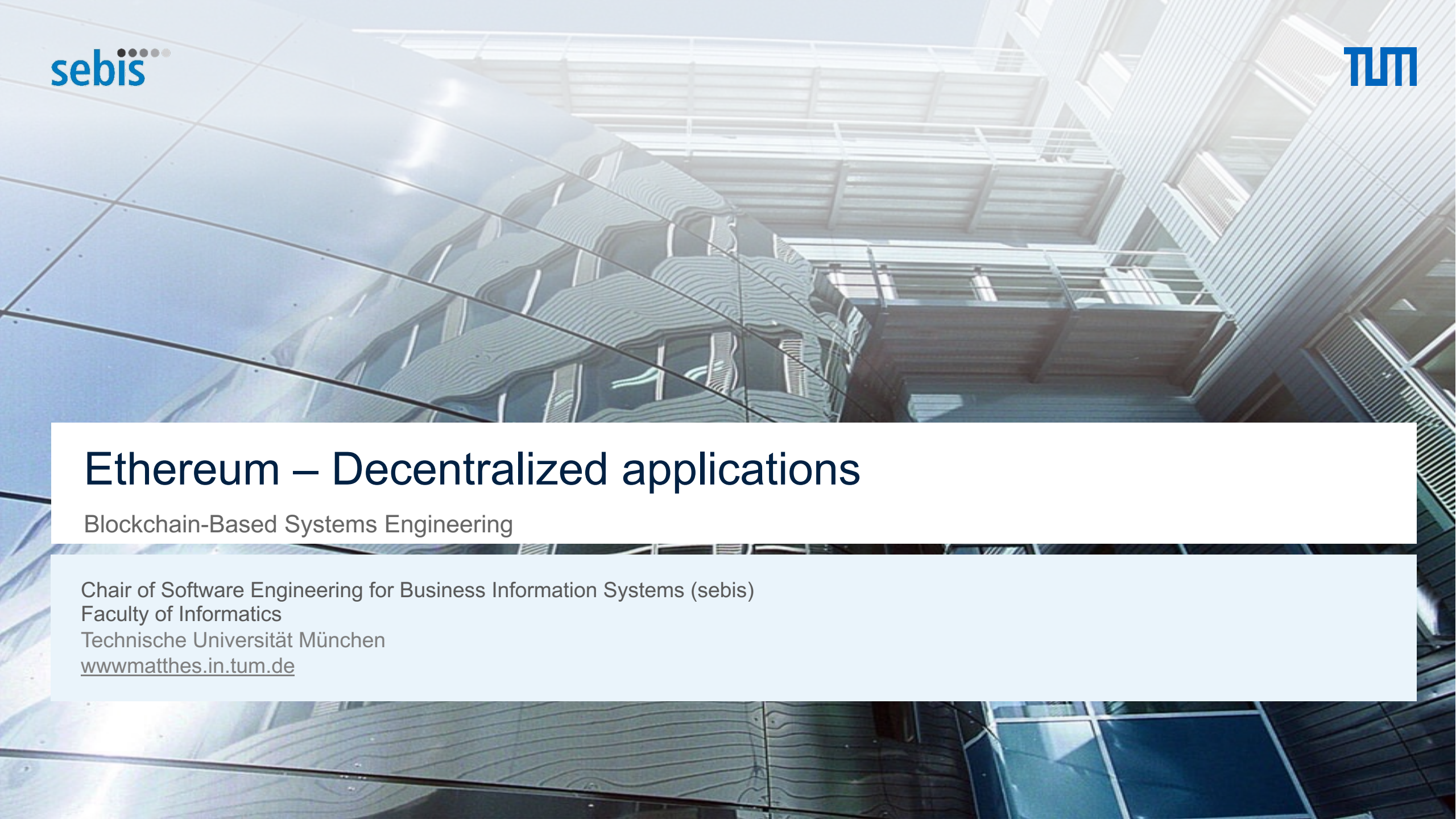


Ethereum – Decentralized applications

Blockchain-Based Systems Engineering

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

1. Introduction
   - Deploying a Smart Contract
   - Motivation
   - Definition
   - Benefits
   - Drawbacks

2. Architecture
   - Web-based systems
   - Private key management

3. Libraries and Frameworks
   - Web3
   - Development tools
     - Local
     - Cloud
   - Test networks

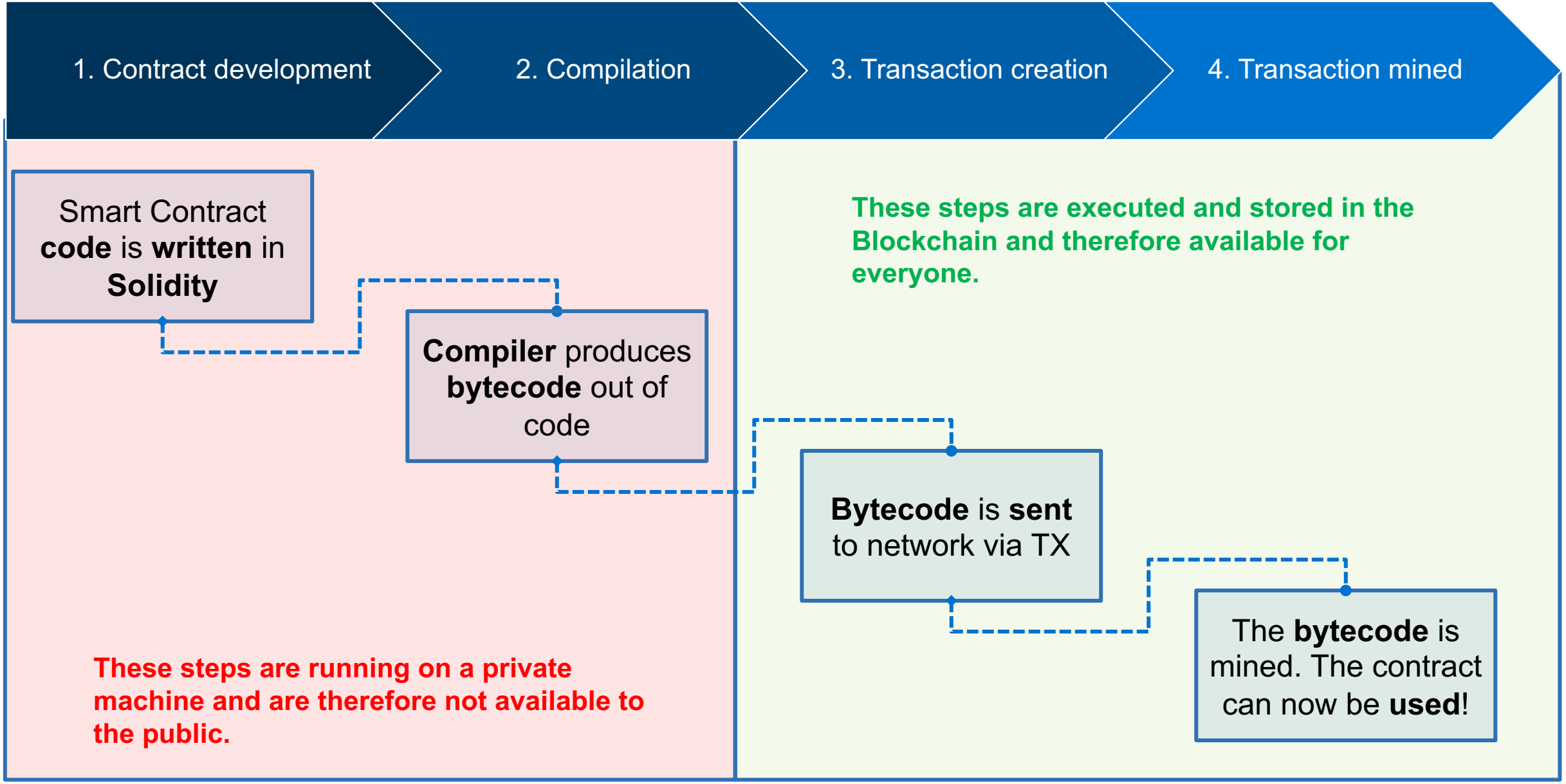
Deploying a Smart Contract

You should understand how Ethereum networks, transactions and the anatomy of smart contracts work before deploying smart contracts.

Requirements

- Bytecode of the smart contract (i.e., output of the compiler)
- A script to handle deployment steps
- A wallet with sufficient funds
- Access to an Ethereum node
  - Run your own node
  - Use a node service like Infura or Alchemy that provides instant access over HTTPS and WebSockets to the Ethereum network

Recap: How is a Smart Contract deployed on the Ethereum Blockchain?



Source code is typically not stored on the Blockchain, only byte code.

Without further analysis, the purpose of this Smart Contract is unclear.

Source code can be made publicly available.

Etherscan.io is the only service which verifies source codes and the respective byte code.

