

Information Technology for Digital Humanities

Lecture 8

Mario Verdicchio

Università degli Studi di Bergamo

Academic Year 2023-2024

Lecture 8 (October 18 2023)

- Electronic circuits for arithmetic, logic, and control
- Computer science subfields

Where are the addresses stored? We have seen that, as data, they can be stored inside the memory itself. But how are they used as addresses?

?

0000

01101010

0001

11001110

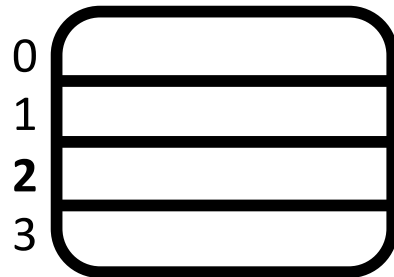
0010

11011011

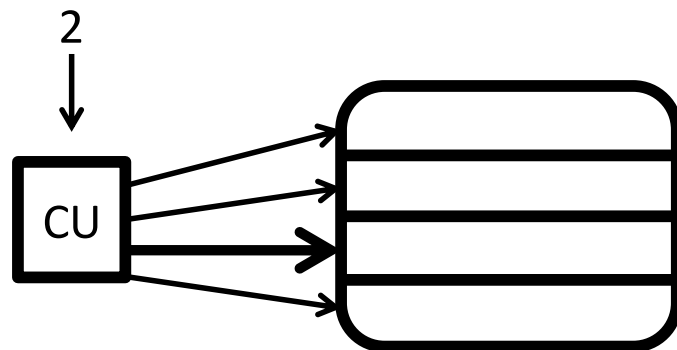
0011

10001011

How memories are addressed



This is our mental model of a computer memory: each word is tagged with a numerical address, which we use to access a specific word. In this example, we want to address word 2.



In reality, what happens is as follows.

No word is tagged with a number, but each word is physically connected to the Control Unit. In the early days of computers, such connection was a cable; nowadays it is an electronic circuit trace.

When we want to access word 2, we send 2 as an input to the CU, which activates the relevant connection, thus granting access to the word. Our choice (input 2) is translated into a physical activation.

To understand how human choice maps onto parts of an electronic circuit, we must take a dive into the basic electronic components that constitute a computer hardware.

This technology dates back to the 1950s, when american scientists Bardeen, Brattain and Shockley invented the transistor, a small device based on a special property of semiconductive materials.

Semiconductive materials are particular: at rest, they do not allow electricity to flow, but if electrically stimulated they become conductive.

Hence, transistors are a special kind of switches: very small and electronic, that is, not mechanical but controlled by means of electricity.



John Bardeen

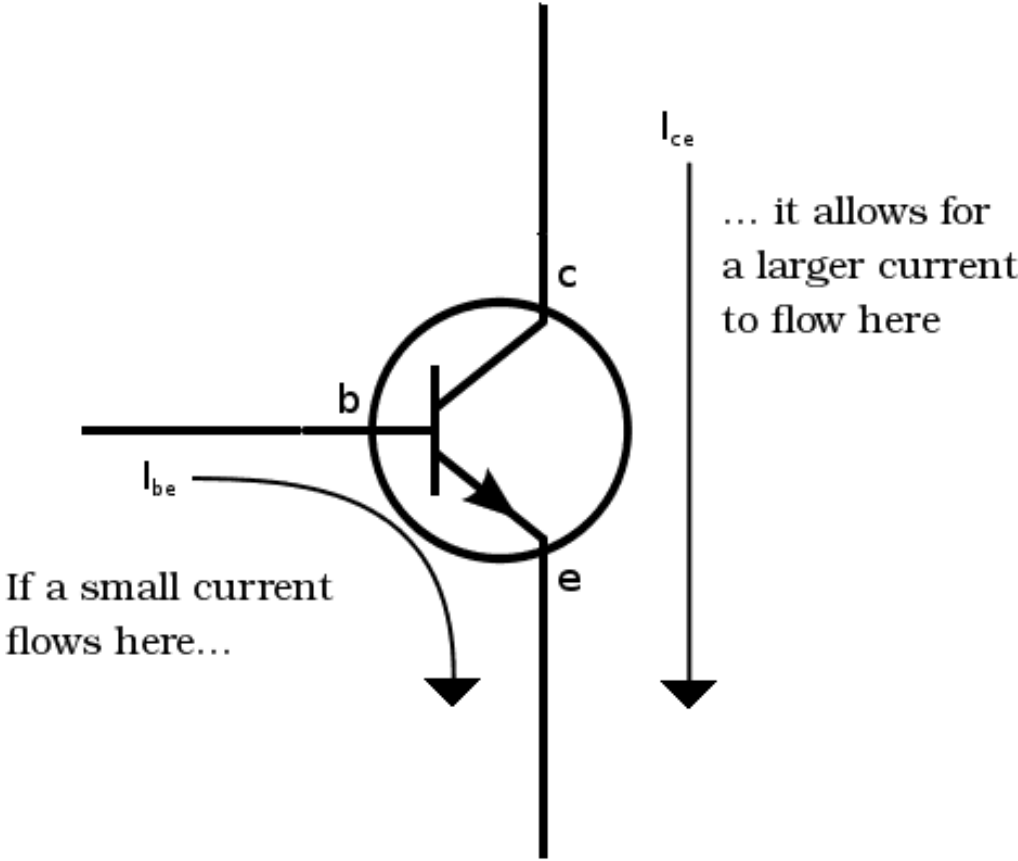
Walter Brattain

William Shockley

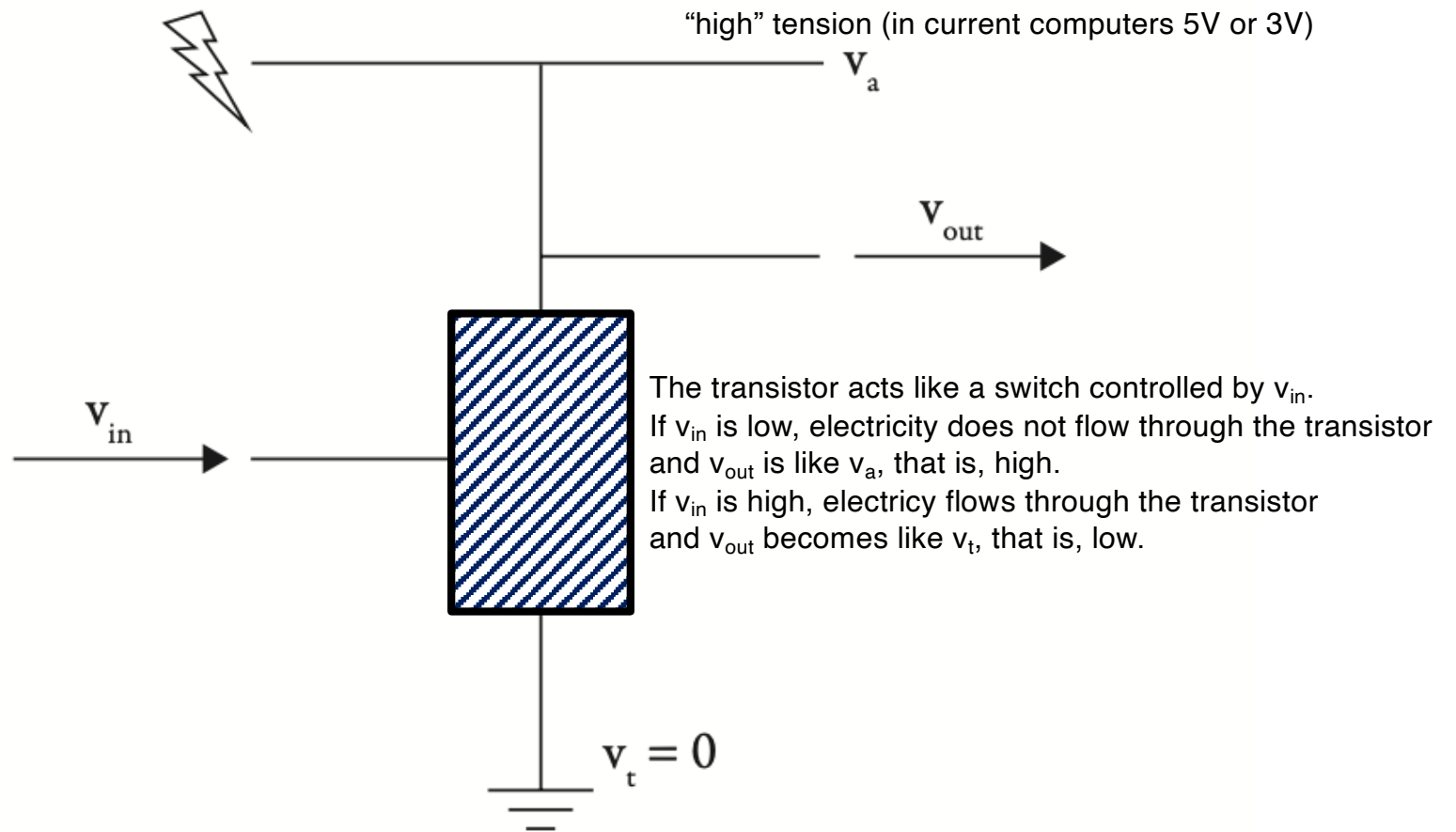
a transistor from the 1950s



how it works



A more abstract description of a transistor



The transistor acts like a switch controlled by v_{in} .
If v_{in} is low, electricity does not flow through the transistor and v_{out} is like v_a , that is, high.
If v_{in} is high, electricity flows through the transistor and v_{out} becomes like v_t , that is, low.

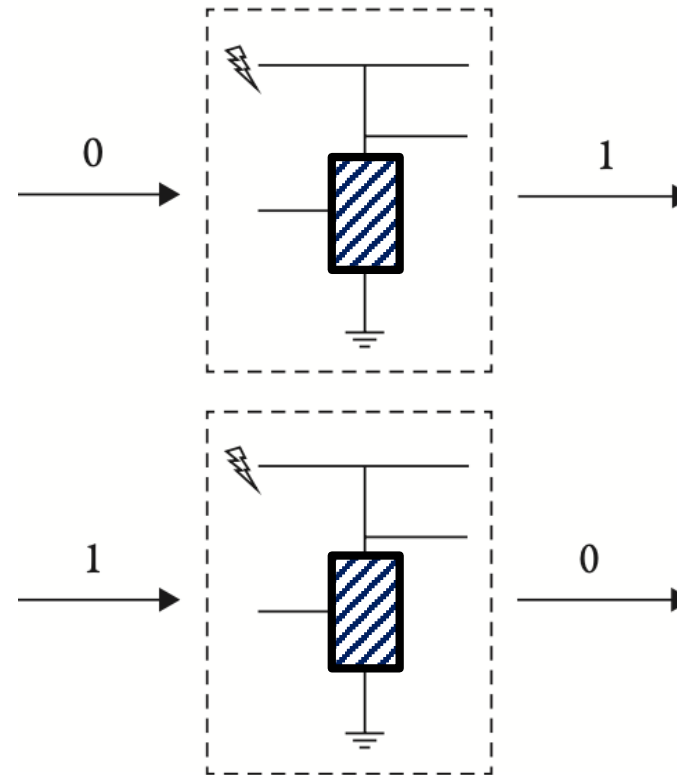
Ground: no tension here. Electricity flows from points of high tension to points of low tension.

