# Bringing the Web Up to Speed with Webassembly

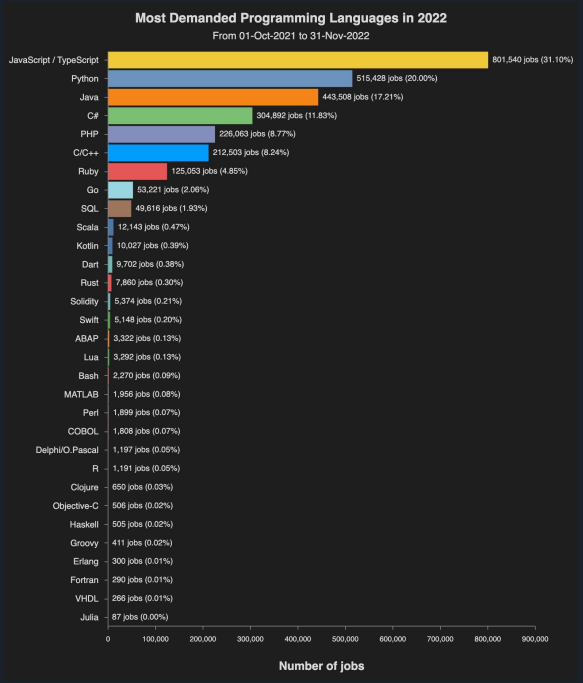
Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, JF Bastien

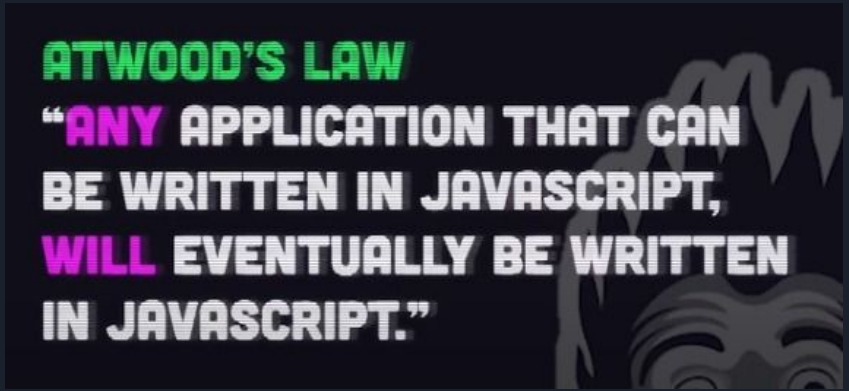
## Group 13

Matt Martin, Luke Hobeika, Jason Qian

# Motivation

# Motivation


Most Demanded Programming Languages in 2022
From 01-Oct-2021 to 31-Nov-2022
JavaScript / TypeScript 801,540 jobs (31.10%)
Python 515,428 jobs (20.00%)
Java 443,508 jobs (17.21%)
C# 304,892 jobs (11.83%)
PHP 226,063 jobs (8.77%)
C/C++ 212,503 jobs (8.24%)
Ruby 125,053 jobs (4.85%)
Go 53,221 jobs (2.06%)
SQL 49,616 jobs (1.93%)
Scala 12,143 jobs (0.47%)
Kotlin 10,027 jobs (0.39%)
Dart 9,702 jobs (0.38%)
Rust 7,860 jobs (0.30%)
Solidity 5,374 jobs (0.21%)
Swift 5,148 jobs (0.20%)
ABAP 3,322 jobs (0.13%)
Lua 3,292 jobs (0.13%)
Bash 2,270 jobs (0.09%)
MATLAB 1,956 jobs (0.08%)
Perl 1,899 jobs (0.07%)
COBOL 1,808 jobs (0.07%)
Delphi/O.Pascal 1,197 jobs (0.05%)
R 1,191 jobs (0.05%)
Clojure 650 jobs (0.03%)
Objective-C 506 jobs (0.02%)
Haskell 505 jobs (0.02%)
Groovy 411 jobs (0.02%)
Erlang 300 jobs (0.01%)
Fortran 290 jobs (0.01%)
VHDL 266 jobs (0.01%)
Julia 87 jobs (0.00%)
0
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000
900,000
Number of jobs

