INTRODUCTION to BLOCKCHAIN

CHAPTER4: SMART-CONTRACT & DAPPS

Dr. Noureddine Lasla

Chapter Overview

Objective:

Understanding smart-contracts and their use-cases

Key Areas of Focus:

- Smart-contract definition
- Ethereum
  - Ethereum Accounts, Ethereum Transactions,
  - Ethereum Blocks, Patricia trie
- Smart-contract Development
  - Programming languages, IDEs, Test networks, Wallet

What is Smart Contract?

Contract: (WIKIPEDIA The Free Encyclopedia)

- A voluntary arrangement between two or more parties that is **enforceable by law** as a binding legal agreement.

Smart contract:

- Refers to any contract capable of **automatically enforcing itself**, without a third party between individual participants.

What is Smart Contract?

- “A set pf promises,
- specified in digital form,
- including protocols
- within which the parties perform on these promises. “

Nick Szabo, 1996

However…

- smart contract may not be so smart
- smart contract may not be contract

Bit Gold 1998

What is Smart Contract?

From a software developer's perspective:

User-defined programs running on top of a blockchain

Support execution of Turing complete code

ETHEREUM:
SMART CONTRACT ENABLER

Ethereum

In 2013, Vitalik Bultern (was 19 at that time) proposed a new blockchain (Ethereum) platform that support a self program execution for building decentralized applications.

- New and improved version of Bitcoin
- Rich prog. language to create contacts
- Create a decentralized global computer

  (exec. the binary code and save the sate in the mem.)
- New currency (Ether): pricing/gas for transaction fees and comp. services

Ethereum → blockchain to run program in trusted environment

Ethereum

Ethereum is a decentralized platform for building **dApps**.

- **Ethereum client**:
  - **E**thereum **V**irtual **M**achine (**EVM**) engine:
    - Smart contract execution & gas management
  - The memory pool: for pending tx …
  - A JSON-RPC API:
    - JSON-RPC server: interface through which external applications (like wallets, dApps, explorers) communicate with the Ethereum client
    - Web3 interface: interacting with the Ethereum from web

Bitcoin Vs. Ethereum

Bitcoin (TXs ledger)	**Ethereum (Balance ledger)**
Genesis: January 2009	July 2015
Transaction Input & Outputs	State Transitions
Private keys own simple values (UTXO)	Private kye own accounts
All values are owned by a private key	There are internal and external accounts
Non Turing (Script)	Turing complete (Solidity, Vyper, Yul, …)
Merkle trees: Transactions	Transactions, state, storage, receipts

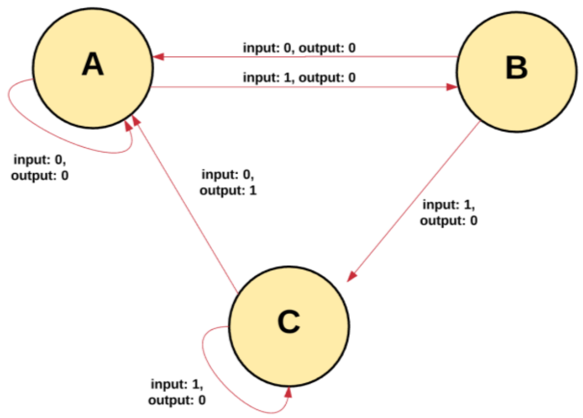
Bitcoin Vs. Ethereum

Bitcoin (TXs ledger)	**Ethereum (Balance ledger)**
Currency: Bitcoin	ETH
Bitcoin is subdivided into smaller units ***satoshi***, 1 BTC = 10^8 sats	Ether is subdivided into smaller units ***wei***, 1 ETH = 10^18 wei
Reward: 3.13 BTC/block	2 ETH/block
Monetary policy: 1/2s every 210,000 blocks (~4 years)	Fixed, but change by update (was 5 ETH/blocks)
Fees: voluntary	Needed
Consensus: PoW	Was PoW, now PoS

Ethereum State Machine

Ethereum as a State Machine

- Ethereum is a **state machine**: It transitions from one state to another is based on transactions.
- **State**: A snapshot of all accounts, balances, and smart contract data at a given time.
- **Transactions**: Actions that modify the state
    - Transferring ETH,
    - calling a smart contract.



Ethereum Accounts

Account has state and 20-byte identifier.

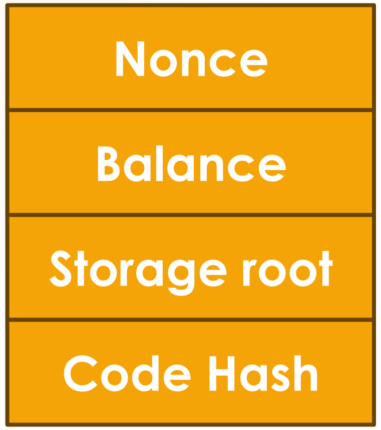
Two types of account:

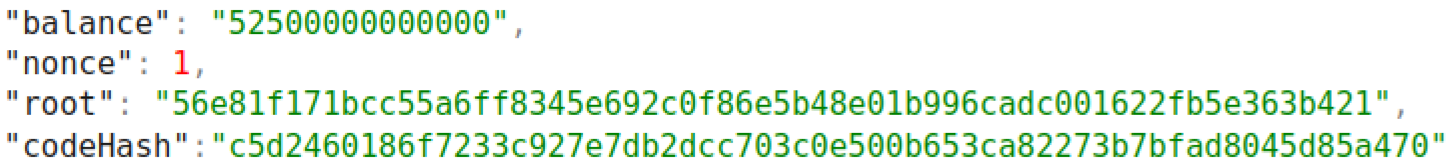
- **Externally owned accounts (EOAs):** accounts that are controlled by users
  - EOAs can hold and manage Ether (ETH) and other tokens
- **Contract accounts**: are owned by smart contracts and can be used to interact with the Ethereum blockchain
  - When a smart contract is deployed, it is assigned a contract address

Ethereum Accounts

The account has a **state** which consists of four components:

- **Nonce**:
  - EOAs: the number of transactions sent from the account’s address.
  - Contract account: the number of contracts created by the account.
- **Balance**: The number of Wei owned by this address.
- **Storage Root**: A 256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account.
- **Code Hash**: The hash of the EVM (Ethereum Virtual Machine) code of this account
  - Contract accounts: this is the code that gets hashed and stored as the **codeHash**.
  - EOAs: the **codeHash** field is the hash of the empty string.





```
"balance": "52500000000000",
"nonce": 1,
"root": "56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
"codeHash":"c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470"
```

