

# A hands-on tutorial: Working with Smart Contracts in Ethereum

Mohammad H. Tabatabaei

Roman Vitenberg

Kaiwen Zhang

Mohammad Sadoghi

Hans-Arno Jacobsen

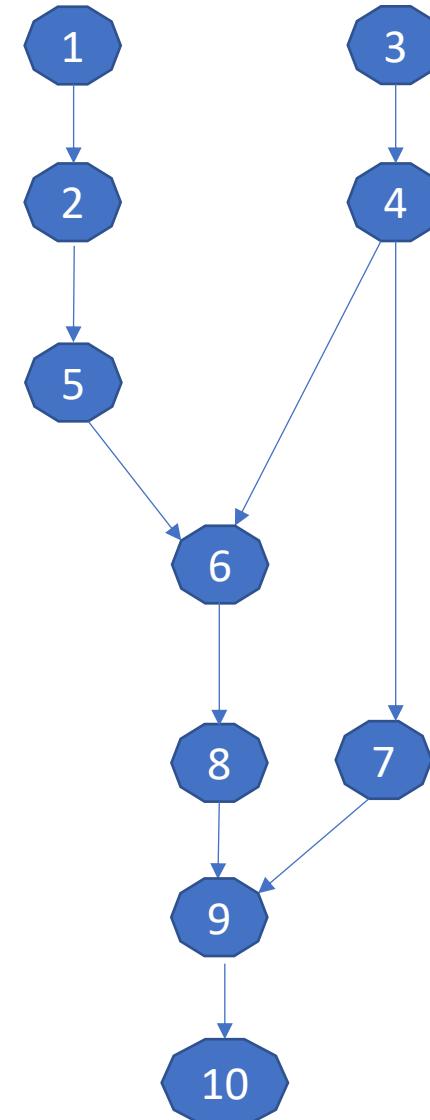
# Different tools provide different functionality

	Tools	Remix	Ganache	MyEtherWallet	Geth
	Activities				
1	Configure the Blockchain	-	-	-	+
2	Deploy the Blockchain	Not Persistent	+	-	+
3	Develop the contract	+	-	-	+
4	Compile the contract	+	-	-	+
5	Create user account	+	+	+	+
6	Deploy the contract	+	-	+	+
7	Create the UI for interacting	+	-	+	+
8	Run the client	+	-	+	+
9	Interact with the contract & have fun	+	-	+	+
10	Monitor the execution	-	+	-	+

<https://remix.ethereum.org/>

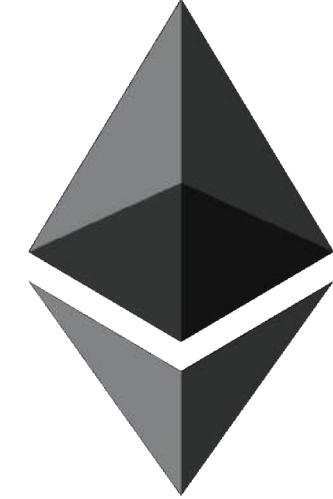
<http://truffleframework.com/ganache/>

<https://github.com/kvhnuke/etherwallet/releases/tag/v3.21.06>



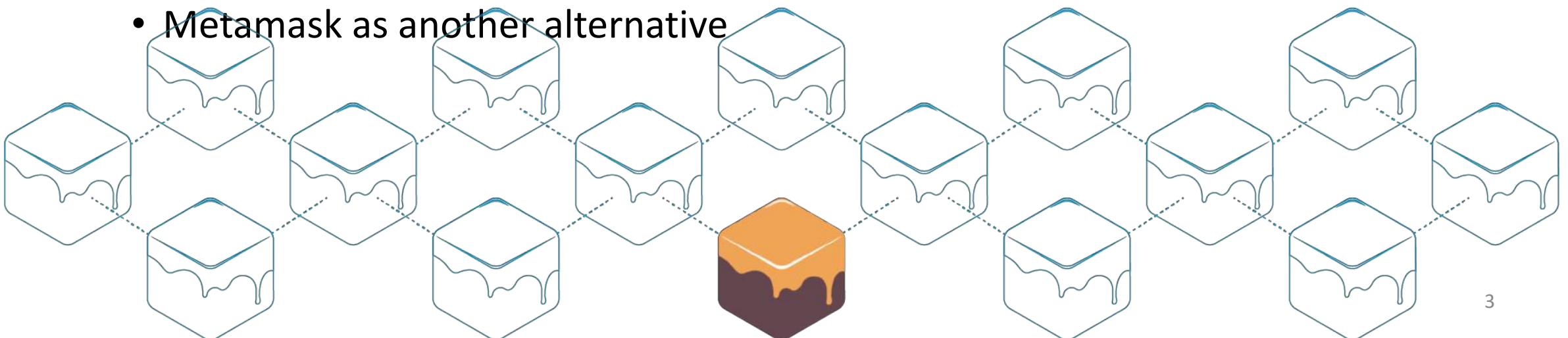
# Use which tool for what purpose? (1/2)

- Use Geth for everything?
  - Powerful but command-line only
- What should I use?
  - As a starting point for developing contracts – mostly Remix
- What cannot Remix do?
  - Configure the blockchain
  - Create real (non-test) user accounts and transfer funds between user accounts
  - Monitor the execution
  - Other advanced operations



