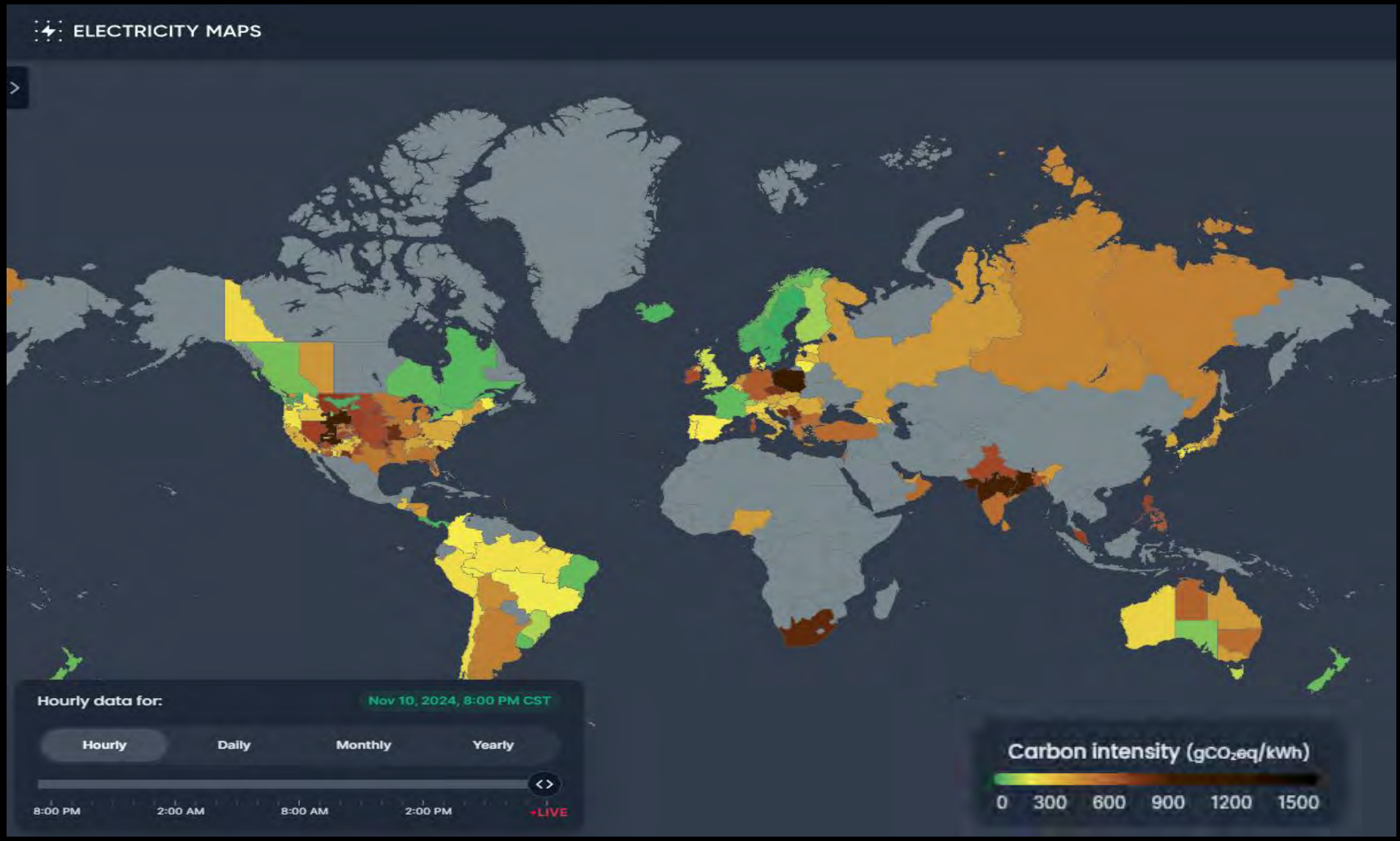
Green Code

or

“Code Green”

Green code refers to environmentally sustainable programming practices focused on reducing energy consumption in software development. This involves writing and optimizing code to improve efficiency, which in turn lowers the energy needed for processing and running applications.

Carbon Intensity - Geographically



What are we trying to accomplish?

