



**Department of Physics,  
Computer Science & Engineering**

CPSC 410 – Operating Systems I

# Computer System Overview

Keith Perkins

Original slides by Dr. Roberto A. Flores

Additional content from [https://onlinecourses.nptel.ac.in/noc17\\_cs29/preview](https://onlinecourses.nptel.ac.in/noc17_cs29/preview)

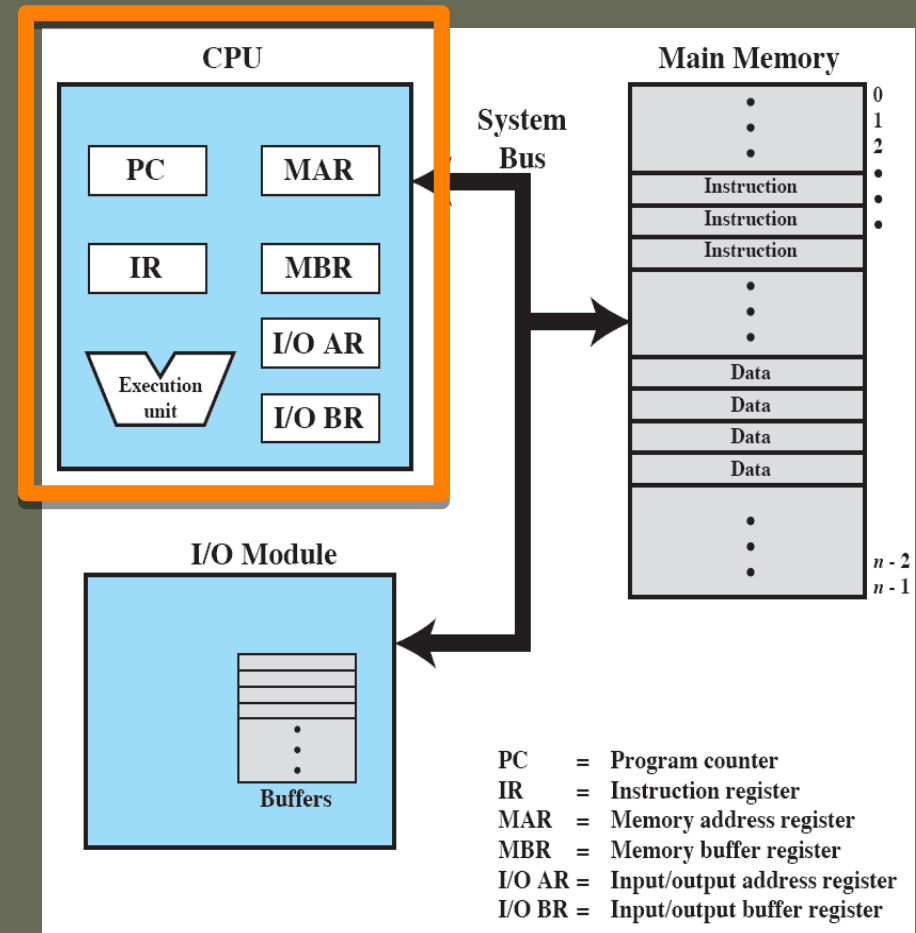
# Topics

---

- **Basic Elements**
  - Processor, main memory, I/O modules, system bus
- **Microprocessors**
  - General purpose, graphics, digital signal, system on chip
- **Instructions**
  - Execution, fetch & execute (F&E), instruction register
- **Interrupts**
  - Types, flow of control, F&E&I, multiple interrupts
- **Memory**
  - Hierarchy, principle of locality, cache
- **I/O Techniques**
  - Programmed, interrupt-driven, direct memory access
- **Symmetric multi-processors**
  - Advantages, organization, multi-core

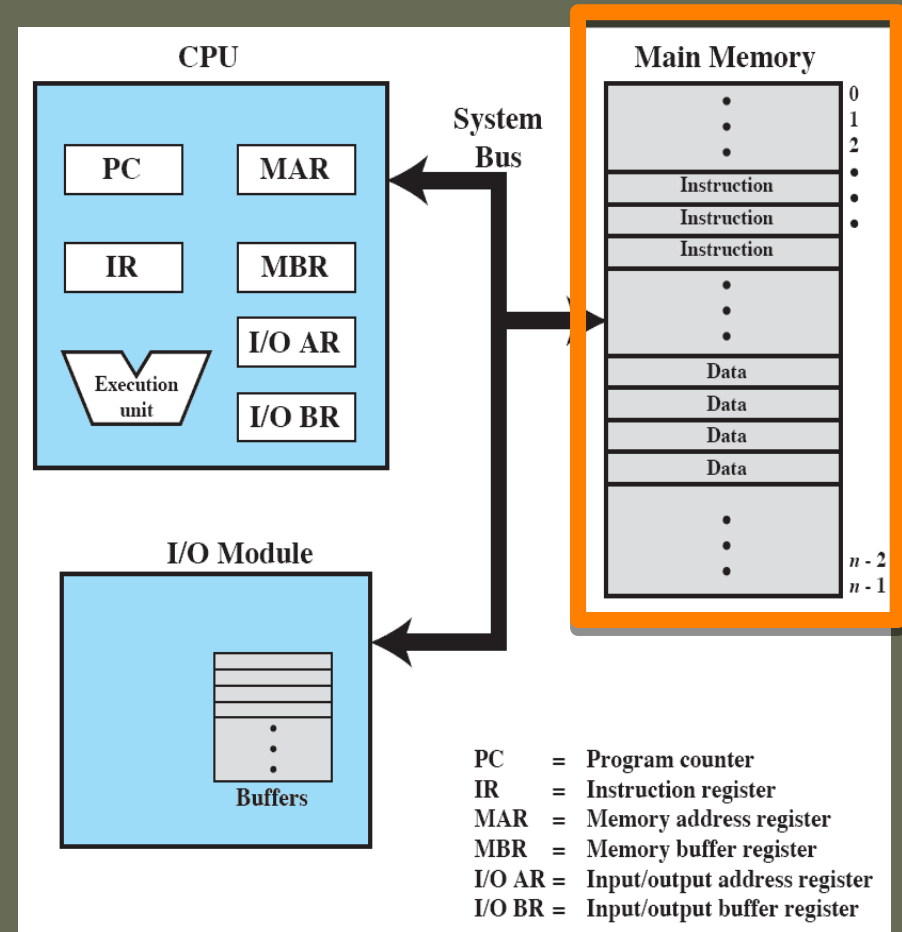
# Basic Elements

- Processor
  - aka CPU
    - ▢ Central Processing Unit
  - Controls execution of instructions
  - Performs data processing



# Basic Elements

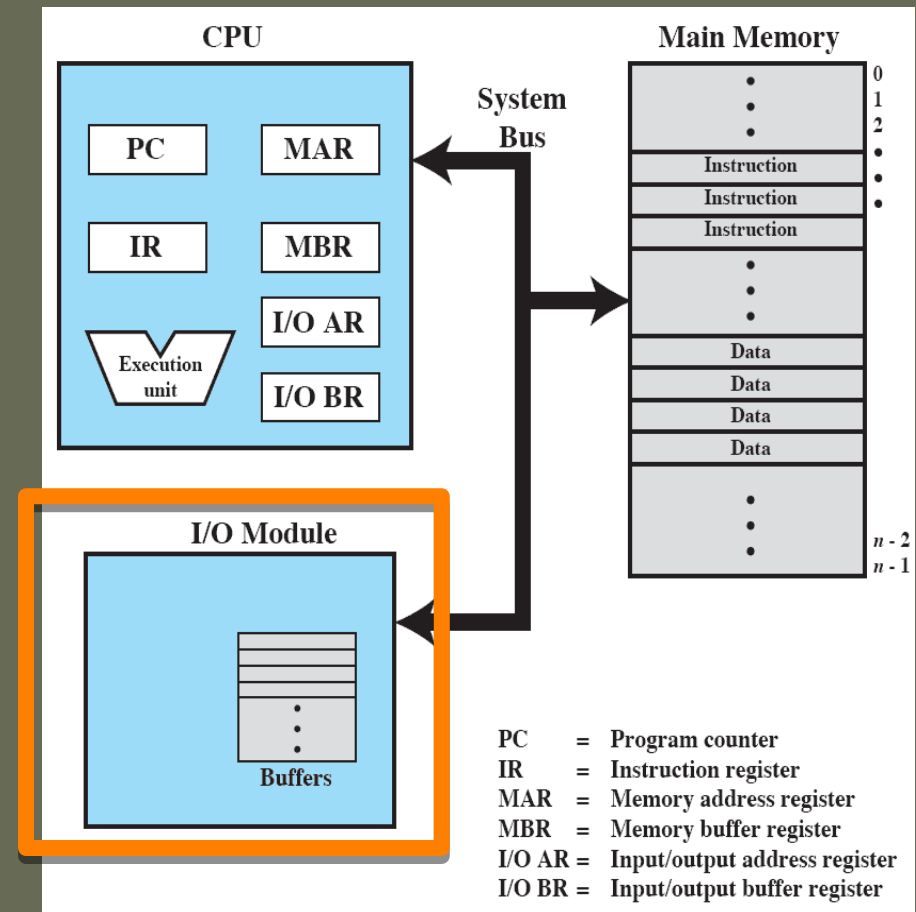
- Memory
  - aka primary/main memory, RAM
    - ▢ Random Access Memory
  - Stores instructions & data
  - Volatile
    - ▢ Contents are lost when the computer is shut down