Contract Source Code Verified (Exact Match)

Contract Name:	SparkleCrowdsale	Optimization Enabled:	No
Compiler Text:	v0.4.25+commit.59dbf8f1	Runs (Optimiser):	200

Contract Source Code </> Copy Find Similiar Contracts

```
1  pragma solidity 0.4.25;
2
3  // File: openzeppelin-solidity/contracts/ownership/Ownable.sol
4
5  /**
6   * @title Ownable
7   * @dev The Ownable contract has an owner address, and provides basic authorization control
8   * functions, this simplifies the implementation of "user permissions".
9   */
10 contract Ownable {
11   address private _owner;
12
13   event OwnershipTransferred(
14     address indexed previousOwner,
15     address indexed newOwner
16   );
17
18   /**
19    * @dev The Ownable constructor sets the original `owner` of the contract to the sender
20    * account.
21    */
22   constructor() internal {
23     _owner = msg.sender;
24     emit OwnershipTransferred(address(0), _owner);
25   }
```

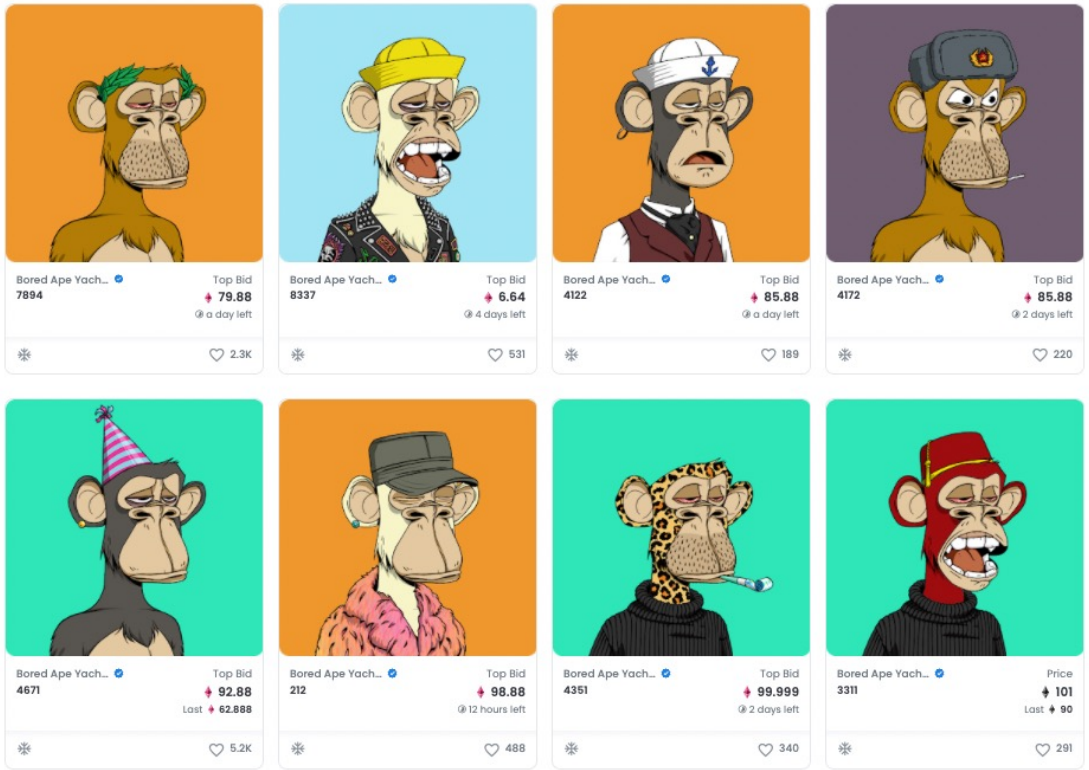
Motivation

- **Direct interaction** with smart contracts is **complicated**
- Smart contracts **don’t provide a graphical user interface** on their own
- **Programming knowledge** or **special tools** are **required** to make function calls

Screenshot of popular Ethereum wallet: MyEtherWallet.com

Motivation

- Building **UIs on top of smart contracts** to make them accessible to average users.
- The UI **abstracts the complicated function calls** and allows a user to interact with them just like with a regular (web) application.



Screenshot of a popular NFT Collection on the OpenSea dApp

Definition

- In this lecture, we consider a dApp as a decentralized application (in the terms of Blockchain) based on one or more smart contracts and accessible via a dedicated web-based user interface.
- In particular, the following properties must hold:
  - The core data records of the application must be stored on the blockchain
  - The functions that change the core data records must be executed on the Blockchain, i.e. via a smart contract

Benefits

The meaningfulness of implementing a distributed application is dependent on the concrete use case and / or the problem that is being solved.

Some general properties of Ethereum-based dApps:

Trust
The source code of any verified smart contract can be checked by anyone.

Payment
Payment is implemented by default since anyone can send / receive Ether.

Accounts
dApps can be build on top of Ethereum's account system, so there is no need to implement an additional user account management system.

Storage
dApps can leverage the Blockchain as common (expensive) data storage.

Drawbacks

Decentralized applications have also some intrinsic disadvantages:

Costs
Any state change and computation costs money. For that reason, only mission-critical data and functionality should leverage the Blockchain.

Time
The current block time of Ethereum is around 14 seconds, i.e., it takes at least 14 seconds from the function call to the definite result of it.

Availability
In theory, availability is one of the key advantages of dApps. However, in high transaction scenarios (e.g. the release of crypto kitties) it is possible that the network throttles and is not able to process function calls anymore.

Transparency
Without third party services, it is impossible to access and verify a Smart Contract source code.

Outline

1. Introduction
   - Deploying a Smart Contract
   - Motivation
   - Definition
   - Benefits
   - Drawbacks

2. Architecture
   - Web-based systems
   - Private key management

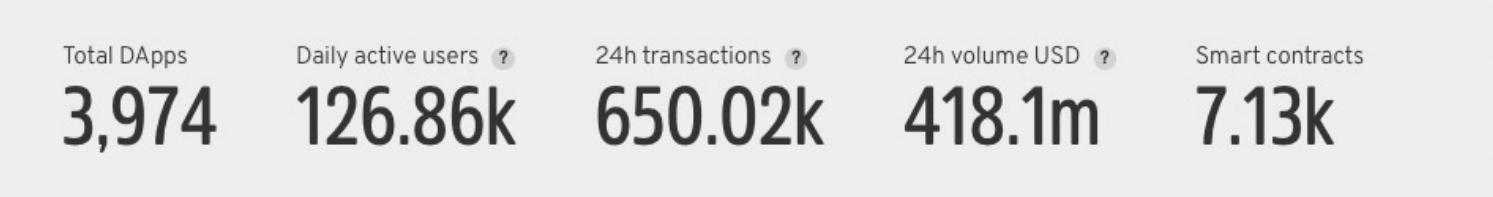
3. Libraries and Frameworks
   - Web3
   - Development tools
     - Local
     - Cloud
   - Test networks

Web-based dApps

Currently available decentralized applications are usually web-based and run in the browser.

State of the ĐApps is a curated and community-driven directory of Ethereum-based decentralized applications. The directory is one of the largest and most relevant of its kind and in some talks referenced by Vitalik Buterin himself [1].

As of April 2022, the directory contains 3974 dApps (2021: 3040, 2020: 2825 dApss, 2019: 2667 dApps).

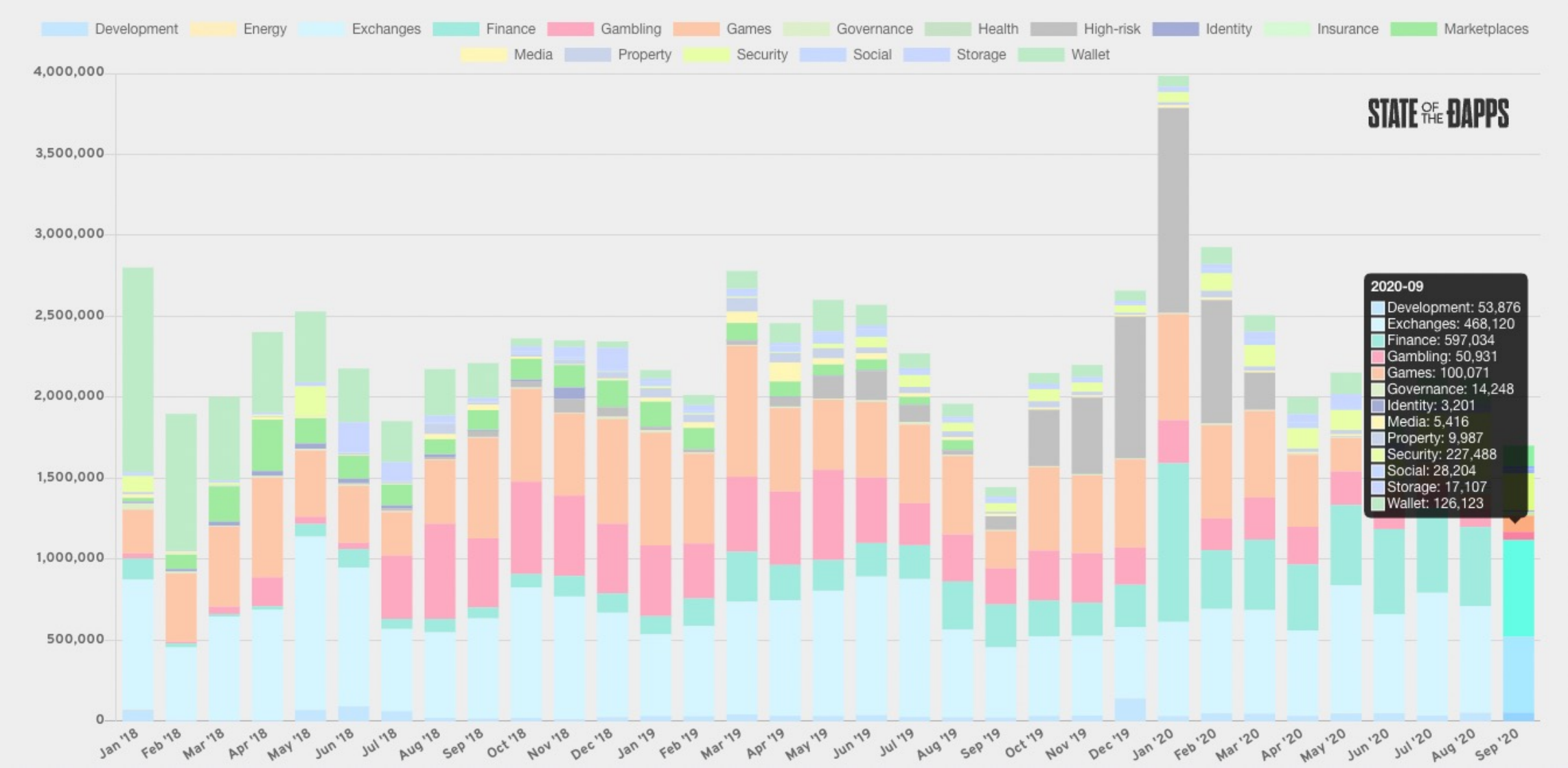


Examples of current dApps

- ICOs – Token systems that can be used to sell securities for Ether
- Games – Usually collectibles are issued and exchanged for Ether
- Gambling – Betting and lottery application
- Exchanges – Decentralized token and ether exchanges

