Compile it!

Uladzimir Dziomin

@UladzimirD

https://github.com/spzm

Agenda

1. Introduction
2. Javascript compiling
3. Ams.js
4. WebAssembly

Intro

What is JavaScript

- Language of the web
- Scripting language
- Dynamically typed
- Proto-based object model
- Functional features and clojures

Javascript engine time spends

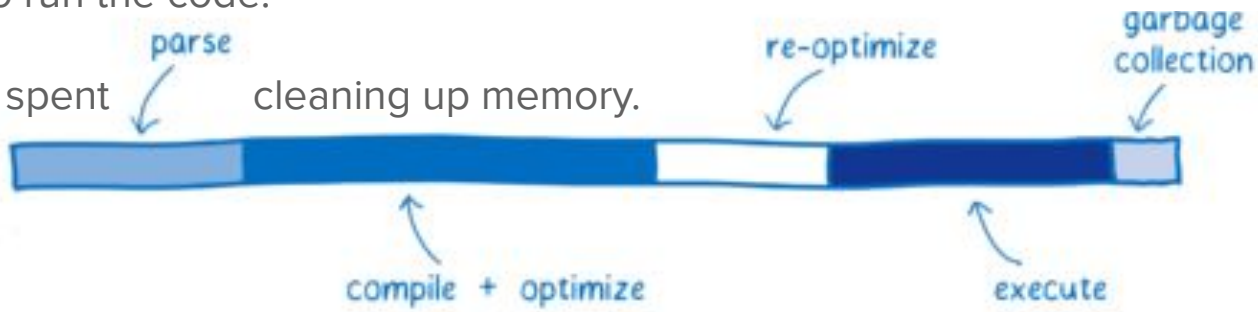
Parsing - process the source code into something that the interpreter can run.

Compiling + optimizing - baseline compiler and optimizing compiler work. Some of the optimizing compiler's work is not on the main thread.

Re-optimizing - JIT readjusting when its assumptions have failed, both re-optimizing code and bailing out of optimized code back to the baseline code.

Execution - the time it takes to run the code.

Garbage collection - the time spent cleaning up memory.



Compilation strategies

Just-in-time (JIT) compilation (also known as dynamic translation) is compilation done **during execution** of a program – at run time – rather than prior to execution.

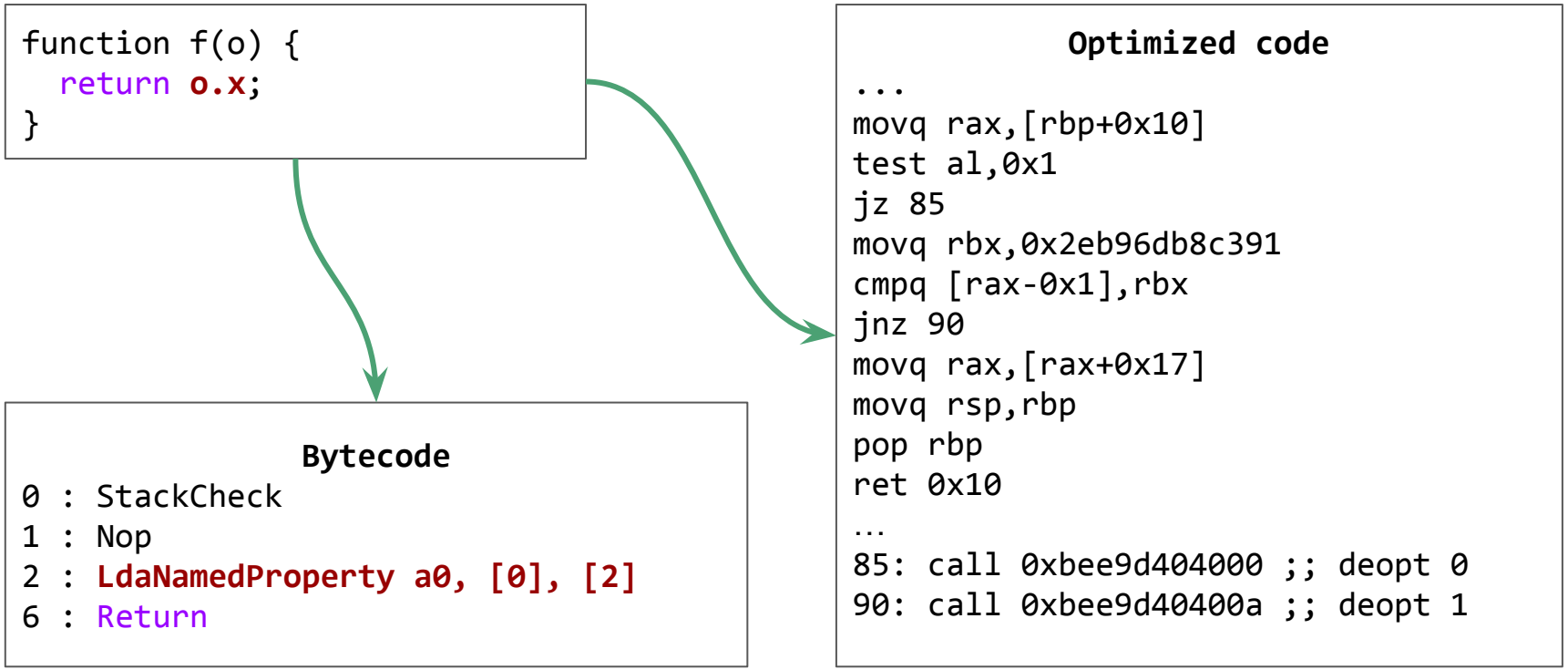
Ahead-of-time (AOT) compilation is the act of compiling a high-level programming language into a **native (system-dependent)** machine code with the intention of executing the resulting binary file natively.

Bytecode

Bytecode is a form of instruction set designed for efficient execution by a software interpreter.

Bytecodes are compact numeric codes, constants, and references (normally numeric addresses) that encode the result of compiler parsing and semantic analysis of things like type, scope, and nesting depths of program objects.

