

CSE 2421

Computer Systems


- Required Reading: *Computer Systems: A Programmer's Perspective, 3rd Edition*
 - Chapter 1, Sections 1.4 through 1.7.4, Section 1.9.3
 - Chapter 6 thru 6.1.3
 - Chapter 8, Section 8.2 through 8.2.4

What is a system?

- ▶ “A collection of intertwined **hardware** and **systems software** that must cooperate in order to achieve the ultimate goal of running application programs”

Computer hardware

The hardware in a digital computer system must perform a number of basic operations:

1. Data storage: storing digital data in a form which balances considerations of cost, reliability, and access speed.
 2. Data movement: moving data from one system component to another, or within a given component.
 3. Data transformation: changing a given bit pattern to a related one.
- 

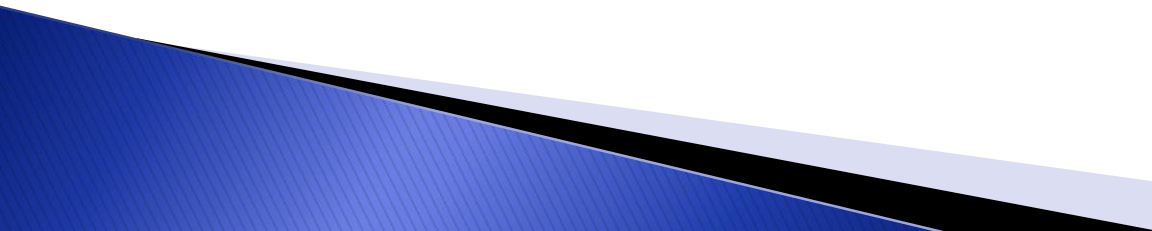
Data storage

1. Solid-state data storage uses state elements – latches or flip-flops
2. Two different kinds of logic elements:
 - a. Combinational – do not store state: current output depends *only* on current input.
 - b. Sequential (clocked) – capable of storing state: i.e., these elements have “memory” – they can remember or store a prior state.


Data movement

1. Data movement is done with buses – channels for transferring data from one system component to another (or within a component). The buses consist of “wires,” or conductive circuit traces (e.g., copper traces on a circuit board, or doped conductive silicon patterns in a processor, such as a CPU)
2. The system memory bus is divided into an **address bus** and a **data bus**, as well as some **control lines** (read line, write line, memory read complete line).
3. The CPU has two registers which it uses for memory accesses:
 - a. MAR (or Memory Address Register): to put an address on the address bus;
 - b. MDR (or Memory Data Register): to put data on the data bus, or to fetch data from the data bus.

CPU and Main Memory – 8 Byte Reads

- ▶ When the CPU wants to read eight bytes of data from memory, it puts the address of the first byte to be read in the MAR, and sets the read line to 1.
 - ▶ When the memory unit detects that the read line is set to 1, it gets the address off of the address bus, gets the eight bytes at that address from memory, and puts them on the data bus, and sets the memory read complete line to 1.
 - ▶ The 8 bytes from the data bus are transferred to the MDR when the CPU detects that the memory read complete line is 1.
- 

