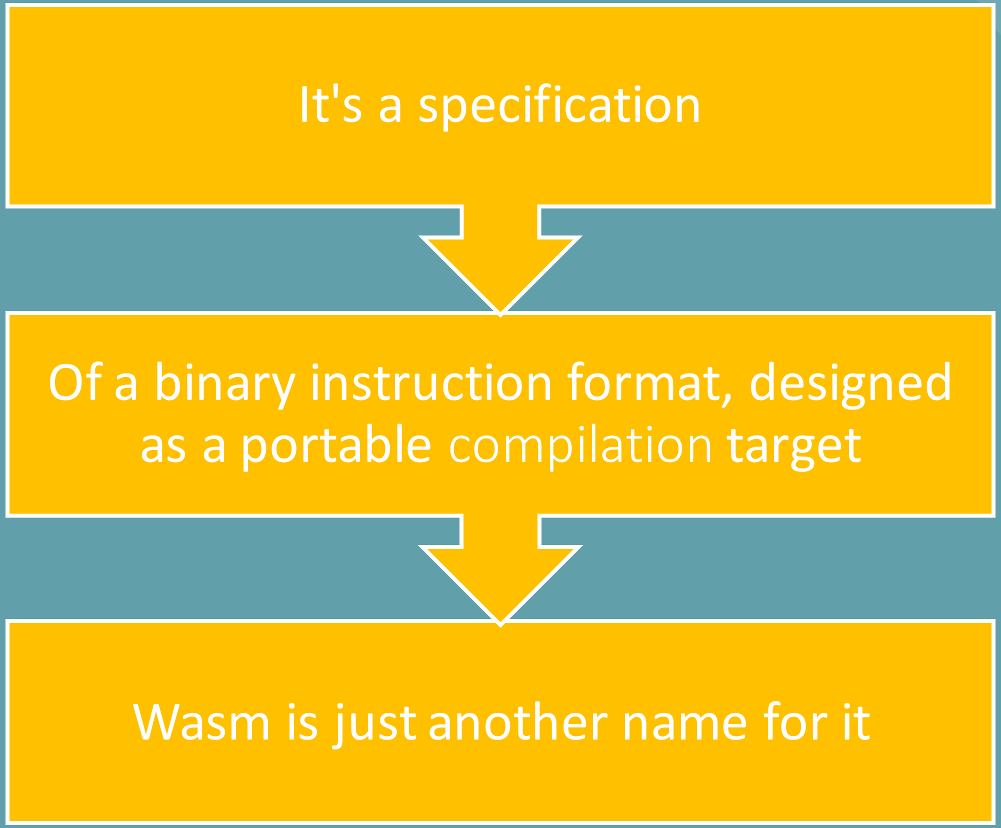
# Evaluating WASM for Computation Pushdown

Ashutosh Parida, Varun Kaundinya,
Suhas Hebbar, Sadman Sakib

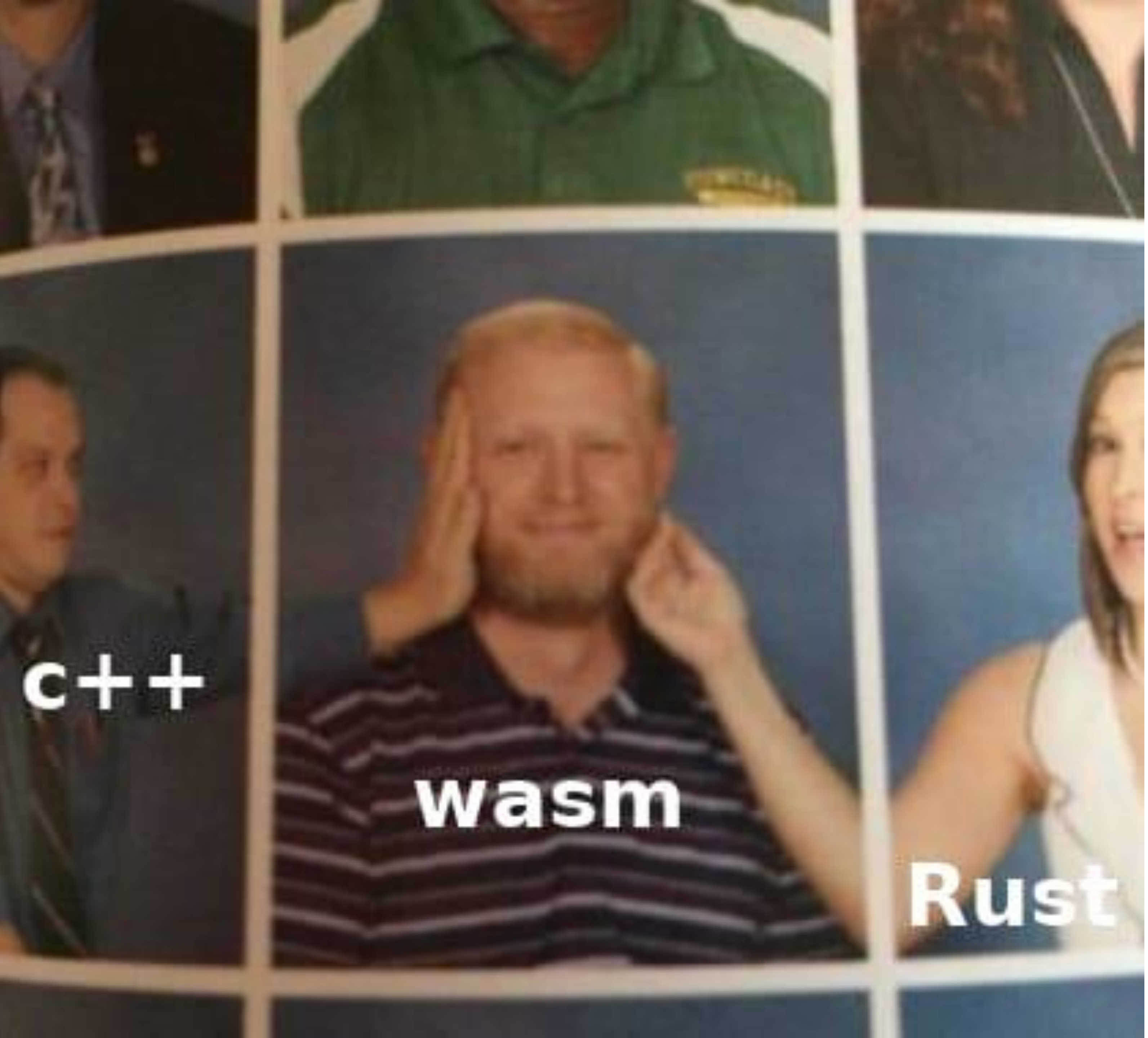
# What's the 3 most important things to know about WebAssembly ?



