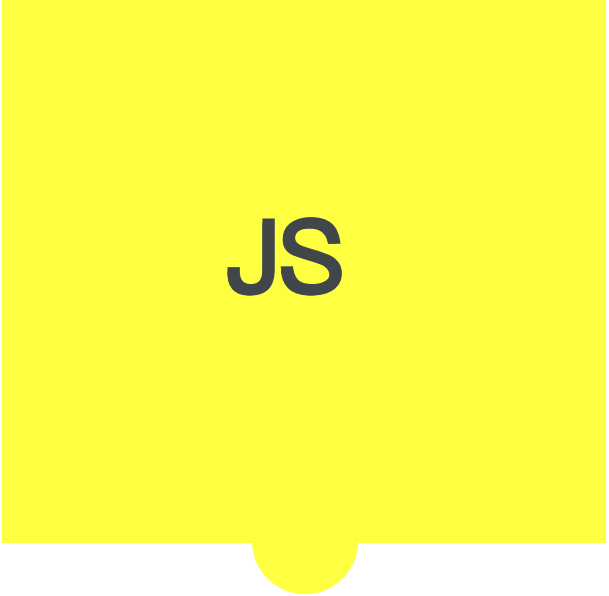
Happy path –
Getting started with
WebAssembly & Rust

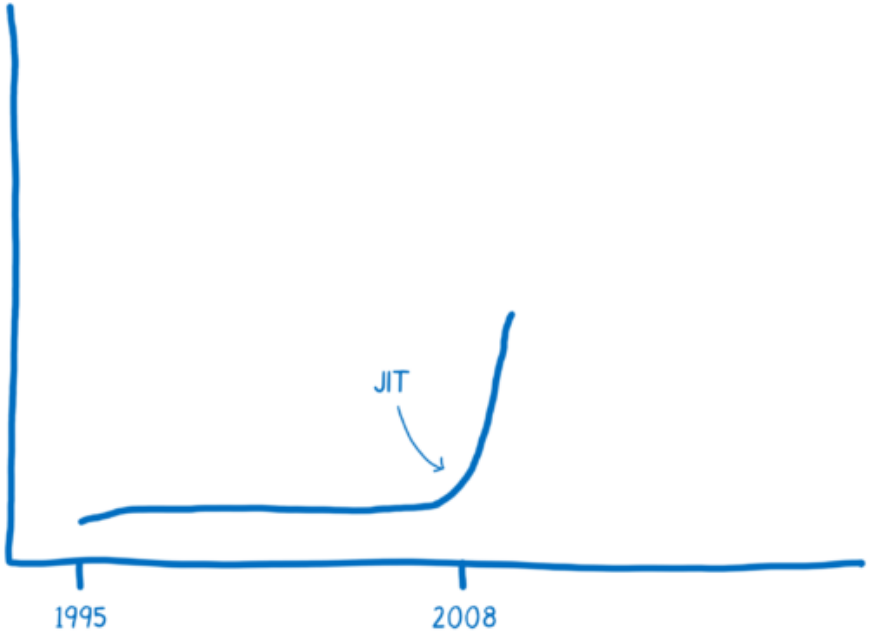


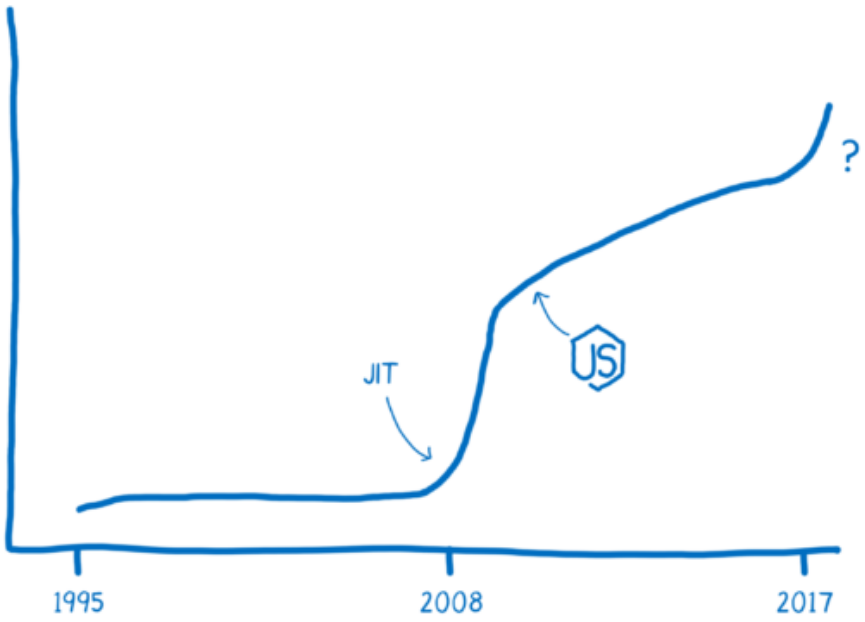
Agenda

1. Where does the JavaScript to go?
2. What is WebAssembly
3. JavaScript VS WASM
4. How to code
5. wasm-bindgen
6. Demo
7. What does the WASM can do and Not
8. More info
9. WASI
10. Q&A

Where does the JavaScript to go?

Where does the JavaScript to go?





