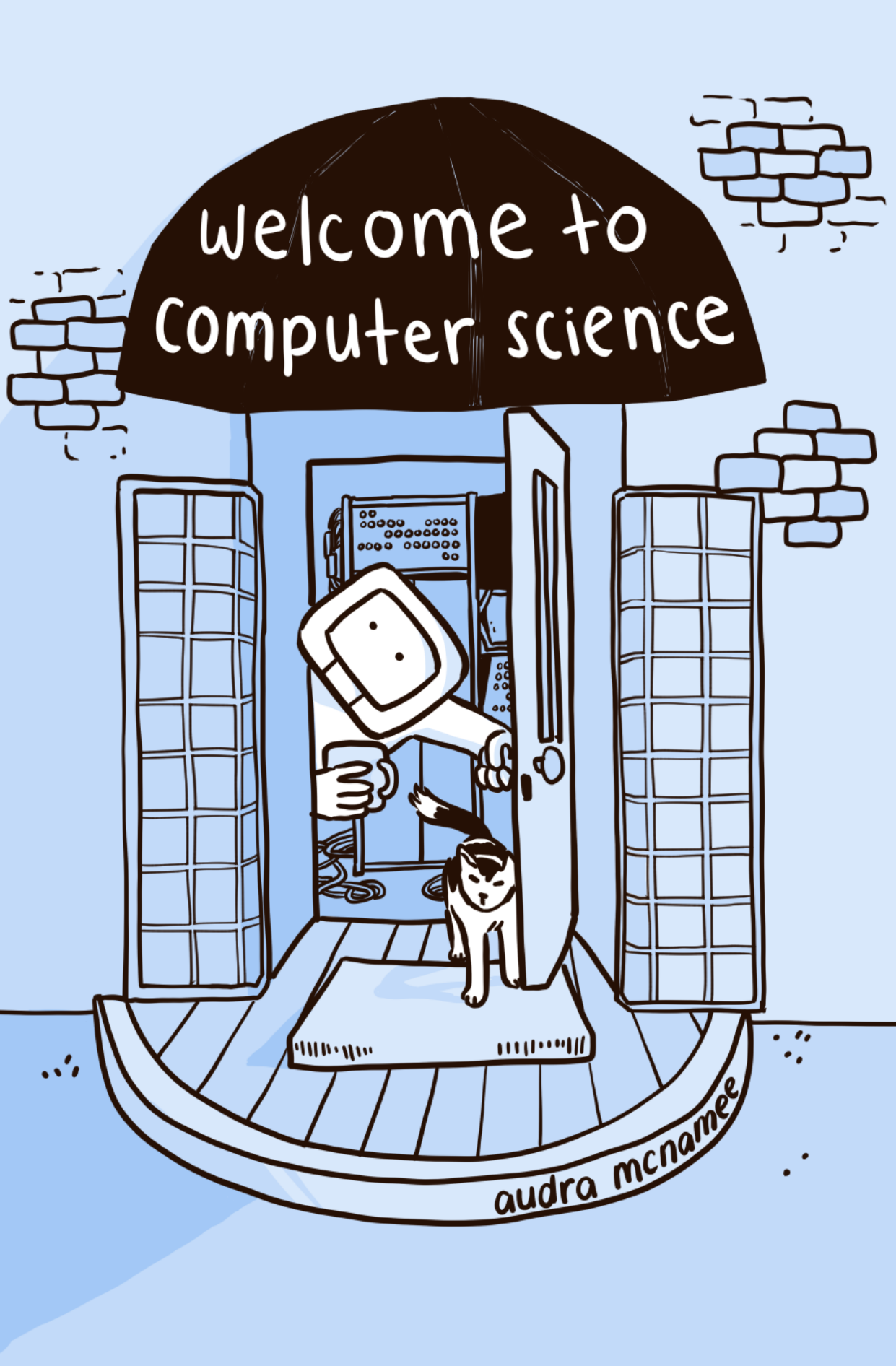
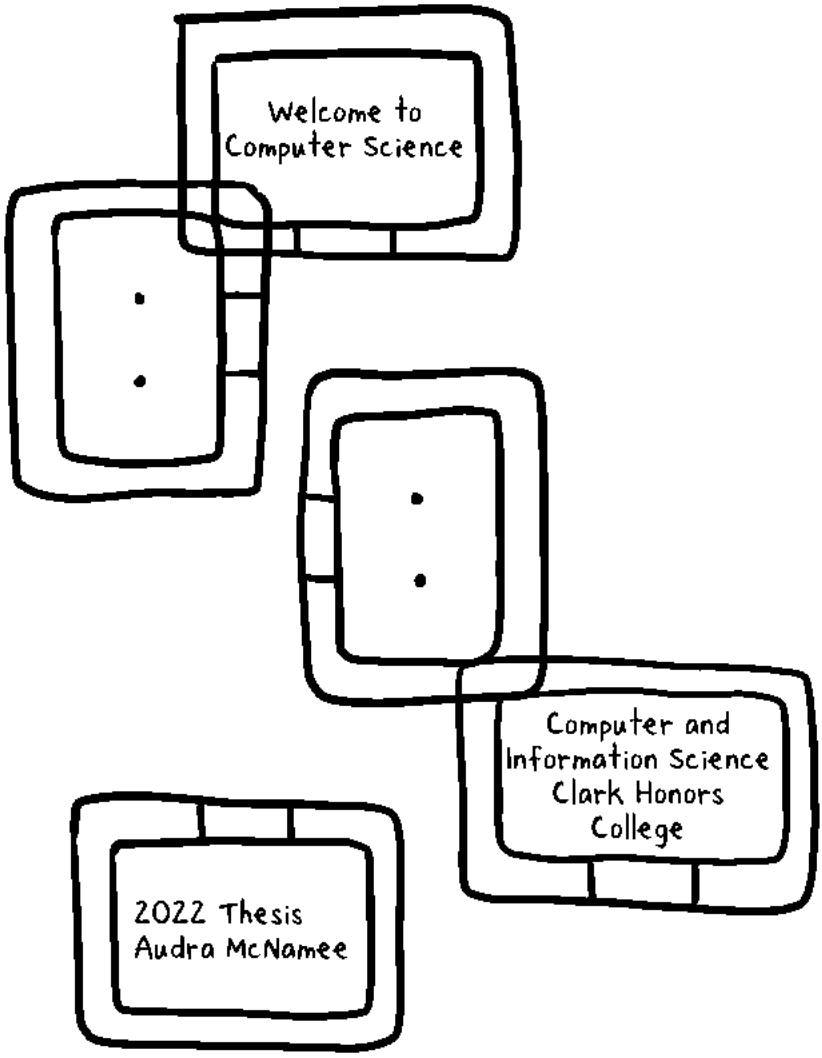
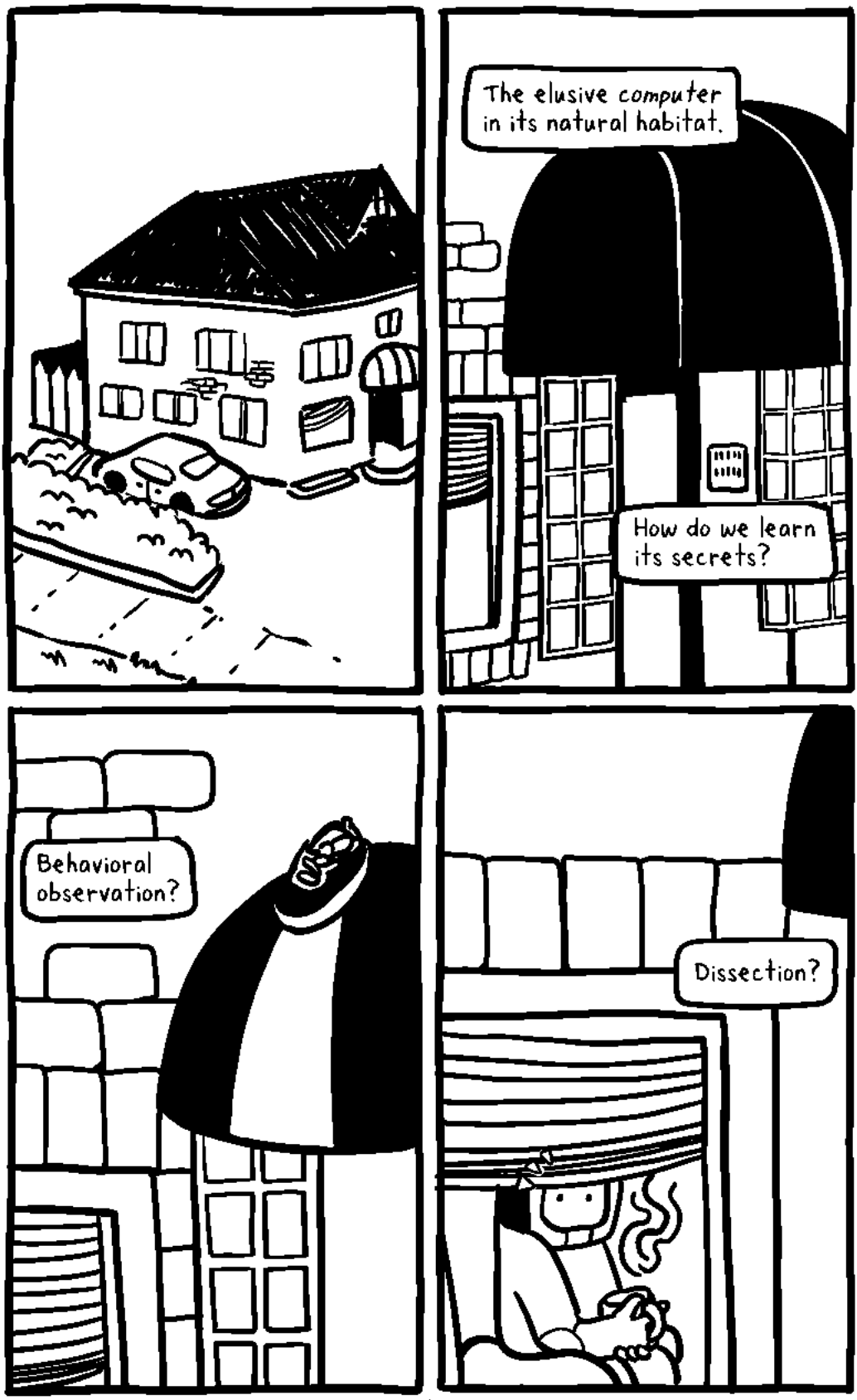
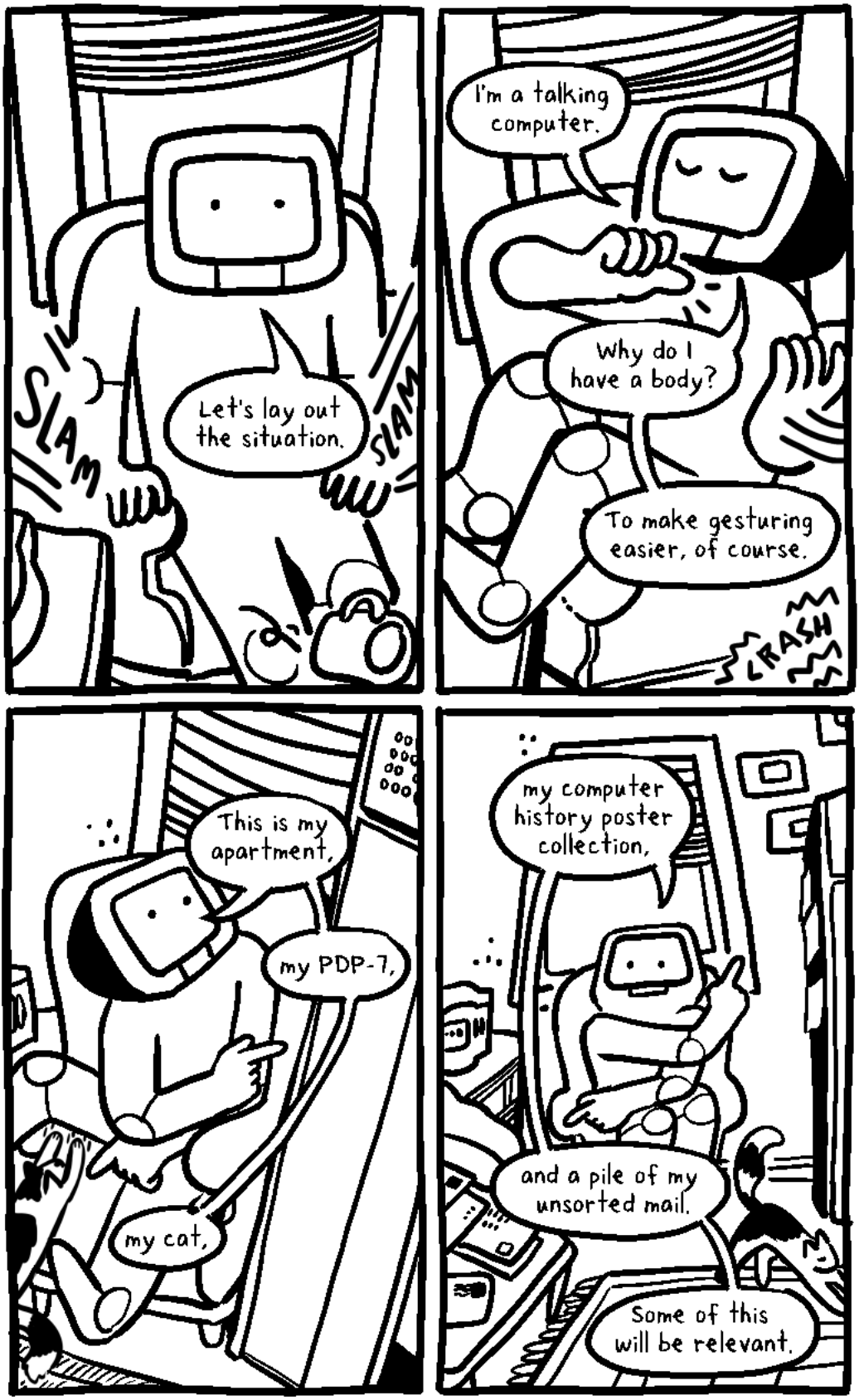
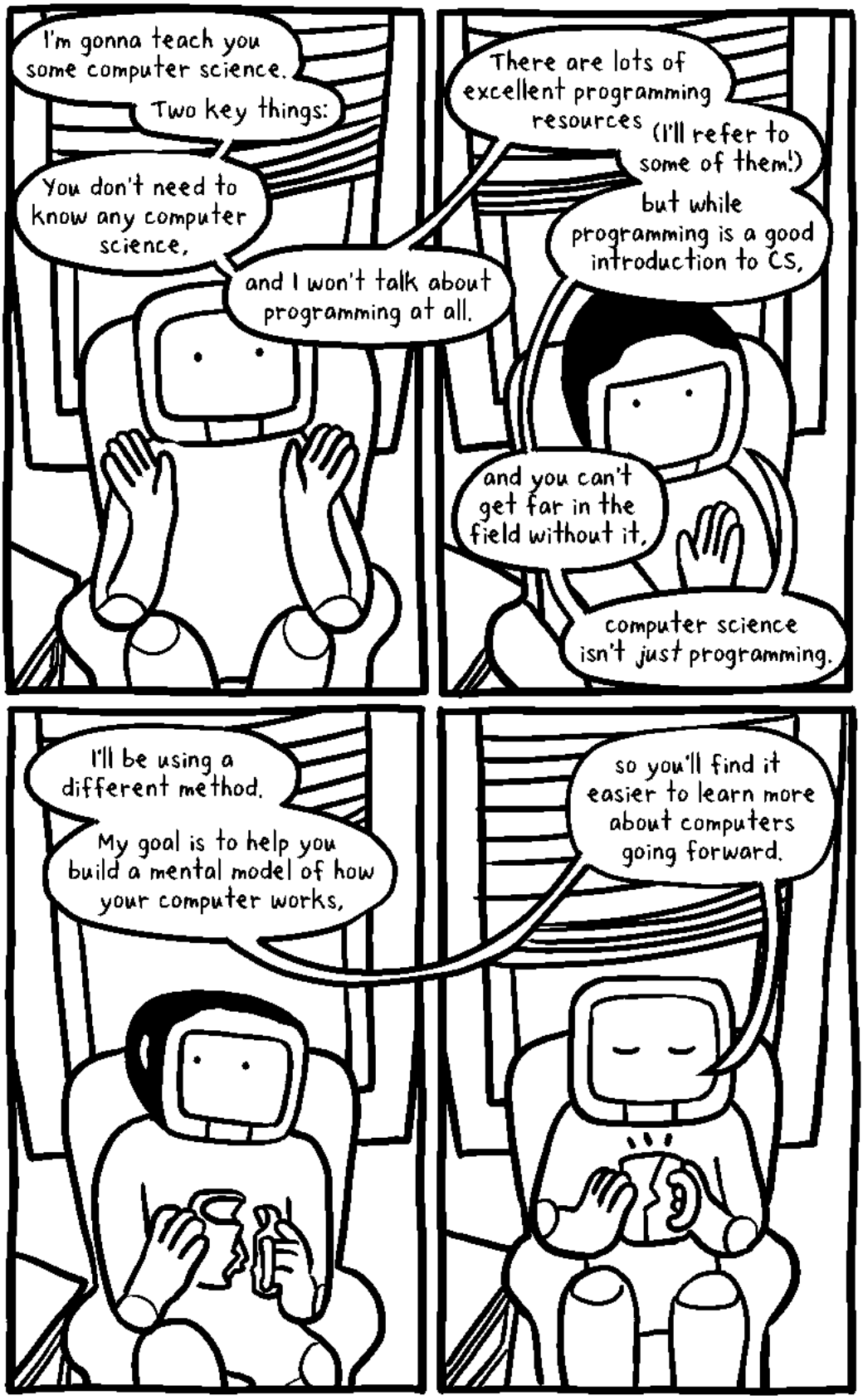

