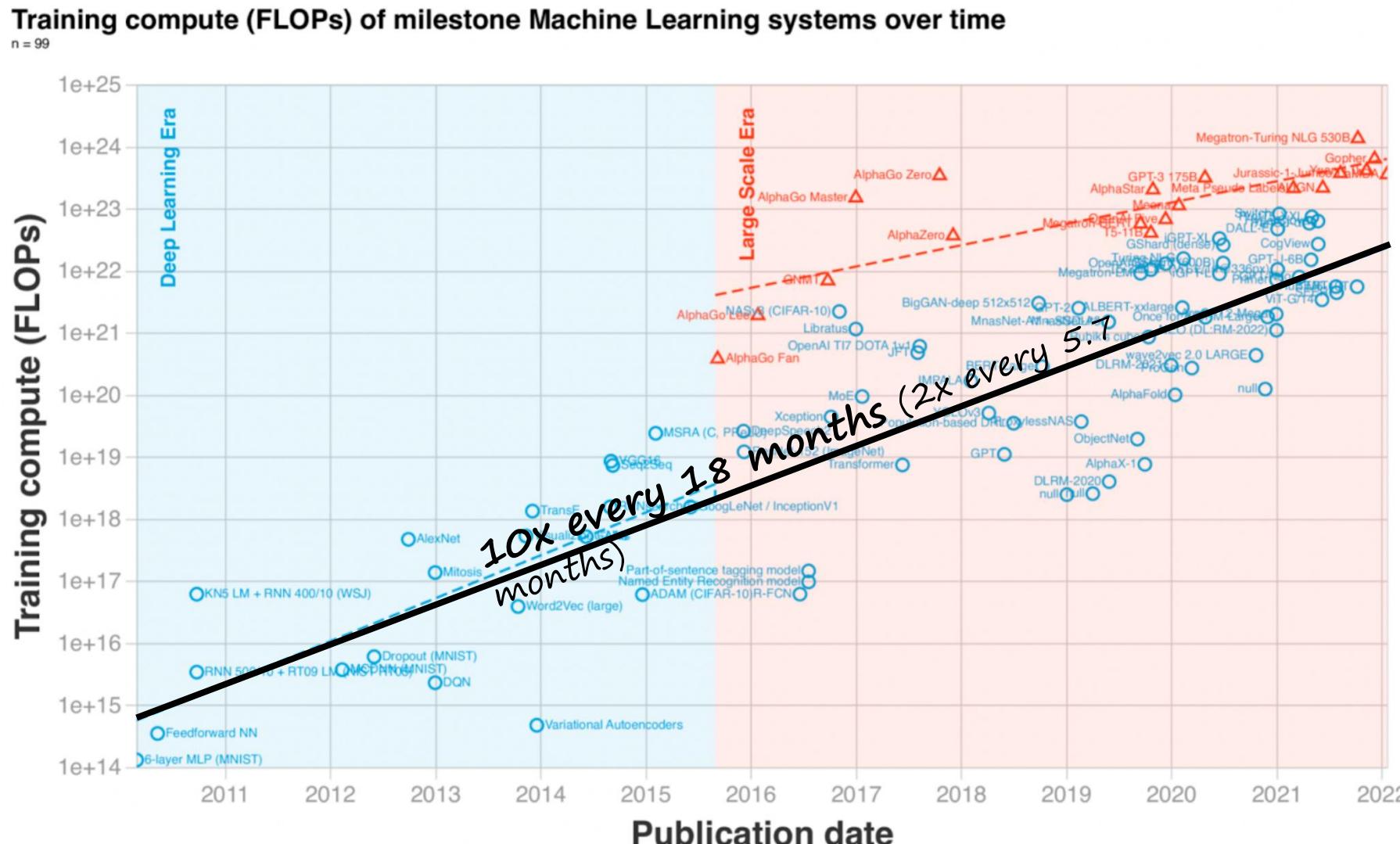
...

