

# CS 731: Blockchain Technology And Applications

**Sandeep K. Shukla**  
**IIT Kanpur**

C3I Center



# Acknowledgement

---

- The material of this lecture material is from various websites in particular related to Truffle and Ganache-Cli (including:
- <https://medium.com/haloblock/deploy-your-own-smart-contract-with-truffle-and-ganache-cli-beginner-tutorial-c46bce0bd01e>
- <https://github.com/ethereum/wiki/wiki/Design-Rationale>
- <https://blockgeeks.com/guides/solidity/>
- <https://truffleframework.com/tutorials/ethereum-overview>

# Introduction to Ethereum

# Revisit Blockchain

---

- So far, with bitcoin, we saw use of blockchain as a
  - Creation engine for digital currency
  - A distributed, tamper-resistant log of transactions in crypto-currency
  - A distributed ledger that solves heuristically Byzantine-safe distributed consensus
- Now, we look into more generic blockchain
  - That can be thought of as a distributed execution engine for consistent program execution
  - That can also support crypto-currency and in fact uses crypto-currency to solve consensus problem
  - That can be used to enforce contracts between parties through smart contracts

# Agenda of this lecture

---

- Relook at the Blockchain technology
  - Why use a blockchain?
  - What is a blockchain?
  - How does a blockchain work?
- The Ethereum Blockchain
  - What is Ethereum?
  - What is a smart contract?
  - Ethereum Networks
  - Distributed Applications (Dapps)
- Truffle and Ganache-CLI for your first smart contract

# Why use Blockchain?

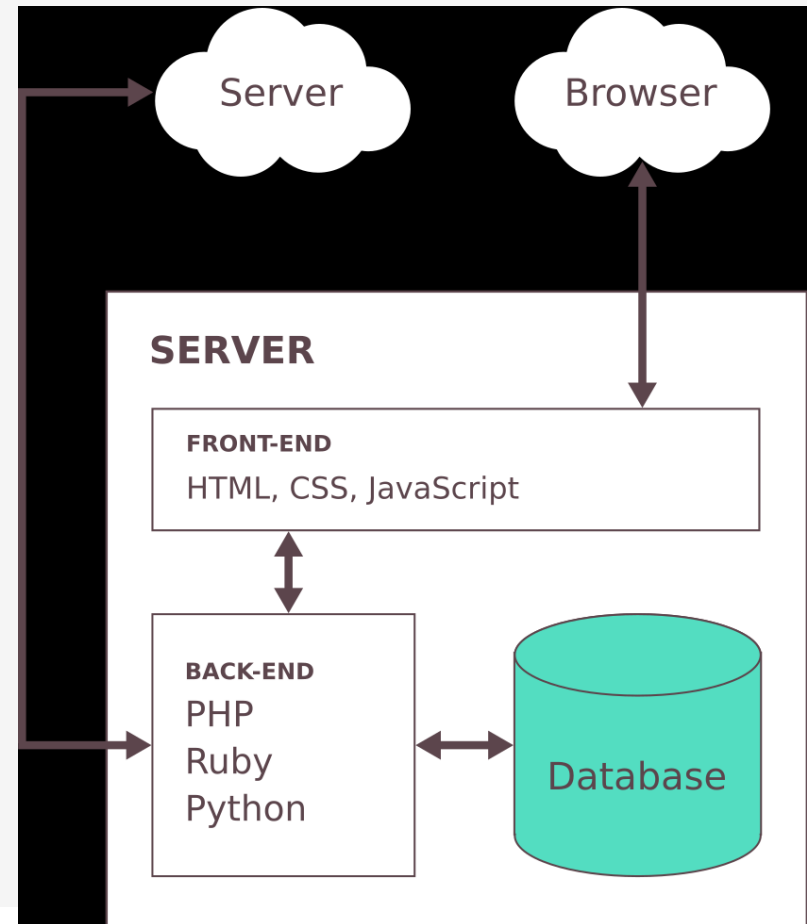
---

- Blockchains are used when **multiple parties**, perhaps located across the world, **need to share data and transfer value** without **trusting** each other.
- The financial world describes this trust as the **counterparty risk**:
  - *the risk that the other party won't hold up their end of the bargain.*
- Blockchains attempt **remove the counterparty risk** through a clever usage of mathematics, cryptography, and peer-to-peer networking.

# Classical Database Applications

## centralized approach

- Can be manipulated from inside and outside
- We have to trust the owners of the database and servers to keep data secure and with integrity
- Hackers can also infiltrate the server and change data
- Centralized backup and restore can not be necessarily trusted



## Mitigating security/integrity issues in Centralized Data stores

---

- Every time data changes, make a backup copy and retain all historical backups
  - Hash the backup and keep it safe to prove integrity violation
- To share data, all stake holders must agree that the data is not tampered with (some proof mechanism might be required)
- Only way to ensure all that is through trusted 3<sup>rd</sup> party audit



