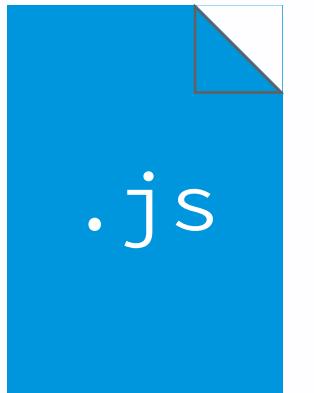


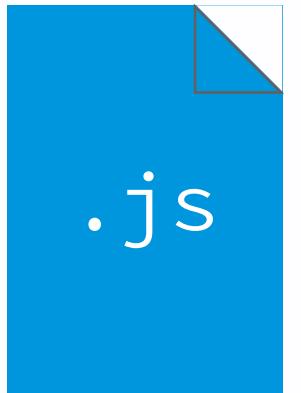
# ブラウザ用のCPUを作るよ！ WebAssemblyで

HTML5 Conference, 2016/09/03 @東京電機大学北千住キャンパス

Mozilla Japan  
N.Shimizu ([nshimizu@mozilla-japan.org](mailto:nshimizu@mozilla-japan.org) / [@chikoski](https://twitter.com/chikoski))



JavaScript VM



JavaScript VM

WASM VM

# N. Shimizu / @chikoski



- Mozilla Japan
  - Developer relation
  - Localizer: MDN, SUMO, Firefox for iOS
- html5j Web プラットフォーム部 / ゲーム部
- プログラミング言語そのものが好きです
- <https://slideshare.net/chikoski/>
- <https://html5experts.jp/chikoski/>



× HTML5Experts.jp > 清水智公 > Webブラウザで高速な演算を可能にする低...



## Webブラウザで高速な演算を可能にする低水準言語asm.jsと、WebAssembly詳解—JavaScriptが動く仕組み

清水智公 (Mozilla Japan) 

[asm.js](#), [WebAssembly](#)

2016年7月7日

 60  0  3  125  206

Webブラウザの上で動作するアプリを書くための言語、といえば何が想起されるでしょうか。Flash、Silverlight、Java、さまざまな言語が利用されてきましたが、やはり今のメインストリームはJavaScriptでしょう。

JavaScriptはさまざまな言語の特徴を併せ持つ動的言語で、Web技術の発展とAPIの整備の結果、Virtual Reality(VR)や画像認識、DAW/Desktop Audio Workstationといった、少し前まではネイティブでの実装しかありえなかった種類のアプリケーションもWebブラウザをランタイムとするJavaScriptで実装されるようになりました。

× 週間PVランキング

> more

- 1 モバイルWebのUIを速くする基本テクニックがわかる—Google I/O 2016 High Performance Web UI
- 2 Webブラウザで高速な演算を可能にする低水準言語asm.jsと、WebAssembly詳解—JavaScriptが動く仕組み
- 3 Windows 10の2つのWebブラウザ、Microsoft EdgeとInternet Explorer 11
- 4 今話題のReact.jsはどのようなWebアプリケーションに適しているか？ Introduction To React—Frontend Conference
- 5 初心者でもわかる・できる！Arduinoを使った初めての電子工作実践

× 新着記事

> more

mozilla



*Proudly non-profit, Mozilla makes products like Firefox with a mission to keep the power of the Web in the hands of users everywhere.*

*Mozilla Mission (<https://www.mozilla.org/en-US/mission/>)*

*Our mission is to promote **openness**,  
**innovation** & **opportunity** on the Web.*

*Mozilla Mission(<https://www.mozilla.org/en-US/mission/>)*



# WEBASSEMBLY

<https://github.com/WebAssembly/design/issues/112>

<https://github.com/Fogaccio/OpenDesign>

# WebAssembly

- W3C のコミュニティグループで議論されている新しいWeb標準
- 小さくてポータブルなバイナリフォーマット
- ネイティブに近いスピードでの実行
- 安全な実行環境
- 1対1で対応するテキスト表現



# COMMUNITY & BUSINESS GROUPS

[CURRENT GROUPS](#)[REPORTS](#)[ABOUT](#)[Home](#) / WebAssembly Community Group

## WEBASSEMBLY COMMUNITY GROUP

The mission of this group is to promote early-stage cross-browser collaboration on a new, portable, size- and load-time-efficient format suitable for compilation to the web.

*Note: Community Groups are proposed and run by the community. Although W3C hosts these conversations, the groups do not necessarily represent the views of the W3C Membership or staff.*

**No Reports Yet Published**

Chairs, when logged in, may publish draft and final reports. Please see [report requirements](#).

[PUBLISH REPORTS](#)

<https://www.w3.org/community/webassembly/>

### Resources

[Code of Conduct](#)

Get involved



Luke Wagner

*chairs*



BRHAM GIRI  
ABHIJITH CHATRA



Ben Titzer



Filip Jerzy Pizlo



Luke Wagner



BRHAM GIRI  
ABHIJITH CHATRA



Ben Titzer



Filip Jerzy Pizlo

*chairs*



Mozilla



Microsoft



Google



Apple

# WebAssembly

[Overview](#)[Demo](#)[Design](#)[Specification](#)[Community Group](#)[Step by Step](#)