# Example: What's in a Search Query?

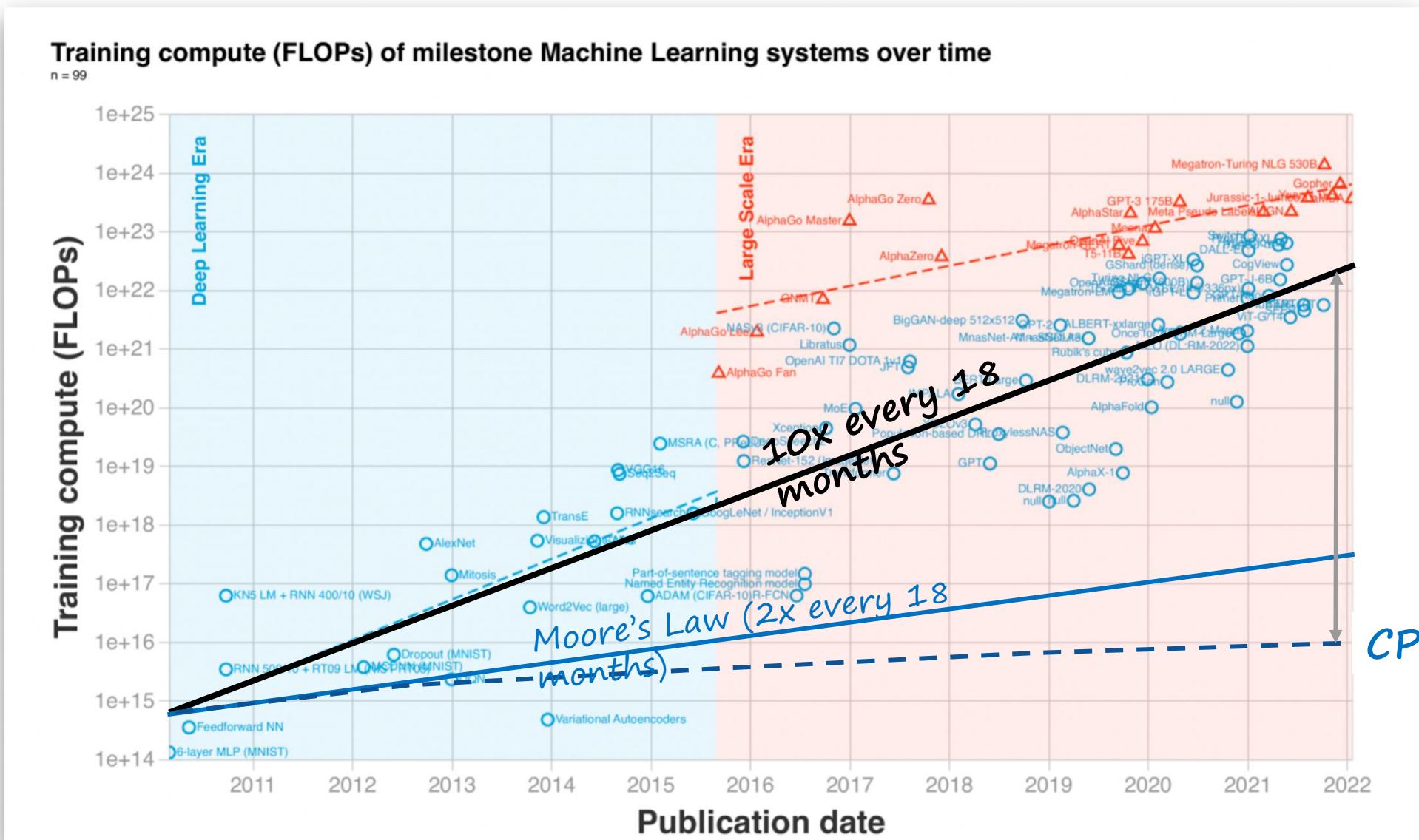


- Complex interaction of multiple components in multiple administrative domains
  - Systems, services, protocols, ...

