Operating Systems Fundamentals

Operating System Structures

Operating System Structure

- A general purpose operating system is a very large program

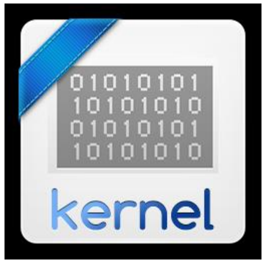
- So what is the best way to structure such a program?
  - Is it good to have all functionality available to everyone/every program?

- Operating Systems typically provide some level of separation of user applications and control of the underlying hardware

Modes of Operation

- Operating Systems provide different modes of operation for running application code
- These are:
  - User Mode

  - Kernel Mode

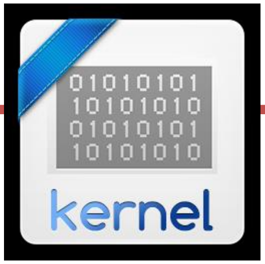


- These help support security and protection functionality
  - Ensures code is run only in an appropriate mode

User Mode

- User Mode
  - Can start new processes
  - Has a **restricted amount of operations** that it can perform
  - Is **highly abstract from the underlying hardware** of the computer
  - **Must exchange data with the Kernel through API's** to provide user functionality and operations
    - API = Application Programming Interfaces
  - This is the typical level your applications operate at.

Kernel Mode



- Kernel Mode
  - **Controls the underlying hardware** of the computer
  - **Supports the exchange of data between User Mode operations and hardware**
    - APIs provide User Mode applications with the means to access Kernel Mode operations
  - **Has high level privileges to change and modify the hardware structures of the computer**
    - Think of difference between an User account and an Administrator account
  - Applications run in Kernel Mode need to be trusted, secure and reliable

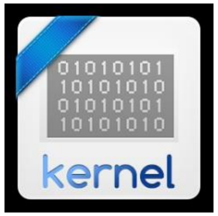
User Mode and Kernel Mode

User Mode

- User Programs execute in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected area of memory may be accessed



OS Structures

- The internal structure of different Operating Systems can vary widely
  - Monolithic
  - Microkernel
  - Modular
  - Layered

- Affected by choice of hardware, type of system and operating system goals.

OS Structures

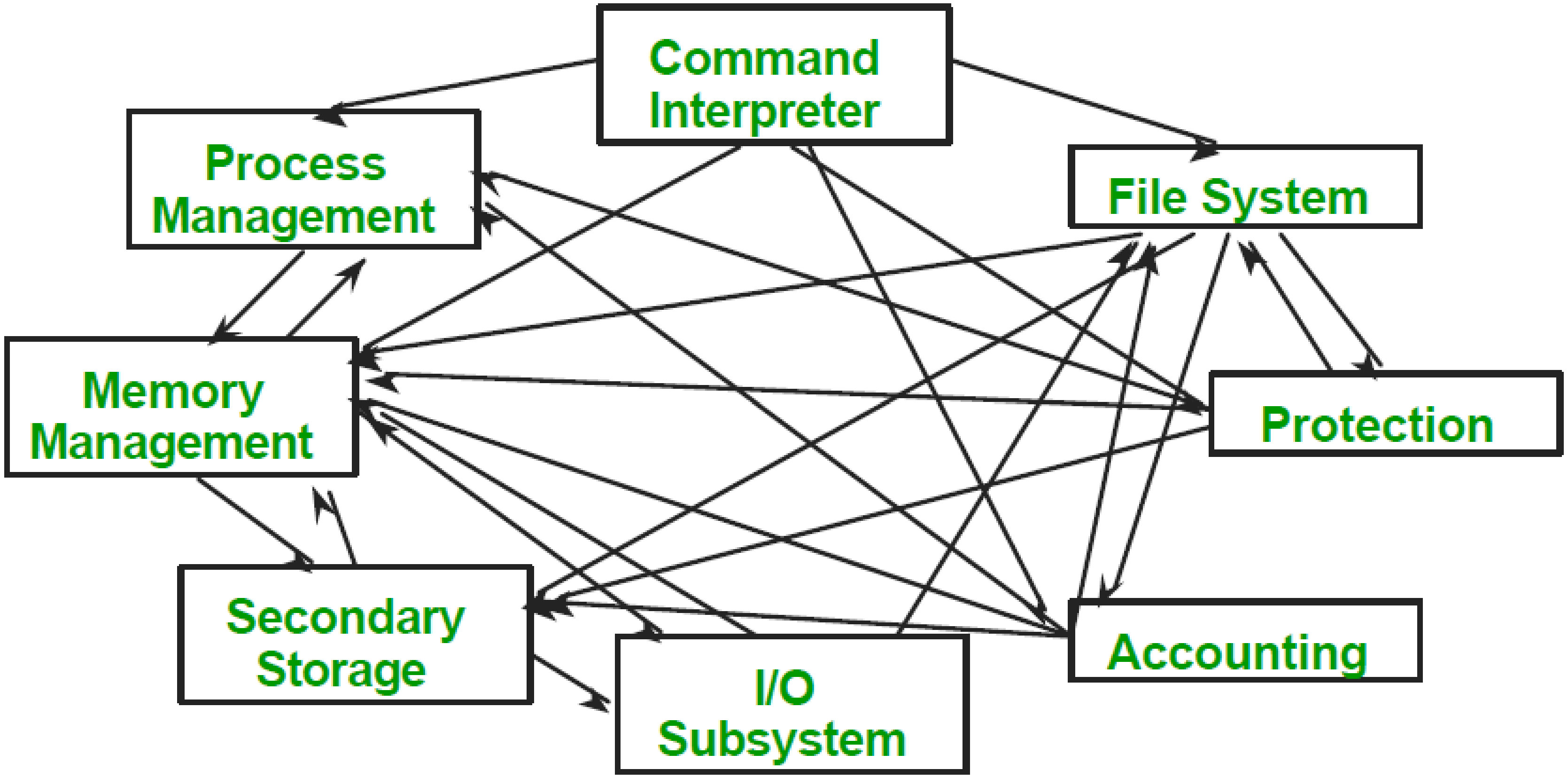
- **User goals and System goals**

  - **User goals**: Operating System should be
    - Convenient to use
    - Easy to learn
    - Reliable
    - Safe
    - Fast

  - **System goals**: Operating System should be
    - Easy to design, implement, and maintain
    - Flexible
    - Reliable
    - Error-free
    - Efficient
    - Secure

