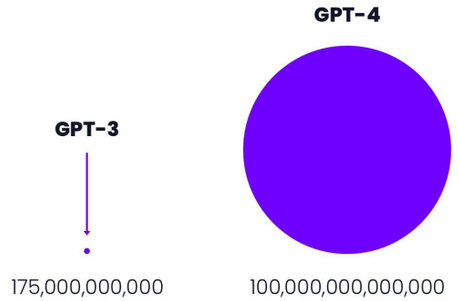
Introduction to Operation Systems

2023-24 COMP3230A

To learn or not to learn, that is the question.

- We already have ChatGPT, why do we still learn OS???
- What made ChatGPT successful?



- What made ChatGPT successful: An OS perspective
  - "GPT-4 was trained on about 25,000 Nvidia A100 GPUs for 90–100 days"
  - Resource management, scalability, reliability, inter-process communication, security, and many more…

To learn or not to learn, that is the question.

- (OS) questions to ask about GPT?
  - And to find answers through this course!

- Example: How to efficiently allocate and manage hardware resources, like CPU, GPU, memory, and storage?
- More?

Learning Outcome

- ILO 1 – [**Fundamentals**] discuss the characteristics of different structures of the Operating Systems (such as microkernel, layered, virtualization, etc.) and identify the core functions of the Operating Systems.

Contents

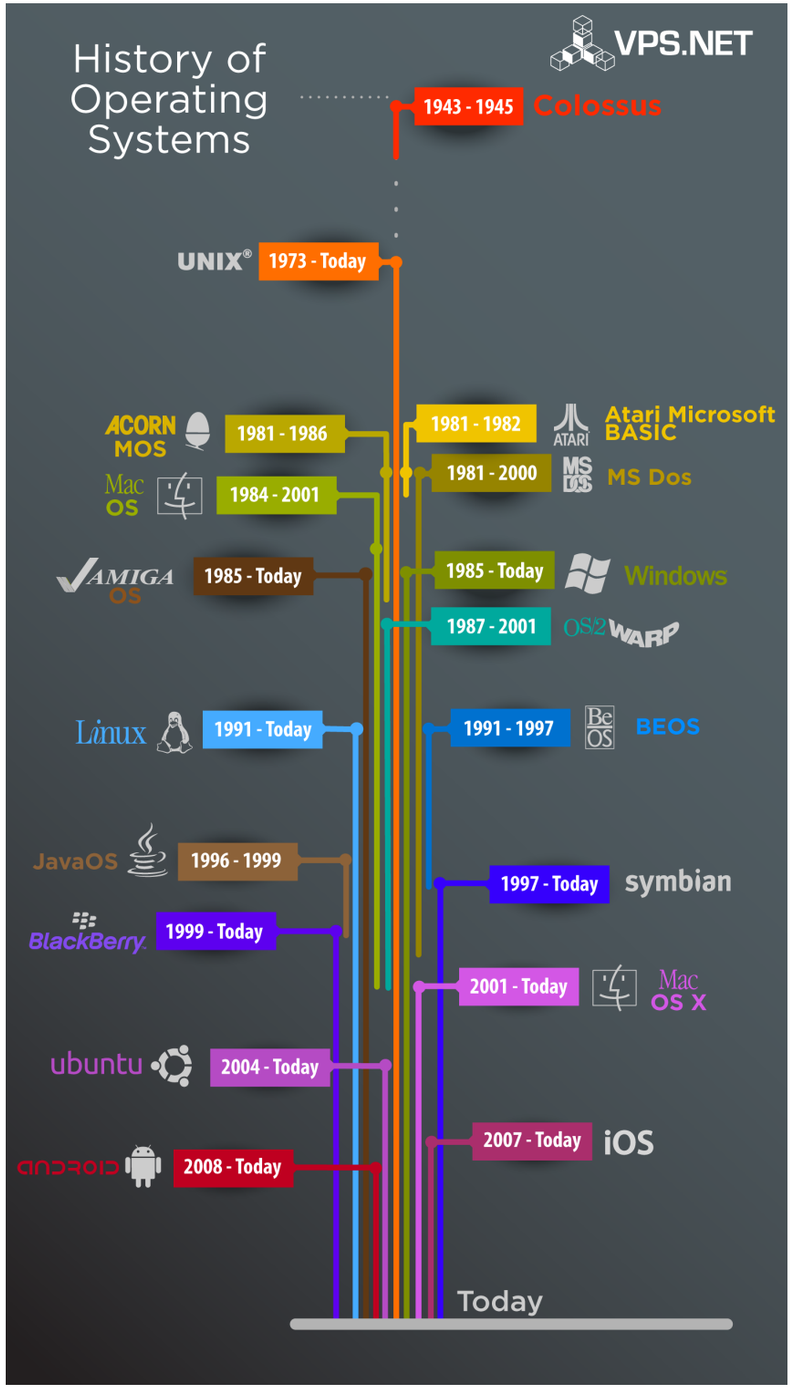
- Purpose and Functions of Operating Systems
- Core Services
- OS Architectures (Implementation design)

Reading & Reference

- Required Readings
  - Chapter 2, Introduction to Operating Systems, Operating Systems: Three Easy Pieces by Arpaci-Dusseau et. al
    - http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf
  - Section 1.10 to 1.13 of Chapter 1 of Operating Systems, 3rd edition by Deitel et. al
    - http://www.deitel.com/books/os3e/os3e_01.pdf

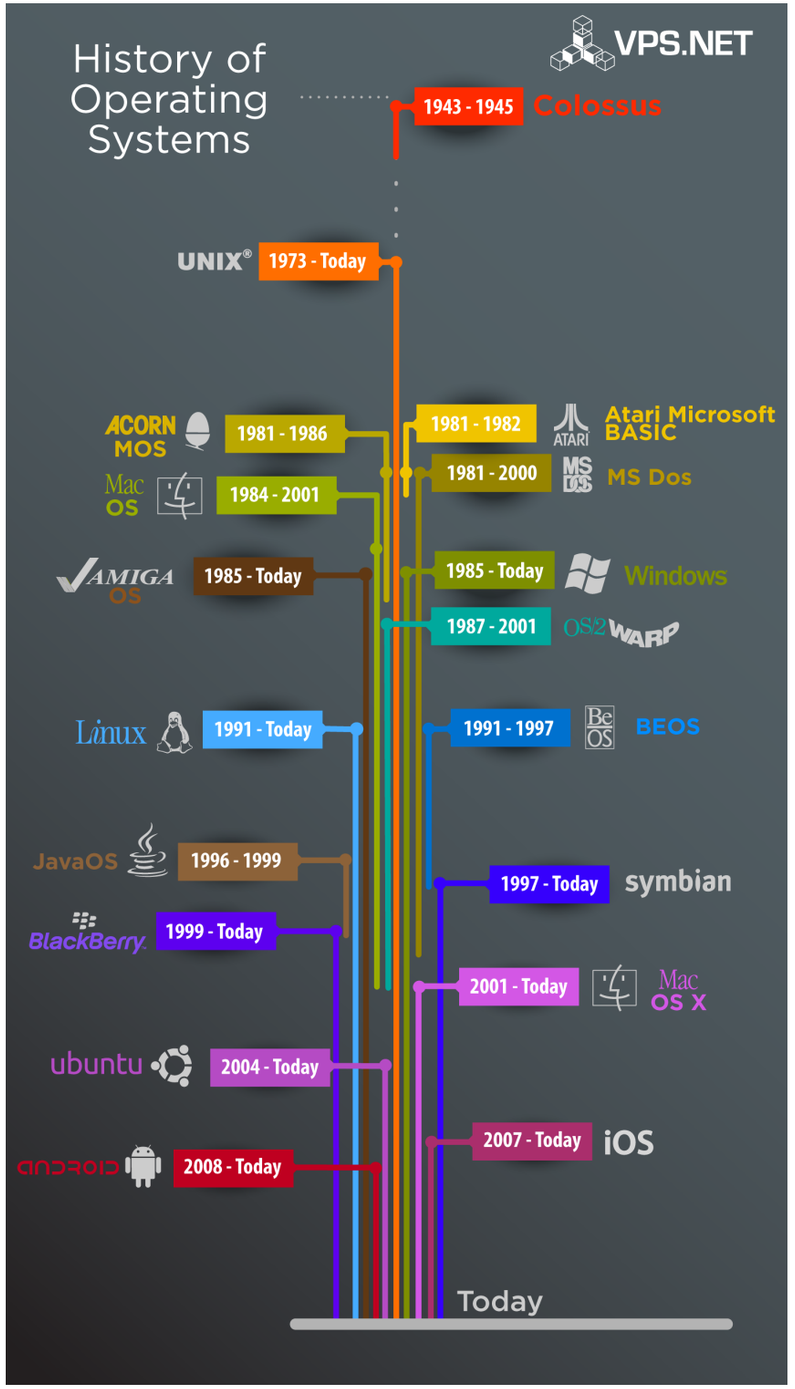
A Brief History of OS

- mid 1950s – mid 1960s (mainframe): **Batch Systems**
- 1960s (minicomputer): **Multiprogramming Systems** -- UNIX born (Ken Thompson & Dennis Ritchie)
- 1980s (PC):
  - **MS-DOS** (1981), **SunOS** (1982), ULTRIX (1982), **Macintosh** (1984), **Windows 1.0** (1985), **IBM AIX** (1986), **HP HP-UX** (1986), **Minix** (1987), **SGI IRIX** (1987), **Windows 2.0** (1987), **NeXTSTEP** (1989)
- 1990s (Internet):
  - **OS support for networking tasks became standard.**
  - **Windows 3.0** (1990), 3.1 (1993), **Linux (1991, Linus Torvalds), Solaris 2.0** (1992), **FreeBSD** (1993), **NetBSD** (1993), **OS/2**, **Windows NT 3.1** (1993), **Windows 95**, **Windows NT 4.0** (1996), **Windows 98**.



A Brief History of OS

- 2000s and beyond (Mobile)
  - Mac OS X, Windows 2000, Windows Server 2003, Windows ME, Windows XP (2001), Windows Vista (2005).
- Android: Nov. 5, 2007
  - Mobile operating system based on the Linux kernel.
- iPhone OS (iOS): March 6, 2008
  - For iPhone, iPad, iPod
  - iOS 6: Sept 19, 2012
  - iOS 8.4.1: Aug. 13, 2015 (800 MB)
  - iOS 14.7.1 : July 26, 2021
- Microsoft Windows:
  - Windows 7 : July 22, 2009, NT kernel
  - Windows 8: Oct. 26, 2012, NT kernel
  - Windows 10 : July 29, 2015, NT kernel
  - Windows 10 Mobile: Dec. 08, 2015, NT kernel
  - Windows 11
- IoT OS

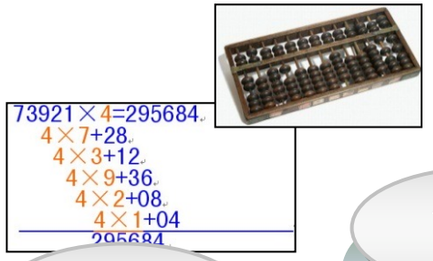


A Brief History of OS – Your History?

