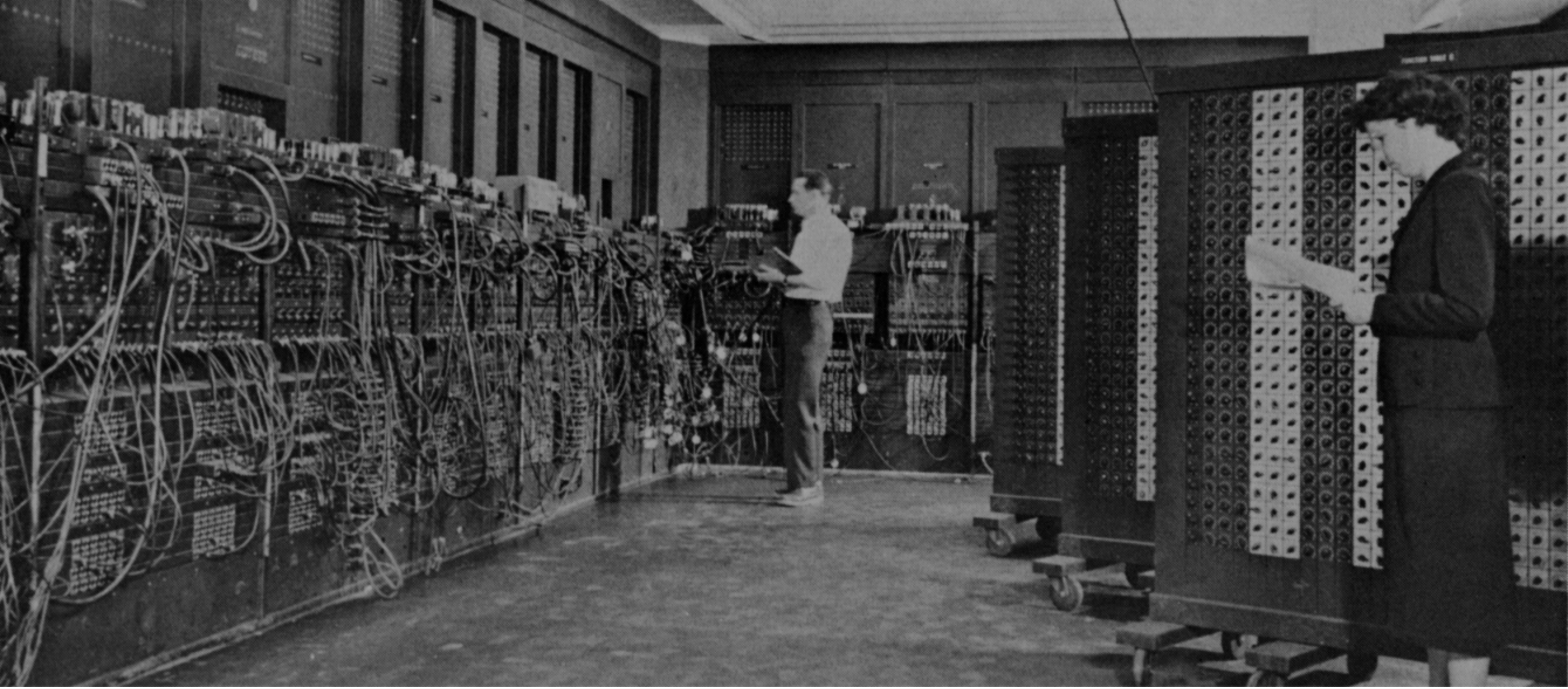


A Practical Guide to Your Computer

for Social and Policy Analysts

What are we up to?

- Where we've are: We want to get to data for analysis, but to do it in a structured and reliable way, we need to take care of the underlayment first
  - The very first part of the course: a lot of housekeeping
  - Tools, computational concepts, light programming
- Where we're going today:
  - A bit of history, a bit of science, a bit of data life advice

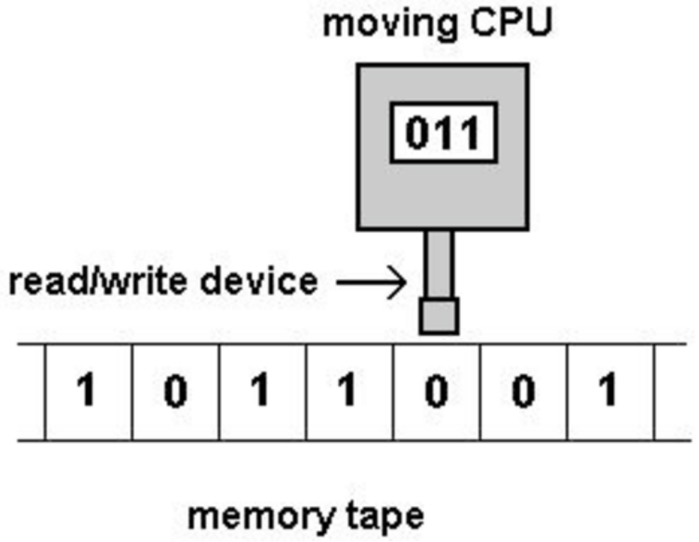
title page: ENIAC, the first programmable digital computer

What is a computer?

com·put·er | kəm'pyo͞odər |

noun

an electronic device for storing and processing data, typically in binary form, according to instructions given to it in a variable program: *my computer is frozen | the computer handles the entire process | [as modifier] : a computer program | there is a hugely expensive new computer system.*



a Turing machine

Course Housekeeping

- What are we doing?
  - We're on the road to data work - but we're taking care of some computational preliminaries first. If you are:
    - not terribly interested in the inner workings of a computer *and/or*
    - not that interested in doing arithmetic with R or learning about the differences between string, boolean, and numeric variables . . .
  - **That's OK!** Just bear with us for a little bit, because these things do become *useful* for the kinds of things you do want to do

Useful?

Verbatim, from job ads in public administration, policy, and campaigns:

- 1 “Quantitative, analytical and research skills. Ability to distill analysis into key results for broad consumption. A desire to grow expertise in a new subject area through research and analysis. **Skilled at analyzing reports and datasets to solve specific research objectives**.”
- 2. “Effective written and oral communication skills. Comfort with public speaking. Skill at **packaging and explaining complex concepts to a wide range of audiences through clear written materials**.”
- 3. “Experience using **GIS mapping** and/or Tableau for d**ata visualisation**”
- 4. “Ability to apply concepts from statistics and explain your decisions clearly to others”
- 5. “**Proficiency in R**, Python, Stata, or another statistical computing language”
- 6. “Experience in **data standardization and data manipulation**”
- 7. “Facility with research design, particularly survey design or experiment design”
- 8. “Experience in **natural language processing and textual analysis**”
- 9. “Proficiency with **GIS or other mapping software**”
- 10. “Conducting quantitative analysis using SPSS and/or **R statistical packages and other tools**”
- 11. “Building **data visualizations** and graphic representations of data”
- 12. “Writing analysis memos and summaries that provide strategic guidance to clients”
- 13. “Solid knowledge of **tidyverse R** and SQL”

Source Ads: Data science, The Messina Group, Energy Campaigns Analyst, Sierra Club,Research Associate, WPA Intelligence, Data science, WPA Intelligence

Course Housekeeping

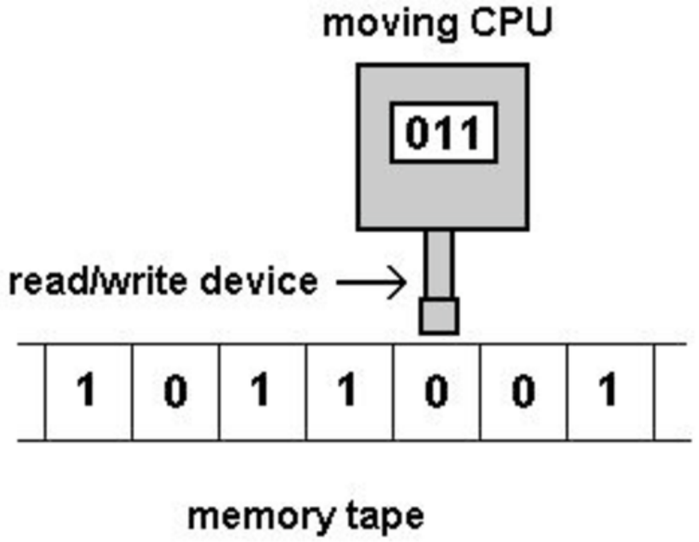
- Reminders:
  - Course materials in three places:
    - **Website** (jacklreilly.github.io/dwv_f25) - syllabus, assignments, readings
    - **Blackboard** - assignment submission
    - **Course folder** - course slides, data, etc
  - Does everyone have access to everything?

What is a computer?

com·put·er | kəm'pyo͞odər |

noun

an electronic device for storing and processing data, typically in binary form, according to instructions given to it in a variable program: *my computer is frozen | the computer handles the entire process | [as modifier] : a computer program | there is a hugely expensive new computer system.*



An illustration of a Turing Machine

title page: ENIAC, the first programmable digital computer

What is a computer?

- Really two things:
  - **Memory** - stored data that gets manipulated
  - **Processor** - thing that reads data, manipulates it, and writes it back to memory
    - according to a series of **instructions**

Today

- Talk about three things
  - Hardware: what a computer is
  - Software: how we give a computer commands
  - Tools: the software we use to give computers commands

Hardware

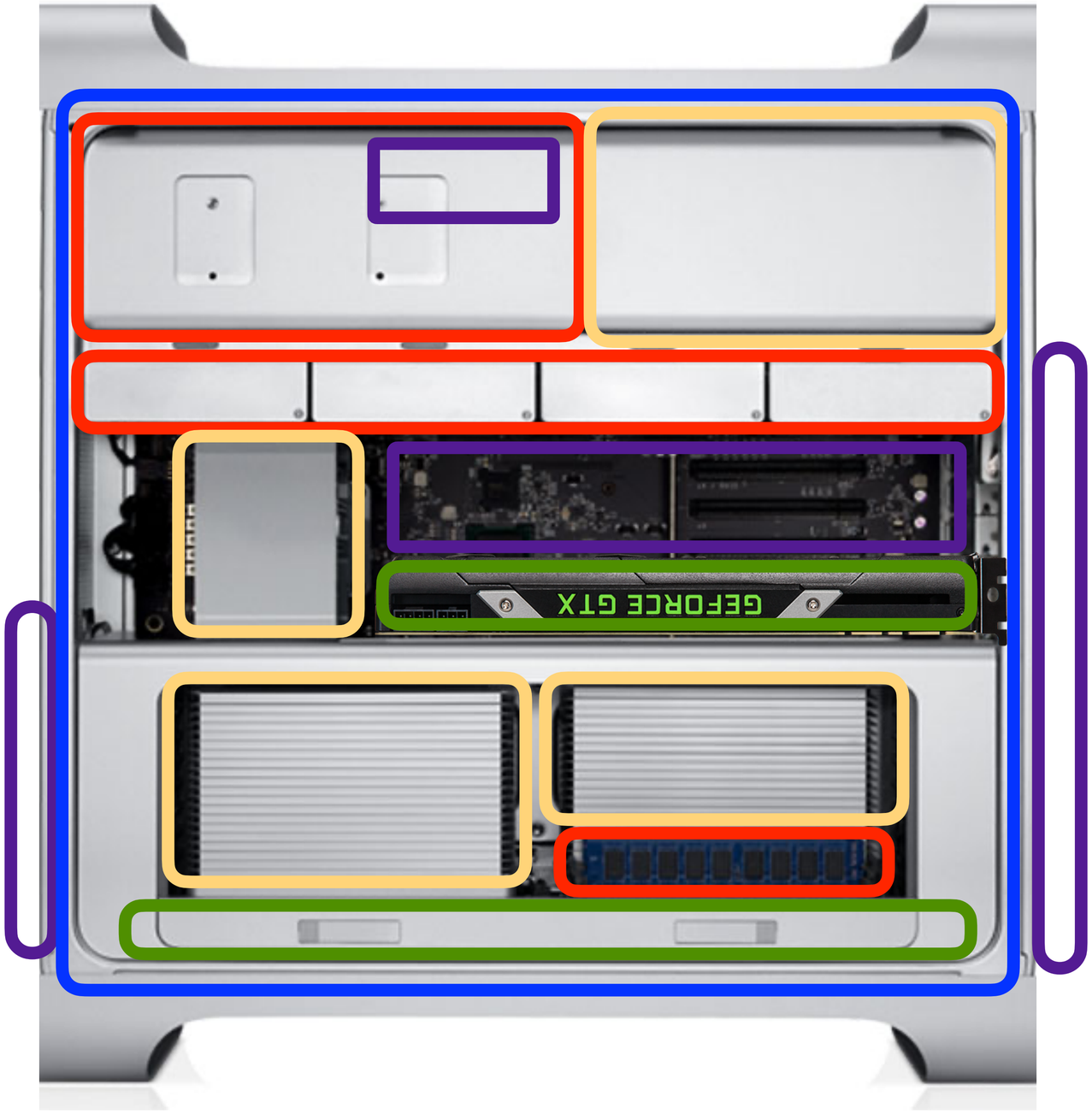
Processing

Core components

- **Processing**
  - The thing that performs operations
- **Memory**
  - Where data from those operations is read and stored

It must be more complex, right? (It is.)

- Each computer has a variety of different kinds of processing units:
  - Central Processing Unit (CPU), Graphics Processing Unit (GPU), Neural Processing Unit (NPU), Tensor Processing Unit (TPU), Media Processors, other *hard-coded* computational engines for specific purposes
- And each can have a variety of different kinds of memory:
  - Primary memory (RAM), secondary memory (drives/hard disks/solid state drives), tertiary/removable storage (usb sticks, floppy drives, CD/DVD), off-site (not under direct control of the processing units)
- All of which must be properly connected together through a databus (or just bus) that then must be powered and cooled while allowing for other I/O and connectivity to other computers and humans


GEFORCE GTX

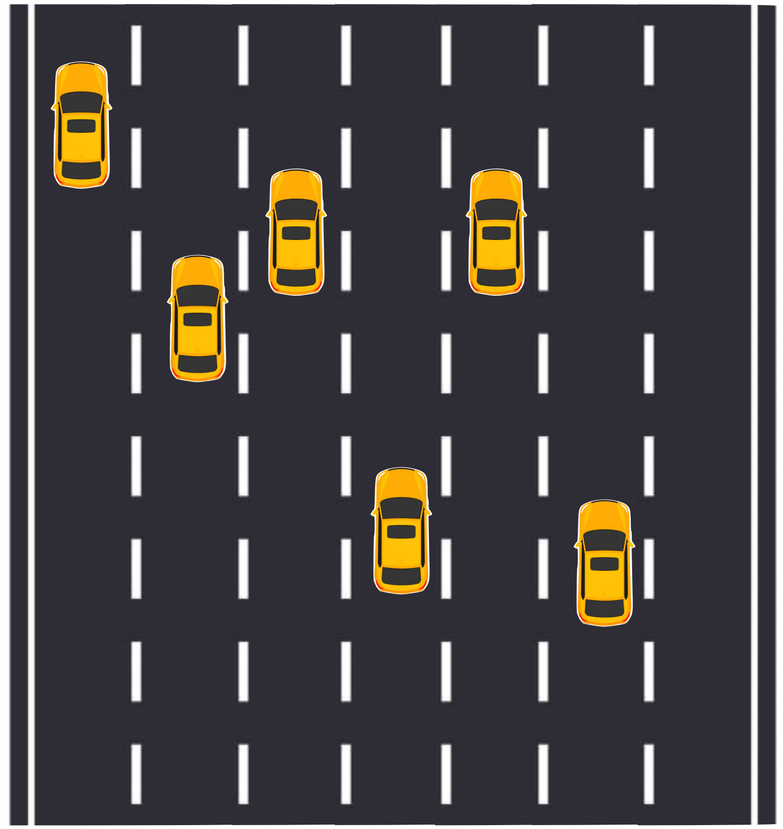
Different processing units?