welcome to
computer science
audra mcnamee


Welcome to
Computer Science
Computer and
Information Science
Clark Honors
College
2022 Thesis
Audra McNamee


The elusive computer in its natural habitat.
How do we learn its secrets?
Behavioral observation?
Dissection?


SLAM
Let's lay out the situation.
SLAM
I'm a talking computer.
Why do I have a body?
To make gesturing easier, of course.
CRASH
This is my apartment,
my PDP-7,
my cat,
my computer history poster collection,
and a pile of my unsorted mail.
Some of this will be relevant.

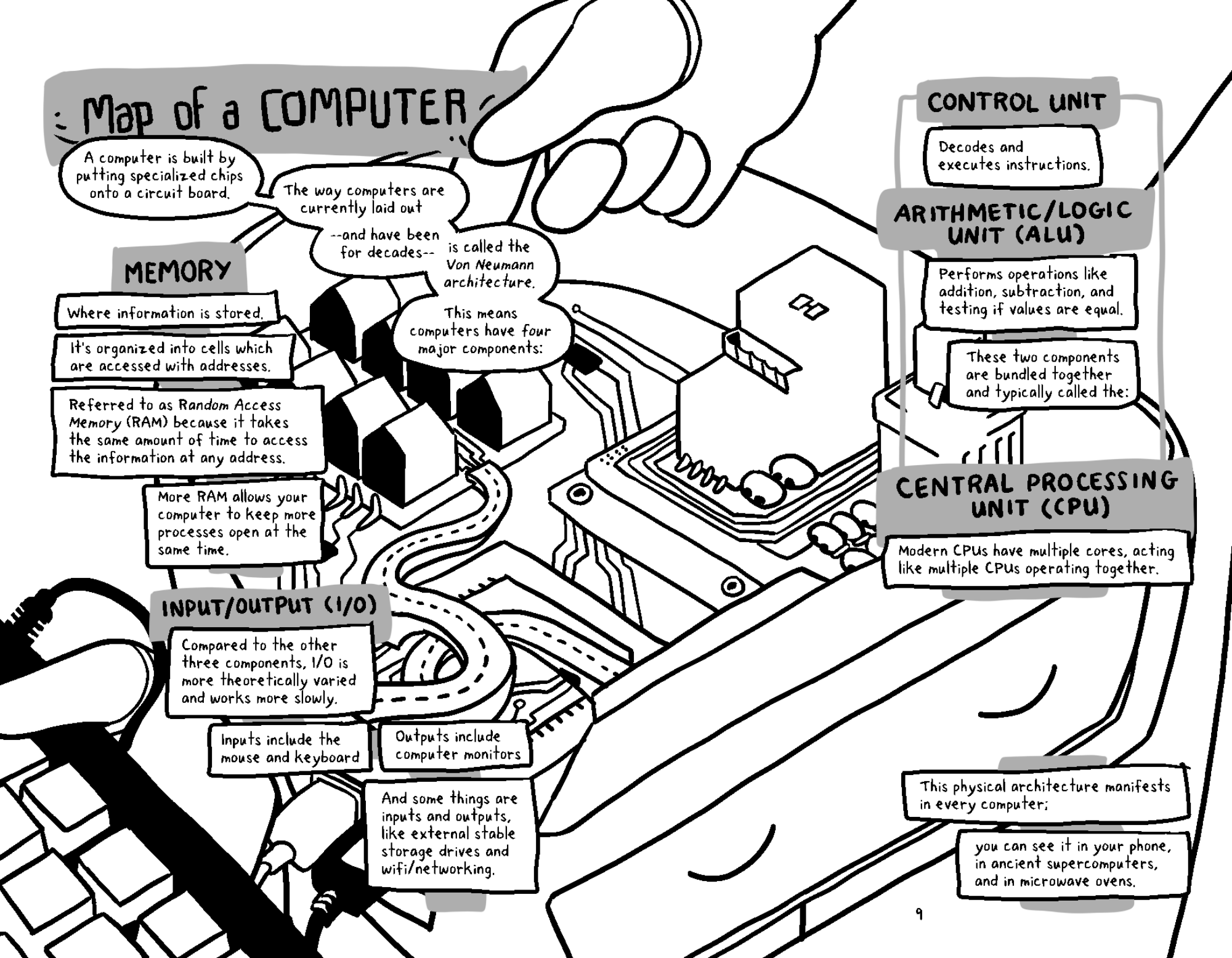

I'm gonna teach you some computer science.
Two key things:
You don't need to know any computer science,
and I won't talk about programming at all.
There are lots of excellent programming resources (I'll refer to some of them!)
but while programming is a good introduction to CS,
and you can't get far in the field without it,
computer science isn't *just* programming.
I'll be using a different method.
My goal is to help you build a mental model of how your computer works,
so you'll find it easier to learn more about computers going forward.

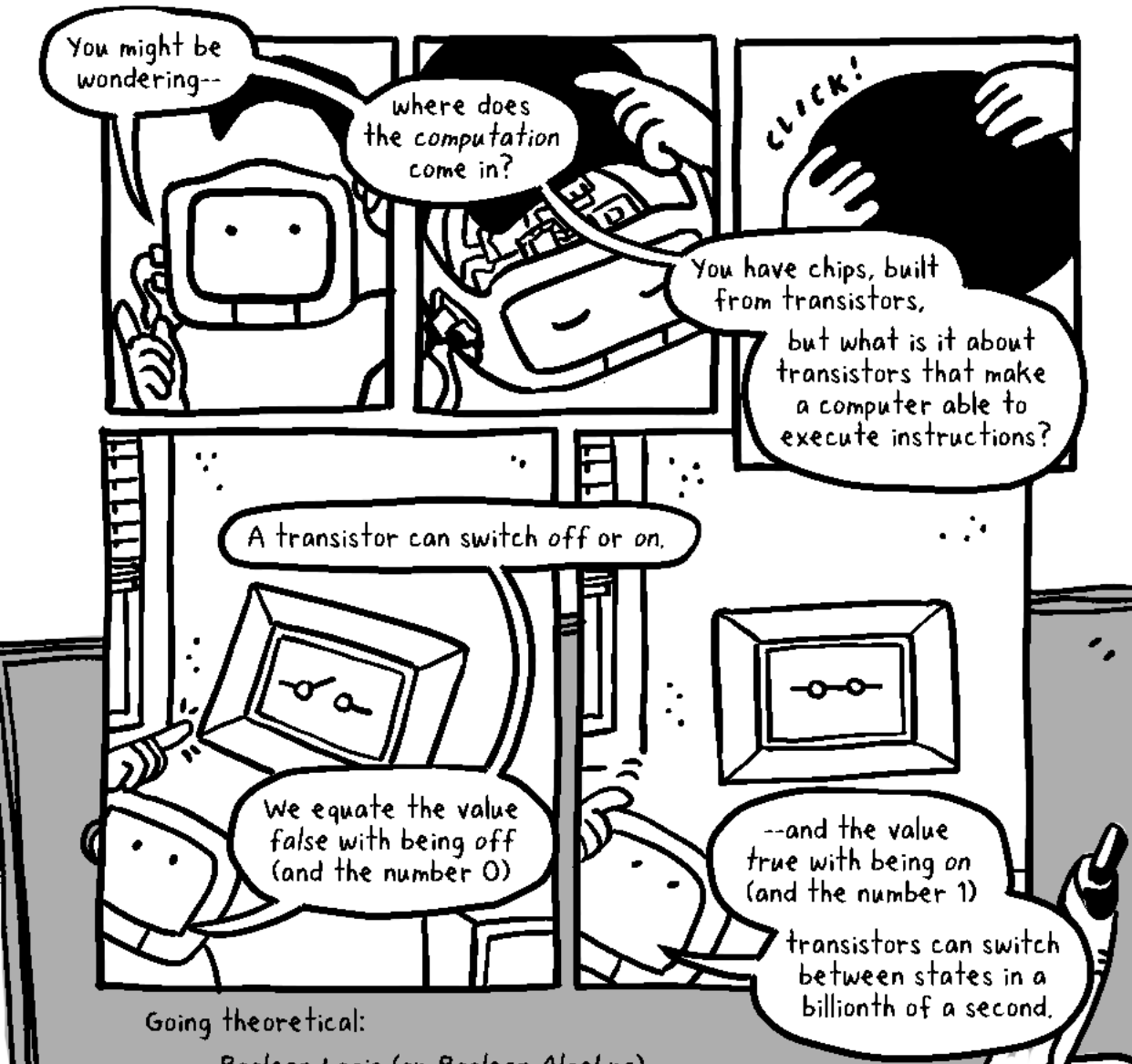

There's an old quip,
"computer science is as much about computers as astronomy is about telescopes."
But an astronomer who didn't know how their telescope worked would have a rough time.
PART ONE:
WHAT'S A COMPUTER?
NO, LITERALLY,
WHAT IS IT?

