Blockchain Programming

Seminar - Autumn semester 2021

Lecture 2. Ethereum

Prof. Dr. Claudio J. Tessone
Dr. Matija Piškorec

Blockchain and Distributed Ledger Technologies
University of Zurich

Index

1. Ethereum basics
2. Smart contracts on Ethereum network
3. Tokens and token standards on Ethereum network
4. Programming smart contracts in Solidity
5. Practical demonstration

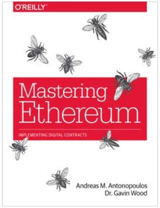
Bibliography

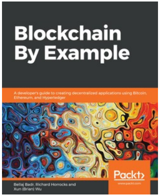
Basic reading for Ethereum:

1. ”Mastering Ethereum” by Antonopoulos A.M., Wood G. (O’Reilly Media)

Recommended readings for Ethereum:

1. ”Blockchain By Example” by Bellaj Badr, Richard Horrocks, Xun Wu (Packt Publishing)




Blockchain By Example
A developer's guide to creating decentralized applications using Bitcoin, Ethereum, and Hyperledger
Packt
Bellaj Badr, Richard Horrocks and Xun (Brian) Wu

Bibliography

Useful links for Ethereum:

- ”Mastering Ethereum”
  https://github.com/ethereumbook/ethereumbook
- ”Solidity documentation”
  https://solidity.readthedocs.io/
- ”Solidity by example”
  https://solidity-by-example.org/
- ”Ethereum whitepaper”
  https://ethereum.org/en/whitepaper/
- ”Go Ethereum documentation”
  https://geth.ethereum.org/docs/

Objectives

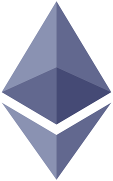
- Learn basic concepts of Ethereum
- Learn basic concepts of Ethereum smart contracts
- Learn how to setup your own Ethereum node and miner
- Learn how to write and deploy smart contracts on Ethereum

Ethereum basics

Ethereum history

Ethereum was conceived in 2013 by Vitalik Buterin [1] as a blockchain platform which would allow deployment of *decentralized applications*.

Ethereum network was launched in 2015 and is managed by a Swiss non-profit Ethereum Foundation.



[1] https://ethereum.org/en/whitepaper/

Accounts in Ethereum

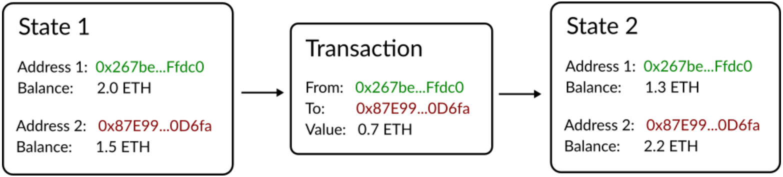
There are two kinds of users in Ethereum:

- Externally Owned Accounts (EOA) - controlled by users with appropriate private keys
- Smart contracts - controlled by the logic of the code

Smart contracts can only execute transactions in response to other transactions, which ultimatelly have to be initiated by an EOA account!

Account based model

Ethereum uses an *account based model* which keeps track of balances associated with addresses.


State 1

Address 1: 0x267be...Ffdc0
Balance: 2.0 ETH

Address 2: 0x87E99...0D6fa
Balance: 1.5 ETH

Transaction

From: 0x267be...Ffdc0
To: 0x87E99...0D6fa
Value: 0.7 ETH

State 2

Address 1: 0x267be...Ffdc0
Balance: 1.3 ETH

Address 2: 0x87E99...0D6fa
Balance: 2.2 ETH

Compared with Bitcoin’s UTXO model:

- ✗ scalability (it’s harder to parallelize transaction validation)
- ✗ privacy (it’s easier to track users)
- ✓ easier implementation of smart contracts

Ethereum addresses

Ethereum addresses are 20 least significant digits (40 hexadecimal characters) of the **Keccak-256** hashes of the public keys.

