Bookkeeping --> accounting --> balance --> state

Bookkeeping is the recording of financial transactions, and is part of the process of accounting in business.

Transactions include purchases, sales, receipts and payments by an individual person or an organization/corporation.

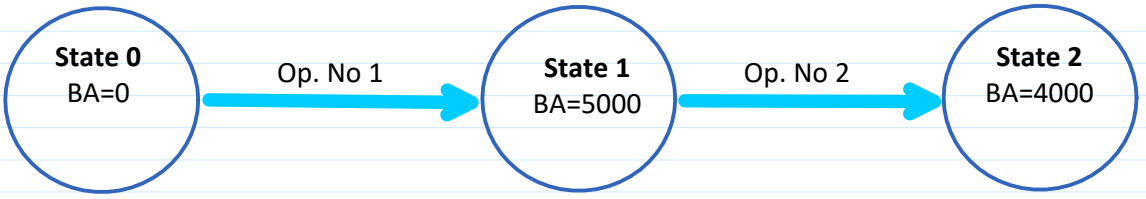
There are several standard methods of bookkeeping, including the single-entry and double-entry bookkeeping systems.

From <https://en.wikipedia.org/wiki/Bookkeeping>

https://www.dreamstime.com/stock-image-d-life-cycle-accounting-process-illustration-circular-flow-chart-image30625511

Business Resources:
Authorized capital
Credit
Fixed Assets
Costs
Incomes

Op.No	Input Incomes	Output Expenses	Balance Amount (BA)	**State** No
0	0	0	0	0
1	5000	0	5000	1
2	0	1000	4000	2

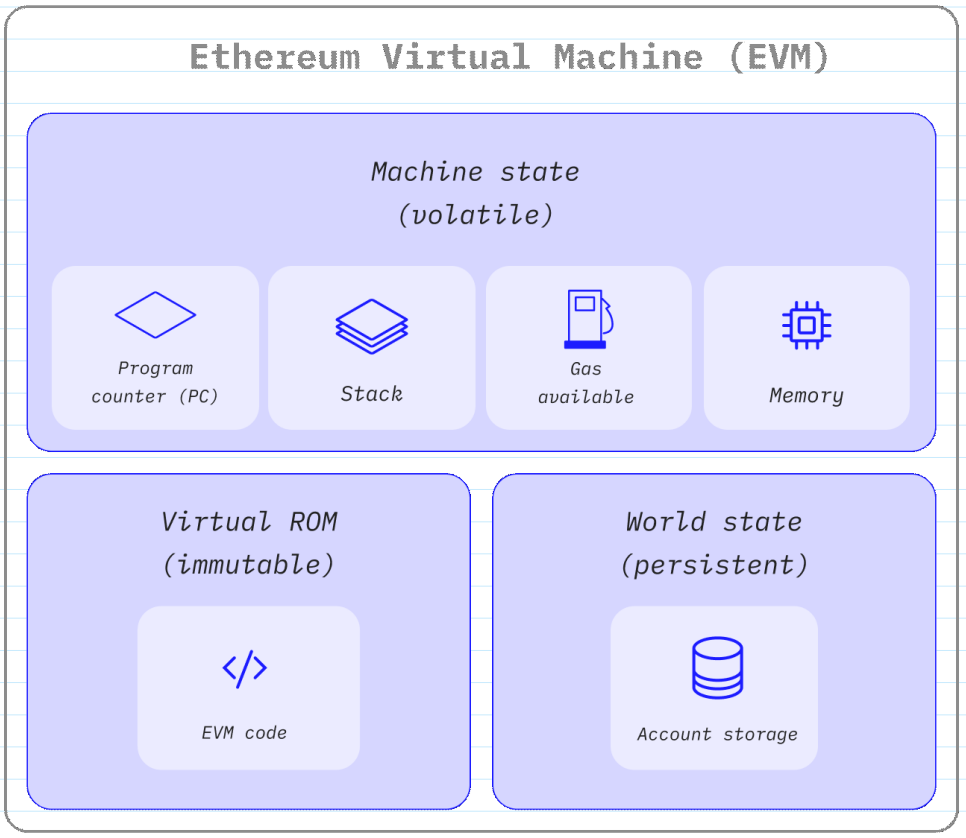


Ethereum is a blockchain with a computer embedded in it. It is the foundation for building Dapps and organizations in a decentralized, permissionless, censorship-resistant way.

In the Ethereum universe, there is a single, canonical computer called the Ethereum Virtual Machine - EVM whose state everyone on the Ethereum network agrees on. Everyone who participates in the Ethereum network (every Ethereum node) keeps a copy of the state of this computer. Additionally, any participant can broadcast a request for this computer to perform arbitrary computation. Whenever such a request is broadcast, other participants on the network verify, validate, and carry out ("execute") the computation. This execution causes a state change in the EVM, which is committed and propagated throughout the entire network.