- Fundamental theory in computing (Universal Turing Machine) states that any sufficient computing machine can, in theory, simulate another machine
- In other words, any computable sequence can be run by a UTM
  - But it does not say anything about **efficiency**
- We care about two forms of efficiency, as well:
  - Computing time
  - Energy cost

The very basics of computer processing

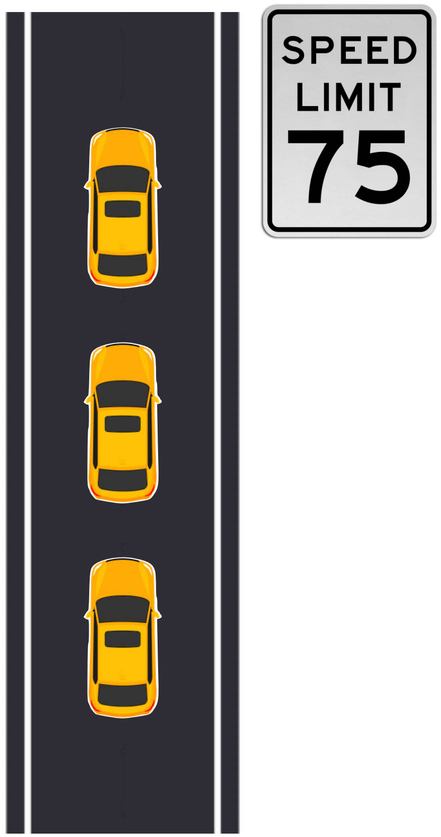
Serial

Parallel

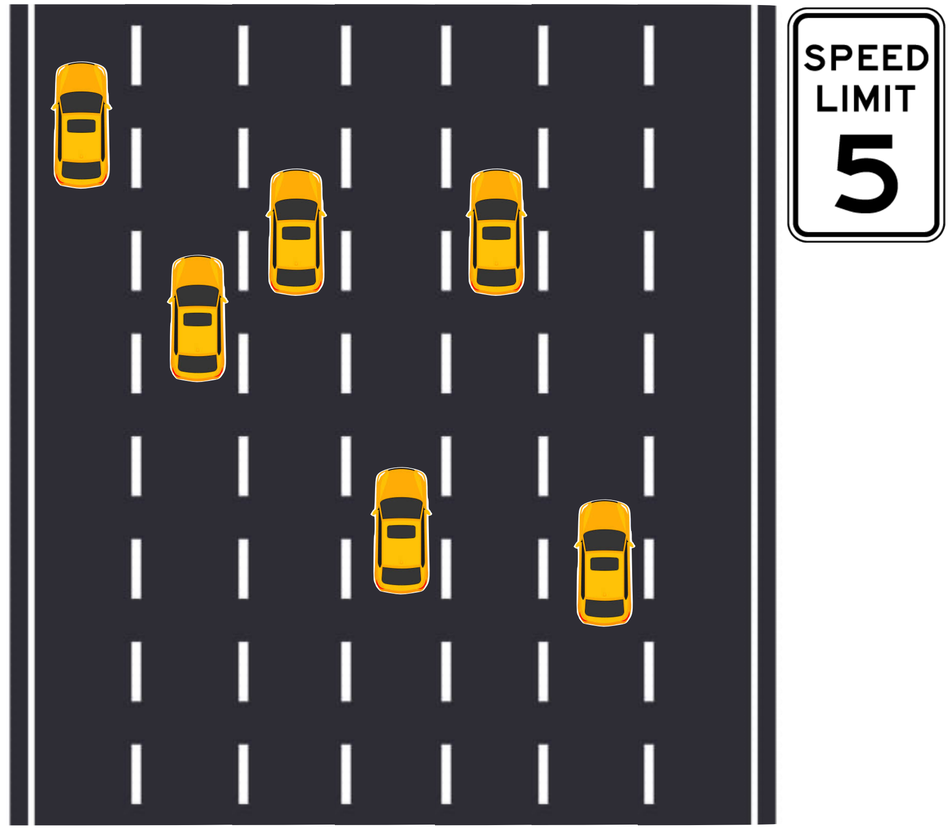


Why wouldn’t you always use parallel processing?

Serial

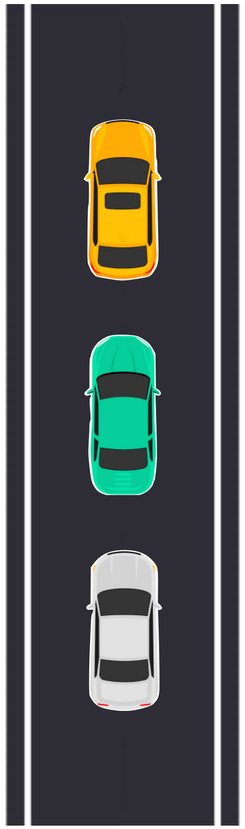


Parallel



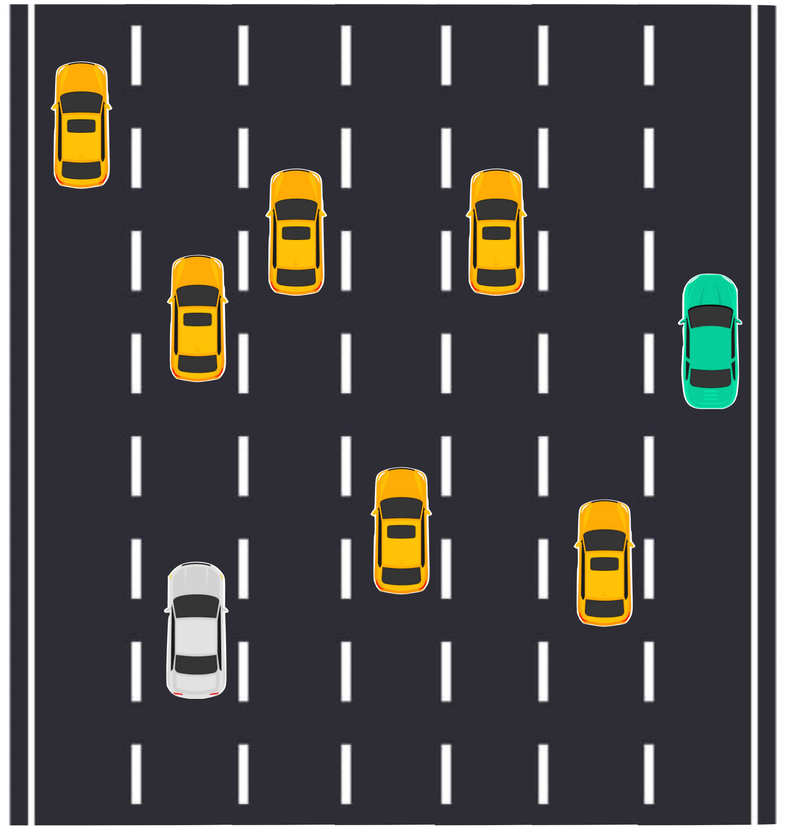
Why wouldn’t you always use parallel processing?

Serial



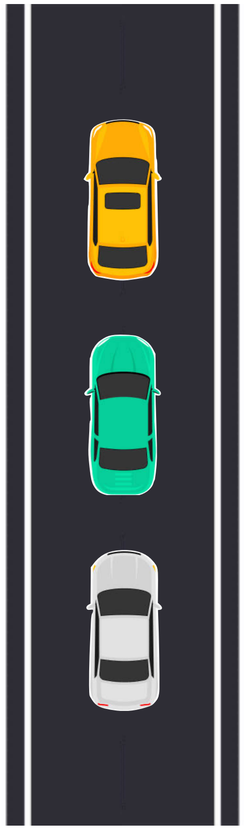
What if we have *contingent* processes?

Parallel



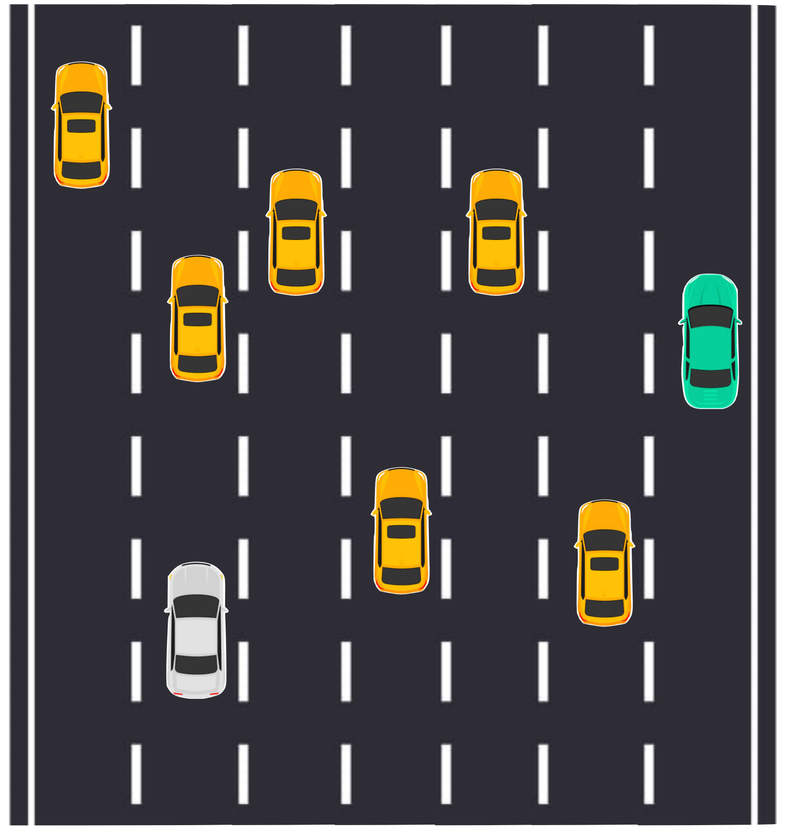
Why wouldn’t you always use parallel processing?

Serial



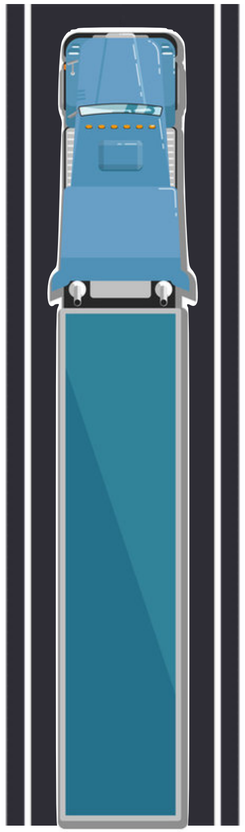
How do we get all these cars on and off efficiently?

Parallel



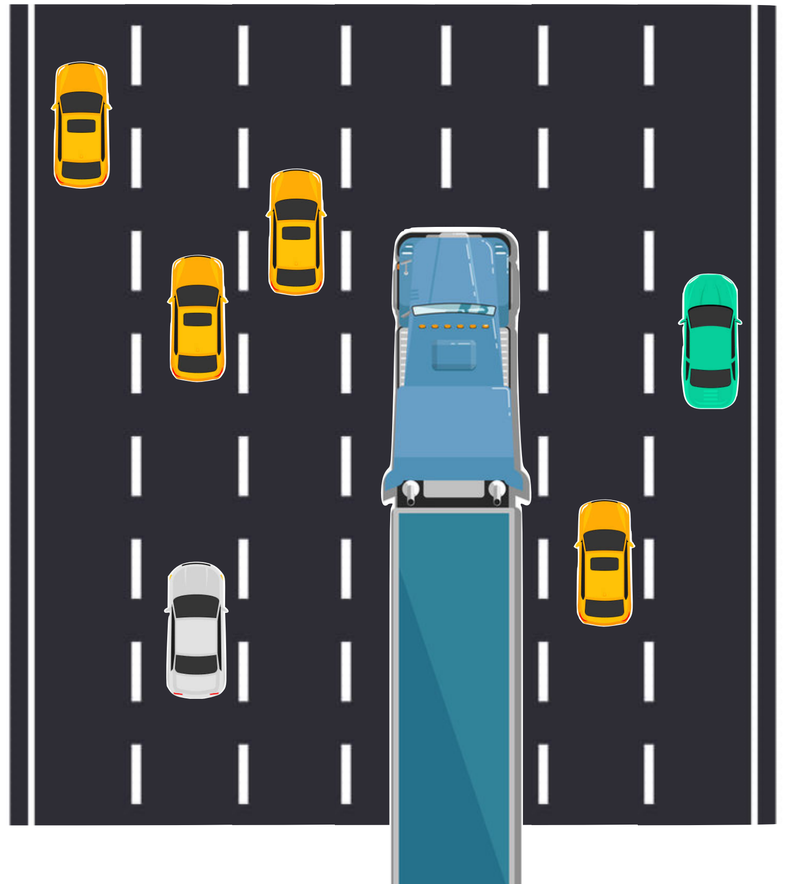
Why wouldn’t you always use parallel processing?

Serial



Serial process lines can often fit bigger trucks, more complex tasks

Parallel



Different computing problems, different solutions

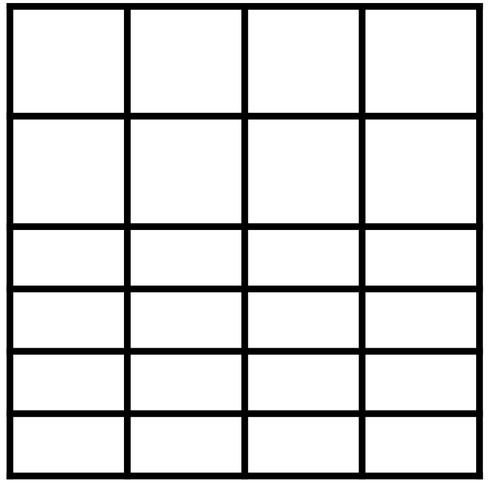
- Some computing problems are easily *parallelizable*
  - Image and video decoding, many AI tasks, some data tasks
- Some computing problems are more naturally *serial*
  - Contingent processes, simple processes with little practical benefit to complex organizational efforts, many kinds of user interaction

CPUs and GPUs are generally targeted at these two different spaces

- The CPU has a small number of **very fast**, **very big**, driving lanes (**processing cores**)
- The GPU has many **slower, smaller, less capable** driving lanes (processing cores)
  - But has **so many** it makes up for lack of capability through brute force
- Other specialized processing units are even more specialized, able to take only a very particular kind of car going in a very particular direction (NPU/TPUs, media encoders, etc)

