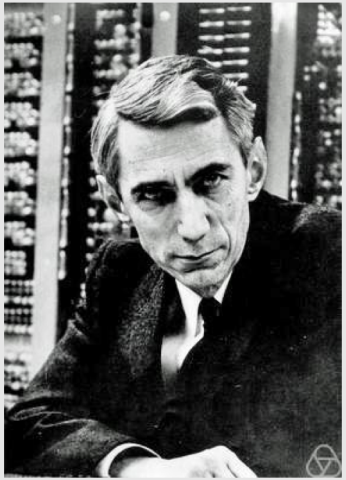
Digital Circuits

An introduction to circuits and data storage

Agenda

- Digital circuits
  - Logic gates
  - Circuits and blocks
- Computer storage
  - Types of memory
  - Memory hierarchy
  - Locality

Logic gates



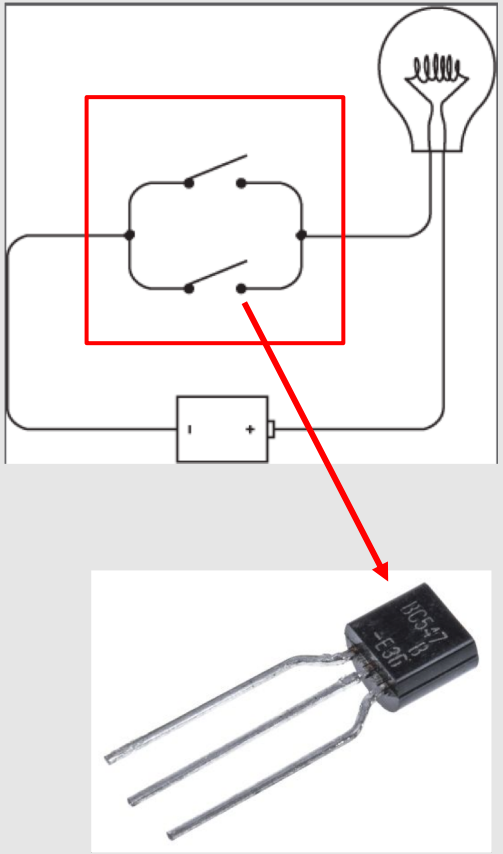
Claude Elwood Shannon, 1916-2001.
“The father of information theory”

Logic gates

Logic gates are simple digital circuits that

- take one, two, or more binary inputs
- produce a binary output

Boolean operators can be represented by logic gates.

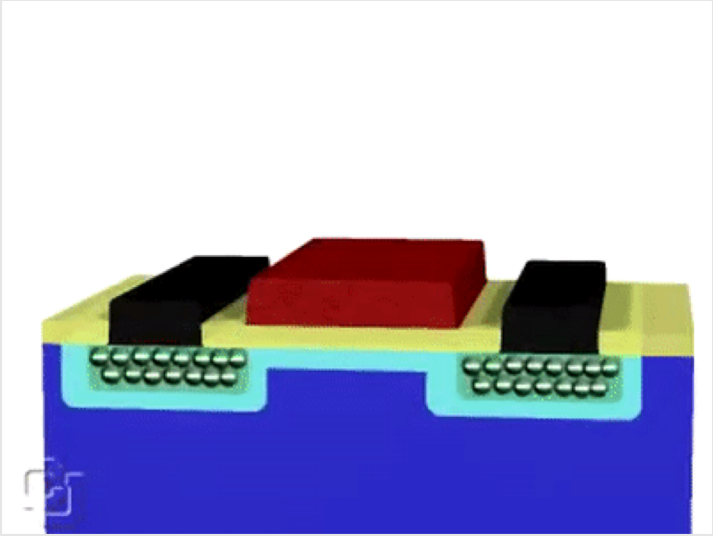


In today's computers, switches are usually constructed out of transistors.

Transistors

We can control the states of the switch by the voltage.

- High voltage: switch is on
- Low voltage: switch in off.

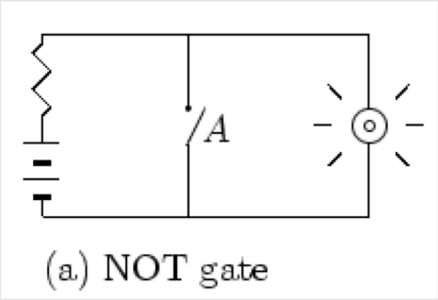


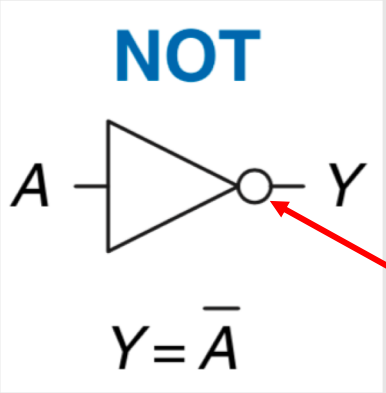
If you would like to know more about transistors, you may watch this video.

NOT gate

A NOT gate has one input, A, and one output, Y.

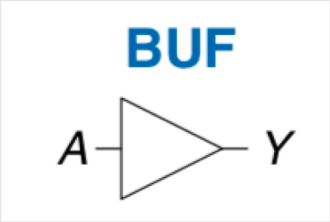
- The output is the inverse of its input.
  - If A is False, then Y is True.
  - If A is True, then Y is False.





A	Y
0	1
1	0

This circle means “not/inversion”.



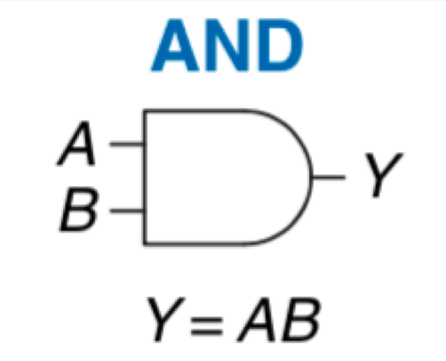
A	Y
0	0
1	1

Without the circle, the gate is called buffer, which simply copies the input to the output.

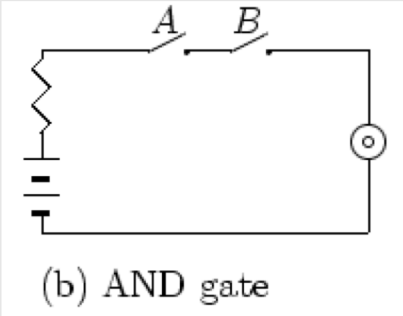
AND gate

The AND gate has two inputs, A and B, and one output, Y.

- It produces a True output Y, if and only if both A and B are True.
- Otherwise, the output is False.



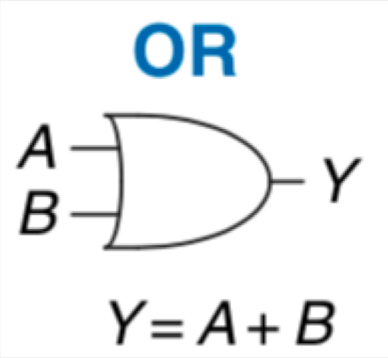
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



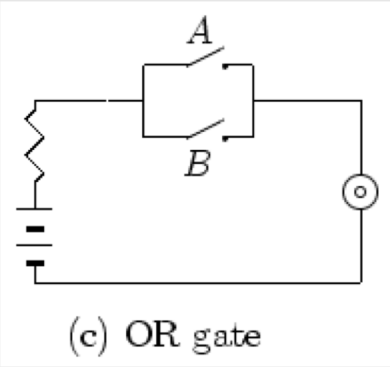
OR gate

The OR gate has two inputs A and B, and one output Y.

- It produces a True output Y, if either A or B (or both) are True.
- Otherwise, Y is False.



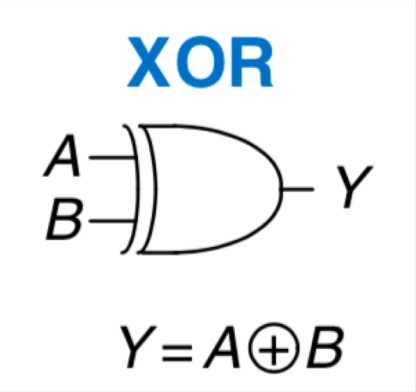
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



XOR gate

The XOR gate has two outputs A and B, and one output Y. (pronounced as exclusive OR)

- Y is True if A or B, but not both, are True.

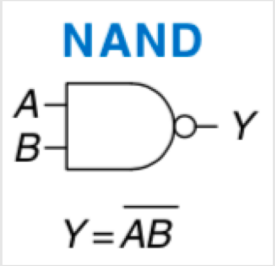


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

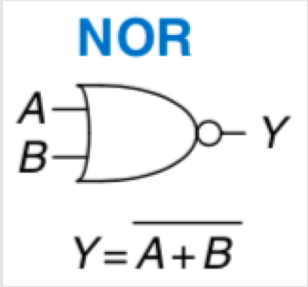
Other Two-input gates

Any gate can be followed by a bubble to invert its operation.

