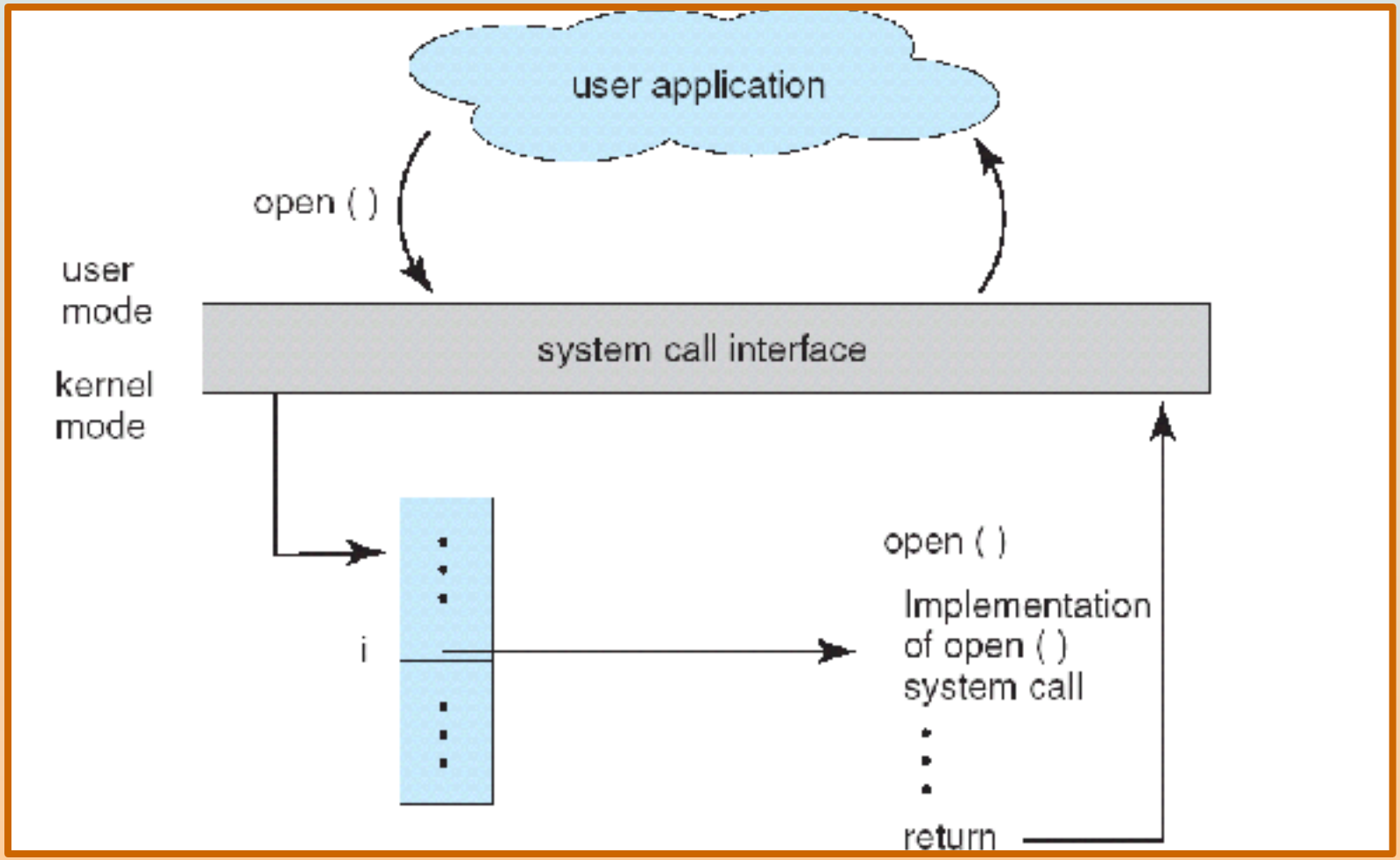
OS Structures

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

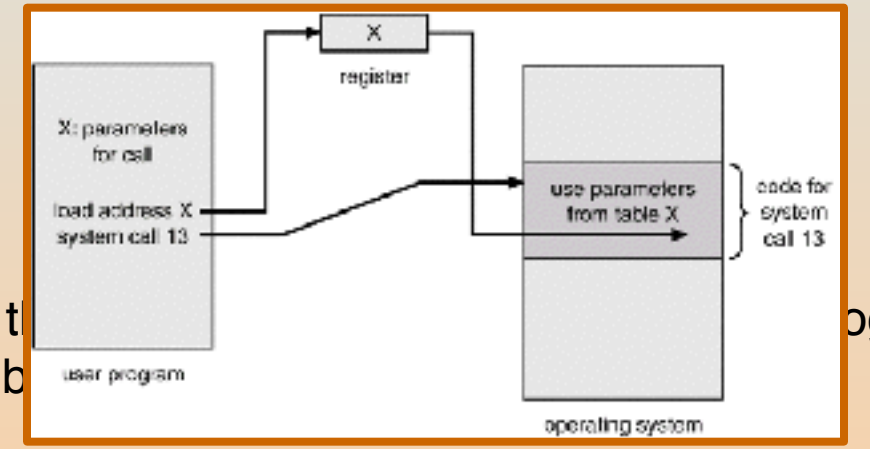
System Calls

System calls provide the interface between a running program and the operating system



System Calls

- Three general methods are used to *pass parameters* between a running program and the operating system
  - Pass parameters in *registers*
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register



- *Push* (store) t ogram, and *pop* off the stack b

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

Abstraction from the Hardware

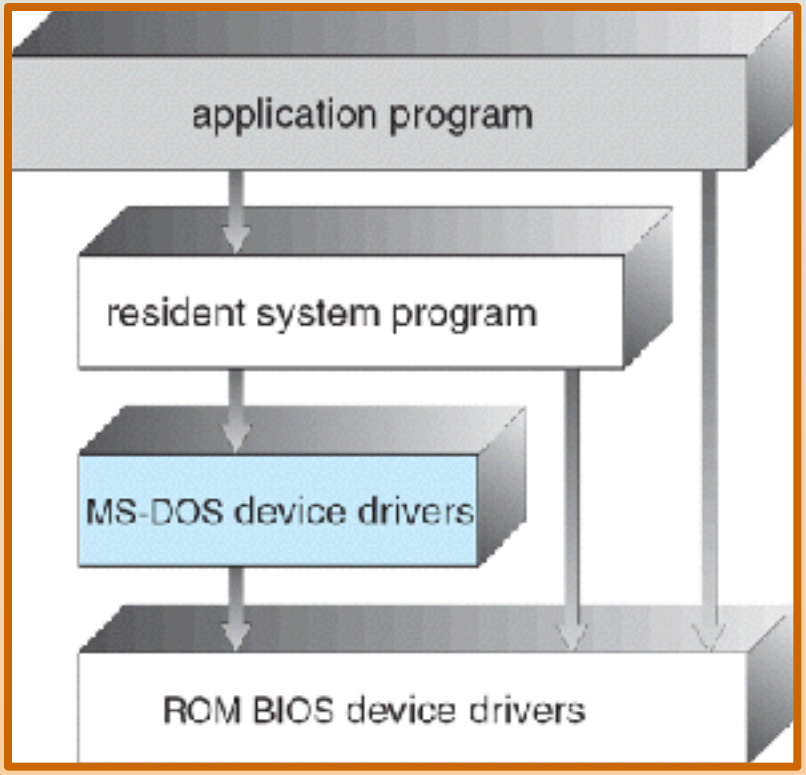
- From transistors to 0/1 bits
- Logic gates abstract away the details of CMOS.
- Machine language abstracts away the details of logic gates.
- Assembly language abstracts away the details of machine languages.
- Programming language abstracts away the details of assembly languages

