

# ~~超入門~~ WebAssembly

～入門では収まらなかったバージョン～

2025.01.17 Fri. 東京Ruby会議12 前夜祭

近藤 うちお (@udzura)

SmartHR プロダクトエンジニア

## 初めにお断り ...

- 元々「超入門 WebAssembly」というタイトルで応募していましたが、入門レベルでは収まりませんでした。
- なので発表タイトルがただの「 WebAssembly」になります(?)

# 自己紹介

近藤 宇智郎 (@udzura)

株式会社 SmartHR (2024/11~)

プロダクトエンジニア 労務基本機能担当

Fukuoka.rb  やってます！

10年ほど池袋住んでたということで ...

マイブーム：WebAssembly、中学受験



# テーマ: WebAssembly



# WebAssembly とは

- ブラウザで動かせるバイナリ形式のこと
  - いろいろな言語 ( C, C++, Rust, ... ) をコンパイルして、バイナリを作り、それらをブラウザの上で実行できる
- WASMとも呼ぶ
  - 短くしたいときこの表記になるぐらいの使い分け(か?)
  - 仕様・概念を WebAssembly、実プログラムを WASM と呼ぶ傾向はありそう

# WebAssembly とは

色々なところで

- ~~ブラウザで動かせる~~バイナリ形式のこと
  - ! ! !
  - つまり...??



## e.g. WASMはターミナルでも動く

- ターミナルで WebAssemblyのプログラムを実行可能
  - 例えば [wasmtime](#) というコマンドで実行できる :

```
> wasmtime --invoke fib ./fib.wasm 20 2>/dev/null
6765
```

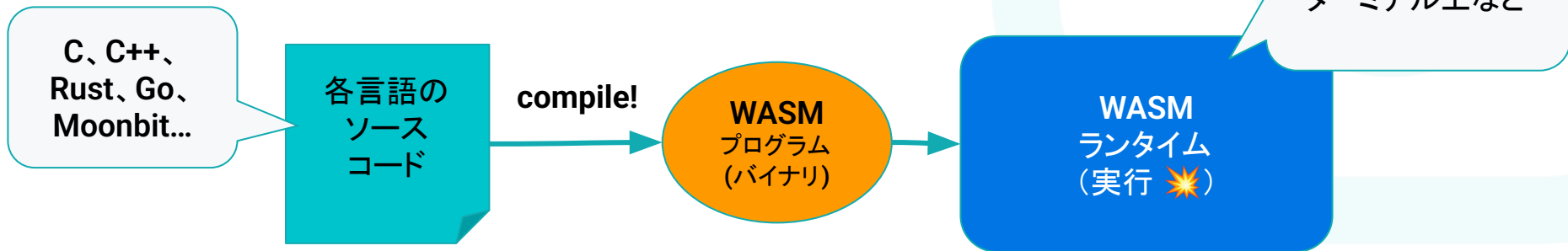
# WebAssembly のメリット

- 言語を選ばずにバイナリを作れる
- そのバイナリは、ブラウザを中心に いろいろなところで動かせる
  - Write Once, Run Anywhere なのじゃよ ...
- 多くのランタイムで高速に動くことが期待できる



# WebAssemblyの動かし方

- WASMランタイムには色々ある( Wasmtimeなどが有名)
- WASMプログラムも、何言語で書いても OK
  - 一般には C、C++、Rustをコンパイルする

