ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE-2

BLOCKCHAIN AND SMART CONTRACT BASICS

Raja Rizwan Saleem
Lead Blockchain Trainer

Class-05

Solidity compiler

Solidity compiler compiles the source code and generate bytecode and ABI which can be deployed on blockchain

What is ABI

- Application Binary Interface (ABI) as an interface consisting of all external and public function declarations along with their parameters and return types.
- The ABI defines the contract and any caller wanting to invoke any contract function can use the ABI to do so.
- The bytecode is what represents the contract and it is deployed in the Ethereum ecosystem.

What is ABI

- The bytecode is required during deployment and ABI is needed for invoking functions in a contract.
- A new instance of a contract is created using the ABI definition.
- Deploying a contract itself is a transaction.
- A transaction is created for deploying the contract on Ethereum.
- The bytecode and ABI are necessary inputs for deploying a contract.

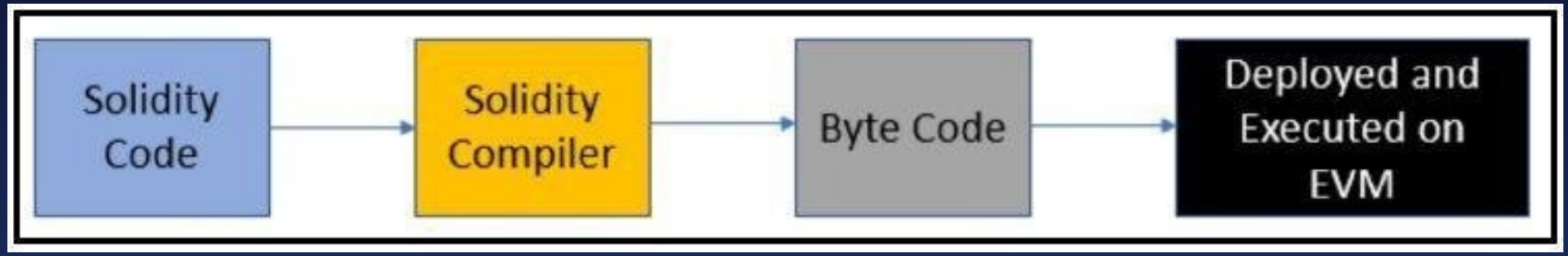
Solidity compiler

1. The code written using Solidity is compiled using a Solidity compiler, which outputs byte code and other artifacts needed for deployment of smart contracts.
2. The Solidity compiler also known as solc can be installed using npm:
    a. npm install -g solc

Ethereum Virtual Machine

1. EVM executes code that is part of smart contracts
2. Smart contracts are written in Solidity
3. EVM does not understand Solidity
4. EVM understands bytecode.
5. Solidity comes with a compiler ‘solc’ which convert Solidity code into bytecode

Ethereum Virtual Machine

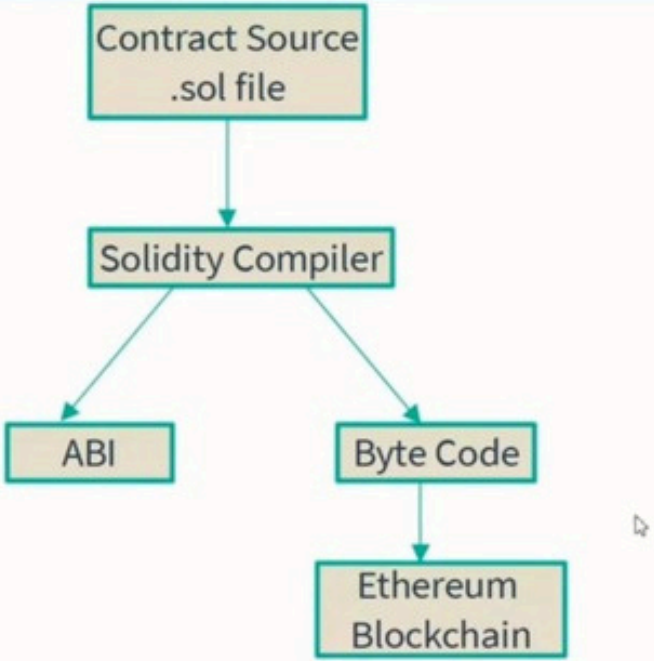


Solidity and Solidity Files

1. Solidity is a programming language that is very close to JavaScript
2. Solidity is a statically-typed, case-sensitive, and object-oriented programming (OOP) language
3. The statement terminator in Solidity is the semicolon: ;
4. Solidity code is written in Solidity files that have the extension .sol and it is human readable text file

