WASM, Rust, and the state of Async/Await

Isaac Clayton

What is this talk about?

1. A Quick **Refresher on WebAssembly** (WASM)
    a. What is it **Useful for**?
    b. How do I use it with **Rust**?
2. Using WASM for **More than just the Web**
    a. WebAssembly – Web = **Assembly!?**
    b. V8, **Isolates**, and Networking on the Edge
3. Async, Await, and Other **Models of Concurrency**
    a. Concurrency **!= Parallelism**
    b. **Mapping** Rust's **Concurrency Model** to JS
4. Building a Website with **Rust, WASM, and Workers**
    a. About **slightknack.dev**
    b. A Primer on **CloudFlare Workers**
    c. **Programming** Time!
5. **Key Takeaways**

Who am I?

- My name is **Isaac Clayton**
- I live in Rome, Italy – born in Provo, Utah
- I'm 16 years old
    - I've been programming for about 7 years
- My languages of choice are:
    - **Rust**, Python, Go, etc.
    - Also Passerine
- Problem-areas I'm interested in:
    - Reinforcement Learning
    - **Programming Language Design**
    - Graphics Programming

Let’s get started!

1. A Quick Refresher on WebAssembly (WASM)

WebAssembly (**WASM**) is an **open standard** that defines a **portable binary-code format** as well as **interfaces** for interactions between **programs** and their **host environment**.

— *Wikipedia*

What is it?

Two things I want to highlight:

1. Open, portable **binary-code** format
2. Interactions between **programs** and **their host environment**.

What is it?

1. Open, portable **binary-code** format
    a. It’s binary!
    b. And it runs everywhere!
    c. I can not underscore how useful this is!

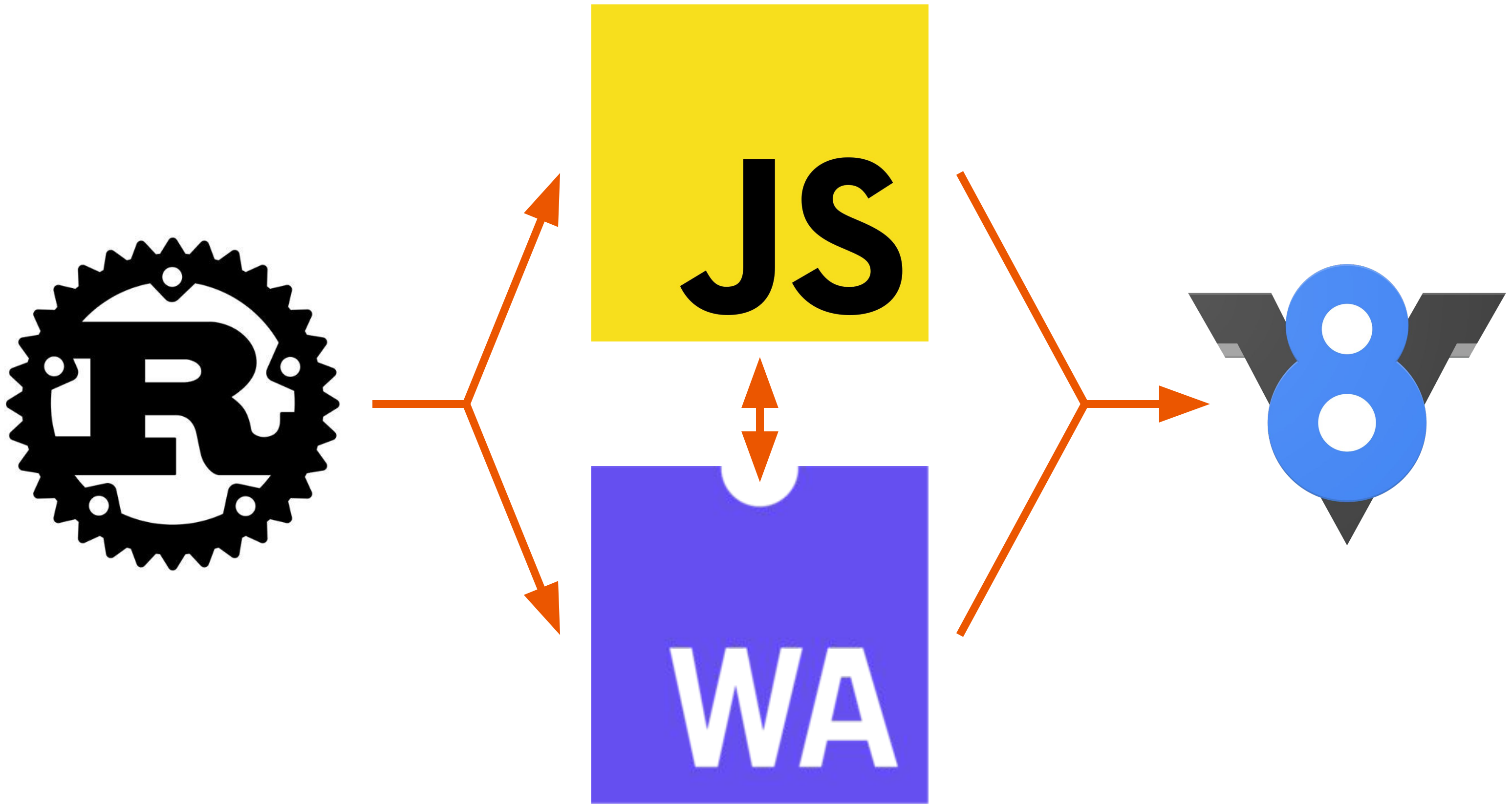
What is it?

