OPERATING SYSTEM STRUCTURE (2B)

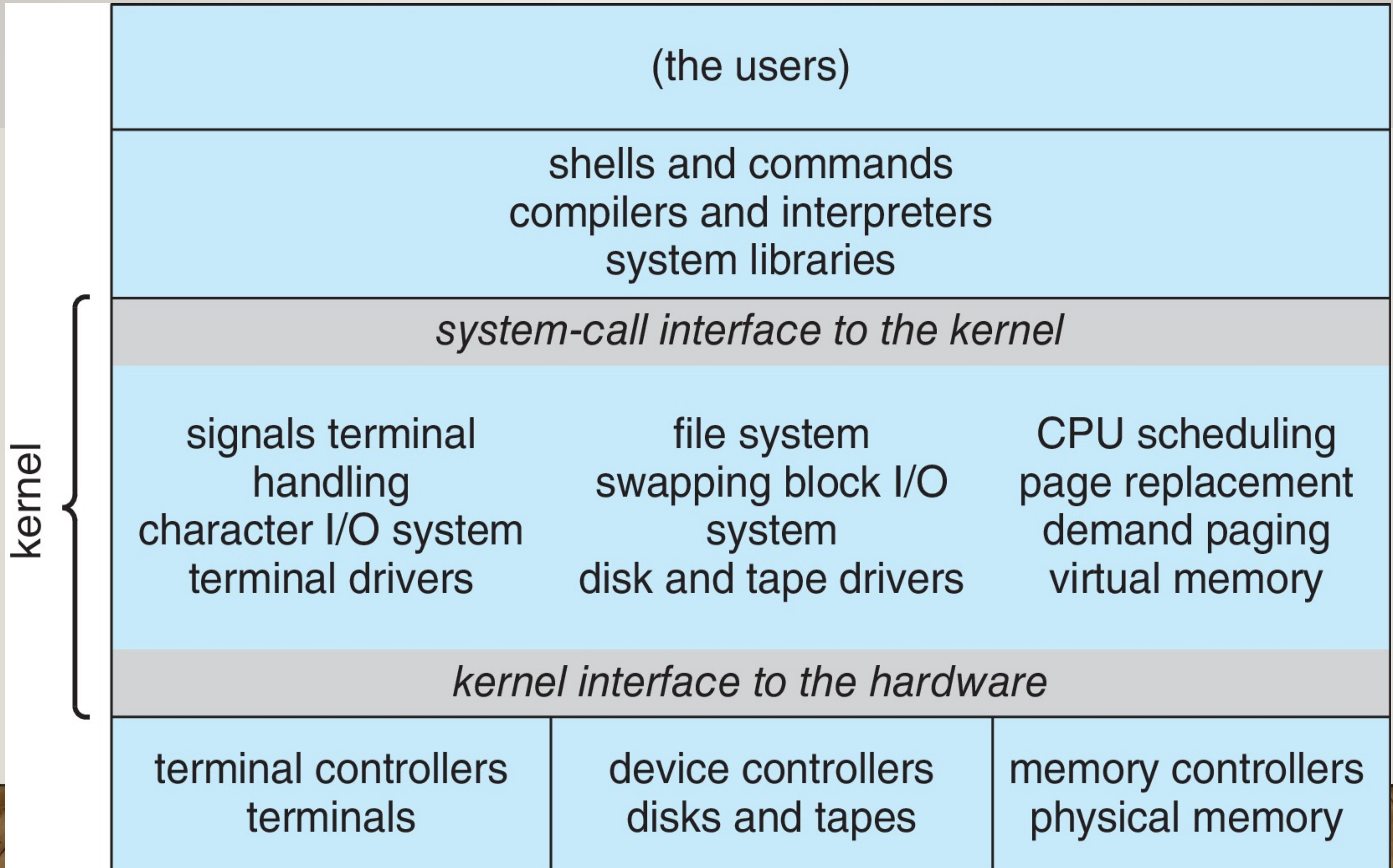
- General-purpose OS is very large program
- OS should be easily modifiable.
- A common approach is to partition the task into small components, or modules, rather than have one single system.
- Each of these modules should be a well-defined portion of the system, with carefully defined interfaces and functions
- Various ways to structure:
  - Simple structure – MS-DOS
  - More complex – UNIX
  - Layered – an abstraction
  - Microkernel – Mach

MONOLITHIC STRUCTURE – ORIGINAL UNIX

- Simplest structure: no structure at all. Place all of the functionality of the kernel into a single, static binary file that runs in a single address space. This is known as Monolithic Structure.
- Eg: 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