# What is a block chain in this context?

---

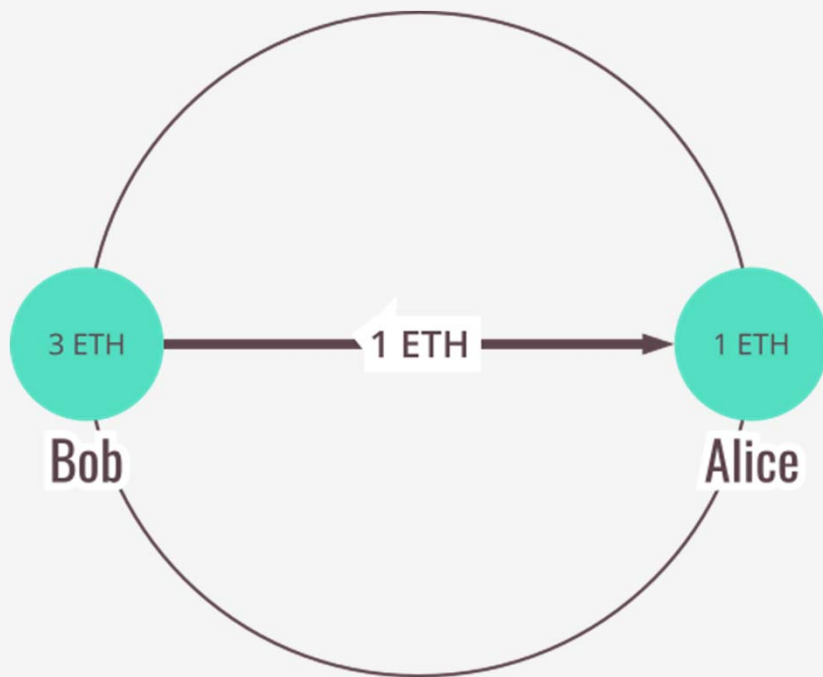
- Shared database consisting of ledger of transactions
- Every stake holder keeps a copy of the ledger and can verify all transactions that are put in the ledger
- Reading/writing on the ledger is completely decentralized and secure
- Fault tolerance
- Independent verification by anyone interested :  
disintermediation (anyone can audit)

# How does Blockchain work?

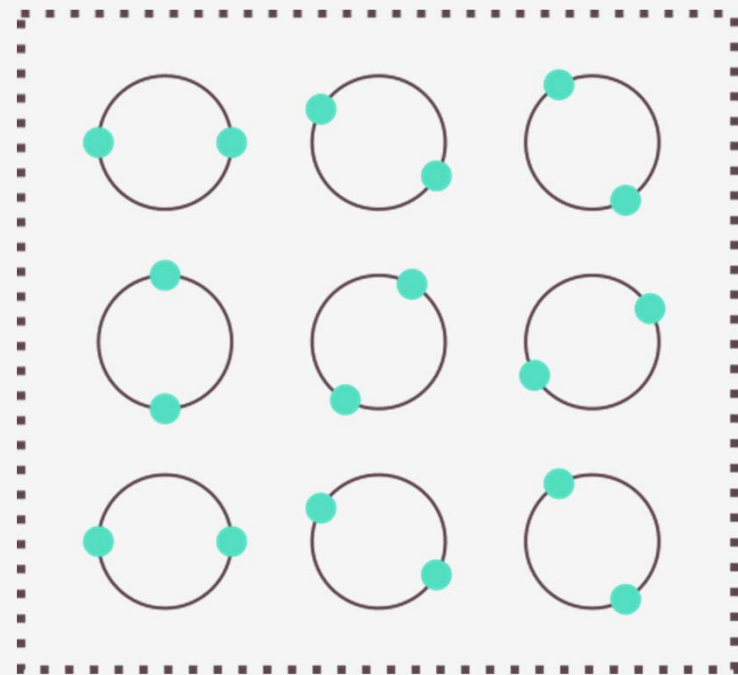
- Nodes, Transactions, Blocks
- Mining through solving hard problems (solving Byzantine fault-tolerant consensus)
- Hashing for integrity
- Digital Signature for Authenticity and/or authorization
- Permanence (Tamper resistance)

# Blockchain in Pictures

---

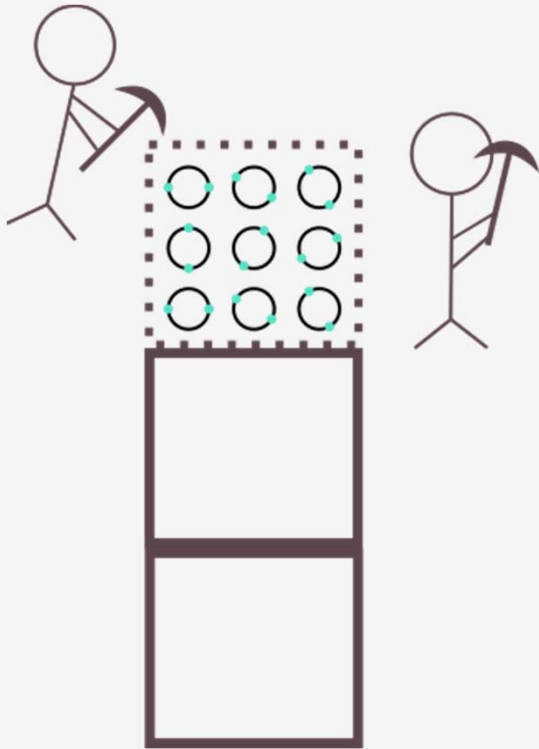


***Bob attempts to send Alice 1 ETH***



***Bob and Alice's transaction is combined with other transactions that have occurred since the last block***

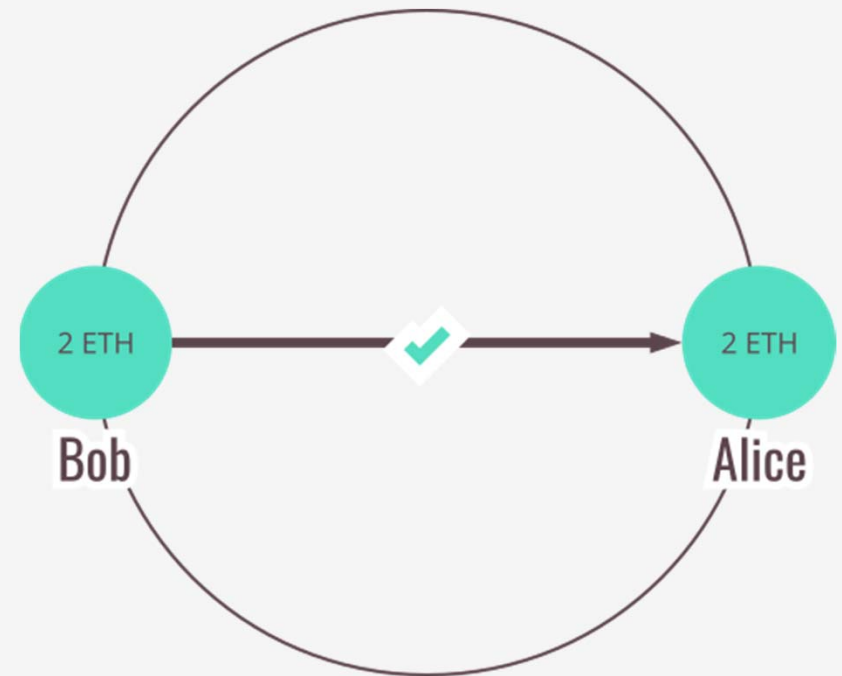
## Blockchain in Pictures (2)



