Welcome to CS162

David E. Culler

CS162 – Operating Systems and Systems Programming

http://cs162.eecs.berkeley.edu/

Lecture 1

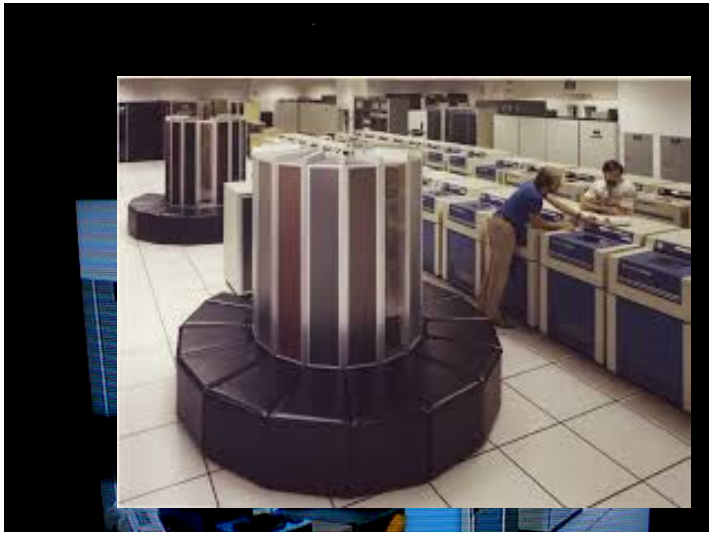
August 29, 2019

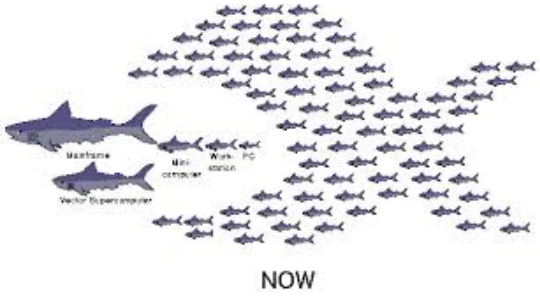
Overflow Lecture 8/30 @ 2:00 in 306 Soda

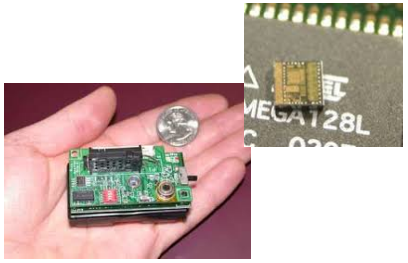
Read A&D Ch1
HW0 out, due 9/6

Introducing the CS162 Team - me

- David Culler (culler@berkeley.edu)
  - 783 Soda Hall
  - http://www.cs.berkeley.edu/~culler
  - Office hours: M 2-3, Tu 3:30-5, W 1-2
  - Before/after class
- Extreme Systems
  - Cray Time Sharing System
  - OS386, OS286
  - Active Messages
  - Massive Clusters for Internet Services
  - TinyOS / Berkeley Motes (IoT)
  - BOSS – OS for the Built Environment







Your TAs / Mentors

Jack Kolb
jkolb@cs.berkeley.edu

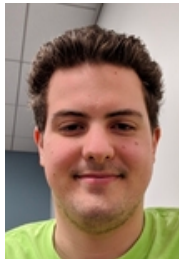
Sam Kumar
samkumar@cs.berkeley.edu

Varsha Ramakrishnan
vio@berkeley.edu

Nicholas Riasanovsky
njriasanovsky@berkeley.edu

Annie Ro
anniero@berkeley.edu

Alan Ton
alanton@berkeley.edu

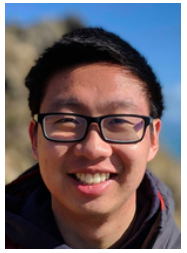


William Walker
wewalker@berkeley.edu

Sharie Wang
sharie@berkeley.edu

William Wang
hwang97@berkeley.edu

Alexander Wu
alexanderwu@berkeley.edu

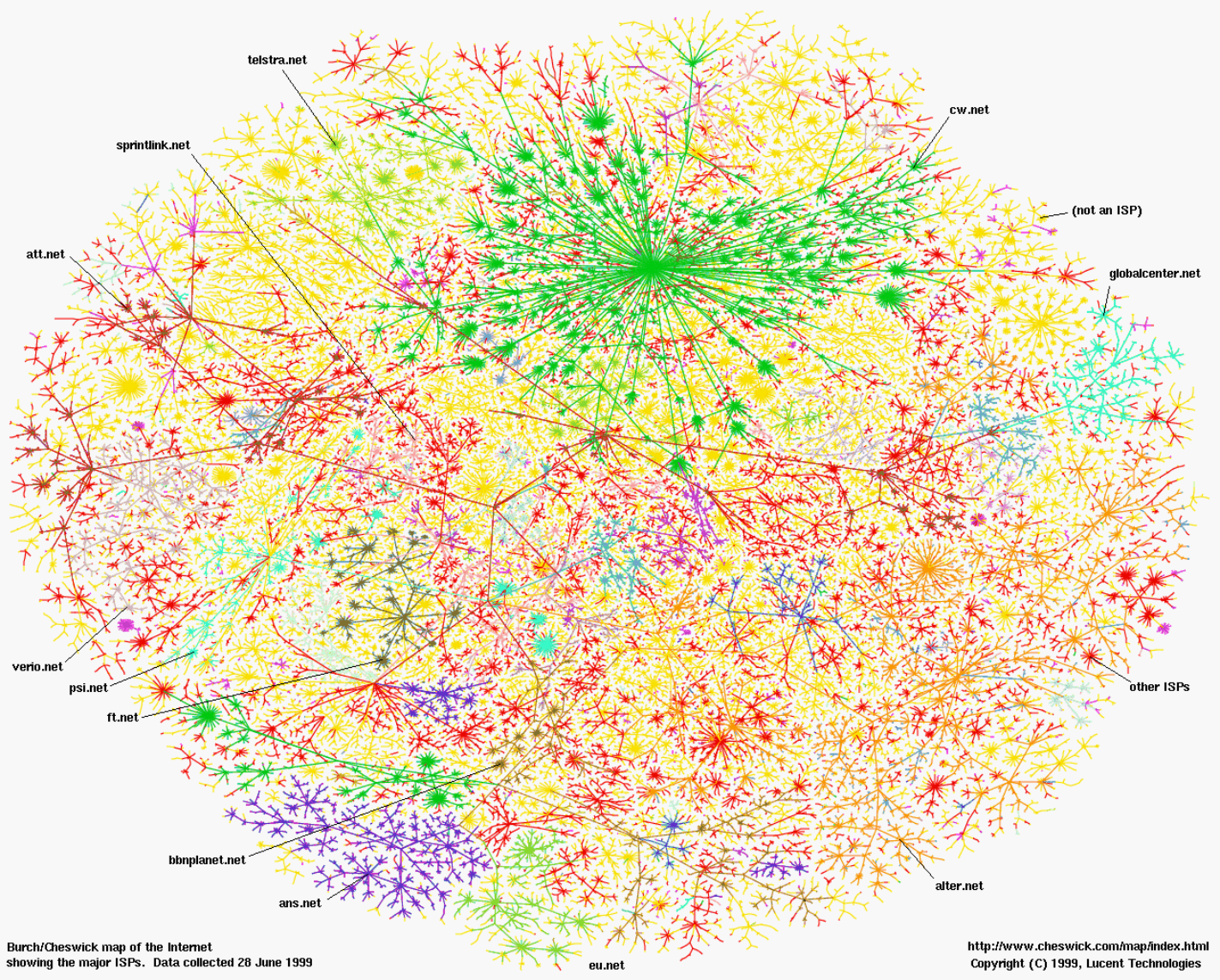


Yi Zhao
yi@berkeley.edu

Today's Objectives

- Introduce you to Operating Systems & Their Design
- Introduce the CS162 instructional team & Plan
- Establish expectations and logistics
- Maybe get a little excited about how OS is so essential in creating and advancing this "connected world" …

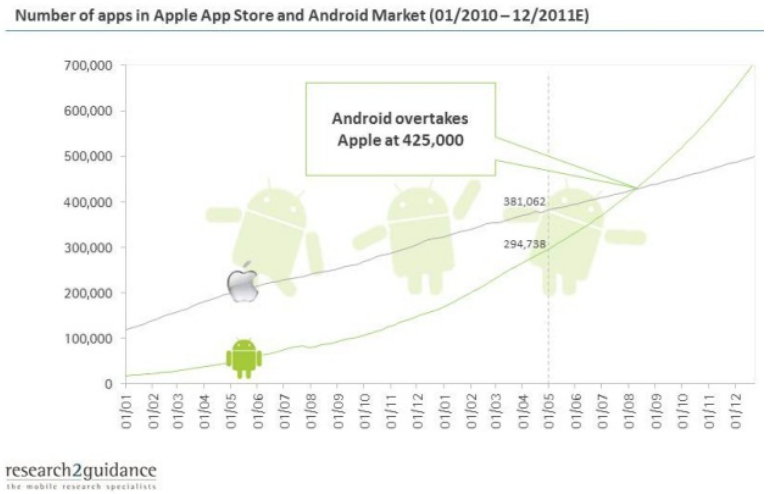
Most Transformative Artifact of Human Civilization …



Worldwide Internet Users
ARPANet
RFC 675 TCP/IP
Internet
HTTP 0.9
WWW
2.0 B 1/26/11
70%
60%
50%
40%
30%
20%
10%
0%
5,000
4,500
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
1965
1975
1985
1995
2005
2015
2025
% Population
Million
1969
1974
1990

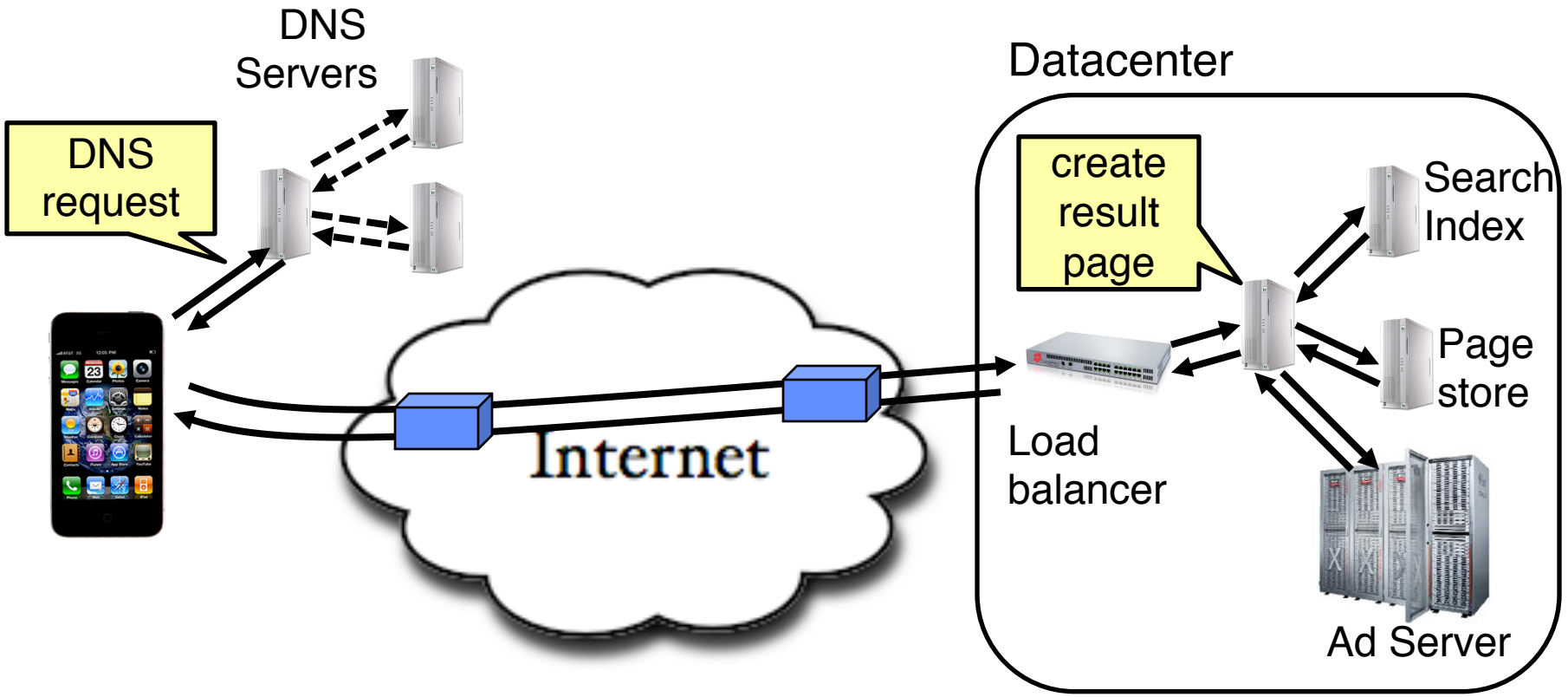
Operating Systems at the heart of it all …

- **Make the incredible advance in the underlying technology available to a rapid evolving body of applications.**
  - **Processing, Communications, Storage, Interaction, Protected Sharing**



- **The key building blocks**
  - **Processes, Scheduling**
  - **Concurrency, Coordination**
  - **Address spaces, Translation**
  - **Protection, Isolation, Sharing, Security**
  - **Communication, Protocols**
  - **Persistent storage, transactions, consistency, resilience**
  - **Interfaces to all devices**

Example: What's in a Search Query?



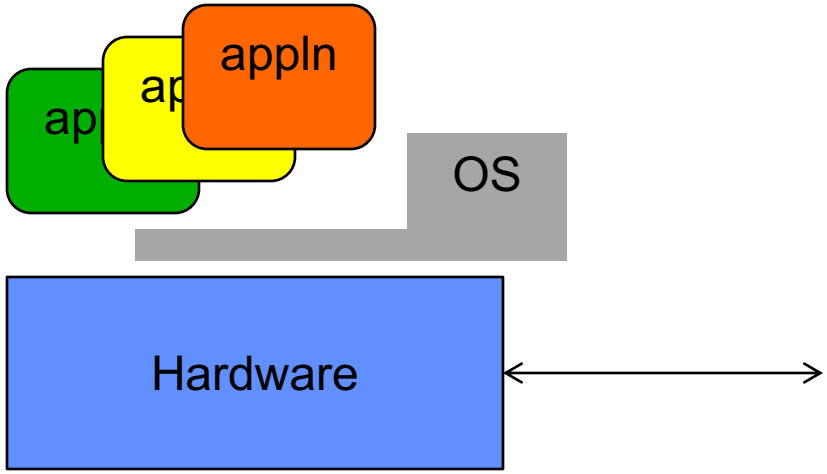
- **Complex interaction of multiple components in multiple administrative domains**
  - **Systems, services, protocols, …**

Why take CS162?

