System Programs

- System programs provide a **convenient environment for program development and execution**. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs

System Programs

- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the **system for info** - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files

System Programs (Cont.)

- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

System Programs (Cont.)

- **Background Services**
  - Launch at boot time
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons**
- **Application programs**
  - Don't pertain to system , Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Operating System Design and Implementation

- 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

Operating System Design and Implementation (Cont)

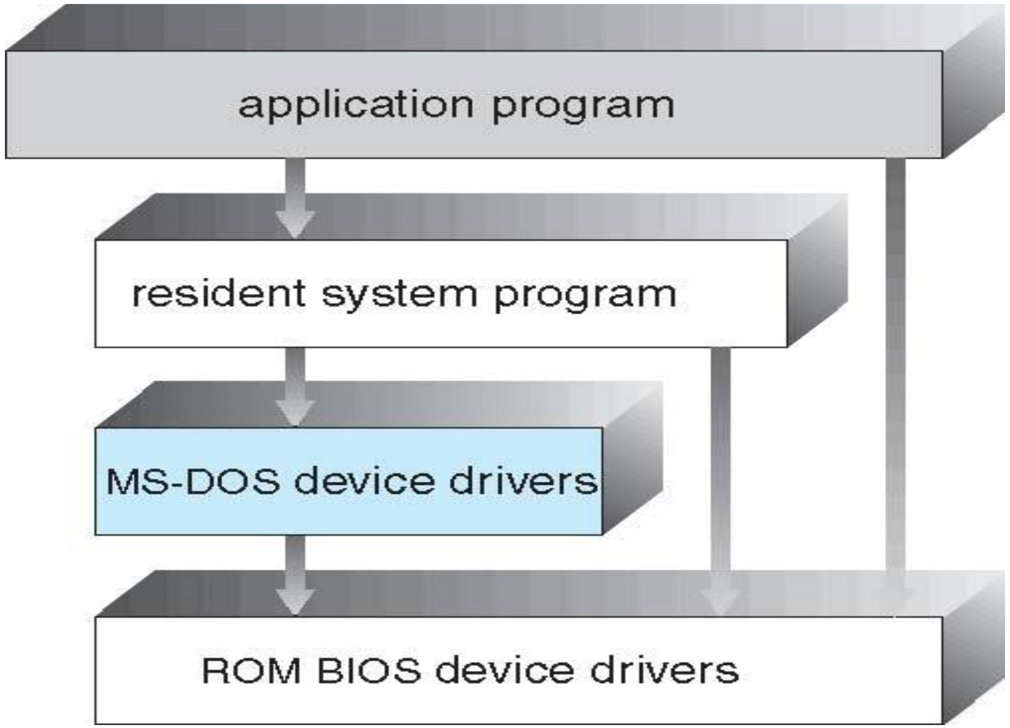
- Important principle to separate
- **Policy:** What will be done?
  Mechanism: How to do it?
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Operating System Structure