- What OSs have you ever used?
- What OSs have your heard about (but never used)?
- *What OSs would be created or developed by you?*

Computing as Science

Computing infrastructure



1946

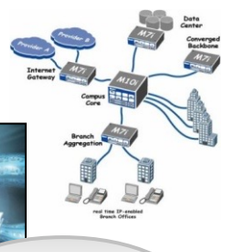
First Computer

1965

High Performance Computing

1980

Personal Computer



1995

Internet

2010

Smart Planet/ CPS/Intern et of Things

2020

AIoT

Omnipresent connectivity & sensing

Science Computing

Internet Service

AI Service

OS is the core

NextG OS

- Mark D. Weiser (1952 –1999)
- "Father of Ubiquitous Computing"
- The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.
  - The Computer for the 21st Century, 1991

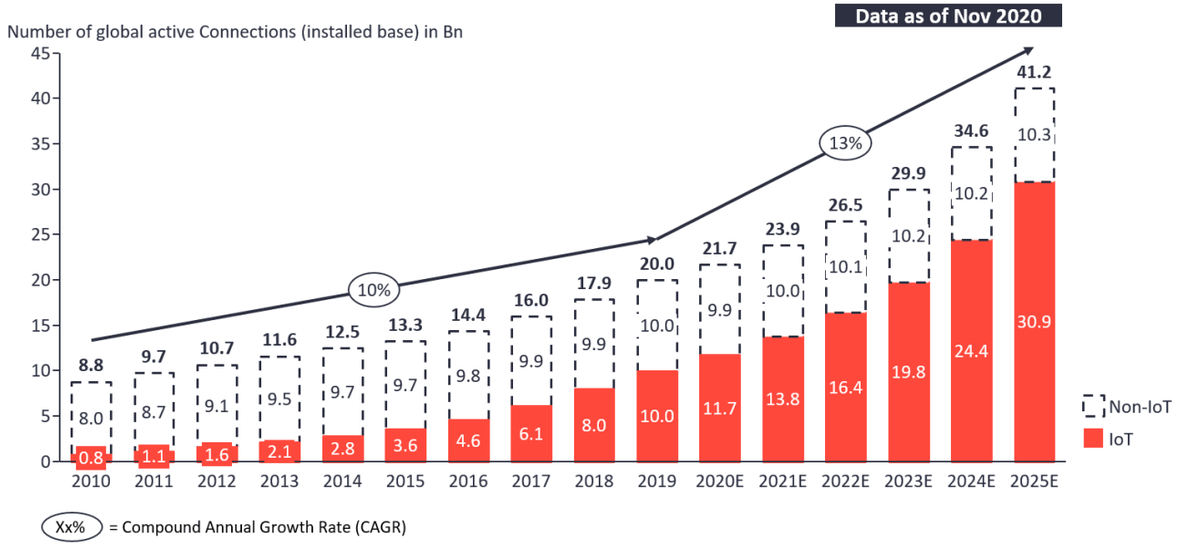
Internet of Everything

IOT ANALYTICS

Insights that empower you to under

Total number of device connections (incl. Non-IoT)

20.0Bn in 2019– expected to grow 13% to 41.2Bn in 2025



Note: Non-IoT includes all mobile phones, tablets, PCs, laptops, and fixed line phones. IoT includes all consumer and B2B devices connected – see IoT break-down for further details

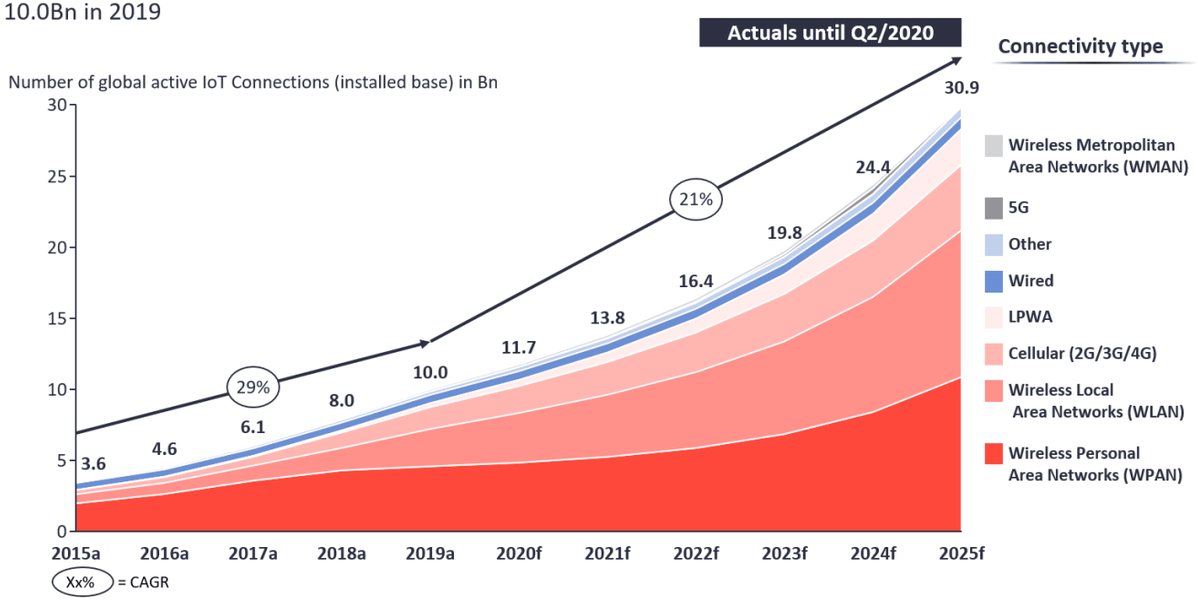
Source(s): IoT Analytics - Cellular IoT & LPWA Connectivity Market Tracker 2010-25

IOT ANALYTICS

Insights that empow

Global Number of Connected IoT Devices

10.0Bn in 2019

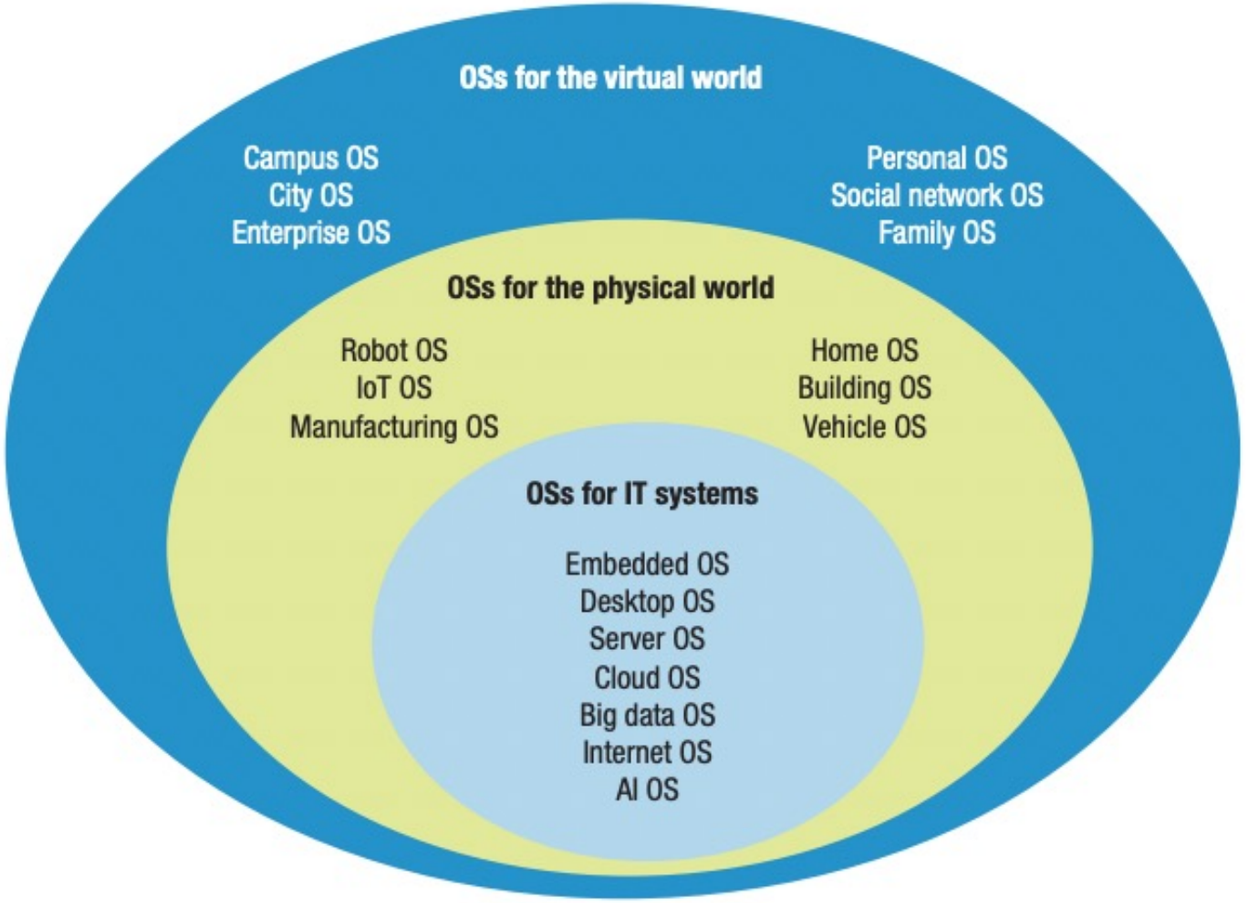


Note: IoT Connections do not include any computers, laptops, fixed phones, cellphones or tablets. Counted are active nodes/devices or gateways that concentrate the end-sensors, not every sensor/actuator. Simple one-directi considered (e.g., RFID, NFC). Wired includes Ethernet and Fieldbuses (e.g., connected industrial PLCs or I/O modules); Cellular includes 2G, 3G, 4G; LPWAN includes unlicensed and licensed low-power networks; WPAN includes includes Wi-fi and related protocols; WMAN includes non-short range mesh, such as Wi-SUN; Other includes satellite and unclassified proprietary networks with any range.

Source(s): IoT Analytics - Cellular IoT & LPWA Connectivity Market Tracker 2010-25

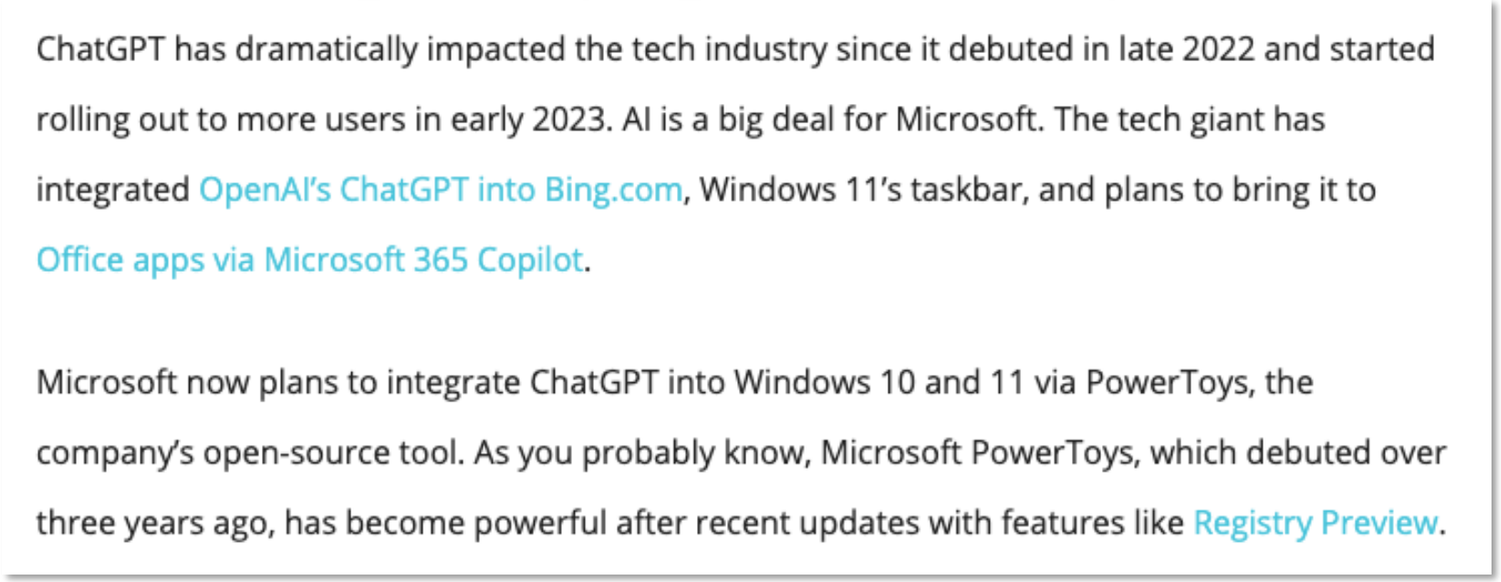
Ubiquitous OS

- Google Fuchsia OS
- Microsoft HomeOS (2010!)
- Apple homeOS
- vw.os
- AliOS
- XiUOS
- HarmonyOS
- FreeRTOS, RIOT, Zephyr, TinyOS, Contiki, Android Things…



ChatGPT as an OS?

