Demystifying EVM Opcodes

macro

Overview

- Why learn EVM opcodes?
- What are Virtual Machines?
- Intro to the EVM
- A Slightly Easier Syntax (Trim)
- Solidity code in opcodes!

Why learn EVM Opcodes?

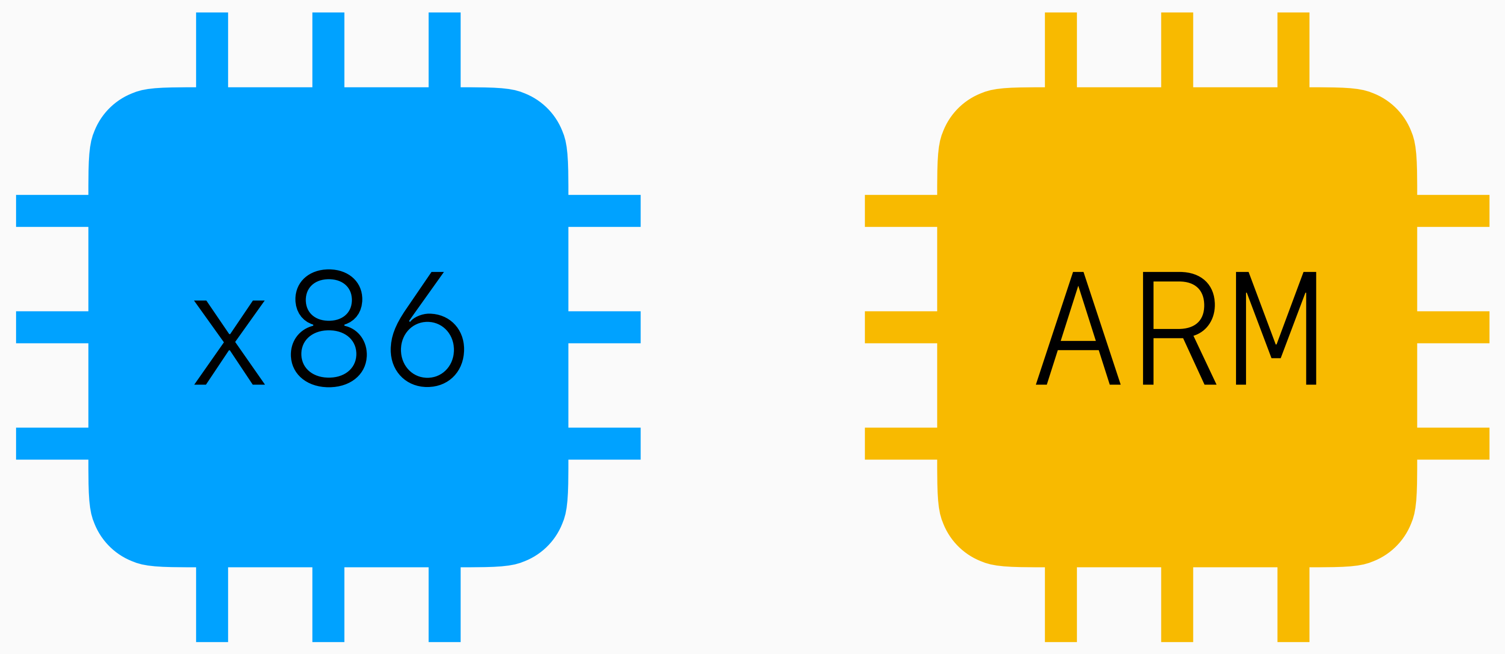
To become a better Solidity engineer.

A Better Solidity Engineer

- Understands why Solidity is designed the way it is
- Has a deeper understanding of common design patterns
- Has internalized how smart contracts run on the EVM
- Can easily gas-optimize low hanging fruit scenarios
- **KNOWS WHAT THEIR CODE IS DOING UNDER THE HOOD**

macro

What are virtual machines?