*Miners compete to validate the block with the new set of transactions*



*The victorious miner creates a new block and receives a reward*



*With the transaction validated, Alice receives 1 ETH*

# What is Ethereum?

---

- **Ethereum is a blockchain that allows you to run programs in its trusted environment.**
  - contrasts with the Bitcoin blockchain, which only allows you to manage cryptocurrency.
- Ethereum has a virtual machine -- Ethereum Virtual Machine (EVM).
- The EVM allows code to be verified and executed on the blockchain,
  - providing guarantees it will be run the same way on everyone's machine.
- This code is contained in "smart contracts"
- Ethereum maintains the state of the EVM on the blockchain.
  - All nodes process smart contracts to verify the integrity of the contracts and their outputs.

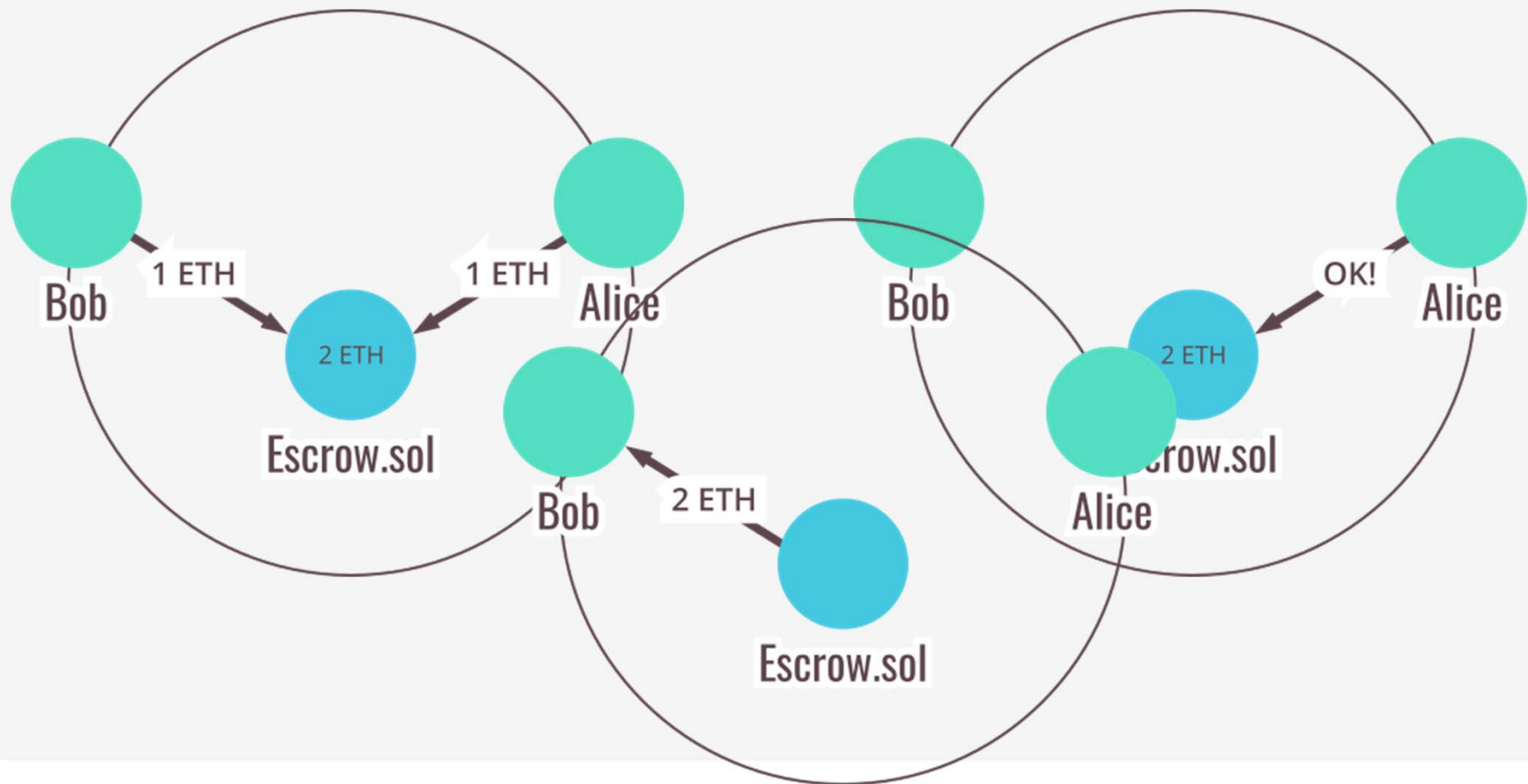
# What is a smart contract?

---

- **A smart contract is code that runs on the EVM.**
- Smart contracts can accept and store ether, data, or a combination of both.
- Using the logic programmed into the contract,
  - it can distribute that ether to other accounts or even other smart contracts.
- Example:
  - Alice wants to hire Bob to build her a patio
  - they are using an escrow contract (a place to store money until a condition is fulfilled) to store their ether before the final transaction.

