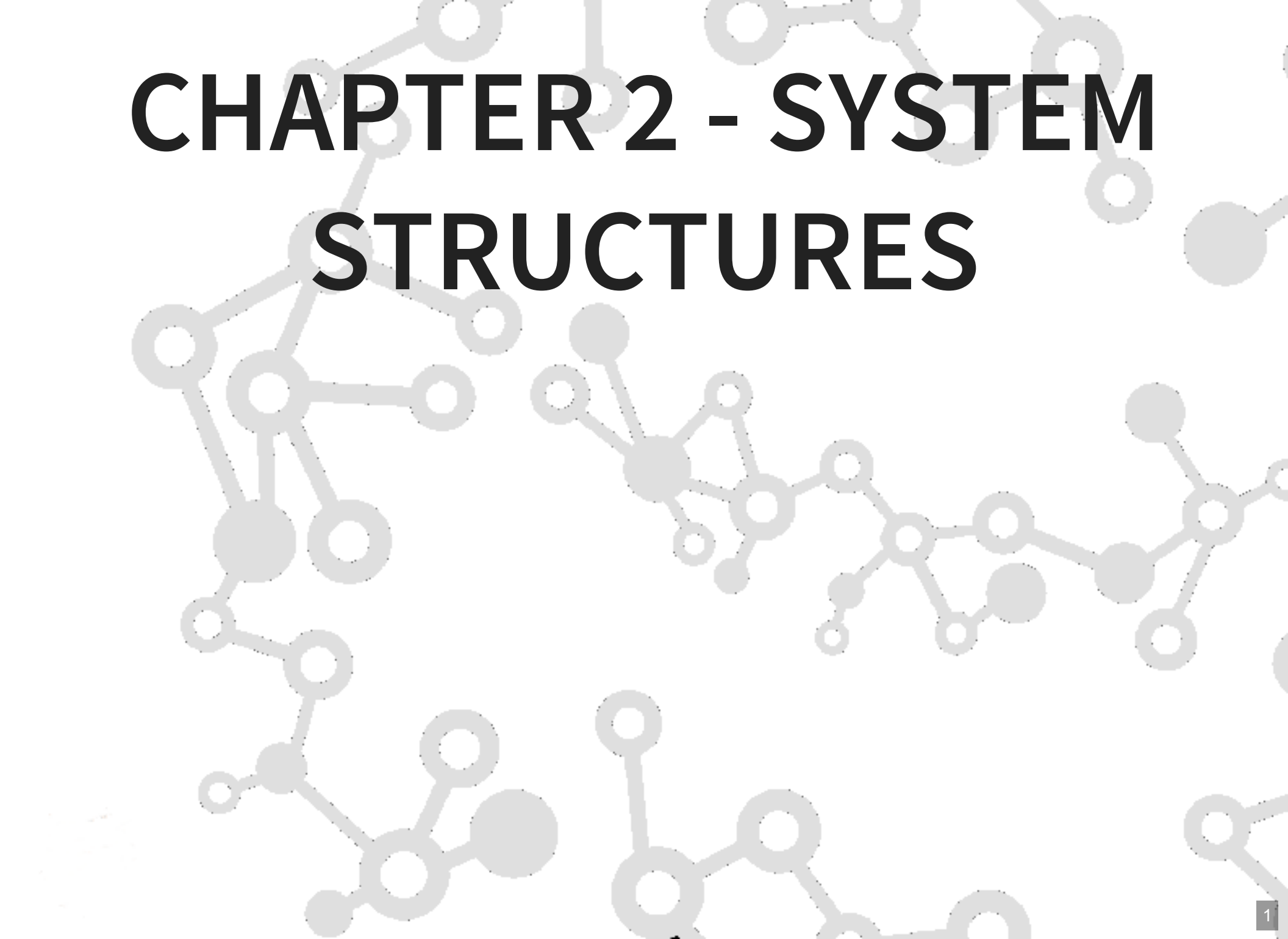
CHAPTER 2 - SYSTEM STRUCTURES

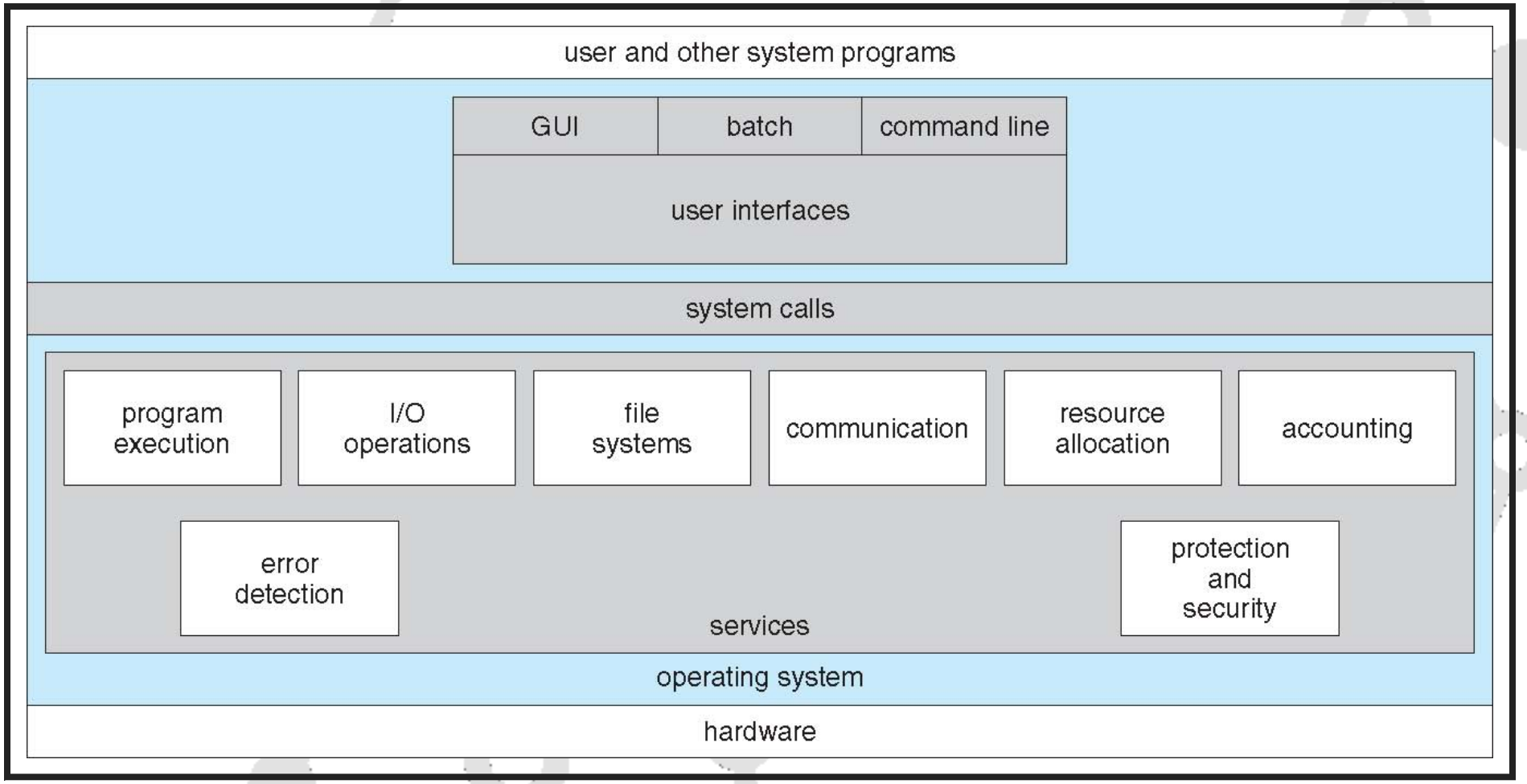


OBJECTIVES

- 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

OPERATING-SYSTEM SERVICES

OPERATING-SYSTEM SERVICES



USER INTERFACE

Almost all operating systems have a user interface (UI).

Varies 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 error)

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.

Programs need to

- read and write files and directories
- create and delete them
- search them
- list file Information
- permission management.