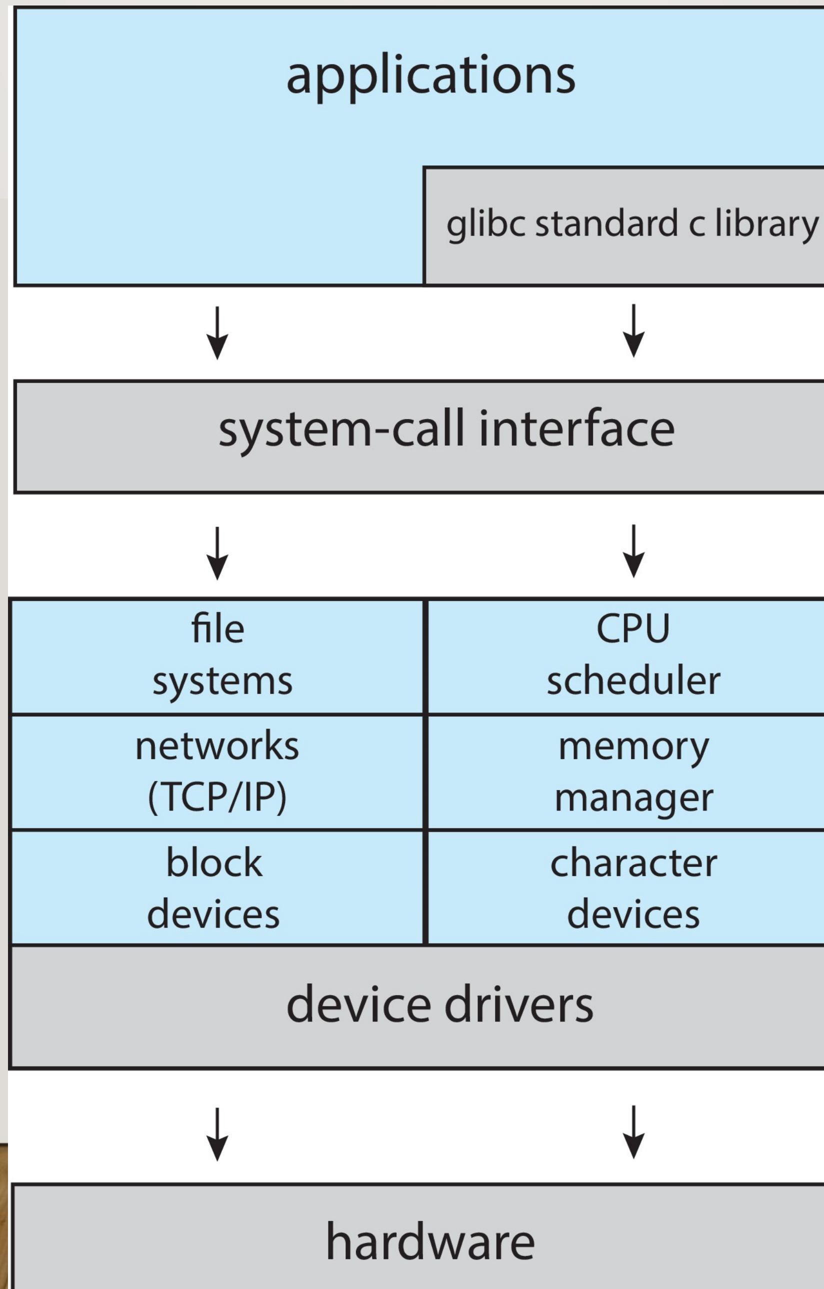
TRADITIONAL UNIX SYSTEM STRUCTURE

Beyond simple but not fully layered



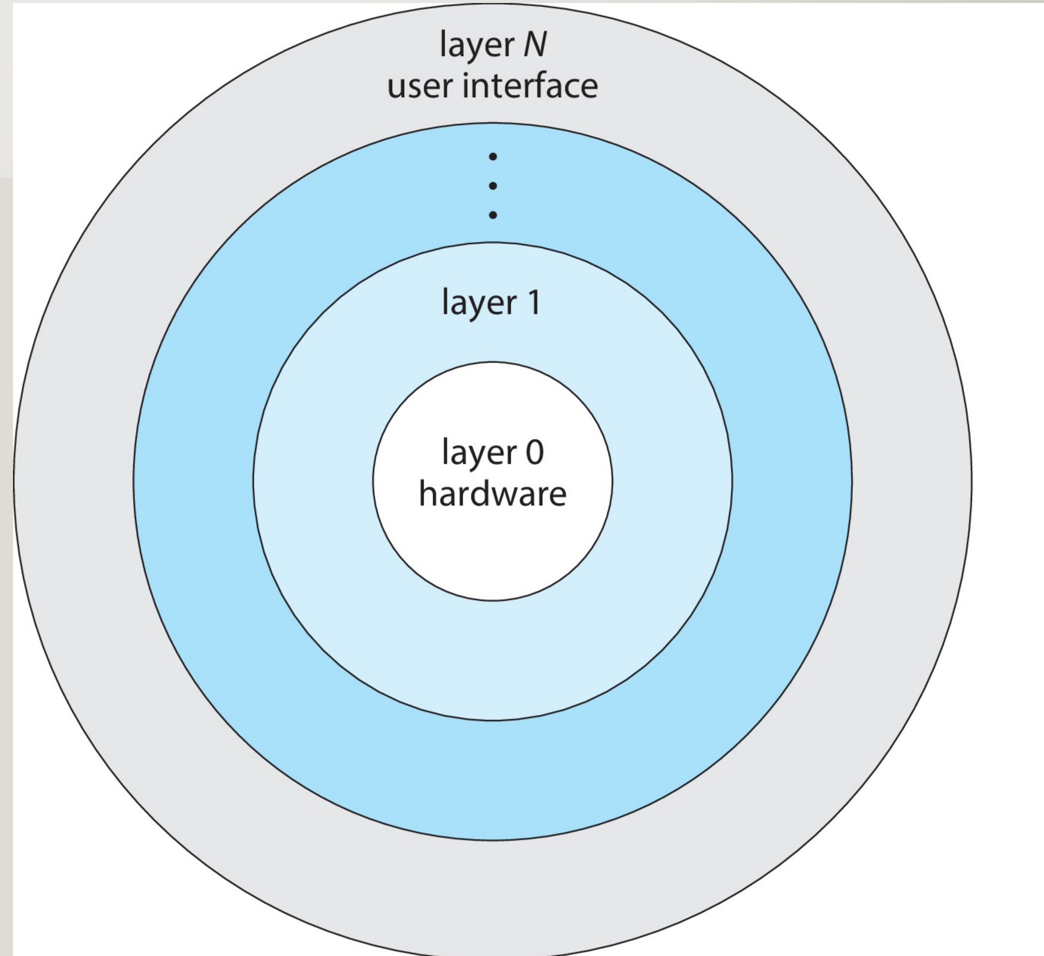
LINUX SYSTEM STRUCTURE

Monolithic plus modular design



LAYERED APPROACH

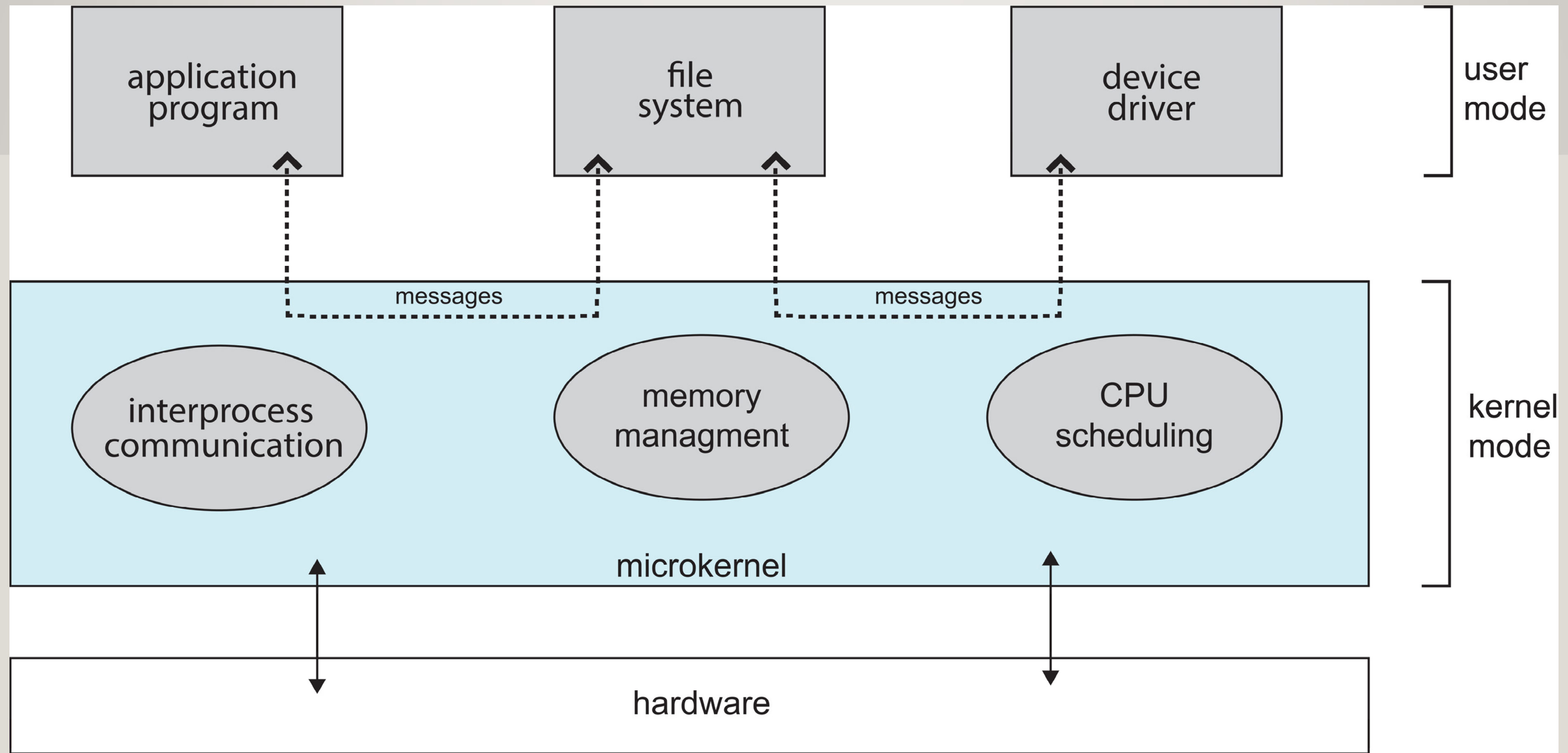
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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



MICROKERNELS

- Moves as much from the kernel into user space
- **Mach** is an 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) and secure
- Detriments:
  - Performance overhead of user space to kernel space communication

