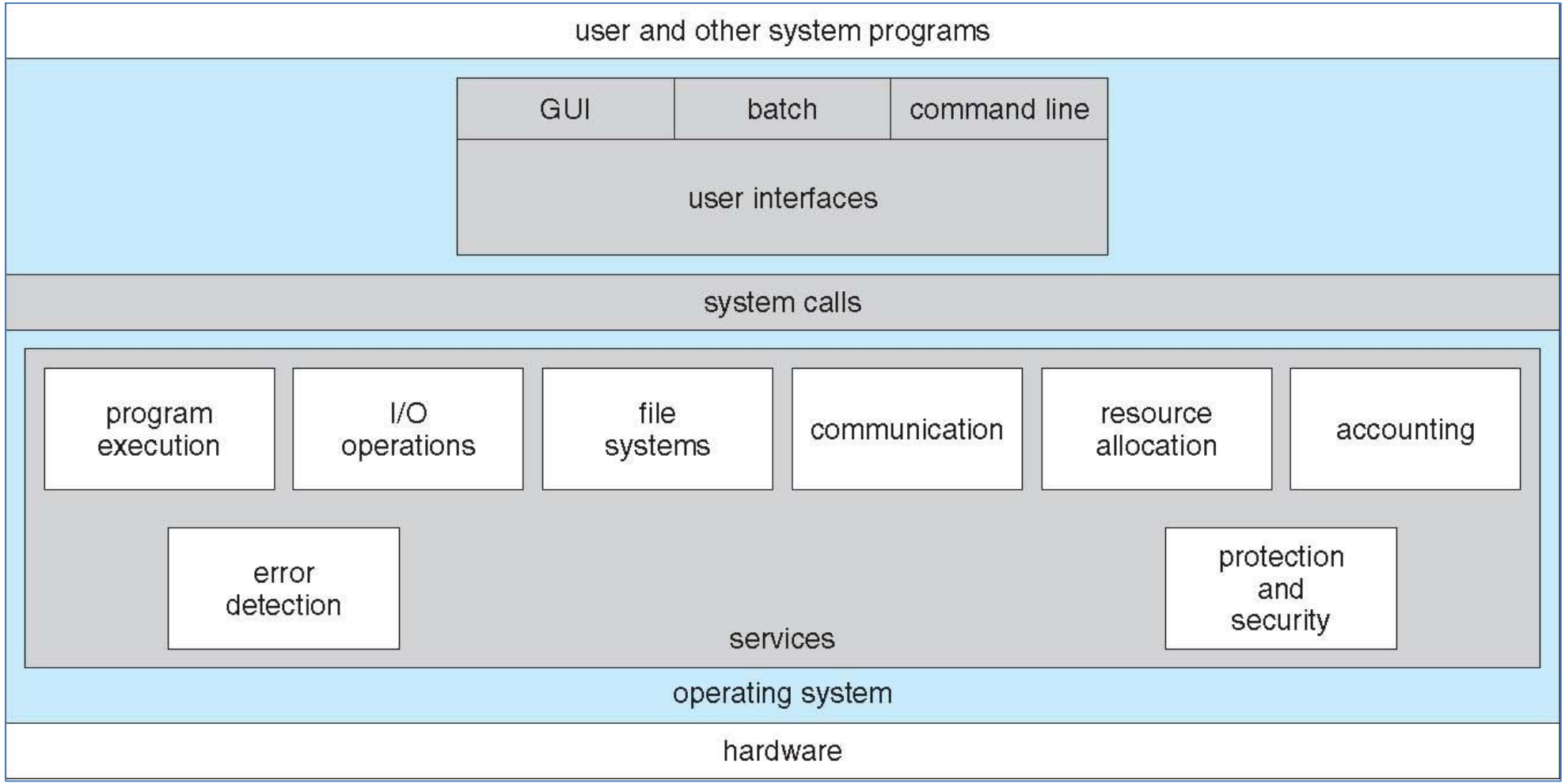
Computer and Operating Systems (COS)

Lecture 7

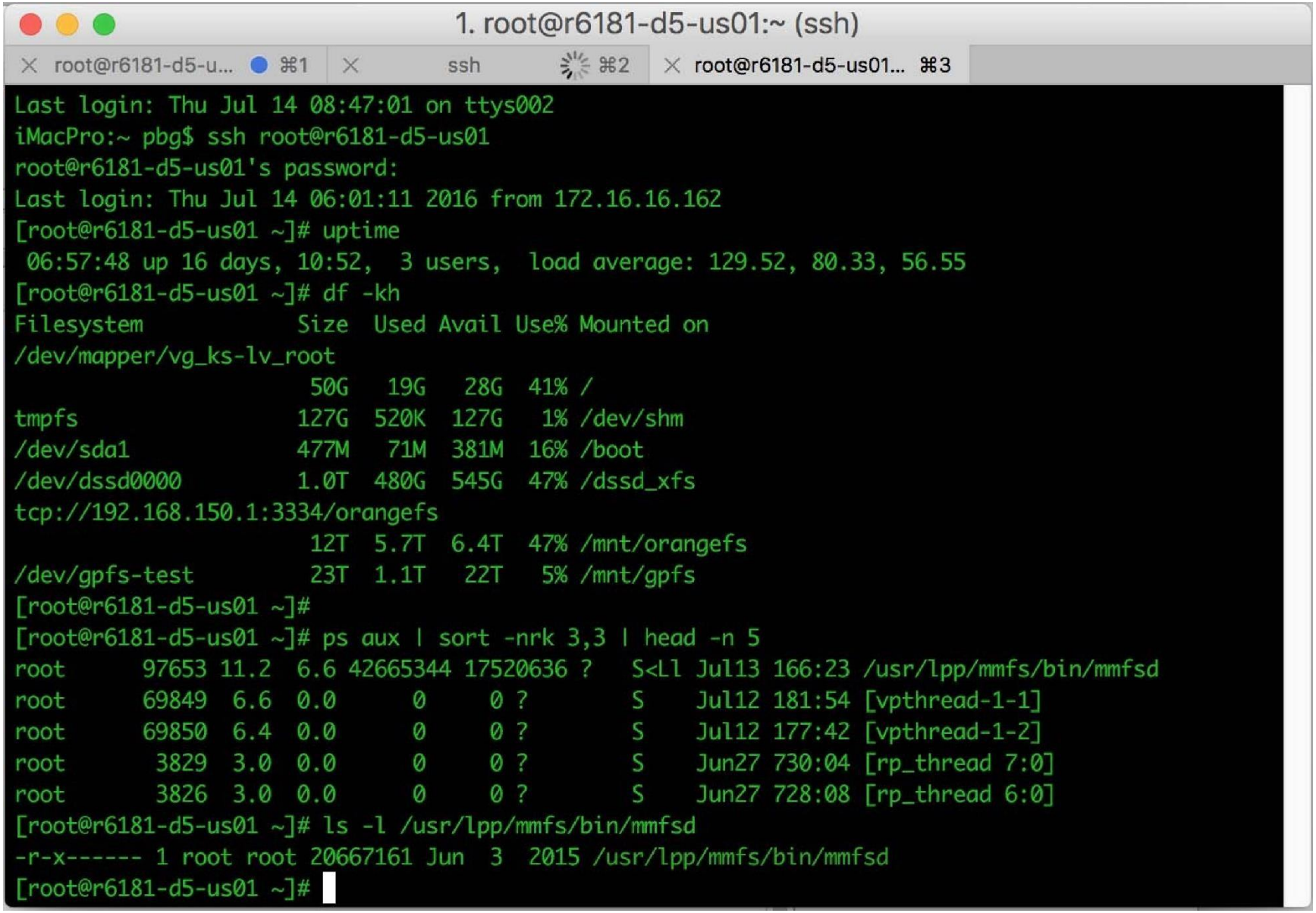
Overview of the contents

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Operating-System Structure
- Process Concept
- Process Scheduling
- Interprocess Communication