What is WebAssembly

What's WebAssembly

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

WEBASSEMBLY

https://webassembly.org

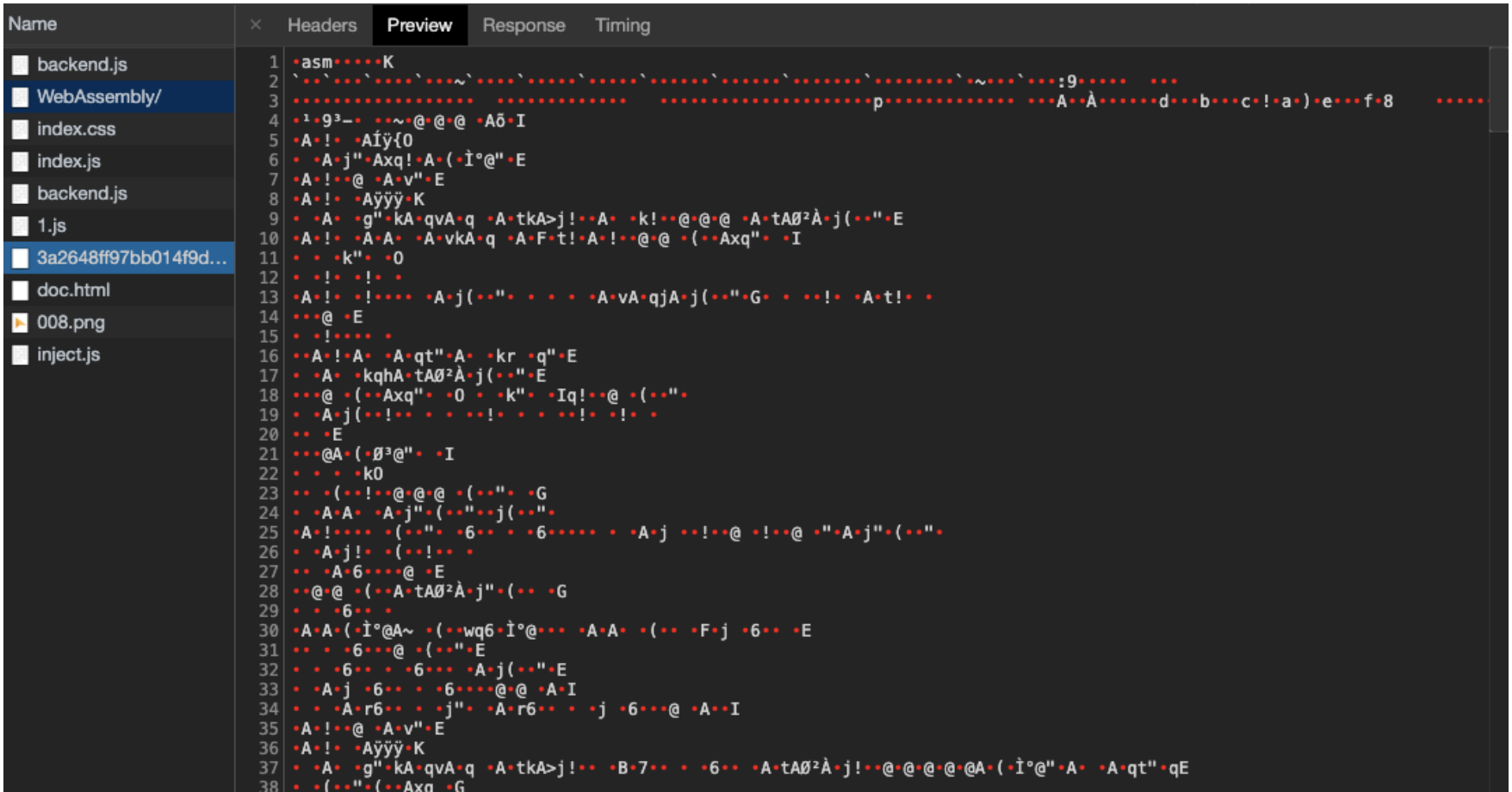
What's WebAssembly

WebAssembly (abbreviated *Wasm*) is a **binary instruction** format for a **stack-based virtual machine**. Wasm is designed as a portable target for **compilation of high-level languages** like C/C++/Rust, **enabling deployment** on the web for client and server applications.

WEBASSEMBLY

https://webassembly.org

What's WebAssembly



JavaScript VS WASM

JavaScript vs WASM

- fetch
- parse
- compile + optimize
- re-optimize
- execute
- garbage collection

https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

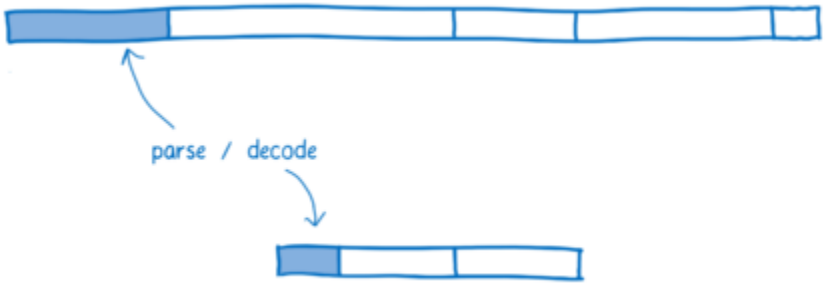
Fetching

- WebAssembly is more compact than JavaScript, Even though JavaScript use compaction algorithms to reduce the size of a bundle
- To transfer fastly in various networks with small size of WASM

Parsing

- JS :
  - Source –> AST(Abstract Syntax Tree) –> bytecode
  - JS Engein parsing what they really need to at first and just creating stubs for functions which haven't been called yet.

- WASM:
  - Not need to transform, and validated to make sure there aren't any errors in it



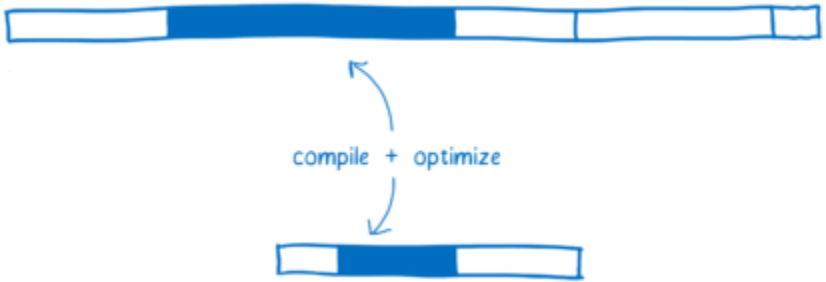
https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

Compiling + optimizing

- JS
  - Compile js **depending on what types are used at runtime**
  - **Multiple versions** of the same code may need to be compiled

- WASM
  - The compiler doesn't have to spend time running the code to observe what types are being used **before it starts compiling optimized code.**
  - The compiler doesn't have to compile **different versions** of the same code based on those different types it observes.
  - More optimizations have already been done **ahead of time in LLVM**. So less work is needed to compile and optimize it.

