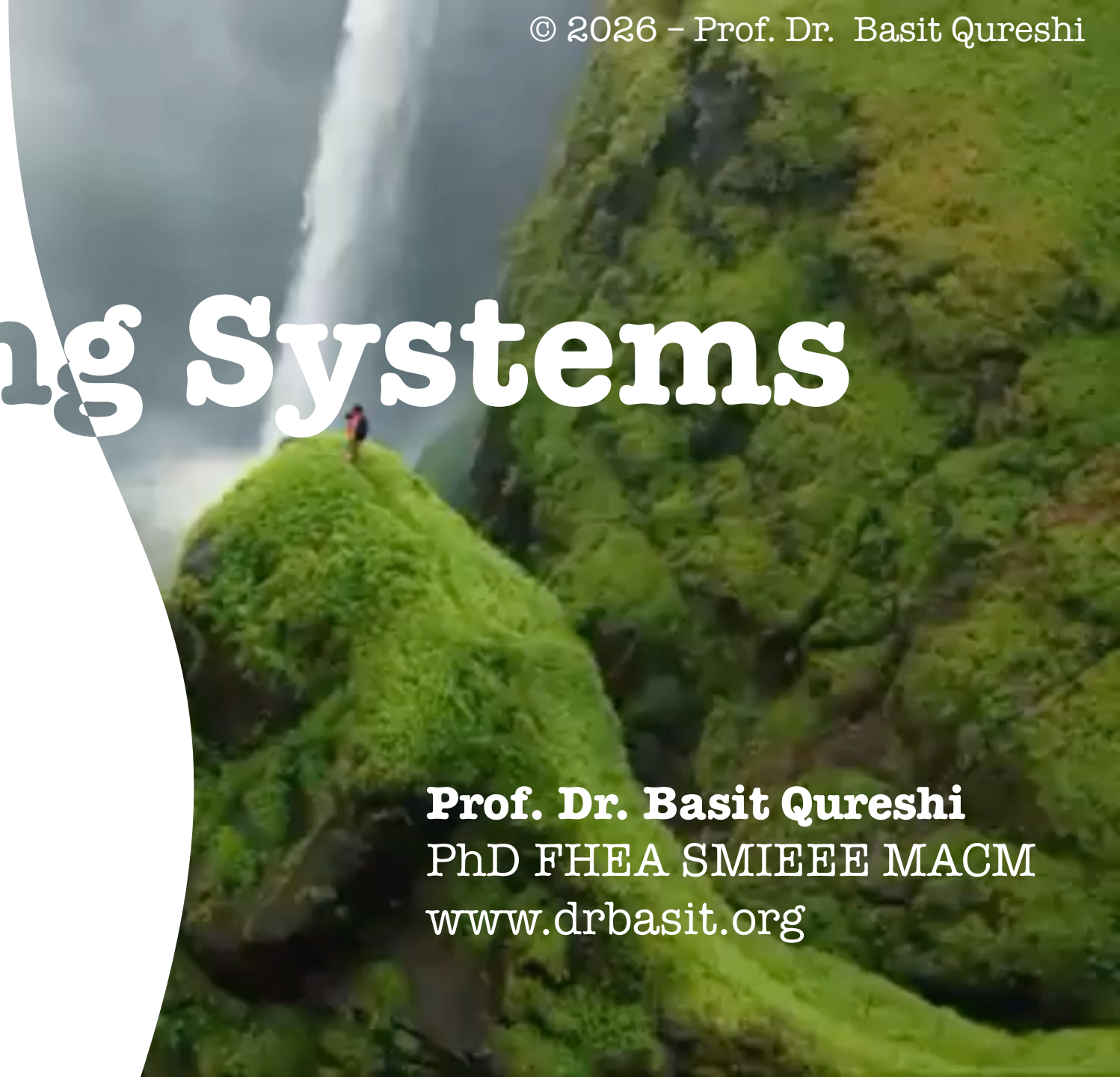
CS330 Operating Systems



OS SERVICES

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

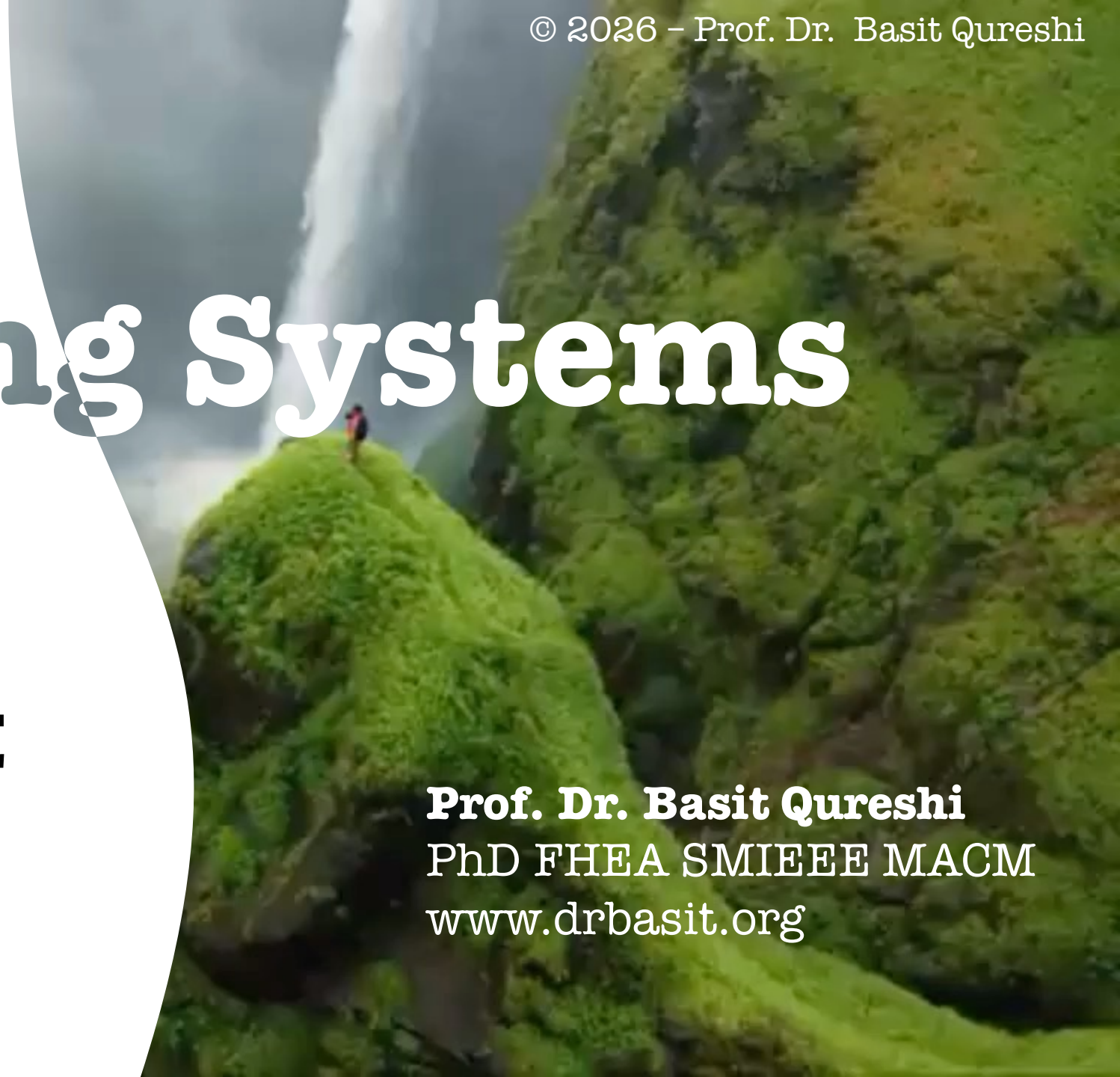
TOPICS

- Operating System user interface
- System Calls
- Operating System Structure
- Building an OS

Credit:
These notes are extracted from the following resources:

- Silberschatz, Galvin and Gagne, *Operating System Concepts*, 10 ed, Wiley, 2018
- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, CreateSpace, 2018
- Robert Love, *Linux Kernel Development*, Addison-Wesley, 2010
- William Shotts, *The Linux command line*, No Starch Press, 2026

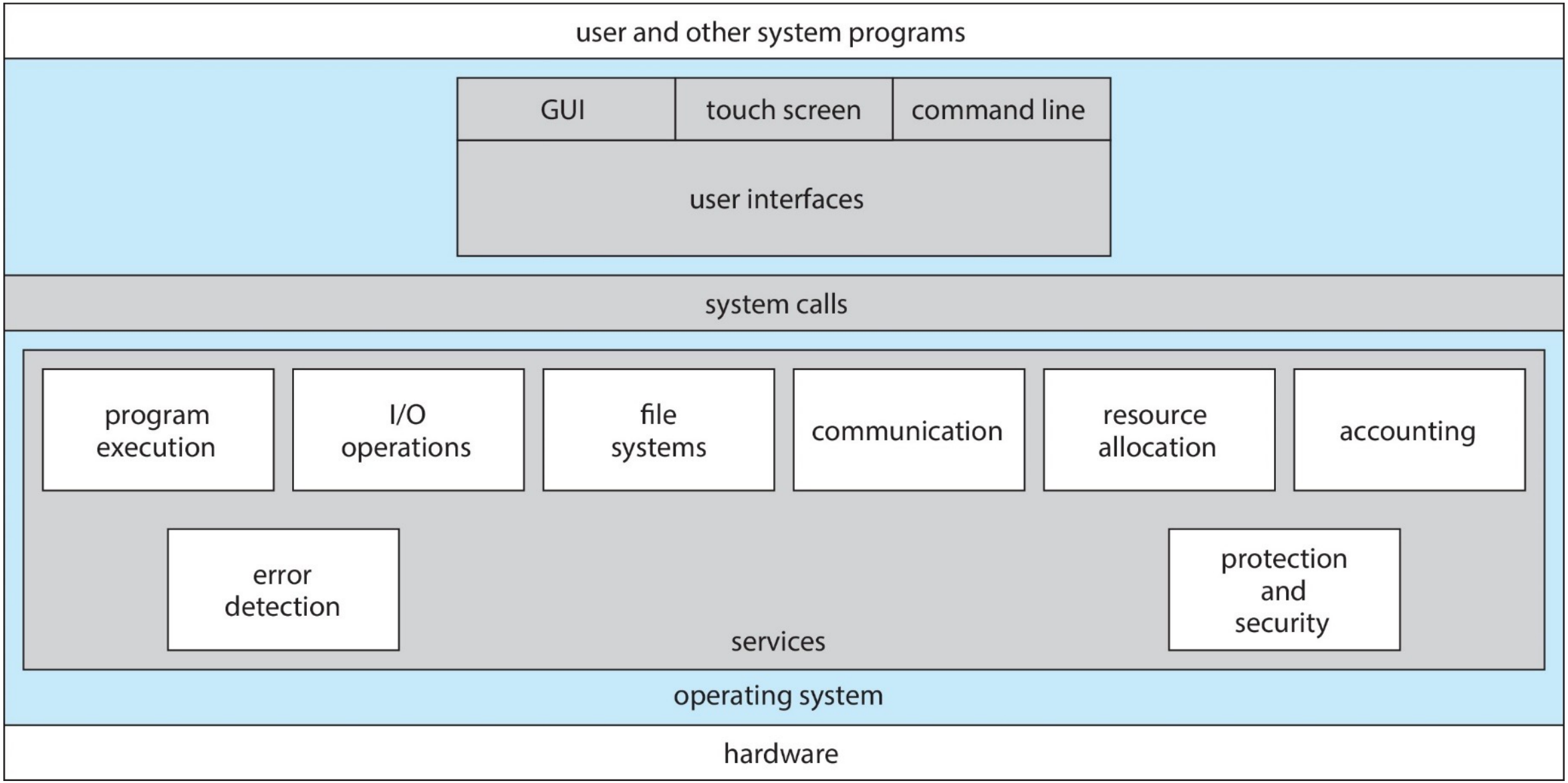
CS330 Operating Systems



OS USER INTERFACE

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

A VIEW OF OPERATING SYSTEM SERVICES



OPERATING SYSTEM SERVICES

OS provides an **environment** for **execution** of *programs* and *services to* ***programs*** *and* ***users***.