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

RESOURCE ALLOCATION

When multiple users or multiple jobs running concurrently, resources must be allocated to each of them

- Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code

ACCOUNTING

Accounting - To keep track of which users use how much and what kinds of computer resources

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

USER AND OPERATING-SYSTEM INTERFACE

CLI

CLI or command interpreter allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
  - Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn’t require shell modification

GUI

User-friendly desktop metaphor interface

- Usually mouse, keyboard, and monitor
- Icons represent files, programs, actions, etc
- Various mouse buttons over objects in the interface cause various actions
- Invented at Xerox PARC

CLI & GUI

Many systems now include both CLI and GUI interfaces

- Microsoft Windows is GUI with CLI “command” shell
- Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available
- Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

TOUCHSCREEN INTERFACES

Touchscreen devices require new interfaces

- Mouse not possible or not desired
- Actions and selection based on gestures
- Virtual keyboard for text entry

TOUCHSCREEN INTERFACES

CHOICE OF INTERFACE

System administrators who manage computers and **power users** who have deep knowledge of a system frequently use the command-line interface.

On some systems, only a subset of system functions is available via the GUI

Shell scripts are very common on systems that are command-line oriented

SYSTEM CALLS

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 Program Interface (API)** rather than direct system call

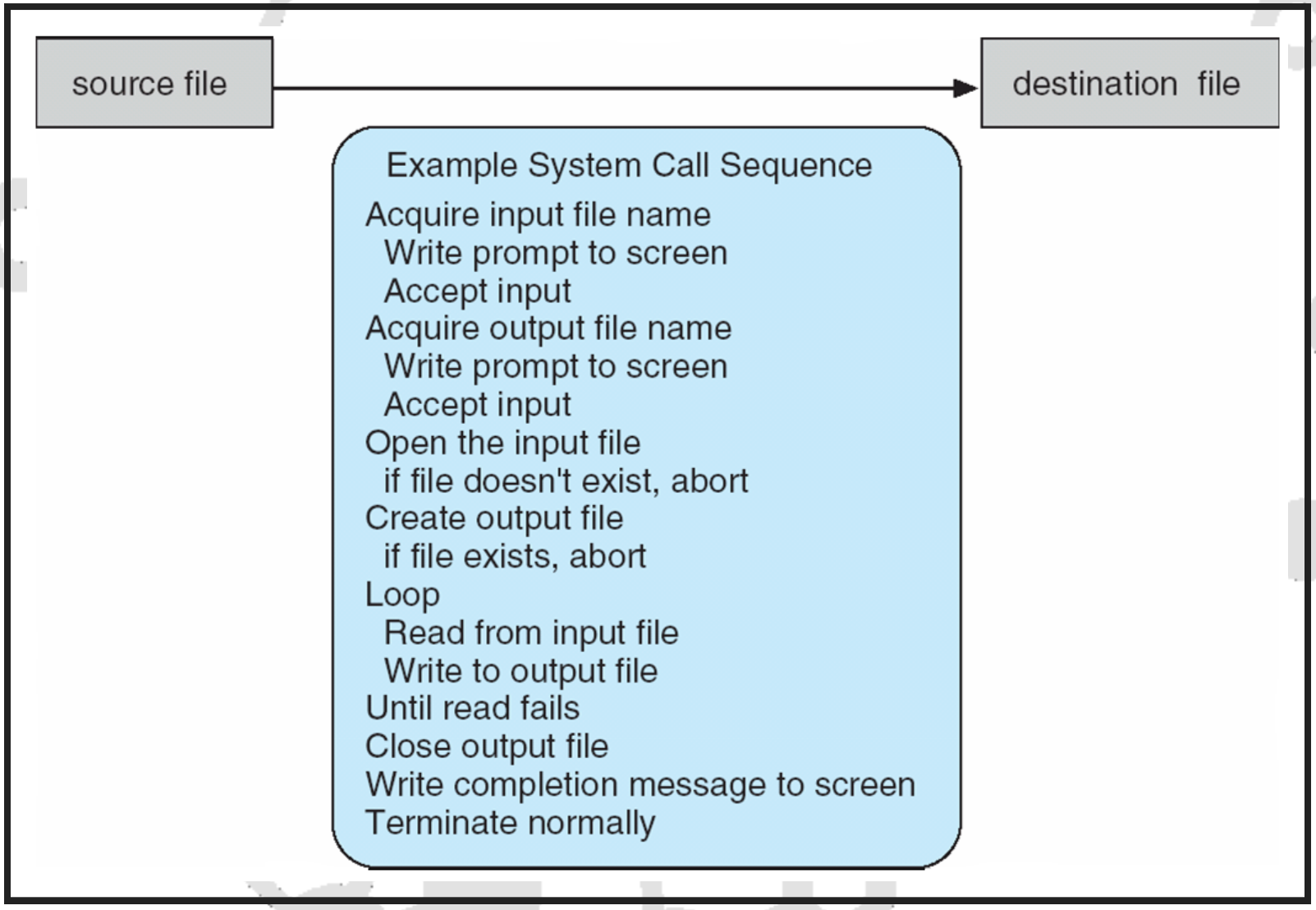
SYSTEM CALLS APIS

Three most common APIs are:

- Win32 API for Windows
- POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
- Java API for the Java virtual machine (JVM)

Why use APIs rather than system calls?

EXAMPLE OF SYSTEM CALLS



EXAMPLE OF STD API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

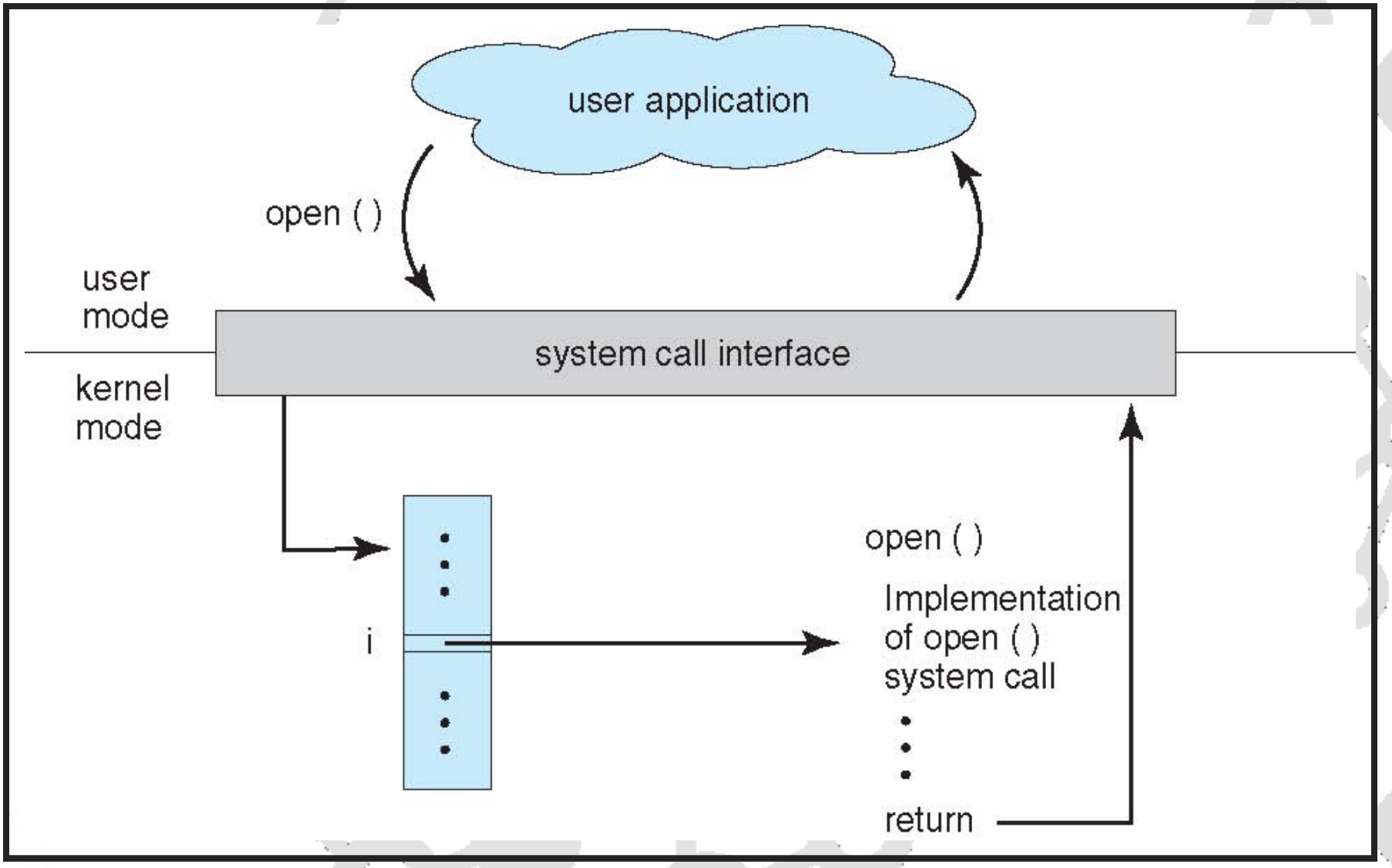
return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