- The NAND gate performs NOT AND.
- The NOR gate performs NOT OR.

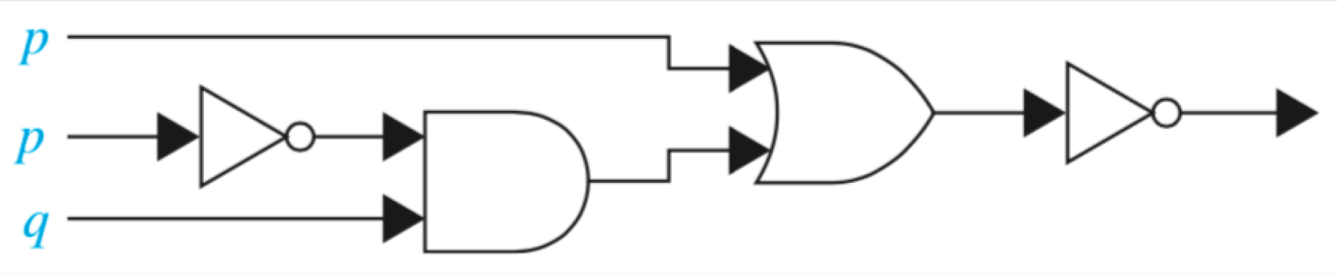


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Find the output of the circuits



Can you tell what is the output of this circuit?

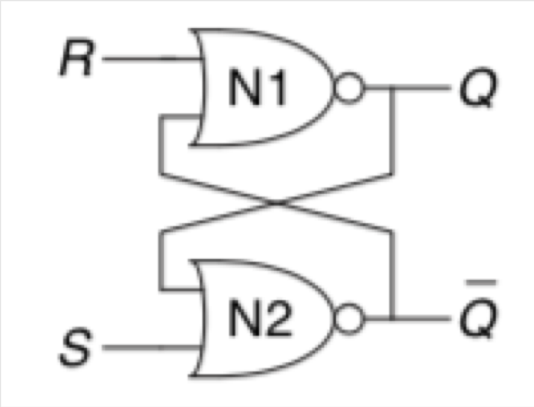
$$\neg (p + (\neg p \times q))$$

SR Latch

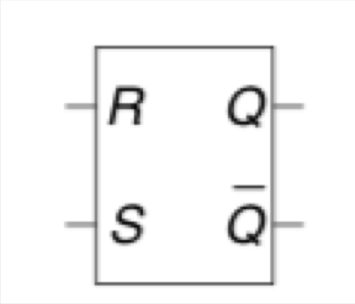
SR latch is composed of two cross-coupled NOR gates.

- It has two inputs S and R which refer to set and reset respectively.
- One input of N1 is the output of N2 and vice versa. (referring to the schematic)

SR latch schematic



SR latch symbol



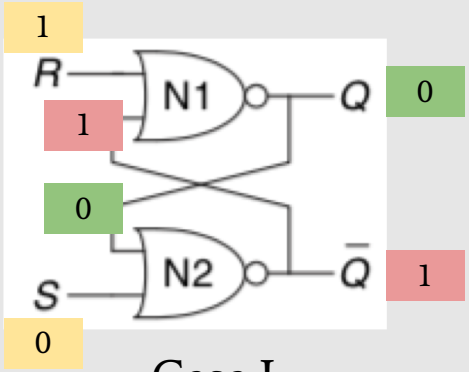
Truth table of SR latch

There are four possible cases of the inputs.

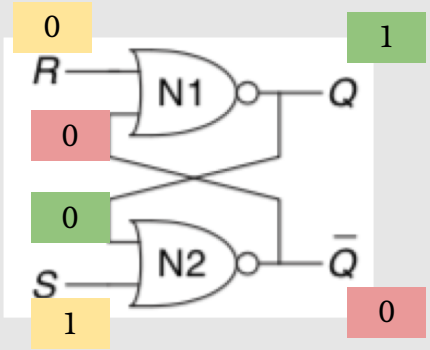
Case	S	R	Q	¬Q
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0
IV	0	0	Q_{PREV}	$\neg Q_{PREV}$

This circuit has memory.

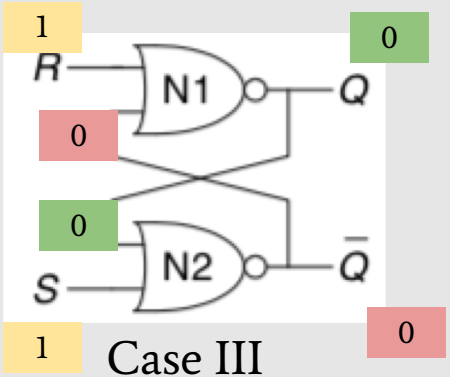
- If there are no inputs (R=0, S=0), Q will keep the value before we enter Case IV.
- If we reset (R=1), Q = 0.
- If we set (R=0, S=1), Q = 1.



Case I

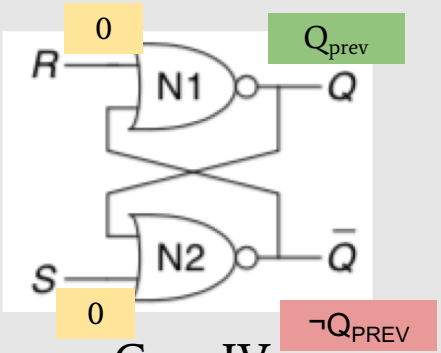


Case II



Case III

Invalid output



Case IV

(see next slides)

Truth table of SR latch

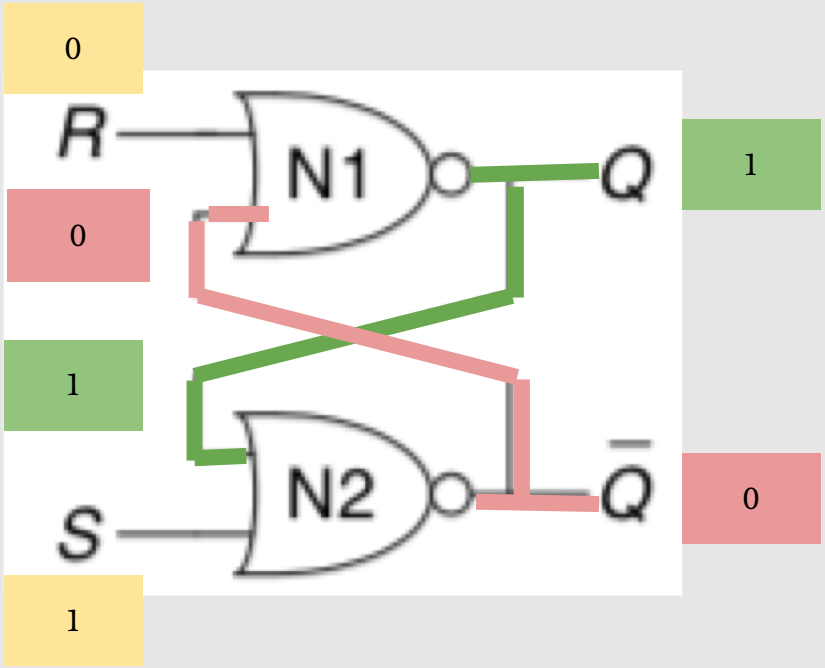
There are four possible cases of the inputs.

Case	S	R	Q	¬Q
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0
IV	0	0	Q_{PREV}	$\neg Q_{PREV}$

This circuit has memory.

- If there are no inputs (R=0, S=0), Q will keep the value before we enter Case IV.
- If we reset (R=1), Q = 0.
- If we set (R=0, S=1), Q = 1.

Set: S=1



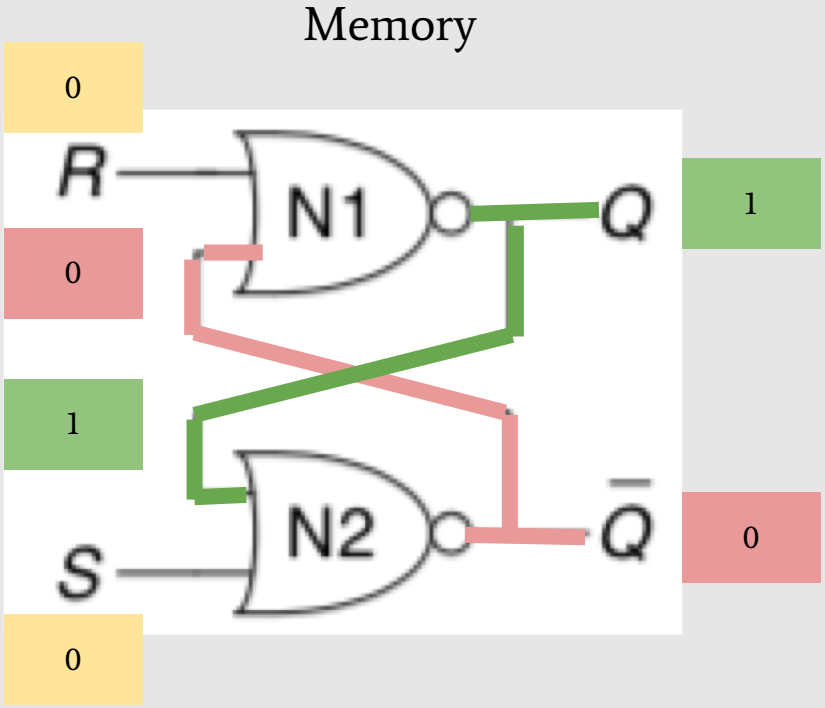
Truth table of SR latch

There are four possible cases of the inputs.

