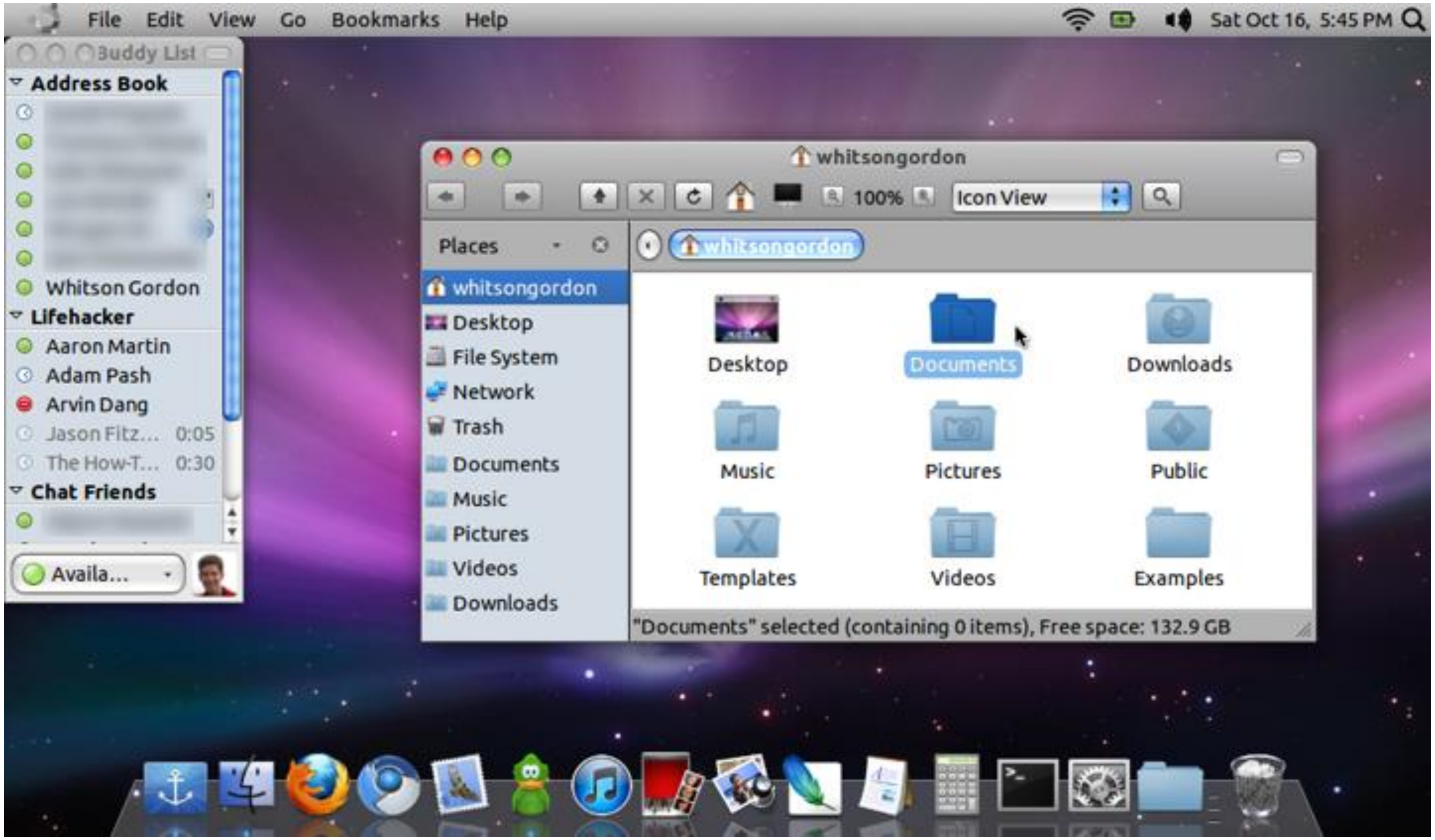
2. Operating System Structures

ECE30021/ITP30002 Operating Systems

Hybrid Systems

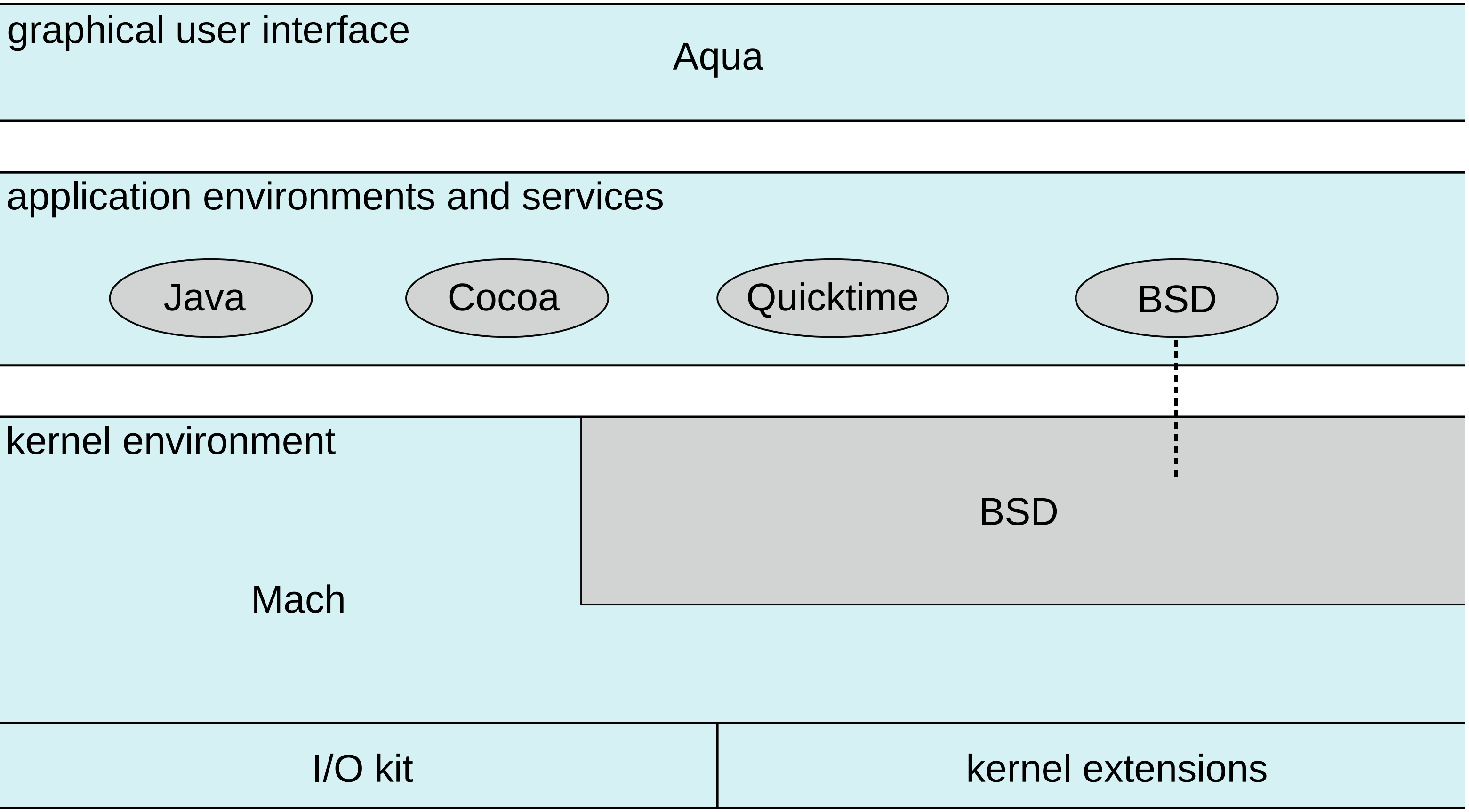
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem personalities
- Apple Mac OS X
  - Hybrid, layered, Aqua UI plus Cocoa programming environment
  - Kernel environment
    Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)

Mac OS X, Aqua User Interface



"Documents" selected (containing 0 items), Free space: 132.9 GB

Mac OS X Structure

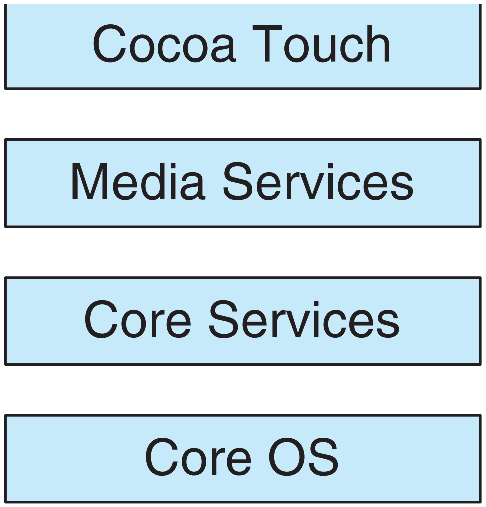


BSD application environment can make use of BSD facilities directly

I/O kit for development of device drivers
Kernel extensions: Dynamically loadable modules

iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
- Cocoa Touch
  - Objective-C API for developing apps
  - Cocoa vs. Cocoa Touch?
    Cocoa Touch provides support for hardware features unique to mobile devices.
- Media services
  - layer for graphics, audio, video
- Core services
  - provides cloud computing, databases
- Core operating system,
  based on Mac OS X kernel