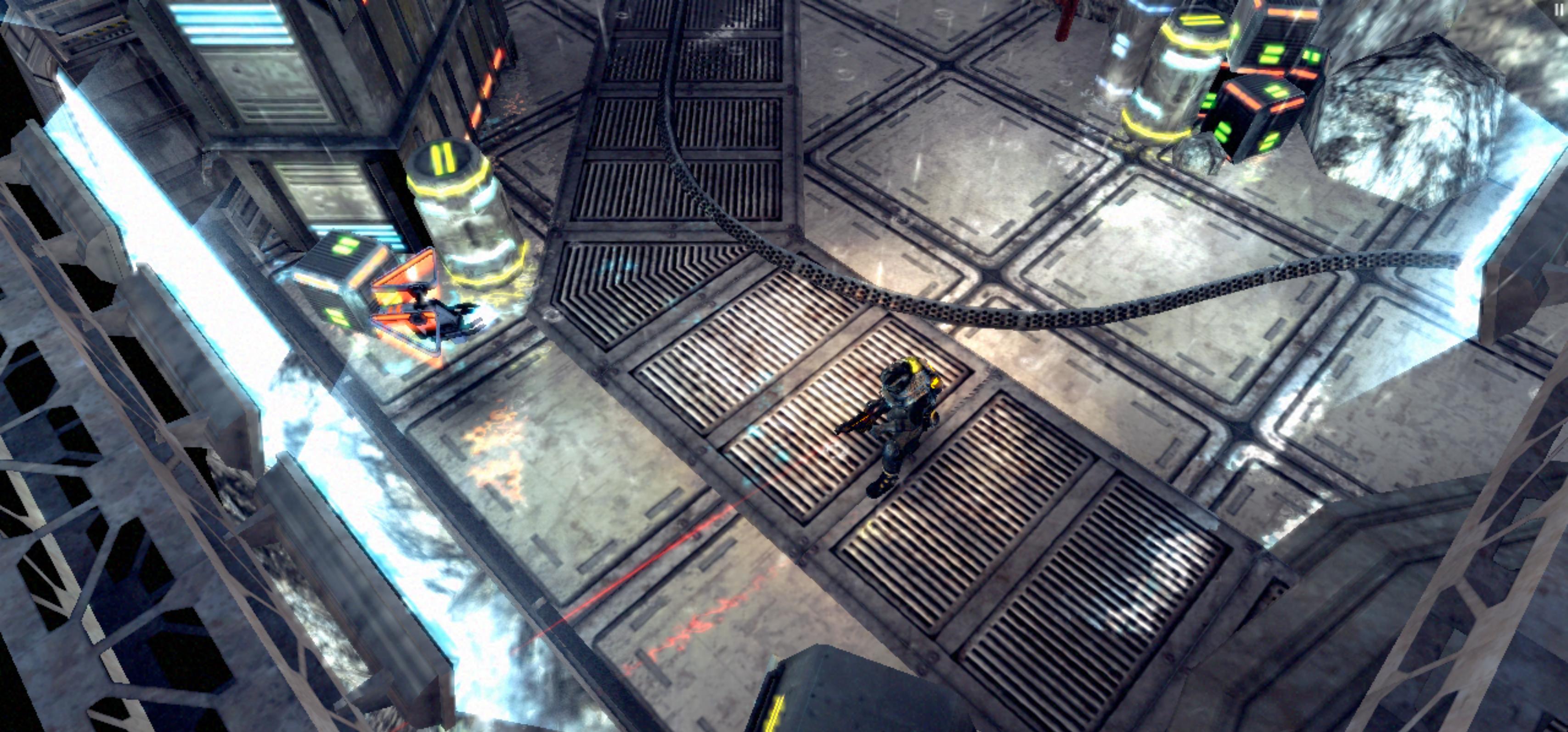
WebAssembly or *wasm* is a new portable, size- and load-time-efficient format suitable for compilation to the web.

WebAssembly is currently being designed as an open standard by a [W3C Community Group](#) that includes representatives from all major browsers. Expect the contents of this website and its associated design repositories to be in flux: everything is still under discussion and subject to change.

## Overview

- **Efficient and fast:** The *wasm AST* is designed to be encoded in a size- and load-time-efficient [binary format](#). WebAssembly aims to execute at native speed by taking advantage of [common hardware capabilities](#) available on a wide range of platforms.

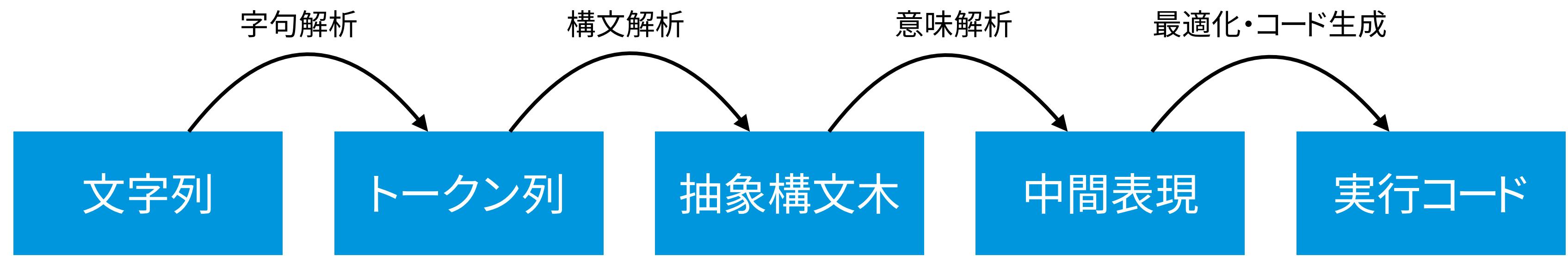
- <https://www.w3.org/community/webassembly/> will be implemented inside existing JavaScript virtual machines. When embedded in the web, WebAssembly will enforce the same-origin and permissions security policies of the browser.