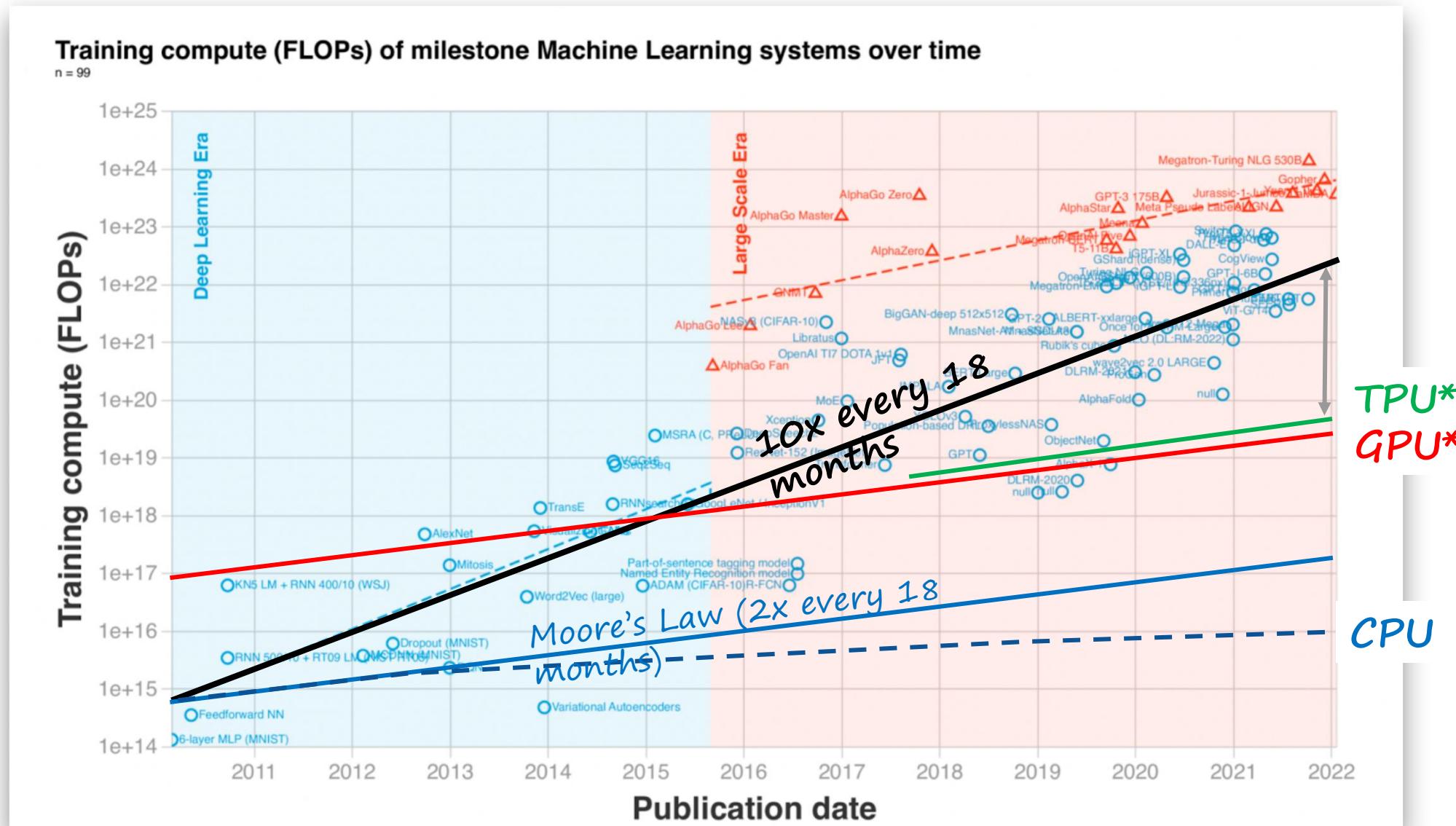
# AI demands are exploding



# Growing gap between demand and supply



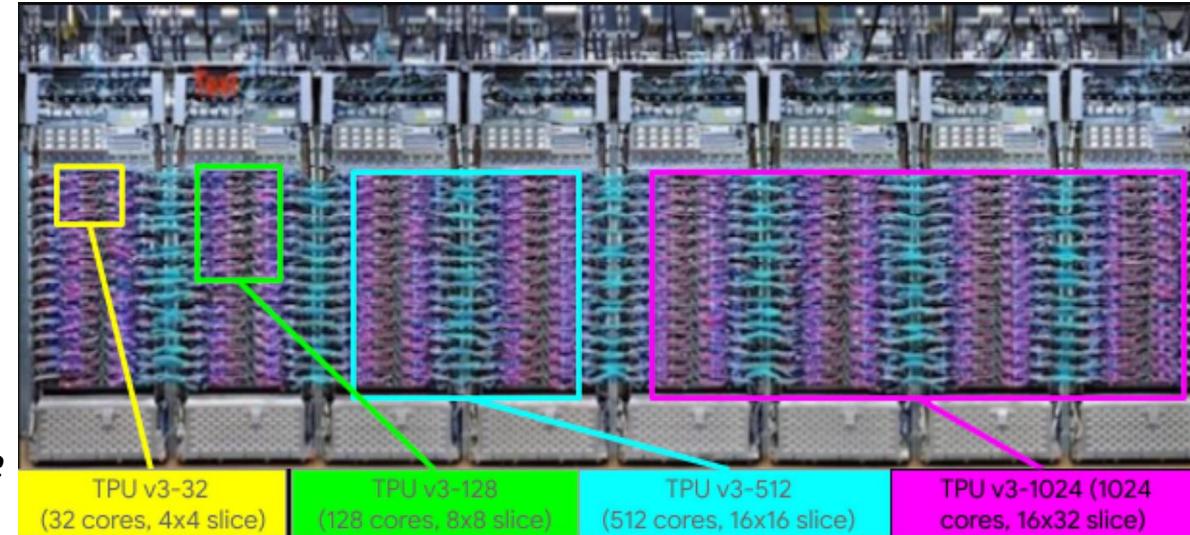
# Specialized hardware not enough



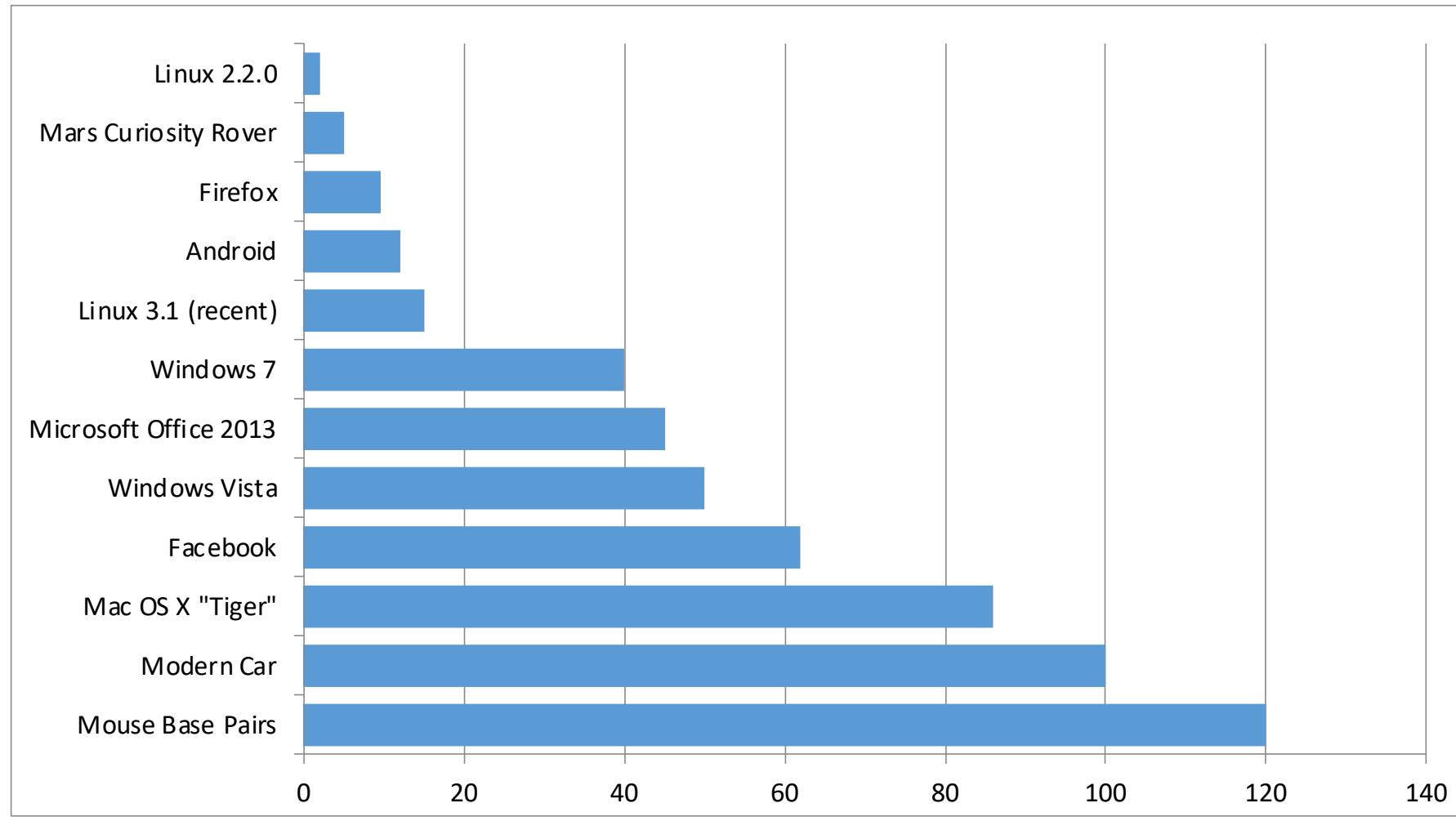
# From Servers to Pods



From servers to  
Pods (hundreds of  
thousands of  
GPUs tightly  
connected by  
high-speed  
networks, i.e.,  
supercomputers)



# Increasing Software Complexity



**Millions of Lines of Code**

(source <https://informationisbeautiful.net/visualizations/million-lines-of-code/>)

# How do we tame complexity?

---

- Every piece of computer hardware different
