greeNsort®

„Why and how to measure greenness of algorithms“

Presentation at Green Software Foundation

Jens Oehlschlägel, Munich, 28.4.2022

Jens Oehlschägel

Education

Dipl. Psych. specialized on empirical problem solving
Dr. hum. biol. in applied medical statistics
Heterogenous Systems Engineer
McKinsey Mini-MBA

Experience

Programming since 1978, S/R-programming since 1996, learning Julia
Expert Data Scientist at McKinsey 2001-2011
Expert System Architect focusing OLTP and OLAP Database Clusters at Bertelsmann since 2011

***greeNsort®* project**

virtually save a nuclear power station, the technology is developed

Aspiration

full-time research and teaching on energy-efficient algorithms

Private

married, playing piano

greeNsort®

greeNsort® makes software greener
less runtime, energy and CO2
less memory hence better hardware amortization
saves up to 4 nuclear power stations worldwide

How? Better sorting algorithms in central software components
Method: analyzing the prior-art and
designing and testing many innovative algorithms
such that they are sustainable

Result? A a thorny 12-year journey
identifying poisened limiting assumptions
designing a new theory of sorting with
new sustainability measurements
new methods that are simple and beautiful
new algorithms with better trade-offs

greensort.org

github.com/greensort

greeNsort® web of innovations – here heavily simplified

already public
yet confidential

D* *new* ***Definition of Sorting*** *with implications for API structure*

C* *new* ***Machine Model*** *with implications for mathematical evaluation & proof*

C* *new* ***Measurement Model*** *with implications for ordering algorithms with different requirements*

better unstable inplace
("Quicksort dilemma")
better trade-off between
- *Efficiency (random data)*
- *Adaptivity (structured data)*

better stable sorting
(than Mergesort 100% buffer)
better trade-off between
- *Memory (embodied fixed cost)*
- *Efficiency (operating variable cost)*
- *Adaptivity (task difficulty)*

greeNsort®

Disclaimer: my Perspective

Manage future

I am not an accountant, I am not interested in reporting the past. At McKinsey I learned to take a **managerial perspective on decisions as being about different choices for the future.**

Scientific ethos

I am a consultant, data scientist and system architect and want sustainable results. Hence I **measure in controlled experiments, unbiased with minimal error variance and try to optimize with impact**.

Pragmatic action

I am not a theoretician, I am not interested in pseudo-precision. I am a practitioner, and in a partial developer role having worked with more than 12 languages **I need pragmatic solutions.**

We have basic agreement on cost measurement

assigning fixed costs	=	asset costs x share of asset x share of lifetime
M	=	TE x RS x TS
measuring variable costs	=	average operational costs x runtime
Energy E	=	W x h
CO2 O	=	E x I
basis for scaling	=	cost / task
SCI	=	(O+M) / R

greeNsort®

Measuring and Optimizing *independent* factors that affect CO2

pre

C2B
- Demand: Functional and non-functional requirements
- Embodied costs for development, testing, maintenance, …

Green Software Foundation · Microsoft · foss

SW
- **Efficiency of nudging, design, algorithms, languages, compilers, …**
- **Cleverness and Parsimony of hardware requirements**

intel · AMD

HW
- Consumed KWh / Instruction
- Embodied CO2 / Hardware

SDIA SUSTAINABLE DIGITAL INFRASTRUCTURE ALLIANCE · Azure

DC
- Redundancy, Background & Competing Services, Server utilization, PUE, …
- Duration of hardware amortization

LC
- Location-based, time-dependent marginal carbon emissions

post

B2C
- Number of users, usages, … (assigning share of development costs)
- Habits of usage (Carbon-Awareness, toggling feature packages, …)

CO2

greeNsort®

Measuring Software Factors that affect CO2 is hard

today out of scope

C2B

- **harder to know and harder to influence**

software effect

SW

- **influence of Software Engineering**

error variance

HW

- impossible to know and not a feature of software, **but …**

error variance

DC

- impossible to kn… of software, **but …**

strive for
HW DC LC
obliviousness

error variance

LC

- impossible to know and not a feature of software, **but …**

today out of scope

B2C

- **harder to know and harder to influence**

end-to-end CO2 **impossible to know or complicated to measure**

greeNsort®

Do you want to be powered-by renewables? The answer is no.*

Asim Hussain

Why?

- Until less than 100% renewables: someone else uses fossile energy, to avoid greenwashing, we need to operate with I = **maximum marginal** CO2/KWh
- Once 100% renewables: I = zero CO2/KWh, hence an invitation to waste energy

What follows?

Variable costs: reducing CO2 is not a proper goal for sustainable software

- reducing energy is!

