goto;

GOTO
Copenhagen 2023

#GOTOcph

WebAssembly in Production: A Compiler in a Web Page

Brian Carroll, NoRedInk

WebAssembly

How many people

- Have worked in web development?
- Have heard of WebAssembly?
- Have WebAssembly live in production?

WebAssembly (Wasm) Basics

Binary code format

Runs in browsers

JavaScript is not the only option any more

Beyond the basics

Live production web app

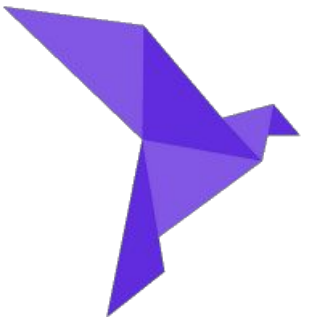
Large codebase

Advantages

Challenges

Roc

- New (pre-release) programming language
- Fast, friendly, functional
  - Aims to be faster than Go/Java/C#
  - Not as fast as C/C++/Rust
- Open source



roc-lang.org

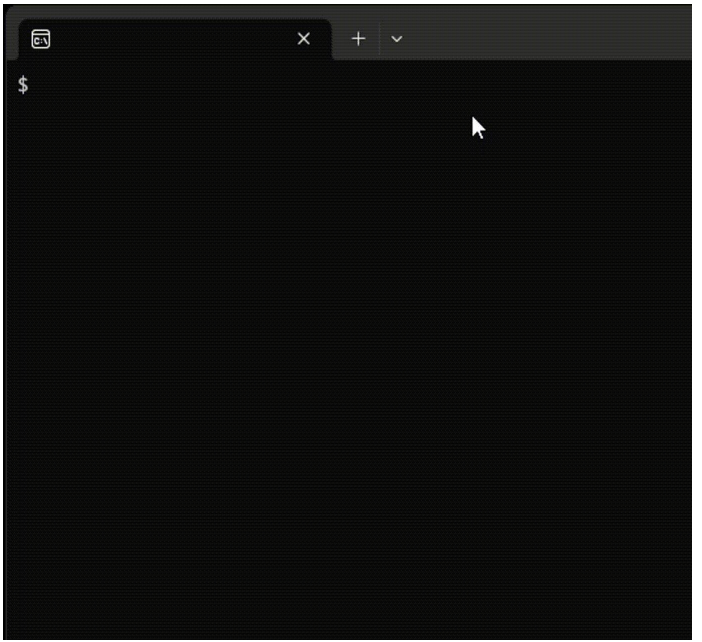
Roc REPL

Read some code the user typed

Evaluate the result

Print it out as text

Loop back to “read” again



Can we port it to WebAssembly?

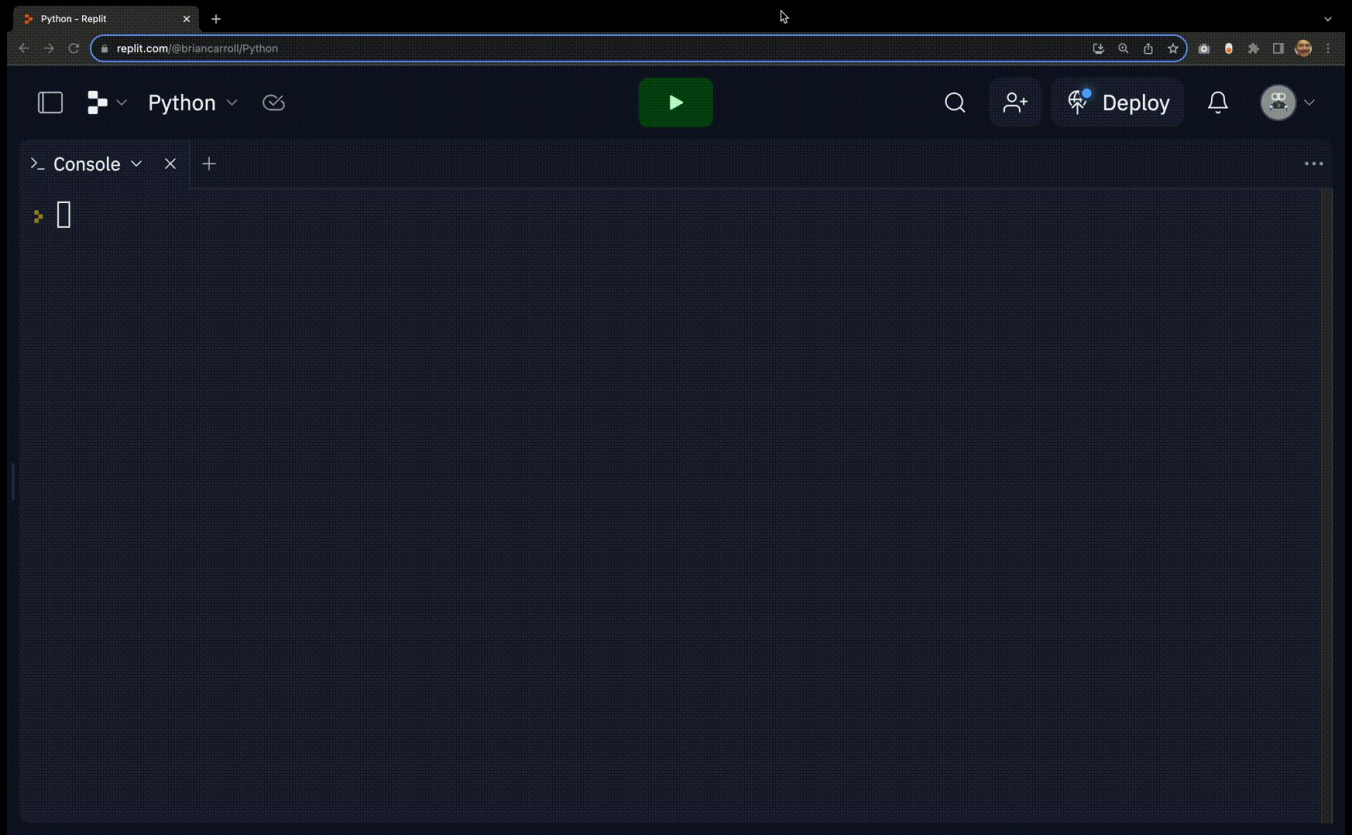
Why

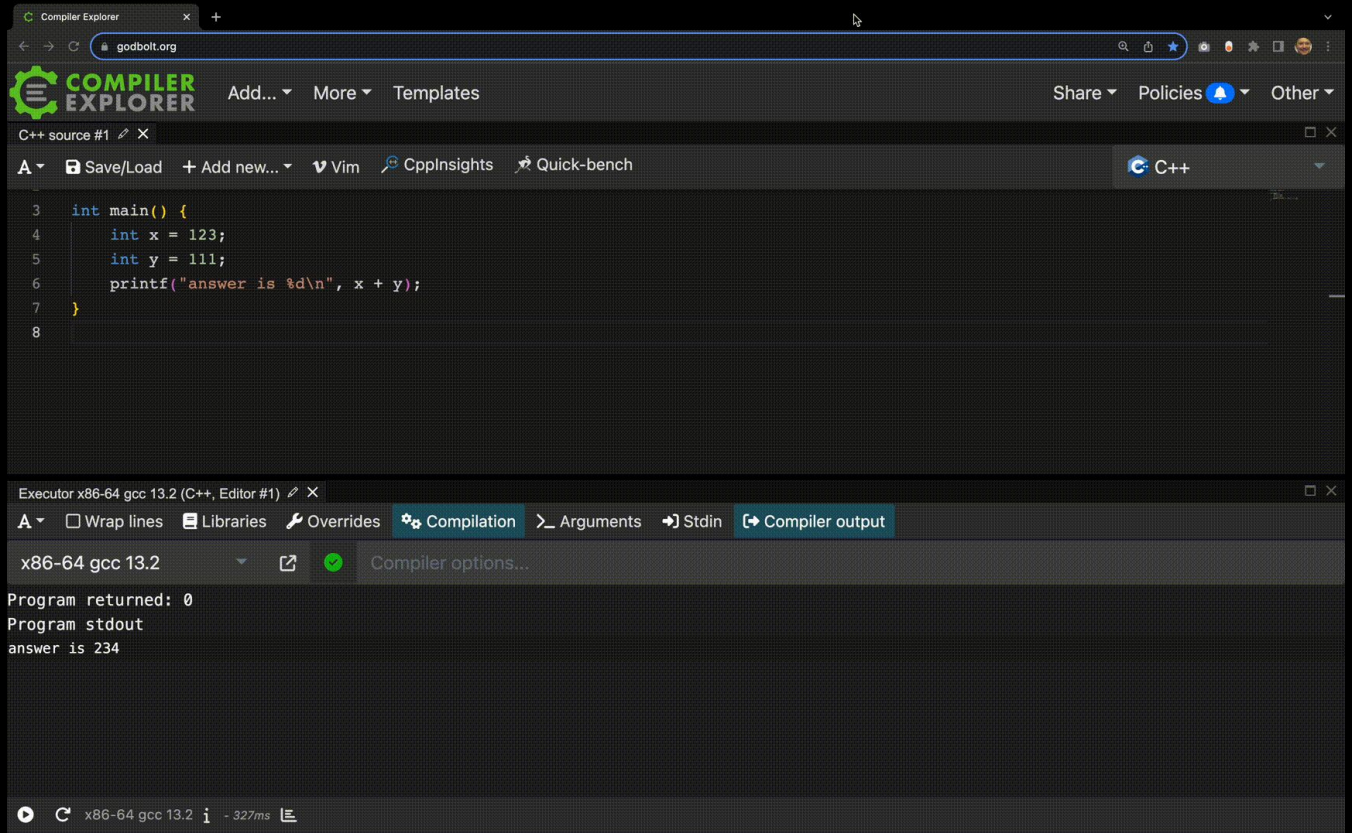
- Great for newcomers
- Static site: easy to manage