The History of OS

- 1940s and 1950s
  - IOCS, Storage, Batch processing
- 1960s
  - Time sharing, Multiprogramming
  - IBM OS/360, Multics
- 1960s-1970s
  - UNIX
  - “the genius of the UNIX system is its framework, which enables programmers to stand on the work of others”
- 1980s
  - PC
  - Apple, IBM, CP/M, Bill Gates
  - Macintosh, Windows
- 1990s
  - Windows, Unix, Linux

Simple Structure

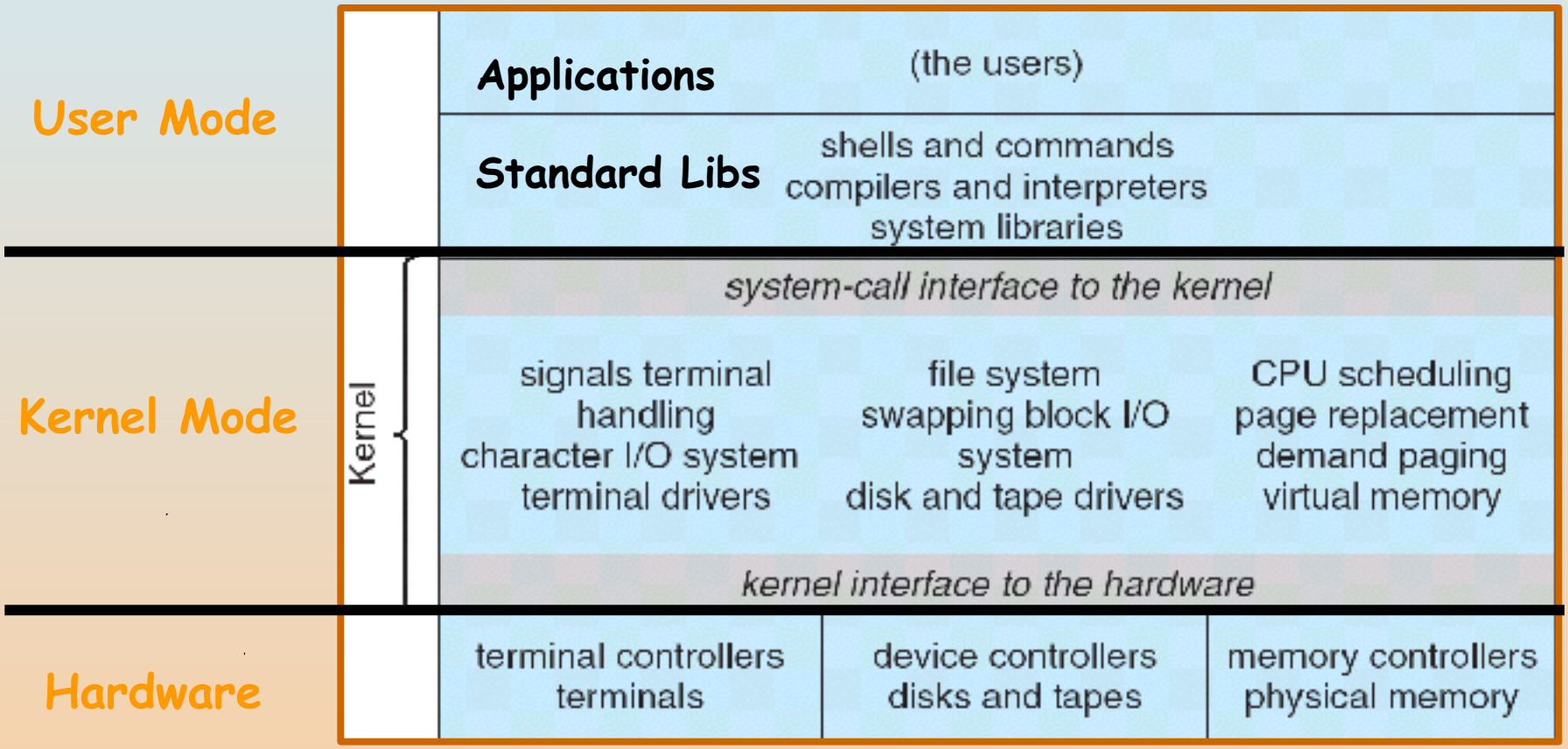
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Interfaces and levels of functionality not well separated



UNIX: Also “Simple” Structure

- UNIX – limited by hardware functionality
- Original UNIX operating system consists of two separable parts:
  - Systems programs
  - 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 operating-system functions;
    - Many interacting functions for one level

