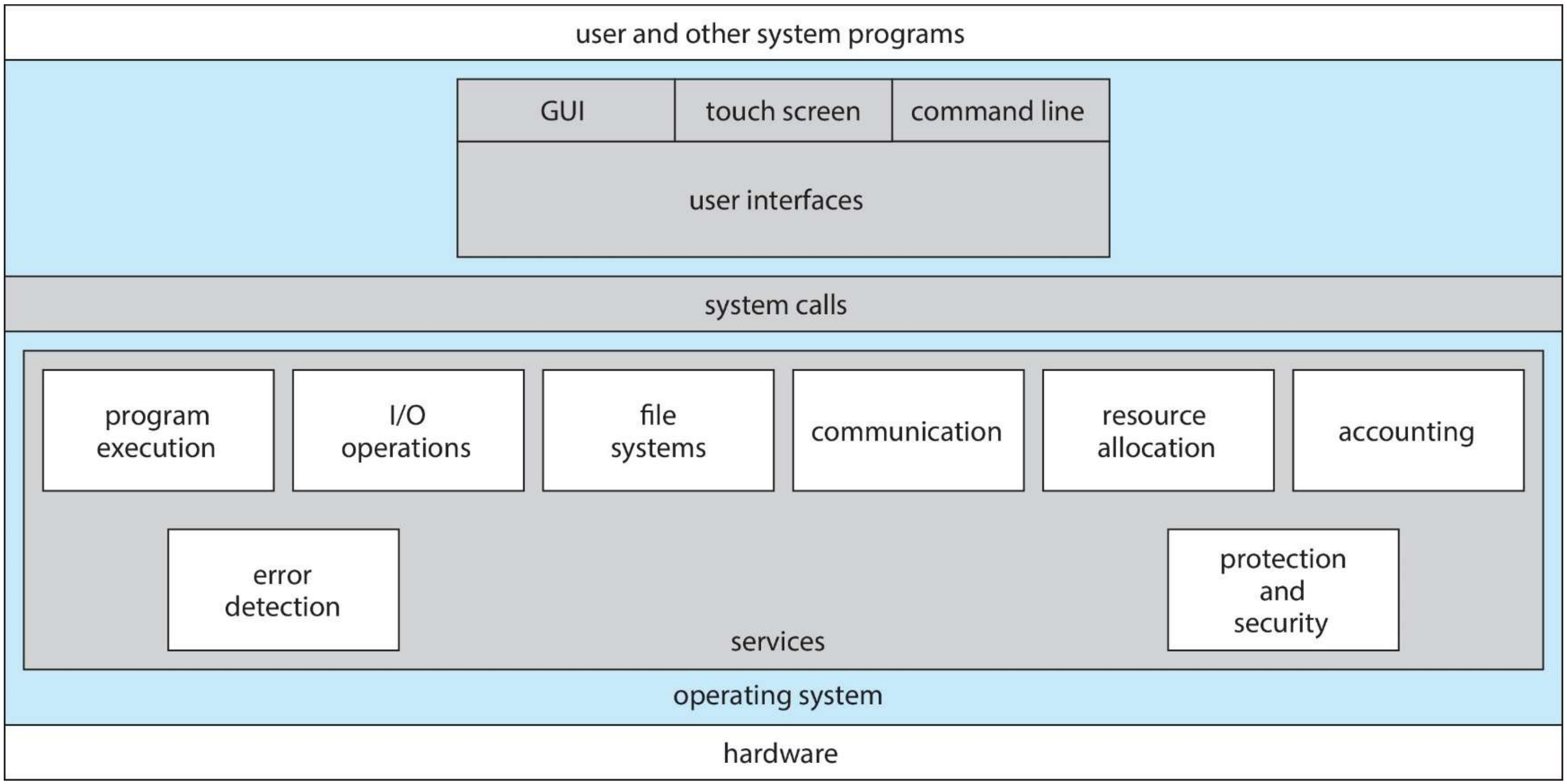
Operating Systems

Dr. Vivek Nallur (vivek.nallur@ucd.ie)

