Lecture 18

Ethereum

A Next-Generation Smart Contract and Decentralized Application Platform

Bitcoin: Transaction Based Cryptocurrency

- Each block is a list of transactions.
- Each transaction is a software instruction to be executed by nodes.

Bitcoin: Transaction Based Cryptocurrency

- Each block is a list of transactions.
- Each transaction is a software instruction to be executed by nodes.

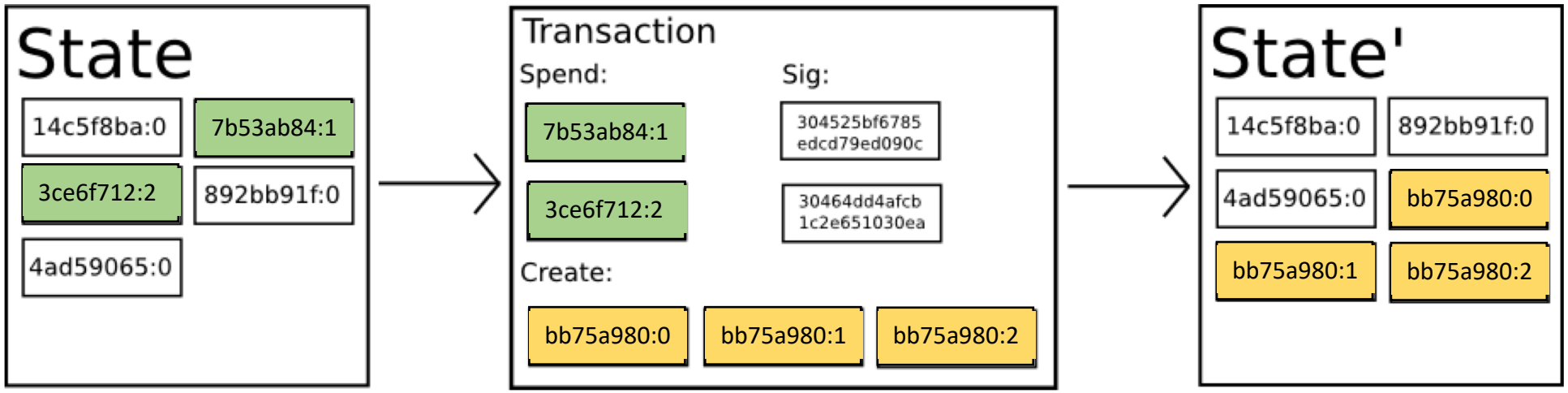
- Each coin has an **owner** and an **amount**.
- An owner spends a coin by referencing it in a transaction.

Bitcoin: State Transition System

- State: List of coins available for use
- Block: Set of instructions on how to edit the state by deleting & adding coins

Bitcoin: State Transition System

- State: List of coins available for use
- Block: Set of instructions on how to edit the state by “deleting” & “adding” coins



Ethereum: Motivation I

- I, **User X**, own this **unspent** coin
- Please **delete** it
- **Output** a new coin for **User Y**.

Can be implemented easily in Bitcoin.

Ethereum: Motivation I

- I, **User X**, own this **unspent** coin
- Please **delete** it
- and **Output** a new coin for **User Y**.

Can be implemented easily in Bitcoin.

- I, **User X**, own this **unspent** coin
- If presidential candidate **Z** wins the election:
  - Please **delete** it
  - and **output** a new coin for **User Y** at time **10/08/2019**

Ethereum: Motivation I

- I, **User X**, own this **unspent** coin
- Please **delete** it
- and **Output** a new coin for **User Y**.

Can be implemented easily in Bitcoin.

- I, **User X**, own this **unspent** coin
- If presidential candidate **Z** wins the election:
  - Please **delete** it
  - and **output** a new coin for **User Y** at time **10/08/2019**

Bitcoin's language makes it difficult to describe complex transactions.

Ethereum: Motivation I

"I, **User X**, own this **unspent** coin [proof: **signed** coin reference showing coin is part of state/list] so please **delete** it and **output** a new coin (into the state/list) owned by and signed for **User Y**".

Can be implemented easily in Bitcoin.

Since Bitcoin's language is not Turing Complete, transaction size grows with complexity.

"I, **User X**, own this **unspent** coin [proof: **signed** coin reference] so please **delete** it and **output** a new coin owned by and signed for **User Y** at time **10/08/2019** if presidential candidate **Z** wins the election".

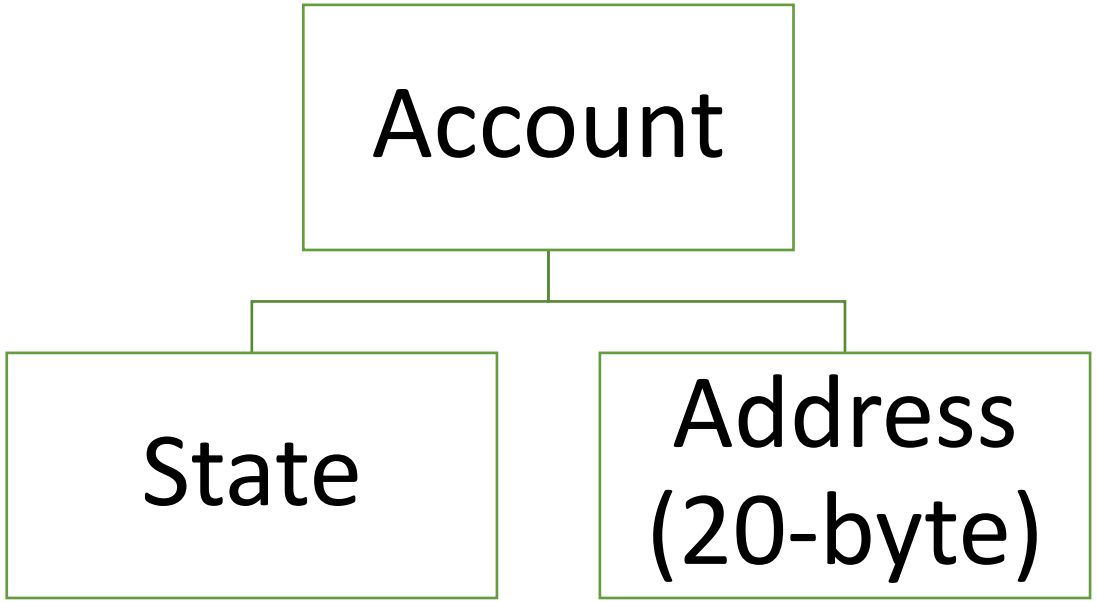
Bitcoin's language makes it difficult to describe complex transactions.

Ethereum: Motivation II

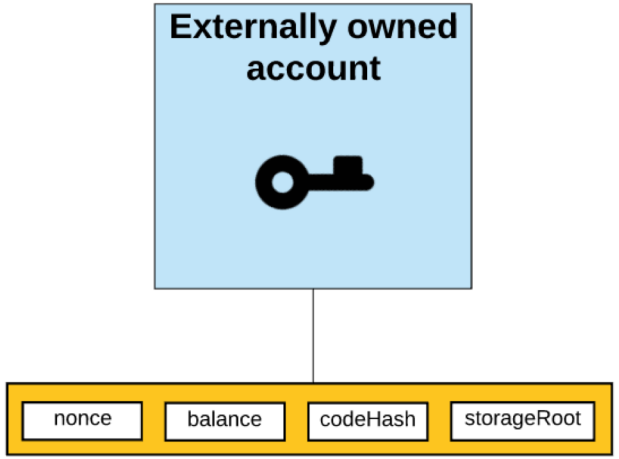
- Bitcoin blocks are capped at 1 Mb
- Users will pay higher fees to incentivize miners to include their transactions sooner.
- Since a transaction can’t be bigger than a block, the higher the demand for block space, the more users will pay for space to issue larger (and more complex) transactions.

Ethereum: **Account Based** Cryptocurrency

- Global State: consists of many small objects (“accounts”)
- Accounts interact with one another through a message-passing framework.



