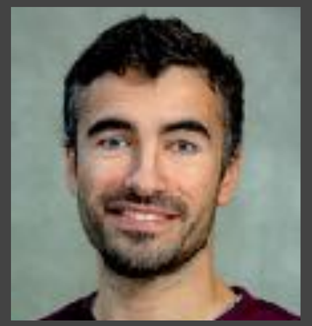
3. Scientific Guide for Reliable Energy Experiments

Sustainable Software Engineering
CS4575


Luís Cruz
L.Cruz@tudelft.nl


Carolin Brandt
C.E.Brandt@tudelft.nl


Enrique Barba Roque
E.BarbaRoque@tudelft.nl

1. Scientific guide for energy measurements
2. Energy consumption data analysis

Energy tests are **flaky** ?

- Multiple runs might yield different results
- There are many **confounding factors** that need to be ~~controlled~~/**minimized**.

Zen mode 🧘🏾‍♀️

- **Close all applications.**
- **Turn off notifications.**
- **Only the required hardware** should be connected (avoid USB drives, external disks, external displays, etc.).
- **Kill unnecessary services** running in the background (e.g., web server, file sharing, etc.).
- If you do not need an internet or intranet connection, **switch off your network.**
- Prefer **cable over wireless** – the energy consumption from a cable connection is more stable than from a wireless connection.

Freeze and report your settings 🥶

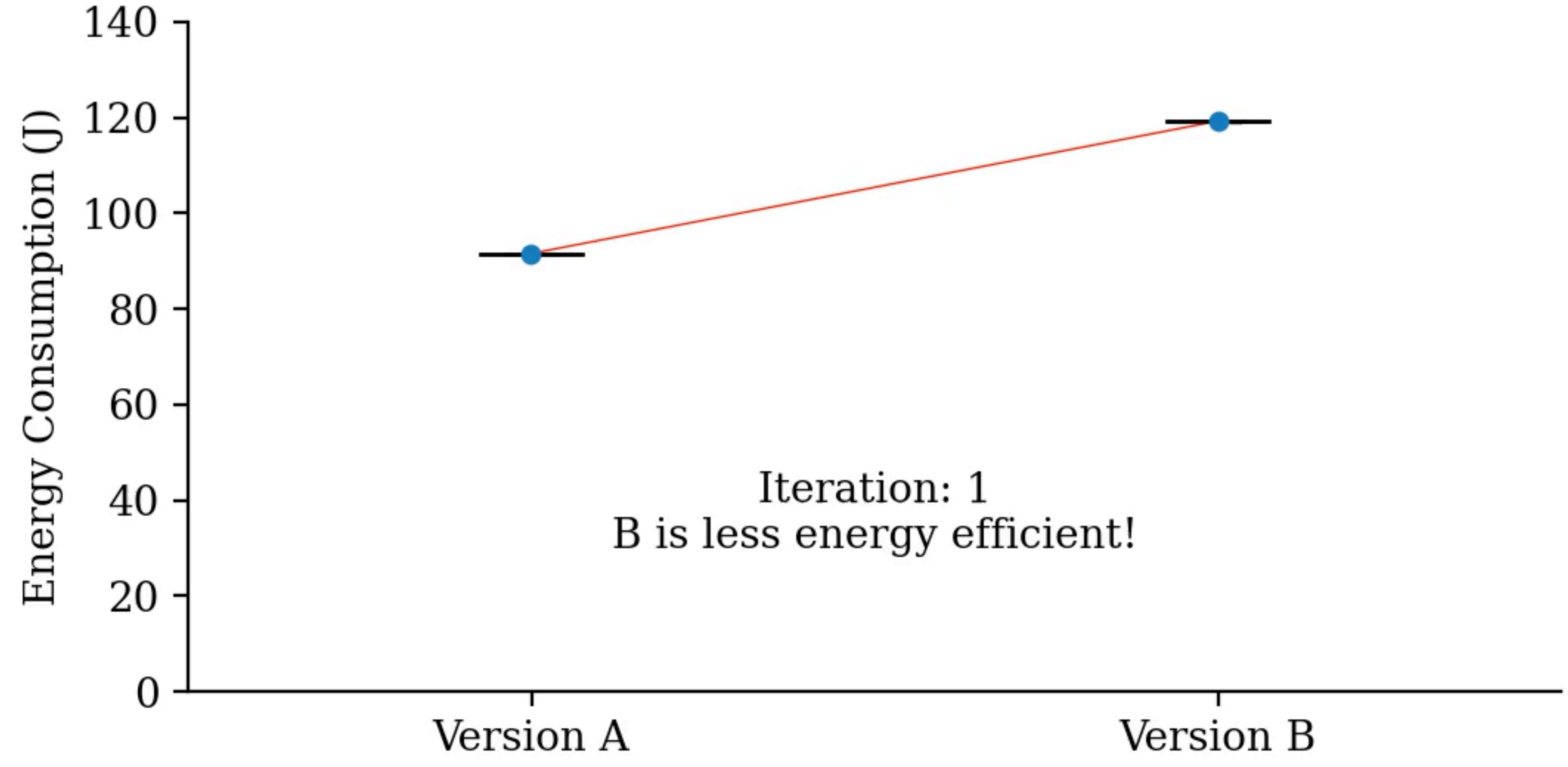
- Fix display brightness; **switch off auto brightness**
- If Wifi is on, it should always be on, connected to the same network/endpoint….

