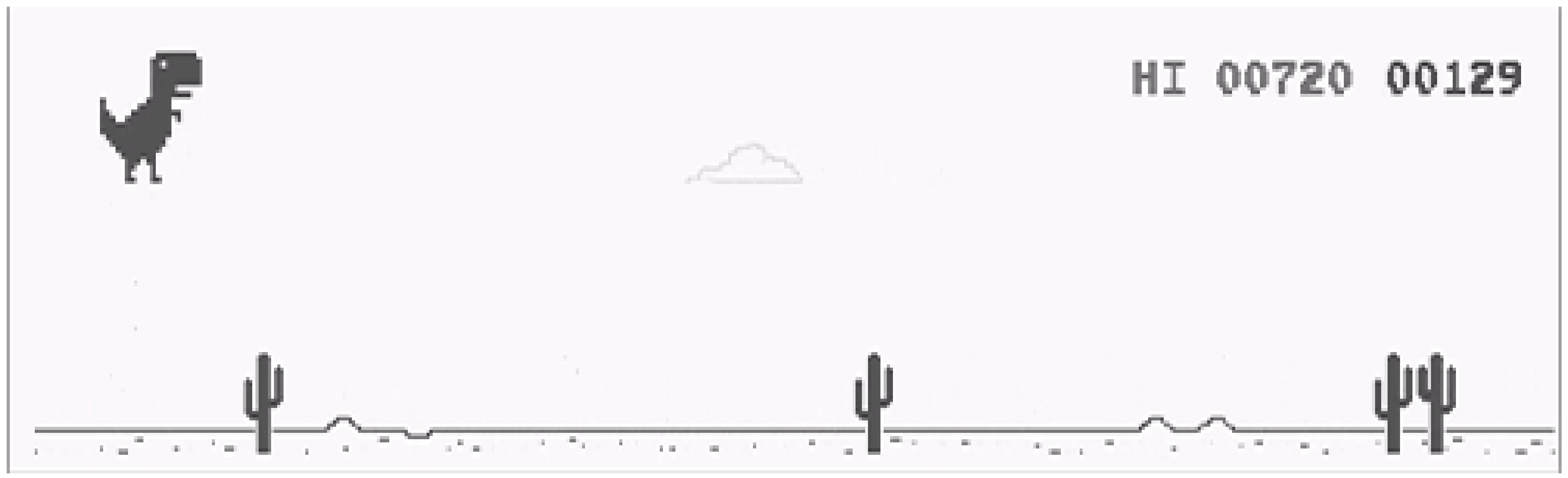

HI 00720 00129

Rust DC—Learn+Try Rust in the Browser via WebAssembly

Scott Steele (github.com/scooter-dangle)

Hello!!

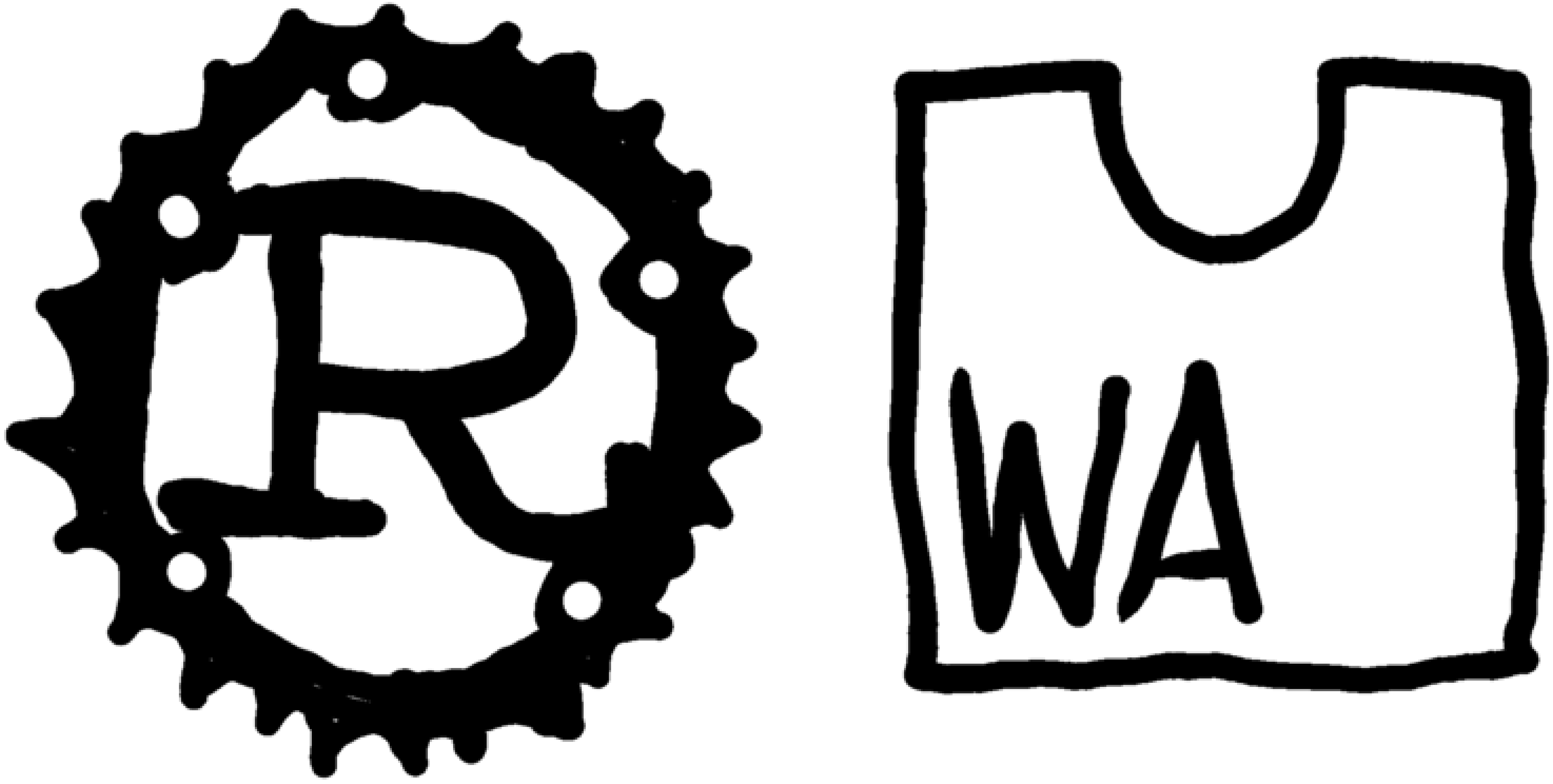
Hello!!

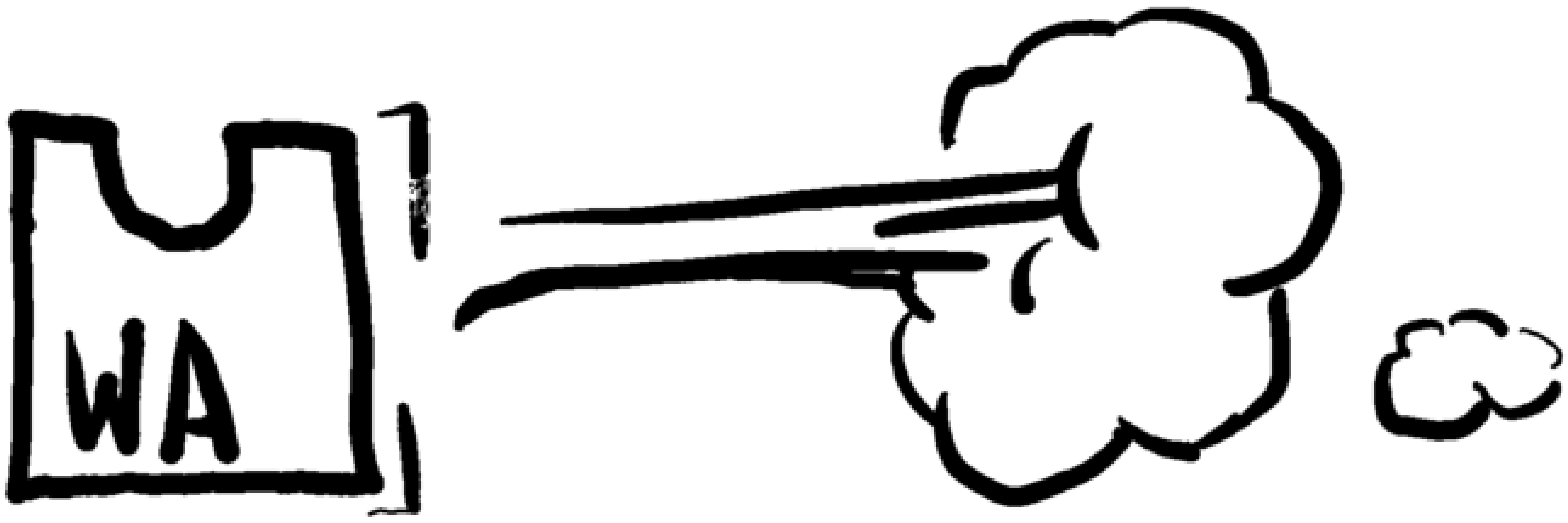
My bona fides

Hello!!