Requests for computation are called transaction requests; the record of all transactions and the EVM's present state gets stored on the blockchain, which in turn is stored and agreed upon by all nodes.

Cryptographic mechanisms ensure that once transactions are verified as valid and added to the blockchain, they can't be tampered with later. The same mechanisms also ensure that all transactions are signed and executed with appropriate "permissions" (no one should be able to send digital assets from Alice's account, except for Alice herself).



Execution model
Now, we'll look at how the transaction actually executes within the VM.
The part of the protocol that actually handles processing the transactions is Ethereum's own in EVM.
The EVM is a Turing complete virtual machine, as defined earlier. The only limitation the EVM has that a typical Turing complete machine does not is that the EVM is intrinsically bound by gas. Thus, the total amount of computation that can be done is intrinsically limited by the amount of gas provided.

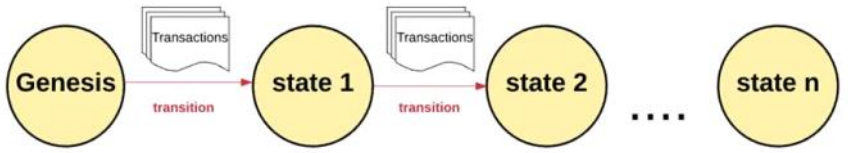
Ethereum Virtual Machine (EVM) | ethereum.org

Ethereum blockchain cryptocurrency is Ether - **ETH**
It is the native cryptocurrency of Ethereum. The purpose of ETH is to allow for a market for computation. Such a market provides an economic incentive for participants to verify and execute transaction requests and provide computational resources to the network.
Any participant who broadcasts a transaction request must also offer some amount of ETH to the network as a bounty. The network will burn part of the bounty and award the rest to whoever eventually does the work of verifying the transaction, executing it, committing it to the blockchain, and broadcasting it to the network. The amount of ETH paid corresponds to the resources required to do the computation. These bounties also prevent malicious participants from intentionally clogging the network by requesting the execution of infinite computation or other resource-intensive scripts, as these participants must pay for computation resources.

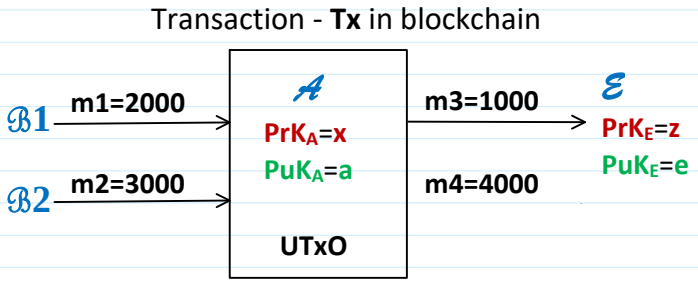
In addition ETH is also used to provide crypto-economic security to the network in three main ways:
1) it is used as a means to reward validators who propose blocks or call out dishonest behavior by other validators;
2) It is staked by validators, acting as collateral against dishonest behavior—if validators attempt to misbehave their ETH can be destroyed;
3) it is used to weigh 'votes' for newly proposed blocks, feeding into the fork-choice part of the consensus mechanism.

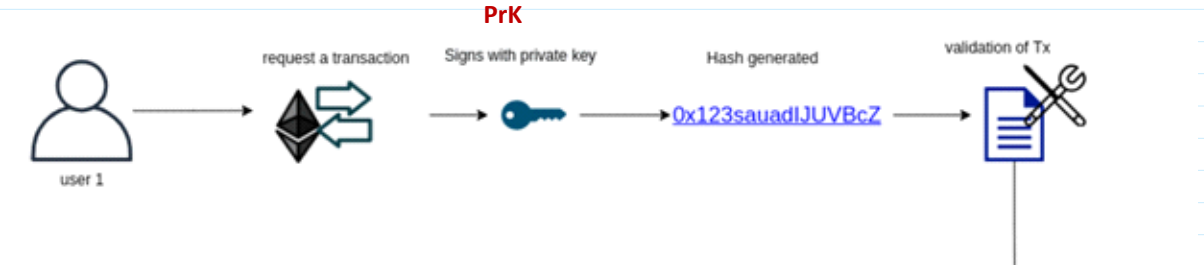
Unit	Alternative Name	Wei Value	Gwei Value	Ether Value
Wei	Wei	1 Wei	10^-9 Gwei	10^-18 ETH
Kwei	Babbage	10^3 Wei	10^-6 Gwei	10^-15 ETH
Mwei	Lovelace	10^6 Wei	10^-3 Gwei	10^-12 ETH
Gwei	Shannon	10^9 Wei	1 Gwei	10^-9 ETH
Twei	Szabo	10^12 Wei	10^3 Gwei	10^-6 ETH
Pwei	Finney	10^15 Wei	10^6 Gwei	10^-3 ETH
Ether	Ether	10^18 Wei	10^9 Gwei	1 ETH