Warm-up 📶

- Energy consumption is highly affected by the **temperature of your hardware**.
- **Higher the temperature** -> higher the resistance of electrical conductors -> -> higher dissipation -> **higher energy consumption**
- The first execution will appear more efficient because the hardware is still cold.
- Run a **CPU-intensive task** before measuring energy consumption. E.g., Fibonacci sequence. At least 1min; 5min recommended.
  -

Repeat 🔁

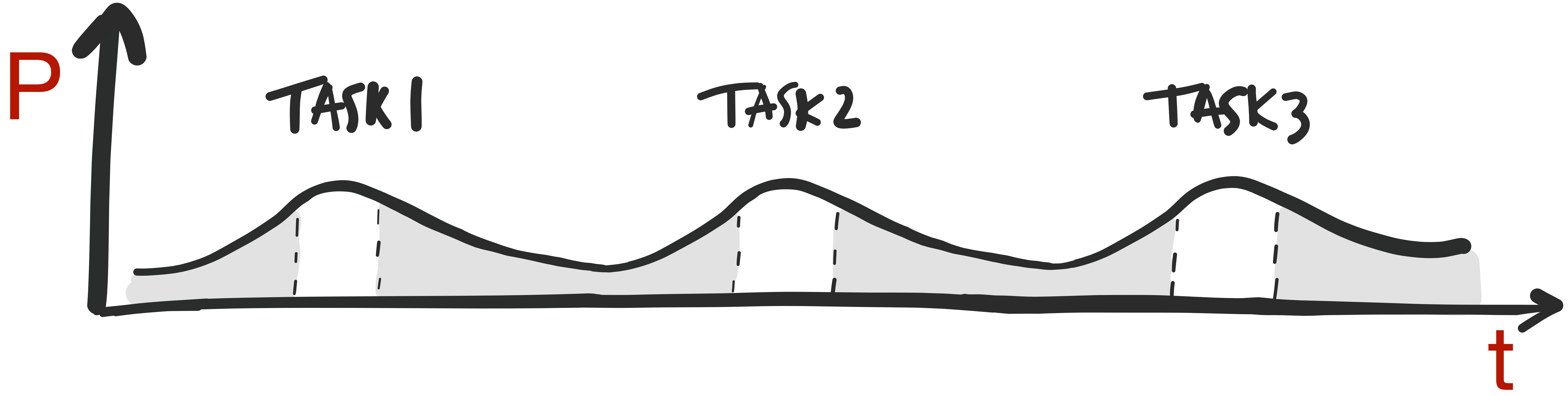
- The best way to make sure a measurement is **valid** is by **repeating** it.
- In a scientific project, the **magic number is 30.**


Energy Consumption (J)
140
120
100
80
60
40
20
0
Iteration: 1
B is less energy efficient!
Version A
Version B

Rest ⏸

- It is **common practice** to do a pause/sleep between executions/ measurements.
- Prevent **tail energy consumption** from previous measurements. ?
- Prevent collateral tasks of previous measurement from affecting the next measurement.
- There is no golden rule but **one minute** should be enough. It can be more or less depending on your **hardware** or the **duration** of your energy test.

Tail Energy Consumption



Shuffle 🔀

- It is not a mystery that energy consumption depends on so many factors that it is impossible to control all of them.
- If you run 30 executions for version A and another batch for version B:
  - **External conditions that change over time** will have a **different bias** in the 2 versions (e.g., room temperature changes).
  - If you shuffle, you reduce this risk.

Keep it cool 🌡️

- Always make sure there is a **stable room temperature**.
- Tricky because, some times, experiments may have to run over a few days.
- If you cannot control room temperature: **collect temperature data** and **filter out** measurements where the room temperature is clearly deviating.