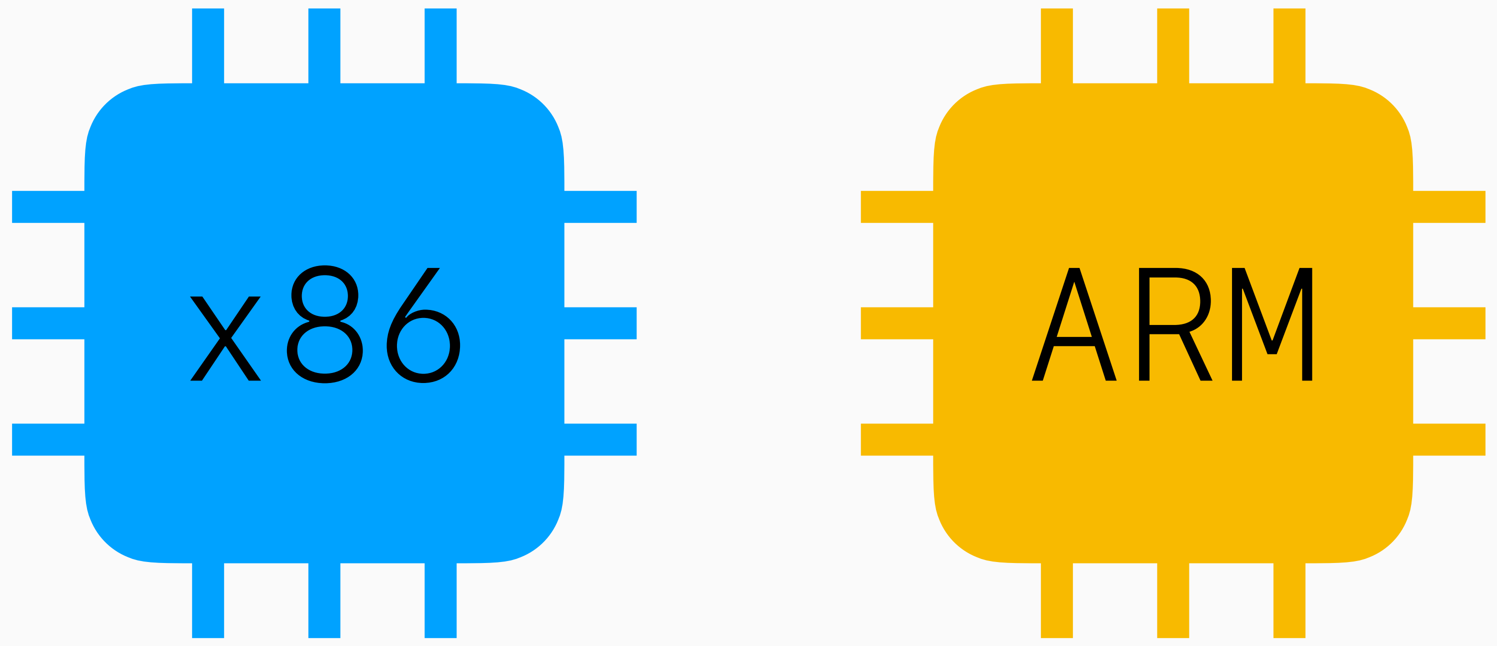
Physical Machines

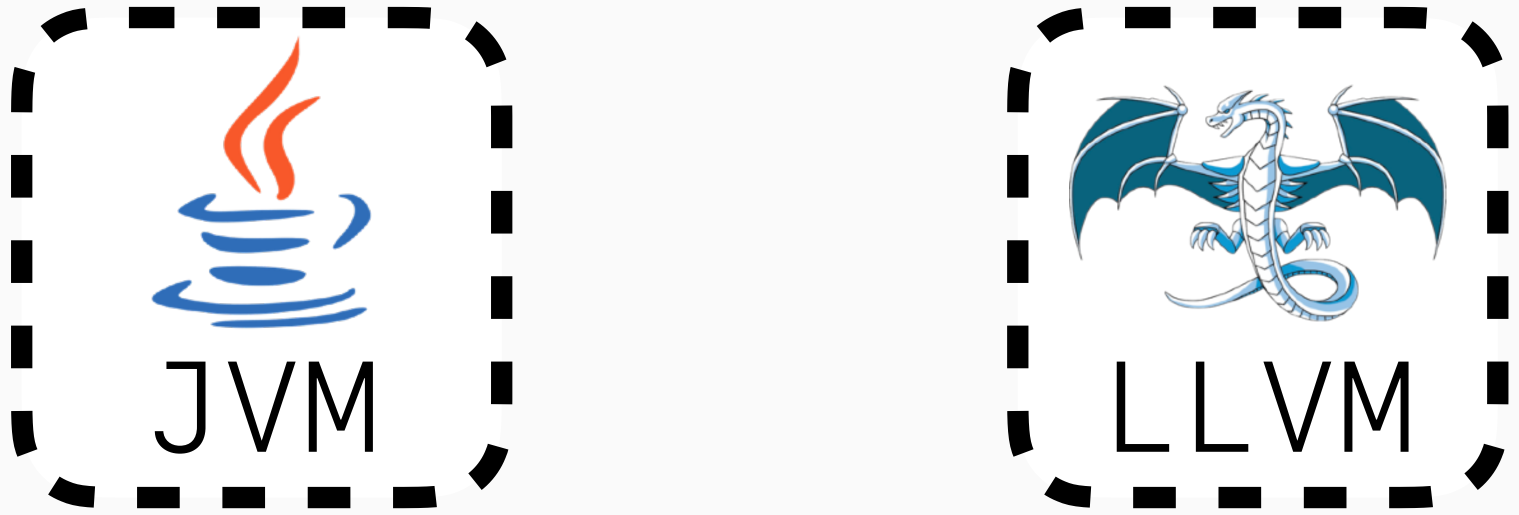
```
0000 0100 – ADD
0010 1100 – SUB
```

} **Opcodes!**

macro

What are virtual machines?





