

Ethereum

Overview

- Highly recommended intro
- <https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369>

Medium



Preethi Kasireddy [Follow](#)

Blockchain Engineer. I have a passion for understanding things at a fundamental level and sharing it as clearly as possible.

Sep 27, 2017 · 33 min read

How does Ethereum work, anyway?

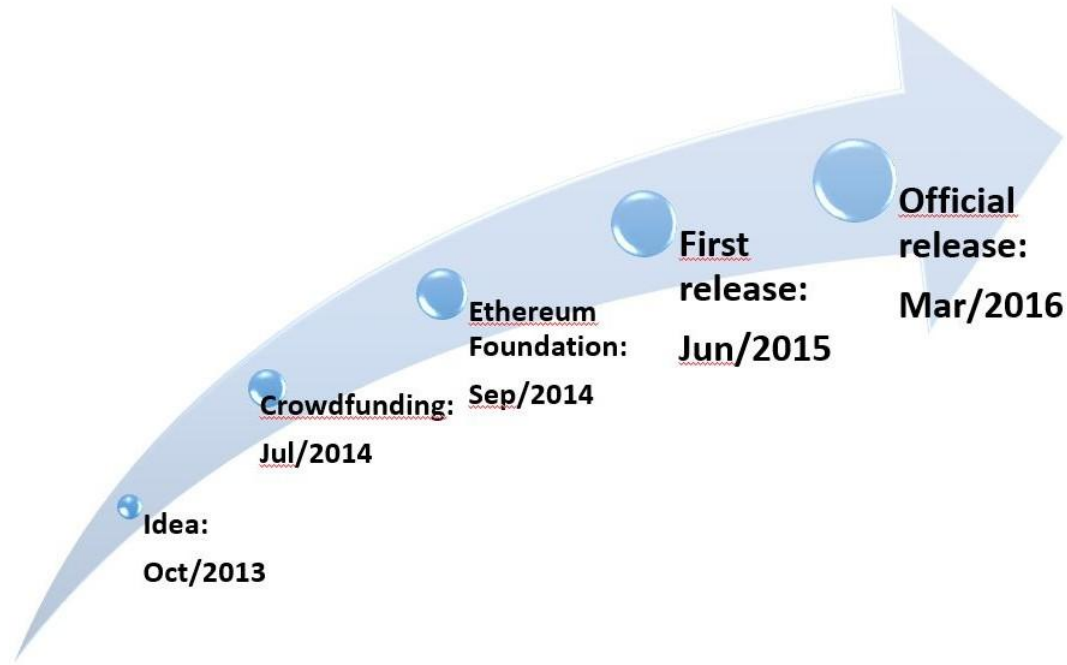


History of Ethereum



- Russian-Canadian programmer
- Co-founded Ethereum when he was 19 years old

History of Ethereum - Timeline



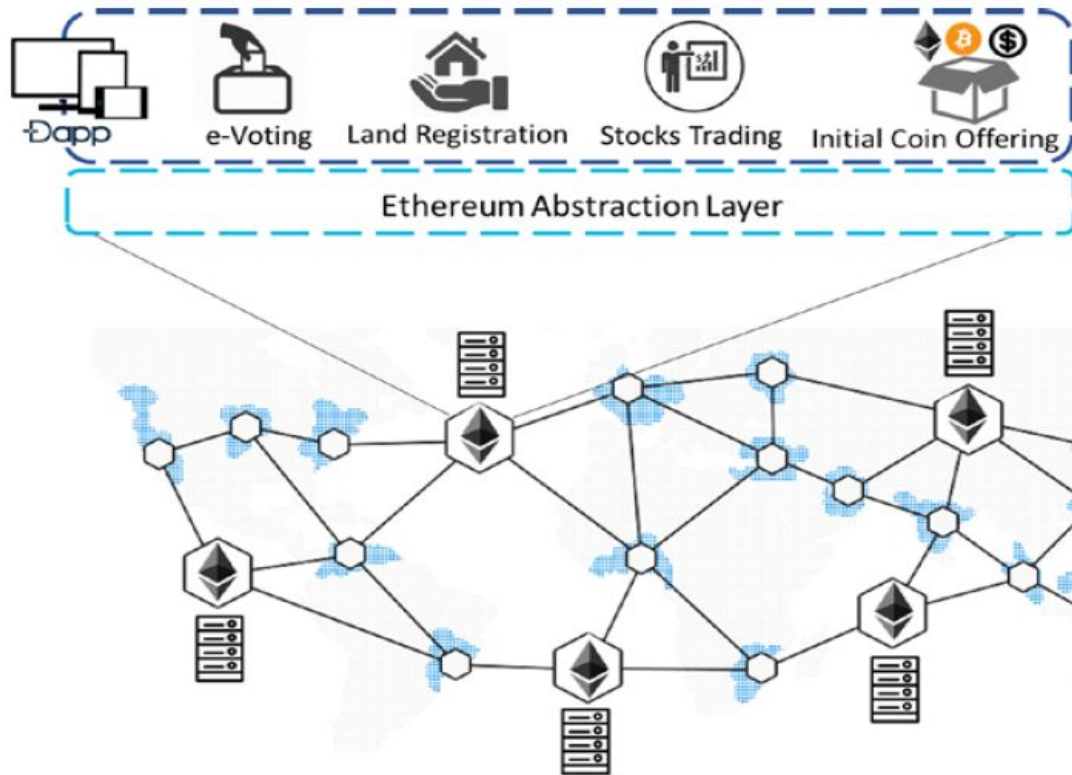
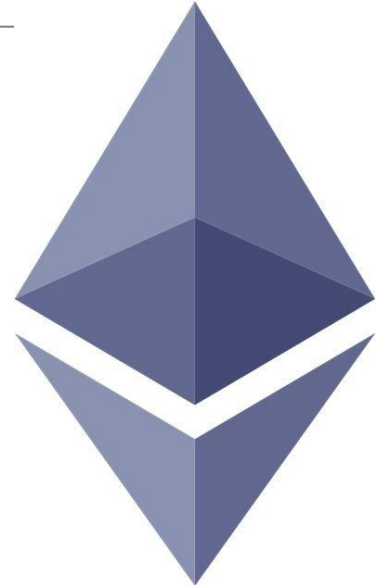


Figure 4-1. Multiple decentralized applications on one Ethereum platform

Important Concepts

- Cryptography (similar to Bitcoin)
- Blockchain
 - Accounts (Two types) and Wallets
 - Transactions
- Smart Contracts
 - Solidity
 - Language Used for Smart Contract Development



Cryptography

- Hash functions
- Symmetric Cryptography
- Asymmetric Cryptography
- Signatures

Hash Functions

- BTC uses SHA-256
- Ethereum uses Keccak-256
 - Similar to SHA-3 (variant)
 - Won contest for security in 2007
 - Used for all hashing in Ethereum
 - Derived differently than standard block-cipher based hashes or previous SHA functions

Digital Signatures (Digital Proof)

- Same use-case/cryptographic method (ECDSA) as BTC
- Signer uses private key to generate a signed message
- Signed message can be verified using the signer's public key
- Hashes are signed in Ethereum, not the data itself

Blockchain

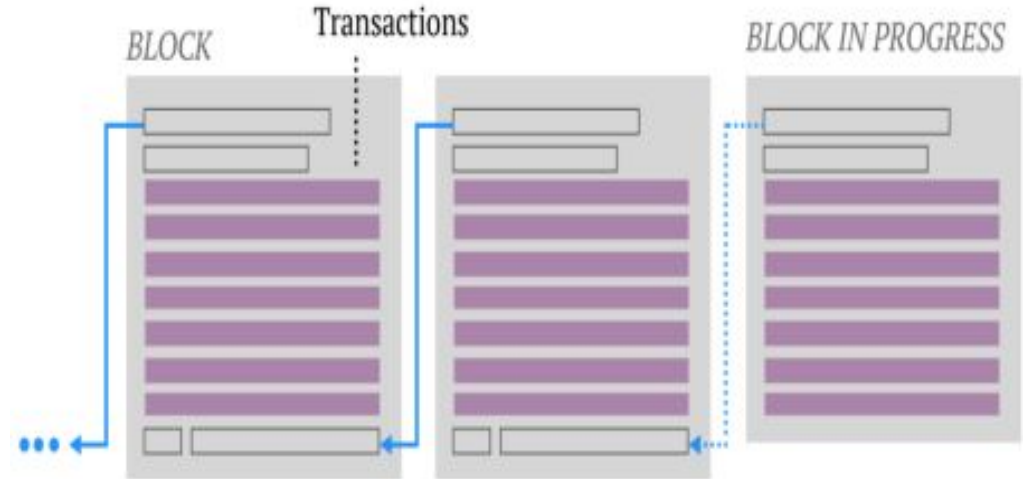
Fully Distributed Database like BTC

Advantages:

- Highly Secure
- Transparent
- Immutable

Disadvantages:

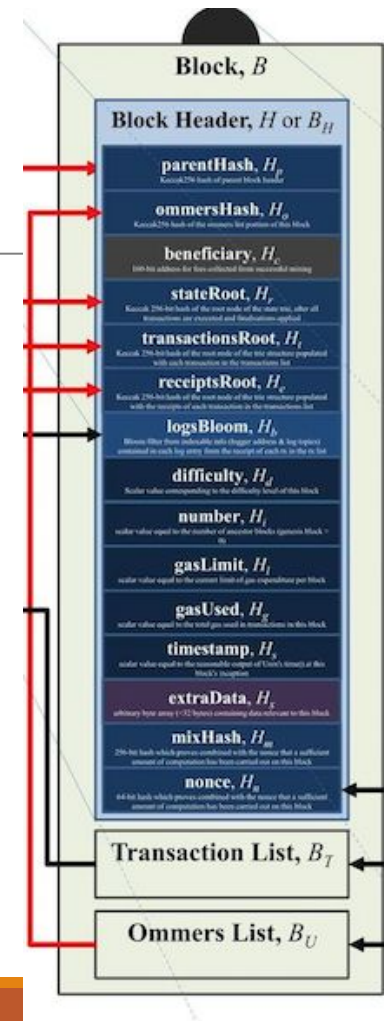
- Scaling
- Performance



Ethereum Blockchain

Blocks consist of 3 elements

- Transaction List
 - List of all transactions included in a block
- Block Header
 - Group of 15 elements
- Ommer List
 - List of all Uncle blocks included



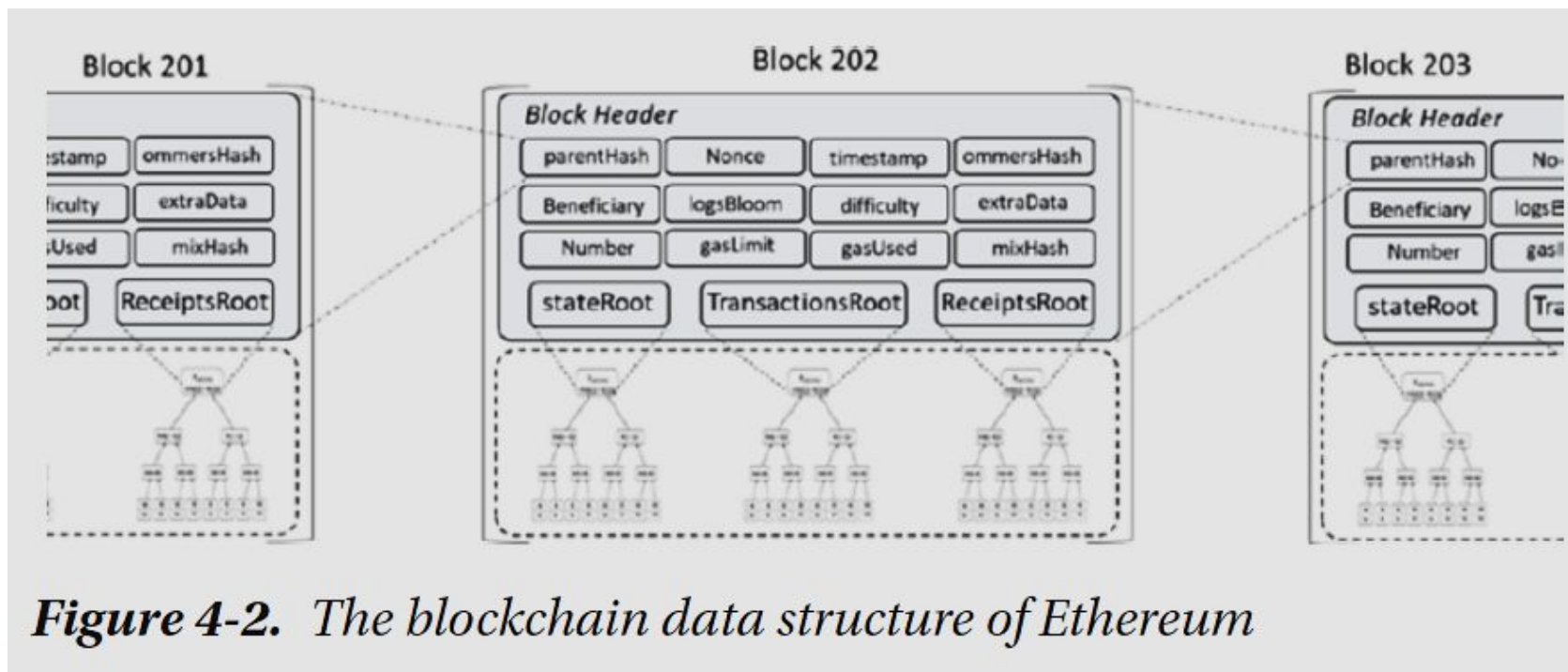


Figure 4-2. The blockchain data structure of Ethereum

In Bitcoins, there was only one Merkle root in the block header for all the transactions in a block. In Ethereum, there are two more Merkle roots, so there are three Merkle roots in total as follows:

- **stateRoot**: It helps maintain the global state.
- **transactionsRoot**: It is to track and ensure integrity of all the transactions in a block, similar to Bitcoin's Merkle root.
- **receiptsRoot**: It is the root hash of the receipts *trie* corresponding to the transactions in a block

Merkle Patricia Tree

- Merkle trees facilitate efficient and secure verification of the contents in decentralized systems.
- Instead of downloading every transaction and every block, the light clients can only download the chain of block headers, that is, 80-byte chunks of data for each block that contain only **five** things:
- **hash of the previous block header, timestamp, mining difficulty, nonce value that satisfied PoW, and the root hash of the Merkle tree containing all the transactions for that block.**
- Merkle Patricia Trie is a data structure that stores key-value pairs, just like a map. In addition to that, it also allows us to verify data integrity and the inclusion of a key-value pair.

Ethereum Blockchain

Section-1: Block metadata

- **parentHash:** Keccak 256-bit hash of the parent block's header, like that of Bitcoin's style
- **timestamp:** The Unix timestamp current block
- **number:** Block number of the current block
- **Beneficiary:** The 160-bit address of "author" account responsible for creating the current block to which all the fees from successfully mining a block are collected

Section-2: Data references

- **transactionsRoot:** The Keccak 256-bit root hash (Merkle root) of the transactions trie populated with all the transactions in this block
- **ommersHash:** It is otherwise known as “uncleHash.” It is the hash of the uncles segment of the block, i.e., Keccak 256-bit hash of the ommers list portion of this block (blocks that are known to have a parent equal to the present block’s parent’s parent).
- **extraData:** Arbitrary byte array containing data relevant to this block. The size of this data is limited to 32 bytes

Section-3: Transaction execution information

- **stateRoot**: The Keccak 256-bit root hash (Merkle root) of the final state after validating and executing all transactions of this block
- **receiptsRoot**: The Keccak 256-bit root hash (Merkle root) of the receipts trie populated with the recipients of each transaction in this block
- **logBloom**: The accumulated Bloom filter for each of the transactions' receipts' Blooms, i.e., the "OR" of all of the Blooms for the transactions in the block
- **gasUsed**: The total amount of *gas* used through each of the transactions in this block
- **gasLimit**: The maximum amount of *gas* that this block may utilise (dynamic value depending on the activity in the network)

Section-4: Consensus-subsystem information

- **difficulty:** The difficulty limit for this block calculated from the previous block's difficulty and timestamp
- **mixHash:** The 256-bits mix hash combined with the 'nonce' for the PoW of this block
- **nonce:** The nonce is a 64-bit hash that is combined with mixHash and can be used as a PoW verification.

