An introduction to

WebAssembly

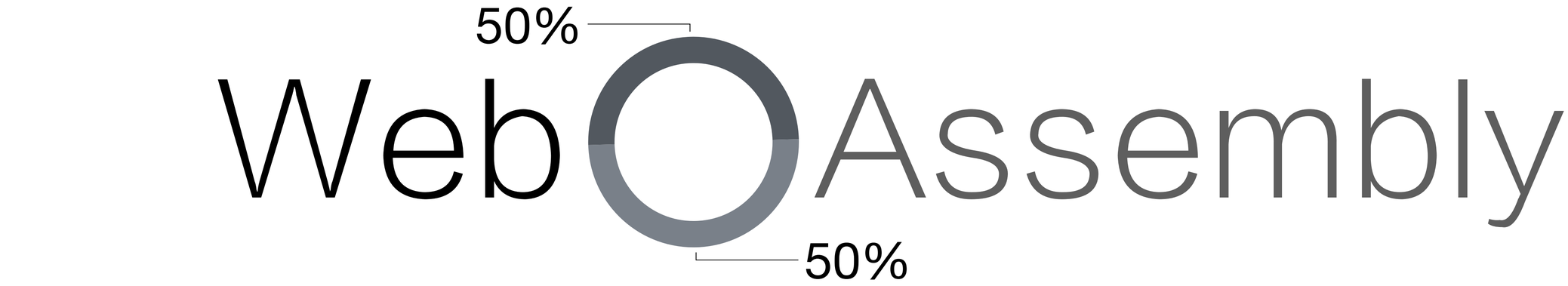
WA

Mohammad Hashemi

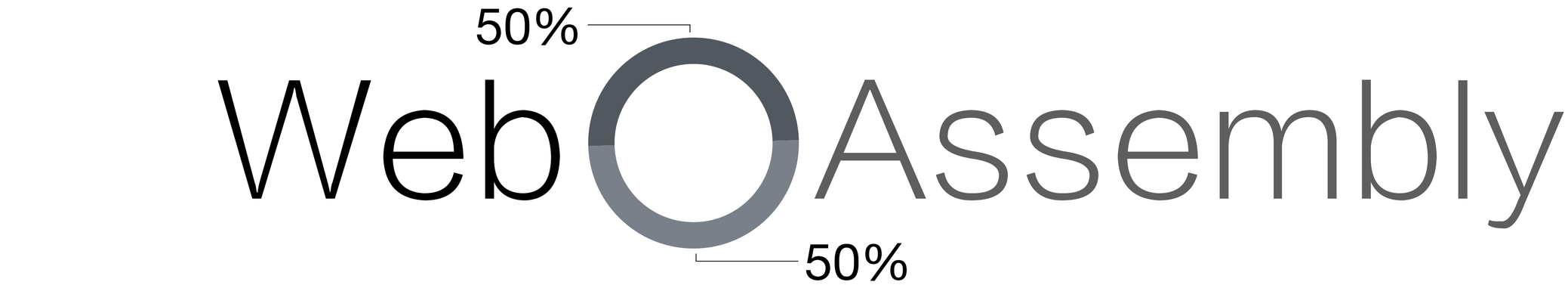
Narges Ghasemi

WebAssembly


50%
Web
Assembly
50%



An **Assembly** language for **Web** browser.



- An **Assembly** language for **Web** browser.
- Its not Assembly, its **bytecode**.

But, its a solution to a problem.

```
NaN === NaN;
// -> false
parseInt("parham");
// -> NaN
parseInt("alvani", 16);
// -> 10
console.log.call.call.call.apply(a => a, [1, 2]);
// -> 2
Math.min() > Math.max();
// -> true
```

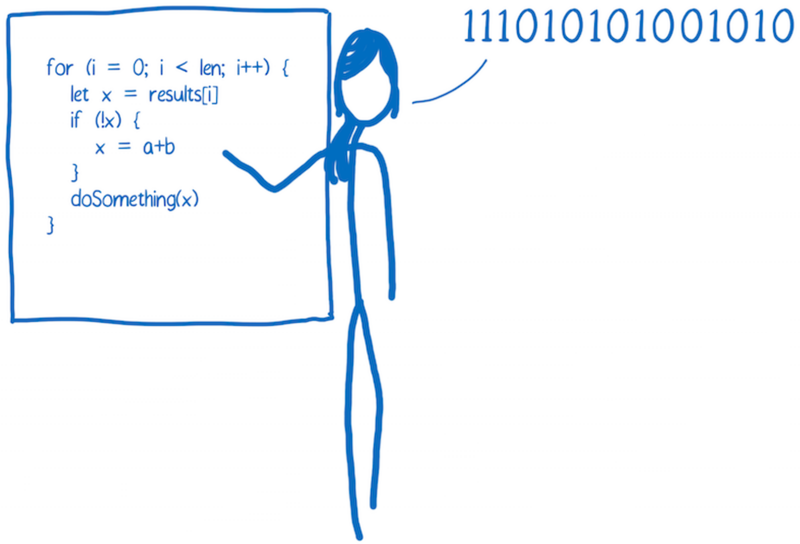
How JavaScript is run in the browser?

```
NaN === NaN;
// -> false
parseInt("parham");
// -> NaN
parseInt("alvani", 16);
// -> 10
console.log.call.call.call.apply(a => a, [1, 2]);
// -> 2
Math.min() > Math.max();
// -> true
```

How JavaScript is run in the browser?

How JavaScript is run in the browser?

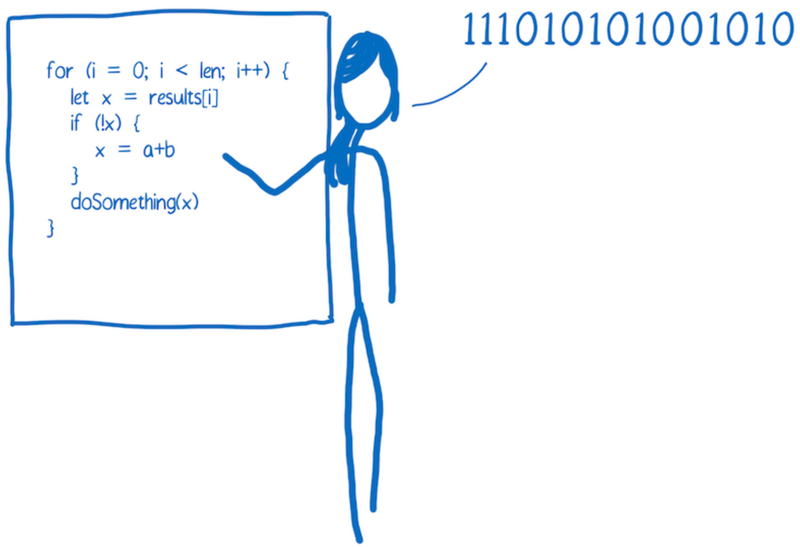
Interpreter



Interpreters — Source: Lin Clark

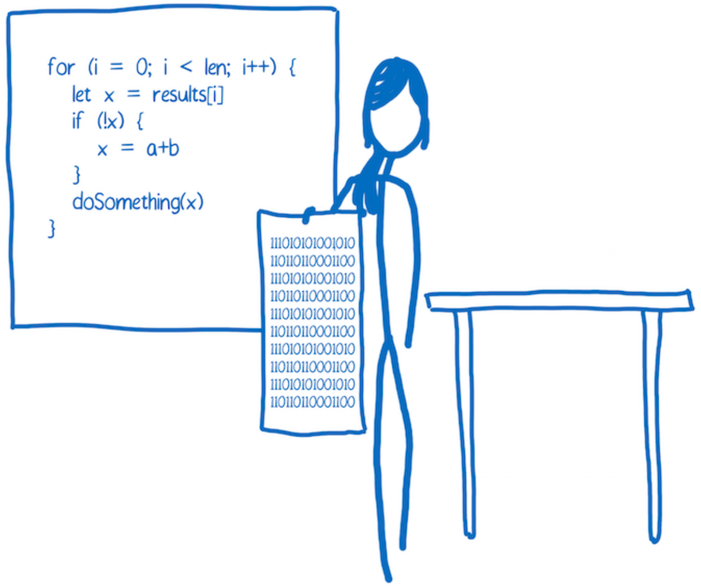
How JavaScript is run in the browser?

Interpreter



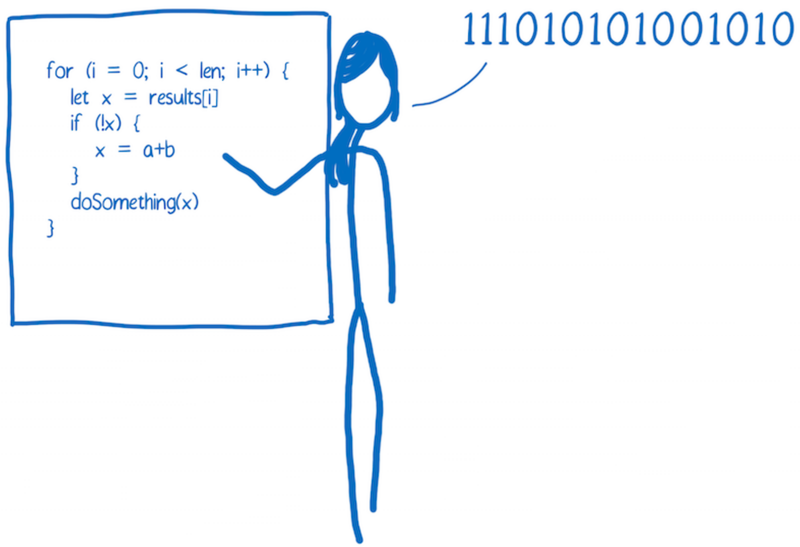
Interpreters — Source: Lin Clark

Compiler



Compilers — Source: Lin Clark

Interpreter



Interpreters — Source: Lin Clark

Pros

You do not have to go through that whole compilation step before you can start running your code.

Cons

You have to do the same translation over and over again.

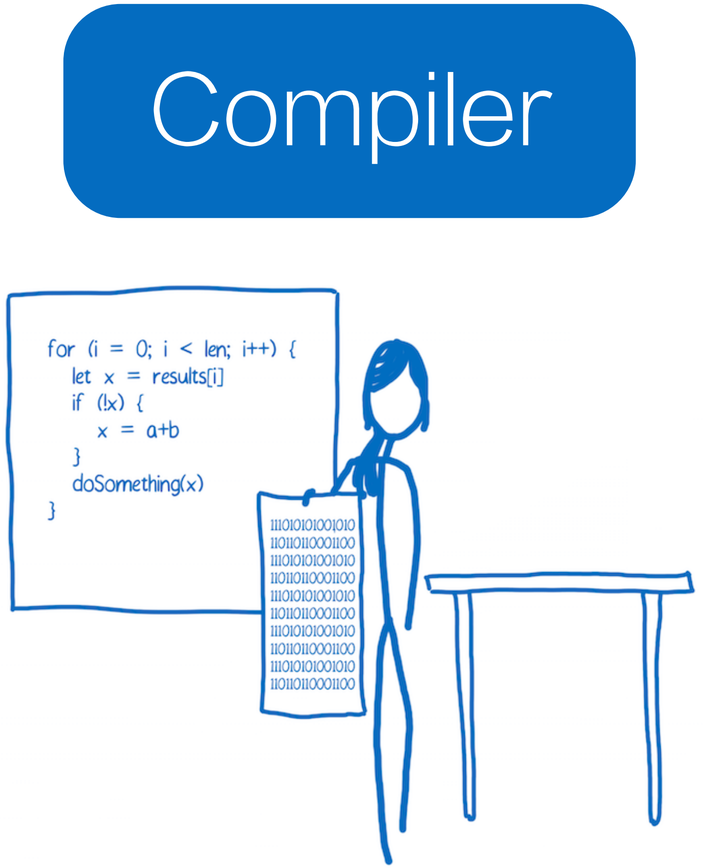
Pros

It does not need to repeat the translation for each pass through the loop. (using Monitor or Profiler)