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

Simple Structure -- MS-DOS

- **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



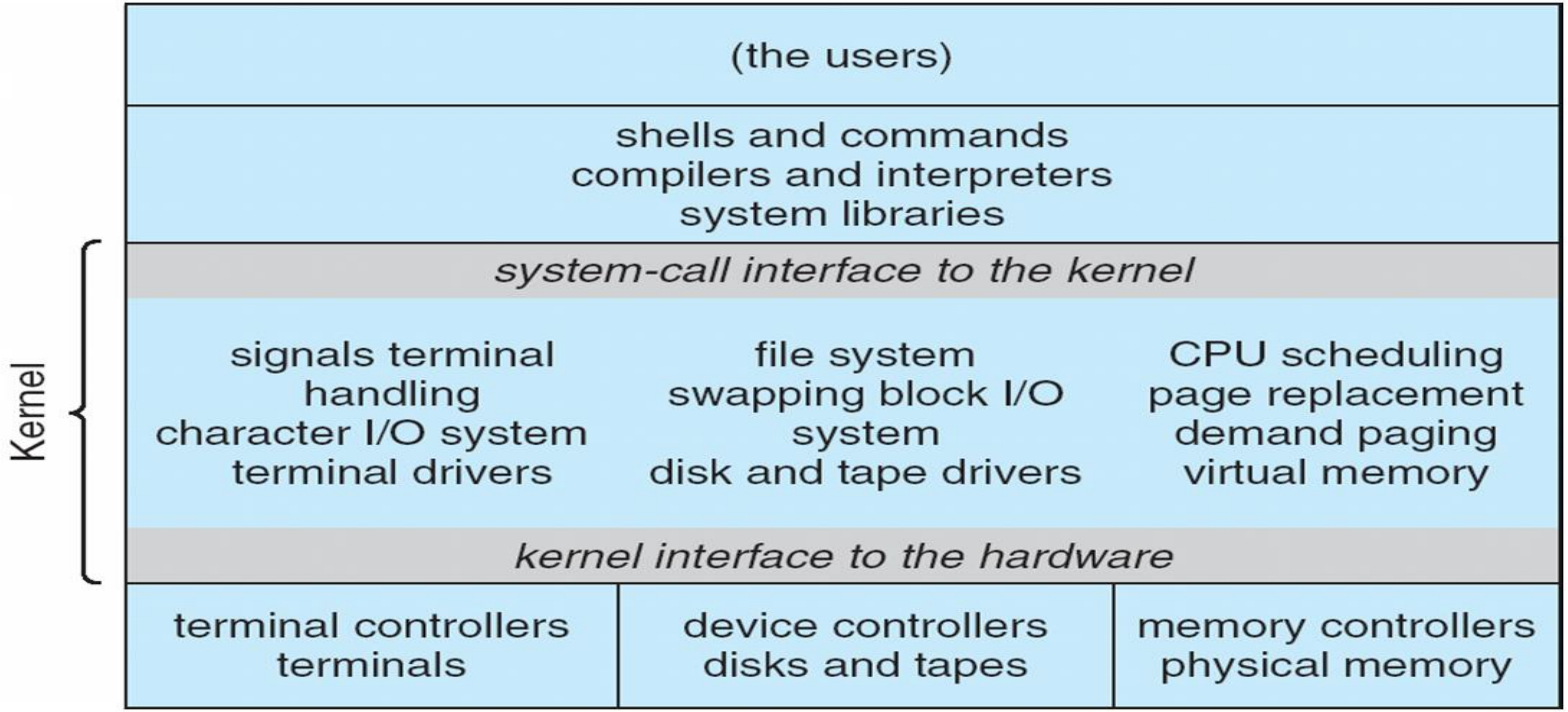
Non Simple Structure -- UNIX

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

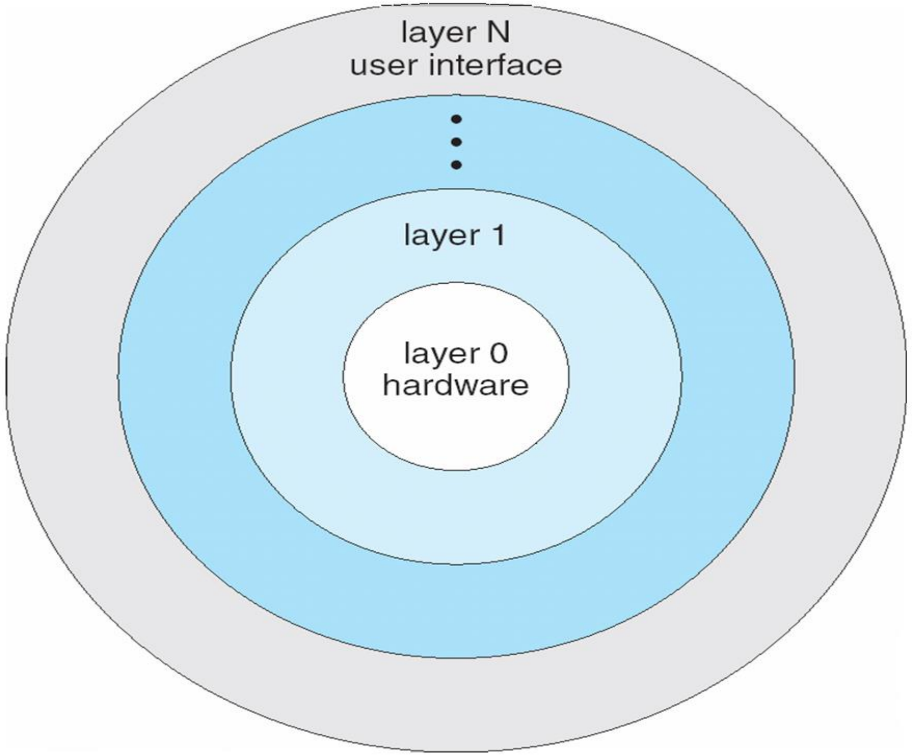
Traditional UNIX System Structure

Beyond simple but not fully layered



Layered Approach