Fixed costs: reducing embodied CO2

- by reducing hardware requirements

I is wrong → Standardize I

HW causes CO2 → minimize HW

Complication

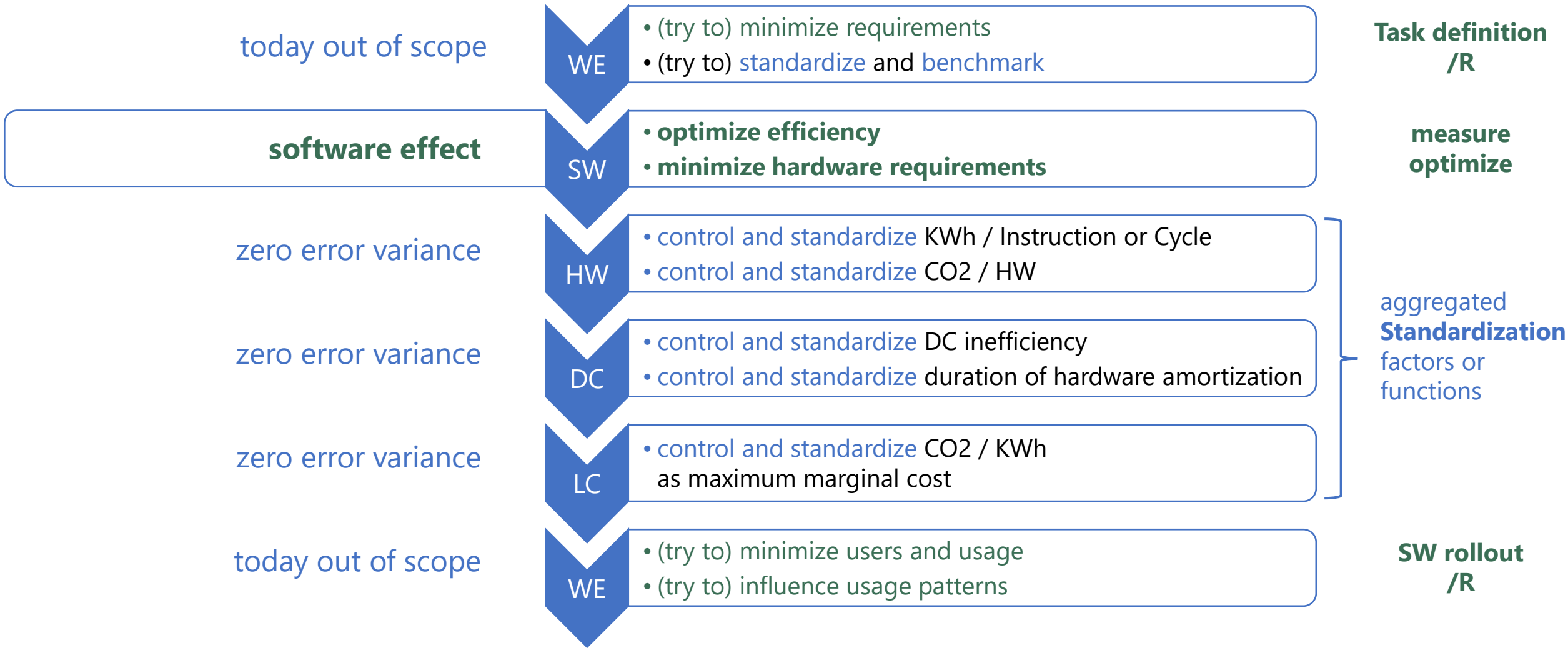
- by choosing architecture with less embodied carbon
- by choosing architecture with less operational energy

} Not independent of HW!
Which common scale?

* https://devblogs.microsoft.com/sustainable-software/do-you-want-to-be-powered-by-renewables-the-answer-is-no/

greeNsort®

Strategy for software factors that affect CO2



greeNsort®

Strategy for Sustainable Algorithms

CPU-parsimony
is key to saving electricity
(reducing variable cost CO2)

RAM-parsimony
is key to using old hardware longer
(reducing fixed cost CO2)