Public key (512 bits, 128 hex characters)

```
6e145ccef1033dea239875dd00dfb4fee6e3348b84985c92f103444683bae07b
83b5c38e5e2b0c8529d7fa3f64d46daa1ece2d9ac14cab9477d042c84c32ccd0
```

Keccak-256 hash of the public key (256 bits, 64 hex characters)

```
2a5bc342ed616b5ba5732269001d3f1ef827552ae1114027bd3ecf1f086ba0f9
```

https://github.com/ethereumbook/ethereumbook/blob/develop/04keys-addresses.asciidoc

Ethereum addresses

Ethereum addresses are usually prefixed with `0x` to signify hexadecimal encoding and are **case insensitive**! EIP-55 mixed-capitalization checksum ensures that errors in typing or reading are easily detectable (although not automatically corrected).

Ethereum address (160 bits, 40 hex characters)

```
0x001d3f1ef827552ae1114027bd3ecf1f086ba0f9
```

EIP-55 mixed-capitalization checksum:

```
0x001d3F1ef827552Ae1114027BD3ECF1f086bA0F9
```

https://github.com/ethereumbook/ethereumbook/blob/develop/04keys-addresses.asciidoc

Ether (ETH)

Ether (ETH) is the native currency of the Ethereum blockchain.

Minimum amount of ETH is 1 Wei, but there are other units as well:

- 10^{18} Wei = 1 ETH
- 10^{6} Szabo = 1 ETH
- 10^{3} Finney = 1 ETH

Structure of an Ethereum block 1/2

Each Ethereum block contains at least:

- `timestamp` - time when the block was mined
- `blockNumber` - length of the blockchain
- `baseFeePerGas` - minimum gas fee required for a transaction to be included in the block
- `difficulty` - effort required to mine the block
- ...

https://ethereum.org/en/developers/docs/blocks/

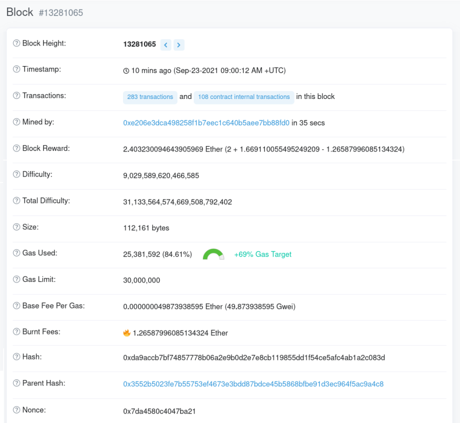
Structure of an Ethereum block 2/2

Each Ethereum block contains at least:

- ...
- `mixHash` - a unique identifier for the block
- `parentHash` - unique identifier of the previous block
- `transactions` - transactions included in the block
- `stateRoot` - entire state of the system (account balances, contract storage, contract code and account nonces)
- `nonce` - a hash that, when combined with the mixHash, proves that the block has gone through proof of work

https://ethereum.org/en/developers/docs/blocks/

Example of an Ethereum block



Block #13281065

Block Height:	**13281065** ‹ ›
Timestamp:	10 mins ago (Sep-23-2021 09:00:12 AM +UTC)
Transactions:	283 transactions and 108 contract internal transactions in this block
Mined by:	0xe206e3dca498258f1b7eec1c640b5aee7bb88fd0 in 35 secs
Block Reward:	2.403230094643905969 Ether (2 + 1.669110055495249209 - 1.26587996085134324)
Difficulty:	9,029,589,620,466,585
Total Difficulty:	31,133,564,574,669,508,792,402
Size:	112,161 bytes
Gas Used:	25,381,592 (84.61%) +69% Gas Target
Gas Limit:	30,000,000
Base Fee Per Gas:	0.000000049873938595 Ether (49.873938595 Gwei)
Burnt Fees:	🔥 1.26587996085134324 Ether
Hash:	0xda9accb7bf74857778b06a2e9b0d2e7e8cb119855dd1f54ce5afc4ab1a2c083d
Parent Hash:	0x3552b5023fe7b55753ef4673e3bdd87bdce45b5868bfbe91d3ec964f5ac9a4c8
Nonce:	0x7da4580c4047ba21

