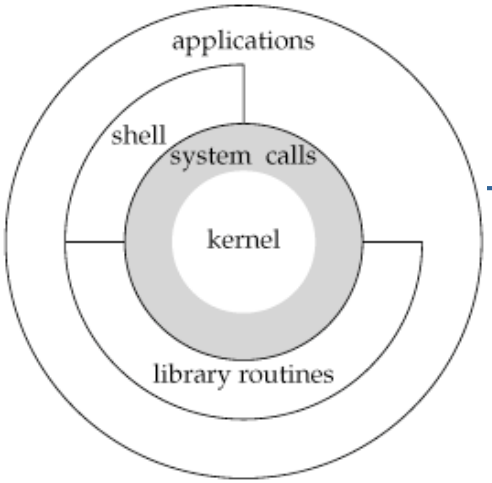
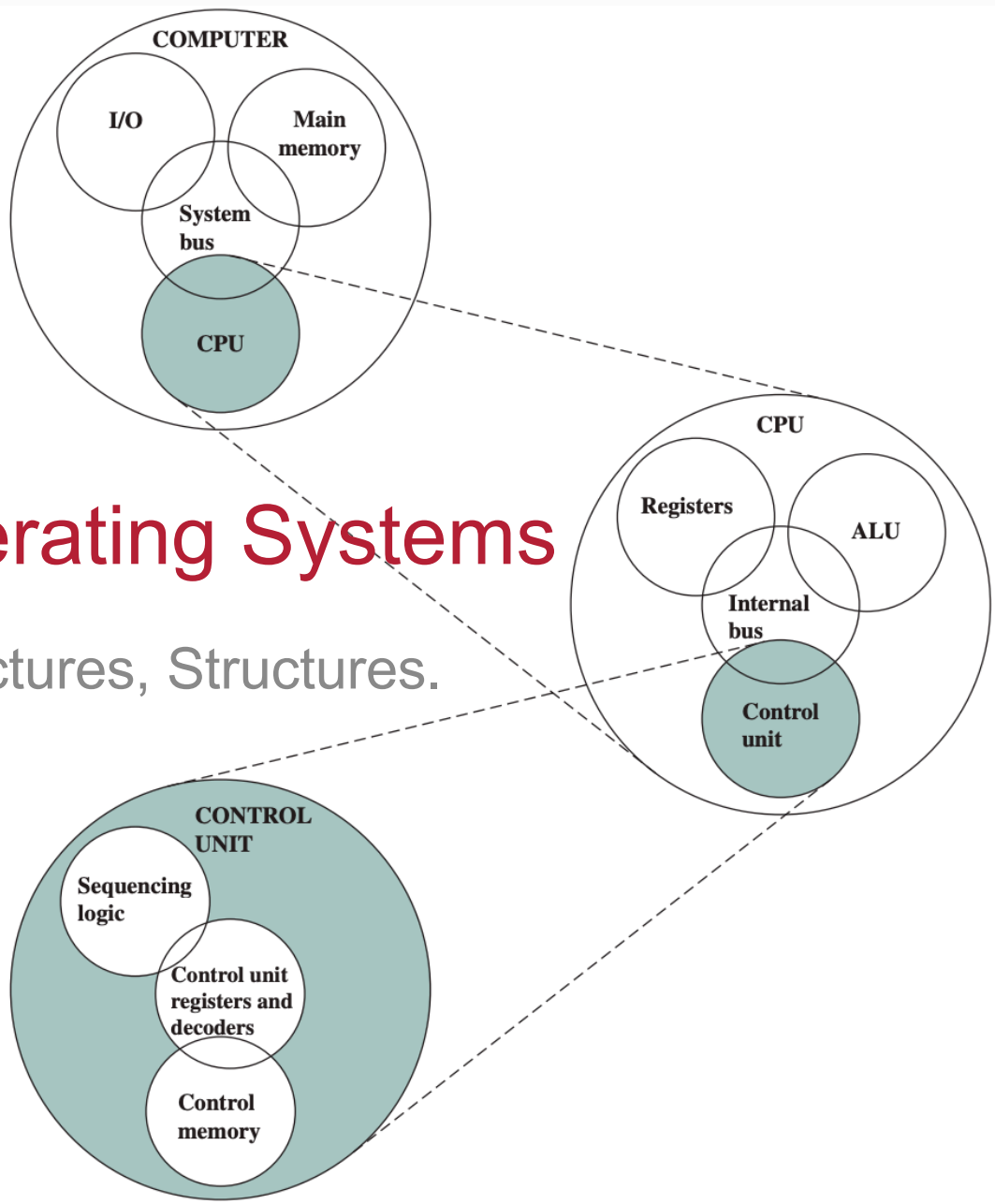
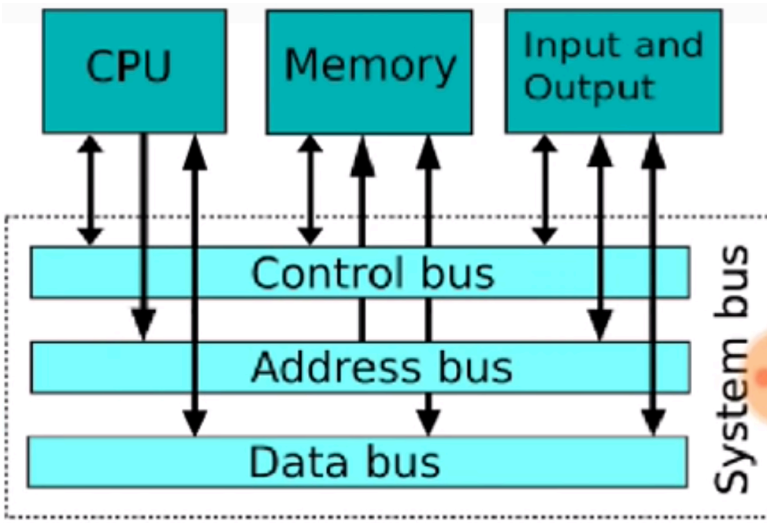




Computers & Operating Systems

LS-02. Concepts, Architectures, Structures.

1. Computer System Architecture



Von Neumann Machine Architecture

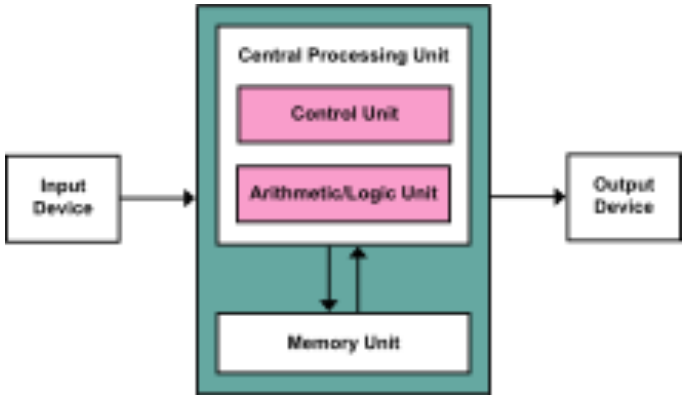
In the 1930s, the US government commissioned Harvard and Princeton Universities to develop a computer architecture for naval artillery.

Won by Princeton, but …

Princeton Machine Architecture

The von Neumann architecture, or von Neumann model or Princeton architecture.
Components:

- A processing unit that contains an arithmetic logic unit and processor registers
- A control unit that contains an instruction register and program counter
- Stored-program concept (memory that stores data and instructions)
- External mass storage
- Input and output mechanisms



The term "von Neumann architecture" has evolved to mean any stored-program computer in which an instruction fetch and a data operation cannot occur at the same time because they share a common bus.
This is referred to as the von Neumann bottleneck and often limits the performance of the system.

