CS 370: OPERATING SYSTEMS [INTRODUCTION]

Computer Science
Colorado State University

Instructor: Louis-Noel Pouchet
Spring 2024

** Lecture slides created by: SHRIDEEP PALLICKARA

Course Overview

- All course materials will be accessible via the public-facing webpage

  https://www.cs.colostate.edu/~cs370

  - Schedule (Lecture slide sets for each lecture)
  - Assignments
  - Syllabus
  - Grading policy
- Grades will be posted on **Canvas**; assignment submissions will be via Canvas
- The course website, MS Teams Channel, and Canvas are all online
  - Well, the MS Teams channels will be finalized today Jan. 22 only!

Team and Office Hours

Teaching assistants:

- Temitope Adekunle (GTA)
- Tarun Sai Pamulapati (GTA)
- Nate Bennick (UTA)
- Samuel White (UTA)

Office hours:

- TAs: office hours daily, available in person and zoom/teams. For any question regarding the course, assignments, etc.
- Pouchet: office hours typically regarding lecture content: 1pm-2pm Mondays, noon-2pm Fridays

Use compsci_cs370@colostate.edu for email communications: it reaches all TAs and Pouchet

Topics Covered in CS370

- Processes and Threads
- Process Synchronization (plus Atomic Transactions)
- CPU Scheduling
- Deadlocks
- UNIX I/O
- Memory Management
- File System interface and management. Unix file system. NTFS.
- Virtualization and Containers, and modern safety mechanisms in OS

Course Textbook

Operating Systems Concepts, 9th/10th edition

Avi Silberschatz, Peter Galvin, and Greg Gagne Publisher - John Wiley & Sons, Inc.
ISBN-13: 978-1118063330.

Grading Policy

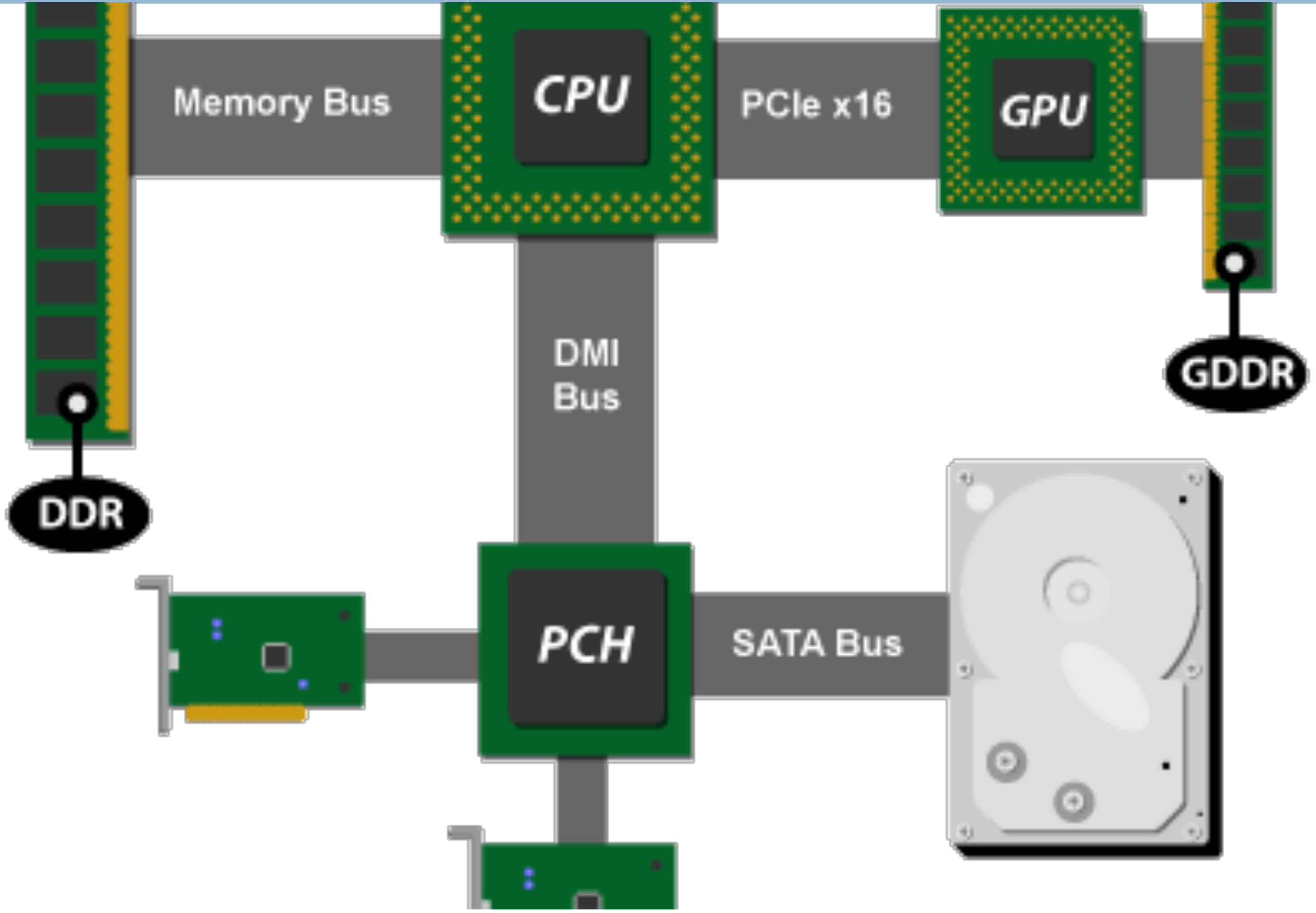
Course Element	Weight
Assignments	45% [5, 5, 5, 10, 10, 10]
Quizzes	10%
Mid Term	20%
Final Exam	25%