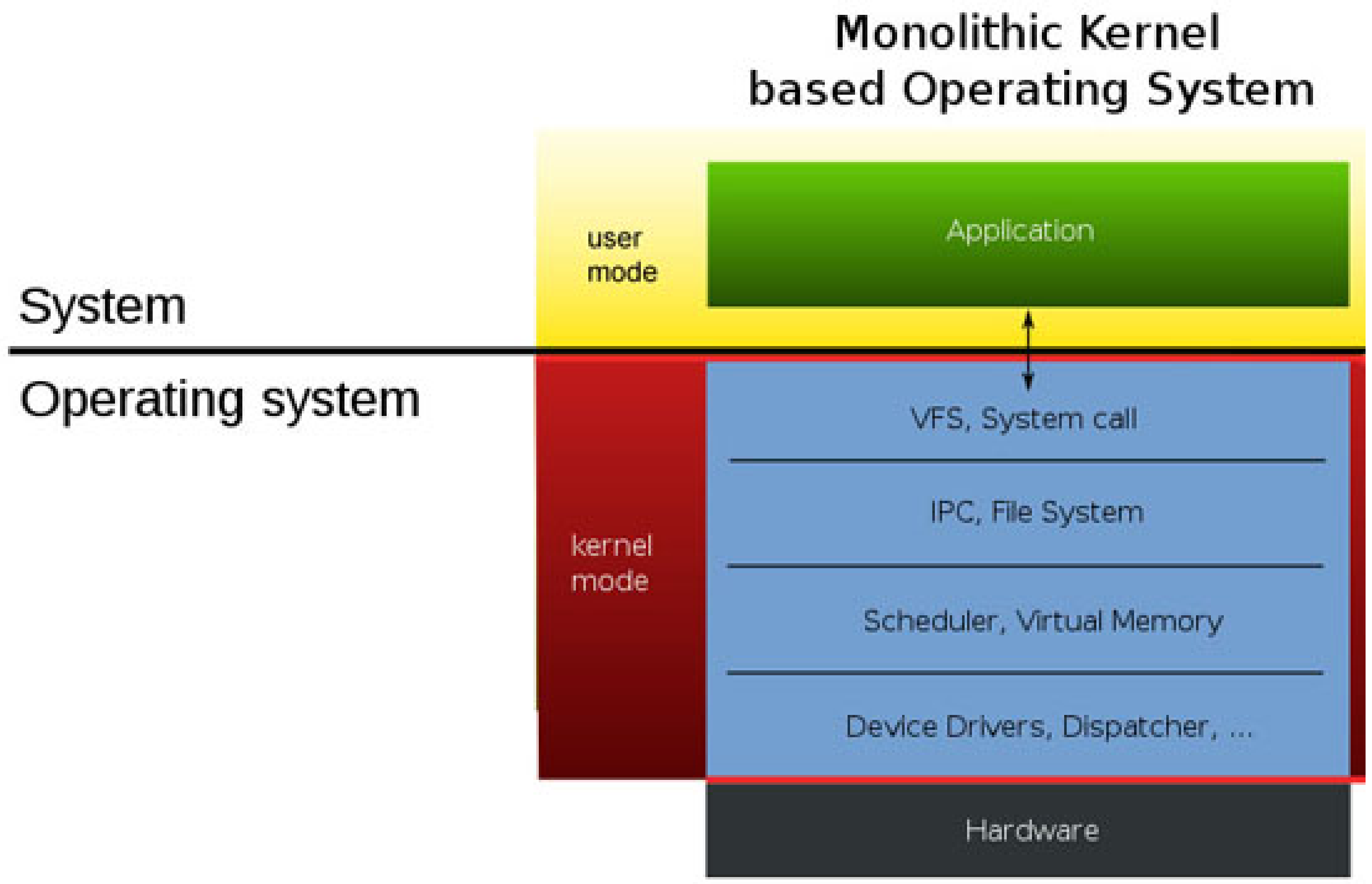
The Challenge of OS Structure

How should all the different components of an OS be connected?



We will take a look at some of the different approaches.

Monolithic

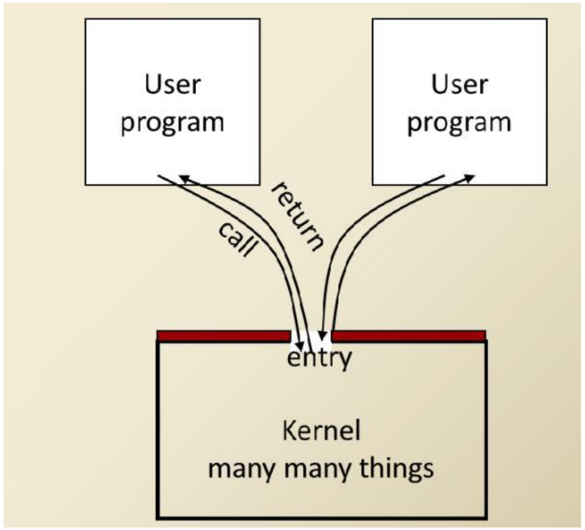

Monolithic Kernel
based Operating System
System
Operating system
user
mode
Application
kernel
mode
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware

Monolithic Systems (simple view)

- The entire operating system runs as a single program in kernel mode.

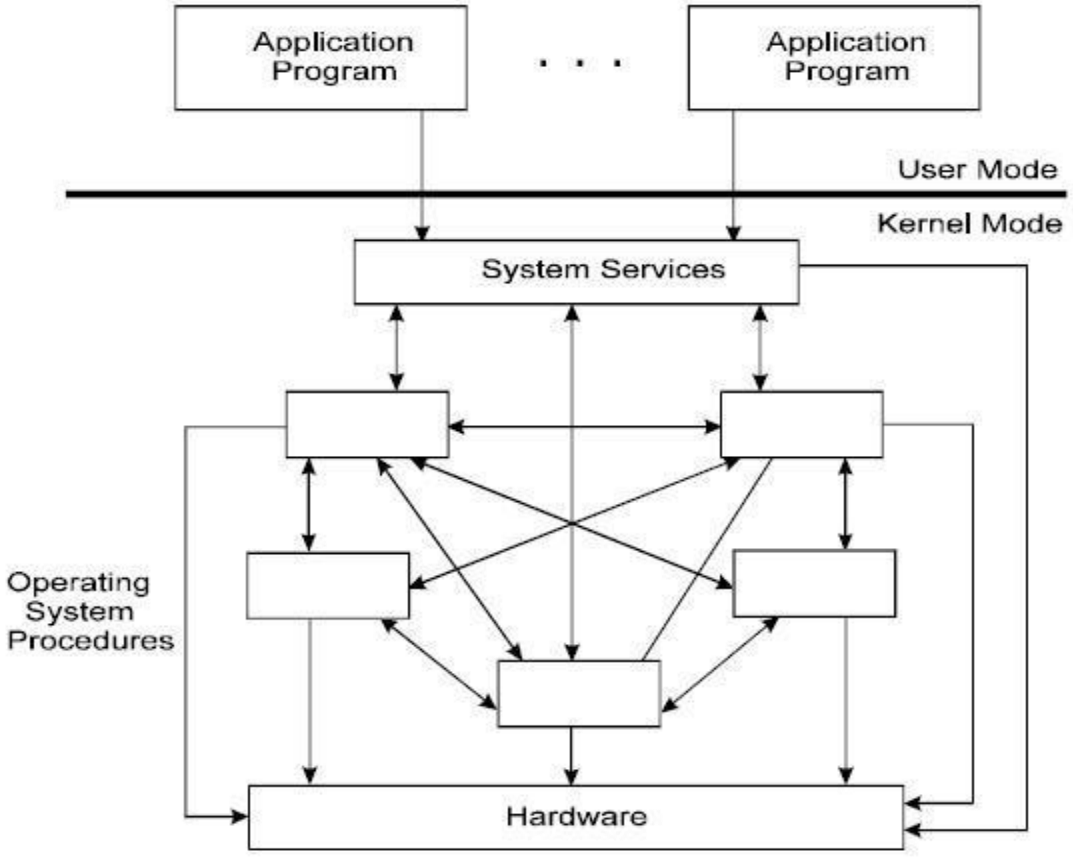
- The operating system is written as a collection of procedures. Initial versions linked these procedures together into a single large executable binary program

- Operating system whose kernel contains every component of the operating system. The kernel typically operates with unrestricted access to the computer system. Initially there were no clear division between units in the kernel.