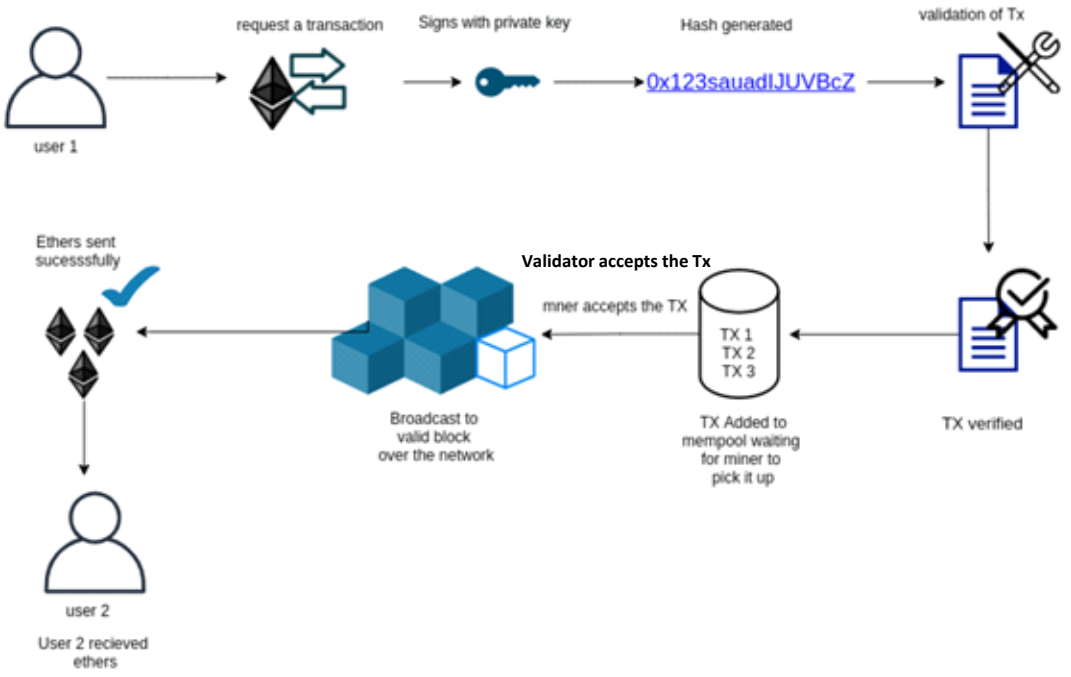
The **state** of Ethereum has millions of transactions. These transactions are grouped into "blocks." A block contains a series of transactions, and each block is chained together with its previous block.



To cause a transition from one state to the next, a transaction must be valid.
For a transaction to be considered valid, it must go through a validation process known (e.g. as mining).
Validation is when a validator expend their compute resources to create a block of valid transactions.







Ethereum system is comprised of:

- **accounts**
- **state**
- **gas and fees**
- **transactions**
- **blocks**
- **transaction execution**
- **validation: Proof of Stake - PoS**