CPU and Main Memory – 8 Byte Writes

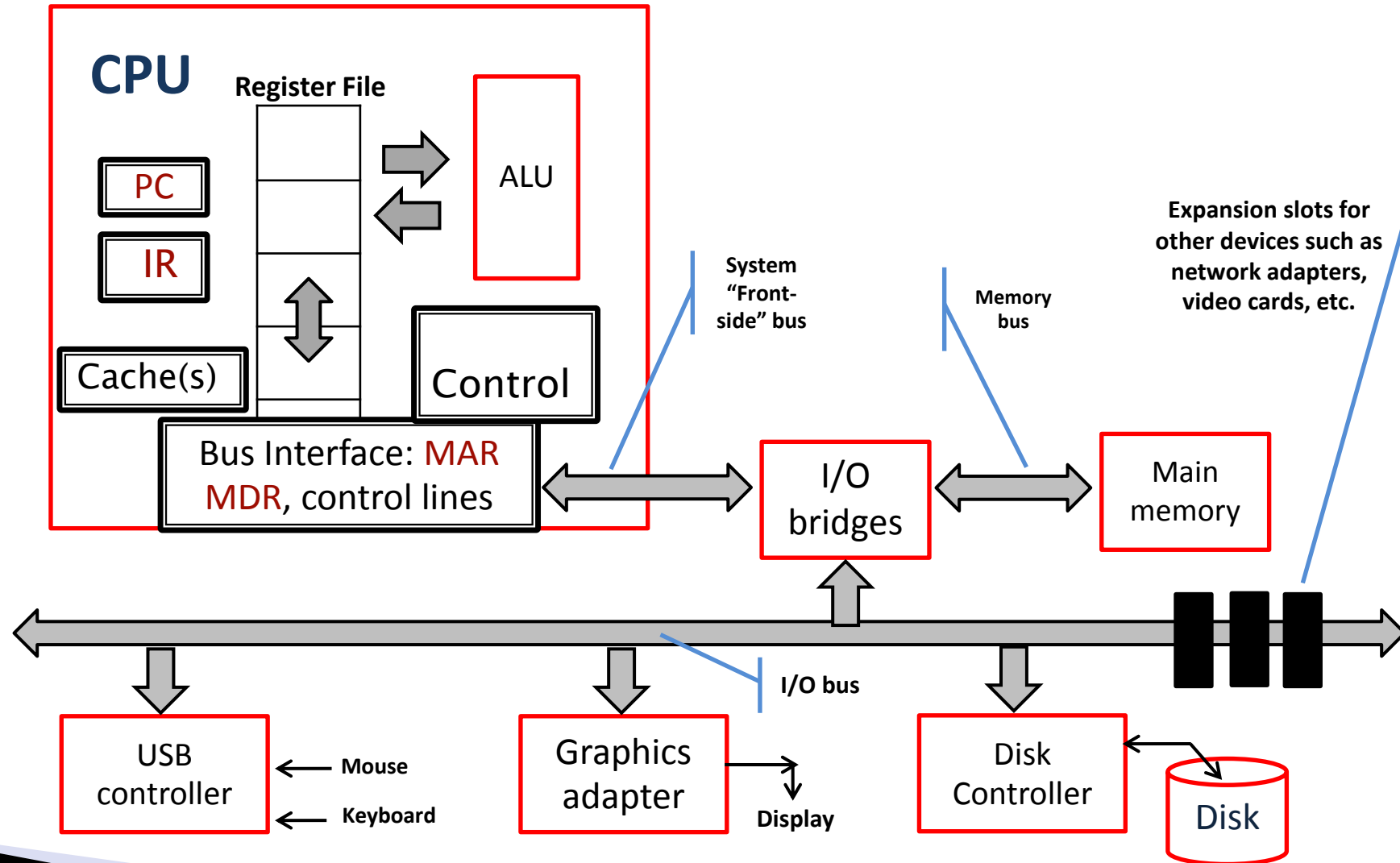
- ▶ When the CPU wants to write data to memory, it puts the address of the first byte to be written in the MAR, puts the data to be written in MDR, and sets the write line to 1.
 - ▶ When the memory unit detects that the write line is set to 1, it gets the address off of the address bus, gets the eight bytes to be written from the data bus, and writes them to memory at the address it got from the address bus.
- 

Data transformation

Data transformation, for example, in the CPU, is done with a combination of:

- a. state elements, which store input(s) and output of the operation, plus
- b. combinational circuits which actually perform the transformation (modify the appropriate bits in the input to produce the output).

Hardware Organization (big picture)



Instruction Set Architecture

- Instruction Set Architecture (ISA) – an **abstraction** which specifies
 - The processor state
 - The format of the instructions
 - The effect each instruction will have on the processor state
 - PC's use x86 (mostly, X86-64 these days)
- The underlying hardware can do anything as long as it *transparently* maintains the abstraction above. (This doesn't always happen)

CPU components

Registers (not directly accessible by user programs):

Note: All registers described in the following slides are 64 bits in 64 bit processors, unless otherwise noted.

a. **PC (Program Counter)**: This holds the address of the current instruction. It will be incremented to point to the next instruction as part of the execution of the current instruction.

b. **IR (or Instruction Register)**: A register where the bits of the instruction are stored in order for the instruction to be executed. Before execution, the bytes in memory pointed to by the PC will be brought into the IR, and then the instruction is executed.