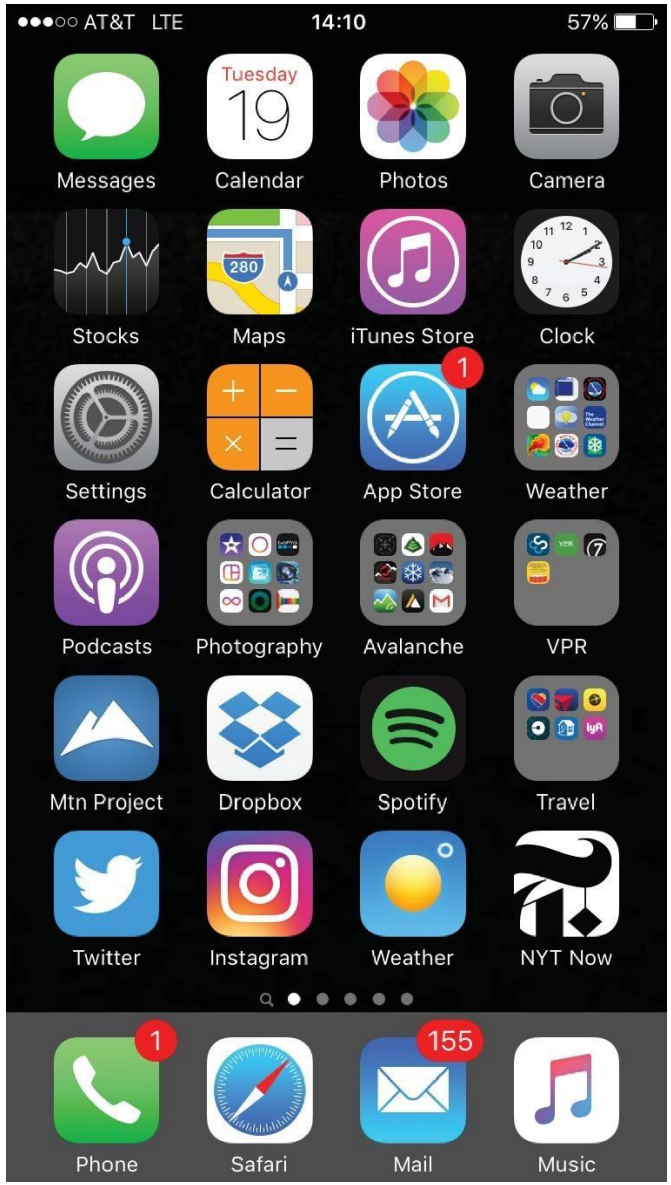
Overview of Operating System Service



Example of a CLI (**C**ommand **L**ine **I**nterface) or
a command interpreter

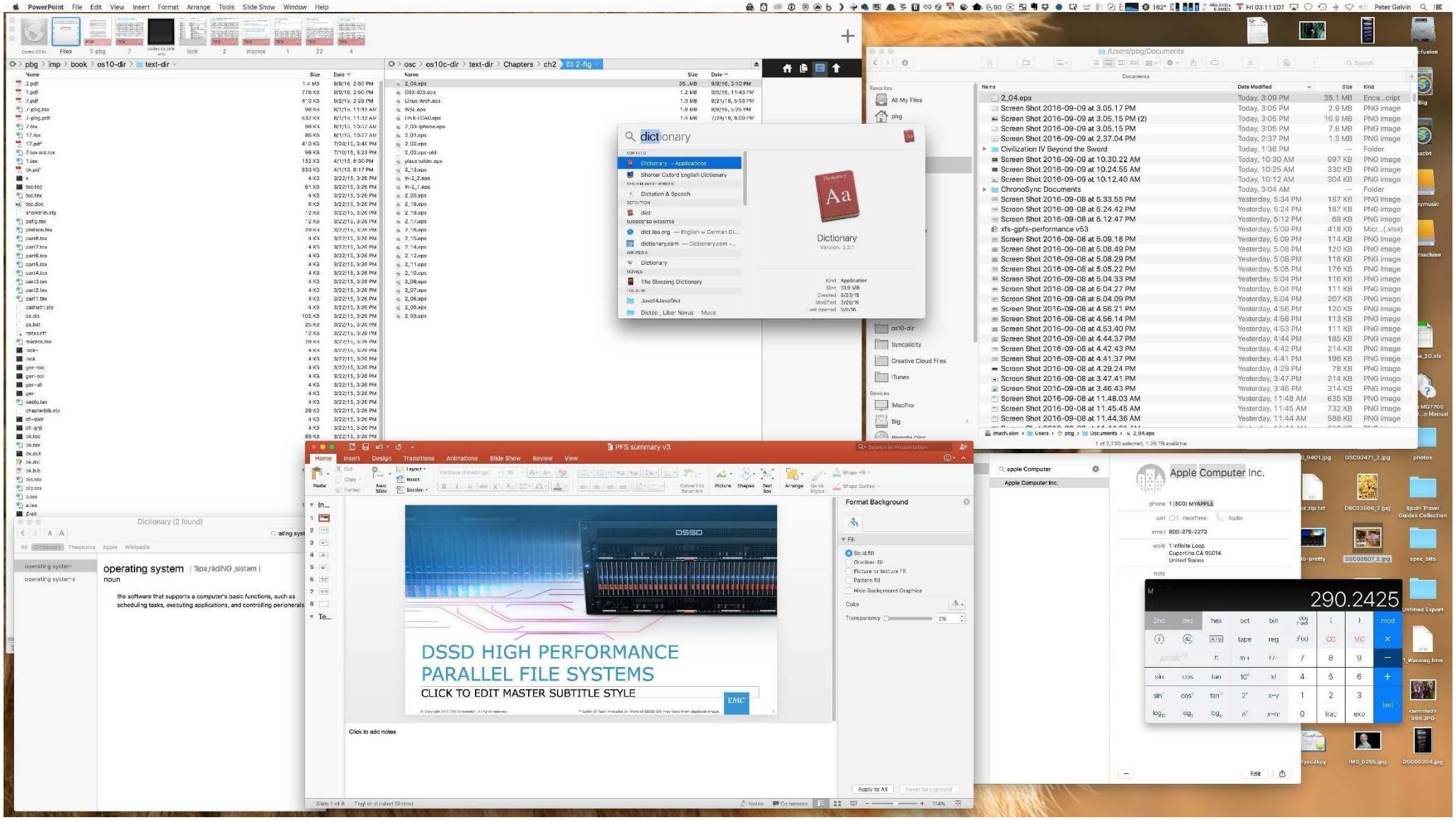


Examples of GUI (**G**raphical **U**ser **I**nterface)



IPhone GUI

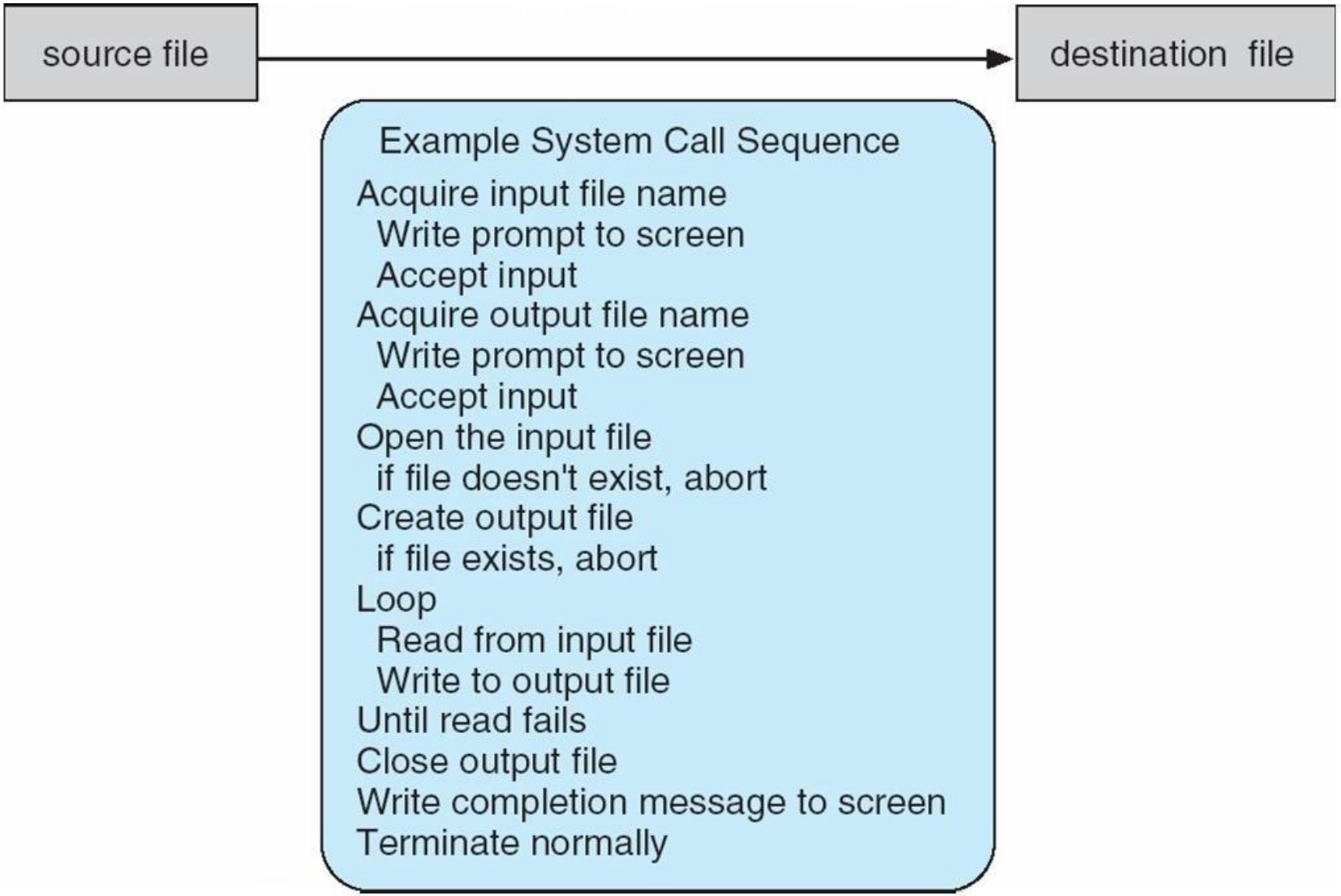
Examples of GUI (**G**raphical **U**ser **I**nterface)