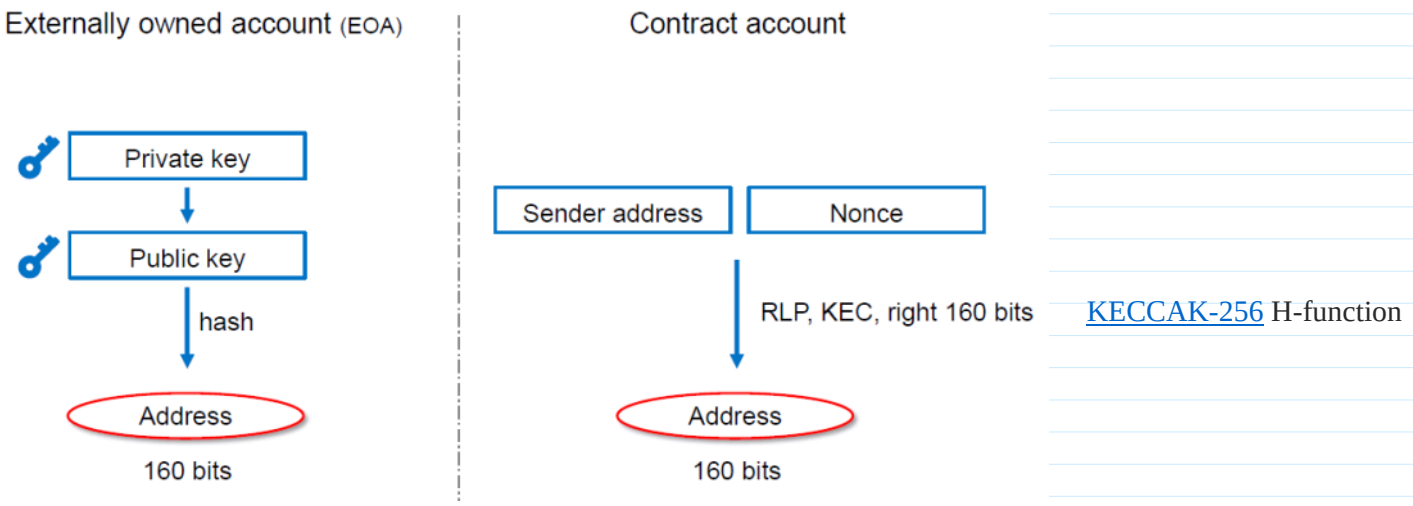
Accounts

The global “shared-state” of Ethereum is comprised of many small objects (“**accounts**”) that are able to interact with one another through a message-passing framework. Each **account** has a **state** associated with it and a 20-byte **address**. An address in Ethereum is a 160-bit identifier that is used to identify any account.

Addresses

Ethereum addresses are composed of the prefix "0x" (a common identifier for hexadecimal) concatenated with the rightmost 20 bytes of the Keccak-256 hash of the ECDSA public key.
The curve used in ECDSA is named as **secp256k1**.
In hexadecimal, two digits represent a byte, and so addresses contain 40 hexadecimal digits after the "0x".
E.g.,
0xb794f5ea0ba39494ce839613fffba74279579268
Contract addresses are in the same format, however, they are determined by sender and creation transaction nonce.
From <https://en.wikipedia.org/wiki/Ethereum>

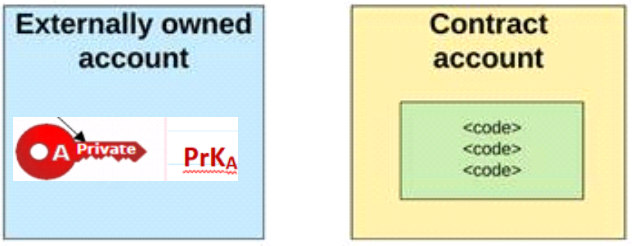
Address of account



There are two types of **accounts**:

- **Externally owned accounts**, which are **controlled by private keys** and have **no code associated with them.**
- **Contract accounts**, which are **controlled by their contract code** and **have code associated with them.**

Two types of account



Externally owned accounts vs. contract accounts

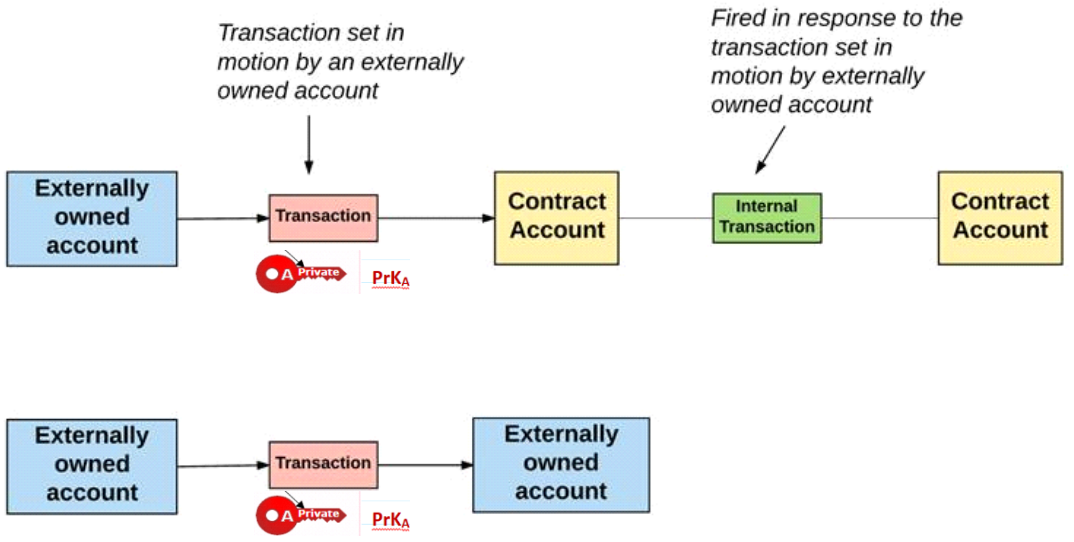
It's important to understand a fundamental difference between externally owned accounts and contract accounts.