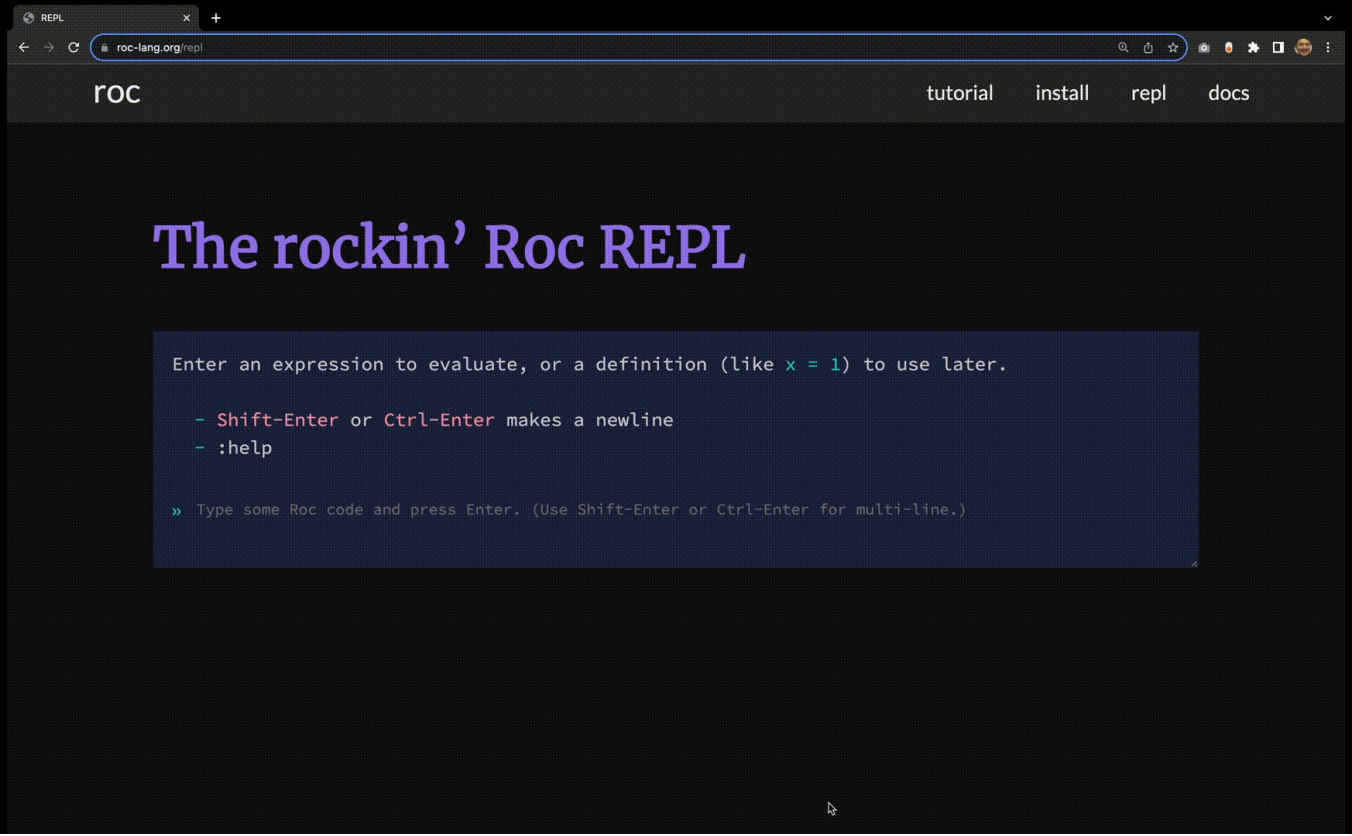
How

- Roc compiles to WebAssembly
- Compiler is Rust, which also compiles to WebAssembly!

Demo:
Online coding tools

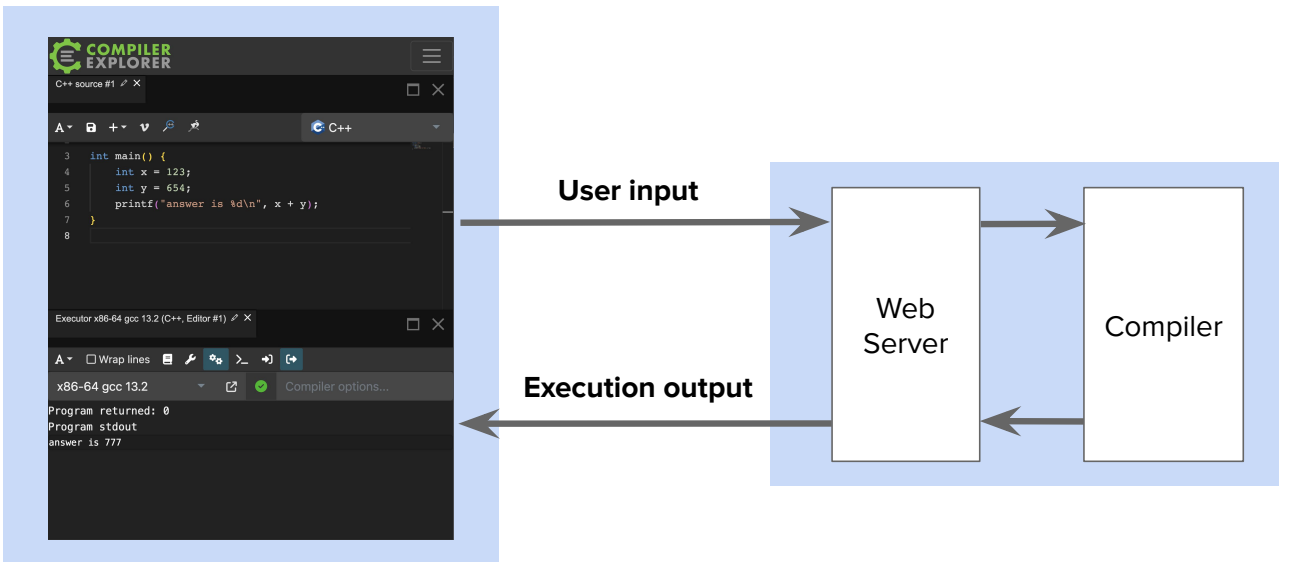

Python – Replit
replit.com/@briancarroll/Python
Python
Deploy
Console


Compiler Explorer
godbolt.org
COMPILER EXPLORER
Add...
More
Templates
Share
Policies
Other
C++ source #1
Save/Load
Add new...
Vim
CppInsights
Quick-bench
C++
int main() {
int x = 123;
int y = 111;
printf("answer is %d\n", x + y);
}
Executor x86-64 gcc 13.2 (C++, Editor #1)
Wrap lines
Libraries
Overrides
Compilation
Arguments
Stdin
Compiler output
x86-64 gcc 13.2
Compiler options...
Program returned: 0
Program stdout
answer is 234
x86-64 gcc 13.2
- 327ms


REPL
roc-lang.org/repl
roc
tutorial
install
repl
docs
The rockin' Roc REPL
Enter an expression to evaluate, or a definition (like x = 1) to use later.
- Shift-Enter or Ctrl-Enter makes a newline
- :help
» Type some Roc code and press Enter. (Use Shift-Enter or Ctrl-Enter for multi-line.)

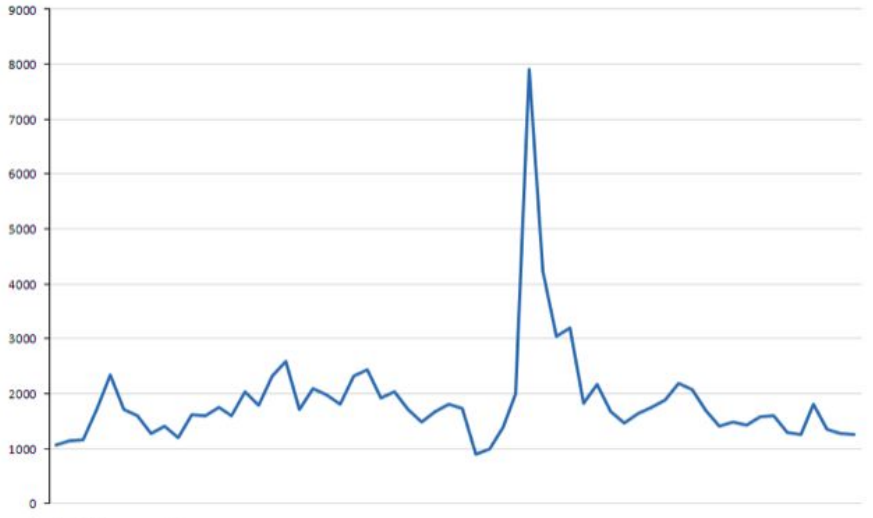
How do they work?

