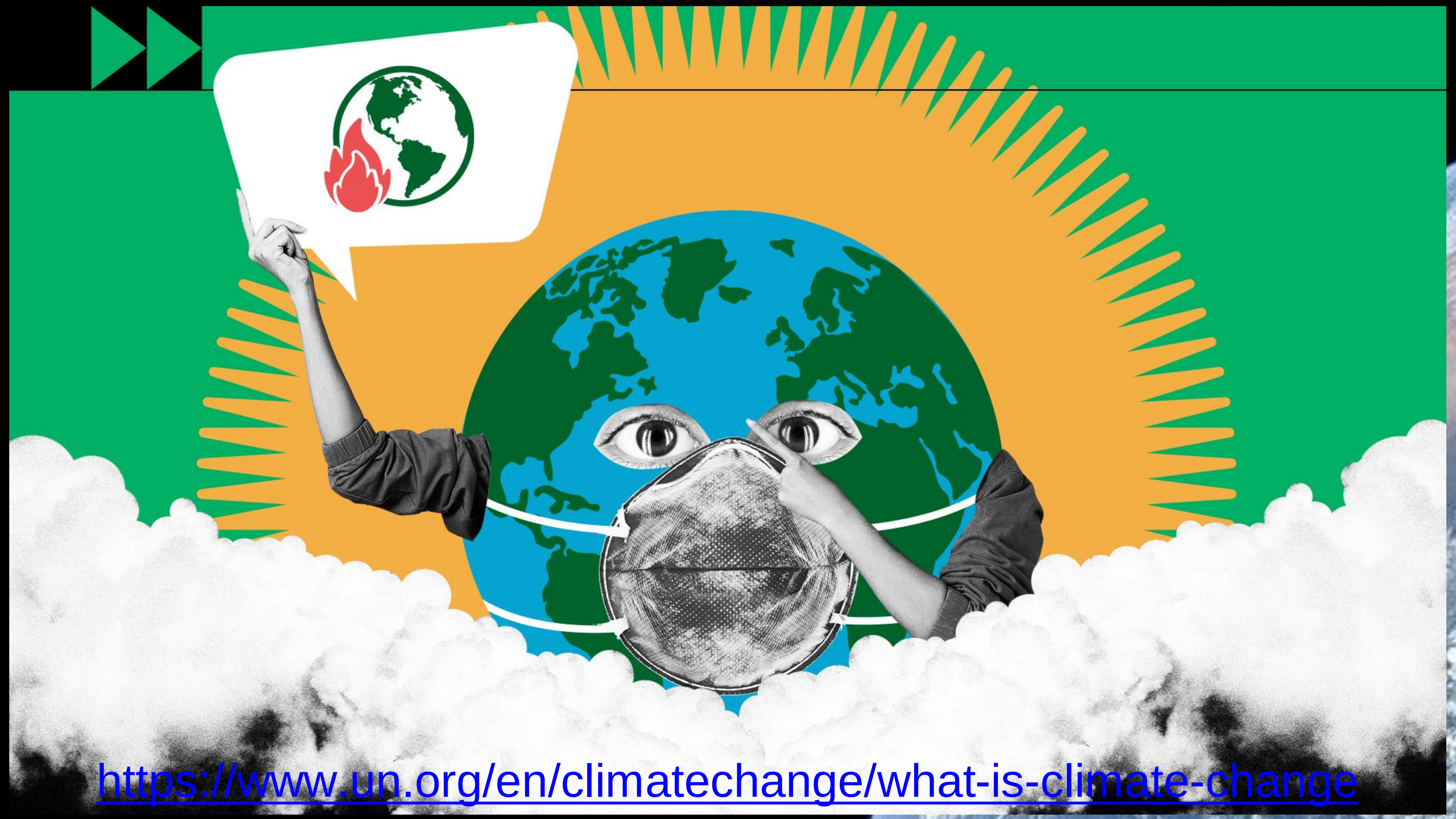
Towards a better planet with AI and IoT

Florian Michahelles

TU WIEN

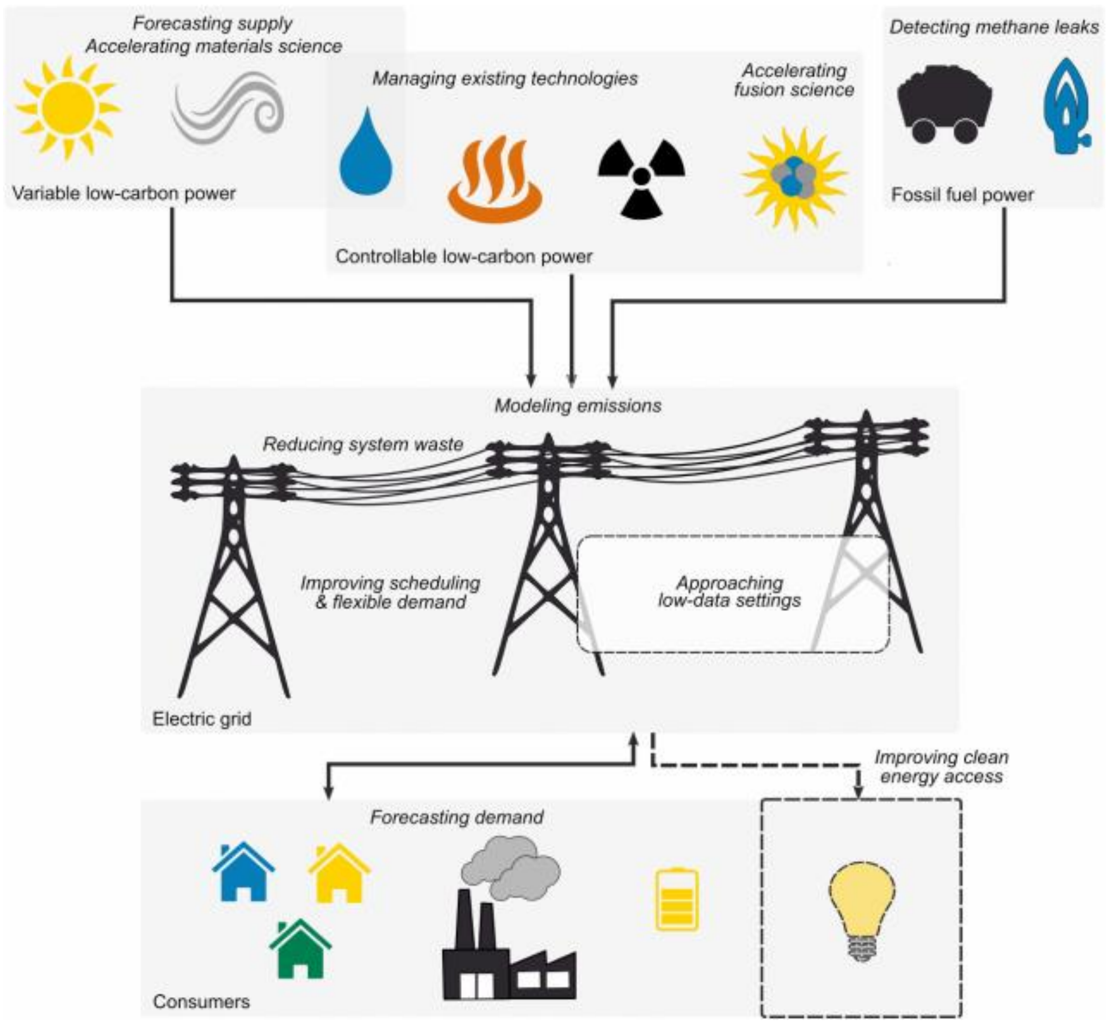
ARTIFACT-BASED COMPUTING & USER RESEARCH