  - Different CPU
    - » Intel, ARM, AMD, RISC-V
  - Different specialized hardware
    - » Nvidia GPUs, TPUs, Intel Gaudi, AWS Trainium/Inferentia, ...
  - Different amounts of memory, disk, ...
  - Different types of devices
    - » Mice, Keyboards, Sensors, Cameras, Fingerprint readers, Face recognition
  - Different networking environment
    - » Fiber, Cable, DSL, Wireless, Firewalls,...
- Questions:
  - Does the programmer need to write a single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does a faulty program crash everything?
  - Does every program have access to all hardware?

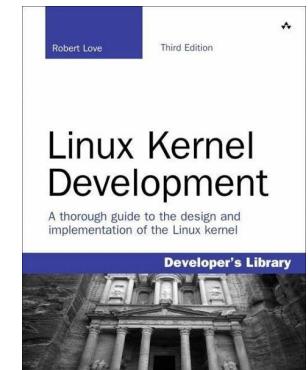
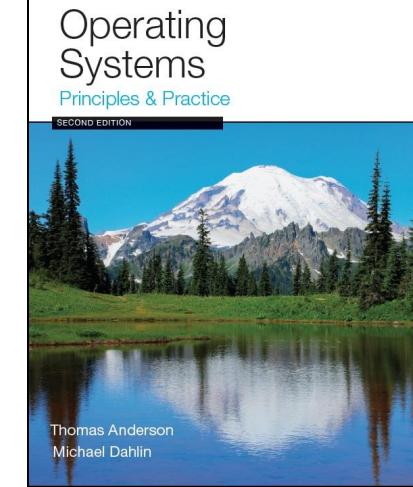
# Syllabus

---

- OS Concepts: How to Navigate as a Systems Programmer!
  - Process, I/O, Networks and Virtual Machines
- Concurrency
  - Threads, scheduling, locks, deadlock, scalability, fairness
- Address Space
  - Virtual memory, address translation, protection, sharing
- File Systems
  - I/O devices, file objects, storage, naming, caching, performance, paging, transactions, databases
- Distributed Systems
  - Protocols, N-Tiers, RPC, NFS, DHTs, Consistency, Scalability, multicast
- Reliability & Security
  - Fault tolerance, protection, security
- Cloud Infrastructure