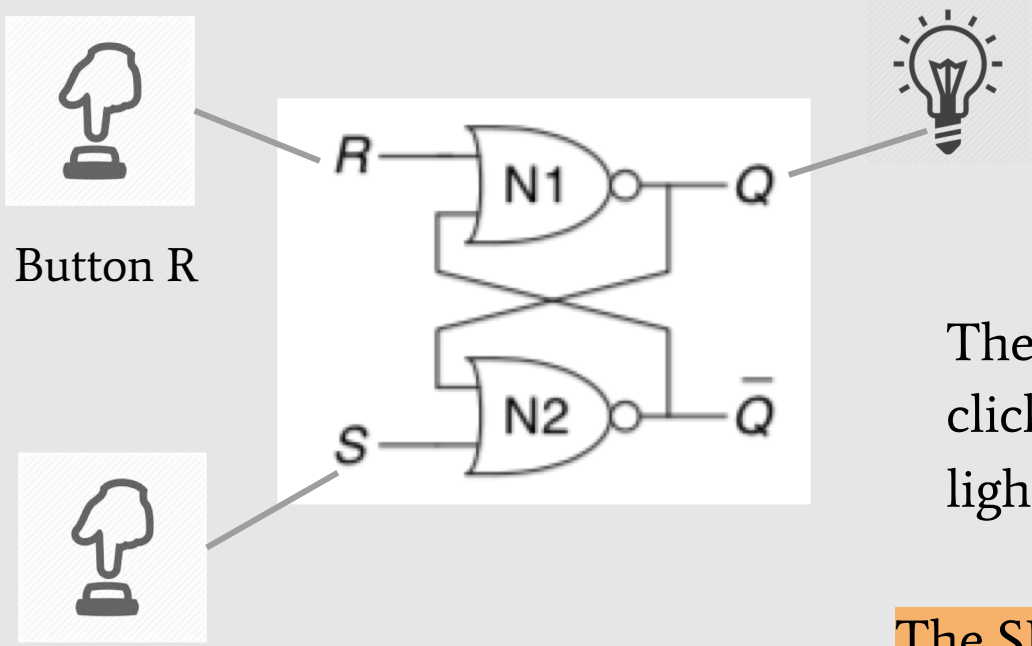
Case	S	R	Q	¬Q
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0
IV	0	0	Q_{PREV}	$\neg Q_{PREV}$

This circuit has memory.

- If there are no inputs (R=0, S=0), Q will keep the value before we enter Case IV.
- If we reset (R=1), Q = 0.
- If we set (R=0, S=1), Q = 1.



What remember means

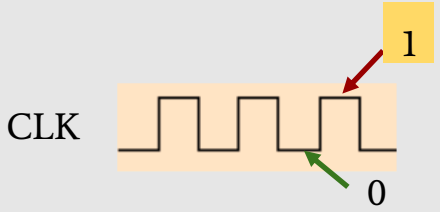


The bulb will light when Button S is clicked once. And, it will keep on lighting until Button R is clicked.

The SR Latch stores the **last** state of the input.

The clock signal

A clock signal is a signal that swings between a high and a low state, and it is used to coordinate actions of digital circuits.



Intel® Core™ i9-9900KS Processor (16M Cache, Up to 5.00 GHz)

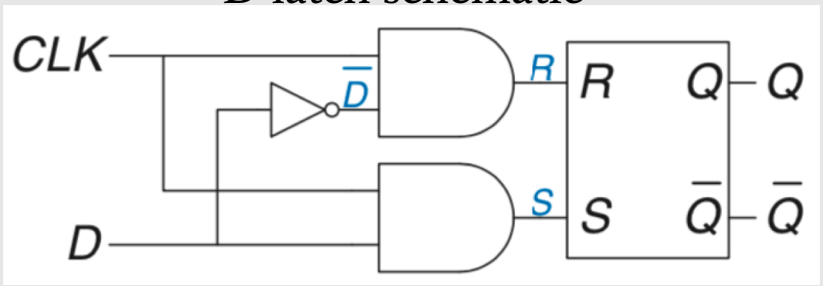
- 16 MB Intel® Smart Cache Cache
- 8 Cores
- 16 Threads
- 5.00 GHz Max Turbo Frequency
- 9th Generation

D Flip-flop

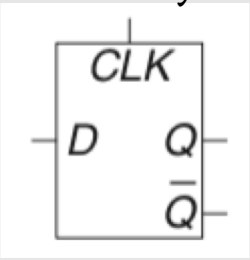
D latch is a modified SR latch.

- It has two inputs.
- The *data* input, D, controls the next state should be.
- The *clock* input, CLK, controls when the state should change.

D latch schematic



D latch symbol

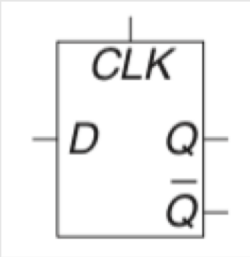


D Flip-flop

D latch is a modified SR latch with two inputs: D and CLK.

- D is the *data* input, , controls the state of the output Q.
- CLK is the *clock* input, , controls when Q should change.

D latch symbol



Truth table of D latch

Case	CLK	D	Q
I	0	X	$Q_{previous}$
II	1	0	0
III	1	1	1

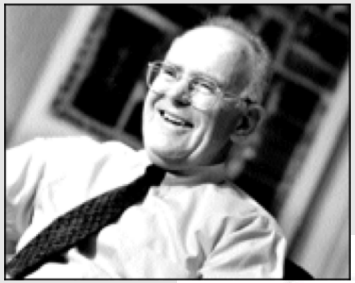
When CLK = 0, the latch is opaque; whatever D is, Q keeps its previous state.

When CLK = 1, the latch is transparent; Q is the same to D.

BBC History of Computers

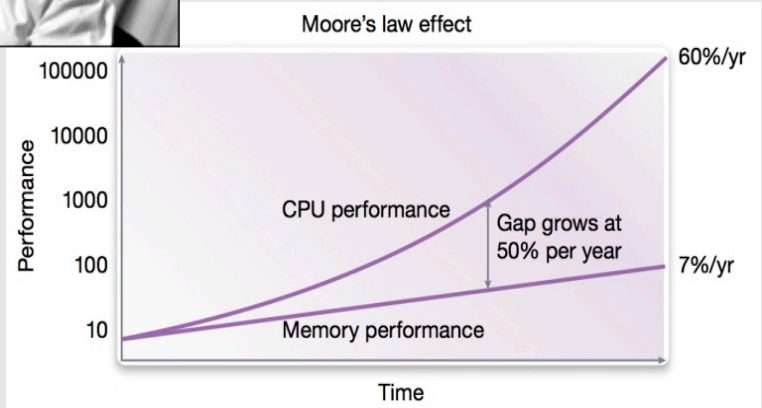
Clip #5

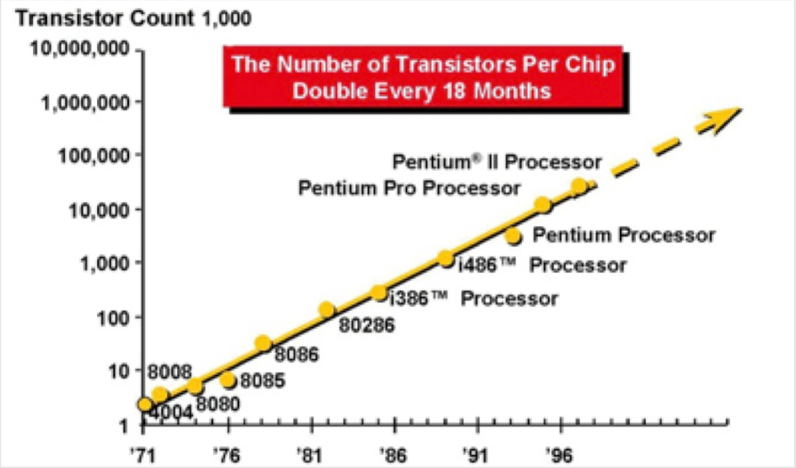
The Moore's law



From careful observation of an emerging trend, Moore extrapolated that computing would dramatically increase in power, and decrease in relative cost, at an exponential pace.