CPU components

Register File: Organized as an array of registers, with each register accessed by an index (i.e. uniquely named); X86-64 has 16 integer registers

ALU: Combinational logic to implement all of the Arithmetic and Logical Instructions which the CPU can perform; execution of these instructions sets some or all of the flags or condition codes. This computes new data and address values.

Control: Logic circuitry to control instruction fetching and execution.

EFLAGS register: A register in processors with one bit flags set by ALU instructions.

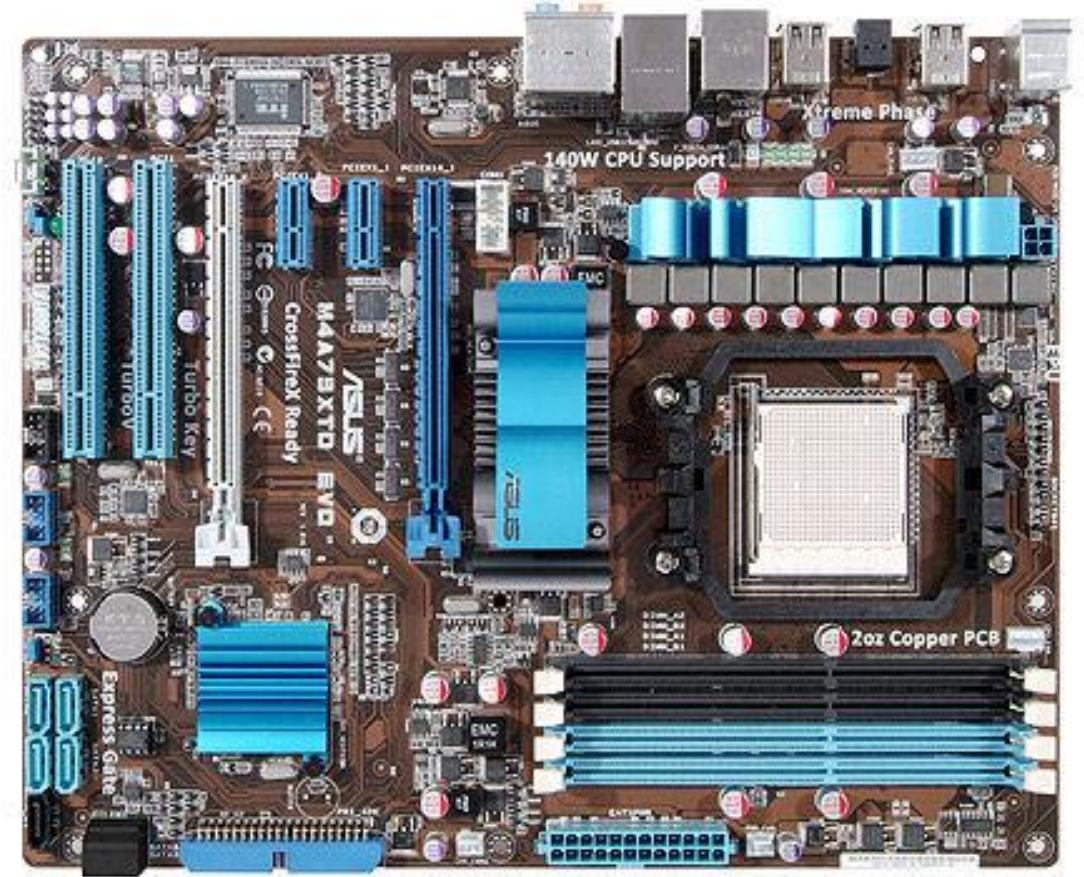


CPU components

MAR: A 64 bit register the CPU uses to put addresses for memory accesses on the address bus.

MDR: A 64 bit register which the CPU uses to put data on the data bus for writes, or to retrieve data from the data bus for reads from memory.

