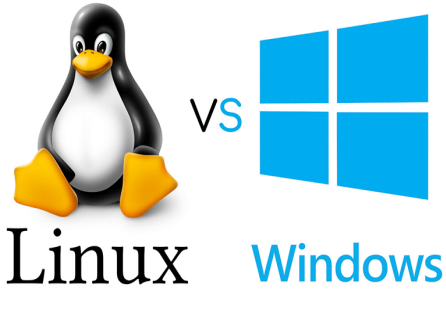
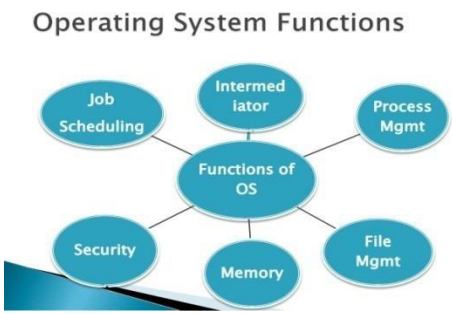
CIT210 OP. SYS. MANAG.

CSUN
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Week1Day2: Operating Systems Structure, Clusters, Distributed Computing Systems, RTOS, Mobile OS, Operating System Modes Systems Development, Object-Oriented Design





Objectives

Define an operating system

Describe the start-up process and shutdown options on computers and mobile devices

Explain how an operating system provides a user interface, manages programs, manages memory, and coordinates tasks

Describe how an operating system enables users to configure devices, establish an Internet connection, and monitor performance

Identify file management and other tools included with an operating system, along with ways to update operating system software

Objectives

Explain how an operating system enables users to control a network or administer security

Summarize the features of several desktop operating systems

Briefly describe various server operating systems

Summarize the features and uses of several mobile operating systems

What *is* an operating system?

- A program that runs on hardware and supports
  - Resource Abstraction
  - Resource Sharing
- Abstracts and standardizes the interface to the user across different types of hardware
  - Virtual machine hides the messy details which must be performed
- Manages the hardware resources
  - Each program gets time with the resource
  - Each program gets space on the resource
- May have potentially conflicting goals:
  - Use hardware efficiently
  - Give maximum performance to each user

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel -Mach

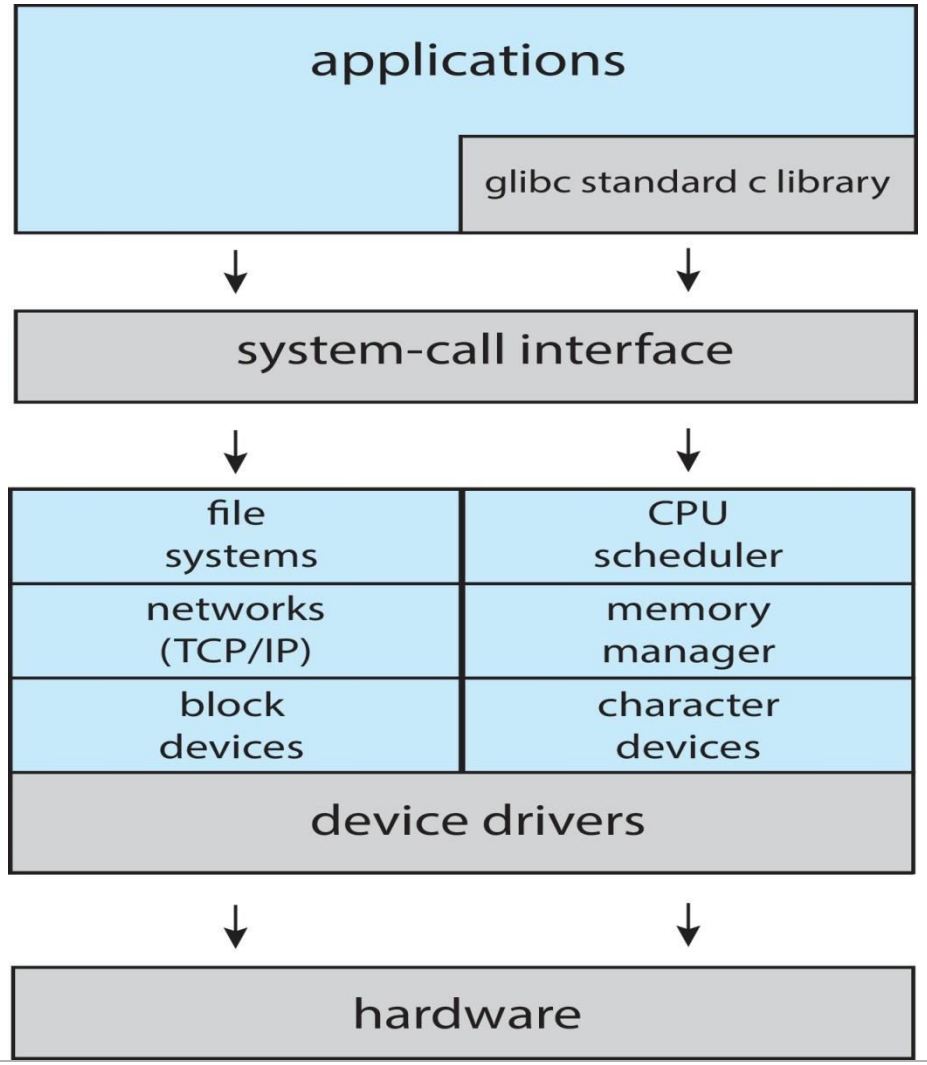
Monolithic Structure – Original UNIX

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS 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; a large number of functions for one level

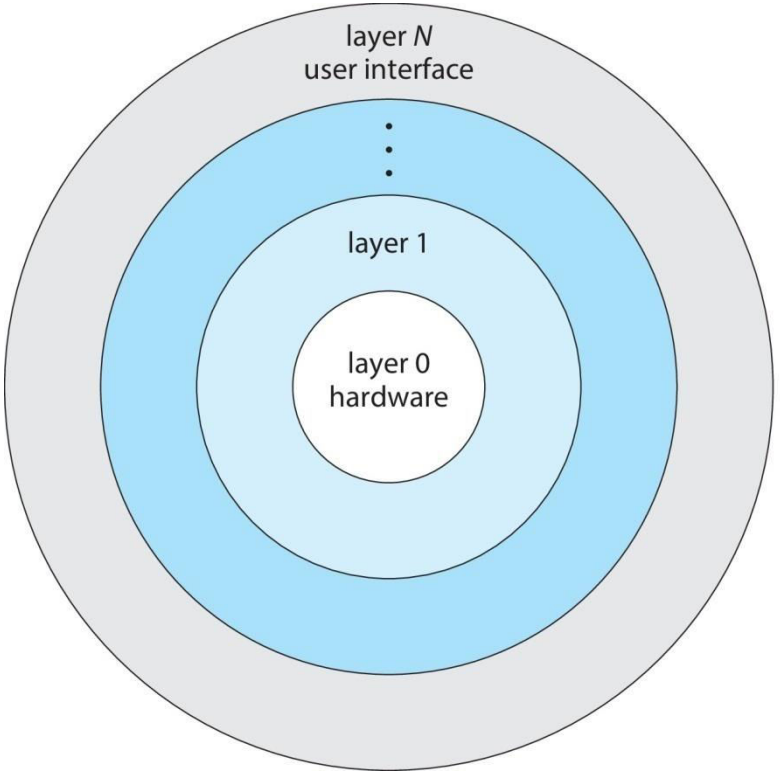
Linux System Structure

Monolithic plus modular design



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



System Structure

- View the system as a series of levels
- Each level performs a related subset of functions
- Each level relies on the next lower level to perform more primitive functions
- This decomposes a problem into a number of more manageable sub-problems

Process Hardware Levels

- Level 1
  - Electronic circuits
  - Objects are registers, memory cells, and logic gates
  - Operations are clearing a register or reading a memory location
- Level 2
  - Processor's instruction set
  - Operations such as add, subtract, load, and store
- Level 3
  - Adds the concept of a procedure or subroutine, plus call/return operations
- Level 4
  - Interrupts

Concepts with Multiprogramming

- Level 5
  - Process as a program in execution
  - Suspend and resume processes
- Level 6
  - Secondary storage devices
  - Transfer of blocks of data
- Level 7
  - Creates logical address space for processes
  - Organizes virtual address space into blocks

Deal with External Objects

- Level 8
  - Communication of information and messages between processes
- Level 9
  - Supports long-term storage of named files
- Level 10
  - Provides access to external devices using standardized interfaces
- Level 11
  - Responsible for maintaining the association between the external and internal identifiers
- Level 12
  - Provides full-featured facility for the support of processes
- Level 13
  - Provides an interface to the operating system for the user

