

# 240-229: SDA (Operating Systems session)

---

## Lecture 7: Protection, Security, System Performance Evaluation

Associate Professor Dr. Sangsuree Vasupongayya

sangsuree.v@psu.ac.th

Department of Computer Engineering

Prince of Songkla University

# Outline

---

- Protection
  - Principles of Protection
  - Domain of Protection
  - Access Matrix
  - Implementation of Access Matrix
- Security
  - The security problem
  - Security Violations
  - Cryptography
- System Performance Evaluation

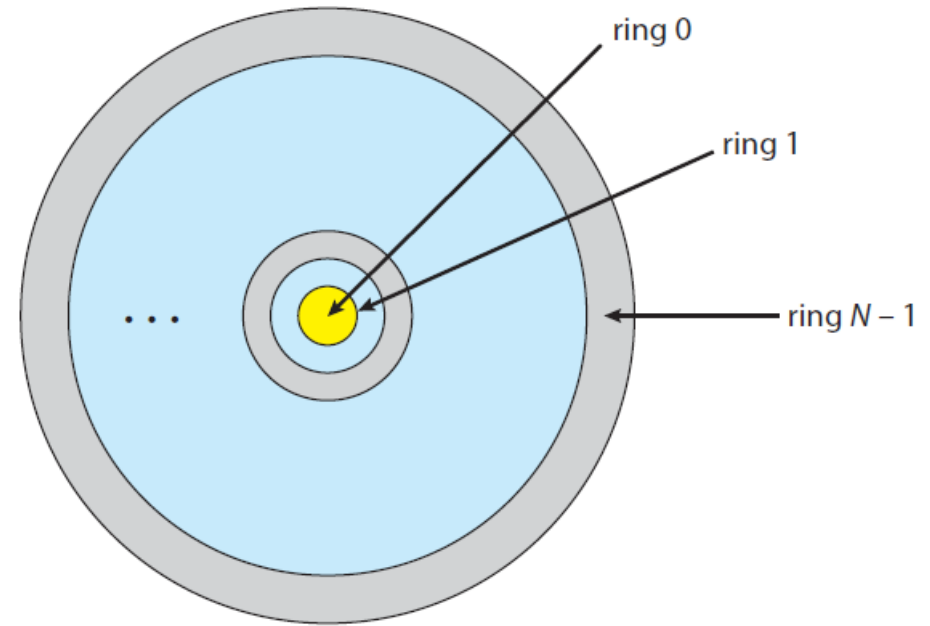
# Principles of Protection

---

- ❑ Principle of least privilege
  - programs, users, and even systems be given just enough privileges to perform their tasks
- ❑ Principle of least privilege gives the system a chance to mitigate the attack
  - if malicious code cannot obtain root privileges, there is a chance that adequately defined permissions may block all, or at least some, of the damaging operations
- ❑ Compartmentalization is the process of protecting each individual system component through the use of specific permissions and access restrictions

# Protection Ring

- ❑ Intel architectures places user mode code in ring 3 and kernel mode code in ring 0
- ❑ Intel defined an additional ring (-1) to allow for **hypervisors** or virtual machine managers, which create and run virtual machines
- ❑ Hypervisors have more capabilities than the kernels of the guest operating systems