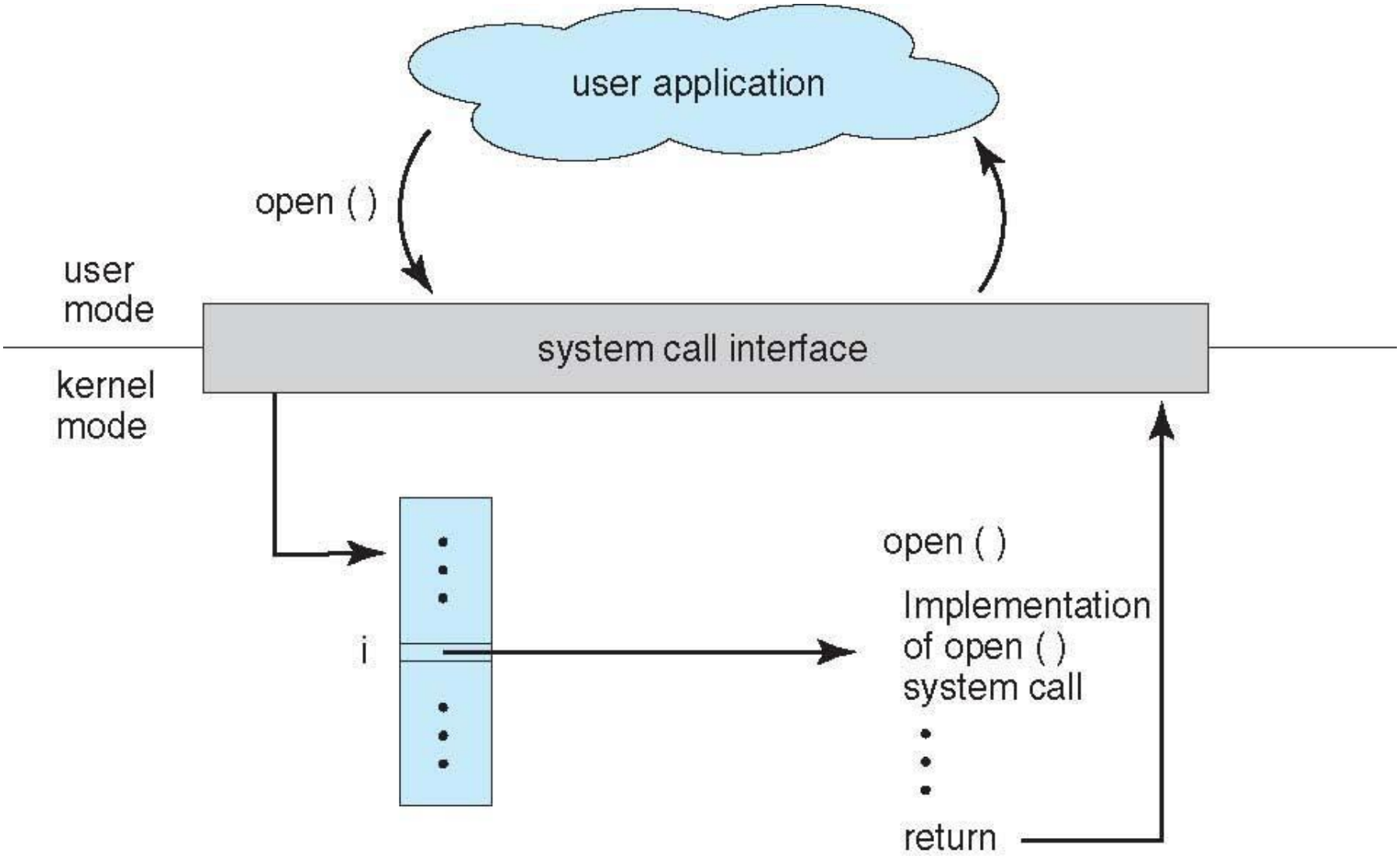
Mac OS GUI

System call

Example: COPY consists of many calls

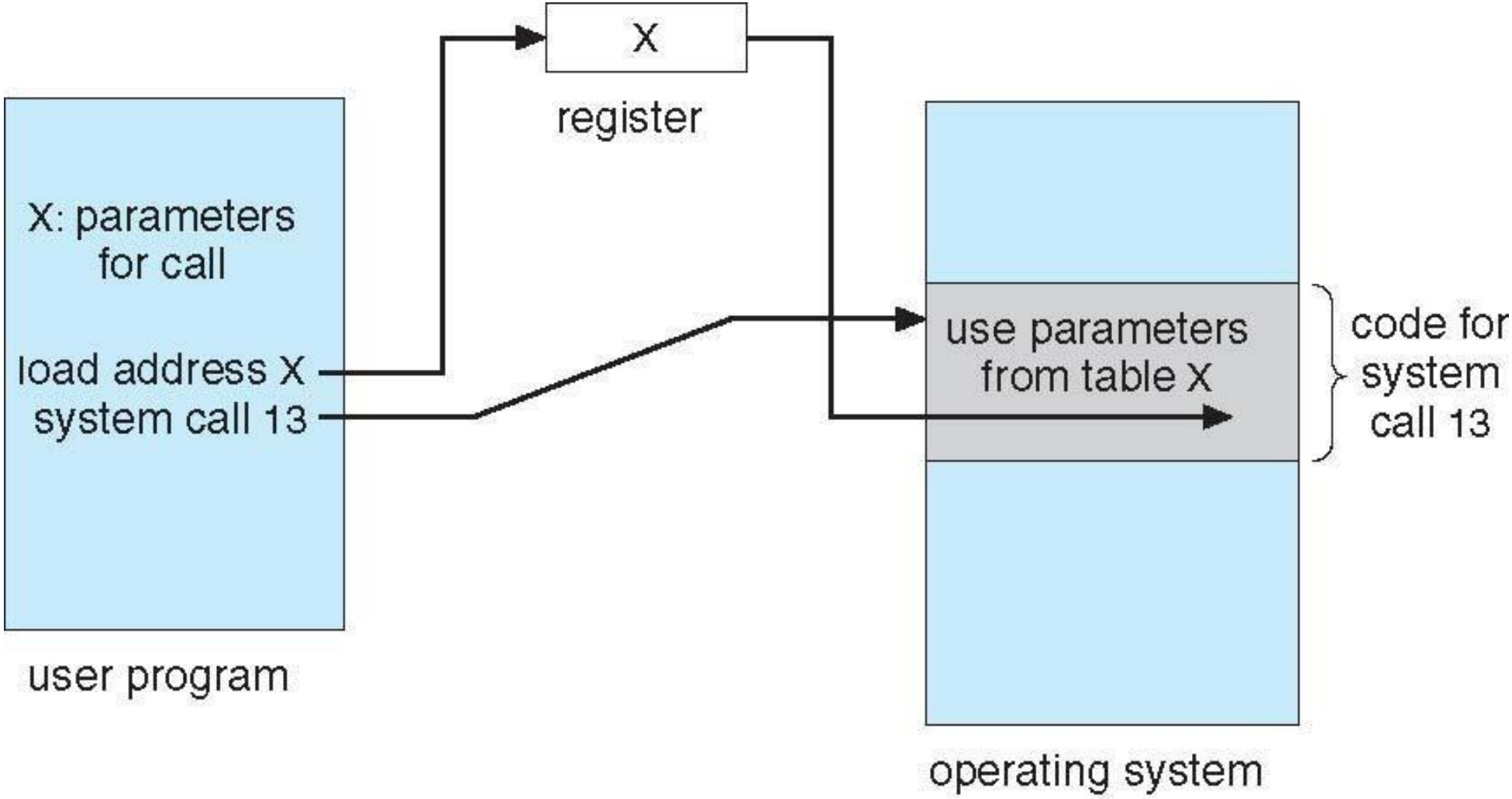


API (**A**pplication **P**rogramming **I**nterface)
typically, a standard library that serves as a link
between the programmer and the system call interface



Parameter transfer for system calls

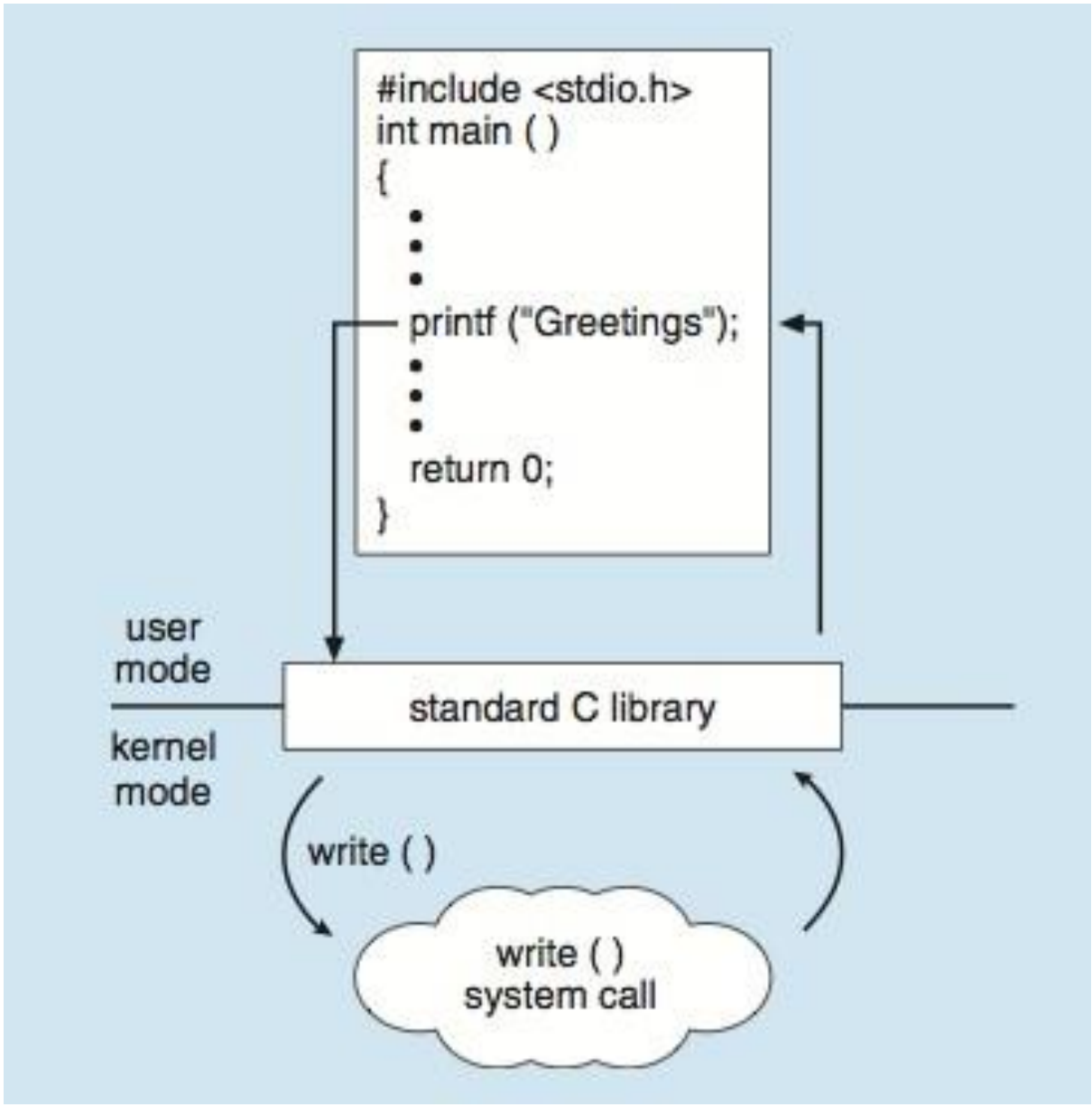
- Register directly
- Register indirectly via table
- Stack



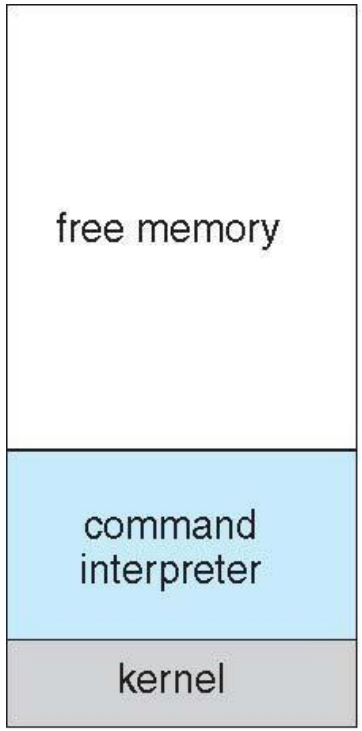
Examples of System Calls

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

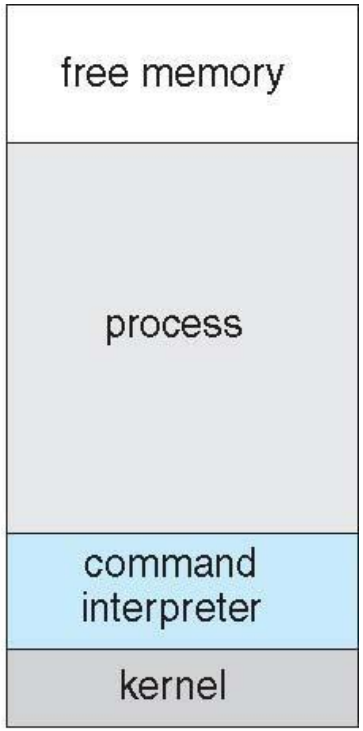
Example of C-library



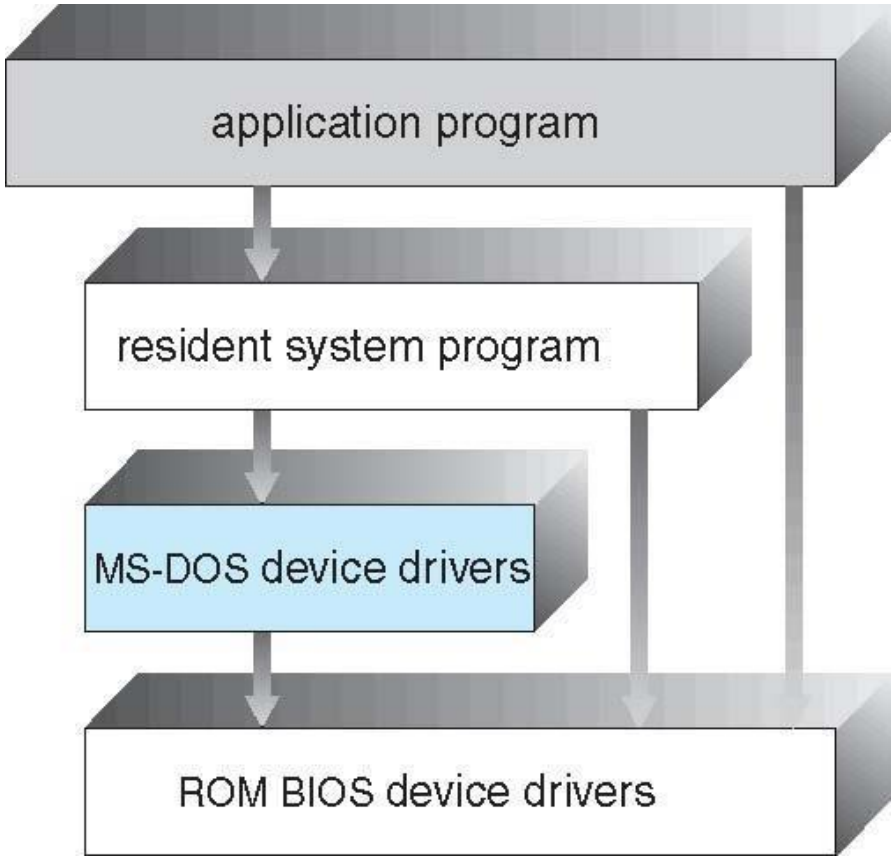
Example of MS-DOS



(a)