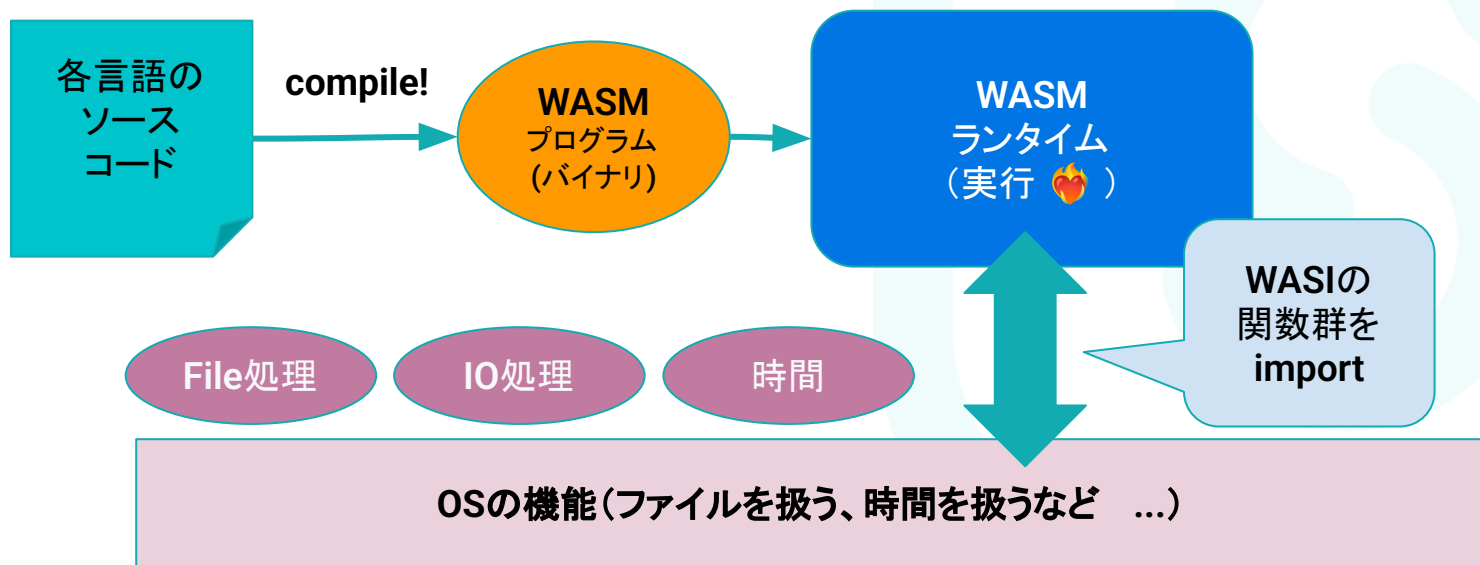


# WebAssemblyの大事な概念

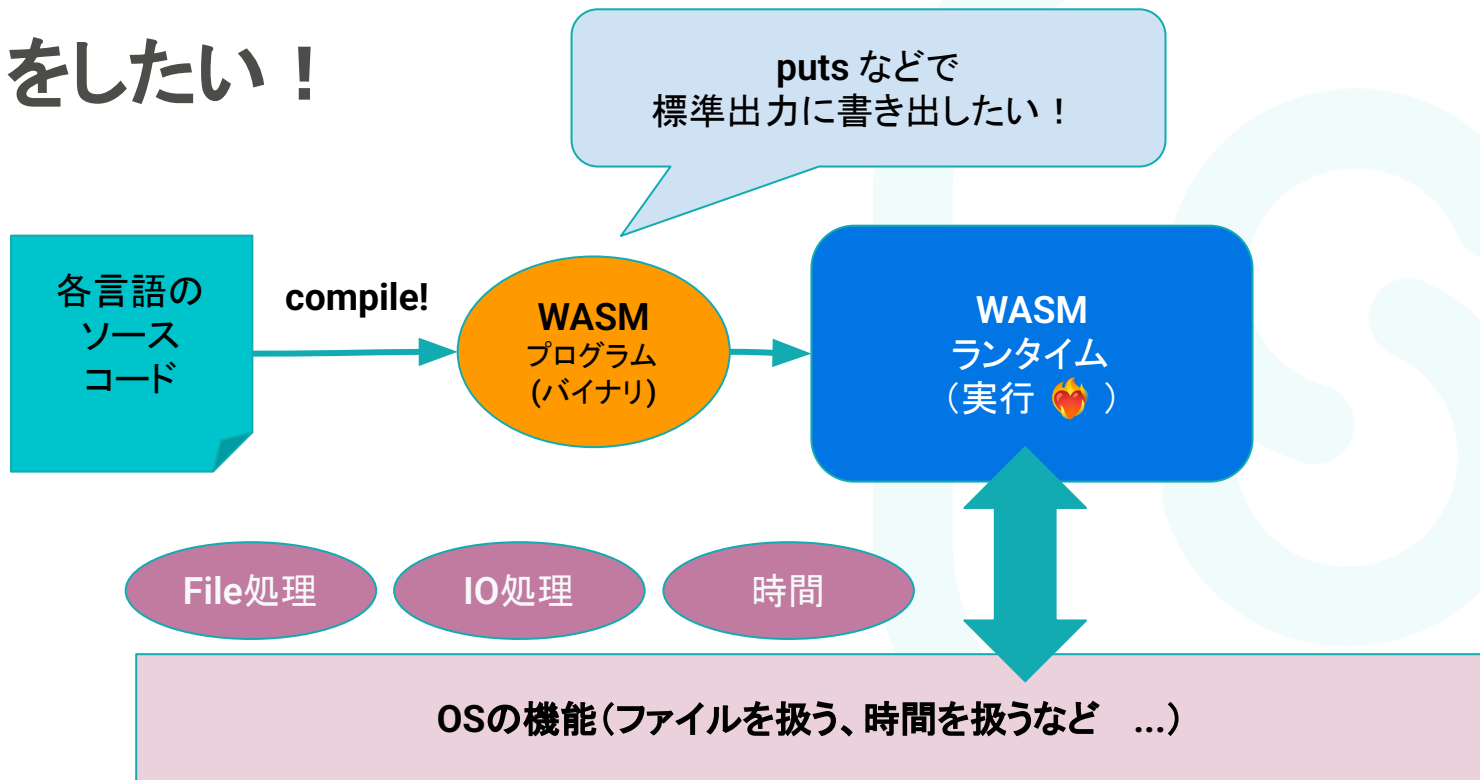
# WebAssembly System Interface について

- WASMとOSの機能(ファイルを扱う、時間を扱う ...)を繋ぐ仕様が WASI (WebAssembly System Interface)
- WASM自体はOSの機能を仕様に含んでいない
  - OSの機能を扱えるようにした拡張仕様のようなもの

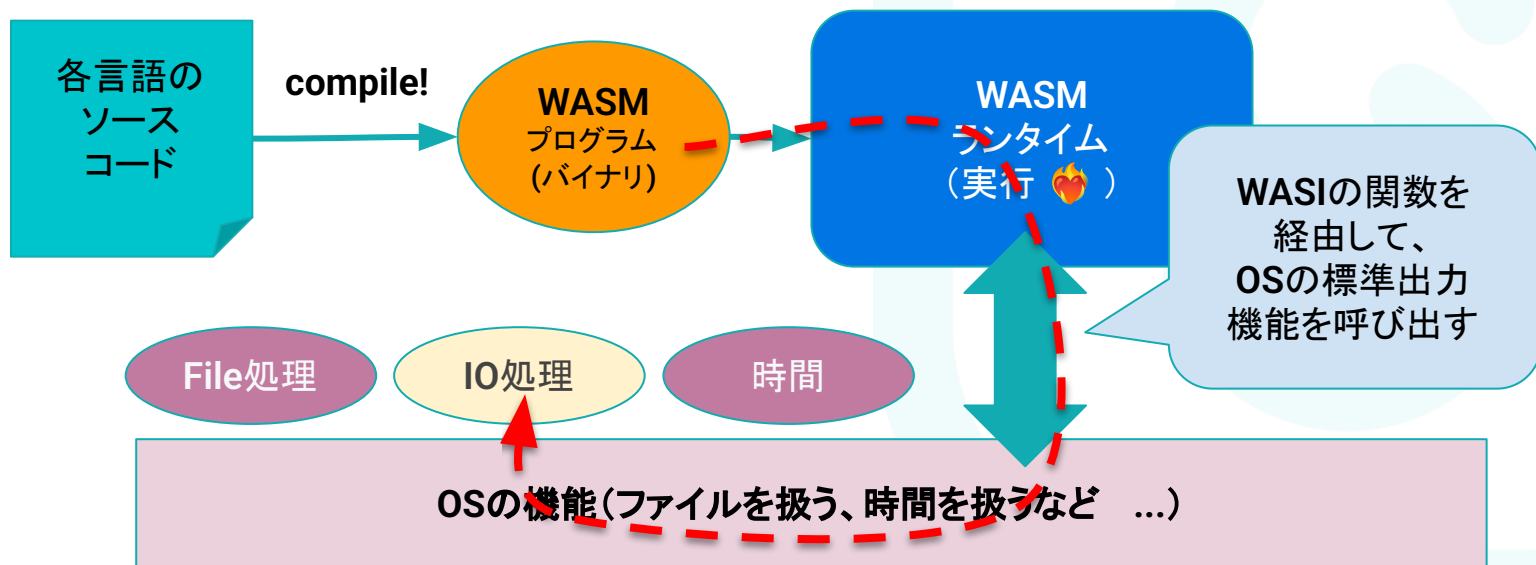
# WASI のざっくりした概念図



# 出力をしたい！



# WASI を通して OS の機能を使う



# WASI の補足...

- preview1とか2とか聞いたことがあるかも知んですが ...
  - preview 2はComponent ModelというWASMの次世代仕様に沿ったシステムインタフェース
  - 今広く使われているのは preview 1、という点を覚えておけば一旦 OKです！

