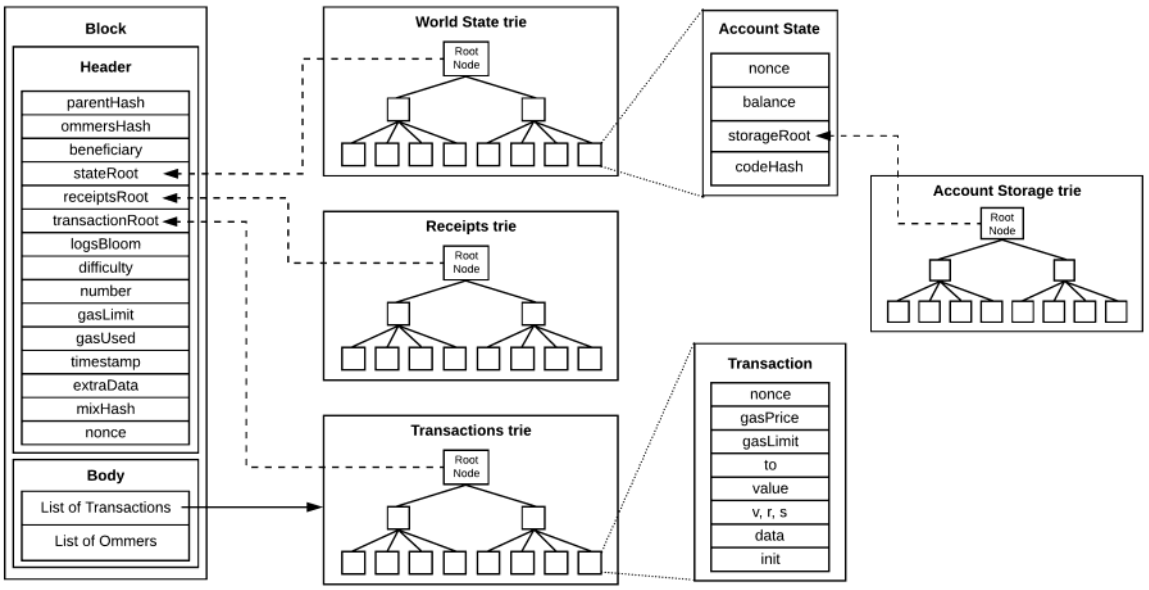
Conclusion

Basically, Ethereum has 4 types of tries:

1. **The world state trie contains the mapping between addresses and account states**. The hash of the root node of the world state trie is included in a block (in the **stateRoot** field) to represent the current state when that block was created. We only have one world state trie.
2. **The account storage trie contains the data associated to a smart contract**. The hash of the root node of the Account storage trie is included in the sccount state (in the **storageRoot** field). We have one Account storage trie for each account.
3. **The transaction trie contains all the transactions included in a block**. The hash of the root node of the Transaction trie is included in the block header (in the **transactionsRoot** field). We have one transaction trie per block.
4. **The transaction receipt trie contains all the transaction receipts for the transactions included in a block**. The hash of the root node of the transaction receipts trie is included in also included in the block header (in the **receiptsRoot** field); We have one transaction receipts trie per block.

And the objects that we discussed are:

1. **World state**: the hard drive of the distributed computer that is Ethereum. It is a mapping between addresses and account states.
2. **Account state**: stores the state of each one of Ethereum's accounts. It also contains the storageRoot of the account state trie, that contains the storage data for the account.
3. **Transaction**: represents a state transition in the system. It can be a funds transfer, a message call or a contract deployment.
4. **Block**: contains the link to the previous block (parentHash) and contains a group of transactions that, when executed, will yield the new state of the system. It also contains the **stateRoot**, the **transactionRoot** and the **receiptsRoot**, the hash of the root nodes of the world state trie, the transaction trie and the transaction receipts trie, respectively.



WHAT IS A SMART CONTRACT?