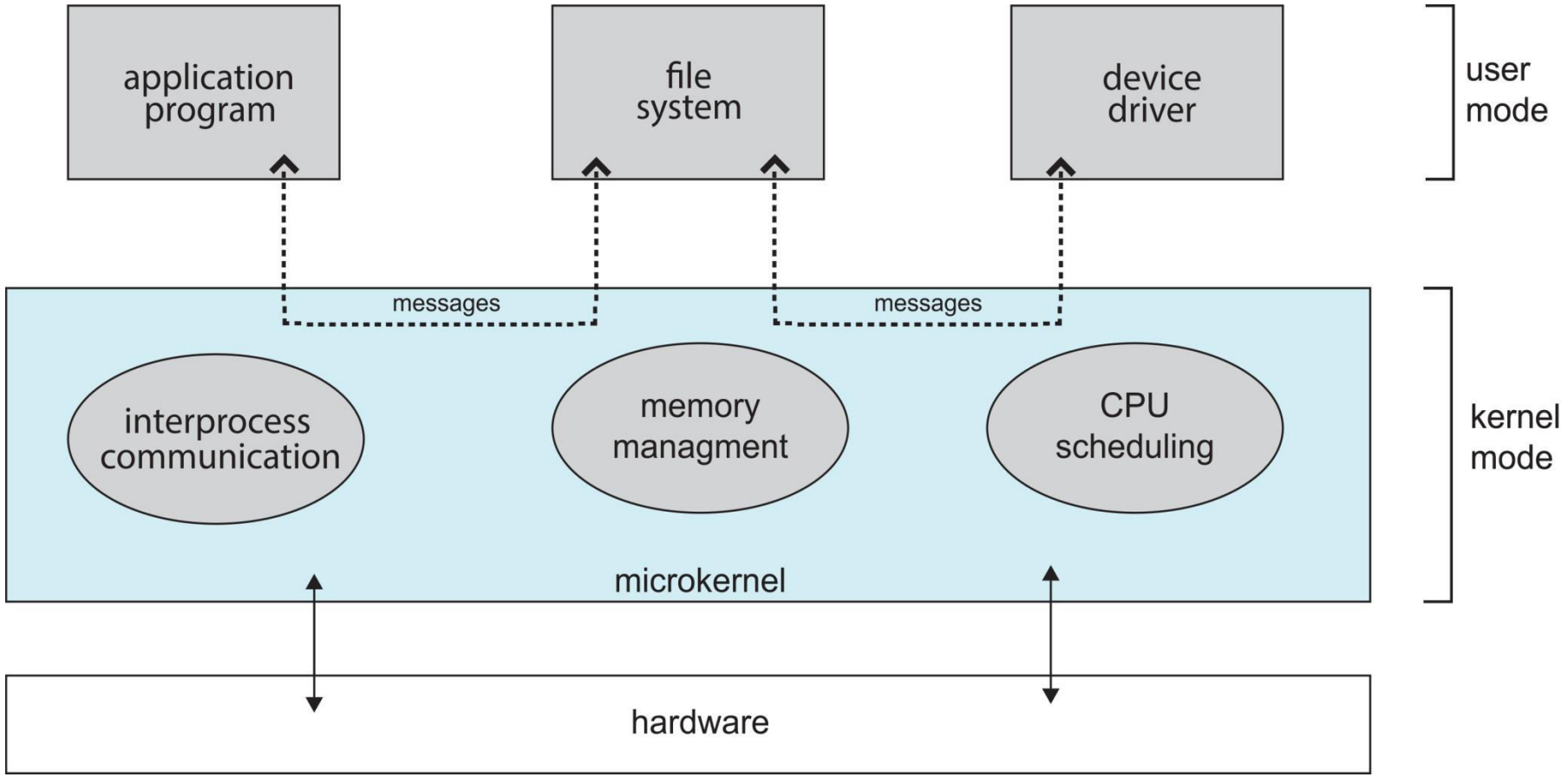
Layered Model of Operating System Concepts

nr	name	typical objects	typical operations
1	Integrated circuits	register, gate, bus	Nand, Nor, Exor
2	Machine language	instruction counter, ALU	Add, Move, Load, Store
3	Subroutine linkage	procedure block	Stack Call, JSR, RTS
4	Interrupts	interrupt handlers	Bus error, Reset
5	Simple processes	process, semaphore	wait, ready, execute
6	Local memory	data block, I/O channel	read, write, open, close
7	Virtual model	page, frame	read, write, swap
8	Process communication	channel (pipe), message	read, write, open
9	File management	files	read, write, open, copy
10	Device management	ext.memory, terminals	read, write
11	I/O data streams	data streams	open, close, read, write
12	User processes	user processes	login, logout, fork
13	Directory management	internal tables	create, delete, modify
14	Graphical user interface	window, menu, icon	OS system calls

Microkernels

- Moves as much from the kernel into user space
- Mach example of microkernel
  - Mac OS X kernel (Darwin) partly based on Mach
- 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

Microkernel System Structure



Modules

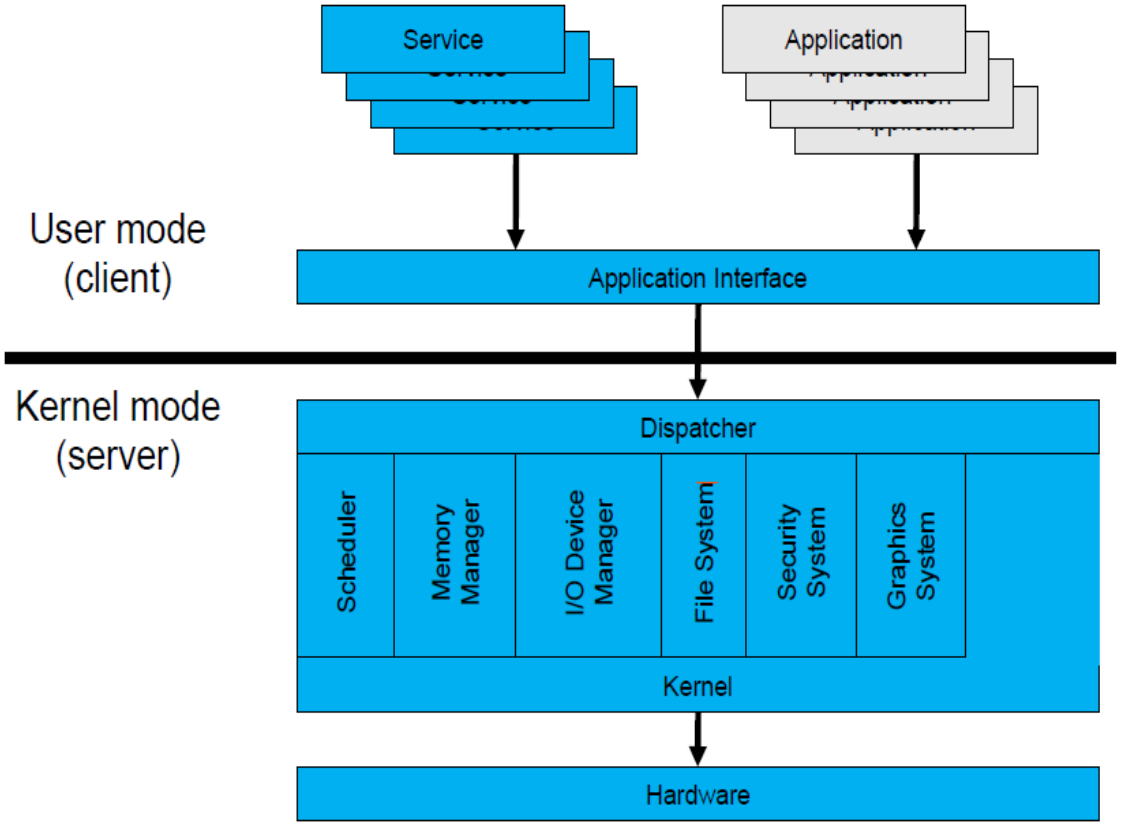
- Many modern operating systems implement loadable kernel modules (LKMs)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc

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
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)

Operating System Mode

- The *User Mode* is concerned with the actual interface between the user and the system.
- It controls things like running applications and accessing files.



- The *Kernel Mode* is concerned with everything running in the background.
- It controls things like accessing system resources, controlling hardware functions and processing program instructions.
- System calls are used to change mode from User to Kernel.

Kernel

- Kernel is a software code that reside in central core of OS. It has complete control over system.
- When operation system boots, kernel is first part of OS to load in main memory.
- Kernel remains in main memory for entire duration of computer session. The kernel code is usually loaded in to protected area of memory.
- Kernel performs it's task like executing processes and handling interrupts in kernel space.
- User performs it's task in user area of memory.
- This memory separation is made in order to prevent user data and kernel data from interfering with each other.
- Kernel does not interact directly with user, but it interacts using SHELL and other programs and hardware.

Kernel cont…

- Kernel includes:-

  1. Scheduler: It allocates the Kernel's processing time to various processes.

  2. Supervisor: It grants permission to use computer system resources to each process.

  3. Interrupt handler : It handles all requests from the various hardware devices which compete for kernel services.

  4. Memory manager : allocates space in memory for all users of kernel service.

- kernel provides services for process management, file management, I/O management, memory management.
- System calls are used to provide these type of services.

System Call

- System call is the programmatic way in which a computer program/user application requests a service from the kernel of the operating system on which it is executed.
- Application program is just a user-process. Due to security reasons , user applications are not given access to privileged resources(the ones controlled by OS).
- When they need to do any I/O or have some more memory or spawn a process or wait for signal/interrupt, it requests operating system to facilitate all these. This request is made through System Call.
- System calls are also called software-interrupts.

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use
- Three most common APIs are Win64 API for Windows, 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)

System Call Implementation

- **Typically, a number 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 and returns status of the system call and any return values**
- **The caller need know nothing about how the system call is implemented**
  - **Just needs to obey API and understand what OS will do as a result call**
  - **Most details of OS interface hidden from programmer by API**
    - Managed by run-time support library (set of functions built into libraries included with compiler)

Types of System Calls

- **Process control**
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - Debugger for determining bugs, single step execution
  - Locks for managing access to shared data between processes

Types of System Calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (Cont.)