OS services provides functions that are **helpful** to the **user**:

- ***User Interface-*** -Almost all operating systems have a user interface (**UI**). Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**
- ***Program execution*** –loads a program into memory and run it. OS is able to end a program's execution either normally or abnormally.
- ***I/O operations –*** since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.

OPERATING SYSTEM SERVICES

- ***File-system manipulation –*** The OS gives the ability to read, write, create, open, close and delete files / directories.
- ***Error detection –*** ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.
- ***Communications –*** exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via ***shared memory*** or ***message passing*** etc

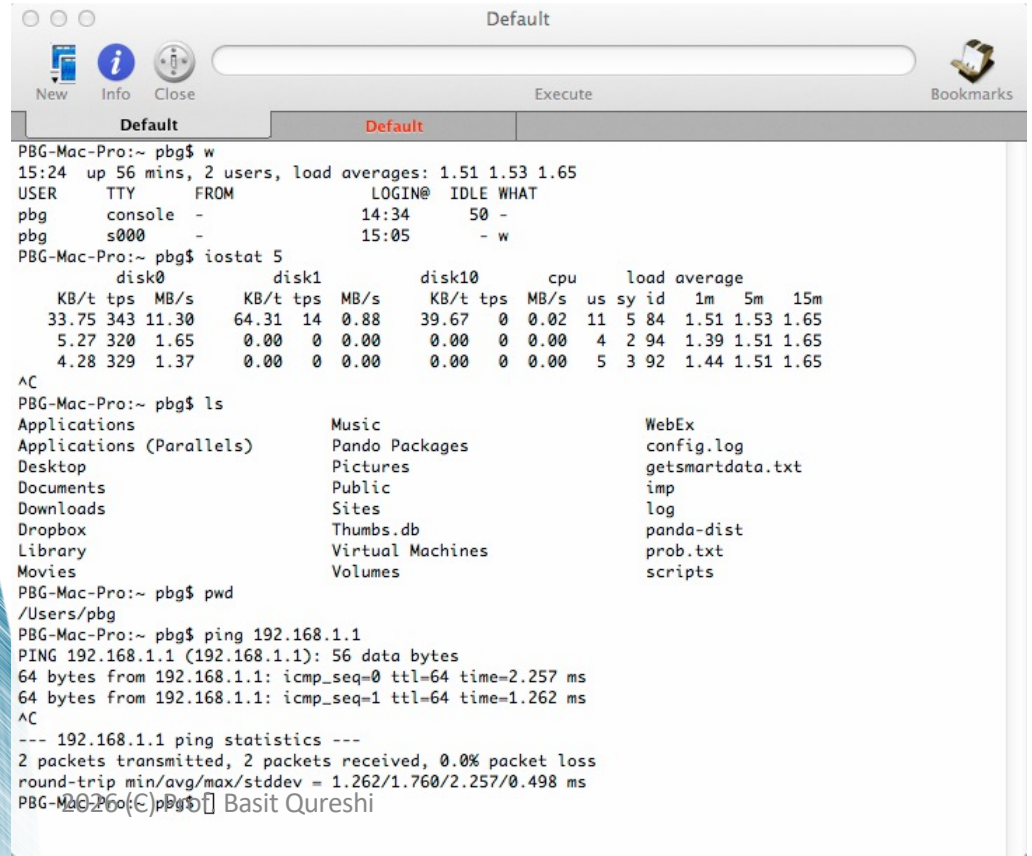
ADDITIONAL OPERATING SYSTEM FUNCTIONS

OS services provides functions that are **helpful** for **efficient system operation**

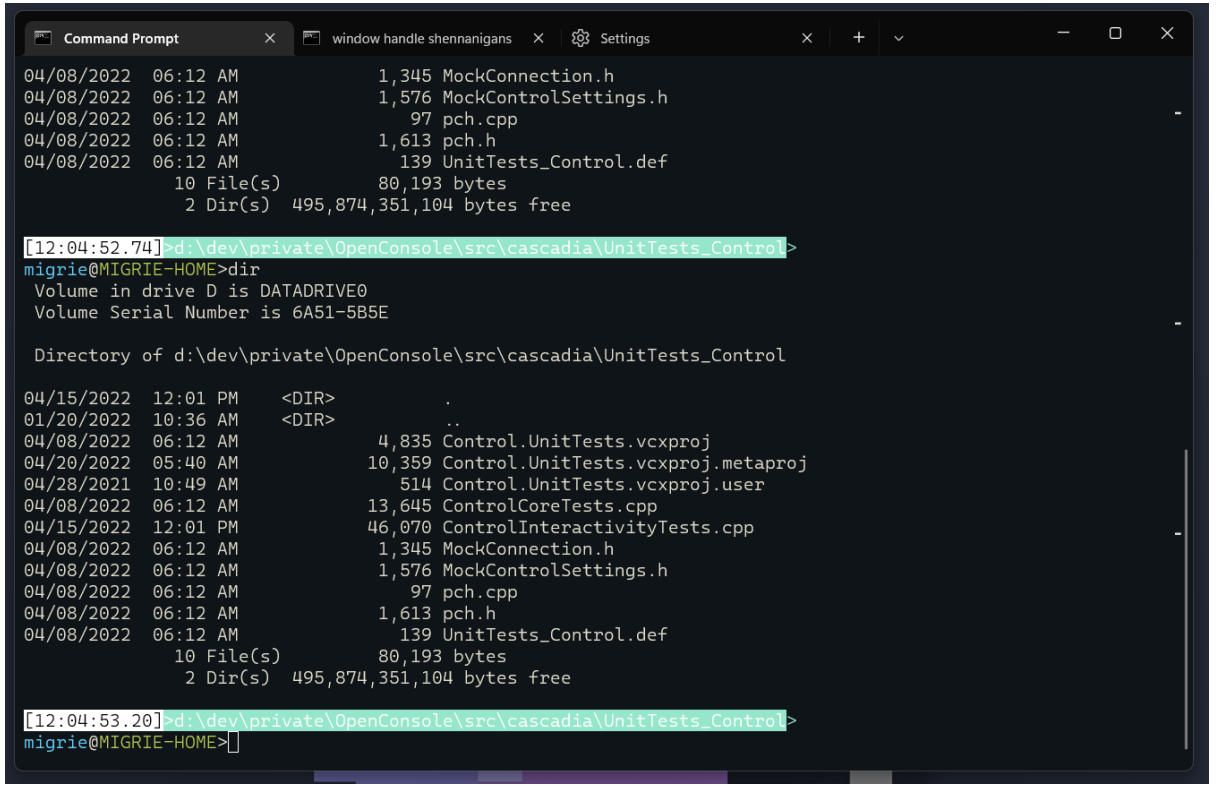
- ***Resource allocation*** – allocating **resources** to *multiple users* or *multiple jobs* running **concurrently**, resources must be allocated to each of them.
- ***Accounting*** – keep track of and record which users use *how much* and *what kinds of computer resources* for account billing or for accumulating usage statistics.
- ***Protection & 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
  - If a system is to be protected and secure, precautions must be instituted throughout it. *A chain is only as strong as its weakest link*.

SERVICES: USER OS INTERFACE - CLI

- CLI or **command interpreter** allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes known as **shells**
  - Fetches a command from user and executes it