Physical Machines

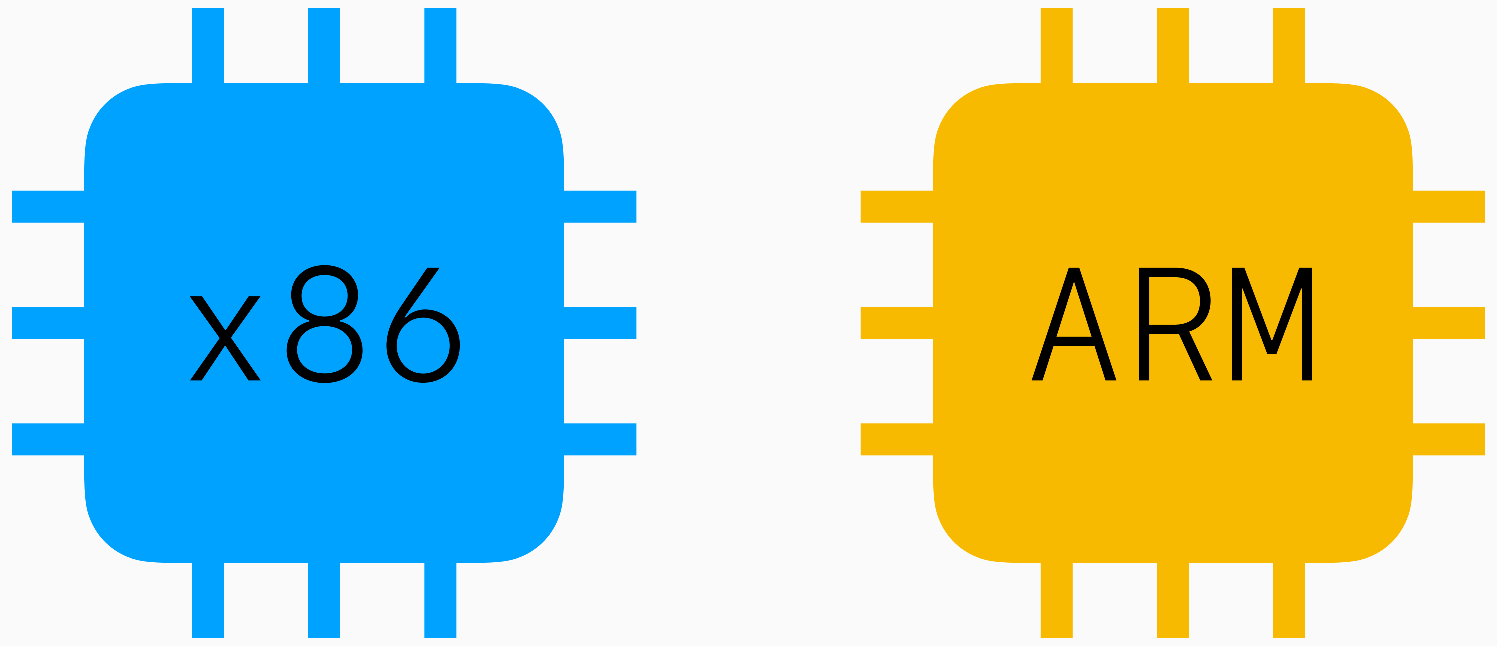
```
0000 0100 — ADD
0010 1100 — SUB
```

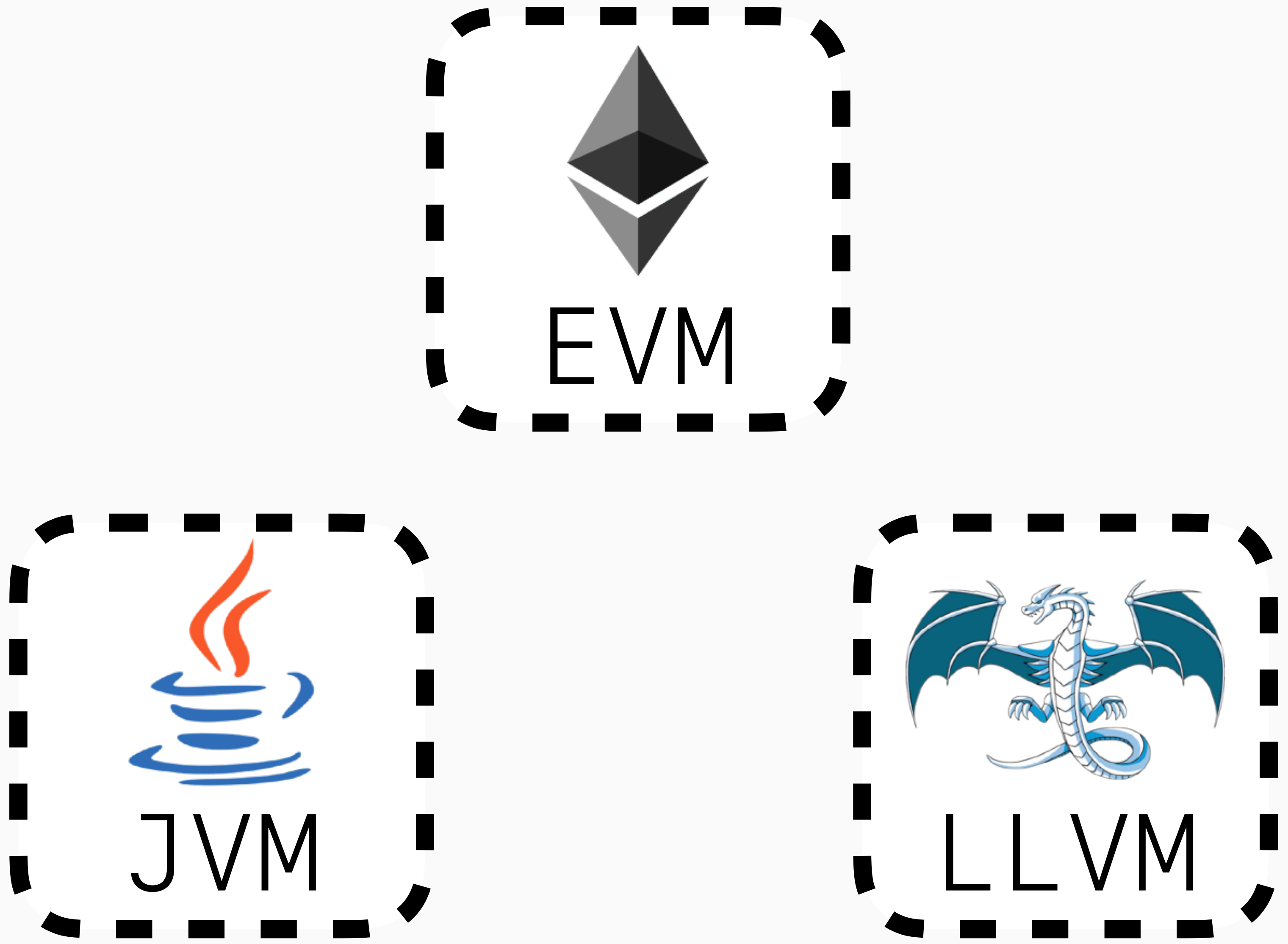
Virtual Machines

```
0110 0000 — IADD (JVM)
```