- **Information maintenance**
  - **get time or date, set time or date**
  - **get system data, set system data**
  - **get and set process, file, or device attributes**
- **Communications**
  - **create, delete communication connection**
  - **send, receive messages if message passing model to host name or process name**
    - From client to server
  - **Shared-memory model create and gain access to memory regions**
  - **transfer status information**
  - **attach and detach remote devices**

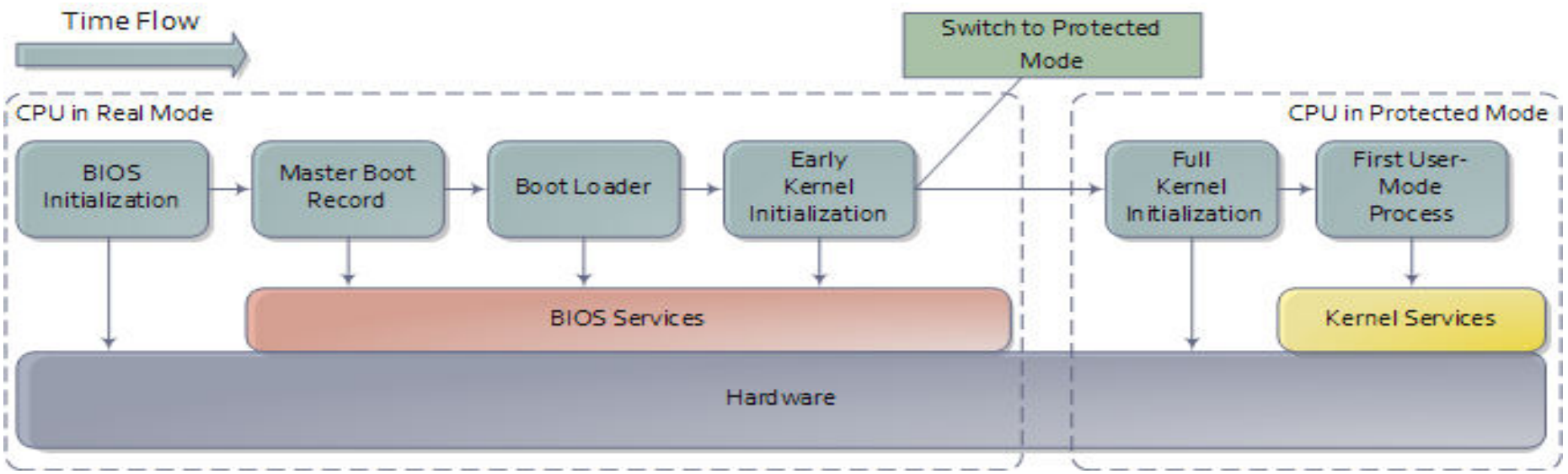
Types of System Calls (Cont.)

- **Protection**
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

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()

Starting an Operating System(Booting)



- ✓ Power On Switch sends electricity to the motherboard on a wire called the *Voltage Good* line.
- ✓ If the power supply is good, then the BIOS (Basic Input/Output System) chip takes over.
- ✓ In Real Mode, CPU is only capable of using approximately 1 MB of memory built into the motherboard.
- ✓ The BIOS will do a Power-On Self Test (POST) to make sure that all hardware are working.

- ✓ BIOS will then look for a small sector at the very beginning of your primary hard disk called MBR.
- ✓ The MBR contains a list, or map, of all of the partitions on your computer‘s hard disk (or disks).
- ✓ After the MBR is found the Bootstrap Loader follows basic instructions for starting up the rest of the computer, including the operating system.
- ✓ In Early Kernel Initialization stage, a smaller core of the Kernel is activated.
- ✓ This core includes the device drivers needed to use computer‘s RAM chips.

BIOS

- BIOS firmware was stored in a ROM/EPROM (Erasable Programmable Read-Only Memory) chip known as firmware on the PC motherboard.
- BIOS can be accessed during the initial phases of the boot procedure by pressing del, F12 or F10 (Depends on computer vendors) .
- Finally, the firmware code cycles through all storage devices and looks for a boot-loader. (usually located in first sector of a disk which is 512 bytes)
- If the boot-loader is found, then the firmware hands over control of the computer to it.

UEFI

- UEFI stands for Unified Extensible Firmware Interface. It does the same job as a BIOS, but with one basic difference: it stores all data about initialization and startup in an .efi file, instead of storing it on the firmware.
- This .efi file is stored on a special partition called EFI System Partition (ESP) on the hard disk. This ESP partition also contains the bootloader.
- UEFI was designed to overcome many limitations of the old BIOS, including:
  - UEFI supports drive sizes upto 9 zettabytes, whereas BIOS only supports 2.2 terabytes.
  - UEFI provides faster boot time.
  - UEFI has discrete driver support, while BIOS has drive support stored in its ROM, so updating BIOS firmware is a bit difficult.
  - UEFI offers security like "Secure Boot", which prevents the computer from booting from unauthorized/unsigned applications. This helps in preventing rootkits.
  - UEFI runs in 32bit or 64bit mode, whereas BIOS runs in 16bit mode. So UEFI is able to provide a GUI (navigation with mouse) as opposed to BIOS which allows navigation only using the keyboard.

Brief History of Operating Systems Development

- **1940s: first generation**
  - **Computers based on vacuum tube technology**
  - **No standard operating system software**
  - **Typical program included every instruction needed by the computer to perform the tasks requested**
  - **Poor machine utilization**
    - CPU processed data and performed calculations for fraction of available time
  - **Early programs**
    - Designed to use the resources conservatively
    - Understandability is not a priority

First generation: direct input

- Run one job at a time
  - Enter it into the computer (might require rewiring!)
  - Run it
  - Record the results
- Problem: lots of wasted computer time!
  - Computer was idle during first and last steps
  - Computers were *very* expensive!
- Goal: make better use of an expensive commodity: computer time

Brief History of Operating Systems Development

- **1950s: second generation**
  - **Focused on cost effectiveness**
  - **Computers were expensive**
    - IBM 7094: $200,000
  - **Two widely adopted improvements**
    - Computer operators: humans hired to facilitate machine operation
    - Concept of job scheduling: group together programs with similar requirements
  - **Expensive time lags between CPU and I/O devices**

Brief History of Operating Systems Development

- **1950s: second generation (continued)**
  - **I/O device speed gradually became faster**
    - Tape drives, disks, and drums
  - **Records blocked *before* retrieval or storage**
  - **Access methods developed**
    - Added to object code by linkage editor
  - **Buffer between I/O and CPU introduced**
    - Reduced speed discrepancy
  - **Timer interrupts developed**
    - Allowed job-sharing

Brief History of Operating Systems Development

- **1960s: third generation**
  - **Faster CPUs**
  - **Speed caused problems with slower I/O devices**
  - **Multiprogramming**
    - Allowed loading many programs at one time
  - **Program scheduling**
    - Initiated with second-generation systems
    - Continues today
  - **Few advances in data management**
  - **Total operating system customization**
    - Suit user's needs

Brief History of Operating Systems Development

- **1970s**
  - **Faster CPUs**
  - **Speed caused problems with slower I/O devices**
  - **Main memory physical capacity limitations**
    - Multiprogramming schemes used to increase CPU
    - Virtual memory developed to solve physical limitation
  - **Database management software**
    - Became a popular tool
  - **A number of query systems introduced**
  - **Programs started using English-like words, modular structures, and standard operations**

Brief History of Operating Systems Development

(figure 1.9)

The Cray I supercomputer, introduced in 1976, boasted 8 MB main memory and a world-record speed of 160 million floating-point operations per second. Its circular design meant that no wire was more than 4 feet (1.2 meters) long.

Brief History of Operating Systems Development

- **1980s**
  - **Cost/performance ratio improvement of computer components**
  - **More flexible hardware (firmware)**
  - **Multiprocessing**
    - Allowed parallel program execution
  - **Evolution of personal computers**
  - **Evolution of high-speed communications**
  - **Distributed processing and networked systems introduced**