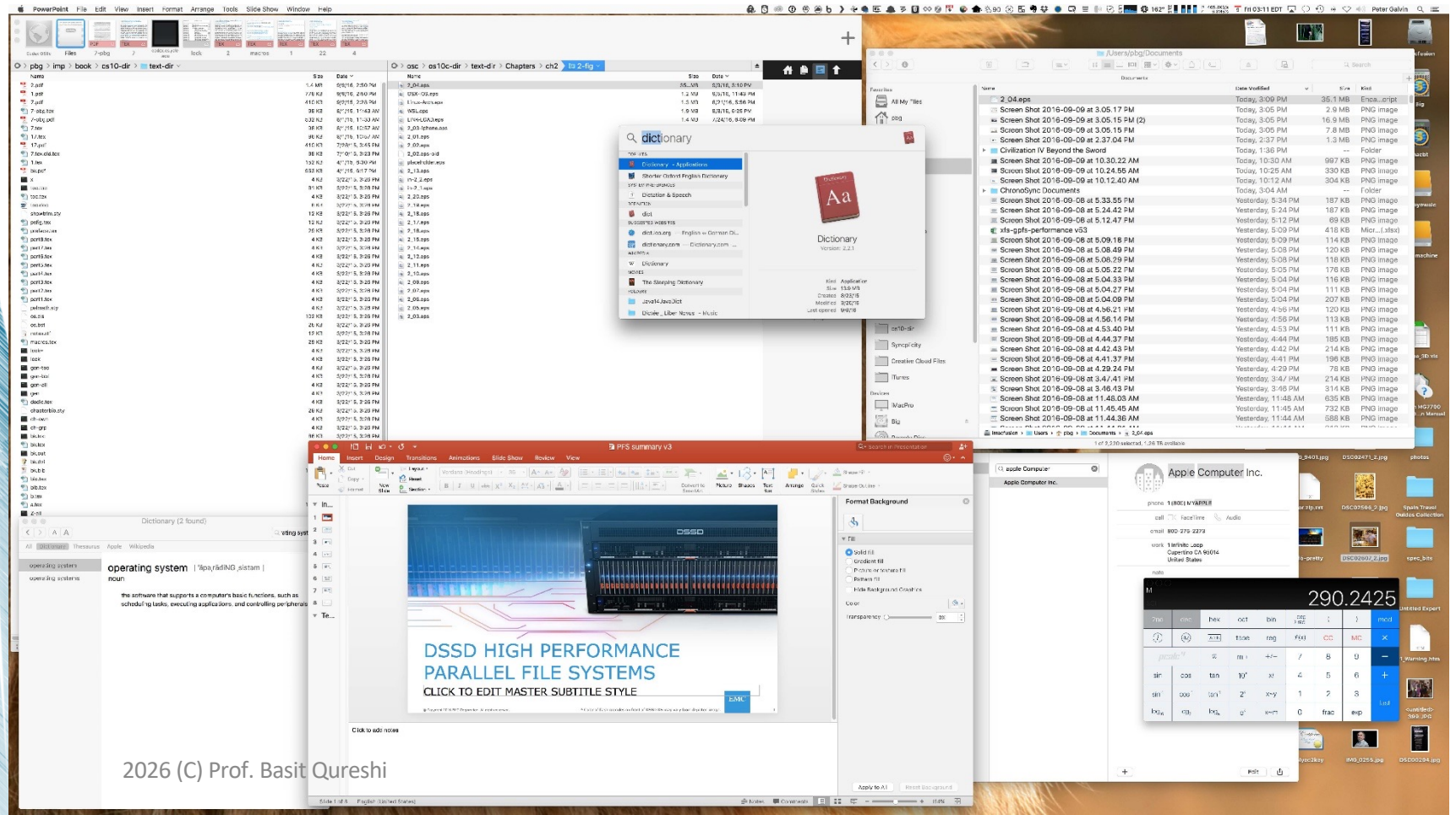


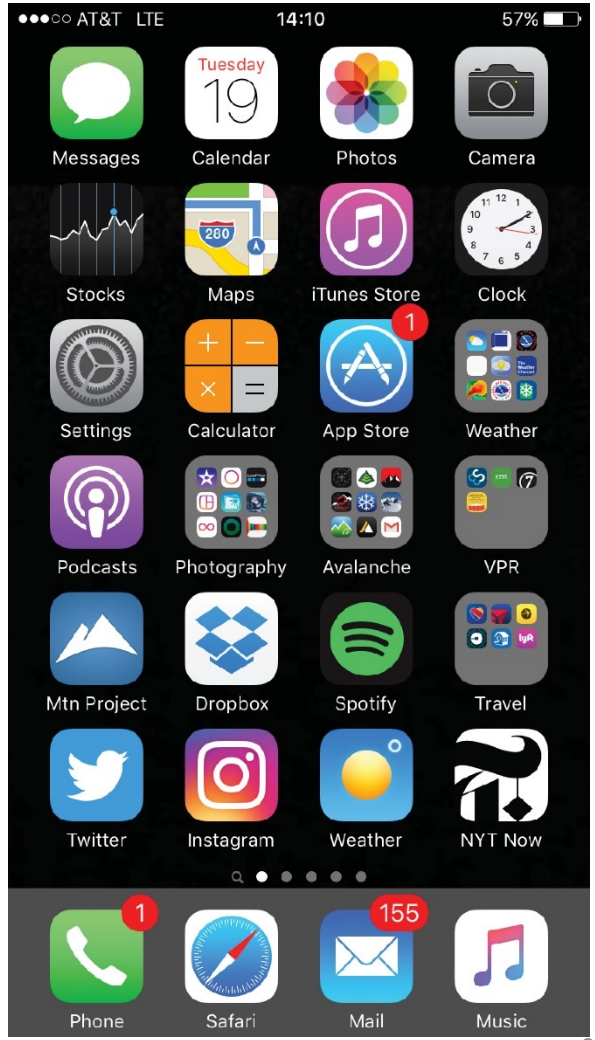
Windows: powershell



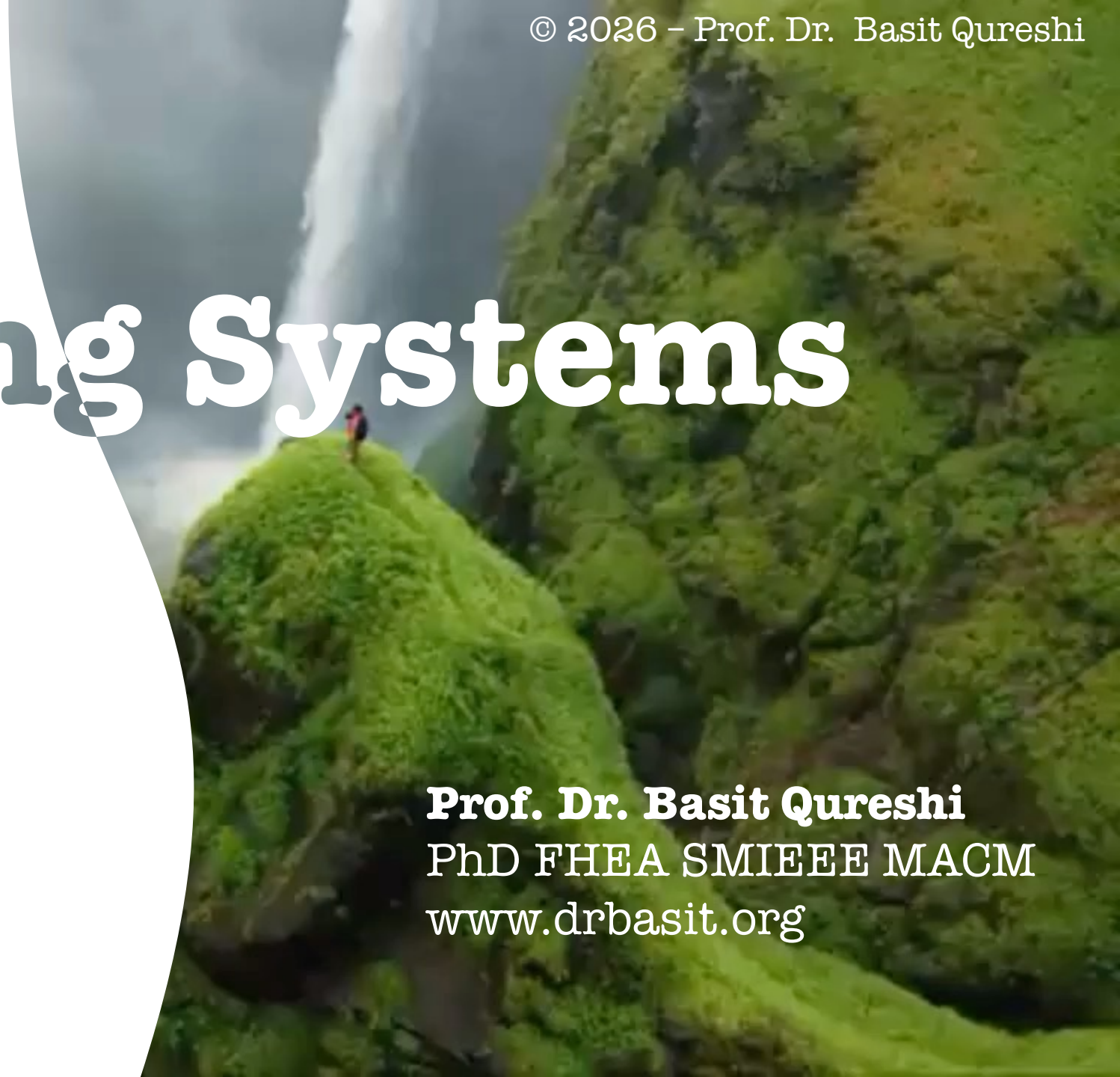
SERVICES: USER OS INTERFACE - GUI

- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
- Touchscreen device interfaces





CS330 Operating Systems



SYSTEM CALLS

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

SYSTEM CALLS

- System calls provide the interface between a running program and 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 use
- 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), and **Java** API for the Java virtual machine (JVM)

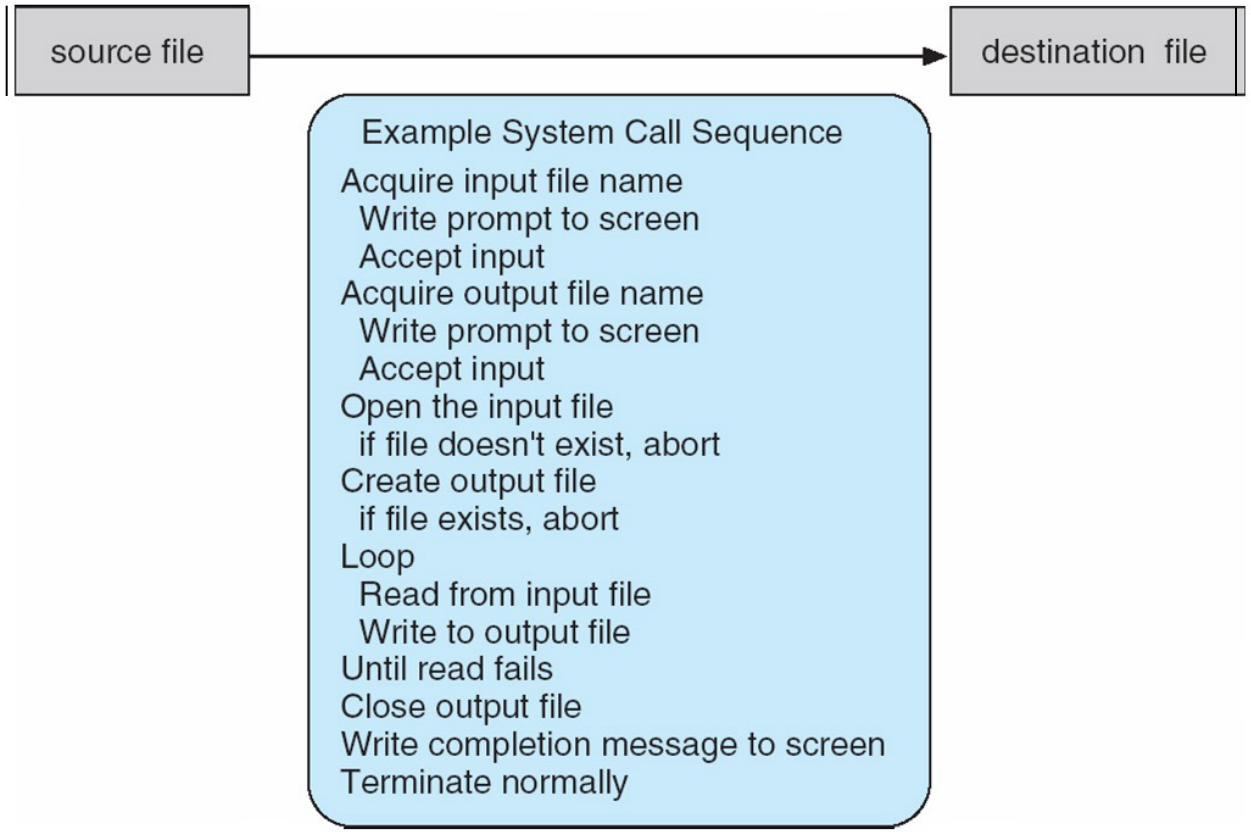
POSIX: Portable Operating System Interface

Created by IEEE. Standardizes System calls such as

- File operations (**open, read, write, close**)
- Process management (**fork, exec, wait**)
- Signals
- Threads (**pthread**)