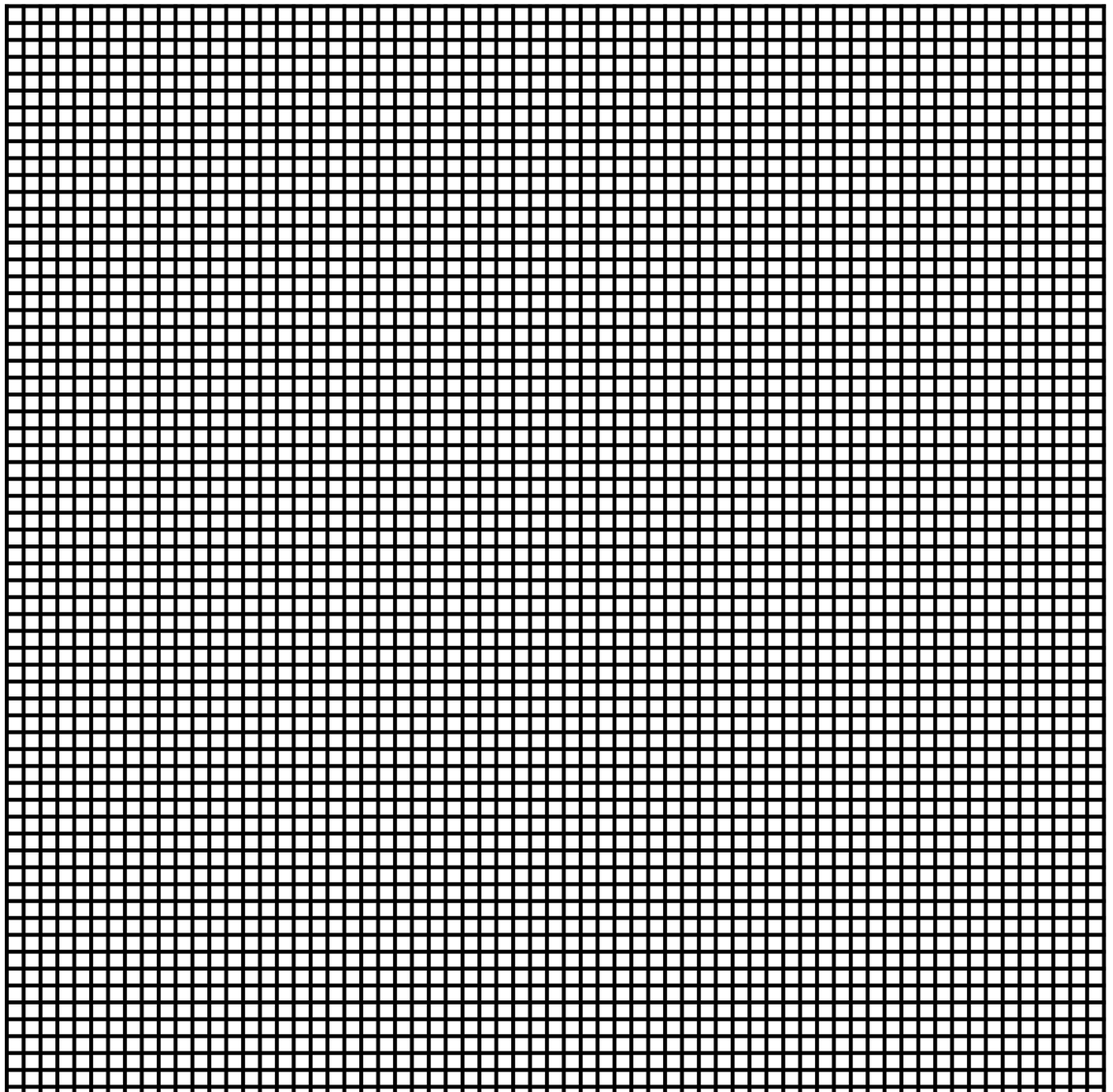
Intel i9-14900K

8 Performance Cores, 16 Efficiency Cores



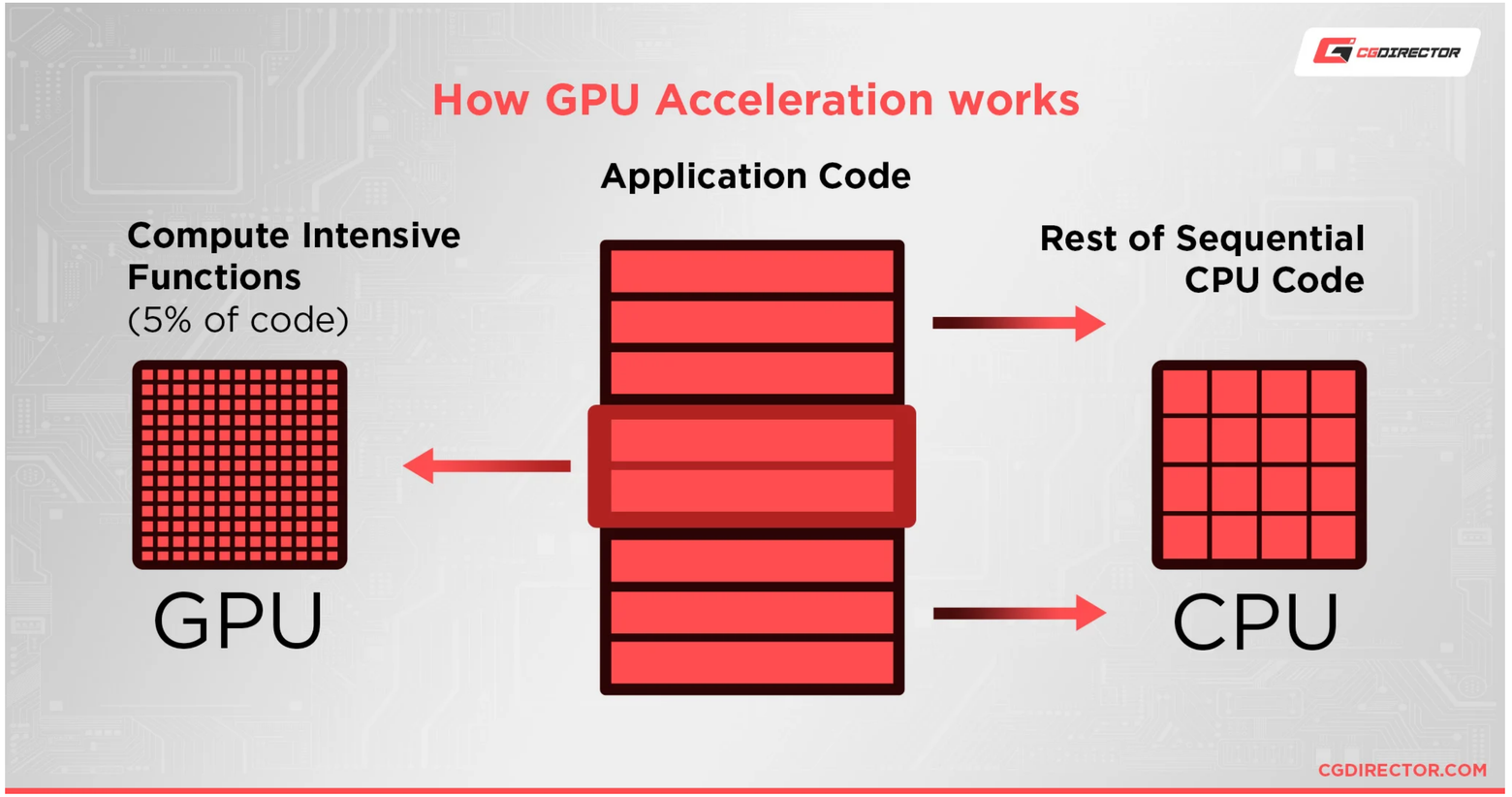
Nvidia GeForce GTX 5090

21,760 CUDA Cores



In practice

- The CPU controls the “computer”, while handing off special tasks that are particularly valuable to happen at greater speed in parallel to specialized processing units (like the GPU)



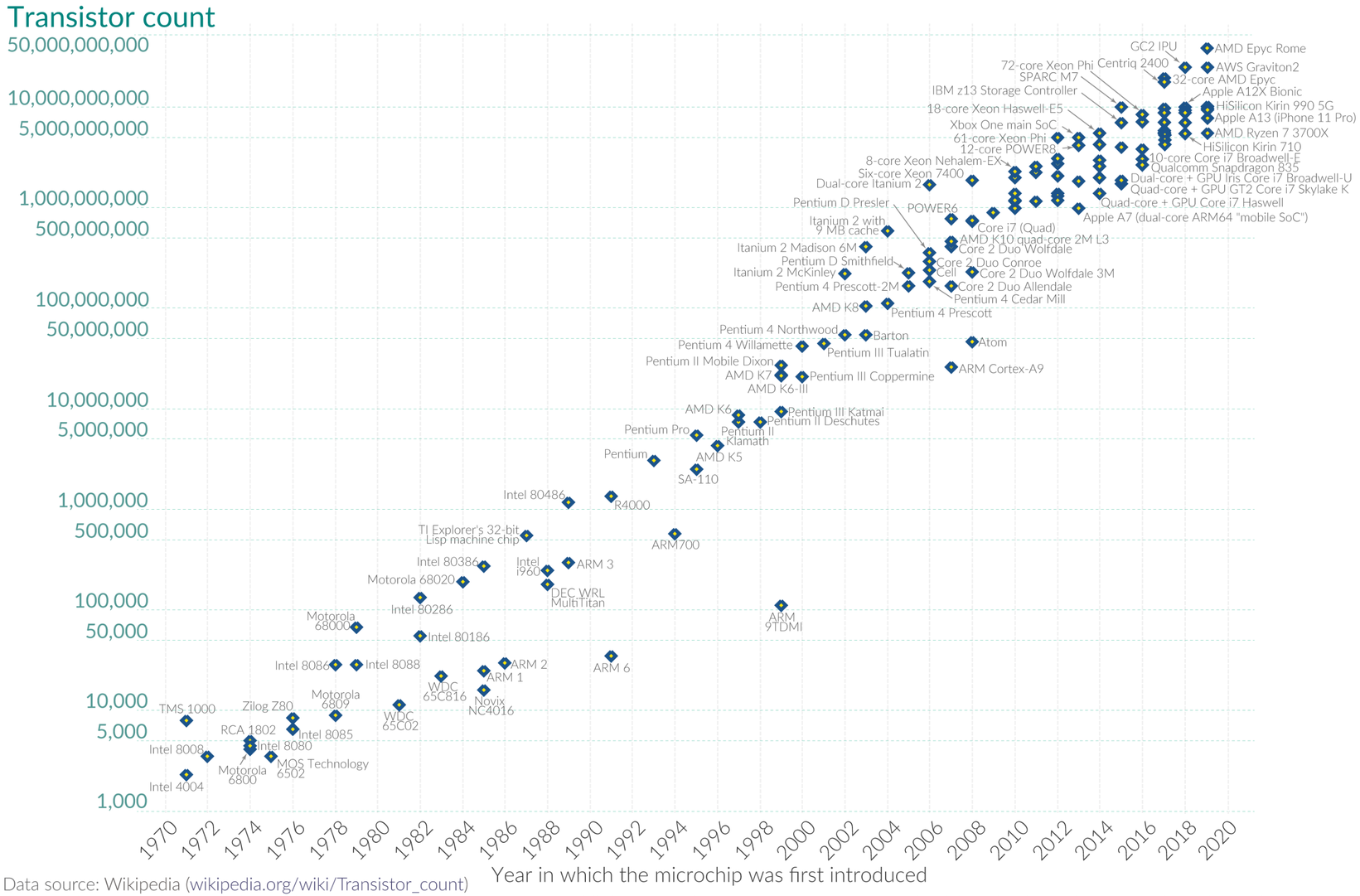
Computers get more powerful over time

- Moore’s Law: the number of transistors doubles every two years
- This process, however, is not automatic
- It is **hard** to make a processor, requiring exceptionally specialized equipment and technology
- Most speed increases come from **die shrinks**, along with design optimizations

Moore’s Law: The number of transistors on microchips doubles every two years

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 for other aspects of technological progress in computing – such as processing speed or the price of computers.



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

Software

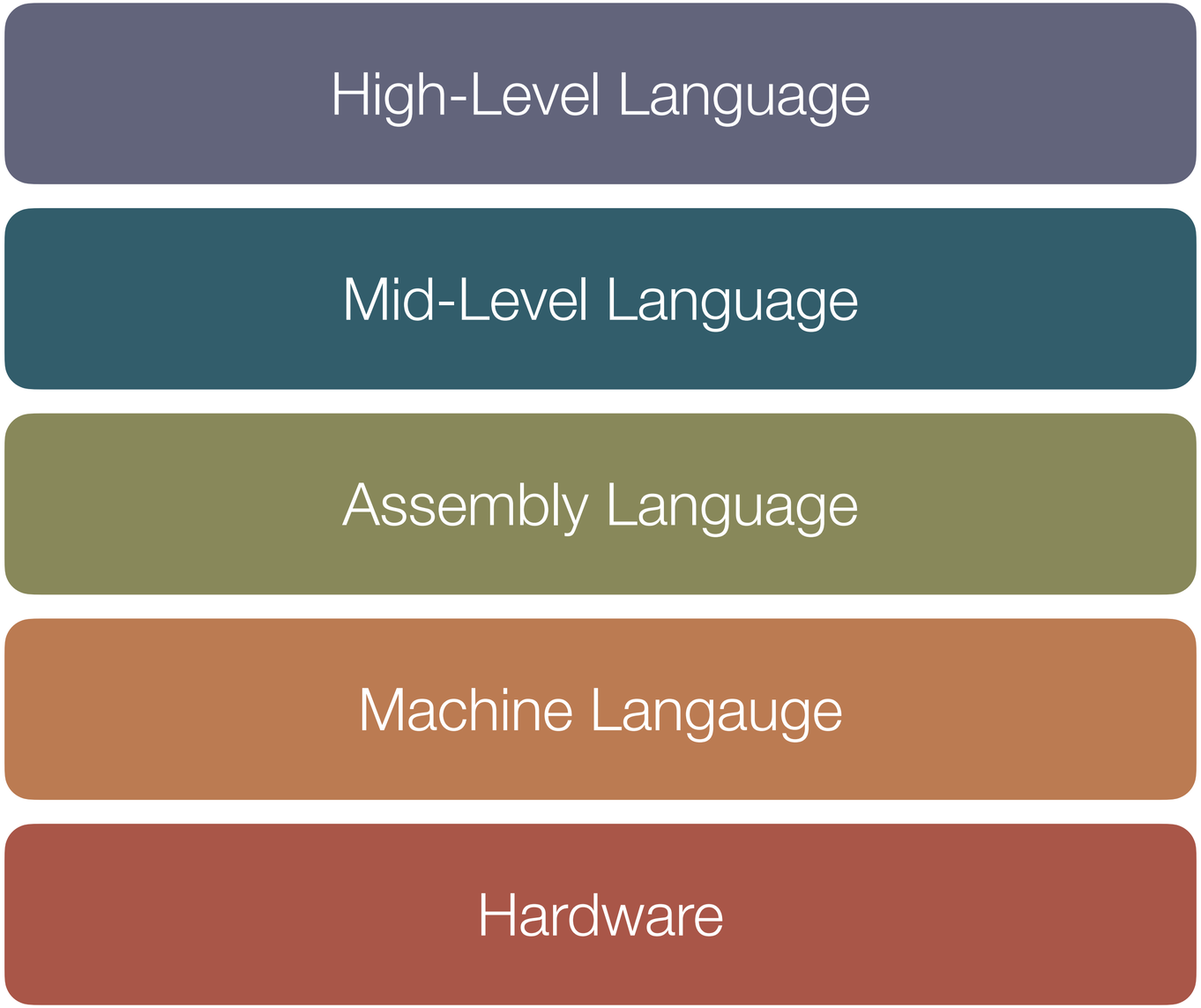
Programming

Programming

- How we tell a computer what we want it to do: a sequence of commands and instructions for the computer to execute
- Hardware takes a very limited number of commands: fundamentally, it all breaks down to manipulating little 0s and 1s
- We program a computer by writing **programs** and calling **libraries** and **APIs** within programming environments
  - **Algorithm** - a process or set series of rules for a computer to follow
  - **Computer Program** - takes certain inputs and returns certain outputs

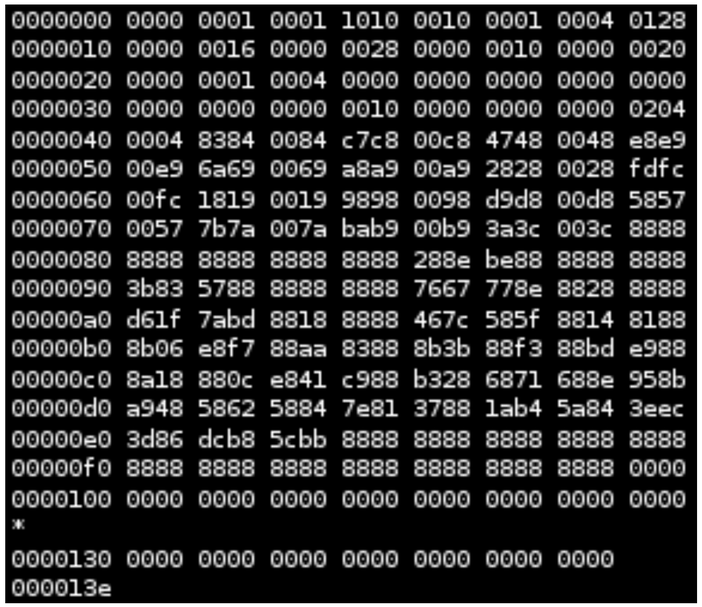
```
15 library(tidyverse)
16 library(osmdata) # package for working with streets
17 library(showtext) # for custom fonts
18 library(ggmap)
19 library(sf)
20 #library(rvest) #not needed here
21
22 #you can find all the available features of OSM here:
23 #https://wiki.openstreetmap.org/wiki/Map_features
24 #or here (can be slow, uncomment to see):
25 available_features()
26
27 #Also use openstreetmap.org to inspect features
28
29 available_tags("highway")
30
31 #getbb = "Get Bounding Box"
32 srqbounds<-getbb("Onondaga County New York")
33
34 #Now we're going to "pipe" commands together to build our map
35 #Piping is another way to pass commands directly to other commands
36 #Essentially: get our bounding box, <pass>, build an openstreemap query (opq), <pass>,
37 #add features, <pass>, return query as formatted SF object
38
39 srq_big_streets <- getbb("Onondaga County New York") %>%
40   opq() %>%
41   add_osm_feature(key = "highway",
42                   value = c("motorway", "trunk", "primary", "motorway_link", "trunk_link", "primary_lin
43   osmdata_sf()
44
45 #Now let's see what we have!
46 srq_big_streets
47
48 #Remember - the core components of graphing.  Points, Lines, Polygons (and iterations on those)
49   # osm_lines, osm_points, osm_polygons
50
51 #Time to do some plotting with our old friend, GGPlot
52 #geom_sf - plot "simple feature" objects (the kind used in GIS)
53
54 ggplot() +
55   geom_sf(data = srq_big_streets$osm_lines,
56           inherit.aes = FALSE,
57           color = "black",
58           size = 1)
59
60 #Can play with aesthetics, like always
61
62 ggplot() +
63   geom_sf(data = srq_big_streets$osm_lines,
64           inherit.aes = FALSE,
65           color = "purple",
66           size = 8)
```

An exceptionally simple program


High-Level Language
Mid-Level Language
Assembly Language
Machine Langauge
Hardware

Machine code

Binary



Hexdecimal

Assembly

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


 C000                    ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START    LDS    #STACK