An externally owned account can send messages to other externally owned accounts OR to other contract accounts by creating and signing a transaction using its private key.

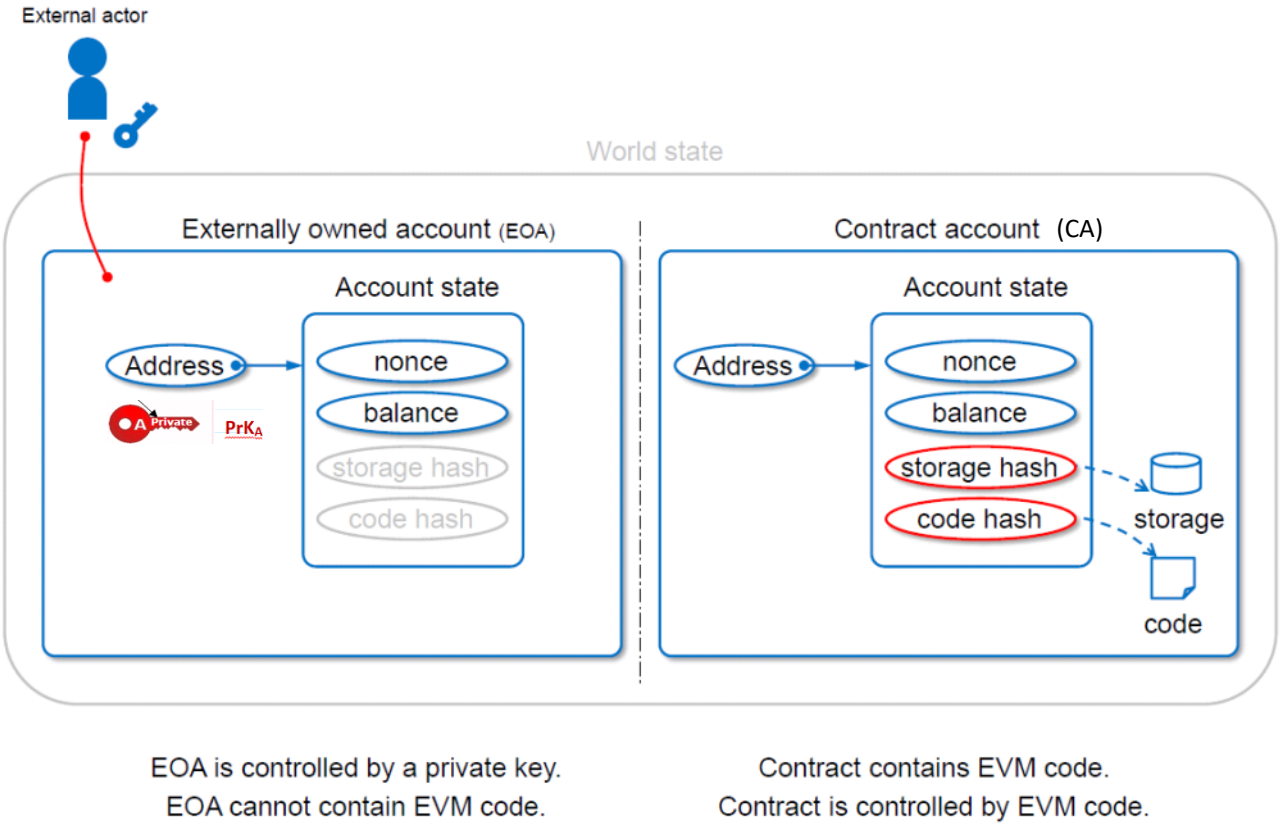
A message between two externally owned accounts is simply a value transfer.

