Operating systems

Lecture 1, Michal Vrábel, 9/10/2019

Who Am I?

- Daniel Gecášek
  - daniel.gecasek@tuke.sk
- In case I’m not available, talk to Ján Genči
  - genci@tuke.sk

Grading

- Final oral exam 60%
- Exercises 40%
  - Homeworks

The book

- Operating Systems - Internals and Design Principles 7th ed - W. Stallings (Pearson, 2012) BBS
  - https://github.com/jyfc/ebook
- For exercises – nothing strict – for example
  - Linux System Programming: Talking Directly to the Kernel and C Library
    - https://github.com/vrnithinkumar/S5_OS_lab/

Operating System

- No universally accepted definition
- **OS is a resource allocator** - Manages all resources, Decides between conflicting requests for efficient and fair resource use
- **OS is a control program** - Controls execution of programs to prevent errors and improper use of the computer
- “The one program running at all times on the computer” is the kernel.
  - Everything else is either a system program (ships with the operating system) or an application program

History

- Longer text (will be provided)
  - P01-01---0201180383---An_introduction_to_operating_systems--Chapter-01-History
- **1940s-1950s.**
  - No operating systems
  - Entering programs one bit at a time → Punched cards
  - Assembly
  - Serial processing, direct access to procesor
  - First OS – one job at a time, transition between jobs

History

- **1960s**
  - Batch processing systems
    - The central idea behind the simple batch-processing scheme is the use of a piece of software known as the monitor - resident monitor.
    - job control language (JCL)
  - Card readers, card punchers, printers, tape drives
  - Multiprogramming – several jobs in main memory
    - https://en.wikipedia.org/wiki/Computer_multitasking#Multiprogramming
  - Running batched jobs
  - Multiple interactive users
    - Terminals
    - Timesharing
    - https://en.wikipedia.org/wiki/Time-sharing

History

- **1970s**
  - Emergence of software engineering (60s - 70s)
  - TCP/IP
  - Commercialization
- **1980s**
  - Personal computing, workstations
  - Networks – Server/client architecture
- **1990s**
  - Distributed systems
  - Internet
  - Information systems - Relational databases
- **2000s**
  - Internet
  - Mobile technologies
  - Massive parallelism
- **2010s**
  - Virtualization
  - Cloud computing
  - Ubiquitous computing

History - generations

- **0-th (1940s - 1950s)**
  - Electromechanical switching
- **1st (1950s - early 1960s)**
  - vacuum tubes
  - Assembly
  - Debugging tools
  - ALGOL, COBOL, FORTRAN
- **2nd (1960s)**
  - Transistors
  - Multiprograming
  - Terminals
  - Time-sharing
- **3rd (late 1960s - 1970s)**
  - Integrated circuits
  - Personal computers
  - Networking (TCP/IP)
  - Parallel systems
  - User friendly OS (maybe more 80s)
- **(3.5th?)**
  - Very Large Scale Integrated (VLSI) circuits
- **4th (1980s)**
  - Microprocessors
- **5th ?**
  - AI?, Change in the user interaction?

Operating system – resoure allocator

- OS is a resource allocator/manager - Manages all resources, Decides between conflicting requests for efficient and fair resource use

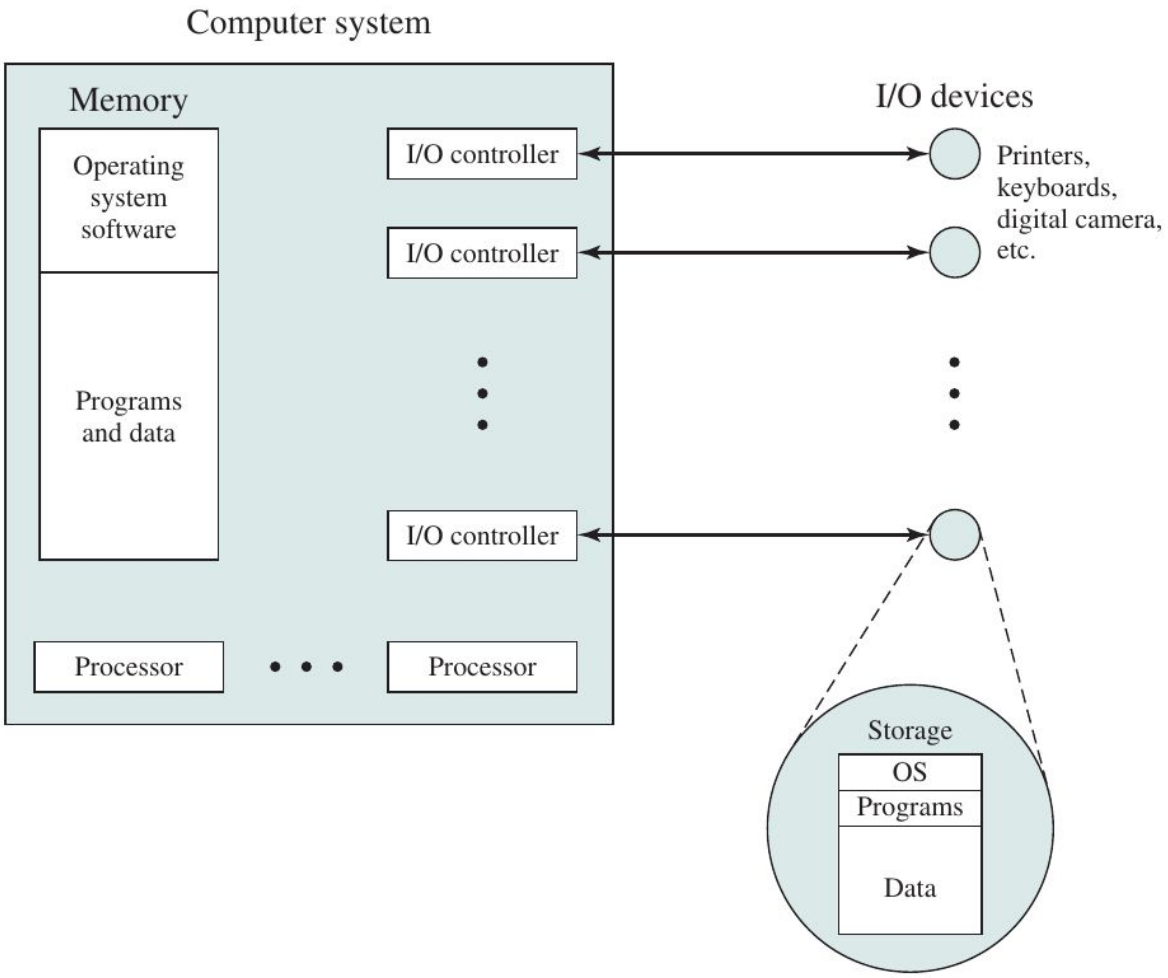


Figure 2.2 The Operating System as Resource Manager

Operating system – unifies the interface

- OS – unifies the interface for applications / for users (to a certain point)