macro

What are virtual machines?





Physical Machines

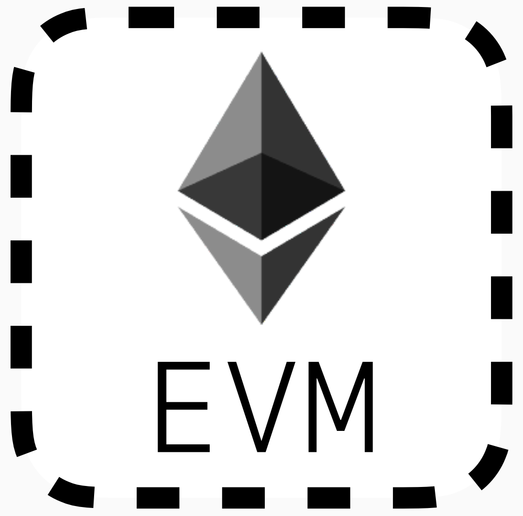
```
0000 0100 – ADD
0010 1100 – SUB
```

Virtual Machines

```
0110 0000 – IADD (JVM)
0000 0001 – ADD (EVM)
```

macro

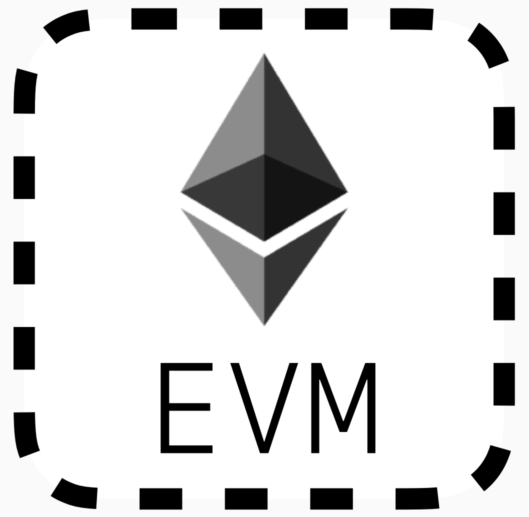
What are virtual machines?



```
0000 0001 — ADD (EVM)
```