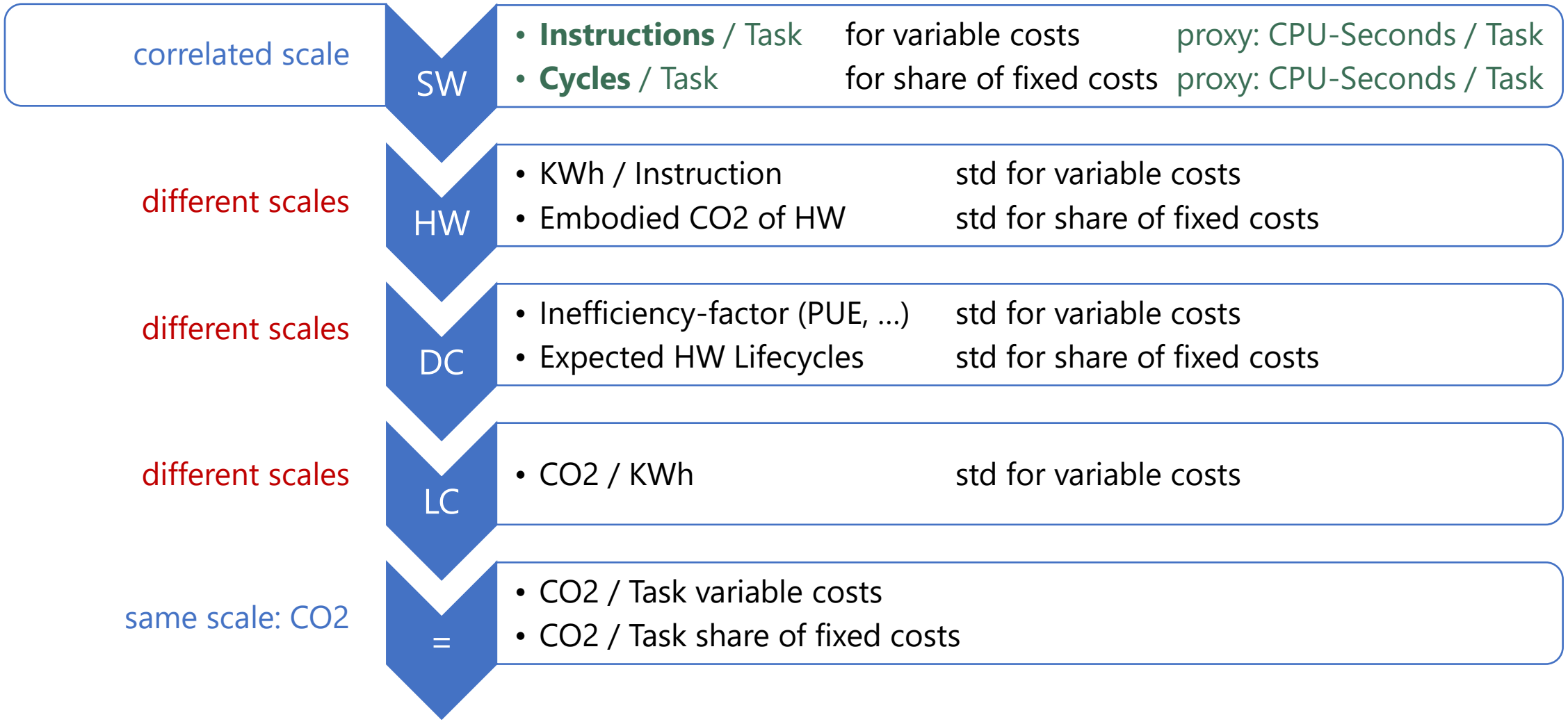
Optimize for both, RAM and CPU,
try to avoid extremes:

- minimum buffer RAM (wasting CPU)
- minimum CPU cycles (wasting RAM)

greeNsort®

Break-down for CPU-cost

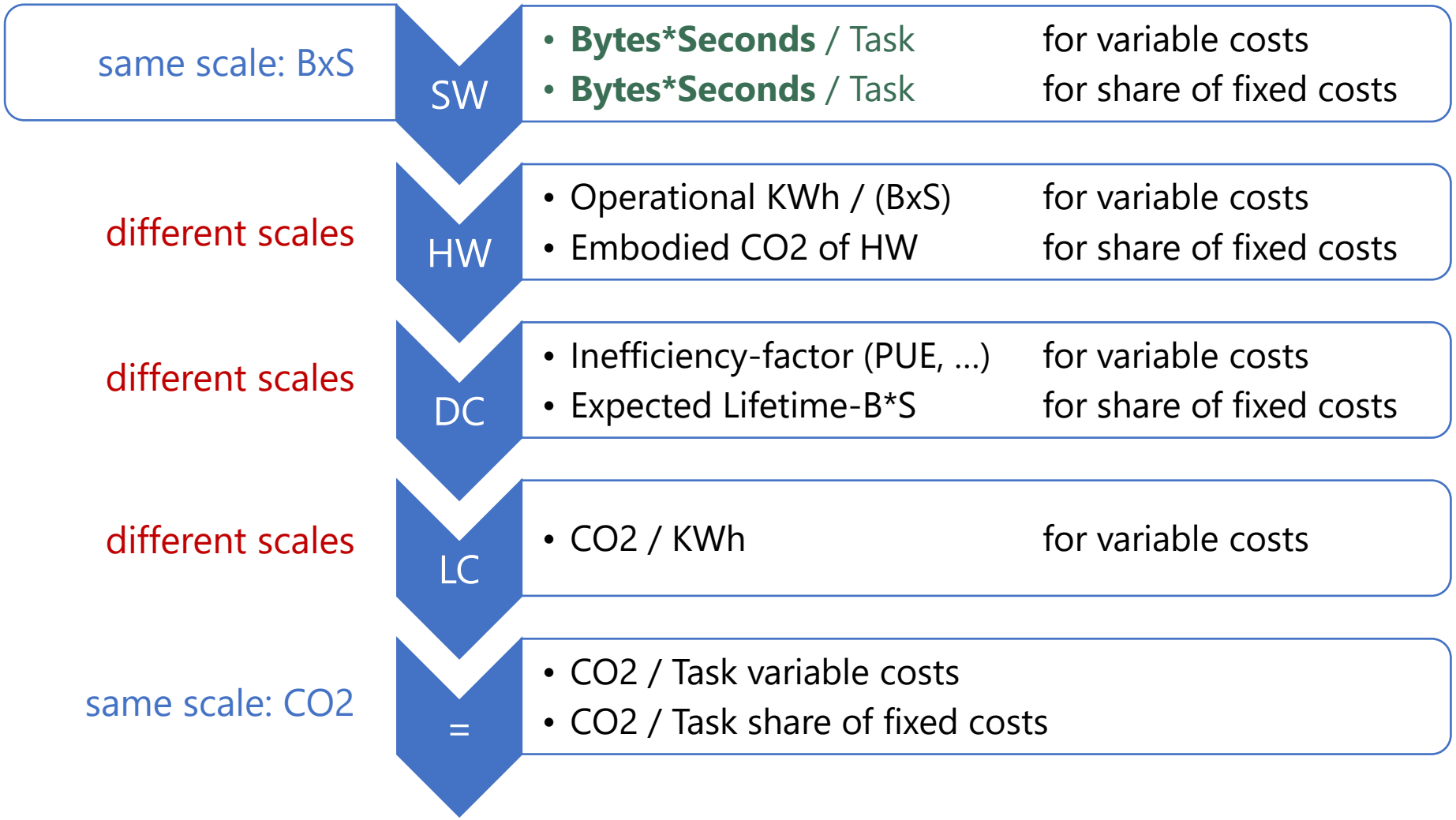
if serial CPU-bound:
just Seconds / Task