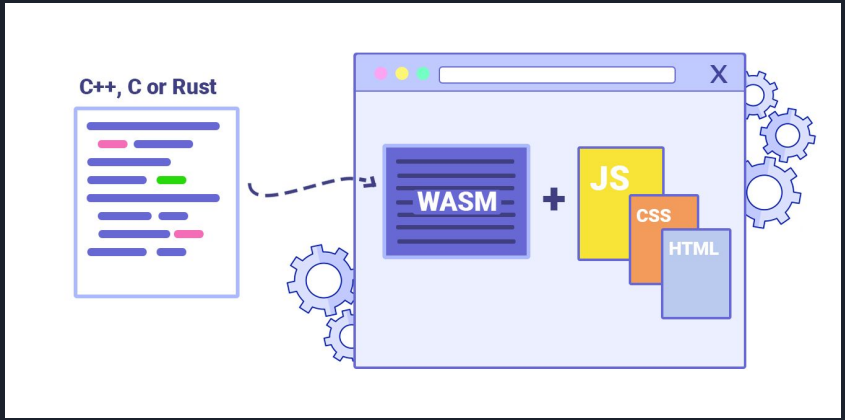

ATWOOD'S LAW
"ANY APPLICATION THAT CAN BE WRITTEN IN JAVASCRIPT, WILL EVENTUALLY BE WRITTEN IN JAVASCRIPT."

# Motivation

- Javascript is interpreted and not compiled
  - Initially the browser ran each line one by one
  - Now it uses a technology called just in time compilation
  - Because javascript is interpreted it is slower than compiled languages such as C++ and Java
    - Overhead from JIT
    - Can apply more powerful optimizations
- Developers have to learn different languages for frontend and backend
  - You can't run c++ in the browser
  - NodeJS tries to fix this issue, but it has limitations

# Webassembly Overview

- WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.
  - You compile other languages into webassembly and then run it in the browser



# Webassembly Overview

- Utilizes a binary code format
- Linear memory
  - The main storage of a web assembly program is a large array of bytes
  - Application developers can grow memory as needed
- Structured control flow
  - No gotos
  - Oly If then blocks, and loops
-