PART ONE:

COMPUTER ARCHITECTURE


Let's start at the bottom,
at the hard metal and squishy silicon
ha ha
silicon isn't squishy.
The modern physical building block of the computer is the *transistor*.
Think of it like the computer's atom.
This primary component *used* to be *vacuum tubes* the size of your hand.
Transistors come in many sizes;
now 10 *billion* transistors can fit in a 1cm² space.
Transistors and pathways between them are fabricated on silicon to create an *integrated circuit*, also called a *chip*.
As of 2018, an estimated 13 *sextillion* transistors have been manufactured
an order of magnitude more than the number of stars in the Milky Way.


Map of a COMPUTER
A computer is built by putting specialized chips onto a circuit board.
The way computers are currently laid out
--and have been for decades--
is called the Von Neumann architecture.
This means computers have four major components:
MEMORY
Where information is stored.
It's organized into cells which are accessed with addresses.
Referred to as Random Access Memory (RAM) because it takes the same amount of time to access the information at any address.
More RAM allows your computer to keep more processes open at the same time.
INPUT/OUTPUT (I/O)
Compared to the other three components, I/O is more theoretically varied and works more slowly.
Inputs include the mouse and keyboard
Outputs include computer monitors
And some things are inputs and outputs, like external stable storage drives and wifi/networking.
CONTROL UNIT
Decodes and executes instructions.
ARITHMETIC/LOGIC UNIT (ALU)
Performs operations like addition, subtraction, and testing if values are equal.
These two components are bundled together and typically called the:
CENTRAL PROCESSING UNIT (CPU)
Modern CPUs have multiple cores, acting like multiple CPUs operating together.
This physical architecture manifests in every computer;
you can see it in your phone, in ancient supercomputers, and in microwave ovens.

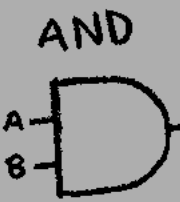


Going theoretical:

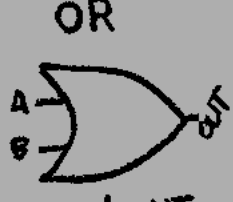
Boolean Logic (or *Boolean Algebra*) uses the values *true* and *false* to build complex conditional statements.

The fundamental Boolean operations are:

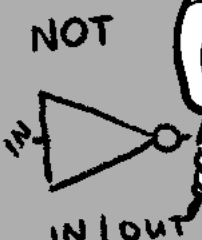
symbolized by these *logic gates*:



A	B	OUT
0	0	0
1	0	0
0	1	0
1	1	1



A	B	OUT
0	0	0
1	0	1
0	1	1
1	1	1



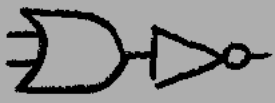
IN	OUT
0	1
1	0

Each of these logic gates can be built with transistors.

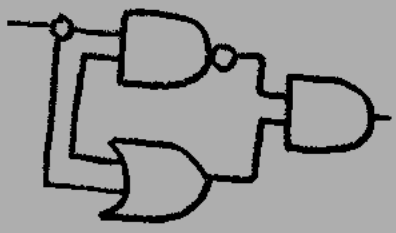
From these three operations you can derive others, including:

NAND

NOR

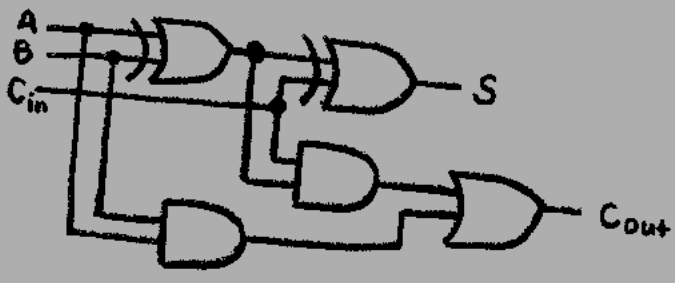


XOR



And more complicated circuits:

ADDER



Each of the chips we've discussed are built from these transistors, using these principles.

The final element of the Von Neumann architecture is

the computer runs off stored programs,

which are fetched one instruction at a time and passed to the control unit, where they're decoded and execut

but we've gotta slow down.

We can't describe *decoding* til we know how information is *encoded*.

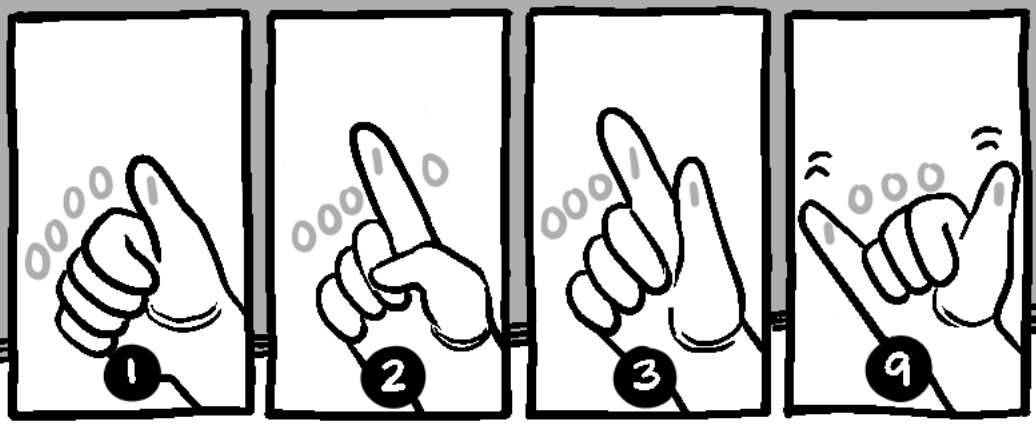
Information is encoded in binary, or base-2,

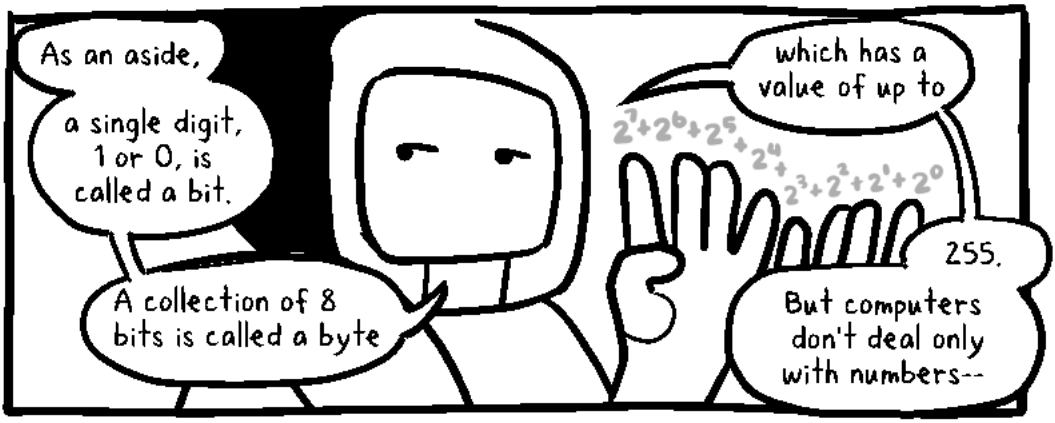
the number system with two digits:

0 off, false

1 on, true

Seeing any patterns?





The goal of encoding information into numbers

(which computers represent in binary)

is to take information humans understand and translate it into a form that computers can store, transport, and replicate.

Text

Usually encoded according to the standard *Unicode*.

```
55 73 75 61 6c 6c
79 20 65 6e 63 6f
64 65 64 20 61 63
63 6f 72 64 69 6e
```

It has standards for encoding characters from 159+ scripts

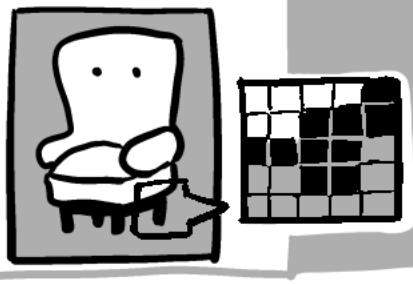
δ	03B4
ñ	00F1
œ	00E6
ü	00FC

As well as symbols

∞	221E
≠	2260
☹	1F643
🙌	1F440
♡	2764

Images

Broken into an array of pixels.



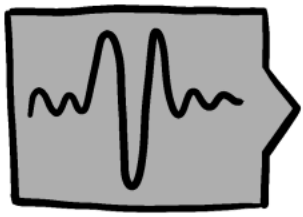
Each pixel stores its color, commonly with the RGB encoding scheme.

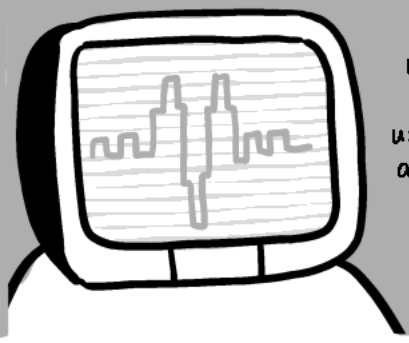
255 255 255 — 0 0 0 — 174 174 174

Red, green, and blue each have an associated byte expressing how much of that color is present in the pixel, between 0 (none), and 255 (max).

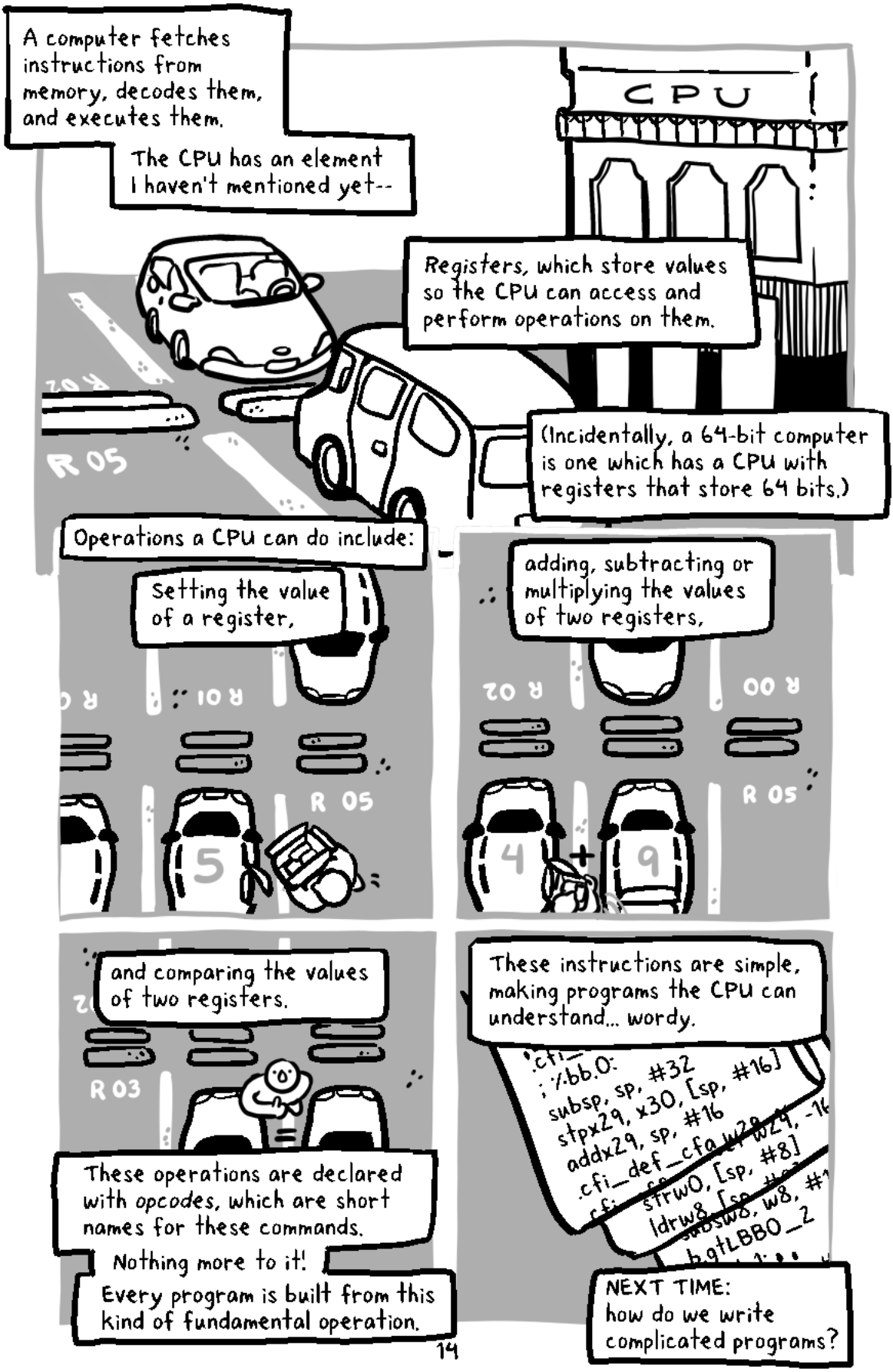
Sound

Encoded according to the wave's amplitude.





Points along the wave are sampled; these points are used to reconstruct an approximation of the wave.


