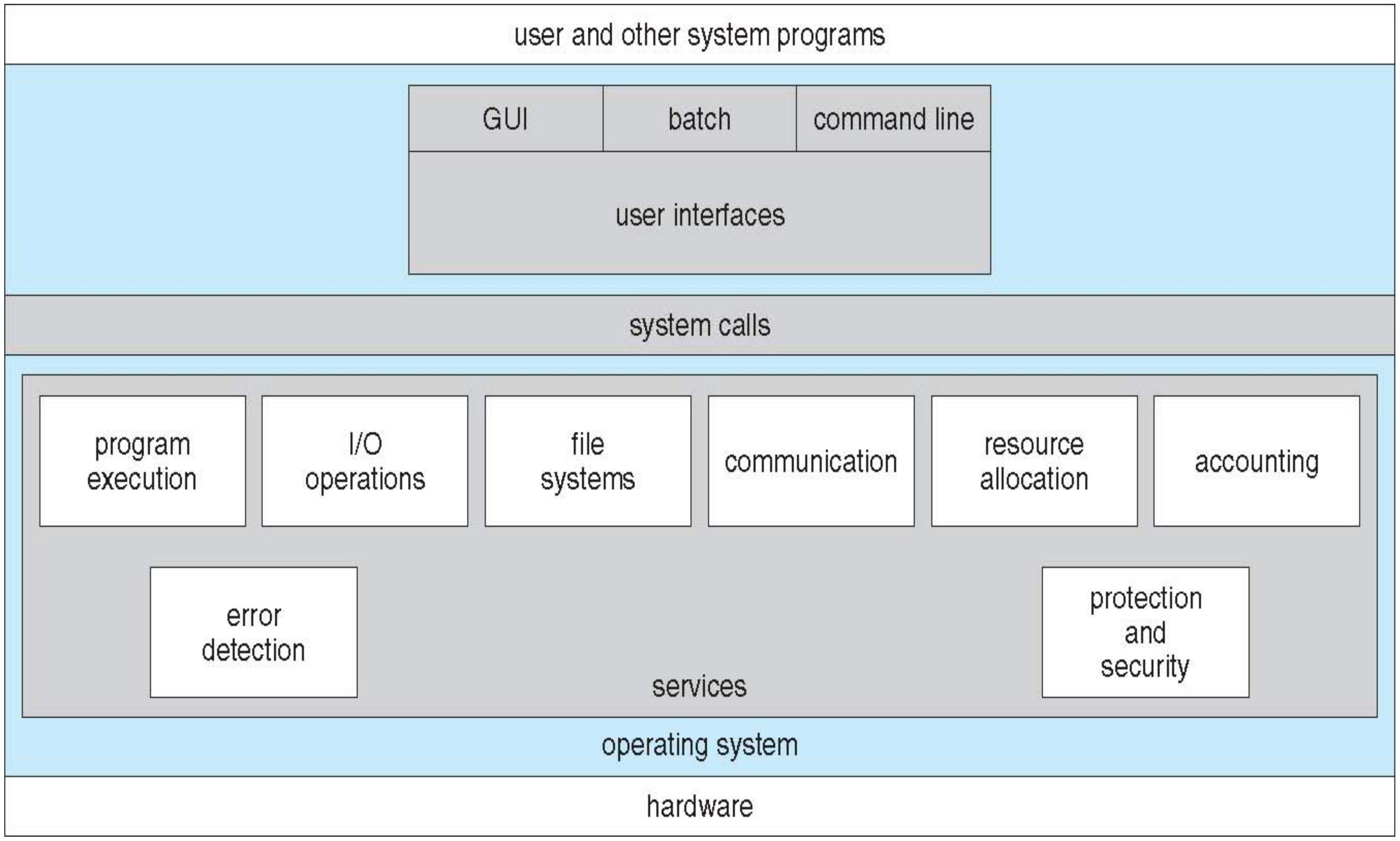
# Chapter 2:
# Operating System Structures

## Content

- Topics
  - Operating System Services
  - User Operating System Interface
  - System Calls, Types of System Calls, and System Programs
  - Operating System Design and Implementation
  - Operating System Structure
  - Virtual Machines
  - System Boot
- Objectives
  - To describe the services an operating system provides to users, processes, and other systems
  - To discuss the various ways of structuring an operating system
  - To explain how operating systems are installed and customized and how they boot

# A View of Operating System Services



## Operating System Services – User (1)

- One set of operating system services provides functions that are helpful to the user:
  - User interface: Almost all operating systems have a user interface (UI), which vary between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - Program execution: The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating errors)
  - I/O operations: A running program may require I/O, which may involve a file or an I/O device
  - File system manipulation: The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file information, permission management.

## Operating System Services – User (2)

- Communications: Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)
- Error detection: OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user’s and programmer’s abilities to efficiently use the system

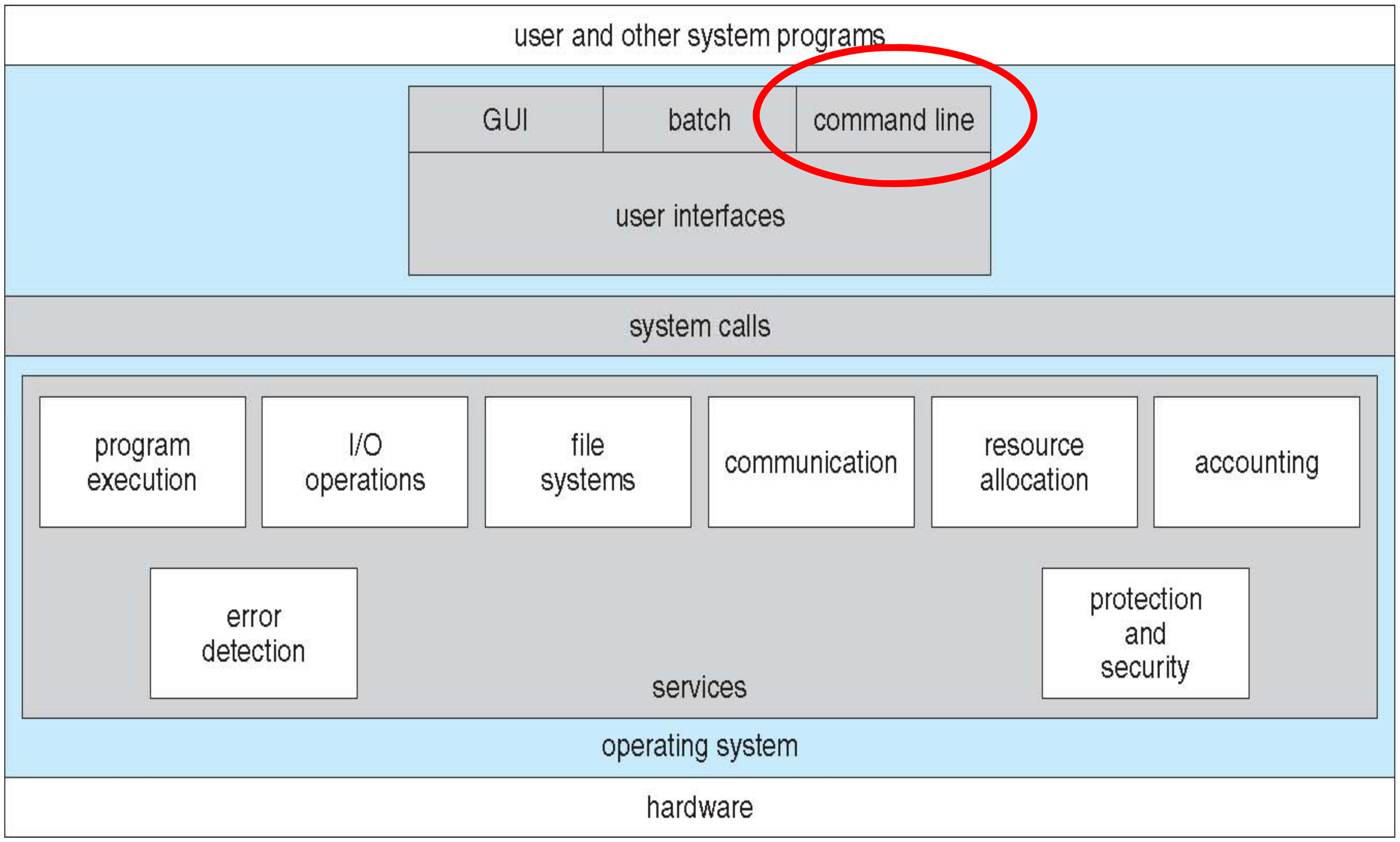
## Operating System Services – Operations (1)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing:
  - Resource allocation: When  multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources exist, some (such as CPU cycles, main memory, and file storage) may need special allocation code, others (such as I/O devices) may need general request and release code
  - Accounting: To keep track of which users use how much and what kinds of computer resources (“quota”)

## Operating System Services – Operations (2)

– Protection and security: The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - Protection involves ensuring that all access to system resources is controlled
  - Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
  - If a system is to be protected and secure, precautions must be instituted throughout it
    Note: A chain is only as strong as its weakest link!