Smart Contract Compilation

Smart Contract Compilation



Contract Deployment

Contract Deployment

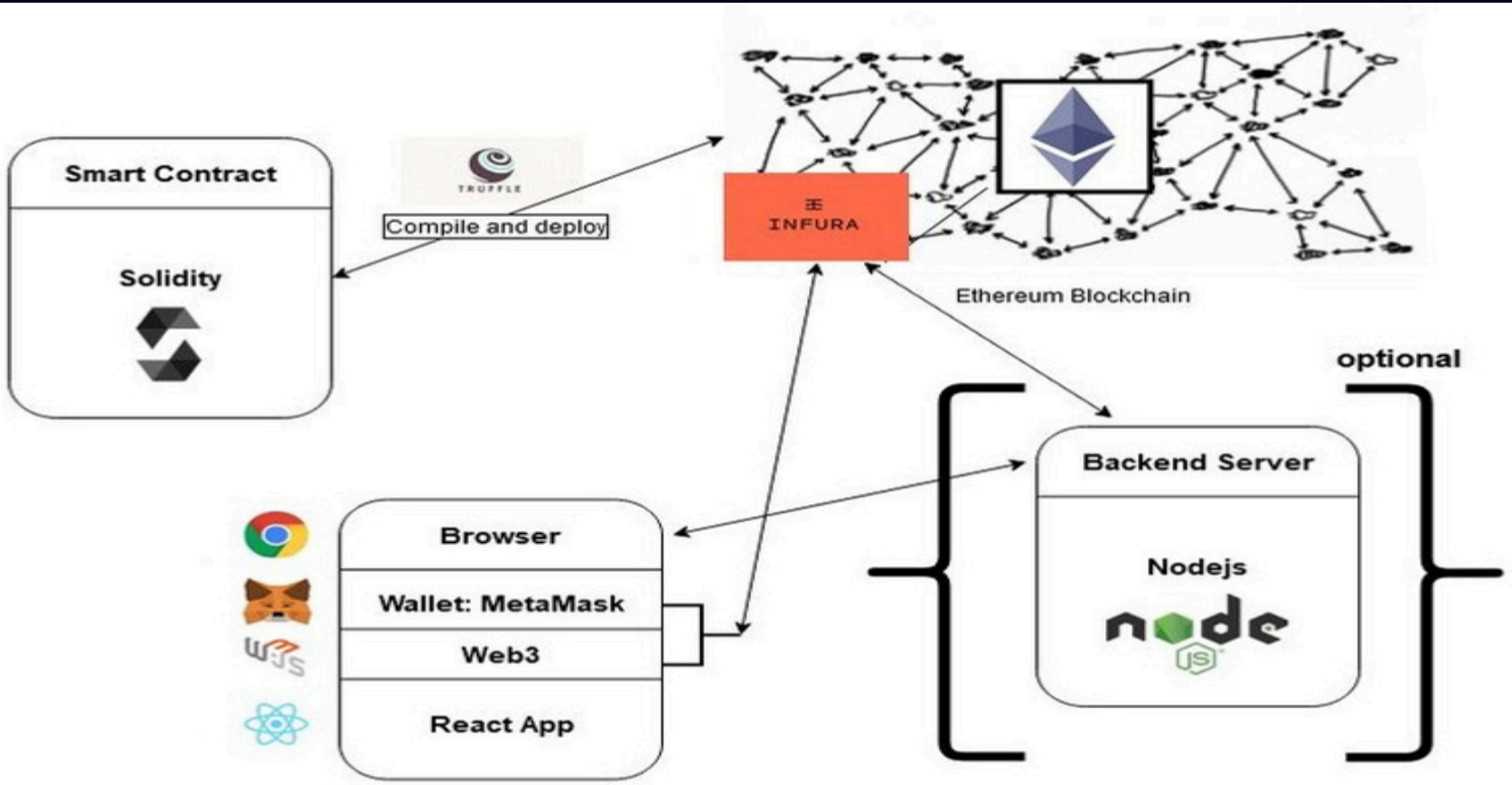
JavaScript Virtual Machine

- Transaction will be executed in a sandbox.
- Own memory blockchain.
- Ideal for testing.