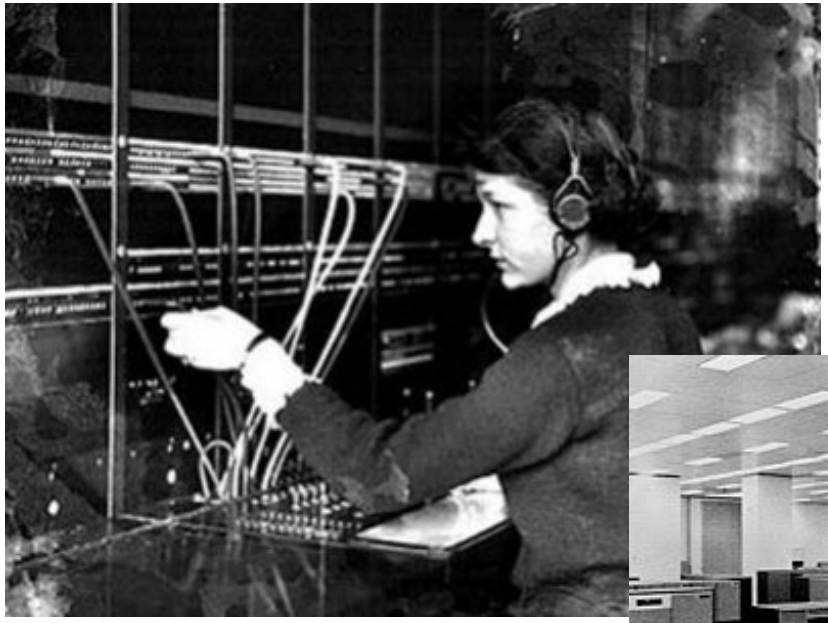
- Some of you will may design and build operating systems or components of them.
  - Perhaps more now than ever
- Many of you will create systems that utilize the core concepts in operating systems.
  - Whether you build software or hardware
  - The concepts and design patterns appear at many levels
- 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 is an operating system?

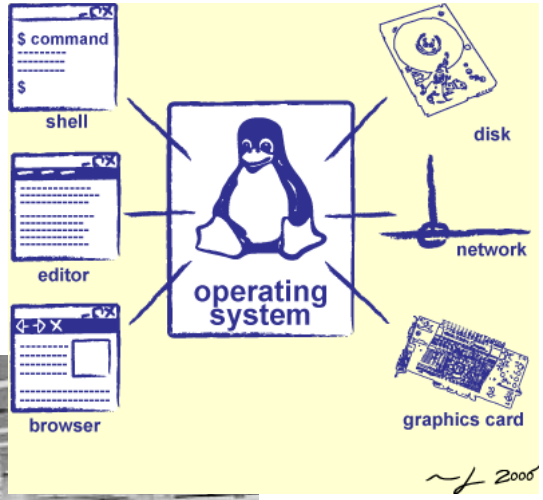
- **Special layer of software that provides application software access to hardware resources**
  - **Convenient abstraction of complex hardware devices**
  - **Protected access to Shared resources**
  - **Security and Authentication**
  - **Communication amongst logical entities**



Operator …



Switchboard Operator

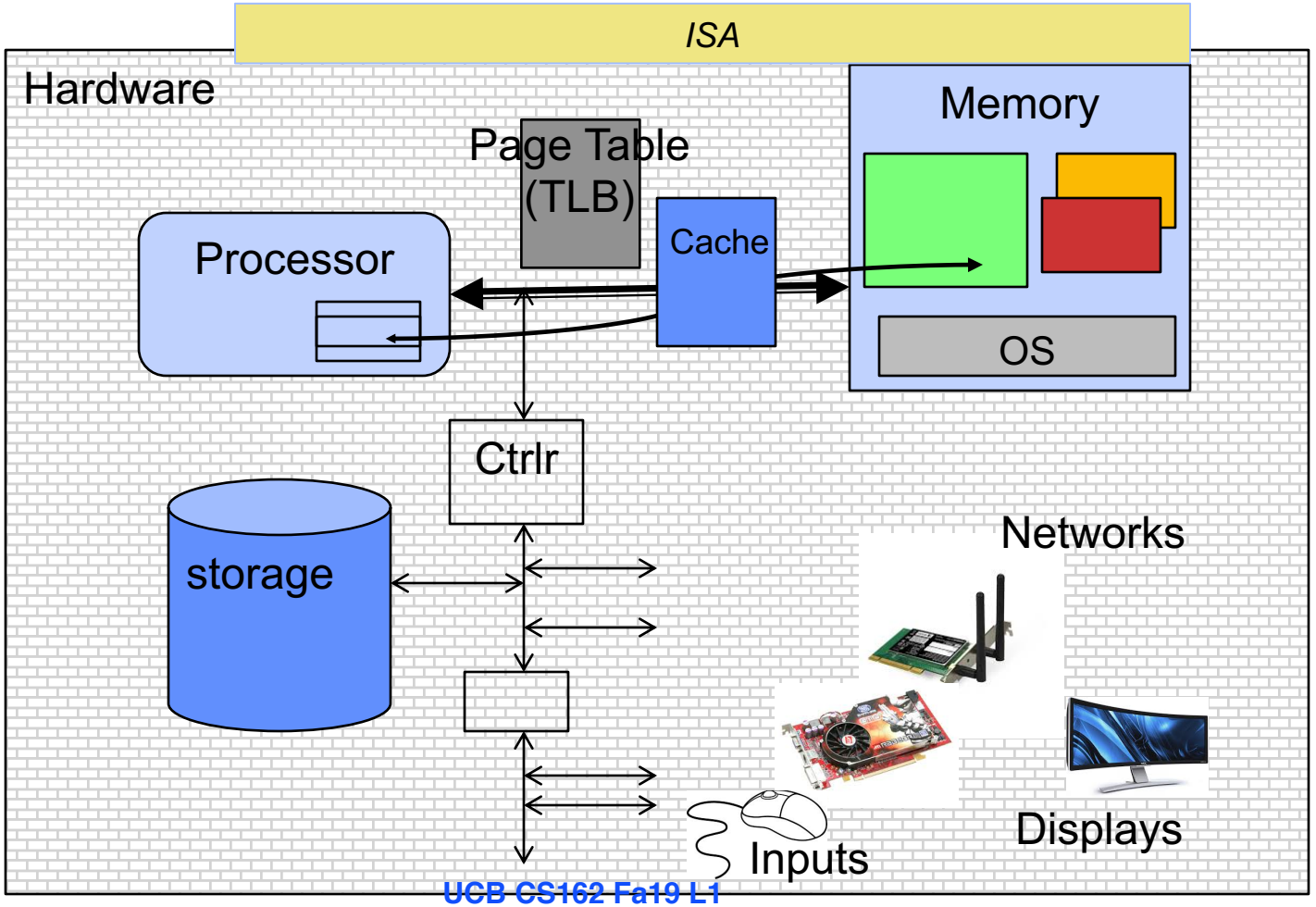


Computer Operators

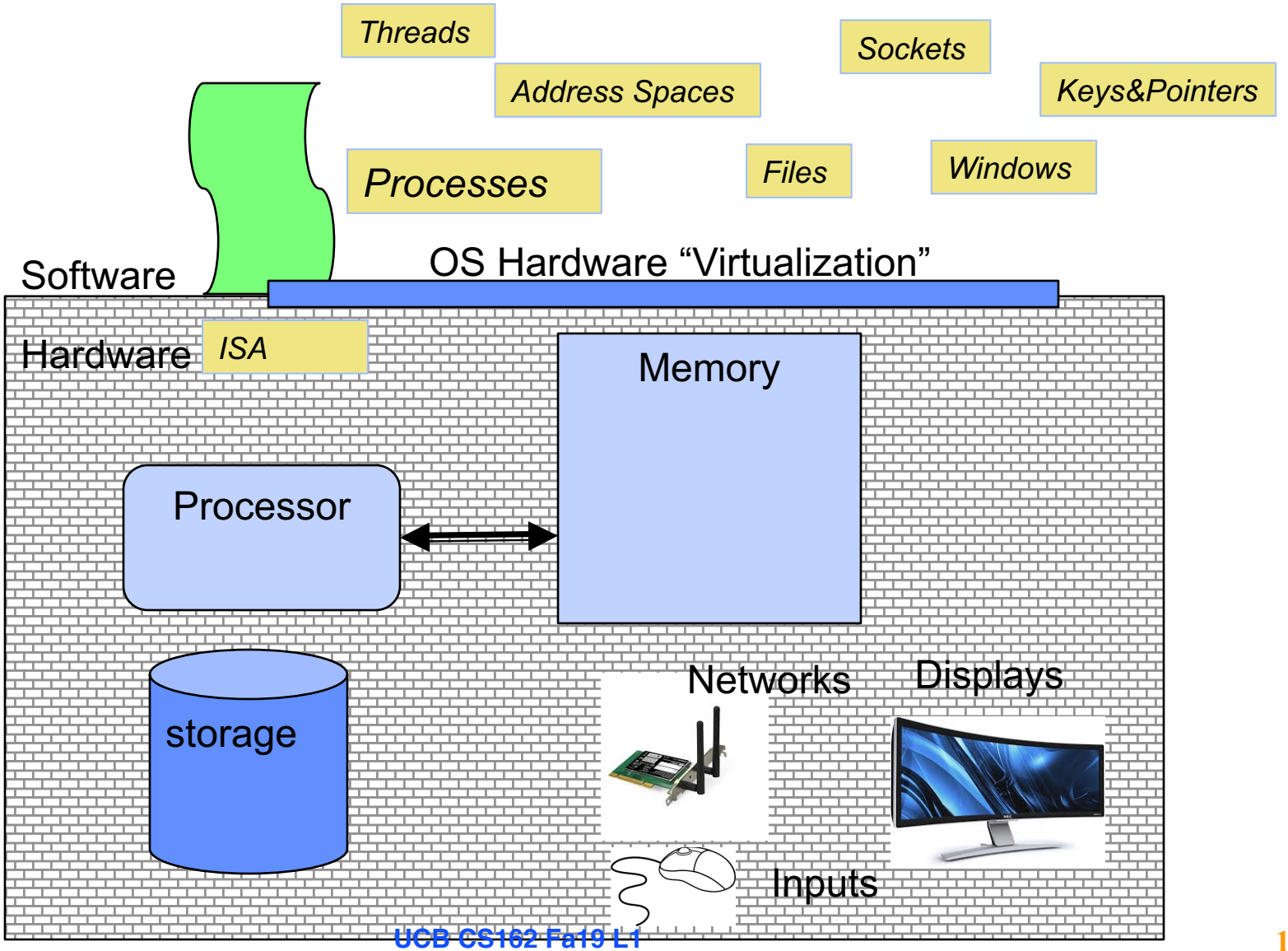
What make something a system?

- **Multiple interrelated parts**
  - **Each potentially interacts with the others, …**
- **Frame of mind:**
  - **meticulous error handling, anticipating all possible failure cases, defending against malicious/careless users, …**
- **An important part of this class**

CS61C – Machine Structures (and C)



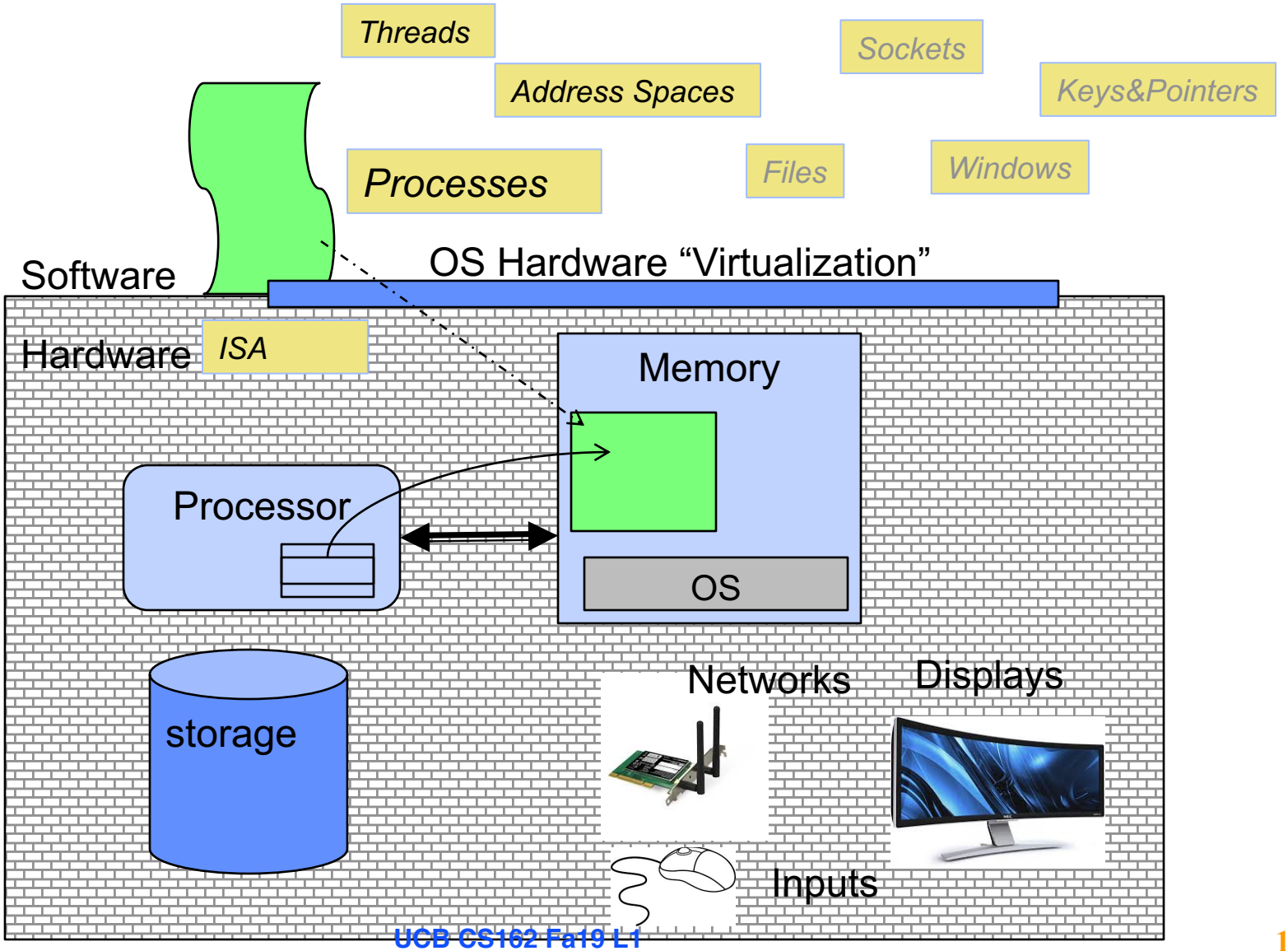
OS Basics: "Virtualizing the Machine"