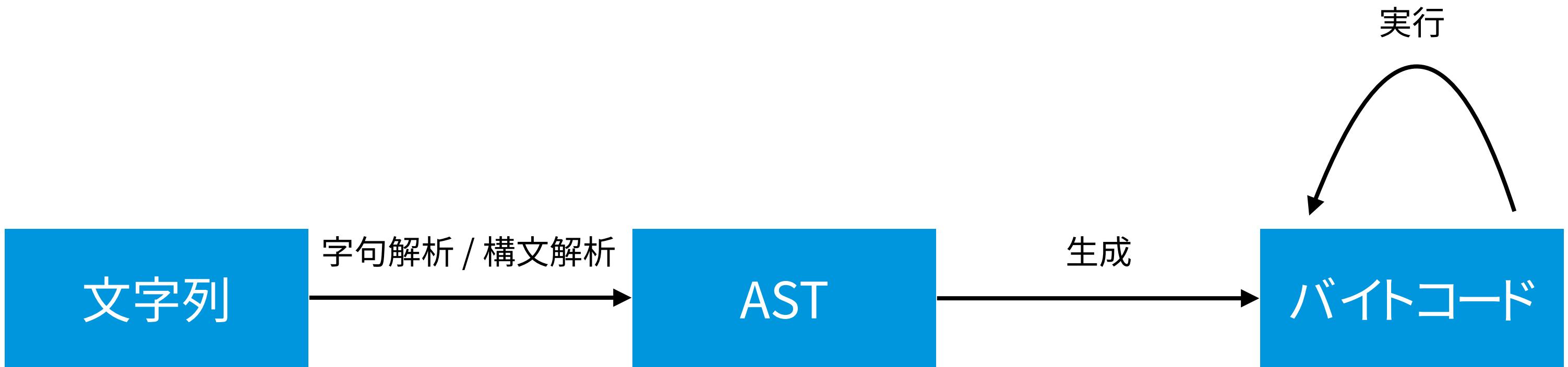


<https://webassembly.github.io/demo/>

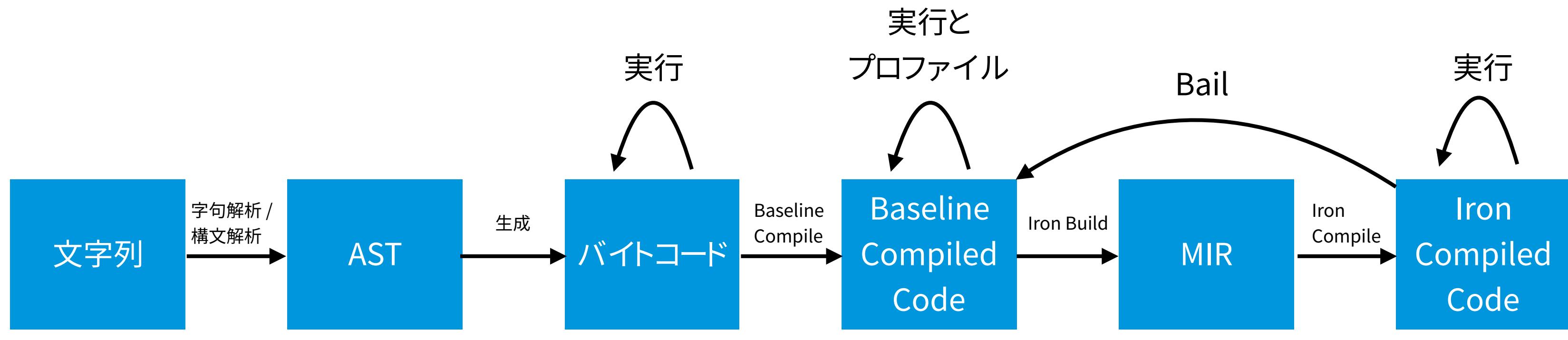
Demo	asm.js	binary	gzip asm.js	gzip binary
<a href="#">AngryBots</a>	19MiB	6.3MiB	4.1MiB	3.0MiB
<a href="#">PlatformerGame</a>	49MiB	18MiB	11MiB	7.3MiB