Monolithic Systems

- Abstract view, Monolithic OS structure with separate User and Kernel mode

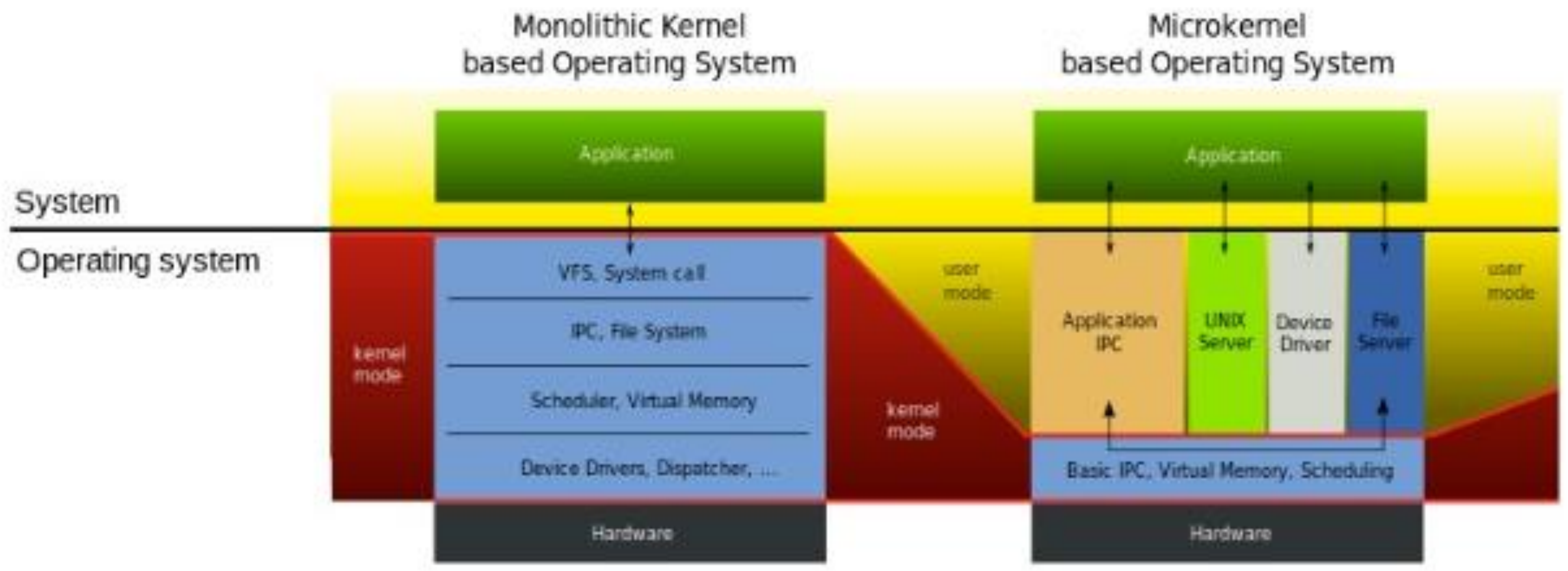


- Also called the “spaghetti nest” approach!

Monolithic Systems

- The monolithic structure **does not enforce data hiding in the OS.**
- It **delivers better application performance**,but **extending such a system can be difficult** because modifying a procedure can introduce bugs in seemingly unrelated parts of the system.
- If any aspect does not work correctly, must reset and re-implement everything to get it working again
  - Can't just change a small part and add in.
  - So… it isn't 'modular'

Micro Kernel



Microkernel Architecture

- As the original UNIX system expanded, the monolithic kernel became large and difficult to manage
- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularises the kernel using the microkernel approach (Mac OS X kernel (Darwin) partly based on Mach)

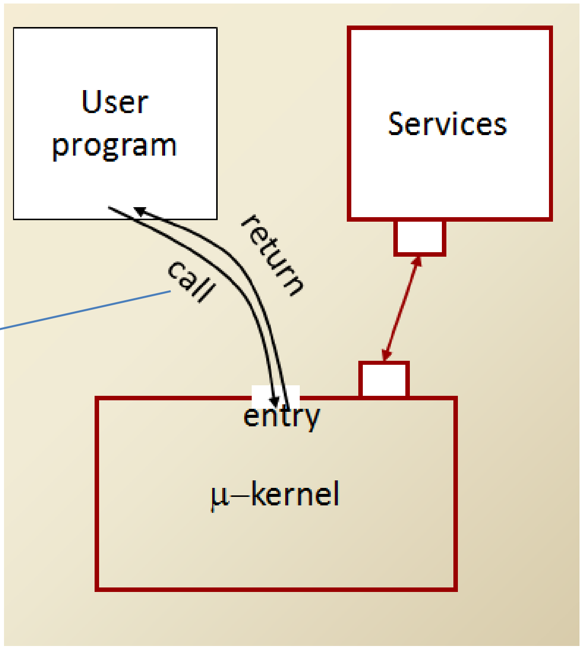
- This method structures the operating system by **removing all nonessential components from the kernel, and implementing them as system- and user-level programs**
  - The result is a smaller kernel
  - There is little consensus regarding which services should remain in the kernel and which should be implemented in user space.
  - In general, however, **microkernels typically provide minimal process and memory management, in addition to a communication facility.**

Microkernel Architecture

- Services are implemented as regular processes than run in user mode

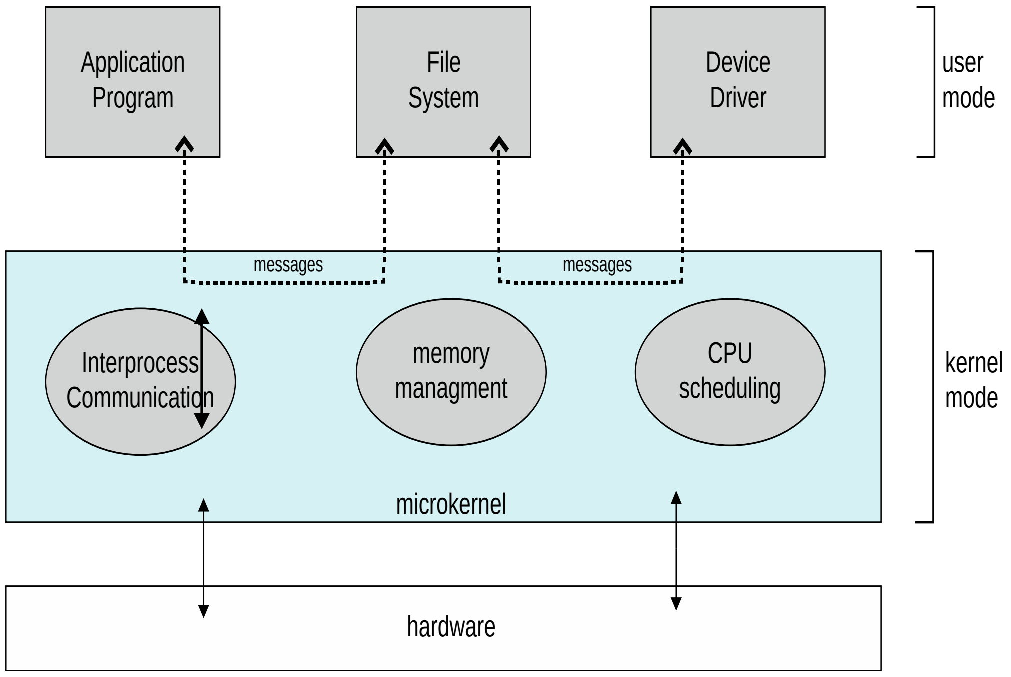
- The microkernel gets the requested services on behalf of the user