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: Program => Process

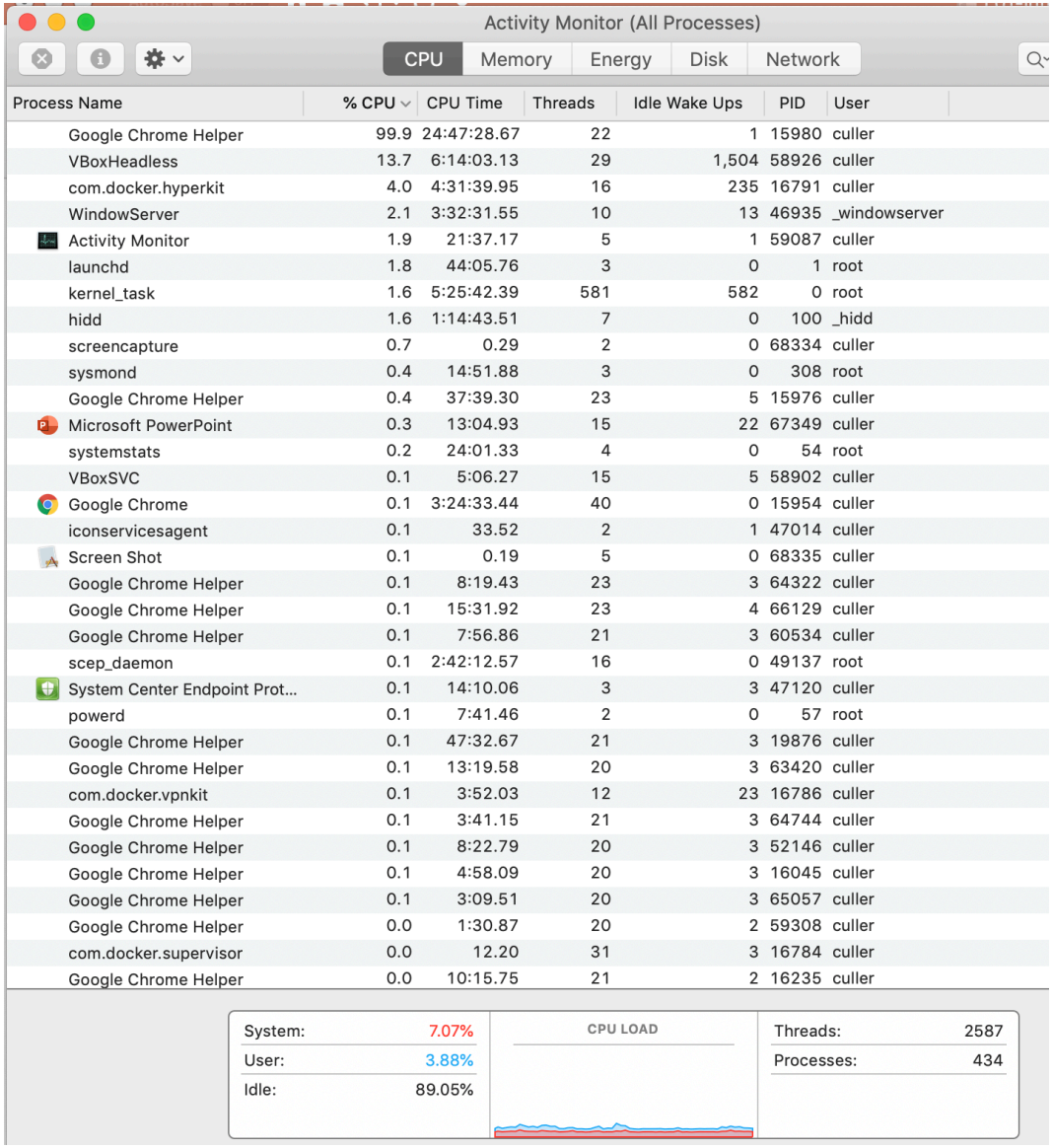


Defn: Process

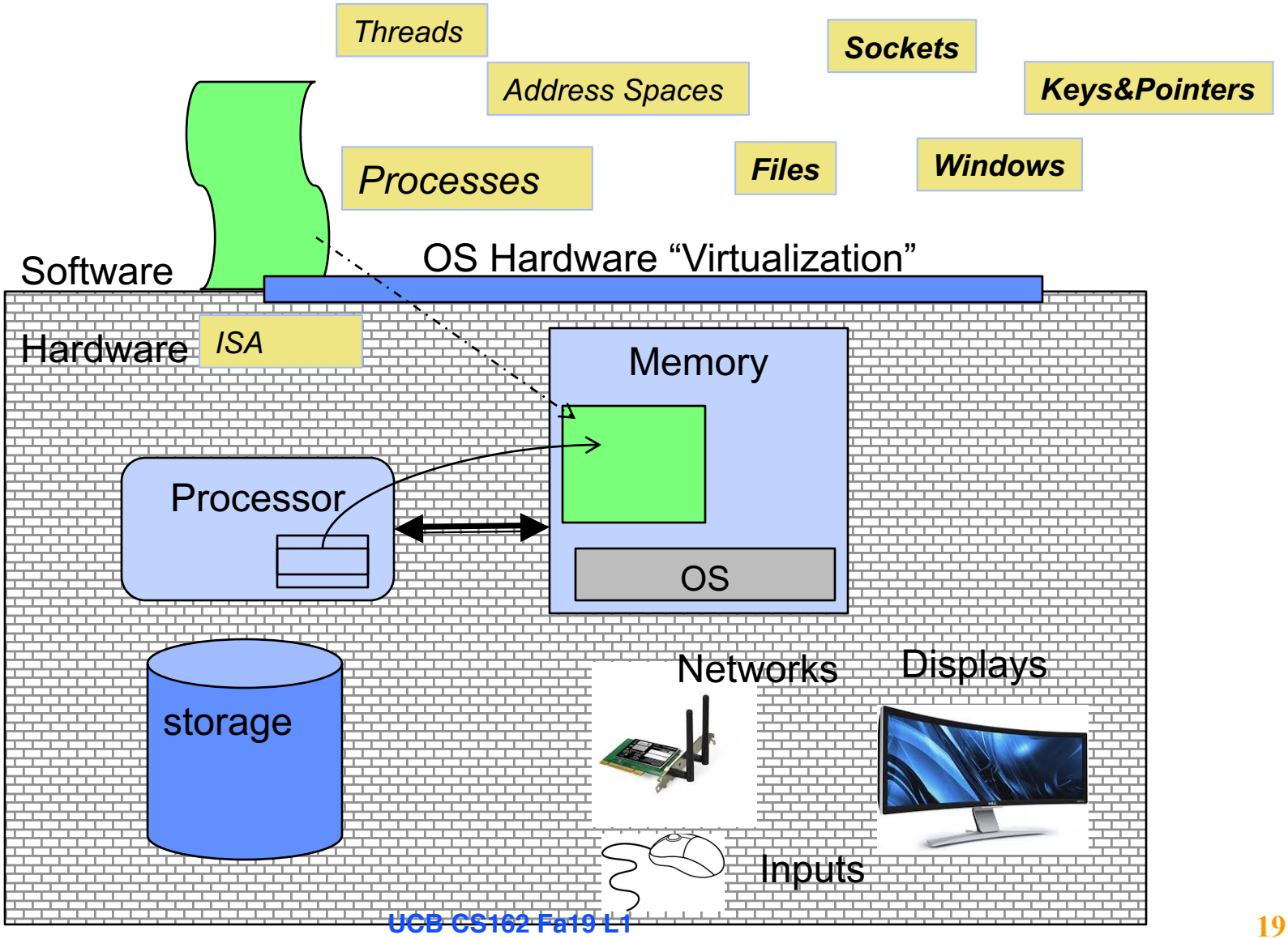
- Address Space
- One or more *threads* of control
- Additional system state associated with it

- Thread:
  - locus of control (PC)
  - Its registers (processor state when running)
  - And its "stack" (SP)
    - » As required by programming language runtime

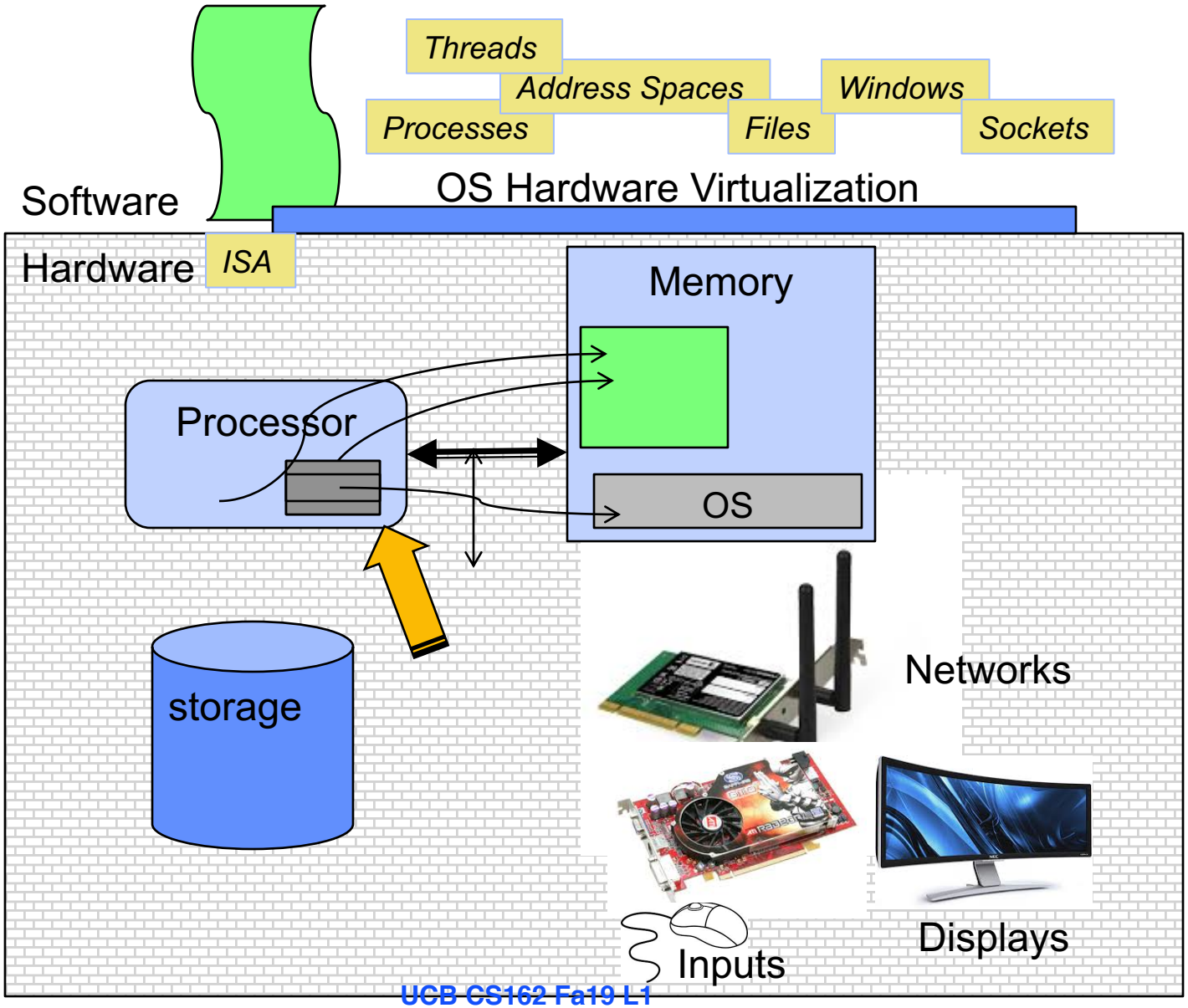
For Example …


Activity Monitor (All Processes)
CPU
Memory
Energy
Disk
Network
Process Name % CPU CPU Time Threads Idle Wake Ups PID User
Google Chrome Helper 99.9 24:47:28.67 22 1 15980 culler
VBoxHeadless 13.7 6:14:03.13 29 1,504 58926 culler
com.docker.hyperkit 4.0 4:31:39.95 16 235 16791 culler
WindowServer 2.1 3:32:31.55 10 13 46935 _windowserver
Activity Monitor 1.9 21:37.17 5 1 59087 culler
launchd 1.8 44:05.76 3 0 1 root
kernel_task 1.6 5:25:42.39 581 582 0 root
hidd 1.6 1:14:43.51 7 0 100 _hidd
screencapture 0.7 0.29 2 0 68334 culler
sysmond 0.4 14:51.88 3 0 308 root
Google Chrome Helper 0.4 37:39.30 23 5 15976 culler
Microsoft PowerPoint 0.3 13:04.93 15 22 67349 culler
systemstats 0.2 24:01.33 4 0 54 root
VBoxSVC 0.1 5:06.27 15 5 58902 culler
Google Chrome 0.1 3:24:33.44 40 0 15954 culler
iconservicesagent 0.1 33.52 2 1 47014 culler
Screen Shot 0.1 0.19 5 0 68335 culler
Google Chrome Helper 0.1 8:19.43 23 3 64322 culler
Google Chrome Helper 0.1 15:31.92 23 4 66129 culler
Google Chrome Helper 0.1 7:56.86 21 3 60534 culler
scep_daemon 0.1 2:42:12.57 16 0 49137 root
System Center Endpoint Prot… 0.1 14:10.06 3 3 47120 culler
powerd 0.1 7:41.46 2 0 57 root
Google Chrome Helper 0.1 47:32.67 21 3 19876 culler
Google Chrome Helper 0.1 13:19.58 20 3 63420 culler
com.docker.vpnkit 0.1 3:52.03 12 23 16786 culler
Google Chrome Helper 0.1 3:41.15 21 3 64744 culler
Google Chrome Helper 0.1 8:22.79 20 3 52146 culler
Google Chrome Helper 0.1 4:58.09 20 3 16045 culler
Google Chrome Helper 0.1 3:09.51 20 3 65057 culler
Google Chrome Helper 0.0 1:30.87 20 2 59308 culler
com.docker.supervisor 0.0 12.20 31 3 16784 culler
Google Chrome Helper 0.0 10:15.75 21 2 16235 culler
System: 7.07%
User: 3.88%
Idle: 89.05%
CPU LOAD
Threads: 2587
Processes: 434