My bona fides

Look what I done!


R
WA


WA

Why to do??

Why to do??

Speeds

Why to do??

Speeds

parsing

Why to do??

Speeds

parsing

jam-packed with that great taste that machine's love!

Why to do??

Speeds

parsing

jam-packed with that great taste that machine's love!

www.smashingmagazine.com/2017/05/abridged-cartoon-introduction-webassembly/

Why to do??

Why to do??

Libraries!

Why to do??

Libraries!

Write once, run from:

- Python
- Ruby
- Go
- Lua
- JS (Node)

Why to do??

Libraries!

Write once, run from:

- Python
- Ruby
- Go
- Lua
- JS (Node)

...

- Every JQuery plugin on God's Green Earth!

Why to do??

Libraries!

Write once, run from:

- Python
- Ruby
- Go
- Lua
- JS (Node)

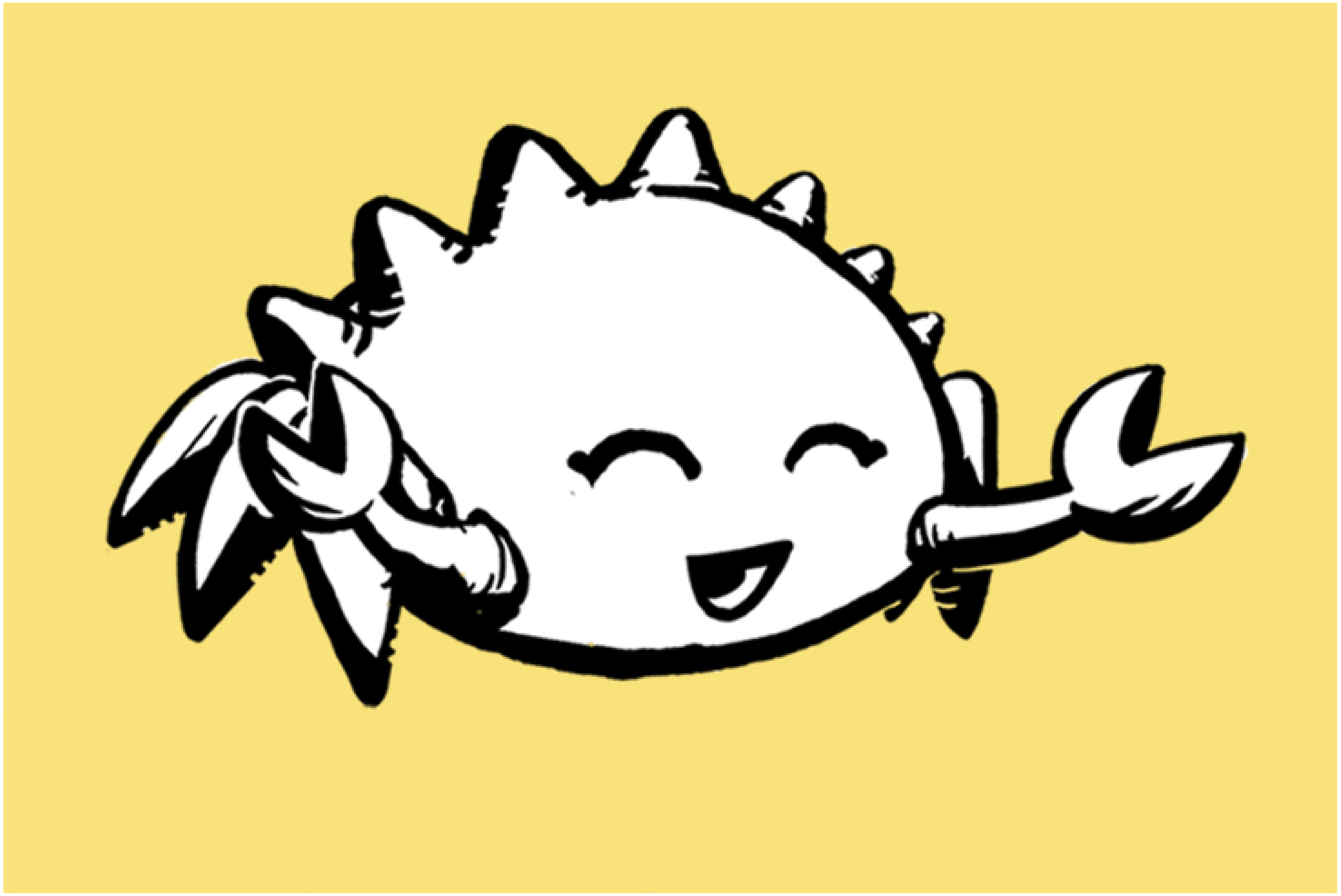
…

- JS (browser)

Why to do??

Rust!

Why to do??



Why to do??

Rust!

Why to do??

Rust!

- Efficient, expressive compile time checks

Why to do??

Rust!

- Efficient, expressive compile time checks
- Fast enough that you can *implement first*

Why to do??

Rust!

- Efficient, expressive compile time checks
- Fast enough that you can *implement first*
- Structured enough that (when you need to) you can *optimize later*

Why to do??

Rust!

- Efficient, expressive compile time checks
- Fast enough that you can *implement first*
- Structured enough that (when you need to) you can *optimize later*
    - *much* later

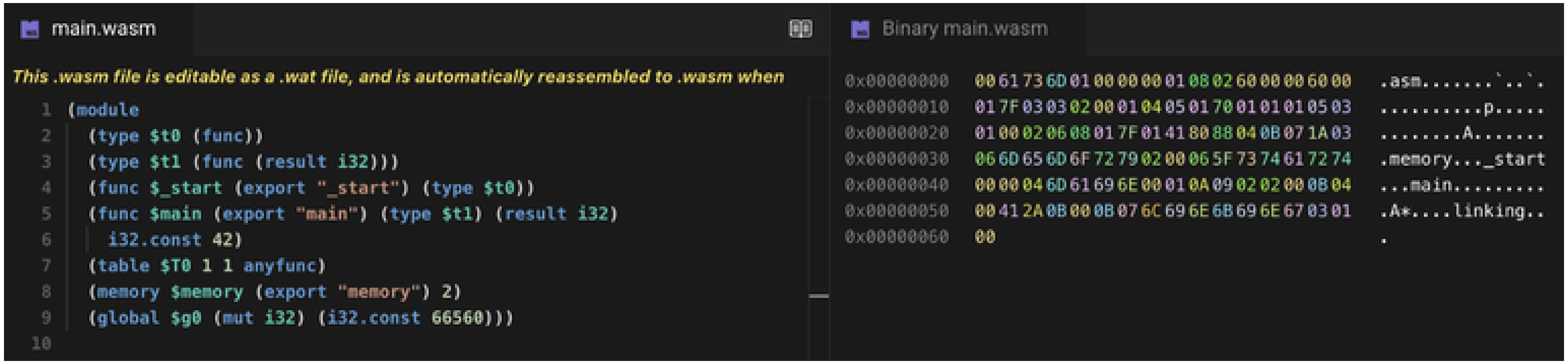
Why to do??

Rust!

- Efficient, expressive compile time checks
- Fast enough that you can *implement first*
- Structured enough that (when you need to) you can *optimize later*
  - *much* later
  - *and* without breaking stuff

Whatsit look like??

Whatsit look like??

