# ICS 143A - Principles of Operating Systems (Spring 2020)

**Lecture 1 - Introduction and Overview**

**MWF 11:00- 11:50 a.m.**

**Prof. Nalini Venkatasubramanian**

**( nalini@uci.edu )**

**[*lecture slides contains some content adapted from : Silberschatz textbook authors, John Kubiatowicz (Berkeley) Anderson textbook, John Ousterhout(Stanford), Prof. Ardalan Sani, )*]**

# ICS 143A Spring 2020 Staff

Instructor:

Prof. Nalini Venkatasubramanian (Venkat)

( nalini@uci.edu )

Teaching Assistants:

Biswadeep Maity (Deep) (maityb@uci.edu)

Saehanseul Yi (Hans) (saehansy@uci.edu)

Andrew Chio (achio@uci.edu)

# Course logistics and details

- **Course Web page -**
  - http://www.ics.uci.edu/~ics143
  - All new announcements and updates will be on class webpage

- **Lectures** – MWF 11:00 – 11:50, on zoom
  - Zoom link: https://uci.zoom.us/j/568633190

- **Discussions**
  - Tuesday 5pm - 6pm;  7pm- 8pm
  - Zoom Link:
    - 5 - 6 pm session:  https://uci.zoom.us/j/556715006
    - 7- 8  pm session:  https://uci.zoom.us/j/563851913

# Course logistics and details

- **ICS 143 Textbook**
  - Operating System Concepts – 9th Edition, Silberschatz, Galvin and Gagne, Addison-Wesley Inc (Eighth, Seventh,Sixth and Fifth editions, and Java Versions are fine ).


- **Other Suggested Books**
  - Modern Operating Systems, by Tanenbaum
  - Lubomir Bic, online book -- zybooks
    - https://www.zybooks.com/catalog/operating-systems/
  - Operating Systems: Principles and Practice, by T. Anderson and M. Dahlin (second edition)
- **Piazza for group discussions** (https://piazza.com/uci/spring2020/cs143)
  - Q/A, Student discussions, monitored by ICS143A staff
  - Avoid duplicate posts, Use clear, descriptive titles
  - Please be respectful to each other

# Course logistics and details

- Homeworks and Assignments
  - 4 written homeworks in the quarter
  - 1 programming assignment (knowledge of C, C++ or Java required).
    - Handed out at midterm; submit during Finals Week
    - Multistep assignment – don't start in last week of classes!!!
  - Late homeworks will not be accepted.
  - All submissions will be made using Gradescope for the course
- Tests
  - Midterm - tentatively Wednesday, Week 6
    - Detailed format and tools to be used -- TBD
  - Final Exam - as per UCI course catalog
    - June 9th (1:30 - 3:30 pm)
    - Detailed format and tools to be used -- TBD

# ICS 143 Grading Policy

- Homeworks - 40% (10% each)
- Programming Assignment - 10%
  - released Week 6
- Midterm - 20% of the final grade
  - Tentatively Wed, Week 6 during lecture
- Final exam - 30% of the final grade
  - Per UCI course catalog, Jun 9th (1:30 - 3:30 pm)
- 3 In-class micro-quizzes (not graded, except in case of borderline grades)

Final assignment of grades will be based on a curve.

# Lecture Schedule

- Week 1:
  - Introduction to Operating Systems, Computer System Structures, Operating System Structures
- Week 2 : Process Management
  - Processes and Threads, CPU Scheduling
- Week 3: Process Management
  - CPU Scheduling, Process Synchronization
- Week 4: Process Management
  - Process Synchronization
- Week 5: Process Management
  - Process Synchronization, Deadlocks

# Course Schedule

- Week 6 – Deadlocks, Storage Management
  - Deadlocks, Midterm revision, exam
- Week 7 - Storage Management
  - Memory Management, Paging, Segmentation
- Week 8 – Storage Management
  - Virtual Memory
- Week 9 - FileSystems
  - Virtual Memory, FileSystems Interface and Implementation
- Week 10 – I/O Subsystems
  - Filesystems, I/O, course revision and summary.

# Other Logistics

- Office Hours
  - Prof. V - Tuesdays (11 am - 12 noon)
  - TA office hours – Mon, Wed. Thurs (to be announced on website)
- Slides
  - Available as draft before lecture, minor modifications likely
- If there are missed lectures
  - Alternatives will be arranged

# Introduction

- What is an operating system?
- Operating Systems History
  - Simple Batch Systems
  - Multiprogrammed Batch Systems
  - Time-sharing Systems
  - Personal Computer Systems
- Parallel and Distributed Systems
- Real-time Systems

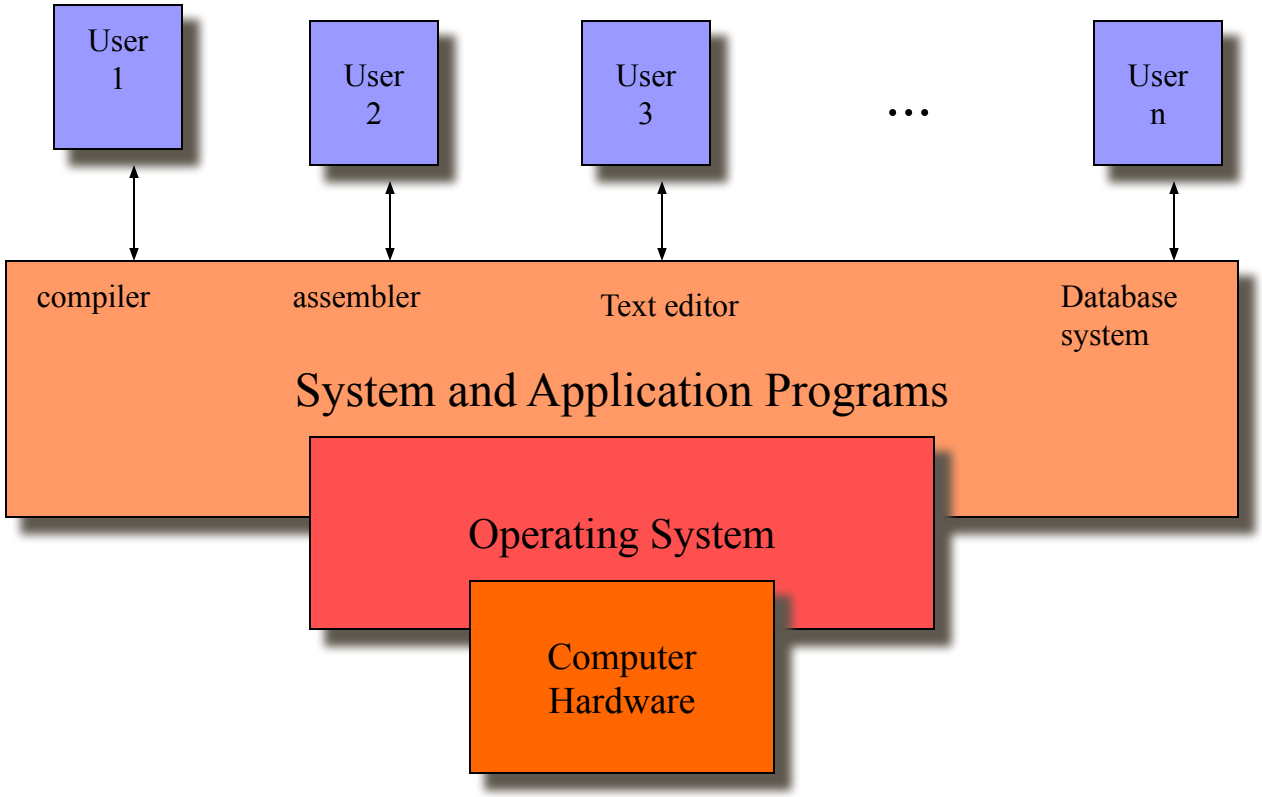
# What is an Operating System?

- An OS is a program that acts an intermediary between the user of a computer and computer hardware.
- Major cost of general purpose computing is software.
  - OS simplifies and manages the complexity of running application programs efficiently.

# Computer System Components

- Hardware
  - Provides basic computing resources (CPU, memory, I/O devices).
- Operating System
  - Controls and coordinates the use of hardware among application programs.
- Application Programs
  - Solve computing problems of users (compilers, database systems, video games, business programs such as banking software).
- Users
  - People, machines, other computers

# Abstract View of System



# Operating System Views

- Resource allocator
  - to allocate resources (software and hardware) of the computer system and manage them efficiently.
- Control program
  - Controls execution of user programs and operation of I/O devices.
- Kernel
  - The program that executes forever (everything else is an application with respect to the kernel).

# Operating system roles

- **Referee**
  - Resource allocation among users, applications
  - Isolation of different users, applications from each other
  - Communication between users, applications
- **Illusionist**
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport
- **Glue**
  - Libraries, user interface widgets, …
  - Reduces cost of developing software

# Example: file systems

- Referee
  - Prevent users from accessing each other's files without permission
- Illusionist
  - Files can grow (nearly) arbitrarily large
  - Files persist even when the machine crashes in the middle of a save
- Glue
  - Named directories, printf, …

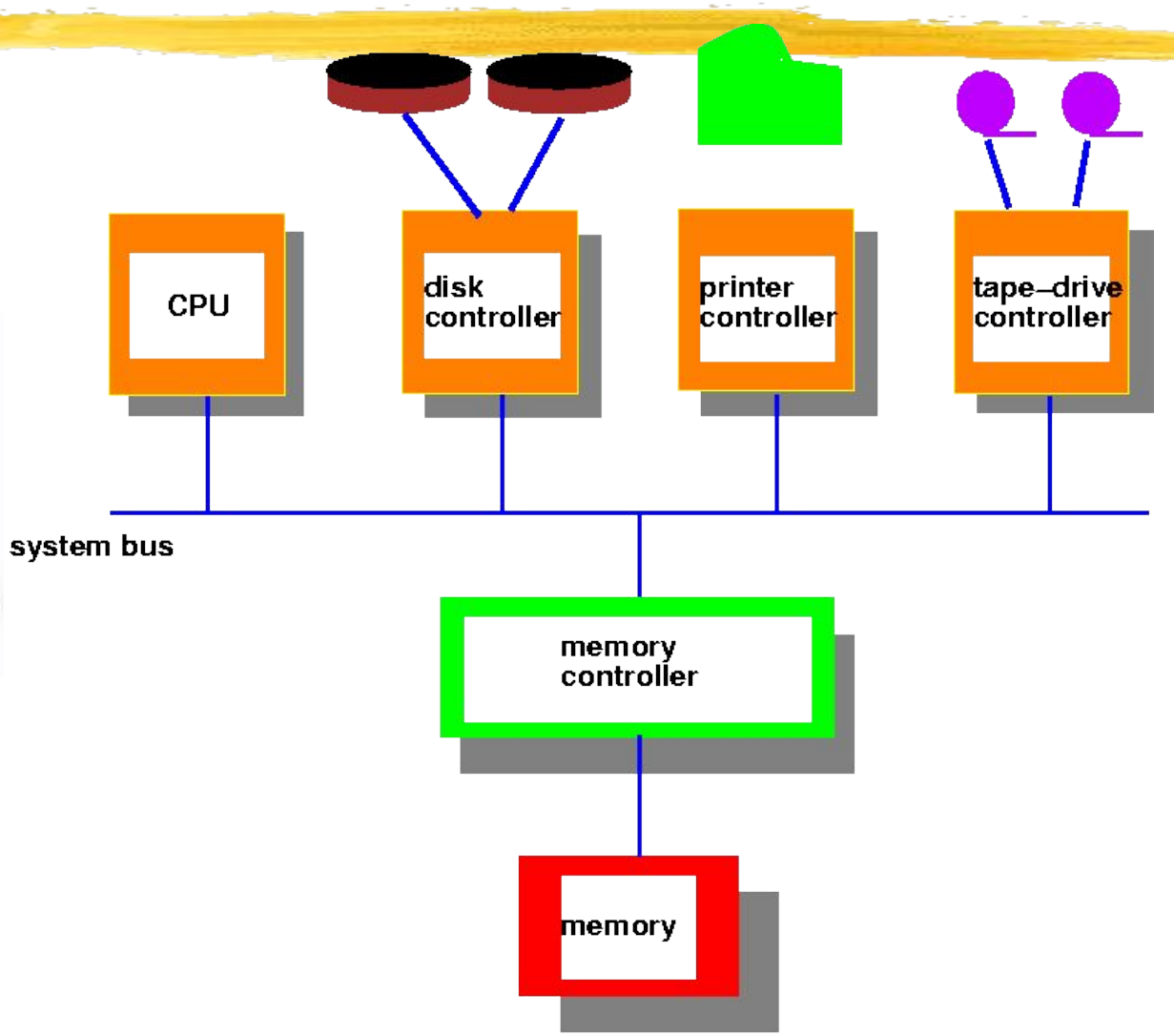
# Goals of an Operating System

- Simplify the execution of user programs and make solving user problems easier.
- Use computer hardware efficiently.
  - Allow sharing of hardware and software resources.
- Make application software portable and versatile.
- Provide isolation, security and protection among user programs.
- Improve overall system reliability
    - error confinement, fault tolerance, reconfiguration.