macro

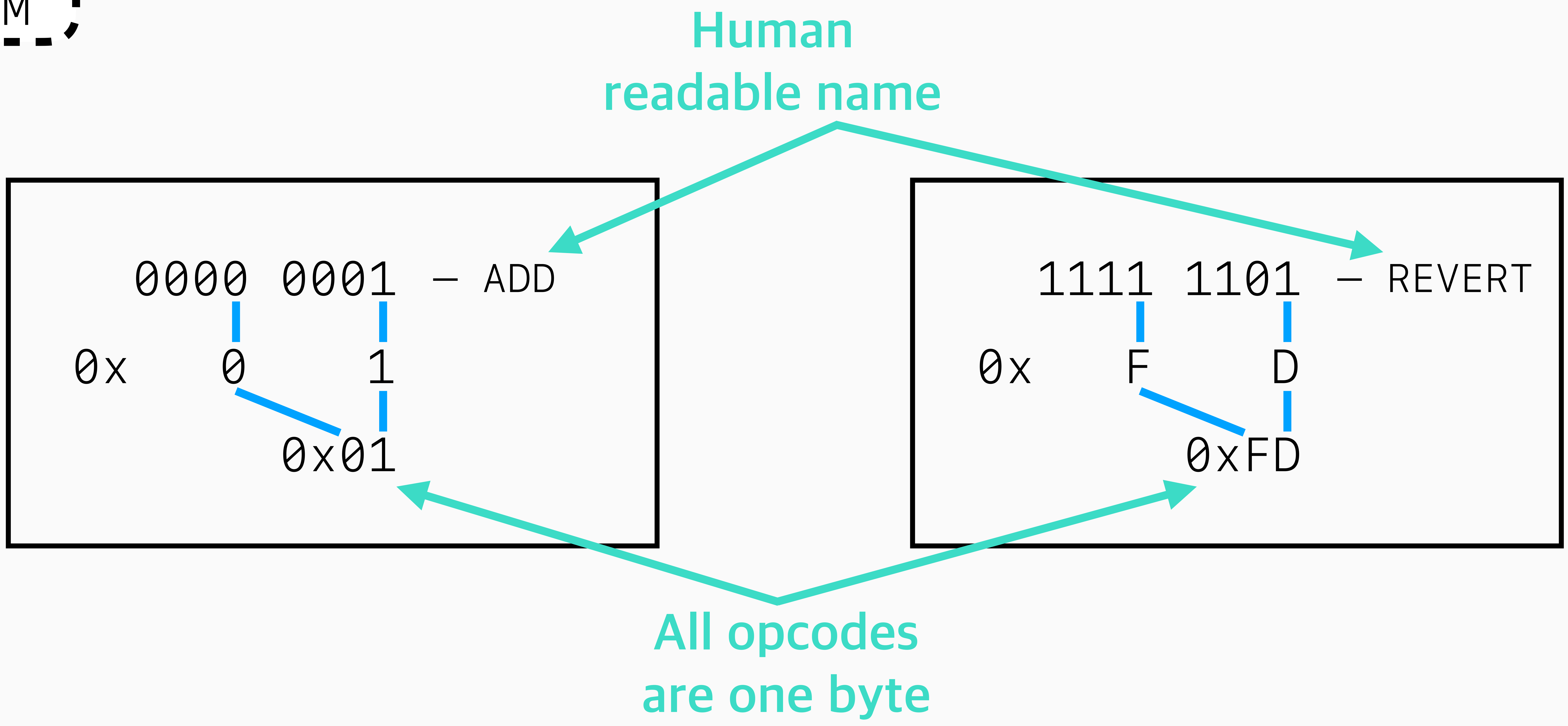
Intro to the EVM



0000 0001 — ADD (EVM)

macro

Opcode Syntax



Intro to the EVM

The EVM is Stack-Based

Bottom of Stack:

```
PUSH1 0x03
PUSH2 0x0004
PUSH1 0x09
SWAP2
ADD
```

The EVM is Stack-Based

Bottom of Stack: 0x03

```
➡ PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
  SWAP2
  ADD
```

EVM

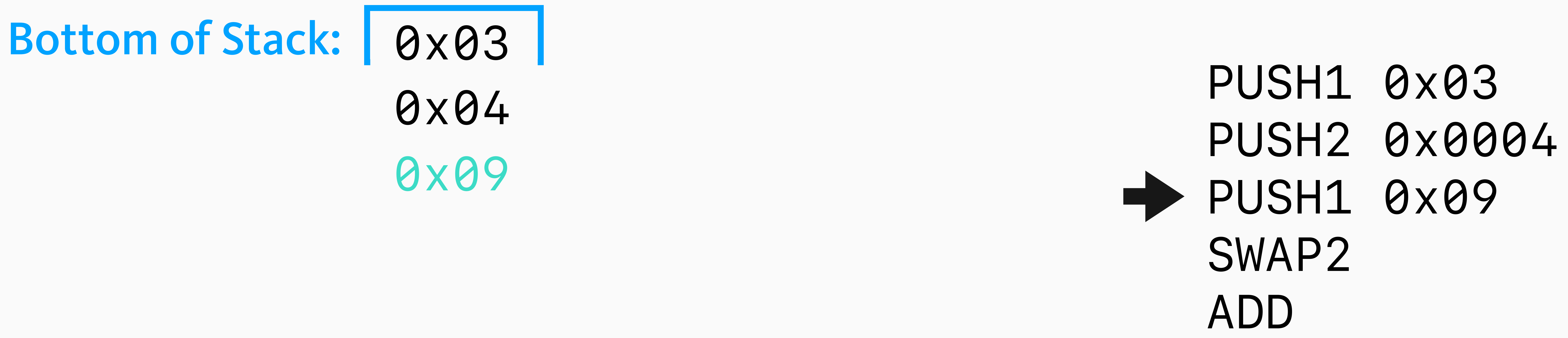
The EVM is Stack-Based

Bottom of Stack: 0x03

0x04

```
 PUSH1 0x03
➡ PUSH2 0x0004
 PUSH1 0x09
 SWAP2
 ADD
```

The EVM is Stack-Based



```
PUSH1 0x03
PUSH2 0x0004
PUSH1 0x09
SWAP2
ADD
```

The EVM is Stack-Based

Bottom of Stack:

```
0x09
0x04
0x03
```

```
  PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
➡ SWAP2
  ADD
```

The EVM is Stack-Based

Bottom of Stack:

```
0x09
0x04
0x03
```

```
  PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
  SWAP2
➡ ADD
```

The EVM is Stack-Based

Bottom of Stack:

```
0x09
0x07
```

```
  PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
  SWAP2
➡ ADD
```

The EVM is Stack-Based

Bottom of Stack:

```
0x09
0x07
```

```
  PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
  SWAP2
➜ ADD
  CALLER
```

The EVM is Stack-Based

Bottom of Stack:

```
0x09
0x07
0xf034…beef
```

```
  PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
  SWAP2
  ADD
➡ CALLER
```