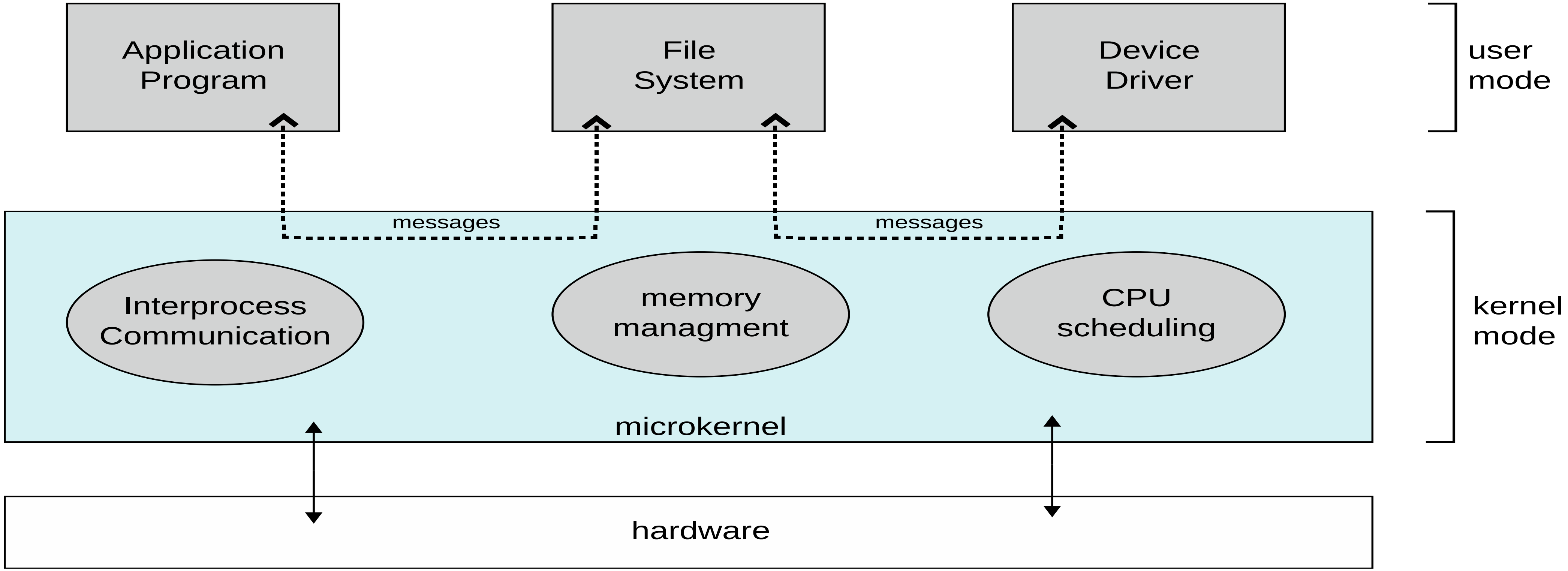
- The operating system is divided into a **number of layers (levels)**, each built on top of lower layers. The bottom layer (layer 0), is the **hardware**; the highest (layer N) is the **user interface**.
- With modularity, layers are selected such that each uses functions (operations) and services of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