# Execution

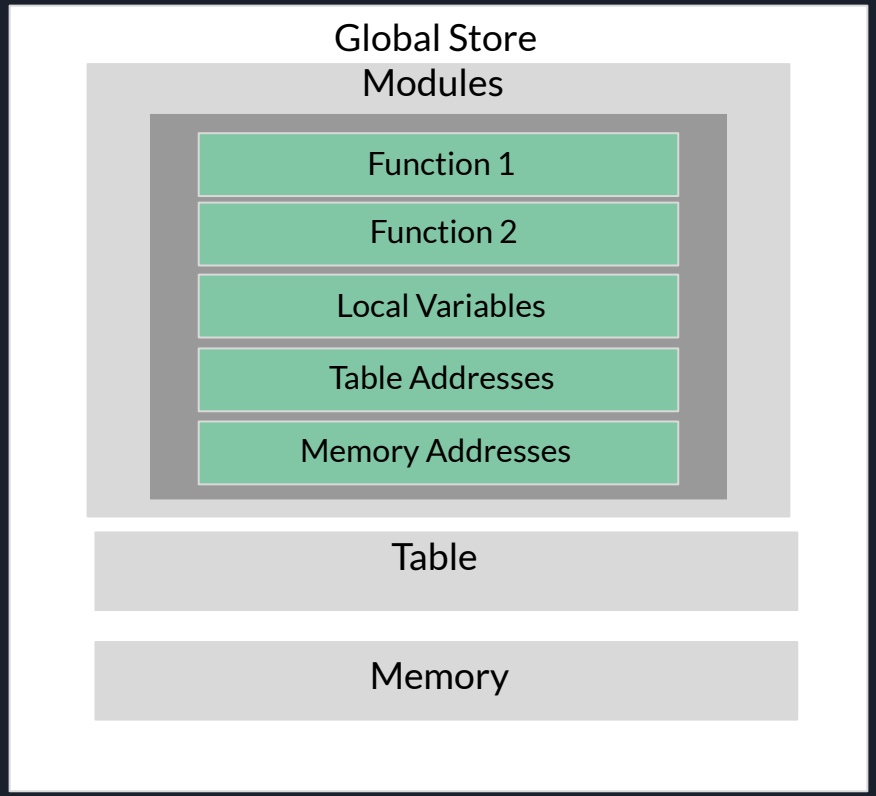
# Stores and Instances

- A store is a record of the lists of module instances, tables and memories that have been allocated in it
- Tables and memories reside in the global store and are only referenced by address, since they can be shared between multiple instances.
- Globals are represented by the values they hold and reside in their defining instance

$$
\begin{array}{lcll}
\mathit{store} & ::= & \{\, \mathsf{funcs} & \mathit{funcinst}^*, \\
 & & \mathsf{tables} & \mathit{tableinst}^*, \\
 & & \mathsf{mems} & \mathit{meminst}^*, \\
 & & \mathsf{globals} & \mathit{globalinst}^*, \\
 & & \mathsf{elems} & \mathit{eleminst}^*, \\
 & & \mathsf{datas} & \mathit{datainst}^* \,\}
\end{array}
$$

https://webassembly.github.io/spec/core/exec/runtime.html

# Webassembly Hierarchy Representation



# Reduction

- Reduction: How an expression can be evaluated to its final value
- Webassembly Reduction is defined over its configuration (global store, local variables, and instruction sequence)
- Reduction is Stack-Based

# Reduction Example

```
6   (module
7       (import "math callback" (func $callback))
8
9       (export "add" (func $add))
10      (export "subtract" (func $subtract))
11
12      (func $add (param $a i32) (param $b i32) (result i32)
13          local.get $a
14          local.get $b
15          i32.add
16      )
17  )
```

Stack

a = 5

# Reduction Example - Continued

```
6   (module
7       (import "math callback" (func $callback))
8
9       (export "add" (func $add))
10      (export "subtract" (func $subtract))
11
12      (func $add (param $a i32) (param $b i32) (result i32)
13          local.get $a
14          local.get $b
15          i32.add
16      )
17  )
```

Stack

| |
|---|
| a = 5 |
| b = 10 |

# Reduction Example - Continued

```
6   (module
7       (import "math callback" (func $callback))
8
9       (export "add" (func $add))
10      (export "subtract" (func $subtract))
11
12      (func $add (param $a i32) (param $b i32) (result i32)
13          local.get $a
14          local.get $b
15          i32.add
16      )
17  )
```

Stack

result = 15

# Validation

# Validation

- Safety is VERY important on the web
- Running much untrusted code
- Need to quickly check code is safe
- Typing guards against dangerous branching and stack corruption
- Code Validation focuses on type correctness
- Can be checked very quickly in a single pass of the code

# Type Safety Considerations

- All functions and instructions specify the state of the stack before and after they are called
- Instructions within a block cannot access any values pushed onto the stack outside the block
- Blocks have to clear values off the stack before branching unless a return value is specified
- Branches can only branch to enclosing blocks, they cannot branch to arbitrary lines
- Branch instructions require operands on stack to match join points
- Input stack of instruction must match output stack of preceding instruction

```
(module
    (import "env" "log" (func $log (param i32)))
    (func $main
        block $UNLESS_BLOCK
            block $THEN
                block $UNLESS
                    i32.const 0 ;; unless false
                    br_if $THEN
                end
                ;; executed unless false
                i32.const 10
                call $log
                br $UNLESS_BLOCK
            end
            ;; executed unless true
            i32.const 20
            call $log
        end
    )
    (start $main)
)
```