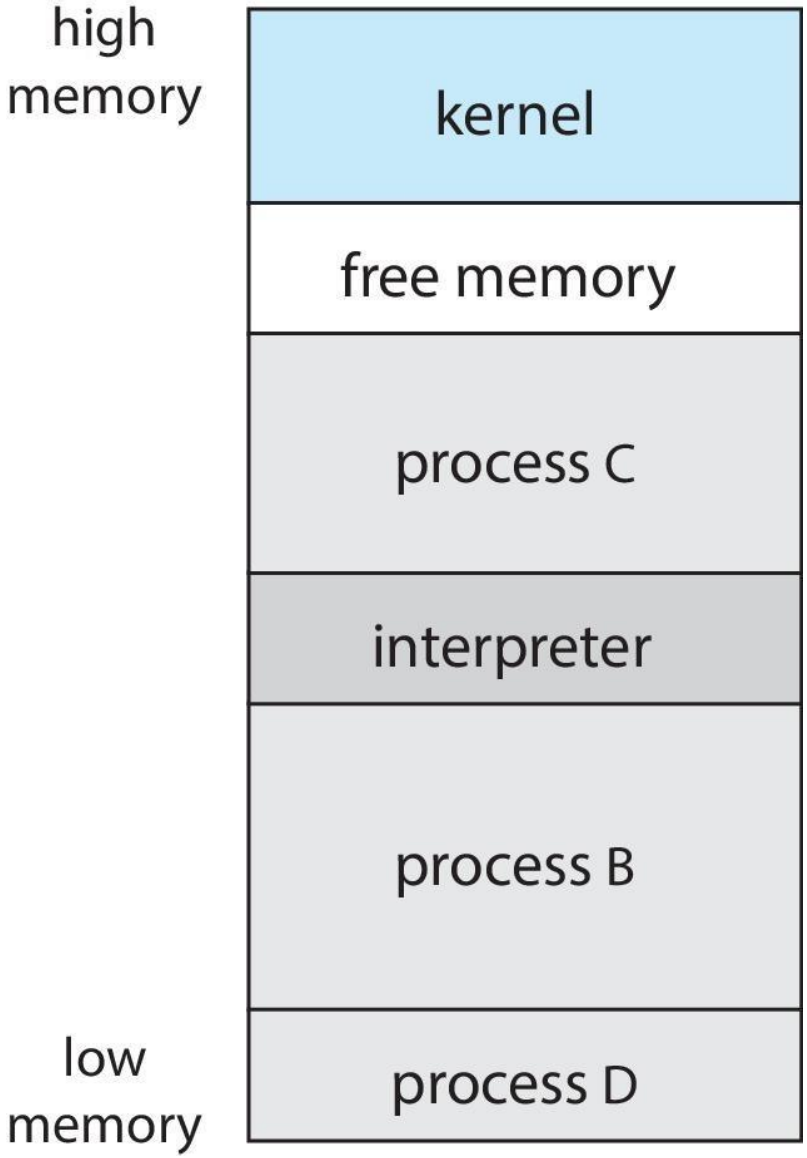


(b)



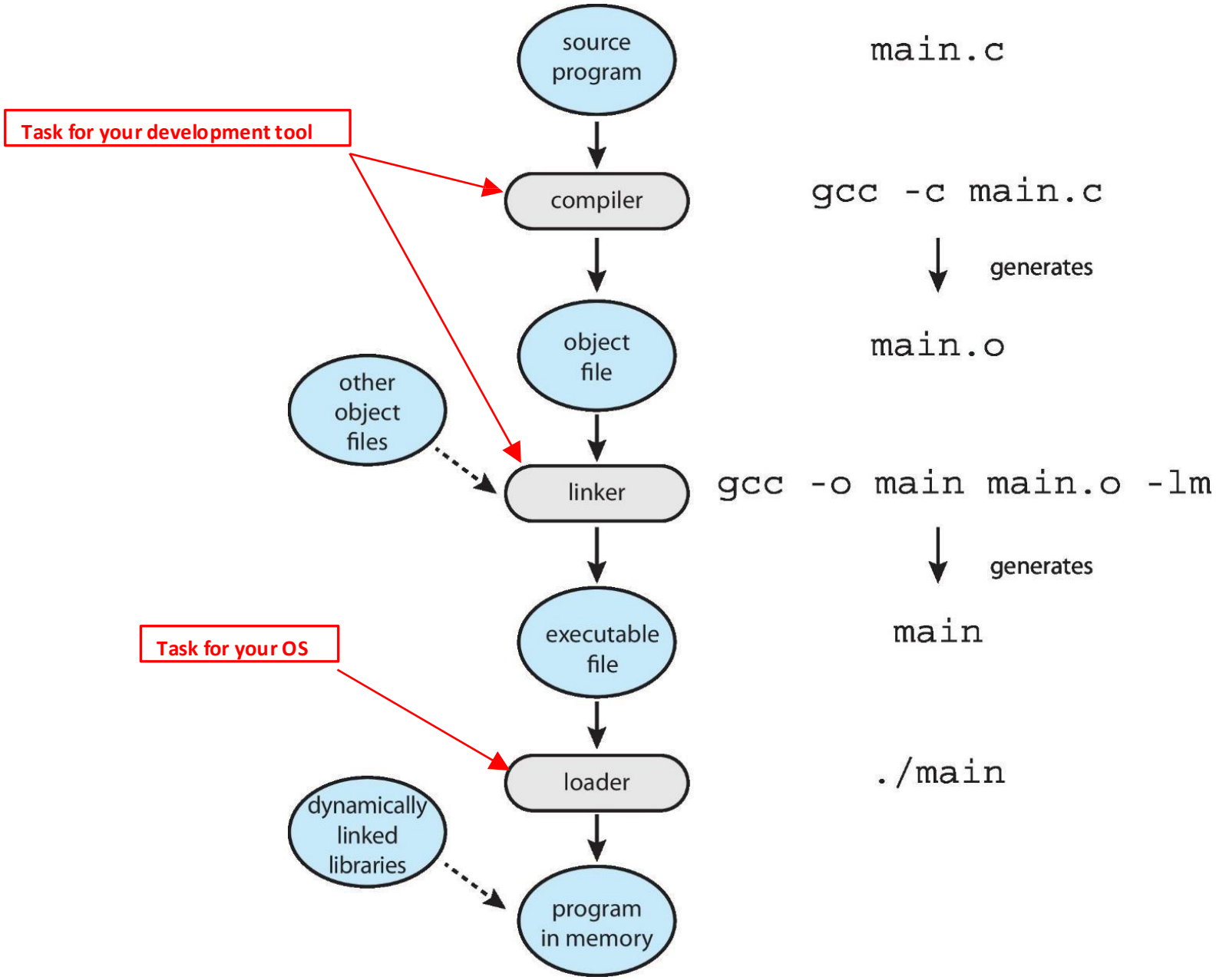
Single-tasking OS (Only one process at a time)

Example FreeBSD (a Unix variant)



Multi-tasking OS (multiple processes simultaneously)

Links and Loader

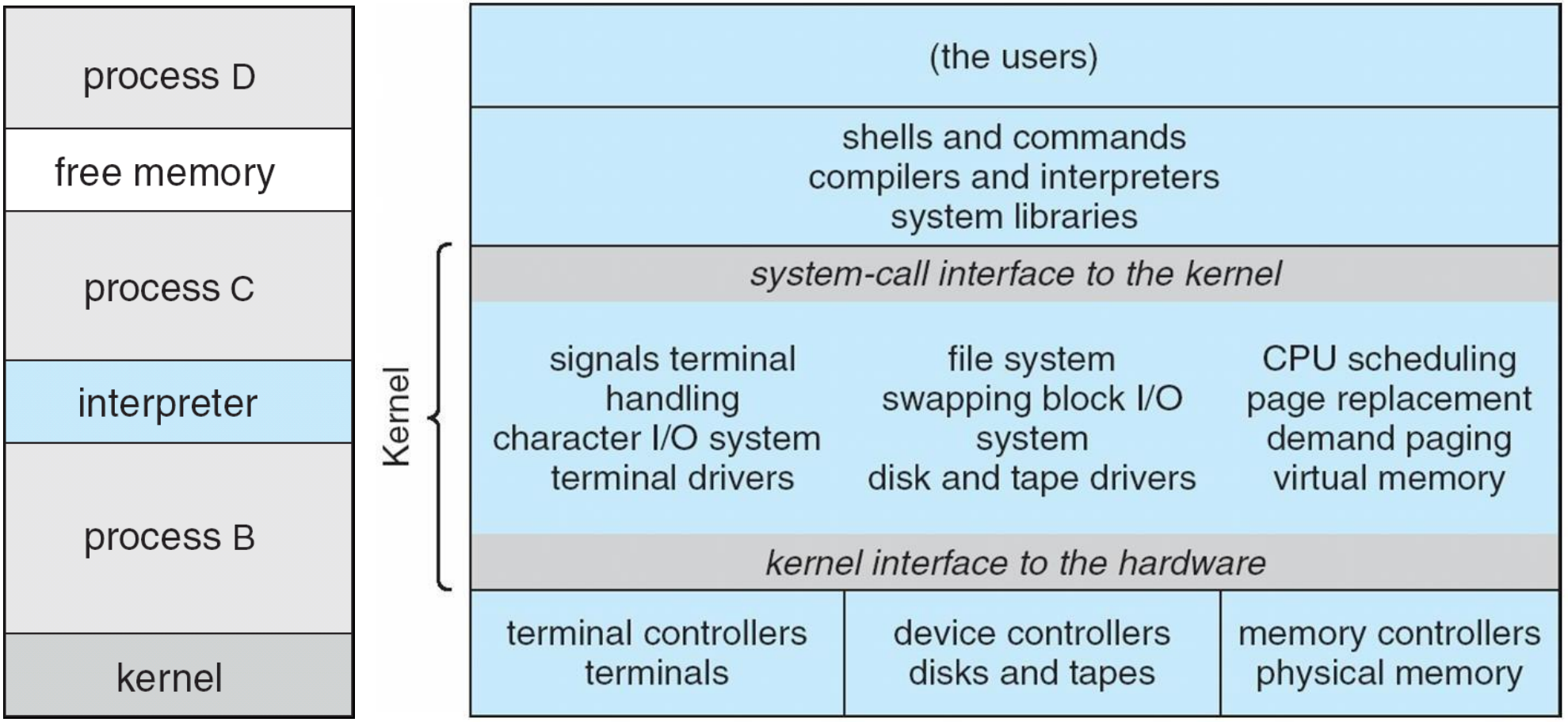


Once the code is compiled and linked together into a program, it is often the case that this program can only run on a specific OS. But …

- The source code and thus the program can be ported to another OS if there is a compiler for this platform.

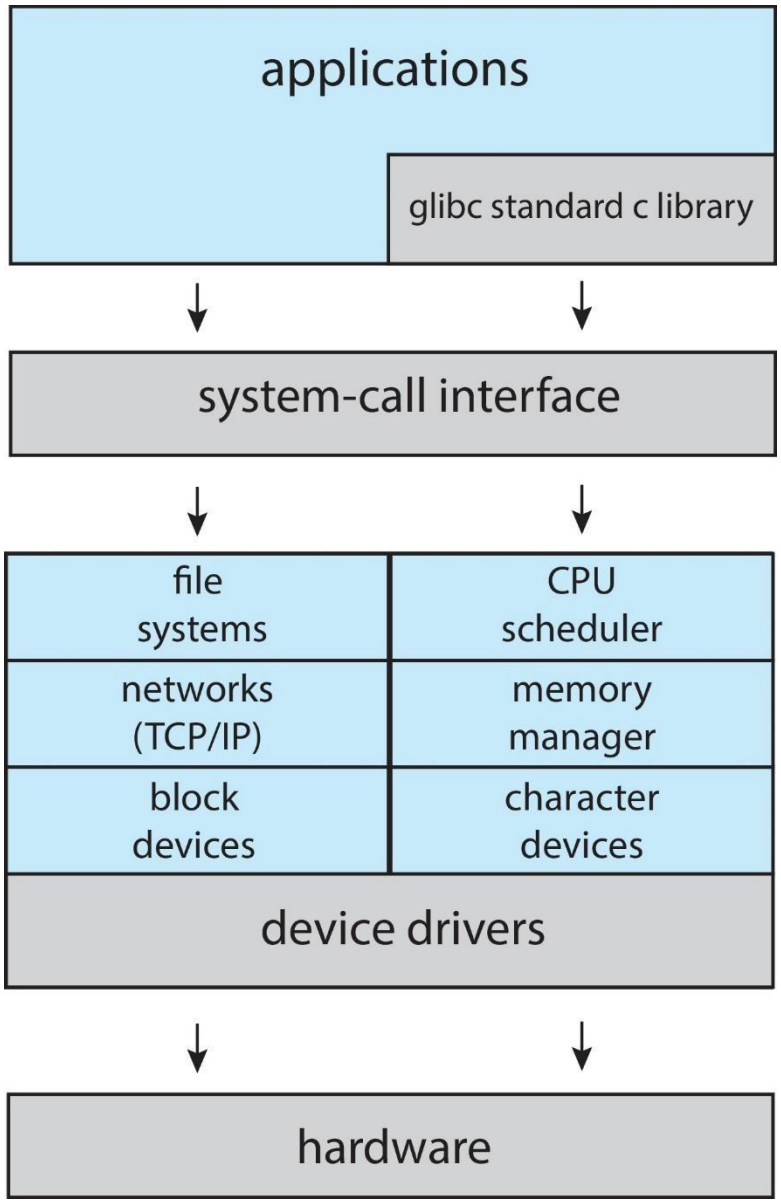
- The program can run on multiple OS if there is a virtual machine for the OS. A **virtual machine** is a layer that lies between the program and the OS. Here, the program's commands (API) are translated (interpreted) into target OS commands (system calls) through the virtual machine.