Typical Operating System Services



Commandline

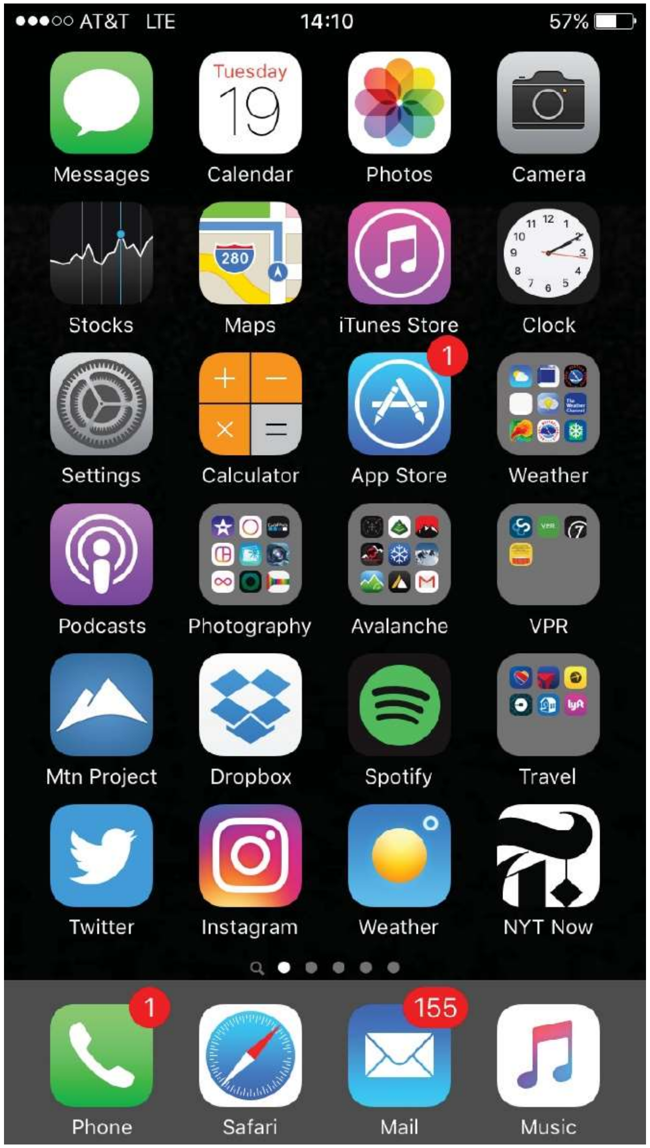
- CLI allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new 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 (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- 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

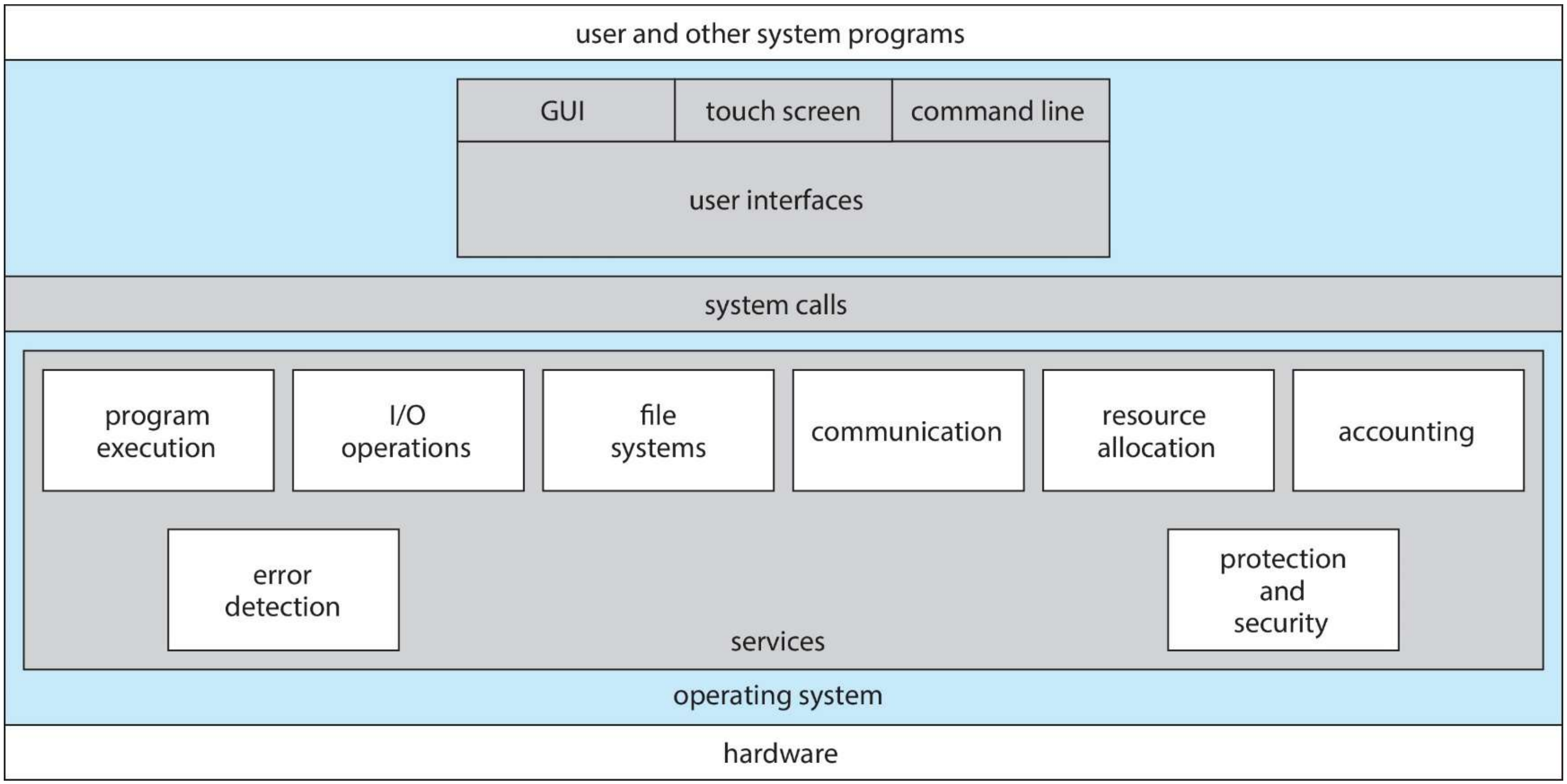
- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands



Operating System Structure

- In order to understand any operating system structure, we must consider its **components** and their organisation
- What are the essential components?
- How do they relate to each other?

Mapping to Structure?



Operating System Structure

- It is important to remember:
  - The concepts we study will exist in some form in every operating system
    - But they will be implemented in different ways

- Divisions between components are not always clearly defined

Operating System Components

- **Kernel**: software containing the core OS components; it may typically include:

- Memory Manager
  - Provides efficient memory allocation and deallocation of memory

- I/O manager
  - Handles input and output requests from and to hardware devices (through device drivers)

Operating System Components

- Inter-process communication (IPC) manager
  - Provides communication between different processes (programs in execution)
- Process Manager (scheduler)
  - Handles what is executed when and where (if more than one CPU)

Operating System Components

- A OS kernel may consist of many more components:
  - System service routines
  - File System (FS) manager
  - Error handling systems
  - Accounting systems
  - System programs
  - And many more

OS Structure Issues:

- How are all of these components organised?
- What are the entities involved and where do they exist?
- How do these entities cooperate?

Operating System Goals

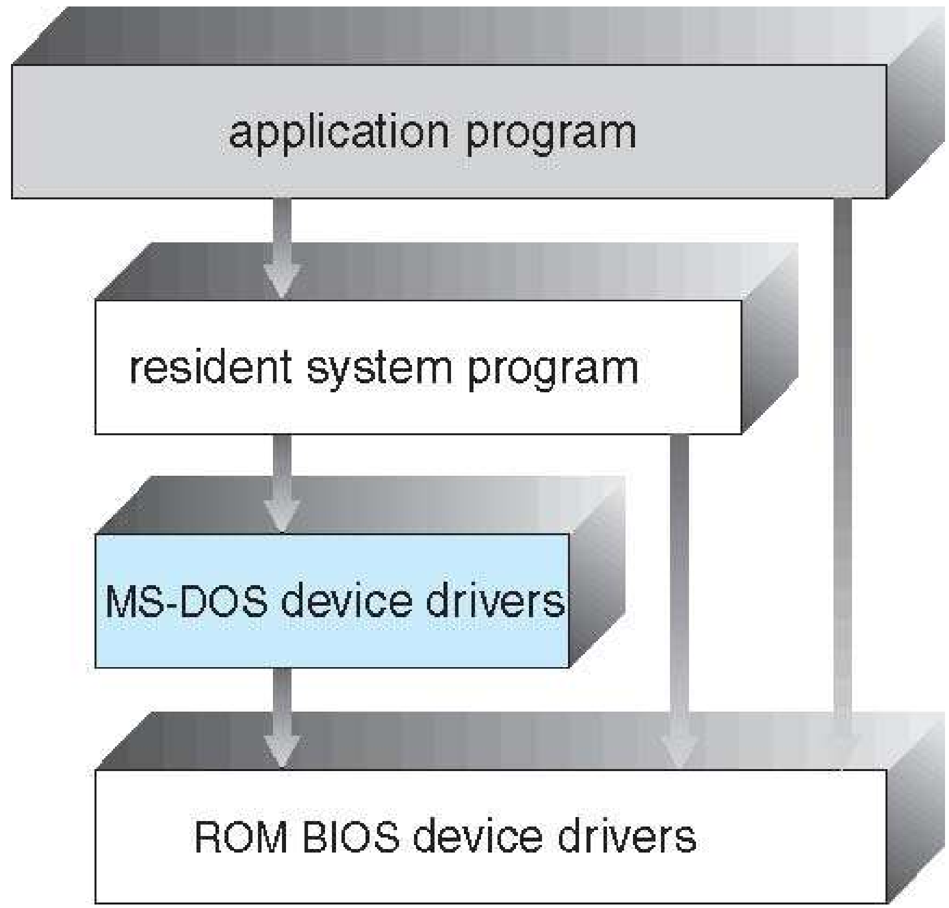
- When we design an operating system we want it to be:
  - Efficient – (High **throughput**)
  - Interactive
  - **Robust** – (Fault tolerant & reliable)
  - Secure
  - Scalable
  - Extensible
  - Portable