The EVM is Stack-Based

Bottom of Stack:

```
0x09
0x07
0xf034…beef
```

Each stack item is 32 bytes

```
  PUSH1 0x03
  PUSH2 0x0004
  PUSH1 0x09
  SWAP2
  ADD
➡ CALLER
```

EVM

Memory & Storage

Memory
Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000000
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
...     ...
```

Stack

```
➡
PUSH1 0x03
PUSH1 0x10
MSTORE
```

EVM

Memory & Storage

Memory
Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000000
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
…       …
```

Stack

```
| 0x03 |
```

```
➡ PUSH1 0x03
  PUSH1 0x10
  MSTORE
```

Memory & Storage

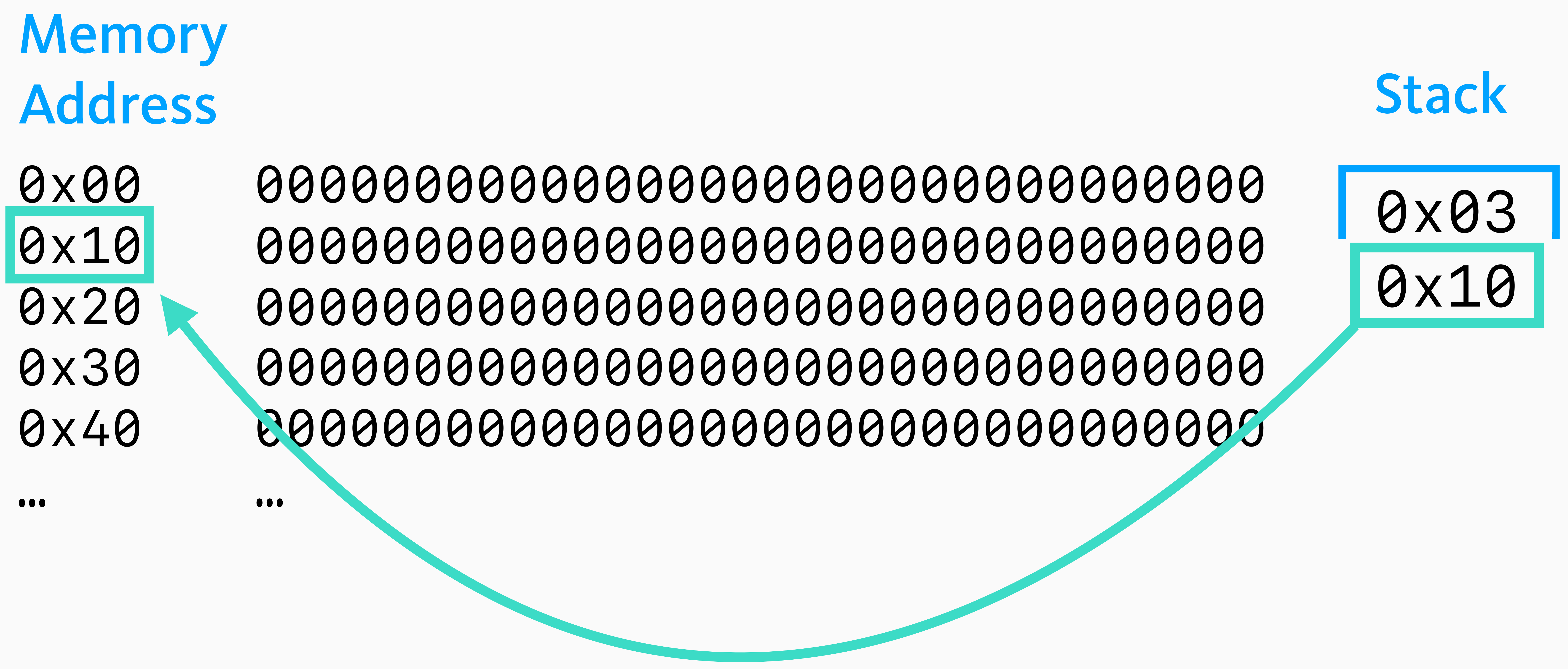
Memory Address	
0x00	00000000000000000000000000000000
0x10	00000000000000000000000000000000
0x20	00000000000000000000000000000000
0x30	00000000000000000000000000000000
0x40	00000000000000000000000000000000
…	…

Stack

```
0x03
0x10
```

```
  PUSH1 0x03
➜ PUSH1 0x10
  MSTORE
```

Memory & Storage



Memory
Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000000
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
...     ...
```

Stack

```
0x03
0x10
```

```
PUSH1 0x03
PUSH1 0x10
MSTORE
```

Memory & Storage

Memory

Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000000
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
...     ...
```

Stack

```
0x03
0x10
```

```
PUSH1 0x03
PUSH1 0x10
MSTORE
```

EVM

Memory & Storage

Memory

Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000003
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
…       …
```

Stack

```
  PUSH1 0x03
  PUSH1 0x10
➡ MSTORE

  PUSH1 0x10
  MLOAD
```

macro

EVM

Memory & Storage

Memory Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000003
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
…       …
```