Referred to as a “System Call”



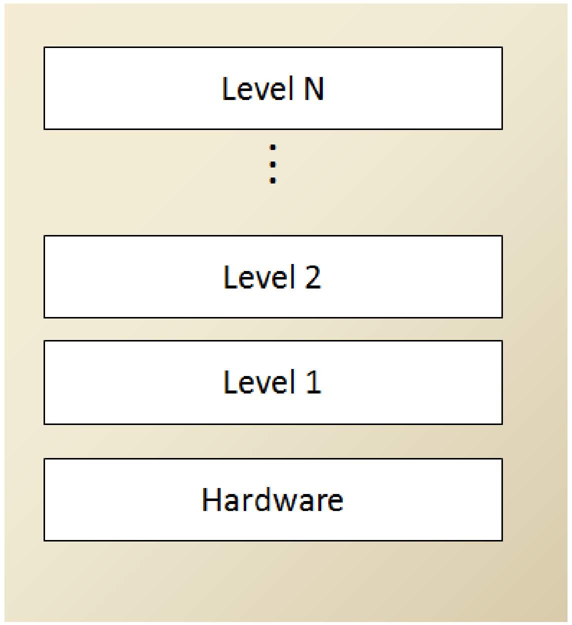
Microkernel Architecture

- **Advantages**
  - Easier to develop services
  - Fault isolation
  - Customization
  - Smaller kernel => easier to optimise
- **Disadvantages**
  - Poor performance
    - Due to lots of boundary crossings



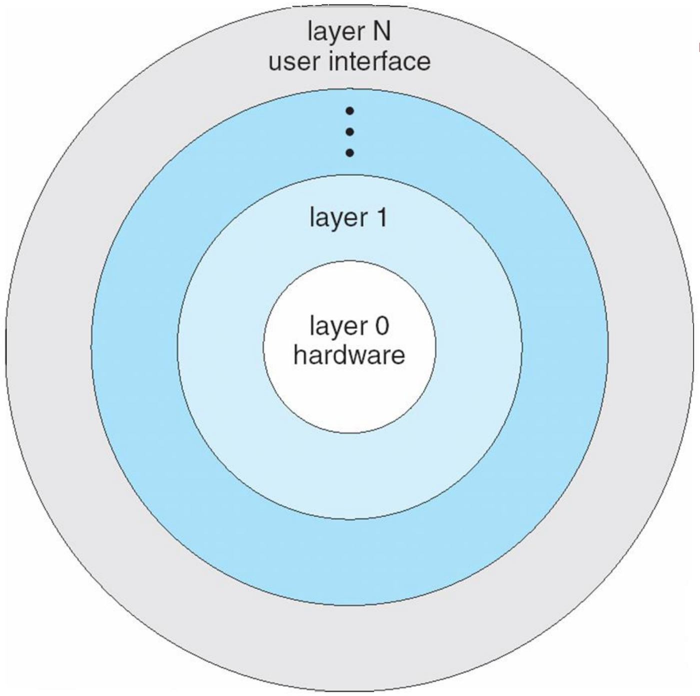
Layered Architecture

- This structure is a generalisation of the monolithic approach
- Functionality built on-top of that provided by other sections
- Functionality increased at each level (layer) based on the operations provided by the level below
- Each layer accesses the services of the layer below and returns results to the layer above.



Layered Architecture

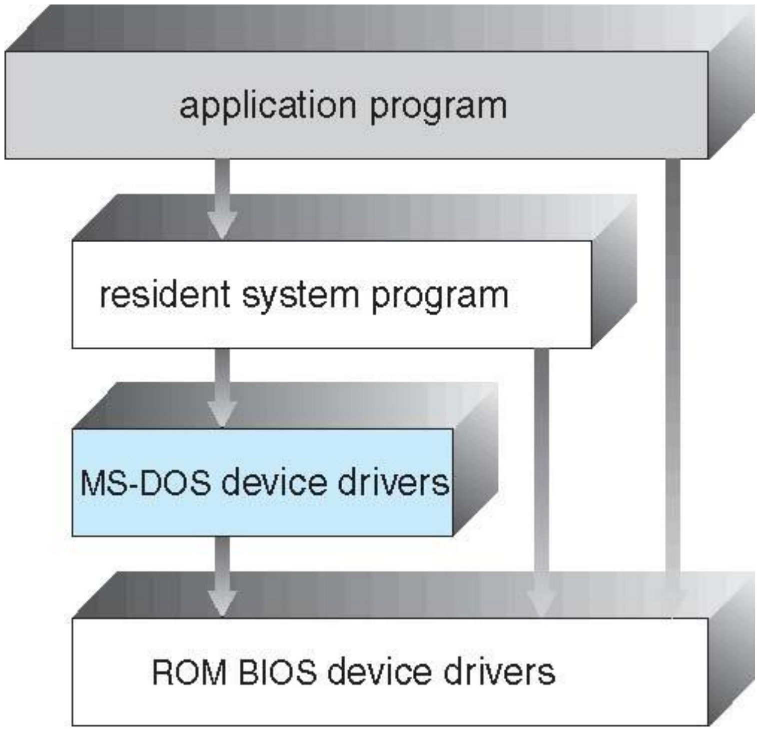
- Often visualised in layered rings
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface
- Hiding information at each layer
- Developed a layer at a time
  - Layer 1 – Processor allocation and multi-programming
  - Layer 2 – Memory management
  - Layer 3 – Devices
  - Layer 4 – File Management
  - ...
  - Layer N - Users



Organised into modules and layers built on top of one another. Each module provides a set of functions that other modules can call.