Reduce CO2 Footprint

Fundamental Architectural Principles

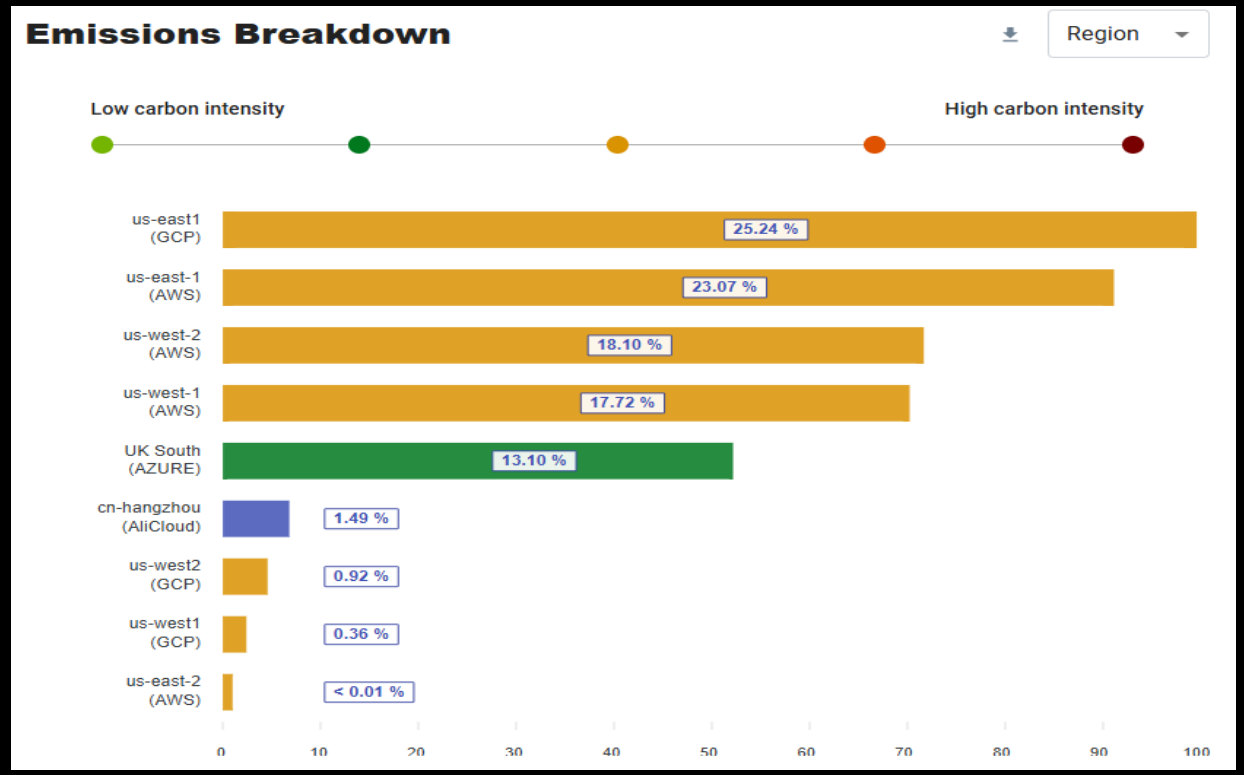
- Application and System Performance
- Security
- Cost $$$
- Reliability
- Availability
- Sustainable Software Development & Operational Practices

Industry Initiatives - Carbon Emission Standards

Green Software Foundation

Cloud Carbon Footprint

- Tools and Frameworks
  - Impact Framework
  - Carbon Aware SDK
- Specification
  - Software Carbon Intensity (SCI)
    - SCI = C per R or ((E*L) + M)/R
      - E - Energy consumed by software system
      - L – Location based marginal carbon intensity
      - M –Embodied emissions of the hardware needed to operate a software system
      - R – Functional Unit (Calling an API)



https://github.com/Green-Software-Foundation/awesome-green-software#general-purpose

Fastest Programming Language - Most Efficient?