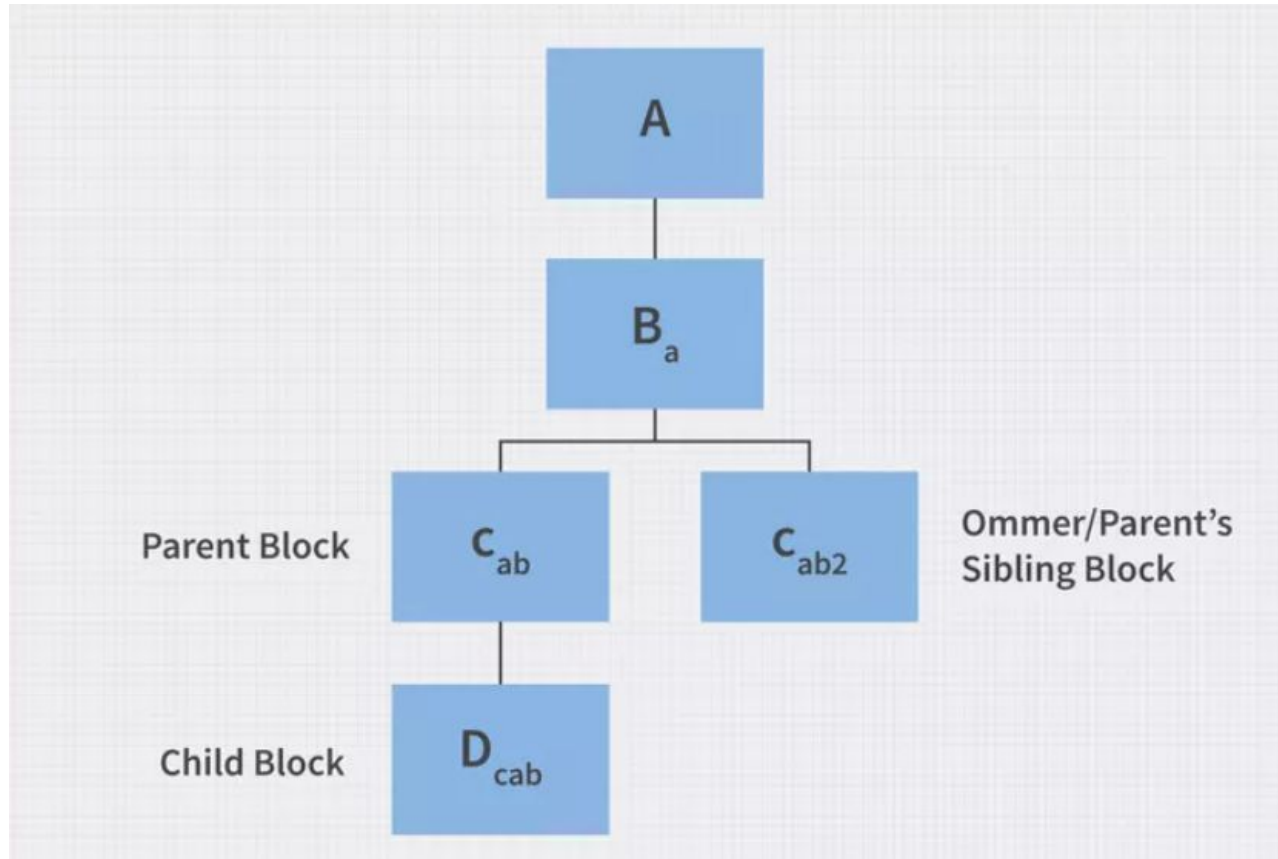
What Is an Ommer Block

It's possible for two blocks to be created simultaneously by a network. When this happens, one block will be left out. This leftover block is called an ommer block.

In the past, they were called uncle blocks, referring to the familial relationships used to describe block positions within a blockchain.

Ommers are created in the Ethereum blockchain when two blocks are created and submitted to the ledger at roughly the same time. Only one can enter the ledger.

Ethereum miners or validators are rewarded for creating ommer blocks in the Ethereum system through transaction fees to pay for their work.



Ethereum Blockchain

Uncles/Ommers

- Sometimes valid block solutions don't make main chain
 - Any broadcast block (up to 6 previous blocks back) with valid PoW and difficulty can be included as an uncle
 - Maximum of two can be included per block
- Uncle block transactions are not included – just header
- Aimed to decrease centralization and reward work

Ethereum Blockchain

Uncles/Ommers Rewards:

- Uncle headers can be included in main block for 1/32 of the main block miner's reward given to said miner
- Miners of uncle blocks receive percent of main reward according to:
 - $(U_n + (8 - B_n)) * 5 / 8$, where U_n and B_n are uncle and block numbers respectively.
 - Example $(1333 + 8 - 1335) * 5/8 = 3.75$ ETH


Ethereum Blockchain

- All blocks visible like BTC
- However, blocks have a different structure than BTC

<https://etherscan.io/>



HOME

Sponsored Link:  **SocialMedia.Market** - The most cost effective advertising platform with 1069



MARKET CAP OF \$94.839 BILLION
\$973.81 @ 0.1049 BTC/ETH (+7.52%)

LAST BLOCK

5024406 (14.0s)

TRANSACTIONS

153.76 M (10.9 TPS)

Hash Rate

228,803.79 GH/s

Network Difficulty

2,757.12 TH

 Blocks

View All

Block 5024406

> 1 min ago

Mined By [ethfans.org_2](#)

249 txns in 28 secs

Block Reward 3.18895 Ether

Block 5024405

> 1 min ago

Mined By [Nanopool](#)

189 txns in 7 secs

Block Reward 3.2864 Ether

Ethereum Blockchain

Blocks faster than BTC and reward is different

- Every 12 seconds
- 5 ETH main reward
- Miners can make a bit more by including uncle blocks (1/32 of an ETH each) up to maximum of two

Ethereum Blockchain

Blocks faster than BTC and reward is different

- Uses EthHash mining algorithm (different than Bitcoin)
 - Helps mitigate ASIC and GPU advantages
 - Involves smart contract execution
- Difficulty is adjusted every block (not every two weeks) – this is an important identifier for the Uncle blocks

Ethereum Blockchain

Key differences

- Blocks keep track of balances – not “unspent transaction outputs” like BTC
- Merkle-Patricia tries used (they have three branches compared to the Merkle tree’s two)
- Will transition from Proof of Work to Proof of Stake with Casper protocol

Ethereum

Nodes

- Validate all transactions and new blocks
- Operate in a P2P fashion
- Each contains a copy of the entire Blockchain
- Light clients - store only block headers
 - Provide easy verification through tree data structure
 - Don't execute transactions, used primarily for balance validation
- Implemented in a variety of languages (Go, Rust, etc.)

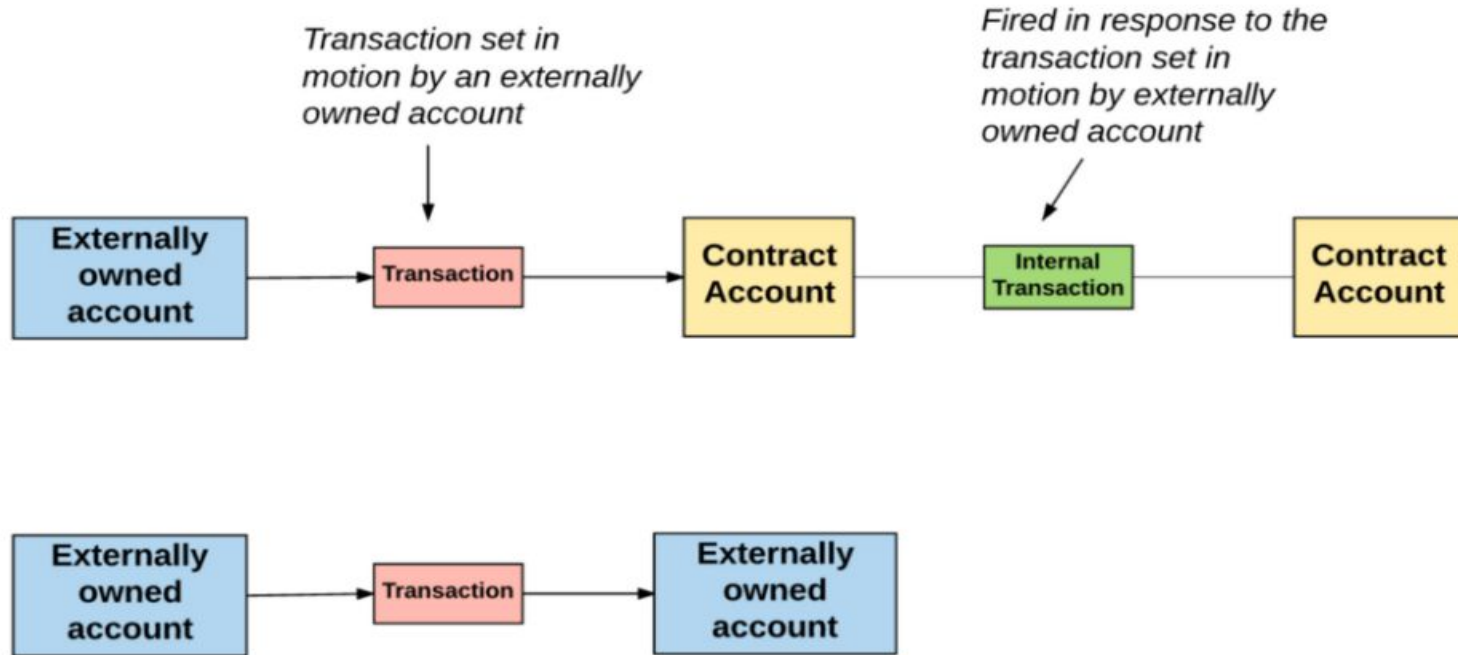
Accounts and Wallets

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



Accounts and Wallets

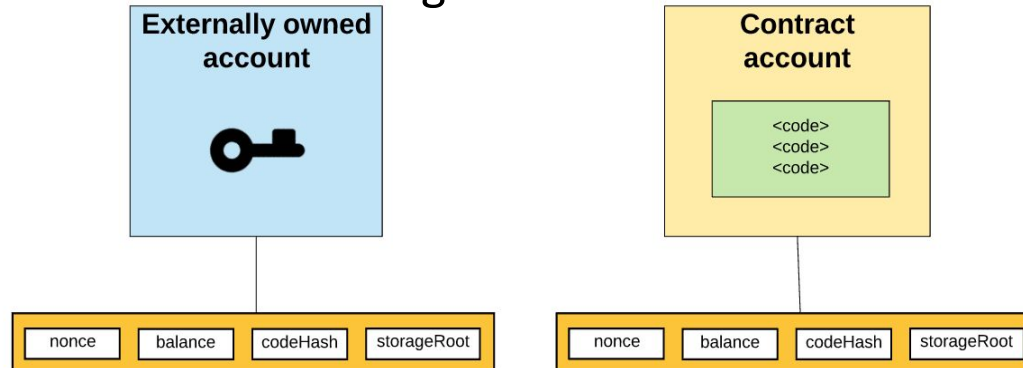
External Owned Account (EOA, Valid Ethereum Address)