Typical Architecture



HTTP or WebSocket

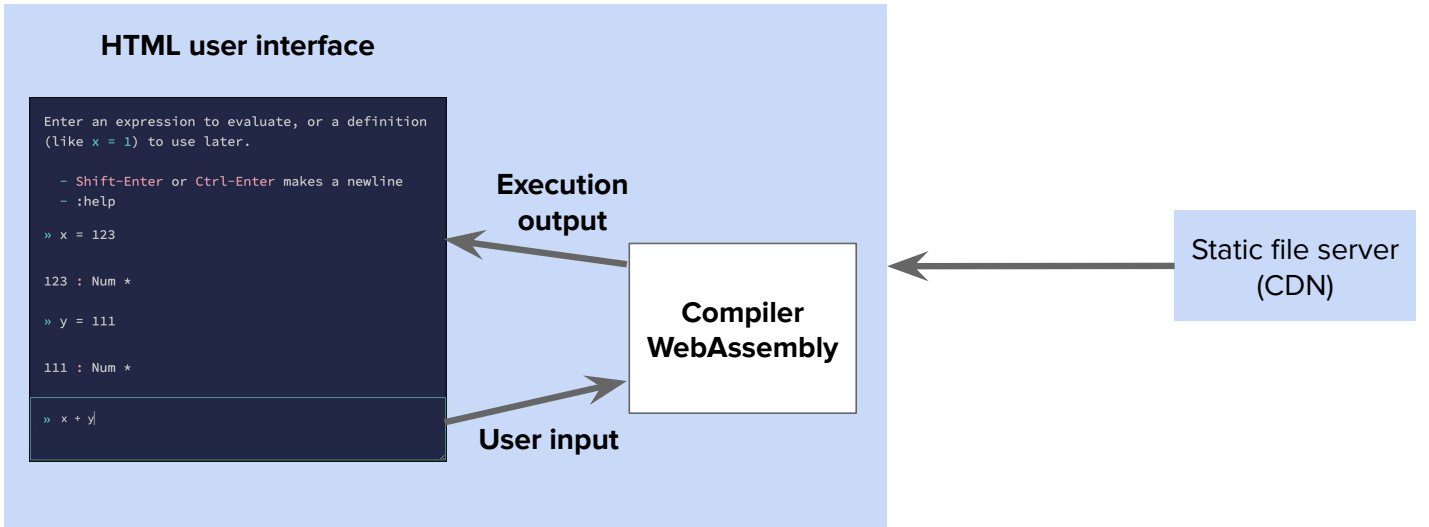
Managing servers?



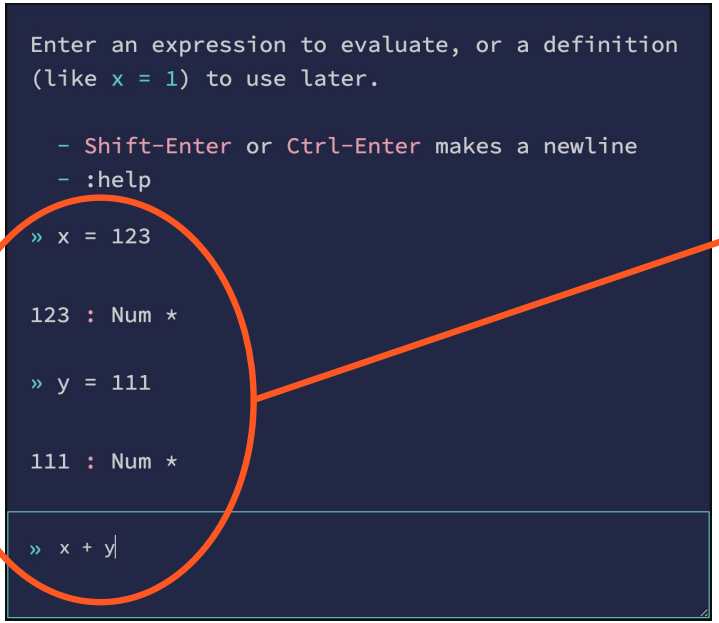




Architecture with WebAssembly



Building a REPL from a compiler




```
main =
    x = 123
    y = 111
    x + y
```

Create a “file” from input history
New “file” every time
Compile it & run it

Challenges

Download size far too big!

- 300k lines of code
- 66MB binary

WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

Challenge: download size

Challenges

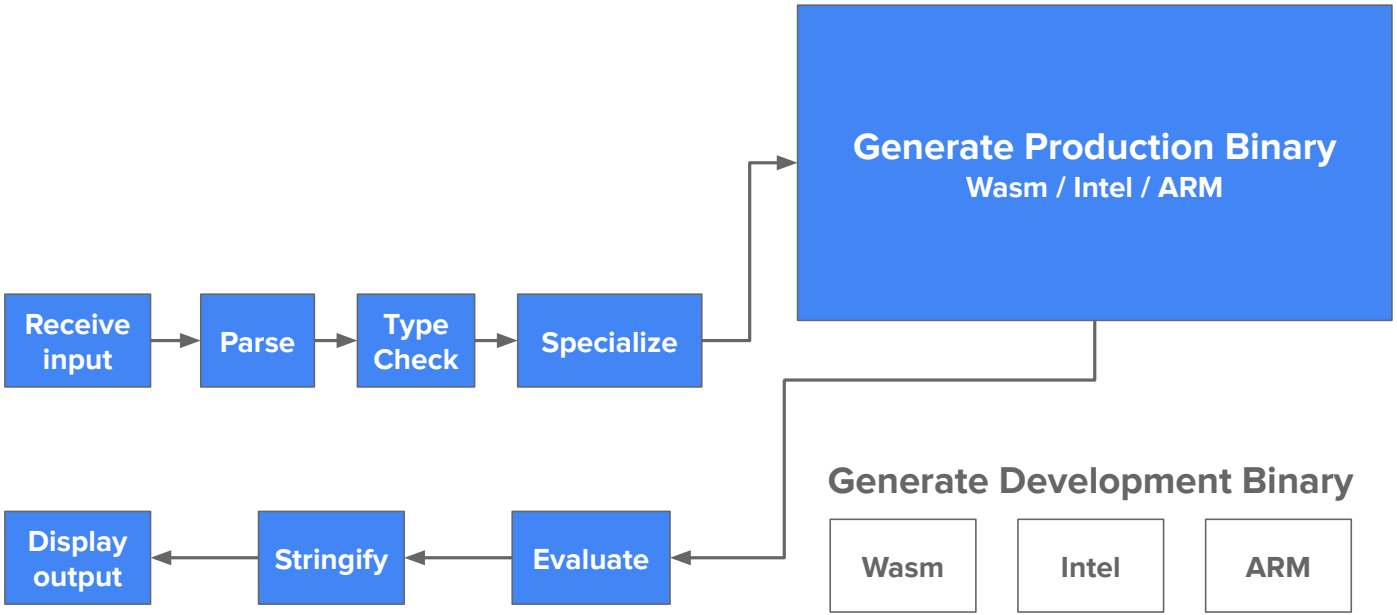
Download size far too big!

- 300k lines of code
- 66MB binary

WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

Terminal REPL



Production

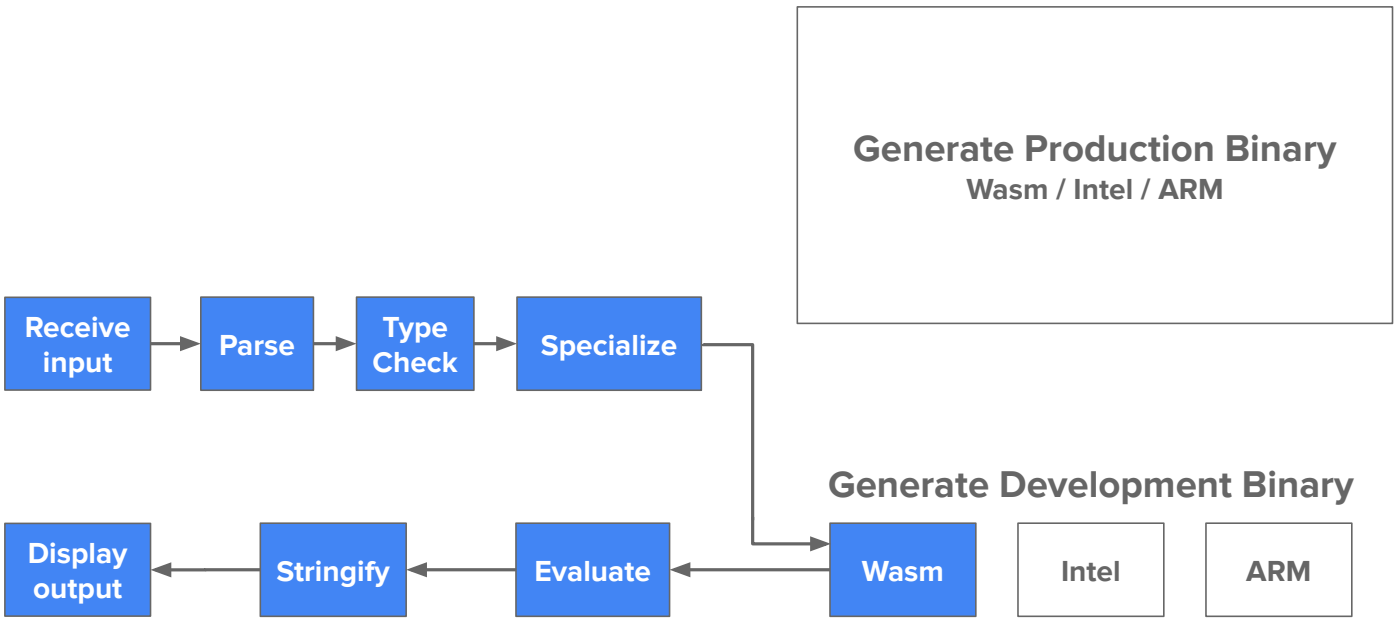
- Compile once
- Run many times
- Fast run time is important
- Optimizations are worth it
  - Longer compile time
  - Faster compiled program