                *************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                *************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                *************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
```

A program in C (“mid” level; compiled)

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

struct stats { int count; int sum; int sum_squares; };

void stats_update(struct stats * s, int x, bool reset) {
    if (s == NULL) return;
    if (reset) * s = (struct stats) { 0, 0, 0 };
    s->count += 1;
    s->sum += x;
    s->sum_squares += x * x;
}

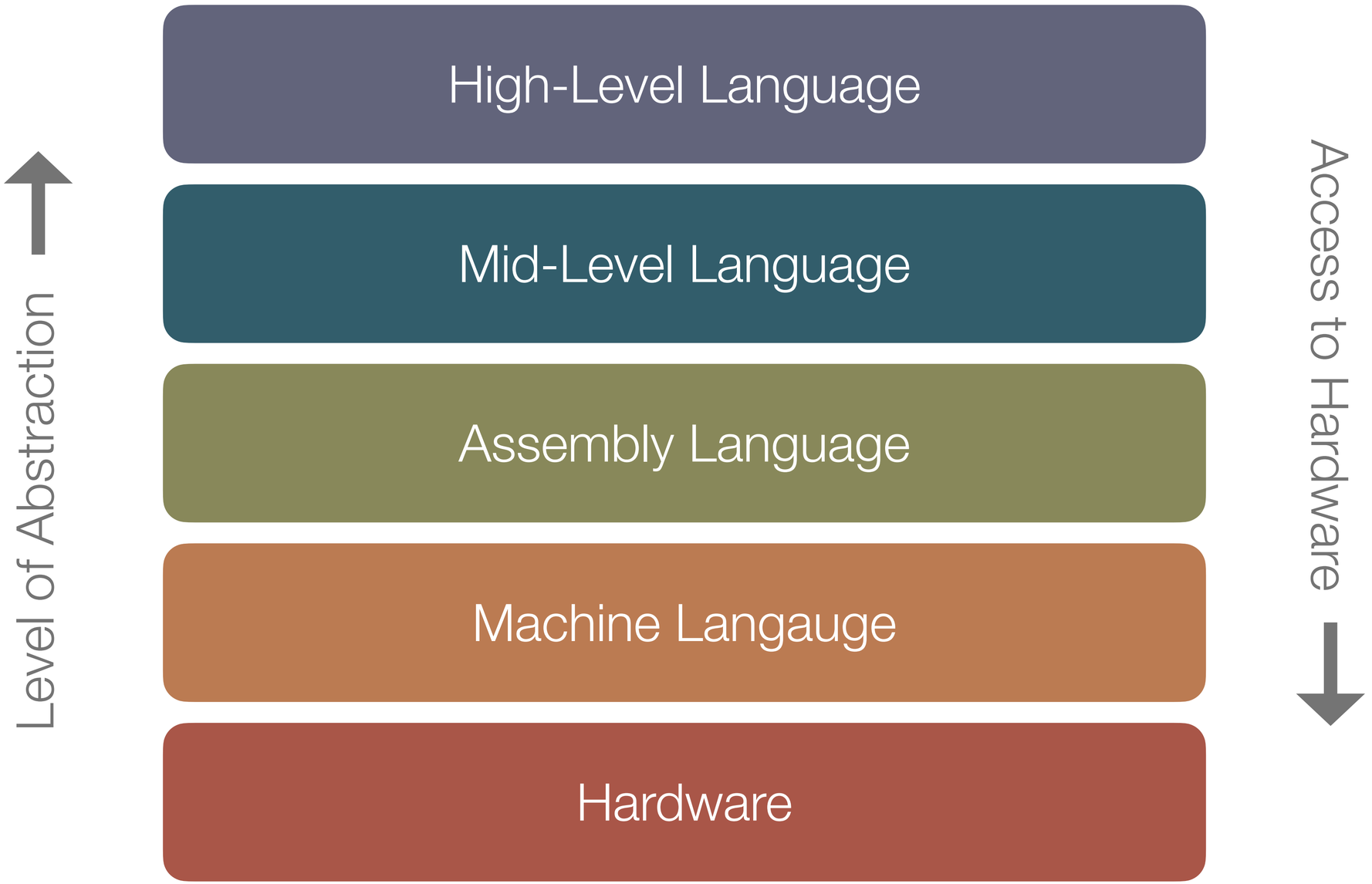
double mean(int data[], size_t len) {
    struct stats s;
    for (int i = 0; i < len; ++i)
        stats_update(&s, data[i], i == 0);
    return ((double)s.sum) / ((double)s.count);
}

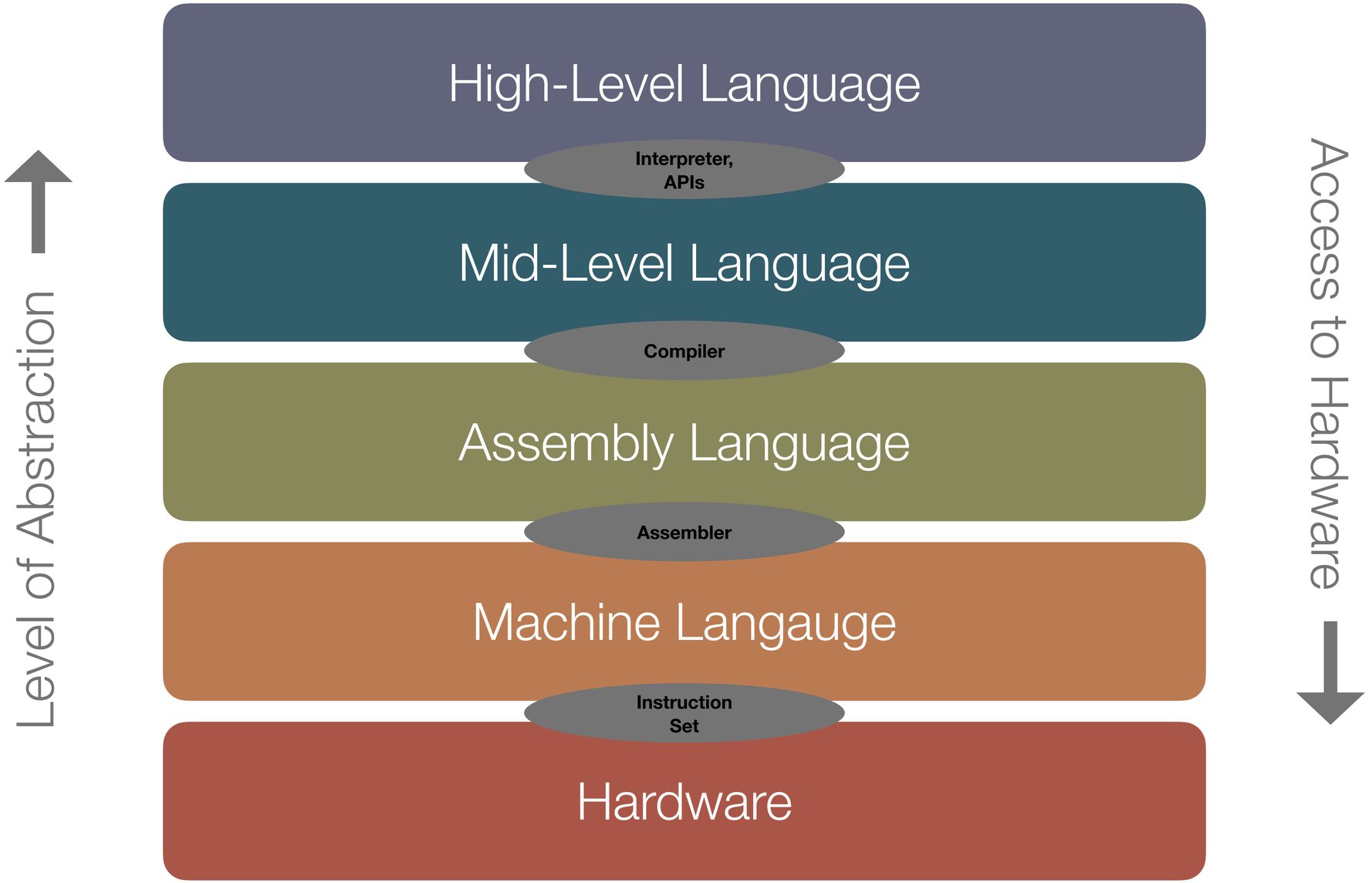
void main() {
    int data[] = { 1, 2, 3, 4, 5, 6 };
    printf("MEAN = %lf\n", mean(data, sizeof(data) / sizeof(data[0])));
}
```

this code calculates a mean

A program in R ("high" level, interpreted)

```
15  library(tidyverse)
16  library(osmdata) # package for working with streets
17  library(showtext) # for custom fonts
18  library(ggmap)
19  library(sf)
20  #library(rvest) #not needed here
21
22  #you can find all the available features of OSM here:
23  #https://wiki.openstreetmap.org/wiki/Map_features
24  #or here (can be slow, uncomment to see):
25  available_features()
26
27  #Also use openstreetmap.org to inspect features
28
29  available_tags("highway")
30
31  #getbb = "Get Bounding Box"
32  srqbounds<-getbb("Onondaga County New York")
33
34  #Now we're going to "pipe" commands together to build our map
35  #Piping is another way to pass commands directly to other commands
36  #Essentially: get our bounding box, <pass>, build an openstreemap query (opq), <pass>,
37  #add features, <pass>, return query as formatted SF object
38
39  srq_big_streets <- getbb("Onondaga County New York") %>%
40    opq() %>%
41    add_osm_feature(key = "highway",
42                    value = c("motorway", "trunk", "primary", "motorway_link", "trunk_link", "primary_link")) %>%
43    osmdata_sf()
44
45  #Now let's see what we have!
46  srq_big_streets
```

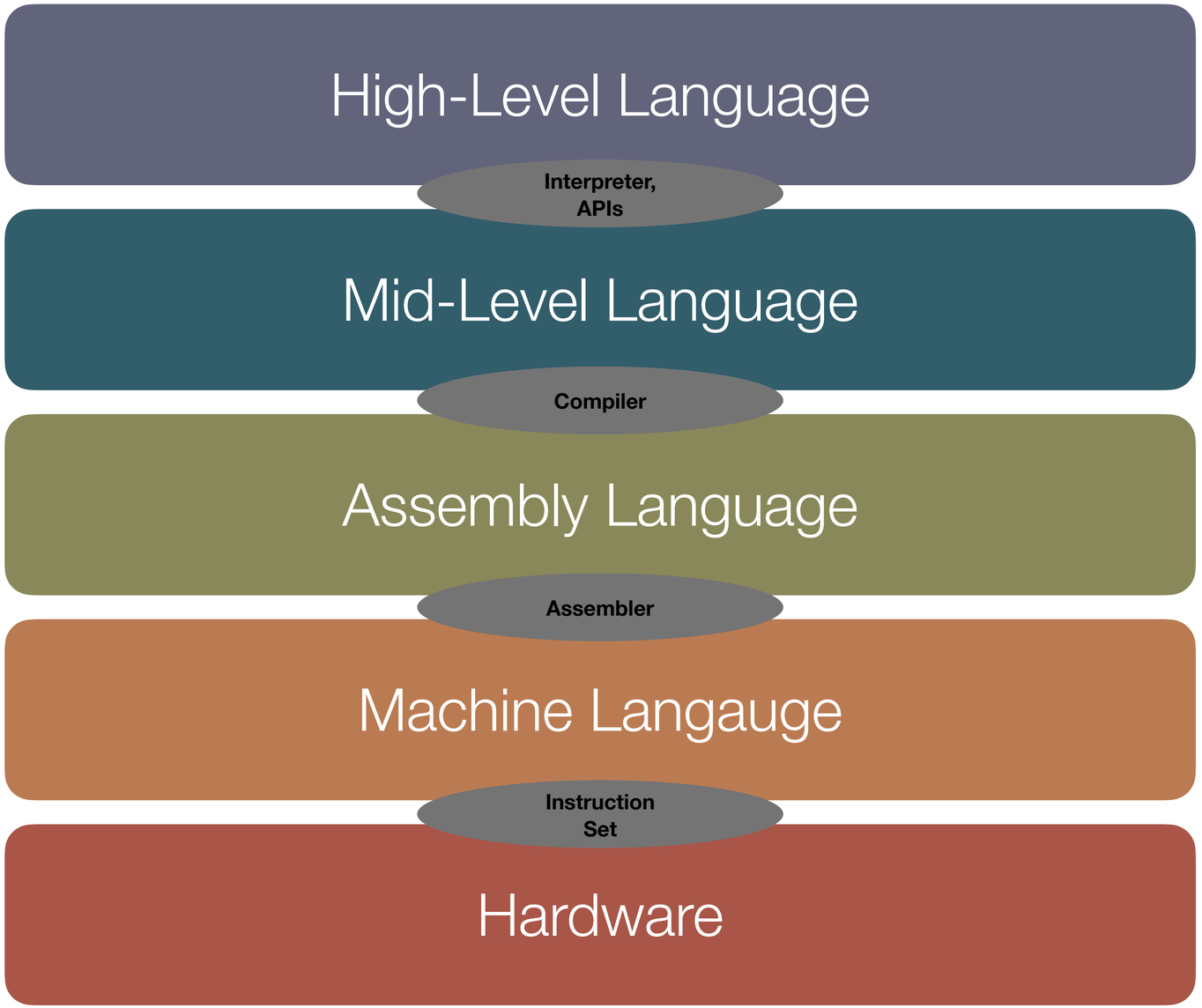

High-Level Language
Mid-Level Language
Assembly Language
Machine Langauge
Hardware
Level of Abstraction
Access to Hardware



*heavily simplified and wrong in some fundamental ways

Langauge tradeoffs

- Higher level languages and APIs (application programming environments), in general on average, **increase** human readability at a cost of **speed** and **hardware access**
- **“Understand your stack”** - understand all the **dependencies** your code has
- Dependencies - not just hardware levels, but software levels and dependences as well
  - Software can call other software to perform functions



*heavily simplified and wrong in some fundamental ways

What do we do when we compute?

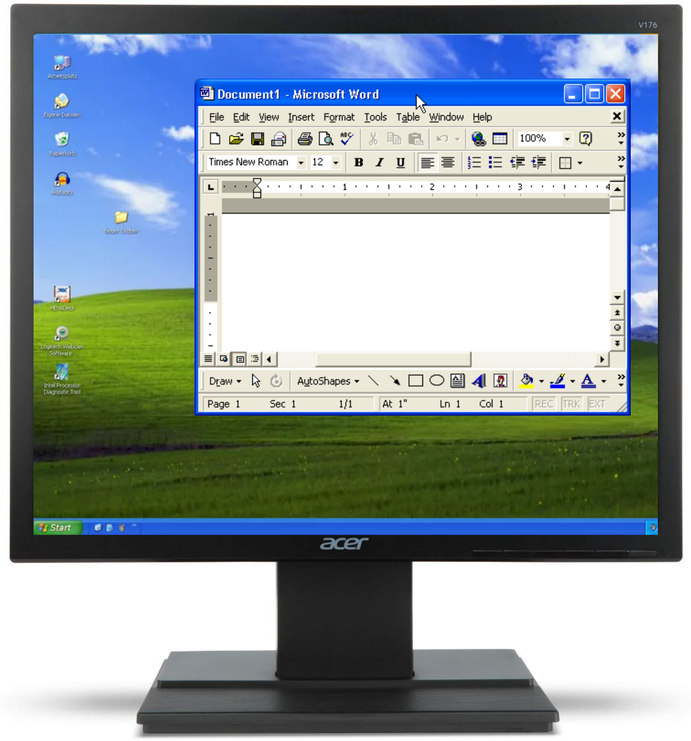
- We’re - to a first approximation - just doing the Turing machine thing
  - **Command** (through software or user interface)
    - a computer to **perform an operation** (analyze data, play a movie, edit a picture)
      - on **data** (tabular data, video file, important cat .gif)
- **How do we command our computers to do things?**

so much power to show us cute pictures of kittehs

The great computing divorce

- How you probably like to use a computing device

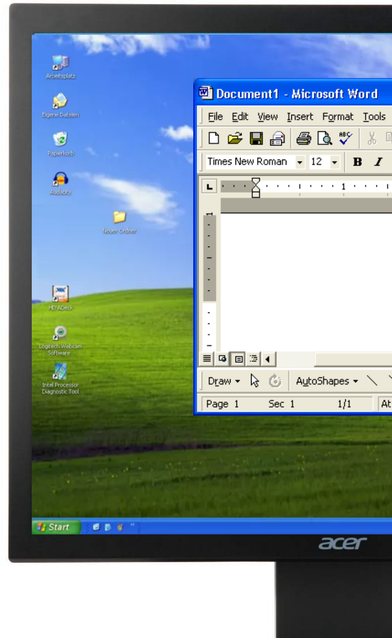
- How you probably use a computing device for work



The great computing divorce

- Snazzy and colorful!
- Touch the screen!
- Apps!
- One application at a time
- File system? What file system?
- It’s the 2020s!