# Infrastructure, Textbook & Readings

- Infrastructure
  - Website: <https://cs162.org>
- Textbook: Operating Systems: Principles and Practice (2nd Edition) Anderson and Dahlin
  - Suggested readings posted along with lectures
  - Try to keep up with material in book as well as lectures
- Supplementary Material
  - Operating Systems: Three Easy Pieces, by Remzi and Andrea Arpaci-Dusseau, available for free online
  - Linux Kernel Development, 3rd edition, by Robert Love
- Online supplements
  - See course website
  - Includes Appendices, sample problems, etc.
  - Networking, Databases, Software Eng, Security
  - Some Research Papers!



# Learning by Doing

---

- Individual Homeworks (2-3 weeks) - preliminary
  - 0. Tools & Environment, Autograding, recall C, executables
  - 1. Lists in C
  - 2. BYOS - build your own shell
  - 3. Sockets & Threads in HTTP server
  - 4. Memory mapping and management
  - 5. Map Reduce
- Three (and ½) Group Projects
  - 0. Getting Started (Individual, before you have a group)
  - 1. User-programs (exec & syscall)
  - 2. Threads & Scheduling
  - 3. File Systems



# Group Projects

---

- Project teams have 4 members!
    - never 5, 3 requires serious justification
    - Must work in groups in “the real world”
    - Same section (at least same TA)
  - Communication and cooperation will be essential
    - Regular in-person meetings very important!
    - Joint work on Design Documents
    - Slack/Messenger/whatever doesn’t replace face-to-face
  - Everyone should do work and have clear responsibilities
    - You will evaluate your teammates at the end of each week
    - Dividing up by Task is the worst approach. Work as a team.
  - Communicate with supervisor (TAs)
    - What is the team’s plan?
    - What is each member’s responsibility?
    - Short progress reports are required
- Design Documents: High-level description for a manager:**



# Getting started

---

- **EVERYONE** (even if you are on the waitlist!):  
Start homework 0 right away (hopefully Today!), project 0 next week
  - Github account
  - VM environment for the course
    - » Consistent, managed environment on your machine
  - Get familiar with all the cs162 tools
  - Submit to autograder via git
- First two weeks, attend any section you want
  - We'll assign permanent sections after forming project groups
  - Section attendance will be mandatory after we form groups
  - These section times will be adjusted after we have a better idea where people are

# Preparing Yourself for this Class

---

- The projects will require you to be very comfortable with programming and debugging C
  - Pointers (including function pointers, void\*)
  - Memory Management (malloc, free, stack vs heap)
  - Debugging with GDB
- You will be working on a larger, more sophisticated code base than anything you've likely seen in 61C!
- Review Session on the C language
  - Time and logistics TBA, but soon!
- "Resources" page on course website
  - Ebooks on "git" and "C"
- C programming reference
  - <https://cs162.org/ladder/>
- First two sections are also dedicated to programming and debugging review:
  - Attend ANY sections in first two weeks

# Grading (Tentative breakdown)

---

- 36% three midterms (12% each)
  - Thursday, 10/3 (7-9pm or 8-10pm), No class on day of Midterm
  - Tuesday, 11/05 (7-9pm or 8-10pm), No class on day of Midterm
  - Thursday, 12/05 (7-9pm or 8-10pm), No class on day of Midterm
  - These will be IN-PERSON!
- 36% projects
- 18% homework
- 10% participation (Sections, Lecture, ...)
- No final exam
- Projects
  - Initial design document, Design review, Code, Final design
  - Submission via git push triggers autograder

# CS 162 Collaboration Policy

---



Explaining a concept to someone in another group

Discussing algorithms/testing strategies with other groups

Discussing debugging approaches with other groups

Searching online for generic algorithms (e.g., hash table)



Sharing code or test cases with another group

Copying OR reading another group's code or test cases

Copying OR reading online code or test cases from prior years

Helping someone in another group to debug their code

- We compare all project submissions against prior year submissions and online solutions and will take actions (described on the course overview page) against offenders
- Don't put a friend in a bad position by asking for help that they shouldn't give!

# Lecture Goal

---

Interactive!!!

Ask Questions in Chat