**WASI の話は後半で登場します！**





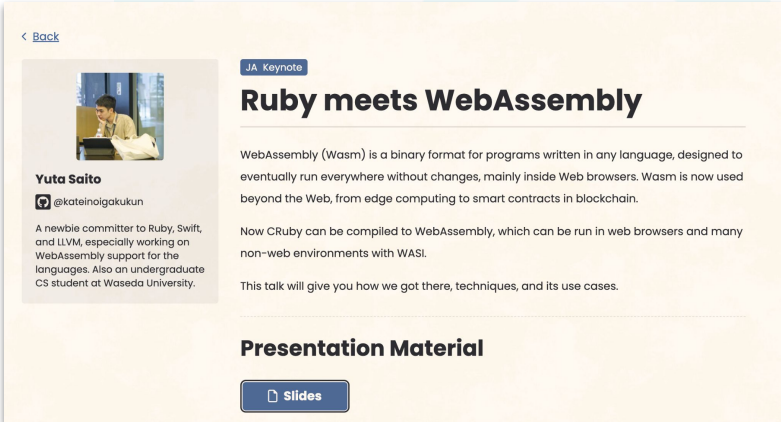
# WebAssembly と Ruby



# Ruby で WebAssembly といえば

- ruby.wasm ですよね
- 公式な

“RubyをWASMで動かす方法”



< Back

JA. Keynote

## Ruby meets WebAssembly

WebAssembly (Wasm) is a binary format for programs written in any language, designed to eventually run everywhere without changes, mainly inside Web browsers. Wasm is now used beyond the Web, from edge computing to smart contracts in blockchain.

Now CRuby can be compiled to WebAssembly, which can be run in web browsers and many non-web environments with WASI.

This talk will give you how we got there, techniques, and its use cases.

### Presentation Material

Slides

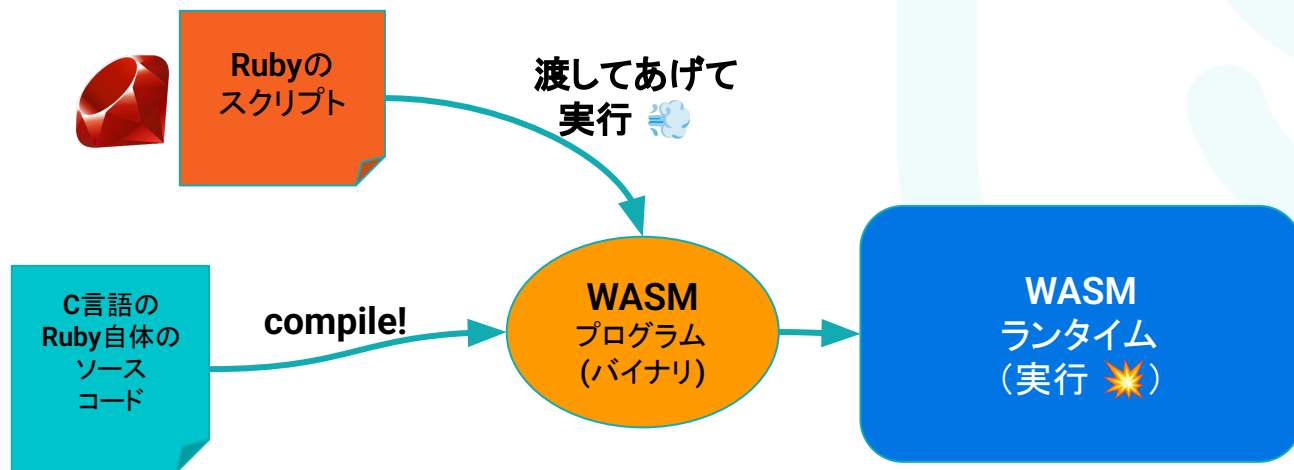
**Yuta Saito**  
@kateinoigakukun

A newbie committer to Ruby, Swift, and LLVM, especially working on WebAssembly support for the languages. Also an undergraduate CS student at Waseda University.