Example UNIX



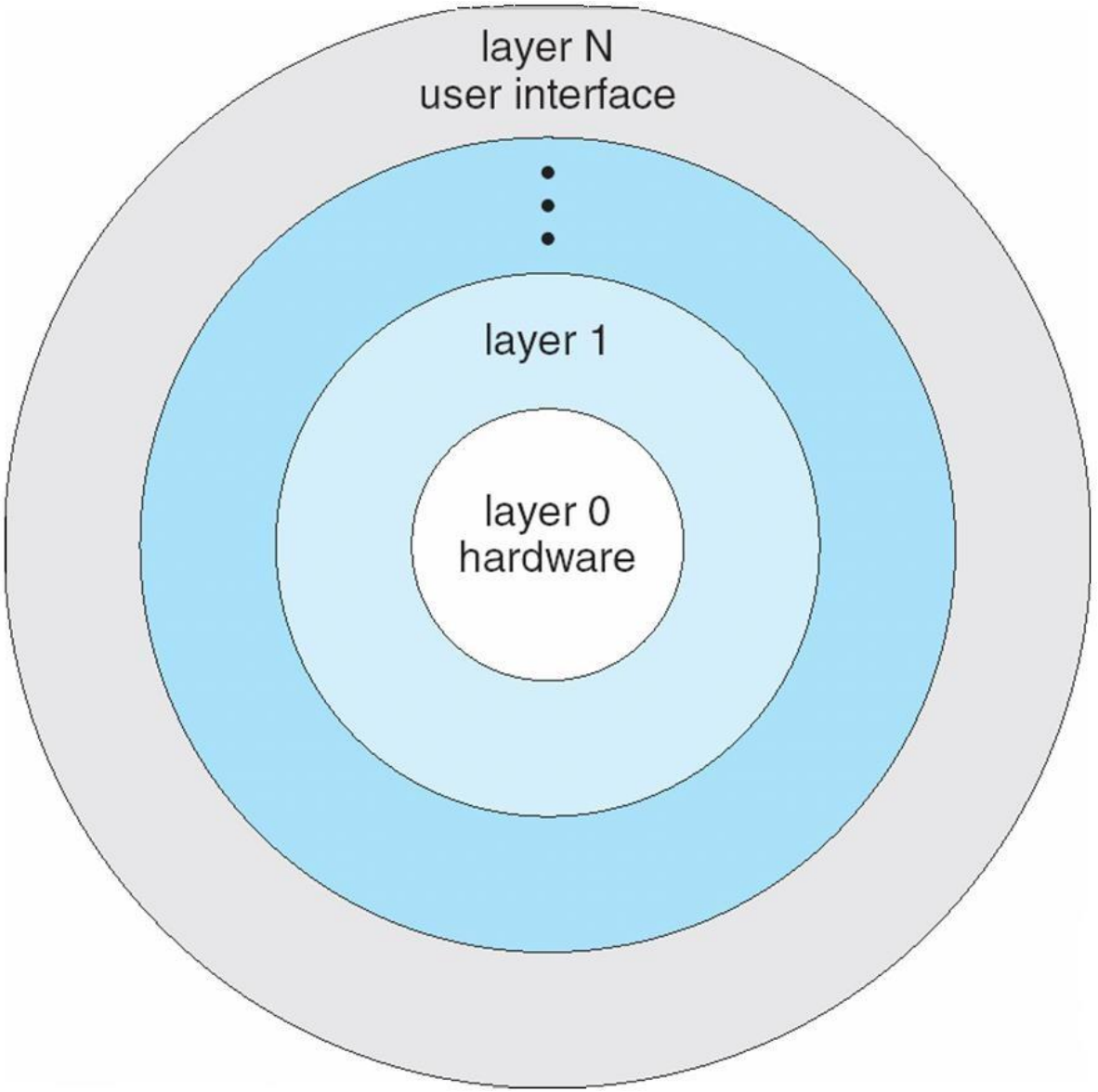
Monolithic structure

Example Linux



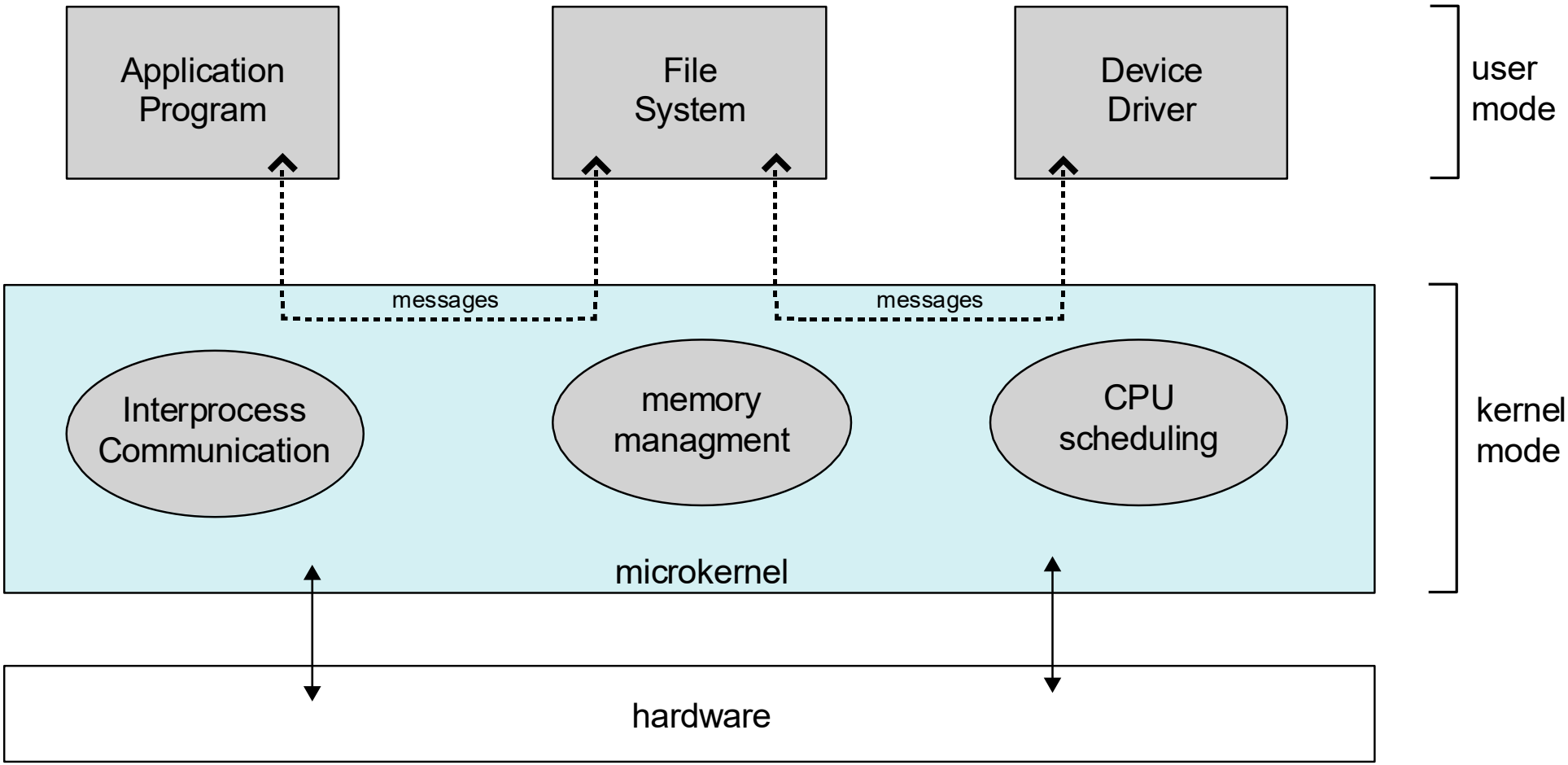
Monolithic plus modular

Example MULTICS



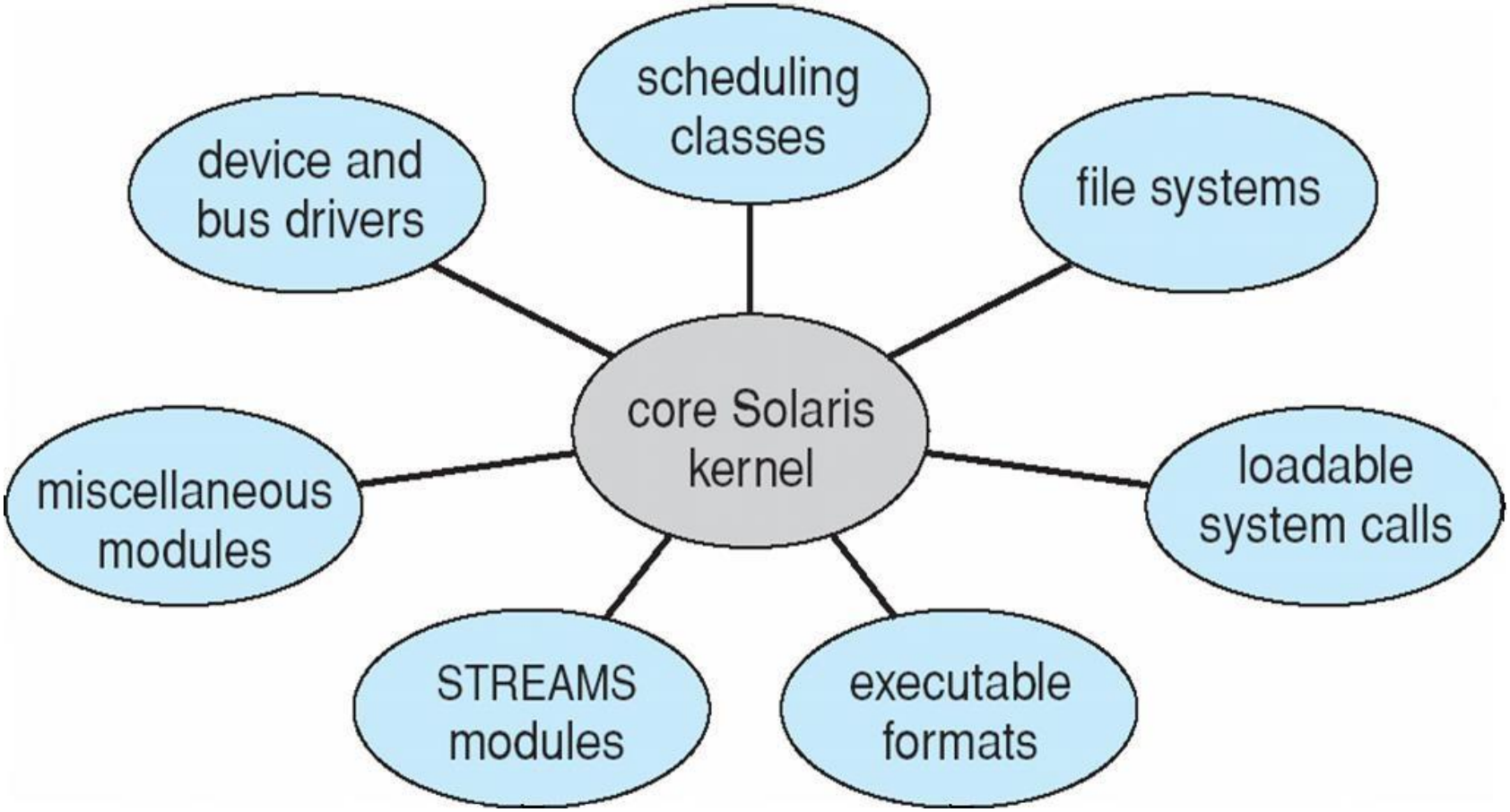
Layered structure

Example MACH



Microkernel (Mac OS is partially built on MACH)