SYSTEM CALLS



SYSTEM CALLS

Most programming languages have run-time support

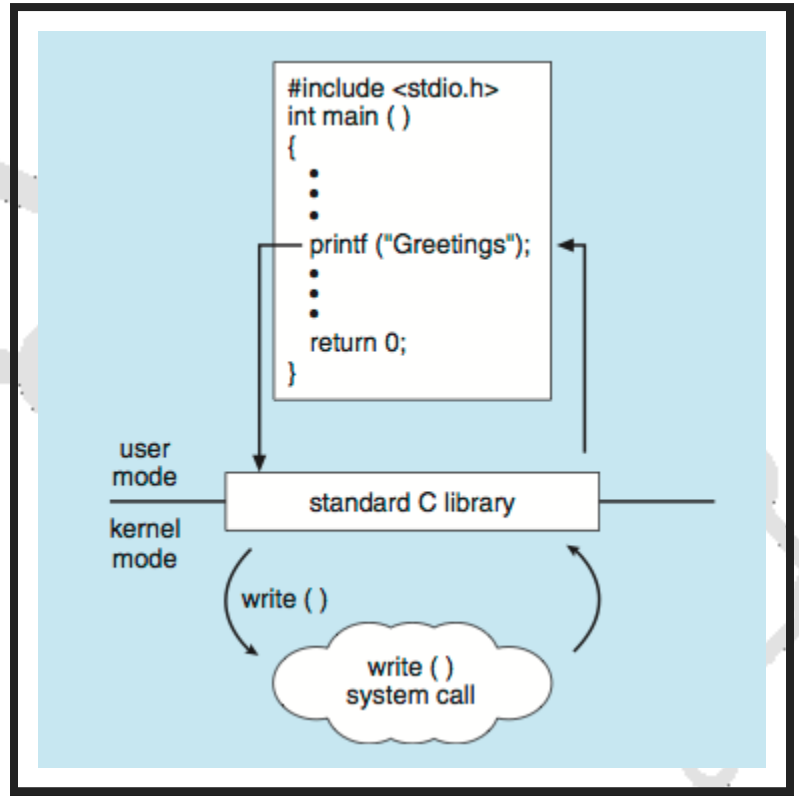
- Set of functions built into libraries included with a compiler
  - provides a **system-call interface** → link to system calls made available by the operating system.
- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers.

SYSTEM CALL APIS

The caller need know nothing about how the system call is implemented or what it does during execution.

Rather, the caller need only obey the API and understand what the operating system will do as a result of the execution of that system call.

STANDARD LIBRARY

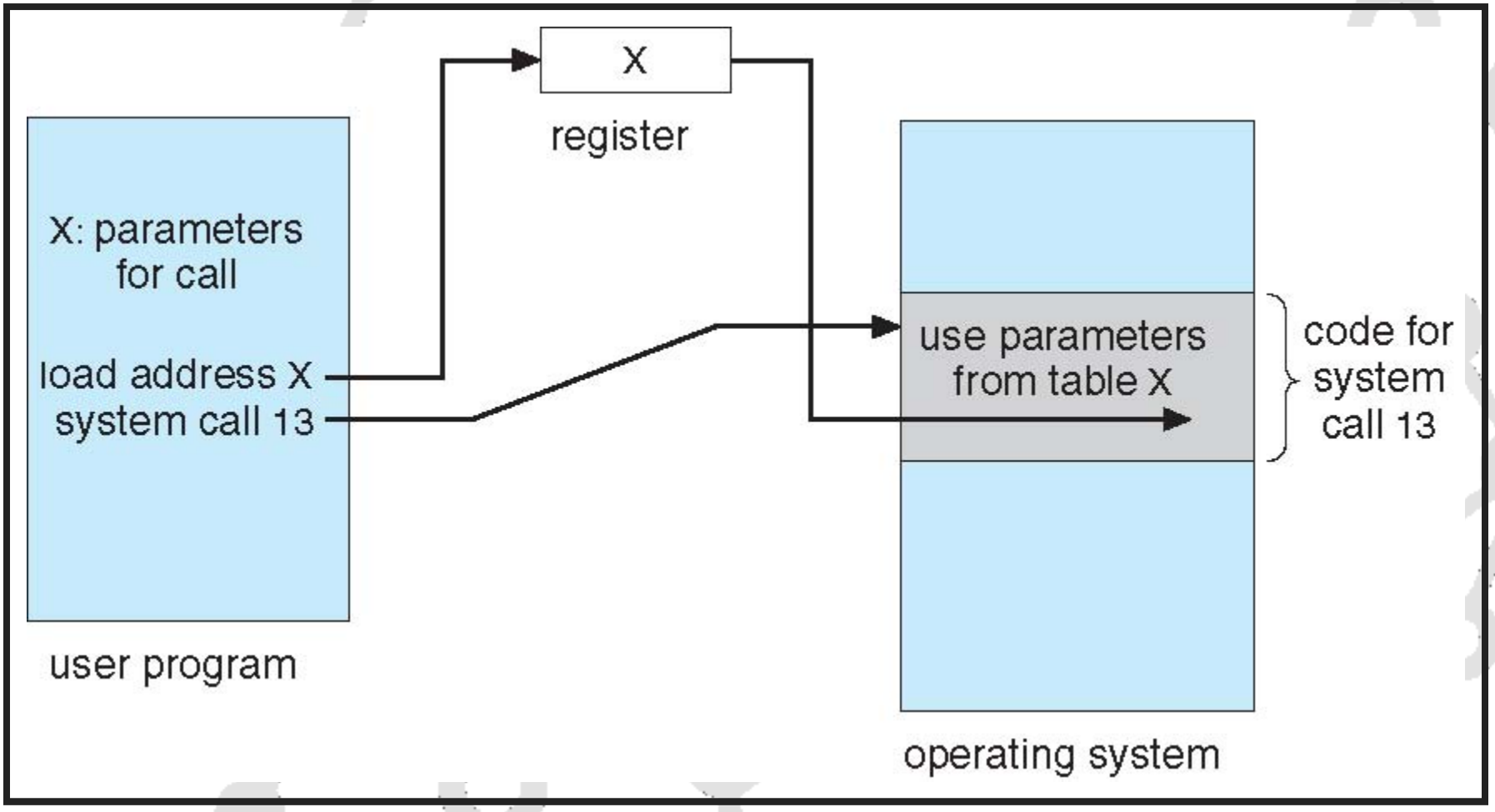


PASSING PARAMETERS

3 general methods to pass parameters to the OS.

- In registers → Simplest
- stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register → Linux & Solaris
- placed, or pushed, onto the stack by the program and popped off the stack by the operating system → do not limit the number or length of parameters

SYSTEM CALLS



TYPES OF SYSTEM CALLS

GROUPS OF SYSTEM CALLS

- Process control
- File manipulation
- Device manipulation
- Information maintenance
- Communications
- Protection.