Number of transistors on cost-effective integrated circuit double every 18 months.



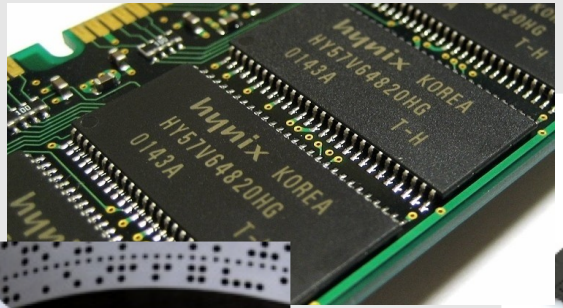


However, many people, including Gordon Moore, expect Moore's law will end by around 2025.

- Transistors eventually would reach the limits of miniaturization at atomic level.

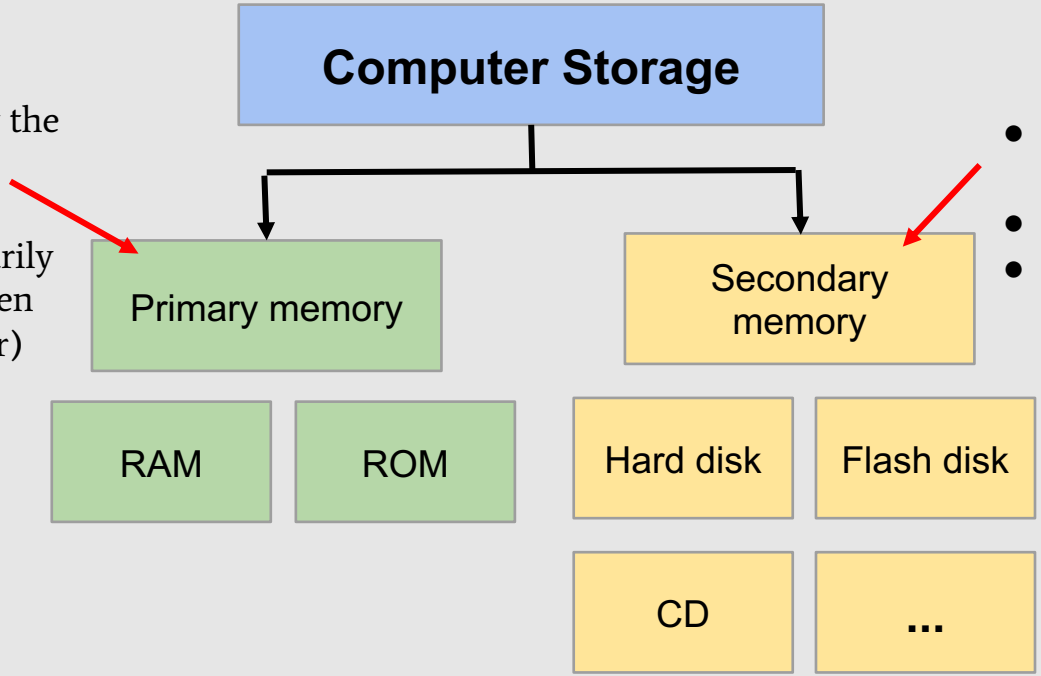
Computer Storage

Computer Storage

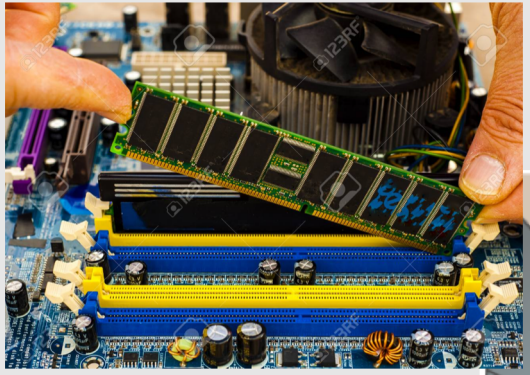


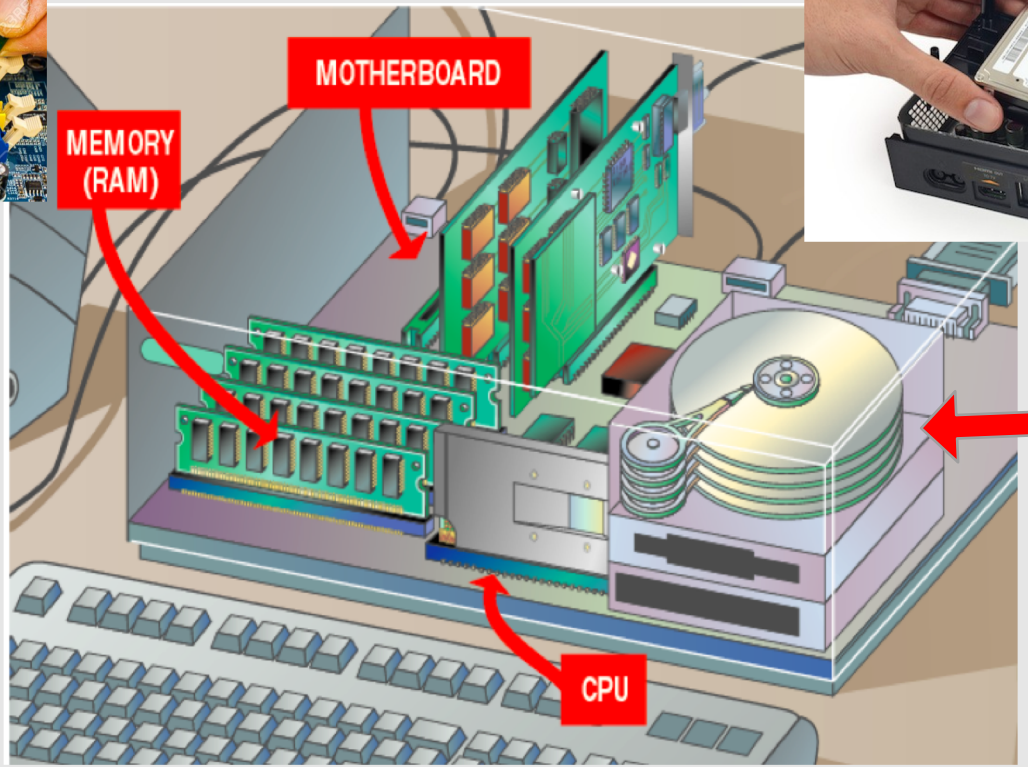
Storage types

- Immediate access by the processor
- Very fast
- Data stored temporarily (RAM loses data when cutting off the power)



- Not directly accessible by the CPU
- Slow
- Data stored permanently

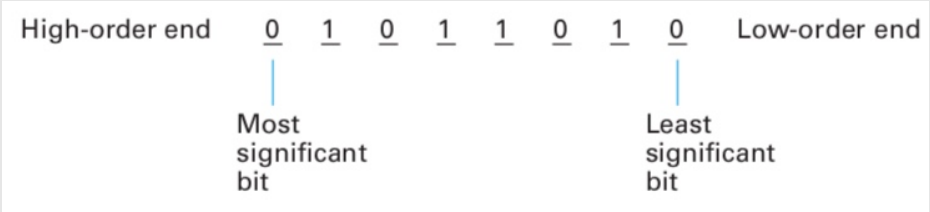




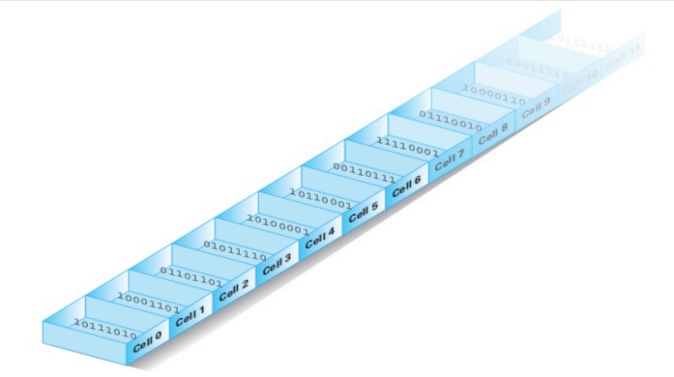
Hard Drive

Memory

A computer's main memory is organized in basic units called **cells,** with a typical cell size being eight bits.