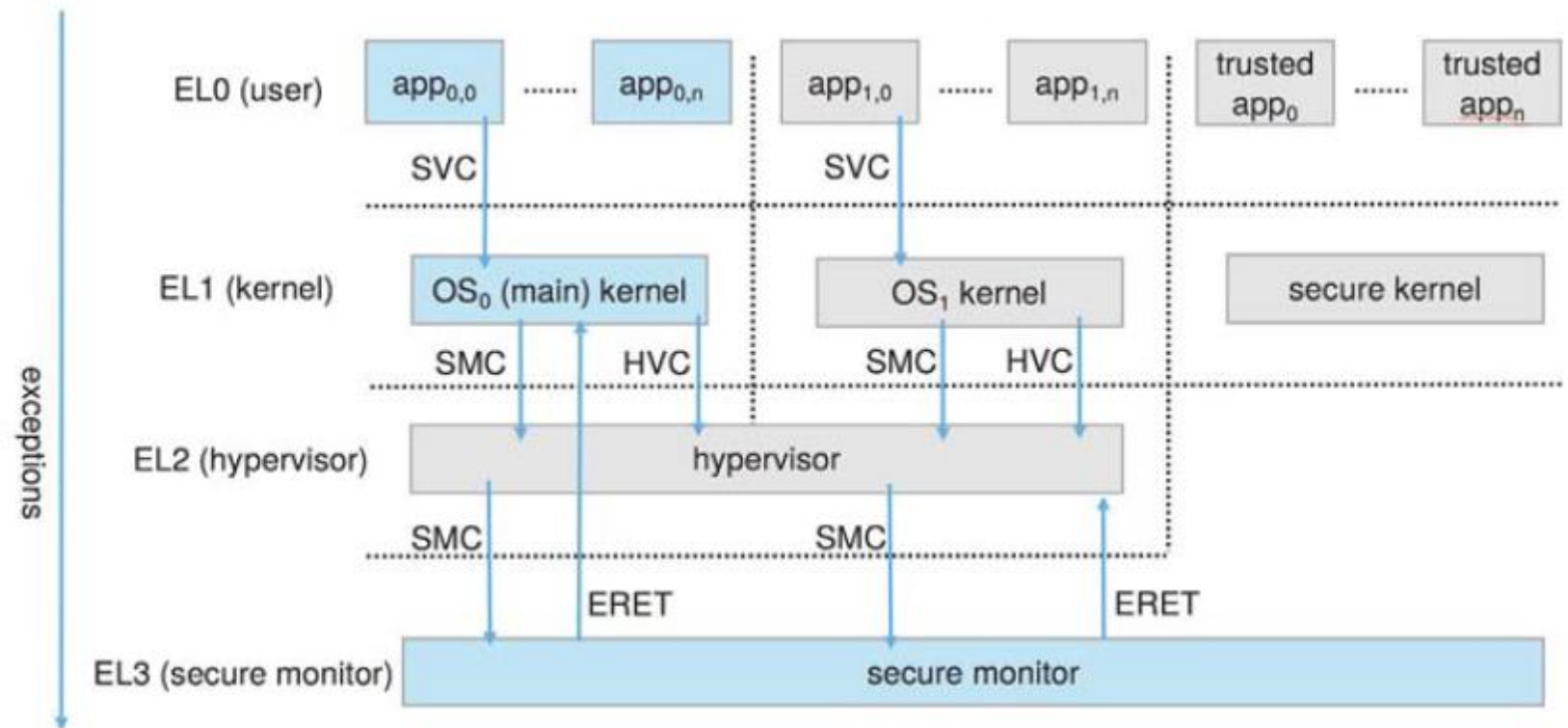


**privilege separation**

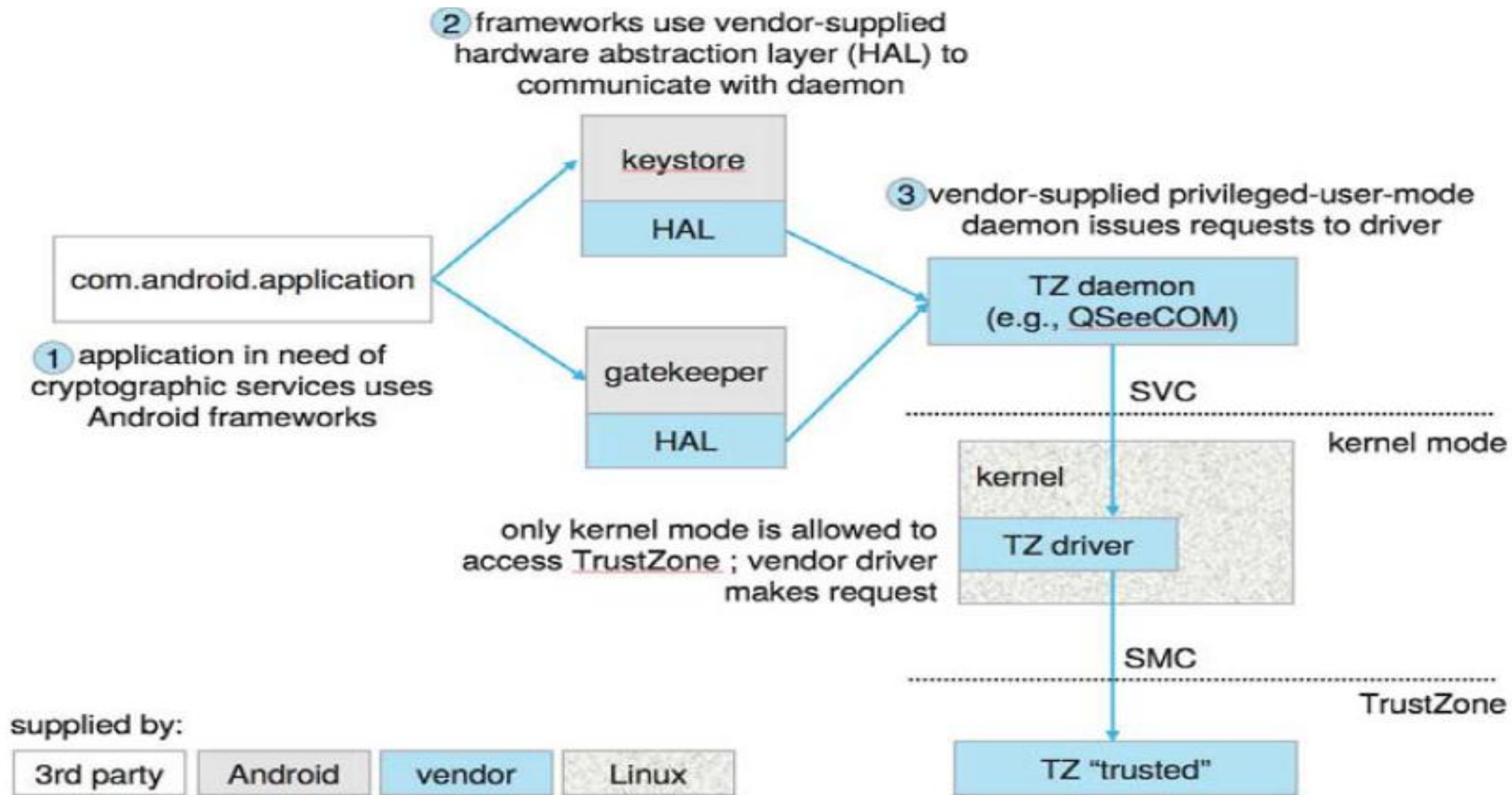
# TrustZone in ARMv7

- TrustZone (TZ) provides an additional ring
  - has exclusive access to hardware-backed cryptographic features
- Even the kernel itself has no access to the on-chip key, and it can only request encryption and decryption services from the TrustZone environment
- if the kernel is compromised, an attacker can't simply retrieve the key from kernel memory

EL0 – user mode;  
EL1 is kernel mode  
EL2 – hypervisors;  
EL3 is for secure monitor  
(TrustZone layer)



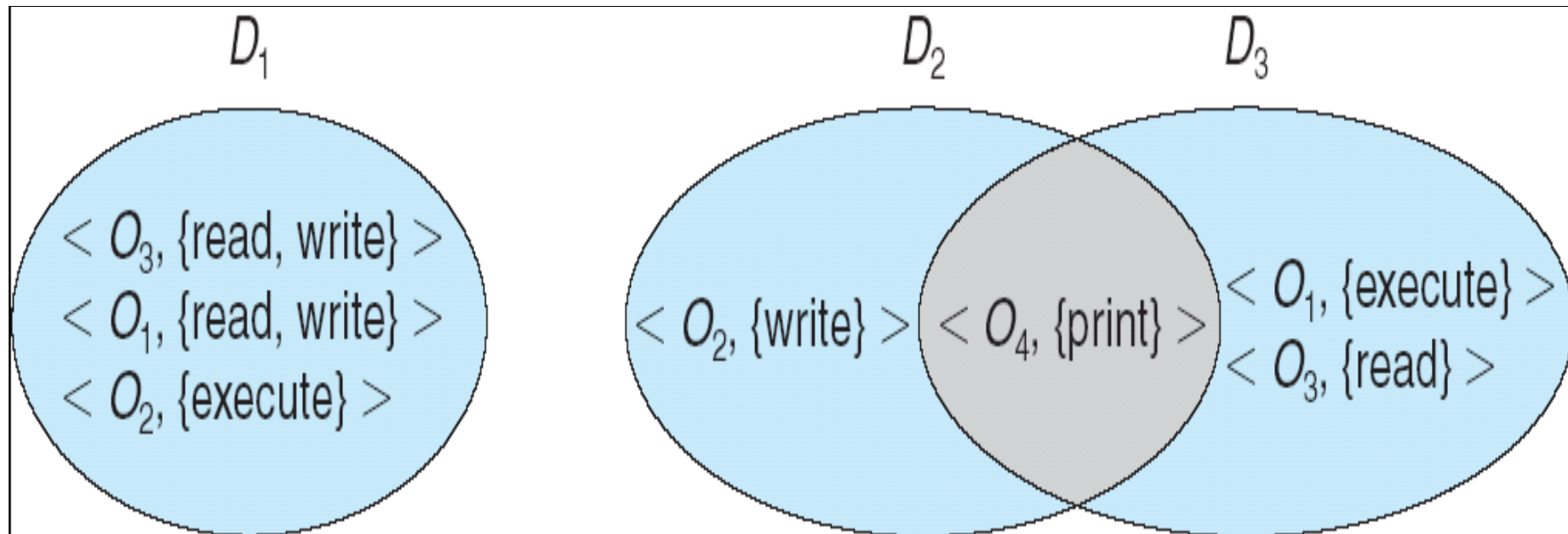
# Android uses of TrustZone



# Domain Structure

## Need-to-know principle

- Access-right =  $\langle \text{object-name}, \text{rights-set} \rangle$   
where *rights-set* is a subset of all valid operations that can be performed on the object.
- Domain = set of access-rights



# Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- $Access(i, j)$  is the set of operations that a process executing in  $Domain_i$  can invoke on  $Object_j$

domain \ object	$F_1$	$F_2$	$F_3$	printer
$D_1$	read		read	
$D_2$				print
$D_3$		read	execute	
$D_4$	read write		read write	

# Implementation of Access Matrix

---

- Each column = Access-control list for one object  
Defines who can perform what operation.

Domain 1 = Read, Write  
Domain 2 = Read  
Domain 3 = Read  
⋮

- Each Row = Capability List (like a key)  
Fore each domain, what operations allowed on what objects.