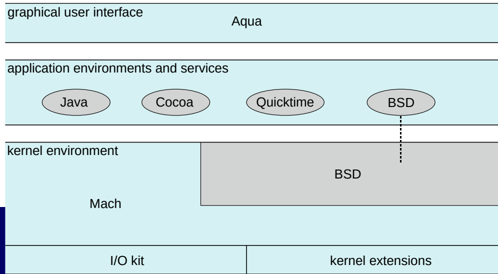


Cocoa Touch

Media Services

Core Services

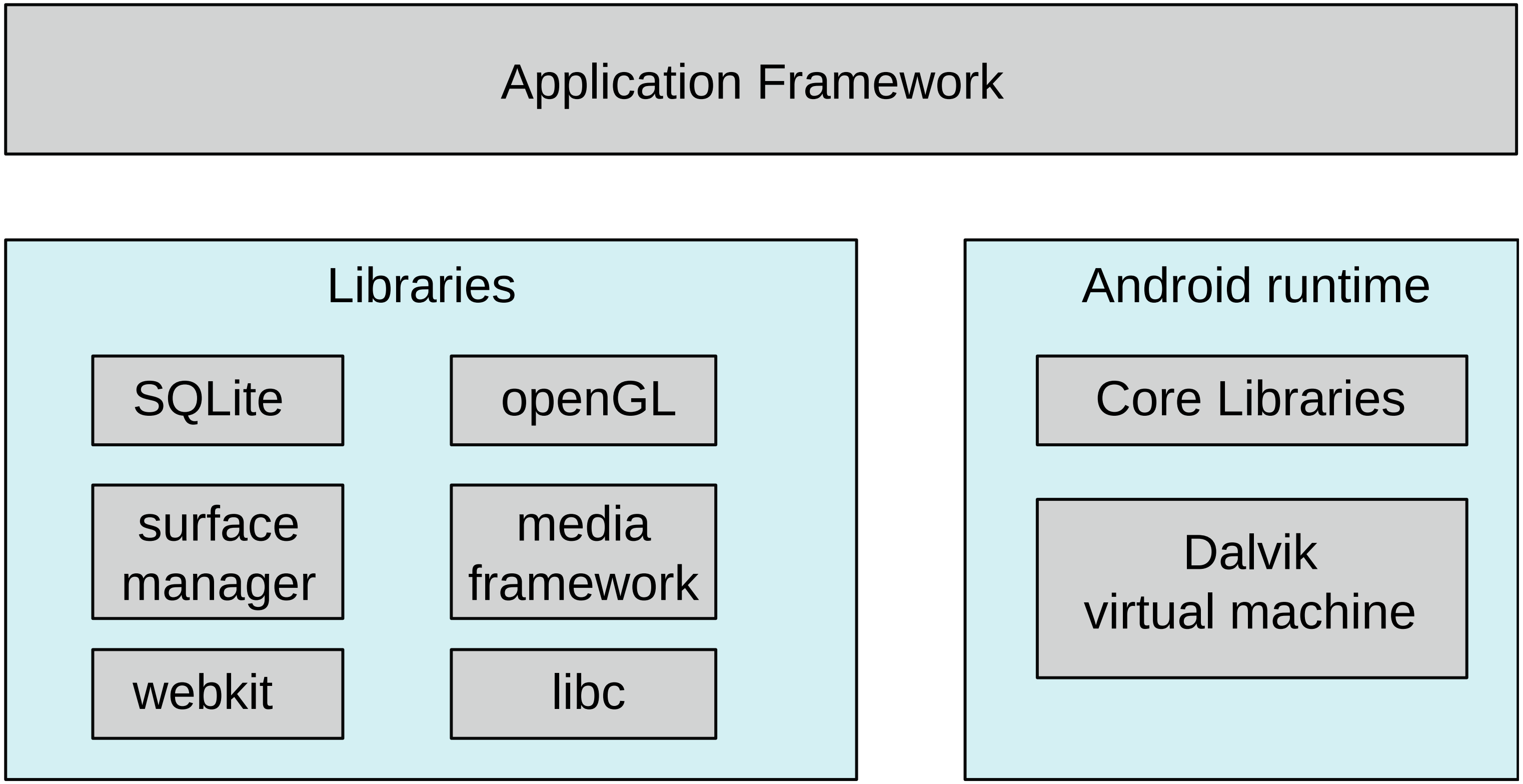
Core OS



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 (now, replaced by ART)
  - 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



Libc library is similar to the standard C library but is much smaller and has been designed for the slower CPUs that characterize mobile devices.