MICROKERNEL SYSTEM STRUCTURE



MODULES

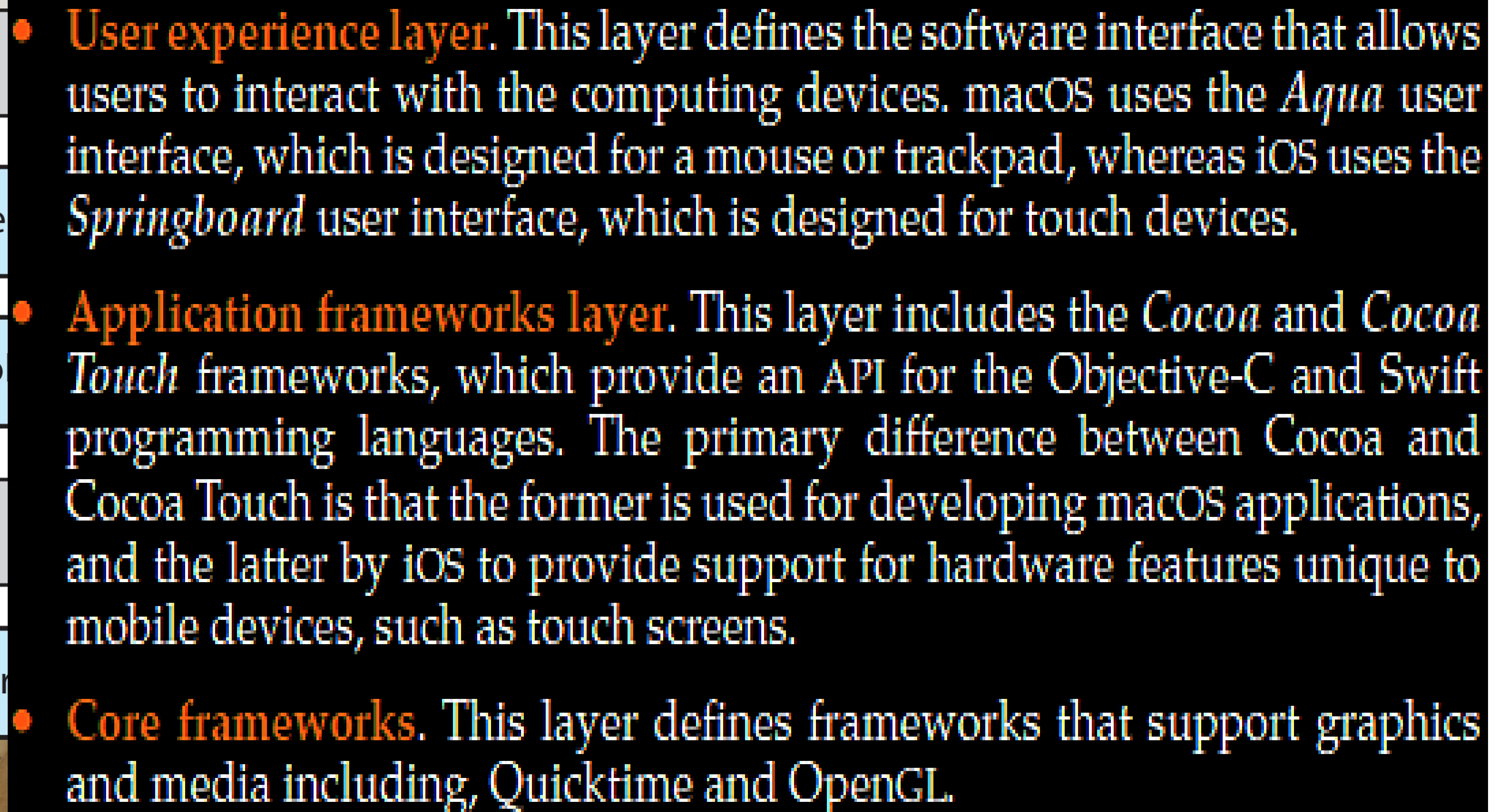
- Many modern operating systems implement **loadable kernel modules** (**LKMs**)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible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

HYBRID SYSTEMS

- Most modern operating systems are 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 support of 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**)

MACOS AND IOS STRUCTURE

- Apple's macOS is designed to run primarily on desktop and laptop computer systems.
- IOS is a mobile operating system designed for the iPhone smartphone and iPad tablet computer



Significant distinctions between macOS and iOS

- macOS is compiled to run on Intel architectures. iOS is compiled for ARM-based architectures.
- Similarly, the iOS kernel has been modified somewhat to address specific features and needs of mobile systems, such as power management and aggressive memory management.
- iOS has more stringent security settings than macOS.
- The iOS is generally much more restricted to developers than macOS and may even be closed to developers.
- For example, iOS restricts access to POSIX and BSD APIs on iOS, whereas they are openly available to developers on macOS

Best known illustration of a microkernel operating system is ***Darwin***, the kernel component of the macOS and iOS operating systems. *Kernel environment: This environment, also known as Darwin, includes the Mach microkernel and the BSD UNIX kernel.*