# Basic Elements

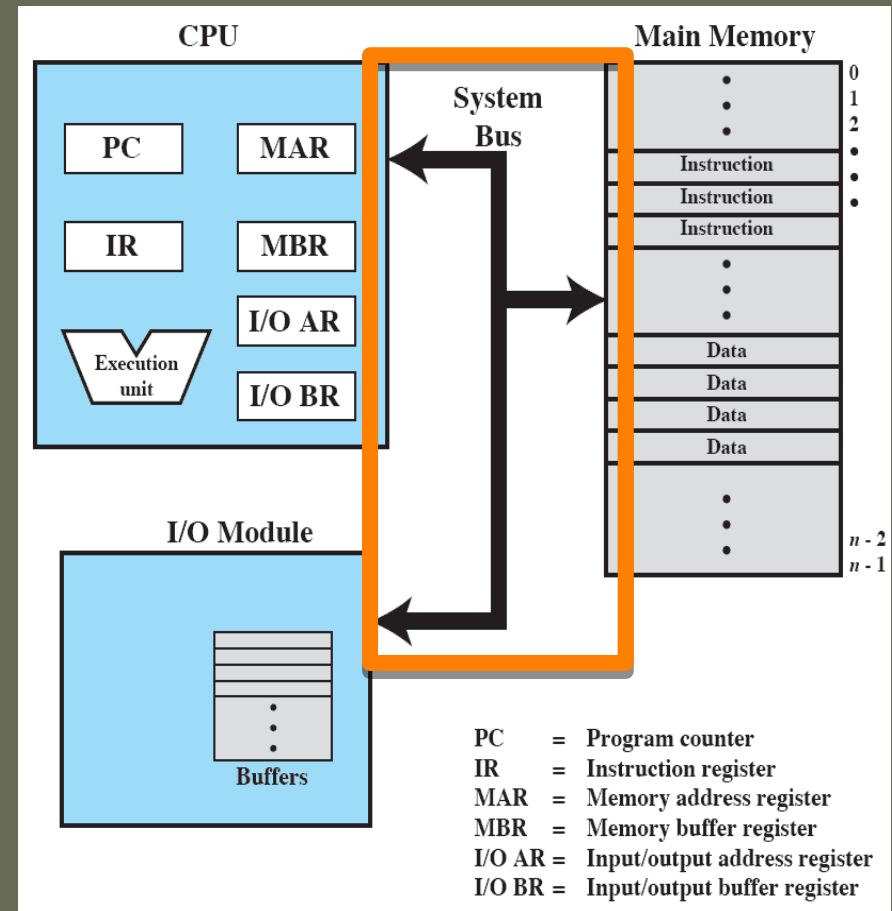
- I/O Modules

- aka device drivers
- Move data between the computer and external devices:
  - ▢ storage (e.g. hard drive)
  - ▢ communications equipment
  - ▢ terminals
- Have buffers to push/pull data



# Basic Elements

- System bus
  - Means of communication among processors, memory & I/O modules
  - Its speed limits computer performance
    - Known as the....



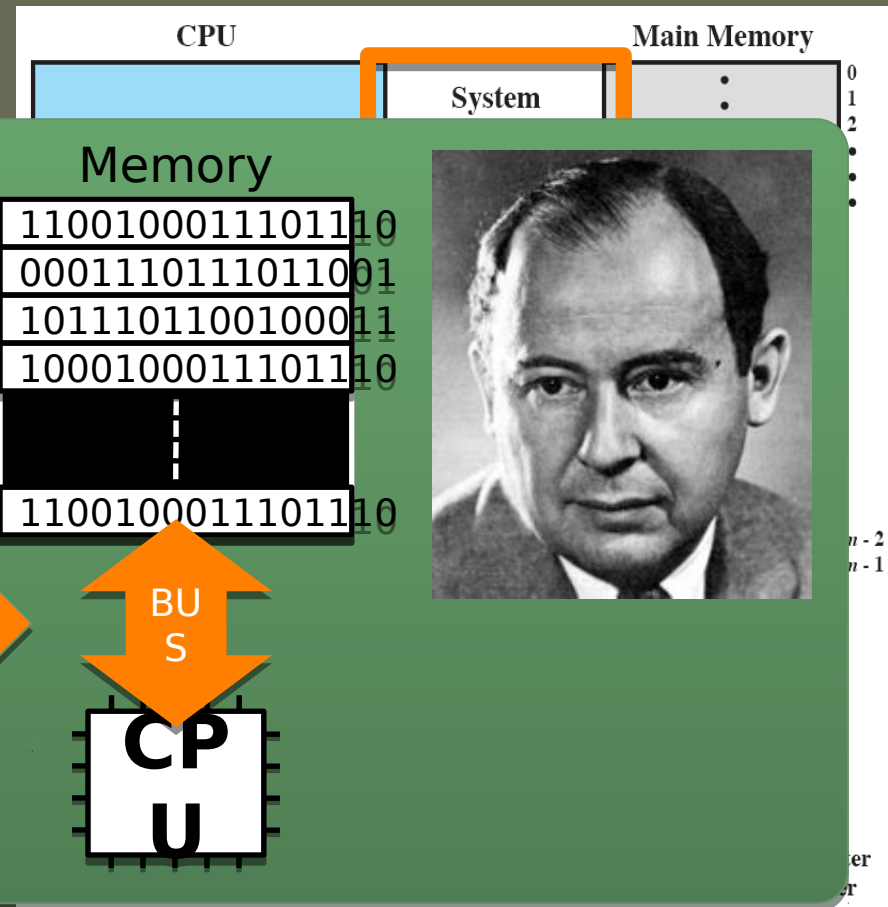
# Basic Elements

- System bus

- Means of

## von Neumann Bottleneck

- Data & Code must pass through the bus (the bottleneck)
- Physical & Intellectual barrier\*



# Topics

---

- Basic Elements

- Processor, main memory, I/O modules, system bus

- **Microprocessors**

- General purpose, graphics, digital signal, system on chip

- Instructions

- Execution, fetch & execute (F&E), instruction register

- Interrupts

- Types, flow of control, F&E&I, multiple interrupts

- Memory

- Hierarchy, principle of locality, cache

- I/O Techniques

- Programmed, interrupt-driven, direct memory access

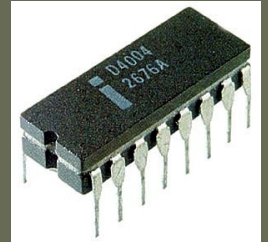
- Symmetric multi-processors

- Advantages, organization, multi-core



# Microprocessors

- General Purpose
  - It brought about PC & handheld computing
  - 1 processor or more (cores) on a single chip
- Graphical Processing Units (GPU)
  - Efficient computation on arrays of data, e.g., math & physics simulations (for games), large spreadsheets
- Digital Signal Processors (DSP)
  - Streaming audio or video signals; en/decoding (codecs)
- System on a Chip (SoC)
  - Embedded systems (handheld)
  - CPU, GPU, DSP, memory in one chip



Intel 4004, wikipedia.org



Exynos5octa, samsung.com

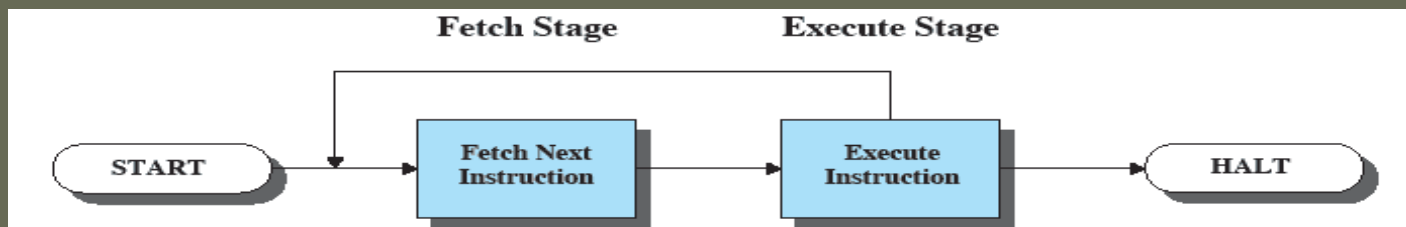
# Topics

---

- Basic Elements
  - Processor, main memory, I/O modules, system bus
- Microprocessors
  - General purpose, graphics, digital signal, system on chip
- Instructions
  - Execution, fetch & execute (F&E), instruction register
- Interrupts
  - Types, flow of control, F&E&I, multiple interrupts
- Memory
  - Hierarchy, principle of locality, cache
- I/O Techniques
  - Programmed, interrupt-driven, direct memory access
- Symmetric multi-processors
  - Advantages, organization, multi-core

# Instructions

- A program is a set of instructions stored in memory
- CPU instruction cycle (fetch & execute)
  - *program counter (PC) has address of next instruction*
  - **processor reads (fetches) an instruction from memory**
  - *instruction stored in instruction register (IR)*
  - *program counter increments address*
  - **processor executes instruction**; repeat until done

