Operating Systems Structures

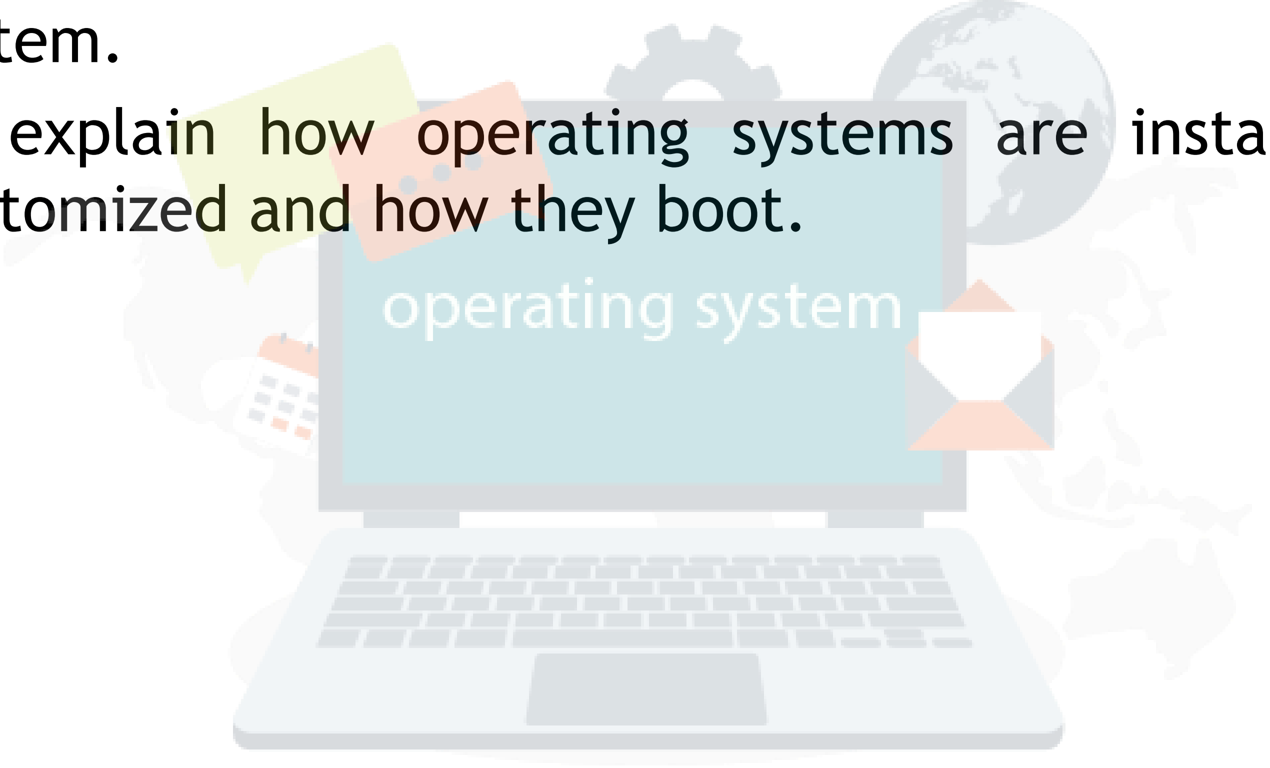
By Anthony Khajira

Strathmore
UNIVERSITY

Class Objectives

At the end of this lecture, each student should be able :

- To discuss the various ways of structuring an operating system.
- To explain how operating systems are installed and customized and how they boot.



Overview

- A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily.
- A common approach is to partition the task into small components, or modules, rather than have one monolithic system.
- Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.
- In this class, we discuss how these components are interconnected and melded into a kernel.

Operating-System Structures

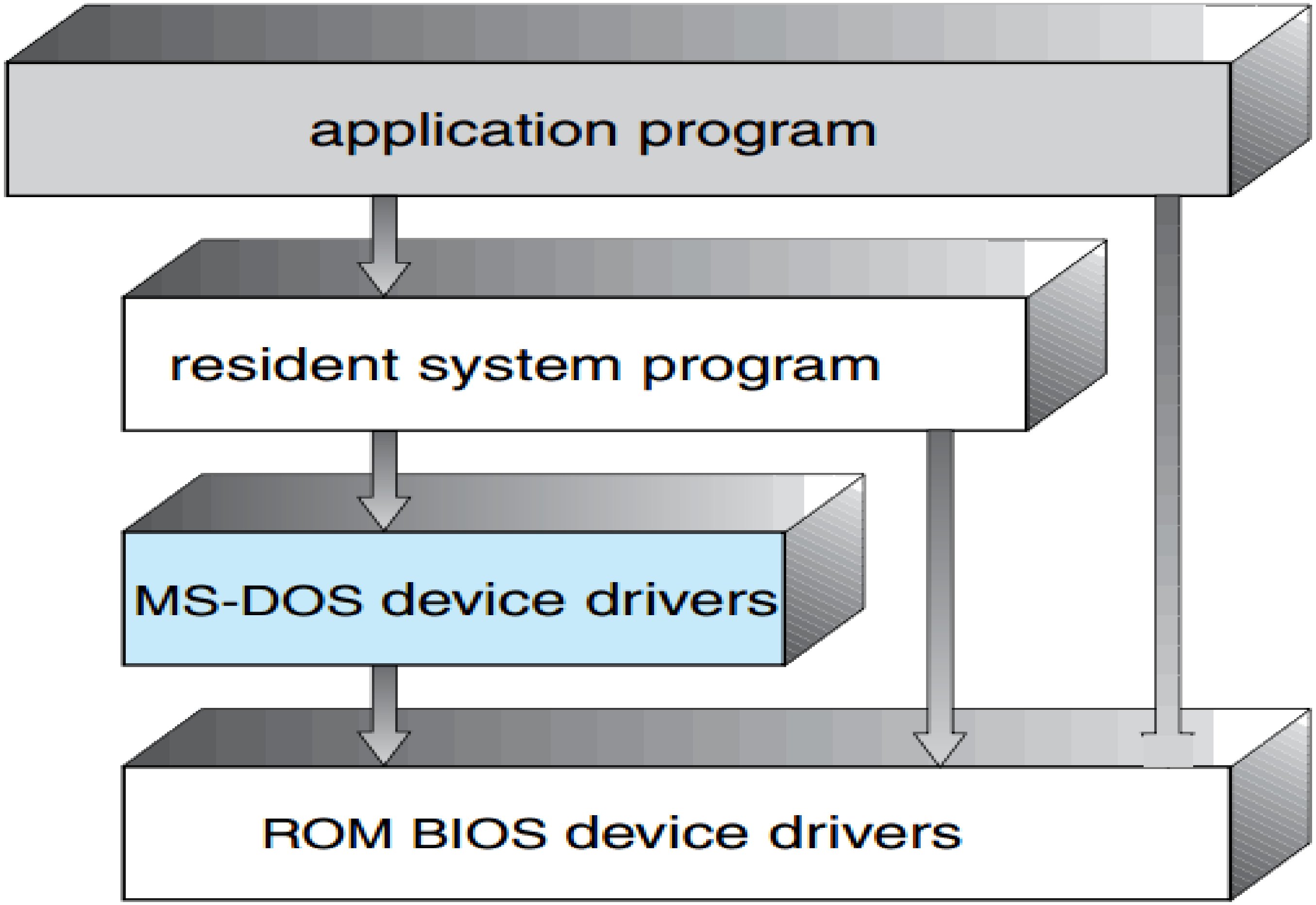
- There are different approaches to structure operating systems :
  1. Simple Structure
  2. Layered Approach
  3. Microkernels
  4. Modules (Modular approach)
  5. Hybrid Systems

1. Simple Structure

Simple Structure - Overview

- Many operating systems do not have well-defined structures.
- Frequently, such systems started as ***small***, ***simple***, ***and limited systems*** and then grew beyond their original scope.
- MS-DOS is an example of such a system.
- It was originally designed and implemented by a few people who had no idea that it would become so popular.
- It was written to provide the most functionality in the least space, so it was not carefully divided into modules.