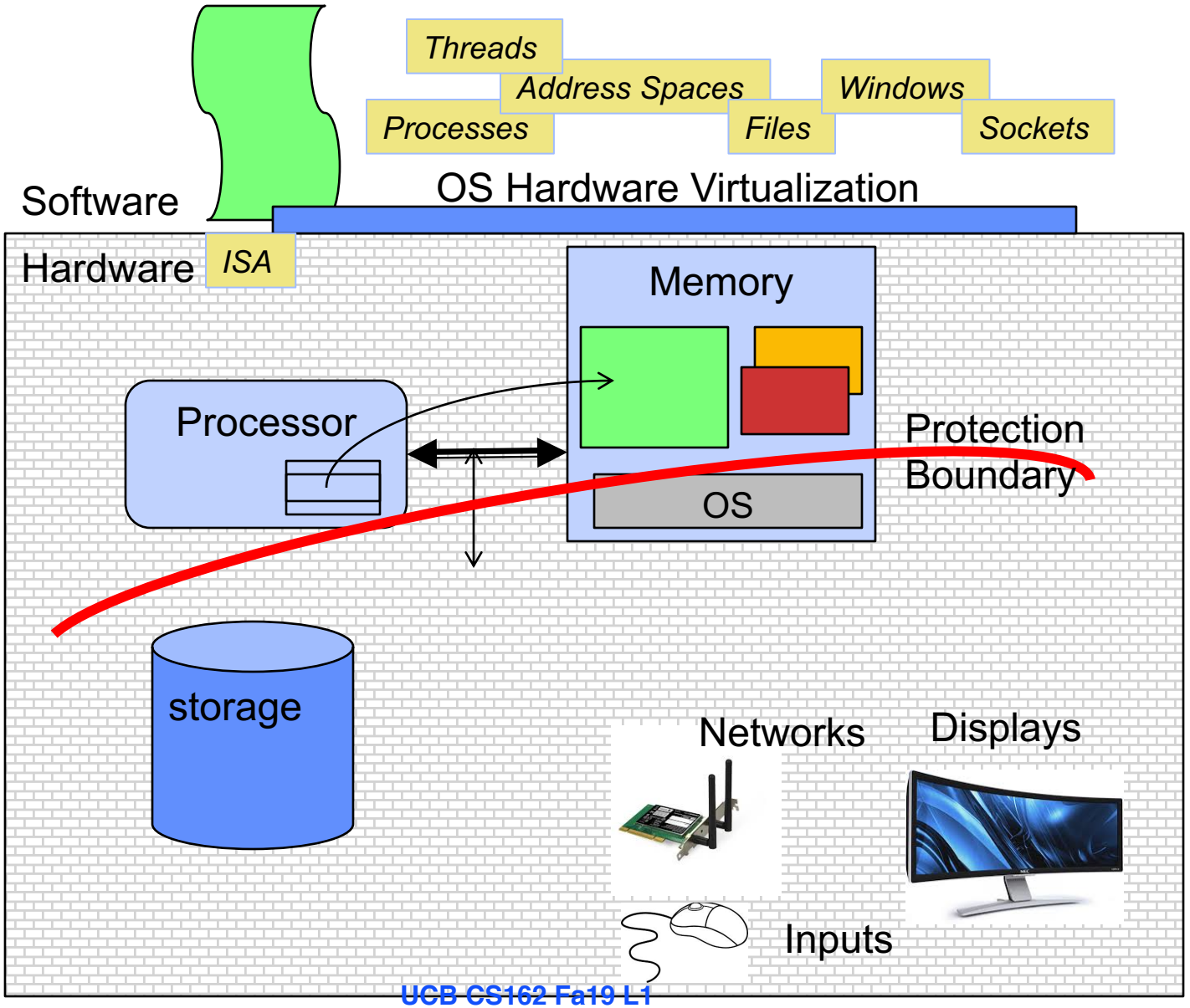
OS Basics: Program => Process



OS Basics: Context Switch



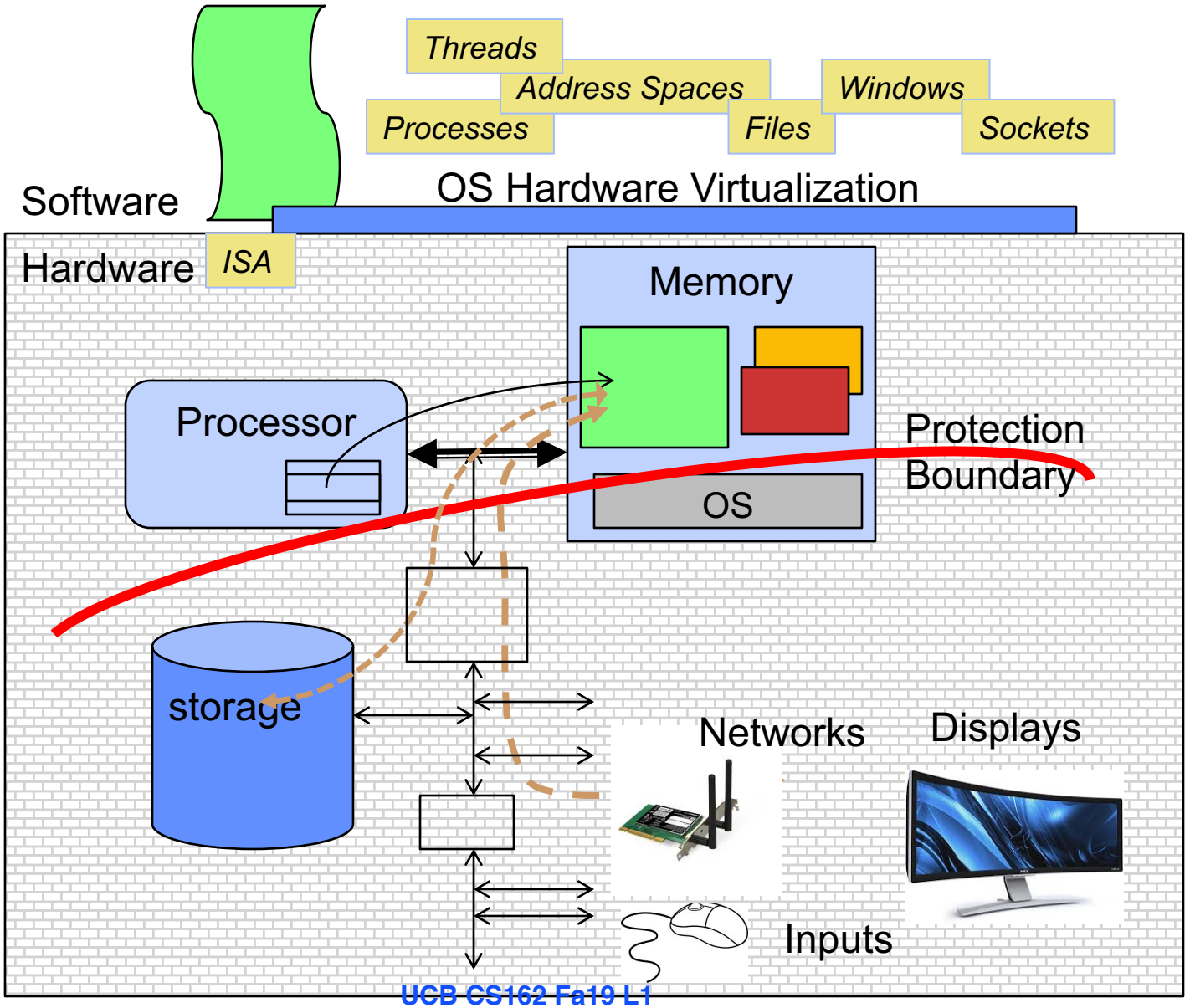
OS Basics: Scheduling, Protection



What is an Operating System?

- Referee
  - Manage sharing of resources, Protection, Isolation
    - » Resource allocation, isolation, communication
- Illusionist
  - Provide clean, easy to use abstractions of physical resources
    - » Infinite memory, dedicated machine
    - » Higher level objects: files, users, messages
    - » Masking limitations, virtualization

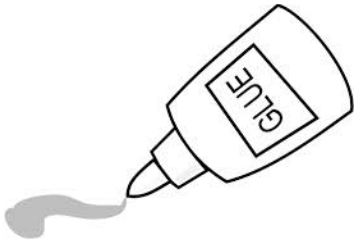
OS Basics: I/O


Threads
Address Spaces
Windows
Processes
Files
Sockets
Software
OS Hardware Virtualization
Hardware
ISA
Memory
Processor
Protection
Boundary
OS
storage
Networks
Displays
Inputs

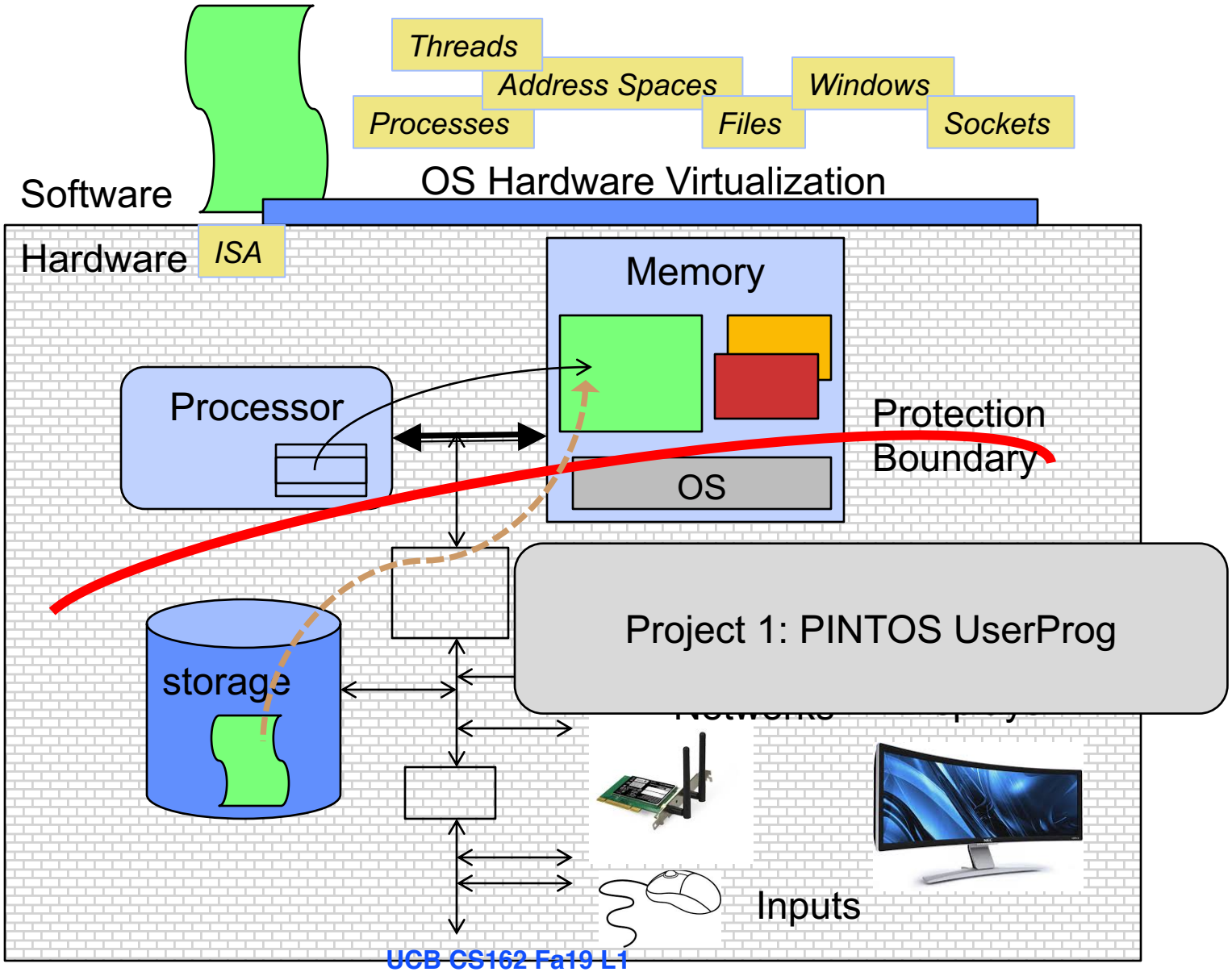
What is an Operating System?