It's a specification

Of a binary instruction format, designed as a portable compilation target

Wasm is just another name for it



# Wasm – whaaat ?

Compilation target: It is not something you write by hand, but something you compile to

Language agnostic: You can compile C, C++, Rust, Go, Ruby, C#, .NET or Zig to Wasm, just to name a few

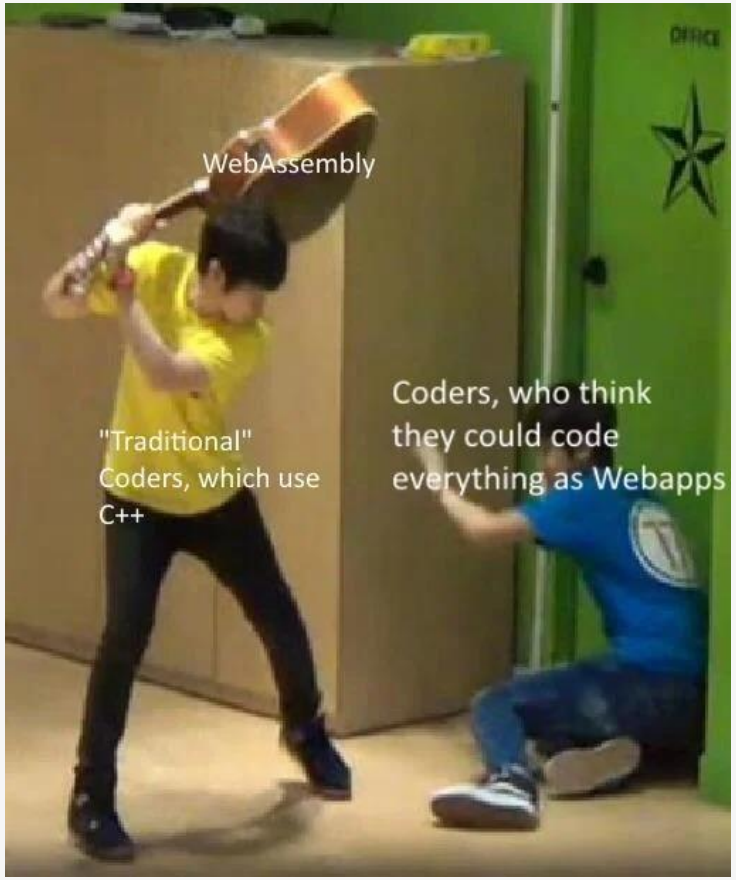
Binary bytecode: The same file can run unmodified in any CPU (your Windows x86 PC, or your Apple MI laptop)

# Born for the web

Security: Secure sandbox by default, has to be able to run untrusted code

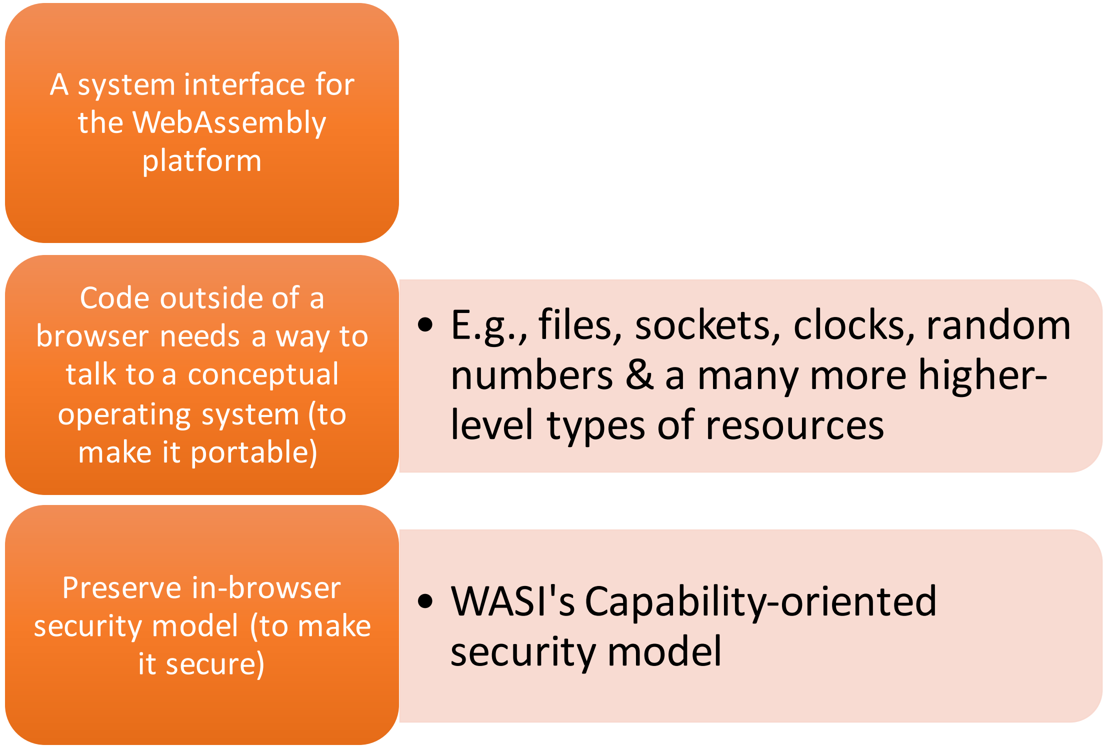
Performance: It can directly execute a compiled version of a language on the system