- Letter grades will be based on the following standard breakpoints:
  - >= 90 is an A, >= 88 is an A-, >=86 is a B+, >=80 is a B, >=78 is a B-, >=76 is a C+, >=70 is a C, >=60 is a D, and <60 is an F.
  - No cut higher than this, but *may* be cut lower (i.e., higher letter grade than displayed above)

Topics covered in this lecture

- Secondary storage
- Relative speeds of the memory hierarchy
- Multiprogramming and time sharing
- Programs and processes
- Program constructs

Reminder: Computer Organization



Grabbed from arstechnica.com

Secondary storage is needed to hold large quantities of data permanently

- Programs use the disk as the source and destination of processing
- Seek time 7 ms
- SPIN: 7200 – 15000 RPM
- Transfer rate
  - Disk-to-buffer: 70 MB/sec (SATA)
  - Buffer-to-Computer: 300 MB/sec
- Mean time between failures
  - 600,000 hours
- 1 TB capacity for less than $100

Improvements in hard disk capacity

- 1980 - 5 MB
- 1991 - 100 MB
- 1995 - 2 GB
- 1997 - 10 GB

Improvements in hard disk capacity

- 2002 - 128 GB addressing space barrier [28 bits]
  - Old IDE/ATA interface: 28-bit addressing
  - $2^{28} \times 512 = 2^{28} \times 2^{9} = 2^{37} = 128$ GB = 137,438,953,472 bytes
- 2003 – Serial ATA introduced
- 2005 - 500 GB hard drives
- 2008 - 1 TB hard drives

Characteristics of peripheral devices & their speed relative to the CPU

Item	time	Scaled time in human terms (2 billion times slower)
Processor cycle	0.5 ns (2 GHz)	1 second
Cache access	1 ns (1 GHz)	2 seconds
Memory access	70 ns	140 seconds
Context switch	5,000 ns (5 µs)	167 minutes
Disk access	7,000,000 ns (7 ms)	162 days
Quantum	100,000,000 ns (100 ms)	6.3 years

Mechanical nature of disks limits their performance

- Disk access times *have not* decreased exponentially.
  - Processor speeds are growing exponentially