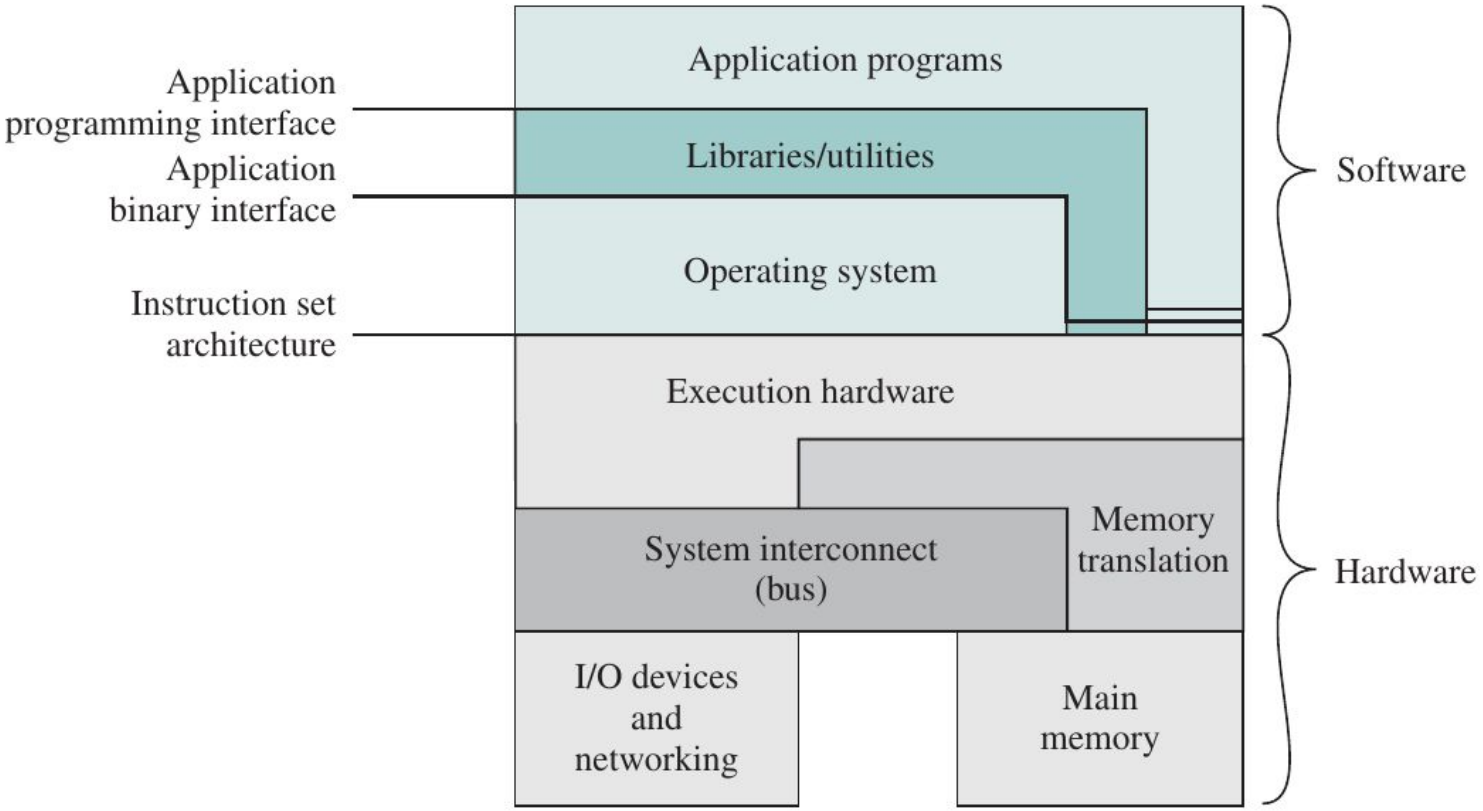


Figure 2.1 **Computer Hardware and Software Structure**

Operating system – unifies the interface

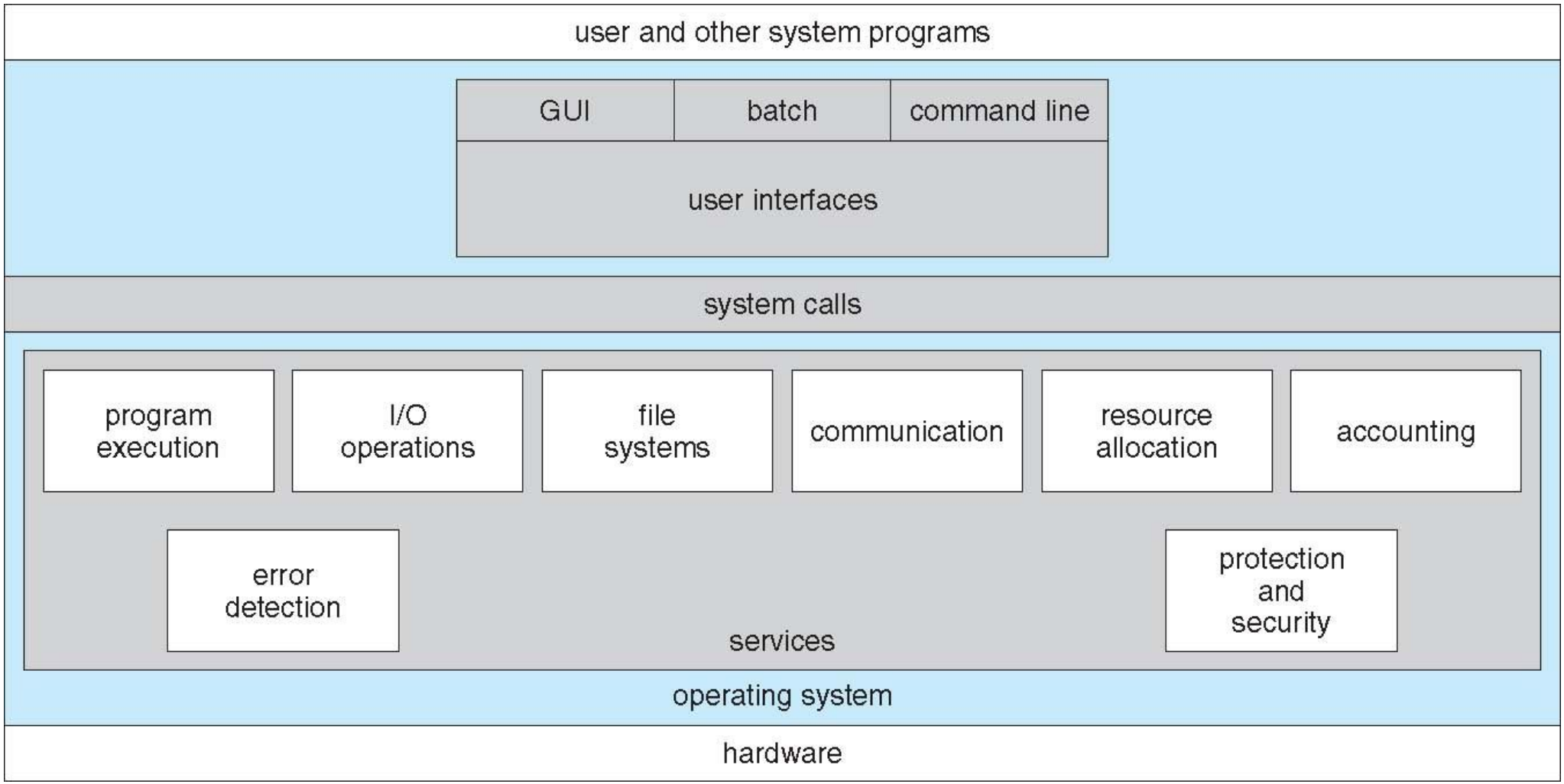
- **OS typically provides services in the following areas**
  - Program development
  - **Program execution**
  - **Access to I/O devices** - nature of the I/O device, the structure of the data contained
  - **Controlled access to files**
  - **System access** - For shared or public systems
  - **Error detection and response**
  - **Accounting** - usage statistics for various resources

- **Three key interfaces in a typical computer system**
  - Instruction set architecture (ISA)
  - Application binary interface (ABI)
  - Application programming interface (API)
    - system calls

Operating system – unifies the interface

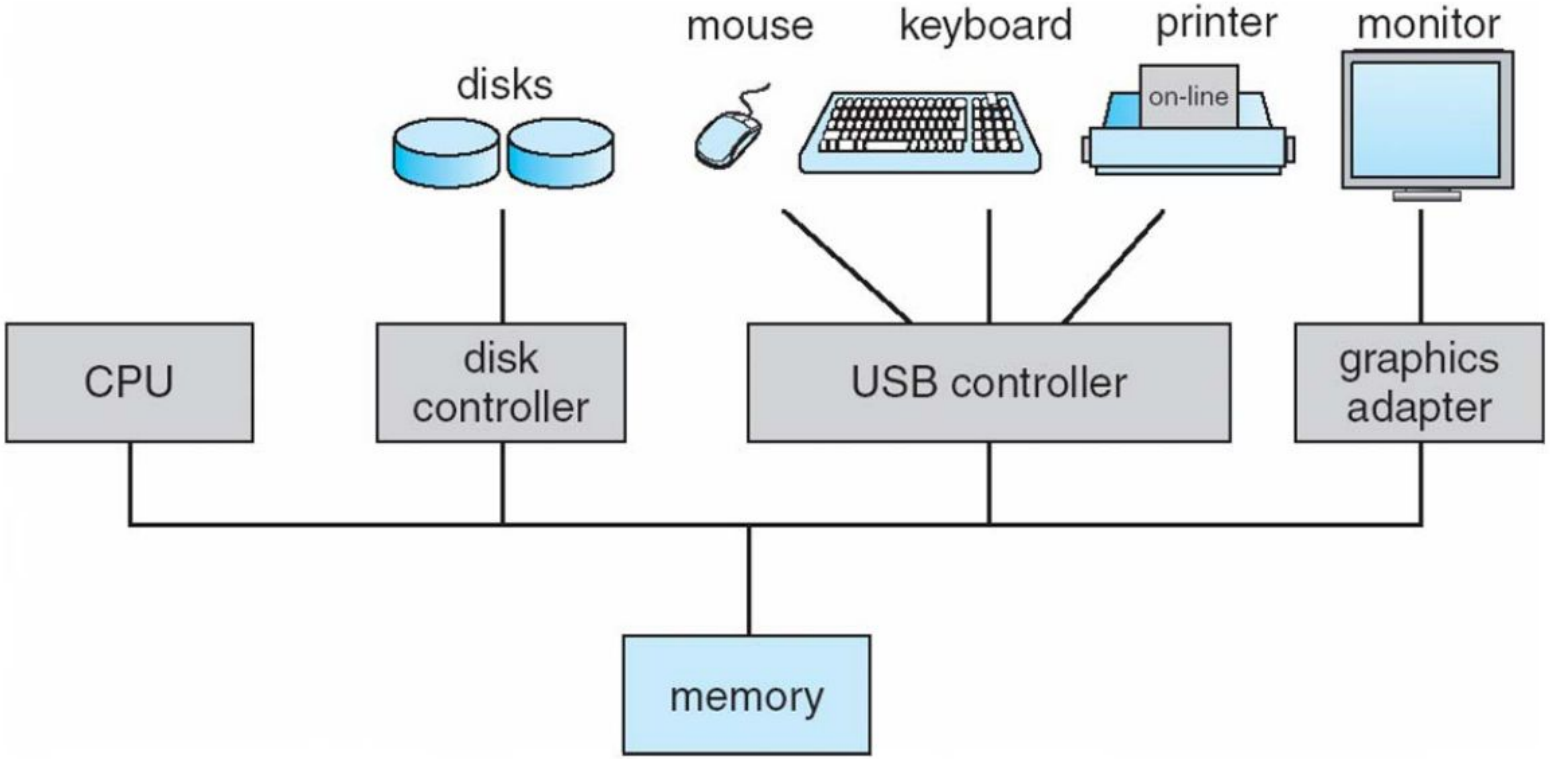
- **POSIX**
  - The Portable Operating System Interface (POSIX)
  - Exercises from the subject
- **POSIX-certified**
  - MacOS, HP-UX, Solaris
- **Mostly POSIX-compliant**
  - Linux, FreeBSD, Android, VMware ESXi
- **POSIX for Microsoft Windows**
  - Cygwin, MinGW, Windows Subsystem for Linux, Windows C Runtime Library

Operating system – unifies the interface



Basic elements of a Computer system

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Basic elements of a Computer system

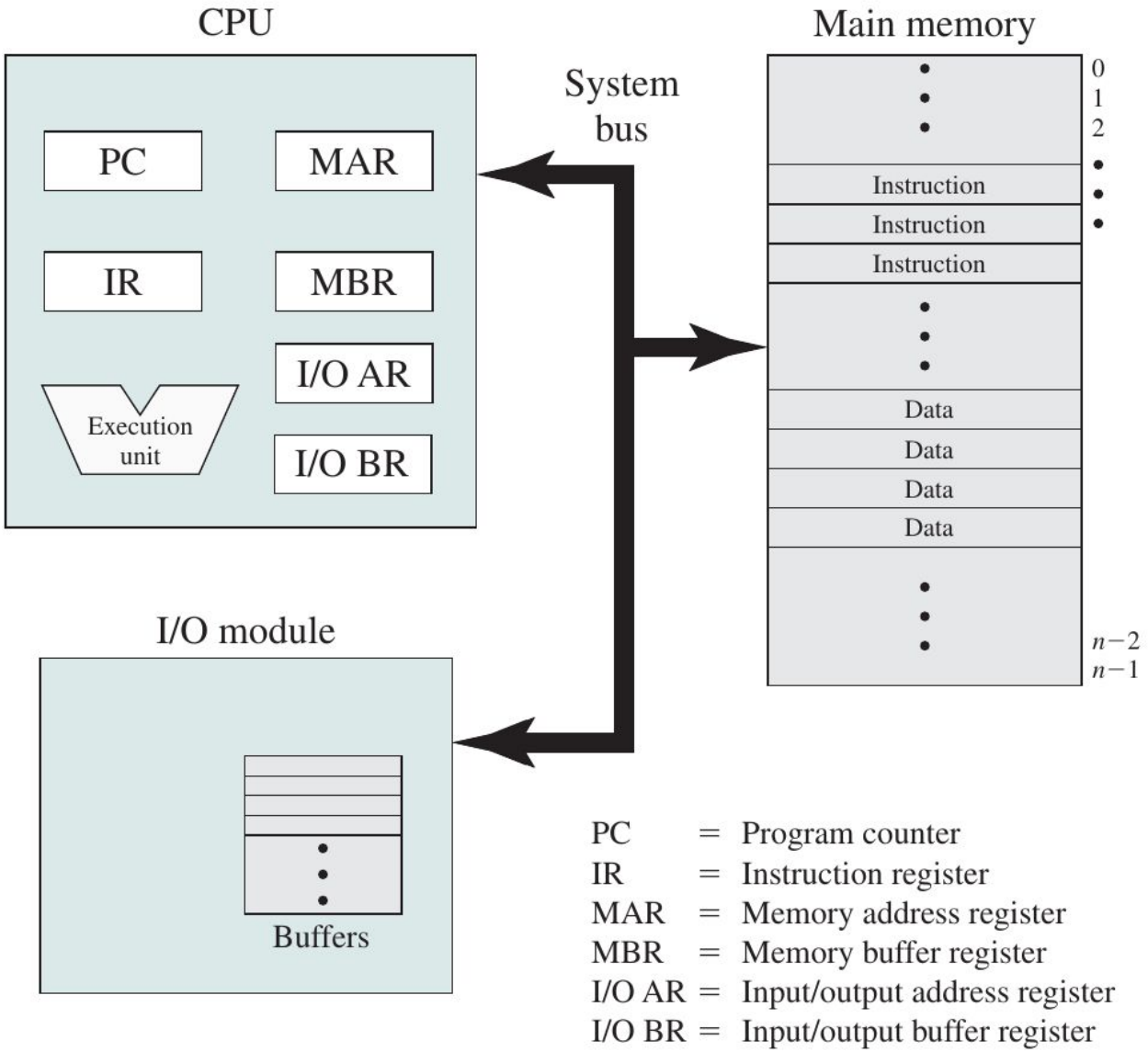


Figure 1.1 **Computer Components: Top-Level View**

In computer architecture, a bus (a contraction of the Latin omnibus) is a **communication system that transfers data between components** inside a computer, or between computers.