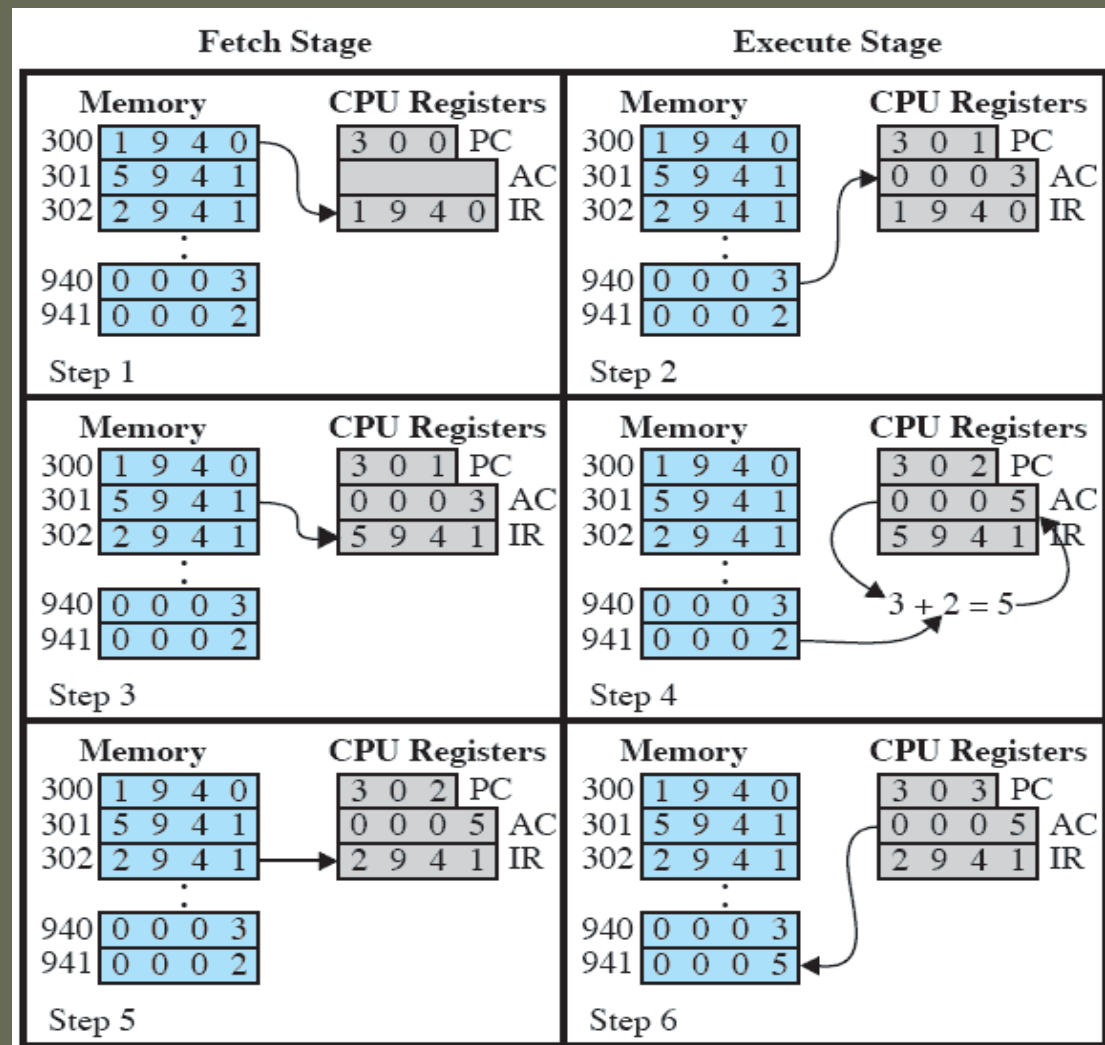


Program counter (PC) has address of next instruction  
 Processor fetches instruction from memory  
 Instruction stored in instruction register (IR)  
 Program counter increments address  
 Processor executes instruction; repeat until forever

1 load AC from memory  
 2 store store AC to memory  
 5 add to AC from memory

# Instructions

## • Example



# Topics

---

- Basic Elements
  - Processor, main memory, I/O modules, system bus
- Microprocessors
  - General purpose, graphics, digital signal, system on chip
- Instructions
  - Execution, fetch & execute (F&E), instruction register
- **Interrupts**
  - Types, flow of control, F&E&I, multiple interrupts
- Memory
  - Hierarchy, principle of locality, cache
- I/O Techniques
  - Programmed, interrupt-driven, direct memory access
- Symmetric multi-processors
  - Advantages, organization, multi-core