The fundamental encoding inside a computer

v_t ----- 0

v_a ----- 1

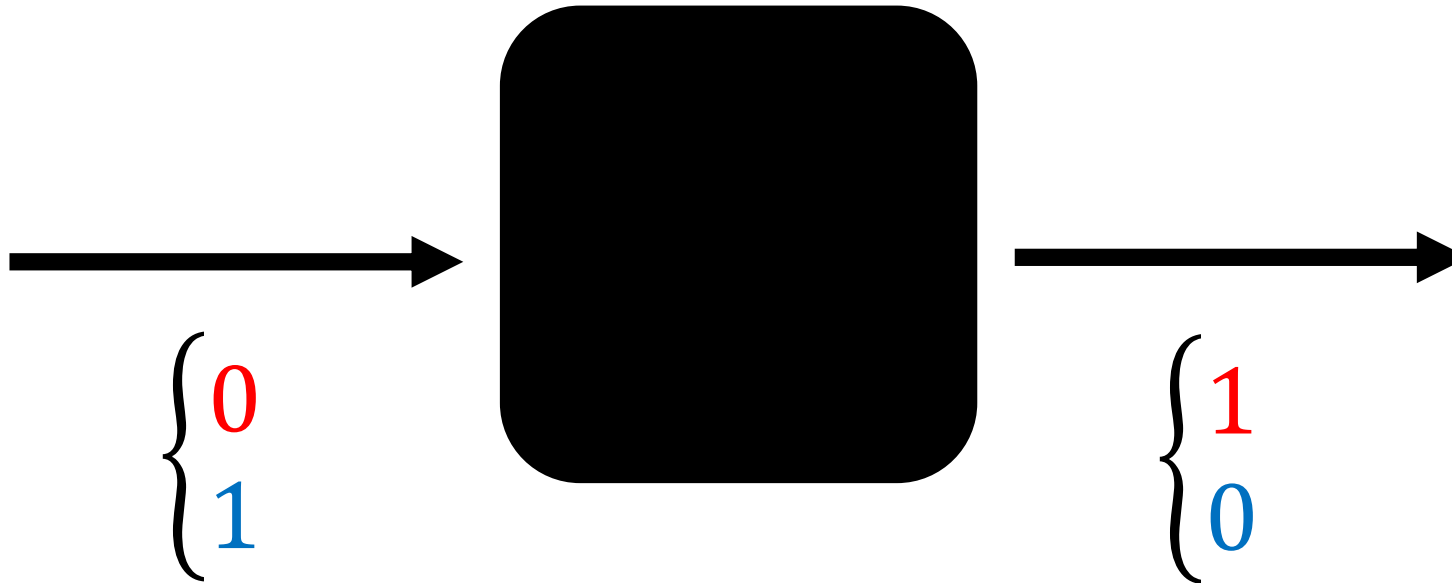
We interpret the high tension inside a computer like a "1", and the low tension like a "0". This is an encoding: it maps two entities in the real world onto a set of natural numbers (0 and 1).



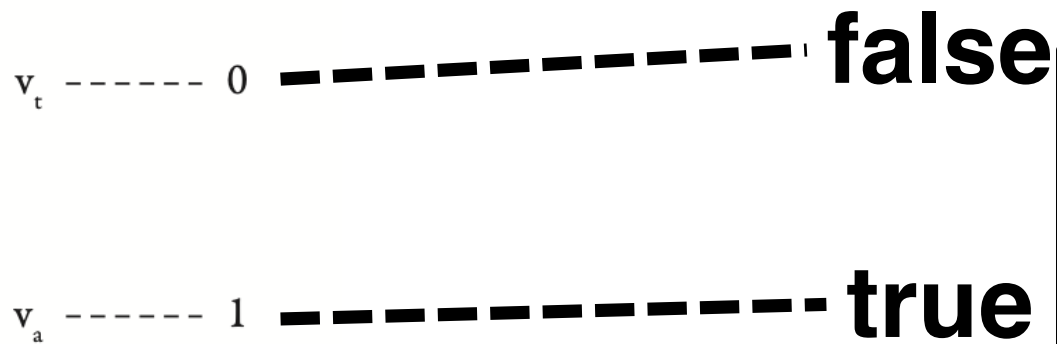
With such encoding in mind, a single transistor acts like a system which replies with "1" when we give it "0", and viceversa.

case 1
case 2

We will use this graphical organization of multiple cases.
Case 1 is when we give the system "0" in input and obtain "1" in output.
Case 2 is when we give "1" in input and obtain "0" in output.



What to do with a system with this behavior?



If we use '0' to encode the concept of "false", and '1' to encode "true", we are moving from arithmetic with numbers to logic with truth values.

Let's introduce another encoding on top of the fundamental one.

Logic is the discipline that formalizes reasoning, that is, aims at making reasoning (e.g. "all men are mortal; Socrates is a man; hence, Socrates is mortal") rigorous by transforming sentences in sequences of symbols (called formulas) that are manipulated by means of rules.

"False" and "true" are "truth values" that we assign to formulas. Logic does not help us establish the truth values of hypotheses from the real world (e.g. "does God exist?") but it supports us in checking whether a certain way of reasoning is correct or not.

case 1
case 2

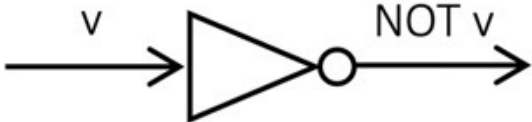
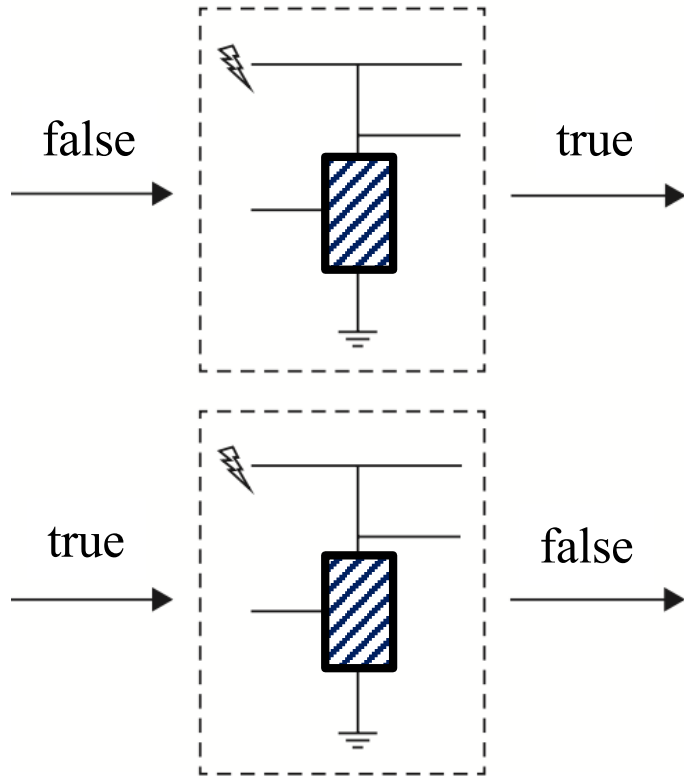
Let's take a look at the system again, with the logic-oriented encoding in mind.



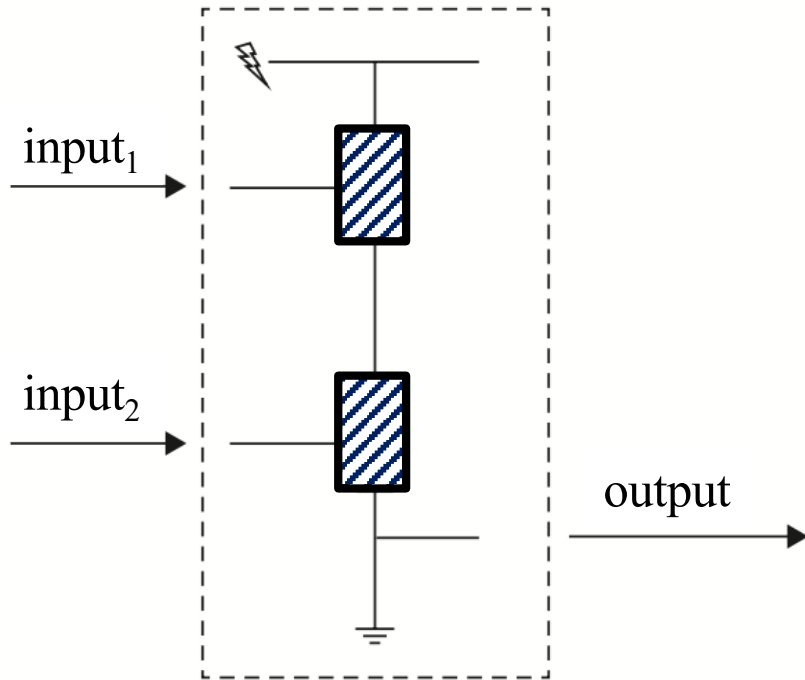
When we give it "false", it replies with "true".
When we give it "true", it replies with "false".

The system acts like a "not".
Indeed, "not false" is the same as "true", and "not true" is the same as "false".

A transistor can then be seen as the electronic embodiment of the “negation” operator in logic.



This is how a system that negates, also known as “NOT gate” (“gate” because the electric signal goes through it), is represented in the graphical standard for circuit design.



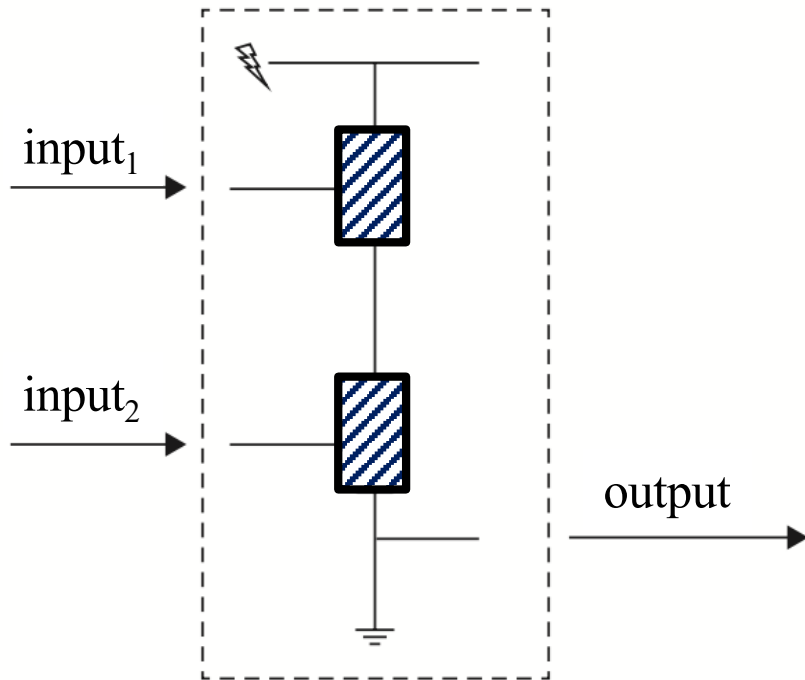
Let's build more complex systems, by using more than one transistor. This system has two inputs and one output.

The transistors are said to be in a "serial" configuration, because they are one after the other (i.e. in a series) between the high tension and the ground.

The tension in the output point is high only when the point is directly connected to the high tension above. This means that both transistors must be conductive, which means that both inputs must be high.