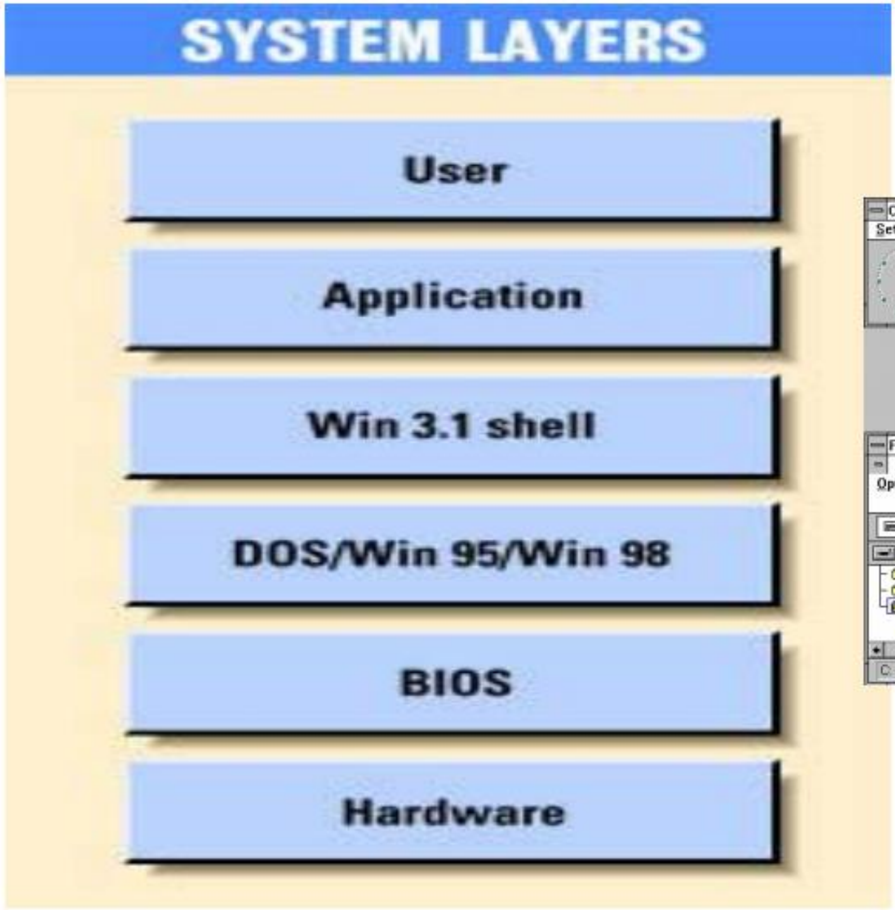
Simple Layered Architecture

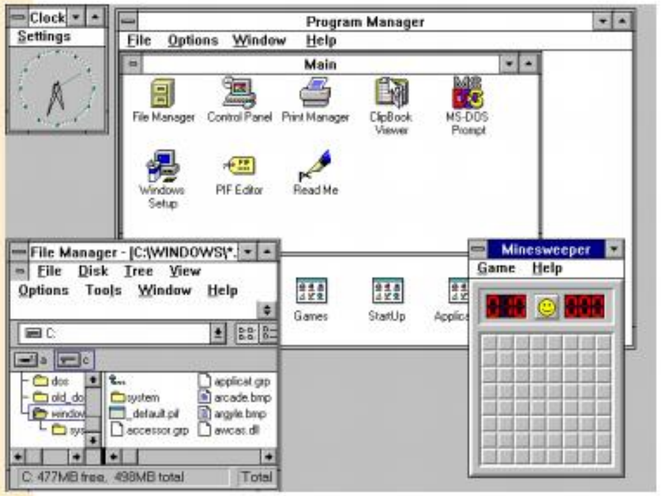
- MS-DOS written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
  - Basic hardware is accessible from high level!



Layered Architecture

Older Windows System Layers





Modular Architecture

- Takes the best features of Monolithic and Layered design approaches
- Breaks Operating System into blocks (modules) where all operations that are being performed are connected. Can easily change modules without affecting Operating System performance
- Linux is a good example of a modular operating system. It allows you to install only those OS components you really need.

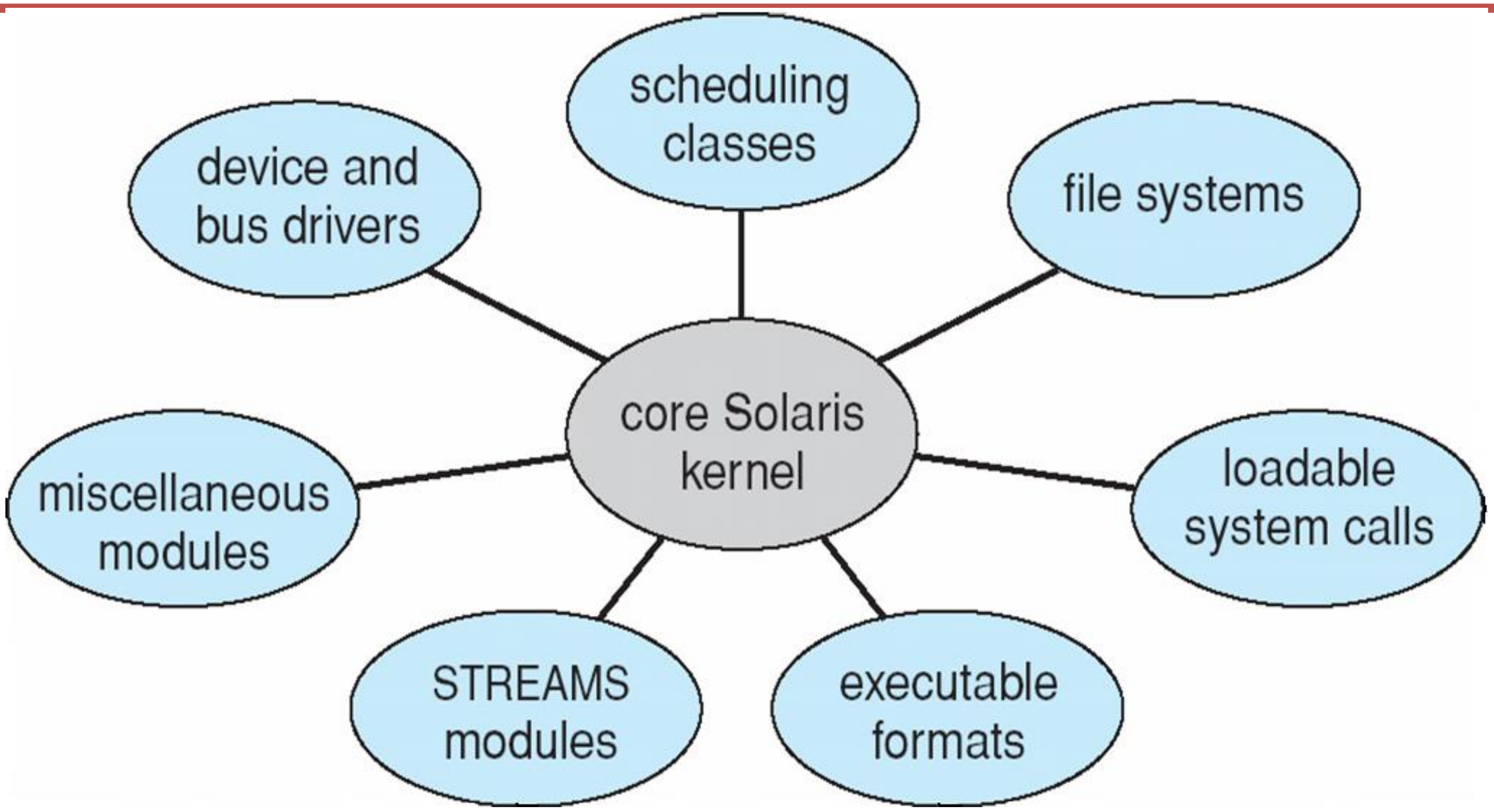
Loadable Modules

- Now used extensively in Linux to dynamically load (and unload) executable kernel modules at runtime.

