Today's Plan

Upcoming:

- Assignment 1

Last time:

- Introduction to the course

Today's topics:

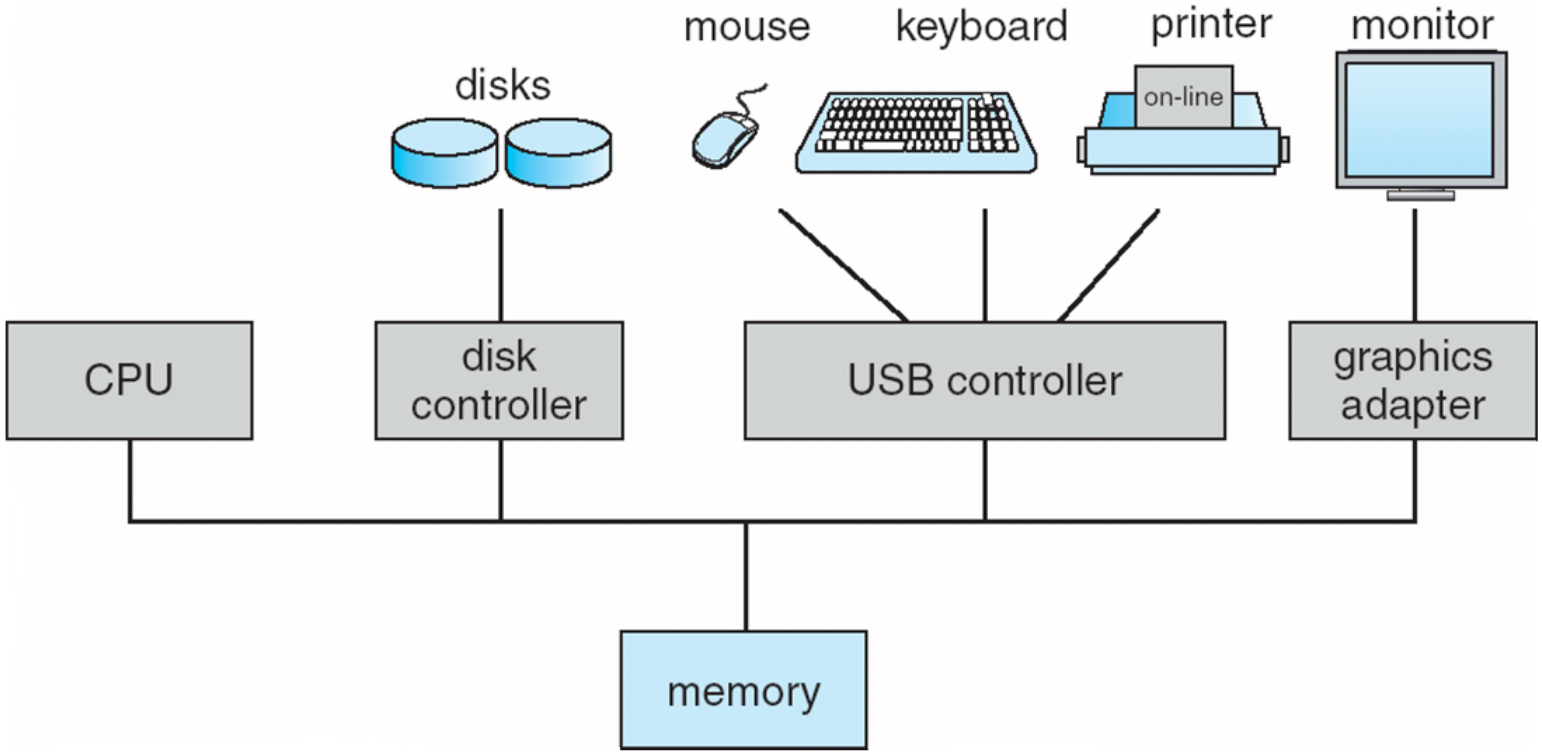
- Computer System Organization
- Interrupts
- I/O Structure
- Storage Structure
- Types of Computer Systems
- Operating System Structure