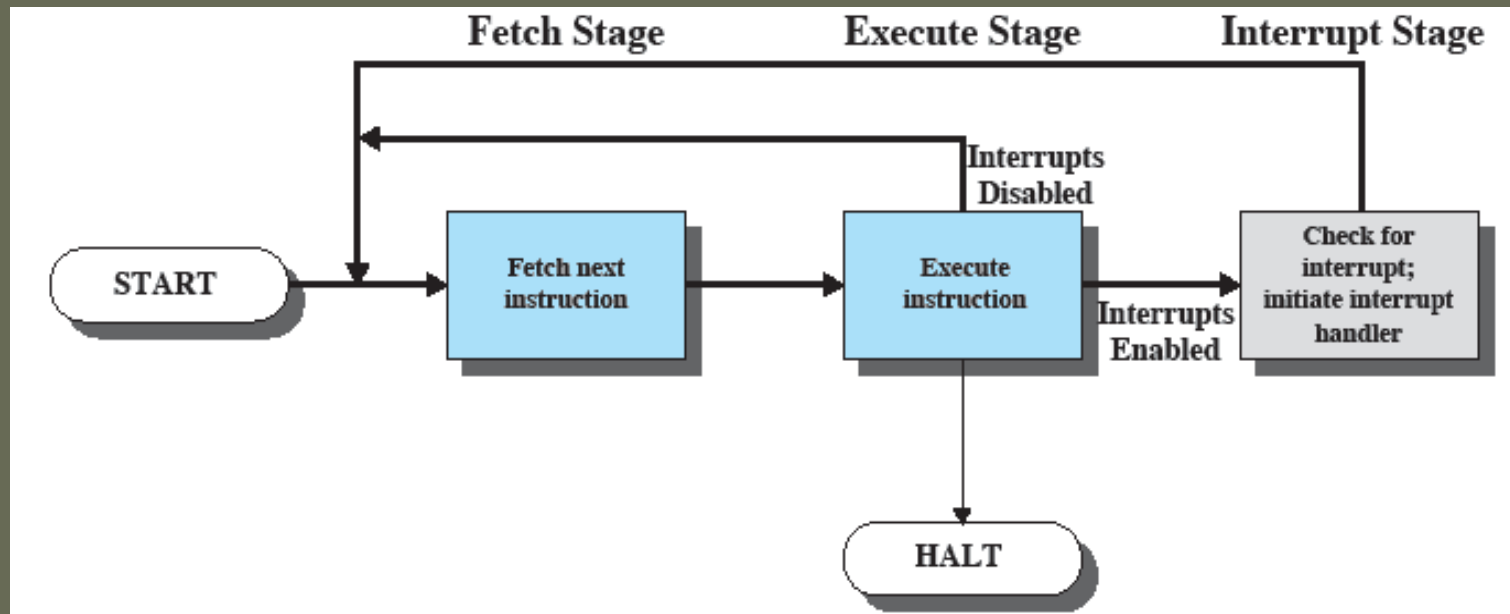
# Interrupts

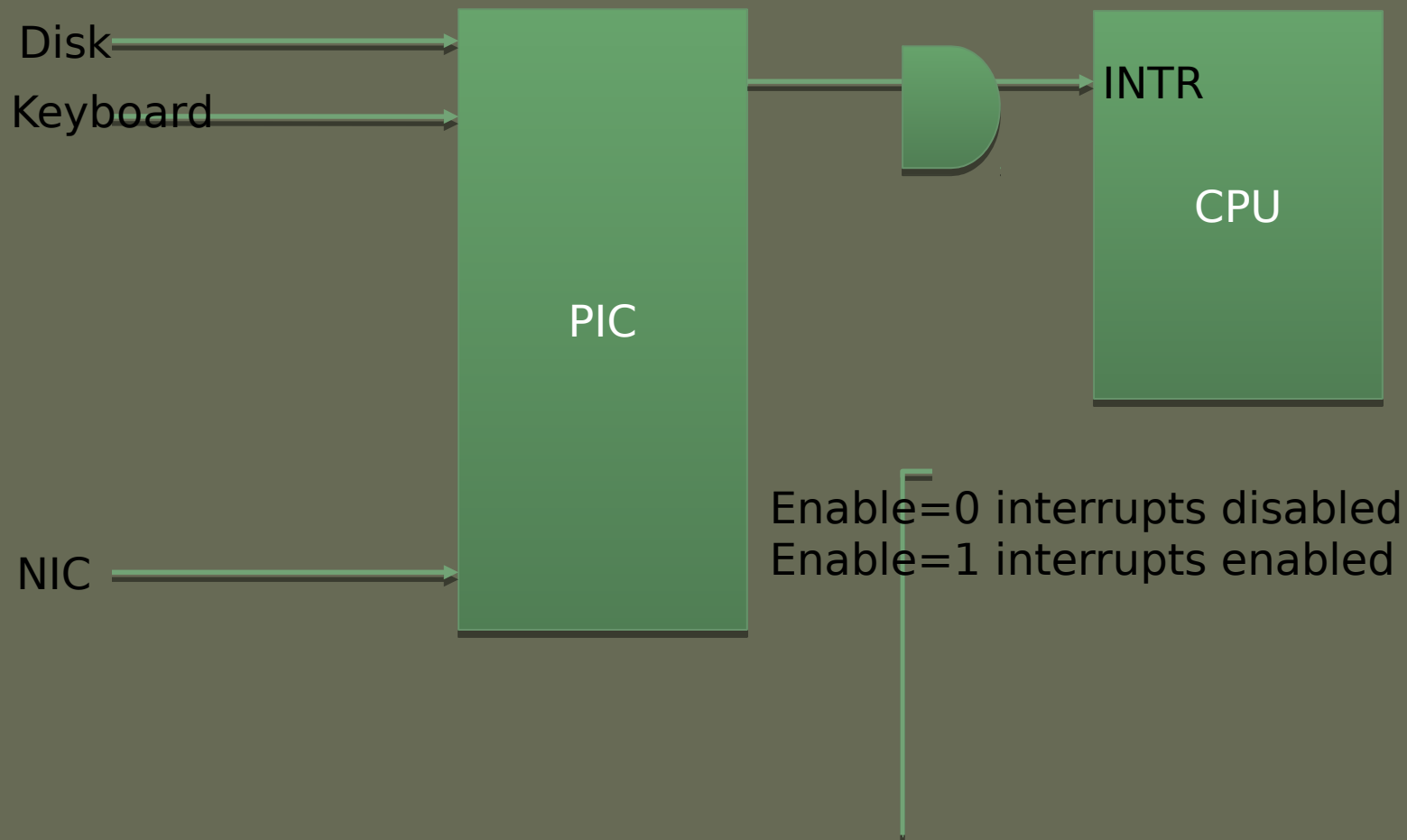
---

- Interrupts normal sequencing of the processor
- Provided to improve processor utilization
  - I/O devices are slower than CPU
  - Without interrupts, CPU must pause to wait for I/O device (wastes its time)
- Interrupts ensure timely process switches
- Interrupts provide safe user access to potentially dangerous instructions (like file read/write)

# Interrupts- where in instruction cycle

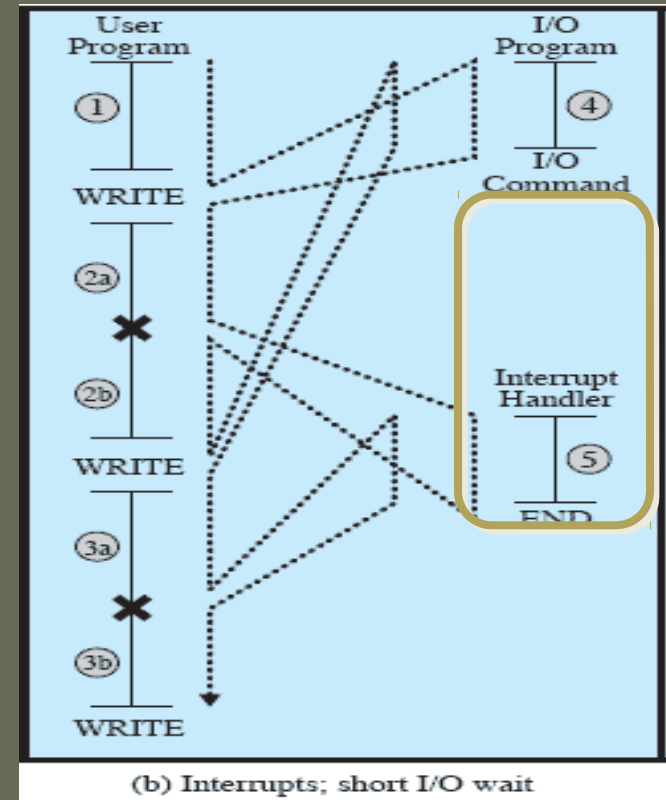
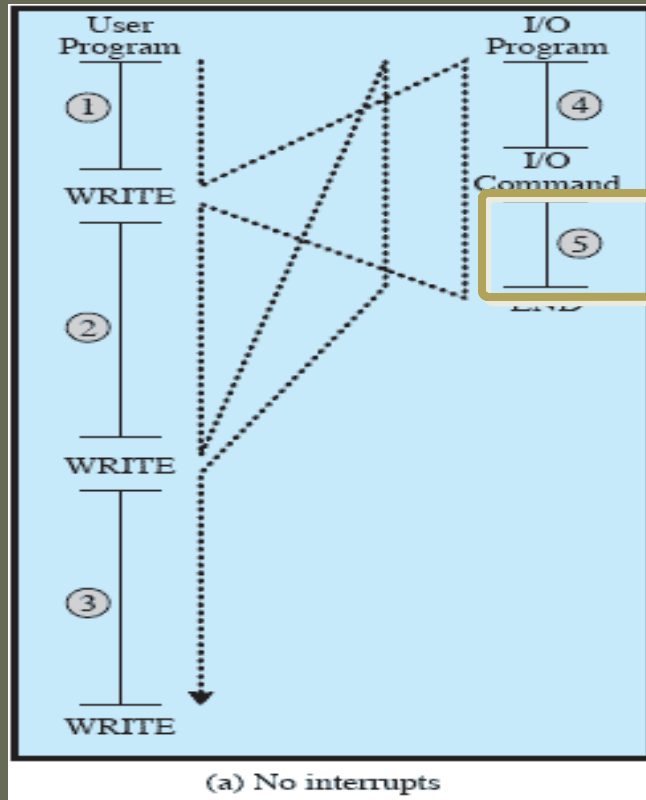
- Fetch & Execute & Interrupt
  - Same as before, plus an Interrupt Stage







# Interrupts-Why



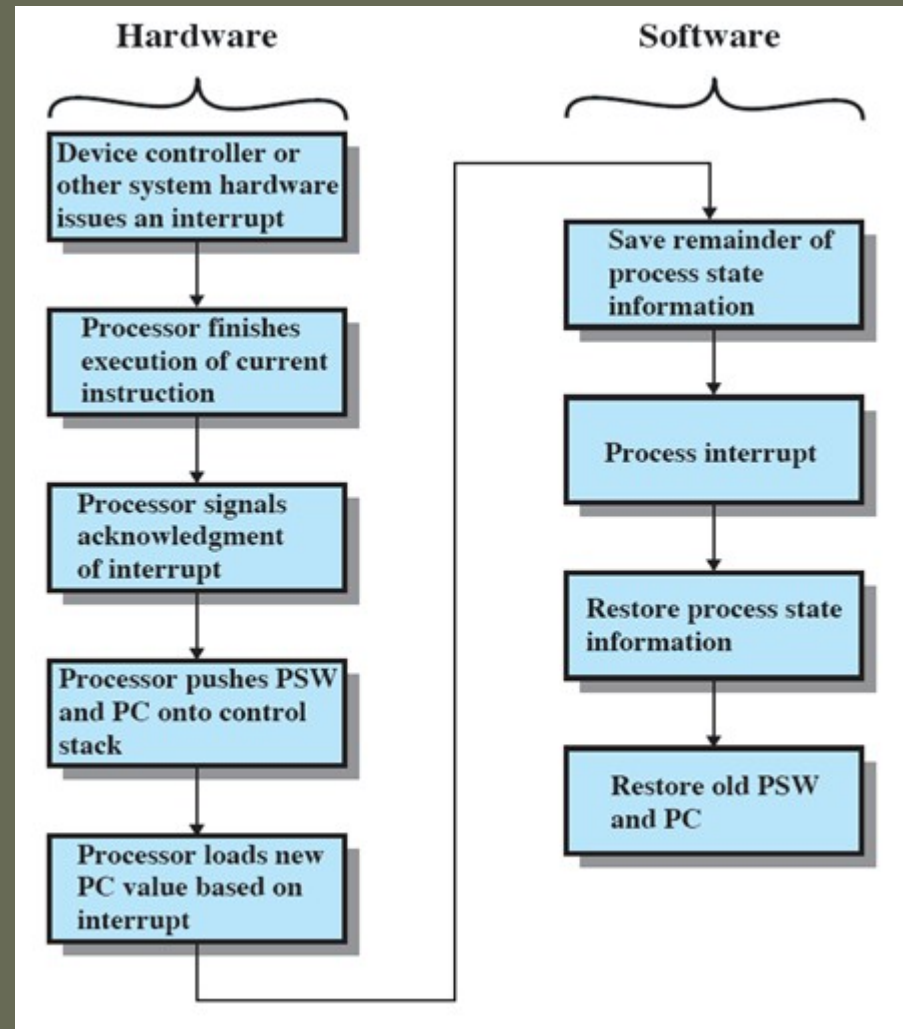
- Polling (left side) CPU periodically checks device to see if it needs attention. Almost always a huge waste of time.
- Instead, start operation, have CPU do other work until notified by interrupt that operation is finished