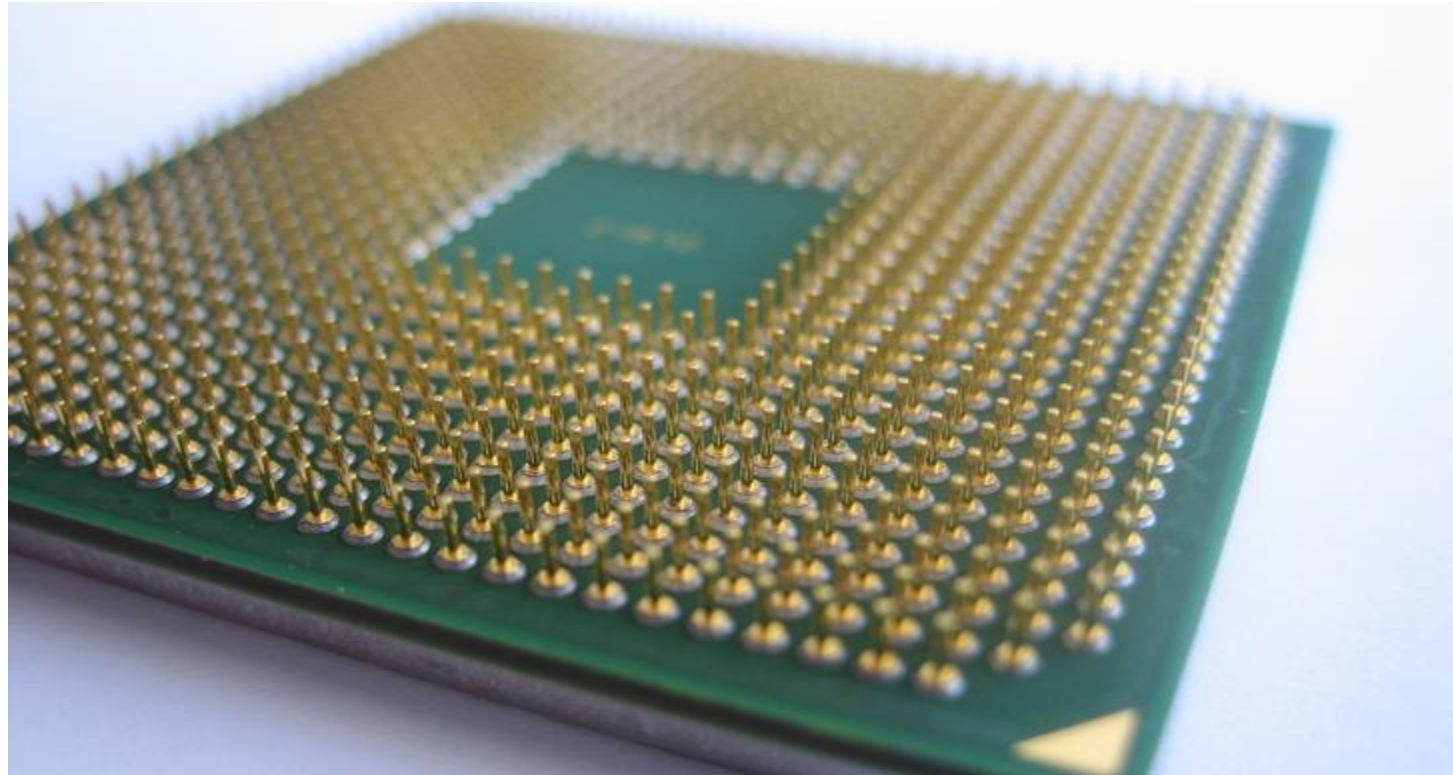
Motherboard or mainboard
(desktop system – server mainboards are similar)



Back panel input/output connectors




CPU package and pins



DIMM memory module




Bus

- Electrical conduit used to carry bytes of information between hardware components
 - Typically transfer fixed-size chunks known as a *word*
 - Size of a *word* depends on the “bitness” of the machine
 - 32 bit systems (4 byte words typically)
 - 64 bit systems (8 byte words typically)
- 


I/O Devices

- Hardware devices connected to the outside world
- Typical I/O devices
 - keyboard
 - mouse
 - display
 - disk drive
- I/O devices are connected to the bus through either a *controller* or an *adapter*
 - Controller – chips that reside on the motherboard
 - Adapter – pluggable slot on the motherboard

Main Memory


- Temporary storage for applications which are being executed (i.e., instructions) and for data used by the applications
 - Organized as a linear array of bytes each with its own unique address starting at zero
 - Two types of memory technology (each with subtypes, which we will not look at)
 - SRAM (Static RAM)
 - DRAM (Dynamic RAM)
- 

SRAM

- Static RAM
 - Each bit is implemented with a six transistor circuit
 - Each bit will retain its current value or state indefinitely as long as it has power
 - Resistant to electrical noise
 - Faster access time than DRAM (i.e., faster to read and write)
- 