Monolithic Architecture

- Traditionally, systems were built around **monolithic** kernels
  - every OS component is contained in the kernel
  - any component can directly communicate with any other (by means of function calls)
  - due to this they tend to be highly efficient (performance)

Example : MS DOS

- Written to provide the most functionality in the least space
- Not divided into modules
- Interfaces and levels of functionality are not well separated



Problems with Monolithic Structure

- Because they are unstructured they are hard to understand, modify, and maintain

- Susceptible to damage from errant or malicious code, as all kernel code runs with unrestricted access to the system

Layered Structure

- To try and solve the problems with the monolithic structure, the layered structure was developed
- Components are grouped into layers that perform similar functions

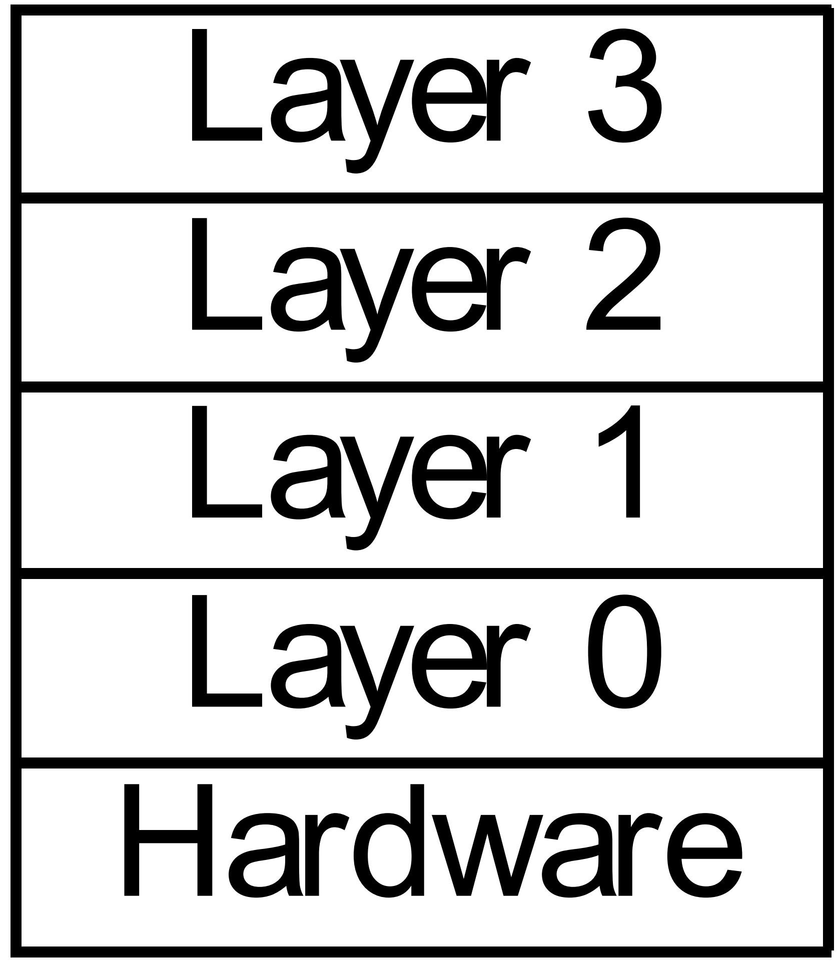
Advantages of Layered Structure

- Designing the system as a number of modules gives the system structure and consistency

- This allows easier debugging, modification and reuse

Layered Architecture

- Each layer communicates only with layers immediately above and below it
  - each layer is a virtual machine to the layer above
  - a higher layer provides a higher-level virtual machine



First Layers-based OS

- Layering was first used in Dijkstra's THE OS (1968)

- Each layer "sees" a logical machine provided by lower layers

The Layers

layer 4	User Programs
layer 3	I/O Management
layer 2	Console Device (commands), IPC
layer 1	Memory Management
layer 0	CPU Scheduling (multiprogramming)
	Hardware

First Layers-based OS

- Each layer "sees" a logical machine provided by lower layers
  - layer 4 (user space) sees virtual I/O drivers
  - layer 3 sees virtual console
  - layer 2 sees virtual memory
  - layer 1 sees virtual processors
- Based on a static set of cooperating processes
- Each process can be tested and verified independently