- Referee
  - Manage sharing of resources, Protection, Isolation
    - » Resource allocation, isolation, communication
- Illusionist
  - Provide clean, easy to use abstractions of physical resources
    - » Infinite memory, dedicated machine
    - » Higher level objects: files, users, messages
    - » Masking limitations, virtualization

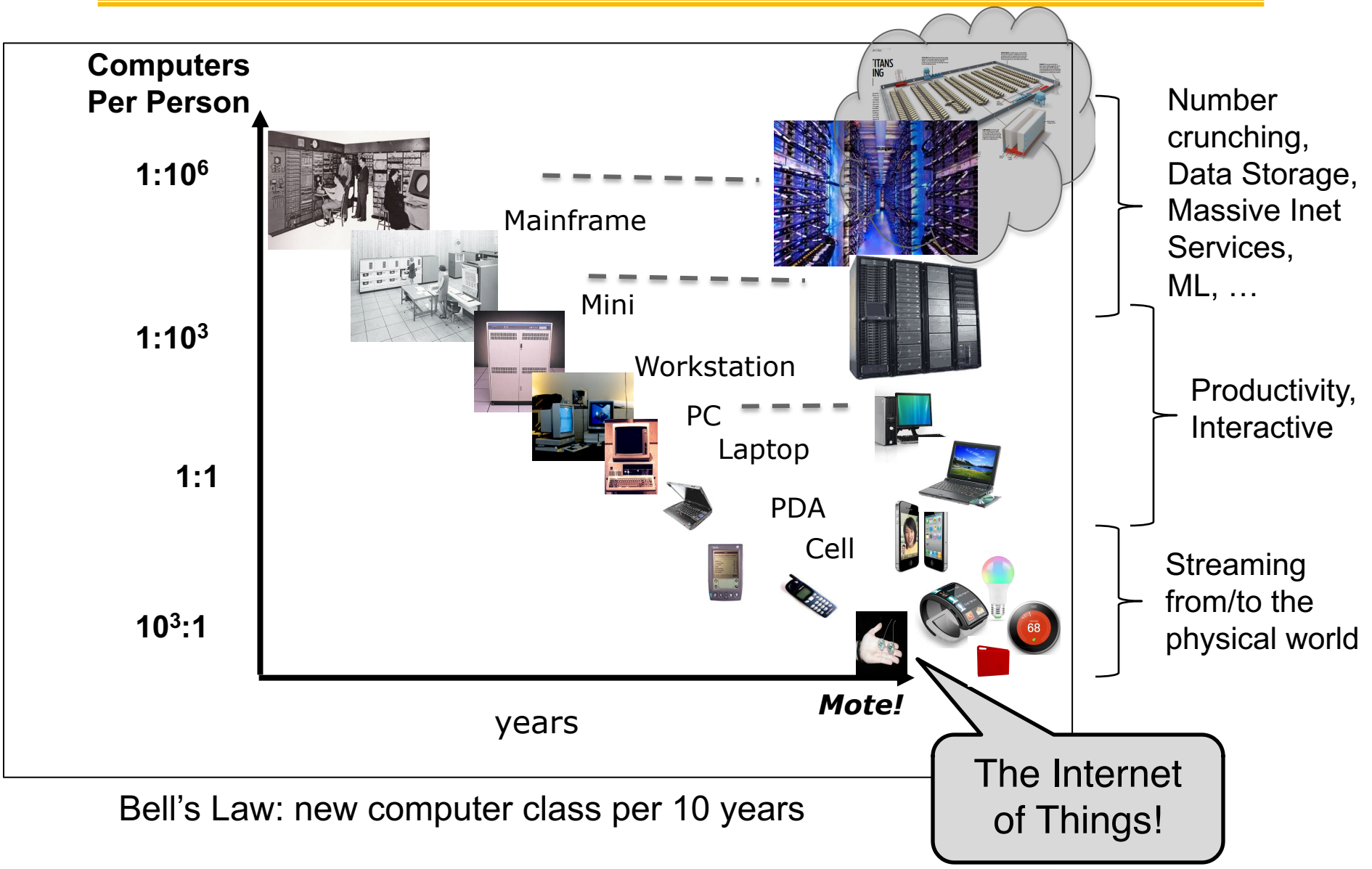
- Glue
  - Common services
    - » Storage, Window system, Networking
    - » Sharing, Authorization
    - » Look and feel



Creating Process & Loading a Program



Across incredibly diversity

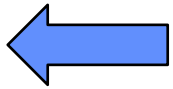


Bell's Law: new computer class per 10 years

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

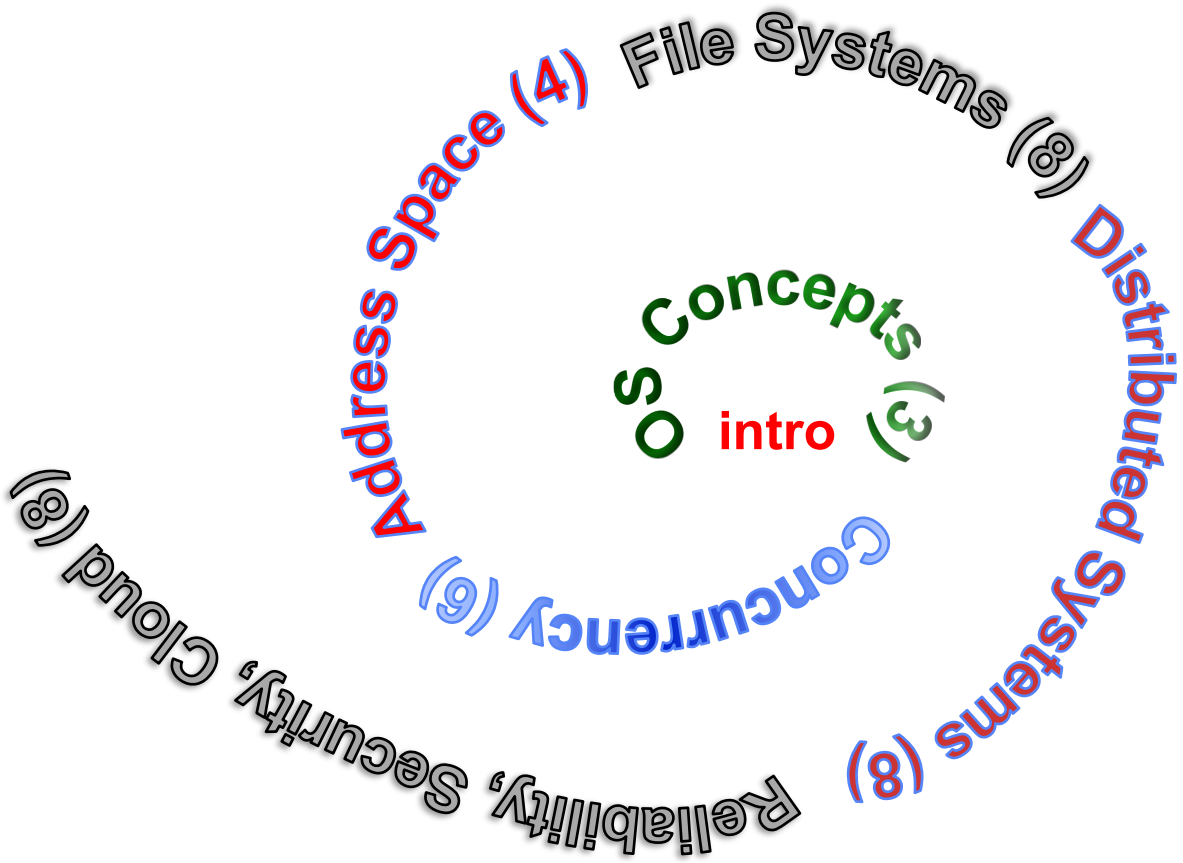


Key Stroke / Click
100 ms

Operating systems

- Provide consistent abstractions to apps, even on diverse hardware
  - File systems, Window Systems, Communications, …
  - Processes, threads
  - VMs, containers
  - Naming systems
- Manage resources shared among multiple applications:
  - Memory, CPU, storage, …
- Achieved by specific algorithms and techniques:
  - Scheduling
  - Concurrency
  - Transactions
  - Security
- Across immense scale – from 1's to Billions

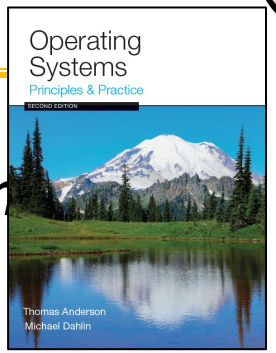
Course Structure: Spiral



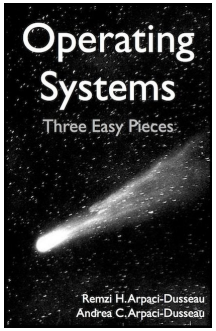
Syllabus – in nutshell

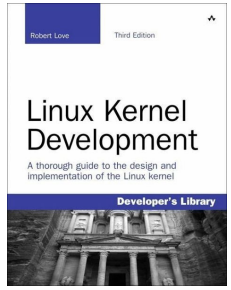
- OS Concepts
  - Process, I/O, Networks and VM
- Concurrency
  - Threads, scheduling, locks, deadlock, scalability, fairness
- Address Space
  - Virtual memory, address translation, protection, sharing
- File Systems
  - i/o devices, file objects, storage, naming, caching, performance, paging, transactions,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

Textbook & Resources



- **Course text: *Operating Systems, Principles an Practice*, 2nd ed., by Anderson and Dahlin**
- **Recommended Supplementary Material:**
  - ***Operating Systems: Three Easy Pieces*, by Remzi and Andrea Arpaci-Dusseau, available for free online at: http://pages.cs.wisc.edu/~remzi/OSTEP**
  - ***Linux Kernel Development*, 3rd ed., by Robert Love**
- **Additional materials on course website "Readings" page**





Infrastructure

- **Course Website:** https://cs162.eecs.berkeley.edu
- **Piazza for Q&A:** https://piazza.com/berkeley/spring2019/cs162
- **Anonymous Feedback Form:**
  - **http://bit.ly/cs162fa19anon**

Learning by Doing

- **Three Group Projects (4 weeks, Pintos in C)**
  - **1. User-programs (exec & syscall)**
  - **1. Threads & Scheduling**
  - **3. File Systems**

  } streamlined
- **Individual Homeworks (2 weeks) - preliminary**
  - **0. Tools & Environment, Autograding, recall C, executable**
  - **1. Getting to Pintos-scale C, Basic Threads**
  - **2. BYOS – build your own shell**
  - **3. Threads and Synchronization**
  - **4. Sockets & Threads in HTTP server**
  - **5. Memory mapping and management**
  - **6. Distributed Computing – KV store in Kubernetes**
  - **7. Cloud APIs, GRPC**
- **Weekly quick surveys**
  - **Ungraded, reinforce material, instructional diagnostic**

Steeping in GO …

- Last one or two homeworks in Go
- Nudges to learn Go at your own pace along the way

Group Projects

- **Project teams have 4 members**
  - never 5, 3 req's serious justification)
  - Must work in groups in "the real world"
  - Same section (at least same TA)
- **Communication and cooperation will be essential**
  - 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 project
  - 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

Grading

- 36% exams
- 36% projects
- 22% homework
- 6% participation
- Project grading
  - Initial design document, Design review, Code, Final design
- Submission via *git push* to release branch
  - Triggers autograder
- Regular *git push* so TA sees your progress

Getting started

- **Start homework 0 immediately**
  - **Get cs162-xx account via https://inst.eecs.berkeley.edu/webacct**
  - **Github account**
  - **Registration survey**
  - **Vagrant virtualbox – VM environment for the course**
    - » **Consistent, managed environment on your machine**
  - **Get familiar with all the cs162 tools**
  - **Submit to autograder via git**
- **Go to any section this week**
- **Waitlist**
  - **Pull hw0, get inst acct, go to a section**
  - **Only enrolled students will form project groups**
  - **If cs162 is not for you now, please allow others to take it**
  - **We are going to try to grow more to enroll majors from current waitlist**

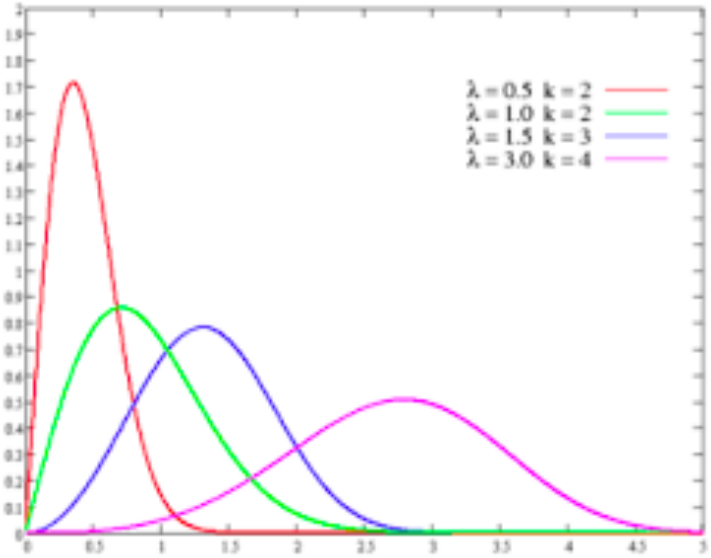
Preparing Yourself for this Class

- **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**
- **C / 61C review session Tuesday 9/3 7-8pm @ 310 Jacobs**

Preparing Yourself for this Class

- "Resources" page on course website
- C programming reference (still in beta):
  - https://cs162.eecs.berkeley.edu/ladder/
- This week's section dedicated to programming and debugging review
  - Attend ANY section this week
- Review session Tuesday evening
  - 306 Soda

Personal Growth …



- **Homeworks will try to offer optional "warm up" and "stretch" problems so you can tailor your learning to your situation**

Personal Integrity

- **UCB Academic Honor Code: "As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others."**

http://asuc.org/honorcode/resources/HC%20Guide%20for%20Syllabi.pdf

Collaboration Policy

OK:

- Explaining a concept 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)

Not OK:

- 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

Late Policy

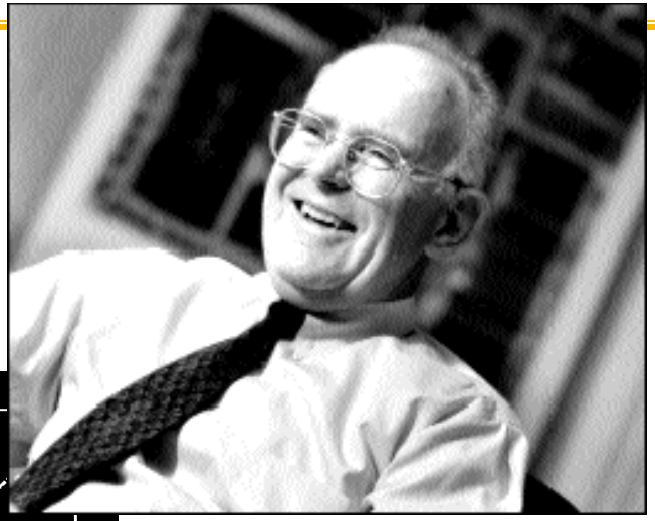
- Deadlines are at 9:00 PM Pacific Time
- 4 slip days to use for homeworks, no more than 1 per assignment
- 4 slip days to use (as a group) for projects code, no more than 2 per project (no slip on report)
- No credit for late work beyond these

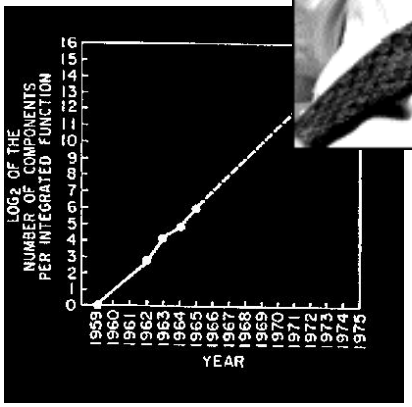
Questions

What make Operating Systems So Exciting and Challenging ?