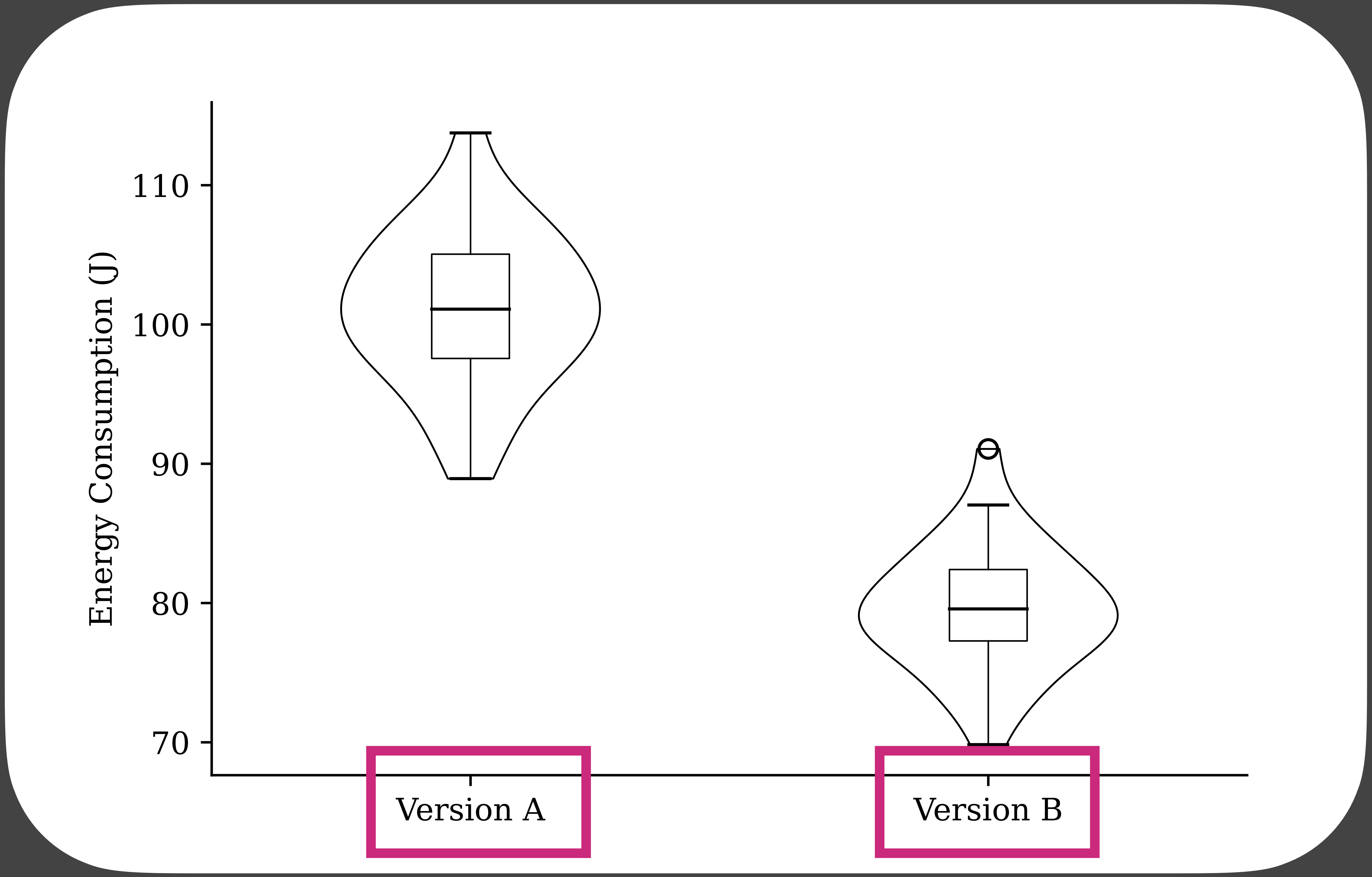
Automate Executions 🤖

- (Already mentioned in the previous classes)
- One cannot run 30 shuffled experiments per version without automation…

Data analysis

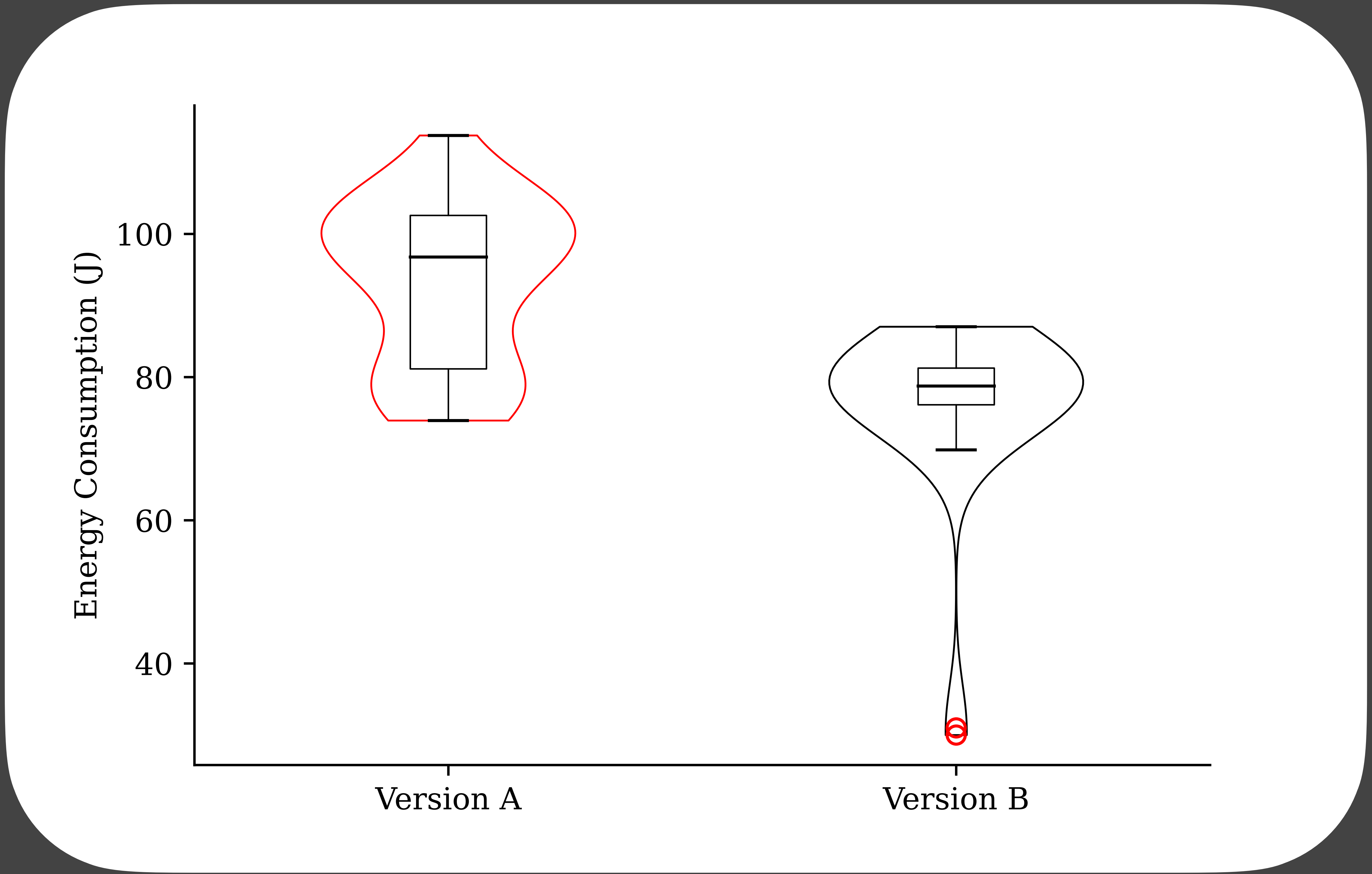
1. Exploratory Analysis

- Plot the data and inspect outliers or unexpected biases.
- **Violin+box plots** are usually handy. (?)
    - It's a nice way of combining the 30 experiments, and of showing descriptive statistics. (?)
    - Shows the **shape of the distribution** of the data.



1. Exploratory Analysis (II)

- Data should be **Normal**. Unless there's a good reason.
- E.g., somewhere amongst the 30 executions, there might be 1 or 2 that failed to finish due to some unexpected error.
    - (It happens and that's ok!)– consequently, the execution is shorter and spends less energy – **falsely appearing as more energy efficient**.
- If data is not Normal there might be some issues affecting the measurements that might be ruining results. It is important to investigate this.



Energy data is not normal. Why?

- It might be caused by one of the following reasons:
  - **There was an error** in some of the executions. If not detected and fixed it might ruin results.
  - Your tests are **not fully replicable** or are **not deterministic**. Quite frequent when you have **internet requests** or **random-based algorithms**.
  - There is an **unusual task** being run by the system during some experiments.
  - The computer entered a **different power mode**.
  - External physical conditions have changed. E.g., someone opened a window.

