Ethereum WebAssembly:

The Future of Ethereum Smart Contracts

Jake Lang
github.com/jakelang

Problems with the EVM

Problems with the EVM

- It’s complicated!

Problems with the EVM

- It’s complicated!
- It’s slow!

Problems with the EVM

- It’s complicated!
- It’s slow!
- Limited language support and developer tools

Problems with the EVM

- It’s complicated!
- It’s slow!
- Limited language support and developer tools

List of production-ready languages targeting the EVM:

Introducing WebAssembly (WASM)

Introducing WebAssembly (WASM)

- Executable binary format designed for the web

Introducing WebAssembly (WASM)

- Executable binary format designed for the web
- Very similar to traditional computer architectures

Introducing WebAssembly (WASM)

- Executable binary format designed for the web
- Very similar to traditional computer architectures
- Highly performant

Introducing WebAssembly (WASM)

- Executable binary format designed for the web
- Very similar to traditional computer architectures
- Highly performant
- Supported by many languages and developer tools

So what is *Ethereum* WebAssembly (ewasm)?

So what is *Ethereum* WebAssembly (ewasm)?

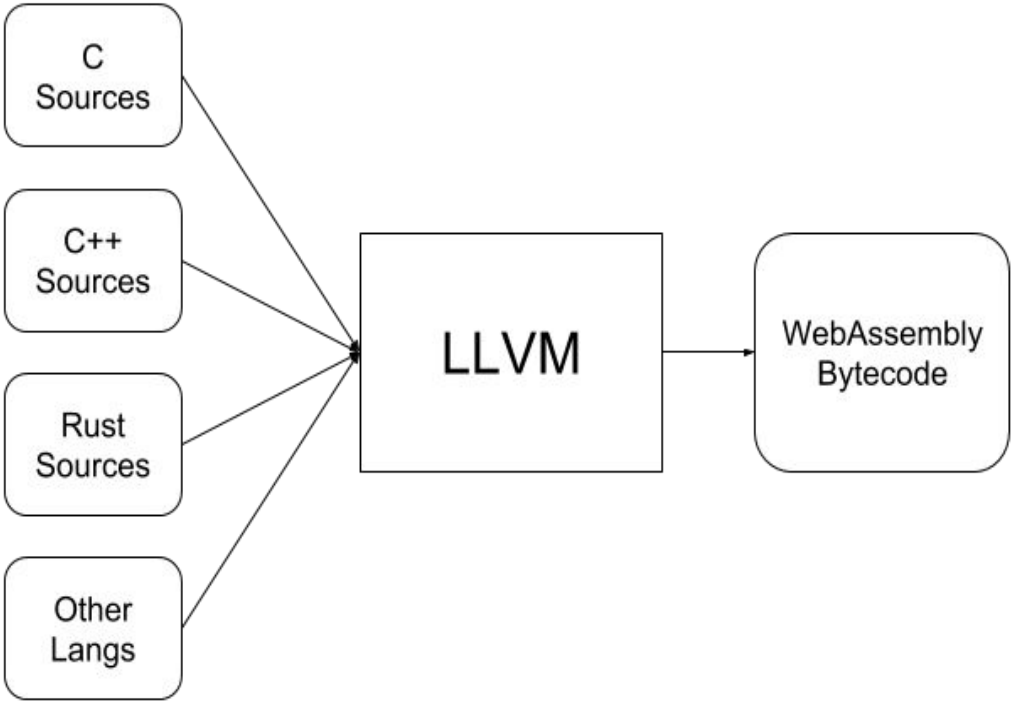
Ewasm is wasm!

So what is *Ethereum* WebAssembly (ewasm)?

Ewasm is wasm!*

*Minus nondeterminism

Language Support for WASM: LLVM



AssemblyScript: the contract language for a post-EVM world

AssemblyScript: the contract language for a post-EVM world

- Subset of TypeScript

AssemblyScript: the contract language for a post-EVM world

- Subset of TypeScript
- Easy to learn and use, like JavaScript

AssemblyScript: the contract language for a post-EVM world

- Subset of TypeScript
- Easy to learn and use, like JavaScript
- Intended to be the primary language for ewasm contract development

A basic token in Solidity

```
pragma solidity ^0.4.20;

contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(
        uint256 initialSupply
        ) public {
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
        return true;
    }
}
```

A basic token
In AssemblyScript

```
@ewasm
class TokenContract extends Contract {
  @store
  _balance: Map<Address, Amount>

  init(owner: Address, balance: Amount): void {
     this.setBalance(owner, balance)
  }

  getBalance(user: Address): Amount {
    return this._balance[user]
  }

  setBalance(user: Address, amt: Amount): void {
    this._balance[user] = amt
  }

  transfer(sender: Address, recipient: Address, amt: Amount): void {
    var balanceSender = this.getBalance(sender)
    var balanceRecip  = this.getBalance(recipient)
    assert(balanceSender >= amt)
    this.setBalance(sender, balanceSender - amt)
    this.setBalance(recipient, balanceRecip + amt)
  }
}
```

Creds to Lane Rettig (@lrettig) for this code, and the API behind it!

Backwards compatibility

Backwards compatibility

- RuneVM (Parity’s EVM interpreter in WASM!)

Backwards compatibility

- RuneVM (Parity’s EVM interpreter in WASM!)
- Planned Solidity compiler support through Yul

Backwards compatibility

- RuneVM (Parity's EVM interpreter in WASM!)
- Planned Solidity compiler support through Yul
- EVM to WebAssembly transcompilation using either evm2wasm or YEVM (using Yul).

Where we are now

- ewasm VM implementation: ✓
- geth and aleth support for ewasm VM: ✓
- APIs for common WASM languages: ✓
- Internal testnet: ✓
- Public testnet: **Scheduled before Devcon**

Deploying Ethereum WebAssembly

Deploying Ethereum WebAssembly

… it’s complicated.

Deploying Ethereum WebAssembly

Option 1: Shasper

The main execution engine thereof

Deploying Ethereum WebAssembly

Option 1: Shasper

The main execution engine thereof

Option 2: Mainnet

First as “precompiles”, then as contracts

Thank you!

Interested?

Join the discussion on Gitter!

https://gitter.im/ewasm/Lobby