Contract Deployment

Injected Web3

- Deploy a contract or run a transaction on Ethereum main or test net.



Structure of a typical application for Ethereum

www.edversity.com.pk

A Solidity file is composed of four high-level constructs

1. Pragma
2. Comments
3. Import
4. Contracts/library/ interface

What is SPDX license?

All software products have a software license. Think of it this way, most published books have a copyright symbol that either gives you permission to reproduce or not reproduce a book. The same logic appears here. To build trust in a software product, most developers make their source code available on the web. However, the type of license used will determine whether or not the source code can be redistributed, or republished so as to avoid legal issues such as copyright infringement.

Pragma

1. pragma is generally the first line of code within any Solidity file.
2. *pragma* is a directive that specifies the compiler version to be used for current Solidity file.
3. Although it is not mandatory, it is a good practice to declare the pragma directive as the first statement in a Solidity file.

Pragma

1. The syntax for the pragma directive is as follows: pragma solidity <<version number>>;
   pragma solidity ^0.5.14;
2. The version number comprises of two numbers, a **major build** and a **minor build** number. 0.**5**.**14**

3. With the help of the pragma directive, you can choose the compiler version and target your code accordingly

pragma

In Solidity, "pragma" is a keyword used to indicate the version of the Solidity compiler that should be used to compile the contract. It sets the compiler version for the contract and helps ensure that the code is compiled using a specific version of the Solidity compiler.
Here's an example of how pragma is used in Solidity:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MyContract {

    // Contract code goes here

}
```

Contract Syntax

```
Contract name {

// write down code logic here

}
```

Pragma Rules

The ^ character, also known as caret, is optional but plays a significant role in deciding the version number based on 3 rules

Rule 1

The ^ character refers to the latest version within a major version. So, ^0.5.0 refers to the latest version within build number 5, which currently would be 0.5.14.

Rule 2

The ^ character will not target any other major build apart from the one that is provided.

So ^0.5.0 will work for any minor version 0.5.1 or 0.5.11 but will not work for 0.6.0 or 0.4.0

Rule 3

The Solidity file will compile only with a compiler with 5 as the major build. It will not compile with any other major build.

So if compiler is major build 6 it will not work for 5

Pragma

As a good practice, it is better to compile Solidity code with an exact compiler version rather than using ^.

Comments

1. Single-line comments

```
// This is a single-line comment in Solidity
//int public a = 5;
```

2. Multiline comments