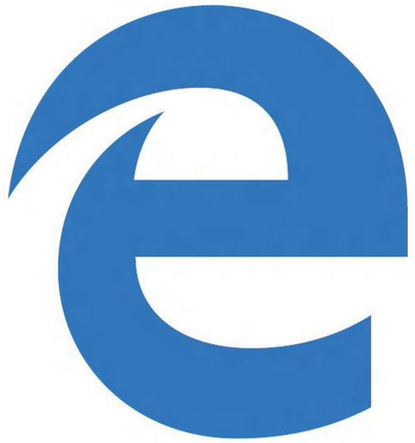
Cons

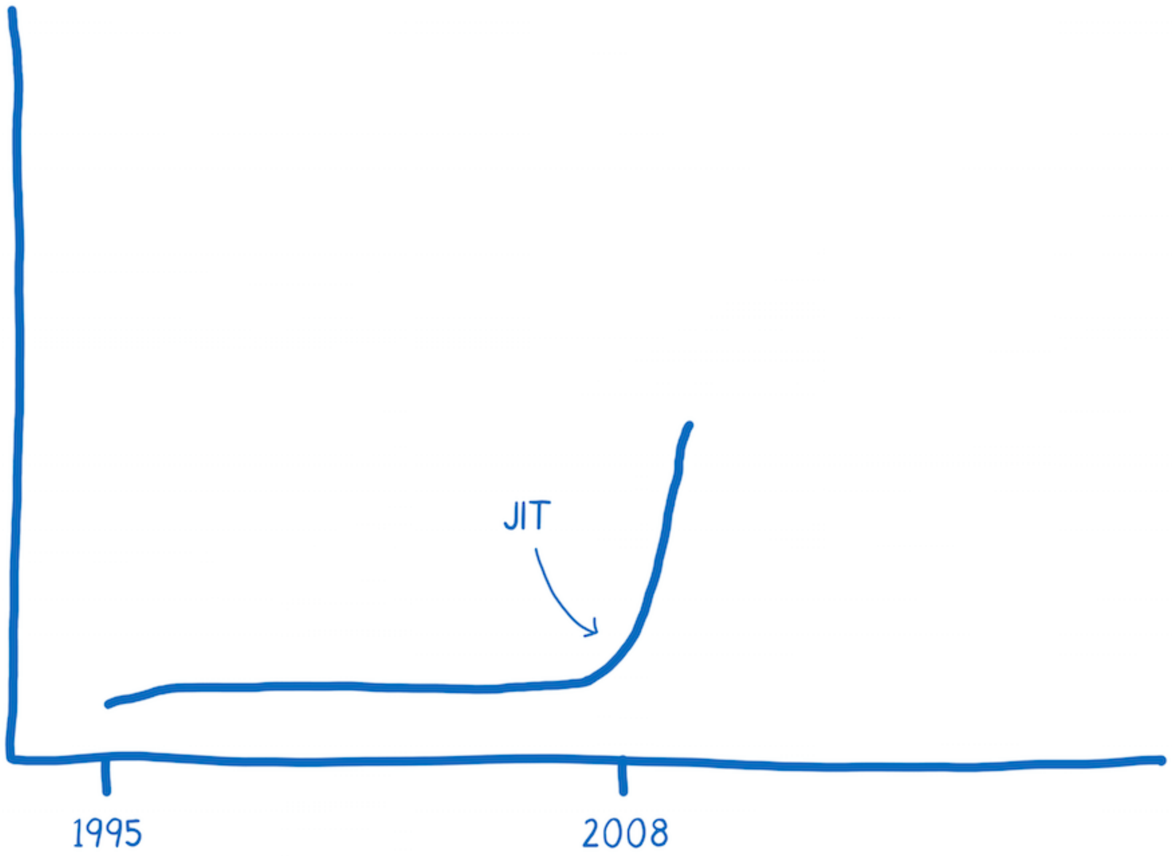
It takes a little bit more time to start up because it has to go through the compilation step at the beginning.

To learn more, read the full article on just-in-time compiling.

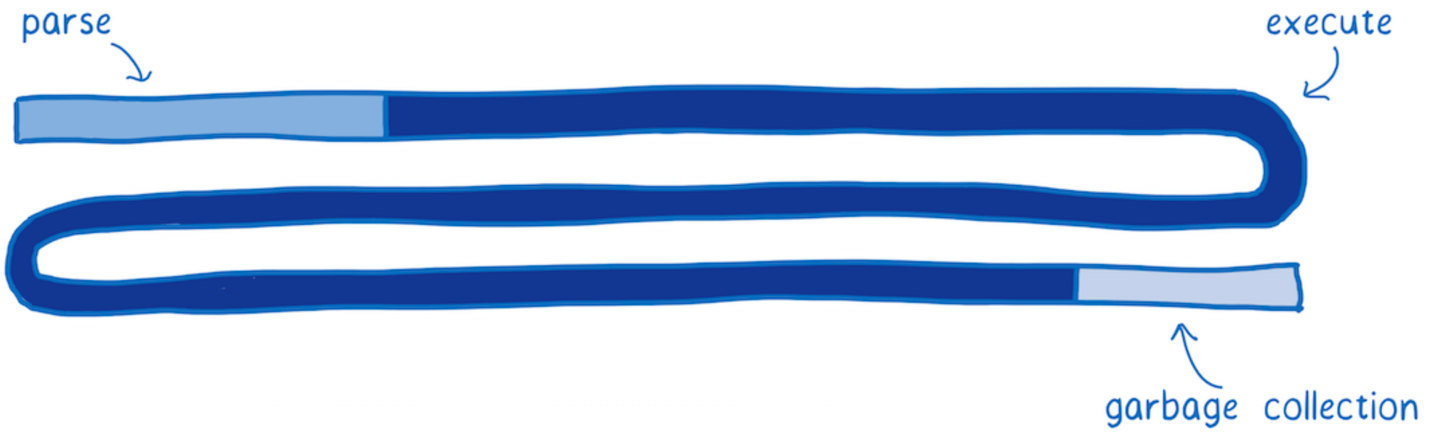


Compilers — Source: Lin Clark

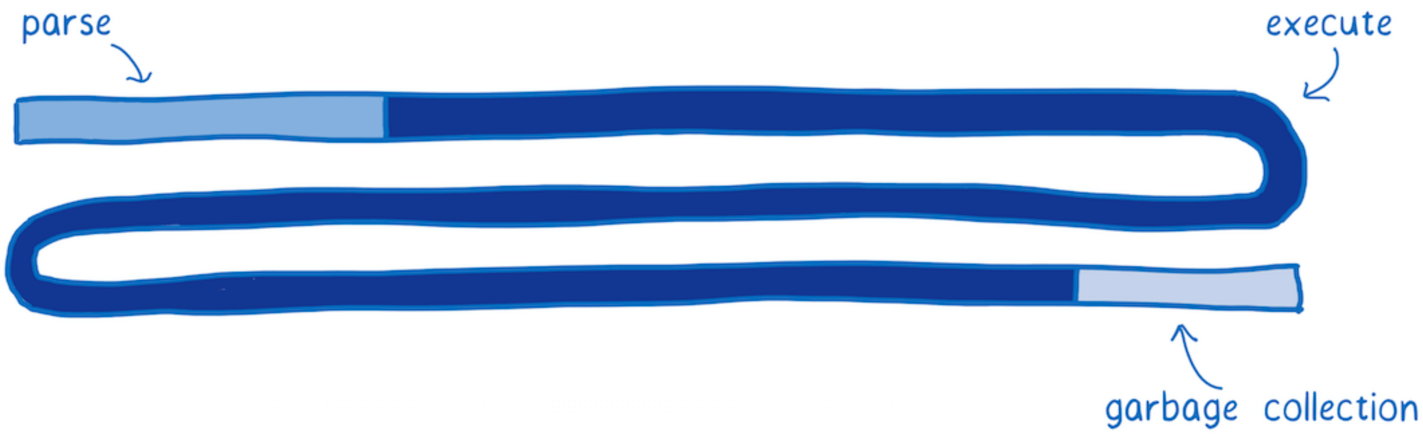




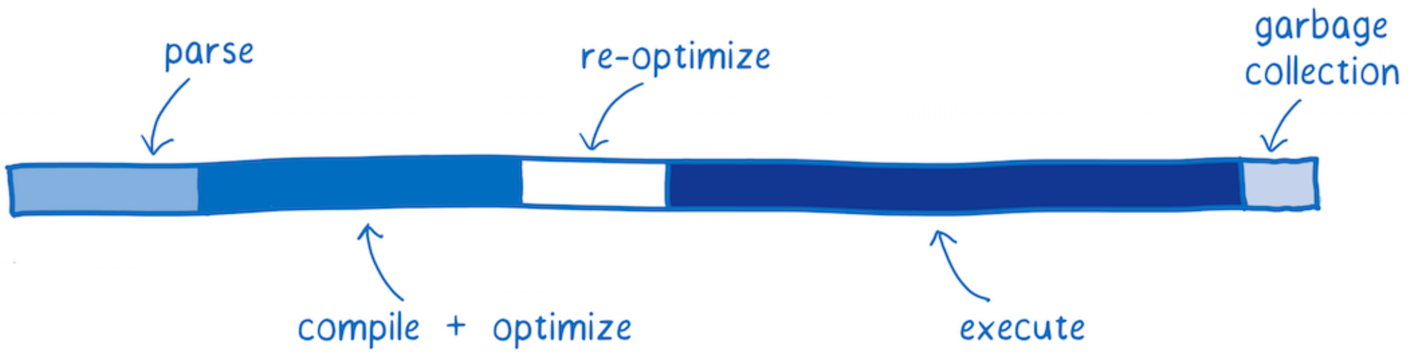
Javascript performance — Source: Lin Clark



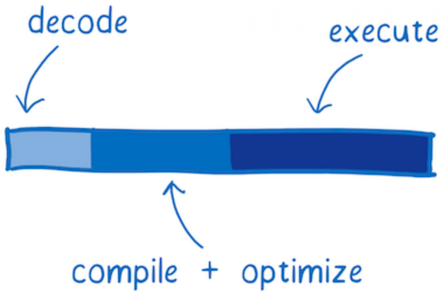
Average time spent by a JS engine using only interpreters — Source: Lin Clark

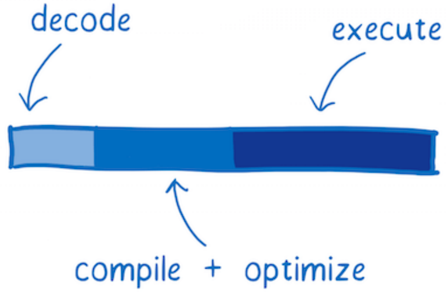


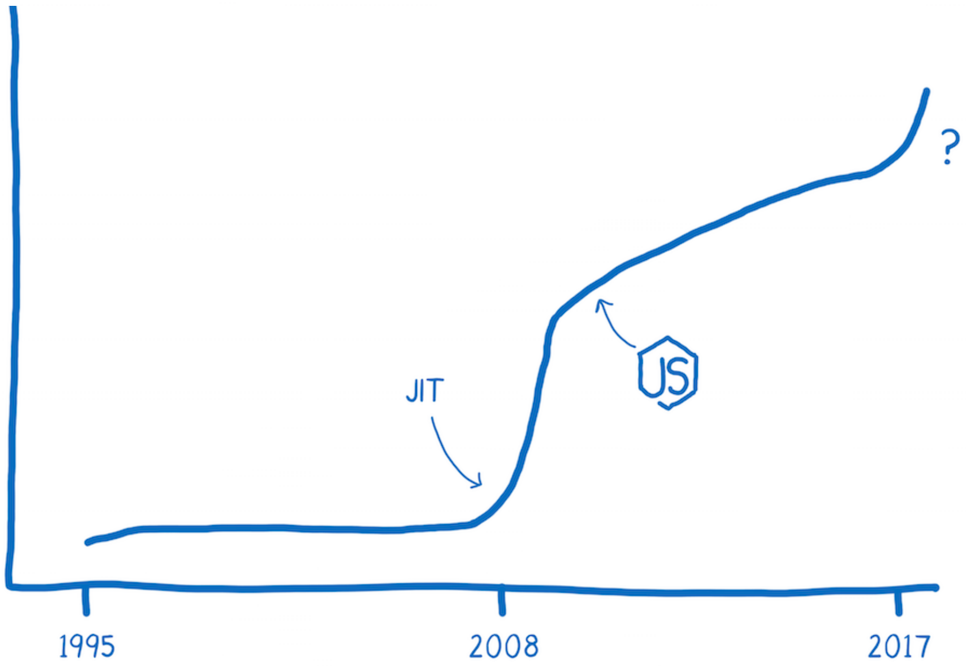
Average time spent by a JS engine using only interpreters — Source: Lin Clark