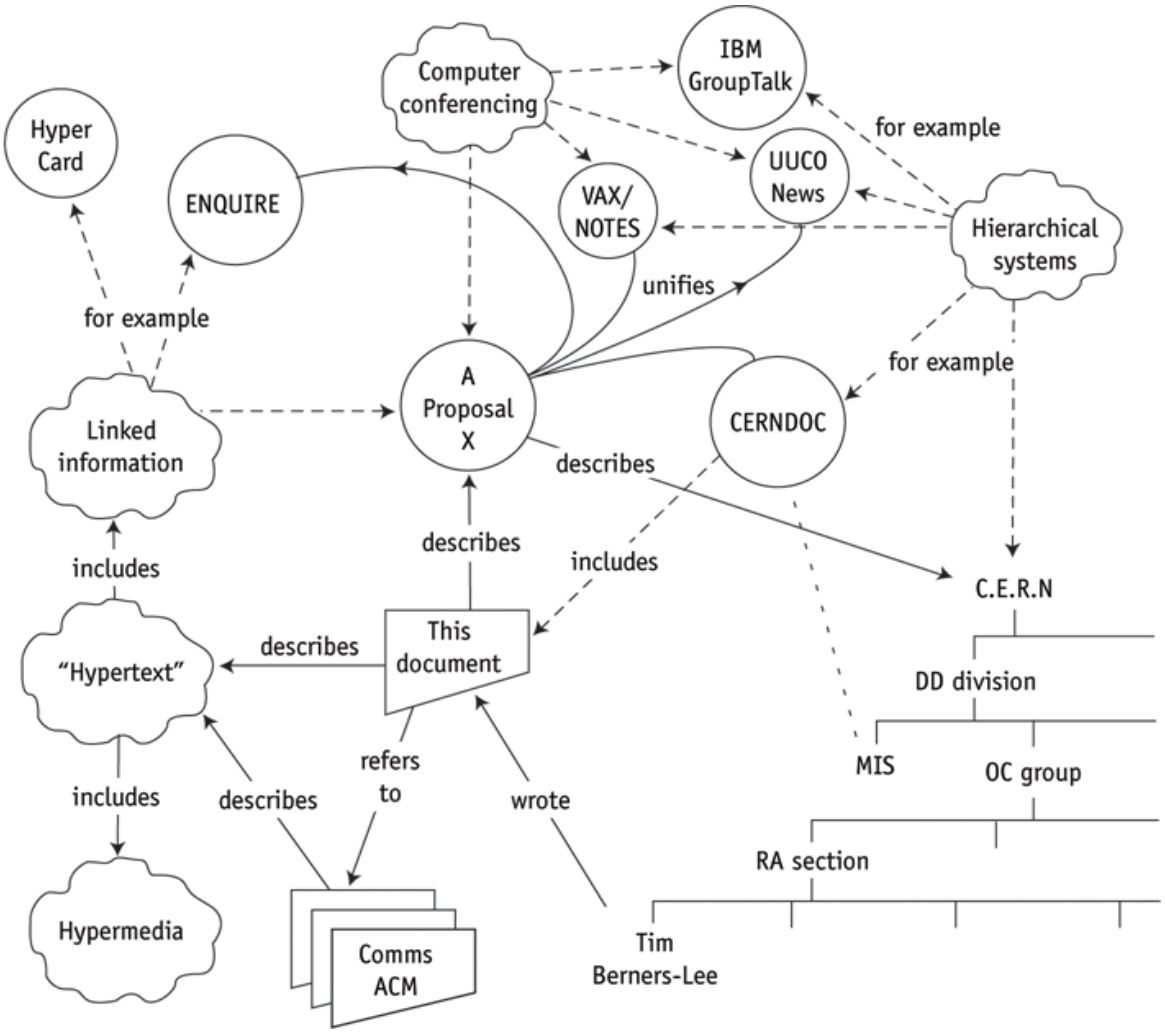
Brief History of Operating Systems Development

- **1990s**
  - **Demand for Internet capability**
    - Sparked proliferation of networking capability
    - Increased networking
    - Increased tighter security demands to protect hardware and software
  - **Multimedia applications**
    - Demanding additional power, flexibility, and device compatibility for most operating systems

Brief History of Operating Systems Development

(figure 1.10)

Illustration from the first page of the 1989 proposal by Tim Berners-Lee describing his revolutionary "linked information system." Based on this research, he designed the first World Wide Web server and browser, making it available to the general public in 1991.



Brief History of Operating Systems Development

- **2000s**
  - **Primary design features support:**
    - Multimedia applications
    - Internet and Web access
    - Client/server computing
  - **Computer systems requirements**
    - Increased CPU speed
    - High-speed network attachments
    - Increased number and variety of storage devices
  - **Virtualization**
    - Single server supports different operating systems

Categories of Operating Systems

- Batch systems, in which a set of jobs are submitted in sequence for processing.
  - Linkage of library routines to programs
  - Management of files, I/O devices, secondary storage
- Interactive systems, which support computing for on-line users. The most common type of operating systems that support interactive computing is time-sharing, which are multi-user systems.
- Real-time systems, which support application programs with very tight timing constraints.
- Hybrid systems, which support batch and interactive computing.

Time-Sharing

- Batch multiprogramming does not support interaction with users.
- Time-sharing extends Batch Multiprogramming to handle multiple interactive jobs –
  it's Interactive Multiprogramming.
- Multiple users simultaneously access the system through commands entered at terminals.
- Processor's time is shared among multiple users.

Why Time-sharing?

- In Time-sharing (multitasking) the CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing:
  - Response time should be < 1 ms.
  - Each user has at least one program (process) executing in memory.
  - CPU scheduling supports several jobs ready to run at the same time.
  - If processes don't fit in memory, swapping moves them in and out to run.
  - Virtual memory allows execution of processes not completely in memory.

Categories of Operating Systems

- Interactive systems, which support computing for on-line users. The most common type of operating systems that support interactive computing is time-sharing, which are multi-user systems.
  - Multiprogramming
    - Resource managment and sharing for multiple programs
    - Quasi-simultaneous program execution
    - Single user
  - Multiuser/Timesharing Systems
    - Management of multiple simultaneous users interconnected via terminals
    - Fair resource management: CPU scheduling, spooling, mutual exclusion

Categories of Operating Systems

- Real-time systems, which support application programs with very tight timing constraints.
  - Management of time-critical processes
  - High requirements with respect to reliability and availability
- Hybrid systems, which support batch and interactive computing.

Real-Time Systems

- Note that not all Operating Systems are general-purpose systems.
- Real-Time (RT) systems are dedicated systems that need to adhere to deadlines , i.e., time constraints.
- Correctness of the computation depends not only on the logical result but also on the time at which the results are produced.

Hard Real-Time Systems

- Hard real-time system must meet its deadline.
- Conflicts with time-sharing systems, not supported by general-purpose OSs.
- Often used as a control device in a dedicated application:
  - Industrial control
  - Robotics
- Secondary storage limited or absent, data/program is stored in short term memory, or Read-Only Memory (ROM).

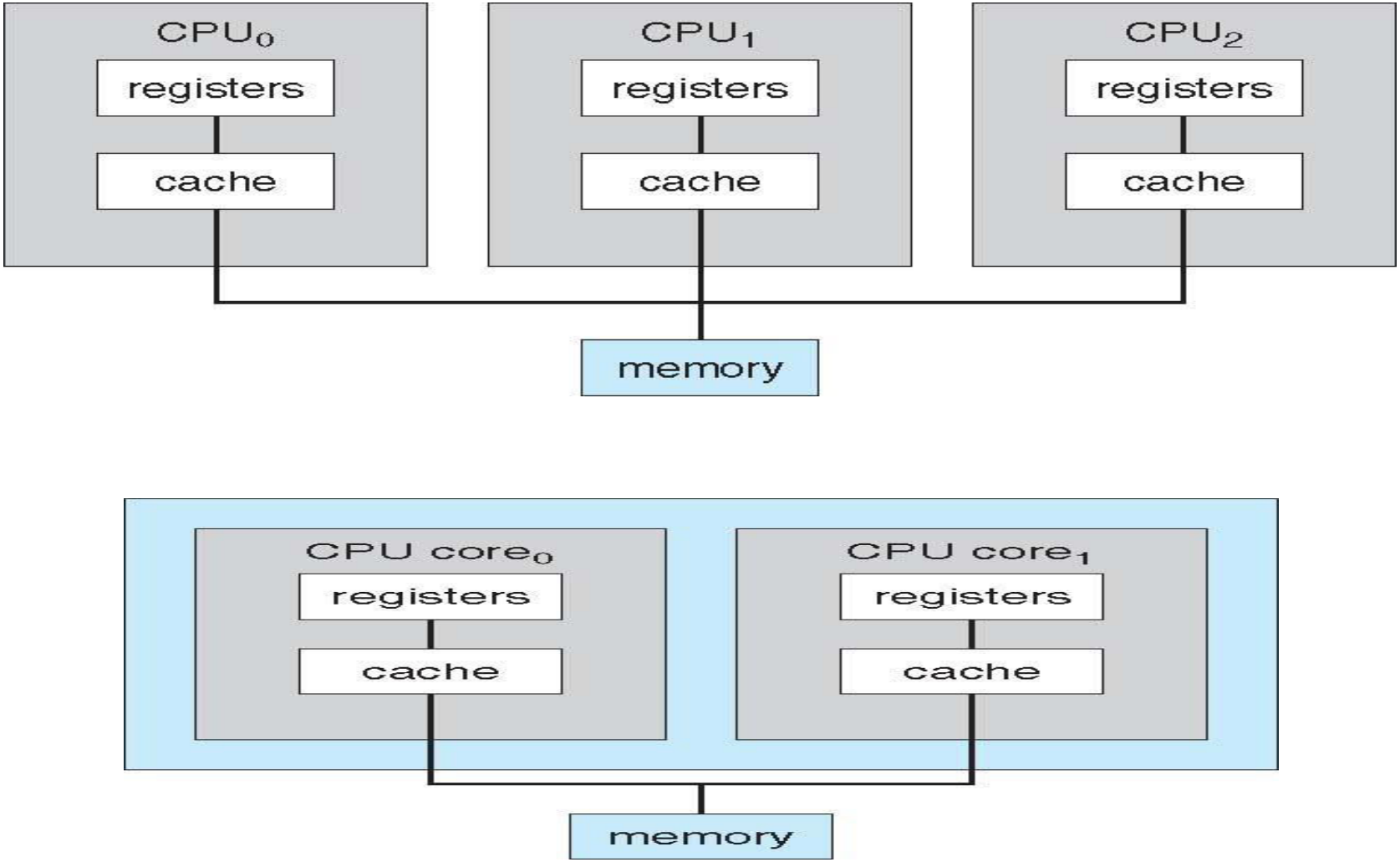
Soft Real-Time Systems

- Soft real-time system:
  - Deadlines desirable but not mandatory.
  - Limited utility in industrial control or robotics.
  - Useful in modern applications (multimedia, video conference, virtual reality) requiring advanced operating-system features.