SYSTEM CALLS EXAMPLES

	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()

PROCESS CONTROL

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

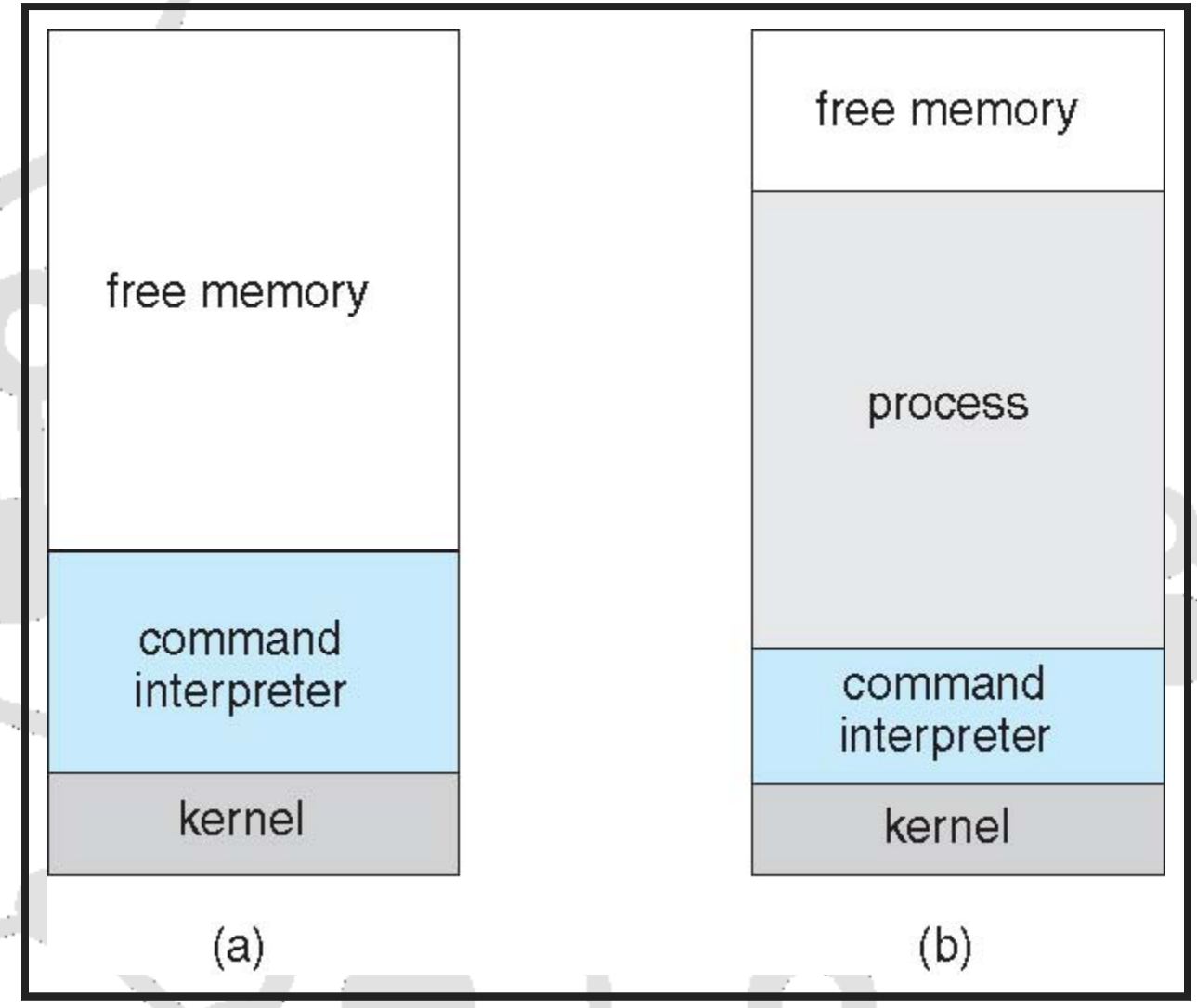
PROCESS CONTROL

- Dump memory if error
- **Debugger** for determining **bugs**, **single step** execution

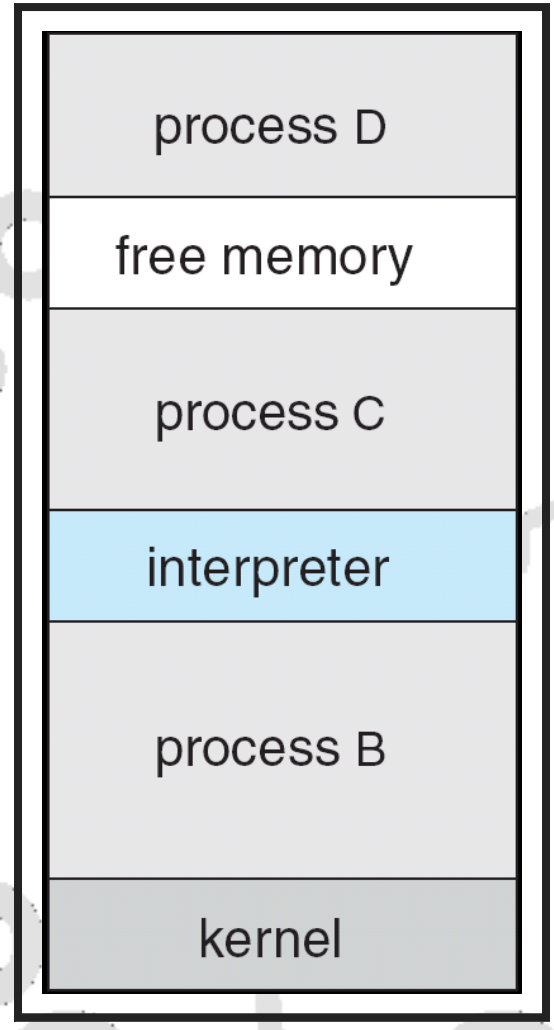
To ensure integrity between shared data

- Locks for managing access

MS-DOS


free memory
command interpreter
kernel
(a)
free memory
process
command interpreter
kernel
(b)

FREEBSD


process D
free memory
process C
interpreter
process B
kernel

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

INFORMATION MAINTENANCE

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes
- single step → trap executed after each command → caught by debugger

COMMUNICATION

- 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

PROTECTION

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

SYSTEM PROGRAMS

SYSTEM PROGRAMS

System programs provide a convenient environment for program development and execution.

Most users' view of the operation system is defined by system programs, not the actual system calls

Provide a convenient environment for program development and execution

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

SYSTEM PROGRAMS

- File management
- Status information
- File modification
- Programming language support
- Program loading and execution
- Communications
- Background services
- Application 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

Typically, these programs format and print the output to the terminal or other output devices

Some systems implement a **registry** - used to store and retrieve configuration information

FILE MODIFICATION

Text editors to create and modify files

Special commands to search contents of files or perform transformations of the text

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 to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

BACKGROUND SERVICES

- Launch at boot time
  - Some for system startup, then terminate
  - Some from system boot to shutdown
- 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

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

Design and Implementation of OS not “solvable”, but some approaches have proven successful

Internal structure of different Operating Systems can vary widely

Affected by choice of hardware, type of system

DESIGN GOALS

Start by defining goals and specifications

- **User goals**
  - should be convenient to use, easy to learn, reliable, safe, and fast
- **System goals**
  - should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

MECHANISMS AND POLICIES

Important to separate

- **Policy**: *What* will be done?
- **Mechanism**: *How* to do it?

The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

IMPLEMENTATION

Much variation

- Early OSes in assembly language
- Then system programming languages like Algol, PL/1
- Now C, C++

IMPLEMENTATION

Actually usually a mix of languages

- Lowest levels in assembly
- Main body in C
- Systems programs in C, C++, scripting languages like PERL, Python, shell scripts