https://etherscan.io/block/13281065

Ethreum block size and gas

Gas

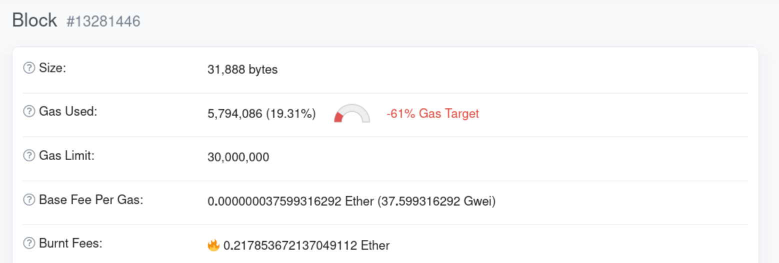
Measure of computational effort needed to execute specific operations on Ethereum network. It needs to be paid in ETH and its price is determined with the auction-based mechanism.

Ethereum block as not limited by size in *storage* but in *computational cost* needed for the execution of all transaction contained in them!

Each block has a target gas limit of 15M gas but this can increase or decrease based on the network demand, up to 30M gas. Typical storage size of a block is less than 50 kB.

Size of an Ethereum block

London Upgrade (August 2021) introduced a *base fee* as a minimum gas price needed for a transaction to be included in the block. These base fees are then *burned*!

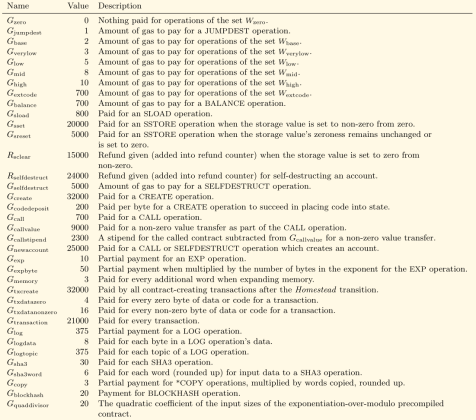


Block #13281446

ⓘ Size:	31,888 bytes
ⓘ Gas Used:	5,794,086 (19.31%) -61% Gas Target
ⓘ Gas Limit:	30,000,000
ⓘ Base Fee Per Gas:	0.000000037599316292 Ether (37.599316292 Gwei)
ⓘ Burnt Fees:	🔥 0.217853672137049112 Ether

https://etherscan.io/block/13281446

Ethereum gas fees



Name	Value	Description
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{jumpdest}	1	Amount of gas to pay for a JUMPDEST operation.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
G_{verylow}	3	Amount of gas to pay for operations of the set W_{verylow}.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
G_{extcode}	700	Amount of gas to pay for operations of the set W_{extcode}.
G_{balance}	700	Amount of gas to pay for a BALANCE operation.
G_{sload}	800	Paid for an SLOAD operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{\text{selfdestruct}}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{\text{selfdestruct}}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{\text{codedeposit}}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{\text{callvalue}}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{\text{callstipend}}$	2300	A stipend for the called contract subtracted from $G_{\text{callvalue}}$ for a non-zero value transfer.
$G_{\text{newaccount}}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
G_{expbyte}	50	Partial payment when multiplied by the number of bytes in the exponent for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
G_{txcreate}	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{\text{txdatazero}}$	4	Paid for every zero byte of data or code for a transaction.
$G_{\text{txdatanonzero}}$	16	Paid for every non-zero byte of data or code for a transaction.
$G_{\text{transaction}}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
G_{logdata}	8	Paid for each byte in a LOG operation's data.
G_{logtopic}	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
G_{sha3word}	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{\text{blockhash}}$	20	Payment for BLOCKHASH operation.
$G_{\text{quaddivisor}}$	20	The quadratic coefficient of the input sizes of the exponentiation-over-modulo precompiled contract.