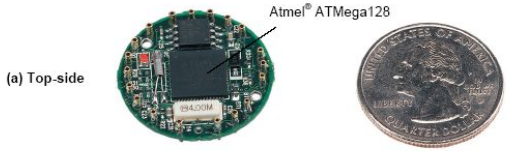
# Why should I study Operating Systems?

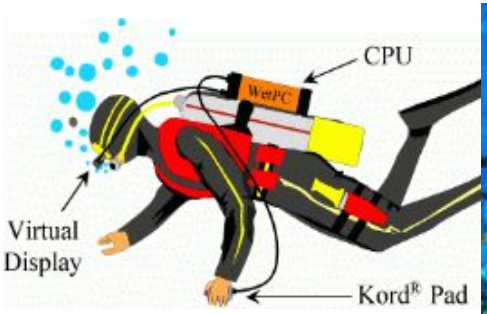
- Need to understand interaction between the hardware and applications
  - New applications, new hardware..
  - Inherent aspect of society today
- Need to understand basic principles in the design of computer systems
  - efficient resource management, security, flexibility
- Increasing need for specialized operating systems
  - e.g. embedded operating systems for devices - cell phones, sensors and controllers
  - real-time operating systems - aircraft control, multimedia services

# Computer System Architecture (traditional)

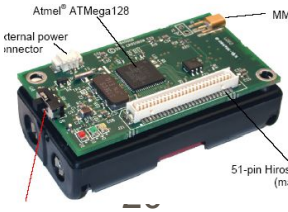


# Systems Today





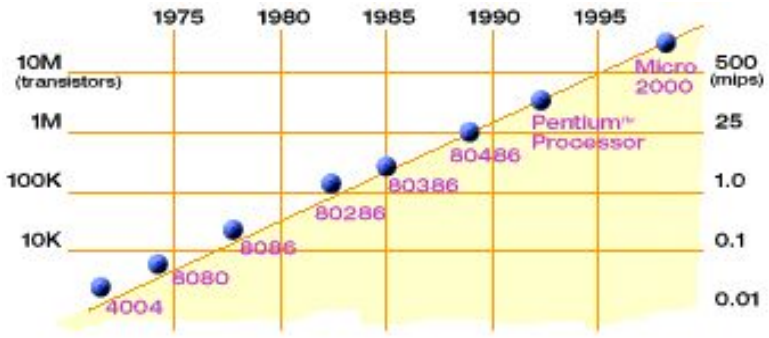


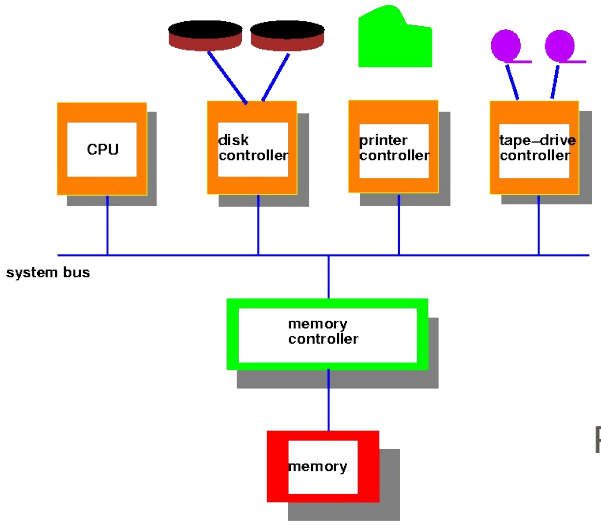



Irvine Sensorium

# Hardware Complexity Increases

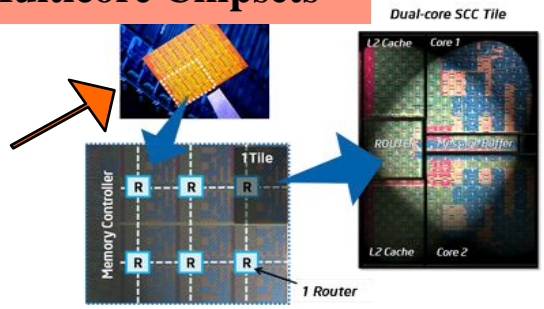
**Moore's Law**: 2X transistors/Chip Every 1.5 years

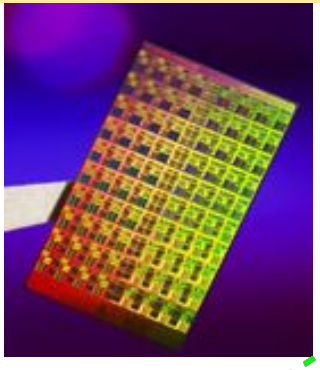


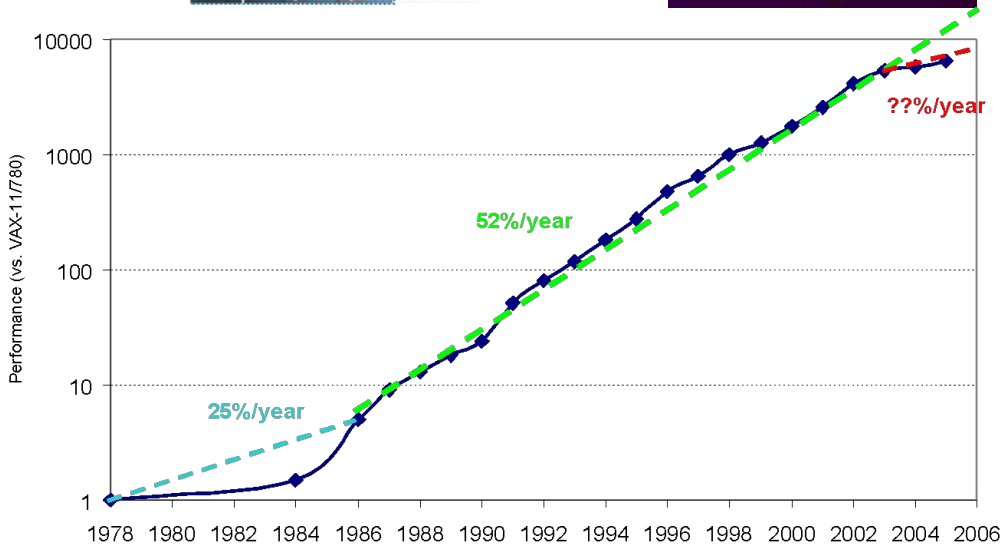




*From Berkeley OS course*

**Intel Multicore Chipsets**



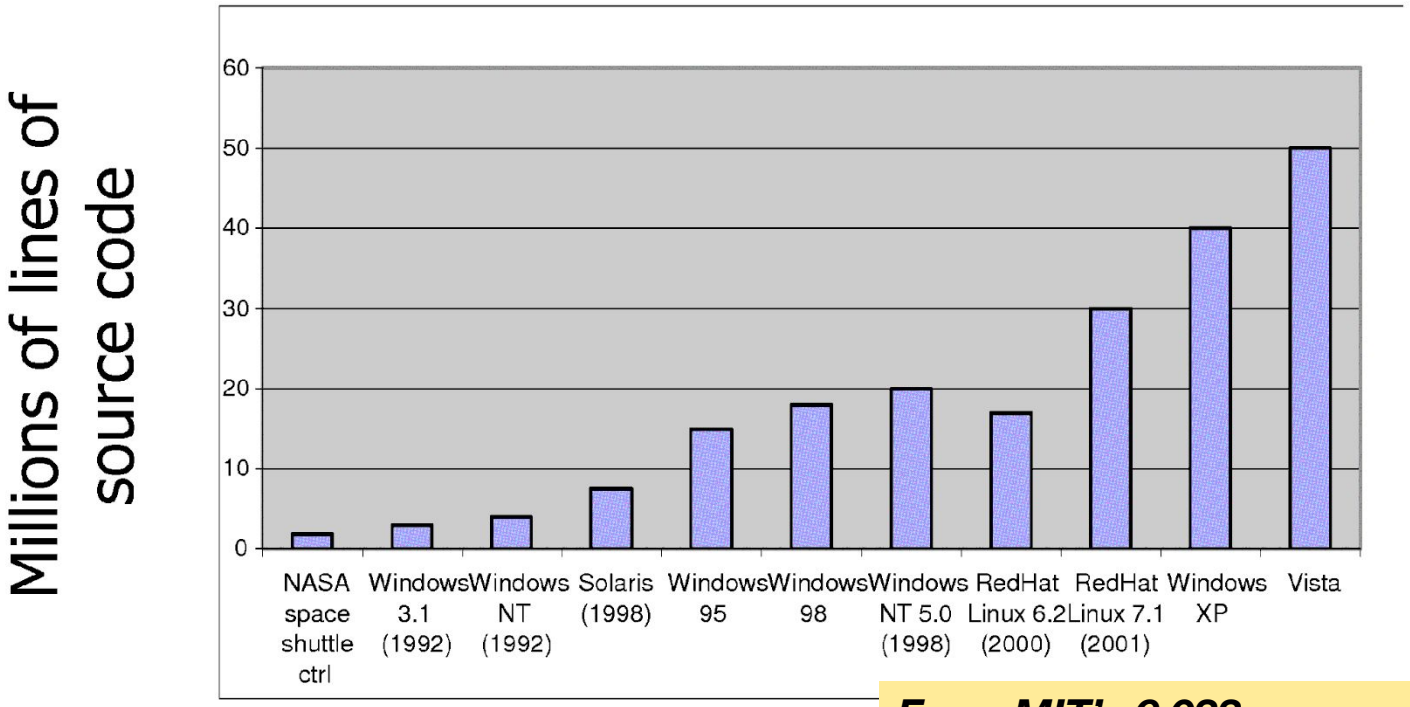



From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Sept. 15, 2006

# OS needs to keep pace with hardware improvements

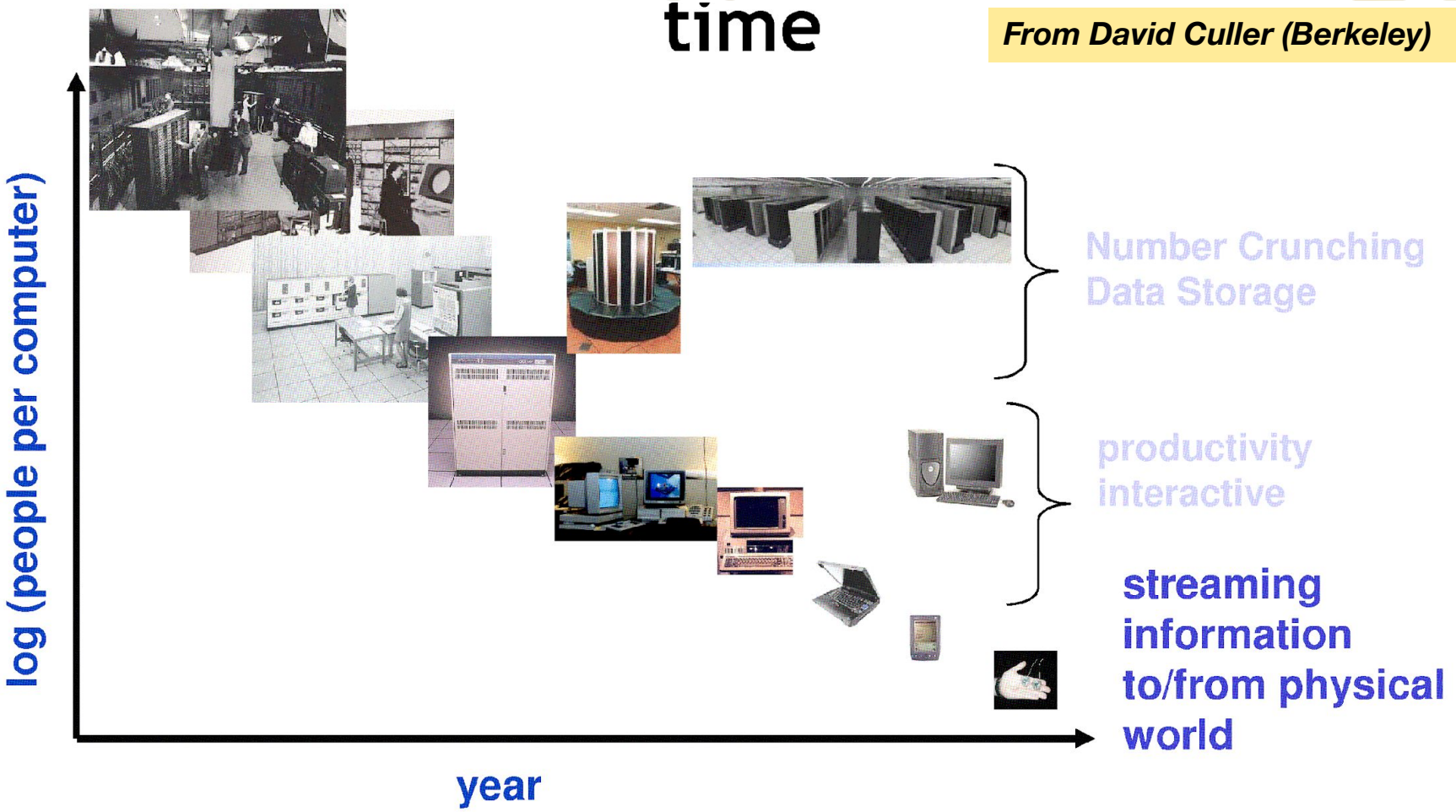
| | 1981 | 1997 | 2014 | Factor (2014/1981) |
|---|---|---|---|---|
| Uniprocessor speed (MIPS) | 1 | 200 | 2500 | 2.5K |
| CPUs per computer | 1 | 1 | 10+ | 10+ |
| $/Processor MIPS | $100K | $25 | $0.20 | 500K |
| DRAM Capacity (MiB)/$ | 0.002 | 2 | 1K | 500K |
| Disk Capacity (GiB)/$ | 0.003 | 7 | 25K | 10M |
| Home Internet | 300 bps | 256 Kbps | 20 Mbps | 100K |
| Machine room network | 10 Mbps (shared) | 100 Mbps (switched) | 10 Gbps (switched) | 1000 |
| Ratio of users to computers | 100:1 | 1:1 | 1:several | 100+ |