Basic elements of a Computer system

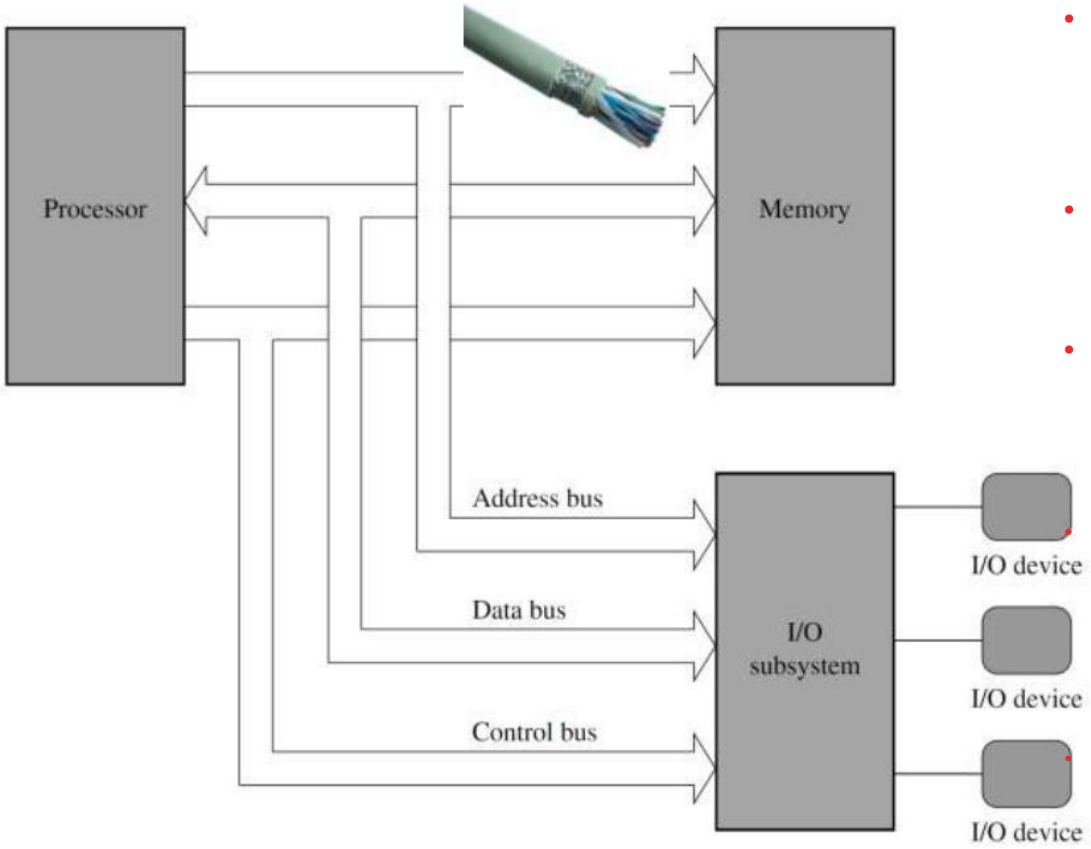


Figure 2.2 Simplified block diagram of a computer system.

[4]

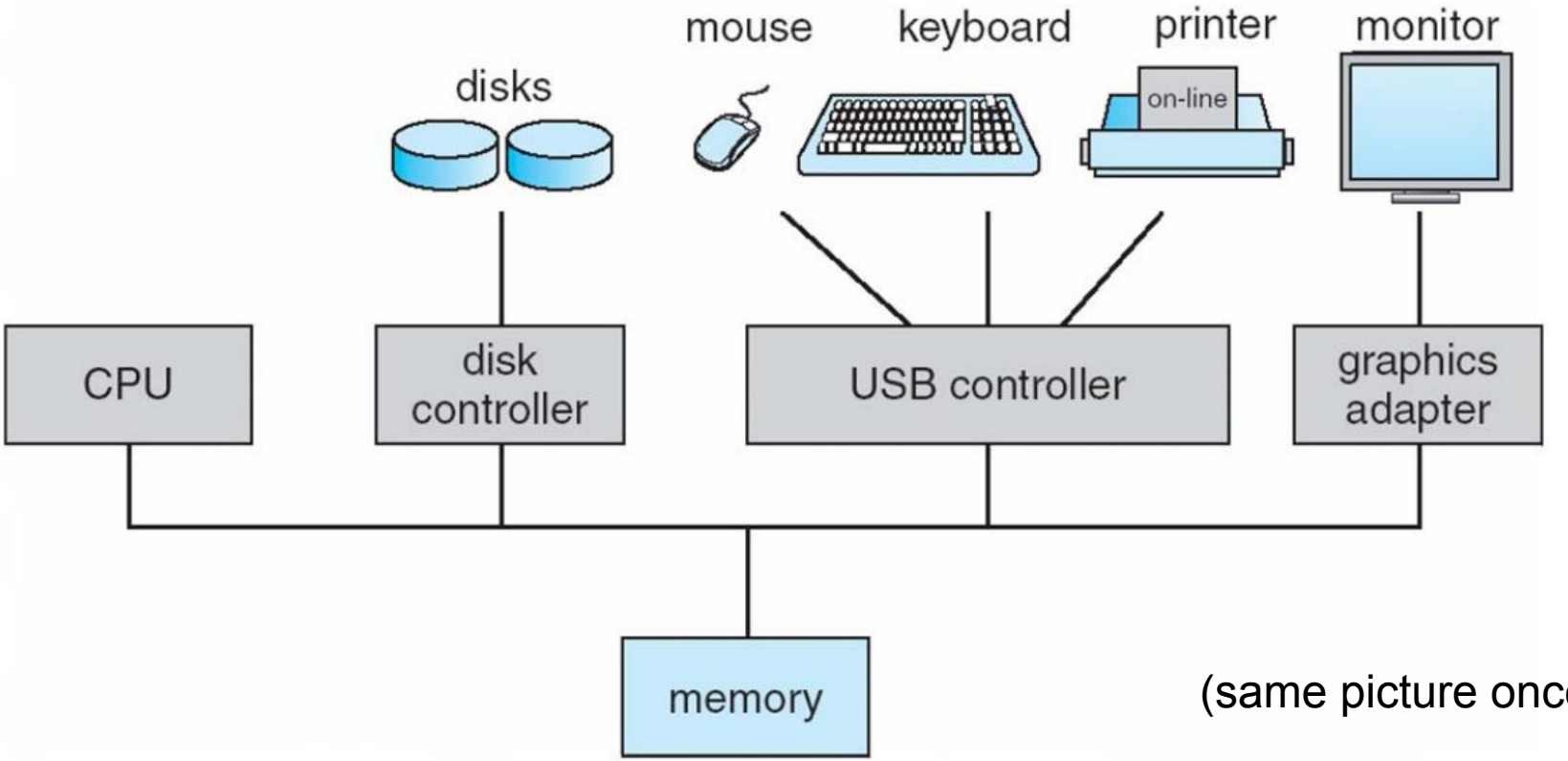
- While the **address bus carries the information about the device** with which the CPU is communicating and
- the **data bus carries the actual data** being processed,
- the **control bus carries commands** from the CPU and returns status signals from the devices.

An address bus is a bus that is used to specify a physical address. When a processor or DMA-enabled device needs to read or write to a memory location, it specifies that memory location on the address bus (the value to be read or written is sent on the data bus).

DMA https://en.wikipedia.org/wiki/Direct_memory_access, IOMMU https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit,

Computer-system operation

- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles

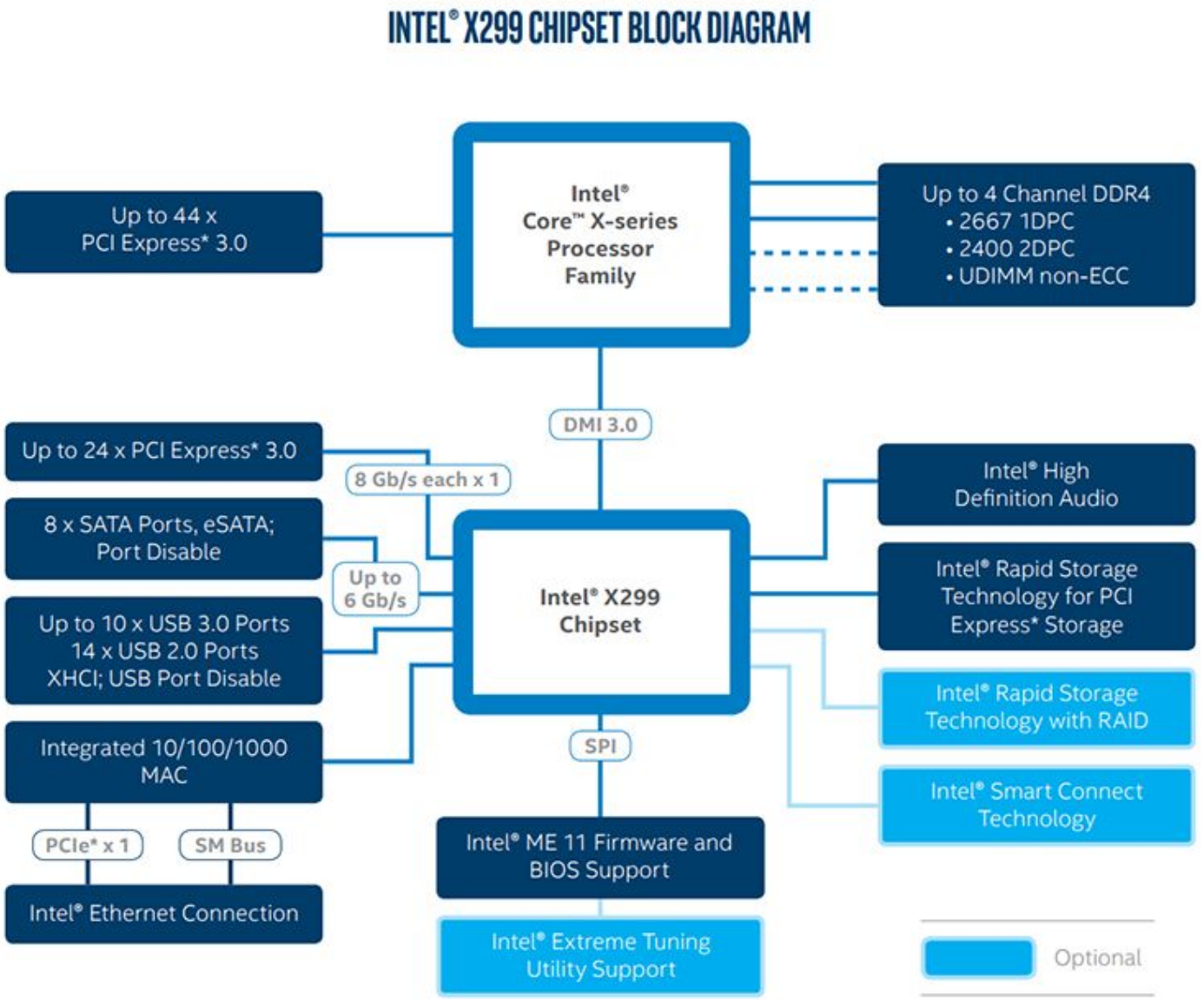


(same picture once again)

Computer-system operation

- I/O devices and the CPU can execute concurrently
- Each **device controller** is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**
-

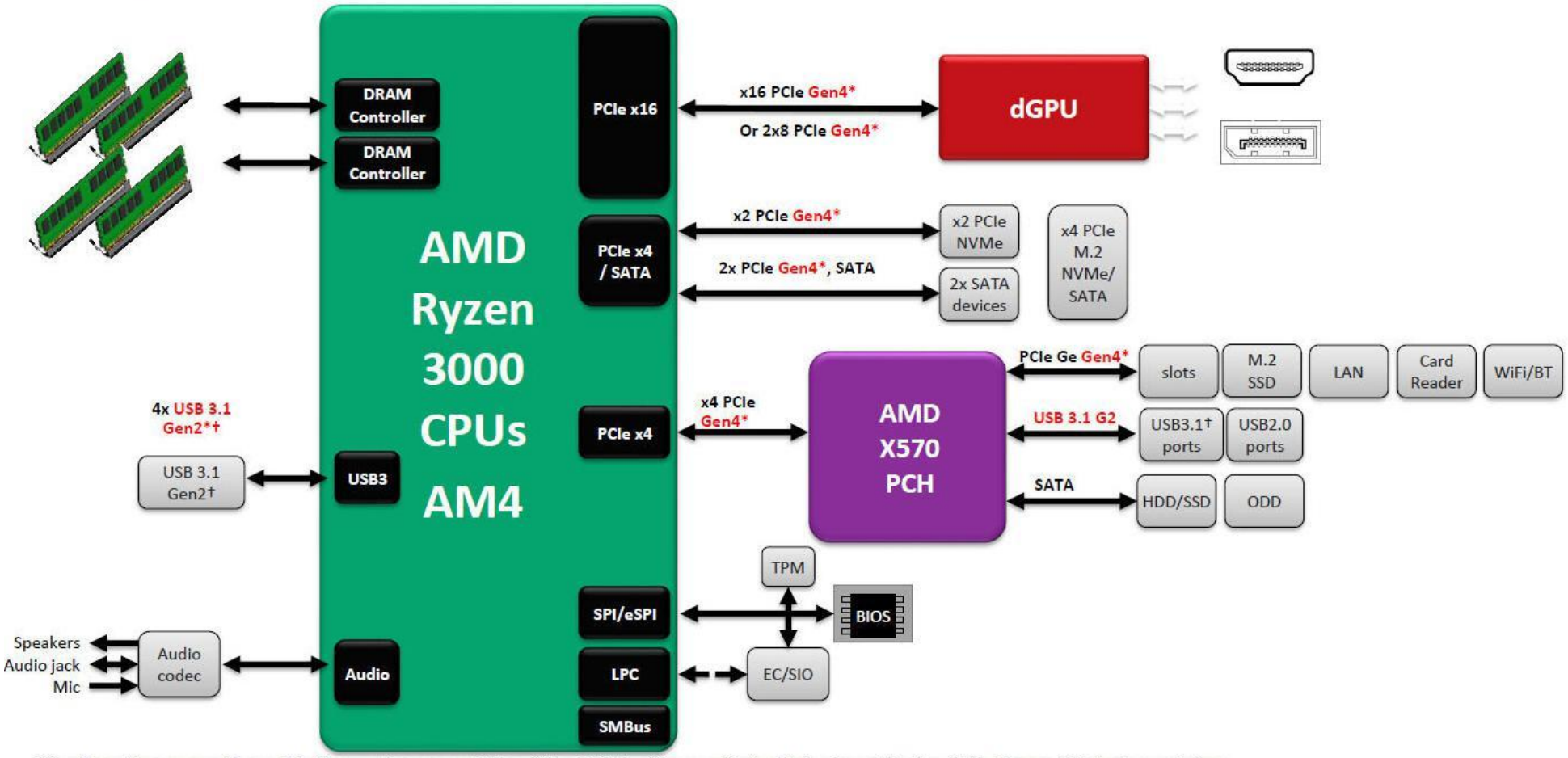
Computer-system operation - Examples



Computer-system operation - Examples

AM4 IO CONSISTENCY WITH PREMIUM CONNECTIVITY: 2019

2019 INTRODUCES PCI EXPRESS GEN4 AND EXTENDED USB3.1 GEN2 10G IO (RED TEXT*) FOR INCREASING PERIPHERAL AND PROCESSING BANDWIDTH REQUIREMENTS