Development

- Compile many times
- Run once or twice
- Fast *total* time is important
- No optimizations
  - Fastest compile time

Best for REPL
(new program for each input line)

Web REPL


Generate Production Binary
Wasm / Intel / ARM
Receive input
Parse
Type Check
Specialize
Generate Development Binary
Display output
Stringify
Evaluate
Wasm
Intel
ARM

Create an alternative entrypoint

Standard compiler

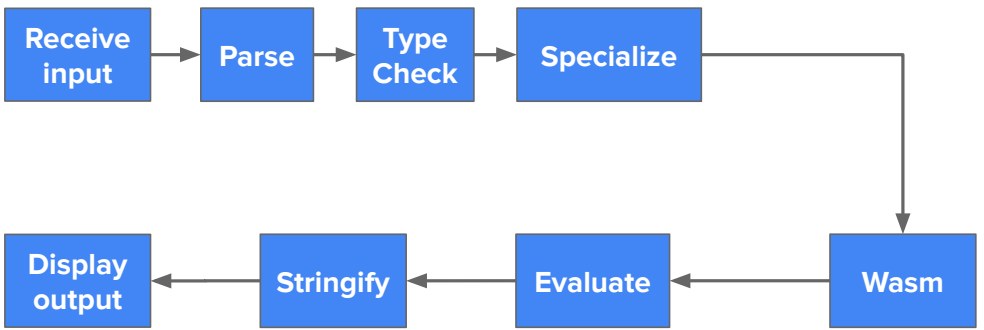
- Made up of many internal packages (Rust “crates”)
- Entrypoint (`main`) is in the CLI package
- CLI imports all other packages, glues them together

Web REPL

- Alternative entrypoint in new package
- Imports only the packages we need on the web
  - No CLI
  - No optimizations
  - No Intel/ARM

Web REPL size-optimized build

Remove code we don’t need
6MB uncompressed
2MB with Brotli compression 🎉



Challenge: Wasm limitations

Challenges

Download size far too big!

- 300k lines of code
- 66MB binary

WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

What are Web APIs?

JavaScript functions built into the browser

They're what make JavaScript useful!

120 APIs

WebAssembly cannot access any of them

Document Object Model API

- Insert text into the HTML
- Listen for user input

```
document.getElementById("output").innerText = "hello";
button.addEventListener("click", validateInput);
```

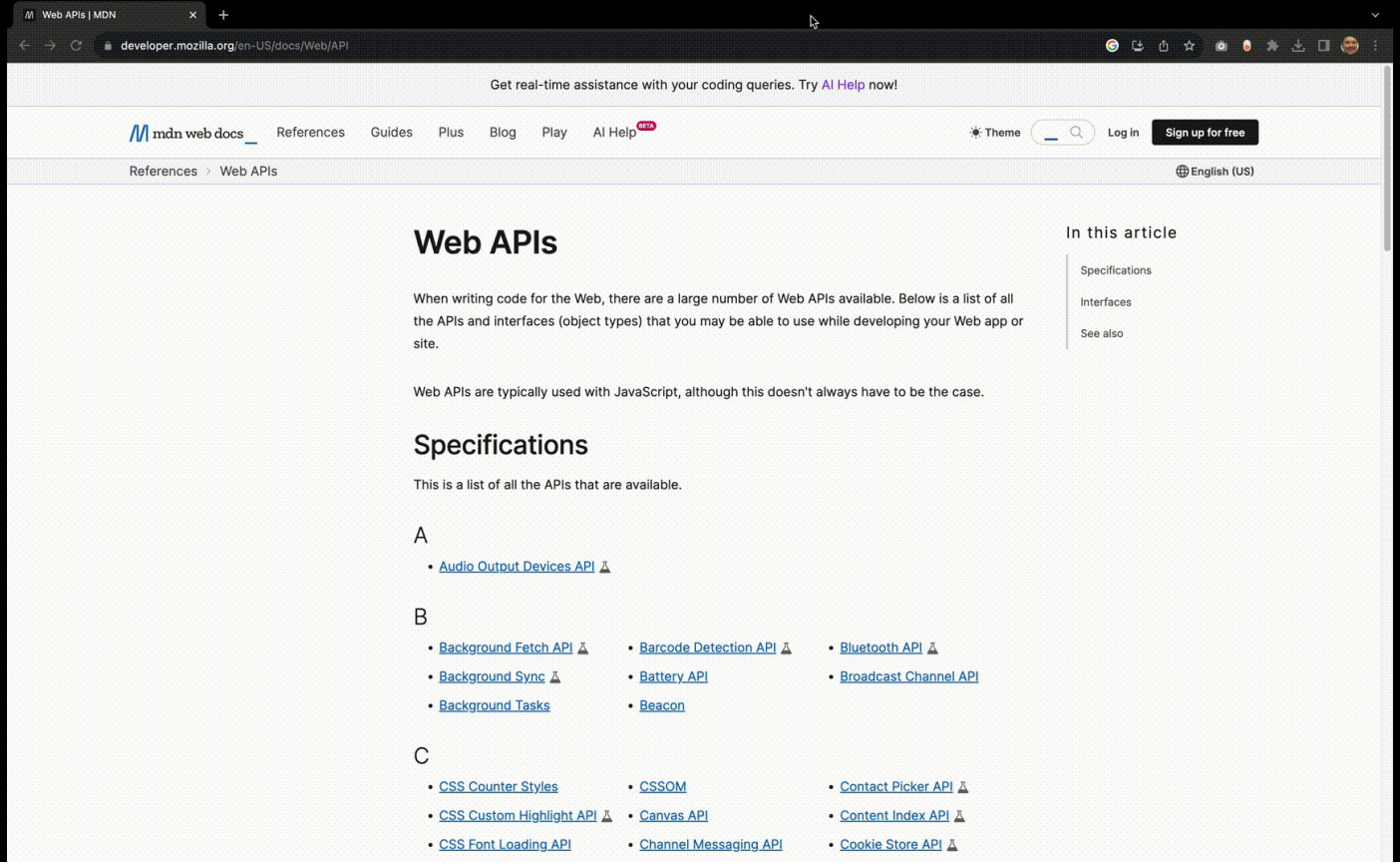
No equivalent in WebAssembly!

Fetch API

Ask the browser to make HTTP requests

```
let response = await fetch(url);
```

No equivalent in WebAssembly!


Web APIs | MDN
developer.mozilla.org/en-US/docs/Web/API
Get real-time assistance with your coding queries. Try AI Help now!
mdn web docs
References
Guides
Plus
Blog
Play
AI Help
BETA
Theme
Log in
Sign up for free
References
Web APIs
English (US)
Web APIs
When writing code for the Web, there are a large number of Web APIs available. Below is a list of all the APIs and interfaces (object types) that you may be able to use while developing your Web app or site.
Web APIs are typically used with JavaScript, although this doesn't always have to be the case.
Specifications
This is a list of all the APIs that are available.
A
Audio Output Devices API
B
Background Fetch API
Background Sync
Background Tasks
Barcode Detection API
Battery API
Beacon
Bluetooth API
Broadcast Channel API
C
CSS Counter Styles
CSS Custom Highlight API
CSS Font Loading API
CSSOM
Canvas API
Channel Messaging API
Contact Picker API
Content Index API
Cookie Store API
In this article
Specifications
Interfaces
See also

Implications

You can run any language in the browser...

...as long as it's just data transformations!