DRAM

Dynamic RAM

- Each bit is stored on a capacitor and 1 transistor – very dense!
 - Retains value or state for only 10–100 milliseconds while power is applied
 - Modern CPU clocks tick in the sub-nanosecond range (A 4 GHz clock means 0.25 nanoseconds)
 - System will periodically (every 64ms) refresh every bit by reading then rewriting it
 - Very susceptible to electrical noise and subject to hardware attacks
 - Slower access time than SRAM (i.e., slower to read and write)
- 

SRAM vs DRAM comparison

	Transistors/ bit	Access Time	Persistent State	Electrically Sensitive	Relative Cost
SRAM	6	1x	yes	no	100x
DRAM	1	10x	no	yes	1x

So what is each typically used for?

- SRAM: CPU Cache (and other caches)
- DRAM: Main memory


Non-Volatile Memory

- Volatile memory will lose its information if the supply voltage is removed
 - Common volatile memory
 - RAM/caches
- Non-Volatile memory will retain its data when powered off
 - Common Non-volatile memory
 - EEPROM (electrically erasable programmable read only memory)
 - Flash
 - Solid State Disks (SSD)
 - (Magnetic) Disk Drives
 - Tape drives


Memory (Chapter 6)

- Slower memory is cheaper, so we have more of it
- Cache
 - ❑ Smaller and faster
 - ❑ Temporary staging areas
 - ❑ Make the transfer/copy operations happen asap. It's easier and cheaper to make processors run faster than it is to make main memory run faster; thus, main memory is the system bottleneck (and the CPU-main memory speed gap has grown over time)
- Memory Hierarchy
 - ❑ The storage at one level serves as a cache for storage at the next lower level
- The farther you are away from the action, the longer it takes to make something happen

Cache

- Caching is the concept of having smaller faster devices act as a “staging area” for larger slower devices (your pockets compared to your apartment)
 - Caching works because computer software exhibits the property of **locality** in memory access (see below).
 - The faster the memory, the more expensive it is to build, so there is a practical limit on how much fast memory we can put in the system and still keep it affordable
 - Develop a caching strategy so that the fastest memory can be kept small
 - A memory system with effective caching can perform data access at an average speed close to that of the fastest storage, but have an effective size and cost per byte close to that of the largest storage device
- 