Streaming Execution: Starts working as soon as it downloads the first byte


WebAssembly
"Traditional" Coders, which use C++
Coders, who think they could code everything as Webapps

# Hey, I want the same on the cloud!

# WASI



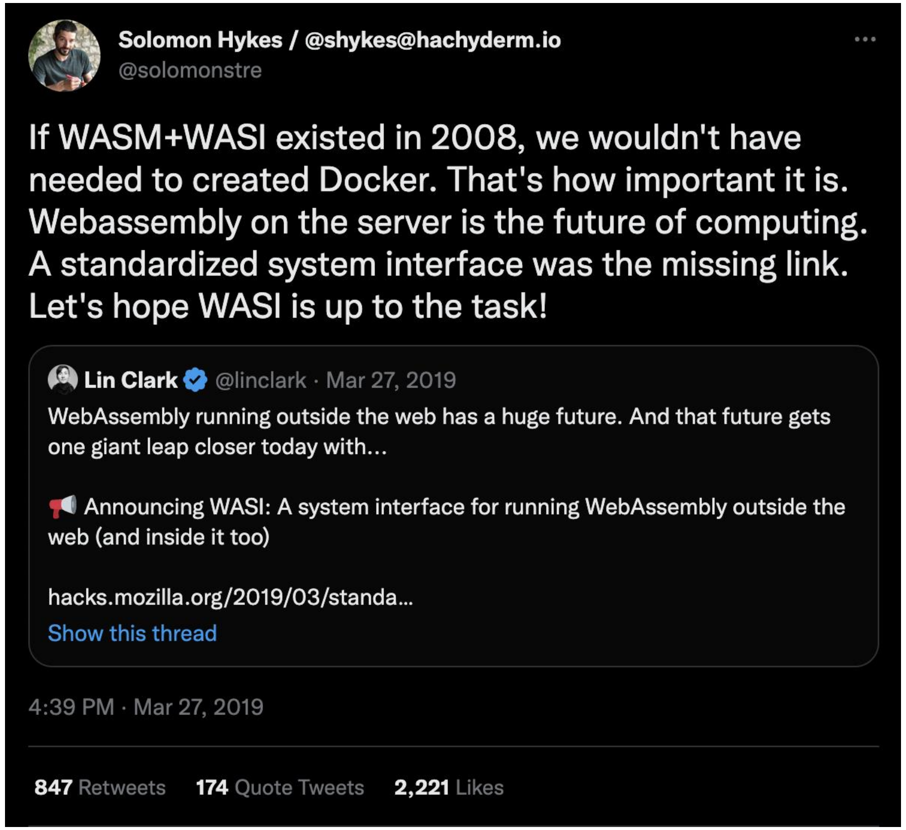
A system interface for the WebAssembly platform

Code outside of a browser needs a way to talk to a conceptual operating system (to make it portable)

- E.g., files, sockets, clocks, random numbers & a many more higher-level types of resources

Preserve in-browser security model (to make it secure)

- WASI's Capability-oriented security model



**Solomon Hykes / @shykes@hachyderm.io**
@solomonstre

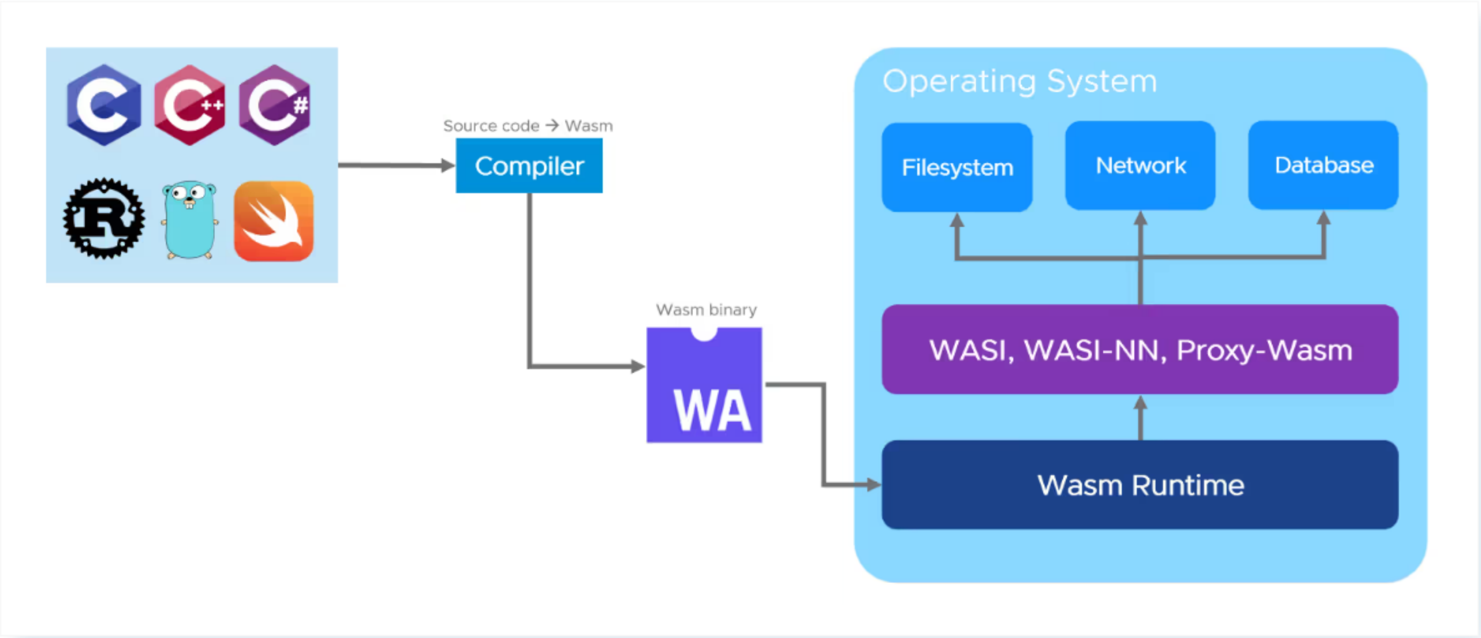
If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

