Computer Science Fundamentals

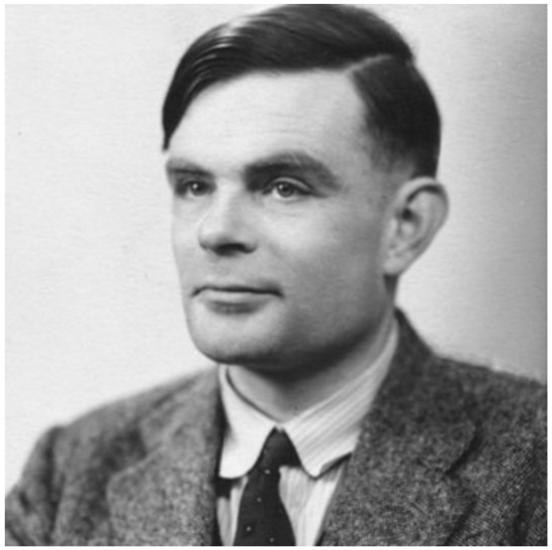
Prof. Jean-Pierre Hubaux
Faculté Informatique et Communications EPFL

With gratitude to Christian Mouchet who prepared most of the material

The analogue computer

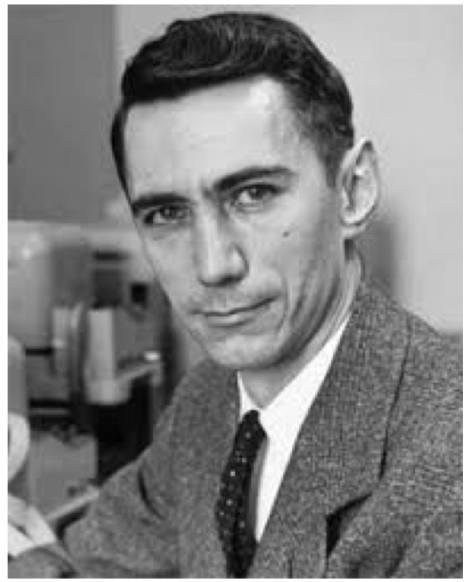
Vannevar Bush and the differential analyzer, 1931

The digitization pioneers (middle of the 20th century)



Alan Turing

John von Neumann



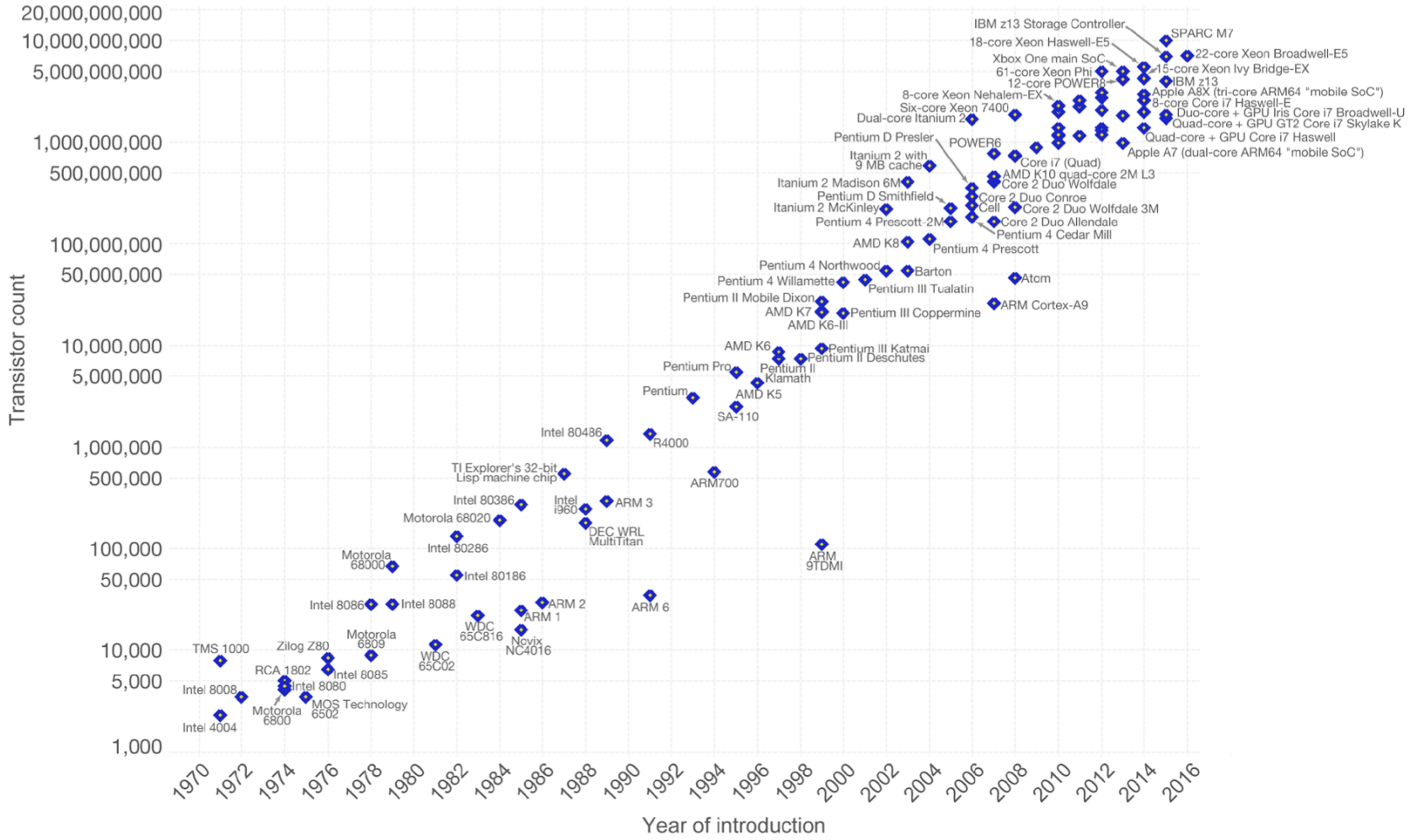
Claude Shannon

Integrated circuits and optical fiber

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

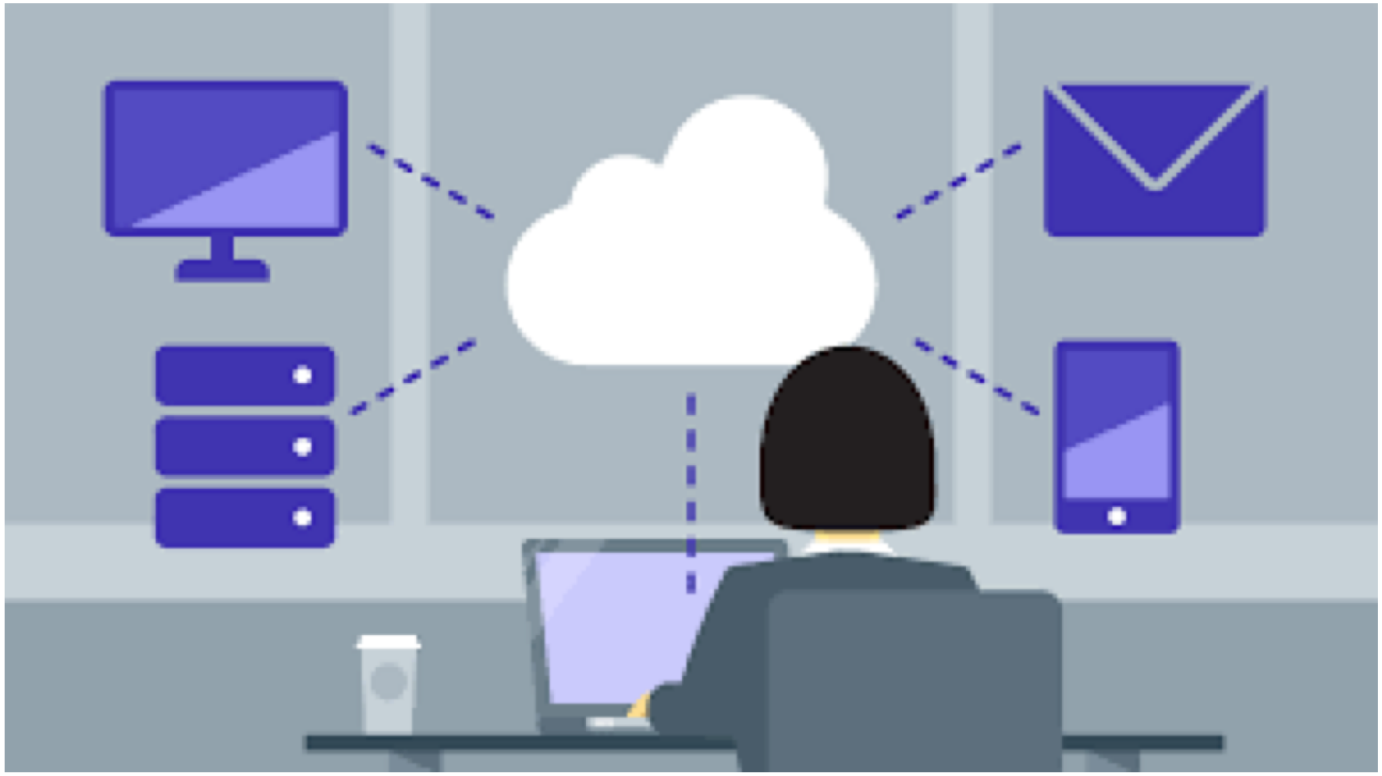
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