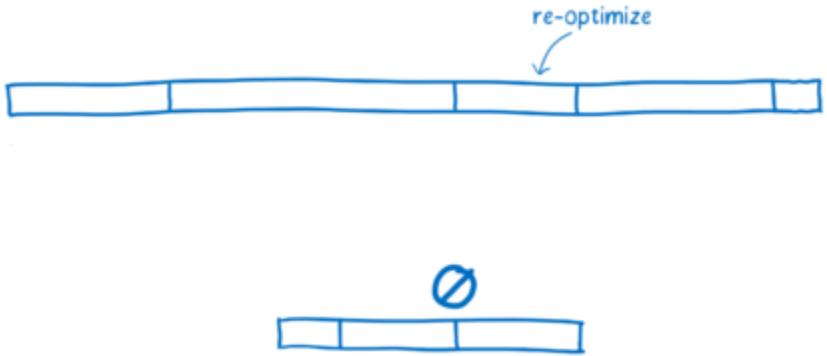
Compiling + optimizing



https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

Reoptimizing

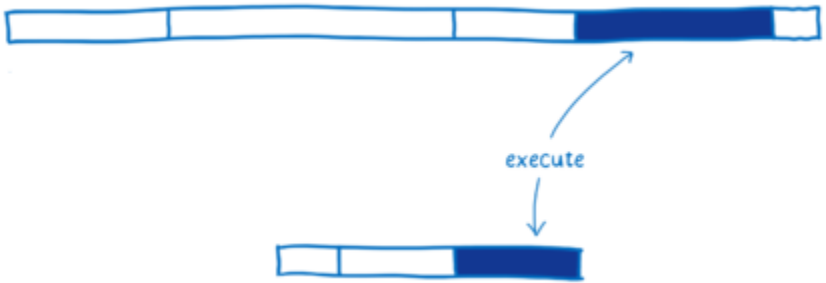
- JS
  - Sometimes the JIT has to throw out an optimized version of the code and retry it.
    - Different variables in previous iterations,
    - A new function is inserted in the prototype chain.
- WASM
  - No



https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

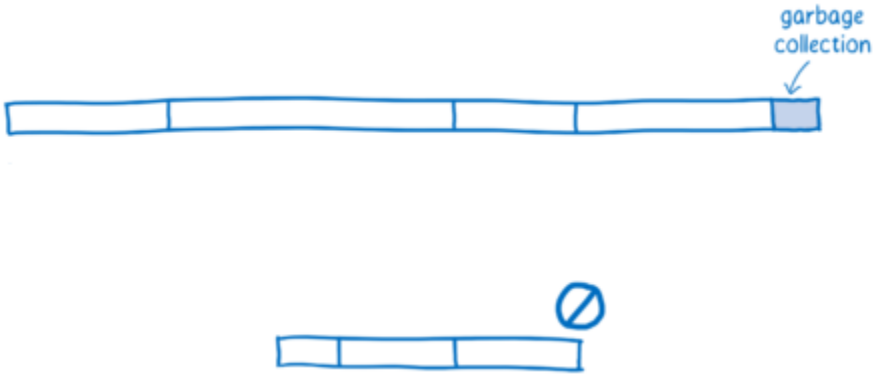
Executing

- JS
  - Need to optimize by different js type in JITs
- WASM
  - execute directly



https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

Garbage collection



https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

How to Code

How to Code – Rust

A language empowering everyone to build reliable and efficient software.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

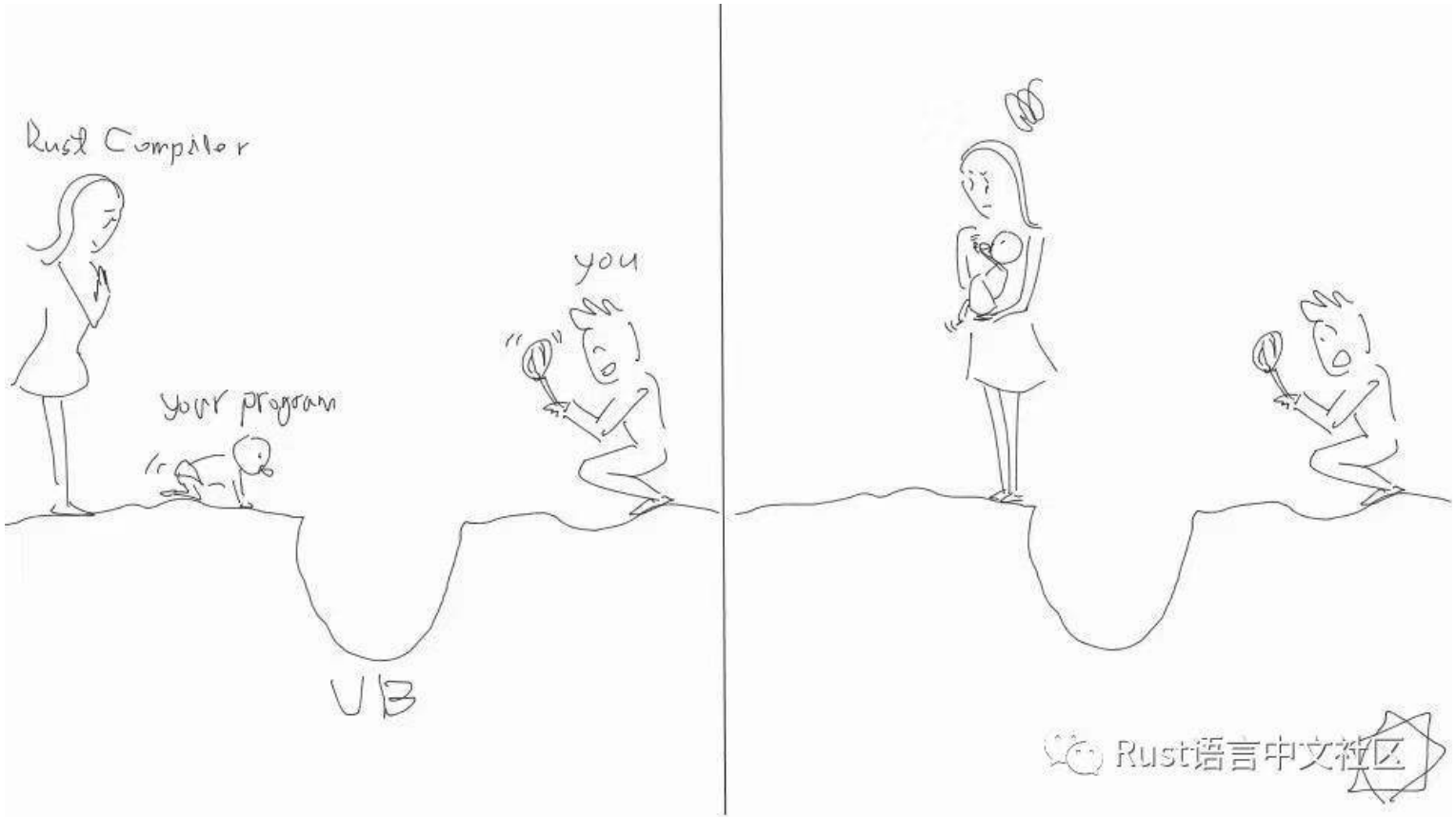
Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — and enable you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Rust – Compile



