

Smart Contracts and Solidity Basics

Prof. (Dr.) Neeraj Kumar



Prerequisite

- Blockchain
- Ethereum
- Smart contracts

Blockchain

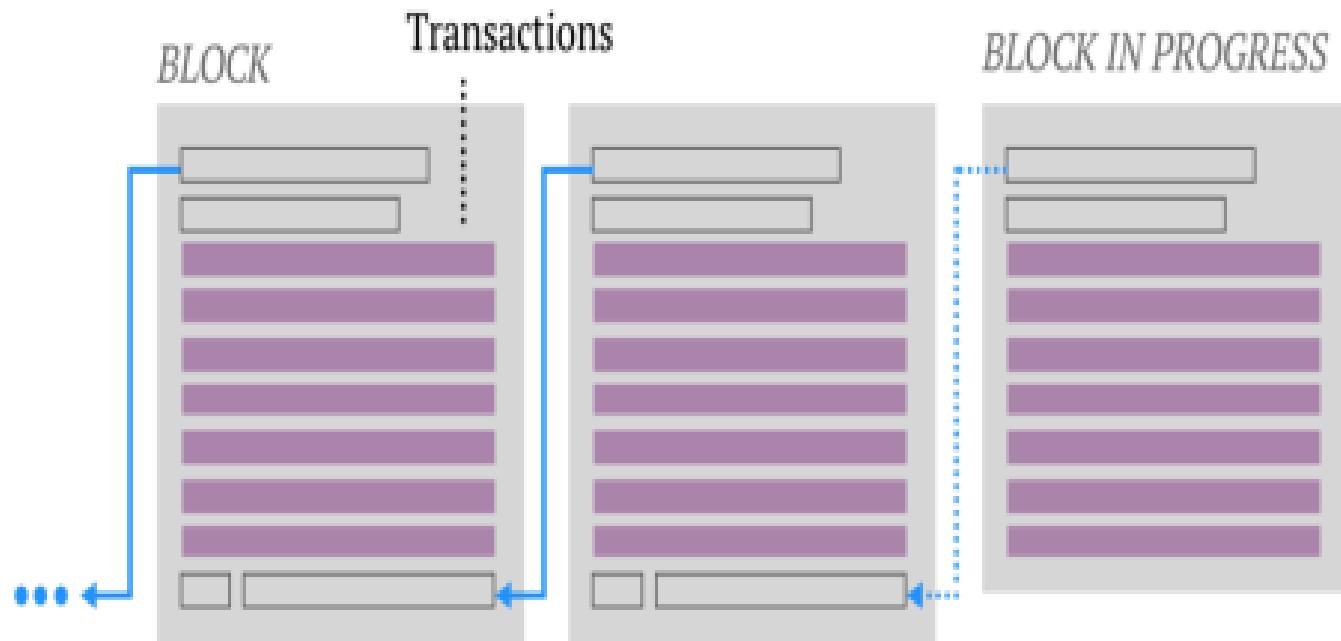
Fully Distributed Database like BTC

Advantages:

- Highly Secure
- Transparent
- Immutable

Disadvantages:

- Scaling
- Performance



Summarizing Blockchain

- It is a decentralized distributed ledger (data structure) where data is being stored inside blocks in form of transactions.
- Remove the dependency on the trusted third party for recording the data in Blocks.
- Since each block is built on top of previous Block immutability has been achieved.
- Very difficult to add a fake block & very easy to detect the fake Block.
- Every participants of the Blockchain contain the almost same copy of the Blockchain.

Blockchain Main Themes

- Longitudinal immutability
- Reliable timestamps
- Cryptographic security
- Fully distributed replication
- Publicly verifiable code governing all transactions
- Value in Smart Contract Assets

What is Ethereum?

- ❖ “A planetary scale computer built on blockchain technology”
- ❖ Open for all to use; zero infrastructure
- ❖ Turing-Complete: Build arbitrarily complex applications
- ❖ “Ethereum is a decentralized platform that runs **smart contracts**: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference.”
- ❖ Ethereum Virtual Machine, runs on **Ether**, Pay per computational step



Ethereum Virtual Machine (EVM)

- ❖ A Turing Complete Virtual Machine accessible from anywhere in the world has a number of benefits for the automated-economy:
- ❖ The Ethereum Blockchain is:
 - ❖ A blockchain with a built-in programming language
 - ❖ A decentralized, massively replicated database where the current state of all accounts is stored
 - ❖ A consensus-based globally executed virtual machine

Source: Melanie Swan; <https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial>

Accounts and Wallets in Ethereum

Accounts:

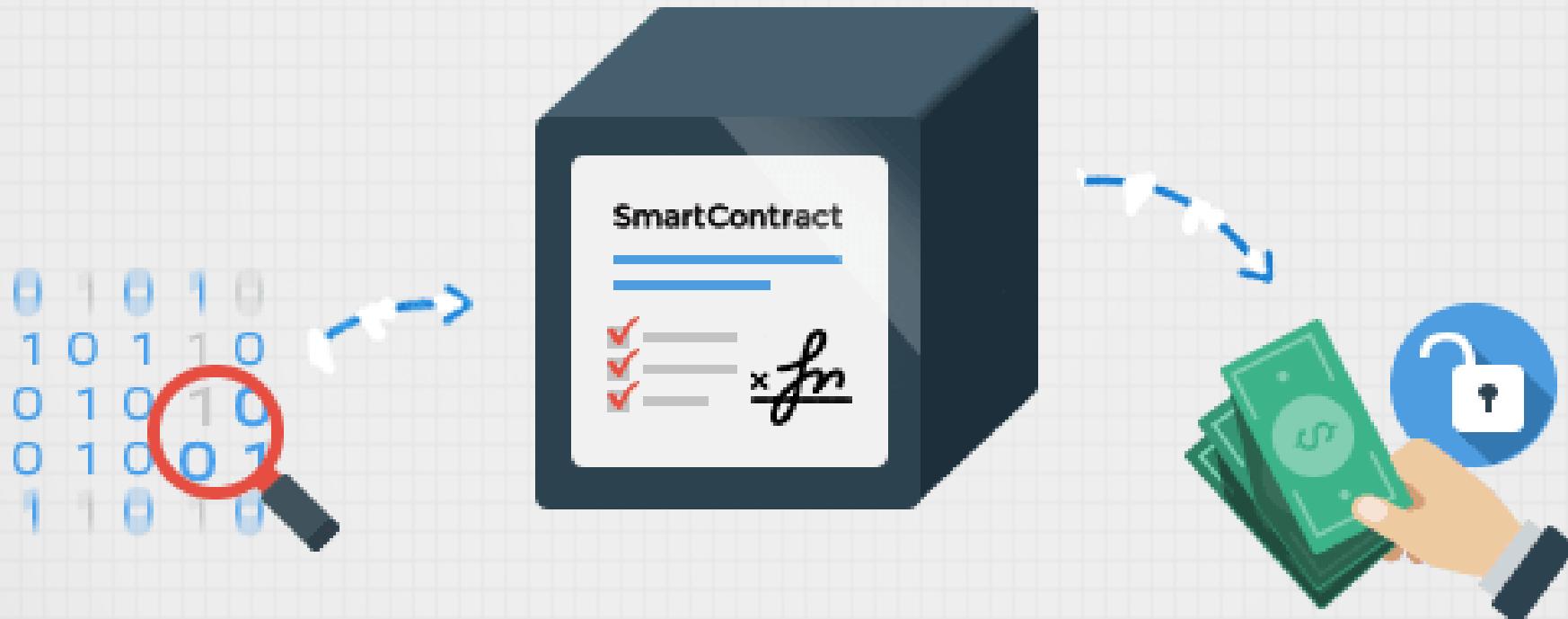
- Two Kinds:
 - External Owned Accounts - (EOA, most common account)
 - Contract Accounts
- Consist of a public/private keypair
- Allow for interaction with the blockchain

Wallets:

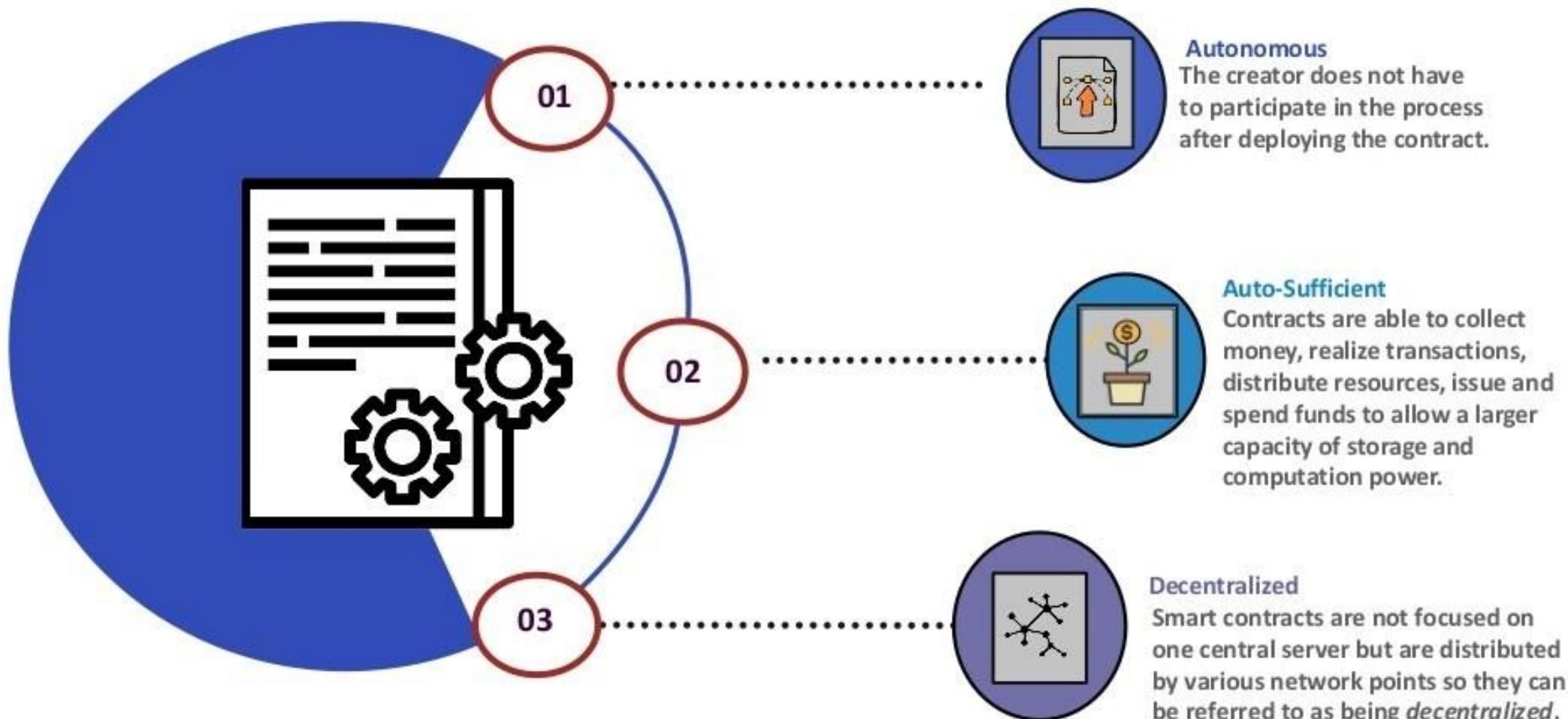
- A set of one or more external accounts
- Used to store/transfer ether

What is a Smart Contract

- Smart contract is a term used to describe computer program code that is capable of facilitating, executing, and enforcing the negotiation or performance of an agreement (i.e. contract) using blockchain technology.
- The entire process is automated can act as a complement, or substitute, for legal contracts, where the terms of the smart contract are recorded in a computer language as a set of instructions.