Object 1 – Read  
Object 4 – Read, Write, Execute  
Object 5 – Read, Write, Delete, Copy

## Access Matrix of Figure A With Domains as Objects

domain \ object	$F_1$	$F_2$	$F_3$	laser printer	$D_1$	$D_2$	$D_3$	$D_4$
$D_1$	read		read			switch		
$D_2$				print			switch	switch
$D_3$		read	execute					
$D_4$	read write		read write		switch			

# Access Matrix with *Copy* Rights

object domain	$F_1$	$F_2$	$F_3$
$D_1$	execute		write*
$D_2$	execute	read*	execute
$D_3$	execute		

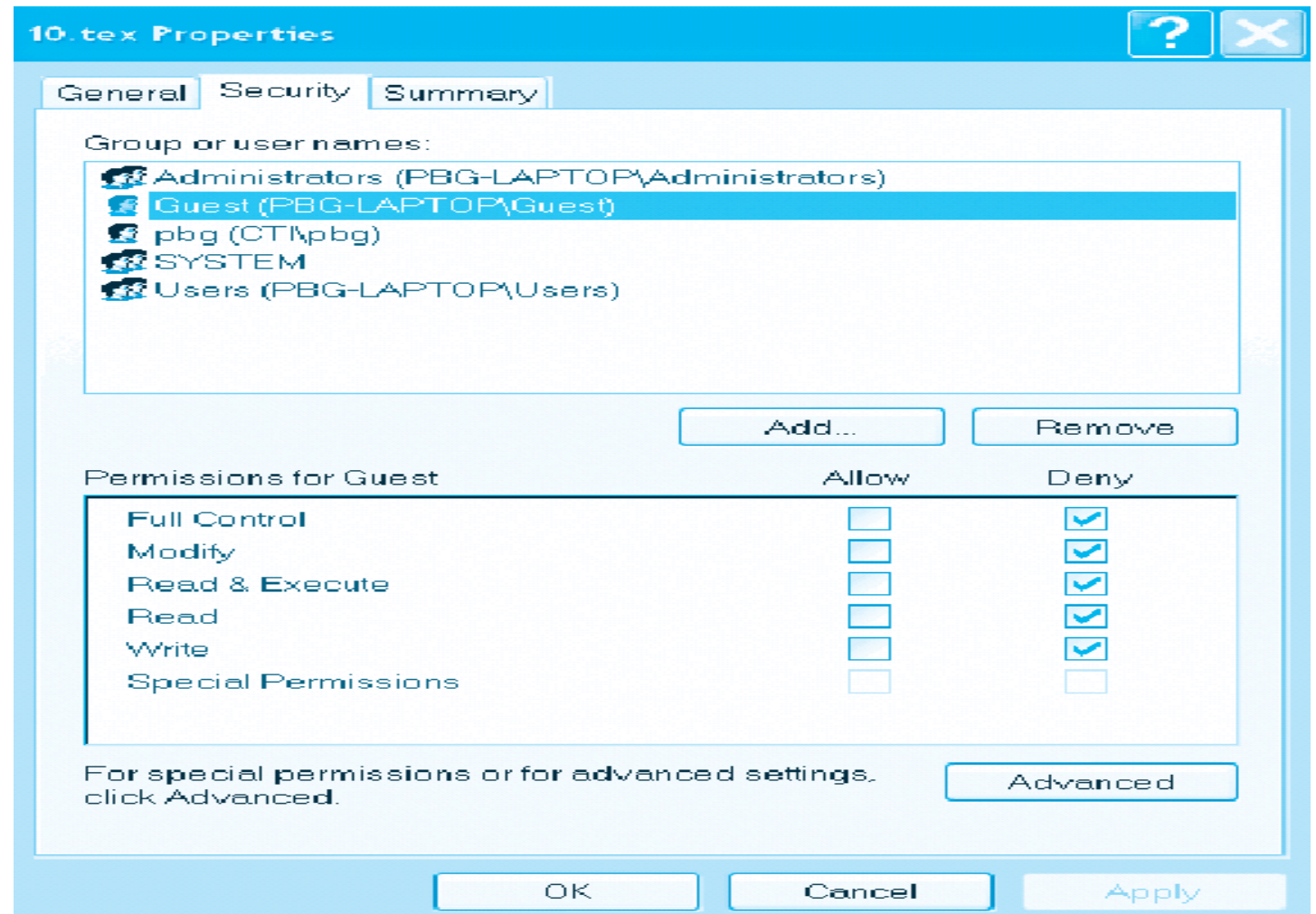
(a)

object domain	$F_1$	$F_2$	$F_3$
$D_1$	execute		write*
$D_2$	execute	read*	execute
$D_3$	execute	read	

(b)

# Access control on Files

- ❑ File owner/creator should be able to control:
  - what can be done
  - by whom
- ❑ Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**



# Access Lists and Groups

---

- ❑ Mode of access: read, write, execute
- ❑ Three classes of users

		RWX
a) <b>owner access</b>	7	$\Rightarrow$ 1 1 1
		RWX
b) <b>group access</b>	6	$\Rightarrow$ 1 1 0
		RWX
c) <b>public access</b>	1	$\Rightarrow$ 0 0 1

- ❑ Ask manager to create a group (unique name), say G, and add some users to the group.
- ❑ For a particular file (say *game*) or subdirectory, define an appropriate access.

# A Sample UNIX Directory Listing

---

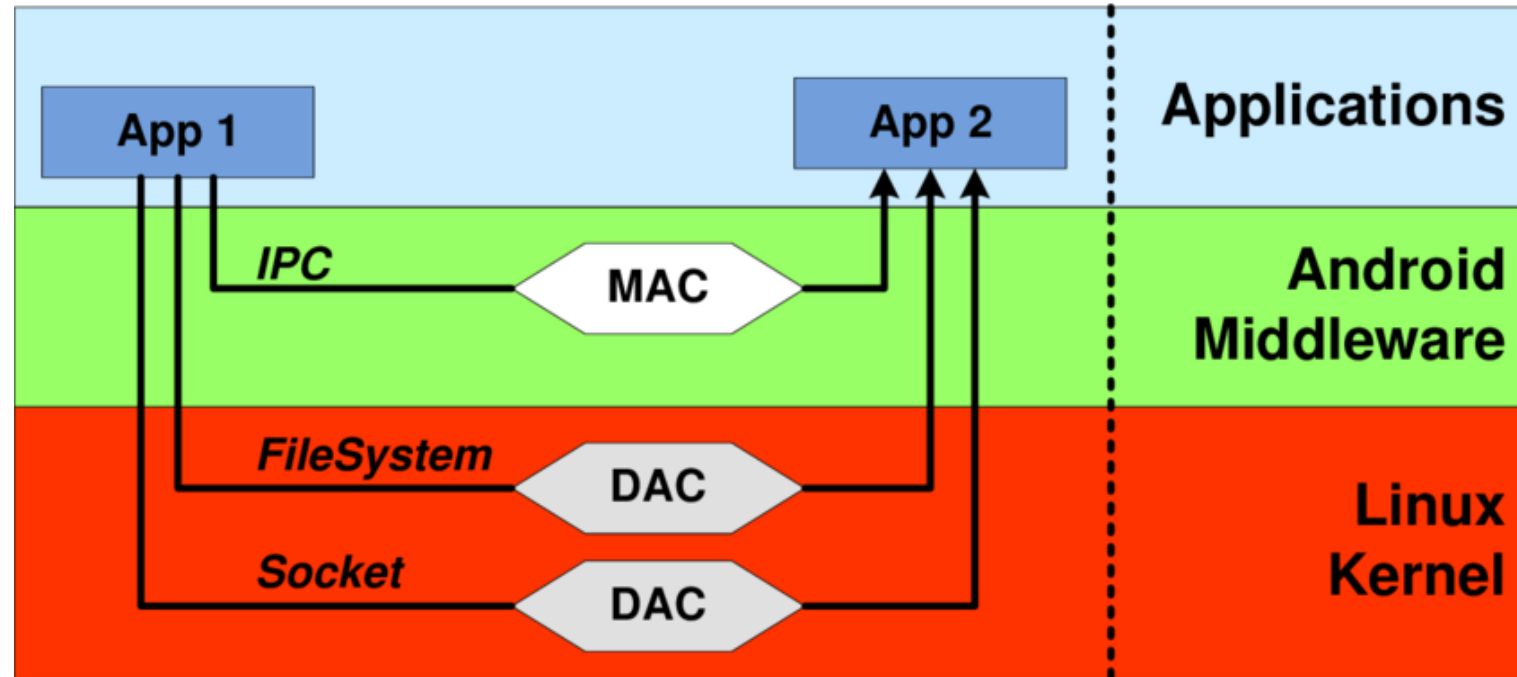
-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5 pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx-----	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