Bytecode vs Optimized code



Javascript

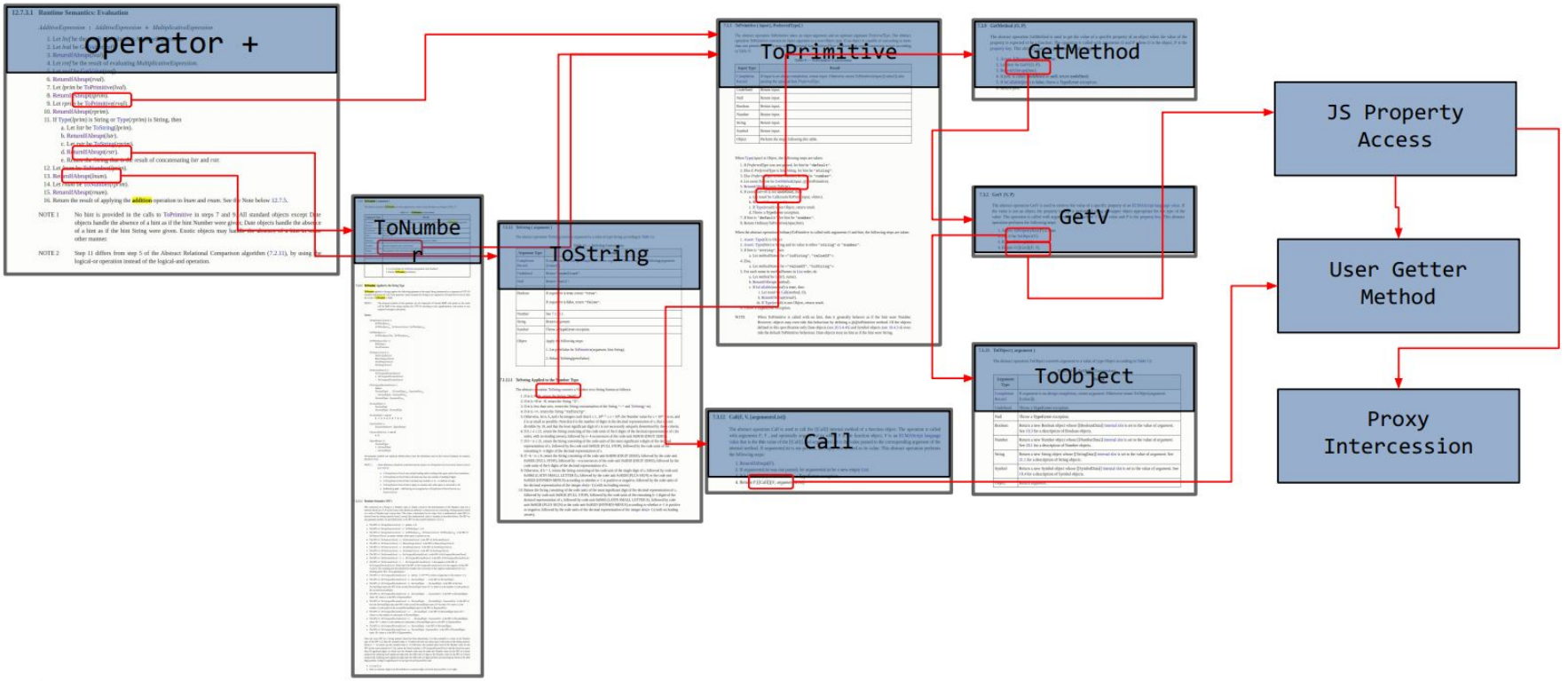
Untyped variables and operations

```
function add(a, b) {
    return a + b;
}

console.log(
    add(1, 2),
    add('foo', 1),
    add(1, 'foo'),
    add({foo: ''}, 1),
    add('hello', { toString: () => 'me' }),
    add(1.01, 3.03)
);
```

```
>> 3 "foo1" "1foo" "[object Object]1" "hellome" 4.04
```

Type cast operation

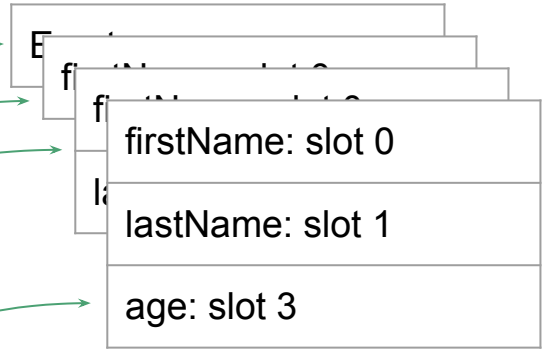


V8 Object Model

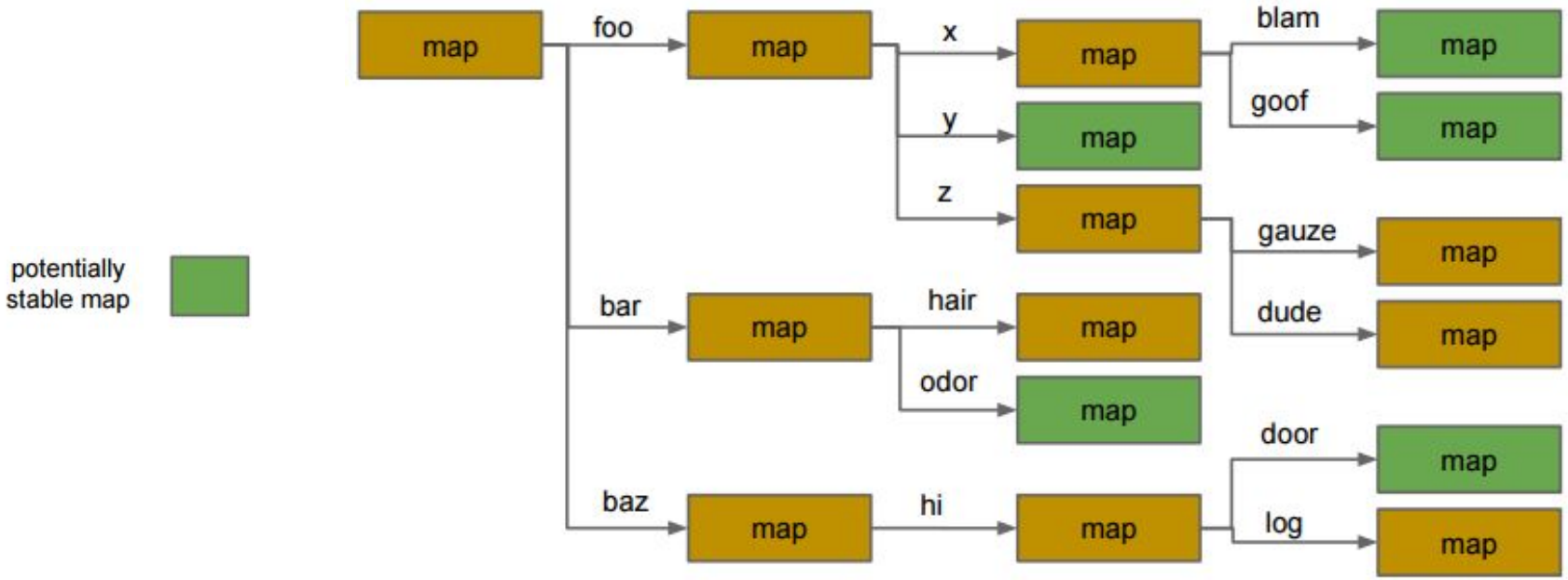
```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName  = lastName;
}

var IronMan        = new Person('Tony',  'Stark');
var CaptainAmerica = new Person('Steve', 'Rogers');
CaptainAmerica.age = 87;

// Warning. IronMan and CaptainAmerica
// now have different hidden classes
```