Multiprocessor Systems

- System with several CPUs in close communication:
  - processors share memory and a clock.
  - communication usually takes place through the shared memory.
- Also known as parallel systems, tightly-coupled systems.
- Multiprocessors systems growing in use and importance – advantages include:
  - Increased throughput
  - Economy of scale
  - Increased reliability – graceful degradation or fault tolerance.

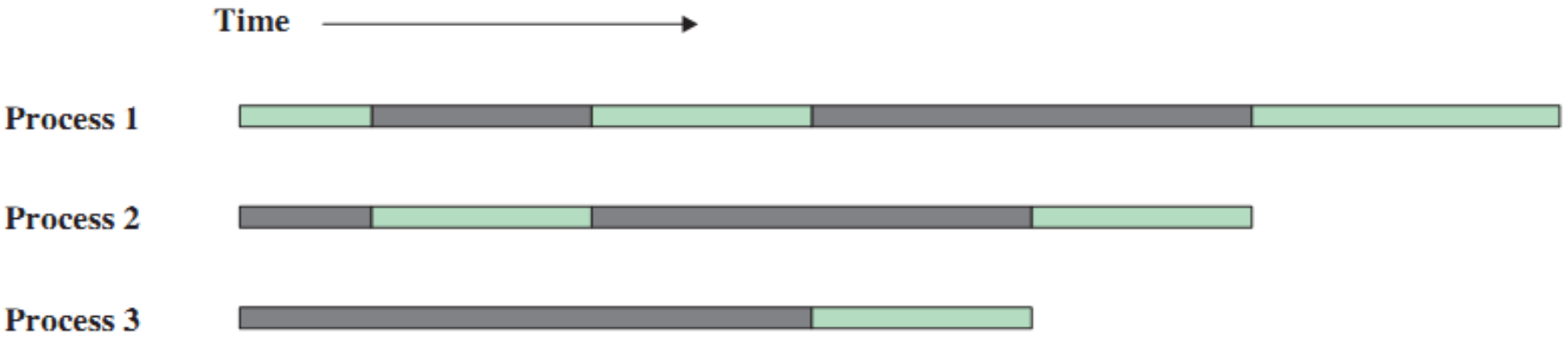
Multiprocessor Architecture



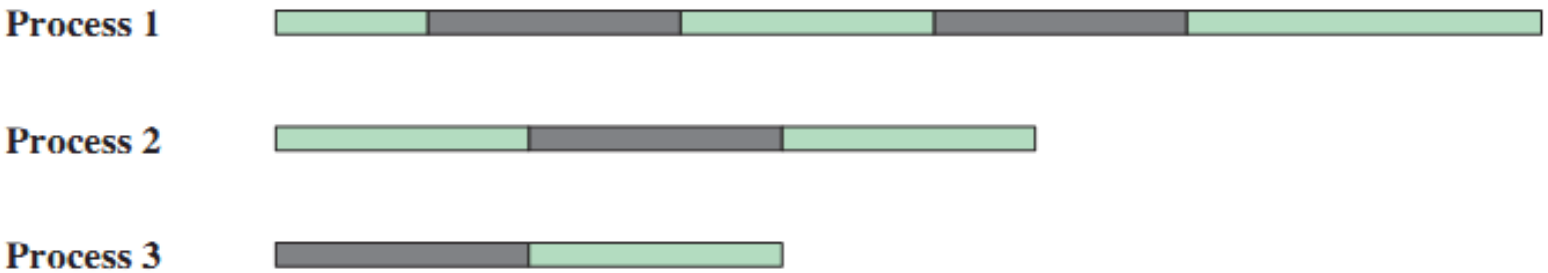
Threads

- **Multiple actions executing simultaneously**
  - **Heavyweight process (conventional process)**
    - Owns the resources
    - Passive element
  - **Lightweight process (thread)**
    - Uses CPU and scheduled for execution
    - Active element
  - **Multithreaded applications programs**
    - Contain several threads running at one time
    - Same or different priorities
    - Examples: Web browsers and time-sharing systems

Multiprogramming vs. Multiprocessing



(a) Interleaving (multiprogramming, one processor)



(b) Interleaving and overlapping (multiprocessing; two processors)

Types of Multiprocessor Systems

- ***Asymmetric Multiprocessing***
  - **master processor schedules and allocates specific work to slave processors.**
- ***Symmetric Multiprocessing (SMP)***
  - **Each processor runs an identical copy of the OS.**
  - **Typically each processor does self-scheduling form the pool of available processes.**
  - **Most modern operating systems support SMP.**

Symmetric Multiprocessing (SMP)

- Each processor can perform the same functions and share same main memory and I/O facilities (symmetric).
- The OS schedule processes/threads across all the processors (real parallelism).
- Existence of multiple processors is transparent to the user.
- Incremental growth: just add another CPU!
- Robustness: a single CPU failure does not halt the system, only the performance is reduced.

Clustered Systems

- Like multiprocessor systems, but multiple systems working together.
- Also known as closely-coupled system.
- Clustering allows two or more systems to share external storage and balance CPU load:
  - processors also have their own external memory.
  - communication takes place through high-speed channels.
  - Provides high-availability

Clustered Systems

- Usually sharing storage via a Storage-Area Network (SAN).
- Provides a high-availability service which survives failures:
  - Asymmetric clustering has one machine in hot-standby mode
  - Symmetric clustering has multiple nodes running applications, monitoring each other.
- Some clusters are used for high-performance computing (HPC) where applications must be written to use parallelization.

Distributed Systems

- **Distributed system is collection of loosely coupled processors interconnected by a communications network.**
- **Processors variously called *nodes, computers, machines, hosts.***
- **Reasons for distributed systems:**
  - **Resource sharing:**
    - sharing and printing files at remote sites.
    - processing information in a distributed database.
    - using remote specialized hardware devices.
  - **Computation speedup – load sharing.**
  - **Reliability – detect and recover from site failure, function transfer, reintegrate failed site.**
  - **Communication – message passing.**

Mobile Systems

- Handheld smartphones, tablets, etc.
- What is the functional difference between them and a “traditional” laptop?
- Extra feature – more OS features (GPS, gyroscope).
- Use IEEE 802.11 wireless, or cellular data networks for connectivity.
- Allows new types of apps like *augmented reality.*
- Leaders are Apple iOS and Google Android.

Popular Operating Systems for Mobile Devices

- **Mobile operating system: an operating system designed specifically for a mobile device**
  - **apps: small application programs managed by a mobile OS**
  - **the OS is stored on a memory chip**
  - **designed to manage hardware and applications with less memory than a personal computer operating system, yet runs fast and efficiently**
  - **often manages input from a touchscreen and/or voice command and routes data to/from wireless connections including web browsing**

Popular Operating Systems for Mobile Devices

- **Android: the OS is based on Linux, making it an open source OS**
  - **currently enjoys the highest market share for mobile devices**
  - **Open Handset Alliance: a consortium of several mobile technology companies of which Google is a member**
  - **Android Open Source Project (AOSP): led by Google and maintains and develops Android**
- **iOS: developed by Apple as the mobile operating system for its iPhone, iPod Touch, and iPad**
- **Embedded operating system: smaller and interacts with fewer resources to perform specific tasks fast and reliably**

Android Mobile OS – Architecture (Contd.)

Linux kernel

- For core system services such as security, memory management, and process management.

Runtime

- Set of core libraries which supports Java functionality
- The Android Virtual Machine known as Dalvik VM
- Relies on the Linux kernel for underlying functionality such as threading,…

Android Mobile OS – Architecture (Contd.)

Libraries

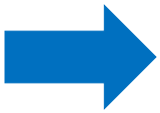
- Includes a set of libraries. These libraries are exposed to developers through the Android application framework. They include media libraries, system C libraries, surface manager, 3D libraries, SQLite and etc.

Application Framework

- An access layer to the framework APIs used by the core applications.
- Allows components to be used by the developers.

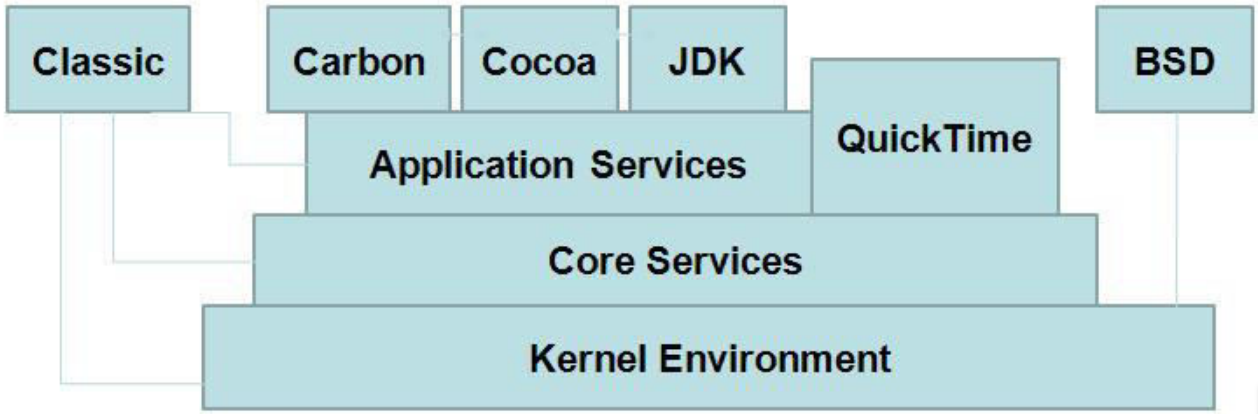
iPhone OS

Based on Mach kernel and Darwin Core as Mac OS X



- BSD
- File Systems
- I/O systems
- Networking components

Mac OS X Architecture



Each application has 4GB space

Multitasking	Pre-emptive, i.e. act of taking the control of operating system from one task and giving it to another task.	Real-time	Strong memory protection

Android vs iOS

- **Programming Language**
  - **Android OS: Java**
  - **iOS: Objective C**
- **Development Platform**
  - **Android OS:**
    - open platform, allowing the use of 3rd party tools
    - Key to OS success
    - can reach core components. More like PC swr
  - **iOS:**
    - Restrictive guidelines
    - Fixed set of tools, nothing outside, nothing deep
    - No Flash!