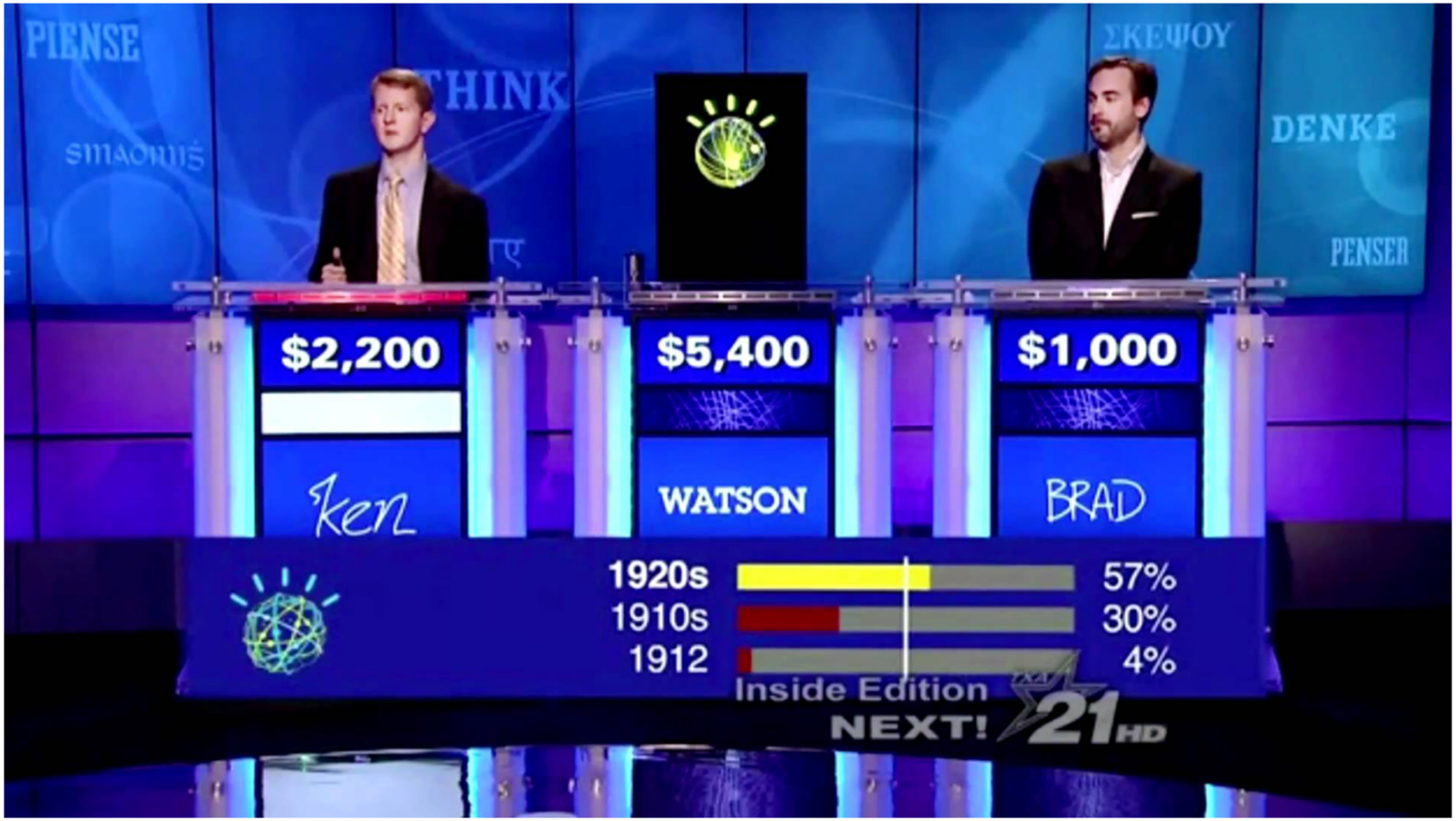
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Cloud Computing:
storage and computing power for rent

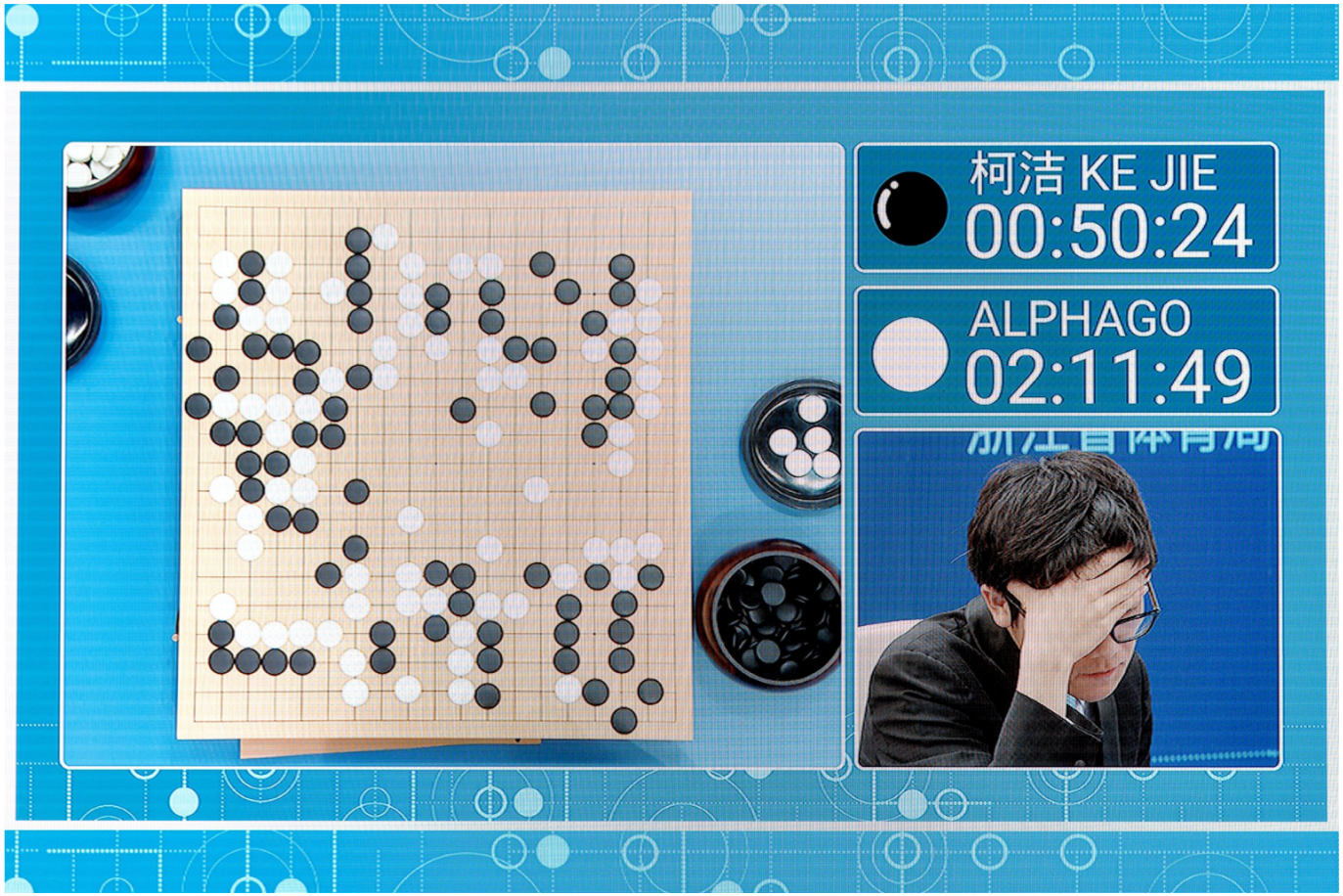


The computer supremacy: Chess (1997)