V8 approach: Map Forest



Example

```
function f(o) {
  return o.x;
}

f({x : 1}); f({x : 2});
%OptimizeFunctionOnNextCall(f);
f({x : 3});
```

Bytecode

```
0 : StackCheck
1 : Nop
2 : LdaNamedProperty a0, [0], [2]
6 : Return
```

Feedback
...
2: { x : Smi }

Optimized code

```
...
movq rax,[rbp+0x10]
test al,0x1
jz 85
movq rbx,0x2eb96db8c391
cmpq [rax-0x1],rbx
jnz 90
movq rax,[rax+0x17]
movq rsp,rbp
pop rbp
ret 0x10
…
85: call 0xbee9d404000 ;; deopt 0
90: call 0xbee9d40400a ;; deopt 1
```

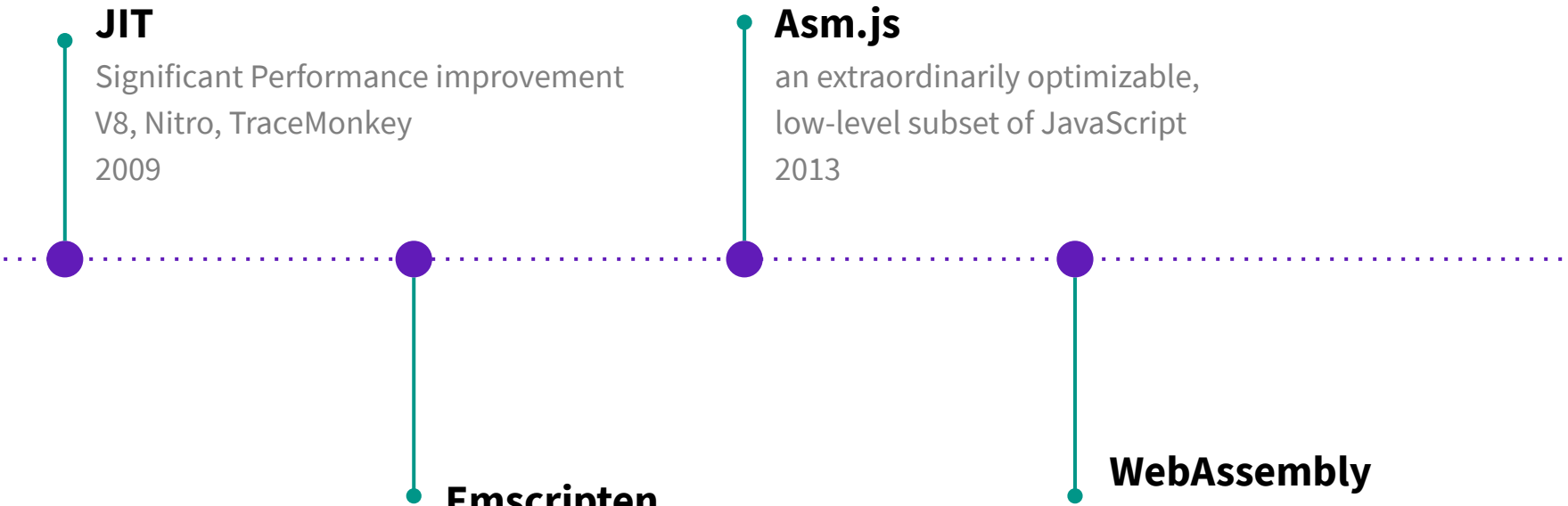
First, we run unoptimized code, gathering feedback.
When hot, we decide to optimize based on feedback.

Optimized code explained:

1. If **o** is Smi, deopt.
2. If **o** does not have the right map, deopt.
3. Load o.x.
4. Tear down frame, return the result.

Asm.js

Timeline


JIT
Significant Performance improvement
V8, Nitro, TraceMonkey
2009
Emscripten
Compile C/C++ to Javascript
2011
Asm.js
an extraordinarily optimizable,
low-level subset of JavaScript
2013
WebAssembly
portable, size-load-time-efficient
format for compilation to the web
2015, Feb 2017