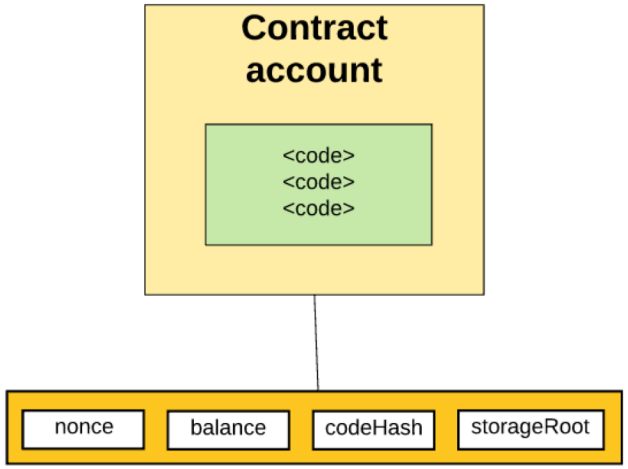
Types of Accounts



- Similar to user accounts in Bitcoin.
- Accounts owned by User X and User Y are of this type.

Presidential Bet

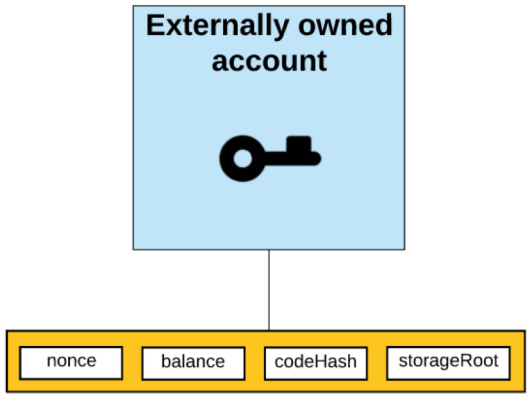
- **User X**, owns some money
- If presidential candidate **Z** wins the election:
  - Please **transfer M** amount to **User Y** at time **10/08/2019**



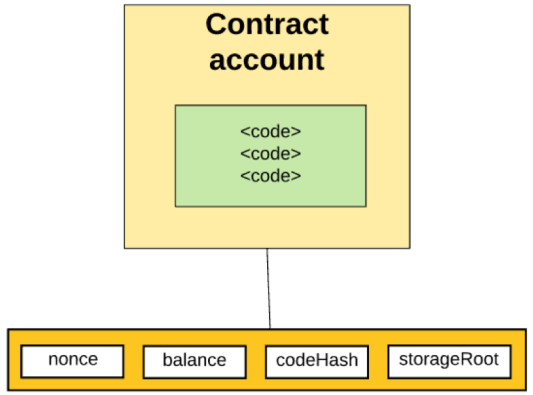
- Contract accounts have an associated code.
- This code specifies the contract details.

  If presidential candidate **Z** wins the election:
  - Please **transfer M** amount User **User X** to **User Y** at time **10/08/2019**

Types of Accounts



- Like Bitcoin, they have an associated private key to sign on transactions.
- Anybody can create these accounts



- Have an associated code.
- Do not have an associated private.
- Anybody can create these accounts.
- They can also be created by other accounts.

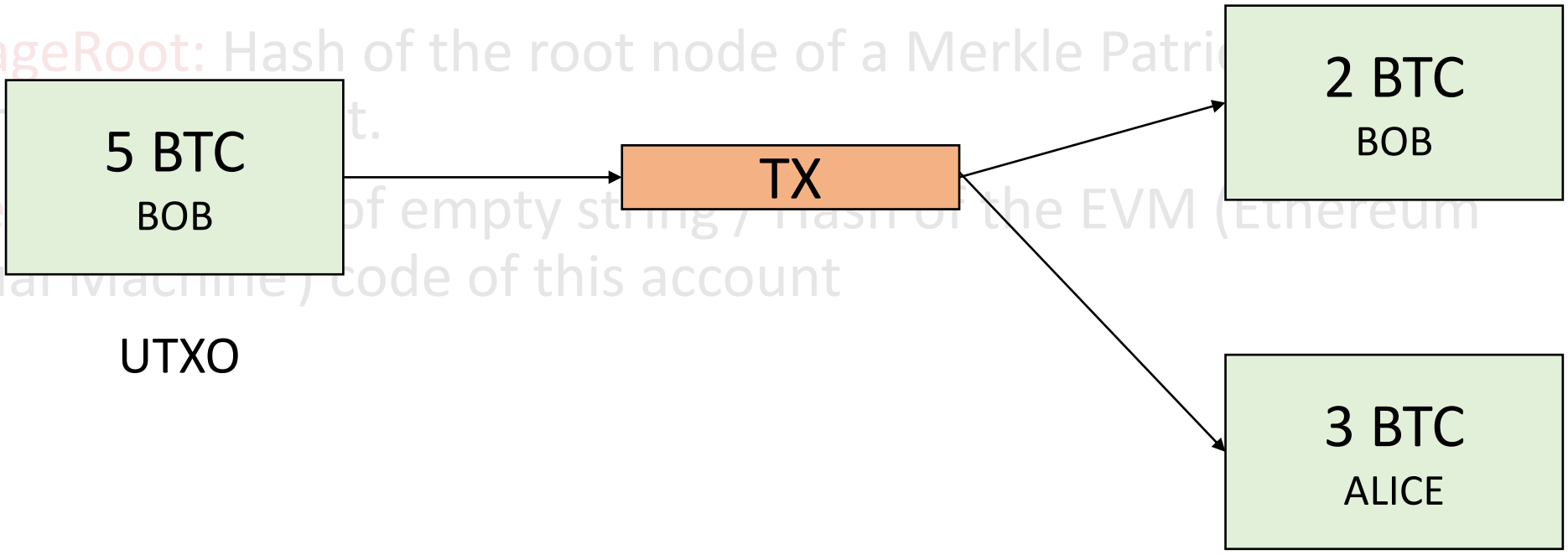
Account State

- nonce: # transactions sent/ # contracts created
- balance: # Wei owned (1 ether=10^{18}Wei)
- storageRoot: Hash of the root node of a Merkle Patricia tree. The tree is empty by default.
- codeHash: Hash of empty string / Hash of the EVM (Ethereum Virtual Machine) code of this account

Account State

Note: This is very different from the concept of accounts in Bitcoin, where each account has a list of unspent transaction outputs (UTXO). Spending these, creates new UTXOs.

- nonce: # transactions sent/ # contracts created
- balance: # Wei owned (1 ether=10^{18}Wei)
- storageRoot: Hash of the root node of a Merkle Patri ree is em t.
- code of empty string) Hash of the EVM (Ethereum Virtual Machine) code of this account



UTXO

New UTXOs

Communication between Accounts

- Accounts can communicate with each other.

- Why do they need to communicate?

Communication between Accounts

- Accounts can communicate with each other.

- Why do they need to communicate?
  - To transfer money to each other.
  - To transfer some information to each other.

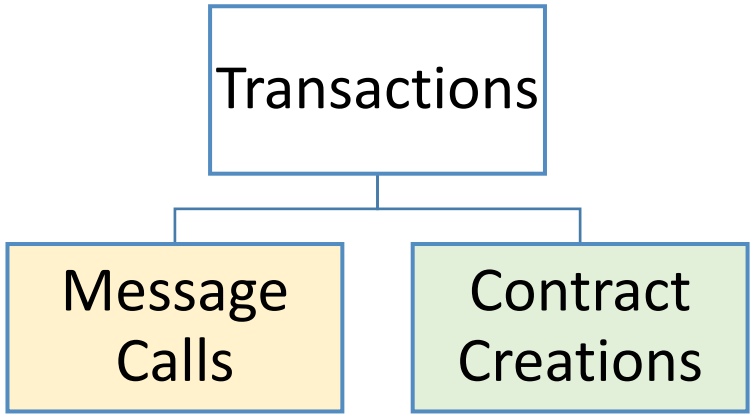
Communication between Accounts

- Accounts can communicate with each other.

- Why do they need to communicate?
  - To transfer money to each other.
  - To transfer some information to each other.

- How do they communicate?
  - They communicate with the help of transactions/internal transactions.