> **Lin Clark** @linclark · Mar 27, 2019
> WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with…
>
> 📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)
>
> hacks.mozilla.org/2019/03/standa…
> Show this thread

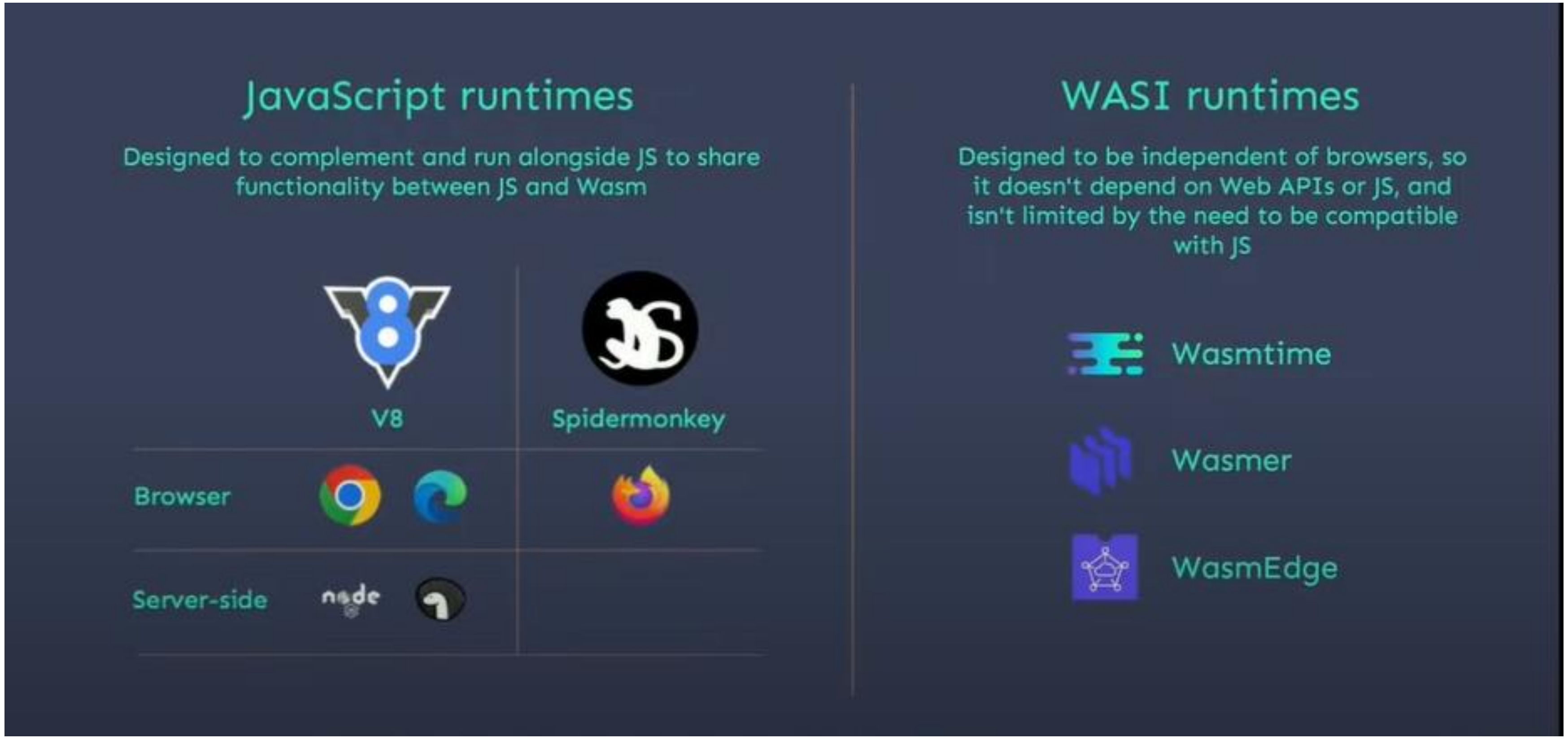
4:39 PM · Mar 27, 2019

**847** Retweets **174** Quote Tweets **2,221** Likes

# Compile and Run



# Runtimes and VMs



## JavaScript runtimes

Designed to complement and run alongside JS to share functionality between JS and Wasm

|  | V8 | Spidermonkey |
|---|---|---|
| Browser |  |  |
| Server-side | node |  |

## WASI runtimes

Designed to be independent of browsers, so it doesn't depend on Web APIs or JS, and isn't limited by the need to be compatible with JS