# Software Complexity Increases




***From MIT's 6.033 course***

# People-to-Computer Ratio Over Time

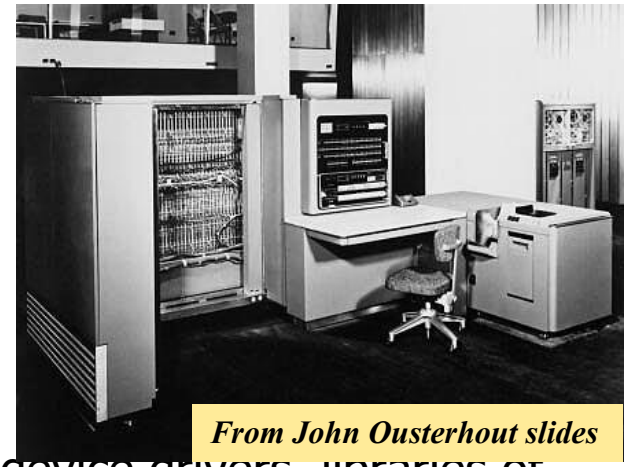


# Operating System Spectrum

- Monitors and Small Kernels
  - special purpose and embedded systems, real-time systems
- Batch and multiprogramming
- Timesharing
  - workstations, servers, minicomputers, timeframes
- Transaction systems
- Personal Computing Systems
- Mobile Platforms, devices (of all sizes)

# Early Systems - Bare Machine (1950s)

**Hardware – *expensive* ; Human – *cheap***



*From John Ousterhout slides*

- Structure
  - Large machines run from console
  - Single user system
    - Programmer/User as operator
  - Paper tape or punched cards
- Early software
  - Assemblers, compilers, linkers, loaders, device drivers, libraries of common subroutines.
- Secure execution
- Inefficient use of expensive resources
  - Low CPU utilization, high setup time.

# Simple Batch Systems (1960's)

- Reduce setup time by batching jobs with similar requirements.
- Add a card reader, Hire an operator
  - User is NOT the operator
  - Automatic job sequencing
    - Forms a rudimentary OS.
  - Resident Monitor
    - Holds initial control, control transfers to job and then back to monitor.
  - Problem
    - Need to distinguish job from job and data from program.

*From John Ousterhout slides*

# Supervisor/Operator Control

- Secure monitor that controls job processing
  - Special cards indicate what to do.
  - User program prevented from performing I/O
- Separate user from computer
  - User submits card deck
  - cards put on tape
  - tape processed by operator
  - output written to tape
  - tape printed on printer
- Problems
  - Long turnaround time - up to 2 DAYS!!!
  - Low CPU utilization
    - I/O and CPU could not overlap; slow mechanical devices.

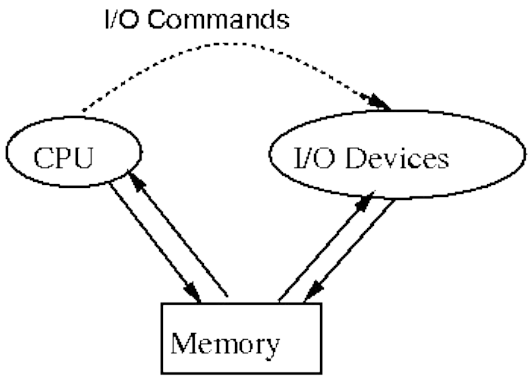

*From John Ousterhout slides*

# Batch Systems - Issues

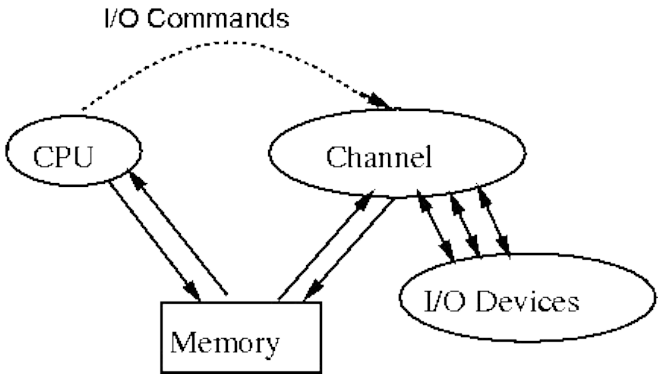
- Solutions to speed up I/O:
- Offline Processing
  - load jobs into memory from tapes, card reading and line printing are done offline.
- Spooling
  - Use disk (random access device) as large storage for reading as many input files as possible and storing output files until output devices are ready to accept them.
  - Allows overlap - I/O of one job with computation of another.
  - Introduces notion of a job pool that allows OS choose next job to run so as to increase CPU utilization.

# Speeding up I/O

- Direct Memory Access (DMA)




- Channels



# Batch Systems - I/O completion

- How do we know that I/O is complete?
  - Polling:
    - Device sets a flag when it is busy.
    - Program tests the flag in a loop waiting for completion of I/O.
  - Interrupts:
    - On completion of I/O, device forces CPU to jump to a specific instruction address that contains the interrupt service routine.
    - After the interrupt has been processed, CPU returns to code it was executing prior to servicing the interrupt.

# Multiprogramming

- Use interrupts to run multiple programs simultaneously
  - When a program performs I/O, instead of polling, execute another program till interrupt is received.
- Requires secure memory, I/O for each program.
- Requires intervention if program loops indefinitely.
- Requires CPU scheduling to choose the next job to run.

# Timesharing

**Hardware – *getting cheaper*; Human – *getting expensive***

- Programs queued for execution in FIFO order.
- Like multiprogramming, but timer device interrupts after a quantum (timeslice).
  - Interrupted program is returned to end of FIFO
  - Next program is taken from head of FIFO
- Control card interpreter replaced by command language interpreter.

# Timesharing (cont.)

- Interactive (action/response)
  - when OS finishes execution of one command, it seeks the next control statement from user.
- File systems
  - online filesystem is required for users to access data and code.
- Virtual memory
  - Job is swapped in  and out of memory to disk.

# Personal Computing Systems

**Hardware – *cheap* ; Human – *expensive***

- Single user systems, portable.
- I/O devices - keyboards, mice, display screens, small printers.
- Laptops and palmtops, Smart cards, Wireless devices.
- Single user systems may not need advanced CPU utilization or protection features.
- Advantages:
  - user convenience, responsiveness, ubiquitous

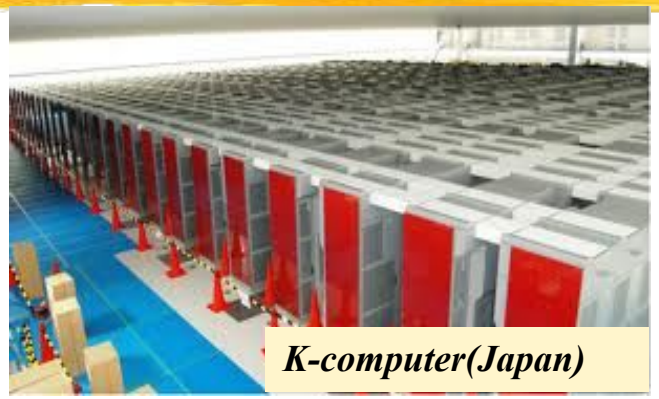
# Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.
- Improved Throughput, economical, increased reliability.
- Kinds:
  - Vector and pipelined
  - Symmetric and asymmetric multiprocessing
  - Distributed memory vs. shared memory
- Programming models:
  - Tightly coupled vs. loosely coupled ,message-based vs. shared variable

# Parallel Computing Systems

*ILLIAC 2 (UIllinois)*

*Climate modeling, earthquake simulations, genome analysis, protein folding, nuclear fusion research, …..*



*K-computer(Japan)*

*Connection Machine (MIT)*

*Tianhe-1(China)*

*IBM Blue Gene*

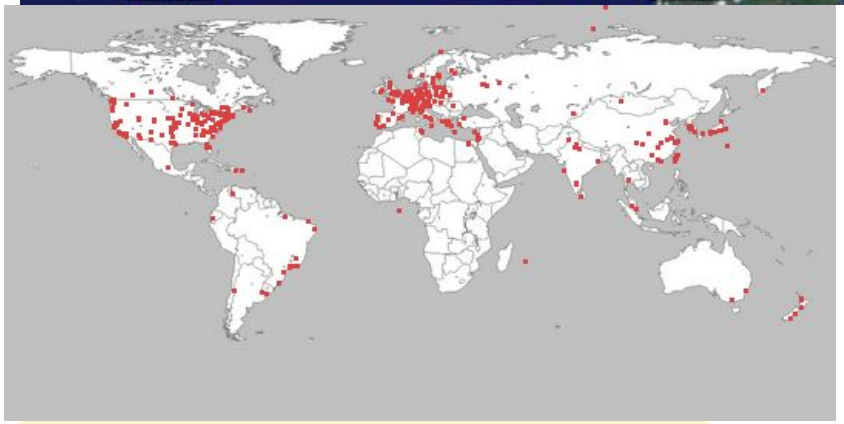
# Distributed Systems

**Hardware – *very cheap* ; Human – *very expensive***

- Distribute computation among many processors.
- Loosely coupled -
  - no shared memory, various communication lines
- client/server architectures
- Advantages:
  - resource sharing
  - computation speed-up
  - reliability
  - communication - e.g. email
- Applications - digital libraries, digital multimedia

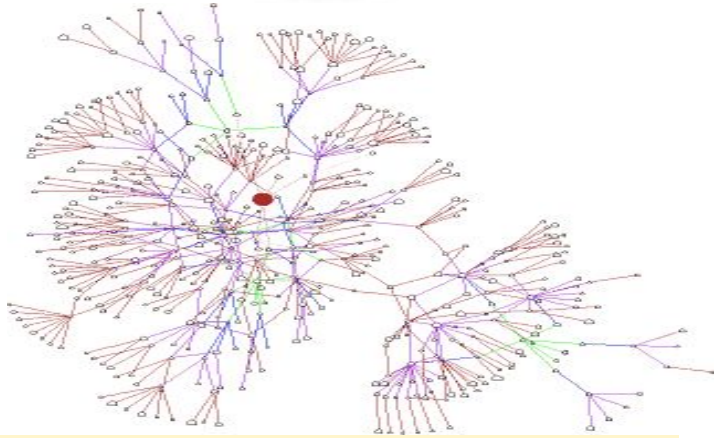
# Distributed Computing Systems

*Globus Grid Computing Toolkit*

*PlanetLab*

*Cloud Computing Offerings*

*Gnutella P2P Network*

# Real-time systems

- Correct system function depends on timeliness
- Feedback/control loops
- Sensors and actuators
- Hard real-time systems -
  - Failure if response time too long.
  - Secondary storage is limited
- Soft real-time systems -
  - Less accurate if response time is too long.
  - Useful in applications such as multimedia, virtual reality.