The Memory Hierarchy

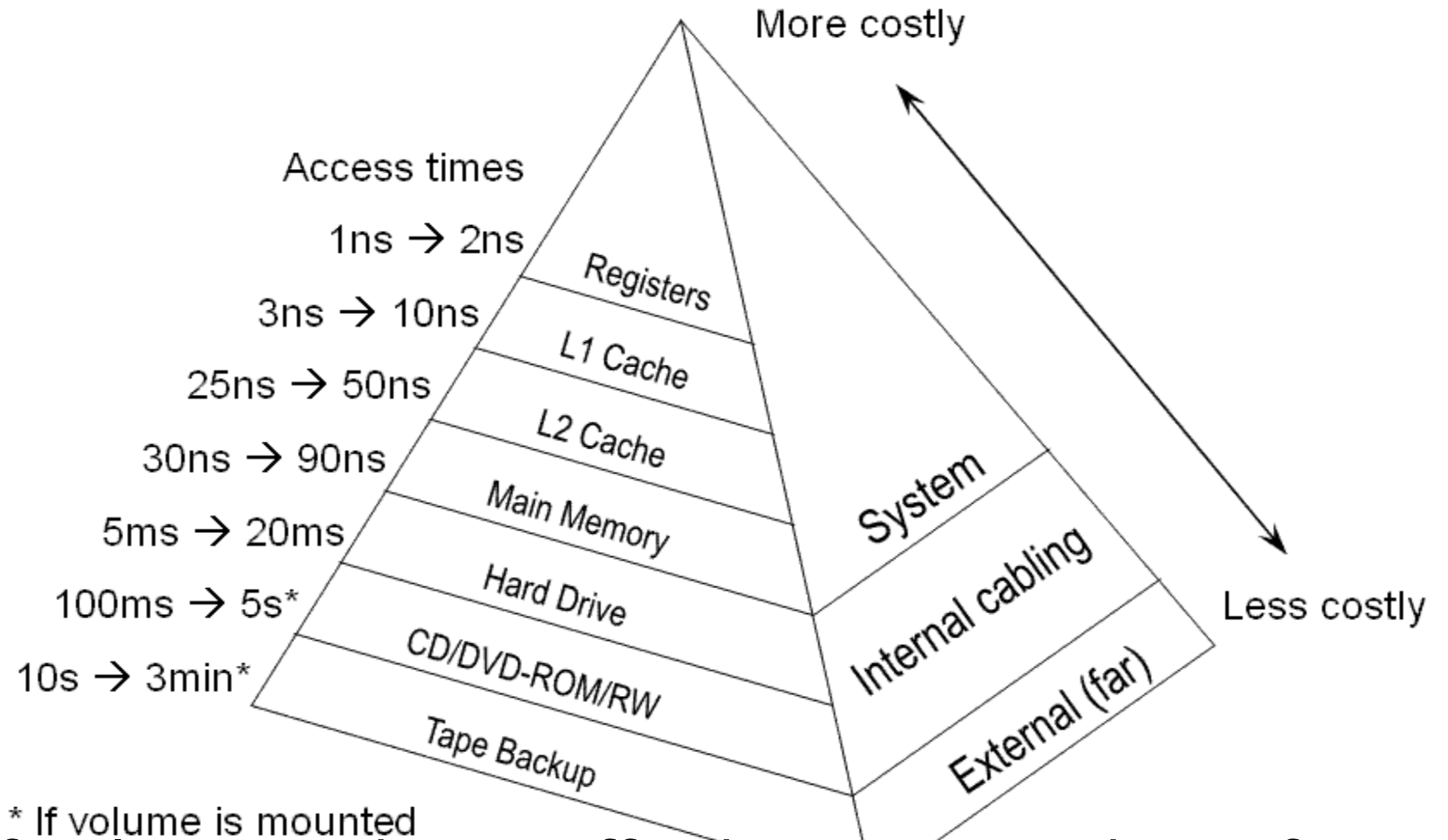
- ▶ This refers to the concept of building a memory subsystem which consists of various levels of storage.
 - ▶ The storage at higher levels of the hierarchy has faster access times, but also has a smaller size (because it is also more expensive)
 - ▶ Conversely, the storage at lower levels of the hierarchy has slower access times, and has a larger size (because it is also less expensive)
 - ▶ The storage at a higher level acts as a cache for the storage at the level below it.
 - ▶ Data transfers from a lower level to a higher level are done in “chunks” or groups; that is, when data is transferred, a number of items of data (instructions or data values) are transferred.
- 

The Memory Hierarchy cont.

- ▶ This results in a memory subsystem with a larger effective size and lower cost (cost per byte close to that of the least expensive storage), but with faster average access speed (close to that of the fastest storage), because programs exhibit locality of memory access.
- ▶ Two types of locality:
 - ***Spatial locality***. If a particular piece of data (or instruction) is accessed by a program, other data/instructions *at nearby locations* in memory are likely to be accessed soon.
 - ***Temporal locality***. If a particular piece of data (or instruction) is accessed by a program, the same piece of data or instruction is likely to be accessed *in the near future*.

Caching and the Memory Hierarchy –

These numbers are old but still informative



• http://faculty.etsu.edu/tarnoff/other2150/mem_hier.gif

Memory Hierarchy Speeds and Sizes

2013	Haswell	L1*	L2*	L3	RAM
	size	32 KB	256 KB	8MB	GB
	latency	4	12	36	~230

Times in clock ticks:
A 4.0 GHz system has a
0.25ns clock time

2017	Skylake-X	L1*	L2*	L3	RAM
	size	32+32 KB	1024 KB	11 MB	GB
	latency	4-5	14	68	~259

*per core

Why use tape??

- ▶ Cheap
- ▶ How often do you want to access it's data?
- ▶ Security. Really?
 - <https://www.wsj.com/articles/companies-look-to-an-old-technology-to-protect-against-new-threats-1505700180>
 - This link is from WSJ on 9/17/17. Hopefully it still works.
 - <https://wikibon.com/why-tape-is-poised-for-a-george-foreman-like-comeback/>