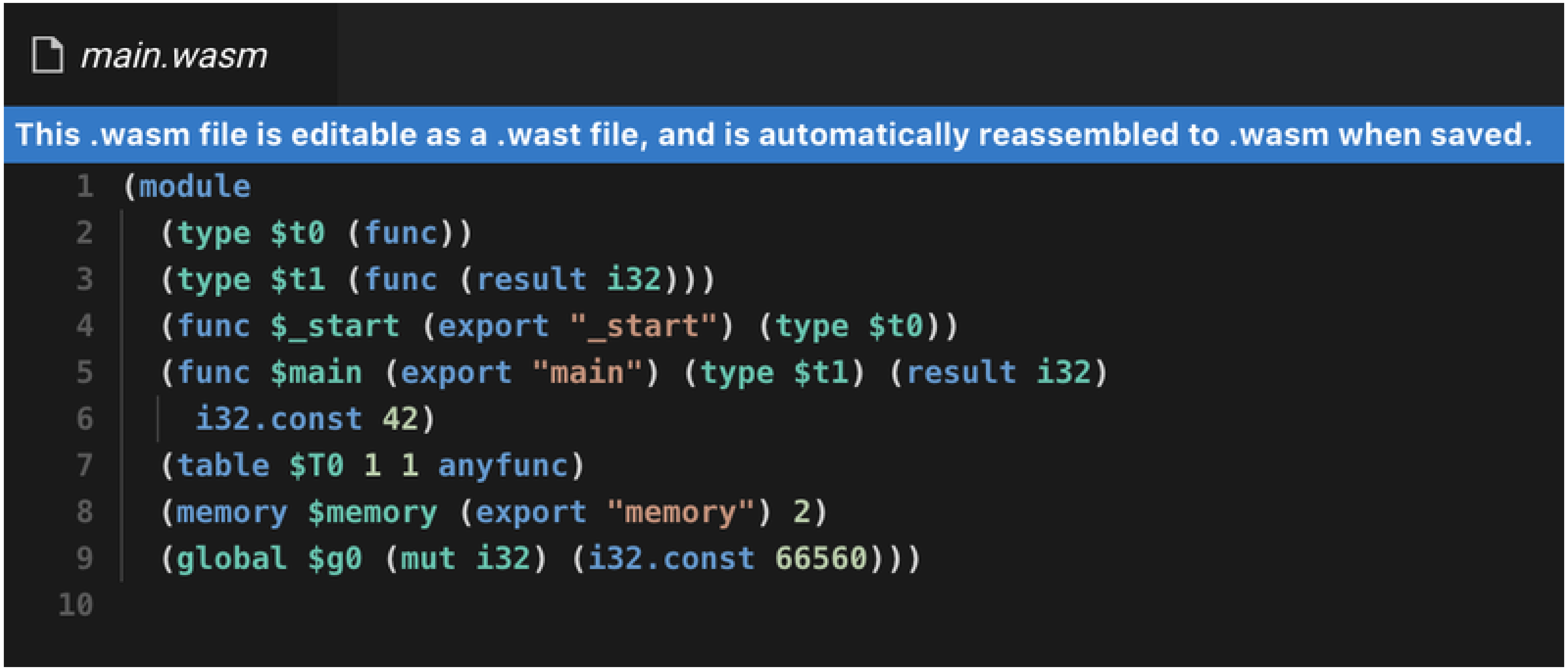


This .wasm file is editable as a .wat file, and is automatically reassembled to .wasm when

```
(module
  (type $t0 (func))
  (type $t1 (func (result i32)))
  (func $_start (export "_start") (type $t0))
  (func $main (export "main") (type $t1) (result i32)
    i32.const 42)
  (table $T0 1 1 anyfunc)
  (memory $memory (export "memory") 2)
  (global $g0 (mut i32) (i32.const 66560)))
```

```
0x00000000   00 61 73 6D 01 00 00 00 01 08 02 60 00 00 60 00   .asm.......`..`.
0x00000010   01 7F 03 03 02 00 01 04 05 01 70 01 01 01 05 03   ..........p.....
0x00000020   01 00 02 06 08 01 7F 01 41 80 88 04 0B 07 1A 03   ........A.......
0x00000030   06 6D 65 6D 6F 72 79 02 00 06 5F 73 74 61 72 74   .memory..._start
0x00000040   00 00 04 6D 61 69 6E 00 01 0A 09 02 02 00 0B 04   ...main.........
0x00000050   00 41 2A 0B 00 0B 07 6C 69 6E 6B 69 6E 67 03 01   .A*....linking..
0x00000060   00                                                .
```

Whatsit look like??

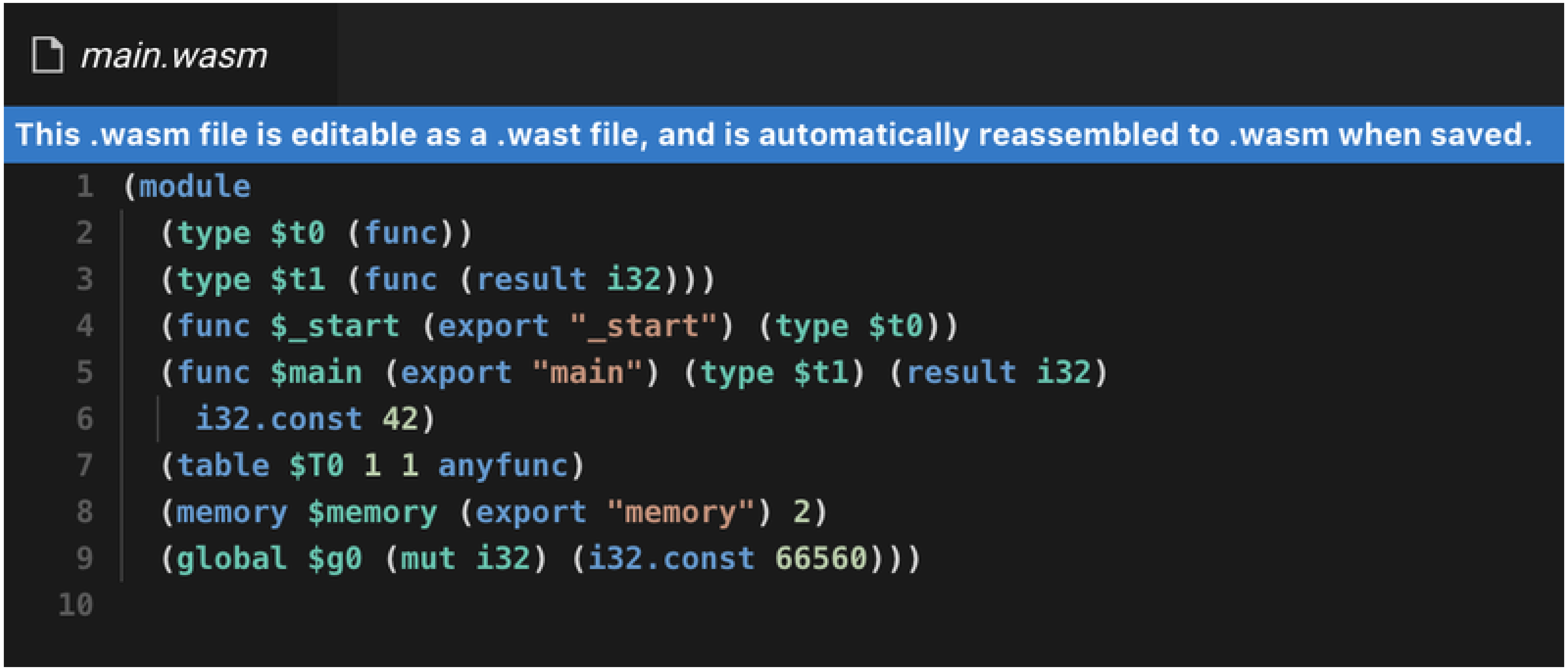


main.wasm

This .wasm file is editable as a .wast file, and is automatically reassembled to .wasm when saved.

```
(module
  (type $t0 (func))
  (type $t1 (func (result i32)))
  (func $_start (export "_start") (type $t0))
  (func $main (export "main") (type $t1) (result i32)
    i32.const 42)
  (table $T0 1 1 anyfunc)
  (memory $memory (export "memory") 2)
  (global $g0 (mut i32) (i32.const 66560)))
```

Whatsit look like??


main.wasm

This .wasm file is editable as a .wast file, and is automatically reassembled to .wasm when saved.

```
(module
  (type $t0 (func))
  (type $t1 (func (result i32)))
  (func $_start (export "_start") (type $t0))
  (func $main (export "main") (type $t1) (result i32)
    i32.const 42)
  (table $T0 1 1 anyfunc)
  (memory $memory (export "memory") 2)
  (global $g0 (mut i32) (i32.const 66560)))

```

- Gonna need an uber editor?

Whatsit look like??

Data types

Whatsit look like??