Benchmark	Description	Input
regex-redux	Match DNA 8mers and substitute magic patterns	fasta output
binary-trees	Allocate, traverse and deallocate many binary trees	21

Language Type	Description
Compiled	Code is translated into machine code before execution
VM (Virtual Machine)	A software environment that simulates a computer, allowing code to run regardless of the underlying hardware
Interpreted	Code is translated line by line at runtime

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

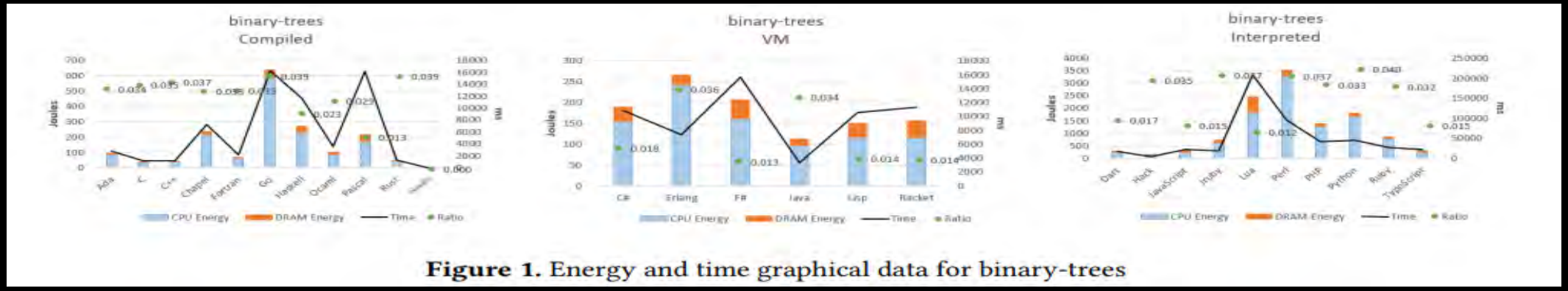
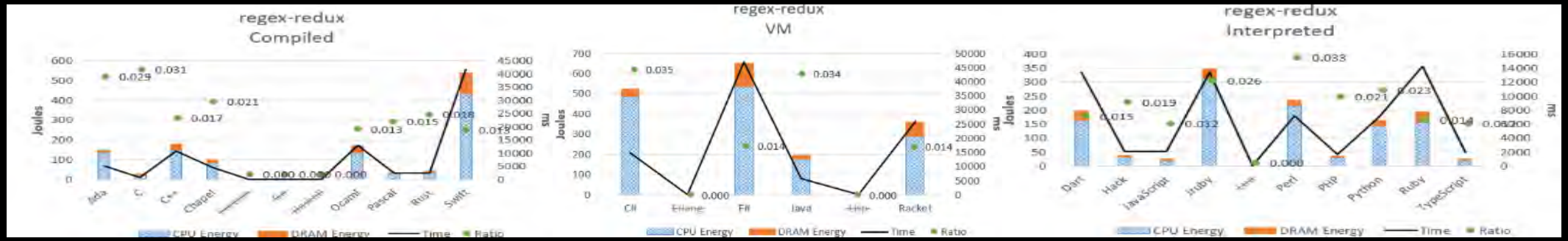