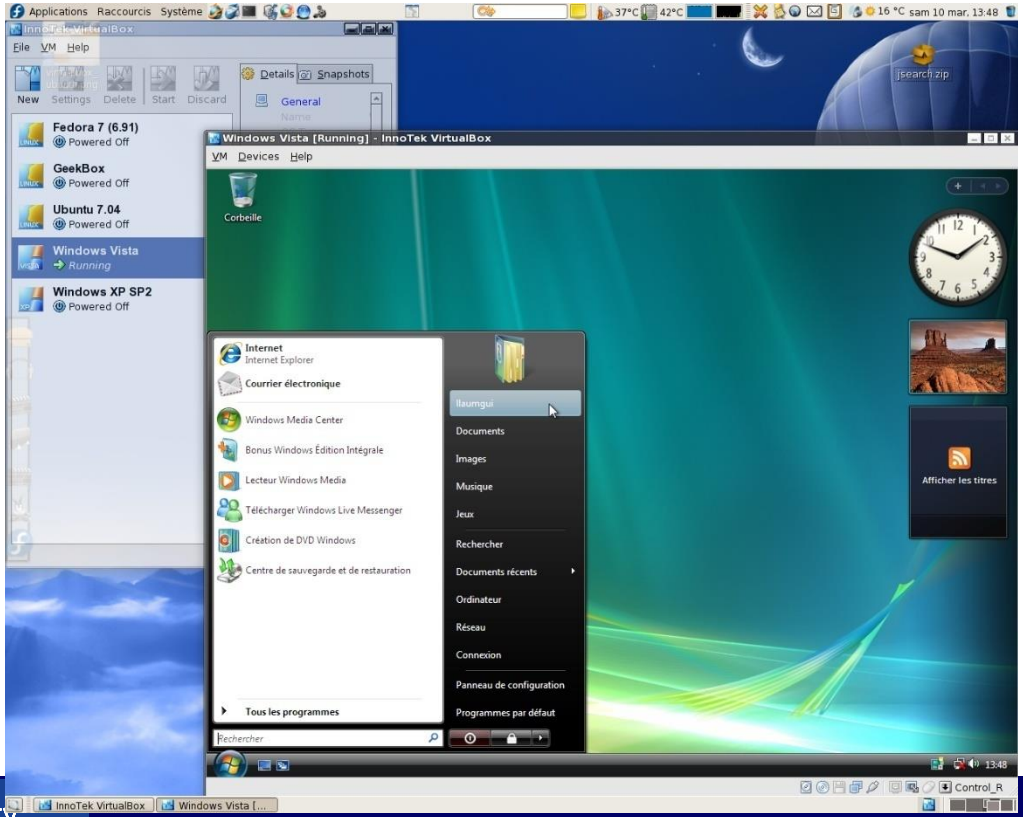
Agenda

- Operating-system services
- Interfaces for users and programmers
- Components and their interconnections
- **<u>Virtual Machines</u>**
- Design, implementation, generation
- System boot

Virtual Machines

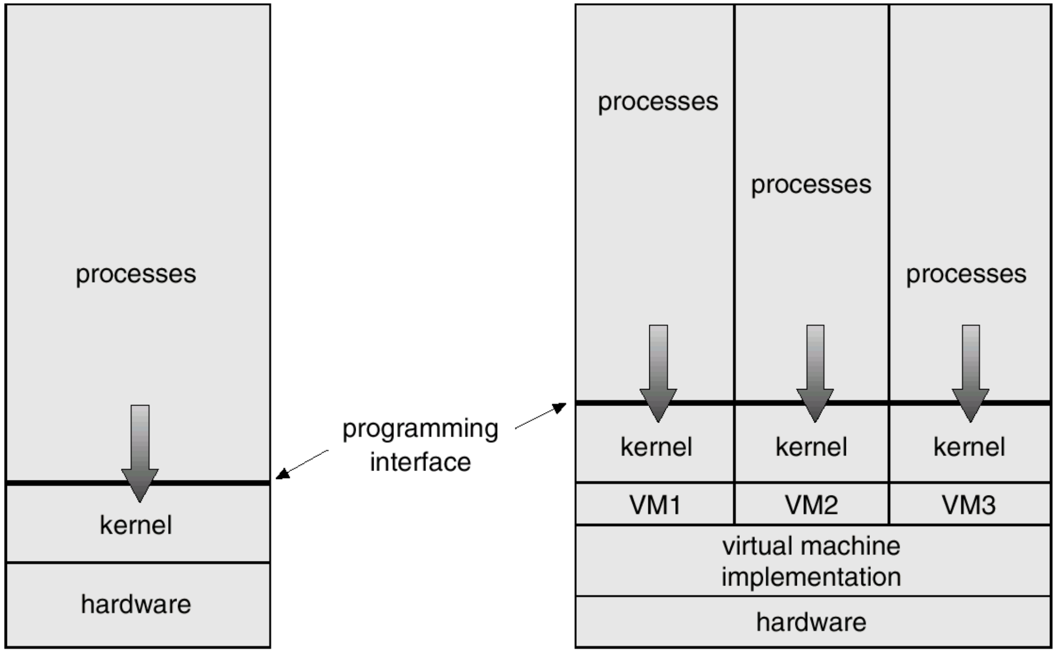
- **Virtual machine**: software that creates a virtualized environment (machine) between the computer platform and its operating system, so that the end user can operate software on an abstract machine.