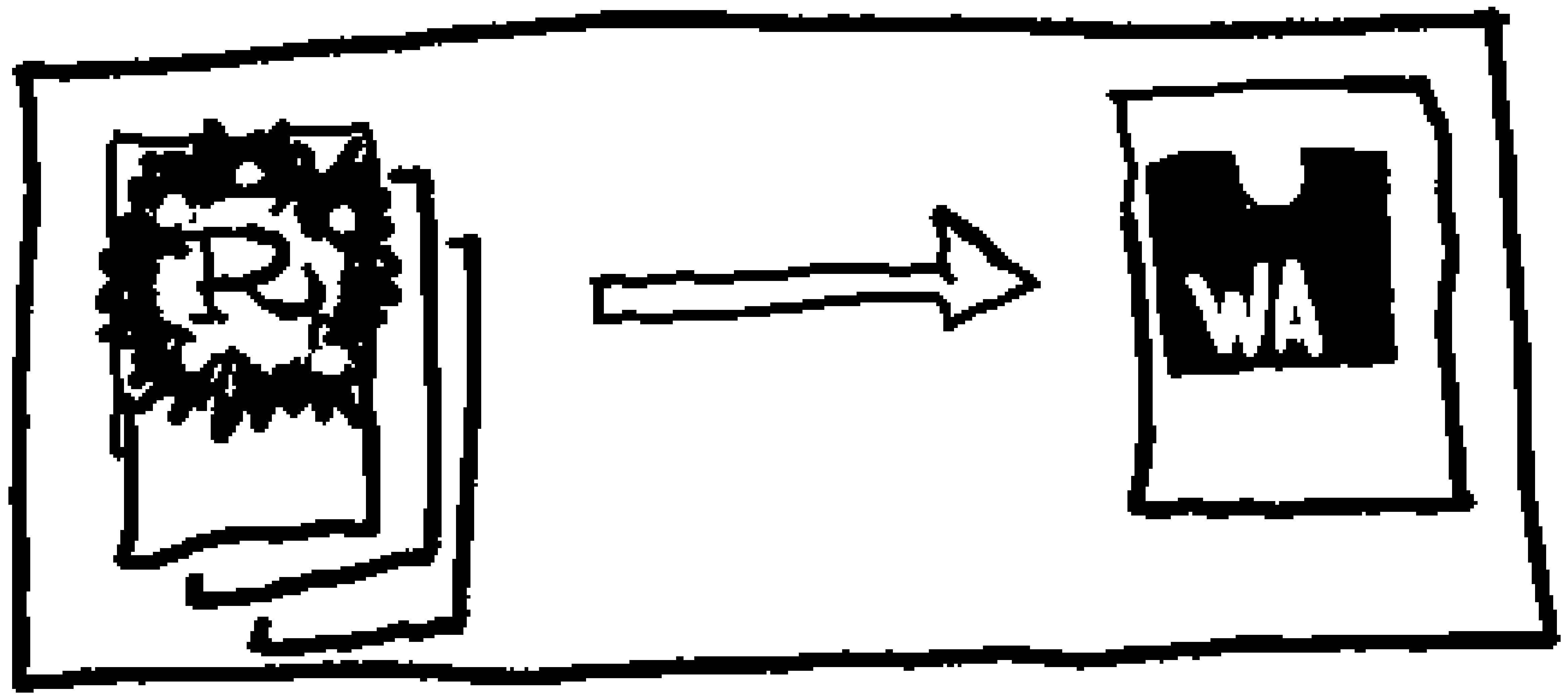
Data types

numbers

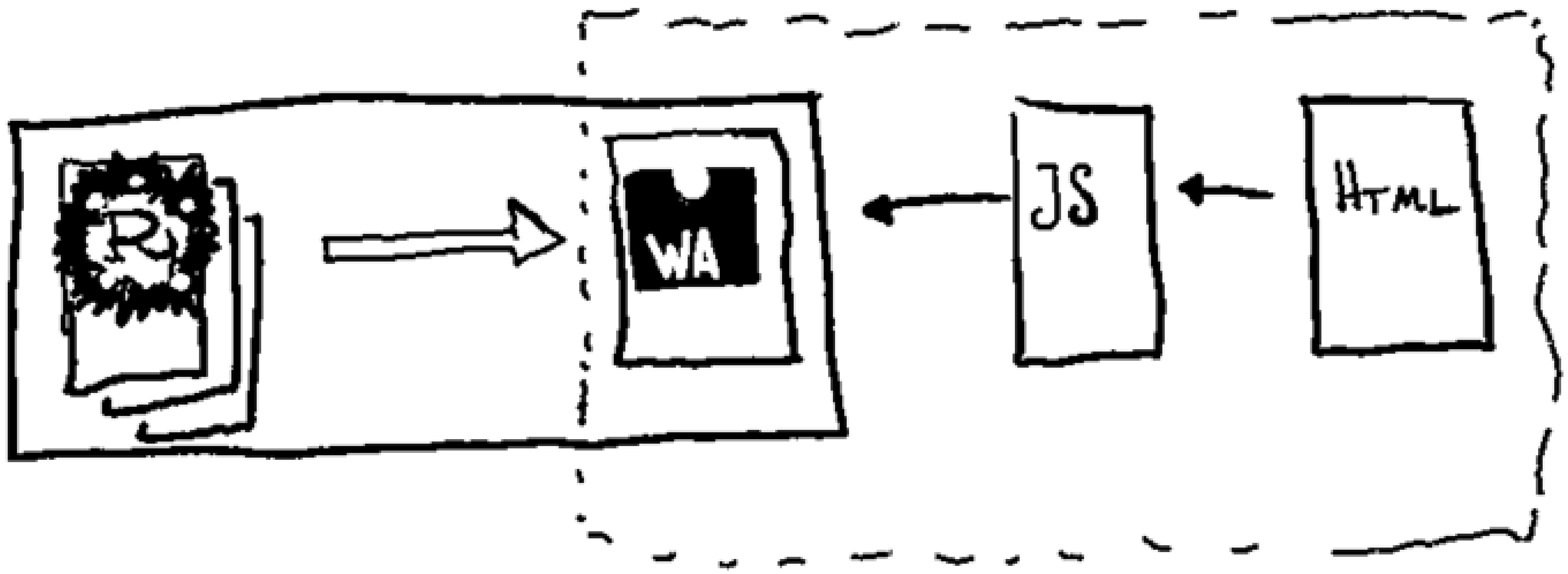
- `i32`
- `i64`
- `f32`
- `f64`

How to do?

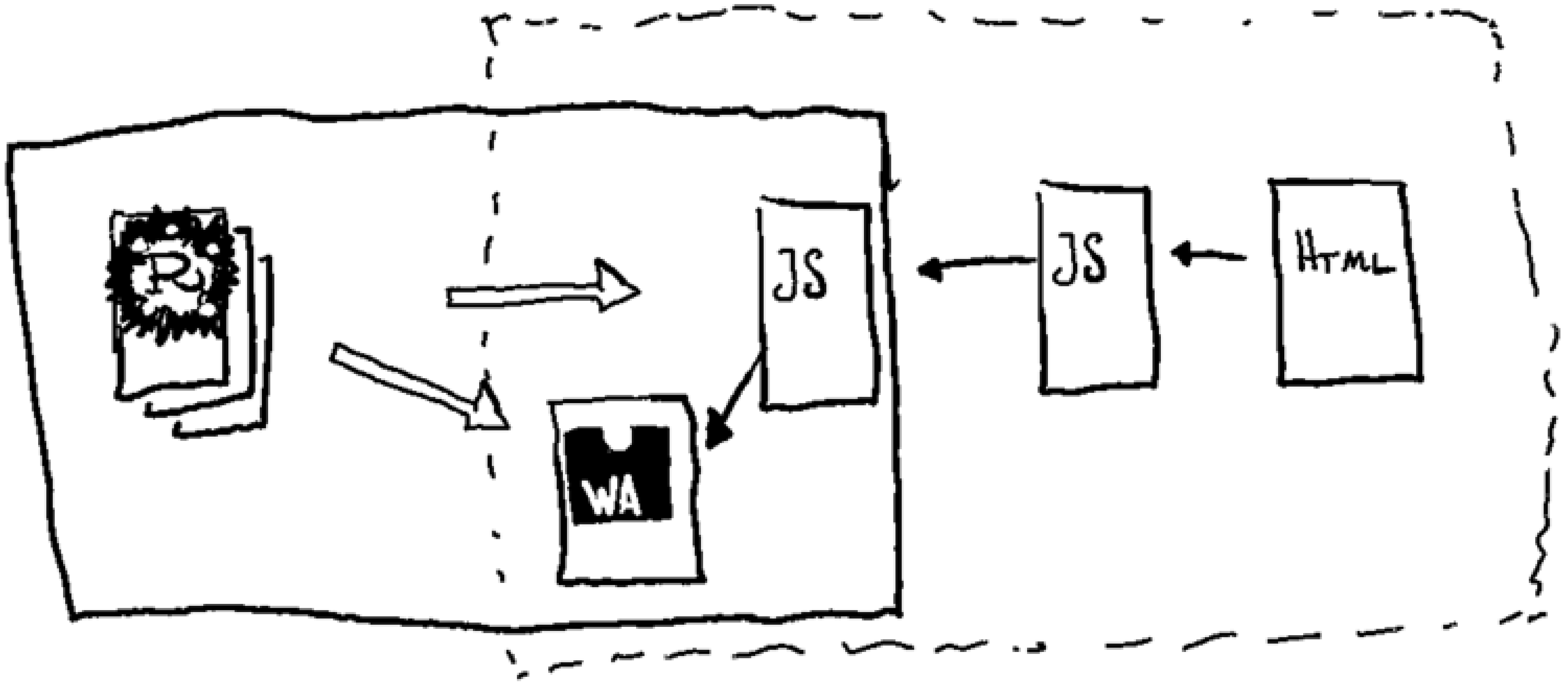
How to do?