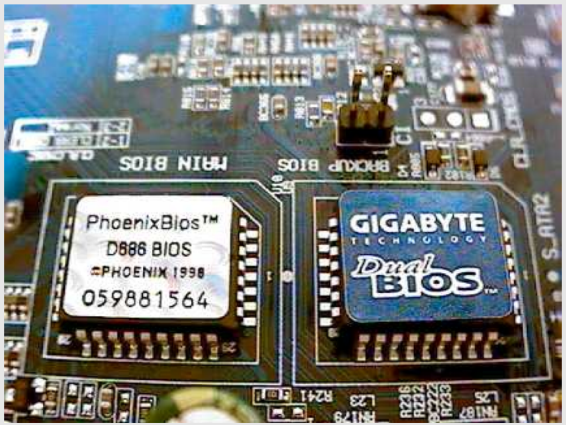


Each memory cell is assigned a unique "name," called its **address.**



Memory

- ROM (Read only memory):
  - Data can only be accessed and read, no overwritten, or modified (today some ROM can be modified)
  - Nonvolatile, a permanent storage of data (regardless of power supply)
  - Used as bootable devices (e.g., BIOS)



Memory

RAM – Random Access Memory

- The contents of RAM **can be altered** so a computer can both **read** from and **write** to memory addresses in RAM.
- RAM is **volatile** meaning that if the power is switched off or the battery removed then the contents will be **lost**.

When a personal computer is in use the following are copied into RAM from the backing storage:

- The Operating System (OS)
- All the other programs that are running
- Any data files that are in use.

Memory

Various unit to measure memory

- Bit -- smallest unit of computer memory
- Byte -- 1 byte = 8 bits
- Kilobyte -- 1 kb = 1024 bytes
- Megabyte -- 1 mb =1024 kb
- Gigabyte --1 gb = 1024 mb
- Terabyte -- 1 tb = 1024 gb

Memory

- DRAM(Dynamic random access memory)
  - DRAM is implemented with a transistor and a capacitor for storing a bit of data
  - Dynamic: must be refreshed periodically
  - Volatile: loses data when power is off

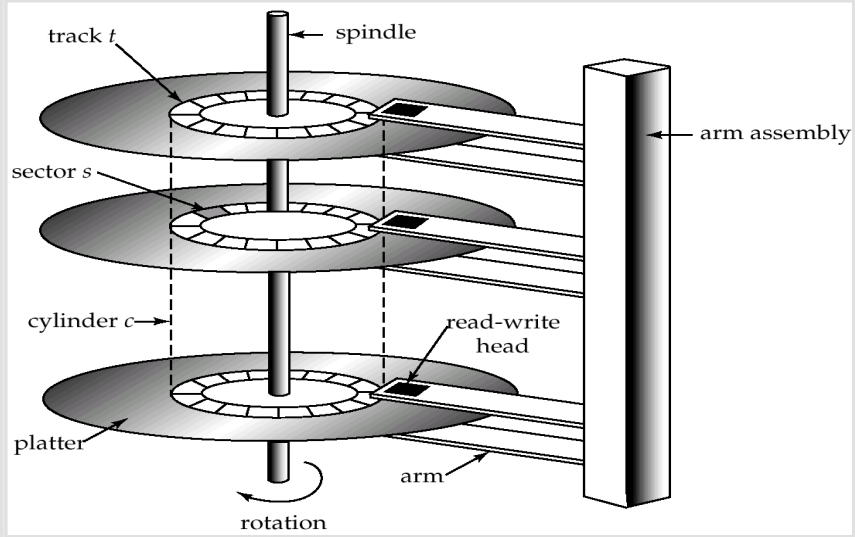
- SRAM (Static random access memory)
  - SRAM uses latches to store data, which is composed of 6 transistors.
  - Static: holds data as long as power is applied
  - Volatile: can not hold data when power is off

DRAM is smaller and less expensive per bit. DRAM consume more power. SRAM is more costly and way faster than DRAM.

Mass Storage -- Magnetic disk

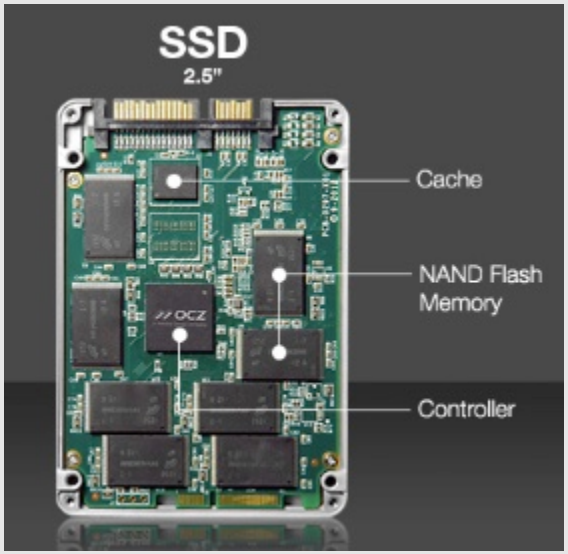
Hard Disk Drive (HDD)

- Magnetic coating on platter to store the data
- Data is read/written by moving the magnetic head
- Random access needs to move the read-write head: slow
- Sequential access afterwards is fast



Mass Storage -- Flash drive and SSD

- Flash memory stores information by sending electronic signals to the storage medium and electrons are trapped within the isolator(tiny chamber of silicon dioxide). Data will be retained after power is off.
- SSD (Solid state drive) use NAND flash memory to store data.



Comparison

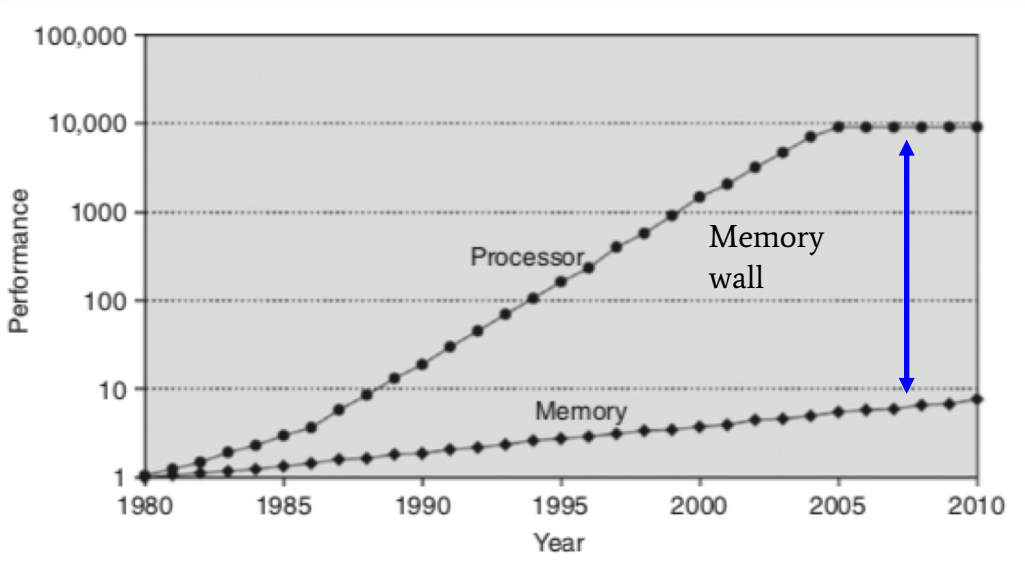
	Memory	Disk
Capacity	~GB	~TB (x1000 bigger)
Price per Giga byte	~8$	~0.04$ (x200 cheaper)
Access time	~100ns	~10ms (x10K slower)
Access pattern	Random	Random (but read a big chunk is much faster)
Durability	Volatile: gone if powered off	Nonvolatile

Memory hierarchy

Memory wall: speed gap between CPU and memory

"Memory wall" is the growing disparity of speed between CPU and memory outside the CPU chip.

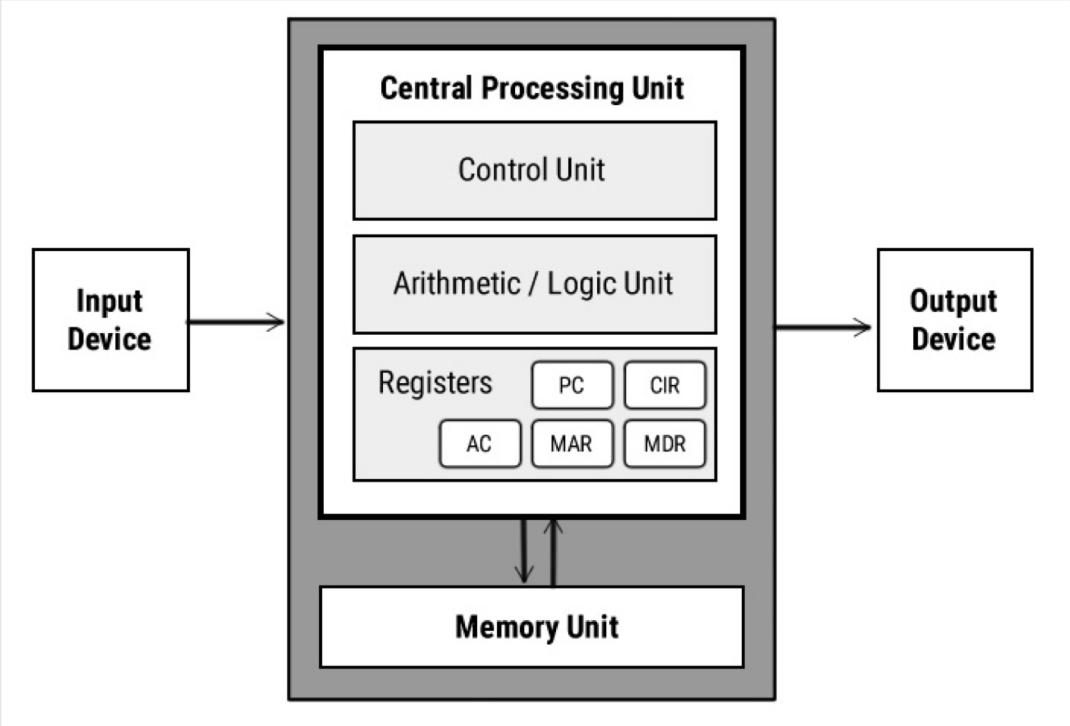
- It is the bottleneck that slows down the speed of computers.