= CPU-Seconds * stdCO2/Instruction(CPU) + CPU-Seconds / stdLifetime(CPU) * stdCO2Embodied(CPU)
= CPU-Seconds * (stdCO2/Instruction(CPU) + stdCO2Embodied (CPU) / stdLifetime(CPU))
= CPU-Seconds * stdCO2(CPU) = CPU-Seconds * const = **CPU-Seconds are a good proxy for benchmarking ratios of CPU!**

greeNsort®

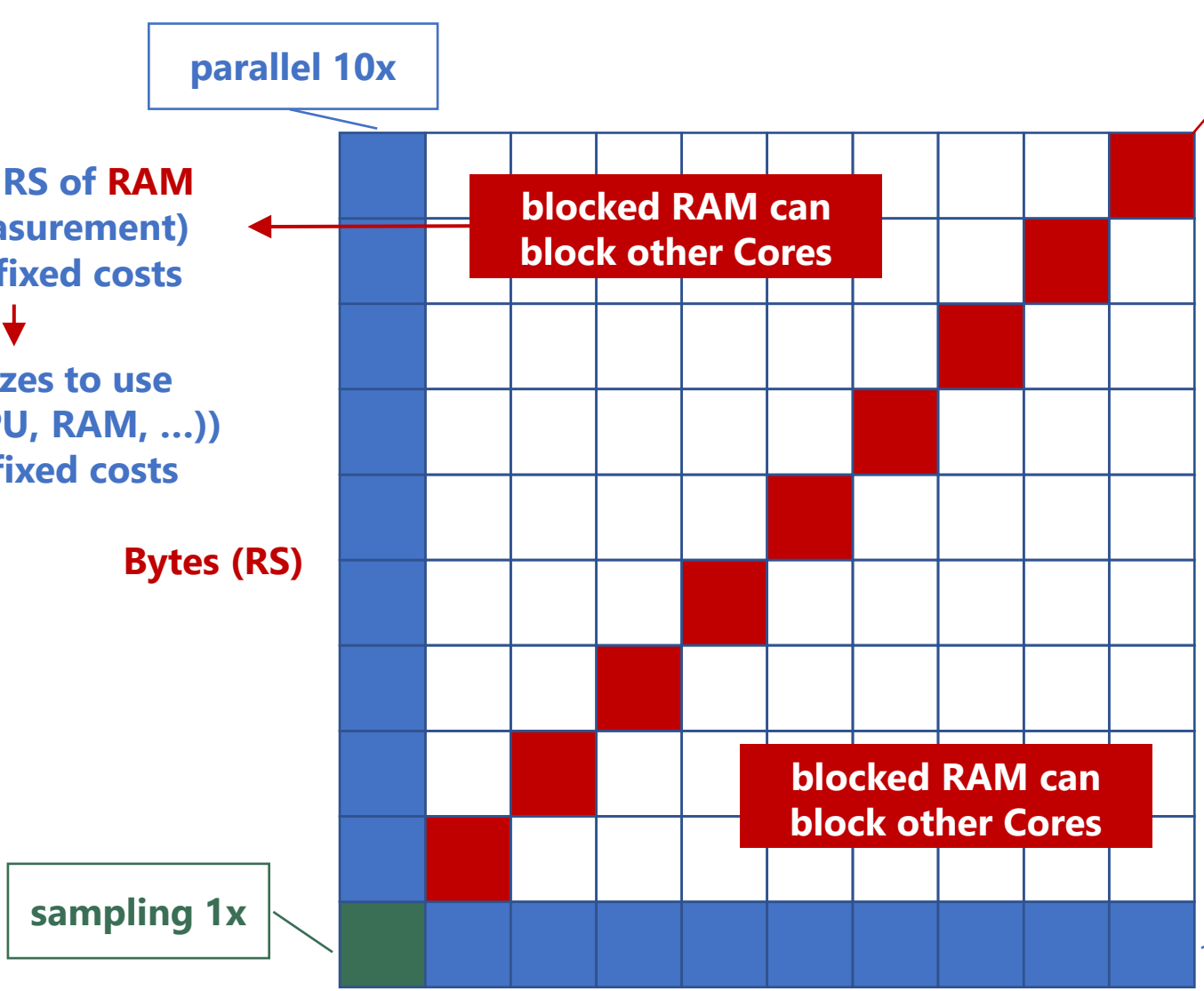
Break-down for RAM-cost is different



= BxS * stdCO2perBxS(RAM) + BxS * (stdCO2EmbodiedperB(RAM) / stdLifeseconds(RAM))
= BxS * (stdCO2perBxS(RAM) + (stdCO2EmbodiedperB(RAM) / stdLifeseconds(RAM)))
= BxS * stdCO2(RAM) = BxS * const = **BxS is a good measurement for benchmarking ratios of RAM!** **? BxS = N^2 ?**

greeNsort®

Multiplying Bytes x Seconds – Scaling 10x bigger data



serial linear scaling
grows share $O(N^2)$

serial $O(N \log(N))$ sorting