A "smart contract" is simply a program that runs on the Ethereum blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.

Smart contracts are a type of Ethereum account. This means they have a balance and can be the target of transactions. However they're not controlled by a user, instead they are deployed to the network and run as programmed. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce them via the code. Smart contracts cannot be deleted by default, and interactions with them are irreversible.

PREREQUISITES

If you're just getting started or looking for a less technical introduction, we recommend our introduction to smart contracts.

From <https://ethereum.org/en/developers/docs/smart-contracts/>

https://ethereum.org/en/developers/docs/accounts/#externally-owned-accounts-eoas

An Ethereum account is an entity with an ether (ETH) balance that can send transactions on Ethereum. Accounts can be user-controlled or deployed as smart contracts.

PREREQUISITES

To help you better understand this page, we recommend you first read through our introduction to Ethereum.

ACCOUNT TYPES

Ethereum has two account types:

- Externally-owned account (EOA) – controlled by anyone with the private keys
- Contract account – a smart contract deployed to the network, controlled by code. Learn about smart

contracts

Both account types have the ability to:

- Receive, hold and send ETH and tokens
- Interact with deployed smart contracts

Key differences

Externally-owned

- Creating an account costs nothing
- Can initiate transactions
- Transactions between externally-owned accounts can only be ETH/token transfers
- Made up of a cryptographic pair of keys: public and private keys that control account activities

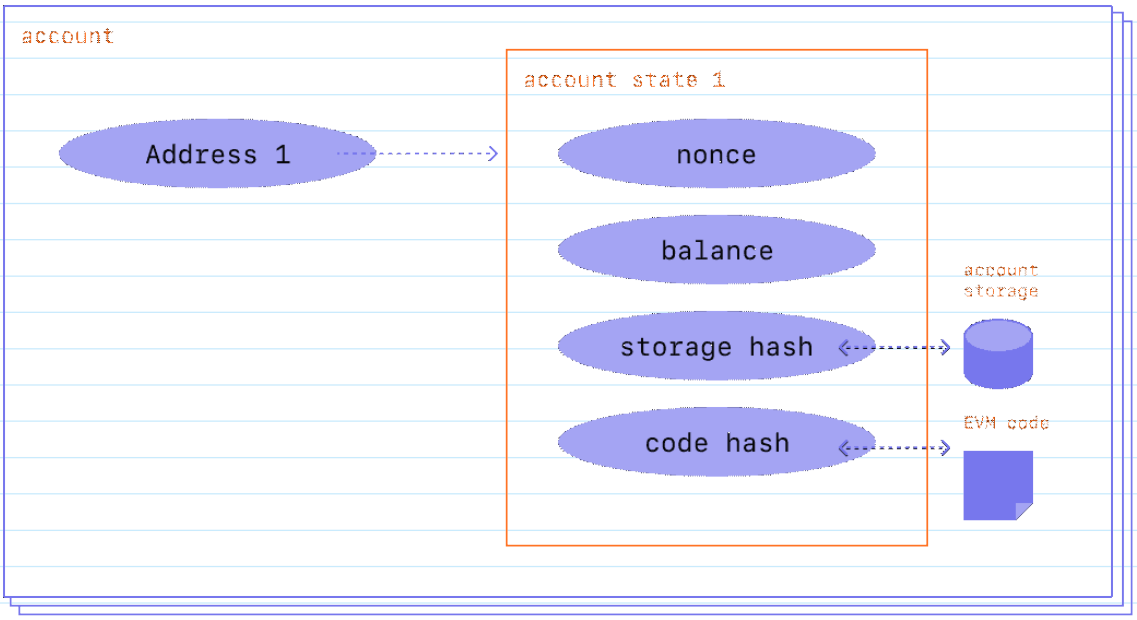
Contract

- Creating a contract has a cost because you're using network storage and processors computations resources
- Can only send transactions in response to receiving a transaction
- Transactions from an external account to a contract account can trigger code which can execute many different actions, such as transferring tokens or even creating a new contract or *message call*
- Contract accounts don't have **private keys**. Instead, they are controlled by the logic of the smart contract code