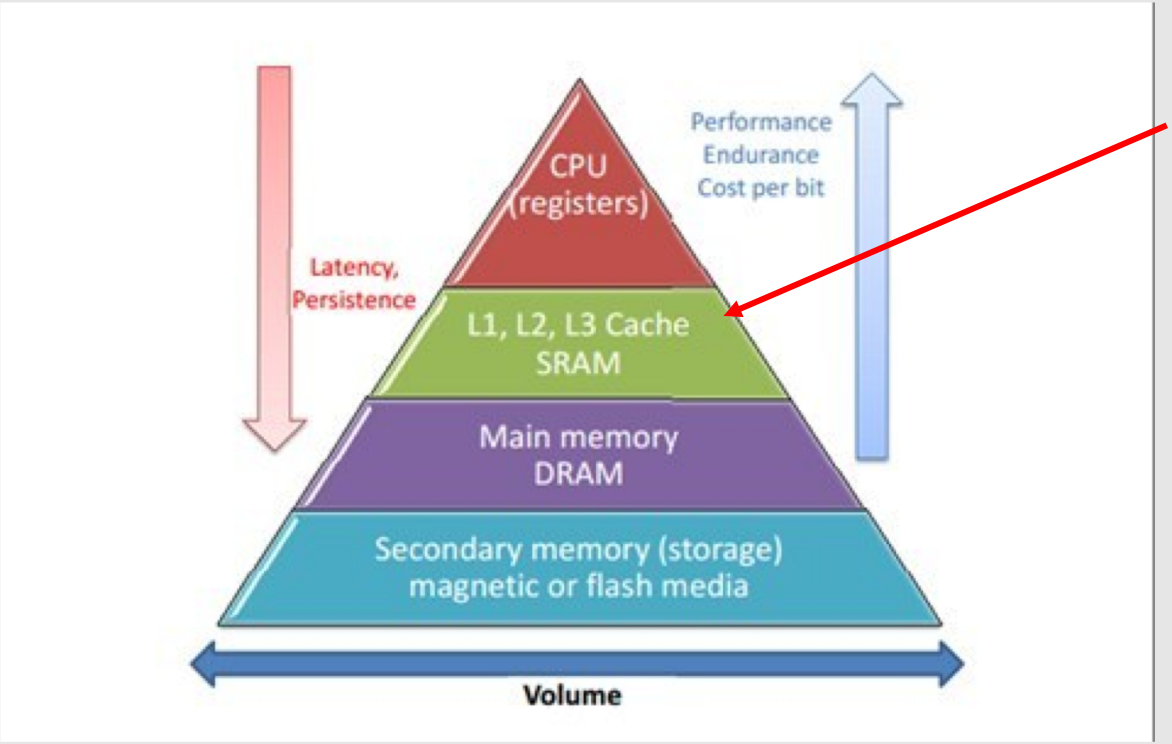
Diverging processor and memory performance: DRAM speed has improved 7%/year, whereas processor performance has improved at a rate of 25% to 50%/year.

CPU (Central processing unit)

Registers: To store data and computer instructions.



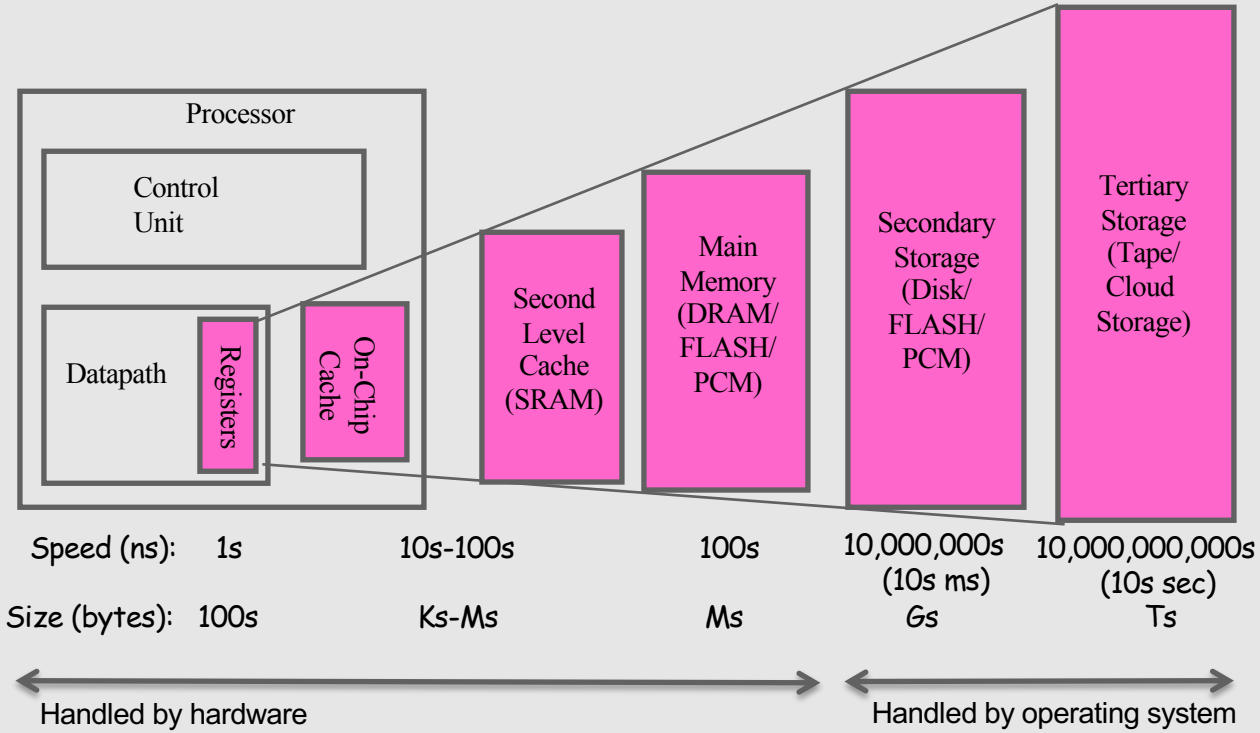
Memory hierarchy



Cache: a faster but smaller memory that built out of SRAM on the same chip as the processor.

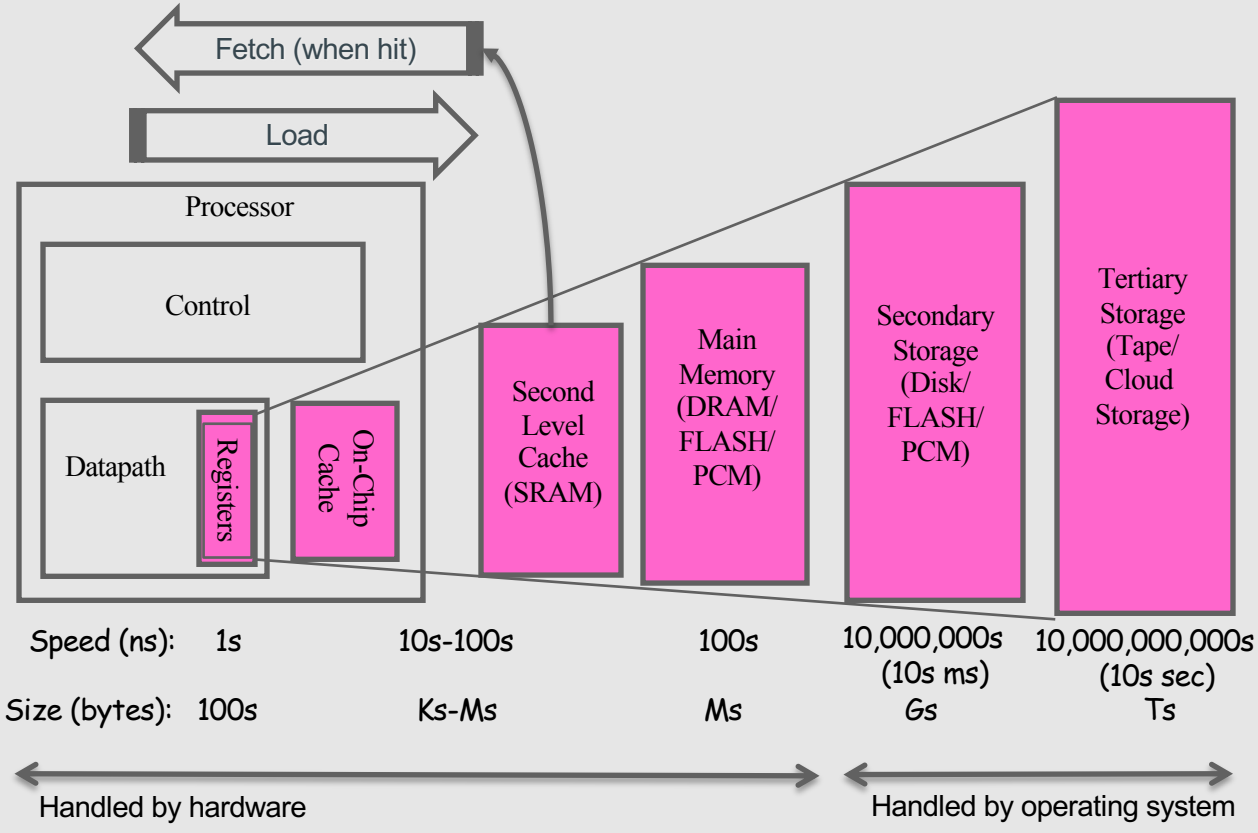
- Loading data used in near future into cache in advance.