- Disparity between processor and disk access times continues to grow.
  - 1:14,000,000

Relative speeds of the memory hierarchy

Since caches have limited size, cache management is critical

Level	1	2	3	4
Name	registers	cache	Main memory	Disk Storage
Typical Size	< 1 KB	< 16 MB	< 64 GB	> 100 GB
Implementation Technology	Custom memory, CMOS	On/off chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access times	0.25 ns	0.5-25 ns	80-250 ns	> 5 ms
Bandwidth (MB/sec)	20,000 – 100,000	5000-10,000	1000-5000	80-300
Managed by	compiler	hardware	OS	OS
Backed by	cache	Main memory	Disk	CD/Tape

DEVICE CONTROLLERS & I/O

A large portion of the OS code is dedicated for managing I/O

- A typical system comprises CPUs and multiple device controllers connected through a **bus**

- High end systems use switch based architecture
  - Components talk to each other concurrently
  - No competition for cycles on the bus

Device controllers and drivers

- A device controller is responsible for a specific type of device.
  - More than 1 device may be attached
- There is a **device driver** for each controller

Device controllers move data between its local buffer storage & peripheral devices

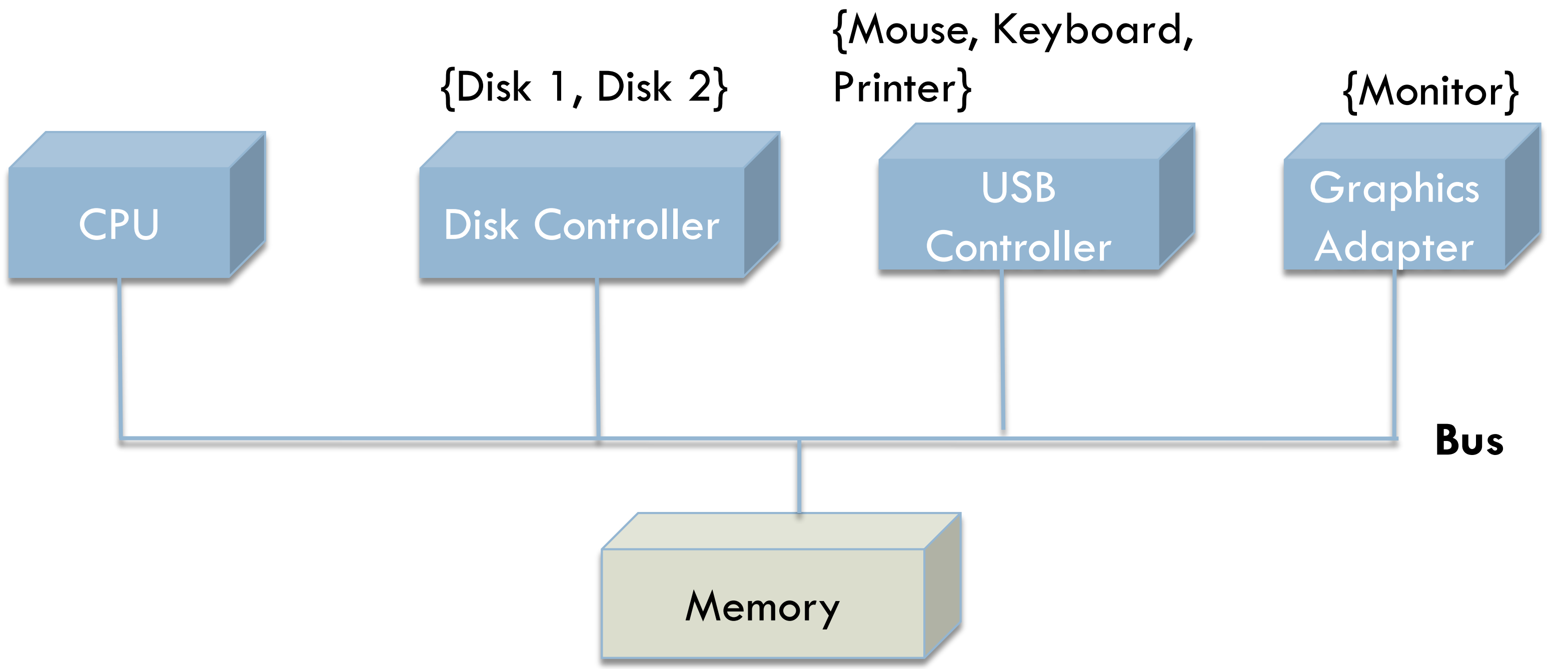
- Device driver loads appropriate registers in the controller
- Controller examines contents to determine action to take
- Controller transfers data from device to its local buffer
- Once transfer is complete, controller informs driver via an **interrupt**
- Device driver then returns control to the OS