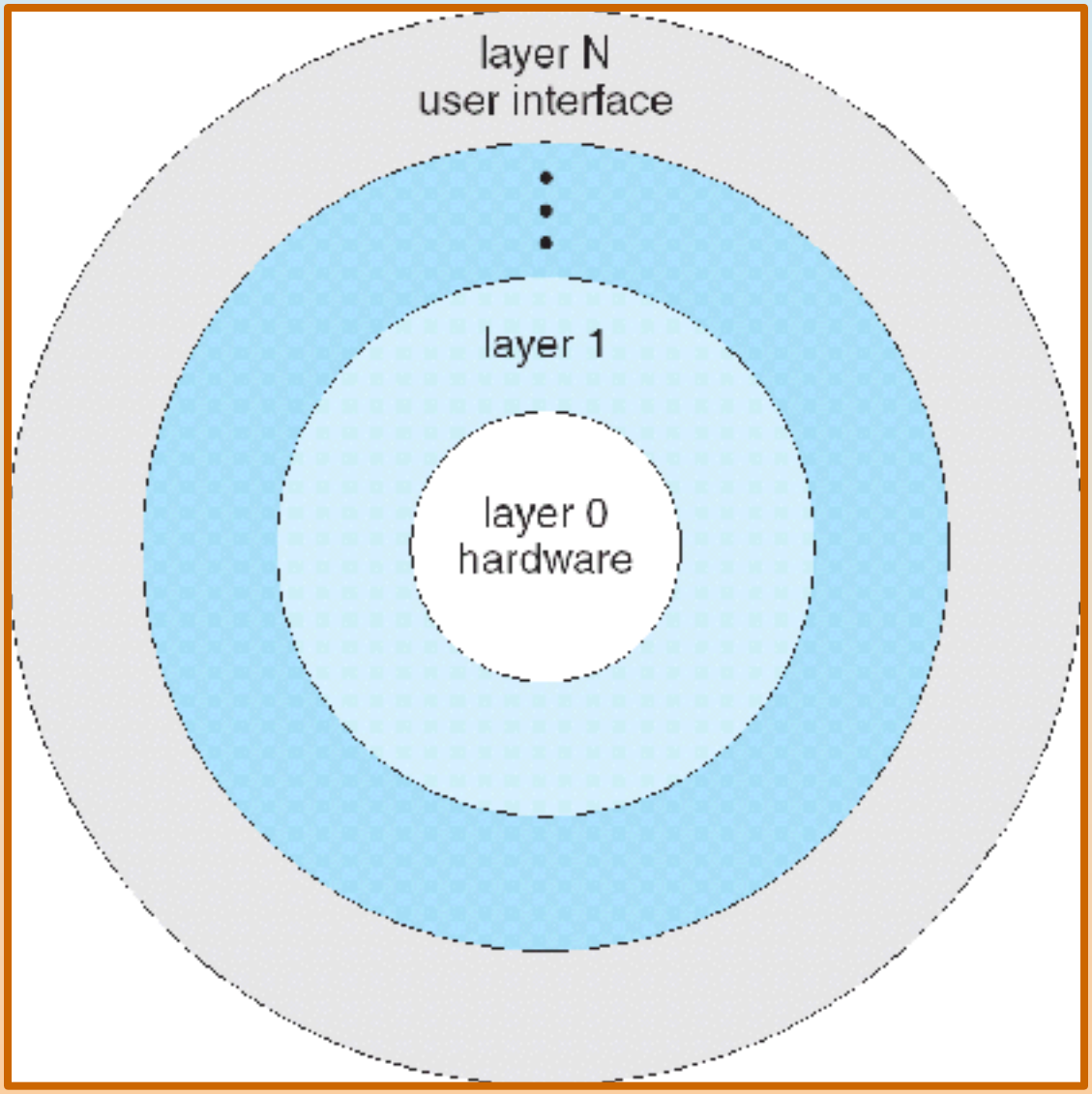
UNIX System Structure



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

Layered Operating System

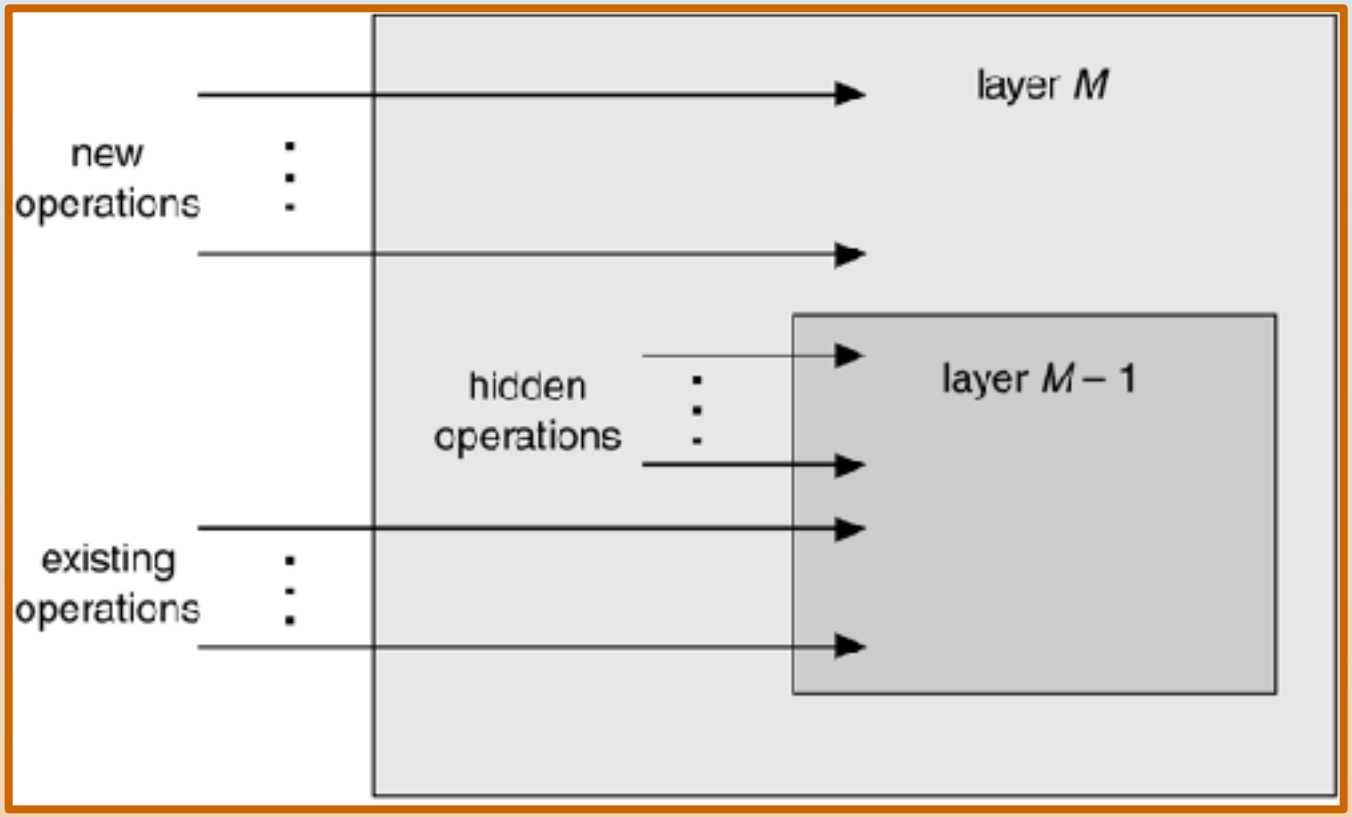


Layered Structure

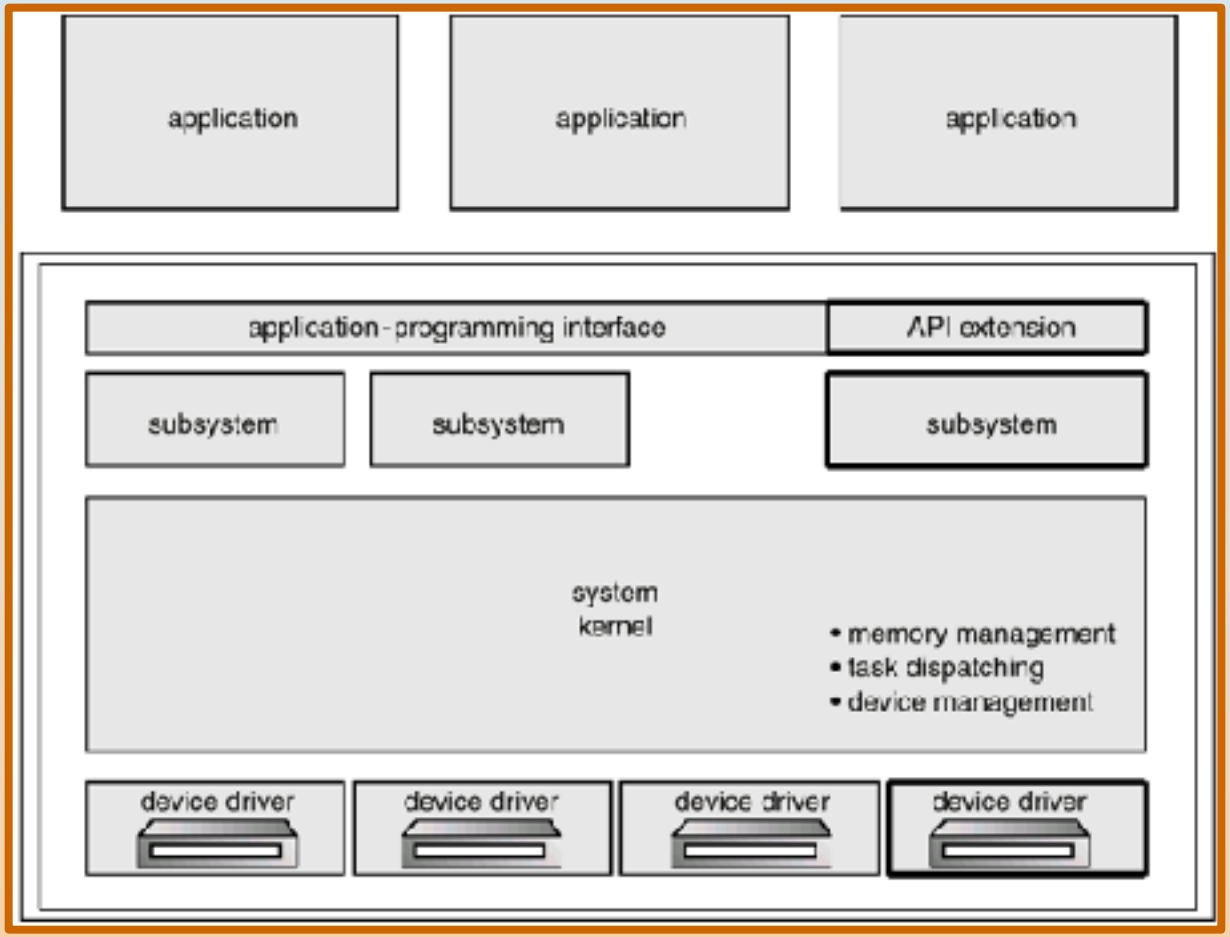
- Each layer uses functions (operations) and services of only lower-level layers
  - Advantage: modularity ⇒ Easier debugging/Maintenance
  - Not always possible: Does process scheduler lie above or below virtual memory layer?
    - Need to reschedule processor while waiting for paging
    - May need to page in information about tasks

- Machine-dependent vs independent layers
  - Easier migration between platforms
  - Easier evolution of hardware platform