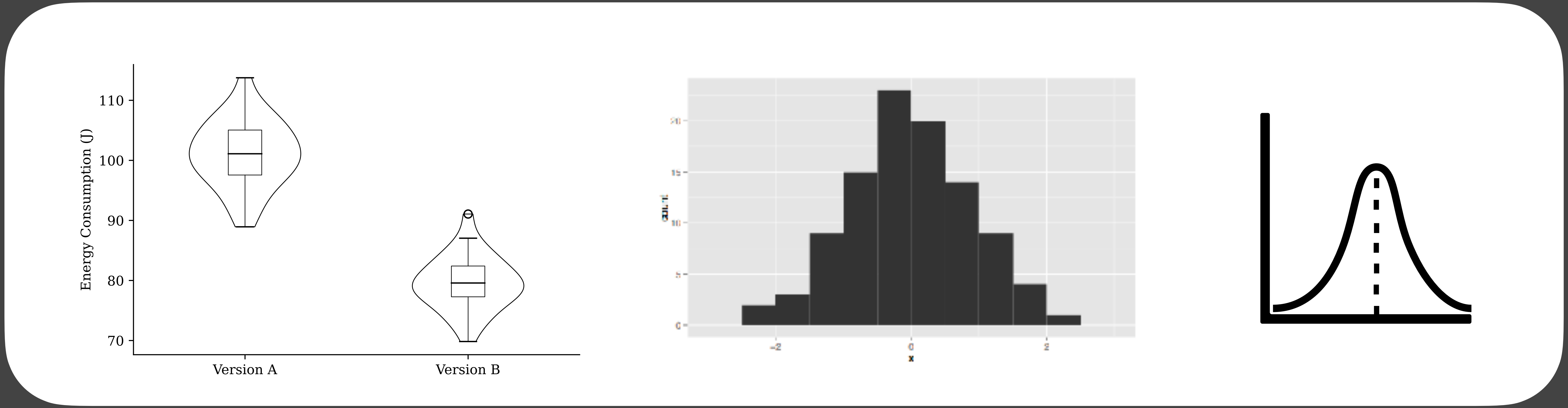
Energy data is not normal. How to fix?

- We have **2**+1 options:

1. **Remove outliers**. If there are only a few points that deviate from the normal distribution, it is okay to simply remove them.
    - Use the **z-score outlier removal**. (?)
        - **Remove** all data points that **deviate** from the **mean** more than **3 standard deviations**: $|\bar{x} - x| > 3s$
2. **Fix the issue** and **rerun** experiments.
3. Conclude that **nothing can be done about it** and data will never be normal. (e.g., in AI, executions are rarely deterministic). ⚠️ Only after ruling out the previous points.

How do we know whether data is Normal?

- Visualise distribution shape: **violin plots**, **histograms**, **density plot**.



- **Shapiro-Wilk test**.
  p-value $< 0.05 \Rightarrow$ **data is not normal**;
  p-value $\geq 0.05 \Rightarrow$ we are not sure but it is okay to assume that **data is normal**.

⚠️ assuming normal distribution

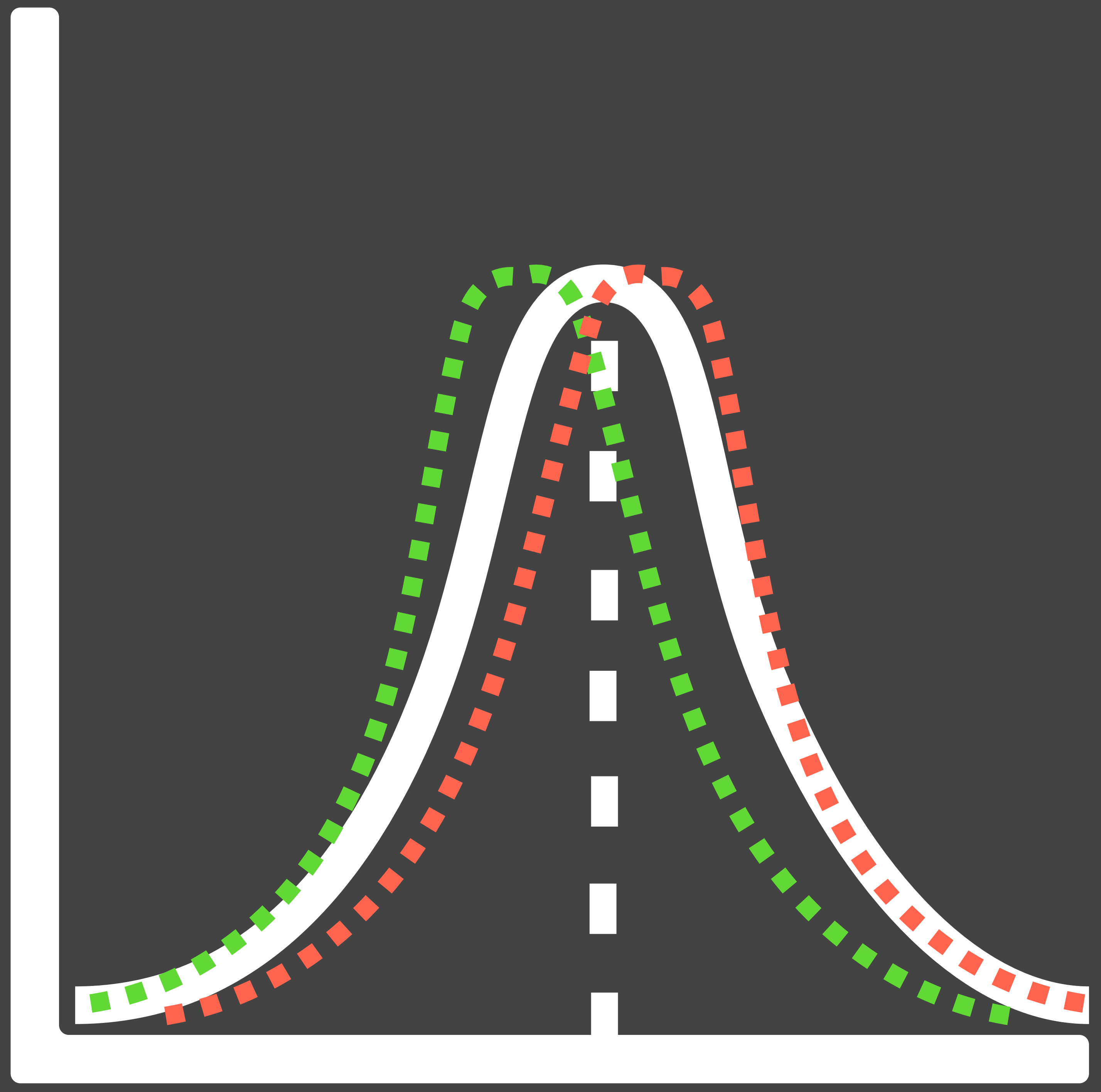
After having all data ready, which artefact is more energy efficient(?)