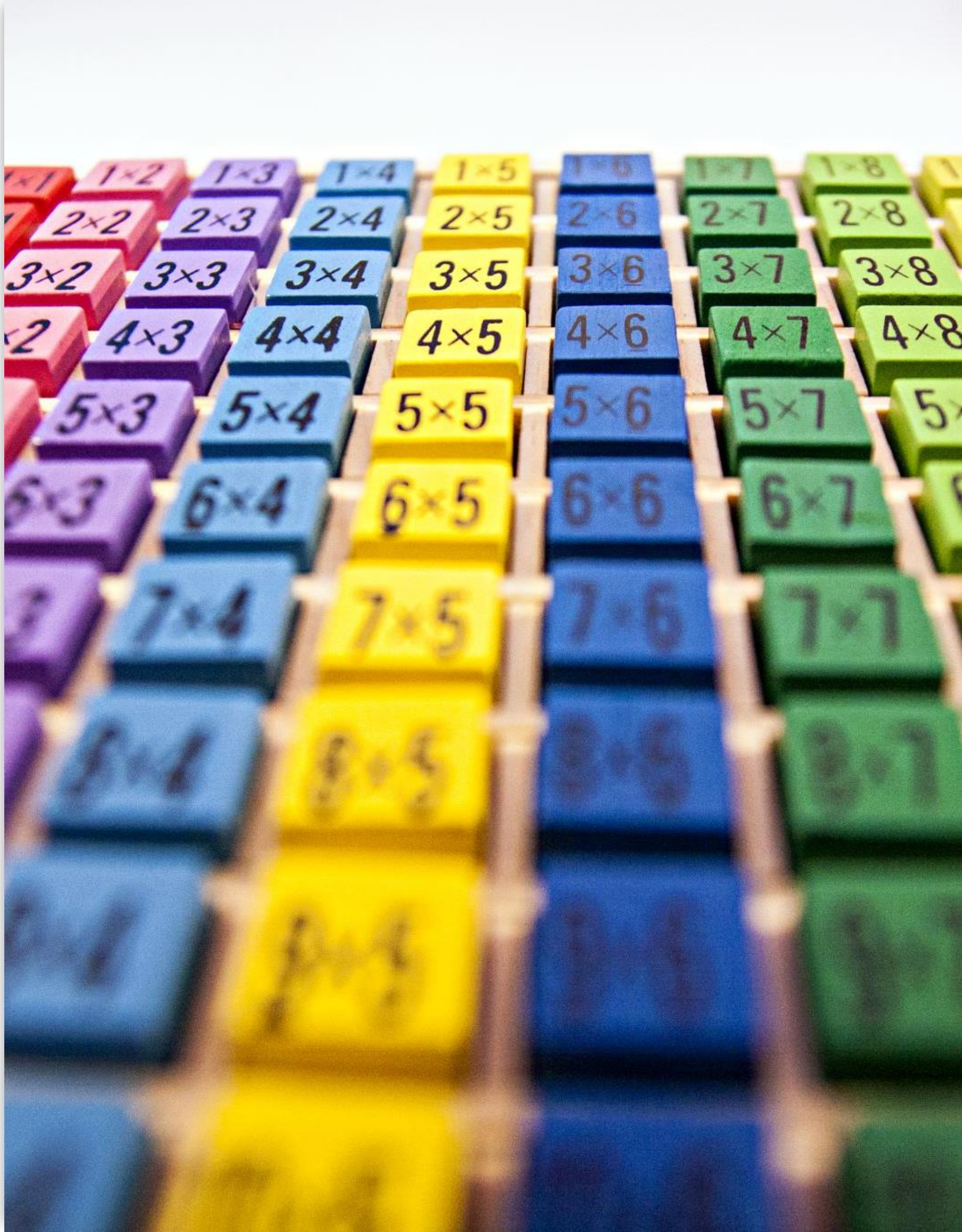
- Wasmtime
- Wasmer
- WasmEdge

# Wasm in Cloud Databases

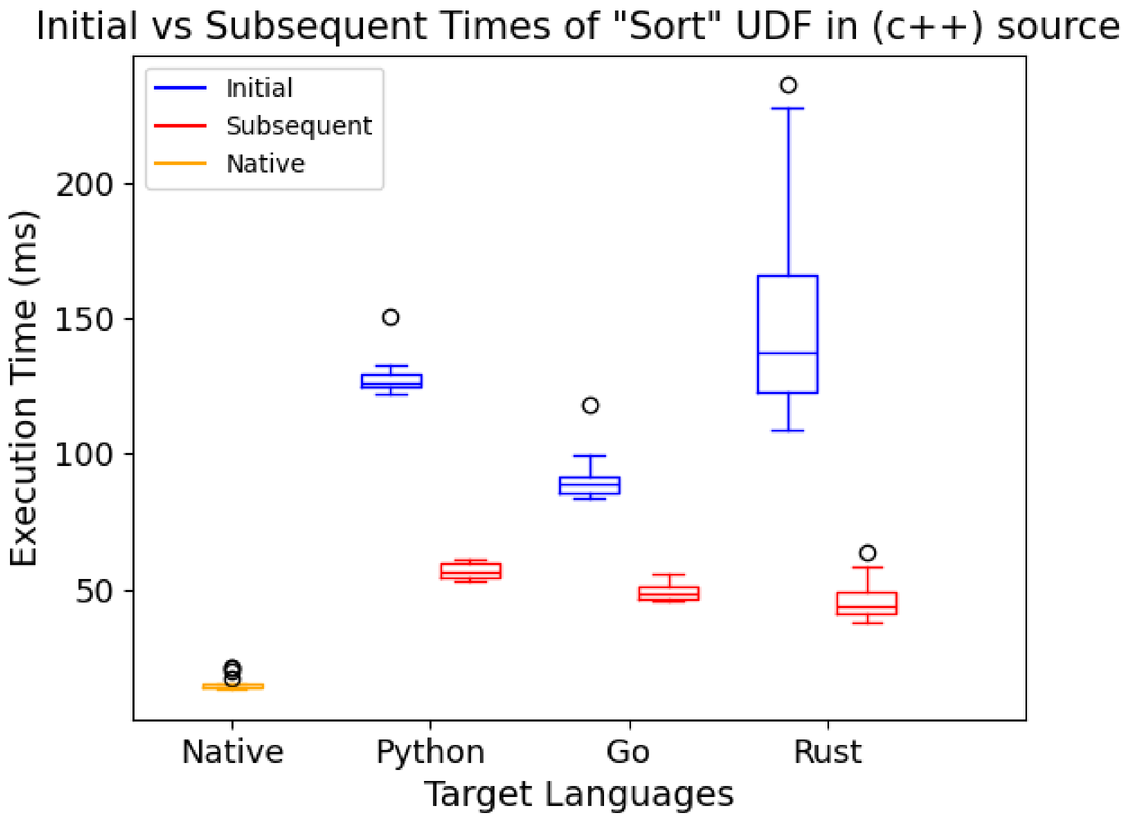
- Many DBs are extensible with non-SQL user-defined functions (UDFs).
- Need a multi-language, platform-independent, lightweight, secure way for UDF execution (docker, micro-VMs and Wasm)
- Wasm deployments: TiDB UDF Engine, SingleStore Code Engine
- Can also be used for serverless functions