More high-level language easier to **port** to other hardware → But slower

Emulation can allow an OS to run on non-native hardware

OPERATING-SYSTEM STRUCTURE

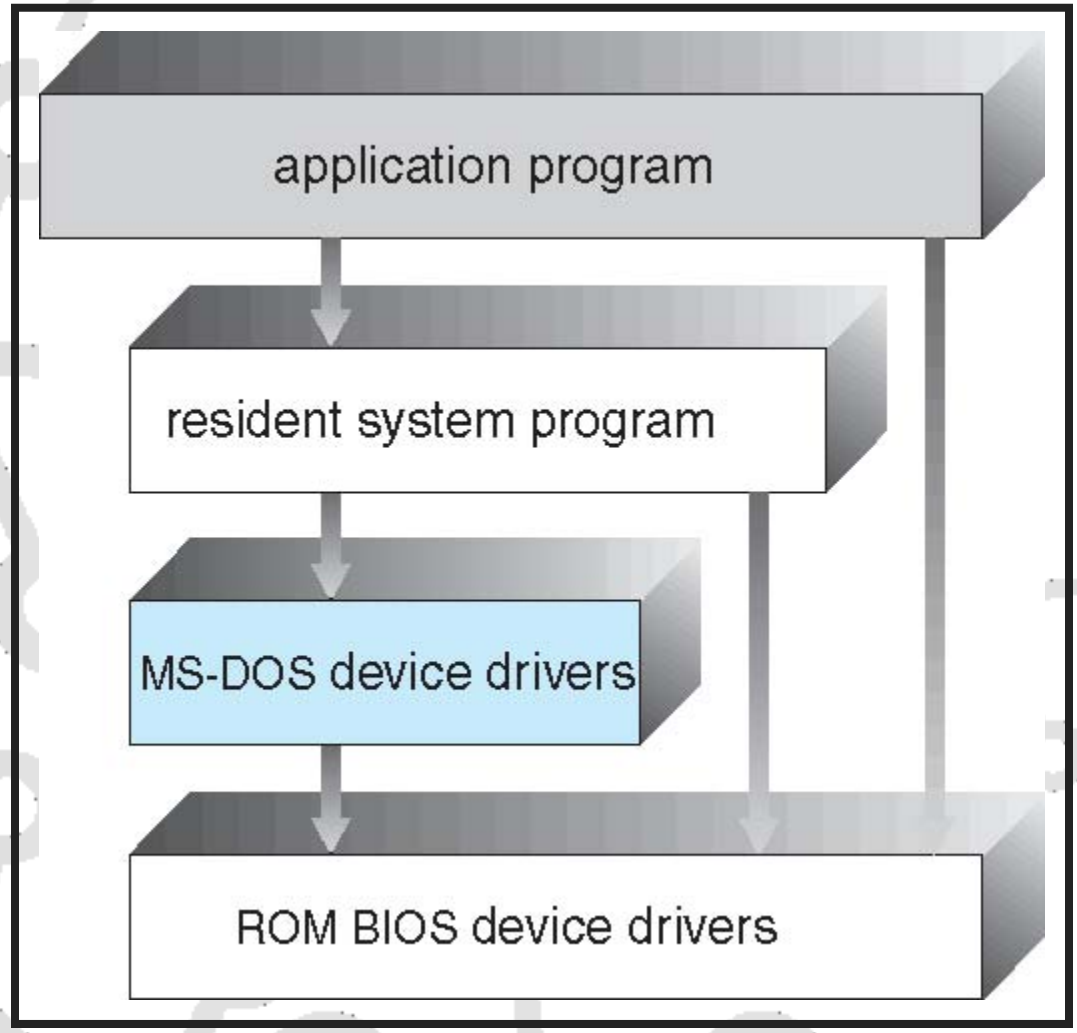
OPERATING-SYSTEM STRUCTURE

General-purpose OS is very large program

Various ways to structure one as follows

SIMPLE STRUCTURE

MS-DOS LAYER STRUCTURE

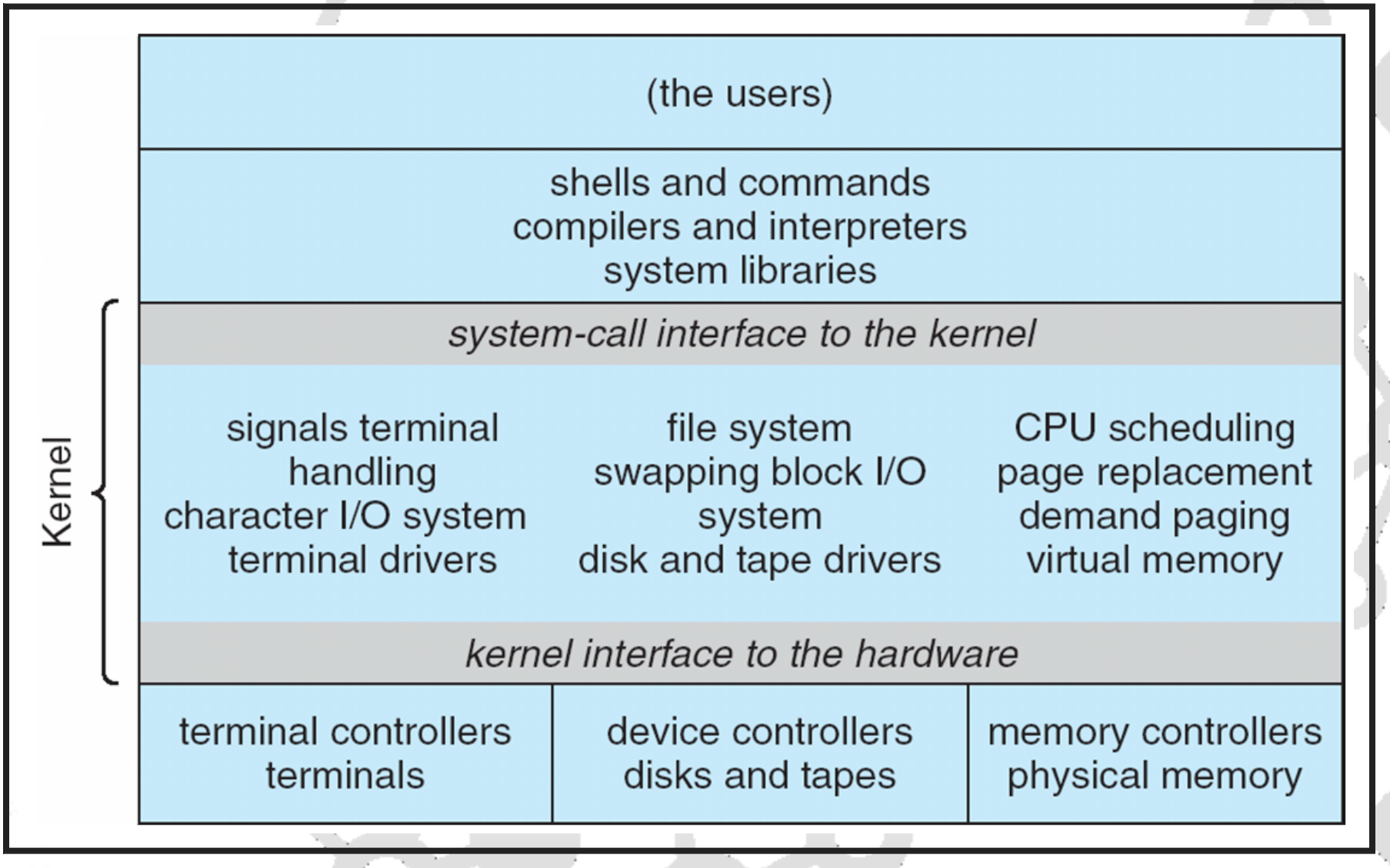


UNIX SYSTEM STRUCTURE

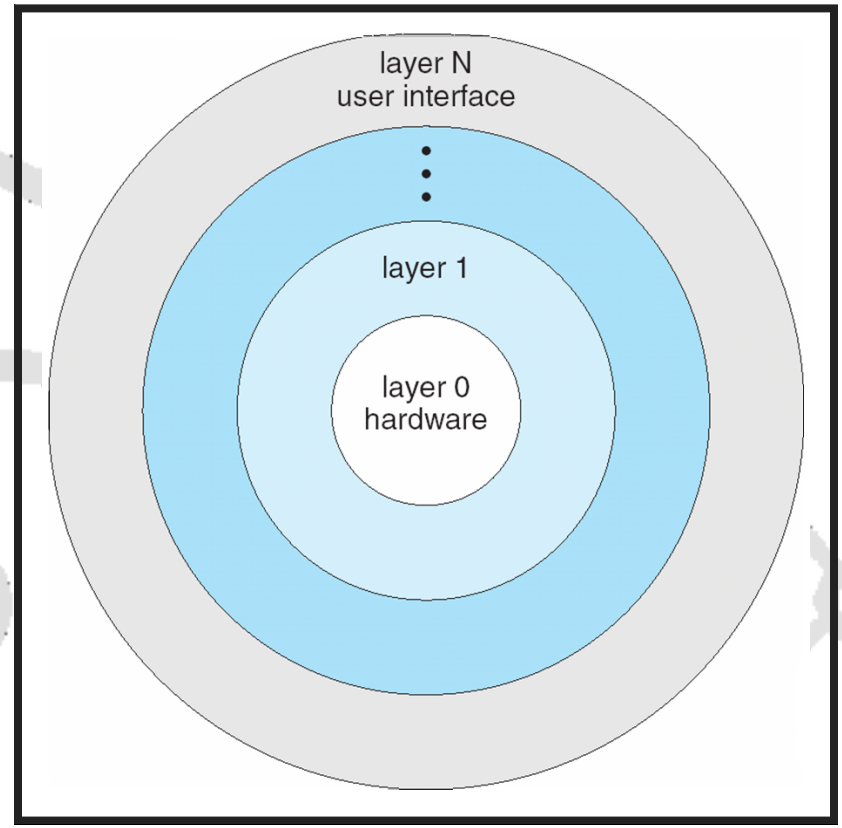
UNIX – limited by hardware functionality, the original UNIX OS had limited structuring. Consists of two separable parts