grows share $O(N^2 logN)$
with constant access cost

serial $O(N \log(N)^2)$ sorting
grows share $O(N^2 \log(N)^2)$
with logarithmic-access cost

Popular sorting algorithms
- Timsort (Java/Python) is inherently serial
- Quicksort is complicated to parallelize
- Mergesort needs double RAM

greeNsort®

Optimize for what? CPU? RAM? Variable? Fixed? End-to-End CO2?

There is no perfect solution – but we need a single scale!

	CPU	RAM
Fixed costs	Max(Cycles share, Bytes*Seconds share)	Bytes*Seconds share
Variable costs	Cycles	Bytes*Seconds

correlated or identical

correlated

correlated or identical

identical

How to penalize apps with unpredictable CPU usage?

How to penalize apps with unpredictable RAM usage?

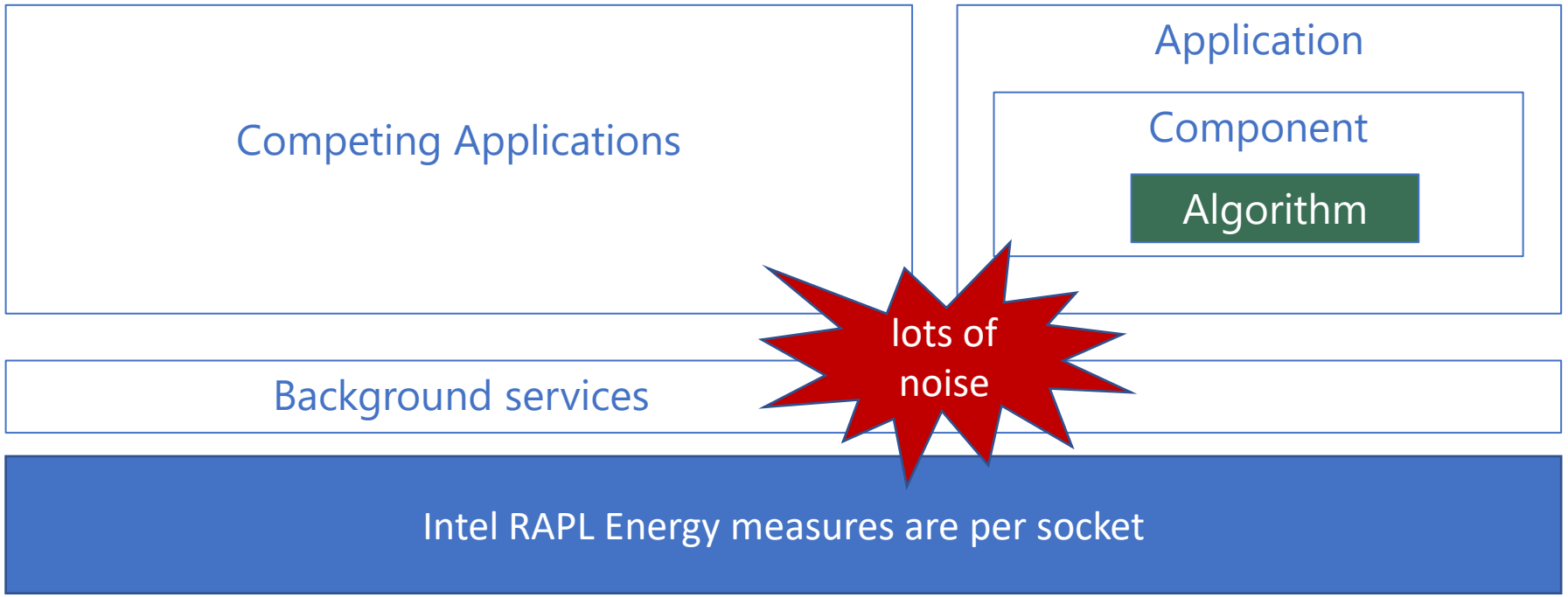
How to penalize apps with random access?