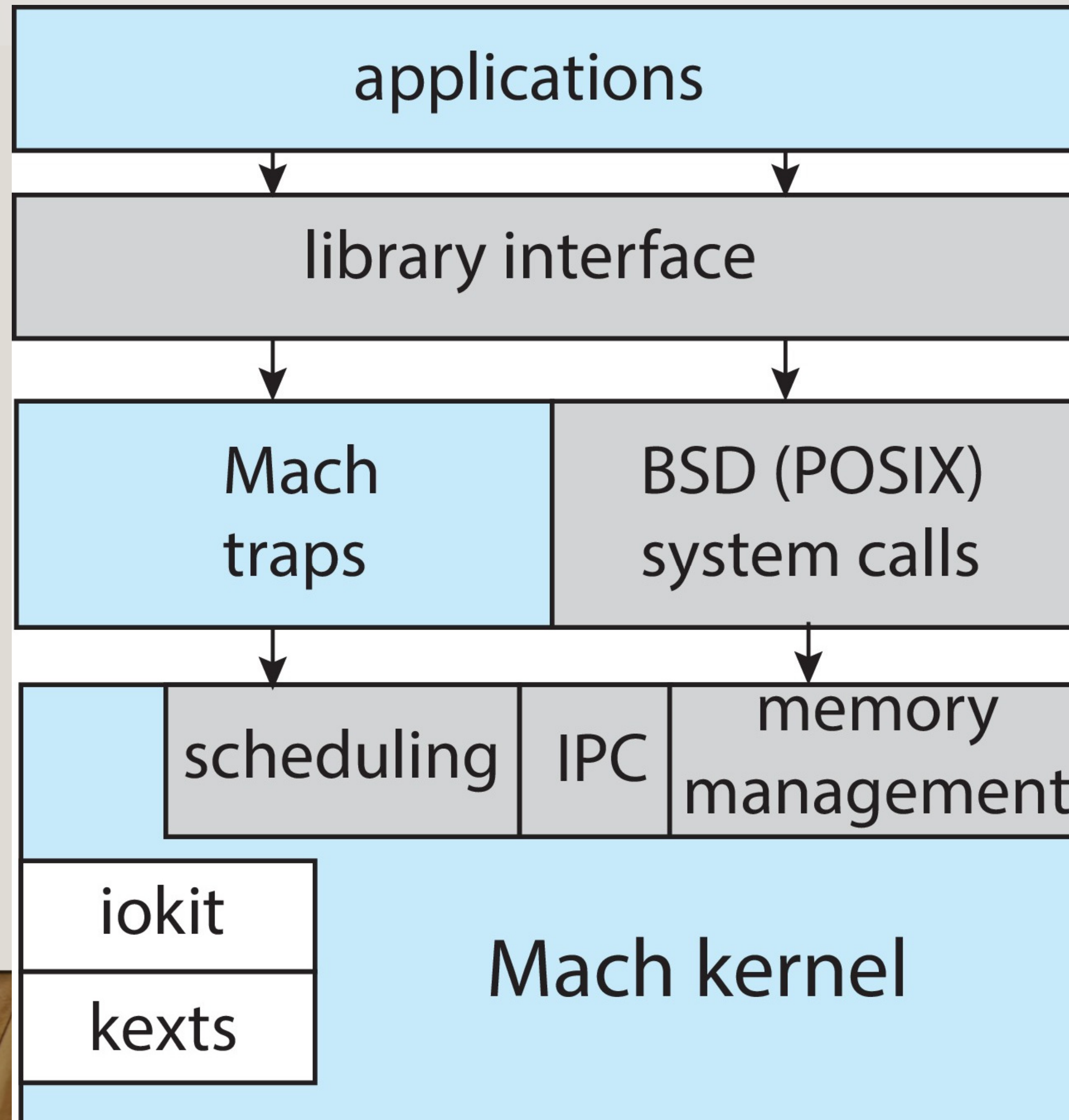
Darwin, in fact, consists of two kernels, one of which is the Mach microkernel. Darwin, which uses a hybrid structure, is also a layered system that consists primarily of the Mach microkernel and the BSD UNIX kernel.

Darwin, the core kernel component of macOS, is based on BSD UNIX and is open-sourced as well - http://www.opensource.apple.com/

Most operating systems provide a single system-call interface to the kernel—such as through the standard C library on UNIX and Linux systems — Darwin provides two system-call interfaces: Mach system calls (known as traps) and BSD system calls (which provide POSIX functionality). The interface to these system calls is a rich set of libraries that includes not only the standard C library but also libraries that provide networking, security, and programming language support. Beneath the system-call interface, Mach provides fundamental operating system services, including memory management, CPU scheduling, and interprocess communication (IPC) facilities such as message passing and remote procedure calls (RPCs). Much of the functionality provided by Mach is available through kernel abstractions, which include tasks (a Mach process), threads, memory objects, and ports (used for IPC).

DARWIN

- Layered system that consists primarily of the Mach microkernel and the BSD UNIX kernel.
- Provides two system-call interfaces