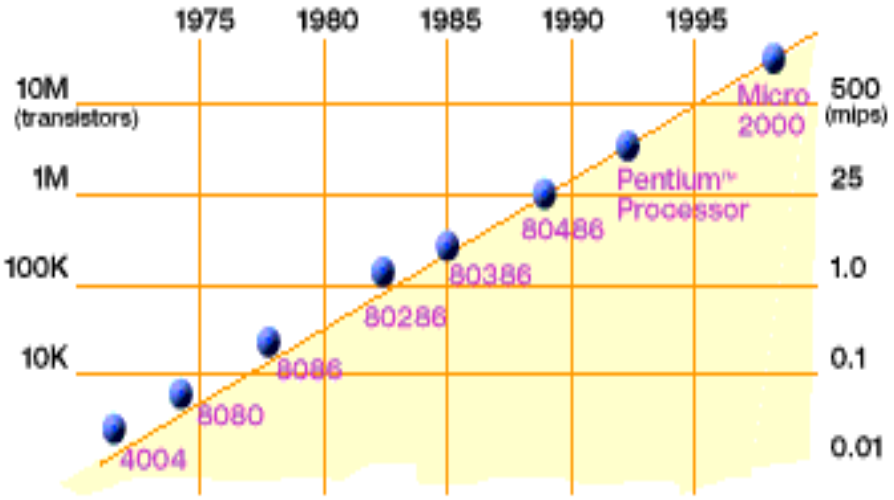
Managing complexity

Technology Trends: Moore's Law





Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

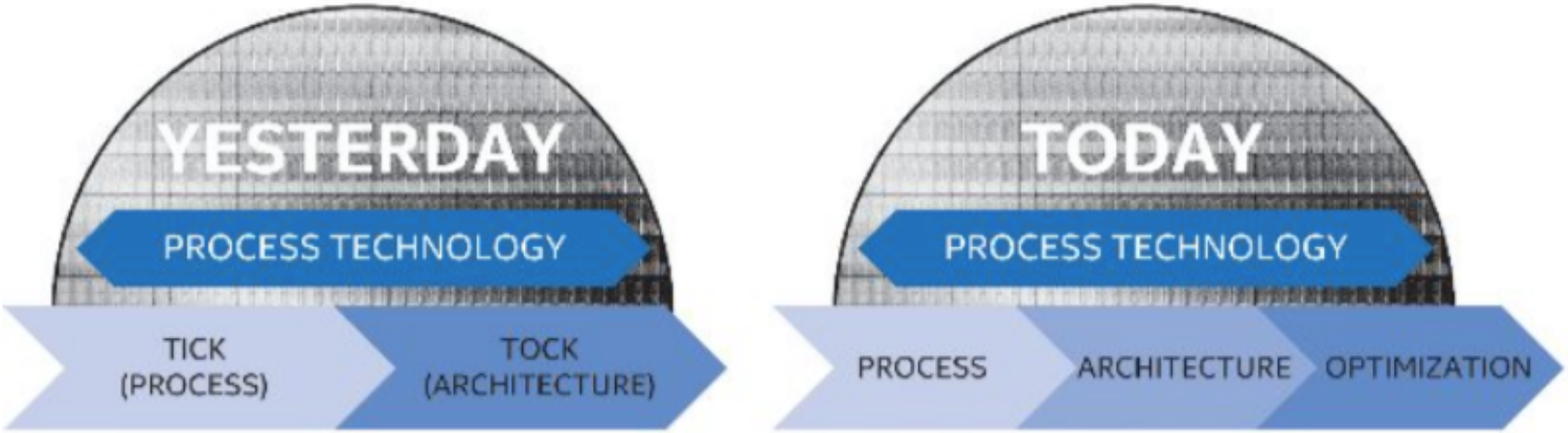


2X transistors/Chip Every 1.5 years

Called "Moore's Law"

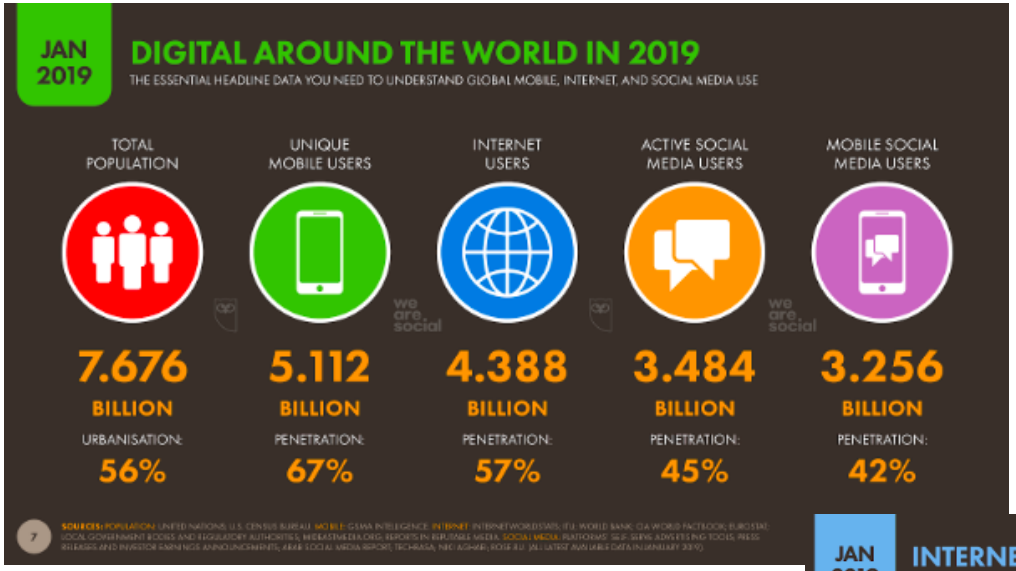
Microprocessors have become smaller, denser, and more powerful.

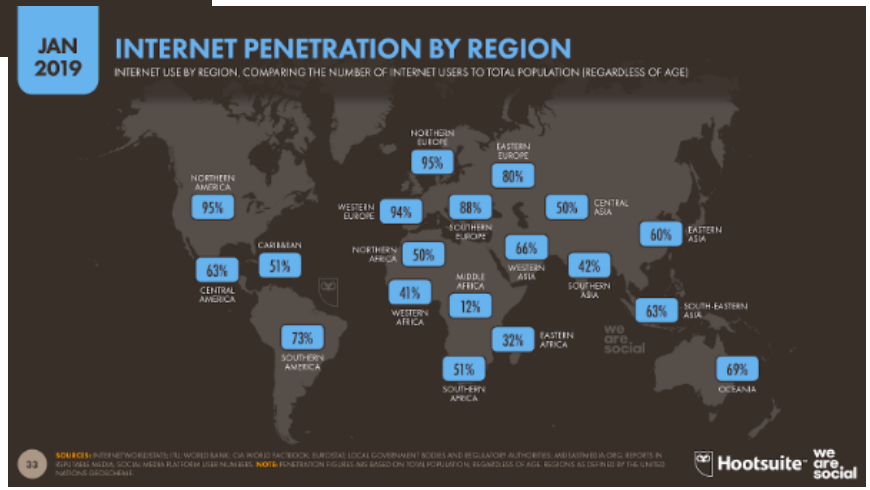
But then Moore's Law Ended…



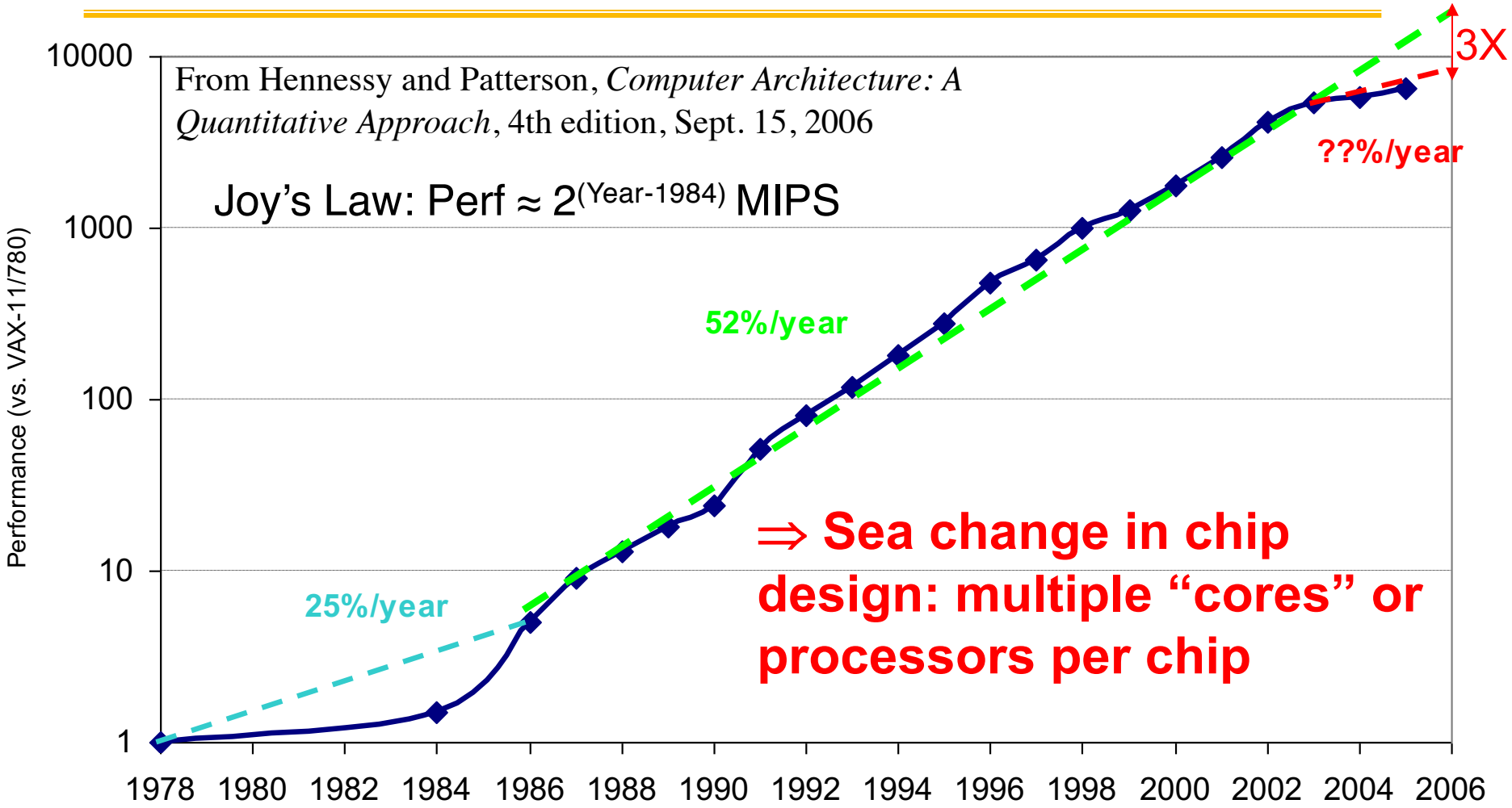
- **Moore's Law has (officially) ended -- Feb 2016**
  - **No longer getting 2 x transistors/chip every 18 months…**
  - **or even every 24 months**
- **May have only 2-3 smallest geometry fabrication plants left:**
  - **Intel and Samsung and/or TSMC**
- **Vendors moving to 3D stacked chips**
  - **More layers in old geometries**

Society connected





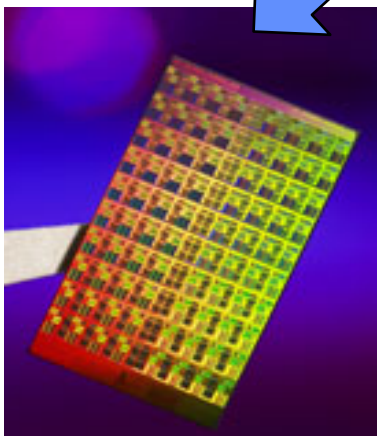
New Challenge: Slowdown in Joy's law of Performance

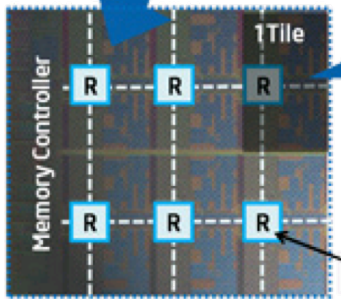


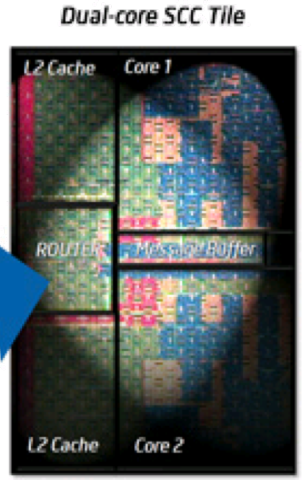
- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

ManyCore Chips: The future is here

- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two FP-engines / core
  - Mesh-like network
  - 100 million transistors



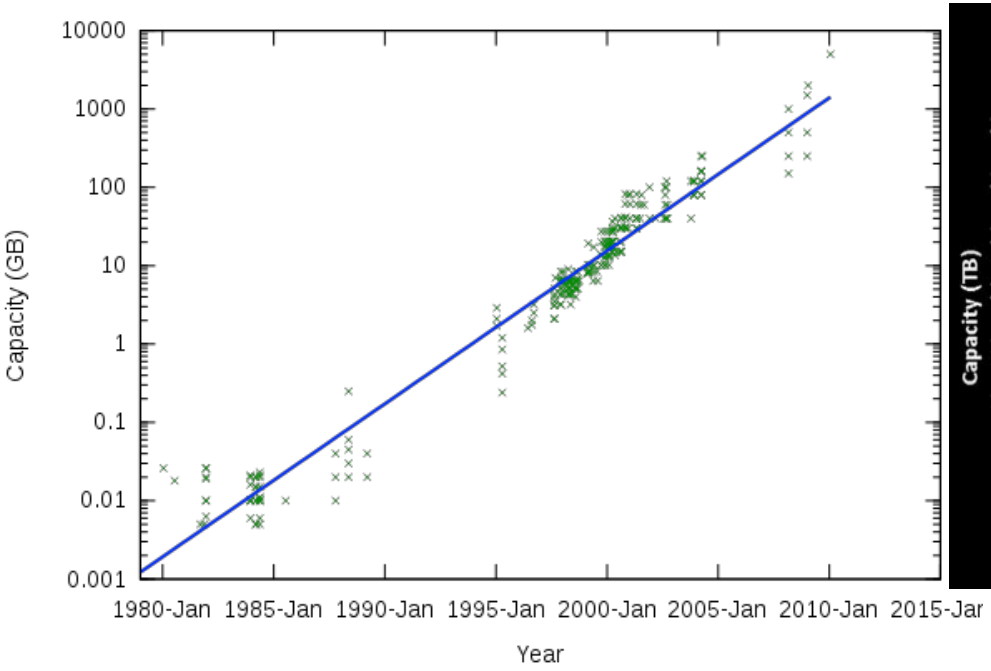


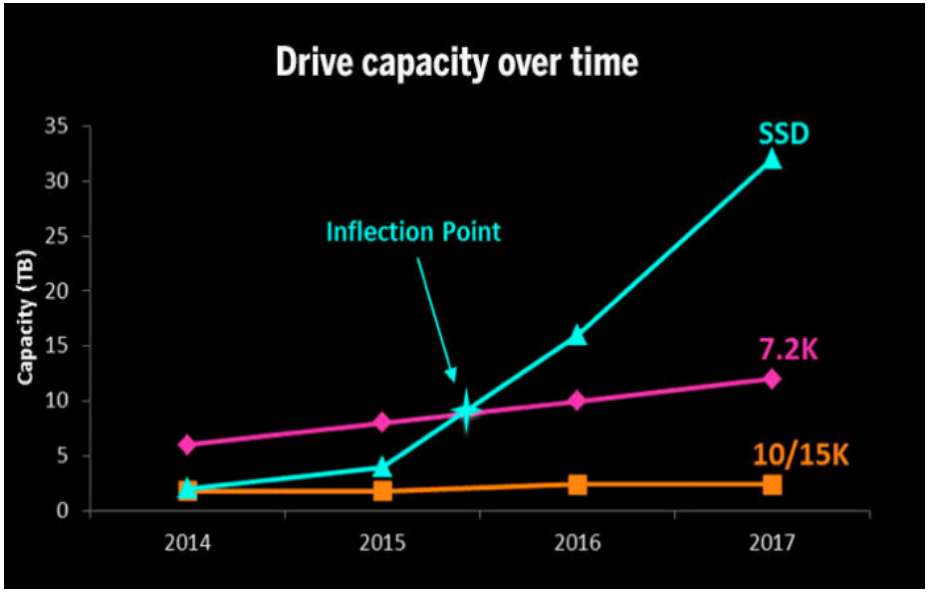


- Intel Single-Chip Cloud Computer (August 2010)
  - 24 "tiles" with two cores/tile
  - 24-router mesh network
  - 4 DDR3 memory controllers
  - Hardware support for message-passing