# Smart Contract in Pictures

---



# Language of Smart Contracts in Ethereum

---

- Smart Contracts for Ethereum are written in Solidity
  - [Solidity](#) is statically typed
  - supports inheritance, libraries, and complex user-defined types
  - Similar to Javascript syntactically
- To learn solidity go to <https://remix.ethereum.org> and you can start programming smart contracts without having to create your own Ethereum network



# Some basic Ideas in Solidity

---

- Meant to execute as bytecode after compilation on Ethereum EVM
- EVM has a stack and Memory Model
  - 32 byte instruction word size
  - Access to program “stack” – like a register space where memory addresses may be stored to make program counter loop/jump
  - An expandable temporary “memory”
  - More permanent “storage” which is actually written into permanent block chain as program states
  - NO non-determinism allowed (e.g., no random() like function calls)

# Program Execution

---

- When an Ethereum block is “mined”,
  - the [smart-contract](#) deployments and function calls within that block get executed on the node that mines the block
    - the new state changes to any storage spaces or transactions within that smart-contract actually occur on that miner node.
  - As the new block gets propagated to all the other nodes
    - Each node tries to independently verify the block,
    - Verifying includes doing those same state changes to their local copy of the blockchain
  - it will fail if the [smart-contract](#) acts non-deterministically.
    - If the other nodes cannot come to a consensus about the state of blockchain after the new block and its contracts get executed, the network could literally halt.

## Program Execution (2)

---

- EVM smart-contracts cannot access data outside the “memory”, and “storage”
  - (we don’t want the smart-contract to be able to read or delete the hard-drives of the nodes it runs on)
- Cannot query outside resources like with a JQuery.
- Do not have access to many library functions like for parsing JSON structures or doing floating-point arithmetic,
  - it’s actually cost-prohibitive to do those sub-routines or store much data in the Ethereum blockchain itself.

## Program Execution (3)

---

- When you call a smart-contract that does some state-changing work or computation, you will incur a **gas “cost”** for the work done by the smart contract
  - this gas cost is related to the amount of computational work required to execute your function.
  - sort of a “micropayment for microcomputing” system, where you can expect to pay a set amount of gas for a set amount of computation, forever.
- The price of gas itself is meant to stay generally constant, meaning that when Ether goes up on the global markets, the **price** of gas against Ether should go down.
- When you execute a function call to a smart-contract, you can get an estimation of the amount of gas you must pay beforehand, but you must also specify the **price** (in ether per gas) that you are willing to pay
  - the [mining nodes](#) can decide if that’s a good enough rate for them to pick up your smart-contract function call in their next block.

# Addresses of Solidity Smart Contracts

- Smart-contracts have their own address, from which they can receive and send Ether.
- Smart contracts can track the “caller” of the function in a verifiable way,
  - it can determine if one of its functions is being called by a privileged “owner” or “admin” account, and act accordingly for administrative functions.
- They have the ability to read data from the Ethereum blockchain, and access info on transactions in older blocks.

# Getting data outside of Blockchain

---

- But are smart-contracts “locked in” into their own little deterministic world, only able to be aware of data stored in the Ethereum blockchain itself?
  - We can make a call to **an oracle** that will tell us something about the outside world in a trustable way, and *act on that data* within the smart contract.
  - even though real-world events themselves are not deterministic, the Oracle can be trusted to always answer every node’s request about what happened in a deterministic way
    - so that all nodes can still come to a consensus.
- An “oracle” will take some data, say a ticker price feed about a real-world stock price, and record that data into “storage” in a simple Oracle smart-contract,

# Ethereum networks

---

- On the **MainNet**, data on the chain—including account balances and transactions—are public, and anyone can create a node and begin verifying transactions.
- Ether on this network has a market value and can be exchanged for other cryptocurrency or fiat currencies like US Dollars.
- But there are other Ethereum networks as well.

## Other Ethereum Networks

---

- The Ethereum blockchain can be simulated locally for development.
- Local test networks process transactions instantly and Ether can be distributed as desired.
- An array of Ethereum simulators exist;
  - [Ganache](#)
- Developers use public test networks (or testnets) to test Ethereum applications before final deployment to the main network.
  - Ether on these networks is used for testing purposes only and has no value.



# Public TestNets

---

- **Ropsten:** The official test network, created by The Ethereum Foundation. Its functionality is similar to the MainNet.
- **Kovan:** A network that uses a consensus method called "proof-of-authority".
  - This means its transactions are validated by select members, leading to a consistent four second block time.
  - The supply of ether on this testnet is also controlled to mitigate spam attacks.
- **Rinkeby:** A testnet also using proof-of-authority, created by The Ethereum Foundation.

# Private/Enterprise Networks

---

- Private Ethereum networks allow parties to share data without making it publicly accessible.
- A private blockchain is a good choice for:
  - Sharing of sensitive data, such as health care records
  - Scaling to handle higher read/write throughput, due to the smaller network size
- An example of a private enterprise blockchain is [Quorum](#), originally written by J.P. Morgan.

# Distributed Applications (Dapps)

---

- **Applications using smart contracts for their processing are called "distributed applications", or "dapps".**
- The user interfaces for these dapps consist of familiar languages such as HTML, CSS, and JavaScript.
- The application itself can be hosted on a traditional web server or on a decentralized file service such as [Swarm](#) or [IPFS](#).
- Dapps based solution available for:
  - Record keeping
  - Finance
  - Supply chains
  - Real estate
  - Marketplaces

# Ethereum Design Principles

---

- **Sandwich Complexity Model**

- Complexity is handled by high-level-language compilers, argument serialization and deserialization scripts, storage data structure models, the leveldb storage interface and the wire protocol,

- **Freedom (inspired by net neutrality)**

- users should not be restricted in what they use the Ethereum protocol for, and we should not attempt to preferentially favor or disfavor certain kinds of Ethereum contracts or transactions based on the nature of their purpose.