A computer fetches instructions from memory, decodes them, and executes them.
The CPU has an element I haven't mentioned yet--
CPU
Registers, which store values so the CPU can access and perform operations on them.
R 05
(Incidentally, a 64-bit computer is one which has a CPU with registers that store 64 bits.)
Operations a CPU can do include:
Setting the value of a register,
R 01
R 05
5
adding, subtracting or multiplying the values of two registers,
R 02
R 00
R 05
4
+
9
and comparing the values of two registers.
R 03
These operations are declared with opcodes, which are short names for these commands.
Nothing more to it!
Every program is built from this kind of fundamental operation.
These instructions are simple, making programs the CPU can understand... wordy.
.cfi_
.LBB0:
subsp, sp, #32
stpx29, x30, [sp, #16]
addx29, sp, #16
.cfi_def_cfa
strw0, [sp, #8]
ldrw8, [sp,
b.gtLBB0_2
NEXT TIME:
how do we write complicated programs?

Further Reading

If you want to learn more about any topics mentioned, the computer science books cited at the end of the end of the comic, or your search engine of choice, can teach you a lot!

Page 7: Computer pictured is the ENIAC.
Total transistors estimation from "13 Sextillion & Counting: The Long & Winding Road to the Most Frequently Manufactured Human Artifact in History" by David Laws on the Computer History Museum blog (2018).

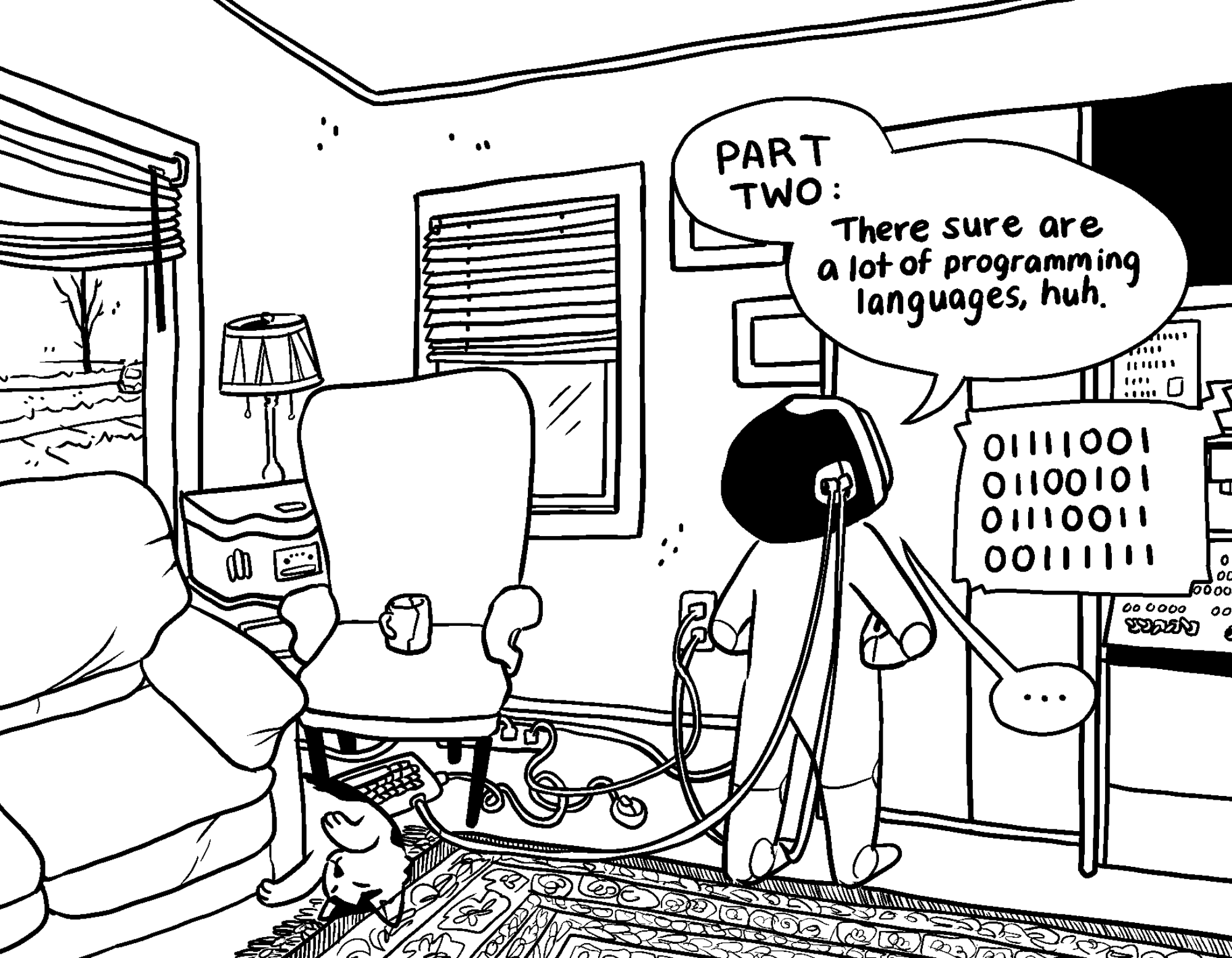
Page 8-9: Reading more about the Von Neumann architecture will explain in much more detail the working of each of the components mentioned. Computer engineering courses are more focused on hardware than computer science is. This Von Neumann architecture will remain the way computers will be built, at least if/until quantum computing is realized.

Page 10-11: Boolean logic is often taught in logic philosophy classes as well as computer science courses. Logic gates are related to circuit design and hardware-- one way to learn more about them is playing the video game SHENZHEN I/O.

Page 12: Binary, or base-2, is how information is encoded in computers, but information encoded in binary within the computer is often written in hexadecimal, or base-16, when humans have to read it. This is because information is more compressed in hexadecimal than it is in binary, but unlike decimal, or base-10, the 'normal' number system, hexadecimal and binary 'translate' directly between each other-- whereas a decimal digit can be between 1 and 4 binary digits, a hexadecimal digit is always exactly 4 binary digits.

Page 13: The Unicode Consortium controls the Unicode standard-- if you want to look at all the scripts they represent, learn about their decision-making process, or look at the way Unicode has developed over time, their website is an interesting resource.

Page 14: This page briefly mentions lower-level programming-- there will be more on this topic in the next section.


PART TWO:
There sure are a lot of programming languages, huh.
01111001
01100101
01110011
00111111
...

PART TWO:
PROGRAMMING LANGUAGES

The Fibonacci Numbers, created three ways

Python

```
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n - 1)
        + fib(n - 2)

n = int(input())
print(fib(n))
```

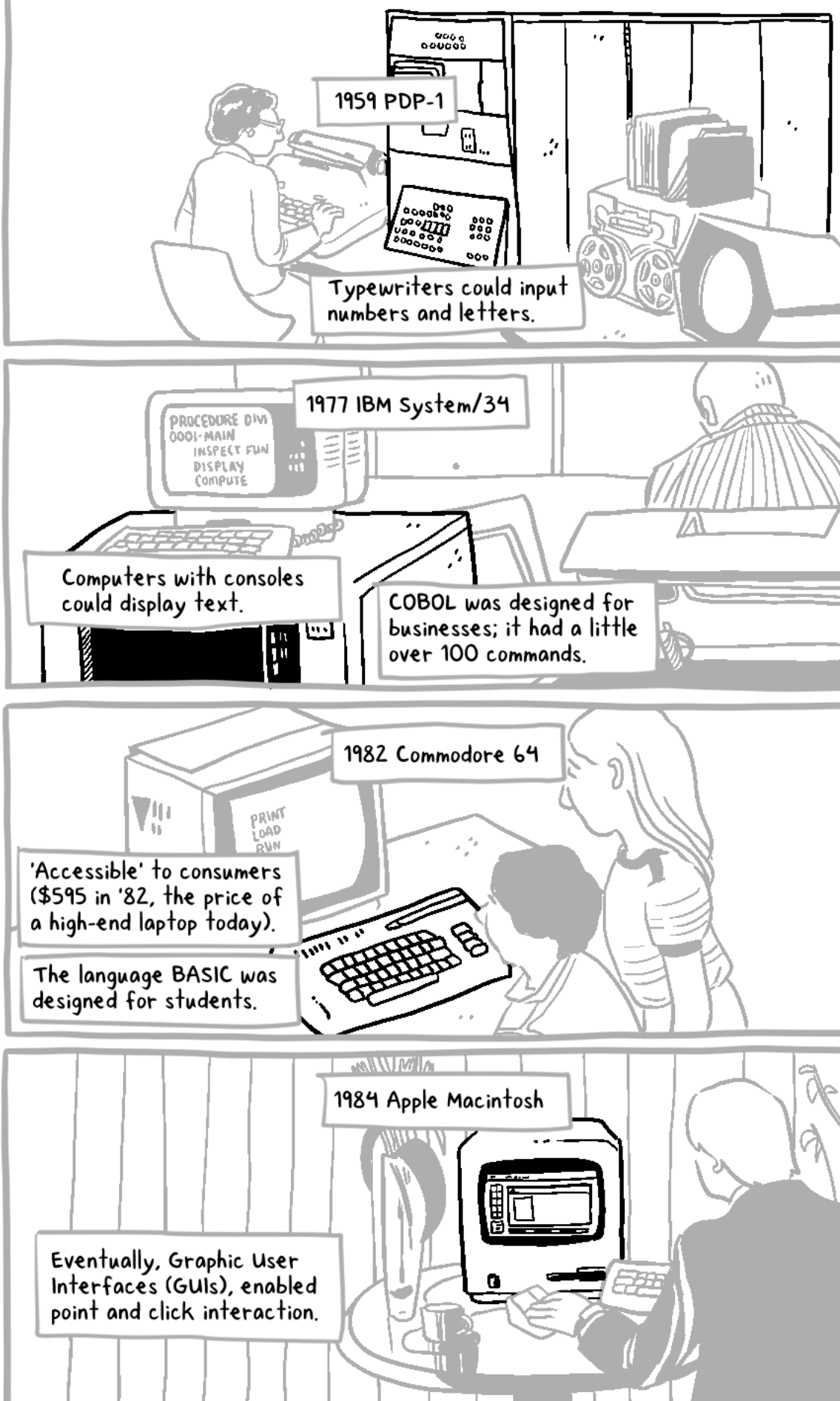
C

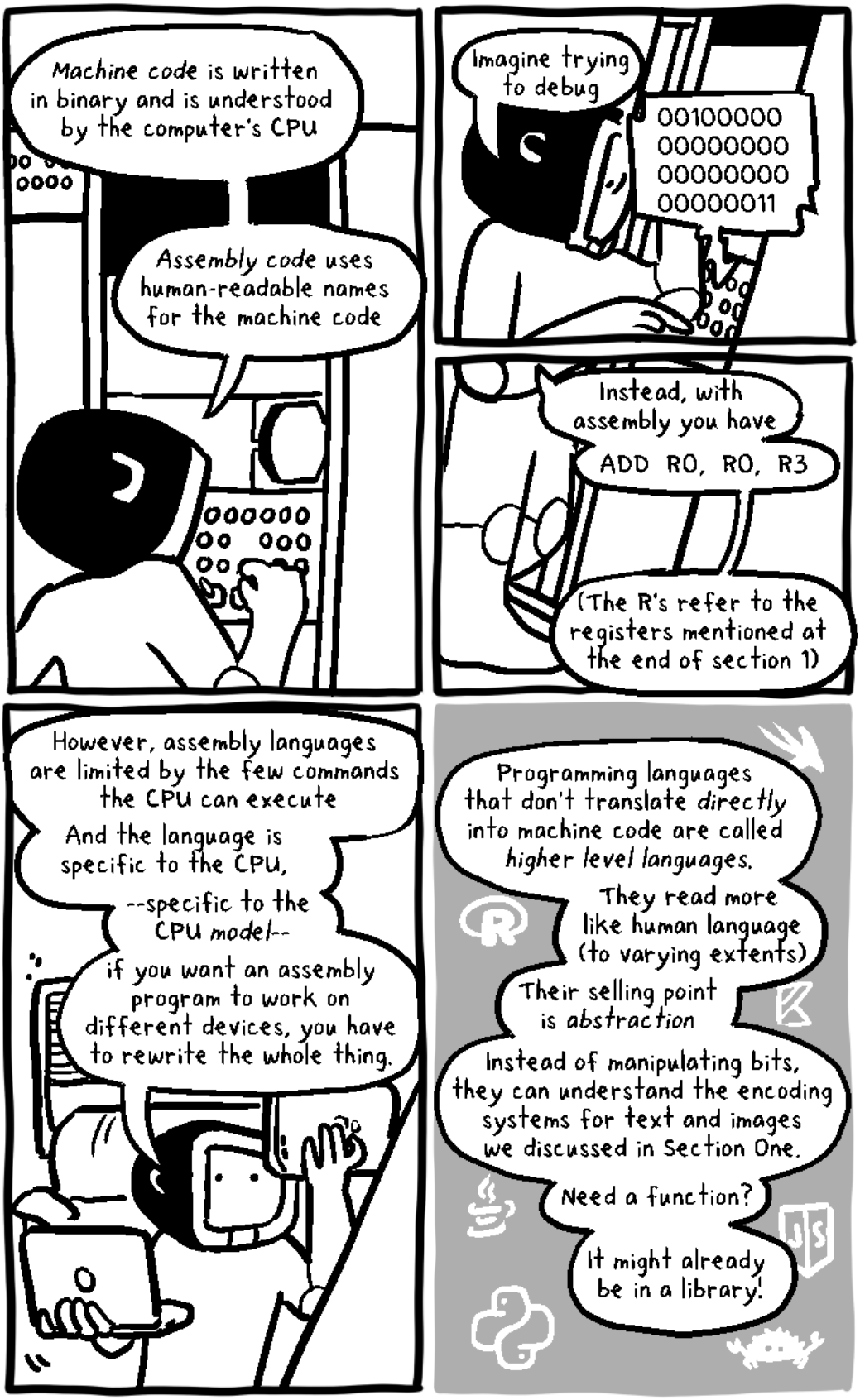
```
#include <stdio.h>
#include <stdlib.h>

int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1)
    + fib(n - 2);
}

int main(int argc,
         char *argv[])
{
    char *a = argv[1];
    int n = atoi(a);
    printf("%d", fib(n));
    getchar();
    return 0;
}
```