<https://ruby.github.io/ruby.wasm/>

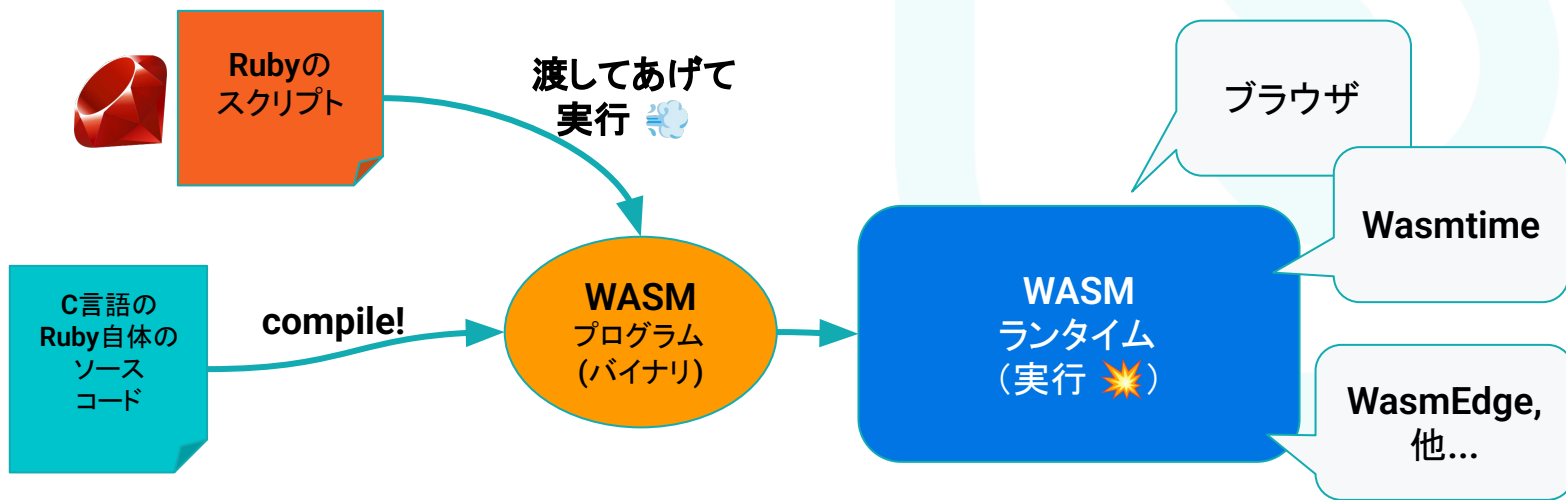
# ruby.wasm の世界観

- 基本的には、RubyのCコードを丸ごと WASMバイナリにし、スクリプトを動かしている



# ruby.wasm は実はどこでも動く

- ランタイムとして、ブラウザでも、Wasmtimeのような cli tool でも、様々な選択肢がある



# WebAssembly と Ruby と私

# WebAssembly x Ruby 関係で...

- OSSをいくつか作ってます！

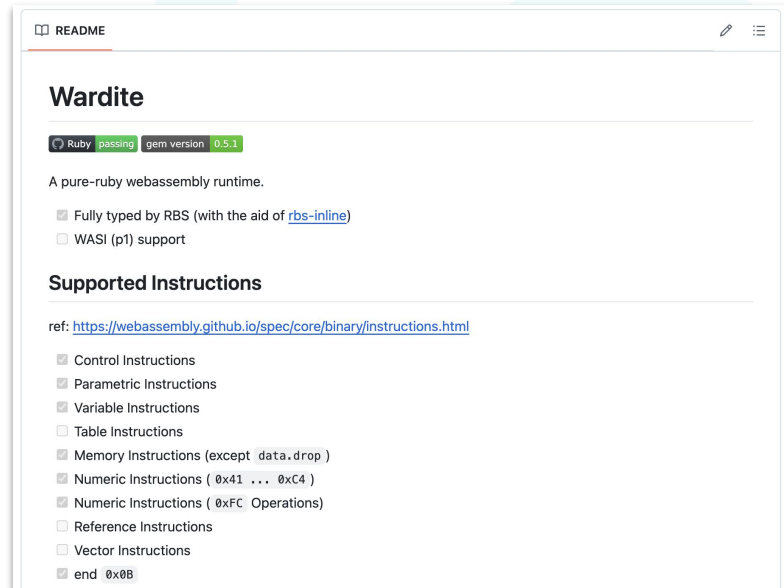


私が  
作りました

- 最近の生活そのものなので軽く紹介します

# Wardite

- @udzura の有休消化中に開発
- Rubyで書いた  
WebAssemblyランタイム



<https://github.com/udzura/wardite>

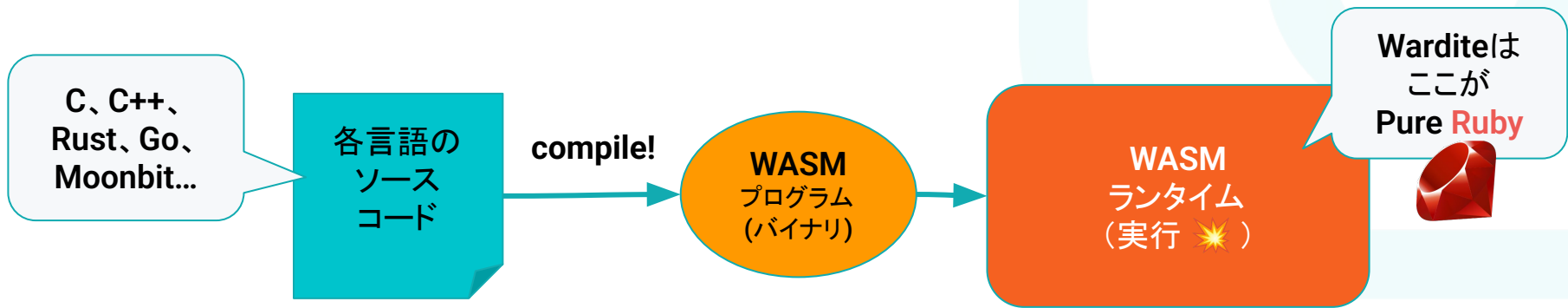
# Wardite is ... ?

- Rubyで書いたWebAssemblyランタイムとは、つまり
  - WebAssemblyのバイナリプログラムを動かすための環境やライブラリのこと
  - これがあればWASMを動かせます！一式セット
    - という理解でOK



# Wardite の立ち位置

- Wardite is ランタイムの方をRubyで書いたという話
- 色々な言語で書いた WASMプログラムを実行できる



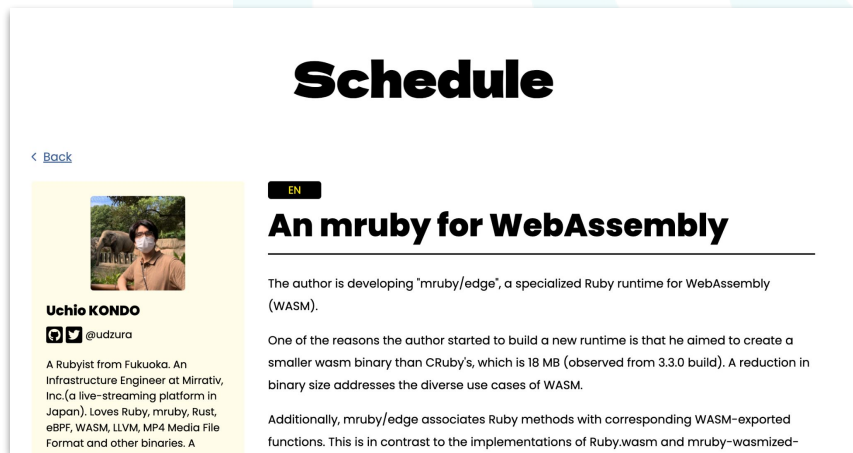
# おまけ: もう一つOSSを紹介



# mruby/edge

※ 正確には... mec (mruby/edge compiler)というツールも別にあってそれでバイナリを作っています

- Rubyのスク립トを WASMバイナリに固める OSS Project
- RubyKaigi 2024 で話した
- その時から肅々と機能追加している



## Schedule

[< Back](#)

**EN**

### An mruby for WebAssembly

The author is developing "mruby/edge", a specialized Ruby runtime for WebAssembly (WASM).

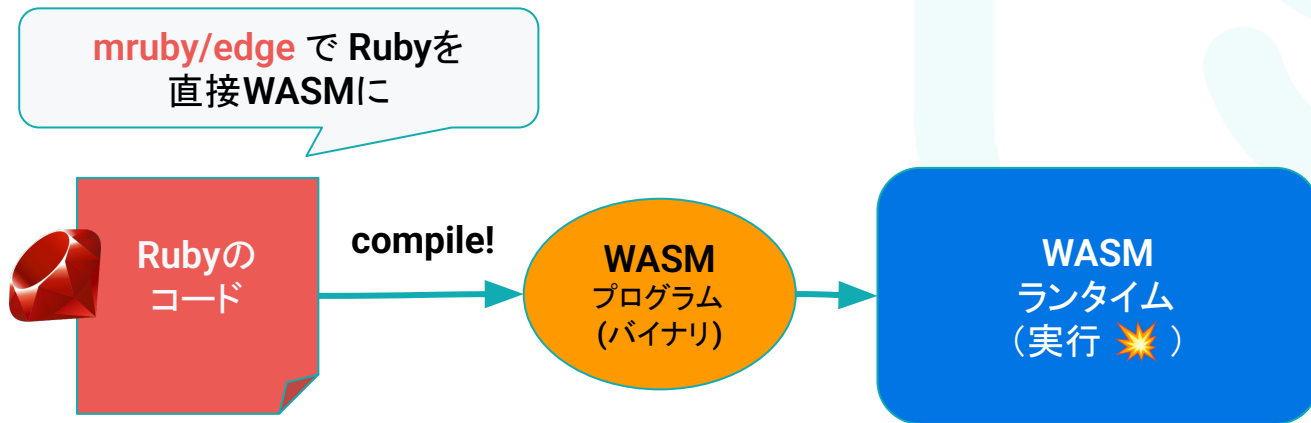
One of the reasons the author started to build a new runtime is that he aimed to create a smaller wasm binary than CRuby's, which is 18 MB (observed from 3.3.0 build). A reduction in binary size addresses the diverse use cases of WASM.