# Example - why we need interrupts (or why polling is often bad)

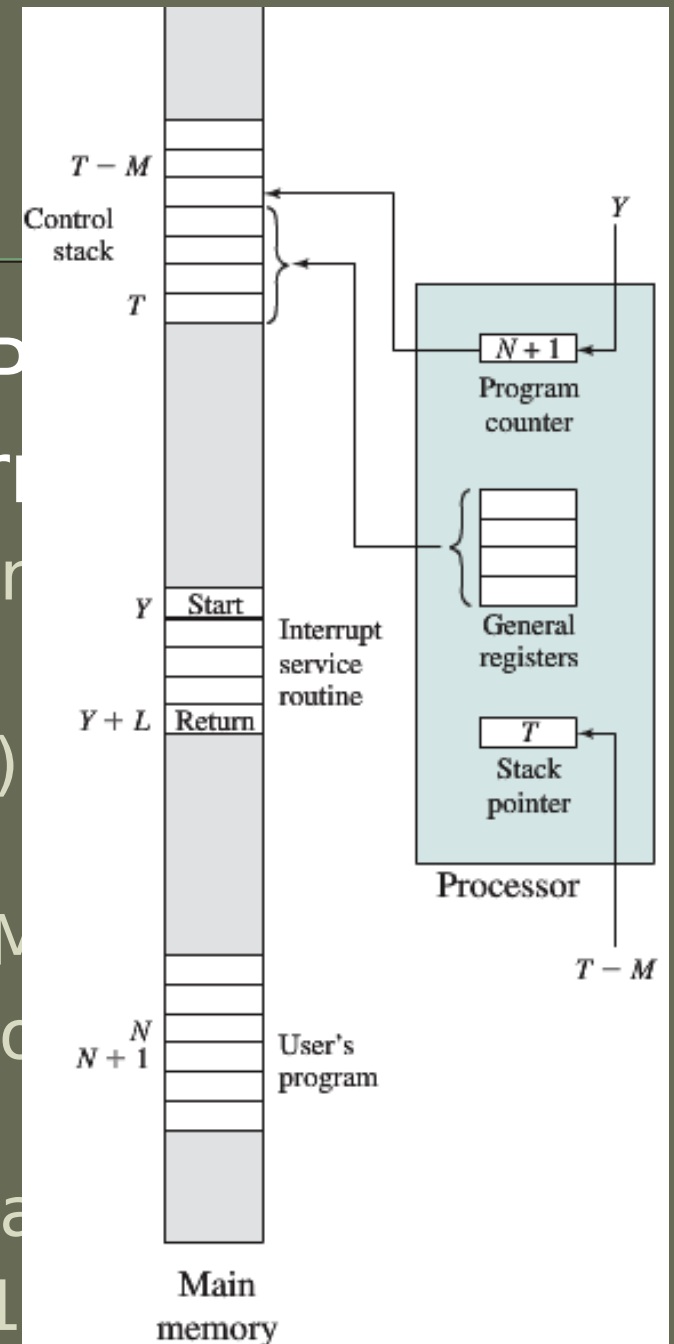
---

- CPU clock 2.5 Ghz or  $4 \times 10^{-10}$  seconds per instruction
- Hard Drive  $4 \times 10^{-3}$  seconds per access
- CPU is 10 million times faster than HD
- Or, if 1 CPU instruction took 1 second, 1 HD access would take 16.5 weeks

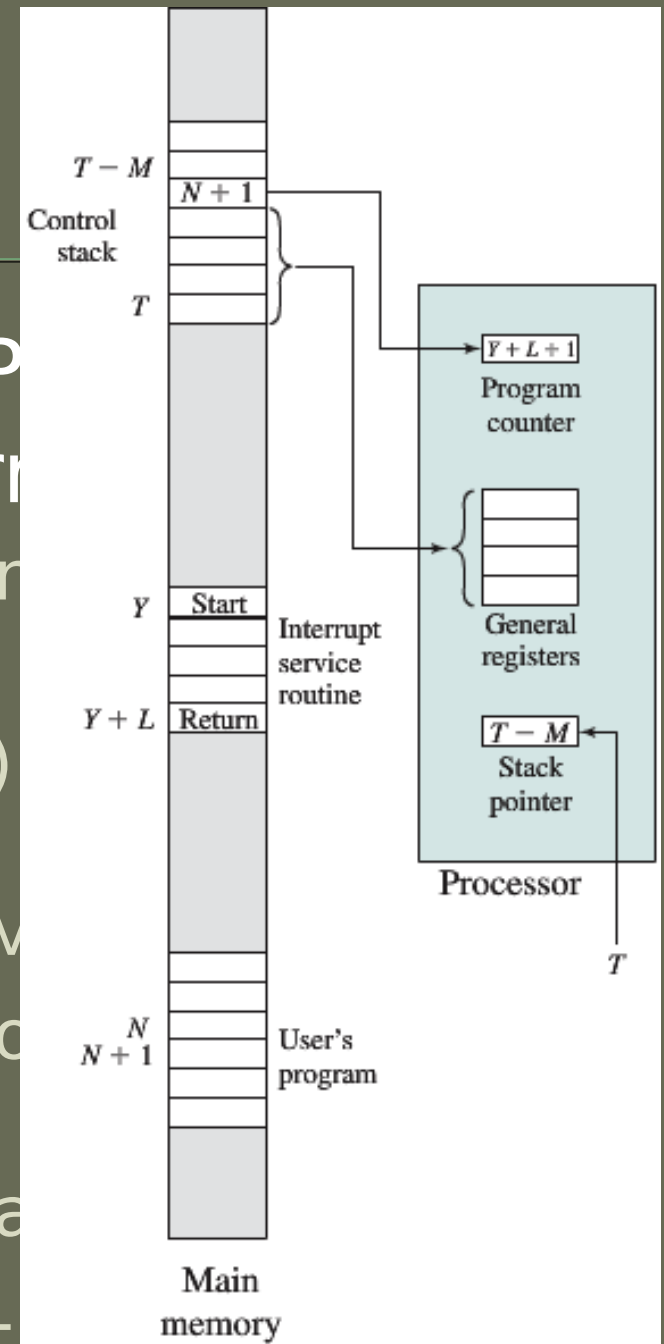
# Simple Interrupt Processing



- What happens when CPU is disrupted by an interrupt
  - Finish executing instruction
  - { interrupt }
  - store registers, PC (size M) control stack @ T
  - update stack pointer to T-M
  - execute interrupt instruction @ Y until finishing @ Y+L
  - load back top of control stack
  - continue executing @ N+1



- What happens when CPU is disrupted by an interrupt
  - Finish executing instruction
  - { interrupt }
  - store registers, PC (size M) control stack @  $T$
  - update stack pointer to  $T-M$
  - execute interrupt instructions @  $Y$  until finishing @  $Y+L$
  - load back top of control stack
  - continue executing @  $N+1$



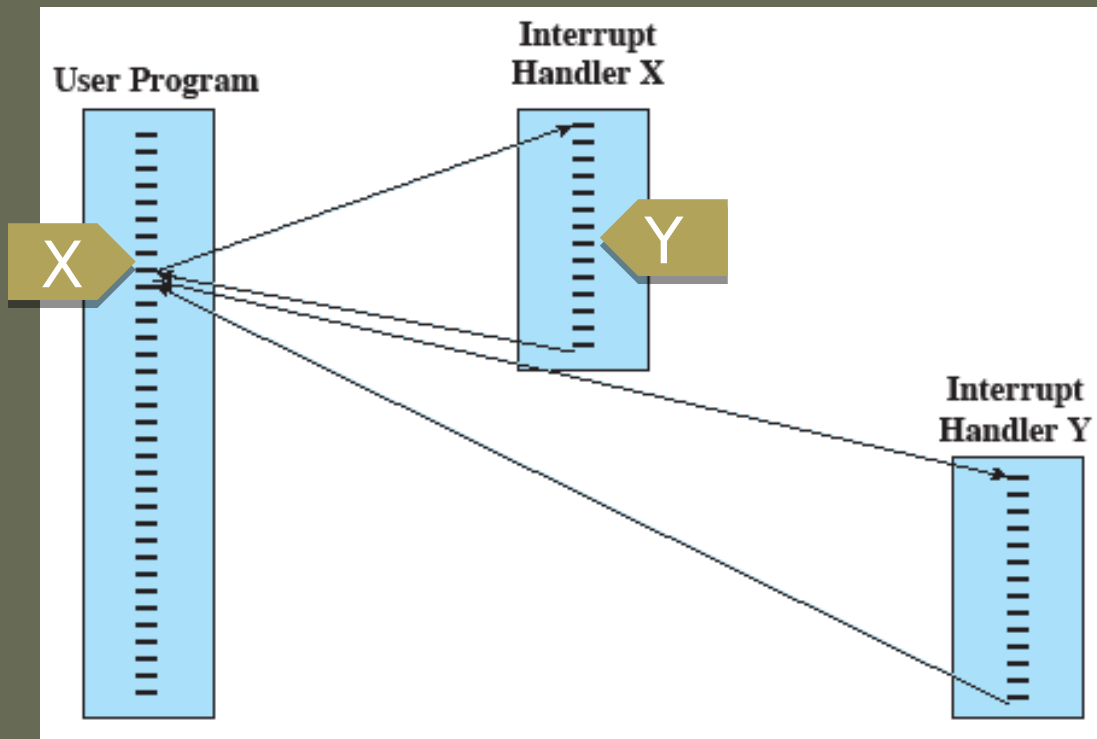
# Interrupts

---

- Multiple overlapping interrupts
  - An interrupt happens when another is being handled
  - 1. **Disable interrupts** (when handling an interrupt)
    - ▢ 2<sup>nd</sup> interrupt waits until 1<sup>st</sup> interrupt is handled
    - ▢ Strictly sequential
  - 2. Use a **priority scheme**
    - ▢ Interrupts can interrupt interrupt-handling...
    - ▢ ...but only if they have a higher priority; otherwise they wait
    - ▢ Hierarchical (by priority)