Average time spent by a JS engine using JIT — Source: Lin Clark


decode
execute
compile + optimize





Javascript performance — Source: Lin Clark

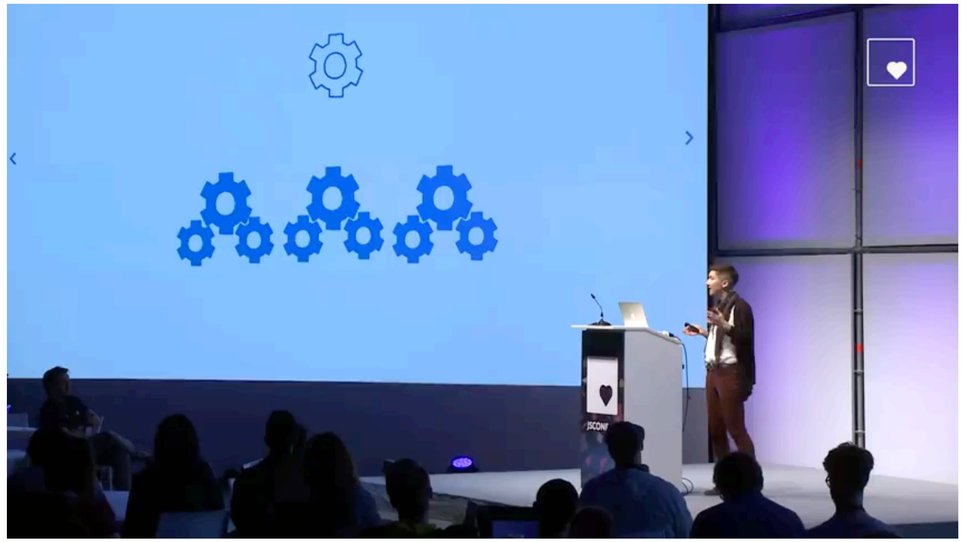
Lin Clark - Engineer on the
Mozilla Developer Relations Team.

Lin Clark - Engineer on the Mozilla Developer Relations Team.

She predicted that the introduction of WebAssembly in 2017 may trigger a new inflexion point in the life of web development.

Lin Clark - Engineer on the Mozilla Developer Relations Team.

She predicted that the introduction of WebAssembly in 2017 may trigger a new inflexion point in the life of web development.

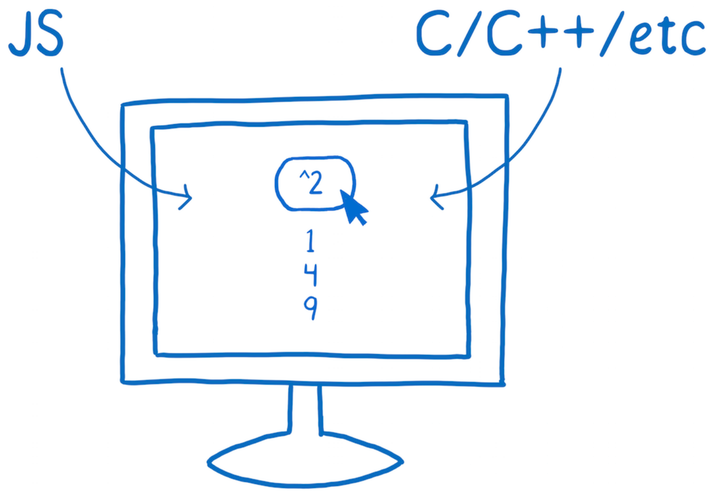


Lin Clark: A Cartoon Intro to WebAssembly [Link]

So, what the WebAssembly is?

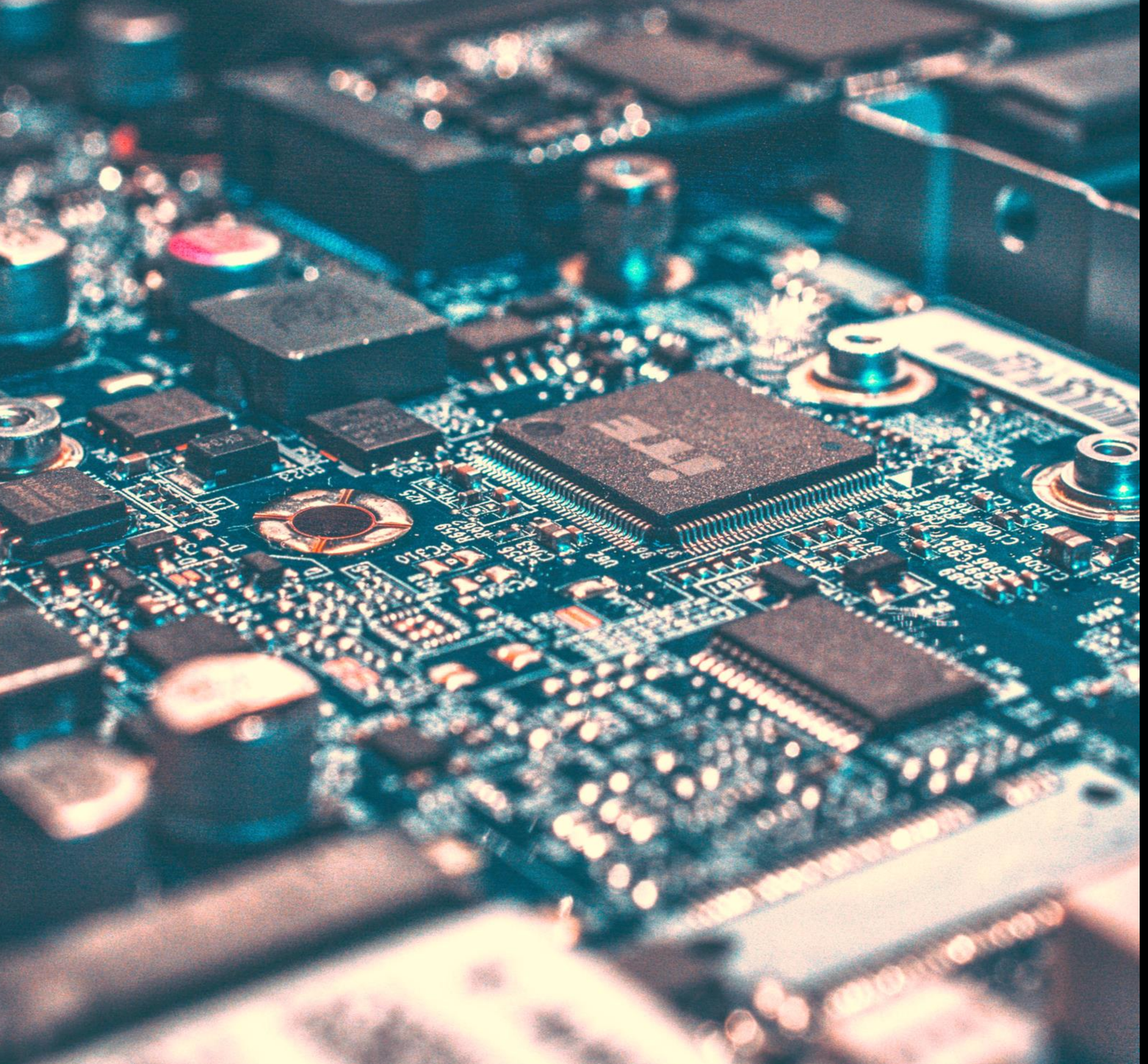

WA

WA



Definition of WA — Source: [LInk]

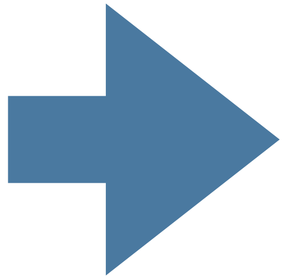
A way of taking code written in programming languages other than JS and running that code in the browser.



10100000 00000000
10100010 00000000
10001010 10000101
11000100 00100000
11011101 11110000
11001000 01110000
00000101 01001100
00000111 00000010
10111000 11101000
01001100 00000100
00000010

Address	Binary	Hex	Assembly
0200	10100000 00000000	A0 00	LDY #$00
0202	10100010 00000000	A2 00	LDX #$00
0204	10001010	8A	LOOP2 TXA
0205	10000101 11000100	85 C4	STA $00C4
0207	00100000 11011101 11110000	20 22 F0	LOOP1 JSR SCAN
020A	11001000	C8	INY
020B	01110000 00000101	70 05	BVS RESET
020D	01001100 00000111 00000010	4C 07 02	JMP LOOP1
0210	10111000	B8	RESET CLV
0211	11101000	E8	INX
0212	01001100 00000100 00000010	4C 04 02	JMP LOOP2