- Boxy desk-bound
- Click the mouse
- Big applications
- Lots of windows
- Files in folders
- Y2k is calling



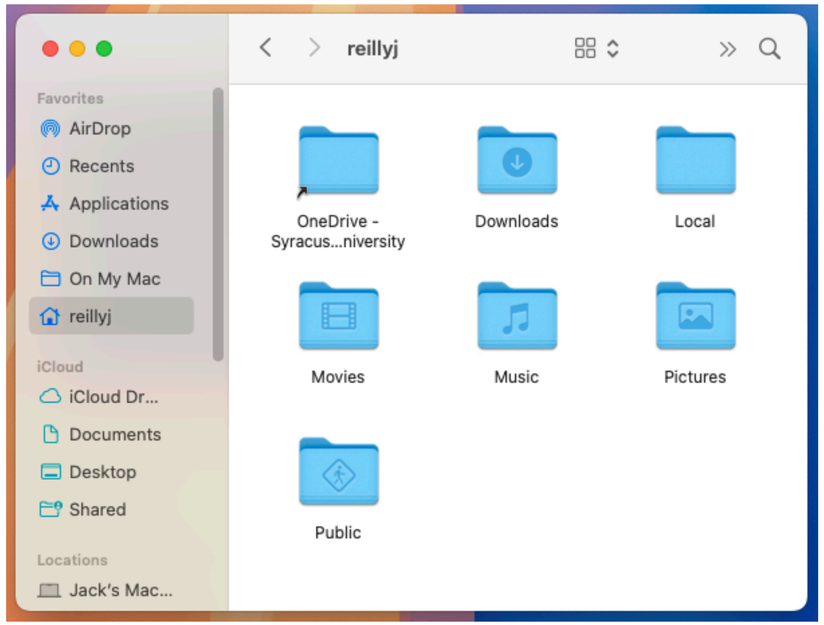
Within the work computer divide, however, lies another divide

A deeper, darker past . . .

Two interface paradigms

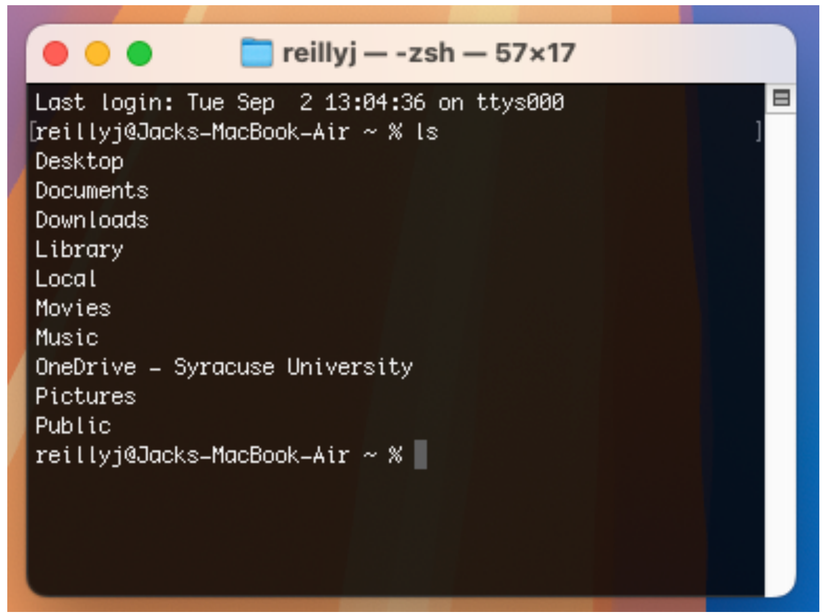
GUIs

Graphical User Interfaces



CLIs

Command Line Interfaces



This is where the heart of technical computing (still) resides

- UNIX, the shell, and the command line
  - UNIX is an operating system used for mainframes originally developed in 1969
- Its derivatives live on today, at the heart of (most) computing systems
  - MacOS, Linux, iOS, Android, ChromeOS, Kindles, even your watch . . . all UNIX
    - Even Windows has a Linux subsystem
- The vast majority of people that use their computers for code live a great portion of their computing lives in software inspired by the 1970s

IBM 1401 mainframes, 1969
https://www.computerhistory.org/revolution/mainframe-computers/7/166/663

The **Good** News

We're **not** going to exclusively use the shell (CLI)

The **Complicating** News

but the kinds of work we do is going to be heavily influenced by it, and it's going to be useful to drop down to it every now and then

Two models of work computing

- **Office model**
  - Formatted documents, application-centric files
  - Intermediate products are cut and pasted, often without link backs or references to source
  - Changes are tracked by application
  - Editing window = formatted window (WYSIWYG)
- **Engineering** model
  - Open, plain text documents, readable with many applications
  - Intermediate products produced via code, linked with reference to source
  - Changes tracked outside files, at the project level
  - Final outputs programmatically *assembled* to final document

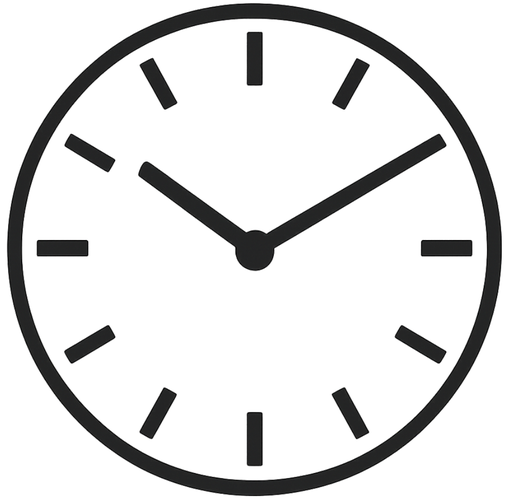
Oooh, a new way to use the computer! Is the engineering model fun?

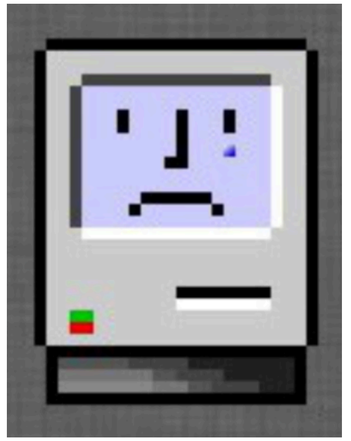
Oooh, a new way to use the computer! Is the engineering model fun?

- Well, no. At times, it can feel more like this.

Oh, OK, but can we make it better?

- Yes! But it takes a long time and a lot of practice.





Uh, well, why do we do it?

- Two **very, very good** reasons

Also, once you get used to it, you may like it

- And it kinda starts to feel like this

CONTROL

- and -

Reproducibility

(also, it's weirdly fun when you get used to it)

So why do we care so much about this?

- Research is difficult.
- It’s easy to make small mistakes that become large errors.
- Doing it badly - or fraudulently - is too easy.
- And even when we don’t make mistakes, and put in our best effort, sometimes we’re . . . just wrong

The Replication Crisis

- A genuinely surprisingly number of scientific and policy articles do not **replicate**
  - **Replicate:** obtain the same, or a similar, finding upon conducting the **same study**
- **Behavioral sciences** are particularly at risk (pesky humans)

News | Published: 27 August 2015

Over half of psychology studies fail reproducibility test

Monya Baker

Nature (2015) | Cite this article

53k Accesses | **123** Citations | **1347** Altmetric | Metrics

Largest replication study to date casts doubt on many published positive results.

Why Most Published Research Findings Are False

John P. A. Ioannidis

Summary

There is increasing concern that most current published research findings are false. The probability that a research claim is true may depend on study power and bias, the number of other studies on the same question, and, importantly, the ratio of true to no relationships among the relationships probed in each scientific field. In this framework, a research finding is less likely to be true when the studies conducted in a field are smaller; when effect sizes are smaller; when there is a greater number and lesser preselection of tested relationships; where there is greater flexibility in designs, definitions, outcomes, and analytical modes; when there is greater financial and other interest and prejudice; and when more teams are involved in a scientific field in chase of statistical significance. Simulations show that for most study designs and settings, it is more likely for a research claim to be false than true. Moreover, for many current scientific fields, claimed research findings may often be simply accurate measures of the prevailing bias. In this essay, I discuss the implications of these problems for the conduct and interpretation of research.

Published research findings are sometimes refuted by subsequent evidence, with ensuing confusion and disappointment. Refutation and controversy is seen across the range of research designs, from clinical trials and traditional epidemiological studies [1–3] to the most modern molecular research [4,5]. There is increasing concern that in modern research, false findings may be the majority or even the vast majority of published research claims [6–8]. However, this should not be surprising. It can be proven that most claimed research findings are false. Here I will examine the key factors that influence this problem and some corollaries thereof.

The Essay section contains opinion pieces on topics of broad interest to a general medical audience.

Modeling the Framework for False Positive Findings

Several methodologists have pointed out [9–11] that the high rate of nonreplication (lack of confirmation) of research discoveries is a consequence of the convenient, yet ill-founded strategy of claiming conclusive research findings solely on the basis of a single study assessed by formal statistical significance, typically for a *p*-value less than 0.05. Research is not most appropriately represented and summarized by *p*-values, but, unfortunately, there is a widespread notion that medical research articles

It can be proven that most claimed research findings are false.

should be interpreted based only on *p*-values. Research findings are defined here as any relationship reaching formal statistical significance, e.g., effective interventions, informative predictors, risk factors, or associations. "Negative" research is also very useful. "Negative" is actually a misnomer, and the misinterpretation is widespread. However, here we will target relationships that investigators claim exist, rather than null findings.

As has been shown previously, the probability that a research finding is indeed true depends on the prior probability of it being true (before doing the study), the statistical power of the study, and the level of statistical significance [10,11]. Consider a 2 × 2 table in which research findings are compared against the gold standard of true relationships in a scientific field. In a research field both true and false hypotheses can be made about the presence of relationships. Let R be the ratio of the number of "true relationships" to "no relationships" among those tested in the field. R is characteristic of the field and can vary a lot depending on whether the field targets highly likely relationships or searches for only one or a few true relationships among thousands and millions of hypotheses that may be postulated. Let us also consider, for computational simplicity, circumscribed fields where either there is only one true relationship (among many that can be hypothesized) or the power is similar to find any of the several existing true relationships. The pre-study probability of a relationship being true is $R/(R + 1)$. The probability of a study finding a true relationship reflects the power $1 - \beta$ (one minus the Type II error rate). The probability of claiming a relationship when none truly exists reflects the Type I error rate, α. Assuming that c relationships are being probed in the field, the expected values of the 2 × 2 table are given in Table 1. After a research finding has been claimed based on achieving formal statistical significance, the post-study probability that it is true is the positive predictive value, PPV. The PPV is also the complementary probability of what Wacholder et al. have called the false positive report probability [10]. According to the 2 × 2 table, one gets $PPV = (1 - \beta)R/(R - \beta R + \alpha)$. A research finding is thus

Citation: Ioannidis JPA (2005) Why most published research findings are false. PLoS Med 2(8): e124.

Abbreviation: PPV, positive predictive value

John P. A. Ioannidis is in the Department of Hygiene and Epidemiology, University of Ioannina School of Medicine, Ioannina, Greece, and Institute for Clinical Research and Health Policy Studies, Department of Medicine, Tufts-New England Medical Center, Tufts University School of Medicine, Boston, Massachusetts, United States of America. E-mail: jioannid@cc.uoi.gr

Competing Interests: The author has declared that no competing interests exist.

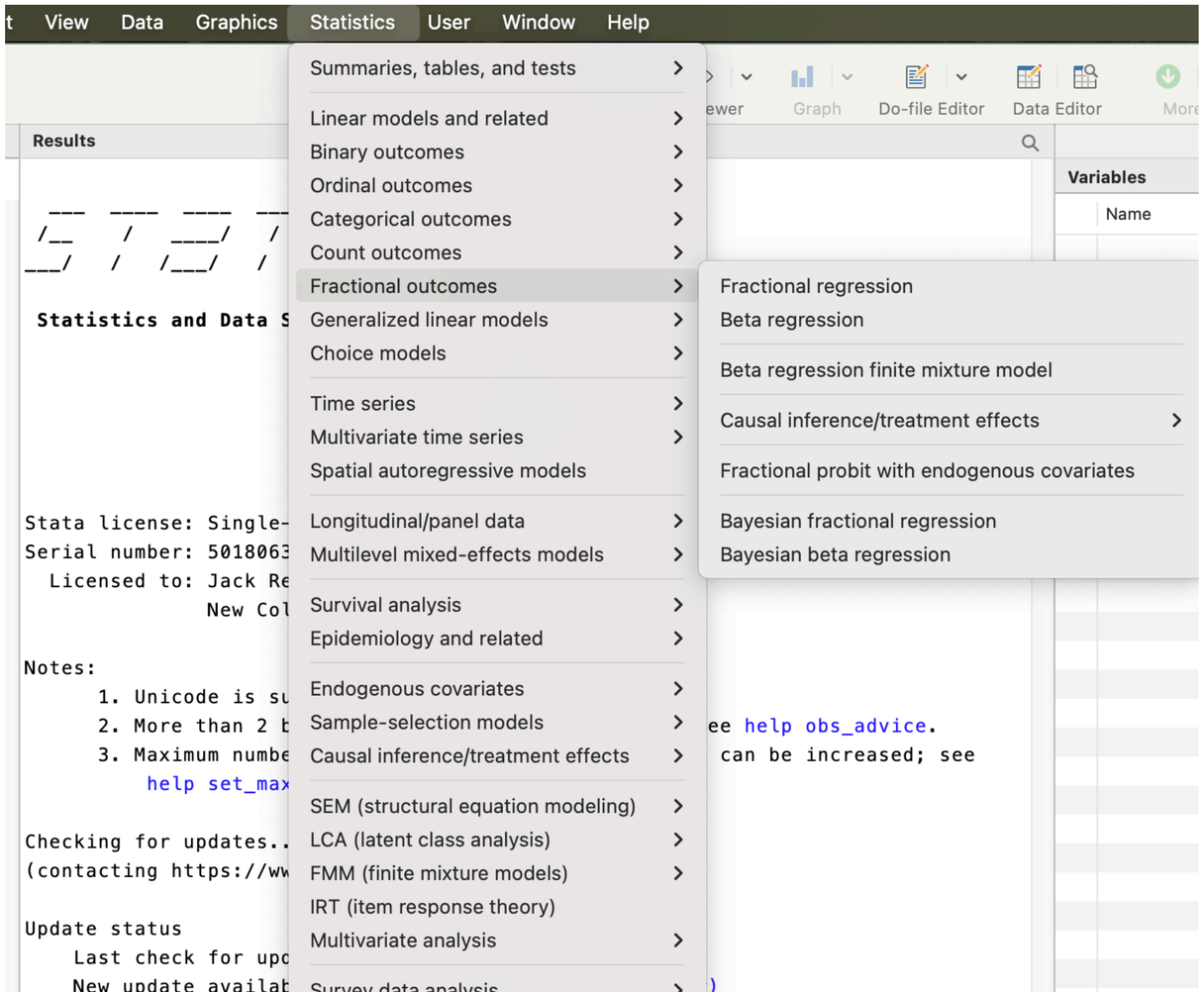
DOI: 10.1371/journal.pmed.0020124

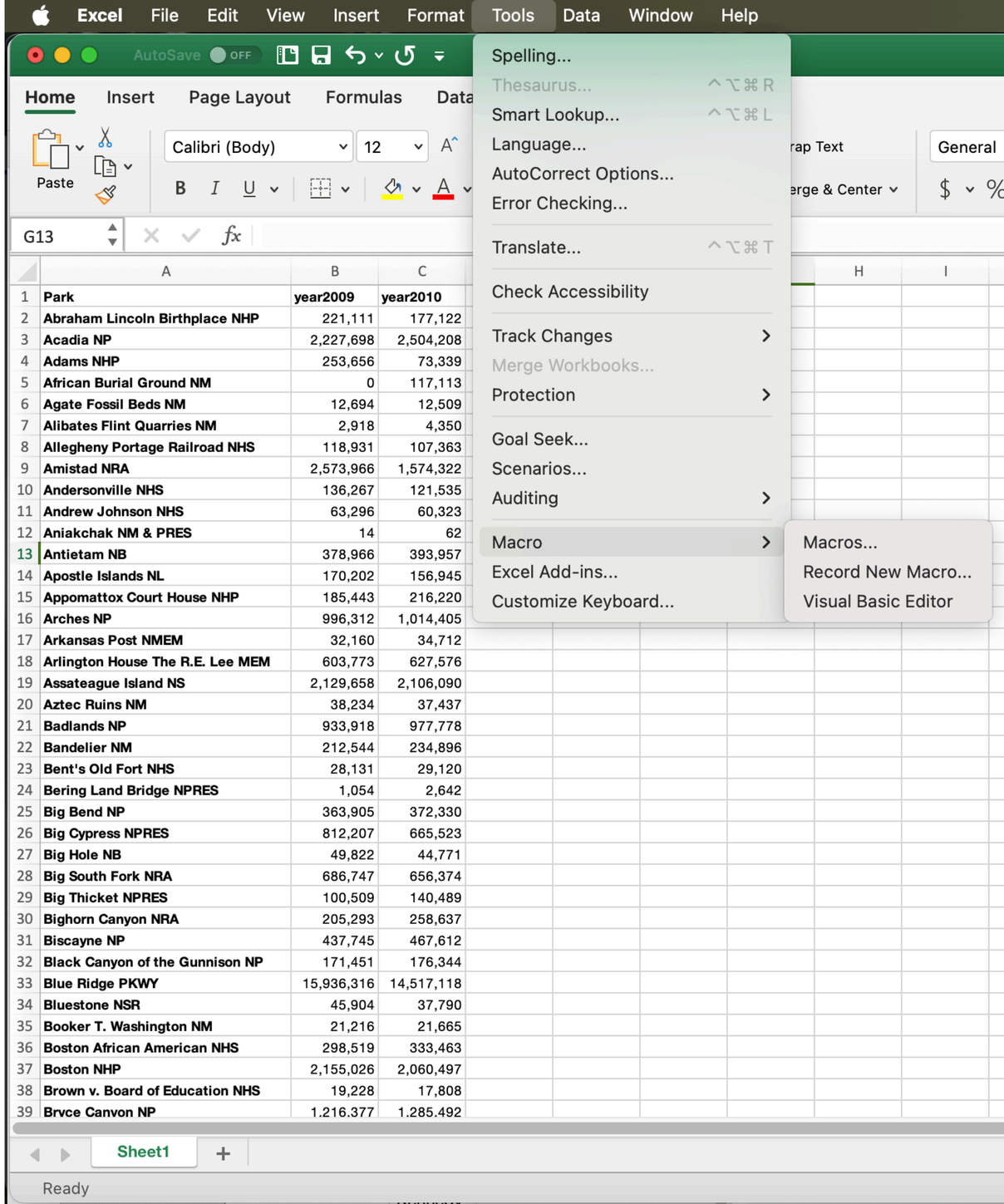
So what do we do?