Storage Vendors

- HP Storage

<http://www8.hp.com/us/en/products/data-storage/product-portfolio.html>

- IBM Storage

<http://www-03.ibm.com/systems/storage>

- Dell Storage


- <http://www.dell.com/us/business/p/storage-products?~ck=anav>

- Oracle Storage

<https://www.oracle.com/storage/index.html>

IBM TS3500 Tape Library


Highlights

- ▶ Support highly-scalable, automated data retention on tape
 - ▶ Deliver extreme scalability and capacity
 - ▶ Provide up to 2.25 exabytes (EB) of automated, low-cost storage under a single library image
 - ▶ Enable data security and regulatory compliance via support for tape drive encryption and write-once-read-many (WORM) cartridges
 - ▶ Flexible library capacity with up to 192 drives in up to 16 TS3500 frames, and up to 15 libraries interconnected in a library complex
 - ▶ Simultaneous attachment of servers and applications to logical library partitions
 - ▶ Remote management with a web-based interface for library control and configuration
- 


How Big is a... ?

- ▶ **Bytes(8 Bits)**
- ▶ **Kilobyte (1 000 Bytes)**
- ▶ **Megabyte (1 000 000 Bytes)**
- ▶ **Gigabyte (1 000 000 000 Bytes)**
- ▶ **Terabyte (1 000 000 000 000 Bytes)**
 - 10 Terabytes: The printed collection of the US Library of Congress
 - 50 Terabytes: The contents of a large Mass Storage System
- ▶ **Petabyte (1 000 000 000 000 000 Bytes)**
 - 2 Petabytes: All US academic research libraries
 - 200 Petabytes: All printed material OR Production of digital magnetic tape in 1995
- ▶ **Exabyte (1 000 000 000 000 000 000 Bytes)**
 - 5 Exabytes: All words ever spoken by human beings.
- ▶ **Zettabyte (1 000 000 000 000 000 000 000 Bytes)**
- ▶ From [wikipedia](#):
 - Research from the [University of California, San Diego](#) reports that in 2008, Americans consumed 3.6 zettabytes of information.
 - [Internet Traffic to Reach 1.3 Zettabytes by 2016](#)
- ▶ **Yottabyte (1 000 000 000 000 000 000 000 000 Bytes)** [Named after Yoda.](#)
- ▶ **Xenottabyte (1 000 000 000 000 000 000 000 000 000 Bytes)**
- ▶ **Shilentnobyte (1 000 000 000 000 000 000 000 000 000 000 Bytes)**
- ▶ **Domegemegrottebyte (1 000 000 000 000 000 000 000 000 000 000 000 Bytes)**

Amdahl's Law (Section 1.9.1)

- Basically says that when we speed up one part of the system, the overall effect on system performance depends upon how significant the part sped up was to the overall system and how big the speed improvement was
 - i.e. to significantly speed up the entire system, the speed of a very large fraction of the overall system must be increased not just one piece
 - This law doesn't just apply to computer systems, but any process or system that we'd like to improve. (e.g. manufacturing processes, cooking a meal, etc.)
- 


Process Abstraction (Chapter 8)

- This abstraction is provided by the OS
 - A process is a **program *in execution*** (every program is capable of being a process, but is not one until it is actually executing)
 - ✓ Allows a program to execute with the view that it is the only thing running (though it is virtually never the only thing running)
 - ✓ Application does not have to worry about sharing CPU time, or memory space (even though it virtually always actually is sharing those resources)
 - ✓ Each process in the system is managed by the OS using a **process control block (PCB)**.
- 

Process Abstraction cont.

- The PCB has information about **the current state** of the process (*ready, running, blocked*), and data that the OS uses to restore the process (PC and register values) to its previous state in order to resume the process after it goes from a blocked state (for input or output, for example) to a running state.
- The process abstraction allows the CPU to be utilized more fully.
 - It increase throughput (the number of processes which can complete per unit time) at the cost of latency, or a delay in completion time for each process.

File abstraction (Chapter 1, Section 1.9.3)

- ▶ Also provided by the OS
 - ▶ A file is just a collection of bytes sitting in some kind of storage.
 - ▶ Opening a file, or executing the program contained in an executable file, copies the bytes from disk into RAM.
 - ▶ The interpretation of the bytes (instructions, numerical data, characters, etc.) is determined by **the software** which is used to open the file.
- 

Summary

- ▶ A computer system is a mixture of hardware and systems software that cooperate to run application programs
 - ▶ Information in a computer is represented as groups of bits that are interpreted in different ways, depending upon the context.
 - ▶ Make a point to read Section 1.10
- 