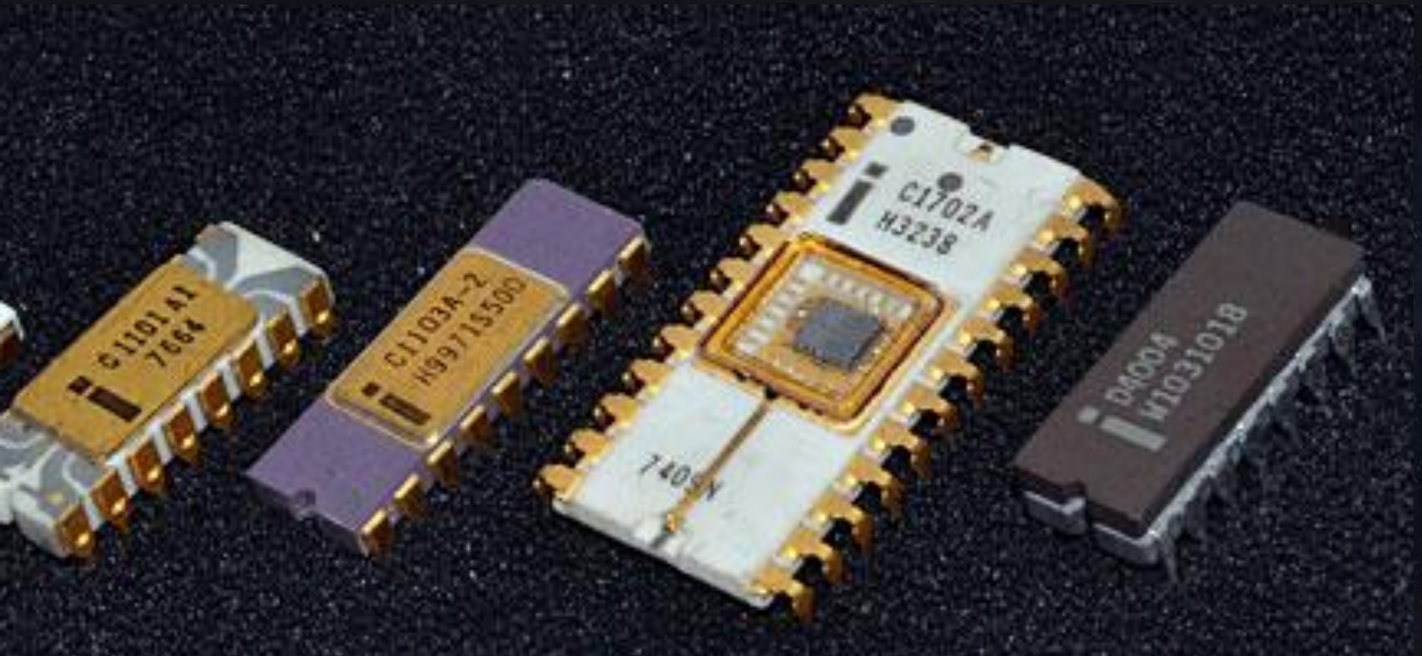
```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```

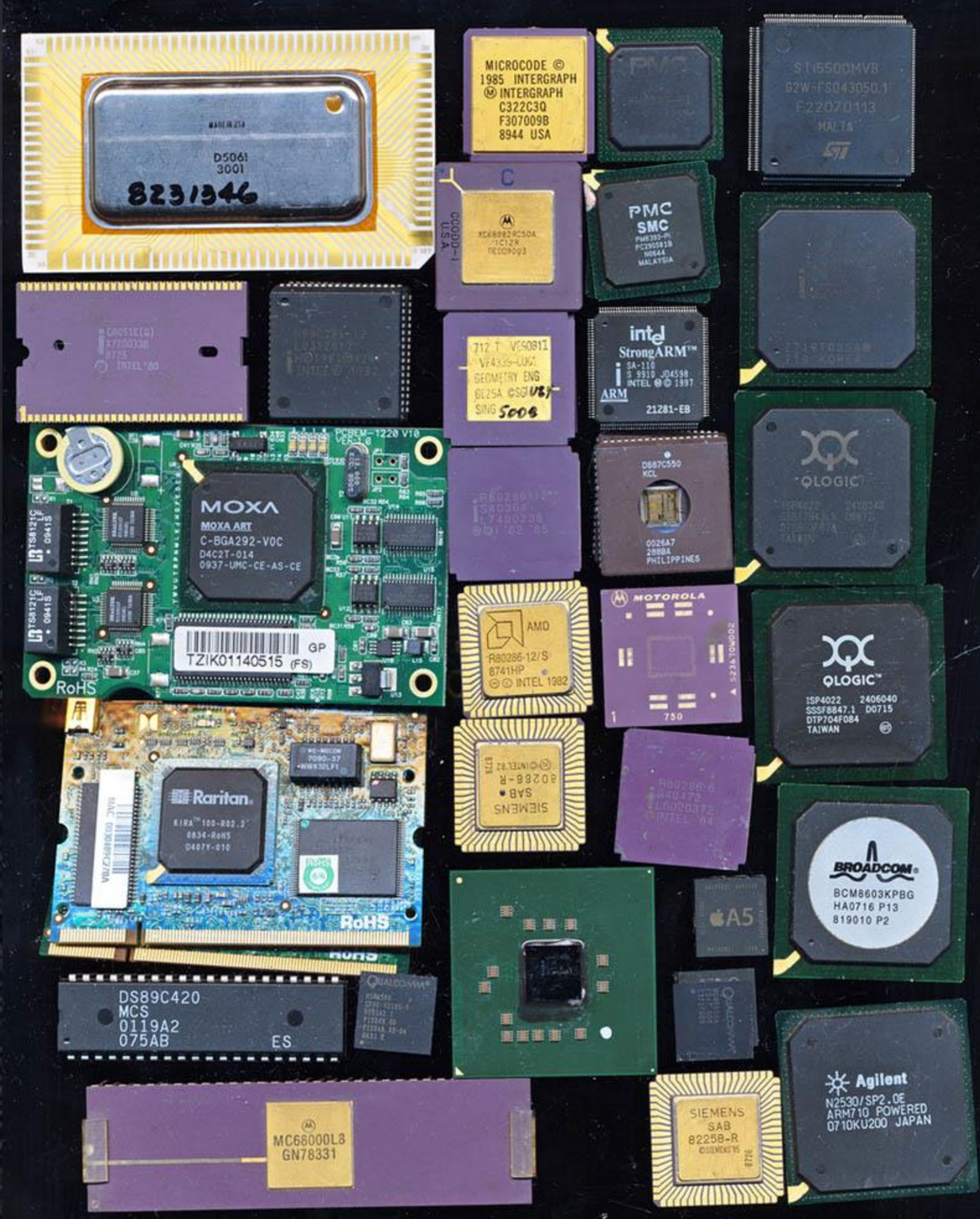


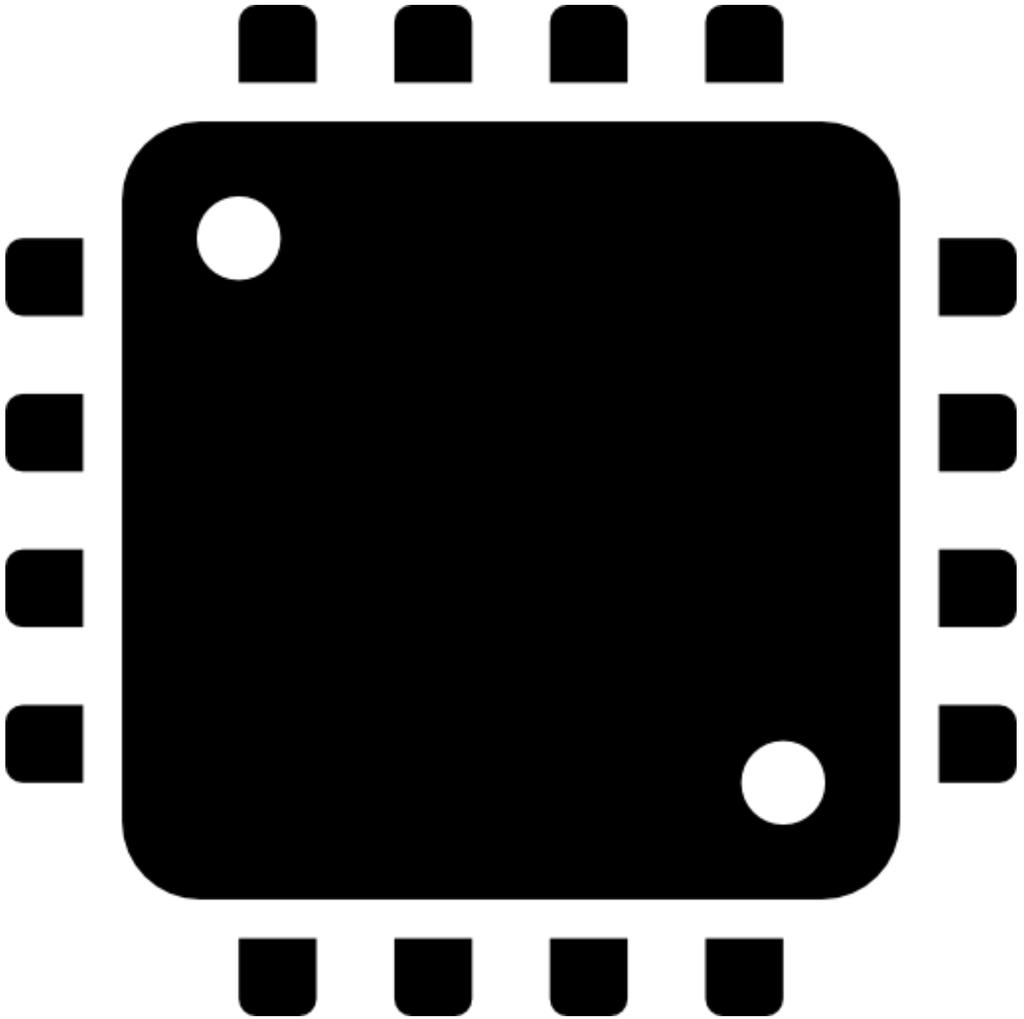
```
10100000 00000000
10100010 00000000
10001010 10000101
11000100 00100000
11011101 11110000
11001000 01110000
00000101 01001100
00000111 00000010
10111000 11101000
01001100 00000100
00000010
```


MOS
6502AD
4585 S

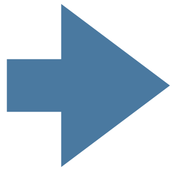

XILINX
HKb2bsite.com


C1101A1
7664
C1103A-2
H9971550
C1702A
H3238
D4004
V1031018

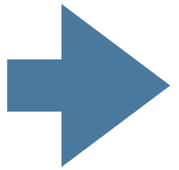

D5061
3001
8231946
MICROCODE ©
1985 INTERGRAPH
INTERGRAPH
C322C3Q
F307009B
8944 USA
MOXA
MOXA ART
C-BGA292-V0C
D4C2T-014
0937-UMC-CE-AS-CE
TZIK01140515 (FS)
RoHS
Raritan
DS89C420
MCS
0119A2
075AB
ES
MC68000L8
GN78331
intel
StrongARM
SA-110
ARM
21281-EB
AMD
R80286-12/S
8741HP
INTEL '1982
MOTOROLA
QLOGIC
ISP4022
SSSF8847.1
DTP704F084
TAIWAN
SIEMENS
SAB
8225B-R
BROADCOM
BCM8603KPBG
HA0716 P13
819010 P2
A5
Agilent
N2530/SP2.0E
ARM710 POWERED
0710KU200 JAPAN

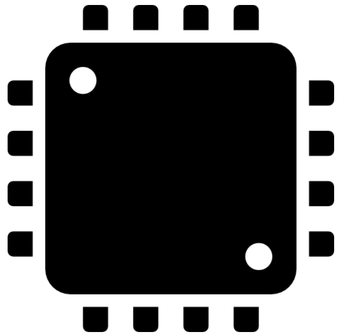


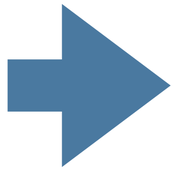
```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```

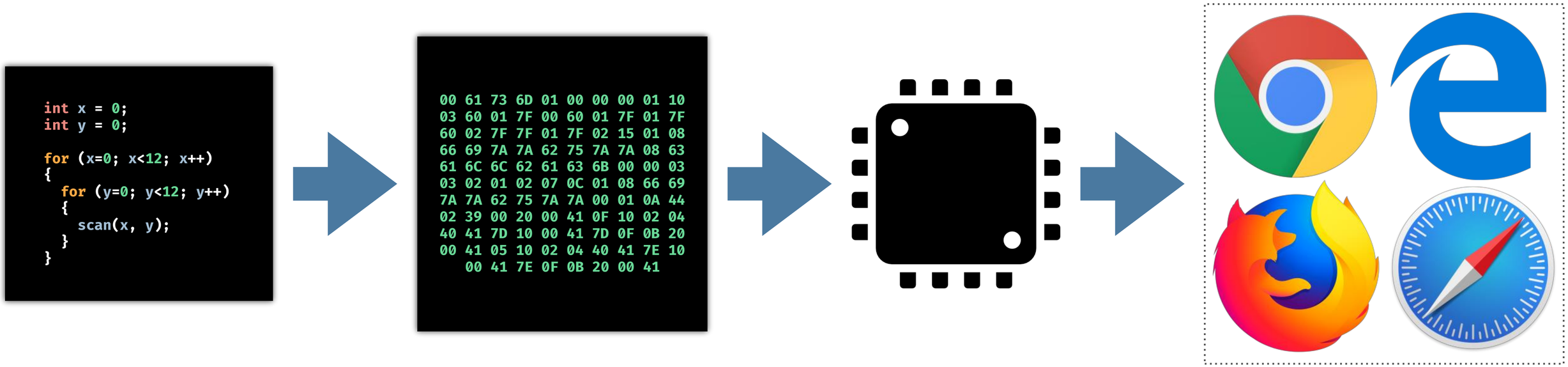


```
CA FE BA BE 00 00 00 31 00 13
07 00 10 07 00 11 01 00 07 63
6F 6E 76 65 72 74 01 00 3A 28
4C 6A 61 76 61 2F 6C 61 6E 67
2F 49 74 65 72 61 62 6C 65 3B
4C 6A 61 76 61 2F 6C 61 6E 67
2F 4F 62 6A 65 63 74 3B 29 4C
6A 61 76 61 2F 6C 61 6E 67 2F
4F 62 6A 65 63 74 3B 01 00 0A
45 78 63 65 70 74 69 6F 6E 73
   07 00 12 01 00 09 53 69
```





