CS3510: OS-I

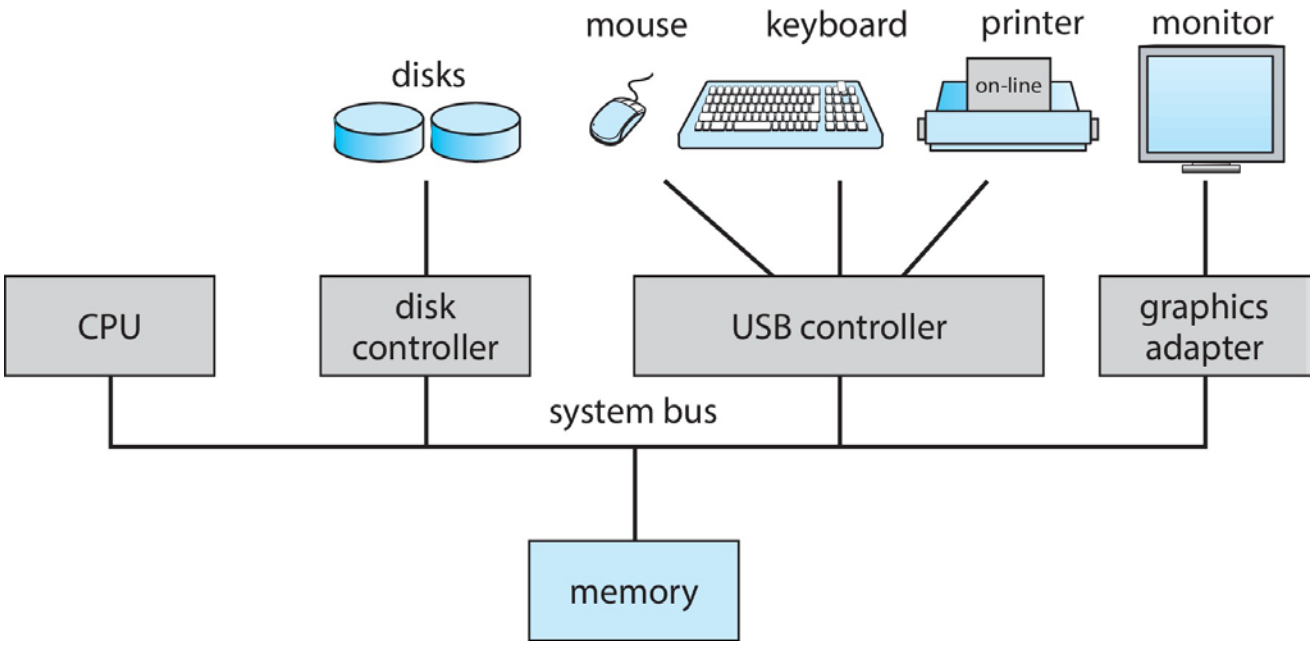
Bheemarjuna Reddy Tamma
IIT HYD

Goals for Week-1

- Computer System Structure
- What is an Operating System?
- Examples of Operating Systems Design
- Why study Operating Systems?
- Different OS Structures?

Note: Some slides and/or pictures in the following are adapted from slides text books on OS by Silberschatz, Galvin, and Gagne AND Andrew S. Tanenbaum and Albert S.Woodhull. Slides courtesy of Kubiatowicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, and David Wagner.

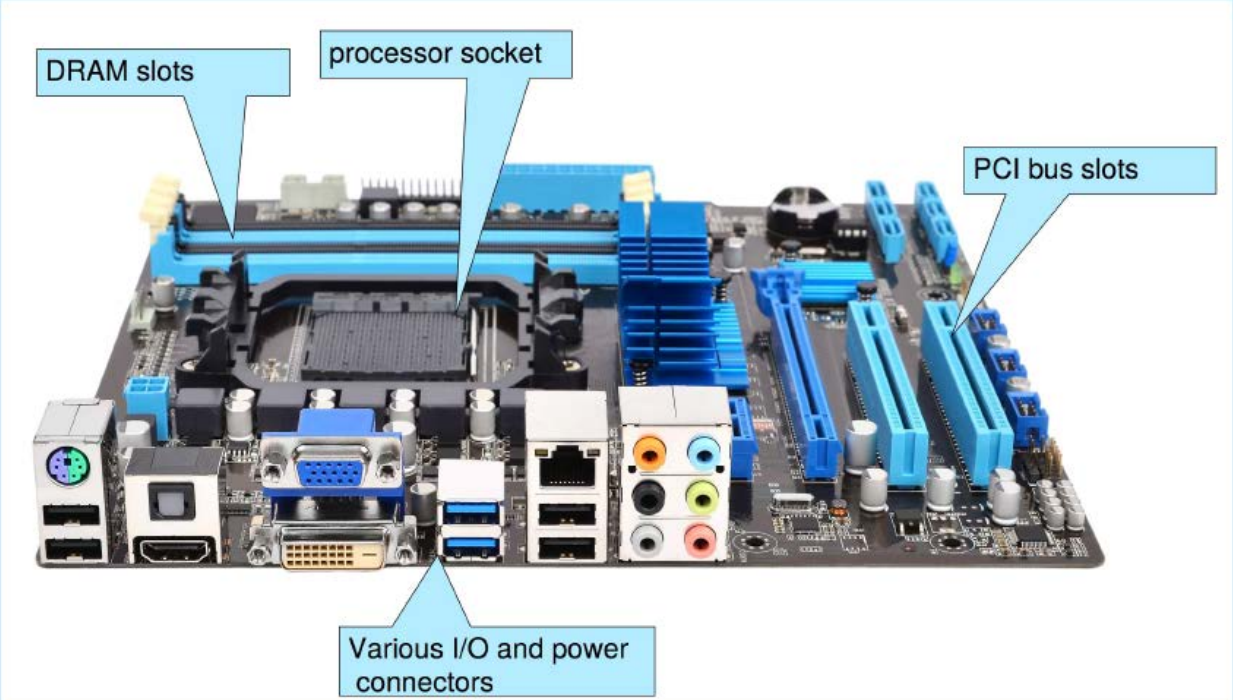
Computer System Organization (Hardware View)



An example of a single processor system

PC Motherboard

Consider the desktop PC motherboard with a processor socket shown below:

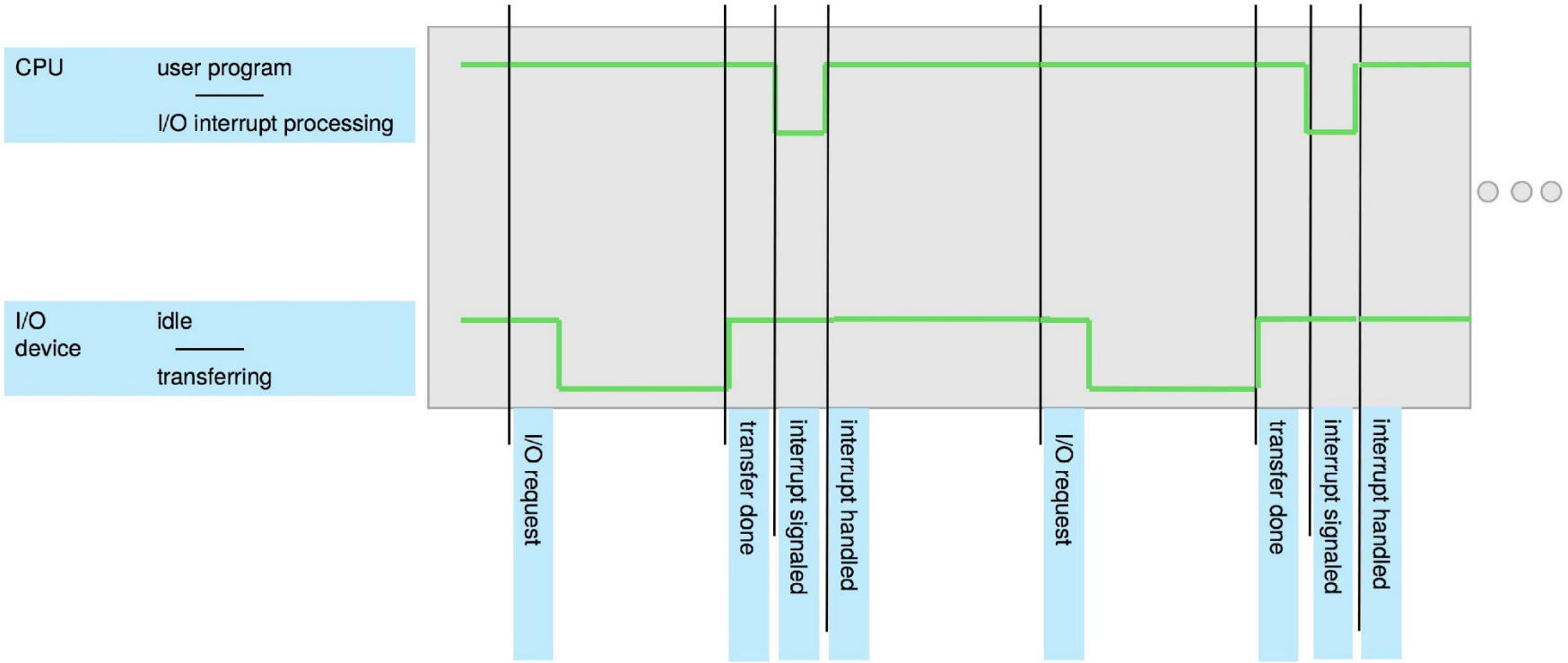


This board is a fully-functioning computer, once its slots are populated. It consists of a processor socket containing a CPU, DRAM sockets, PCIe bus slots, and I/O connectors of various types. Even the lowest-cost general-purpose CPU contains multiple cores. Some motherboards contain multiple processor sockets. More advanced computers allow more than one system board, creating NUMA systems.