How to do?



How to do?



Emscripten

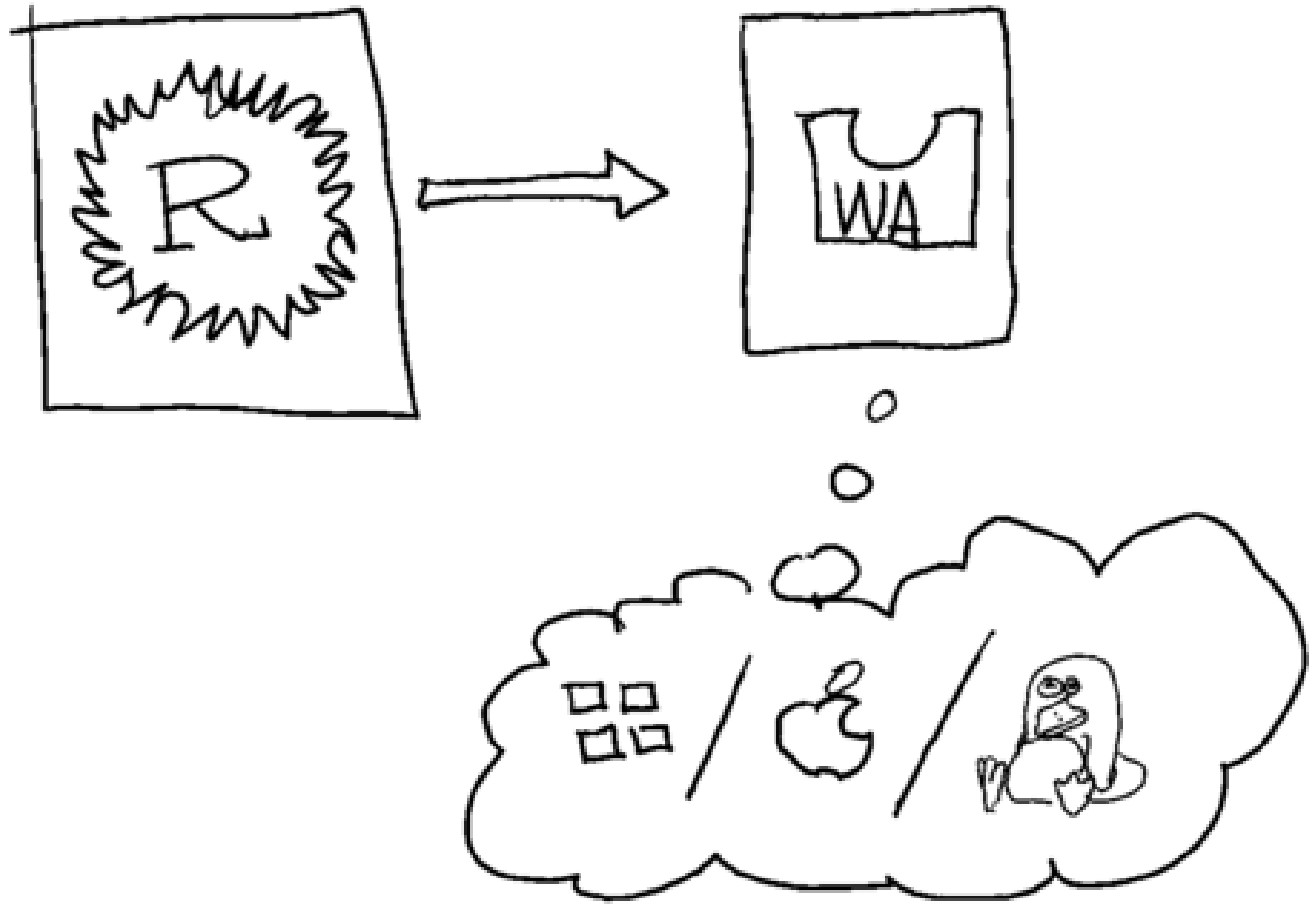
Emscripten

- kripken.github.io/emscripten-site/

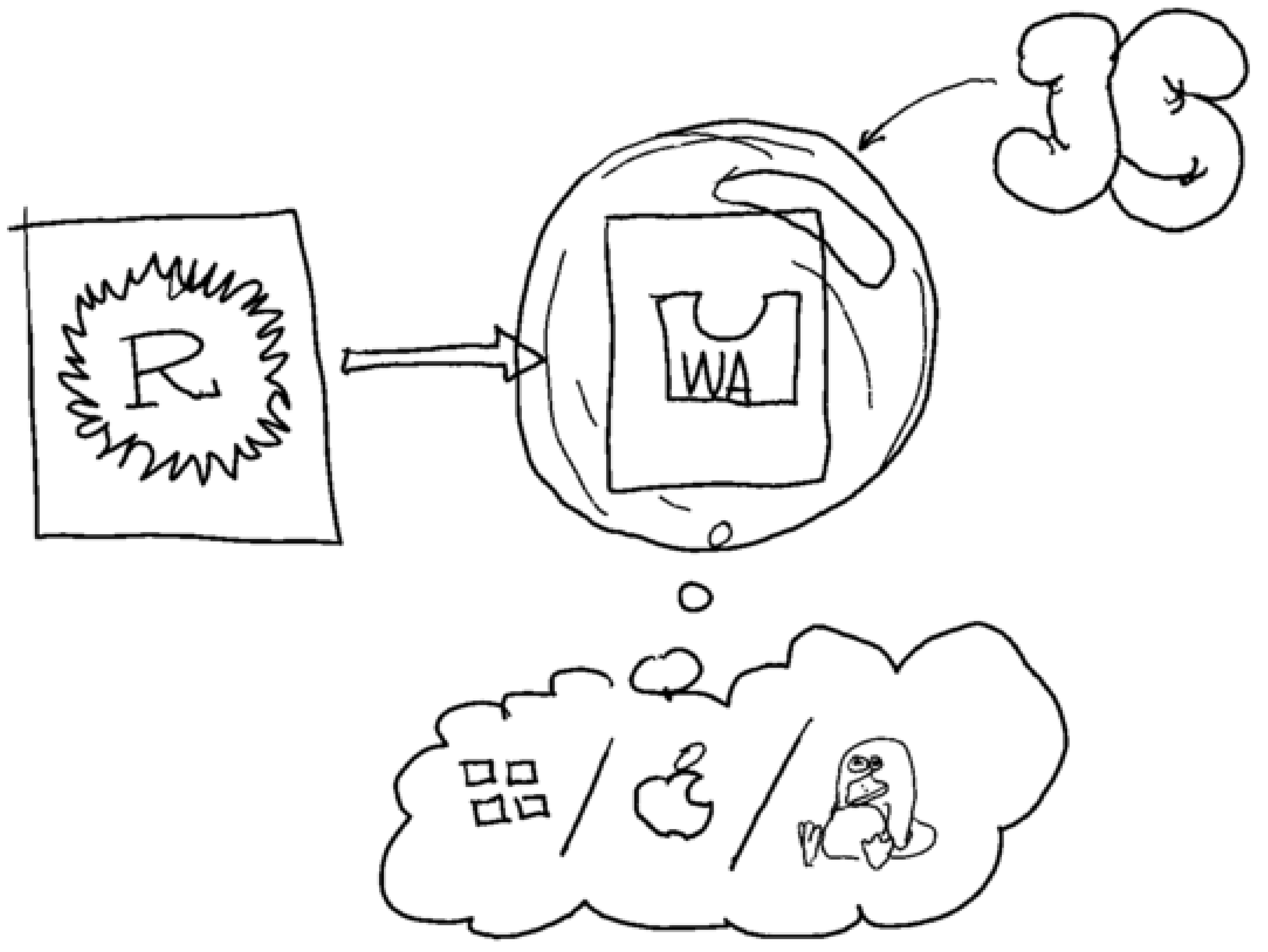
Emscripten

- kripken.github.io/emscripten-site/
- github.com/scooter-dangle/rust-wasm (master branch is Emscripten example)