- Has an associated nonce (amount of transactions sent from the account) and a balance
- codeHash - Hash of associated account code, i.e. a computer program for a smart contract (hash of an empty string for external accounts, EOAs)
- Storage Root is root hash of Merkle-Patricia trie of associated account data

Accounts and Wallets

Contract Account

- Ethereum accounts can store and execute code
 - Has an associated nonce and balance
 - codeHash - hash of associated account code storageRoot contains Merkle tree of associated storage data



Example Account

Private Key:

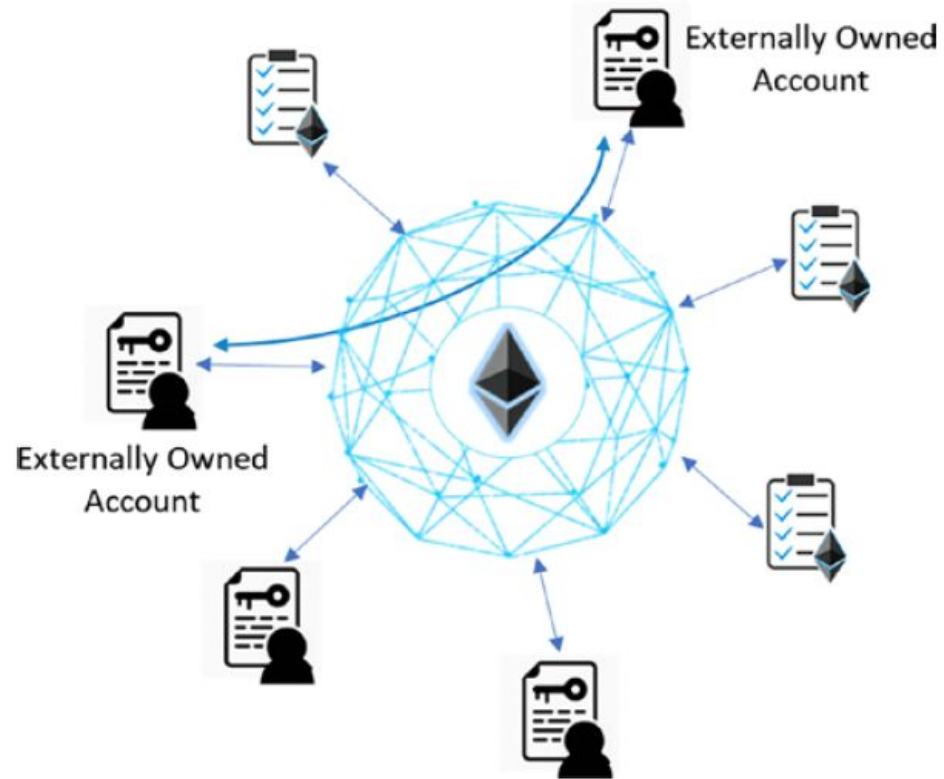
0x2dcef1bfb03d6a950f91c573616cdd778d9581690db1cc43141f7cca06fd08ee

- Ethereum Private keys are 66 character strings (with 0x appended). Case is irrelevant. Same derivation through ECDSA as BTC.

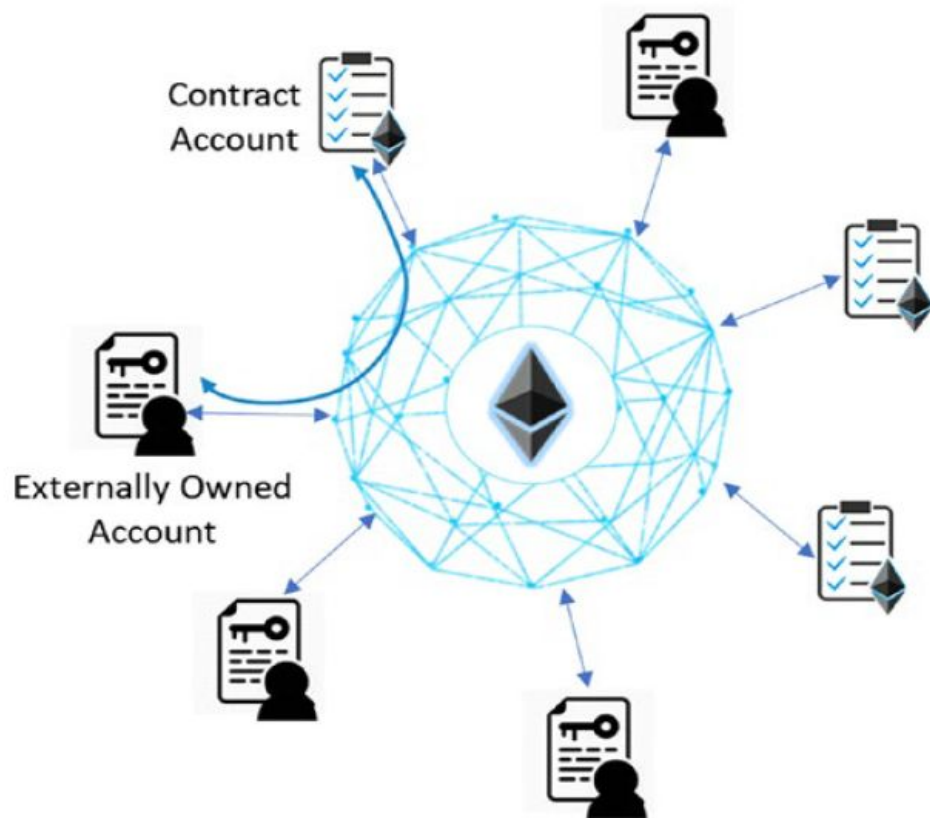
Address: 0xA6fA5e50da698F6E4128994a4c1ED345E98Df50

- Ethereum Private keys map to addresses directly. Simply the last 40 characters of the Keccak-256 hash of the public key. Address is 42 characters total (append 0x to front).

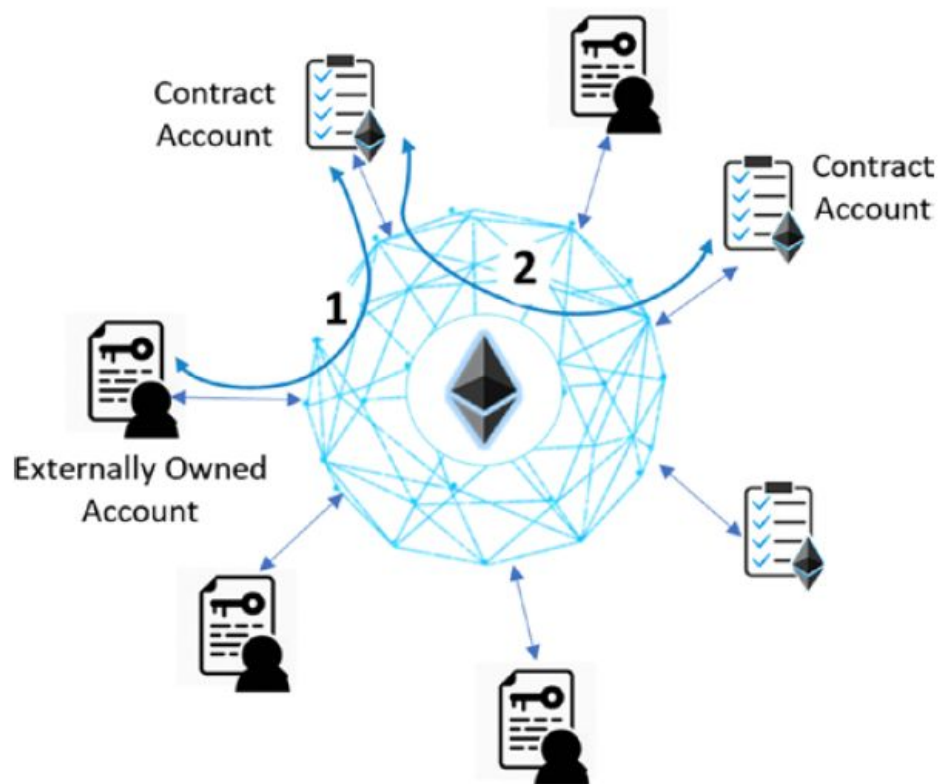
EOA to EOA transaction:



EOA to Contract Account Transaction:

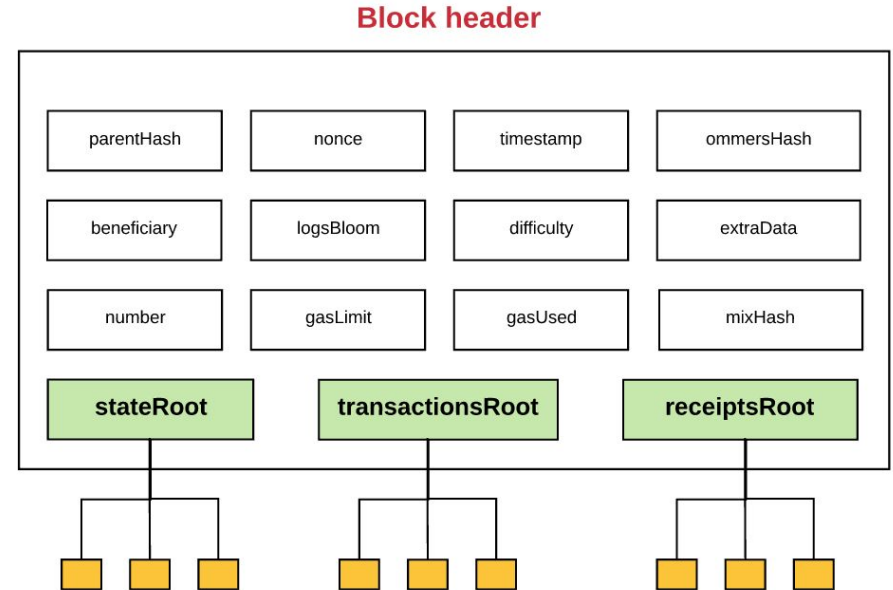


EOA to Contract Account to other Contract Account transaction



Ethereum Blockchain Header

- Hash of included ommer's stored in block header
 - State root is the hash of a merkle trie that holds all account information
 - Similar storage structure for transactions and receipts



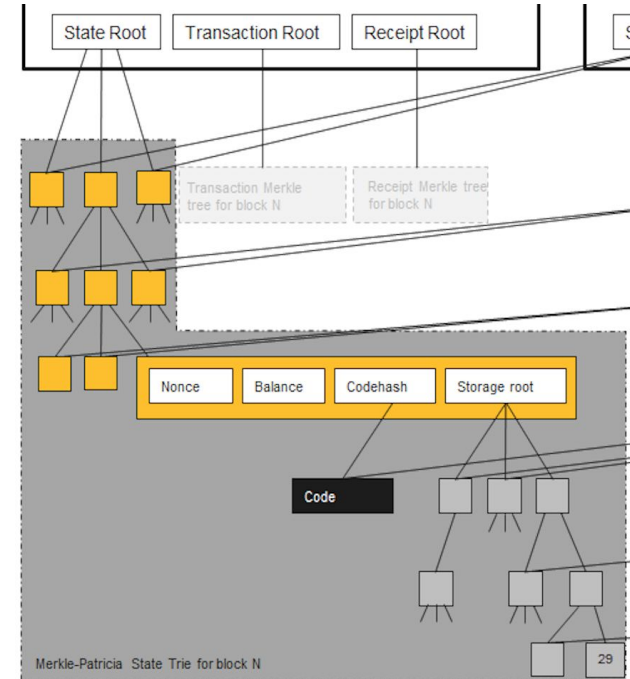
Ethereum Blockchain State

StateRoot, TransactionRoot, and ReceiptsRoot
Stored in data structure known as a
Merkle-Patricia trie

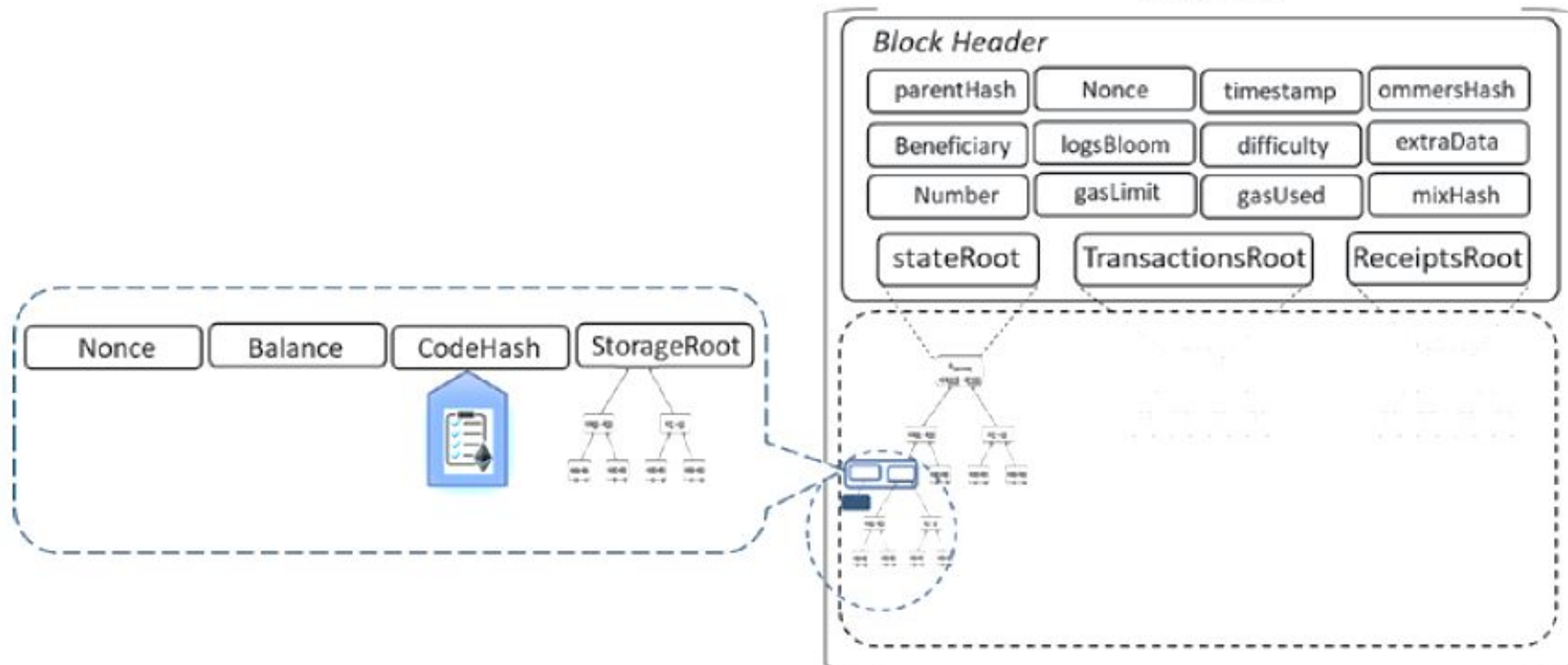
Similar to the Merkle trie used in BTC, but with
three leaves per node

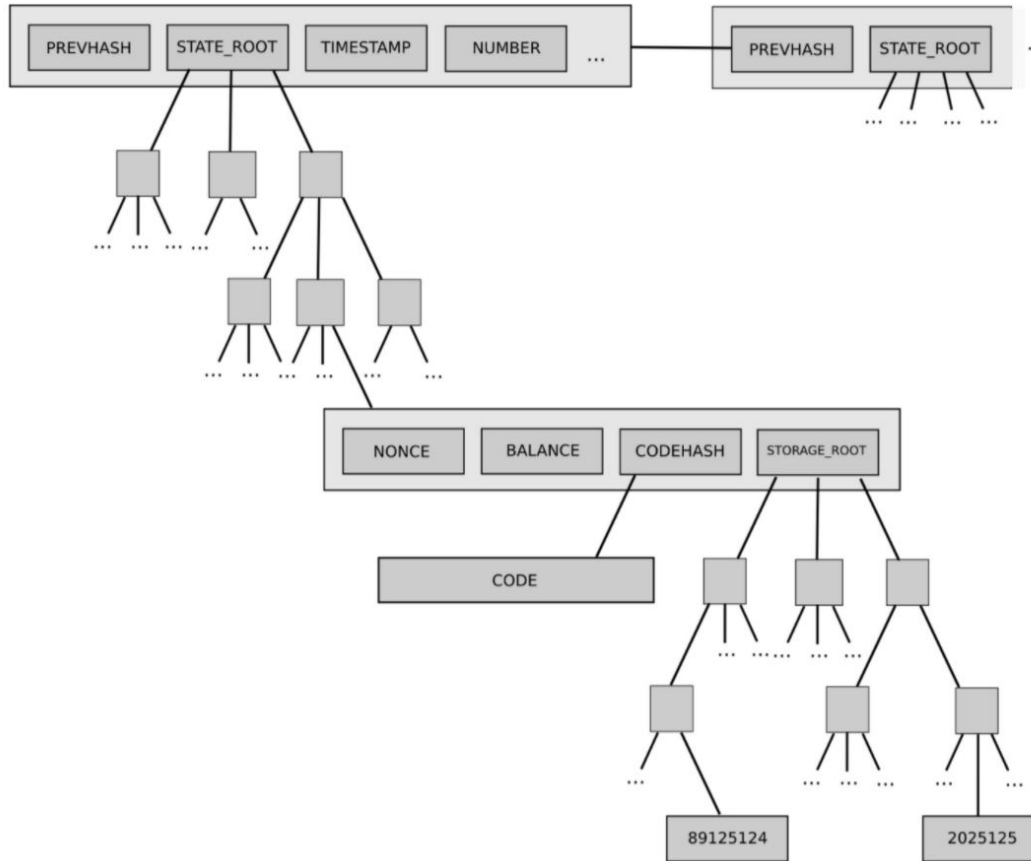
Trie is cryptographically secure as any
alteration of a leaf or intermediary
node results in a different root hash