https://www.stateofthedapps.com/

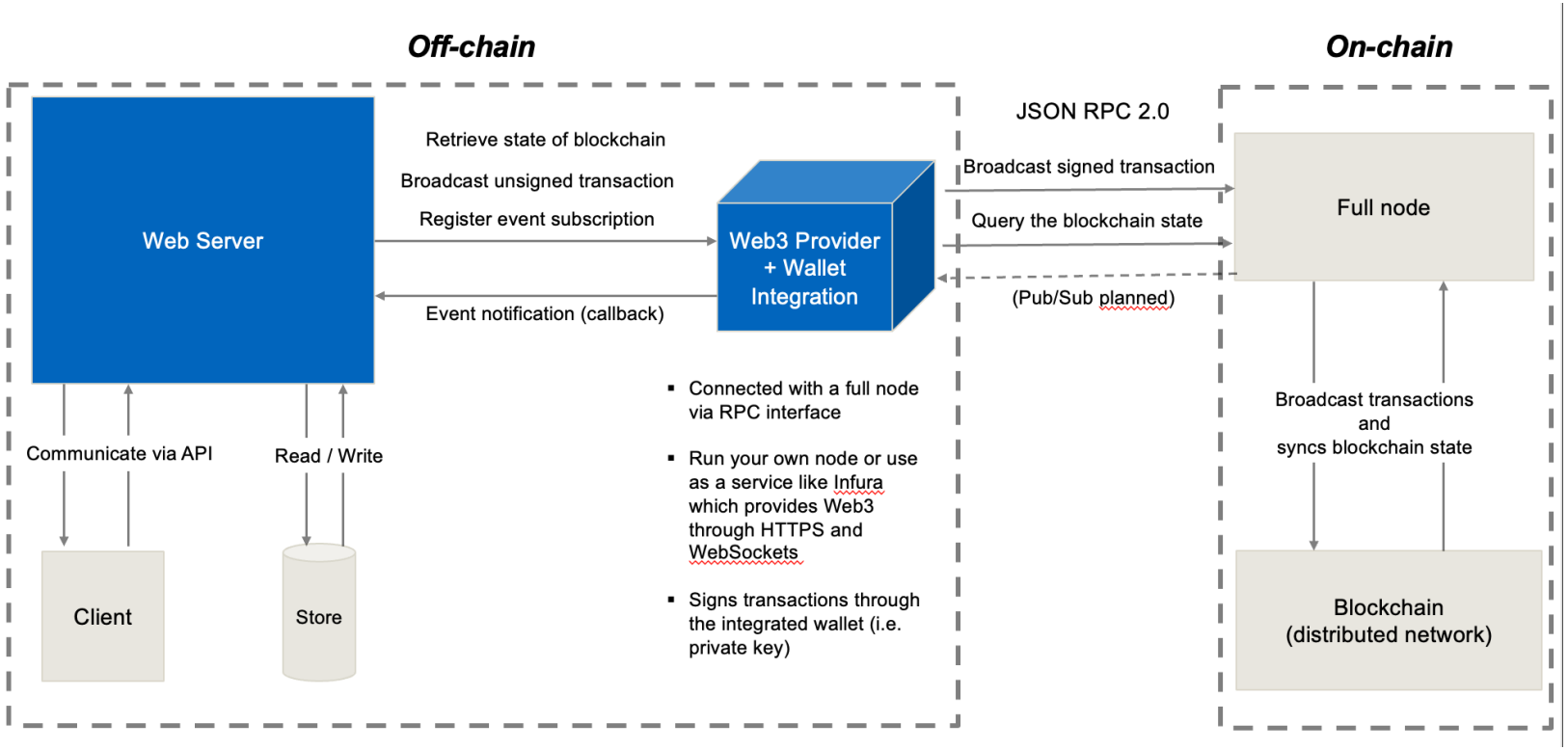
Current usage of dApps



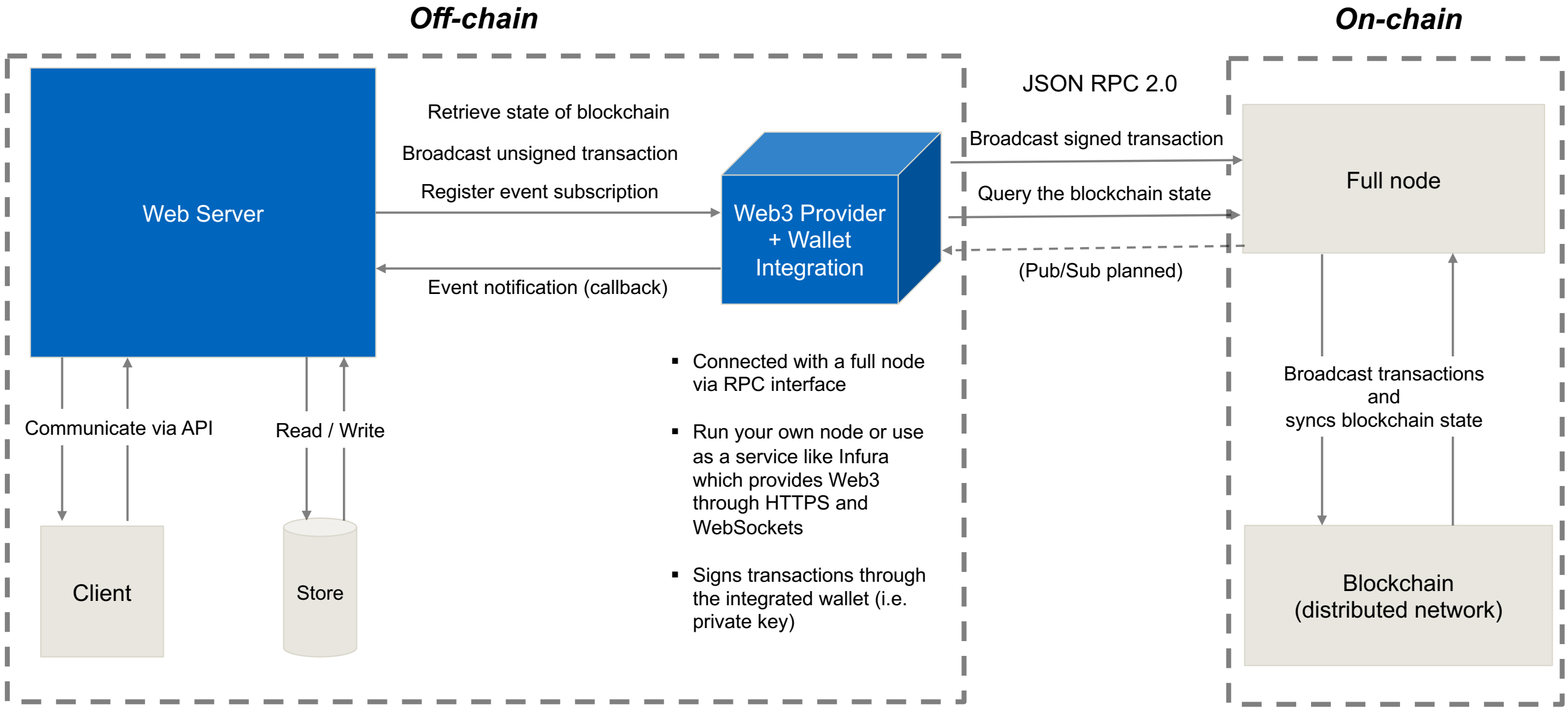
Architecture of a web-based Ethereum dApp – Server-side blockchain interaction

Example Use Case

- A dashboard to display statistics about the state of the Ethereum blockchain (e.g., a blockchain explorer)
  - Last block number
  - Miner of the last block
  - Number of transactions waiting in the mempool
  - …
- User doesn't directly interact with the blockchain (e.g., doesn't sign a transaction), thus user wallet info is not needed
- The website interacts with the blockchain on the server-side
  - Either runs own full node or uses a node service like Infura



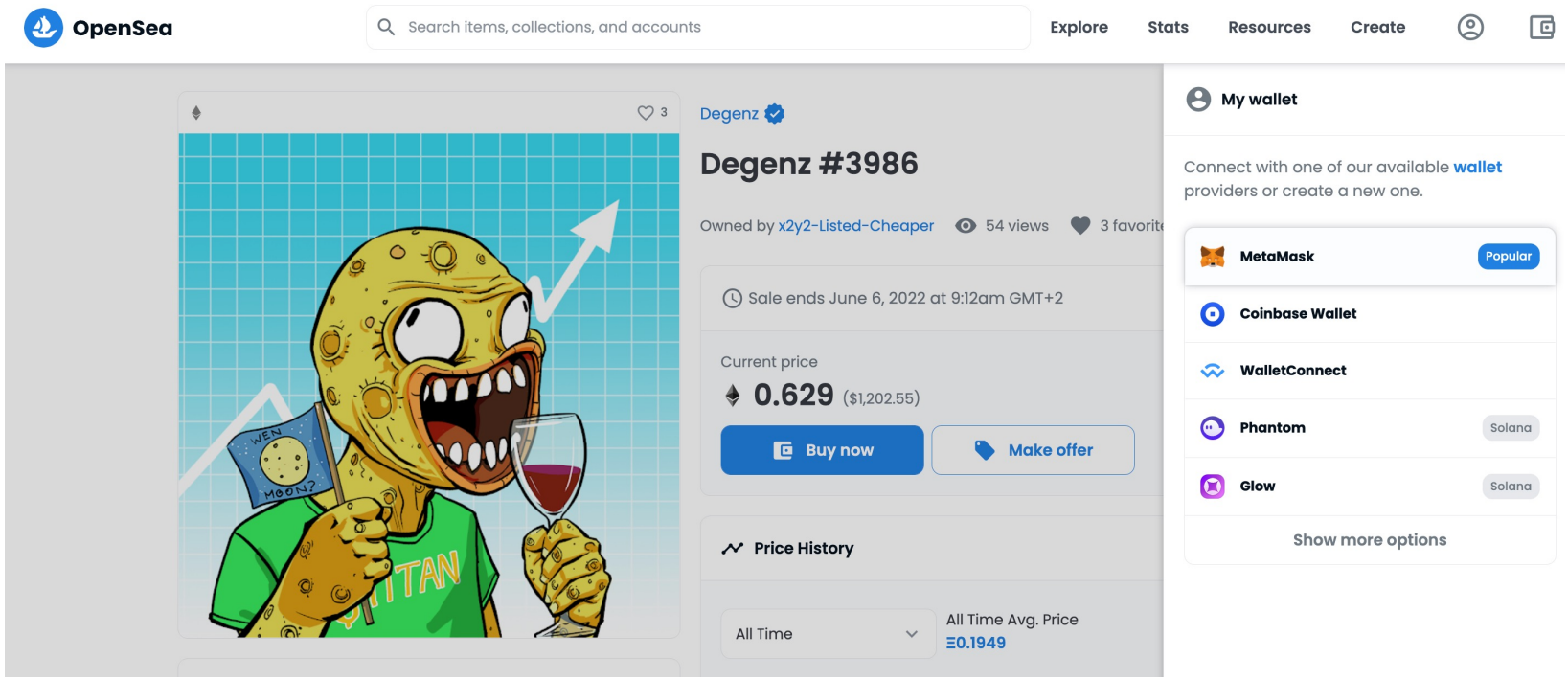
Architecture of a web-based Ethereum dApp – Server-side blockchain interaction