- **Generalization**

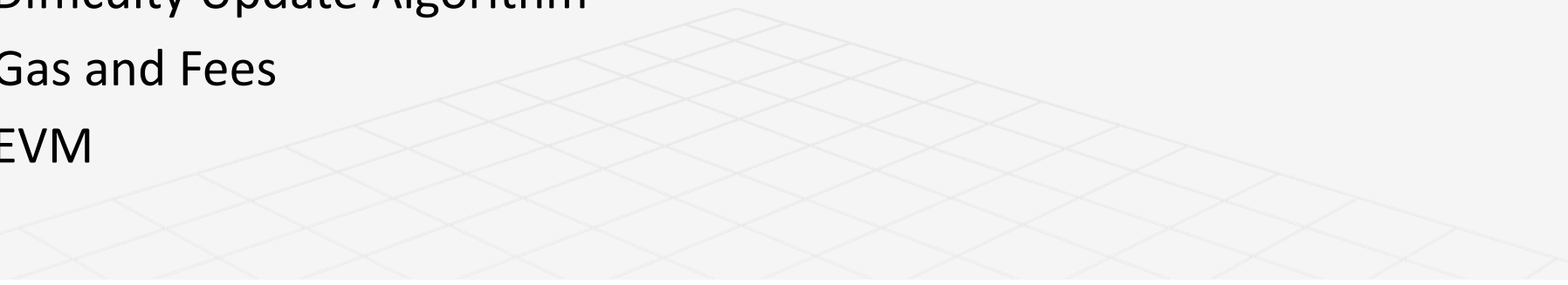
- protocol features and opcodes in Ethereum should embody maximally low-level concepts, so that they can be combined in arbitrary ways including ways that may not seem useful today but which may become useful later,

- **Low in high level features:**

- a corollary to generalization, we often refuse to build in even very common high-level use cases as intrinsic parts of the protocol, with the understanding that if people really want to do it they can always create a sub-protocol (eg. ether-backed subcurrency, bitcoin/litecoin/dogecoin sidechain, etc) inside of a contract.

# Some Differences with Bitcoin blockchain

---

- Accounts and not UTXOs
  - Merkle Patricia Trees
  - RLP – Recursive Length Prefix – main serialization format
  - Compression Algorithm
  - Trie Usage
  - Uncle Incentivization
  - Difficulty Update Algorithm
  - Gas and Fees
  - EVM
- 

# Accounts and not UTXOs

---

- Bitcoin stores data about users' balances in a structure based on *unspent transaction outputs* (UTXOs):
  - the entire state of the system consists of a set of "unspent outputs"
    - such that each coin has an owner and a value, and a transaction spends one or more coins and creates one or more new coins
- Transaction validity
  - Every referenced input must be valid and not yet spent
  - The transaction must have a signature matching the owner of the input for every input
  - The total value of the inputs must equal or exceed the total value of the outputs
- A user's "balance" in the system is thus the total value of the set of coins for which the user has a private key capable of producing a valid signature.

# Accounts vs UTXOs

---

- In Ethereum, the state stores a list of accounts
  - each account has a balance, as well as Ethereum-specific data (code and internal storage)
- a transaction is valid if the sending account has enough balance to pay for it,
  - the sending account is debited and the receiving account is credited with the value.
- If the receiving account has code
  - the code runs, and internal storage may also be changed,
  - or the code may even create additional messages to other accounts which lead to further debits and credits

# Accounts vs UTXOs

---

- Benefits of UTXOs
  - **Higher degree of privacy:** if a user uses a new address for each transaction that they receive then it will often be difficult to link accounts to each other
  - **Potential scalability paradigms:**
    - UTXOs are more compatible with certain kinds of scalability paradigms,
    - we can rely on only the owner of some coins maintaining a Merkle proof of ownership,
    - if everyone including the owner decides to forget that data then only the owner is harmed.



# Accounts vs. UTXOs (Benefits of Accounts)

---

- **Large space savings:**

- if an account has 5 UTXO, then switching from a UTXO model to an account model would reduce the space requirements
- from  $(20 + 32 + 8) * 5 = 300$  bytes (20 for the address, 32 for the txid and 8 for the value) to  $20 + 8 + 2 = 30$  bytes (20 for the address, 8 for the value, 2 for a nonce)
- Transactions can be smaller (e.g., 100 bytes in Ethereum vs. 200-250 bytes in Bitcoin) because
  - every transaction need only make one reference and one signature and produces one output.

## Benefits of Accounts (2)

---

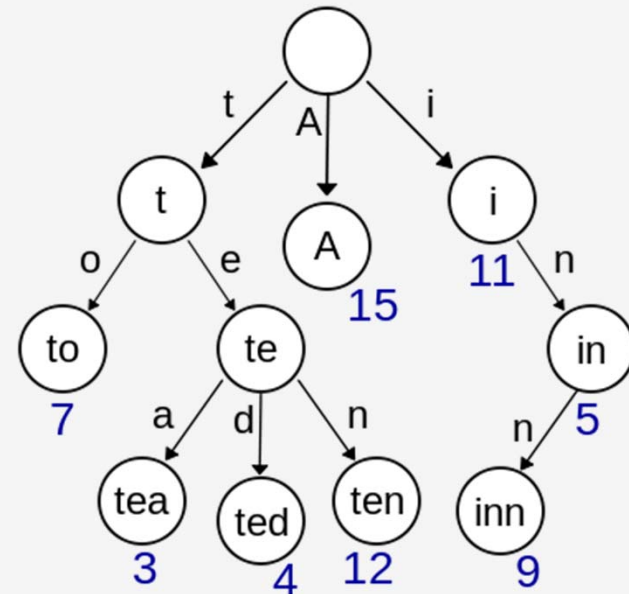
- **Greater fungibility:**
  - there is no blockchain-level concept of the source of a specific set of coins, it becomes less practical to institute a redlist/blacklisting scheme
  - to draw a distinction between coins depending on where they come from.
- **Simplicity:**
  - easier to code and understand, especially once more complex scripts become involved.
- **Constant light client reference:**
  - light clients can at any point access all data related to an account by scanning down the state tree in a specific direction.
- **One problem is that in order to prevent replay attacks,**
  - every transaction must have a "nonce", such that the account keeps track of the nonces used and only accepts a transaction if its nonce is 1 after the last nonce used.