Multitasking Abilities

- **Android OS:**
  - **Very versatile → dynamic**
  - **Highly fragmented → challenging**
    - In USA: **80** Android models *vs*. **9** iOS models
  - **Poor battery performance**
  - **Best notification system (e.g. emails)**

- **iOS:**
  - **Stable and exclusive platform**
  - **Fixed set of tools, with clear potential and boundaries → easier**

Security

- **Android OS:**
  - **Access control, isolation, web security**
  - **Encryption**
  - **Permission-based access control:**
    - Static list in manifest
    - User presented with list at installation time
  - **Wild West app marketplace.**
    - Nearly any app is allowed to market
    - Android-specific malware

Security

- **iOS:**
  - **Access control, isolation, web security**
  - **Encryption**
  - **Permission-based access control:**
    - Dialog box at run time.
  - **Geolocation**
  - **Auto Erase**

Embedded Operating Systems, Cloud Operating Systems

- **Windows Embedded:**
  - **a family of operating systems based on the familiar Windows operating system designed for use in a variety of devices**
- **Embedded Linux:**
  - **applications that can be found running smart appliances, in-flight entertainment systems, personal navigation systems, and a variety of other consumer and commercial electronics**
- **Android:**
  - **an operating system based on Linux and is a popular choice for electronic devices beyond smartphones such as tablets, set-top boxes, and netbooks**
- **Cloud operating systems: operating systems that operate like a virtual desktop**
  - **web-based operating systems: not true operating systems (such as Windows) because you still need a standalone operating system on the computer you are using to access the web browser**

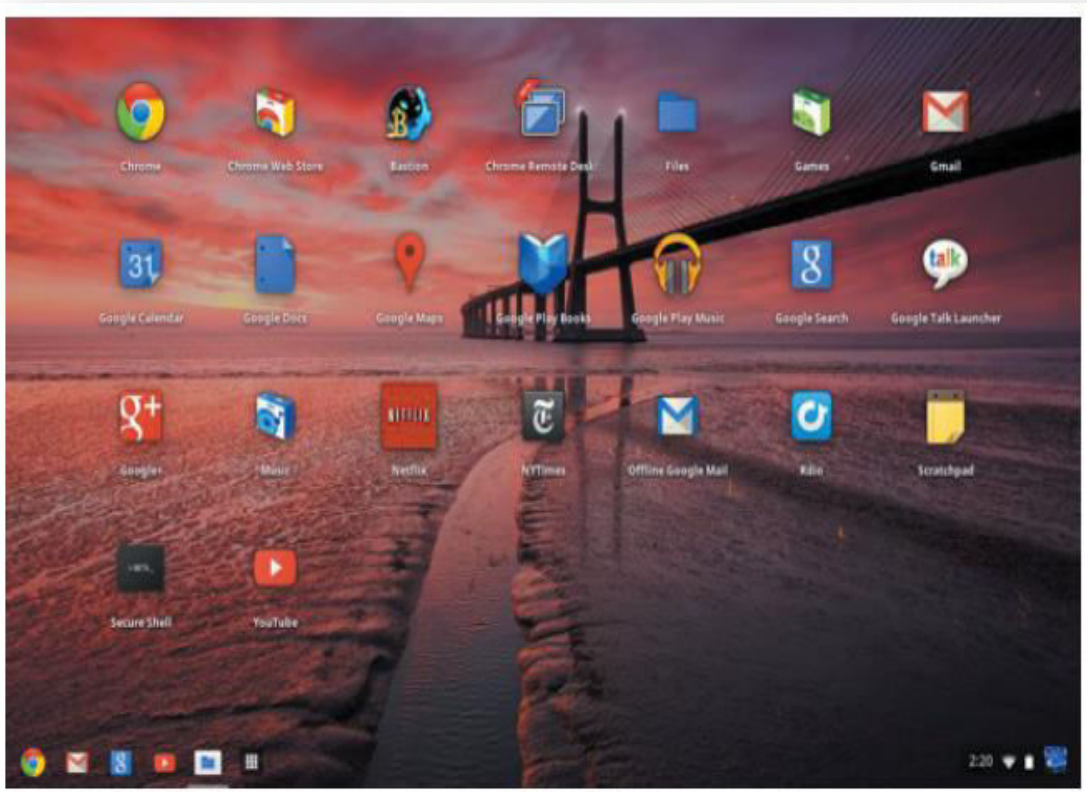
Embedded Operating Systems Cloud Operating Systems

- **iCloud: Apple's iCloud is built into every new iOS device**

After logging in with your Apple ID, the iCloud website connects you to Mail, Contacts, Calendar, Find My iPhone, and Documents from any other Apple device or PC.

Embedded Operating Systems, Cloud Operating Systems

- **Chrome OS: Google's Chrome OS is a Linux-based operating system available on specific hardware called Chromebooks**



desktop image from version 19 of Google's Chrome OS

Tasks of an Operating System

- Processor management - Scheduling
  - Fairness
  - Non-blocking behavior
  - Priorities
- Memory management
  - Virtual versus physical memory, memory hierarchy
  - Protection of competing/conurrent programs
- Storage management – File system
  - Access to external storage media
- Device management
  - Hiding of hardware dependencies
  - Management of concurrent accesses
- Batch processing
  - Definition of an execution order; throughput maximization

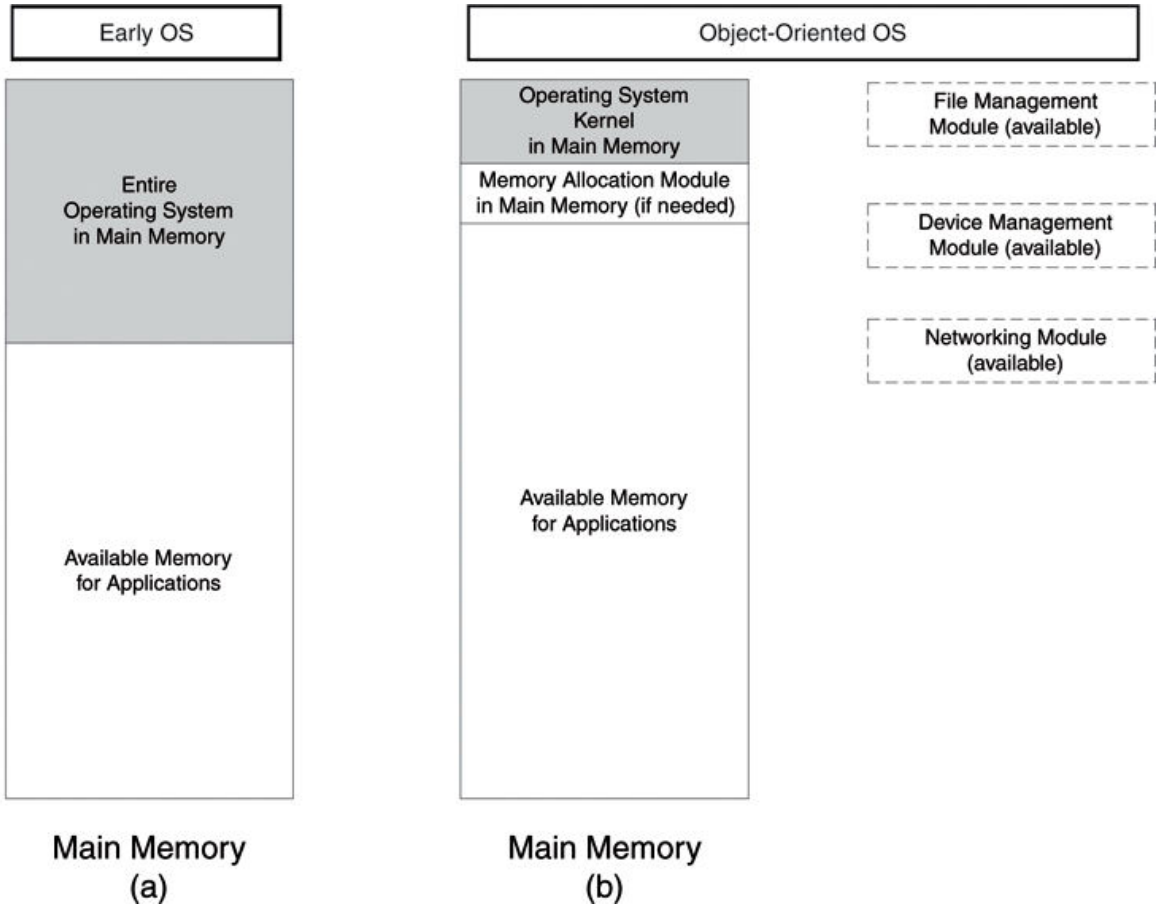
Object-Oriented Design

- **Driving force in system architecture improvements**
  - **Kernel (operating system nucleus)**
    - Resides in memory at all times, performs essential tasks, and protected by hardware
  - **Kernel reorganization**
    - Memory resident: process scheduling and memory allocation
    - Modules: all other functions
  - **Advantages**
    - Modification and customization without disrupting integrity of the remainder of the system
    - Software development more productive