Figure 1. Energy and time graphical data for binary-trees



***Regex Redux* Benchmark**

Energy Efficiency Across Programming Languages

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

Normalized global results for Energy, Time, and Memory

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

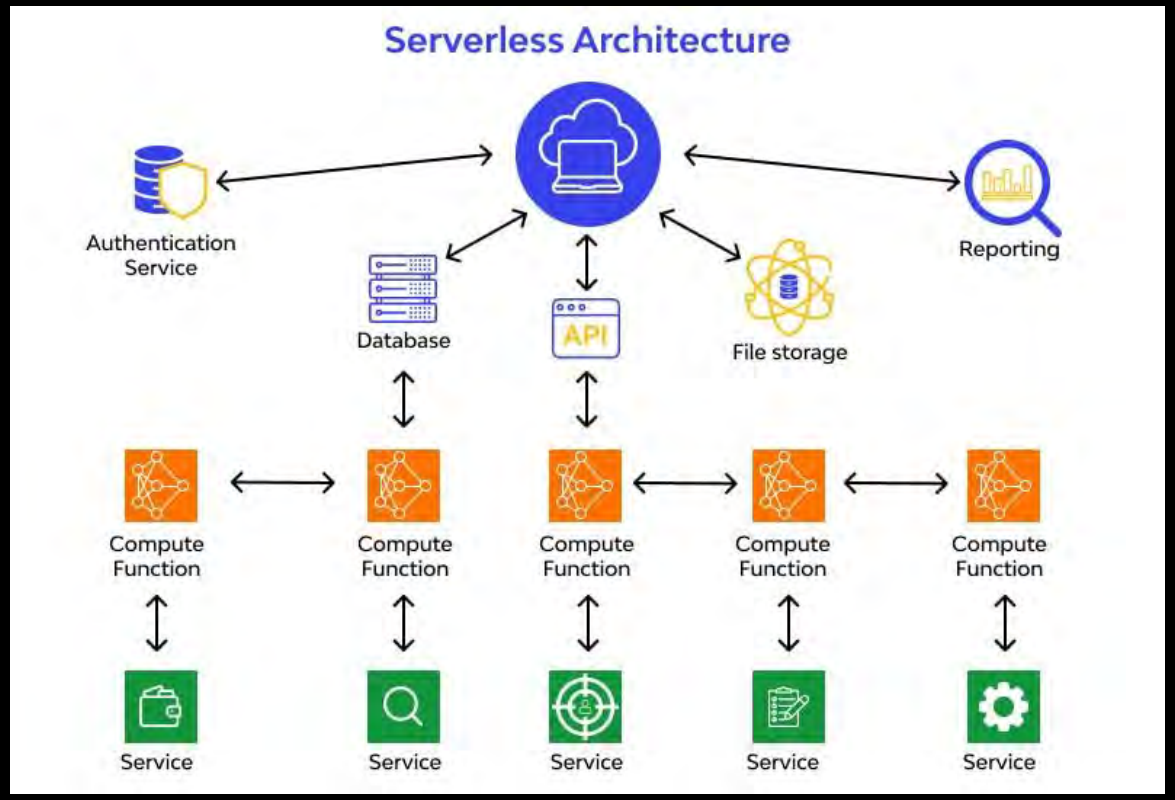
	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

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

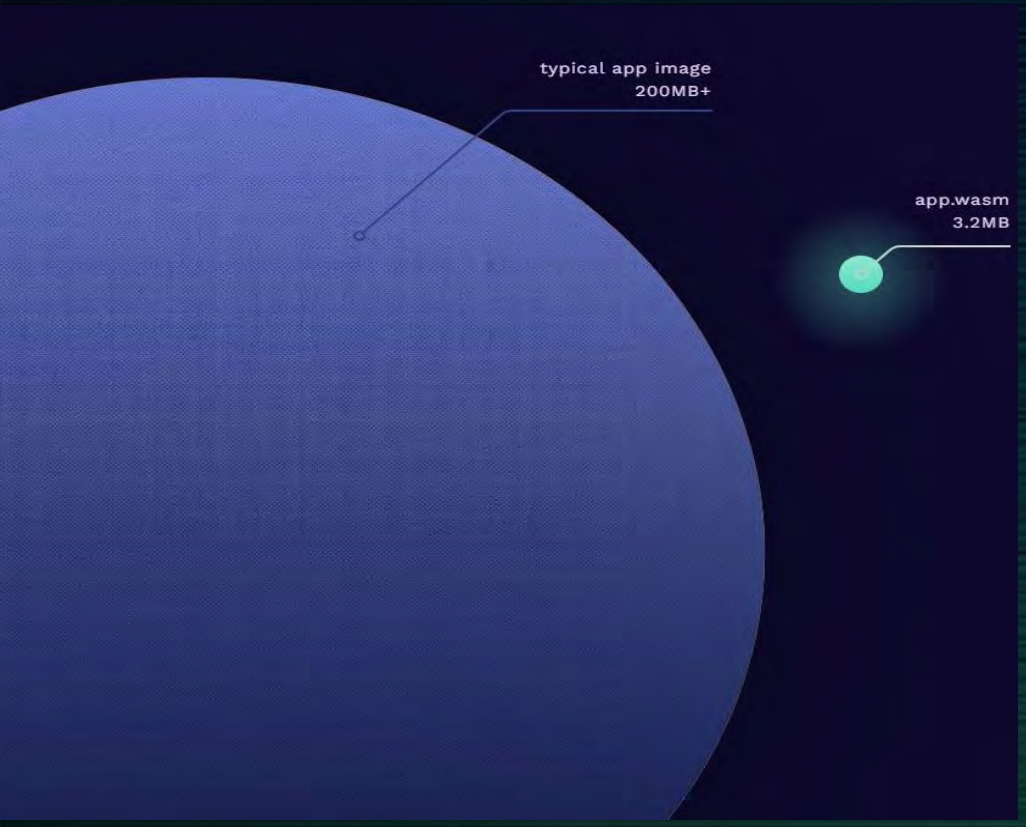
Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	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		