Direct memory access is much faster than interrupt driven I/O

- Controller sets up buffers, pointers, and counters for IO device
- **Transfer entire block** of data directly to (or from) its own buffer storage to main memory
  - **No** CPU intervention needed
- Only *one* interrupt per block
  - As opposed to interrupts-per-byte for low speed devices

Buses

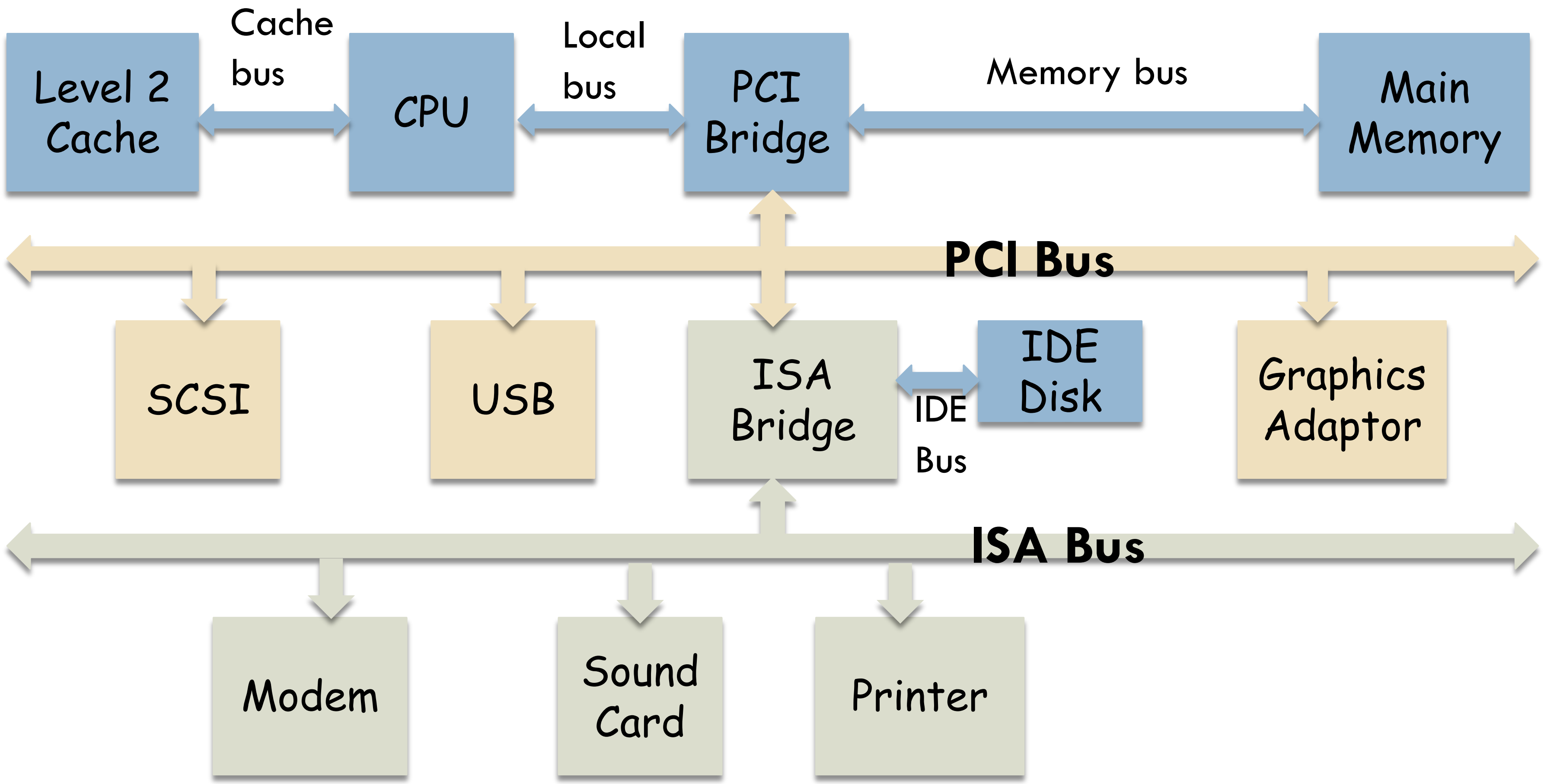
A simple bus-based structure



Limitations of the bus structure from the earlier slide

- As processors and memories got faster
  - Ability of a single bus to handle *all traffic* strained considerably
- Result?
  - Additional buses were added
  - For faster I/O devices and CPU-memory traffic

What a modern bus architecture looks like



There are two main BUS standards

- Original IBM PC ISA (Industry Standard Architecture)

- PCI (Peripheral Component Interconnect)
  - From Intel

The IBM PC ISA bus

- Runs at 8.33 MHz
- Transfers 2 bytes at once
- Maximum speed = 16.67 MB/sec
- Included for backward compatibility
  - Older and slower I/O cards

The PCI bus

- Can run at 66 MHz
- Transfer 8 bytes at once
- Data transfer rate: 528 MB/sec
- Most high-speed I/O devices use PCI
- Newer computers have an updated version of PCI
  - **PCI Express**

Other specialized buses: IDE (Integrated Drive Electronics) bus

- For attaching peripheral devices
  - CD-ROMs and Disks
- Grew out of the disk controller interface

Other specialized buses: USB (Universal Serial Bus)

- Attach **slow** I/O devices to the computer
  - Keyboard, mouse etc
- Uses a small **4-wire** connector
  - **Two** supply electrical power to the USB devices
- Centralized bus
  - Root device polls I/O devices every 1 millisecond