Computer System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system device driver to manage it
- CPU moves data from/to main memory to/from local buffers of I/O devices
- I/O is from the device to local buffer of device controller
- Device controller informs CPU that it has finished its operation by causing a (hardware) interrupt

Interrupt Timeline

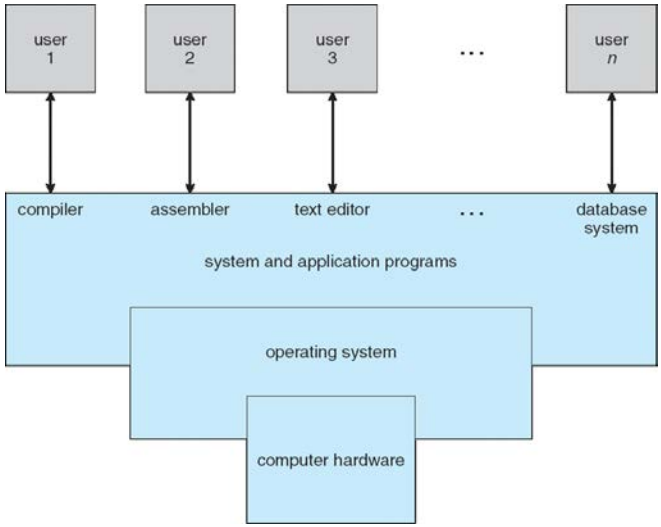


Interrupts

- Modern OSs are interrupt driven
  - Hardware interrupts
  - Software interrupts
- Hardware interrupt by one of I/O devices
  - Keyboard, mouse, printer, etc
- Software interrupt (called as exception or trap)
  - Software error (e.g., division by zero)
  - Request for OS services by users
  - Other process problems include infinite loop, processes modifying each other or the operating system
- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction

Computer System Structure

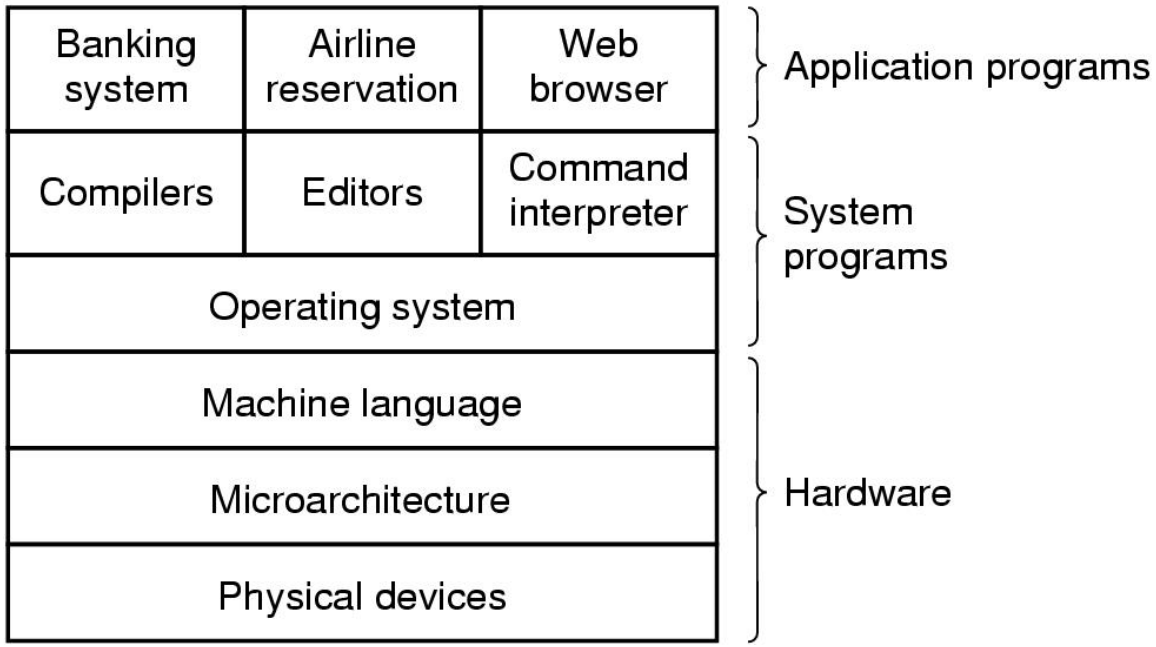
- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - » CPU, memory, I/O devices
  - Operating system
    - » Controls & coordinates use of hardware among various applications and users
  - System & Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - » Word processors, compilers, web browsers, database systems, video games
  - Users
    - » People, machines, other computers



Computer System Structure

A computer system consists of

- hardware
- system programs
- application programs

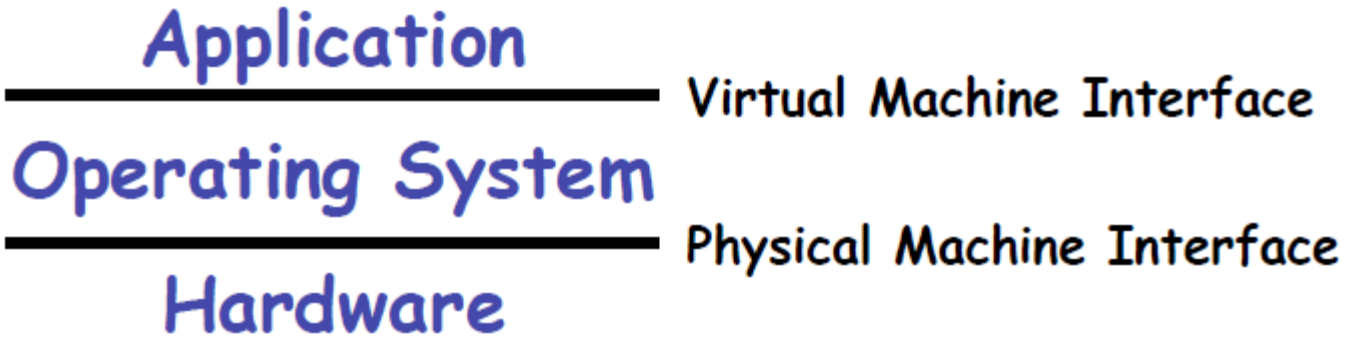


What is OS?

- A program that acts as an intermediary between a user of a computer and the computer hardware

- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

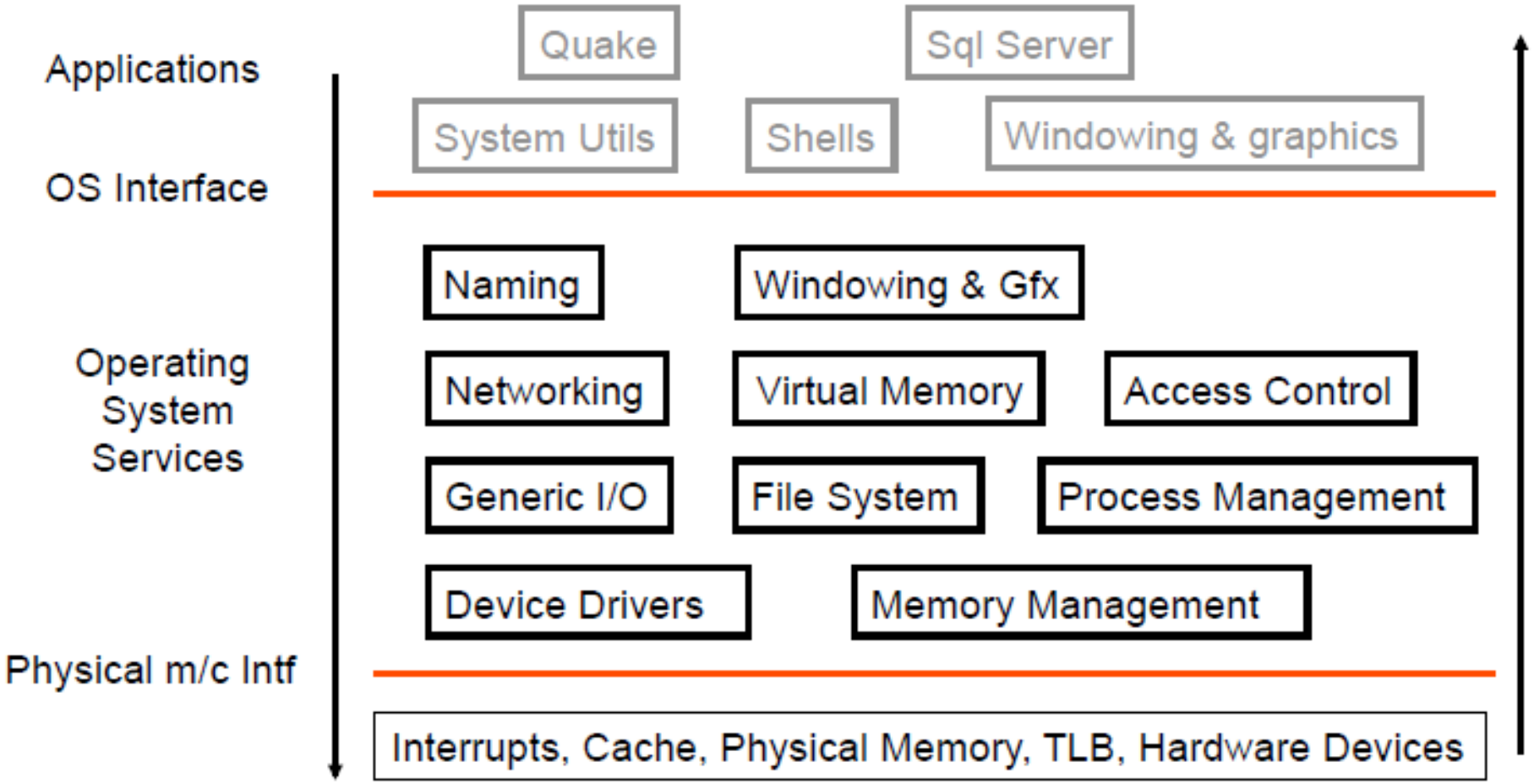
What is OS?