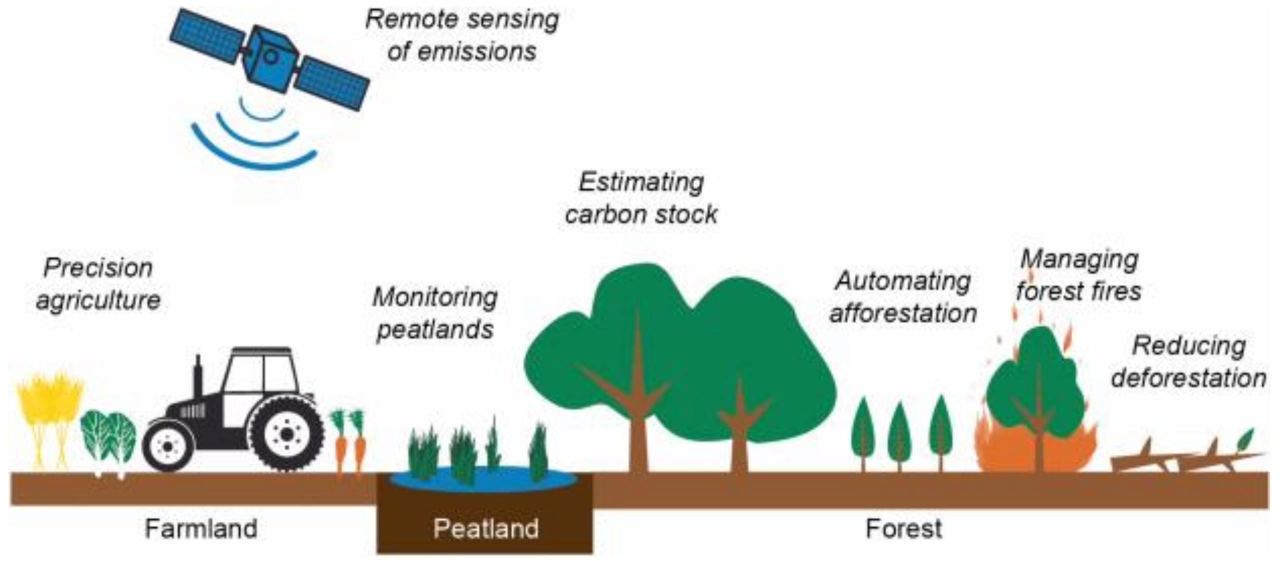


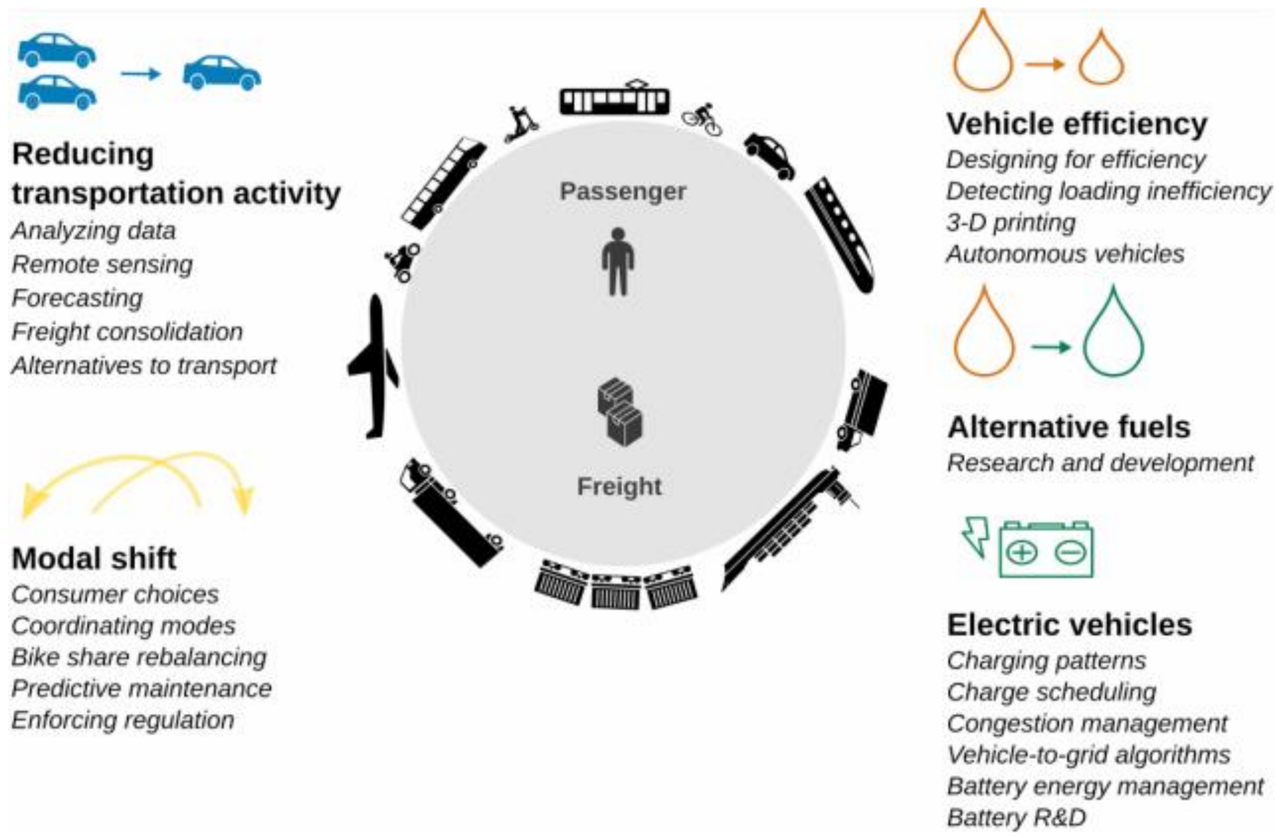
https://www.un.org/en/climatechange/what-is-climate-change

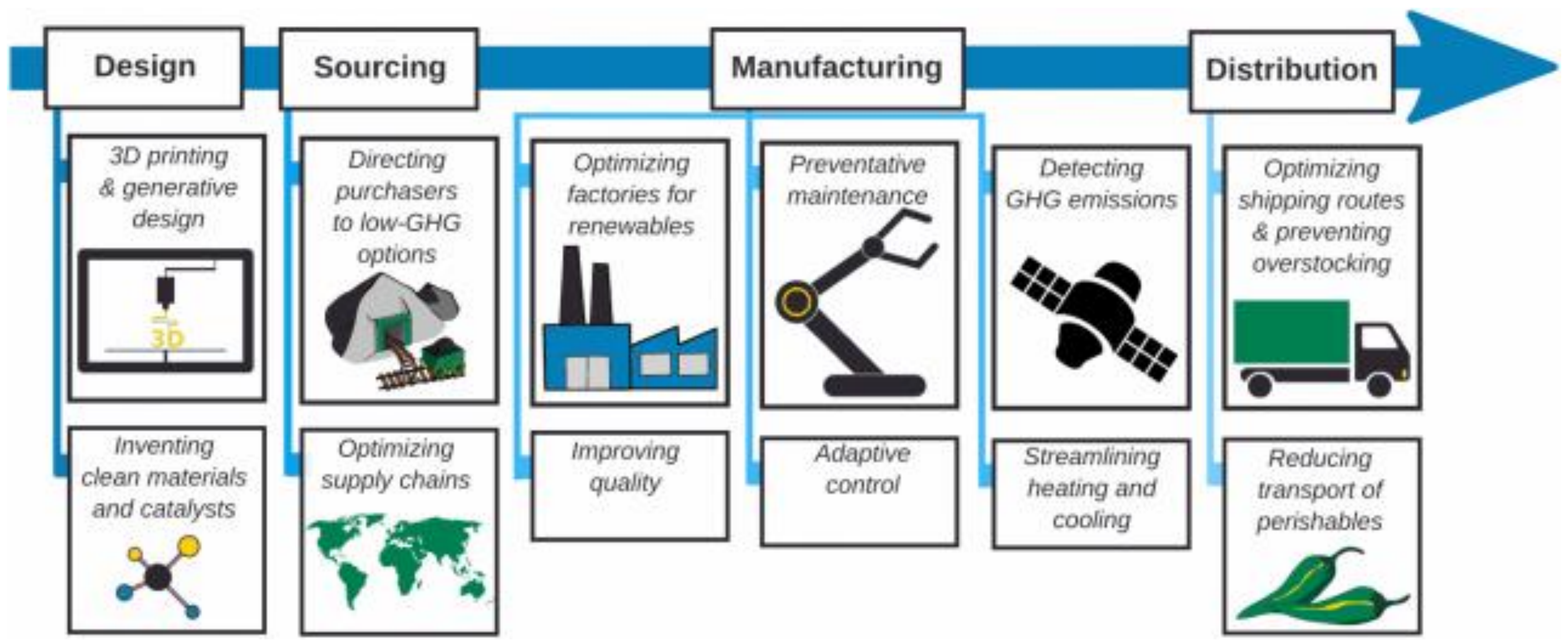
https://www.un.org/en/climatechange/what-is-climate-change

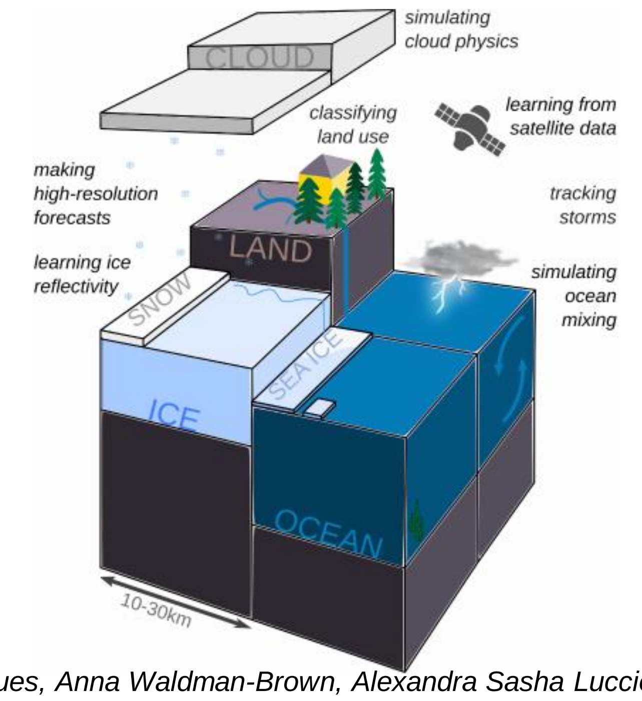
"Tackling Climate Change with Machine Learning"…

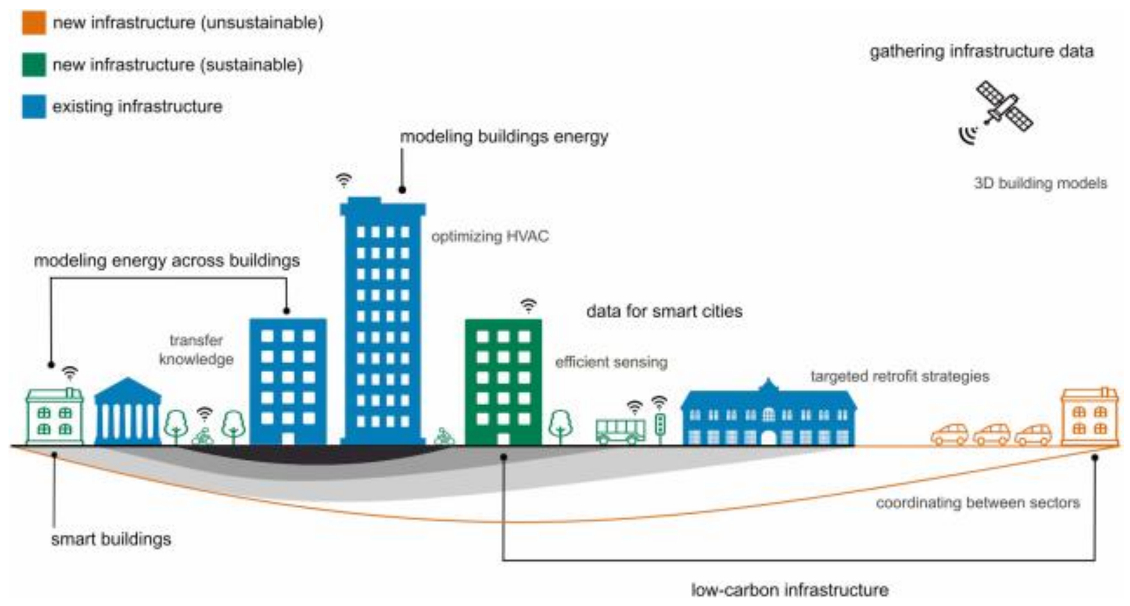










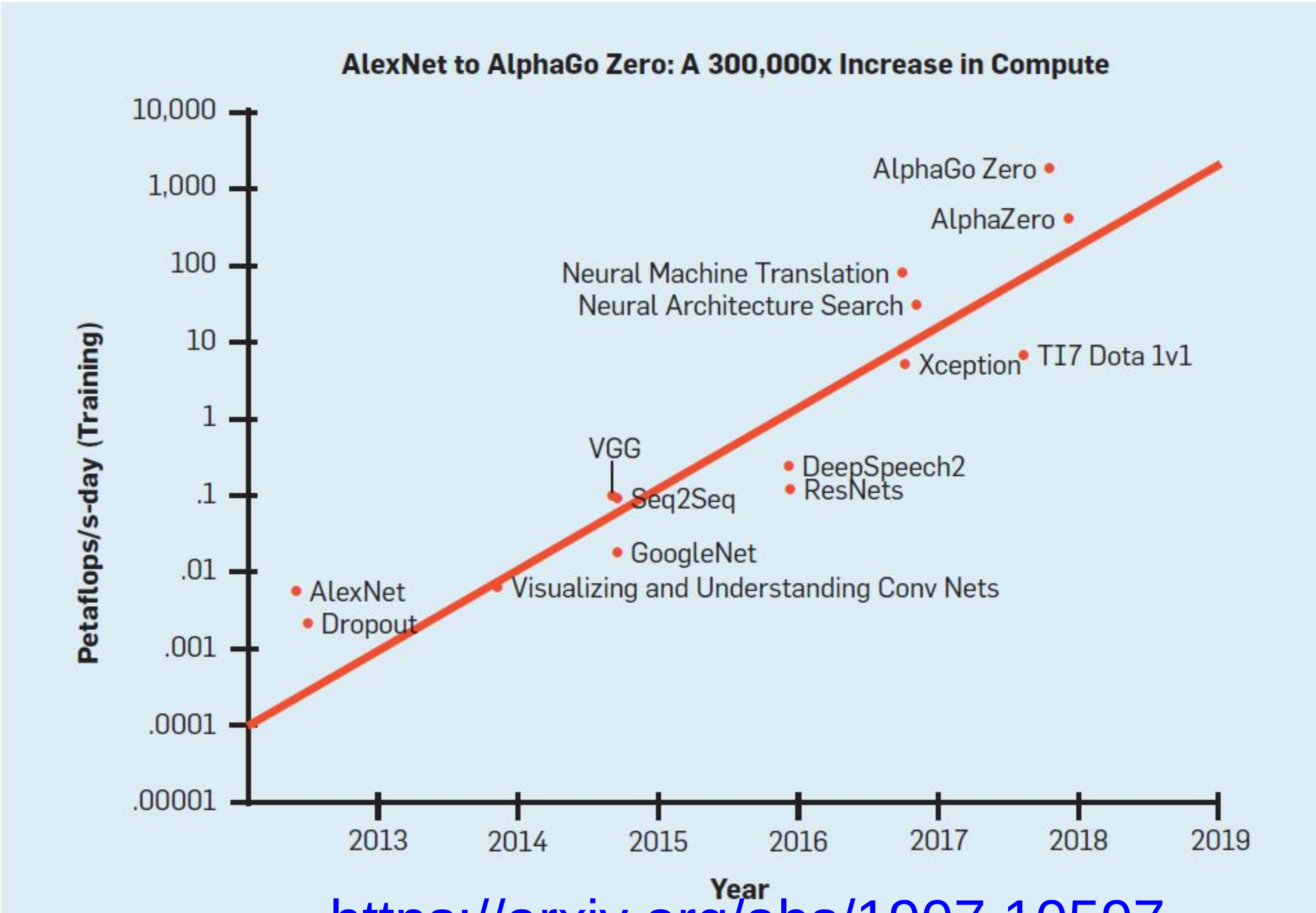


https://dl.acm.org/doi/10.1145/3485128

David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Sasha Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Kording, Carla P. Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer Chayes, and Yoshua Bengio. 2022. Tackling Climate Change with Machine Learning. ACM Comput. Surv. 55, 2, Article 42 (March 2023), 96 pages. DOI:https://doi.org/10.1145/3485128