https://ethereum.github.io/yellowpaper/paper.pdf

Ethereum vs Bitcoin comparison

	Bitcoin	Ethereum
Year of release	2009	2015
Consensus mechanism	PoW	PoW (transitions to PoS)
Blockchain size	∼355 GB	∼980 GB
Block count	∼701K	∼13M
Block production rate	∼10 min	∼10 sec
Block size	∼1 MB	∼20 kB
Transactions per second	∼4-7	∼15
Current supply	∼18M	∼117M
Max supply	21M	uncapped

Smart contracts on Ethereum network

Smart contracts on Ethereum network

Smart contracts

Smart contracts are **immutable**, **deterministic** computer programs that run in **isolated** fashion on blockchain. They are stored in a transactions and are executed by each node of the network.

- **Immutable** - once deployed they cannot change
- **Deterministic** - the outcome should be the same for everyone that runs it
- **Isolated** - should not affect execution of other programs

From Ethereum lecture by Danielle Dell'Aglio at DDIB2021.

A lifecycle of a smart contract

A lifecycle of a smart contract on Ethereum network [10]:

1. A smart contract developer writes a smart contract.
2. Any user can deploy a smart contract.
3. A validator writes the smart contract in a new block - gets *paid*!
4. Any user can invoke read-only functions or state variables from a contract - *free*!
5. Any user can invoke a method that requires writting to blockchain - pays *gas fee*!
6. A validator executes a function and writes the result in a new block - gets *paid*!

[10] From Ethereum lecture by Danielle Dell'Aglio at DDIB2021

Tokens and token standards on Ethereum network

Tokens vs cryptocurrencies

Cryptocurrency

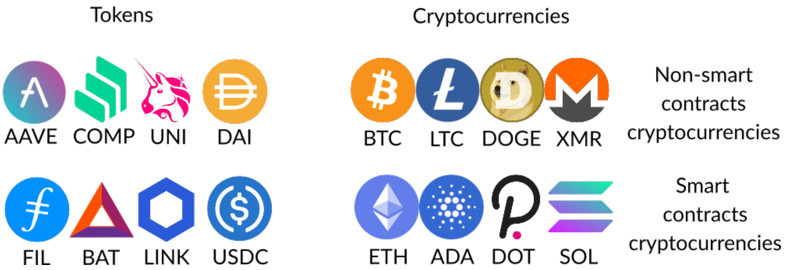
Electronic representation of a value, used as a payment for blockchain transactions (i.e. native currency such as Ether (ETH) in Ethereum network).

Token

Digital asset, issued by a stakeholder and giving certain rights to the holder. Managed by smart contracts!

Tokens vs cryptocurrencies

Not to be confused - Bitcoin (BTC) and Ether (ETH) are *cryptocurrenies*, while for example USD Coin (USDC) is a *token* on Ethereum network:-)



Tokens

AAVE COMP UNI DAI

FIL BAT LINK USDC

Cryptocurrencies

BTC LTC DOGE XMR

Non-smart contracts cryptocurrencies

ETH ADA DOT SOL

Smart contracts cryptocurrencies

Types of tokens

Utility tokens provide access to a service.

FIL BAT LINK CVC

Payment tokens are used as a means of payment.

BUSD USDT USDC DAI

Types of tokens

Asset tokens represent an ownership of an asset (e.g. debt, commodity, collectible).

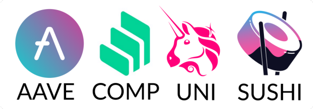
FTX stocks

PAXG

CryptoPunks

EtherRocks

Governance tokens give voting rights in digital or legal system.



Tokens and token standards on Ethereum network

Ethereum Improvement Proposals (EIP) EIP-20 [11] and EIP-721 [12] define token standards in terms of functions that smart contracts must implement in order to be used as exchangeable tokens:

ERC-20 defines standard functions a token contract must implement to allow DApps and wallets to exchange tokens accross multiple interfaces.

ERC-721 defines a standard for unique tradeable non-fungible tokens (NFTs) that may be tracked in standardized wallets and traded on exchanges as assets of value.