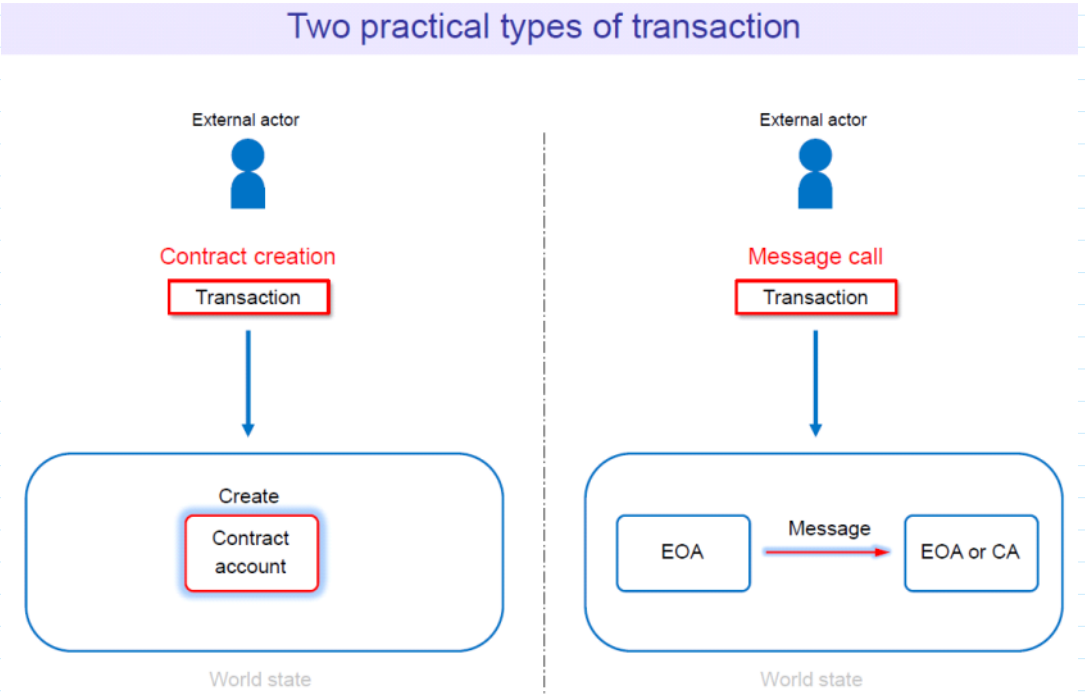
But a message from an externally owned account to a contract account ***activates*** the contract account's code, allowing it to perform various actions (e.g. transfer tokens, write to internal storage, mint new tokens, perform some calculation, **create new contracts**, etc.).

Unlike externally owned accounts, contract accounts can't initiate new transactions on their own. Instead, contract accounts can only fire transactions in response to other transactions they have received (from an externally owned account or from another contract account). We'll learn more about contract-to-contract calls in the.



Therefore, any action that occurs on the Ethereum blockchain is always set in motion by transactions fired from externally controlled accounts.



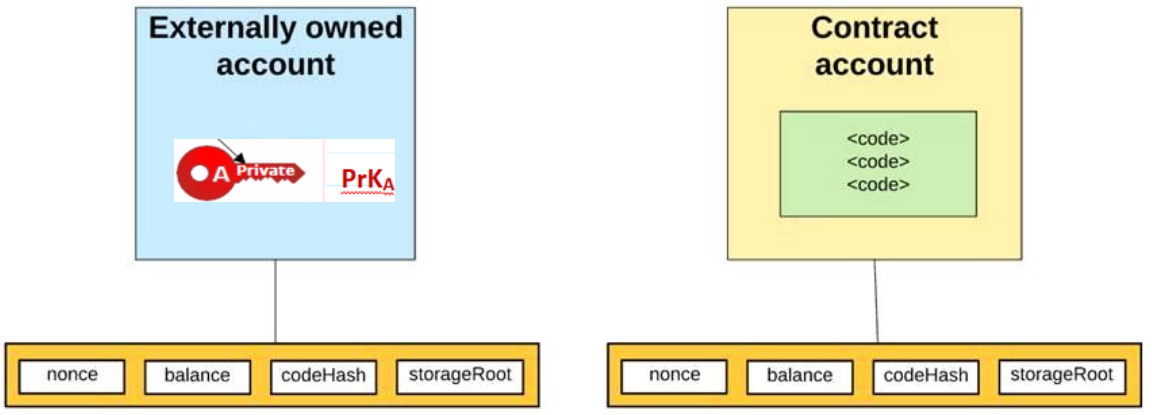


There are two practical types of transaction, contract creation and message call.