- Dynamically loading modules allows kernel functionality to be changed
  - At run time
  - Without recompiling/deploying the whole new kernel
  - Without rebooting the device for the change to take affect

- Loadable modules allow modern embedded devices (which may be memory limited) to only have to have the minimal amount of kernel content loaded.
  - i.e. we don’t load the kernel modules we don’t need

Solaris Modular Approach



Solaris was created by the company ‘Sun’, now acquired by ‘Oracle’

Architectures Compared

- **Monolithic**
  - Quick
  - Single Structure
  - Hard to maintain
  - Bloated
- **Microkernel**
  - Easier to develop services
  - Fault isolation
  - Customization
  - Smaller kernel => easier to optimize
  - Lots of boundary crossings
  - Poor performance
- **Layered**
  - Built on existing operations
  - Defined interface (API) between layers
  - Slow
  - Difficult to change lower layers as functionality must stay the same
- **Modular**
  - Easy to add functionality
  - Easily reset/restart OS services
  - Additional code to exchange information between modules

A Closer look at

```
bash# where there is a shell, there is a way
```

Unix Design

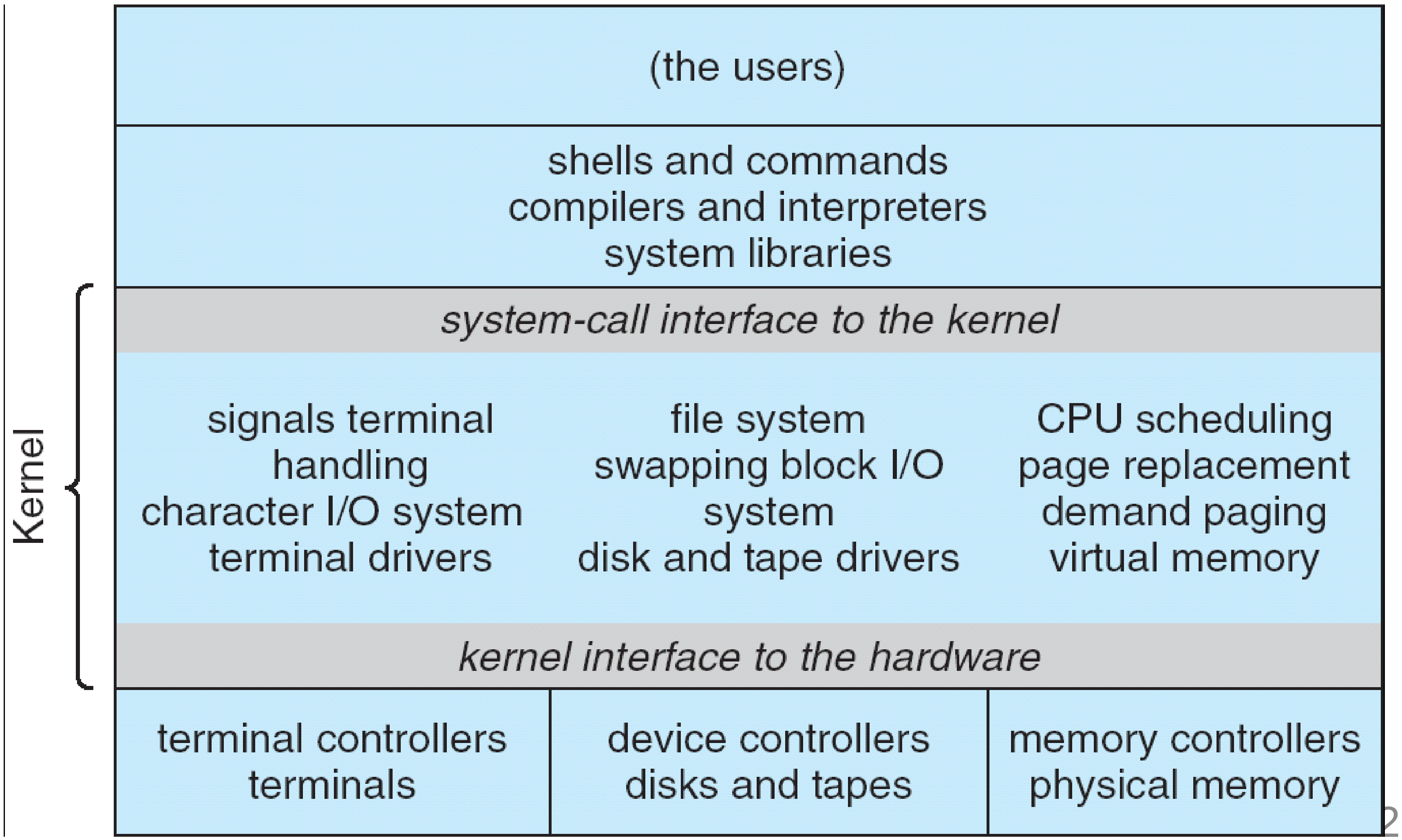
- The UNIX OS consists of two separable parts

1. Systems programs (shells and utilities)
   - The shell is the outermost part of an operating system and a program that interacts with user commands.
   - The utilities, for example, are the editors, filters, file manipulation tools
2. The Kernel
   - Everything below the system-call interface and above the physical hardware
   - It provides the
     - File system, CPU scheduling, Memory management and other operating-system functions

Unix – The *Kernel*

- The **kernel (nucleus**) is the essential centre of the operating system that provides basic services for all other parts of the operating system. typically including
  - memory management, process management, file management and I/O (input/output) management (i.e., accessing the peripheral devices).
  - These services are requested by other parts of the operating system or by application programs through a specified set of program interfaces referred to as system calls.
- The kernel itself does not interact directly with the user, but rather interacts with the shell and other programs, as well as with the hardware devices on the system, including the processor, memory and disk drives.
- The kernel is the first part of the operating system to load into memory during booting and it remains there for the entire duration of the computer session because its services are required continuously.
- Because of its critical nature, the kernel code is usually loaded into a **protected** area of memory, which prevents it from being overwritten by other, less frequently used parts of the operating system or by application programs.

More Detailed Unix Structure



A Closer look at

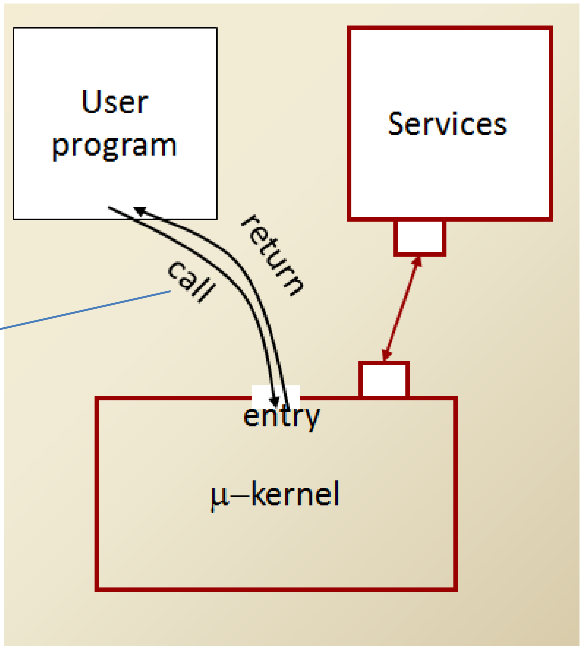
1993 Windows NT 3.1, NT = New Technology
The design forms the basis of Windows releases since then