    - Check if they have any traffic

Some more information about USB

- All USB devices share a **single** USB device driver
  - *No need to install* a driver for each device
  - Can be added to computer *without need to reboot*
- USB 1.0 has a transfer rate of 1.5 MB/sec
- USB 2.0 goes up to 60 MB/sec
- USB 3.0
  - Specification ready on 17 November 2008
  - Theoretical signaling rate: 600 MB/sec (4.8 Gbps)
  - USB 3.1: Jan 2013 will go to 10 Gbps
    - On par with Thunderbolt (developed by Apple and Intel in 2011)

Other buses

- SCSI (Small Computer System Interface)
  - High performance bus
  - For devices that need high bandwidth
    - Fast disks, scanners
  - Up to 320 MB/sec

- IEEE 1394
  - Sometimes called FireWire (used by Apple)
  - Transfer speeds of up to 100 MB/sec
    - Camcorders and similar multimedia devices
  - No need for a central controller (unlike USB)

In this setting the OS must know which devices are connected & how to configure them

- Led Intel and Microsoft to design **plug-and-play**
  - Similar concept had been implemented in the Mac

How things were before plug-and-play

- Each I/O card had a **fixed interrupt level**
  - Fixed addresses for its I/O registers

Device	Interrupt/I/O addresses
Keyboards	Interrupt 1, I/O addresses: 0x60-0x64
Floppy disk controller	Interrupt 6, I/O addresses: 0x3F0-0x3F7
Printer	Interrupt 7, I/O addresses: 0x378-0x37A

How things were before plug-and-play

- What if someone bought a sound card and a modem which happened to use interrupt 4?
  - Conflict
  - Would not work together
- Solution:
  - Use DIP (dual in-line package) switches or jumpers on every I/O card
  - Ask user to select interrupt level and I/O device addresses for the device
  - Tedious!