Key Properties of Smart Contracts



Smart Contracts

- Executable code
- Turing Complete
- Function like an external account
 - Hold funds
 - Can interact with other accounts and smart contracts
 - Contain code
- Can be called through transactions

Code Execution

- Every node contains a virtual machine (similar to Java)
 - Called the Ethereum Virtual Machine (EVM)
 - **Compiles** code from high-level language to bytecode
 - Executes smart contract code and broadcasts state
- *Every full-node on the blockchain processes every transaction and stores the entire state*

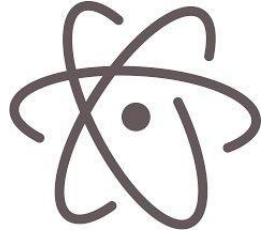
Gas Cost

- Gas Price: current market price of a unit of Gas (in Wei)
 - Check gas price here: <https://ethgasstation.info/>
 - Is always set before a transaction by user
- Gas Limit: maximum amount of Gas user is willing to spend
- Helps to regulate load on network
- Gas Cost (used when sending transactions) is calculated by $\text{gasLimit} * \text{gasPrice}$.
 - All blocks have a Gas Limit (maximum Gas each block can use)

Smart Contract Programming

- Solidity (javascript based), most popular
 - Not yet as functional as other, more mature, programming languages
- Serpent (python based)
- LLL (lisp based)

Smart Contract Programming



[**Atom Ethereum interface**](#) - Plugin for the Atom editor that features syntax highlighting, compilation and a runtime environment (requires backend node).

[**Atom Solidity Linter**](#) - Plugin for the Atom editor that provides Solidity linting.



[**Vim Solidity**](#) - Plugin for the Vim editor providing syntax highlighting.

[**Vim Syntastic**](#) - Plugin for the Vim editor providing compile checking.

B. Smart Contract Programming

- Solidity (javascript based), most popular
 - Not yet as functional as other, more mature, programming languages
- Serpent (python based)
- LLL (lisp based)

B. Smart Contract Programming

Solidity

Solidity is a language similar to JavaScript which allows you to develop contracts and compile to EVM bytecode. It is currently the flagship language of Ethereum and the most popular.

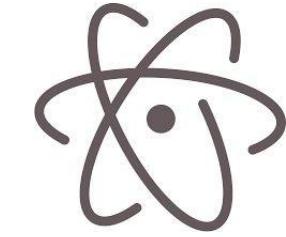
- [Solidity Documentation](#) - Solidity is the flagship Ethereum high level language that is used to write contracts.
- [Solidity online realtime compiler](#)

Serpent

Serpent is a language similar to Python which can be used to develop contracts and compile to EVM bytecode. It is intended to be maximally clean and simple, combining many of the efficiency benefits of a low-level language with ease-of-use in programming style, and at the same time adding special domain-specific features for contract programming. Serpent is compiled using LLL.

- [Serpent on the ethereum wiki](#)
- [Serpent EVM compiler](#)

B. Smart Contract Programming



[**Atom Ethereum interface**](#) - Plugin for the Atom editor that features syntax highlighting, compilation and a runtime environment (requires backend node).

[**Atom Solidity Linter**](#) - Plugin for the Atom editor that provides Solidity linting.



[**Vim Solidity**](#) - Plugin for the Vim editor providing syntax highlighting.

[**Vim Syntastic**](#) - Plugin for the Vim editor providing compile checking.

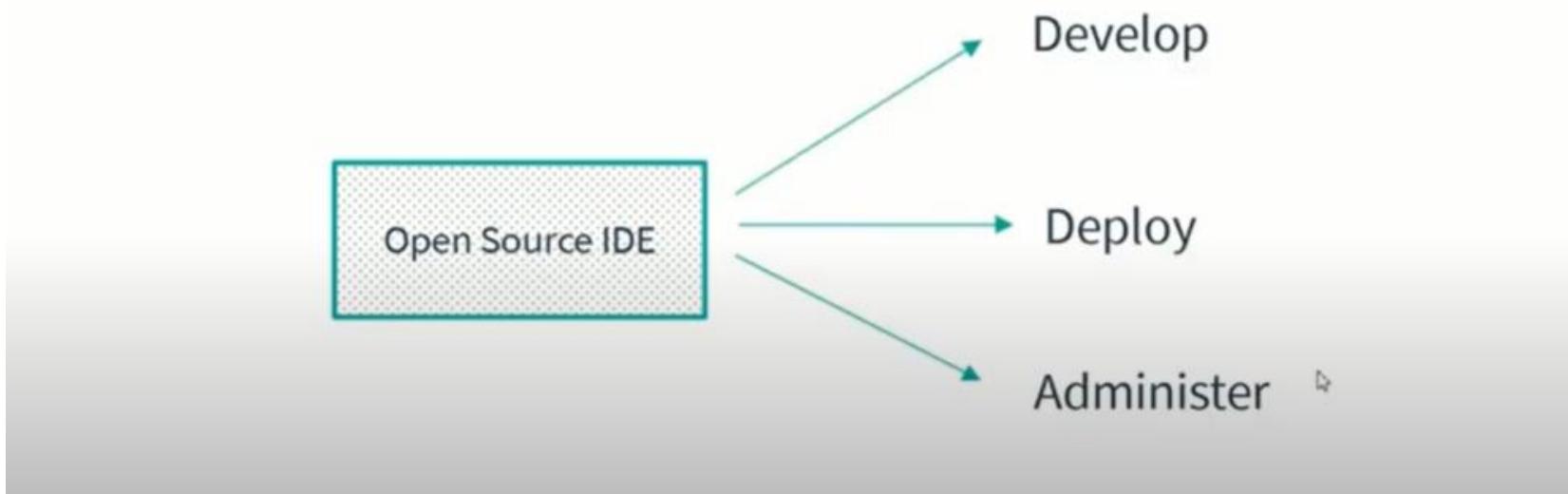
B. Smart Contract Programming: Solidity

```
contract Example {  
  
    uint value;  
  
    function setValue(uint pValue)  
    {   value = pValue;  
    }  
  
    function getValue() returns (uint)  
    {   return value;  
    }  
}
```

B. Smart Contract Programming: Solidity

```
var logIncrement =  
    OtherExample.LogIncrement({sender: userAddress,  
uint value});  
  
logIncrement.watch(function(err, result) {  
    // do something with result  
})
```

Remix IDE



Remix IDE

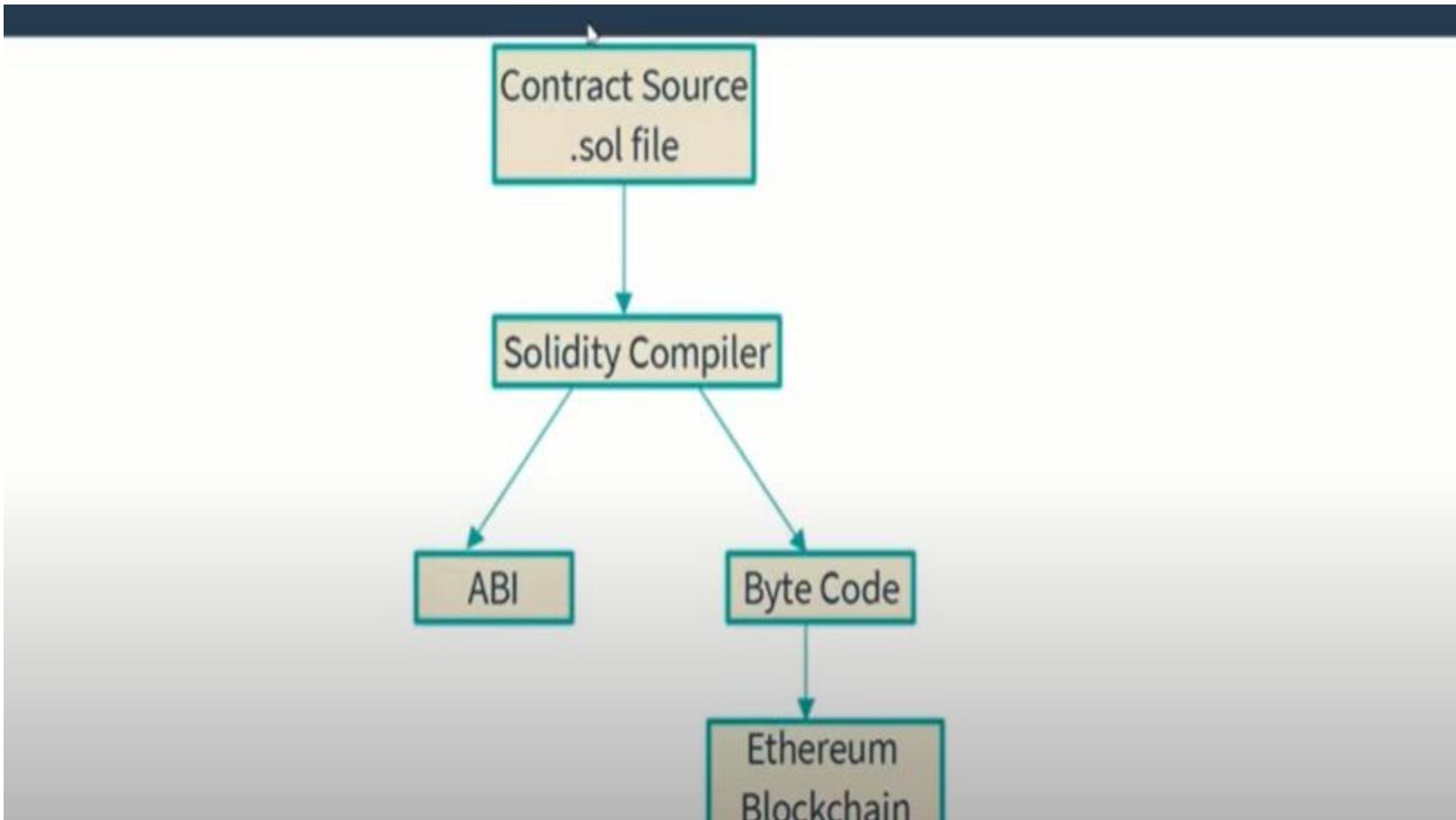
Some important to know about Remix IDE-

- **Language Support -** Solidity and Vyper

- **Written in -** JavaScript

- **Modules -** Testing , Debugging , Deploy

Smart contract compilation



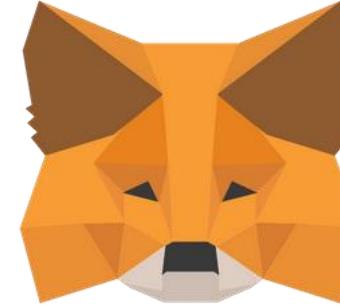
Smart contract compilation

- Contract bytecode is readable in public form
- Contract doesn't have to be public
- Bytecode is immutable
- ABI act as a bridge between application and smart contract
- ABI and bytecode can't be generated without source code