Rust – borrow ownership

```
fn print_sum (v: Vec<i32>) {
    println!("{}", v[0] + v[1]);
}
fn main () {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }    print_sum(v) ;
    println!("Compile error: {}", v);
}
```

Rust – borrow ownership

```
fn print_sum (v: Vec<i32>) {
    println!("{}", v[0] + v[1]);
}
fn main () {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }    print_sum(v) ;
    println!("Compile error: {}", v);
}
```

```
fn print_sum (v: &Vec<i32>) {
    println!(”{}”, v[0] + v[1]);
}
fn main () {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }    print_sum(&v);
    println!(”we still have v: {}”, v);
}
```

wasm-bindgen

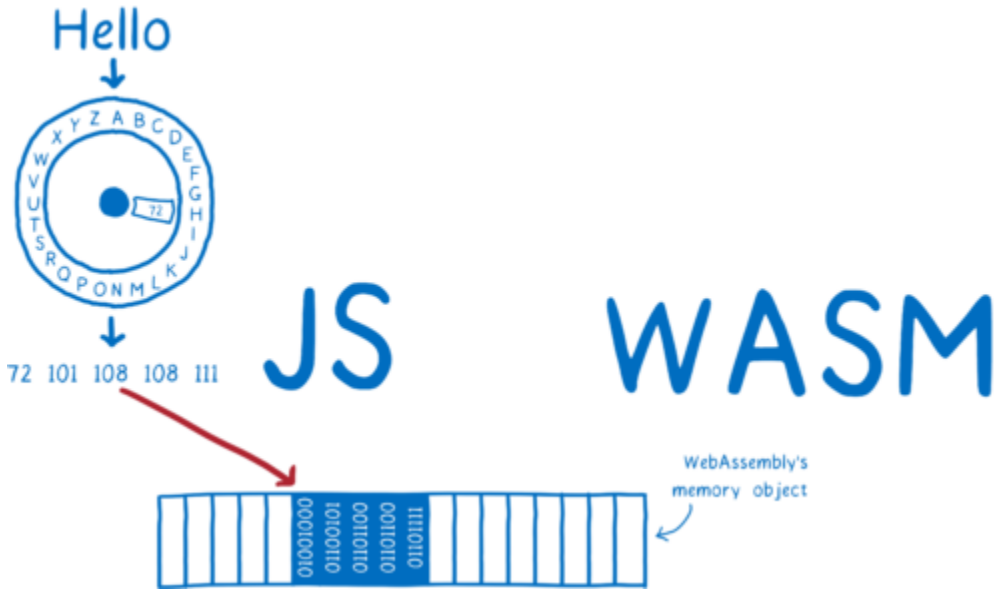
Making WebAssembly better for Rust & for all languages

https://yq.aliyun.com/articles/728341?spm=a2c4e.11155435.0.0.20633312Xvljzv

Data that pass to wasm

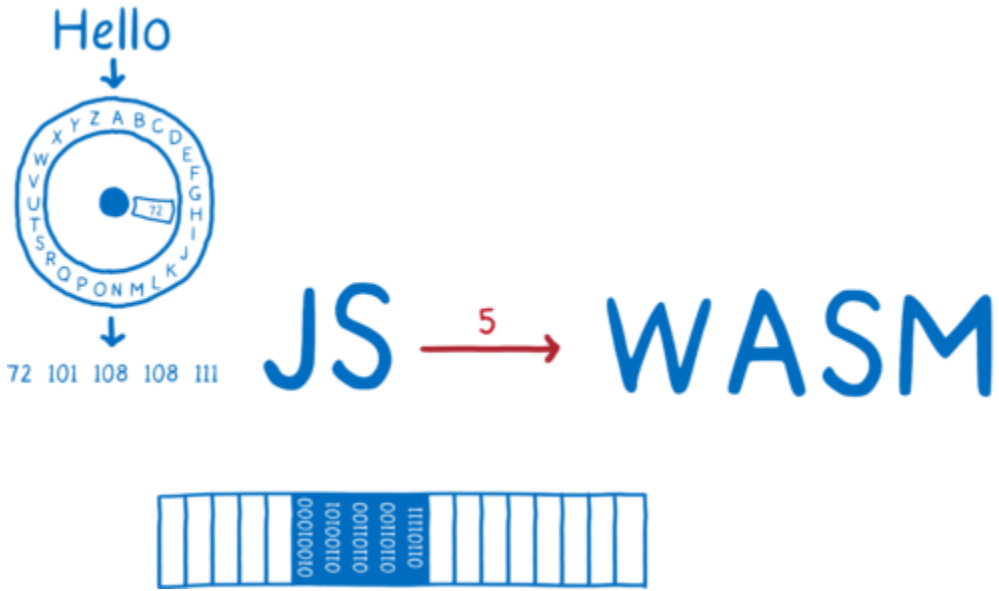
https://hacks.mozilla.org/2018/03/making-webassembly-better-for-rust-for-all-languages/

Data that pass to wasm



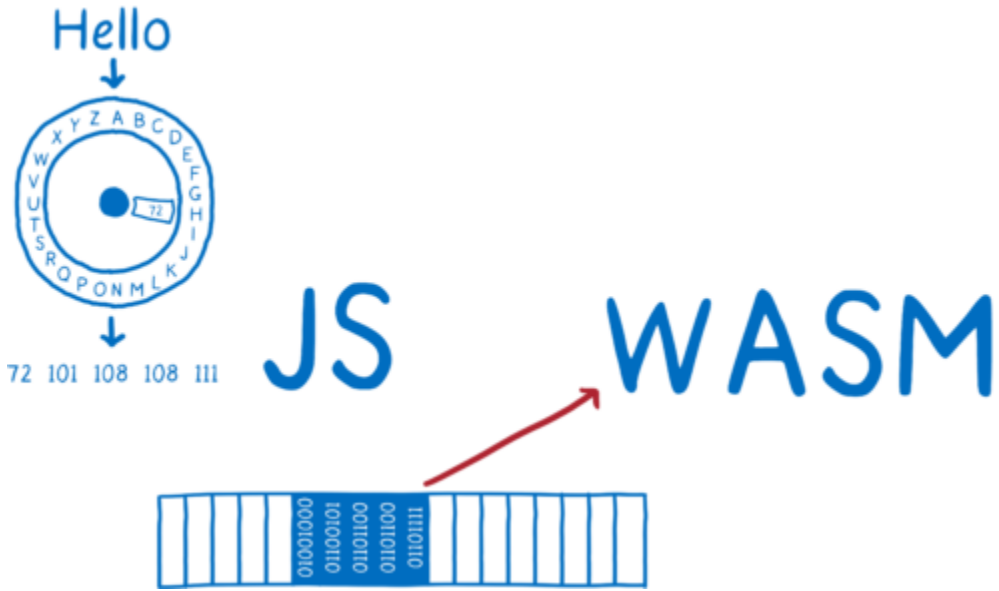
https://hacks.mozilla.org/2018/03/making-webassembly-better-for-rust-for-all-languages/