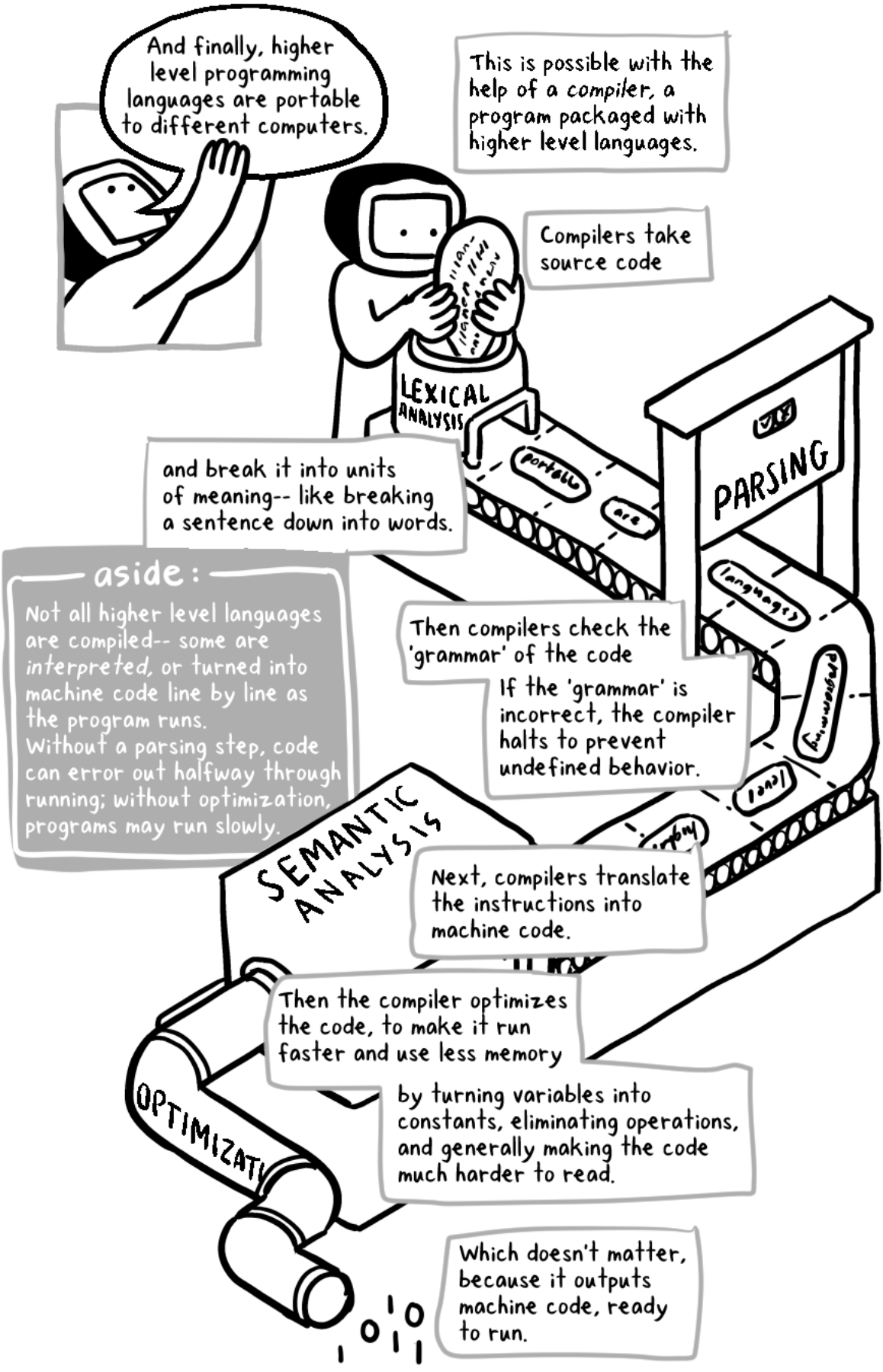
Assembly

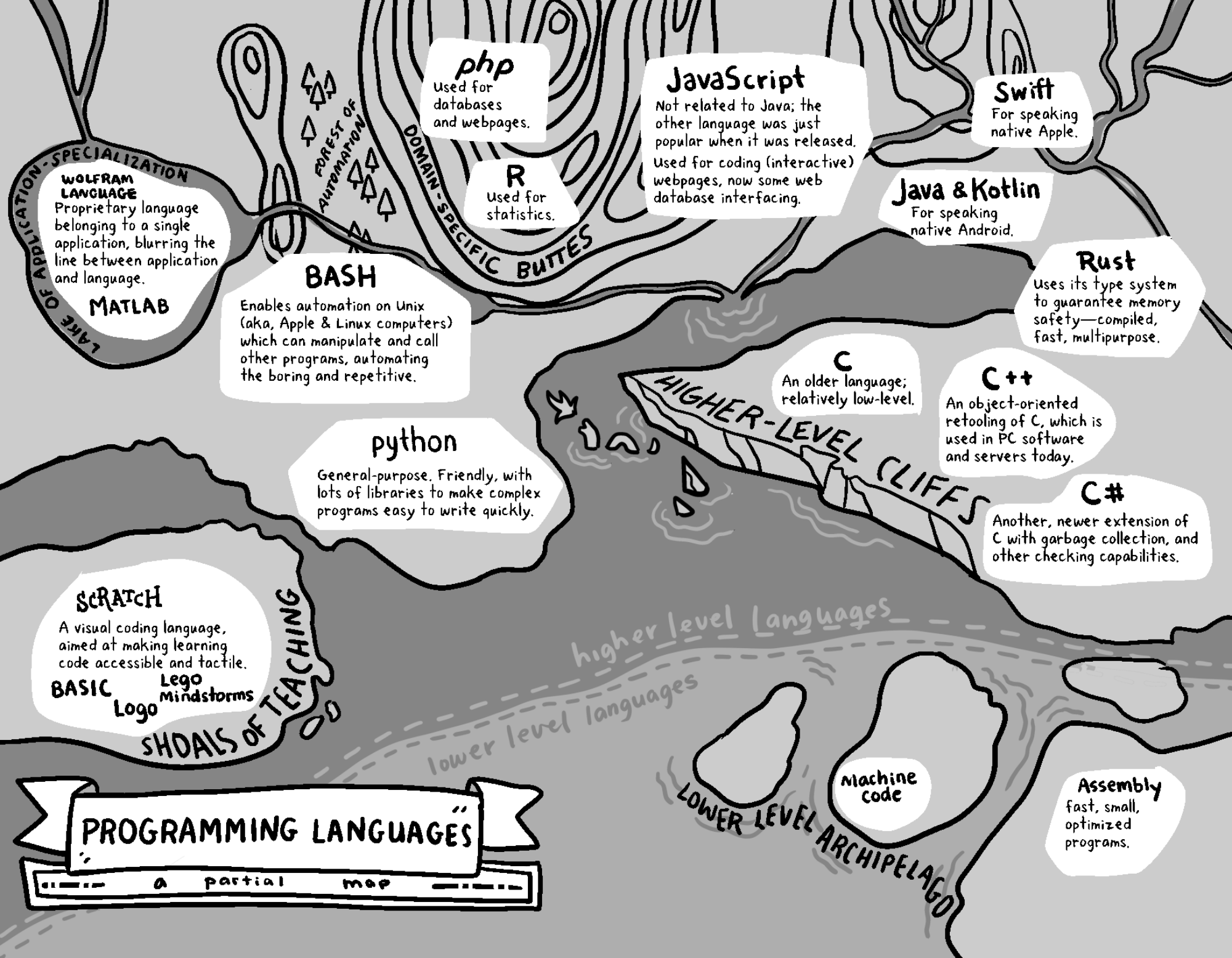
```
.s.globl_fib
.p2align2
_fib:
.cfi_startproc
; %bb.0:
subsp, sp, #32
stpx29, x30, [sp, #16]
addx29, sp, #16
.cfi_def_cfa w29, 16
.cfi_offset w30, -8
.cfi_offset w29, -16
strw0, [sp, #8]
ldrw8, [sp, #8]
subsw8, w8, #1
b.gtLBB0_2
; %bb.1:
ldrw8, [sp, #8]
sturw8, [x29, #-4]
bLBB0_3
LBB0_2:
ldrw8, [sp, #8]
subsw0, w8, #1
bl_fib
strw0, [sp, #4]
ldrw8, [sp, #8]
subsw0, w8, #2
bl_fib
movx8, x0
ldrw0, [sp, #4]
addw8, w0, w8
sturw8, [x29, #-4]
LBB0_3:
ldurw0, [x29, #-4]
ldpx29, x30, [sp, #16]
aaasp, sp,
ret
.cfi_endproc
```


Computers have been programmed in many ways over the years:
The theoretical 1820s Difference Engine
A mechanical calculator.
1945 ENIAC
Programs 'mapped' onto machine by plugboard wires.
It's believed ENIAC did more calculations over the ten years it was in operation than all of humanity had done previously.
1952 Harwell Computer
Stored information on punched tape in binary.
1's are holes, 0's are paper.
1954 IBM 704
Toggled in programs one bit at a time.


1959 PDP-1
Typewriters could input numbers and letters.
1977 IBM System/34
PROCEDURE DIVI
0001-MAIN
INSPECT FUN
DISPLAY
COMPUTE
Computers with consoles could display text.
COBOL was designed for businesses; it had a little over 100 commands.
1982 Commodore 64
PRINT
LOAD
RUN
'Accessible' to consumers ($595 in '82, the price of a high-end laptop today).
The language BASIC was designed for students.
1984 Apple Macintosh
Eventually, Graphic User Interfaces (GUIs), enabled point and click interaction.


Machine code is written in binary and is understood by the computer's CPU
Assembly code uses human-readable names for the machine code
Imagine trying to debug
00100000
00000000
00000000
00000011
Instead, with assembly you have
ADD R0, R0, R3
(The R's refer to the registers mentioned at the end of section 1)
However, assembly languages are limited by the few commands the CPU can execute
And the language is specific to the CPU,
--specific to the CPU model--
if you want an assembly program to work on different devices, you have to rewrite the whole thing.
Programming languages that don't translate directly into machine code are called higher level languages.
They read more like human language (to varying extents)
Their selling point is abstraction
Instead of manipulating bits, they can understand the encoding systems for text and images we discussed in Section One.
Need a function?
It might already be in a library!


And finally, higher level programming languages are portable to different computers.
This is possible with the help of a compiler, a program packaged with higher level languages.
Compilers take source code
LEXICAL ANALYSIS
and break it into units of meaning-- like breaking a sentence down into words.
aside:
Not all higher level languages are compiled-- some are interpreted, or turned into machine code line by line as the program runs.
Without a parsing step, code can error out halfway through running; without optimization, programs may run slowly.
PARSING
Then compilers check the 'grammar' of the code
If the 'grammar' is incorrect, the compiler halts to prevent undefined behavior.
SEMANTIC ANALYSIS
Next, compilers translate the instructions into machine code.
Then the compiler optimizes the code, to make it run faster and use less memory
by turning variables into constants, eliminating operations, and generally making the code much harder to read.
OPTIMIZATI
Which doesn't matter, because it outputs machine code, ready to run.


PROGRAMMING LANGUAGES
a partial map
LAKE OF APPLICATION-SPECIALIZATION
WOLFRAM LANGUAGE
Proprietary language belonging to a single application, blurring the line between application and language.
MATLAB
FOREST OF AUTOMATION
DOMAIN-SPECIFIC BUTTES
php
Used for databases and webpages.
R
Used for statistics.
JavaScript
Not related to Java; the other language was just popular when it was released.
Used for coding (interactive) webpages, now some web database interfacing.
Swift
For speaking native Apple.
Java & Kotlin
For speaking native Android.
BASH
Enables automation on Unix (aka, Apple & Linux computers) which can manipulate and call other programs, automating the boring and repetitive.
Rust
Uses its type system to guarantee memory safety—compiled, fast, multipurpose.
HIGHER-LEVEL CLIFFS
C
An older language; relatively low-level.
C++
An object-oriented retooling of C, which is used in PC software and servers today.
python
General-purpose. Friendly, with lots of libraries to make complex programs easy to write quickly.
C#
Another, newer extension of C with garbage collection, and other checking capabilities.
SHOALS OF TEACHING
SCRATCH
A visual coding language, aimed at making learning code accessible and tactile.
BASIC
Logo
Lego Mindstorms
higher level languages
lower level languages
LOWER LEVEL ARCHIPELAGO
Machine code
Assembly
fast, small, optimized programs.

How do we define a programming language?

We can at least indicate what's *not* a language.

One thing that isn't:

HTML

A system is *Turing complete* named for Alan Turing if (in theory) it can describe *any algorithm.*

1912-1954

$x_0 \approx \sqrt{S}$

$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$

Say, from calculating a square root to implementing Google's search algorithm.