Emscripten

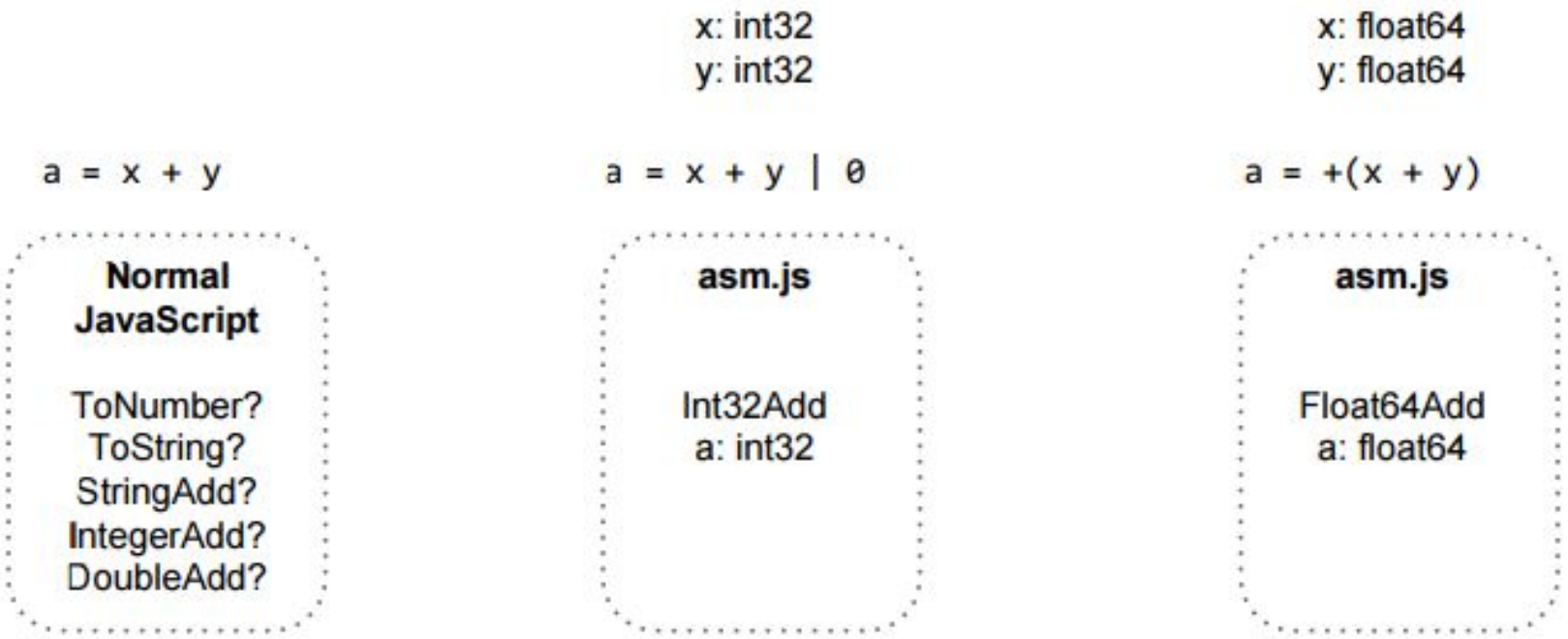
Emscripten is an LLVM-to-JavaScript compiler. It takes LLVM bitcode - which can be generated from C/C++, using llvm-gcc (DragonEgg) or clang, or any other language that can be converted into LLVM - and compiles that into JavaScript, which can be run on the web (or anywhere else JavaScript can run).

Asm.js

Asm.js is an intermediate programming language designed to allow computer software written in languages such as C to be run as web applications while maintaining performance characteristics considerably better than standard JavaScript, the typical language used for such applications.

- No GC
- No Threads
- No Shared Memory

Asm.js


a = x + y
Normal JavaScript
ToNumber?
ToString?
StringAdd?
IntegerAdd?
DoubleAdd?
x: int32
y: int32
a = x + y | 0
asm.js
Int32Add
a: int32
x: float64
y: float64
a = +(x + y)
asm.js
Float64Add
a: float64

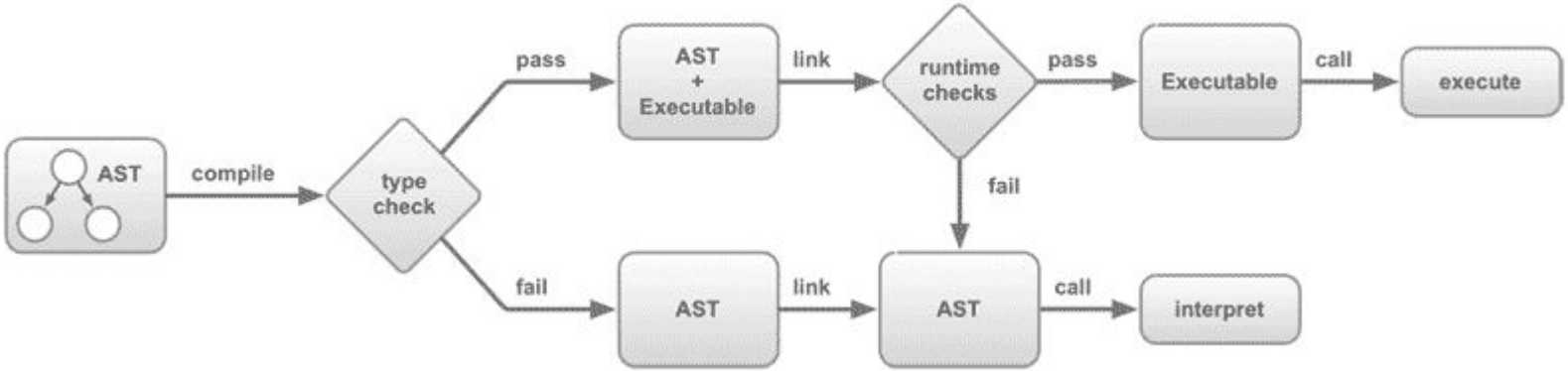
Asm.js module

```
function MyAsmModule(stdlib, foreign, heap) {
    "use asm";
    // ...
    return {
        export1: f1,
        export2: f2,
        // ...
    };
}

var heap = new ArrayBuffer(0x10000);          // 64k heap
init(heap, START, END);                       // fill a region with input values
var fast = GeometricMean(window, null, heap); // produce exports object linked to
                                               // AOT-compiled code
fast.geometricMean(START, END);               // computes geometric mean of values
```

Asm.js validation

- all property access must resolve to data properties;
- the heap object (if provided) must be an instance of ArrayBuffer;
- the heap object's byteLength must be either 2n for n in [12, 24) or 224 · n for n ≥ 1;
- all globals taken from the stdlib object must be the SameValue as the corresponding standard library of the same name.