  Ex) VMWare, VirtualPC, VirtualBox(www.virtualbox.org)



Virtual Machines

- Abstract H/W of single computer into several different execution environment
  - A number of different identical execution environments on a single computer, each of which exactly emulates the host computer.



Virtual Machines

- Each process seems to have its own CPU and memory
  - CPU scheduling + virtual memory technologies
    - Virtual memory allows software to run in a memory address space whose size and addressing are not necessarily tied to the computer's physical memory.

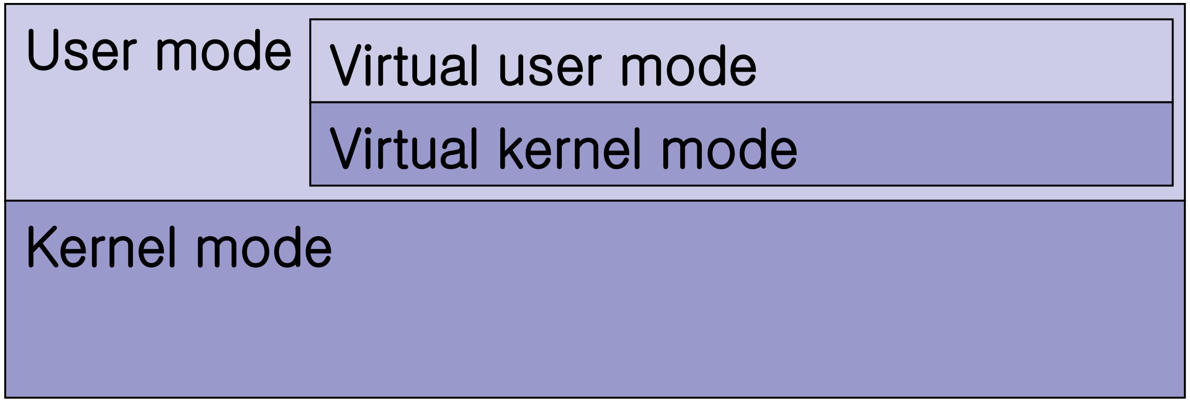
- Major difficulty: disk space
  - It is impossible to allocate same disk drive to each virtual machine
  - Solution: virtual disks (minidisks)
    - Identical in all respects except size

Virtual Machines

- Implementation problems
  - Exact duplication of underlying machine requires much work
  - Support for dual mode operation: virtual dual mode

    Cf. VM S/W can run in kernel mode, but VM itself is executed in user mode

    - Virtual user mode / virtual kernel mode
      - System call from virtual user mode is simulated by VM monitor
    - Many CPUs support more than two privilege levels.



User mode

Virtual user mode

Virtual kernel mode

Kernel mode

Virtual Machines

- Benefits of VM
  - Complete protection of various system resources

  cf. Sharing between VM’s
    - Shared minidisk
    - Virtual network connection

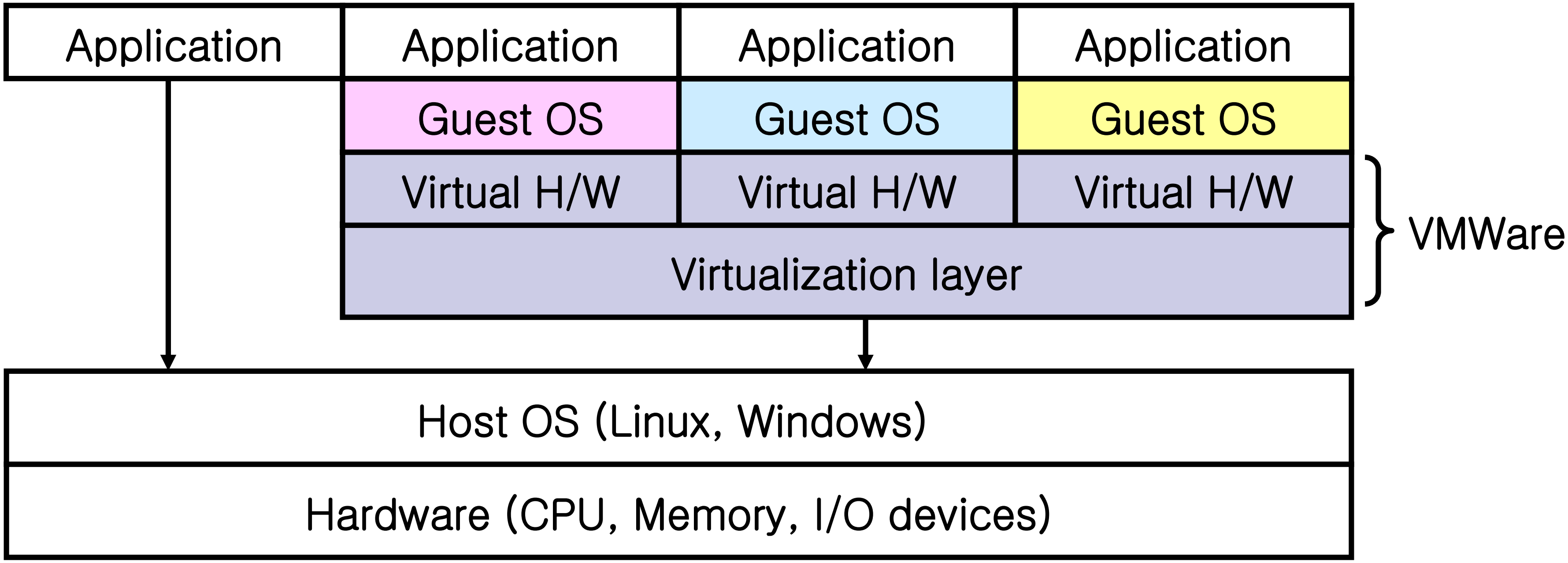
- Perfect vehicle for operating-systems research, development, and education
  - Changing OS is dangerous -> test is very important
  - Working on VM, system programmer don’t have to stop physical machine