EXAMPLE OF SYSTEM CALLS

- System call sequence to copy the contents of one file to another file



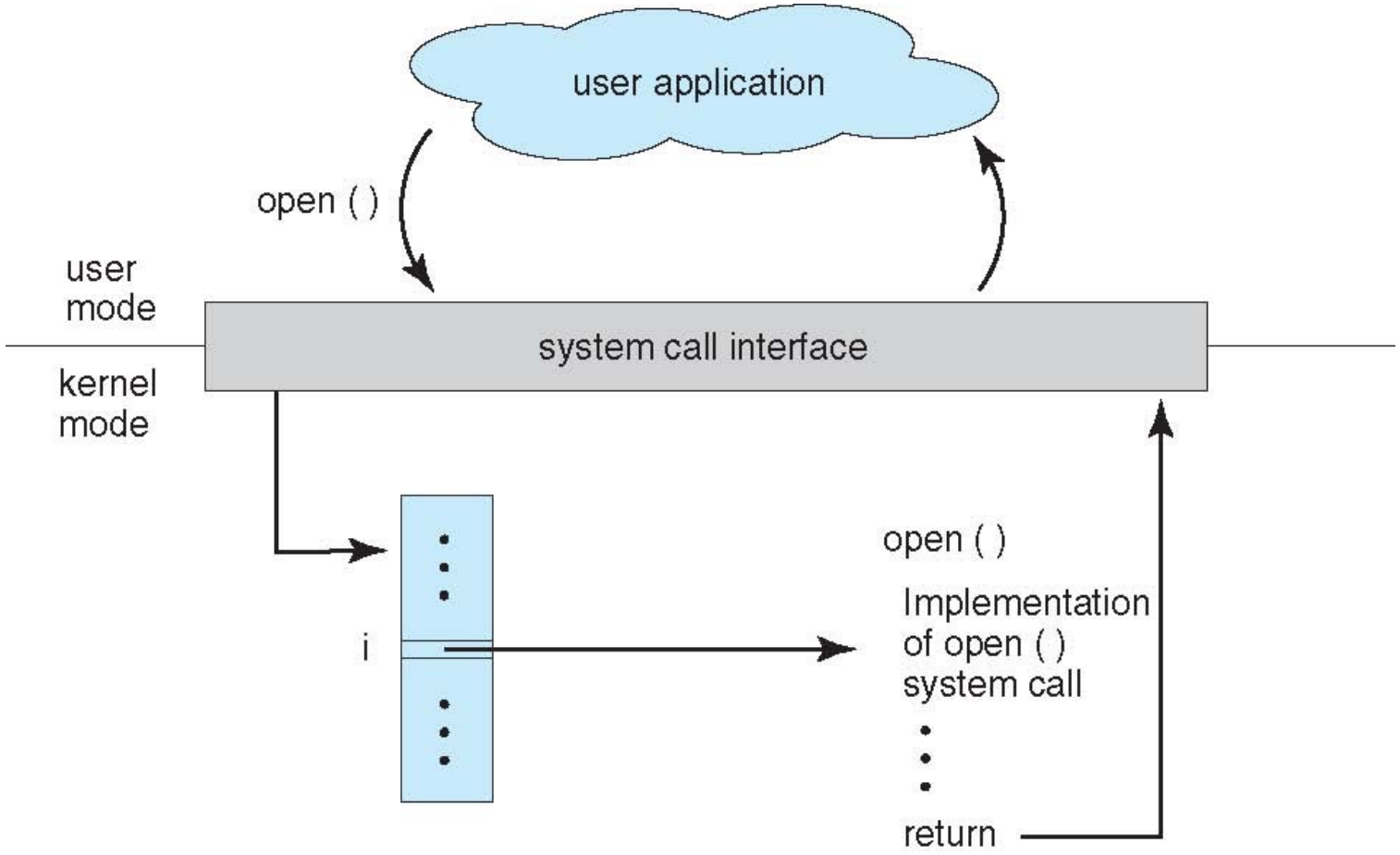
SYSTEM CALL IMPLEMENTATION

- Typically, a number associated with each system call
  - **System-call interface** maintains a **table** indexed according to these numbers

- The **system call interface** invokes intended system call in **OS kernel** and returns **status** of the system call and any **return** values

- The caller needs 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)

API – SYSTEM CALL – OS RELATIONSHIP



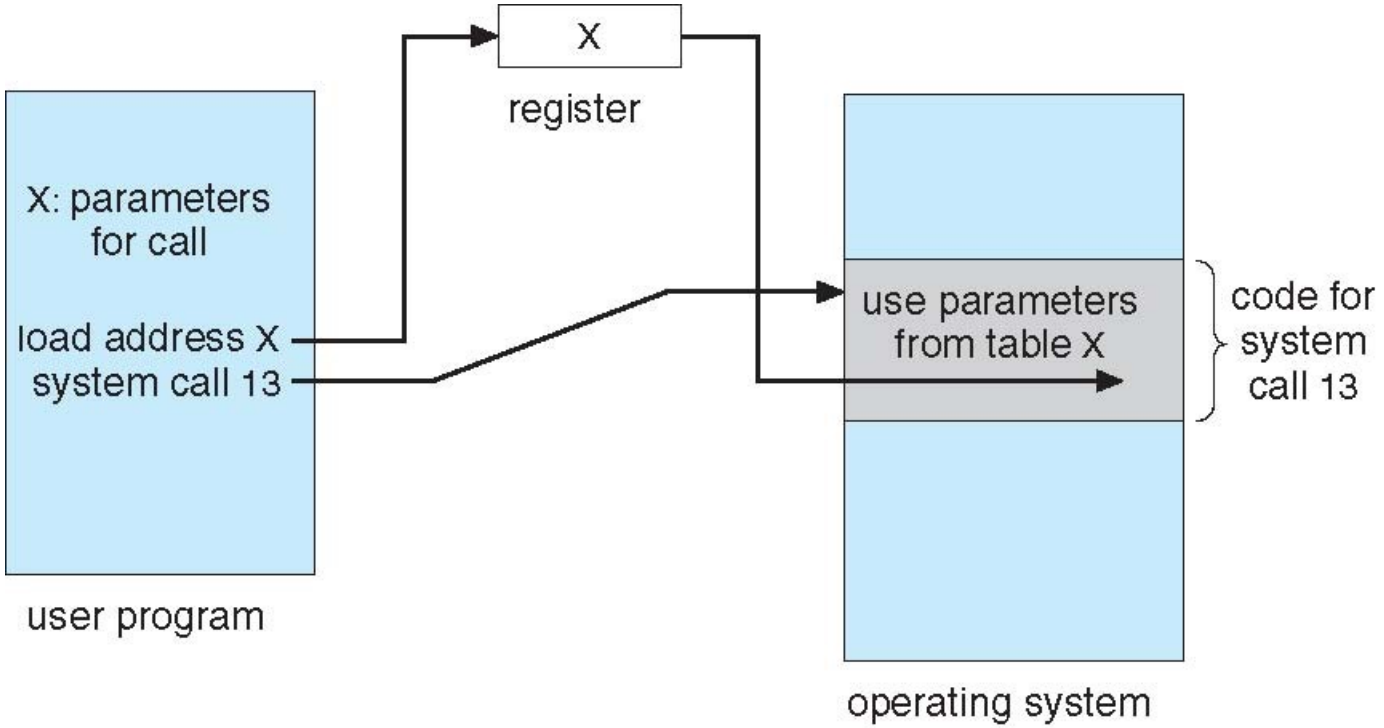
SYSTEM CALL PARAMETER PASSING

- Often, more information is required than simple identity of desired system call
  - Exact type and amount of information vary according to OS and call

- Three general **methods** used to pass parameters to the OS
  1. **Simplest**: pass the parameters in **registers**
     - In some cases, may be more parameters than registers
  2. Parameters stored in a **block**, or **table**, in memory, and *address* of block passed as a parameter in a register
     - This approach taken by Linux and Solaris
  3. Parameters placed, or *pushed,* onto the **stack** by the program and *popped* off the stack by the operating system

* Block and stack methods do not limit the number or length of parameters being passed

PASSING OF PARAMETERS AS A TABLE



TYPES OF SYSTEM CALLS

System Calls can be grouped into 6 categories

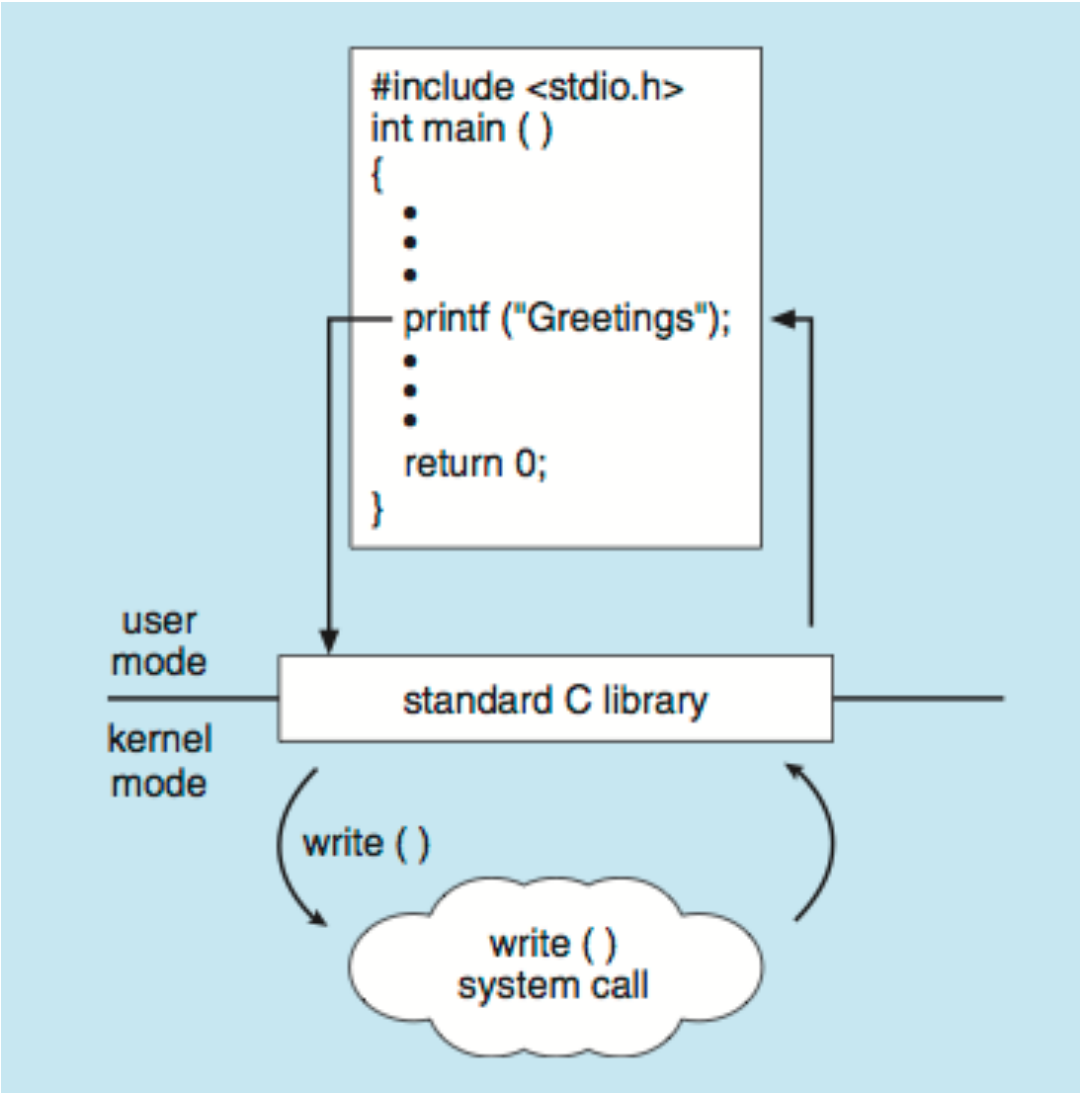
- **Process control** – create/terminate process, load/execute, end/abort, etc.
- **File management-** create/delete file, open/close, read/write, get/set file attributes
- **Device management**- read/write, get/set device attributes, request/release device.
- **Information maintenance-** get/set time or date, get/set file attributes, get/set process/device attributes.
- **Protection** – Control access to resources, Get and set permissions, Allow and deny user access
- **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

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()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