<https://github.com/WebAssembly/design/blob/master/FAQ.md>



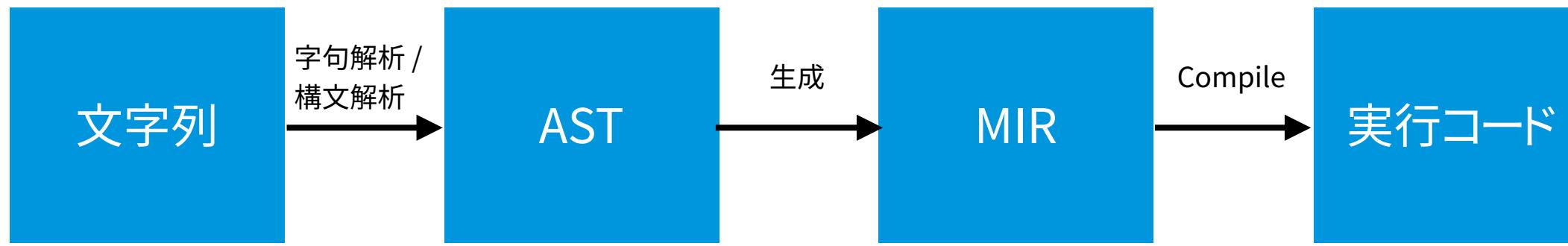


\* AST: Abstract Syntax Tree / 抽象構文木



\* AST: Abstract Syntax Tree / 抽象構文木

MIR: Medium-level Intermediate Representation / 中間表現



\* AST: Abstract Syntax Tree / 抽象構文木

MIR: Medium-level Intermediate Representation / 中間表現

JS

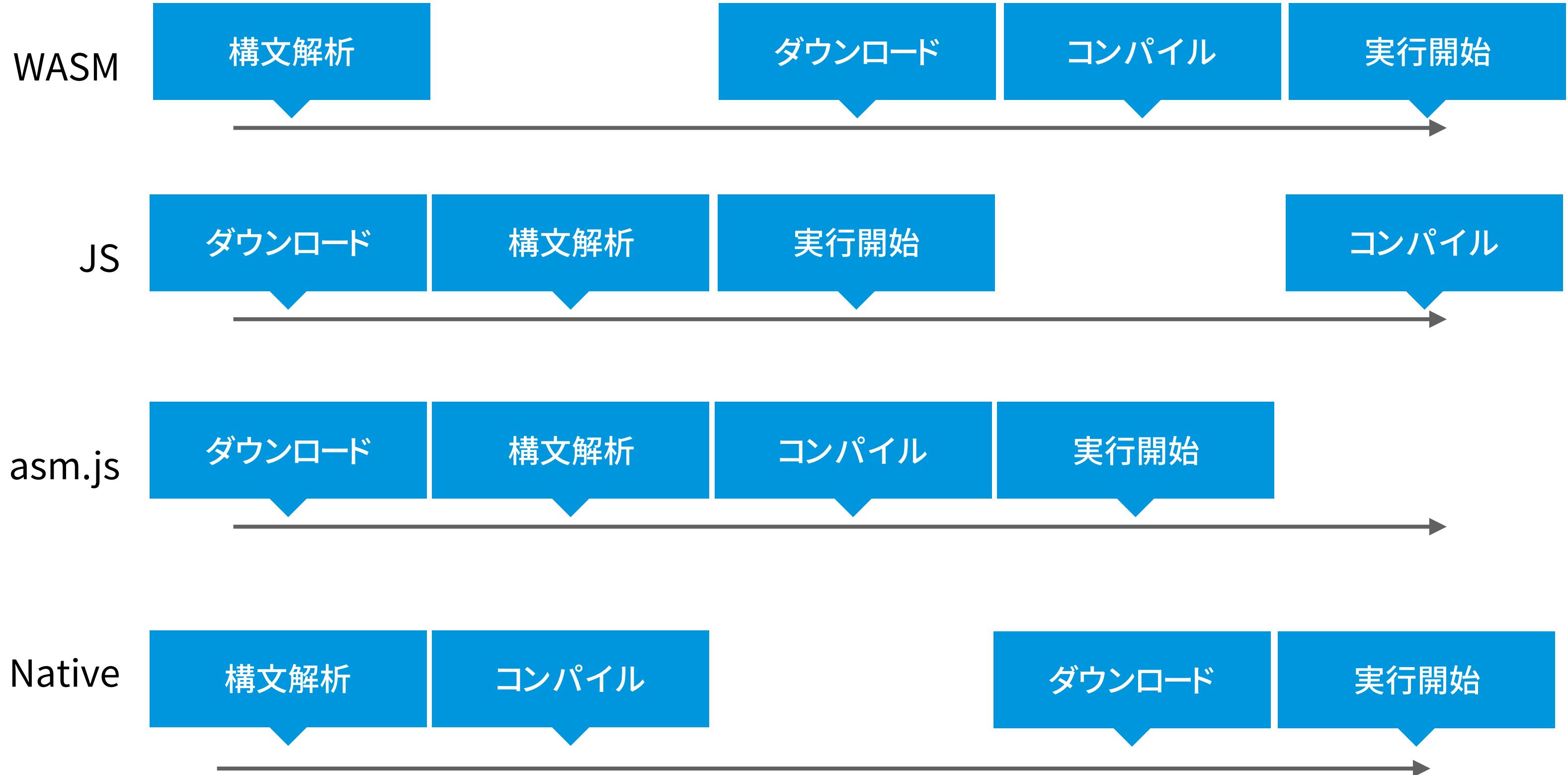


asm.js

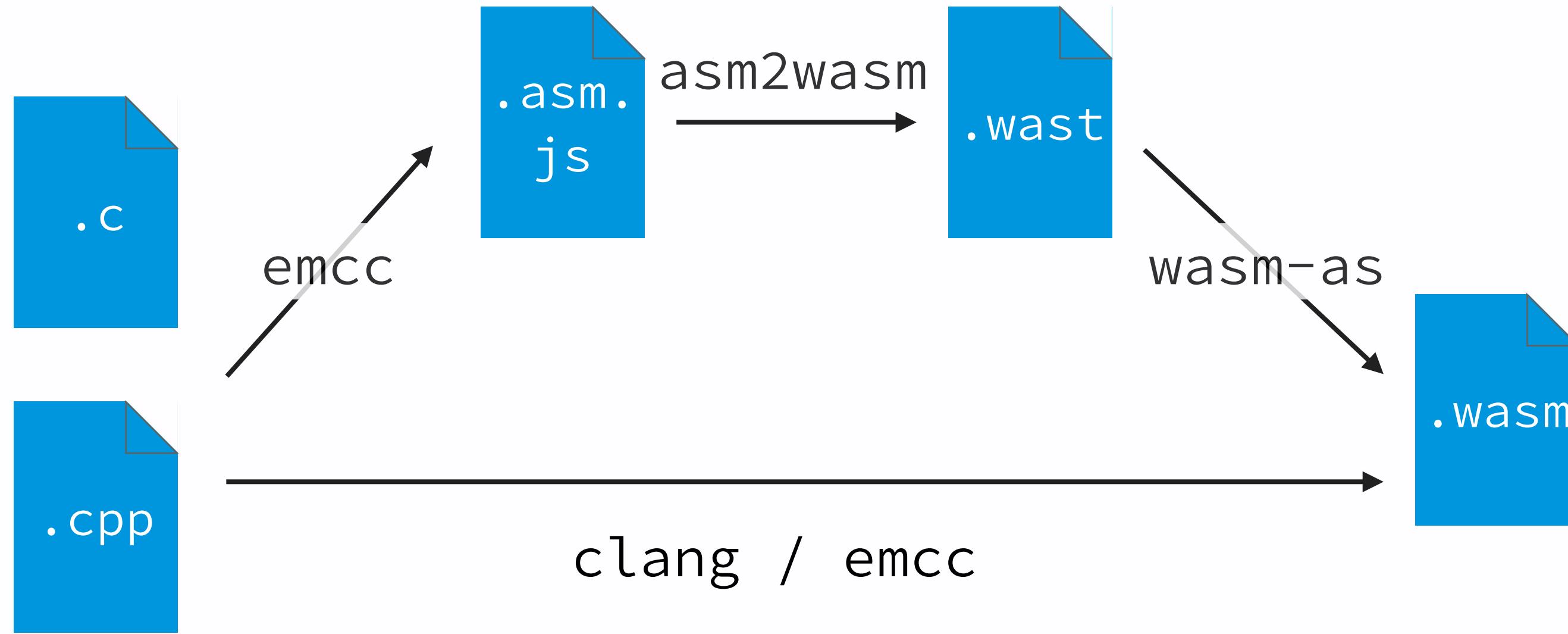


Native





# WASM：コンパイルの生成物



変換パイプライン



**emscripten**

```
% emsdk install emscripten-incoming-64bit  
% emsdk install clang-incoming-64bit  
% emsdk install sdk-incoming-64bit  
% emsdk activate emscripten-incoming-64bit  
% emsdk activate clang-incoming-64bit  
% emsdk activate sdk-incoming-64bit  
% source ${EMSDK_INSTALL_DIR}/emsdk_env.sh
```

Emscripten から wasm を出力する設定

```
% ls
```

```
add.cpp
```

```
% emcc -s BINARYEN=1 -o add.wasm add.cpp
```

```
% ls
```

```
add.cpp add.wasm
```