Mainnet vs Testnet

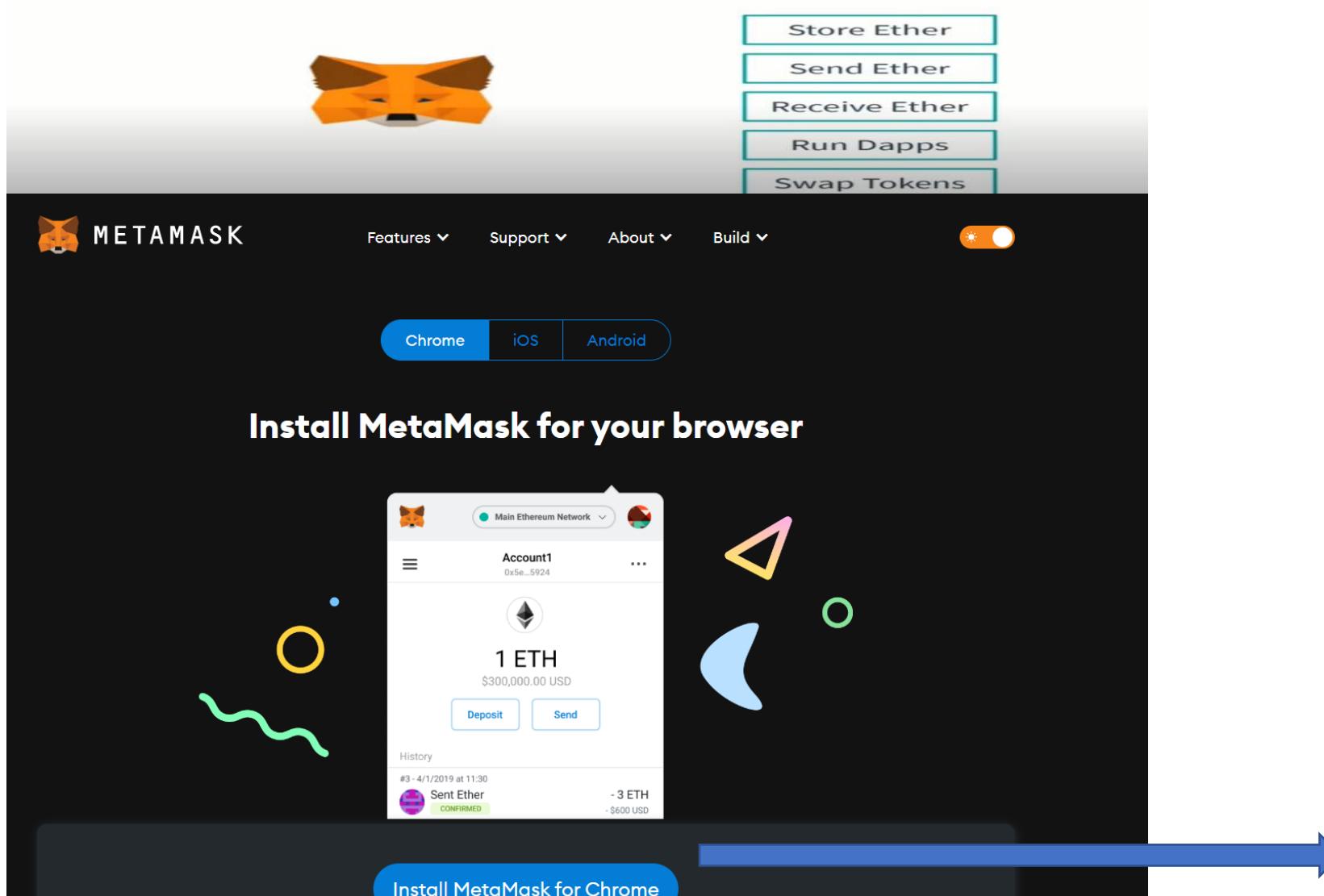
Mainnet	Testnet
Used for actual transaction with value	Used for testing smart contracts and decentralized applications
Mainnet's network ID is 1	Testnets have network IDs of 3, 4, and 42
Example- Ethereum	Example- Rinkeby test networks

Metamask

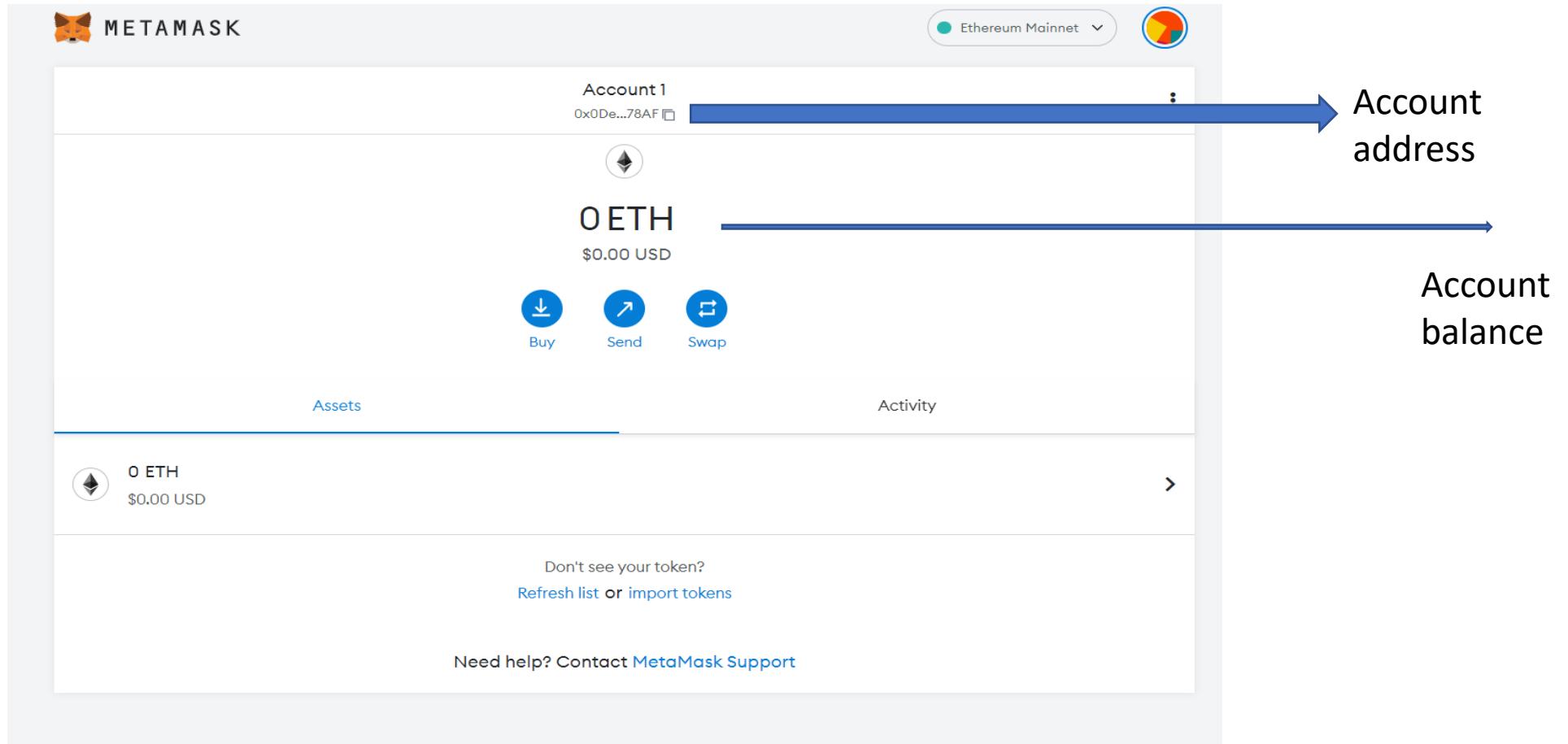


- MetaMask is a lightweight Chrome browser-based extension that helps in interacting with Ethereum networks.
- It is also a wallet that helps in sending and receiving Ether.
- It does not download the entire chaindata locally
- It is a function to inject a javascript library called web3.js into the namespace of each page your browser loads.

Metamask installation



Metamask installation



Testnet (Rinkeby) faucet

- ETH on testnet has no real value; therefore we get testnet ETH from faucet
- Example, Rinkeby faucet, Ropsten faucet

Rinkeby faucet

The screenshot shows a web browser window with the URL faucet.rinkeby.io. The main content area displays the "Rinkeby Authenticated Faucet" page, which lists recent Ethereum address requests from social media accounts. Below this is a section explaining how the faucet works, mentioning that requests are tied to common 3rd party social network accounts. The right side of the screen shows the MetaMask extension interface, indicating "Not connected" and showing an account balance of 0 ETH.

Social network URL containing your Ethereum address...

Rinkeby Authenticated Faucet

User	Ethereum Address	Time Ago
0x8e52bf63cba543592c581cb4bc1d56ad96fb62f4	7 minutes ago	
0x88457d70f920ce8483e71f6eb2c8a5f4ae11ca89	10 minutes ago	
0xc15ec0f15de4b7f53473fdc506d6c723c102882c	11 minutes ago	
0x360d8b25d80dc27d1be65199d4b43e13029e8f71	14 minutes ago	
0xbcd4042de499d14e55001ccb24a551f3b954096	15 minutes ago	
0xc18f1b803d3cde3e8901638976321576be1c31ee	15 minutes ago	
0x6e816319a220e2108ce29cb57b4d921c4881b718	20 minutes ago	

0 peers 11309209 blocks 9.046256971665328e+56 Ethers 803917 funded

How does this work?

This Ether faucet is running on the Rinkeby network. To prevent malicious actors from exhausting all available funds or accumulating enough Ether to mount long running spam attacks, requests are tied to common 3rd party social network accounts. Anyone having a Twitter or Facebook account may request funds within the permitted limits.

To request funds via Twitter, make a [tweet](#) with your Ethereum address pasted into the contents (surrounding text doesn't matter).
Copy-paste the [tweets URL](#) into the above input box and fire away!

To request funds via Facebook, publish a new [public](#) post with your Ethereum address embedded into the content (surrounding text doesn't matter).

Ethereum Mainnet

Account 1
0x0De...78AF

0 ETH
\$0.00 USD

Buy Send Swap

Assets Activity

0 ETH
\$0.00 USD

Don't see your token?
[Refresh list](#) or [import tokens](#)

About solidity

- High level statically typed programming language
- With solidity you can create contracts for uses such as voting, blind auction, crowdfunding, and multi-signature wallets
- Case sensitive
- Latest updates are available on solidity documentation

Variables in solidity

- A variable is basically a placeholder for data which can be manipulated at runtime.
- Syntax:

```
<type> <access modifier> <variable name> ;
```

Example:

```
int public int_var;
```

- A variable can be of three types
 1. State variables
 2. Local variables
 3. Global variables

State variable in solidity

- Permanently stored in contract storage
- Not declared in functions
- Cost gas (expensive)
- Storage not dynamically allowed
- Instance of the contract cannot have other state variable besides these already declared

Local variable in solidity

- Variables whose values are present till function is executing
- Declared inside functions and are kept on stack, not on storage
- Don't cost gas
- There are some types that reference the storage by default

Local vs. state variable

```
pragma solidity ^0.5.0;
contract SolidityTest {
    uint storedData;      // state variable
    constructor() public {
        storedData = 10;   // Using State variable
    }
}
```

```
pragma solidity ^0.5.0;
contract SolidityTest {
    uint storedData; // State variable
    constructor() public {
        storedData = 10;
    }
    function getResult() public view returns(uint){
        uint a = 1; // local variable
        uint b = 2;
        uint result = a + b;
        return result; //access the local variable
    }
}
```

Solidity variable names

- You should not use any of the Solidity reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- Solidity variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one
- Solidity variable names are case-sensitive. For example, Name and name are two different variables.

Solidity-Access modifiers

Access Modifiers are the keywords used to specify the declared accessibility of a function or a type. There are four access modifiers available in Solidity.

- **Public** – Public state variables can be accessed internally as well as via messages. For a public state variable, an automatic getter function is generated.
- **Internal** – Internal state variables can be accessed only internally from the current contract or contract deriving from it without using this.
- **Private** – Private state variables can be accessed only internally from the current contract they are defined not in the derived contract from it.
- **External** - The External element can't be inherited but it can be accessed by external elements. Current contract instance can't access external element, it can be accessed externally only.

Example

```
pragma solidity ^0.5.0;
contract C {
    uint public data = 30;
    uint internal iData= 10;

    function x() public returns (uint) {
        data = 3; // internal access
        return data;
    }
}
contract Caller {
    C c = new C();
    function f() public view returns (uint) {
        return c.data(); //external access
    }
}
contract D is C {
    function y() public returns (uint) {
        iData = 3; // internal access
        return iData;
    }
    function getResult() public view returns(uint){
        uint a = 1; // local variable
        uint b = 2;
        uint result = a + b;
        return storedData; //access the state variable
    }
}
```

Functions in solidity

- A function is a group of reusable code which can be called anywhere in your program
- Syntax:
- Example:

```
function function-name(parameter-list) scope returns() {  
    //statements  
}
```

```
pragma solidity ^0.5.0;  
  
contract Test {  
    function getResult() public view returns(uint){  
        uint a = 1; // local variable  
        uint b = 2;  
        uint result = a + b;  
        return result;  
    }  
}
```

Solidity- View vs Pure functions

- **View** function can be declared view in which case they promise not to modify the state. they can view the state variable but can't modify it
- **Pure** function declares that no state variable will be changed or read
- Both of these don't cost any gas to call if they're called externally from outside the contract (but they do cost gas if called internally by another function).