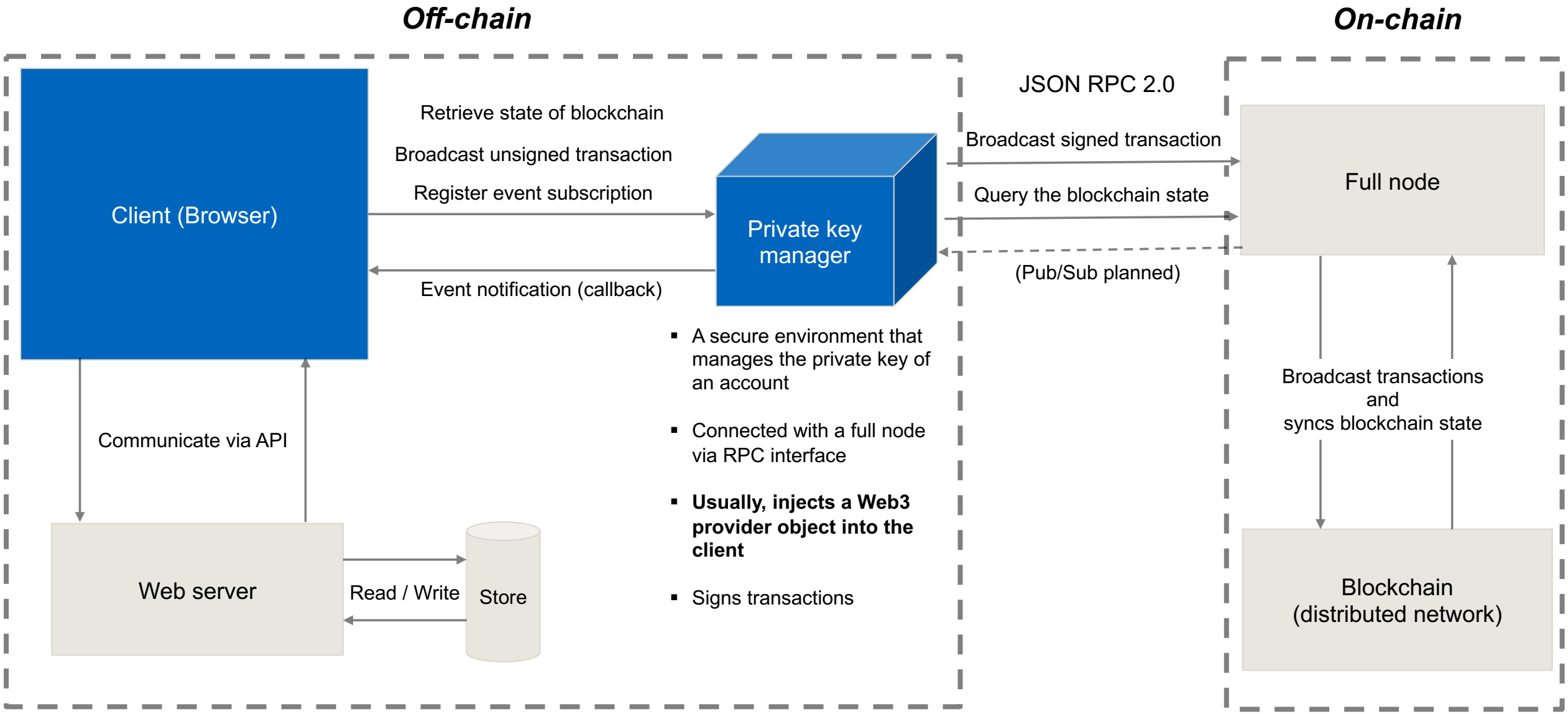
Architecture of a web-based Ethereum dApp – Client-side blockchain interaction

Example Use Case

- Imagine a website (dApp) where users can buy NFTs (e.g., an NFT marketplace like OpenSea)
- To complete a purchase of an NFT, the user has to first connect her/his wallet
  - A wallet connection is required to sign a transaction
- The website interacts with the blockchain on the client-side (i.e., through the browser)
  - Web3 must be injected into the client browser

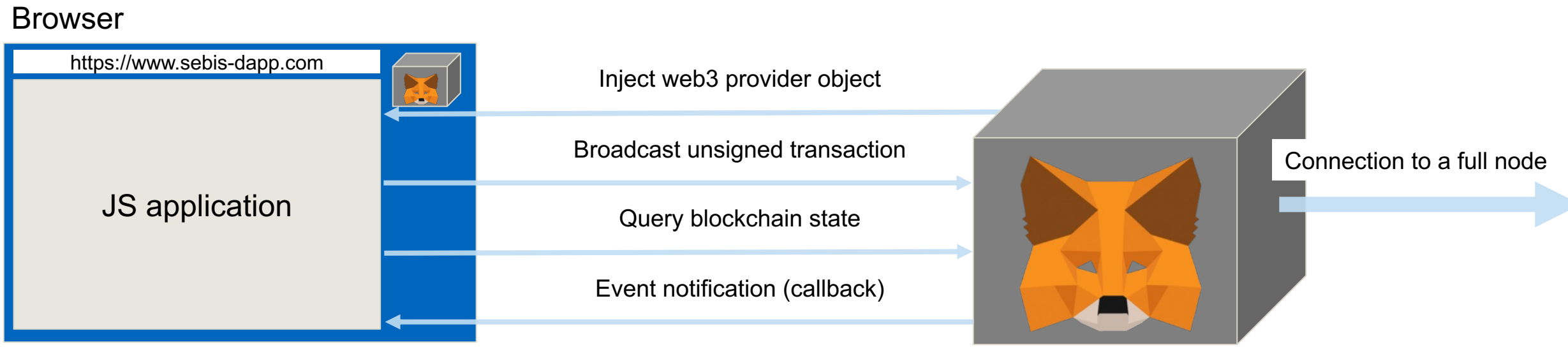


Architecture of a web-based Ethereum dApp – Client-side blockchain interaction



MetaMask – A popular private key manager and even more

Browser Plugin

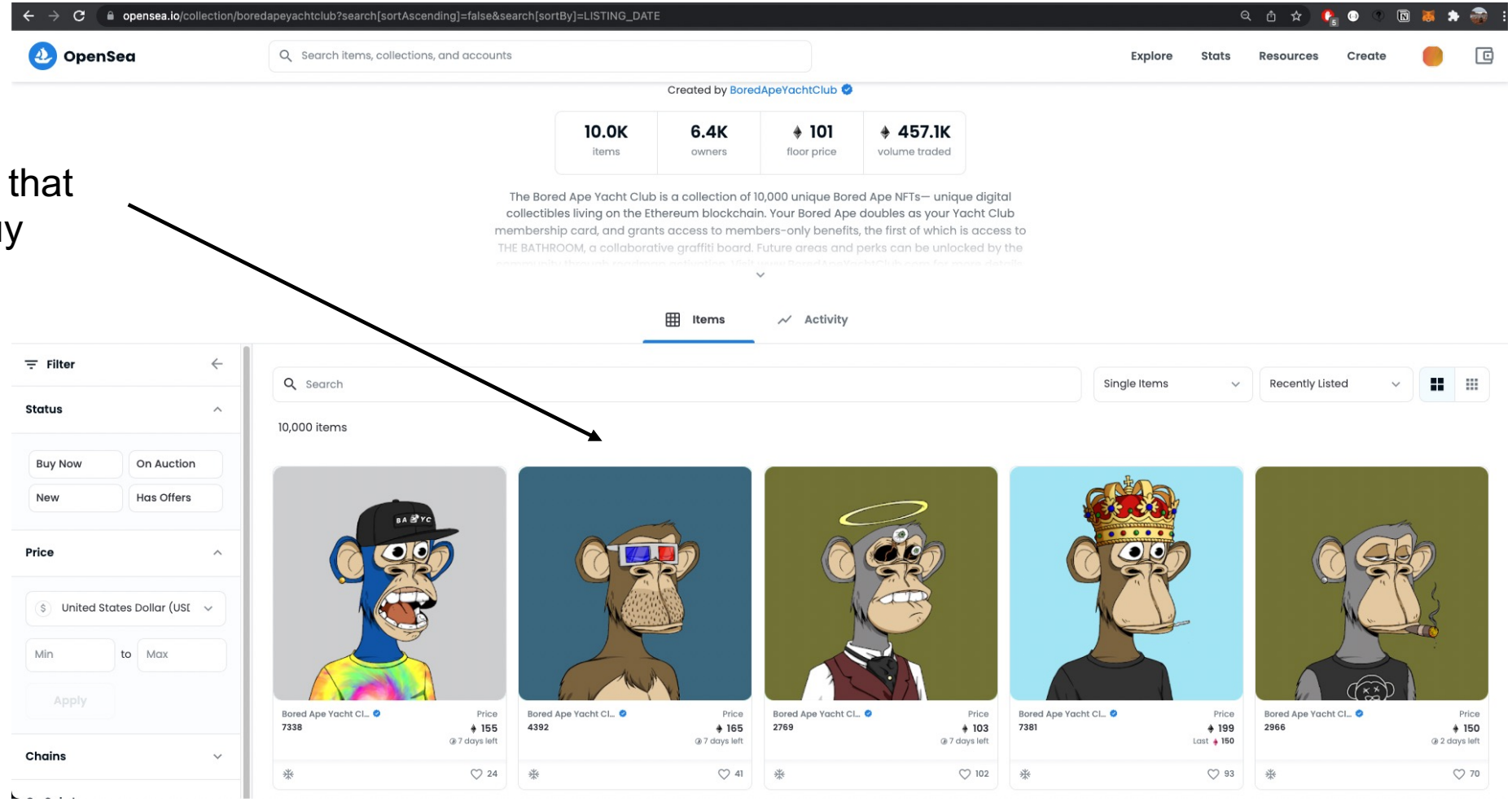


- A browser with MetaMask (🦊) plugin installed
- The plugin is required to manage the private keys of an Ethereum account
- It signs transactions and establishes a connection to a full node

