Introduction to Cloud Computing

Systems I

15-319, spring 2010

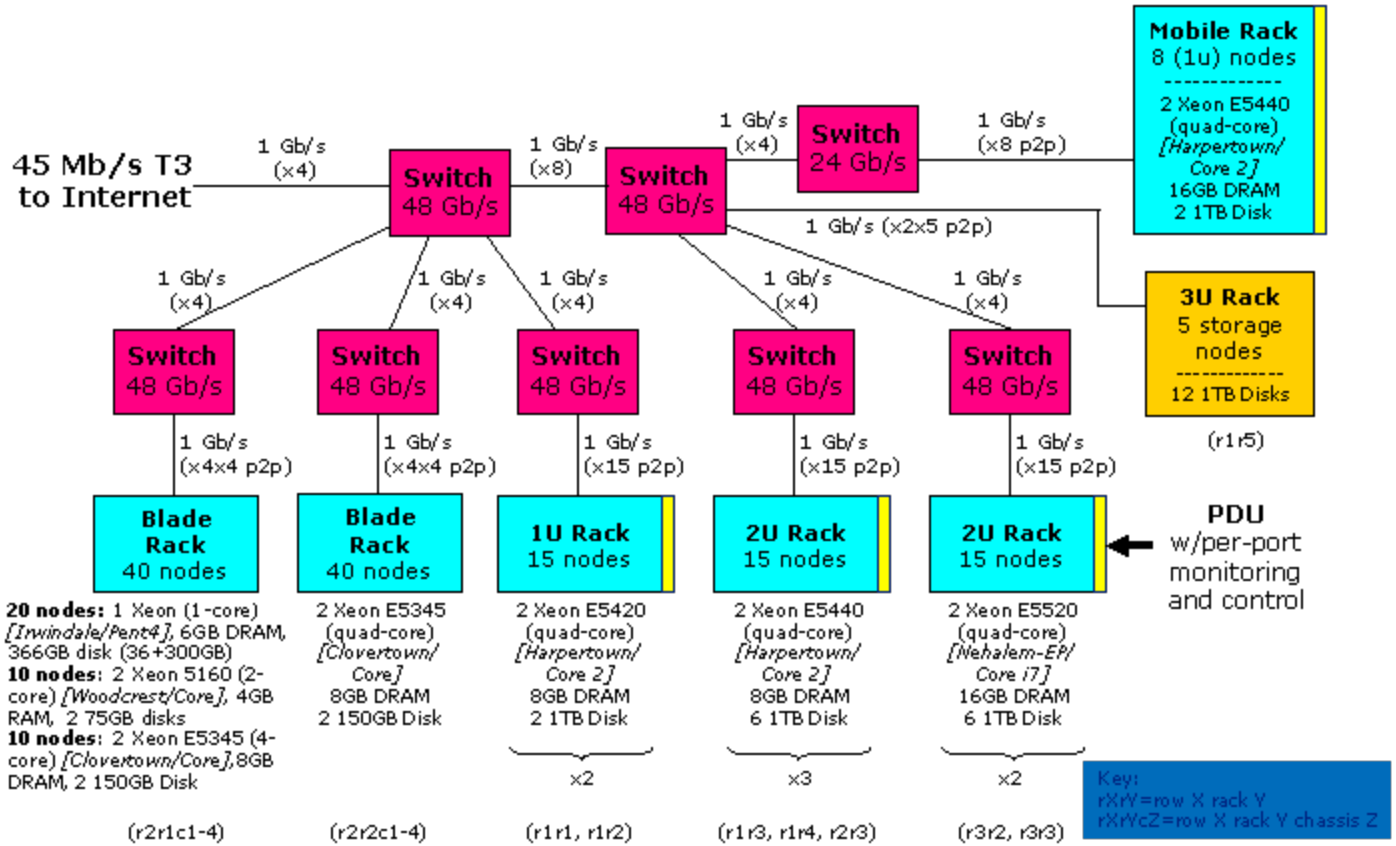
4th Lecture, Jan 26th

Majd F. Sakr

Lecture Motivation

- **Overview of a Cloud architecture**
- **Systems review**
  - ISA
  - Memory hierarchy
  - OS
  - Networks
- **Next time**
  - Impact on cloud performance

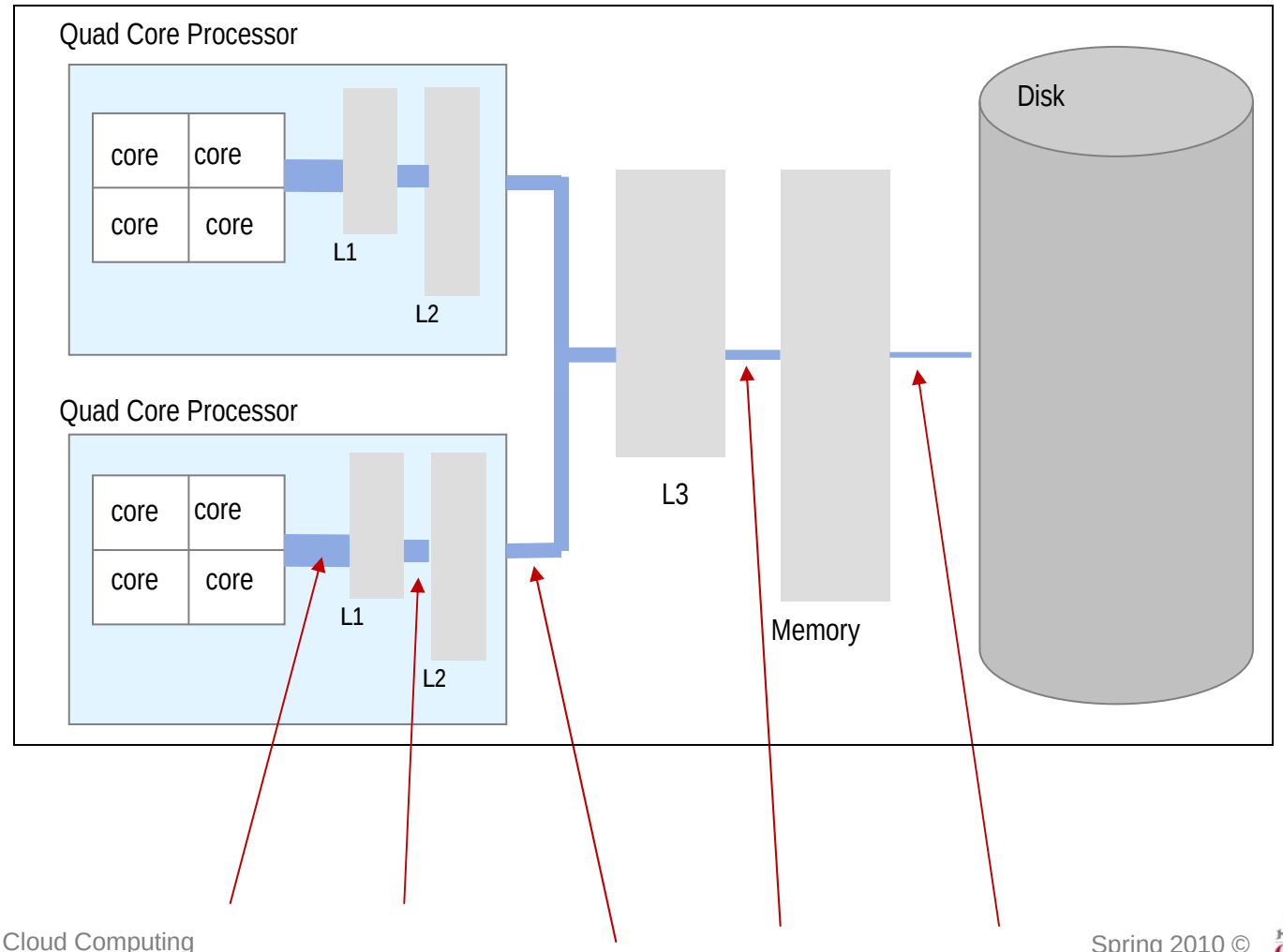
Intel Open Cirrus Cloud - Pittsburgh



	(r2r1c1-4)	(r2r2c1-4)	(r1r1, r1r2)	(r1r3, r1r4, r2r3)	(r3r2, r3r3)			
	r2r1c1-4	r2r2c1-4	r1r1 r1r2	r1r3 r1r4 r2r3	r3r2 r3r3	mobile	storage	TOTAL
Nodes	40	40	30	45	30	8	5	**198**
Cores	140	320	240	360	240	64		**1364**
DRAM (GB)	240	320	240	360	480	128		**1768**
Spindles	80	80	60	270	180	16	60	**746**
Storage (TB)	12	12	60	270	180	16	60	**610**

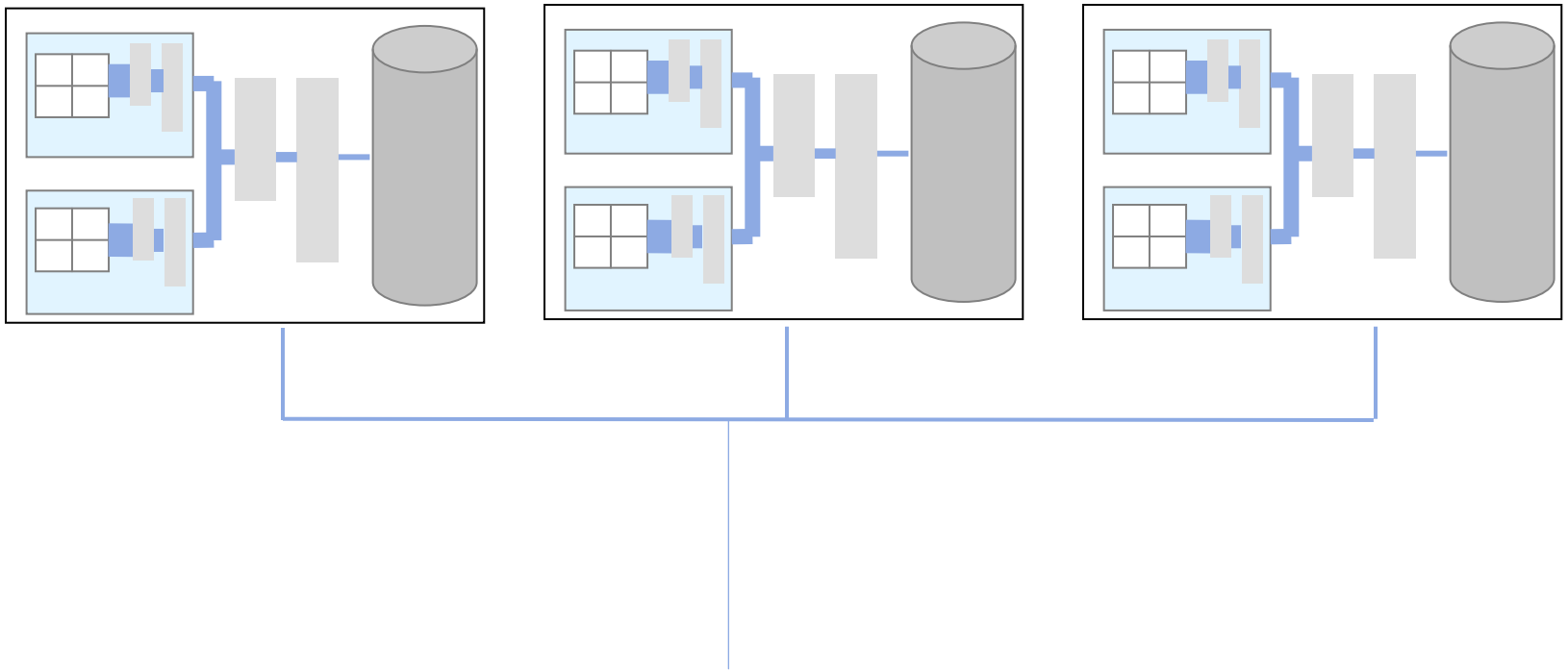
Blade Performance

- Consider bandwidth and latency between these layers



Cloud Performance

- Consider bandwidth and latency of all layers



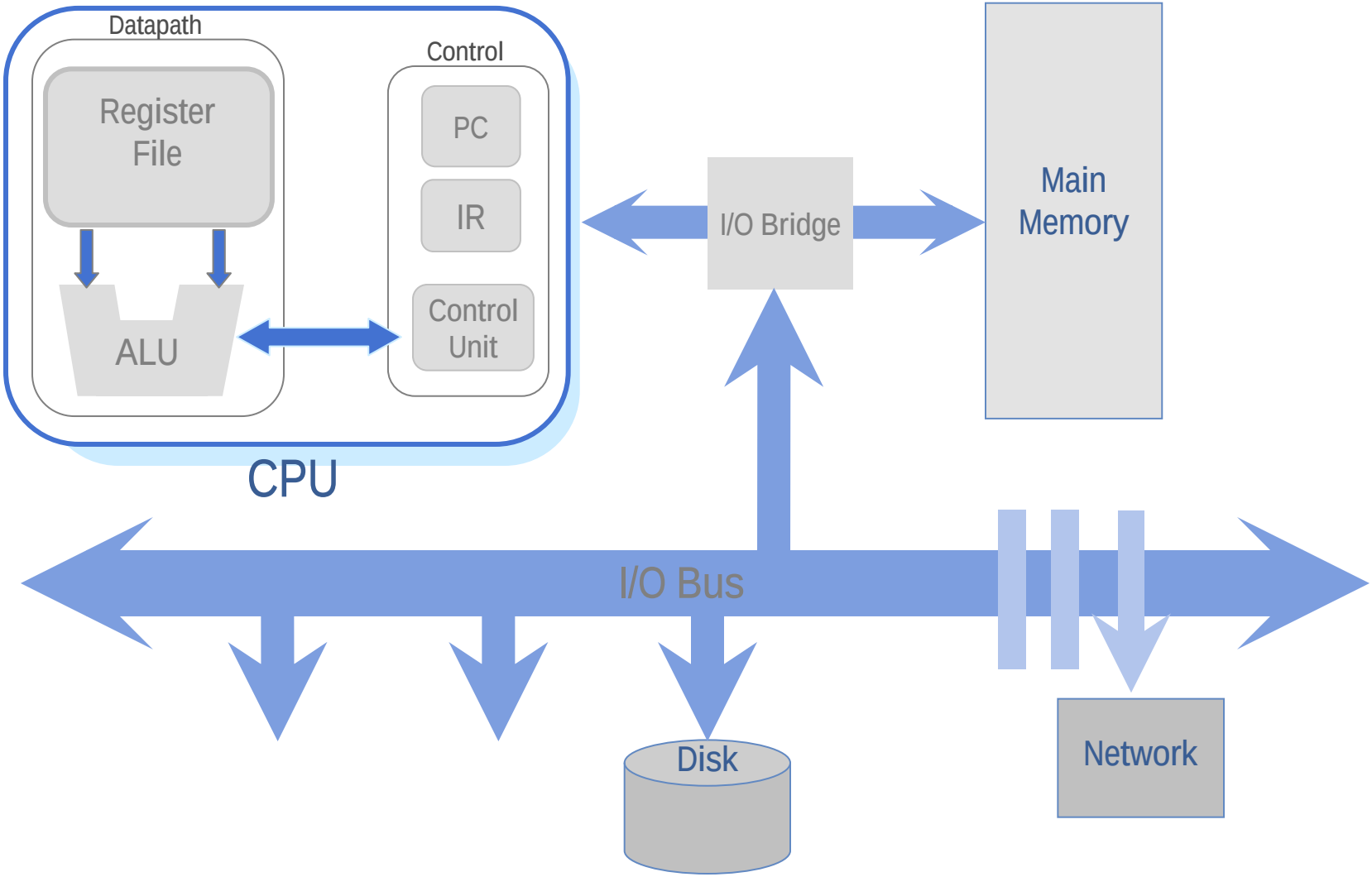
How could these different bandwidths impact the Cloud's performance?

To understand this, we need to go through a summary of the machine's components?

Single CPU Machine's Component

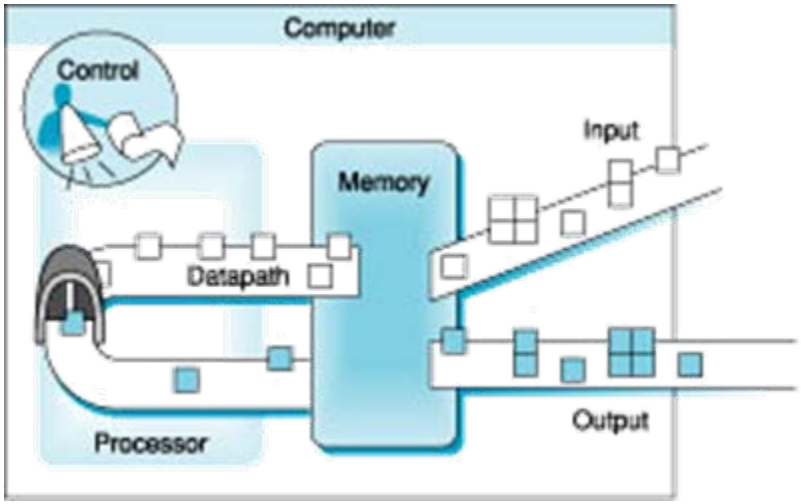
- ISA (Processor)
- Main Memory
- Disk
- Operating System
- Network