Mitigating the Von Neumann performance bottleneck:

- Providing a cache between the CPU and the main memory,
- …
- Implementing the CPU and the memory hierarchy as a system on chip.

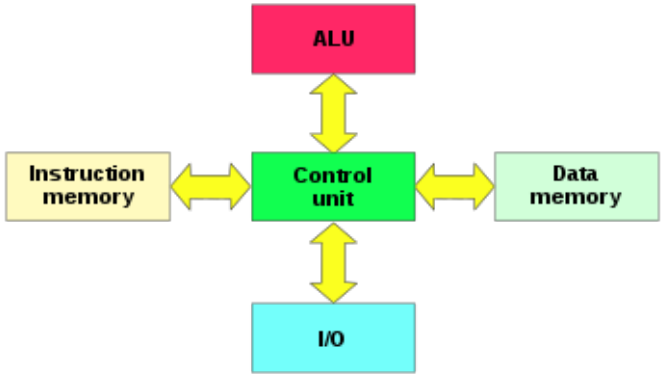
John von Neumann
1903-1956

Harvard Machine Architecture

In the 1930s, the US government commissioned Harvard and Princeton Universities to develop a computer architecture for naval artillery.

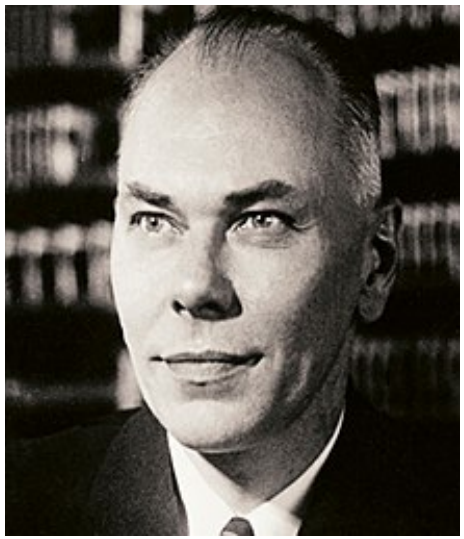
The Harvard architecture is a computer architecture with separate storage and signal buses for instructions and data.
Types of Buses:

- Data Bus: It carries data among the main memory system, processor and I/O devices.
- Data Address Bus: It carries the address of data from processor to main memory system.
- Instruction Bus: It carries instructions among the main memory system, processor and I/O devices.
  Instruction Address Bus: It carries the address of instructions from processor to main memory system.



Modern Machine internal & external design

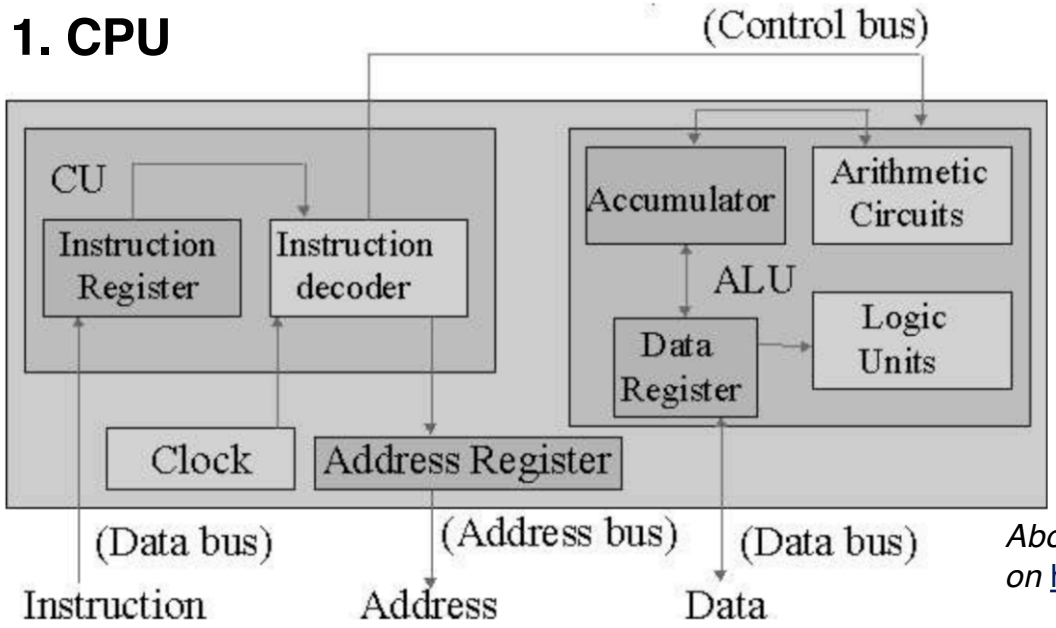
- Modern high performance CPU chip designs incorporate aspects of both Harvard and Princeton architecture.
- For performance reasons, modern processors internally (invisible to the user) use modified Harvard architecture - separate buses and separate RAM (Caches) between CPU and Instruction Cashes, CPU and Data Cashes.



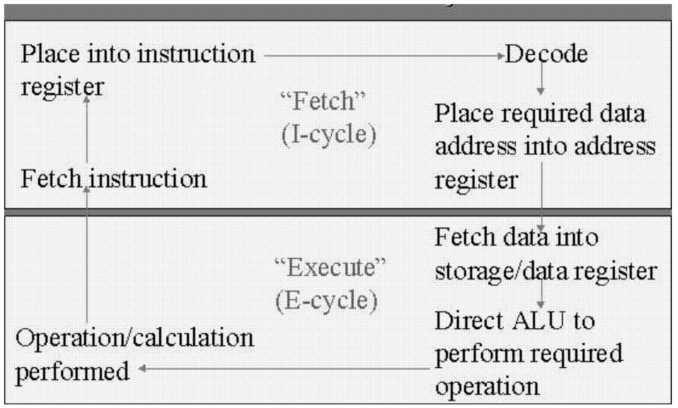
Howard Hathaway Aiken
1900 -1973

Computer Architecture

1. CPU

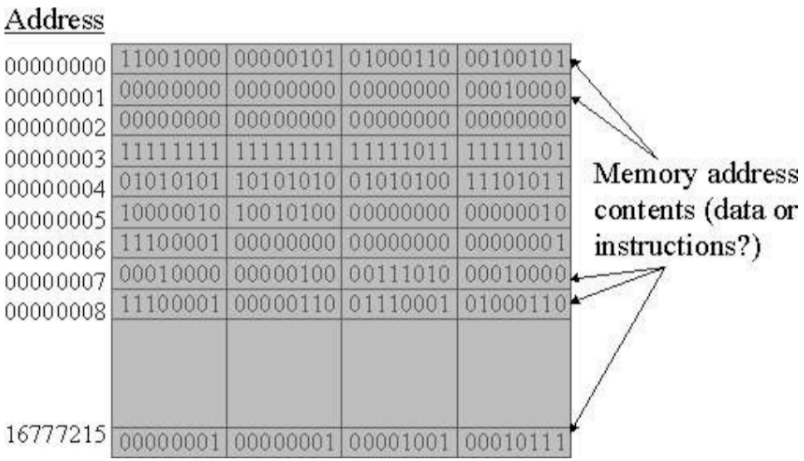


1.1. Two CPU Cycles

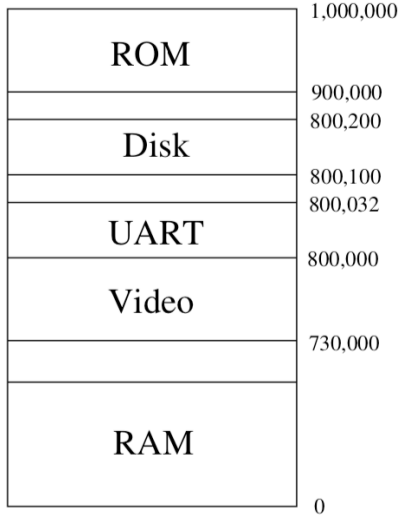


About Subscalar, Scalar, Superscalar Processors read on https://en.wikipedia.org/wiki/Cycles_per_instruction