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

log_if_not_100 ->

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

i32

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

i32

my_block ->

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

i32

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

| |
|---|
| i32 |
| i32 |

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

i32.eq ->

Stack

| |
|---|
| i32 |
| i32 |

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

if ->

Stack

bool

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

br ->

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

i32

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

i32

log ->

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

exit_block ->

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

exit_func ->

# Example Validation Pass

```
(module
  ;; import the browser console object, you'll need to pass
  (import "console" "log" (func $log (param i32)))

  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block

      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq

      (if
        (then

          ;; branch to the end of the block
          br $my_block

        )
      )

      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

Stack

# Implementation and Measurements

# Embedding and Interoperability

- Webassembly does not define how modules are loaded into the execution engine or how they perform I/O. Instead, a Webassembly implementation is embedded into an execution environment.
  - Javascript API
- Webassembly can link together different instances and interoperate different applications *

# Implementation

- Webassembly Design Goal: High performance without sacrificing safety or portability
  - Fast Validation of Code
  - Optimized JIT Compiler

# Measurements

- Webassembly is very competitive relative to native code
- Webassembly is also significantly faster than asm.js
  - 33.7% faster on average

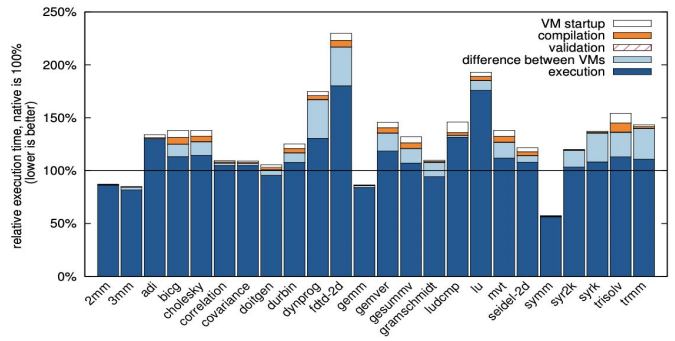



**Figure 5.** Relative execution time of the PolyBenchC benchmarks on WebAssembly normalized to native code

- Webassembly code size is on average 62.5% of asm.js code size and 85.3% of native code size

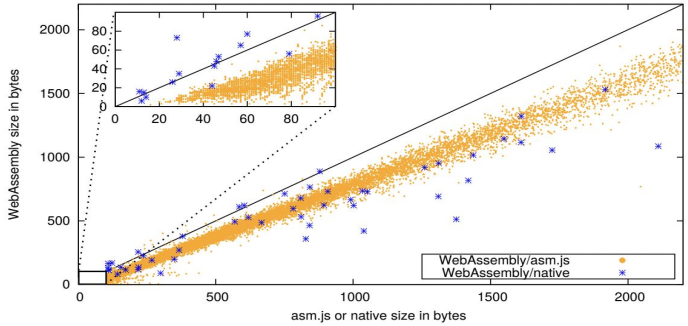



**Figure 6.** Binary size of WebAssembly in comparison to asm.js and native code

# Future Work

# Future Work at the Time of Paper

- Exceptions
- Threads
- SIMD instructions
- Garbage Collection

# Current State 6 Years Later

- Exceptions ✅
- Threads ✅
- SIMD instructions ✅
- Garbage Collection ✅

