Table of Contents

[What is a transaction in Blockchain? 2](#_Toc517456549)

[Transactions in Bitcoin 2](#_Toc517456550)

[How transaction works in Bitcoin? 2](#_Toc517456551)

[Bitcoin State Transition System 3](#_Toc517456552)

[Transactions in Ethereum 4](#_Toc517456553)

[How transaction works in Ethereum? 4](#_Toc517456554)

[Ethereum State Transition Function 6](#_Toc517456555)

[Summary 7](#_Toc517456556)

# What is a transaction in Blockchain?

A transaction is a transfer of value that is broadcast to the [network](https://en.bitcoin.it/wiki/Network) and collected into [blocks](https://en.bitcoin.it/wiki/Block). A transaction typically references previous transaction outputs as new transaction inputs and dedicates all input values to new outputs. Transactions are not encrypted, so it is possible to browse and view every transaction ever collected into a block. Once transactions are buried under enough [confirmations](https://en.bitcoin.it/wiki/Confirmation) they can be considered [irreversible](https://en.bitcoin.it/wiki/Irreversible_Transactions).

# Transactions in Bitcoin

A Bitcoin transaction is a signed piece of data that is broadcasted to the network and if valid, it ends up on the blockchain as a block. A transaction is used to transfer the ownership of Bitcoins from one account address to another account address.

These transactions are the only way possible to have an interaction with the blockchain. When we send some bitcoin to another address from our wallet, a transaction is created and is broadcasted onto to the entire network. Bitcoin nodes on the network will add the transaction to their block which they are mining if the transaction is valid.

**Components of a Bitcoin Transaction:**

* **Transaction ID** – hash
* **Ver** – Transaction (Tx) format version
* **Lock**-**time**- The block will be processed if the height of the chain is bigger than this field. Should be 0 if has to included right now.
* **Size**- Size of the transaction in bytes
* **Inputs**
* **Outputs**

**Transaction Details:**

* Any Bitcoin amount that we send is always sent to an address.
* Any Bitcoin amount we receive is locked to the receiving address.
* Any time we spend Bitcoin, the amount we spend will always come from funds previously received and currently present in our wallet.
* Addresses receive Bitcoin, but they do not send Bitcoin – Bitcoin is sent from a wallet.

## How transaction works in Bitcoin?

The amounts that go into our wallet are not jumbled like the coins in a physical wallet. The received amounts don’t mix but remain separate and distinct as the exact amounts received by the wallet.

So, let’s take an example of a new wallet that we created. Currently this wallet does not have any balance at all. Suppose two transactions of 3 BTC and 4 BTC have been done in favor of this address. The wallet will show the balance of the address as 7 BTC but when we look on the blockchain we do not find anywhere 7 BTC associated with the address rather we find 3 BTC and 4 BTC.

The received bitcoin amounts don’t mix but remain separated as the exact amounts sent to the wallet. The two amounts in the example above are called the outputs of their originating transactions.

Bitcoin wallets always keep outputs separate and distinct.

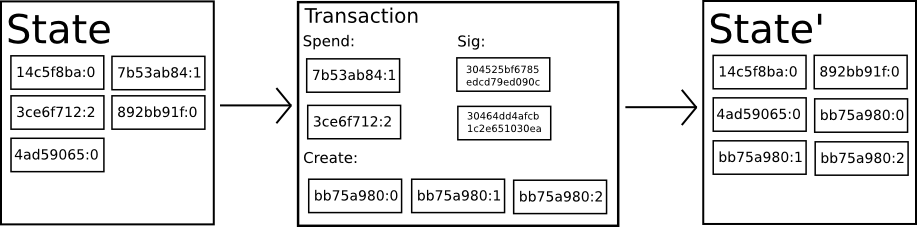
An output is an amount that was sent (via a standard transaction) to a Bitcoin address, along with a set of rules to unlock the output amount. In Bitcoin parlance an output is called an “unspent transaction output”, or UTXO.

A standard transaction output can be unlocked with the private key associated with the receiving address.

Suppose if we have to now send a 2 BTC from this address to another address. The wallet will unlock the 3 BTC output and use it as the input for our new 2 BTC transaction where the 2 BTC will be present as output for the new address. The difference of 1 BTC that is called the “change” and this will be sent back to the address. This change will be again resided as output in your wallet.

More than one output can be selected as the input for a transaction.

## Bitcoin State Transition System



From a technical standpoint, the ledger of a cryptocurrency such as Bitcoin can be thought of as a state transition system, where there is a "state" consisting of the ownership status of all existing bitcoins and a "state transition function" that takes a state and a transaction and outputs a new state which is the result.

In a standard banking system, for example, the state is a balance sheet, a transaction is a request to move $X from A to B, and the state transition function reduces the value in A's account by $X and increases the value in B's account by $X. If A's account has less than $X in the first place, the state transition function returns an error.

# Transactions in Ethereum

A transaction is a single cryptographically-signed instruction constructed by an actor externally to the scope of Ethereum.

There are two mainly types of transactions:

1. Those which result in **message calls**,
2. Those which result in the **creation of new accounts with associated code** (known informally as `**contract creation**').