In numerical terms, since we have 2 inputs, we have 4 cases ($4 = 2^2$), and the output is 1 only in one case (input₁ = 1 and input₂ = 1)

| input₁ | input₂ | output |
|--------------------------|--------------------------|---------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

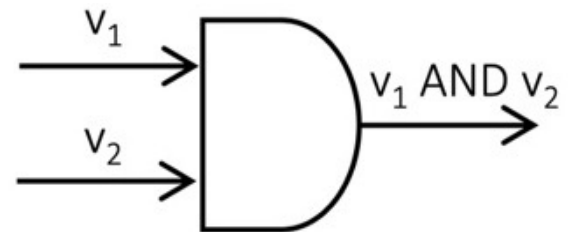


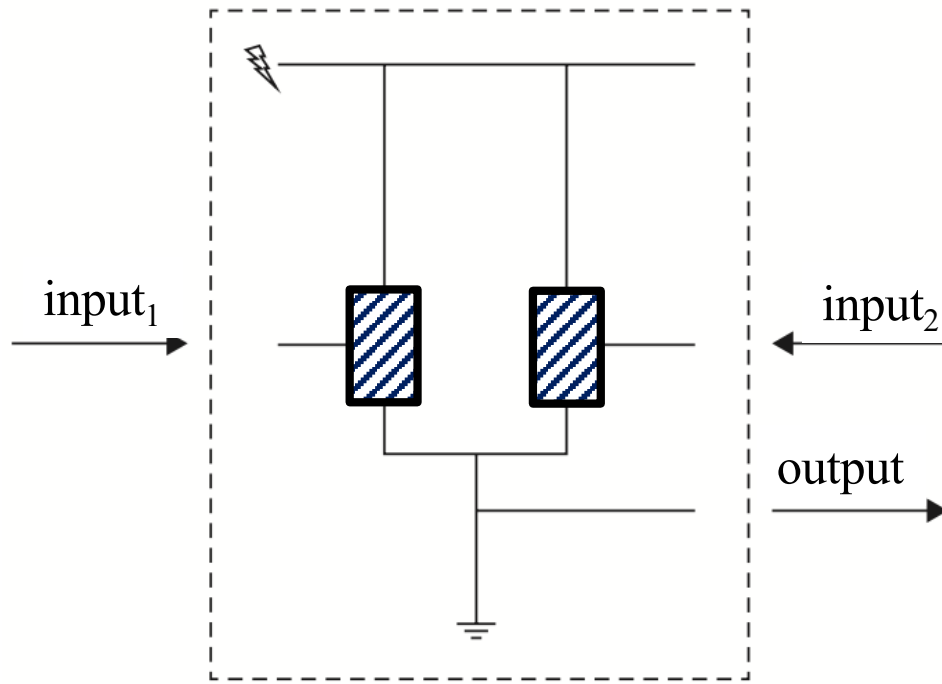
In logical terms, the output is “true” only when both inputs are “false”. In other words, just one false input is enough to make the output false.

| input ₁ | input ₂ | output |
|--------------------|--------------------|--------|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

The two inputs are put together in the same way we use the “AND” conjunction: “the Sun is cold AND one plus one is two” is false, whereas “water is a liquid AND seven is an odd number” is true. This circuit may be seen as the electronic version of the AND logical operator.

This is how an “AND gate” is represented in the graphical standard for circuit design.



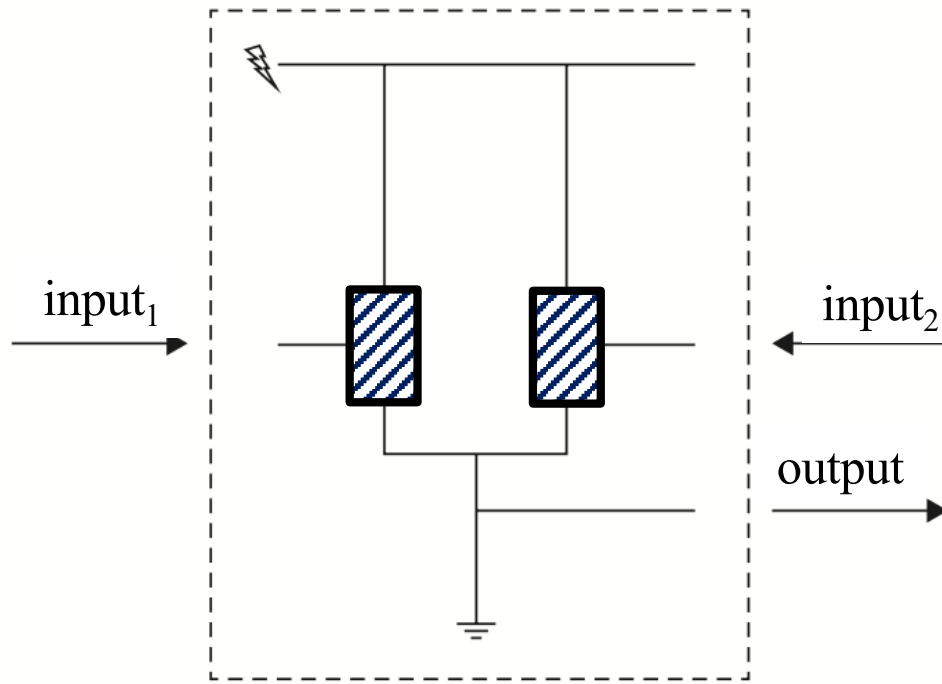


In this other system, the two transistors are said to be in a “parallel” configuration, because they are one next to the other (i.e. on parallel lines) between the high tension and the ground.

The tension in the output point is high in more cases here, because for the output point to be connected to the high tension we just need one transistor to be conductive. This means that only when both transistors are off the output is low tension.

In numerical terms, the output is 0 only in one case ($input_1 = 0$ and $input_2 = 0$)

| input₁ | input₂ | output |
|--------------------------|--------------------------|---------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

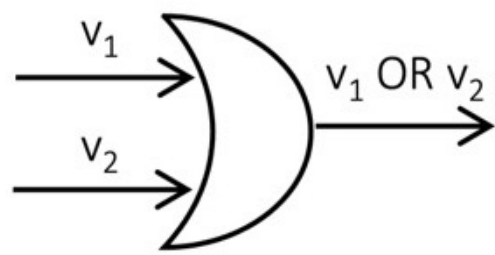


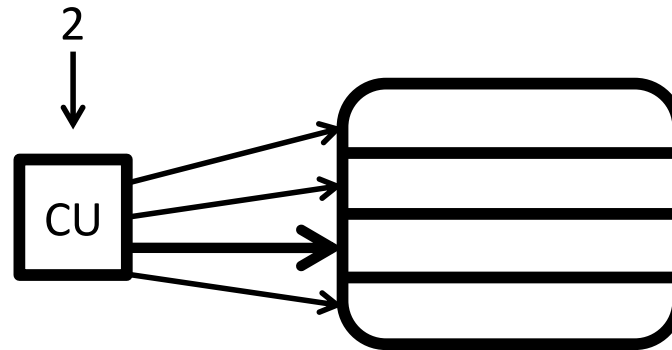
In logical terms, the output is “false” only when both inputs are “false” and “true” in all other cases. We need one true input to make the output true as well.

| input ₁ | input ₂ | output |
|--------------------|--------------------|--------|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

The two inputs are put together in the same way we use the “OR” conjunction (also known as “disjunction”): “the Sun is cold OR one plus one is three” is false, whereas “water is a liquid OR the Earth is flat” is true. This circuit may be seen as the electronic version of the OR logical operator.

This is how an “OR gate” is represented in the graphical standard for circuit design.