greeNsort®

Measuring and attributing Energy is difficult. End-to-end CO2 even more.
Needed are simple pragmatic measurements!

PAPI instrumentation is complicated and instrumentation might influence performance

perf is simpler but measures only complete applications

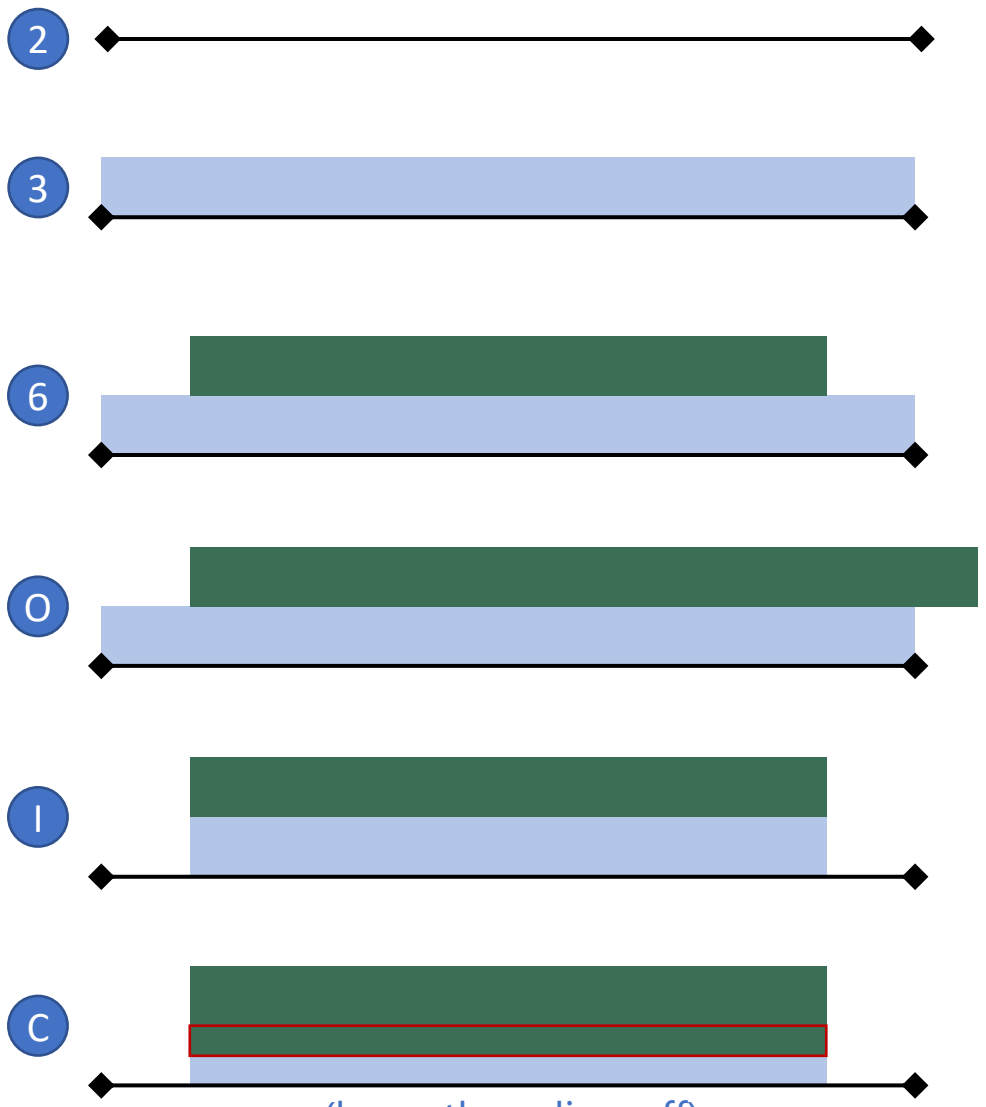


greeNsort®

Even external measurement with perf is complicated and expensive

1. measure algo duration
2. calculate enclosing window
3. baseline window sleep
4. spawn perf for window
5. wait until perf is likely up
6. measure algorithm
7. refine window
8. repeat from 3

B estimate baseline
O mark out-of-window
I substract idle-baseline
C core-share of baseline
A statistical analysis



(hyperthreading off)

multi-seconds for each run to be measurable

$O(R \cdot A \cdot D \cdot N \cdot logN)$

where

$R = 50$ replications
$A = 100$ algoritmns
$D = 30$ data patterns
$N = 2\text{^}24$ data size

~ weeks

$C = \#compiler\ versions$
$H = \#HW\ configuriations$

~ crazy

greeNsort®

Carbon proxies: Measuring the greenness of your application*

"If you can somehow connect what you are measuring to carbon, then it's a proxy. And optimizing for that metric is optimizing for carbon."