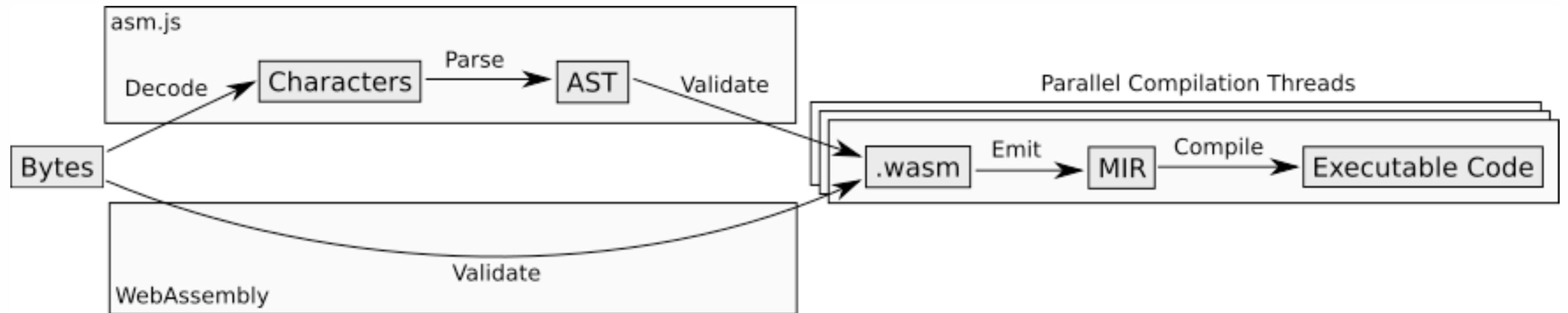
```
poisson% pwd  
/Users/chiko/git/llvm.org/llvm/lib/Target  
poisson% ls  
AArch64           MSP430          TargetLoweringObjectFile.cpp  
AMDGPU            Mips            TargetMachine.cpp  
ARM               NVPTX          TargetMachineC.cpp  
AVR               PowerPC         TargetRecip.cpp  
BPF               README.txt     TargetSubtargetInfo.cpp  
CMakeLists.txt    Sparc           WebAssembly  
Hexagon           SystemZ         X86  
LLVMBuild.txt    Target.cpp      XCore  
Lanai              TargetIntrinsicInfo.cpp  
poisson%
```

TargetLoweringObjectFile.cpp  
TargetMachine.cpp  
TargetMachineC.cpp  
TargetRecip.cpp  
TargetSubtargetInfo.cpp  
WebAssembly  
X86  
XCore



LLVMのターゲットにもWebAssemblyの文字が

# Webページへの組み込み



WebAssemblyは実行コードに変換されてから実行される

```
fetch("add.wasm").then(response =>  
  response.arrayBuffer();  
).then(buffer => {  
  const codeByte = new Uint8Array(buffer);  
  const module = Wasm.instantiateModlue(codeByte);  
  alert(`1 + 2 = ${module.exports.add(1, 2)}`);  
});
```

WASM.instantiateModule でコンパイル

```
export int add(int);  
import int something();  
  
int addSome(int a){  
    return a + something();  
}
```

CからJSの関数を呼び出す場合

```
fetch("addsome.wasm").then(response =>
  response.arrayBuffer();
).then(buffer => {
  const codeByte = new Uint8Array(buffer);
  const module = WASM.instantiateModlue(codeByte, {
    something: () => 3
  });
  alert(`1 + 2 = ${module.exports.addsome(1)}`);
});
```

CからJSの関数を呼び出す場合

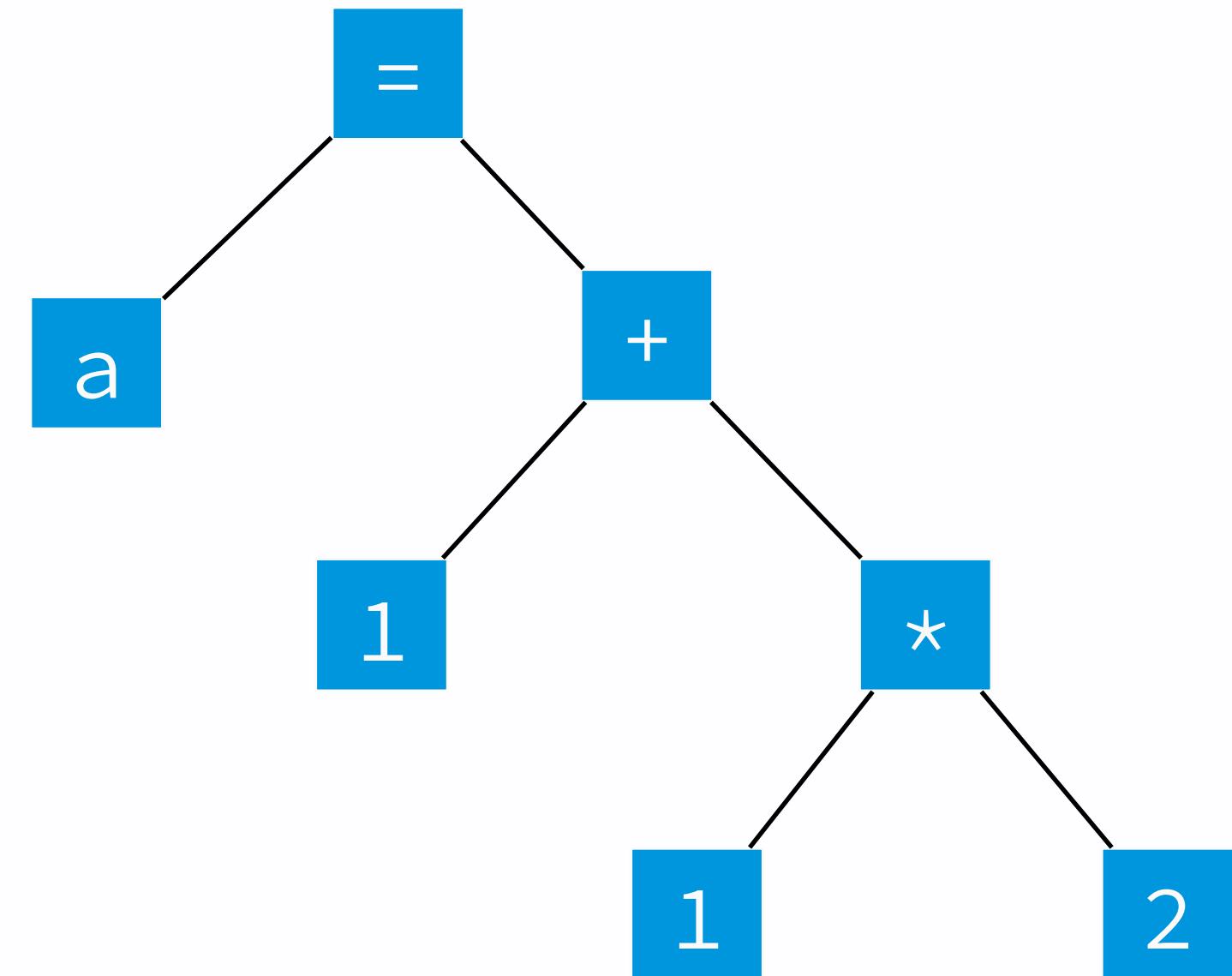
# WebAssemblyの特徴

- 型付けされたスタックマシン
- 連續なメモリモデル
- 抽象構文木 (AST)
- 構造化されたコントロールフローを定義
- モジュールシステム

```
a = 1 + 1 * 2;
```

```
a = 1 + 1 * 2;           (a = (1 + (1 * 2)));
```

```
a = 1 + 1 * 2;
```