The computer supremacy: Jeopardy! (2011)



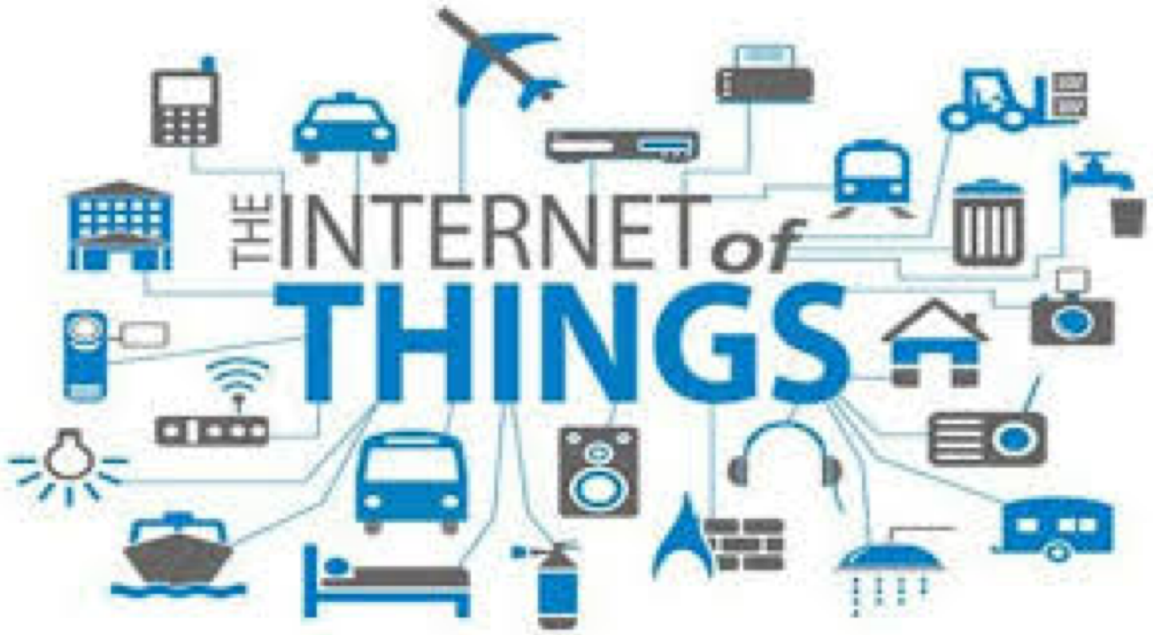
The computer supremacy: Go (2016)



Mobile networks

1G	2G	3G	4G	5G
1981	1992	2001	2010	2020(?)
2 Kbps	64 Kbps	2 Mbps	100 Mbps	10 Gbps
Basic voice service using analog protocols	Designed primarily for voice using the digital standards (GSM/CDMA)	First mobile broadband utilizing IP protocols (WCDMA / CDMA2000)	True mobile broadband on a unified standard (LTE)	'Tactile Internet' with service-aware devices and fiber-like speeds
				?

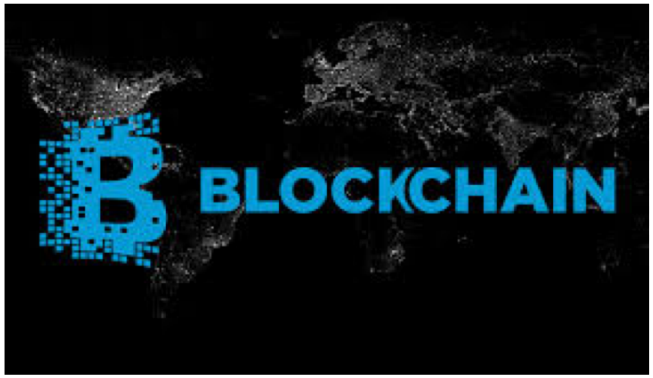
Internet of Things (IoT)

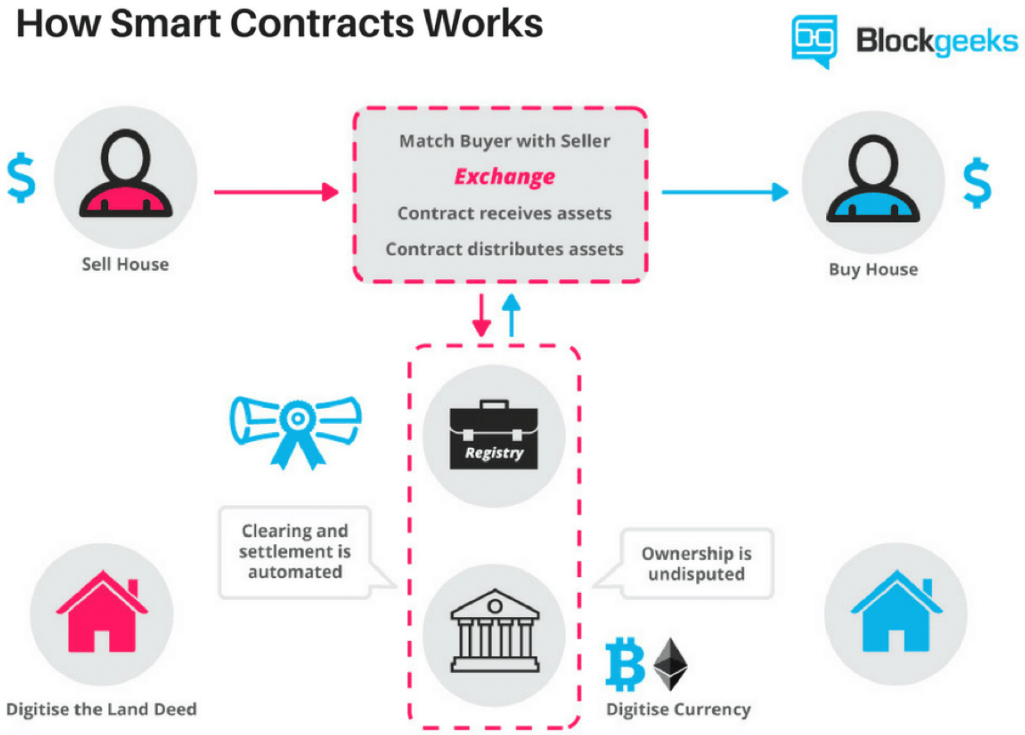


Who will have access to the data ?