AN ACCOUNT EXAMINED

Ethereum accounts have four fields:

- nonce – A counter that indicates the number of transactions sent from an externally-owned account or the number of contracts created by a contract account. Only one transaction with a given nonce can be executed for each account, protecting against replay attacks where signed transactions are repeatedly broadcast and re-executed.
- balance – The number of wei owned by this address. Wei is a denomination of ETH and there are 1e+18 wei per ETH.
- codeHash – This hash refers to the *code* of an account on the Ethereum virtual machine (EVM). Contract accounts have code fragments programmed in that can perform different operations. This EVM code gets executed if the account gets a message call. It cannot be changed, unlike the other account fields. All such code fragments are contained in the state database under their corresponding hashes for later retrieval. This hash value is known as a codeHash. For externally owned accounts, the codeHash field is the hash of an empty string.
- storageRoot – Sometimes known as a storage hash. A 256-bit hash of the root node of a Merkle Patricia trie that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit integer values. This trie encodes the hash of the storage contents of this account, and is empty by default.



https://ethereum.org/en/smart-contracts/

Smart contracts are the fundamental building blocks of Ethereum's application layer. They are computer programs stored on the blockchain that follow "if this then that" logic, and are guaranteed to execute according to the rules defined by its code, which cannot be changed once created.
Nick Szabo coined the term "smart contract". In 1994, he wrote an introduction to the concept(opens in a new tab), and in 1996 he wrote an exploration of what smart contracts could do(opens in a new tab).
Szabo envisioned a digital marketplace where automatic, cryptographically-secure processes enable transactions and business functions to happen without trusted intermediaries. Smart contracts on Ethereum put this vision into practice.

Trust in conventional contracts

One of the biggest problems with a traditional contract is the need for trusted individuals to follow through with the contract's outcomes.
Here is an example:
Alice and Bob are having a bicycle race. Let's say Alice bets Bob $10 that she will win the race. Bob is confident he'll be the winner and agrees to the bet. In the end, Alice finishes the race well ahead of Bob and is the clear winner. But Bob refuses to pay out on the bet, claiming Alice must have cheated.
This silly example illustrates the problem with any non-smart agreement. Even if the conditions of the agreement get met (i.e. you are the winner of the race), you must still trust another person to fulfill the agreement (i.e. payout on the bet).

A digital vending machine

A simple metaphor for a smart contract is a vending machine, which works somewhat similarly to a smart contract - specific inputs guarantee predetermined outputs.

- You select a product
- The vending machine displays the price
- You pay the price
- The vending machine verifies that you paid the right amount
- The vending machine gives you your item *and change*

The vending machine will only dispense your desired product after all requirements are met. If you don't select a product or insert enough money, the vending machine won't give out your product.

Perhaps the best metaphor for a smart contract is a vending machine, as described by Nick Szabo(opens in a new tab). With the right inputs, a certain output is guaranteed.
To get a snack from a vending machine:

```
1  money + snack selection = snack dispensed
2
```

This logic is programmed into the vending machine.
A smart contract, like a vending machine, has logic programmed into it. Here's a simple example of how this vending machine would look if it were a smart contract written in Solidity:

From <https://ethereum.org/en/developers/docs/smart-contracts/>

```
pragma solidity 0.8.7;
2
3contract VendingMachine {
4
5    // Declare state variables of the contract
6    address public owner;
7    mapping (address => uint) public cupcakeBalances;
8
9    // When 'VendingMachine' contract is deployed:
10    // 1. set the deploying address as the owner of the contract
11    // 2. set the deployed smart contract's cupcake balance to 100
12    constructor() {
13        owner = msg.sender;
14        cupcakeBalances[address(this)] = 100;
15    }
16
17    // Allow the owner to increase the smart contract's cupcake balance
18    function refill(uint amount) public {
19        require(msg.sender == owner, "Only the owner can refill.");
20        cupcakeBalances[address(this)] += amount;
21    }
22
23    // Allow anyone to purchase cupcakes
24    function purchase(uint amount) public payable {
25        require(msg.value >= amount * 1 ether, "You must pay at least 1 ETH per cupcake");
26        require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in stock to complete this
purchase");
27        cupcakeBalances[address(this)] -= amount;
28        cupcakeBalances[msg.sender] += amount;
29    }
30}
```