Reminder - Microkernel Architecture

- Services are implemented as regular processes than run in user mode

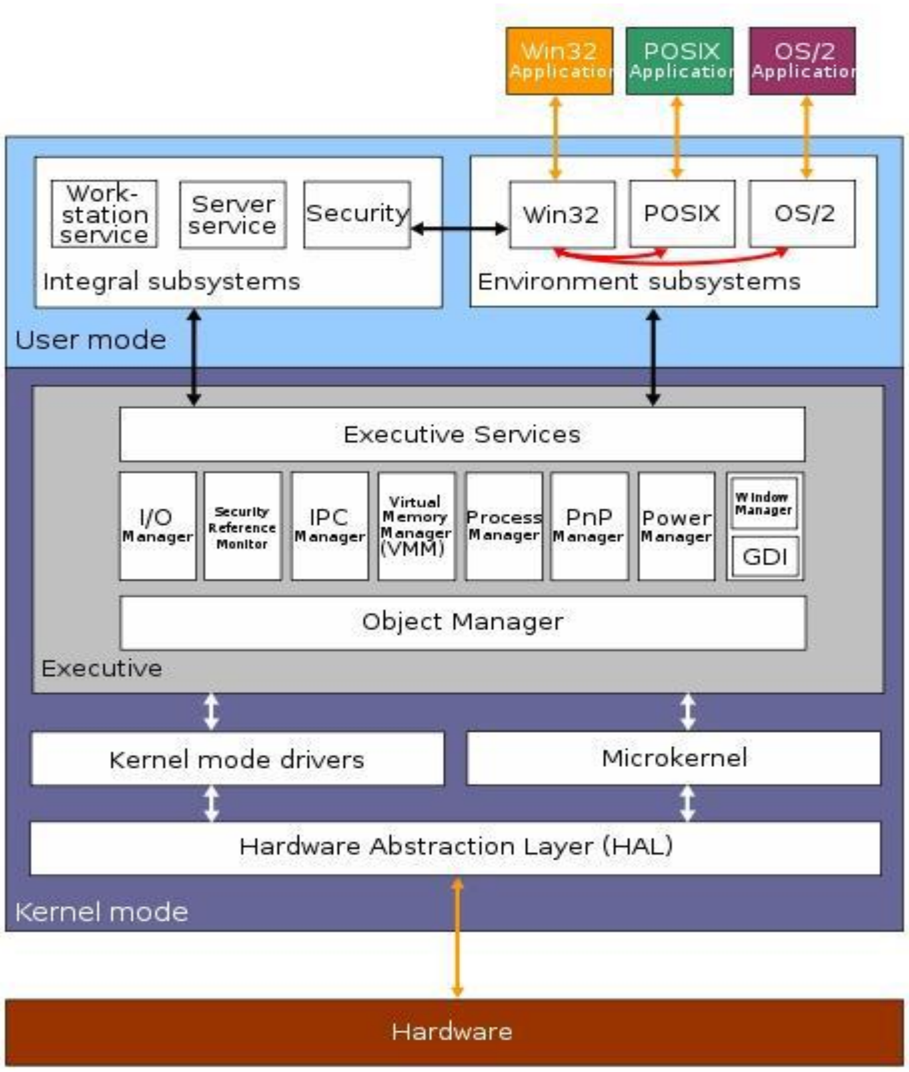
- The microkernel gets the requested services on behalf of the user

Referred to as a "System Call"



Windows NT

- A layered design.
- Modified microkernel design
  Executive not in user space
- Supports two levels of operations
  - User Mode
  - Kernel Mode
- Both isolated from each other

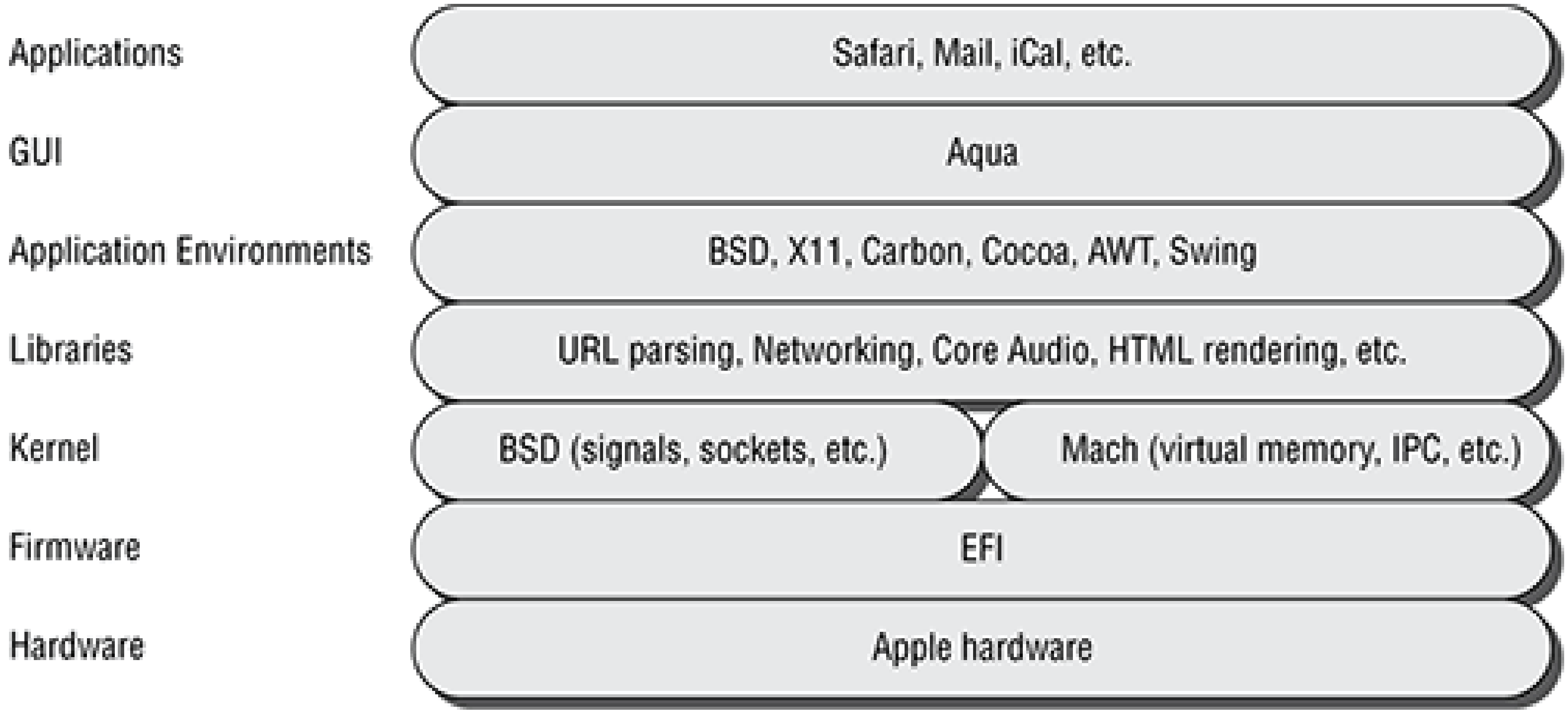


Windows NT since 1993

Windows NT 3.1	(1993)
Windows NT 3.5	(1994)
Windows NT 3.51	(1995)
Windows NT 4	(1996)
Windows NT 5.0 (Windows 2000)	(1997-1999)
Windows NT 5.1 (Windows XP)	(2001)
Windows NT 5.2 (Windows Server 2003, Windows XP x64)	(2003)
Windows NT 6.0 (Windows Vista, Windows Server 2008)	(2006)
Windows NT 6.1 (Windows 7, Windows Server 2008 R2)	(2009)
Windows NT 6.2 (Windows 8, Windows Server 2012)	(2012)
Windows NT 6.3 (Windows 8.1, Windows Server 2012 R2)	(2013)
Windows NT 10.0 (Windows 10)	(2015)
Windows NT 11.0 (Windows 11)	(2021)