<https://ethereum.org/sl/developers/docs/transactions/>



Block 202

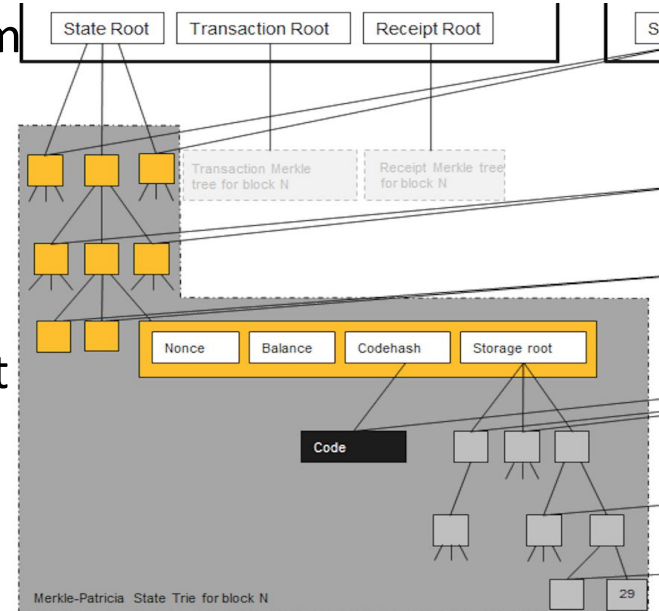




Ethereum Blockchain State

StateRoot

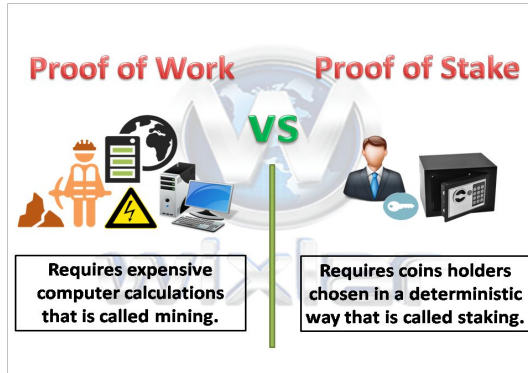
- Each node in the stateRoot trie represents an Ethereum address
- Each address has 4 components
 - Nonce - list of number of Tx's from address
 - CodeHash - hash of associated code
 - StorageRoot - Merkle-Patricia tree root of account storage contents
 - Balance - balance of account



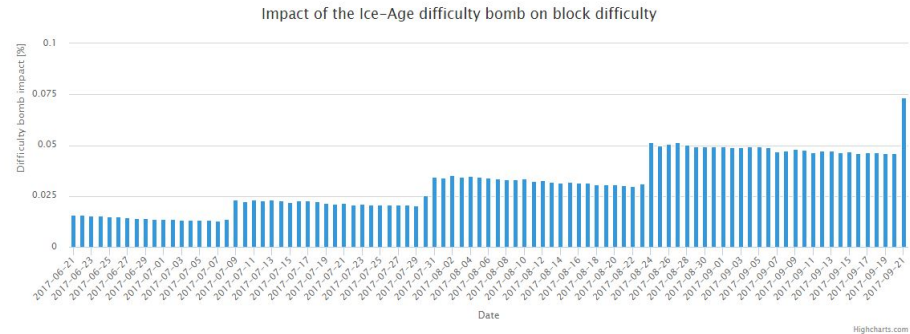
Ethereum Blockchain

Ethereum “difficulty bomb”

- Spike (increase) in mining difficulty
- Introduced to attempt to reduce number of miners
 - Aimed to pre-date shift of algorithm from PoW to Proof-of-Stake (PoS)




Difficulty bomb impact



Transactions

- A request to modify the state of the blockchain
 - Can run code (contracts) which change global state
 - Contrasts only balance updates in BTC
- Signed by originating account
- Types:
 - Send value from one account to another account
 - Create smart contract
 - Execute smart contract code

Transaction Information

TxHash:	0x67aac64c856be1ebe9a9c94a17894e77b16e42b2d11bd8c59af6a9013b0f661a
TxReceipt Status:	Success
Block Height:	5017471 (3 block confirmations)
TimeStamp:	1 min ago (Feb-02-2018 01:24:36 PM +UTC)
From:	0xd307aa93e9bfb5e757b5ae9afb07061edc1da81
To:	Contract 0x7b74c19124a9ca92c6141a2ed5f92130fc2791f2 
Value:	3.21373401 Ether (\$2,950.75)
Gas Limit:	23018
Gas Used By Txn:	23018
Gas Price:	0.000000055 Ether (55 Gwei)
Actual Tx Cost/Fee:	0.00126599 Ether (\$1.16)
Cumulative Gas Used:	986293
Nonce:	1
Input Data:	<div>0x</div>

Ether Denominations

- Wei - lowest denomination
 - Named after Wei Dai - author of b-money paper (1998), many core concepts used in BTC implementation
 - 1/1,000,000,000,000,000,000 (quintillion)
- Szabo - next denomination
 - Named after Nick Szabo - author of Bit-Gold
- Finney – 2nd highest denomination
 - Named after Hal Finney - received first Tx from Nakamoto

Multiplier	Name
10^0	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

Gas

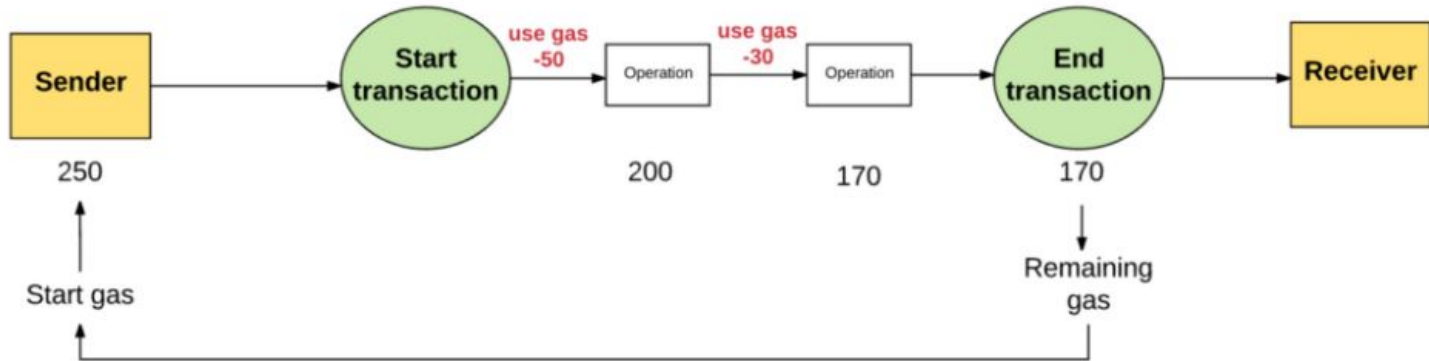
- **Every computation that occurs as a result of a transaction on the Ethereum network incurs a fee** —This fee is paid in a denomination called “gas.”
- **Gas** is the unit used to measure the fees required for a particular computation.
- **Gas price** is the amount of Ether you are willing to spend on every unit of gas, and is measured in “gwei.” “Wei” is the smallest unit of Ether, where 1^{018} Wei represents 1 Ether. One gwei is 1,000,000,000 Wei.
- With every transaction, a sender sets a **gas limit** and **gas price**.
- The product of **gas price** and **gas limit** represents the maximum amount of Wei that the sender is willing to pay for executing a transaction.