- What is the physical (hardware) interface?
  - Physical reality
- What is the virtual (application) interface?
  - Nicer abstraction hiding all complexities

- A Virtual Machine Abstraction
  - An operating system implements a virtual machine that is (hopefully) easier and safer to program and use than the raw hardware

Logical OS Structure



What is OS?

- It is an extended machine
  - Hides the messy details which must be performed
  - Presents user with a virtual machine, easier to use

- It is a resource manager
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
  - Each program gets time with the resource
  - Each program gets space on the resource

- It is a control program
  - Controls execution of programs to prevent errors and improper use of the computer

What does an Operating System do?

- Depends on the point of view
- Users want convenience, ease of use and good performance
  - Don't care about resource utilization
- But shared computer such as mainframe or minicomputer must keep all users happy
- Users of dedicate systems such as workstations have dedicated resources but frequently use shared resources from servers
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

What does an Operating System do?

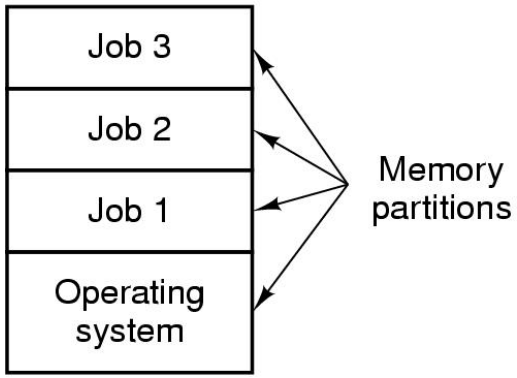
- Coordinator and Traffic Cop:
  - Manages all resources
  - Settles conflicting requests for resources
  - Prevent errors and improper use of the computer
- Facilitator:
  - Provides facilities that everyone needs
  - Standard Libraries, Windowing systems
  - Make application programming easier, faster, less error-prone
- Some features reflect both tasks:
  - E.g. File system is needed by everyone (Facilitator)
  - But File system must be Protected (Traffic Cop)

OS Systems Principles

- OS as illusionist:
  - Make hardware limitations go away
  - Provide illusion of dedicated machine with infinite memory and infinite processors
- OS as government:
  - Protect users from each other
  - Allocate resources efficiently and fairly
- OS as complex system:
  - Constant tension between simplicity and functionality or performance
- OS as history teacher
  - Learn from past
  - Adapt as hardware tradeoffs change

Multiprogramming Systems

- Non-multiprogrammed system: CPU sits idle if job that is scheduled for execution is doing I/O operations (e.g., writing to a file, waiting for input from keyboard)
- Multiprogramming system keeps multiple jobs in main memory simultaneously
- Job scheduler: brings a subset of jobs stored on 2nd Memory (e.g., HDD) into Main memory for execution
- It increases CPU utilization by having at least one job to execute on CPU
  - If current job goes for I/O, picks one of other (waiting) jobs in Main memory for execution
- But, does not strive for providing user interaction with computer

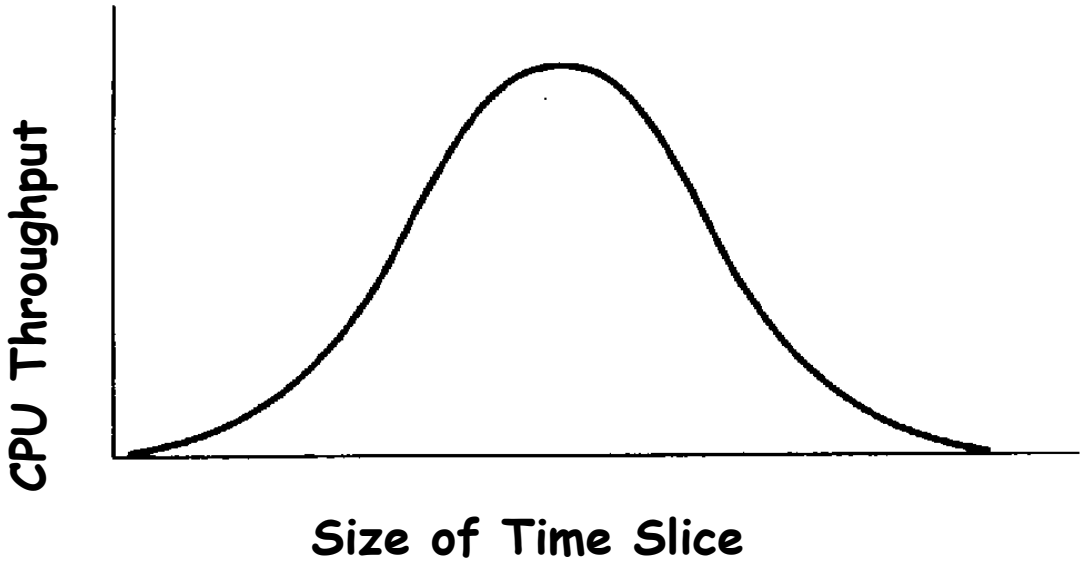


Multitasking (Time sharing) systems

- Logical extension of multiprogramming to provide user interaction with computer
- CPU executes multiple jobs alternatively by switching from one to other, but switching is so frequent to provide interactive computing
- It allows several users/jobs to share computer simultaneously by giving time slice/quantum to each job
- CPU scheduler
  - decides which job to execute in next time slice (order of milliseconds)
  - Takes out control from current job once time slice expires
- Though improves response time (time taken to start executing the job) to users, it may decrease effective CPU utilization compared to Multiprogramming systems!
  - Why?
  - Many Context switches

Multiprogramming vs Multitasking

- CPU throughput and Response time: Complementary or conflicting objectives ??
- Note that high CPU utilization may not translate to high CPU throughput (no. of jobs executed in a time interval, say in a day) due to context switching overhead in multitasking systems
- So, time slice needs to be selected judiciously
  - Neither too small
  - Nor too large



What is an Operating System,… Really?

- Most Likely:
  - CPU Scheduling
  - Memory Management
  - I/O Management
  - Communications? (Does Email belong in OS?)
  - Multitasking/multiprogramming?
- What about?
  - File System (FAT32, ext2)?
  - Multimedia player?
  - User Interface (shell, GUI)?
  - Internet Browser (IE, Chrome, Safari)? ☺

Operating System Definition (Cont.)

No universally accepted definition!

- A control program that manages Computer H/W
- It's intermediary b/w User and Computer H/W
- The set of basic programs and utilities that make your computer run
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the kernel. Everything else is
  - either a system program like Command Prompt/GUI (ships with OS, but not part of Kernel) or
  - an application program which is not related to OS like Web Browser and Word processor

What if we didn't have an Operating System?

- Source Code⇒Compiler⇒Object Code⇒Hardware
- How do you get object code onto the hardware?
- How do you print out the answer?
- Once upon a time, had to Toggle in program in binary and read out answer from LEDs!

Altair 8800

Simple OS: What if only one application to run ?

- Examples:
  - Very early computers
  - Early PCs
  - Embedded controllers (elevators, cars, etc)
- OS becomes just a library of standard services
  - Standard device drivers
  - Interrupt handlers
  - Math libraries
- What about cellphones, Xboxes, etc?
  - Is this organization enough?
  - What about an Android phone or iPhone?

More complex OS: Multiple Apps

- Full Coordination and Protection
  - Manage interactions between different users/jobs
  - Multiple programs running simultaneously
  - Multiplex and protect Hardware Resources
    - » CPU, Memory, I/O devices like disks, printers, etc
- Facilitator
  - Still provides Standard libraries, facilities

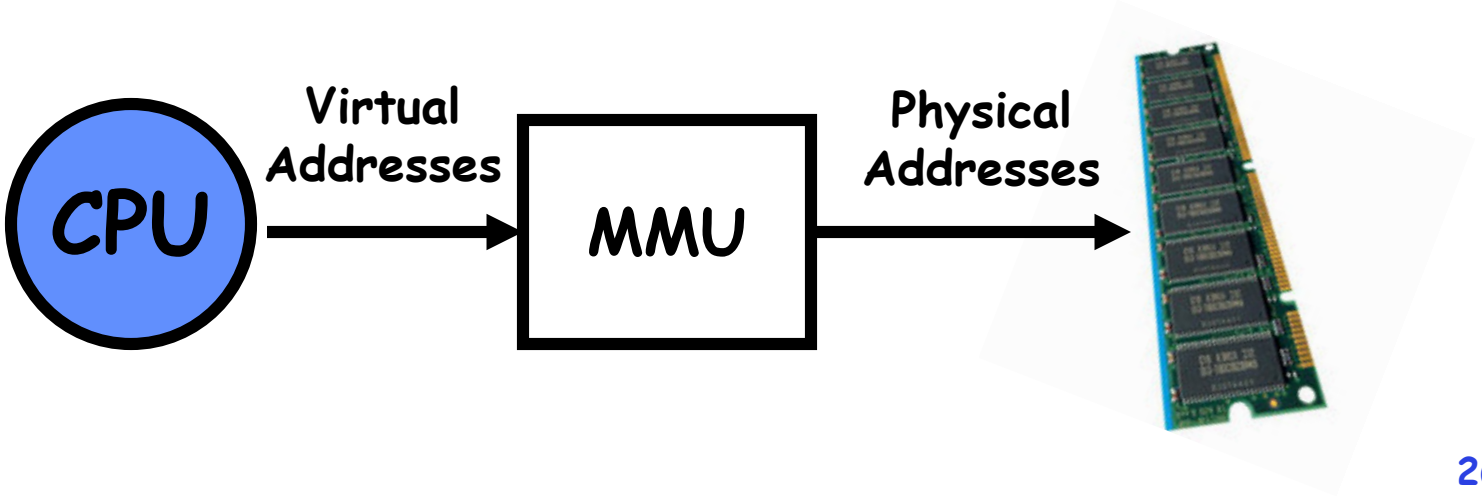
- Would this complexity make sense if there was only one application that you cared about?