Computer Startup

- A *bootstrap program* is loaded at power-up or reboot
- Typically stored in ROM or EEPROM, generally known as *firmware*
- Initializes all aspects of the system
- Loads operating system kernel and starts execution

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through a common bus providing access to *shared memory*



Interrupts

- We want the I/O devices and the CPU to be able to execute concurrently
  - The CPU shouldn't have to wait for the MUCH slower I/O device
- The I/O device signals an *interrupt* when it is ready
- When an interrupt occurs, the operating system must:
  - Transfer control to the appropriate **interrupt handler**
  - Since there is more than one device, we use an **interrupt vector**

Interrupts: Controllers vs. Handlers

- A device controller (hardware / firmware) is responsible for moving data between the media and its own local registers
- An interrupt handler (software) is responsible for moving data between the controller registers and memory (so it can be accessed by the user program)
  - Since there are multiple types of interrupts, there is a handler for each type

Processing an Interrupt

1. User program has control
2. I/O interrupt occurs
   - E.g. disk controller signals data read operation completed
3. User program no longer processed by the CPU
   - Current position saved (Program Counter), and interrupts are disabled
4. Control transferred to interrupt handler
   - CPU registers representing current state are saved

Processing an Interrupt

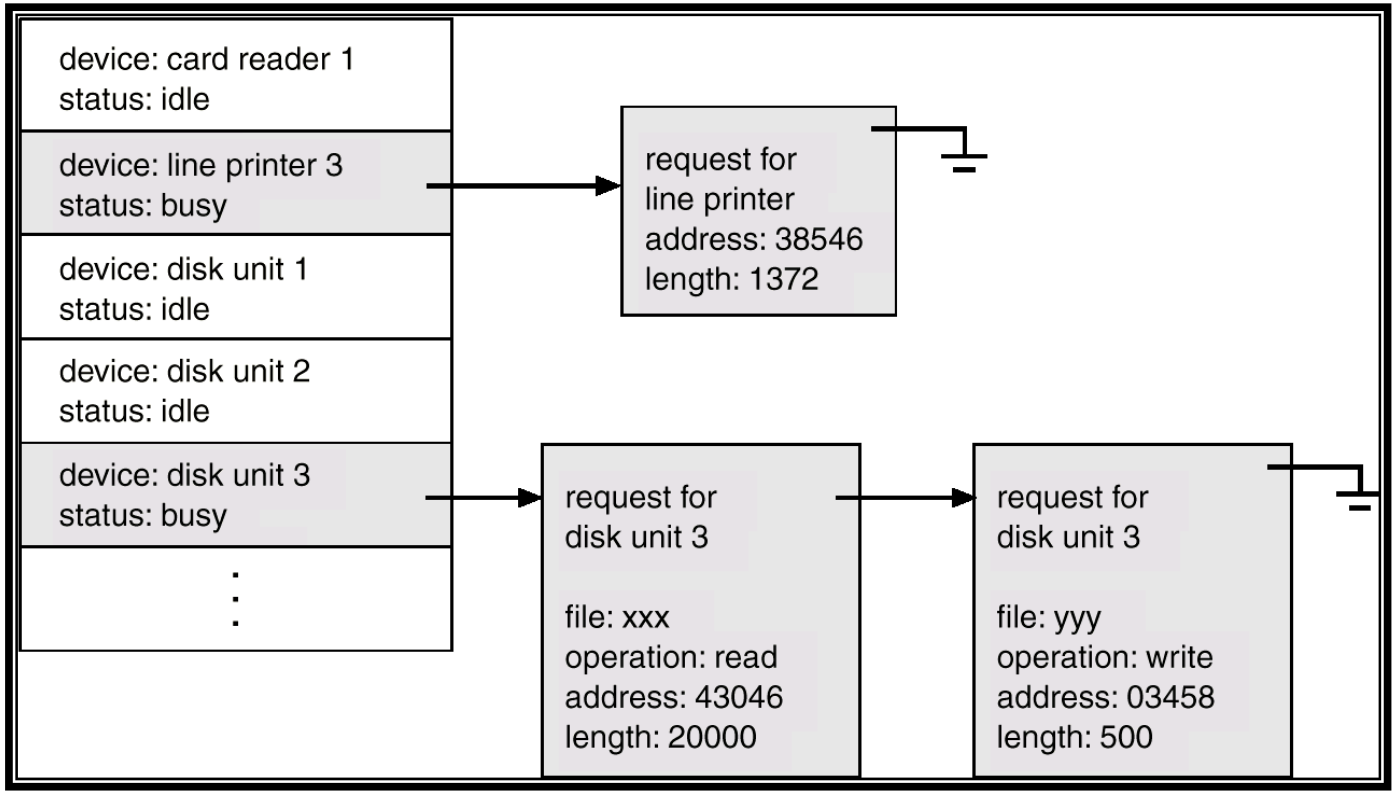
5. Data transferred from controller to memory or from memory to controller depending on whether input or output operation
6. CPU registers restored
7. Control returned to user program
   - Interrupts re-enabled
   - Saved Program Counter restored