$$\begin{array}{|c|} \hline \text{Gas Limit} \\ \hline \mathbf{50,000} \\ \hline \end{array} \times \begin{array}{|c|} \hline \text{Gas Price} \\ \hline \mathbf{20 \text{ gwei}} \\ \hline \end{array} = \begin{array}{|c|} \hline \text{Max transaction fee} \\ \hline \mathbf{0.001 \text{ Ether}} \\ \hline \end{array}$$

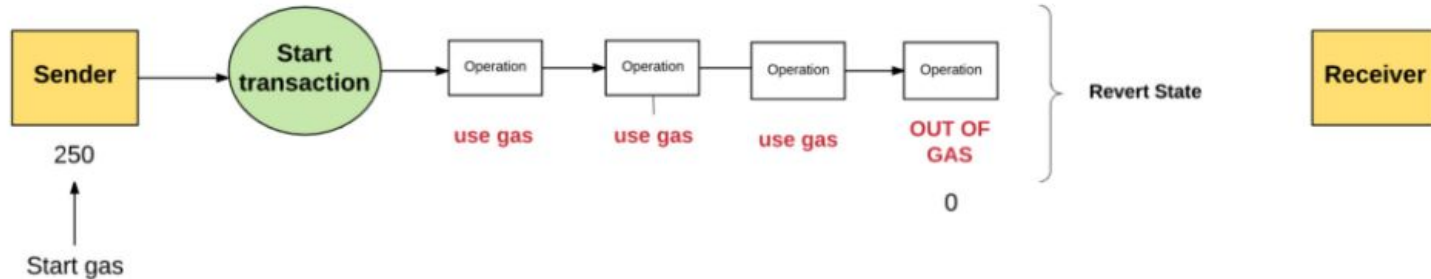
Gas Cost

- 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)

In the case that the sender does not provide the necessary gas to execute the transaction, the transaction runs “out of gas” and is considered invalid.



In this case, the transaction processing aborts and any state changes that occurred are reversed, such that we end up back at the state of Ethereum prior



All the money spent on gas by the sender is sent to the “beneficiary” address, which is typically the miner’s address. Since miners are expending the effort to run computations and validate transactions, miners receive the gas fee as a reward.

PoW vs. PoS

Ethereum in the process of moving to Proof of Stake

- This approach does not require large expenditures on computing and energy
- Miners are now “validators” and post a deposit in an escrow account
- The more escrow you post, the higher the probability you will be chosen to nominate the next block
- If you nominate a block with invalid transactions, you lose your escrow

PoW vs. PoS

Ethereum in the process of moving to Proof of Stake

- One issue with this approach is that those that have the most ethereum will be able to get even more
- This leads to centralization eventually
- On the other hand, it reduces the chance of a 51% attack and allows for near instant transaction approvals
- The protocol is called Casper and this will be a hard fork

<https://blockonomi.com/ethereum-casper/>

Other approaches to consensus

There are many other types of consensus

- (PoW) Proof of Work (Bitcoin, Ethereum, ...)
- (PoS) Proof of Stake (Ethereum in future)
- (PoI) Proof of Importance (used in NEM)
- (PBFT) Practical Byzantine Fault Tolerance (Hyperledger Fabric)
- (FBFT) Federated Byzantine Fault Tolerance (Ripple, Stellar)
- (DPoS) Delegated Proof of Stake
- (PoET) Proof of Elapsed Time (Hyperledger Sawtooth)

<https://medium.com/@chrshmmmr/consensus-in-blockchain-systems-in-short-691fc7d1fefe>

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***

Ethereum Smart Contracts

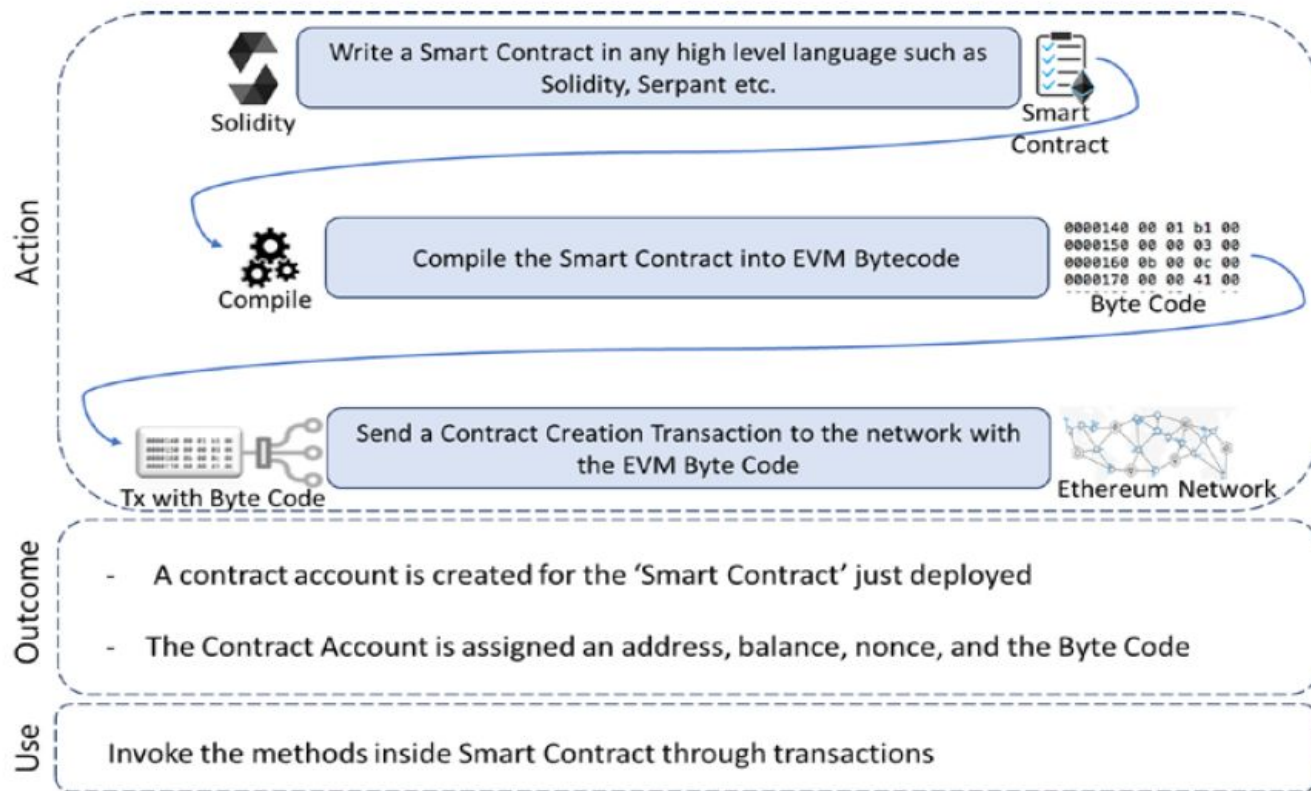


Figure 4-11. *Smart contract deployment and usage*

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

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.

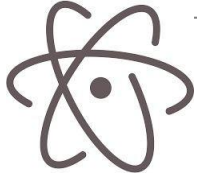
- [Solidity Documentation](#) - Solidity is the flagship Ethereum high level language that is used to write contracts.
- [Solidity online realtime compiler](#)

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](#)

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.