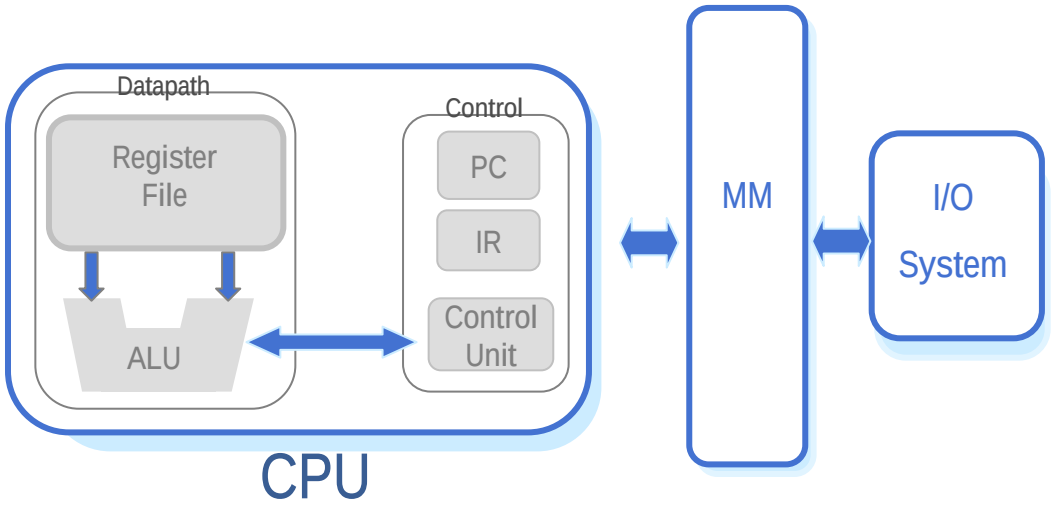
A Single-CPU Computer Components



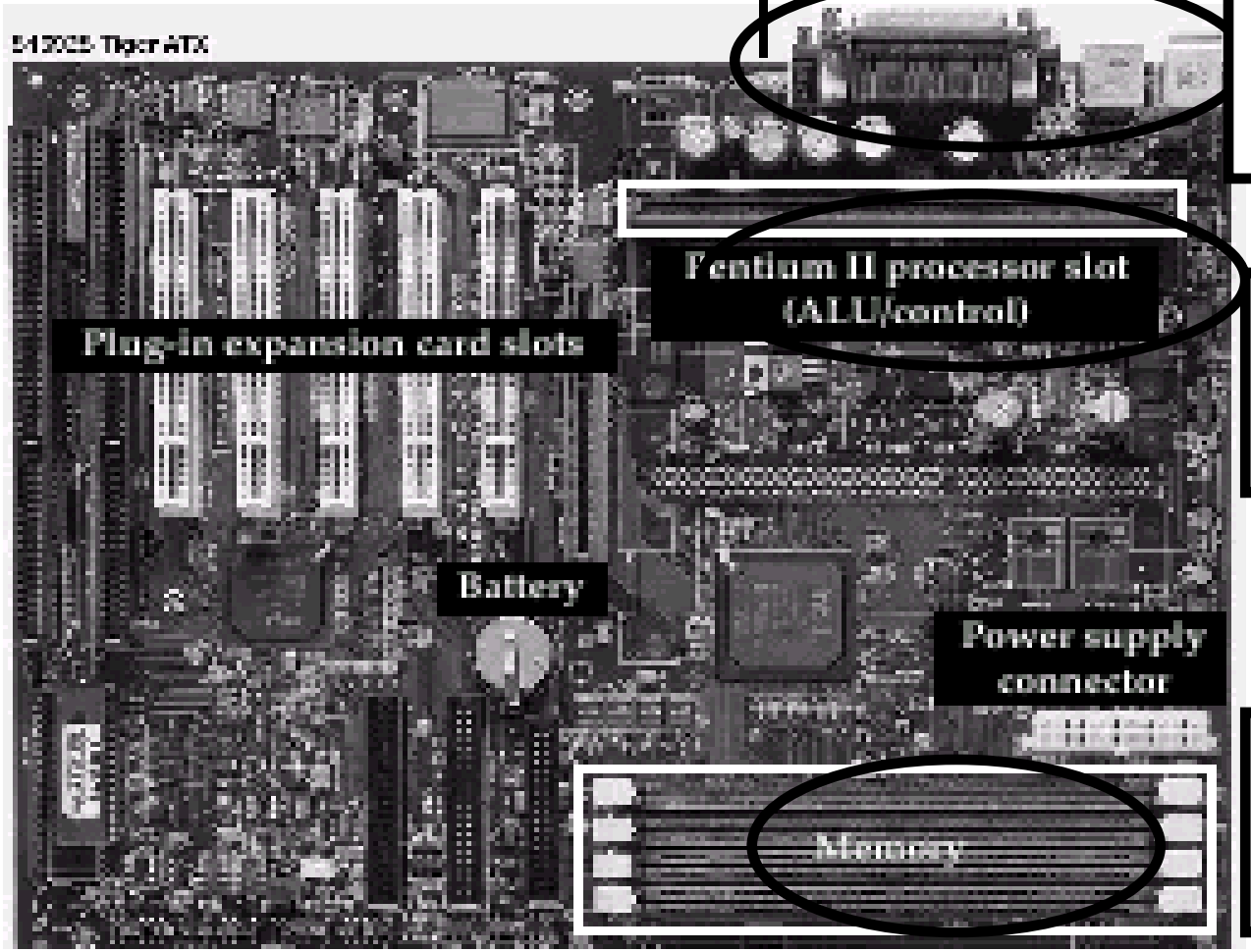
The Von Neuman machine - Completed 1952

- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and Output equipment operated by control unit





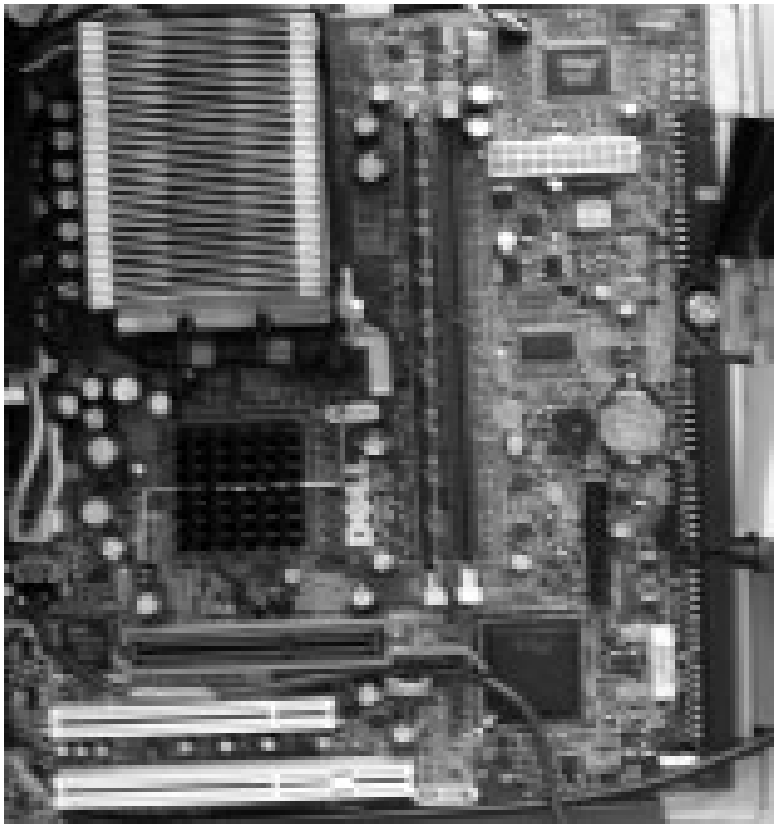
The Five Von Neumann Components

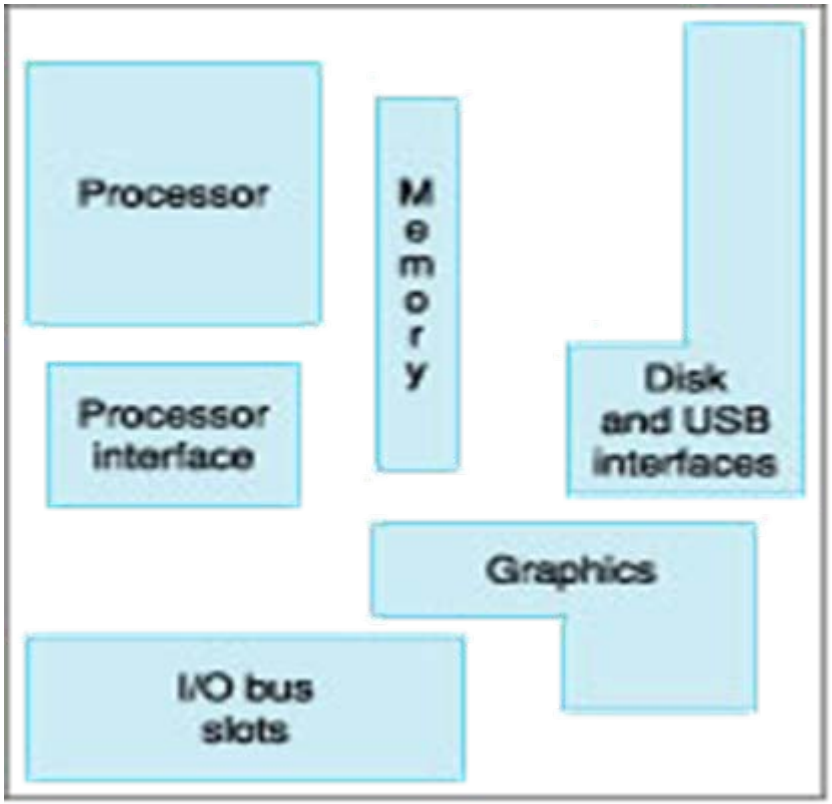


Motherboard

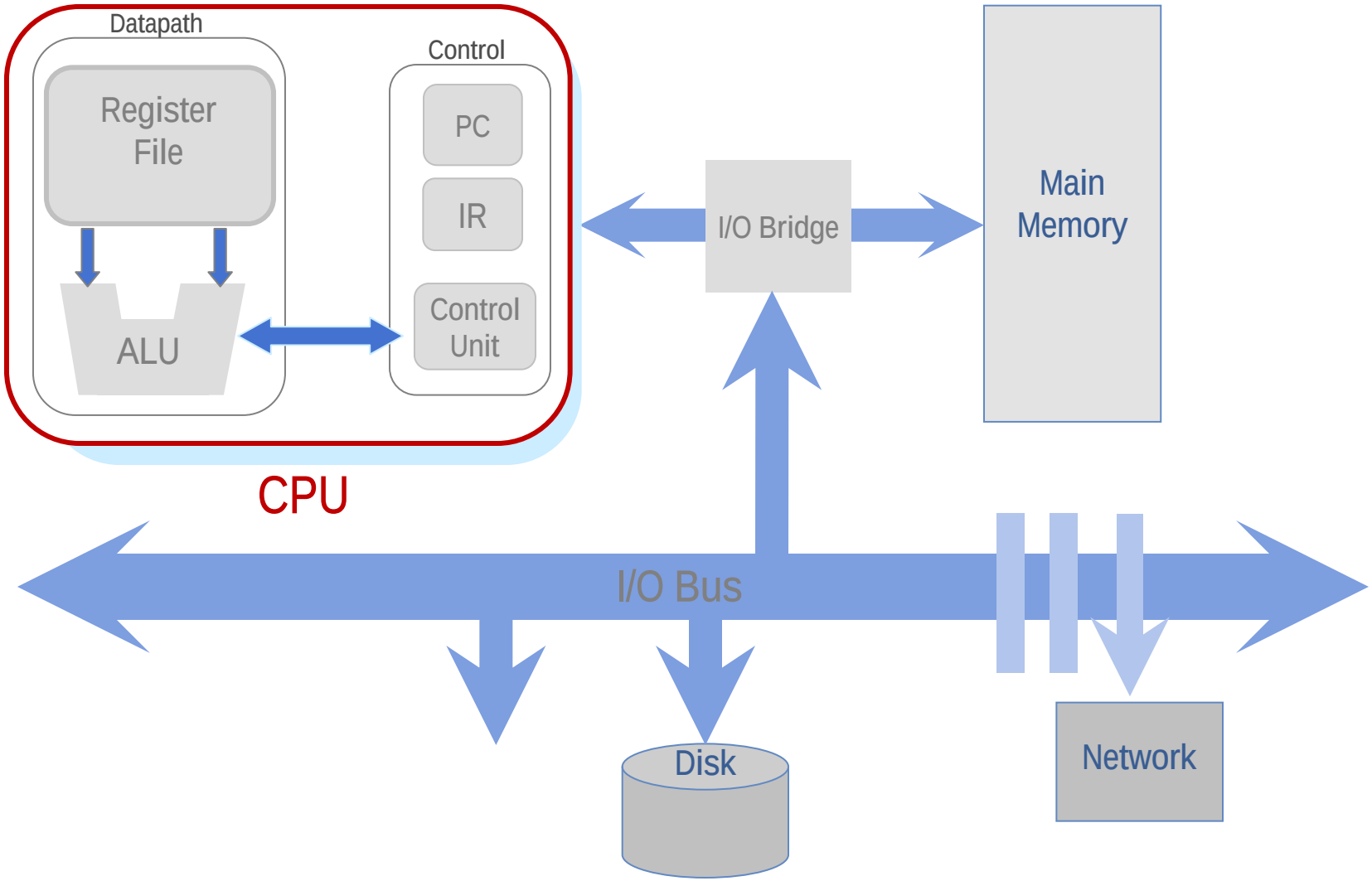
- **Everything is attached to it:**

  CPU/ memory/ monitor/ mouse/ keyboard/ add-on cards/ printers/ scanners/ speakers/ ..etc



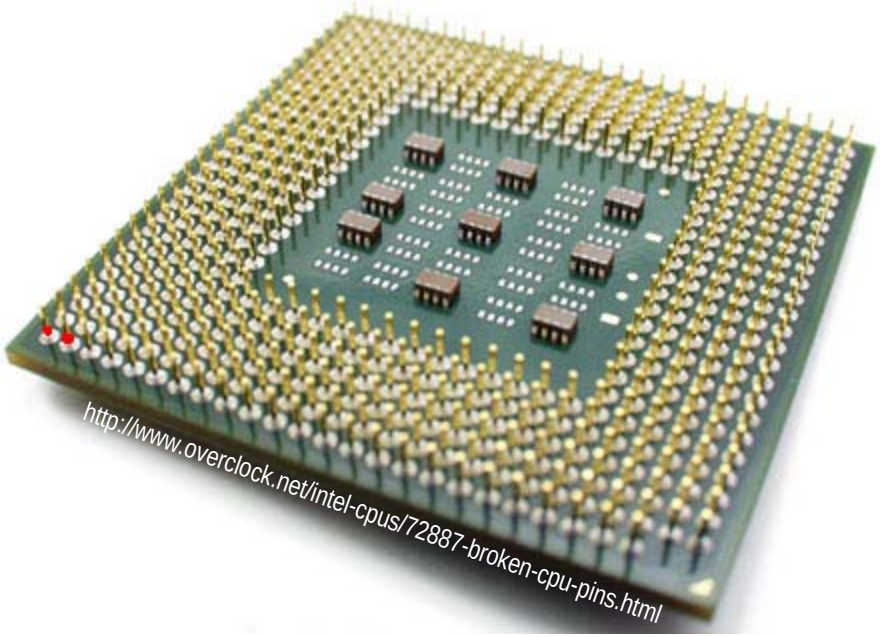


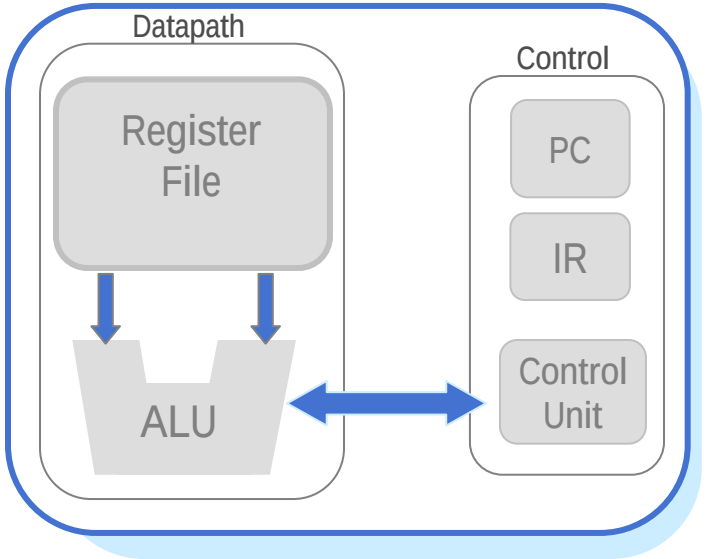
The Processor



Central Processing Unit

- **The Brain:** a functional unit that interprets and carries out instructions (does mathematical manipulation)
- Also called a **Processor**
- Contains tens of millions of tiny transistors





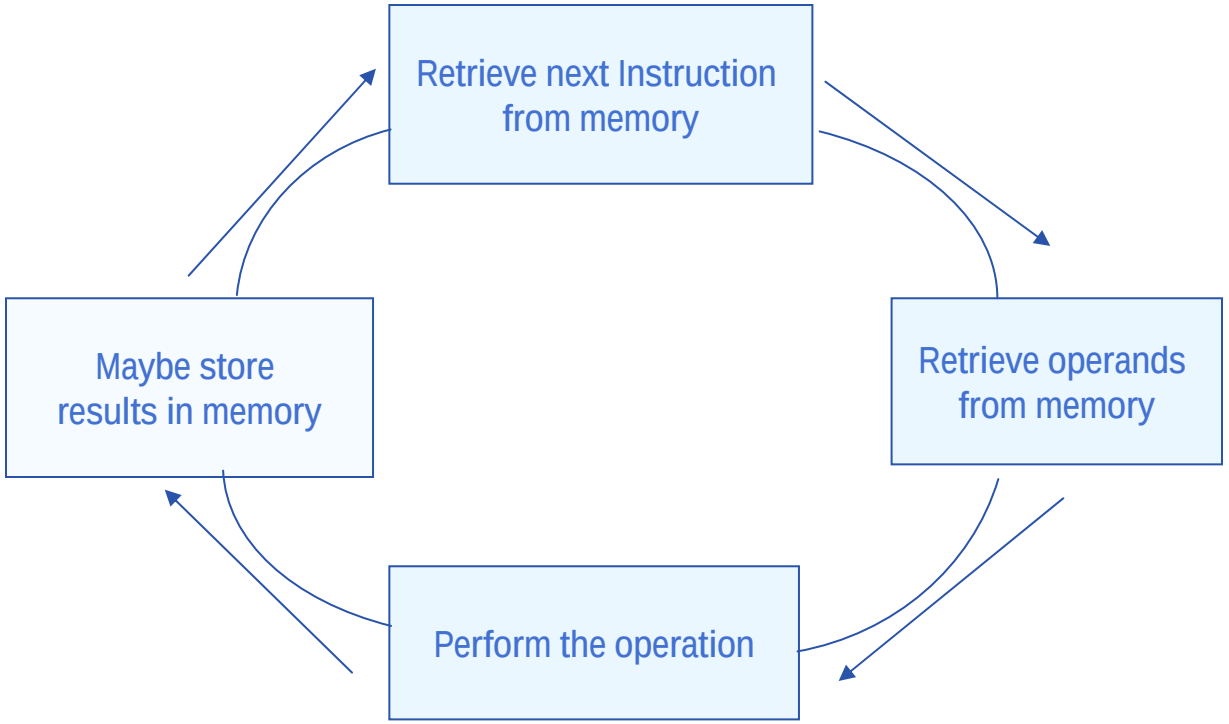
Instruction Set Architecture (ISA)

- Every processor has a unique set of operations (**Instruction Set or Machine Instructions**)
  - Written in binary language called **Machine Language**
  - Each Instruction consists of (operator & operands)
  - Operand: data or a memory address of the data that we need to operate on
  - Instructions Categories:

Category	Example
Arithmetic	Add, subtract, multiply, divide
Logic	And, or, not, exclusive or
Program Control	Branching, subroutines
Data Movement	Move, load, store
I/O	Read, write