Example Solaris



- Each component is independent
- Each component communicates with others over known interfaces
- Each component can be started within the kernel as needed.
- This is similar to layered architecture, but is more flexible

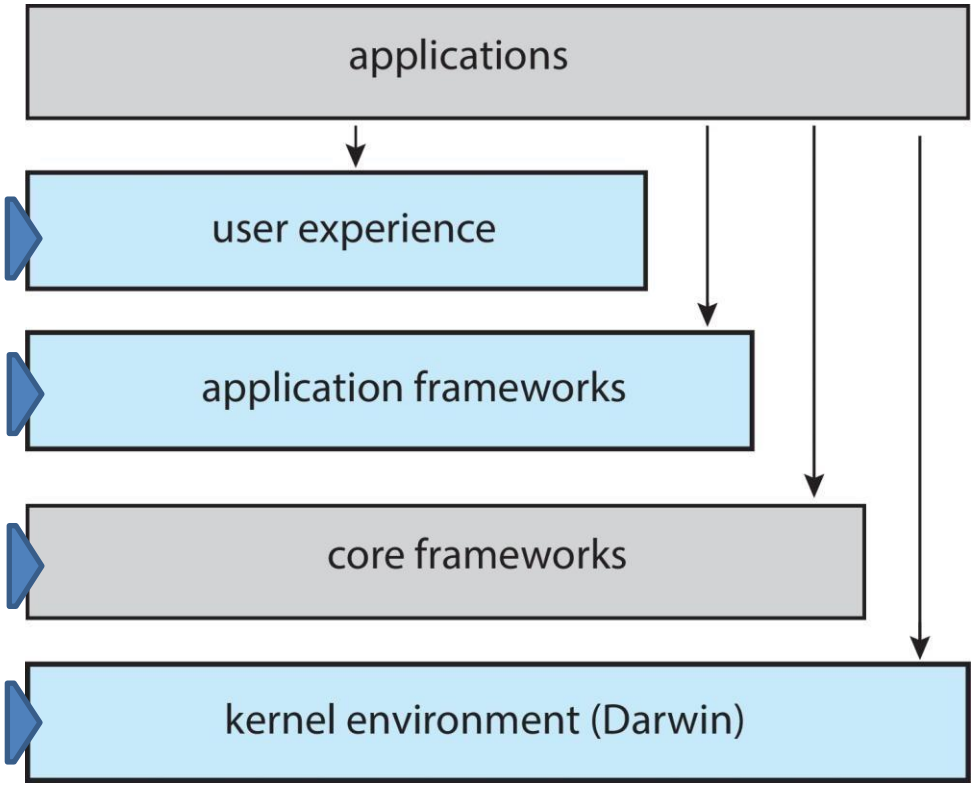
Hybrid systems MacOS and iOS

In the user experience layer, MacOS uses **Aqua** which is designed for mouse and trackpad. iOS uses **Springboard** which is designed for touch devices.

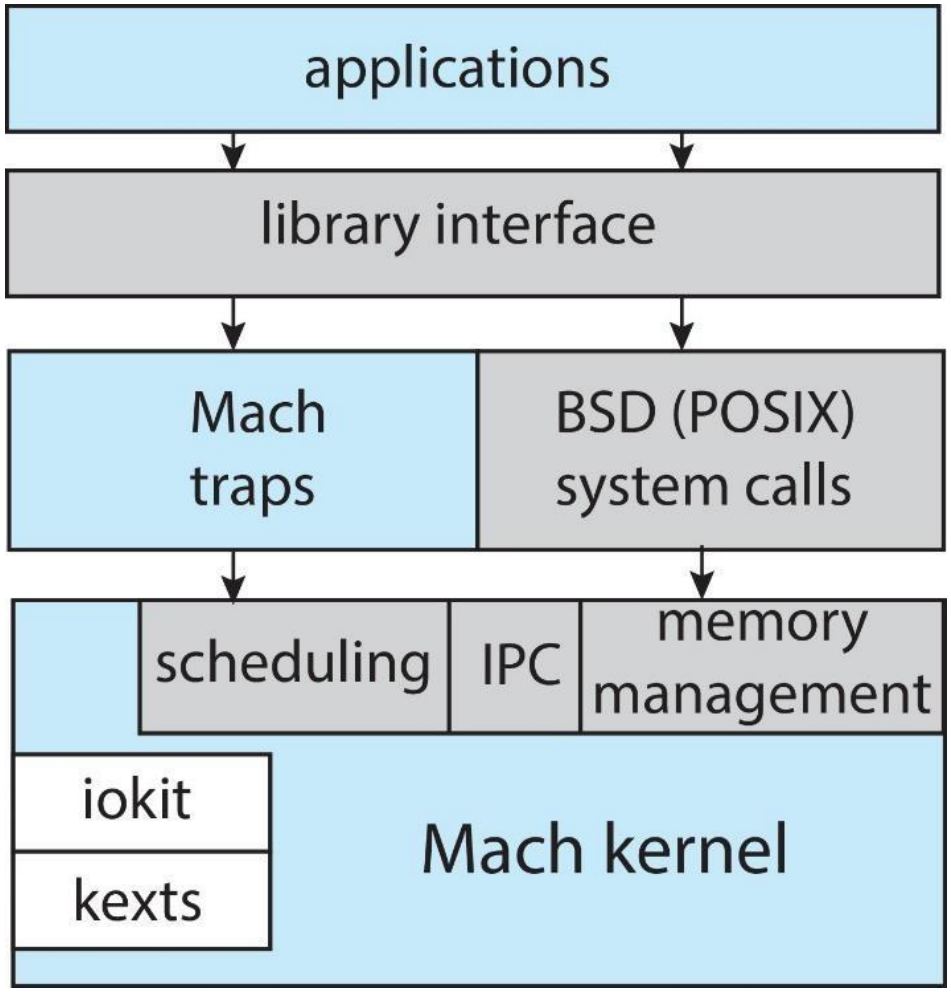
the API interface in the application framework layer for different programming languages: MacOS uses **Cocoa** and iOS uses **Cocoa Touch**

In the core framework layer, you will find support for graphics and media: **Quicktime** and **OpenGL**.

The Darwin (kernel environment) uses **Mach microkernel** and **BSD Unix kernel**.

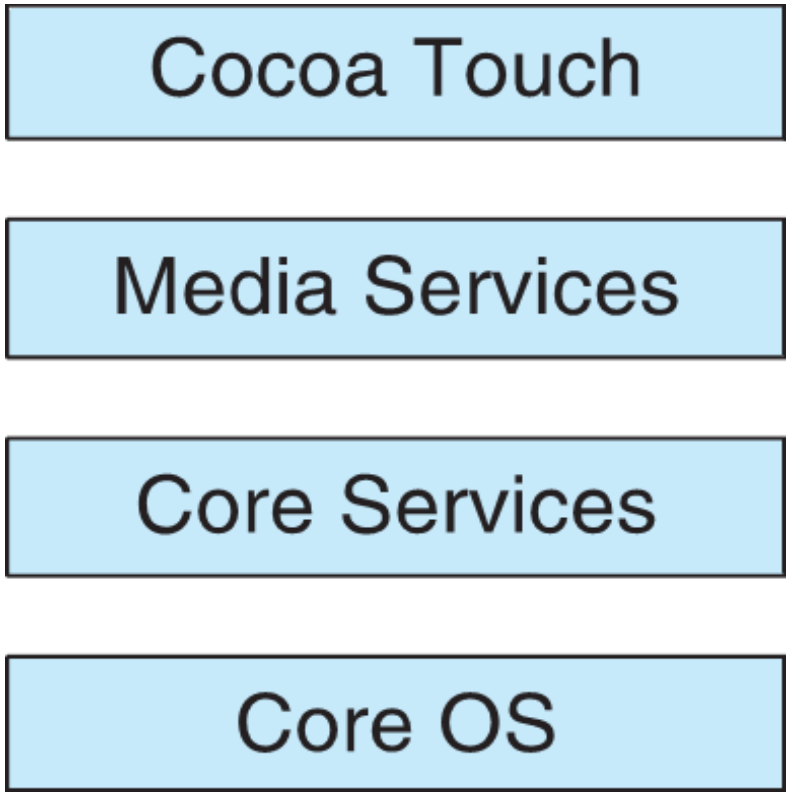


Darwin



- MACH: Microkernel
- BSD Unix: monolithic structure
- Iokit: for developing device drivers
- Kexts (Kernel extension): dynamically loadable modules

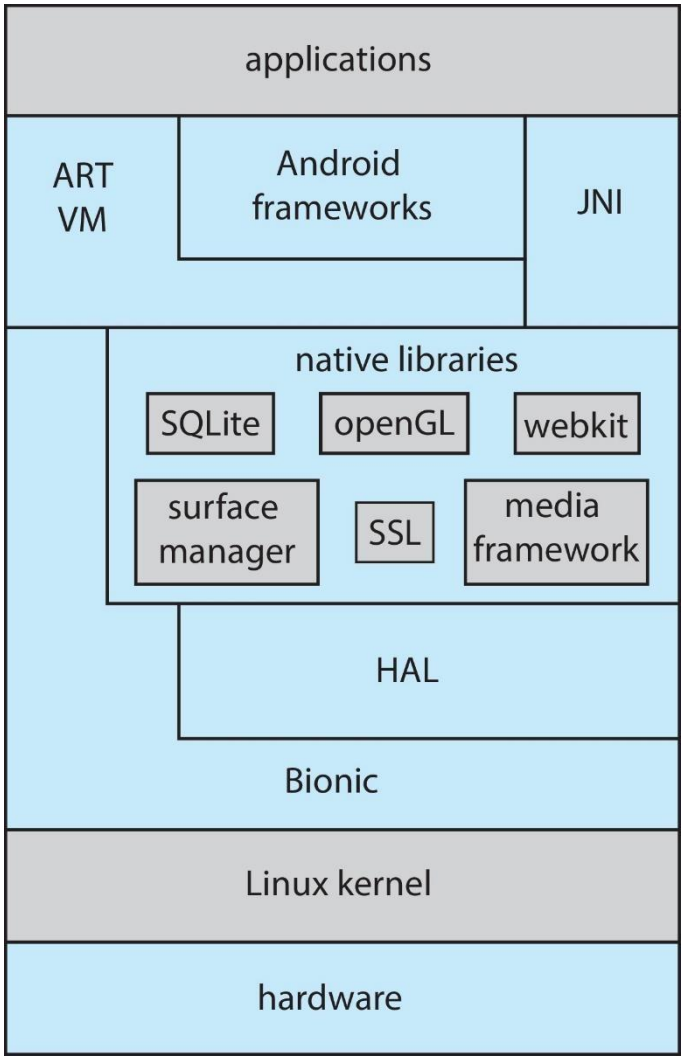
Hybrid systems: iOS



Layered structure

- Cocoa Touch (Springboard): Touch screen interface
- Media Service: Graphics, Audio and Video
- Core Service: supports cloud computing and databases
- Core OS: Based on Mac OS but cannot directly run Mac OS applications

