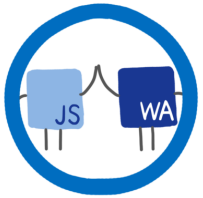
WebAssembly

Concepts of Programming Languages
23 November 2020

Sebastian Macke
Rosenheim Technical University

Content

- What is WebAssembly?
- Design
- Demo in WAT and C
- Hello world in Go
- Exercise
- Calls from and to WebAssembly
- Exercise
- Future of WebAssembly



What is WebAssembly?

webassembly.org (https://webassembly.org)

- WebAssembly is a portable binary-code format for executable programs.
- Timeline

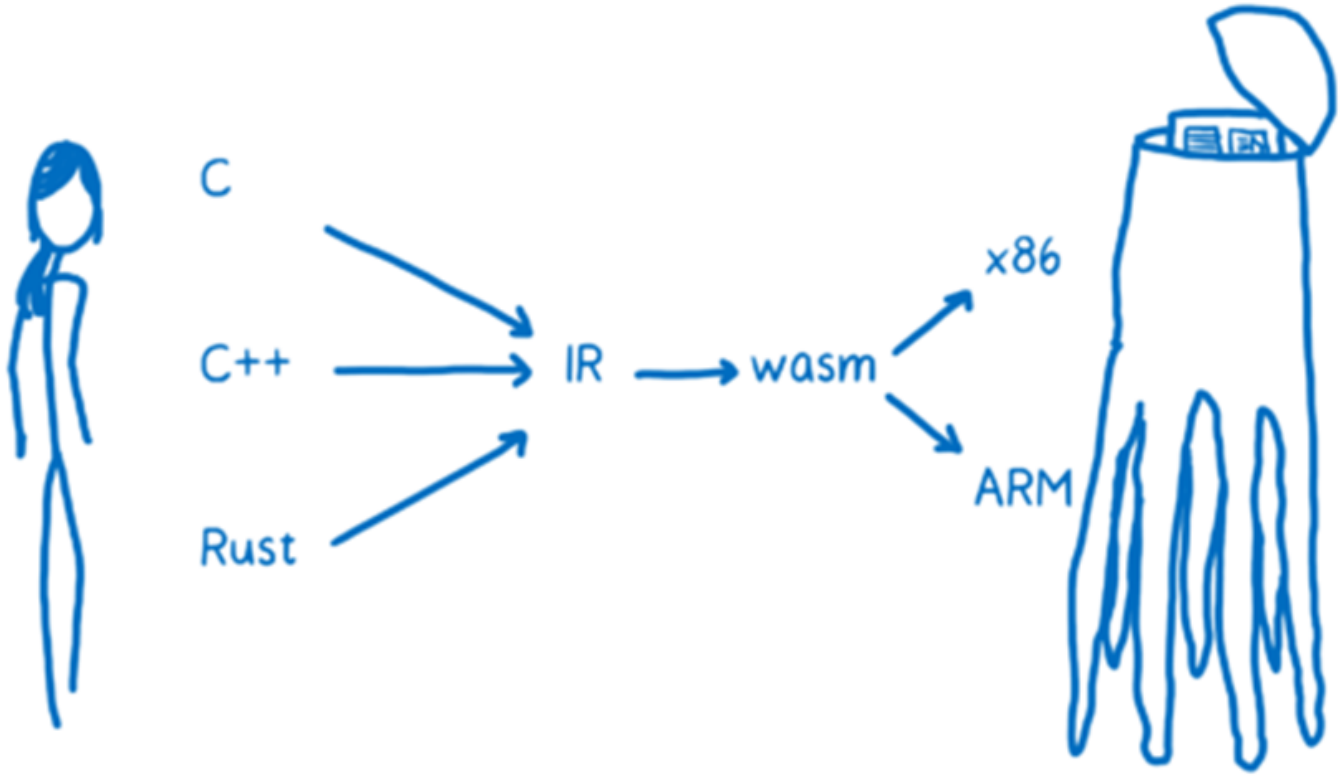
```
- 1995: JavaScript is developed within 10 days by Brendan Eich

- ... Java Applets, Flash, ActiveX, Silverlight
- ... JavaScript improvements, ...

- 2015: First presentation of WebAssembly from the W3C WebAssembly Working Group
- 2016: Google, Microsoft and Mozilla show their first implementations
- 2017: WebAssembly is offically supported by all Web Browsers
- 2019: A WebAssembly system interface is specified for portable applications outside the browser
- Now:  WebAssembly is constantly enhanced (https://webassembly.org/roadmap/)
```

After the introduction of JavaScript in the 90s, it is the second language supported directly by web browsers

3

What is WebAssembly?

- WebAssembly is designed as a portable target for compiling high-level languages

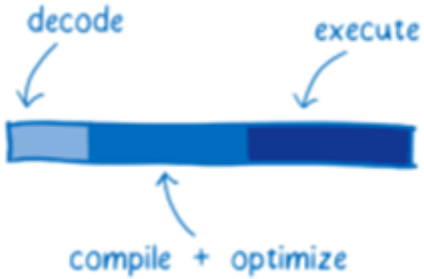


4

JavaScript vs. WebAssembly?