- **Control:**
  - ensure we know *exactly* what is happening at all points in our study
- **Reproducibility:**
  - ensure that we (and others) can precisely reproduce all of our findings, from the same data, as easily as possible

The weaknesses of the office model

- It’s just easy to make mistakes





	A	B	C
1	Park	year2009	year2010
2	Abraham Lincoln Birthplace NHP	221,111	177,122
3	Acadia NP	2,227,698	2,504,208
4	Adams NHP	253,656	73,339
5	African Burial Ground NM	0	117,113
6	Agate Fossil Beds NM	12,694	12,509
7	Alibates Flint Quarries NM	2,918	4,350
8	Allegheny Portage Railroad NHS	118,931	107,363
9	Amistad NRA	2,573,966	1,574,322
10	Andersonville NHS	136,267	121,535
11	Andrew Johnson NHS	63,296	60,323
12	Aniakchak NM & PRES	14	62
13	Antietam NB	378,966	393,957
14	Apostle Islands NL	170,202	156,945
15	Appomattox Court House NHP	185,443	216,220
16	Arches NP	996,312	1,014,405
17	Arkansas Post NMEM	32,160	34,712
18	Arlington House The R.E. Lee MEM	603,773	627,576
19	Assateague Island NS	2,129,658	2,106,090
20	Aztec Ruins NM	38,234	37,437
21	Badlands NP	933,918	977,778
22	Bandelier NM	212,544	234,896
23	Bent's Old Fort NHS	28,131	29,120
24	Bering Land Bridge NPRES	1,054	2,642
25	Big Bend NP	363,905	372,330
26	Big Cypress NPRES	812,207	665,523
27	Big Hole NB	49,822	44,771
28	Big South Fork NRA	686,747	656,374
29	Big Thicket NPRES	100,509	140,489
30	Bighorn Canyon NRA	205,293	258,637
31	Biscayne NP	437,745	467,612
32	Black Canyon of the Gunnison NP	171,451	176,344
33	Blue Ridge PKWY	15,936,316	14,517,118
34	Bluestone NSR	45,904	37,790
35	Booker T. Washington NM	21,216	21,665
36	Boston African American NHS	298,519	333,463
37	Boston NHP	2,155,026	2,060,497
38	Brown v. Board of Education NHS	19,228	17,808
39	Brvce Canvon NP	1.216.377	1.285.492

we want to know the **exact code**

that produces the **exact result**

that we **exactly report**

also, we'd like to **keep records**
preferably **open**

This “engineering” approach

Is it perfect?

No.

Is it easy?

No.

Will I like it?

I hope so!

That “office” approach

Is it bad?

No.

Is it easy?

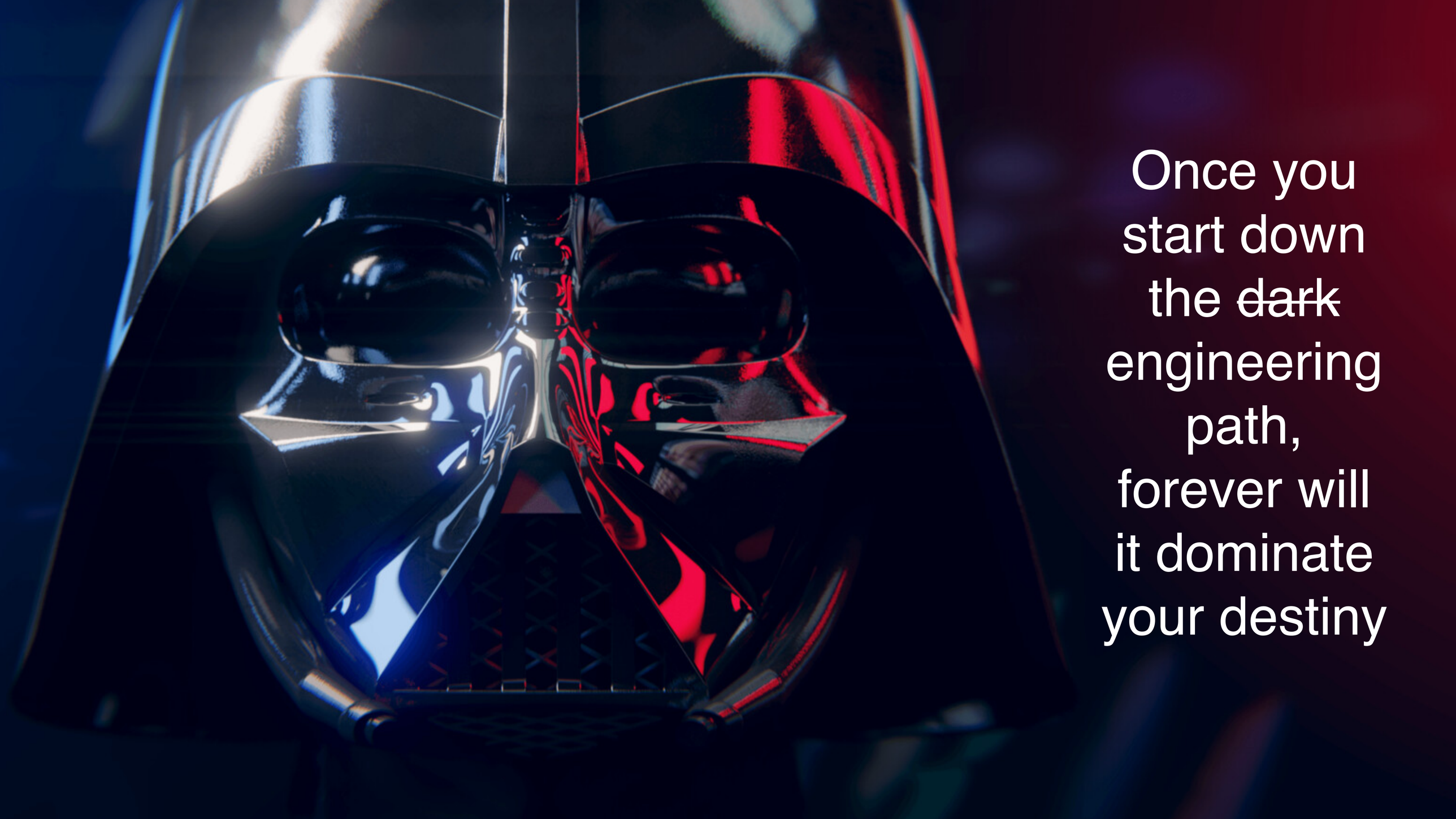
Kind of.

Can I go back?

Not entirely.

Strengths and weaknesses

- **Office** model
  - Documents look like documents
  - Everyone knows Word, Excel, Google docs, etc
  - “Track changes” is super useful in a single document
  - *Hm, I can’t remember how I made this figure.*
  - *Where did this table of results come from?*
  - `Paper_edits_FINAL_really_lastone-6.docx`
- **Engineering** model
  - Plain text is highly portable.
  - Push button, recreate analysis.
  - Can track changes across entire project
  - Tables and figures produced and integrated programatically.
  - For the love of God, why can’t I do this simple #(*&@#$ thing?
  - `Object of type 'closure' is not subsettable`


Once you
start down
the ~~dark~~
engineering
path,
forever will
it dominate
your destiny

Joking aside

- Each approach generates its own problems
  - **Engineering** grants **control** of code, **reproducibility,** and **advanced reporting options**
    - *at the expense of a learning curve and some continued frustration*
  - **Office** grants ease, speed, and nice GUI interfaces
    - *at the expense of control, reproducibility, and reporting options*

Principles | Tools

The to-do list

we want to know the **exact code**
that produces the **exact result**
that we **exactly report**

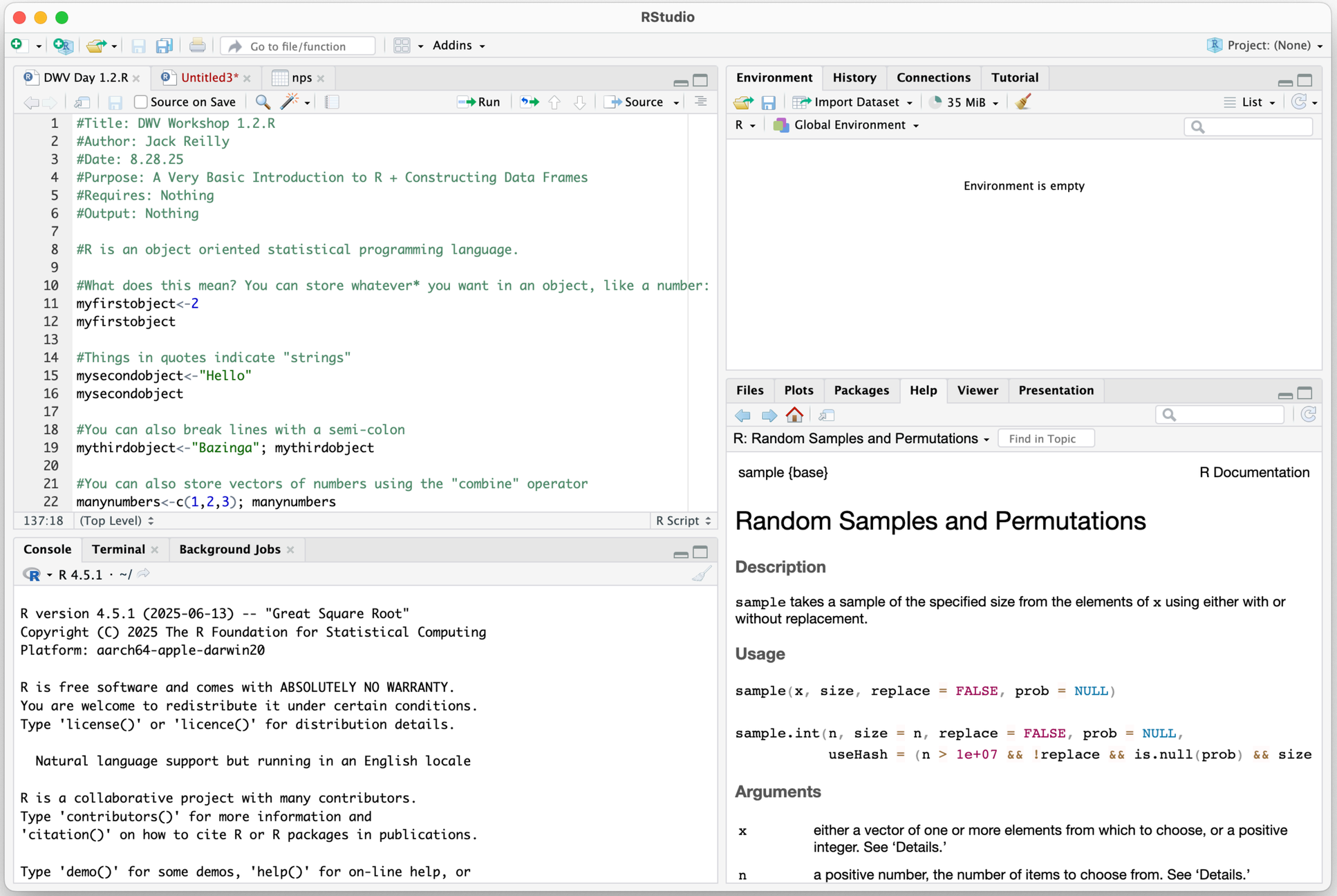
also, we'd like to **keep records**
preferably **open**