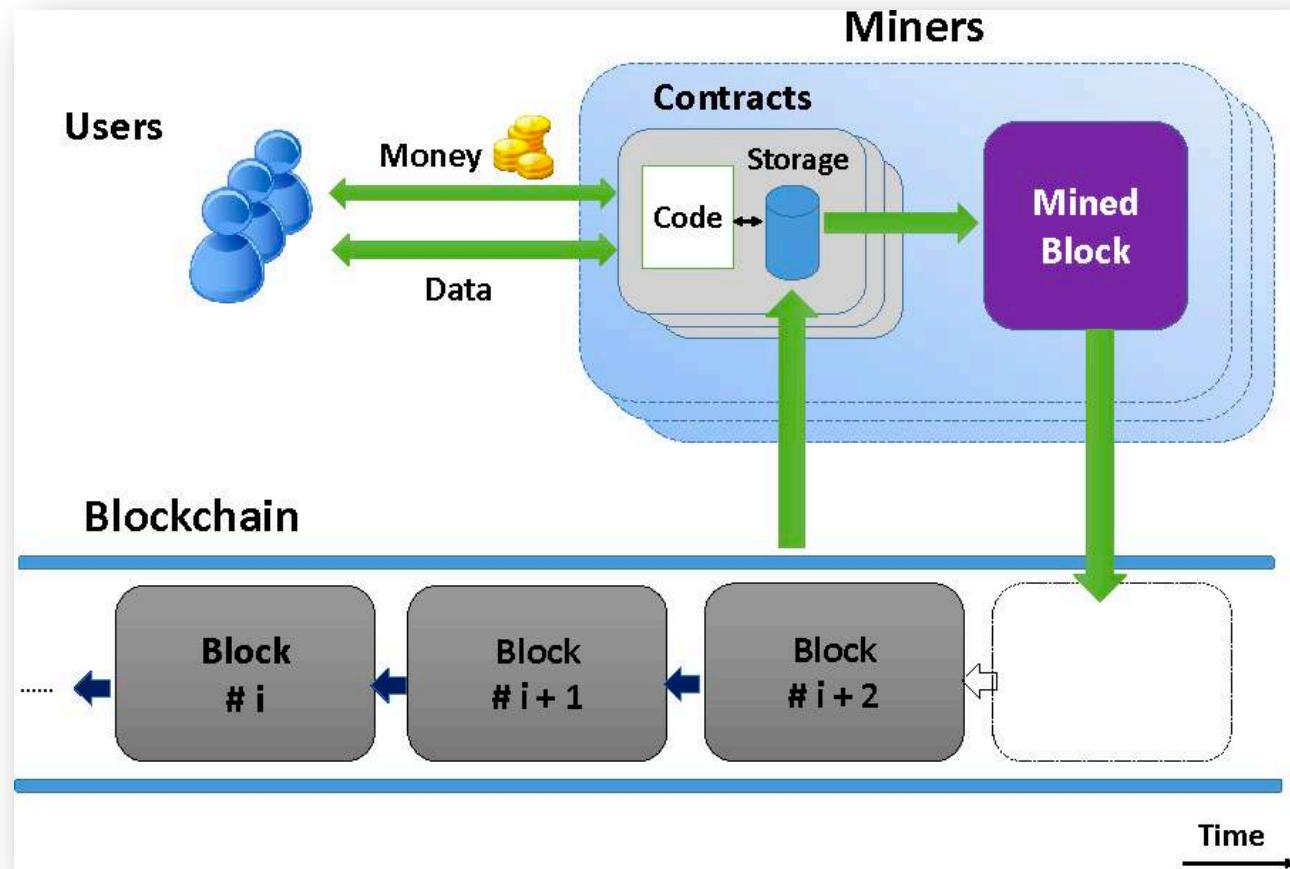
# Use which tool for what purpose? (2/2)

- Why use Ganache?
  - To inspect and monitor the execution
  - To visualize certain elements in a better way
- Why use MyEtherWallet?
  - To create a personal wallet (real user account), transfer funds between user accounts, and interact with contracts
  - Metamask as another alternative