*Configurations vary with model. Diagram is representative of Zen 2 CPUs. Always refer to Motherboard Design Guide for specific implementations.

† Does not support multi-lane or "lane bonding"

Processor – instruction cycle

- Processes instructions
- Compiler explorer https://godbolt.org/

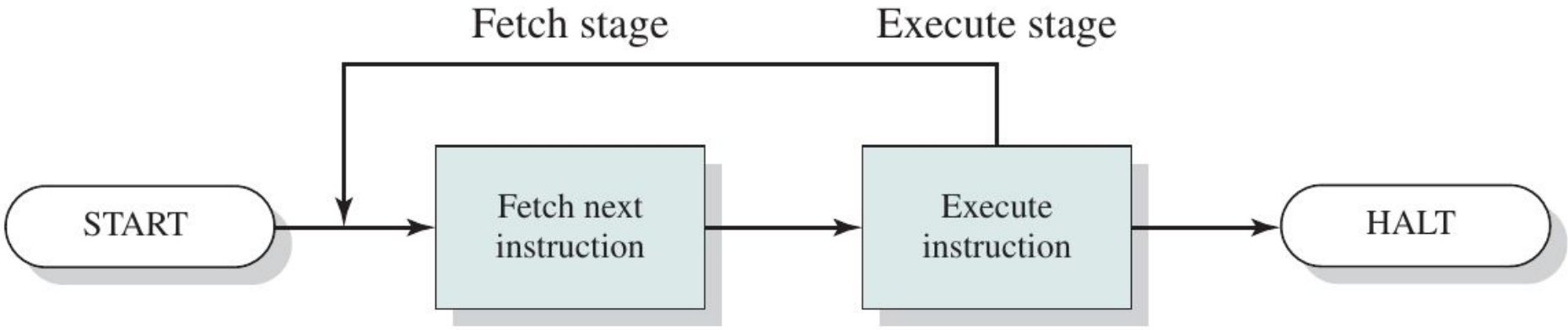


Figure 1.2 **Basic Instruction Cycle**

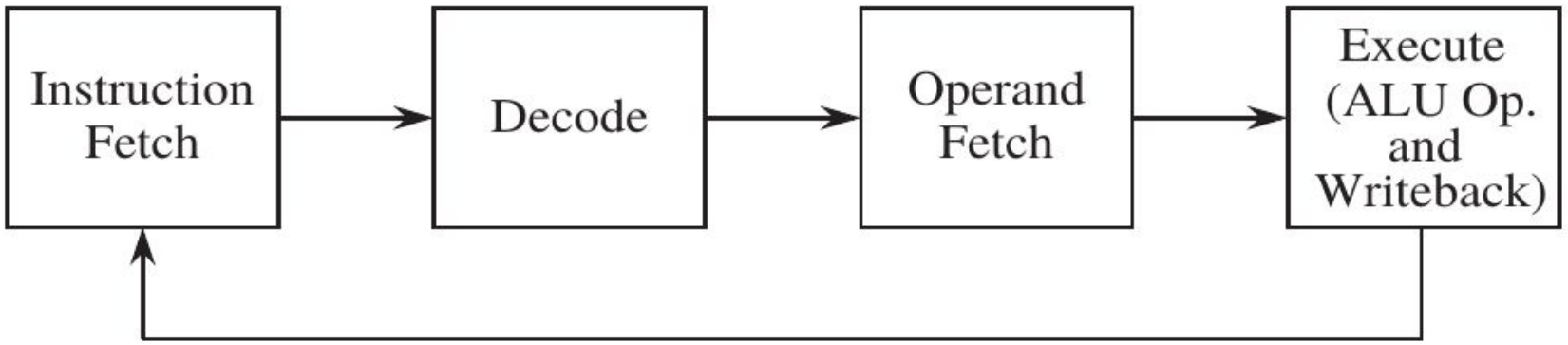
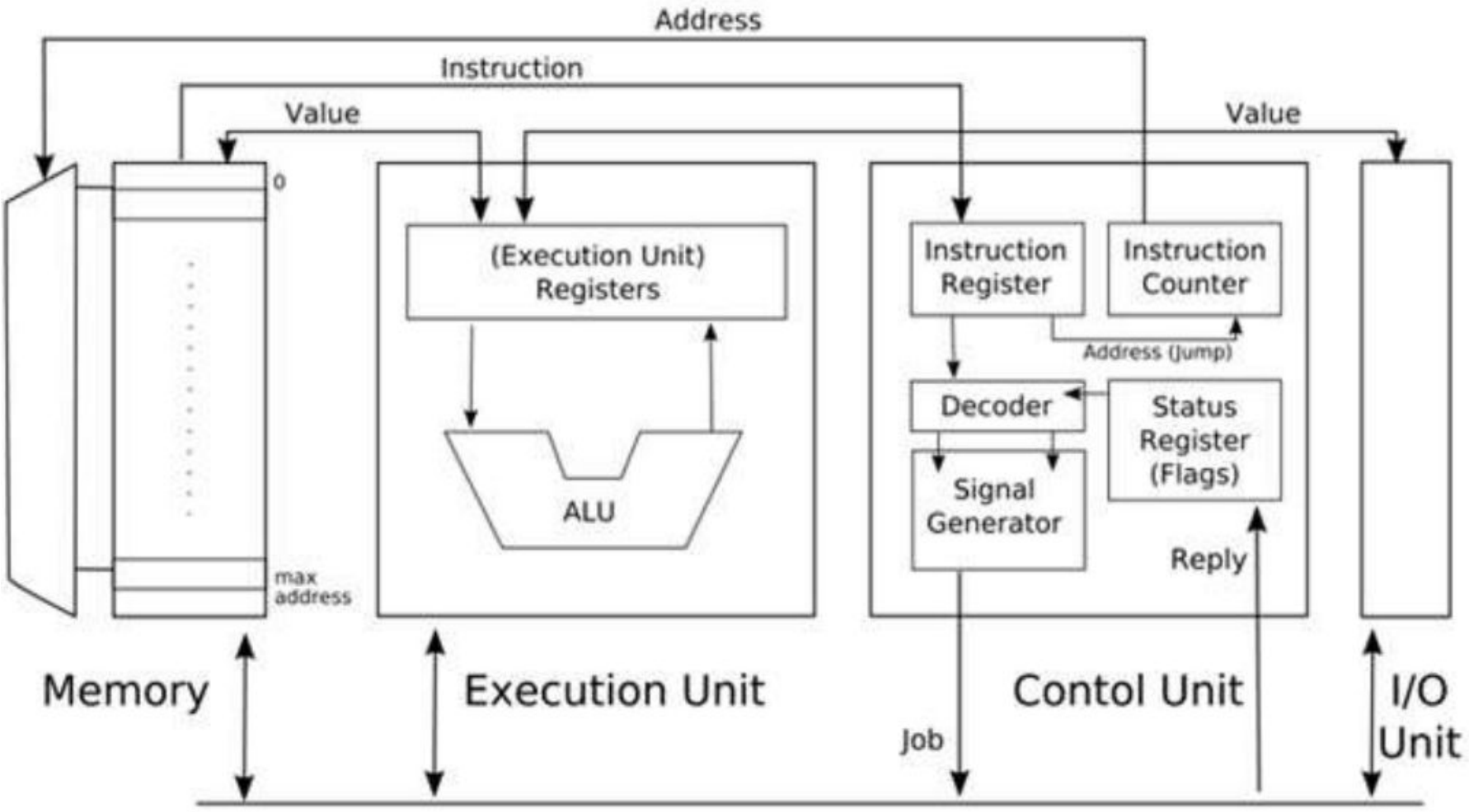


Figure 10-3 **Four-stage instruction pipeline.**

Processor – instruction cycle

(8086)



Processor – instruction cycle

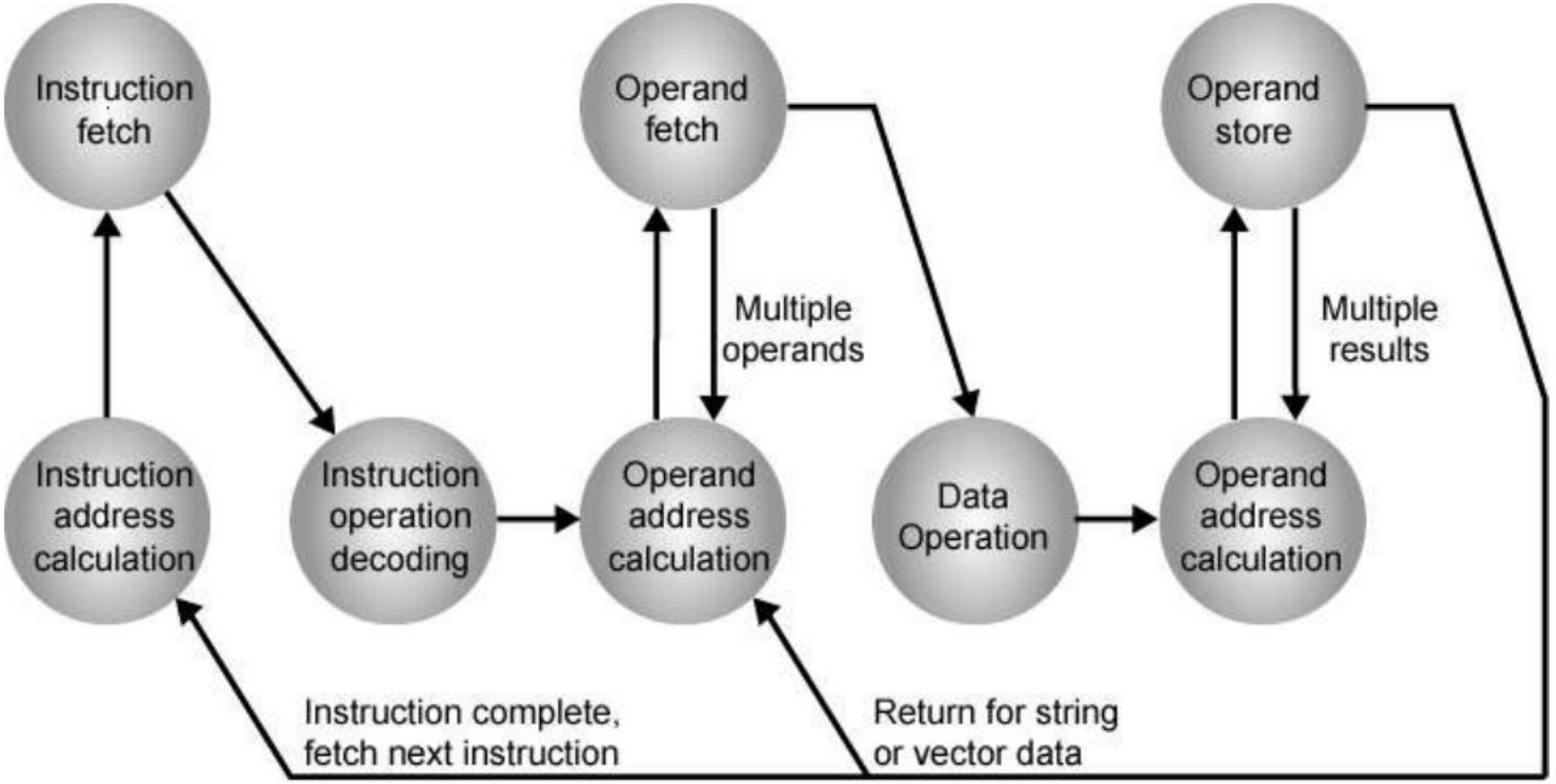


Figure 10.1 Instruction Cycle State Diagram

Processor - interrupts

- Interrupts are provided primarily as a way to **improve processor utilization**.
  - Example: I/O devices are much slower than the processor
- **Interrupt transfers control to the interrupt service routine** generally, through the interrupt vector, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- **An operating system is interrupt driven**