2. Main Memory (RAM & ROM)



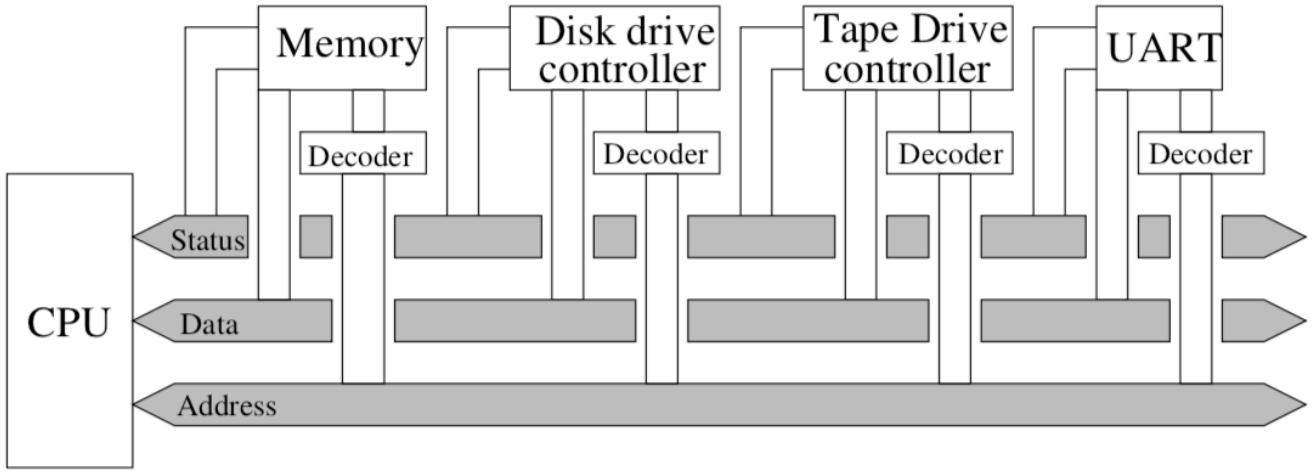
3. Peripheral Devices Addrs.



I/O Decoding Addresses

Computer Architecture

4. System Bus (Von Neumann Architecture)

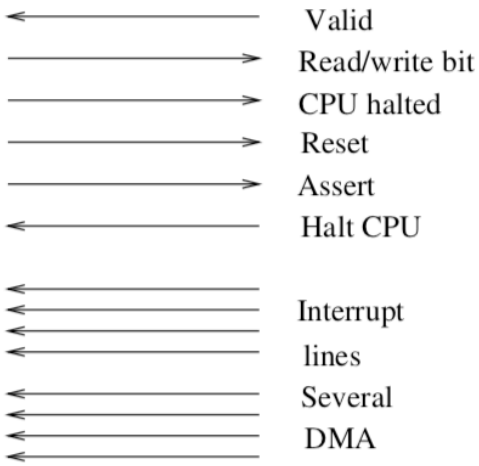


UART - Universal Asynchronous Receiver-Transmitter (COM, USB)

Ex. of Data & Address Buses

Computer	Address Bus	Data Bus
IBM XT	20-bit	8-bit
IBM AT	24-bit	16-bit
486/Pentium	32-bit	32-bit
68030/040	32-bit	32-bit
Sparc ELC	32-bit	32-bit
DEC Alpha	64-bit	64-bit

Status Bus



5a. Interrupts

App or Hardware send Interrupts to CPU about high priority Events.

Interrupt Vectors

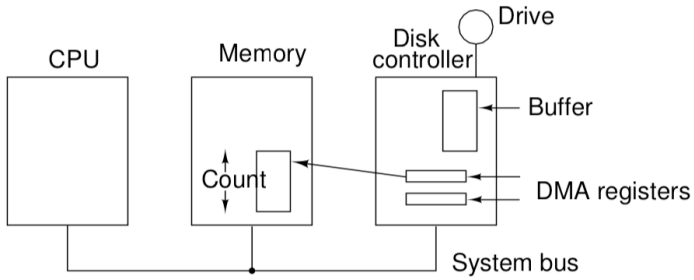
Address	Holds Address Of	For Example
0	Reset Handler	1,000,870
1	IRQ 1 - Keyboard	1,217,306
2	IRQ 2 - Mouse	1,564,988
15	IRQ 15 - Disk	1,550,530
16	Zero Divide	1,019,640
17	Illegal instruction	1,384,200
18	Bad mem access	1,223,904
19	TRAP	1,758,873

5b. System Calls or Traps

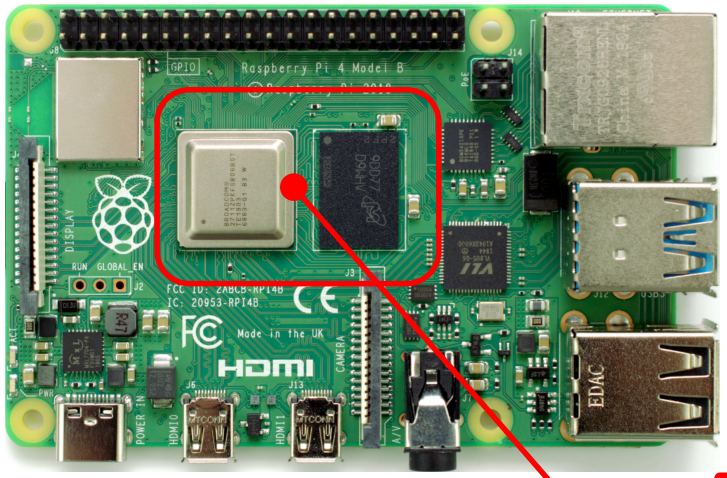
Program-initiated control transfer from user to the OS to obtain service from the OS.

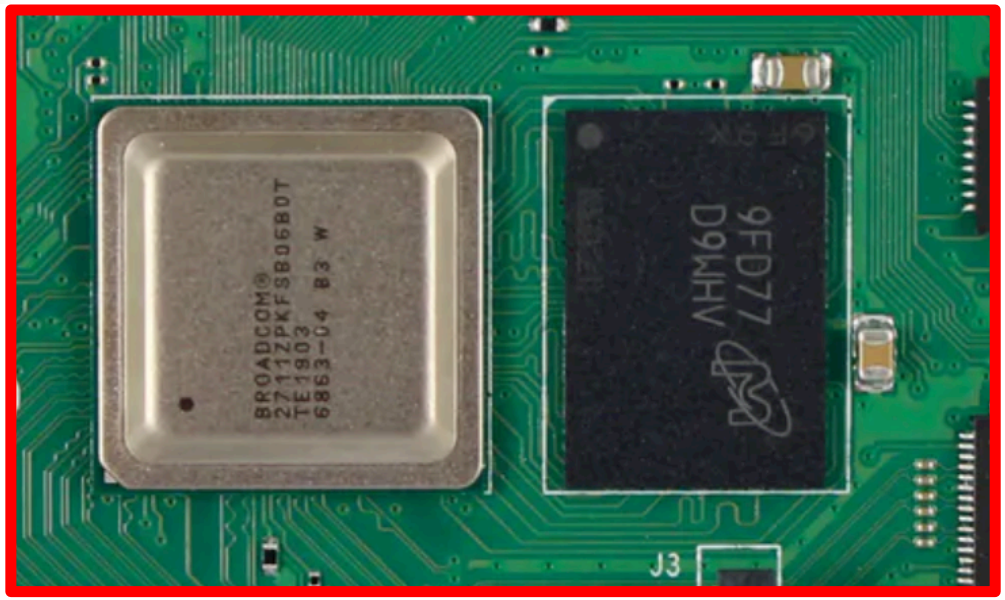
5c. I/O DMA

Allow the peripherals directly communicate without CPU.