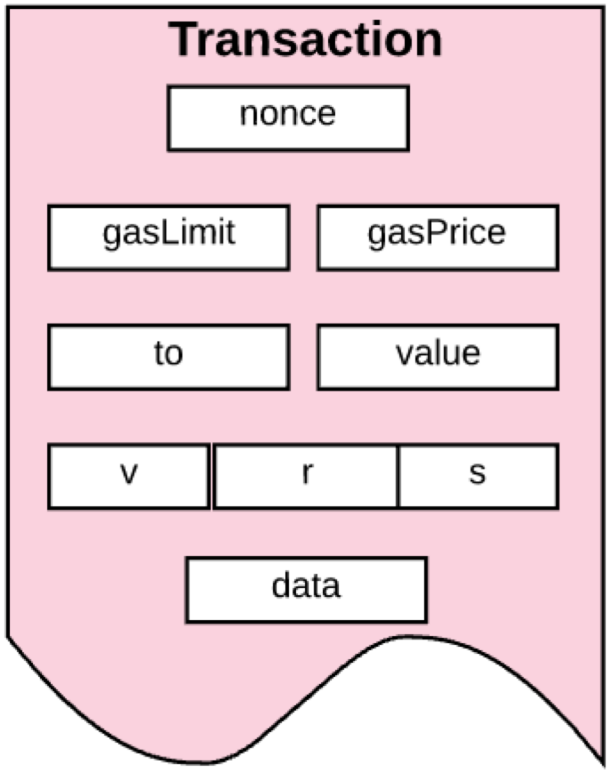
Transactions



Message Calls are transactions that are used for transferring money or information to other accounts (both Externally owned and contract accounts).

Contract creations are transactions that create new contract account.

Transactions

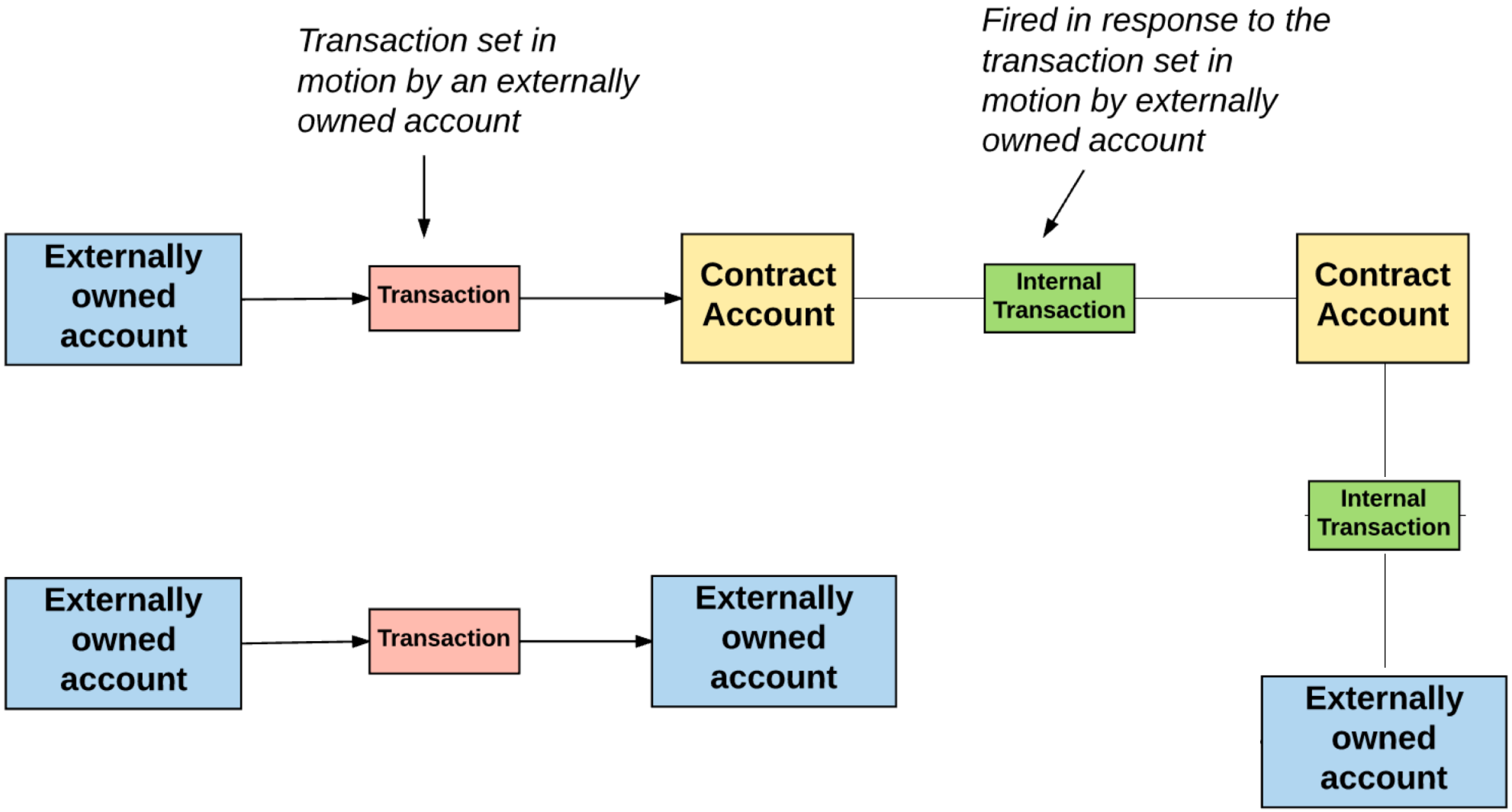


- nonce: A count of the number of transactions sent by the sender.
- gasPrice
- gasLimit
- to: Recepient's address
- value: Amount of Wei Transferred from sender to recipient.
- v,r,s: Used to generate the signature that identifies the sender of the transaction.
- init: EVM code used to initialize the new contract account.
- data: Optional field that only exists for message calls.

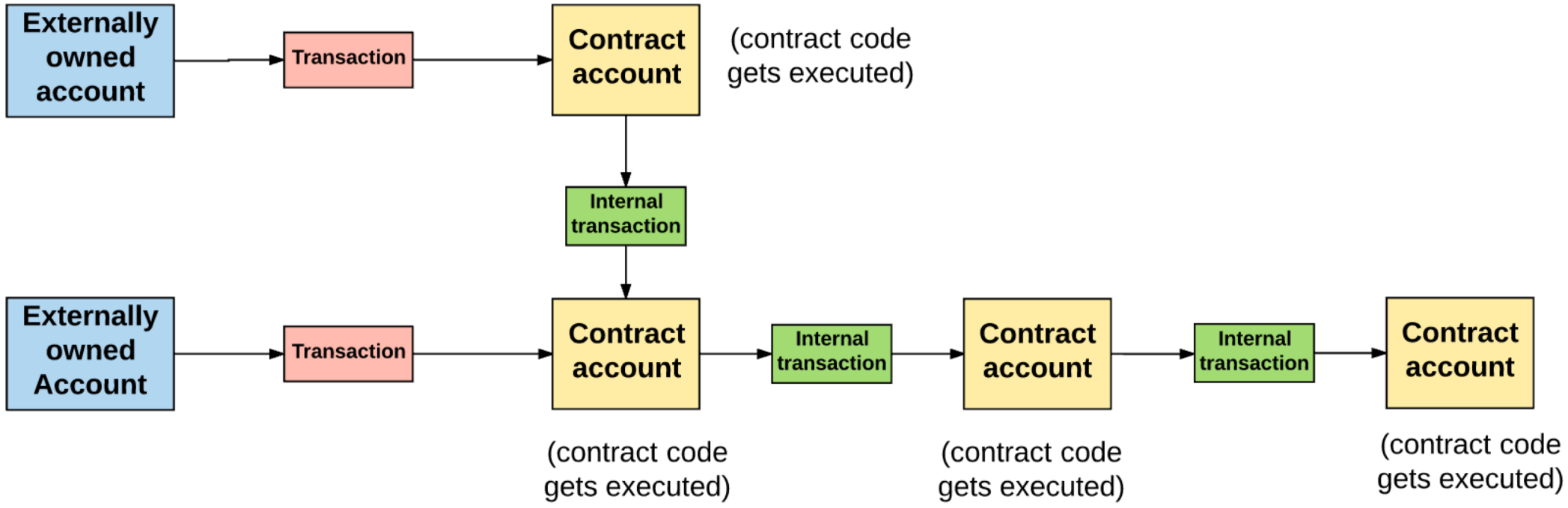
Transactions vs Internal Transactions

- Transactions can only be generated by externally owned accounts.
- Internal Transactions are like transactions but are only generated by contract accounts.
- Internal Transactions are not serialized.
- Internal Transactions can only be generated in response to transactions set in motion by externally owned accounts.