STANDARD C LIBRARY EXAMPLE

- C program invoking printf() library call, which calls write() system call



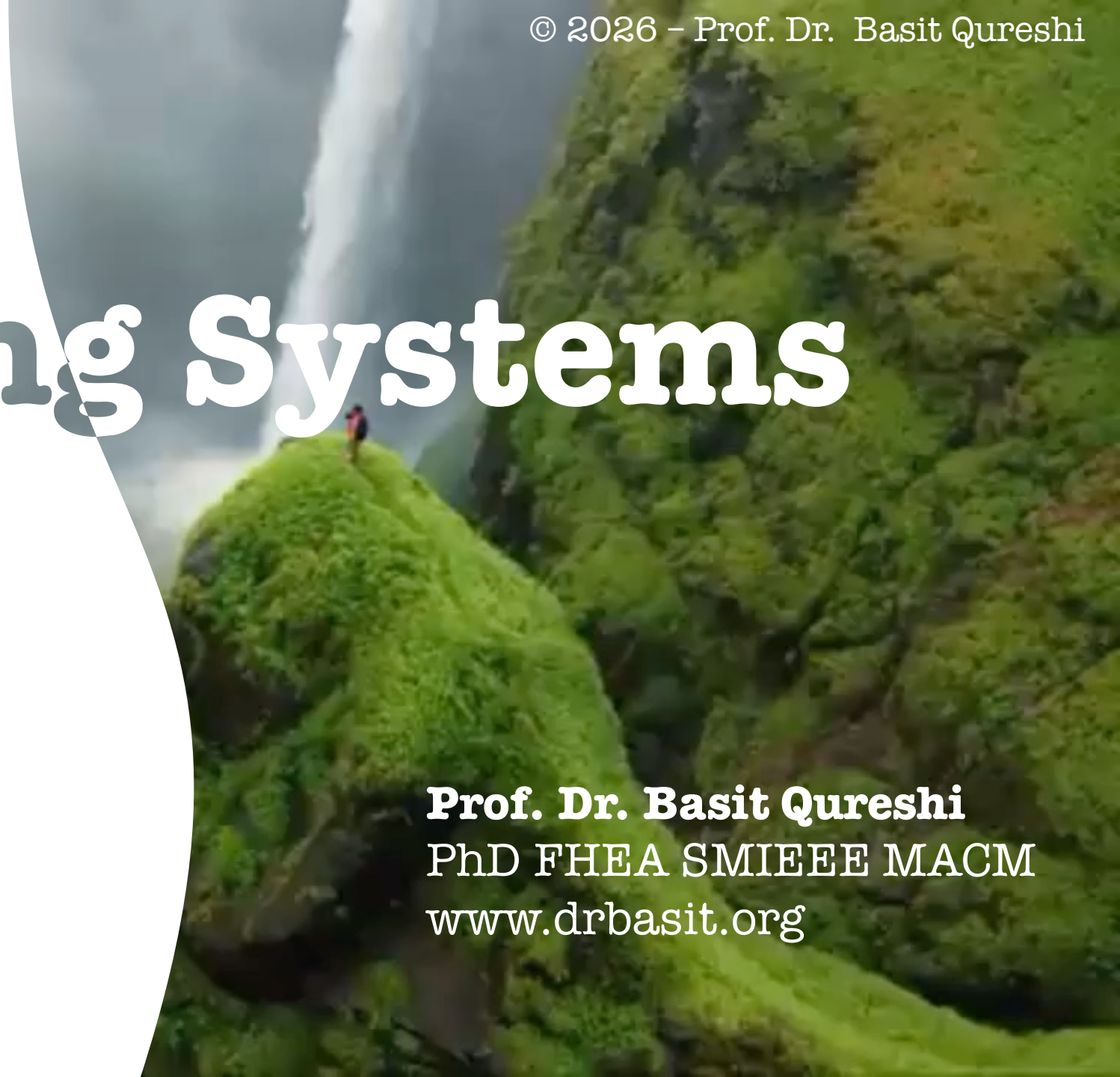
SYSTEM PROGRAMS

- File manipulation : create, delete, copy, rename, print, … files and directories
- Status information: Some programs simply ask the system for the date, time, amount of available memory or disk space,…
- File modification: Several text editors may be available to create and modify the content of files stored on disk or other storage devices.
- Programming language support: Compilers, assemblers, debuggers, and interpreters are often provided with the OS or as a separate download.
- Program loading and execution: The system may provide loaders to load programs into the memory.
- Communications: These programs provide the mechanism for creating virtual connections among processes, users, and computer systems.
- Application programs

Most users' view of a OS is defined by system programs, not the actual system calls.

CS330 Operating Systems

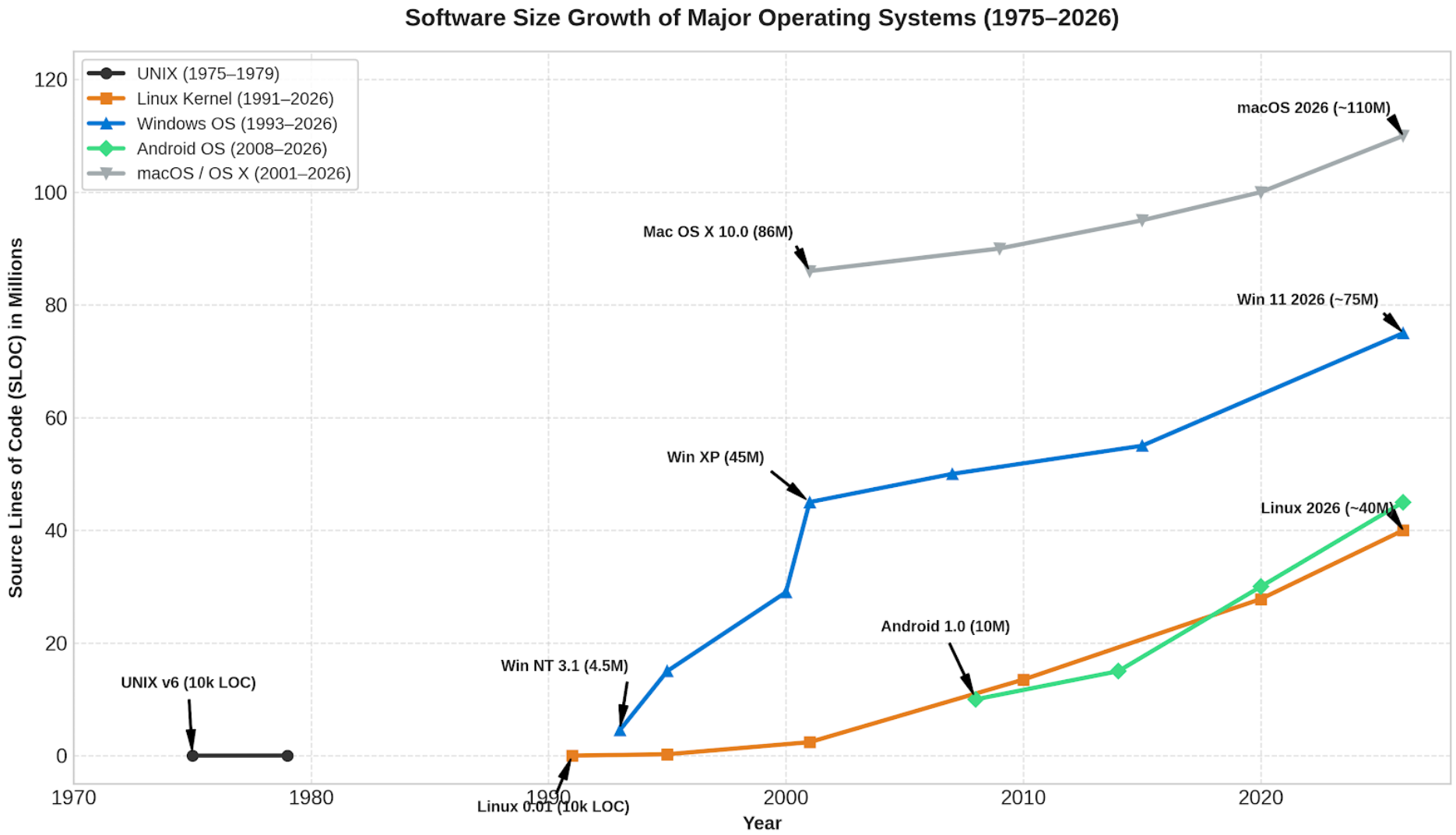
OS STRUCTURE



Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

OPERATING SYSTEM STRUCTURE

- General-purpose OS is **very large program**



OS STRUCTURE APPROACHES

5 popular approaches to structure OS

- **Monolithic** : all the functionality of the kernel placed into a single, static binary file that runs in a single address space.
- **Modular**: Allow kernel components to be added/removed dynamically
- **Layered**: Organize OS into hierarchical layers
- **Microkernel**: Keep kernel minimal; move services outside the kernel
- **Hybrid**: Combine ideas from multiple architectures

`https://en.wikipedia.org/wiki/Unix`

MONOLITHIC

Monolithic : all the functionality of the kernel placed into a single, static binary file that runs in a single address space. Changes to one part of the system can have wide-ranging effects on other parts.

UNIX – (1969 -)

Bell Labs, AT&T, UC Berkeley (BSD), MS-Xenix, SunOS/Solaris, HP-UX and IBM (AIX).

Written in C

Commercialized UNIX System V (1989)

`https://en.wikipedia.org/wiki/Unix`

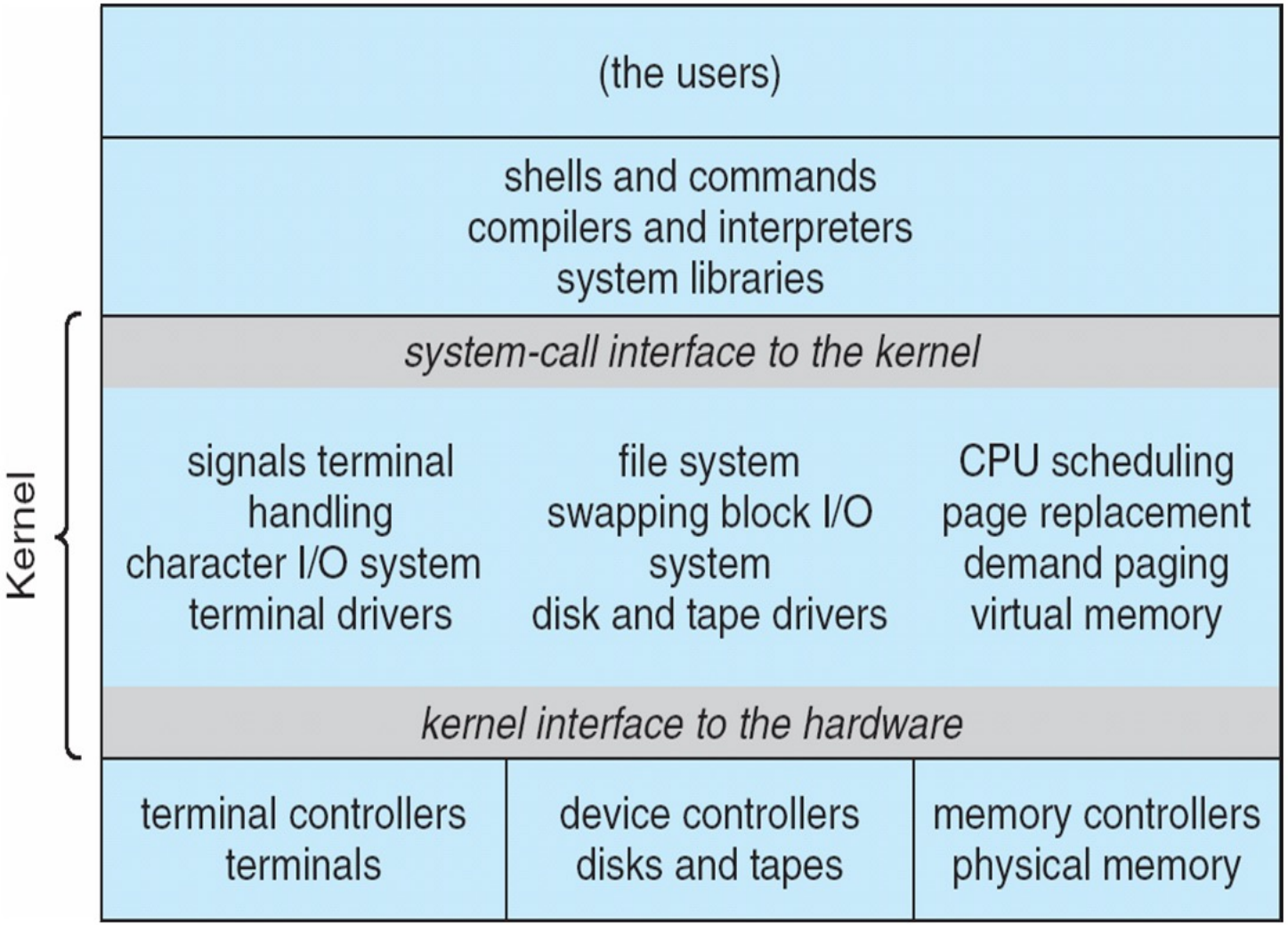
UNIX SYSTEM STRUCTURE

UNIX – limited by hardware

functionality, the original UNIX had limited structuring.