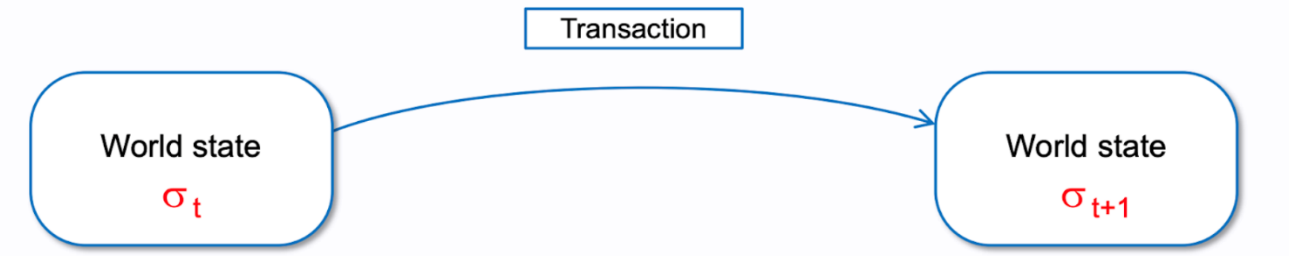
EOAs Vs. Contract Account

EOAs	**Contract account**
Creation: by users	By deploying a smart contract to the Ethereum network
Key Pair: EOAs have a private-public key pair	No public or private keys
Control: by users	by the logic of the smart contract code
Interactions: through transactions	through transactions and events, which are logged on the blockchain and can be observed by external parties
Actions: perform actions explicitly allowed by the user	perform actions explicitly allowed by the code

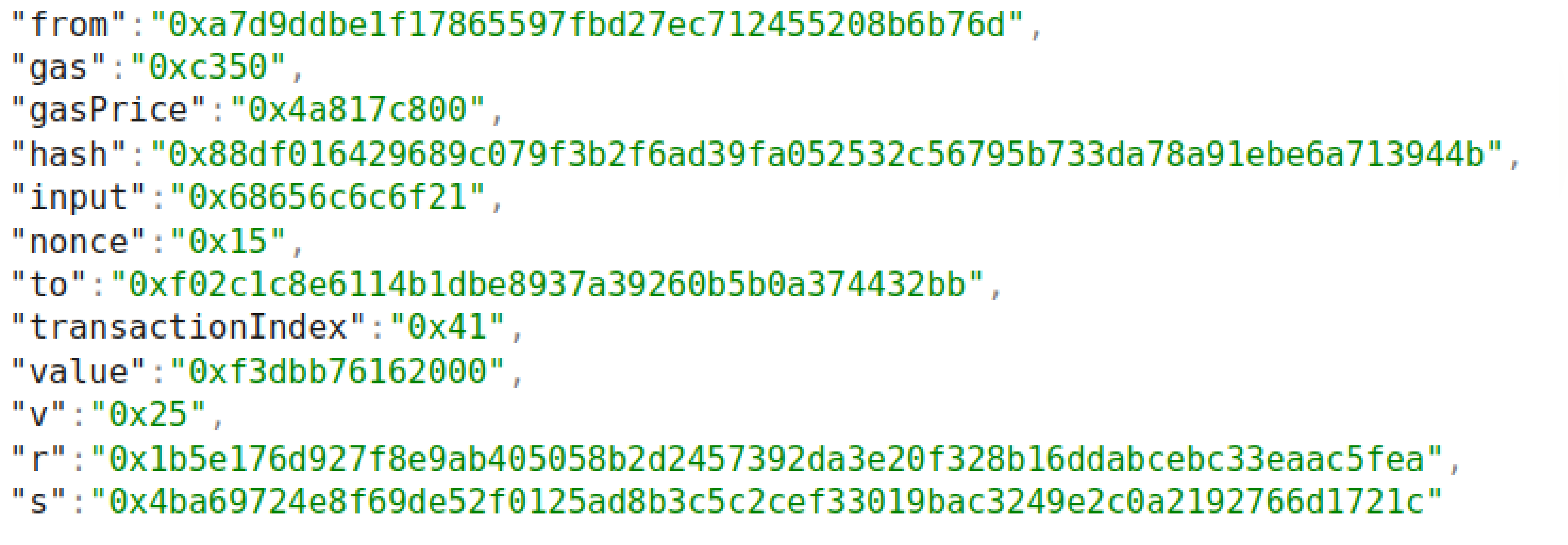
Ethereum Transactions

Transactions move the state of an account within the global state- one state to the next

- **Formal definition**: A transaction is a cryptographically signed piece of instruction that is generated by an externally owned account, serialized, and then submitted to the blockchain
- **Two types**:
  - Message calls
  - Contract creation



Ethereum Transactions



```
"from":"0xa7d9ddbe1f17865597fbd27ec712455208b6b76d",
"gas":"0xc350",
"gasPrice":"0x4a817c800",
"hash":"0x88df016429689c079f3b2f6ad39fa052532c56795b733da78a91ebe6a713944b",
"input":"0x68656c6c6f21",
"nonce":"0x15",
"to":"0xf02c1c8e6114b1dbe8937a39260b5b0a374432bb",
"transactionIndex":"0x41",
"value":"0xf3dbb76162000",
"v":"0x25",
"r":"0x1b5e176d927f8e9ab405058b2d2457392da3e20f328b16ddabcebc33eaac5fea",
"s":"0x4ba69724e8f69de52f0125ad8b3c5c2cef33019bac3249e2c0a2192766d1721c"
```

v, **r**, **s**: Components of the ECDSA (Elliptic Curve Digital Signature Algorithm)

Gas and Payment

Every computation that occurs as a result of transaction on Ethereum network incurs a fee called **gas**

- **Gas** is the unit used to measure the fees for a particular computation
- **Gas price** is the amount of Ether you are willing to spend on every unit of gas
  - Measured in “gwei”- 1 gewi = 1,000,000,000 wei
- With every transaction, a sender sets a **gas limit** and a **gas price**
  - **gas price x gas limit** = max **amount** of wei **sender is willing to pay** for transaction

