The JVM vs WebAssembly

An In-Depth Comparative Analysis

Loophole Labs

Why did we create WebAssembly when we already have the JVM

What are the differences between the JVM and the WebAssembly VM?

Shivansh Vij

Founder, Loophole Labs

Twitter: @ConfusedQubit

Github: https://github.com/shivanshvij

Linkedin: https://linkedin.com/in/shivanshvij

Loophole Labs

Loophole Labs

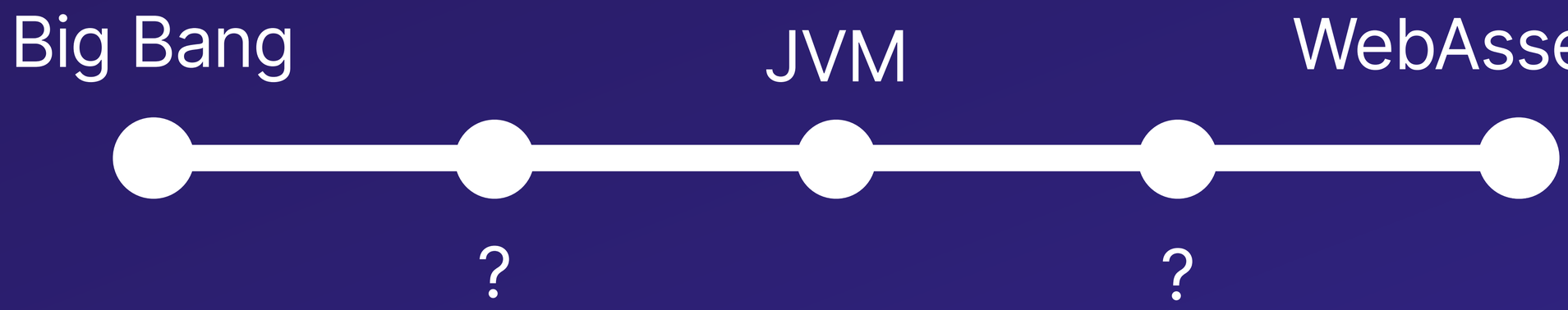
Twitter: @LoopholeLabs

Homepage: https://loopholelabs.io

Scale: https://scale.sh

Discord: https://loopholelabs.io/discord

A Brief History Lesson



Machine Code

In the Beginning, there was ~~the Big Bang~~

- All software is presented to the CPU as machine code
  - Readability is extremely low (not practical to write)
- Assign easy to remember “names” to each machine code operation
  - ADD X Y Z = Add Y and Z, Save into X
- Create an “assembler” to parse these instructions and “assemble” them into native machine code

Assembly

```
MOV     R0  #10
MOV     R1  #3
ADD     R0  R0  R1
END
```

Machine Code

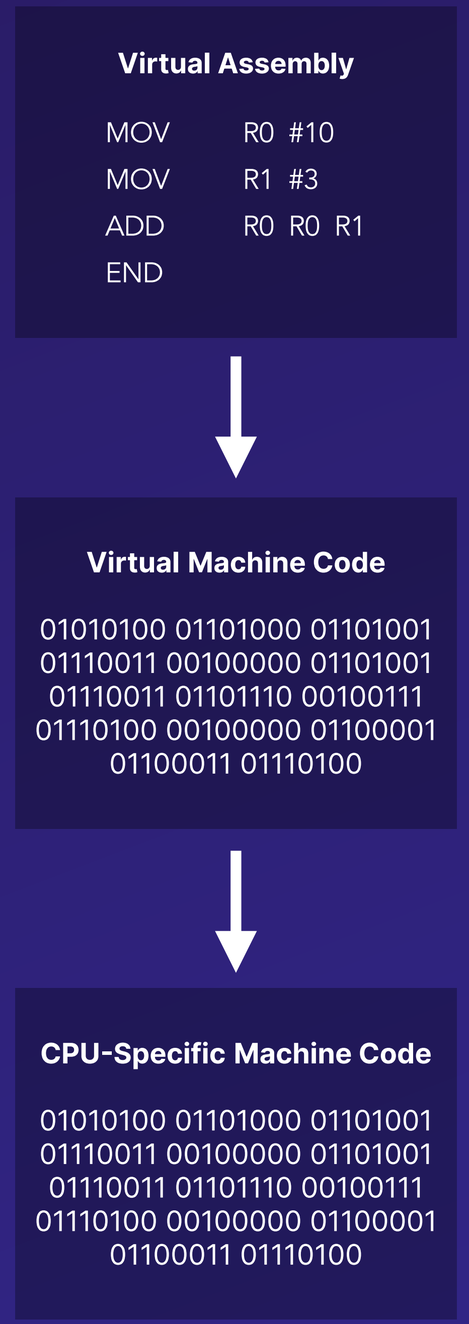
01010100 01101000 01101001
01110011 00100000 01101001
01110011 01101110 00100111
01110100 00100000 01100001
01100011 01110100 01110101
01100001 01101100 01101100
01111001 00100000 01001101
01100001 01100011 01101000
01101001 01101110 01100101
00100000 01000011 01101111
01100100 01100101