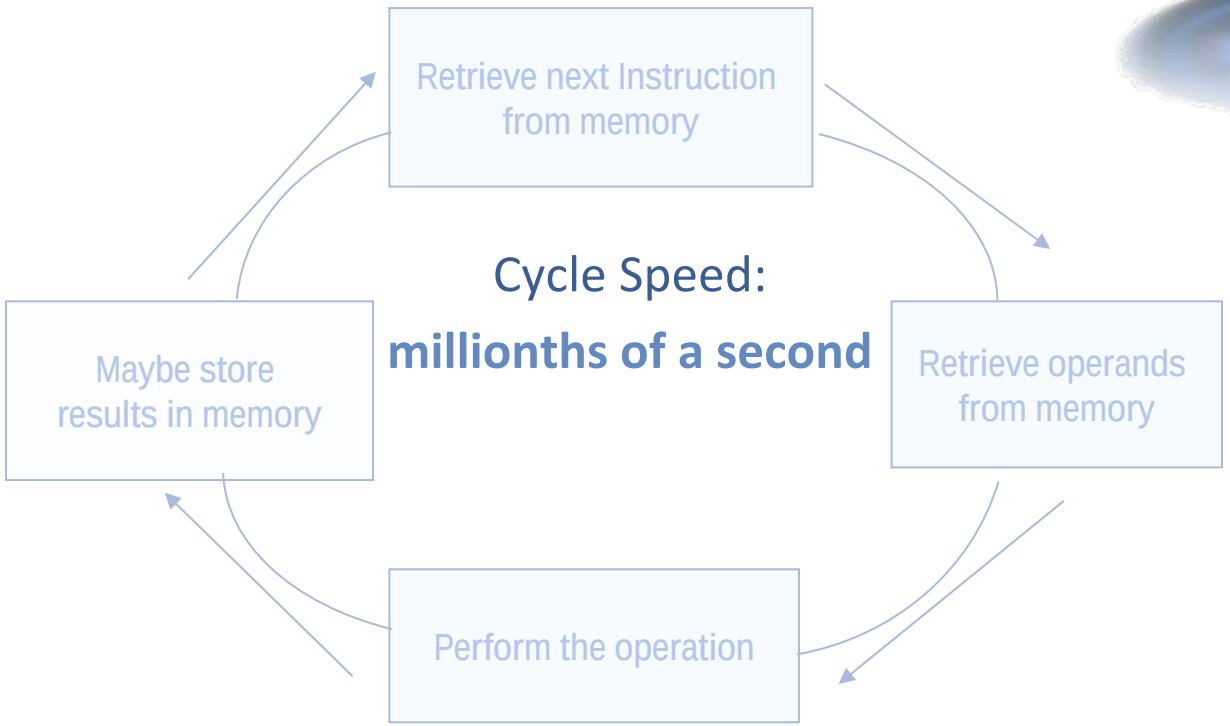
Program Execution

- **Fetch/Execute Cycle:**
  - Fetch Cycle: get instruction from memory
  - Execute Cycle: Decode instruction, get operands, execute the operation on them

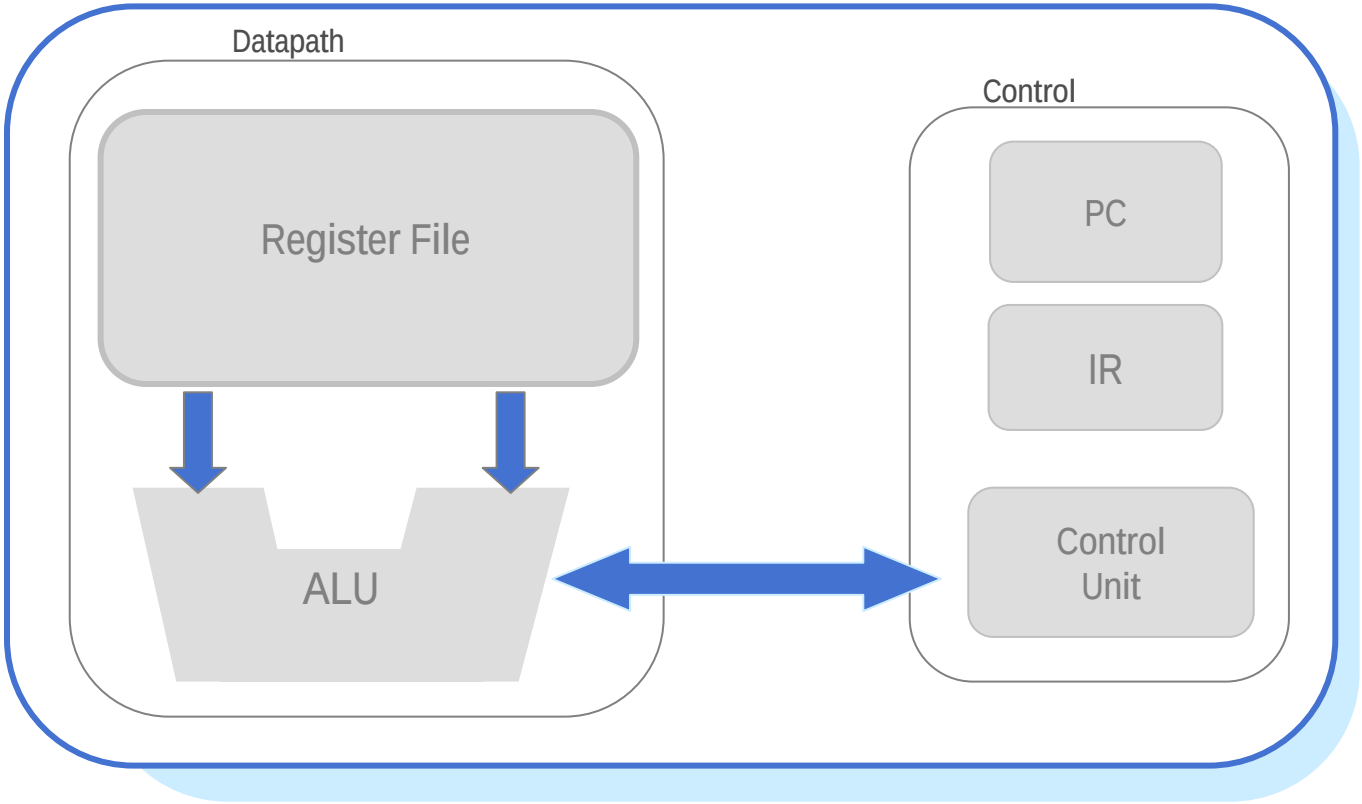


Synchronous Systems

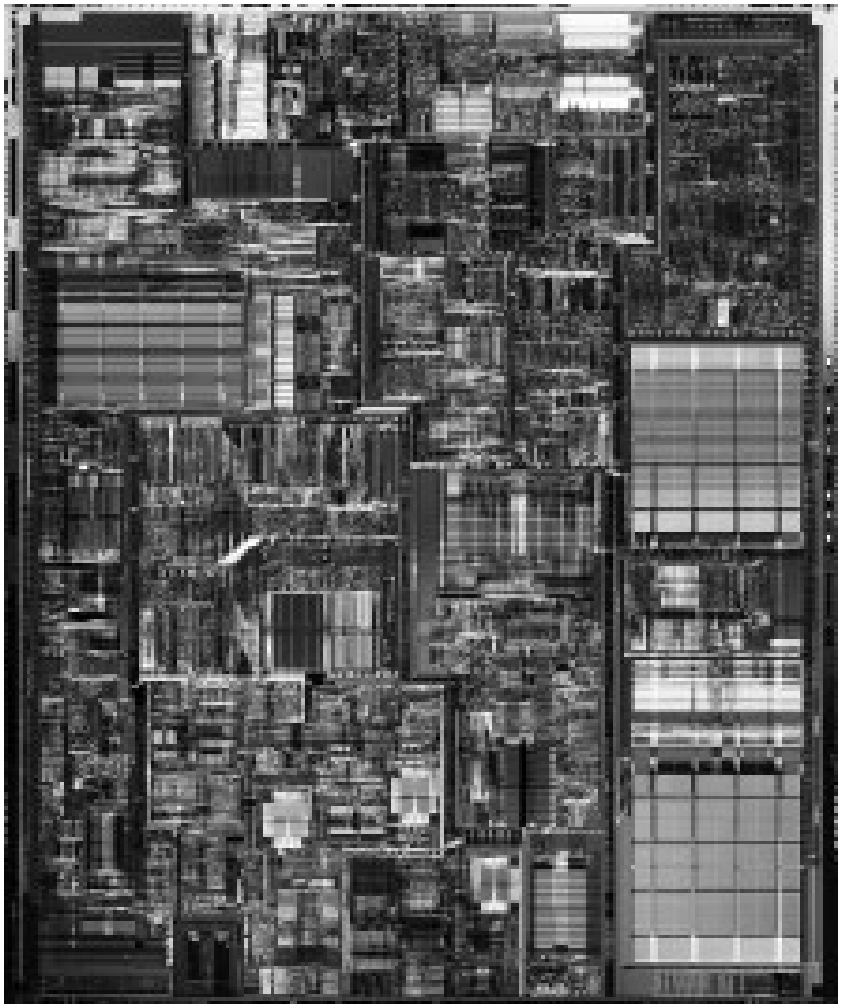
- Each step waits for clock ticks to begin
- A **100 MHz** processor uses a clock that **ticks 100,000,000 times per second**

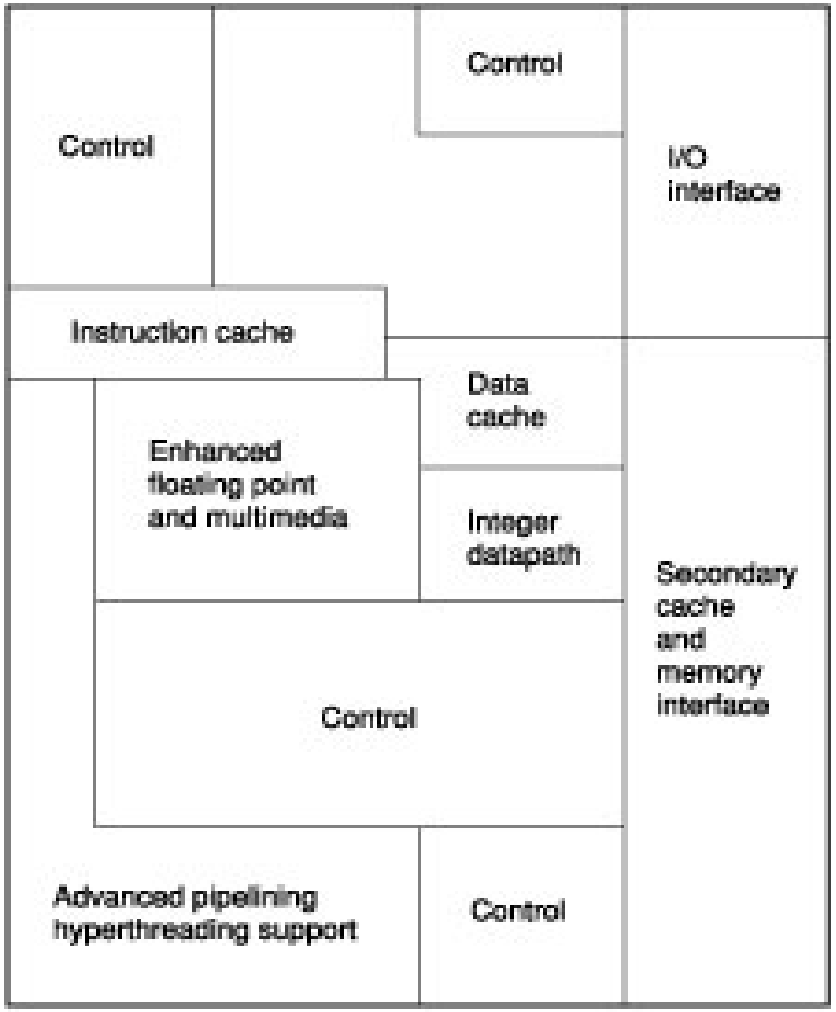


Processor Components (1/3)



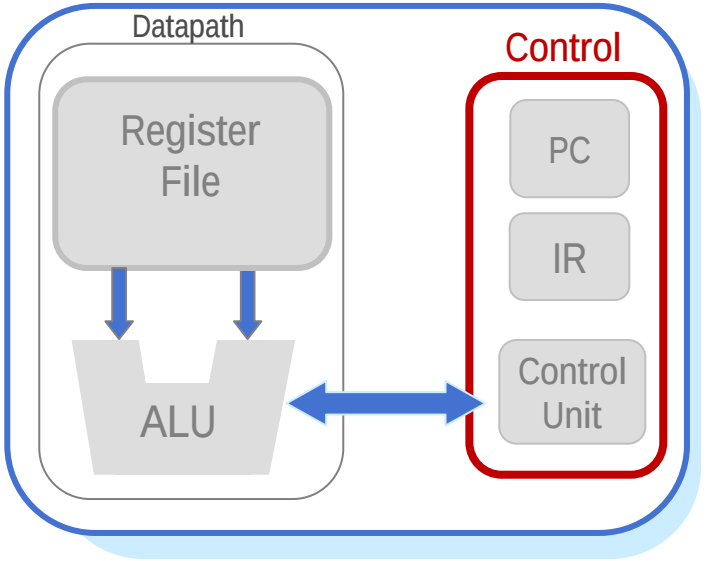
Processor Layout





Processor Components (2/3)

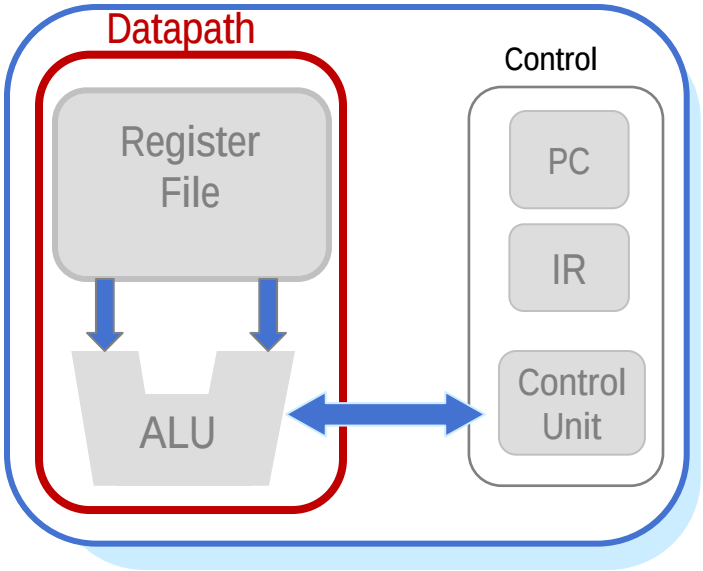
- **Control Unit**
  - Processor's Supervisor
  - Connects and regulates the interaction between the datapath and memory.
  - **Job Steps:**
    - Retrieve instruction from Memory
    - Decode it
    - Break it into a sequence of actions
    - Carries out the execution of an instruction



- **Program Counter (PC):** has the address of the instruction to be fetched
- **Instruction Register (IR):** has the instruction most recently fetched

Processor Components (3/3)

- **Datapath**
  - **Register Files**: General purpose storage locations inside the processor that hold values or addresses
  - **Arithmetic Logic Units:** Set of functional units or multipliers that perform data processing (arithmetic & logical operations)

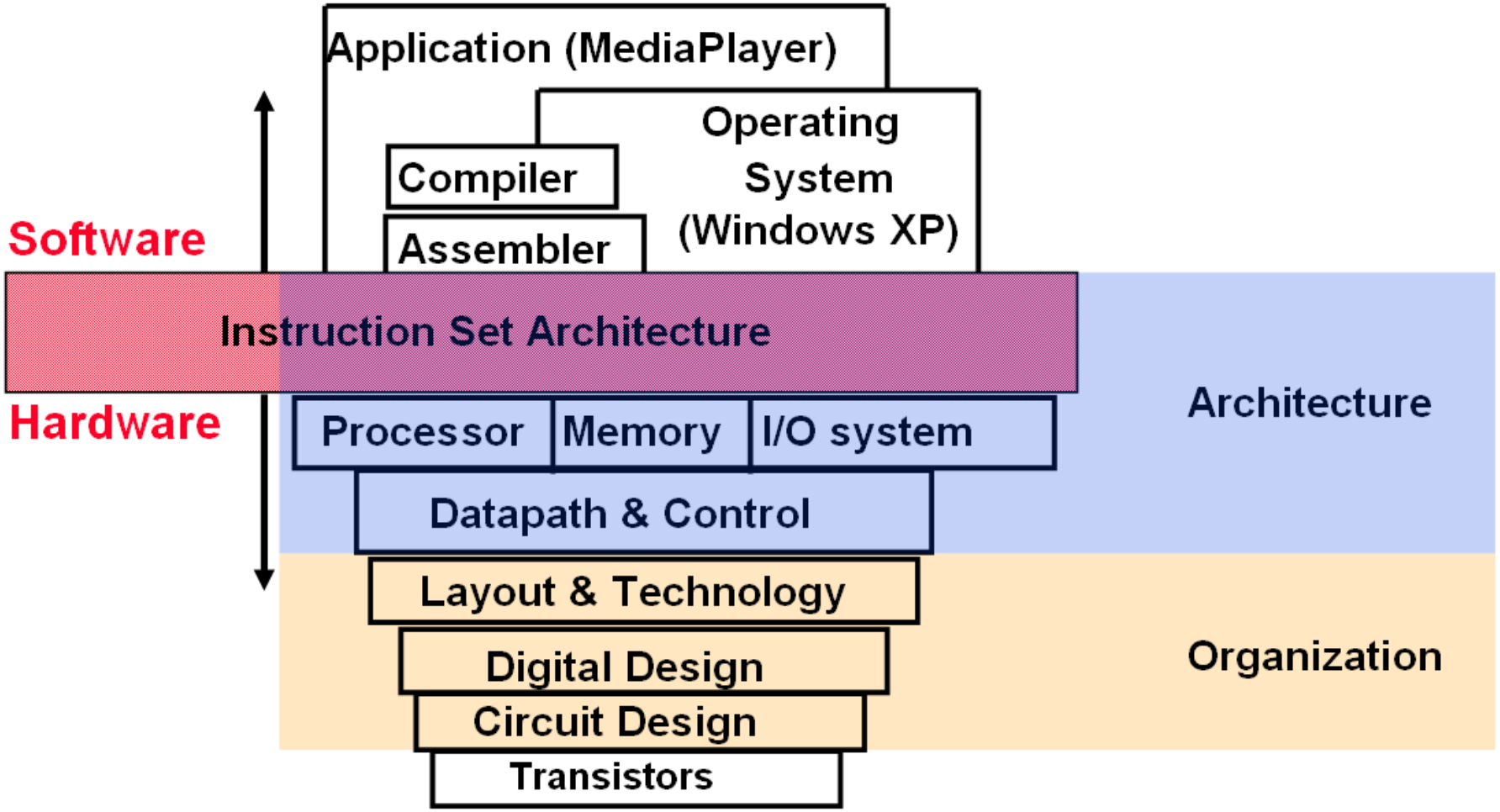


- Operates using instructions provided by the control unit
- Comprises adders, counters, and registers

Instruction Processing

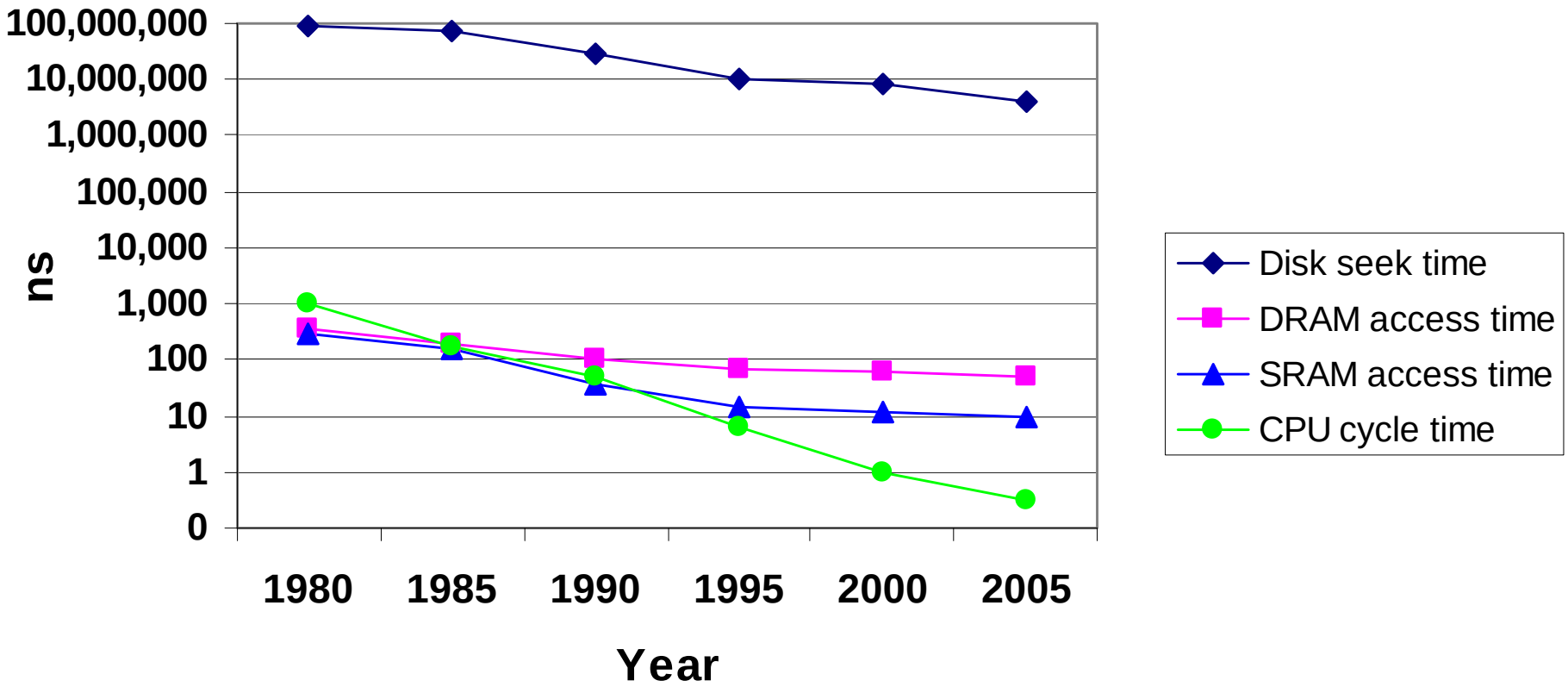
- **Instruction Set Architecture (ISA)**
  - Design of the CPU's **instruction set** (set of unique operations)
  - visible to the programmer/compiler writer
  - the boundary between software and hardware
  - Common Types:
    - **Stack**: The operands are implicitly on top of the stack
      - (+)/ (-)??
    - **Accumulator**: One operand is implicitly the accumulator
      - (+)/(-)??
    - **General Purpose Register (GPR)**: All operands are explicitly mentioned (either registers or memory locations)
      - (+)/(-)??