MS DOS Structure



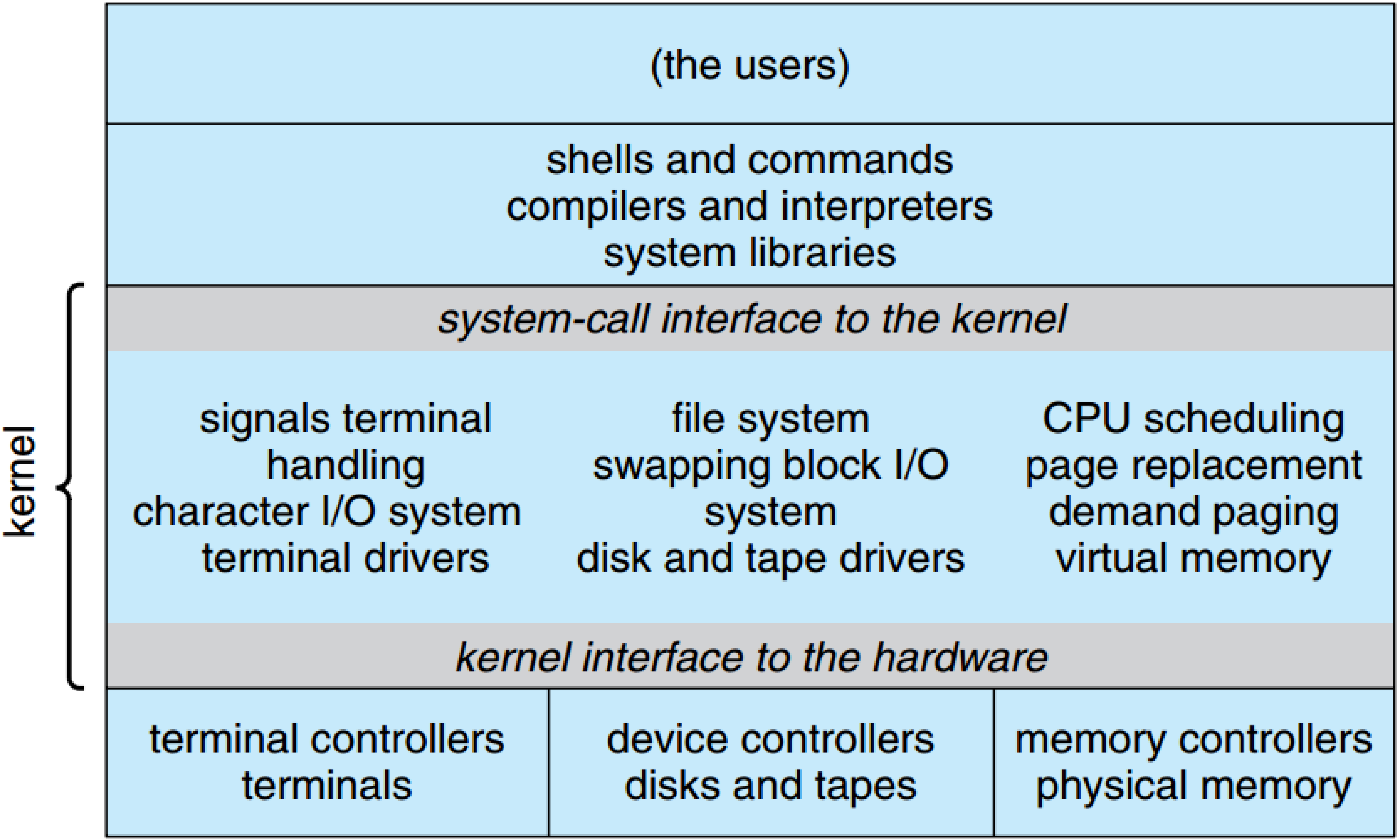
Simple Structure – MS DOS

- In MS-DOS, the interfaces and levels of functionality are ***not well separated***.
- For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives.
- Such freedom leaves MS-DOS ***vulnerable*** to errant (or malicious) programs, causing entire system crashes when user programs fail.
- MS-DOS was also limited by the hardware of its era. Because the Intel 8088 for which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the base hardware accessible.

Simple Structure - UNIX

- Like MS-DOS, UNIX initially was limited by hardware functionality.
- It consists of two separable parts: ***the kernel*** and the ***system programs***.
- The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved.
- We can view the traditional UNIX operating system as being ***layered*** to some extent.
- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.
- Taken in sum, that is an enormous amount of functionality to be combined into one level.

Simple Structure - UNIX



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

device controllers
disks and tapes

memory controllers
physical memory

kernel

Simple Structure - Summary

- This monolithic structure was ***difficult to implement*** and maintain.

- It had a distinct ***performance advantage***, however: there is very little overhead in the system call interface or in communication within the kernel.

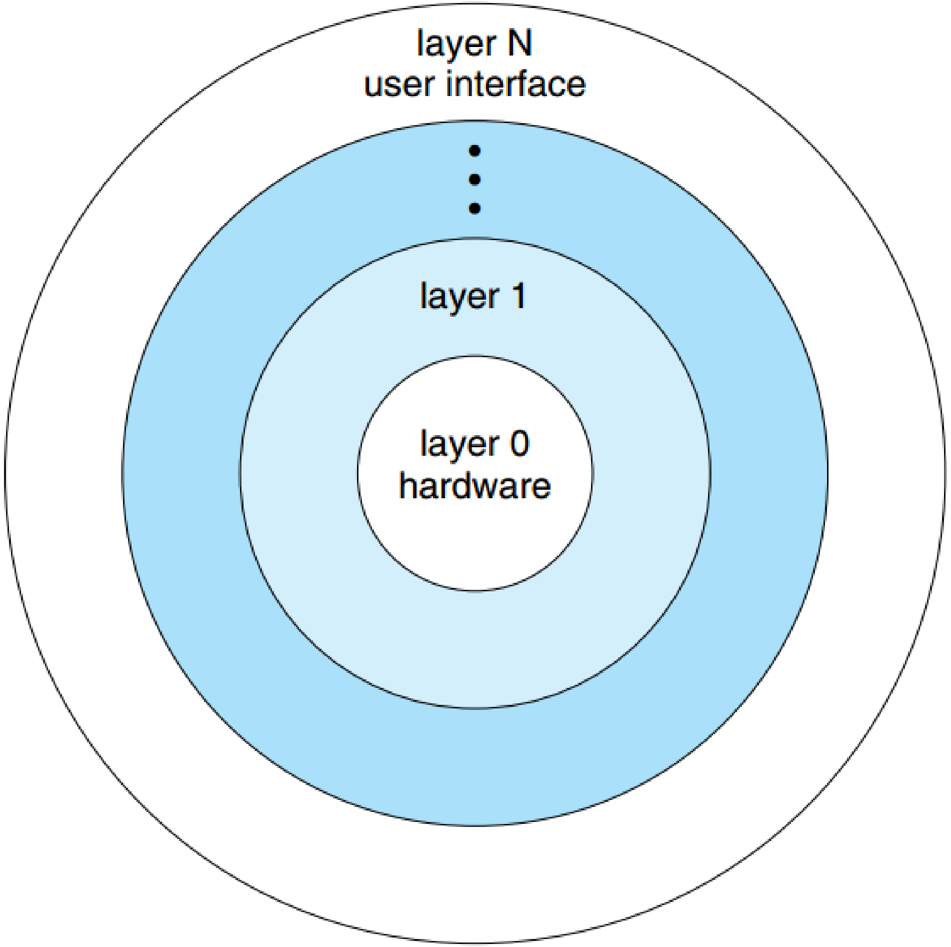
- We still see evidence of this simple, monolithic structure in the UNIX, Linux, and Windows operating systems.