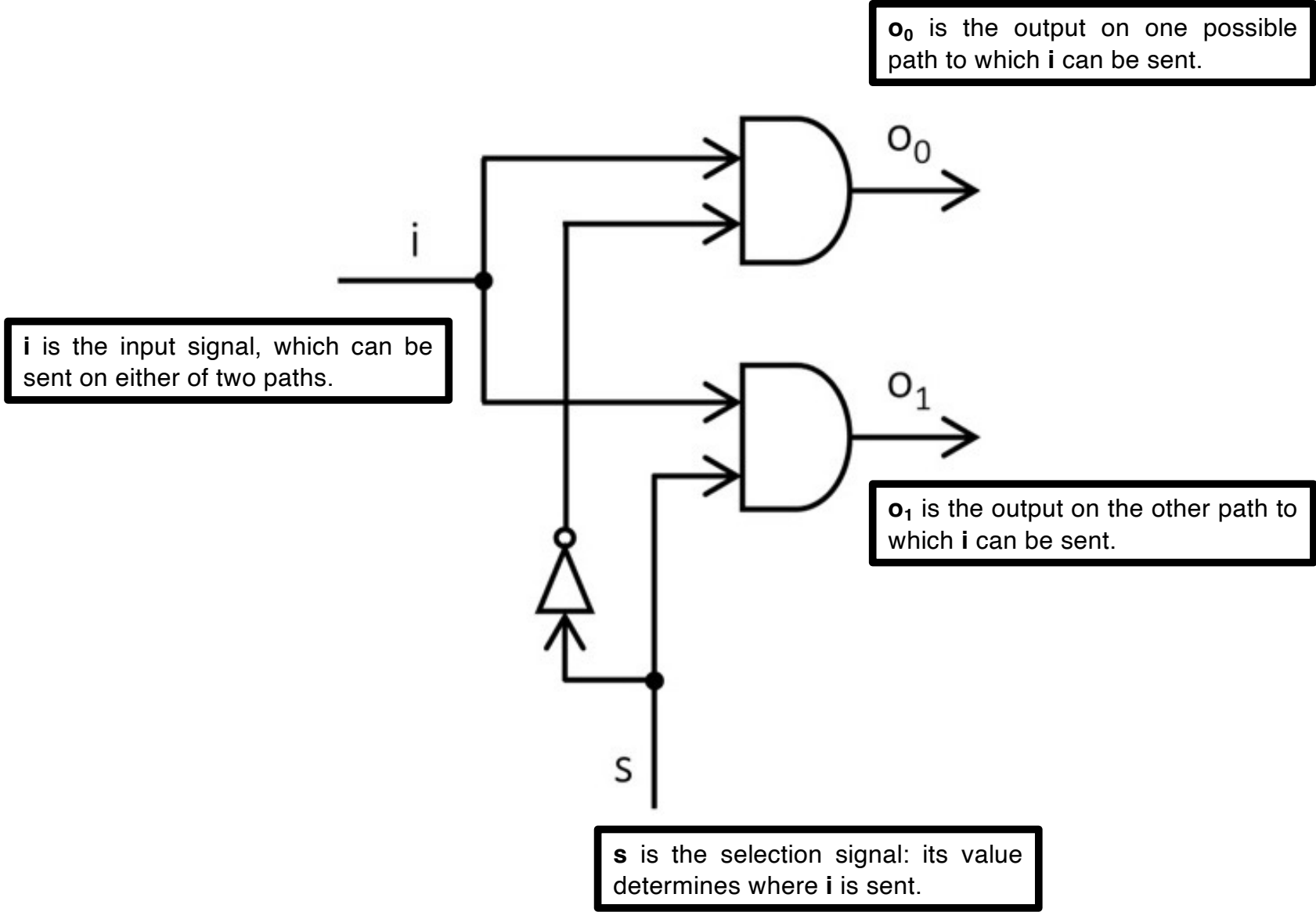




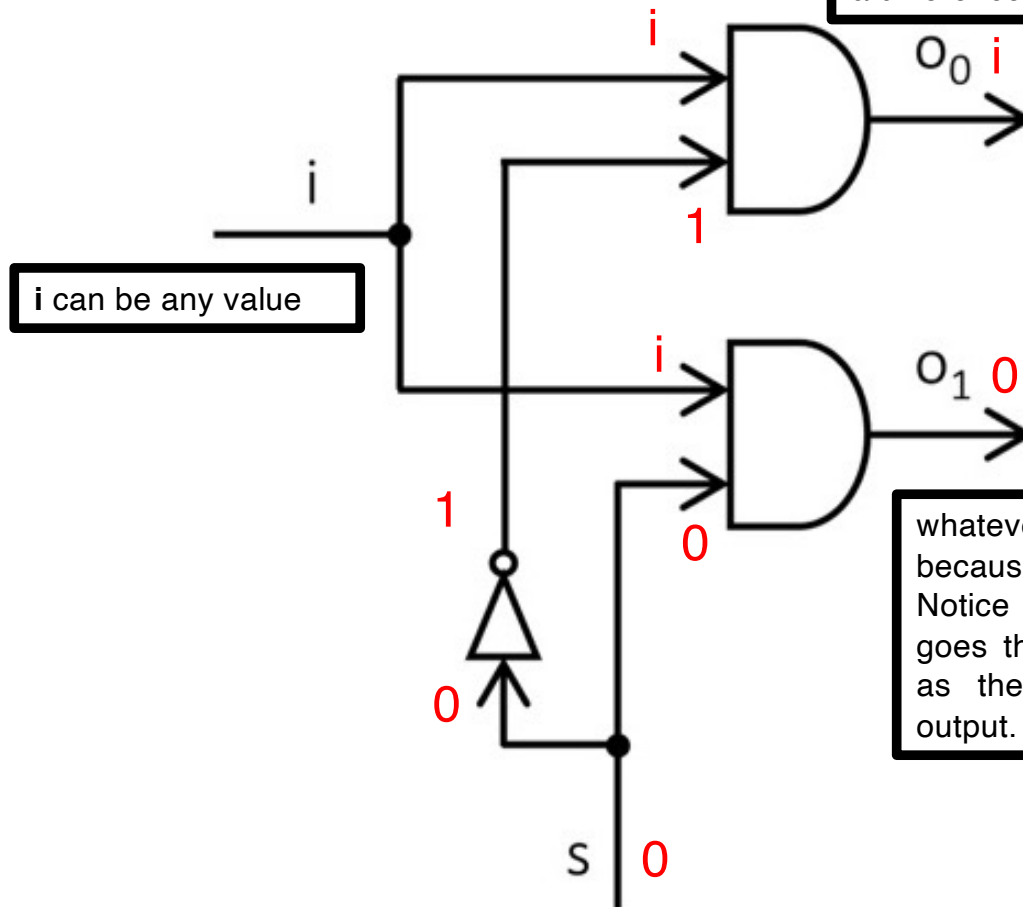
Let's now see how these gates can be combined to operate the selection of a specific path.

Let's focus on a simpler case than the one depicted in the figure, where the CU chooses among 4 different paths.

In what follows, we see how to build a circuit to choose between 2 paths.



if s is 0...



i can be any value

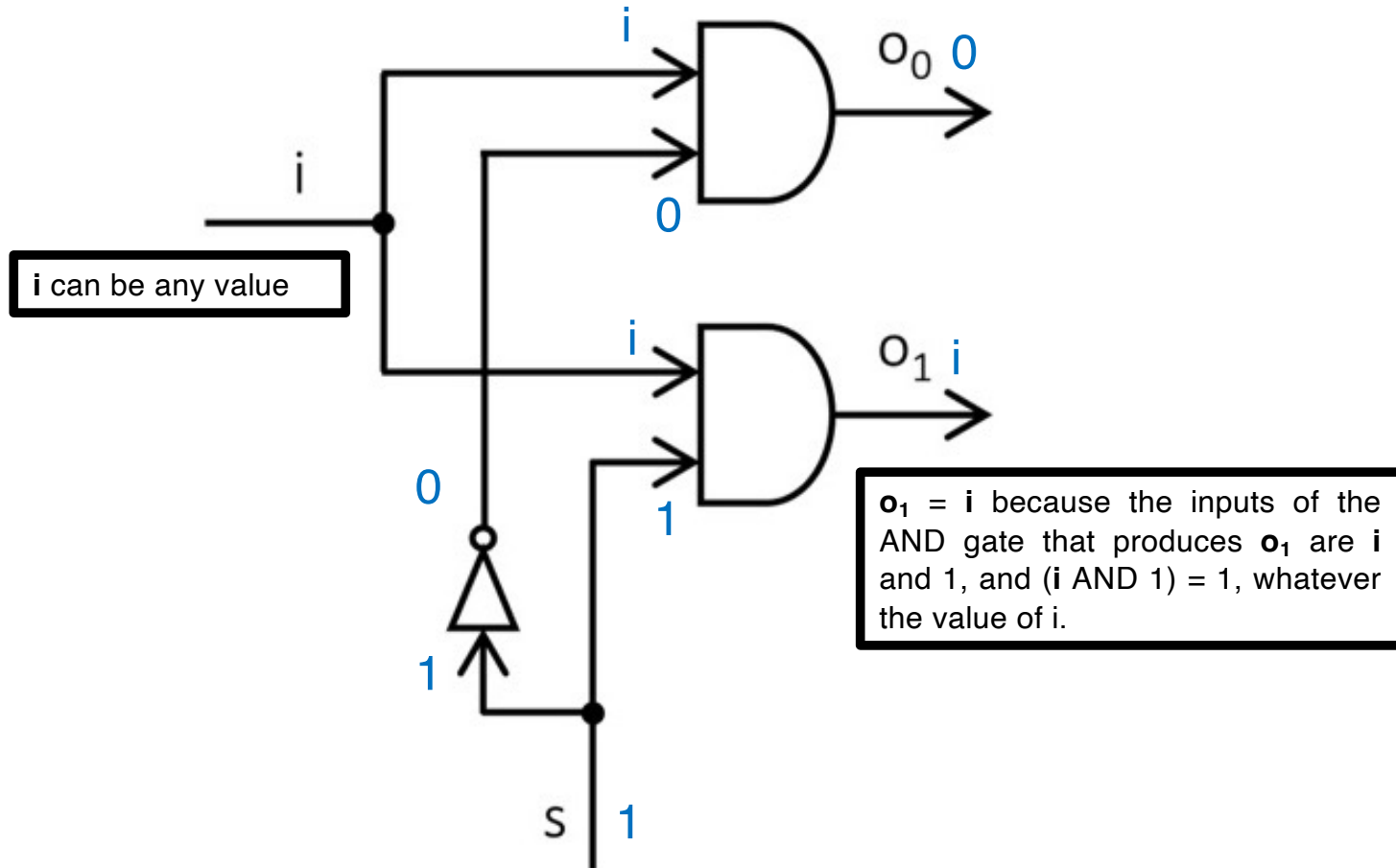
o_0 is the same as i because the other input of the AND gate is 1. Notice that i AND 1 is the same as i (whether i is 0 or 1 does not make a difference).

whatever the value of i , o_1 is 0 because one of its inputs (s) is 0. Notice that any input value that goes through an AND gate with 0 as the other input yields 0 as output.

...then $o_0 = i$.
In other words, i is sent to the first path.

if s is 1...

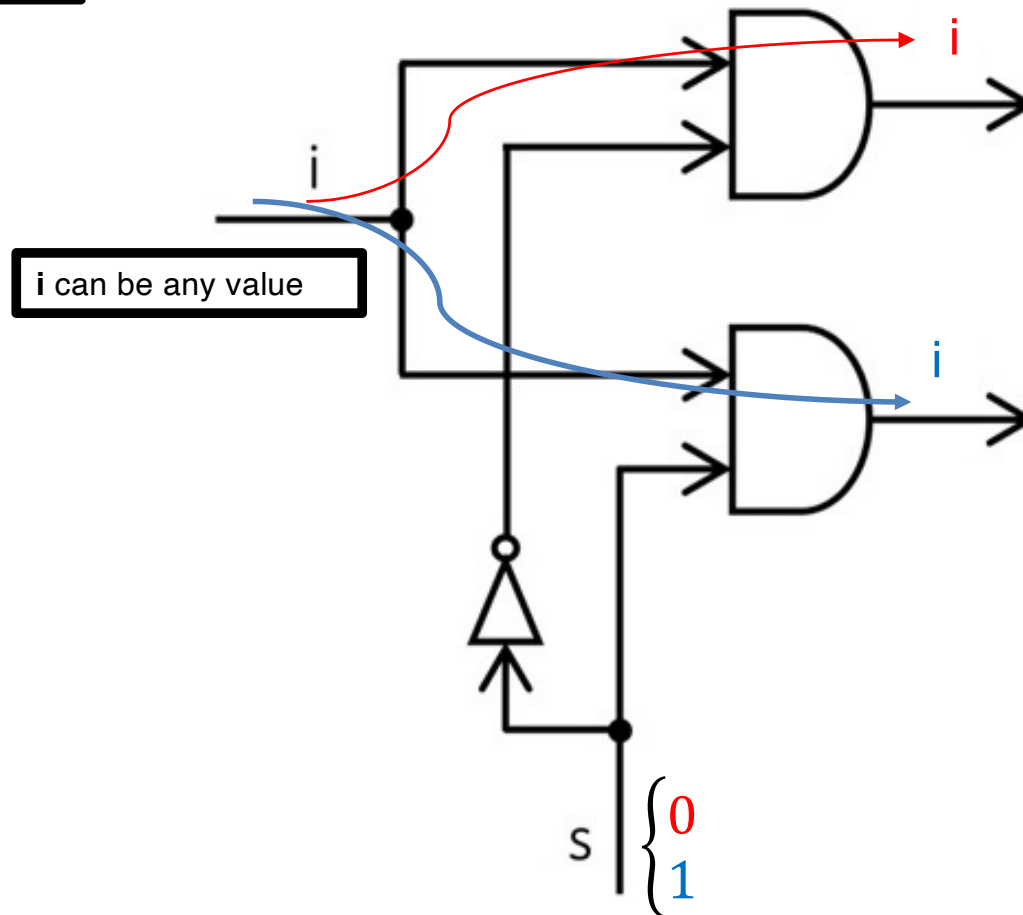
o_0 is 0 no matter the value of i , because the other input of the AND gate is 0.



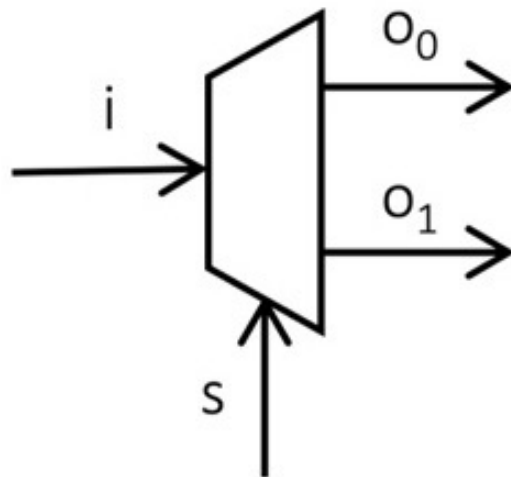
$o_1 = i$ because the inputs of the AND gate that produces o_1 are i and 1 , and $(i \text{ AND } 1) = i$, whatever the value of i .

...then $o_1 = i$.
In other words, i is sent to the second path.

This means...