Additionally, mruby/edge associates Ruby methods with corresponding WASM-exported functions. This is in contrast to the implementations of Ruby.wasm and mruby-wasmized-

<https://github.com/udzura/mrubyedge>

# mruby/edge is ... WASMバイナリをRubyで！

- Rubyスクリプト → WASMバイナリにコンパイル できるように作ったのが mruby/edge



# Wardite について詳しく

# 名前の由来

- **WA** で始まる鉱石の名前を採用
- ワード石:  
 $\text{NaAl}_3(\text{PO}_4)_2(\text{OH})_4 \cdot 2(\text{H}_2\text{O})$



<https://en.wikipedia.org/wiki/Wardite#/media/File:Wardite.jpg>

# Wardite の特徴

- Pure Ruby 製
  - 標準添付ライブラリ以外に動作上の依存なし
- Fully RBS annotated (Thanks to rbs-inline)

# Wardite が達成していること

- WASM Core の基本的な命令を大体実装
- WASM Core = WASMの基本的な標準仕様
- WASI preview 1 も対応(予定)

## Supported Instructions

ref: <https://webassembly.github.io/spec/core/binary/instructions.html>

- Control Instructions
- Parametric Instructions
- Variable Instructions
- Table Instructions
- Memory Instructions (except `data.drop`)
- Numeric Instructions ( `0x41` ... `0xC4` )
- Numeric Instructions ( `0xFC` Operations)
- Reference Instructions
- Vector Instructions
- end `0x0B`



# 参考: 先行実装

- technohippy/wasmrb
  - <https://github.com/technohippy/wasmrb>
  - Referecialな実装。綺麗なコードだけど、4年前最終更新
    - 1. WASIに未対応
    - 2. WASMの仕様自体も、未対応の箇所がそこそこある
  - ということで Warditeを推したいナ ...

# なぜRubyでWebAssemblyランタイムを？

- C or Rustで書かれたランタイムを使うこともできる
  - Wasmtime, WasmEdge, Wasmer, ...などがある
  - 拡張gemがある [bytecodealliance/wasmtime-rb](https://github.com/bytecodealliance/wasmtime-rb) など
- 速度面もCなどが有利ではありそう

# ですが...

- bundle installしてこのメッセージが出た時の気持ち、覚えてますか

```
Building native extensions. This could take a while...
```

- C拡張はちょっとインストールが手間な時がある

# Pure Rubyであることのメリット

- インストール・セットアップが簡単
  - Rubyが動けば動く！
  - プラットフォームに依存しにくい  
(CPUの種類、OSの種類やバージョン ...)
    - 可搬性のために依存ライブラリをなくしている

## 参考: Goの事例

- GoのC拡張組み込みの仕組み( cgo)も結構面倒 ...
- Pure Go の WebAssembly ランタイム [wazero](#) がある
  - Goはコンパイル言語だけど、WebAssembly + wazeroで動的なプラグイン機構を実現したりしている
  - <https://github.com/knqyf263/go-plugin>

と、いろいろ言いましたが ...

面白そうなので作りました

## Just For Fun...

- 開発のきっかけは「勉強目的」「なんか面白そう」
- 作ってみたら、思ったよりちゃんと動くので
  - 色々使ってみたい！
  - 実装を引き続き頑張るぞ



# Wardite の動かし方



# WASMのプログラムを作ってみよう

- 今日の例: Cでフィボナッチ数を求める関数です :
- WASM のプログラム自体を Cで書くのは一般的

```
1  int fib(int n) {  
2      if (n <= 1) return n;  
3      return fib(n - 1) + fib(n - 2);  
4  }
```