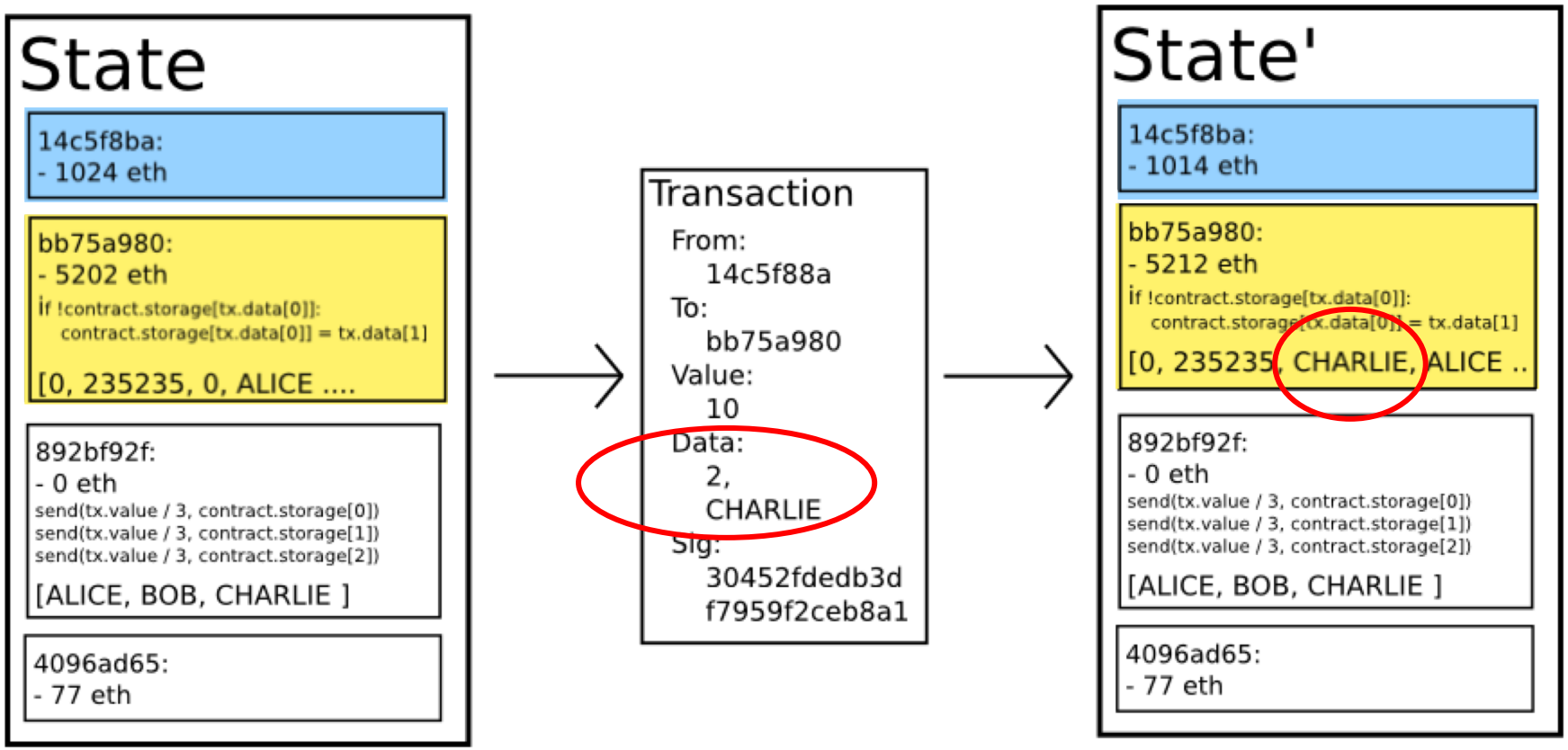
Transactions and Internal Transactions



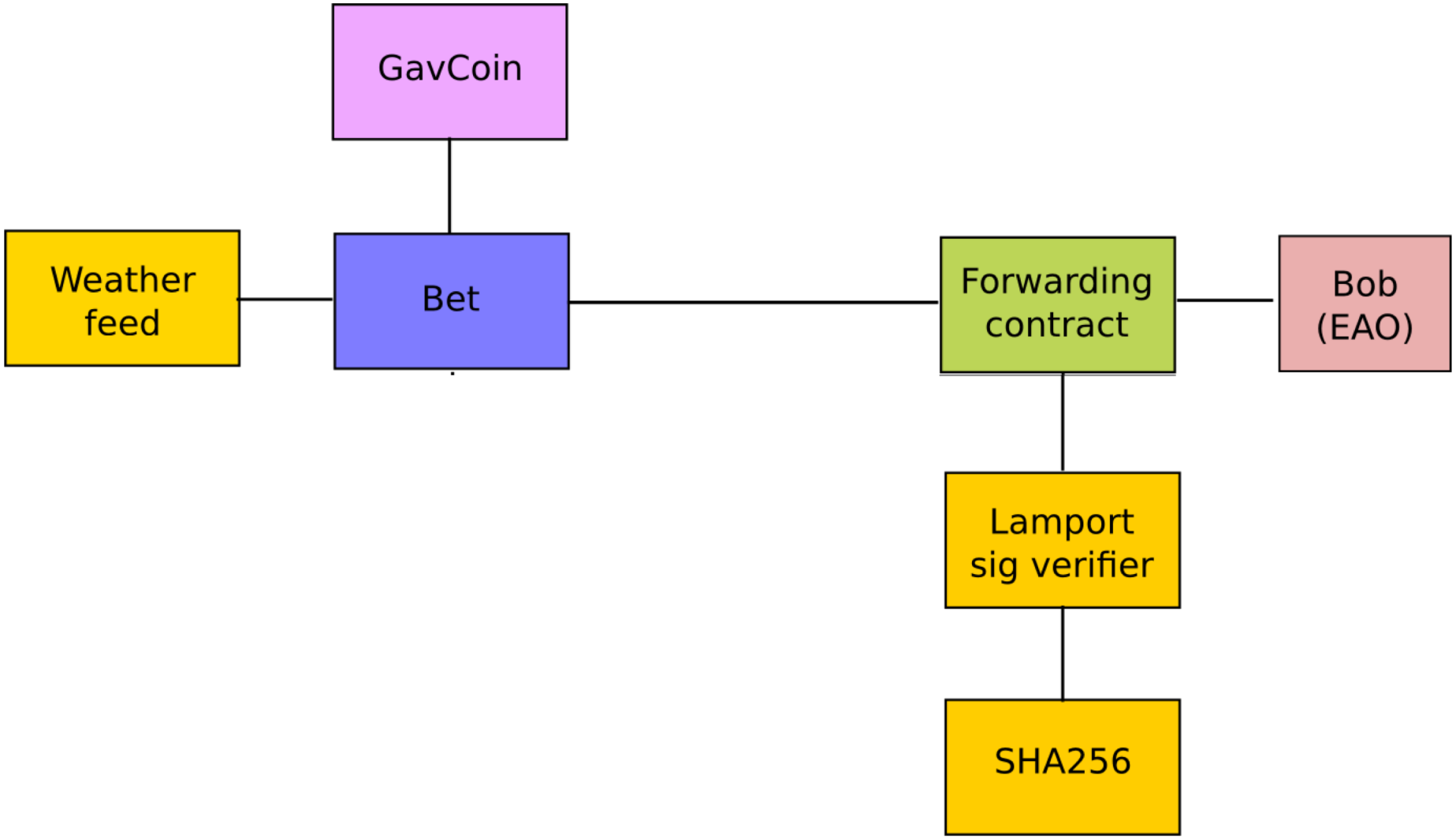
When a contract account receives a message, the associated code that exists on the recipient contract account is executed.