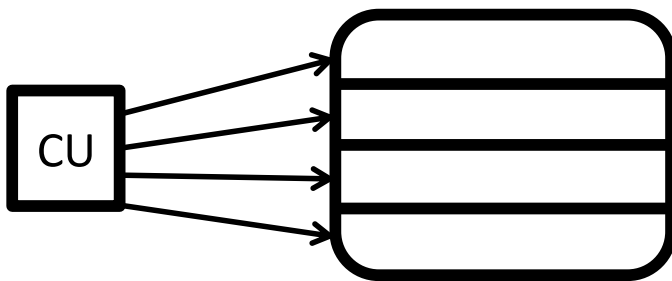


...that it is as if s selects where i goes.



This is the circuit design style for representing such a system, which is called “demultiplexer” (also known as “DEMUX”).

Since the choice is between 2 paths, this one is called a 1:2 DEMUX. In a 1:2 DEMUX, the selection signal is just 1 bit.



In our original example, the choice is between 4 paths, so we actually need a 1:4 DEMUX, with 2 bits for the selection signal (s_0 and s_1 , enabling the selection among 4 paths: o_{00} , o_{01} , o_{10} , and o_{11}). The electronics is a bit more complicated, but the principles are the same as the ones guiding the design of the 1:2 DEMUX.

The story so far

- With all these considerations around logic, and the construction of electronic circuits that embody logical operators like NOT, AND, and OR, we have focused on the L in ALU
- Moreover, we have combined NOT and AND gates to create a DEMUX device, to enable the selection of a path among many, by which the CU can send signals where they are needed
- But what about the A in ALU?
- Aren't computer built to compute, that is, do arithmetics after all?

Back to basics of arithmetic

- Numbers work as input for arithmetic operations no matter the numerical system they are expressed in
- We can do $3 + 4$ and obtain 7 (base 10)
- In the same way, we can do $011 + 100$ and obtain 111 (base 2)

Simple arithmetics in binary

- Given two bits in input, the rules for adding them are very simple:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10, \text{ that is } 0 \text{ with carry } 1$$

- Since $+$ applies to 2 bits and yields 1 bit in output (possibly with a carry) just like AND and OR, we can imagine to build a system with logical gates that manipulate bits in a way that coincides with what $+$ does

Addition in electronics

- Let's ignore the carry in the last case for now, and focus on what happens with the bits:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0$$

The output of + is almost identical to the output of OR

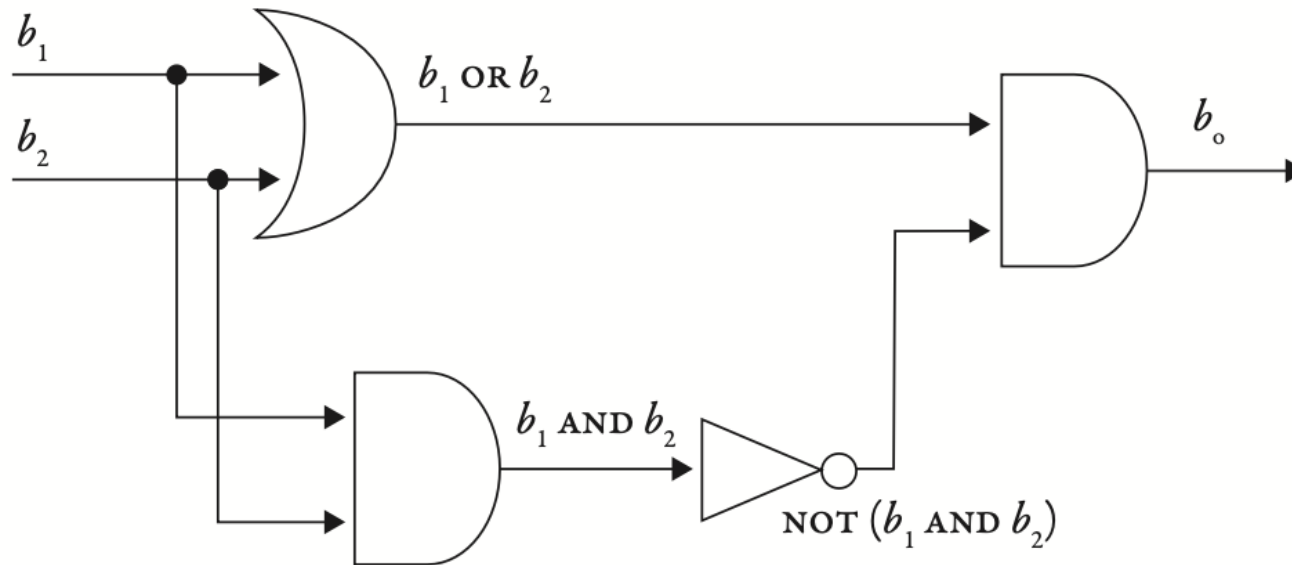
| input ₁ | input ₂ | output |
|--------------------|--------------------|--------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Except for the last case, where both input bits are 1 and the output is 0 instead of 1.

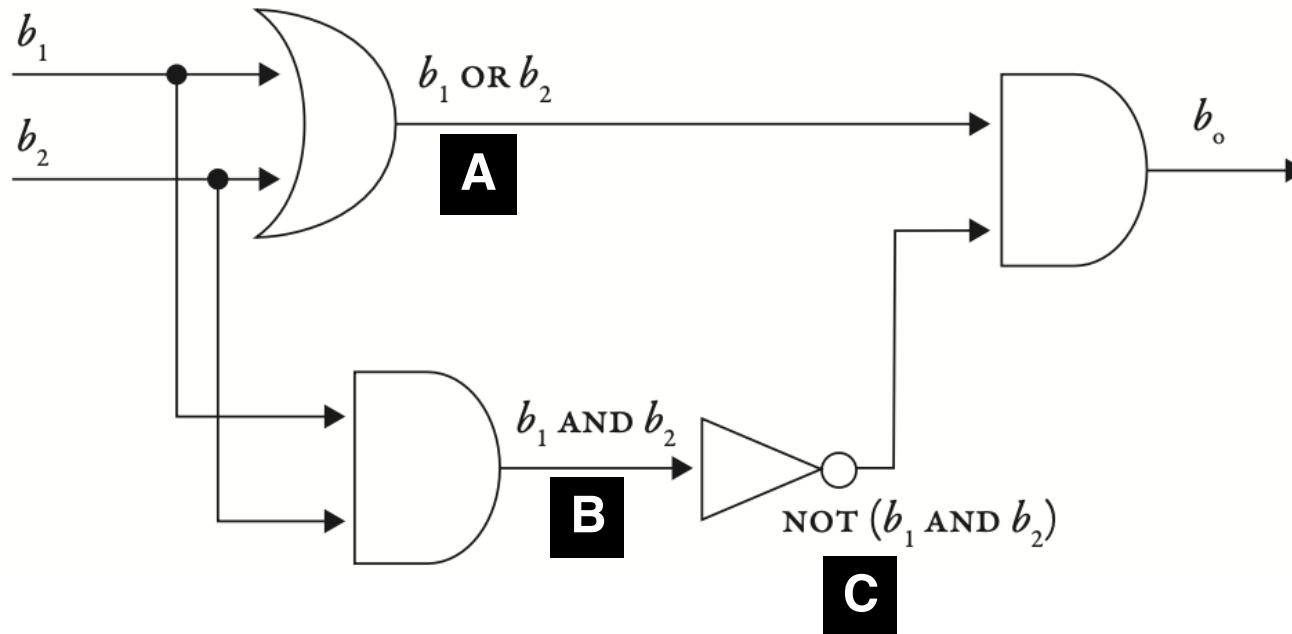
So, given two bits in input, b_1 and b_2 , $b_1 + b_2$ is like b_1 OR b_2 but not if they are both 1.

Given two bits in input, b_1 and b_2 , $b_1 + b_2$ is like b_1 OR b_2 but not if they are both 1.

To design a system that electronically realizes this behavior, we can combine the outputs of:
 b_1 OR b_2
AND
NOT (b_1 AND b_2),
as shown in the system in the figure (taken from "Che cos'è un computer" by Mario Verdicchio, published by Carocci, 2023).



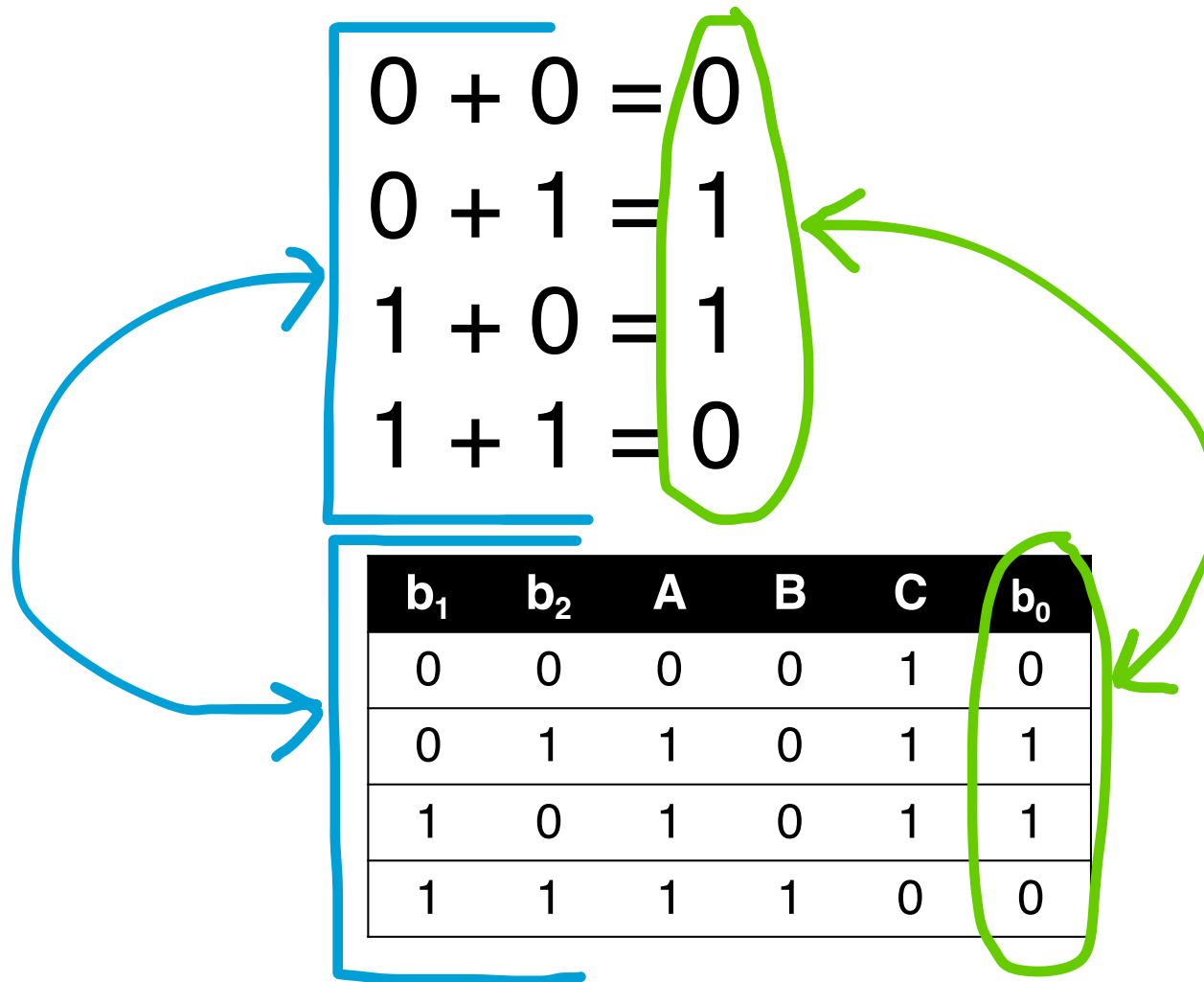
Let's see how this system works in the following slide.