Asim Hussain

	correlated with E	Simple and correlated with E and M	correlated with M
state of the prior art for measuring algorithms	eFootprint ***greeNsort®*** cFootprint	***greeNsort®*** tFootprint	embodied part of SCI once I=0 then total SCI
Operation-counters or CPU-Seconds measures CPU-variable costs ignoring RAM (no squared scaling)	**%Bytes*Energy** **%Bytes*CPU-Seconds** measures CPU-variable costs penalized for relat. RAM-costs (no squared scaling)	**%Bytes*Seconds** Proxy for CPU-variable costs penalized for relat. RAM-costs (if same number of cores) (no squared scaling)	**Bytes*Seconds** Proxy for CPU-variable costs penalized for absol. RAM-costs (if same number of cores) Measurement for RAM-costs (squared scaling)

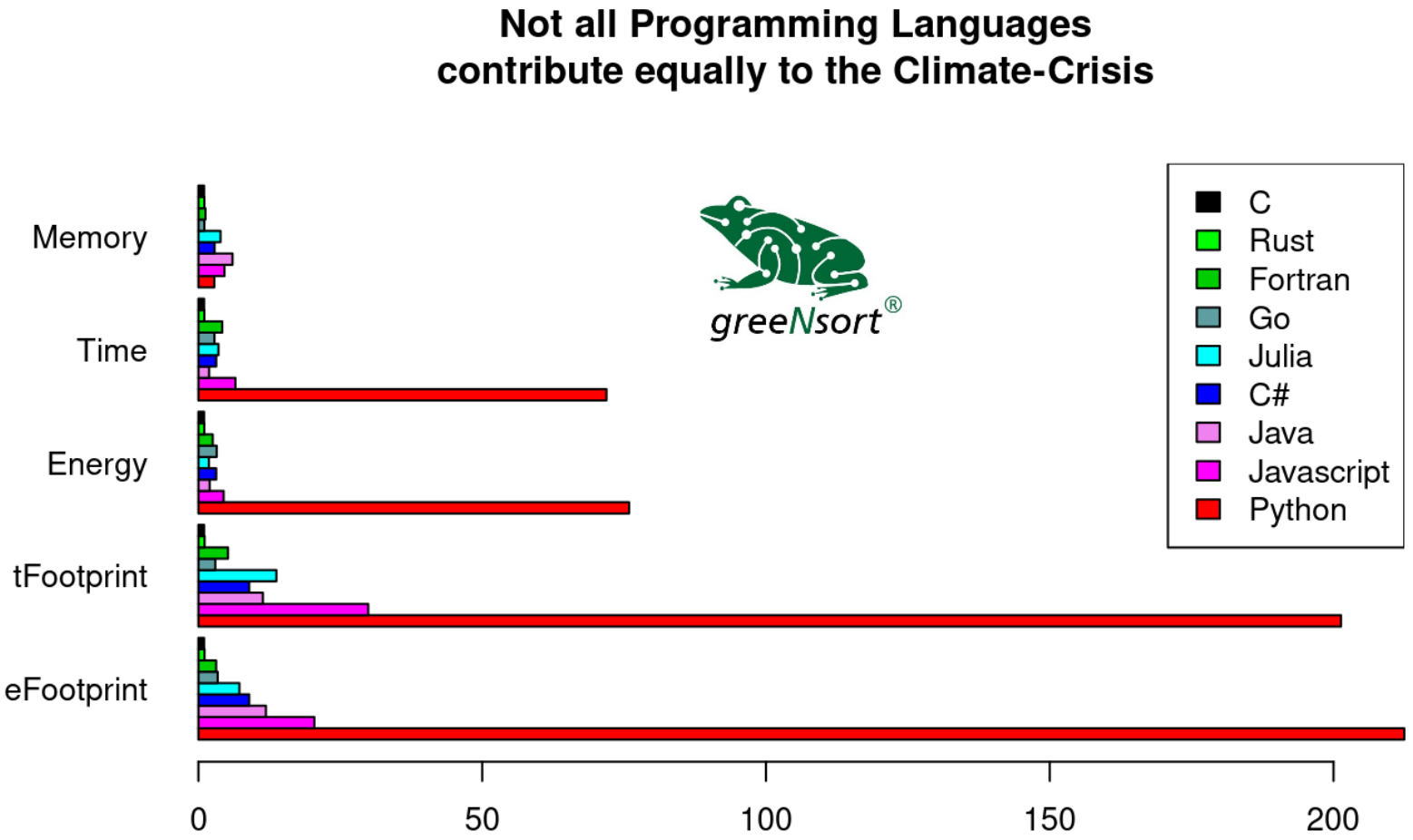
same-weighted*Seconds

* https://devblogs.microsoft.com/sustainable-software/carbon-proxies-measuring-the-greenness-of-your-application/

greeNsort®

Example for the power of the Footprint approach

Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva
"Energy Efficiency across Programming Languages: How does Energy, Time and Memory Relate?"