- JavaScript requires a complex optimization

- WebAssembly simplifies processing



5

Design

- Binary format

```
- small size and easy to interpret
```

- Stack Machine with local and global variables

```
- Efficient decoding and compilation
```

- Structured control flow and structured use of the stack

```
- One-Pass Verification
```

- Simple types: i32, i64, f32, f64

```
- Representation of today's CPU architecture
```

6

Design

- Linear memory with load/store addressing

```
- Sandboxed
- 64kB segmentt. Enables hardware-supported boundary check
- Size adaptable at runtime
```

- Interface via defined interface

```
- Export/import of functions, global variables and so-called tables
```

- Assembler text format

```
- Open and possibility for debugging
```

7

Stackmachine in Detail

C input source	Linear assembly bytecode (intermediate representation)	Wasm binary encoding (hexadecimal bytes)

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`get_local 0`
	`i64.eqz`
	`if (result i64)`
	`  i64.const 1`
	`else`
	`  get_local 0`
	`  get_local 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`end`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`

Demo in WebAssembly Text Format

- Example in `webassembly/wat` folder

add.wat

```
(module
  (type (;0;) (func (param i32 i32) (result i32)))
  (func (;0;) (type 0) (param i32 i32) (result i32)
    local.get 0
    local.get 1
    i32.add)
  (export "add" (func 0)))
```

add.js

```
const fs = require('fs');
const buf = fs.readFileSync('./add.wasm');

WebAssembly.instantiate(buf)
.then(result => {
    console.log(result.instance.exports.add(5, 2))
});
```

9

Supported languages in 2020

- Source:

github.com/appcypher/awesome-wasm-langs (https://github.com/appcypher/awesome-wasm-langs)

.Net	AssemblyScript	Astro
Brainfuck	C	C#
C++	D	Elixir
F#	Faust	Forest
Forth	Go	Grain
Haskell	Java	JavaScript
Julia	Idris	Kotlin/Native
Kou	Lua	Nim
Ocaml	Perl	PHP
Plorth	Poetry	Python
Prolog	Ruby	Rust
Scheme	Speedy.js	Turboscript
Wah	Walt	Wam
Wracket	Xlang	Zig

10

Demo in C

- Example in `webassembly/C` folder

add.c

```
int add(int a, int b) {
    return a+b;
}
```

add.js

```
const fs = require('fs');
const buf = fs.readFileSync('./add.wasm');

WebAssembly.instantiate(buf)
.then(result => {
    console.log(result.instance.exports.add(5, 2))
});
```

11

Demo in Go

```
package main

func main() {
    println("Hello World")
}
```

index.html

```
<html>
<body>
    <script src="wasm_exec.js"></script>
    <script>
        const go = new Go();

        fetch('lib.wasm')
            .then(response => response.arrayBuffer())
            .then(bytes =>WebAssembly.instantiate(bytes, go.importObject))
            .then(result => {
                go.run(result.instance)
            }
        );
    </script>
</body>
</html>
```

Exercise

- Write Hello world in Go and execute as node app or in the browser.

13

Applications

webassembly.org/docs/use-cases/ (https://webassembly.org/docs/use-cases/)

- Better execution for languages and toolkits that are currently cross-compiled to the Web (C/C++, GWT, ...).
- Image / video editing.
- Games:

```
- Casual games that need to start quickly.
- AAA games that have heavy assets.
- Game portals (mixed-party/origin content).
```

- Peer-to-peer applications (games, collaborative editing, decentralized and centralized).
- Music applications (streaming, caching).
- Image recognition.
- Live video augmentation (e.g. putting hats on people's heads).

Applications

- VR and augmented reality (very low latency).
- CAD applications.
- Scientific visualization and simulation.
- Interactive educational software, and news articles.
- Platform simulation / emulation (ARC, DOSBox, QEMU, MAME, ...).
- Language interpreters and virtual machines.
- POSIX user-space environment, allowing porting of existing POSIX applications.
- Developer tooling (editors, compilers, debuggers, ...).
- Remote desktop.
- VPN.
- Encryption.

Outside the browser

- Game distribution service (portable and secure).
- Server-side compute of untrusted code.
- Server-side application.
- Hybrid native apps on mobile devices.
- Generic Plugins for your software.

Calls from and to Go

```
package main

import (
    "syscall/js"
)

func add(this js.Value, input []js.Value) interface{} {
    result := input[0].Float() + input[1].Float()
    return js.ValueOf(result)
}

func main() {
    document := js.Global().Get("document")
    p := document.Call("createElement", "p")
    p.Set("innerHTML", "Hello WASM from Go!")
    document.Get("body").Call("appendChild", p)

    // register function
    js.Global().Set("add", js.FuncOf(add))

    // prevent exit
    c := make(chan struct{}, 0)
    <-c
}
```

Calls from and to Go

index.html

```
        const go = new Go();

        fetch('lib.wasm')
            .then(response => response.arrayBuffer())
            .then(bytes =>WebAssembly.instantiate(bytes, go.importObject))
            .then(result => {
                go.run(result.instance)
                document.body.innerHTML += add(41, 1);
            }
        );
    </script>
```