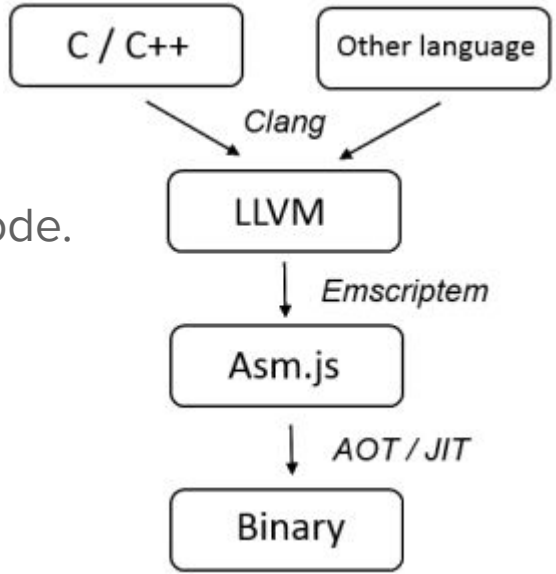


Emscripten: Asm.js approach

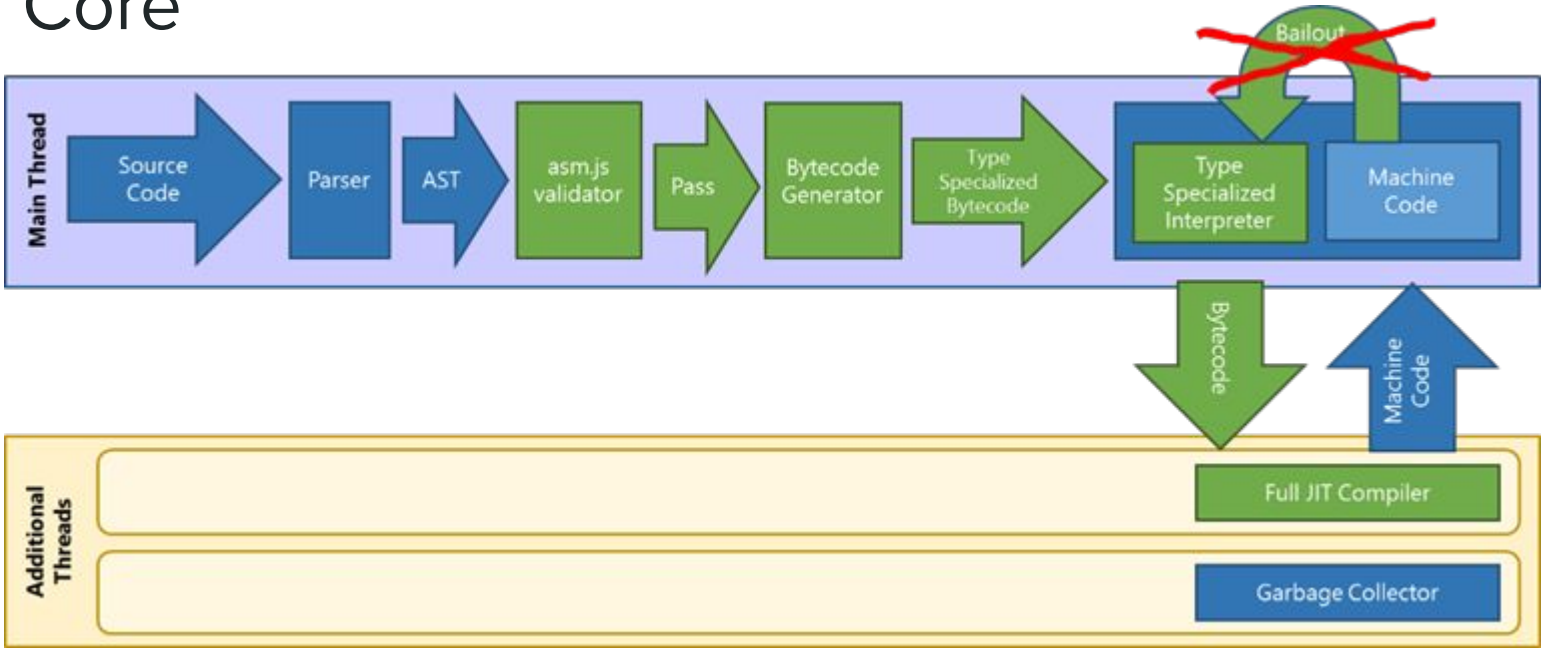
1. Create a C/C++ app.