How does Plug-and-play work?

① Automatically **collect** information about devices

② Centrally **assign** interrupt levels + I/O addresses

③ **Tell** each card what its numbers are

Performance

Single processor systems have 1 CPU that can execute *general-purpose* instructions

- The system may have **special purpose processors**
  - Incapable of running user processes
  - Limited instruction set
- Disk controller micro-processor
  - Implements disk queue and scheduling algorithms
- Keyboard microprocessors
  - Convert keystrokes into CPU-bound codes

There are two approaches to improving performance

- Determine component **bottlenecks**
  - Replicate
  - Improve

To replicate or improve?

"If one ox could not do the job, they [pioneers] *did not grow a bigger ox, but used two oxen."*

-- Admiral Grace Murray Hopper
Computer Software pioneer

"If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?"

-- Seymour Cray
Computer Hardware pioneer

Multiprocessor systems have 2-or-more processors in close communications

- The processors share the bus, and *may* share clock, memory and peripheral devices
- Advantages:
  - Increased throughput
  - Reliability

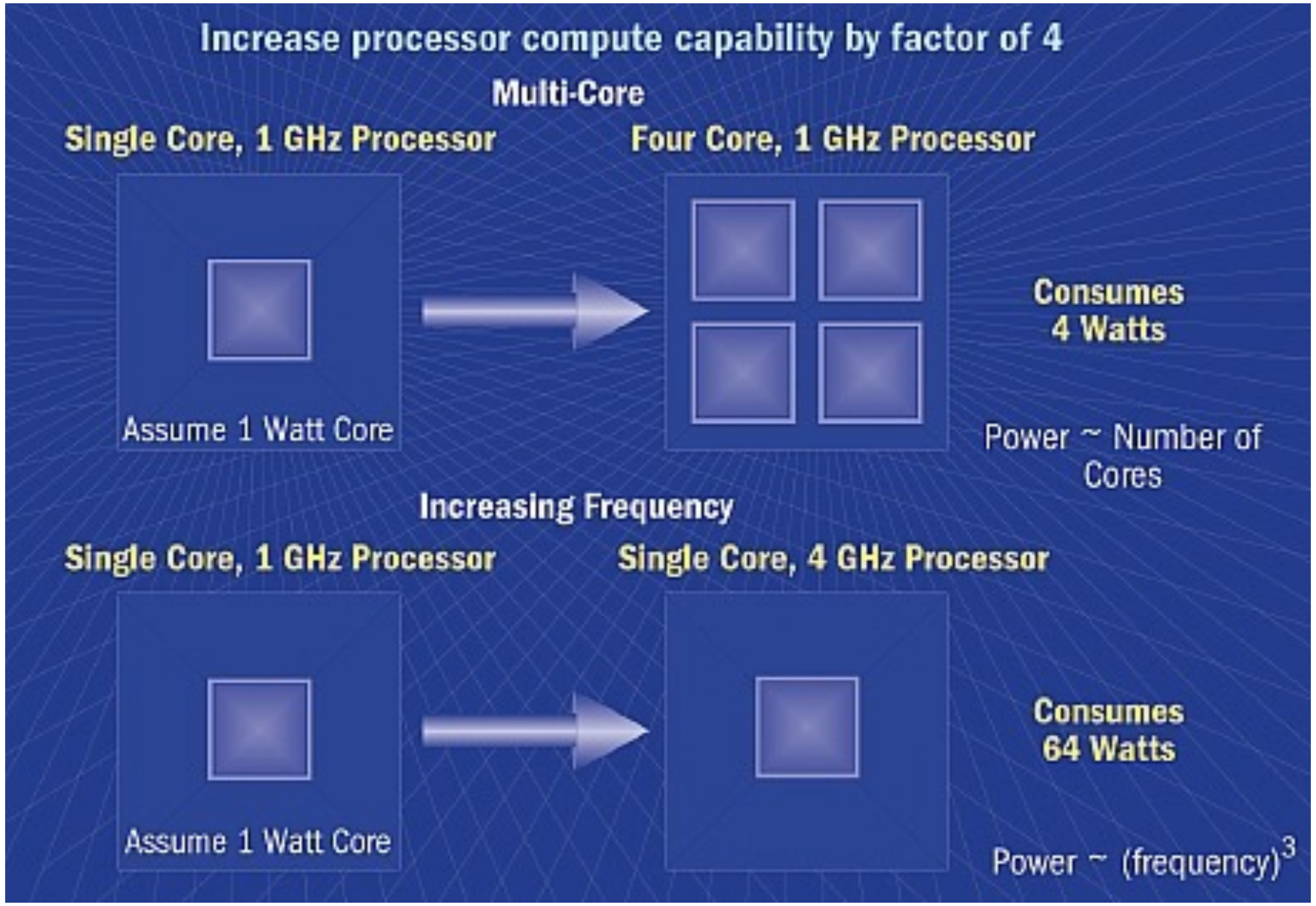
Multiprocessor systems fall in two categories based on control