# Principles of Protection

---

- ❑ Principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
- ❑ Solution - Access control
  - Discretionary access control (DAC)
    - ❑ Based on the identity of the requestor
  - Mandatory access control (MAC)
    - ❑ Comparing security labels with the security clearances
  - Role-based access control (RBAC)
    - ❑ Based on the roles that users have within the system

# Android security enforcement



- ❑ IPC Reference Monitor follows Mandatory Access Control (MAC) access control type
- ❑ Apps run in low-privileged Application Sandboxes
- ❑ an application must request the corresponding permissions in the file `AndroidManifest.xml`

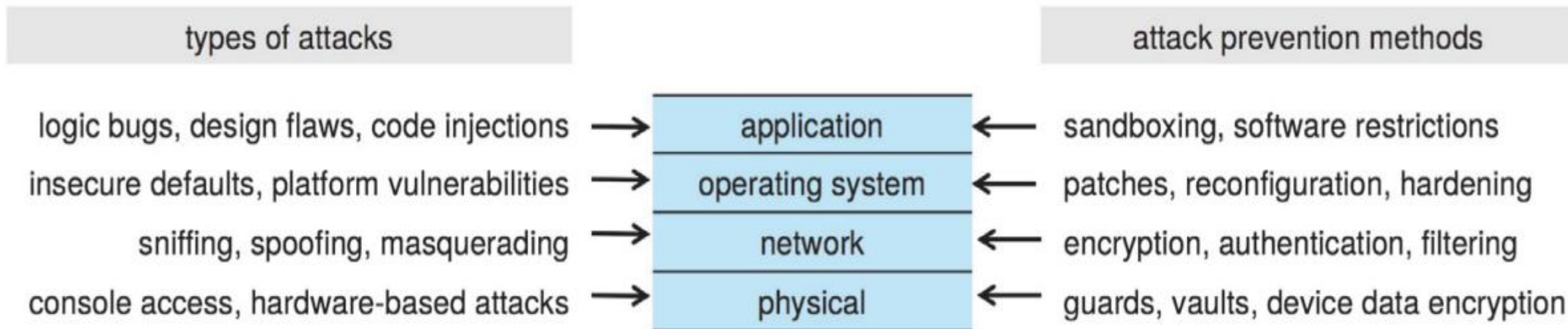
# Security

---

- ❑ Security must consider external environment of the system, and protect the system resources
- ❑ **Threat** is potential security violation
- ❑ **Attack** is an attempt to breach security
  - Attack can be accidental or malicious
  - Easier to protect against accidental than malicious misuse
- ❑ **Principles**
  - Confidentiality: Data confidentiality & Privacy
  - Integrity: Data integrity & System integrity
  - Availability: Theft of service & Denial of service

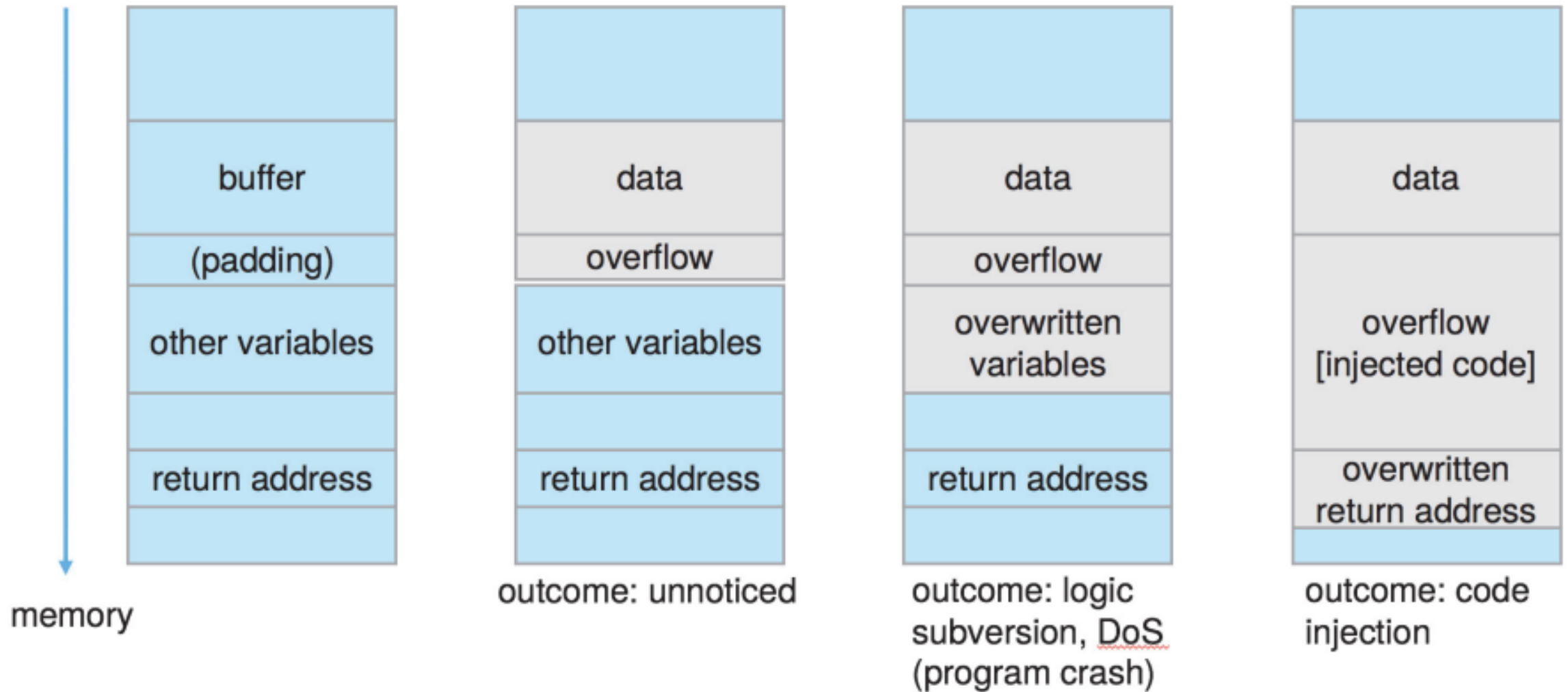
# Security Measure Levels

- ❑ Security must occur at four levels to be effective:
  - Physical
  - Human: Avoid social engineering, phishing, dumpster diving
  - Operating System
  - Network
- ❑ Security is as weak as the weakest chain