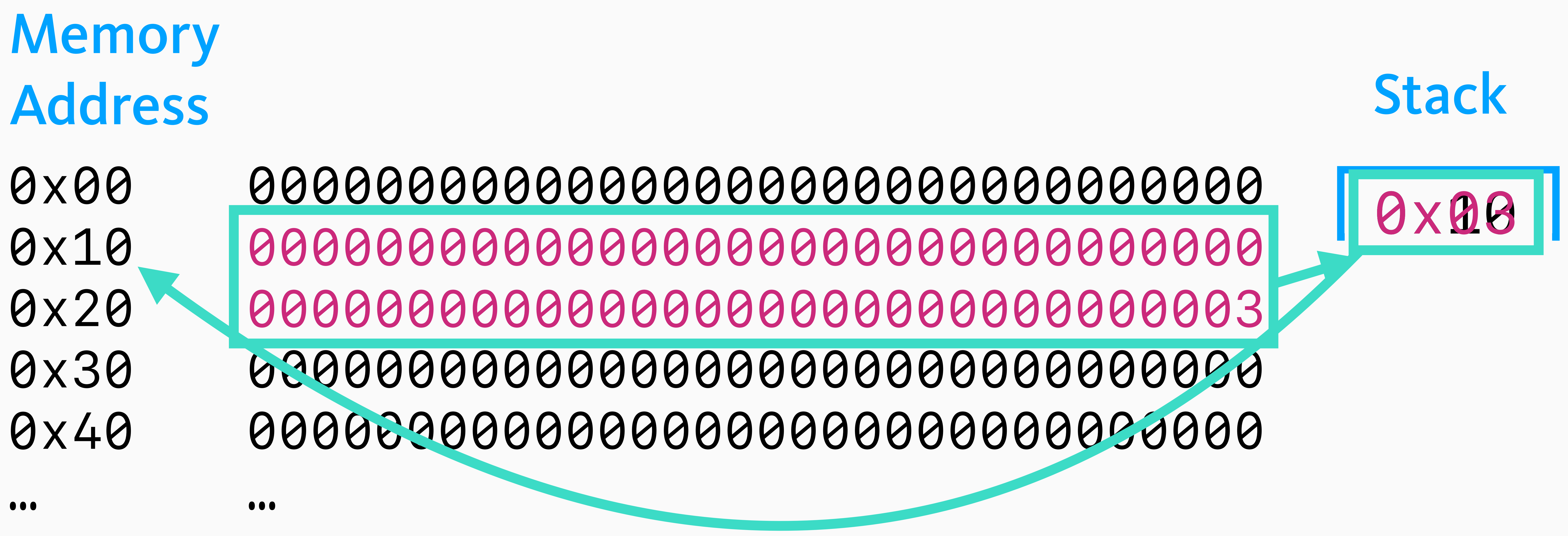
Stack

```
0x10
```

```
PUSH1 0x03
PUSH1 0x10
MSTORE

➡ PUSH1 0x10
MLOAD
```

Memory & Storage



```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000003
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
…       …
```

```
PUSH1 0x03
PUSH1 0x10
MSTORE

PUSH1 0x10
MLOAD
```

Memory & Storage

EVM

Memory

Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000003
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
…       …
```

Stack

```
0x03
```

```
  PUSH1 0x03
  PUSH1 0x10
  MSTORE

  PUSH1 0x10
➜ MLOAD
```

Memory & Storage

Memory
Address

```
0x00    00000000000000000000000000000000
0x10    00000000000000000000000000000000
0x20    00000000000000000000000000000000
0x30    00000000000000000000000000000000
0x40    00000000000000000000000000000000
…       …
```

Stack

```
0x03
```

```
PUSH1 0x03
PUSH1 0x10
SSTORE

PUSH1 0x10
SLOAD
```

```
SSTORE: 2,900 – 20,000+ gas
MSTORE: 3+ gas
```

Introducing: Trim

Let's learn a slightly easier syntax!

Why? For easier reading!

macro

https://github.com/0xMacro/trim

Trim: S-Expressions

Instead of writing these…

…you get to write these!

```
PUSH1 0x03
PUSH1 0x20
MSTORE
```

$\longrightarrow$

```
(MSTORE 0x20 0x03)
```

```
PUSH1 0x20
MLOAD
```

$\longrightarrow$

```
(MLOAD 0x20)
```

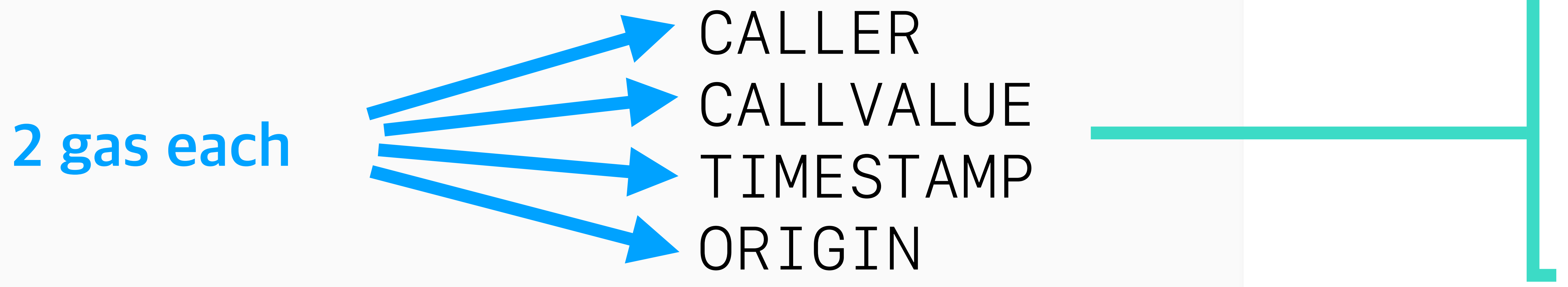
macro

https://github.com/0xMacro/trim

Let’s Learn Solidity Opcodes!

macro

Solidity Opcodes: Primitives