I/O Structure

- The CPU and device controllers operate in parallel
- Two types of I/O operations:
  - *synchronous:* user program waits until I/O operation completes
  - *asynchronous:* user program allowed to continue while I/O operation is in progress
- Asynchronicity is essential for *multiprogramming*
- Asynchronous I/O for **program A** allows the CPU to transfer control to another **program B** until I/O for A is complete.

I/O Structure – Device Status Table

- A device status table keeps track of which devices are busy, what they are doing for which program, and which programs are waiting for access to devices.



I/O Structure – Direct Memory Access (DMA)

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per **block**, rather than one per byte.

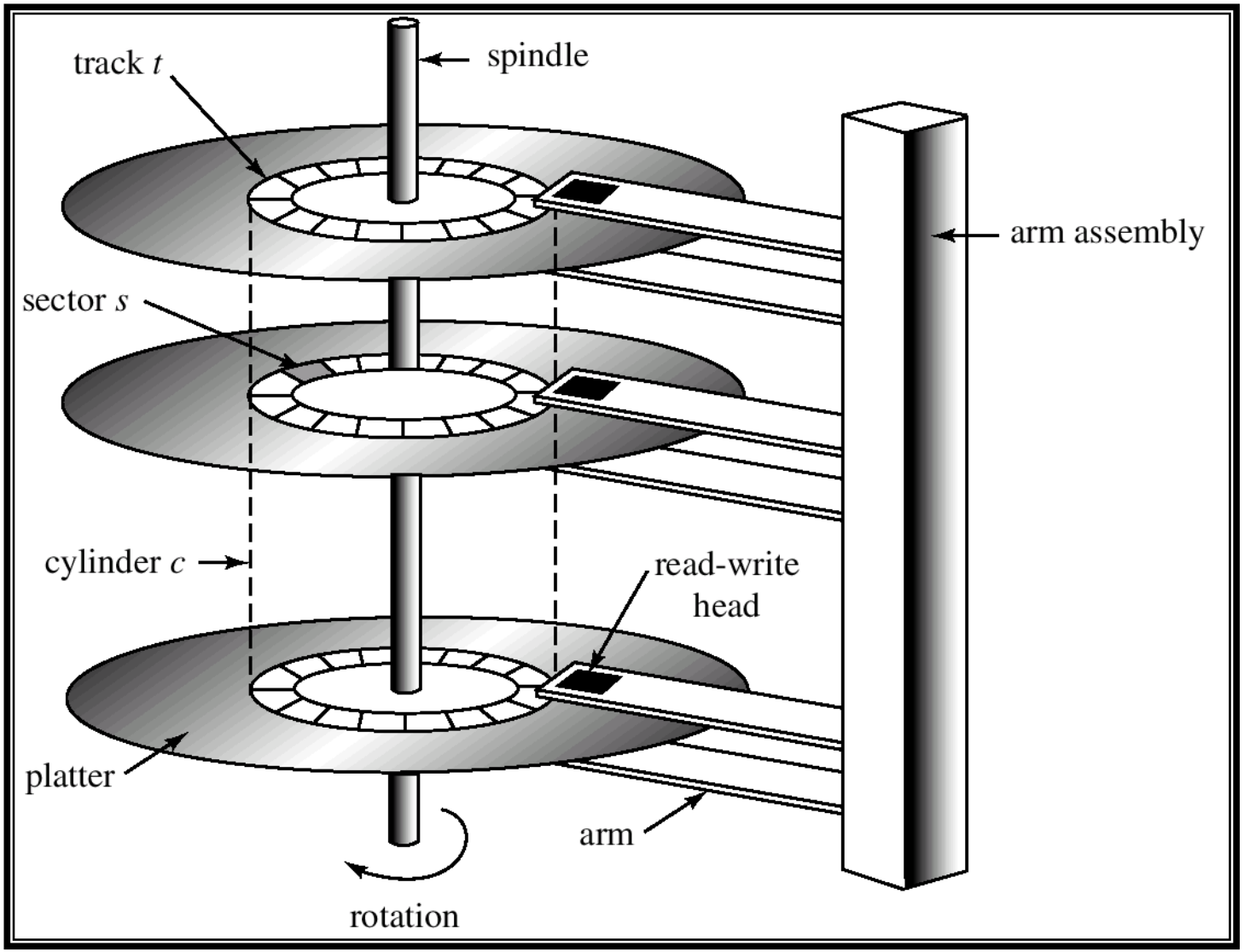
Storage Structure

- Typical (von Neumann) instruction-execution cycle
  - Instruction loaded from *main memory* into CPU
    - Main memory – only large storage media that the CPU can access directly
  - Instruction decoded
    - May cause other operands to be retrieved from memory
  - Instruction executed, repeat
- *Secondary storage* – extension of main memory that provides large nonvolatile storage capacity

Storage Structure – Magnetic Disks

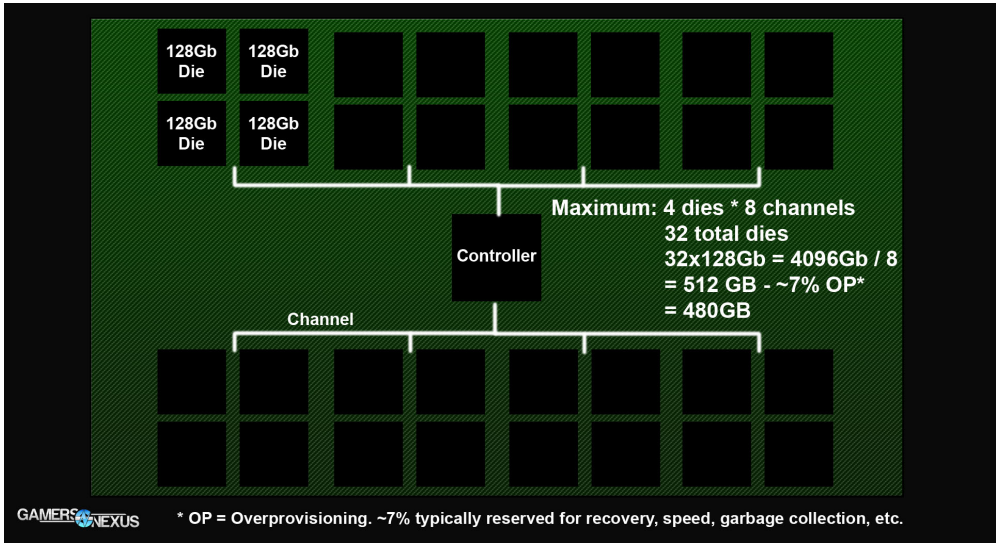
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
  - The *disk controller* determines the logical interaction between the device and the computer.

