Operating Systems

CSE 29: Systems Programming and Software Tools

Aaron Schulman



C

Introduction to Operating Systems

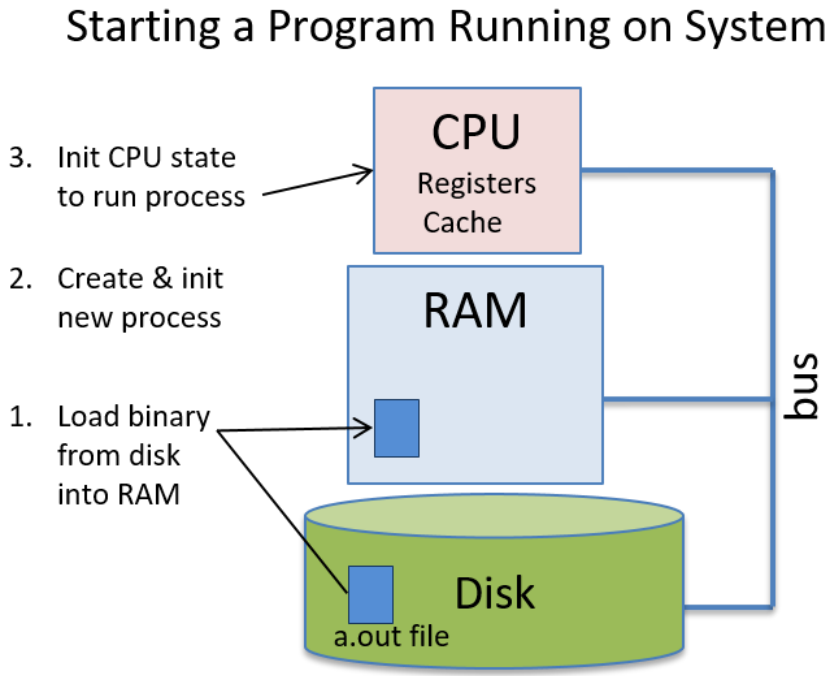
- An Operating System is the *program* on a computer that runs user *programs*
  - Starts and ends program execution
  - Switches between running programs
  - Manages access to hardware (e.g., RAM and Disk) so they are shared across programs

- Topics covered in CSE 120 Operating Systems
  - How does the Operating System start running?
  - How does an Operating System run a program on a computer?
  - How does an Operating System share access to hardware in the computer?
  - How does a program ask the Operating System to do things for it?
  - How does an Operating System allow programs to run independently of other programs?

Booting an Operating System

- The Operating System Boots or “Picks itself up from its Bootstraps” when the computer powers on:
  - The CPU jumps to the first instruction when it is powered on:
  - This code is called the “**Bootloader**” it contains enough code to load the Operating System
  - Examples of Bootloaders are:
    - **BIOS** (Basic Input/Output System)
    - **UEFI** (Unified Extensible Firmware Interface)
  - The Bootloader reads the Operating System’s first chunk of instructions from secondary storage into RAM
  - Then it calls the main() function of the Operating System