2. Layered Approach

Layered Approach - Overview

- A system can be made modular in many ways.
- One method is the ***layered approach***, in which the operating system is broken into a number of layers (levels).
- The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

Layered Approach



Layered Approach

- An operating-system layer is an implementation of an abstract object made up of ***data*** and the ***operations*** that can manipulate those data.
- A typical operating-system layer—say, layer M—consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M, in turn, can invoke operations on lower-level layers.

Layered Approach -Benefits

- ***Simplicity of construction and debugging*:**
  - The layers are selected so that each uses functions (operations) and services of only lower-level layers.
  - This approach simplifies debugging and system verification.
  - The first layer can be debugged without any concern for the rest of the system.
  - Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on.
  - If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system are simplified.

Layered Approach -Benefits

- ***Abstraction***
  - Each layer is implemented only with operations provided by lower-level layers.
  - A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

Layered Approach -Challanges

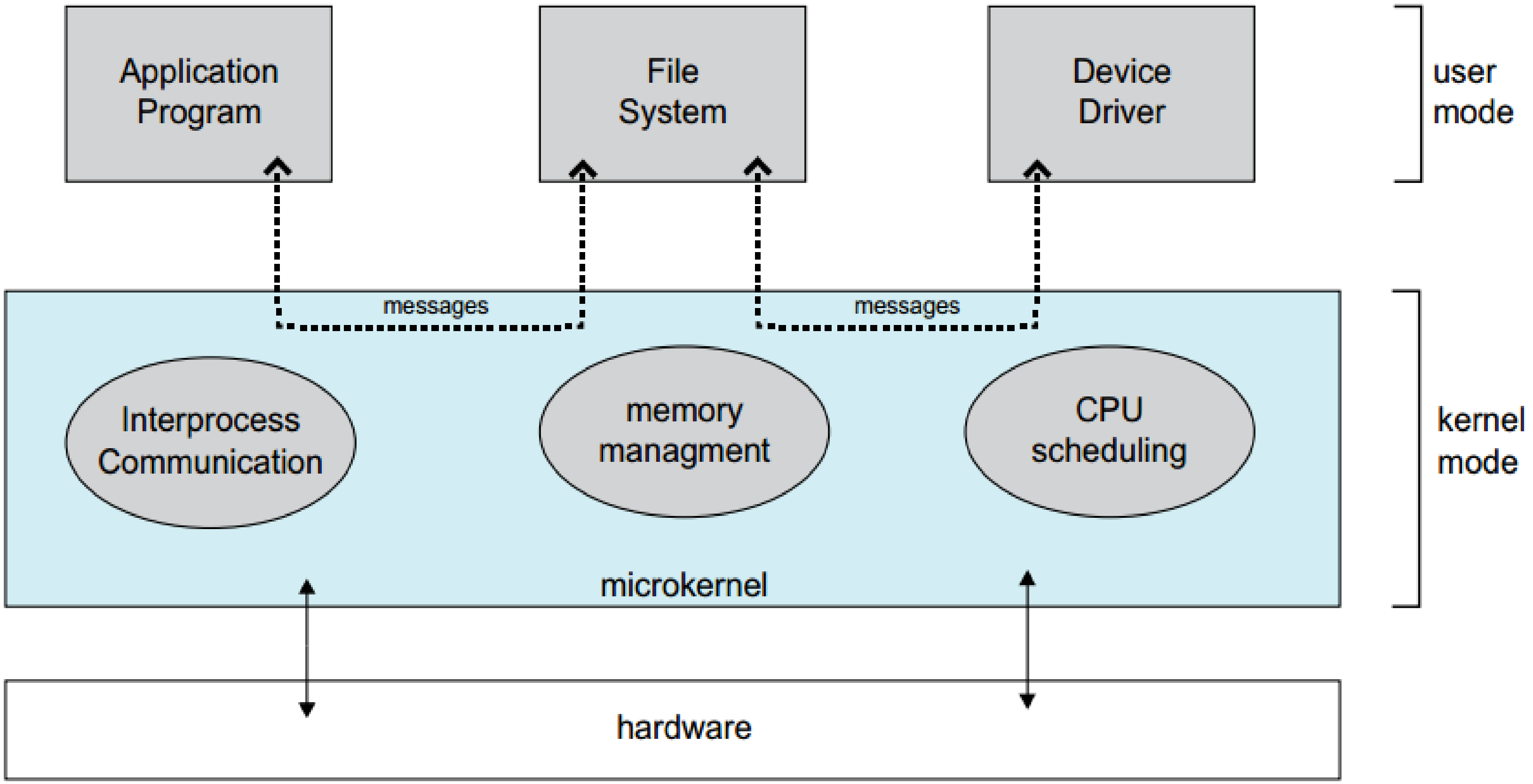
- ***Challenges appropriately defining the various layers:***
  - SInce a layer can use only lower-level layers, careful planning is necessary.
  - For example, the device driver for the backing store (disk space used by virtual-memory algorithms) must be at a lower level than the memory-management routines, because memory management requires the ability to use the backing store.
- ***Less efficient than other types:***
  - For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.
  - At each layer, the parameters may be modified, data may need to be passed, and so on.
  - Each layer adds overhead to the system call. The net result is a system call that takes longer than does one on a nonlayered system

3. Microkernels

Microkernels - Overview

- This method structures the operating system by removing all nonessential components from the kernel and implementing them as ***system and user-level programs***.
- The result is a smaller kernel.
- There is little consensus regarding which services should remain in the kernel and which should be implemented in user space.
- Microkernels provide ***minimal process and memory management***, in addition to a communication facility.
- The main function of the microkernel is to ***provide communication*** between the client program and the various services that are also running in user space. Communication is provided through message passing

Architecture of a typical microkernel



Microkernels - Benefits

- ***It makes extending the operating system easier:***
  - All new services are added to user space and consequently do not require modification of the kernel.
  - When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel. The resulting operating system is easier to port from one hardware design to another.
- ***Provides more security and reliability:***
  - Since most services are running as user—rather than kernel—processes. If a service fails, the rest of the operating system remains untouched.