…only works if we…

…provide equitable access to computation resources
…prioritize computationally efficient hardware and algorithms
…report training time and sensitivity to hyperparameters.



https://arxiv.org/abs/1907.10597

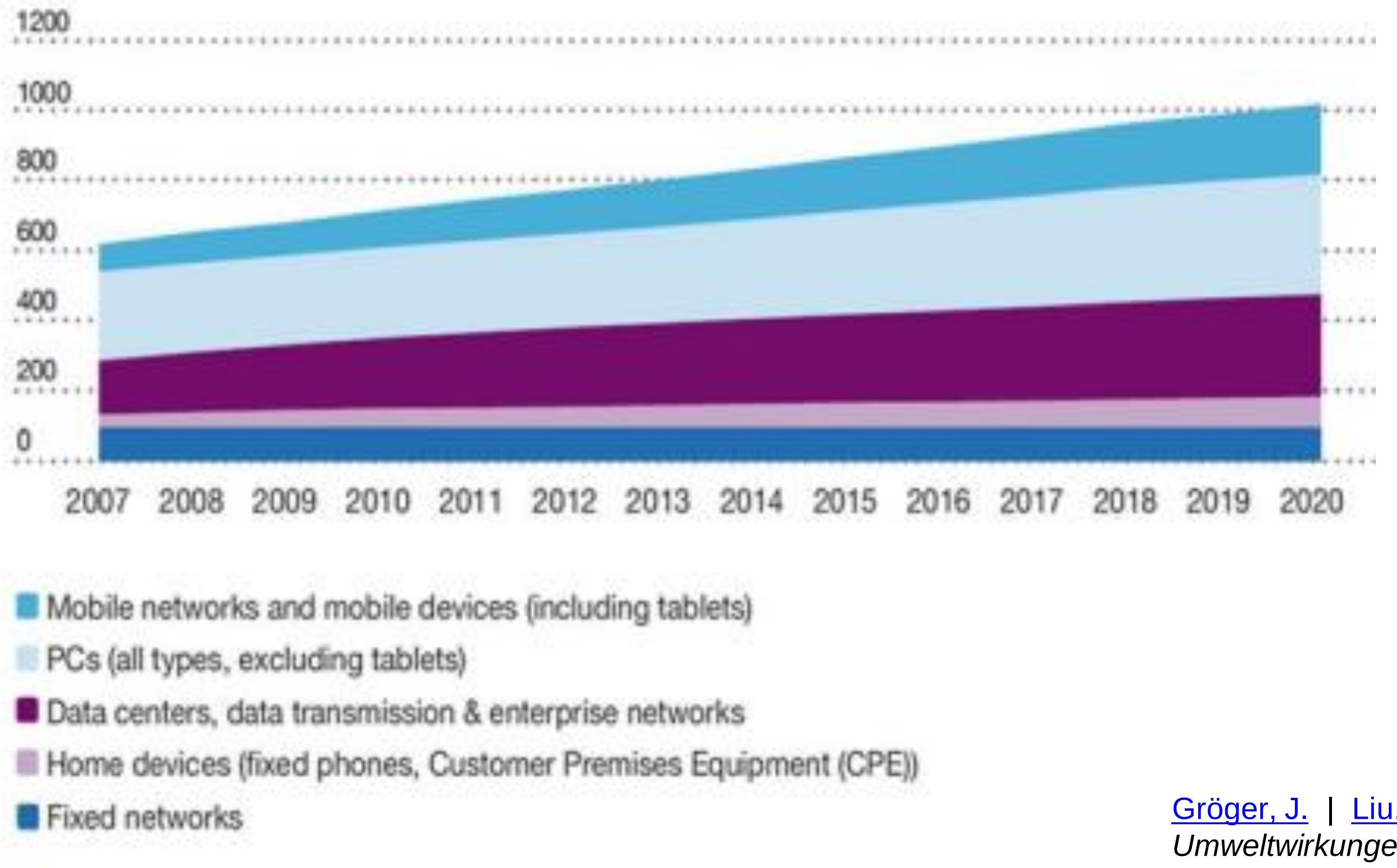
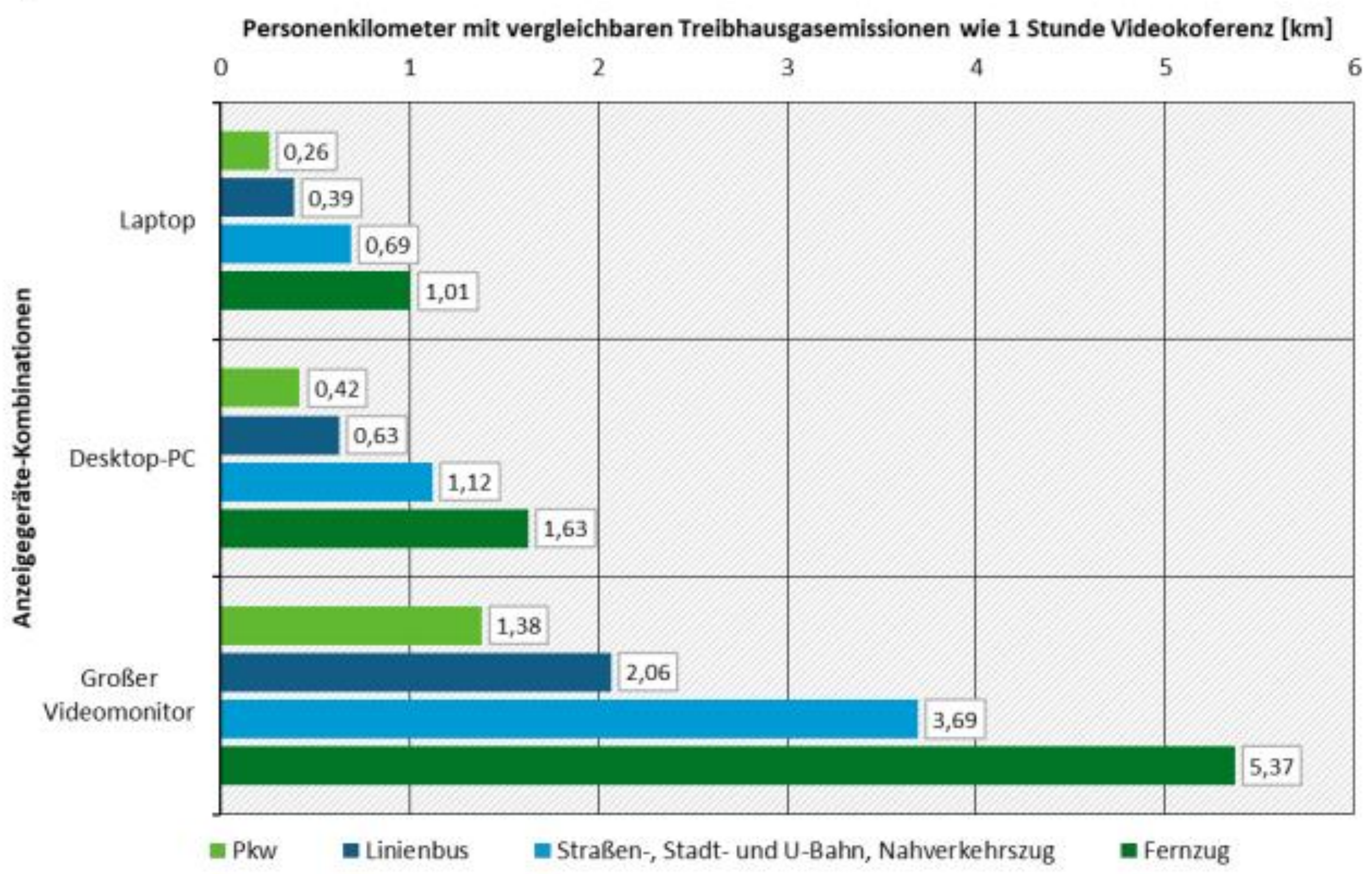


Fig. 1.1 ICT carbon footprint outlook (Mtonnes CO2e) (from [2])

Abbildung 27: Vergleich der Treibhausgasemissionen von Videokonferenzen mit verschiedenen Anzeigegeräten mit den Personenkilometern verschiedener Verkehrsmittel



Quelle: Eigene Darstellung, Öko-Institut

Gröger, J. | Liu, R. | Stobbe, L. | Druschke, J. | Richter, N.: Green Cloud Computing *Lebenszyklusbasierte Datenerhebung zu Umweltwirkungen des Cloud Computing – Abschlussbericht* 06 / 2021
https://www.umweltbundesamt.de/sites/default/files/medien/5750/publikationen/2021-06-17_texte_94-2021_green-cloud-computing.pdf

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$\text{Transformer}_{base}$	P100x8	1415.78	12	27	26	$41–$140
Transformer_{big}	P100x8	1515.43	84	201	192	$289–$981
ELMo	P100x3	517.66	336	275	262	$433–$1472
BERT_{base}	V100x64	12,041.51	79	1507	1438	$3751–$12,571
BERT_{base}	TPUv2x16	—	96	—	—	$2074–$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	$942,973–$3,201,722
NAS	TPUv2x1	—	32,623	—	—	$44,055–$146,848
GPT-2	TPUv3x32	—	168	—	—	$12,902–$43,008

Table 3: Estimated cost of training a model in terms of CO_2 emissions (lbs) and cloud compute cost (USD).[7] Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.

https://arxiv.org/pdf/1906.02243.pdf

Florian Michahelles

What's the efficiency of software languages?

A GREENER LOGIC

Table 1. CLBG corpus of programs.