Ethereum Block

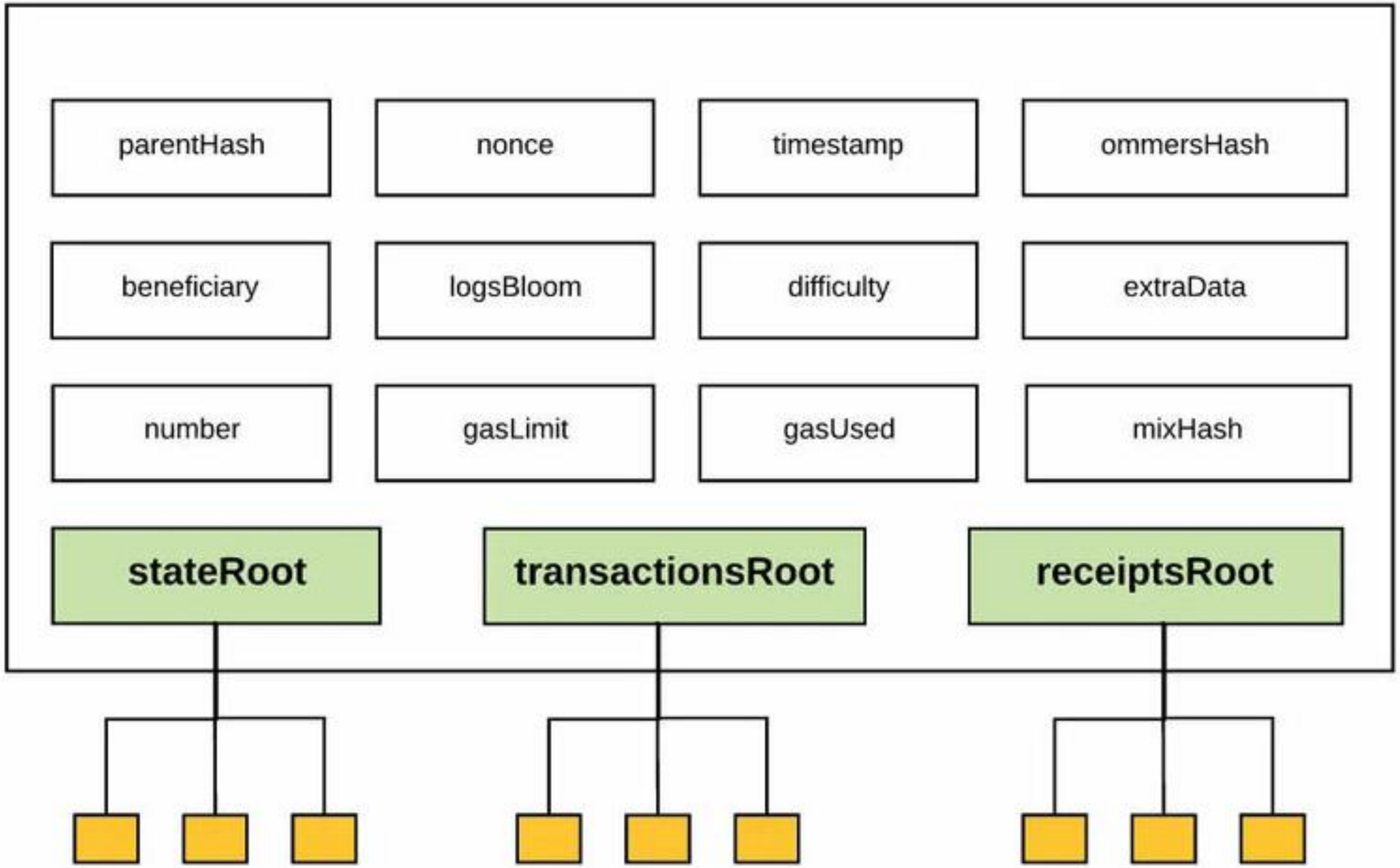
Purpose of blocks:

- Store transactions (e.g., ETH transfers, smart contract calls).
- Update the global state of Ethereum (e.g., account balances, contract storage).
- Provide a tamper-proof record of all activity on the net.

Block structure:

- Block Header: Metadata about the block.
- Transactions List: List of transactions included in the block.

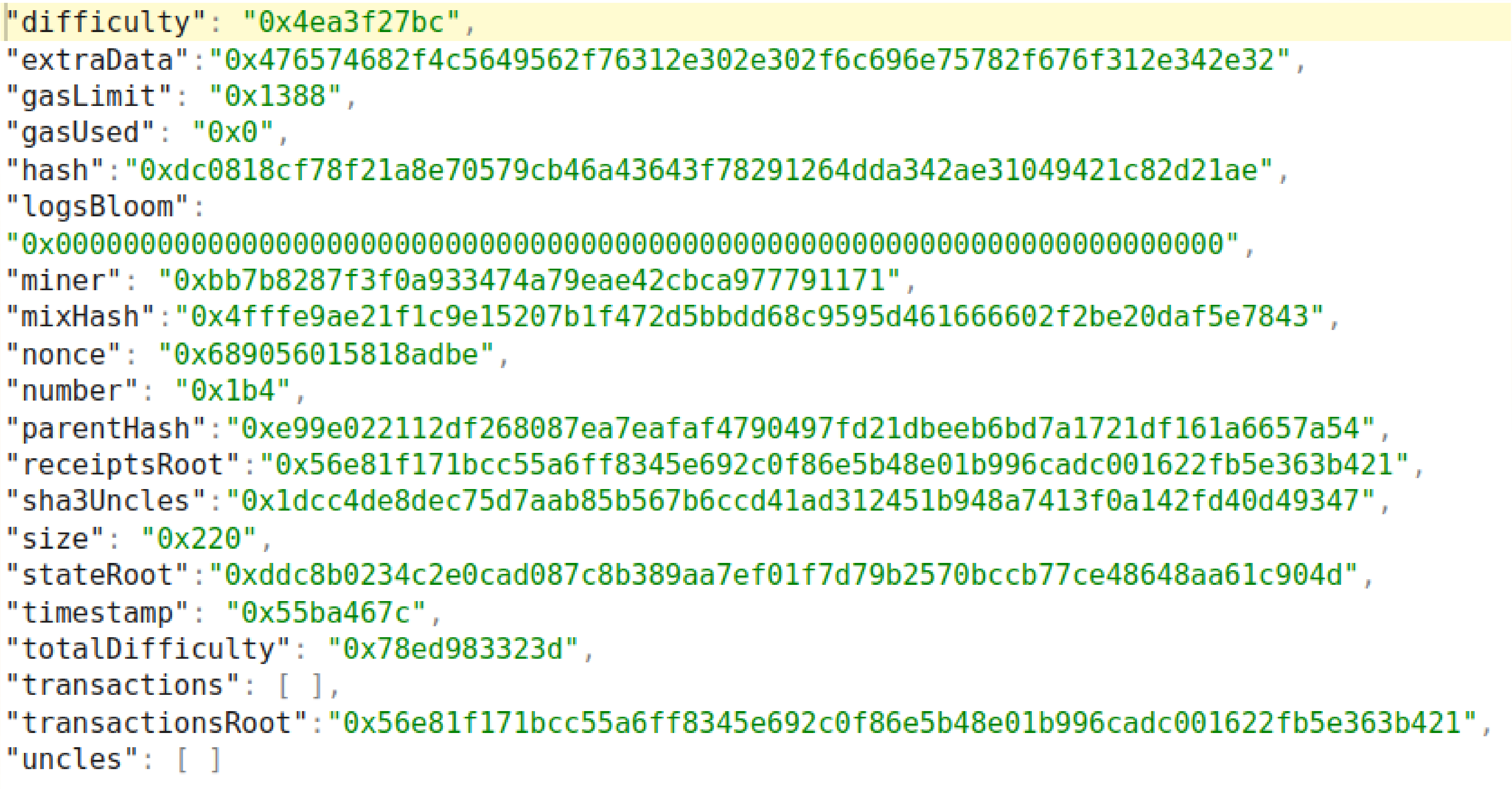
Ethereum Block (Header)



Ethereum Block (Header)

Parent Hash	Hash of the previous block (links blocks in the chain).
State Root	Hash of the root of the state trie (global state after applying transactions).
Transactions Root	Hash of the root of the transactions trie (list of transactions in the block).
Receipts Root	Hash of the root of the receipts trie (outcomes of transactions).
Block Number	Height of the block in the chain.
Gas Limit	Maximum gas allowed in the block.
Gas Used	Total gas used by transactions in the block.
Timestamp	When the block was mined.
Nonce	A value used in mining (Proof of Work).
Miner	Address of the miner who created the block.