4. Modules

Modules - Overview

- Perhaps the best current methodology for operating-system design involves using **loadable kernel modules**.
- Here, the kernel has a set of core components and links 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 Solaris, Linux, and Mac OS X, as well as Windows.
- The idea of the design is for the kernel to provide core services while other services are implemented dynamically, as the kernel is running.

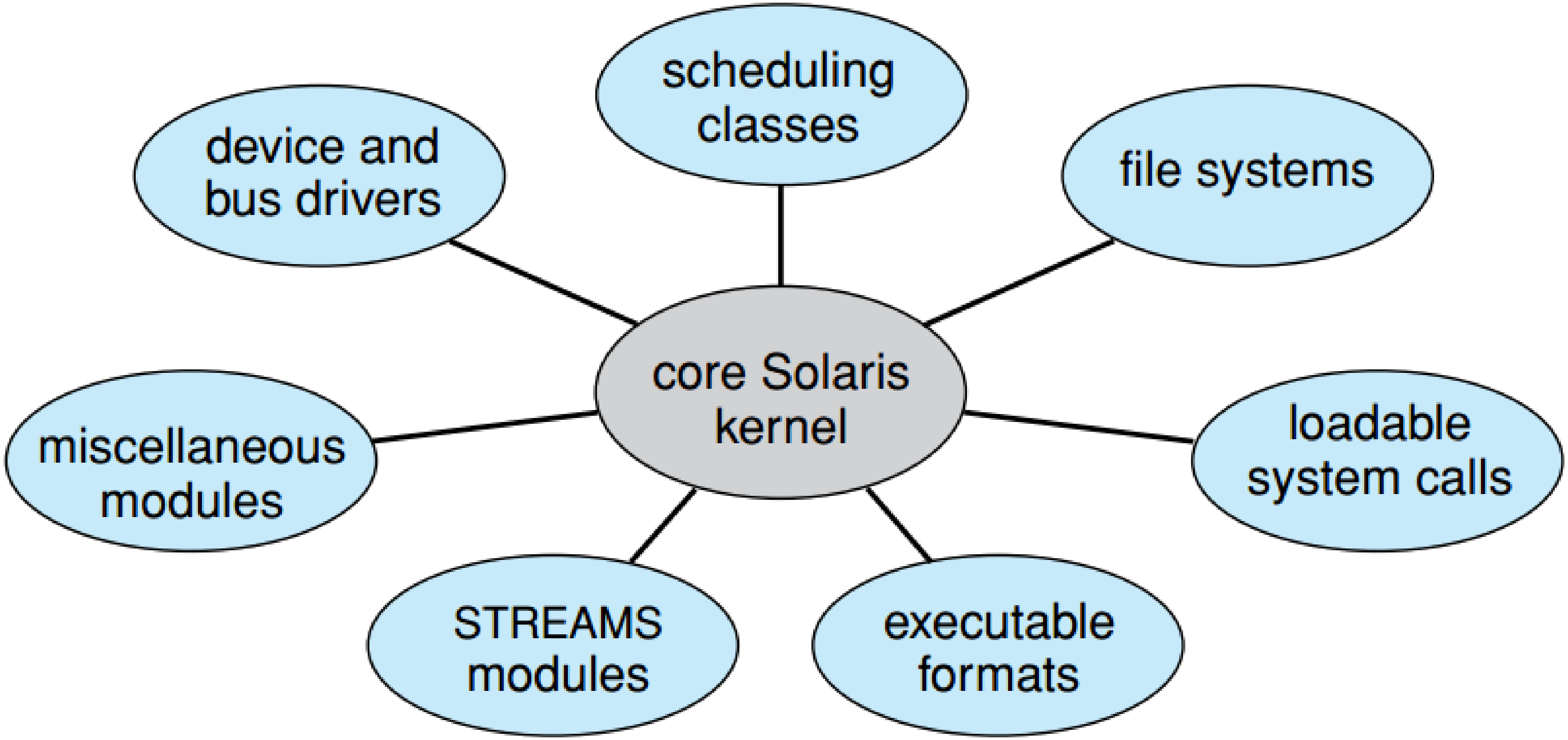
Modules - Overview

- Linking services dynamically is preferable to adding new features directly to the kernel, which would require recompiling the kernel every time a change was made.
- The approach is also similar to the microkernel approach in that the primary module has only core functions and knowledge of how to load and communicate with other modules; but it is more efficient, because modules do not need to invoke message passing in order to communicate.

Modules - Solaris operating system

- The Solaris operating system structure, is organized around a core kernel with seven types of loadable kernel modules:
    1. Scheduling classes
    2. File systems
    3. Loadable system calls
    4. Executable formats
    5. STREAMS modules
    6. Miscellaneous
    7. Device and bus drivers

Modules – Solaris operating system



5. Hybrid Systems

Hybrid Systems- Overview

- In practice, very few operating systems adopt a single, strictly defined structure.
- Instead, they combine different structures, resulting in ***hybrid systems*** that address performance, security, and usability issues.
  - For example, both Linux and Solaris are monolithic, because having the operating system in a single address space provides very efficient performance. However, they are also modular, so that new functionality can be dynamically added to the kernel.
  - Windows is largely monolithic as well (again primarily for performance reasons), but it retains some behavior typical of microkernel systems, including providing support for separate subsystems (known as operating-system personalities) that run as user-mode processes

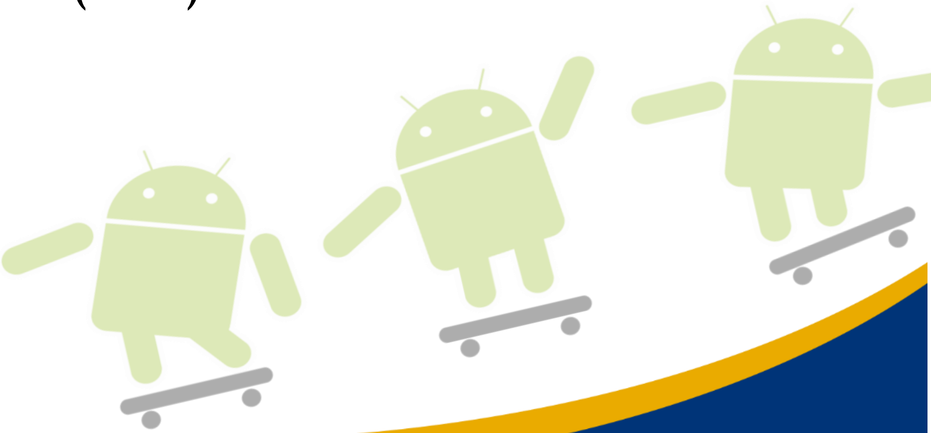
Task 1

- In this task, students are expected to explore the structure of three hybrid systems:
  - The Apple Mac OS X operating system
  - Mobile operating systems
    - iOS
    - Android.
- Draw the diagrams of each of the Operating system structures, illustrating the roles of the different layers of the operating system.