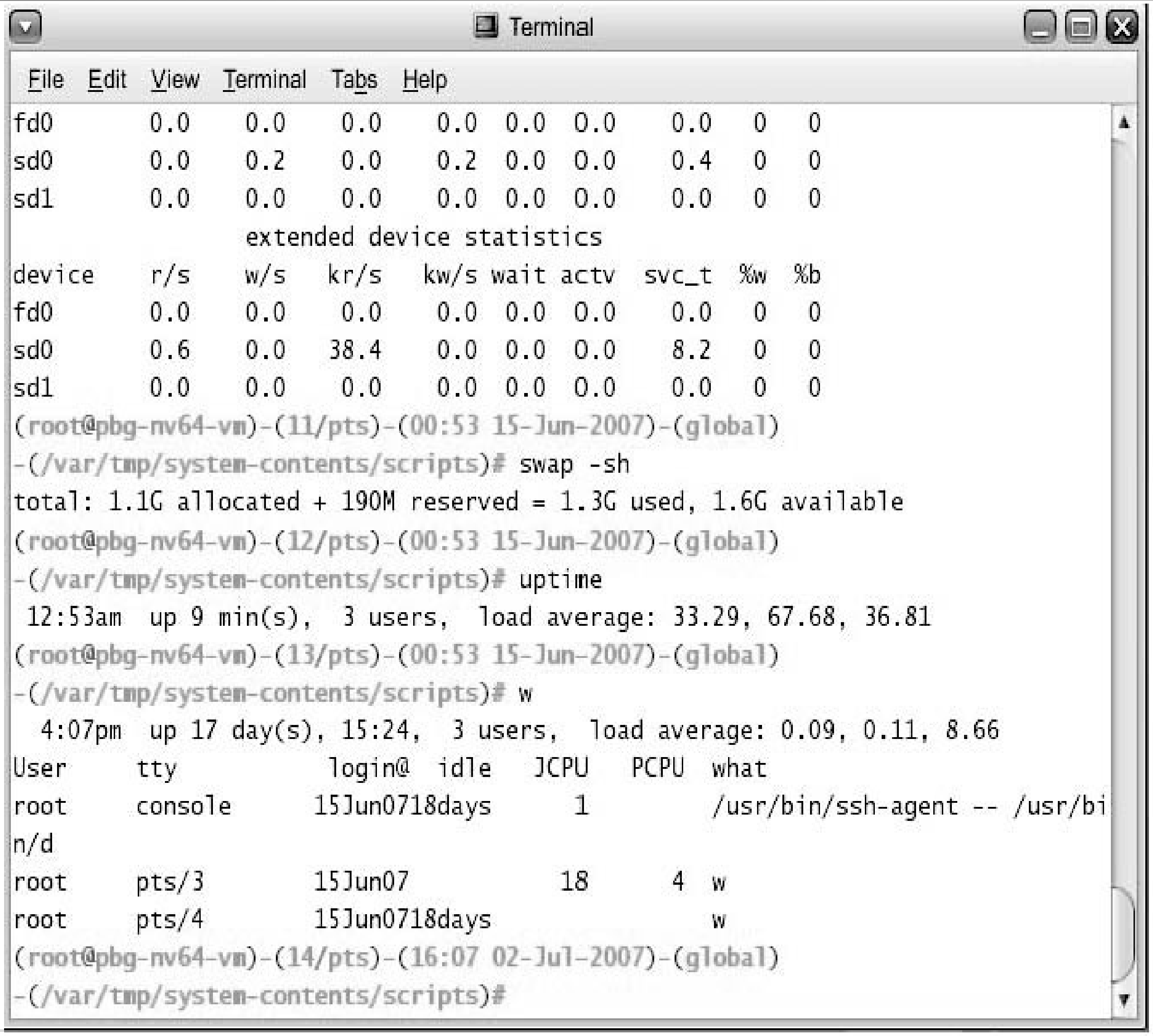
# A View of Operating System Services



## User Operating System Interface – CLI (1)

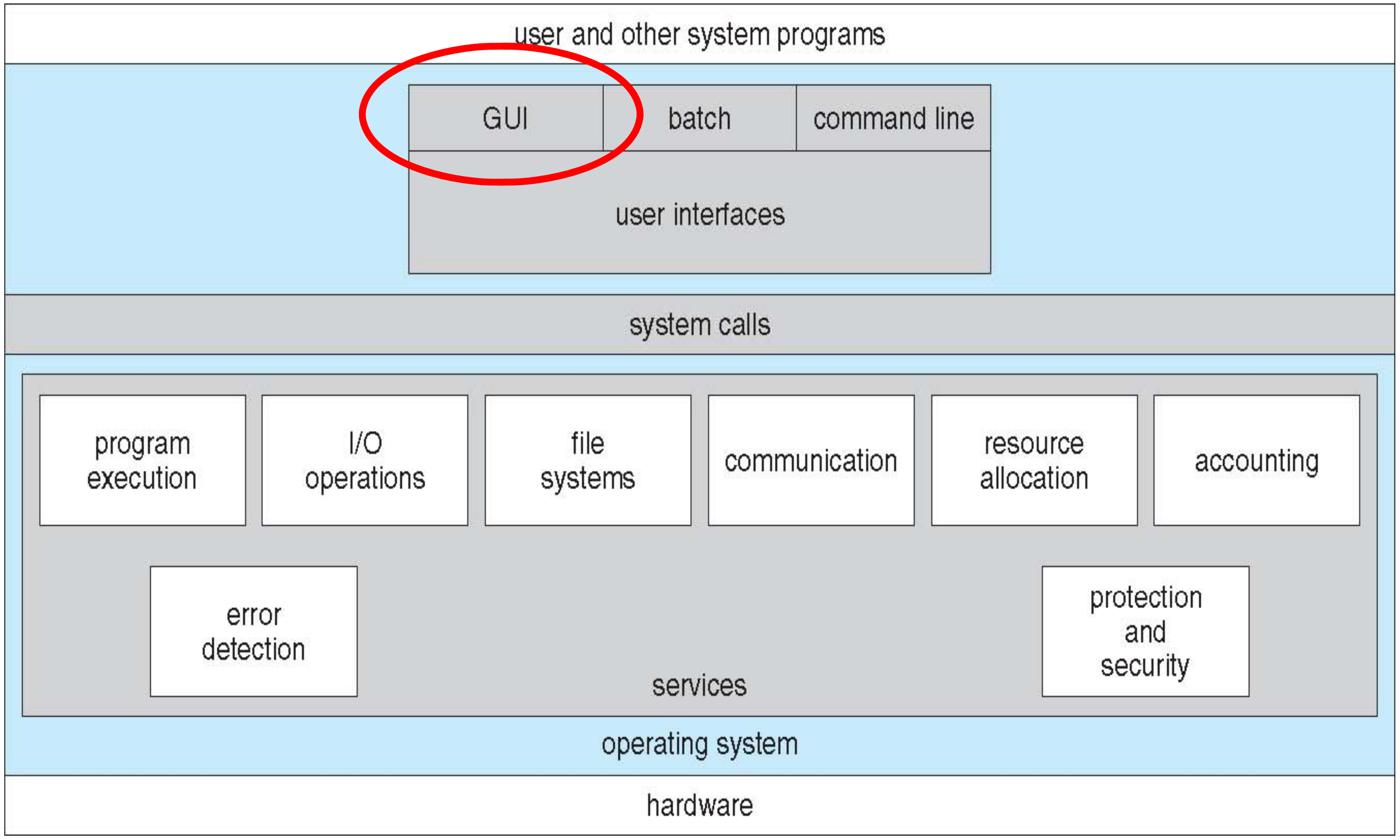
- Command Line Interface (CLI) or command interpreter allows direct command entry
  - Sometimes in past OSes CLIs are implemented in the kernel, sometimes by systems program, today, CLIs are accessing the kernel functionality via the user space
  - Sometimes multiple implementation flavors available – shells
  - Primarily fetches a command from a user and executes it
    - Sometimes commands are built-in, sometimes commands are just names of programs
      - If the latter, adding new features doesn't require a shell modification