Storage Structure – Magnetic Disk Architecture



Storage Structure – Solid-State Drives

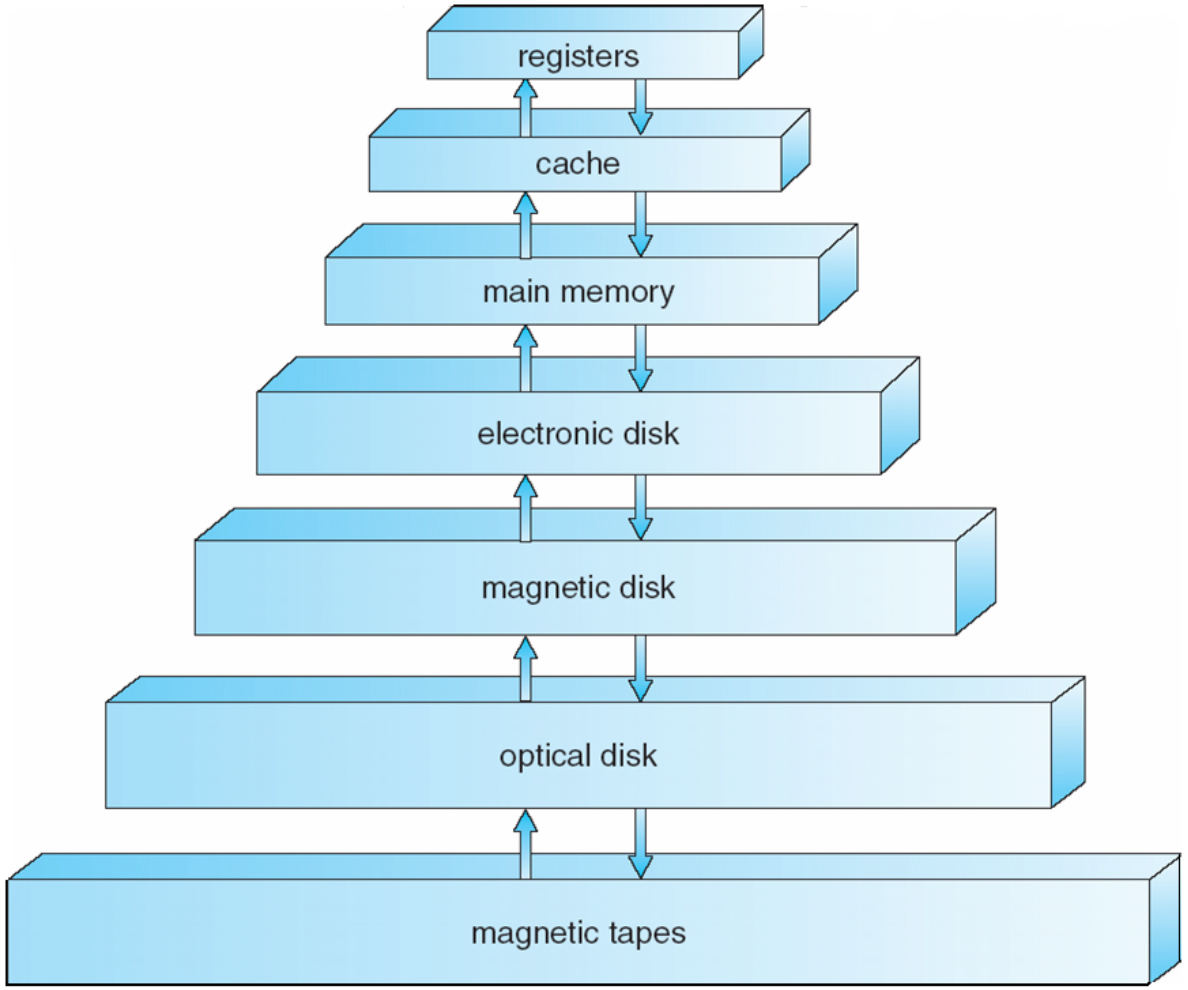
- Solid-State Drives use **NAND Flash** technology
- Bits stored as charges in cells made of transistors
- SLC, MLC, and TLC drives differ in how many bits are stored per cell
- Organization:
  8 channels, 4 dies per channel