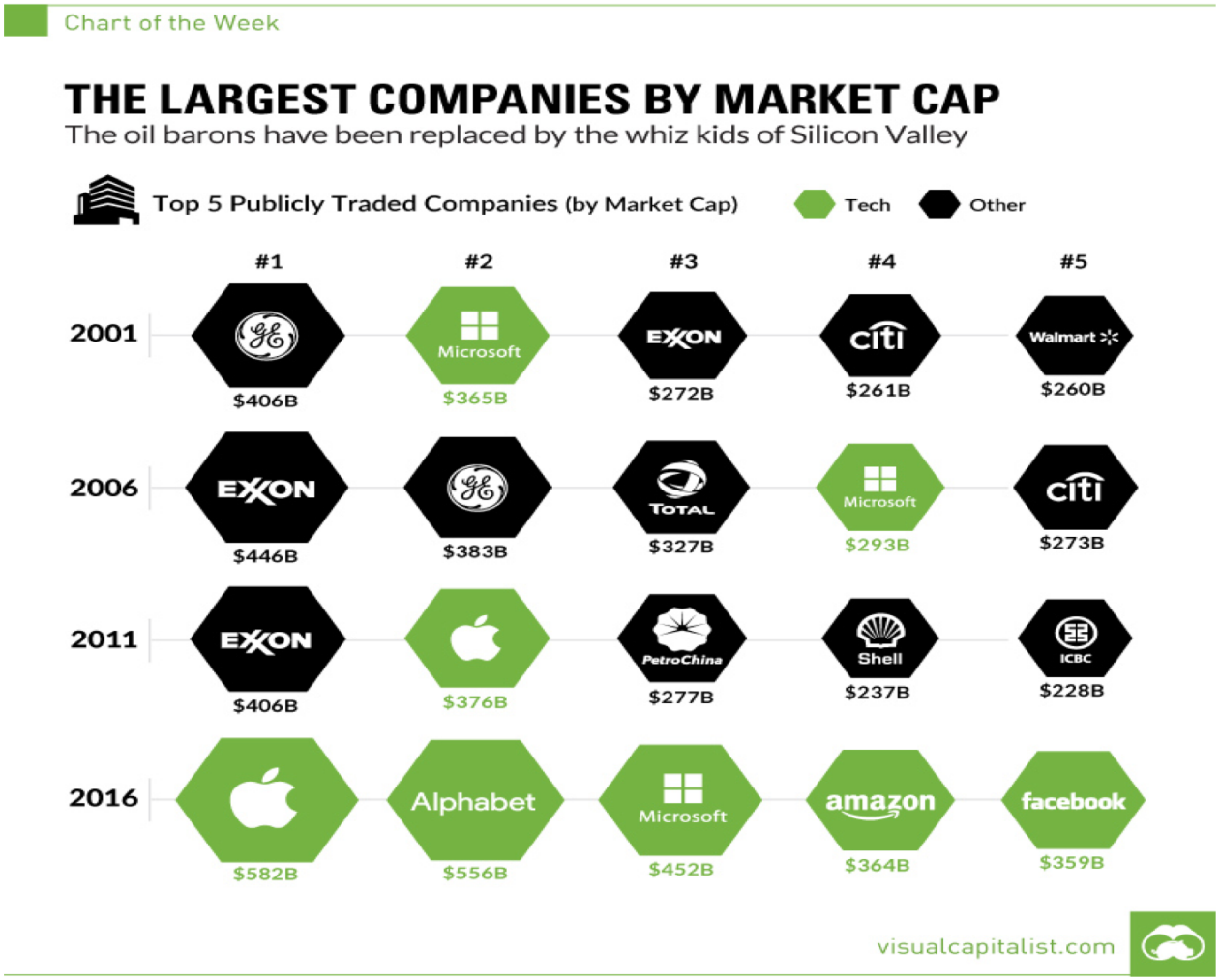
Uber and autonomous vehicles

Cryptocurrencies and *smart contracts*





The triumph of software (companies)



Cyber attacks, cyber war, fake news

Stuxnet, the computer virus discovered in 2010

Mass surveillance

EU GDPR:
General Data Protection Regulation
(Effective: may 2018)

Computer Science Fundamentals

How does this all work ?

What is computing

- **Input/output** view
- Nearly all computer program can be viewed this way (yes)
  - Sending an e-mail: Input is the text, output is the internet
  - Receiving an e-mail: Input is the internet, output is the mailbox
  - Reading an e-mail: input is the mailbox, output is a screen
- But this is an **abstraction**
- Let's dive into what computing is, in a bottom-up approach

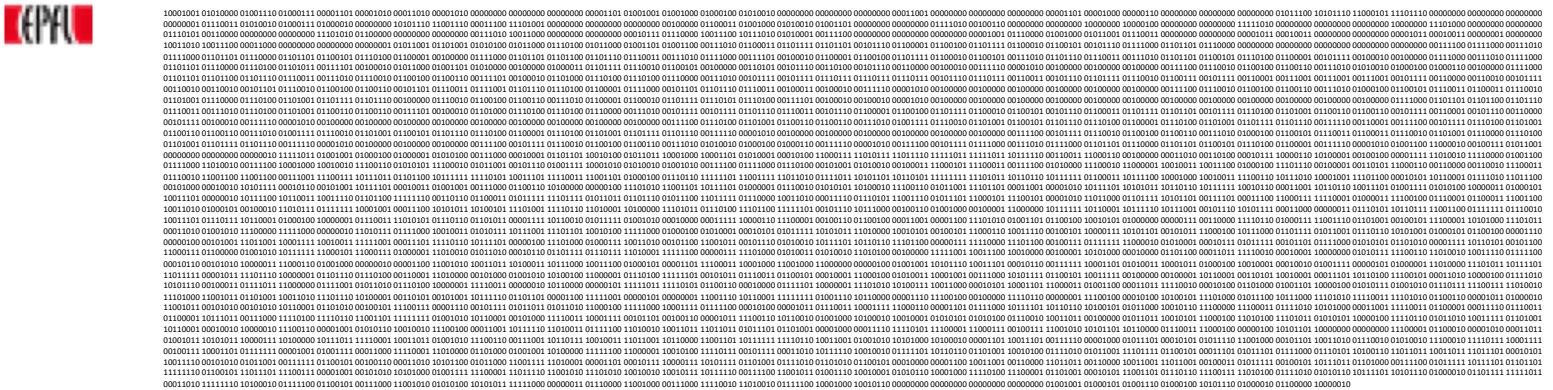
Binary

- Examples of **binary** data

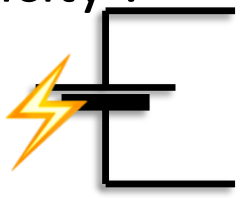
1: 1 2: 10 3: 11 4: 100

"EPFL": 01000101 01010000 01000110 01001100

69 80 70 76



- Why ? Electricity !



"**Bits**" for **bi**nary digi**ts** are convenient to

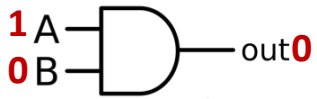
1. **Operate on**
2. **Store**
3. **Transmit**

Binary circuits

1. **Operate on**
2. **Store**
3. **Transmit**

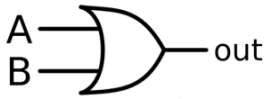
- We can perform very simple operations on these bits:

AND



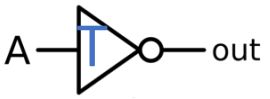
A	B	out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

OR



A	B	out
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

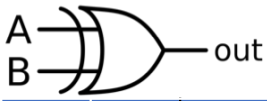
NO



A	out
0	**1**
1	**0**

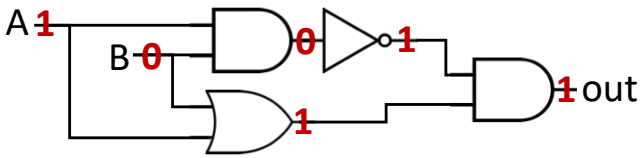
- These simple operation “gates” can be **built** as electric circuits from a few (2-3) **transistors**