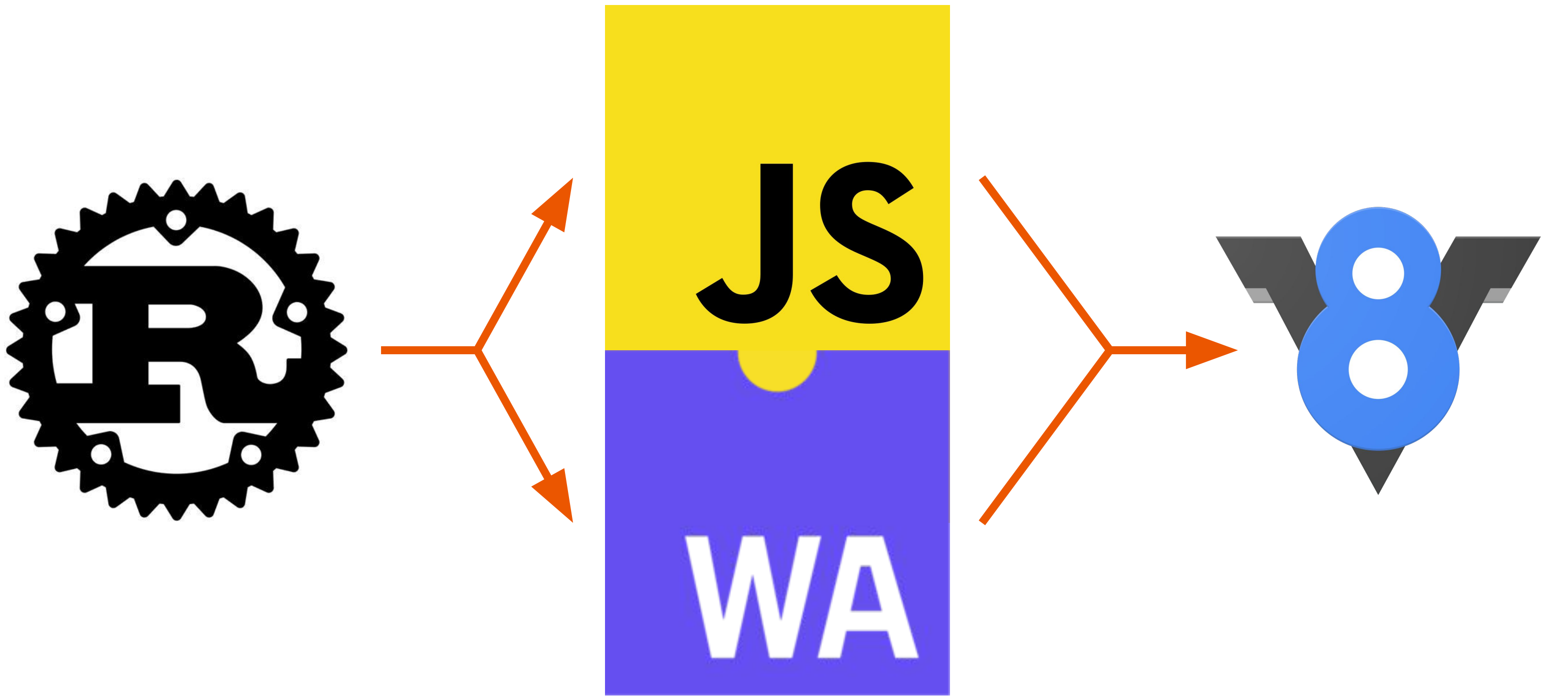
2. Interactions between **programs** and **their host environment**.
    a. What sort of interactions 🤔 – we'll get to this 😉
    b. No specific host environment
        i. (for more than just the web?)