Now consider all the people doing
Big² Data
Auto³ ML
with
200 · Python !

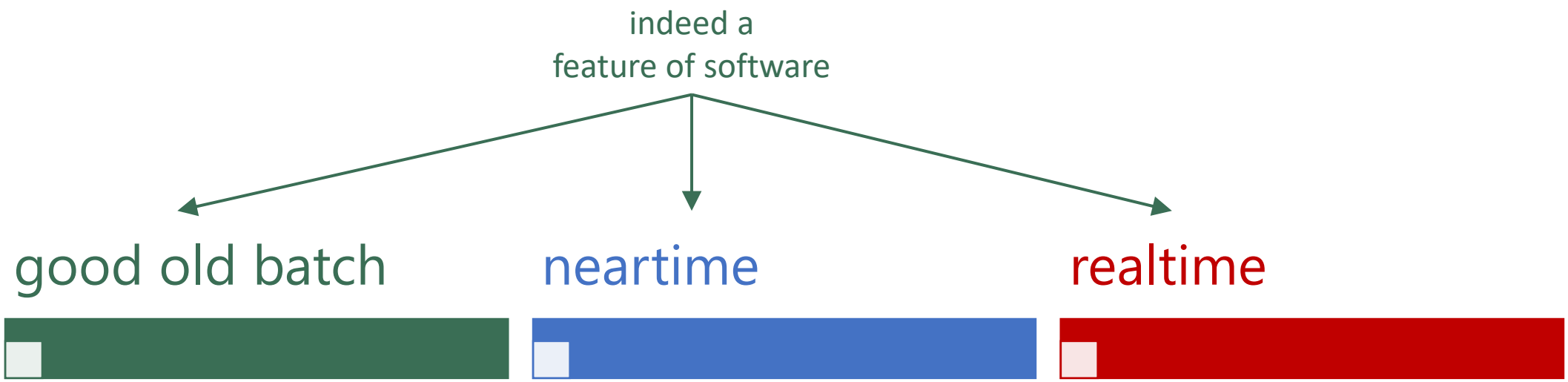
possible cooperation on draft paper about 12 software factors

Result of *greeNsort*®

popular Algorithms	RAM	Parallel?
Quicksort (everywhere)	100% (unstable)	complicated (practically big tasks remain serial)
Timsort (Java & Python)	150%	no (extremely adaptive but even serially inefficient)
Mergesort (everywhere)	200%	possible (little adaptive)

greeNsort Algorithms	RAM	Advantage
Ducksort	100% (unstable)	more efficient and adaptive, very simple
Frogsort	105% .. 150%	quite .. very parallel, adaptive, very simple
Squidsort	105% .. 150%	quite .. very parallel, very adaptive, simpler
Octosort	200%	fully parallel, extremely adaptive, very simple

A final thought: Carbon-Awareness – Solution or Illusion?



- ☐ can delay a lot
- ☐ can collect for macro-batch
- ☐ can shift to optimal time, e.g. solar daytime

- ☐ can delay a bit
- ☐ can collect for micro-batch
- ☐ cannot shift to optimal time

- ☐ cannot delay at all
- ☐ cannot collect for micro-batch
- ☐ cannot shift to optimal time

but: if not using hardware during nighttime this halves lifetime and doubles fixed costs

but: often batch is needed in the night

greeNsort®

Sorry for leaving little time for final discussion – I still give a conclusion from my side

The most accepted common scale is absolute CO2. For evaluation of a minimal (in-memory) software we need four measurements for fixed cost and variable cost of CPU and RAM, if necessary more:

MEASURE	CPU	RAM	GPU, SSD, ...
Fixed costs	CxS Cores*Seconds	BxS Bytes*Seconds	
Variable costs	Wh Energy	BxS Bytes*Seconds	

Then we need for each of the four measurements a GSF standardized default conversion-factor to CO2:

CO2-FACTOR	CPU (core)	RAM (byte)	GPU, SSD, ...
Fixed costs	gCO2/CxS	gCO2/BxS (for fixed)	
Variable costs	gCO2/Wh	gCO2/BxS (for variable)	

Since the Energy-measurement is complicated and expensive, software-engineers might benefit from quick and easy measurement of **proxies** which replace above measurements, namely %Bytes*(CPU)Seconds, as a validation-bridge %Bytes*Wh and for final validation the full CO2 calculation above.

Any helper-tools for the above measurements would be highly welcome. They should be easy to use, scale to large code-bases and many users and ideally be applicable for confidential code. Thank you for the invitation.