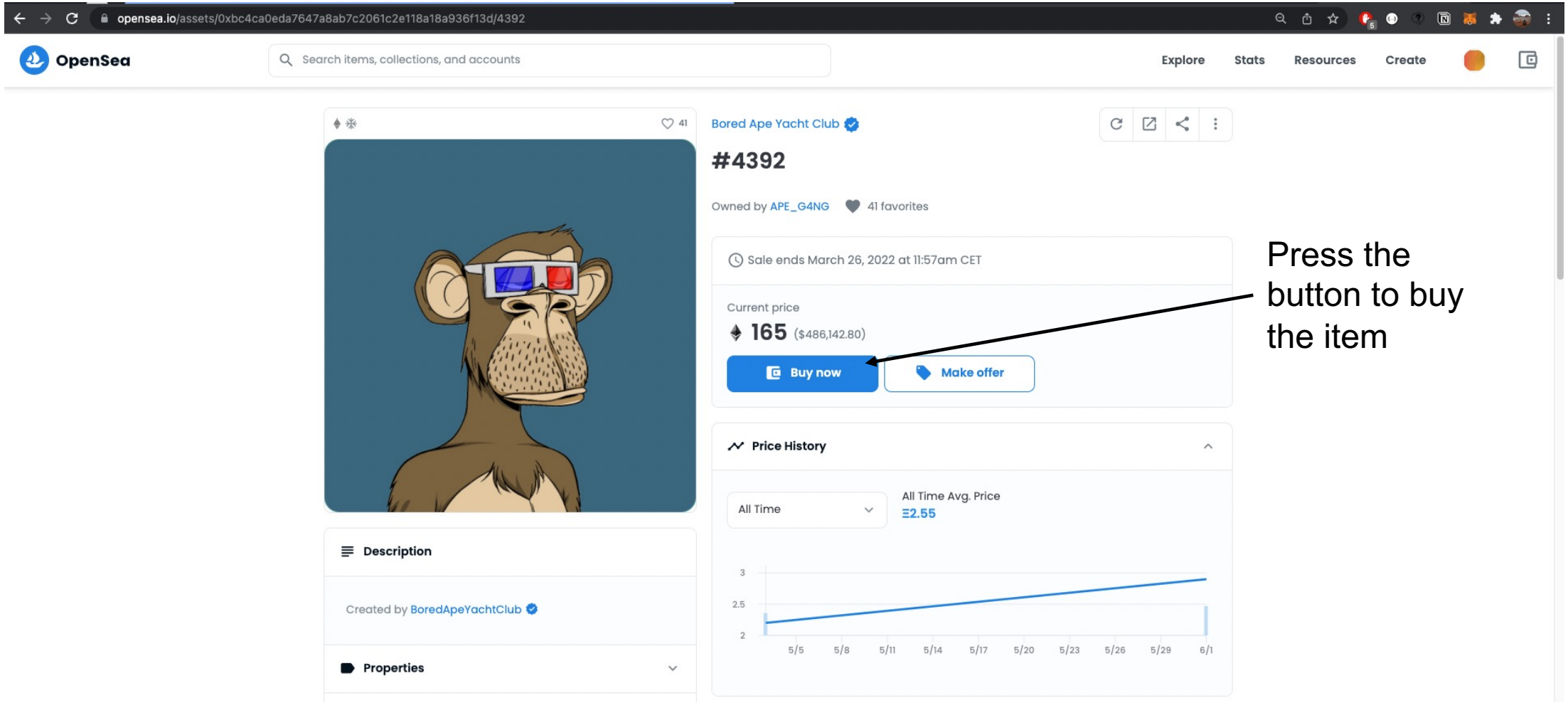
Read more about MetaMask's Ethereum provider API: https://docs.metamask.io/guide/ethereum-provider.html#table-of-contents