# Patricia Tree

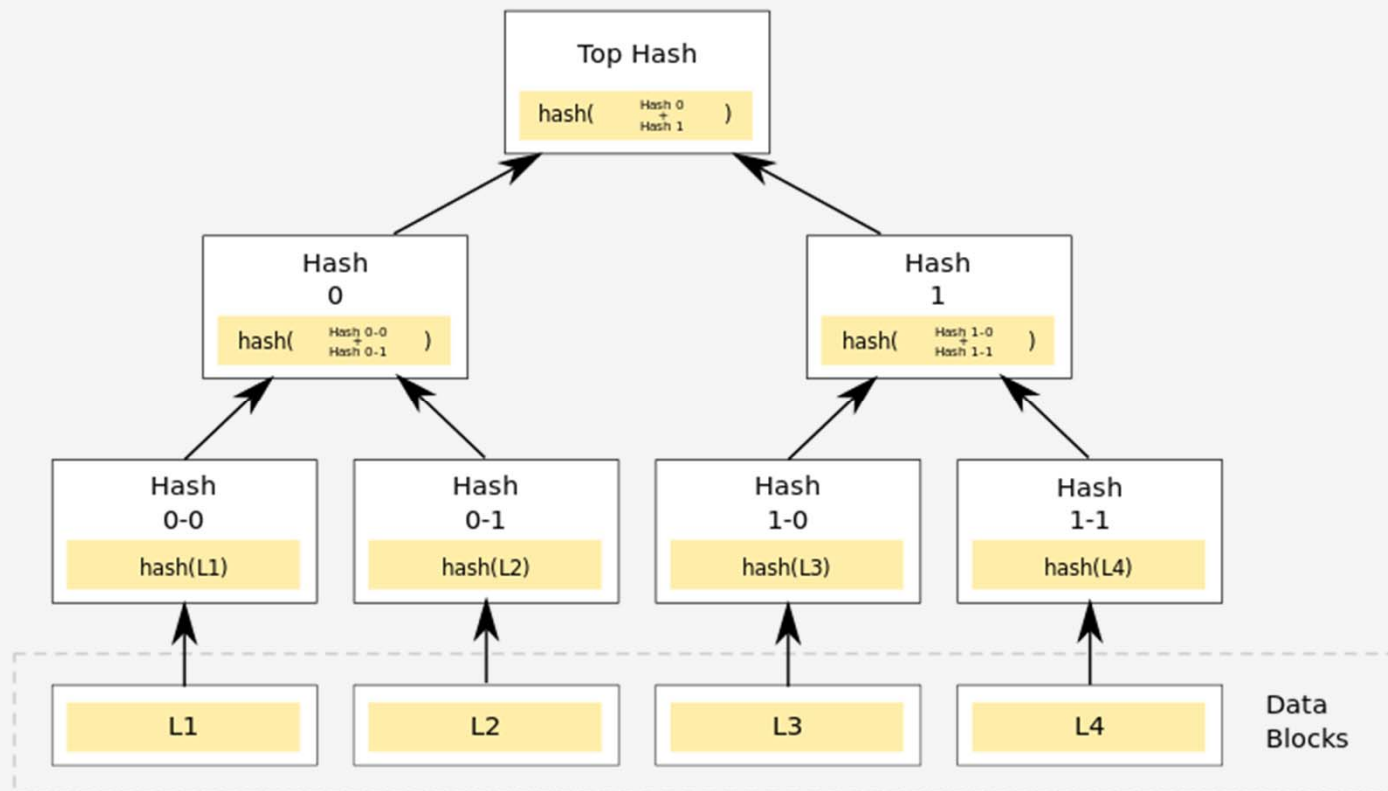
---

## Prefix tree/Radix tree/Trie

- Trie uses a key as a path so the nodes that share the same prefix
- This structure is fastest at finding common prefixes, and requires small memory.
- commonly used for implementing routing tables, systems that are used in low specification machines like the router.