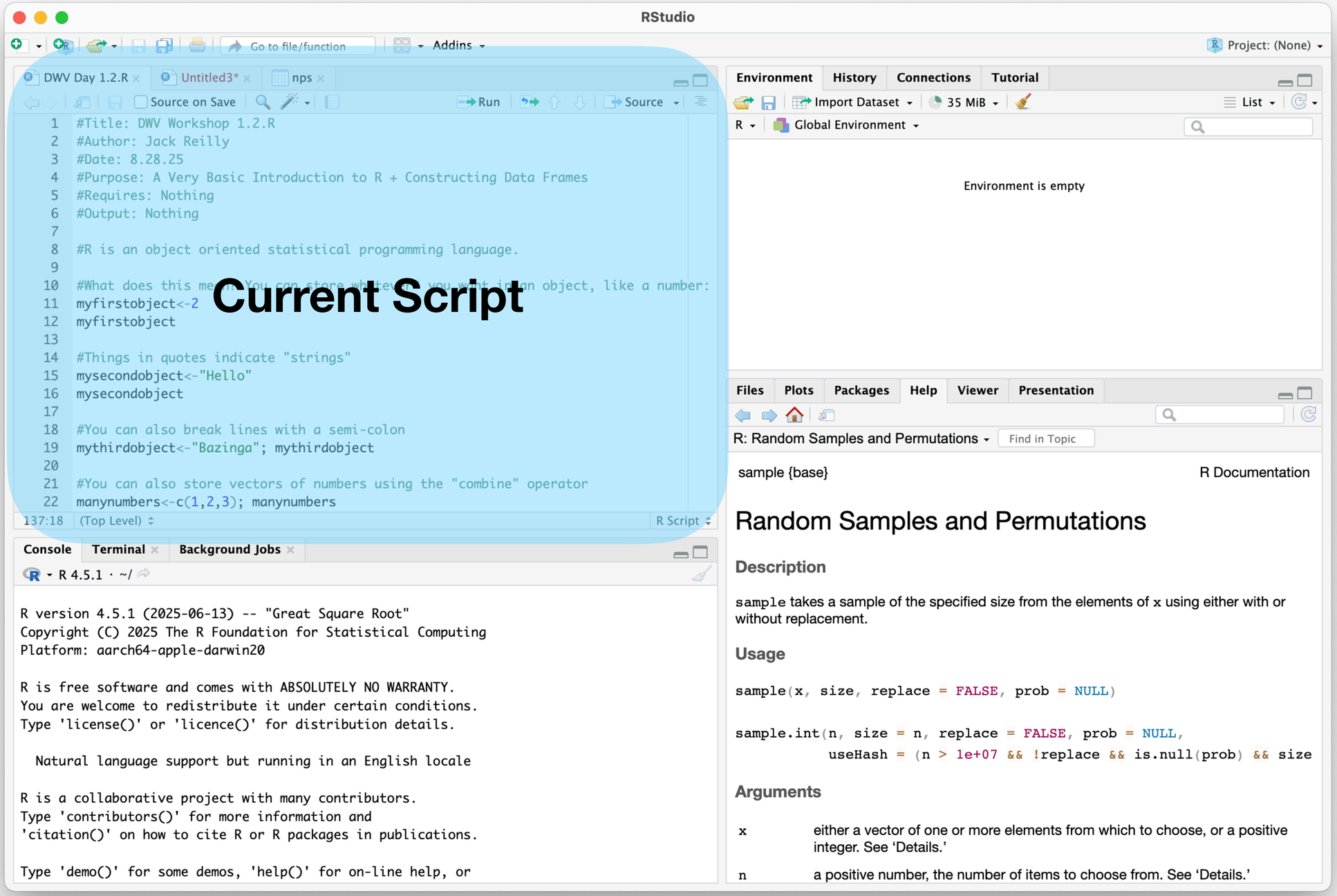
The Open Science Toolkit

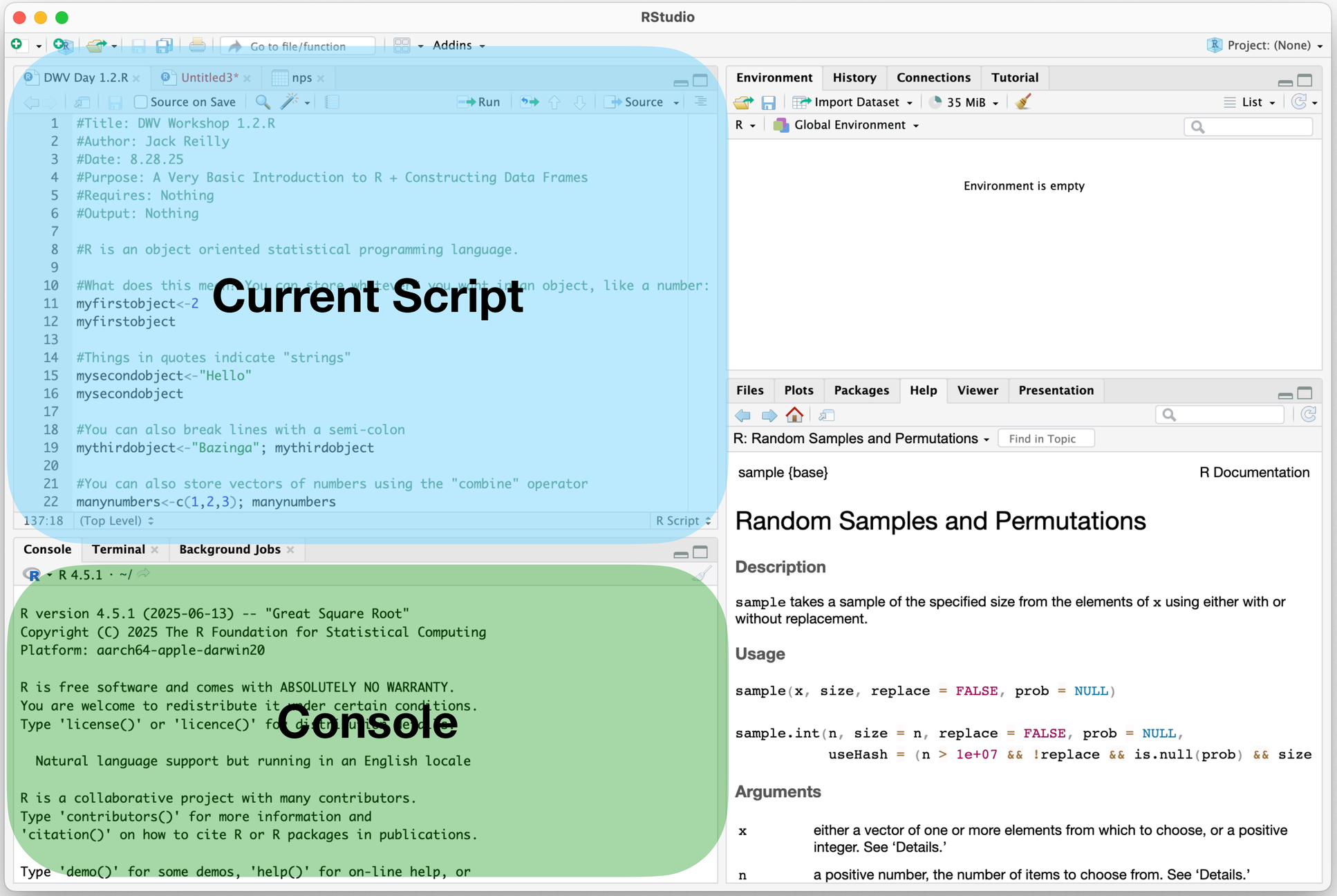
- So, first, we want to do our work **openly** and **reproducibly**
  - First, we write **in code**, rather than point and click
  - Second, we use open software (R, in our case) and save our *scripts*
  - In an IDE (integrated development environment) known as Rstudio

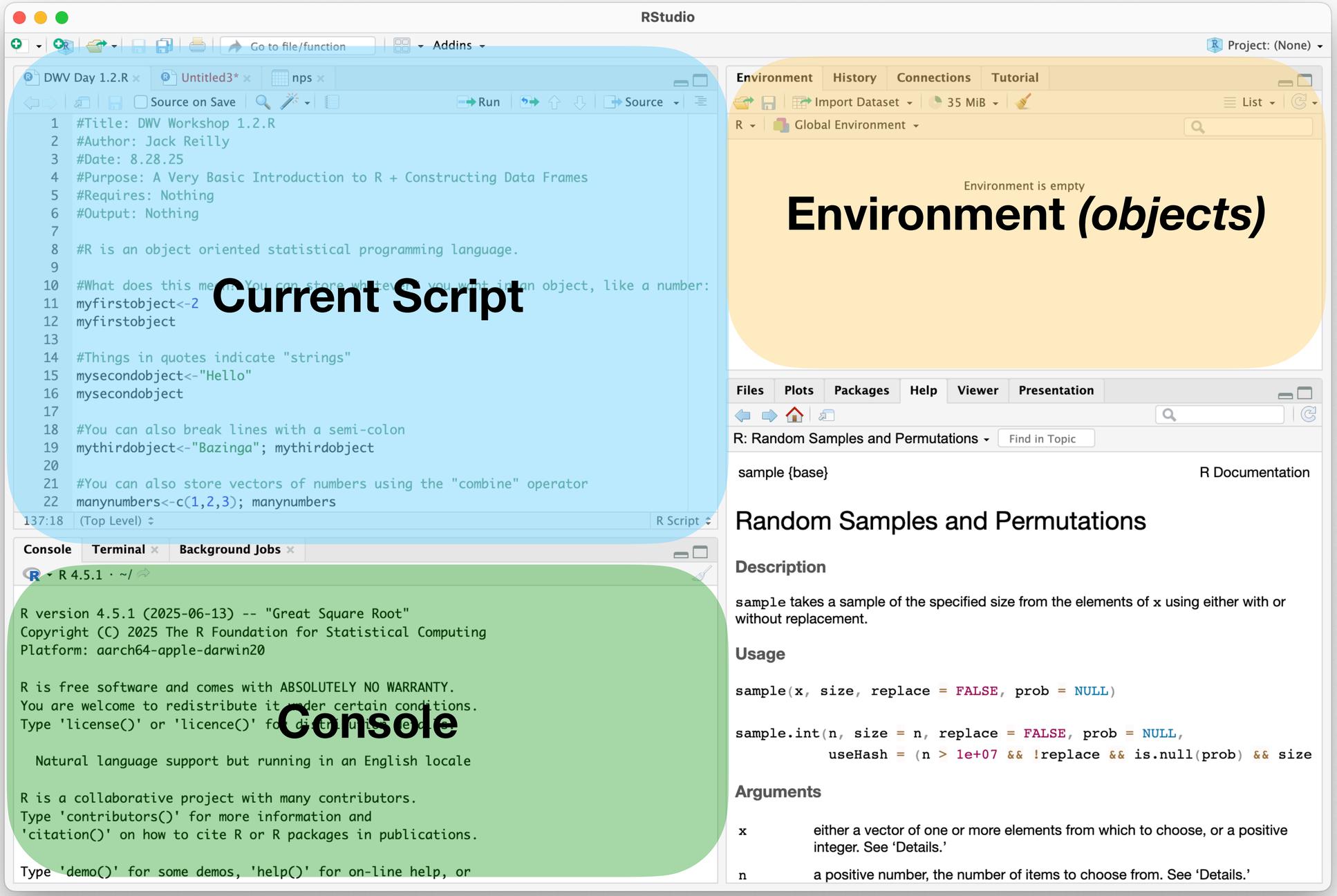
An IDE?

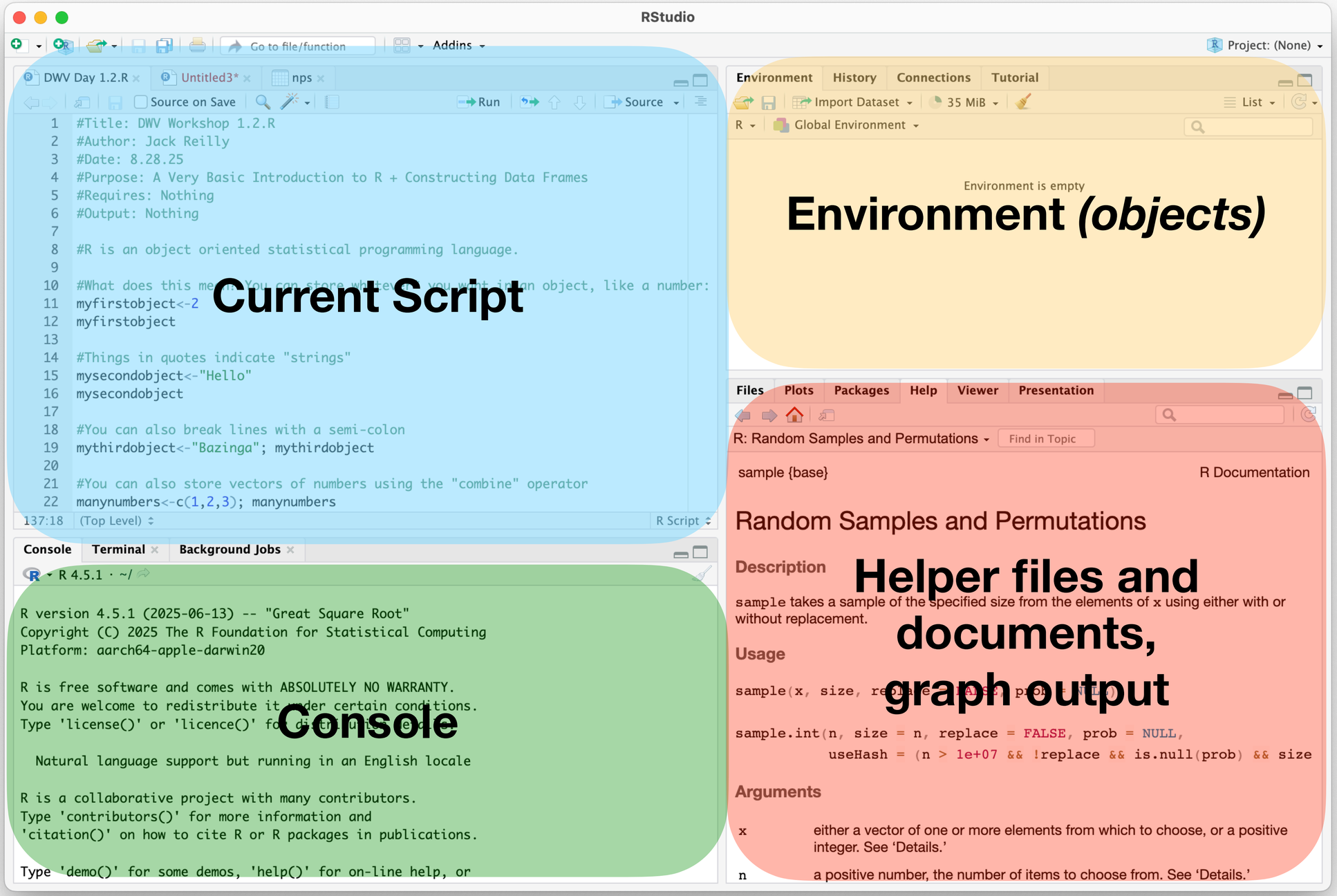
- *Integrated Development Environment*
  - Think of a kitchen
  - You’re the chef. The kitchen (IDE) has everything you need in it make the recipe (the script file) and produce the food (data analysis)


RStudio
Go to file/function
Addins
Project: (None)
DWV Day 1.2.R
Untitled3*
nps
Source on Save
Run
Source
#Title: DWV Workshop 1.2.R
#Author: Jack Reilly
#Date: 8.28.25
#Purpose: A Very Basic Introduction to R + Constructing Data Frames
#Requires: Nothing
#Output: Nothing
#R is an object oriented statistical programming language.
#What does this mean? You can store whatever* you want in an object, like a number:
myfirstobject<-2
myfirstobject
#Things in quotes indicate "strings"
mysecondobject<-"Hello"
mysecondobject
#You can also break lines with a semi-colon
mythirdobject<-"Bazinga"; mythirdobject
#You can also store vectors of numbers using the "combine" operator
manynumbers<-c(1,2,3); manynumbers
137:18
(Top Level)
R Script
Console
Terminal
Background Jobs
R 4.5.1 · ~/
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
Environment
History
Connections
Tutorial
Import Dataset
35 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
R: Random Samples and Permutations
Find in Topic
sample {base}
R Documentation
Random Samples and Permutations
Description
sample takes a sample of the specified size from the elements of x using either with or without replacement.
Usage
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
useHash = (n > 1e+07 && !replace && is.null(prob) && size
Arguments
x
either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
n
a positive number, the number of items to choose from. See 'Details.'


RStudio
Go to file/function
Addins
Project: (None)
DWV Day 1.2.R
Untitled3*
nps
Source on Save
Run
Source
#Title: DWV Workshop 1.2.R
#Author: Jack Reilly
#Date: 8.28.25
#Purpose: A Very Basic Introduction to R + Constructing Data Frames
#Requires: Nothing
#Output: Nothing
#R is an object oriented statistical programming language.
#What does this mean? You can store whatever you want in an object, like a number:
myfirstobject<-2
myfirstobject
#Things in quotes indicate "strings"
mysecondobject<-"Hello"
mysecondobject
#You can also break lines with a semi-colon
mythirdobject<-"Bazinga"; mythirdobject
#You can also store vectors of numbers using the "combine" operator
manynumbers<-c(1,2,3); manynumbers
137:18
(Top Level)
R Script
Current Script
Console
Terminal
Background Jobs
R 4.5.1 · ~/
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
Environment
History
Connections
Tutorial
Import Dataset
35 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
R: Random Samples and Permutations
Find in Topic
sample {base}
R Documentation
Random Samples and Permutations
Description
sample takes a sample of the specified size from the elements of x using either with or without replacement.
Usage
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
useHash = (n > 1e+07 && !replace && is.null(prob) && size
Arguments
x
either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
n
a positive number, the number of items to choose from. See 'Details.'


RStudio
Go to file/function
Addins
Project: (None)
DWV Day 1.2.R
Untitled3*
nps
Source on Save
Run
Source
#Title: DWV Workshop 1.2.R
#Author: Jack Reilly
#Date: 8.28.25
#Purpose: A Very Basic Introduction to R + Constructing Data Frames
#Requires: Nothing
#Output: Nothing
#R is an object oriented statistical programming language.
#What does this mean? You can store whatever you want in an object, like a number:
myfirstobject<-2
myfirstobject
#Things in quotes indicate "strings"
mysecondobject<-"Hello"
mysecondobject
#You can also break lines with a semi-colon
mythirdobject<-"Bazinga"; mythirdobject
#You can also store vectors of numbers using the "combine" operator
manynumbers<-c(1,2,3); manynumbers
137:18
(Top Level)
R Script
Current Script
Console
Terminal
Background Jobs
R 4.5.1 · ~/
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
Console
Environment
History
Connections
Tutorial
Import Dataset
35 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
R: Random Samples and Permutations
Find in Topic
sample {base}
R Documentation
Random Samples and Permutations
Description
sample takes a sample of the specified size from the elements of x using either with or without replacement.
Usage
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
useHash = (n > 1e+07 && !replace && is.null(prob) && size
Arguments
x
either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
n
a positive number, the number of items to choose from. See 'Details.'


