K B A

Web3 Basics

How to make Ethereum smart contract Interactive?

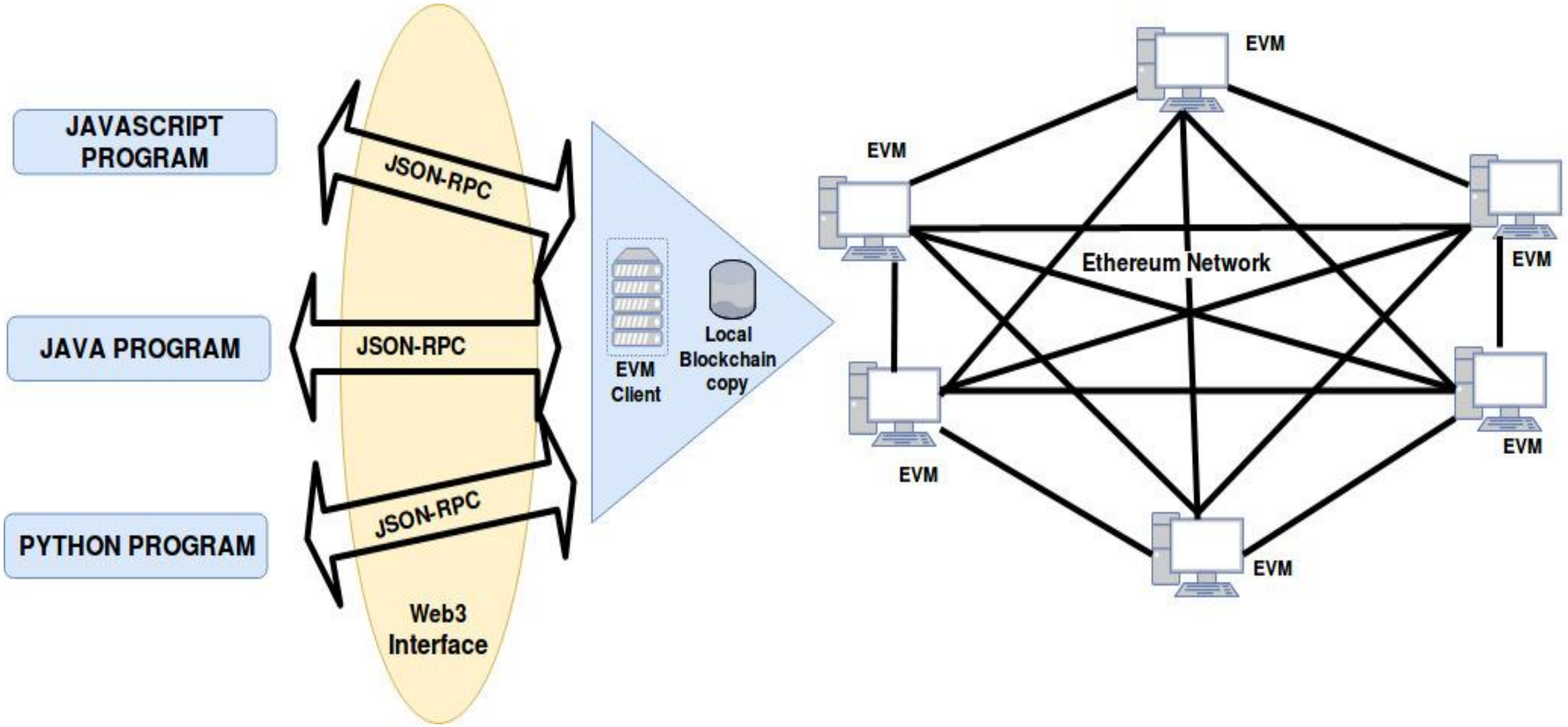
Introduction

- Web3 is a library that can be used to communicate with an Ethereum node via RPC communication.

- Web3 works by exposing methods that have been enabled over RPC.

- This allows the development of user interfaces that make use of the web3 library in order to interact with the contracts deployed over the blockchain.

Web3


JAVASCRIPT PROGRAM
JAVA PROGRAM
PYTHON PROGRAM
JSON-RPC
JSON-RPC
JSON-RPC
Web3 Interface
EVM Client
Local Blockchain copy
Ethereum Network
EVM
EVM
EVM
EVM
EVM
EVM
EVM

Web3.js

- Official Ethereum JavaScript API

- Wrapper over JSON RPC functionality (available in go-ethereum-geth, parity, cpp-ethereum, pyethapp)

- Other API/clients

★web3j ---------->Java	https://github.com/web3j/web3j
★Nethereum----------->C#.NET	https://github.com/Nethereum/Nethereum
★Ethereum-ruby----------->Ruby	https://github.com/DigixGlobal/ethereum-ruby

Features of Web3.js

- Managing accounts
- Signing transactions
- Sending of client requests
- Interaction with Ethereum clients over JSON-RPC

Talking to the Blockchain

- Web3 installation
  ```
  $ npm install web3
  ```

- Require web3 module
  ```
  >Web3 = require('web3');
  ```

Talking to the Blockchain

- Web3.js provides the web3 object that enables us to exploit the web3 API functions in Javascript. Therefore, the first action to take to instantiate a web3 object

- The object needs to be connected to an RPC provider to communicate with the Blockchain. We can set local or remote web3 provider using
  var web3 = new Web3(Web3.setProvider(new Web3.providers.HttpProvider('http://RPC-IP:RPC-Port')));

  Here RPC-IP is RPC Provider's IP &
  RPC-Port is its RPC port

Interaction with Smart Contract

- Web3 also provides a Javascript object **web3.eth.Contract**, which represents your deployed contract

- To find and interact with your newly deployed contract on the blockchain, this object needs to know the contracts address and its application binary interface(ABI)

```
var contractABI= web3.eth.Contract("your contract ABI",
"Your contract addrs");
```

Web3.js Important Modules

- Eth -Function: the Eth module for interacting with the Ethereum network.

  Web3.eth

- Net -Function: the Net module for interacting with network properties.
  Web3.eth.net

- Personal -Function: the Personal module for interacting with the Ethereum accounts.

  Web3.eth.personal

Web3.js Important Modules

- Shh - Function: the Shh module for interacting with the whisper protocol
  Web3.shh

- Bzz - Function: the Bzz module for interacting with the swarm network
  Web3.bzz

Some Basic Functions under Web3.eth Module

- **getAccounts**

```
web3.eth.getAccounts([callback])
```

Returns a list of accounts the node controls by using the provider and calling the RPC method eth_accounts

- **getBalance**

```
web3.eth.getBalance(address [, defaultBlock] [, callback])
```

Get the balance of an address

Some Basic Functions under Web3.eth Module

- **web3.eth.Contract**

  The **web3.eth.Contract** object makes it easy to interact with smart contracts on the ethereum blockchain.

  ```
  new web3.eth.Contract(jsonInterface, address, options)
  ```

  Creates a new contract instance with all its methods and events defined in its json interface object.

Some Basic Functions under Web3.eth Module

- **methods.myMethod.send**

```
myContract.methods.myMethod([param1[, param2[, ...]]]).send(options[, callback])
```

Will send a transaction to the smart contract and execute its method. Note this can alter the smart contract state.

Here `myContract` ===> Contract Instance

Some Basic Functions under Web3.eth Module

- **methods.myMethod.call**

```
myContract.methods.myMethod([param1[, param2[, ...]]]).call(transactionObject,
blockNumber, callback])
```

Will call a “constant” method and execute its smart contract method in the EVM without sending any transaction. Note calling can not alter the smart contract state.

Here `myContract ===>` Contract Instance

THANK YOU