Data that pass to wasm



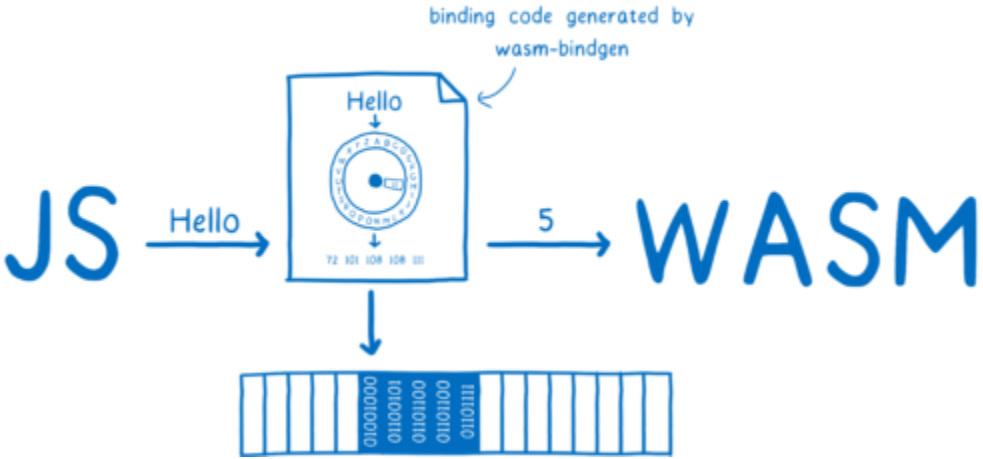
https://hacks.mozilla.org/2018/03/making-webassembly-better-for-rust-for-all-languages/

Data that pass to wasm



https://hacks.mozilla.org/2018/03/making-webassembly-better-for-rust-for-all-languages/

Data that pass to wasm



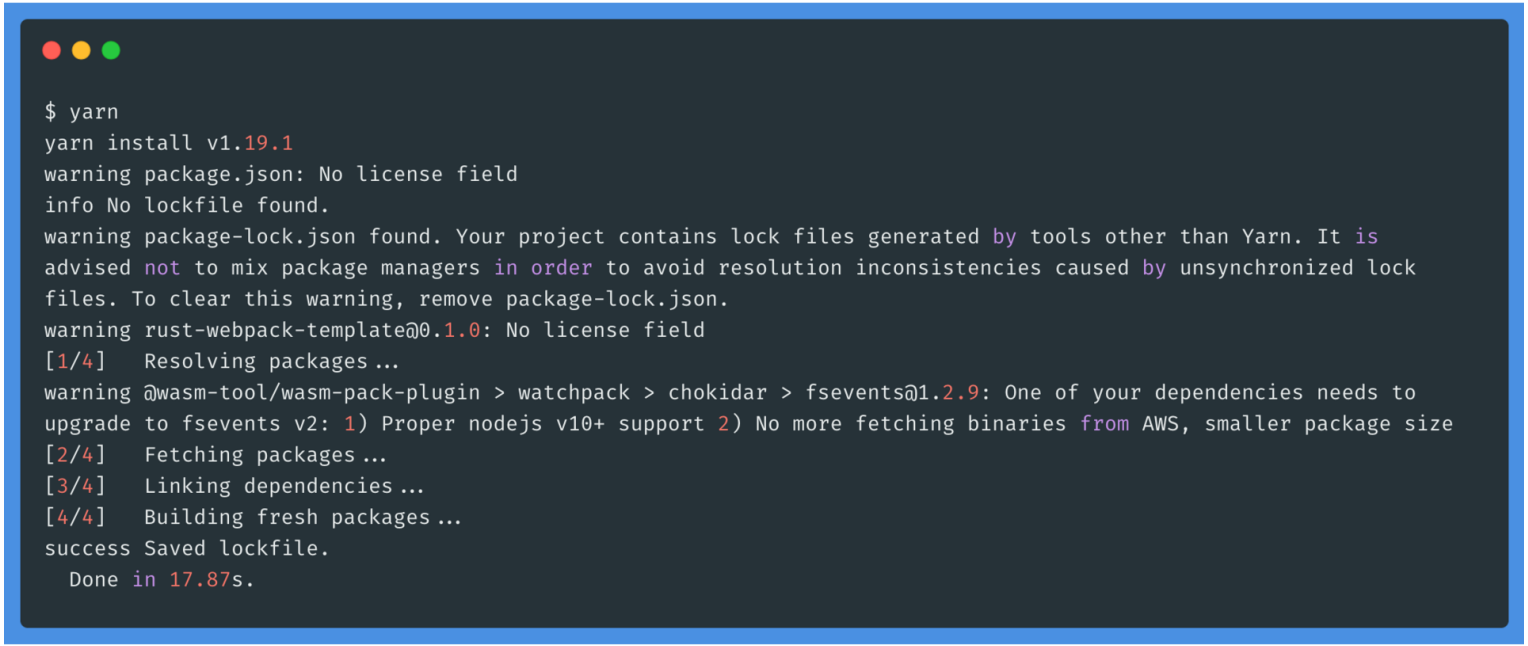
https://hacks.mozilla.org/2018/03/making-webassembly-better-for-rust-for-all-languages/

Demo

init

https://yq.aliyun.com/articles/727954?spm=a2c4e.11155435.0.0.20633312Xvljzv

yarn



```
$ yarn
yarn install v1.19.1
warning package.json: No license field
info No lockfile found.
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is
advised not to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock
files. To clear this warning, remove package-lock.json.
warning rust-webpack-template@0.1.0: No license field
[1/4]   Resolving packages ...
warning @wasm-tool/wasm-pack-plugin > watchpack > chokidar > fsevents@1.2.9: One of your dependencies needs to
upgrade to fsevents v2: 1) Proper nodejs v10+ support 2) No more fetching binaries from AWS, smaller package size
[2/4]   Fetching packages ...
[3/4]   Linking dependencies ...
[4/4]   Building fresh packages ...
success Saved lockfile.
  Done in 17.87s.
```