Both types specify a number of common fields:

* **nonce**: a count of the number of transactions sent by the sender.
* **gasPrice**: the number of Wei that the sender is willing to pay per unit of gas required to execute the transaction.
* **gasLimit**: the maximum amount of gas that the sender is willing to pay for executing this transaction. This amount is set and paid upfront, before any computation is done.
* **to**: the address of the recipient. In a contract-creating transaction, the contract account address does not yet exist, and so an empty value is used.
* **value**: the amount of Wei to be transferred from the sender to the recipient. In a contract-creating transaction, this value serves as the starting balance within the newly created contract account.
* **v, r, s**: used to generate the signature that identifies the sender of the transaction.
* **init** (only exists for contract-creating transactions): An EVM code fragment that is used to initialize the new contract account. **init** is run only once, and then is discarded. When **init** is first run, it returns the body of the account code, which is the piece of code that is permanently associated with the contract account.
* **data** (optional field that only exists for message calls): the input data (i.e. parameters) of the message call. For example, if a smart contract serves as a domain registration service, a call to that contract might expect input fields such as the domain and IP address.

## How transaction works in Ethereum?

The execution of a transaction is the most complex part of the Ethereum protocol: it defines the state transition function. It is assumed that any transactions executed first pass the initial tests of intrinsic validity. These include:

1. The transaction is well-formed RLP (Recursive Length Prefix), with no additional trailing bytes;
2. The transaction signature is valid;
3. The transaction nonce is valid (equivalent to the sender account's current nonce);
4. The gas limit is no smaller than the intrinsic gas, used by the transaction; and
5. The sender account balance contains at least the cost, required in up-front payment.

The execution of a valid transaction begins with an irrevocable change made to the state: the **nonce** of the account of the sender is incremented by one and the balance is reduced by part of the up-front cost.

The computation, whether *contract creation or a message call*, results in an eventual state (which may legally be equivalent to the current state), the change to which is deterministic and never invalid: there can be no invalid transactions from this point.

After the message call or contract creation is processed, *the refund counter* has to be incremented for the accounts that were self-destructed throughout its invocation.

Then the state is finalized by determining the *amount to be refunded*, from the *remaining gas*, plus some allowance from the refund counter, to the sender at the original rate.

The *total refundable amount is the legitimately remaining gas*, added to the refund balance, and with the latter component being capped up to a maximum of half (rounded down) of the total amount used.

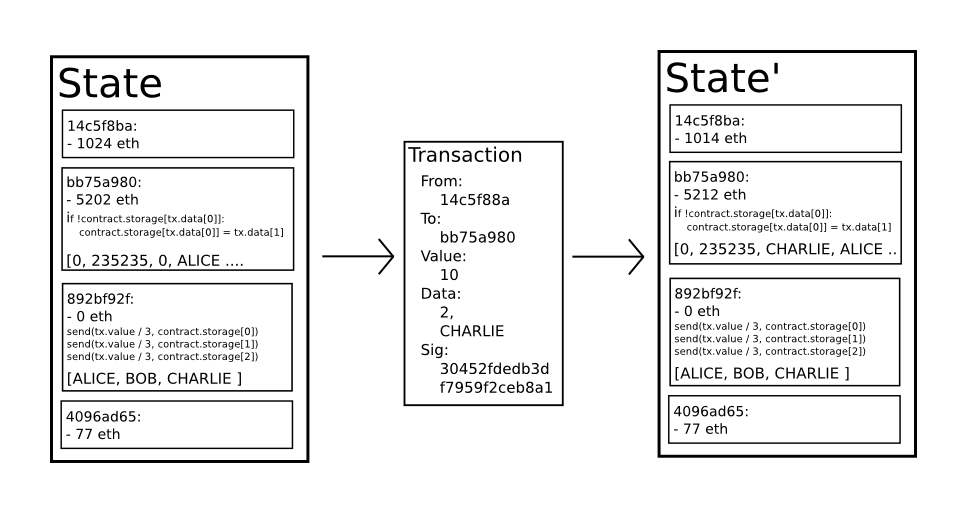
The Ether for the gas is given to the miner, whose address is specified as the beneficiary of the present block.

The final state is reached after deleting all accounts that either appear in the self-destruct list or are touched and empty.

And finally, we specify the total gas used in this transaction, the logs created by this transaction and the status code of this transaction.

These are used to help define the transaction receipt and are also used later for state and nonce validation.

## Ethereum State Transition Function



The Ethereum state transition function, APPLY(S,TX) -> S' can be defined as follows:

1. Check if the transaction is well-formed (i.e. has the right number of values), the signature is valid, and the nonce matches the nonce in the sender's account. If not, return an error.
2. Calculate the transaction fee as STARTGAS \* GASPRICE, and determine the sending address from the signature. Subtract the fee from the sender's account balance and increment the sender's nonce. If there is not enough balance to spend, return an error.
3. Initialize GAS = STARTGAS, and take off a certain quantity of gas per byte to pay for the bytes in the transaction.
4. Transfer the transaction value from the sender's account to the receiving account. If the receiving account does not yet exist, create it. If the receiving account is a contract, run the contract's code either to completion or until the execution runs out of gas.
5. If the value transfer failed because the sender did not have enough money, or the code execution ran out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.
6. Otherwise, refund the fees for all remaining gas to the sender, and send the fees paid for gas consumed to the miner.

# Summary

**Bitcoin uses unspent transaction outputs** to track who has how much bitcoin.

While it sounds more complex, the idea is fairly simple. Every time a bitcoin transaction is made, the network 'breaks' the total amount as if it was paper money, issuing back bitcoins in a way that makes the data behave similarly to physical coins or change.

To make future transactions, the bitcoin network must add up all your pieces of change, which are classed as either 'spent' or 'unspent'.

**Ethereum, on the other hand, uses accounts.**

Like bank account funds, ether tokens appear in a wallet, and can be ported (so to speak) to another account. Funds are always somewhere; yet don't have what you might call a continued relationship.