A Closer look at

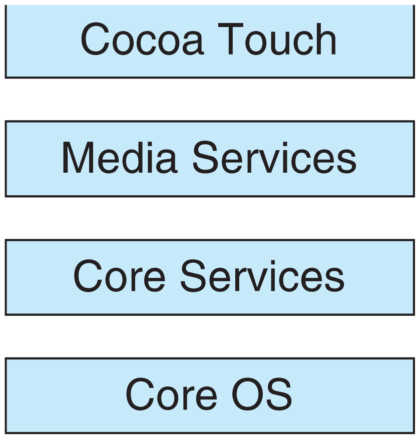
Mac OS X Structure


Applications
Safari, Mail, iCal, etc.
GUI
Aqua
Application Environments
BSD, X11, Carbon, Cocoa, AWT, Swing
Libraries
URL parsing, Networking, Core Audio, HTML rendering, etc.
Kernel
BSD (signals, sockets, etc.)
Mach (virtual memory, IPC, etc.)
Firmware
EFI
Hardware
Apple hardware

BSD = Berkeley Software Distribution
(sometimes called Berkeley Unix)

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 (**Swift UI** is still incomplete???)
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel

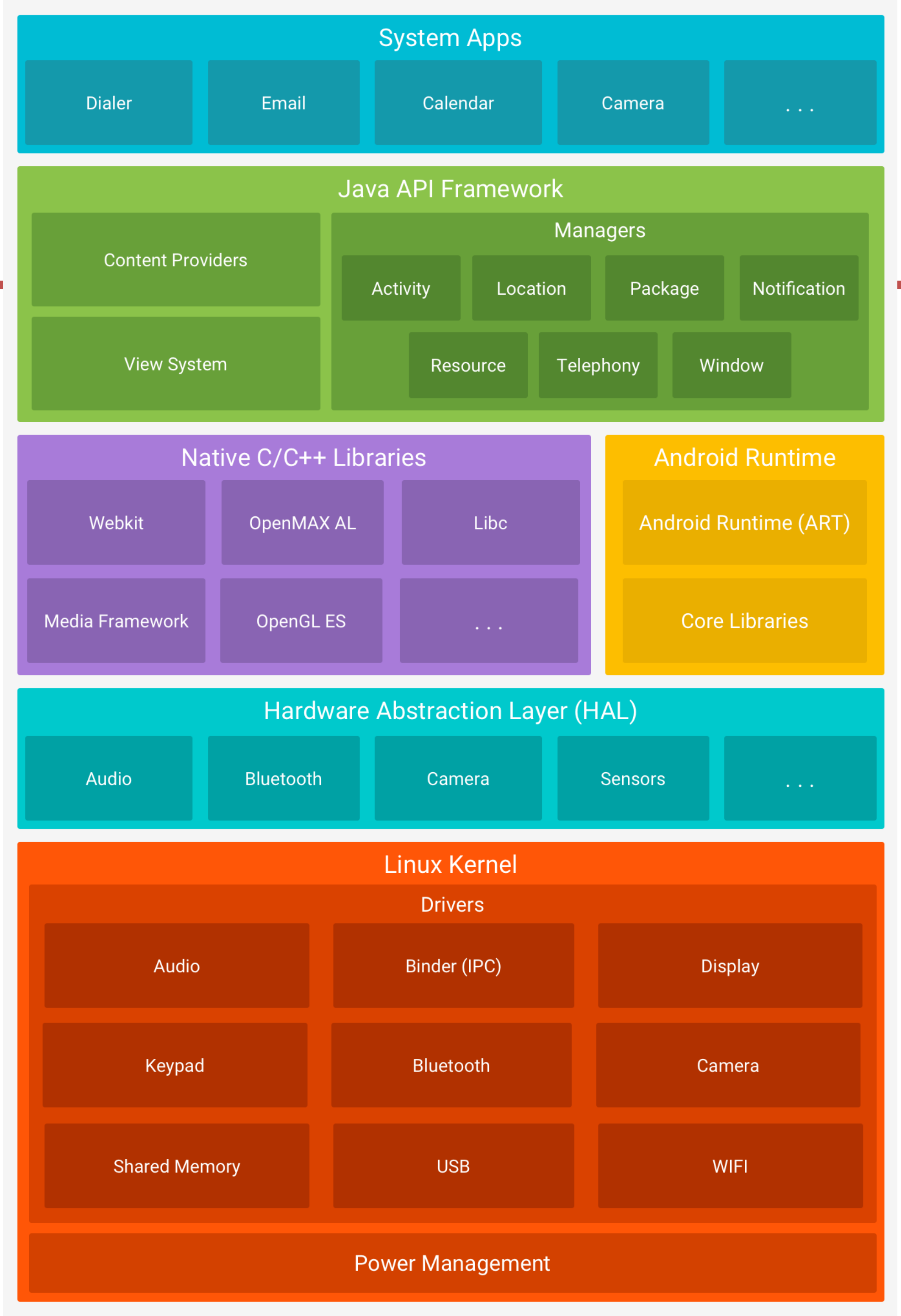


A Closer look at

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
  - Essentially a ‘distribution’ of Linux
- Android Runtime (ART) environment includes core set of libraries (replacing the Dalvik virtual machine)
  - Apps developed in Java plus Android API
    - It converts the applications bytecode into native instructions.
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



Where does Linux fit in?

Android Version	Linux Kernel
2.0/1	2.6.29
2.2.x	2.6.32
2.3.x	2.6.35
3.x.x	2.6.36
4.0.x	3.0.1
4.1.x	3.0.31
4.2.x	3.4.0
4.3	3.4.39
4.4	3.10
5.X	3.16.1
6.0	3.18.10
7.0	4.4
8.0	4.10
9.0	4.15
10.0	4.19
11.0	5.4
12.0	5.10
13.0	5.15
14.0	6.1

Latest OS Versions
as of 13th February 2024

- Microsoft Windows 11
  - Release date: 5th October 2021
- Linux Kernel
  - Version 6.1
  - Release date: 11th December 2022
- MacOS Sonoma
  - Version 14
  - Release date: 26th September 2023
- Android
  - Version 14 Codename Upside Down Cake
  - Release date: 4th October 2023
- iOS
  - Version 17
  - Release date: 18th September 2023