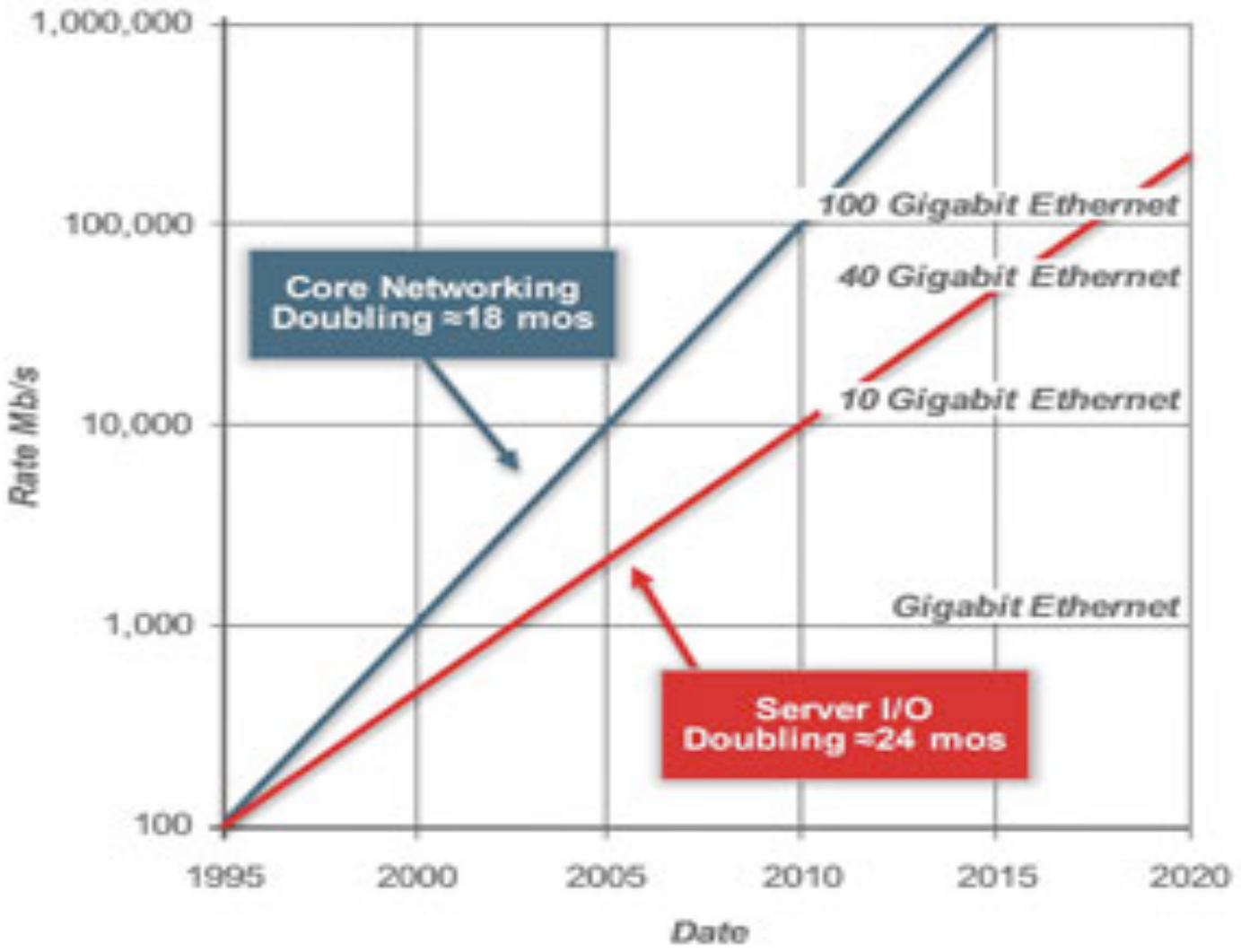
- **"ManyCore" refers to many processors/chip**
  - **64? 128? Hard to say exact boundary**
- **How to program these?**
  - **Use 2 CPUs for video/audio**
  - **Use 1 for word processor, 1 for browser**
  - **76 for virus checking???**
- **Parallelism must be exploited at all levels**

Storage Capacity





Network Capacity



(source: http://www.ospmag.com/issue/article/Time-Is-Not-Always-On-Our-Side)

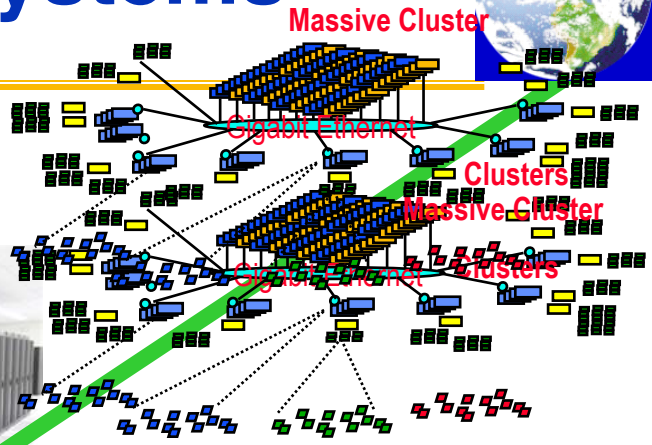
Not Only PCs connected to the Internet

- **In 2011, smartphone shipments exceeded PC shipments!**
- **2011 shipments:**
  - **487M smartphones**
  - **414M PC clients**
    - » **210M notebooks**
    - » **112M desktops**
    - » **63M tablets**
  - **25M smart TVs**

- **4 billion phones in the world → smartphones over next few years**
- **Then…**

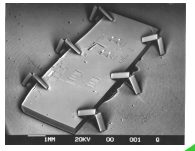
Societal Scale Information Systems

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



Scalable, Reliable, Secure Services

Internet Connectivity



Databases
Information Collection
Remote Storage
Online Games
Commerce
…

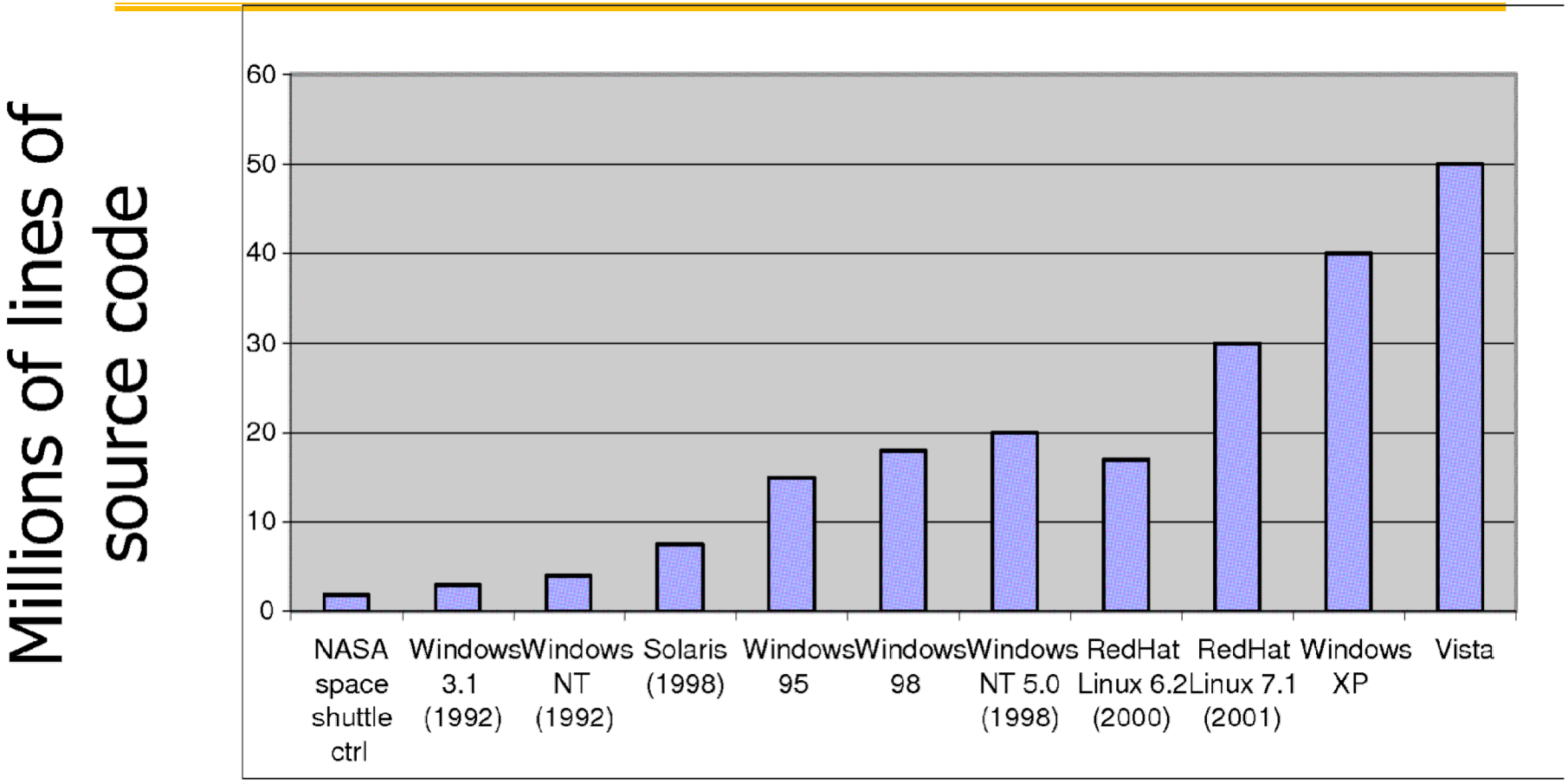
MEMS for Sensor Nets

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 well are they managed.

Increasing Software Complexity



From MIT's 6.033 course

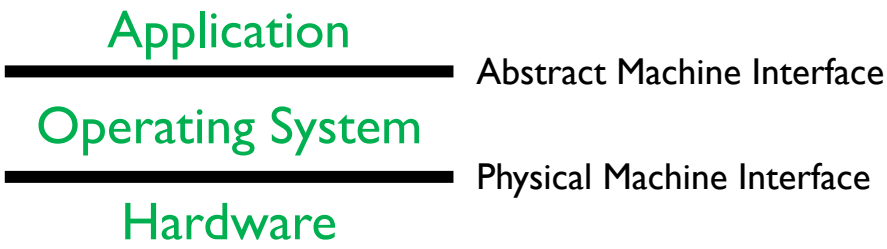
How do We Tame Complexity?

- Every piece of computer hardware different
  - Different CPU
    - » 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 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?

Virtualization ??