How do I use WASM with Rust?

1. **rustc** targets **wasm32-unknown-unknown**
2. This creates a WASM library and potentially some JS bindings
3. This library can be called from JS

Tools exist to automate different parts of this process – we’ll be using one called **wrangler** which also handles deployment.


JS
WA


JS
WA

What is it useful for?

…

“Cross-platform applications built on web technologies are bloated, slow and awful…”

— *Everyone*

This doesn’t have to be the case!

What is it useful for?

…

What is it useful for?

1. Native code on the web
    a. No more JavaScript
    b. Use whatever language you need

But WASM is just a general portable binary format, right?

2. Using WASM for More than just the Web

What is it useful for?

1. Native code on the web
    a. No more JavaScript
    b. Use whatever language you need
2. ...

What is it useful for?

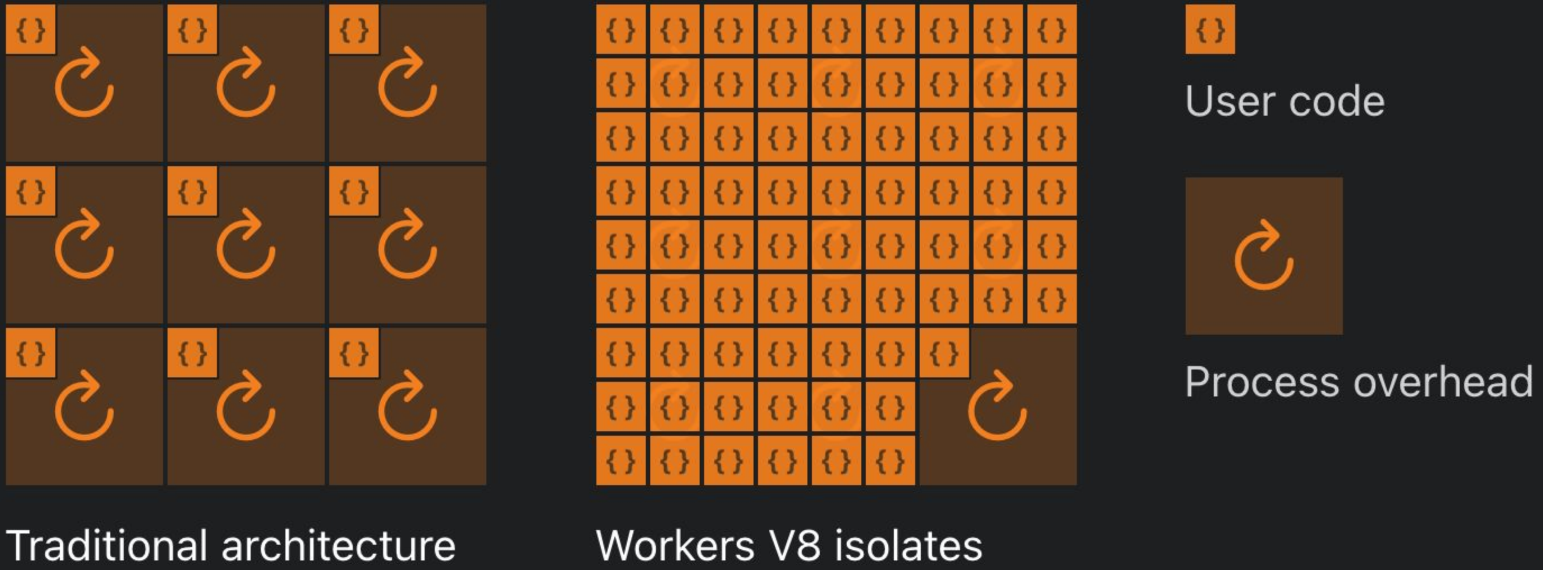
1. Native code on the web
    a. No more JavaScript
    b. Use whatever language you need
2. **Native code… anywhere?**
    a. **Mobile**
    b. **Desktop**
    c. **Serverside**
    d. **Etc.**

WebAssembly – **Web** = **Assembly**!?