First approach: compare sample **means**.

⚠️ assuming normal distribution

Statistical significance

- Even if, on average, one artefact has lower energy consumption than other, it might be just a random difference.
- When we extract a sample from a normal distribution it will never be the exact same
- **Statistical significance** tests help you understand the differences in the average are conclusive/significant or inconclusive/insignificant.



⚠️ assuming normal distribution

Statistical significance test

- Two-sided (?) parametric test **Welch's t-test**.
  - Less known alternative to **student's t-test**.

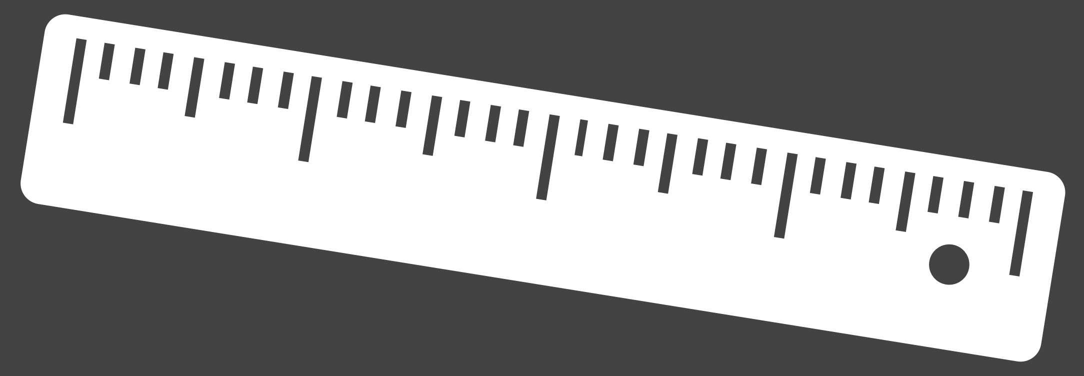
Welch's t-test in Python

```
from scipy.stats import ttest_ind

_,pvalue = ttest_ind(sample_a, sample_b,
                     equal_var=False,
                     alternative='two-sided')
```

⚠️ assuming normal distribution

Effect Size analysis



- Now that we know that results are statistically significant we need to **measure the difference** between the two samples.
  - **Mean difference**: $\Delta \bar{x}$
  - **Percent change**: $\dfrac{x_B - x_A}{x_A} \times 100\,\% = \dfrac{\Delta \bar{x}}{x_A} \times 100\,\%$
  - **Cohen's d** (informal definition: mean difference normalized by a **combined standard deviation**): $\dfrac{\Delta \bar{x}}{\frac{1}{2}\sqrt{s_1^2 + s_2^2}}$

(?)

Imagine that version **A** spends **70J** and version **B** spends **69J** with a p-**value** $= 0.04$.

On average, version **B** spent **less energy than** version **A** — ✅

There is statistical significance — ✅

Effect size, percent-change is ≈1% — 🤔

⚠️ **Do we care?**

Practical Significance

- Depending on the case, a 2% improvement might be either **wonderful** or **completely useless**.
  - Effect size analyses help assess practical significance but **might not be enough**.
  - A critical discussion always needs to be performed. **Consider context** and explain in what sense the effect size might be **relevant**.
    - E.g.:
      - to improve 2% in energy efficiency the code will be less readable or the user experience is not so appealing.
      - A particular method improves 2% but it will only be used 1% of the time.
- **There is no particular metric or structure**, but this kind of critical analysis is very important.

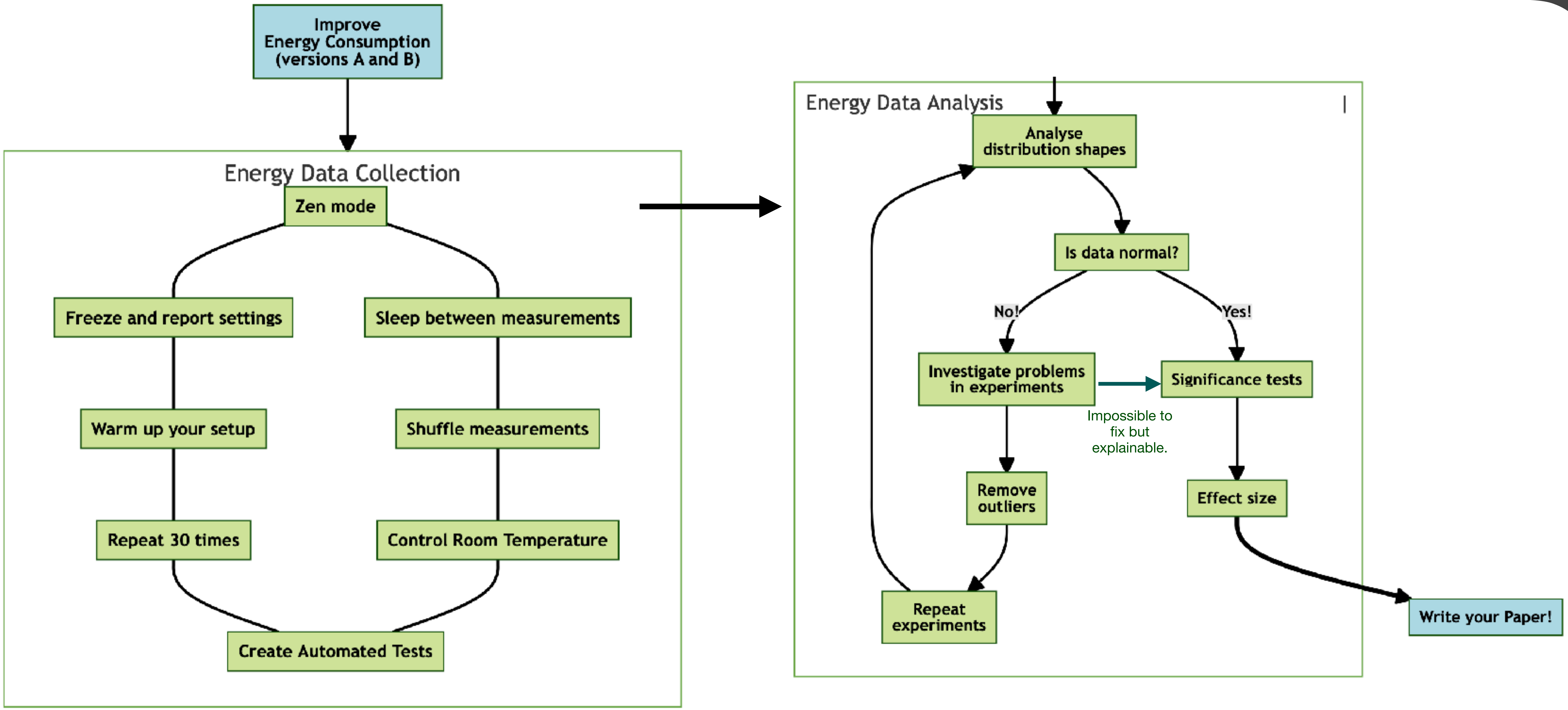
What if data is **not Normal**?