# A personal computer today


*interaction*

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen

- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared

- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture Sensor
- GPS

# A personal computer today


- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen

- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared

- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture Sensor
- GPS

***sensing***

# A personal computer today


- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen

- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared

*connectivity*

- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture Sensor
- GPS

# A personal computer today


- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen

- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared

- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP *acceleration*
- Multimedia processor

- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture Sensor
- GPS

# Operating systems are everywhere



# Operating systems are everywhere





# Info-tainment!!



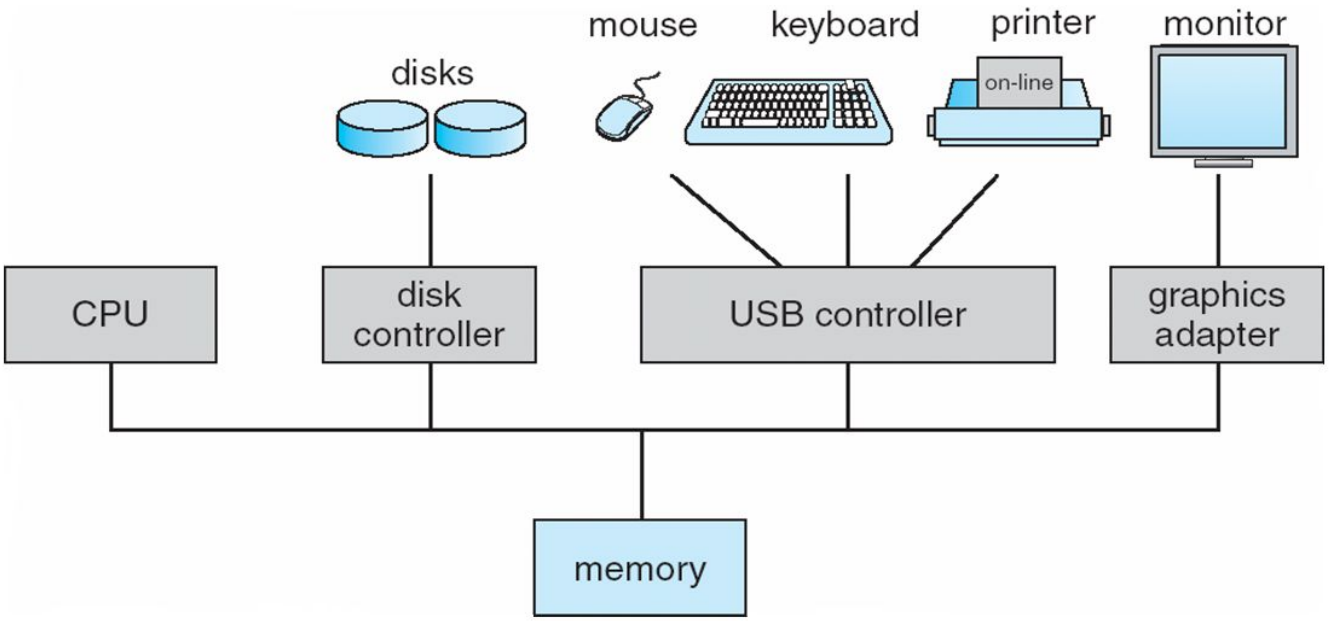
# Summary of lecture

- What is an operating system?
- Early Operating Systems
- Simple Batch Systems
- Multiprogrammed Batch Systems
- Time-sharing Systems
- Personal Computer Systems
- Parallel and Distributed Systems
- Real-time Systems

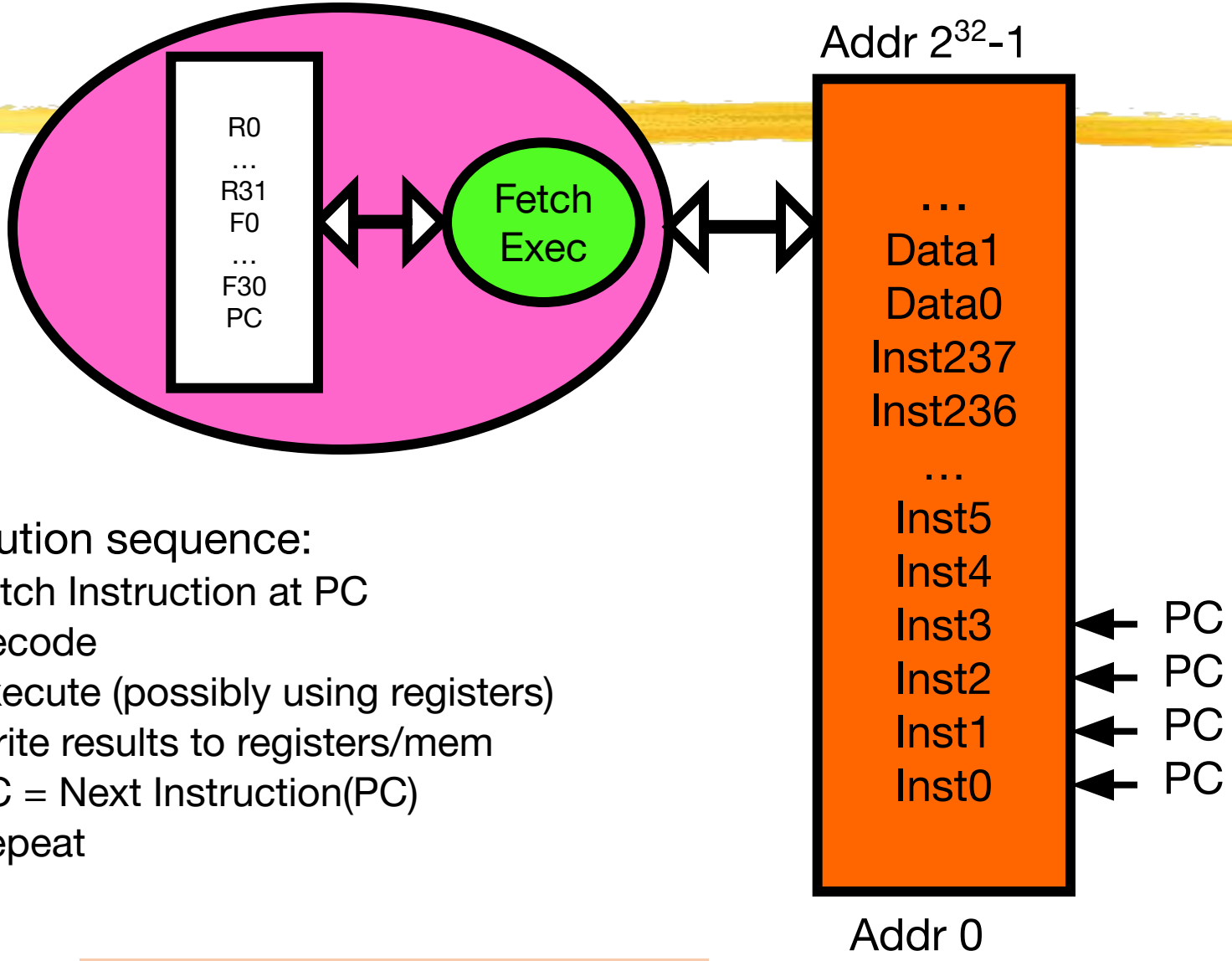
# Computer System & OS Structures

- Computer System Organization
- Operational Flow and hardware protection
- System call and OS services
- Storage architecture
- OS organization
- OS tasks
- Virtual Machines

# Computer System **Organization**



# CPU execution

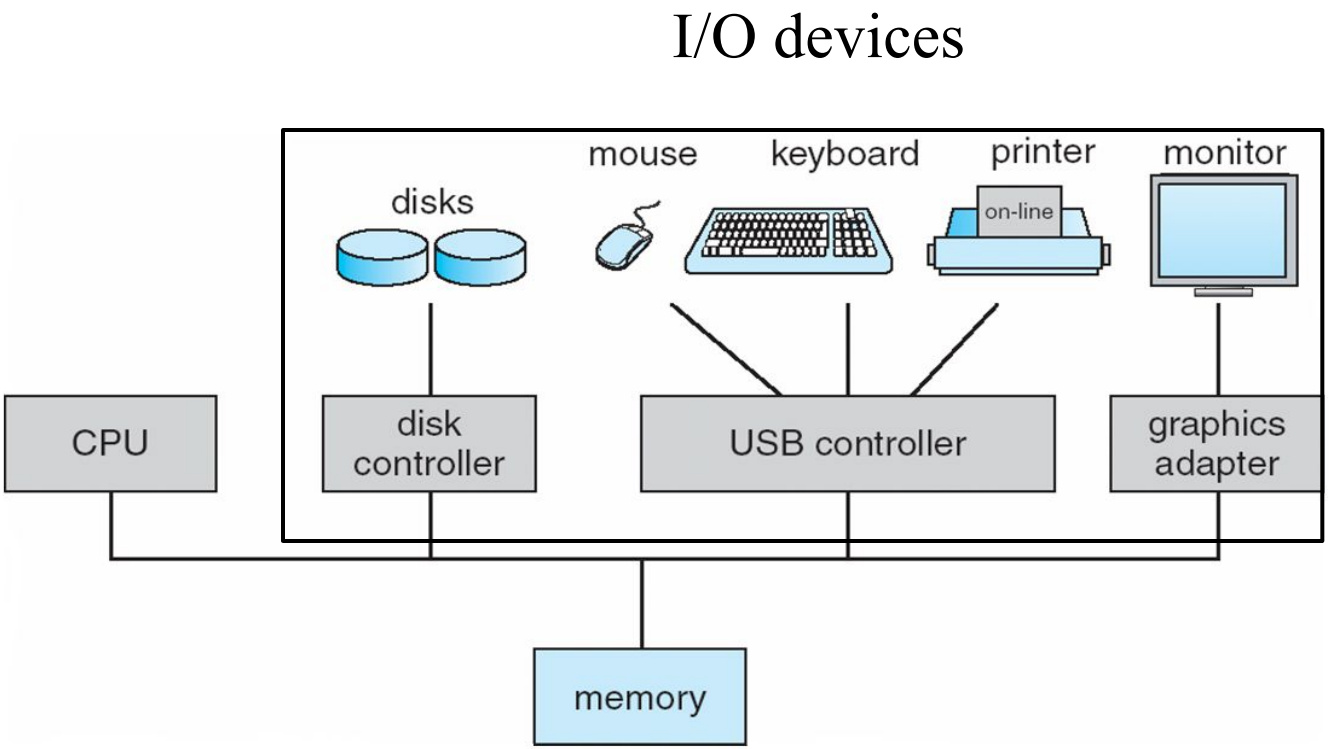



Execution sequence:

- Fetch Instruction at PC
- Decode
- Execute (possibly using registers)
- Write results to registers/mem
- PC = Next Instruction(PC)
- Repeat

***From Berkeley OS course***

# Computer System **Organization**



# I/O devices

- I/O devices and the CPU execute concurrently.
- Each device controller is in charge of a particular device type
  - Each device controller has a local buffer. I/O is from the device to local buffer of controller
- CPU moves data from/to main memory to/from the local buffers

# Interrupts

- **Interrupt** transfers control to the **interrupt service routine**
  - Interrupt Service Routine: Segments of code that determine action to be taken for interrupt.
- Determining the type of interrupt
  - Polling: same interrupt handler called for all interrupts, which then polls all devices to figure out the reason for the interrupt
  - Interrupt Vector Table: different interrupt handlers will be executed for different interrupts

| Interrupt Number | Address |
|---|---|
| 0 | 0003h |
| 1 | 000Bh |
| 2 | 0013h |
| 3 | 001Bh |
| 4 | 0023h |
| 5 | 002Bh |
| 6 | 0033h |
| 7 | 003Bh |
| 8 | 0043h |
| 9 | 004Bh |
| 10 | 0053h |
| 11 | 005Bh |
| 12 | 0063h |
| 13 | 006Bh |
| 14 | 0073h |
| 15 | 007Bh |

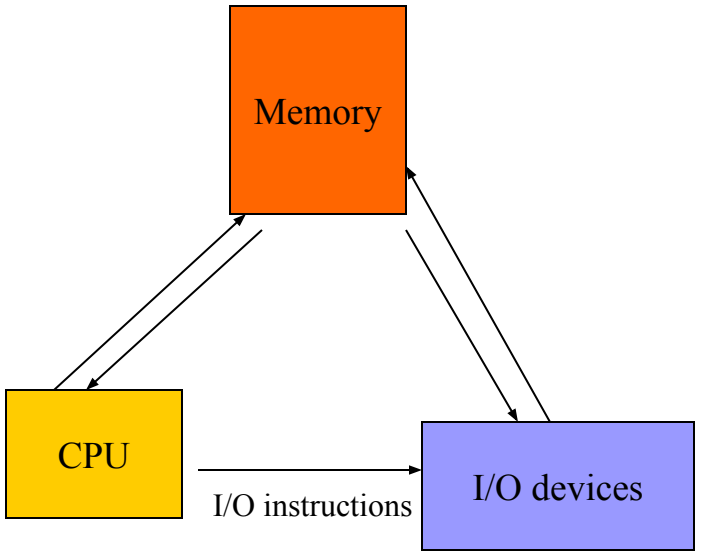
| Interrupt Number | Address |
|---|---|
| 16 | 0083h |
| 17 | 008Bh |
| 18 | 0093h |
| 19 | 009Bh |
| 20 | 00A3h |
| 21 | 00ABh |
| 22 | 00B3h |
| 23 | 00BBh |
| 24 | 00C3h |
| 25 | 00CBh |
| 26 | 00D3h |
| 27 | 00DBh |
| 28 | 00E3h |
| 29 | 00EBh |
| 30 | 00F3h |
| 31 | 00FBh |

# Interrupt **handling**

- OS preserves the state of the CPU
  - stores registers and the program counter (address of interrupted instruction).
- What happens to a new interrupt when the CPU is handling one interrupt?
  - Incoming interrupts can be disabled while another interrupt is being processed. In this case, incoming interrupts may be lost or may be buffered until they can be delivered.
  - Incoming interrupts can be masked (i.e., ignored) by software.
  - Incoming interrupts are delivered, i.e., nested interrupts.