- Integrating ChatGPT into existing OSs (e.g., Windows) is already happening.



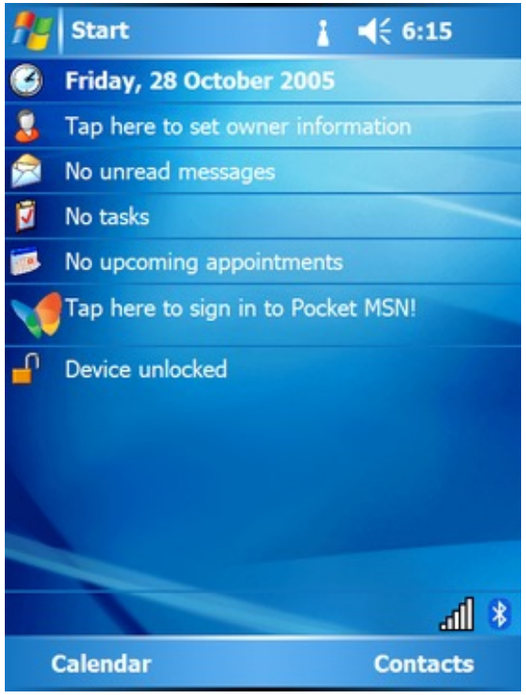
ChatGPT has dramatically impacted the tech industry since it debuted in late 2022 and started rolling out to more users in early 2023. AI is a big deal for Microsoft. The tech giant has integrated OpenAI's ChatGPT into Bing.com, Windows 11's taskbar, and plans to bring it to Office apps via Microsoft 365 Copilot.

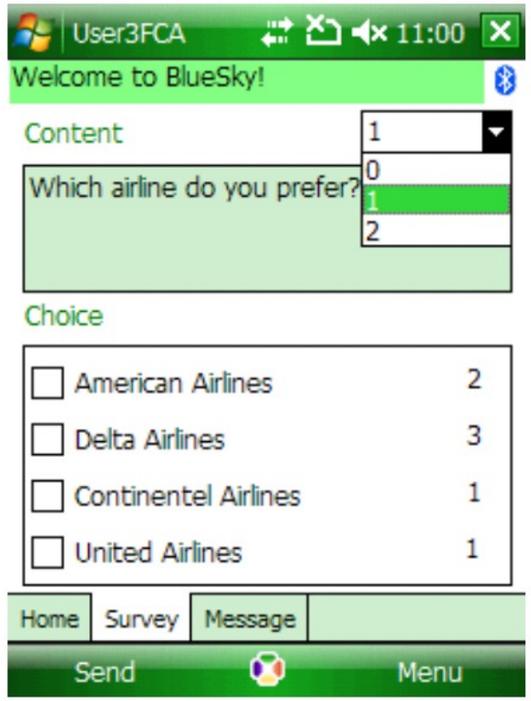
Microsoft now plans to integrate ChatGPT into Windows 10 and 11 via PowerToys, the company's open-source tool. As you probably know, Microsoft PowerToys, which debuted over three years ago, has become powerful after recent updates with features like Registry Preview.

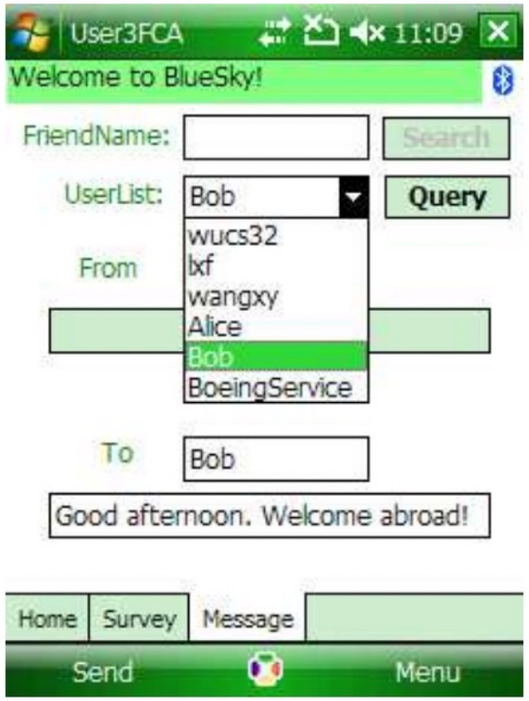
- What if ChatGPT being a Sole (Human-oriented) OS?
  - The OS of the Future

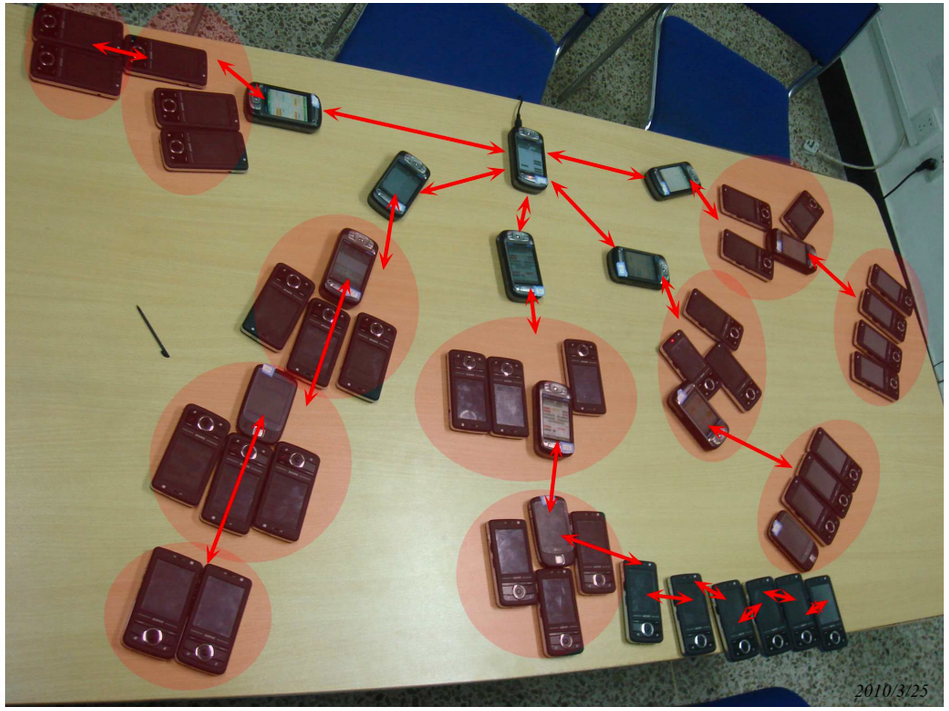
Why Learning Operating System?

- An example: Windows Mobile



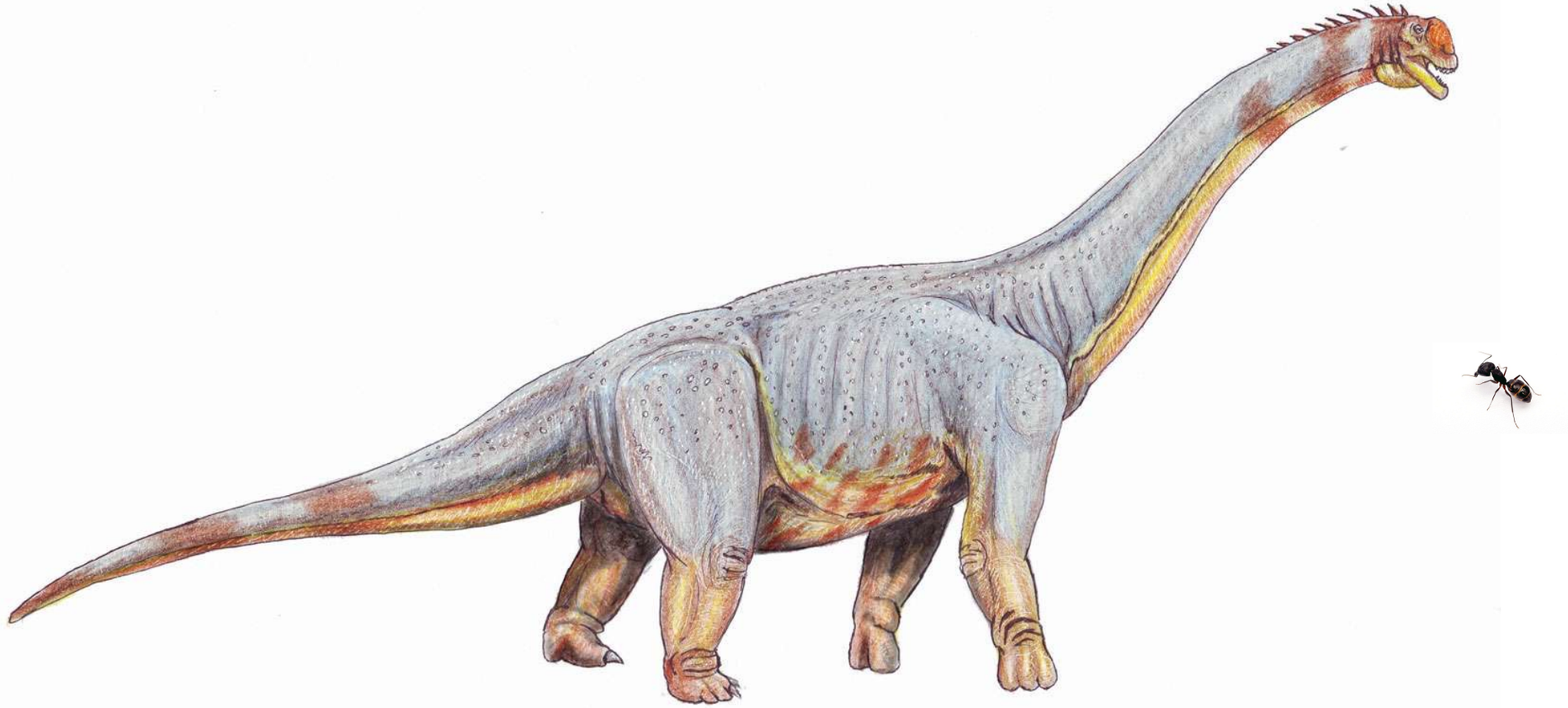






Why Learning Operating System?