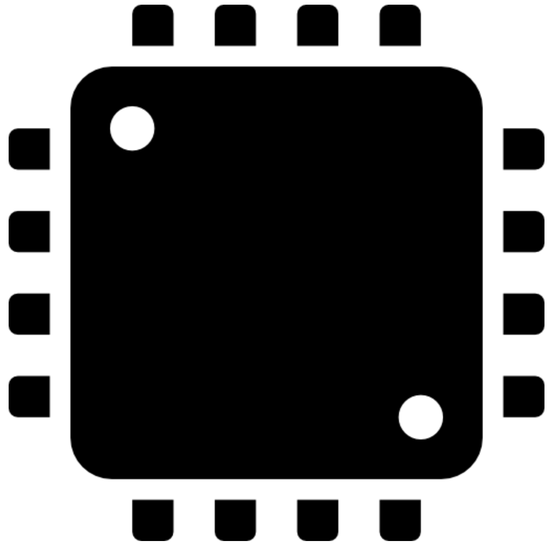



int x = 0;
int y = 0;
for (x=0; x<12; x++)
{
for (y=0; y<12; y++)
{
scan(x, y);
}
}
00 61 73 6D 01 00 00 00 01 10
03 60 01 7F 00 60 01 7F 01 7F
60 02 7F 7F 01 7F 02 15 01 08
66 69 7A 7A 62 75 7A 7A 08 63
61 6C 6C 62 61 63 6B 00 00 03
03 02 01 02 07 0C 01 08 66 69
7A 7A 62 75 7A 7A 00 01 0A 44
02 39 00 20 00 41 0F 10 02 04
40 41 7D 10 00 41 7D 0F 0B 20
00 41 05 10 02 04 40 41 7E 10
00 41 7E 0F 0B 20 00 41

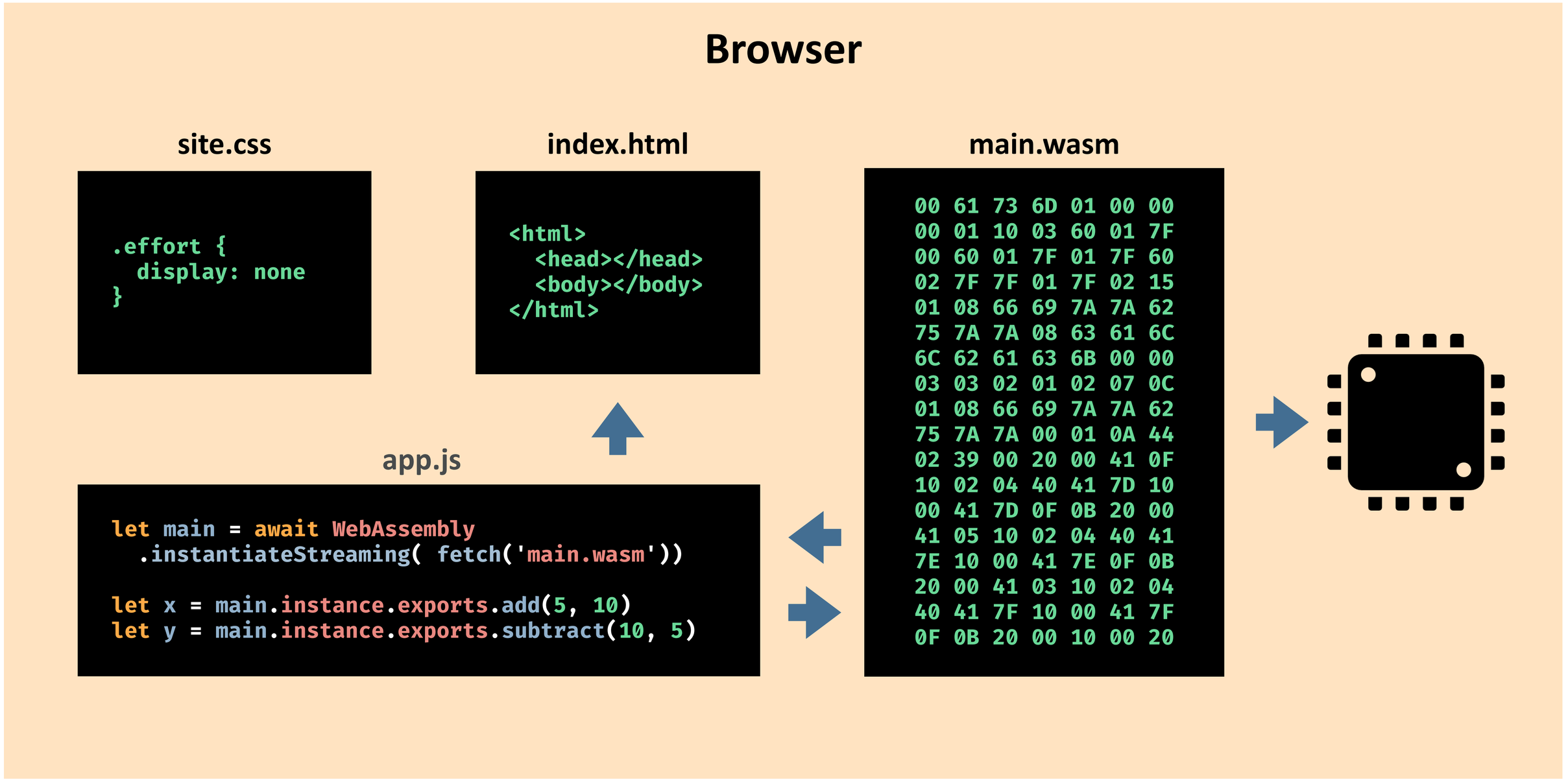
WebAssembly Modules

- .add
- .subtract
- .multiply
- .divide
- .modulo

```
00 61 73 6D 01 00 00
00 01 10 03 60 01 7F
00 60 01 7F 01 7F 60
02 7F 7F 01 7F 02 15
01 08 66 69 7A 7A 62
75 7A 7A 08 63 61 6C
6C 62 61 63 6B 00 00
03 03 02 01 02 07 0C
01 08 66 69 7A 7A 62
75 7A 7A 00 01 0A 44
02 39 00 20 00 41 0F
10 02 04 40 41 7D 10
00 41 7D 0F 0B 20 00
41 05 10 02 04 40 41
7E 10 00 41 7E 0F 0B
20 00 41 03 10 02 04
40 41 7F 10 00 41 7F
0F 0B 20 00 10 00 20
```

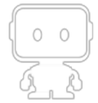


WebAssembly in the Browser



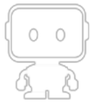

JS

Using WebAssembly From JavaScript



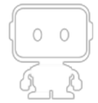
```
let imports = {
  math : {
    callback : x => console.log("result is", x)
  }
}


let module = await WebAssembly.instantiateStreaming(fetch('main.wasm'), imports)

let x = module.instance.exports.add(5, 10)
let y = module.instance.exports.subtract(10, 5)
let z = module.instance.exports.multiply(2, 5)
```

Instantiating Modules



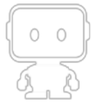
```
let imports = {
  math : {
    callback : x => console.log("Result", x)
  }
}


let module = await WebAssembly.instantiateStreaming(fetch('main.wasm'), imports)

let x = module.instance.exports.add(5, 10)
let y = module.instance.exports.subtract(10, 5)
let z = module.instance.exports.multiply(2, 5)
```

Using Functions in Modules



```
let imports = {
  math : {
    callback : x => console.log("result is", x)
  }
}


let module = await WebAssembly.instantiateStreaming(fetch('main.wasm'), imports)

let x = module.instance.exports.add(5, 10)
let y = module.instance.exports.subtract(10, 5)
let z = module.instance.exports.multiply(2, 5)
```

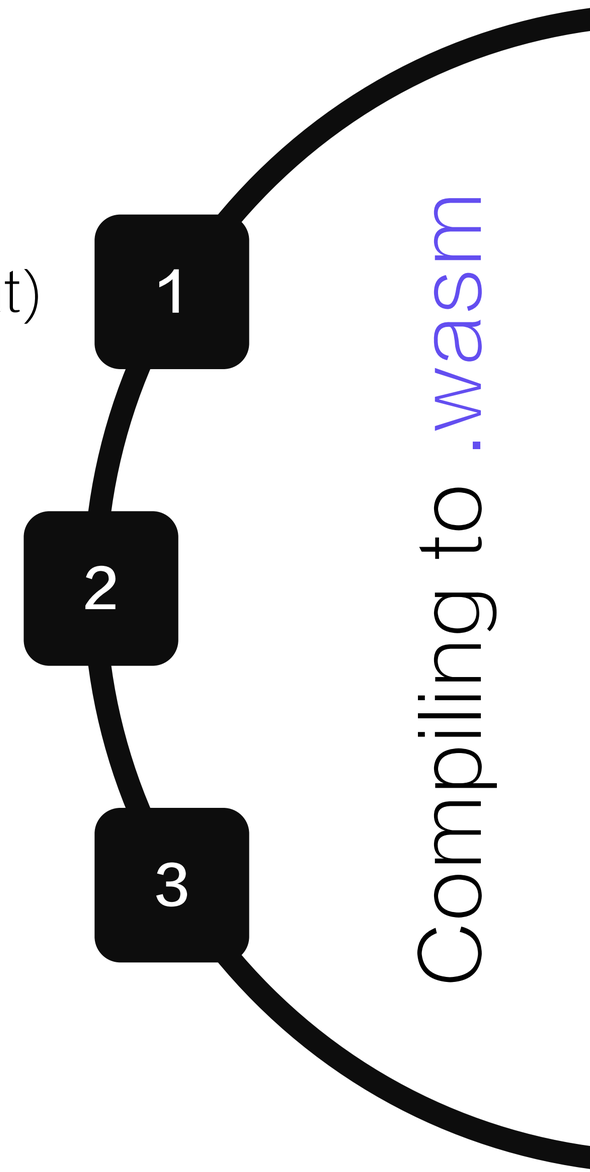
Importing JavaScript Functions into WebAssembly Modules



```
let imports = {
  math : {
    callback : x => console.log("result is", x)
  }
}


let module = await WebAssembly.instantiateStreaming(fetch('main.wasm'), imports)

let x = module.instance.exports.add(5, 10)
let y = module.instance.exports.subtract(10, 5)
let z = module.instance.exports.multiply(2, 5)
```

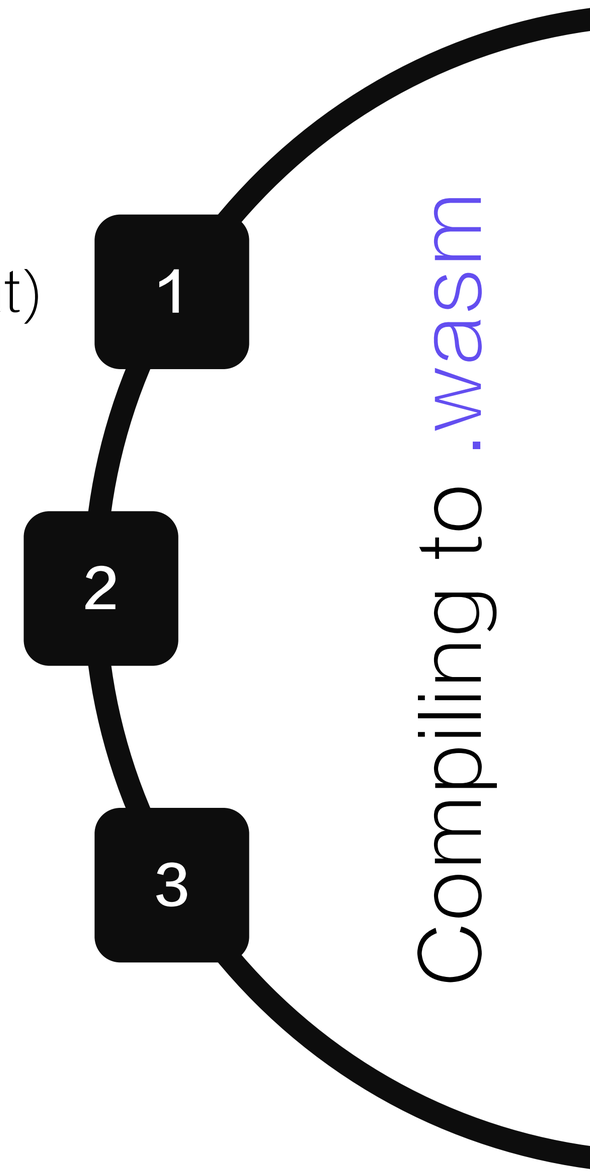

WA

Code in the **text representation** of WebAssembly directly(.wat)



