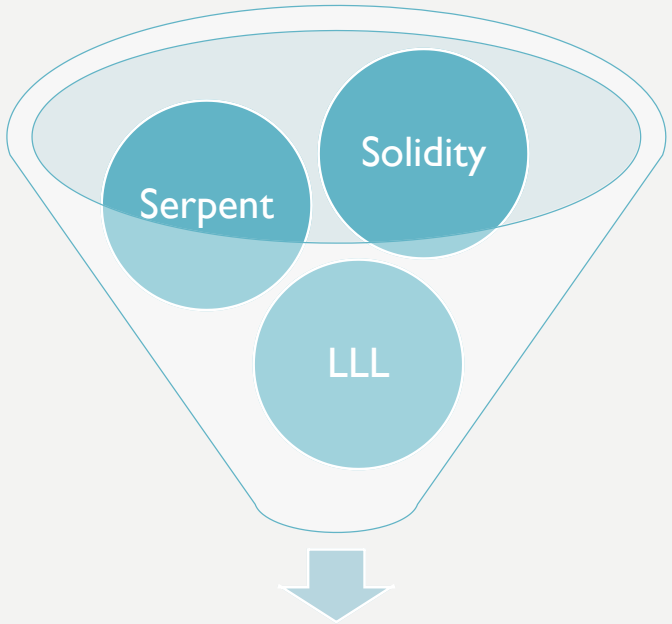
SMART CONTRACTS

THIS IS HOW THEY ARE WORKING

WHAT IS A SMART CONTRACT?

- "smart contracts" was coined by computer scientist Nick Szabo in 1994
- A piece of code running on the blockchain
  - It‘s a state machine
  - Needs transactions to change state
  - Can do logic operations
- Statechange happens through mining through transactions.
- It‘s turing complete

SMART CONTRACT PROGRAMMING LANGUAGES



SOLIDITY

- The most popular language
  - This course is building upon it
- It‘s compared to Javascript
- Every „high language“ code compiles to bytecode (Ethereum Virtual Machine Assembly Code)
- Every ethereum node in the network executes the same code
  - Because every node has a copy of the chain

OTHER LANGUAGES

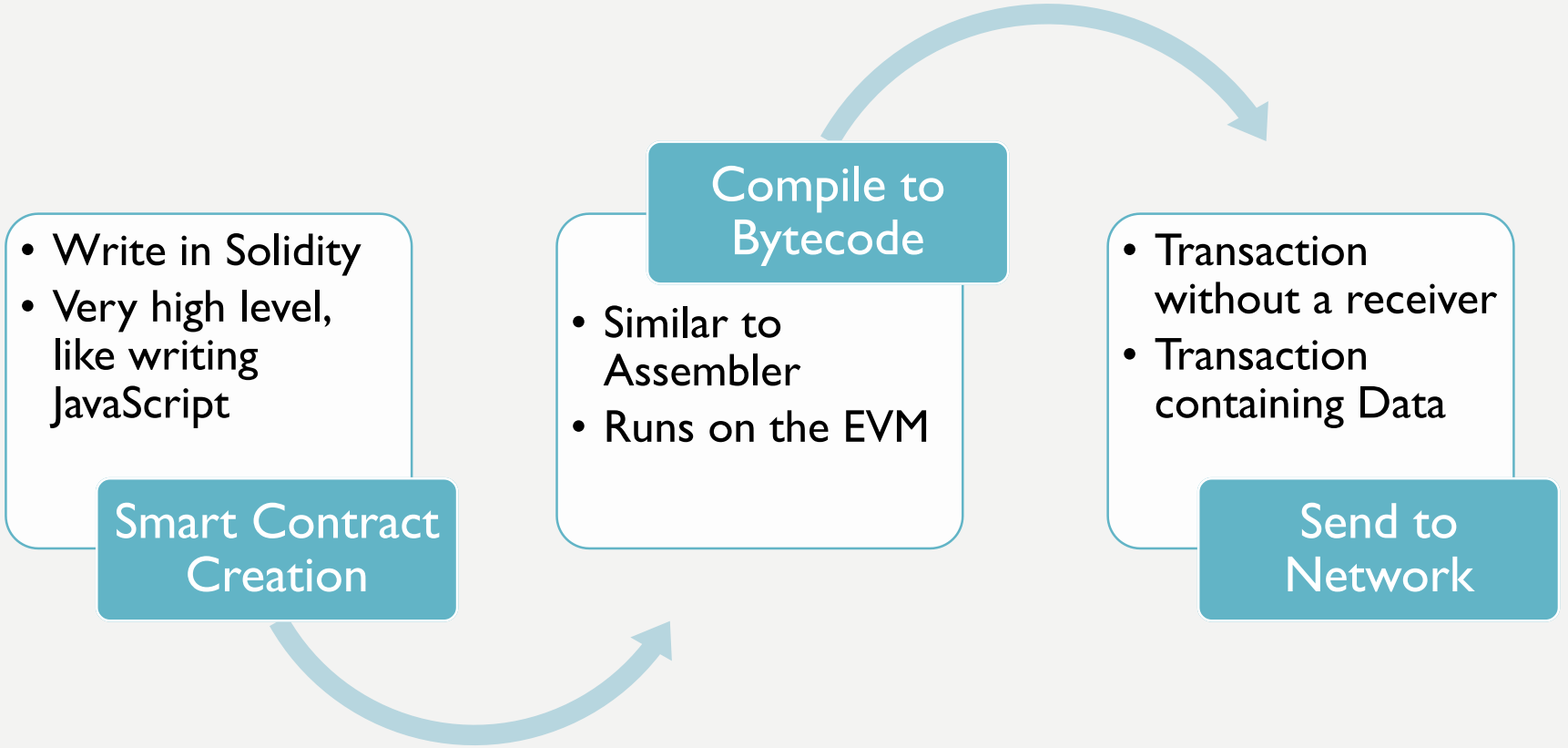
- Serpent
  - Similar to Python
- LLL
  - Like low-level LISP
- Mutan
  - Deprecated Go-based language
- Viper
  - Research-oriented, derived from Python