Example: OpenSea

Select an item that you want to buy

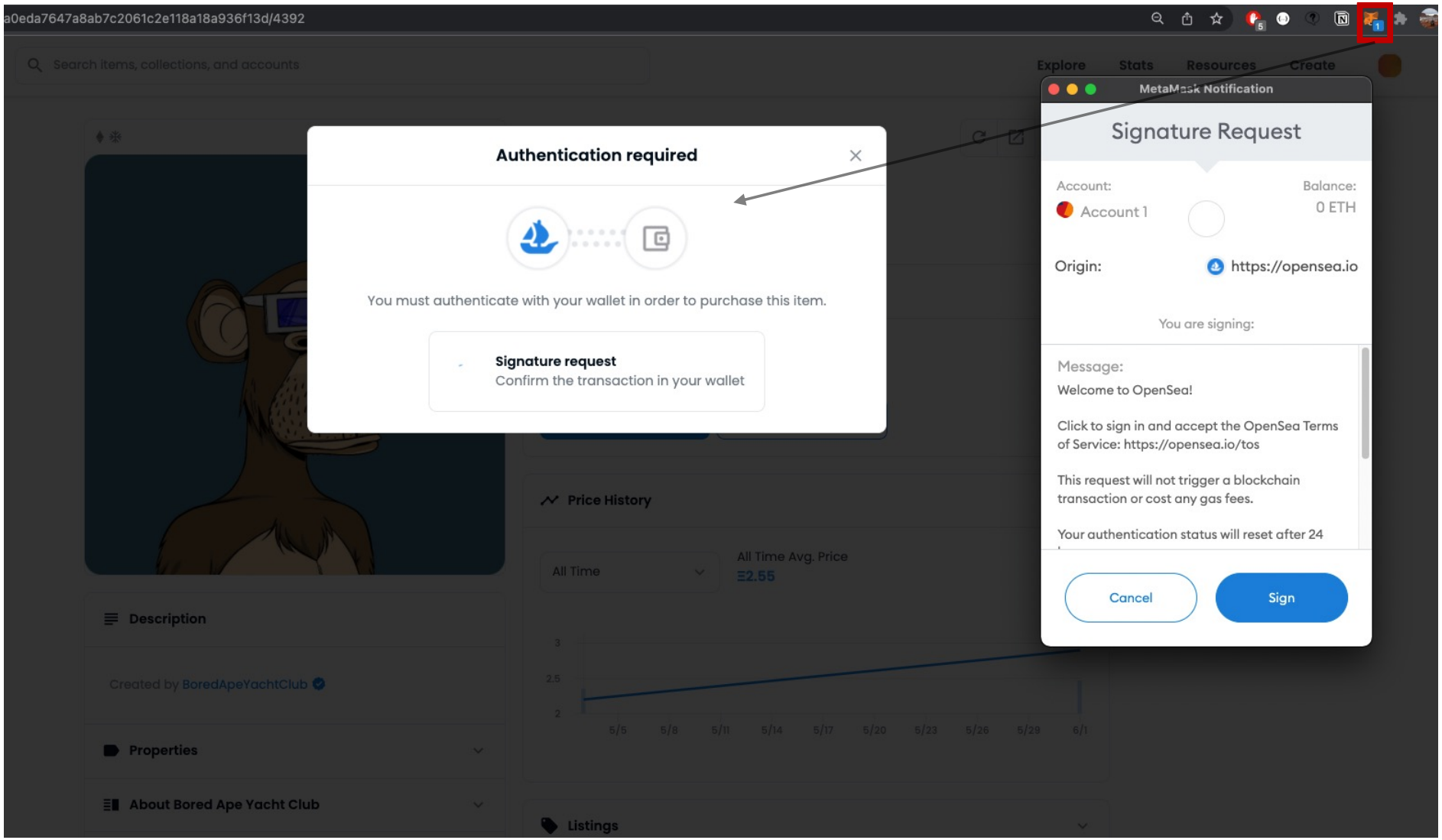


Example: OpenSea



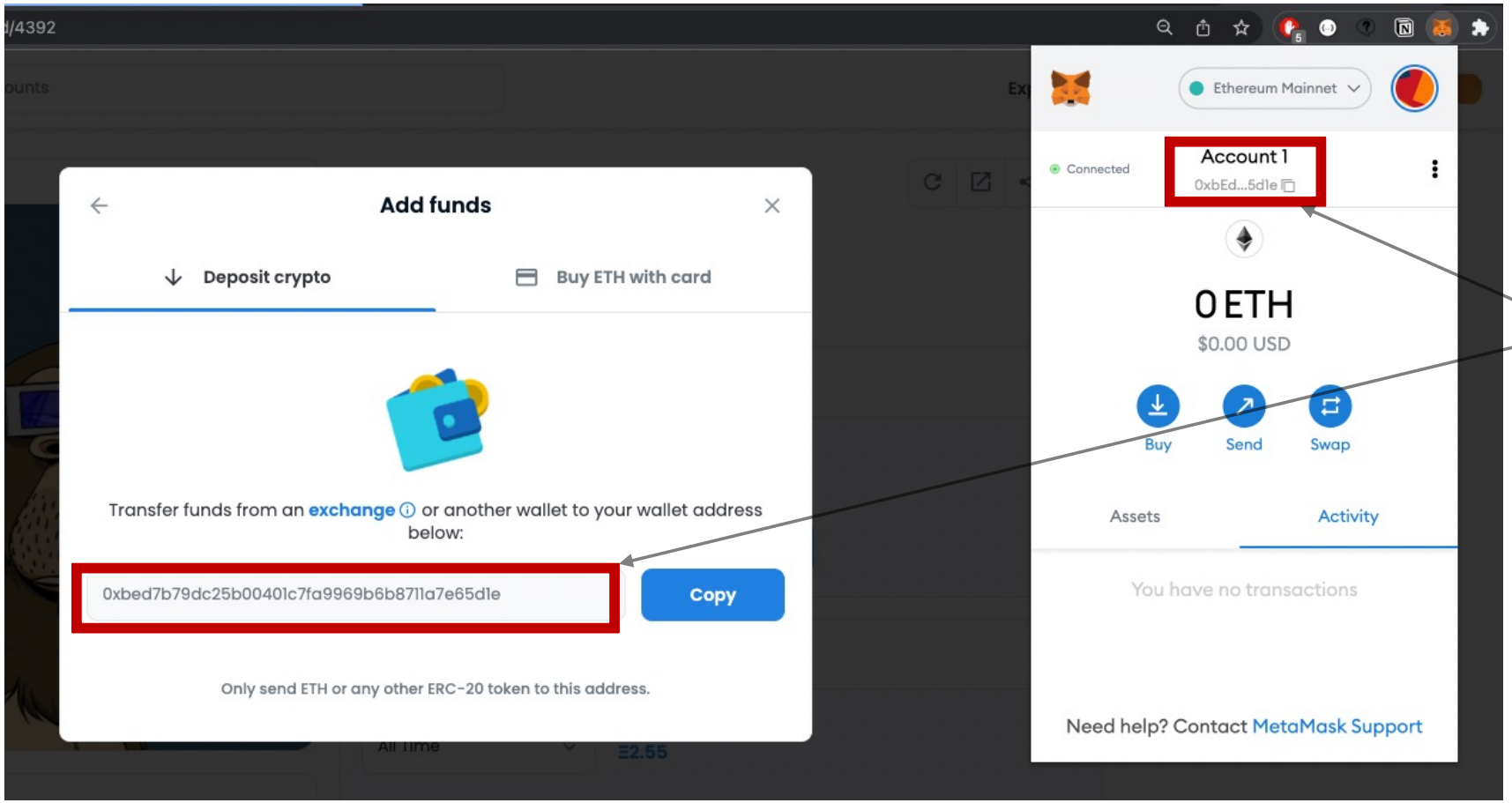
Example: OpenSea (cont.)

MetaMask as private key manager



- MetaMask injects a Web3 provider into the OpenSea application
- OpenSea can access your account through the injected Web3 provider

Example: OpenSea (cont.)



- Since there is no ETH available at the address, OpenSea asks you to transfer funds from an exchange to your address

Outline

1. Introduction
   - Deploying a Smart Contract
   - Motivation
   - Definition
   - Benefits
   - Drawbacks

2. Architecture
   - Web-based systems
   - Private key management

3. Libraries and Frameworks
   - Web3
   - Development tools
     - Local
     - Cloud
   - Test networks

web3.js

web3.js is the **official Ethereum JavaScript API** that provides a **wrapper** for the **JSON RPC interface** to interact directly with the blockchain.

- Can be used on **server-side** and **client-side**
  - On server-side, Web3 is injected by a full node or a node service like Infura
  - On client-side, Web3 is injected by a private key manager like MetaMask
- The Framework is available as a npm module (`npm install web3`)
- Provides **methods** to deploy and **interact** with **smart contracts**
- Provides methods to sign and send transactions
- Uses callback functions and promises for asynchronous events
- Widely used framework to interact with the Ethereum Blockchain

To read the documentation of web3.js vist: https://web3js.readthedocs.io/en/v1.7.3/

Web3 implementation for other languages

Web3 is available for other languages besides JavaScript, too. However, not all of those implementations are officially maintained by the Ethereum foundation.

Python (official)
Web3.py - https://github.com/ethereum/web3.py

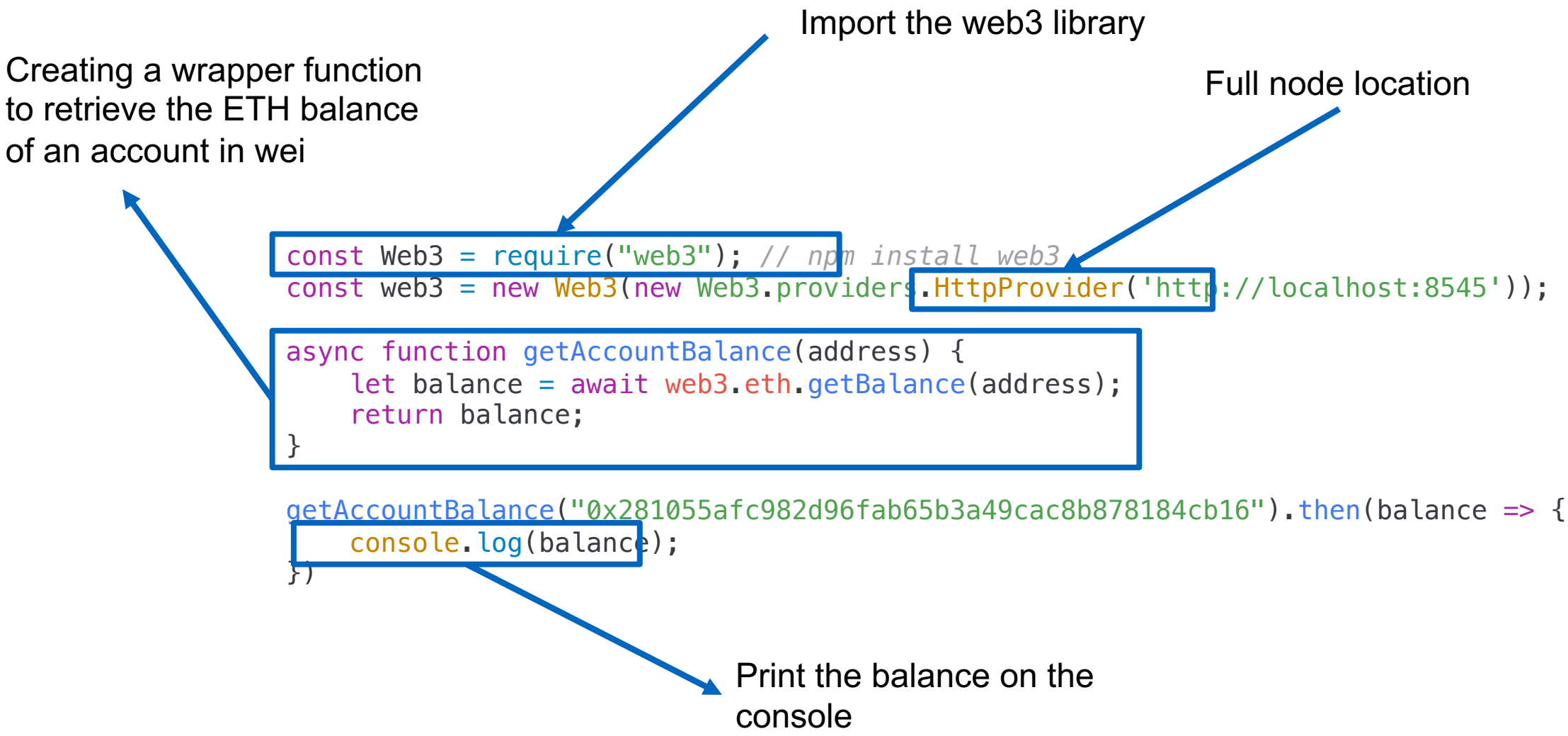
Haskell
Hs-web3 - https://github.com/airalab/hs-web3

Java
Web3j - https://github.com/web3j/web3j

Scala
Web3-scala - https://github.com/mslinn/web3j-scala

Purescript
purescript-web3 - https://github.com/f-o-a-m/purescript-web3

Example: Getting the balance of a specific account



Example: Sending Ether to another account

```
var gasPrice = 2;//or get with web3.eth.gasPrice
var gasLimit = 3000000;

var rawTransaction = {
"from": addr,
"nonce": web3.toHex(nonce), //or use web3.eth.getTransactionCount(address, 'pending')
"gasPrice": web3.toHex(gasPrice * 1e9),
"gasLimit": web3.toHex(gasLimit),
"to": toAddress,
"value": amountToSend ,
"chainId": 4 // Id of the blockchain (1=main net)
};

var privKey = new Buffer('0x....', 'hex');
var tx = new Tx(rawTransaction);

tx.sign(privKey);
var serializedTx = tx.serialize();

web3.eth.sendRawTransaction('0x' + serializedTx.toString('hex'), function(err, hash) {
        if (!err) {
                console.log('Txn Sent and hash is '+hash);
        }
        else {
                console.error(err);
        }
});
```

Deployment lifecycle

- Compilation of the Solidity source code
- Generate an **Application Binary Interface** (ABI) in JSON that can be used by other applications (e.g., dApps) to interact with the contract

HelloWorld.sol

```
contract HelloWorld {
        function greet() public returns(bytes32) {
                return "Hello World!";
        }
}
```

ABI

```
[
  {
    "constant": false,
    "inputs": [],
    "name": "greet",
    "outputs": [
      {
        "name": "",
        "type": "bytes32"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
```

Bytecode

```
608060405234801560f57600080fd5b5060ba8061001e6000396
000f3fe6080604052600436106039576000357c01000000000000
0000000000000000000000000000000000000000000090048063c
fae321714603e575b600080fd5b348015604957600080fd5b50605
06066565b604051808281526020019150506040518091039 0f35b
60007f48656c6c6f20576f726c64210000000000000000000000
0000000000000000090509056fea165627a7a72305820a4741ad95
e951c73637b4a34111570c749d47ab815d9838dd50c29ed524020
560029
```

Deployment lifecycle (cont.)

```
[
  {
    "constant": false,
    "inputs": [],
    "name": "greet",
    "outputs": [
      {
        "name": "",
        "type": "bytes32"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
```

ABI

```
608060405234801560f57600080fd5b5060ba8061001e6000396
000f3fe6080604052600436106039576000357c010000000000000
0000000000000000000000000000000000000000000090048063c
fae321714603e575b600080fd5b34801560495760080fd5b50605
06066565b604051808281526020019150506040518091039035b
60007f48656c6c6f20576f726c6421000000000000000000000000
0000000000000000090509056fea165627a7a72305820a4741ad95
e951c73637b4a34111570c749d47ab815d9838dd50c29ed524020
560029
```

Bytecode

dApp client or server via Web3

Uses ABI

Uses contract address

Send TX to network

Mined contract

0xb48060405234801560f576

Recap: Solidity Events

```
contract ERC20 {
    //...
    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

- Convenient way to listen for smart contract function calls and state changes without scanning the whole blockchain.
- Mostly used to trigger callback function in dApp event listeners.
- Fast way for third party (Web3) applications to check what happened in a certain block.

Events and logs are expected to change in the future. Currently, there are discussions in the Ethereum community to remove them from the persistent blockchain state.

Example: Subscribing to future ”Transfer” events in JavaScript

Accessing contract events

- Events can be accessed via the global events object
- Each event emitter can emit 3 types of events:
  - data: Fires on each incoming event with the event object as argument.
  - changed: Fires on each event which was removed from the blockchain.[1]
  - error: Fires when an error in the subscription occurs.