System Bus





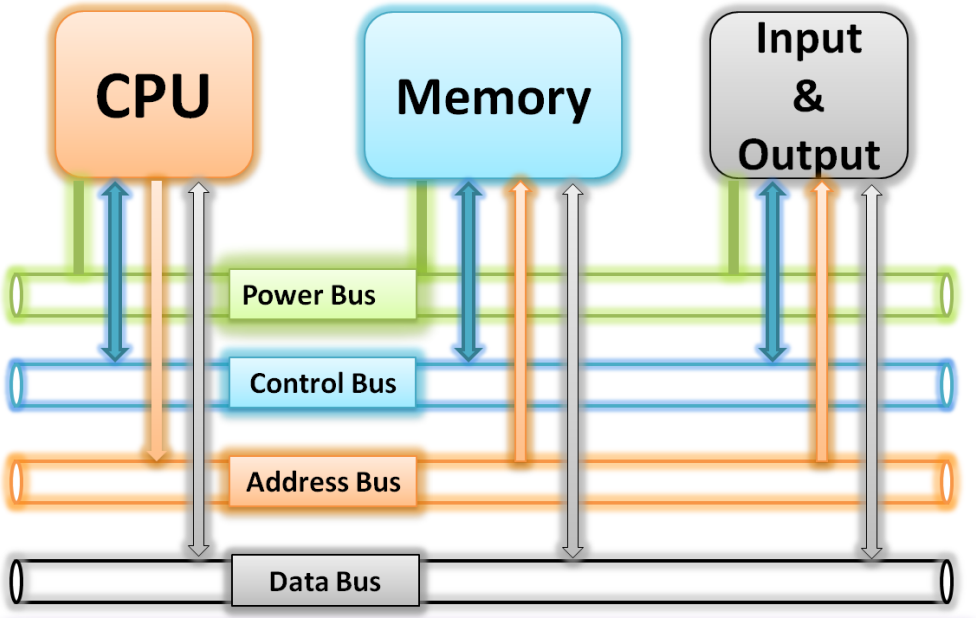
Microcomputer Raspberry Pi 3 with Linux/Windows on board.

When you zoom in, you see Bus lines.

System Bus

- The bus in the computer is a subsystem that is used to connect computer components and transfer data between them, they make use of wires that are known as a ‘bus’.

- Bus speed is listed in MHz.
- Bus may be parallel or serial.
- Parallel buses transmit data across multiple wires.
- Serial buses transmit data in the bit-serial format.



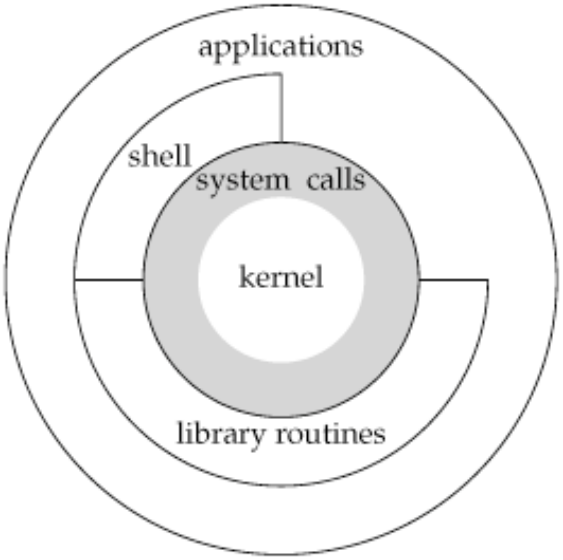
Functions of Bus in the computer

- Power: It provides power to various peripherals connected to it.
- Addressing: It allows data to be sent to or from specific memory locations.
- Data Sharing: All types of buses found in computer transfer data between the computer peripherals connected to it.
- Timing: It provides a system clock signal to synchronize the peripherals attached to it with the rest of the system.

2. OS Structure

What is an operating system?

- The first program that loads, “application with no top”.
- Operating System provides interface b/w user and software/hardware.
- A program that lets you run other programs.
- A program that provides controlled access to resources: CPU, memory, display, keyboard, mouse, persistent storage, Network. This includes: naming, sharing, protection, communication.
- Type of operating system includes single and multiuser OS, multiprocessor OS, real-time OS, Distributed OS.
- Operating system including:
  - kernel,
  - command interpreters,
  - utility programs,
  - window managers,
  - help subsystem,
  - file manager,
  - editors, …



Architecture of the UNIX operating system

Operating System Function

- **Security** – The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.
- **Control over system performance** – Monitors overall system health to help improve performance.
- **Job accounting** – Operating system Keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of user.
- **Error detecting aids** – Operating system constantly monitors the system to detect errors and avoid the malfunctioning of computer system.
- **Coordination between other software and users** – Operating systems also coordinate and assign interpreters, compilers, assemblers and other software to the various users of the computer systems.