XOR



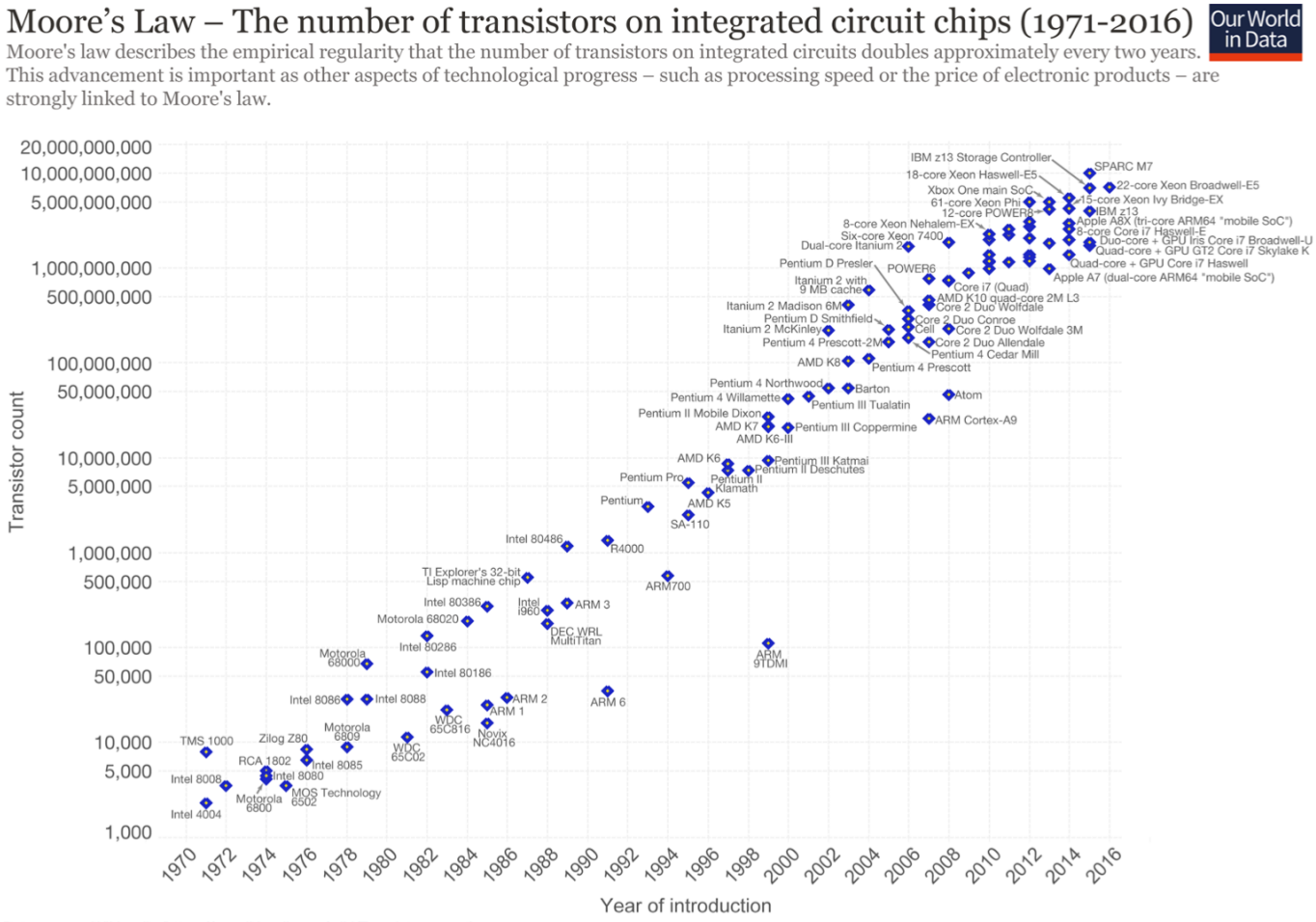
A	B	out
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

- The **wires** can be **connected** to build new operations:



(Very large) Binary circuits

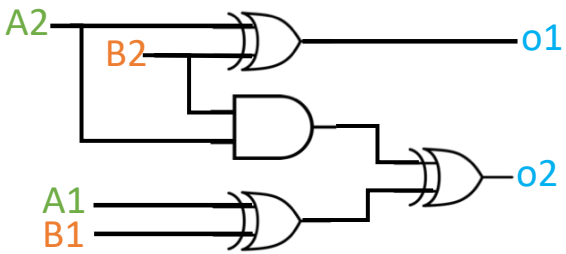
1. **Operate on**
2. **Store**
3. **Transmit**



- These simple operation "gates" can be **built** as electric circuits from a few (2-3) **transistors**
- The number of transistors we can put on one circuit is now in the order of Billions
- Moore's law predicts that this number doubles every 2 years.
- … but we are slowly reaching the physical limits of the size of an atom (expected around ~2025)

Binary circuits – Activity

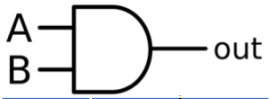
1. **Operate on**
2. **Store**
3. **Transmit**



Complete this operation's table:

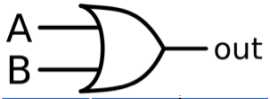
A1	A2	B1	B2	o1	o2
0	0	0	0		
0	1	0	0		
0	1	0	1		
1	0	0	1		

AND



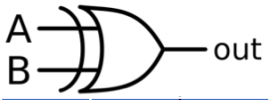
A	B	out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

OR



A	B	out
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

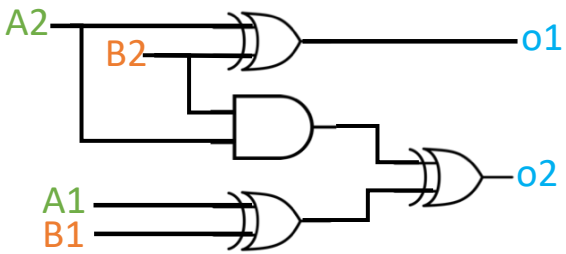
XOR



A	B	out
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

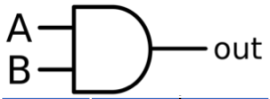
Binary circuits – Activity (solution)

1. **Operate on**
2. **Store**
3. **Transmit**



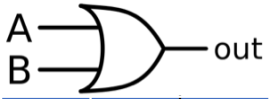
	A1	A2	B1	B2	o1	o2	
A=0 B=0	0	0	0	0	**0**	**0**	out = 0
A=1 B=0	0	1	0	0	**0**	**1**	out = 1
A=1 B=1	0	1	0	1	**1**	**0**	out = 2
A=2 B=1	1	0	0	1	**1**	**1**	out = 3

AND



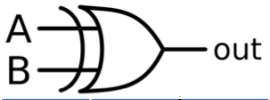
A	B	out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

OR



A	B	out
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

XOR



A	B	out
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

- This is the binary addition operation on 2 bits numbers

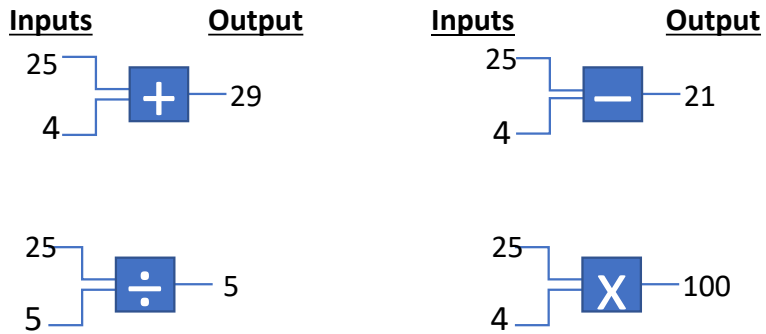
- Computers use **64** bits numbers
  → A and B above are 64 wires going in the circuit
  → Can represent numbers up to:
  18,446,744,073,709,551,615

Arithmetic circuits – Specific operations

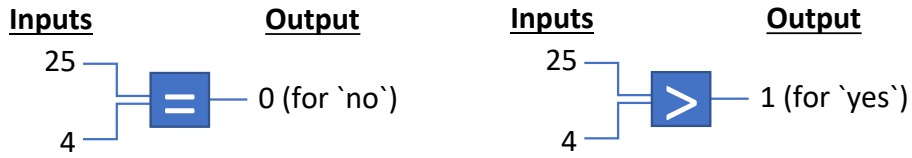
1. **Operate on**
2. **Store**
3. **Transmit**

- We can build all types of circuit we want, some examples:

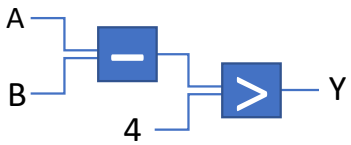
Arithmetic: ***add, subtract, multiply, divide,…***



Comparisons: ***equal, smaller than, greater than***



- Like binary circuit: **composable**, very small and very cheap
- By combining them, we can build circuits that compute what we want

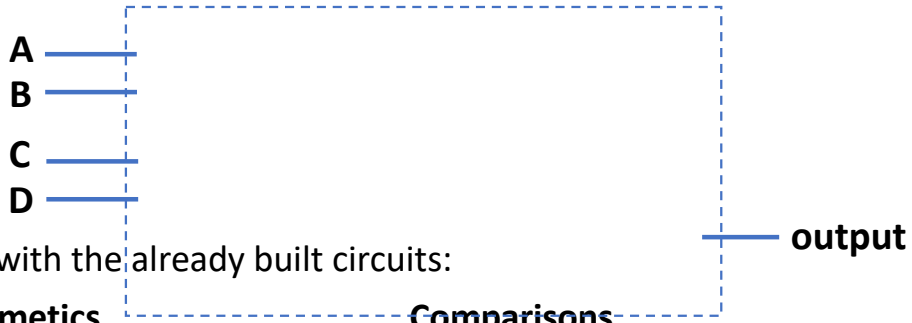


"Output is 1 (yes) if the difference between the two inputs is more than 4, otherwise output is 0 (no)"

1. **Operate on**
2. **Store**
3. **Transmit**

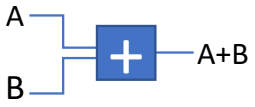
Arithmetic circuits – Activity

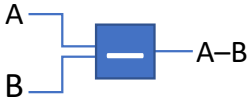
- Build a circuit that compute the average of 4 grades **A**, **B**, **C**, **D** and **outputs**:
  - "1" if the student failed the course
  - "0" if he did not.
- The passing grade is 3.

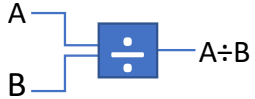


- … with the already built circuits:

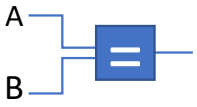
Arithmetics







Comparisons



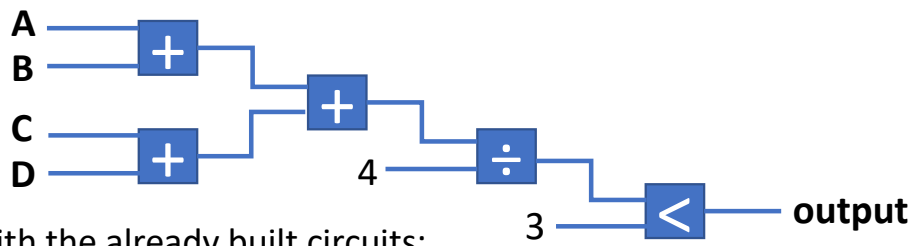
***1** if A is equal to B*
***0** if not equal*

***1** if A is smaller than B*
***0** if A larger or equal than B*

Arithmetic circuits – Activity (solution)

1. **Operate on**
2. **Store**
3. **Transmit**

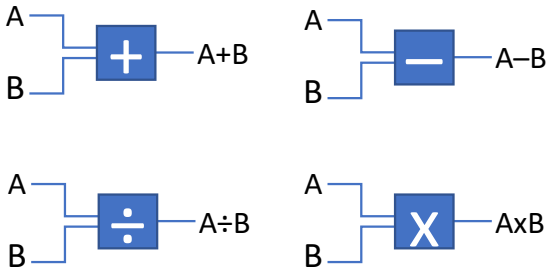
- Build a circuit that compute the mean of 4 grades **A**, **B**, **C**, **D** and **outputs**:
  - "1" if the student failed the course
  - "0" if he did not.
- The passing grade is 3.



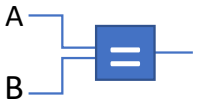
- … with the already built circuits:

- We can build the circuit corresponding to each of our needed computations
- And each time we need to compute something else, we change the circuit… Is there a problem ?
- Circuit is physical: **hardware** is "hard" to change

Arithmetics



Comparisons



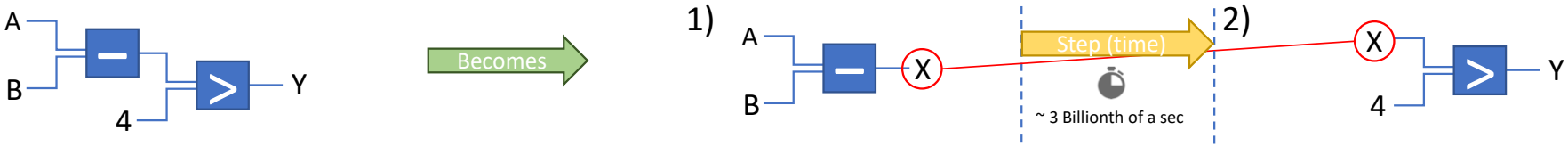
1 *if A is equal to B*
0 *if not equal*

1 *if A is smaller than B*
0 *if A larger or equal than B*

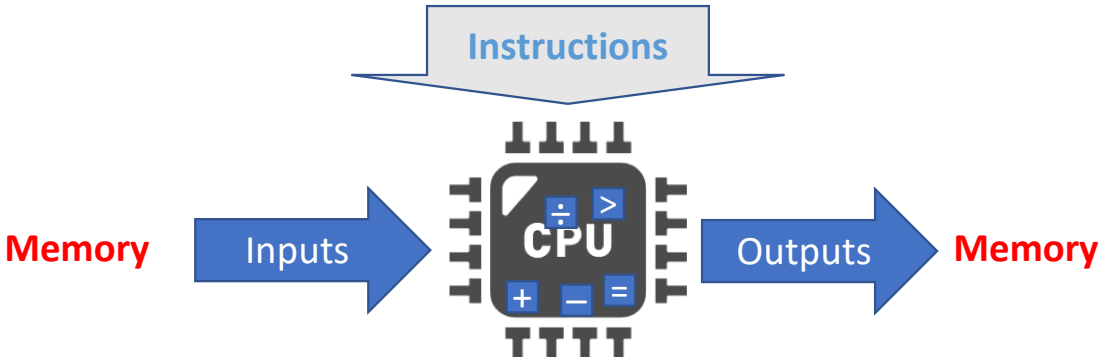
Arithmetic circuits – Generic operations

1. **Operate on**
2. **Store**
3. **Transmit**

- There is an infinity of possible circuits: we cannot ”build” them all !
- Solution: evaluate each part of the circuit one by one



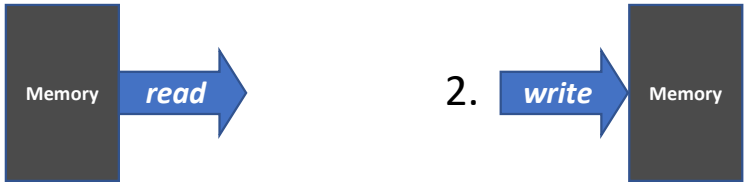
- Any circuit becomes possible given sufficient number of **steps** (i.e. time) and **two conditions**:
  1. The machine needs to be capable of **memorizing** **variable** X (next slide)
  2. The machine needs to know what are the steps and their order: the **instructions** (in two slides)
- We call the component that can do that the **C**entral **P**rocessing **U**nit

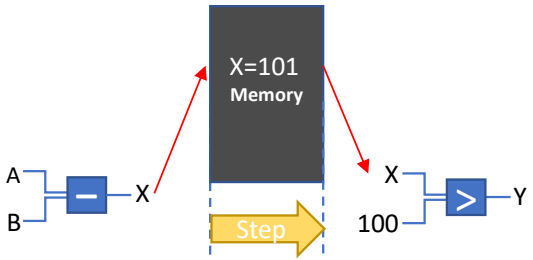


1. **Operate on**
2. **Store**
3. **Transmit**

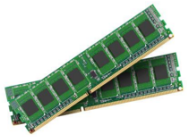
Memory

- The machine needs to be capable of **memorizing** bits between “steps”
- Thus, it needs **memory** and two operations