# Direct Memory Access (DMA)

- Typically used for I/O devices with a lot of data to transfer (in order to reduce load on CPU).
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Device controller interrupts CPU on completion of I/O



# Process Abstraction

# Process Abstraction

- Process: an *instance* of a program, running with limited rights

# Process Abstraction and rights

- Process: an *instance* of a program, running with limited rights
- Address space: set of rights of a process
    - Memory that the process can access
- Other permissions the process has (e.g., which system calls it can make, what files it can access)

# Hardware Protection

- CPU Protection:
  - Dual Mode Operation
  - Timer interrupts
- Memory Protection
- I/O Protection

# How to limit process rights?

# Should a process be able to execute any instructions?

# Should a process be able to execute any instructions?

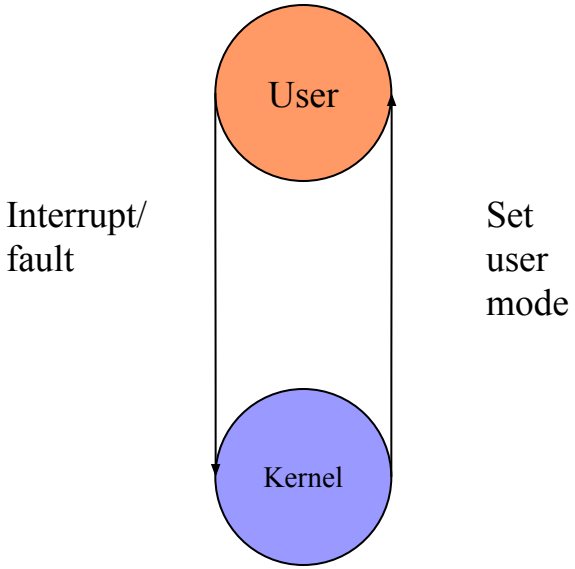
- No
  - Can alter system configuration
  - Can access unauthorized memory
  - Can access unauthorized I/O
  - etc.
- How to prevent?

# Dual-mode operation

- Provide hardware support to differentiate between at least two modes of operation:
  1. User mode -- execution done on behalf of a user.
  2. Kernel mode (monitor/supervisor/system mode) -- execution done on behalf of operating system.
- "Privileged" instructions are only executable in the kernel mode
- Executing privileged instructions in the user mode "traps" into the kernel mode

# Dual-mode operation(cont.)

- Mode bit added to computer hardware to indicate the current mode: kernel(0) or user(1).
- When an interrupt or trap occurs, hardware switches to kernel mode.



# CPU Protection

- How to prevent a process from executing indefinitely?

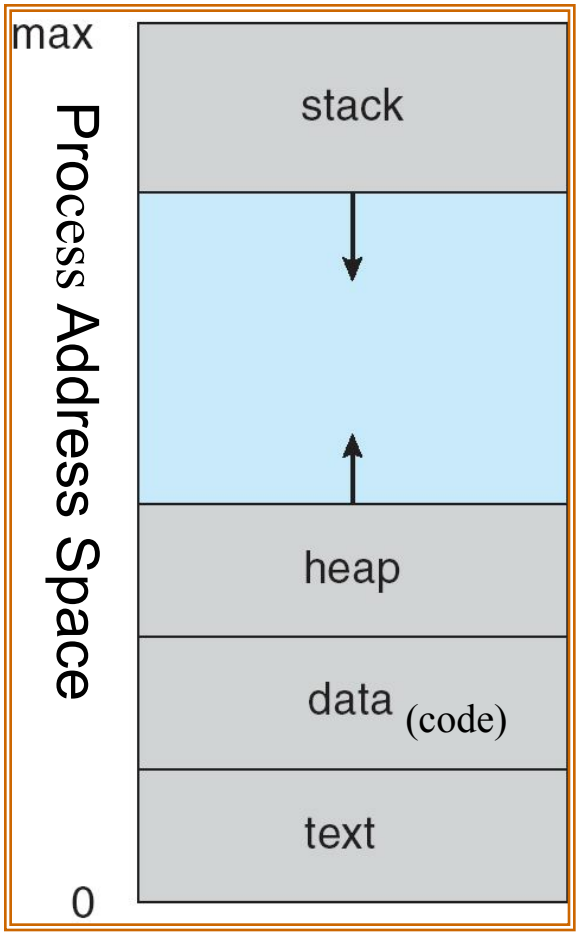
# CPU Protection

- Timer - interrupts computer after specified period to ensure that OS maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches a value of 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
- Timer is also used to compute the current time.
- Programming the timer can only be done in the kernel since it requires privileged instructions.

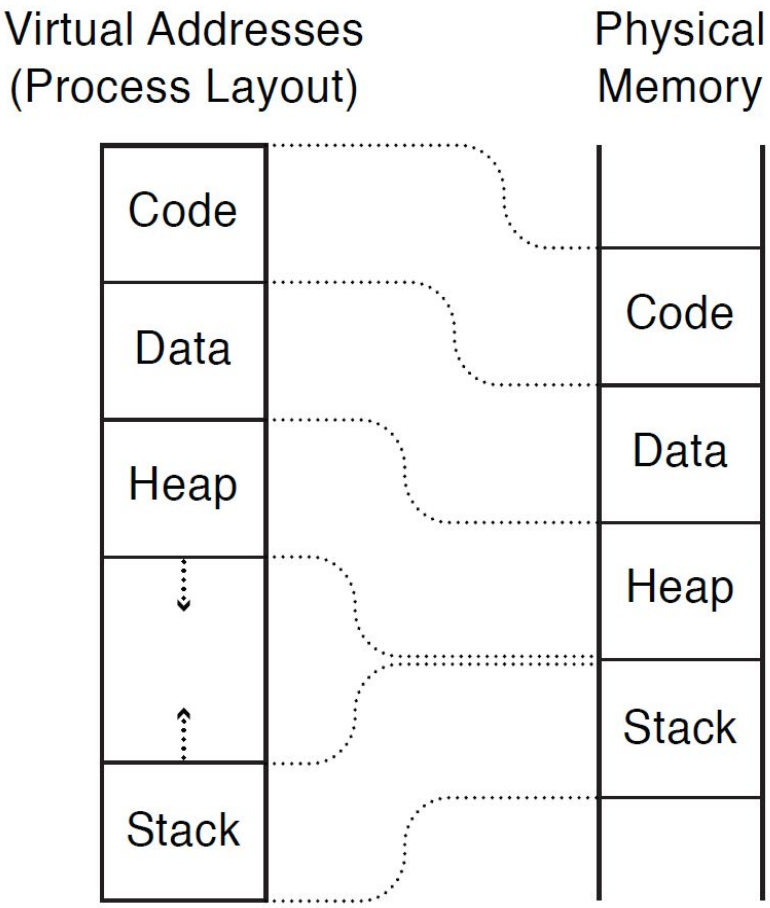
# How to isolate memory access?

# Process address space

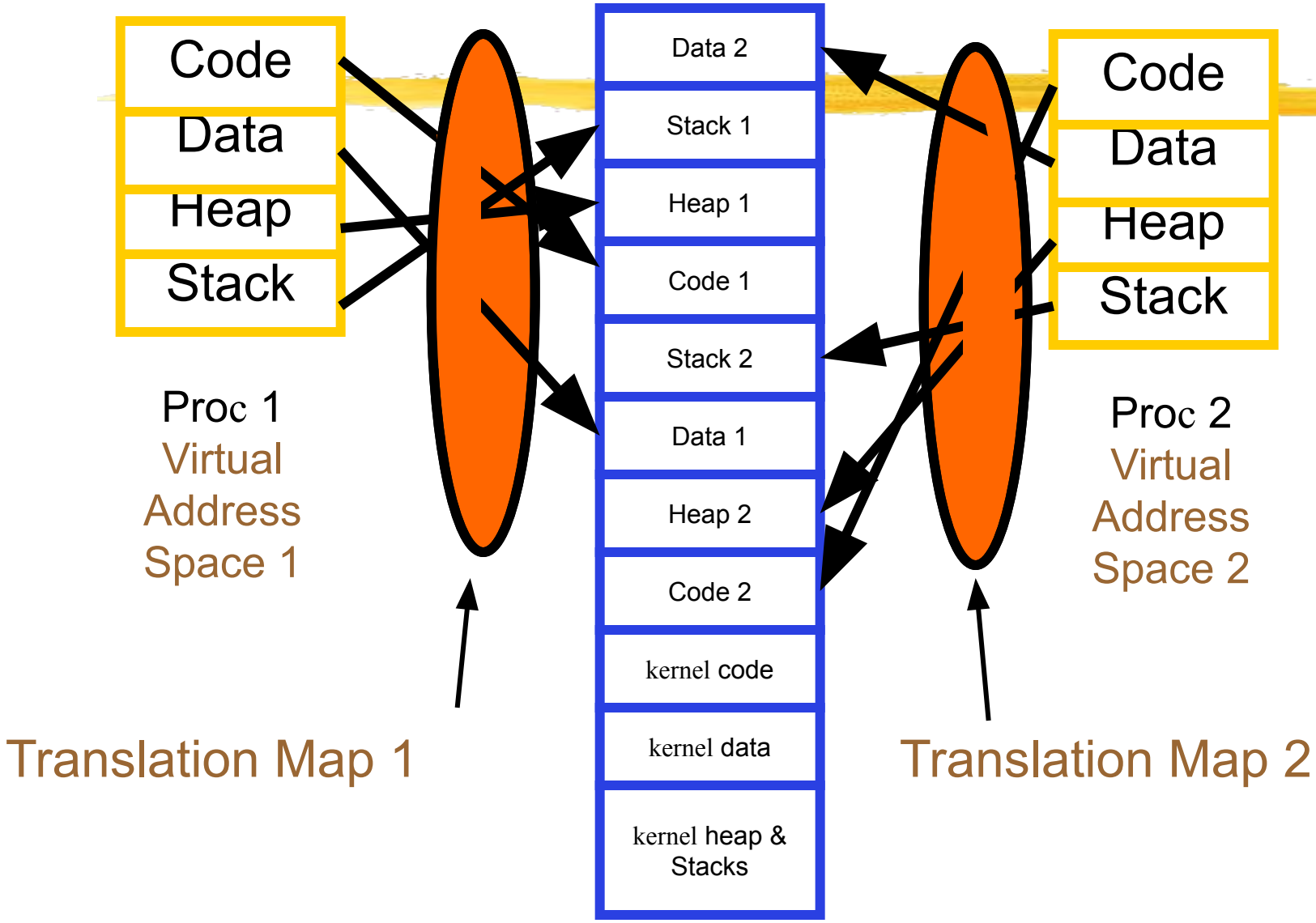
- Address space ⇒ the set of accessible addresses :
  - For a 32-bit processor there are $2^{32}$ = 4 billion addresses