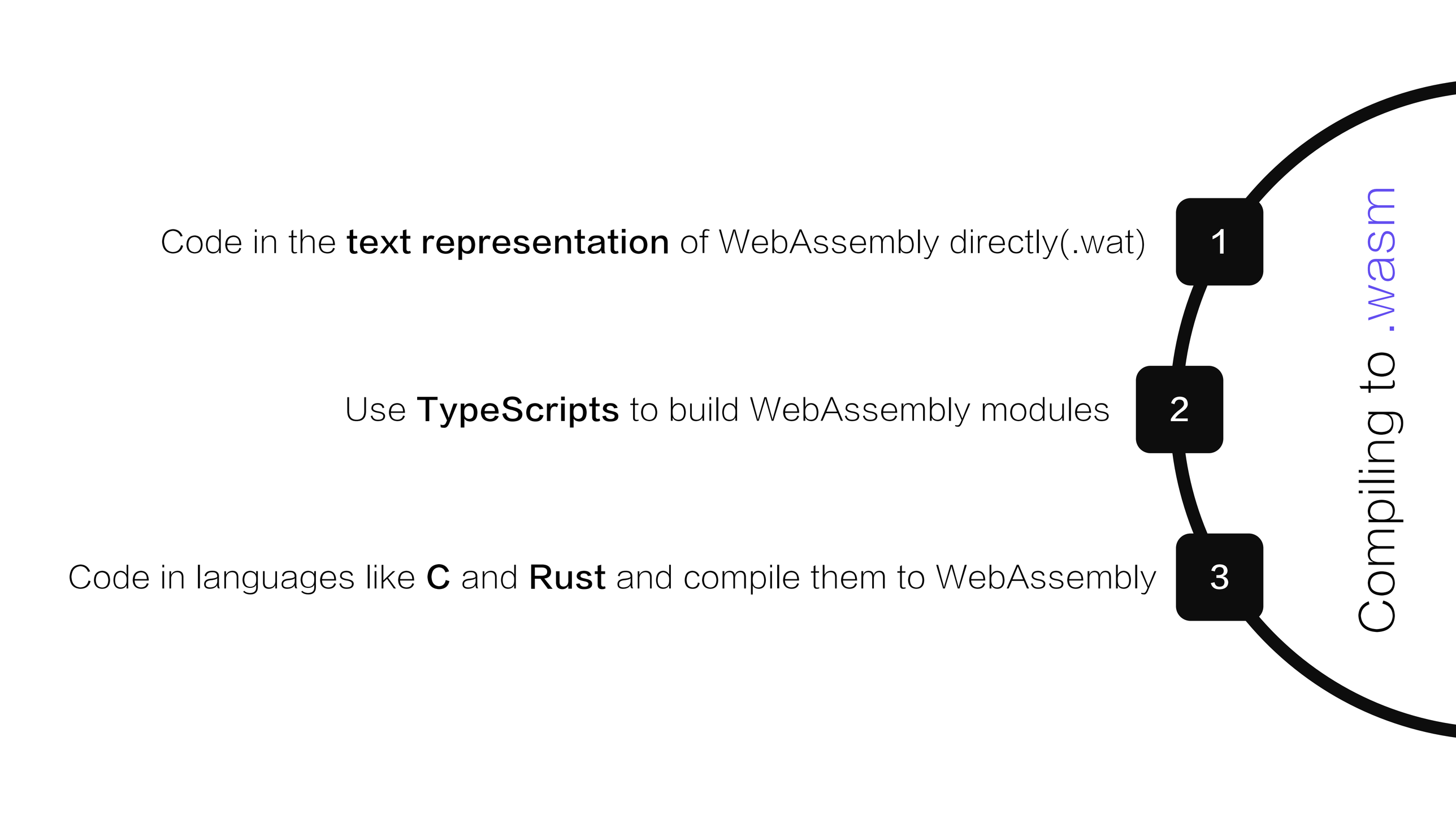
Code in the **text representation** of WebAssembly directly(.wat)

Use **TypeScripts** to build WebAssembly modules



Compiling to .wasm

1. Code in the **text representation** of WebAssembly directly(.wat)
2. Use **TypeScripts** to build WebAssembly modules
3. Code in languages like **C** and **Rust** and compile them to WebAssembly



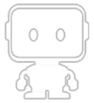
Compiling to .wasm

1. Code in the **text representation** of WebAssembly directly(.wat)
2. Use **TypeScripts** to build WebAssembly modules
3. Code in languages like **C** and **Rust** and compile them to WebAssembly

Compiling to .wasm

1. Code in the **text representation** of WebAssembly directly(.wat)
2. Use **TypeScripts** to build WebAssembly modules
3. Code in languages like **C and Rust** and compile them to WebAssembly

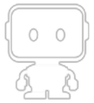
A Simple Module in WebAssembly Text Format



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

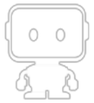
Declaring a Module



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

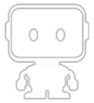
Creating Functions



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

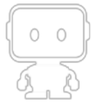
Exporting Functions



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

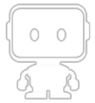
WebAssembly is Stack-based



```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

Stack

Push an Argument

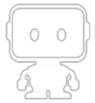


```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

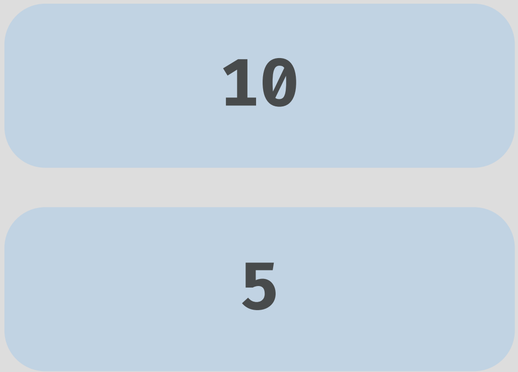
5

Stack

Push Another Argument

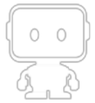


```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

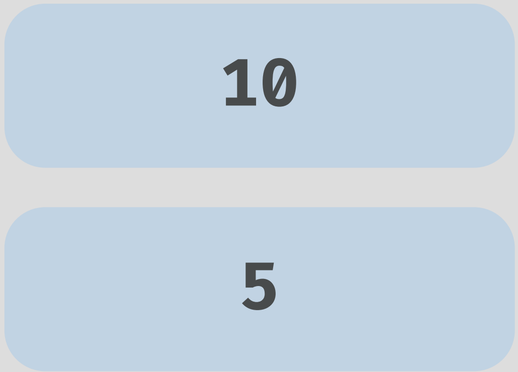


Stack

Pop the Arguments

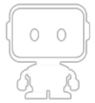


```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```



Stack

Add & Push

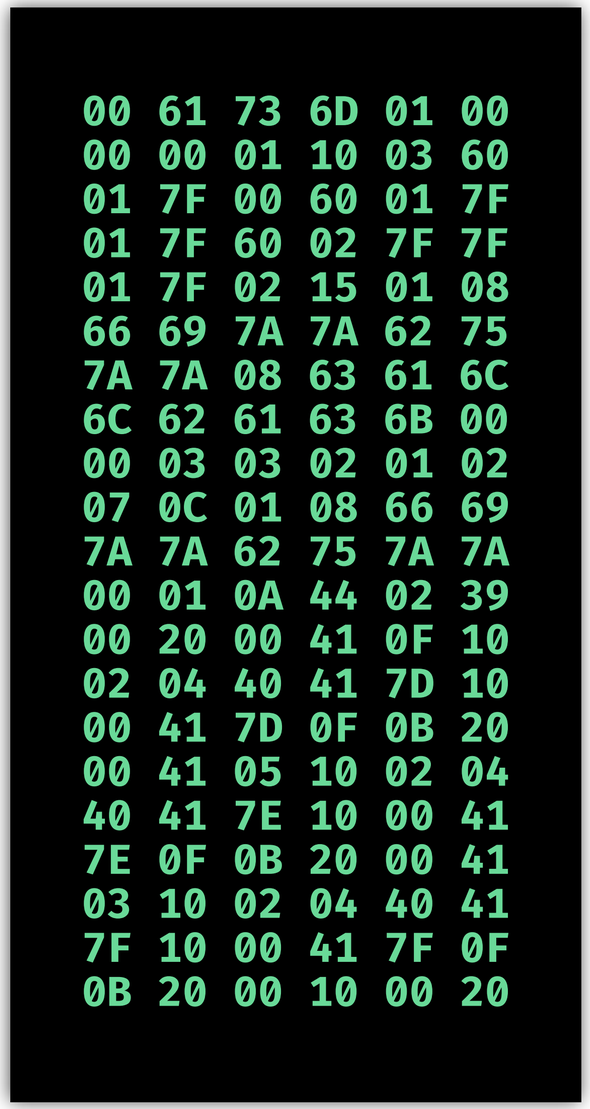


```
(module

  (import "math" "callback" (func $callback))

  (export "add" (func $add))
  (export "subtract" (func $subtract))

  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (func $subtract (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.sub
  )
)
```

15

Stack

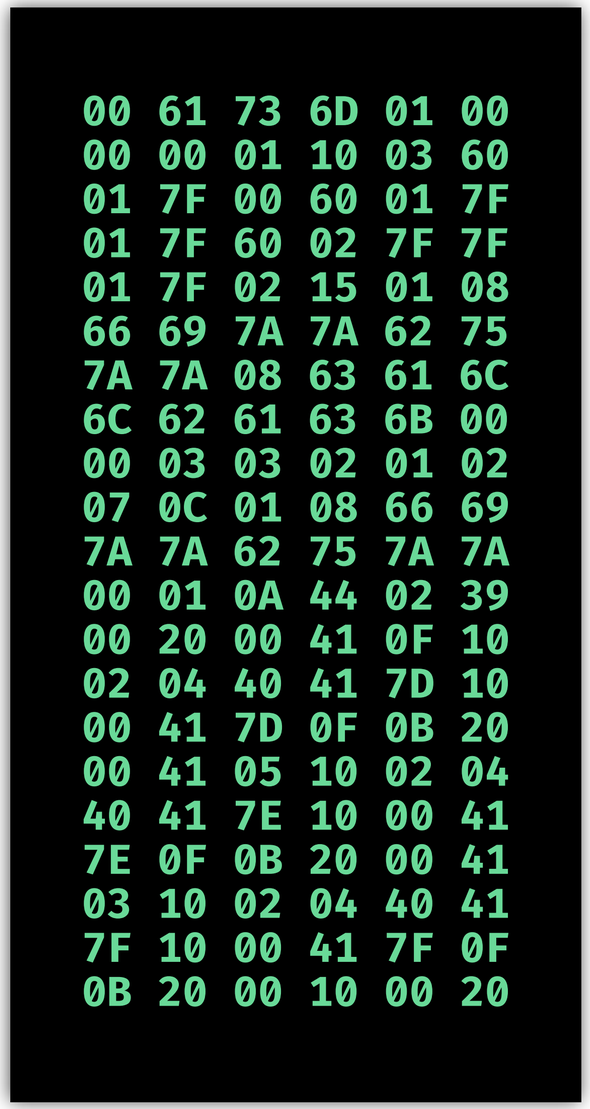
A Fancier Example (with Comments)