[11] https://eips.ethereum.org/EIPS/eip-20

[12] https://eips.ethereum.org/EIPS/eip-721

ERC-20 token standard 1/2

ERC-20 token requires following functions from a smart contract [13]:

- `totalSupply` - returns the token supply
- `balanceOf` - balance of an address
- `transfer` - transfer tokens to an address
- `transferFrom` - transfer from one address to another
- `approve` - allows spender to withdraw from account multiple times
- ...

[13] https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

ERC-20 token standard 2/2

Also, an ERC-20 smart contract should handle these events [14]:

- `Transfer` - triggers when tokens are transferred
- `Approval` - triggers when `approve` is called

OpenZeppelin provides a template for ERC-20 tokens [15].

[14] https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

[15] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/9b3710465583284b8c4c5d2245749246bb2e0094/contracts/token/ERC20/ERC20.sol

ERC-721 token standard

ERC-721 [16] tokens have globally unique `uint256 tokenId` and require similar functionalities as the ERC-20 token (e.g. `balanceOf` and `transferFrom`) with an addition of:

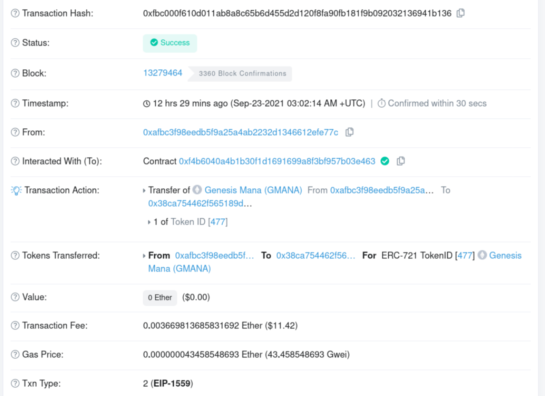
- `ownerOf` - identifies owner of a token (makes sense only if the token is non-fungible!)

OpenZeppelin provides a template for ERC-721 tokens [17].

[16] https://ethereum.org/en/developers/docs/standards/tokens/erc-721

[17] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol

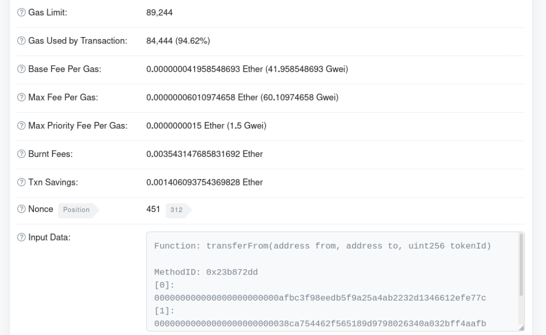
ERC-721 token transaction 1/2



ⓘ Transaction Hash:	0xfbc000f610d011ab8a8c65b6d455d2d120f8fa90fb181f9b092032136941b136
ⓘ Status:	✓ Success
ⓘ Block:	13279464 3360 Block Confirmations
ⓘ Timestamp:	12 hrs 29 mins ago (Sep-23-2021 03:02:14 AM +UTC) | Confirmed within 30 secs
ⓘ From:	0xafbc3f98eedb5f9a25a4ab2232d1346612efe77c
ⓘ Interacted With (To):	Contract 0xf4b6040a4b1b30f1d1691699a8f3bf957b03e463 ✓
Transaction Action:	▸ Transfer of Genesis Mana (GMANA) From 0xafbc3f98eedb5f9a25a… To 0x38ca754462f565189d… ▸ 1 of Token ID [477]
ⓘ Tokens Transferred:	▸ **From** 0xafbc3f98eedb5f… **To** 0x38ca754462f56… **For** ERC-721 TokenID [477] Genesis Mana (GMANA)
ⓘ Value:	0 Ether ($0.00)
ⓘ Transaction Fee:	0.003669813685831692 Ether ($11.42)
ⓘ Gas Price:	0.000000043458548693 Ether (43.458548693 Gwei)
ⓘ Txn Type:	2 (**EIP-1559**)

https://etherscan.io/tx/0xfbc000f610d011ab8a8c65b6d455d2d120f8fa90fb181f9b092032136941b136