抽象構文木 (Abstract Syntax Tree : AST)

<b>WebAssembly</b>	<b>x86</b>	<b>ARM</b>
i32.add	addl	ADD
call	call	BL
i32.load	CHECK + mov	CHECK + LDR

実マシンのコードに近い仮想的な命令セットを定義

## WASM

i32.add

call

i32.load

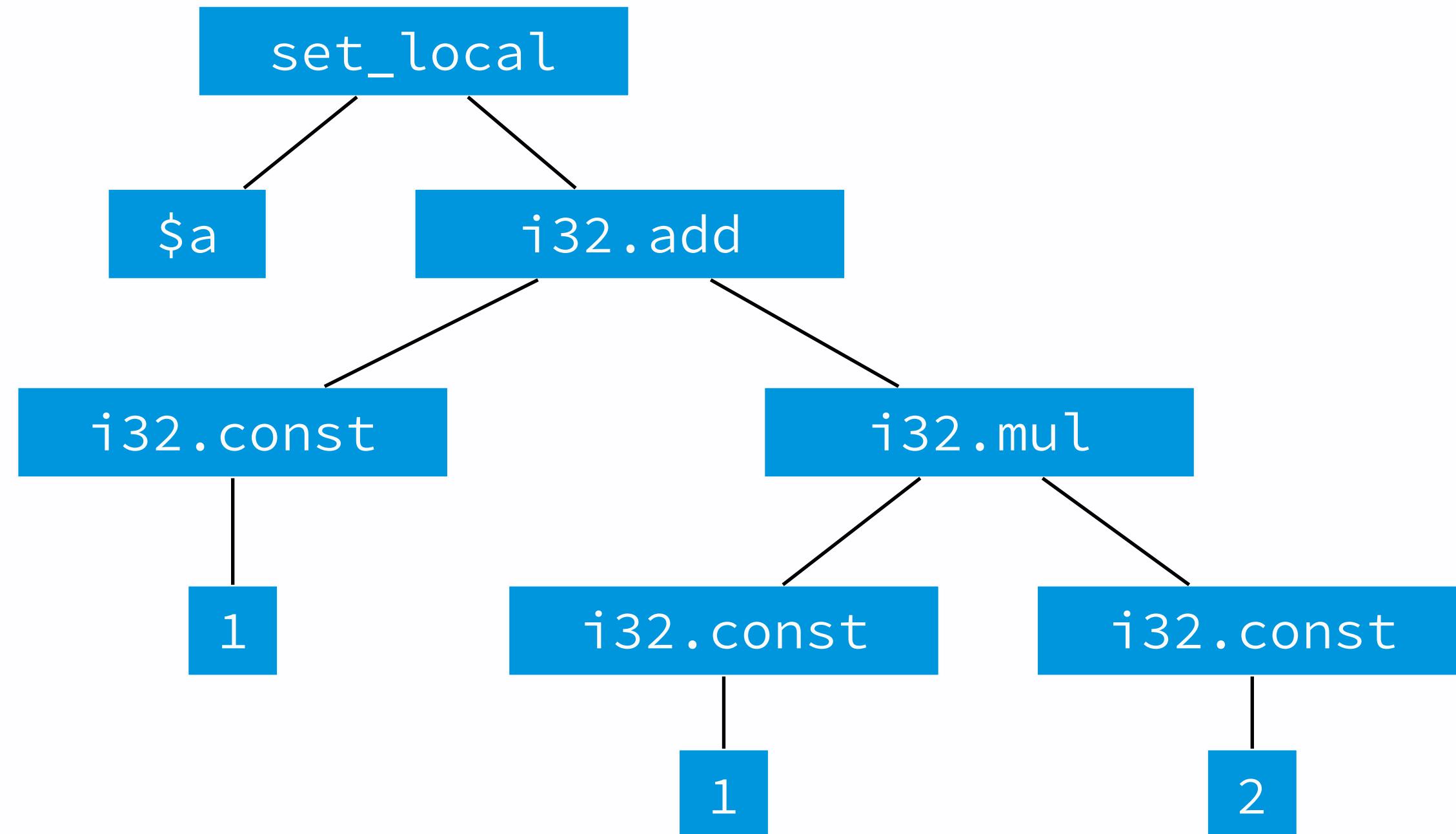
## asm.js

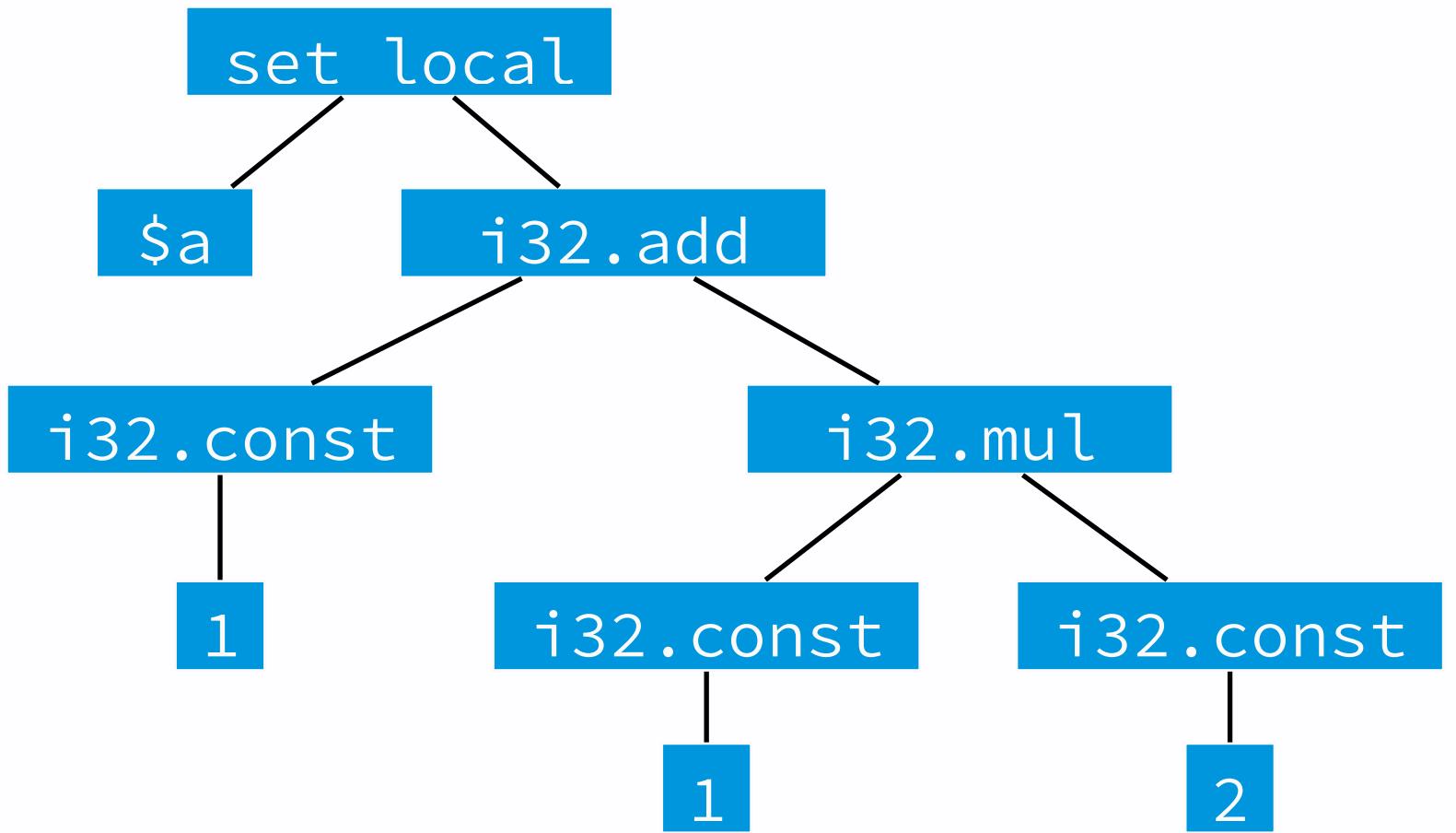