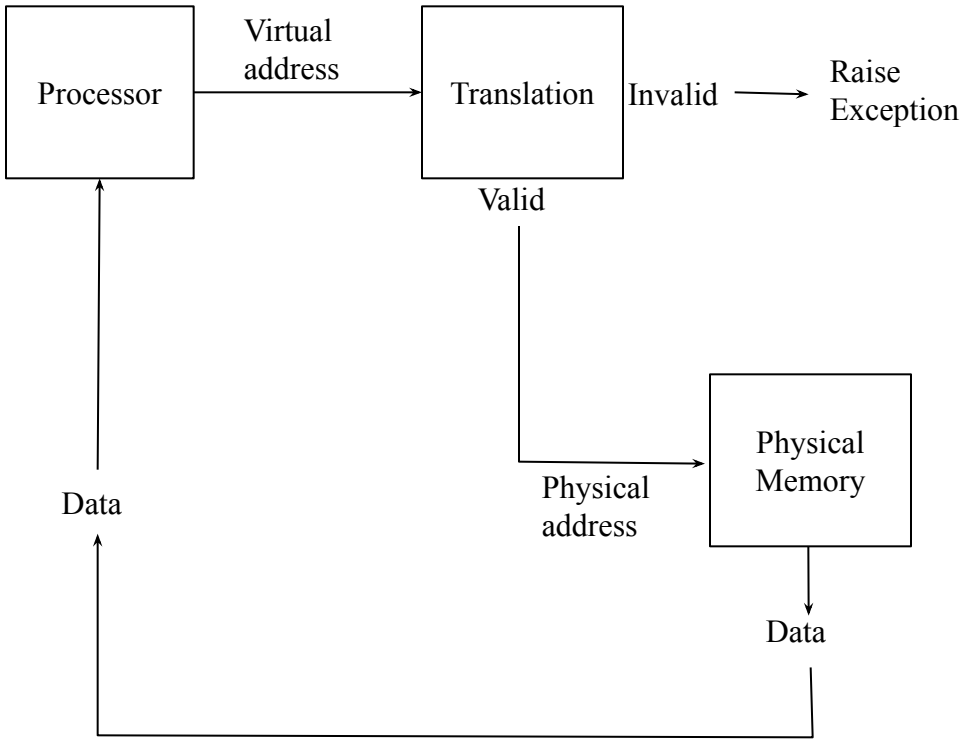
# Virtual **Address**



# Providing the Illusion of Separate Address Spaces



# Address translation and memory protection

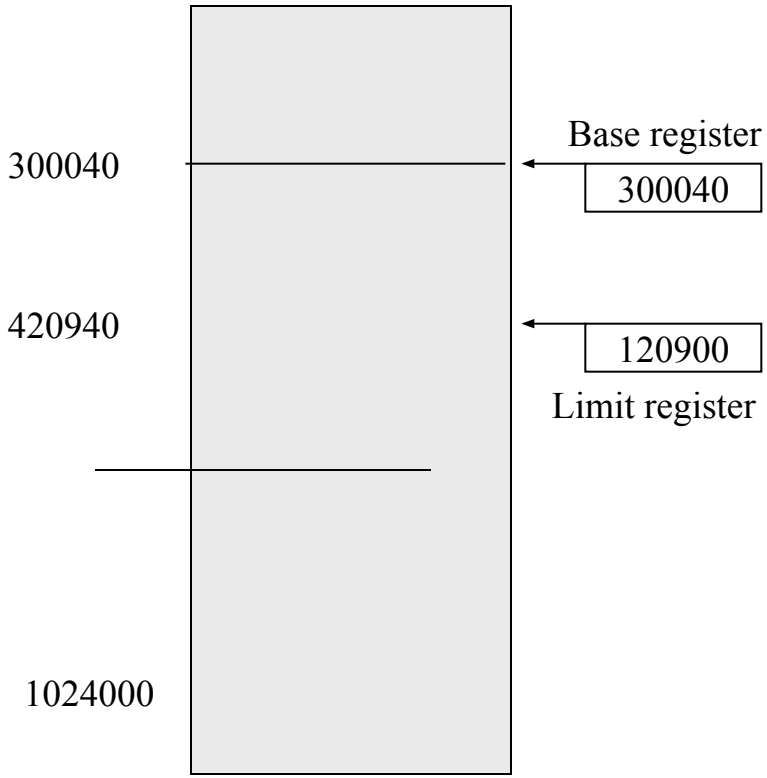


# Memory Protection

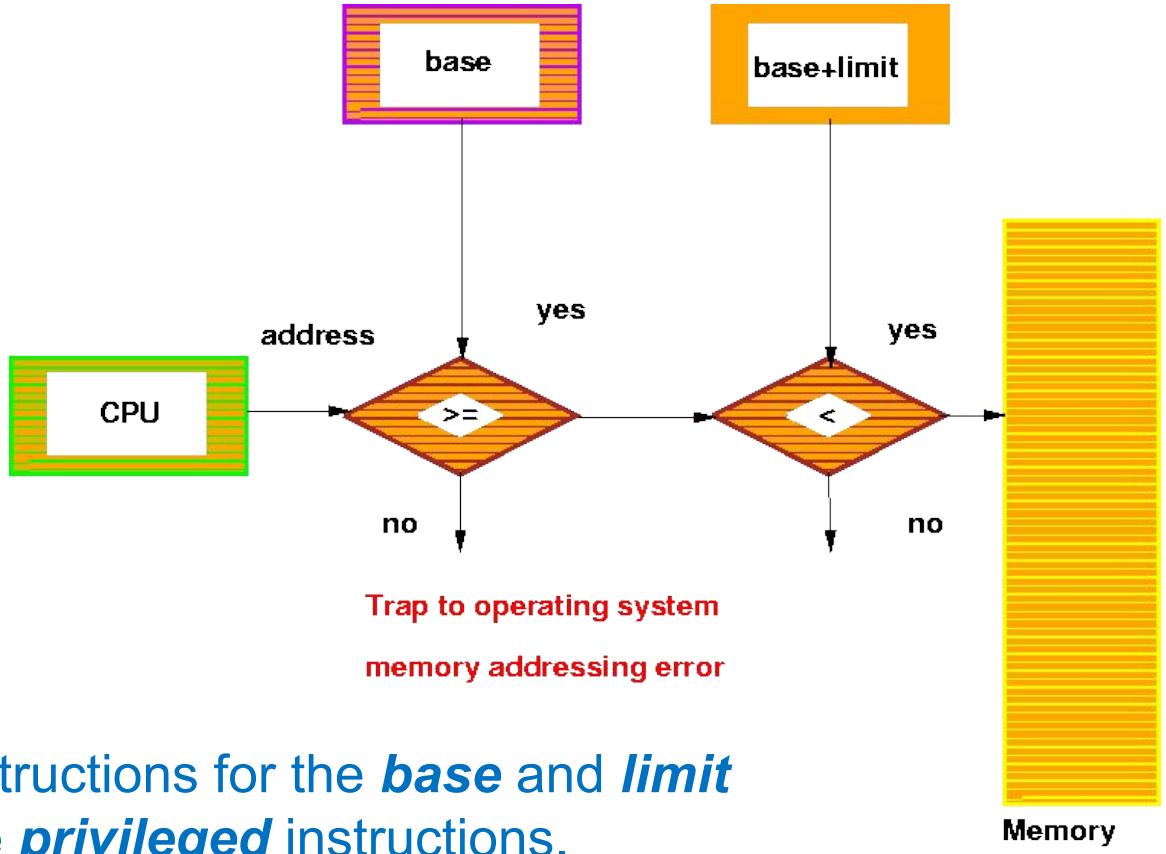
- When a process is running, only memory in that process address space must be accessible.
- When executing in kernel mode, the kernel has unrestricted access to all memory.

# Memory Protection: **base and limit**

- To provide memory protection, add two registers that determine the range of legal addresses a program may address.
  - **Base Register** - holds smallest legal physical memory address.
  - **Limit register** - contains the size of the range.
- Memory outside the defined range is protected.
- Sometimes called ***Base and Bounds*** method

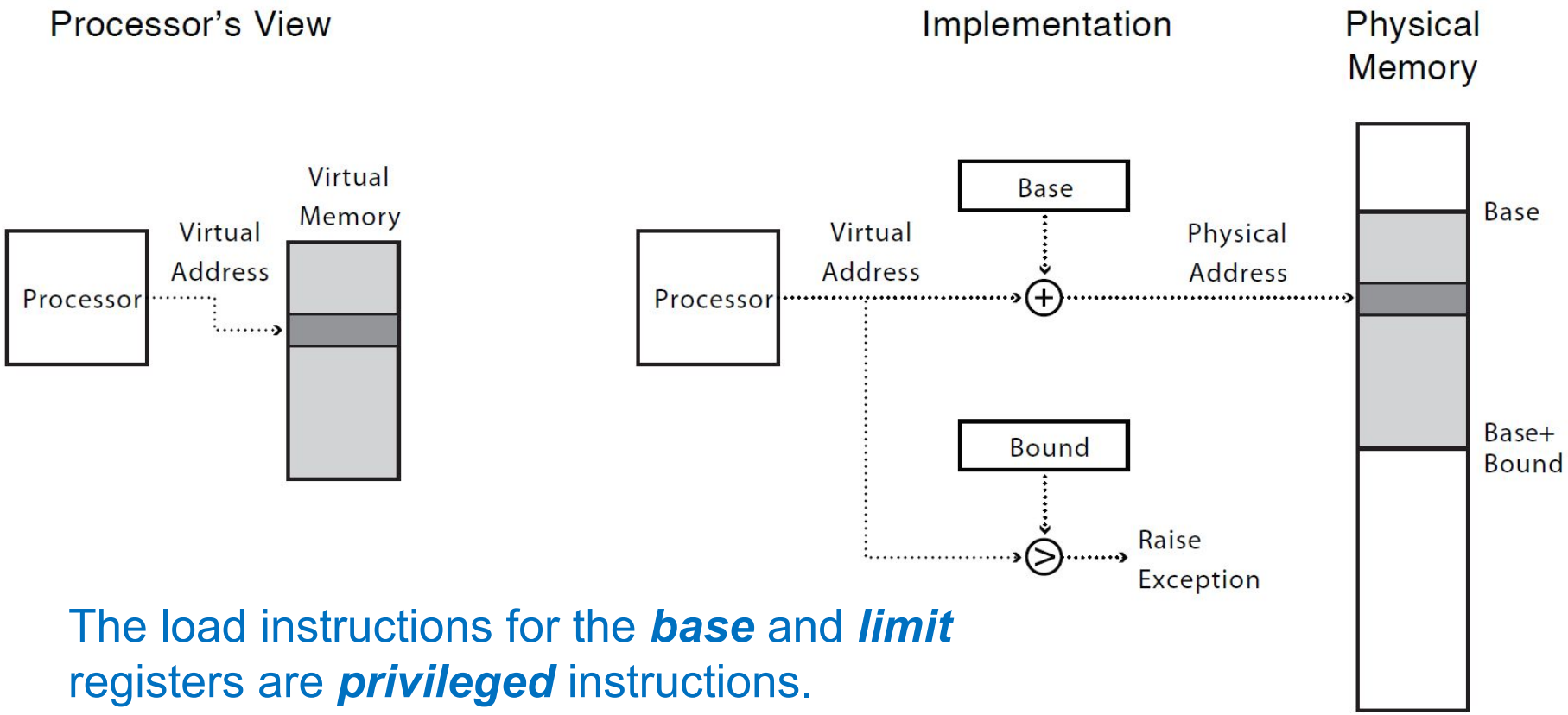


# Hardware Address Protection




The load instructions for the ***base*** and ***limit*** registers are ***privileged*** instructions.

# Virtual Address translation using the Base and Bounds method




The load instructions for the ***base*** and ***limit*** registers are ***privileged*** instructions.

# I/O Protection

- All I/O instructions are privileged instructions.

# Question

- Given the I/O instructions are privileged, how do users perform I/O?