Ethereum Block (Header)



```
"difficulty": "0x4ea3f27bc",
"extraData":"0x476574682f4c5649562f76312e302e302f6c696e75782f676f312e342e32",
"gasLimit": "0x1388",
"gasUsed": "0x0",
"hash":"0xdc0818cf78f21a8e70579cb46a43643f78291264dda342ae31049421c82d21ae",
"logsBloom":
"0x000000000000000000000000000000000000000000000000000000000000000000000",
"miner": "0xbb7b8287f3f0a933474a79eae42cbca977791171",
"mixHash":"0x4fffe9ae21f1c9e15207b1f472d5bbdd68c9595d461666602f2be20daf5e7843",
"nonce": "0x689056015818adbe",
"number": "0x1b4",
"parentHash":"0xe99e022112df268087ea7eafaf4790497fd21dbeeb6bd7a1721df161a6657a54",
"receiptsRoot":"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
"sha3Uncles":"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
"size": "0x220",
"stateRoot":"0xddc8b0234c2e0cad087c8b389aa7ef01f7d79b2570bccb77ce48648aa61c904d",
"timestamp": "0x55ba467c",
"totalDifficulty": "0x78ed983323d",
"transactions": [ ],
"transactionsRoot":"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
"uncles": [ ]
```

Merkle Patricia Trie

Ethereum uses a modified version of a Merkle tree called a Merkle Patricia Trie. It combines:

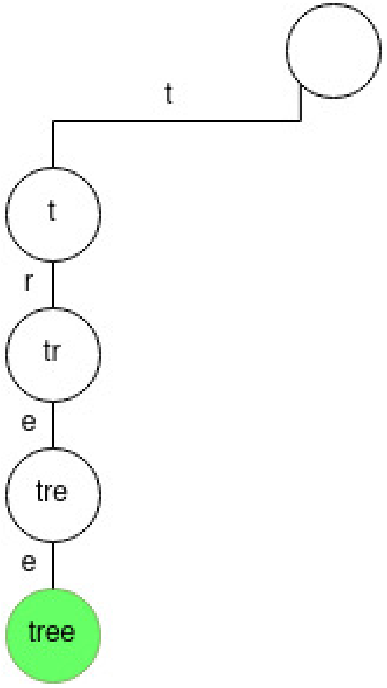
- Merkle Tree (you know it already)
- Patricia Trie: A radix tree that compresses paths for efficient storage and lookups.

The Merkle Patricia Trie is used for:

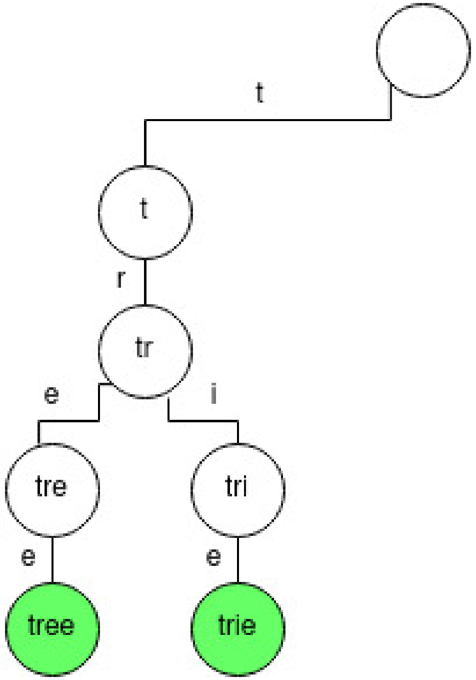
- State Trie: Stores all accounts and their current state.
- Transactions Trie: Stores all transactions in a block.
- Receipts Trie: Stores the outcomes of transactions (e.g., logs, gas used)

Trie: re**trie**val

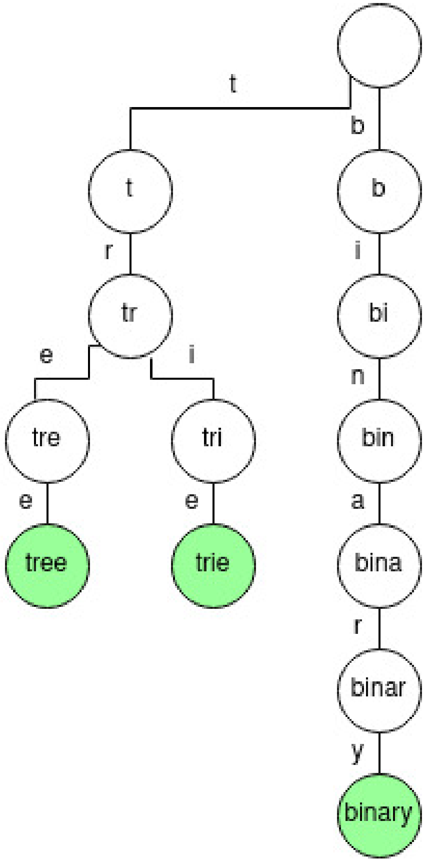
Insert *"tree"*



Insert *"trie"*. The prefix *tr* of the word is already represented. So we are adding a branch after the path *t->r*

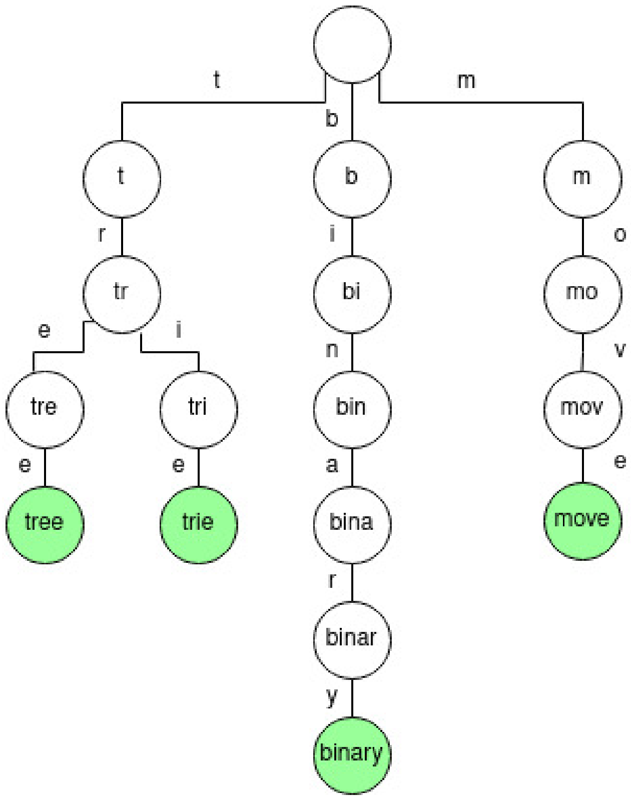


Insert *"binary"*.