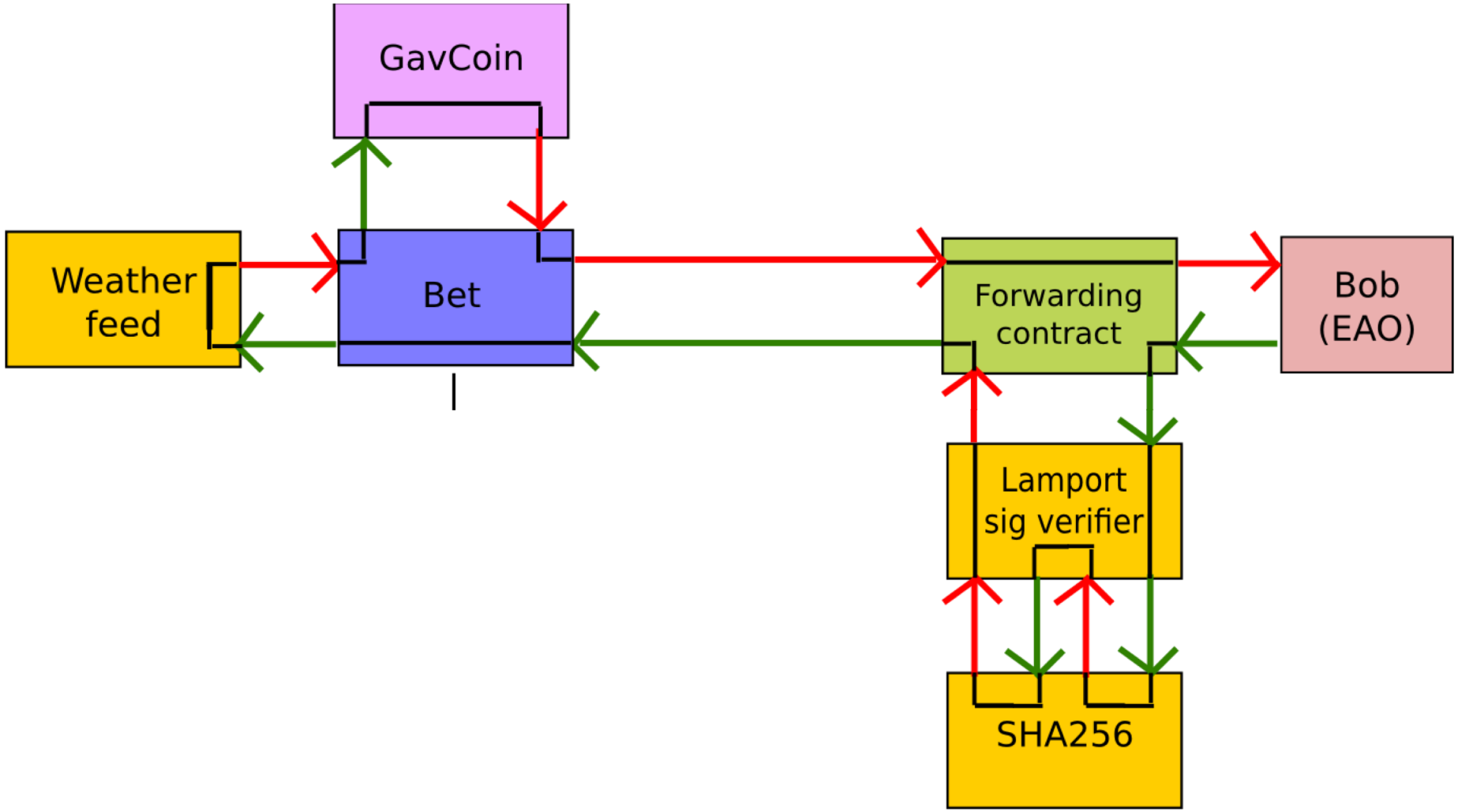
Transaction Execution



Betting GavCoins that the temperature in San Francisco will not exceed 35ºC



Bob bets 100 GavCoins that the temperature in San Francisco will not exceed 35ºC



Gas and Payment

- Every computation that occurs as a result of a transaction incurs a fee.
- This fee is paid in Gas.

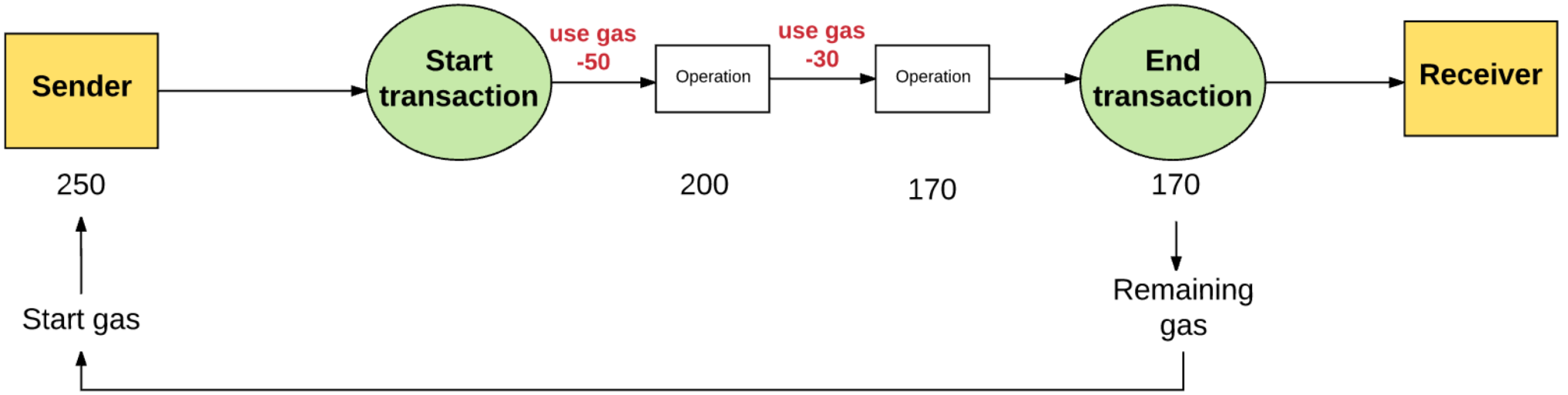
Gas and Payment

- Every computation that occurs as a result of a transaction incurs a fee.
- This fee is paid in Gas.
- Gas: Unit used to measure the fees required for a particular computation.
- Gas price: Amount of Ether you are willing to spend on every unit of gas.
- Gas is also used to pay for storage.

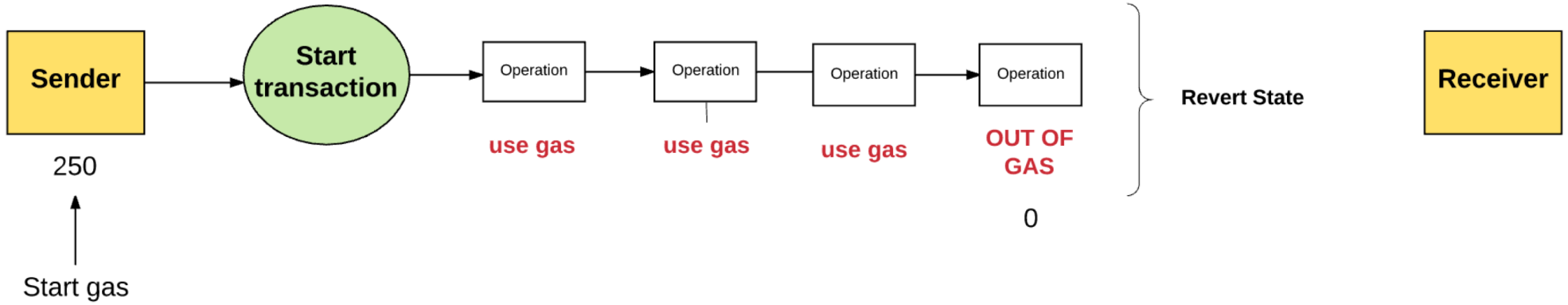
Gas and Payment

- Gas limit: Max no. of computational steps the transaction is allowed.
- Gas Price: Max fee the sender is willing to pay per computation step.

The sender is refunded for any unused gas at the end of the transaction.