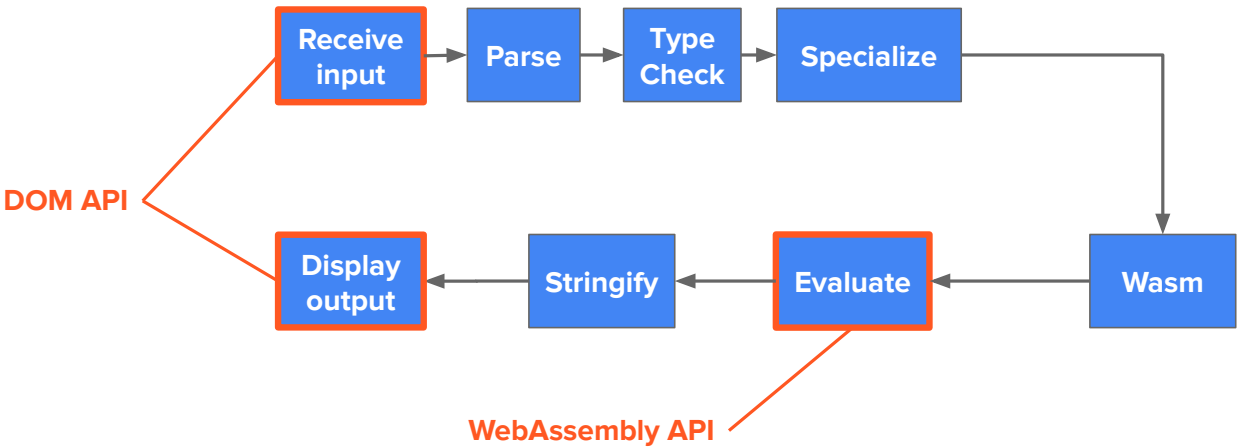
Only JavaScript can

Perform input/output

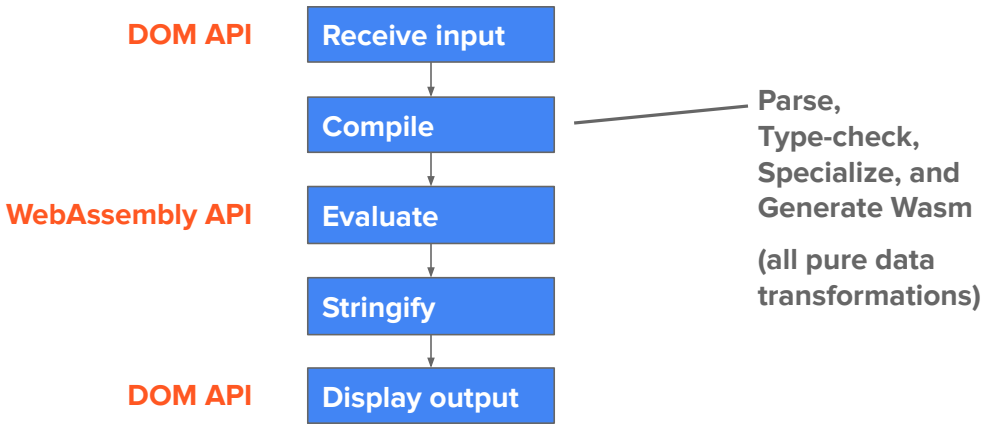
Access browser built-in functions

Using Web APIs

Where do we need Web APIs?



Rearranging our diagram



Receiving input

Challenges

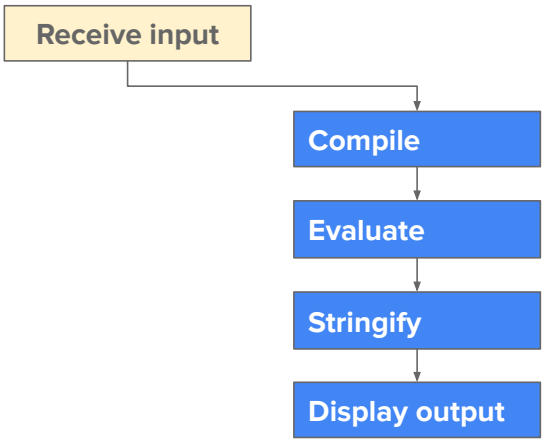
Download size far too big!

- 300k lines of code
- 66MB binary

WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

Listen for input in JS, then call WebAssembly



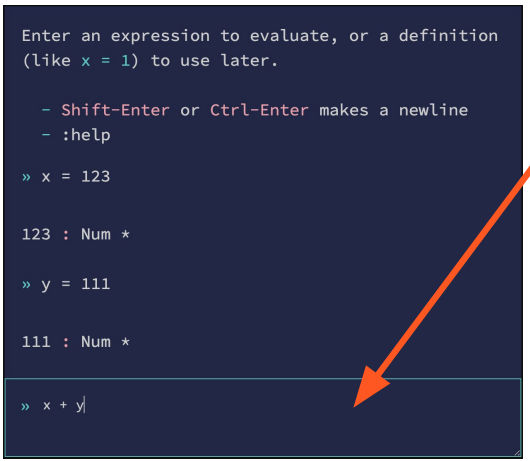
Light yellow background = JavaScript

Dark blue background = Rust/WebAssembly

Left side = caller

Right side = callee

JavaScript EventListener




```
const elemSourceInput = document.getElementById("source-input");
elemSourceInput.addEventListener("change", onInputChange);

function onInputChange(event) {
  const inputText = event.target.value;
  entrypoint_from_js(inputText);
  // TODO: finish this code later...
}
```

JS/Wasm interop

Challenges

Download size far too big!

- 300k lines of code
- 66MB binary

WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

JavaScript / WebAssembly interop

- JavaScript engines use Just-In-Time compilation
  - “Hot” functions are compiled to an optimized form
  - Thousands of calls to “warm up” & gather type info

- WebAssembly translates to the same optimized form
  - Both look the same inside the runtime!
  - And no “warming up” time. Types are known.

JavaScript / WebAssembly interop

- Function calls
  - Can go both ways
  - Arguments must be numbers
  - Return value must be a number (or else no return value)
- Shared memory
  - JavaScript can access WebAssembly's memory (array of numbers)
  - Used for passing data structures
  - One-way: WebAssembly can't "see inside" JavaScript objects

Passing a string using only numbers

```
function entrypoint_from_js(str) {
  const encoder = new TextEncoder();
  const bytes = encoder.encode(str);

  const start_index = wasm.allocate_memory_space(bytes.length);
  const end_index = start_index + bytes.length;

  wasm.memory.subarray(start_index, end_index).set(bytes);

  wasm.entrypoint_from_js(start_index, bytes.length);
}
```

Displaying output

Challenges

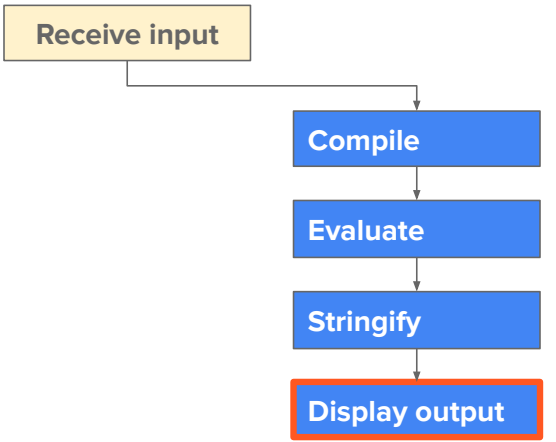
Download size far too big!

- 300k lines of code
- 66MB binary

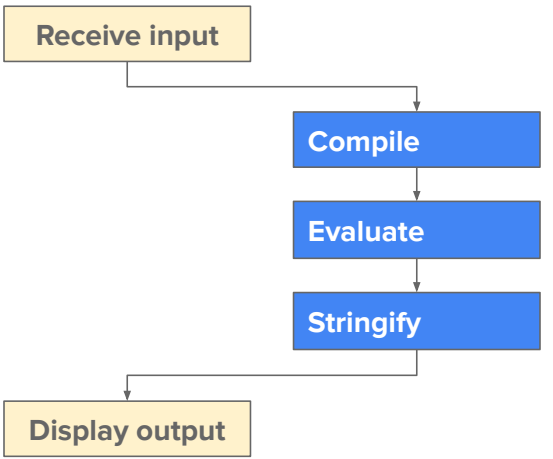
WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

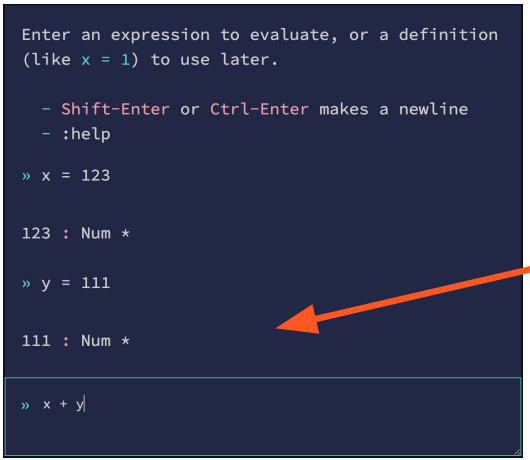
Wasm can't use DOM API to display output!