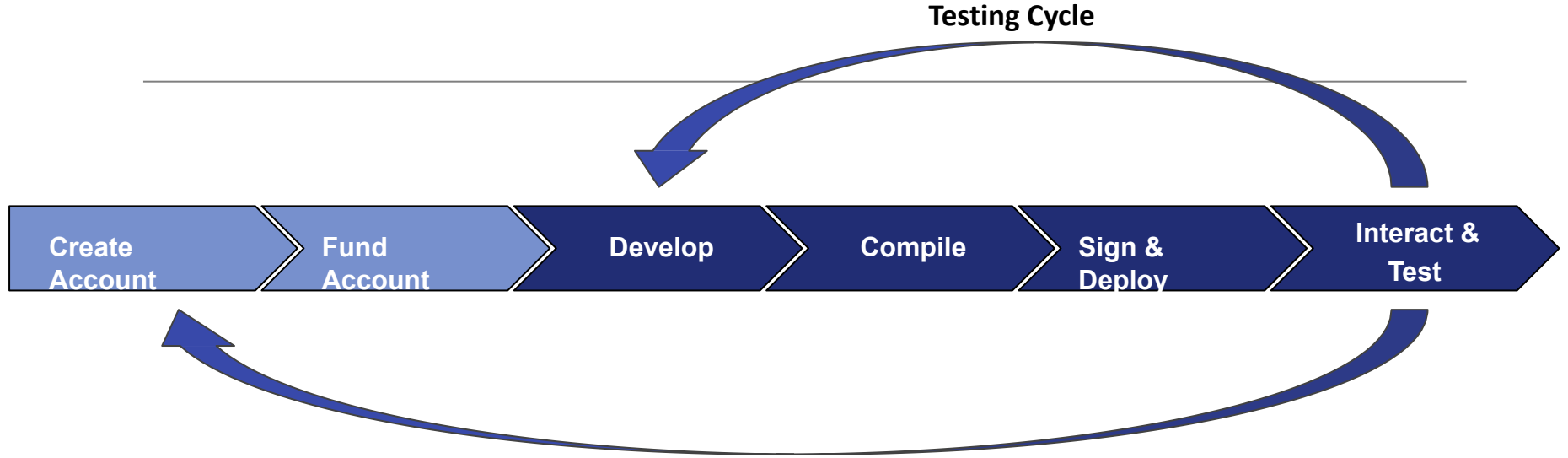
Smart Contract Programming: Solidity

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

Smart Contract Programming: Solidity

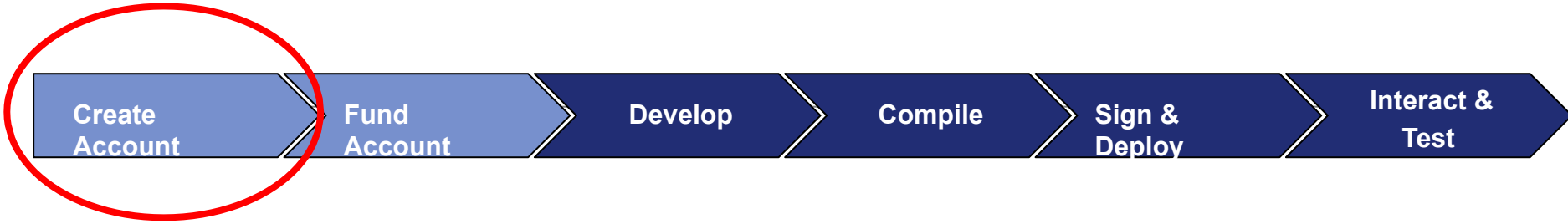
```
var logIncrement =  
    OtherExample.LogIncrement({sender: userAddress,  
uint value});  
  
logIncrement.watch(function(err, result) {  
    // do something with result  
})
```

Development Workflow



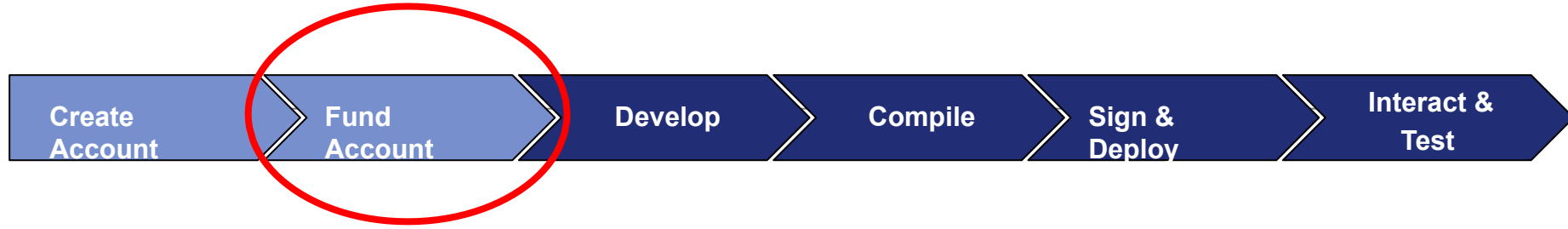
- **Onboard Additional Users**
- **Create New Accounts**
- **Develop New Applications**

Development Workflow: Create Account



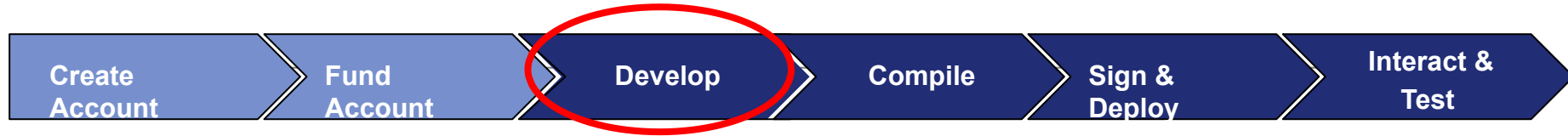
- Programmatically: Go, Python, C++, JavaScript, Haskell
- Tools
 - MyEtherWallet.com
 - MetaMask, Mist
 - TestRPC
 - Many other websites

Development Workflow: Fund Account



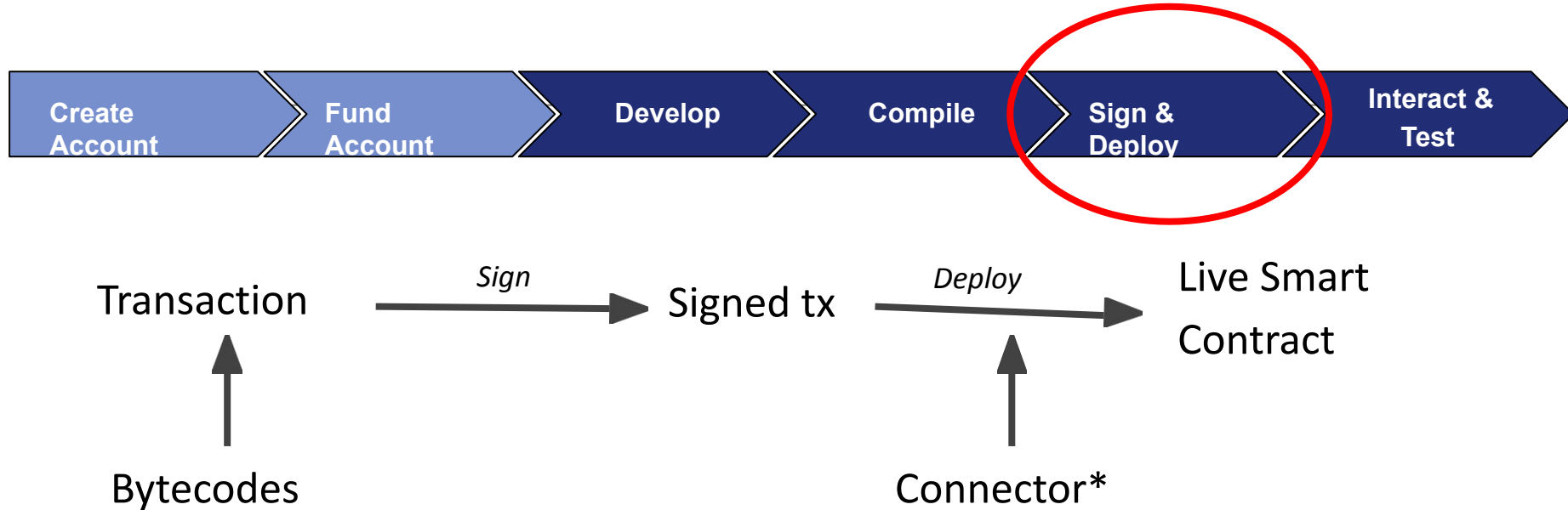
- From friends
- Faucet
- Exchanges (for public blockchain)

Development Workflow: Develop



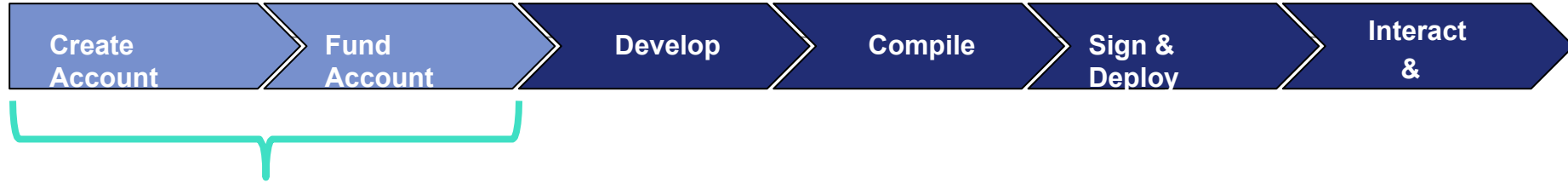
- **Ethereum Application Components:**
 - **Base application:** can be developed in **any** language
 - **Smart contract:** developed in Solidity or one of the other contract compatible languages
 - **Connector library:** facilitates communication between base application and smart contracts (Metamask)

Development Workflow: Sign and Deploy



*Library that facilitates communication and connection with Blockchain;
Connects your code to a running node.

Development Workflow: TestRPC



TestRPC/TestChain

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

Development Workflow: TestRPC

- EthereumJS TestRPC: <https://github.com/ethereumjs/testrpc> is suited for development and testing
- It's a complete blockchain-in-memory that runs only on your development machine
- It processes transactions instantly instead of waiting for the default block time – so you can test that your code works quickly – and it tells you immediately when your smart contracts run into errors
- It also makes a great client for automated testing
- Truffle knows how to use its special features to speed up test runtime by almost 90%.