(a + b) | 0

f() | 0

HEAP[i>>2] | 0

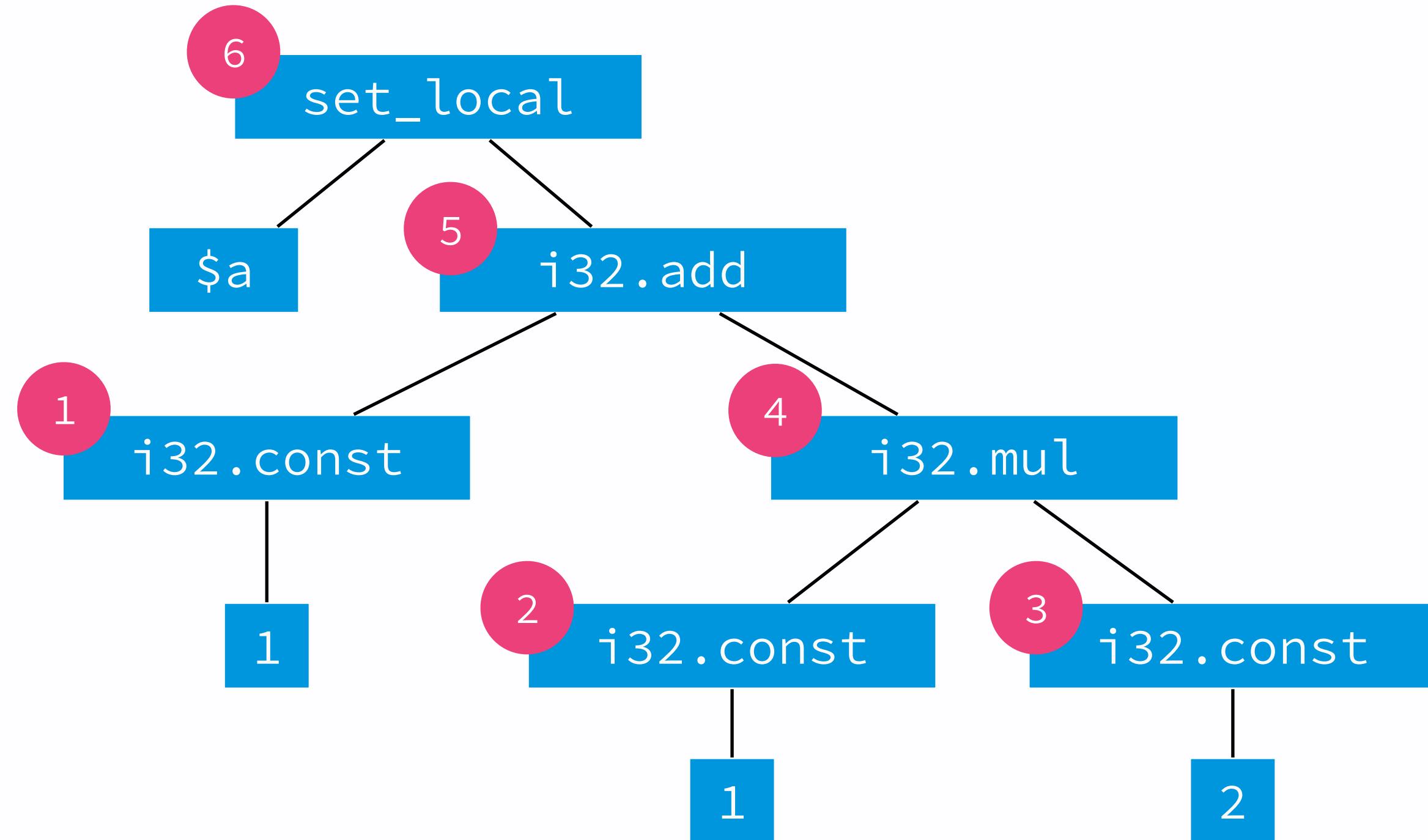
asm.jsとの互換性





```
(set_local $a  
        (i32.add  
            (i32.const 1)  
            (i32.mul  
                (i32.const 1)  
                (i32.const 2)  
            )  
        )  
)
```