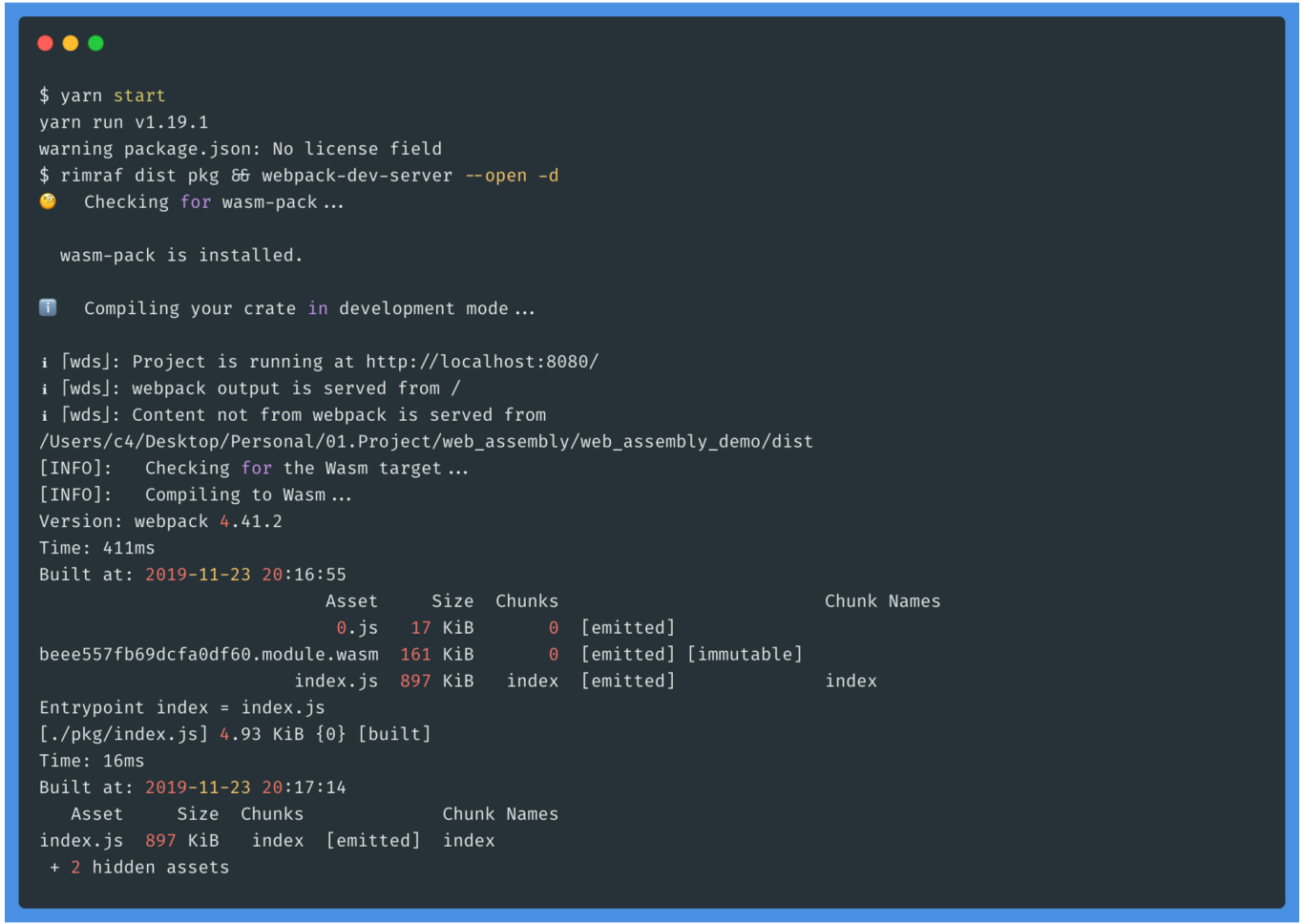
https://yq.aliyun.com/articles/727954?spm=a2c4e.11155435.0.0.20633312Xvljzv

cargo.toml

```
# You must change these to your own details.
[package]
name = "web_assembly_demo"
description = "My super awesome Rust, WebAssembly, and Webpack project!"
version = "0.1.0"
authors = ["guzhongren <guzhoongren@live.cn>"]
categories = ["wasm"]
readme = "README.md"
edition = "2018"

[lib]
crate-type = ["cdylib"]

[profile.release]
# This makes the compiled code faster and smaller, but it makes compiling slower,
# so it's only enabled in release mode.
lto = true

[features]
# If you uncomment this line, it will enable `wee_alloc`:
#default = ["wee_alloc"]

[dependencies]
# The `wasm-bindgen` crate provides the bare minimum functionality needed
# to interact with JavaScript.
wasm-bindgen = "0.2.45"

# `wee_alloc` is a tiny allocator for wasm that is only ~1K in code size
```