An Operating System Layer



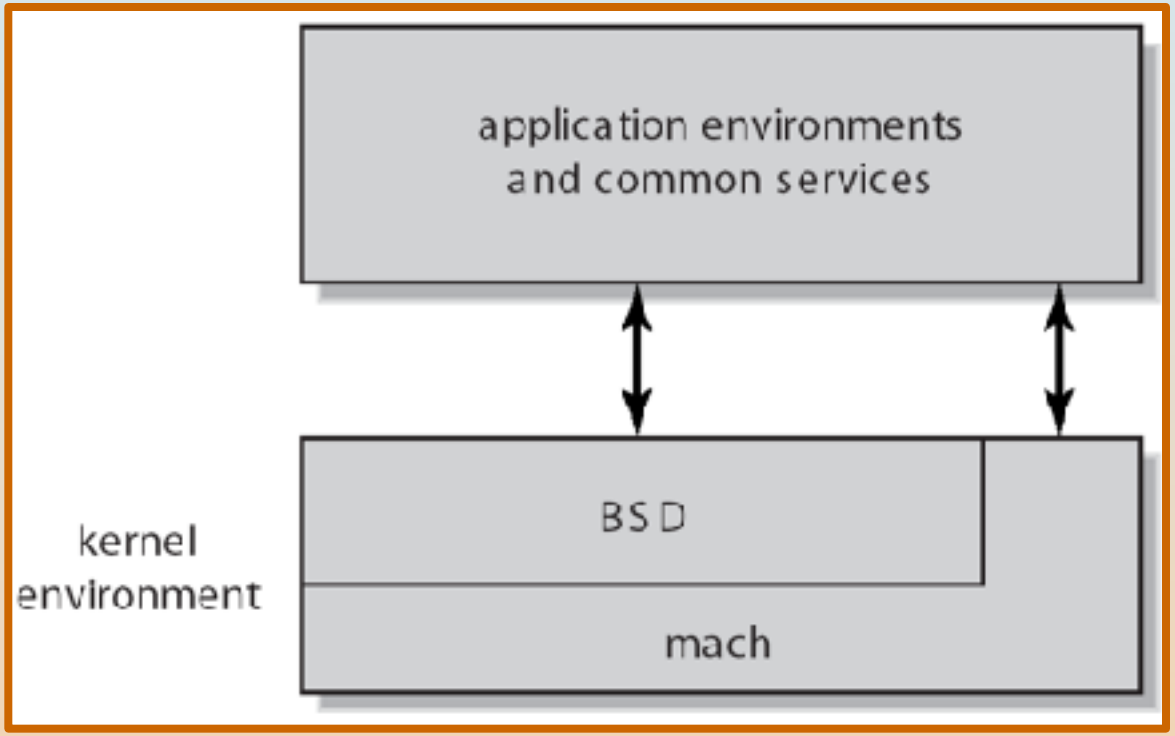
OS/2 Layer Structure



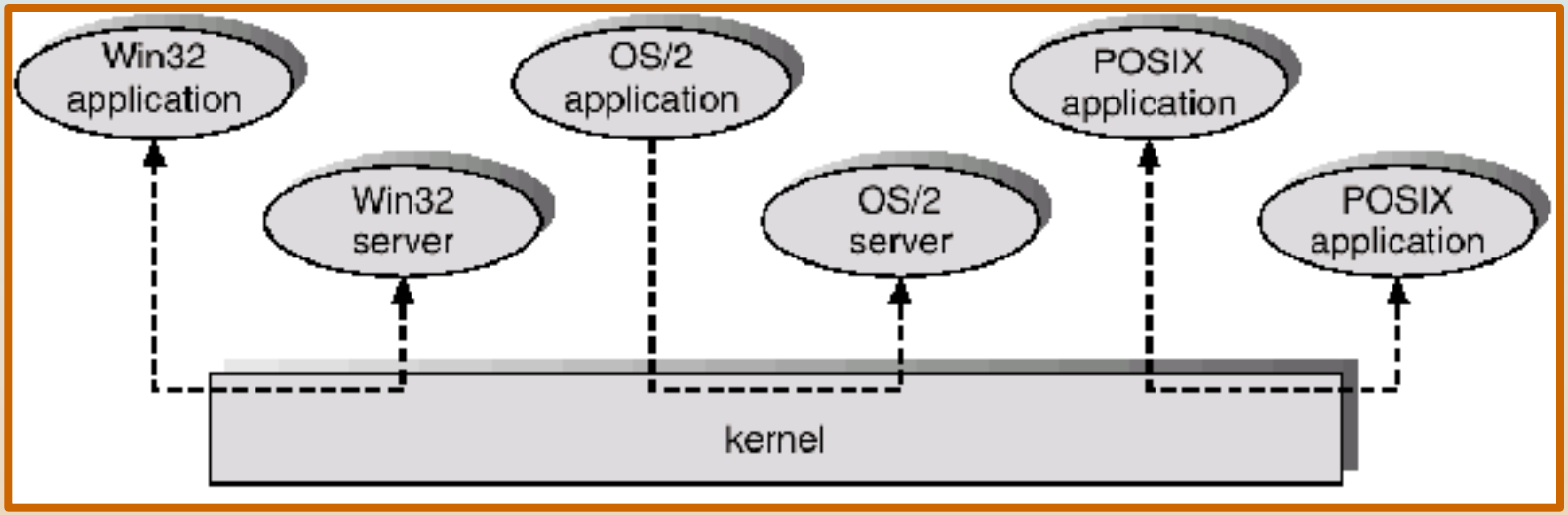
Microkernel System Structure

- Moves as much from the kernel into "*user*" space as possible
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Mac OS X Structure



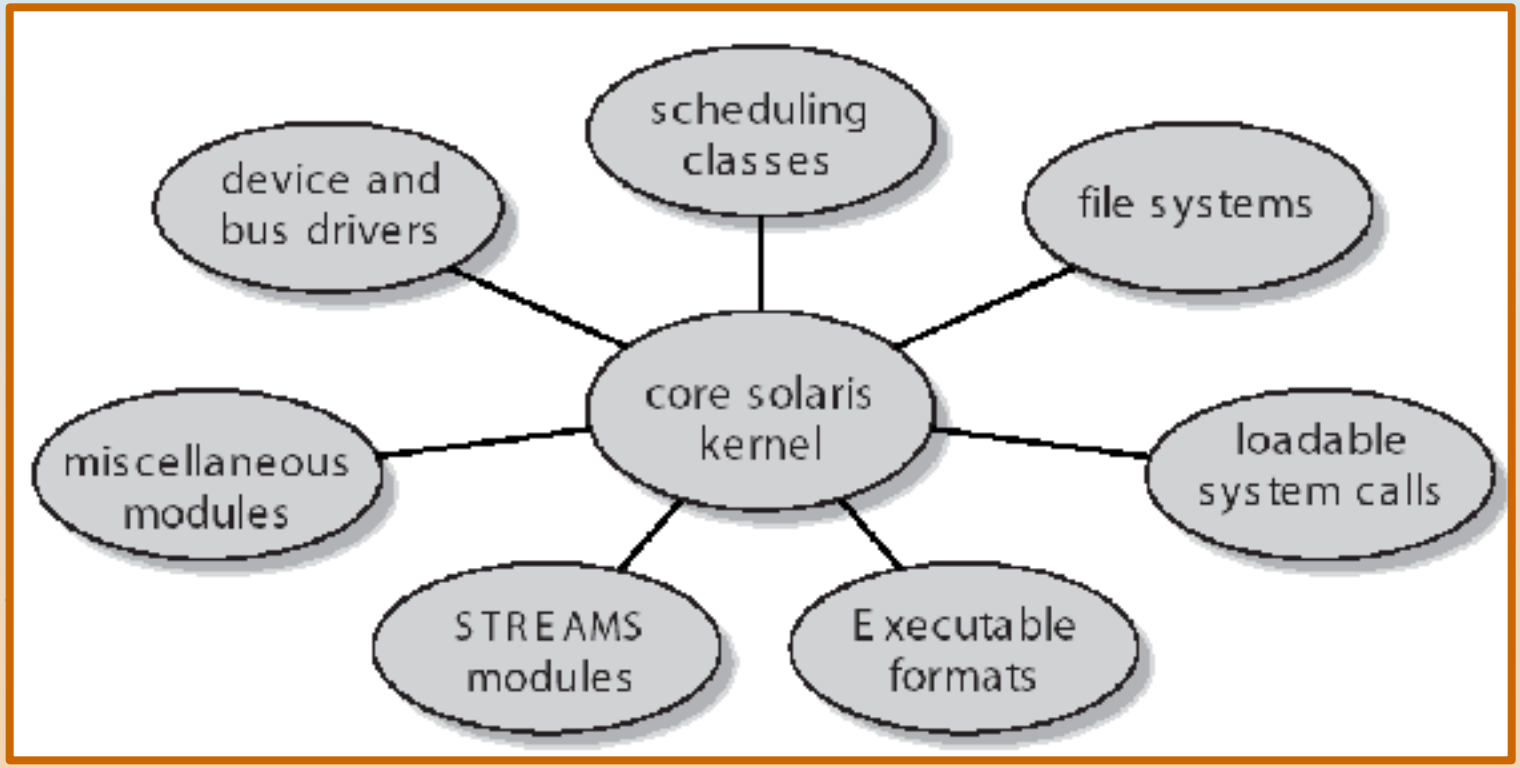
Windows NT Client-Server Structure



Module based

- Most modern operating systems implement kernel *modules*
  - 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 layers but with more flexible