Protecting Programs from Each Other

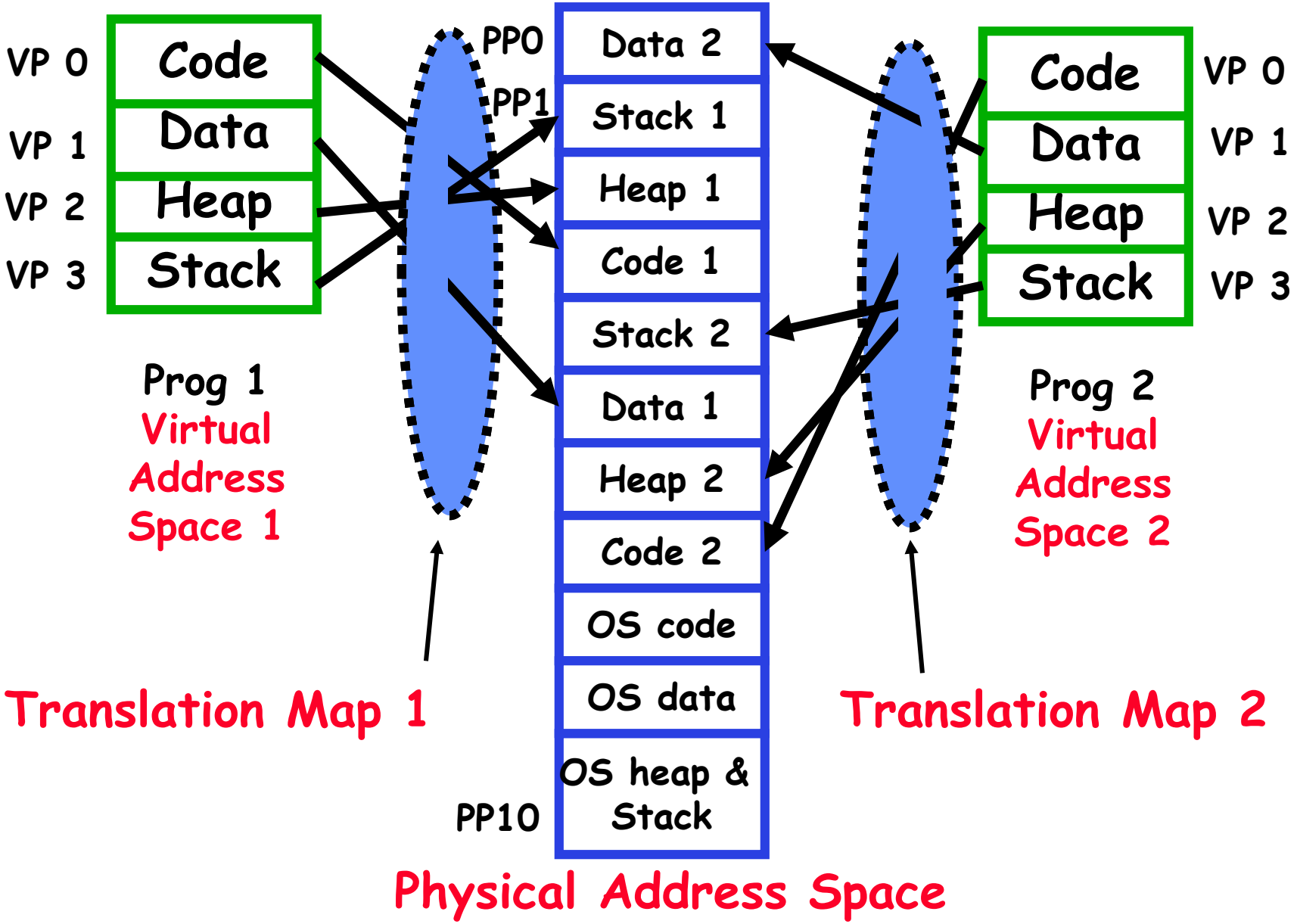
- Problem: Run multiple applications in such a way that they are protected from one another
- Goal:
  - Keep User Programs from Crashing OS
  - Keep User Programs from Crashing each other
  - [Keep Parts of OS from crashing other parts?]
- (Some of the required) Mechanisms:
  - Address Translation
  - Dual Mode Operation
- A simple policy:
  - Programs are not allowed to read/write memory of other Programs or of Operating System

Address Translation

- Address Space
  - A group of memory addresses usable by some job
  - Each program (process) and kernel has potentially different address spaces
- Address Translation:
  - Translate from Virtual Addresses (emitted by CPU) into Physical Addresses (of memory)
  - Mapping *often* performed in Hardware by Memory Management Unit (MMU)