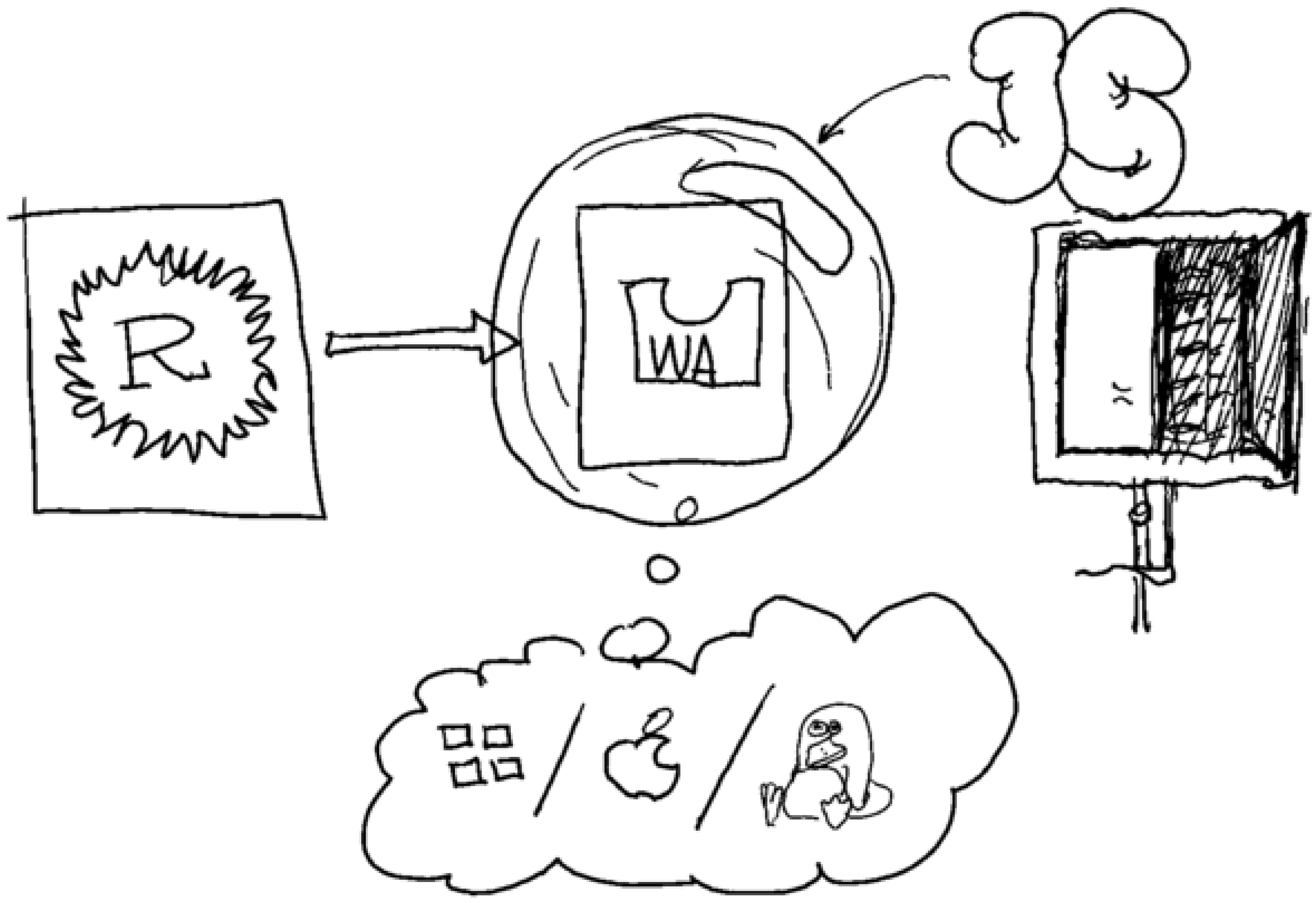
Emscripten



Emscripten



Emscripten



Emscripten

```
// main.rs
fn main() {
    /* Intentionally left blank */
}

//
// ...
//

fn safe_regex_compare(reg: String, target: String) -> Result<bool, Box<Error>> {
    regex!(&reg).map(|reg| reg.is_match(&target))
}

// See the declarations of this function in .cargo/config and wasm.html
#[no_mangle]
pub unsafe fn regex_compare(reg: *const c_char, target: *const c_char) -> bool {
    let reg    = ptr_to_string!(reg);
    let target = ptr_to_string!(target);
    safe_regex_compare(reg, target).unwrap()
}

//
// ...
//

#[no_mangle]
pub unsafe fn validate_regex(reg: *const c_char) -> *mut c_char {
  /* ... */
}
```

Emscripten

```javascript
var Module = {
  wasmBinaryFile: 'site.wasm',
  onRuntimeInitialized: () => {
    // Wrap exported functions
    const cwrappings = {
      regex_compare:  ['regex_compare', 'bool', ['string', 'string']],
      validate_regex: ['validate_regex', 'string', ['string']],
    }

    var lib = {}

    for (var fn in cwrappings) {
      lib[fn] = Module.cwrap(...cwrappings[fn])
    }

    // Wrap regex validation
    const validate_regex = function(string) {
      const regex_err = lib.validate_regex(string)
      return regex_err === "" ? null : regex_err
    }
    /* ... */
  },
  /* ... */
}
```

Emscripten

```
var Module = {
  wasmBinaryFile: 'site.wasm',
  onRuntimeInitialized: () => {
    // Wrap exported functions
    const cwrappings = {
      regex_compare:  ['regex_compare', 'bool', ['string', 'string']],
      validate_regex: ['validate_regex', 'string', ['string']],
    }

    var lib = {}

    for (var fn in cwrappings) {
      lib[fn] = Module.cwrap(...cwrappings[fn])
    }

    // Wrap regex validation
    const validate_regex = function(string) {
      const regex_err = lib.validate_regex(string)
      return regex_err === "" ? null : regex_err
    }
    /* ... */
  },
  /* ... */
}
```

Emscripten

```
$ cat .cargo/config
[target.wasm32-unknown-emscripten]
rustflags = [
    "-Clink-args=-O3 -s EXPORTED_FUNCTIONS=['_regex_compare','_validate_regex'] -s TOTAL_MEM
]
```

Emscripten

```
$ head --lines 1 Dockerfile
FROM apiaryio/ecc
```

Emscripten

// matches ""