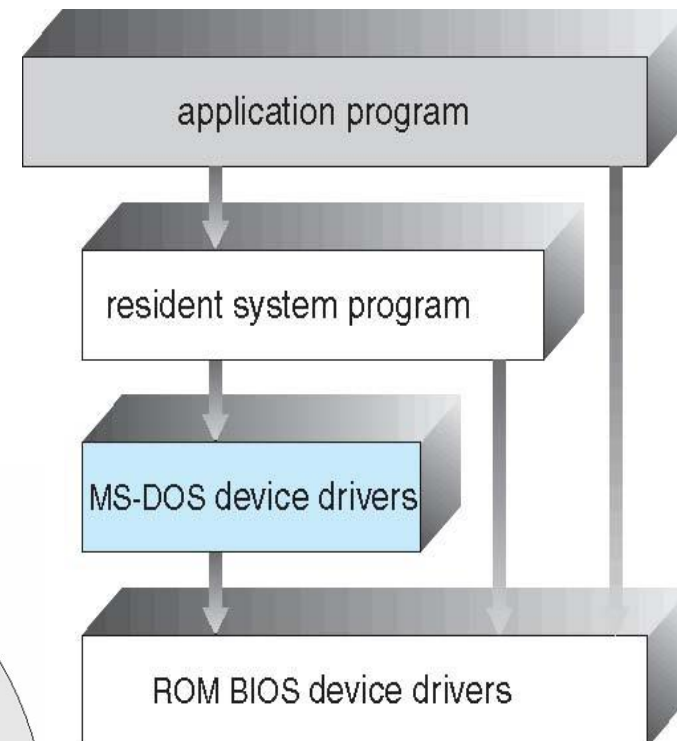
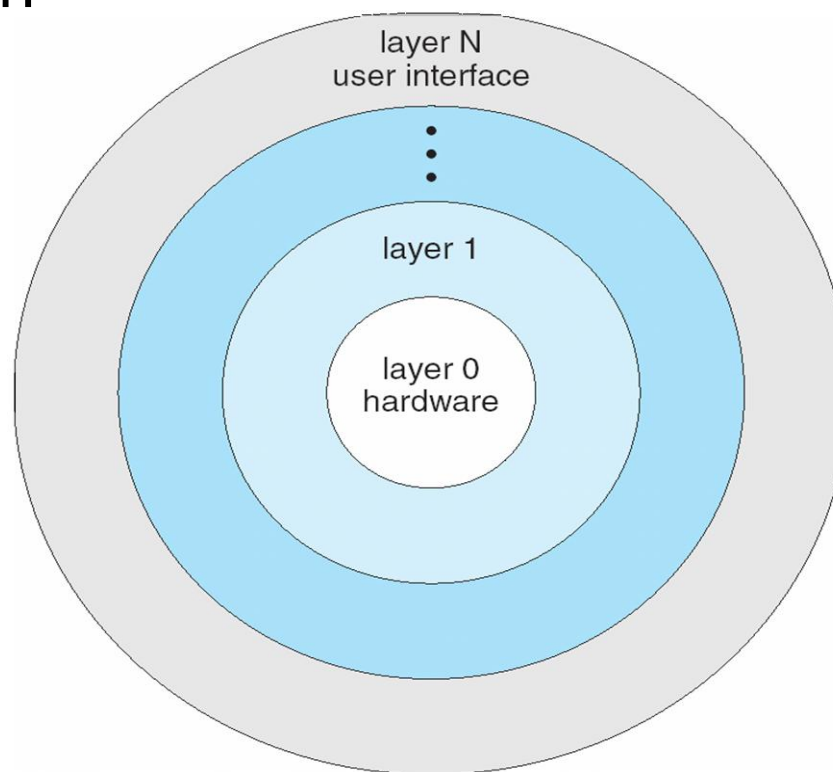
# Buffer overflow attack

---



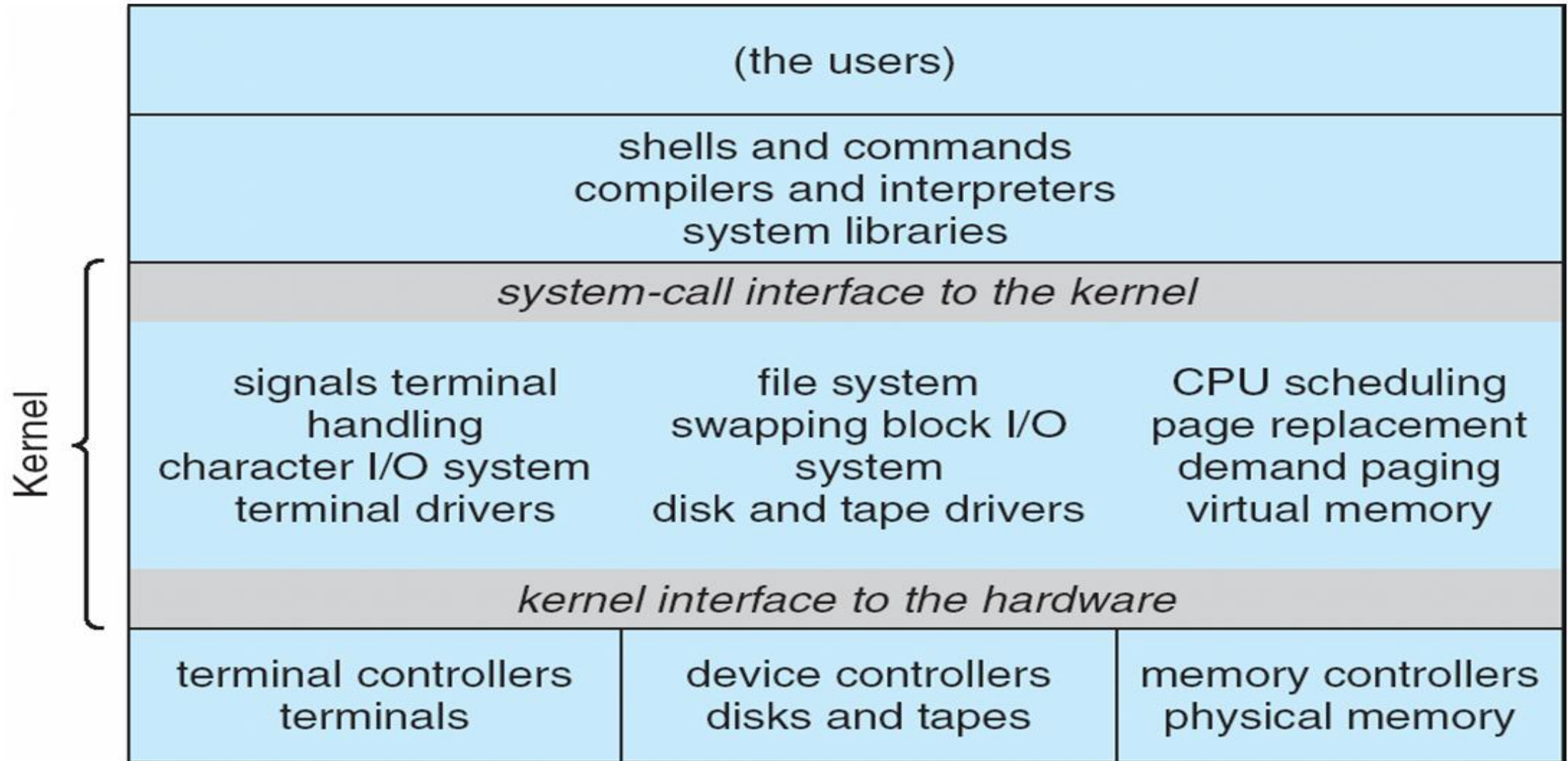
# Operating system structure

- Simple structure
  - MS-DOS – written to provide the most functionality in the least space
- Layered Approach
  - The operating system is divided into a number of layers (levels), each built on top of lower layers.