2. Compile it using Clang to generate LLVM bytecode.

3. Pass the bytecode to Emscripten to get the JavaScript code.



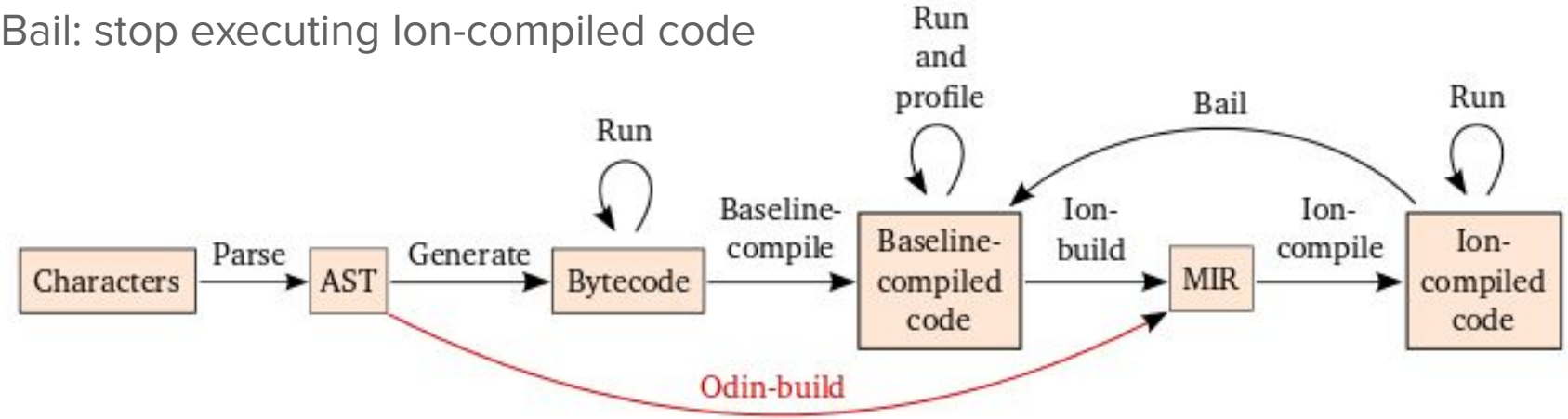
Chakra Core



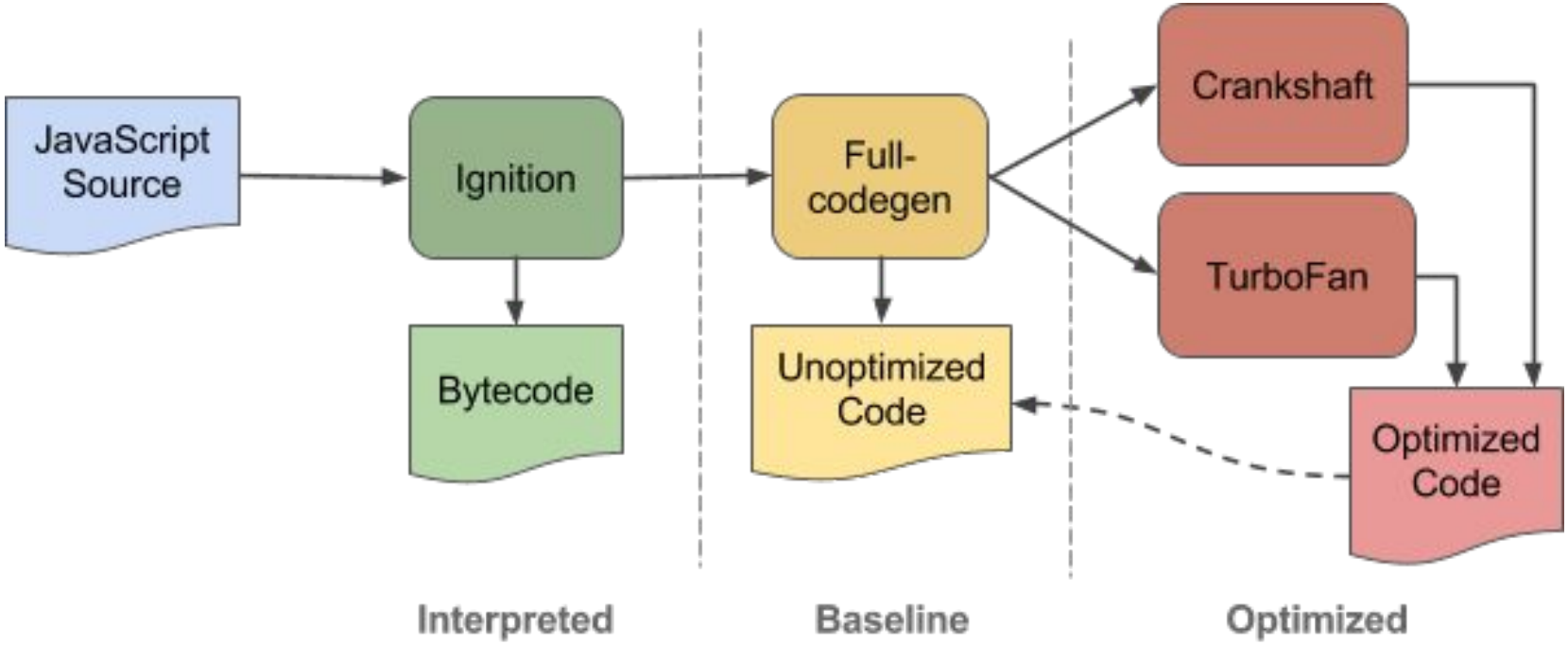
Green - upgraded parts for Asm.js

OdinMonkey

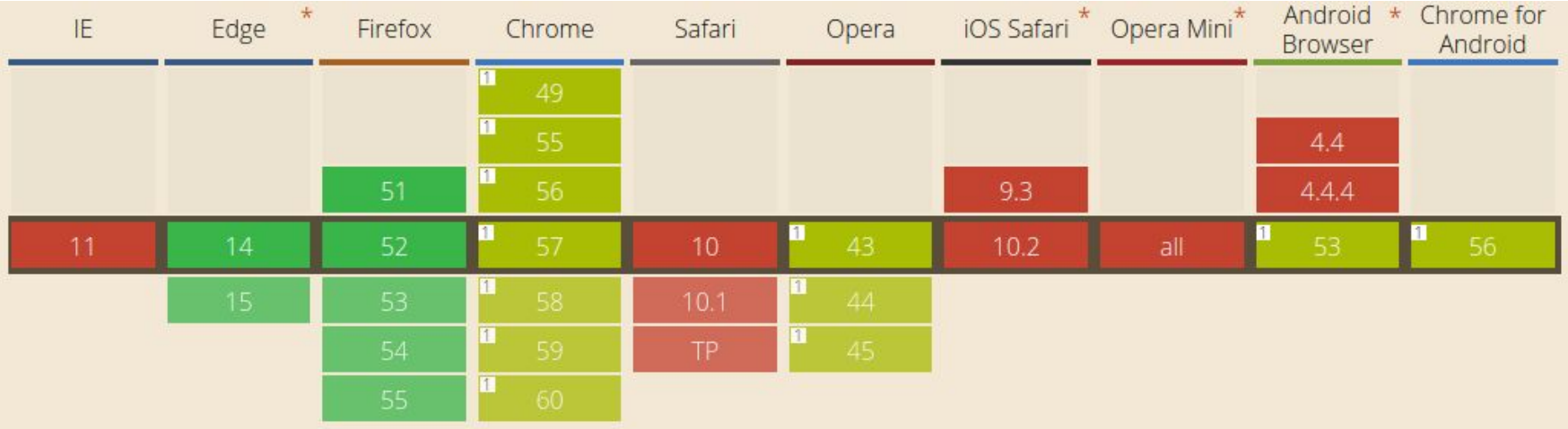
- AST: Abstract Syntax Tree
- Baseline: a JIT compiler that balances compilation speed and the performance
- Ion: a JIT compiler that produces highly-optimized code
- MIR: an SSA-based representation of code used throughout Ion
- Profile: collect metadata describing the runtime behavior of the code
- Bail: stop executing Ion-compiled code



The way of V8


JavaScript Source
Ignition
Bytecode
Full-codegen
Unoptimized Code
Crankshaft
TurboFan
Optimized Code
Interpreted
Baseline
Optimized