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

Table 2. Languages sorted by paradigm

Paradigm	Languages
Functional	Erlang, F#, Haskell, Lisp, Ocaml, Perl, Racket, Ruby, Rust;
Imperative	Ada, C, C++, F#, Fortran, Go, Ocaml, Pascal, Rust;
Object-Oriented	Ada, C++, C#, Chapel, Dart , F#, Java, JavaScript, Ocaml, Perl, PHP, Python, Racket, Rust, Smalltalk, Swift, TypeScript;
Scripting	Dart, Hack, JavaScript, JRuby, Lua, Perl, PHP, Python, Ruby, TypeScript;

```
...
for (i = 0 ; i < N ; i++){
  time_before = getTime(...);
  //performs initial energy measurement
  rapl_before(...);

  //executes the program
  system(command);

  //computes the difference between
  //this measurement and the initial one
  rapl_after(...);
  time_elapsed = getTime(...) - time_before;
  ...
}
...
```

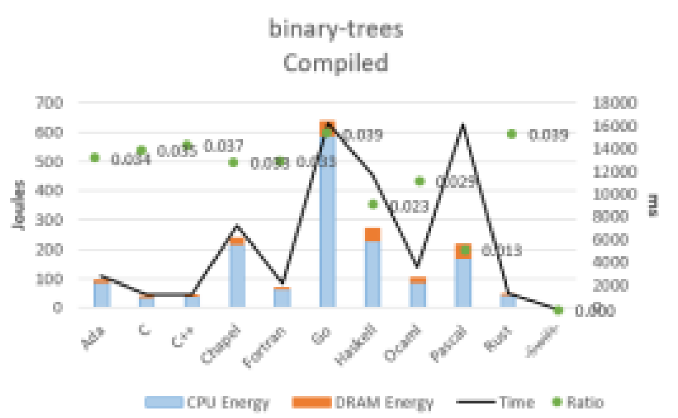
Listing 1. Overall process of the energy measuring framework.

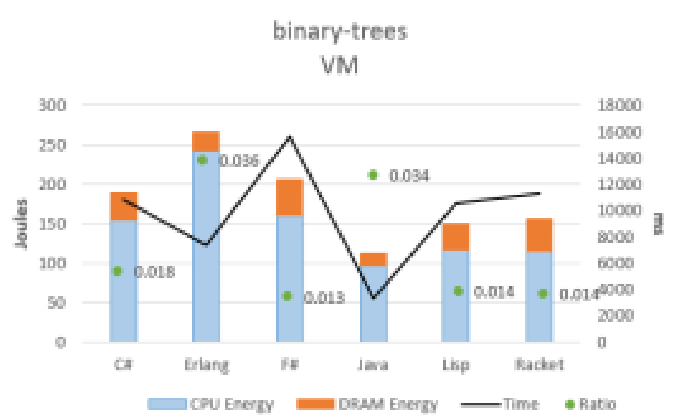
Table 3. Results for binary-trees, fannkuch-redux, and fasta

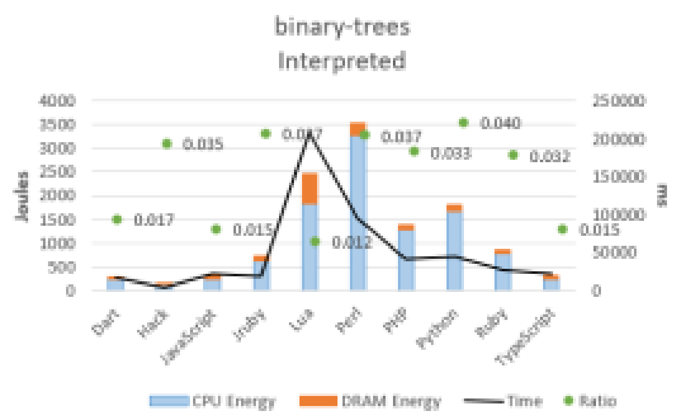
binary-trees	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\downarrow_1$ $\Uparrow_2$	100.74	3525	0.029	148
(v) Java $\uparrow_1$ $\Downarrow_{16}$	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.		

fannkuch-redux	Energy	Time	Ratio	Mb
(c) C $\Downarrow_2$	215.92	6076	0.036	2
(c) C++ $\Uparrow_1$	219.89	6123	0.036	1
(c) Rust $\Downarrow_{11}$	238.30	6628	0.036	16
(c) Swift $\Downarrow_5$	243.81	6712	0.036	7
(c) Ada $\Downarrow_2$	264.98	7351	0.036	4
(c) Ocaml $\downarrow_1$	277.27	7895	0.035	3
(c) Chapel $\uparrow_1$ $\Downarrow_{18}$	285.39	7853	0.036	53
(v) Lisp $\downarrow_3$ $\Downarrow_{15}$	309.02	9154	0.034	43
(v) Java $\uparrow_1$ $\Downarrow_{13}$	311.38	8241	0.038	35
(c) Fortran $\Downarrow_1$	316.50	8665	0.037	12
(c) Go $\uparrow_2$ $\Uparrow_7$	318.51	8487	0.038	2
(c) Pascal $\Uparrow_{10}$	343.55	9807	0.035	2
(v) F# $\downarrow_1$ $\Downarrow_7$	395.03	10950	0.036	34
(v) C# $\uparrow_1$ $\Downarrow_5$	399.33	10840	0.037	29
(i) JavaScript $\downarrow_1$ $\Downarrow_2$	413.90	33663	0.012	26
(c) Haskell $\uparrow_1$ $\Uparrow_8$	433.68	14666	0.030	7
(i) Dart $\Downarrow_7$	487.29	38678	0.013	46
(v) Racket $\Uparrow_3$	1,941.53	43680	0.044	18
(v) Erlang $\Uparrow_3$	4,148.38	101839	0.041	18
(i) Hack $\Downarrow_6$	5,286.77	115490	0.046	119
(i) PHP	5,731.88	125975	0.046	34
(i) TypeScript $\downarrow_4$ $\Uparrow_4$	6,898.48	516541	0.013	26
(i) Jruby $\uparrow_1$ $\Downarrow_4$	7,819.03	219148	0.036	669
(i) Lua $\downarrow_3$ $\Uparrow_{19}$	8,277.87	635023	0.013	2
(i) Perl $\uparrow_2$ $\Uparrow_{12}$	11,133.49	249418	0.045	12
(i) Python $\uparrow_2$ $\Uparrow_{14}$	12,784.09	279544	0.046	12
(i) Ruby $\uparrow_2$ $\Uparrow_{17}$	14,064.98	315583	0.045	8

fasta	Energy	Time	Ratio	Mb
(c) Rust $\Downarrow_9$	26.15	931	0.028	16
(c) Fortran $\downarrow_6$	27.62	1661	0.017	1
(c) C $\uparrow_1$ $\Downarrow_1$	27.64	973	0.028	3
(c) C++ $\uparrow_1$ $\Downarrow_2$	34.88	1164	0.030	4
(v) Java $\uparrow_1$ $\Downarrow_{12}$	35.86	1249	0.029	41
(c) Swift $\Downarrow_9$	37.06	1405	0.026	31
(c) Go $\downarrow_2$	40.45	1838	0.022	4
(c) Ada $\downarrow_2$ $\Uparrow_3$	40.45	2765	0.015	3
(c) Ocaml $\downarrow_2$ $\Downarrow_{15}$	40.78	3171	0.013	201
(c) Chapel $\uparrow_5$ $\Downarrow_{10}$	40.88	1379	0.030	53
(v) C# $\uparrow_4$ $\Downarrow_5$	45.35	1549	0.029	35
(i) Dart $\Downarrow_6$	63.61	4787	0.013	49
(i) JavaScript $\Downarrow_1$	64.84	5098	0.013	30
(c) Pascal $\downarrow_1$ $\Uparrow_{13}$	68.63	5478	0.013	0
(i) TypeScript $\downarrow_2$ $\Downarrow_{10}$	82.72	6909	0.012	271
(v) F# $\uparrow_2$ $\Uparrow_3$	93.11	5360	0.017	27
(v) Racket $\downarrow_1$ $\Uparrow_5$	120.90	8255	0.015	21
(c) Haskell $\uparrow_2$ $\Downarrow_8$	205.52	5728	0.036	446
(v) Lisp $\Downarrow_2$	231.49	15763	0.015	75
(i) Hack $\Downarrow_3$	237.70	17203	0.014	120
(i) Lua $\Uparrow_{18}$	347.37	24617	0.014	3
(i) PHP $\downarrow_1$ $\Uparrow_{13}$	430.73	29508	0.015	14
(v) Erlang $\uparrow_1$ $\Uparrow_{12}$	477.81	27852	0.017	18
(i) Ruby $\downarrow_1$ $\Uparrow_2$	852.30	61216	0.014	104
(i) JRuby $\uparrow_1$ $\Downarrow_2$	912.93	49509	0.018	705
(i) Python $\downarrow_1$ $\Uparrow_{18}$	1,061.41	74111	0.014	9
(i) Perl $\uparrow_1$ $\Uparrow_8$	2,684.33	61463	0.044	53







Pereira, Rui, et al. "Energy efficiency across programming languages: how do energy, time, and memory relate?." *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. 2017.

Is the faster language always the more energy efficient?

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		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

Pereira, Rui, et al. "Energy efficiency across programming languages: how do energy, time, and memory relate?." *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. 2017.

Once we can decide what is the best software language…

- execution time and energy consumption, then yes, it is almost always possible to choose the best language.
- if memory is also a concern, it is no longer possible to automatically decide for a single language

Table 5. Pareto optimal sets for different combination of objectives.

Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Swift • PHP	Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
F# • Racket • Hack • Python	Lisp • Ocaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
JavaScript • Ruby	Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Dart • TypeScript • Erlang	Swift	TypeScript	Lua • JRuby • Perl
JRuby • Perl	Dart • F#	Erlang • Lua • Perl	
Lua	JavaScript	JRuby	
	Racket		
	TypeScript • Hack		
	PHP		
	Erlang		
	Lua • JRuby		
	Ruby		