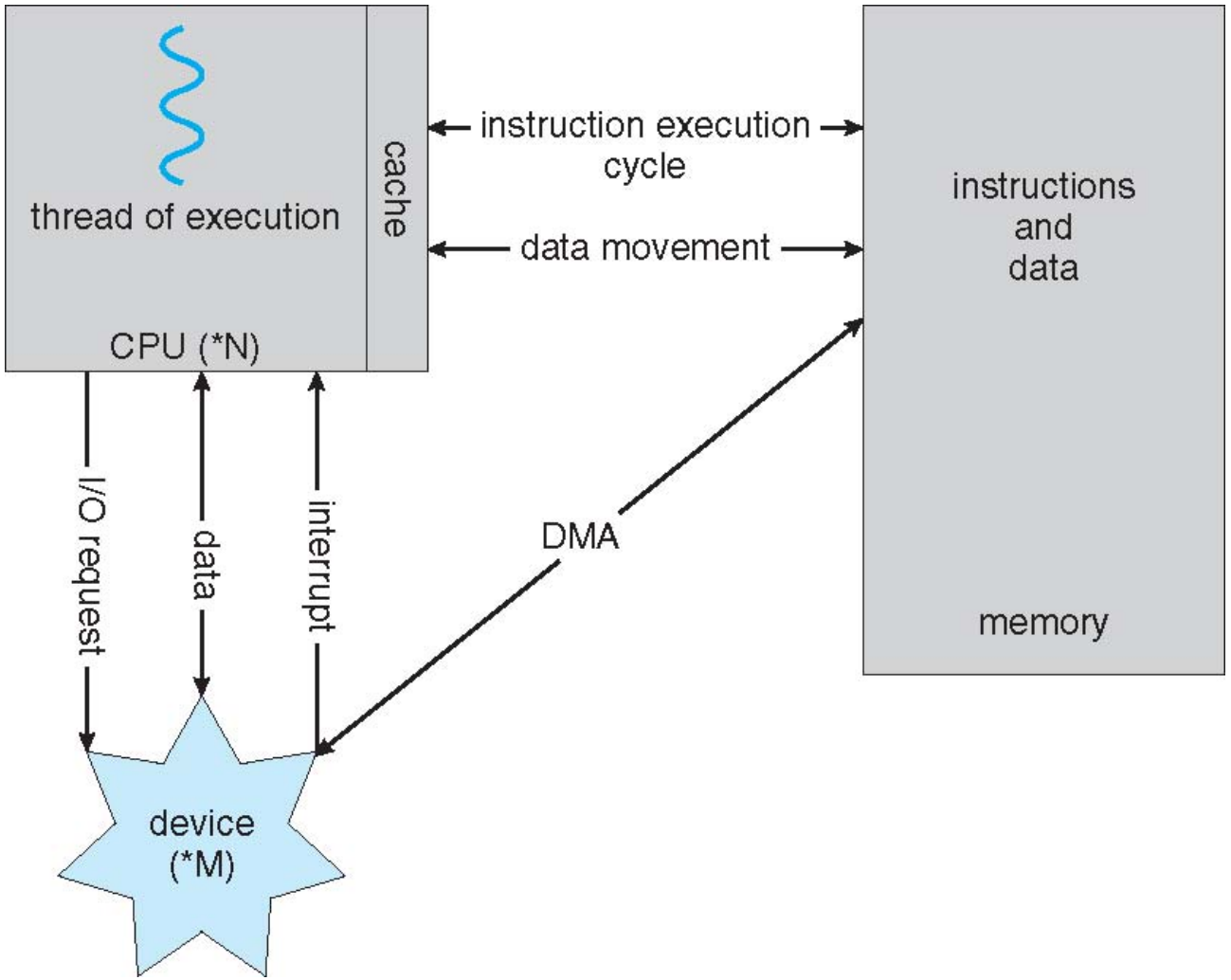
Storage Structure - Caching

- Caching is the use of a smaller, but faster, memory system to speed up a bigger, but slower, memory system
  - The cache holds the data most recently accessed by the CPU
  - Effective because most programs access the same data or instructions repeatedly
- *Cache management* is an important design problem. A well chosen cache size and replacement policy can result in 80+% of all accesses being in the cache.
- Problem: Consistency
  - Different processes must see the same value for an item.