https://yq.aliyun.com/articles/7

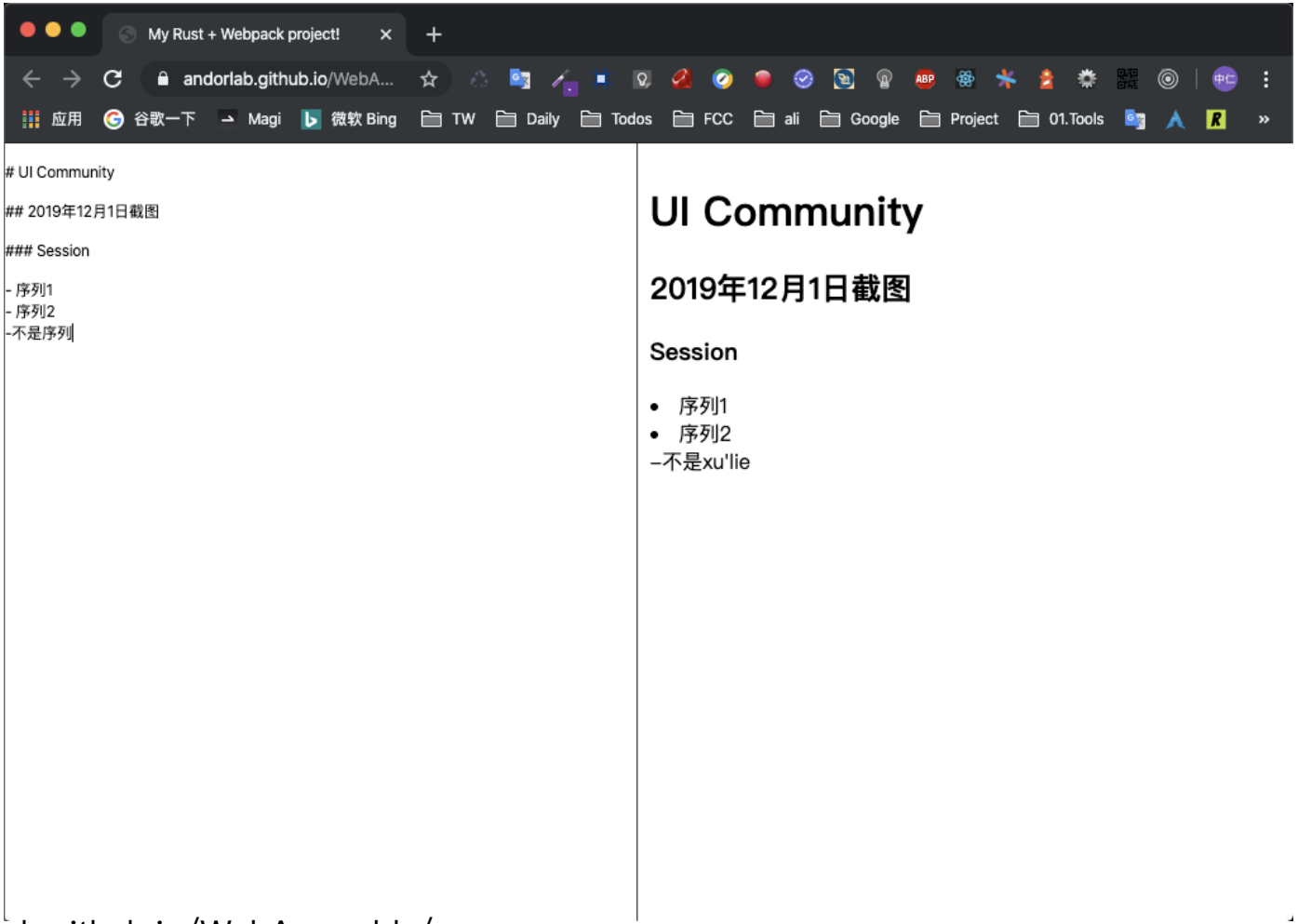
yarn start



```
$ yarn start
yarn run v1.19.1
warning package.json: No license field
$ rimraf dist pkg && webpack-dev-server --open -d
🤔   Checking for wasm-pack ...

  wasm-pack is installed.

ℹ️   Compiling your crate in development mode ...

ℹ [wds]: Project is running at http://localhost:8080/
ℹ [wds]: webpack output is served from /
ℹ [wds]: Content not from webpack is served from
/Users/c4/Desktop/Personal/01.Project/web_assembly/web_assembly_demo/dist
[INFO]:   Checking for the Wasm target ...
[INFO]:   Compiling to Wasm ...
Version: webpack 4.41.2
Time: 411ms
Built at: 2019-11-23 20:16:55
                          Asset     Size  Chunks                         Chunk Names
                           0.js   17 KiB       0  [emitted]
beee557fb69dcfa0df60.module.wasm  161 KiB       0  [emitted] [immutable]
                       index.js  897 KiB   index  [emitted]              index
Entrypoint index = index.js
[./pkg/index.js] 4.93 KiB {0} [built]
Time: 16ms
Built at: 2019-11-23 20:17:14
   Asset     Size  Chunks             Chunk Names
index.js  897 KiB   index  [emitted]  index
 + 2 hidden assets
```