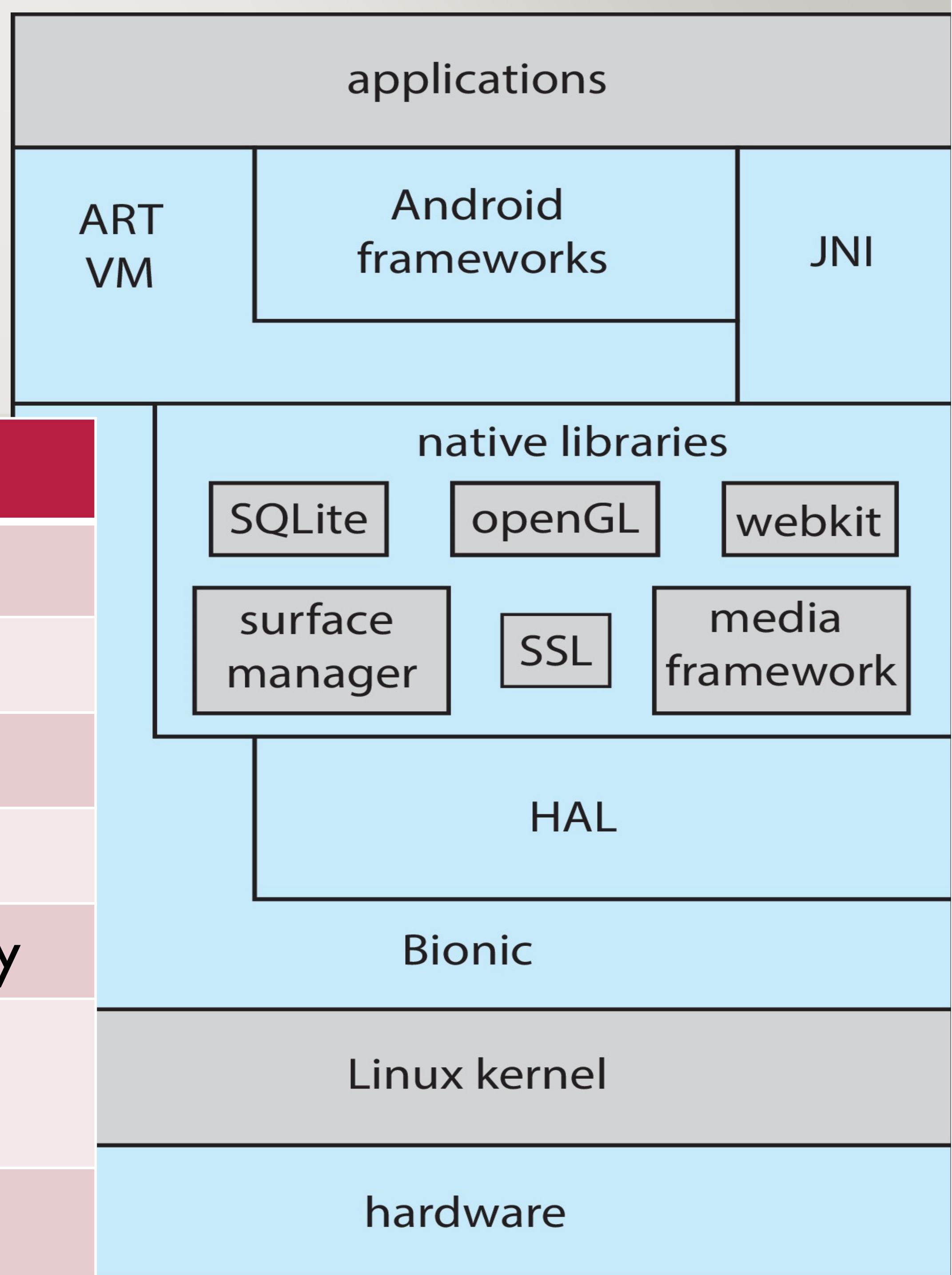


ANDROID

- Open Source: Developed by Open Handset Alliance (mostly Google)
- Similar stack to iOS
- Based on Linux kernel but modified
    - Provides process, memory, device-driver and 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; then runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

ANDROID ARCHITECTURE

Sl. No.	Acronym	Full Form
1	ART	Android Runtime
2	VM	Virtual Machine
3	JNI	Java Native Interface
4	SSL	Secure Socket Layer
5	openGL	Open Graphics Library
6	HAL	Hardware Abstraction Layer
7	SQLite	Opensource SQL relational database – Text mode



BUILDING AND BOOTING AN OPERATING SYSTEM

- Operating systems generally designed to run on a class of systems with variety of peripherals
- Commonly, operating system already installed on purchased computer
  - But can build and install some other operating systems
  - If generating an operating system from scratch
    - Write the operating system source code
    - Configure the operating system for the system on which it will run
    - Compile the operating system
    - Install the operating system
    - Boot the computer and its new operating system

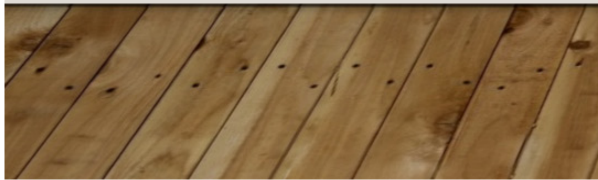
BUILDING AND BOOTING LINUX

- Download Linux source code (http://www.kernel.org)
- Configure kernel via "`make menuconfig`"
- Compile the kernel using "`make`"
  - Produces `vmlinuz`, the kernel image
  - Compile kernel modules via "`make modules`"
  - Install kernel modules into `vmlinuz` via "`make modules_install`"
  - Install new kernel on the system via "`make install`"

SYSTEM BOOT

- When power initialized on system, execution starts at a fixed memory location
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, **BIOS**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
  - Modern systems replace BIOS with **Unified Extensible Firmware Interface** (**UEFI**)
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and sys
- Boot loaders freque

(Boot parameters for the system are set in a GRUB configuration file, which is loaded at startup. GRUB is flexible and allows changes to be made at boot time, including modifying kernel parameters and even selecting among different kernels that can be booted).