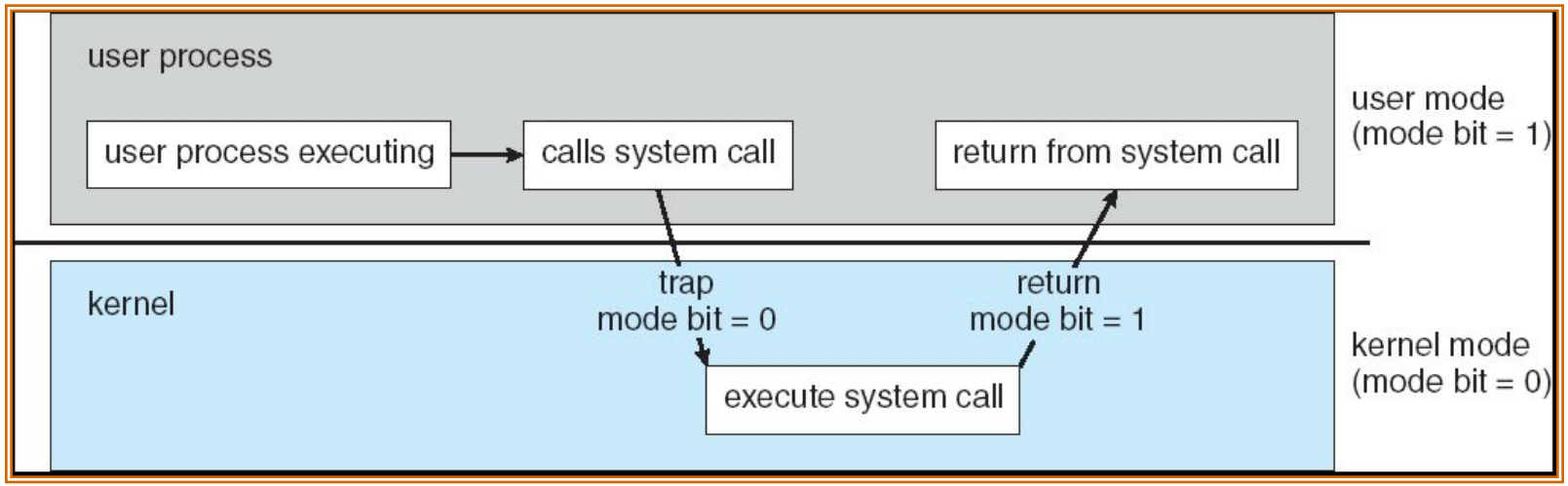
# Question

- Given the I/O instructions are privileged, how do users perform I/O?

- Via system calls - the method used by a process to request action by the operating system.

# System Calls

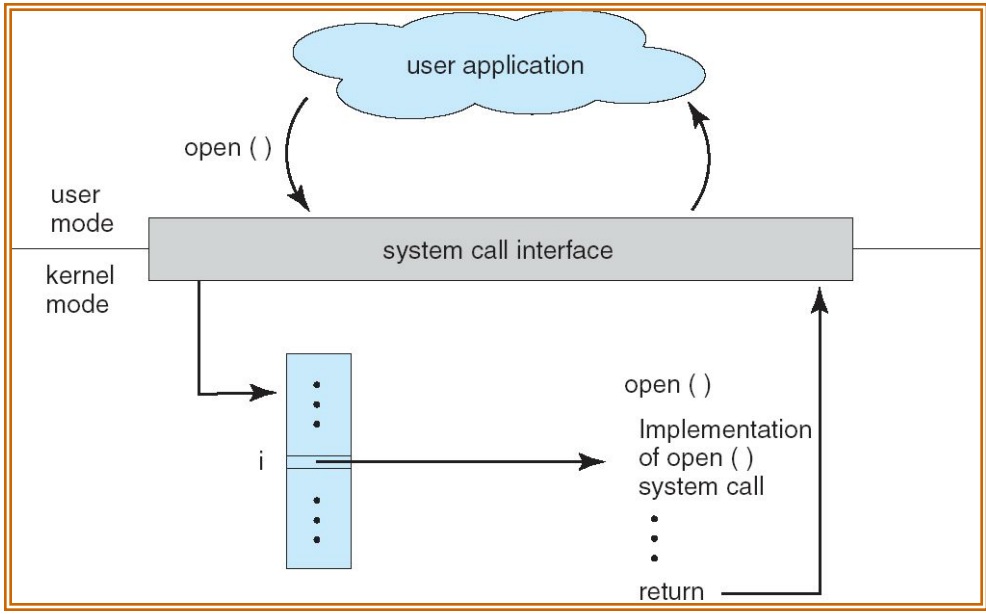
- User code can issue a syscall, which causes a trap
- Kernel handles the syscall



# System Calls

- Interface between applications and the OS.
  - Application uses an assembly instruction to trap into the kernel
  - Some higher level languages provide wrappers for system calls (e.g., C)
- System calls pass parameters between an and OS via registers or memory

  .
- Linux has about 300 system calls
  - read(), write(), open(), close(), fork(), exec(), ioctl(),…..



# System **services or system programs**

- Convenient environment for program development and execution.
  - Command Interpreter (i.e., shell) - parses/executes other system programs
  - Window management
  - System libraries, e.g., libc

# Command Interpreter System

- Commands that are given to the operating system via command statements that execute
  - Process creation and deletion, I/O handling, secondary storage management, main memory Management, file system access, protection, networking, etc.
- Obtains the next command and executes it.
- Programs that read and interpret control statements also called -
  - Command-line interpreter, shell (in UNIX)

# Storage Structure

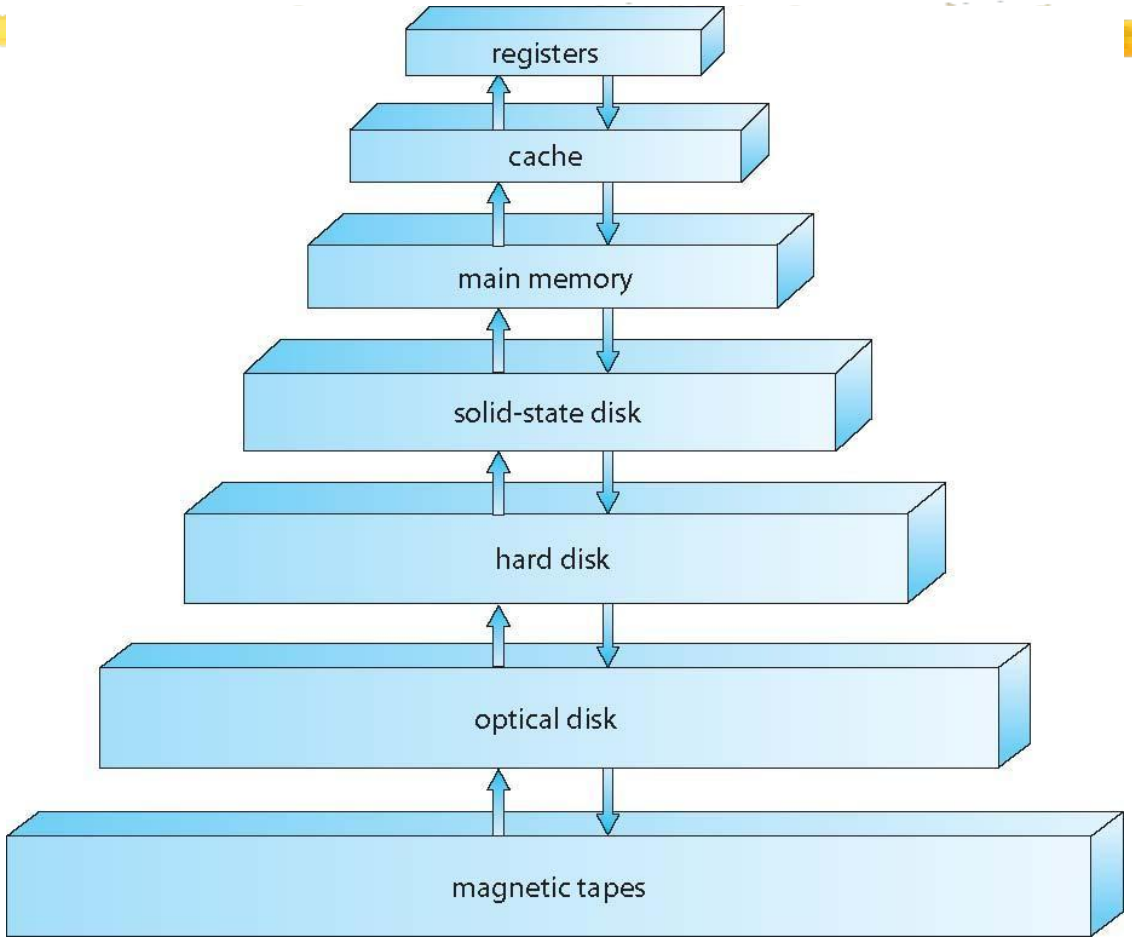
- Main memory - only large storage media that the CPU can access directly.
- Secondary storage - has large nonvolatile storage capacity.
  - Magnetic disks - rigid metal or glass platters covered with magnetic recording material.
    - Disk surface is logically divided into tracks, subdivided into sectors.
    - Disk controller determines logical interaction between device and computer.

# Storage Hierarchy

- Storage systems are organized in a hierarchy based on
  - Speed
  - Cost
  - Volatility

- Caching - process of copying information into faster storage system; main memory can be viewed as fast cache for secondary storage.

# Storage Device Hierarchy



# OS Task: Process Management

- Process - fundamental concept in OS
  - Process is an instance of a program in execution.
  - Process needs resources - CPU time, memory, files/data and I/O devices.
- OS is responsible for the following process management activities.
  - Process creation and deletion
  - Process suspension and resumption
  - Process synchronization and interprocess communication
  - Process interactions - deadlock detection, avoidance and correction

# OS Task: Memory Management

- Main Memory is an array of addressable words or bytes that is quickly accessible.
- Main Memory is volatile.
- OS is responsible for:
  - Allocate and deallocate memory to processes.
  - Managing multiple processes within memory - keep track of which parts of memory are used by which processes. Manage the sharing of memory between processes.
  - Determining which processes to load when memory becomes available.

# OS Task: Secondary Storage and I/O Management

- Since primary storage (i.e., main memory) is expensive and volatile, secondary storage is required for backup.
- Disk is the primary form of secondary storage.
  - OS performs storage allocation, free-space management, etc.
- I/O system in the OS consists of
  - Device driver interface that abstracts device details
  - Drivers for specific hardware devices

# OS Task: File System Management

- File is a collection of related information - represents programs and data.

- OS is responsible for
  - File creation and deletion
  - Directory creation and deletion
  - Supporting primitives for file/directory manipulation.
  - Mapping files to disks (secondary storage).

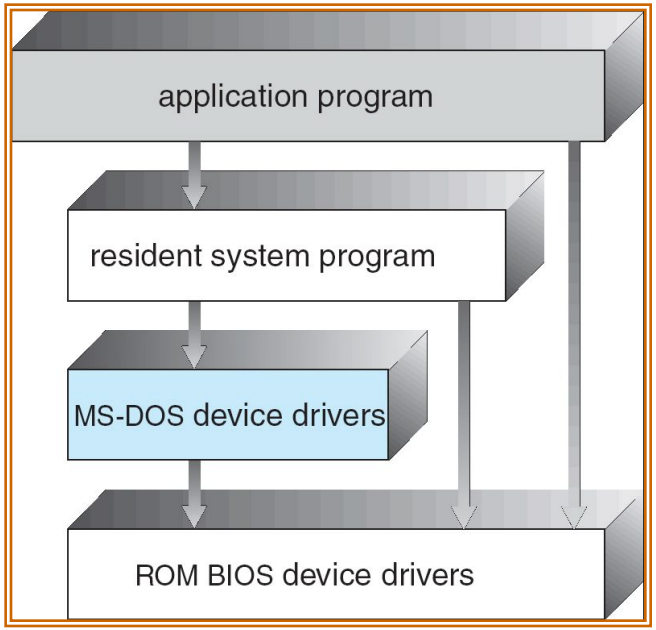
# OS Task: Protection and Security

- Protection mechanisms control access of processes to user and system resources.
- Protection mechanisms must:
  - Distinguish between authorized and unauthorized use.
  - Specify access controls to be imposed on use.
  - Provide mechanisms for enforcement of access control.