Microkernel System Structure

- **Mach** example of **microkernel**
- 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

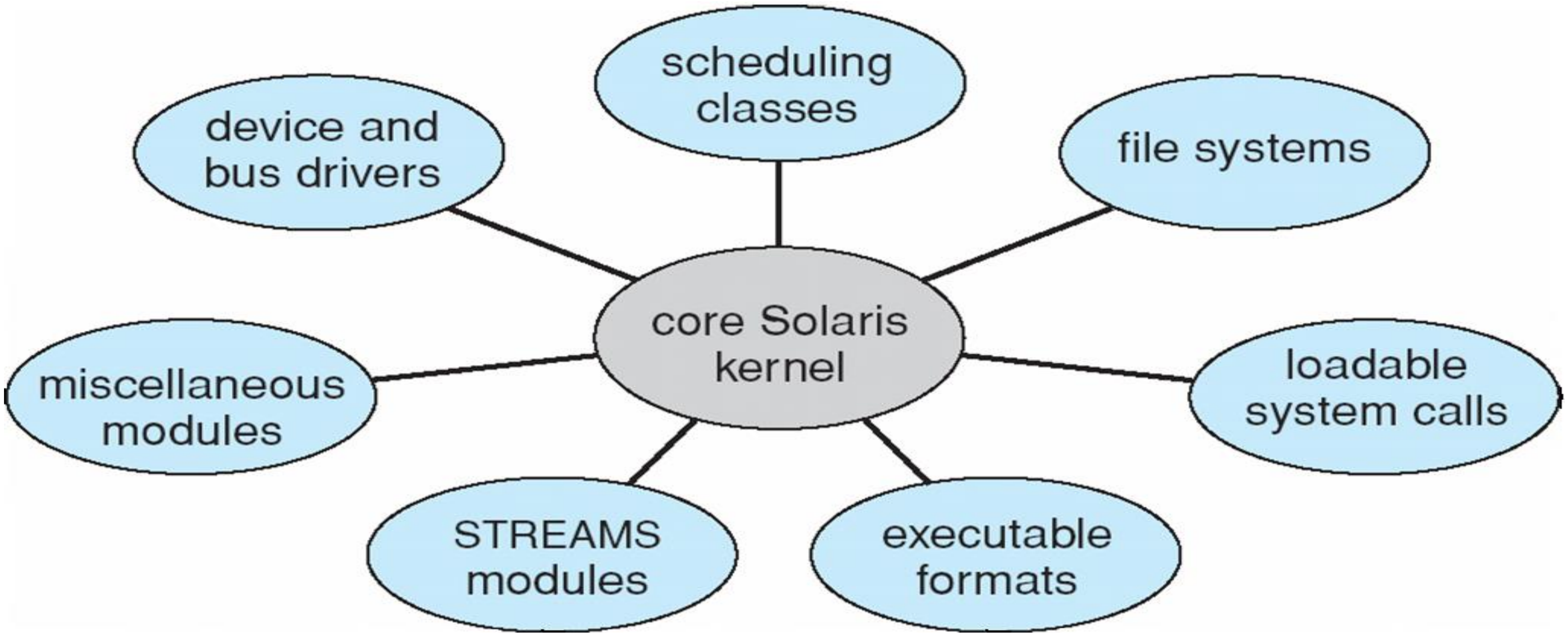
Microkernel System Structure



Modules

- Many modern operating systems implement **loadable kernel modules**
  - 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