| b_1 | b_2 | A | B | C | b_0 |
|-------|-------|----------|----------|----------|-------|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

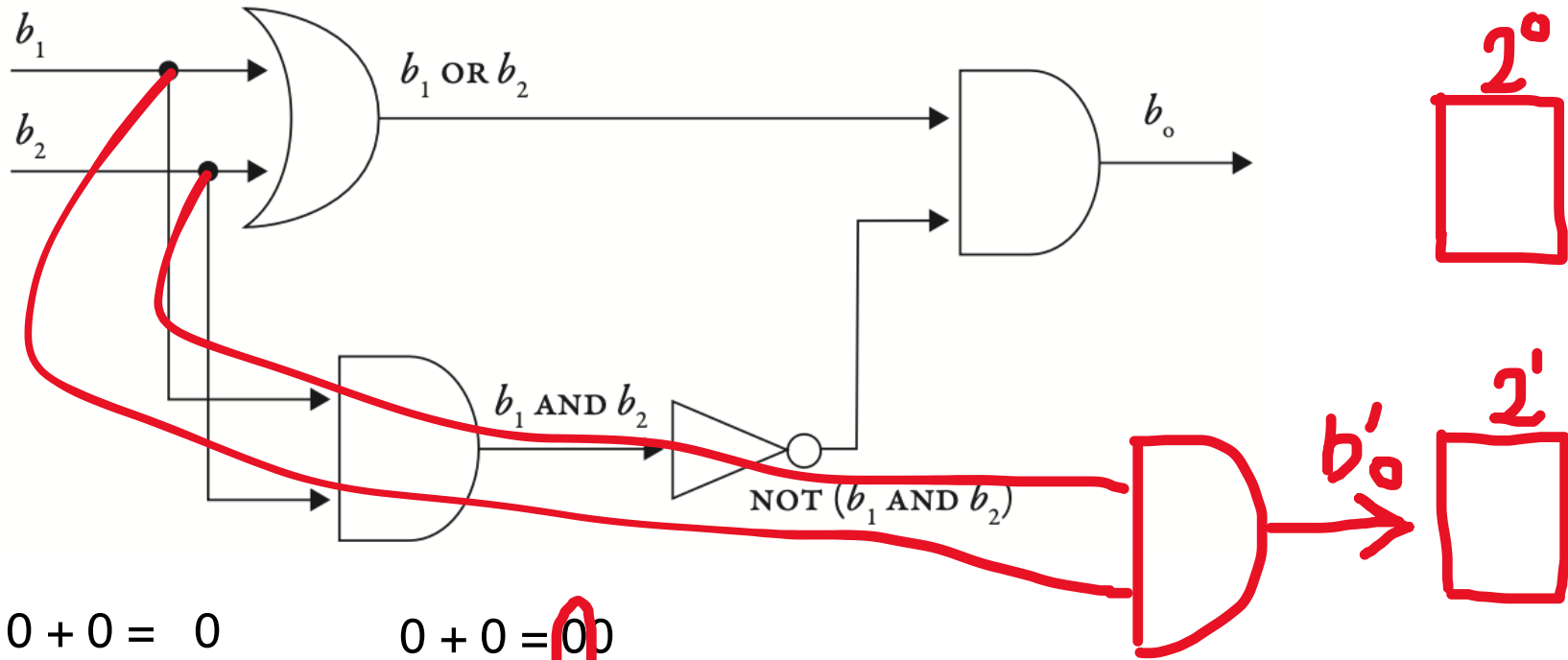
The system uses the inputs b_1 and b_2 to compute $(b_1 \text{ OR } b_2)$ (which we called **A**) on the one side, and the negation of $(b_1 \text{ AND } b_2)$ on the other. The negation of $(b_1 \text{ AND } b_2)$ is computed by means of an AND gate (obtaining **B**) and then a NOT gate (obtaining **C**).

The final computation, which gives the output, combines **A** and **C** with an AND, which corresponds to setting the final result to 1 in all cases when $(b_1 \text{ OR } b_2)$ is 1, except for when $(b_1 \text{ AND } b_2)$ is 1: in that case **C** is 0 and, through the last AND gate, sets the final result to 0.



That electronic circuit, comprised of one OR gate, one NOT gate, and two AND gates combined as previously shown, yields the same output as the addition applied to two single-bit inputs. Thus, it can work as an ADDER.

From a physical perspective, it is just a circuit that manipulates electric tensions, but with the fundamental encoding in mind, we can see its (electric) operation as an (arithmetic) operation of addition.



$0 + 0 = 0$
 $0 + 1 = 1$
 $1 + 0 = 1$
 $1 + 1 = 10$

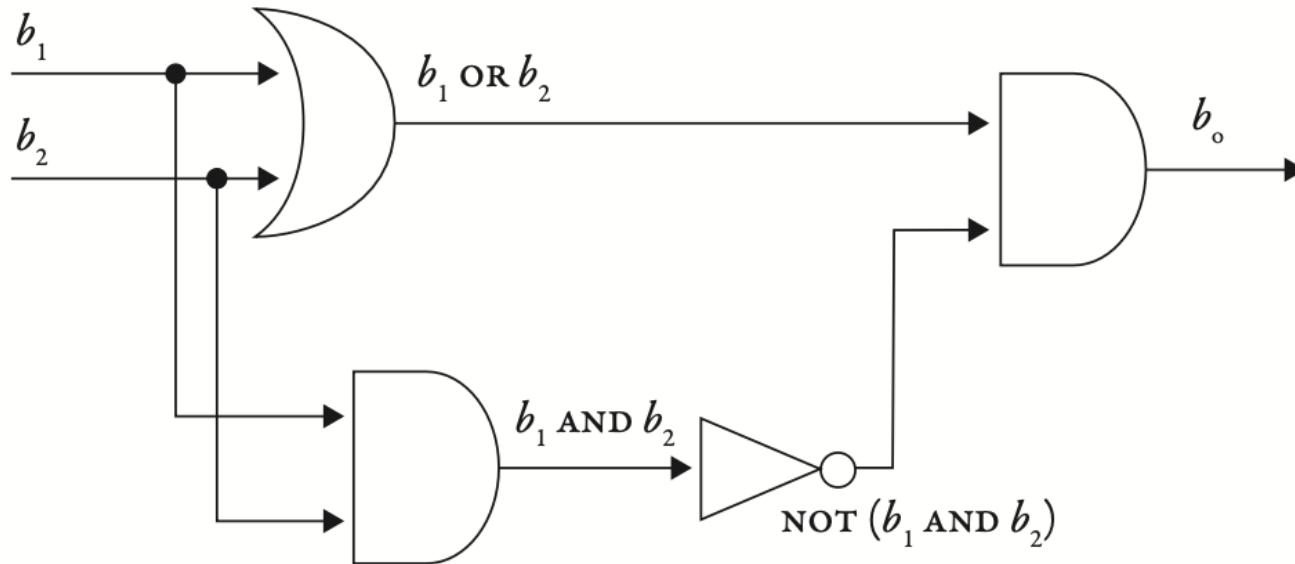
$0 + 0 = 00$
 $0 + 1 = 01$
 $1 + 0 = 01$
 $1 + 1 = 10$

We must not forget about the carry.

The carry is another bit that is part of the final result. It is 0 in the first 3 cases, and 1 in the last. The output bits are exactly like the output of $(b_1 \text{ AND } b_2)$.

So to build the circuit that computes the carry, we just need to put the two inputs through an AND gate.

The bit in output from the first system (in black, printed) is saved in a 1-bit memory space that represents how many units (2^0) are in the final result. The bit in output from the second system (in red, handwritten) is saved in another memory unit, indicating how many twos (2^1) are in the final result.



Since we have built an arithmetic system by combining a logical system of AND, OR, and NOT, does this mean that arithmetic is based on logic?

Absolutely not. In the real world, arithmetic and logic have a very distinct origin. Actually, since counting with fingers came before formalizing reasoning, if anything, arithmetic should be considered the basis of a way of thinking that lead humanity to conceive logic at a later stage, but these are questions for historians and philosophers of science.

It is just that, in the material world of electronic circuits built by means of transistors, logical operators are much simpler to implement than arithmetic operations. You can see it by noticing that we need only one transistor to build a NOT gate, whereas an ADDER has a much more complex structure.

Computer Science gives us extremely useful devices, but those devices are the result of discoveries in science (e.g. semiconductors) and socio-political agreements (e.g. encodings like RGB, standards like JPG). They should not be taken as an indication of some fundamental principles regarding how the universe works or how our thinking and reasoning work.

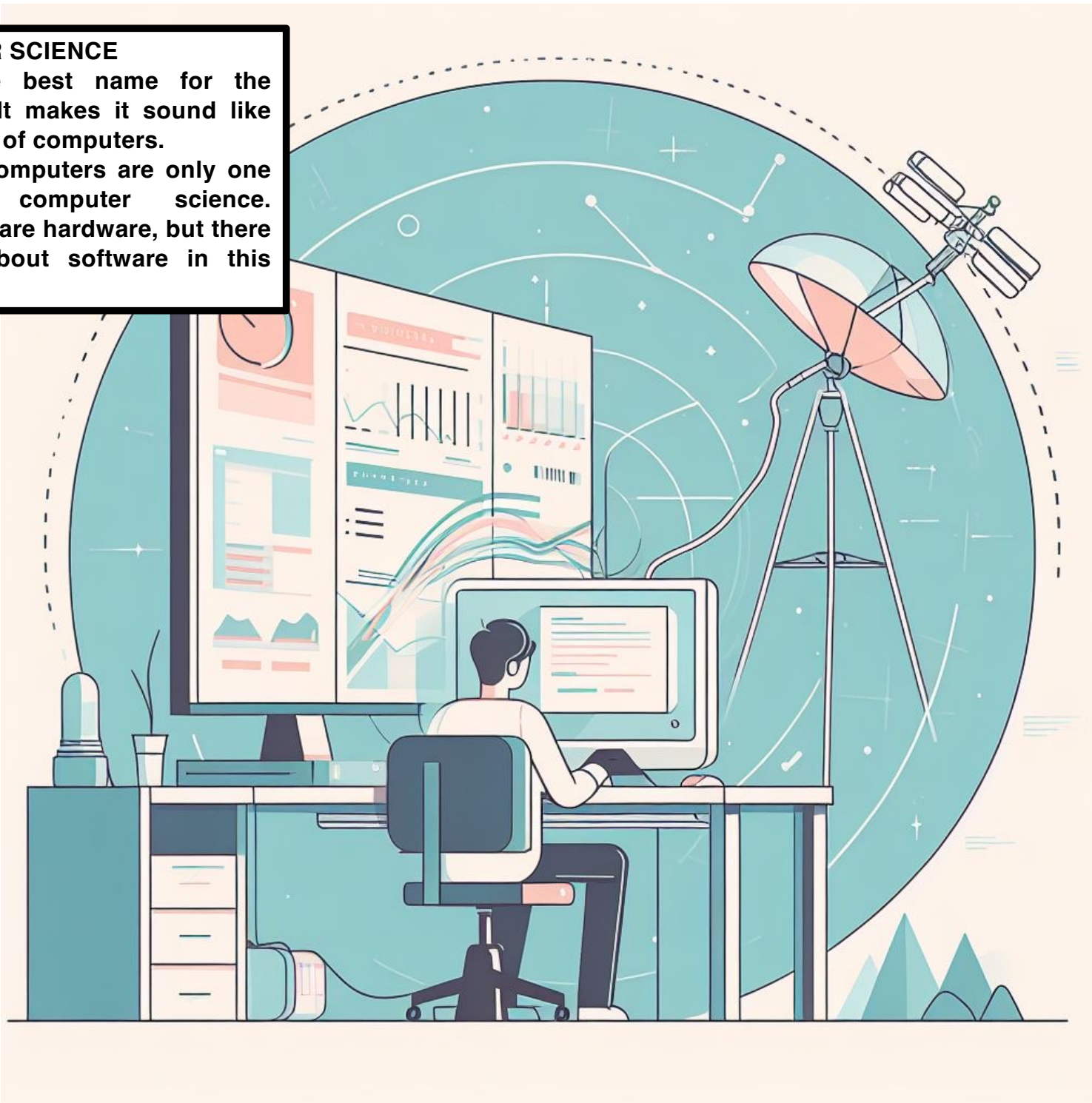
And yet, there are some subfields of this discipline whose experts forget about this. You must not.