# User Operating System Interface – CLI (2)

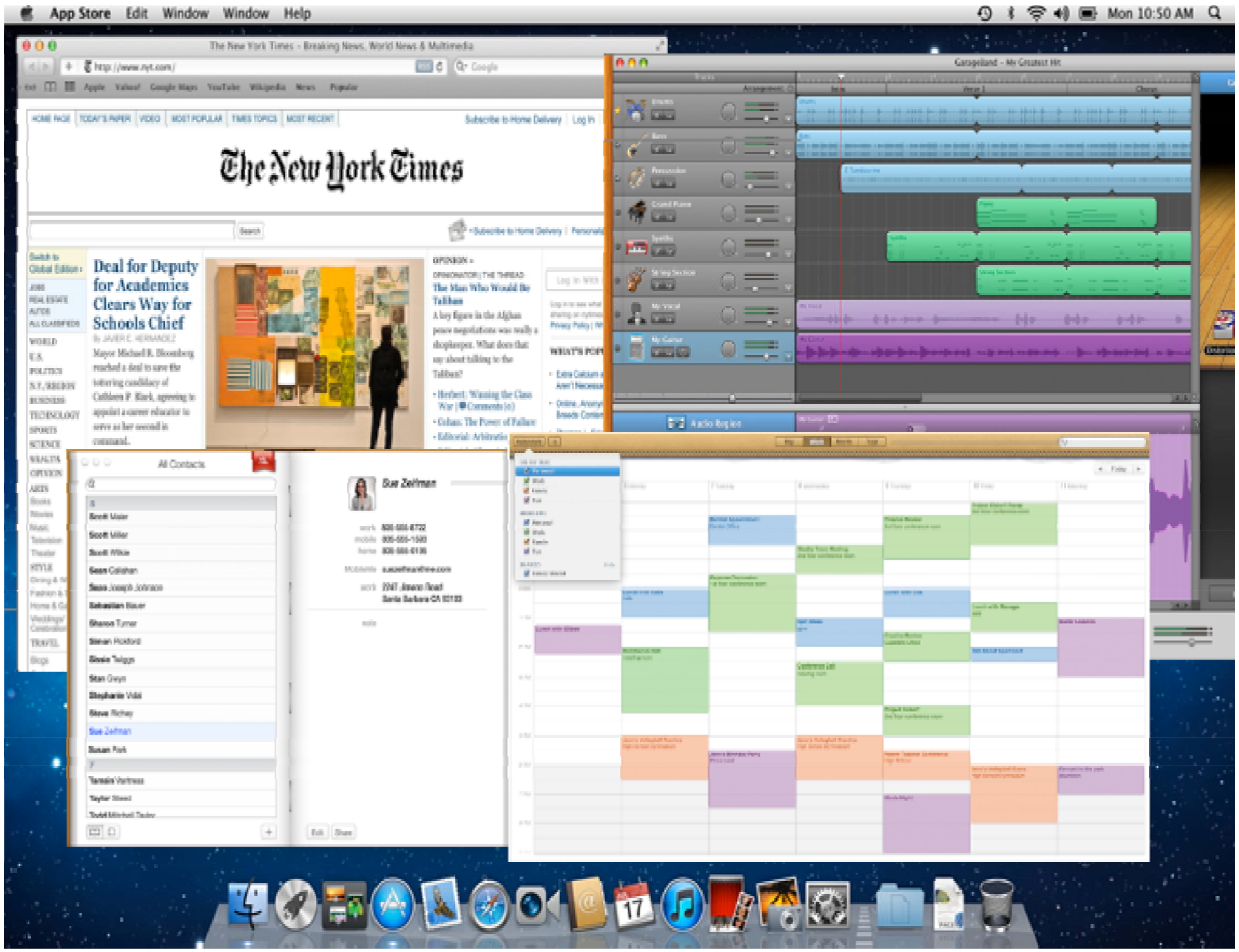


```
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.0    0.2    0.0    0.2  0.0  0.0    0.4   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                 extended device statistics
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.6    0.0   38.4    0.0  0.0  0.0    8.2   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07           18      4  w
root     pts/4        15Jun0718days                w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

# A View of Operating System Services

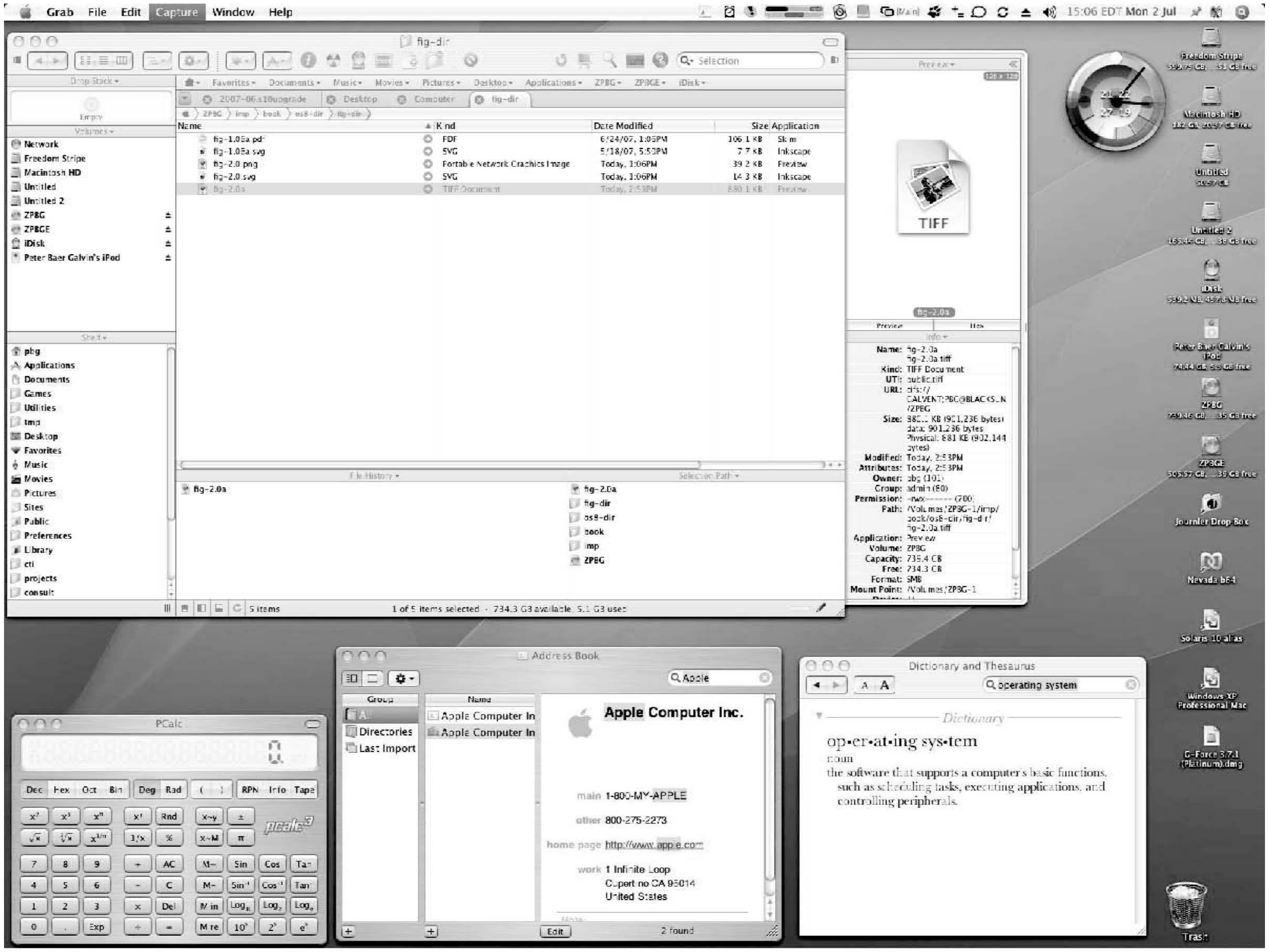


# User Operating System Interface – GUI (1)


App Store Edit Window Window Help
Mon 10:50 AM
The New York Times - Breaking News, World News & Multimedia
The New York Times
Deal for Deputy for Academies Clears Way for Schools Chief
GarageBand - My Greatest Hit
All Contacts
Sue Zeifman

# User Operating System Interface – GUI (2)

- Mac OS X



Apple Computer Inc.

main 1-800-MY-APPLE

other 800-275-2273

home page http://www.apple.com

work 1 Infinite Loop
Cupertino CA 95014
United States

2 found

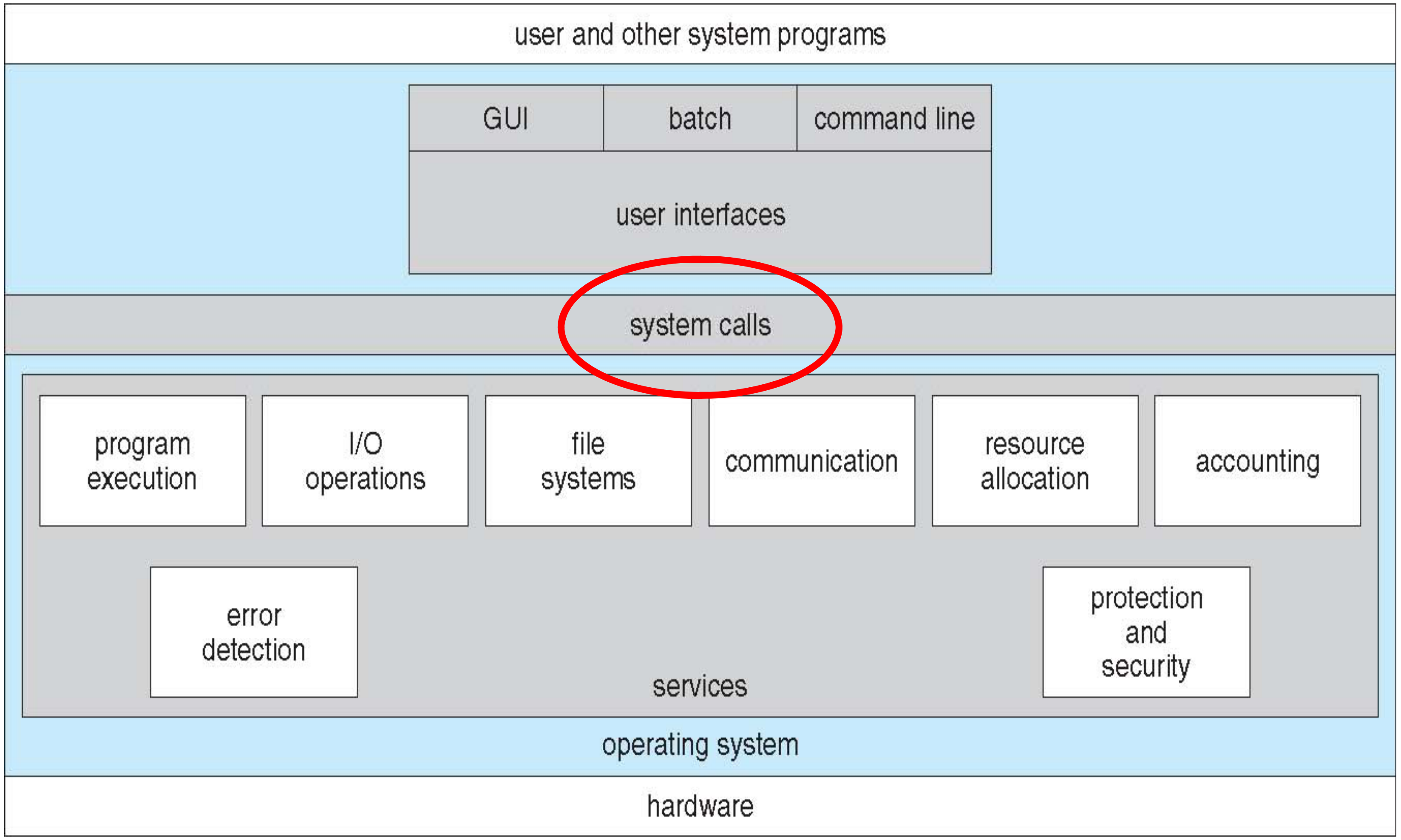
op•er•at•ing sys•tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

1 of 5 items selected · 734.3 GB available, 5.1 GB used

## User Operating System Interface – GUI (3)

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions
  - Various mouse buttons over objects in the interface cause various actions, such as providing information, options, executing function, opening a directory (known as a folder)
- Most systems now include both CLI and GUI interfaces
  - Microsoft Windows runs as a GUI with CLI “command” shell
  - Apple Mac OS X utilizes the “Aqua” GUI interface with a UNIX kernel underneath and shells available
  - Solaris (and Linux) is CLI-based with optional GUI interfaces (such as Java Desktop or KDE)

# A View of Operating System Services



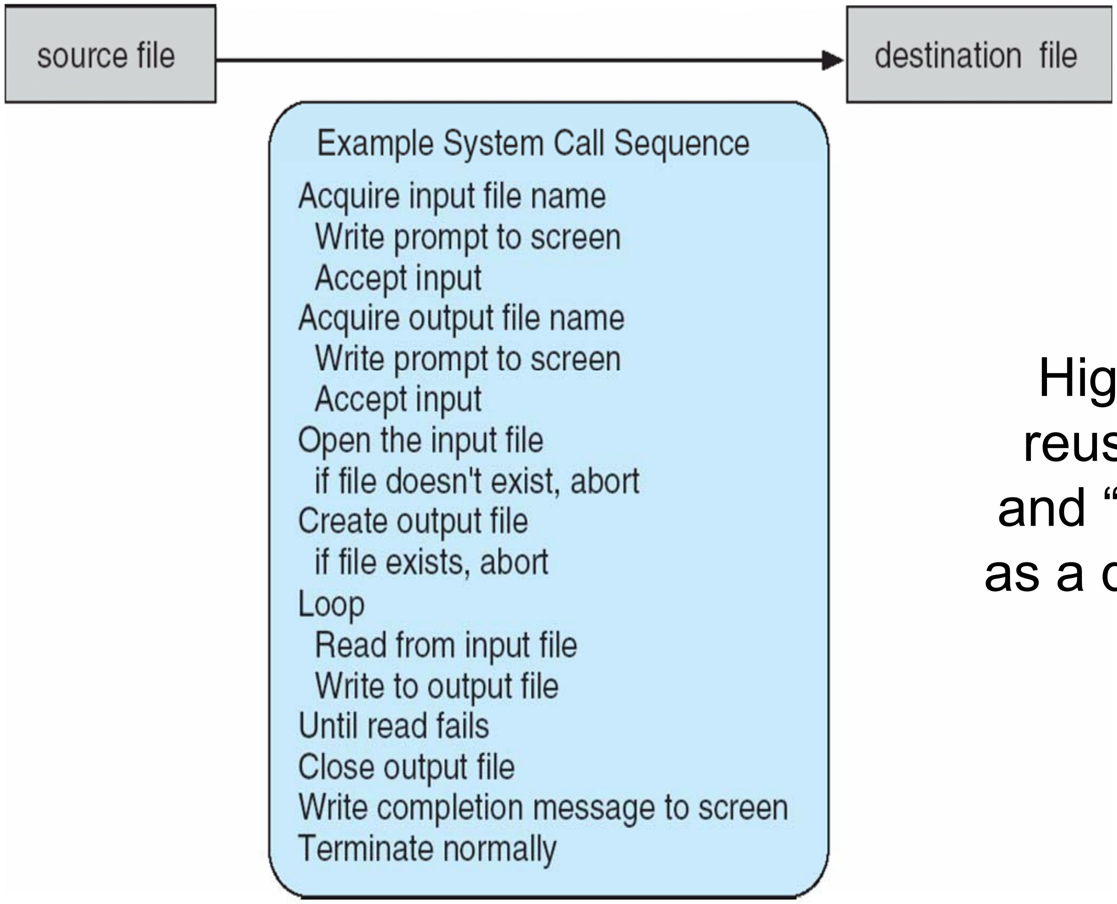
## System Calls

- Programming interface to the services provided by the OS
  - Typically written in a high-level system language (C or C++)
  - Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use

- Three most common APIs are Windows API, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

# Example of System Calls

- A system call sequence “to copy the content of one file to another file”:

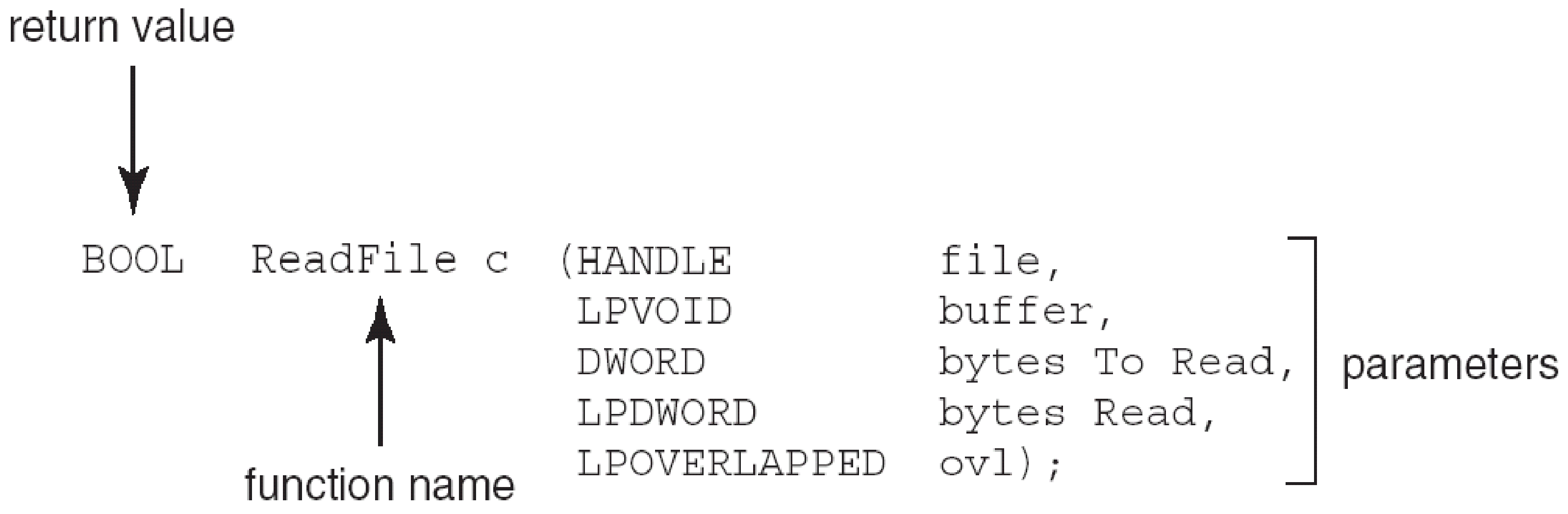