OS Kernel

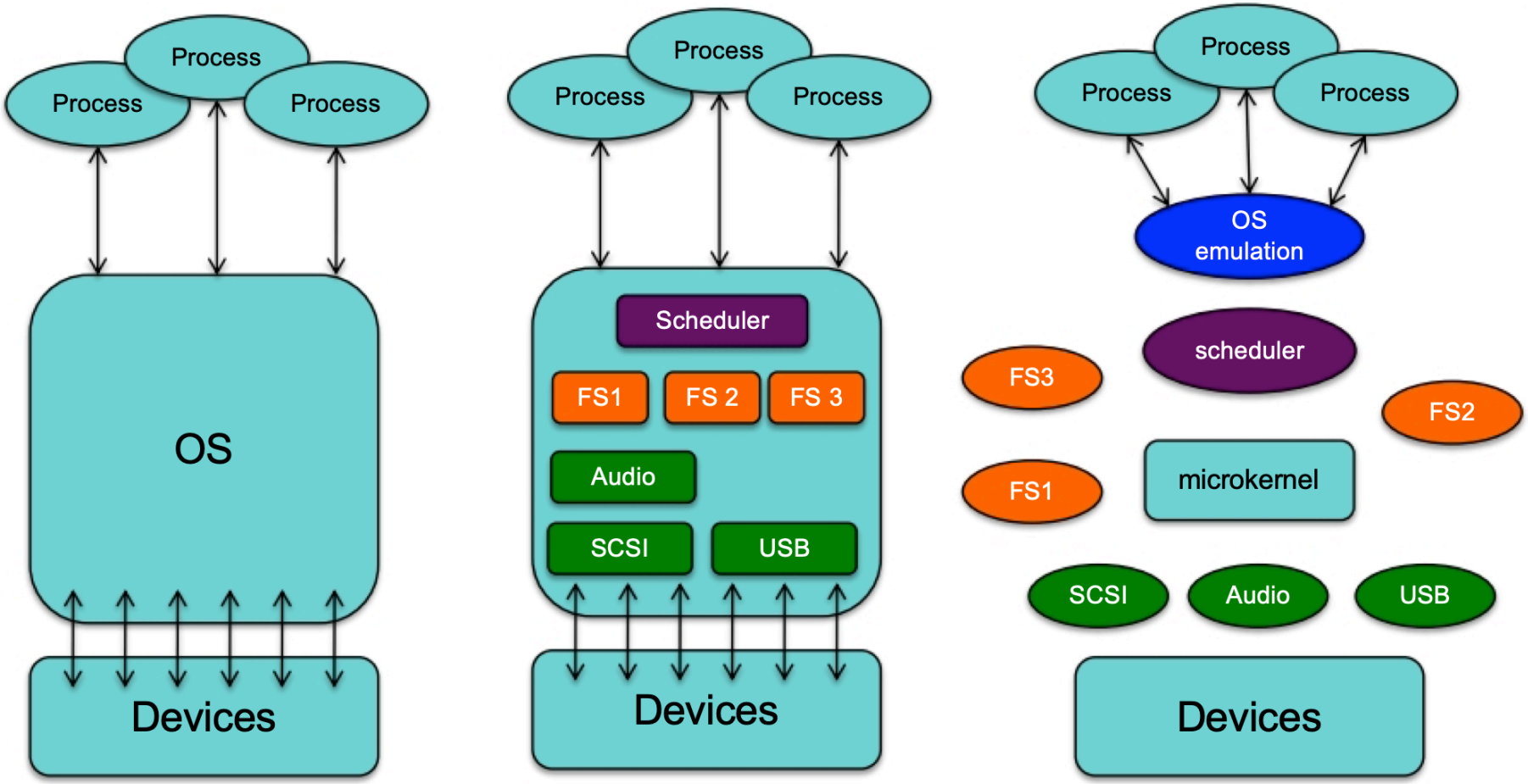
- Kernel
  - Kernel provides interface b/w application and hardware.
  - Core component of the system that manages resource access, memory, and process scheduling.
  - The kernel of an operating system is the part responsible for all general operations.
  - Kernel is always loaded into memory, and kernel functions always run, handling processes, memory, files and devices.

- Kernel Structure Types:
  - Monolithic kernel (Layered system): IBM/360, AIX, BSD, Linux.
  - Modular kernel: Linux, FreeBSD, Android.
  - Micro Kernel: Mach, QNX, MINIX.
  - Hybrid kernel: Windows XP, Windows NT, Windows 20xx, Windows 10, Mac OS, DragonFlyBSD.

OS Kernel Function

- Manages a processes
  - Control execution of processes and threads
  - Creation, termination, communication (IPC)
  - Schedules processes for execution on the CPU(s)
- Manages memory
  - Allocates memory for an executing process
  - Sets memory protection
  - Coordinates swapping pages of memory to a disk if low on memory
- Manages a file systems
  - Allocation and retrieval of disk data
  - Enforcing access permissions & mutual exclusion
- Manages a devices
  - Disk drives, networks, keyboards, displays, printers, … using device drivers
  - Enforces access permissions & mutual exclusion

OS Kernel Structure



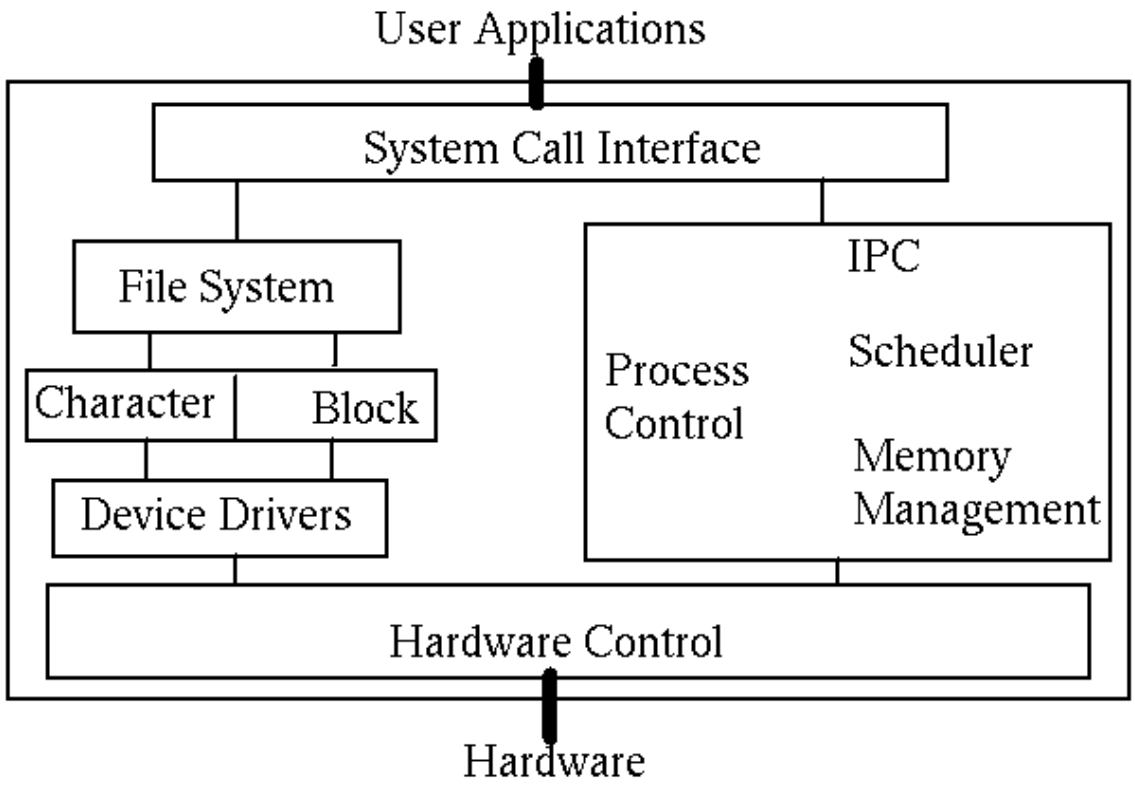
Monolithic Kernel

Modular Kernel

Micro Kernel

OS Kernel Structure

- UNIX Monolithic Kernel (Layered)
  - The traditional structure of a kernel is a layered system, such as Unix.
  - In this, all layers are part of the kernel, and each layer can talk to only a few other layers.
  - Application programs and utilities live above the kernel.