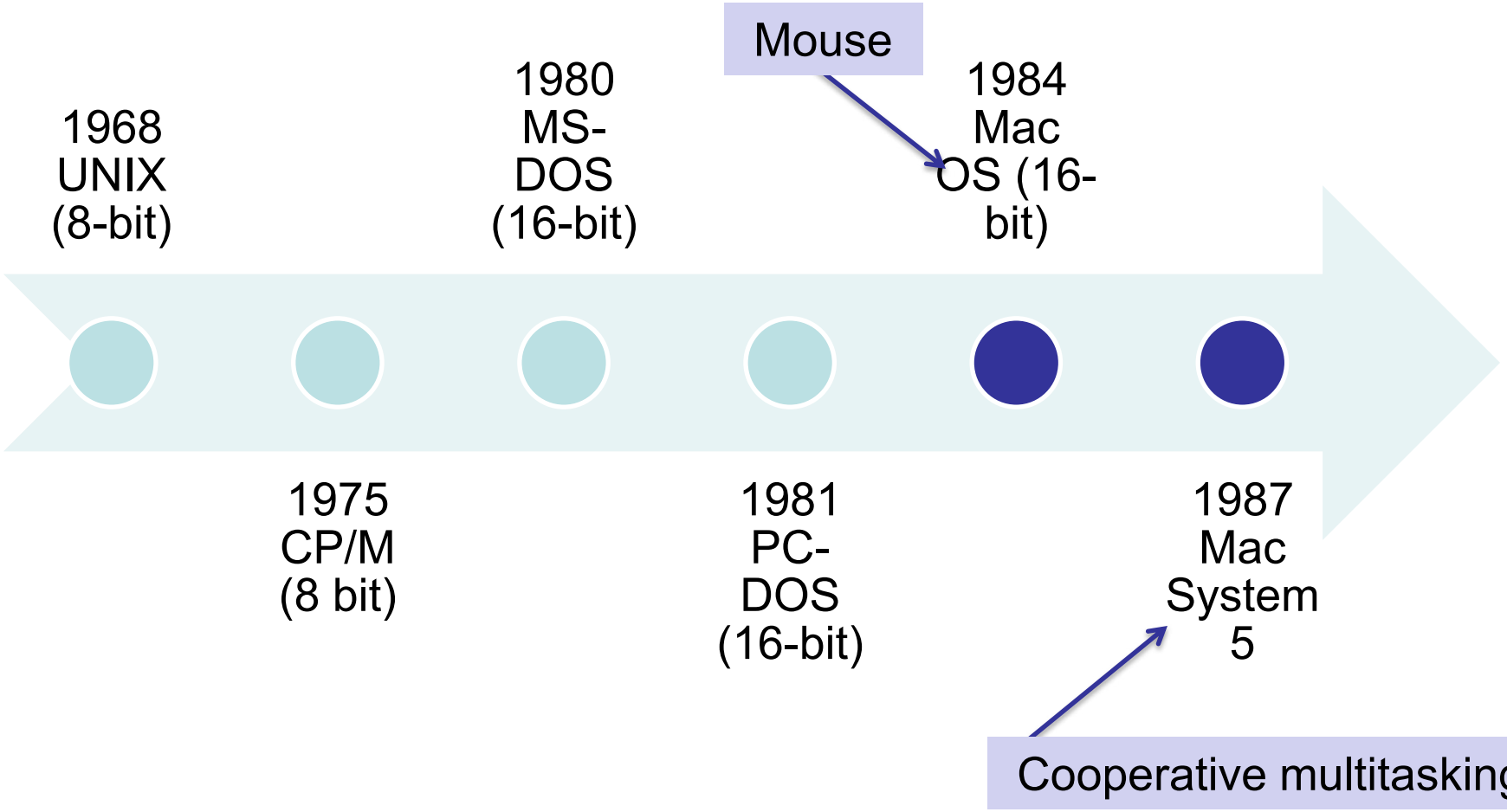
Object-Oriented Design (continued)

(figure 1.13)

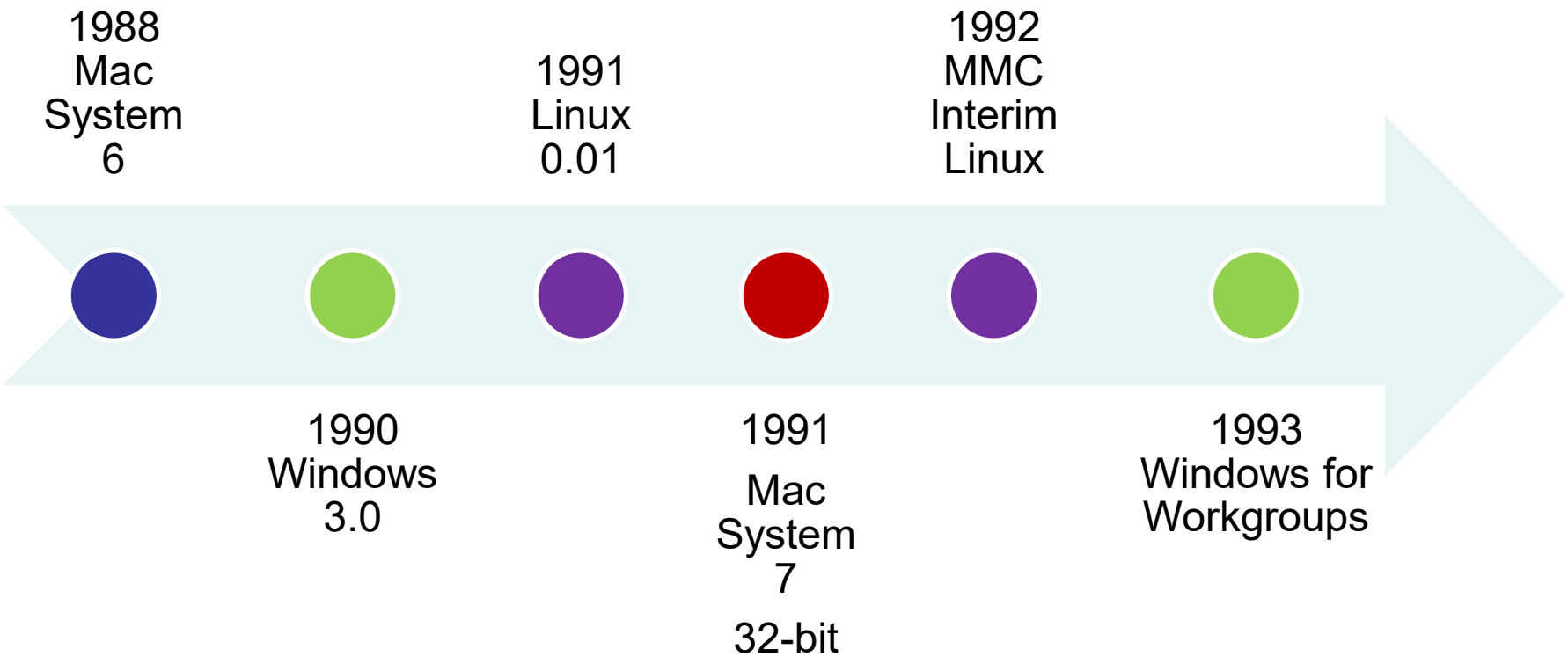
Early operating systems (a) loaded in their entirety into main memory. Modern object-oriented systems (b) load only the critical elements into main memory and call other objects as needed.