Problems with layering

- Appropriate definition of layers is difficult
  - A layer is implemented using only those operations provided by lower-lever layers
  - A real system structure is often **more complex** than the strict hierarchy required by layering

Problems with layering

- The secondary memory (disk) driver would normally placed **above** the CPU scheduler (because an I/O wait may trigger a CPU rescheduling operation)
- However, in a large system the CPU scheduler may need more memory than can fit in memory: parts of the memory can be swapped to disk (virtual memory), and then the secondary memory driver should be **below** the CPU scheduler
- Both things cannot be achieved at the same time; therefore the layered approach is not flexible

Problems with Layering

- Performance issues
  - Processes' requests might pass through many layers before completion (layer crossing)
  - System throughput can be lower than in monolithic kernels

Problems with Layering

- Still susceptible to malicious/errant code if all layers can have unrestricted access to the system
  - As we will see later, this can only be avoided through hardware

- As a consequence, (imperfect) layers are often used for convenience in system design, but any OS implementation cannot be purely layered

Microkernel Architecture

- The **microkernel** (μ-kernel) architecture was designed to minimise the services offered by the kernel

- This was an attempt to keep it small and scalable

Microkernel Architecture

- There is no agreement about minimal set of services inside the microkernel
- At least: minimal process and memory management capabilities, plus inter-process communications
- Services such as networking and file system tend to run nonprivileged at the user process level

Benefits of Microkernel

- Modularity
- It promotes uniform interfaces
- Distributed systems support:
  - modules communicate through the microkernel, even through a network