```
MyContract.events.Transfer({fromBlock: 0},
                function(error, event){ console.log(error) })
                  .on('data', (log) => {
                    console.log(`Data: ${log}`)
                  })
                  .on('changed', (log) => {
                    console.log(`Changed: ${log}`)
                  })
                  .on('error', (log) => {
                    console.log(`Error: ${log}`)
                  });
```

[1] *Removed from Blockchain means that the event was created inside a block which was later orphaned. See details here: https://ethereum.stackexchange.com/questions/7905/ethereum-event-log-removed-field*

Working with smart contracts in JavaScript

Create contract object in JavaScript that references an existing contract on the blockchain.

- Pass ABI object as first constructor parameter
- Pass contract address as second parameter

```
const MyContract = new web3.eth.Contract([
  {
    "constant": false,
    "inputs": [],
    "name": "greet",
    "outputs": [
      {
        "name": "",
        "type": "bytes32"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
, "0xC04229E8Edd4402D030cf81efF3e25df0E84BAA1");
```

Call contract method

- Methods defined in the ABI can be accessed via the global methods object

```
MyContract.methods.greet().send({from:
'0xcc8d743....97278fe497ee90...'},
 (error, txHash) => {
        if(err) {
        // Handle error here
        }
        else {
        // No error occured
        }
});
```

Overview development tools

Local environment

IDE / Code editor

Atom

Sublime

Visual Studio

Artifacts

- Solidity source code files
- Test cases

Build management

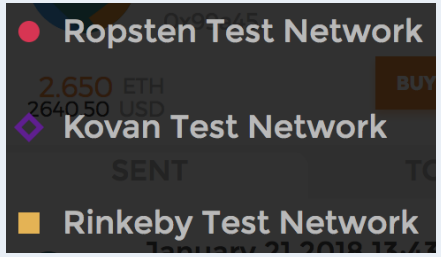
Artifacts

- Compiled .sol files / Bytecode
- Contract ABI / JSON

Network deployment

Local test network

Public test network



Main network

Artifacts

- Transaction for creating the contract
- An address for the contract if the creation was successful

Truffle – Development framework for smart contracts

Truffle is a popular framework to facilitate the development of Ethereum smart contracts. The framework provides tools to compile, test and deploy Solidity contracts.

- Open source under the MIT license and hosted at Github: https://github.com/trufflesuite/truffle
- Built-in network management that allows a developer to deploy a smart contract on various networks, e.g. live and test
- Web-pack like automated re-compilation on code changes