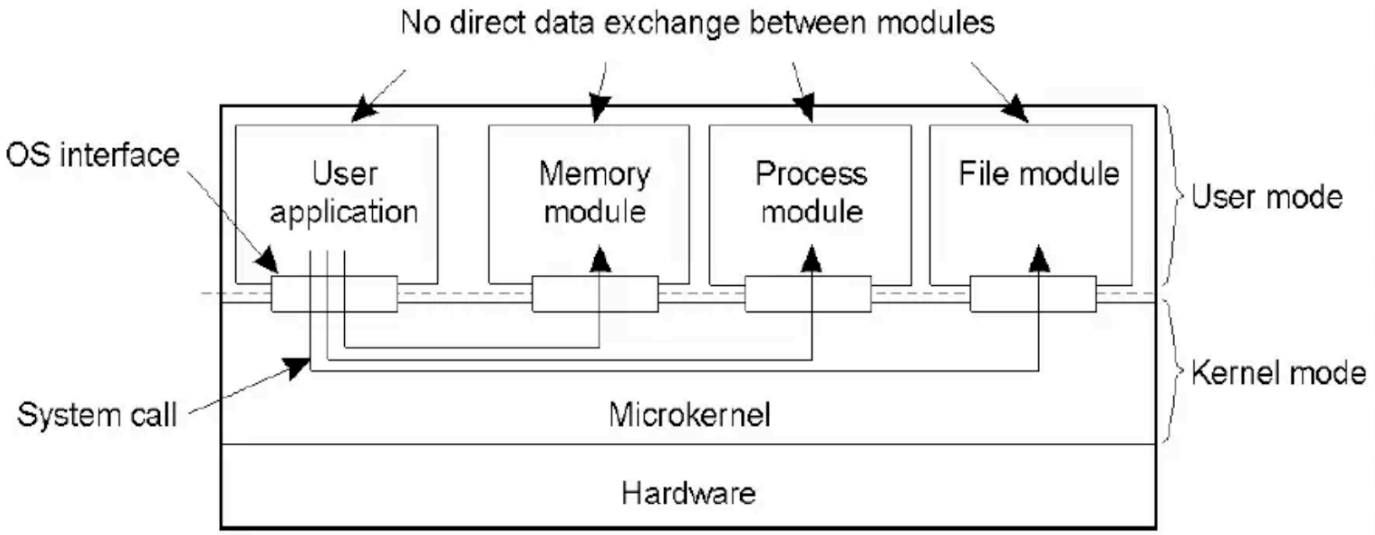


OS Kernel Structure

- Advantages of Monolithic kernel
  - Simple structure.
  - Communication between components very fast.
  - Fastest operating system.
- Disadvantages of Monolithic kernel
  - Code written in this operating system (OS) is difficult to port.
  - Monolithic OS has more tendency to generate errors and bugs. The reason is that user processes use same address locations as the kernel.
  - Adding and removing features from monolithic OS is very difficult. All the code needs to be rewritten and recompiled to add or remove any feature.
- Modular kernel
  - The modular core is a modern, advanced modification of the architecture of monolithic kernels. Smallest error and size then monolithic kernel.
  - Do not require a complete recompilation of the kernel when changing the composition of the computer hardware, provide one mechanism for loading kernel modules that support hardware (for example, drivers).

OS Kernel Structure

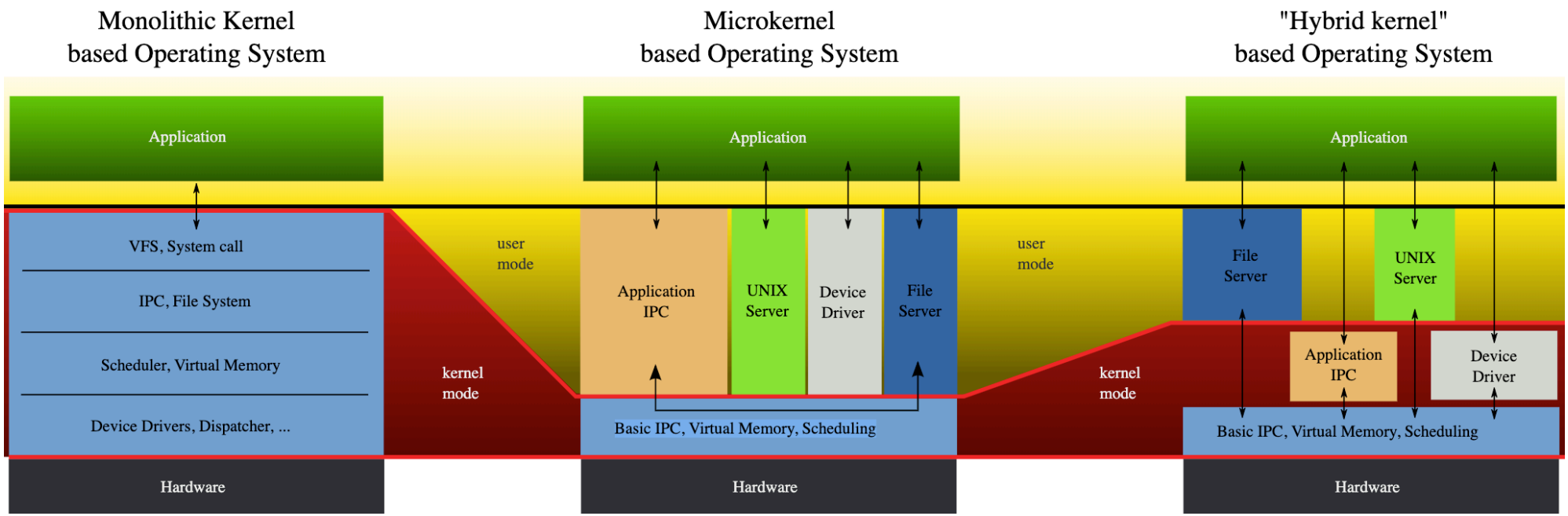
- Micro Kernel
  - Most of the Modern Operating Systems use a microkernel.
  - Many traditional services are made into user level services.
  - Micro Kernel is slower than Monolithic or Modular Kernels.



- Hybrid Kernel
  - Some systems use a mixed approach, use both modularity and microkernel.
  - Hybrid kernel often emphasizes not only the advantages, but also the shortcomings of both types of kernel.

OS Kernel Structure

IPC - inter-process communication



OS Kernel Structure

Basic for Comparision	Mikrokernel	Monolithic Kernel
Size	Microkernel is smaller in size	
Execution	Slow	Fast
Extendible	It is easily extendible	It is hard to extend
Security	If a service crashes, it does effects on working on the microkernel	If a service crashes, the whole system crashes in monolithic kemel
Code	To write a microkernel less code is required	To write a monolithic kernel more code is required
Example	QNX, Symbian, L4Linux, etc.	Linux, BSDs (FreeBSD, OpenBSD, NetBSD, etc.

Execution: User Mode vs. Kernel Mode

- Kernel mode = privileged, system, supervisor mode – Access restricted regions of memory
  - Modify the memory management unit
  - Set timers
  - Define interrupt vectors
  - Halt the processor
  - Etc.
- CPU knows what mode it's in via a status register
  - You can set the register in kernel mode
  - OS & boot loaders run in kernel mode
  - User programs run in user mode

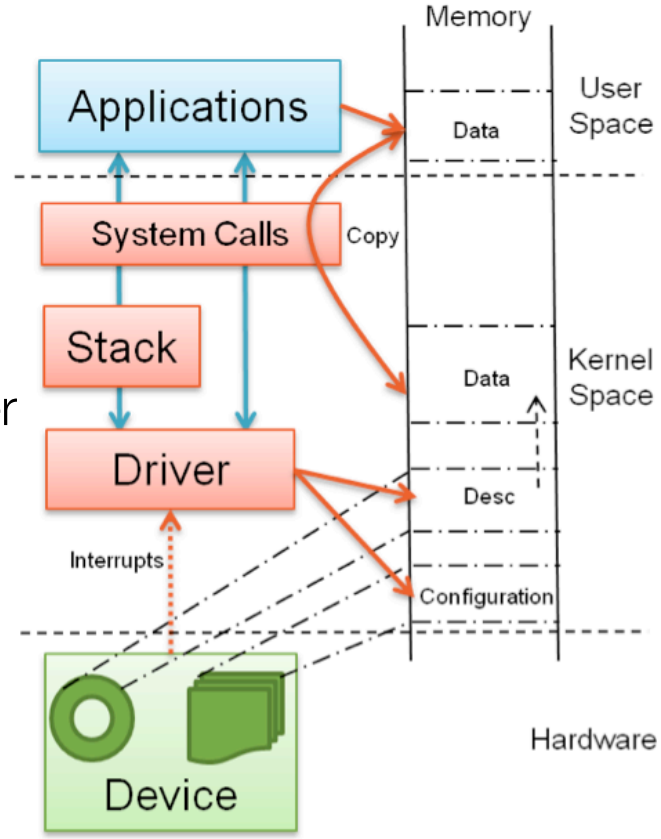


Figure 1: Kernel space network driver

How do you get to kernel mode?