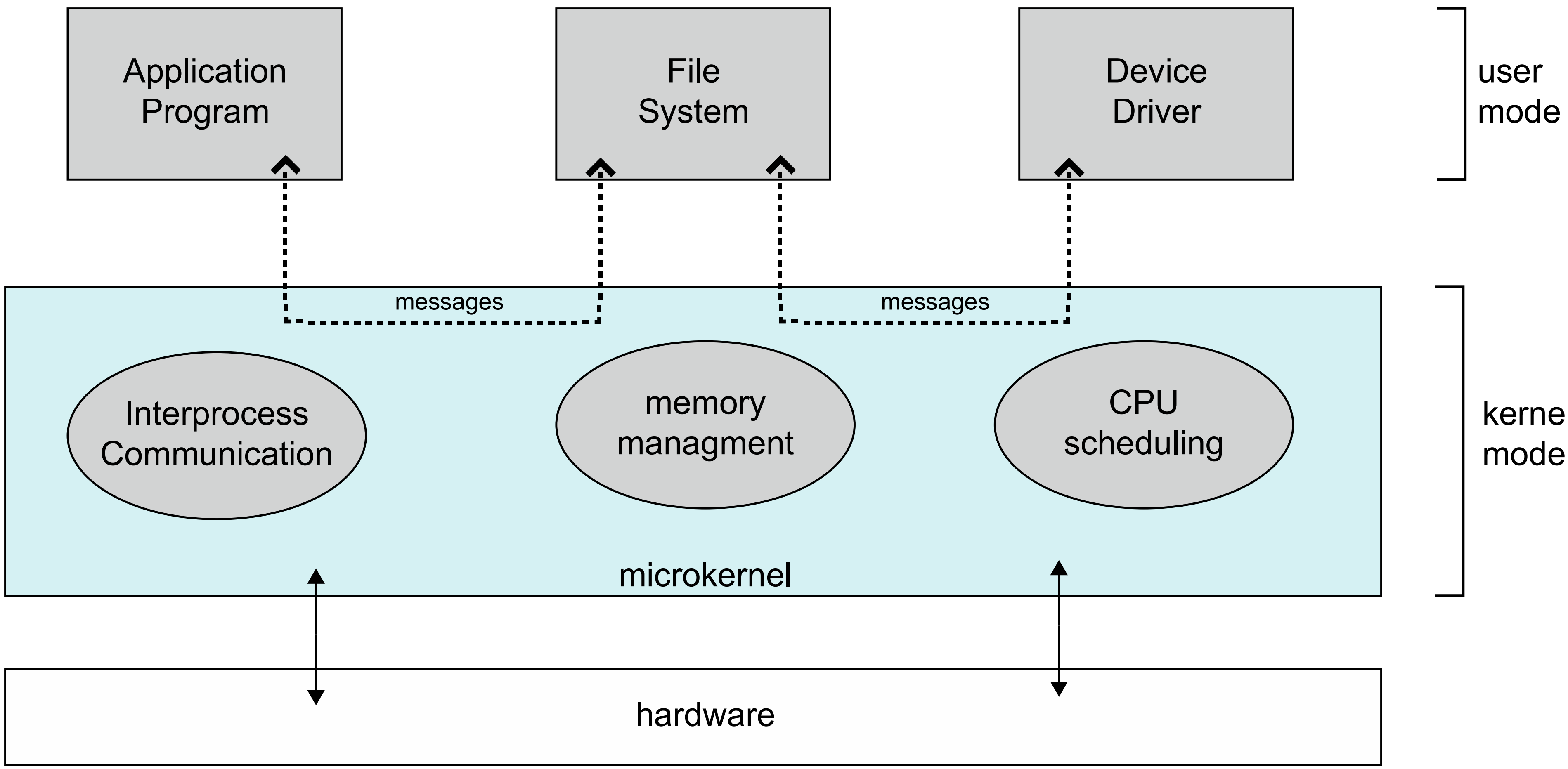
Benefits of Microkernel

- Reliability
- Scalability
- Portability
- Easy to extend and customise

Disadvantages of Microkernel

- System performance can be worse than in monolithic kernels, especially if kernel minimisation is taken too far

Microkernel System Structure

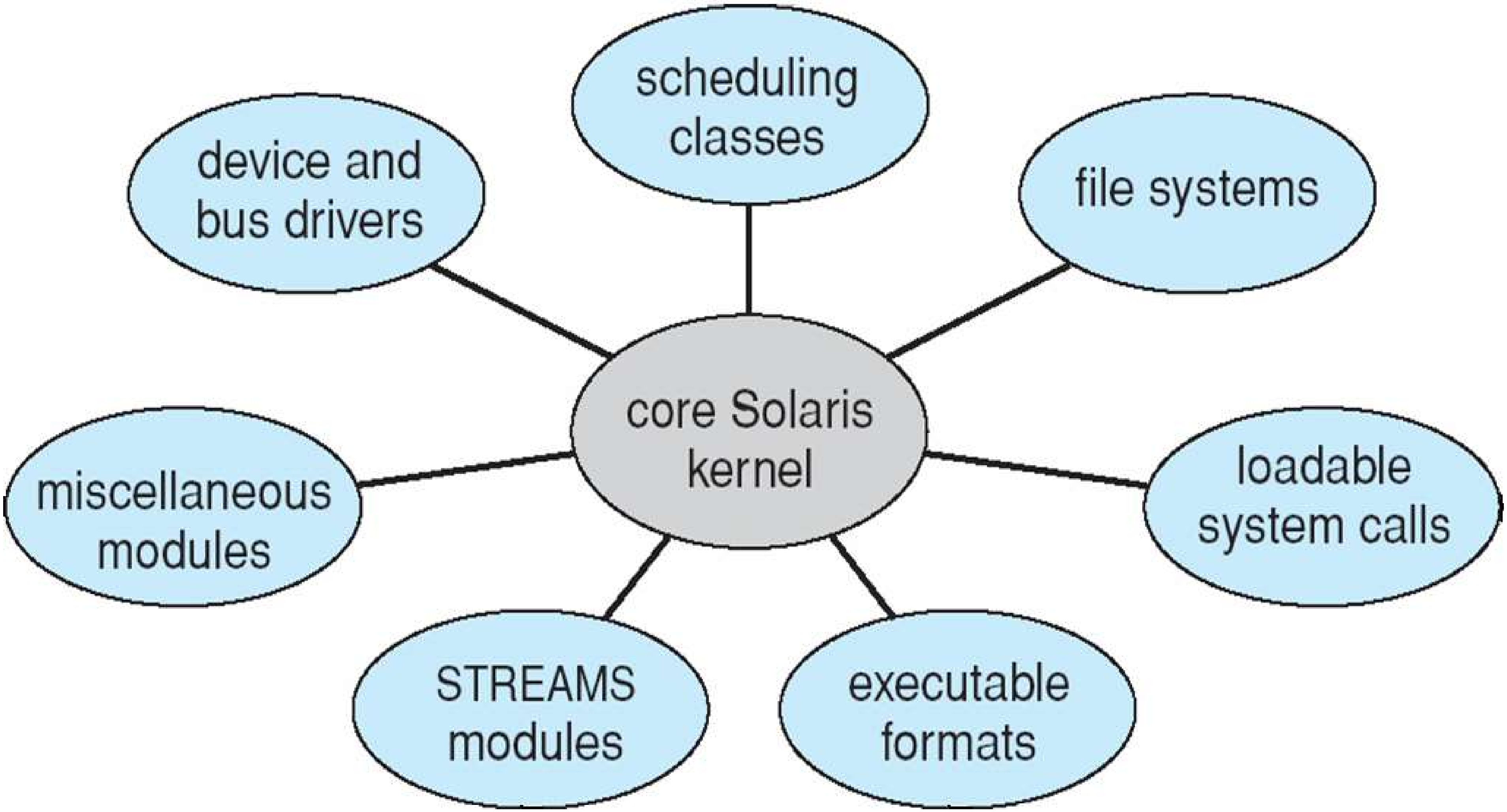


Modules

- Many 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 flexibility
  - Linux, Solaris, etc