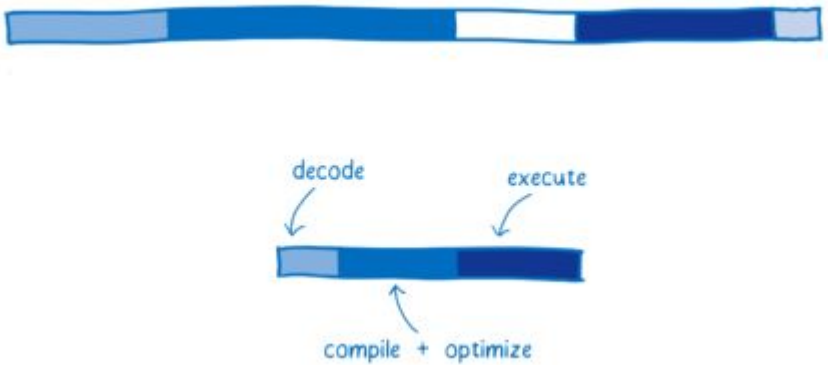
Browser support



WebAssembly

WebAssembly

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



Sebastian Markbåge
@sebmarkbage

Following

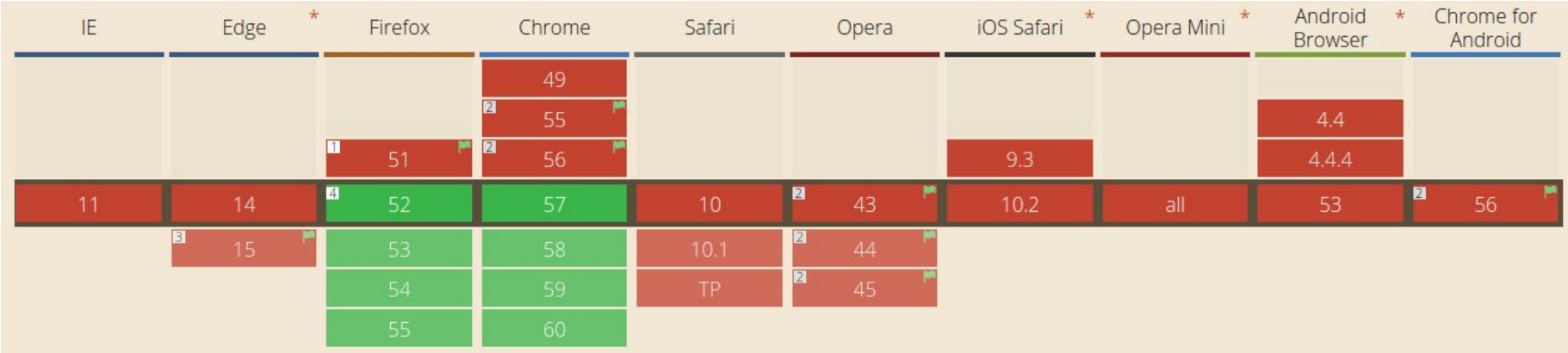
We compiled C++ to asm.js and shipped games with slow compile times. The reaction was Web Assembly, not "ship smaller games".

RETWEETS 4 LIKES 42

6:13 AM - 1 Feb 2017

WebAssebly shipped in Firefox and Chrome

WebAssembly CG members representing four browsers, Chrome, Edge, Firefox, and WebKit, have reached consensus that the design of the initial (MVP) WebAssembly API and binary format is complete to the extent that no further design work is possible without implementation experience and significant usage.



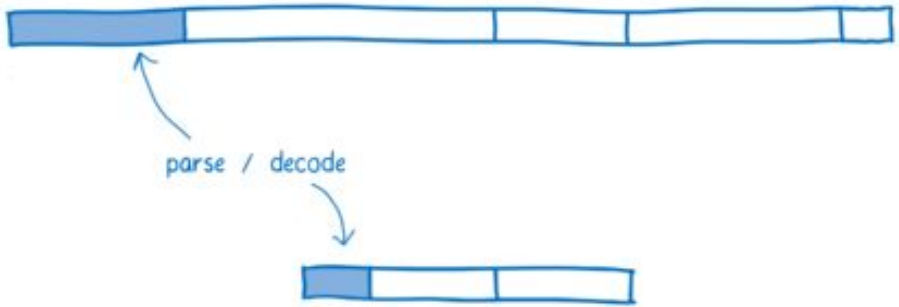
What is WebAssembly and What is not

- A compilation target for native
  - C/C++, other languages -> WASM
- A new capability for the web
  - More than just compressed asm.js
  - float32, int64, threads*, SIMD*
- A complement to JavaScript
  - interface to/from JS code
  - integrate with WebAPIs
- Performance guarantee
  - Fast calling conventions
  - no boxing, no GC
  - AOT

- A value judgment about languages
  - JavaScript vs C++ vs Java vs Dart
- The backend of some C compiler
  - LLVM bitcode, gcc GIMPLE, sea of nodes
- A programming language
  - generated and manipulated by tools
- A separate VM within Chrome
  - instead: built on TurboFan and V8

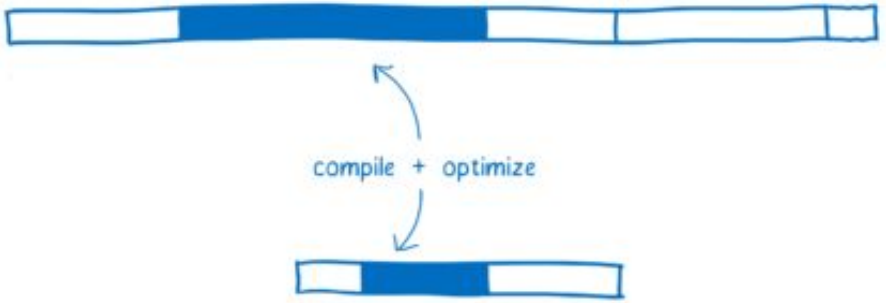
Parsing time

1. Once it reaches the browser, JavaScript source gets parsed into an AST.
2. Browsers often do this lazily.
3. From there, the AST is converted to an intermediate representation (called bytecode) that is specific to that JS engine.
4. **WebAssembly doesn't need to go through this transformation** because it is already an intermediate representation. It just needs to be decoded and validated to make sure there aren't any errors in it.



Compiling + optimizing

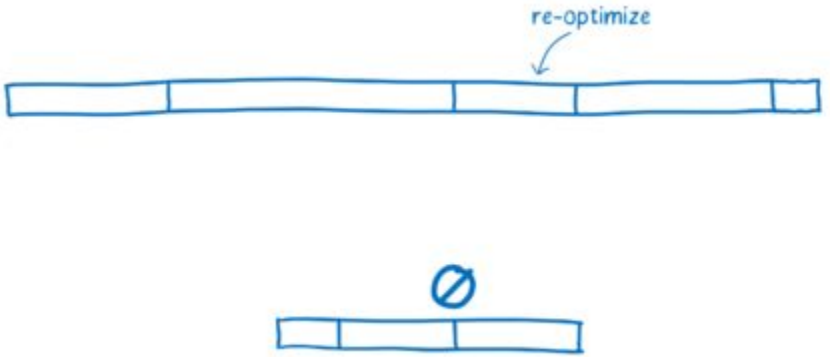
1. The compiler doesn’t have to spend time running the code to observe what types are being used before it starts compiling optimized code.
2. The compiler doesn’t have to compile different versions of the same code based on those different types it observes.
3. More optimizations have already been done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



Reoptimizing

This happens when assumptions that the JIT makes based on running code turn out to be incorrect.