Functions-Start Coding

- Setter and Getter: Set and get the information.

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4
5     uint balance = 313000;           ← Variable
6
7     function getBalance() public view returns(uint){ ← Getter function
8         return balance;
9     }
10
11    function deposit(uint newDeposit) public{ ← Setter function
12        balance = balance + newDeposit;
13    }
14
15 }
```

Compile the Contract

- Compile tab: Start to compile button

The screenshot shows a web-based development environment for Solidity contracts. On the left, there is a code editor window titled "browser/firstContract.sol" containing the following Solidity code:

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4     uint balance = 313000;
5
6     function getBalance() public view returns(uint){
7         return balance;
8     }
9
10    function deposit(uint newDeposit) public{
11        balance = balance + newDeposit;
12    }
13
14 }
15 }
```

On the right, there is a "Compile" tab with the following interface elements:

- A status message: "Current version:0.5.1+commit.c8a2cb62.Emscripten clang".
- A dropdown menu: "Select new compiler version".
- Checkboxes: "Auto compile" (checked), "Enable Optimization" (unchecked), and "Hide warnings" (unchecked).
- A prominent red box highlights the "Start to compile (Ctrl-S)" button.
- A dropdown menu for selecting the contract: "financialContract".
- A small "Swarm" icon.

Set Deployment Parameters (1/2)

- Run tab: Environment = JavaScript VM

The screenshot shows a web-based Ethereum development environment. On the left, there is a code editor window titled "browser/firstContract.sol" containing the following Solidity code:

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4
5     uint balance = 313000;
6
7     function getBalance() public view returns(uint){
8         return balance;
9     }
10
11    function deposit(uint newDeposit) public{
12        balance = balance + newDeposit;
13    }
14
15 }
16
```

On the right, there is a "Run" tab interface with the following settings:

- Environment:** JavaScript VM (highlighted with a red box)
- Account:** 0xca3...a733c (100 ether)
- Gas limit:** 3000000
- Value:** 0 wei

Below these settings, there is a deployment section with the following options:

- Contract Name:** financialContract
- Deploy** button
- or**
- At Address:** Load contract from Address

Set Deployment Parameters (2/2)

- JavaScript VM: All the transactions will be executed in a sandbox blockchain in the browser. Nothing will be persisted and a page reload will restart a new blockchain from scratch, the old one will not be saved.
- Injected Provider: Remix will connect to an injected web3 provider. Mist and Metamask are example of providers that inject web3, thus they can be used with this option.
- Web3 Provider: Remix will connect to a remote node. You will need to provide the URL address to the selected provider: geth, parity or any Ethereum client.
- Gas Limit: The maximum amount of gas that can be set for all the instructions of a contract.
- Value: Input some ether with the next created transaction (wei = 10^{-18} of ether).

Deploy the Contract on the Private Blockchain of Remix

- Run tab: Deploy button

The screenshot shows the Remix IDE interface with the following details:

- Code Editor:** The file `browser/firstContract.sol` contains the following Solidity code:

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4     uint balance = 313000;
5
6     function getBalance() public view returns(uint){
7         return balance;
8     }
9
10    function deposit(uint newDeposit) public{
11        balance = balance + newDeposit;
12    }
13 }
14
15 }
```
- Run Tab:** The tabs at the top are `Compile`, **Run**, `Analysis`, `Testing`, `Debugger`, `Settings`, and `Support`. The `Run` tab is selected.
- Environment:** Set to `JavaScript VM`.
- Account:** Account address is `0xca3...a733c (99.999999999998644)`.
- Gas limit:** Set to `3000000`.
- Value:** Set to `0` in `wei` units.
- Deployment:** A dropdown menu is set to `financialContract`. The **Deploy** button is highlighted with a red circle.
- Transactions recorded:** 1 transaction.
- Deployed Contracts:** A list shows `financialContract at 0x692...77b3a (memory)`. Below it, two functions are listed: `deposit` and `getBalance`, both highlighted with red circles.
- Bottom Bar:** Shows tabs for `[2] only remix transactions, script` and `Search transactions`. A note at the bottom says: "Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script."

Interact with the Contract

- Setter = Red Button: Creates transaction
- Getter= Blue Button: Just gives information

The image consists of two screenshots of a blockchain interface, likely Truffle TestRPC, showing interactions with a deployed financial contract.

Screenshot 1: This screenshot shows the initial state of the contract. The "getBalance" button is highlighted with a blue arrow pointing to a callout box labeled "1". The box contains the text: "Press getBalance to see the initial amount". The contract's balance is shown as 0: uint256: 313000.

Screenshot 2: This screenshot shows the state after a deposit has been made. The "deposit" input field now contains the value 12. A red arrow points from this field to a callout box labeled "2", which contains the text: "Input a value and press deposit button to create and confirm the transaction". The contract's balance is now shown as 0: uint256: 313012.

Screenshot 3: This screenshot shows the final state after the deposit has been confirmed. The "getBalance" button is highlighted with a blue arrow pointing to a callout box labeled "3", which contains the text: "Press getBalance again to see the result". The contract's balance remains at 0: uint256: 313012.

Constructor function in solidity

- Executed only once
- You can create only one constructor and this is optimal
- A default constructor is created by the compiler if there is no explicit defined constructor
- A constructor can be either public or internal
- After a constructor code executed, the final code is deployed to blockchain.

Constructor

- Will be called at the creation of the instance of the contract

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4
5     address owner;
6
7     constructor() public{
8         owner = msg.sender;
9     }
10
11    function receiveDeposit() payable public{
12
13    }
14
15    function getBalance() public view returns(uint){
16        return address(this).balance;
17    }
18 }
```

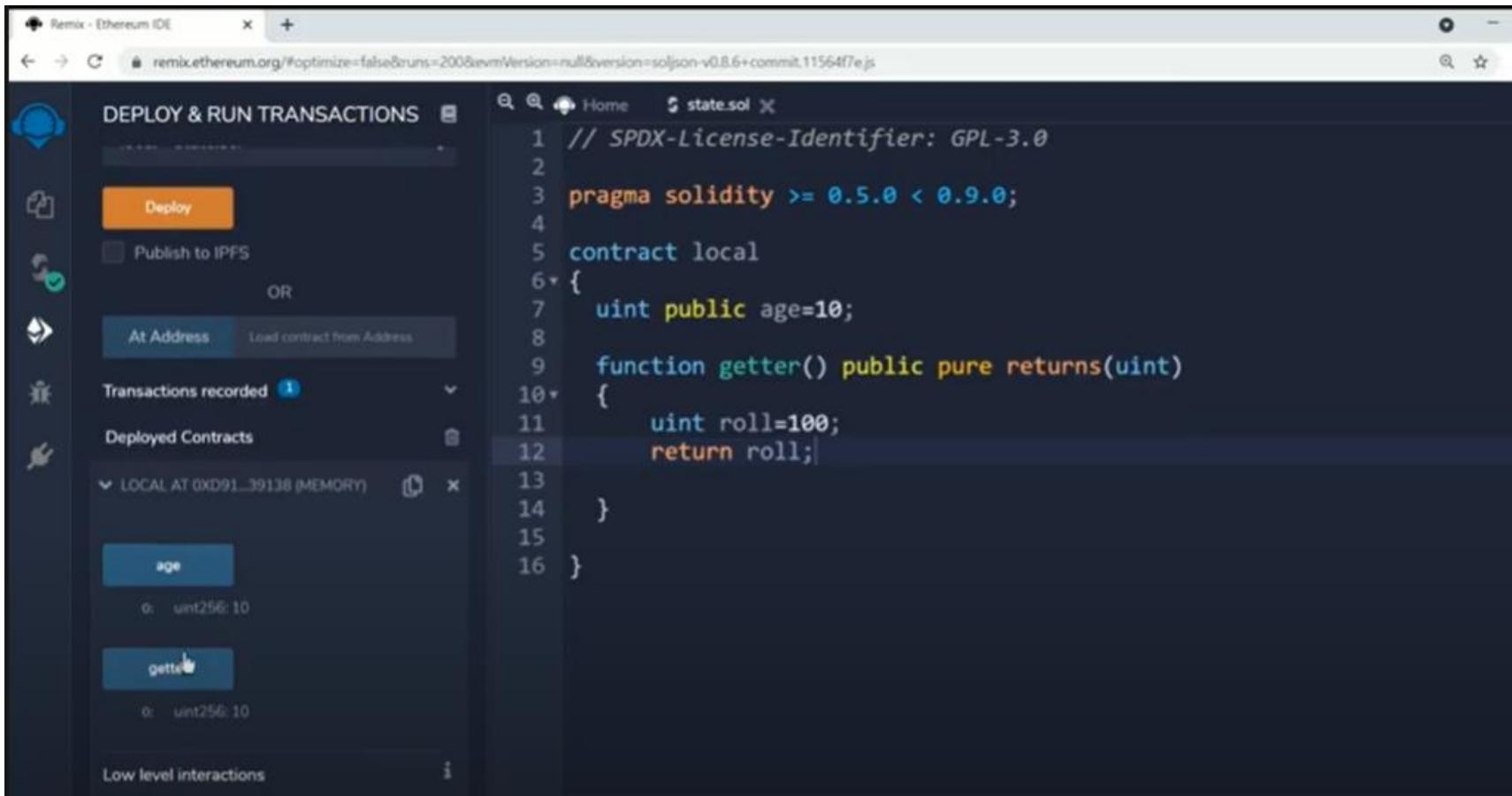
We want to save
the address of the
contract creator

A sample program

The screenshot shows a web-based Ethereum development environment. On the left, a sidebar titled "DEPLOY & RUN TRANSACTIONS" lists "Transactions recorded" (1) and "Deployed Contracts". A deployed contract named "IDENTITY AT 0xD91...39138 (MEMORY)" is expanded, revealing two buttons: "getAge" and "getName". Below this, sections for "Low level interactions" and "CALLDATA" are shown, along with a "Transact" button at the bottom. On the right, the "identity.sol" file is open in a code editor. The code defines a Solidity contract named "Identity" with two state variables: "name" (string) and "age" (uint). It includes a constructor that initializes these variables to "Ravi" and "17" respectively. It also includes two view functions: "getName()" which returns the name, and "getAge()" which returns the age.

```
2 pragma solidity >= 0.5.0 < 0.9.0;
3
4 contract Identity
5 {
6     string name;
7     uint age;
8
9     constructor() public
10    {
11        name="Ravi";
12        age=17;
13    }
14
15    function getName() view public returns(string memory)
16    {
17        return name;
18    }
19
20    function getAge() view public returns(uint)
21    {
22        return age;
23    }
24
25 }
```

Solidity- View vs Pure functions Example



The screenshot shows the Ethereum IDE (Remix) interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, featuring a 'Deploy' button and options to publish to IPFS or deploy at a specific address. The main area displays a Solidity code editor with the file name 'state.sol'. The code defines a local contract with a public variable 'age' set to 10, and a public pure function 'getter()' that returns the value of 'age'. The right side of the interface shows the deployed contract's state, with 'age' set to 10 and a 'getter' function listed.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.5.0 < 0.9.0;
contract local {
    uint public age=10;
    function getter() public pure returns(uint)
    {
        uint roll=100;
        return roll;
    }
}
```

Solidity- View vs Pure functions Example

The screenshot shows the Remix Ethereum IDE interface. On the left, the Solidity Compiler sidebar is visible, showing 'compiler default' selected under 'COMPILER CONFIGURATION'. It includes options for 'Auto compile', 'Enable optimization' (set to 200), and 'Hide warnings'. A prominent blue button labeled 'Compile state.sol' is at the bottom of this sidebar. Below it, a message says 'No Contract Compiled Yet'.