# Smart Contracts

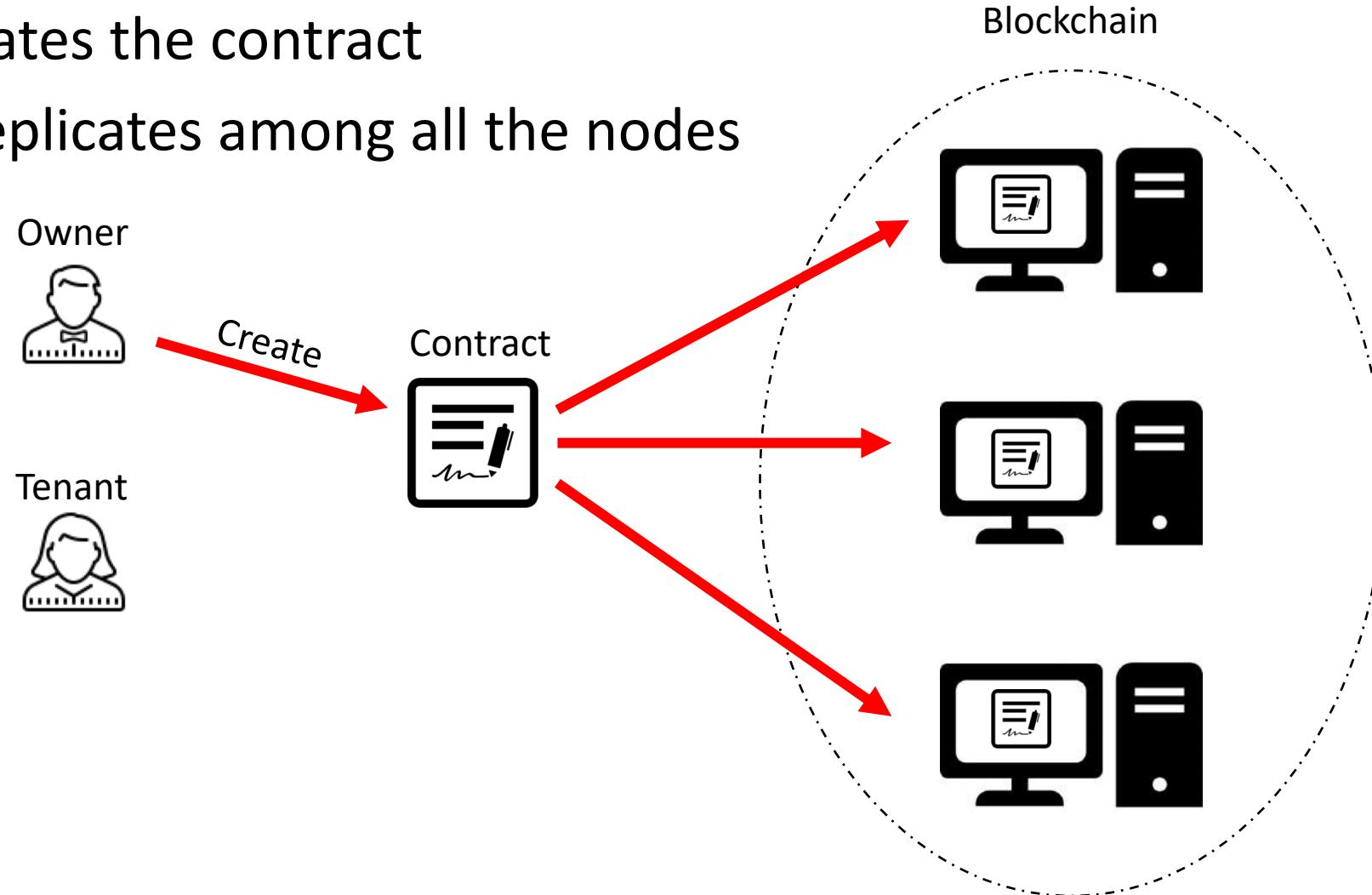
- In the form of code
- Stored on a blockchain
- Executes under given conditions



- K. Delmolino, M. Arnett, A. E. Kosba, A. Miller, and E. Shi, “Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab,” *IACR Cryptology ePrint Archive*, vol. 2015, p. 460, 2015.

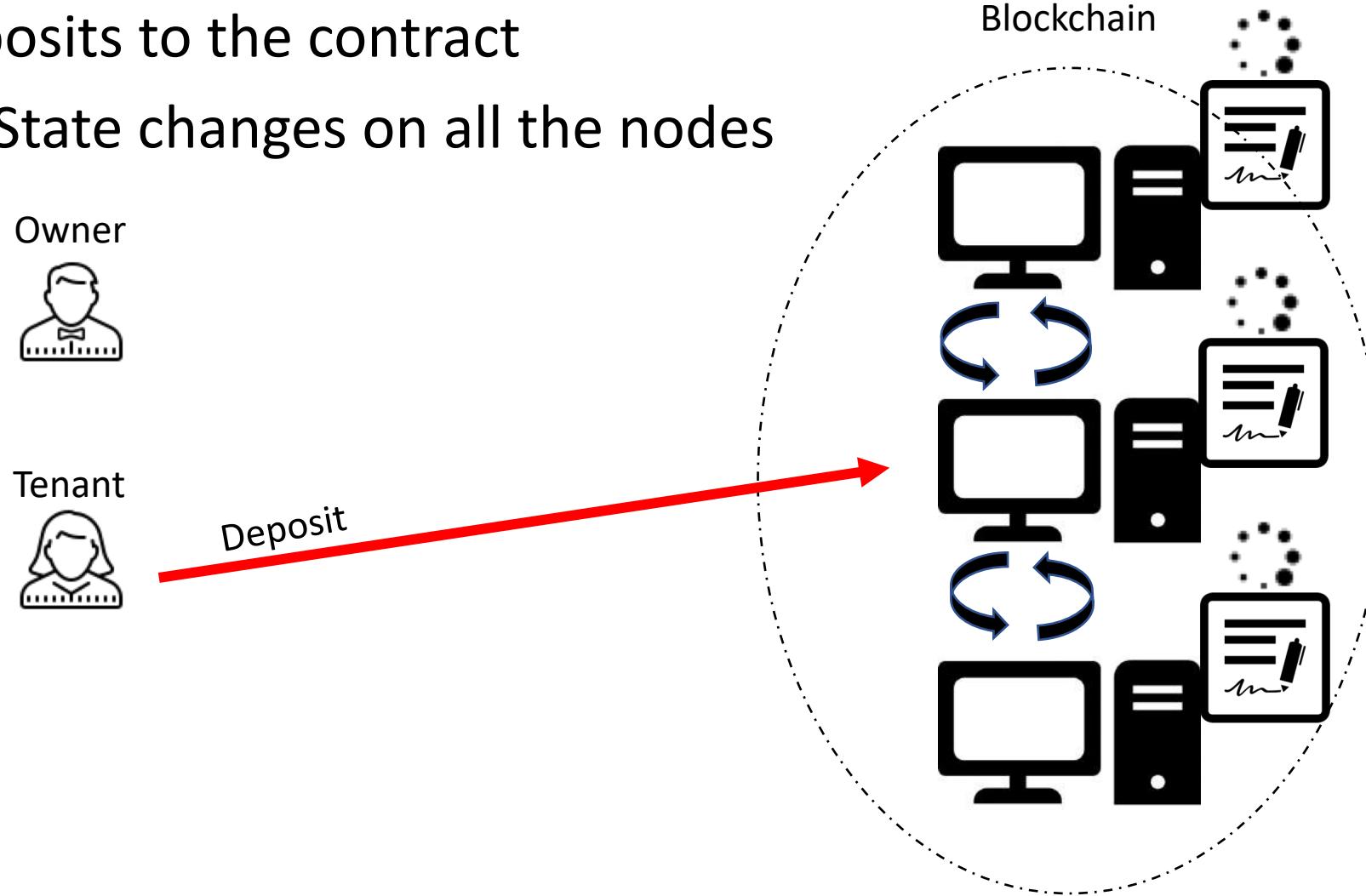
# Smart Contracts Example (1/3)