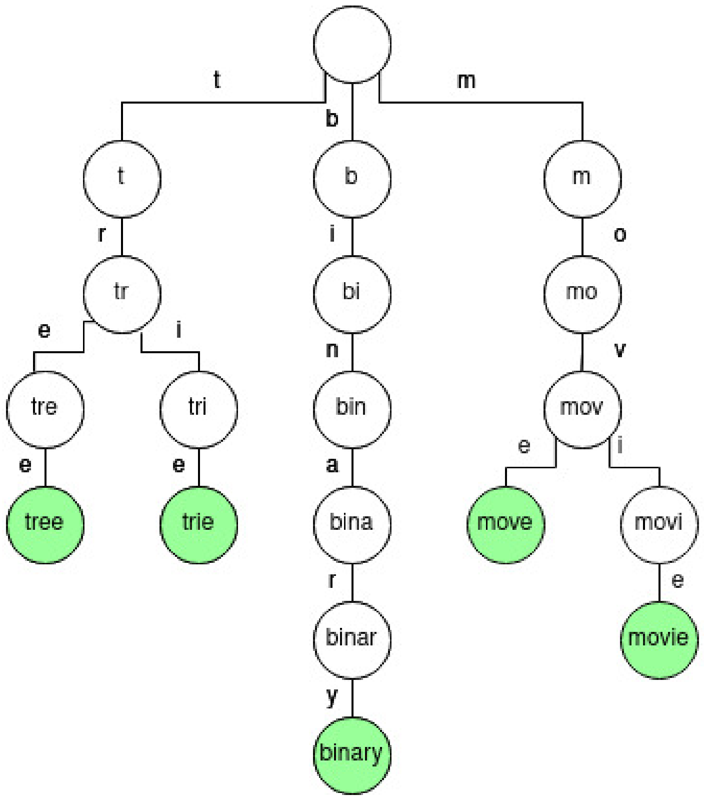


Trie: re**trie**val

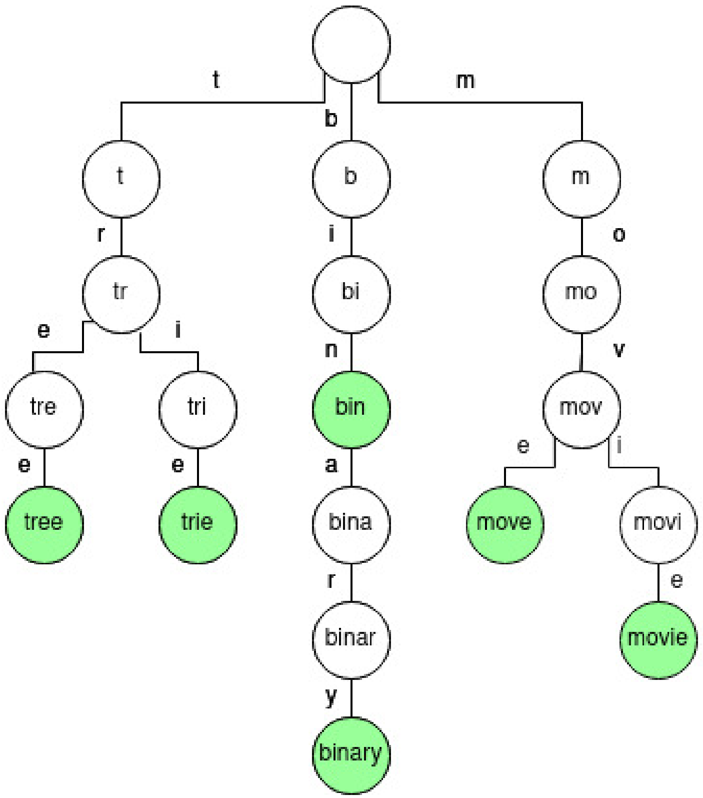
Insert *"move"*



Insert *"movie"*. Here also the prefix *mov* is already represented. We continue by adding a branch after the path *m->o->v*.



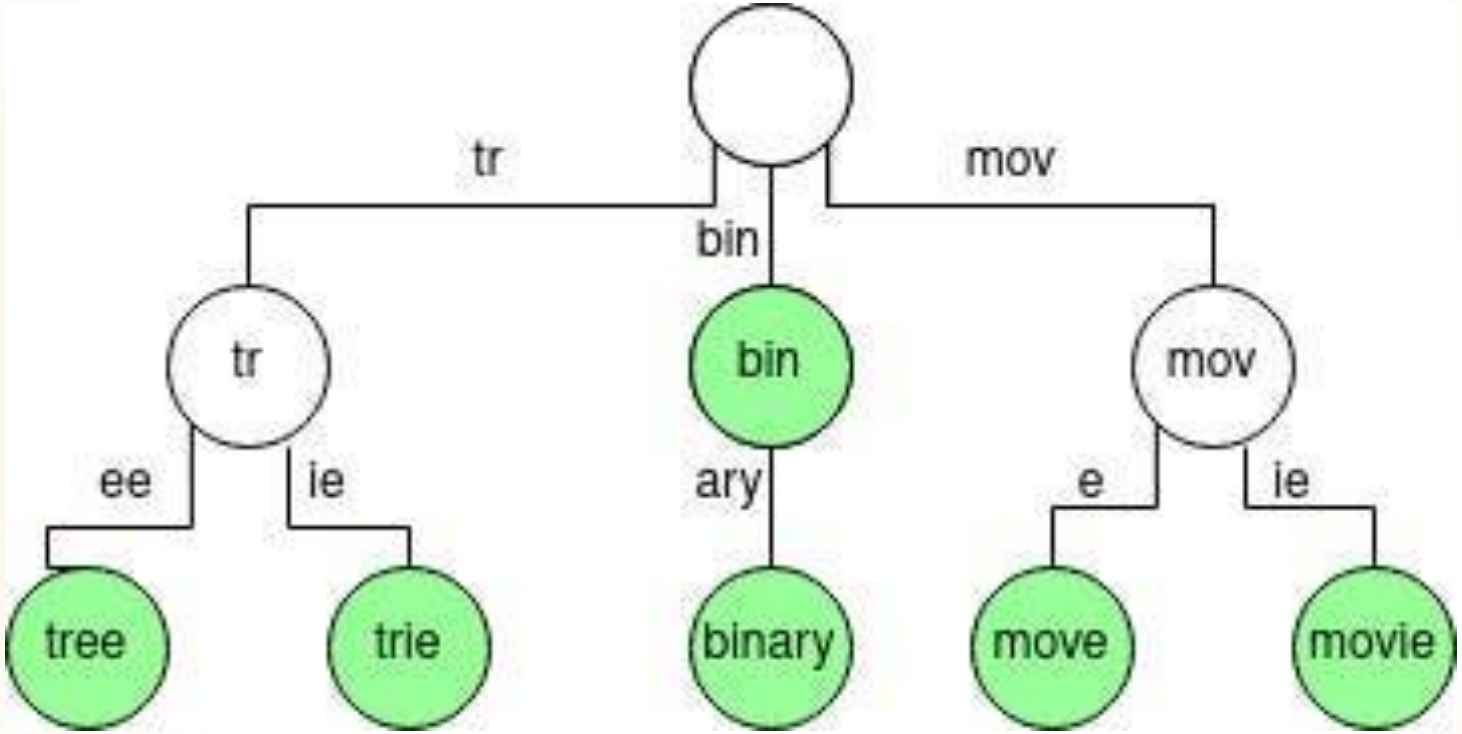
Insert *"bin"*. This word is a prefix of an already inserted word binary. So mark the node at the end of the path *b->i->n*.