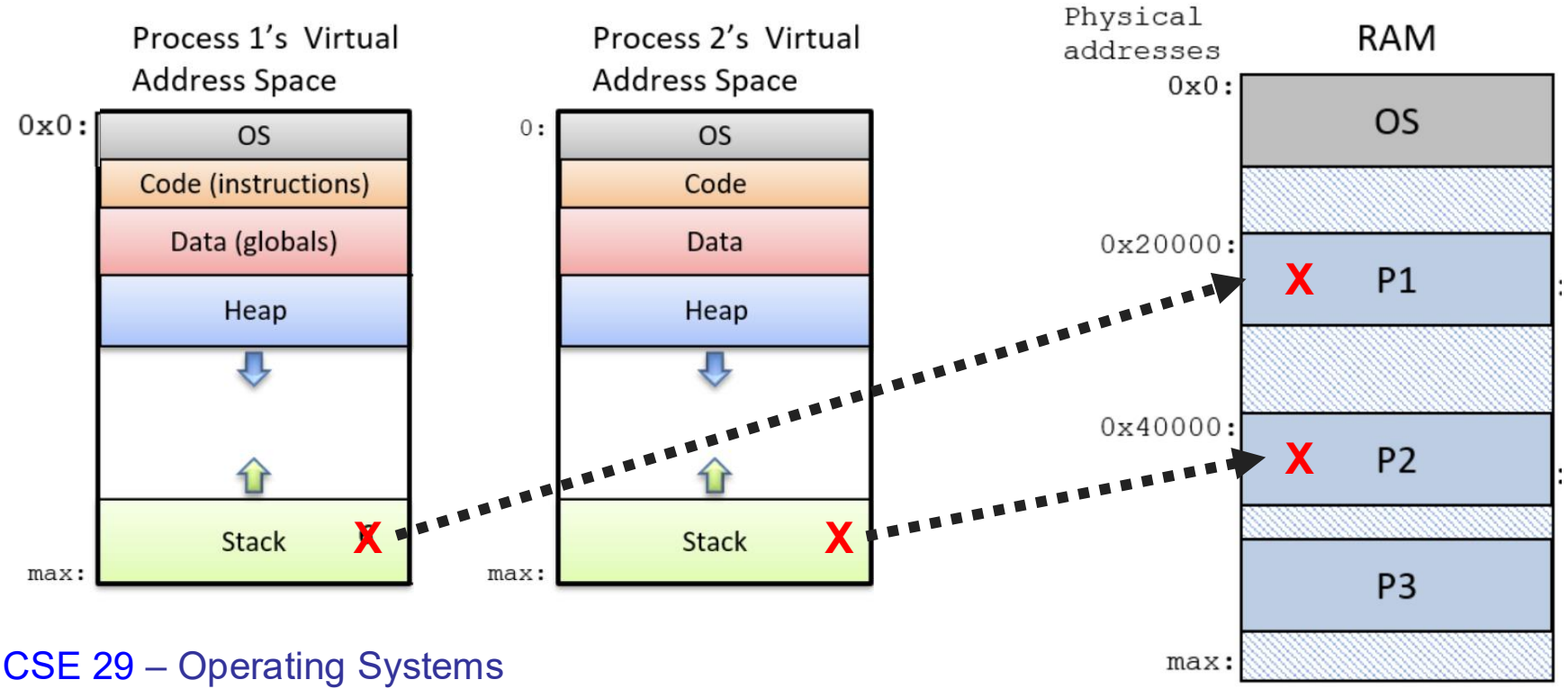
How does an OS run a program?



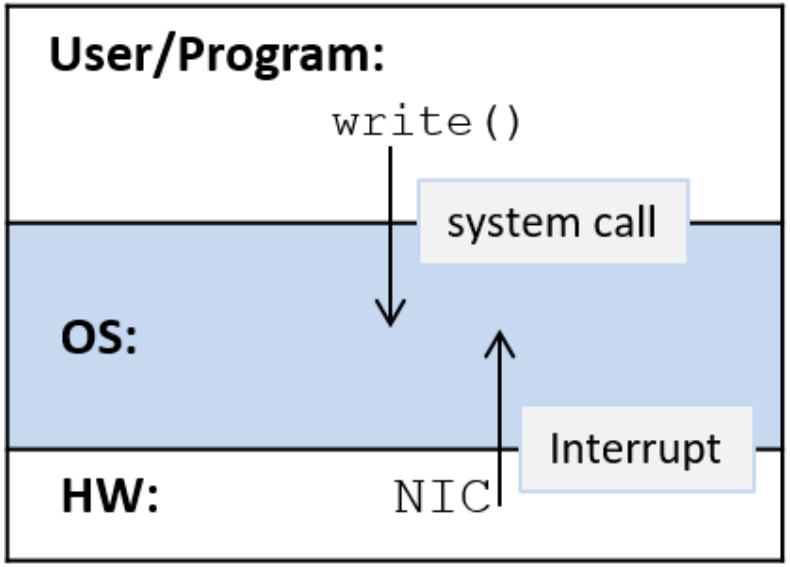
Virtual Memory: Aaron is a liar

- C programs don't actually provide full control over the entire address space of the RAM of the computer
- OS provides an abstraction where each process acts like it can control the entire memory space of the computer
  - OS does this by maintaining a lookup table for each process called a "Page Table"
  - Page Tables map process Virtual Addresses to Physical Addresses
- Page Tables even allow a program to use more RAM than the system has
  - The OS will automatically move "Pages" (~4KB) back and forth from disk

Where is var "X" running same code twice



How programs and hardware communicate



System calls:

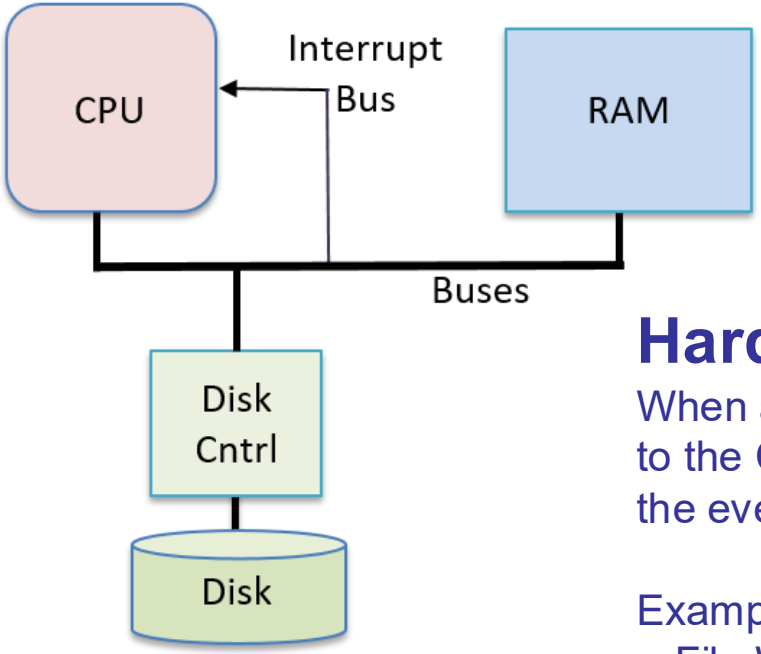
Function calls into the Operating System's code to ask it to interact with the hardware of the computer (e.g., Disk)

fork(), printf(), write(), execv()….

Interrupts:

Function calls into the Operating System's code from hardware to inform the OS that a task has completed

Interrupts notify the OS of hardware events



Hardware Interrupts

When a hardware device fires an Interrupt a wire to the CPU calls a function in the OS to handle the event that occurred

Examples:
- File Write to disk has Completed
- Received network packet

Types of memory in a computer

- Primary memory used by running programs (stack, heap, and code)
- Time taken to access data is not affected by location of data

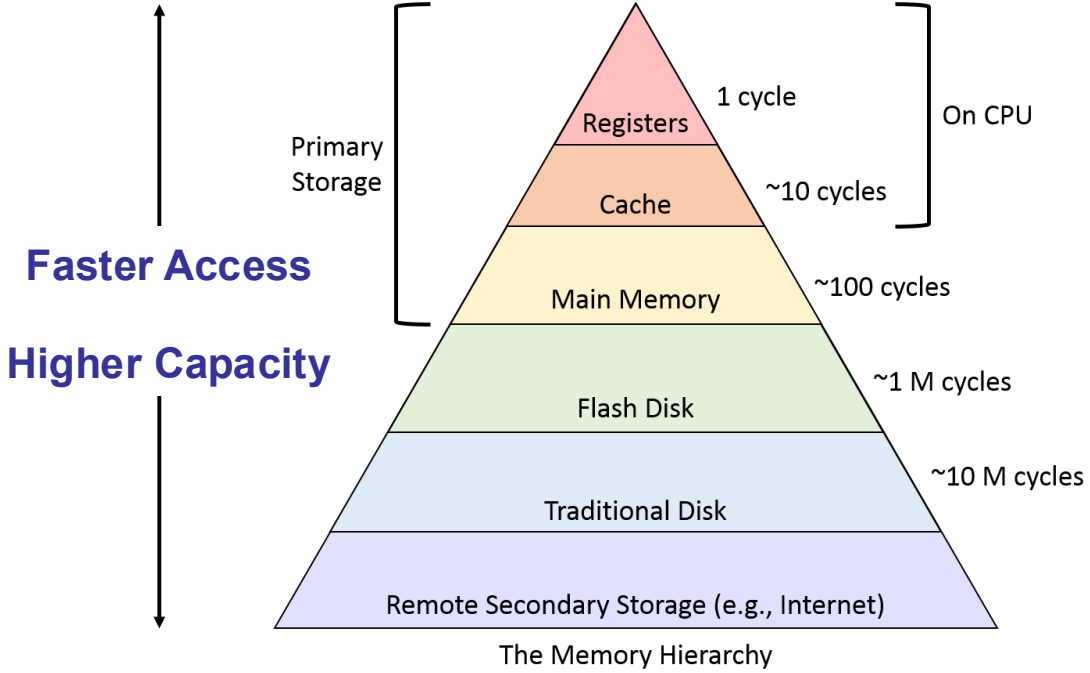
	Capacity	Latency	Cost/MB
Register (SRAM)	4 – 8 Bytes	< 1 nsec	*Not a choice*
Cache (SRAM)	1 – 32 MBytes	5 nsec	$50 – 100
Main Memory (DRAM)	4-64 GBytes	100 nsec	*$3 – 6*