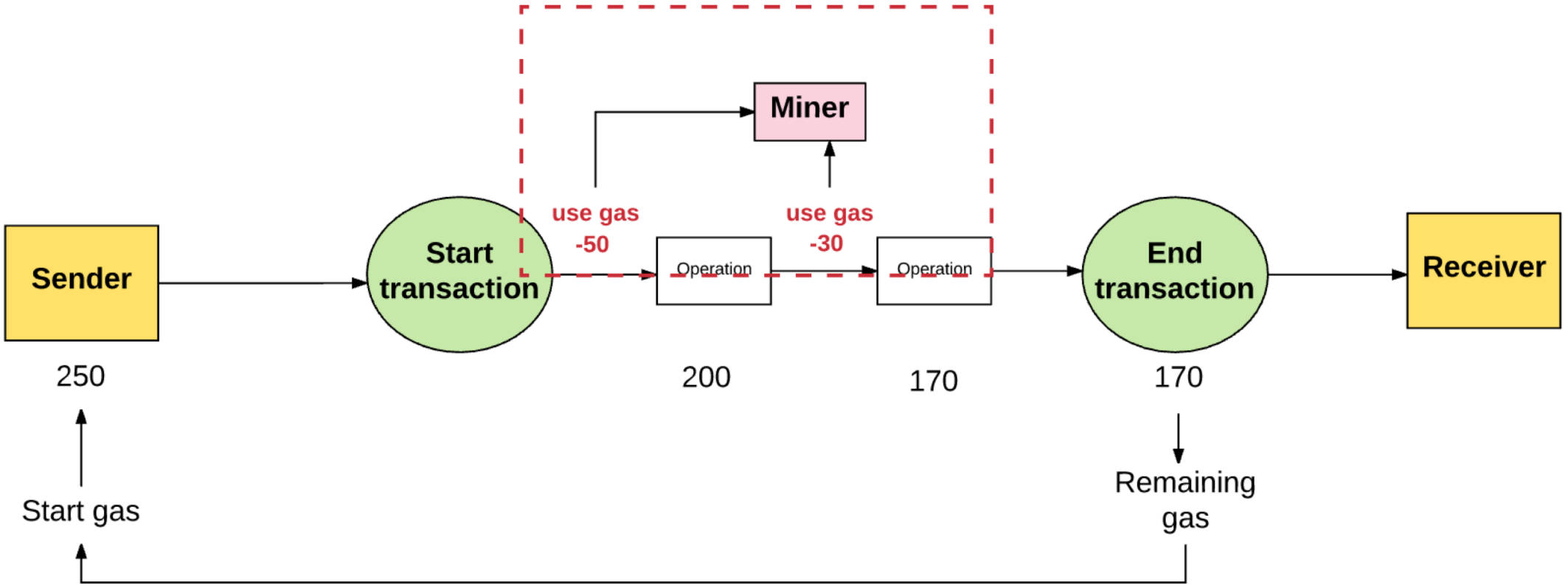


If sender does not provide the necessary gas to execute the transaction, the transaction runs “out of gas” and is considered invalid.



- The changes are reverted.
- None of the gas is refunded to the sender.

All the money spent on gas by the sender is sent to the miner's address.



What is the purpose of Fees?

- Imposing fees prevents users from overtaxing the network.
- Ethereum is a Turing complete language.
- This allows for loops and makes Ethereum susceptible to the halting problem.
- If there were no fees, an attacker could disrupt the network by executing an infinite loop within a transaction, without any repercussions.
- Thus, fees protect the network from deliberate attacks.

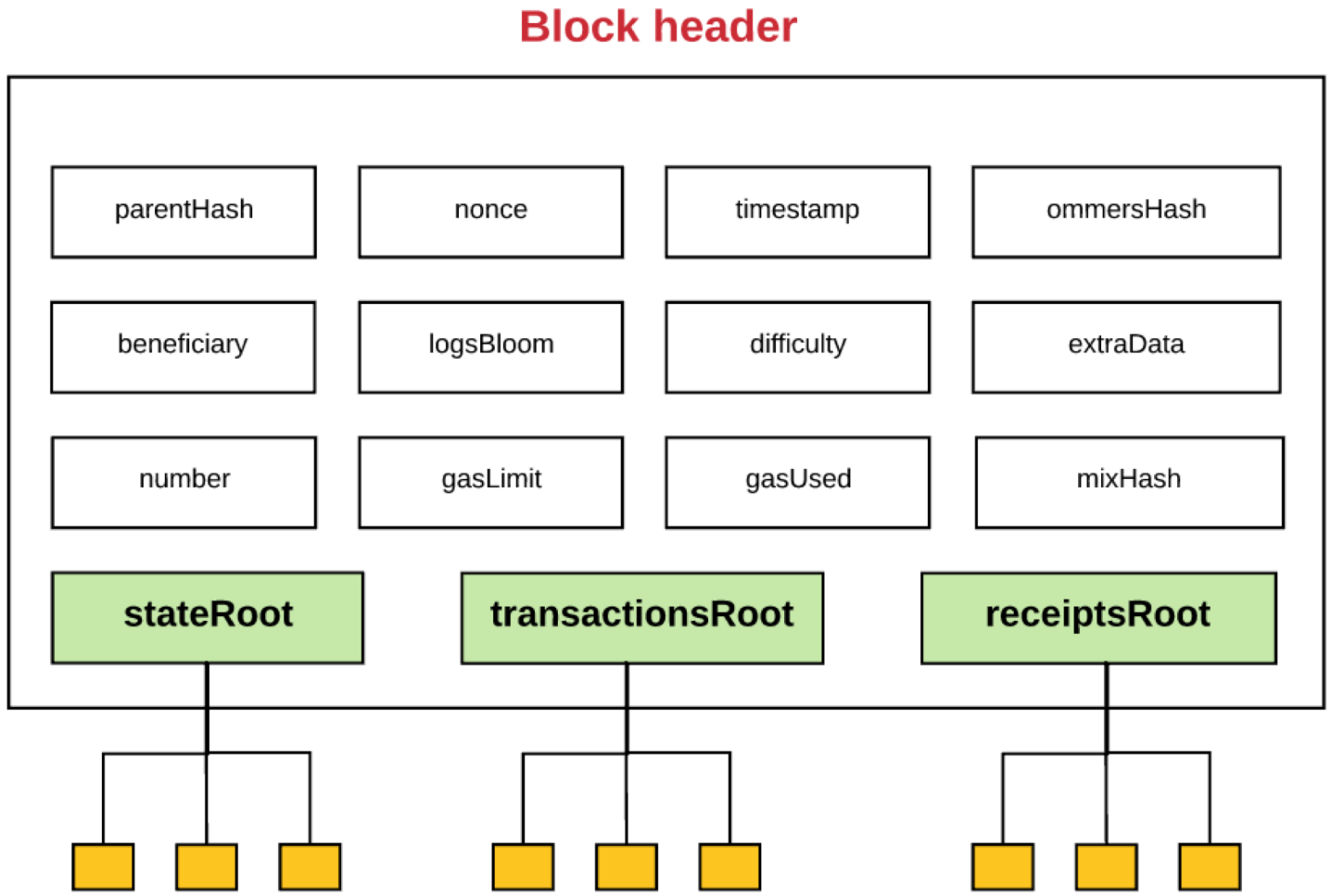
Bitcoin prevents this by putting a cap on the transaction size!!

Ethereum Blocks

In Ethereum, a block consists of:

- The block header
- Information about the set of transactions included in that block.
- A set of other block headers for the current block's ommers/uncles.

Block Header



Merkle Trees

Binary Merkle Trees:

- Good data structure for authenticating information.
- Any edits/insertions/deletions are costly.