Storage Hierarchy

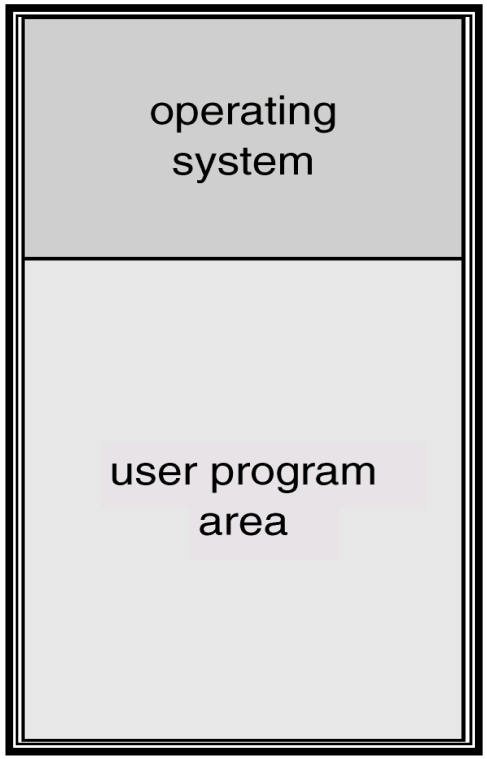


How a Modern Computer Works



Mainframe Systems

- Mainframes are large, powerful computers designed to handle large numbers of users and jobs
- The first operating systems appeared in mainframes
  - One process/job executed at a time – no multi-tasking!
- The OS (called a monitor) transfers control to the job, when the job is done control is returned to the OS

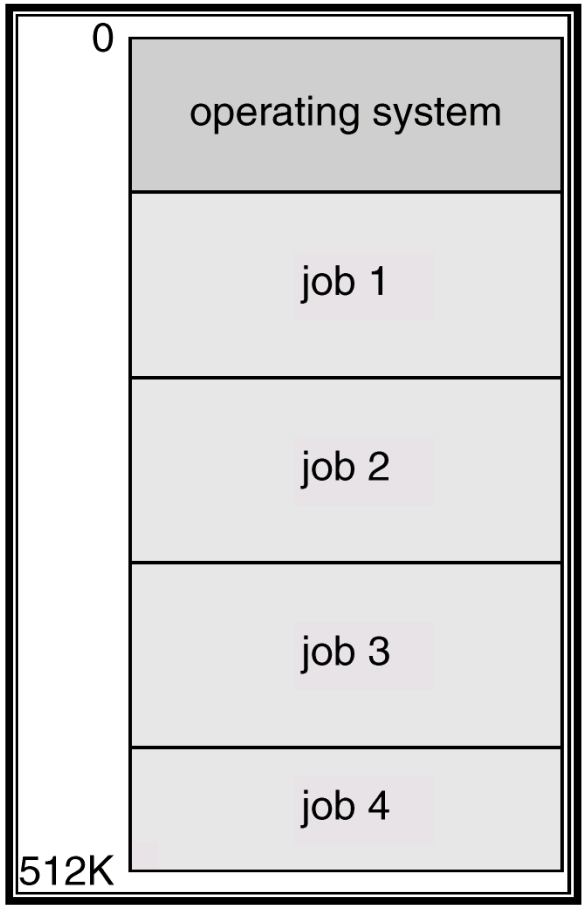


Mainframe Systems

The next step up is the Multiprogrammed Batch System

- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them

- Each job has its own memory space



Mainframe Systems

Some of the OS features needed for Multiprogramming:

- I/O routine supplied by the system
- Memory management
  - The system must allocate the memory to several jobs
- CPU scheduling
  - The system must choose among several jobs ready to run
- Allocation of devices (storage devices, etc.)

Desktop Systems

- *Personal computers* – computer system dedicated to a single user
- I/O devices – keyboards, mice, monitors, printers
- User convenience and responsiveness
- Can adopt technology developed for larger operating systems
  - Usually individuals have sole use of computer and do not need advanced CPU utilization or protection features
- May run several different types of operating systems (Windows, Mac OS/X, UNIX, Linux)