- Owner creates the contract
- Contract replicates among all the nodes



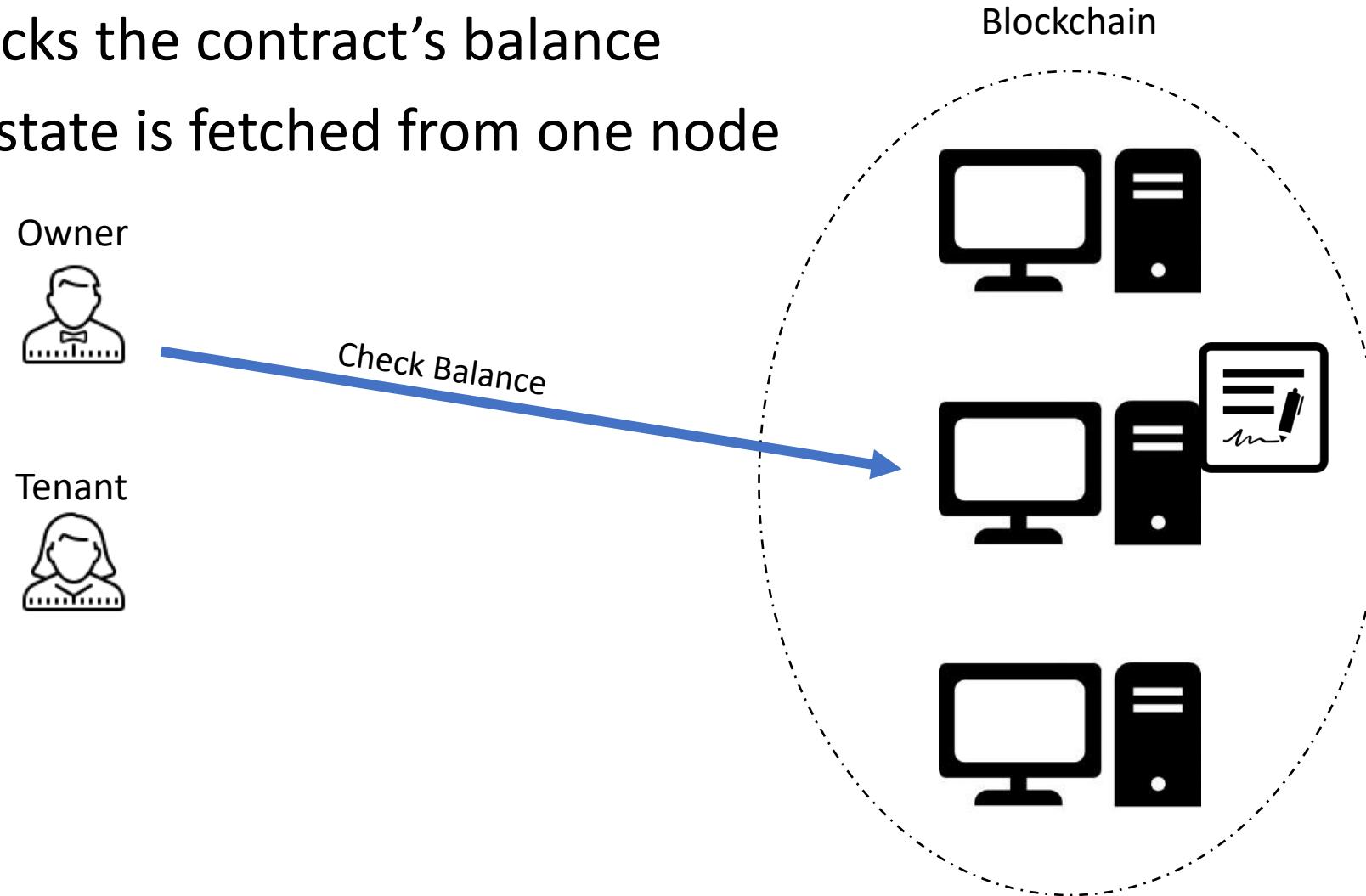
# Smart Contracts Example (2/3)

- Tenant deposits to the contract
- Contract's State changes on all the nodes



# Smart Contracts Example (3/3)

- Owner checks the contract's balance
- Contract's state is fetched from one node



# Smart Contracts

1. Developing a simple contract
2. Compiling the contract
3. Deploying the contract
4. Interacting with the contract
5. Adding more functions to our code to make it more practical

# Open Remix : remix.ethereum.org

- An open source tool for writing, compiling and testing Solidity contracts

The screenshot shows the Remix Ethereum IDE interface. On the left, there's a file tree with 'browser' and 'config' expanded, and a file 'browser/firstContract.sol' selected. The code editor displays 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 }
```

The right side of the interface includes a toolbar with 'Compile', 'Run', 'Analysis', 'Testing', 'Debugger', 'Settings', and 'Support'. A status bar at the top right says 'Current version:0.5.1+commit.c8a2cb62.Emscripten.clang'. Below it is a dropdown for 'Select new compiler version' and checkboxes for 'Auto compile', 'Enable Optimization', and 'Hide warnings'. A large button labeled 'Start to compile (Ctrl-S)' is prominent. To the right of the compiler controls is a sidebar for the 'financialContract' artifact, showing tabs for 'Details', 'ABI', and 'Bytecode'. At the bottom, a transaction history section shows '[2] only remix transactions, script' with a note about accessible libraries like web3, ethers.js, and swarmgw.

# Solidity

- Object-oriented
- Contract-oriented
- High-level language
- Influenced by C++, Python, and JavaScript
- Target Ethereum Virtual Machine (EVM)



## Serpent as an Alternative?

- Low-level language
- Complex compiler

# 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){
8         return balance;
9     }                                ← Getter function
10
11    function deposit(uint newDeposit) public{
12        balance = balance + newDeposit;
13    }                                ← Setter function
14
15 }
```