Solaris Modular Approach



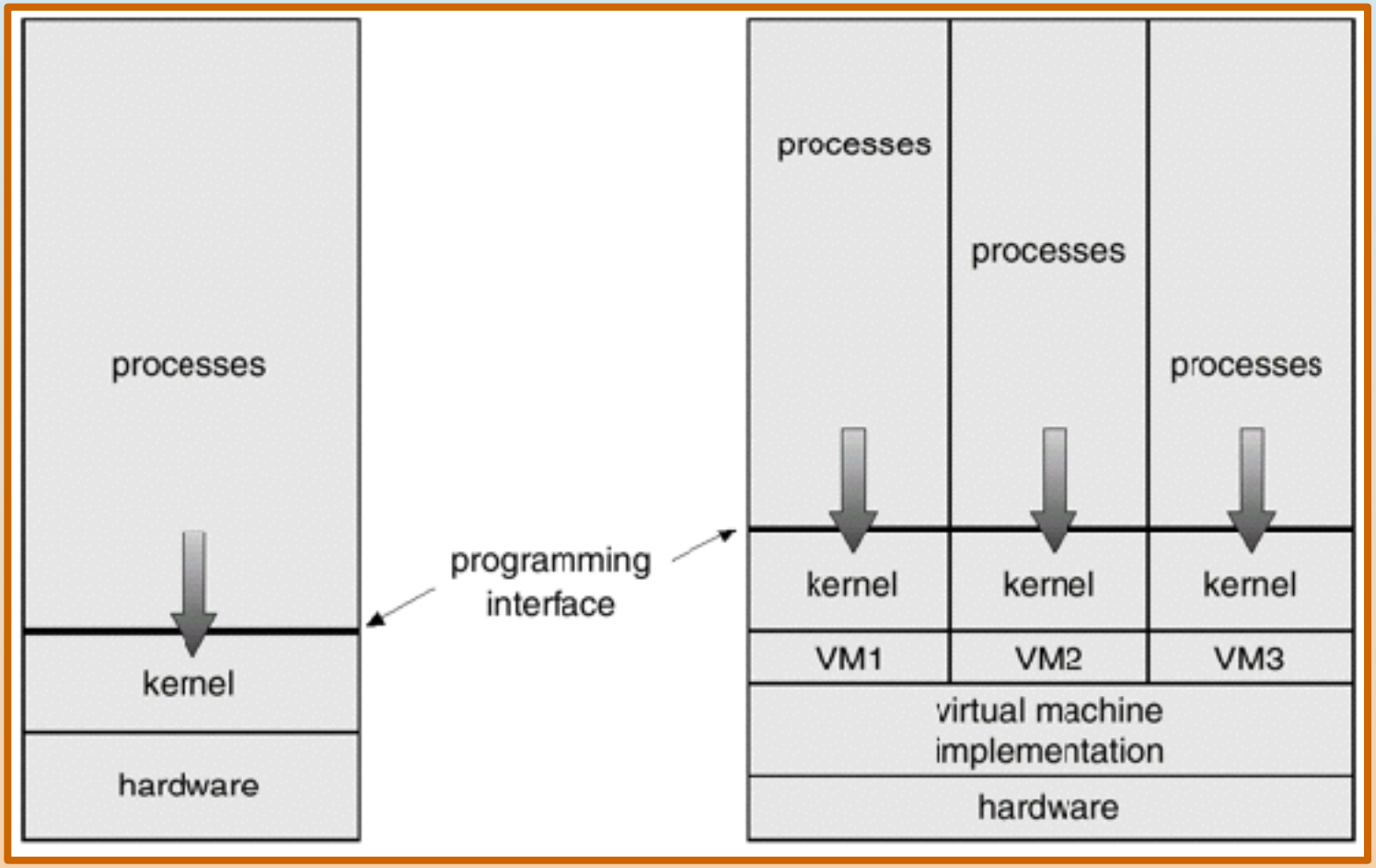
Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware

- A virtual machine provides an interface *identical* to the underlying bare hardware

- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

System Models



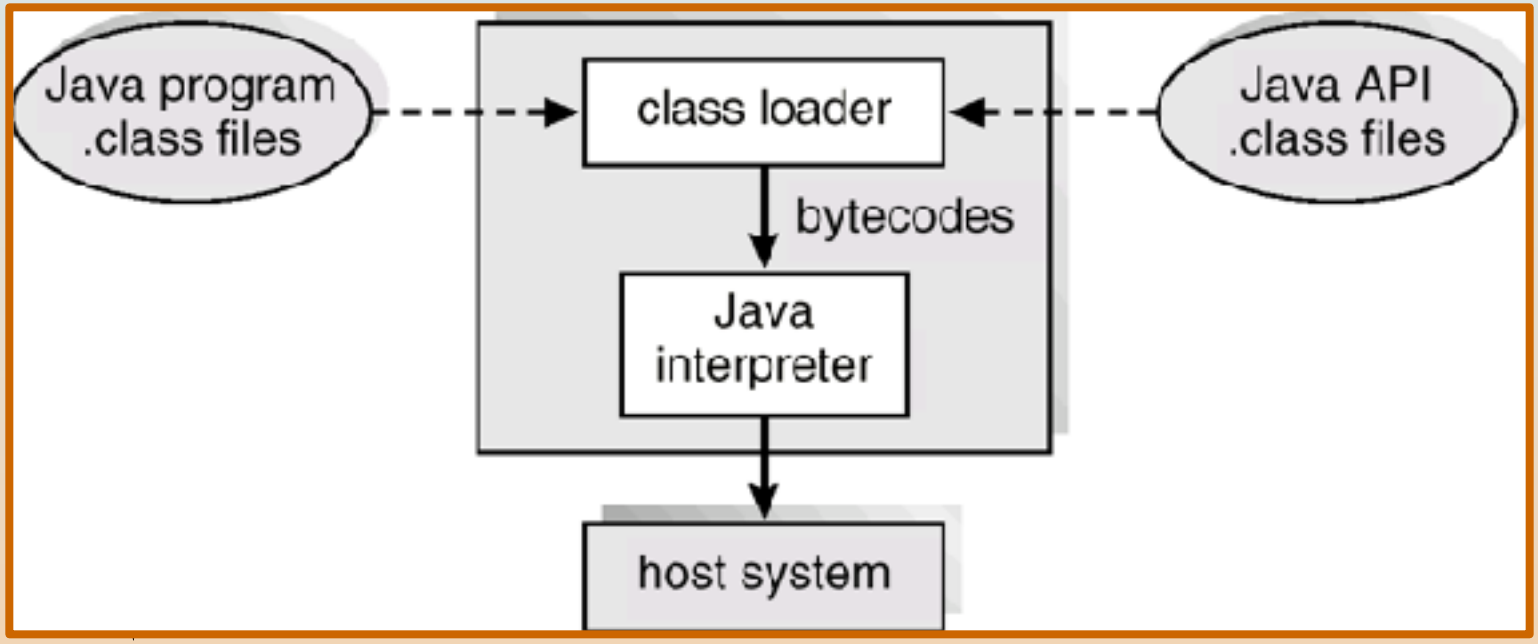
Non-virtual Machine

Virtual Machine

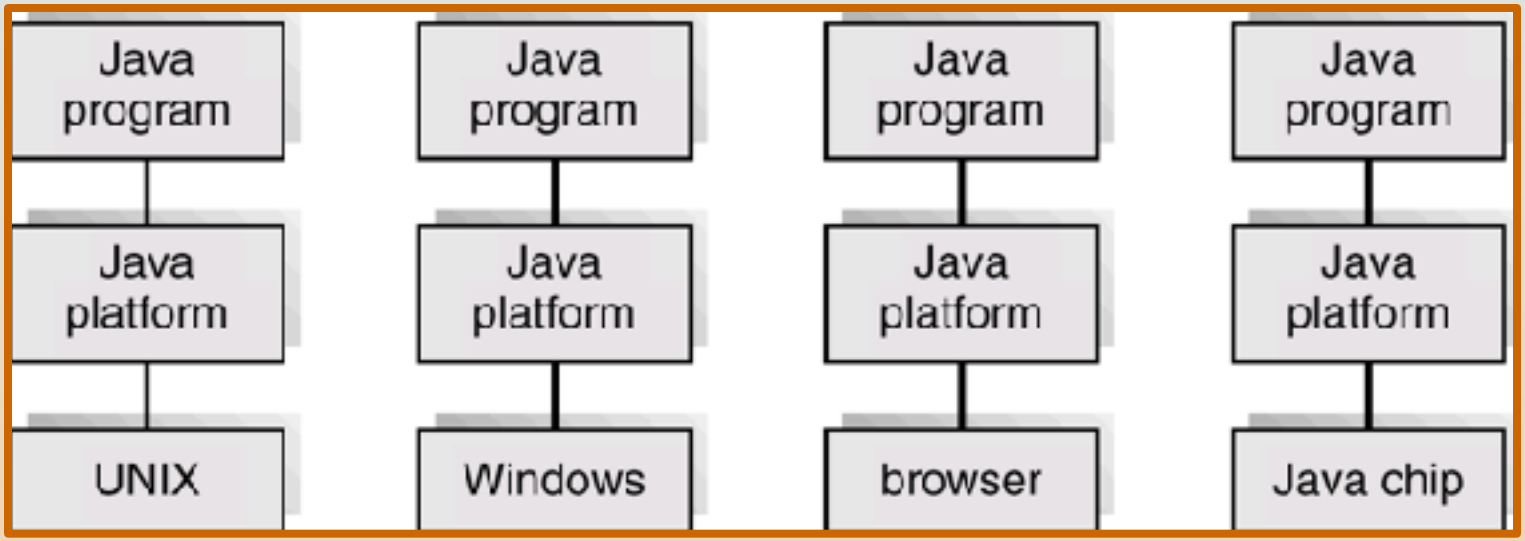
Advantages/Disadvantages of Virtual Machines

- Isolation from all other virtual machines.
- No disruption on normal system operation.
- Difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