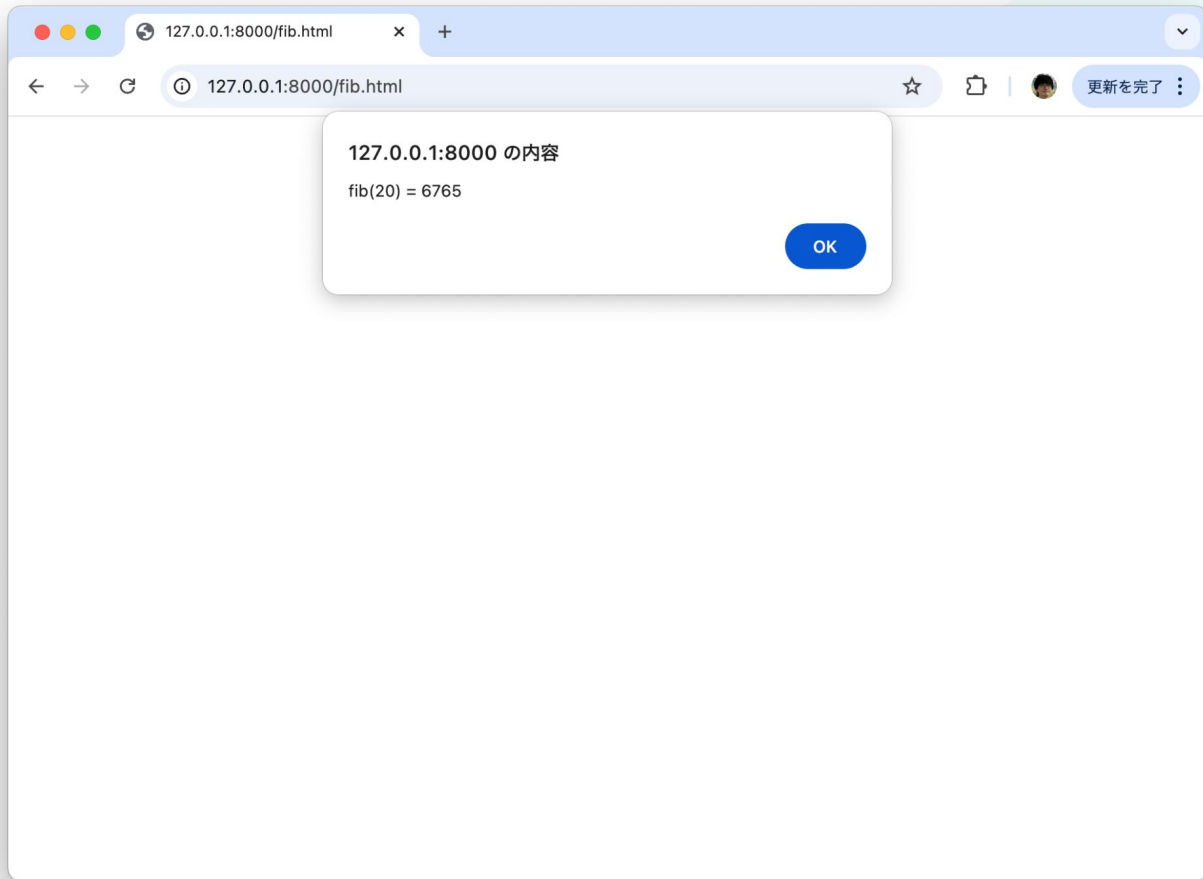
## これをWebAssemblyにコンパイル

- 例えば、clangというCコンパイラで作れる
- `fib.wasm` というWASMバイナリを作成

```
> clang --target=wasm32 \  
    --no-standard-libraries \  
    -Wl,--export-all -Wl,--no-entry \  
    -o fib.wasm \  
    fib.c
```

## ブラウザで使うには：

```
1 // WebAssemblyをfetchしてインスタンス化する
2 WebAssembly.instantiateStreaming(
3   fetch('./fib.wasm')
4 ).then(obj => {
5   // obj.instance にインスタンスがあり、
6   // さっきの fib がexportされている
7   const value = obj.instance.exports.fib(20)
8   alert(`fib(20) = ${value}`);
9 });
```



# Warditeコマンドを使うには？

- gemにしているなので、すぐインストールできる！

```
> gem install wardite
Successfully installed wardite-0.6.1
Parsing documentation for wardite-0.6.1
Done installing documentation for wardite after 0 seconds
1 gem installed
```

# Warditeコマンドの結果

- wardite gem と一緒にコマンドがインストールされます

さっき作った  
fib.wasm

```
> file ./fib.wasm
./fib.wasm: WebAssembly (wasm) binary module version 0x1 (MVP)
> wardite ./fib.wasm fib 20
warning: unimplemented section: 0x00
return value: I32(6765)
```

同じ  
fib(20) = 6765

# Wardite を支える技術





## ここから

- 内部実装の話です！！！！
- やや難しい用語を説明なしに使ってるかもです、すいません....。

# 最初の実装

- ゴリラさんの「[RustでWasm Runtimeを実装する](#)」を参考に実装を開始した
- RBSの練習のため rbs-inlineを全面的に有効にして書いた

# どんな順番？

- 最初はとにかくデータ構造を把握
  - ref: [Wasmバイナリの全体像](#)
- バイナリパーサを書く
- 命令とVMを書く
- 命令を頑張って実装する( 190個ぐらい)

```
0x0 | 00 61 73 6d | version 1 (Module)
    | 01 00 00 00
0x8 | 01 06       | type section
0xa | 01         | 1 count
--- rec group 0 (implicit) ---
0xb | 60 01 7f 01 | [type 0] SubType { is_final:
    | 7f         | ed: false }
0x10 | 03 02      | func section
0x12 | 01         | 1 count
0x13 | 00         | [func 0] type 0
0x14 | 07 07      | export section
0x16 | 01         | 1 count
0x17 | 03 66 69 62 | export Export { name: "fib",
    | 00 00
0x1d | 0a 1f      | code section
0x1f | 01         | 1 count
===== func 0 =====
0x20 | 1d         | size of function
0x21 | 00         | 0 local blocks
0x22 | 20 00      | local_get local_index:0
0x24 | 41 03      | i32_const value:3
0x26 | 48         | i32_lt_s
0x27 | 04 40      | if blockty:Empty
0x29 | 41 01      | i32_const value:1
0x2b | 0f         | return
0x2c | 0b         | end
```

※ 画像はイメージです

# わからない単語がいっぱい ...

- 読み進めていて例えば こういう表現 が

```
let (input, code) = le_u8(input);  
let (input, size) = leb128_u32(input);
```

section code は1バイト固定なので le\_u8() を使っている。

section size はLEB128<sup>[1]</sup>でエンコードされた u32 なので、値を読み取る際は leb128\_u32() を使う必要がある。

- > ~section sizeはLEB128[1]でエンコードされた u32～
- LEB128とは...??

※ WASMバイナリはいくつかのセクションに分かれている。  
そのパースをする処理の説明での文言  
※ 本文にも解説はしっかり存在します

# LEB128 (Little Endian Base 128) とは何か

- 数値のエンコーディングの方式の一つ。可変長方式
  - WASMバイナリの随所で登場する大事な実装

```
# - 0 ~ 6bit目: 数値表現  
# - 7bit目: 次のバイトも使う  
# - i番目の表現は << (7*i) して足す  
[0b10101010, 0b01101101]  
# この場合、  
(0b10101010 & 0b01111111) | (0b01101101) << 7  
=> 13994
```

# Rubyで実装した例

```
def to_i_by_uleb128(bytes)
  dest = 0
  bytes.each_with_index do |b, level|
    upper, lower = (b >> 7), (b & (1 << 7) - 1)
    dest |= lower << (7 * level)
    if upper == 0
      return dest
    end
  end
  raise "unreachable"
end

to_i_by_uleb128 "\xAA\x6D".unpack("C*")
# => 13994
```