Every node except the root node represents a prefix of a string, we call trie also "prefix-tree"

Patricia Trie

Patricia-Practical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric



Merkle Patricia Trie

State Root

Is the **root hash** of the **state trie**, which stores the **global state** of all **accounts** (balances, nonces, contract code, and storage).

- Each account is stored as a key-value pair in the trie:
  - `<key = account_address, value = account_data>`
- The trie is traversed, and each node is hashed.
- The state root (hash) is computed and included in the block header.
- Purpose: Allows anyone to verify the state of any account without storing the entire state.

Merkle Patricia Trie

Transactions Root

Is the root hash of the **transactions trie**, which stores **all transactions** in the block.

- Each transaction is stored as a key-value pair in the trie:
  - `<key = transaction_index, value = transaction_data>`
- The trie is traversed, and each node is hashed.
- Transaction trie never gets updated (similar to Merkle tree represent.)
- The state root (hash) is computed and included in the block header.
- <u>Purpose</u>: Ensures the integrity of the transactions in the block.

Merkle Patricia Trie

Receipts Root

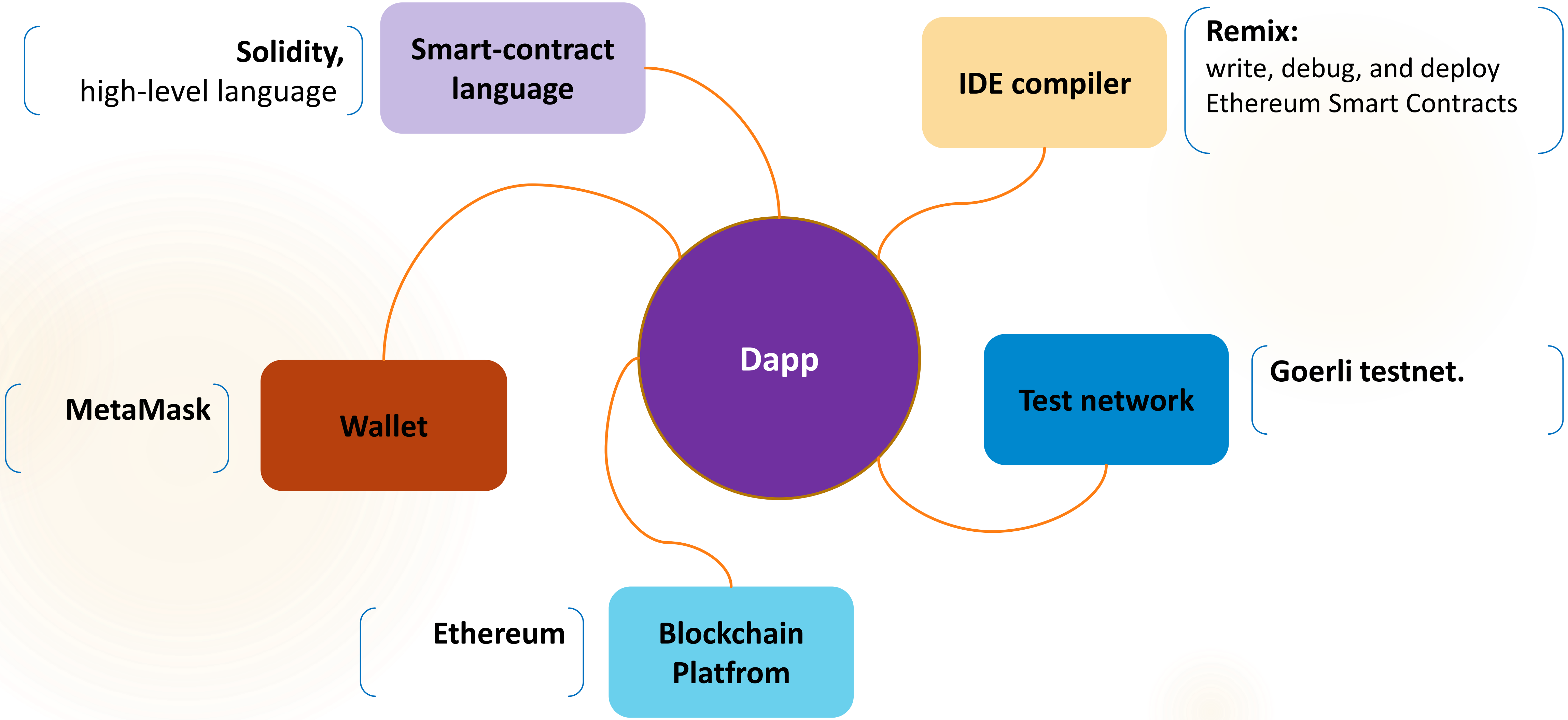
Is the root hash of the **receipts trie**, which stores the **outcomes of transactions** (e.g., logs, gas used, status)

- Each transaction receipt is stored as a key-value pair in the trie:
  - `<key = transaction_index, value = receipt_data>`
- The trie is traversed, and each node is hashed.
- Like transaction trie, receipt trie never gets updated
- Purpose: Allows anyone to verify the outcomes of transactions without storing all receipts.