Like how a vending machine removes the need for a vendor employee, smart contracts can replace intermediaries in many industries.

PERMISSIONLESS

Anyone can write a smart contract and deploy it to the network. You just need to learn how to code in a smart contract language, and have enough ETH to deploy your contract. Deploying a smart contract is technically a transaction, so you need to pay gas in the same way you need to pay gas for a simple ETH transfer. However, gas costs for contract deployment are far higher.
Ethereum has developer-friendly languages for writing smart contracts:

- Solidity
- Vyper

COMPOSABILITY
Smart contracts are public on Ethereum and can be thought of as open APIs. This means you can call other smart contracts in your own smart contract to greatly extend what's possible. Contracts can even deploy other contracts.
Learn more about smart contract composability.
LIMITATIONS
Smart contracts alone cannot get information about "real-world" events because they can't retrieve data from off-chain sources. This means they can't respond to events in the real world. This is by design. Relying on external information could jeopardise consensus, which is important for security and decentralization. However, it is important for blockchain applications to be able to use off-chain data. The solution is oracles which are tools that ingest off-chain data and make it available to smart contracts.
Another limitation of smart contracts is the maximum contract size. A smart contract can be a maximum of 24KB or it will run out of gas. This can be circumnavigated by using The Diamond Pattern(opens in a new tab).
MULTISIG CONTRACTS
Multisig (multiple-signature) contracts are smart contract accounts that require multiple valid signatures to execute a transaction. This is very useful for avoiding single points of failure for contracts holding substantial amounts of ether or other tokens. Multisigs also divide responsibility for contract execution and key management between multiple parties and prevent the loss of a single private key leading to irreversible loss of funds. For these reasons, multisig contracts can be used for simple DAO governance. Multisigs ***Threshold*** require N signatures out of M possible acceptable signatures (where $N \leq M$, and $M > 1$) in order to execute. N = 3, M = 5 and N = 4, M = 7 are commonly used. A 4/7 multisig requires four out of seven possible valid signatures. This means the funds are still retrievable even if three signatures are lost. In this case, it also means that the majority of key-holders must agree and sign in order for the contract to execute.
SMART CONTRACT RESOURCES
OpenZeppelin Contracts - *Library for secure smart contract development.*

- openzeppelin.com/contracts/(opens in a new tab)
- GitHub(opens in a new tab)
- Community Forum(opens in a new tab)

Automatic execution
The main benefit of a smart contract is that it deterministically executes unambiguous code when certain conditions are met. There is no need to wait for a human to interpret or negotiate the result. This removes the need for trusted intermediaries.
For example, you could write a smart contract that holds funds in escrow for a child, allowing them to withdraw funds after a specific date. If they try to withdraw before that date, the smart contract won't execute. Or you could write a contract that automatically gives you a digital version of a car's title when you pay the dealer.
Predictable outcomes
Traditional contracts are ambiguous because they rely on humans to interpret and implement them. For example, two judges might interpret a contract differently, which could lead to inconsistent decisions and unequal outcomes. Smart contracts remove this possibility. Instead, smart contracts execute precisely based on the conditions written within the contract's code. This precision means that given the same circumstances, the smart contract will produce the same result.
Public record

Smart contracts are useful for audits and tracking. Since Ethereum smart contracts are on a public blockchain, anyone can instantly track asset transfers and other related information. For example, you can check to see that someone sent money to your address.

Till this place