1. Systems programs
2. 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

UNIX SYSTEM STRUCTURE



LAYERED APPROACH



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

MICROKERNELS

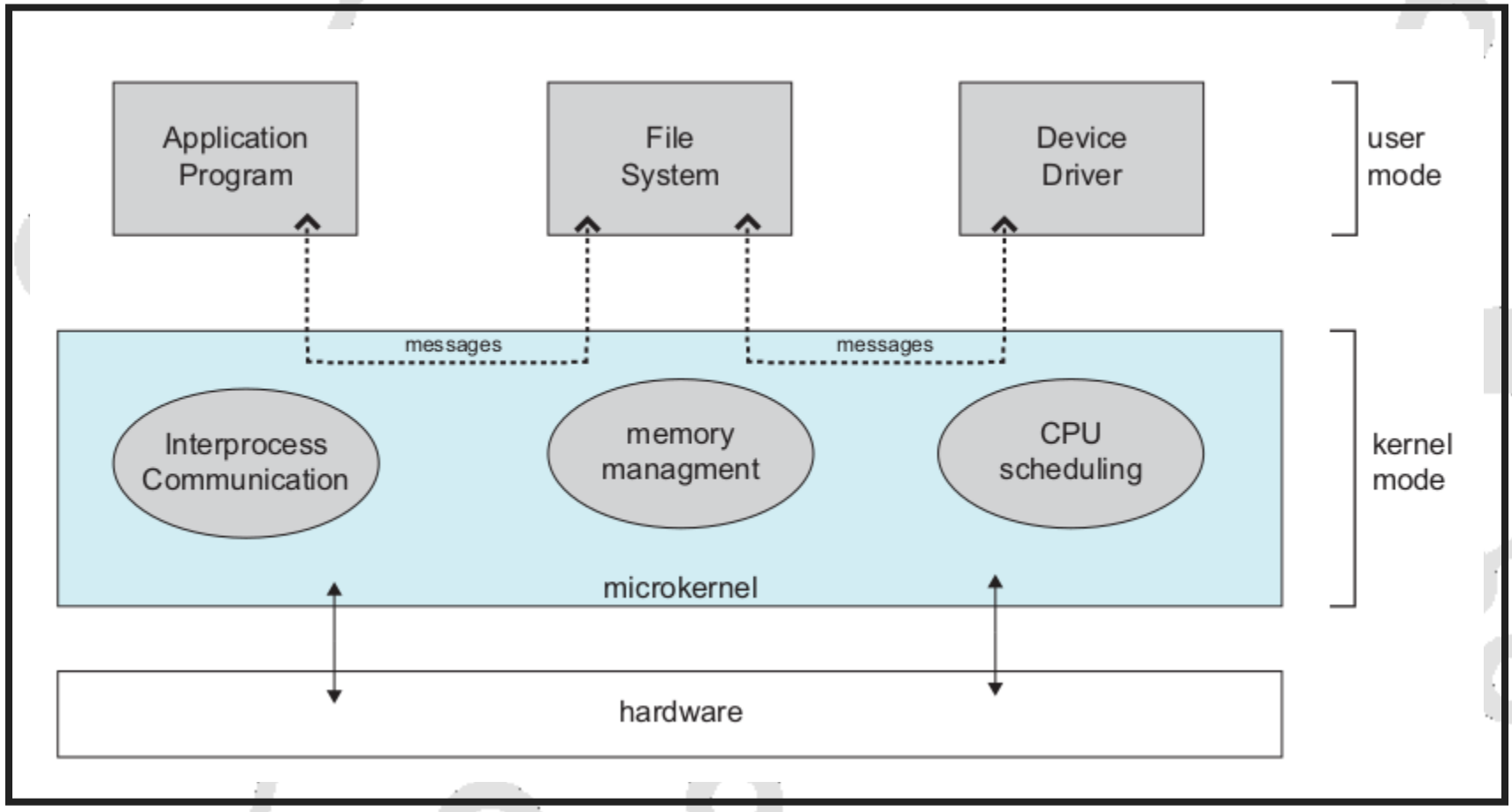
Benefits

- 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

MICROKERNELS



MODULES

Most modern operating systems implement **loadable kernel modules**

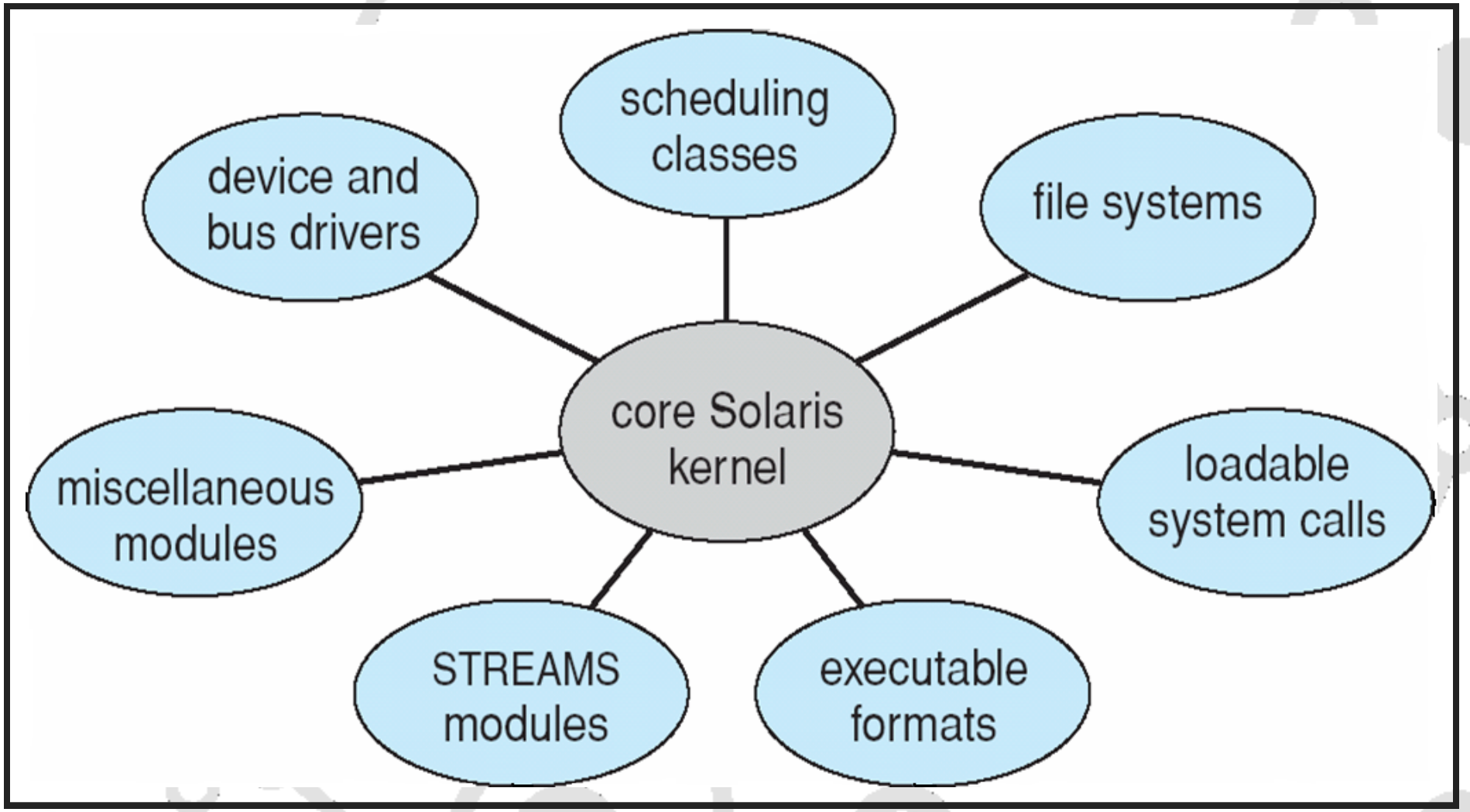
- Uses object-oriented approach
- Each core component is separate
- Each talks to the others over known interfaces
- Each is loadable as needed within the kernel

MODULES

Overall, similar to layers but with more flexibility, as all modules can call other modules