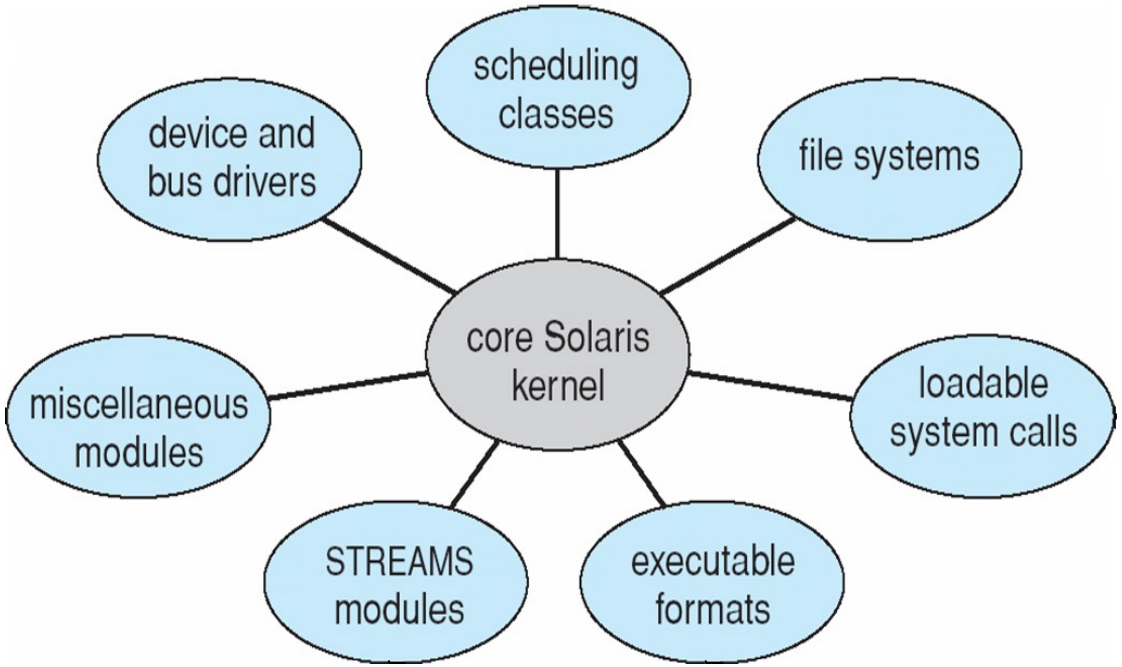
The UNIX OS has two parts:

- Systems programs
- The kernel
  - 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



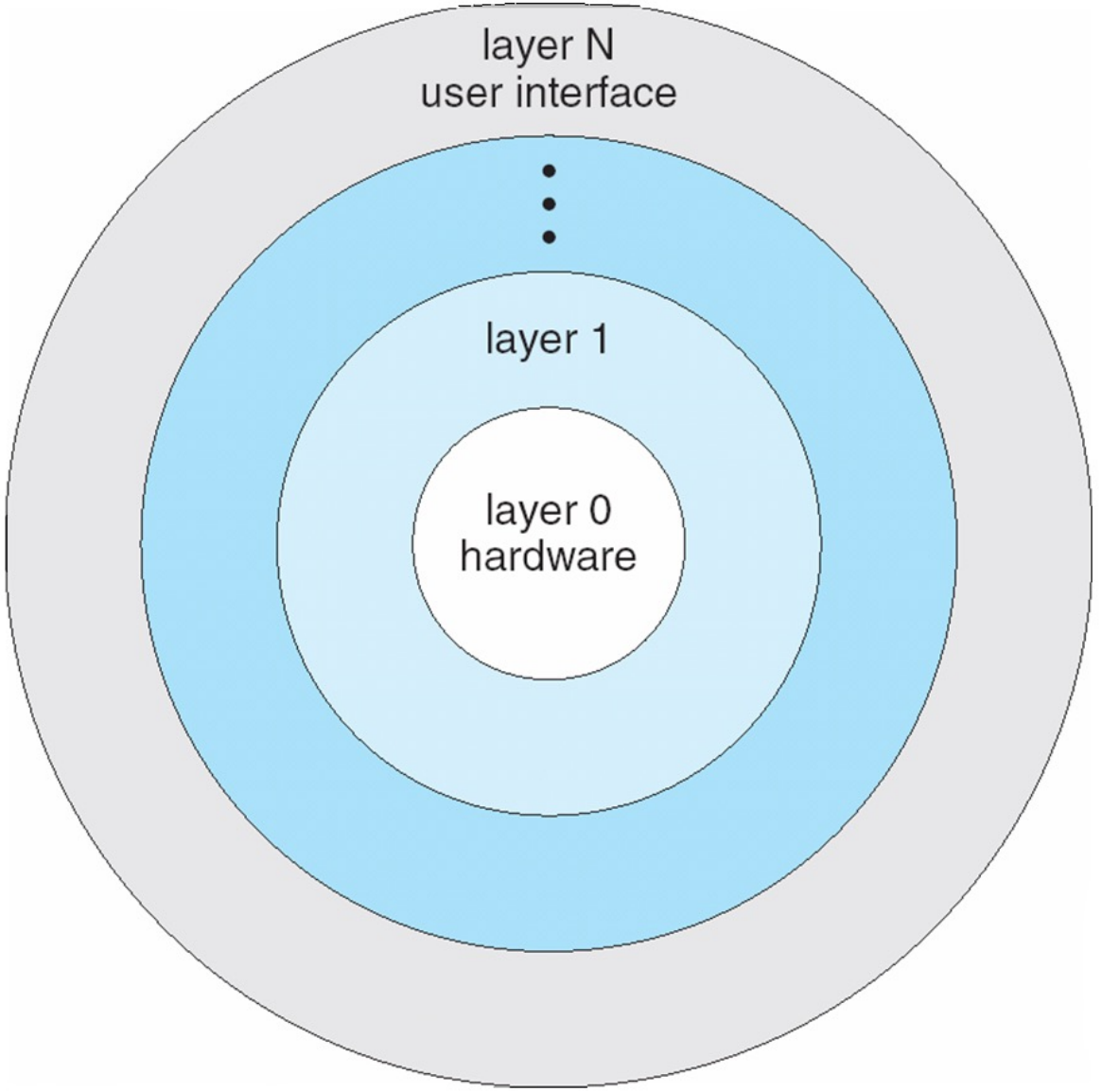
MODULES

- Most modern operating systems implement **loadable kernel modules**
- Core components link in additional services via modules, either at boot time or during run time. This type of design is common in modern implementations of UNIX, such as Linux, macOS, and Solaris, as well as Windows.
  - 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 more flexible.



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.
- Each layer hides the complexity of the layer below. Higher layers don't need to understand the implementation details of lower layers
- The system is divided into separate, smaller components that have specific and limited functionality.
- With **modularity**, layers are selected such that each uses functions (operations) and services of only lower-level layers



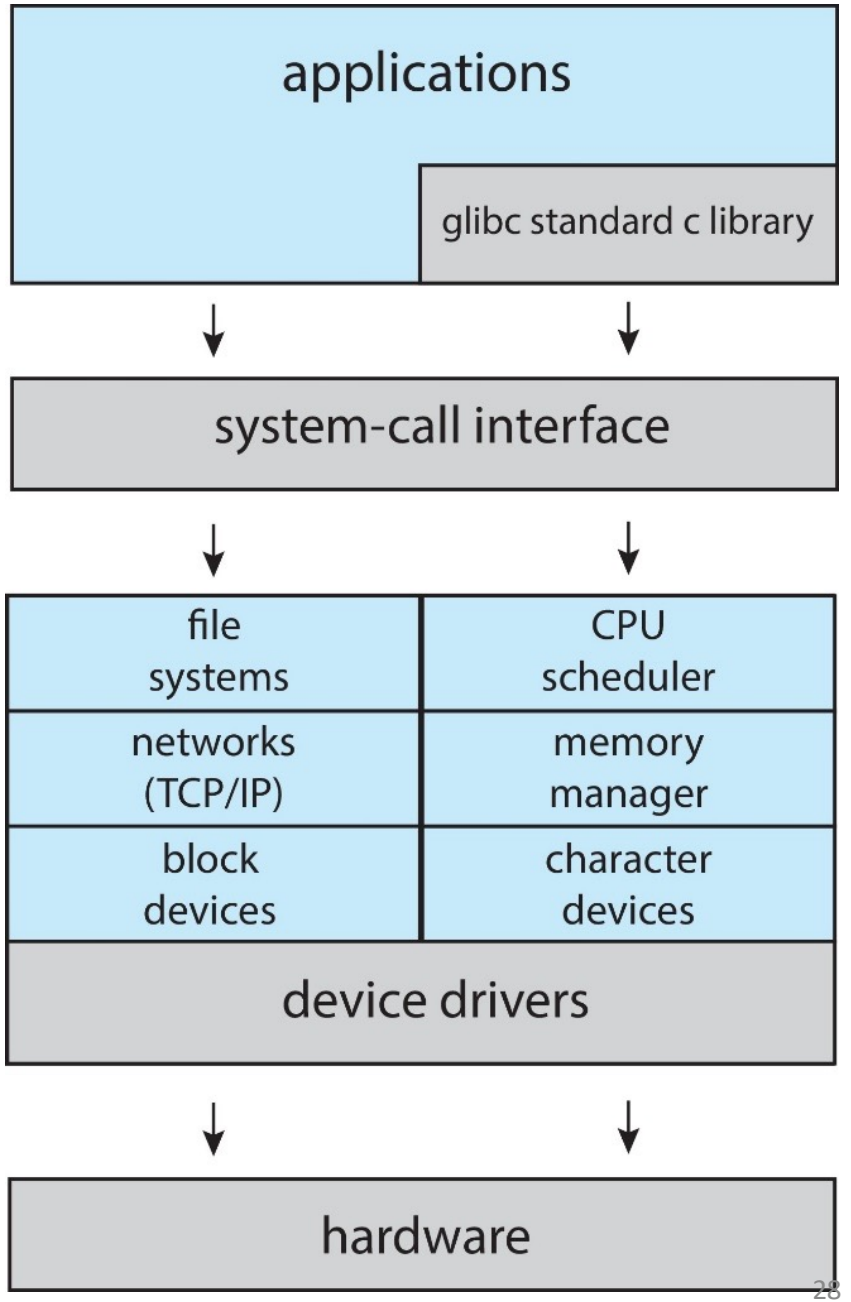
LINUX SYSTEM STRUCTURE

- **Monolithic**: all the functionality of the kernel placed into a single, static binary file that runs in a single address space.