ISA

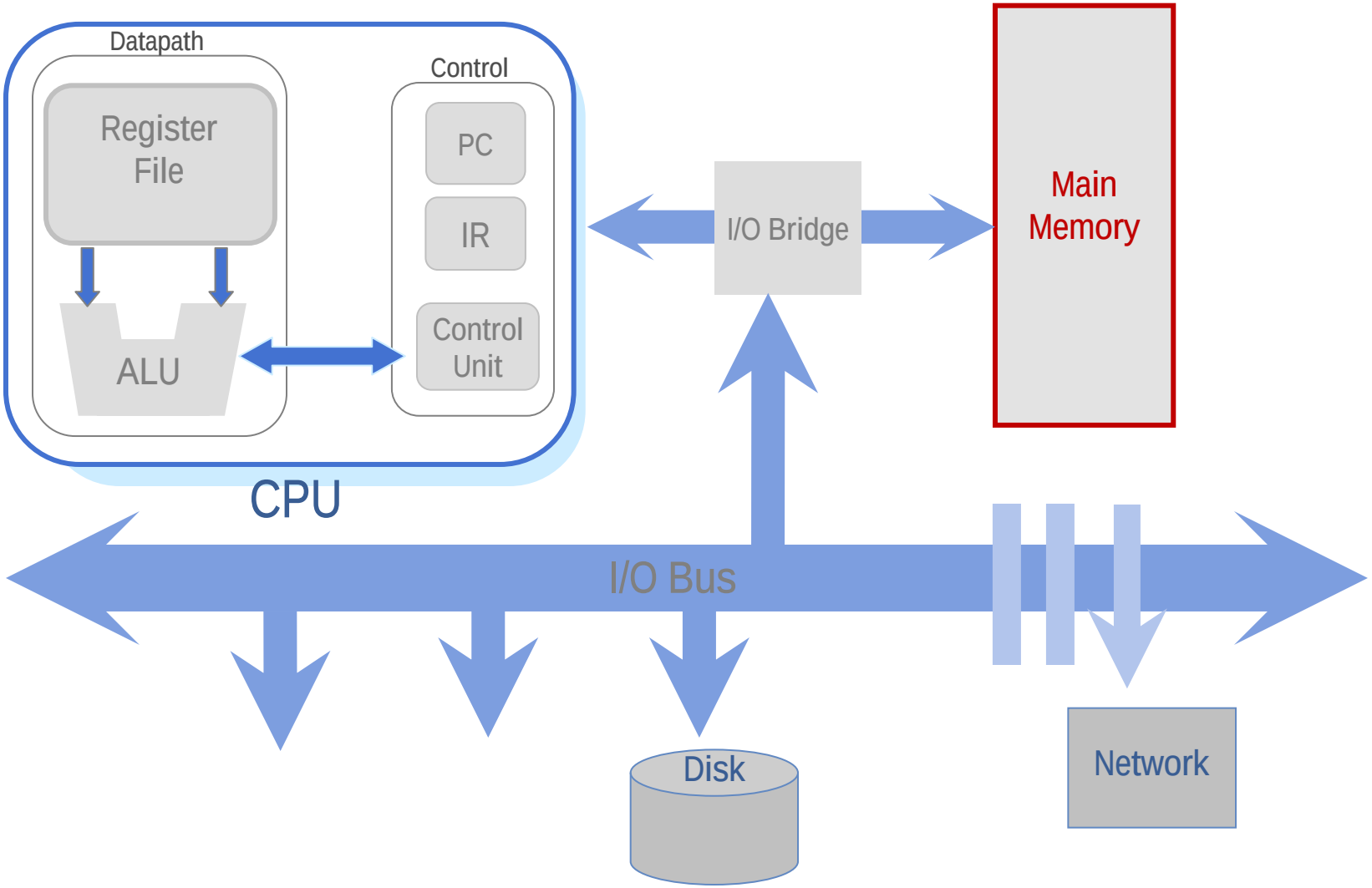

Application (MediaPlayer)
Operating System (Windows XP)
Compiler
Assembler
Software
Instruction Set Architecture
Hardware
Processor
Memory
I/O system
Datapath & Control
Architecture
Layout & Technology
Digital Design
Circuit Design
Transistors
Organization

Processor-Memory Gap

- A big problem:
  - The gap widens between DRAM, disk, and CPU speeds.



Main Memory


Datapath
Register File
ALU
Control
PC
IR
Control Unit
CPU
I/O Bridge
Main Memory
I/O Bus
Disk
Network

Locality

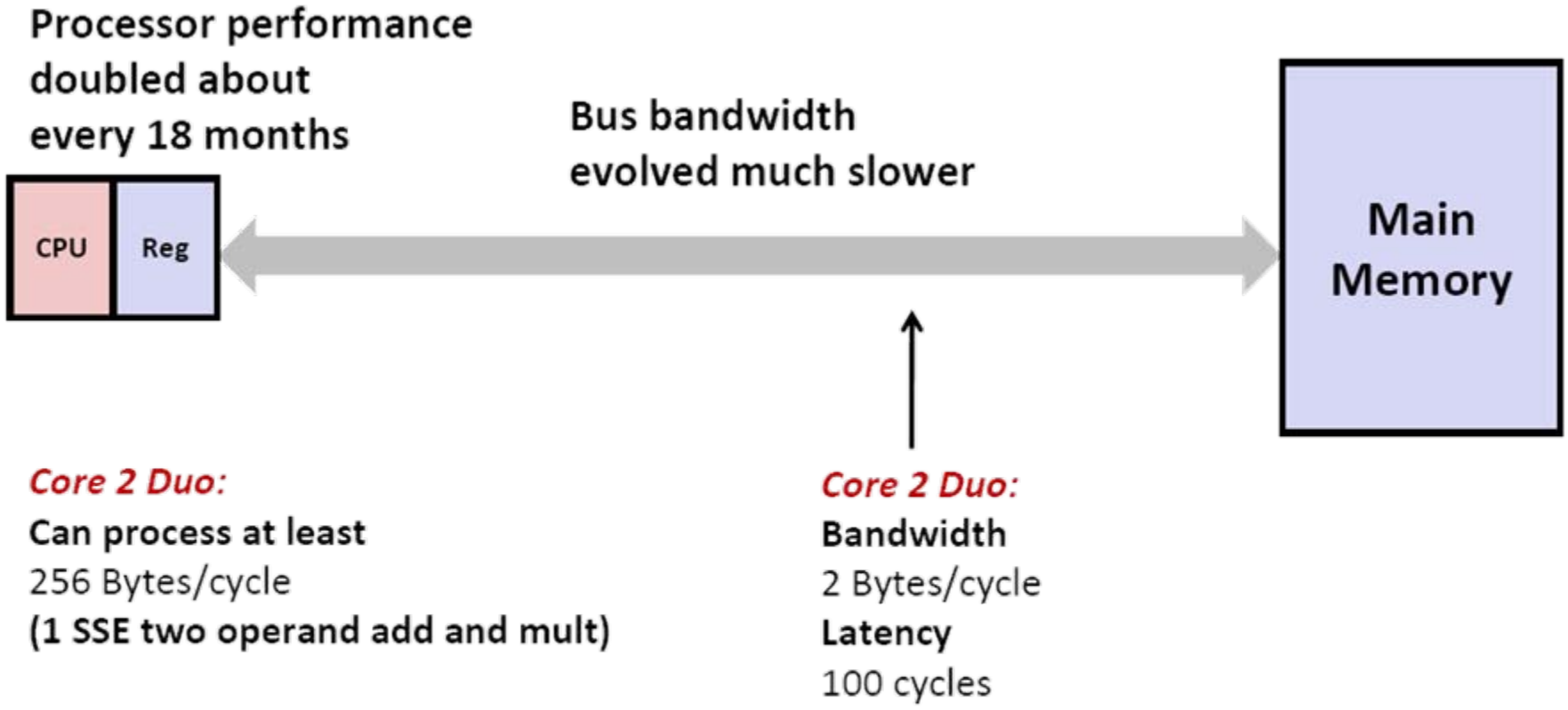
- **Principle of Locality:**
  - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
  - Temporal locality: Recently referenced items are likely to be referenced in the near future.
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time.

- **Examples: Do they have good locality?**

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

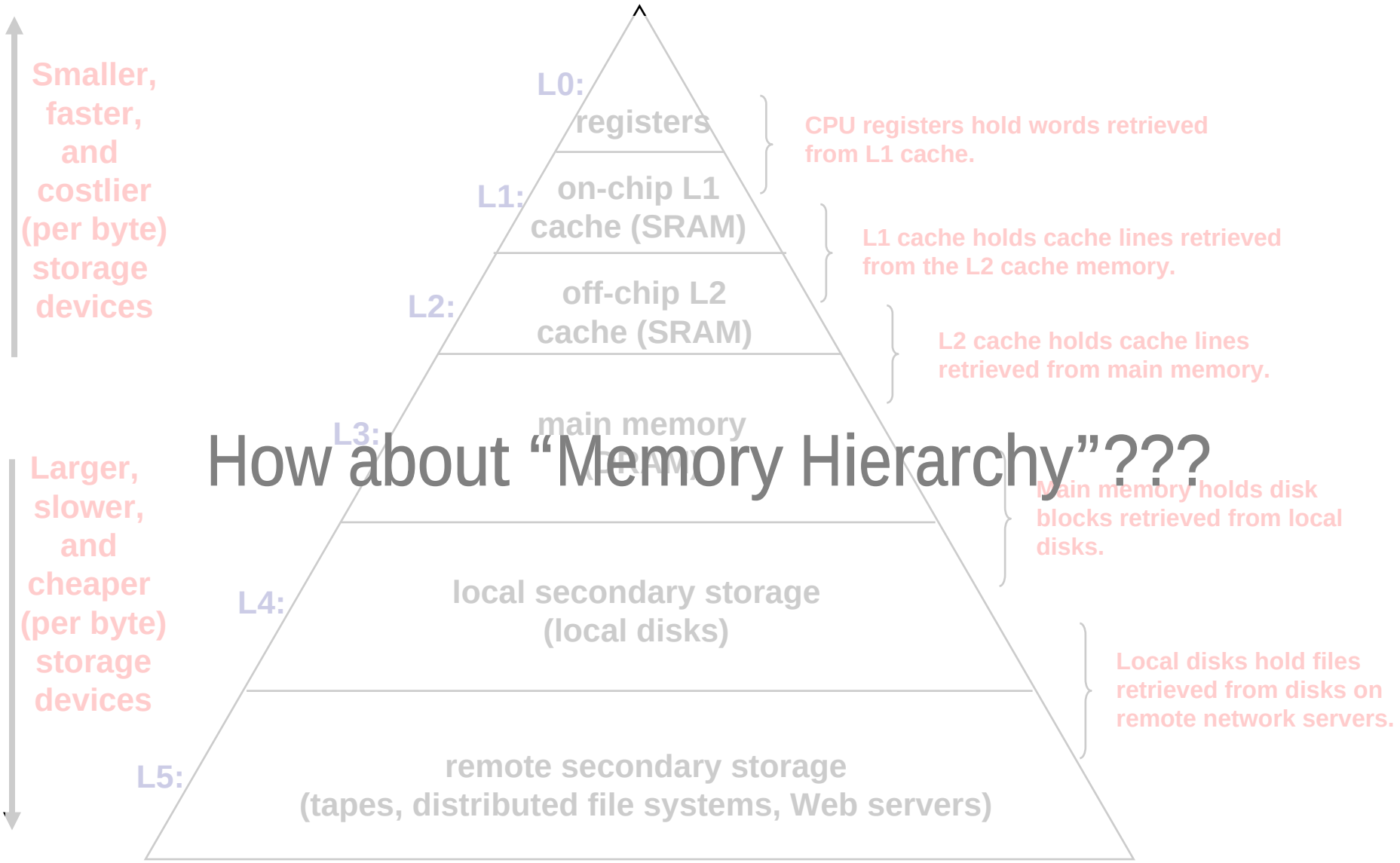
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Problem: Processor-Memory Bottleneck



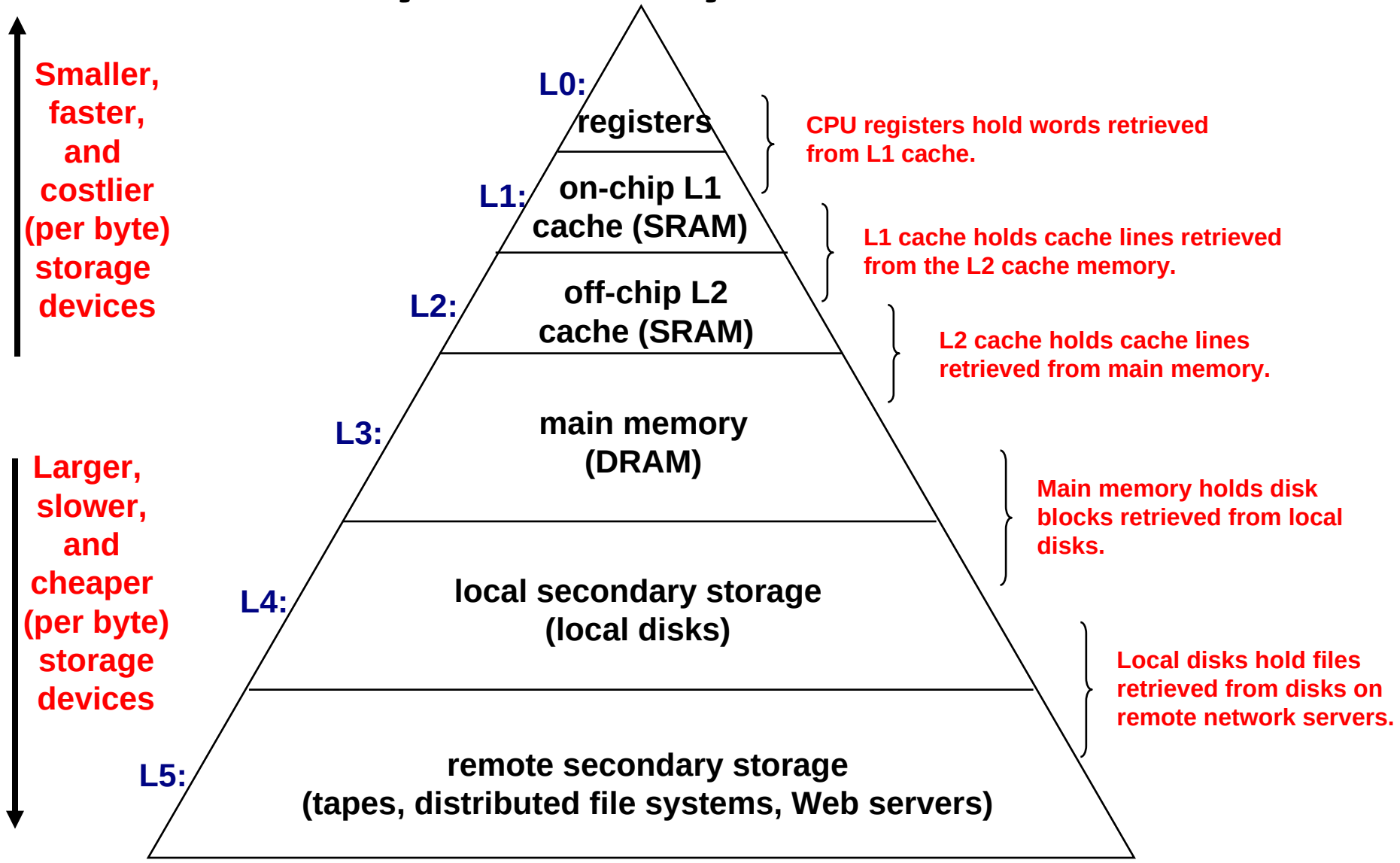
Solution: Caches

Solution



How about “Memory Hierarchy”???