- An example: ML systems



Why Learning Operating System?

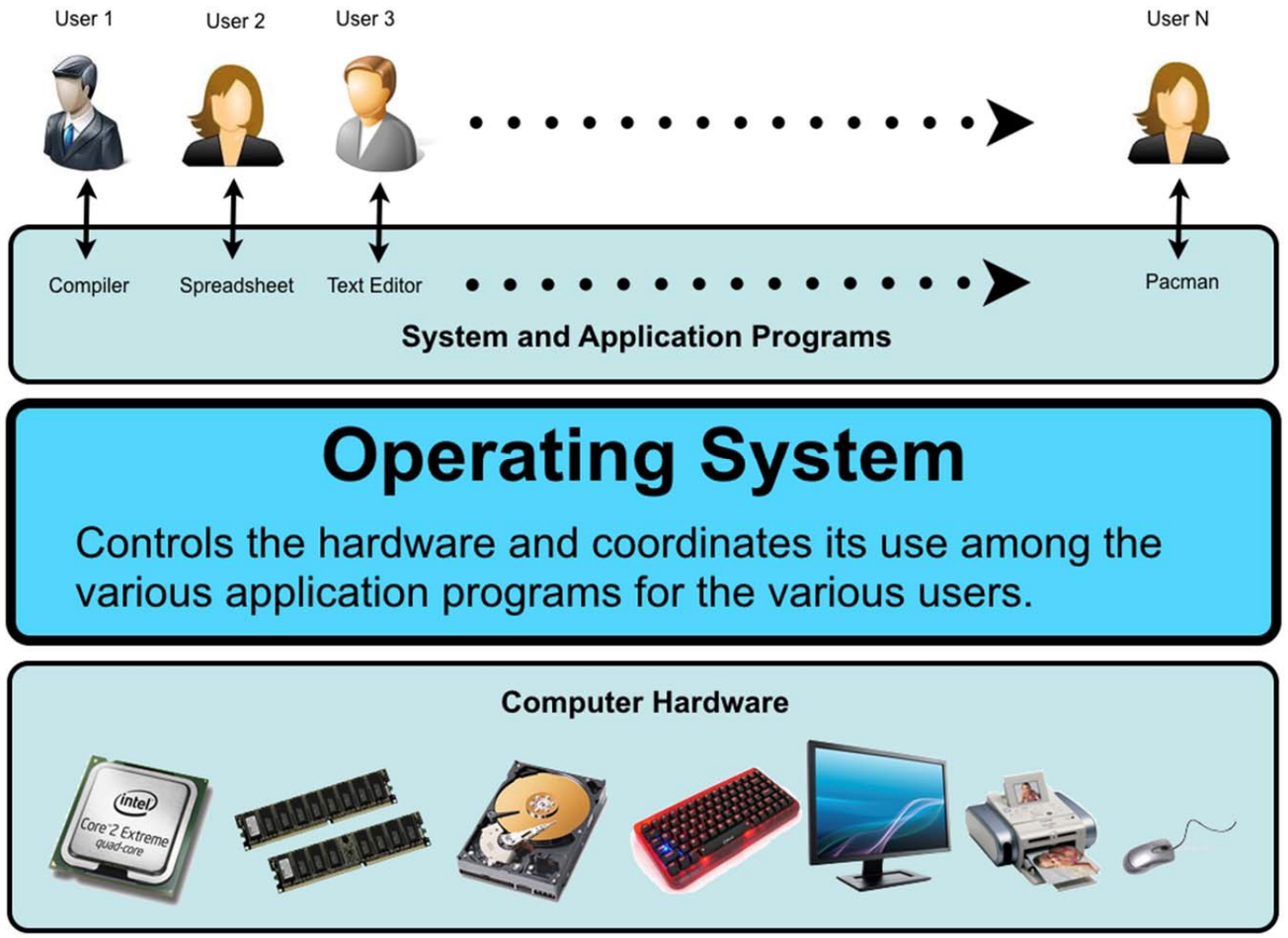
- *Beyond many good reasons….*

- *Computational Thinking*[1]
  - *It's all about problem solving*
  - *There are always more problems than you could have imagined, but luckily there are always more solutions.*

- *Systems Thinking*
  - *(Large-scale) systems are complex.*
  - *Engineering is nothing but trade-off.*

[1]*Coined by Jeannette M. Wing in 2006*

What is an Operating System?



What is an Operating System?

- A PROGRAM that
  - controls the execution of application programs
    - which program runs first? for how long? which program to be suspended or swapped out? . . . .
- Two main functions
  - primarily as a **resource manager**
    - Managing resources – CPUs, memory, disks, files, . . .
    - Decides between conflicting requests for **efficient** and **fair use** of resources
  - An **interface** between applications and hardware
    - **Separates** applications from the hardware they access
    - OS provides the APIs (Application Program Interfaces) for programs to ask for OS services/resources
      - This greatly simplifies application development

Core Services

- Allow applications to run on the system (easily, correctly, and effectively)
- Allow running applications to use memory as well as share the memory
- Allow running applications to interact with each other
- Allow applications to access and share data that stored in persistent storage

The crux of our problem

- How does the operating system support these services?
  - "Note that why the OS does this is not the main question, as the answer should be obvious: it makes the system easier to use. "
  - "Thus, **we focus on the how**: what **mechanisms and policies** are implemented by the OS to support its services? How does the OS do so **efficiently**? What **hardware support** is needed?"

Three Pieces

- Virtualization
  - Make each application believe it has each resource to itself
  - CPU, Memory, etc.
- Concurrency
  - Events are occurring simultaneously and may interact with one another
- Persistence
  - Access information permanently
  - File system, hard drive, etc.

Process & Processor Management

- A process is basically a **program in execution**
  - Process needs resources to accomplish its task
    - CPU, memory, I/O, files, initial data
  - Process termination requires reclamation of any reusable resources

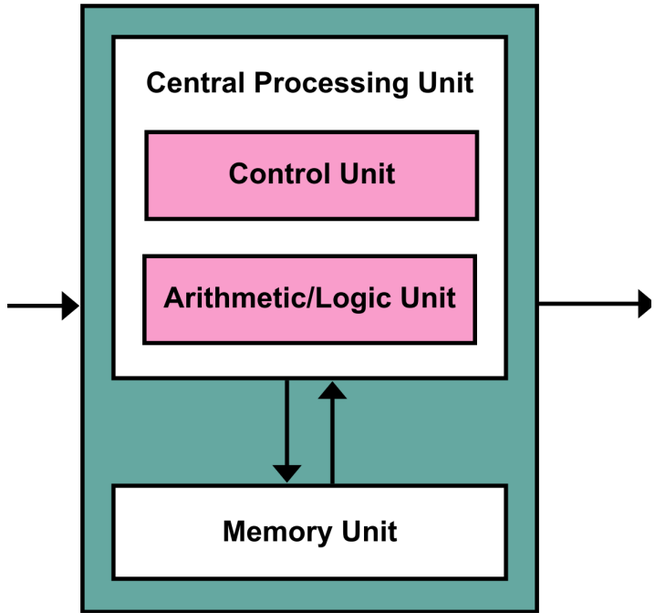
- Typically system has many processes, some are user processes, some are OS/kernel/system processes, running concurrently on one or more CPUs (cores)

Process & Processor Management

- Responsibilities of OS
  - How to manage and control application processes?
    - There are many processes running in the system
  - How to perform restricted operations?
    - Processes must be allowed to perform I/O and other restricted operations
  - How to provide the illusion of many CPUs?
    - Each given to a process
  - How to regain control of the CPU?
    - Even if processes are not being cooperative
  - How to design an effective and efficient processor scheduler?
    - Even without perfect knowledge of processes' characteristics

Memory Management

- Von Neumann model of computing
  - Instructions & data of a running process must be in physical memory in order to execute
- Main objective
  - With limited memory and many processes are running concurrently, OS determines what should be in memory
    - To optimize **CPU utilization** and computer **response** to users



Memory Management

- Responsibilities of OS
  - How can running processes share the single pool of physical memory?
    - Giving an abstraction of a private memory space for each process
  - How to maintain control over which memory locations an application can access?
    - We don't want other processes to write to another process's memory
  - How to run many processes with total memory demand higher than physical limit?
    - Getting help from the larger, slower hard disks
  - How to manage free space? What should OS do if running out of free space?
    - If running out of free space, getting back some memory from running processes

Concurrency

- OS is a concurrent program
  - Many internal kernel data structures may be updated concurrently by multiple execution logics
- Nowadays, multi-core systems and multi-thread programs are prevalent
  - Threads can update shared data simultaneously
- When there are many concurrently tasks running within the same memory space, how can we build a correctly working program?
  - Programs or OS have to carefully access shared data, with the uses of proper synchronization primitives, in order to work correctly

Concurrency

- Responsibilities of OS
  - What kind of primitives are needed for synchronization?
    - We may have to examine different kinds of synchronization and concurrency issues
  - What support do we need from hardware and OS in order to build useful synchronization primitives?
    - We want the primitives to work correctly and efficiently
- How can we use them to solve concurrency problems?

File Management

- The component in OS that manages storage disks is called the file system
  - Responsible for storing any data the user created in a **reliable and efficient** manner on the disks of the system

- File System
  - Provides a uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Maps files onto physical media and **provides mechanisms** for applications to manage and access files