# バイナリパーサ自体の実装

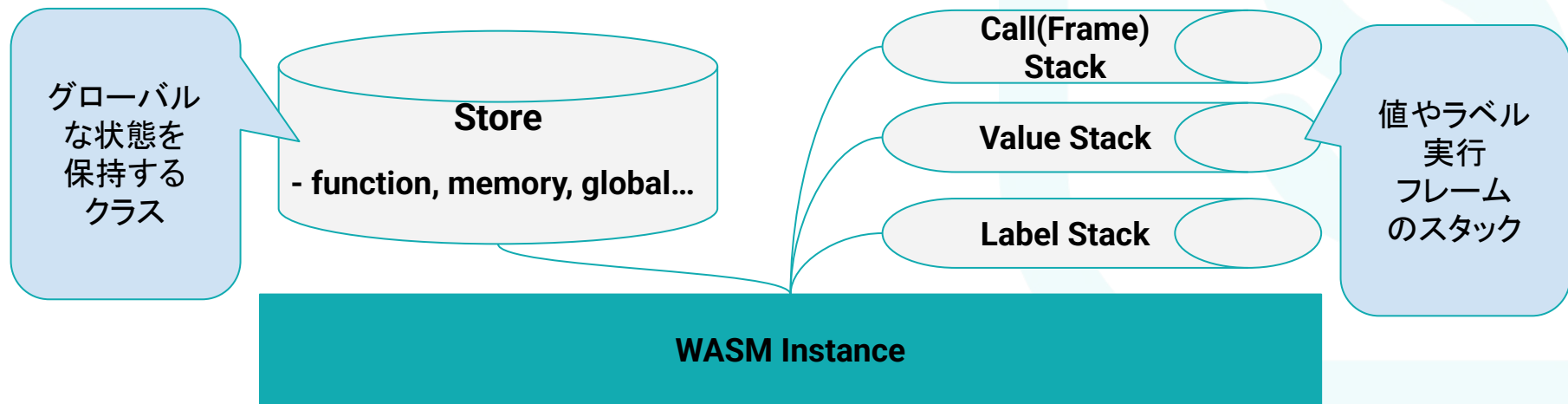
- 素朴に1 byte 1 byte取る感じにした。こういうコード

```
# @rbs return: Integer
def self.preamble
  asm = @buf.read 4
  raise LoadError, "buffer too short" if !asm
  raise LoadError, "invalid preamble" asm != "\u0000asm"
  vstr = @buf.read(4)
  version = vstr.to_enum(:chars)
  .with_index
  .inject(0) {|dest, (c, i)| dest | (c.ord << i*8) }
  raise LoadError, "unsupported ver: #{version}" if version != 1

  version
end # ...
```

# VM周りの実装

- [Runtime Structure](#) を参照。以下のようなクラスを作った





# VMの命令実行部分

- 基本的に一つ取得して  
実行、ループするだけ
- 命令は case 文で  
分岐する

```
# @rbs return: void
def execute!
  loop do
    cur_frame = self.call_stack.last #: Frame
    if !cur_frame
      break
    end
    cur_frame.pc += 1
    insn = cur_frame.body[cur_frame.pc]
    if !insn
      break
    end
    eval_insn(cur_frame, insn)
  end
end
```

# 共通の処理が多い( i32⇔i64、f32⇔f64)

- 数値の型が  
違うだけの命令
- Generatorで  
まとめて  
作るようにした

```
when :i32_add
  right, left = runtime.stack.pop, runtime.stack.pop
  if !right.is_a?(I32) || !left.is_a?(I32)
    raise EvalError, "maybe empty or invalid stack"
  end
  runtime.stack.push(I32(left.value + right.value))

# ...

when :i64_add
  right, left = runtime.stack.pop, runtime.stack.pop
  if !right.is_a?(I64) || !left.is_a?(I64)
    raise EvalError, "maybe empty or invalid stack"
  end
  runtime.stack.push(I64(left.value + right.value))
```

# 命令を楽しく実装していた頃



Uchio Kondo

@udzura

プロモーションする

#Wardite あと1.0標準命令で実装してないの6つだけだ  
table: [pengowray.github.io/wasm-ops/](https://pengowray.github.io/wasm-ops/)  
なおテストも書いたとは言っていない

|                | .0            | .1          | .2                   | .3          | .4              | .5           | .6          | .7           | .8          | .9           | .A           | .B           | .C           | .D           | .E           | .F           |
|----------------|---------------|-------------|----------------------|-------------|-----------------|--------------|-------------|--------------|-------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| 0. unreach     | no            | block       | loop                 | if          | else            | try          | catch       | throw        | throw_ref   | throw_ref    | return       | br           | br_if        | br_table     | return       |              |
| 1. call        | call_indirect | return_call | return_call_indirect | call_ref    | return_call_ref |              |             | delegate     | catch_all   | gcp          | select       | select1      |              |              |              | try_table    |
| 2. local_get   | local_set     | local_get   | global_get           | global_set  | table_get       | table_set    |             | i32_load     | i64_load    | i32_store    | i64_store    | i32_load     | i64_load     | i32_store    | i64_store    | i32_load     |
| 3. i64_load    | i64_store     | i32_load    | i32_store            | i32_load    | i32_store       | i32_store    | i32_store   | i32_store    | i32_store   | i32_store    | i32_store    | i32_store    | i32_store    | i32_store    | i32_store    | i32_store    |
| 4. memarg      | const         | const       | const                | const       | const           | const        | const       | const        | const       | const        | const        | const        | const        | const        | const        | const        |
| 5. i64_eqz     | i64_eq        | i64_ne      | i64_le               | i64_ge      | i64_lt          | i64_gt       | i64_le      | i64_ge       | i64_lt      | i64_gt       | i64_le       | i64_ge       | i64_lt       | i64_gt       | i64_le       | i64_ge       |
| 6. i32_eq      | i32_ne        | i32_le      | i32_ge               | i32_lt      | i32_gt          | i32_le       | i32_ge      | i32_lt       | i32_gt      | i32_le       | i32_ge       | i32_lt       | i32_gt       | i32_le       | i32_ge       | i32_lt       |
| 7. i32_rem     | i32_and       | i32_or      | i32_xor              | i32_shl     | i32_shr         | i32_rol      | i32_ror     | i32_rol      | i32_ror     | i64_popcount | i64_popcount | i64_popcount | i64_popcount | i64_popcount | i64_popcount | i64_popcount |
| 8. i64_div     | i64_rem       | i64_and     | i64_or               | i64_xor     | i64_shl         | i64_shr      | i64_rol     | i64_ror      | i64_rol     | i64_ror      | i64_rol      | i64_ror      | i64_rol      | i64_ror      | i64_rol      | i64_ror      |
| 9. i32_nearest | i32_sqrt      | i32_add     | i32_sub              | i32_mul     | i32_div         | i32_min      | i32_max     | i32_copysign | i32_abs     | i64_neg      | i64_cell     | i64_floor    | i64_ceil     | i64_trunc    | i64_nearest  | i64_sqrt     |
| A. i64_add     | i64_sub       | i64_mul     | i64_div              | i64_min     | i64_max         | i64_copysign | i64_wrap    | i64_wrap     | i64_trunc   | i64_trunc    | i64_trunc    | i64_trunc    | i64_trunc    | i64_trunc    | i64_trunc    | i64_trunc    |
| B. i64_trunc   | i64_convert   | i32_convert | i32_convert          | i32_convert | i32_convert     | i32_convert  | i32_convert | i32_convert  | i32_convert | i32_convert  | i32_convert  | i32_convert  | i32_convert  | i32_convert  | i32_convert  | i32_convert  |
| C. extend8     | extend16      | extend32    | extend64             | extend64    | extend64        | extend64     | extend64    | extend64     | extend64    | extend64     | extend64     | extend64     | extend64     | extend64     | extend64     | extend64     |

午後8:35 · 2024年11月9日 · 627 件の表示

# 命令を楽しく実装していた頃

- なんか入社直後に  
趣味コードめっちゃ  
書いているな...