The Memory Hierarchy



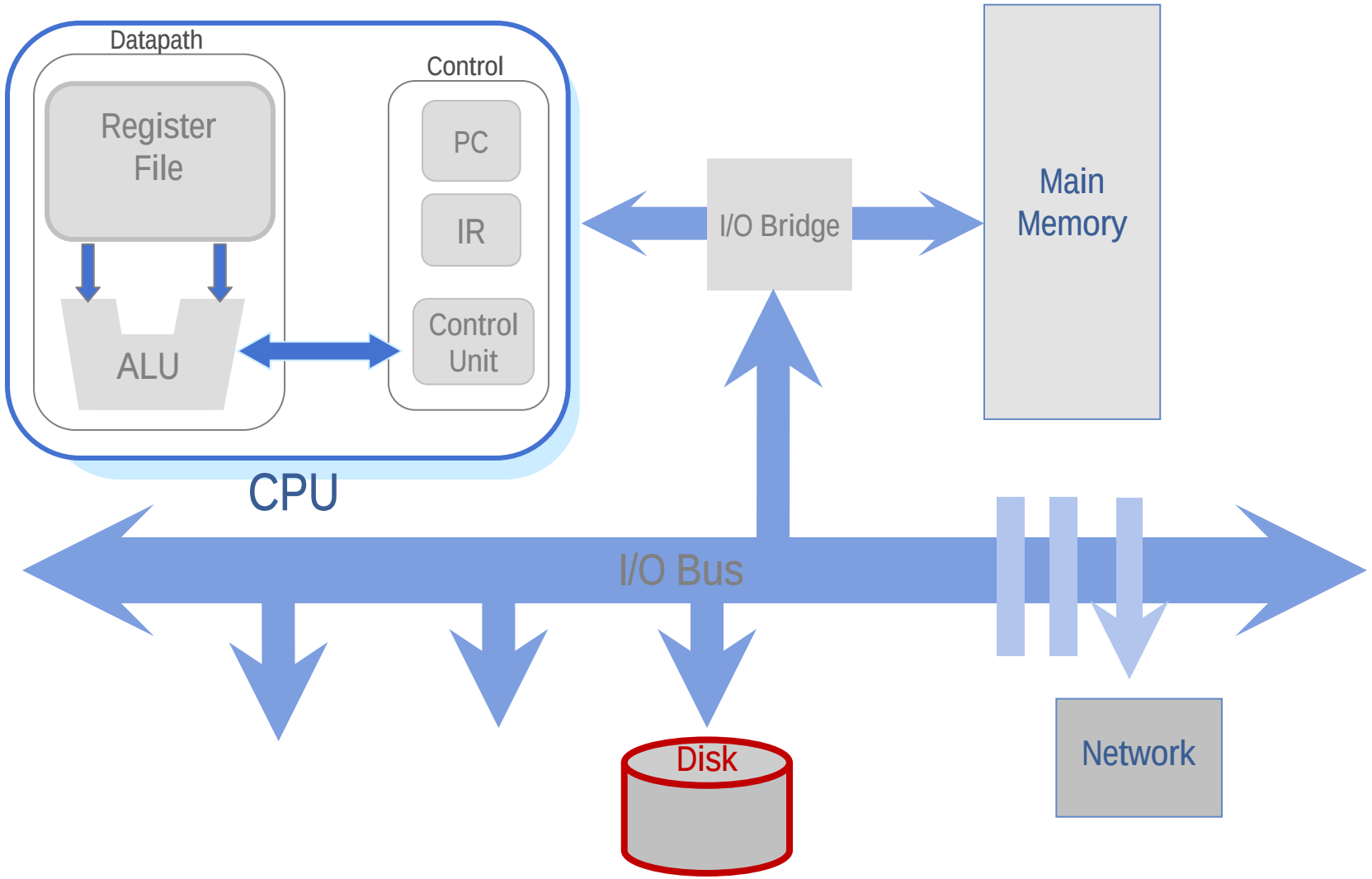
How does it work?

- **Caches: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.**
- **Fundamental idea of a memory hierarchy:**
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- **Why do memory hierarchies work?**
  - Programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.

Memory Design: Hierarchy

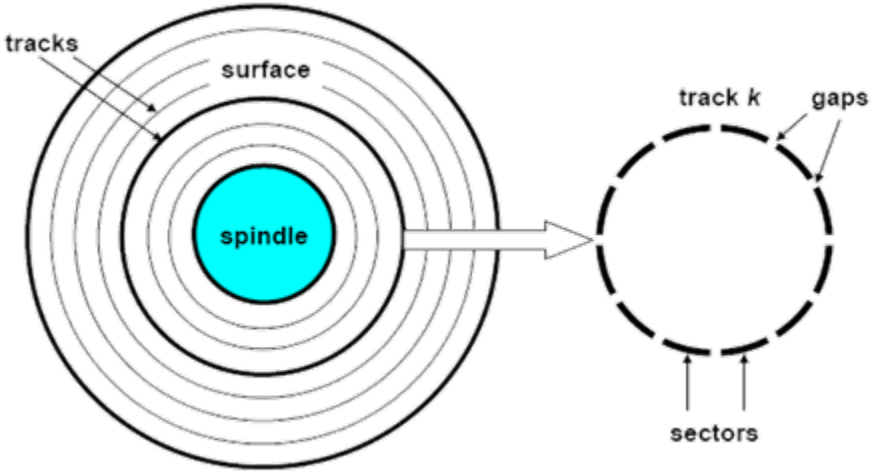
- **Why can't all of our memory be of the fastest type?**
  - Size and Cost
- **Memory Hierarchy effect:** A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.
  - Gives illusion that the processor is always retrieving data from the fastest memory level (if locality is high)
  - To take advantage of spatial locality:
    - When the needed data is found at level k+1: the needed data & its neighboring data is handed up to level k

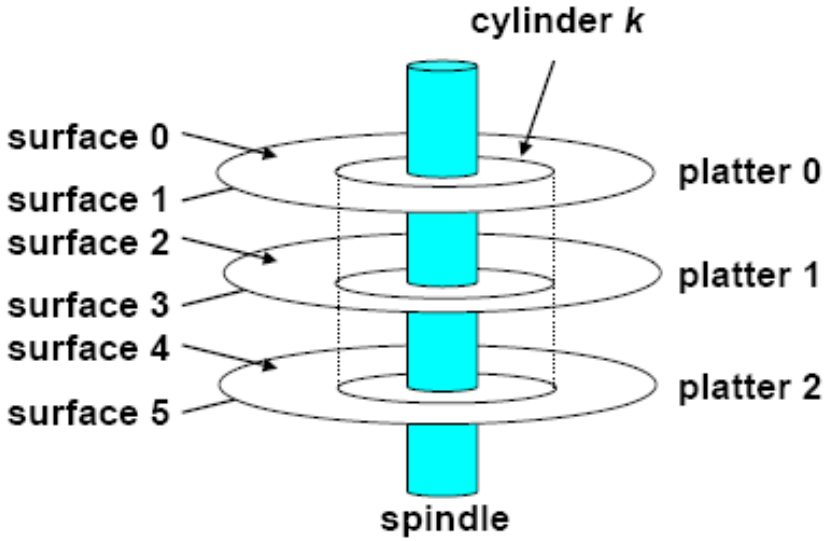
Disk


Datapath
Register File
ALU
Control
PC
IR
Control Unit
CPU
I/O Bridge
Main Memory
I/O Bus
Disk
Network

Disk Geometry

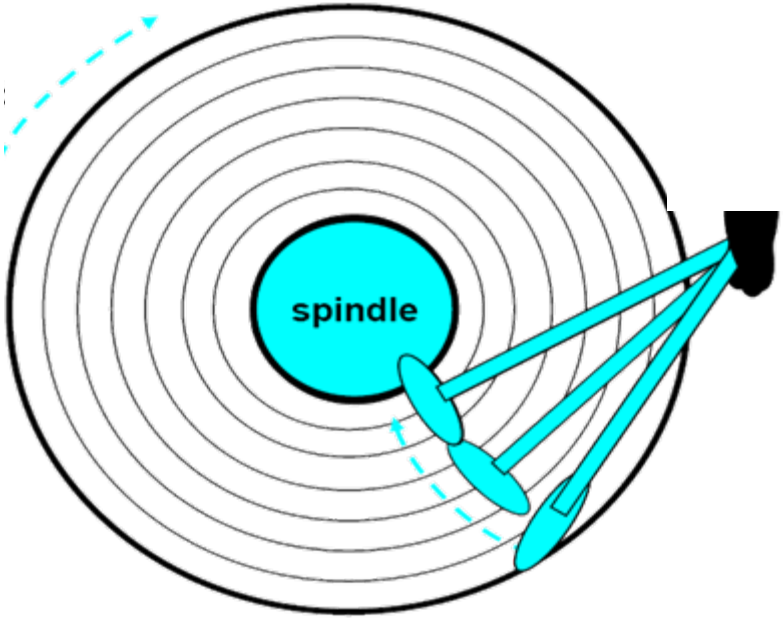
- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.
- Aligned tracks from a cylinder.

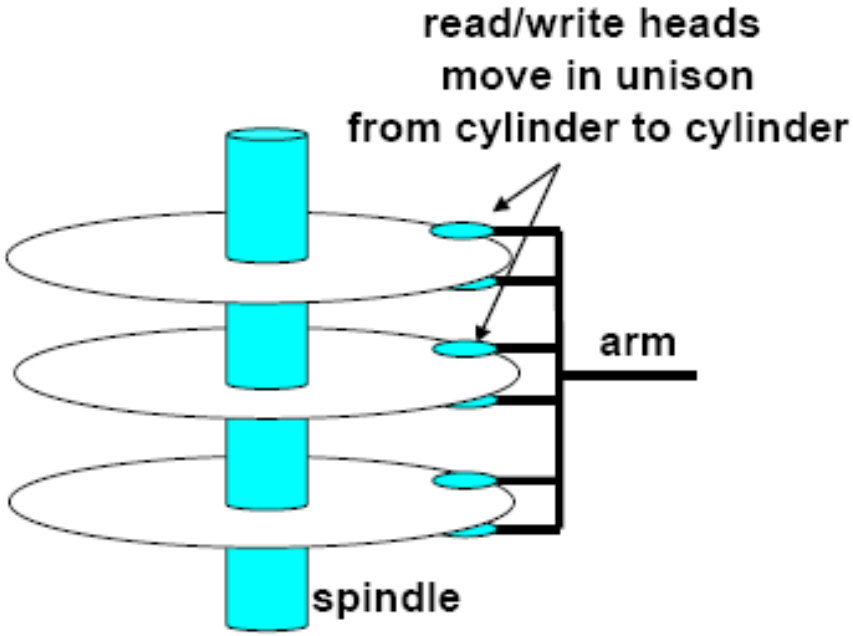




Disk Operation (Single-Platter View)

- The disk surface spins at a fixed rotational rate
- The read/write head is attached to the end of the arm and flies over the disk surface
- By moving radially, the arm can position the read/write head over any track





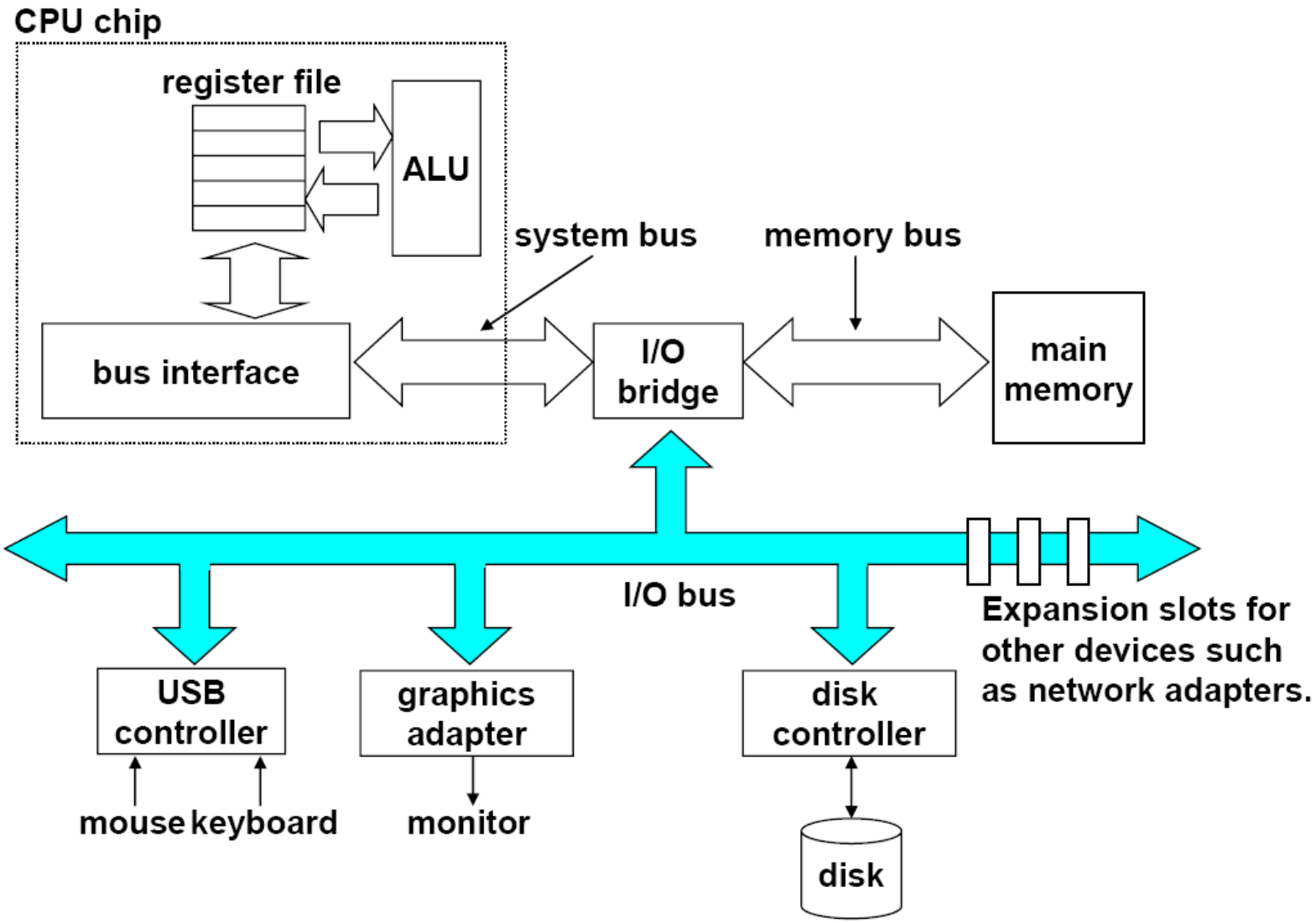
Disk Access Time

- Average time to access some target sector approximated by:
  - Taccess= Tavg seek + Tavg rotation + Tavg transfer
- Seek time (Tavg seek)
  - Time to position heads over cylinder containing target sector.
  - Typical Tavg seek = 9 ms
- Rotational latency (Tavg rotation)
  - Time waiting for first bit of target sector to pass under r/w head.
  - Tavg rotation = 1/2 x 1/RPMs x 60 sec/1 min
- Transfer time (Tavg transfer)
  - Time to read the bits in the target sector.
  - Tavg transfer = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min.

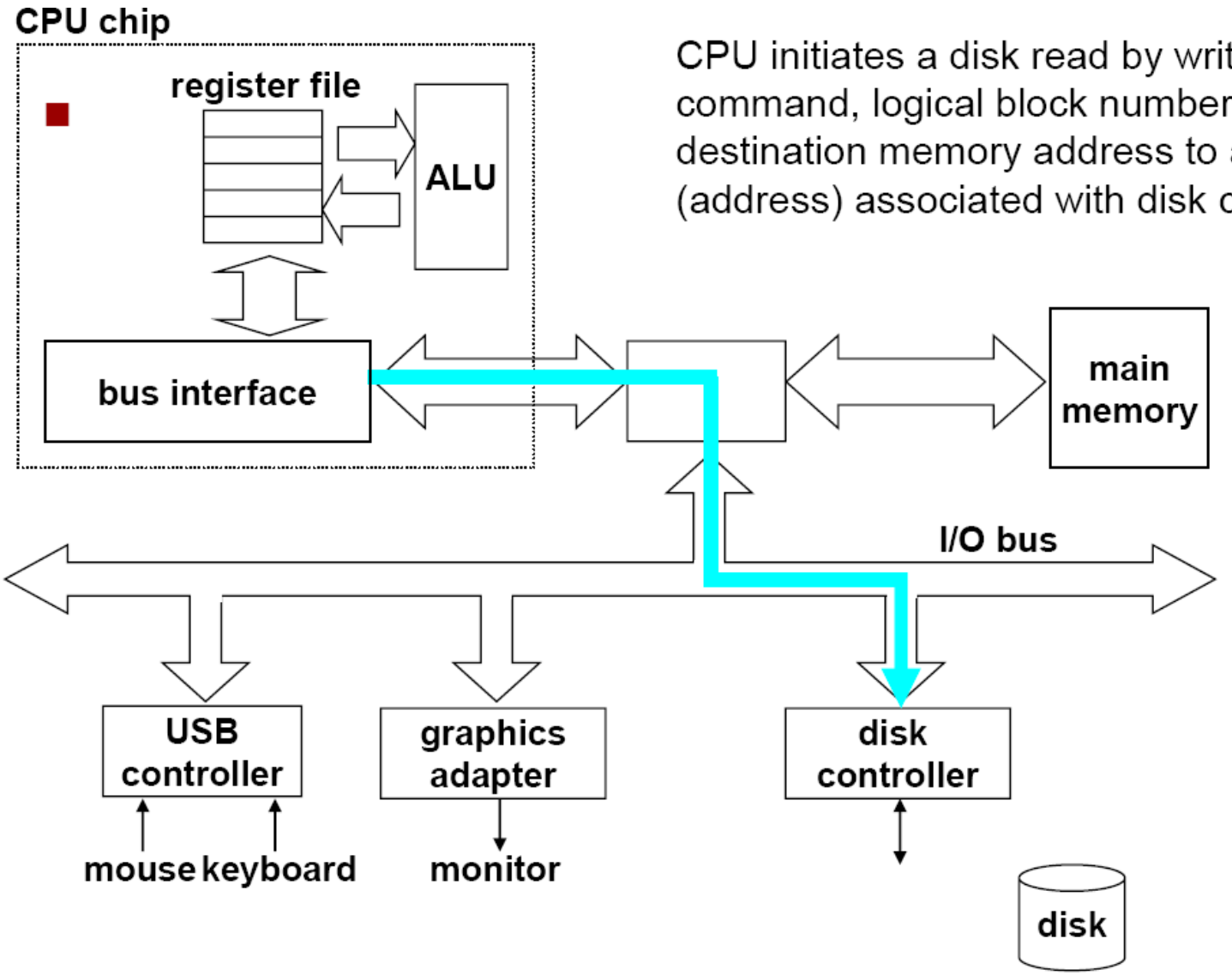
Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
  - The set of available sectors is modeled as a sequence of b sized logical blocks(0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller.
  - Converts requests for logical blocks into (surface, track, sector) triples.
- Allows controller to set aside spare cylinders for each zone.
  - Accounts for the difference in “formatted capacity” and “maximum capacity”.