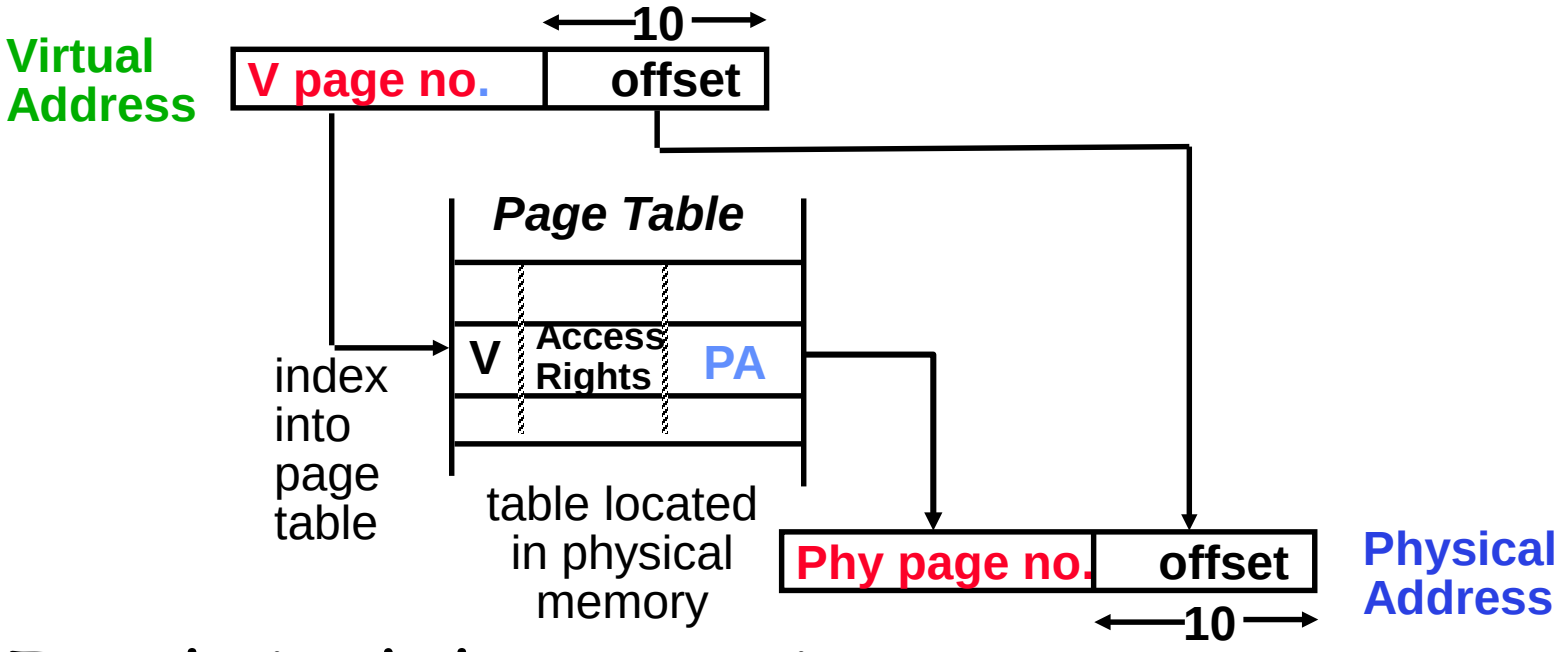


Example of Address Translation



Address Translation Details

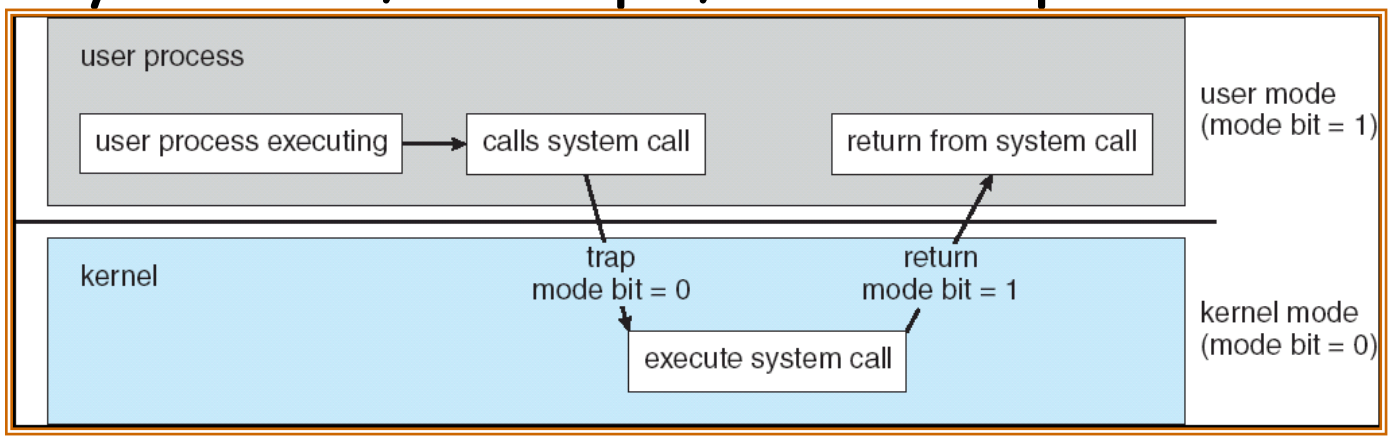
- For now, assume translation happens with table (called a Page Table):



- Translation helps protection:
  - Control translations, control access
  - Should Users be able to change Page Table?

Dual Mode Operation

- Hardware provides at least two modes:
  - "Kernel" mode (or "supervisor" or "protected"): privileged instructions executed (e.g., I/O ops)
  - "User" mode: Normal programs executed
- Dual-mode operation allows OS to protect itself and other system components
  - Mode bit provided by hardware in modern processor
- Some instructions/ops prohibited in user mode:
  - Example: cannot modify page tables in user mode
    - » Attempt to modify ⇒ Exception generated
- Transitions from user mode to kernel mode:
  - System Calls, Interrupts, Other exceptions



Multi Mode Operation

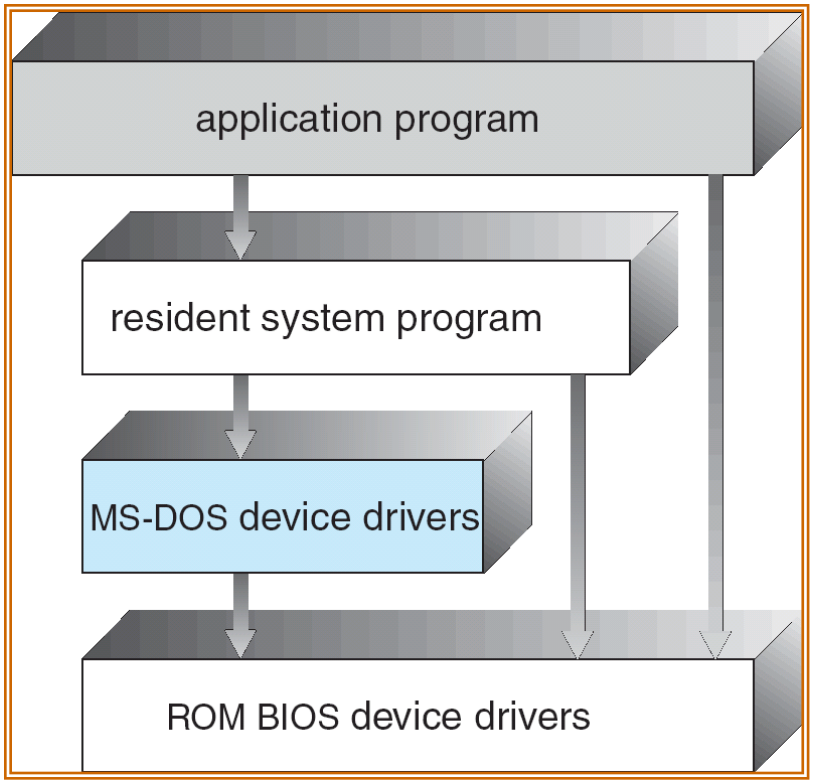
- CPUs supporting virtualization have a separate mode to indicate when VM manager is running
  - VMM has more privileges than User processes, but less than that of Kernel
- Intel x86_64 family supports 4 privilege levels
  - Different kernel components use diff levels
- MS-DoS on early Intel CPUs does not have mode bit
  - Malicious user prg can wipe out entire OS
- H/W protection detects errors that violate modes
  - H/W traps to OS, terminates user prg, memory dump is written to a file for later examination

Operating Systems Structure (What is the organizational Principle?)

- Simple
  - Only one or two levels of code
- Monolithic
  - Everything in one level
- Layered
  - Lower levels independent of upper levels
- Microkernel
  - OS built from many user-level processes
- Modular
  - Core kernel with Dynamically loadable modules
- Hybrid

Simple Structure

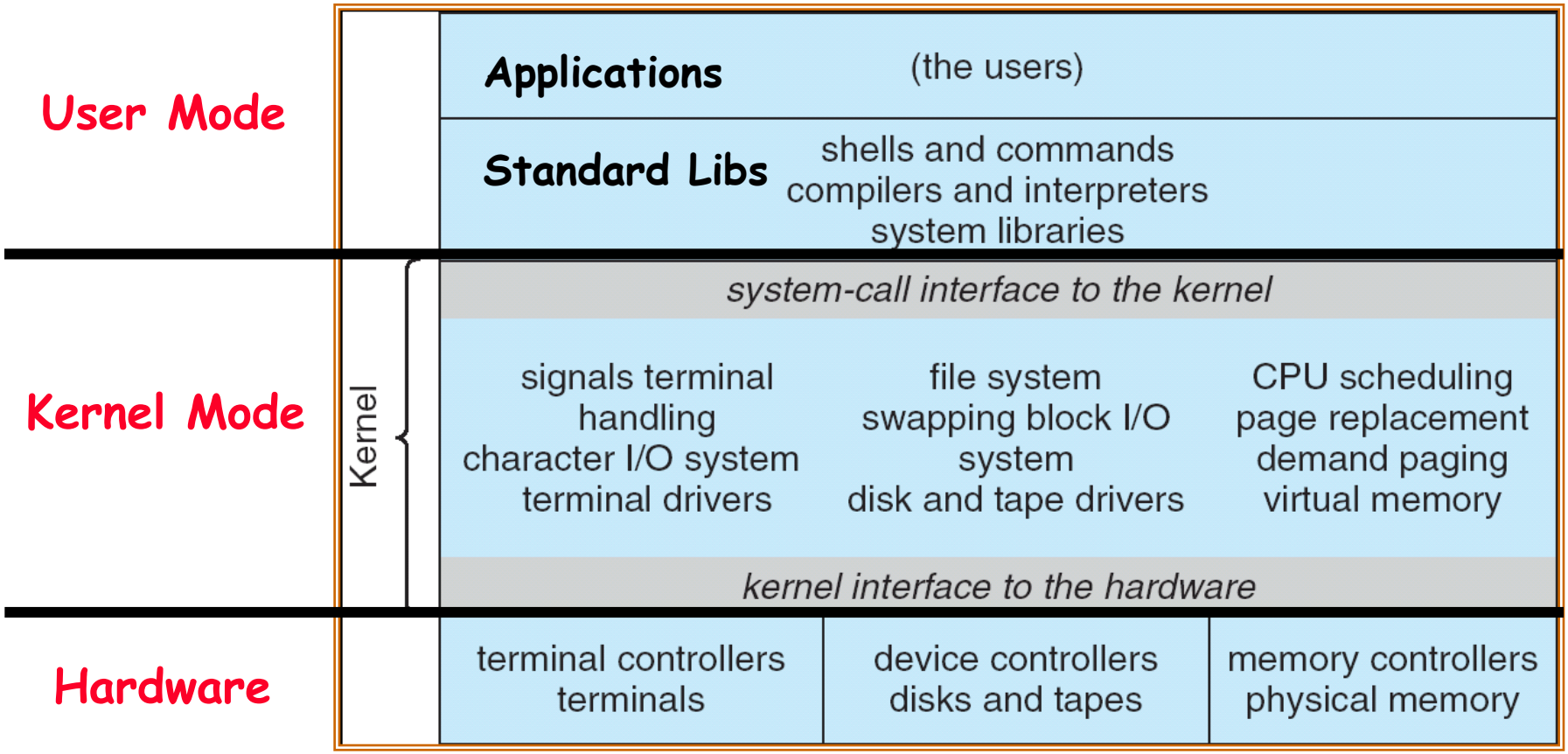
- MS-DOS – written for Intel 8088 to provide the most functionality in the least space
  - Not divided into modules
  - Interfaces and levels of functionality not well separated
  - No h/w protection
  - Users can access basic I/O routes and write directly to display and disk drives



UNIX: Also "Simple" Structure

- UNIX – limited by hardware functionality, it had limited structuring
- Original UNIX operating system consists of two separable parts:
  1. Systems programs
  2. The kernel
     - » Consists of everything below the system-call interface and above the physical hardware
     - » Provides the file system, CPU scheduling, memory management, and other OS functions;
     - » Many interacting functions in one level → monolithic structure
       - Difficult to implement and maintain
       - But, little overhead in the sys call interface or in communication within the kernel
     - » Linux and Windows also come under this category