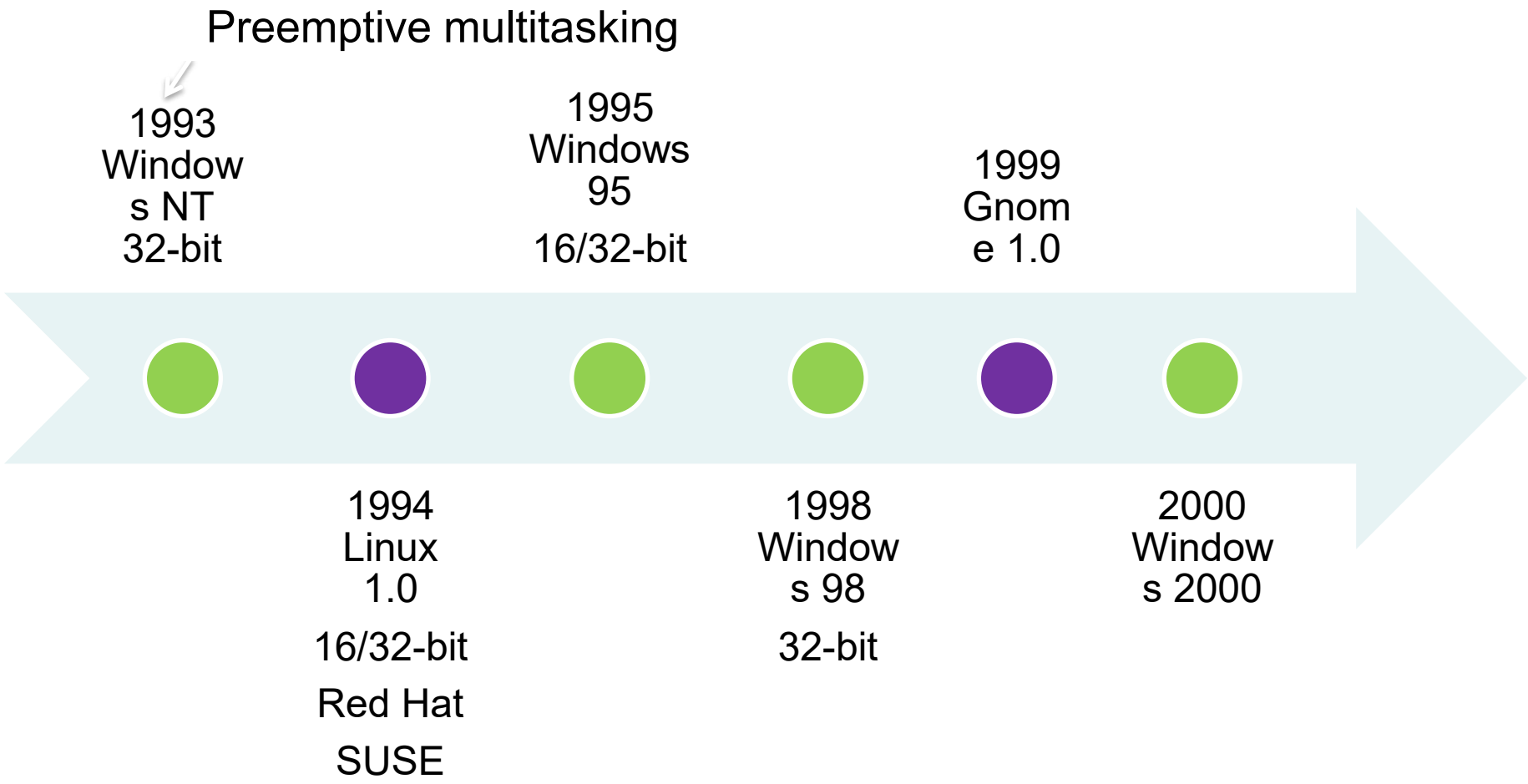
History of Desktop Operating Systems



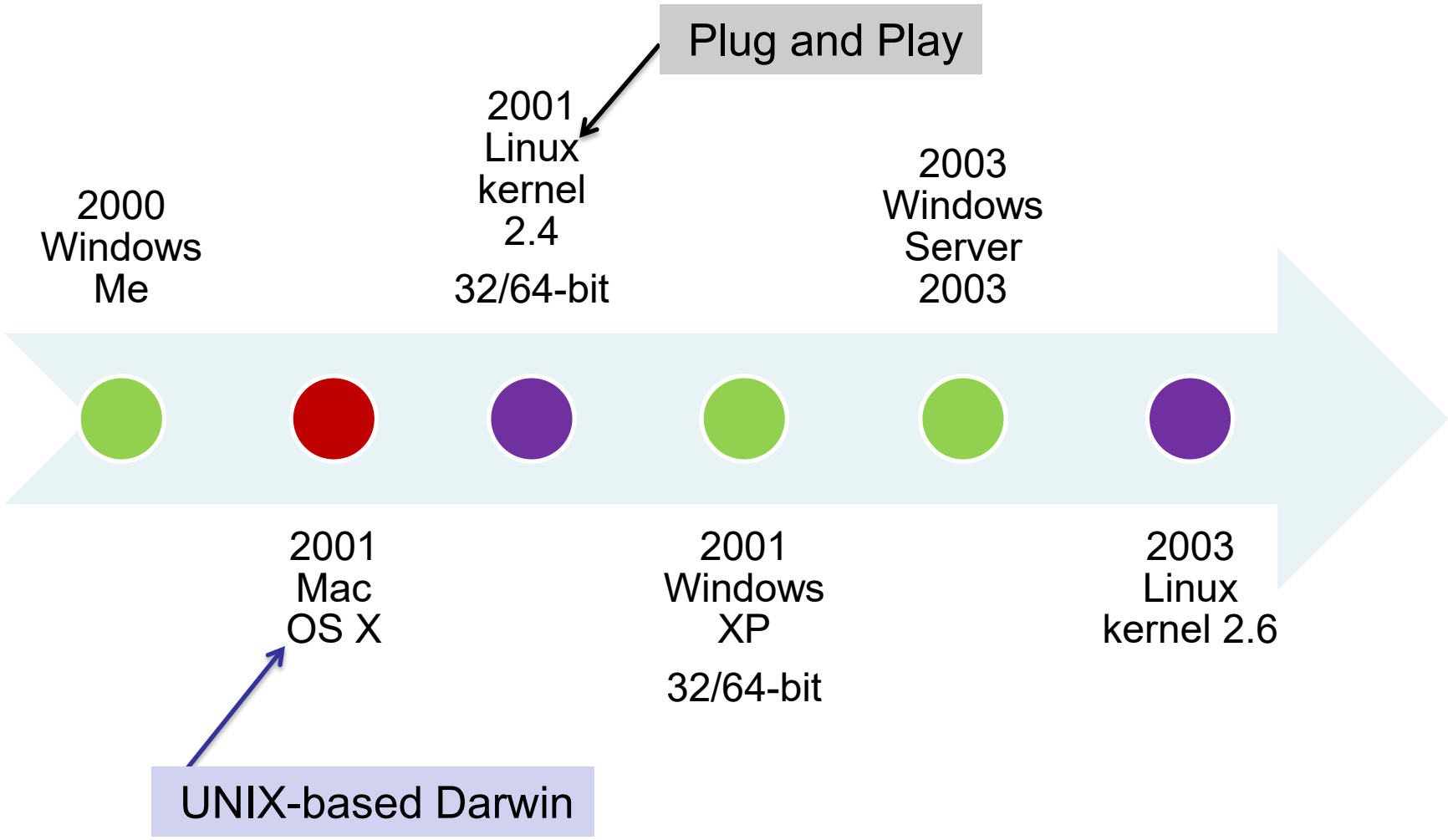
History of Desktop Operating Systems



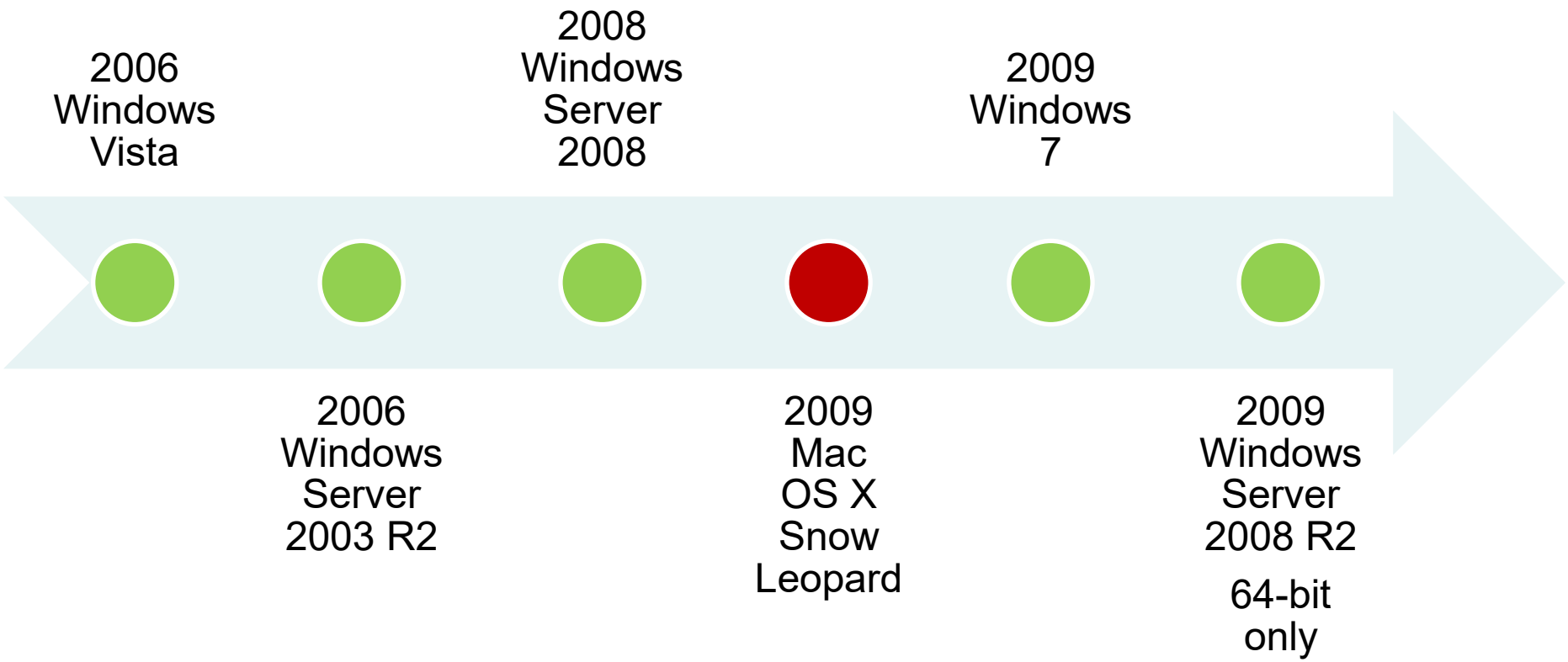
History of Desktop Operating Systems



History of Desktop Operating Systems



History of Desktop Operating Systems



History of Desktop Operating Systems

History of Desktop Operating Systems