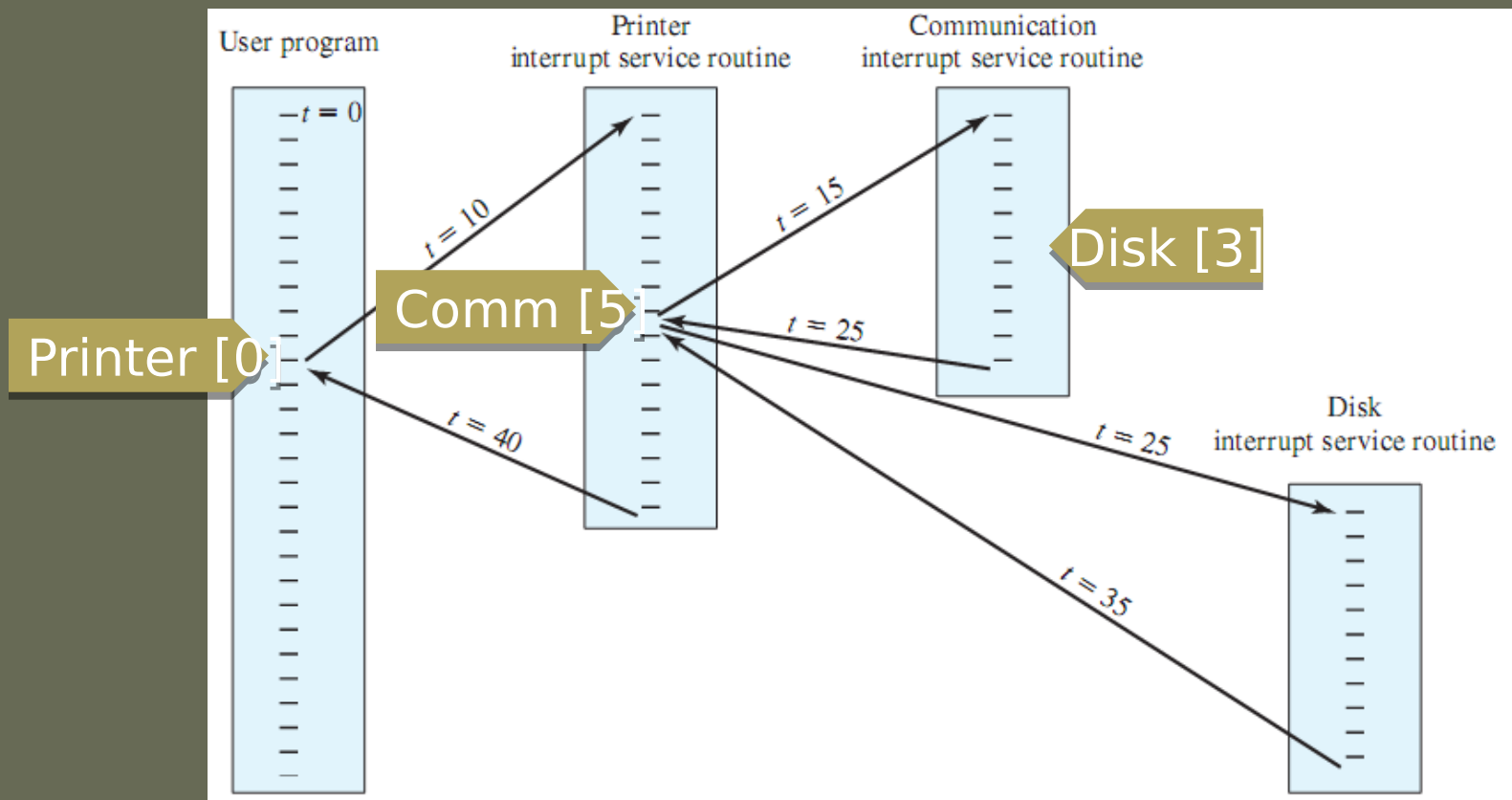
# Interrupts

- Multiple overlapping interrupts
  - (approach 1) **Interrupts disabled**



# Interrupts

- Multiple overlapping interrupts
  - (approach 2) **Priority Scheme**





# Topics

---

- Basic Elements
  - Processor, main memory, I/O modules, system bus
- Microprocessors
  - General purpose, graphics, digital signal, system on chip
- Instructions
  - Execution, fetch & execute (F&E), instruction register
- Interrupts
  - Types, flow of control, F&E&I, multiple interrupts
- Memory
  - Hierarchy, principle of locality, cache
- I/O Techniques
  - Programmed, interrupt-driven, direct memory access
- Symmetric multi-processors
  - Advantages, organization, multi-core

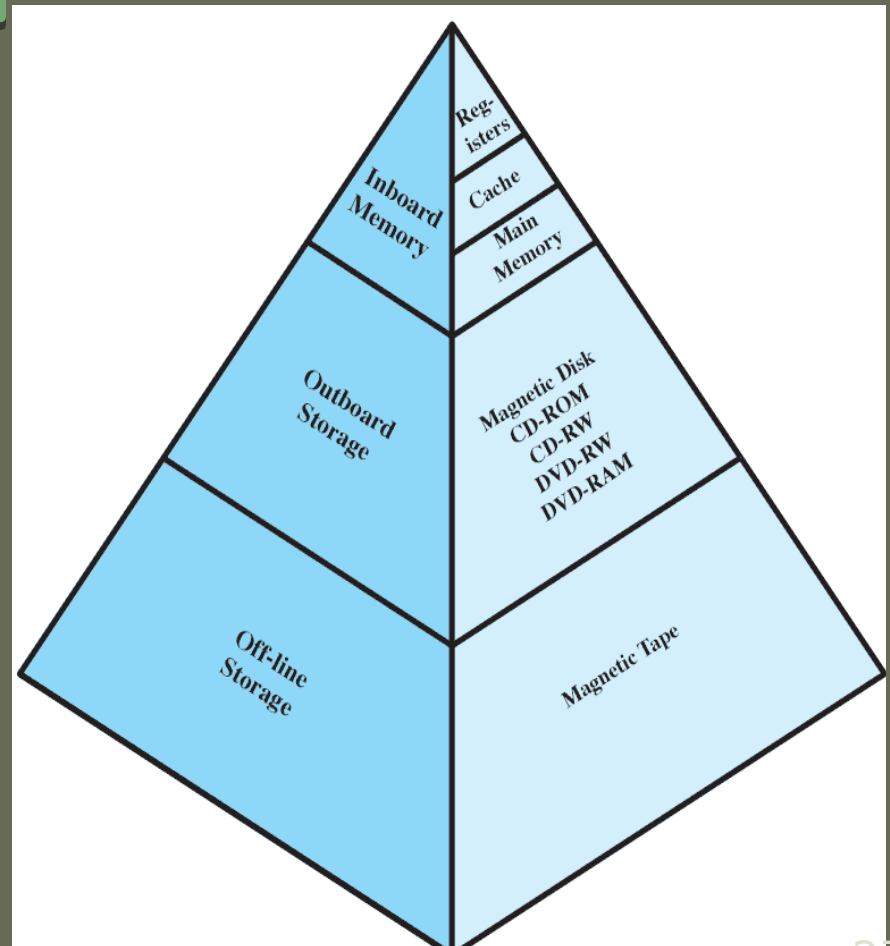
# Memory

---

- Major (conflicting) constraints
  - **speed** (access time), **amount**, **cost**
  - Memory must keep up with CPU (speed)
    - ▮ Faster access time = greater cost
  - Memory must satisfy data volumes (amount)
    - ▮ Greater capacity = smaller cost = slower access speed

# Memory

- Hierarchy going down
  - cost decreases
  - capacity increases
  - access time increases
  - frequency of access by CPU decreases
    - really? how does it happen?