# Compile the Contract

- Compile tab: Start to compile button

The screenshot shows the Truffle UI interface for compiling 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
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 }
```

On the right, the "Compile" tab is active. It displays the current compiler version as "0.5.1+commit.c8a2cb62.Emscripten clang". Below this, there is a section for selecting a new compiler version, a checkbox for "Auto compile" (which is checked), and options for "Enable Optimization" and "Hide warnings". The "Start to compile (Ctrl-S)" button is highlighted with a red box. At the bottom, there is a dropdown menu set to "financialContract" and a "Swarm" icon.

# Set Deployment Parameters (1/2)

- Run tab: Environment = JavaScript VM

The screenshot shows the Truffle UI 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
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, the "Run" tab is selected, showing deployment parameters:

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

A deployment target field labeled "financialContract" is shown, along with a "Deploy" button and options for "At Address" or "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).

# Types of Blockchain Deployment

- Private: e.g., Ganache sets a personal Ethereum blockchain for running tests, executing commands, and inspecting the state while controlling how the chain operates.
- Public Test (Testnet): Like Ropsten, Kovan and Rinkeby which are existing public blockchains used for testing and which do not use real funds. Use faucet for receiving initial virtual funds.
- Public Real (Mainnet): Like Bitcoin and Ethereum which are used for real and which available for everybody to join.

# 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 is named `browser/firstContract.sol`. The code defines a contract named `financialContract` with two functions: `getBalance()` and `deposit(uint newDeposit)`.
- Run Tab:** The `Run` tab is selected. The environment is set to `JavaScript VM`. The account is set to `0xca3...a733c (99.9999999999998644)`. The gas limit is `3000000` and the value is `0 wei`.
- Deployment Section:** The contract name is `financialContract`. The `Deploy` button is highlighted with a red circle.
- Transactions Recorded:** One transaction is recorded, which is the deployment of the contract.
- Deployed Contracts:** The deployed contract is listed as `financialContract at 0x692...77b3a (memory)`. It shows two functions: `deposit` and `getBalance`, both highlighted with red circles.
- Bottom Bar:** The status bar shows `[2] only remix transactions, script` and a search bar for transactions.
- Footnote:** A note at the bottom states: "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. The callout box contains the text: "Press getBalance to see the initial amount". A blue numbered circle labeled "1" is positioned next to the callout box.

Method	Description
deposit	uint256 newDeposit
getBalance	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", indicated by a red arrow pointing to a callout box. The callout box contains the text: "Input a value and press deposit button to create and confirm the transaction". A green numbered circle labeled "2" is positioned next to the callout box.

**Screenshot 3:** This screenshot shows the state after the deposit has been confirmed. The "getBalance" button is highlighted with a blue arrow pointing to another callout box. The callout box contains the text: "Press getBalance again to see the result". A blue numbered circle labeled "3" is positioned next to the callout box.

Method	Description
deposit	12
getBalance	0: uint256: 313012

# Additional features

- Transferring funds from an account to the contract
- Saving the address of the contract creator
- Limiting the users' access to functions
- Withdrawing funds from the contract to an account

# Receive ether (1/2)

- Transfer money to the contract

Payable keyword  
allows receiving  
ether

```
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 }
```

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

We can get the  
balance of the  
contract

# Receive ether (2/2)

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



# 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

# 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 }
```

# 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 website interface. On the left, there's a form titled "Create New Wallet" with a password input field containing "Do NOT forget to save this!". Below it is a "Create New Wallet" button. A note below the button states: "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." At the bottom of this section are links to "How to Create a Wallet" and "Getting Started". On the right, there's a "Network ETH (myetherapi.com)" dropdown menu. This menu lists various Ethereum networks and a section for "Already have a wallet". A red arrow points from the "Add Custom Network / Node" link at the bottom of the dropdown to the "Add Custom Network / Node" link highlighted with a red circle in the main content area.

3.21.05 English ▾ Gas Price: 41 Gwei ▾ Network ETH (myetherapi.com) ▾

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

The network is really full right now. Choose another network.

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 a wallet

- Ledger / TREZOR: Use your hardware wallet.
- MetaMask Chrome Extension: So you can interact with your wallet without being on a phishy site.
- Jaxx / imToken: To access your wallet.
- Mist / Geth / Parity / JSON-RPC: ELLA (ellaism.org), ETSC (etsc.network), Add Custom Network / Node

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 (expanse.tech)  
UBQ (ubiqscan.io)  
POA (core.poa.network)  
TOMO (core.tomocoin.io)  
ELLA (ellaism.org)  
ETSC (etsc.network)

Add Custom Network / Node

MyEtherWallet.com does not hold your keys for you. We cannot access accounts, recover keys, reset passwords, nor reverse transactions. Protect your keys & always check that you are on correct URL. You are responsible for your security.

25

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

The image shows the Ganache application window and a 'Set Up Your Custom Node' dialog box.