```
/* This is a multiline comment In Solidity. Use this when
multiple consecutive
lines Should be commented as a whole */
```

Comments

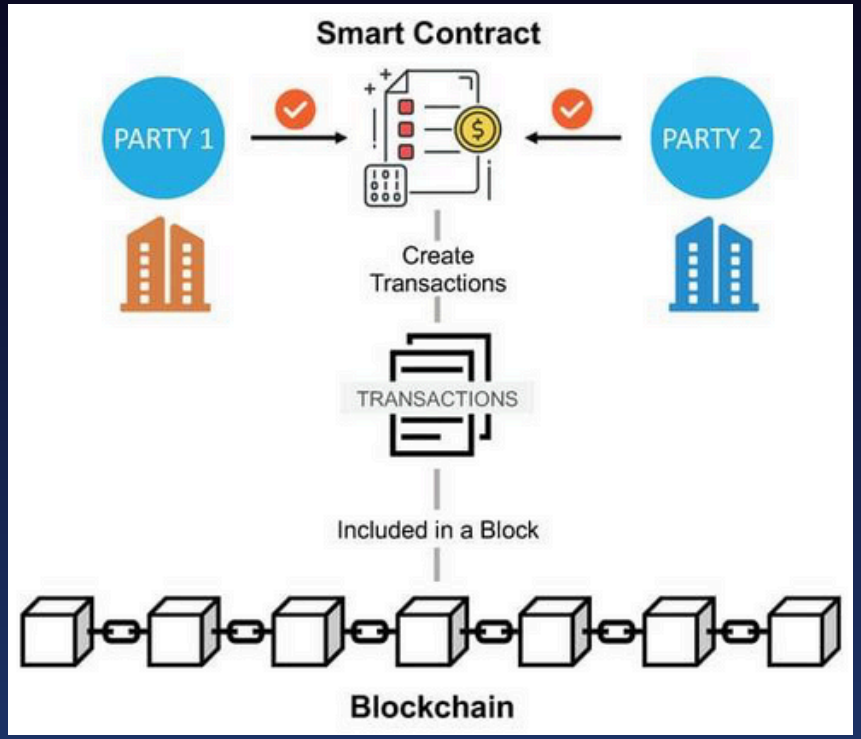
3. Ethereum Natural Specification (Natspec)
4. Natspec has two formats:
    a. / / for single-line and a
    b. combination of /** for beginning and */ for end of multiline comments.

Comments

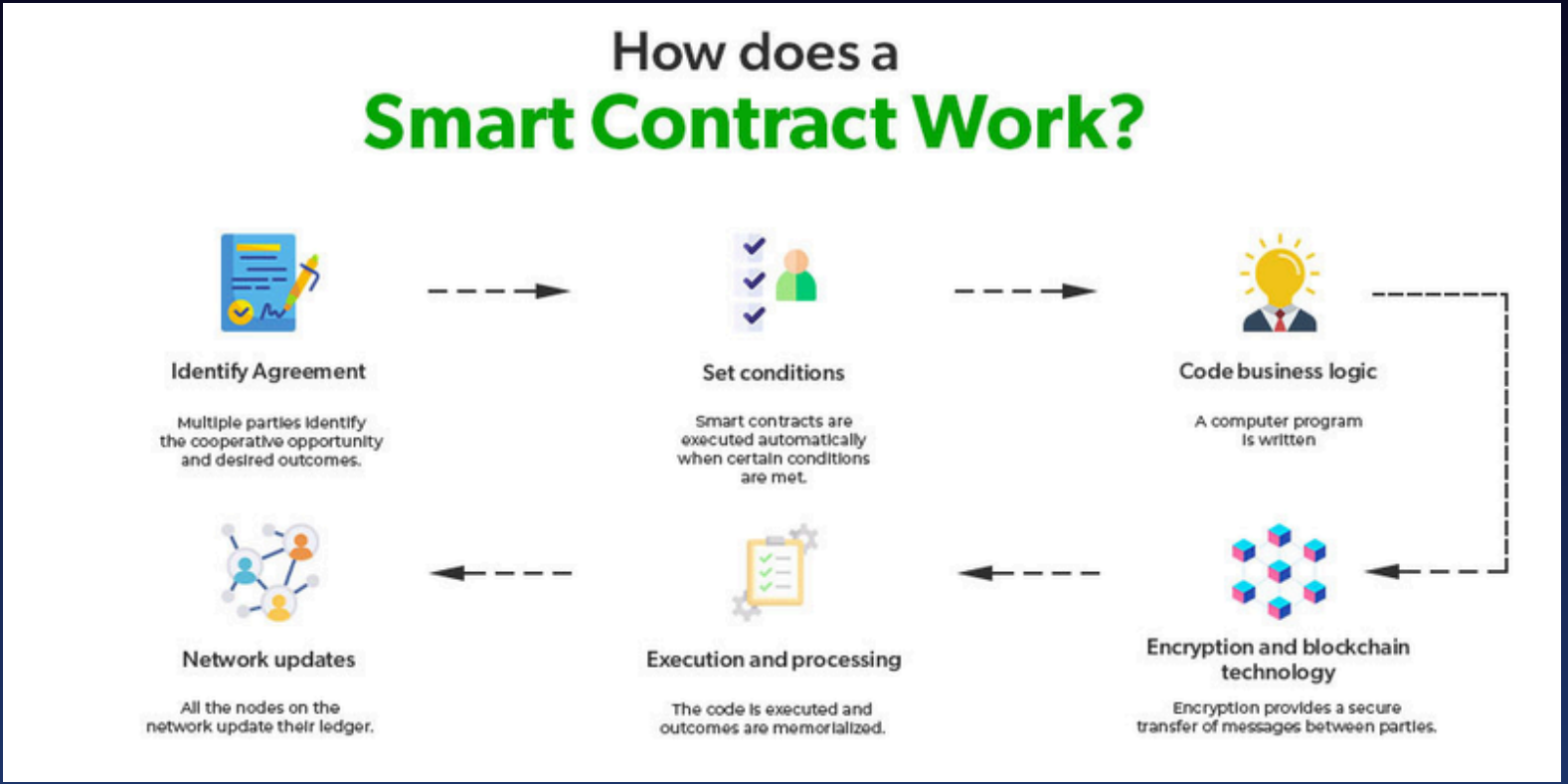
```
pragma solidity ^0.5.6;