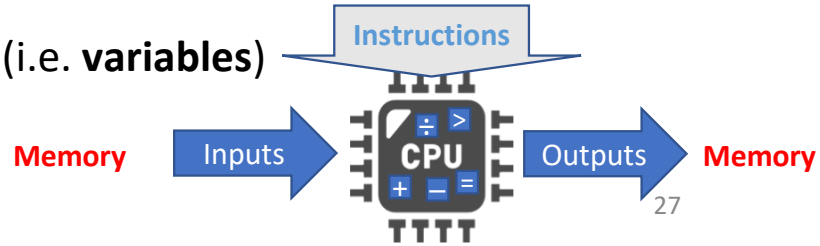


- In **practice**, many types of memory of varying **capacity** and **speed**

	Digital Versatile Disk	Hard Disk Drive	Flash Memory	Random Access Memory
				
Capacity	4.7 GB	1-5 TB	8-500 GB	8-64 GB
Latency	0.5-1 s	3-10 ms	30 µs	15 ns

- In **theory**, they are all the same: a place to store information (i.e. **variables**)

01110011010110010101100101010 1



Computer program

1. **Operate on**
2. **Store**
3. **Transmit**

- The machine needs to know what are the steps and their order: the **instructions**
- A computer program is a **list** of instructions that the CPU follows

1. ***subtract*** A and B; store result in X
2. **compare** X to 4; store result in Y

- Possible instructions not only arithmetic (-) or condition (>), but also **control**

Arithmetics

add *X A B : set X to A+B*

sub *X A B : set X to A–B*

mul *X A B : set X to A x B*

div *X A B : set X to A ÷ B*

Conditions

equ *X A B : set X to 1 if A = B*

neq *X A B : set X to 1 if A not = B*

less *X A B : set X to 1 if A < B*

mor *X A B : set X to 1 if A > B*

Controls

jmp *S* : *jump to the Sth Instruction*

jmpc *C S* : *if C is 1, do **jmp** S, else continue*

exit : *stop the program (end)*

- Conditions and controls make it possible for the program to adapt to different inputs

1.	***inputs***	*A and B*
2.	***sub***	X A B
3.	**mor**	Y X 4
4.	**jmpc**	Y 7
5.	**output**	*"Difference is less than or equal to 4."*
6.	**exit**	
7.	**output**	*"Difference is more than 4."*

Computer program – Activity

- Write the two following programs

The program that takes the grades A, B, C, D as input and outputs:

- "The student passes" if the average grade is at least 3
- "The student fails" if the average grade less than 3

```
1.  inputs   A B C D
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
```

The program that takes the number C as input, counts from C down to 0 and then outputs "done"

```
1.  inputs   C
2.
3.
4.
5.
6.
7.
8.
9.
10.
```

Arithmetics

add *X A B : set X to A+B*

sub *X A B : set X to A–B*

mul *X A B : set X to A x B*

div *X A B : set X to A ÷ B*

Conditions

equ *X A B : set X to 1 if A = B*

neq *X A B : set X to 1 if A not = B*

less *X A B : set X to 1 if A < B*

mor *X A B : set X to 1 if A > B*

Controls

jmp *S* : *jump to the S^{th} Instruction*

jmpc *C S* : *if C is 1, do **jmp** S, else continue*

exit : *stop the program (end)*

Computer program – Activity (solution)

- Write the two following programs

The program that takes the grades A, B , C, D as input and outputs:

- "The student passes" if the average grade is at least 3
- "The student fails" if the average grade less than 3

```
1.  inputs    A B C D
2.  add       X A B
3.  add       X X C
4.  add       X X D
5.  div       X X 4
6.  less      Y X 3
7.  jmpc      Y 10
8.  output    "The student passes"
9.  exit
10. output    "The student fails"
11.
12.
13.
14.
15.
```

For Real

```
pushq   %rbp
movq    %rsp, %rbp
subq    $0x20, %rsp
leaq    0xd9(%rip), %rdi
leaq    -0x8(%rbp), %rsi
leaq    -0xc(%rbp), %rdx
leaq    -0x10(%rbp), %rcx
leaq    -0x14(%rbp), %r8
movl    $0x0, -0x4(%rbp)
movb    $0x0, %al
callq   0x100000f3a
movsd   0xa3(%rip), %xmm0
movsd   0xa3(%rip), %xmm1
movss   -0x8(%rbp), %xmm2
addss   -0xc(%rbp), %xmm2
addss   -0x10(%rbp), %xmm2
addss   -0x14(%rbp), %xmm2
cvtss   %xmm2, %xmm2
divsd   %xmm1, %xmm2
ucomisd %xmm2, %xmm0
movl    %eax, -0x18(%rbp)
jbe     0x100000f13
leaq    0x87(%rip), %rdi
movb    $0x0, %al
callq   0x100000f34
movl    $0x0, -0x4(%rbp)
movl    %eax, -0x1c(%rbp)
jmp     0x100000f2b
leaq    0x7e(%rip), %rdi
movb    $0x0, %al
callq   0x100000f34
movl    $0x0, -0x4(%rbp)
movl    %eax, -0x20(%rbp)
movl    -0x4(%rbp), %eax
addq    $0x20, %rsp
popq    %rbp
retq
```

The program that takes the number C as input, counts from C down to 0 and then outputs "done"

```
1.  inputs    C
2.  sub       C C 1
3.  neq       X C 0
4.  jmpc      X 2
5.  output    "Done"
6.
7.
8.
9.
10.
```

→ Programming this way is not easy

Arithmetics

add *X A B : set X to A+B*

sub *X A B : set X to A–B*

mul *X A B : set X to A x B*

div *X A B : set X to A ÷ B*

Conditions

equ *X A B : set X to 1 if A = B*

neq *X A B : set X to 1 if A not = B*

less *X A B : set X to 1 if A < B*

mor *X A B : set X to 1 if A > B*

Controls

jmp *S* : *jump to the S^{th} Instruction*

jmpc *C S* : *if C is 1, do **jmp** S, else continue*

exit : *stop the program (end)*

Computer programming languages

- We need a faster, human-friendly way of programming computers
- Idea: create ourselves the languages we want to express our programs
- have another program translate it to machine instructions: **the compiler**

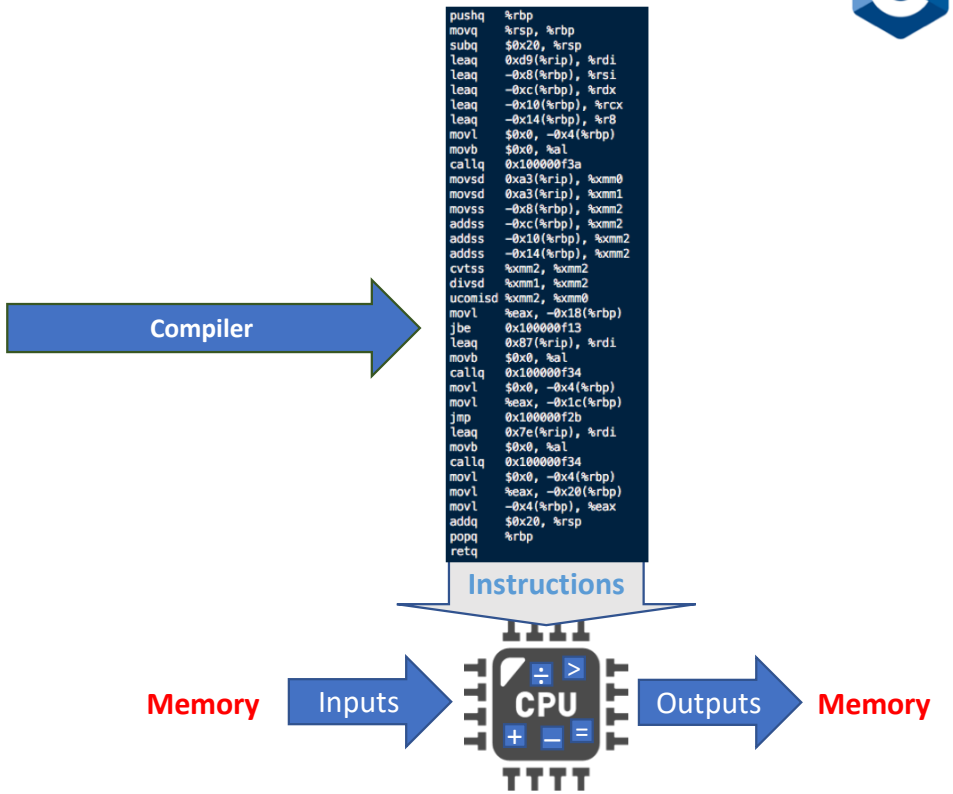
The grade average example in the C language:

```c
int main()
{
    float A;
    float B;
    float C;
    float D;

    scanf("%f %f %f %f", &A, &B, &C, &D);

    if ((A+B+C+D)/4.0 < 3.0) {
        printf("The student fails \n");
        return 0;
    }

    printf("The student passes \n");
    return 0;
}
```