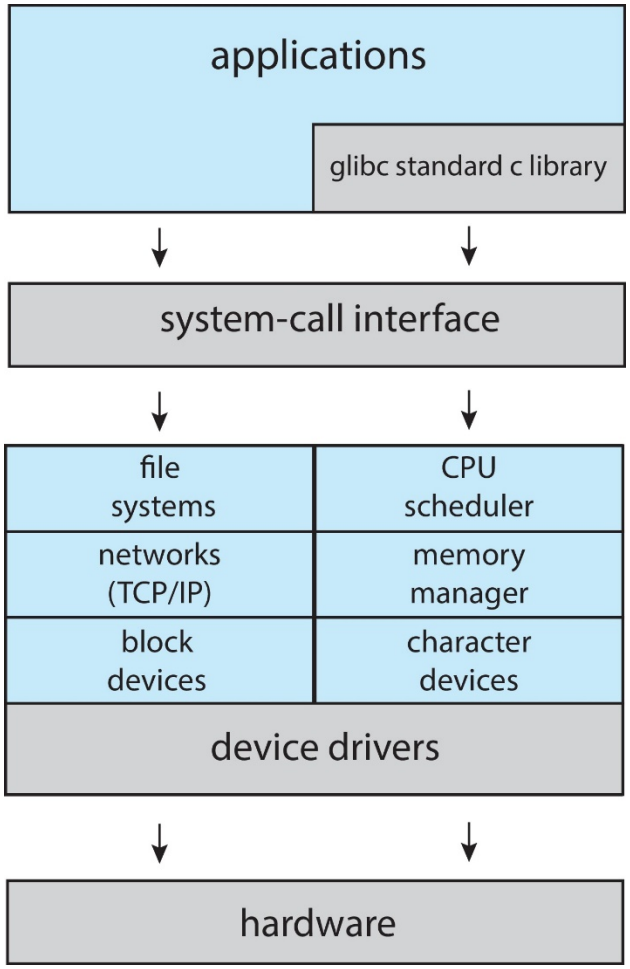
Traditional UNIX System Structure



Too much functionality at one level !
Beyond simple, but not fully layered.

Linux System Structure

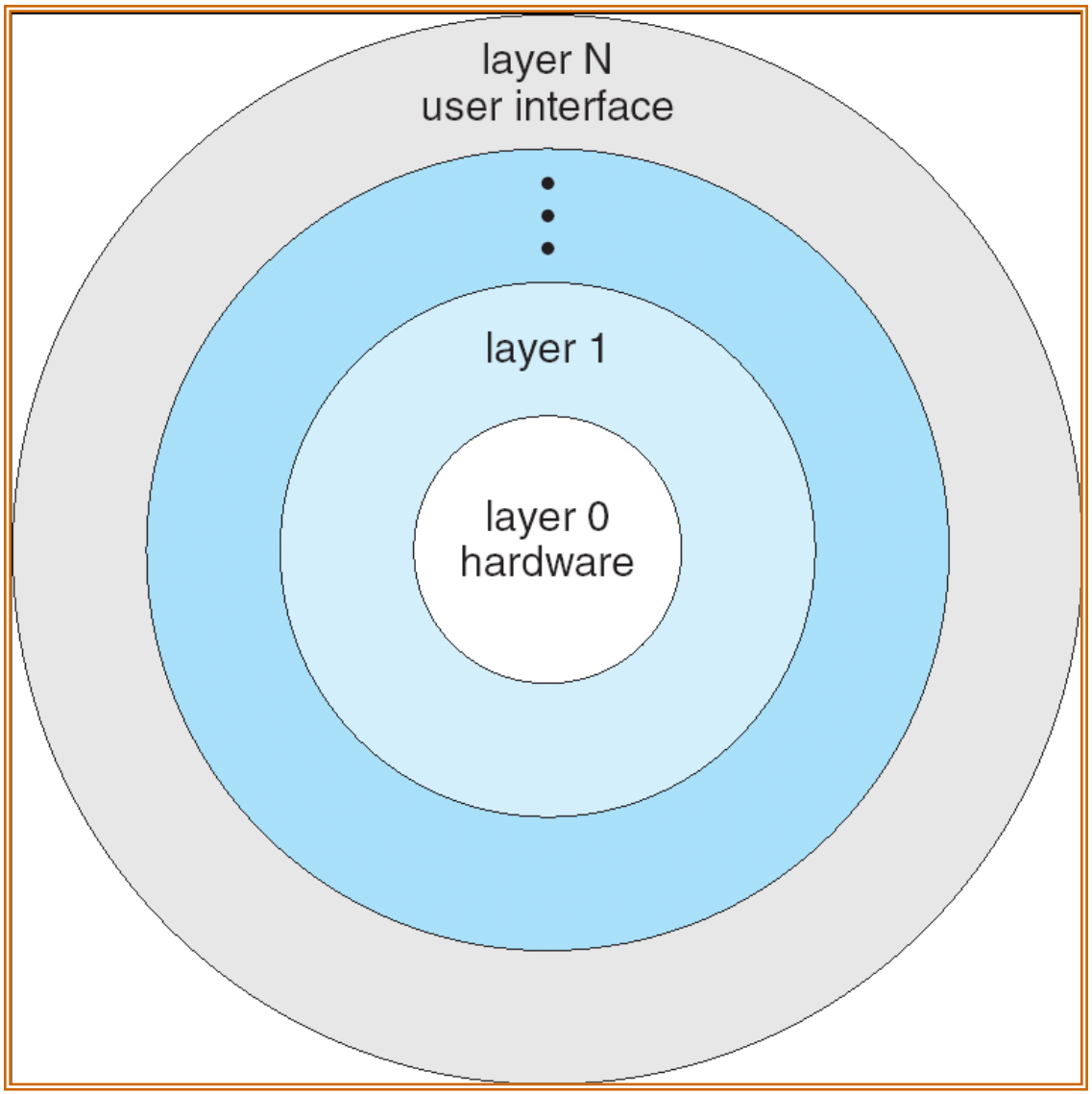
Monolithic plus modular design



Layered Structure

- Operating system is divided many layers (levels)
  - Each built on top of lower layers
  - Bottom layer (layer 0) is hardware
  - Highest layer (layer N) is the user interface
- Each layer uses functions (operations) and services of only lower-level layers
  - Advantage: modularity ⇒ Easier debugging/Maintenance
  - How many layers? Each layer adds overhead
  - Not always possible: Does process scheduler lie above or below virtual memory layer?
    - » Need to reschedule processor while waiting for paging
    - » May need to page in information about tasks
  - Each layer adds overhead to system calls
  - So, systems with fewer layers are preferred
- Important: Machine-dependent vs independent layers
  - Easier migration between platforms
  - Easier evolution of hardware platform

Layered Operating System



Microkernel Structure

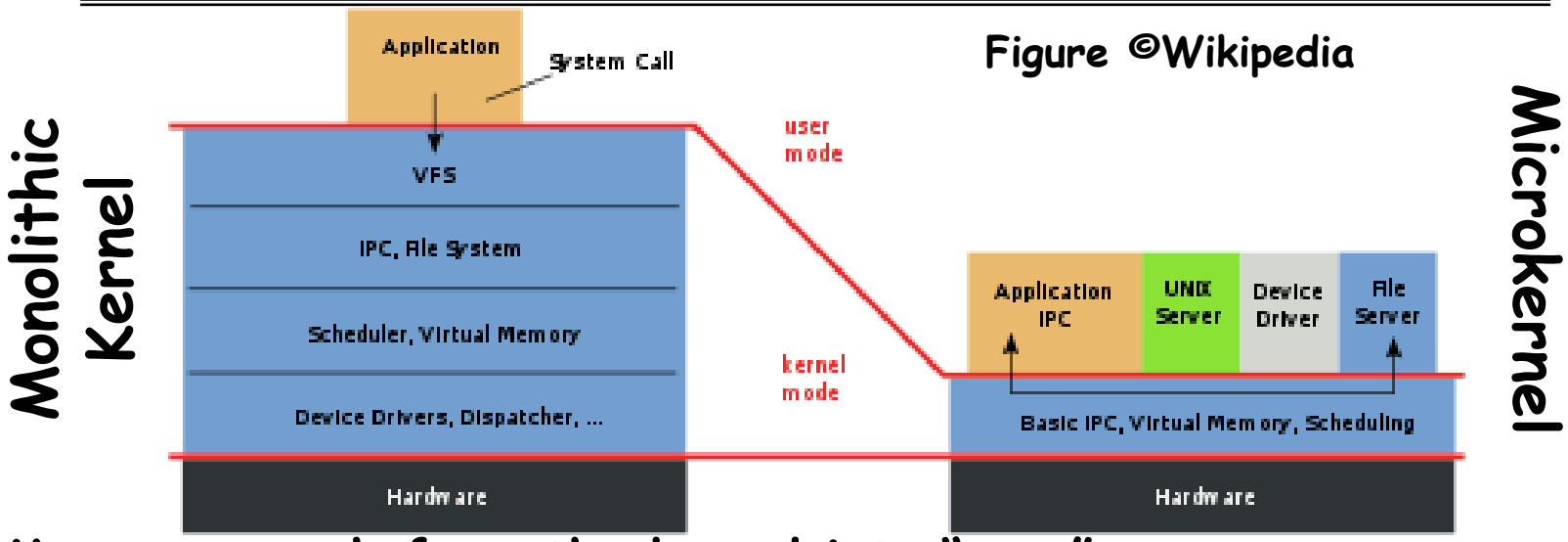
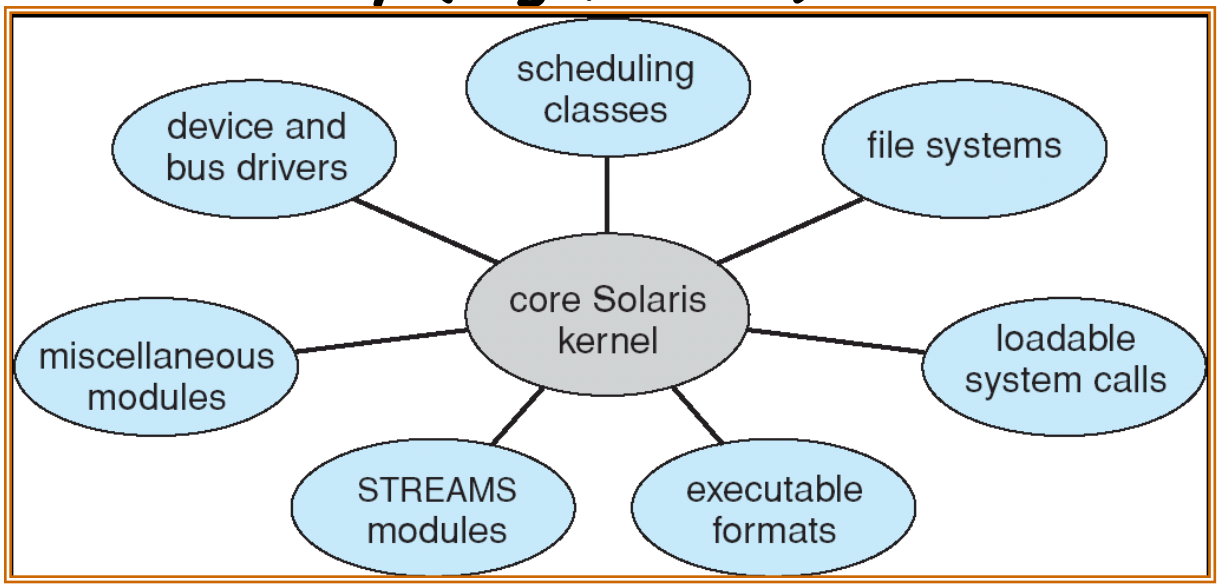