Return from WebAssembly to JavaScript



Displaying the output with JavaScript




```
function onInputChange(event) {
  const inputText = event.target.value;
  const outputText = wrapper.entrypoint_from_js(inputText);
  outputElement.innerHTML = outputText;
}
```

Create JS string from Wasm




```
let global_array = [];

function create_js_string(ptr, length) {
  const bytes = wasm.memory.subarray(ptr, ptr + length);

  const decoder = new TextDecoder();
  const str = decoder.decode(bytes);

  const js_array_index = global_array.length;
  heap.push(str);
  return js_array_index;
}
```

Rust calls the JS helper before returning



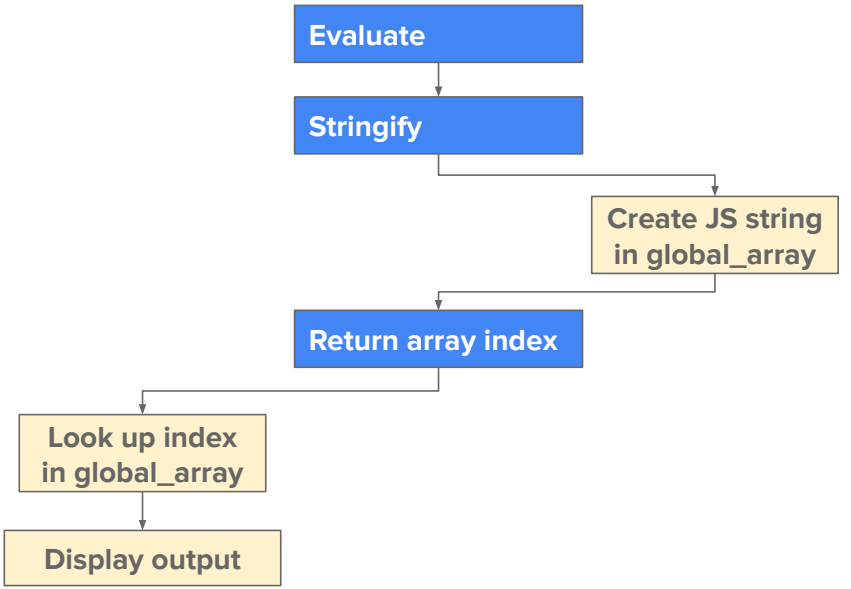
```
extern "C" {
  fn create_js_string(ptr: *const u8, len: usize) -> usize;
}

fn entrypoint_from_js(input_ptr: *mut u8, input_length: usize) -> usize {
  let user_input = String::from_raw_parts(input_ptr, input_length, input_length);

  let output: String = compile_and_evaluate_and_stringify(user_input);
  let output_ptr = output.as_ptr();
  let output_len = output.len();

  let js_array_index = create_js_string(output_ptr, output_len);
  return js_array_index;
}
```

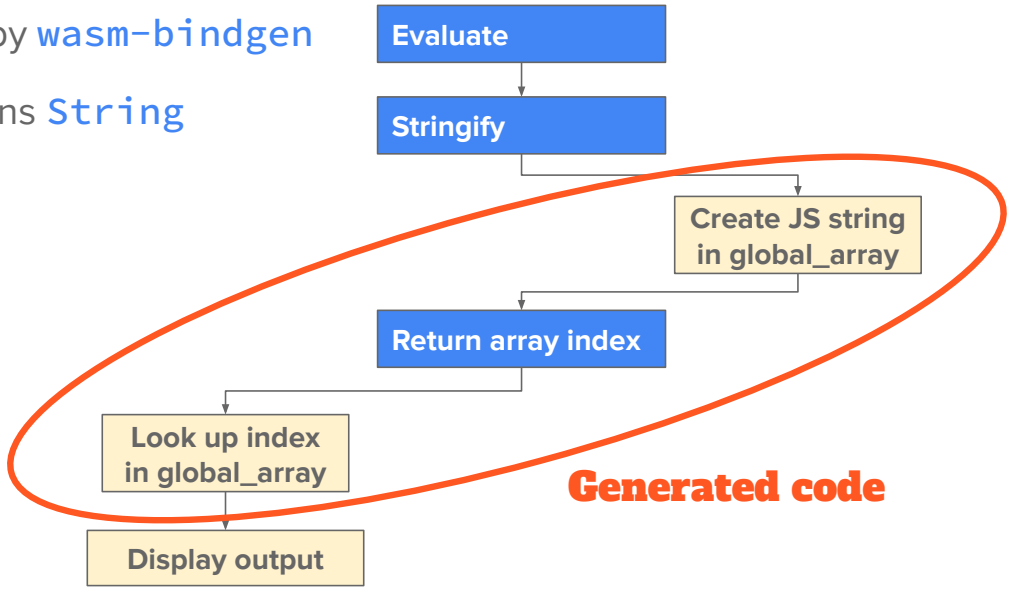
Returning the output string



Code generation

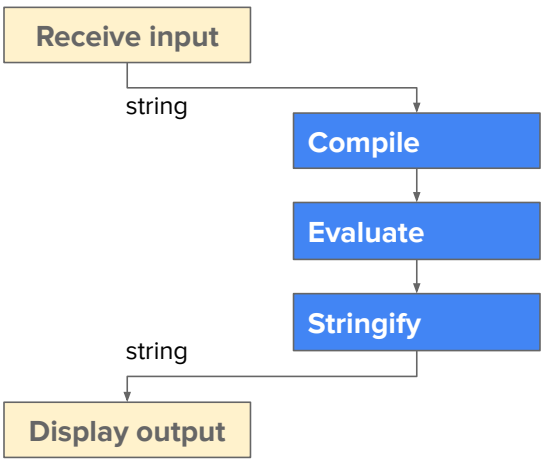
Glue code generated by `wasm-bindgen`

Handwritten Rust returns `String`



https://rustwasm.github.io/wasm-bindgen/

Simplified mental model with code gen



Evaluating the user's code

Challenges

Download size far too big!

- 300k lines of code
- 66MB binary

WebAssembly limitations

- Can’t listen for user input
- Can only understand JS numbers
- Can’t modify the HTML page
- Can’t run Wasm inside Wasm

Evaluating the user's compiled code

```
Enter an expression to evaluate, or a definition
(like x = 1) to use later.

  - Shift-Enter or Ctrl-Enter makes a newline
  - :help

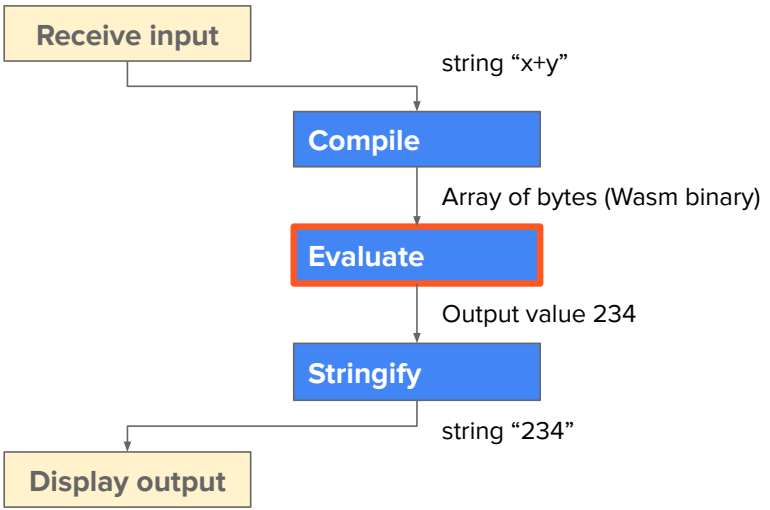
» x = 123

123 : Num *

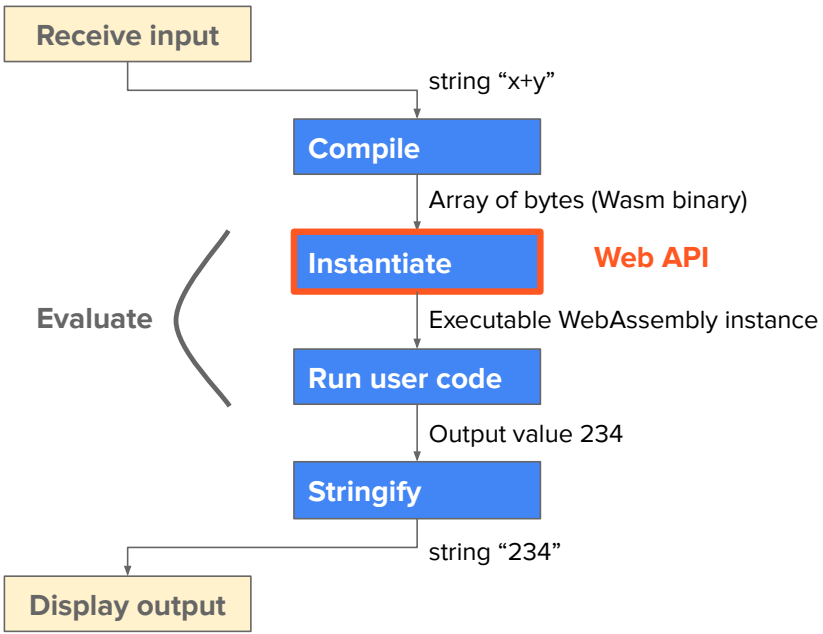
» y = 111

111 : Num *

» x + y
```