**Ganache Window:**

- Accounts: 0
- Blocks: 20000000000
- Gas Price: 6721975
- Gas Limit: 5777
- Network ID: 5777
- RPC Server: **HTTP://127.0.0.1:7545** (circled in red)
- Mining Status: AUTOMINING

**Set Up Your Custom Node Dialog:**

- Node Name: Private ETH Node
- URL: http://127.0.0.1 (with a red arrow pointing from the Ganache RPC address)
- Port: 7545 (with a red arrow pointing from the Ganache port number)
- HTTP Basic access authentication:
- Network Selection:
  - ETH
  - ETC
  - Ropsten
  - Kovan
  - Rinkeby
  - Custom
  - Supports EIP-155
- Buttons: Cancel, Save & Use Custom Node (highlighted in blue)

# MyEtherWallet

- Contracts tab: Deploy Contract

The screenshot shows the MyEtherWallet web application 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 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 circled in red), ENS, DomainSale, Check TX Status, View Wallet Info, and Help. The main content area has a large heading "Interact with Contract or Deploy Contract" with "Deploy Contract" highlighted by a red oval and arrow. Below this, there are sections for "Byte Code" (empty) and "Gas Limit" (set to 300000).

3.21.05 English ▾ Gas Price: 41 Gwei ▾ Network My Ether Node:eth (Custom) ▾  
The network is really full right now. Check Eth Gas Station for gas price to use.

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

Interact with Contract or **Deploy Contract**

Byte Code

Gas Limit  
300000

# Remix

- Type your contract and compile it

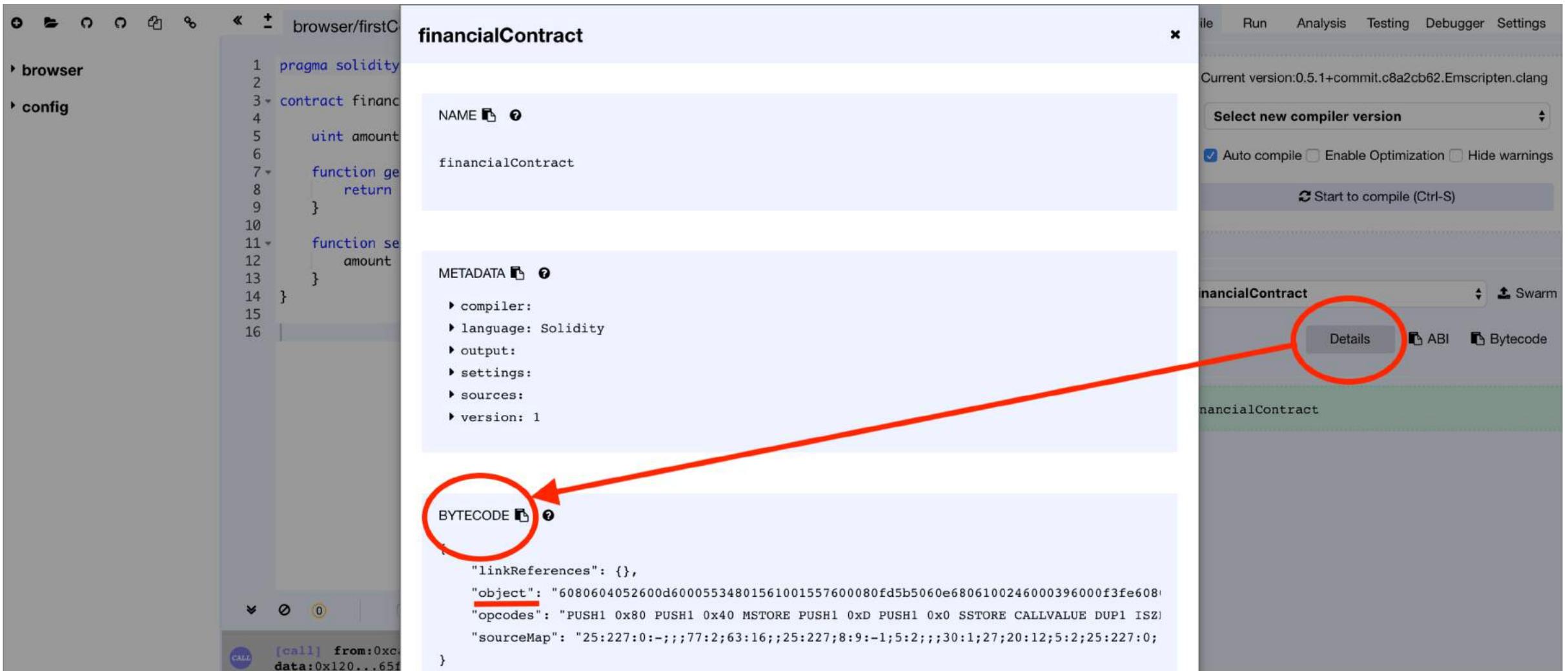
The screenshot shows the Remix IDE interface. On the left, there is a code editor window titled "browser/firstContract.sol" containing the following Solidity code:

```
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 }
```

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". Underneath this, there is a dropdown menu labeled "Select new compiler version" and several checkboxes: "Auto compile" (checked), "Enable Optimization" (unchecked), and "Hide warnings" (unchecked). A prominent red rectangle highlights the "Start to compile (Ctrl-S)" button, which is located in a blue bar below the checkboxes. At the bottom of the interface, there are tabs for "financialContract", "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 (highlighted with a red circle)
- Address:** 0x231eAeEF9EA93F5370a1F633F32E45AF570980E8, Balance: 100.00 ETH
- TX Count:** 0, Index: 1 (highlighted with a red circle)
- Address:** 0x970fc818790E900598C
- TX Count:** 0, Index: 2 (highlighted with a red circle)
- Address:** 0xb59BD5568d0be42C13f
- Address:** 0x280AFA533B9fa1A97a6
- Address:** 0xD6D30E82AB17c3046AE7CAC88475ECcaRF2757c5

A modal window is open for the first address, showing the PRIVATE KEY: a53cf8cb7b66d91ca388ef9ce4e45e39997f2773247c27bb2c7cae35a1b3d383. A red arrow points from the 'PRIVATE KEY' label to the key icon in the red circle, and another red arrow points from the key icon to the red circle itself.