https://yq.aliyun.com/articles/727954?spm=a2c4e.11155435.0.0.20633312Xvljzv

Demo

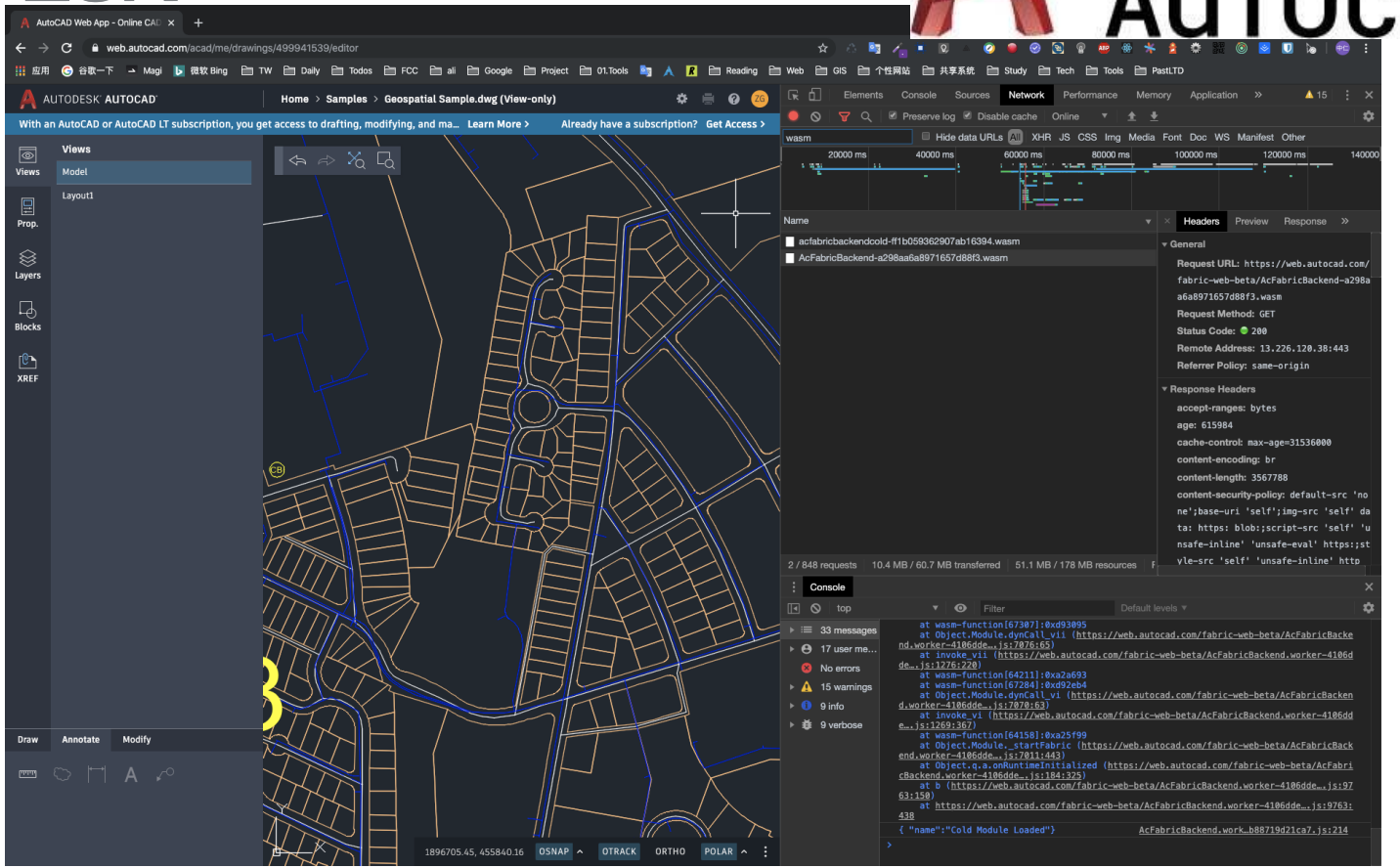


https://andorlab.github.io/WebAssembly/

What does the WASM can do and Not

AutoDESK

AUTODESK®
AUTOCAD®

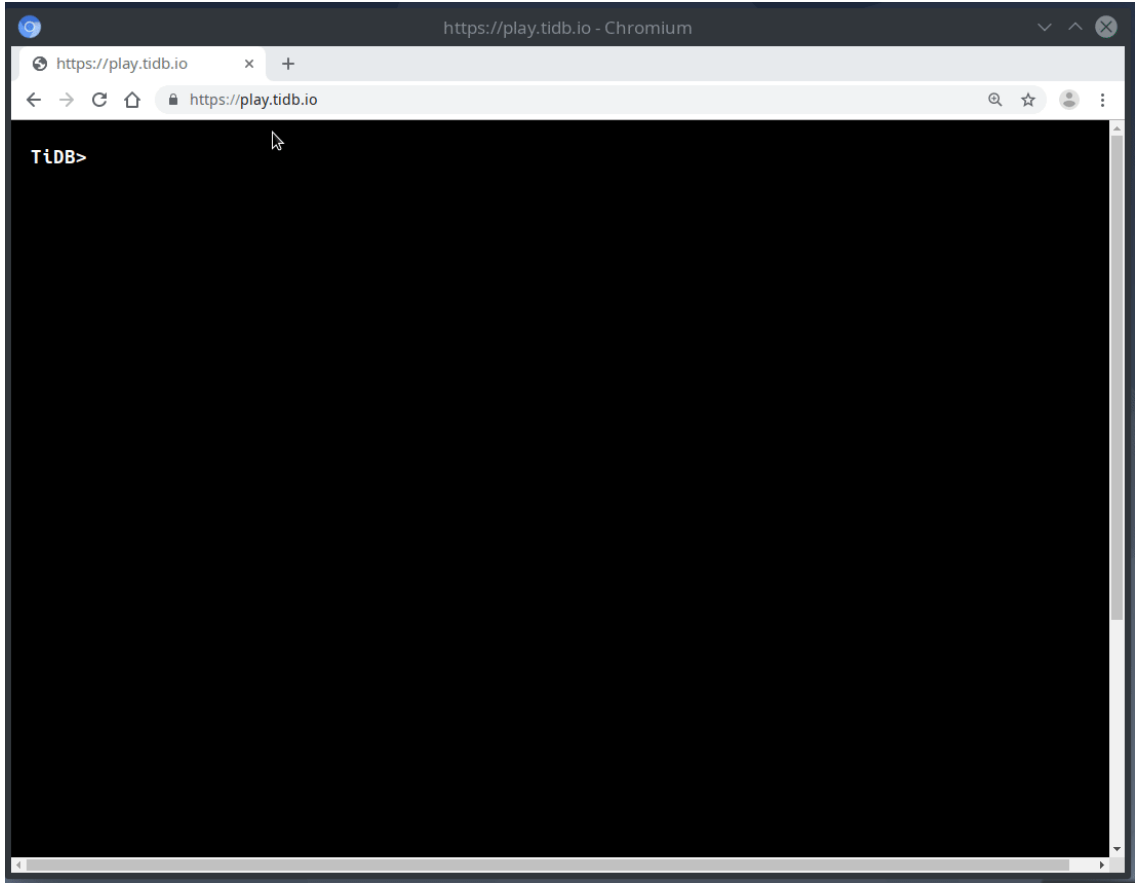


https://web.autocad.com/acad/me/drawings/499941539/editor

TiDB

$ show databases;

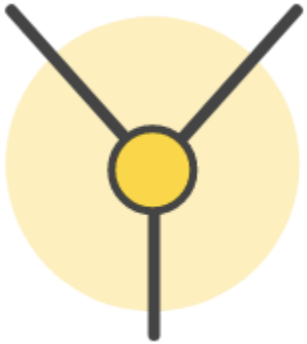
$ use test



https://play.pingcap.com/

WASM can do anything

More info



Watch 256 Star 9.4k Fork 352

https://yew.rs/docs/

https://github.com/yewstack/yew

Seed

Watch 36 Star 722 Fork 44

https://seed-rs.org/

https://github.com/seed-rs/seed

src/main.rs

```
use yew::{html, Component, ComponentLink, Html, ShouldRender};

struct Model { }

enum Msg {
    DoIt,
}

impl Component for Model {
    // Some details omitted. Explore the examples to see more.

    type Message = Msg;
    type Properties = ();

    fn create(_: Self::Properties, _: ComponentLink<Self>) -> Self {
        Model { }
    }

    fn update(&mut self, msg: Self::Message) -> ShouldRender {
        match msg {
            Msg::DoIt => {
                // Update your model on events
                true
            }
        }
    }

    fn view(&self) -> Html<Self> {
        html! {
            // Render your model here
            <button onclick=|_| Msg::DoIt>{ "Click me!" }</button>
        }
    }
}

fn main() {
    yew::start_app::<Model>();
}
```

WASI(The WebAssembly System Interface)

- Support System Call Interface for WASM virtual machine
- Let .wasm modules can be run in any WASI-compliant runtime

https://wasi.dev/

Q&A

Thanks

ThoughtWorks Zhongren.Gu