```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Highly modular, reusable “steps”, and “small” actions as a dedicated task!

## Example of a Standard API (1)

- Consider the ReadFile() function in the Win32 API
  - A function for reading from a file



```
BOOL   ReadFile c  (HANDLE         file,
                    LPVOID         buffer,
                    DWORD          bytes To Read,
                    LPDWORD        bytes Read,
                    LPOVERLAPPED   ovl);
```

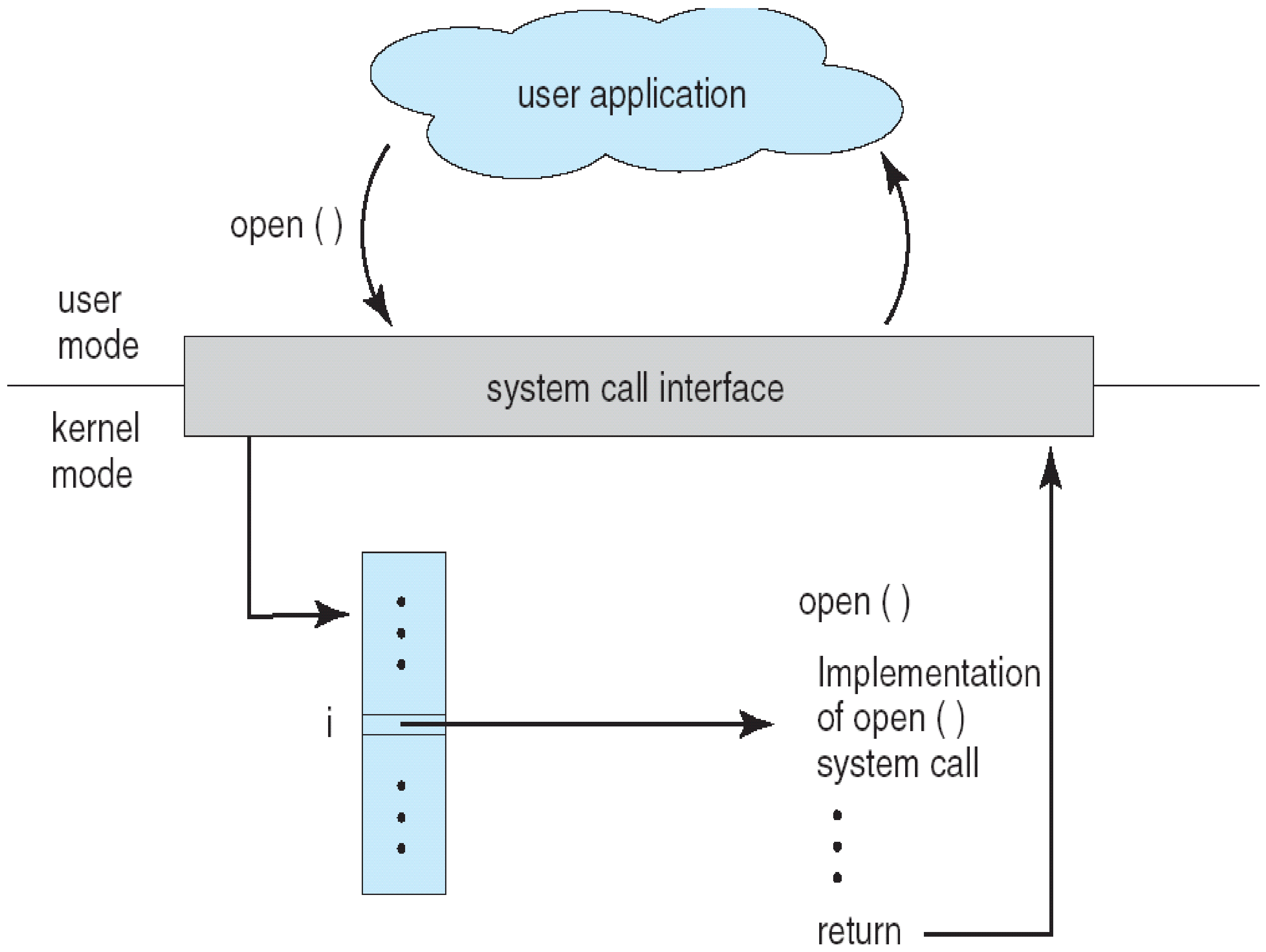
## Example of a Standard API (2)

- A description of the parameters passed to ReadFile()
  - HANDLE file — the file to be read
  - LPVOID buffer — a buffer where the data will be read into and written from
  - DWORD bytesToRead — the number of bytes to be read into the buffer
  - LPDWORD bytesRead — the number of bytes read during the last read
  - LPOVERLAPPED ovl — indicates if overlapped I/O is being used

## System Call Implementation

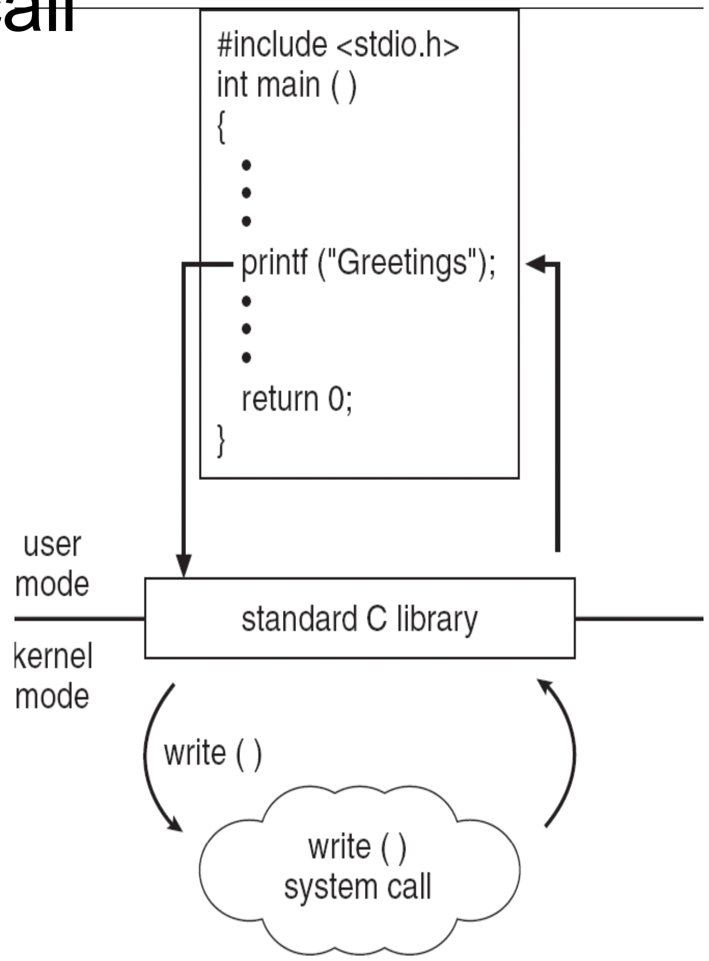
- Typically, a number is associated with each system call
  - System call interface maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel
  - Returns status of the system call and any return values
- The caller needs to know nothing about how the system call is implemented
  - Just needs to obey the API and understand what the OS will do as a result of the call
  - Most details of OS interfaces hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