Virtual Machines

- Inevitable differences from host system
  - Disk size
  - Execution time
    - Multiprogramming among many VM’s can slow down VM’s in unpredictable ways
    - Privileged instructions on VM are slow because they are simulated
    - Virtual I/O can be faster (spooling) or slower (interpreted)

Examples of VM: VMware

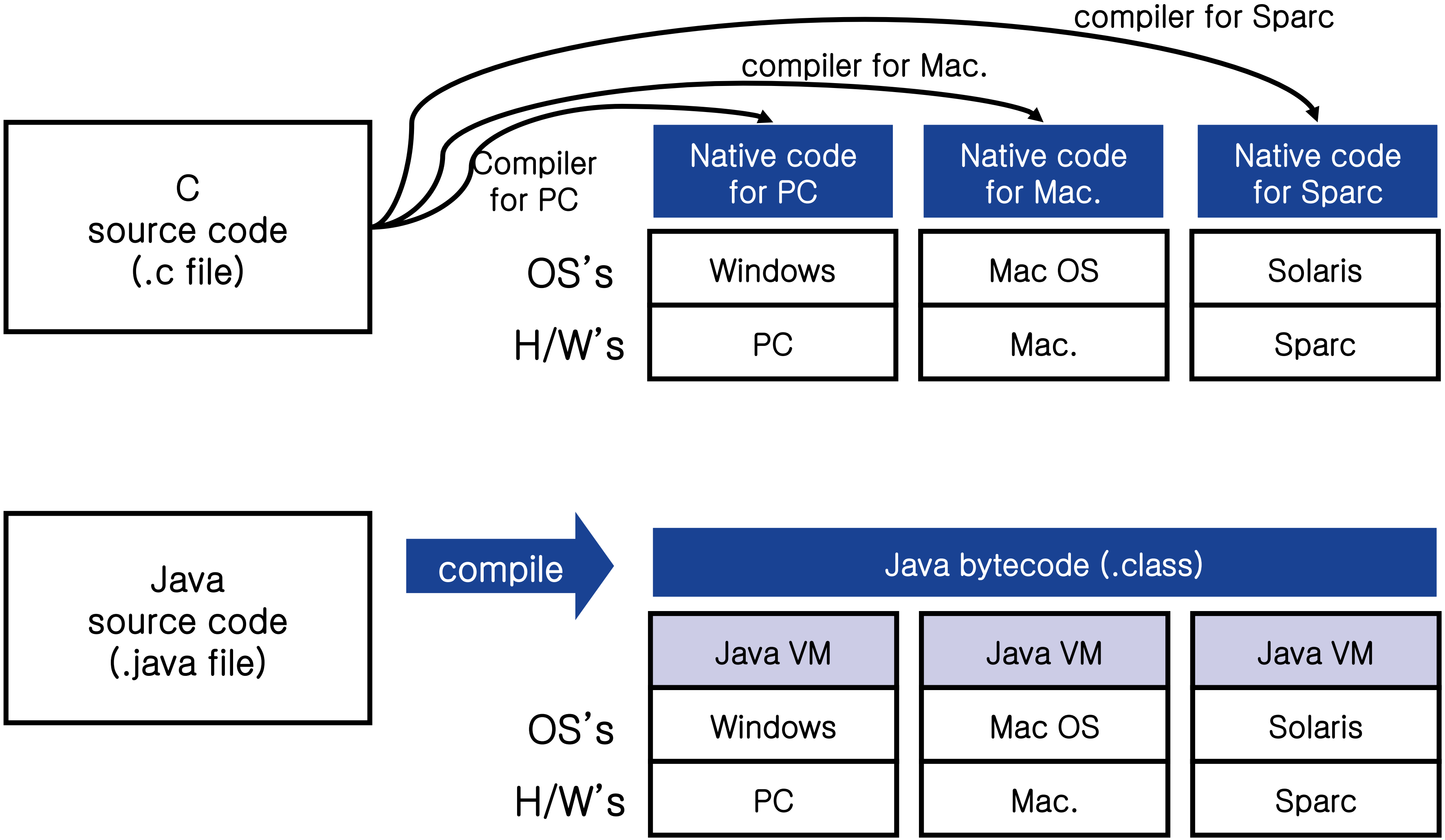
- A commercial VM of Intel 80x86 H/W
  - Runs on Windows or Linux
  - Allows the host to run guest operating systems as VM's
  - Major use
    - Testing an application on several different OS's