Processor - interrupts

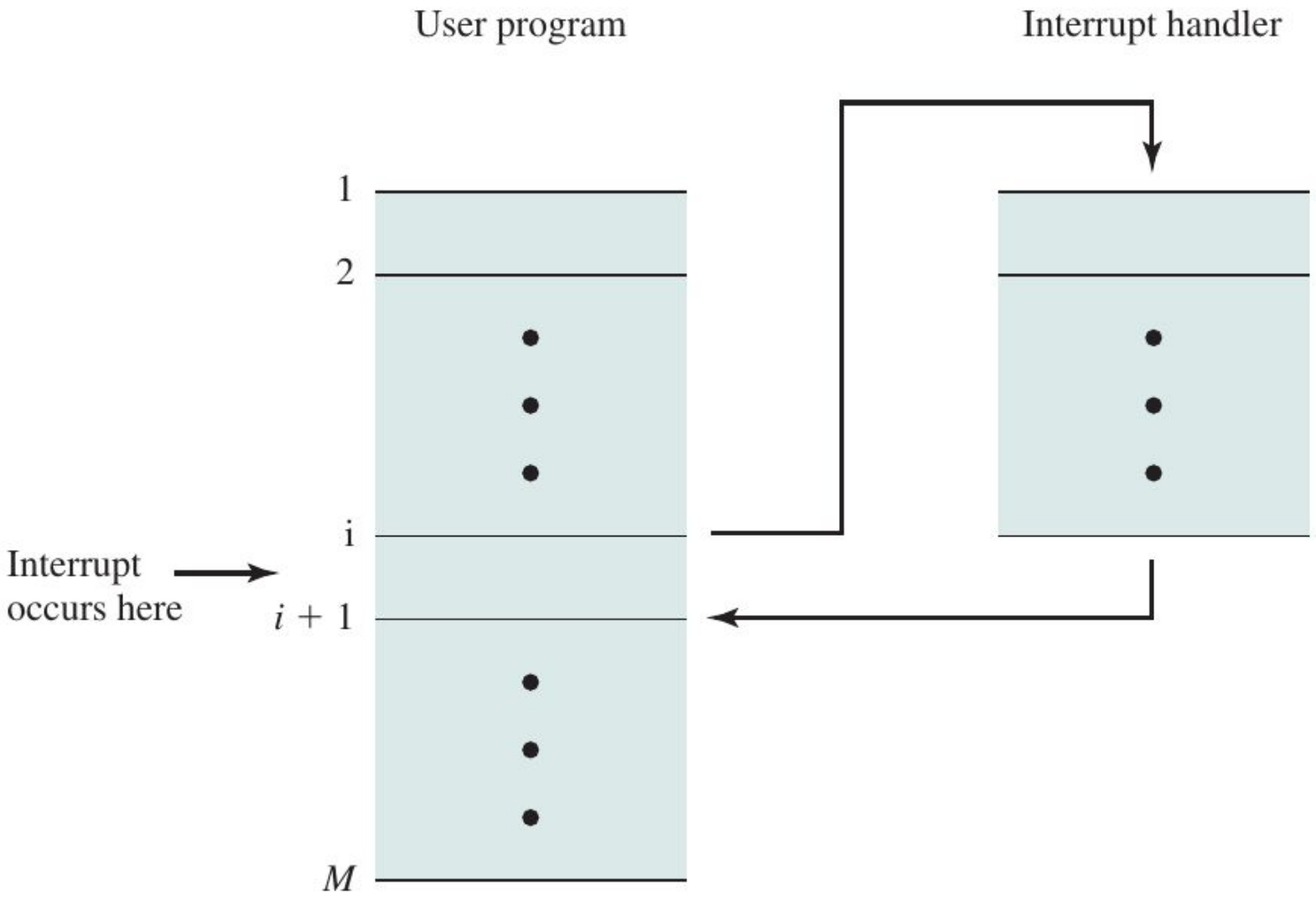


Figure 1.6 Transfer of Control via Interrupts

Processor - interrupts

- To accommodate interrupts, **an interrupt stage** is added to the instruction cycle
- If an interrupt is pending,the processor suspends execution of the current program and executes an interrupt-handler routine

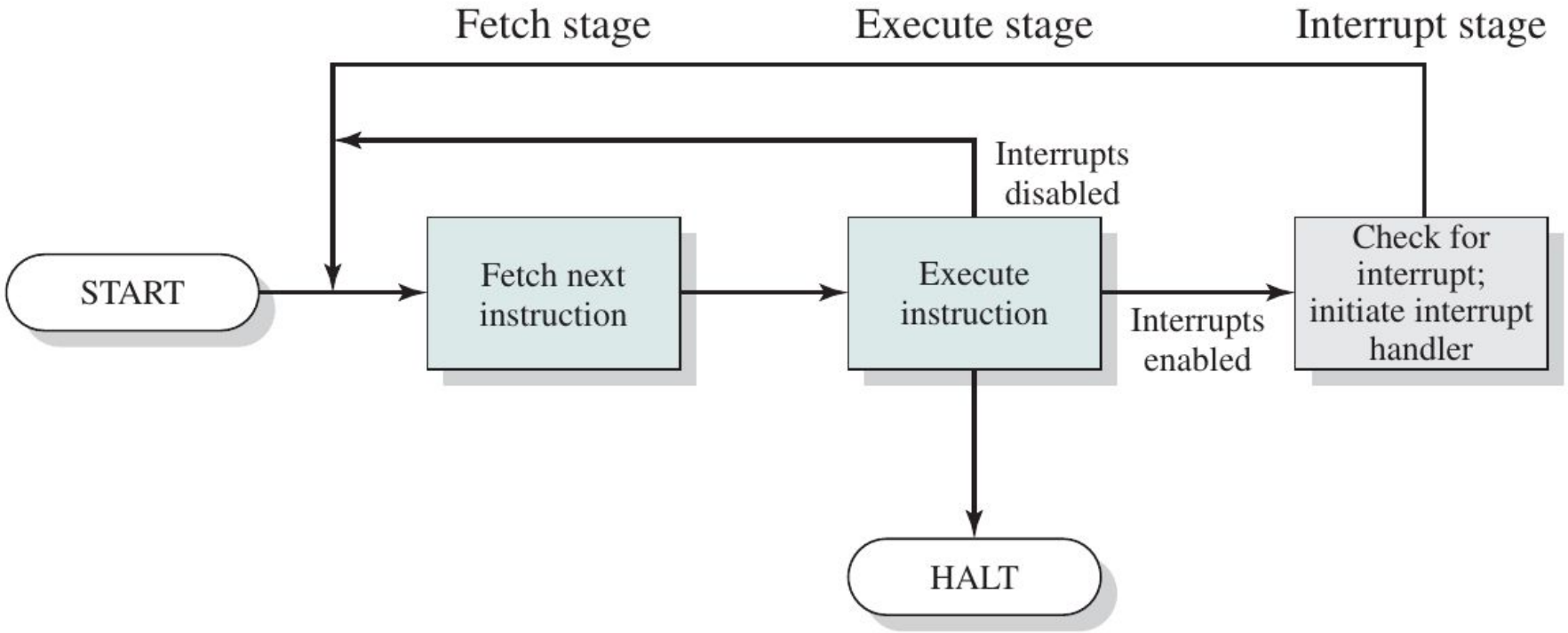
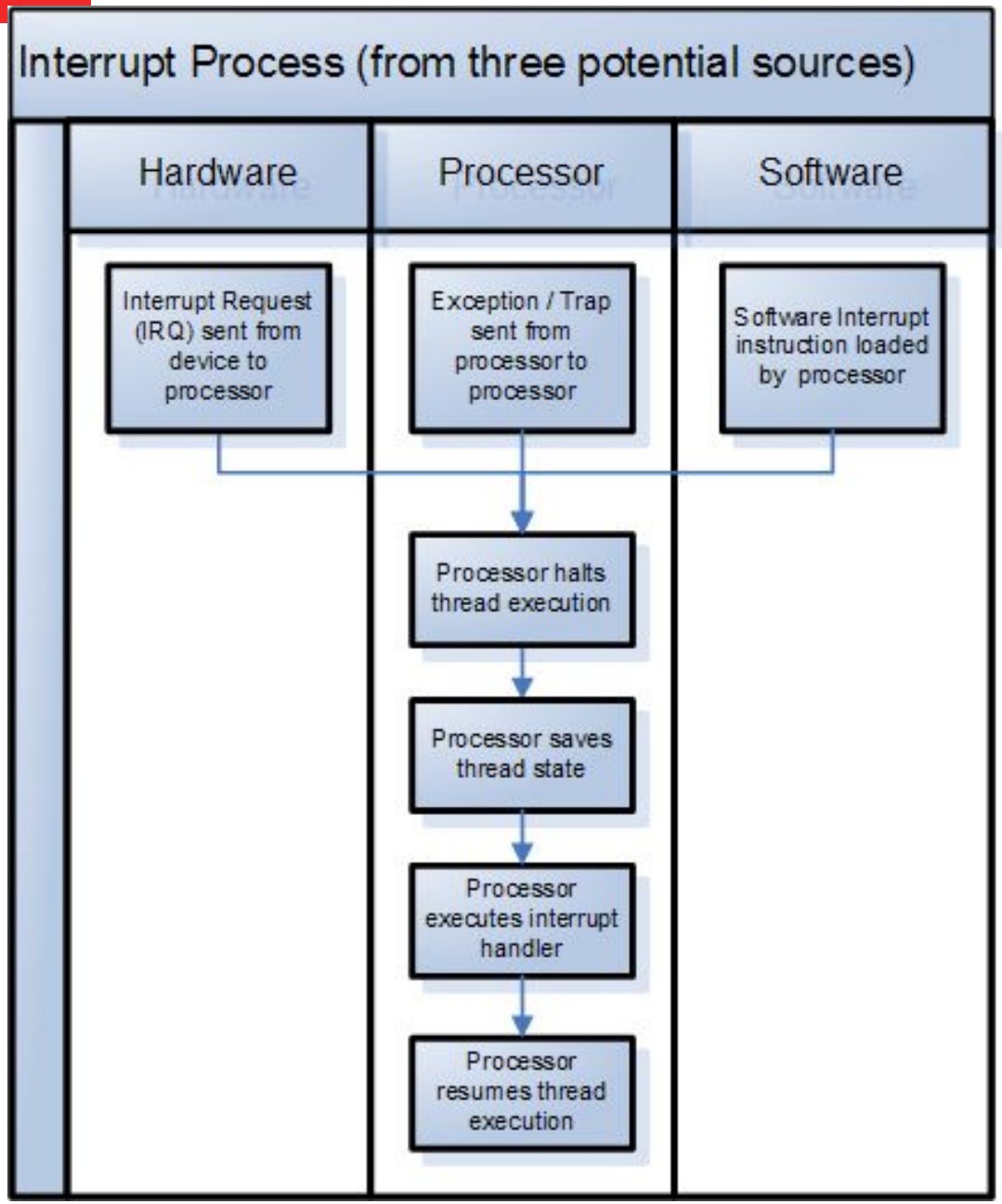


Figure 1.7 **Instruction Cycle with Interrupts**

Processor - interrupts



- **Hardware interrupts**
  - issued by an external (to the processor) hardware device
  - part of the computer, external peripherals
- **A software interrupt**
  - requested by the processor itself upon executing particular instructions or when certain conditions are met.
  - A software interrupt may be intentionally caused by executing a special instruction
  - Software interrupts may also be unexpectedly triggered by program execution errors.