- Trap: Transfer of control
  - Like a subroutine call (return address placed on stack)
  - Mode switch: user mode → kernel mode
- Interrupt Vector Table
  - Configured by kernel at boot time
  - Depending on architecture
    - Code entry points
      - Control jumps to an entry in the table based on trap number
      - Table will contain a set of JMP instructions to different *handlers* in the kernel
    - List of addresses
      - Each entry contains a structure that defines the target address & privilege level
      - Table will contain a set of addresses for different *handlers* in the kernel
- Returning back to user mode
  - *Return from exception*

How do you get to kernel mode?

Three types of traps:

1. Software interrupt – explicit instruction
   - Intel architecture: INT instruction (interrupt)
   - ARM architecture: SWI instruction (software interrupt)
2. Violation
3. Hardware interrupt

Traps give us a mechanism to transfer to *well-defined* entry points in the kernel

System Calls: Interacting with the OS

- A system call is a way for a user program to request services from the operating system
  - The operating system remains in control of devices
  - Enforces policies

- Use *trap* mechanism to switch to the kernel
  - User ↔Kernel mode switch: Mode switch
  - Note: most architectures support an optimized trap for system calls
    - Intel: SYSENTER/SYSEXIT
    - AMD: SYSCALL/SYSRET

System Calls: Interacting with the OS

- Use *trap* mechanism to switch to the kernel

- Pass a number that represents the OS service (e.g., *read*)
  - System call number; usually set in a register

- A system call does the following:
  - Set the system call number
  - Save parameters
  - Issue the trap (jump to kernel mode)
    - OS gets control
    - Saves registers, does the requested work
    - Return from exception (back to user mode)
  - Retrieve results and return them to the calling function

- System call interfaces are encapsulated as library functions

Regaining control: Timer interrupts

- How do we ensure that the OS can get control?
  - If your process is running, the operating system is *not* running
- Program a timer interrupt

- Crucial for:
  - Preempting a running process to give someone else a chance (force a context switch)
    - Including ability to kill the process
  - Giving the OS a chance to poll hardware
  - OS bookkeeping

Timer interrupts

- Windows
  - Typically 64 or 100 interrupts per second
  - Apps can raise this to 1024 interrupts per second
- Linux
  - Interrupts from Programmable Interval Timer (PIT) or HPET (High Precision Event Timer) and from a local APIC timer (one per CPU)
  - Interrupt frequency varies per kernel and configuration
    - Linux 2.4: 100 Hz
    - Linux 2.6.0 – 2.6.13: 1000 Hz
    - Linux 2.6.14+ : 250 Hz
    - Linux 2.6.18 and beyond: aperiodic – tickless kernel
      - PIT not used for periodic interrupts; just APIC timer interrupts

Context switch & Mode switch

- An interrupt or trap results in a *mode switch*:
- An operating system may choose to save a process' state and restore another process' state → preemption
  - Context switch
  - Save all registers (including stack pointers, PC, and flags)
  - Load saved registers (including SP, PC, flags)
  - To return to original context: restore registers and return from exception
- Context switch:
  - Switch to kernel mode
  - Save state so that it can be restored later
  - Load another process' saved state
  - Return (to the restored process)

Devices

- Character: mice, keyboard, audio, scanner
  - *Byte streams*
- Block: disk drives, flash memory
  - *Addressable blocks (suitable for caching)*
- Network: Ethernet & wireless networks
  - *Packet based I/O*
- Bus controllers
  - *Interface with communication busses*

Interacting with devices

- Devices have command registers
  - *Transmit, receive, data ready, read, write, seek, status*
- Memory mapped I/O
  - Map device registers into memory
  - Memory protection now protects device access
  - Standard memory load/store instructions can be used to interact with the device

Getting data to/from devices

- When is the device ready?
  - Polling
    - Wait for device to be ready
    - To avoid busy loop, check each clock interrupt
  - Interrupts from the device
    - Interrupt when device has data or when the device is done transmitting
    - No checking needed – but context switch may be costly