18

Exercise

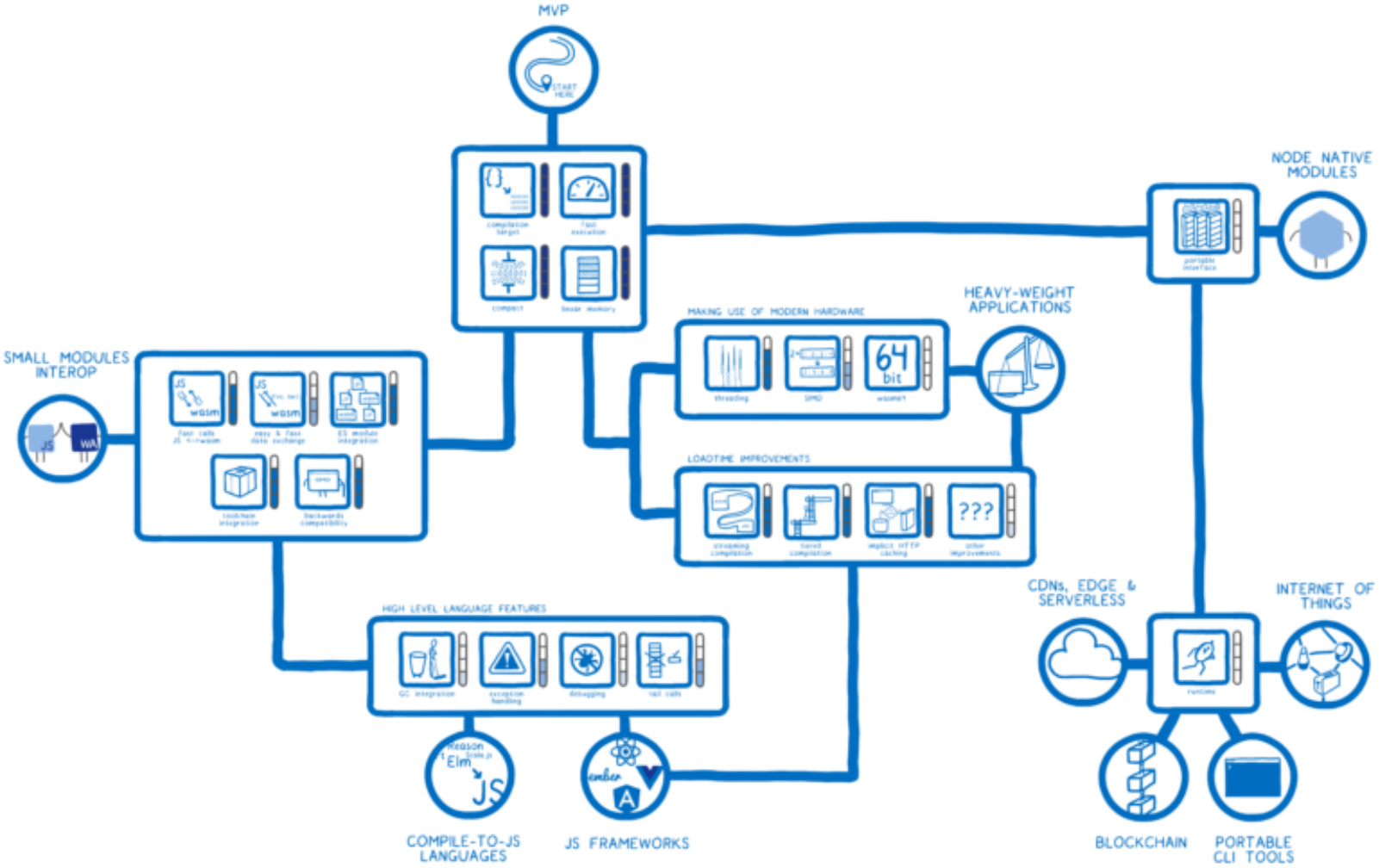
- Show generated image in the browser

WebAssembly and Go Routines

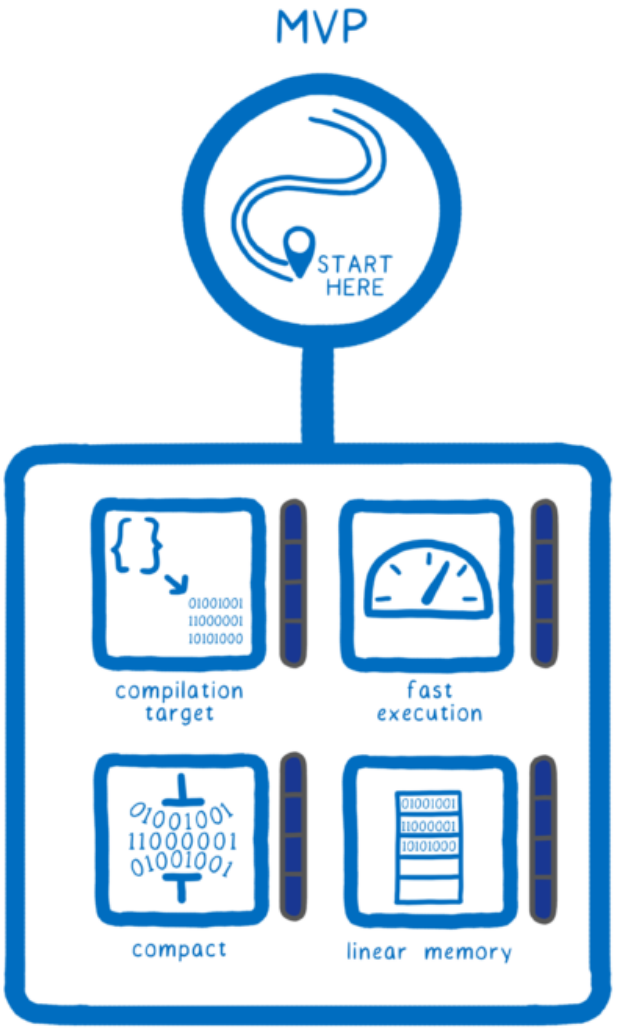
- Demo

20

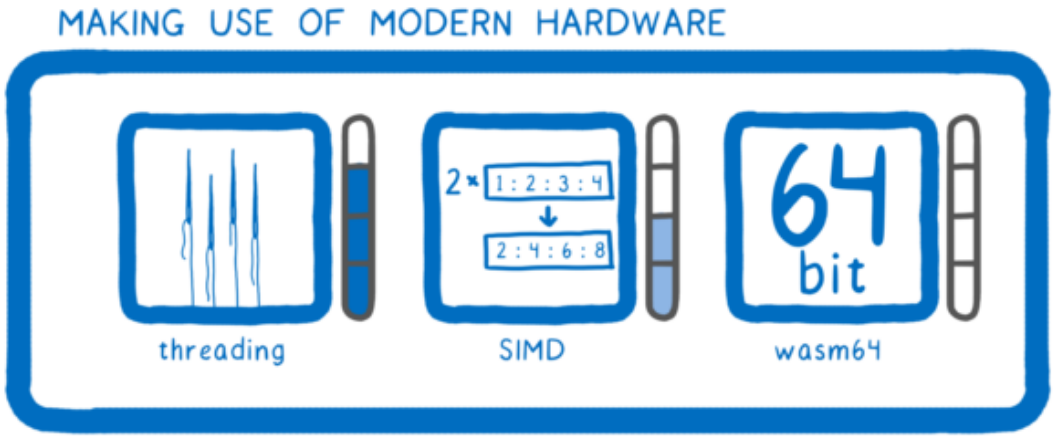