Solaris Modular Approach

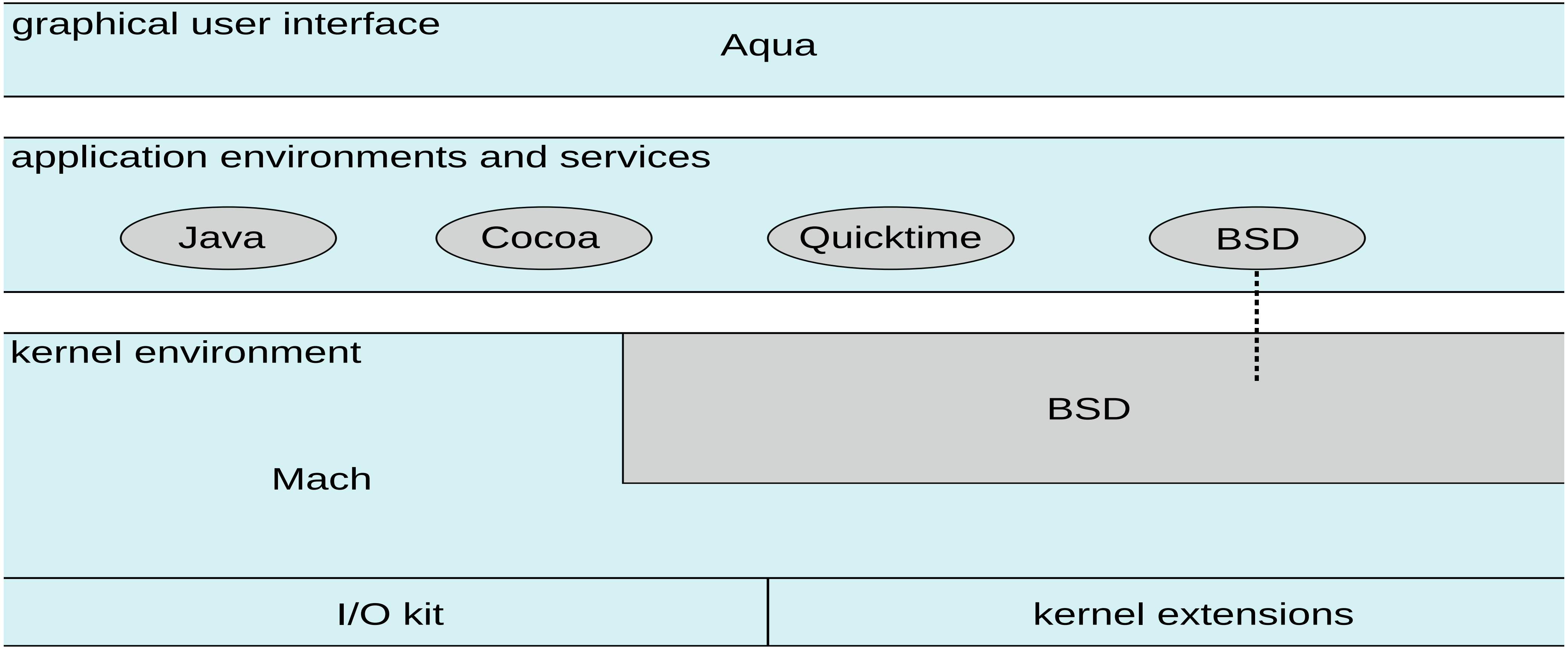


Hybrid Systems

- 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***

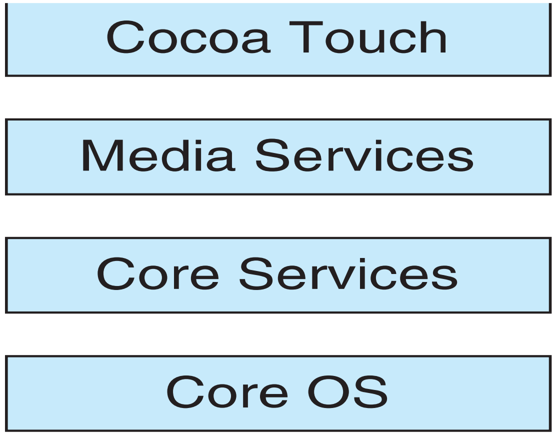
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment

Mac OS X Structure



iOS

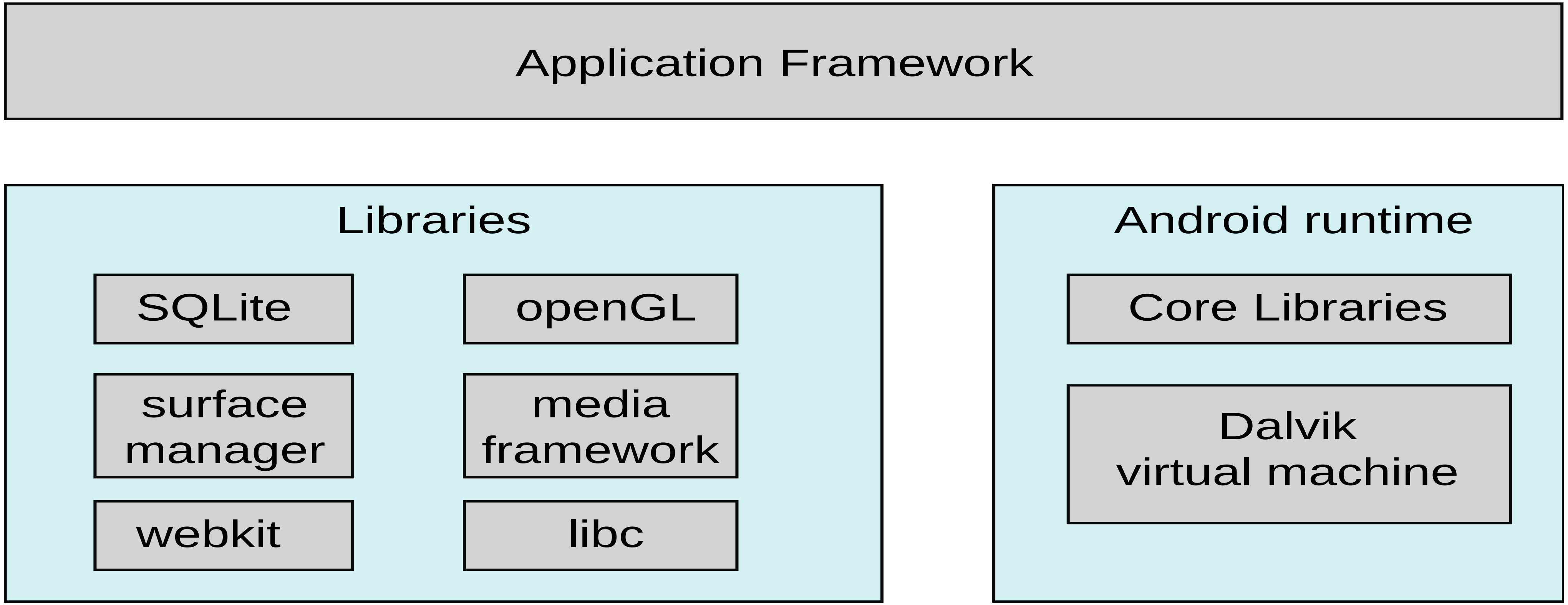
- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio,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



Android

- Developed by Open Handset Alliance (mostly Google) - Open Source
- 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



System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

TEXT BOOK

1. Abraham Silberschatz, Peter B. Galvin, “Operating System Concepts”, 10th Edition, John Wiley & Sons, Inc., 2018.

2. Andrew S Tanenbaum, Herbert Bos, Modern Operating Pearson , 2015.

REFERENCES

1. Ramaz Elmasri, A. Gil Carrick, David Levine, “ Operating Systems – A Spiral Approach”, Tata McGraw Hill Edition, 2010.
2. William Stallings, Operating Systems: Internals and Design Principles, 7th Edition, Prentice Hall, 2018
3. Achyut S.Godbole, Atul Kahate, “Operating Systems”, McGraw Hill Education, 2016

THANK YOU