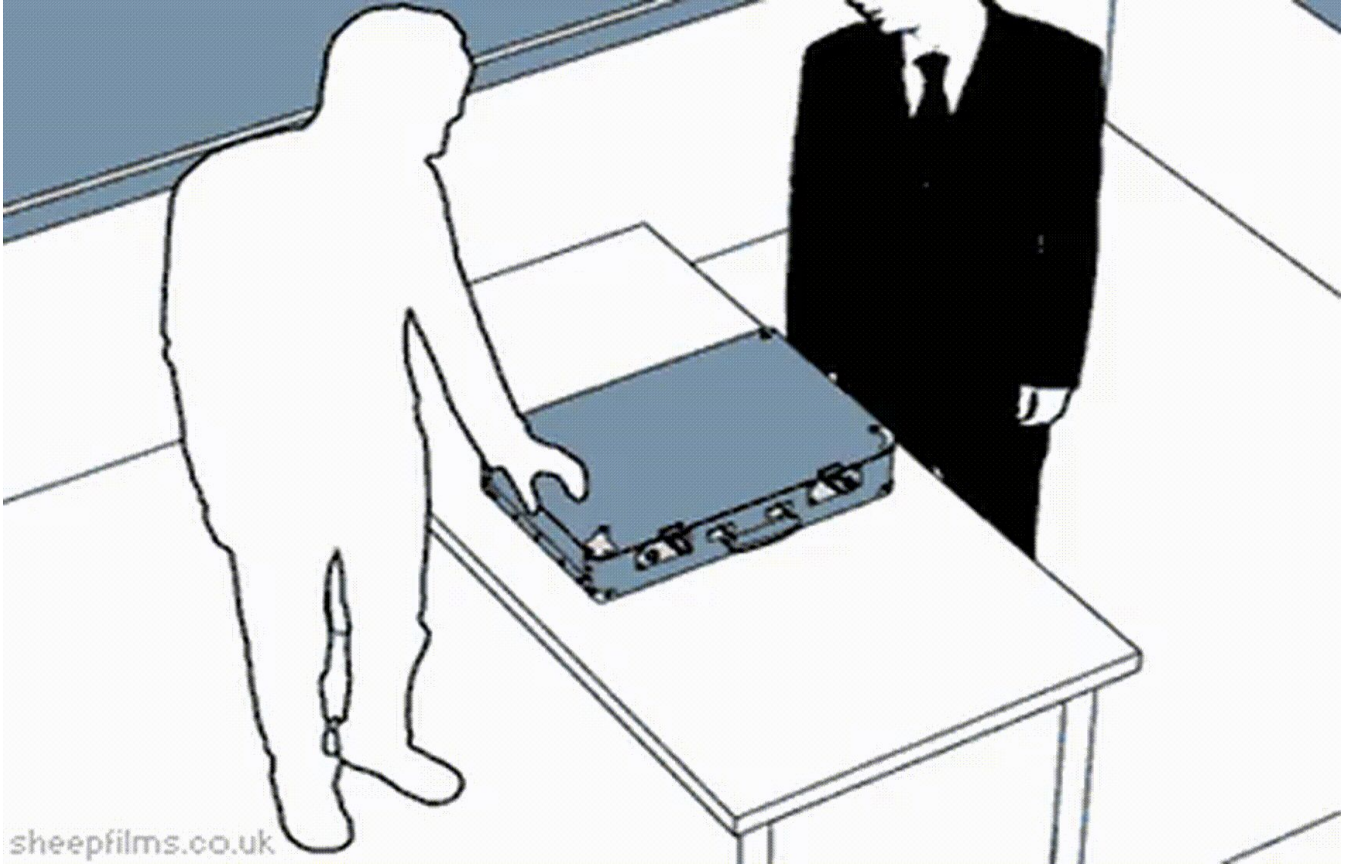
We need another Web API call

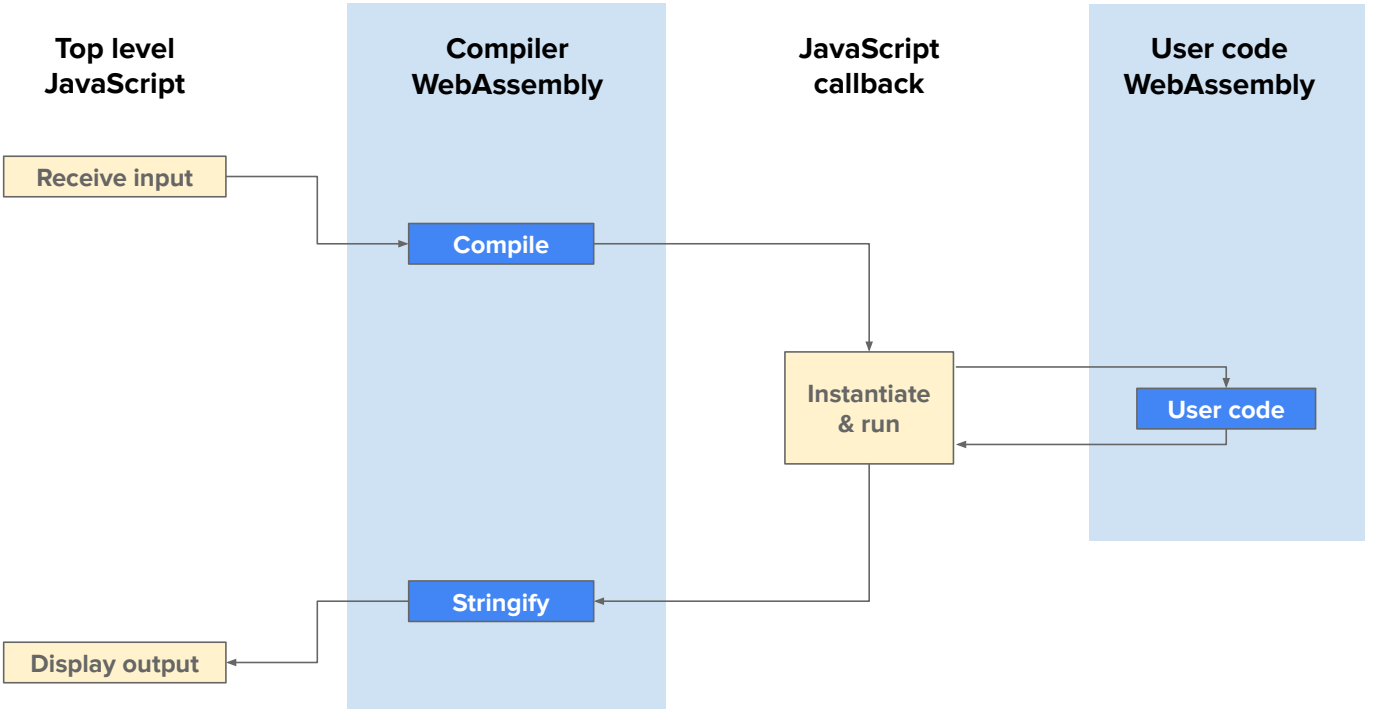


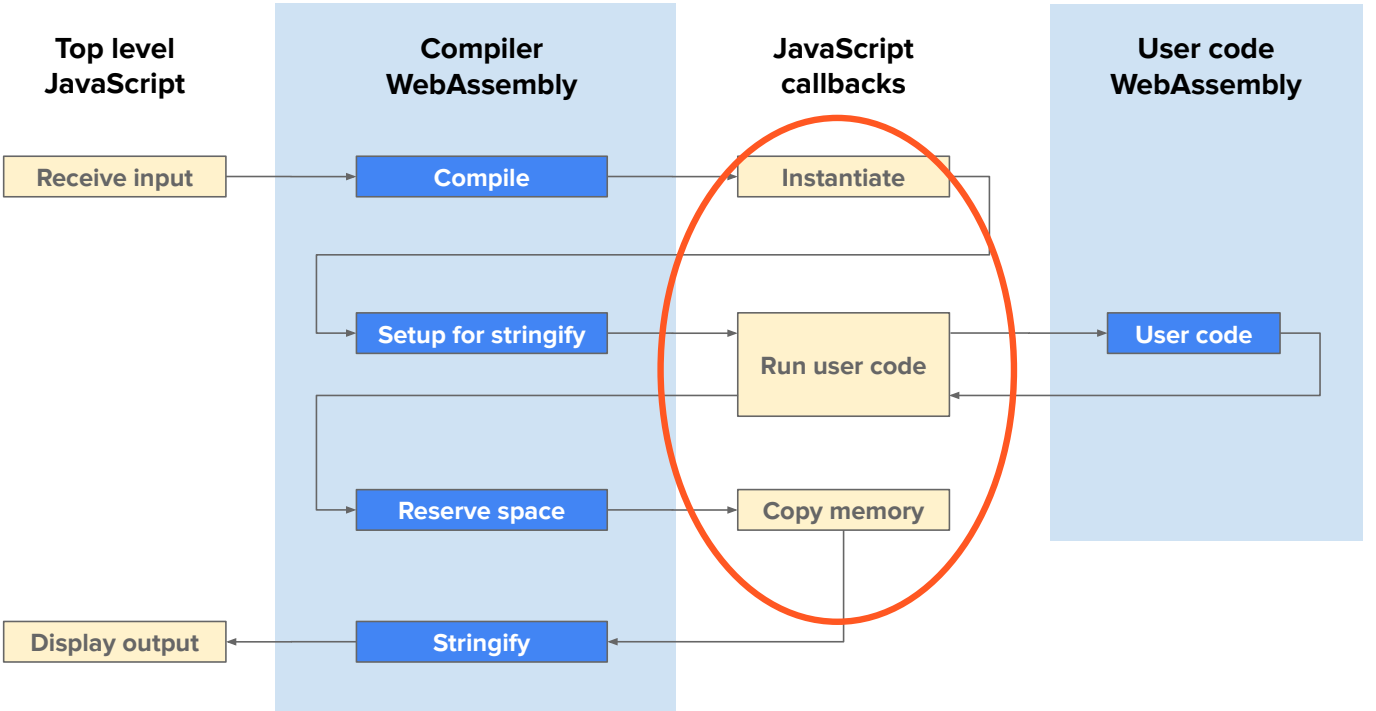
WebAssembly API

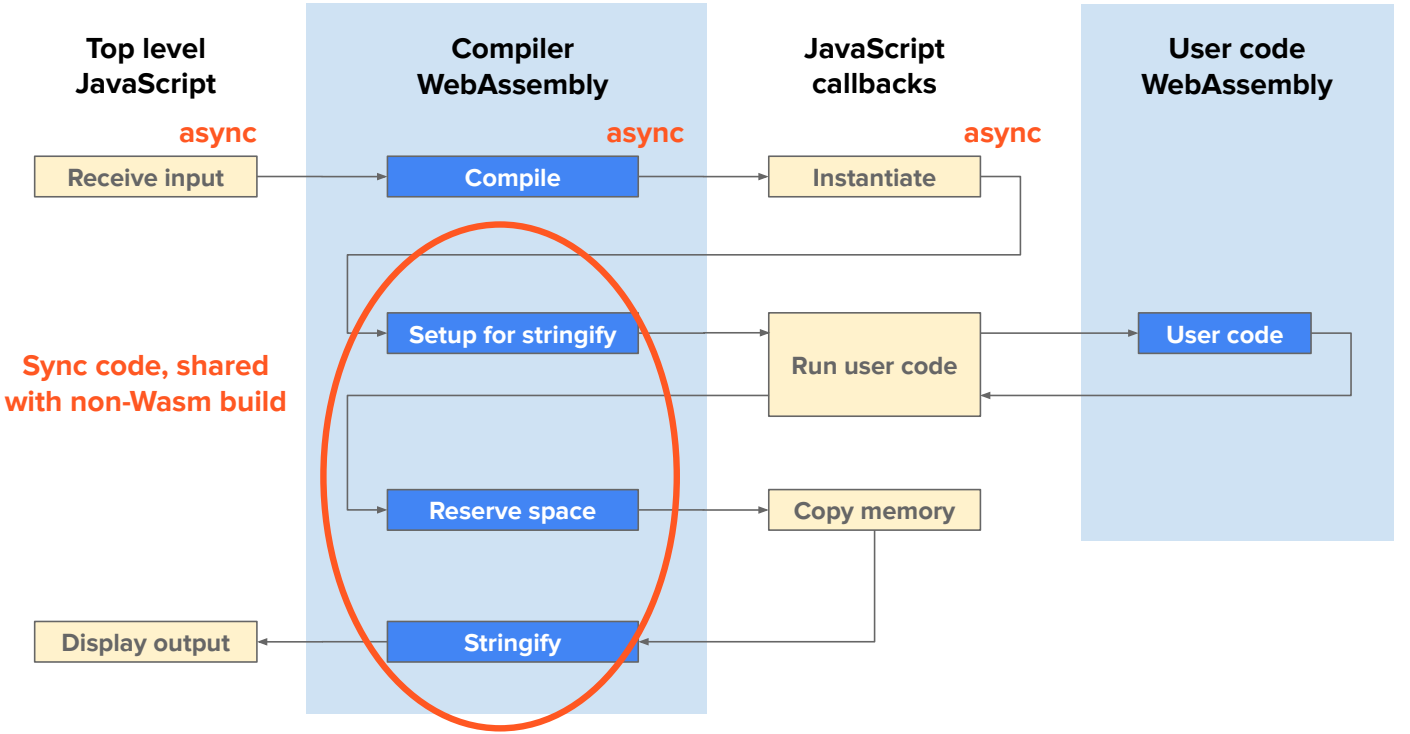
`WebAssembly.instantiate(wasm_bytes, callback_fns)`

- Loads bytes into browser's Virtual Machine
- Provide an object of JS functions that Wasm can call
- `WebAssembly.Instance` contains Wasm functions that JS can call


sheepfilms.co.uk

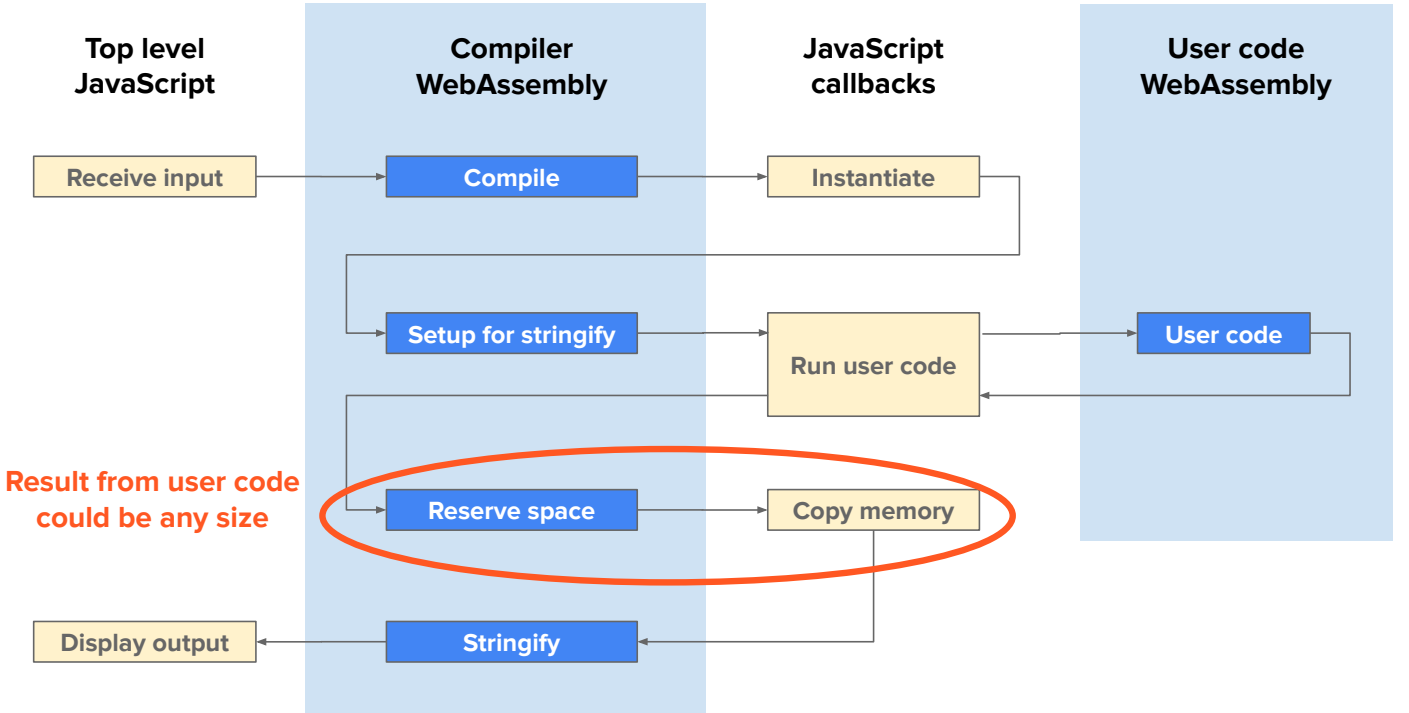

Top level JavaScript
Compiler WebAssembly
JavaScript callback
User code WebAssembly
Receive input
Compile
Instantiate & run
User code
Stringify
Display output


Top level JavaScript
Compiler WebAssembly
JavaScript callbacks
User code WebAssembly
Receive input
Compile
Instantiate
Setup for stringify
Run user code
User code
Reserve space
Copy memory
Display output
Stringify


Top level JavaScript
Compiler WebAssembly
JavaScript callbacks
User code WebAssembly
async
async
async
Receive input
Compile
Instantiate
Setup for stringify
Run user code
User code
Sync code, shared with non-Wasm build
Reserve space
Copy memory
Display output
Stringify

Async

- Generated code
  - Rust Futures ⟷ JS Promises
- Terminal REPL is sync
  - Rust async runtime would add some cost
  - Refactoring would be needed
  - Splitting the callback is easy


Top level JavaScript
Compiler WebAssembly
JavaScript callbacks
User code WebAssembly
Receive input
Compile
Instantiate
Setup for stringify
Run user code
User code
Result from user code could be any size
Reserve space
Copy memory
Display output
Stringify

Summary

Summary

- Architecture advantage: no servers to manage!
- Reduced download size
  - Dedicated entrypoint, removed subsystems
- Always need JavaScript
  - HTML input & display
  - Creating WebAssembly
  - … and 118 other APIs your app might need
- JavaScript/Wasm interop
  - Engine treats Wasm like optimized JS
  - Low level API: numbers & shared memory
  - Code generation for strings, async, etc.
  - Reorganise code to work around limitations

Links

Website

https://www.roc-lang.org

https://www.roc-lang.org/repl

Source Code

https://github.com/roc-lang/roc

https://github.com/roc-lang/roc/blob/e91b50ac/crates/repl_wasm/README.md

https://github.com/roc-lang/roc/tree/e91b50ac/www/public/repl/

goto;

Don't forget to
rate this session
in the **GOTO Guide app**