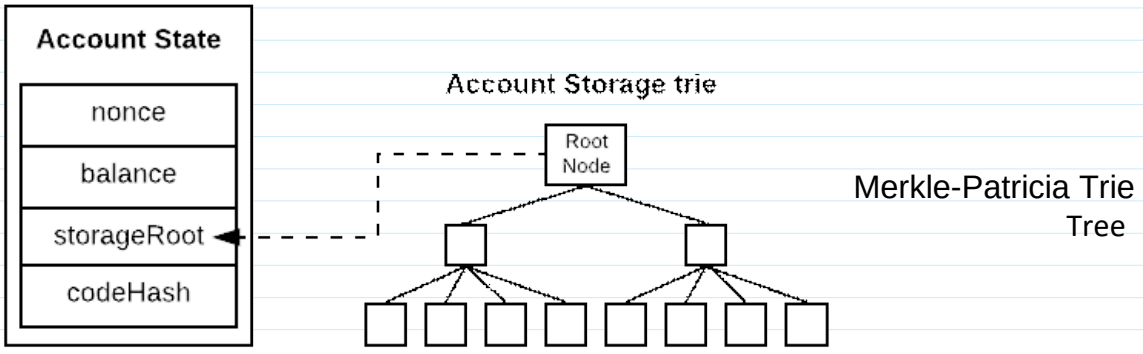
Account state

The **account state** consists of four components, which are present regardless of the type of account:

- **nonce**: If the account is an externally owned account, this number represents the number of transactions sent from the account's address. If the account is a contract account, the nonce is the number of contracts created by the account.
- **balance**: The number of Wei owned by this address. There are $\mathbf{10^{18}}$ Wei per Ether.
- **storageRoot**: A hash of the root node of a **Merkle Patricia tree** (we'll explain Merkle trees later on). This tree encodes the hash of the storage contents of this account, and is empty by default.
- **codeHash**: The hash of the EVM (more on this later) code of this account. For contract accounts, this is the code that gets hashed and stored as the **codeHash**. For externally owned accounts, the **codeHash** field is the hash of the empty string.

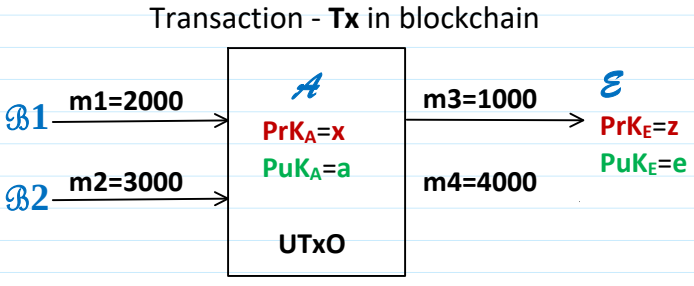


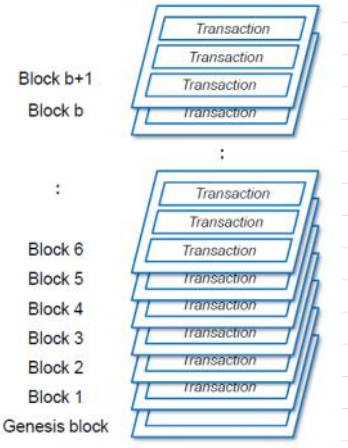
One important details about the **account state** is that **all fields (except the codeHash) are mutable**.
For example, when one account sends some Ether to another, the nonce will be incremented and the balance will be updated to reflect the new balance.
One of the consequences of the codeHash being immutable is that if you deploy a contract with a bug, you can't update the same contract. You need to deploy a new contract (*the buggy version will be available forever*). This is why it is important to use Truffle to develop and test your smart contracts and follow the best practices when working with Solidity.



The **Account Storage trie is Merkle-Patricia Trie where the data associated with an account is stored**.
This is only relevant for Contract Accounts,
For EOAs the storageRoot is empty and the codeHash is the hash of an empty string.
All smart contract data is persisted in the account storage trie as a mapping between 32-bytes integers.
We won't discuss in details how the contract data is persisted in the account state trie.
If you really want to learn about the internals, I suggest reading this post.
The hash of an account storage root node is persisted in the storageRoot field in the account state of the respective account.

Ethereum can also be seen as a stack of transactions.
Stack of transactions --> Ledger