Used in modern UNIX (Solaris), Linux, Mac OS and Windows

SOLARIS LOADABLE MODULES



HYBRID SYSTEMS

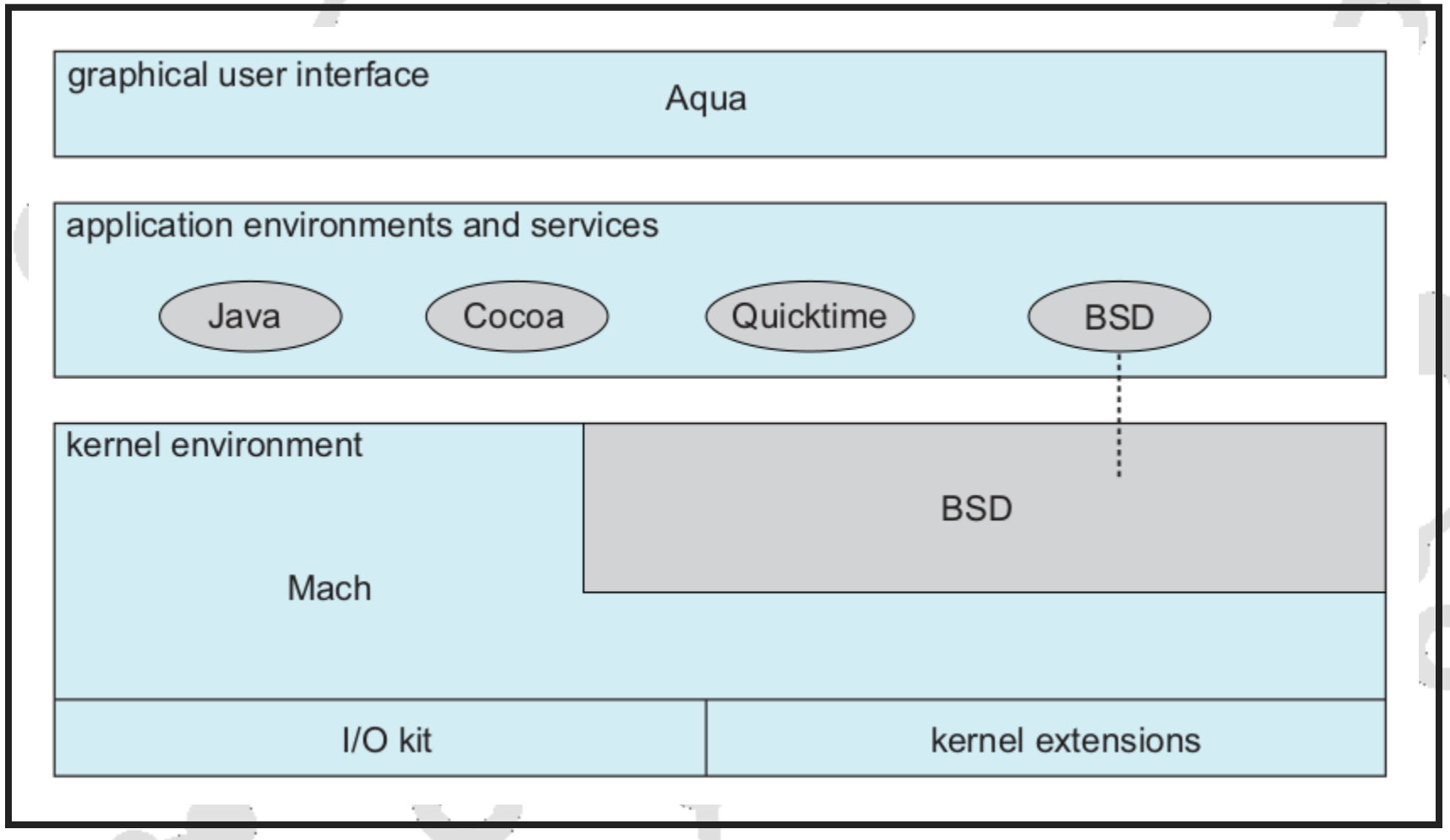
- Most modern operating systems actually not one pure model

  Hybrid combines multiple approaches to address performance, security, usability needs