I/O Bus

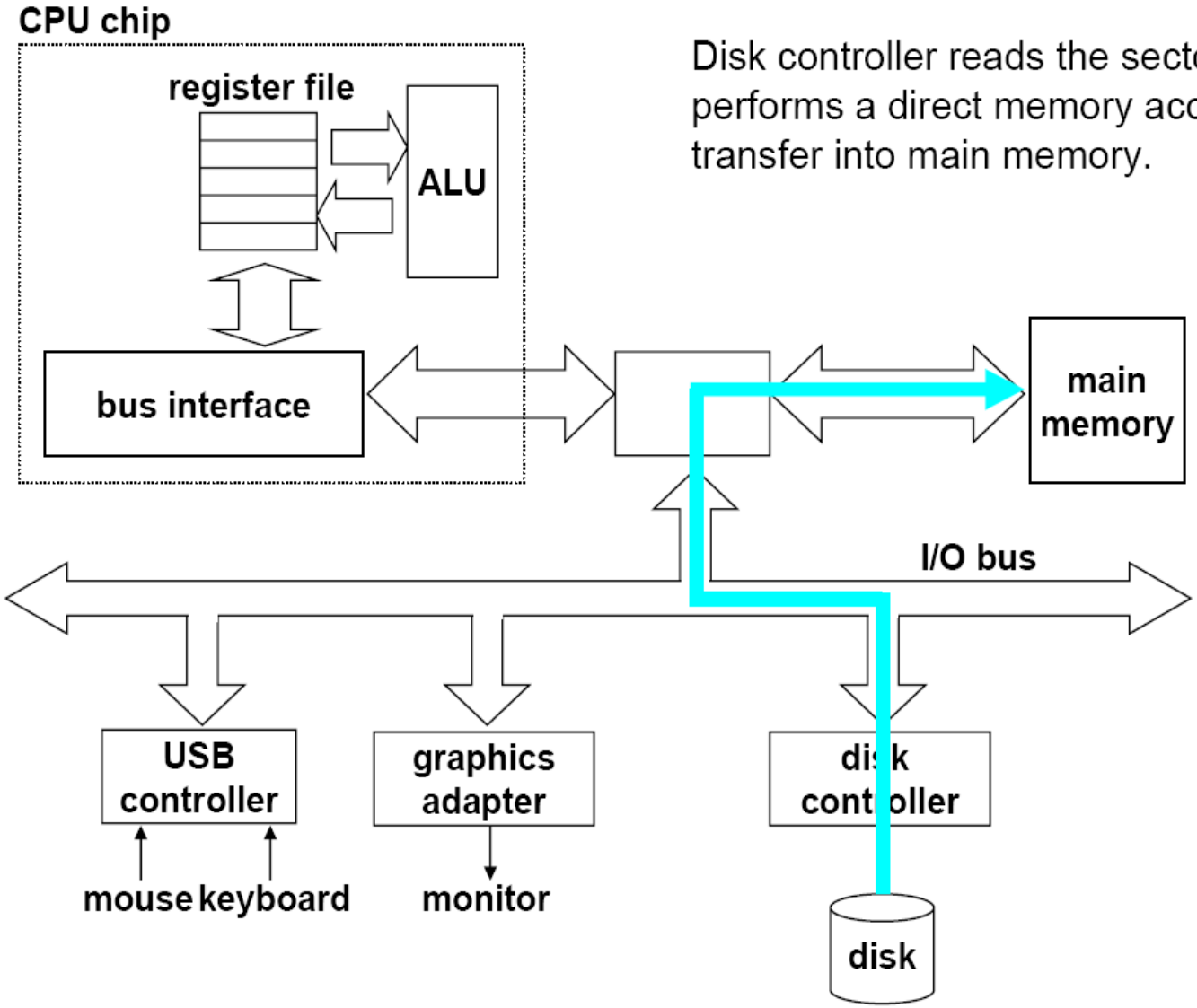

CPU chip
register file
ALU
system bus
memory bus
bus interface
I/O bridge
main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
graphics adapter
disk controller
mouse
keyboard
monitor
disk

Reading Disk Sector (1)



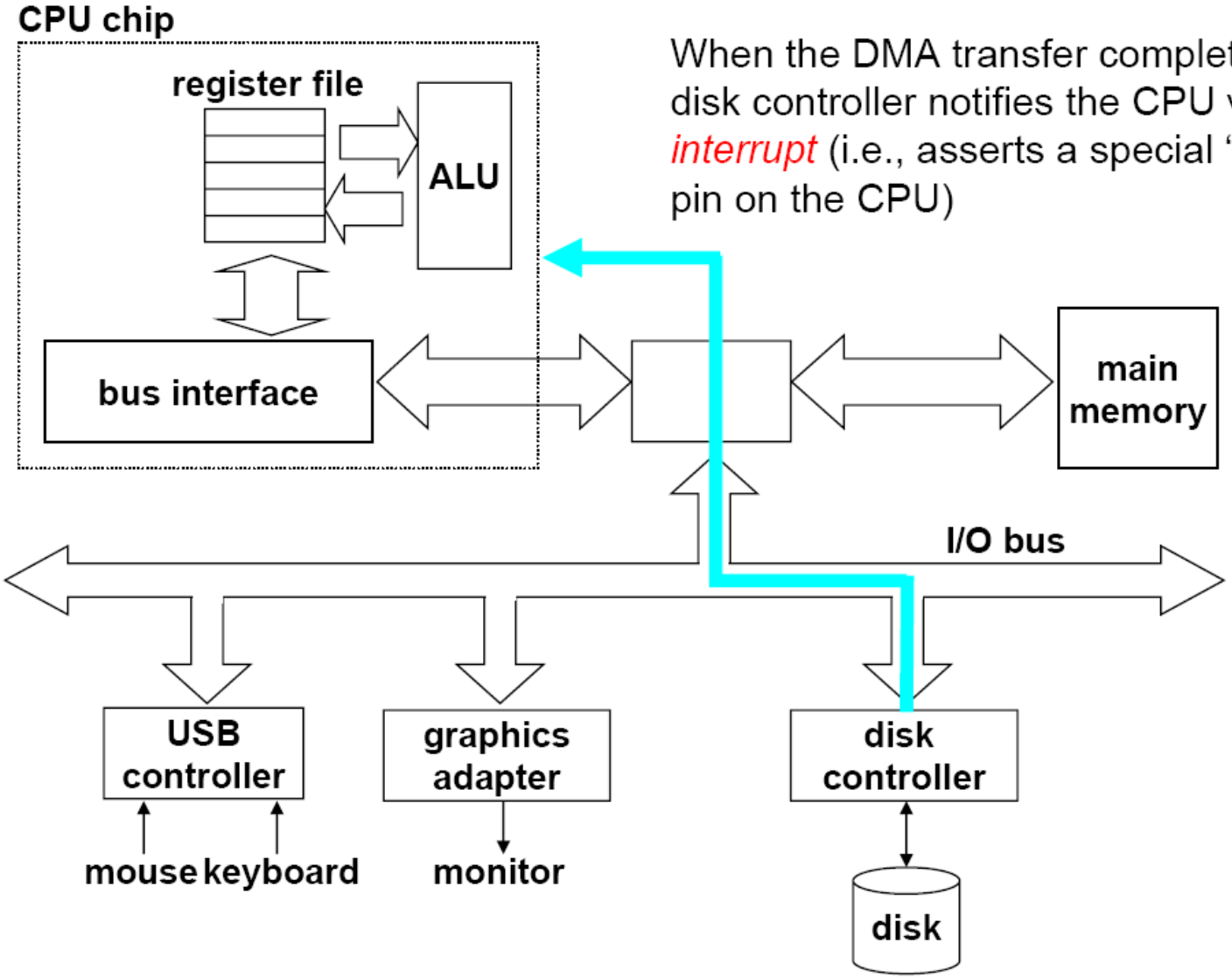
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading Disk Sector (2)



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading Disk Sector (3)



When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

Storage Trends

SRAM

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	19,200	2,900	320	256	100	75	*256*
access (ns)	300	150	35	15	12	10	*30*

DRAM

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	8,000	880	100	30	1	0.20	*40,000*
access (ns)	375	200	100	70	60	50	*8*
typical size(MB)	0.064	0.256	4	16	64	1,000	*15,000*

Disk

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	500	100	8	0.30	0.05	0.001	*10,000*
access (ms)	87	75	28	10	8	*4*	*22*
typical size(MB)	1	10	160	1,000	9,000	400,000	*400,000*

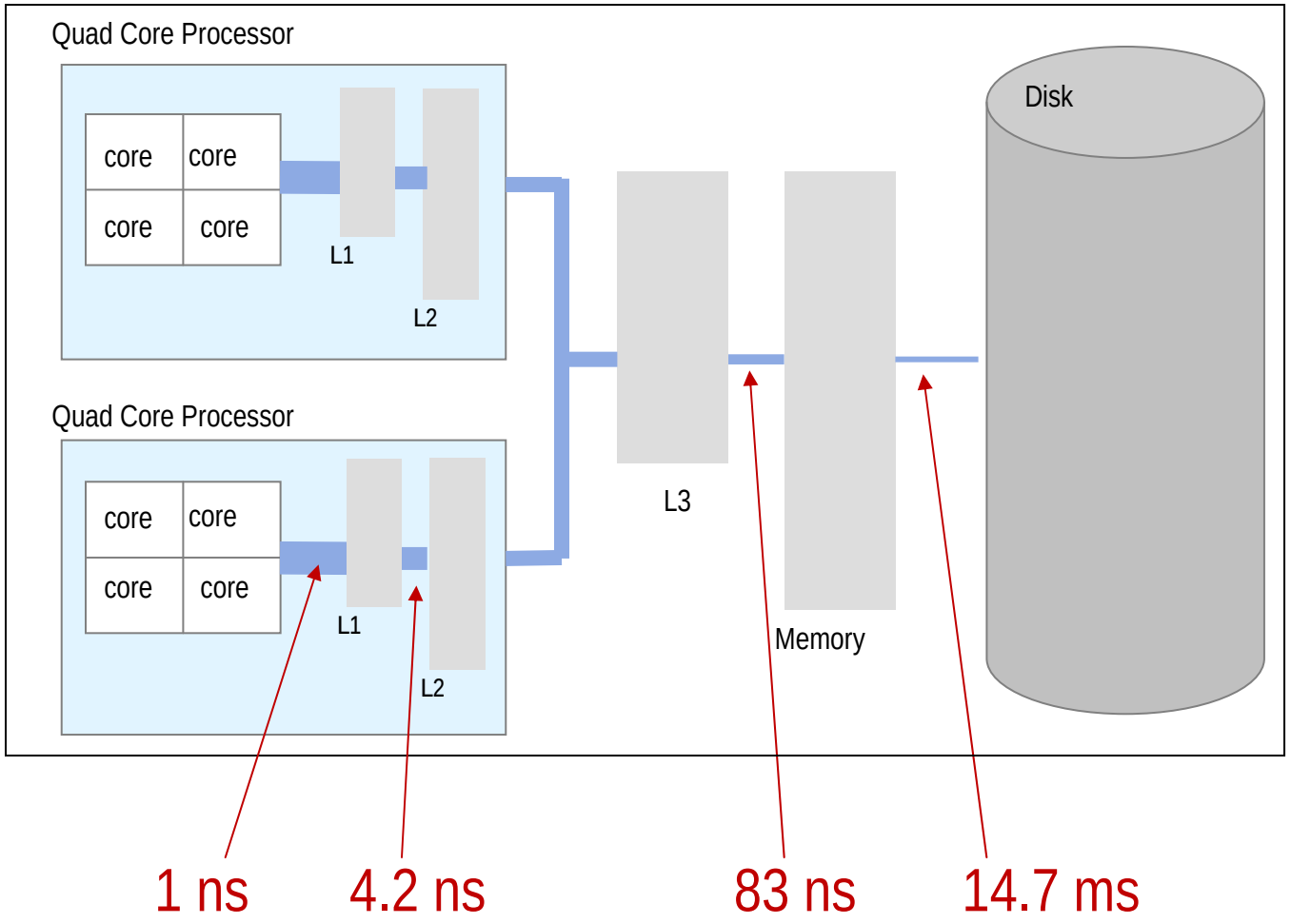
What is the Typical Bandwidth Between … ?

- L_1 & L_2 ?
- L_2 & L_3?
- L_3 & Main Memory?
- Main Memory & Disk?

- How can this affect the machine's performance?

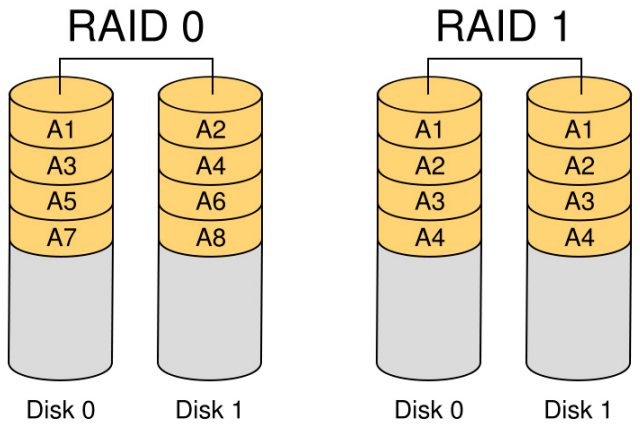
- How can it affect the Cloud's performance?

What is the Typical Bandwidth Between Layers?



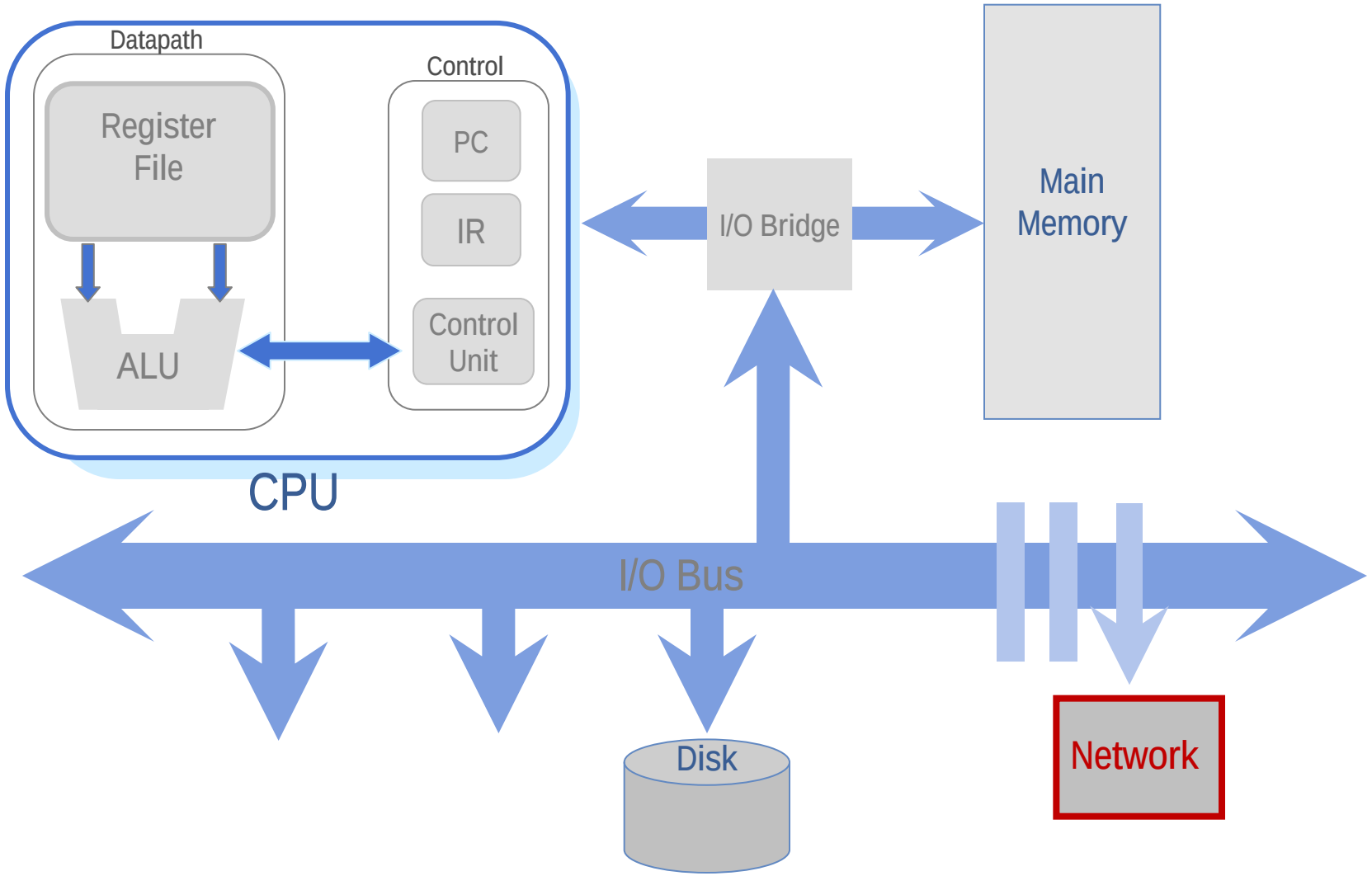
RAID: Redundant Array of Inexpensive Disks

- RAID 0: Striping (misnomer: non-redundant)
- RAID 1: Mirroring
- RAID 2: Striping + Error Correction
- RAID 3: Bit striping + Parity Disk
- RAID 4: Block striping + Parity Disk
- RAID 5: Block striping + Distributed Parity
- RAID 6: multiple parity checks



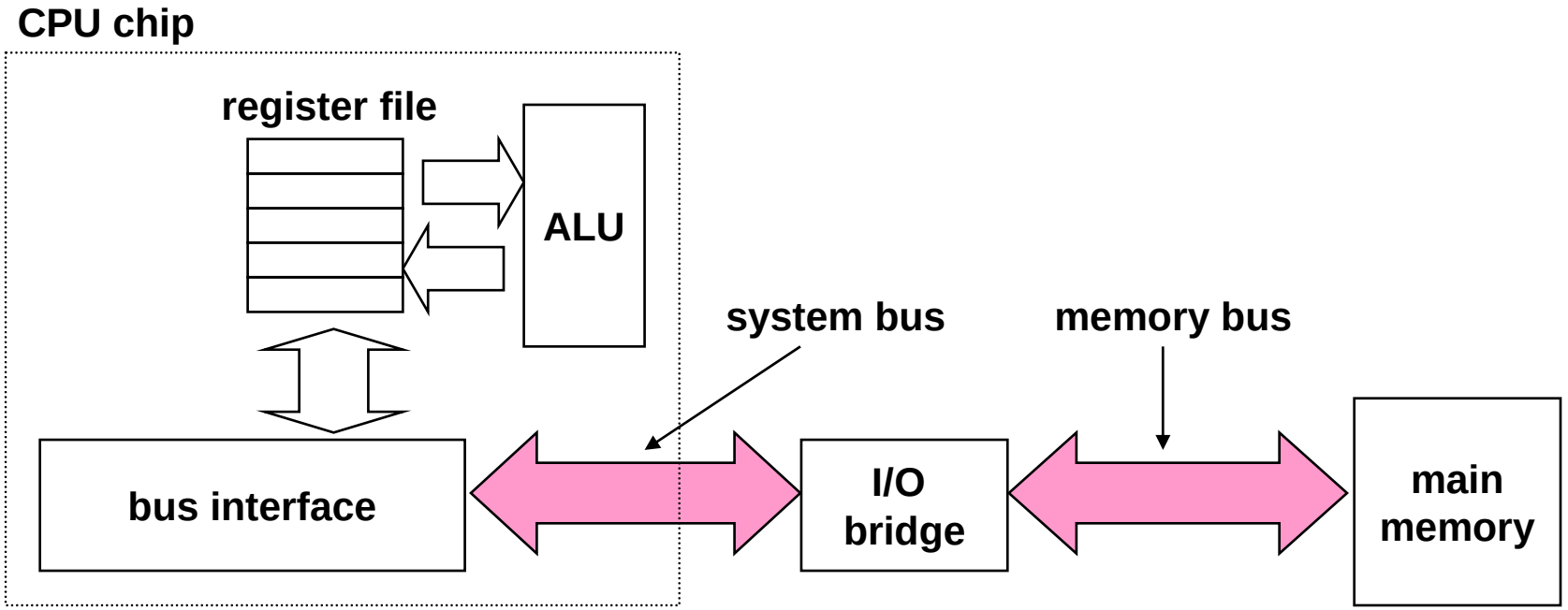
- Files are "striped" across multiple spindles
- Redundancy yields high data availability
- When disks fail, contents are reconstructed from data redundantly stored in the array
- High reliability comes at a cost:
  - Reduced storage capacity
  - Lower performance

Network


Datapath
Register File
ALU
Control
PC
IR
Control Unit
CPU
I/O Bridge
Main Memory
I/O Bus
Disk
Network