STRUCTURE OF A SOLIDITY CONTRACT

- „Class“ like Structure
- Contains functions
- Controll structures
  - IF/ELSE
- Loops
  - For/while
- DataTypes
  - (U)Int, Boolean, Array
  - Struct, Mapping, Address
  - No Floats!
- Inhertiable
- Special structures like „modifiers“
- Imports

```
pragma solidity ^0.4.11;


import "./owned.sol";
import "./FixedSupplyToken.sol";


contract Exchange is owned {
    mapping (address => mapping (uint8 => uint)) tokenBalanceForAddress;

    mapping (address => uint) balanceEthForAddress;

    struct Offer {
        uint amount;
        address who;
    }

    struct OrderBook {
        uint higherPrice;
        uint lowerPrice;
        mapping (uint => Offer) offers;
        uint offers_key;
        uint offers_length;
    }

    struct Token {
        mapping (uint => OrderBook) buyBook;
        mapping (uint => OrderBook) sellBook;
        uint curBuyPrice;
        uint lowestBuyPrice;
        uint amountBuyPrices;

        uint curSellPrice;
        uint highestSellPrice;
        uint amountSellPrices;

        address tokenContract;
    }
```

DEPLOYMENT


• Write in Solidity
• Very high level, like writing JavaScript
Smart Contract Creation
Compile to Bytecode
• Similar to Assembler
• Runs on the EVM
• Transaction without a receiver
• Transaction containing Data
Send to Network

INTERACTION

- Low Level, through the „data“ field when sending a transaction
  - The Keccak-256 SHA3 of the given function name including parameters
  - Function `thisFunction(uint variable, uint otherVariable) {…}`
    - Results in `data = sha3('thisFunction(uint,uint)')`
  - We will not do this, there are better ways now
- Client use an “ABI”
  - Application Binary Interface
- Contains all the Functions/Parameters/Return values of the Contract
- Used to Interact

ABI – APPLICATION BINARY INTERFACE

- Is a Json-File
- Contains all the information to interact with the contract

- The Smart Contract on the blockchain is a binary file
  - Client-Software doesn‘t know the inferfaces
  - Needs to be told via an external „configuration“ -> ABI


```
[
  {
    "constant": true,
    "inputs": [
      {
        "name": "token",
        "type": "string"
      }
    ],
    "name": "getBuyOrderBook",
    "outputs": [
      {
        "name": "",
        "type": "uint256[]"
      },
      {
        "name": "",
        "type": "uint256[]"
      }
    ],
    "payable": false,
    "type": "function"
  },
```

CHANGE A SMART CONTRACT

- Immutable
- Changes are not possible
- Be careful when programming them!
- TESTING TESTING TESTING!

WHAT YOU LEARNED

- Smart Contracts are running on the blockchain (in EVM Assembly)
- They are executed by every node and are turing complete
- We write them in Solidity, they compile to bytecode

- They have to be tested extensively

- ABI Array is the key to interact with the contracts

QUESTIONS/SUGGESTIONS?

- Head over to the Q&A section
  - We answer there regularly
- Feedback?
  - We love it! ☺
- Disappointed?
  - Shoot us a message!