Whether something is *Turing complete* is a common criteria for a programming language.

Surprising things are Turing complete-- including Microsoft Excel (and many spreadsheet programs) and Minecraft's Redstone system

(it's easier to implement Facebook in PHP and Javascript than Excel, though-- using the right tool for the job is important!)

```
<div id="ex">
<p> HTML is not Turing complete so it's not typically considered a programming language. </p>
<p> Instead, it's a markup language— it formats text. </p>
</div>
```

This doesn't mean HTML isn't valuable-- just that it's a tool with a different purpose.

Further Reading

Page 19: The Fibonacci numbers are the sequence 1, 1, 2, 3, 5, 8, 13, 21,... where each number in the sequence is formed by adding the two previous numbers.
The formal study of "programming languages" is focused on topics like how to prove the 'correctness' of a program, and the surprisingly tricky question of whether the program will end, or loop forever. If you like proofs and math, it's exciting stuff!

Page 20-21: The computers mentioned on these pages are only some of the notable computers from history. A great resource for learning more about these computers is the Computer History Museum's website (the source for the ENIAC computation fact). The history of the ENIAC in particular is partially recounted in *Broad Band: The Untold Story of the Women Who Made the Internet*.

Page 22: Just because programming in assembly is hard, doesn't mean it's not useful--if you need a small optimized program for specific hardware, there's no better tool to use than an assembly language. Also, just because it's fiddly, doesn't mean it's not fun-- the videogames TIS-100 and Exapunks are both focused on solving problems with assembly-like languages.

Page 23: Compilers are a whole section of computer science unto themselves-- there's an enormous amount of further information to learn about them.

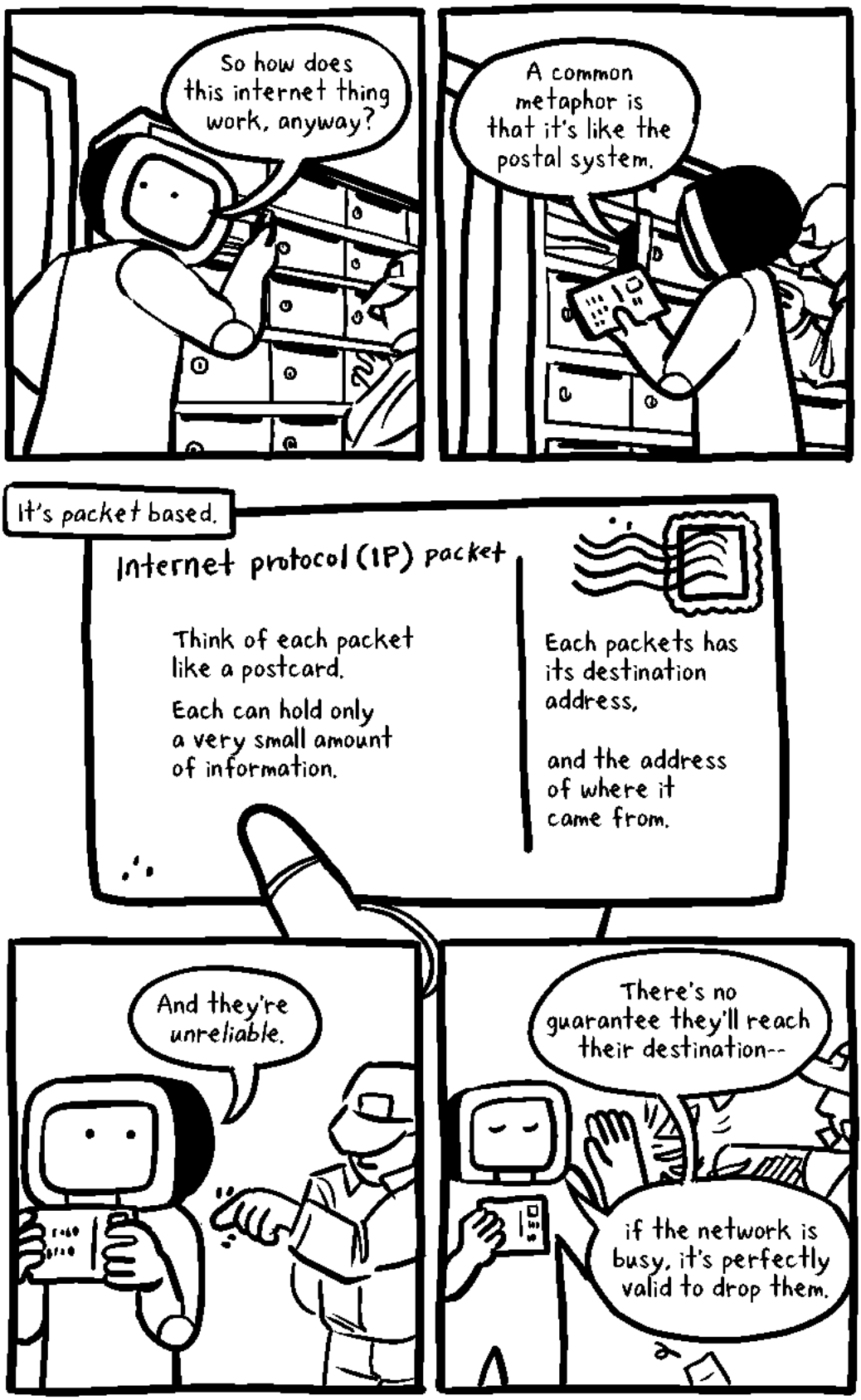
Page 24-25: Of course, not all programming languages are invented for a purpose. There's a whole subcategory of esoteric programming languages, or esolangs, whose wiki describes them as "computer programming language[s] designed to experiment with weird ideas, to be hard to program in, or as a joke, rather than for practical use." They range from baffling, with having deliberately hard-to-follow programming structure and syntax, to silly, like having syntax designed to make the program look like a cooking recipe (Chef) or a power ballad (Rockstar).

Page 26: Turing completeness is insufficiently described here, because describing what makes a language Turing-complete requires programming constructs we haven't covered; to see more surprisingly Turing-complete languages check out the video "On the Turing Completeness of PowerPoint"
Obligatory graphic novel mention: *The Imitation Game: Alan Turing Decoded* by Jim Ottaviani and Leland Purvis is a biography of Alan Turing.


PART THREE:
EVERYTHING IS CONNECTED

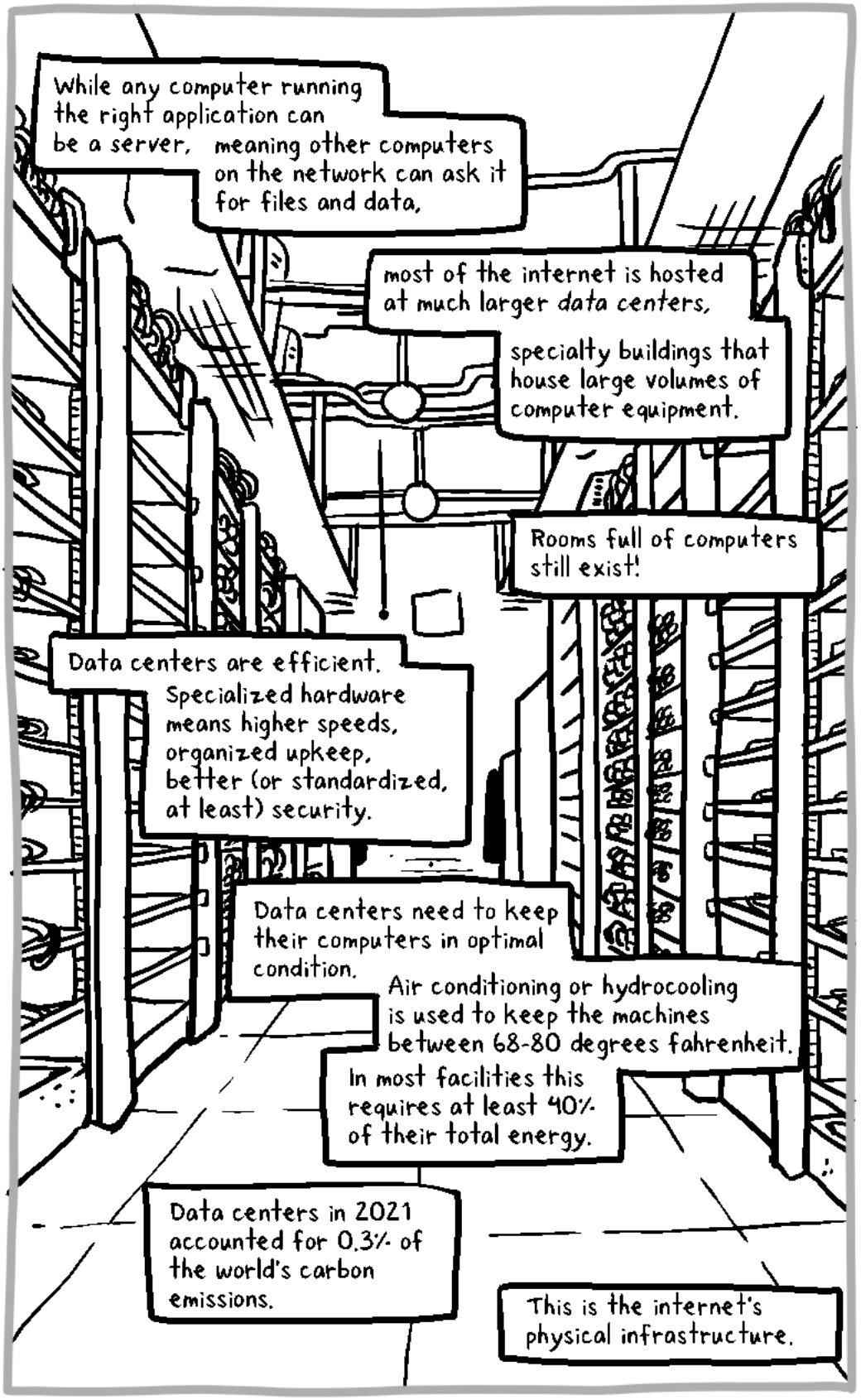
PART THREE: THE INTERNET

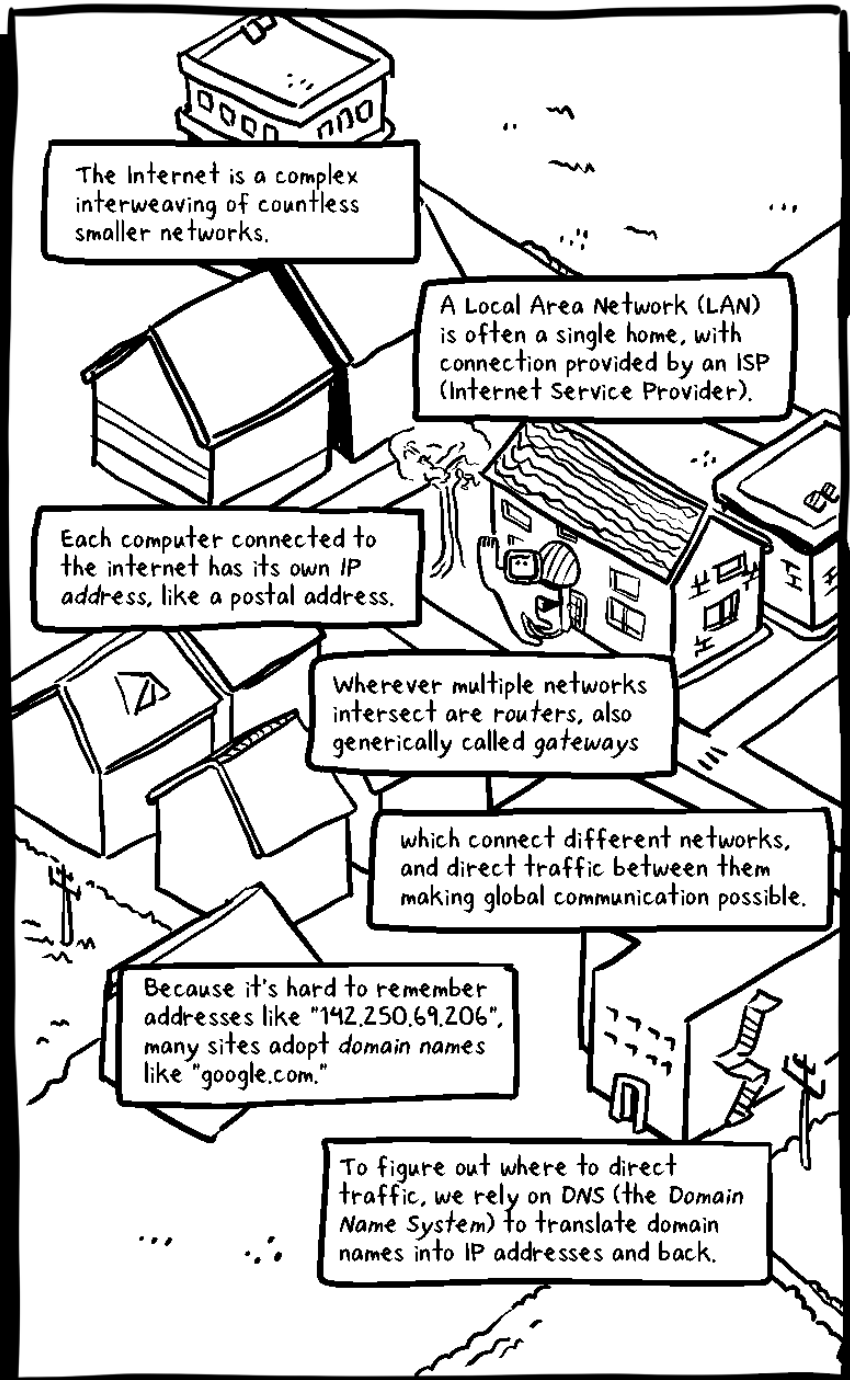

While we may think of computers as our
laptops,
desktops,
tablets,
smartphones,
they're also in modern cars, game consoles, smart thermostats and cameras...
These devices don't stand alone.
The value that we get out of them is by sending and receiving information.
CLACK
The shift in our society isn't that we've all got computers; it's that we're all networked.