# 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

# MyEtherWallet

Click on *Sign Transaction* button to deploy your contract

# 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.

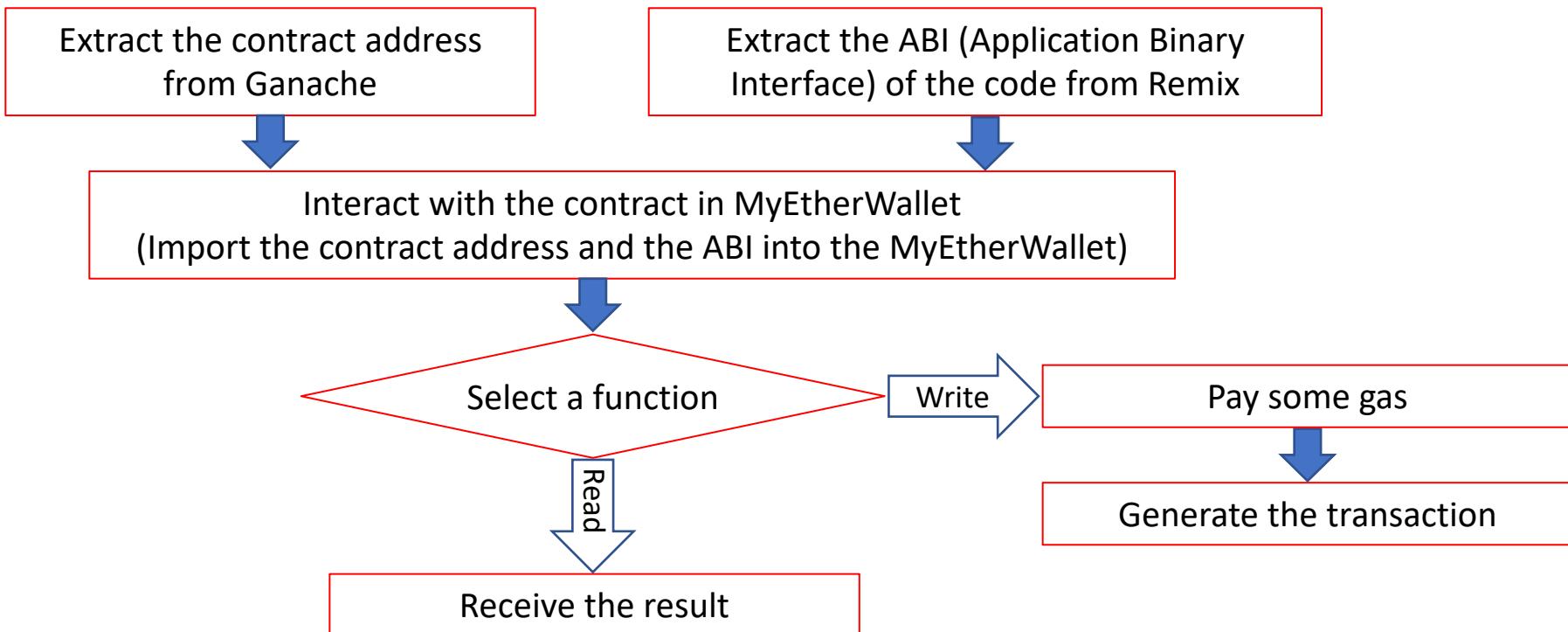
The screenshot shows the Ganache interface with the following details:

ACCOUNT	BALANCE	TX COUNT	INDEX	KEY
0x231eAeEF9EA93F5370a1F633F32E45AF570980E8	99.99 ETH	1	0	🔑
0x970fc818790E900598C57E48b89B6D3D8896D416	100.00 ETH	0	1	🔑
0xb59BD5568d0be42C13fB521f845243F1CDaF2eF1	100.00 ETH	0	2	🔑
0x280AFA533B9fa1A97a6D2E4640412FD86FC5dd36	100.00 ETH	0	3	🔑
0xD6D39E82AB17c30460F2CAc88425ECcaBf2757c5	100.00 ETH	0	4	🔑

Key UI elements highlighted with red circles:

- The "CURRENT BLOCK" value "1" in the top navigation bar.
- The "BALANCE" value "99.99 ETH" for the first account.
- The "TX COUNT" value "1" for the first account.

# 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 searching for block numbers or tx hashes. The main area displays a transaction entry. The TX HASH is listed as `0x1e40cc28802d152e810bd9f40bea83d83b1655fc9bace6e801ec6db5fc84b1a`. The FROM ADDRESS is `0x231eAeEF9EA93F5370a1F633F32E45AF570980E8`. The CREATED CONTRACT ADDRESS is `0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d`. On the right side, there are buttons for CONTRACT CREATION, GAS USED (124604), and VALUE (0).