## API – System Call: OS Relationship



# Standard C Library Example

- C program invoking printf() library call, which calls the write() system call


```
#include <stdio.h>
int main ( )
{
  •
  •
  •
  printf ("Greetings");
  •
  •
  •
  return 0;
}
```

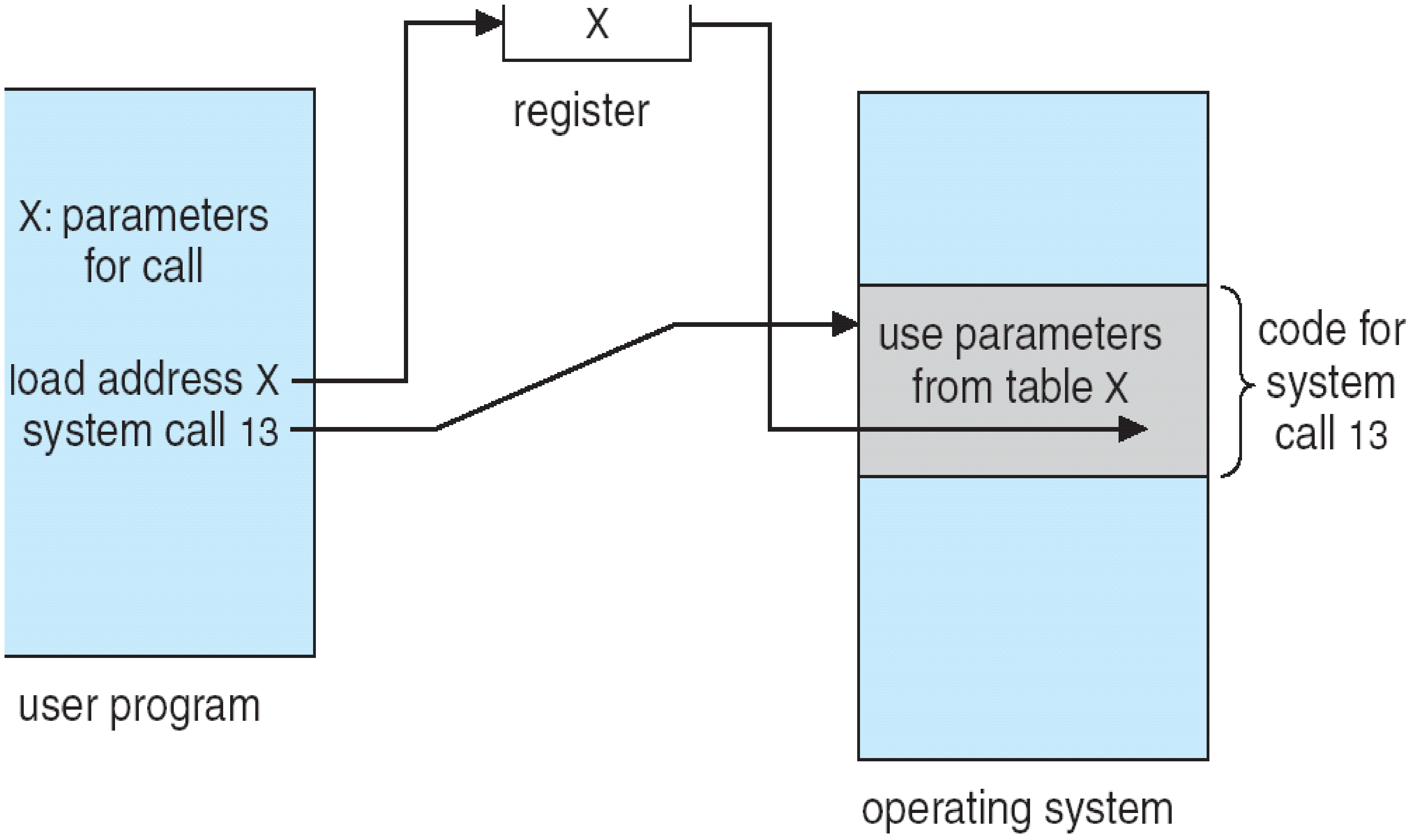
## System Call Parameter Passing (1)

- Often, more information is required than simply to identify a desired system call
  - Exact type and amount of information vary according to OS and call

## System Call Parameter Passing (2)

- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

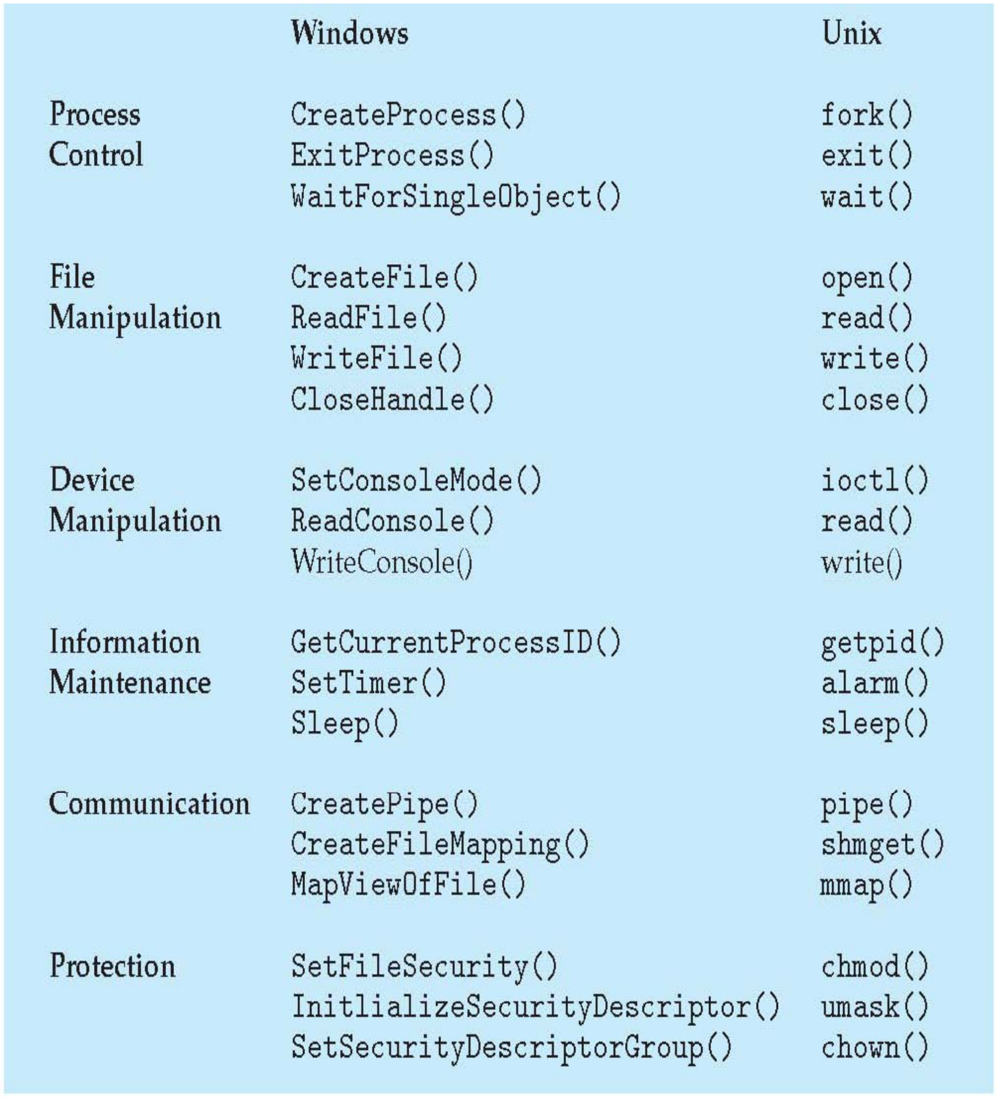
# Parameter Passing via Table



## Types of System Calls

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

# Examples of Windows and Unix System Calls



| | Windows | Unix |
|---|---|---|
| Process Control | CreateProcess() | fork() |
| | ExitProcess() | exit() |
| | WaitForSingleObject() | wait() |
| File Manipulation | CreateFile() | open() |
| | ReadFile() | read() |
| | WriteFile() | write() |
| | CloseHandle() | close() |
| Device Manipulation | SetConsoleMode() | ioctl() |
| | ReadConsole() | read() |
| | WriteConsole() | write() |
| Information Maintenance | GetCurrentProcessID() | getpid() |
| | SetTimer() | alarm() |
| | Sleep() | sleep() |
| Communication | CreatePipe() | pipe() |
| | CreateFileMapping() | shmget() |
| | MapViewOfFile() | mmap() |
| Protection | SetFileSecurity() | chmod() |
| | InitlializeSecurityDescriptor() | umask() |
| | SetSecurityDescriptorGroup() | chown() |

## System Programs (1)

- System programs provide a convenient environment for program development and execution
- They can be divided into:
  - File manipulation
    - Create, delete, copy, print
  - Status information
    - Date, time, amount of available memory, disk space, number of users
  - File modification
    - Rename, edit
  - Programming language support
    - Compilers, assemblers, debuggers and interpreters

## System Programs (2)

- Program loading and execution
- Communications
  - Send messages, browse web pages, send electronic-mail, transfer files
- Application programs
  - Word, Acrobat, Photoshop

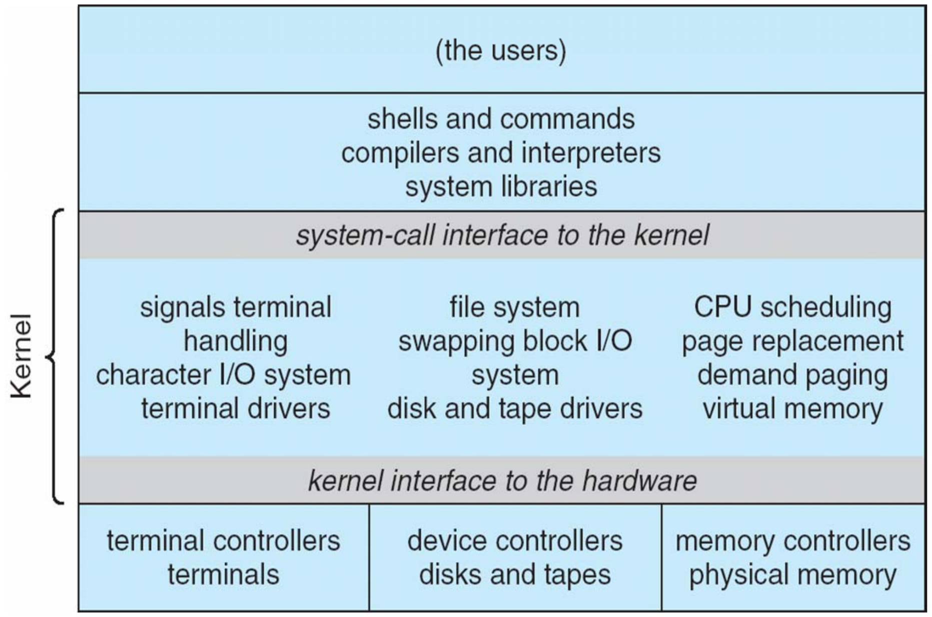
- Some of them are simply user interfaces to system calls; others are considerably more complex