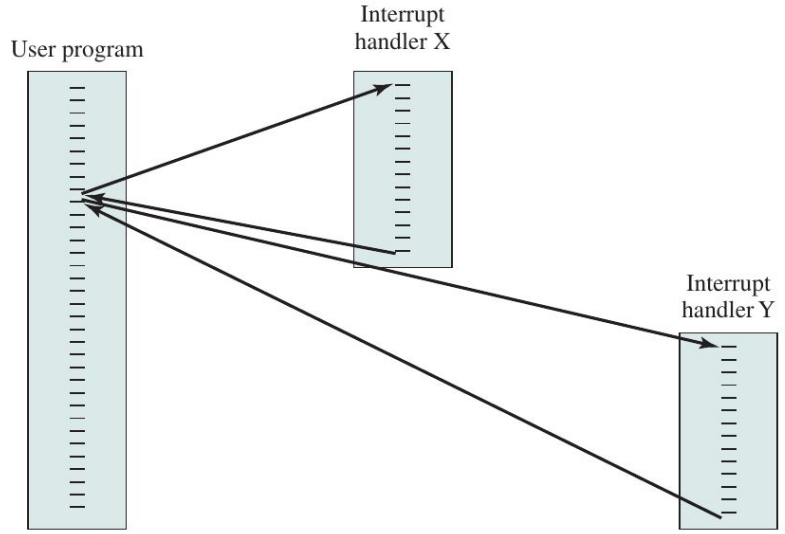
Processor - interrupts

- **Hardware interrupts**
  - issued by an external (to the processor) hardware device
  - part of the computer (e.g., disk controller) or external peripherals (e.g., pressing a keyboard key or moving the mouse)
- **A software interrupt**
  - requested by the processor itself upon executing particular instructions or when certain conditions are met. Every software interrupt signal is associated with a particular **interrupt handler**.
  - A software interrupt may be **intentionally caused by executing a special instruction** which, by design, invokes an interrupt when executed. Such instructions function similarly to subroutine calls and are used for a variety of purposes, such as **requesting operating system services** and interacting with device drivers (e.g., to read or write storage media).
  - Software interrupts may also be unexpectedly triggered by program execution errors. These interrupts typically are called **traps or exceptions**. For example, a divide-by-zero exception

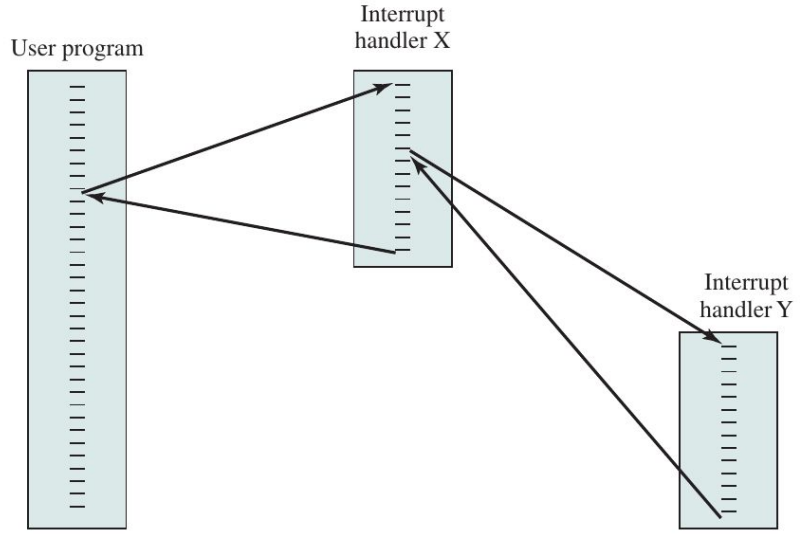
Processor - interrupts

- The operating system **preserves the state of the CPU** by storing registers and the program counter
- Processors typically have an **internal interrupt mask** register which allows selective enabling and disabling of hardware interrupts.
- Two approaches can be taken to dealing with **multiple interrupts**.
  - Incoming interrupts are **disabled while another interrupt is being processed** to prevent a lost interrupt
  - Define **priorities for interrupts** and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted

Processor - interrupts



(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 1.12 **Transfer of Control with Multiple Interrupts**

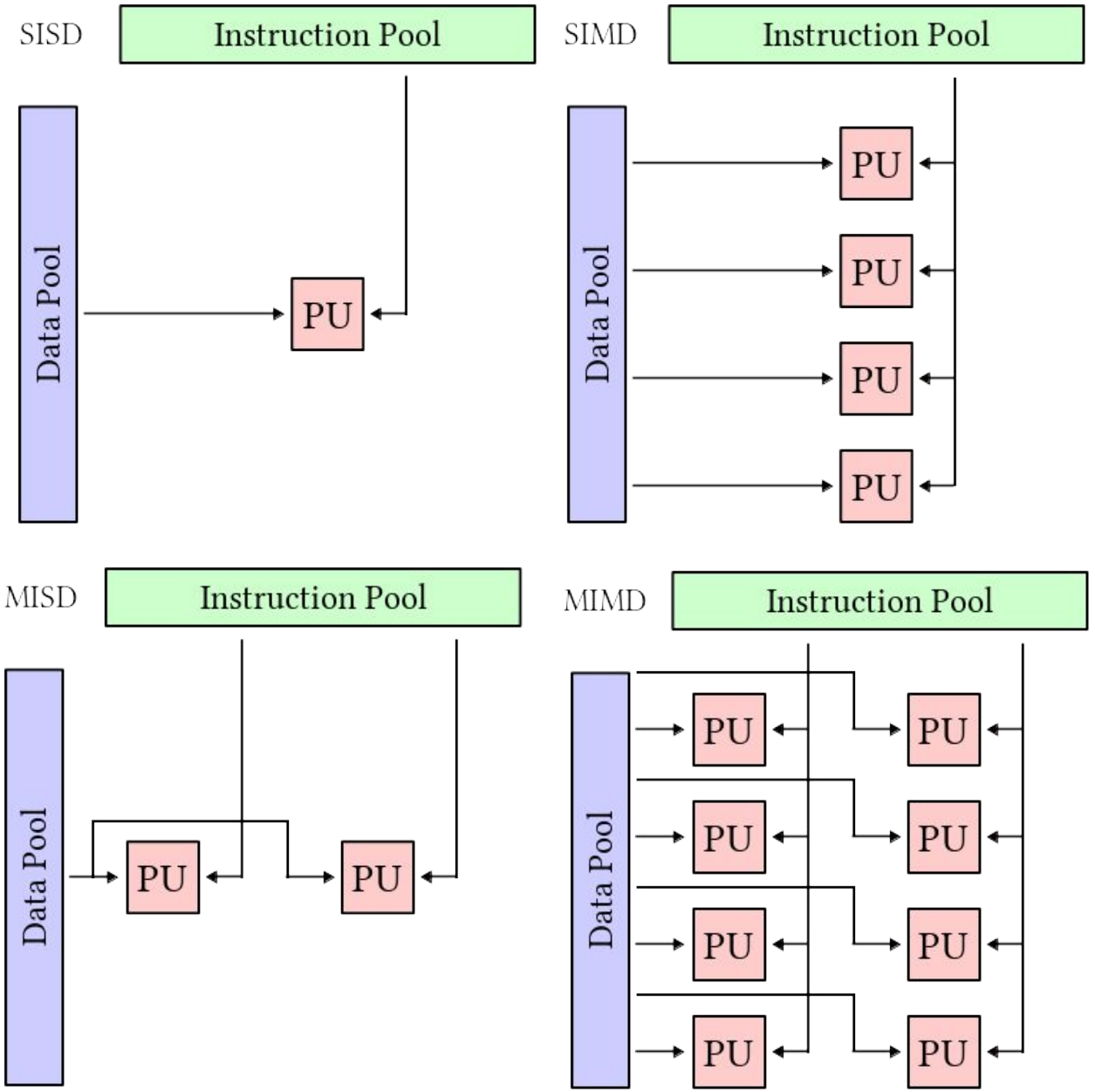
- Two approaches can be taken to dealing with **multiple interrupts**.
  - Incoming interrupts are **disabled while another interrupt is being processed** to prevent a lost interrupt
  - Define **priorities for interrupts** and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted

Processor - interrupts

INT_NUM	Short Description PM [*clarification needed*]
0x00	Division by zero
0x01	Single-step interrupt (see trap flag)
0x02	NMI
0x03	Breakpoint (callable by the special 1-byte instruction 0xCC, used by debuggers)
0x04	Overflow
0x05	Bounds
0x06	Invalid Opcode
0x07	Coprocessor not available
0x08	Double fault
0x09	Coprocessor Segment Overrun *(386 or earlier only)*
0x0A	Invalid Task State Segment
0x0B	Segment not present
0x0C	Stack Fault
0x0D	General protection fault
0x0E	Page fault
0x0F	*reserved*
0x10	Math Fault
0x11	Alignment Check
0x12	Machine Check
0x13	SIMD Floating-Point Exception
0x14	Virtualization Exception
0x15	Control Protection Exception

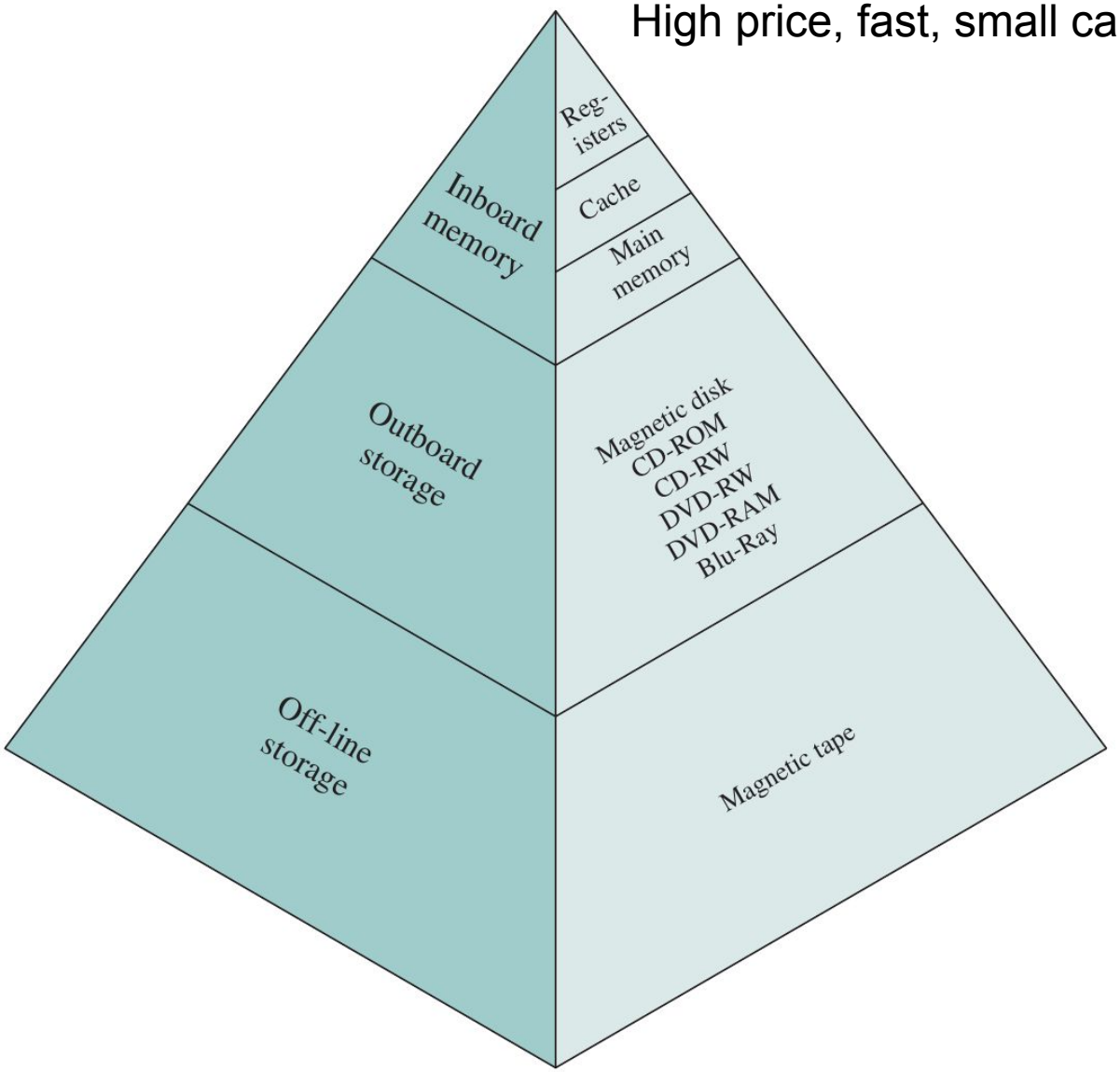
- The Interrupt Descriptor Table (IDT) is a data structure used by the x86 architecture to implement an interrupt vector table.