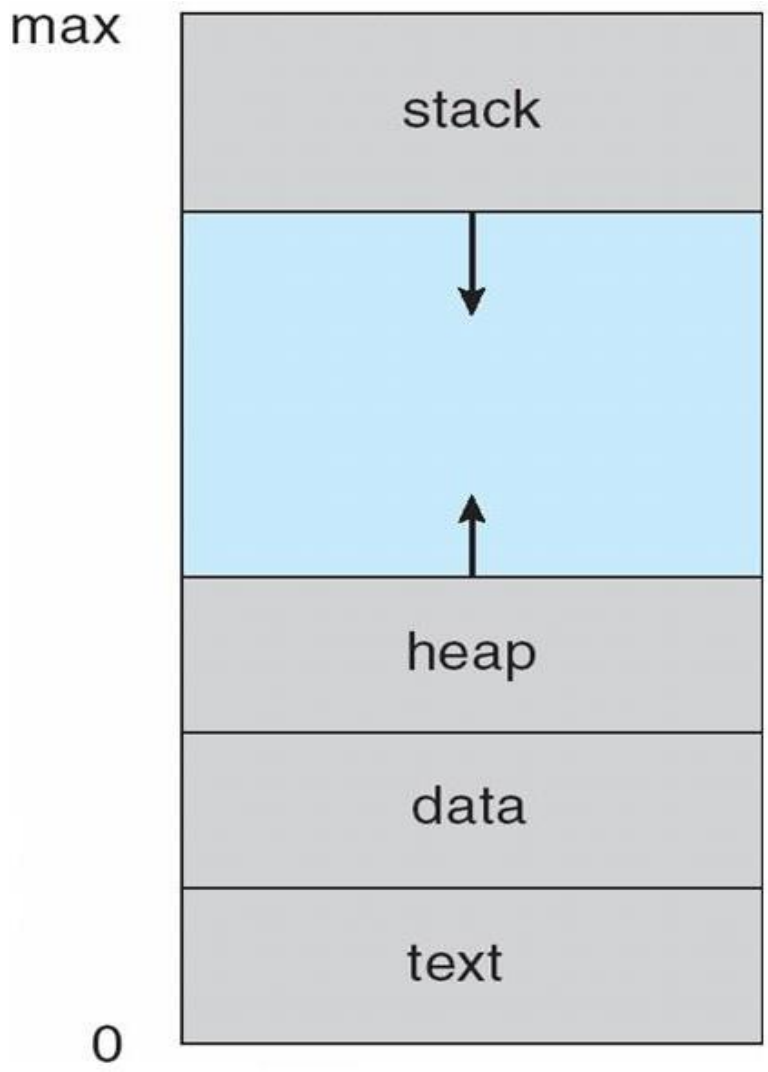
Hybrid systems: Android



Layered structure

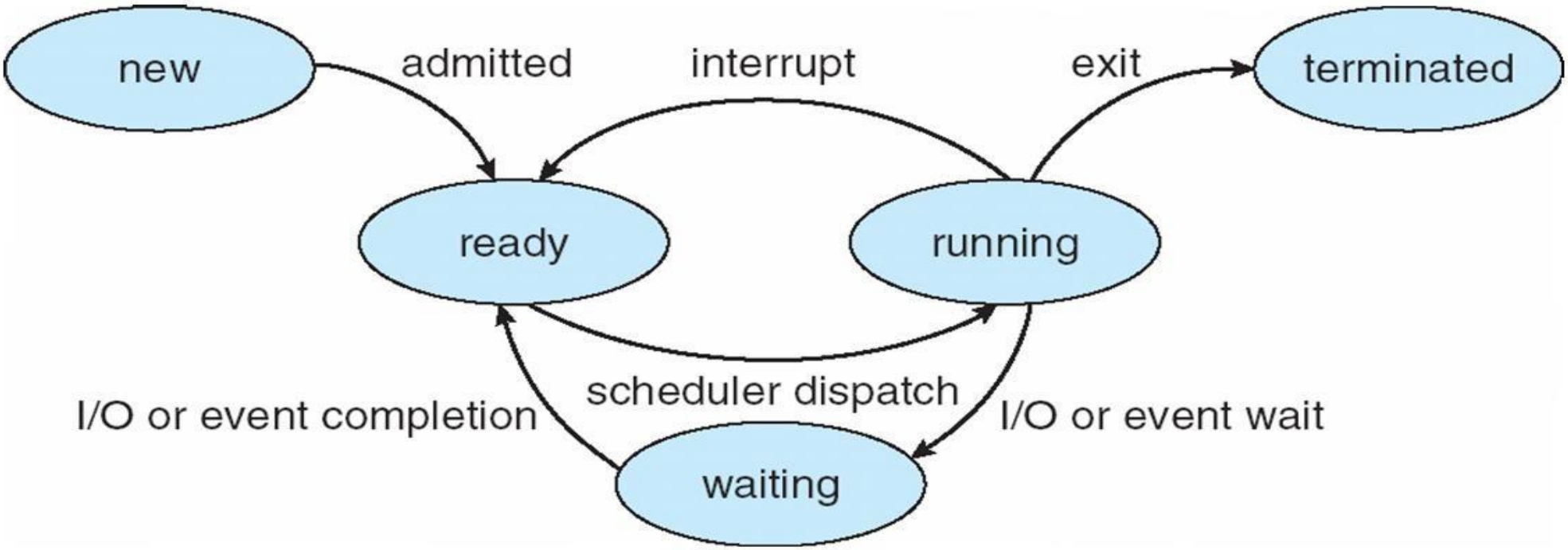
- Modified Linux kernel: Process, Memory and device driver support, as well as power management
- Applications are being developed in Java, but Google has modified this and calls it the Android API
- The Java bytecode is translated into Dalvik bytecode (.dex - Dalvik EXecutable), which runs on Dalvik Virtual machine

A Process

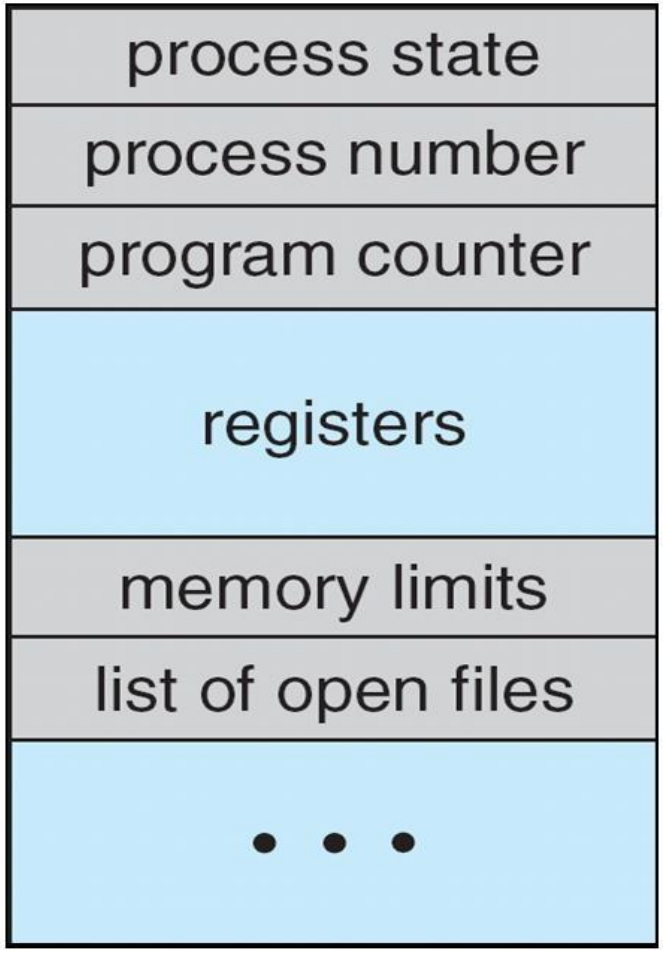


A Process is NOT a program located on the hard disk. Only when it is brought into memory it is a process

Diagram of process states

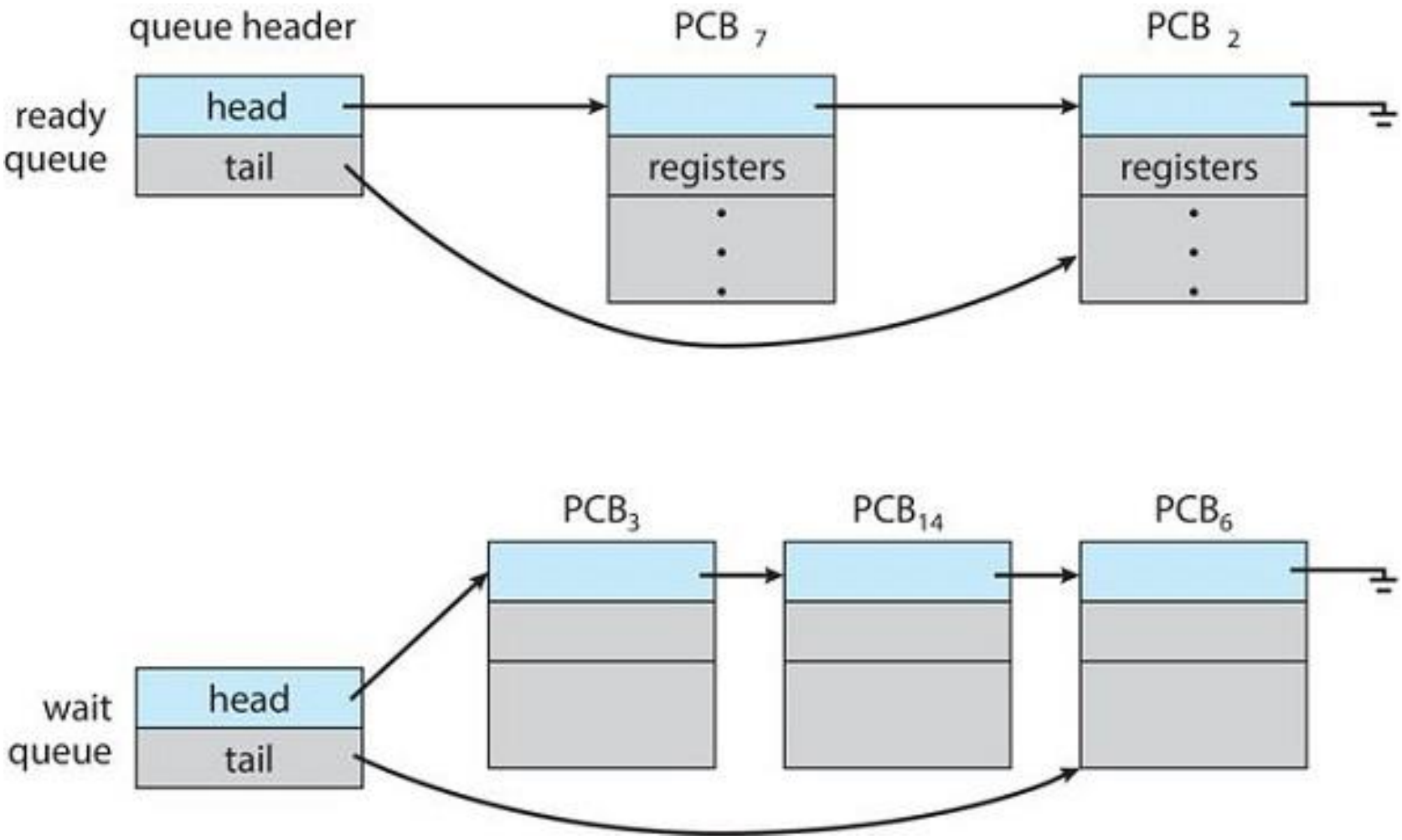


Process **C**ontrol **B**lock (**PCB**)