The Memory Hierarchy



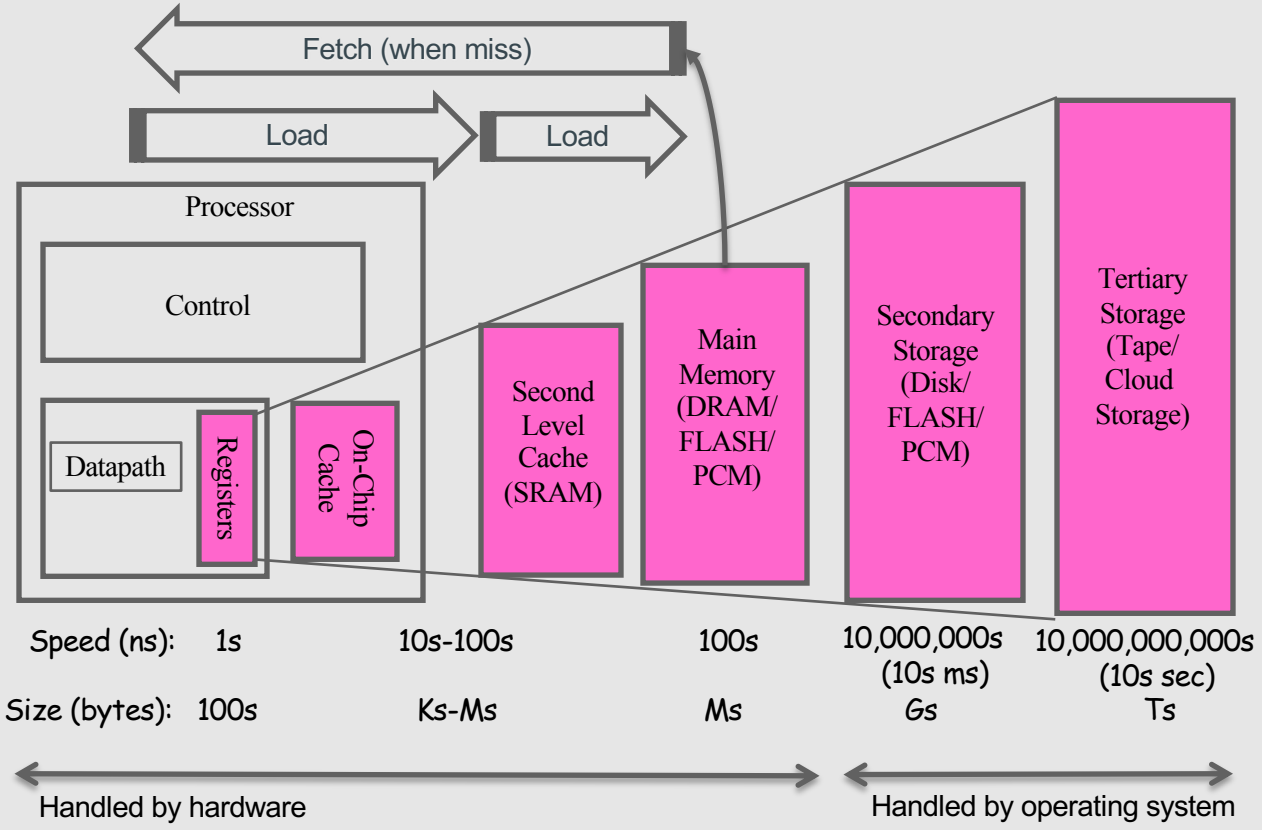
Data is transferred between memory and cache in blocks of fixed size, called *cache lines* or *cache blocks*. When a cache line is copied from memory into the cache, a cache entry (the copied data as well as the requested memory location) is created.

The Memory Hierarchy

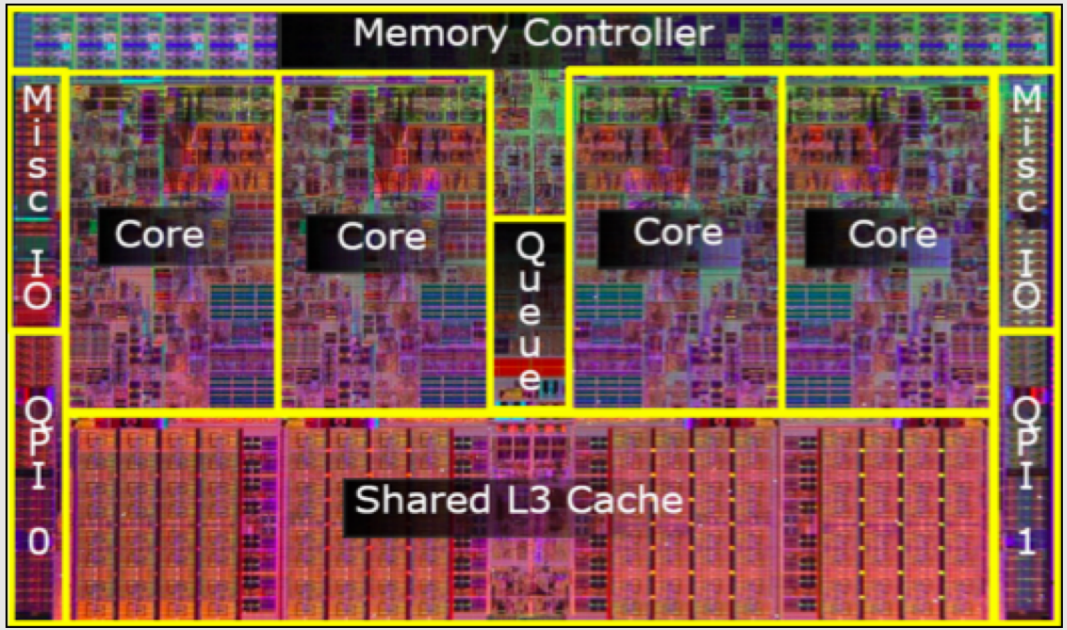


The Memory Hierarchy

Goal:
- As much as memory as the **cheapest** technology
- Access at speed of the **fastest** technology



Example of modern core: Nehalem



- ON-chip cache resources:
  - For each core: L1: 32K instruction and 32K data cache, L2: 1MB
  - L3: 8MB shared among all 4 cores
- Integrated, on-chip memory controller (DDR3)

Two principles of locality

- Temporal locality: recent accesses have shorter latency
- Spatial locality: nearby accesses have shorter latency