RStudio
Go to file/function
Addins
Project: (None)
DWV Day 1.2.R
Untitled3*
nps
Source on Save
Run
Source
#Title: DWV Workshop 1.2.R
#Author: Jack Reilly
#Date: 8.28.25
#Purpose: A Very Basic Introduction to R + Constructing Data Frames
#Requires: Nothing
#Output: Nothing
#R is an object oriented statistical programming language.
#What does this mean? You can store whatever you want in an object, like a number:
myfirstobject<-2
myfirstobject
#Things in quotes indicate "strings"
mysecondobject<-"Hello"
mysecondobject
#You can also break lines with a semi-colon
mythirdobject<-"Bazinga"; mythirdobject
#You can also store vectors of numbers using the "combine" operator
manynumbers<-c(1,2,3); manynumbers
137:18
(Top Level)
R Script
Current Script
Console
Terminal
Background Jobs
R 4.5.1 · ~/
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
Console
Environment
History
Connections
Tutorial
Import Dataset
35 MiB
List
R
Global Environment
Environment is empty
Environment (objects)
Files
Plots
Packages
Help
Viewer
Presentation
R: Random Samples and Permutations
Find in Topic
sample {base}
R Documentation
Random Samples and Permutations
Description
sample takes a sample of the specified size from the elements of x using either with or without replacement.
Usage
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
useHash = (n > 1e+07 && !replace && is.null(prob) && size
Arguments
x
either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
n
a positive number, the number of items to choose from. See 'Details.'


RStudio
Go to file/function
Addins
Project: (None)
DWV Day 1.2.R
Untitled3*
nps
Source on Save
Run
Source
#Title: DWV Workshop 1.2.R
#Author: Jack Reilly
#Date: 8.28.25
#Purpose: A Very Basic Introduction to R + Constructing Data Frames
#Requires: Nothing
#Output: Nothing
#R is an object oriented statistical programming language.
#What does this mean? You can store whatever you want in an object, like a number:
myfirstobject<-2
myfirstobject
#Things in quotes indicate "strings"
mysecondobject<-"Hello"
mysecondobject
#You can also break lines with a semi-colon
mythirdobject<-"Bazinga"; mythirdobject
#You can also store vectors of numbers using the "combine" operator
manynumbers<-c(1,2,3); manynumbers
137:18
(Top Level)
R Script
Current Script
Console
Terminal
Background Jobs
R 4.5.1 · ~/
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
Console
Environment
History
Connections
Tutorial
Import Dataset
35 MiB
List
R
Global Environment
Environment is empty
Environment (objects)
Files
Plots
Packages
Help
Viewer
Presentation
R: Random Samples and Permutations
Find in Topic
sample {base}
R Documentation
Random Samples and Permutations
Description
sample takes a sample of the specified size from the elements of x using either with or without replacement.
Usage
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
useHash = (n > 1e+07 && !replace && is.null(prob) && size
Arguments
x
either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
n
a positive number, the number of items to choose from. See 'Details.'
Helper files and documents, graph output

Do I need to use RStudio? Or an IDE?

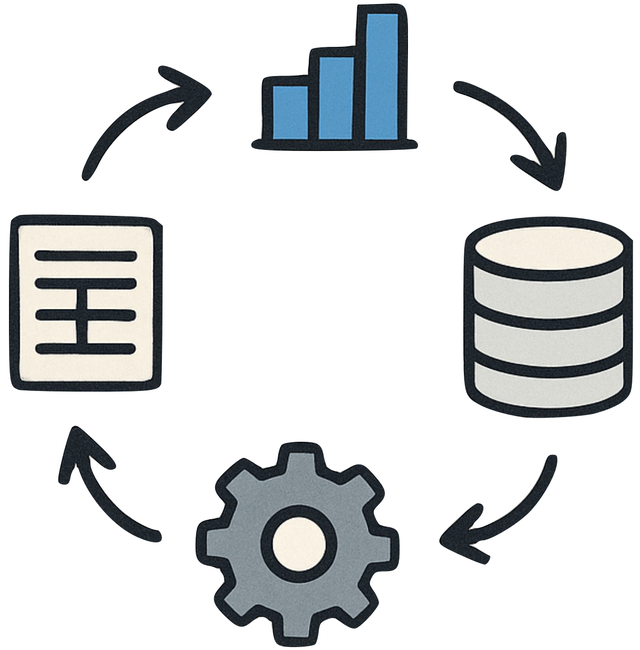
- No, actually!
  - A script file is just a plain text file. You can edit it anywhere. (*Although you **do** have to use a script file to make things reproducible*)
- There are other IDEs and text editors that others use
  - Visual Studio Code is getting very popular; Positron is kind of a next-generation RStudio; lots of people just write things in text editors like Sublime Text, TextMate, BBEdit, etc
- But it's handy to have everything together

RStudio does all of the following for you

- A **text editor** for writing code and documents
- A **console** or REPL (Read-Eval-Print Loop) for running code interactively
- A **terminal** to talk to the operating system
- A **debugger** to help find problems in your code
- A **file manager** to navigate your project
- A **version control interface** to manage changes to your code
- A **viewer** for plots, tables, and other outputs
- An **inspector** to see what's in your environment

Two (more) rough philosophies of computer usage

- The **omnipresent, monolithic** application approach
  - *“Do everything”*



- The **modular, interoperable** application approach
  - *“Do one thing, and do it well”*

Neither is perfect

- *When done well:*
  - One application for everything!
- *When done poorly:*
  - Jack of all trades, master of none

- *When done well:*
  - Symphony of parts.
- *When done poorly:*
  - Intractable coordination

Rstudio is kind of both! A monolithic wrapper around a bunch of smaller utilities

Two (more) rough philosophies of computer usage

- Most people lean towards the first
    - The second has some virtues, too
- Even if you kind of like the modular approach, odds are RStudio (or another IDE) has enough niceties you'll find yourself there pretty often for at least one or two of them
    - And, because the filetypes are all **open**, you can switch to other applications pretty easily

The to-do list

we want to know the **exact code**
that produces the **exact result**
that we **exactly report**

also, we'd like to **keep records**
preferably **open**

*we want to know the **exact code***
*that produces the **exact result***
*that we **exactly report***
*also, we'd like to **keep records***
*preferably **open***

The to-do list

- OK, so we have **code**, that hopefully produces **results**, and we're **open**.
  - **EXCELLENT.** Thanks R and RSTUDIO!
- What about **reporting** and **record-keeping**?
  - We have two important additional tools:
    - **Reporting**: Quarto (and Rmarkdown, and/or LaTeX)
    - **Record-keeping** (aka **version tracking**): git and GitHub

Reporting

- Wait, can't I just write this thing in Word? *What new devilry are you going to foist upon me?*
  - Well, you can.
  - And unless you work in a pretty modern, tech-forward environment, most people do.
  - The replacement of Excel with more powerful (and open) tools is more complete than the replacement of Word
- But once you've started down the open science path, you'll find aspects of Word to be . . . less than desirable

Word

- Do you want to manually copy/paste a figure or table into word every time you re-create it?
- Would you like better interoperability with different file formats?
- Would you like better control over where figures go, rather than them jumping all over pages?
- Declarative formatting, indexing, and the like?

Quarto and Rmarkdown: a notebook approach

- We're going to write an plain-text document that is interspersed with two things:
  - Analysis code *(writing for machines)*
  - Interpretation and explanation *(writing for humans)*
- Imagine we start like this
  - Script files have some nice features, but: we want it to look **pretty**. Namely:
    - No code! Prefer to see results
    - Comments ugly! Prefer nice looking prose

```
#Title: DWV Workshop 1.2.R
#Author: Jack Reilly
#Date: 8.28.25
#Purpose: A Very Basic
Introduction to R +
Constructing Data Frames
#Requires: Nothing
#Output: Nothing

#R is an object oriented
statistical programming
language.

#What does this mean? You
can store whatever* you
want in an object, like a
number:
myfirstobject<-2
myfirstobject

#Things in quotes indicate
"strings"
mysecondobject<-"Hello"
mysecondobject
```

````
---
title: "My First Quarto"
author: "Jack Reilly"
format: html
---

## Quarto

Quarto enables you to weave together content and
executable code into a finished document. To learn
more about Quarto see <https://quarto.org>.

## Running Code

When you click the **Render** button a document
will be generated that includes both content and
the output of embedded code. You can embed code
like this:

```{r}
1 + 1
```

You can add options to executable code like this

```{r}
#| echo: false
2 * 2
```

The `echo: false` option disables the printing of
code (only output is displayed).
````

My First Quarto

AUTHOR
Jack Reilly

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

You can add options to executable code like this

```
[1] 4
```

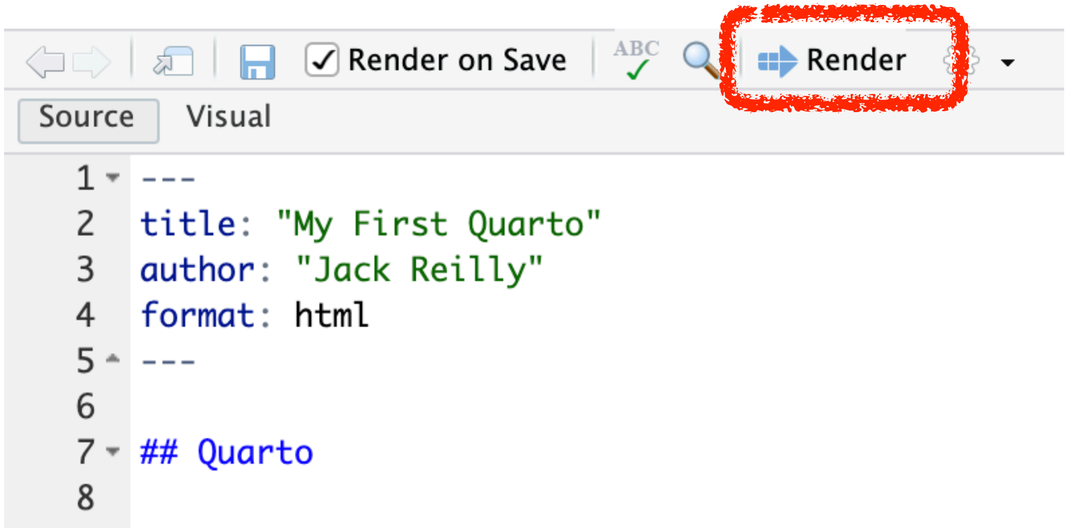
The `echo: false` option disables the printing of code (only output is displayed).

Ooo! Ooo! That's pretty!

- Yes, yes it is. And easy!
- How does it work?
  - Write in plain text with appropriate *markup* indicators
    - Including indicators to let the computer know when you're writing human language to communicate and when you're writing computer language to calculate.
  - **knit** or **render** the document together
    - **Run** all calculations and **typeset** the text, pretty-like

Rough outline

- File>>New File>>Quarto Document



- Declare format, and render it to html, pdf, or docx (or others)

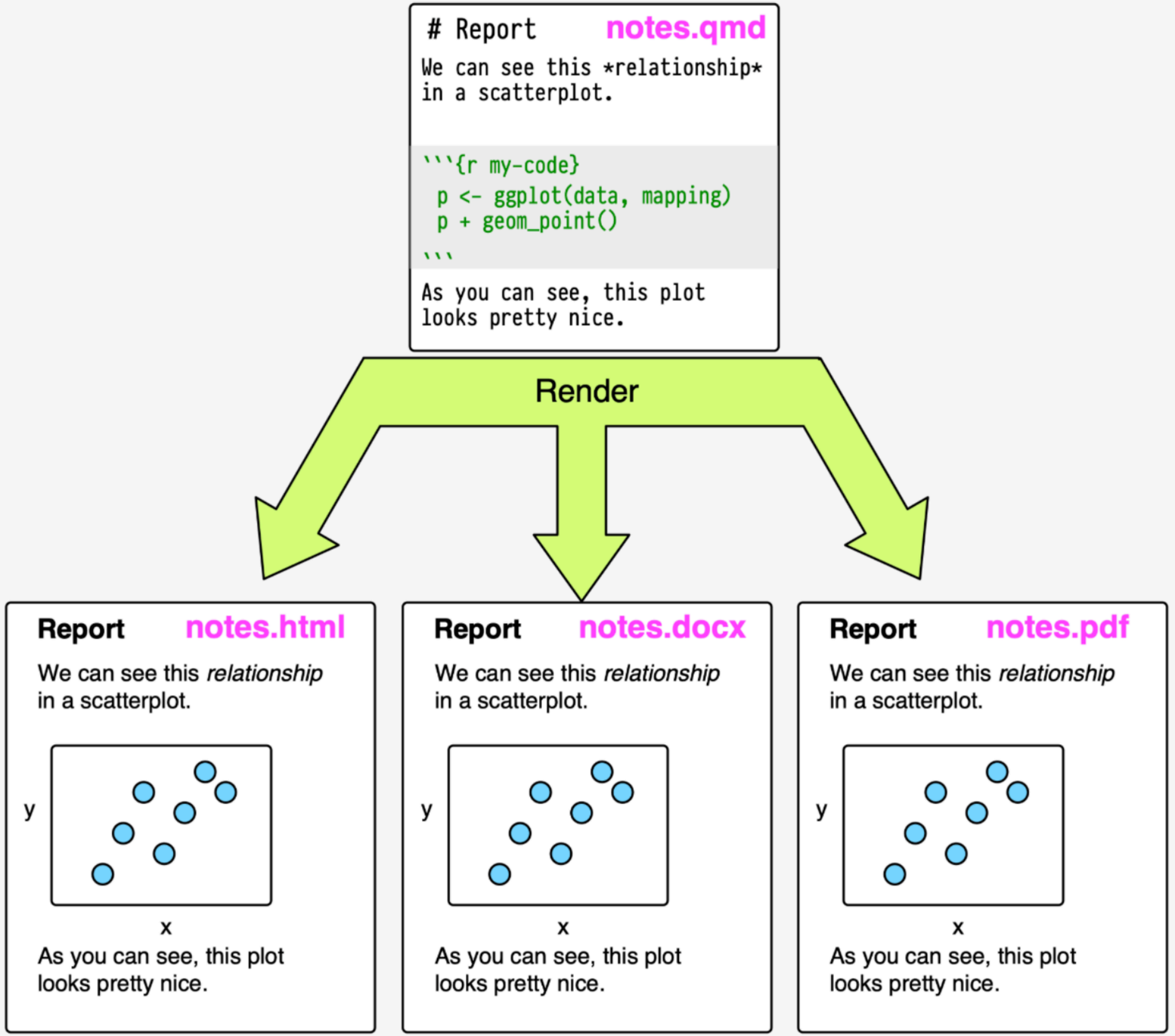


Image Credit: Kieran Healy

Rules of the road

- Basic markdown formatting (for text)
- Quarto *assumes* text unless you specifically declare code with ticks, like so:

```
```{r}
1 + 1
```
```

- If code, will just display result

Desired style	Use the following Markdown annotation
Heading 1	`# Heading 1`
Heading 2	`## Heading 2`
Heading 3	`### Heading 3` (Actual heading styles will vary.)
Paragraph	Just start typing
Bold	`**Bold**`
Italic	`*Italic*`
Images	`[Alternate text for image](path/image.jpg)`
Hyperlinks	`[Link text](https://www.visualizingsociety.com/)`
Unordered Lists	
- First	`- First`
- Second.	`- Second`
- Third	`- Third`
Ordered Lists	
1. First	`1. First`
2. Second.	`2. Second`
3. Third	`3. Third`
Footnote.[1]	`Footnote[^notelabel]`
[1]The note's content.	`[^notelabel] The note's content.`

Image Credit: Kieran Healy

In practice

- Notebooks work **smoothly** when
  - Your document or report is small and self-contained.
  - Your analysis is quick or lightweight.
  - You are making slides.
  - You are making a lot of similar reports from a template.
  - You regularly refer to calculated items in the text of your analysis.
- Notebooks can get **awkward** when
  - Your analysis has many pieces.
  - Your project has many authors.
  - Your analysis needs a lot of cleaning, scaffolding, and other prep-work before you can produce the tables and figures in your output document.
  - You have a lot of different outputs

Monolith vs modular

- Quarto is more of a monolith approach
- Naturally suited to small projects
- Substantial advantages for large, as well, but complexity rises
  - The great thing about plain text files is that you can always ***refactor*** them (split up functionality/purpose across files) to make them ***more modular***
  - *Course website, for instance, is written in quarto files!*
- Other options, as well: LaTeX, etc

Last, but not least

- Our final major tool: **Record Keeping (version tracking)**

Last, but not least

- **Record Keeping (version tracking)**
  - Next week

Keeping the right frame of mind

- This is like learning how to drive a car, or how to cook in a kitchen … or learning to speak a language.
- After some orientation to what's where, you will learn best by doing.
- Software is a pain . . .
  - but at least you won't crash the car or burn your house down.

For now

- Time (hopefully) to troubleshoot any R problems you're having
- Workshop this week: getting used to Quarto and Markdown files