- Asymmetric multiprocessing:
  - Controller processor manages the system
  - Workers **rely on controller** for instructions

- Symmetric multiprocessing
  - Processors are **peers** and perform all OS tasks
  - Have own set of registers and local cache
    - Share physical memory
  - **Supported by virtually all modern OS**

Trend: going multi-core for CPUs

- Driven by power / physics
- Problem: parallelism in the application?
- We merely see 16-core CPUs as HEDT in 2024



Increase processor compute capability by factor of 4
Multi-Core
Single Core, 1 GHz Processor
Four Core, 1 GHz Processor
Assume 1 Watt Core
Consumes 4 Watts
Power ~ Number of Cores
Increasing Frequency
Single Core, 1 GHz Processor
Single Core, 4 GHz Processor
Assume 1 Watt Core
Consumes 64 Watts
Power ~ (frequency)^3

Grabbed from DoE Scidac

Multiprogramming

Multiprogramming organizes jobs so that the CPU always has one to execute

- A single program (generally) cannot keep CPU & I/O devices busy at all times
- A user frequently runs multiple programs
- When a job needs to **wait**, the CPU **switches** to another job.
- Utilizes resources (cpu, memory, peripheral devices) effectively.

Time sharing is a logical *extension* of the multiprogramming model

- CPU switches between jobs **frequently**, users can interact with programs
- Time shared OS allows many users to use computer simultaneously
- Each action in a time shared OS tends to be **short**
  - CPU time needed for each user is small

Grocery checkout : Several checkout counters (processes) & 1 checker (CPU)

- Multiprogramming
  - Checker checks one item (instruction) at a time
  - Continue checking till price check
  - During price check move to another counter
- Time sharing
  - Checker starts a 10-second timer
  - Process items for maximum of 10 seconds
    - Move to another customer <u>even if there is NO</u> price check

Multiprogramming requires several jobs to be held simultaneously in memory

- Job scheduling: Decision about which of the *ready* jobs need to brought into memory
- CPU scheduling: Deciding which job needs to be run
- Swapping: The shuffling of processes in and out of memory to the disk

Programs and Processes

Programs and processes: Process is a program in execution.