## Operating System Design: Implementation

- Best design and implementation of OS not “solvable”, but some approaches have proven successful
  - Internal structure of different Operating Systems can vary widely
  - Start by defining goals and specifications
  - Affected by choice of hardware, type of system
- User goals and system goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

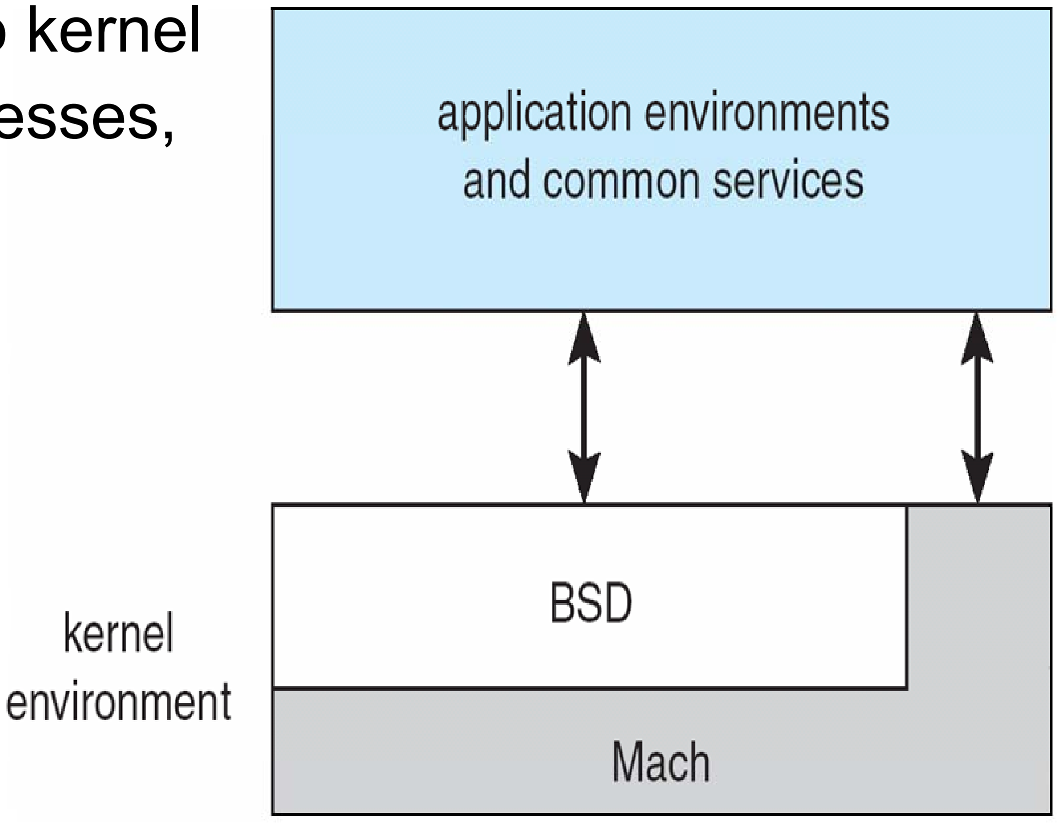
## UNIX – Monolithic Kernels

- Limited by hardware functionality, the original UNIX opera-ting system had limited structuring. UNIX OS consists of:
  - Systems programs
  - Monolithic kernel
    - Consists of everything below the system-call interface and above physical hardware
    - Single large process encompassing all services in a single kernel address space
    - Provides the file system, CPU scheduling, memory management, and other operating system functions; a large number of functions for one level
  - Examples: Unix/BSD, Linux, MS-DOS/earlier



# Micro Kernel System Structure (1)

- Moves as much from the kernel into “user” space
  - Device drivers, protocol stacks, and file systems are removed from the micro kernel to run as separate processes, servers
  - Often in user space
- Communication takes place between micro kernels using message passing



BSD: Berkeley Software Distribution

## Micro Kernel System Structure (2)

- Benefits
  - Micro kernel easier to extend
  - 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 communications
- Mach and OS X use this approach

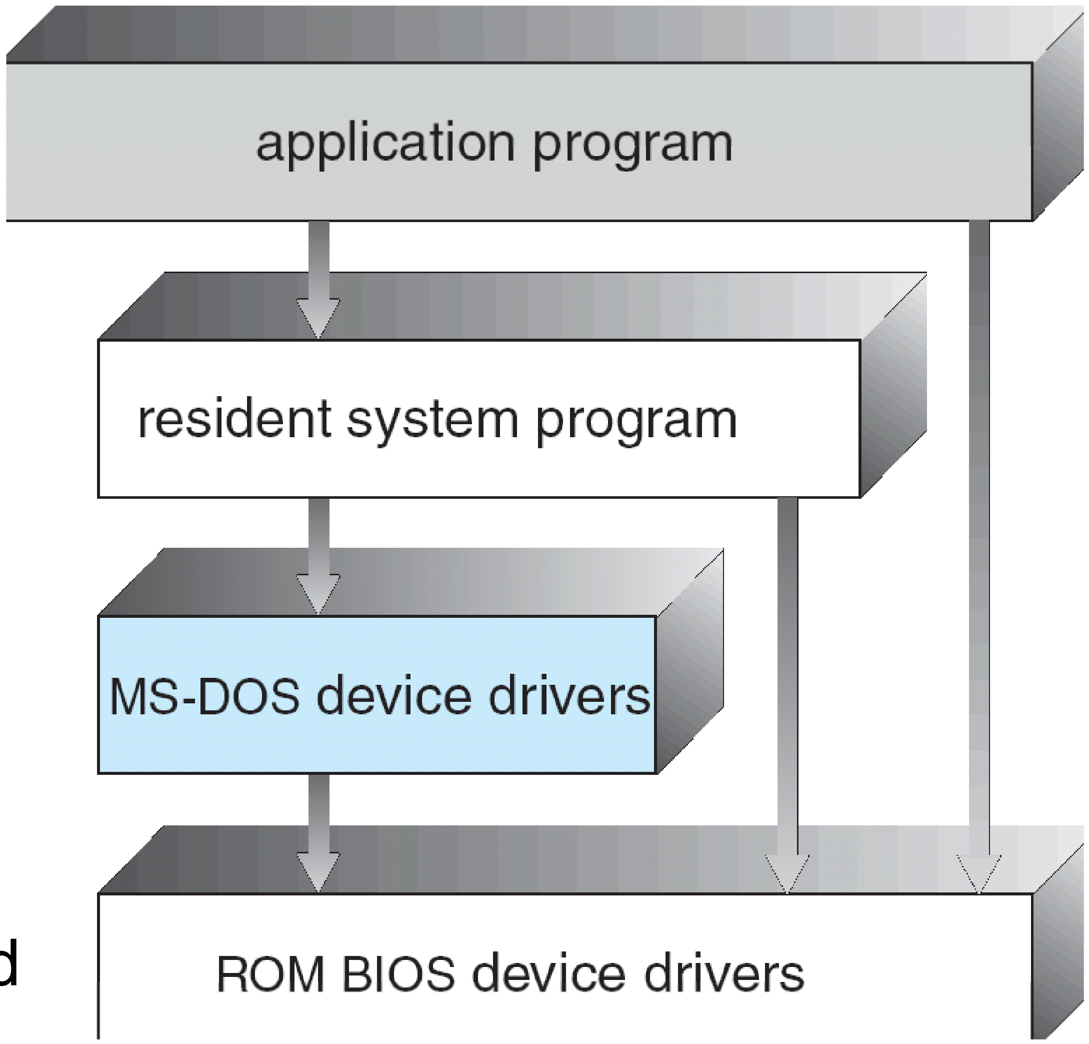
## Hybrid System Structure

- Kernel structure similar to a micro kernel, but implemented in terms of a monolithic kernel
  - In contrast to a micro kernel, all (or nearly all) operating system services are in kernel space
    - No performance overhead for message passing and context switching between kernel and user mode as in monolithic kernels
    - No reliability benefits of having services in user space as in micro kernels

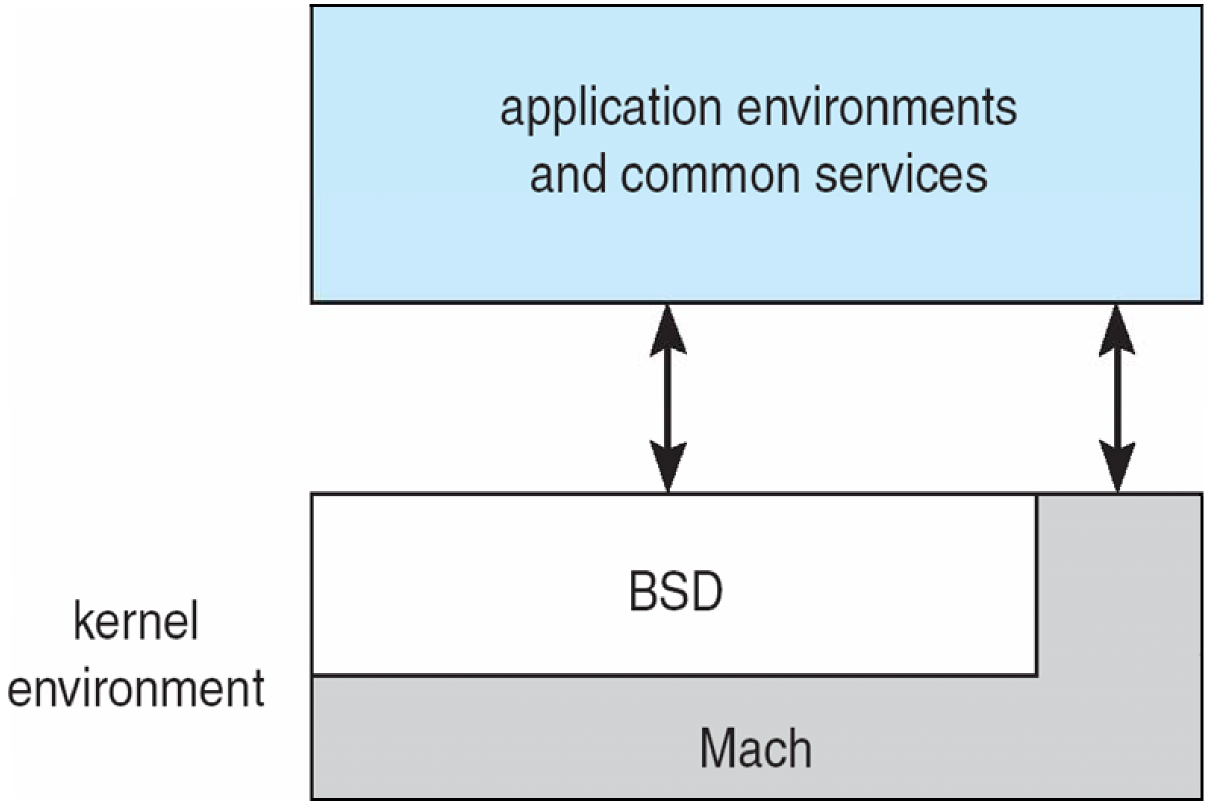
- Windows NT and Windows 8 used hybrid micro kernels