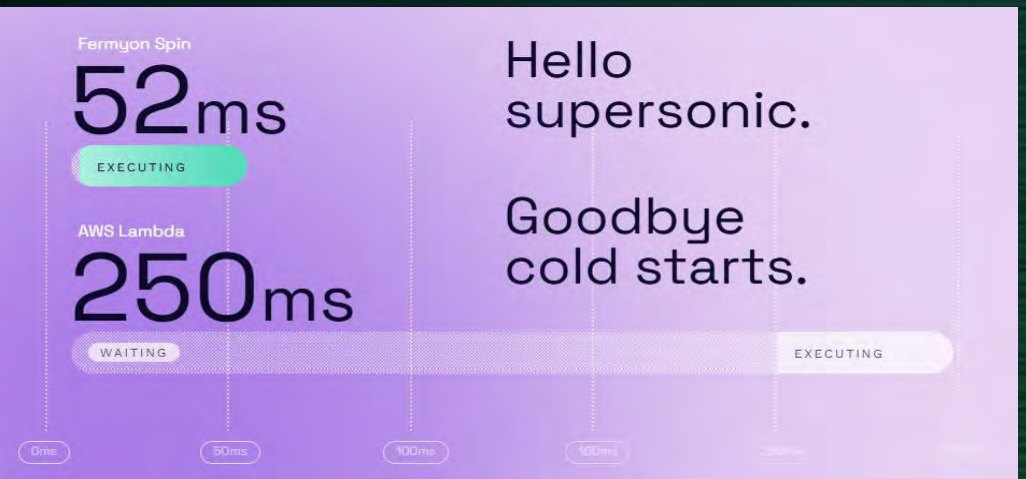
Possible Carbon Intensity Reduction Solution - Serverless



- ✓ **Containerization**
- ✓ **Event –Driven**
- ✓ **MicroVMs**
- ✓ **Custom Runtime Environments**
- ✓ **Performance**
- ✓ **Managed**

Possible Carbon Intensity Reduction Solution - WASM





Portability: Wasm is designed to run on any platform that supports the WebAssembly runtime, making it highly portable and ideal for cross-platform applications.

Performance: Wasm runs at near-native speed, which is crucial for performance-sensitive applications, especially on edge devices with limited resources.

Security: Wasm provides a secure execution environment by running code in a sandboxed environment, which helps protect against malicious code.

Resource Efficiency: Wasm's compact binary format allows for efficient transmission and execution of code, reducing bandwidth and storage requirements.

Possible Carbon Intensity Reduction Solution - GraalVM

Polyglot Capabilities: GraalVM supports multiple programming languages, including Java, JavaScript, Ruby, Python, and LLVM-based languages. This makes it a versatile tool for developers working with different languages.

Performance: GraalVM offers advanced Just-In-Time (JIT) compilation and Ahead-Of-Time (AOT) compilation, which can significantly improve the performance of applications.

Native Image: GraalVM's Native Image feature allows for the creation of small, self-contained binaries that start up quickly and use less memory, which is beneficial for reducing energy consumption and carbon emissions.

Interoperability: GraalVM enables seamless interoperability between different languages, allowing developers to use the best tools and libraries for their specific needs.

Effective DevOps Strategies

- ✓ **Efficient Resource Utilization**
- ✓ **Cloud Migration**
- ✓ **Green Software Practices**
- ✓ **Monitoring and Reporting**
- ✓ **Collaboration and Innovation**

References

Background Image credit: Oseloka Obiora, RiverSafe
https://app.electricitymaps.com/map
https://greensoftware.foundation
https://sci.greensoftware.foundation
https://demo.cloudcarbonfootprint.org
https://github.com/Green-Software-Foundation/awesome-green-software#general-purpose
https://sites.google.com/view/energy-efficiency-languages/results
https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf
https://www.fermyon.com/spin
https://www.graalvm.org
https://www.datacenterdynamics.com/en/news/only-13-of-provisioned-cpus-and-20-of-memory-utilized-in-cloud-computing-report
https://squaredup.com/dashboard-gallery/scom-server-monitoring-dashboard