All Our Problems are Solved, Right?

Different Processors
=
Different Assembly Languages

One CPU To Rule Them All

- What if there was a “Virtual CPU” that had its own dialect of machine code?
  - Software could target the Virtual CPU’s machine code
  - Translate the vCPU’s machine code to the unique machine code for various CPUs
- Software supports only the virtual layer, which is responsible for supporting real CPUs



A Stack-Based Approach to Bytecode

- JVM's Virtual CPU needs a bytecode format
  - It needs to be CPU-agnostic
    - We can't use registers because CPUs often have unique registers
- A "Stack-Based" Virtual Machine
  - Store values on stack, pop them off to "consume" them
  - Will run on any CPU that supports stacks

Register Assembly

```
MOV   R0  #10
MOV   R1  #3
ADD   R0  R0  R1
END
```

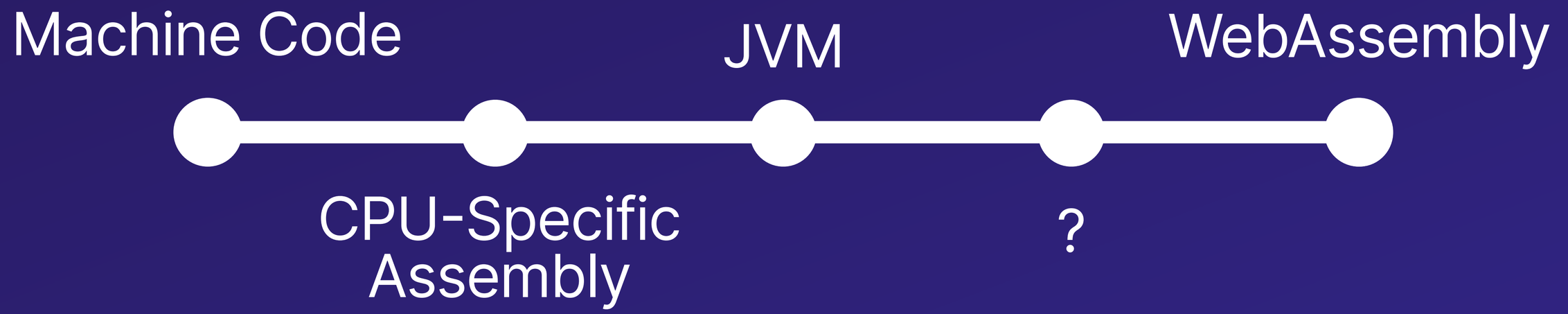
vs

Stack Assembly

```
PUSH  #10
PUSH  #3
ADD
END
```

Java
The “JVM” is Born!