## Simple Structure

- MS-DOS: written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
  - The “first” wider deployed OS for PCs

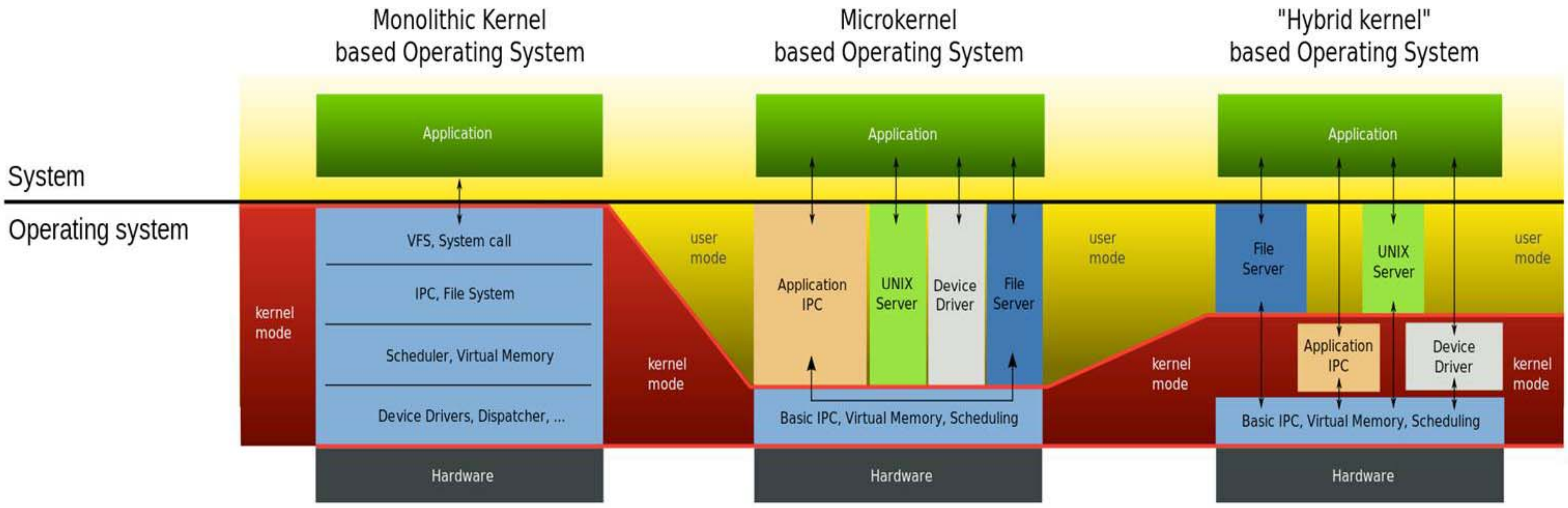


# Mac OS X Structure



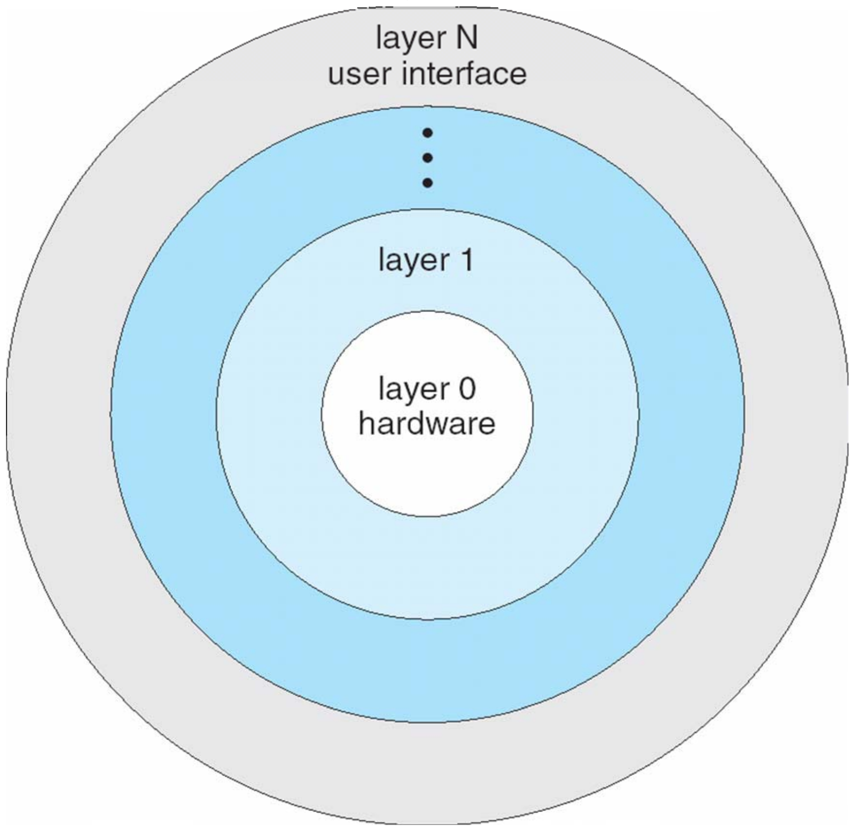
Micro kernel

# System Structure Comparison



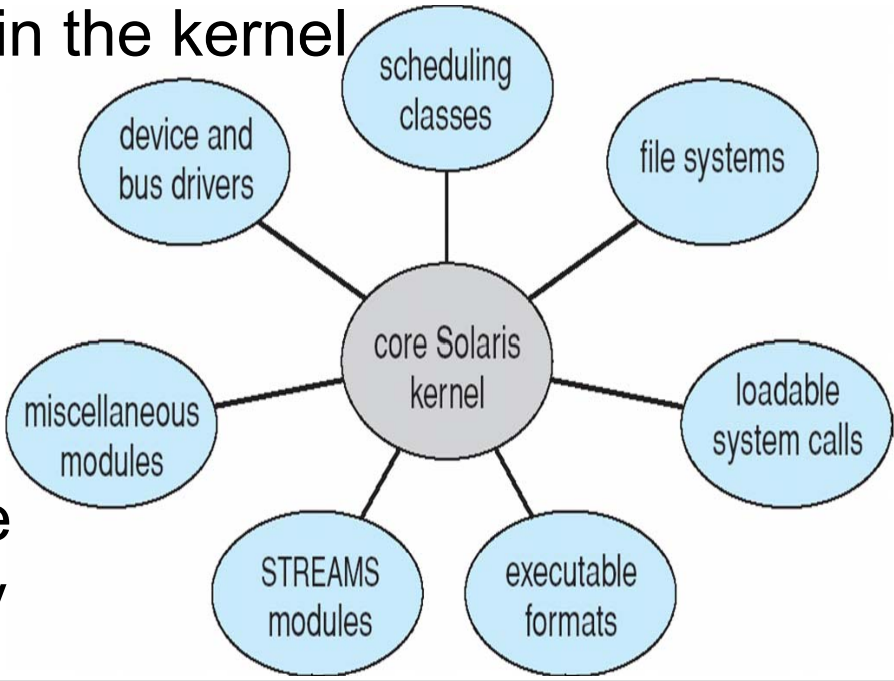
## Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



## Modules

- 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 flexibility
- OpenVMS, Linux, BSD, and UNIX variants such as SunOS
  - Can dynamically load executable modules at runtime, at the binary (image) level



# Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion:
  - It treats hardware and the native operating system kernel as though they were all hardware
  - A virtual machine provides an interface identical to the underlying bare hardware

- The operating system host creates the illusion that a process has its own processor and (virtual memory)
- Each guest provided with a (virtual) copy of the underlying computer

## Virtual Machines History and Benefits (1)

- First appeared commercially in IBM mainframes 1972

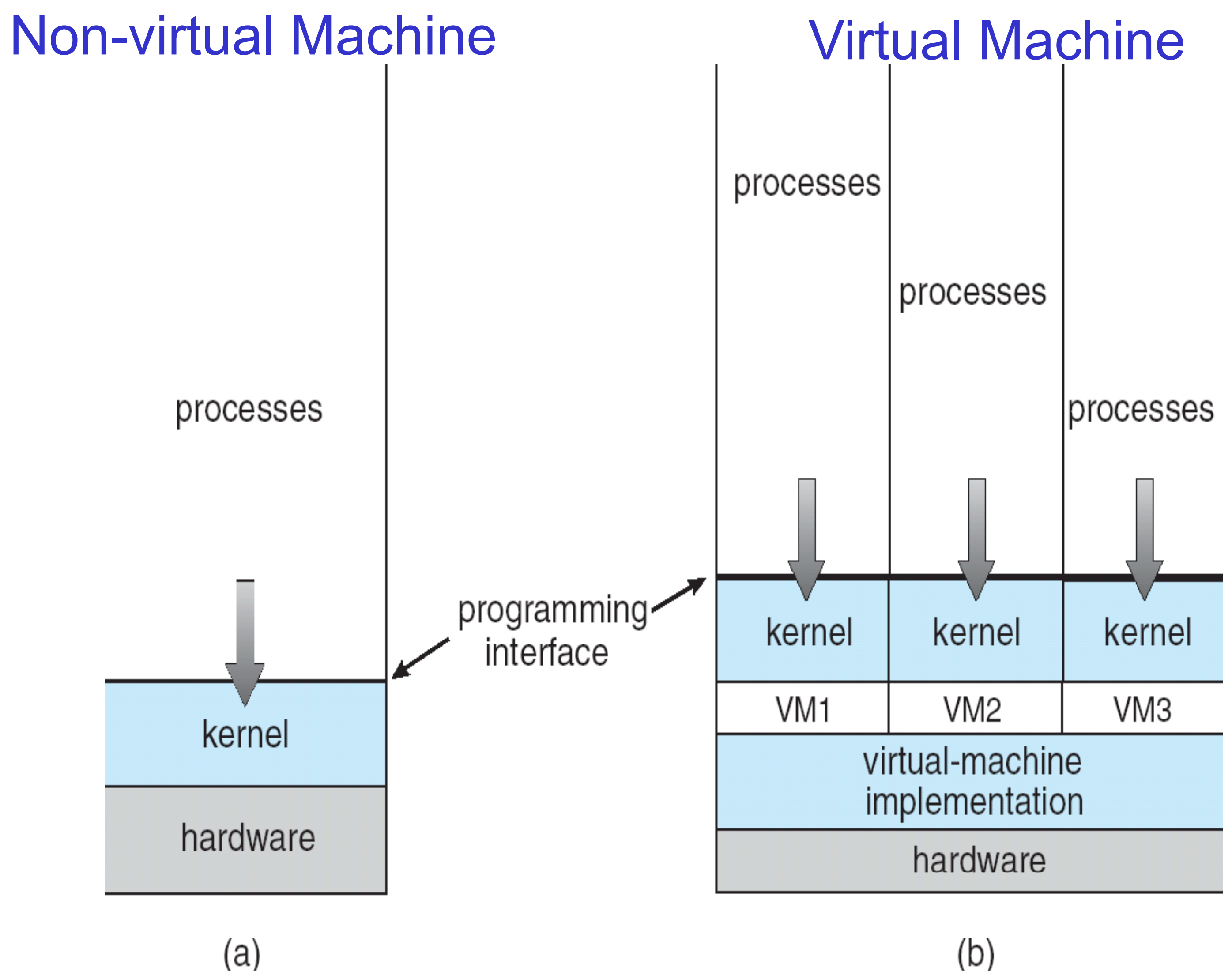
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protected from each other
- Some sharing of a file can be permitted, is controlled