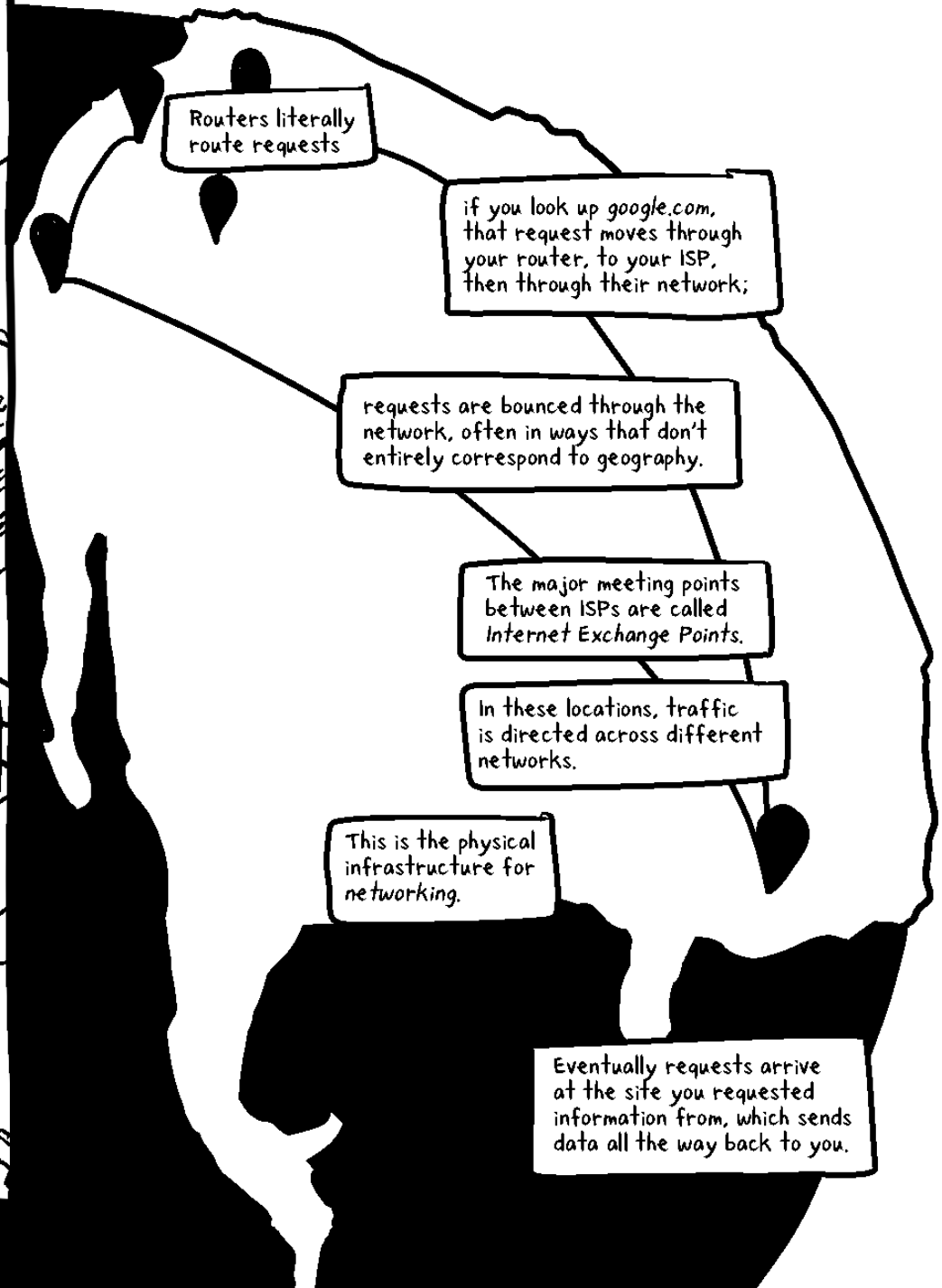

So how does this internet thing work, anyway?
A common metaphor is that it's like the postal system.
It's packet based.
Internet protocol (IP) packet
Think of each packet like a postcard.
Each can hold only a very small amount of information.
Each packets has its destination address,
and the address of where it came from.
And they're unreliable.
There's no guarantee they'll reach their destination--
if the network is busy, it's perfectly valid to drop them.

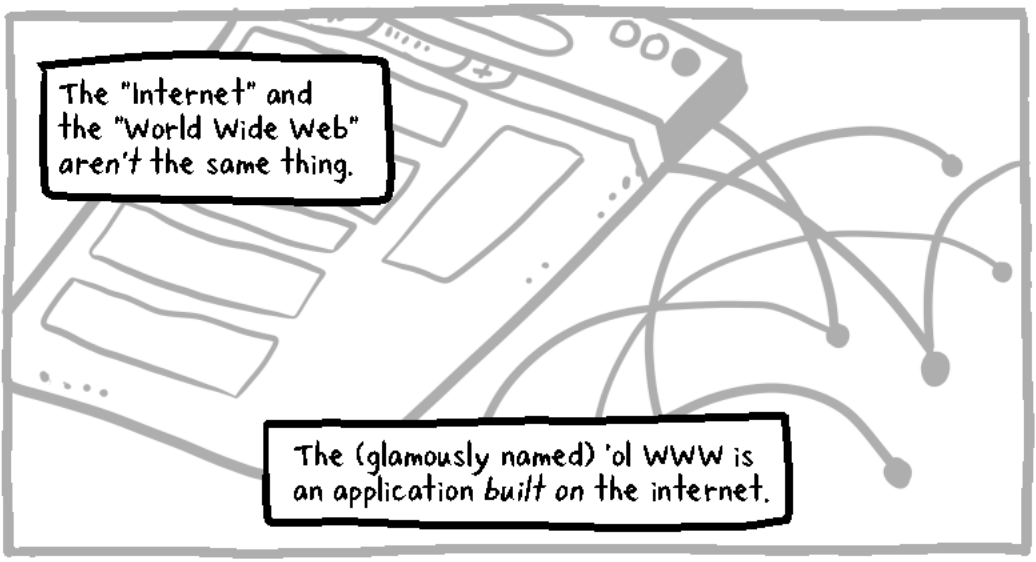

So, when, say, your penpal wants to send you a novel:
She breaks it up into postcards.
She numbers each postcard according to where they are in the sequence.
Then one by one, she sends them all.
Like a good pal, when you get them you write her back
and if you're missing any in the sequence, you let her know that, too
and she resends them.
By repeating this enough times and resending information if there's no response
the unreliable system of sending packets (IP) turns into a functionally reliable stream of data.
This system is called TCP (transmission control protocol).

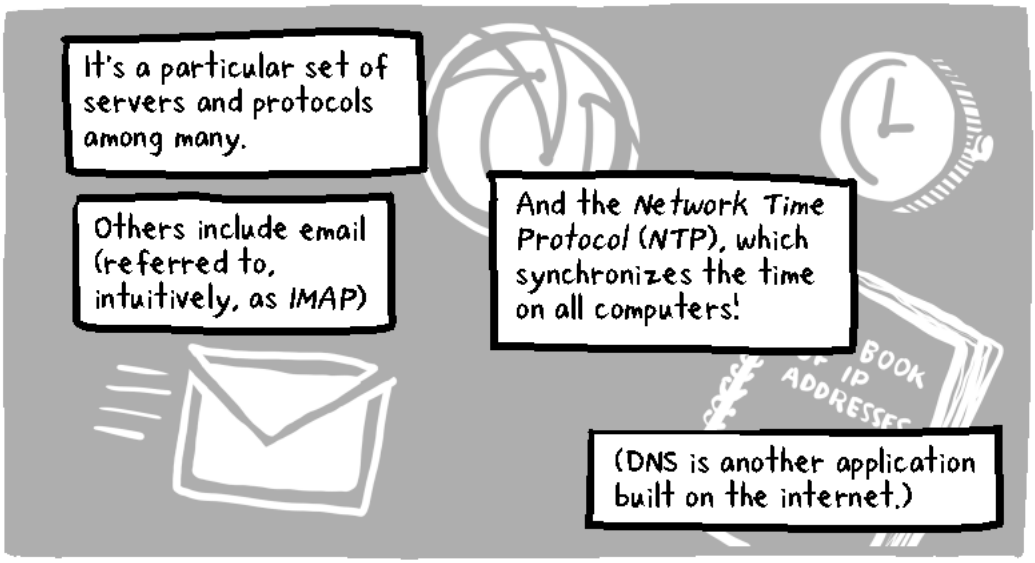

There's no "cloud."
Data is stored on faraway computers, and it travels on wires (at speeds on the order of magnitude of the speed of light).
These wires run through neighborhoods and between cities.
Some of them spanning oceans.
They're significant installations--
this is why rural areas often don't have sufficient connection.
While satellites are accessible from almost anywhere, the distance the data has to travel (up to the satellite and back down)
gives satellites a delayed response time that's less than ideal for modern internet browsing.

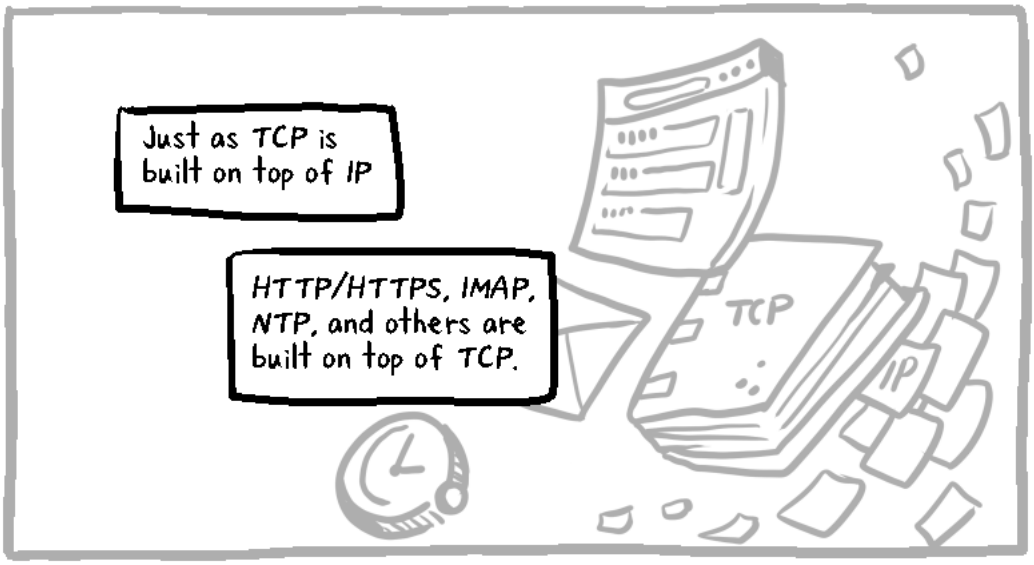

While any computer running the right application can be a server, meaning other computers on the network can ask it for files and data,
most of the internet is hosted at much larger *data centers*,
specialty buildings that house large volumes of computer equipment.
Rooms full of computers still exist!
Data centers are efficient.
Specialized hardware means higher speeds, organized upkeep, better (or standardized, at least) security.
Data centers need to keep their computers in optimal condition.
Air conditioning or hydrocooling is used to keep the machines between 68-80 degrees fahrenheit.
In most facilities this requires at least 40% of their total energy.
Data centers in 2021 accounted for 0.3% of the world's carbon emissions.
This is the internet's physical infrastructure.


The Internet is a complex interweaving of countless smaller networks.
A Local Area Network (LAN) is often a single home, with connection provided by an ISP (Internet Service Provider).
Each computer connected to the internet has its own IP address, like a postal address.
Wherever multiple networks intersect are routers, also generically called gateways
which connect different networks, and direct traffic between them making global communication possible.
Because it's hard to remember addresses like "142.250.69.206", many sites adopt domain names like "google.com."
To figure out where to direct traffic, we rely on DNS (the Domain Name System) to translate domain names into IP addresses and back.


Routers literally route requests
if you look up google.com, that request moves through your router, to your ISP, then through their network;
requests are bounced through the network, often in ways that don't entirely correspond to geography.
The major meeting points between ISPs are called Internet Exchange Points.
In these locations, traffic is directed across different networks.
This is the physical infrastructure for networking.
Eventually requests arrive at the site you requested information from, which sends data all the way back to you.


The "Internet" and the "World Wide Web" aren't the same thing.
The (glamously named) 'ol WWW is an application *built on* the internet.


It's a particular set of servers and protocols among many.
Others include email (referred to, intuitively, as *IMAP*)
And the *Network Time Protocol* (*NTP*), which synchronizes the time on all computers!
BOOK OF IP ADDRESSES
(DNS is another application built on the internet.)