Future - The Skill Tree

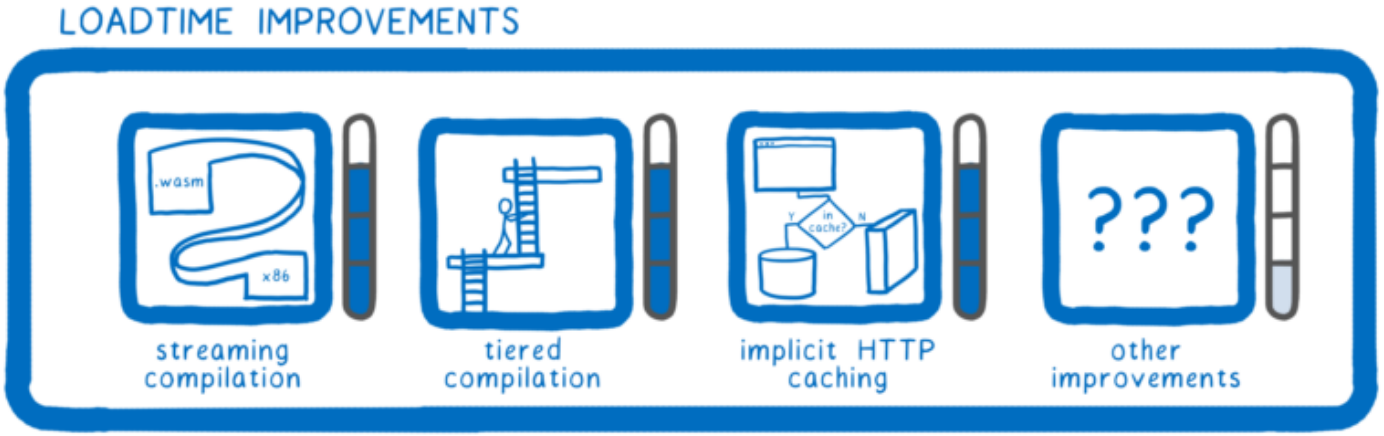


Future - MVP

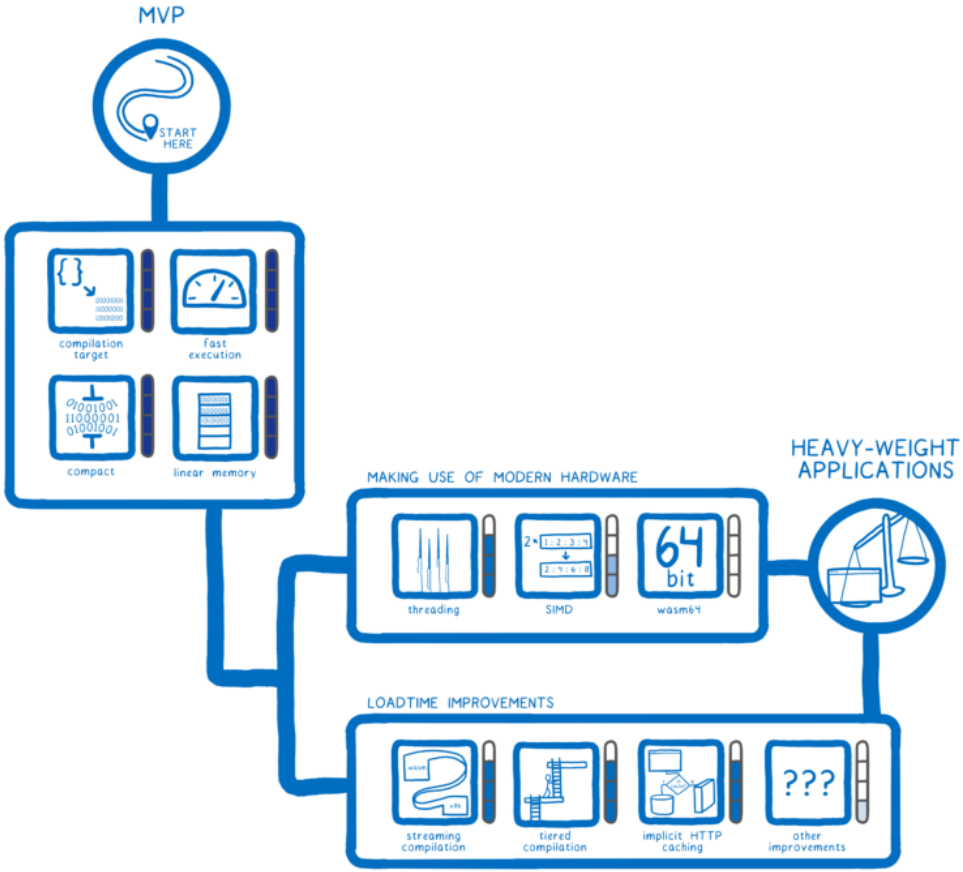

MVP
START HERE
compilation target
fast execution
compact
linear memory

Future

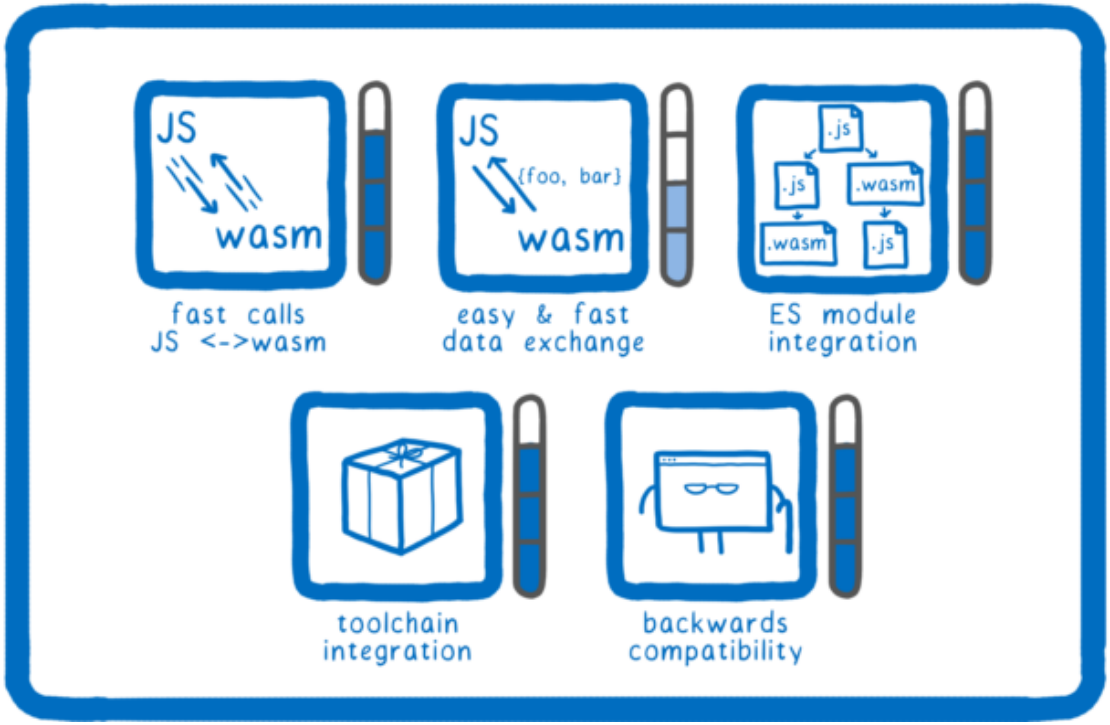

MAKING USE OF MODERN HARDWARE
threading
2×
1 : 2 : 3 : 4
2 : 4 : 6 : 8
SIMD
64
bit
wasm64


LOADTIME IMPROVEMENTS
.wasm
x86
streaming compilation
tiered compilation
Y
in cache?
N
implicit HTTP caching
???
other improvements