The main workspace displays a Solidity contract named 'state.sol'. The code is as follows:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.5.0 < 0.9.0;
contract local {
    uint public age=10;
    function getter() public pure returns(uint)
    {
        return age;
    }
}
```

A tooltip at the bottom left of the code editor area displays the error message: 'TypeError: Function declared as pure, but this expression (potentially) reads from the environment or state and thus requires "view". --> state.sol:11:14: | 11 | return'. The word 'pure' in the error message is highlighted in red, and the line number 11 is also highlighted.

Receive ether (Payable function)

- Transfer money to the contract

Payable keyword
allows receiving
ether

Hidden Code:
Address(this).balance += msg.value;

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4
5     function receiveDeposit() payable public{
6         }
7
8
9     function getBalance() public view returns(uint){
10        return address(this).balance;
11    }
12 }
```

We can get the
balance of the
contract

Receive ether

1

Input the value as wei
(10^{-18} of ether)



Environment JavaScript VM VM (-)

Account 0xca3...a733c (99.9999999999998944)

Gas limit 3000000

Value 100 wei

financialContract

Deploy

or

At Address

Load contract from Address

Transactions recorded: 1

Deployed Contracts

financialContract at 0x692...77b3a (memory)

receiveDeposit

getBalance

2

Click the receiveDeposit button to transfer the money to the contract



Contract balance update

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing a gas limit of 3,000,000 and a value of 3 ether. The 'CONTRACT' dropdown is set to 'pay - state.sol'. Below it, there are buttons for 'Deploy', 'Publish to IPFS', and 'At Address' (which is selected). The 'Transactions recorded' section shows one transaction. Under 'Deployed Contracts', there is a list item 'PAY AT 0X9EC... (MEMORY)' with a delete button. At the bottom, there are two buttons: 'payEther' and 'sendEtherAcc...'. The main right panel displays the Solidity source code for the 'pay' contract:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 < 0.9.0;

contract pay {
    address payable user=payable(0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c);
    function payEther() public payable
    {
    }
    function getBalance() public view returns(uint)
    {
        return address(this).balance;
    }
    function sendEtherAccount() public
    {
        user.transfer(1 ether);
    }
}
```

Withdraw funds

- Modifier: Conditions you want to test in other functions
- First the modifier will execute, then the invoked function

Only the contract's creator is permitted to withdraw

Transfer some money from the contract's balance to the owner

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4
5     address owner;
6
7 constructor() public{
8     owner = msg.sender;
9 }
10
11 modifier ifOwner(){
12     if(owner != msg.sender){
13         revert();
14     }else{
15         -
16     }
17 }
18
19
20 function receiveDeposit() payable public{
21
22 }
23
24 function getBalance() public view returns(uint){
25     return address(this).balance;
26 }
27
28 function withdraw(uint funds) public ifOwner{
29     msg.sender.transfer(funds);
30 }
31 }
```

Mappings in solidity

- Mappings in solidity acts like a hash table or dictionary in other languages
- Store data in form of a key value pair
- Mostly used to associate the Ethereum address with associated value type
- Syntax:

```
mapping(key => value) <access specifier> <name>;
```

Example-Mappings

```
// Solidity program to
// demonstrate mapping
pragma solidity ^0.4.18;

// Defining contract
contract mapping_example {

    //Defining structure
    struct student
    {
        // Declaring different
        // structure elements
        string name;
        string subject;
        uint8 marks;
    }

    // Creating a mapping
    mapping (
        address => student) result;
    address[] public student_result;
}
```

Now deploying a smart contract on an external blockchain

	Activities	Tools	Remix	Ganache	MyEtherWallet	Geth
1	Configuring the Blockchain		-	-	-	+
2	Deploying the Blockchain	Not Persistent		+	-	+
3	Developing the contract		+	-	-	+
4	Compiling the contract		+	-	-	+
5	Creating user account		+	+	+	+
6	Deploying the contract		+	-	+	+
7	Creating the UI for interacting		+	-	+	+
8	Run the client		+	-	+	+
9	Interact with the contract & have fun		+	-	+	+
10	Monitoring the execution		-	+	-	+

Run Ganache

Ganache

ACCOUNTS BLOCKS TRANSACTIONS LOGS SEARCH FOR BLOCK NUMBERS OR TX HASHES 🔎 ⚙️

CURRENT BLOCK 0 GAS PRICE 20000000000 GAS LIMIT 6721975 NETWORK ID 5777 RPC SERVER HTTP://127.0.0.1:7545 MINING STATUS AUTOMINING 🔄

MNEMONIC ?	HD PATH
slim rain lawn kiwi elegant behind vibrant dentist puppy reduce kidney there	m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX	🔑
0x231eAeEF9EA93F5370a1F633F32E45AF570980E8	100.00 ETH	0	0	🔑
0x970fc818790E900598C57E48b89B6D3D8896D416	100.00 ETH	0	1	🔑
0xb59BD5568d0be42C13fB521f845243F1CDaF2eF1	100.00 ETH	0	2	🔑

MyEtherWallet

- add your custom network that you want to test your contracts on

The screenshot shows the MyEtherWallet interface. On the left, there's a 'Create New Wallet' form with fields for 'Enter a password' and a note 'Do NOT forget to save this!'. A red box highlights this note. Below it is a 'Create New Wallet' button. At the bottom of the form, a note says: 'This password encrypts your private key. This does not act as a seed to generate your keys. You will need this password + your private key to unlock your wallet.' To the right, a red arrow points from the 'Create New Wallet' button to the 'Network ETH' dropdown menu. The dropdown menu lists various Ethereum networks and a 'Add Custom Network / Node' option, which is circled in red.

3.21.05 English Gas Price: 41 Gwei
The network is really full right now. Ch Network ETH (myetherapi.com)

New Wallet Send Ether & Tokens Swap Send Offline Contracts ENS DomainSale Check TX Status View Wallet Info Help

Create New Wallet

Enter a password

Do NOT forget to save this!

Create New Wallet

This password encrypts your private key. This does not act as a seed to generate your keys. You will need this password + your private key to unlock your wallet.

How to Create a Wallet • Getting Started

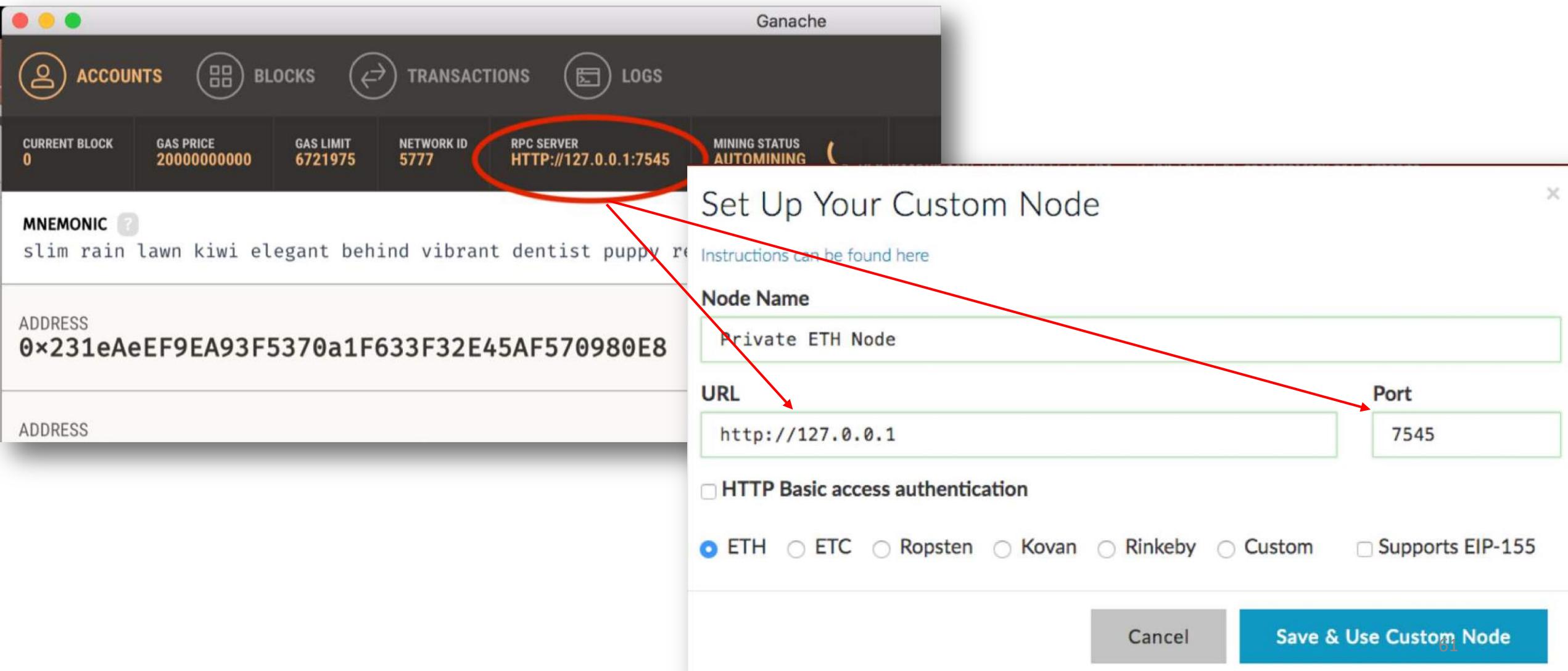
Already have

- Ledger / TREX : Use your hardware wallet to access your wallet.
- MetaMask Chrome Extension . So make sure you're not on a phishy site.
- Jaxx / imToken : Use your hardware wallet to access your wallet.
- Mist / Geth / Parity / Vyper (UTC / JSON) : Use your hardware wallet to access your wallet.

ETH (myetherapi.com)
ETH (etherscan.io)
ETH (infura.io)
ETH (giveth.io)
ETC (Ethereum Commonwealth)
ETC (epool.io)
Ropsten (myetherapi.com)
Ropsten (infura.io)
Kovan (etherscan.io)
Kovan (infura.io)
Rinkeby (etherscan.io)
Rinkeby (infura.io)
EXP (expansetech.com)
UBQ (ubiqscan.io)
POA (core.poa.network)
TOMO (core.tomochain.io)
ELLA (ellism.org)
ETSC (etscentral.com)

Add Custom Network / Node

Import your RPC server address and the port number from Ganache to MyEtherWallet



MyEtherWallet

- Contracts tab: Deploy Contract

The screenshot shows the MyEtherWallet web interface. At the top, there is a dark header bar with the MyEtherWallet logo, version 3.21.05, language settings (English), gas price (41 Gwei), and network selection (My Ether Node:eth). A message at the top right indicates that the network is full. Below the header, a navigation bar includes links for New Wallet, Send Ether & Tokens, Swap, Send Offline, Contracts (which is underlined and circled in red), ENS, DomainSale, Check TX Status, View Wallet Info, and Help.

The main content area features a large button labeled "Interact with Contract or Deploy Contract", with "Deploy Contract" specifically circled in red and an arrow pointing from the "Contracts" link in the header to this button.

Below this button, there are two input fields: "Byte Code" (empty) and "Gas Limit" (set to 300000).