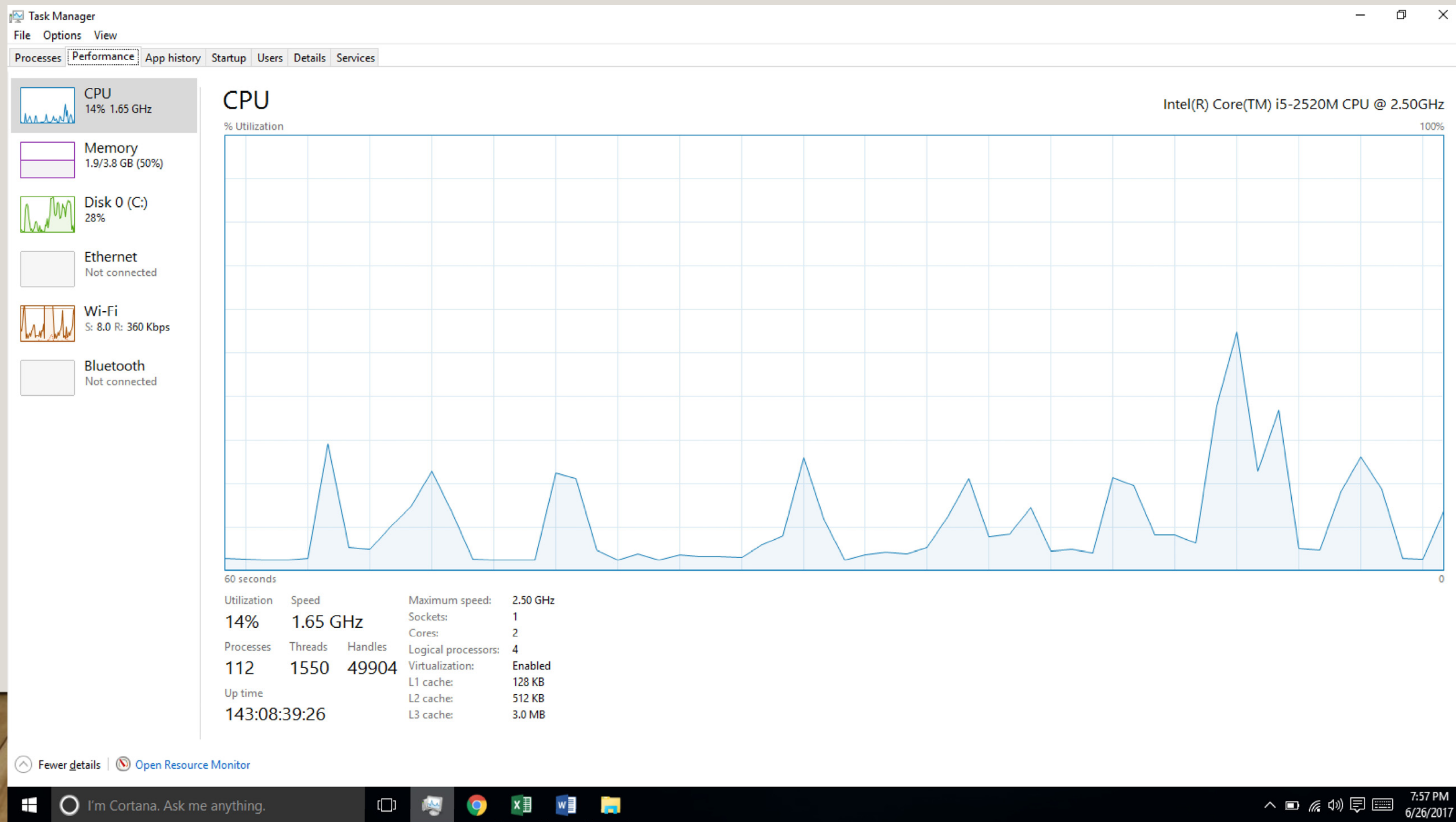
OPERATING-SYSTEM DEBUGGING

- **Debugging** is finding and fixing errors, or **bugs**
- Also **performance tuning**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

PERFORMANCE TUNING

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager



TRACING

- Collects data for a specific event, such as steps involved in a system call invocation
- Tools include
  - strace – trace system calls invoked by a process
  - gdb – source-level debugger
  - perf – collection of Linux performance tools
  - tcpdump – collects network packets

BCC

- Debugging interactions between user-level and kernel code nearly impossible without toolset that understands both and an instrument their actions
- BCC (BPF Compiler Collection) is a rich toolkit providing tracing features for Linux
  - See also the original DTrace
- For example, disksnoop.py traces disk I/O activity

TIME(s)	T	BYTES	LAT(ms)
1946.29186700	R	8	0.27
1946.33965000	R	8	0.26
1948.34585000	W	8192	0.96
1950.43251000	R	4096	0.56
1951.74121000	R	4096	0.35