Algorithms

- There are a lot of programming languages available to write a program

The student grade program in **C** (truncated)

```c
scanf("%f %f %f %f", &A, &B, &C, &D);

if ((A+B+C+D)/4.0 < 3.0) {
    printf("The student fails \n");
    return 0;
}

printf("The student passes \n");
```

The student grade program in **Python**

```python
A,B,C,D = input().split()

if (float(A)+float(B)+float(C)+float(D))/4 < 3:
    print("The student fails")
    exit()

print("The student passes")
```

- Although the textual representation of the program differs, two things remain the same
  1. The problem to be solved: computing the average grade and output the corresponding result
  2. The **way of computing the solution**:
     - Summing up all the outputs
     - Dividing by 4
     - Test if the average grade is sufficient
       - If no, output a failure
       - If yes, output a success

This is called the "**algorithm**"

- **Independent** of the programming language
- The "tricky" part when the problem becomes **more complicated**
- A good algorithm can make **huge difference** on the **time** it takes to compute something

Algorithms

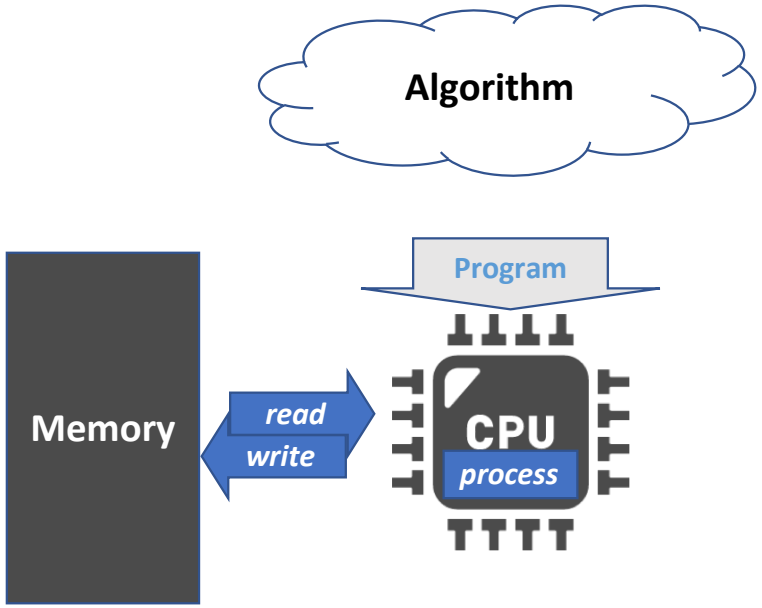
- Other examples of well known algorithms:
  - Search for a word in a (potentially very long) sentence
  - Search for the shortest path on a map
  - Optimize the current flow in a power grid
  - Sort an array of numerical values

- More complex ones:
  - Predict if a given user is likely to like a movie (e.g. Netflix)
  - Recognize and label subjects in pictures
  - Find the most relevant web pages given a search query (e.g. Google)

- The (near) future
  - Predict that a patient will probably develop some disease
  - Predict crimes
  - ???

A ”computer” (in the 70s)

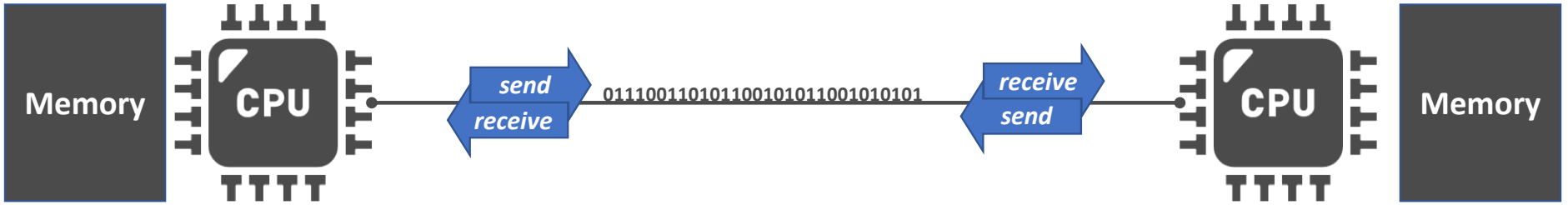
1. Operate on ✓
2. Store ✓
3. Transmit ✗



- What is missing ?

Computer networking

1. **Operate on**
2. **Store**
3. **Transmit**



- Two computer that are **connected** can use specific **protocols** to communicate
- In **practice**, several kind of connection

Ethernet Cable

400 Gbits/s

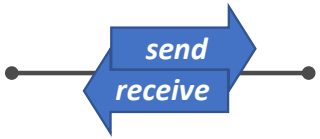
Wireless Adapter

200 Mbits/s

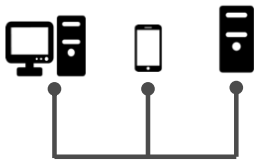
4G Antenna

100 Mbits/s

- In **theory**, they are all the same



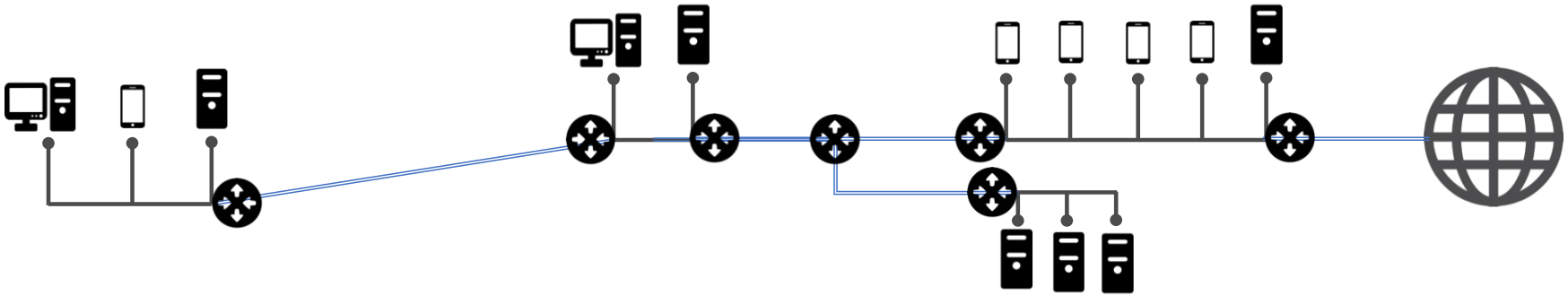
Not only 1-to-1, but **many-to-many**:



"a network"

The Internet

- Basic idea of the Internet: connect networks together in a larger network



- Is it possible to have cables between each and every networks in the world ? No !
- Use multiple «hops» to communicate with distant networks
- Intermediary nodes:
  - Are **computers** too…
  - Can **see the data** they forward: need for encryption
  - Need to know where the message should be transmitted next: need for a way of **addressing** other computers

  → Every computer on the Internet has an address called **IP address** (for **I**nternet **P**rotocol)
- More on the networks and the Internet in a later module

A ”computer” today