Crucially, WASM is ‘just’ an open binary format

It’s supported on just about **every platform**

Hence, it’s a suitable cross-platform compilation target, about **as fast as native binaries**



V8, Isolates, and Networking on the Edge

3. Async, Await, and Other Models of Concurrency

Concurrency

How different interleave different series of instructions, different sets of data manipulations.

Concurrency ≠ Parallelism

- **Rendering engine:**
    - Can be written in a concurrent manner
    - **Highly parallelizable**
- **Generators:**
    - Can be written in a concurrent manner (in some languages)
    - **Not parallelizable** if each iteration depends on the previous one (i.e. fibonacci, prime sieve),
- **Both of these things are concurrent, but both aren't parallel!**

Concurrency in Rust

```rust
async fn foo(x: &str) -> &str {
   return falafel(x).await;
}

foo("Naan").await;
```

Concurrency in JS

```
async function foo(x) {
  return await falafel(x);
}

await foo("Naan");
```

Mapping Rust's Concurrency Model to JS

```
#[wasm_bindgen]
extern "C" {
    #[wasm_bindgen]
    pub async fn foo(x: &str) -> JsValue;
}
```

Necessary Disambiguation

This took me a while to figure out:

- **Futures** in Rust are called **Promises** in JS
- wasm_bindgen_futures::**JsFuture** converts a JS **Promise** to a Rust **Future**
    - Which can then be **await**'d
- **JsFuture**, when awaited, will return a **Result**'d js_sys::**JsValue**
- A **JsValue** can be converted into a Rust value (with e.g. **as_string**)

4. Building a Website with Rust, WASM, and Workers

About slightknack.dev

What is it?

- My personal website/knowledgebase
- Built with technologies outlined in this presentation

How does it work?

- Custom-made distributed version-controlled database (HRDB)
- CMS built-in for easy publishing

A peek at slightknack.dev

How did I make it?

1. Rust + WASM
2. Workers KV

A Primer on CloudFlare Workers

1 A *Worker* **is a serverless application** that runs on CloudFlare's edge network.

2 Workers are **pushed out to *all* edge nodes**.

3 Each ***edge node* runs thousands** of different **Worker *Isolates***.

4 *Sites* Provides a **distributed KV-store, called a *Namespace***.

Let’s build a Website!

We’re going to cover everything necessary to:

- Run WASM locally with **Wrangler**
- Interface with JS from Rust with **wasm-bindgen**
- Call **asynchronous JS from WASM**

Programming time!

Plan of Attack

1. Making a new project with Wrangler (**wrangler.toml**)
2. Basic setup (**rust-worker-template|master**):
   a. Configuring Dependencies (**Cargo.toml**)
   b. Requests from JS → Rust/WASM (**wrangler/worker.js**)
   c. Responding to Requests (**src/lib.rs**)
   d. Allocators and Panic Hooks (**src/utils.rs**)
3. MVP (**rust-worker-demo|scrubbed**):
   a. Routing Requests (**src/lib.rs::respond**)
   b. Binding KV Namespaces (**src/kv.rs**)
   c. Parsing Markdown (**src/markdown.rs**)
   d. Editing the Page (**src/lib.rs::update** & **src/form.rs**)
4. Let’s run it!

Running our Worker: `wrangler dev`

Key Takeaways

- WASM is a great new technology that will play a critical role in future projects.
    - (I might be a bit biased, haha)
- WASM can be used for more than just web-facing applications
- Rust has a great concurrency model, use it!
- Wrangler is a nice way to get started with WASM for more than just the web.

Thank You!

Elsewhere on the Internet:
@slightknack (Isaac Clayton)

Contact me:

- Email: **hello@slightknack.dev**
- Website: **slightknack.dev**
- Discord: **@slightknack#4221**

Slides can be found at:
github.com/slightknack/wasm-rust-pres

Q & A