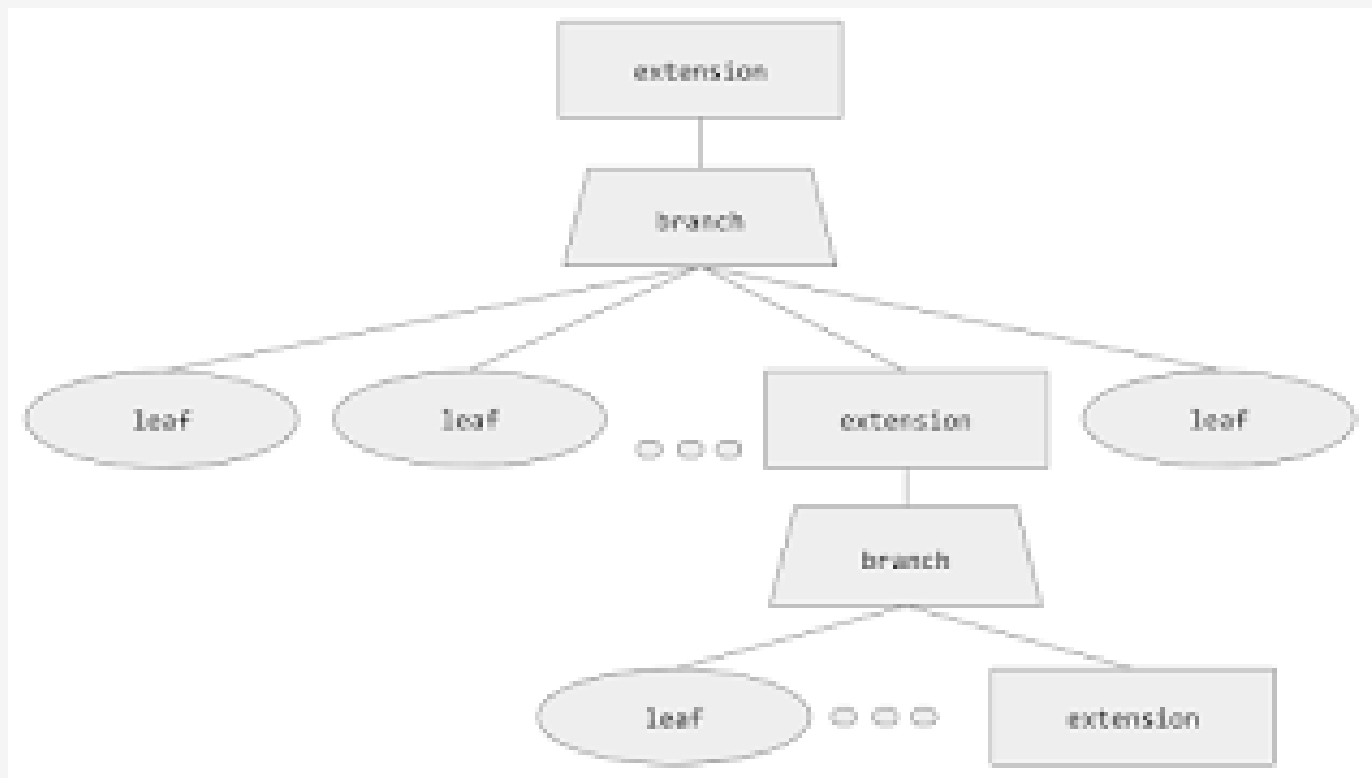


# Merkle Tree



# Merkle Patricia Tree (MPT)

---



## Now some actual work (assuming ubuntu)

---

- > sudo apt-get update
- > sudo apt-get nodejs
- > sudo apt-get install npm
- > npm -v (make sure 5.0+)
- > sudo npm install -g truffle
- > sudo npm install -g ganache-cli
- > sudo ganache-cli

```
william@william-VirtualBox:~$ ganache-cli
Ganache CLI v6.1.3 (ganache-core: 2.1.2)
```

#### Available Accounts

```
=====
(0) 0xa6df0f09b87787cb8532ac90d603600708ddca23
(1) 0xf3980a037b5bc2c55b4bf20ef519f3e51847a4e
(2) 0xf2c2c2df9acf60b20b9bb4baa4108821a4a7f6ba
(3) 0x2be6125a439342d7a5f767957fa7ca9c9be357de
(4) 0x9e3e2240dca5e02284759438b05dcaa535e0f166
(5) 0x03c2b9f7275c2fba0c90aeb24cf0660d06908b30
(6) 0xb75c95dbbb359526cbdac8bfcc3d33d3fc4006c0
(7) 0x51509f625bc11638826d1349aa142e95c4f6c003
(8) 0x37c5d4abfdd92d56ce21f5cfcacb907f49f4a08d
(9) 0x24d2b74e63b83382750a53a0870977c14ccbdf07
```

#### Private Keys

```
=====
(0) c014d77f61a413d909d5d33142013b3c924085364de618aa5b94c604bf092770
(1) ebbff0e4394a42632683d2fd5ed33d31dfbc51f83f5c09be4f0d3eecbf91eb71
(2) 5ed898e70aaefld1a148432d67c89b9707e94ac830958c14961ed4034eb1e4e
(3) d73735195314d1885a173f92772240780b93b8769f57dd218ae8170b2283272b
(4) 80165a637888ad8e95e14b78f625b61087ad5a80cf7ab20c6fa7ff765acef176
(5) 80c42d081bb6b1a0d25a91c23010142b67d210820d8804adbd0cd0fc17806163
(6) e9d661c3b05d05588f78ef9f2e35faacc874f2ca14dd30d887dd90e1635c9018
(7) f90ebc1c0ff8fa08305dee613aaf65e51a2d09d7f9d19d1cf1a0371a5ff1f72a
(8) 2b9af99fb51f4ced631091f6392e80fb24185abedeabb1e85af1a95b5ee52f8e
(9) af1defd9a5cb596b2982481e804699633b99a84057100be9dbe0a94ce635bbb1
```

#### HD Wallet

```
=====
Mnemonic:      rookie mistake degree announce episode bright result nation tomorrow pond outer force
Base HD Path:  m/44'/60'/0'/0/{account_index}
```

#### Gas Price

```
=====
20000000000
```

#### Gas Limit

```
=====
6721975
```

```
Listening on localhost:8545
```

# HelloWorld smart contract

---

- > mkdir SmartContractDemo
  - > cd SmartContractDemo
  - > mkdir HelloWorld
  - > cd HelloWorld
  - > truffle init
- 