Remix

- Type your contract and compile it

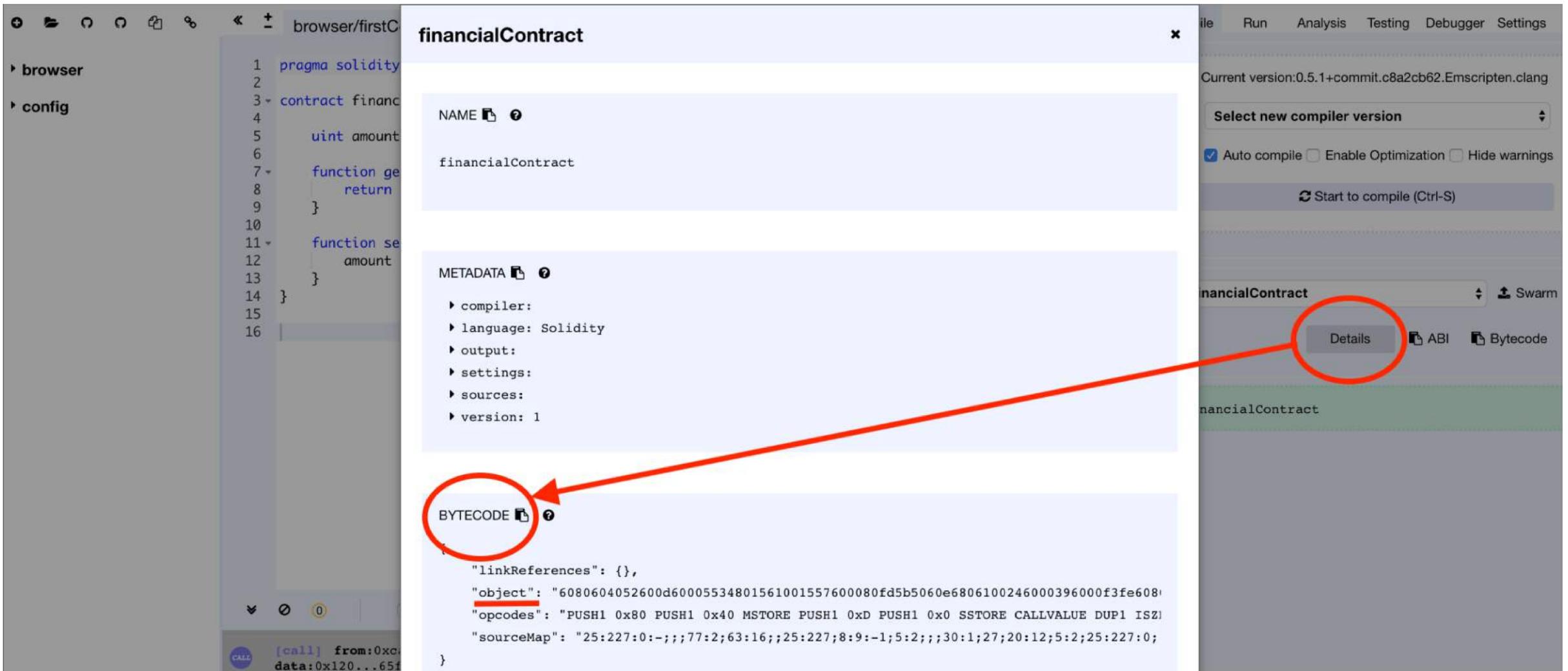
The screenshot shows the Remix IDE interface. On the left, there is a code editor window titled "browser/firstContract.sol" containing the following Solidity code:

```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4     uint amount = 13;
5
6     function getValue() public view returns(uint){
7         return amount;
8     }
9
10    function setValue(uint newAmount) public{
11        amount = newAmount;
12    }
13}
14
15
16
```

On the right, there is a toolbar with various tabs: Compile, Run, Analysis, Testing, Debugger, Settings, and Swarm. Below the toolbar, there is a message: "Current version:0.5.1+commit.c8a2cb62.Emscripten clang". A dropdown menu says "Select new compiler version". There are three checkboxes: "Auto compile" (checked), "Enable Optimization" (unchecked), and "Hide warnings" (unchecked). A button labeled "Start to compile (Ctrl-S)" is highlighted with a red rectangle. At the bottom, there is a dropdown menu set to "financialContract" and buttons for "Details", "ABI", and "Bytecode".

Remix

Click on Details Button: access ByteCode to import it to MyEtherWallet



Ganache

Access your private key for signing your contract in MyEtherWallet.

The screenshot shows the Ganache interface with the following details:

- Accounts:** CURRENT BLOCK 0, GAS PRICE 20000000000, GAS LIMIT 6721975, NETWORK ID 5777, RPC SERVER HTTP://127.0.0.1:7545, MINING STATUS AUTOMINING.
- Mnemonic:** slim rain lawn kiwi elegant behind vibrant dentist puppy reduce kidney there
- HD Path:** m/44'/60'/0'/0/account_index
- Address:** 0x231eAeEF9EA93F5370a1F633F32E45AF570980E8, Balance: 100.00 ETH
- TX Count:** 0, Index: 0 (key icon circled)
- Address:** 0x231eAeEF9EA93F5370a1F633F32E45AF570980E8, Balance: 100.00 ETH
- TX Count:** 0, Index: 1 (key icon circled)
- Address:** 0x970fc818790E900598C
- TX Count:** 0, Index: 2 (key icon circled)
- Address:** 0xb59BD5568d0be42C13f
- Address:** 0x280AFA533B9fa1A97a6
- Address:** 0xD6D30E87AB17c30460E9CAC88475ECcaRF7757c5

A modal window is open for the first address, showing the PRIVATE KEY: a53cf8cb7b66d91ca388ef9ce4e45e39997f2773247c27bb2c7cae35a1b3d383. A red arrow points from the PRIVATE KEY text area to the red circle around the key icon in the TX COUNT 0, INDEX 0 row. Another red arrow points from the PRIVATE KEY text area to the red circle around the key icon in the TX COUNT 1, INDEX 1 row.

MyEtherWallet

1. Paste the contract's ByteCode from Remix

2. Gas Limit will automatically be calculated

3. Paste your private key from Ganache

4. Click Unlock

5. Now you have access to your wallet

The screenshot shows the MyEtherWallet interface with the following fields filled:

- Byte Code:** A long string of hex digits representing the contract's bytecode.
- Gas Limit:** A value set to 124604.
- How would you like to access your wallet?**: A list of options with "Private Key" selected.
- Paste Your Private Key:** A large input field containing a long hex string: `a53cf8cb7b66d91ca388ef9ce4e45e39997f2773247c27bb2c7cae35a1b3d383`.
- Unlock**: A blue button at the bottom.

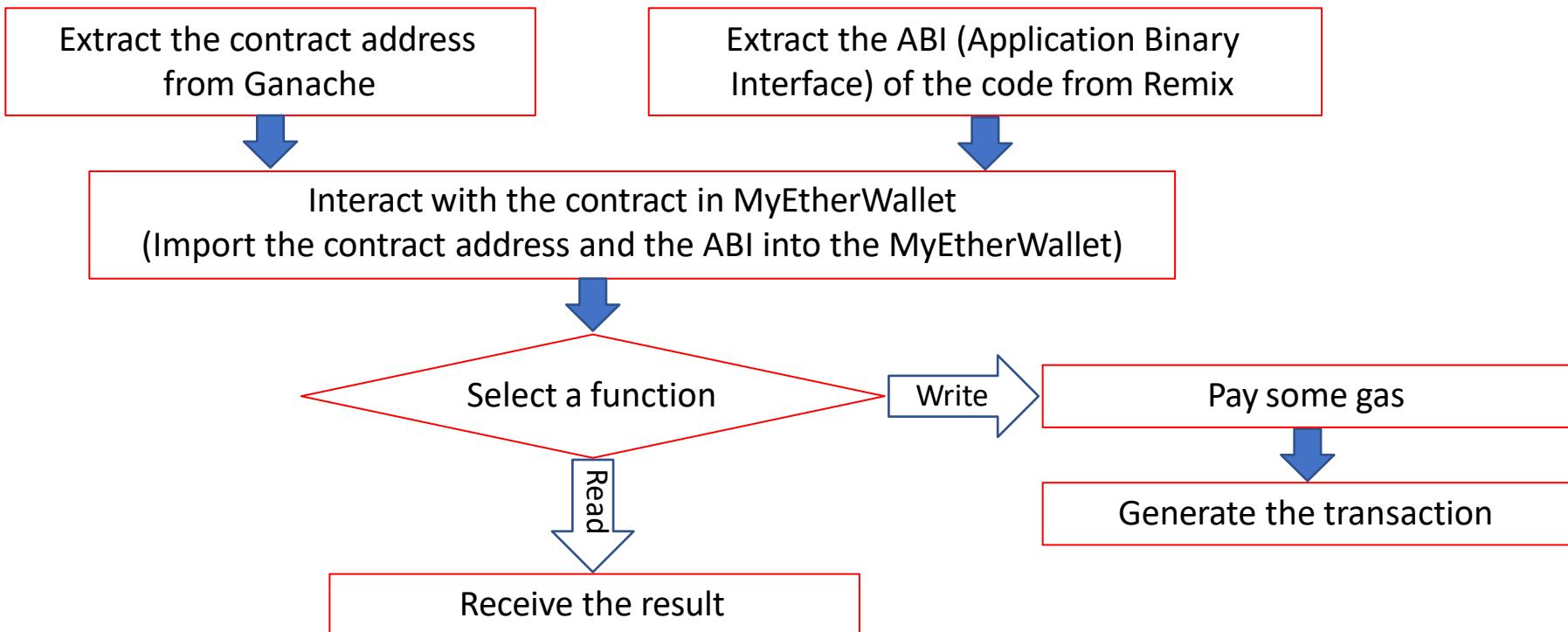
Ganache

You can see now you have one transaction for your address and your balance has been changed because of the amount of gas you paid for creating the contract.

The screenshot shows the Ganache interface with the following details:

- Accounts:** CURRENT BLOCK 1, GAS PRICE 20000000000, GAS LIMIT 6721975, NETWORK ID 5777, RPC SERVER HTTP://127.0.0.1:7545, MINING STATUS AUTOMINING.
- MNEMONIC:** slim rain lawn kiwi elegant behind vibrant dentist puppy reduce kidney there
- HD PATH:** m/44'/60'/0'/0/account_index
- Address 1:** ADDRESS 0x231eAeEF9EA93F5370a1F633F32E45AF570980E8, BALANCE 99.99 ETH, TX COUNT 1, INDEX 0, Key icon.
- Address 2:** ADDRESS 0x970fc818790E900598C57E48b89B6D3D8896D416, BALANCE 100.00 ETH, TX COUNT 0, INDEX 1, Key icon.
- Address 3:** ADDRESS 0xb59BD5568d0be42C13fB521f845243F1CDaF2eF1, BALANCE 100.00 ETH, TX COUNT 0, INDEX 2, Key icon.
- Address 4:** ADDRESS 0x280AFA533B9fa1A97a6D2E4640412FD86FC5dd36, BALANCE 100.00 ETH, TX COUNT 0, INDEX 3, Key icon.
- Address 5:** ADDRESS 0xD6D39E82AB17c30460F2CAc88425ECcaBf2757c5, BALANCE 100.00 ETH, TX COUNT 0, INDEX 4, Key icon.

Interacting with the smart contract



Ganache

Transactions tab: Copy the created contract address

The screenshot shows the Ganache application window. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS (which is highlighted with a red oval), and LOGS. Below the tabs, there are several status indicators: CURRENT BLOCK (1), GAS PRICE (20000000000), GAS LIMIT (6721975), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). A search bar at the top right allows for searching by block number or tx hash. The main area displays a transaction entry. The TX HASH is listed as `0x1e40cc28802d152e810bd9f40bea83d83b1655fc9bace6e801ec6db5fcd84b1a`. The FROM ADDRESS is `0x231eAeEF9EA93F5370a1F633F32E45AF570980E8`. The CREATED CONTRACT ADDRESS is `0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d`, which is also circled in red. To the right of the transaction details, there are buttons for CONTRACT CREATION, GAS USED (124604), and VALUE (0).

TX HASH	FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
<code>0x1e40cc28802d152e810bd9f40bea83d83b1655fc9bace6e801ec6db5fcd84b1a</code>	<code>0x231eAeEF9EA93F5370a1F633F32E45AF570980E8</code>	<code>0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d</code>	124604	0

Remix

Copy the ABI

(ABI is the interface that tells MyEtherWallet how to interact with the contract)



```
1 pragma solidity ^0.5.0;
2
3 contract financialContract {
4     uint amount = 13;
5
6     function getValue() public view returns(uint){
7         return amount;
8     }
9
10    function setValue(uint newAmount) public{
11        amount = newAmount;
12    }
13 }
14
15
16
```

The screenshot shows the Remix IDE interface. On the left, there is a code editor with the file name "browser/firstContract.sol". The code itself is a simple Solidity contract named "financialContract" with two functions: "getValue()" and "setValue(uint newAmount)". On the right side of the interface, there is a toolbar with various tabs: "Compile", "Run", "Analysis", "Testing", "Debugger", "Settings", and "Sup". Below the toolbar, there is a message about the current compiler version: "Current version: 0.5.1+commit.c8a2cb62.Emscripten clang". There is also a dropdown menu for selecting a new compiler version, and several checkboxes for "Auto compile", "Enable Optimization", and "Hide warnings", with "Auto compile" being checked. A large blue button labeled "Start to compile (Ctrl-S)" is present. Below the toolbar, there is a section for the contract "financialContract" with tabs for "Details", "ABI", and "Bytecode", where the "ABI" tab is circled in red. At the bottom, there is a green bar with the same "financialContract" label.