LINUX BCC/BPF TRACING TOOLS

Linux bcc/BPF Tracing Tools

filetop
filelife fileslower
vfscount vfsstat

opensnoop
statsnoop
syncsnoop

c* java* node*
php* python*
ruby*

mysqld_qslower
bashreadline

gethostlatency
memleak
sslsniff

Other:
capable

ucalls uflow
ugc uobjnew
ustat uthreads

syscount
killsnoop

cachestat cachetop
dcstat dcsnoop
mountsnoop

execsnoop
pidpersec

Applications

System Libraries

System Call Interface

VFS | Sockets | Scheduler

File Systems | TCP/UDP

Volume Manager | IP | Virtual Memory

Block Device Interface | Ethernet

trace
argdist
funccount
funcslower
funclatency
stackcount
profile

cpudist
runqlat runqlen
deadlock_detector
cpuunclaimed

offcputime
wakeuptime
offwaketime

softirqs

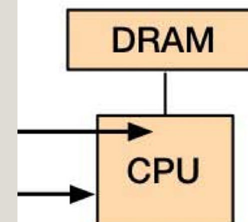
kill memleak
abratetop

mdflush

btrfsdist
btrfsslower
ext4dist ext4slower
xfsdist xfsslower
zfsdist zfsslower

hardirq

biotop biosnoop
biolatency bitesi

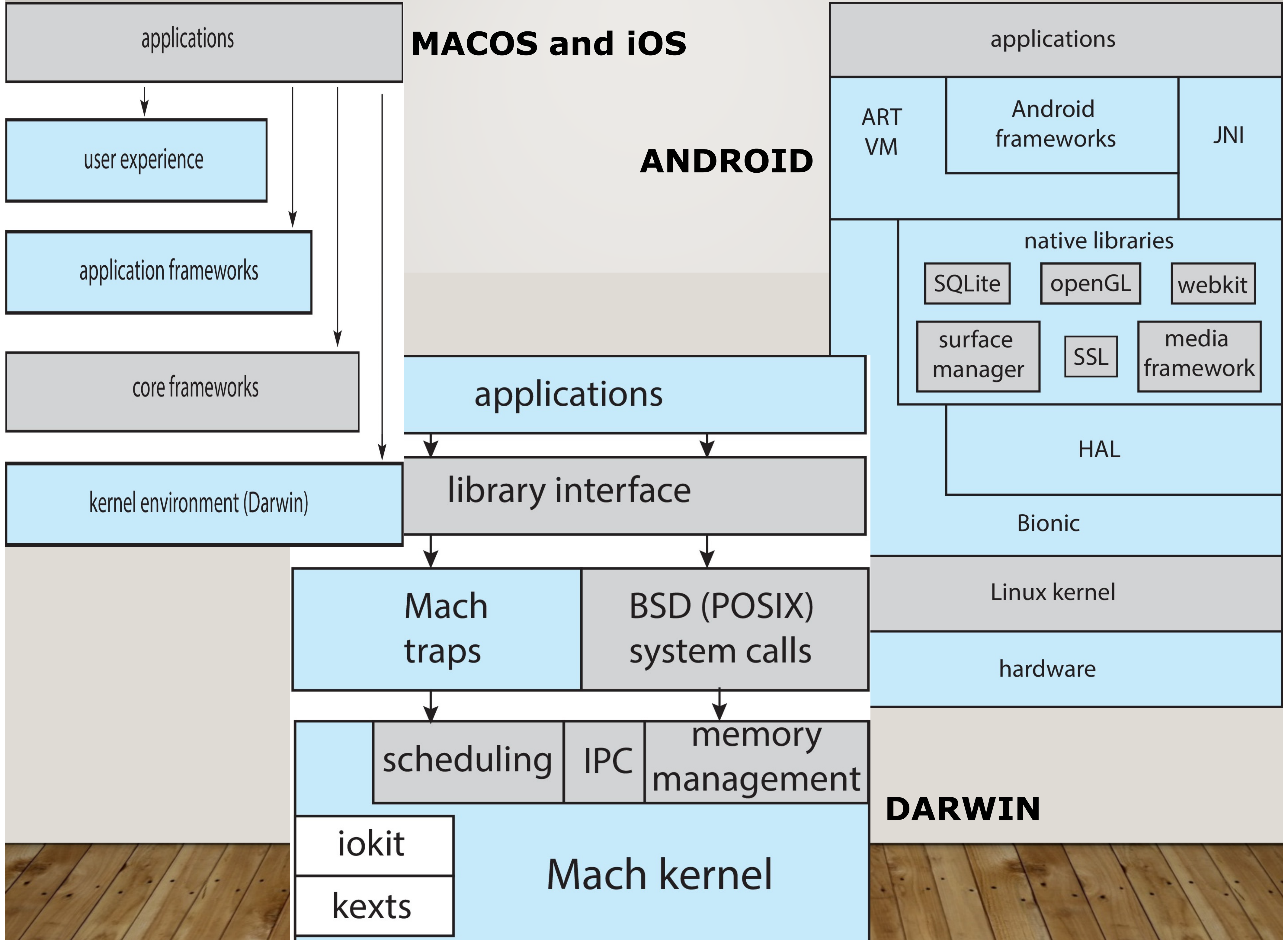


sor/bcc#tools 2017

For example, the command

./opensnoop -p 1225

*will trace **open()** system calls performed only by the process with an identifier of 1225.*


MACOS and iOS
applications
user experience
application frameworks
core frameworks
kernel environment (Darwin)
ANDROID
applications
ART VM
Android frameworks
JNI
native libraries
SQLite
openGL
webkit
surface manager
SSL
media framework
HAL
Bionic
Linux kernel
hardware
applications
library interface
Mach traps
BSD (POSIX) system calls
scheduling
IPC
memory management
iokit
kexts
Mach kernel
DARWIN