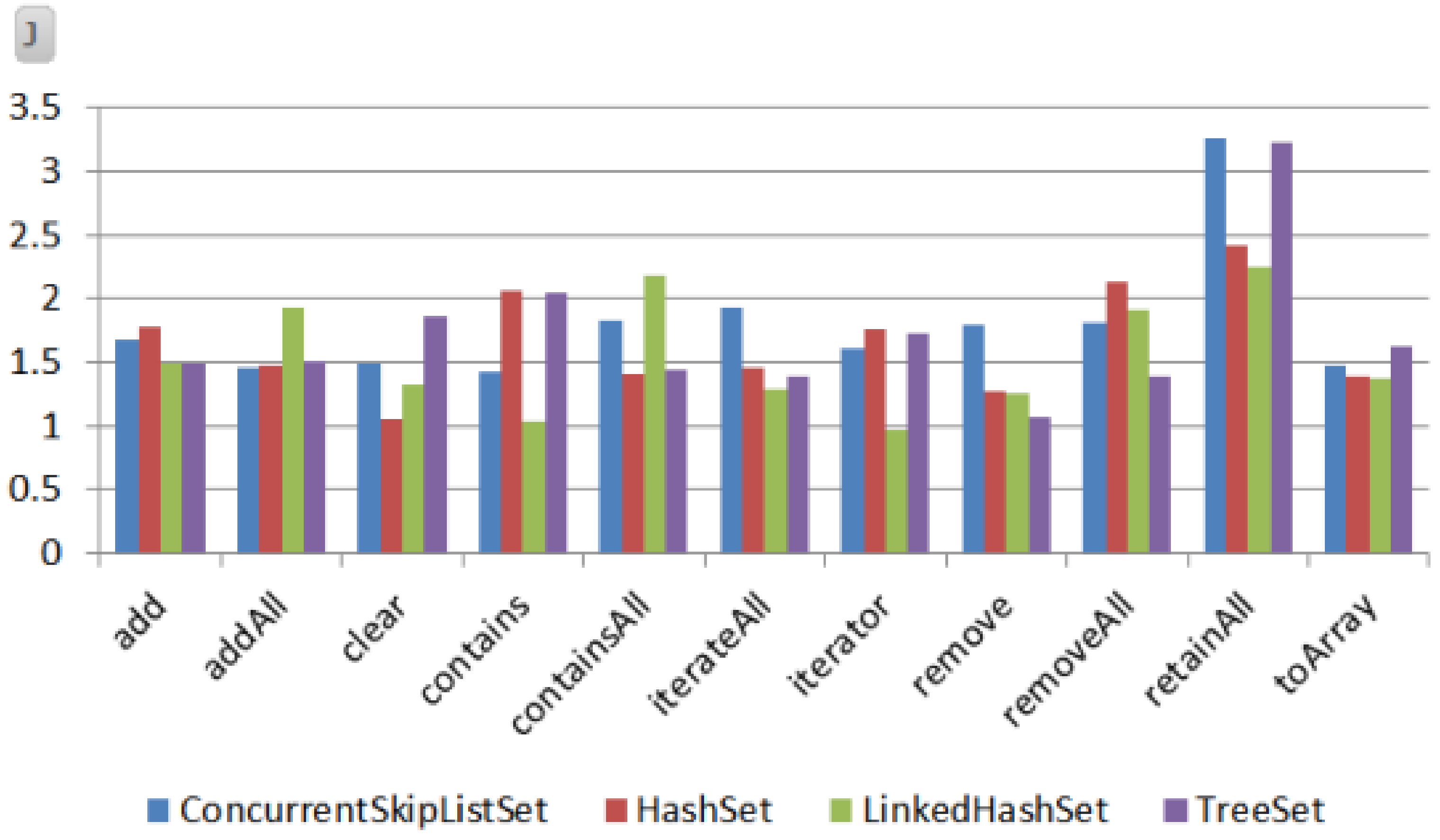
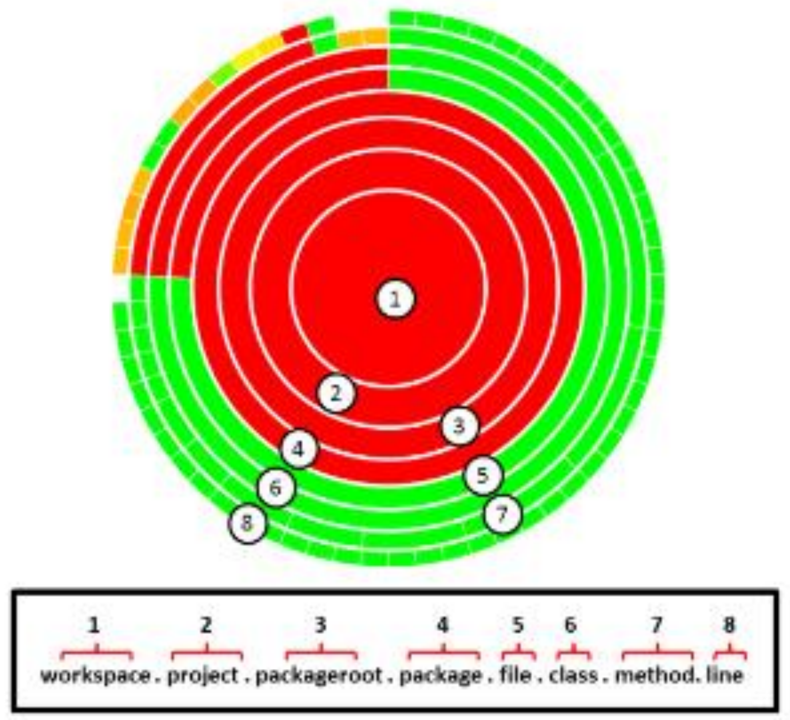


Figure 5.4: Set results graph for population of 25k

https://greenlab.di.uminho.pt/wp-content/uploads/2019/02/Rui-Alexandre-Afonso-Pereira.pdf p.89

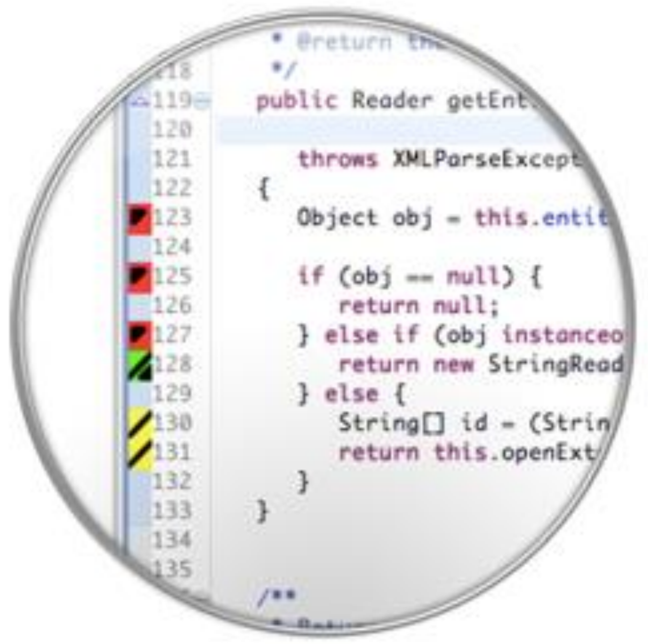
Pereira, Rui, et al. "Energy efficiency across programming languages: how do energy, time, and memory relate?." *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. 2017.

…tools for green coding emerge!

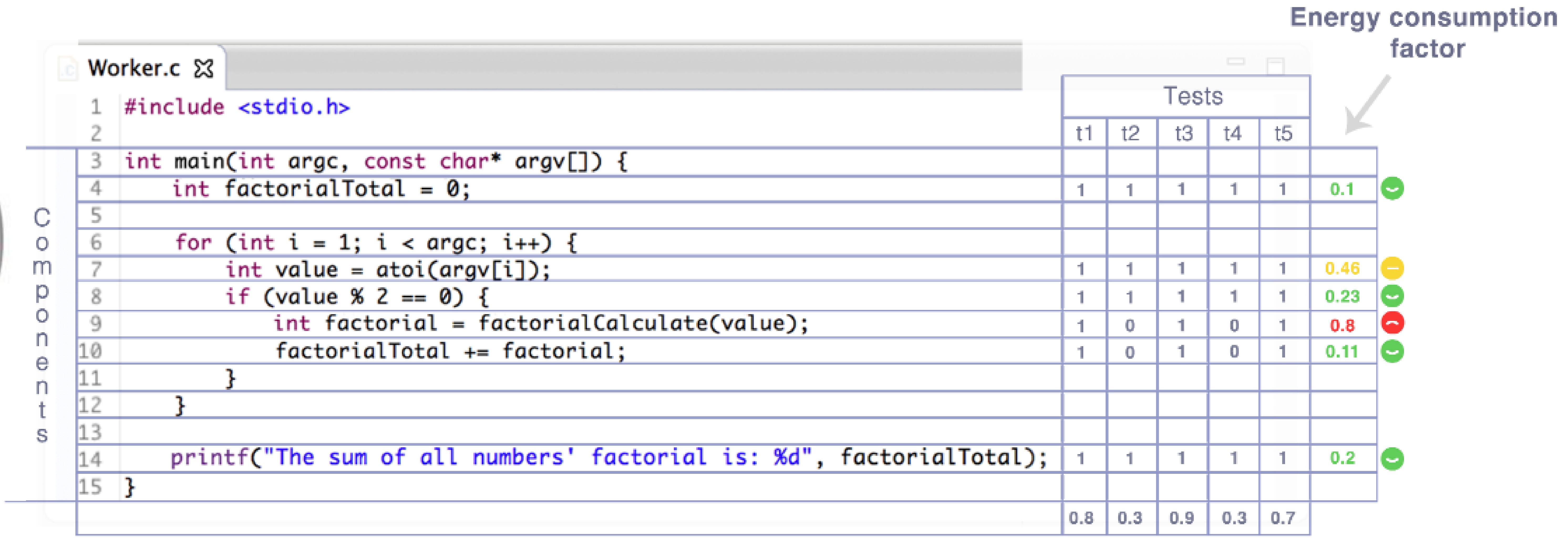


Sandburst visualization

+



Flag the code



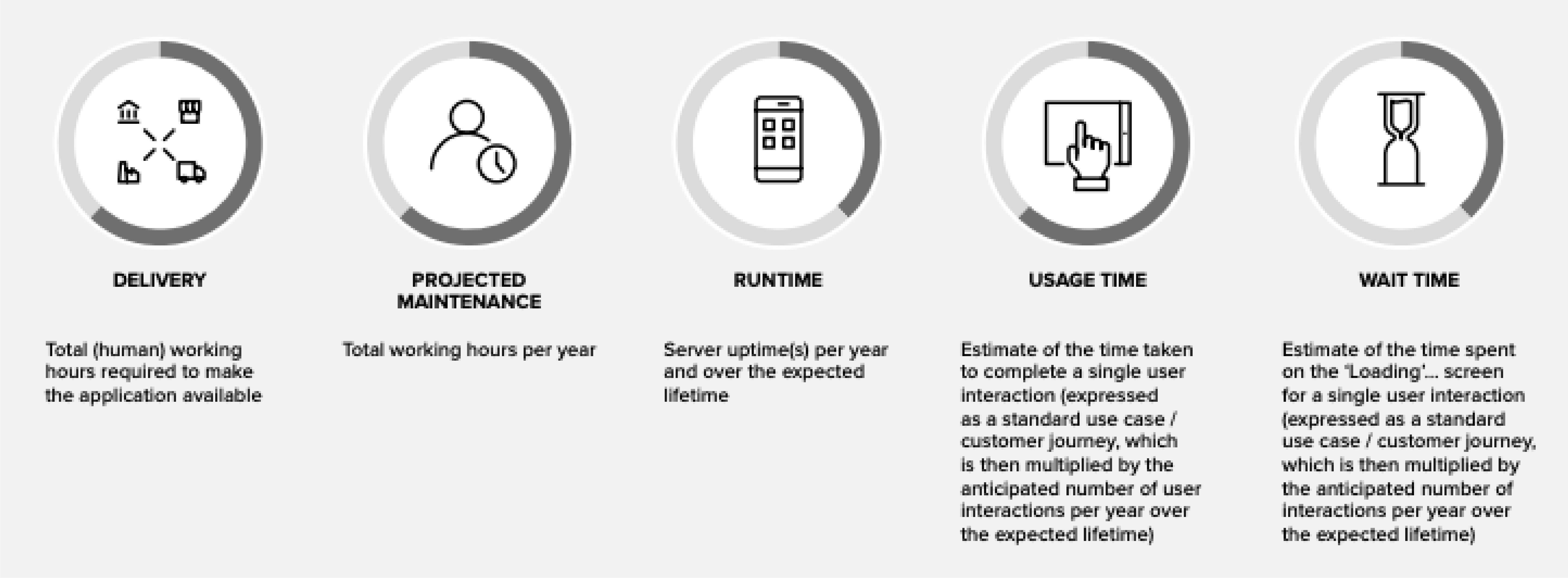
Use the Spectrum-based Fault Localization techniques to identify energy leaks

T. Carção, "Measuring and visualizing energy consumption within software code," *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, pp. 181-182, doi: 10.1109/VLHCC.2014.6883045.