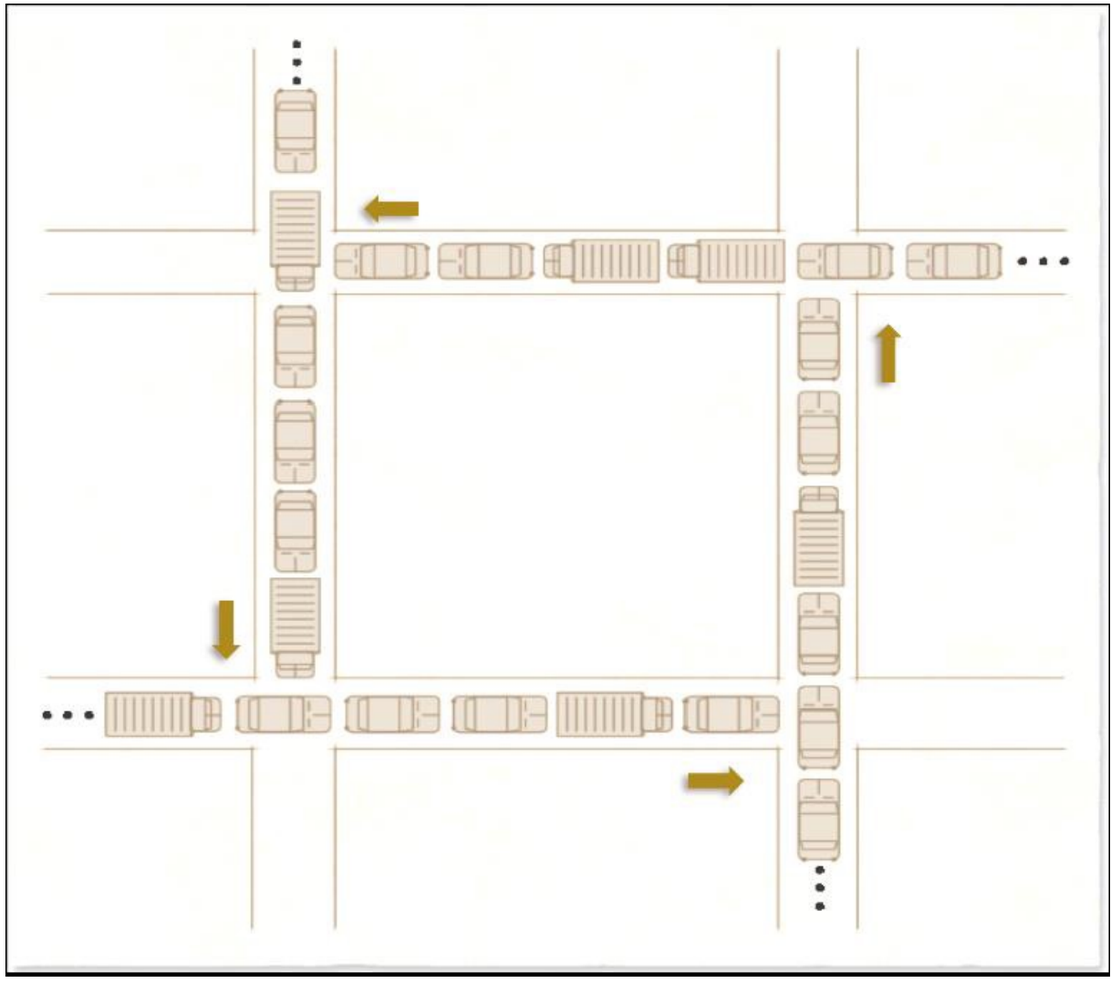
File Management

- Responsibilities of OS
  - How to manage a persistent device?
    - For example, where to find suitable storage units for a newly created file? what will happen if deleting a file?
  - How to implement the file system?
    - We need some data structures on the disk for the file system to work correctly and efficiently; e.g., given a file name, how to access the data of the file?
  - How to reduce file system I/O costs?
    - Disk access is slow, is there any way to make file access faster??

Design Goals of OS

- Makes the system convenient and easy to use by abstraction
- Provides protection between applications and between OS and applications
- Allows computer system resources to be used in an efficient manner
- Can operate on many hardware configurations
- Provides a high degree of reliability and will not fail due to isolated application/hardware errors
- Protect resources from unauthorized access by users and software
- Security, energy-efficiency, mobility…

Problems in real life



Mechanism:

Policy:

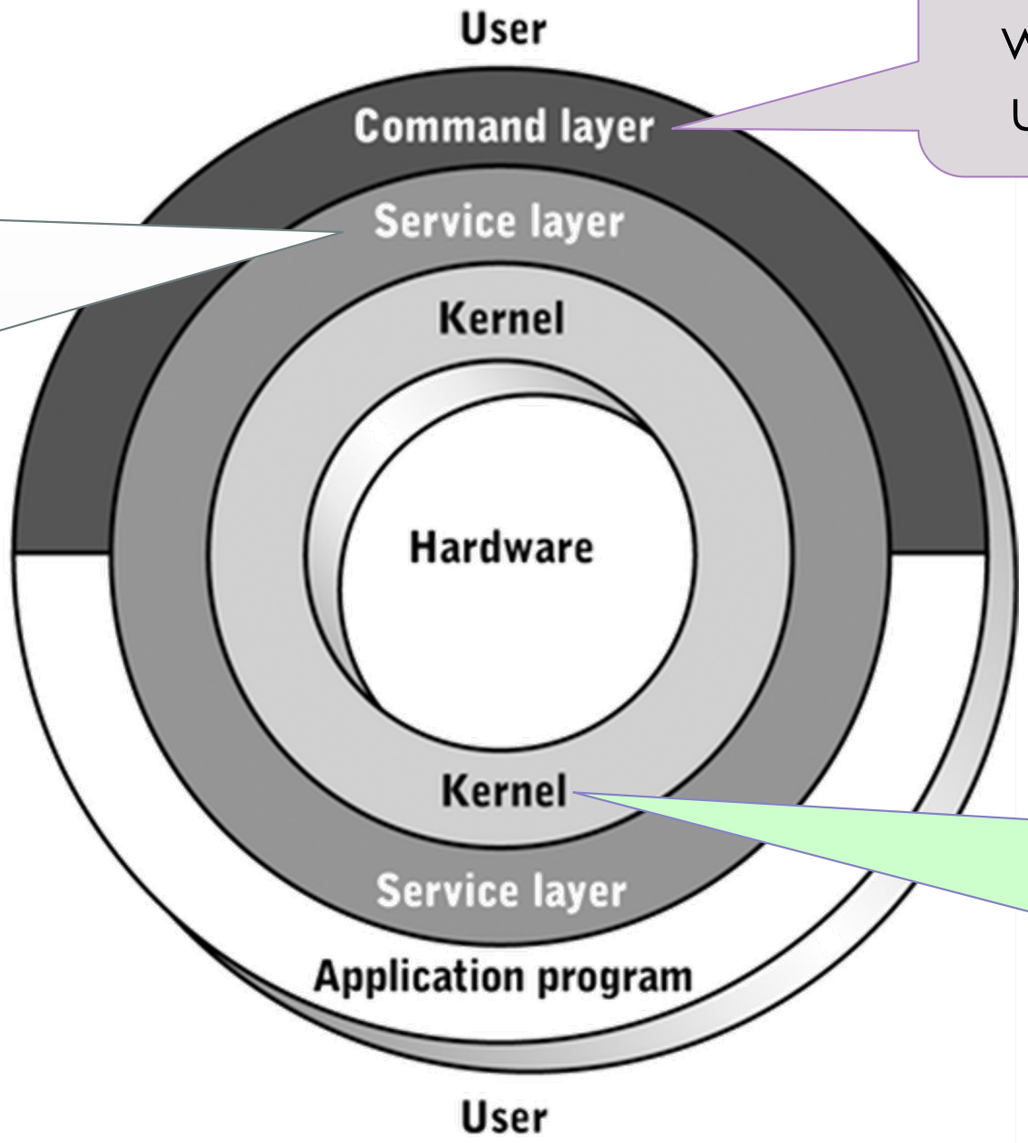
Problems in real life

OS Architectures

Operating System Structure

Usually we call it the shell, which is the interface for users to interact with OS

Contains a set of utility functions that will be used or called by application programs and the command layer



Which is the core of OS; it manages resources and directly interacts with computer hardware

Architectures

- Operating Systems tend to be complex
  - Provide many services (really too many!)
  - support variety of hardware and software
- Operating system architectures help manage this complexity
  - Organize operating system components (functionalities)
  - **Specify the privilege level** with which each component executes
- Common options
  - Monolithic Architecture
  - Layered Architecture
  - Microkernel Architecture
  - Modular Approach

(5) Hybrid Kernels: compromise between the **monolithic** and **microkernel** designs. Used in **most commercial operating systems**: Microsoft Windows NT 2000, XP, Vista, 7, 8, 10, ..; Mac OS X (XNU)

User Mode / Kernel Mode

- CPU operates in (at least) two modes
  - Kernel mode
    - Also known as privilege mode or supervisor mode
    - CPU is put in *kernel mode* **when** the OS kernel executes
    - It is at the **highest privilege** level and can execute any instruction the machine is capable of executing
  - User mode
    - CPU is put in *user mode* **when** a user application is executing
    - Running in a lower privilege level
      - Only a **subset** of the machine instructions is available
      - Ensures that one user program cannot execute instruction that may interfere with operation of other user programs

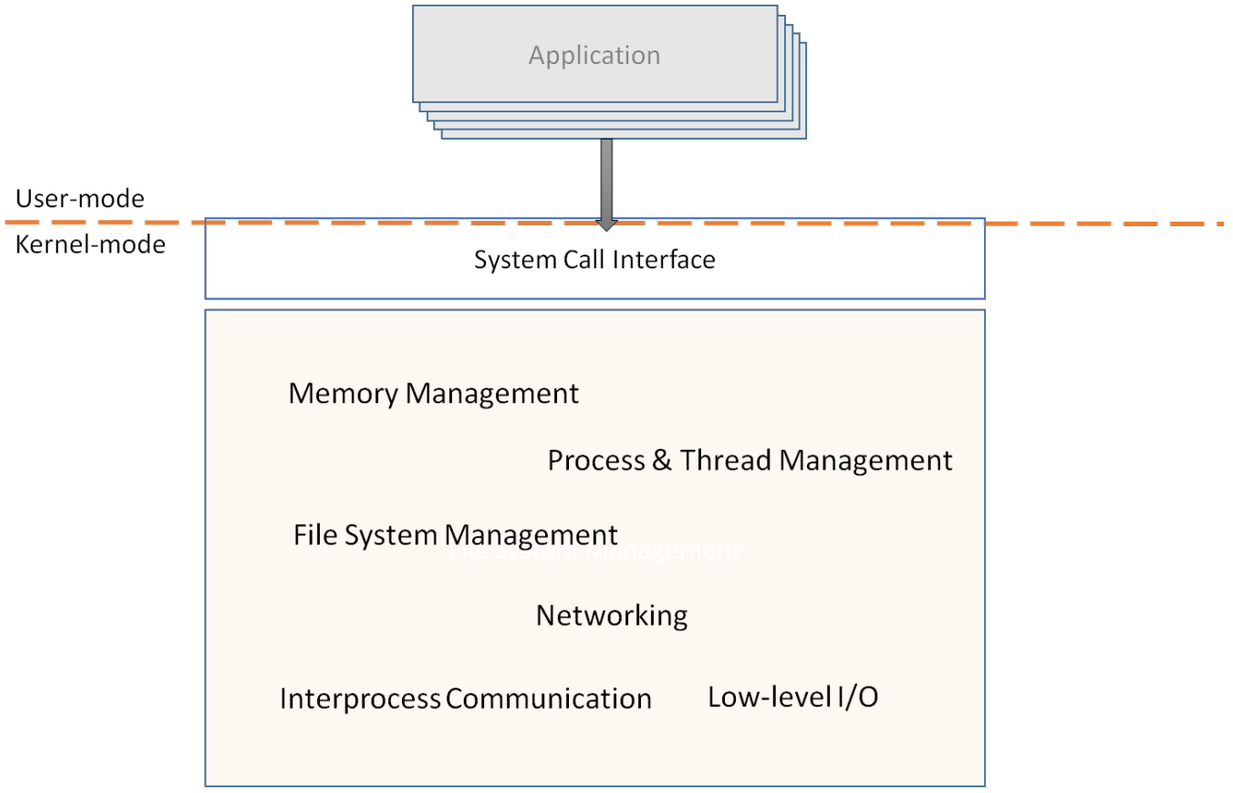
System Calls

- A system call is a mechanism allows an application program to make a request to the OS for resources/services
  - the set of system calls is the **interface (API) to the services** provided by the OS