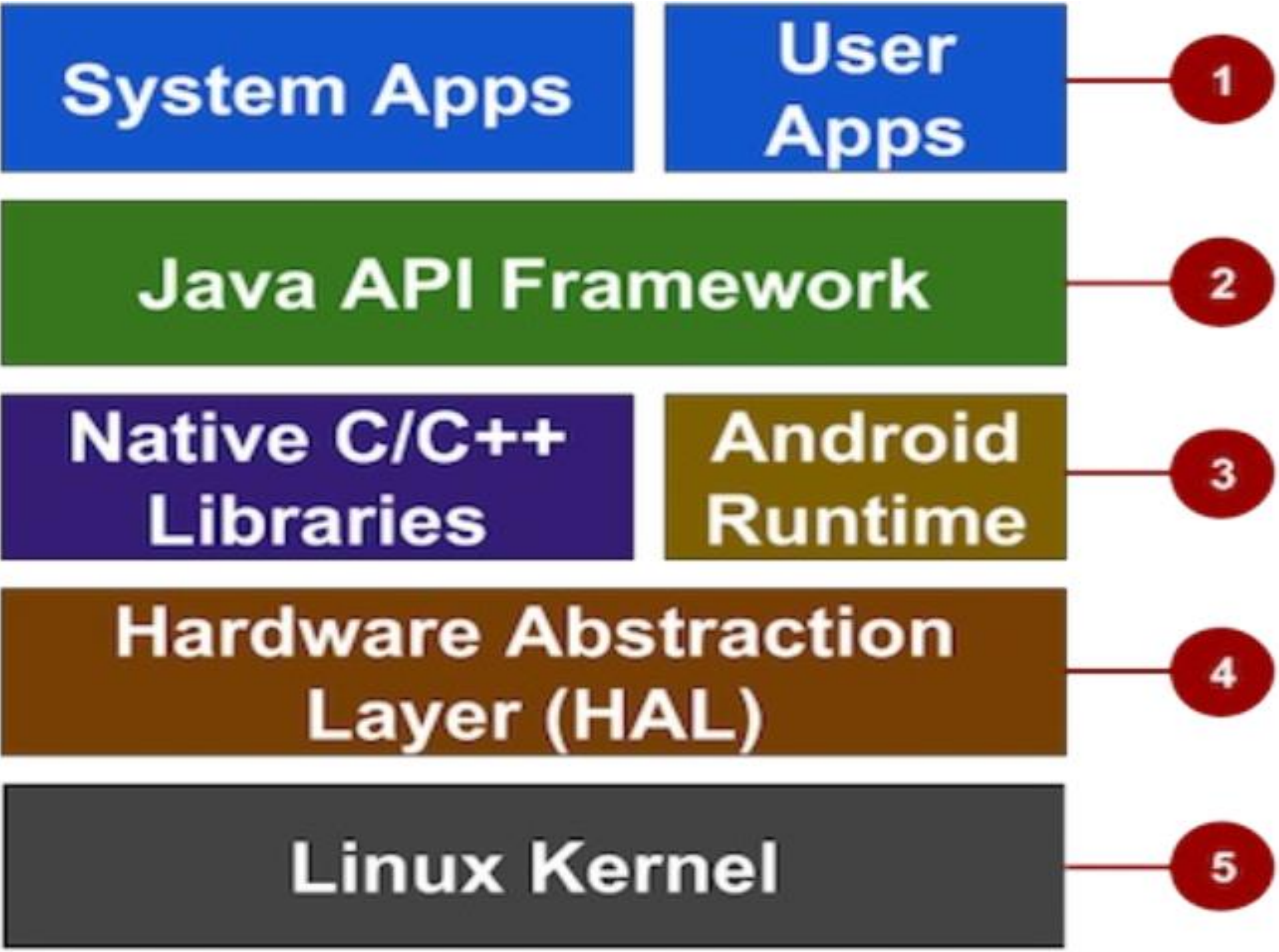
Android Operating System Structure

Platform Architecture

- Android is an open source, Linux-based software stack created for a wide array of devices and form factors.
- The following list shows the major components of the Android platform:
    1. System Apps
    2. Java Api Framework
    3. Native C++ Libraries
    4. Android Runtime
    5. Hardware Abstraction Layer (HAL)
    6. Linux Kernel



Platform Architecture



System Apps

User Apps

Java API Framework

Native C/C++ Libraries

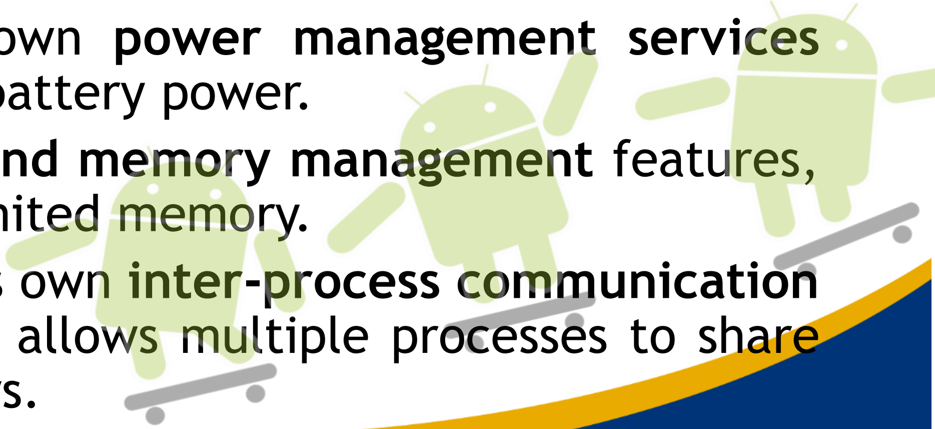
Android Runtime

Hardware Abstraction Layer (HAL)

Linux Kernel

1. Linux Kernel

- The foundation of the Android platform is the Linux kernel. The above layers rely on the Linux kernel for underlying functionalities such as:
- It provides a **permissions** architecture so that you can restrict access to data and resources to only those processes that have the proper authorizations.
- It supports **memory and process management**, so that multiple processes can run simultaneously without interfering with each other.
- It handles low-level details of **file and network I/O**.
- And it also allows **device drivers** to be plugged in so that Android can communicate with a wide range of low-level hardware components that are often coupled to computing devices, things like memory and radios and cameras.
- Android's Linux kernel includes its own **power management services** because mobile devices often run on battery power.
- It provides its own **memory sharing and memory management** features, because mobile devices often have limited memory.
- Android's Linux kernel also includes its own **inter-process communication mechanism** called the **binder**, which allows multiple processes to share data and services in sophisticated ways.



2. Hardware Abstraction Layer (HAL)

- This layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
- The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module.
- When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.
- Android gives you the freedom to implement your own device specifications and drivers. The hardware abstraction layer (HAL) provides a standard method for creating software hooks between the Android platform stack and your hardware.