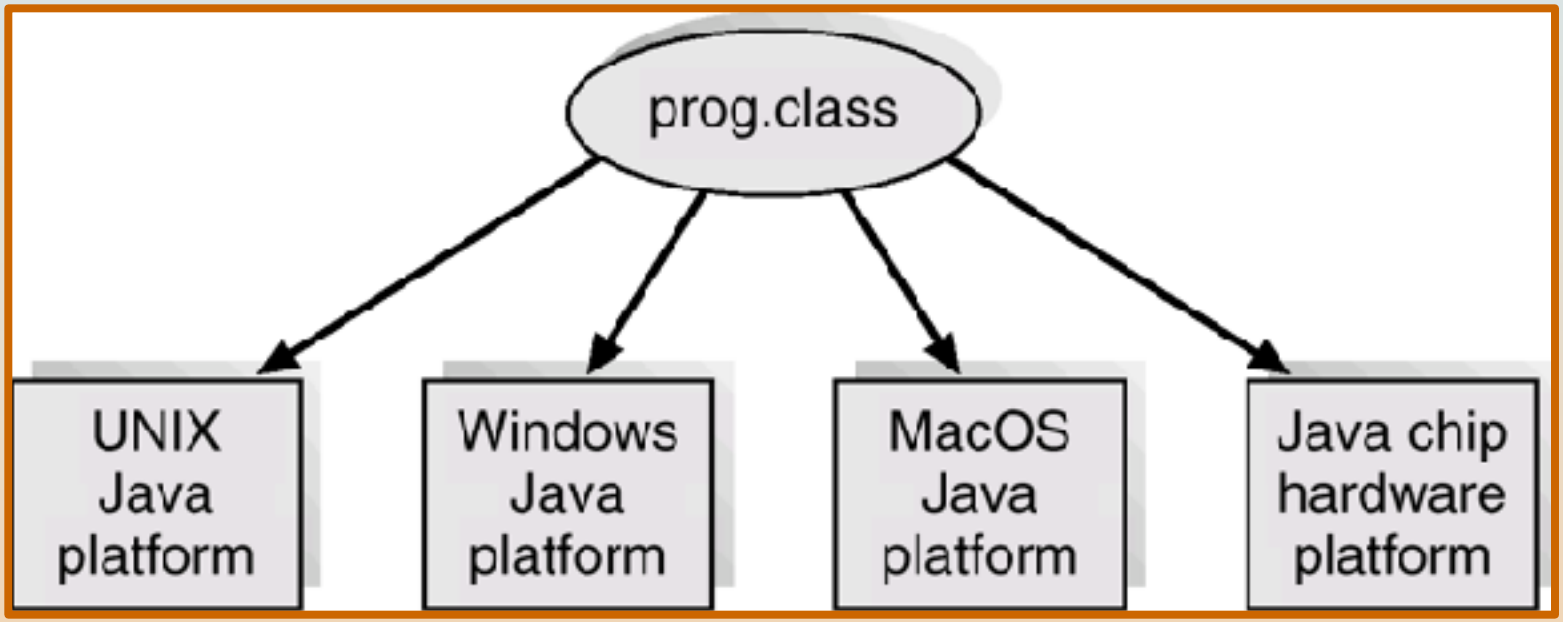
The Java Virtual Machine



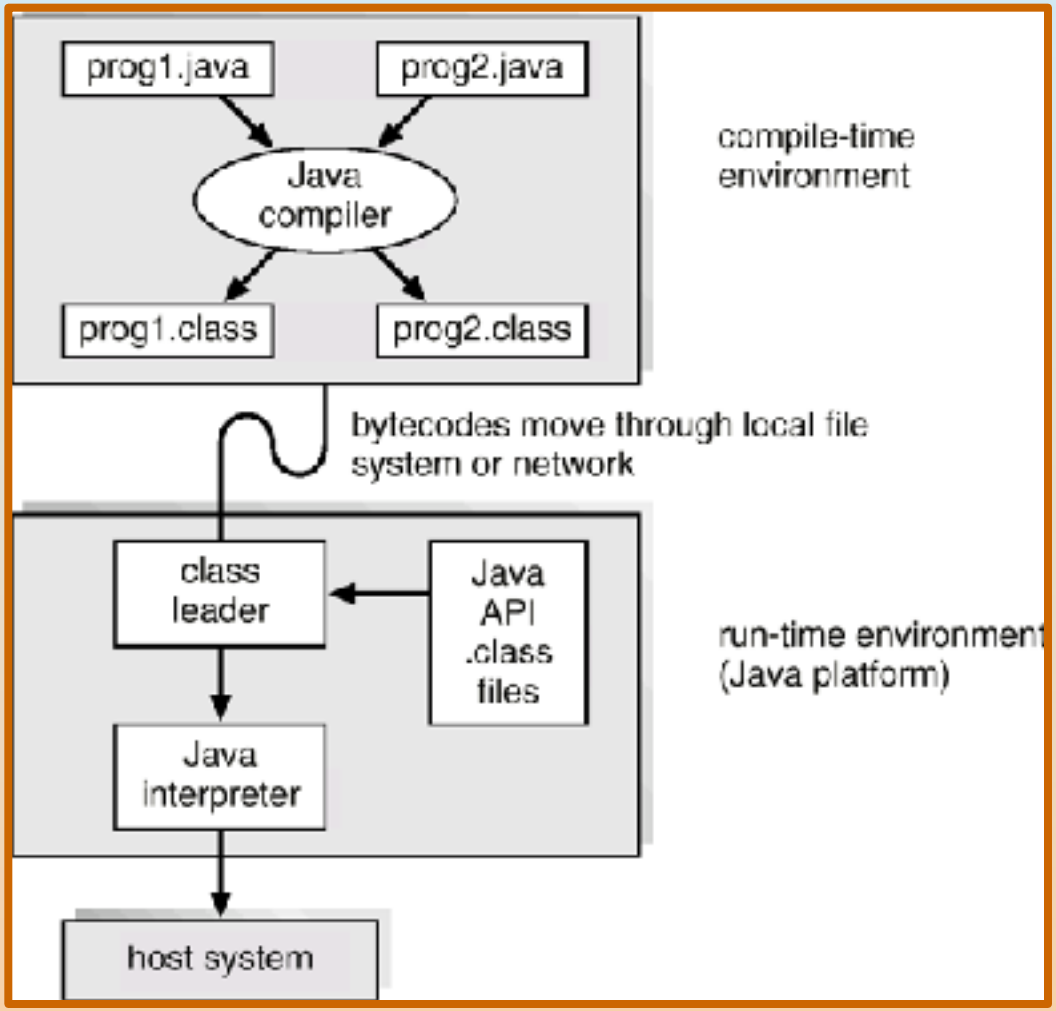
The Java Platform



Java .class File on Cross Platforms



Java Development Environment

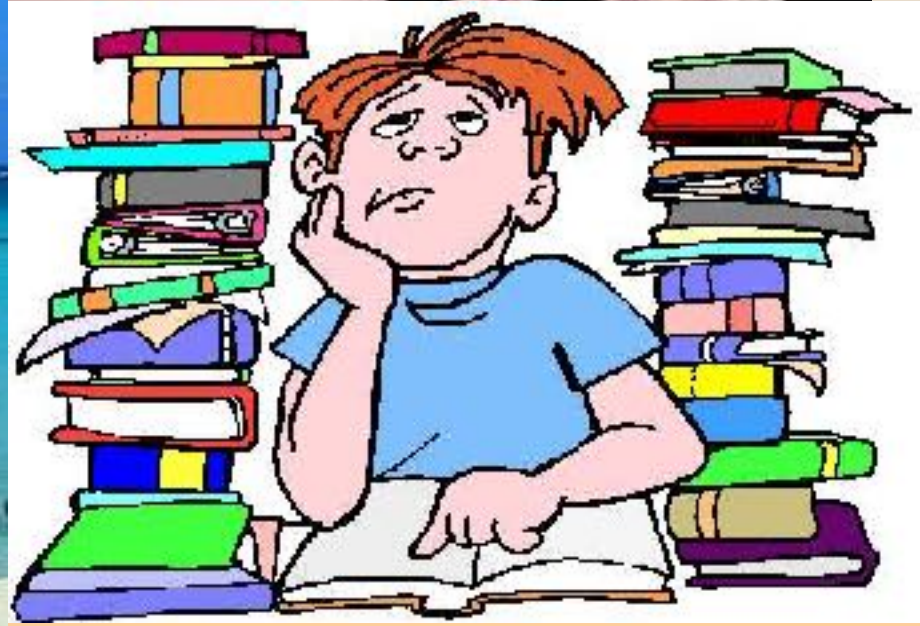


Implementation Issues

- Policy vs. Mechanism
  - Policy: What will be done?
  - Mechanism: How to do it?
  - Should be separated, since both change
- High-level language?
- Backward compatibility issues
  - Very important for Windows 2000/XP

Implementation Issues

- Algorithms used
  - Linear, Tree-based, Log Structured, etc…
- Event models used
  - threads vs event loops
- System generation/configuration
  - How to make generic OS fit on specific hardware
- Rapid Change in Hardware Leads to changing OS
  - Batch ⇒ Multiprogramming ⇒ Timeshare ⇒ Graphical UI ⇒ Ubiquitous Devices ⇒ Cyberspace/Metaverse/??



作业调度

- 作业集
- 作业发布时间, 耗费时间, 截止期,容易程度

作业集

作业名称	发布时间	耗费时间	距截止期剩余时间	容易程度
AI, paper	4 days ago	2.5 days	25 days left	* * * *
Game design, homework	today	4 day	20 days left	* * * * *
Stat Learning, project	2 days ago	6 days	20 days left	* *
Adv. Prog, midterm	yesterday	3 days	10 days left	*
Senior soft. & Engi., project	today	5 days	50 days left	* * *

调度策略

- First come first serve
- Easiest first
- Shortest time-cost first
- Earliest deadline first

First come first serve

作业名称	发布时间	耗费时间	距截止期剩余时间	容易程度
AI, paper	4 days ago		25 days left	
Game design, homework	today		20 days left	
Stat Learning, project	2 days ago		20 days left	
Adv. Prog., midterm	yesterday		10 days left	
Senior soft. & Engi., project	today		50 days left	

Easiest first

作业名称	发布时间	耗费时间	距截止期剩余时间	容易程度
AI, paper			25 days left	* * * *