Uchio Kondo ✨  
@udzura

プロモーションする ...

#Wardite あと1.0標準命令で実装してないの6つだけだ  
table: [pengowray.github.io/wasm-ops/](https://pengowray.github.io/wasm-ops/)  
なおテストも書いたとは言っていない

|              | .0            | .1                   | .2              | .3              | .4              | .5              | .6              | .7              | .8              | .9              | .A              | .B              | .C              | .D              | .E              | .F              |
|--------------|---------------|----------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0. unreach   | no            | block                | loop            | try             | catch           | throw           | throw_ref       | throw_ref       | throw_ref       | throw_ref       | throw_ref       | throw_ref       | throw_ref       | throw_ref       | throw_ref       | throw_ref       |
| 1. call      | call_indirect | return_call_indirect | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref | return_call_ref |
| 2. local_get | local_set     | local_get            | global_get      | global_set      | table_get       | table_set       | table_get       | table_set       | table_get       | table_set       | table_get       | table_set       | table_get       | table_set       | table_get       | table_set       |
| 3. load8     | load16        | load32               | load64          | load8_u         | load16_u        | load32_u        | load64_u        | store8          | store16         | store32         | store64         | store8          | store16         | store32         | store64         | store8          |
| 4. memeq     | const         | const                | const           | const           | const           | const           | const           | const           | const           | const           | const           | const           | const           | const           | const           | const           |
| 5. eqz       | eq            | ne                   | lt              | gt              | le              | ge              | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             |
| 6. ge        | eq            | ne                   | lt              | gt              | le              | ge              | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             | eqz             |
| 7. rem       | and           | or                   | xor             | shr             | shl             | shr             | shl             | shr             | shl             | shr             | shl             | shr             | shl             | shr             | shl             | shr             |
| 8. div       | rem           | and                  | or              | xor             | shr             | shl             | shr             | shl             | shr             | shl             | shr             | shl             | shr             | shl             | shr             | shl             |
| 9. nearest   | sqrt          | add                  | sub             | mul             | div             | min             | max             | copysign        | abs             | neg             | ceil            | floor           | trunc           | nearest         | nearest         | nearest         |
| A. add       | sub           | mul                  | div             | min             | max             | copysign        | wrap            | wrap            | wrap            | wrap            | wrap            | wrap            | wrap            | wrap            | wrap            | wrap            |
| B. trunc     | convert       | convert              | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         | convert         |
| C. extend8   | extend16      | extend32             | extend64        | extend8_u       | extend16_u      | extend32_u      | extend64_u      | extend8_u       | extend16_u      | extend32_u      | extend64_u      | extend8_u       | extend16_u      | extend32_u      | extend64_u      | extend8_u       |

午後8:35 · 2024年11月9日 · 627 件の表示

Wardite で ruby.wasm は  
動く？

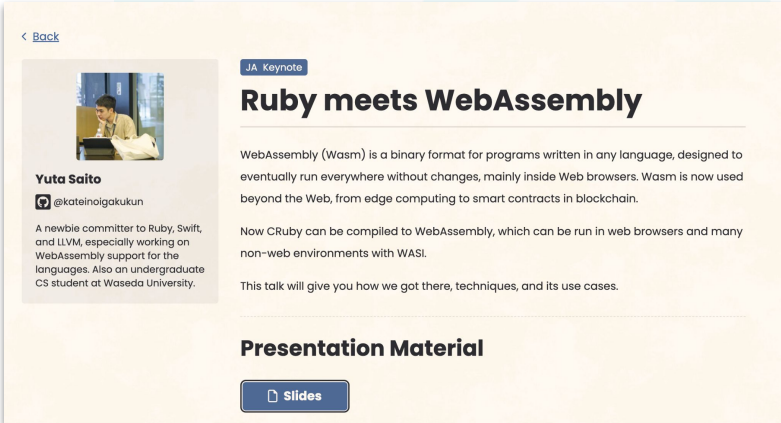
# 再掲: Ruby で WebAssembly といえば

- ruby.wasm ですよ
- ですよ～

< Back

JA. Keynote

## Ruby meets WebAssembly



**Yuta Saito**  
@kateinoigakukun

A newbie committer to Ruby, Swift, and LLVM, especially working on WebAssembly support for the languages. Also an undergraduate CS student at Waseda University.

WebAssembly (Wasm) is a binary format for programs written in any language, designed to eventually run everywhere without changes, mainly inside Web browsers. Wasm is now used beyond the Web, from edge computing to smart contracts in blockchain.

Now CRuby can be compiled to WebAssembly, which can be run in web browsers and many non-web environments with WASI.

This talk will give you how we got there, techniques, and its use cases.

### Presentation Material

Slides

<https://ruby.github.io/ruby.wasm/>

# ruby.wasm をRubyで動かしたい ...

- ruby.wasm を動かせるようにするのがマイルストーン
  - 現実世界の WASMプログラムを動かしたい
  - ~~RubyでRubyが動いたぞ！って言いたいただけ~~
- そのためには WASI preview 1への対応が必要
  - Wardite の中で使う WASI関数もRubyで書く必要

# WASIの関数の例

- 例: `random_get()`
- OSが用意した乱数を取り出すための関数
  - C言語(Linux)の `getrandom(2)`
- プログラムで普通に乱数を使いたいとき使う  
(自力でアルゴリズムを実装しないなら)



# WASIの関数を実装するには

- 普通のRubyの  
コードで乱数  
を扱う

```
class WasiSnapshotPreview1
  # @rbs store: Store
  # @rbs args: Array[wasmValue]
  # @rbs return: Object
  def random_get(store, args)
    buf, buflen = args[0].value, args[1].value
    randoms = SecureRandom.random_bytes(buflen) #: String
    store.memories[0].data[buf...(buf+buflen)] = randoms
    0
  end
end
```

- こういう感じの関数を 90個ぐらい用意すれば完了 ...

## で、ruby.wasmは動いたの？

- デモをします！
- Live ruby.wasm bootstrap challenge!!!!



**Demo.....**



# まとめと今後



# Pure RubyでWASMランタイム作った

- 勉強目的だったが色々頑張ったので割と作り込まれてる
- `ruby.wasm` も動.....!!!???
- 興味ある方、Contribution 歓迎です！

# 今後やりたいことなど

- テスト不足を対応
  - WASM Core specをちゃんと満たした！と言いたい
  - 満たすか？ のテストは自動実行できるので対応
  - → [WebAssembly specification, reference interpreter, and test suite](#)
- あとはパフォーマンス向上とか