MyEtherWallet

Contracts tab:

Interact with Contract = Paste the contract address from Ganache and the ABI from Remix

New Wallet Send Ether & Tokens Swap Send Offline **Contracts** ENS DomainSale Check TX Status View Wallet Info Help

Interact with Contract or Deploy Contract

Contract Address
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

Select Existing Contract
Select a contract...

ABI / JSON Interface

```
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}
```

Access

MyEtherWallet

You now can interact with the contract by selecting a function and invoking it

New Wallet Send Ether & Tokens Swap Send Offline Contracts ENS DomainSale Check TX Status View Wallet Info Help

Interact with Contract or Deploy Contract

Contract Address
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

Select Existing Contract
Select a contract...

ABI / JSON Interface

```
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}
```

Access

Read / Write Contract
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

Select a function ▾

getValue
setValue

MyEtherWallet

If you select the `getValue` function you will receive the value without paying any gas
(There is no operation cost for getting information)

Read / Write Contract

0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

getValue ▾

↳ uint256

13

MyEtherWallet

If you choose a function that updates the state of the contract, you will need to pay gas for it in a transaction.

Read / Write Contract

0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

setValue ▾

newValue uint256

6|

WRITE



Warning!

You are about to execute a function on contract.
It will be deployed on the following network: ETH (Custom).

Amount to Send *In most cases you should leave this as 0.*

0

Gas Limit

41633

Generate Transaction

Raw Transaction

```
{"nonce": "0x01", "gasPrice": "0x098bca5a00", "gasLimit": "0xa2a1", "to": "0xf22A8cA21D7eeF564FD5Ea743d"}|
```

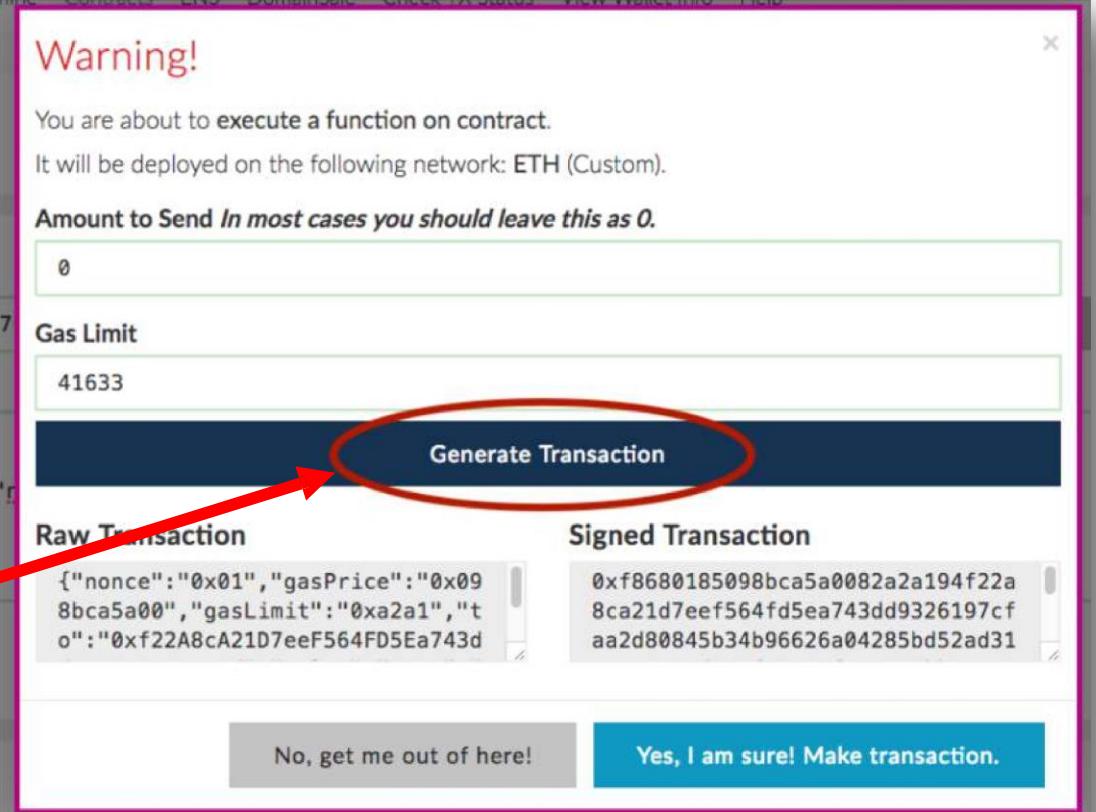
Signed Transaction

0xf8680185098bca5a0082a2a194f22a8ca21d7eeef564fd5ea743dd9326197cfaa2d80845b34b96626a04285bd52ad31|

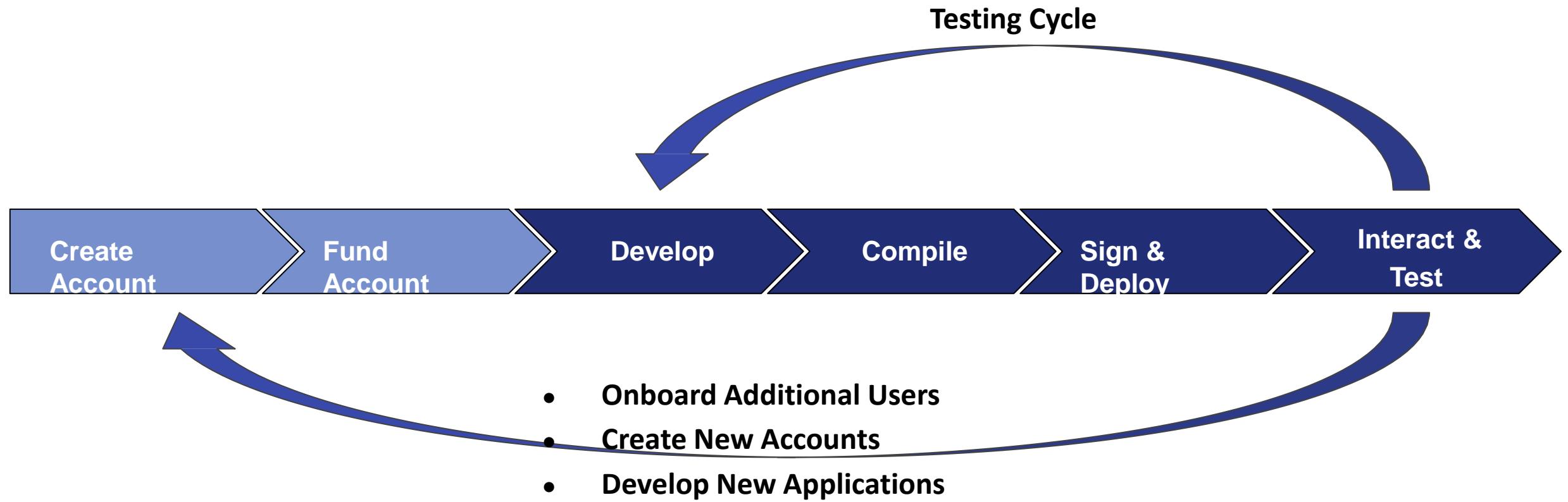
No, get me out of here! Yes, I am sure! Make transaction.

197CFAA2d

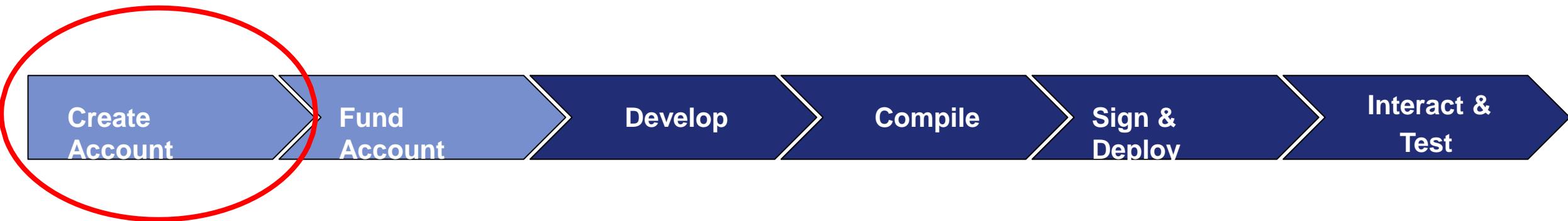
WRITE



C. Development Workflow

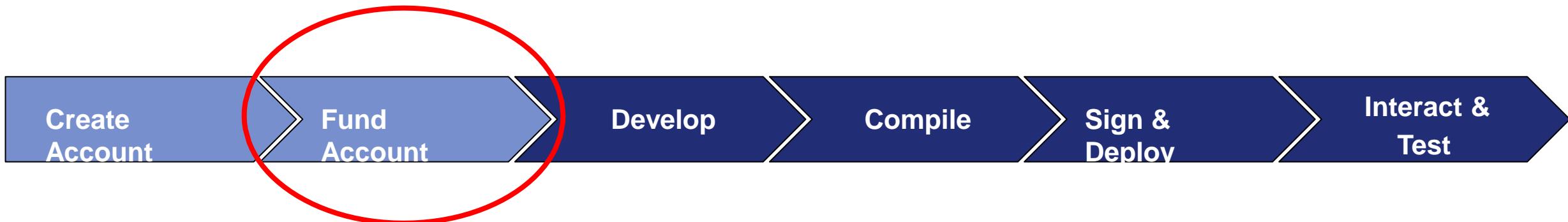


C. Development Workflow: Create Account



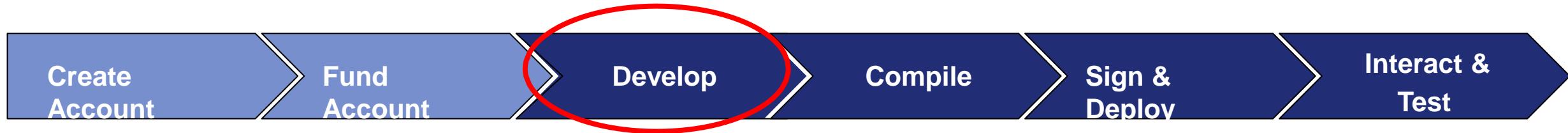
- Programmatically: Go, Python, C++, JavaScript, Haskell
- Tools
 - MyEtherWallet.com
 - MetaMask
 - TestRPC
 - Many other websites