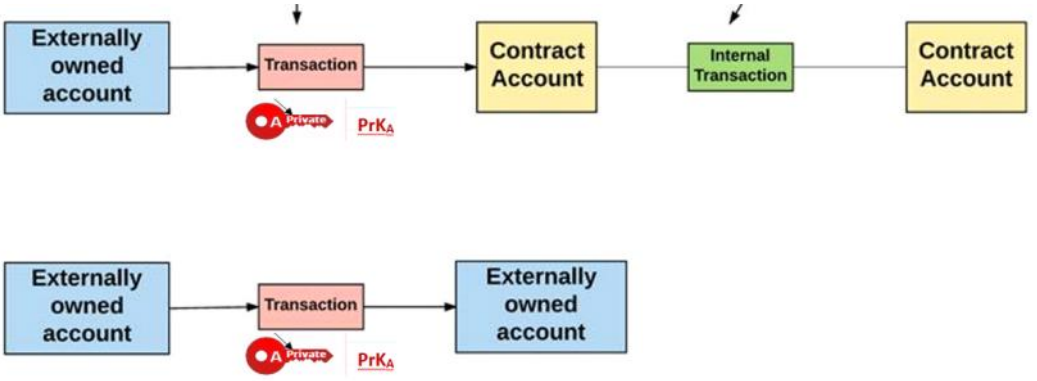




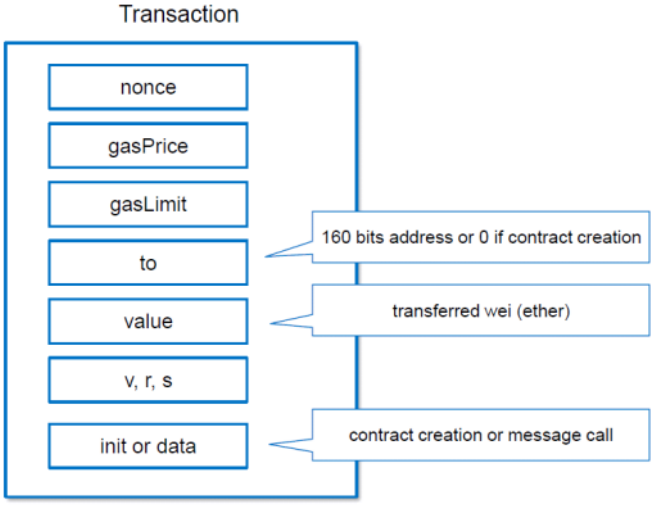
T**ransactions are what makes the state change from the current state to the next state**.
In Ethereum, we have three types of transactions:
1. Transactions that **transfer value** between two EOAs, e.g., change the sender and receiver account balances

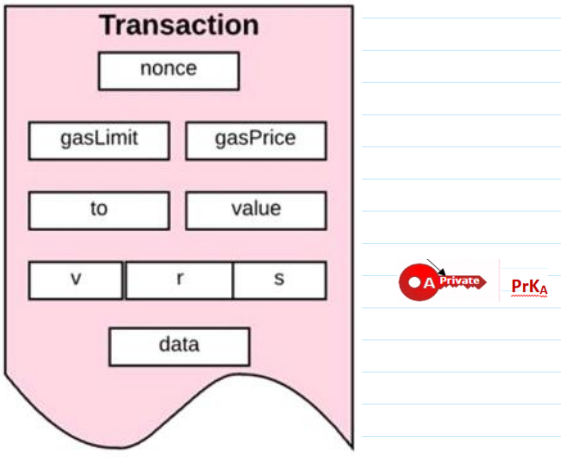
2. Transactions that **send a message call** to a contract, e.g., set a value in the smart contract by sending a message call that executes a setter method
3. Transactions that **deploy a contract** (therefore, create an account, the contract account

Technically, types 1 and 2 are the same. transactions that send message calls that affect an account state, either EOA or contract accounts. But is it easier to think about them as three different types



Fields of a transaction





The fields of a transaction:

- **Nonce:** Number of transactions sent by the account that created the transaction.
- **gasPrice:** Value in **Wei** that will be paid per unit of gas for the computation costs of executing this transaction.
- **gasLimit:** Maximum amount of gas to be used while executing this transaction. In the case if limit is exceeded …
- **to:** If this transaction is transferring Ether, address of the EOA account that will receive a value transfer.
  If this transaction is sending a message to a contract (e.g, calling a method in the smart contract), this is address of the contract. If this transactions is creating a contract, this value is always empty.
- **value:** If this transaction is transferring Ether, amount in **Wei** that will be transferred to the

recipient account.
If this transaction is sending a message to a contract, amount of Wei payable by the smart contract receiving the message.
If this transaction is creating a contract, this is the amount of **Wei** that will be added to the balance of the created contract.

- **v, r, s:** Values used in the cryptographic signature of the transaction used to determine the sender of the transaction: **v** - version; **r** - first e-signature component; **s** - second e-signature component.
- **data:** (only for value transfer and sending a message call to a smart contract)
  Input data of the message call (e.g, imagine you are trying to execute a setter method in your smart contract, the data field would contain the identifier of the setter method and the value that should be passed as parameter).
- **init:** (only for contract creation)
  The EVM-code utilized for initialization of the contract.

Some fields like the *data* field or the *init* field require you to have a deeper understanding of the internals of Ethereum to really understand what they mean and how to use them. This is not the time to deeply understand any of these fields.
Not surprisingly, all transactions in a block are stored in a trie. And the root hash of this trie is stored in the... block header! Let's take a look into the anatomy of an Ethereum block.