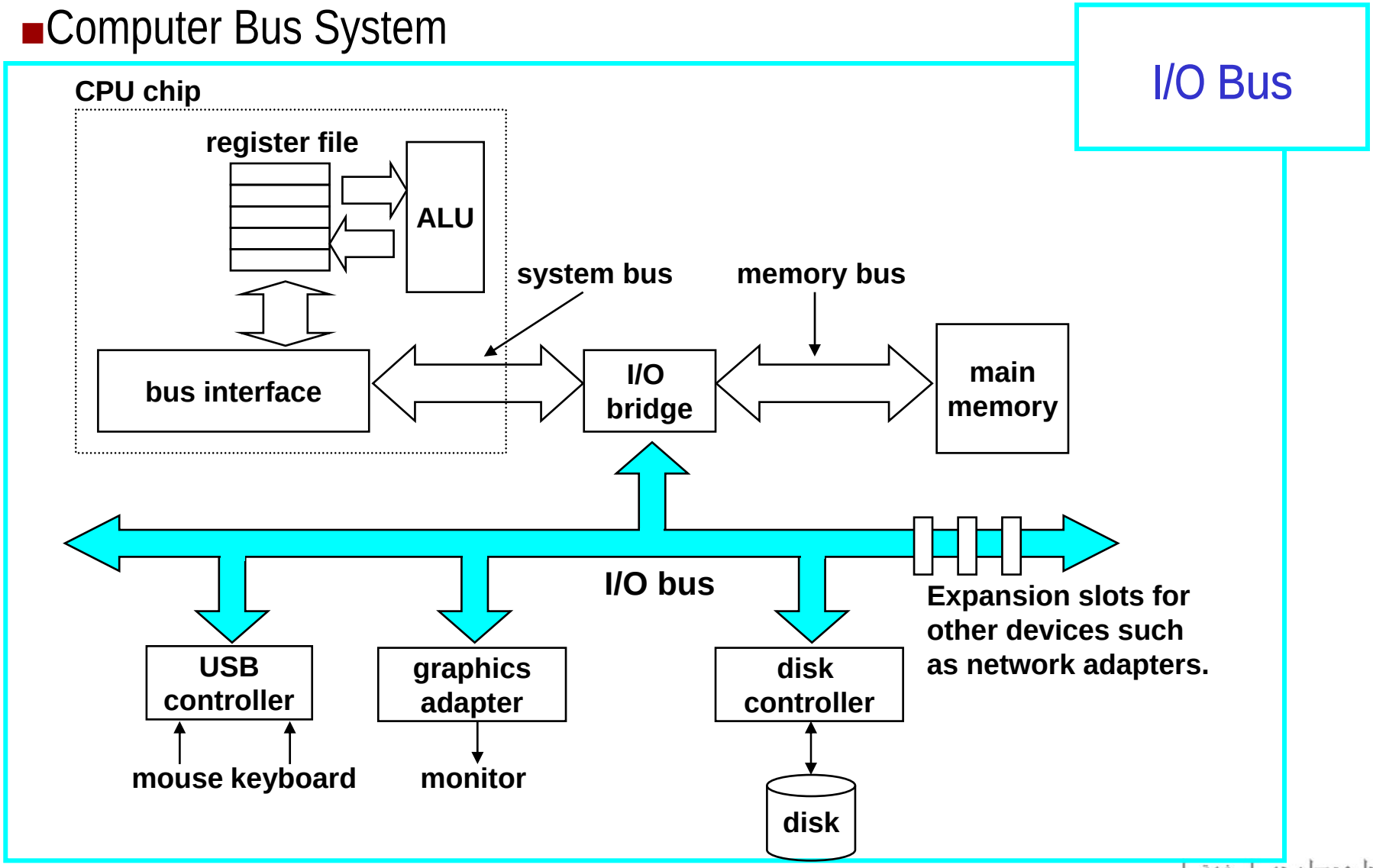
Internal Computer Network (1/2)

- Computer Bus System
  - A bus is a collection of parallel wires that carry address, data, and control signals.
  - Buses are typically shared by/ Connect multiple devices.
  - Since the 70's



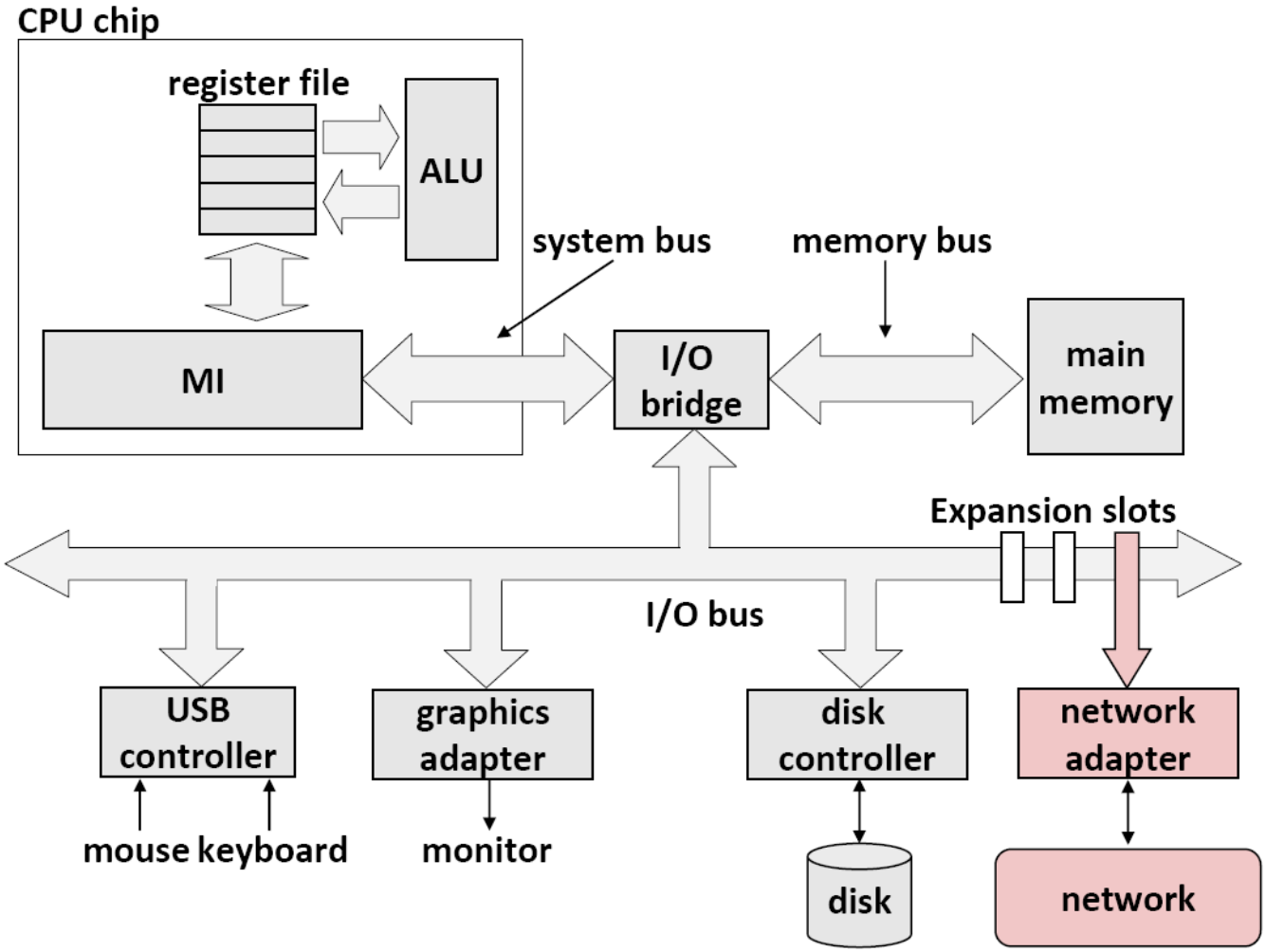
Internal Computer Network (2/2)

- Computer Bus System



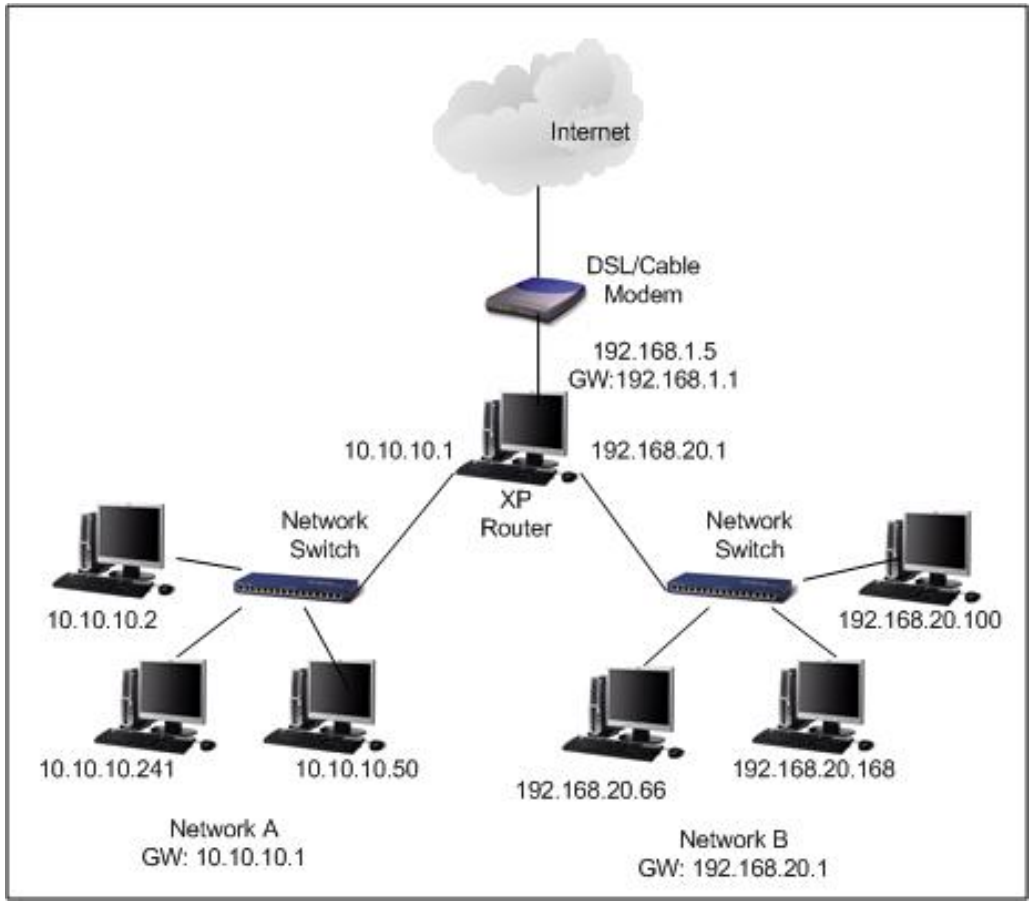
Connection Between Computers

- To connect it to other computers add a “Network Adaptor”



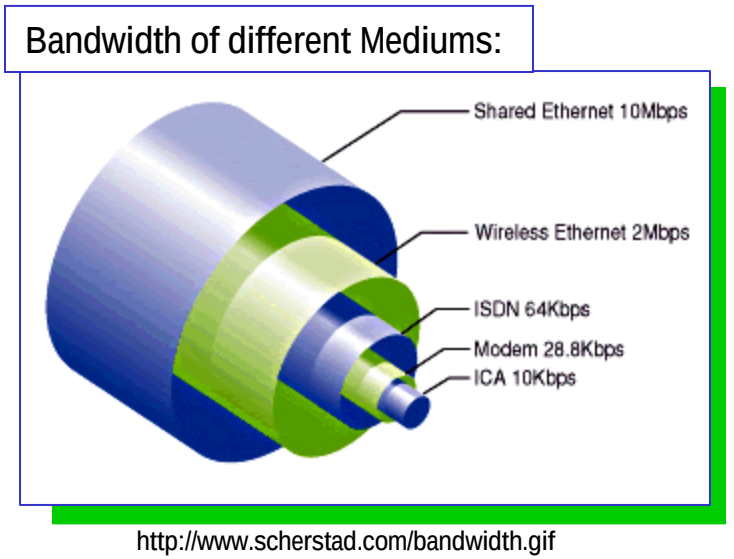
Connection Between Computers

- **Network:** two or more communication devices connected together.
- **Networks Mediums:**
  - Wired: Fiber optic, coaxial, Ethernet
  - Wireless: air



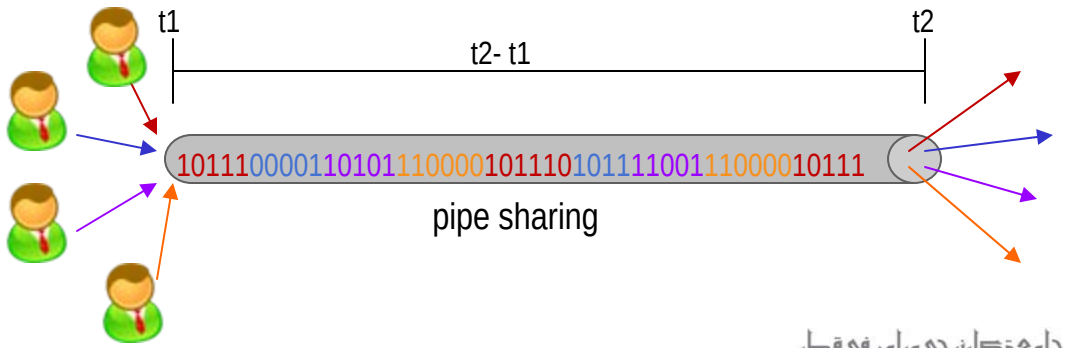
Connection Between Computers

- **Bandwidth:**
  - Data transmission rate
  - The maximum amount of information (bits/second) that can be carried along the transmission channel
  - size of the pipe…

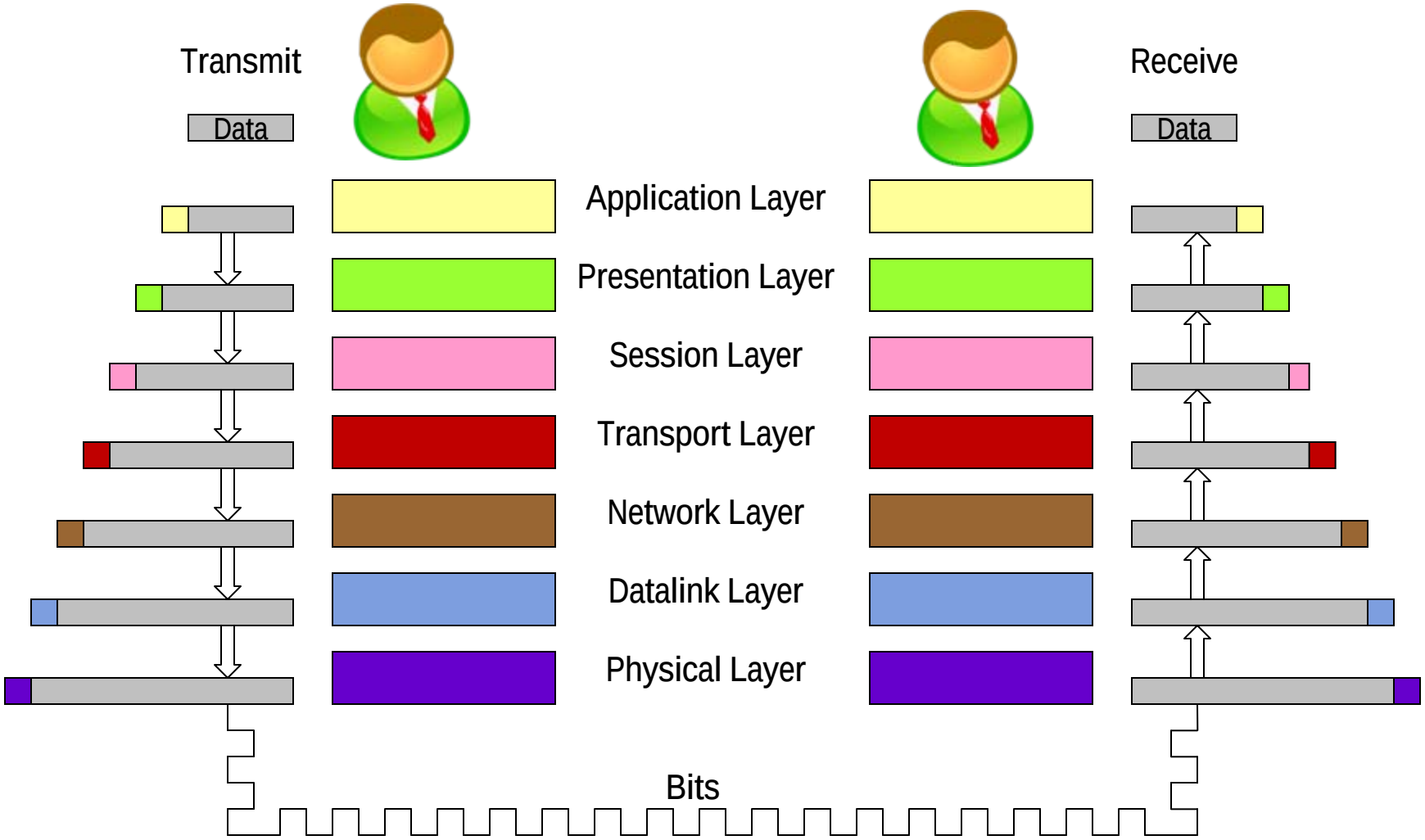


http://www.scherstad.com/bandwidth.gif

- **Latency:** time to transfer data through the pipe

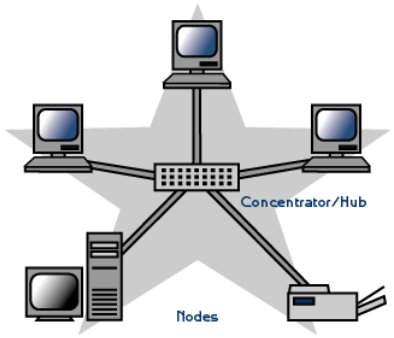


OSI Model

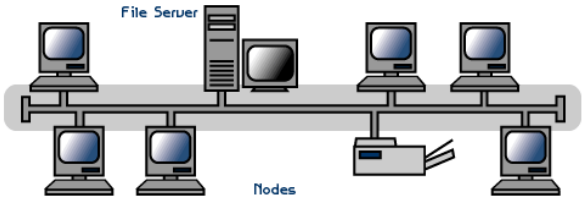

Transmit
Data
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Datalink Layer
Physical Layer
Bits
Receive
Data