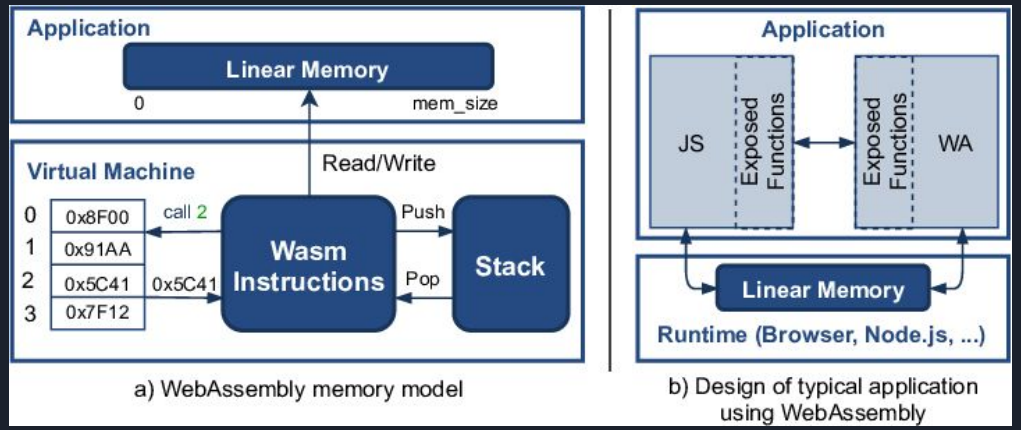
| | Your browser | Chrome | Firefox | Safari | Wasmtime | Wasmer | Node.js | Deno | wasm2c |
|---|---|---|---|---|---|---|---|---|---|
| **Standardized features** | | | | | | | | | |
| JS BigInt to Wasm i64 integration | ✓ | 85 | 78 | 14.1 [d] | N/A | N/A | 15.0 | 1.1.2 | N/A |
| Bulk memory operations | ✓ | 75 | 79 | 15 | 0.20 | 1.0 | 12.5 | 0.4 | 1.0.30 |
| Extended constant expressions | × | ⚑ [a] | ⚑ [c] | × | × | × | ⚑ [g] | ⚑ [l] | × |
| Multi-value | ✓ | 85 | 78 | ✓ | 0.17 | 1.0 | 15.0 | 1.3.2 | 1.0.24 |
| Mutable globals | ✓ | 74 | 61 | ✓ | ✓ | 0.7 | 12.0 | 0.1 | 1.0.1 |
| Reference types | ✓ | 96 | 79 | 15 | 0.20 | 2.0 | 17.2 | 1.16 | 1.0.31 |
| Non-trapping float-to-int conversions | ✓ | 75 | 64 | 15 | ✓ | ✓ | 12.5 | 0.4 | 1.0.24 |
| Sign-extension operations | ✓ | 74 | 62 | 14.1 [d] | ✓ | ✓ | 12.0 | 0.1 | 1.0.24 |
| Fixed-width SIMD | ✓ | 91 | 89 | 16.4 | 0.33 | 2.0 | 16.4 | 1.9 | × |
| Tail calls | × | 112 | × | × | × | × | ⚑ [j] | ⚑ [o] | × |
| **In-progress proposals** | | | | | | | | | |
| Exception handling | ✓ | 95 | 100 | 15.2 | × | × | 17.0 | 1.16 | ⚑ [q] |
| Garbage collection | × | ⚑ [b] | × | × | × | × | × | × | × |
| Memory64 | × | ⚑ [a] | ⚑ [c] | × | ⚑ [e] | × | ⚑ [h] | ⚑ [m] | ⚑ [r] |
| Multiple memories | ? | × | × | × | ⚑ [f] | × | × | × | ⚑ [s] |
| Relaxed SIMD | × | ⚑ [a] | ⚑ [c] | × | × | × | ⚑ [i] | ⚑ [n] | × |
| Threads and atomics | ✓ | 74 | 79 | 14.1 [d] | N/A | N/A | 16.4 | 1.9 | × |
| Type reflection | ? | ⚑ [a] | ⚑ [c] | × | × | 2.0 | ⚑ [k] | ⚑ [p] | × |

Source: https://webassembly.org/roadmap/

# Appendix

# Might be helpful

https://www.researchgate.net/figure/WebAssembly-high-level-architecture_fig1_360232889



# Threads and Atomics

Web workers are multi-process

- only way to get parallel execution in web development

Can create wasm modules that take in memory address as a parameter so now have two modules sharing a memory buffer. Commonly written in rust.

The module is the code, the instance is the process in webassembly

Exception will get thrown if the spawned web assembly ends up blocking the main code

shared array buffer has been implemented

threads still use webworkers, but shared array buffers make message passing significantly faster than provided web worker API

# References

[1] https://www.devjobsscanner.com/blog/top-8-most-demanded-languages-in-2022/

[2]https://people.mpi-sws.org/~rossberg/papers/Haas,%20Rossberg,%20Schuff,%20Titzer,%20Gohman,%20Wagner,%20Zakai,%20Bastien,%20Holman%20-%20Bringing%20the%20Web%20up%20to%20Speed%20with%20WebAssembly.pdf