- Testing based on Mocha and Chai
- Provides project scaffolding
- Truffle Quickstart

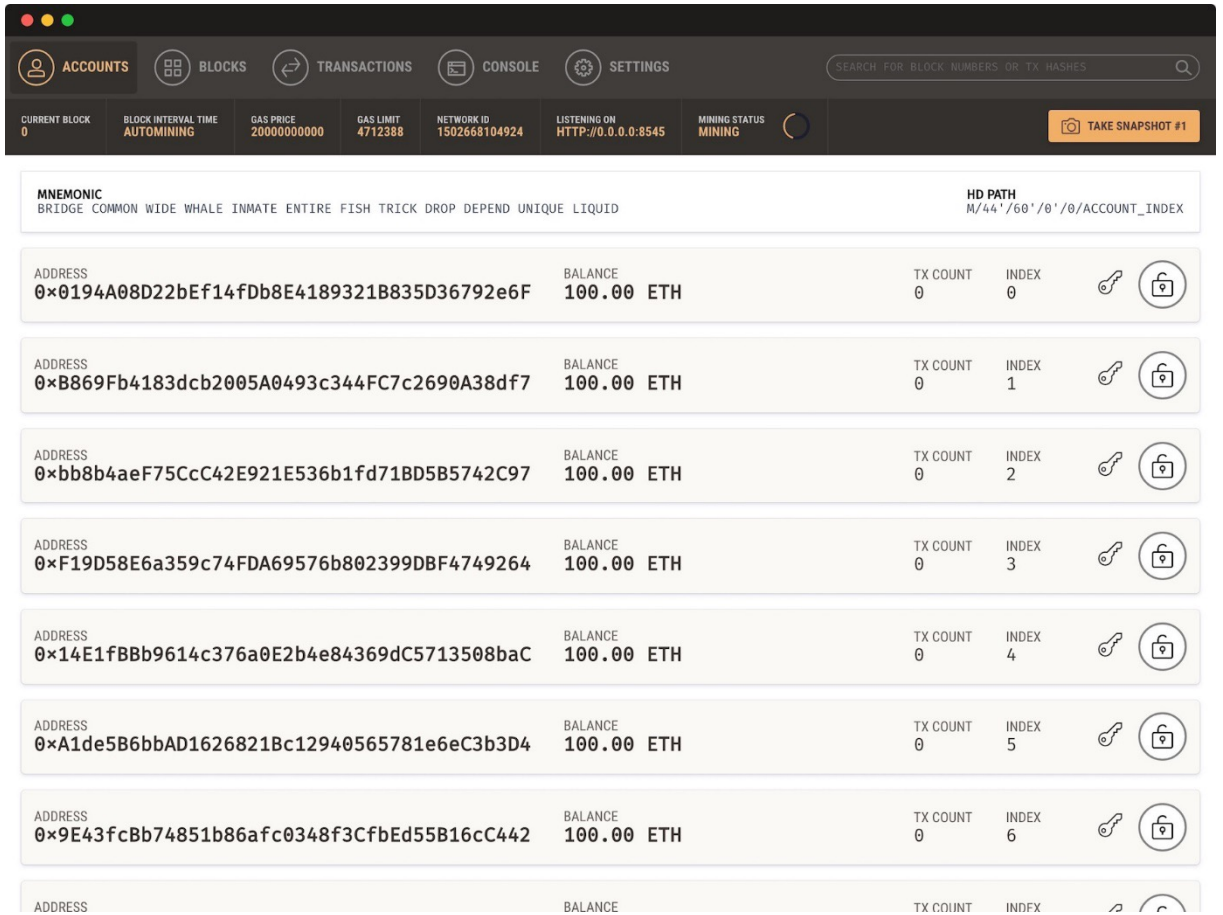
Ganache – Private Ethereum test network

Ganache is a local blockchain for Ethereum smart contract development. It can be used to deploy, simulate, and test smart contracts.

- Open source under the MIT license and hosted at Github: https://github.com/trufflesuite/ganache
- Integrates a custom block explorer interface with additional debugging features.
- Uses workspaces to provide multiple Ethereum blockchains with different settings (blocktime etc.)
- Can be linked to Truffle projects to automate tests for smart contracts

Ganache – Private Ethereum test network (cont.)

Ganache ships with a ready-made and developer friendly block explorer



Overview development tools (cont.)

Cloud environment

IDE / Code editor + Build management

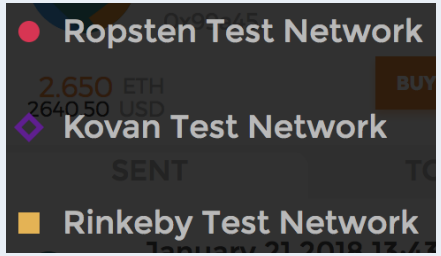
Artifacts

- Solidity source code files
- Compiled contracts bytecode
- Contract ABIs / JSON

Network deployment

Remix emulation

Public test network



Main network

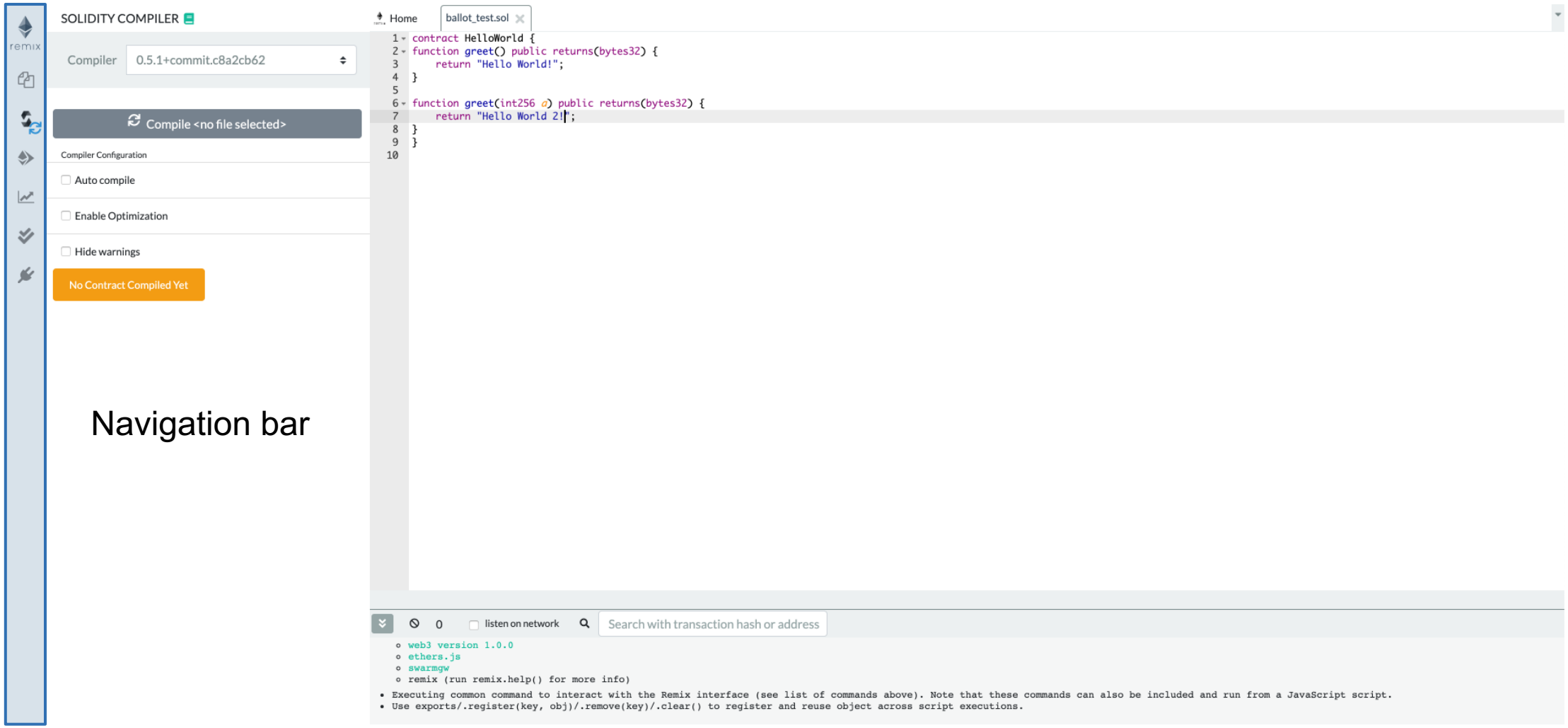
Artifacts

- Transaction for creating the contract
- An address for the contract if the creation was successful

Remix IDE

Check out Remix IDE: https://remix.ethereum.org/

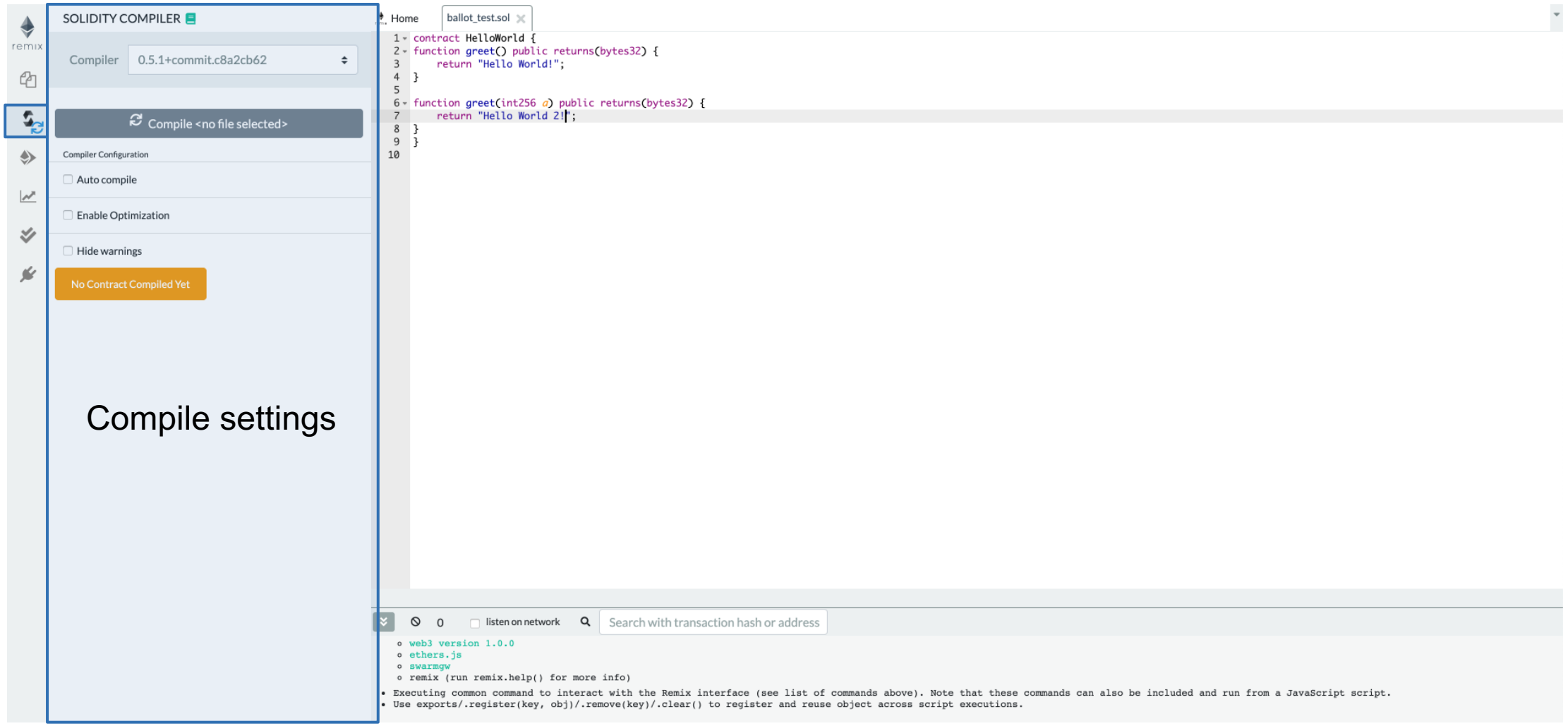
Remix IDE



Navigation bar

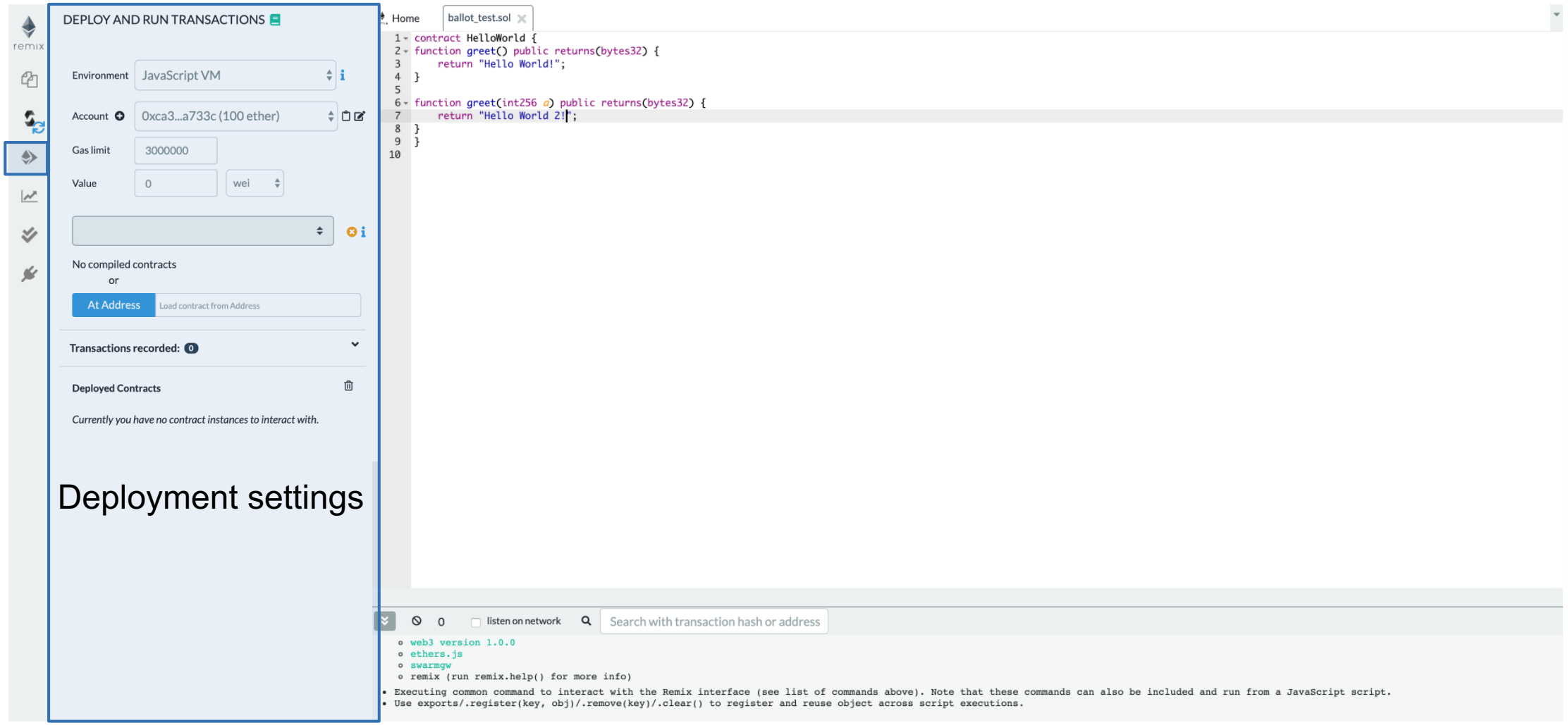
Check out Remix IDE: https://remix.ethereum.org/

Remix IDE



Check out Remix IDE: https://remix.ethereum.org/

Remix IDE



Check out Remix IDE: https://remix.ethereum.org/

Test networks

Test networks provide a convenient way to publicly deploy and test smart contracts in a realistic environment.

Ropsten
- Free to use
- Public
- Block time of ~30s
- Proof of work consensus
- Geth and Parity compatibility
- Ether distribution via faucet: https://faucet.ropsten.be/
- Anonymous

Rinkeby
- Free to use
- Public
- Block time of ~15s
- Proof of authority consensus, i.e., one central instance decides what transaction will be mined.
- Geth only
- Ether distribution via faucet: https://faucet.rinkeby.io/
- Twitter or Facebook account required

Kovan
- Free to use
- Public
- Block time of ~4s
- Proof of authority consensus, i.e., one central instance decides what transaction will be mined.
- Parity only
- Ether distribution via faucet: https://github.com/kovan-testnet/faucet
- Github account required