テキスト表現(仕様策定中)



子ノードが出現順に評価される

型	扱える値の種類
i32	32ビットの整数値
i64	64ビットの整数値
f32	32ビットの整数値
f64	64ビットの整数値

命令	振る舞い
i32.load8_s	1バイトロードして符号付きi32へ拡張
i32.load8_u	1バイトロードして符号なしi32へ拡張
i32.load16_s	2バイトロードして符号付きi32へ拡張
i32.load16_u	2バイトロードして符号なしi32へ拡張
i32.load	4バイトロード

## 命令

## 振る舞い

i32.store8

i32をi8に変換してストア

i32.store16

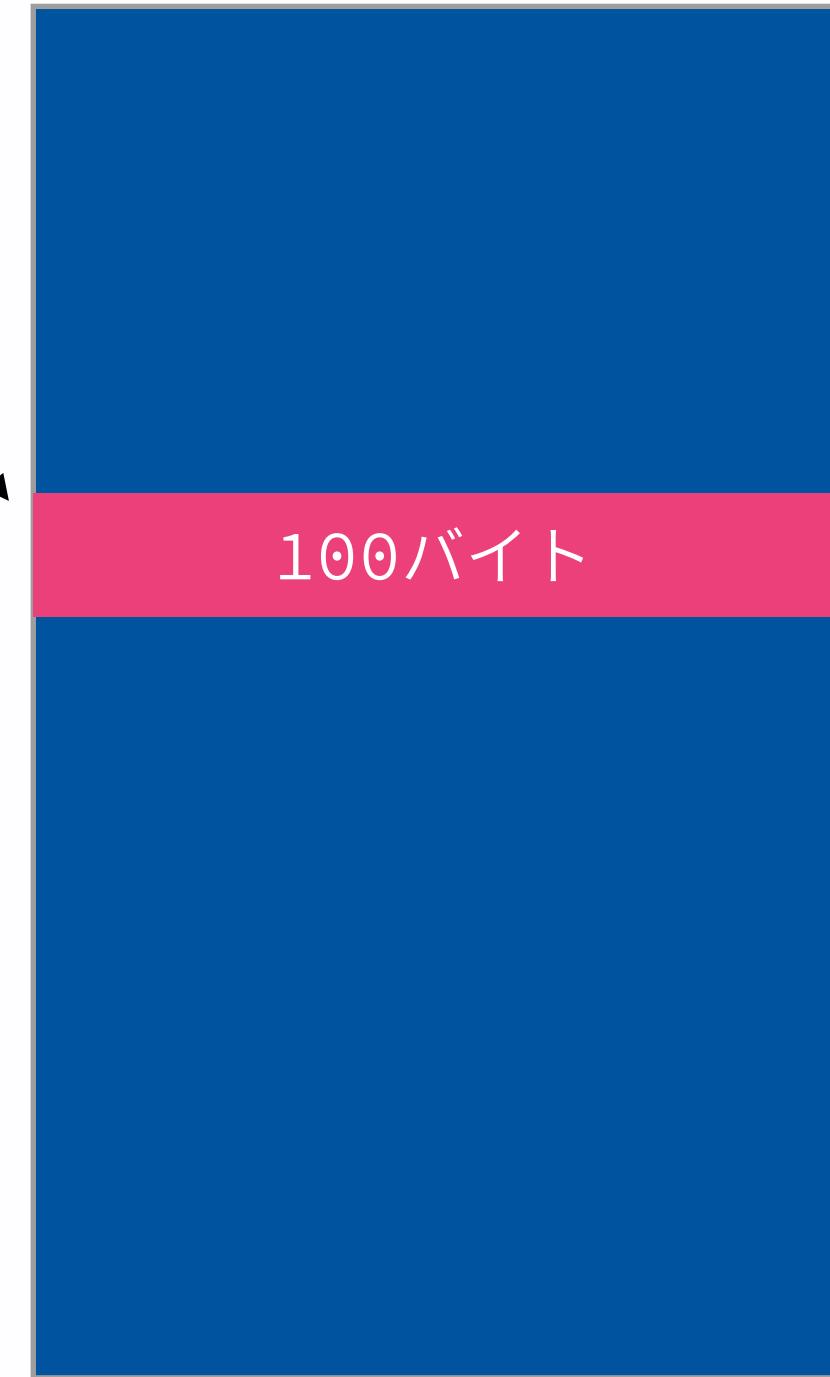
i32をi8に変換してストア

i32.store

変換せずに4バイトをストア

```
ptr = malloc(100)
```

i32.load  
i32.store



0

address\_space\_max  
(< 4GB)

連続的にとられるメモリ

## 命令

grow\_memory

## 振る舞い

メモリを指定されたページ分増やす

current\_memory 現在のメモリサイズをページ数で返す

命令	振る舞い
nop	何もしない
block	ブロックの定義
loop	繰り返し
if	if 文 (then に複数の式がある場合)
br	分岐
br_if	条件付き分岐
br_table	ジャンプテーブル
return	返り値

```
(module
  (memory 256 256)
  (export "memory" memory)
  (export "add" $add)
  (func $add (param $0 i32) (param $1 i32) (result i32)
    (i32.add
      (get_local $0)
      (get_local $1)
    )
  )
)
```

同等のWASMモジュール

# WebAssembly の今後

## MVPの後に予定されている機能

- スレッド
- 共有メモリ
- ダイナミックリンク
- 0コストの例外
- 固定長のSIMD

## 将来的な機能

- 細かい粒度でのメモリ管理
- 大きいサイズのページ
- 制御構造の充実
- GCやDOMの統合
- 4GB以上の線形メモリ
- etc

# まとめ

- WASMはコンパイラの生成物
- Polyfill
- JS の置き換えではない
  - いまのところDOMを直接利用できない
  - 高速で数値計算をしたい場面は限られる
  - 生産性の方を優先する場合も多い

# 関連情報

- WebAssembly Community Group

<https://www.w3.org/community/webassembly/>

- Modest (<https://dev.mozilla.jp/>)
- emscripten (<https://emscripten.org/>)
- WASM Explorer

<http://mbebenita.github.io/WasmExplorer/>