Figure ©Wikipedia

- Moves as much from the kernel into "*user*" space
  - Small core OS running at kernel level (e.g., Mach)
  - OS Services built from many independent user-level processes
  - Communication is provided through message passing via kernel indirectly
- Benefits:
  - Easier to extend a microkernel
  - Easier to port OS to new architectures
  - More reliable (less code is running in kernel mode)
  - Fault Isolation (parts of kernel protected from other parts)
  - More secure
- Detriments:
  - Performance overhead severe for naïve implementation & increased system-function overhead

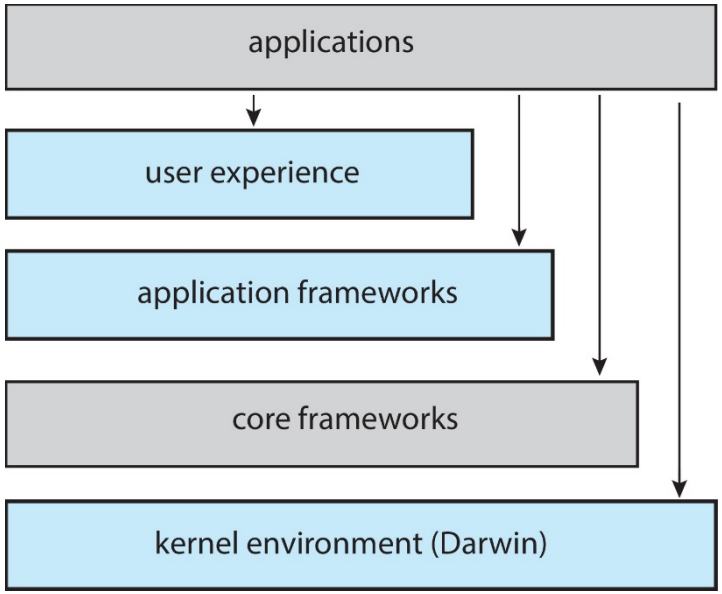
Modules-based Structure

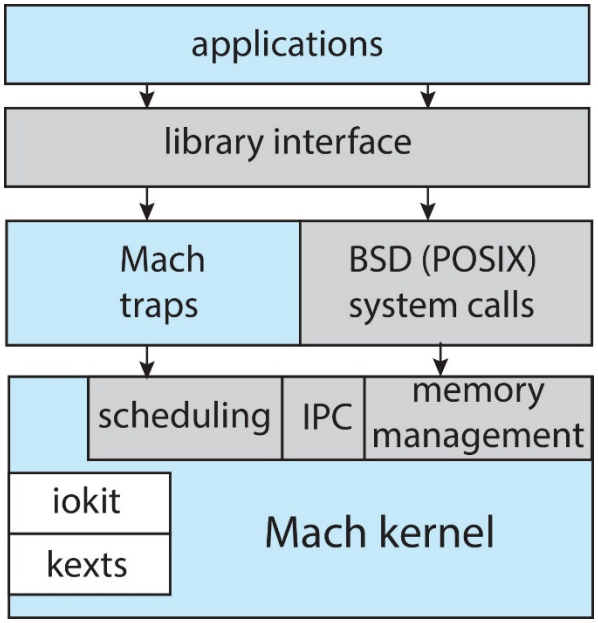
- Most modern operating systems implement loadable kernel modules (LKMs)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layered and microkernel but with more flexibility (e.g., Linux)



Hybrid Structure

- Most modern operating systems are Hybrid so as to address performance, security, usability needs
- E.g.,Apple MAC OS X
  - Mach Microkernel (MM, Scheduling, RPC, IPC)
  - BSD kernel (cmd interface, file system, POSIX API, networking)
  - Apps can make use of BSD/Mach facilities directly



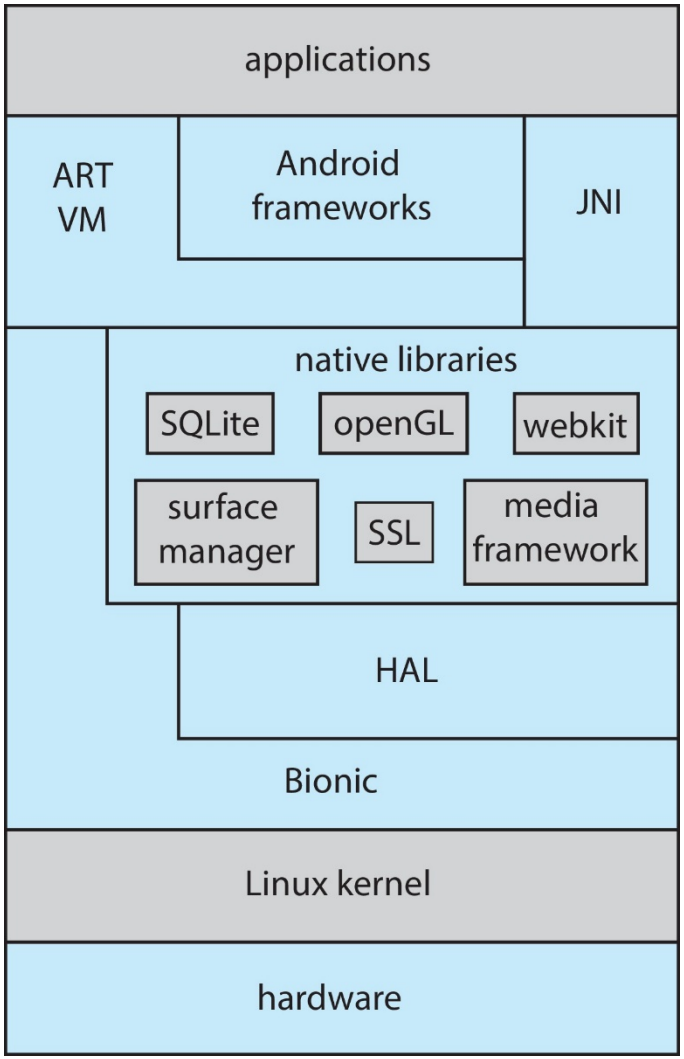


Darwin

Android

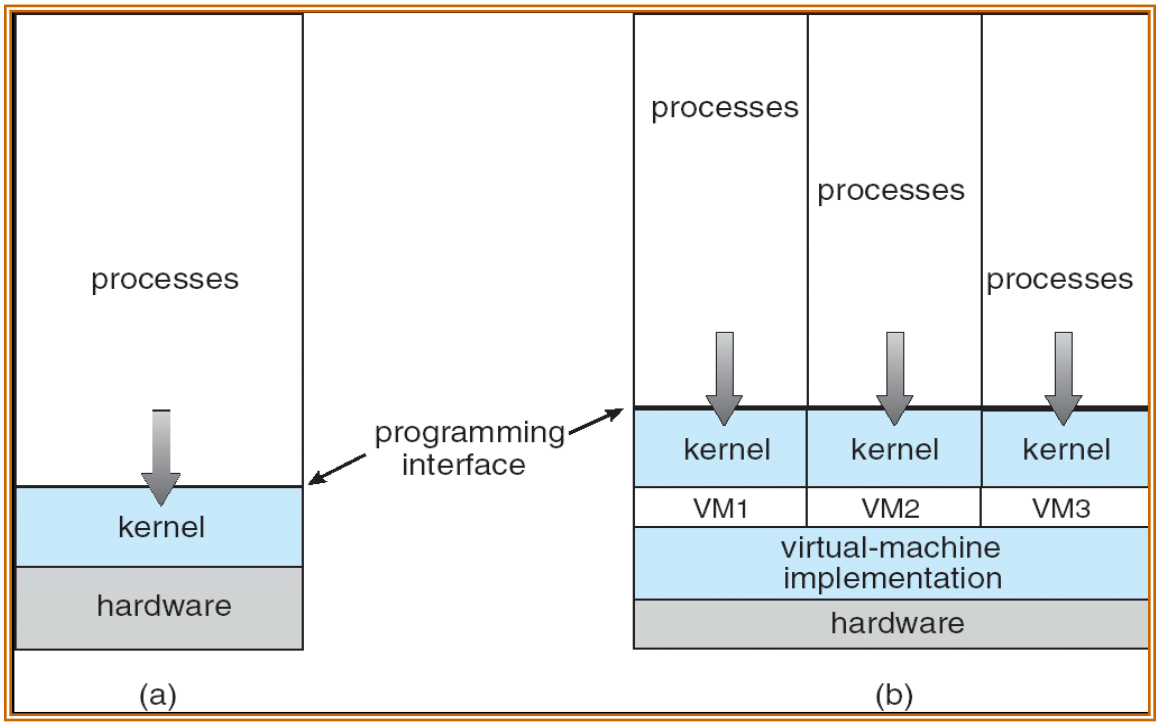
- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - » Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