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

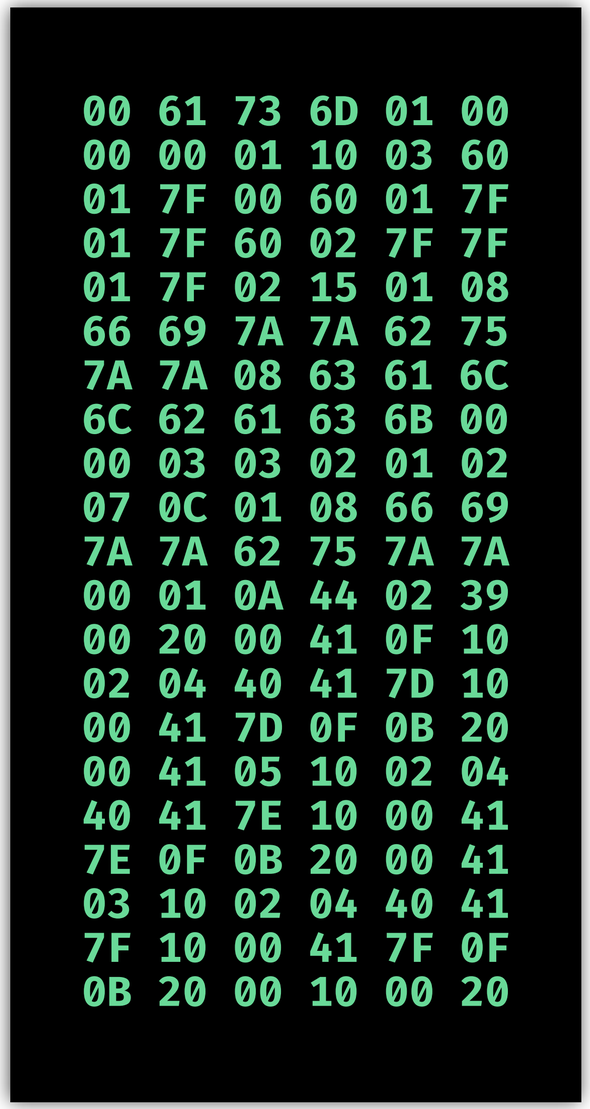
```
(module

  (import "math" "callback" (func $callback))
  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    (local $sum i32)   ;; define a local variable

    local.get $a
    local.get $b
    i32.add
    local.set $sum     ;; set the result to $sum

    local.get $sum     ;; put $sum on the stack
    call $callback     ;; call the callback with $sum

    local.get $sum     ;; put $sum on the stack again
  )
)
```

Defining and Setting a Local Variable




```
(module

  (import "math" "callback" (func $callback))
  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    (local $sum i32)   ;; define a local variable

    local.get $a
    local.get $b
    i32.add
    local.set $sum     ;; set the result to $sum

    local.get $sum     ;; put $sum on the stack
    call $callback     ;; call the callback with $sum

    local.get $sum     ;; put $sum on the stack again
  )
)
```

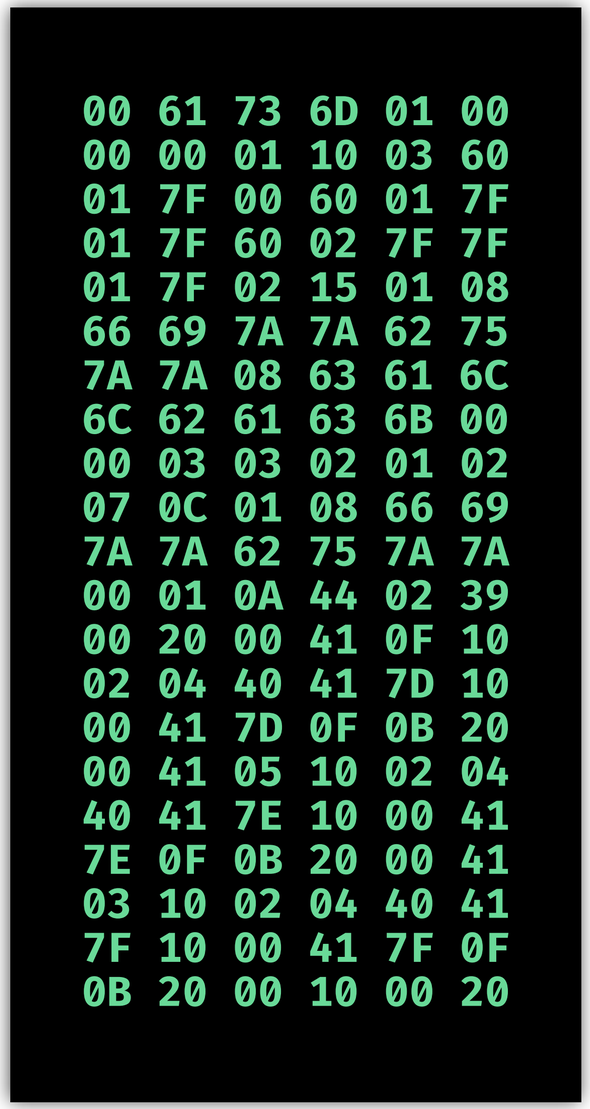
Calling the Callback



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

```
(module

  (import "math" "callback" (func $callback))
  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    (local $sum i32)   ;; define a local variable

    local.get $a
    local.get $b
    i32.add
    local.set $sum     ;; set the result to $sum

    local.get $sum     ;; put $sum on the stack
    call $callback     ;; call the callback with $sum

    local.get $sum     ;; put $sum on the stack again
  )
)
```

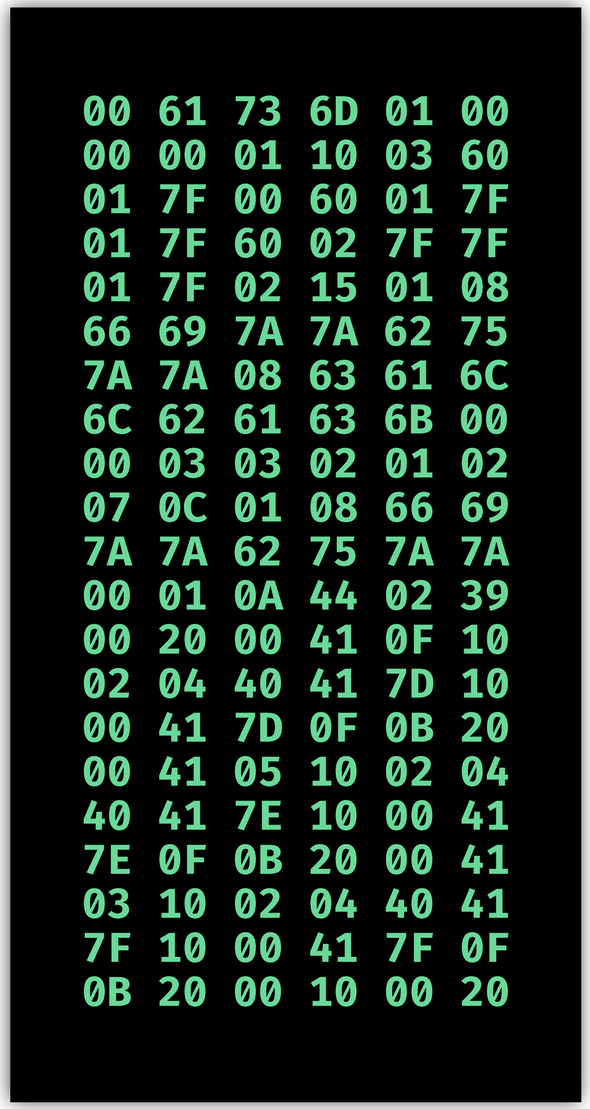
Mind the Stack



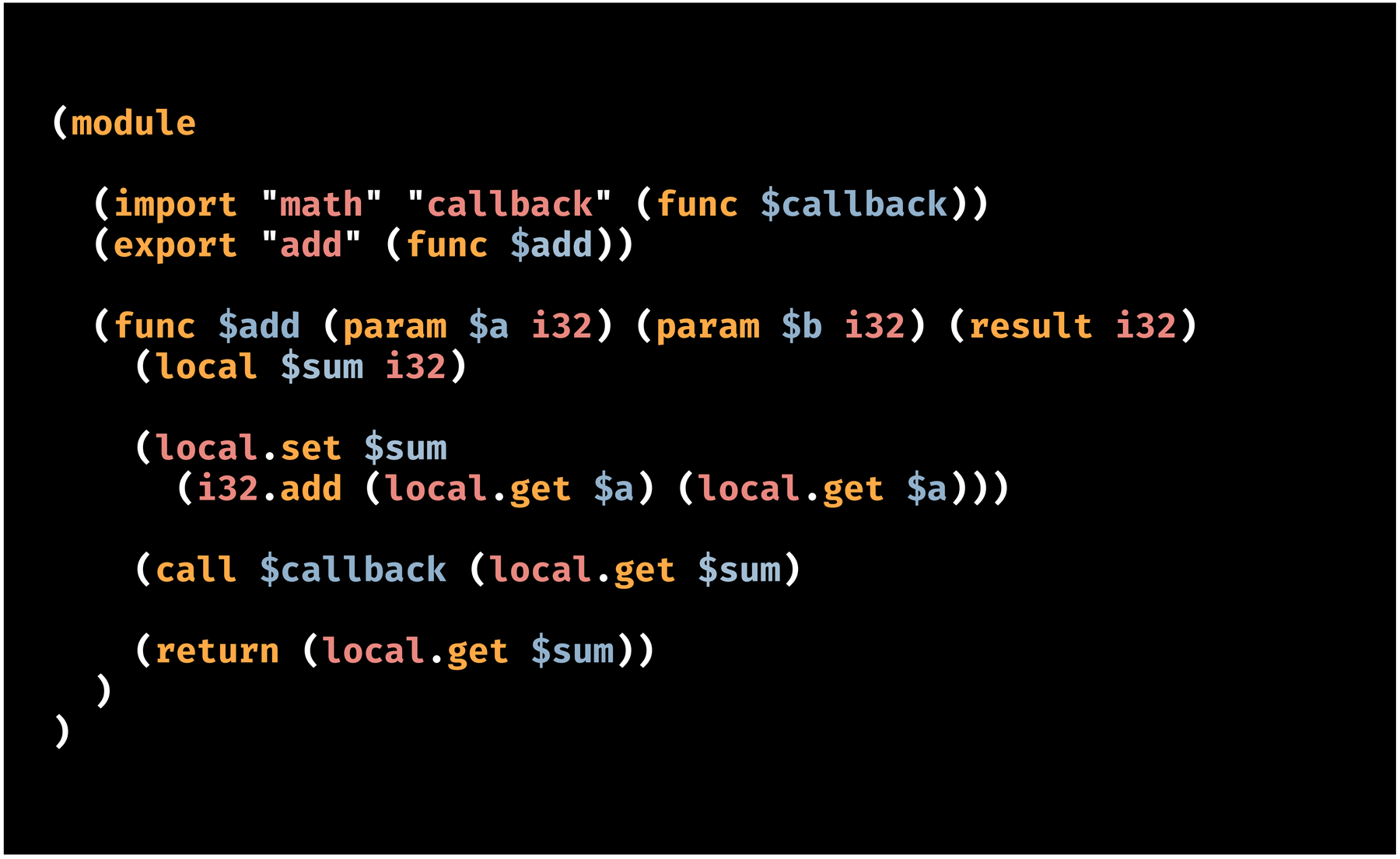
```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```

```
(module

  (import "math" "callback" (func $callback))
  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    (local $sum i32)   ;; define a local variable

    local.get $a
    local.get $b
    i32.add
    local.set $sum     ;; set the result to $sum

    local.get $sum     ;; put $sum on the stack
    call $callback     ;; call the callback with $sum

    local.get $sum     ;; put $sum on the stack again
  )
)
```

S-Expressions

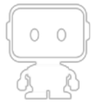