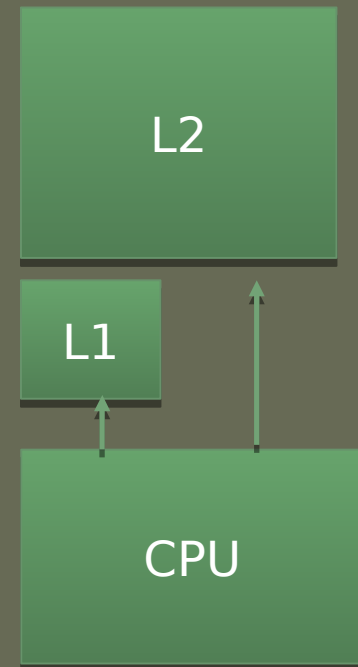
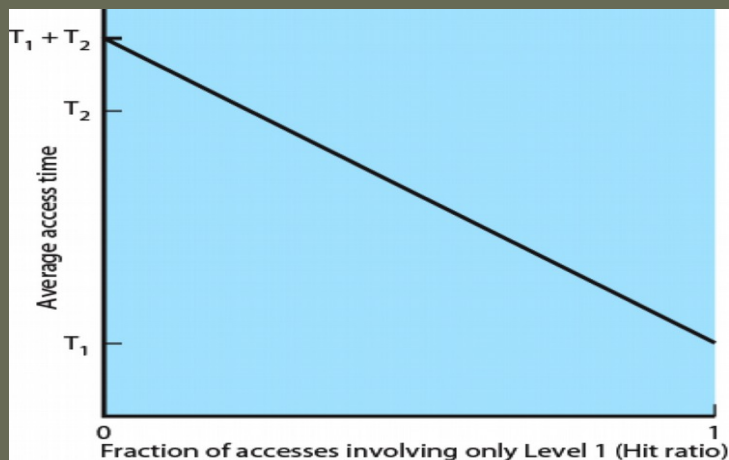
# Memory

---

- Principle of Locality
  - Why does frequency of access by CPU decrease?
    - ▢ Memory references (i.e., data) needed by CPU (i.e., the current set of instructions in a program) tend to cluster
      - ▢ e.g. array “a” being read in a loop
  - Eventually a set of data is replaced by another, but it's less frequent proportionally to the use within a set, which makes overhead bearable.

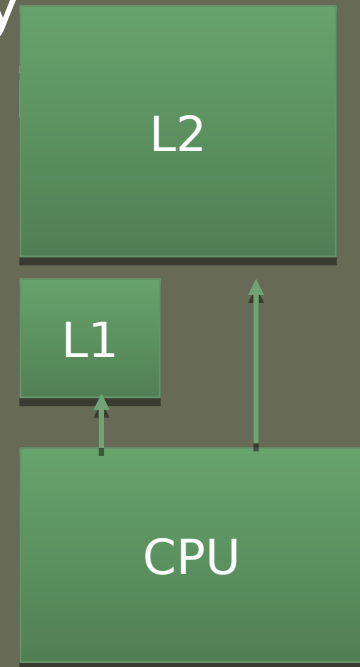
# Memory

- Performance Example
  - 2 levels of memory (L1,L2)
    - ▮ T1 @  $0.1\mu\text{s}$  (1kb total ) faster but scarce
    - ▮ T2 @  $1\mu\text{s}$  (100kb total ) slower but plenty
  - CPU checks L1 first then L2



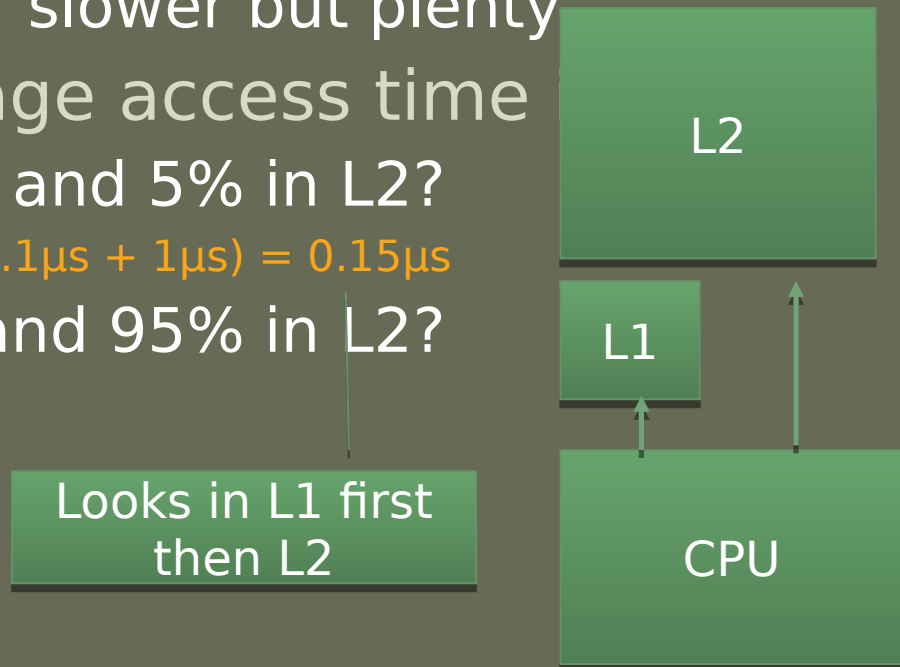
# Memory

- Performance Example
  - 2 levels of memory (L1,L2)
    - ▮ T1 @  $0.1\mu\text{s}$  (1kb) faster but scarce
    - ▮ T2 @  $1\mu\text{s}$  (100kb) slower but plenty
  - What's the average access time
    - ▮ 95% of data in L1 and 5% in L2?



# Memory

- Performance Example
  - 2 levels of memory (L1,L2)
    - ▮ T1 @  $0.1\mu\text{s}$  (1kb) faster but scarce
    - ▮ T2 @  $1\mu\text{s}$  (100kb) slower but plenty
  - What's the average access time
    - ▮ 95% of data in L1 and 5% in L2?
      - ▮  $0.95 * 0.1\mu\text{s} + 0.05 * (0.1\mu\text{s} + 1\mu\text{s}) = 0.15\mu\text{s}$
    - ▮ 5% of data in L1 and 95% in L2?



# Memory

- Performance Example

- 2 levels of memory (L1, **What if you have 3 levels?**)

- ▮ T1 @  $0.1\mu\text{s}$  (1kb) faster but scarce

- ▮ T2 @  $1\mu\text{s}$  (100kb) slower but plenty

- What's the average access time if...

- ▮ 95% of data in L1 and 5% in L2?

- ▮  $0.95 * 0.1\mu\text{s} + 0.05 * (0.1\mu\text{s} + 1\mu\text{s}) = 0.15\mu\text{s}$

- ▮ 5% of data in L1 and 95% in L2?

- ▮  $0.05 * 0.1\mu\text{s} + 0.95 * (0.1\mu\text{s} + 1\mu\text{s}) = 1.05\mu\text{s}$

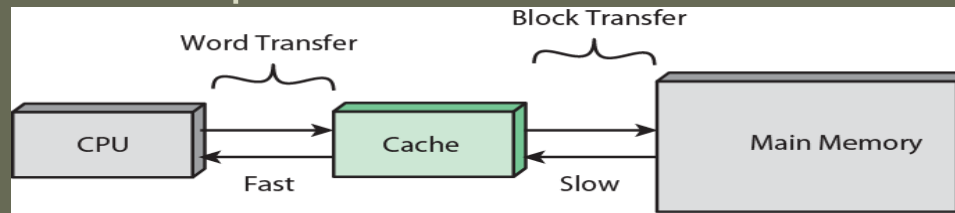
- ▮ It's to our advantage to have frequently accessed data in faster memory locations

**reason why caches exist**



# Memory

- Cache
  - Exploits the Principle of Locality
  - Controlled by hardware (i.e., it is invisible to OS)
  - Principles
    - ▮ Contains a copy of a portion of main memory
    - ▮ CPU checks cache for data
      - ▮ **if found**: use data
      - ▮ **if not found**: reads block of data from memory (where data is) and copies it into cache



# Memory

- Cache

- Explains

- Contention (OS)

- Principle

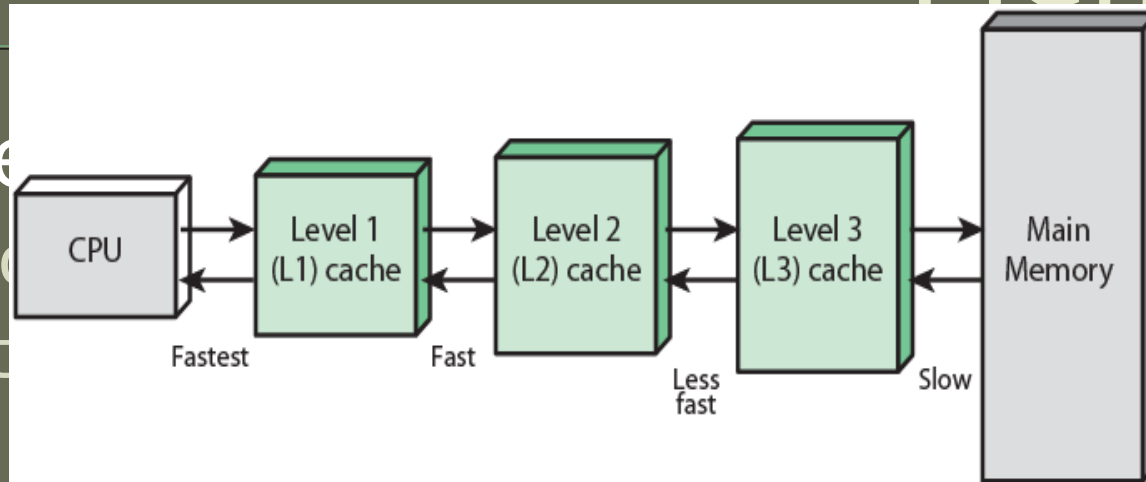
- Contention

- CPU of several levels are common

- if four

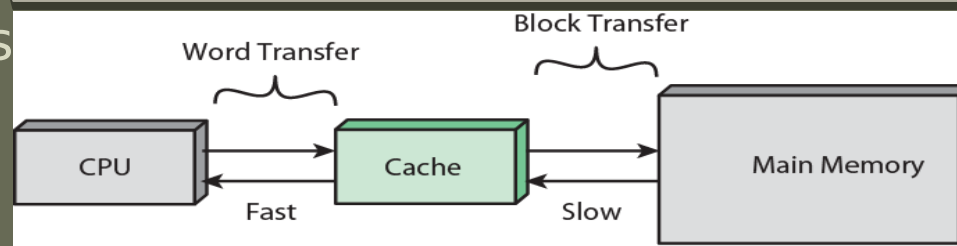
- if not four

data is



In practice:

several levels are common



# Topics

---

- Basic Elements
  - Processor, main memory, I/O modules, system bus
- Microprocessors
  - General purpose, graphics, digital signal, system on chip
- Instructions
  - Execution, fetch & execute (F&E), instruction register
- Interrupts
  - Types, flow of control, F&E&I, multiple interrupts
- Memory
  - Hierarchy, principle of locality, cache
- I/O Techniques
  - Programmed, interrupt-driven, direct memory access
- Symmetric multi-processors
  - Advantages, organization, multi-core

# I/O Techniques

---

- \* When the processor encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module

Three techniques are possible for I/O operations:

Programm  
ed I/O

Interrupt-  
Driven I/O

Direct  
Memory  
Access  
(DMA)

# I/O Techniques

What does the CPU do during an I/O (read/write) instruction?

- 1) Programmed I/O
  - CPU waits for completion of command and periodically checks the status of the I/O module until it determines the instruction is complete
    - ▮ 1. CPU sends I/O command to I/O module
    - ▮ *If writing, CPU transfers data.*
    - ▮ 2. CPU waits until I/O module completes command.
    - ▮ *If reading, CPU transfers data*
    - ▮ 3. CPU resumes execution
  - Extreme Inefficiency!

Remember the 1 sec verses  
16 weeks example?

# I/O Techniques

What does the CPU do during an I/O (read/write) instruction?

- 2) Interrupt-driven I/O
  - CPU keeps executing while I/O command is completed
    - ▢ 1. CPU sends I/O command to I/O module
      - *If writing, CPU transfers data.*
    - ▢ 2. CPU resumes execution.
    - ▢ 3. I/O module triggers an interrupt when command is done
      - *If reading, CPU transfers data*
    - ▢ 4. CPU resumes execution
  - No wait, but CPU still heavily involved in data transfer

# I/O Techniques

---

- 2) Interrupt-driven I/O Drawbacks
  - Transfer rate is limited by the speed with which the processor can service a device
  - The processor is tied up in managing an I/O transfer since a number of instructions must be executed for each I/O transfer

# I/O Techniques

---

What does the CPU do during an I/O (read/write) instruction?

- 3) Direct Memory Address (DMA)
  - Performed by a separate module on the system bus (DMA)
  - CPU keeps executing while I/O command is completed by DMA module
    - ▢ 1. CPU sends I/O command, memory address (where data is read or written), data size and I/O module to DMA
    - ▢ 2. CPU resumes execution.
    - ▢ 3. I/O module triggers an interrupt when command is done (including data transfer)
    - ▢ 4. CPU resumes execution
  - No wait & CPU doesn't transfer data (may have bus contention though)



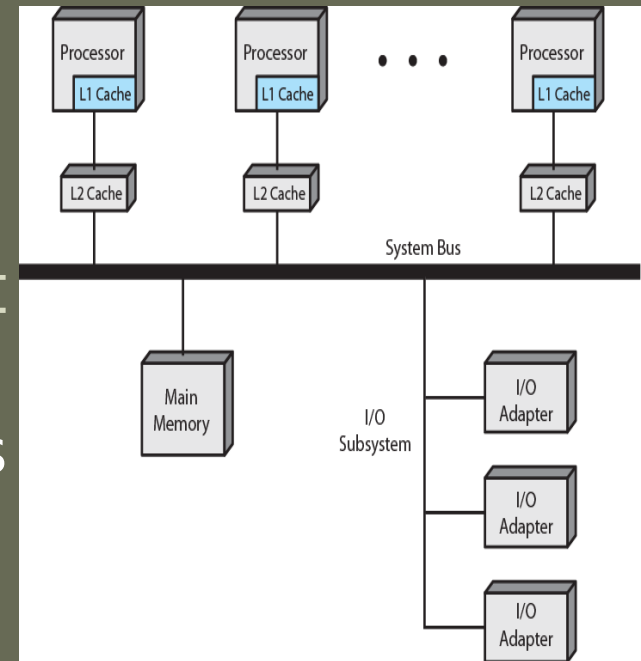
# Chapter 1 Topics

---

- Basic Elements
  - Processor, main memory, I/O modules, system bus
- Microprocessors
  - General purpose, graphics, digital signal, system on chip
- Instructions
  - Execution, fetch & execute (F&E), instruction register
- Interrupts
  - Types, flow of control, F&E&I, multiple interrupts
- Memory
  - Hierarchy, principle of locality, cache
- I/O Techniques
  - Programmed, interrupt-driven, direct memory access
- Symmetric multi-processors
  - Advantages, organization, multi-core

# Symmetric Multi-Processors

- A stand-alone computer system with:
  - 2+ similar processors:
    - ▢ capable of performing the same functions
  - which (physically):
    - ▢ are interconnected by a bus
    - ▢ share memory & I/O devices
  - are controlled by an OS that
    - ▢ provides interaction between processors and their programs (at the job, task, file, and data levels)



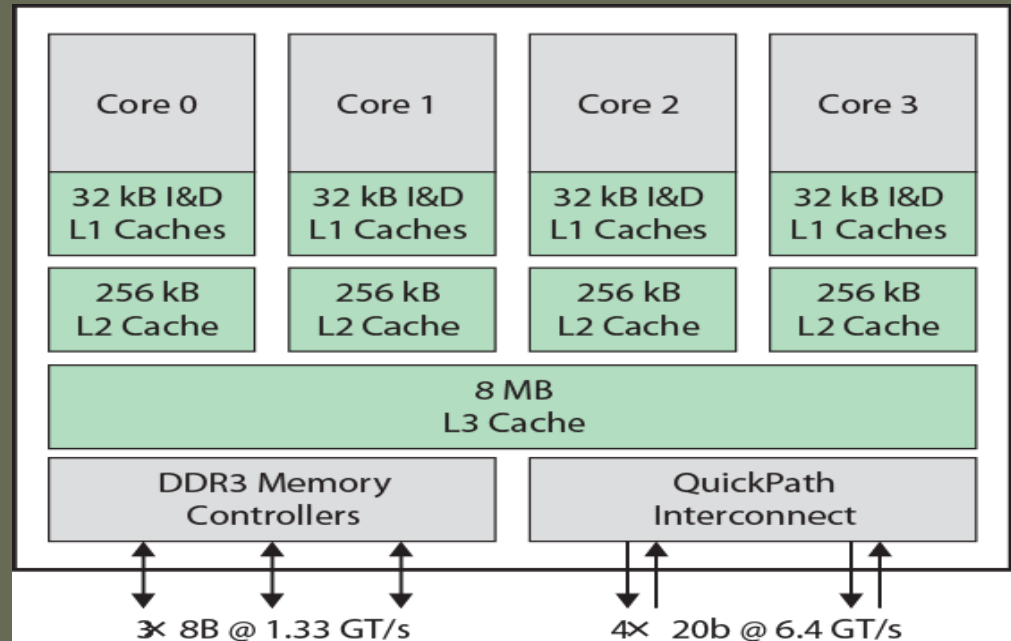
# Symmetric Multi-Processors

---

- Advantages
  - Performance
    - ▮ can yield greater performance (if OS can handle work in parallel)
  - Availability
    - ▮ failure of one processor does not halt the machine
  - Scaling
    - ▮ additional processors result in a range of products of different price and performance

# Symmetric Multi-Processors

- Multi-Core
  - 2+ processors (cores) in 1 micro-chip
    - ▢ each core has all components of an independent processor (including 2 or 3 cache levels)
  - Intel Core i7
    - ▢ 4-8 cores
    - ▢ 8 Mb L3 cache



# Topics

---

- **Basic Elements**
  - Processor, main memory, I/O modules, system bus
- **Microprocessors**
  - General purpose, graphics, digital signal processor, system on chip
- **Instructions**
  - Execution, fetch & execute cycle, instruction register
- **Interrupts**
  - Types, flow of control, interrupt enable & disable, multiple interrupts
- **Memory**
  - Hierarchy, principle of locality, cache
- **I/O Techniques**
  - Programmed, interrupt-driven, direct memory access
- **Symmetric multi-processors**
  - Advantages, organization, multi-core

Done!