Future

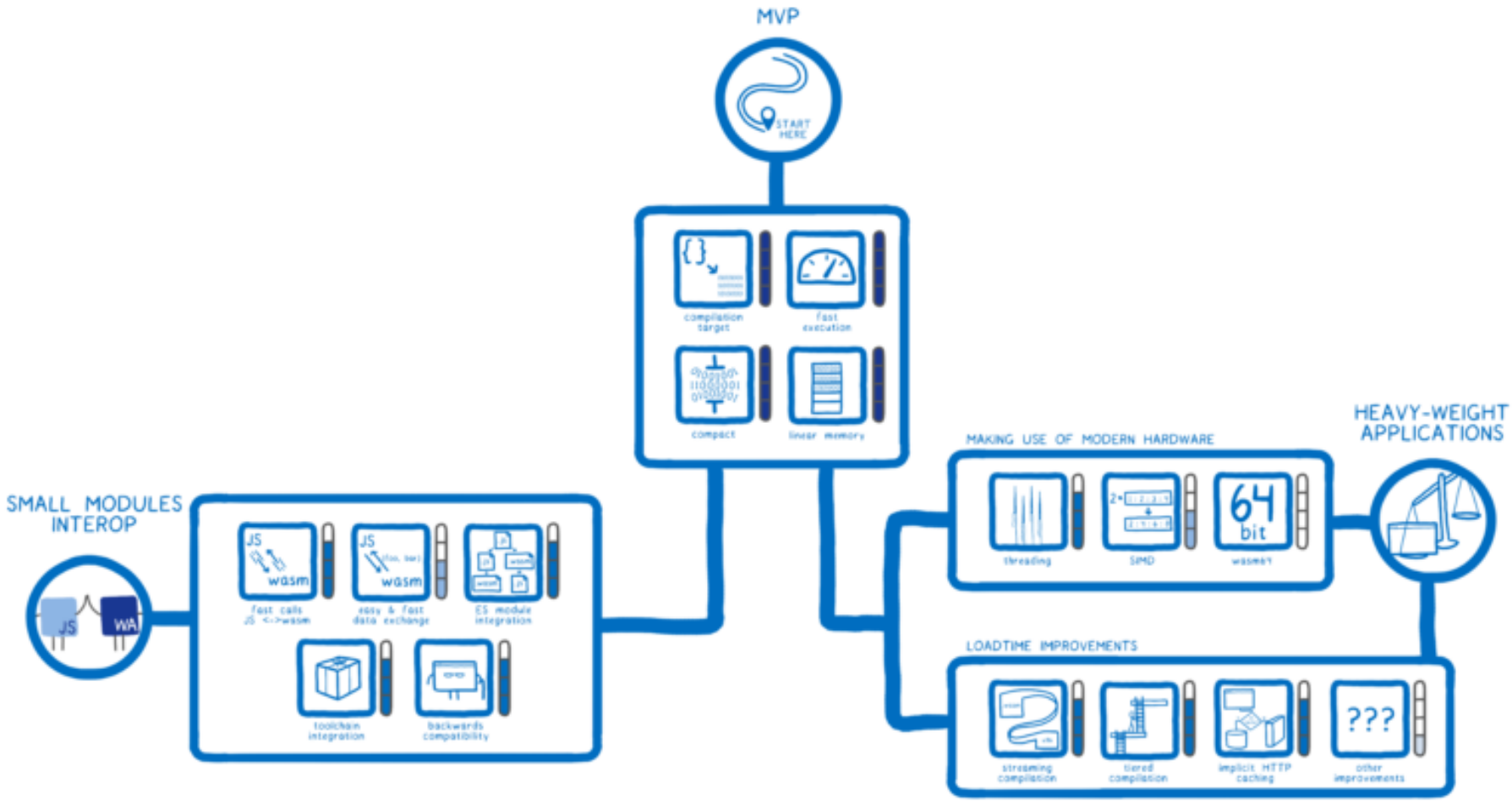

MVP
START HERE
compilation target
fast execution
compact
linear memory
MAKING USE OF MODERN HARDWARE
threading
SIMD
64 bit
wasm64
HEAVY-WEIGHT APPLICATIONS
LOADTIME IMPROVEMENTS
streaming compilation
tiered compilation
implicit HTTP caching
???
other improvements

Future

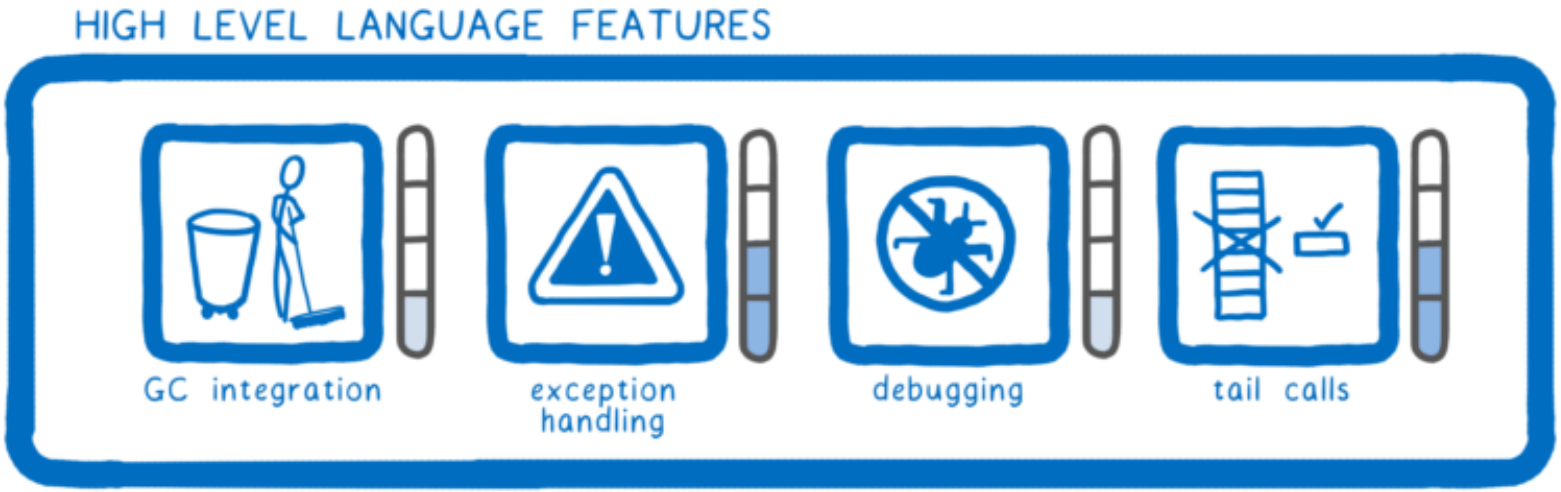

JS
wasm
fast calls
JS <->wasm
JS
{foo, bar}
wasm
easy & fast
data exchange
.js
.js
.wasm
.wasm
.js
ES module
integration
toolchain
integration
backwards
compatibility