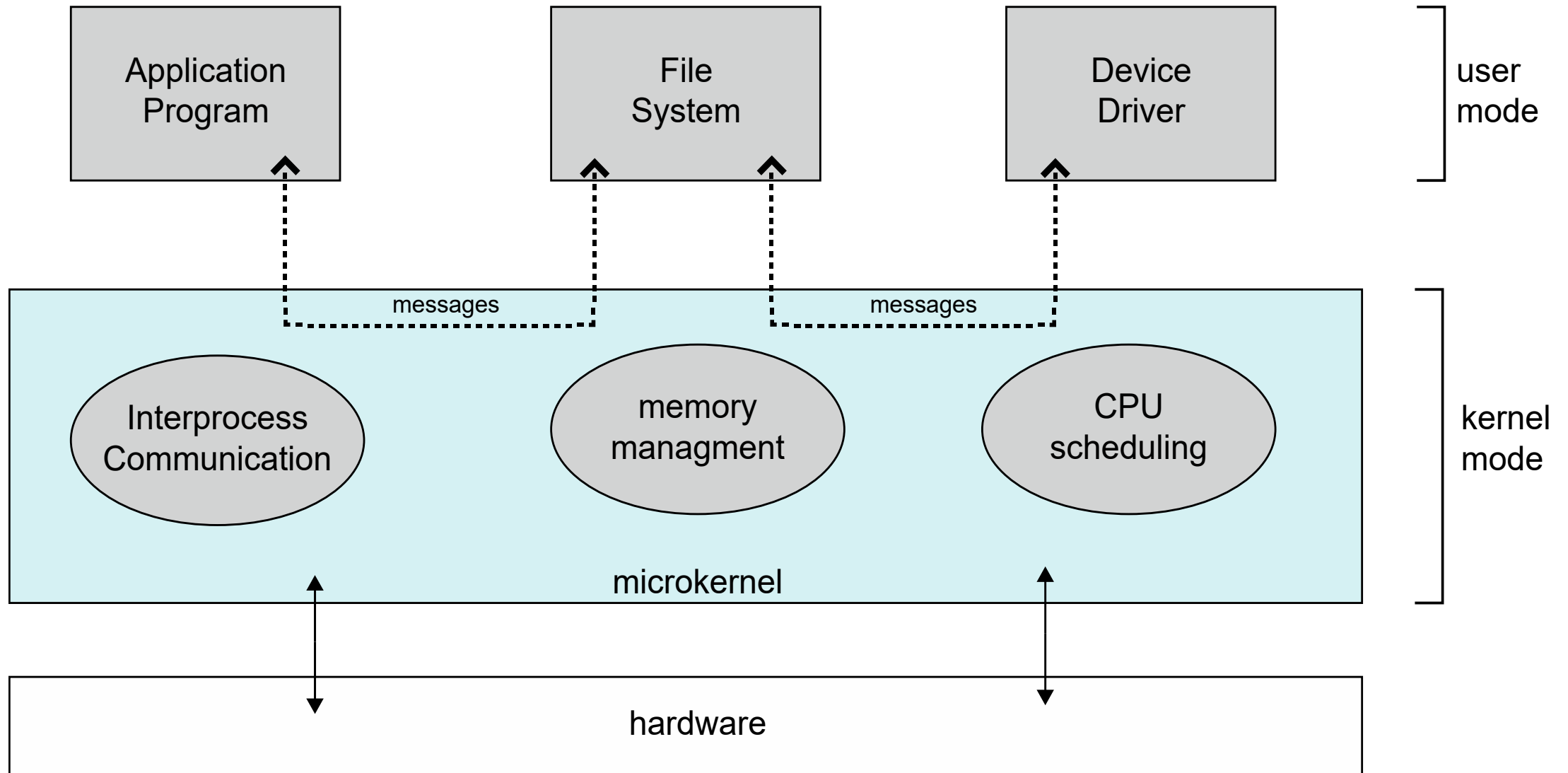


# Traditional UNIX System Structure

Beyond simple but not fully layered



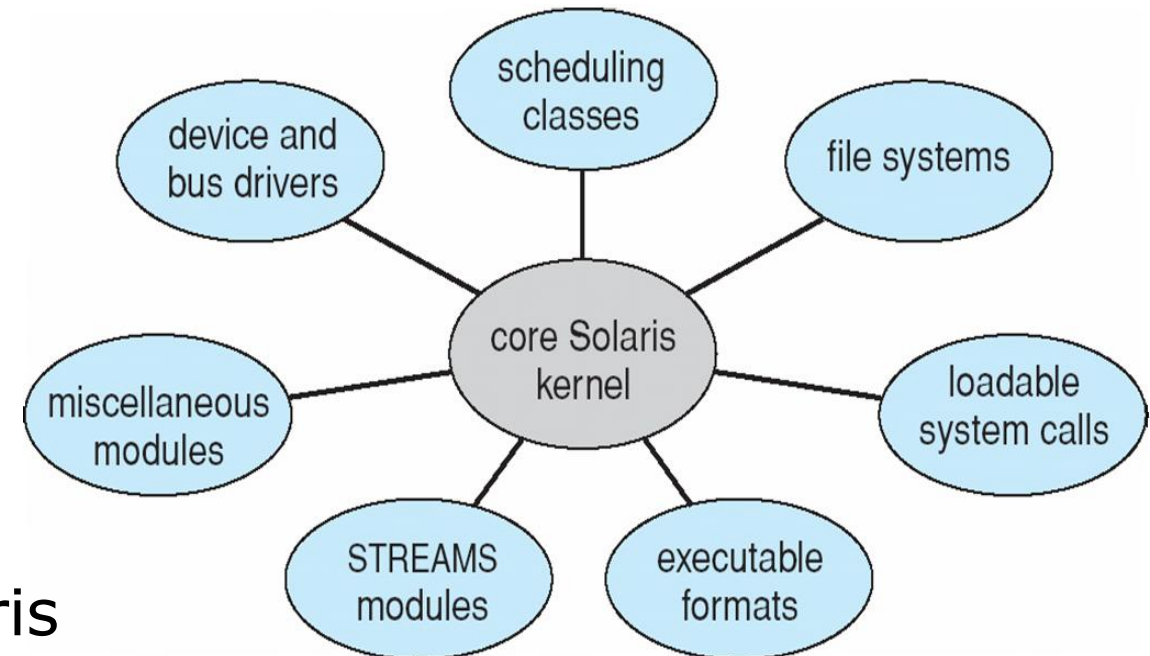
# Microkernel System Structure



Moves as much from the kernel into user space

# Modular Approach

- ❑ 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
- ❑ Overall, similar to layers but with more flexible
  - Linux, Solaris, etc



E.g., Solaris

# Hybrid Systems

---

- Most modern OS not one pure model
- Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris
    - Monolithic + modular for dynamic loading
  - Windows
    - mostly monolithic + microkernel for different subsystem