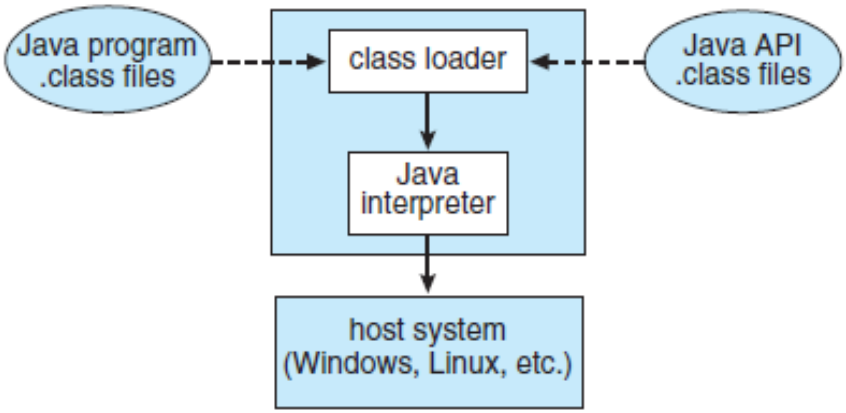
Virtual Machines

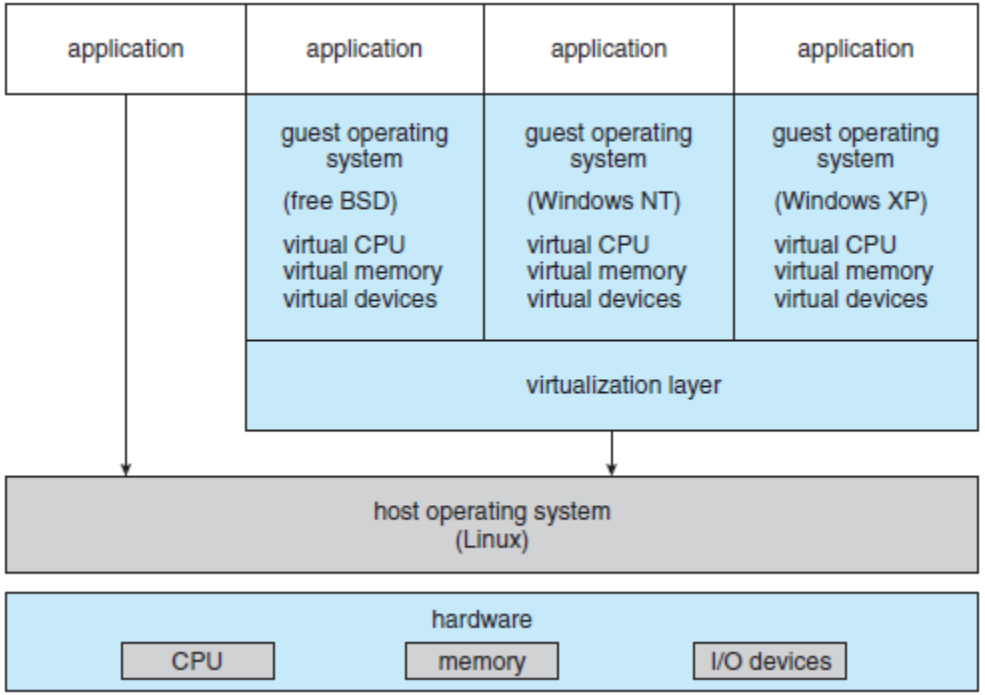
- Implements an observation that dates to Turing
  - One computer can "emulate" another computer
  - One OS can implement abstraction of a cluster of computers, each running its own OS and applications
- Incredibly useful!
  - System building
  - Protection
  - Consolidation
- Cons
  - Implementation
  - H/W support
  - Kernel support
- Examples
  - IBM VM370
  - VMWare, JVM



Virtual Machines

- VMWare
  - Vmware Workstation
  - Virtualization in user space on the host OS
- Java Virtual Machine
  - Abstract computer running in user space
  - Consists of Class loader and Java interpreter
  - Implemented in User space or inside browser





Issues in OS Design?

- **Structure:** how is an operating system organized ?
- **Sharing:** how are resources shared among users ?
- **Naming:** how are resources named by users or programs ?
- **Protection:** how is one user/program protected from another ?
- **Security:** how to authenticate, control access, secure privacy ?
- **Performance:** why is it so slow ?
- **Reliability and fault tolerance:** how do we deal with failures ?
- **Extensibility:** how do we add new features ?

Issues in OS Design?

- **Communication:** how can we exchange information ?
- **Concurrency:** how are parallel activities created and controlled ?
- **Scale, growth:** what happens as demands or resources increase ?
- **Persistence:** how can data outlast processes that created them
- **Compatibility:** can we ever do anything new ?
- **Distribution:** accessing the world of information
- **Accounting:** who pays bills, and how to control resource usage

Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary closed-source
- Counter to the copy protection and Digital Rights Management (DRM) movement
- Started by Free Software Foundation (FSF), which has "copyleft" GNU Public License (GPL)
  - Free means liberty and freedom to run/copy/distribute/study/change/improve S/W
- Examples include GNU/Linux and BSD UNIX (including core of Mac OS X), and many more
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
  - Use to run different guest OSs for exploration

Why Study Operating Systems?

- Learn how to build complex systems:
  - How can you manage complexity for future projects?
- Engineering issues:
  - Why is the web so slow sometimes? Can you fix it?
  - What features should be in the next Mars Rover?
  - How do large distributed systems work? (DNS, skype, etc)
- Buying and using a personal computer:
  - Why different PCs with same CPU behave differently
  - How to choose a processor (i3/i5/i7, Xeron, etc)? Should you get Windows 10, Linux, Mac OS …?
- Business issues:
  - Should your division buy thin-clients vs PC?
- Security, viruses, and worms
  - What exposure do you have to worry about?

The Study of Operating Systems

There has never been a more interesting time to study operating systems, and it has never been easier. The open-source movement has overtaken operating systems, causing many of them to be made available in both source and binary (executable) format. The list of operating systems available in both formats includes Linux, BUSD UNIX, Solaris, and part of macOS. The availability of source code allows us to study operating systems from the inside out. Questions that we could once answer only by looking at documentation or the behavior of an operating system we can now answer by examining the code itself.

Operating systems that are no longer commercially viable have been open-sourced as well, enabling us to study how systems operated in a time of fewer CPU, memory, and storage resources. An extensive but incomplete list of open-source operating-system projects is available from https://curlie.org/Computers/Software/Operating_Systems/Open_Source/

In addition, the rise of virtualization as a mainstream (and frequently free) computer function makes it possible to run many operating systems on top of one core system. For example, VMware (http://www.vmware.com) providesa free “player” for Windows on which hundreds of free “virtual appliances” can run. Virtualbox (http://www.virtualbox.com) provides a free, open-source virtual machine manager on many operating systems. Using such tools, students can try out hundreds of operating systems without dedicated hardware.

The advent of open-source operating systems has also made it easier to make the move from student to operating-system developer. With some knowledge, some effort, and an Internet connection, a student can even create a new operating-system distribution. Just a few years ago, it was difficult or impossible to get access to source code. Now, such access is limited only by how much interest, time, and disk space a student has.

"In conclusion…"

- Operating systems provide a virtual machine abstraction to handle diverse hardware
- Operating systems coordinate resources and protect users from each other
- Operating systems simplify application development by providing standard services
- Operating systems can provide an array of fault containment, fault tolerance, and fault recovery


MY COMPUTER DOESN'T WORK! THE HARD DRIVE CRASHED! WHAT DO I DO?!
DID YOU BACK UP?
WHY? IS IT GONNA BLOW?!
STUNFELD ©96

Reading Assignments & References

- Watch UCB Video lecture 1-2 on OS

https://www.youtube.com/playlist?list=PLggtecHMfYHA7j2rF7nZFgnepu_uPuYws

Also checkout this playlist as well:

https://www.youtube.com/playlist?list=PL--jIyXjDXf6Q4XA6q8RYnyChYzJ0K0F2

- Chapters 1-2 from OSC by Galvin et al
  - System calls, OS Structures
- https://www.youtube.com/watch?v=X5lpOskKF9I (Altair 8800)
- Reference material and Linux VM from Galvin et al

http://www.os-book.com/

- Free vs Open-source: https://www.debian.org/intro/free