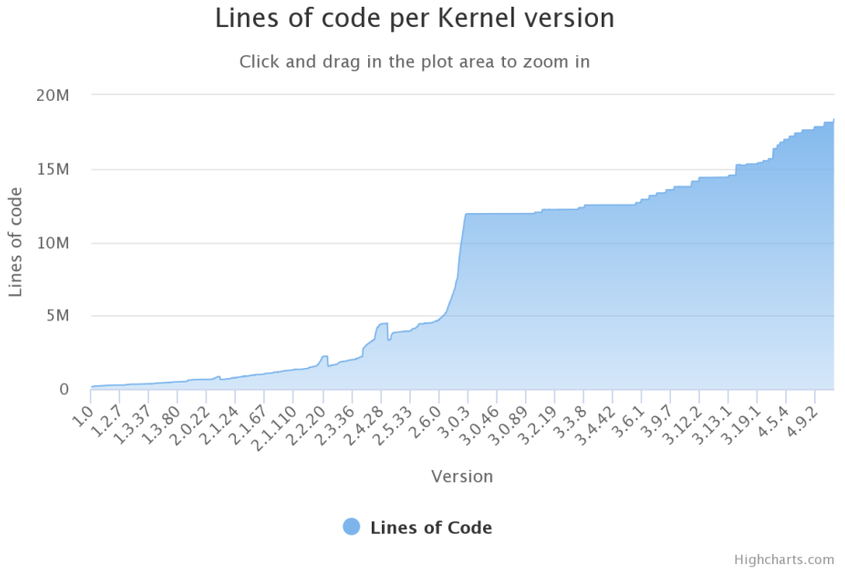
- When a system call occurs, the system **switches** from **user mode** to **kernel mode** and executes the corresponding kernel's system function
  - We call this *mode switch*
  - This is achieved by some special instructions that helps the process to ***trap*** into the kernel and ***return-from-trap*** back to user mode

Monolithic Architecture

- Every OS component is contained **in the kernel**
  - Traditionally, OS code **did not** structure its components as modules with clearly defined interfaces
  - All its components are interwoven into one large program that **runs in privilege mode**
  - Any component (within kernel) can directly communicate with any other (e.g. by using function calls)
  - Data structures are easily shared as all in one program



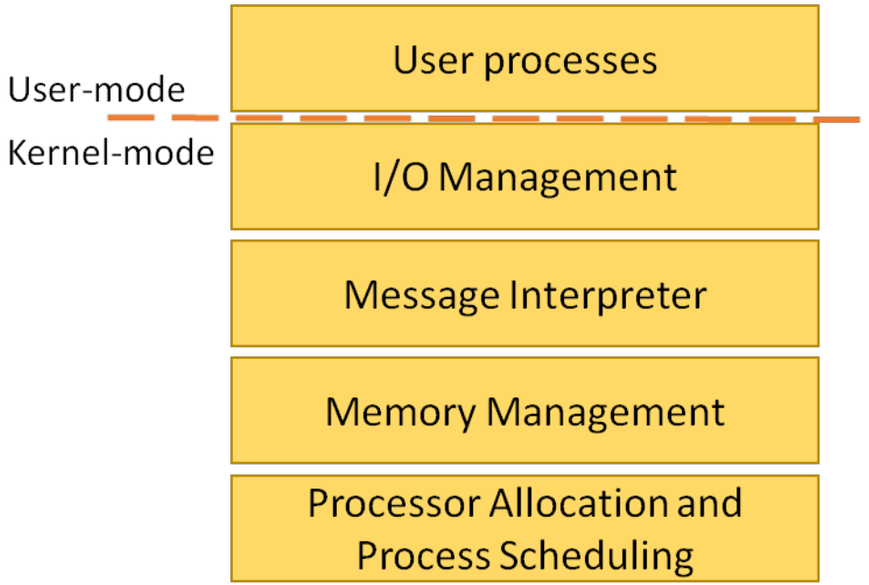
Monolithic Architecture



- Adv: Tend to be **highly efficient**
- Disadvantages
  - Components can interact with hardware directly
    - Architecture dependent code was spread throughout the kernel
  - Components can access each other's data and functions directly
    - Changes made in one component could affect other components
    - Bugs in one component can adversely affect another component
      - Difficulty in determining source of subtle errors
    - More susceptible to damage from malicious code (component)
- Examples: UNIX, Linux, MSDOS, and many more.

Layered Architecture

- Groups components that perform similar function or specific role into a module and layers the modules one on top of the other
- Each layer **communicates only** with layers immediately above and below it
  - Processes' requests might pass through many layers before being serviced
  - **System efficiency can be less** than monolithic kernels



Layered Architecture

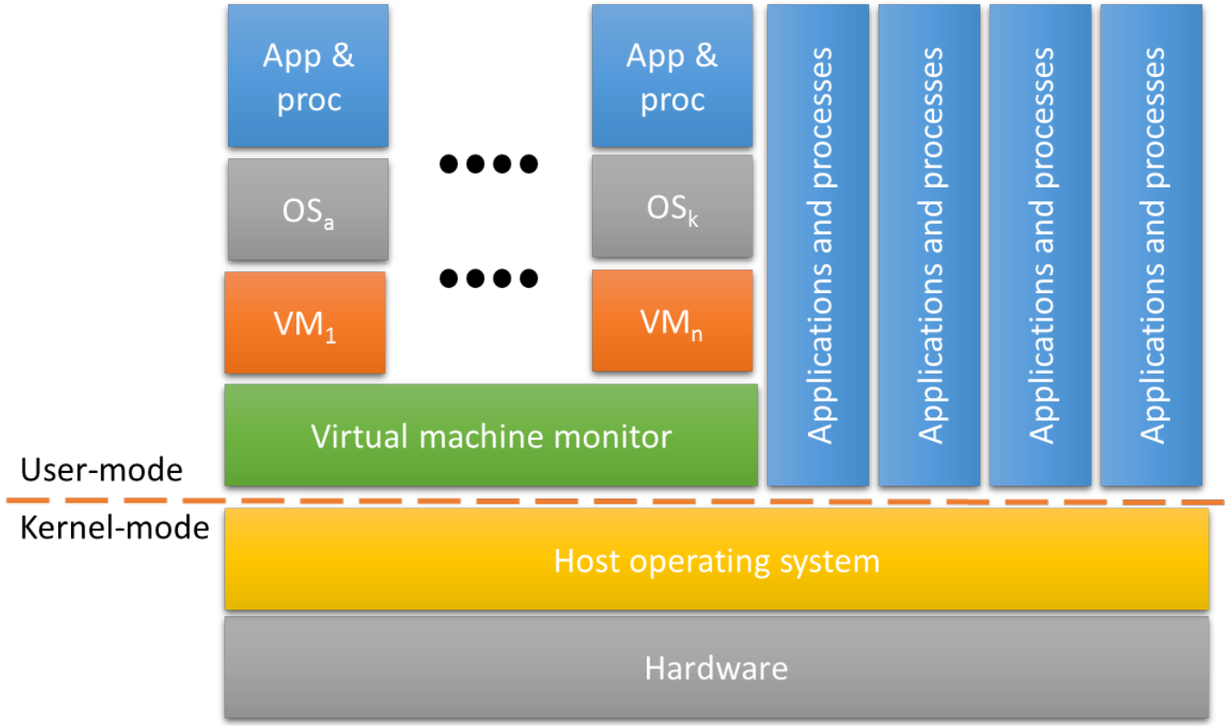
- Pros
  - Simplicity of construction and debugging
    - Allows debugging starting at lowest layer, adding one layer at a time until the whole system works correctly
  - **Information hiding** – each layer only knows the interface provided by immediate lower layer
    - We can replace one layer by another implementation without affecting other parts of the system
- With the layered approach, the designers have a choice where to draw the kernel-user boundary

Layered Architecture

- Cons
  - Poor performance
  - Difficult to define and order layers
  - Not flexible
  - Disjunction between model and reality: real systems are more complex, difficult for such layered abstraction (see Windows NT story)

*The first release of **Windows NT (3.x)** using the **layered organization** suffered low performance compared to that of **Windows 95**.*
Windows NT 4.0 moves layers from user space to kernel space and more closely integrating them to improve the performance.

Virtual Machine – An Example of Layered Approach

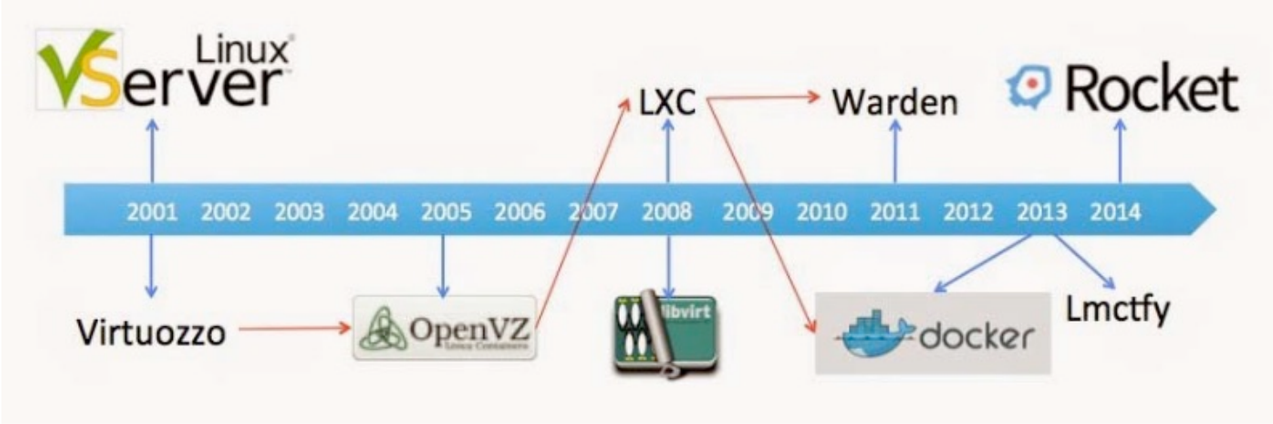


- Virtual machine monitor
  - Also named as ***hypervisor***
  - Runs on top of host OS – in user mode OR incorporated into host OS – kernel mode
  - Virtualizing the hardware resources and giving the illusion to each running OS that it controls the machine (which is a virtual machine)
  - In essence, it serves as an OS for OSs.