***Examples shown next are***

- ***Android, iOS, Mac OS X***

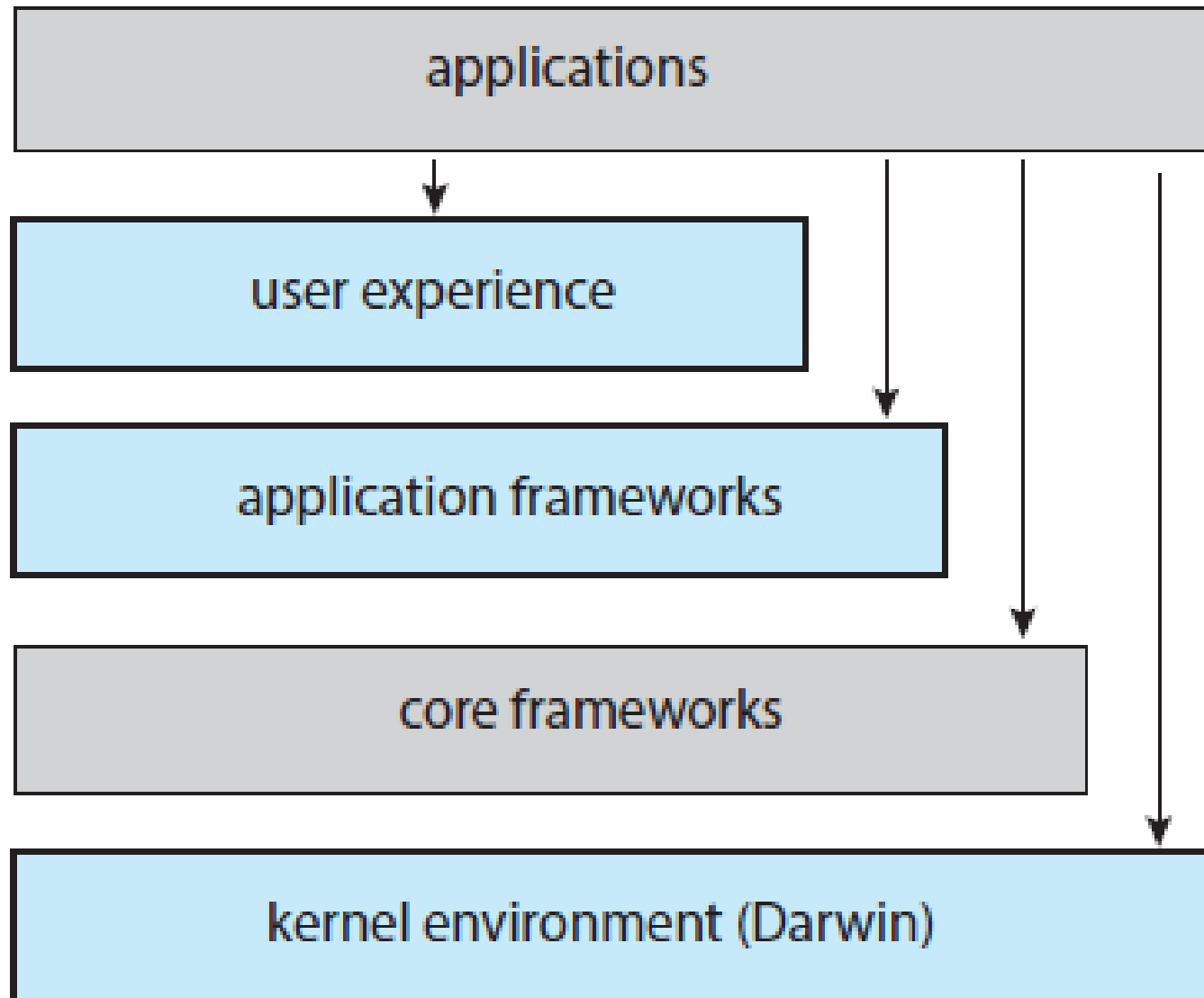
# Architecture of Apple's macOS and iOS

---

- ❑ Apple's macOS is designed to run primarily on desktop and laptop computer systems,
- ❑ iOS is designed for the iPhone and iPad
- ❑ Consisting of 4 layers
  - User experiences: the software interface that allows users to interact with the computing devices
  - Application framework: the Cocoa and Cocoa Touch frameworks, which provide an API for the Objective-C and Swift programming languages
  - Core framework: defines frameworks that support graphics and media including, Quicktime and OpenGL
  - Kernel environment: a.k.a Darwin, includes the Mach microkernel and the BSD UNIX kernel