ERC-721 token transaction 2/2



Gas Limit:	89,244
Gas Used by Transaction:	84,444 (94.62%)
Base Fee Per Gas:	0.000000041958548693 Ether (41.958548693 Gwei)
Max Fee Per Gas:	0.00000006010974658 Ether (60.10974658 Gwei)
Max Priority Fee Per Gas:	0.0000000015 Ether (1.5 Gwei)
Burnt Fees:	0.003543147685831692 Ether
Txn Savings:	0.001406093754369828 Ether
Nonce Position	451 312
Input Data:	see below

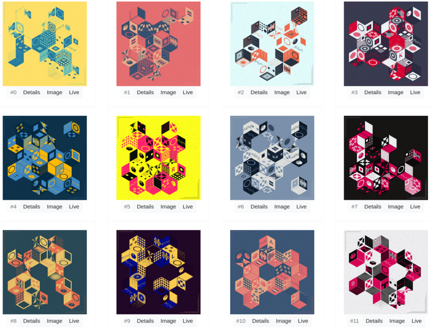
```
Function: transferFrom(address from, address to, uint256 tokenId)

MethodID: 0x23b872dd
[0]:
000000000000000000000000afbc3f98eedb5f9a25a4ab2232d1346612efe77c
[1]:
00000000000000000000000038ca754462f565189d9798026340a032bff4aafb
```

https://etherscan.io/tx/0xfbc000f610d011ab8a8c65b6d455d2d120f8fa90fb181f9b092032136941b136

NFT project example

All NFTs have globally unique `tokenID`, which in combination with owner's `address` allows for minting of unique generative art!



#0 Details Image Live

#1 Details Image Live

#2 Details Image Live

#3 Details Image Live

#4 Details Image Live

#5 Details Image Live

#6 Details Image Live

#7 Details Image Live

#8 Details Image Live

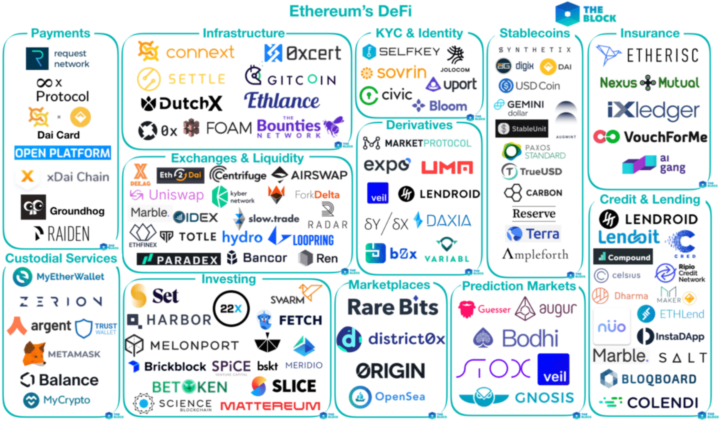
#9 Details Image Live

#10 Details Image Live

#11 Details Image Live

https://www.artblocks.io/project/74

Decentralized Finance (DeFi) on Ethereum



https://www.theblockcrypto.com

Programming smart contracts in Solidity

Programming smart contracts in Solidity

Solidity [22] is a high-level programming language for writing smart contracts on Ethereum network, inspired by Javascript, Java and C++, which compiles to Ethereum Virtual Machine (EVM) [23].

Other high-level languages for compiling to EVM:

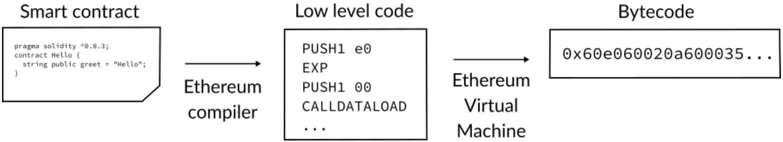
- Vyper (inspired by Pyhton)
  https://vyper.readthedocs.io/en/latest/
- Bamboo
  https://github.com/cornellblockchain/bamboo

[22] https://docs.soliditylang.org/en/v0.8.7/

[23] https://ethereum.org/en/developers/docs/evm/

Ethereum Virtual Machine (EVM)

Smart contracts get compiled to the bytecode which EVM can nativelly execute - it is stored on the Ethereum blockchain and becomes available for invocation.