Types of persistent storage (Disk)

Secondary memory used for long-term storage and when primary storage is full

	Capacity	Latency	Transfer Rate	Cost/GB
Hard Disk (HDD)	0.5 – 30TB	5 – 10 msec	100 – 200 MB/s	$0.05– $0.10
Flash (SSD)	0.5 – 30TB	0.1 – 1 msec	200 – 3,000 MB/s	$0.20 – $0.50
"Cloud" server (with its own SSD or HDD)	*Varies*	20 – 200 msec	*Varies*	*$0.02/Month*

Memory/Disk is is hierarchical



The Memory Hierarchy

I/O on Hard Disk (Rotating Magnetic)

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

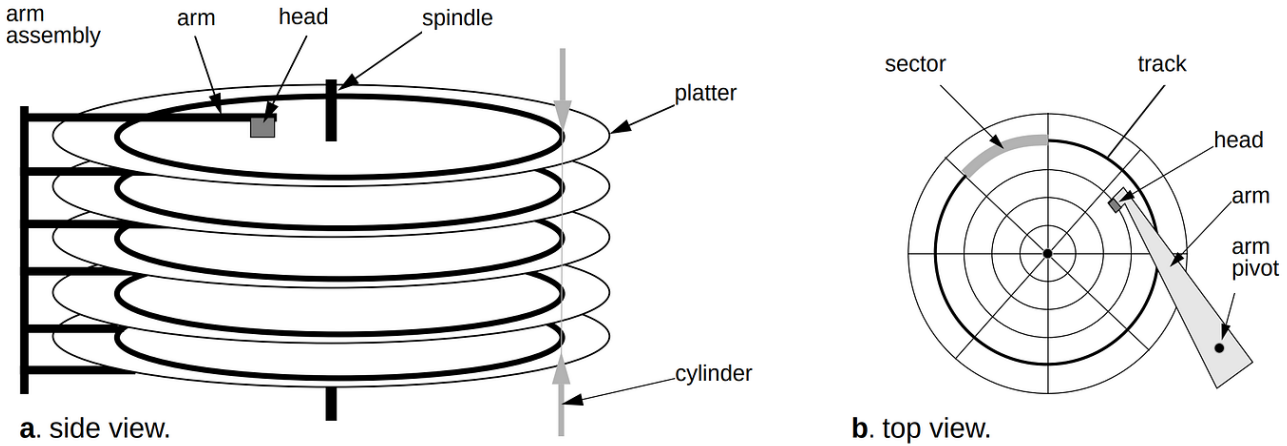
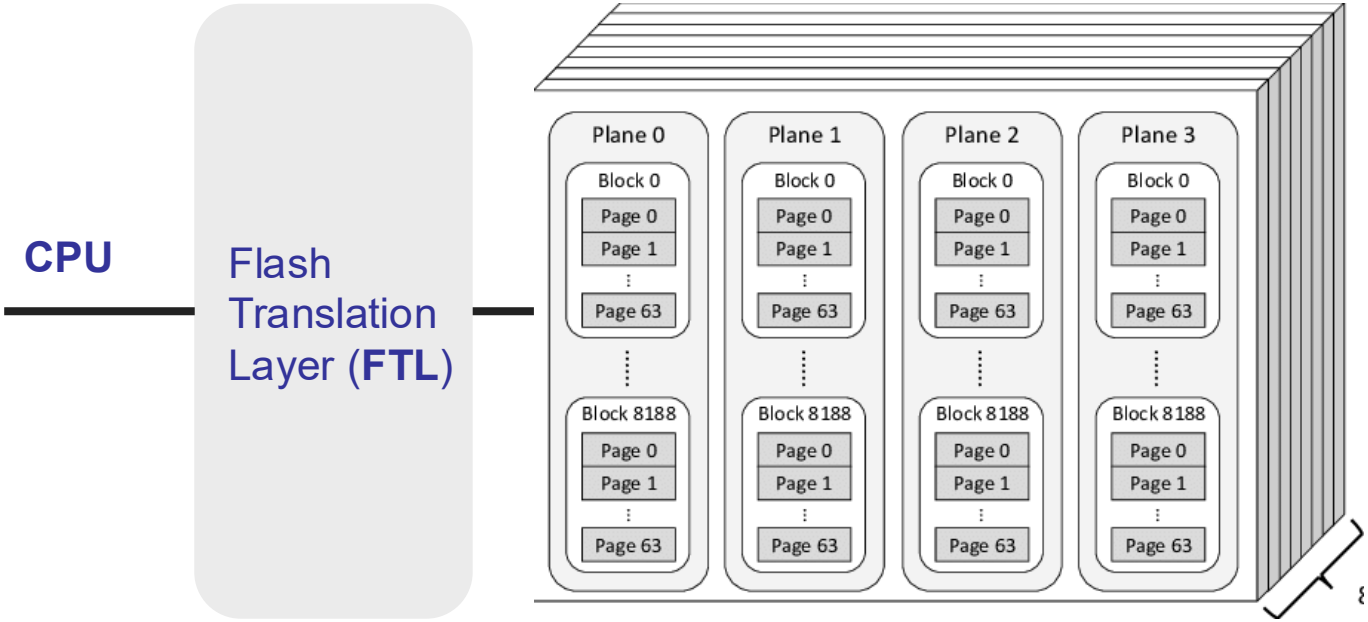