Network Topologies

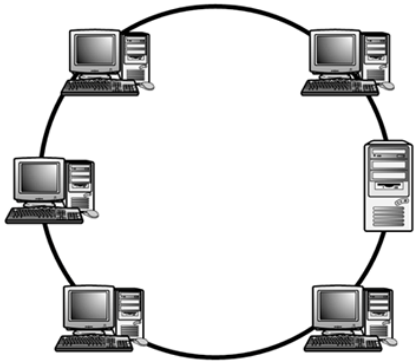
- Star



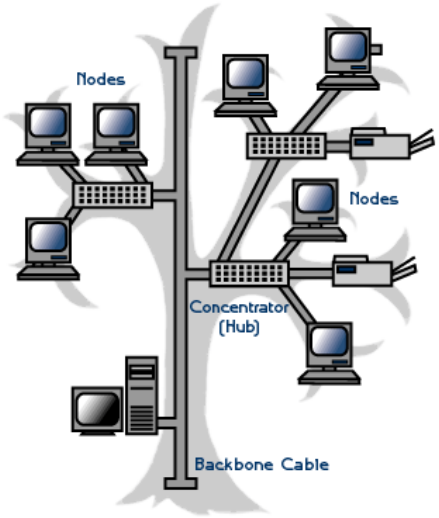
- Bus



- Ring

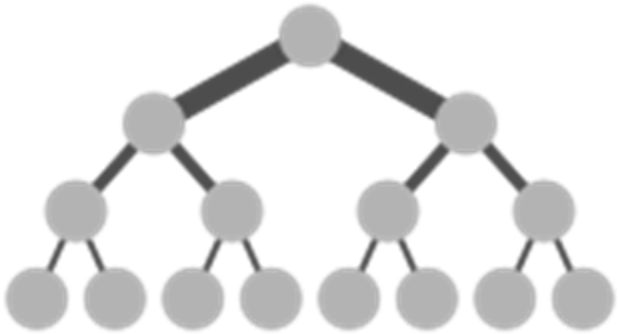


- Tree

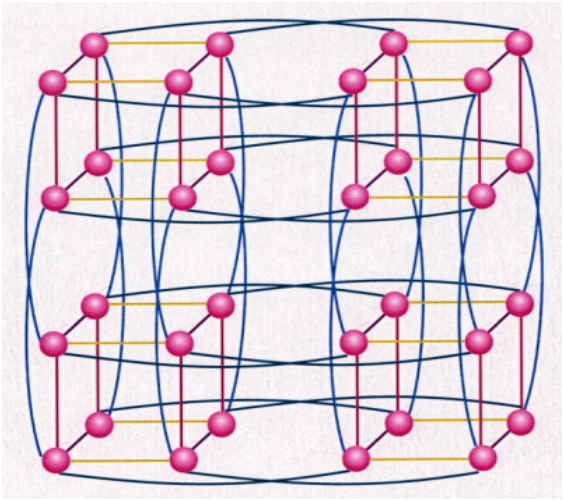


Network Topologies

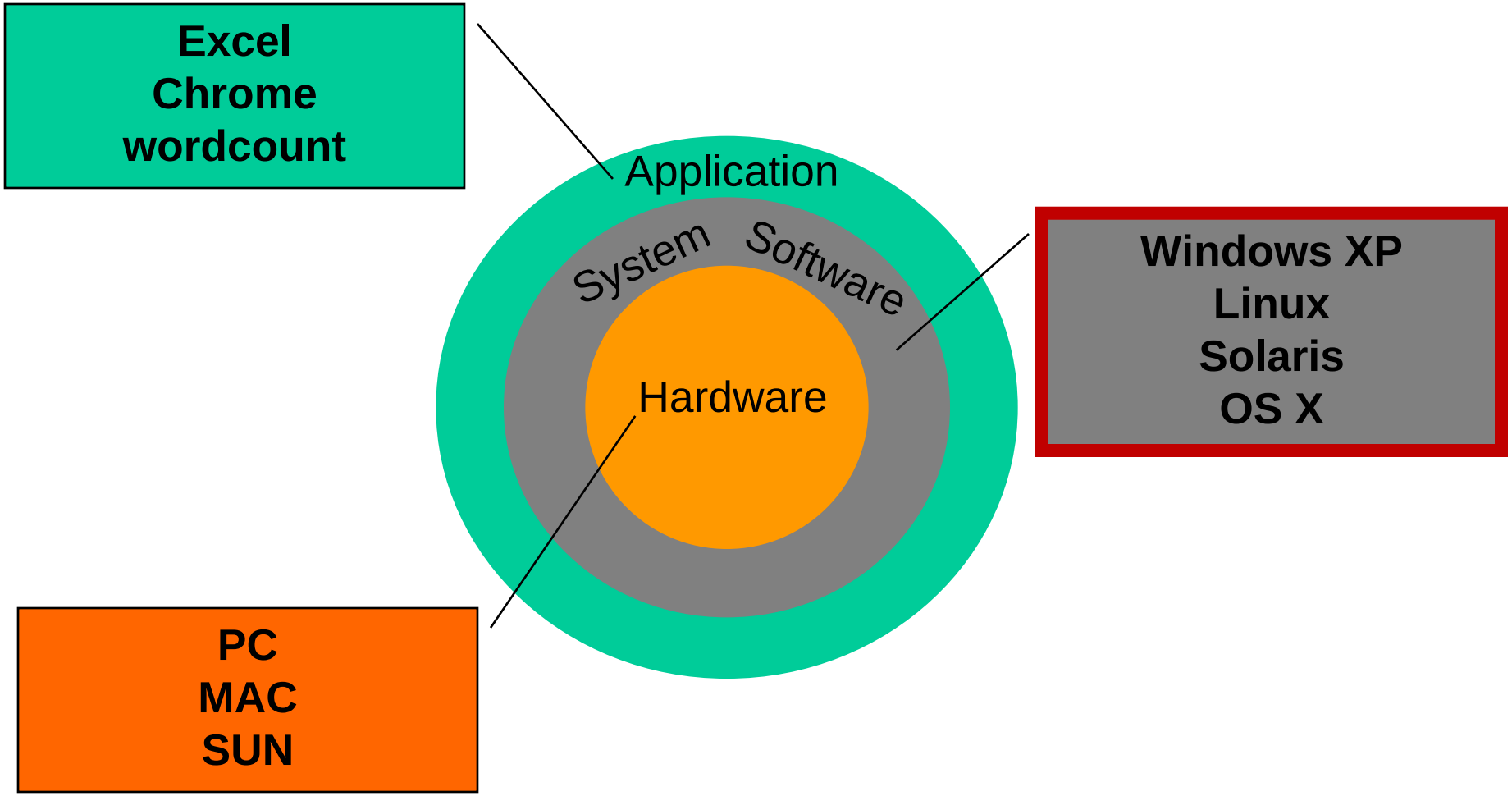
- **Fat Tree**



- **Hypercube**



How does it all work?



Program Path to Execution

High Level Language Program (.c file)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly Language Program (.asm file)

```
swap:
        mul1 $2, $5,4
        add  $2, $4,$2
        lw   $15, 0($2)
        lw   $16, 4($2)
        sw   $16, 0($2)
        sw   $15, 4($2)
        jr   $31
```

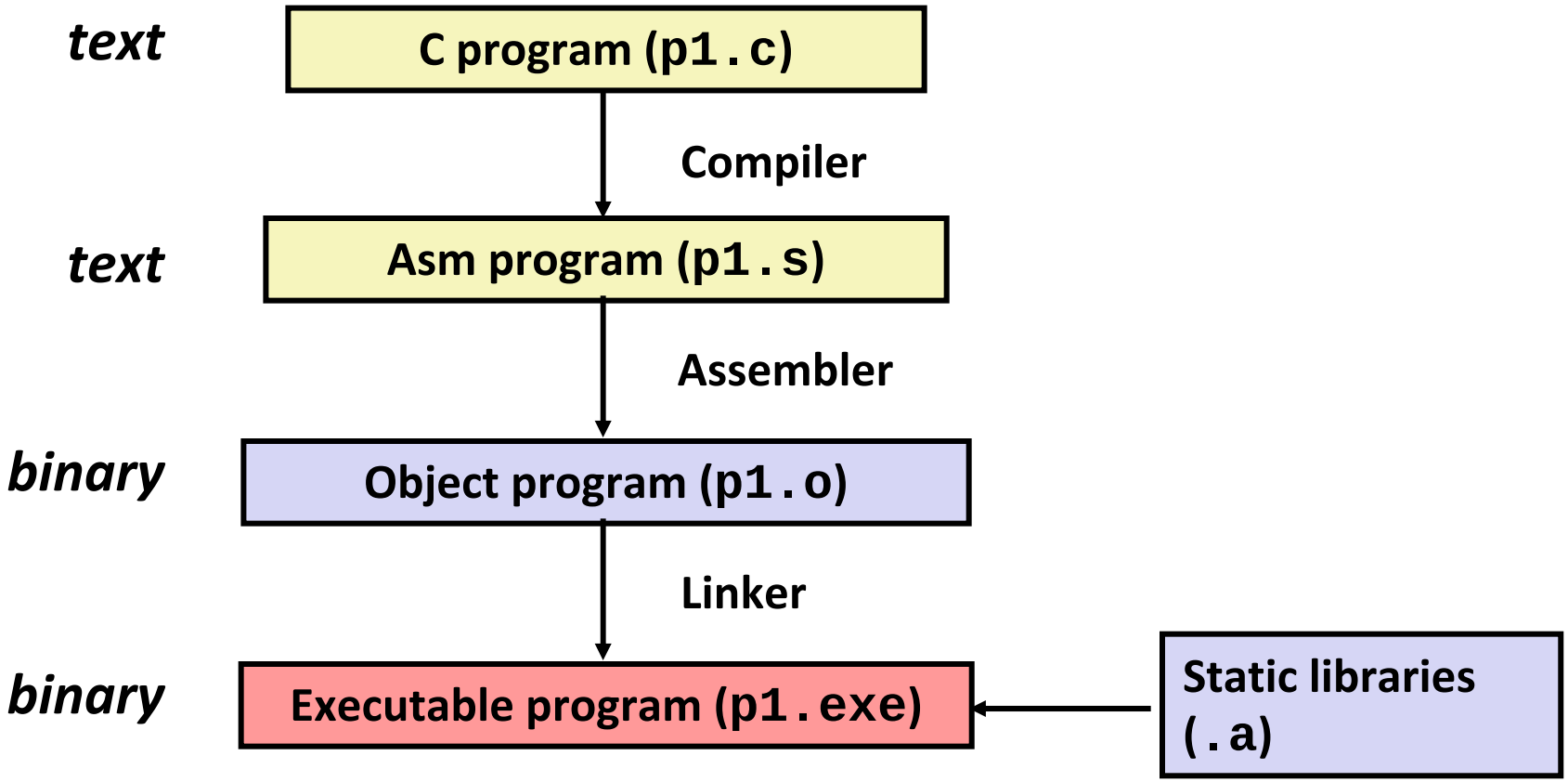
Assembler

Binary Machine Language Program (.exe file)

```
00000000101000010000000000011000
00000000000011000000110000010001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

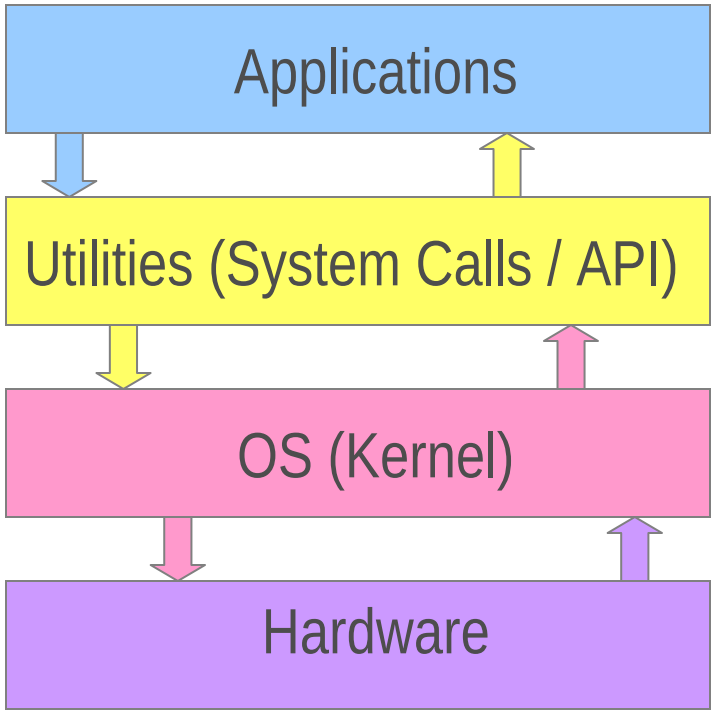
From High Level (English) to Binary (Machine)

- High Level Code in files `p1.c`
- Compile with command: `gcc p1.c`



What is Operating System?

- **A Program that controls the execution of application programs and acts as an interface between applications and the computer hardware.**



Role of the Operating System

- Process Control
- Process Scheduling
- Main Memory Management
- File System Management
- I/O Subsystem Management

Process Control

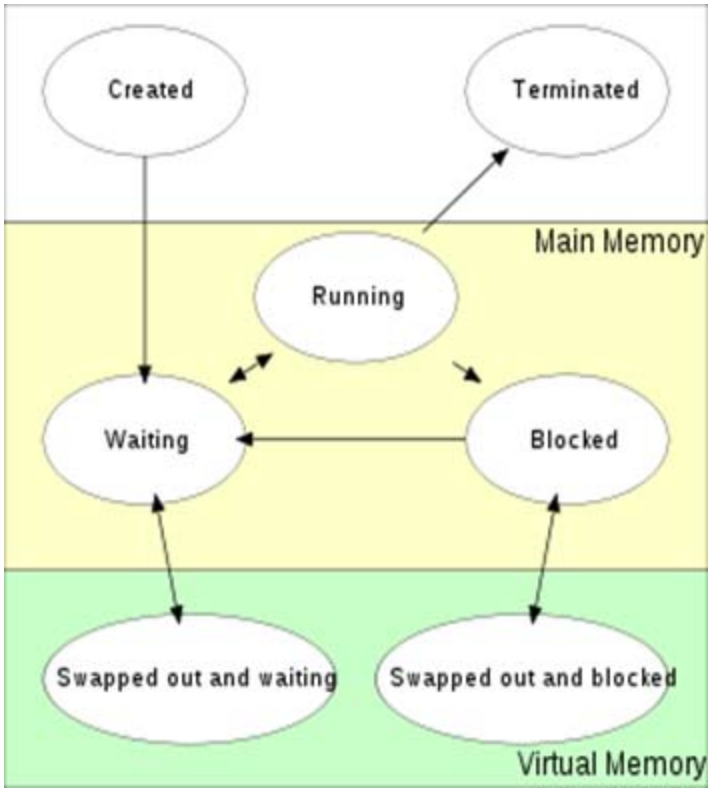
- **A process is an instance of a program in execution. It's the unit of work in the OS.**

- **Processes can be executed concurrently, each needs resources (ex: CPU time, memory, files, I/O devices) to accomplish its task.**

- **For process management, the OS is responsible for:**
  - Process creation and deletion through system calls (ex: fork & exit).
  - Process suspension and resumption through system calls (ex: wait) and signals (ex: issig).
  - Process synchronization and process communication
  - Invoking other programs through processes (ex: exec)

Process Control

- **The OS kernel keeps track of the different states the process goes through:**
  - A process is created: it's loaded from a storage device (HD) into MM.
  - Scheduler assigns state waiting to it.
  - Process waits for the scheduler to do a context switch (so it can be assigned to be executed by a processor)
  - When scheduler changed process context, its state becomes running, and execution starts.
  - If the process need resources (ex: input from a file), its state become blocked.
  - When resource is ready, process status becomes waiting until it's assigned to a processor again.
  - Once the process execution is over, it's terminated by OS, then it's removed from MM.



Source: http://en.wikipedia.org/wiki/Process_(computing)

Process Scheduling

- **Process Scheduling: is the way of determining when to run a process, assigning it to an available CPU to run on, and deciding for how long it should execute.**

- **The scheduler goals:**
  - CPU utilization
  - Maximize throughput (processes finished / time unit)
  - Minimize turnaround (time between process submission and its completion)
  - Minimize waiting time (in ready queue)
  - Minimize response time (time between request submission until the first response)
  - Increase Fairness (in CPU time allocated)

Scheduler Algorithm

```
Algorithm schedule process
Input: none
Output: none
{
        while( no process picked to execute)
        {
                for (every process on run queue)
                   pick highest priority process
        loaded in memory;

                if (no process eligible to execute)
                   idle the machine;
        }

        remove chosen process from run queue;

        switch context to chosen process, resume its
        execution;
}
```

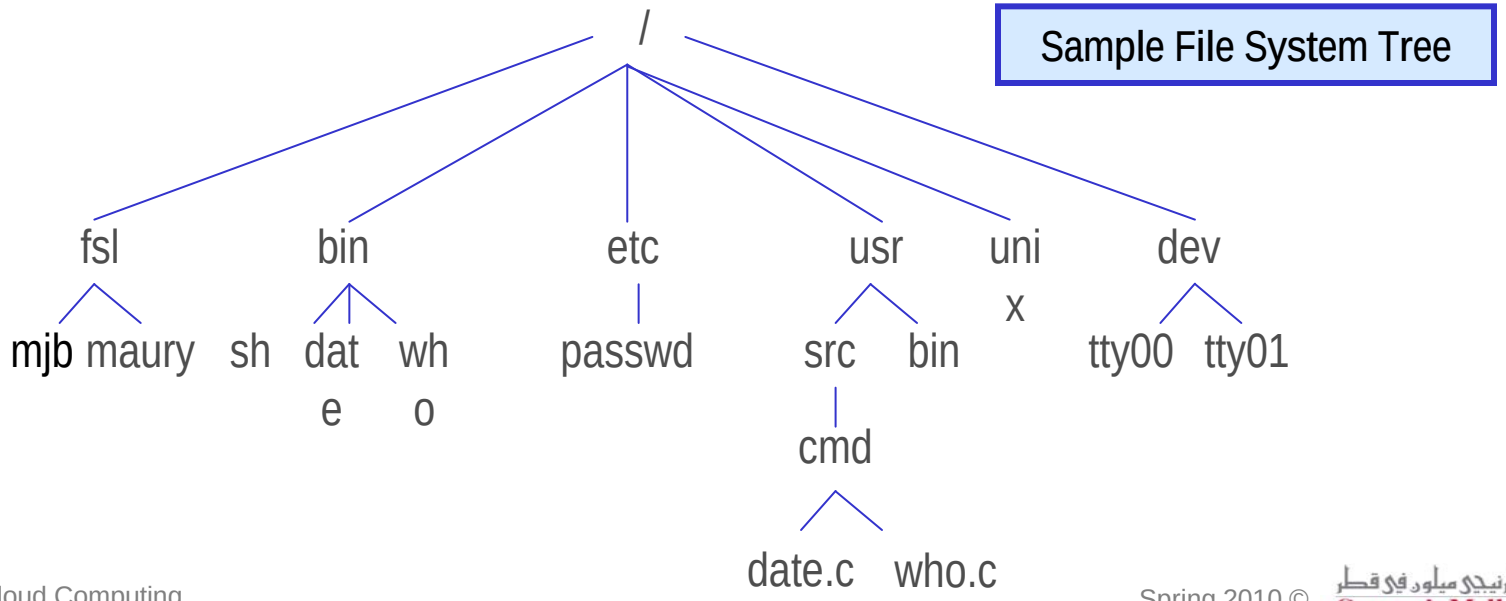
Memory Management

- **Main Memory acts as a storage area for quickly accessible data shared by the CPU and I/O devices.**
- **During instruction-fetch cycle, CPU reads instruction from MM. During data-fetch-cycle, CPU reads data from and writes data to MM.**
- **Many programs are kept in MM to improve CPU utilization and response time, but because MM is expensive it can not fit all processes in the system.**
- **For MM, OS is responsible for:**
  - Keeping track of which parts of memory are currently being used and by whom.
  - Deciding which processes to load when memory space becomes available.
  - Allocating and de-allocating memory space as required by different processes.

File System Management

- **File System: is a way of storing and organizing the files of the computer and representing the data they contain to ensure easy file search and access.**

- **For FS, the OS is responsible for:**
  - Supporting the feature of hierarchical structure.
  - Files and Directories creation and deletion.
  - Consistently treatment and protection of files' data.
  - Handling the dynamic growth of files.
  - Files backup on nonvolatile storage.



Sample File System Tree

I/O Subsystem Management

- **I/O subsystem allows processes to communicate with the peripheral devices such as disks, terminals, printers, and networks.**
- **The kernel modules controlling the peripheral devices are known as device drivers.**
- **The I/O subsystem consists of:**
  - A memory management component that includes buffering, caching and spooling
  - A general device-driver interface
  - Drivers for specific hardware devices
- **Each I/O device, requires its own set of instructions or control signals to operate.**
- **The OS provides a uniform interface that hides these details so the programmers can access such devices using simple read and write commands.,**