Smart contract

```
pragma solidity ^0.8.3;
contract Hello {
  string public greet = "Hello";
}
```

Low level code

```
PUSH1 e0
EXP
PUSH1 00
CALLDATALOAD
...
```

Bytecode

```
0x60e060020a600035...
```

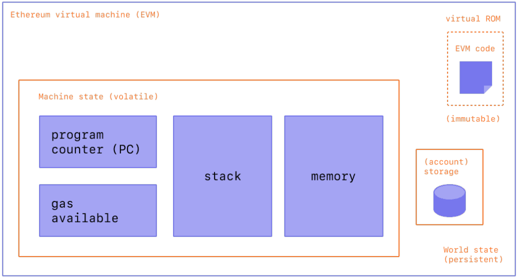
As an example, see contract [24] which powers the OpenSea [25] NFT marketplace.

[24] https://etherscan.io/address/0x7be8076f4ea4a4ad08075c2508e481d6c946d12b#code

[25] https://opensea.io/

Ethereum Virtual Machine (EVM)

Ethereum is a distributed *state machine* whose state consists of accounts, balances and all state variables, and which updates with each block by rules defined by the EVM.



https://ethereum.org/en/developers/docs/evm/

Programming smart contracts in Solidity

A simple hello world example in Solidity:

```
pragma solidity ^0.8.3;
contract HelloWorld {
    string public greet = "Hello World!";
}
```

`pragma` keyword defines minimum Solidity version which is needed to compile the contract.

`contract` keyword defines a contract which contains functions and variables, similar to classes in object-oriented programming languages.

https://solidity-by-example.org/hello-world/

Value types in Solidity 1/2

Value types are passed as value - they are always copied when passed as function arguments or in assignments.

```
// Integers: signed and unsigned with precision up to 8 bits
// Comparisons: <=, <, ==, !=, >=, >
// Bit operators: &, ¦, ^, ~, <<, >>
// Arithemtic operators: +, -, *, /, %, **
int, int8, int16, ..., int256
uint, uint8, uint16, ..., uint256

// Booleans: with possible values of true and false
// Logical operators - !, &&, ¦¦, ==, !=
bool
```

https://docs.soliditylang.org/en/v0.8.7/types.html

Value types in Solidity 2/2

```
// Fixed Point Numbers: signed and unsigned with varying precision of format MxN
// M: number of bits taken by the type (8-256 in 8 bit steps)
// N: number of decimal points (0-80)
fixed, fixed128x18
ufixed, ufixed128x18

// Address
address // 20 byte size
address payable // with additional members transfer and send

// Fixed-size byte arrays: sequence of bytes
bytes1, bytes2, ..., bytes32

// Enums
enum PriceLevel {Low, Medium, High}
```

https://docs.soliditylang.org/en/v0.8.7/types.html

Reference types in Solidity 1/2

Reference types can be modified through multiple different names. They have to defined by explicitly providing the data area where the type is stored:

- `memory` - lifetime limited to the function call
- `storage` - area where state variables are stored, lifetime limited to the lifetime of a contract
- `calldata` - non-modifiable, non-persistent area where function arguments are stored

https://docs.soliditylang.org/en/v0.8.7/types.html

Reference types in Solidity 2/2

```
// Arrays: with fixed or dynamic size
int[5], uint[10] // fixed-size arrays
int[], uint[] // dynamic arrays
int[][5], uint[][3] // fixed-size arrays of dynamic arrays

// Structs
struct Funder {
    address addr;
    uint amount;
}

// Dynamically sized bytes arrays
bytes
string

// Mappings: hash tables with key/value pairings, only data area allowed is storage!
// _KeyType: a built-in value type
// _ValueType: any type, including mappings, structs and arrays
mapping(_KeyType => _ValueType) _VariableName
```