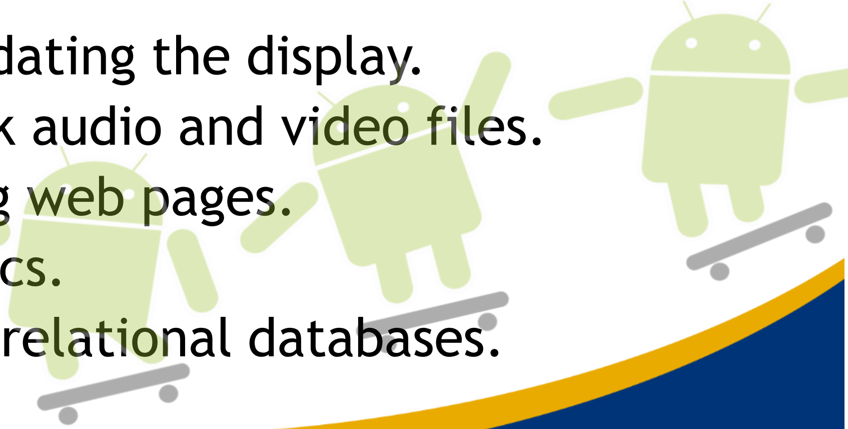
3. Android Runtime

- For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART).
- ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed specially for Android that's optimized for minimal memory footprint.
- Build toolchains, such as Jack, compile Java sources into DEX byte code, which can run on the Android platform.
- Some of the major features of ART include the following:

1. Ahead-of-time (AOT) and just-in-time (JIT) compilation
2. Optimized garbage collection (GC)
3. Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watch points to monitor specific fields

4. Native C/C++ Libraries

- Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++.
- These libraries are typically written in C and C++, and for that reason they are often referred to as the native libraries.
- These native libraries handle a lot of the core, performance-sensitive activities on your device, things like quickly rendering web pages and updating the display.
- Android has its own **System C Library**, which implements the standard OS system calls, which do things like process and thread creation, mathematical computations, memory allocation, and much more.
- There's the **Surface Manager**, for updating the display.
- A **media framework**, for playing back audio and video files.
- **Webkit**, for rendering and displaying web pages.
- **OpenGL** for high performance graphics.
- And **SQLite** for managing in-memory relational databases.



5. Java API Framework

- These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

i. View System you can use to build an app's UI, including lists, grids, text boxes, buttons, webviews

ii. A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files

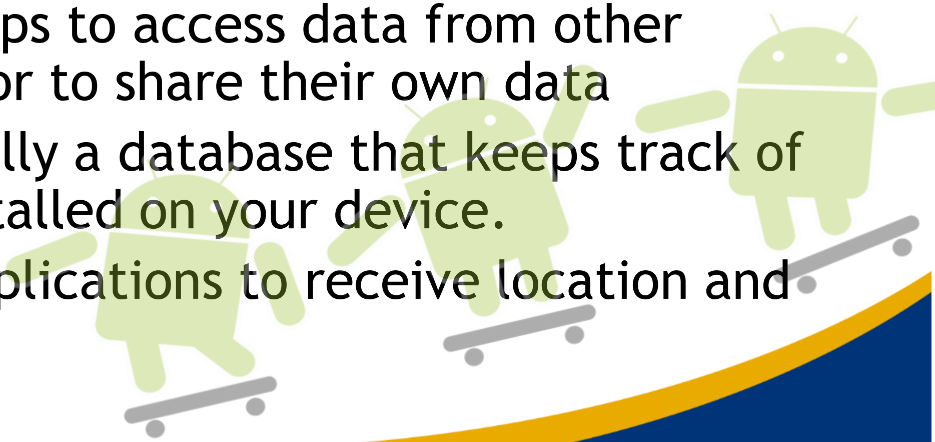
iii. A Notification Manager that enables all apps to display custom alerts in the status bar

iv. An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack

v. Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

vi. The Package Manager is essentially a database that keeps track of all the applications currently installed on your device.

vii. The Location Manager allows applications to receive location and movement information



6. System Apps/ Application layer

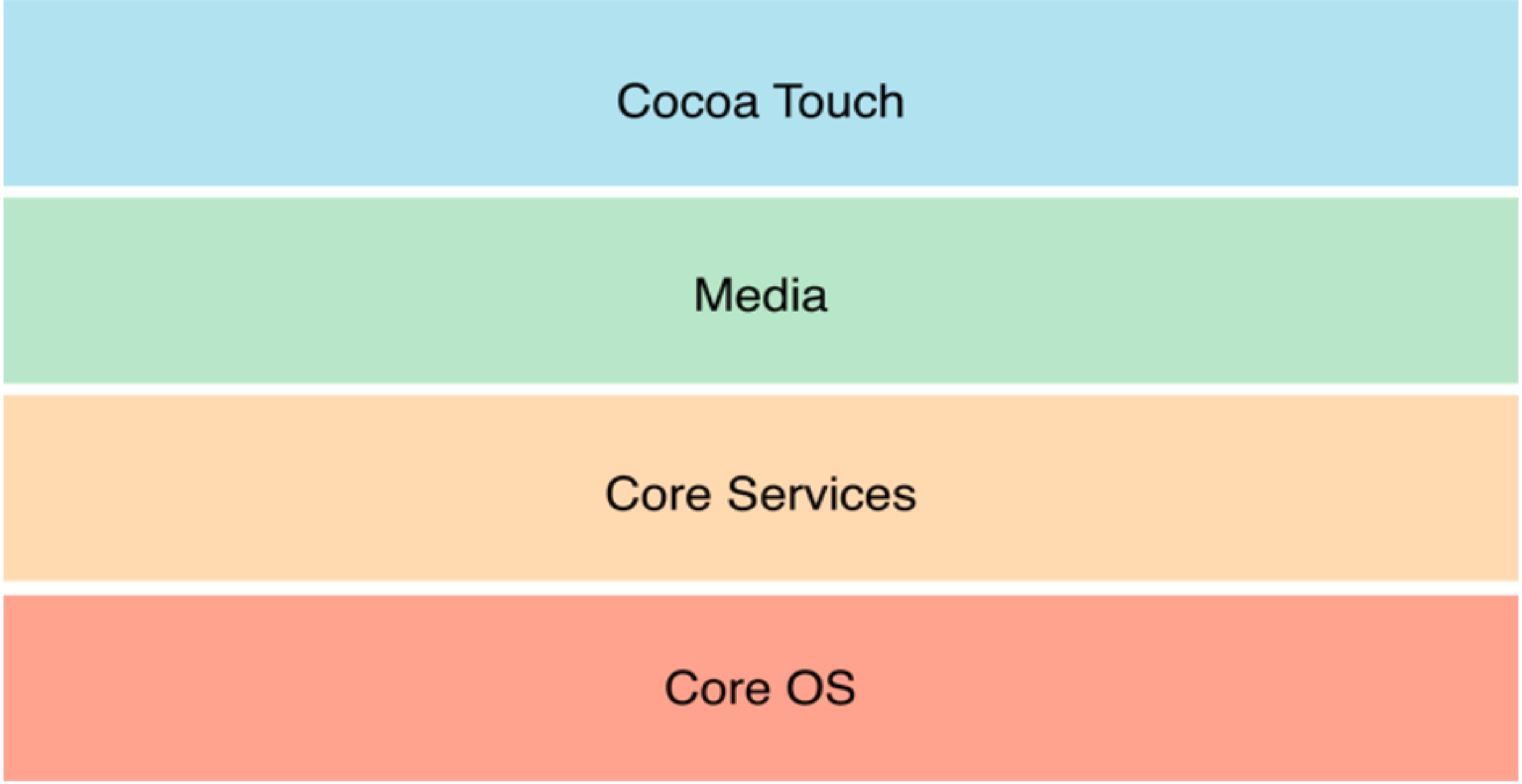
- Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more.
- Apps included with the platform have no special status among the apps the user chooses to install.
- So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app).
- The system apps function both as apps for users and to provide key capabilities that developers can access from their own app.
- For example, if your app would like to deliver an SMS message, you don't need to build that functionality yourself—you can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.