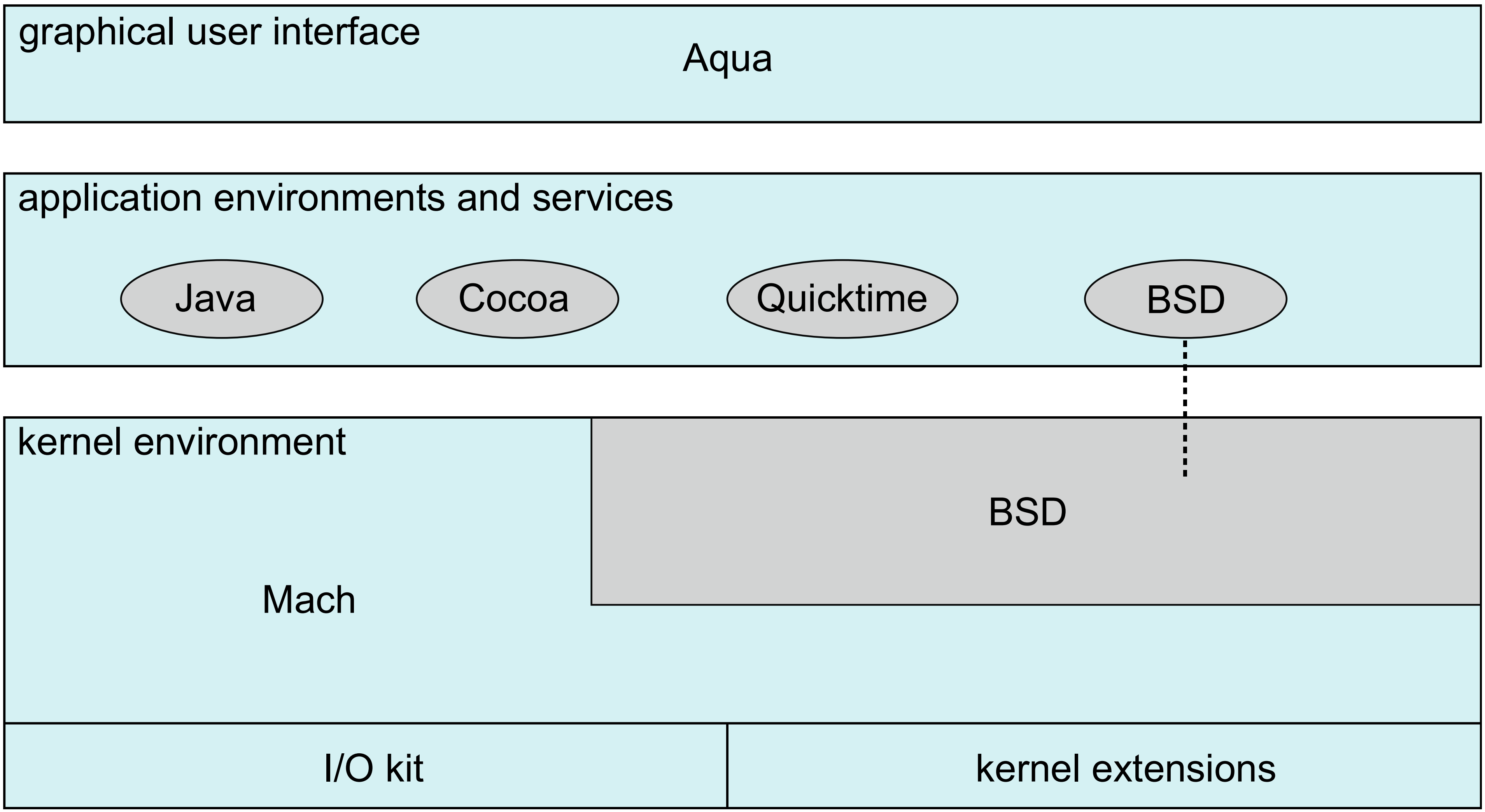
Solaris Modular Approach



Hybrid Systems

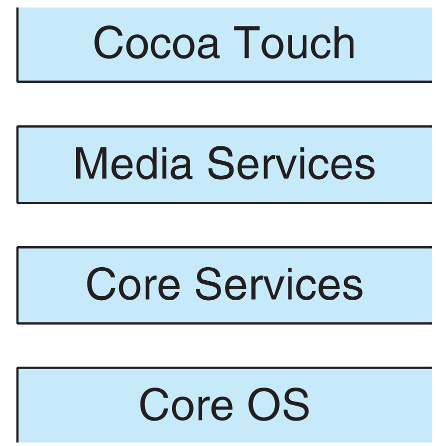
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels are in kernel address space, so they are monolithic, but also modular for dynamic loading of functionality
  - Apple Mac OS X combines a layered approach with a kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules

Mac OS X Structure



iOS

- Based on Mac OS X
  - 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



Android

- Open Source OS Developed by Open Handset Alliance (mostly Google)

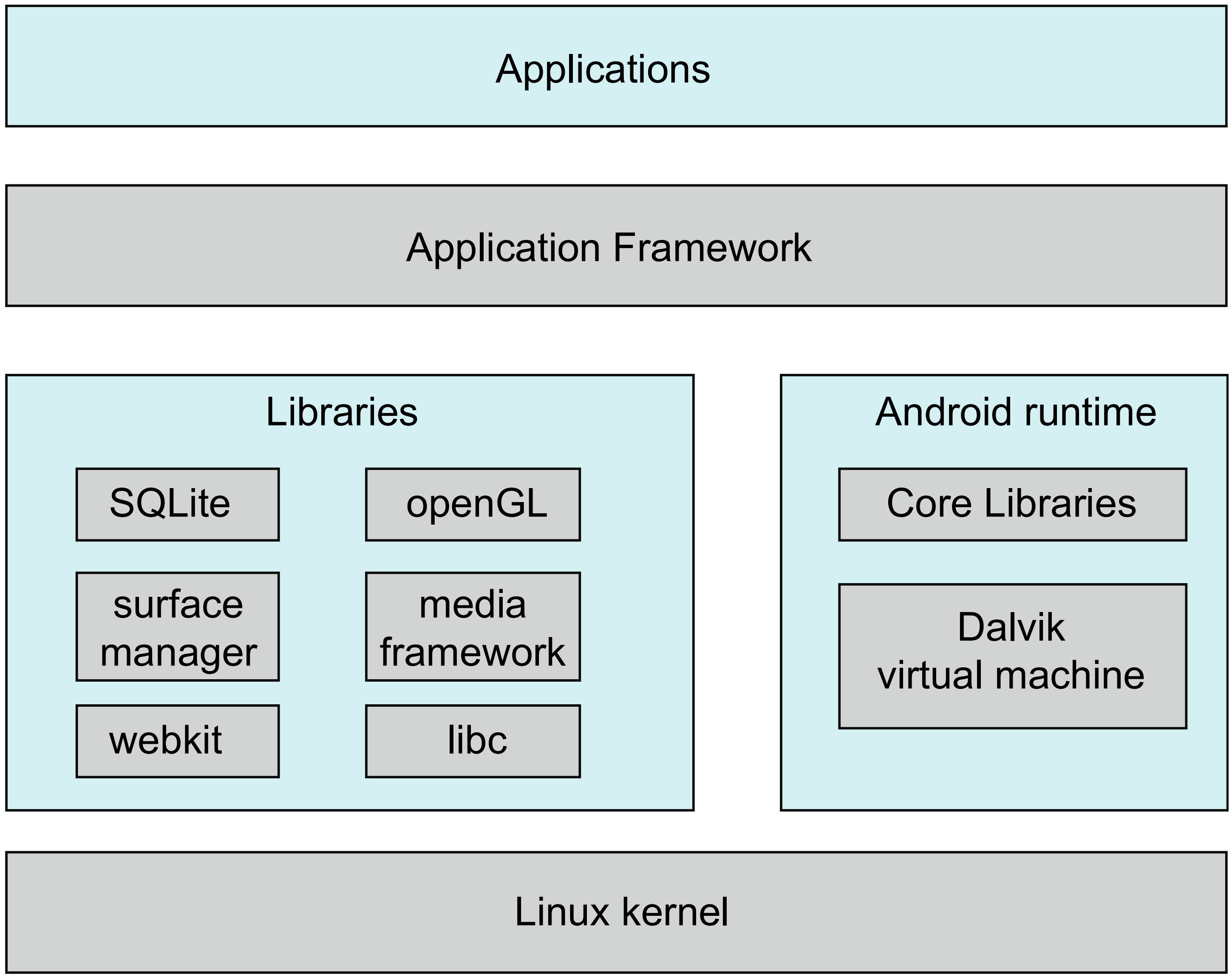
- Similar stack to iOS

- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management

Android

- Runtime environment includes core set of libraries and Dalvik/ART(2015) virtual machine
  - Apps developed in Java plus Android API
  - Java class files compiled to Java bytecode and then translated to executable and then runs in Dalvik VM

Android Architecture



That's all, folks!

- Questions?