but

- **Modular**: Can load multiple modules, e.g. Device drivers File-system modules Network modules Hardware-specific modules

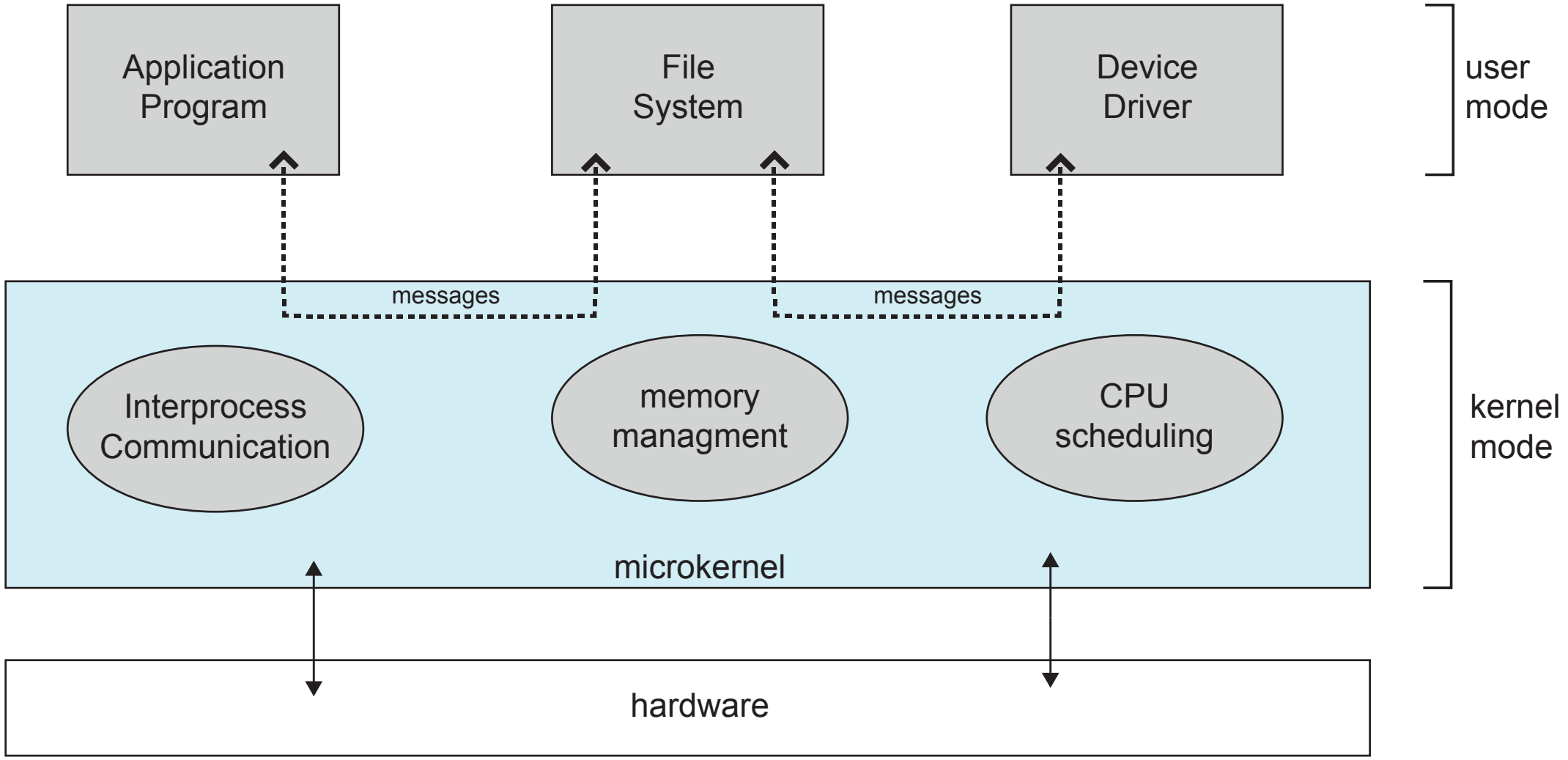
Linux = Monolithic + Modular + Layers



MICROKERNEL SYSTEM STRUCTURE

- Removes all nonessential components from the kernel & implementing them as system & user-level programs. E.g. Mach OS X Kernel (DARWIN) based on Mach
- **Microkernel** provides communication between client programs & various services that are running in user space using *message passing*.
- Benefits:
  - Easier to **extend** microkernel
  - Easier to **port** the operating system to a *different hardware platform* because there's less kernel to re-implement.
  - **Reliable** (less code is running in kernel mode).
  - **Secure** (most services are running as user rather than kernel)

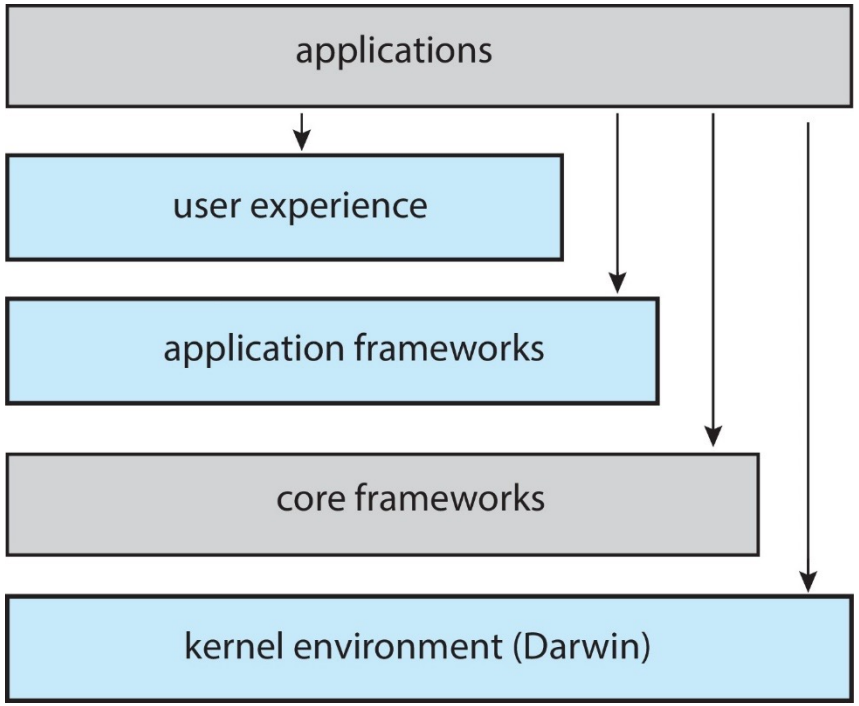
MICROKERNEL SYSTEM STRUCTURE



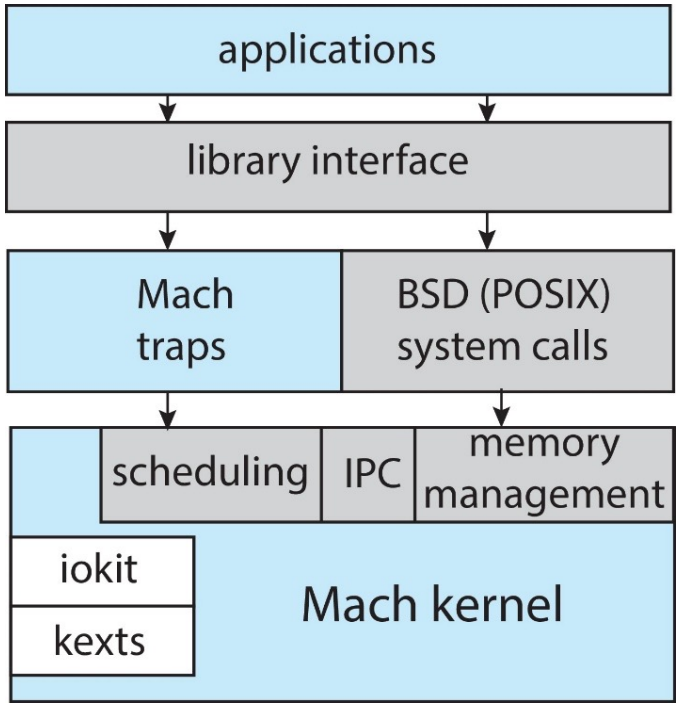
HYBRID SYSTEMS

- 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* providing support for separate subsystems that run as user-mode processes.
- **Apple Mac OS X** *hybrid*, *layered* programming environment