OS *Abstracts* underlying hardware

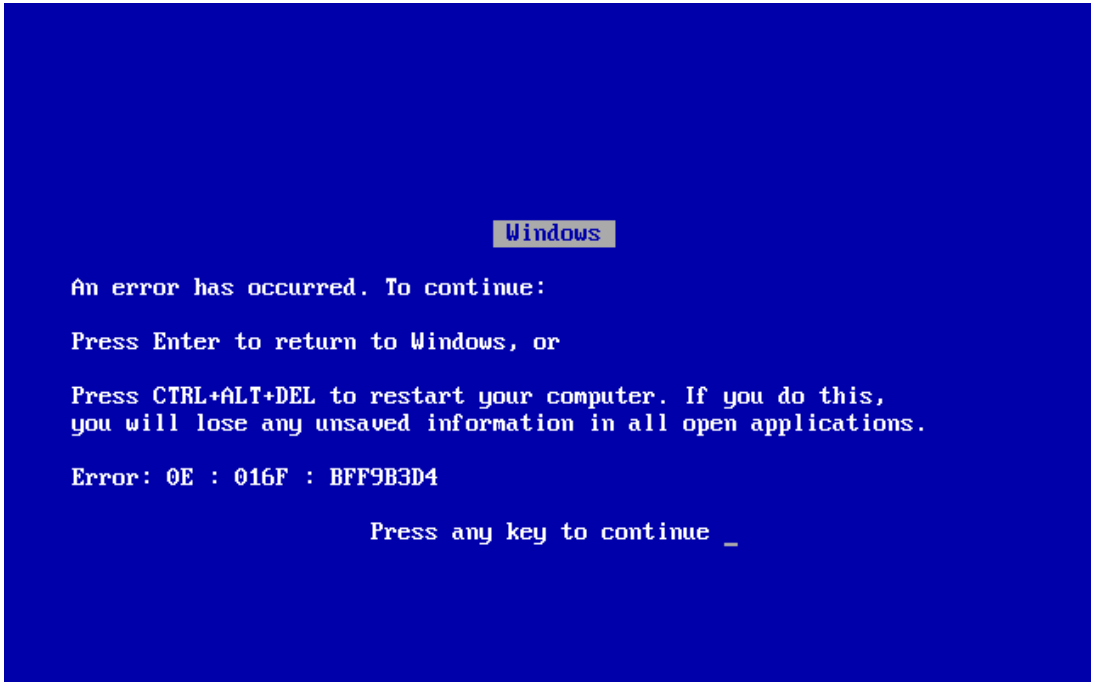
- Processor => Thread
- Memory => Address Space
- Disks, SSDs, … => Files
- Networks => Sockets
- Machines => Processes



- **OS Goals:**
  - Remove software/hardware quirks (*fight complexity)*
  - Optimize for convenience, utilization, reliability, … *(help the programmer)*
- **For any OS area (e.g. file systems, virtual memory, networking, scheduling):**
  - What hardware interface to handle? (physical reality)
  - What's software interface to provide? (nicer abstraction)

OS Goal: Protecting Processes & The Kernel

- **Run multiple applications and:**
  - **Keep them from interfering with or crashing the operating system**
  - **Keep them from interfering with or crashing each other**

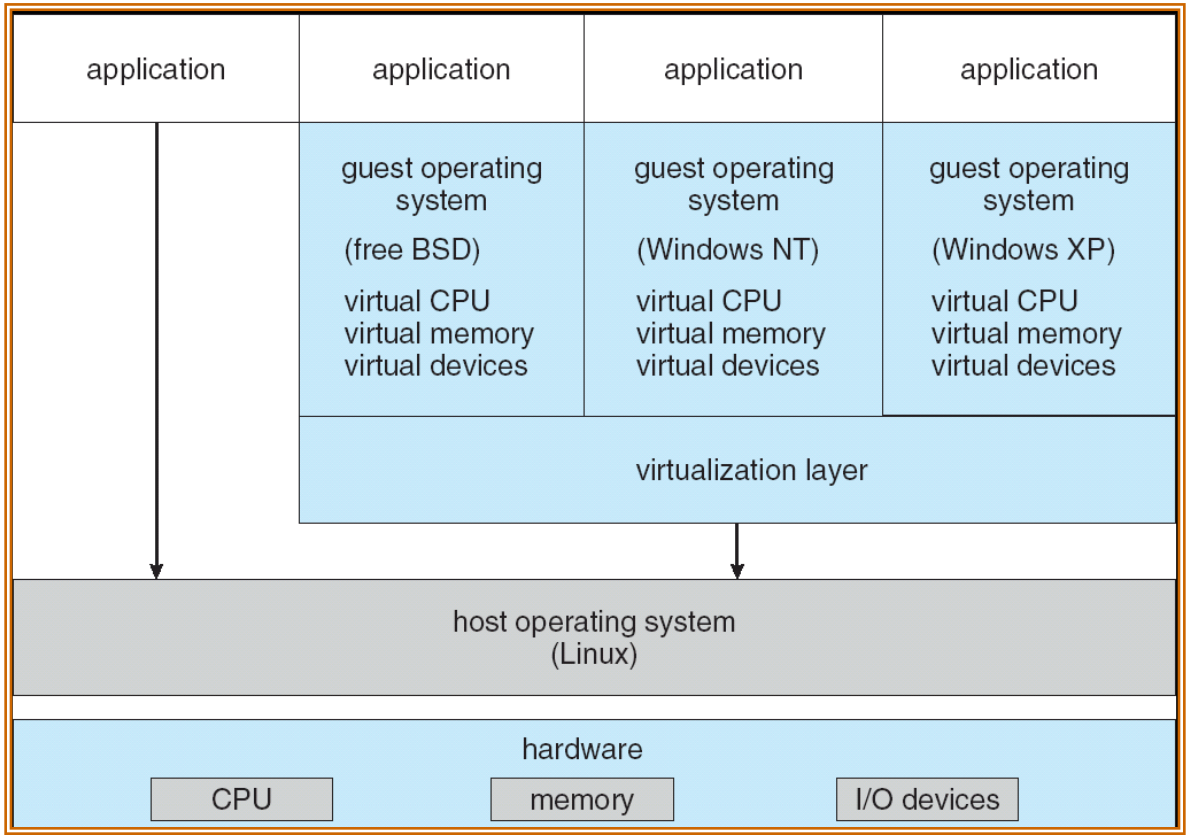


Virtual Machines

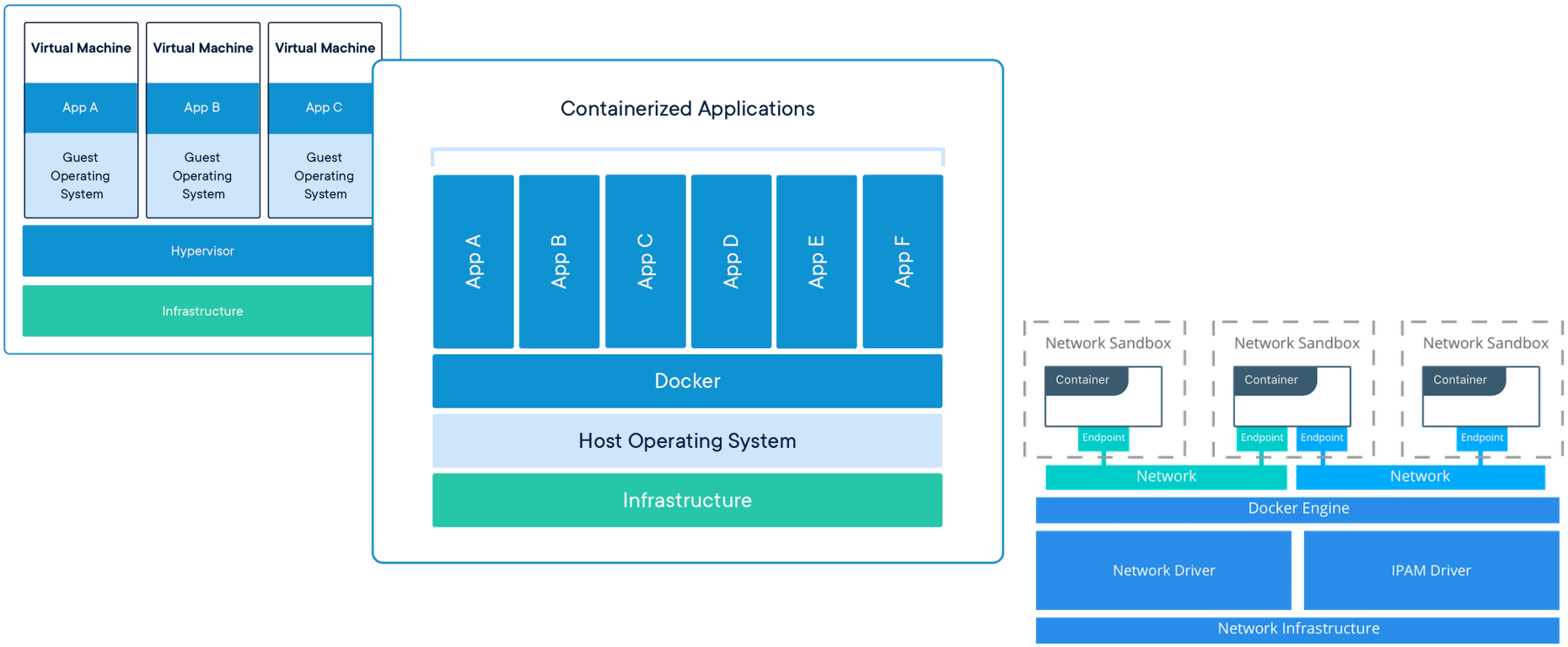
- Virtualize every detail of a hardware configuration so perfectly that you can run an operating system (and many applications) on top of it.
  - VMWare Fusion, Virtual box, Parallels Desktop, Xen, Vagrant
- Provides isolation
- Complete insulation from change
- The norm in the Cloud (server consolidation)
- Long history (60's in IBM OS development)
- All our work will take place INSIDE a VM
  - Vagrant (new image just for you)

System Virtual Machines: Layers of OSs

- **Useful for OS development**
  - **When OS crashes, restricted to one VM**
  - **Can aid testing programs on other OSs**



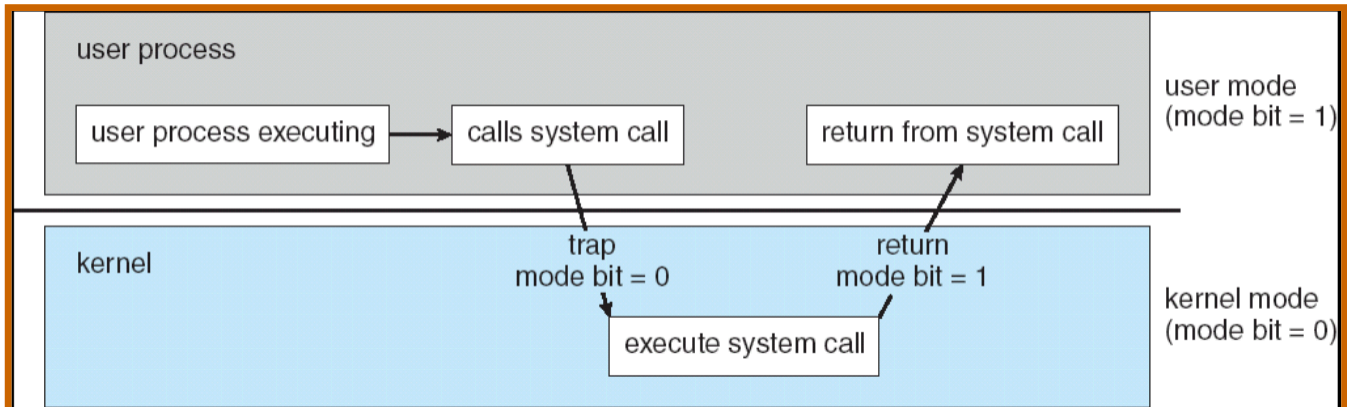
Containers *virtualize the OS*



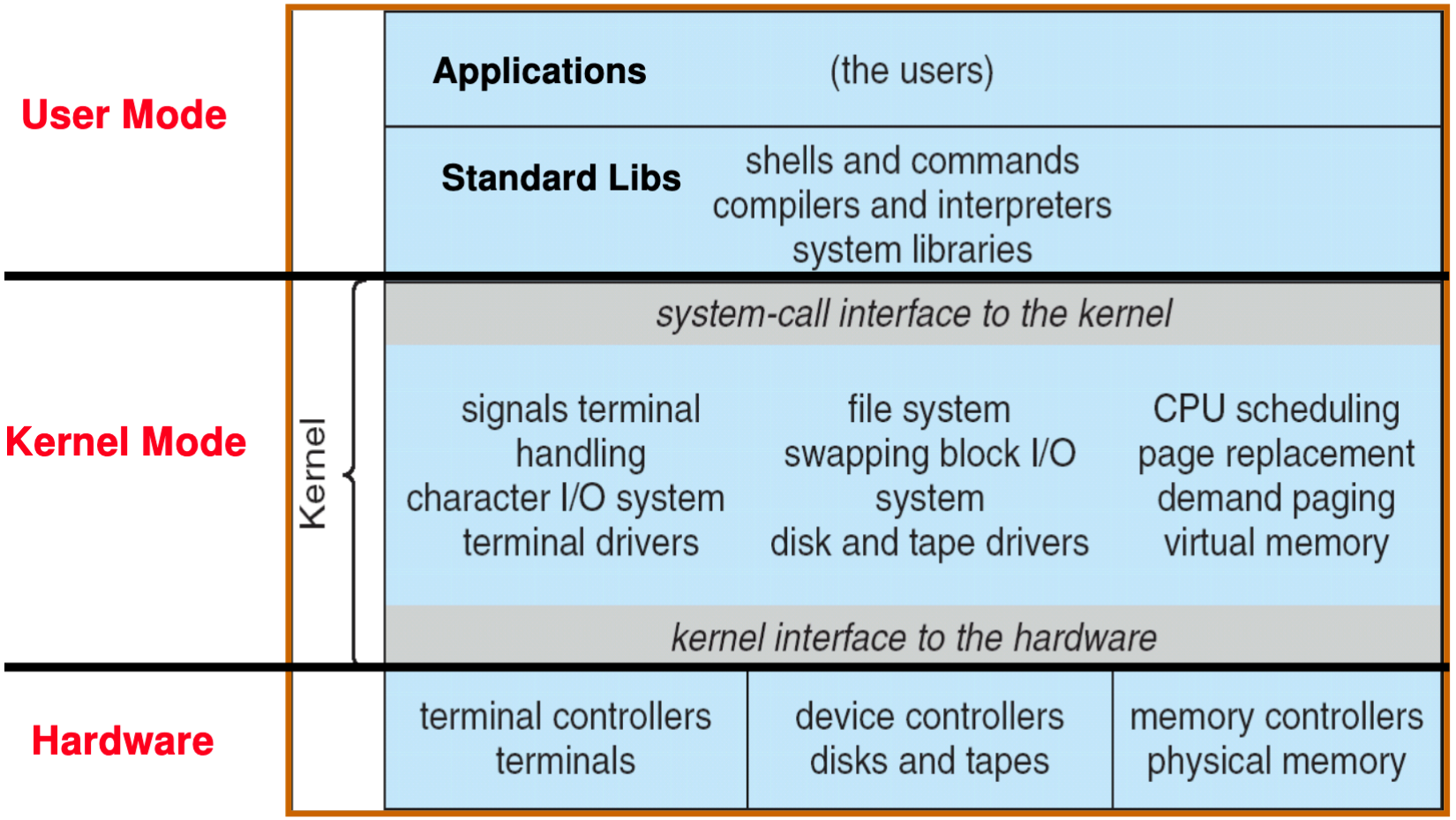
- **Roots in OS developments to provide protected systems abstraction, not just application abstraction**
  - **User-level file system (route syscalls to user process)**
  - **Cgroups – predictable, bounded resources (CPU, Mem, BW)**

Basic tool: Dual Mode Operation

- **Hardware provides at least two modes:**
  1. **Kernel Mode (or "supervisor" / "protected" mode)**
  2. **User Mode**
- **Certain operations are prohibited when running in user mode**
  - **Changing the page table pointer**
- **Carefully controlled transitions between user mode and kernel mode**
  - **System calls, interrupts, exceptions**



UNIX OS Structure



Summary: Operating Systems…

- Provide a machine abstraction to handle diverse hardware
  - Convenience, protection, reliability obtain in creating the illusion
- Coordinate resources and protect users from each other
  - Utilizing a few critical hardware mechanisms
- Simplify application development by providing standard services
- Provide fault containment, fault tolerance, and fault recovery
- Key concepts: concurrency, isolation, protected sharing, virtualization