```
contract Basics {

    function foo() external {
        msg.sender;
        msg.value;
        block.timestamp;
        tx.origin;
    }
}
```

Solidity Opcodes: Payable

```
(EQ 0x00 CALLVALUE)
ISZERO
(JUMPI #payable-revert)
```

No generated bytecode!

```
contract Payable {

    function foo() external {
        // …
    }



    function bar() payable external {
        // …
    }
}
```

macro

Solidity Opcodes: Storage Variables

No generated bytecode!

```
(SSTORE 0x01 0x07)
(SSTORE 0x02 0x09)
```

Index based!

```
(SLOAD 0x01)
(MSTORE …)
(RETURN …)
```

```
contract StorageVars {
    uint x;
    uint y;
    uint z;

    function foo() external {
        y = 7;
        z = 9;
    }

    function getY() external view returns
(uint) {
        return y;
    }
}
```

macro

Solidity Opcodes: Storage Variables

```
(SSTORE 0x01 0x07)
(SSTORE 0x02 0x09)

(SLOAD 0x01)
(MSTORE …)
(RETURN …)
```

```
contract StorageVars {
    uint x;
    uint y;
    uint z;

    function foo() external {
        y = 7;
        z = 9;
    }

    function getY() external view returns
(uint) {
        return y;
    }
}
```

Solidity Opcodes: Storage Variables

```
(SSTORE 0x02 0x07)
(SSTORE 0x01 0x09)
```

```
(SLOAD 0x02)
(MSTORE …)
(RETURN …)
```

```
contract StorageVars {
    uint x;
    uint z;
    uint y;

    function foo() external {
        y = 7;
        z = 9;
    }

    function getY() external view returns
(uint) {
        return y;
    }
}
```

macro

Solidity Opcodes: Compact Storage Vars

Less than 256 bits, so
Solidity compacts its usage

```
contract StorageCompact {
    uint8 x; // Storage slot 0
    uint8 y; // Storage slot 0

    function foo() external {
        uint8 sum = x + y;
    }
}
```

macro

Solidity Opcodes: Compact Storage Vars

Bit masking!

Load x
```
(AND 0xFF (SLOAD 0x00))
```

Load y
```
(AND 0xFF00 (SLOAD 0x00))
(SHR 0x08 _)
```

x + y
```
ADD
```

Cold SLOAD: 2100 gas
Hot SLOAD: 100 gas

```
contract StorageCompact {
    uint8 x; // Storage slot 0
    uint8 y; // Storage slot 0

    function foo() external {
        uint8 sum = x + y;
    }
}
```

macro

Solidity Opcodes: If Statement

```
(EQ 0x03 (SLOAD 0x00))
(JUMPI #then)
(JUMP #else)

#then
JUMPDEST
(MSTORE … “Can’t be 3”)
(REVERT …)

#else
JUMPDEST
(ADD 0x07 (SLOAD 0x00))
(SSTORE 0x00)
```

Another storage load!

```
contract IfElse {
    uint x;
   function foo() external {
      if (x == 3) {
          revert(“Can’t be 3”);
      }
      else {
          x = x + 7;
      }
   }
}
```

macro

Solidity Opcodes: Ext Function Calls

```
(CALL
  … ; 63/64 gas
  … ; token address
  … ; 0 wei
  … ; ABI fn call mem addr
  … ; ABI fn call length
  … ; Return value dest mem addr
  … ; Return value dest length
)
```

```
contract FnCalls {
    ERC20 token;
    address receiver;

    function foo() external {
        token.transfer(receiver, 7);
    }
}
```

macro

Solidity Opcodes: Internal Function Calls

```
(push #afterwards)
(JUMP #bar-fn)
#afterwards


#bar-fn
JUMPDEST
PUSH1 0x07
(SLOAD 0x00)
ADD
(SSTORE 0x00 _)
JUMP
```

```
contract FnCalls {
    uint x;

    function foo() external {
        bar();
    }


    function bar() internal {
      x = x + 7;
    }
}
```

macro

Solidity Opcodes: Internal Function Calls (2)

Push param onto stack

```
PUSH1 0x07
(push #afterwards)
(JUMP #bar-fn)
#afterwards
```

Change: ASSUME param is on stack!

```
#bar-fn
JUMPDEST
~~PUSH1 0x07~~
(SLOAD 0x00)
ADD
(SSTORE 0x00 _)
JUMP
```

```
contract FnCalls {
    uint x;

    function foo() external {
        bar(7);
    }


    function bar(uint _amount) internal {
      x = x + _amount;
    }
}
```

macro

That's all, folks!

Check out all the opcodes:

github.com/wolflo/evm-opcodes

Request an Audit:

0xMacro.com

Learn more in our Fellowship:

0xMacro.com/engineering-fellowship

Thanks!