/// @title A simulator for trees
/// @author Larry A. Gardner
/// @notice You can use this contract for only the most basic simulation
/// @dev All function calls are currently implemented without side effects
contract Tree {
    /// @author Mary A. Botanist
    /// @notice Calculate tree age in years, rounded up, for live trees
    /// @dev The Alexandr N. Tetearing algorithm could increase precision
    /// @param rings The number of rings from dendrochronological sample
    /// @return age in years, rounded up for partial years
    function age(uint256 rings) external pure returns (uint256) {
        return rings + 1;
    }
}
```

Solidity file Components

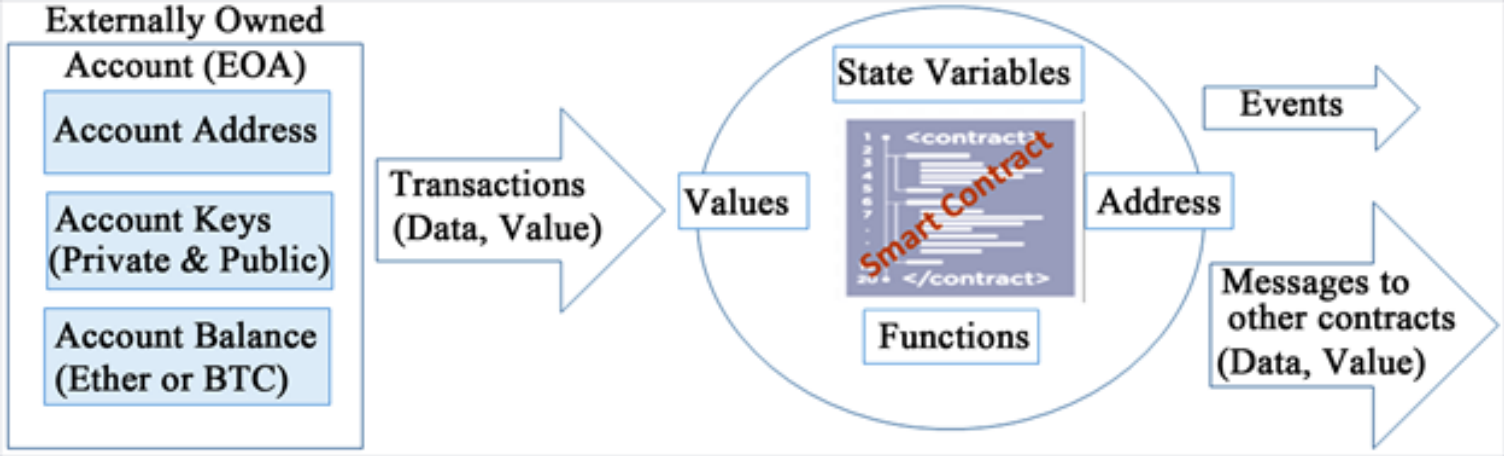
Structure of a Smart Contract



How Does a Smart Contract Works



How Does a Smart Contract Works



THANK-YOU