A (Brief?) History Lesson



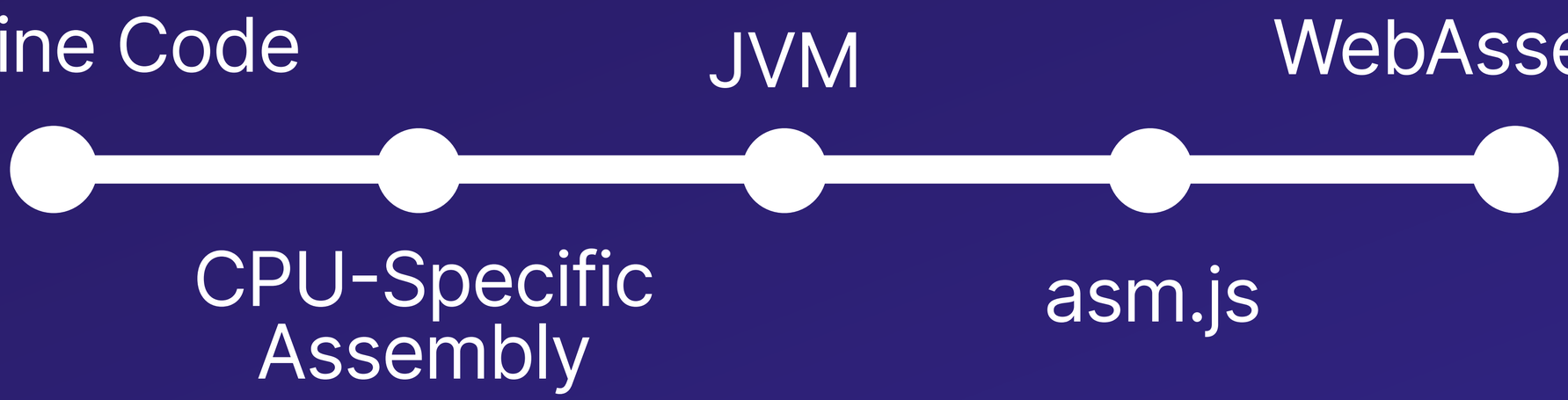
Now Let's Take It To the Browser

Why not use the JVM in the Browser?

JavaScript wasn't Fast Enough

```
function Example(stdlib, foreign, heap) {
  "use asm";
  var exp = stdlib.Math.exp;
  var log = stdlib.Math.log;
  var values = new stdlib.Float64Array(heap);
  function logSum(start, end) {
    start = start|0;
    end = end|0;
    var sum = 0.0, p = 0, q = 0;
    for (p = start << 3, q = end << 3; (p|0) < (q|0); p = (p + 8)|0) {
      sum = sum + +log(values[p>>3]);
    }
    return +sum;
  }
}
```

A Brief History Lesson



A New Build Target

- Similar to x86, languages can be “compiled” for WebAssembly
  - Browsers will ship with a Wasm VM that can run the compiled bytecode
- Key Requirements for the bytecode
  - Near-native performance
  - Streamable
  - Stack-Based (with “structured control flow”)
  - Sandboxing by default (with extensibility)

WebAssembly Bytecode Format

- Represented as an Abstract Syntax Tree (AST)
  - Can be encoded/decoded very efficiently
  - Load/instantiate on-the-fly as it’s streamed in
  - Language Agnostic
  - Easier for AOT/JIT compilers to optimize ASTs
  - Validation and Verification
  - Structured Control Flow
  - Future Flexibility

Structured Control Flow

- JVM has unstructured control flow
  - Java needs to load Java classes and verify them at startup
  - Instructions like “goto” and “ifeq” need to be validated
  - Utilizes Stack Maps to achieve this in a single pass
    - Required because the bytecode format cannot be modified
- WebAssembly control flow requires structured constructs
  - “if, “then", and “else”
  - Blocks and loops

```
void print(boolean x) {
    if (x) {
        System.out.println(1);
    } else {
        System.out.println(0);
    }
}
```

```
  void print(boolean);
  Code:
  0: iload_1
  1: ifeq 14
  4: getstatic #7 // java/lang/System.out:Ljava/io/PrintStream
  7: iconst_1
  8: invokevirtual #13 // java/io/PrintStream.println
 11: goto 21
 14: getstatic #7 // java/lang/System.out:Ljava/io/PrintStream
 17: iconst_0
 18: invokevirtual #13 // java/io/PrintStream.println
 21: return
```

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

 (func
   ;; change to positive number (true)
   ;; if you want to run the if block
   (i32.const 0)
   (call 0)
 )

 (func (param i32)
   local.get 0
   (if
     (then
       i32.const 1
       call $log ;; should log '1'
     )
     (else
       i32.const 0
       call $log ;; should log '0'
     )
   )
 )

 (start 1) ;; run the first function automatically
)
```

Do More by Doing Less

- The JVM footprint makes it problematic in the browser
  - It provides many capabilities (in an opinionated way)
- What does WebAssembly VM do differently?
  - Has no opinions
  - Provides the bare minimum
    - No garbage collector
    - No standard library
    - Few Types (i32, i64, f32, f64 - no strings)
    - Easy and safe to extend

Small But Mighty

- Fast starts (microsecond range)
- Extremely small memory footprint (few kilobytes)
- Fast cleanups (recover the linear memory chunk)
- Ideal for environments like the browser
  - But also interesting on the server-side
  - Opens the door to polyglot programming
- A true “universal compilation target”

Loophole Labs

Twitter: @LoopholeLabs

Homepage: https://loopholelabs.io

Scale: https://scale.sh

Discord: https://loopholelabs.io/discord