# Operating Systems: How are they organized?

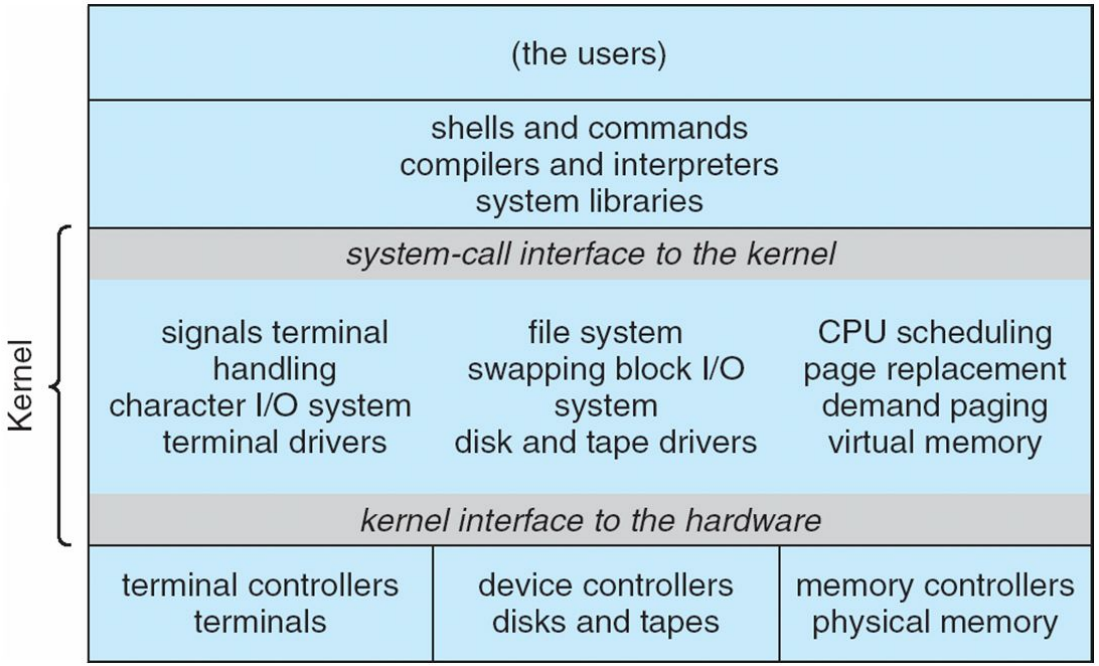
# OS Structure - Simple Approach

- MS-DOS - provides a lot of functionality in little space.
  - Not divided into modules, Interfaces and levels of functionality are not well separated



# **Original** UNIX System Structure

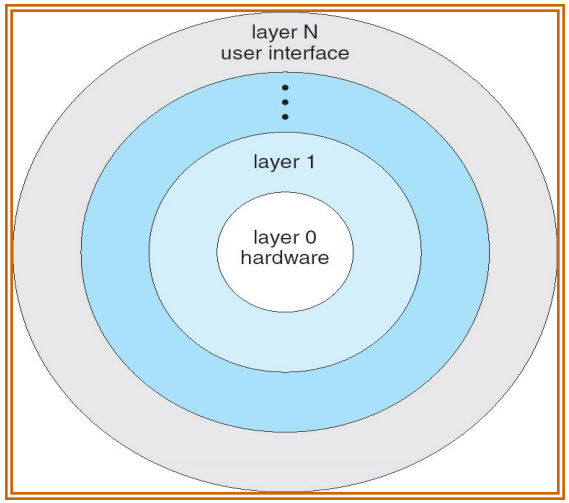
- Limited structuring, has 2 separable parts
  - Systems programs
  - Kernel
    - everything below system call interface and above physical hardware.
    - Filesystem, CPU scheduling, memory management



# Layered OS Structure

- OS divided into number of layers - bottom layer is hardware, highest layer is the user interface.
- Each layer uses functions and services of only lower-level layers.
- THE Operating System and Linux Kernel has successive layers of abstraction.

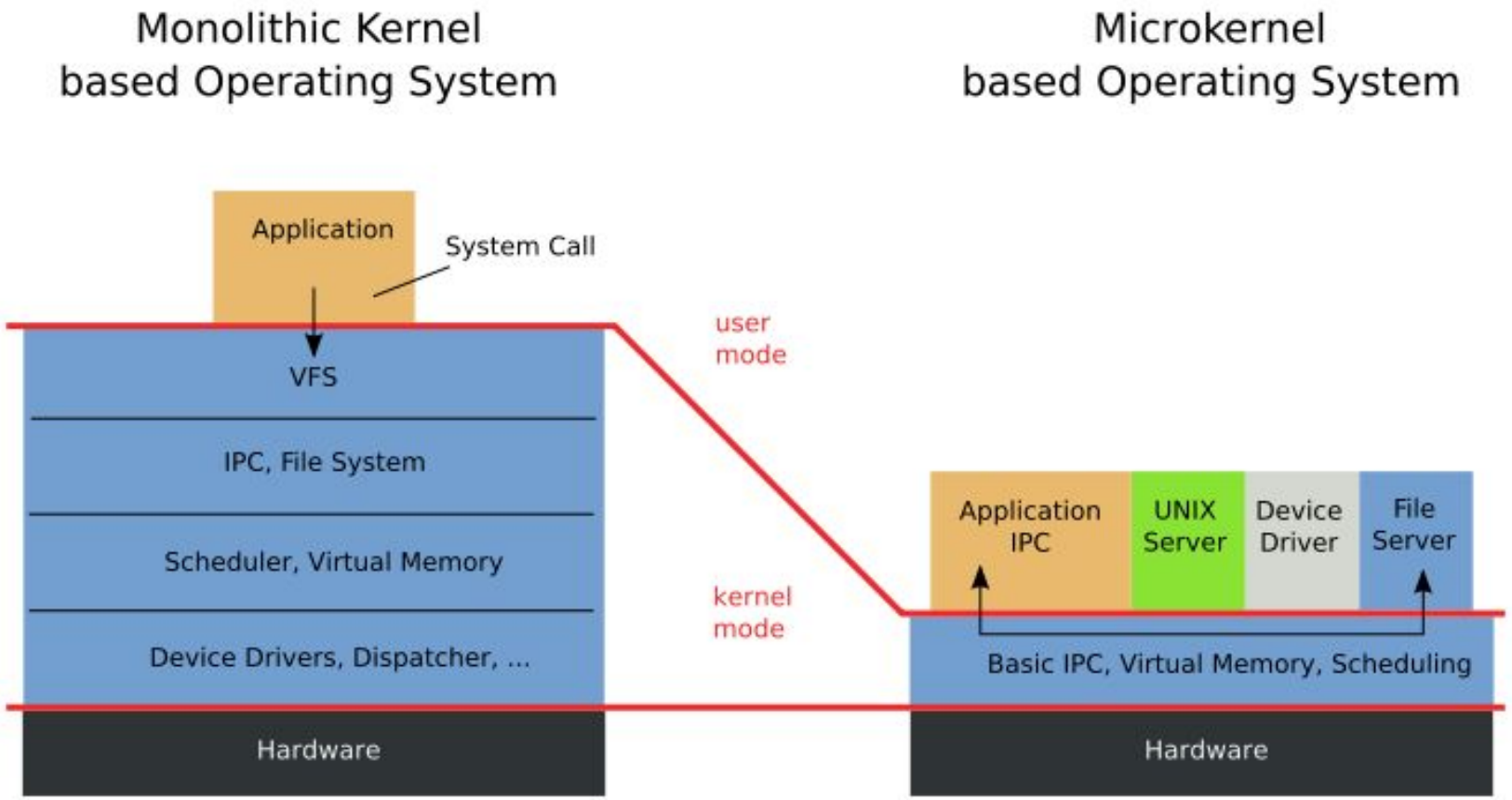
| User Programs |
| --- |
| Interface Primitives |
| Device Drivers and Schedulers |
| Virtual Memory |
| I/O |
| CPU Scheduling |
| Hardware |



# **Monolithic vs.** Microkernel **OS**

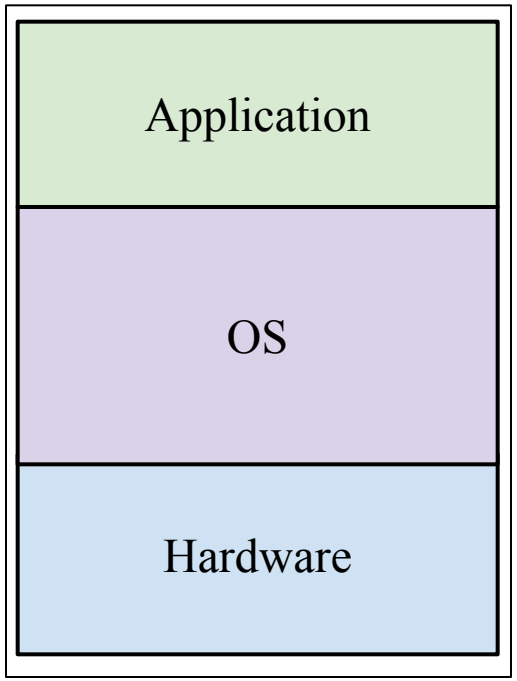
- Monolithic OSes have large kernels with a lot of components
  - Linux, Windows, Mac
- Microkernels moves as much from the kernel into "*user*" space
  - Small core OS components running at kernel level
  - OS Services built from many independent user-level processes
- Communication between modules with message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port OS to new architectures
  - More reliable and more secure (less code is running in kernel mode)
- Detriments:
  - Performance overhead severe for naïve implementation

# A microkernel OS

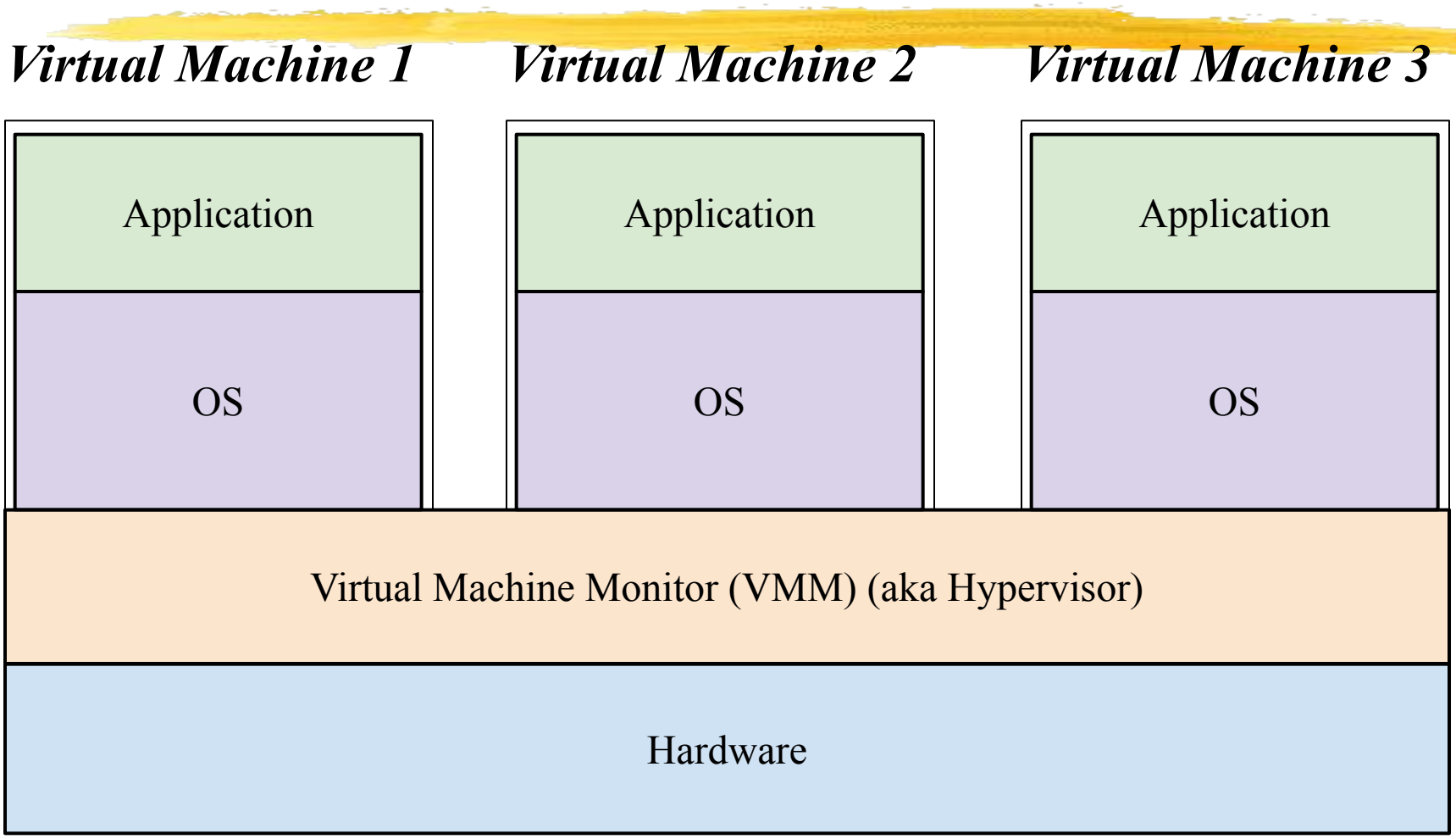



Slide adapted from http://web.cecs.pdx.edu/~walpole/class/cs533/fall2015/home.html

# Virtual Machines



# Virtual Machines



# Virtual Machines

- Use cases
  - Resource configuration
  - Running multiple OSes, either the same or different OSes
  - Run existing OS binaries on different architecture

# Summary of Lecture set 1

- What is an operating system?
- Operating systems history
- Computer system and operating system structure