Merkle Trees

Binary Merkle Trees:

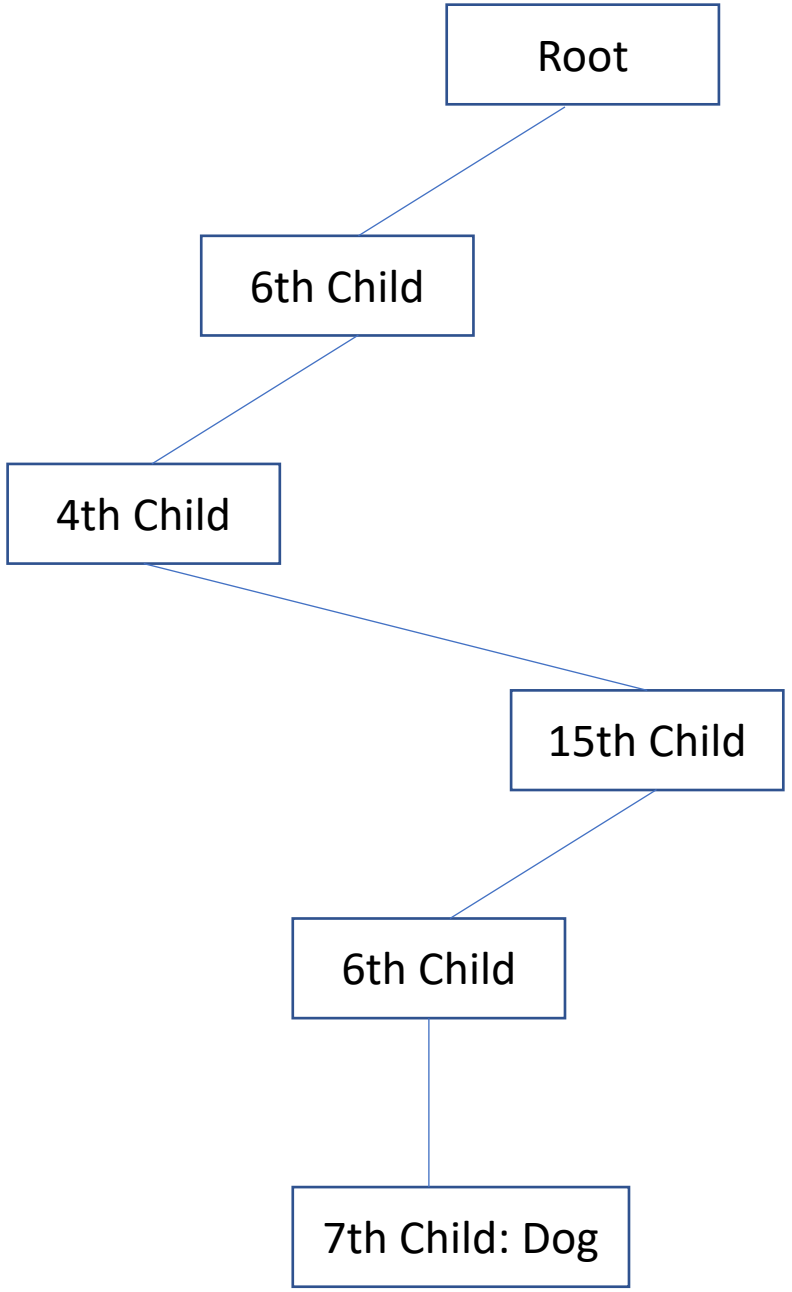
- Good data structure for authenticating information.
- Any edits/insertions/deletions are costly.

Merkle Patrica Trees:

- New tree root can be quickly calculated after an insert, update edit or delete operation in O(log n) time.
- Key-Value Pairs: Each value has a key associated with it.
- Key under which a value is stored is encoded into the path that you have to take down the tree.

Merkle Patricia Trees

- Each node has 16 children.
- eg: Hex(dog)= 6 4 6 15 6 7



Ommers/Uncles

- An ommer is a block whose parent is equal to the current block's parent's parent.
- Block times in Ethereum are around 15 sec. This is much lower than that in Bitcoin (10 min).
- This enables faster transaction. But there are more competing blocks, hence a higher number of orphaned blocks

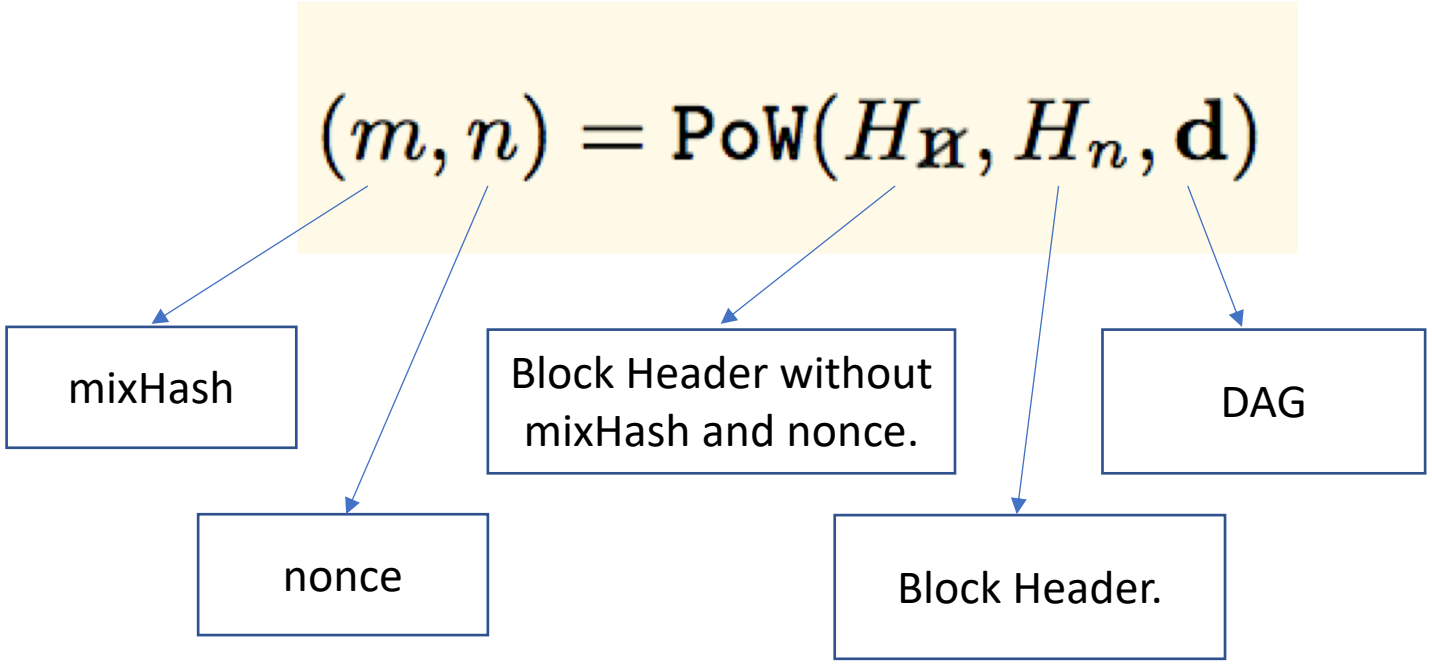
Ommers/Uncles

- An ommer is a block whose parent is equal to the current block's parent's parent.
- Block times in Ethereum are around 15 sec. This is much lower than that in Bitcoin (10 min).
- This enables faster transaction. But there are more competing blocks, hence a higher number of orphaned blocks
- The purpose of ommers is to help reward miners for including these orphaned blocks.
- The ommers that miners include must be within the sixth generation or smaller of the present block.

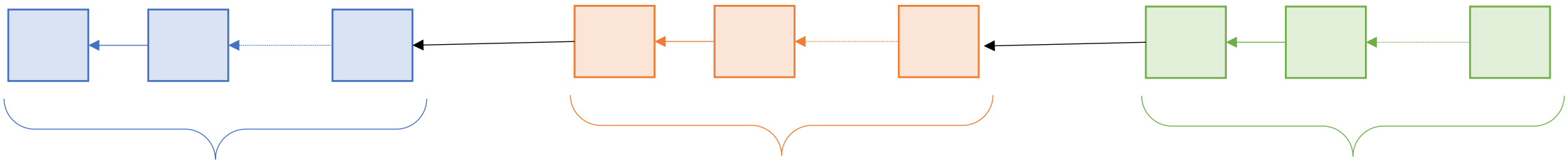
Mining: Proof of Work

- Ethereum's proof-of-work algorithm is called Ethash.
- Ethash is memory hard (or memory bound).
- The algorithm is formally defined as:

$$(m, n) = \mathtt{PoW}(H_{\not\mathtt{M}}, H_n, \mathbf{d})$$



Ethash

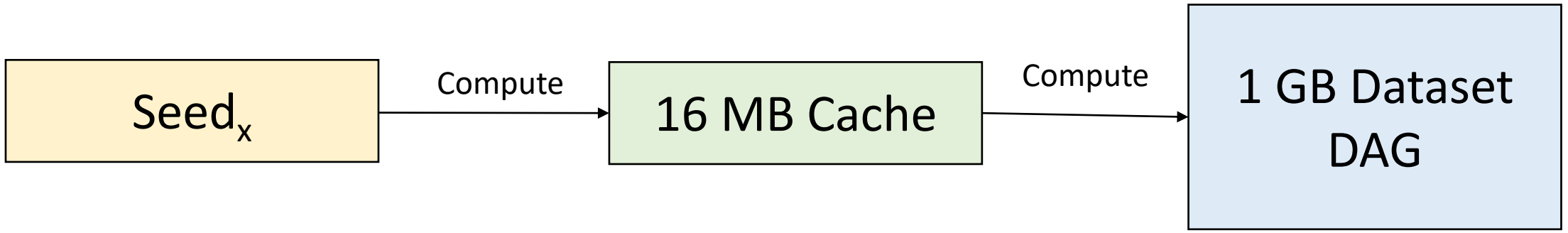


1st Epoch: 1st 30,000 blocks
$Seed_1$ = Hash(32 bytes of zeros)