TX HASH	CONTRACT CREATION
<code>0x1e40cc28802d152e810bd9f40bea83d83b1655fc9bace6e801ec6db5fc84b1a</code>	
FROM ADDRESS	GAS USED
<code>0x231eAeEF9EA93F5370a1F633F32E45AF570980E8</code>	124604
CREATED CONTRACT ADDRESS	VALUE
<code>0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d</code>	0

# Remix

Copy the ABI

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

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, the interface includes a toolbar with "Compile", "Run", "Analysis", "Testing", "Debugger", "Settings", and "Sup". Below the toolbar, it says "Current version:0.5.1+commit.c8a2cb62.Emscripten clang". There is a dropdown menu for "Select new compiler version" and checkboxes for "Auto compile", "Enable Optimization", and "Hide warnings". A button labeled "Start to compile (Ctrl-S)" is highlighted in purple. Under the contract name "financialContract", there are tabs for "Details", "ABI" (which is circled in red), and "Bytecode". A green bar at the bottom also displays the contract name "financialContract".

# 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": "0x098bc5a00", "gasLimit": "0xa2a1", "to": "0xf22A8cA21D7eeF564FD5Ea743d"}

Signed Transaction



0xf8680185098bc5a0082a2a194f22a8ca21d7eeef564fd5ea743dd9326197cfaa2d80845b34b96626a04285bd52ad31



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


```

197CFAA2d

**WRITE**

# 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)
```

# Genesis block

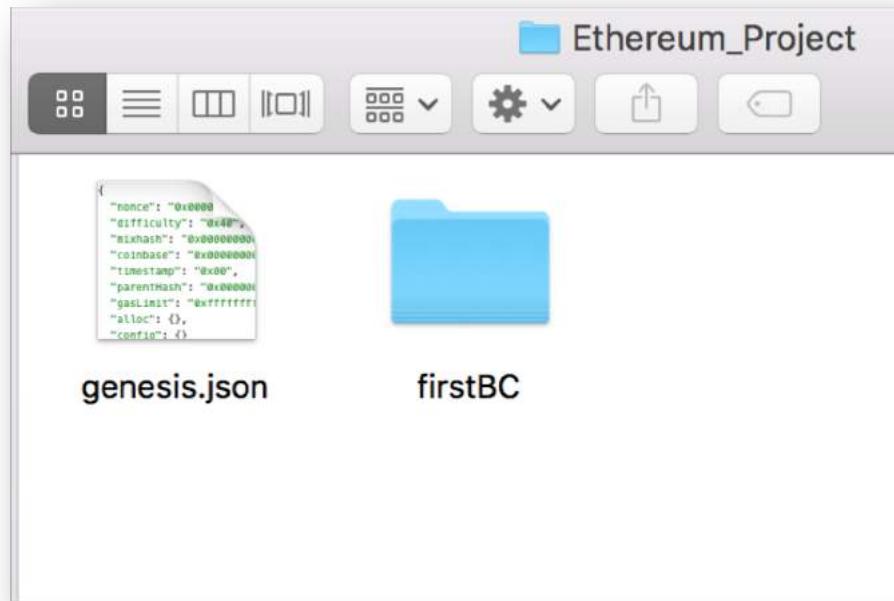
- The first block in the chain and a json file that stores the configuration of the chain

- Create and store the file as genesis.json

# 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

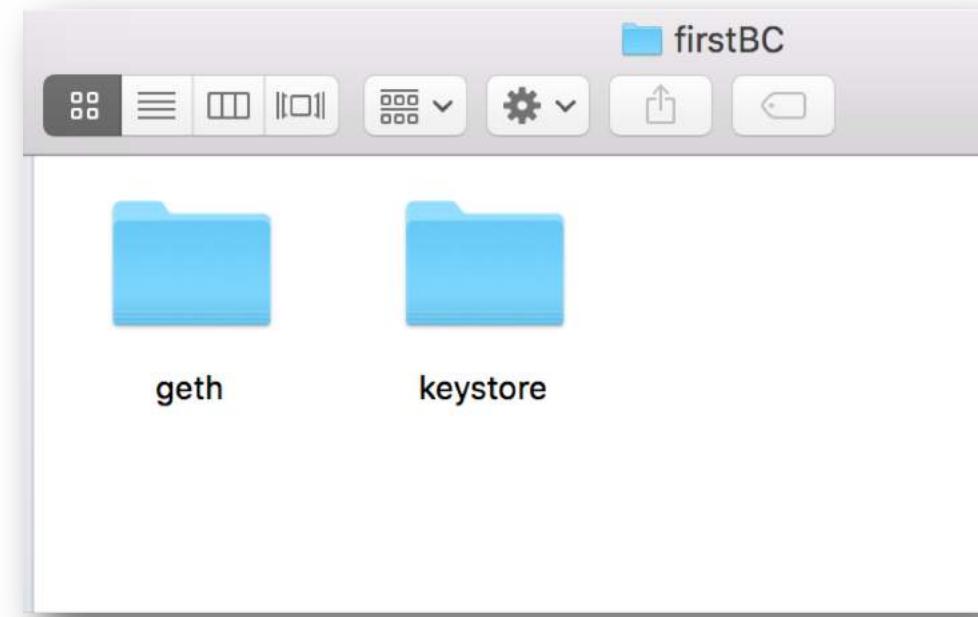
```
[ds-install:Documents mohammht$ cd Ethereum_Project/  
ds-install:Ethereum_Project mohammht$ geth --datadir firstBC init genesis.json]
```



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://4561ccdd7fdf3f0bdbc903b7bef7d472e136fe2b63012151a1dd3c27e52f49bda2ef66631e67022b7ca7b9fba06bb0eda8b47210b198f3eeff7e67414d695ed6@[::]:30303",
  id: "4561ccdd7fdf3f0bdbc903b7bef7d472e136fe2b63012151a1dd3c27e52f49bda2ef66631e67022b7ca7b9fba06bb0eda8b47210b198f3eeff7e67414d695ed6",
  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

# Thank You!

## Any Questions?



Mohammad H. Tabatabaei  
[mohammht@ifi.uio.no](mailto:mohammht@ifi.uio.no)