Same approach but different tests/metrics!

Non-normal data

- Statistical significance: **non-parametric** test (?)
  - **Mann-Whitney *U*** test. Instead of looking at standard deviation or mean, it orders samples and compares with each other.
  - **Less power** than parametric-tests (?)
- Effect size
  - Median difference: ΔM
  - **Percentage of pairs** supporting a conclusion (i.e., # pairs where version A > version B/ total pairs)
  - **Common language effect size**: $\dfrac{U_1}{N_1 N_2}$

Recap



Improve
Energy Consumption
(versions A and B)
Energy Data Collection
Zen mode
Freeze and report settings
Sleep between measurements
Warm up your setup
Shuffle measurements
Repeat 30 times
Control Room Temperature
Create Automated Tests
Energy Data Analysis
Analyse
distribution shapes
Is data normal?
No!
Yes!
Investigate problems
in experiments
Significance tests
Impossible to
fix but
explainable.
Remove
outliers
Effect size
Repeat
experiments
Write your Paper!

Energy Efficiency Across Programming Languages

Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Abstract

This paper presents a study of the runtime, memory usage and energy consumption of twenty seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as, slower/faster languages consuming less/more energy, and how memory usage influences energy consumption. We show how to use our results to provide software engineers support to decide which language to use when energy efficiency is a concern.

CCS Concepts • **Software and its engineering → Software performance; General programming languages;**

Keywords Energy Efficiency, Programming Languages, Language Benchmarking, Green Software

ACM Reference Format:
Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?. In *Proceedings of 2017 ACM SIGPLAN International Conference on Software Language Engineering (SLE'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3136014.3136031

1 Introduction

Software language engineering provides powerful techniques and tools to design, implement and evolve software languages. Such techniques aim at improving programmers productivity - by incorporating advanced features in the language design, like for instance powerful modular and type systems - and at efficiently execute such software - by developing, for example, aggressive compiler optimizations. Indeed, most techniques were developed with the main goal of helping software developers in producing faster programs. In fact, in the last century *performance* in software languages was in almost all cases synonymous of *fast execution time* (embedded systems were probably the single exception).

In this century, this reality is quickly changing and software energy consumption is becoming a key concern for computer manufacturers, software language engineers, programmers, and even regular computer users. Nowadays, it is usual to see mobile phone users (which are powerful computers) avoiding using CPU intensive applications just to save battery/energy. While the concern on the computers' energy efficiency started by the hardware manufacturers, it quickly became a concern for software developers too [28]. In fact, this is a recent and intensive area of research where several techniques to analyze and optimize the energy consumption of software systems are being developed. Such techniques already provide knowledge on the energy efficiency of data structures [15, 27] and android language [25], the energy impact of different programming practices both in mobile [18, 22, 31] and desktop applications [26, 32], the energy efficiency of applications within the same scope [2, 17], or even on how to predict energy consumption in several software systems [4, 14], among several other works.

An interesting question that frequently arises in the software energy efficiency area is whether *a faster program is also an energy efficient program*, or not. If the answer is yes, then optimizing a program for speed also means optimizing it for energy, and this is exactly what the compiler construction community has been hardly doing since the very beginning of software languages. However, energy consumption does not depends only on execution time, as shown in the equation $E_{nergy} = T_{ime} \times P_{ower}$. In fact, there are several research works showing different results regarding

SLE'17, October 23–24, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5525-4/17/10...$15.00
https://doi.org/10.1145/3136014.3136031

https://sites.google.com/view/energy-efficiency-languages

- Is a faster programming language also more energy efficient?
- Comparing different programming languages is not an easy task.
  - They differ in many mechanisms:
    - Interpreted vs Compiled
    - Optimisations at the compiler level
    - Virtual machine
    - Garbage collector
    - Available libraries