2nd Epoch: Next 30,000 blocks
$Seed_2$ = Hash($Seed_1$)

3rd Epoch: Next 30,000 blocks
$Seed_3$ = Hash($Seed_2$)

Ethash



- Light nodes only need to store the cache for verification.
- They can efficiently verify a transaction without storing the entire blockchain dataset.

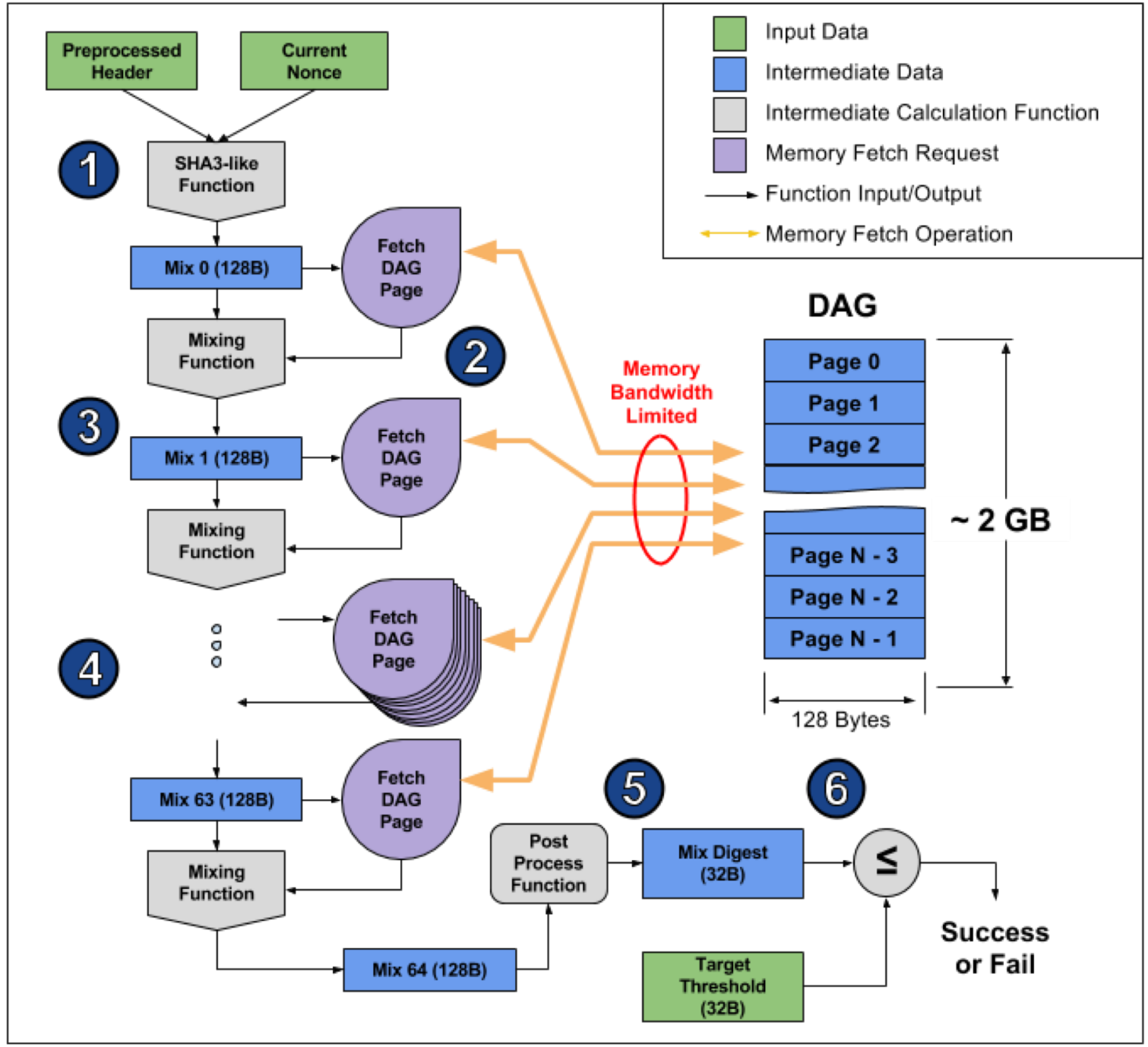
- Each item in the dataset depends on only a small number of items from the cache.
- The dataset DAG grows linearly with time.
- Miners need to store this entire dataset DAG.

Ethash Mining Algorithm

- Miners take random slices of DAG and put them through a mathematical function to hash them together into a **mixHash.**
- A miner will repeatedly generate a **mixHash** until the output is below the desired target **nonce**.
- When the output meets this requirement, this nonce is considered valid and the block can be added to the chain.

$$m = H_{\mathrm{m}} \quad \wedge \quad n \leqslant \frac{2^{256}}{H_{\mathrm{d}}}$$

Ethash Hashing Algorithm



Why is Ethash Memory Hard?

- Every mixing operation requires a 128 byte read from the DAG.
- Hashing a single nonce requires 64 mixes, resulting in (128 Bytes x 64) = 8 KB of memory read.
- The reads are random access, so putting a small chunk of the DAG in an L1 or L2 cache isn’t going to help much.
- Fetching the DAG pages from memory is much slower than the mixing computation
- The best way to speed up the ethash hashing algorithm is to speed up the 128 byte DAG page fetches from memory.
- Thus, we consider the ethash algorithm to be memory hard.

Mining Reward

- A *static block reward* of 3 ether for the winning block.
- The cost of gas expended within the block by the transactions included in the block.
- An extra reward for including ommers as part of the block.

Applications: Meta-Coins/ Token Systems

Metacoin: A protocol that lives on top of Bitcoin, using bitcoin transactions to store metacoin transactions but having a different state.

Limitations of Bitcoin:

- Metacoin cannot prevent invalid transactions from appearing on the blockchain.
- A secure meta-protocol would need to backward scan all the way to the beginning of the Bitcoin Blockchain to determine if a transaction was valid.
- Cannot obtain “light-weight” meta-protocols.

Applications: Meta-Coins/ Token Systems

- Token system/currency: Is fundamentally a database with one operation:

> Subtract X units from A and give X units to B, with the provision that
> (1) A had at least X units before the transaction
> (2) the transaction is approved by A.

- This logic can be easily implemented into a contract.

```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

Applications: Namecoin/ Identity Systems

- Namecoin: A decentralized name registration database, attempted to use a Bitcoin-like blockchain.
- Use cases:
  - DNS System
  - Email authentication
- Can be easily implemented using Ethereum smart contract:

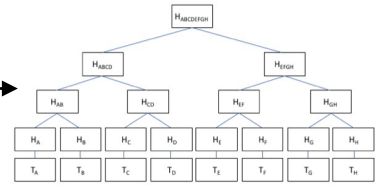
```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

Applications: Decentralized File Storage

Individual users can earn small quantities of money by renting out their own hard drives and unused space can be used to further drive down the costs of file storage.

Data

Contract



Merkle Tree with encrypted blocks
Broadcasted on the network

Contract

- Randomly chooses a index from this Merkle tree.
- Give X ether to the first node that provides a proof of storage for this block.

Other Applications

- Smart Multi-signature Escrow
- Cloud Computing
- Peer-to-peer gambling
- Prediction Markets
- …..

Thank You. ☺