Wasm Bindgen

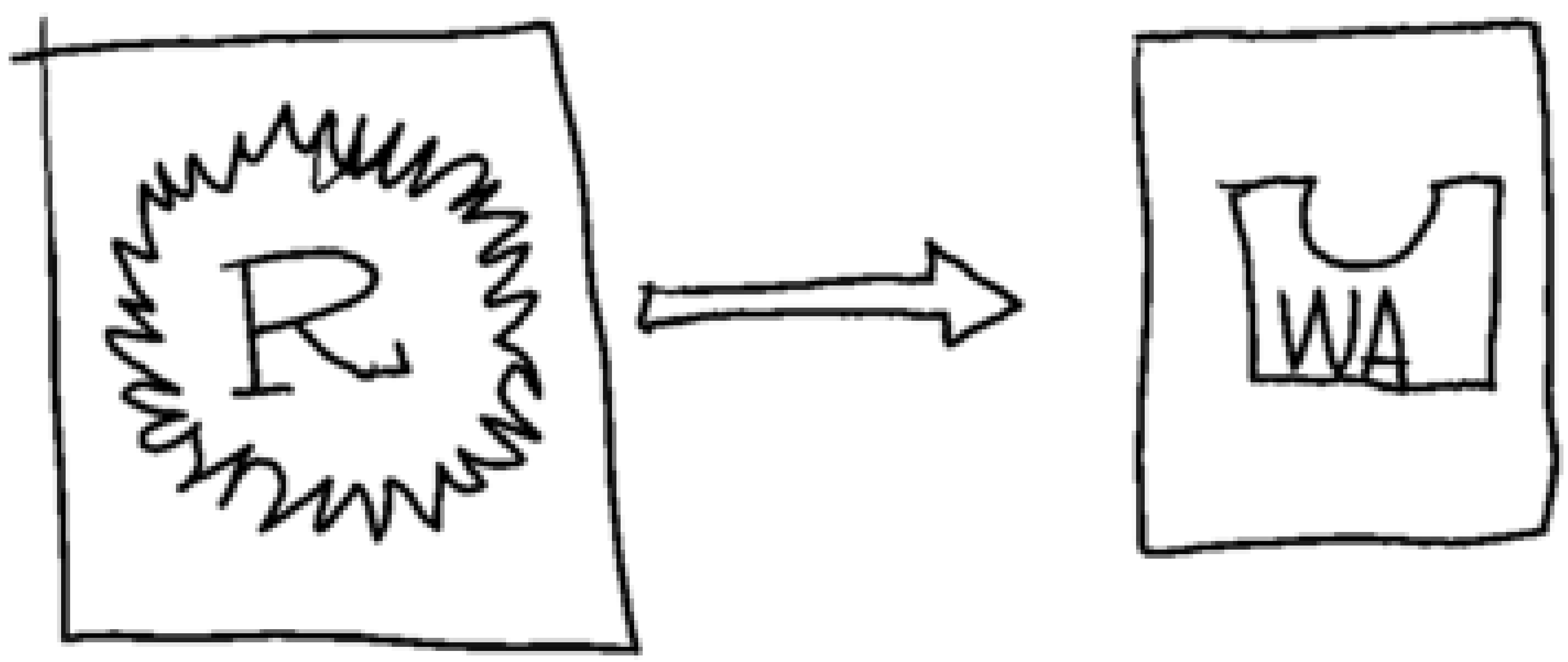
Wasm Bindgen

- github.com/alexcrichton/wasm-bindgen

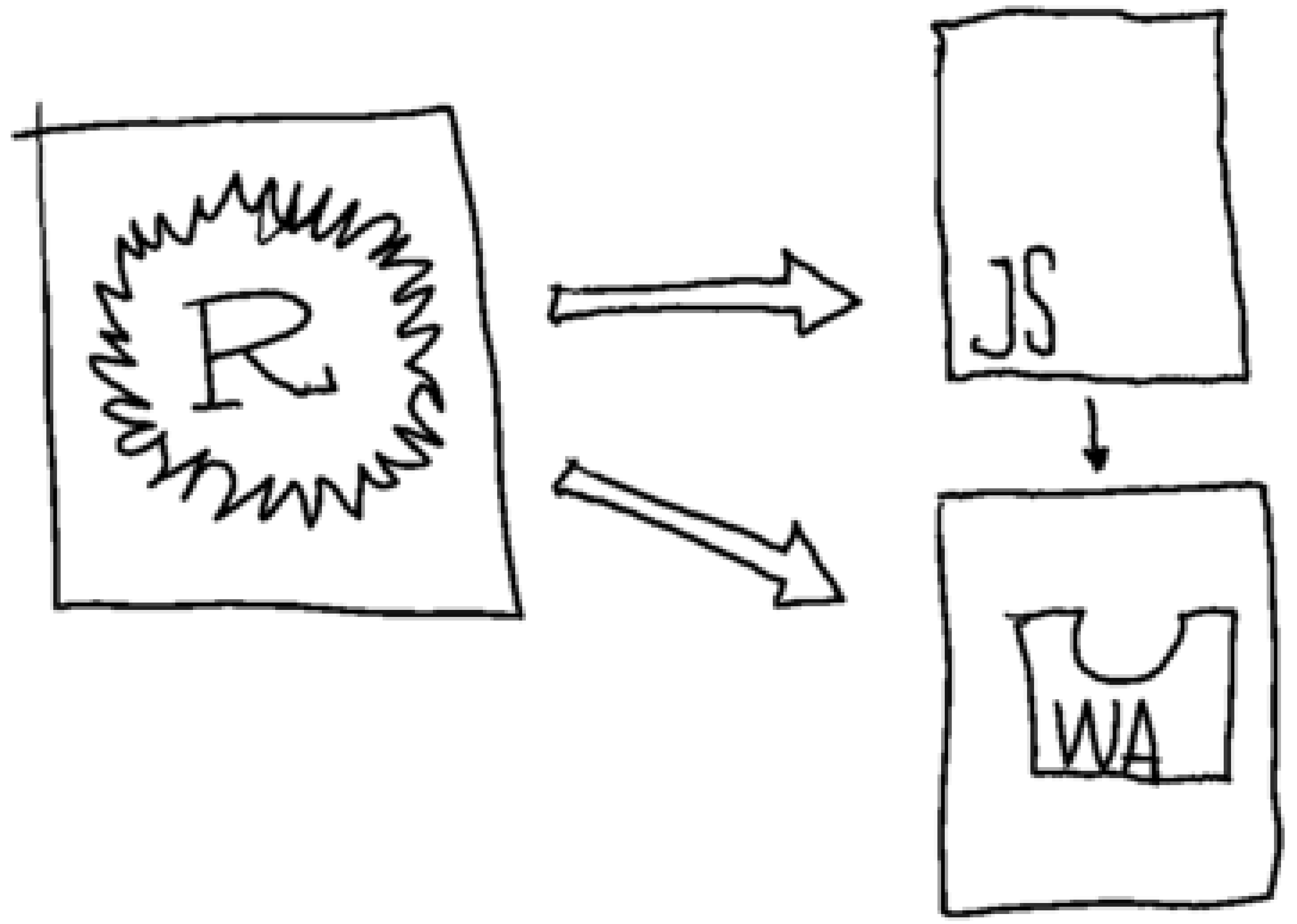
Wasm Bindgen

- github.com/alexcrichton/wasm-bindgen
- github.com/scooter-dangle/rust-wasm/tree/wasm-bindgen (`wasm-bindgen` branch is Wasm Bindgen example)