Multiprocessor Systems

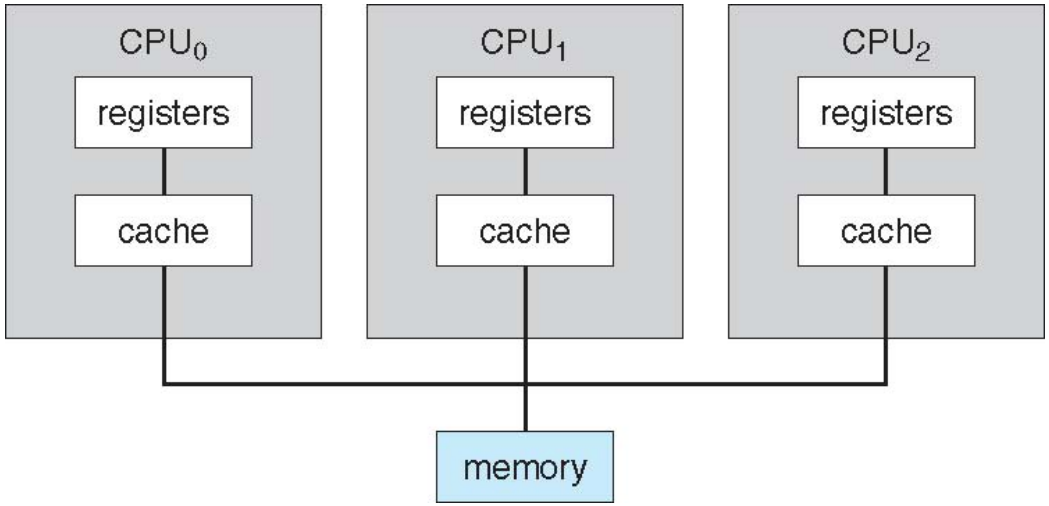
- Multiprocessor systems are systems with more than one CPU in close communication.
  - E.g. multi-core CPUs
- This is a *tightly coupled system*
  - Processors share memory and a clock
  - Communication between them occurs through some form of shared memory
- Advantages of parallel system:
  - Increased throughput
  - Economical
  - Increased reliability

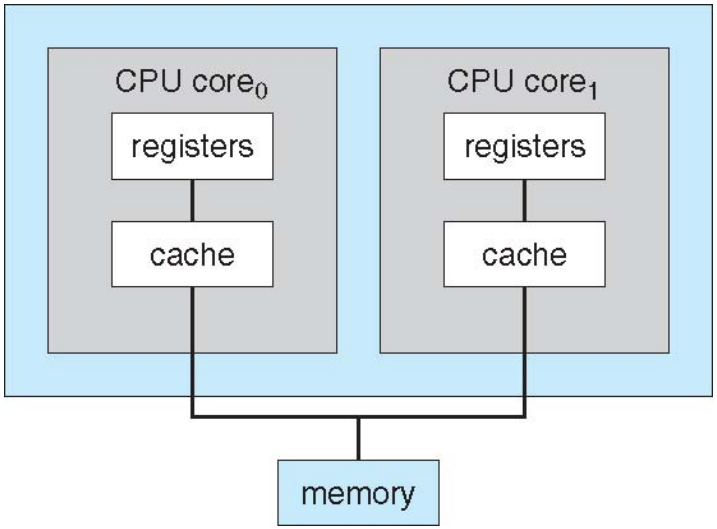
Types of Multiprocessor Systems

- *Symmetric multiprocessing (SMP)*
  - Each processor runs an identical copy of the operating system
  - Many processes can run at once without performance deterioration
  - Most modern OSes support SMP
- *Asymmetric multiprocessing*
  - Each processor is assigned a specific task
  - Master processor schedules and allocates work to slave processors

Symmetric & Multi-core Processors

Symmetric Multiprocessor:





Dual-core processor

Distributed Systems

- Another idea is to distribute the computation among several physical processors
- This is a *loosely coupled system* –
  - Each processor has its own local memory
  - Processors communicate with one another through communications lines, such as networks (LAN's and WAN's)

Real-Time Systems

- A *real-time system* is one where there are well-defined fixed-time constraints
  - I.e. things need to happen in a reasonable amount of time and in the correct order
- Some examples:
  - Controlling robots for manufacturing
  - Controlling medical devices

Handheld Systems

Some examples:

- Personal Digital Assistants (PDAs)
- Smartphones

Some of the issues when designing handheld systems:

- Limited memory
- Slow processors
- Small display screens
- Touchscreen input