Figure 1: the mechanical components of a disk drive.

I/O on Flash (Solid State Drive)

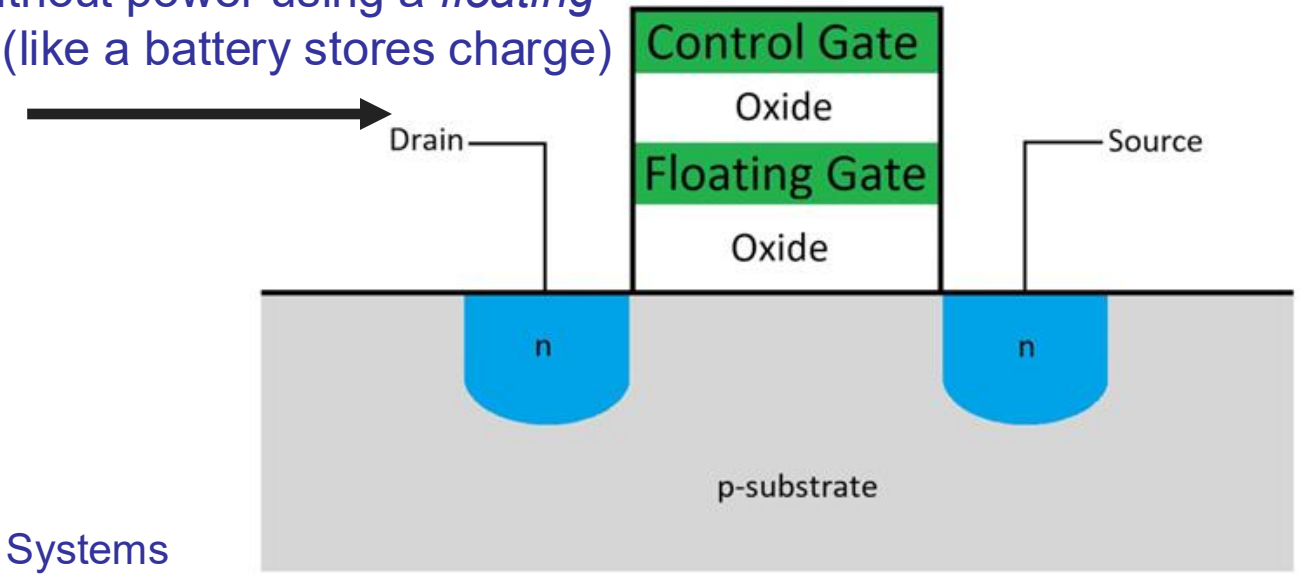
Hierarchical design – Pages can be **read *fast*** but only **written slowly** because entire block must be erased before writing new data



How does Flash store bits?

Recall **SRAM** stores a bit using an infinite switching loop that requires power

Flash stores a bit without power using a *floating* gate that is isolated (like a battery stores charge)



Review Questions

1. What device would hold data for a frequently accessed memory address.

2. When a hardware device finished a task (e.g., the user touches the mouse) how does the OS learn that the action happened?

3. How does a bit get written to flash memory (SSD)?