- Commutate with each other, to other physical systems via networking

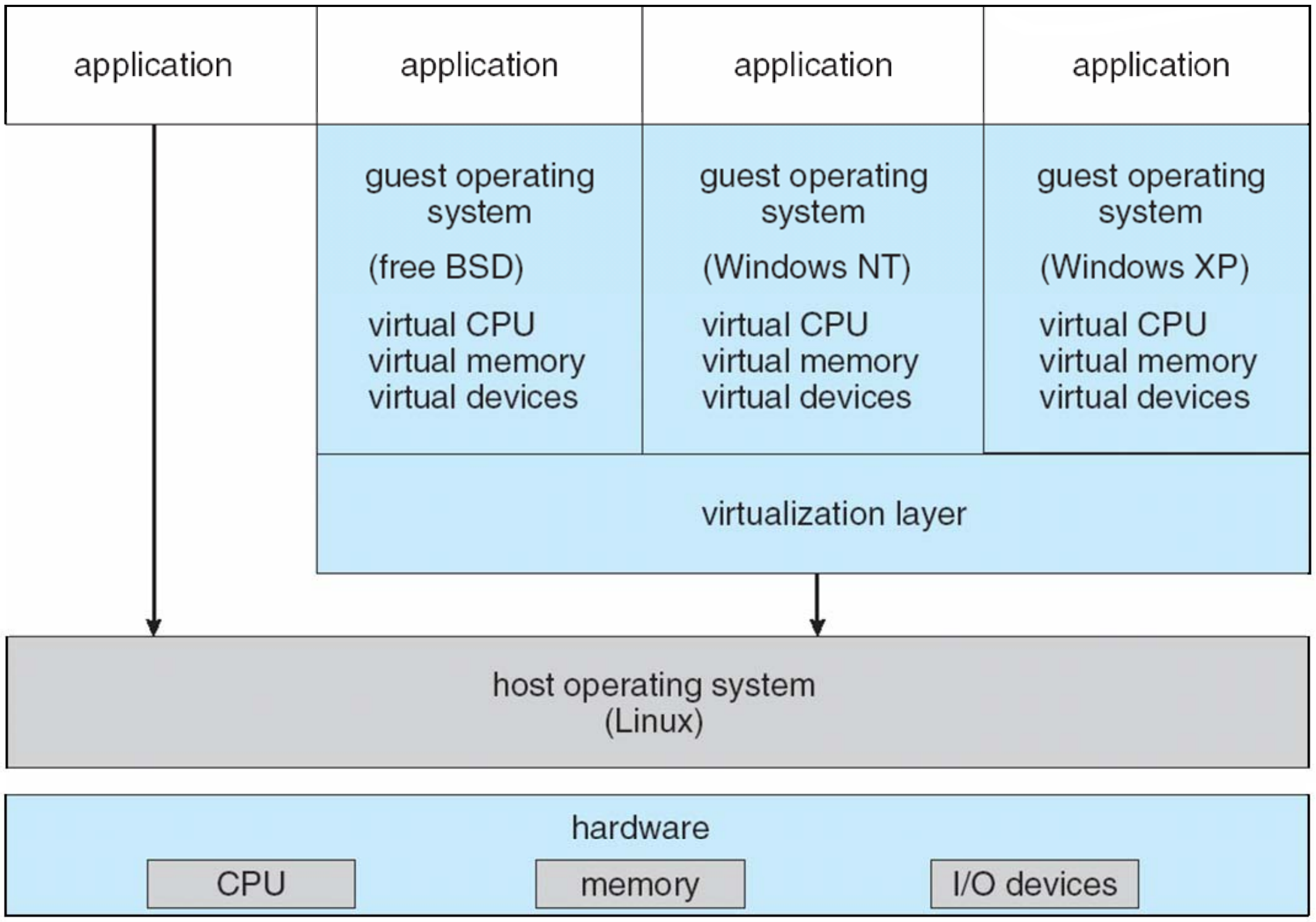
## Virtual Machines History and Benefits (2)

- Useful for development, testing
- Consolidation of many low-resource use systems onto fewer busier systems
- “Open Virtual (VM) Machine Format”, standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms

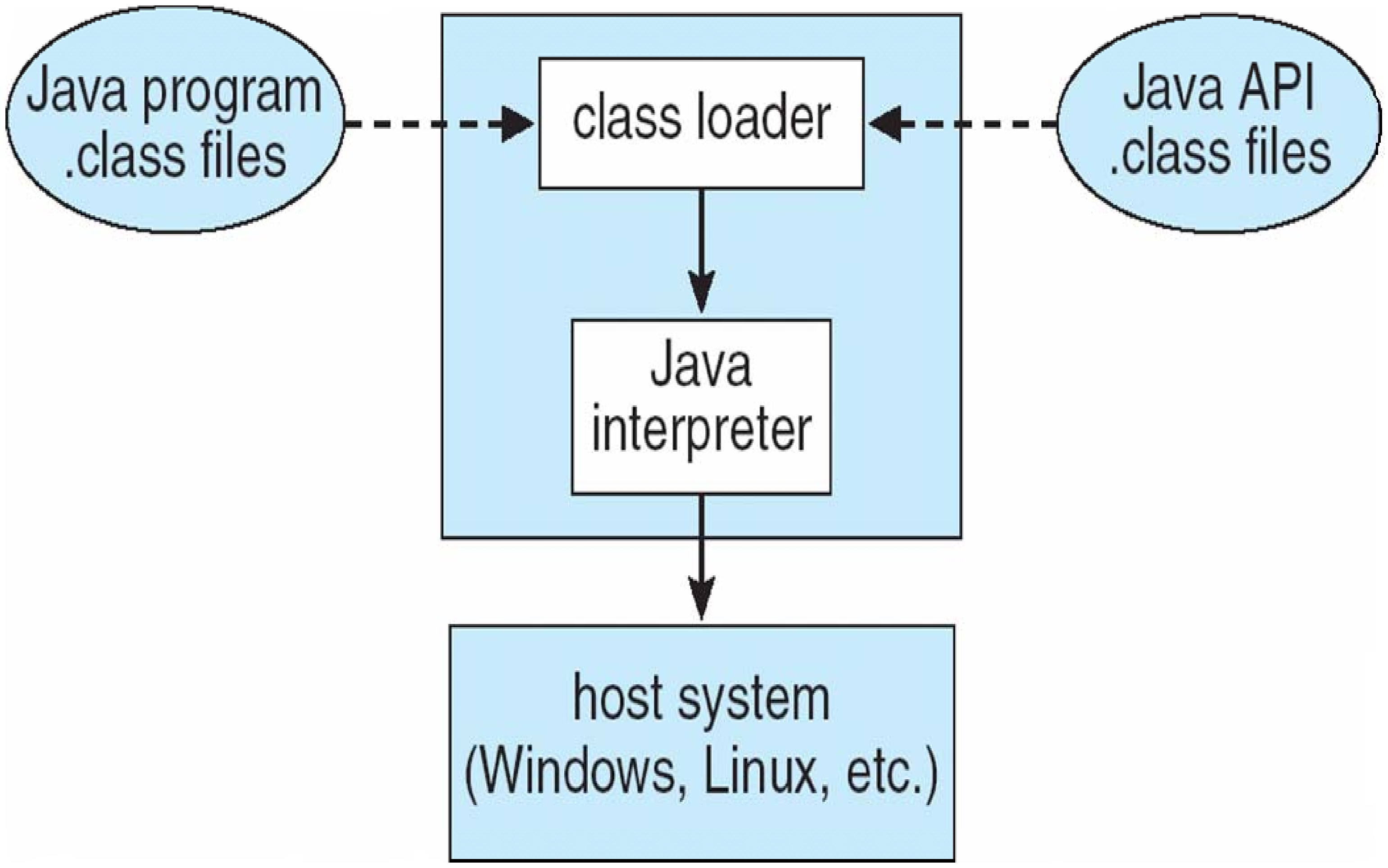
# Virtual Machines – Concept



## VMware Architecture



## The Java Virtual Machine



## .NET Framework

- Collection of technologies
  - Set of class libraries
  - Execution environment (and development platform)
- Target code written for .NET
  - Hides all specifics of underlying hardware (and OS software)
    - Execution environment abstracts away
  - Virtual Machine
- Core characteristics
  - Common Language Runtime (CLR), the .NET VM
  - Just-in-time compiler compliles intermediate from C#, VB.NET
    - Assemblies contain MS-IL (Intermediate Language) instructions (.DLL)
    - Results of CLR executed on host system

## System Boot

- Operating systems must be made available to the hardware, such that the hardware can start it
  - Small piece of code – bootstrap loader –, locates the kernel, loads it into memory, and starts it
  - Sometimes implemented as a two-step process, where the boot block located at a fixed location (address) loads the bootstrap loader
  - When power initialized on the system (hardware), the execution starts at a fixed memory location (address)
    - Firmware used to hold initial boot code