iOS Operating System Structure

Architecture of iOS

- Architecture of IOS is a layered architecture.
- Apps do not communicate to the underlying hardware directly.
  Apps talk with the hardware through a collection of well defined system interfaces.
- These interfaces make it simple to write apps that work constantly on devices having various hardware abilities.
- The layers are:
  - *The Core OS layer*
  - *The Core Services layer*
  - *The Media layer*
  - *The Cocoa-Touch layer*

Architecture of iOS



Cocoa Touch

Media

Core Services

Core OS

1. Core OS Layer

- The Core OS layer holds the low level features that most other technologies are built upon.
- This is the bottom layer of the hierarchy and is responsible for the foundation of the operating system which the other layers sit on top of.
- This important layer is in charge of managing memory—allocating and releasing memory once the application has finished with it, taking care of file system tasks, handling networking, and other operating system tasks. It also interacts directly with the hardware.

2. Core Services Layer

- Some of the Important Frameworks available in the core services layers are detailed:
  - **Address book framework** – Gives programmatic access to a contacts database of user.
  - **Cloud Kit framework** – Gives a medium for moving data between your app and iCloud.
  - **Core data Framework -** Technology for managing the data model of a Model View Controller app.
  - **Core Foundation framework -** Interfaces that gives fundamental data management and service features for iOS apps.
  - **Core Location framework -** Gives location and heading information to apps.

2. Core Services Layer (cont)

- **Core Motion Framework -** Access all motion based data available on a device. Using this core motion framework Accelerometer based information can be accessed.
- **Foundation Framework -** Objective C covering too many of the features found in the Core Foundation framework
- **Healthkit framework -** New framework for handling health-related information of user
- **Homekit framework -** New framework for talking with and controlling connected devices in a user's home.
- **Social framework -** Simple interface for accessing the user's social media accounts.
- **StoreKit framework -** Gives support for the buying of content and services from inside your iOS apps, a feature known asIn-App Purchase.

3. Media Layer - Graphics Framework

- **UIKit Graphics -** It describes high level support for designing images and also used for animating the content of your views.
- **Core Graphics framework -** It is the native drawing engine for iOS apps and gives support for custom 2D vector and image based rendering.
- **Core Animation -** It is an initial technology that optimizes the animation experience of your apps.
- **Core Images -** gives advanced support for controling video and motionless images in a nondestructive way
- **OpenGl ES and GLKit -** manages advanced 2D and 3D rendering by hardware accelerated interfaces
- **Metal -** It permits very high performance for your sophisticated graphics rendering and computation works. It offers very low overhead access to the A7 GPU.

3. Media Layer - Audio Framework:

- **Media Player Framework -** It is a high level framework which gives simple use to a user’s iTunes library and support for playing playlists.
- **AV Foundation -** It is an Objective C interface for handling the recording and playback of audio and video.
- **OpenAL -** is an industry standard technology for providing audio.

3. Media Layer - Video Framework

- **AV Kit -** framework gives a collection of easy to use interfaces for presenting video.
- **AV Foundation -** gives advanced video playback and recording capability.
- **Core Media -** framework describes the low level interfaces and data types for operating media.

4. Cocoa Touch Layer

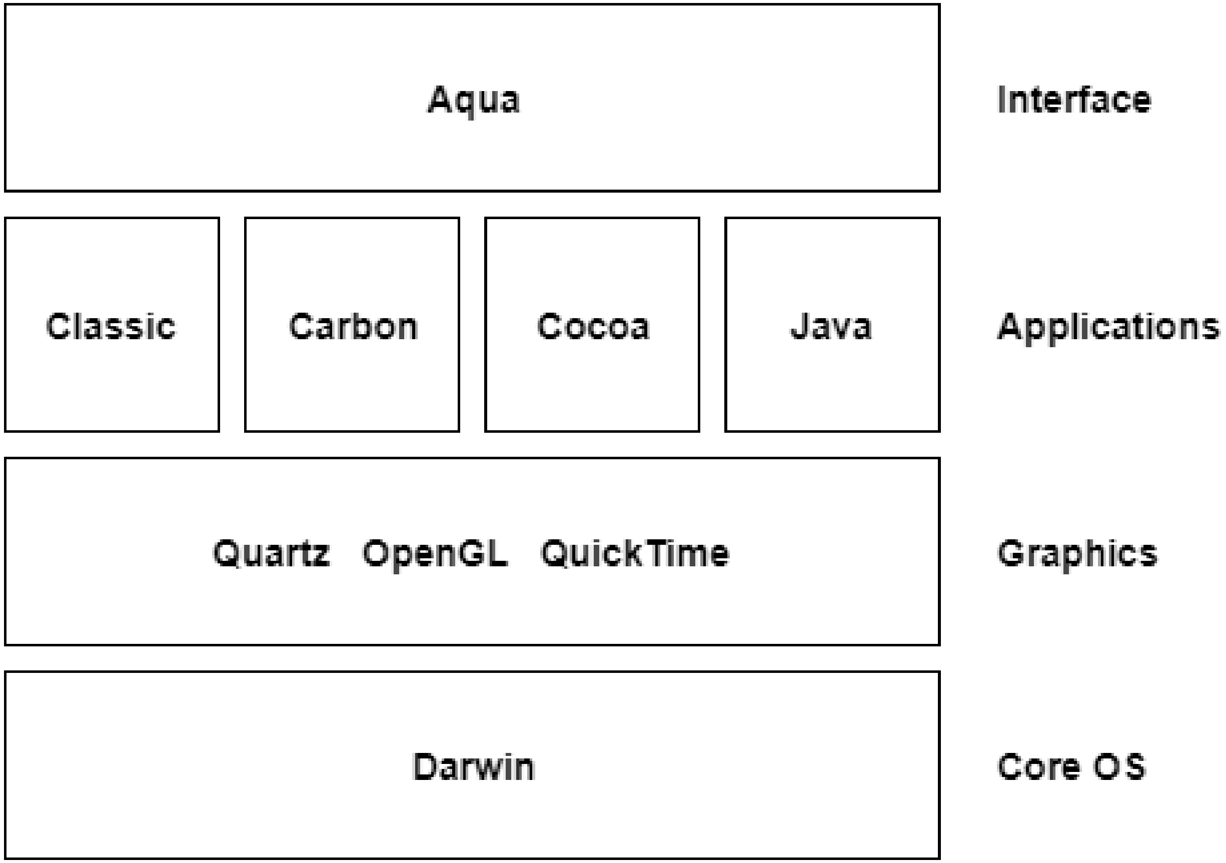
- **EventKit framework -** gives view controllers for showing the standard system interfaces for seeing and altering calendar related events
- **GameKit Framework -** implements support for Game Center which allows users share their game related information online
- **iAd Framework -** allows you deliver banner-based advertisements from your app.
- **MapKit Framework -** gives a scrollable map that you can include into your user interface of app.
- **PushKitFramework -** provides registration support for VoIP apps.
- **Twitter Framework -** supports a UI for generating tweets and support for creating URLs to access the Twitter service.
- **UIKit Framework -** gives vital infrastructure for applying graphical, event-driven apps in iOS. Some of the Important functions of UI Kit framework:

MAC OSX Operating System Structure

MAC OSX Operating System Structure

- The structure of the Mac OS X includes multiple layers.
- The base layer is Darwin which is the Unix core of the system.
- Next layer is the graphics system which contains Quartz, OpenGL and QuickTime.
- Then is the application layer which has four components, namely Classic, Carbon, Cocoa and Java.
- The top layer is Aqua, which is the user interface.

MAC OSX Operating System Structure



Aqua — Interface

Classic Carbon Cocoa Java — Applications

Quartz OpenGL QuickTime — Graphics

Darwin — Core OS

1. The Core OS: Darwin

- Mac OS X is built on a Unix core; the Darwin core is based on the ***Berkeley Software Distribution (BSD)*** version of Unix.
- The heart of the Darwin core is called Mach.
- This part of the operating system performs the fundamental tasks, such as data flow into and from the CPU, memory use, and so on. Mach's major features include the following:
  - ***Protected memory*****.** Mach provides a separate memory area in which each application can run. It ensures that each application remains in its own memory space and so does not affect other applications. Therefore, if a running application crashes or hangs, other applications aren't affected. You can safely shut down the hung application and continue working in the others.
  - ***Automatic memory management***: Mac OS X manages RAM for you; it automatically allocates RAM to applications that need it. Under Mac OS X, you don't need to think about how RAM is being used; the OS takes care of it for you.

1. The Core OS: Darwin (2)

- ***Preemptive multitasking:*** Under Mac OS X (or, more specifically, Mach), the operating system controls the processes that the processor is performing to ensure that all applications and system services have the resources they need and that the processor is used efficiently. This ensures both stability and maximum performance for both foreground and background processes.
- ***Advanced virtual memory:*** The Mach core uses a virtual memory system that is always on. It manages the virtual memory use efficiently so that virtual memory is used only as necessary to ensure maximum performance.

1. The Core OS: Darwin (2)

- Mac OS X uses bundles;
- ***A bundle*** is a directory containing a set of files that provide services.
- A bundle contains executable files and all the resources associated with those executables; when they are a file package, a bundle can appear as a single file.

1. The Core OS: Darwin (3)

- The three types of bundles under Mac OS X are as follows:
  - ***Applications***: Under Mac OS X, applications are provided in bundles. Frequently, these bundles are designed as file packages so the user sees only the files with which he needs to work, such as the file to launch the application.
  - ***Framework***: A framework bundle is similar to an application bundle except that a framework provides services that are shared across the OS; frameworks are system resources. A framework contains a dynamic shared library, meaning different areas of the OS as well as applications can access the services provided by that framework.
  - ***Loadable bundle***: Loadable bundles are executable code (just like applications) available to other applications and the system (similar to frameworks) but must be loaded into an application to provide their services. The two main types of loadable bundles are plug-ins (such as those used in Web browsers) and palettes (which are used in building application interfaces). Loadable bundles can also be presented as a package so the user sees and works with only one file.

2. The Graphics Subsystem

- Mac OS X includes an advanced graphics subsystem, which has three main components: Quartz Extreme, OpenGL, and QuickTime.
  - ***Quartz Extreme*** handles 2D graphics. Quartz provides the interface graphics, fonts, and other 2D elements of the system, as well as on-the-fly rendering and antialiasing of images. Under Mac OS X, the Portable Document Format (PDF) is native to the OS. This means you can create PDF versions of any document without using a third-party application, such as Adobe Acrobat (to get special features in PDF documents, such as navigation features, you still need to use an application that provides those features).
  - The ***OpenGL component*** of the graphics subsystem provides 3D graphics support for 3D applications. OpenGL is an industry standard that is also used on Windows and Unix systems. Because of this, it is easier to create 3D applications for the Mac from those that were designed to run on those other operating systems. The Mac OS X implementation of OpenGL provides many 3D graphics functions, such as texture mapping, transparency, antialiasing, atmospheric effects, other special effects, and more

2. The Graphics Subsystem

- ***QuickTime*** provides support for many types of digital media, such as digital video, and is the primary enabler of video and audio streaming under Mac OS X. QuickTime enables both viewing applications, such as the QuickTime Player, and creative applications, such as iMovie, iTunes, and many more. QuickTime is also an industry standard, and QuickTime files can be used on Windows and other computer platforms.

4. The Application Subsystem

- Mac OS X provides the Classic environment to enable it to run Classic applications. It also includes three application development environments: Carbon, Cocoa, and Java 2.
  - The ***Classic environment*** enables Mac OS X to run applications that were written for previous versions of the OS without modification. This provides access to thousands of existing applications that will run under Mac OS X. Classic applications run as they do under previous versions of the Mac OS; in other words, they do not benefit from the advanced features of Mac OS X such as protected memory (Classic applications can be affected by other Classic applications, and the Classic environment itself can be affected when a Classic application has problems).
  - The ***Carbon environment*** enables developers to port existing applications to use Carbon application program interfaces (APIs); the process of porting a Classic application into the Carbon environment is called Carbonizing it. The Carbon environment offers the benefits of Darwin for Carbonized applications, such as protected memory and preemptive multitasking. Carbonizing an application is significantly less work than creating a new application from scratch, which enabled many applications to be delivered near the release of Mac OS X.

4. The Application Subsystem (2)

- The ***Cocoa environment*** offers developers a state-of-the-art, object-oriented application development environment. Cocoa applications are designed for Mac OS X from the ground up and take the most advantage of Mac OS X services and benefits. Most of the applications included with Mac OS X are Cocoa versions; as time passes, more and more Cocoa applications will become available and will eventually be the dominant type under Mac OS X.
- The ***Java environment*** enables you to run Java applications, including pure Java applications and Java applets. Java applications are widely used on the Web because they enable the same set of code to be executed on various platforms. You can also develop Java applications under Mac OS X.

4.The User Interface (2)

- The Mac OS X user interface, called Aqua, provides Mac OS X's great visual experience as well as the tools you use to interact with and customize the interface to suit your preferences.
- From the drop shadows on open windows to the extensive use of color and texture to the extremely detailed icons, Aqua provides a user experience that is both pleasant and efficient.

Strathmore

UNIVERSITY

Ole Sangale Road, Madaraka Estate. PO Box 59857-00200, Nairobi, Kenya
Tel: (+254) (0)703 034000/200/300 Fax : +254 (0)20 607498
Email: info@strathmore.edu Website: www.strathmore.edu