```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
07 0C 01 08 66 69
7A 7A 62 75 7A 7A
00 01 0A 44 02 39
00 20 00 41 0F 10
02 04 40 41 7D 10
00 41 7D 0F 0B 20
00 41 05 10 02 04
40 41 7E 10 00 41
7E 0F 0B 20 00 41
03 10 02 04 40 41
7F 10 00 41 7F 0F
0B 20 00 10 00 20
```



```
(module

  (import "math" "callback" (func $callback))
  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    (local $sum i32)

    (local.set $sum
      (i32.add (local.get $a) (local.get $a)))

    (call $callback (local.get $sum)

    (return (local.get $sum))
  )
)
```

Other Stuff



Shared Memory

```
(memory 1)

(data
  (i32.const 0)
  "Hello World\n"
)

(func $foo
  (i32.store8
    (i32.const 12)
    (i32.const 0)
  )
)
```

Globals

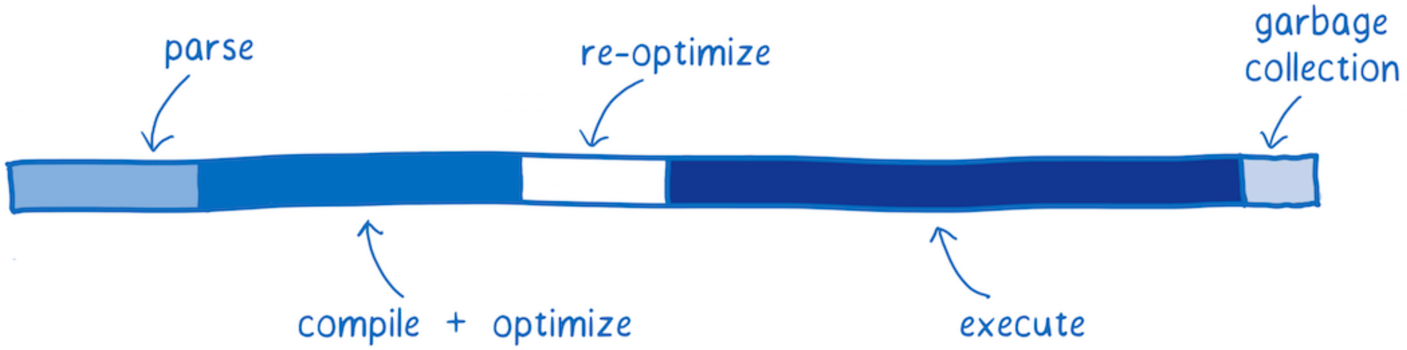
```
(global $n
  (import "foo" "bar")
  (mut i32)
)

(func $baz (result i32)

 (global.set $n
   (i32.const 42))

 (return
   (global.get $n ))
)
```

Tables

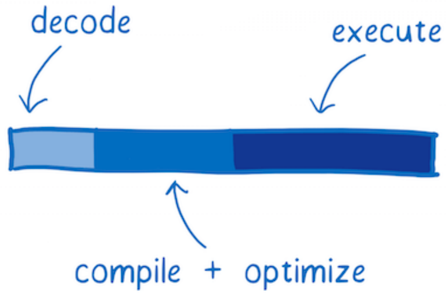
```
(table 3 anyfunc)

(elem (i32.const 0)
  $foo)

(elem (i32.const 1)
  $bar $baz)

(func $foo)
(func $bar)
(func $baz)
```

Time for Coding

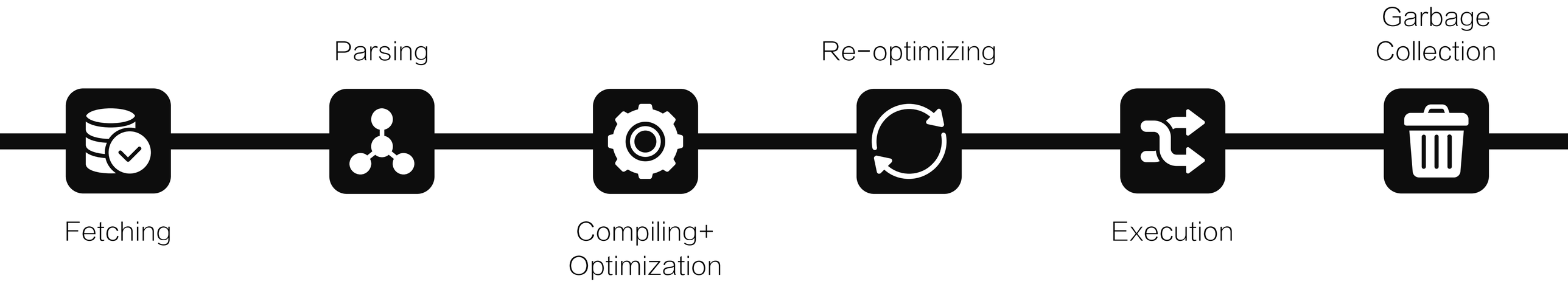
https://github.com/mohammadhashemii/IE-Assignments/tree/master/WebAssembly/wat2wasm



Average time spent by a JS engine using JIT — Source: Lin Clark



Average time spent by a JS engine using WA — Source: Lin Clark


Fetching
Parsing
Compiling+
Optimization
Re-optimizing
Execution
Garbage
Collection

WA

Fetching

```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```

JS

WA

More compact!

```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
```

Fetching

```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```

JS

WA

Parsing

Fetching

JS

WA

Parsing

Fetching

```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```

Source

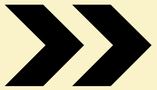
JS

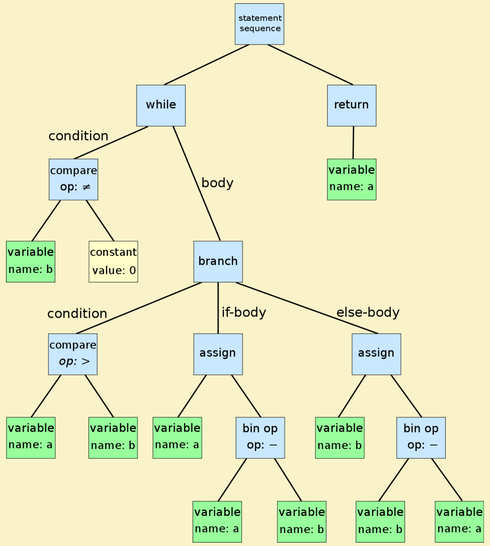
WA

Parsing

Fetching

```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```





Source

AST

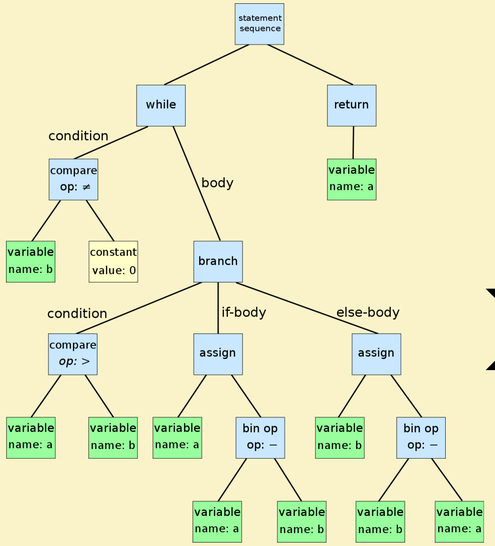
JS

WA

Parsing

Fetching

```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```



```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
```

JS

Source

AST

JS bytecode

WA

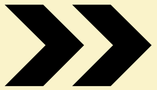
```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
```

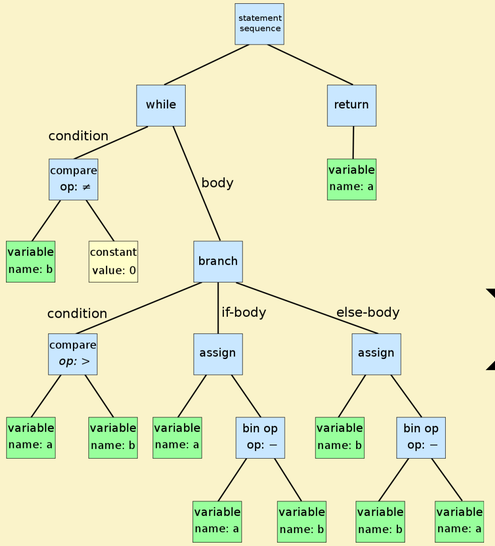
WA bytecode

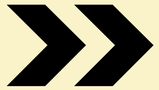
Parsing

Fetching

```
int x = 0;
int y = 0;

for (x=0; x<12; x++)
{
  for (y=0; y<12; y++)
  {
    scan(x, y);
  }
}
```







```
00 61 73 6D 01 00
00 00 01 10 03 60
01 7F 00 60 01 7F
01 7F 60 02 7F 7F
01 7F 02 15 01 08
66 69 7A 7A 62 75
7A 7A 08 63 61 6C
6C 62 61 63 6B 00
00 03 03 02 01 02
```

JS

Source

AST

JS bytecode

WA

Parsing

Fetching

Compiling+
Optimization

JS

WA

Parsing

Fetching

Compiling+
Optimization

Compiled during execution time!

JS

WA

Parsing

Fetching

Compiling+
Optimization

Compiled during execution time!

Dynamic types

JS

WA

Compiled during execution time!

Dynamic types Multiple version of the same code

JS

1. The compiler does not have to spend time running the code to observe what types are being used before it starts compiling optimized code.

WA

Compiled during execution time!

Dynamic types Multiple version of the same code

JS

WA

1. The compiler does not have to spend time running the code to observe what types are being used before it starts compiling optimized code.

2. The compiler does not have to compile different version of the same code based on those different types it observers.

Compiled during execution time!

Dynamic types Multiple version of the same code

JS

WA

1. The compiler does not have to spend time running the code to observe what types are being used before it starts compiling optimized code.

2. The compiler does not have to compile different version of the same code based on those different types it observers.

3. More optimization have already been done ahead of time in LLVM. So less work is needed to compile and optimize it.

Compiled during execution time!

Dynamic types Multiple version of the same code

JS

WA

Parsing

Re-optimizing

Fetching

Compiling+
Optimization

JS

WA

Parsing

Re-optimizing

Fetching

Compiling+
Optimization

Wrong assumptions made by JIT!

JS

WA

🤣🤣🤣🤣🤣

Parsing

Re-optimizing

Fetching

Compiling+
Optimization

Wrong assumptions made by JIT!

😔

JS

WA

Parsing

Re-optimizing

Fetching

Compiling+
Optimization

Execution

JS

WA

Parsing

Re-optimizing

Fetching

Compiling+
Optimization

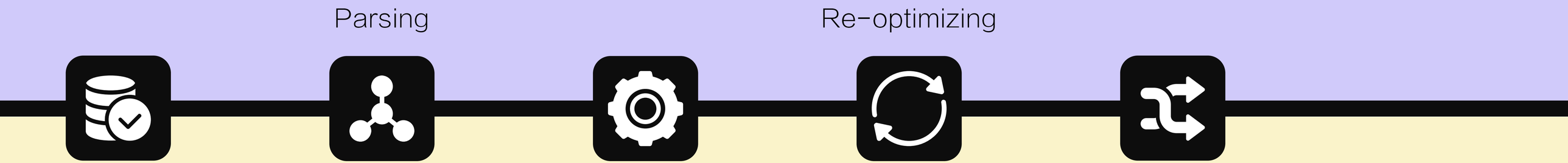
Execution

More human readable  Harder for JIT to optimize

JS

WA

was designed as compiler target!



More human readable  Harder for JIT to optimize

JS

10% to 800%

WA

Parsing

Re-optimizing

Garbage Collection

Fetching

Compiling+ Optimization

Execution

JS


unity WebGL
AngryBots


unity WebGL
AngryBots

WA + Games = ❤️

https://github.com/mohammadhashemii/IE-Assignments/tree/master/WebAssembly/snake-game

Resources

[1] An introduction to WebAssembly – Guy Royse

[2] A cartoon intro to WebAssembly – Lin Clark

We hope you enjoyed it!

Mohammad Hashemi

Narges Ghasemi