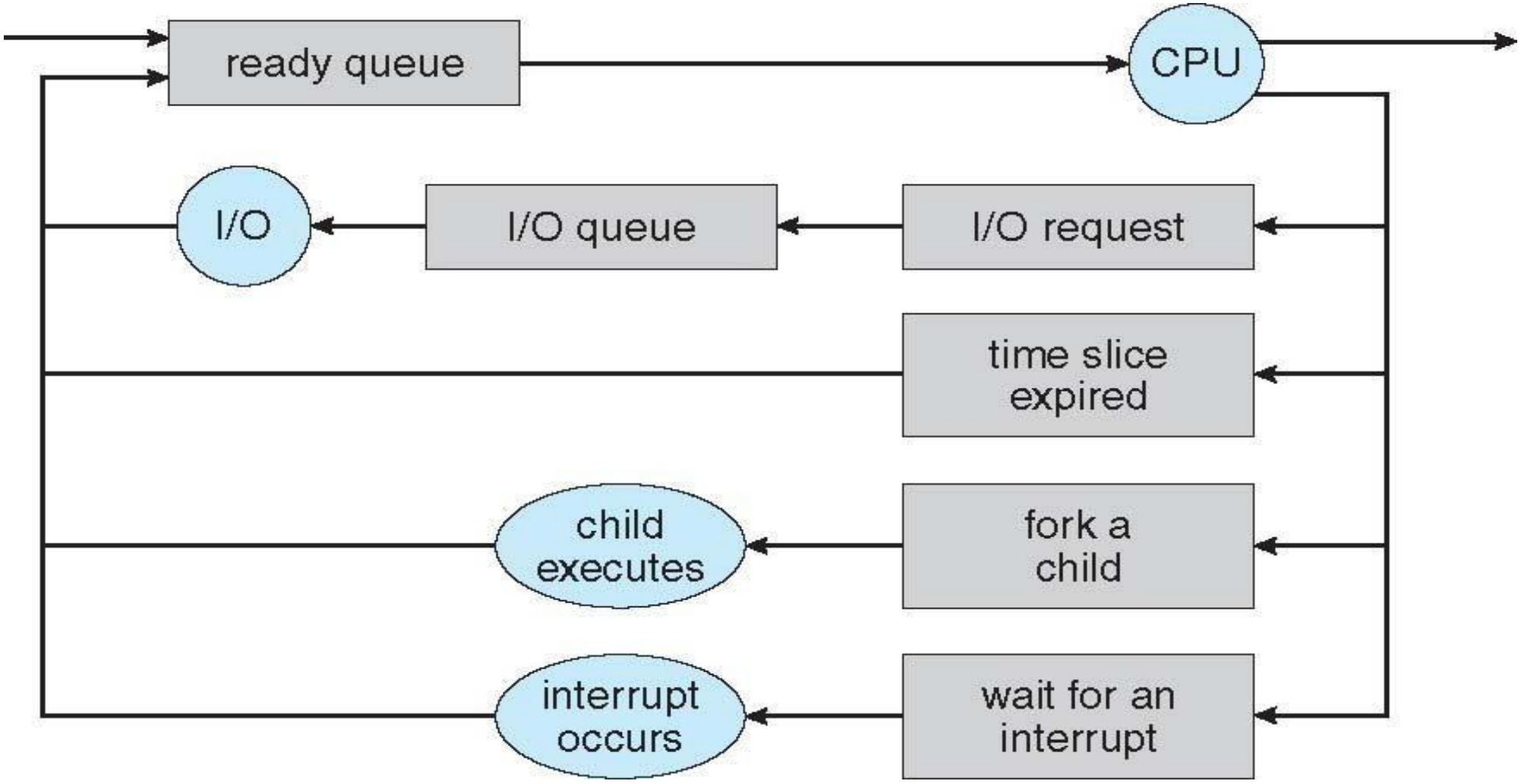


The information that the operating system stores about the running processes in order to make the best possible use of the system's resources

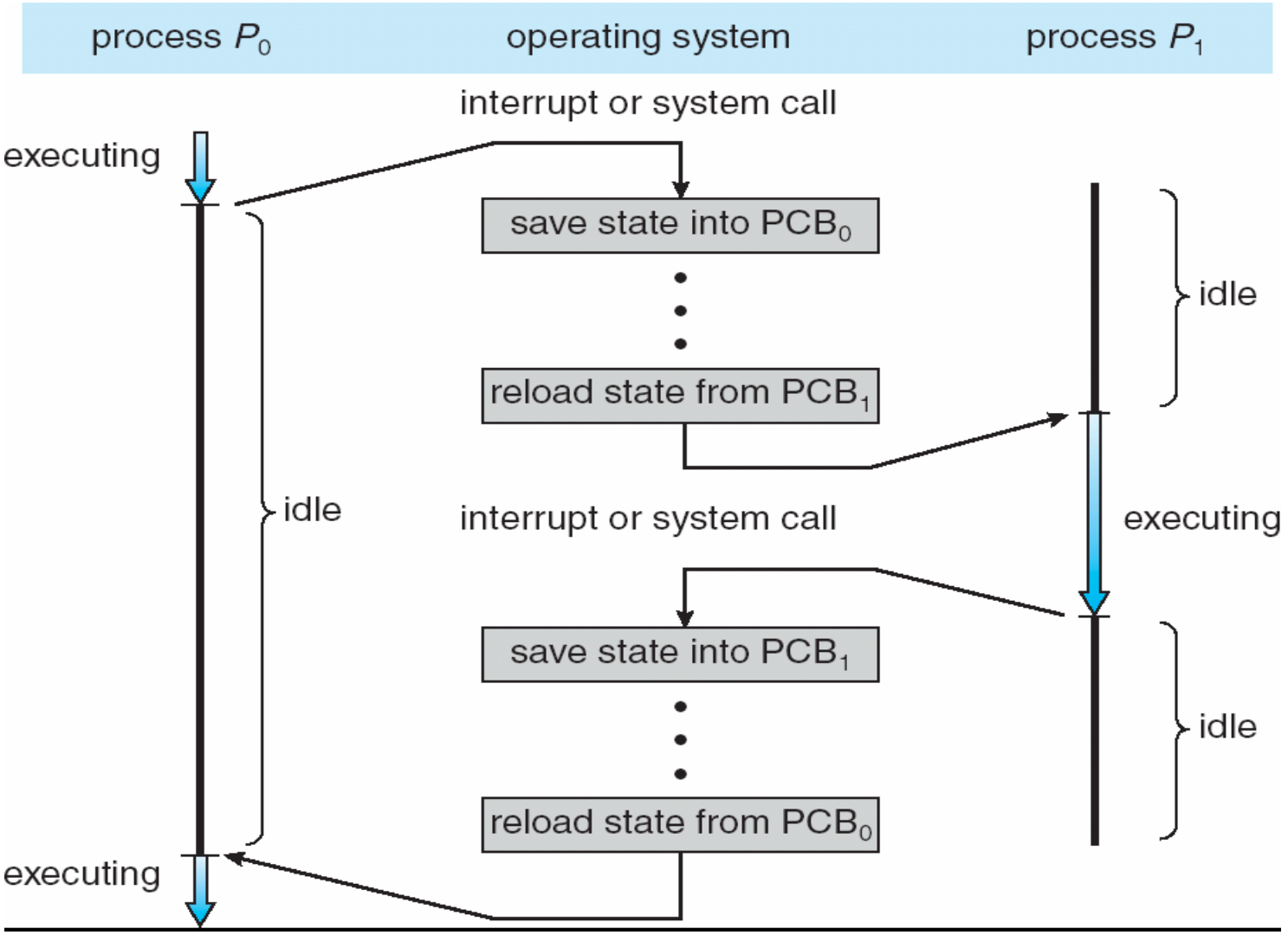
Scheduling queues



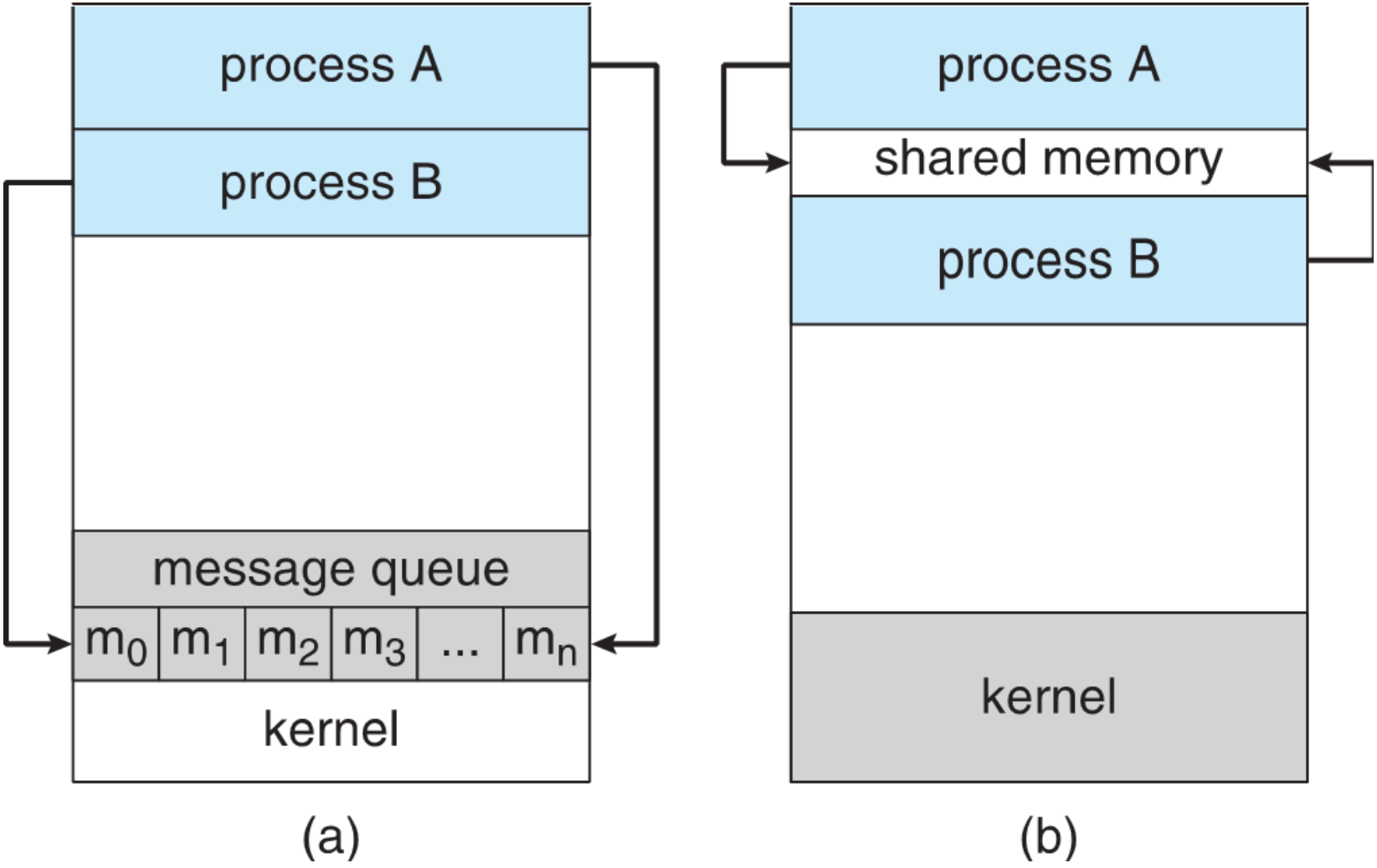
Queueing diagram



Context Switch



Interprocess communication models



Message passing

Shared memory