Game design, homework			20 days left	* * * * *
Stat Learning, project			20 days left	* *
Adv. Prog, midterm			10 days left	*
Senior soft. & Engi., project			50 days left	* * *

Shortest time-cost first

作业名称	发布时间	耗费时间	距截止期剩余时间	容易程度
AI, paper		2.5 days	25 days left	
Game design, homework		4 day	20 days left	
Stat Learning, project		6 days	20 days left	
Adv. Prog, midterm		3 days	10 days left	
Senior soft. & Engi., project		5 days	50 days left	

Earliest deadline first

作业名称	发布时间	耗费时间	距截止期剩余时间	容易程度
AI, paper			25 days left	
Game design, homework			20 days left	
Stat Learning, project			20 days left	
Adv. Prog, midterm			10 days left	
Senior soft. & Engi., project			50 days left	

最优策略?

- First come first serve?
- Easiest first?
- Shortest time-cost first?
- Earliest deadline first?

如何判断策略是否可行?

- 可调度性
  - The tasks are schedulable if there exists a scheduling solution such that all the tasks can be scheduled to meet their deadlines

- 可调度性测试

- U =

$$\sum_{i=1}^{n} \frac{T_{cost}(i)}{T_{remains}(i)}$$

The tasks are schedulable if U <=1 !

可调度性

作业名称	发布时间	耗费时间	距截止期剩余时间	容易程度
AI, paper		2.5 days	25 days left	
Game design, homework		4 day	20 days left	
Stat Learning, project		6 days	20 days left	
Adv. Prog, midterm		3 days	10 days left	
Senior soft. & Engi., project		5 days	50 days left	

Earliest deadline first (EDF)

- Order jobs by deadline

- EDF is optimal
  - EDF can always produce a feasible schedule for a set of tasks if they are schedulable (U<=1).

Question

你的作业是可调度的么?

- Schedulability test result ?

延伸思考

- What about the overload case (U>1)?
- What if the objective is to minimize the sum of the lateness?
  - EDF does not seem to work