Developing and Deploying SMART CONTRACT

REMIX

Development Environment



Dapp

Smart-contract language

Solidity,
high-level language

IDE compiler

Remix:
write, debug, and deploy Ethereum Smart Contracts

Wallet

MetaMask

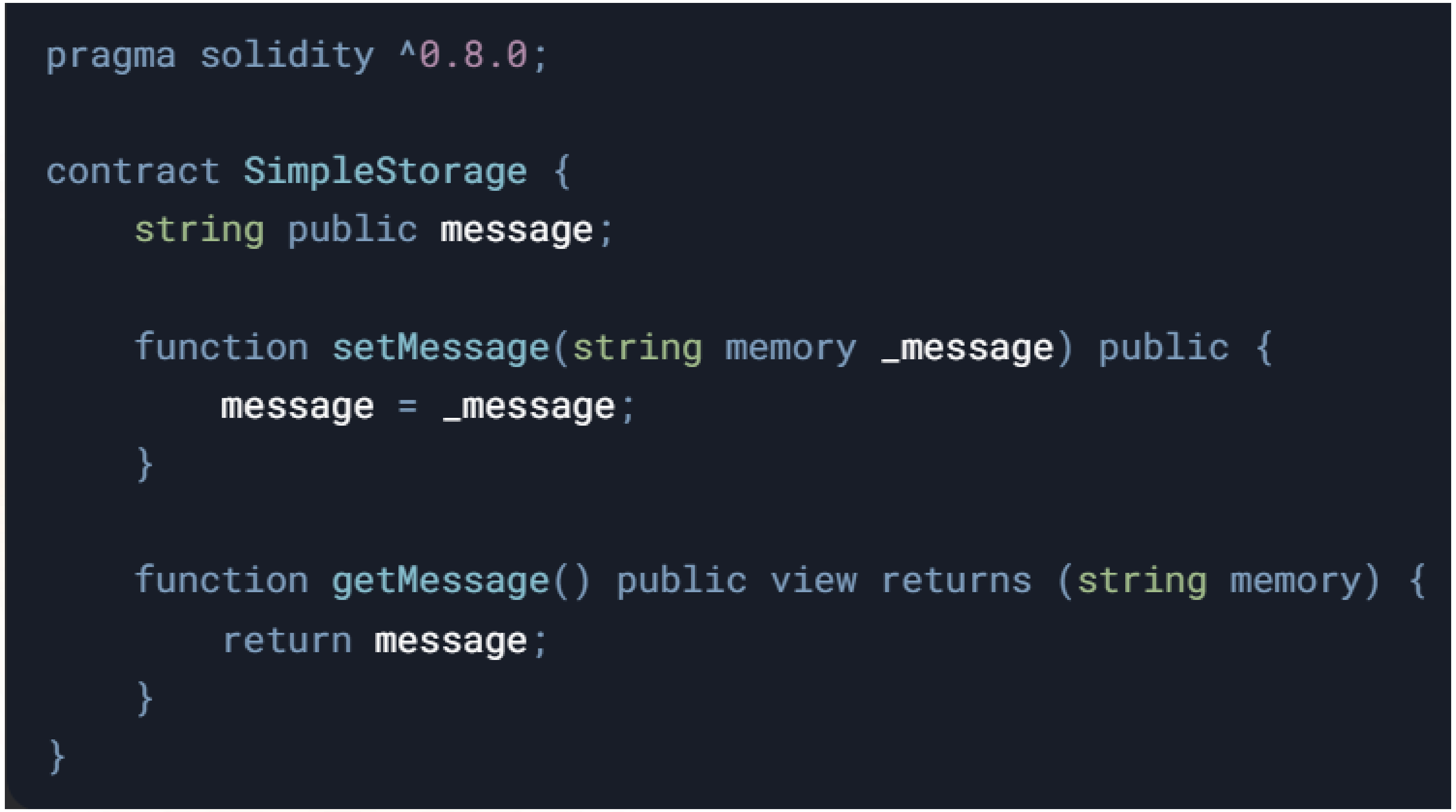
Test network

Goerli testnet.

Blockchain Platfrom

Ethereum

Writing a Smart Contract

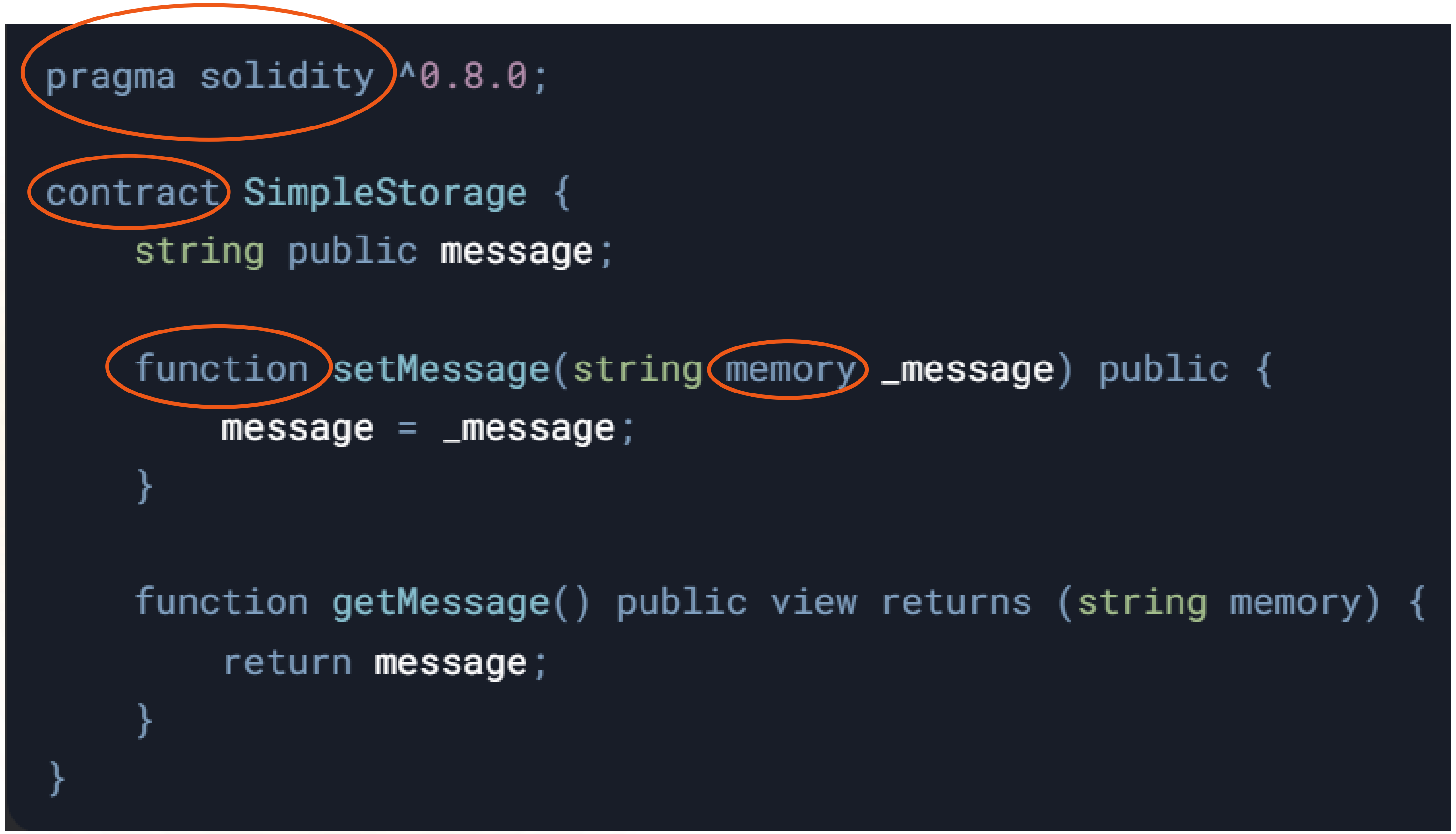


```
pragma solidity ^0.8.0;

contract SimpleStorage {
    string public message;

    function setMessage(string memory _message) public {
        message = _message;
    }

    function getMessage() public view returns (string memory) {
        return message;
    }
}
```