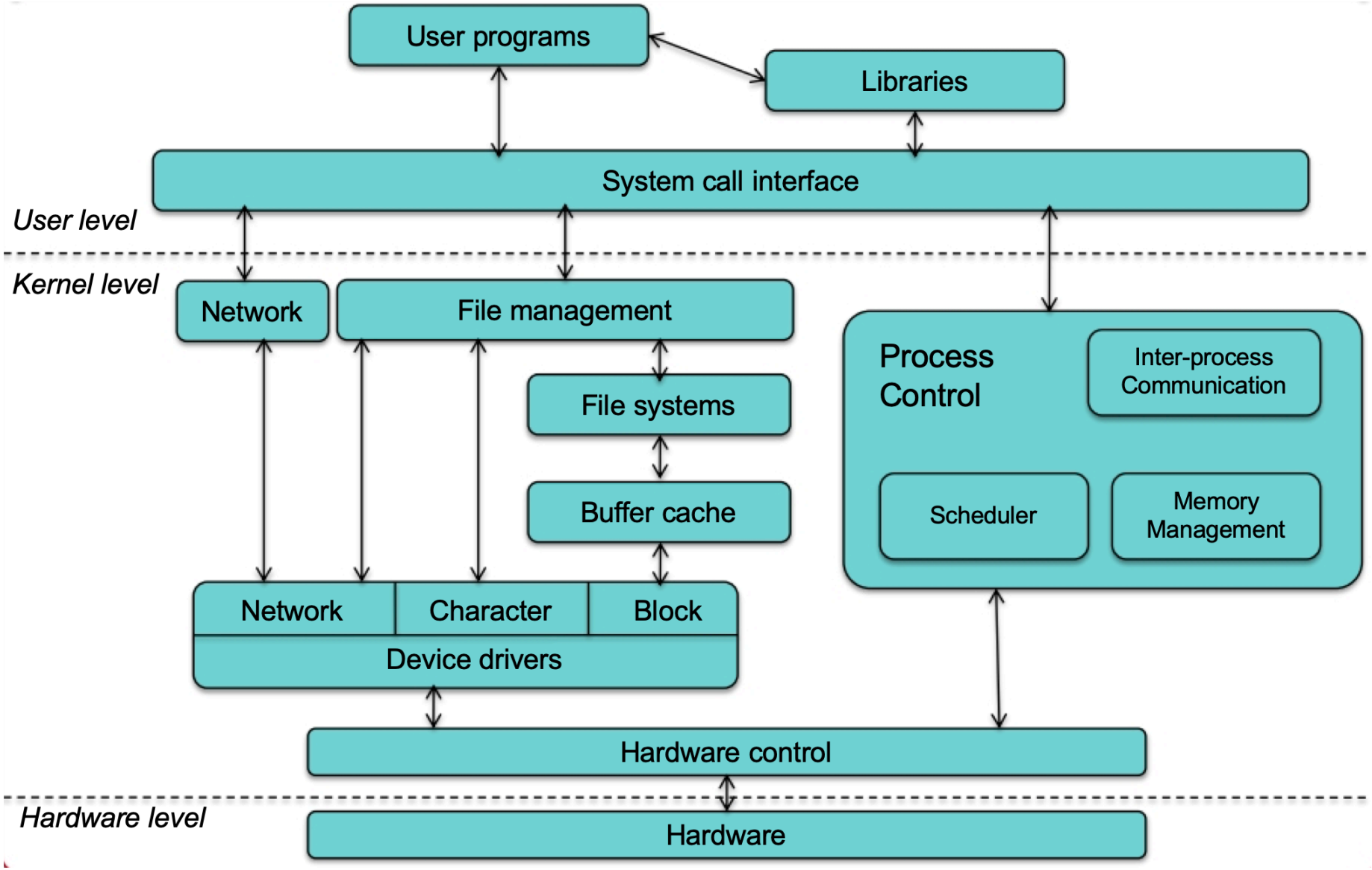
Getting data to/from devices

- How do you move data?
  - Programmed I/O (PIO)
    - Use memory-mapped device registers
    - The processor is responsible for transferring data to/from the device by writing/reading these registers
  - DMA
    - Allow the device to access system memory directly

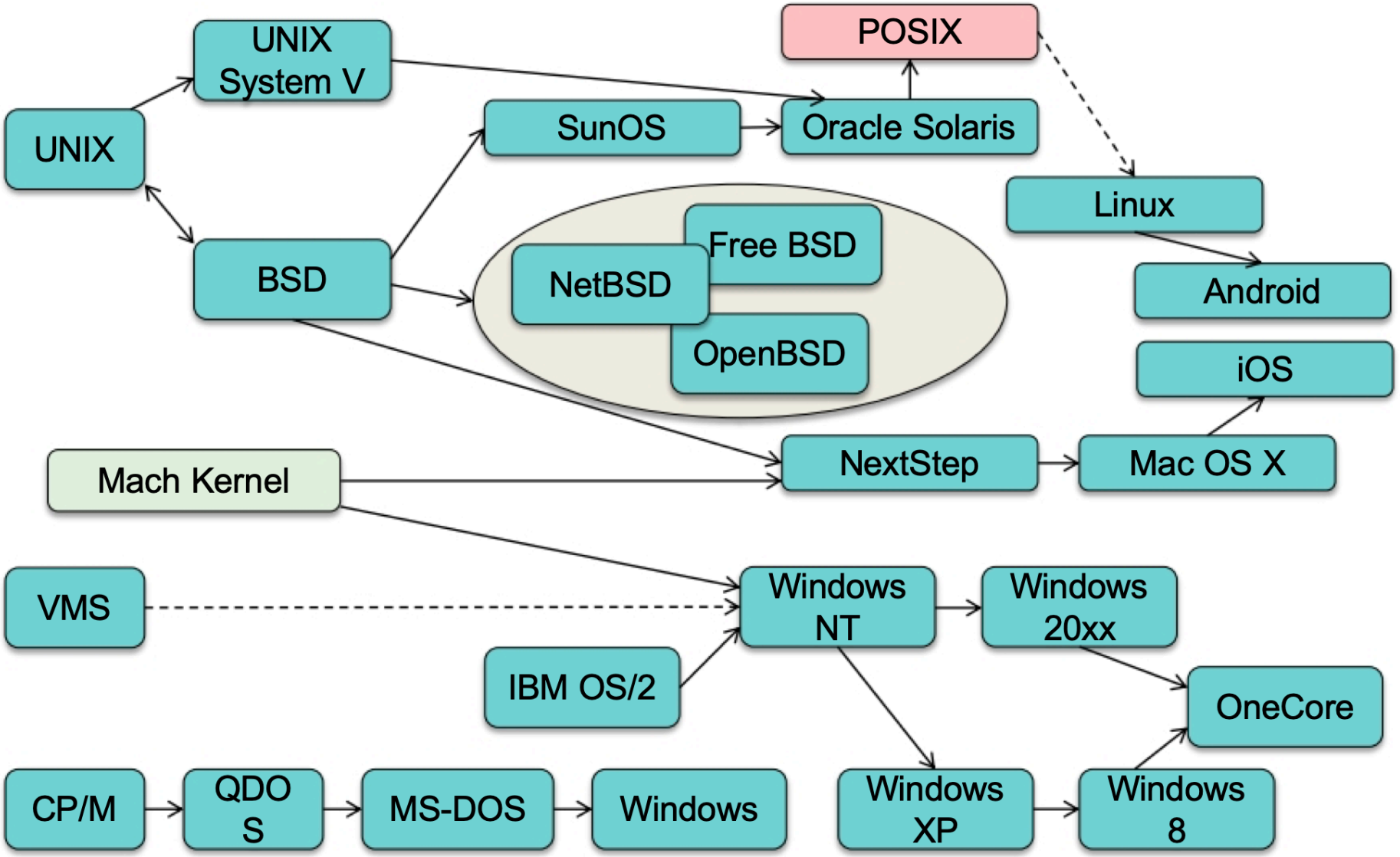
Files and file systems

- Persistent storage of data
  - Handle allocation of disk space
- Provide user-friendly names to identify the data
- Associate attributes with the data
  - Create time, access time, owner, permissions, …
  - Device or data file?

Structure of an operating system



UNIX? NT? POSIX?



POSIX

- UNIX → POSIX (IEEE interface specification)
  - Portable OS Interface UNIX Like
- IEEE (ISO/IEC 9945): defines POSIX environment
  - System interfaces
  - Shell & scripting interface
  - Common utilities
  - Networking interfaces
  - Security interfaces
- POSIX (or close to) systems include
  - Solaris, BSD, Mac OS X, VxWorks, Microsoft Windows Services for UNIX
  - Linux, FreeBSD, NetBSD, OpenBSD, BeOS

3. OS Mechanisms & Policies

OS Mechanisms & Policies

- Mechanisms:
  - Presentation of a software abstraction:
    - Memory, data blocks, network access, processes
- Policies:
  - Procedures that define the behavior of the mechanism
    - Allocation of memory regions, replacement policy of data blocks
- Permissions
  - Enforcement of access rights
- Keep mechanisms, policies, and permissions separate

Processes

- Mechanism:
  - Create, terminate, suspend, switch, communicate
- Policy
  - Who is allowed to create and destroy processes?
  - What is the limit?
  - What processes can communicate?
  - Who gets priority?
- Permissions
  - Is the process making the request allowed to perform the operation?

Threads

- Mechanism:
  - Create, terminate, suspend, switch, synchronize
- Policy
  - Who is allowed to create and destroy threads?
  - What is the limit?
  - How do you assign threads to processors?
  - How do you schedule the CPU among threads of the same process?

Virtual Memory

- Mechanism:
  - Logical to physical address mapping

- Policy
  - How do you allocate physical memory among processes and among users?
  - How do you share physical memory among processes?
  - Whose memory do you purge when you're running low?

File Systems

- Mechanism:
  - Create, delete, read, write, share files
  - Manage a cache; memory map files

- Policy
  - What protection mechanisms do you enforce?
  - What disk blocks do you allocate?
  - How do you manage cached blocks of data (Per file? Per user? Per process?)

Messages

- Mechanism:
  - Send, receive, retransmit, buffer bytes
- Policy
  - Congestion control, dropping packets, routing, prioritization, multiplexing

Character Devices

- Mechanism:
  - Read, write, change device options
- Policy
  - Who is allowed to access the device?
  - Is sharing permitted?
  - How do you schedule device access?

The End