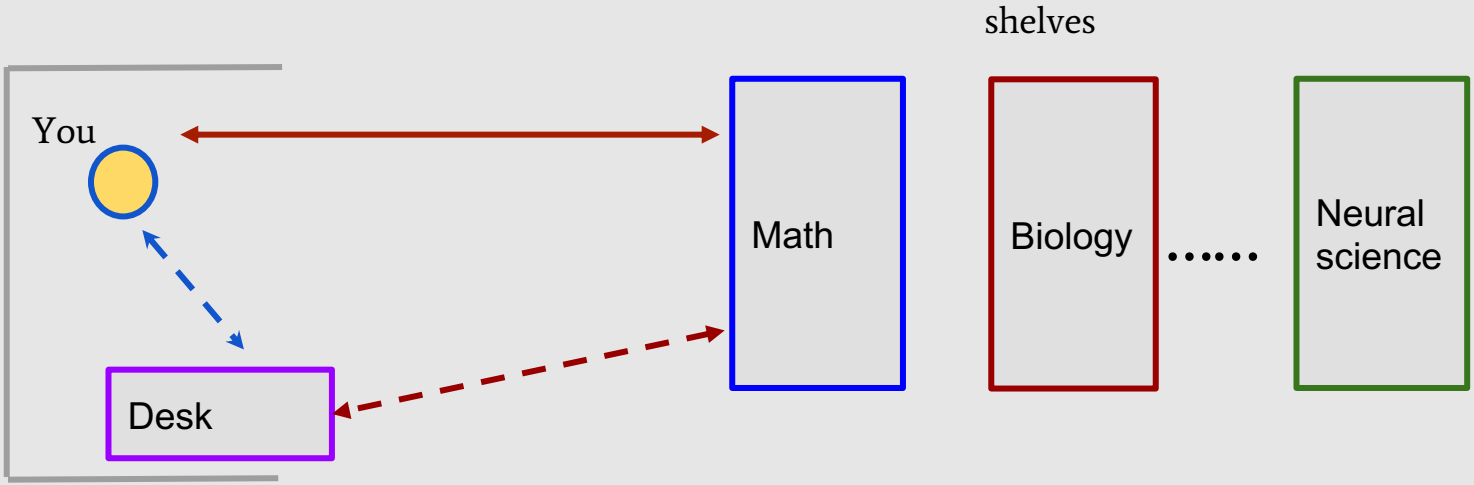
Your computer are designed to exploit these two forms of locality.

If a data item is recently used by a program
- it will be stored in the cache
- a few of its nearest neighbors will be loaded into cache

(Because they have a high probability to be used again in the next CPU clock cycle)

Your experience in a library

Assume you are writing a report on machine learning in the library:



- Temporal locality: if you have used a book recently, you are likely to use it again soon.
- Spatial locality: when you use one particular book, you are likely to be interested in other books on the same shelf.

Letting your code access locality

Hardware constraints that programmers need to know and take advantage of

- Temporal locality: recent accesses have shorter latency
  - Program tips: not all data is accessed equally often, keep those frequently access in inner loop

```
a = [x for x in range(0, 100)]
s = 0
for i in range(0, 100):
    s += a[i]
```

Variable S is referenced in each iteration, so there is good temporal locality with respect to S.

Letting your code access locality

Hardware constraints that programmers need to know and take advantage of

- Spatial locality: nearby accesses have shorter latency
  - Program tips: if you are accessing a big array (list), try to access them in order, avoid random access

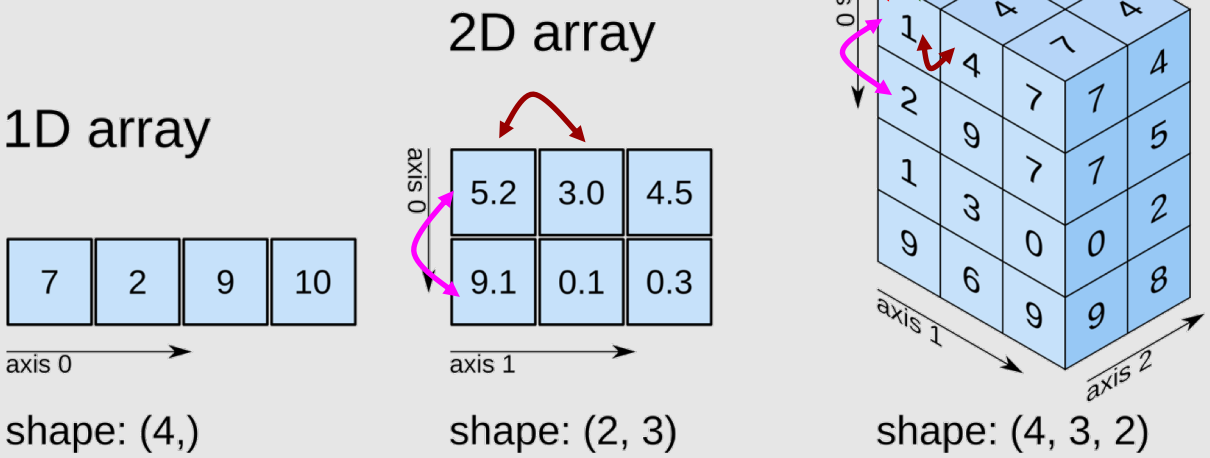
```
a = [x for x in range(0, 100)]
s = 0
for i in range(0, 100):
    s += a[i]
```

a is accessed by incrementally +1

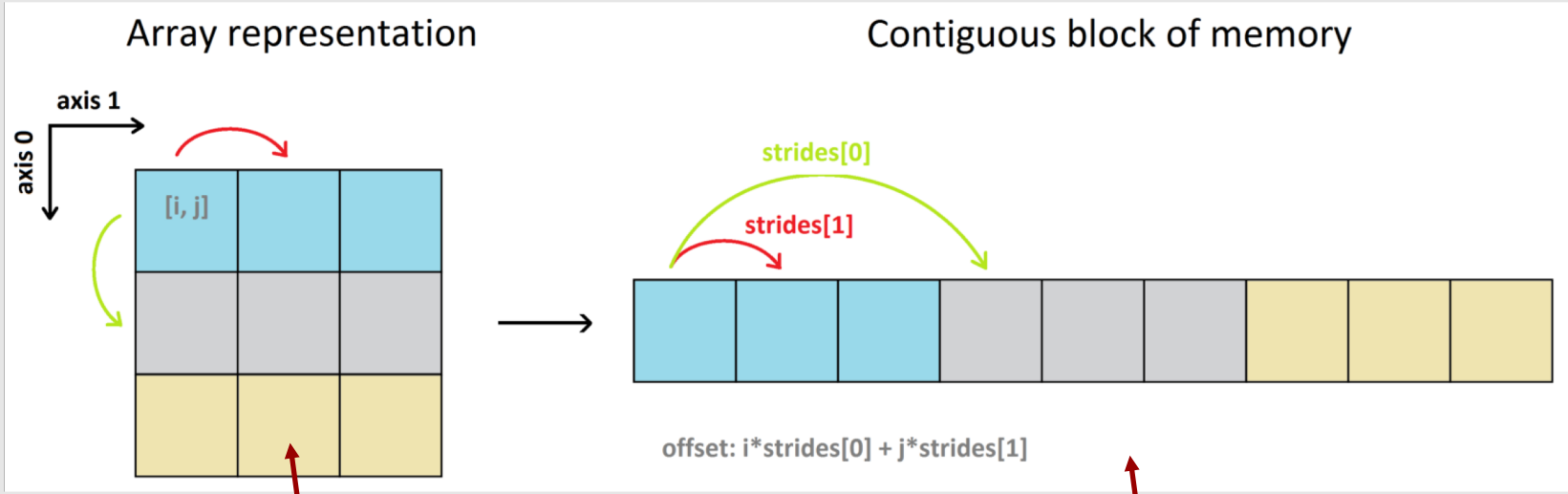
Indexing an array of Numpy (optional)

In computer memory, elements in a numpy array are stored along the rows, which means the location of an element is closer to those of its neighbors in row than those of neighbors in column.

- In the 2D numpy array, 5.2 has two neighbors: 3.0 and 9.1. But 3.0 is the nearest neighbor.
- In the 3D numpy array, 1 has three neighbors: 4, 2, 2. But 4 is the nearest one, and 2 on axis2 is the farthest one.



How a Numpy array is allocated in memory (optional)



This is how the array looks like in the view of Python users.

This is how the array being stored in the computer memory.

So, indexing the elements along the rows has better spatial locality.

Maximize locality for better performance (in 2d array)

There are two ways to sum up the elements in a 2d array.

```python
import numpy as np # package for generating the array
import timeit # package for counting the runtime

array_2d = np.random.rand(1000, 1000)
s = 0

## in array_2d[rows, columns]

## Case I: we sum up the elements along the column
start = timeit.default_timer()
for i in range(1000):
    for j in range(1000):
        ## we fix the column index, change the row index
        s += array_2d[j,i]
stop = timeit.default_timer()
print("Case I")
print("Time for summing all elements:", stop-start)
```

```python
import numpy as np # package for generating the array
import timeit # package for counting the runtime

array_2d2 = np.random.rand(1000, 1000)
s = 0
## Case II: we sum up the elements along the row
start = timeit.default_timer()
for i in range(1000):
    for j in range(1000):
        ## We fix the row index, change the column index
        s += array_2d2[i,j]
stop = timeit.default_timer()
print("Case II")
print("Time for summing all elements:", stop-start)
```

Different orders of indexing the elements.

Please run the codes on your laptop,

- Which one takes more time, why?
- Which program is better?

Assignment: BBC the history of computers

Clip 3 : Computer as a “crystal ball” has a long (and also short) history

- Which kind of ideas excite you more?
  - Bright ideas
  - Good ideas
  - Successful ideas

Clip 4 : Without software computers are useless.

- What was the problem that caused programmer insufficiency in 1950s?
- How did people solve that problem?