- Programs are *passive*; processes are **active**
- Processes **need resources** to accomplish task
- Single-threaded processes have one program counter pointing to next instruction to execute
- Multithreaded processes have <u>multiple</u> program counters
  - One for each thread

Some terms related to processes

- **Context switch** time: Time to switch from executing one process to another
- **Quantum**: Amount of CPU time allocated to a process before another process can run

OS process management activities

- Schedule processes and threads on CPUs
- Create and delete processes
- Suspend and resume
- Mechanisms for process synchronization
- Mechanisms for process communications

System Calls

System Calls

- Request to the OS for service
- Causes normal CPU processing to be interrupted
- Control to be given to the OS

System calls provide an interface to OS services

- **Runtime support** for most languages provide a system call interface.
- API hides details of the OS interface
- Runtime library manages the invocation
- Passing parameters to the OS
  - Registers
  - Block, or table, in memory
  - Stack: *pushed* by the program, *popped* by OS

Types of system calls

- Process control
- File manipulation
- Device manipulation
- Information maintenance
- Communications
- Protection

Mode bit allows us to distinguish between task executed on behalf of OS/user

- **Mode bit**: kernel (**0**) and user (**1**)
- Designate (potentially harmful) machine instructions as **privileged** instructions.
  - Hardware enforces kernel mode executions

Mode bit

- MS-DOS/Intel 8088 had no mode bit
  - No dual-mode
  - A program can wipe out OS by writing over it
- Most modern OS take advantage of **dual mode** and provide greater protection for OS.

VIRTUAL MEMORY

Main memory is generally the only large storage device the CPU deals with

- To execute a program, it must be **mapped** to absolute addresses and loaded into memory
- Execution involves accesses to instructions and data from memory
  - By generating absolute addresses
- When program terminates, memory space is **reclaimed**

What do we do if there are more processes than memory to accommodate all of them?

Virtual memory allows processes not completely memory resident to execute

- Enables us to run programs that are **larger** than the actual physical memory
- Separates **logical memory** as viewed by user from *physical memory*
- Frees programmers from memory storage limitations

Program Constructs

Important Program Constructs

- Communication, Concurrency & Asynchronous operation
- Challenges & Implications
  - Improper handling can lead to failures for no apparent reason
  - Run for weeks or *months*
  - Avoid resource leaks
  - Cope with *outrageously malicious* input
  - Recover from errors

Program Construct: Asynchronous operation

- Events happen at unpredictable times AND in unpredictable order.
  - Interrupts from peripheral devices
  - For e.g. keystrokes and printer data
- To be *correct*, a program must work will ***all*** possible timings
- Timing errors are very hard to repeat
- SYNCHRONOUS: Divide by zero

Program Construct: Concurrency

- Sharing resources in the same ***time frame***
- Interleaved execution
- Major task of modern OS is **concurrency control**
- Bugs are hard to reproduce, and produce unexpected side effects

Concurrency occurs at the hardware level because devices operate at the same time

- Interrupt: **Electrical signal** generated by a peripheral device to set hardware flag on CPU
- Interrupt detection is part of instruction cycle
- If interrupt detected
  - **Save current** value of program counter
  - **Load new** value that is address of interrupt service routine or interrupt handler: device drivers
    - Drivers use signals (software) to notify processes

Signal is the software notification of an event

- Often a *response* of the OS to an interrupt
  - OS uses signals to notify processes of completed I/O operations or errors
- Signal generated when event that causes signal occurs
  - For example: keystroke and Ctrl-C
- A process catches a signal by executing handlers for the signal

Concurrency constructs: I/O operations

- Coordinate resources so that CPU is not idle
- Blocking I/O blocks the progress of a process
- Asynchronous I/O (dedicated) threads circumvent this problem
- Ex: Application monitors 2 network channels
  - If application is blocked waiting for input from one source, it *cannot respond* to input on 2nd channel

Concurrency constructs: Processes & threads

- User can create multiple processes; `fork()` in UNIX
- Inter process communications
  - Common ancestor: pipes
  - No common ancestor: signals, semaphores, shared address spaces, or messages
- Multiple threads within process = concurrency

The contents of this slide-set are based on the following references

- *Andrew S Tanenbaum. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 1]*
- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 1, 2]
- *Kay Robbins & Steve Robbins. Unix Systems Programming, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2.* [Chapter 1]