Think bigger:
Review of the software develoment cycle

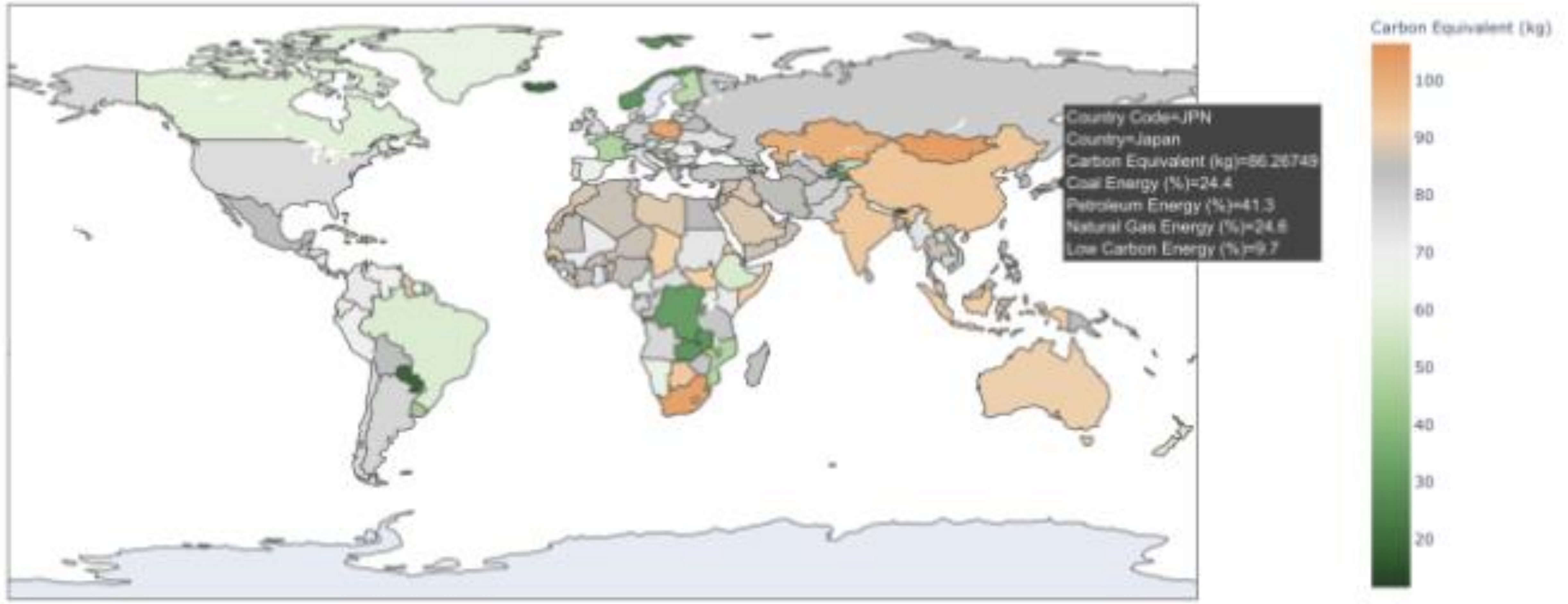
A GREENER METHODOLOGY

- sustainability criteria and metrics for software products
- model all extractions from the environment (e.g. ores, fossil resources or fuels, water) and all emissions to the environment (e.g. heavy metals, CO_2, CH_4, radiating particles) that occur in each life cycle phase.

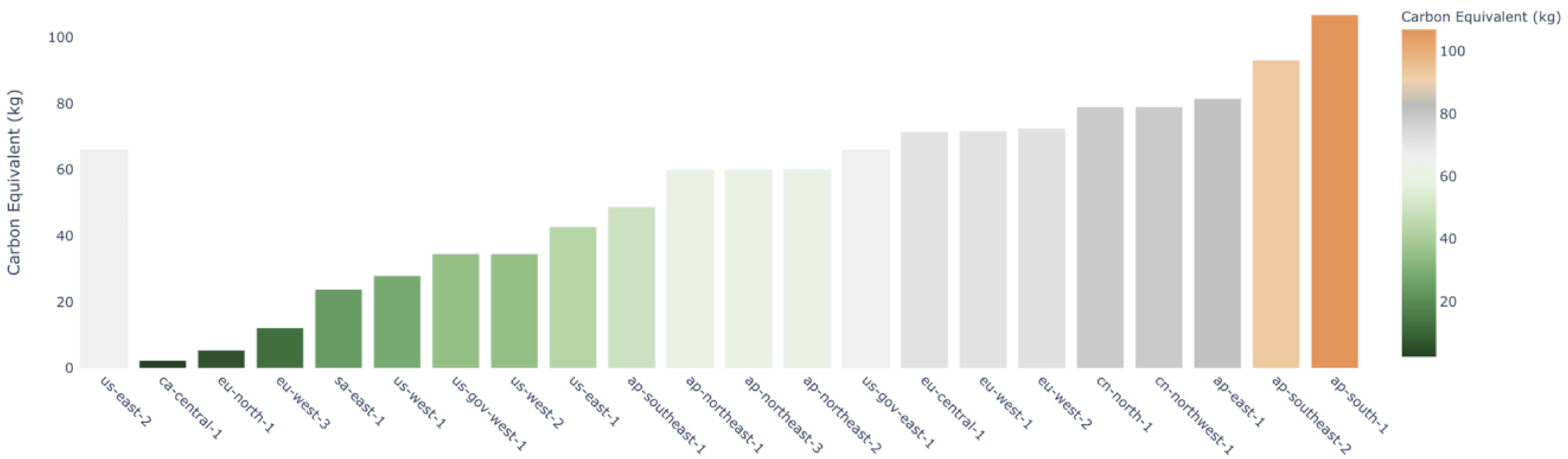


Example: Track CO2 from _your_ computing

```
from codecarbon import EmissionsTracker

tracker = EmissionsTracker()

tracker.start()

# GPU Intensive code goes here

tracker.stop()
```



https://codecarbon.io/



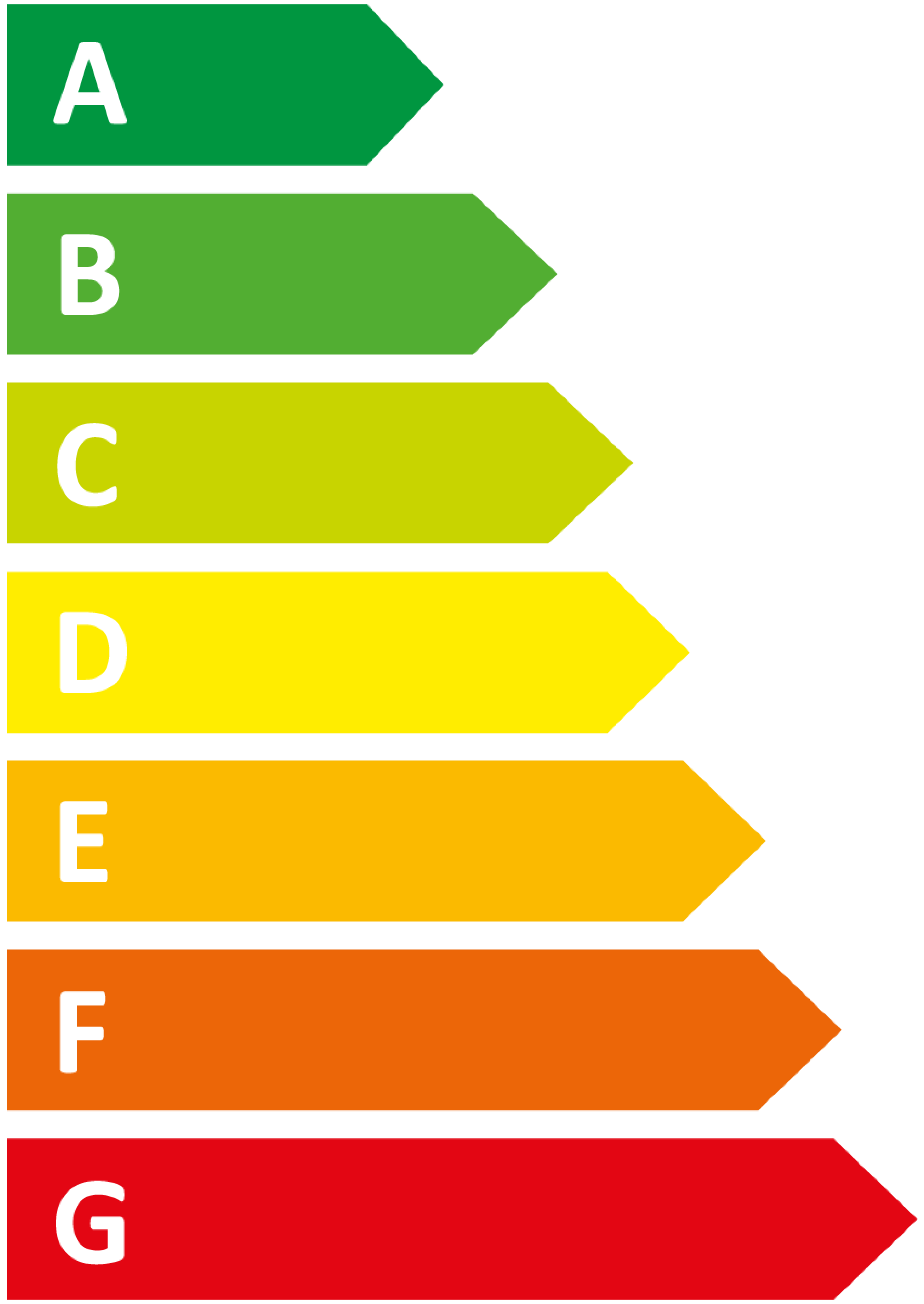
Florian Michahelles

Pop Quiz

- How much energy does your home TV use in an hour?
- How many full phone charges can the same amount of energy provide?
  - 10
  - 20
  - 40
  - 80
  - more

Source: https://carbon.kilowh.at/

But.. Are all devices created equal?



Can’t we just “read the label”?

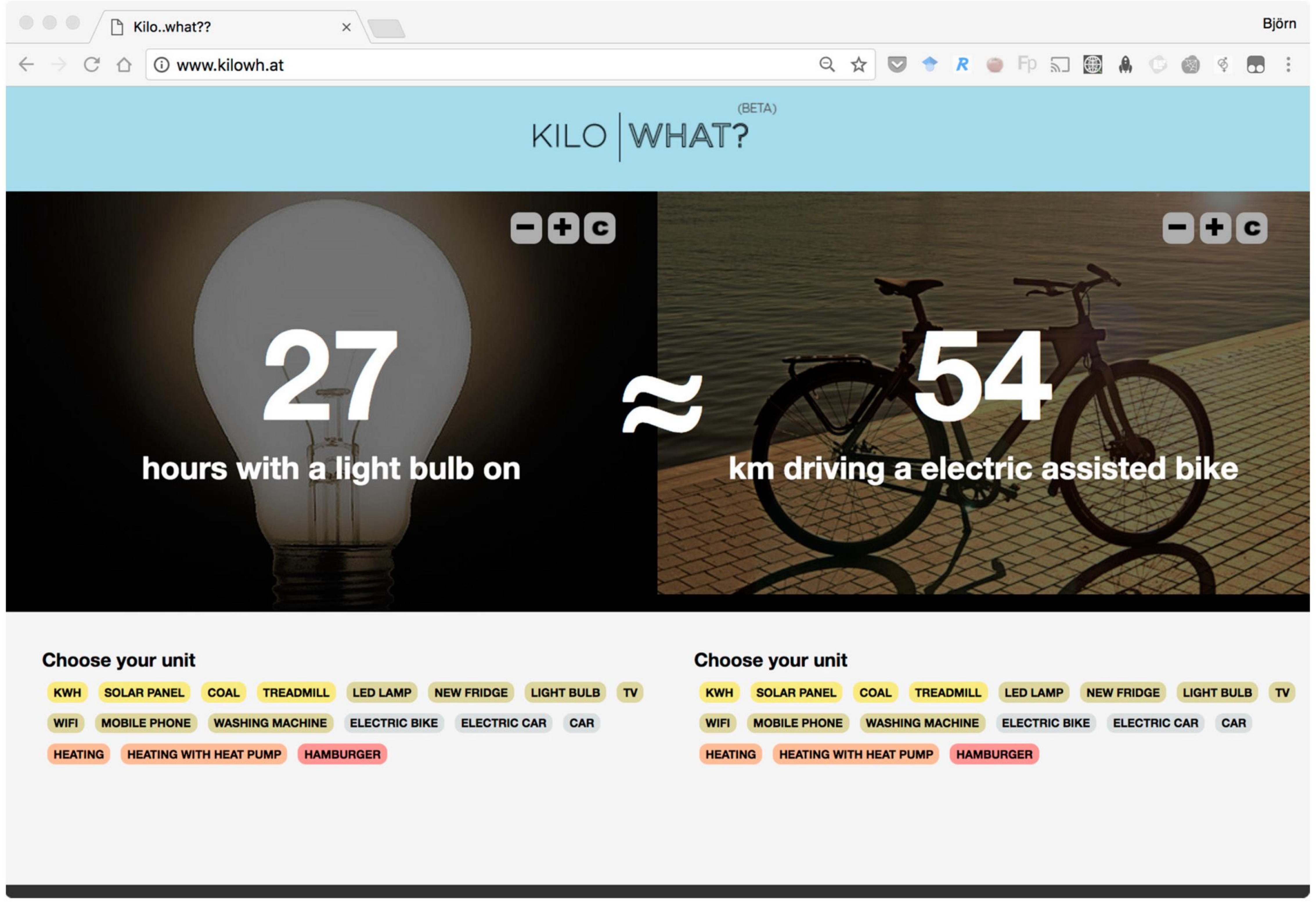
Florian Michahelles

TU WIEN

How many do you have at home?