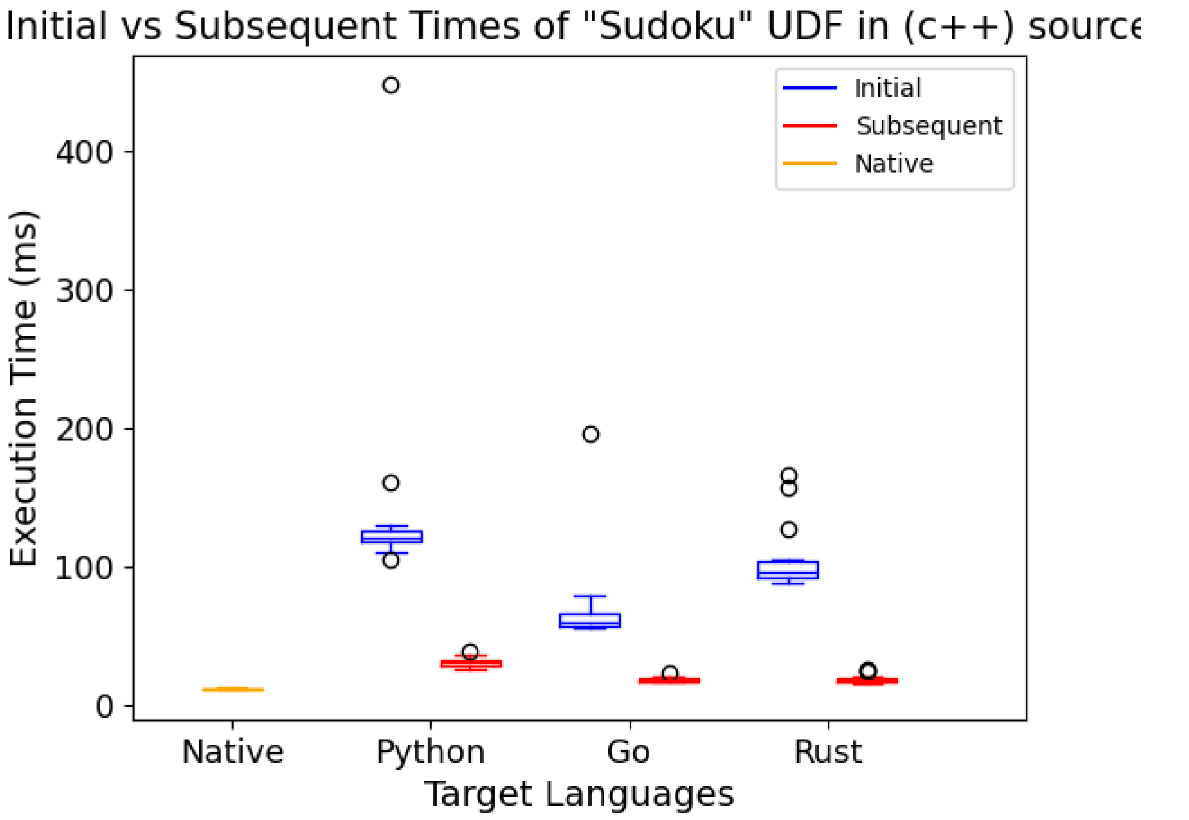
# Experimental Setup

- All benchmarks run on 130G DRAM cloudlab servers.
- Project and Filter UDFs on 10K records in parquet format.
- UDF implementation in different source languages compiled to Wasm and run against Wasm runtime libraries through bindings in different target languages.



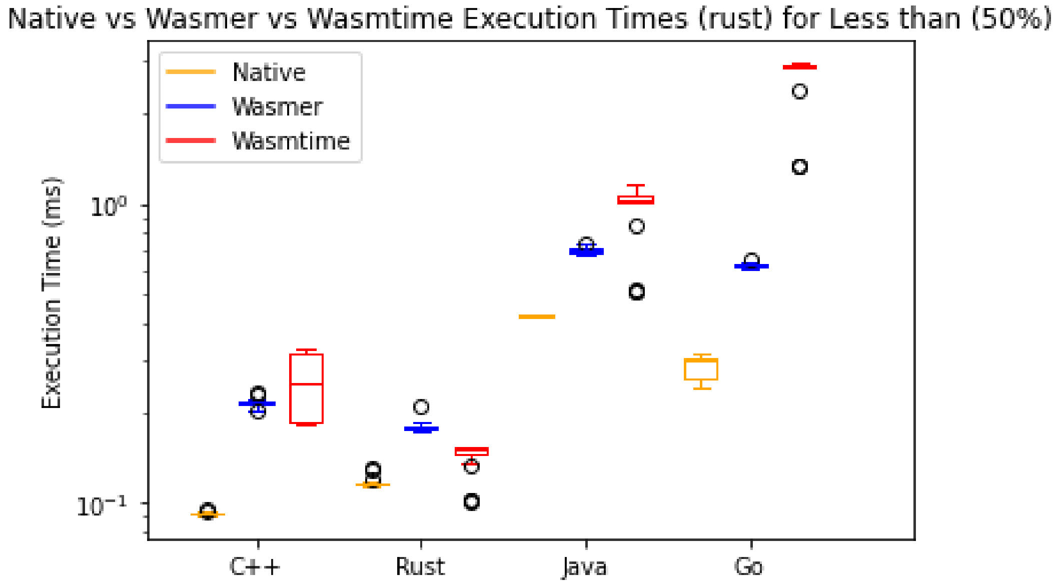
# Native vs Initial vs Subsequent: Langbench