# Truffle init

---

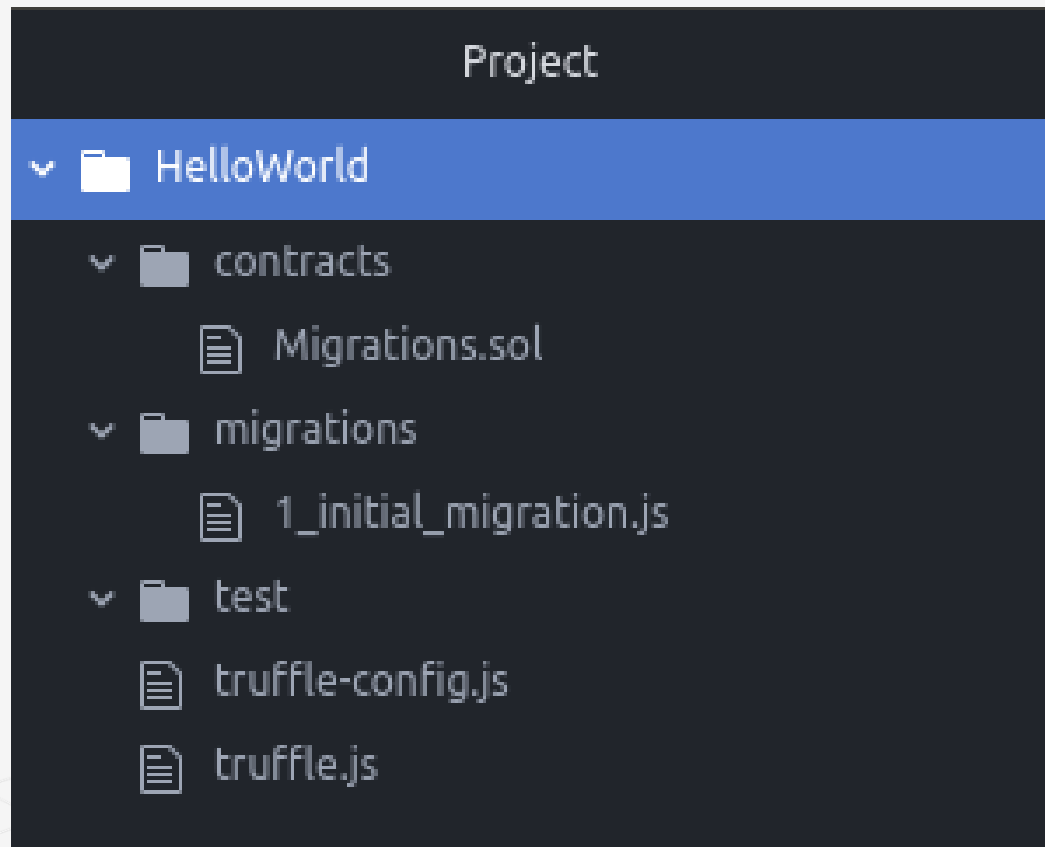
```
william@william-VirtualBox:~/ethereum$ mkdir SmartContractDemo
william@william-VirtualBox:~/ethereum$ cd SmartContractDemo/
william@william-VirtualBox:~/ethereum/SmartContractDemo$ mkdir HelloWorld
william@william-VirtualBox:~/ethereum/SmartContractDemo$ cd HelloWorld/
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ truffle init
Downloading...
Unpacking...
Setting up...
Unbox successful. Sweet!

Commands:

  Compile:      truffle compile
  Migrate:      truffle migrate
  Test contracts: truffle test
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ ls
contracts migrations test truffle-config.js truffle.js
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ █
```

# Effect of unbox

---



## Directory structure created by unbox

- ***/contracts***: store original codes of the smart contract. We will place our **HelloWorld.sol** file here.
- ***/migrations***: deploy the smart contract in the “*contracts*” folder.
- ***/test***: test codes for your smart contract, support both JavaScript and Solidity.
- ***truffle.js***: configuration document.
- ***truffle-config.js***: configuration document for windows user. We can just leave it alone.  
if your Ubuntu 16.04 is running on a virtual machine under Windows, then your configuration should also go here

# Creating contract

---

- > truffle create contract HelloWorld

```
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ truffle create contract HelloWorld
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ cd contracts/
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld/contracts$ ls
HelloWorld.sol  Migrations.sol
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld/contracts$
```

# HelloWorld.sol

---

```
pragma solidity ^0.5.0;  
  
Contract HelloWorld {  
    function hi() public pure returns (string memory) {  
        return ("Hello World");  
    }  
}
```

**> truffle compile**

# Compiling and deploying contract

---

- Compilation creates bytecode for EVM
- .json file in build/contracts directory as HelloWorld.json
- Now time to deploy the contracts
- Create a new file under migrations directory
  - 2\_deploy\_contracts.js

```
var HelloWorld=artifacts.require (“./HelloWorld.sol”);

module.exports = function(deployer) {

    deployer.deploy(HelloWorld);

}
```

# Configuring to deploy on local Ethereum network

---

```
15  module.exports = {  
16    networks: {  
17      development: {  
18        host: "localhost",  
19        port: 8545  
20      }  
21    }  
22  };  
23
```

**> truffle migrate**

# Migration in work

---

```
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x942a4817ab47ebdbae62d86552656f427f5efb92b60d67ce2afb86963d5d4fa0
  Migrations: 0x84992eff29fbac77a60f5470e48ba767718b8f6c
Saving successful migration to network...
  ... 0x4dcd3528ba8ee21ec2e81cac9e2aabe209b53fe03000160c80e83e853c4df6c4
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying HelloWorld...
  ... 0x9b1066d86effeba988c111542f2d1ae55864fa6d9bc017708ba7850bd43f0e40
  HelloWorld: 0xd679d052b3eeefb041d9393fb9ceeb7145b0a173
Saving successful migration to network...
  ... 0x1c6a33d2cb74baf0b0e51ecfffe5b79a78de636d949715fc7e85b33bd591fc62
Saving artifacts...
```



# Effect of the deployment on the network

---

```
net_version
eth_accounts
eth_accounts
net_version
net_version
eth_sendTransaction

Transaction: 0x942a4817ab47ebdbae62d86552656f427f5efb92b60d67ce2afb86963d5d4fa0
Contract created: 0x84992eff29fbac77a60f5470e48ba767718b8f6c
Gas usage: 277462
Block Number: 1
Block Time: Fri Jun 22 2018 16:39:21 GMT-0700 (PDT)

eth_newBlockFilter
eth_getFilterChanges
eth_getTransactionReceipt
eth_getCode
eth_uninstallFilter
eth_sendTransaction

Transaction: 0x4dcd3528ba8ee21ec2e81cac9e2aabe209b53fe03000160c80e83e853c4df6c4
Gas usage: 42008
Block Number: 2
Block Time: Fri Jun 22 2018 16:39:22 GMT-0700 (PDT)

eth_getTransactionReceipt
eth_accounts
net_version
net_version
eth_sendTransaction

Transaction: 0x9b1066d86effeba988c111542f2d1ae55864fa6d9bc017708ba7850bd43f0e40
Contract created: 0xd679d052b3eeefb041d9393fb9ceeb7145b0a173
Gas usage: 136299
Block Number: 3
Block Time: Fri Jun 22 2018 16:39:22 GMT-0700 (PDT)
```

# Summary

---

- In this lecture, you learnt the design philosophy of Ethereum Blockchain
- Some Basic differences between Bitcoin blockchain and Ethereum blockchain
- The concept of smart contracts
- The concept of Ethereum network and simulation of such networks for testing
- Finally, clue about how to simulate a local private network, and try out a simple smart contract on your own