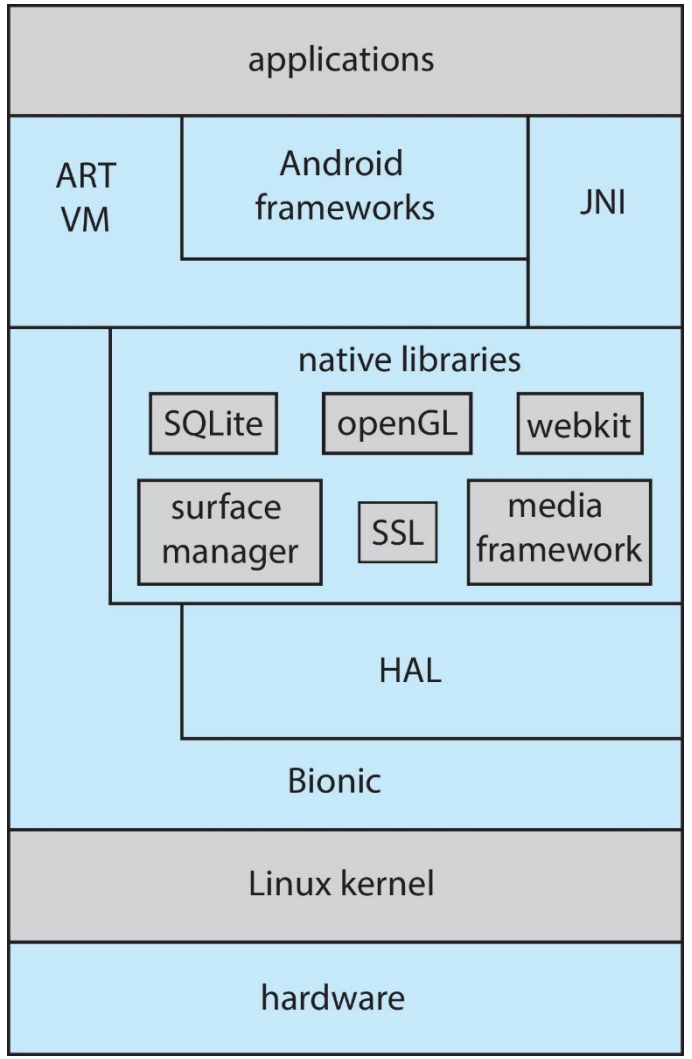
HYBRID SYSTEMS



MacOS



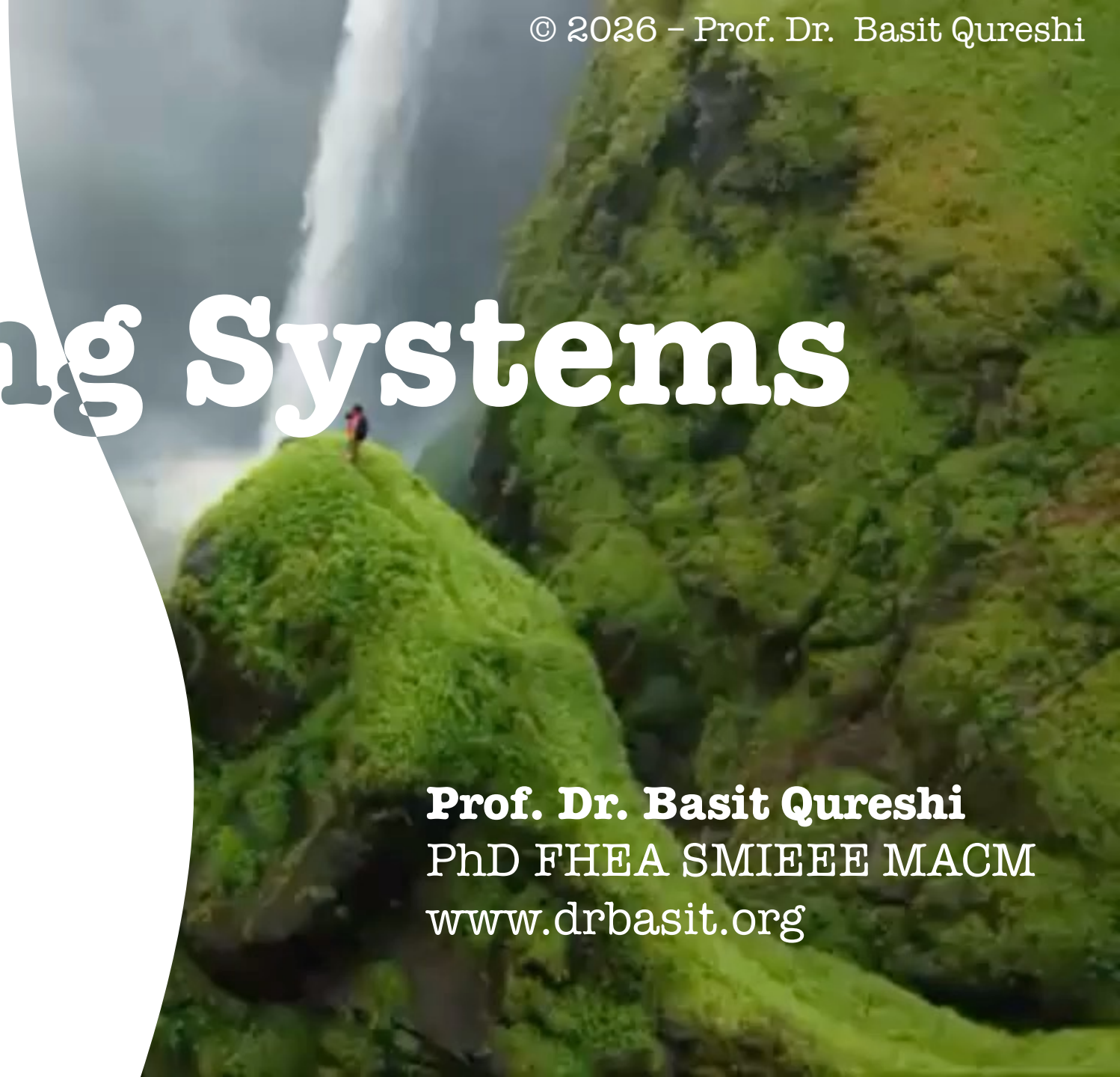
Darwin



Android

Apple built macOS on top of Darwin (Unix based).

CS330 Operating Systems



BUILDING OS

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

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 system is then **running**
- Boot loaders frequently allow various boot states, such as single user mode

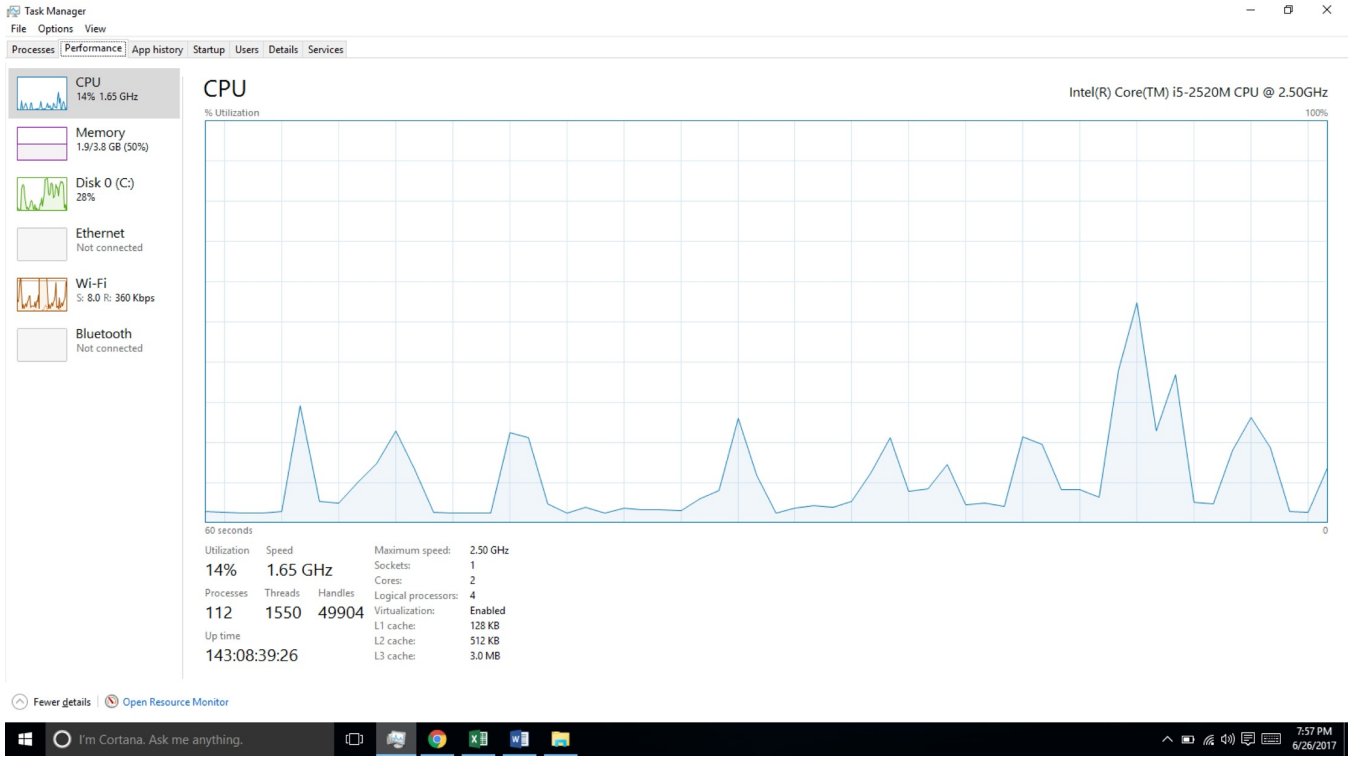
OPERATING-SYSTEM DEBUGGING

- 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

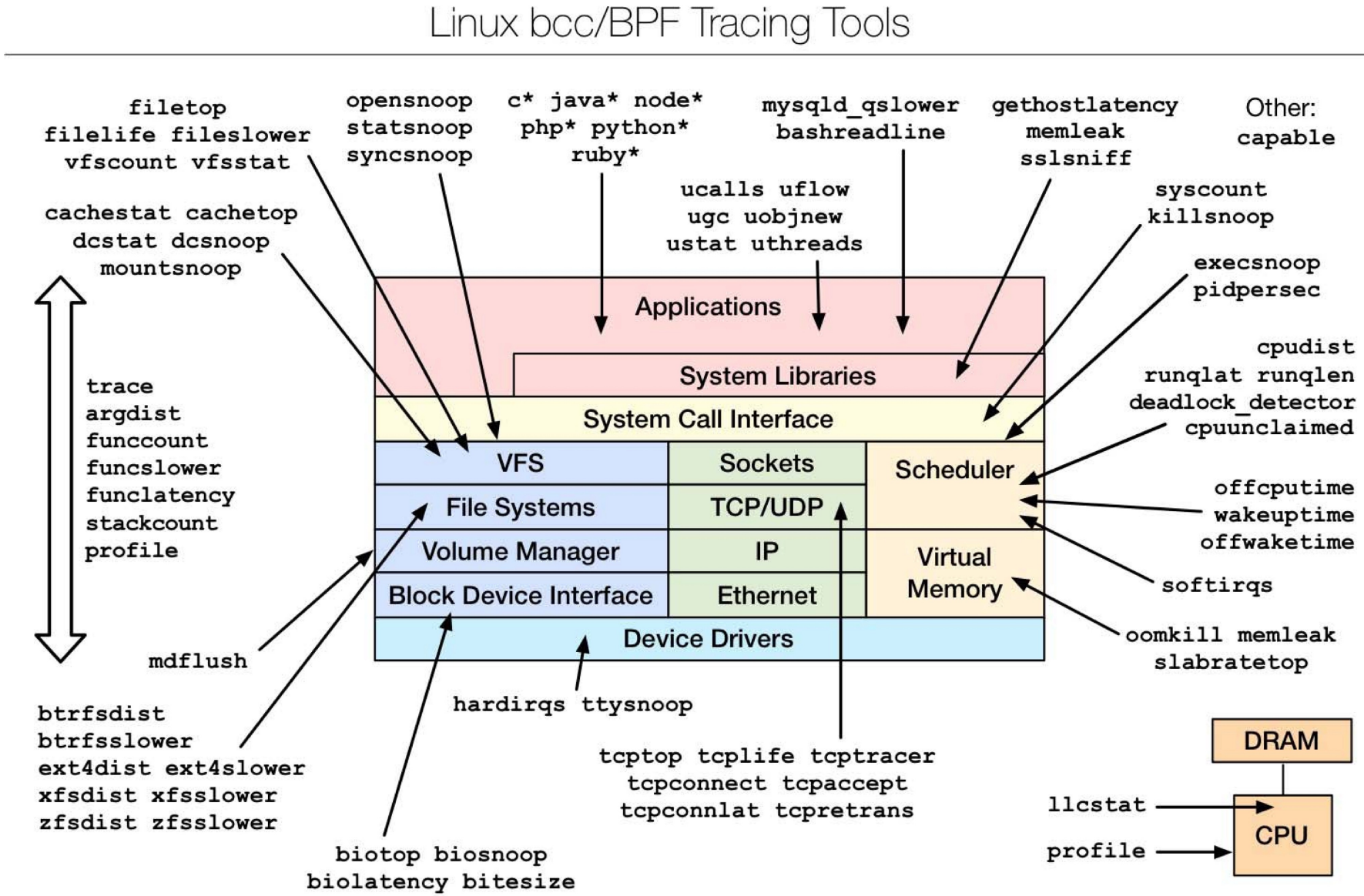
- Collects data for a specific event, such as steps involved in a system call invocation
- Tools include
  - strace – trace system calls invoked by a process
  - gdb – source-level debugger
  - perf – collection of Linux performance tools
  - tcpdump – collects network packets

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



https://github.com/iovisor/bcc#tools 2017

SUMMARY

- OS provides an **environment** for **execution** of *programs* and *services to* ***programs*** *and* ***users***.
- System calls provide the interface between a running program and OS
- General-purpose OS is **very large program.** It needs to be structured. Various approaches to structuring a OS.
- Modern OS are Hybrid and combine multiple approaches to address performance, security, usability needs