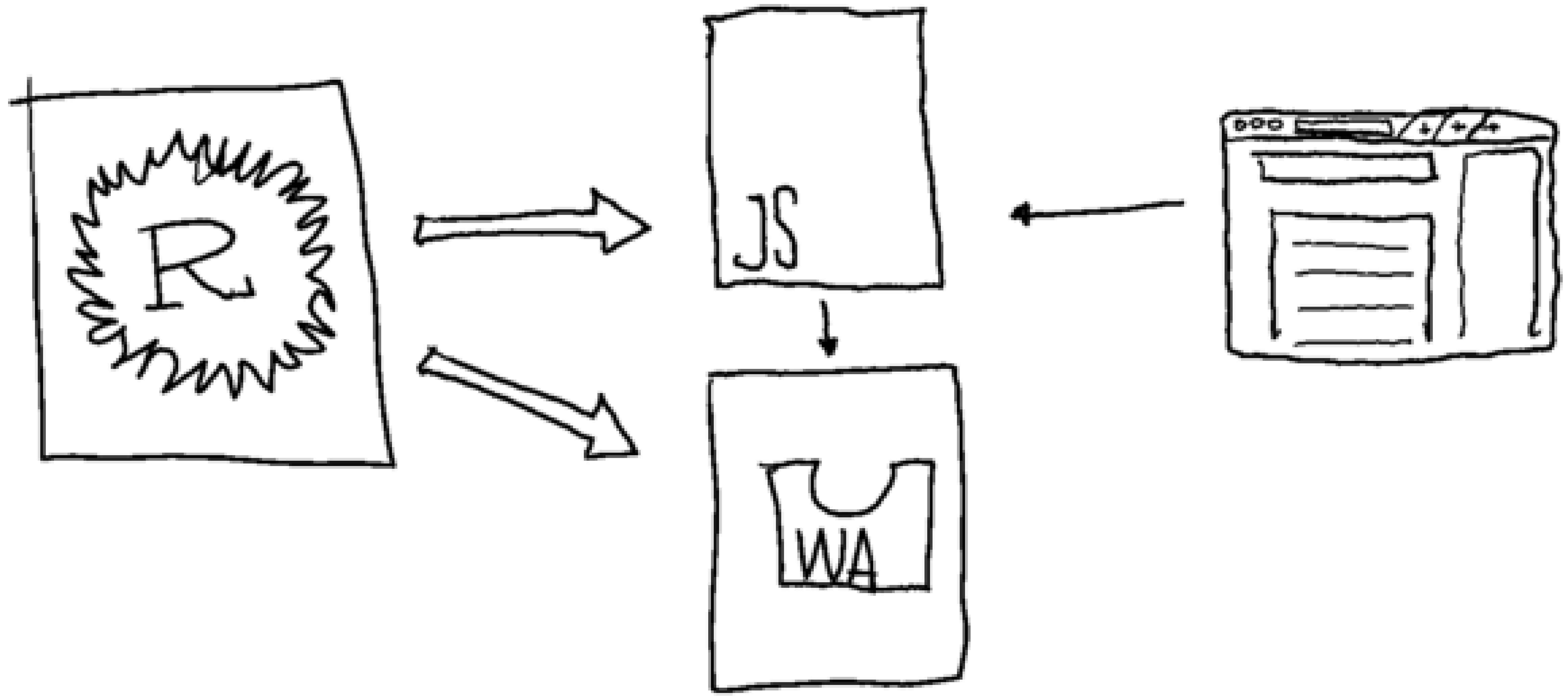
Wasm Bindgen



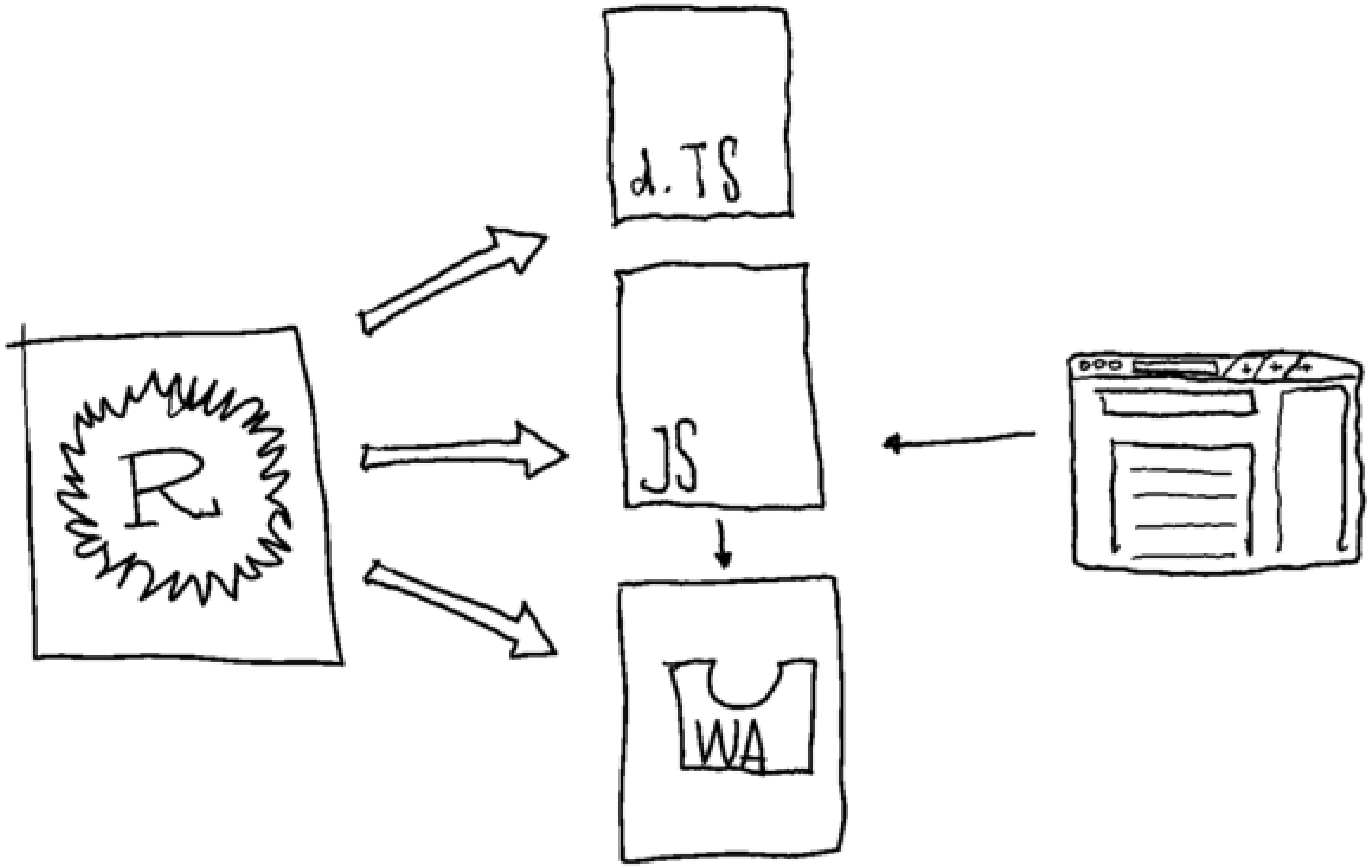
Wasm Bindgen



Wasm Bindgen



Wasm Bindgen



Wasm Bindgen

```rust
#[wasm_bindgen]
pub struct CompiledRegex {
    compiled_regex: Option<Regex>,
    pub error: String,
}
```

Wasm Bindgen

```rust
#[wasm_bindgen]
impl CompiledRegex {
    pub fn new(reg: &str) -> CompiledRegex {
        let (compiled_regex, error) = match Regex::new(reg) {
            Ok(reg) => (Some(reg), "".into()),
            Err(error) => (None, error.description().into()),
        };

        Self { compiled_regex, error }
    }

    pub fn is_match(&self, target: &str) -> bool {
        self.compiled_regex
            .as_ref()
            .map(|reg| reg.is_match(target))
            .unwrap_or(false)
    }

    pub fn is_valid(&self) -> bool {
        self.compiled_regex.is_some()
    }

    pub fn error_message(&self) -> String {
        self.error.clone()
    }
}
```

Wasm Bindgen

```
/* tslint:disable */
export class CompiledRegex {

  public ptr: number;
  constructor(ptr: number);
  free(): void;
  static  new(arg0: string): CompiledRegex;
  is_match(arg0: string): boolean;
  is_valid(): boolean;
  error_message(): string;
}
```

Wasm Bindgen

```
/* tslint:disable */
export class CompiledRegex {

  public ptr: number;
  constructor(ptr: number);
  free(): void;
  static  new(arg0: string): CompiledRegex;
  is_match(arg0: string): boolean;
  is_valid(): boolean;
  error_message(): string;
}
```

Wasm Bindgen

Pseudo index.js

Wasm Bindgen

Pseudo index.js

```javascript
import { CompiledRegex } from './rust_wasm'
import { booted } from './rust_wasm_bg'

booted.then(() => {
  // Will be caching the compiled regular expression between
  // changes. Needs to respond to the `free` function.
  var compiled_reg = {
    free: function() {},
  }

  /* ... */

  compiled_reg.free()
  compiled_reg = CompiledRegex.new(reg)

  /* ... */

  if (compiled_reg.is_valid()) {
    result.ok(reg, compiled_reg.is_match(target), target)
  } else {
    result.err(compiled_reg.error_message())
  }
})
```

Wasm Bindgen

Pseudo index.js

```
import { CompiledRegex } from './rust_wasm'
import { booted } from './rust_wasm_bg'

booted.then(() => {
  // Will be caching the compiled regular expression between
  // changes. Needs to respond to the `free` function.
  var compiled_reg = {
    free: function() {},
  }

  /* ... */

  compiled_reg.free()
  compiled_reg = CompiledRegex.new(reg)

  /* ... */

  if (compiled_reg.is_valid()) {
    result.ok(reg, compiled_reg.is_match(target), target)
  } else {
    result.err(compiled_reg.error_message())
  }
})
```

Wasm Bindgen

Pseudo index.js

```
import { CompiledRegex } from './rust_wasm'
import { booted } from './rust_wasm_bg'

booted.then(() => {
  // Will be caching the compiled regular expression between
  // changes. Needs to respond to the `free` function.
  var compiled_reg = {
    free: function() {},
  }

  /* ... */

  compiled_reg.free()
  compiled_reg = CompiledRegex.new(reg)

  /* ... */

  if (compiled_reg.is_valid()) {
    result.ok(reg, compiled_reg.is_match(target), target)
  } else {
    result.err(compiled_reg.error_message())
  }
})
```

Wasm Bindgen

```
// matches ""
```

Wasm Bindgen

Scott's great FFI strategy

Host language-sensitive C struct

Wasm Bindgen

Scott's great FFI strategy

~~Host language-sensitive C struct~~

Wasm Bindgen

Scott's great FFI strategy

~~Host language sensitive C struct~~

Opaque pointer exposing rich method sets

Other tools

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
    - Likely makes it easier to integrate Rust with larger application

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
    - Likely makes it easier to integrate Rust with larger application
    - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
    - Likely makes it easier to integrate Rust with larger application
    - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application
- Stdweb: https://github.com/koute/stdweb

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
  - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
  - Likely makes it easier to integrate Rust with larger application
  - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application
- Stdweb: https://github.com/koute/stdweb
  - Include JS in Rust

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
  - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
  - Likely makes it easier to integrate Rust with larger application
  - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application
- Stdweb: https://github.com/koute/stdweb
  - Include JS in Rust
  - Less assistance for setting up correctly typed interface (?)

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
    - Likely makes it easier to integrate Rust with larger application
    - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application
- Stdweb: https://github.com/koute/stdweb
    - Include JS in Rust
    - Less assistance for setting up correctly typed interface (?)
- Wargo

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
    - Likely makes it easier to integrate Rust with larger application
    - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application
- Stdweb: https://github.com/koute/stdweb
    - Include JS in Rust
    - Less assistance for setting up correctly typed interface (?)
- Wargo
    - Encountered same issues running on macOS as when attempting to install Emscripten locally

Other tools

- Webpack 4: https://medium.com/webpack/webpack-4-beta-try-it-today-6b1d27d7d7e2#4523
    - As of last tinkering, produced larger build artifacts than Emscripten (but this could be due to my not know how to configure correctly)
    - Likely makes it easier to integrate Rust with larger application
    - Based on the scuttlebutt, this and/or Parcel will become the default means of including Rust in a client-side web application
- Stdweb: https://github.com/koute/stdweb
    - Include JS in Rust
    - Less assistance for setting up correctly typed interface (?)
- Wargo
    - Encountered same issues running on macOS as when attempting to install Emscripten locally
    - Does not appear to be under active development any longer

Final comparison

Final comparison

- Use wasm-bindgen to play around for now, but wait until it runs on `stable` to use in production

Final comparison

- Use wasm-bindgen to play around for now, but wait until it runs on `stable` to use in production
- Wade through Emscripten for now, but use Docker as soon as you start having trouble installing it

Other tools (cont'd)

- WebAssembly Studio: https://webassembly.studio

Other tools (cont'd)

- WebAssembly Studio: https://webassembly.studio
  - WAT

Demos and hacking