https://docs.soliditylang.org/en/v0.8.7/types.html

Programming smart contracts in Solidity

A simple contract to increment and decrement the count in the contract store:

```
pragma solidity ^0.8.3;
contract Counter {
    uint public count;
    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }
    // Function to increment count by 1
    function inc() public {
        count += 1;
    }
    // Function to decrement count by 1
    function dec() public {
        count -= 1;
    }
}
```

https://solidity-by-example.org/first-app/

Visibility of variables and functions

Both variables and function can have following visibilities:

- **Internal** - default, only visible within a contract and contracts derived from it.
- **External** - accessible only from outside of the contract.
- **Public** - access from inside and outside of your code, getter function is automatically generated.
- **Private** - accessible only within the contract where it was defined.

Programming smart contracts in Solidity

An infinite-loop contract that eventually spends all gas available for transaction - the spent gas is **not** refunded to the user!

```
pragma solidity ^0.8.3;
contract Gas {
    uint public i = 0;
    // Using up all of the gas that you send causes your transaction to fail.
    // State changes are undone.
    // Gas spent are not refunded.
    function forever() public {
        // Here we run a loop until all of the gas are spent
        // and the transaction fails
        while (true) {
            i += 1;
        }
    }
}
```

https://solidity-by-example.org/gas/

Functions in Solidity

A general syntax for functions in Solidity is the following:

```
function functionName(
             [[bool|int|uint|string|address] parameterName] // parameters, optional
        )
        [internal|external|public|private] // visibility, optional
        [view|pure|payable] // type, optional
        [returns ([bool|int|uint|string|address])] // return value, optional
```

In addition to visibility, functions can be of following type:

- **View** - function does not change state.
- **Pure** - function does not change nor reads from state.
- **Payable** - function can receive Ether.

An interface for ERC-20 token

```
pragma solidity ^0.8.3;
interface IERC20 {
    function totalSupply() external view returns (uint);
    function balanceOf(address account) external view returns (uint);
    function transfer(address recipient, uint amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint amount) external returns (bool);
    function transferFrom(address sender,
                          address recipient,
                          uint amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}
```

https://solidity-by-example.org/app/erc20/

Interface for an ERC-721 token

```
pragma solidity ^0.8.0;
interface IERC721 {
    function balanceOf(address owner) external view returns (uint256 balance);
    function ownerOf(uint256 tokenId) external view returns (address owner);
    function safeTransferFrom(address from, address to, uint256 tokenId) external;
    function transferFrom(address from, address to, uint256 tokenId) external;
    function approve(address to, uint256 tokenId) external;
    function getApproved(uint256 tokenId) external view returns (address operator);
    function setApprovalForAll(address operator, bool _approved) external;
    function isApprovedForAll(address owner,
                              address operator) external view returns (bool);
    function safeTransferFrom(address from, address to, uint256 tokenId,
                              bytes calldata data) external;
    event Transfer(address indexed from, address indexed to,
                   uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved,
                   uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);
}
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/IERC721.sol

Practical demonstration

Practical demonstration

We will demonstrate how to install and run a node on our own private Ethereum network (UZHEthereum), and use Go Ethereum client (`geth`) [36], Metamask wallet [37] and Remix IDE [38] for executing transactions and deploying smart contracts to UZHEthereum:

- Install geth client
- Setup Metamask wallet
- Mine UZHETH
- Execute transactions on UZHEthereum network
- Deploy ERC-20 token to UZHEthereum network

[36] https://geth.ethereum.org/

[37] https://metamask.io/

[38] https://remix.ethereum.org/

Prof. Dr. Claudio J. Tessone
Dr. Matija Piškorec

Blockchain and Distributed Ledger Technologies
University of Zurich

✉ claudio.tessone@uzh.ch
piskorec@ifi.uzh.ch

↗ http://www.blockchain.uzh.ch