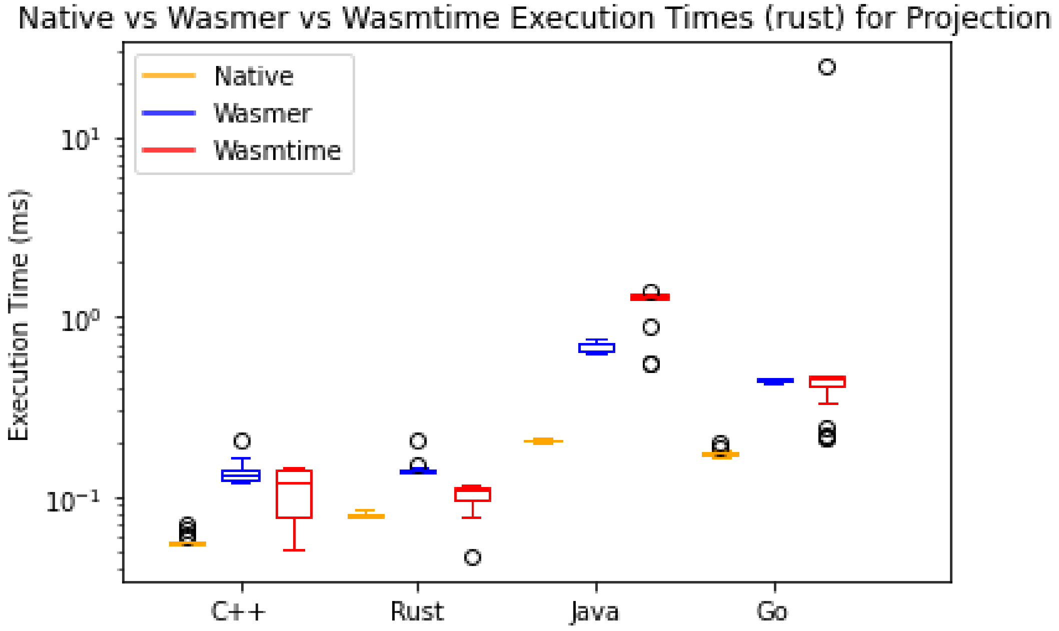




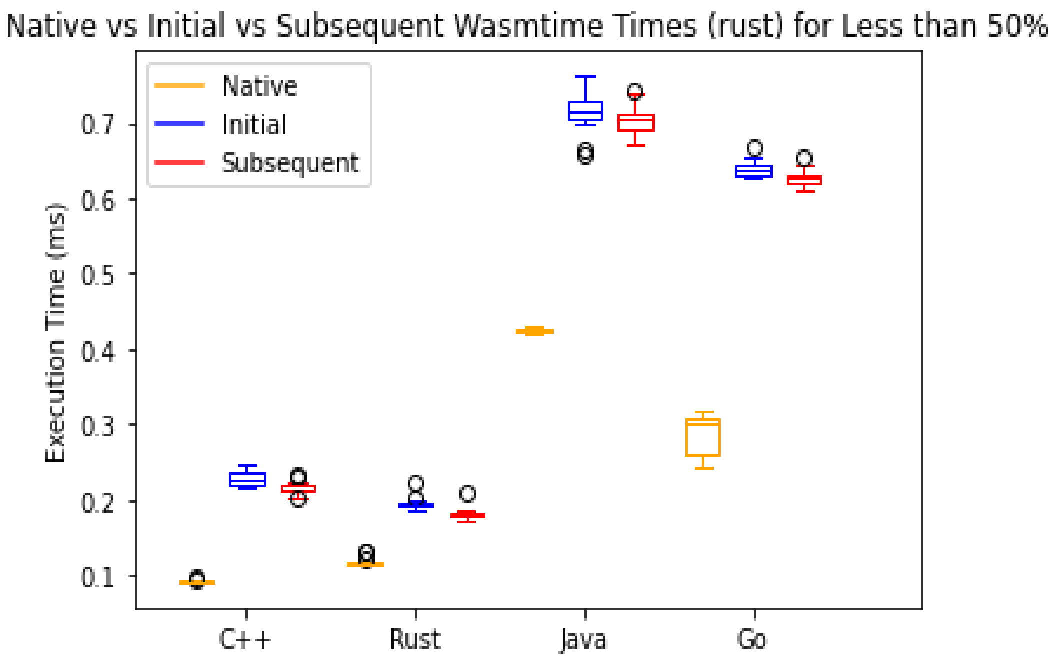
https://github.com/dsrg-uoft/LangBench

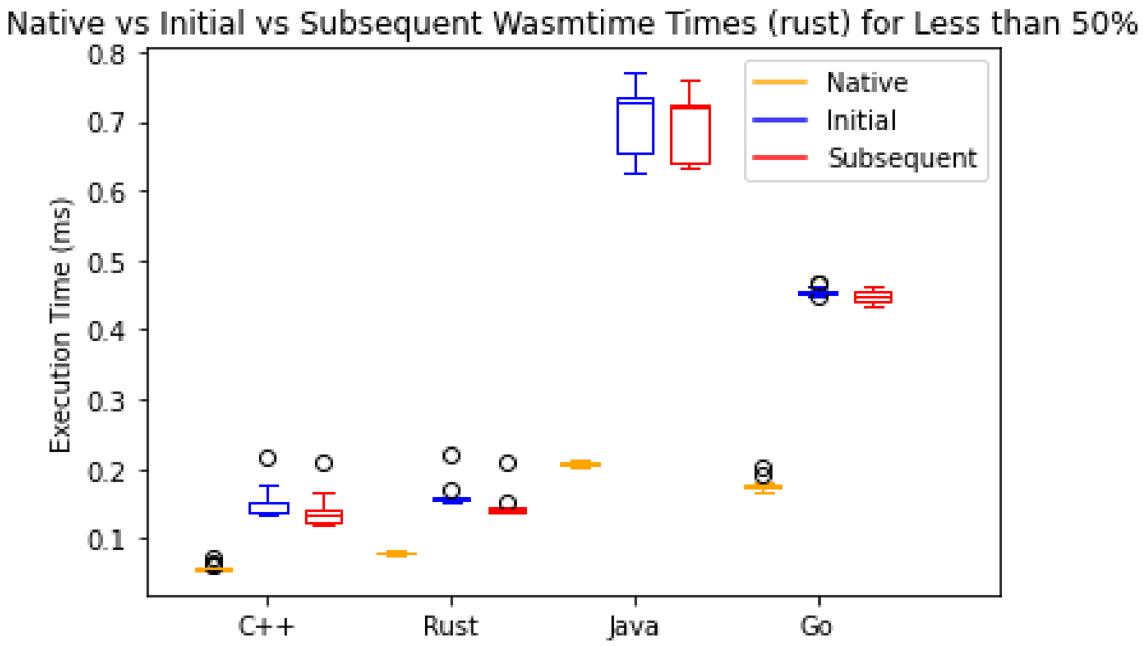
# Native vs Wasmtime vs Wasmer





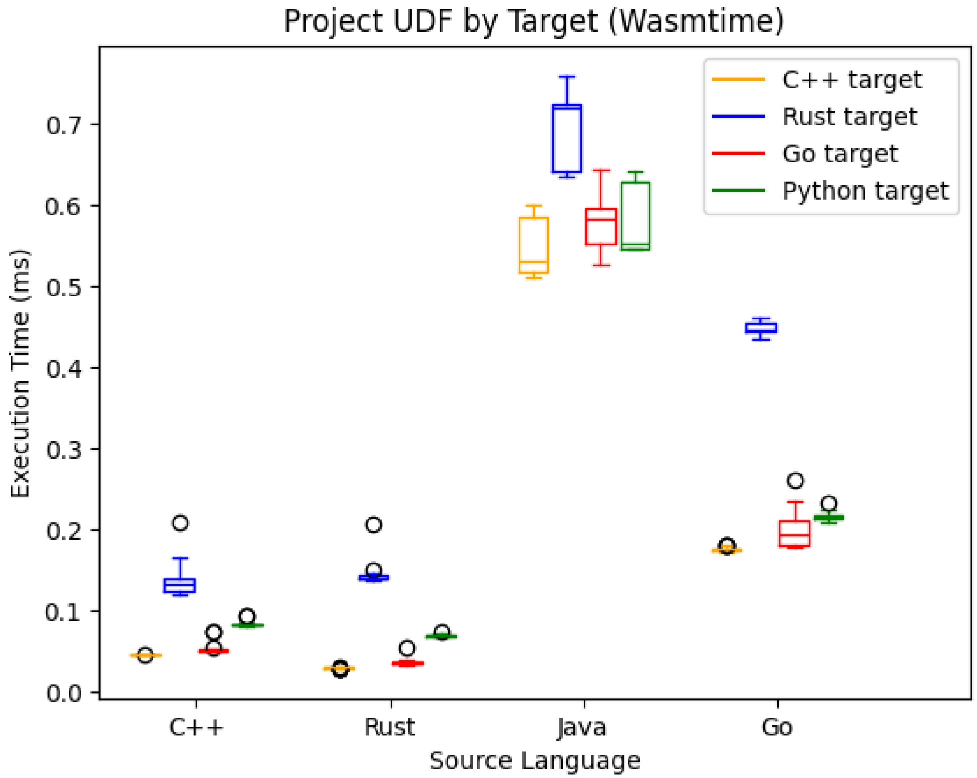
# Native vs Initial vs Subsequent Wasmtime





# Wasm performance across targets

Pushdown layer may be implemented in other languages, so it is important to evaluate support for calling Wasm from different target languages.



# Issues

- Language with a large runtime like Java or Python have limitations as a source language for Wasm.
  - But there are efforts such as WasmGC to improve its viability for higher level languages.
- Ecosystem is still relatively nascent for Wasm on the server.
  - WASI only recently added support for network sockets.

**Wasmer implementation matrix**

| | C++(src) | Rust (src) | Go (src) | Java (src) |
|---|---|---|---|---|
| C++(tgt) | ✓ | ✓ | ✓ | ✓ |
| Python (tgt) | ✓ | ✓ | ✓ | ✓ |
| Go (tgt) | ✓ | ✓ | ✓? | ✓? |
| Rust (tgt) | ✓ | ✓ | ✓ | ✓ |

**Wasmtime implementation matrix**

| | C++(src) | Rust (src) | Go (src) | Java (src) |
|---|---|---|---|---|
| C++((tgt) | ✓ | ✓ | ✓ | ✓ |
| Python (tgt) | ✓ | ✓ | ✓ | ✓ |
| Go (tgt) | ✓ | ✓ | ✓ | ✓ |
| Rust (tgt) | ✓ | ✓ | ✓ | ✓ |

# Final Thoughts

- While support for compiling higher level languages to Wasm are still nascent Systems programming languages like C++ and Rust have pretty good support.
- Wasm runtimes have bindings to support being called from a good range of target languages typically used in production for databases or backend services.
- Wasm while introducing overhead still gives performance within an order of magnitude of native.

# Questions ?