Examples of VM: JVM

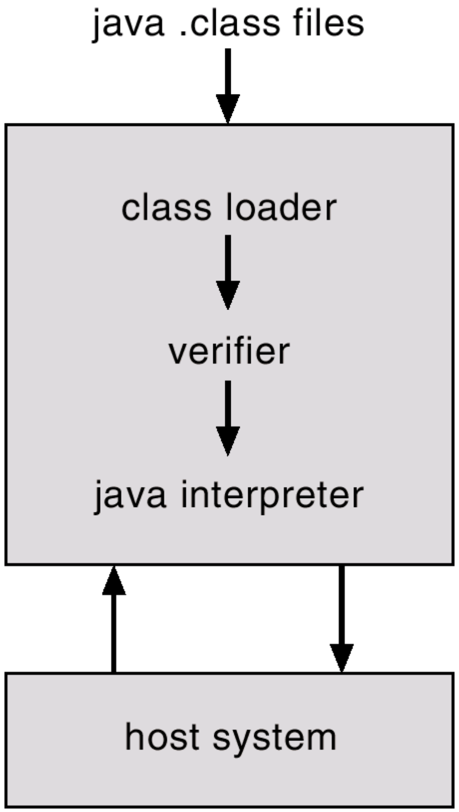
- Java
  - OOP language developed by SUN, 1995
  - Components
    - Language specification + Large API library
    - Specification for JVM (Java Virtual Machine)
  - Java objects are specified with class structure in bytecode
    - Bytecode: architecture-neutral code executed on JVM
    - *“Compile Once! Run Everywhere!”*

Examples of VM: JVM



Examples of VM: JVM

- JVM
  - Class loader: loads class file
  - Verifier: check loaded class file
    - Is it valid Java bytecode?
    - Doesn’t it overflow or underflow the stack?
    - Doesn’t it perform pointer arithmetic that can provide illegal memory access
  - Java interpreter: executes bytecode
    - Automatic memory management
      - Reclaim memory not in use and return it to system (garbage collection)



Examples of VM: JVM

- Implementation of JVM
  - S/W implementation on host OS’s or web-browser
    - Interpreter
    - Just-in-time (JIT) complier
      - The first time a Java method is invoked, its bytecode is compiled into native machine language
  - Specialized H/W
    - Java chip: execute Java bytecodes as native code

Similar Concepts

- **Emulator**
  - An emulator is a piece of software (it was microcode back then, but that's basically software as well) to "migrate" applications from one computer system to another.
    - WIPI emulator, MAME, AppleWin, …
    - Wine (?): An open source implementation of the Windows API on top of X and Unix. , …
  - Unlike a simulation, which only attempts to reproduce a program's behavior, an emulation generally attempts to model to various degrees the state of the device being emulated.
- **Simulator**
  - An imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system.
    - Flight simulator, …
  - Simulator is quite similar to emulator. But a simulator tends to be used more for the demonstration how a system works.

Agenda

- Operating-system services
- Interfaces for users and programmers
- Components and their interconnections
- Virtual Machines
- **Design, implementation, generation**
- System boot

OS Design and Implementation

- Design goals
  - No unique solution
  - Depend on H/W and type of systems
- User goals: obvious properties
  - Convenient, easy to learn/use, reliable, safe, fast
- System goals
  - Easy to design, implement, maintain
  - Flexible, reliable, error-free, efficient

OS Design and Implementation

- Mechanisms and policies
  - Mechanism: how to do something
    - Should be insensitive to change in policies
  - Policies: what will be done

    Ex) I/O intensive programs have priority over CPU-intensive programs

- Separation of mechanisms and policies -> flexibility
  - Microkernel: OS is implemented by a basic set of primitive building block, independent of policies
    - In Solaris, scheduling is controlled by loadable tables
  - MS-Windows: mechanism and policy are encoded in system
    - Similar interfaces over all applications

OS Design and Implementation

- Language to implement OS
  - Assembly (early systems)
  - High-level language
    - MCP (Master Control Program) – ALGOL
    - MULTICS – PL/1
    - Linux, Windows – C + assembler

Assembly	High-level language
Fast Reduced space (not major issues in today's system)	Convenient (same language with application program) Fast implementation Easy to understand and debug Take benefit of advance in compilers Portability

OS Design and Implementation

- Current trend
  1. Write most of OS in high-level languages
  2. Improve data structures and algorithms
  3. Rewrite only critical routines in Assembly
     - Performance analysis is essential.

Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs** in S/W and H/W
- Performance problems are considered bugs, so debugging can also include performance tuning.

- Most operating systems generate **log files** containing error information fi a process fails.
- Failure of an application can generate **core dump** file capturing memory of the process (memory was referred to as the “core” in the early days of computing).

- But OS kernel debugging is even more complex because of
  - the size and complexity of the kernel,
  - its control of the hardware, and
  - the lack of user-level debugging tools.

Operating-System Debugging

- A crash: a failure in the kernel
- Operating system failure (a crash) can generate **crash dump** file containing kernel memory
- But may use different tools and techniques.
  - Process failure: writing a file
  - OS kernel failure: saving the kernel's memory state to the isolated section of the disk.
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan's Law

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Operating-System Generation

- System generation (SYSGEN)
  - OS's are designed to run on various environment
    - Any class of machines
    - Variety of sites
    - Variety of peripheral configuration
  - In order to run an OS on an actual system, it should be configured to the environment.

- How to acquire information about environment
  - File
  - User's response
  - Automatic probe

Operating-System Generation

- Information required for SYSGEN
  - CPU and option (extended instruction set, floating point arithmetic, …)
  - Size of available Memory
  - Available devices
  - Options and parameters of desired OS
    - Number and size of buffers
    - CPU-scheduling algorithm
    - Maximum number of processes
    - Etc.

Agenda

- Operating-system services
- Interfaces for users and programmers
- Components and their interconnections
- Virtual Machines
- Design, implementation, generation
- **System boot**

System Boot

- Booting
  procedure of starting a computer by loading the kernel

- Bootstrap program (bootstrap loader)
  - Run diagnostics
  - Initialize system
    - CPU register, device controller, memory contents
  - Locates, loads, starts kernel

System Boot

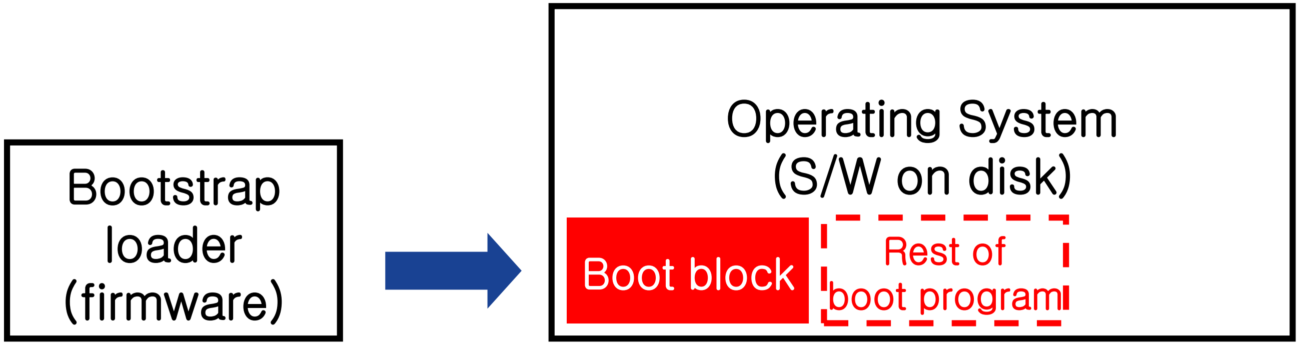
- OS's in ROM (firmware)

  Ex) Cellular phones, PDA's, game consoles

  - Small OS, simple supporting H/W, rugged operation
  - Problems
    - Difficult to change bootstrap code
      - Use EPROM (Erasable Programmable ROM) instead of ROM
    - Execution of a program in firmware is slower than that in RAM
      - Copy execution program to RAM for fast execution
    - Expensive
      - Applicable for small OS

System Boot

- Large OS's: two-step booting process
  - Bootstrap loader in firmware loads **boot block** of OS
    - Bootstrap loader is too simple to load entire OS
  - Boot block
    - Sophisticated boot program to load entire OS or
    - Simple code and information about remained part of bootstrap program


Bootstrap loader (firmware)
Operating System (S/W on disk)
Boot block
Rest of boot program