C. Development Workflow: Fund Account



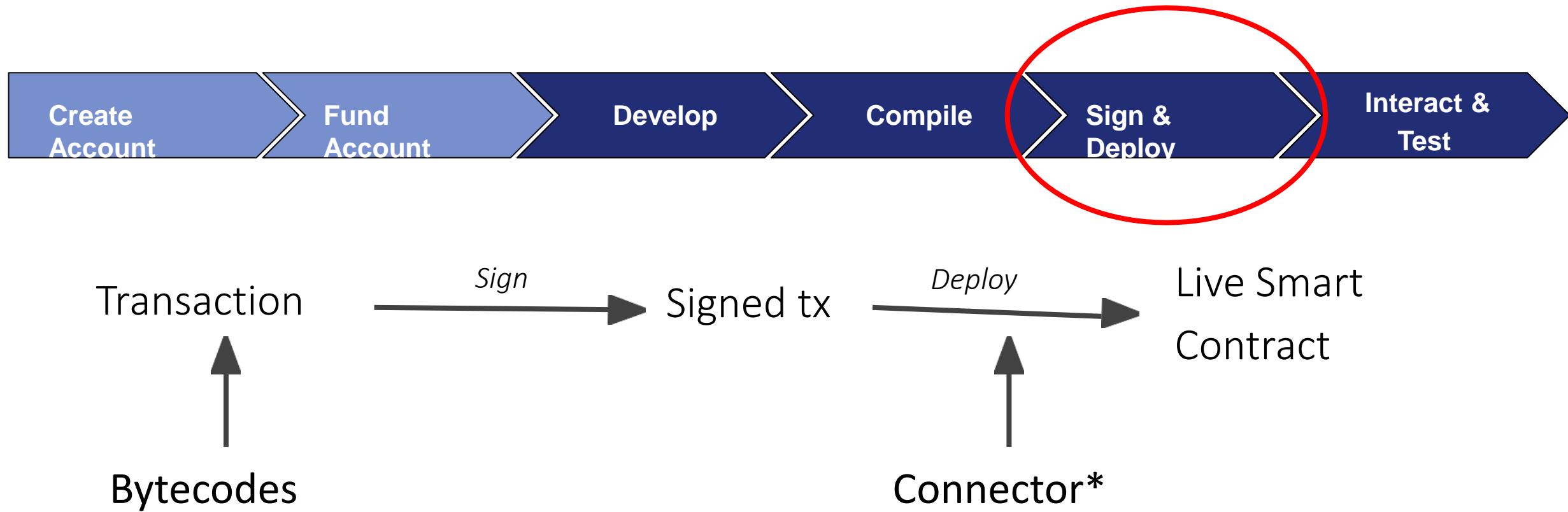
- From friends
- Faucet
- Exchanges (for public blockchain)

C. Development Workflow: Develop



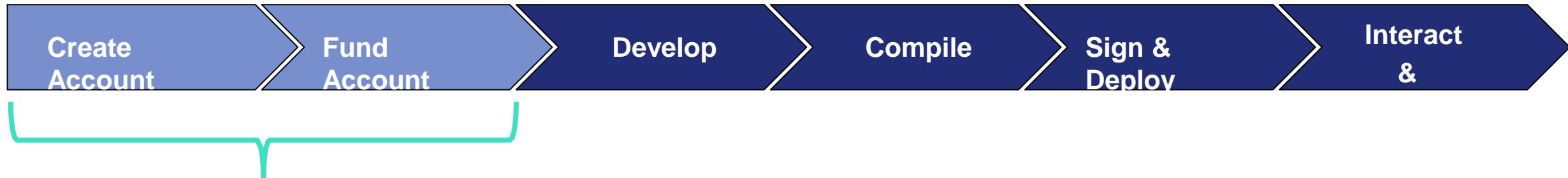
- **Ethereum Application Components:**
 - **Base application:** can be developed in **any** language
 - **Smart contract:** developed in Solidity or one of the other contract compatible languages
 - **Connector library:** facilitates communication between base application and smart contracts (Metamask)

C. Development Workflow: Sign and Deploy



*Library that facilitates communication and connection with Blockchain; Connects your c

C. Development Workflow: TestRPC



TestRPC/TestChain

- Local development or Test Blockchain
- <https://github.com/ethereumjs/testrpc>

Create Custom Ethereum Blockchain

- Instead of using Ganache with its default properties for private blockchain you can run your own blockchain
- Install Geth: One of the implementations of Ethereum written in Go
- Create the genesis block
- Create storage of the blockchain
- Deploy blockchain nodes
- Connect MyEtherWallet to your blockchain to interact with it

Geth help

```
[ds-install:~ mohammht$ geth help
NAME:
    geth - the go-ethereum command line interface

    Copyright 2013-2018 The go-ethereum Authors

USAGE:
    geth [options] command [command options] [arguments...]

VERSION:
    1.8.9-stable

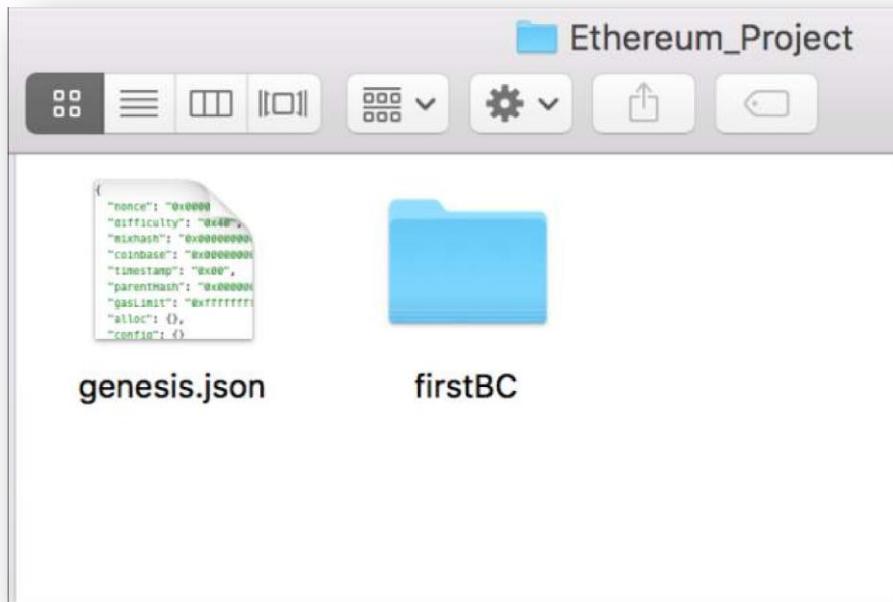
COMMANDS:
    account          Manage accounts
    attach           Start an interactive JavaScript environment (connect to node)
    bug              opens a window to report a bug on the geth repo
    console          Start an interactive JavaScript environment
    copydb           Create a local chain from a target chaindata folder
    dump             Dump a specific block from storage
    dumpconfig       Show configuration values
    export            Export blockchain into file
    export-preimages Export the preimage database into an RLP stream
    import            Import a blockchain file
    import-preimages Import the preimage database from an RLP stream
    init              Bootstrap and initialize a new genesis block
    js                Execute the specified JavaScript files
    license           Display license information
    makecache         Generate ethash verification cache (for testing)
    makedag           Generate ethash mining DAG (for testing)
    monitor           Monitor and visualize node metrics
    removedb          Remove blockchain and state databases
    version           Print version numbers
    wallet            Manage Ethereum presale wallets
    help, h           Shows a list of commands or help for one command

ETHEREUM OPTIONS:
    --config value      TOML configuration file
    --datadir "/Users/mohammht/Library/Ethereum" Data directory for the databases and keystore
    --keystore          Directory for the keystore (default = inside the
    datadir)
```


Create the storage of the blockchain

- Go to the directory of the genesis.json file
- Specify directory of your blockchain
- Create the storage from the genesis block

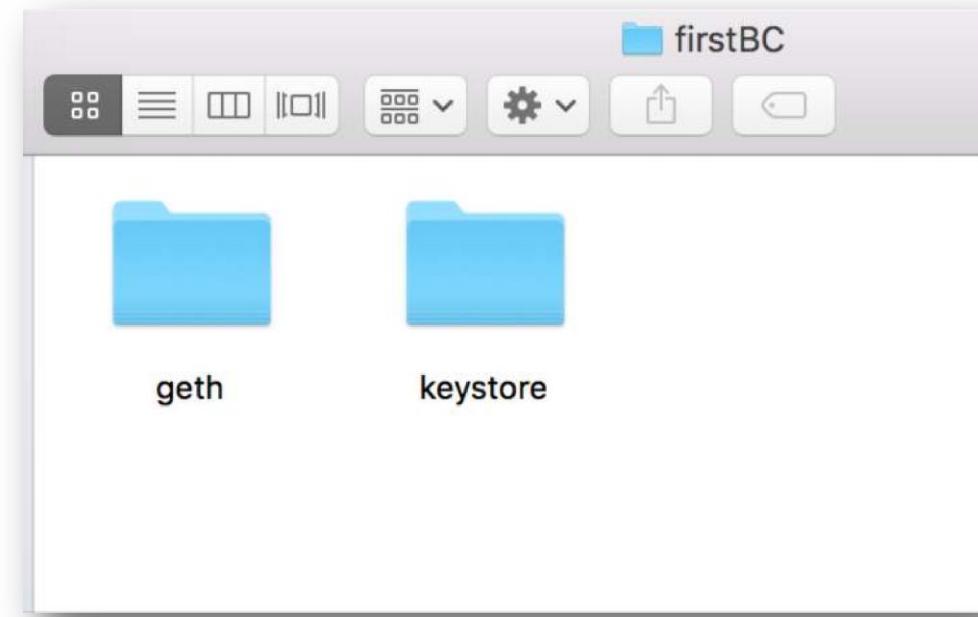
```
[ds-install:Documents mohammht$ cd Ethereum_Project/  
ds-install:Ethereum_Project mohammht$ geth --datadir firstBC init genesis.json]
```



Folder name of your
blockchain

Inside the Blockchain Folder

- **geth folder:** Store your database
- **keystore:** Store your Ethereum accounts



Start the Ethereum peer node

- Start the blockchain

```
geth --datadir fistBC --networkid 100 console
```

- Networkid provides privacy for your network.
- Other peers joining your network must use the same networkid.

Blockchain started

- Type
admin.nodeInfo
to get the
information
about your
current node

```
[> admin.nodeInfo
{
  enode: "enode://4561ccdd7fdf3f0bdbc903b7bef7d472e136fe2b63012151a1dd3c27e52f49bda2ef66631e67022
b7ca7b9fba06bb0eda8b47210b198f3eff7e67414d695ed6@[::]:30303",
  id: "4561ccdd7fdf3f0bdbc903b7bef7d472e136fe2b63012151a1dd3c27e52f49bda2ef66631e67022b7ca7b9fba0
6bb0eda8b47210b198f3eff7e67414d695ed6",
  ip: "::",
  listenAddr: "[::]:30303",
  name: "Geth/v1.8.9-stable/darwin-amd64/go1.10.2",
  ports: {
    discovery: 30303,
    listener: 30303
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 4370000,
        chainId: 1,
        daoForkBlock: 1920000,
        daoForkSupport: true,
        eip150Block: 2463000,
        eip150Hash: "0x2086799aeebeae135c246c65021c82b4e15a2c451340993aacfd2751886514f0",
        eip155Block: 2675000,
        eip158Block: 2675000,
        ethash: {},
        homesteadBlock: 1150000
      },
      difficulty: 17179869184,
      genesis: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      head: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      network: 100
    }
  }
}
>
```

Create an account

- Type *personal.newAccount* to create as many accounts as you need

```
[> personal.newAccount('Type your password here')
  "0xa78eb41a10f096d4d8c4c9ca5196427aaa3fdb33"
> ]
```

- See the created account(s)

```
> eth.accounts
[ "0xa78eb41a10f096d4d8c4c9ca5196427aaa3fdb33", "0x354d952e40fc35a47562d479c86e41f6623e5f8c"]
>
```

Mining

- Type *miner.start()* to start mining

```
[> miner.start()
INFO [05-30|12:07:54] Updated mining threads                                     threads=0
INFO [05-30|12:07:54] Transaction pool price threshold updated price=180000000000
null
> INFO [05-30|12:07:54] Starting mining operation
INFO [05-30|12:07:54] Commit new mining work                                     number=1 txs=0 uncles=0 elapsed=22
8.827µs
INFO [05-30|12:07:57] Generating DAG in progress                               epoch=1 percentage=0 elapsed=2.013
s
INFO [05-30|12:07:59] Generating DAG in progress                               epoch=1 percentage=1 elapsed=4.151
s
INFO [05-30|12:08:03] Generating DAG in progress                               epoch=1 percentage=2 elapsed=7.322
s
INFO [05-30|12:08:06] Generating DAG in progress                               epoch=1 percentage=3 elapsed=10.70
5s
INFO [05-30|12:08:09] Generating DAG in progress                               epoch=1 percentage=4 elapsed=14.04
3s
INFO [05-30|12:08:13] Generating DAG in progress                               epoch=1 percentage=5 elapsed=17.56
5s
INFO [05-30|12:08:16] Generating DAG in progress                               epoch=1 percentage=6 elapsed=20.99
9s
INFO [05-30|12:08:20] Generating DAG in progress                               epoch=1 percentage=7 elapsed=24.40
9s]
```

- Type *miner.stop()* to stop mining