Programming Languages:

- **Solidity**: Most popular language similar to JavaScript
- **Vyper**: Python-based lang. simpler than solidity
- **Yul**: intermediate-level language designed for optimizations.
- **LLL** (Low-Level Lisp-like Language): provides direct access to the EVM bytecode,

Writing a Smart Contract



```
pragma solidity ^0.8.0;

contract SimpleStorage {
    string public message;

    function setMessage(string memory _message) public {
        message = _message;
    }

    function getMessage() public view returns (string memory) {
        return message;
    }
}
```

Code descriptions

- **`pragma solidity`**: specify solidity version
- **`contract`**: define the smart contract
- **`function`**: contains executable code
- **memory**: temp. data location
  - storage, calldata, stack

Writing a Smart Contract

Data types in Solidity

Value Types:

- **uint:** Unsigned integer (e.g., uint256).
- **int:** Signed integer (e.g., int128).
- **bool:** Boolean (true or false).
- **address:** Ethereum address (e.g., 0x...).
- **bytes:** Fixed-size byte arrays (e.g., bytes32).

Reference Types:

- **string:** Dynamic-sized string.
- **array:** Fixed or dynamic arrays (e.g., uint[]).
- **mapping:** Key-value pairs (e.g., mapping(address => uint) balances).

Special Types:

- **enum:** User-defined types (e.g., enum State { Created, Locked, Inactive }).
- **struct:** Custom data structures

Writing a Smart Contract
Variables in Solidity

State Variables::

- Stored permanently on the blockchain.
- **Example:** `unit public balance;`

Local Variables:

- Temporary variables used within functions.
- **Example:** `function foo() public {`

  `uint localVar = 10; }`

Global Variables:

- Provide information about the blockchain (msg.sender, block.timestamp).

Writing a Smart Contract

Functions in Solidity

```
Function functionName(parameters) visibility modifiers returns (returnType) {
      // Function body
}
```

Visibility:

- **public:** Accessible from anywhere.
- **private:** Only accessible within the contract.
- **internal:** Accessible within the contract and derived contracts.
- **external:** Only accessible from outside the contract.

Modifiers

- **view:** Does not modify state.
- **pure:** Does not read or modify state.
- **payable:** Can receive Ether.

Writing a Smart Contract

Events in Solidity

Emit logs for external consumers (e.g., frontend applications).

```
event EventName(parameters);
function triggerEvent() public {
    emit EventName(parameters);
}
```

```
event BalanceUpdated(address user, uint newBalance);
function updateBalance(uint _newBalance) public {
    balances[msg.sender] = _newBalance;
    emit BalanceUpdated(msg.sender, _newBalance);
}
```

Writing a Smart Contract

Error Handling in Solidity

Require: reverts the transaction if the condition is false.

- **Example:** `function withdraw(uint _amount) public {`

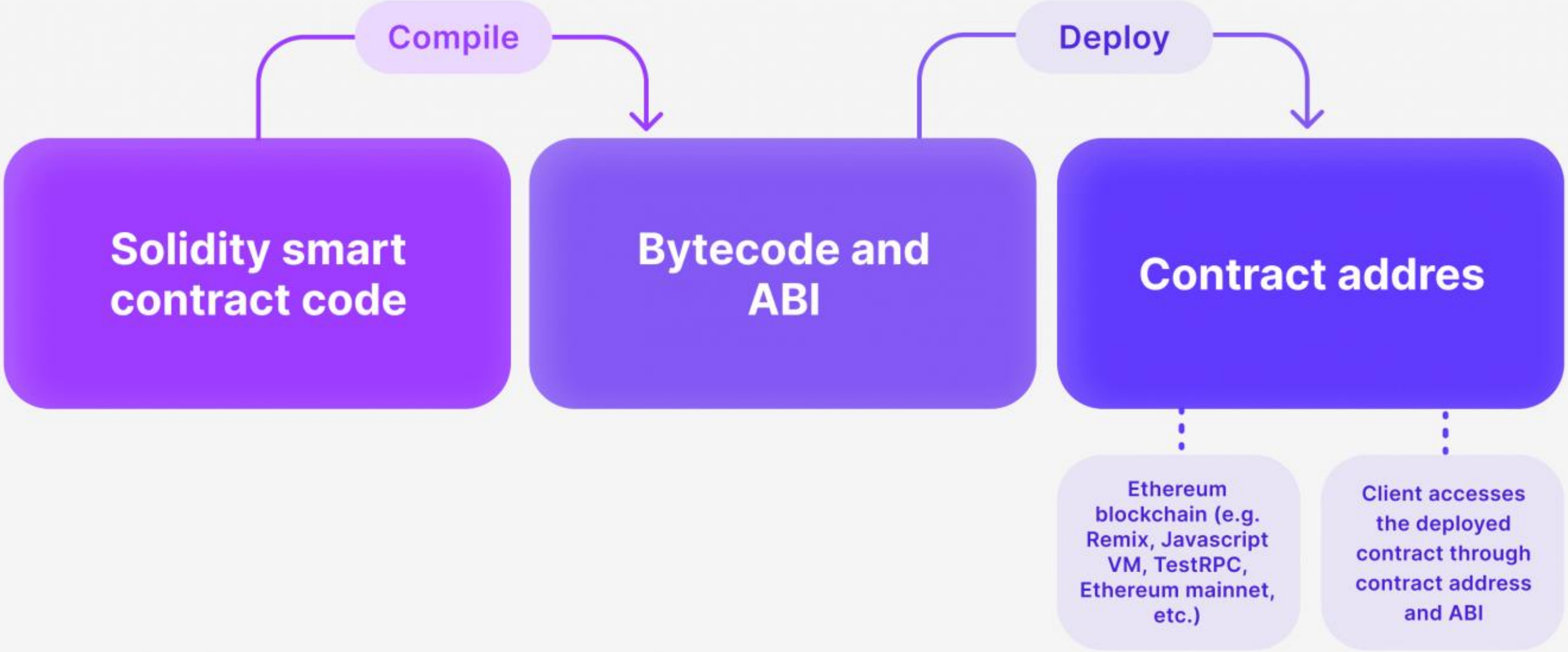
```
require(_amount <= balances[msg.sender], "Insufficient balance");
balances[msg.sender] -= _amount; };
```

Revert: Reverts the transaction with a custom error message

- **Example:** `function checkAge(uint _age) public pure {`

```
if(_age <= 18){
     revert(“must be at least 18 years old”)};
}
```

Compiling and Deploying



Compile

Deploy

Solidity smart contract code

Bytecode and ABI

Contract addres

Ethereum blockchain (e.g. Remix, Javascript VM, TestRPC, Ethereum mainnet, etc.)

Client accesses the deployed contract through contract address and ABI