Let's take a look at the most important subfields of Computer Science.

COMPUTER SCIENCE

is not the best name for the discipline. It makes it sound like the science of computers.

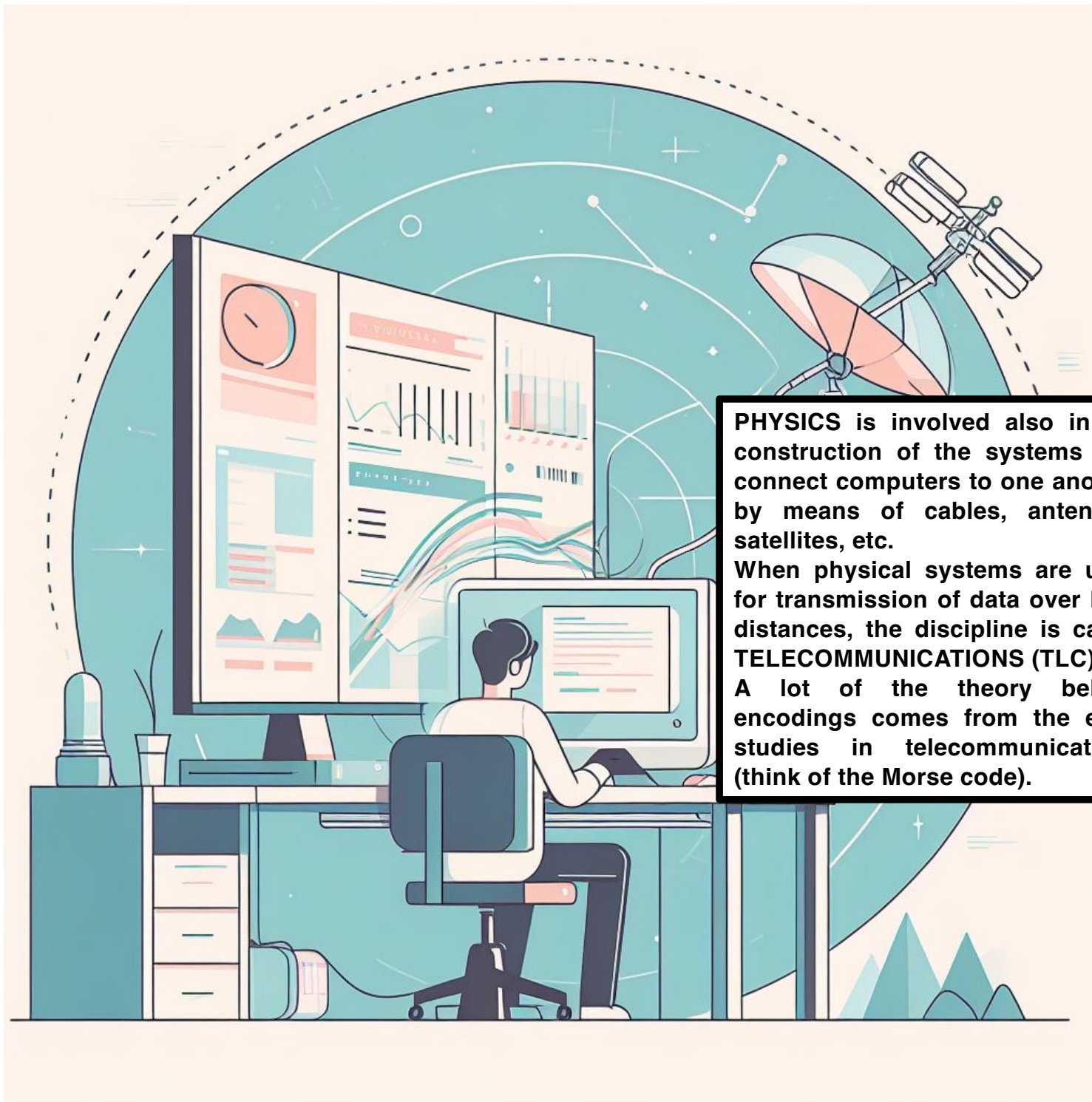
However, computers are only one part of computer science. Computers are hardware, but there is a lot about software in this discipline.



Indeed, the subfields that deal with hardware and the ones that are the farthest from Computer Science, because they may even be considered entirely different fields.

We have **ELECTRONICS** that deals with the design and construction of the circuits computers are made of, and let's not forget **PHYSICS** that allowed us to discover semiconductors.

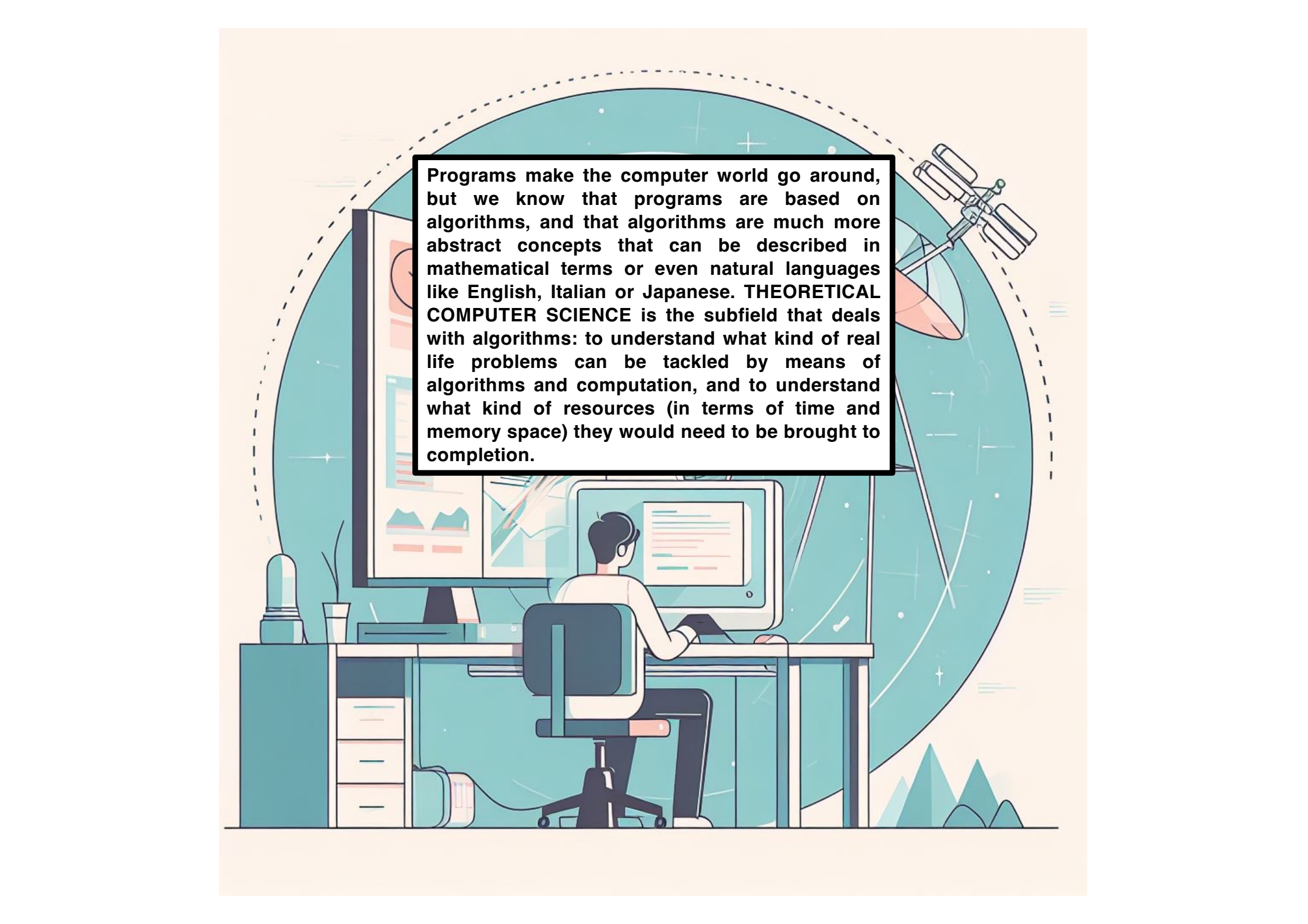




PHYSICS is involved also in the construction of the systems that connect computers to one another by means of cables, antennas, satellites, etc. When physical systems are used for transmission of data over long distances, the discipline is called **TELECOMMUNICATIONS (TLC)**. A lot of the theory behind encodings comes from the early studies in telecommunications (think of the Morse code).



It is great to build computers, but they exist to be used.
SOFTWARE ENGINEERING (SE) is the subfield of computer science that is aimed at the creation of software, in terms of programs and programming languages (artificial languages used to write programs).

An illustration of a person sitting at a desk working on a computer. The person is wearing a headset and is viewed from behind. The desk has a large monitor displaying code, a keyboard, and a mouse. To the left of the desk is a water bottle and a glass. The background is a stylized space scene with a large teal planet, a satellite in orbit, and some stars. A dashed white circle outlines the planet. The overall style is clean and modern with a teal and orange color palette.

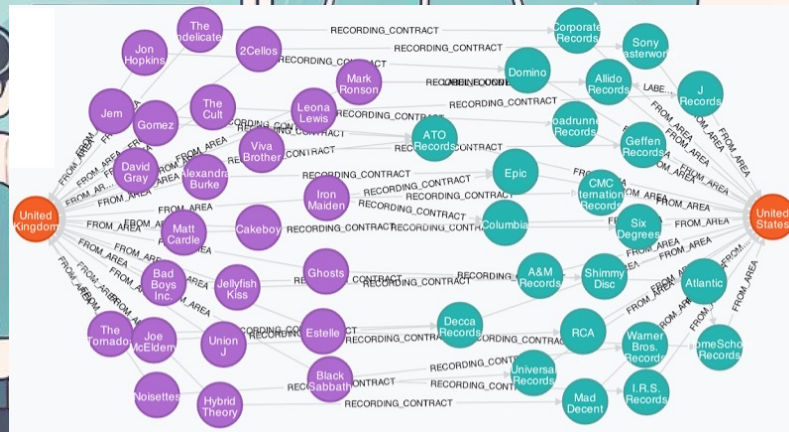
Programs make the computer world go around, but we know that programs are based on algorithms, and that algorithms are much more abstract concepts that can be described in mathematical terms or even natural languages like English, Italian or Japanese. THEORETICAL COMPUTER SCIENCE is the subfield that deals with algorithms: to understand what kind of real life problems can be tackled by means of algorithms and computation, and to understand what kind of resources (in terms of time and memory space) they would need to be brought to completion.