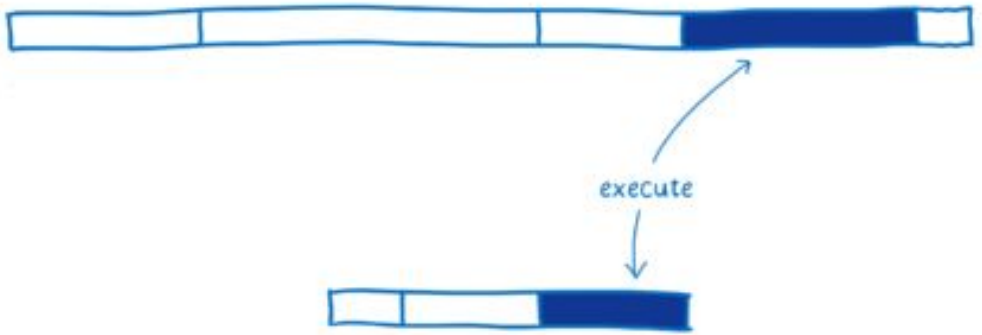
In WebAssembly, things like types are explicit, so the JIT doesn't need to make assumptions about types based on data it gathers during runtime.



Executing

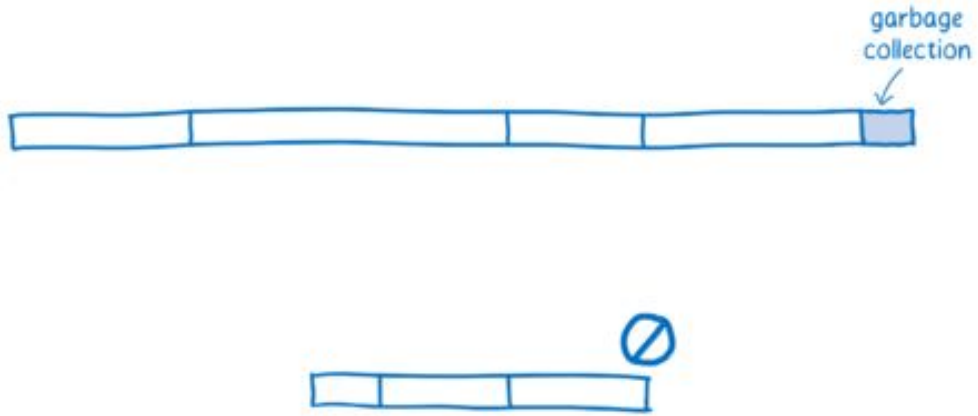
WebAssembly compiles with full optimization.

Optimizers can generate more efficient code based on standard of WebAssembly



Garbage Collection

At least for now, WebAssembly does not support garbage collection at all. Memory is managed manually (as it is in languages like C and C++). While this can make programming more difficult for the developer, it does also make performance more consistent.



WebAssembly WAST

```
int add(int a, int b) {
  return a + b;
}
```

```
(module
  (table 0 anyfunc)
  (memory $0 1)
  (export "memory" (memory $0))
  (export "_Z3addii" (func $_Z3addii))
  (func $_Z3addii (param $0 i32) (param $1 i32) (result i32)
    (i32.add
      (get_local $1)
      (get_local $0)
    )
  )
)
```

Binary format

```
0000000: 0061 736d                 ; WASM_BINARY_MAGIC
0000004: 0d00 0000                 ; WASM_BINARY_VERSION
; section "TYPE" (1)
0000008: 01                        ; section code
0000009: 00                        ; section size (guess)
000000a: 01                        ; num types
; type 0
000000b: 60                        ; func
000000c: 02                        ; num params
000000d: 7f                        ; i32
000000e: 7f                        ; i32
000000f: 01                        ; num results
0000010: 7f                        ; i32
0000009: 07                        ; FIXUP section size
; section "FUNCTION" (3)
0000011: 03                        ; section code
0000012: 00                        ; section size (guess)
0000013: 01                        ; num functions
0000014: 00                        ; function 0 signature index
0000012: 02                        ; FIXUP section size
```

...

```
; section "EXPORT" (7)
0000015: 07                                ; section code
0000016: 00                                ; section size (guess)
0000017: 01                                ; num exports
0000018: 06                                ; string length
0000019: 6164 6454 776f                    ; export name
000001f: 00                                ; export kind
0000020: 00                                ; export func index
0000016: 0a                                ; FIXUP section size
; section "CODE" (10)
0000021: 0a                                ; section code
0000022: 00                                ; section size (guess)
0000023: 01                                ; num functions
; function body 0
0000024: 00                                ; func body size (guess)
0000025: 00                                ; local decl count
0000026: 20                                ; get_local
0000027: 00                                ; local index
0000028: 20                                ; get_local
0000029: 01                                ; local index
000002a: 6a                                ; i32.add
000002b: 0b                                ; end
0000024: 07                                ; FIXUP func body size
0000022: 09                                ; FIXUP section size
```

WebAssembly fetch & execute

```
const imports = {
  global: { Math: window.Math },
  env: {}
};

(async() => {
  try {
    const response = await fetch('add.wasm');
    const bytes = await response.arrayBuffer();
    const results = await WebAssembly.instantiate(bytes, imports);
    console.log(results.instance.exports.add(3, 5));
  } catch (e) {
    console.log(`Error: ${e}`)
  }
})();
```

https://github.com/mdn/webassembly-examples/blob/master/wasm-utils.js

WASM explorer

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

Other languages to WebAssebly

https://github.com/sebmarkbage/ocamlrun-wasm

https://github.com/brson/mir2wasm

Questions time!

1. Ben L. Titzer
   https://goo.gl/VFUObr
2. Dmitry Lomov
   https://goo.gl/n7QF18
3. Links Gist
   https://goo.gl/3yUtw3