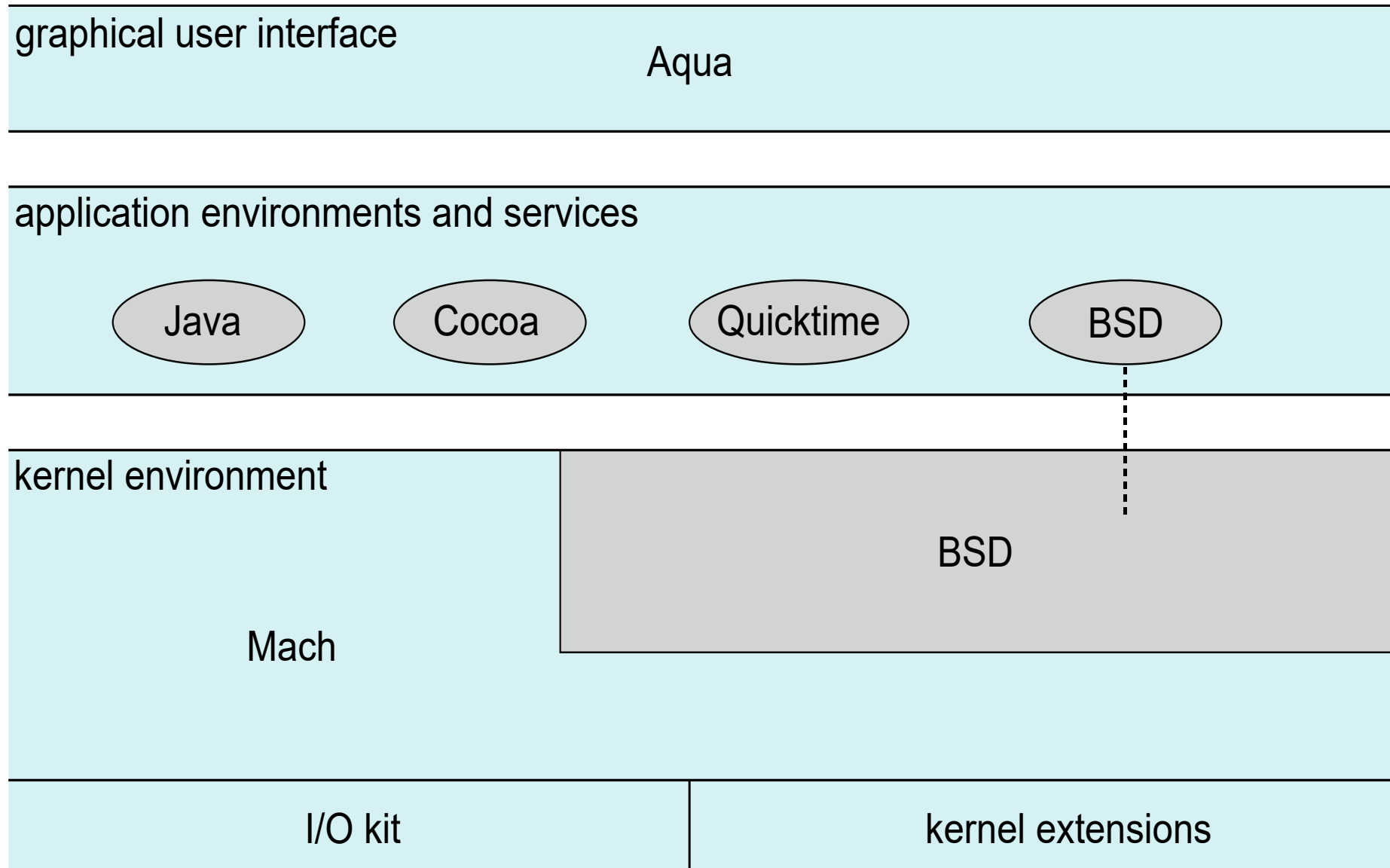
# Architecture of Apple's macOS and iOS

---



# 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

Cocoa Touch

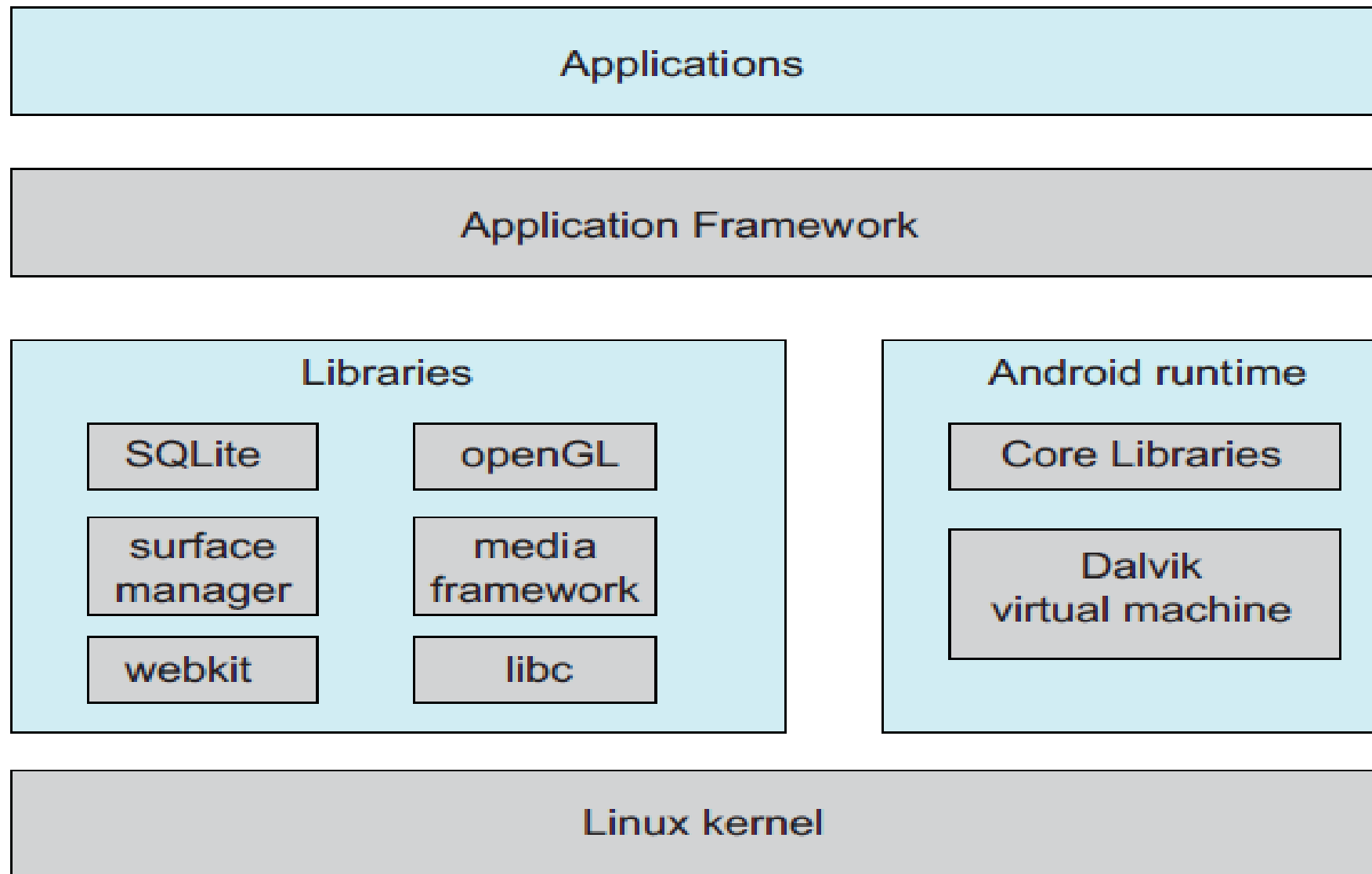
Media Services

Core Services

Core OS

# Android

---



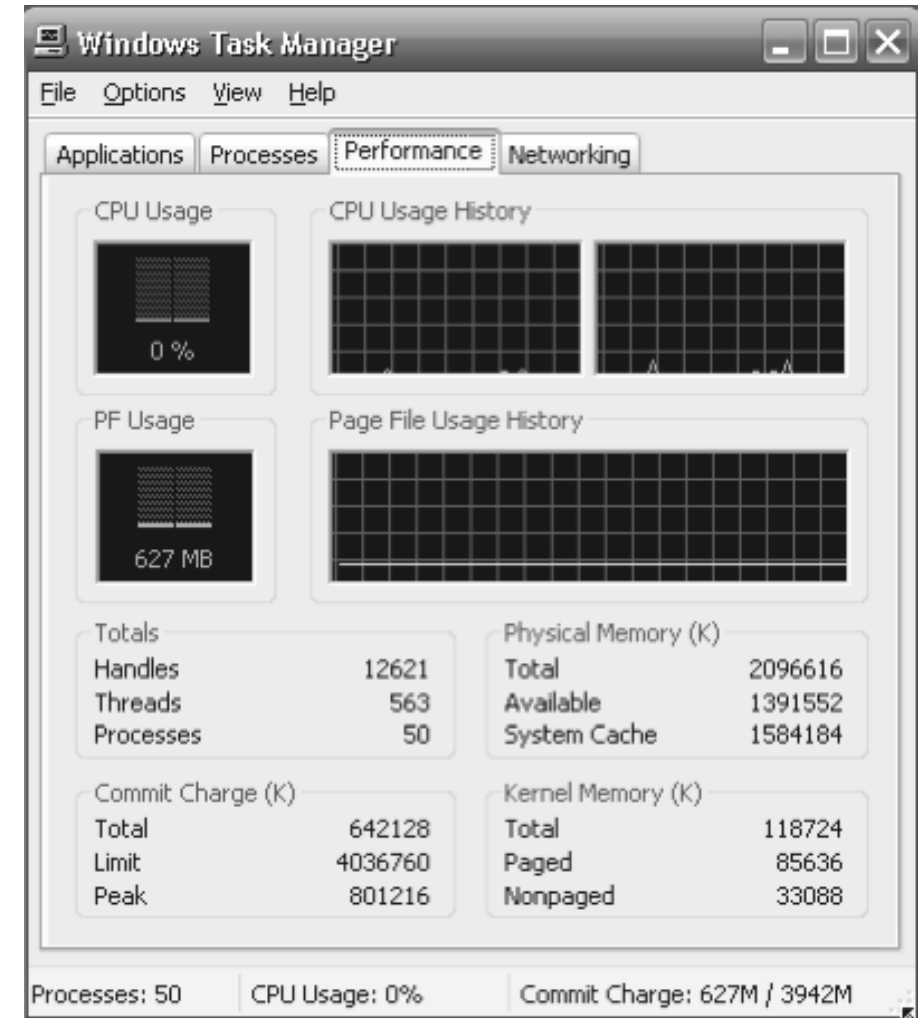
# System Performance

---

- ❑ finding and fixing errors, or **bugs**
- ❑ 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

# 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



# 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
- ❑ Example of following XEventsQueued system call move from libc library to kernel and back

```
# ./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
```

# Amdahl's Law

---

- Formula identifies potential performance gain

$$\text{Speedup} \leq 1/(S + ((1-S)/N))$$

Where S = portion of the application that must be performed serially on a system with N processing cores

- As N approaches infinity, the speedup converges to 1/S
- Does not take into account the hardware performance enhancements

Example: An application is 75% parallel and 25% serial. If we run this application on a system with two processing cores, we can get a speed up of 1.6 times.

$$\begin{aligned}\text{Speedup} &= 1/(0.25 + (0.75/2)) \\ &= 1/(0.25 + 0.375) \\ &= 1/0.625 \\ &= 1.6\end{aligned}$$

# I/O Performance

---

Improve the I/O performance

- ☐ Reduce number of context switches
- ☐ Reduce data copying
- ☐ Reduce interrupts by using large transfers, smart controllers, polling
- ☐ Use DMA
- ☐ Use smarter hardware devices
- ☐ Balance CPU, memory, bus, and I/O performance for highest throughput
- ☐ Move user-mode processes / daemons to kernel threads

# Storage Performance

---

- Best method depends on file access type
  - Contiguous great for sequential and random
  - Linked good for sequential, not random
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead
- Declare access type at creation -> select either contiguous or linked
- Adding instructions to the execution path to save one disk I/O is reasonable

# Evaluation: Criteria

---

- ☐ Relationship between the criteria and measurements

E.g., two measurements

Set A: 1, 1, 1, 1, 1, 1, 1, 1, 8

Set B: 2, 2, 2, 2, 2, 2, 2, 2, 2

- Are they the same?
- If the average value is used in this case then
  - ☐ Both data sets produce the same performance
- If the max value is used in this case then
  - ☐ They are different
- ☐ **KEY: Select a good measurements for your criteria**

# Evaluation Method

---

- Deterministic modeling:
  - Evaluating the performance of the algorithm for a selected workload
  - KEY: the goodness of workloads
  
- Queueing models:
  - E.g.,  
when the distribution of CPU and I/O bursts and the arrival-time distribution are available, they can be used to construct the system queueing model
  - KEY: determining a suitable model

# Evaluation Method (Cont.)

---

## □ Simulations

- programming a model of the computer system
- the input can be a distribution-driven workload or a real workload trace
- KEY: Availability of workload & model

## □ Implementation:

- test on the real system
- KEY: Cost and time