Just as *TCP* is built on top of *IP*
HTTP/HTTPS, *IMAP*, *NTP*, and others are built on top of *TCP*.
TCP
IP

Page 32: This page is talking about Internet Protocol (IP).

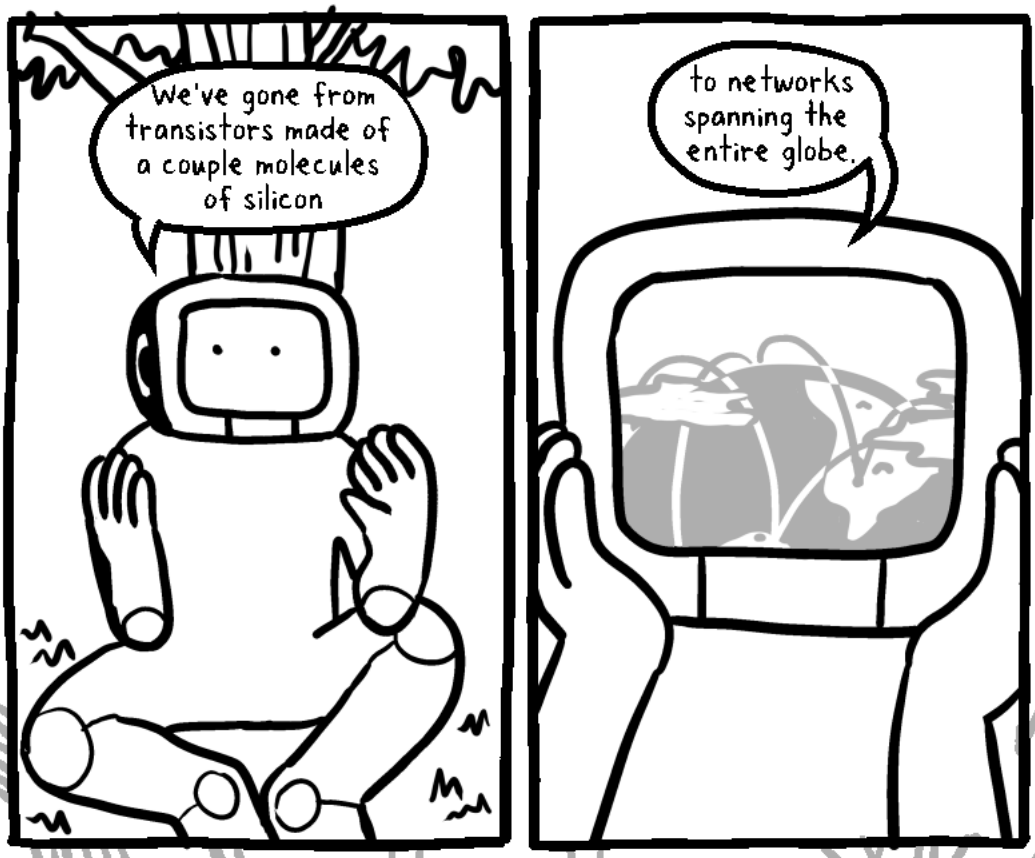
Page 33: This page is talking about Transmission control protocol (TCP).

Page 35: Information on this page is sourced from the article "The Staggering Ecological Impacts of Computation and the Cloud" by Steven Gonzalez Monserrate (2022).

Page 36-37: I based the map of the second part of this page on the way my own visit to google.com traveled across the network. If you want to see the way your requests travel, you can use the Traceroute mapper website I've linked on the following page, which walks you through using the BASH command 'traceroute,' and then visualizes the output.

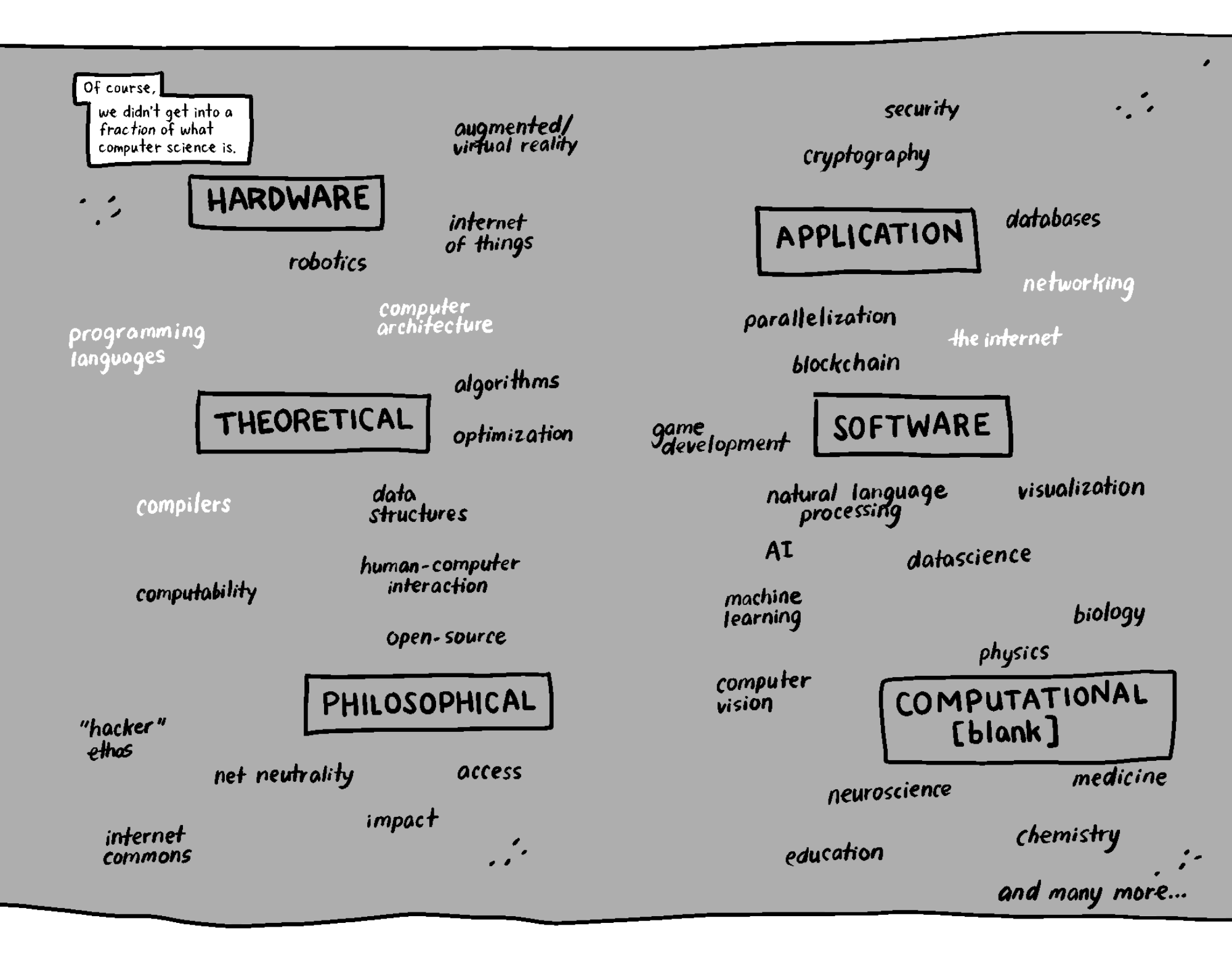

AND THAT'S ENOUGH
FOR NOW

CONCLUSION



The fact that computers exist, that they work at all, is a testament to human ingenuity

and the product of collaboration among countless people.

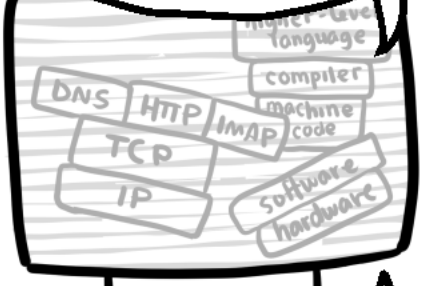

Of course, we didn't get into a fraction of what computer science is.
HARDWARE
augmented/ virtual reality
internet of things
robotics
computer architecture
programming languages
THEORETICAL
algorithms
optimization
compilers
data structures
computability
human-computer interaction
open-source
PHILOSOPHICAL
"hacker" ethos
net neutrality
access
impact
internet commons
security
cryptography
APPLICATION
databases
networking
parallelization
the internet
blockchain
SOFTWARE
game development
natural language processing
visualization
AI
datascience
machine learning
biology
physics
computer vision
COMPUTATIONAL [blank]
neuroscience
medicine
chemistry
education
and many more...

Stick around for next time

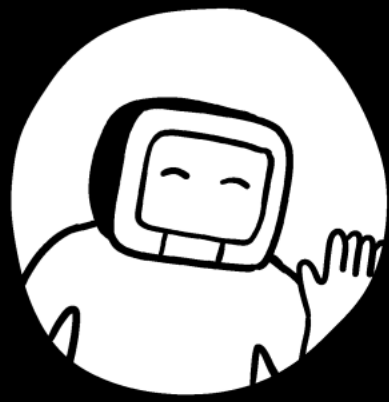
Maybe to learn about algorithms and efficiency, or computer security, artificial intelligence, what being 'open source' means

Who's to say, we could even get *programming* involved

I hope you got a sense of how fundamentally *simple* the information that computers store and organize is



And how you can understand the computer as built in *layers* out of these simple pieces.



Further Reading

While working on this project the three textbooks that I referenced most frequently were:

Justice, M. (2020). *How computers really work: A hands-on guide to the inner workings of the machine*. No Starch Press, Inc.

> Covers computers, from binary, through circuitry and hardware, to machine code, operating systems, and the internet. This book spends time on the hardware side of the equation, and has hands-on breadboard and circuits projects.

Kernighan, B. (2021). *Understanding the digital world* (2nd edition). Princeton University Press.

> This book is written for a not strictly technical audience. Its fourth section is titled "Data," and is about how computers are used in society from a philosophical and ethical standpoint.

Schneider, G. M., & Gersting, J. L. (2019). *Invitation to computer science* (8th edition). Cengage Learning.

> This is a traditional textbook and a complete introductory computer science course focused on delivering the breadth of the field.

If you're interested in any topic in mentioned here, I highly recommend you pick one of these books, or your favorite search engine, and look it up to learn some more!

You may have noticed that I recommended three Zachtronics video games (*SHENZHEN I/O*, *TIS-100*, and *Exapunks*). The game *Human Resource Machine* is also a fun way to get your hands on some programming in a game environment.

It's years out of date, but I have to acknowledge this project's similarity to *The Cartoon Guide to Computer Science* by Larry Gronick (1983). Other books combining computer science and cartooning include *Illustrated Basic*, which teaches the reader the programming language Basic (not Visual Basic, a later language). The *Secret Coders* series provides an introduction to programming for a middle-reader audience.

My personal favorite computer-adjacent graphic novel is *Logicomix*, about the life and work of logician and philosopher Bertrand Russell. It's mostly about math and logic, much of which relates closely to computer science, particularly the field of functional programming. The art is beautiful, and the masterful way it combines the narrative of Russell's life and an impressive amount of math is unparalleled.

Links

Unicode consortium site: unicode.org

"13 Sextillion & Counting: The Long & Winding Road to the Most Frequently Manufactured Human Artifact in History" by David Laws on the Computer History Museum blog (2018): https://computerhistory.org/blog/13-sextillion-counting-the-long-winding-road-to-the-most-frequently-manufactured-human-artifact-in-history/

"On the Turing Completeness of PowerPoint": https://www.youtube.com/watch?v=uNjxe8ShM-8

Surprisingly Turing-complete systems: https://www.gwern.net/Turing-complete

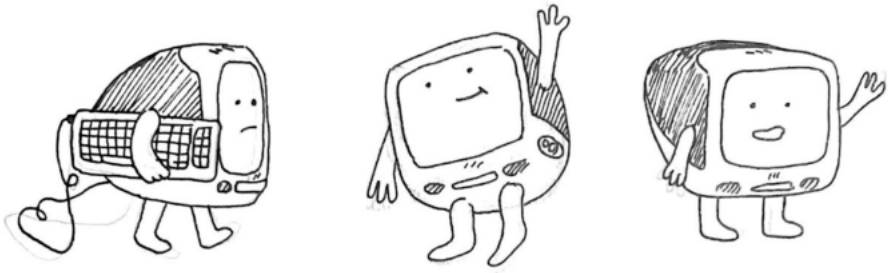
Esoteric programming languages wiki: https://esolangs.org/wiki/Esoteric_programming_language

Computer History Museum website: computerhistory.org

"The Staggering Ecological Impacts of Computation and the Cloud" by Steven Gonzalez Monserrate (2022): https://thereader.mitpress.mit.edu/the-staggering-ecological-impacts-of-computation-and-the-cloud/

Traceroute mapper: https://stefansundin.github.io/traceroute-mapper/

Developmental art



ABOVE: Sketches of the narrator before making the decision they had a full body.

BELOW: Sketches of pages 20 and 21.

About the author

Audra McNamee is a senior at UO, class of '22, majoring in Math and Computer science and minoring in Comics and Cartoon Studies. In 2020 Audra participated in the UO Science and Comics Initiative, creating the 8 page comic "A Trip into Serotonin" with computational neuroscientist Dr. Luca Mazzucato.

Outside of school, Audra makes short comics for fun, on topics ranging from the 60-year history of a liberatarian billboard off I-5 to an abridged history of Jell-O.

Audra's personal website is audmcname.com.