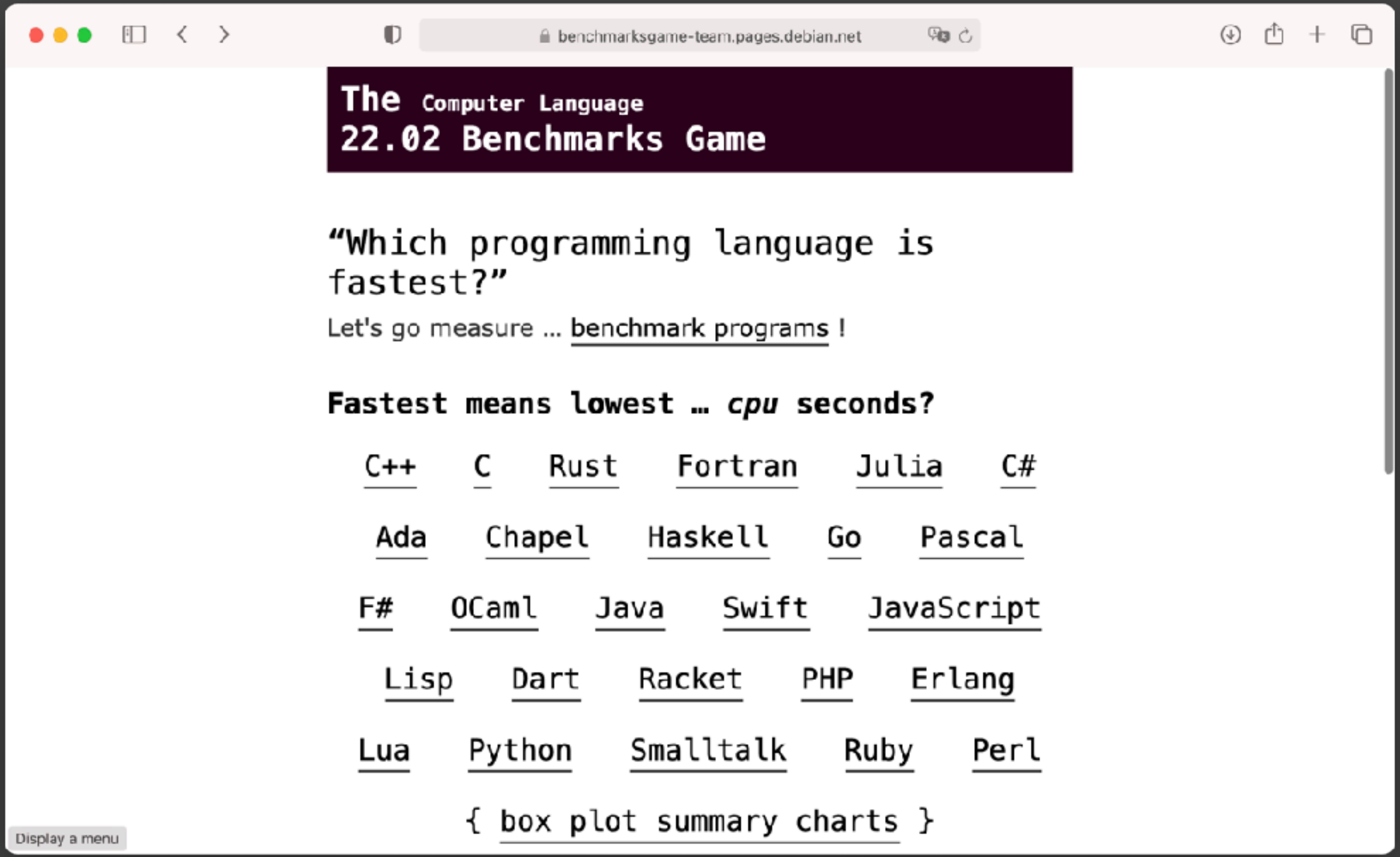
Research Questions

- **Can we compare** the energy efficiency of software languages?
- Is the **faster** language always the most **energy efficient**?
- ~~How does **memory usage** relate to **energy consumption**? (We don’t cover this one)~~
- Can we automatically decide what is the best programming language considering **energy**, **time**, and **memory usage**?
- How do the results of our energy consumption analysis of programming languages gathered from rigorous **performance benchmarking solutions compare to** results of **average day-to-day solutions**?

Methodology

The Computer Language Benchmarks Game

- https://benchmarksgame-team.pages.debian.net/benchmarksgame/



Problems in the Computer Language Benchmarks Game

Benchmark	Description	Input
`n-body`	Double precision N-body simulation	50M
`fannkuch-redux`	Indexed access to tiny integer sequence	12
`spectral-norm`	Eigenvalue using the power method	5,500
`mandelbrot`	Generate Mandelbrot set portable bitmap file	16,000
`pidigits`	Streaming arbitrary precision arithmetic	10,000
`regex-redux`	Match DNA 8mers and substitute magic patterns	fasta output
`fasta`	Generate and write random DNA sequences	25M
`k-nucleotide`	Hashtable update and k-nucleotide strings	fasta output
`reverse-complement`	Read DNA sequences, write their reverse-complement	fasta output
`binary-trees`	Allocate, traverse and deallocate many binary trees	21
`chameneos-redux`	Symmetrical thread rendezvous requests	6M
`meteor-contest`	Search for solutions to shape packing puzzle	2,098
`thread-ring`	Switch from thread to thread passing one token	50M

- 27 Programming languages across different paradigms
  - **Functional** (e.g., Ocaml, F#, Haskell)
  - **Imperative** (e.g., C, Go, Python)
  - **Object-oriented** (e.g., C++, C#, Java)
  - Scripting (or **interpretative**) (e.g., JavaScript, Python, Ruby)
  - (These are not mutual exclusive)
- **Intel RAPL**'s C library to measure energy consumption