For Your Information Only

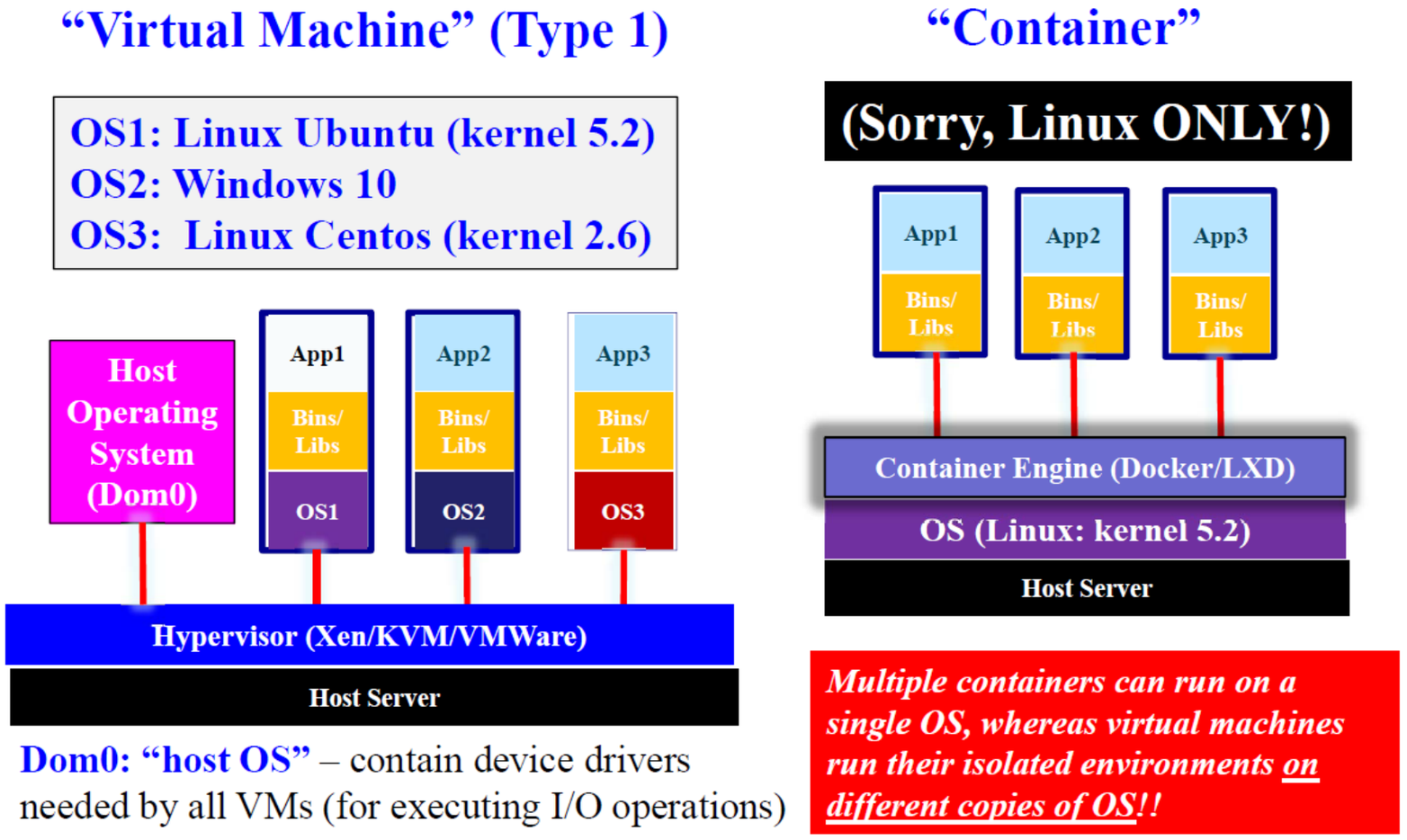
“Container” Technologies

- OS-Level Virtualization: Virtualizes servers on top of the operating system.
  - Anything running inside a container can only see resources that have been assigned to that container.
- Examples:
  - LXC (LinuX Containers), LXD, Solaris “Zones”, Docker, Google Containers, …



"Virtual Machine" vs "Container"

- (Simplest view for comparison)

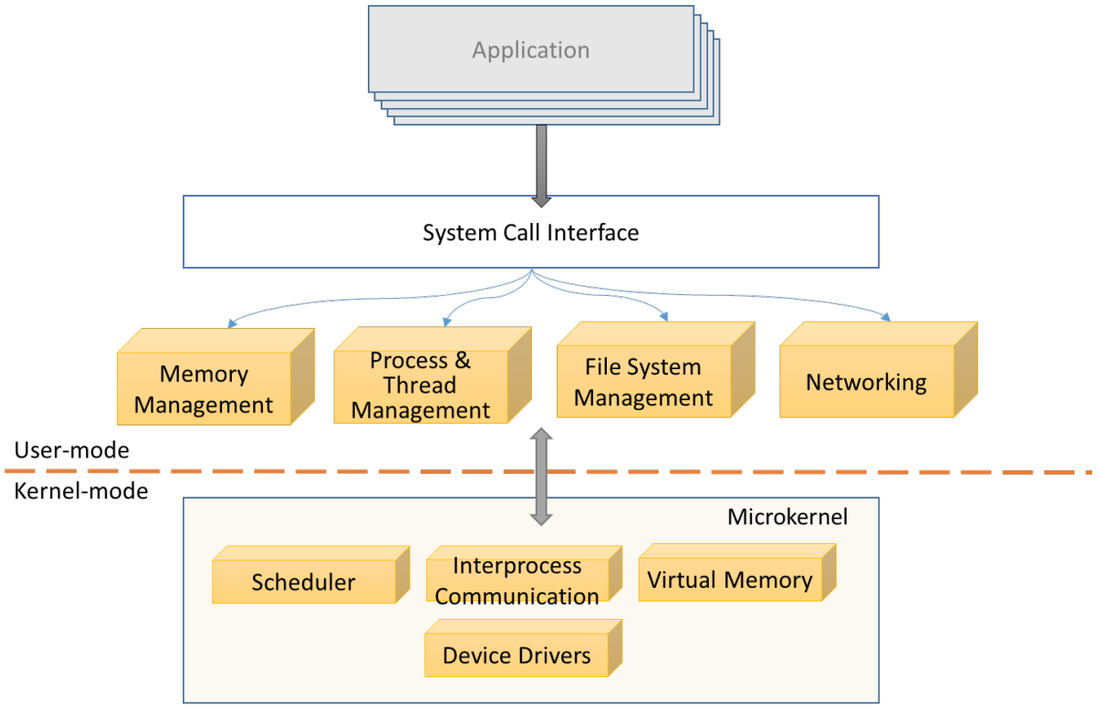


Dom0: "host OS" – contain device drivers needed by all VMs (for executing I/O operations)

Multiple containers can run on a single OS, whereas virtual machines run their isolated environments on different copies of OS!!

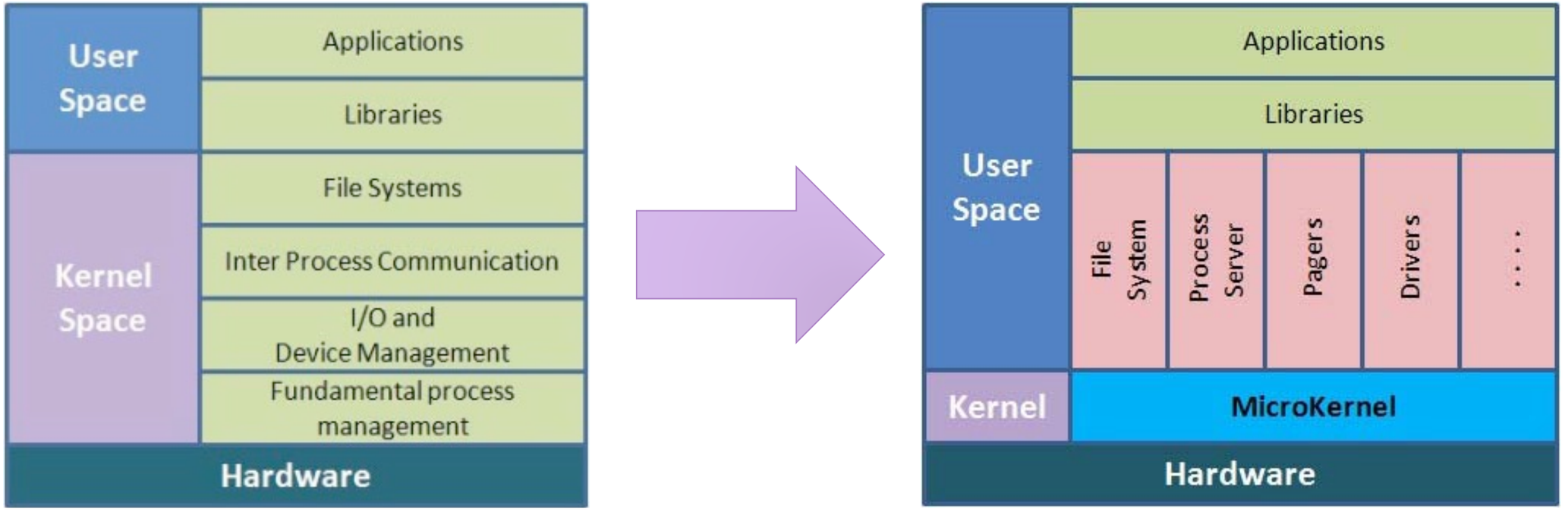
Microkernel Architecture

- **Moves** as many functionalities from the kernel space into user space processes (being called as servers)
  - Attempt to keep kernel small
  - The system becomes more stable as only the bare essentials are running in kernel mode
  - Easier to extend and port to other platforms



Microkernel Architecture

- reduce the kernel to **basic process communication** and **I/O control**, and let the other system services reside in **user space** in form of ***normal processes*** (called ***servers***).



Monolithic Kernel **Microkernel**

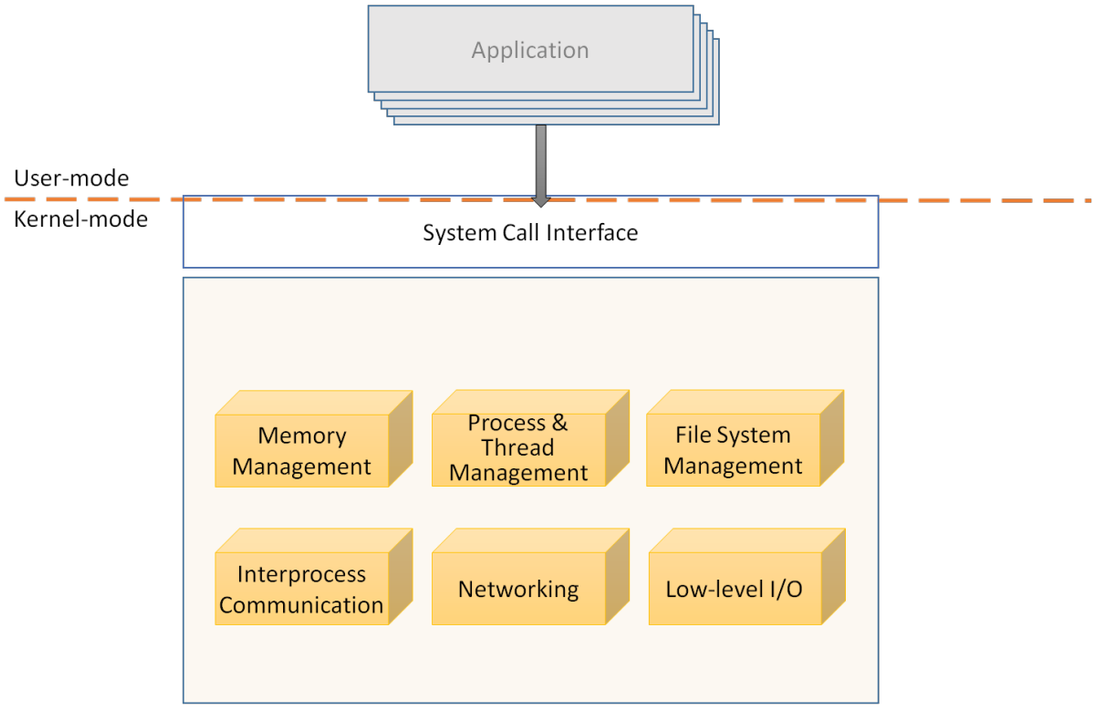
Microkernel Architecture

- Pros
  - Extensible, portable and scalable
  - More secure & reliable (less code is running in kernel mode)

- Con
  - Poor performance: Server processes (in user space) are interacting by means of message exchanges, which need the helps from kernel
    - **Induce significant performance overhead** because of communications have to go through kernel
- Most embedded systems adopt microkernel

Modular Approach

- Most modern operating systems still adopt the monolithic architecture but implement kernel modules
  - Each core component is separated and implemented as module
  - The whole kernel is a collection of modules
  - E.g., Windows NT 4.0 = monolithic + microkernel