Using metaphors

J. L. Zapico and B. Hedin, "Energy weight: Tangible interface for increasing energy literacy," 2017 Sustainable Internet and ICT for Sustainability (SustainIT), 2017, pp. 1-3, doi: 10.23919/SustainIT.2017.8379807.



Hedin, B.; Luis Zapico, J. What Can You Do with 100 kWh? A Longitudinal Study of Using an Interactive Energy Comparison Tool to Increase Energy Awareness [†]. *Sustainability* **2018**, *10*, 2269. https://doi.org/10.3390/su10072269

The Goal

Create a tool that enables non-experts (i.e., the layperson) to estimate and quantify personal energy use through understandable tangible metaphors.

Use Case: Indoor Awareness

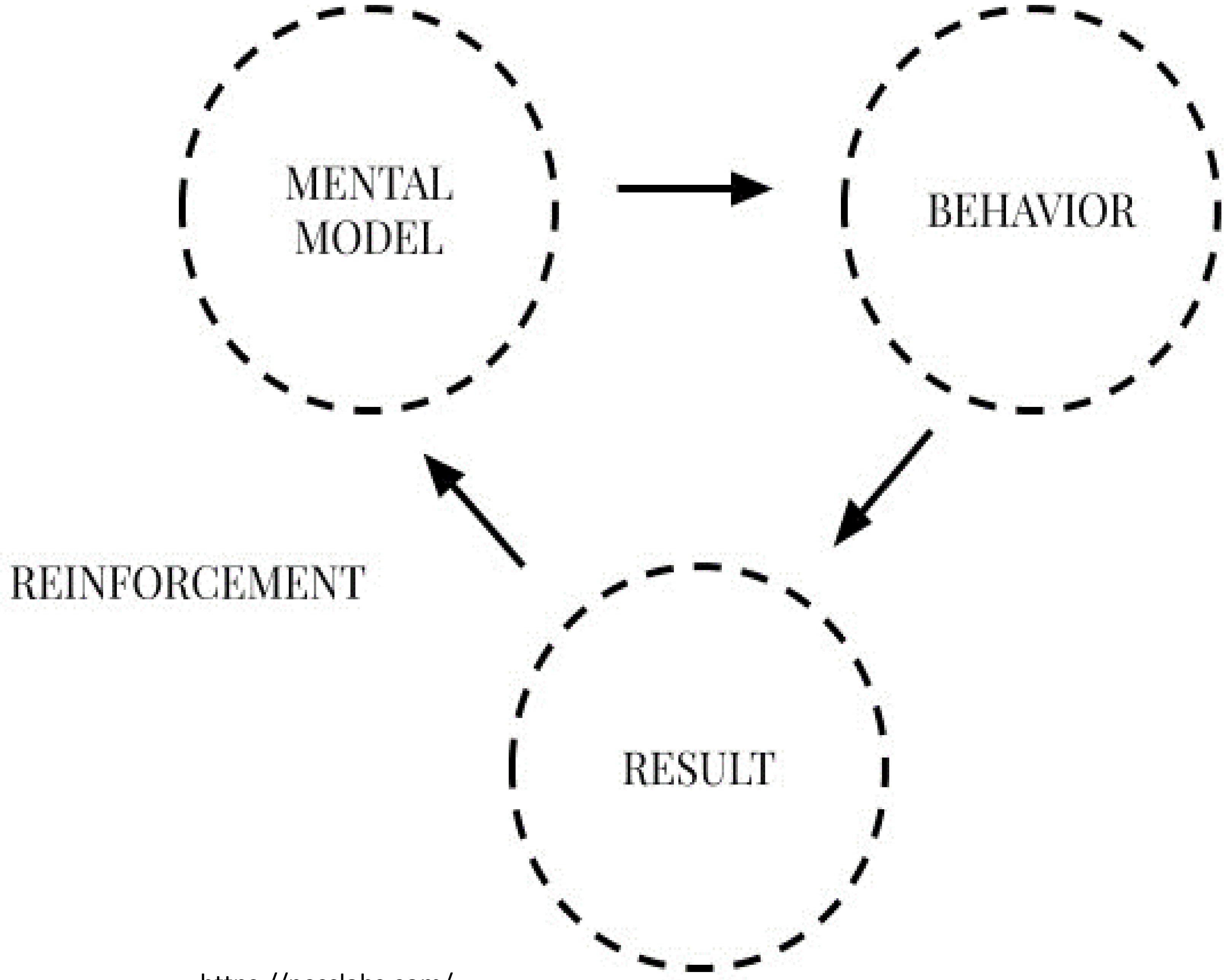
“using YOUR laptop for X hours takes as much energy as charging YOUR phone from 0-100% Y times.”

Use Case: Outdoor "Autarky"

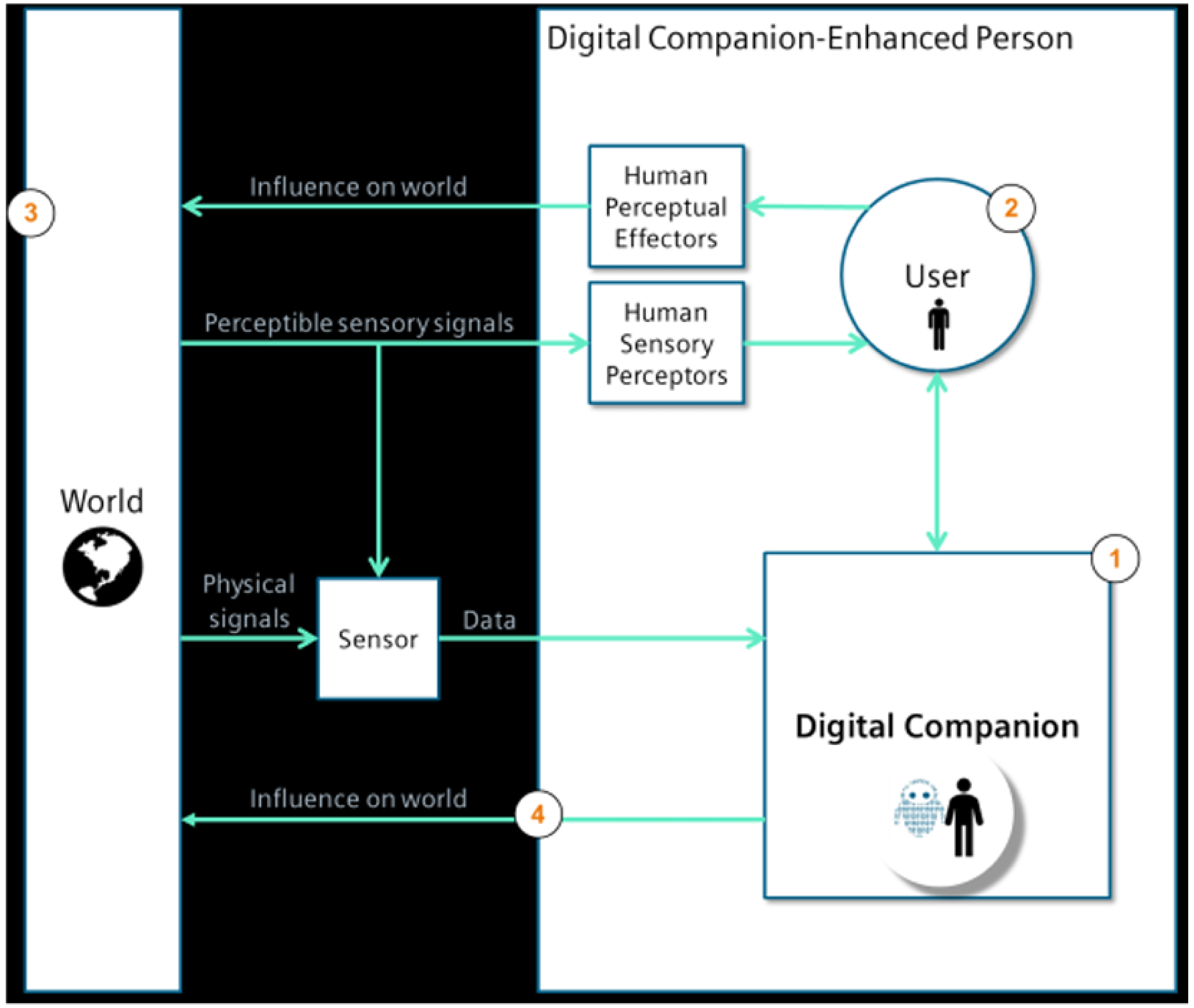
Autarky comes from the Greek words *autos* (self) and *arkein* (to govern)

The Bigger Picture

- Making informed decisions
  - Education
  - Energy use “mental model”
  - Digital companions
  - Decision support



https://nesslabs.com/



Kritzler et al. (2019), doi: https://doi.org/10.1145/3308560.3316510

Let's...

- …look into how our cloud-services are powered and where they run.
- …document the consumption of ML model training.
- …quantify systematically the impacts of software
- …refine our methodologies: document, reconsider, redesign

- …provide an understanding of energy

Thank you.