25

Future

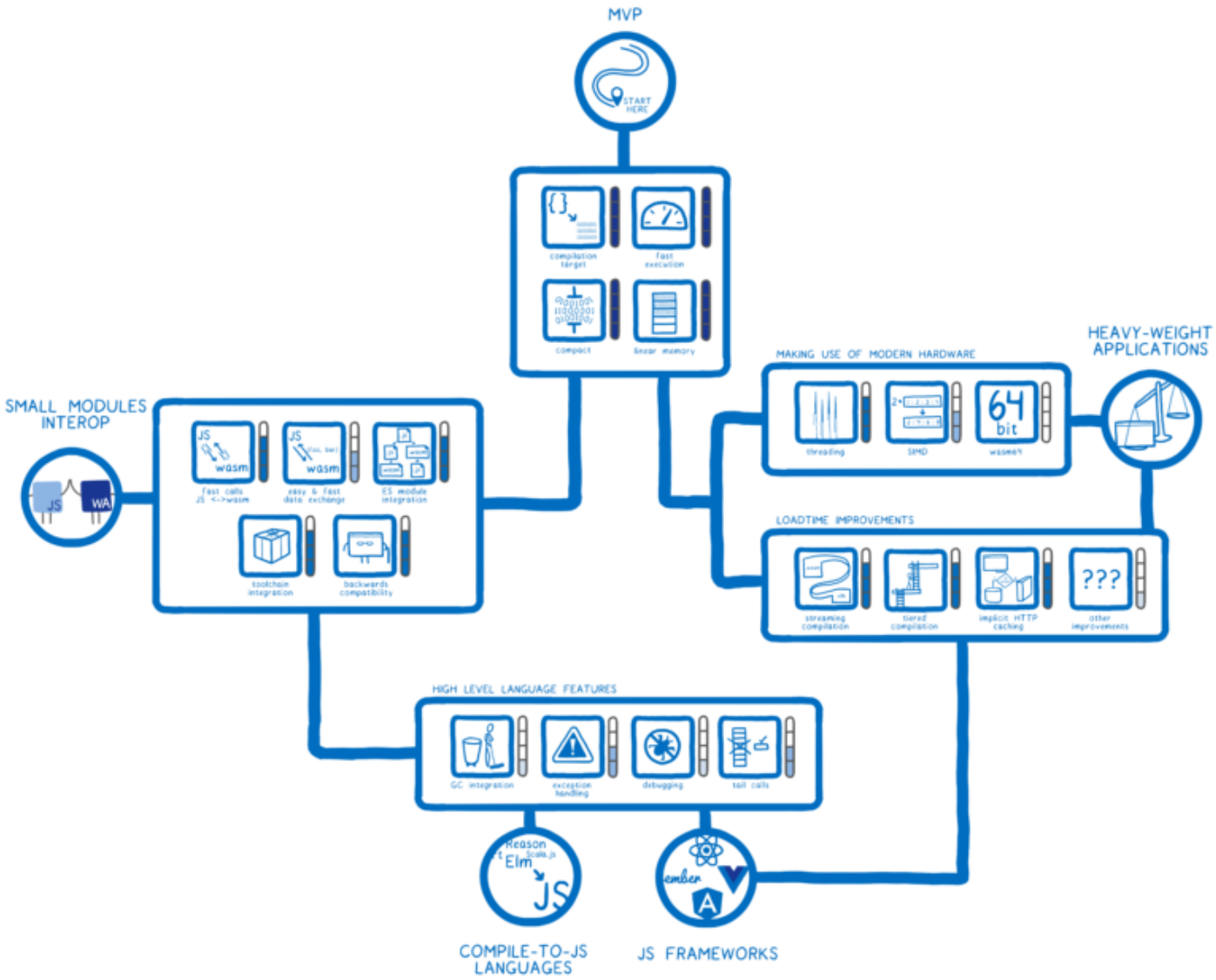

MVP
START HERE
compilation target
fast execution
compact
linear memory
SMALL MODULES INTEROP
JS
WA
JS
wasm
fast calls JS <->wasm
JS
wasm
easy & fast data exchange
ES module integration
toolchain integration
backwards compatibility
MAKING USE OF MODERN HARDWARE
threading
SIMD
64 bit
wasm64
HEAVY-WEIGHT APPLICATIONS
LOADTIME IMPROVEMENTS
???
streaming compilation
tiered compilation
implicit HTTP caching
other improvements