Flynn taxonomy



Memory

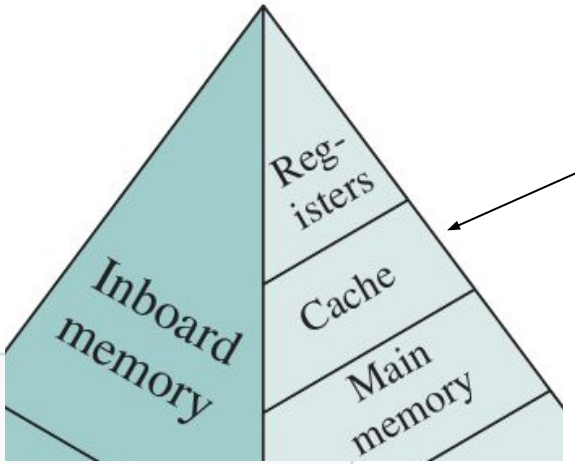
High price, fast, small capacity



Low price, slow, large capacity

Figure 1.14 The Memory Hierarchy

Cache - size



AMD Ryzen 9 3900x

Cache Organization

L1$	768 KiB	L1I$	384 KiB	12x32 KiB	8-way set associative
		L1D$	384 KiB	12x32 KiB	8-way set associative write-back
L2$	6 MiB			12x512 KiB	8-way set associative write-back
L3$	64 MiB			4x16 MiB	

Intel i9 9900k

[Edit/Modify Cache Info]

Cache Organization

L1$	512 KiB	L1I$	256 KiB	8x32 KiB	8-way set associative
		L1D$	256 KiB	8x32 KiB	8-way set associative write-back
L2$	2 MiB			8x256 KiB	4-way set associative write-back
L3$	16 MiB			8x2 MiB	16-way set associative write-back

Cache – mapping function

- cache size
- block size: the unit of data exchanged between cache and main memory.
- **mapping function** determines which cache location the block will occupy.
  - **replacement algorithm** chooses, within the constraints of the mapping function, which block to replace when a new block is to be loaded into the cache

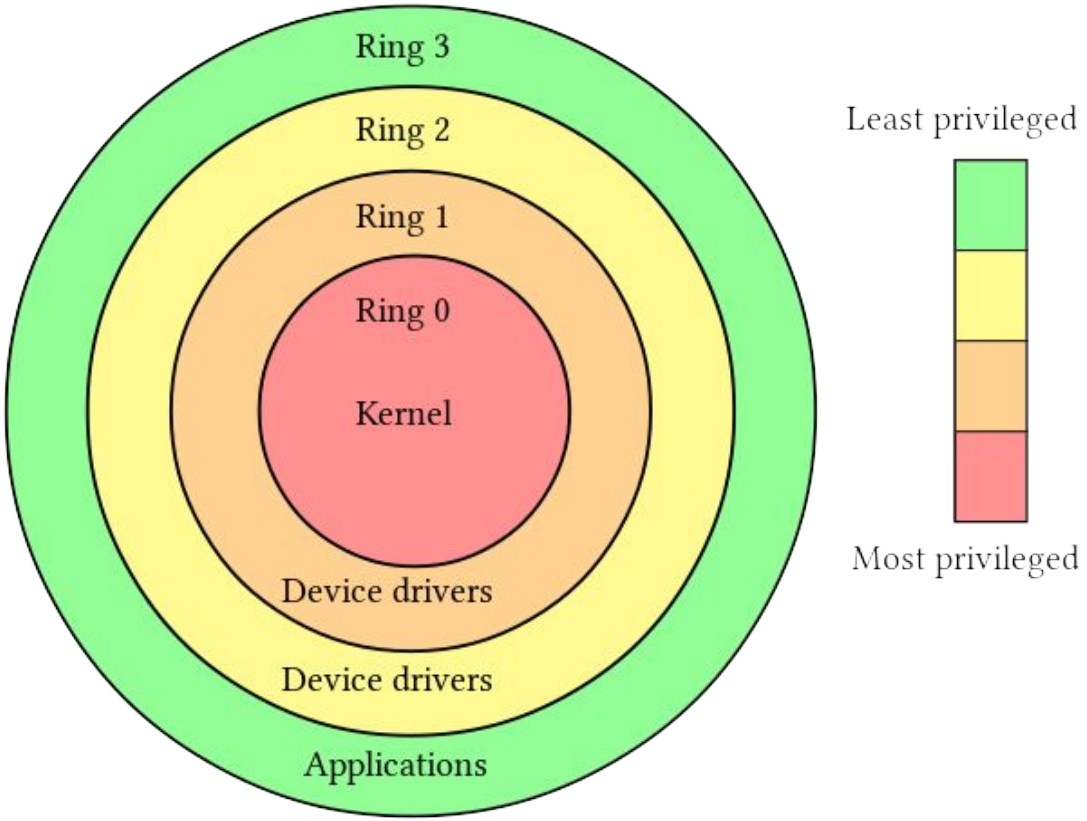
Cache - locality of reference

- Well described in: https://en.wikipedia.org/wiki/Locality_of_reference
- Tendency of a processor to access the **same set of memory locations repetitively** over a short period of time
- **Temporal locality** refers to the reuse of specific data, and/or resources, within a relatively small time duration.
- **Spatial locality** refers to the use of data elements within relatively close storage locations

CPU modes (x86)

- **Real mode**
  - Real mode is characterized by a 20-bit segmented memory address space (giving exactly 1 MiB of addressable memory) and **unlimited direct software access to all addressable memory, I/O addresses and peripheral hardware**. Real mode provides no support for memory protection, multitasking, or code privilege levels.
  - DOS
- **Protected mode**
  - Typical
  - It allows system software to use features such as virtual memory, paging and safe multi-tasking
  - **Privilege rings** for the x86 available in protected mode

CPU modes (x86)



- Privilege rings for the x86 available in protected mode

Booting process

- Well described inhttps://en.wikipedia.org/wiki/Booting#Boot_sequence
- In real mode, execute the instruction located at **reset vector** CS:IP = F000:FFF0
  - The reset vector is the **default location a central processing unit will go to find the first instruction** it will execute after a reset.
  - Usually pointing to the firmware (UEFI or BIOS)
- BIOS runs a **power-on self-test (POST)** to check and initialize required devices such as DRAM and the PCI bus (including running embedded ROMs
- The firmware (UEFI or BIOS) goes through pre-configured list of non-volatile storage devices ("boot device sequence") until it finds one that is bootable - **MBR boot signature**

Source: (http://www.independent-software.com/operating-system-development-first-and-second-stage-bootloaders.html)

Booting process – bootloaders

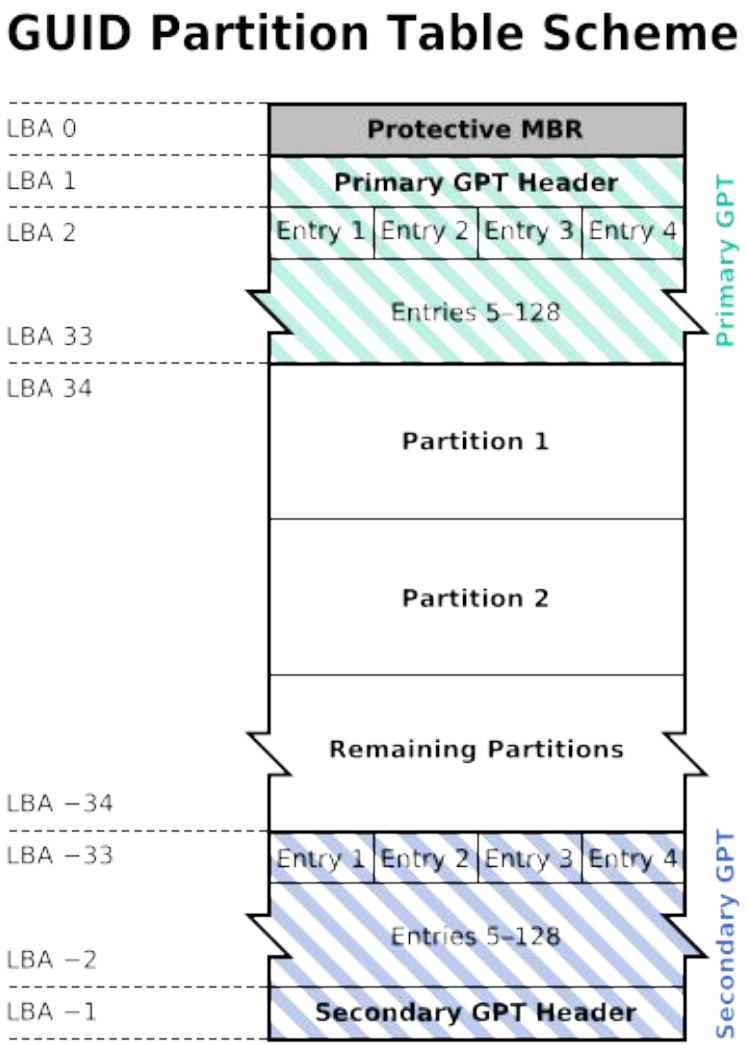
- **First-stage boot loader** (~~usually part of BIOS?~~)
  - Must fit into the first 446 bytes of the Master Boot Record
  - For example **coreboot**
  - Must do the following:

1) Setup the memory segments and stack used by the bootloader code
2) Reset the disk system
3) Display a string saying “Loading OS…”
4) Find the second-stage boot loader in the FAT directory
5) Read the second-stage boot loader image into memory at 1000:0000
6) **Transfer control to the second-stage bootloader**

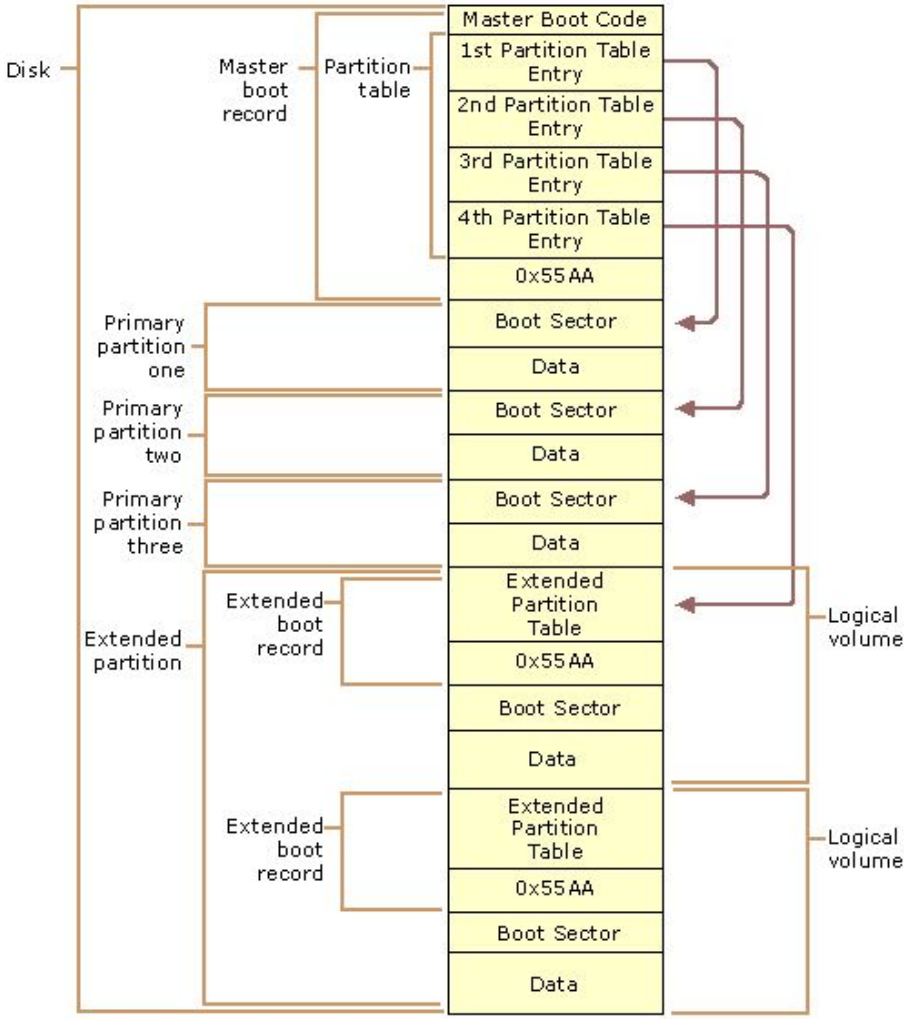
- **Second-stage boot loader**
- Not operating systems, but are able to load an operating system properly and transfer execution to it
- For example GNU **GRUB** - Grand **Unified** Bootloader (includes both 1. and 2.)
- In the second-stage bootloader, we must do the following:

1) Copy the boot sector data bytes to a local memory area, as they will be overwritten
2) Find the kernel image in the FAT directory
3) Read the kernel image into memory at 2000:0000
4) Reset the disk system
5) Enable the A20 line
6) Setup the interrupt descriptor table at 0000:0000
7) Setup the global descriptor table at 0000:0800
8) Load the descriptor tables into the CPU
9) Switch to protected mode
10) Clear the prefetch queue
11) Setup protected mode memory segments and stack for use by the kernel code
12) Transfer control to the kernel code using a long jump

Master Boot Record

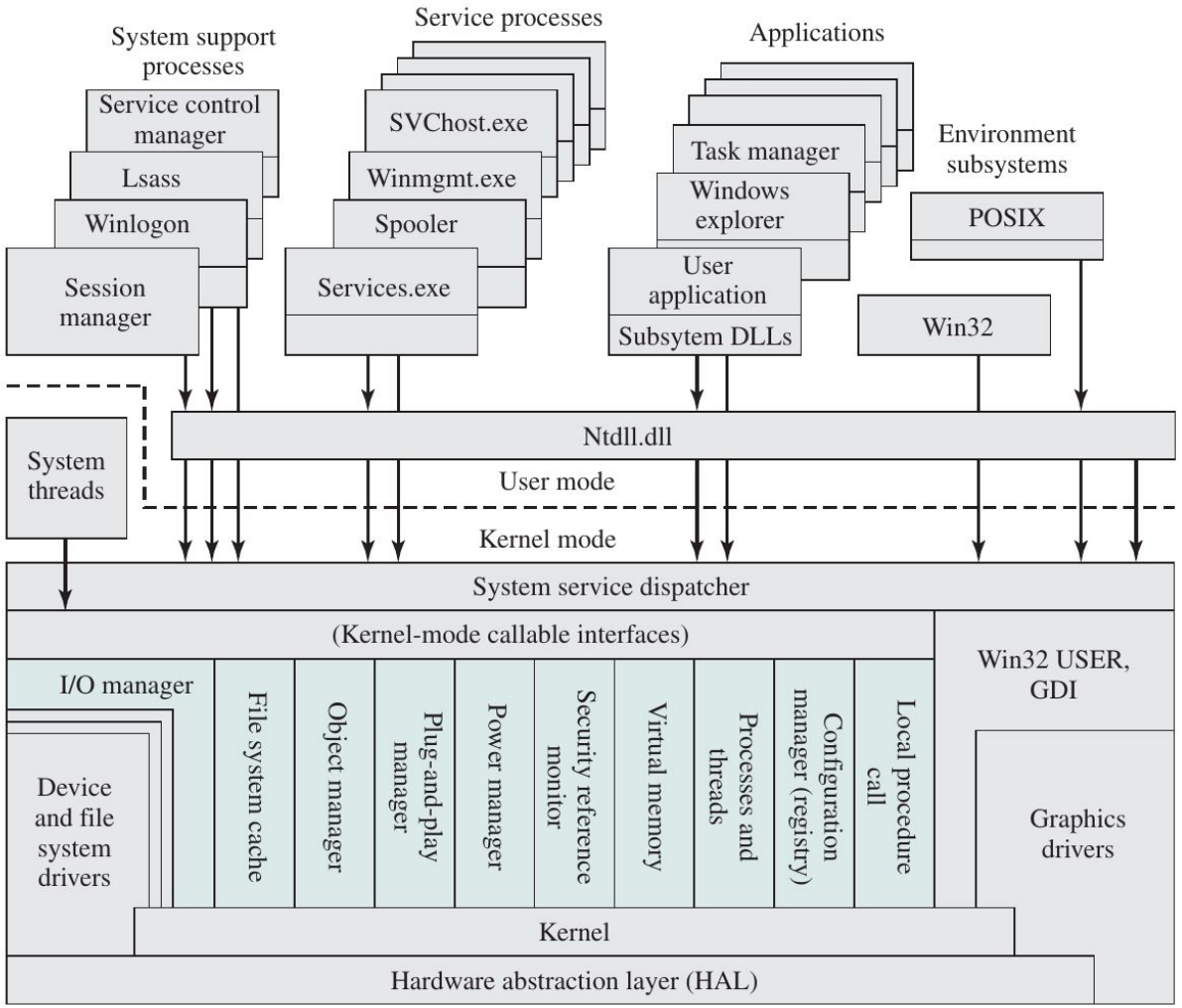


GPT



MBR

OS Kernel



Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

Figure 2.15 **Windows and Windows Vista Architecture [RUSS11]**

- Kernel types
  - Monolithic
  - Microkernel
  - Hybrid (or modular) kernel
- Windows and macOS - Hybrid (or modular) kernel
- Hybrid kernels are similar to micro kernels, except they include some additional code in kernel-space to increase performance
- Linux – originally monolithic kernel

OS Kernel - Unix

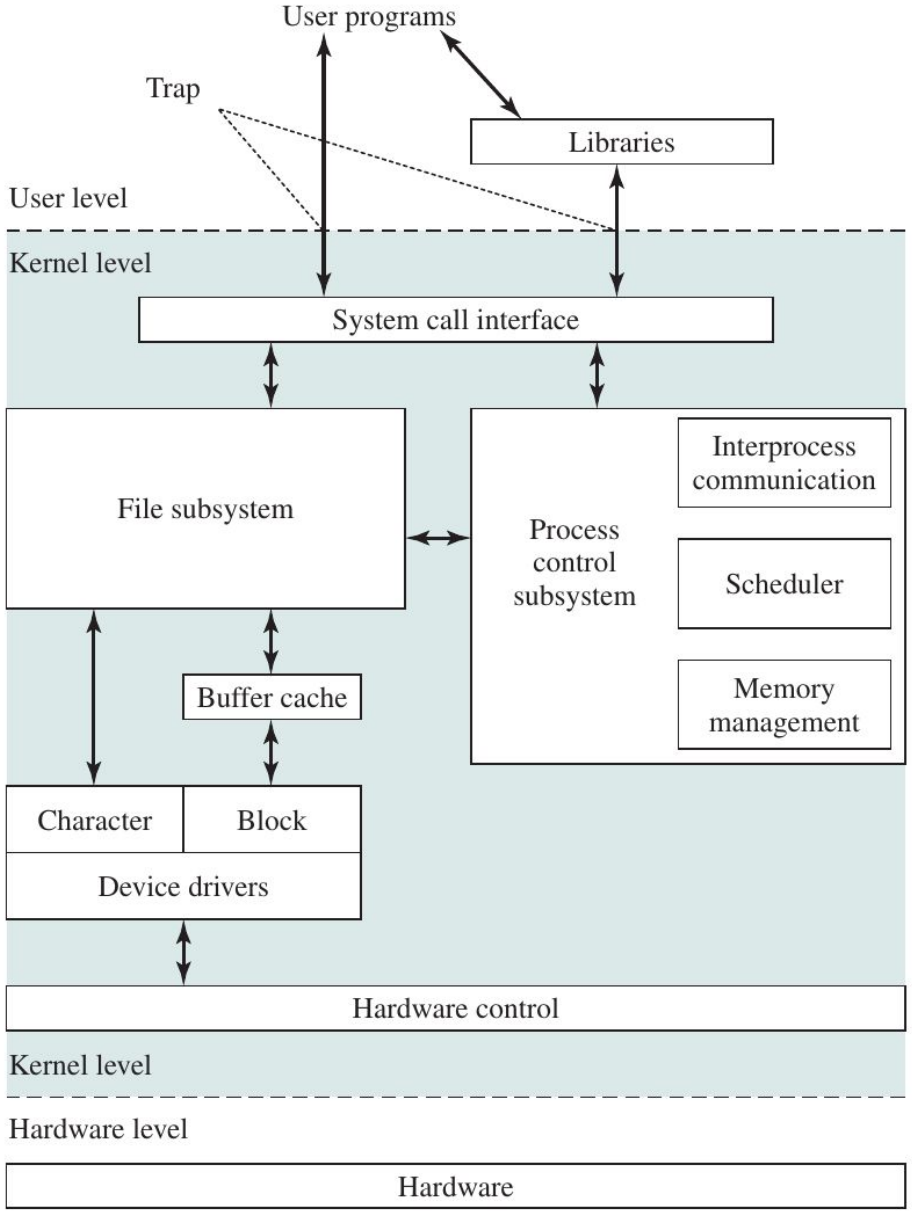


Figure 2.17 Traditional UNIX Kernel

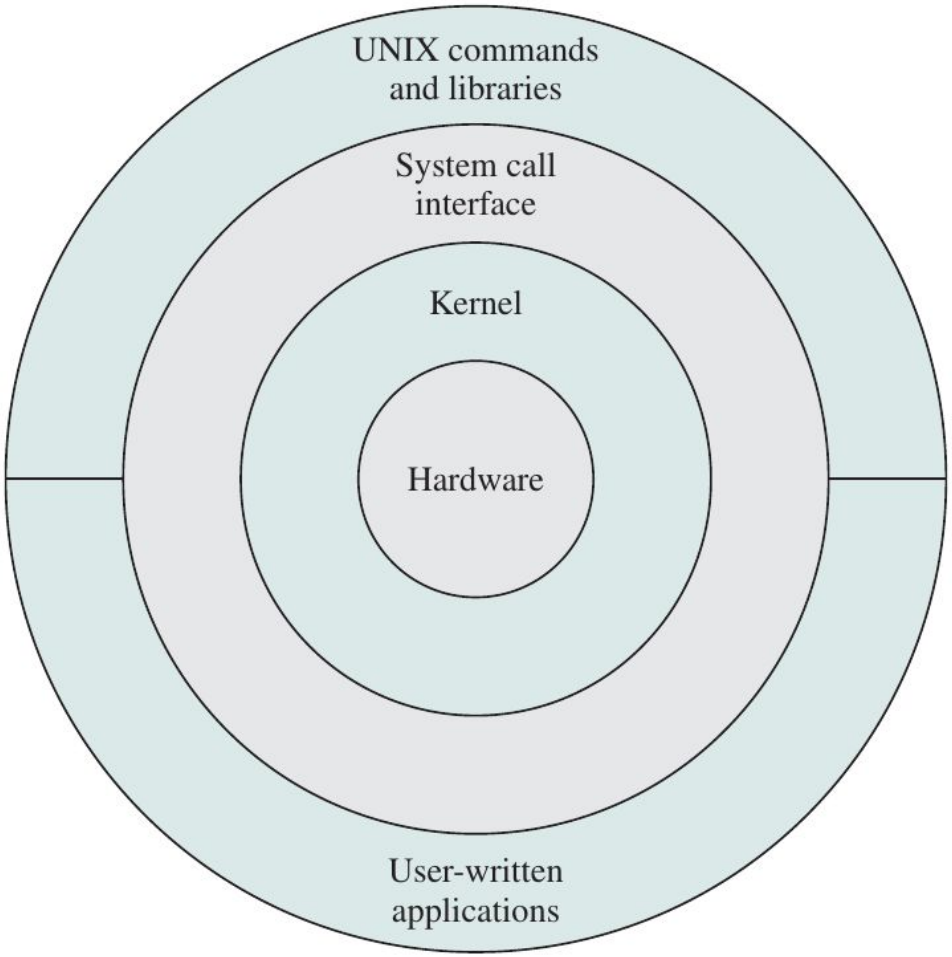


Figure 2.16 General UNIX Architecture