References

- Green software
  - The Green Computing Grand Challenge (GC2)
  - SCI Reporting
  - Principles of Green Software Engineering
  - SCI Open Ontology
  - SCI Open Data
  - Carbon Aware SDK
  - Software Carbon Intensity (SCI) Specification

- Cloud Computing
  - https://principles.green/
  - https://greensoftware.foundation/
  - https://github.com/Green-Software-Foundation/awesome-green-software
  - https://github.com/Breakend/experiment-impact-tracker
  - https://github.com/etsy/cloud-jewels

Florian Michahelles
professor of ubiquitous computing

TU WIEN
ARTIFACT-BASED COMPUTING & USER RESEARCH

florian.michahelles@tuwien.ac.at
media.tuwien.ac.at

Baking sustainability into the development process

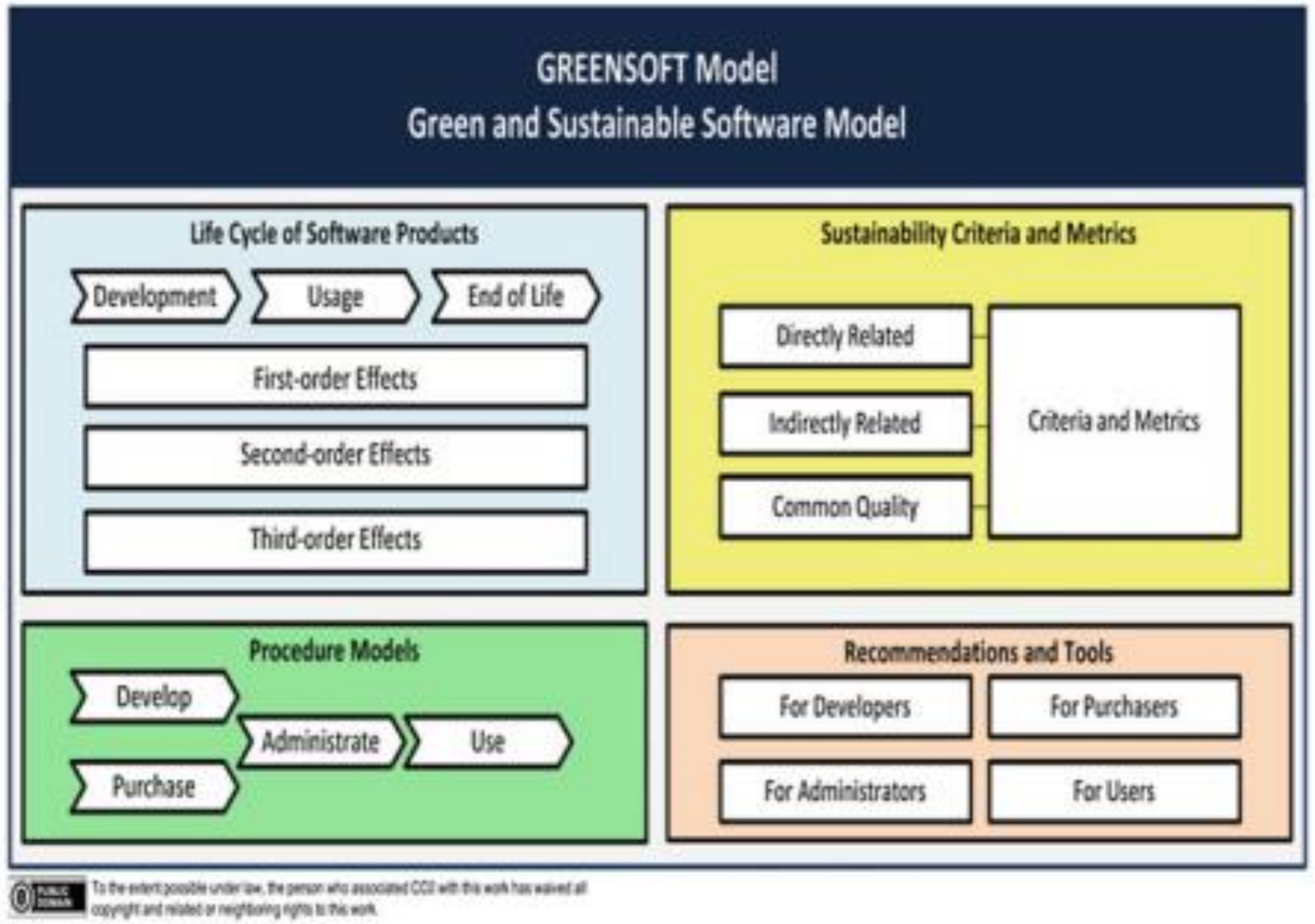


Fig. 3.1 The GREENSOFT reference model [37] (cf. [31])

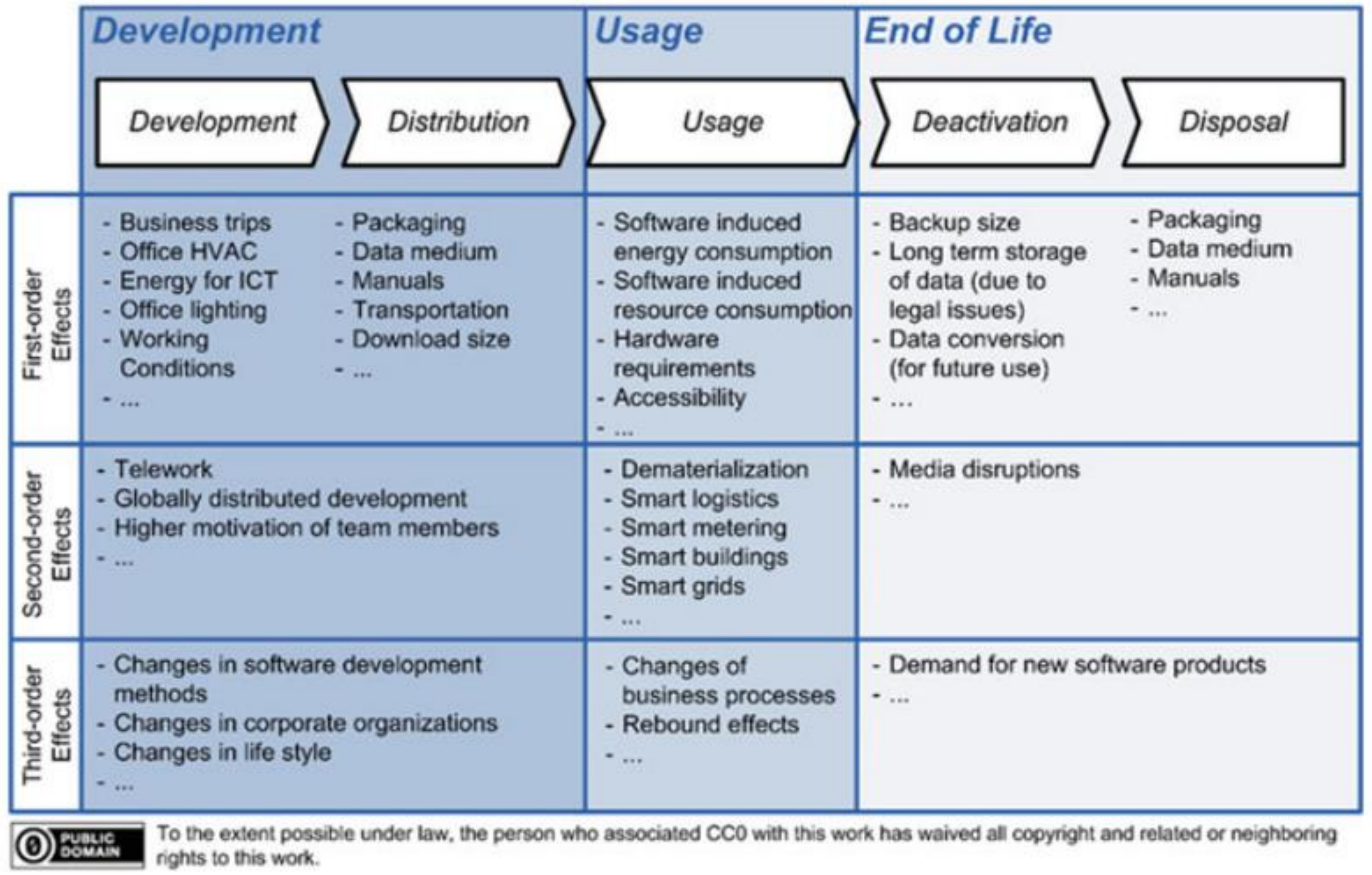
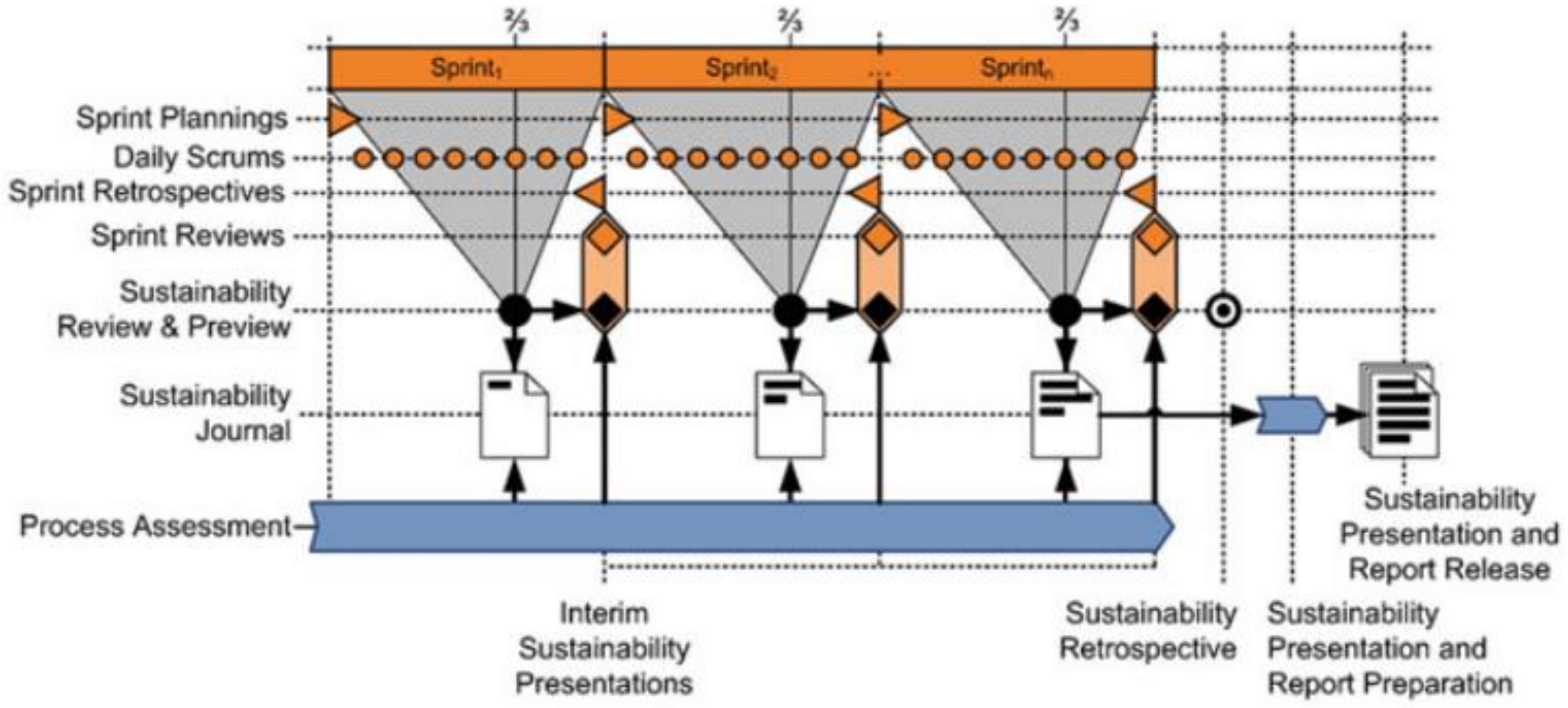


Fig. 3.2 Life cycle for software products, oriented towards life cycle thinking and giving some exemplary effects [37] (cf. [31])



Coral Calero and Mario Piattini. 2015. Green in Software Engineering. Springer Publishing Company, Incorporated. https://doi.org/10.1007/978-3-319-08581-4

SCI Procedure

- **What**: software boundary, i.e. the components of a software system to include.
- **Scale**: carbon emissions per one functional unit, pick the functional unit which best describes how the application scales.
- **How**: quantification method, real-world measurements based on telemetry
- **Quantify**: Calculate a rate, an SCI value, for every software component. The SCI value of the whole application is the sum of the SCI values for every software component in the system.
- **Report**. The SCI has standards for reporting that must be met, including a disclosure of the software boundary and the calculation methodology.