HYBRID SYSTEMS

- 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

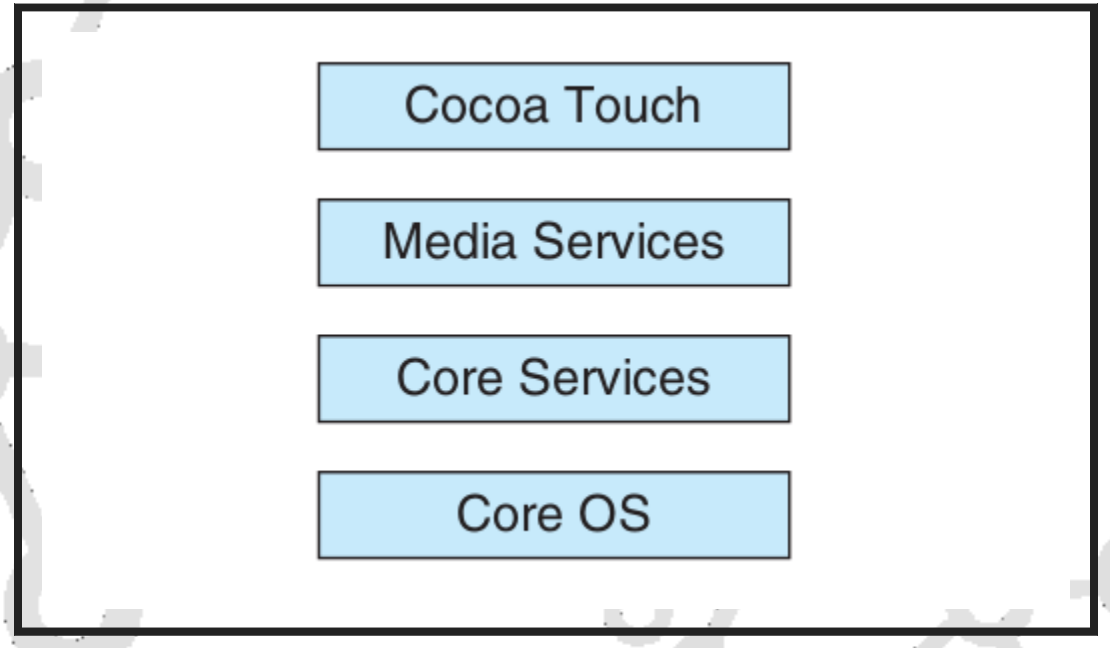


IOS

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

IOS



ANDROID

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

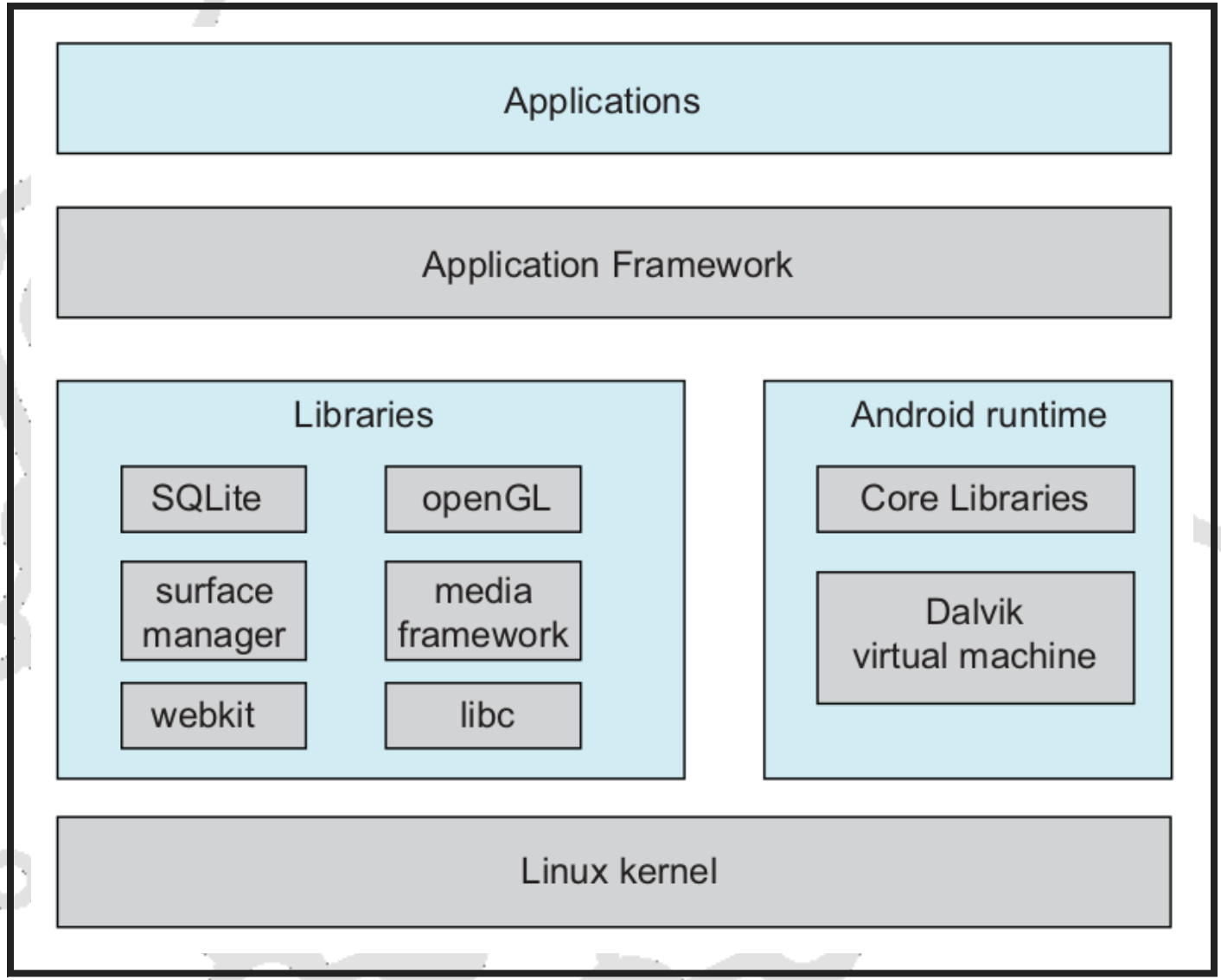
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and **Dalvik** virtual machine

ANDROID

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 ARCHITECHTURE



OPERATING-SYSTEM DEBUGGING

DEBUGGING

Debugging: The activity of finding and fixing errors in a system, both in hardware and in software

FAILURE ANALYSIS

OSes 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

FAILURE ANALYSIS

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

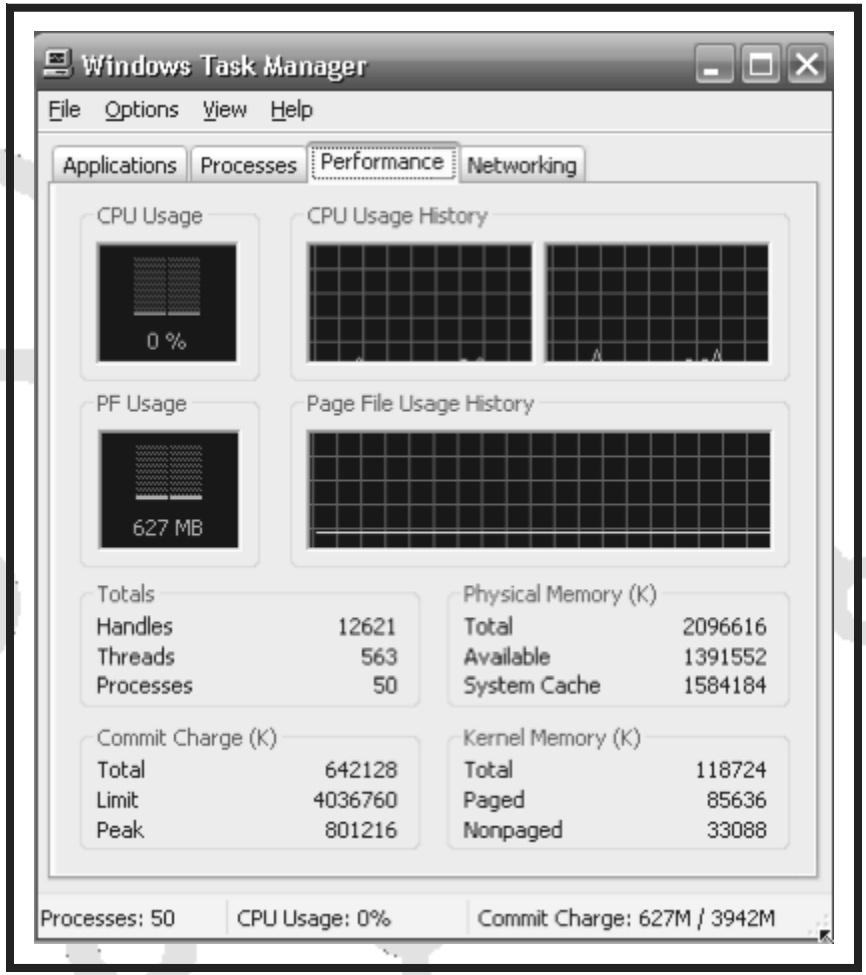
— Kernighan's Law

PERFORMANCE TUNING

Improve performance by removing bottlenecks

OS must provide means of computing and displaying measures of system behavior

WINDOWS TASK MANAGER



DTRACE

DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems

Probes fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes

DTRACE

Composed of

- compiler
- a framework
- **providers** of probes (written in framework)
- **consumers** of these probes

Only users with DTrace priveledges are allowed to use DTrace

DTRACE

```
# ./all.d `pgrep xclock` XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                       U
  0   -> _XEventsQueued                    U
  0     -> _X11TransBytesReadable          U
  0     <- _X11TransBytesReadable          U
  0     -> _X11TransSocketBytesReadable    U
  0     <- _X11TransSocketBytesreadable    U
  0     -> ioctl                           U
  0       -> ioctl                         K
  0         -> getf                        K
  0           -> set_active_fd             K
  0           <- set_active_fd             K
  0         <- getf                        K
  0         -> get_udatamodel              K
  0         <- get_udatamodel              K
...
  0         -> releasef                    K
  0           -> clear_active_fd           K
  0           <- clear_active_fd           K
  0           -> cv_broadcast              K
  0           <- cv_broadcast              K
  0         <- releasef                    K
  0       <- ioctl                         K
  0     <- ioctl                           U
  0   <- _XEventsQueued                    U
  0 <- XEventsQueued                       U
```

DTRACE

```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
   @time[execname] = sum(timestamp - self->ts);
   self->ts = 0;
}
```

DTRACE

```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
        gnome-settings-d                142354
        gnome-vfs-daemon                158243
        dsdm                            189804
        wnck-applet                     200030
        gnome-panel                     277864
        clock-applet                    374916
        mapping-daemon                  385475
        xscreensaver                    514177
        metacity                        539281
        Xorg                           2579646
        gnome-terminal                 5007269
        mixer_applet2                  7388447
        java                          10769137
```

Figure 2.21 Output of the D code.

OPERATING-SYSTEM GENERATION

OPERATING-SYSTEM GENERATION

Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site

OPERATING-SYSTEM GENERATION

System Generation (**SYSGEN**) program obtains information concerning the specific configuration of the hardware system

- Used to build system-specific compiled kernel or system-tuned
- Can general more efficient code than one general kernel

OPERATING-SYSTEM GENERATION

The kinds of information that must be determined

- CPU is used?
  - What options (extended instruction sets, floating-point arithmetic, etc.) are installed?
- Partitions on disk
- Boot partition formatting
- How much memory is available
- What devices are available

SYSTEM BOOT

SYSTEM BOOT

Bootstrap program or **bootstrap loader** locates the kernel

I.e. the instruction register is loaded with a predefined memory location, and execution starts there.

This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup.

BOOTSTRAP CODE

Changing the bootstrap code requires changing the ROM hardware chips.

Some systems resolve this problem by using erasable programmable read-only memory (**EPROM**), which is read-only except when explicitly given a command to become writable.

FIRMWARE

All forms of ROM are known as **firmware**, since their characteristics fall between hardware and software.

A disk that has a **boot partition** is called a **boot disk** or **system disk**.

EPROM

http://thenextweb.com/insider/2016/02/01/running-a-single-delete-command-can-permanently-brick-laptops-from-inside-linux/

github issue and discussion

QUESTIONS

BONUS

- Exam question number 1: **Operating-System Structures**