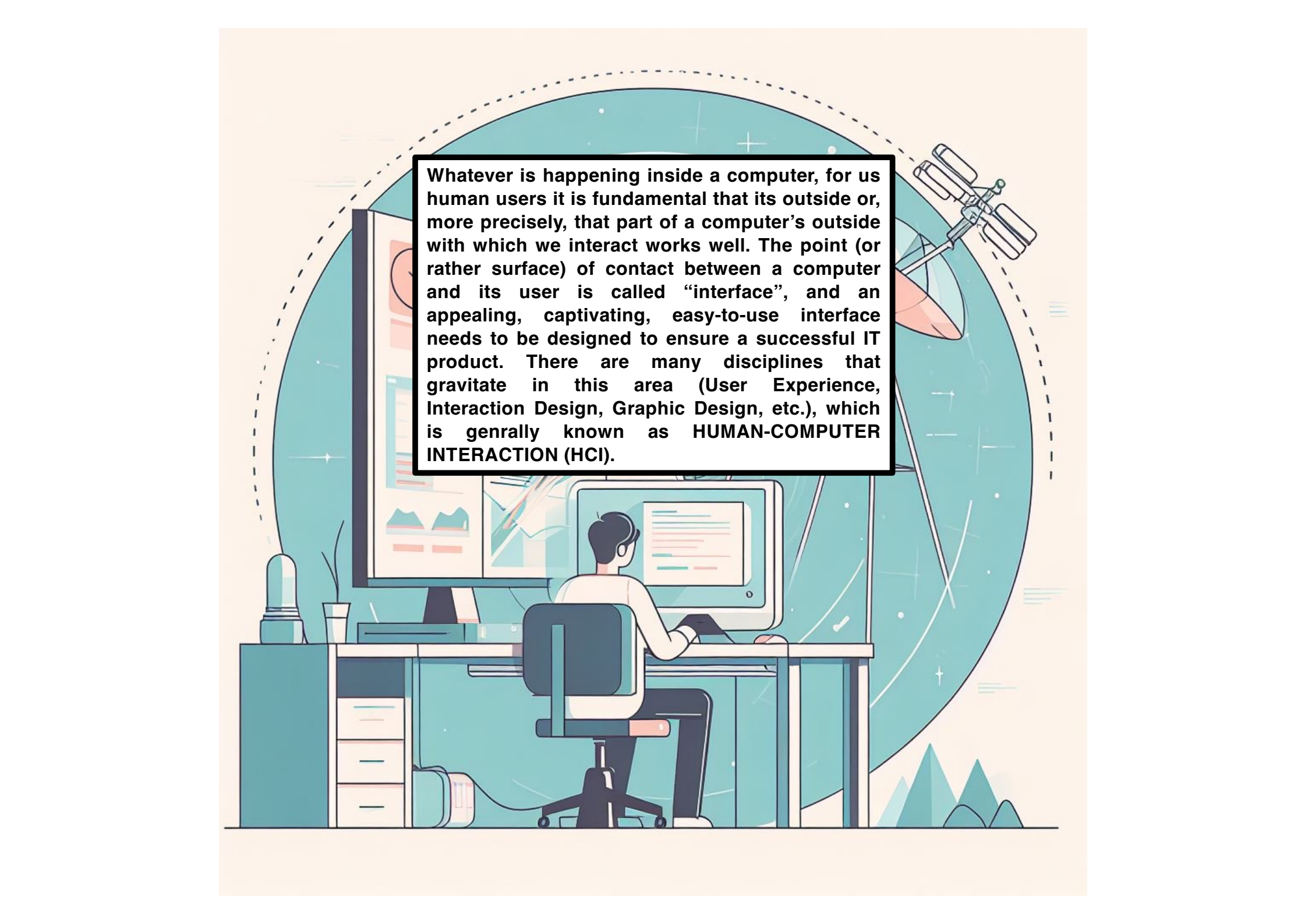
The connectivity ensured by telecommunications and the Internet has notably increased the quantity of data that computers have to deal with. DATABASE MANAGEMENT (DBMS) is a traditional subfield that is aimed at the organization and analysis of the data inside a computer. Typically, data are organized in the form of tables (also known as relational databases), but the more and more dynamic nature of data coming from the Internet is pushing for more flexibility, and in many systems a graph-like structured is preferred over tables (non-relational) for databases. Searches (like when you google stuff you don't know) and big data (massive amounts of data that are collected not only through the Internet but also by means of digital devices like smartphones, smartwatches, credit cards) are strongly emerging as key topics in this context, up to the point hat they might become independent subfields themselves.

Table in a database

| First Name | Last Name | Address | City | Age |
|------------|-----------|---------------------|----------|-----|
| Mickey | Mouse | 123 Fantasy Way | Anaheim | 73 |
| Bat | Man | 321 Cavem Ave | Gotham | 54 |
| Wonder | Woman | 987 Truth Way | Paradise | 39 |
| Donald | Duck | 555 Quack Street | Mallard | 65 |
| Bugs | Bunny | 567 Carrot Street | Rascal | 58 |
| Wiley | Coyote | 999 Acme Way | Canyon | 61 |
| Cat | Woman | 234 Purrfect Street | Hairball | 32 |
| Tweety | Bird | 543 | Itotttaw | 28 |

Graph-based database



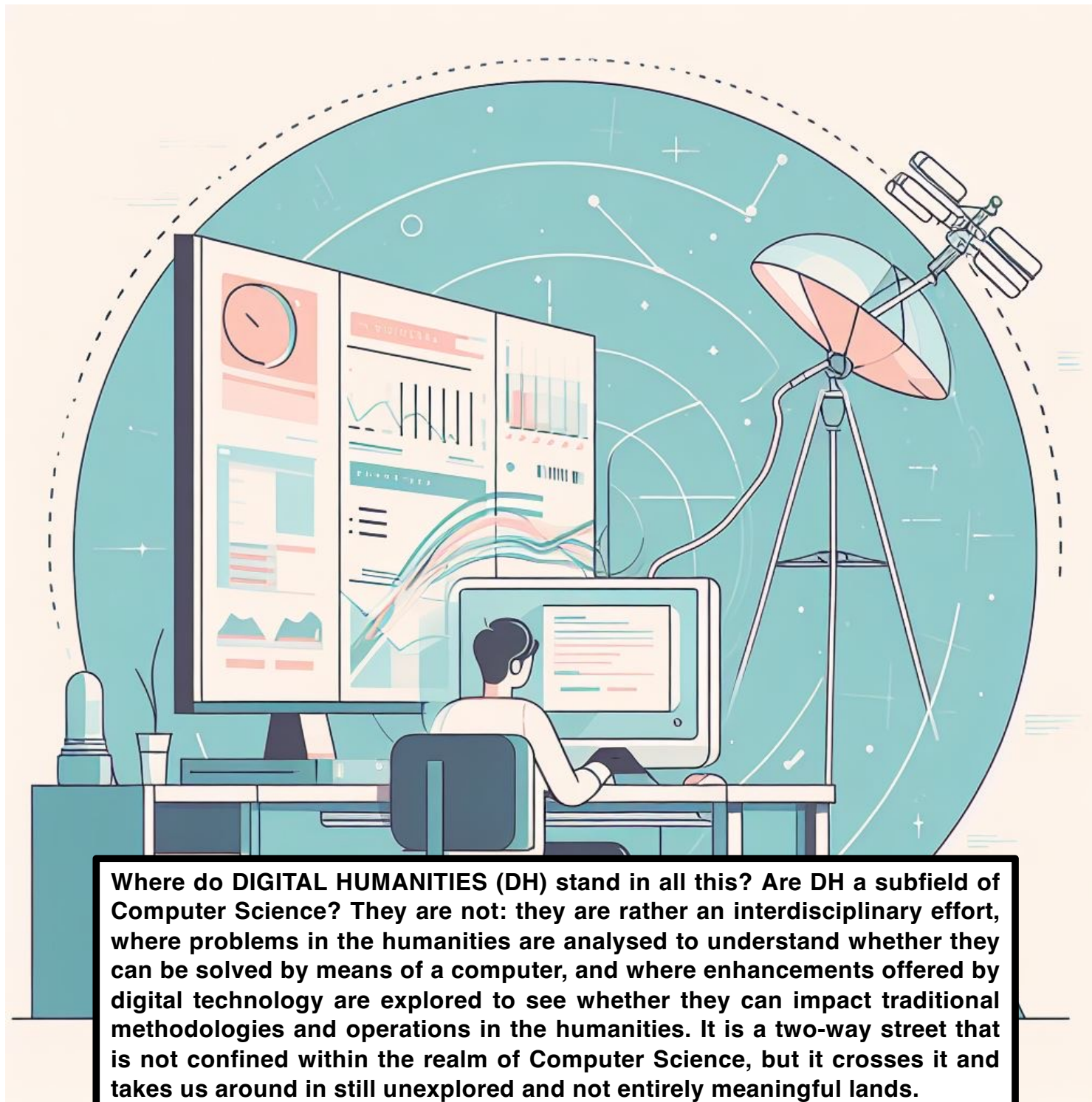
An illustration of a person sitting at a desk working on a computer. The person is wearing a headset and is viewed from behind. The desk has a large monitor displaying a webpage, a keyboard, and a mouse. To the left of the desk is a water bottle and a glass. The background is a stylized space scene with a large teal planet, a satellite with solar panels, and stars. A dashed white line forms a circle around the planet. A white text box with a black border is overlaid on the scene, containing text about Human-Computer Interaction (HCI).

Whatever is happening inside a computer, for us human users it is fundamental that its outside or, more precisely, that part of a computer's outside with which we interact works well. The point (or rather surface) of contact between a computer and its user is called "interface", and an appealing, captivating, easy-to-use interface needs to be designed to ensure a successful IT product. There are many disciplines that gravitate in this area (User Experience, Interaction Design, Graphic Design, etc.), which is generally known as HUMAN-COMPUTER INTERACTION (HCI).

Finally, the hottest subfield of the moment: **ARTIFICIAL INTELLIGENCE (AI)**. In its traditional form (it was born in the 1950s), AI was about “automated reasoning”, that is, trying to capture in computational form the rules of logic that sustain human reasoning, to make computers do the reasoning. After decades of minor successes and major failures, another form of AI emerged, called “machine learning” (ML), which has a radically different approach: no more formal logical reasoning, but statistical analysis of great quantities of data, in search for correlations and recurrent patterns. ML is considered a major success because it just works, mainly thanks to the abundant quantities of data available on the Internet, which ML-programmed computers can analyse. The most successful applications are image analysis and classification (e.g. in the medical field to detect cancer from x-ray images), or text generation (e.g. ChatGPT) and image generation (e.g. Midjourney , DALL-E).



This background image itself is the output of DALL-E, to which this input was given: “image in neat minimal graphical style with pastel colors depicting a person working in front of a desktop computer with a big screen showing the interface of a very complex word processing software, while the computer is connected to a cable that goes outside the room, connected to a parabolic antenna.”



Where do DIGITAL HUMANITIES (DH) stand in all this? Are DH a subfield of Computer Science? They are not: they are rather an interdisciplinary effort, where problems in the humanities are analysed to understand whether they can be solved by means of a computer, and where enhancements offered by digital technology are explored to see whether they can impact traditional methodologies and operations in the humanities. It is a two-way street that is not confined within the realm of Computer Science, but it crosses it and takes us around in still unexplored and not entirely meaningful lands.