- Execute each benchmark solution 10 times.
  - Collect energy data and timestamps.
- **Two-minute interval** between executions

binary-trees

	Energy (J)	Time (ms)	Ratio (J/ms)	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\downarrow_1$ $\Uparrow_2$	100.74	3525	0.029	148
(v) Java $\uparrow_1$ $\Downarrow_{16}$	111.84	3306	0.034	1120
(v) Lisp $\downarrow_3$ $\Downarrow_3$	149.55	10570	0.014	373
(v) Racket $\downarrow_4$ $\Downarrow_6$	155.81	11261	0.014	467
(i) Hack $\uparrow_2$ $\Downarrow_9$	156.71	4497	0.035	502
(v) C# $\downarrow_1$ $\Downarrow_1$	189.74	10797	0.018	427
(v) F# $\downarrow_3$ $\Downarrow_1$	207.13	15637	0.013	432
(c) Pascal $\downarrow_3$ $\Uparrow_5$	214.64	16079	0.013	256
(c) Chapel $\uparrow_5$ $\Uparrow_4$	237.29	7265	0.033	335
(v) Erlang $\uparrow_5$ $\Uparrow_1$	266.14	7327	0.036	433
(c) Haskell $\uparrow_2$ $\Downarrow_2$	270.15	11582	0.023	494
(i) Dart $\downarrow_1$ $\Uparrow_1$	290.27	17197	0.017	475
(i) JavaScript $\downarrow_2$ $\Downarrow_4$	312.14	21349	0.015	916
(i) TypeScript $\downarrow_2$ $\Downarrow_2$	315.10	21686	0.015	915
(c) Go $\uparrow_3$ $\Uparrow_{13}$	636.71	16292	0.039	228
(i) Jruby $\uparrow_2$ $\Downarrow_3$	720.53	19276	0.037	1671
(i) Ruby $\Uparrow_5$	855.12	26634	0.032	482
(i) PHP $\Uparrow_3$	1,397.51	42316	0.033	786
(i) Python $\Uparrow_{15}$	1,793.46	45003	0.040	275
(i) Lua $\downarrow_1$	2,452.04	209217	0.012	1961
(i) Perl $\uparrow_1$	3,542.20	96097	0.037	2148
(c) Swift		n.e.		

binary-trees				
	Energy (J)	Time (ms)	Ratio (J/ms)	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\downarrow_1$ $\Uparrow_2$	100.74	3525	0.029	148
(v) Java $\uparrow_1$ $\Downarrow_{16}$	111.84	3306	0.034	1120
(v) Lisp $\downarrow_3$ $\Downarrow_3$	149.55	10570	0.014	373
(v) Racket $\downarrow_4$ $\Downarrow_6$	155.81	11261	0.014	467
(i) Hack $\uparrow_2$ $\Downarrow_9$	156.71	4497	0.035	502
(v) C# $\downarrow_1$ $\Downarrow_1$	189.74	10797	0.018	427
(v) F# $\downarrow_3$ $\Downarrow_1$	207.13	15637	0.013	432
(c) Pascal $\downarrow_3$ $\Uparrow_5$	214.64	16079	0.013	256
(c) Chapel $\uparrow_5$ $\Uparrow_4$	237.29	7265	0.033	335
(v) Erlang $\uparrow_5$ $\Uparrow_1$	266.14	7327	0.036	433
(c) Haskell $\uparrow_2$ $\Downarrow_2$	270.15	11582	0.023	494
(i) Dart $\downarrow_1$ $\Uparrow_1$	290.27	17197	0.017	475
(i) JavaScript $\downarrow_2$ $\Downarrow_4$	312.14	21349	0.015	916
(i) TypeScript $\downarrow_2$ $\Downarrow_2$	315.10	21686	0.015	915
(c) Go $\uparrow_3$ $\Uparrow_{13}$	636.71	16292	0.039	228
(i) Jruby $\uparrow_2$ $\Downarrow_3$	720.53	19276	0.037	1671
(i) Ruby $\Uparrow_5$	855.12	26634	0.032	482

Normalized global results for Energy, Time, and Memory.

Total					
	Energy (J)		Time (ms)		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Critical thinking

- There is no doubt this is an excellent study. Yet, as any excellent study, there's a lot we can **discuss** and **criticise** constructively.
- What kind of issues you see in drawing conclusions from such a table of results?
    - Is the benchmark representative of the most common usage behaviour?
    - Are the implemented solutions representative?
    - Does it make sense to use the average to compare energy consumption across different problems?
    - …

Reproducing with Rosetta Code

- Rosetta Code is a **programming chestomathy repository** (?)

> **chrestomathy**
>
> /krɛˈstɒməθi/
>
> *noun* **FORMAL**
>
> a selection of passages from an author or authors, designed to help in learning a language.

- 900 usecases/tasks solved across 700 different programming languages
- **Purpose**: if you know a programming language we can easily learn how the same task is solved in a language you are not familiar with.

Remove-duplicates		
	Energy (J)	Time (ms)
(c) Rust	0.01	1
(c) C++	0.12	5
(c) C	0.14	10
(c) Go	0.32	13
(i) Lua	0.51	21
(i) Perl	1.31	53
(i) JavaScript	1.73	73
(v) Erlang	2.36	96
(v) Java	2.96	214
(i) PHP	2.99	121
(i) Python	4.93	206
(i) Ruby	6.13	259
(v) Racket	7.54	318

Rosetta Code global ranking based on Energy.

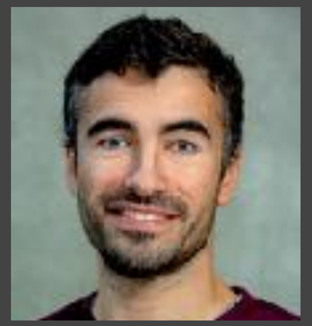
Rosetta Code Global Ranking	
Position	Language
1	C
2	Pascal
3	Ada
4	Rust
5	C++, Fortran
6	Chapel
7	OCaml, Go
8	Lisp
9	Haskell, JavaScript
10	Java
11	PHP
12	Lua, Ruby
13	Perl
14	Dart, Racket, Erlang
15	Python

Revisiting Research Questions

- **Can we compare** the energy efficiency of software languages?
- Is the **faster** language always the most **energy efficient**?
- ~~How does **memory usage** relate to **energy consumption**?~~
- Can we automatically decide what is the best programming language considering **energy**, **time**, and **memory usage**?
- How do the results of our energy consumption analysis of programming languages gathered from rigorous **performance benchmarking solutions compare to** results of **average day-to-day solutions**?
    - What would happen if we cherry picked the tasks?

3. Scientific Guide for Reliable Energy Experiments

Sustainable Software Engineering
CS4575


Luís Cruz
L.Cruz@tudelft.nl


Carolin Brandt
C.E.Brandt@tudelft.nl


Enrique Barba Roque
E.BarbaRoque@tudelft.nl