Future

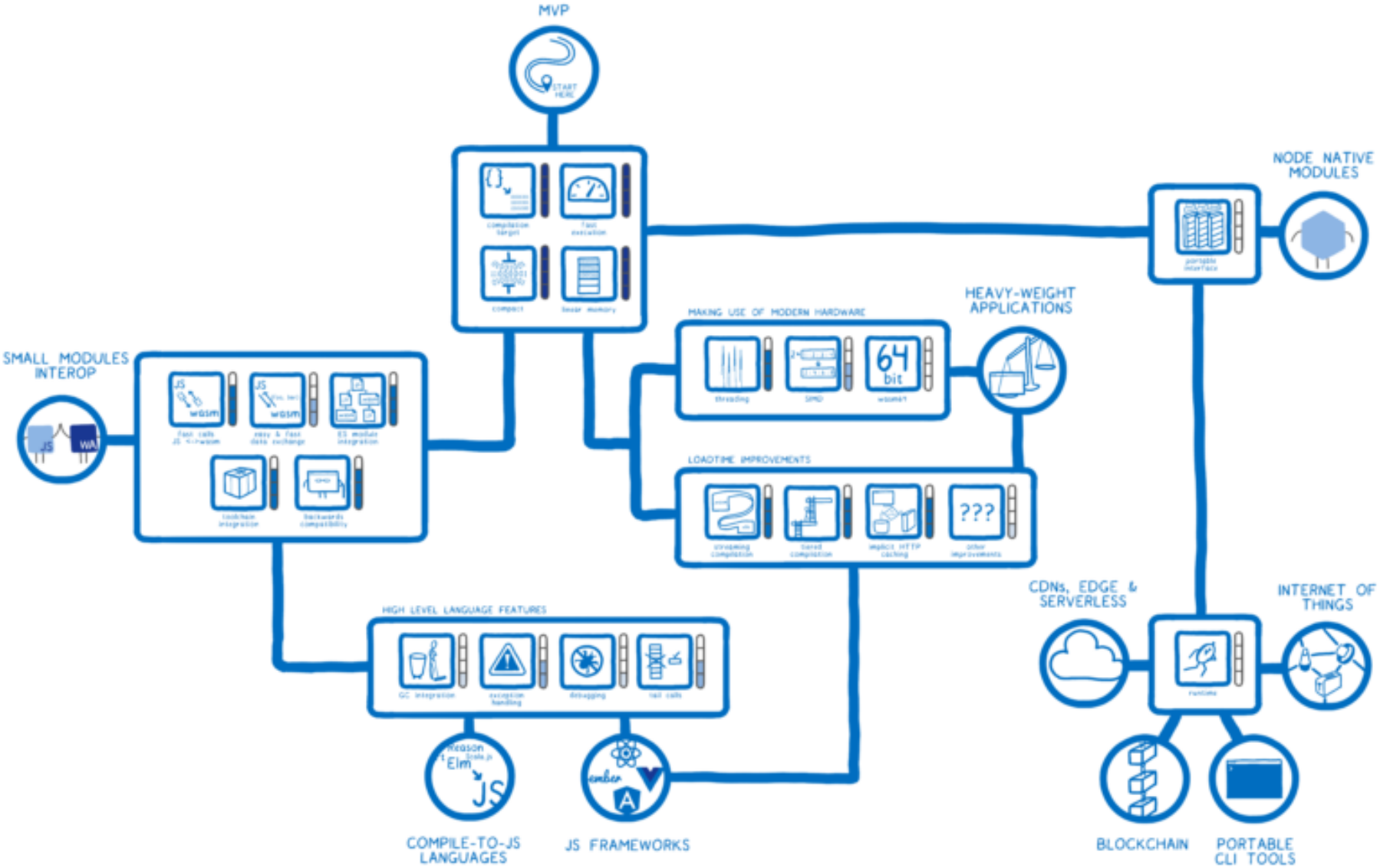

HIGH LEVEL LANGUAGE FEATURES
GC integration
exception handling
debugging
tail calls

27

Future


MVP
START HERE
compilation target
fast execution
compact
linear memory
SMALL MODULES INTEROP
JS
WA
JS
wasm
fast calls JS <->wasm
JS
wasm
easy & fast data exchange
ES module integration
toolchain integration
backwards compatibility
MAKING USE OF MODERN HARDWARE
threading
SIMD
64 bit
wasm64
HEAVY-WEIGHT APPLICATIONS
LOADTIME IMPROVEMENTS
streaming compilation
tiered compilation
implicit HTTP caching
???
other improvements
HIGH LEVEL LANGUAGE FEATURES
GC integration
exception handling
debugging
tail calls
Reason
Elm
scala.js
JS
ember
COMPILE-TO-JS LANGUAGES
JS FRAMEWORKS

Future


MVP
START HERE
NODE NATIVE MODULES
HEAVY-WEIGHT APPLICATIONS
MAKING USE OF MODERN HARDWARE
64 bit
SMALL MODULES INTEROP
LOADTIME IMPROVEMENTS
???
HIGH LEVEL LANGUAGE FEATURES
CDNs, EDGE & SERVERLESS
INTERNET OF THINGS
COMPILE-TO-JS LANGUAGES
JS FRAMEWORKS
BLOCKCHAIN
PORTABLE CLI TOOLS

Links

- Awesome-wasm, Curated list of awesome things regarding WebAssembly

  github.com/mbasso/awesome-wasm (https://github.com/mbasso/awesome-wasm)

- WasmWeekly, WebAssembly Today, Newletter

  wasmweekly.news/ (https://wasmweekly.news/)

  webassemblytoday.substack.com/ (https://webassemblytoday.substack.com/)

- Webassembly Binary Toolkit

  github.com/WebAssembly/wabt (https://github.com/WebAssembly/wabt)

- Binaryen, Compiler infrastructure and toolchain library

  github.com/WebAssembly/binaryen (https://github.com/WebAssembly/binaryen)

Links

- WebAssembly Studio

webassembly.studio/ (https://webassembly.studio/)

- WebAssembly Roadmap

webassembly.org/roadmap/ (https://webassembly.org/roadmap/)

Thank you

Sebastian Macke
Rosenheim Technical University
sebastian.macke@qaware.de (mailto:sebastian.macke@qaware.de)
http://www.qaware.de (http://www.qaware.de)