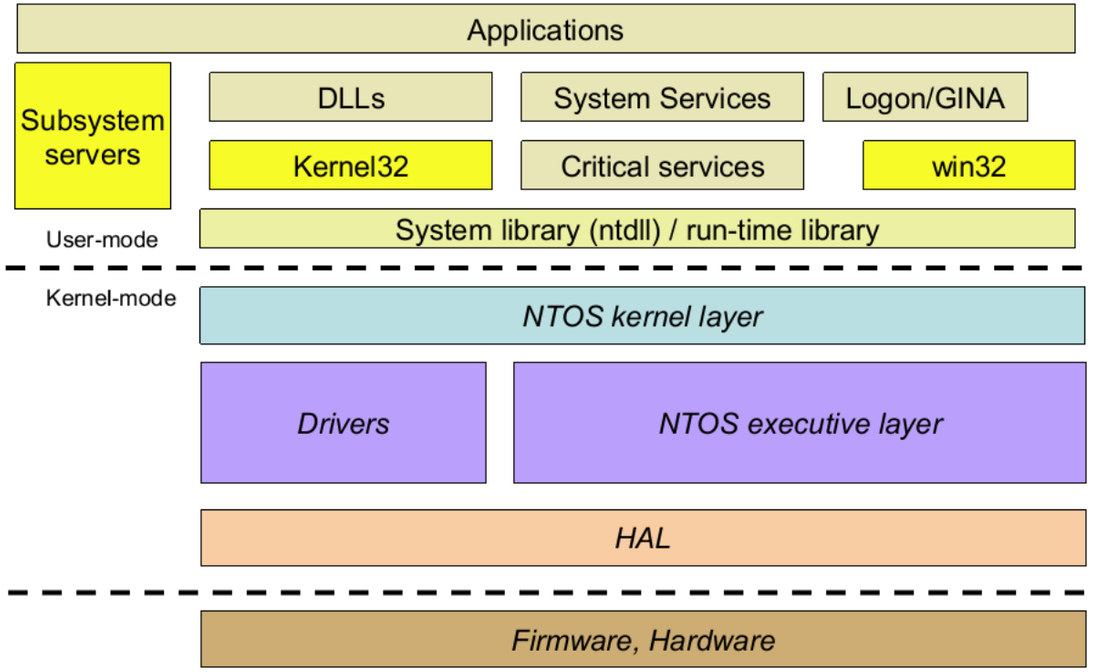


Modular Approach

- Overall, similar to layers but is more flexible and efficient as
  - each kernel module has well-defined, **protected interfaces**; any module can call any other module
  - more efficient to communicate between modules as they are all in kernel
  - allows certain features to be implemented dynamically and loaded as needed (dynamically loadable modules)
    - Memory is conserved as only required modules are loaded in memory
  - more extensible as kernel modules can be modified separately and new modules can be added easily
- Examples: Solaris, Linux, and traditional Unix kernels

Examples

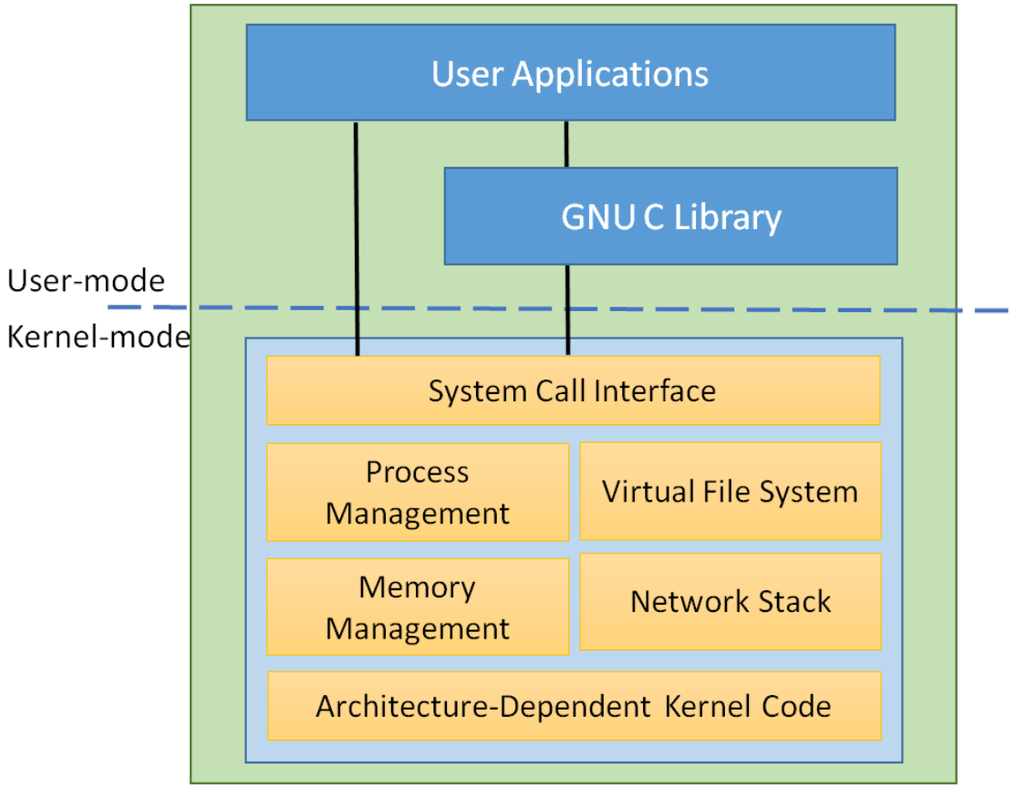
Windows System Architecture



(Source: Windows Kernel Internals by Dave Probert)

A hybrid kernel with modular design principles

Linux System Architecture

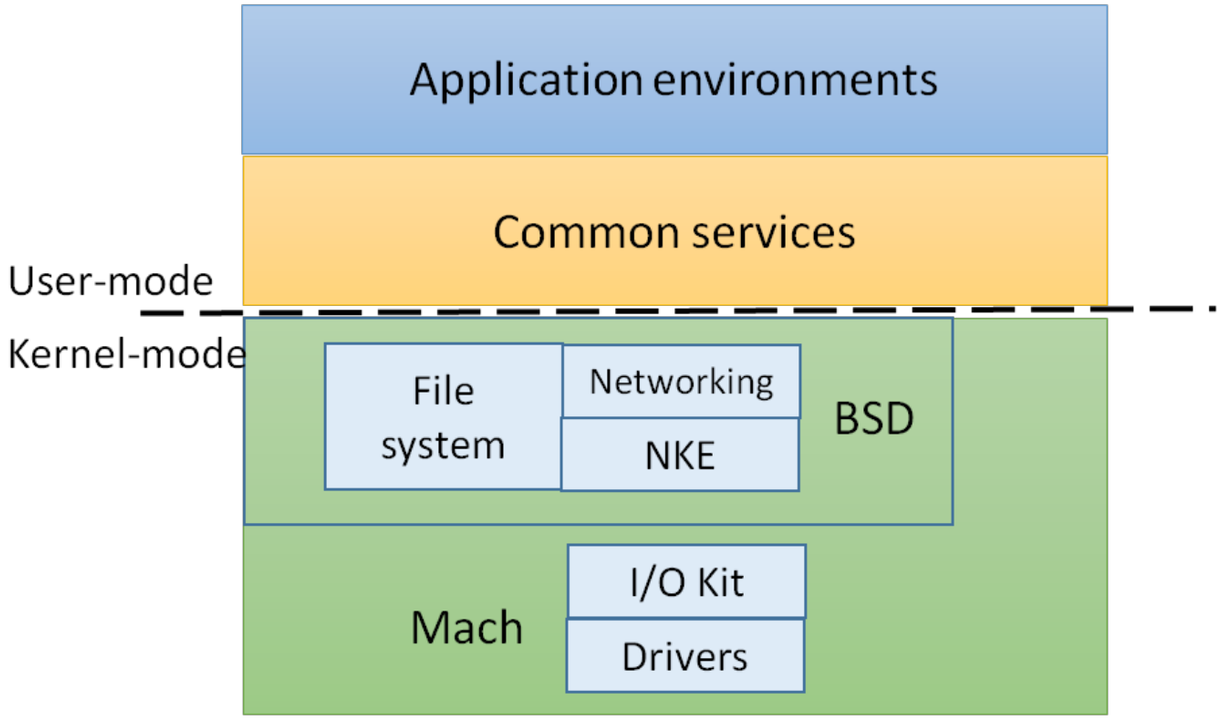


(Source: Anatomy of the Linux Kernel by M. Tim Jones)

A monolithic approach with the support of loadable kernel modules

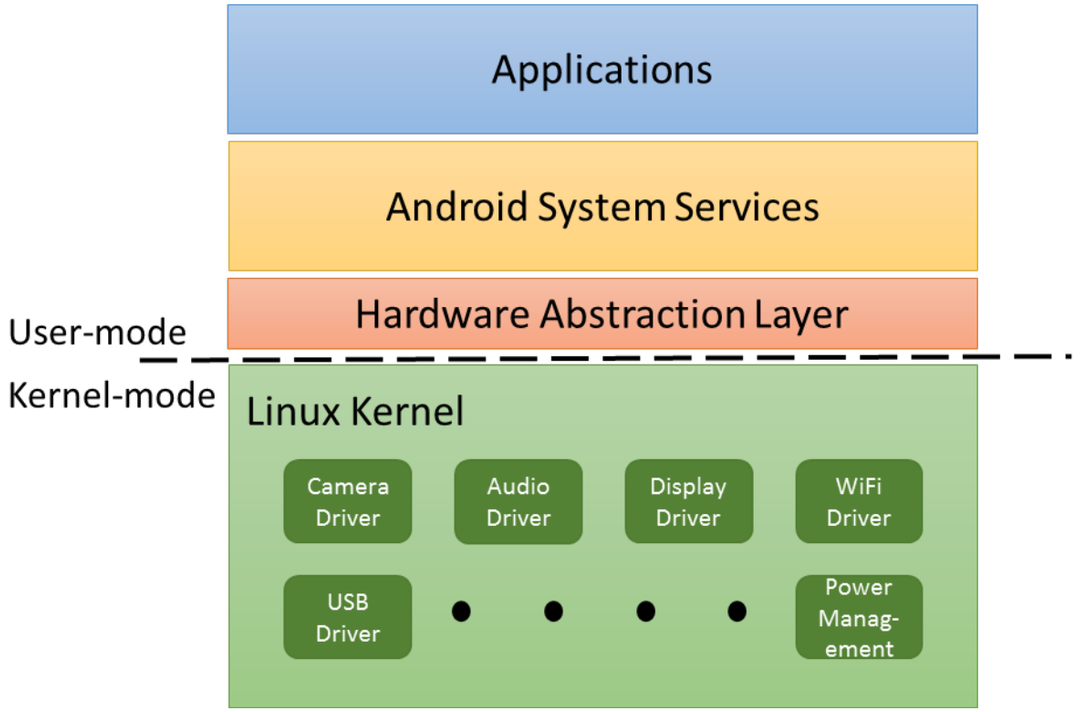
Examples

Mac OS X (XNU) Architecture



(Source: Kernel Programming Guide by Apple)

Android System Architecture



(Source: Android Open Source Project)

A hybrid kernel with the support of loadable kernel extensions

Summary

- What is OS? It is a resource manager that manage & coordinate the use of resources amongst users, and it also provides an abstractive view on the computer to users.

- Different OS architectures have their advantages and disadvantages; however, modern OSs tend to use the monolithic approach with layers and kernel modules for extensibility.