Decentralized Systems

Solidity & Remix demo

Write Smart Contract

Solidity

- Introduced by the Ethereum foundation to write smart contracts
- To work with Solidity we need
  - A framework to **write** and **compile** a contract
  - An Ethereum node to **deploy** the contract
  - A library to build the **front-end** and to **interact** with the contract through JSON-RPC calls

Solidity

- Extension **.sol**
- Needs
  - **License** (// SPDX-License-Identifier: MIT)
  - **Version** (**pragma** solidity *version_number)*
- **Contracts** are similar to **classes** and contain declaration of
  - Typed variables
  - Constructors
  - Functions and function modifiers
  - Events

Solidity: data location

- **Data location**
  - **Storage**: for permanent data (stored on blockchain, expensive)
  - **Memory**: used to save temporary variables during function execution (often required in return parameters)
  - **Calldata**: non-modifiable and non-persistent data location, default location for function arguments

The amount of gas used during a transaction depends on the data location used in the smart contract. The best practice is to write an optimized code that uses a minimum amount of storage

Solidity: variables

- **Variables**
  - **Private**: accessible only by functions within the contract itself
  - **Public**: part of the **contract interface**, can be accessed both inside and outside of the contract
  - **Internal**: (default) can only be accessed by functions within the contract itself, and by functions in derived contracts

It is important to choose the correct visibility for Solidity variables, as this can have a significant impact on the security of the contract

Solidity: functions

- **Functions**
  - **Private**: can only be called inside the current contract
  - **Public**: part of the contract interface, they can be called via transactions and from other contracts
  - **View**: do not modify the state
  - **Pure**: do not modify the state and do not read from the state
  - **Payable**: can receive Ether

Use the payable modifier only when necessary and be careful with reentrancy attacks (more later)

Solidity: contract scheleton

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;

contract DummyContract {
        state variables
        events
        modifiers
        constructor
        functions
}
```

Solidity: contract scheleton

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;

contract DummyContract {
        state variables
        events
        modifiers
        constructor
        functions
}
```

Solidity: data types

- **Boolean** (bool; default false)
- **Integer** (int8,…, int256, uint8, uint256; default 0)
  - The use of uint prevents programming errors for values which cannot be negative
  - In case of int, uint the compiler allocates 256 bits

  Overflows of integer variables were exploited by malicious actors to attack smart contracts (today not possible anymore)
- **Strings** (text data; default “”)
- **Bytes** (raw binary data; default 0x00)

Solidity: data types

- **Array** (values of the same type)
  - uint256[ ] lottery;
  - string[12] months;
- **Struct** (group of related variables)
  - struct Player {

```
    string playerNickname;
    uint8 intelligence;
    uint8 strenght;
}
Player player1;
Player player2;
```

Solidity: data types

- **Address** (store Ethereum addresses, 20 bytes)
  - Built in function **balance** to read the balance of a given account
- **Mapping** (key/value pairs)
  - Example: we want to keep track of the players in a game

    1) Player[ ] players
    (integer indexes in the array)
    2) mapping (address => Player) players
    (we can keep track of players' addresses)

Solidity: constant

- **Constant**
  - *Variables whose values cannot be changed after initialization*
  - **Constant** variables are initialized at **compile time**

    ```
    Examples:
    uint constant MAX_SUPPLY = 100000;
    address constant OWNER_ADDRESS = 0x1234567890....;
    ```

  - Stored in the contract's **bytecode**, they can be accessed without accessing the contract's storage
  - **Do not consume any gas when accessed**, since they do not require a storage read

Solidity: immutable

- **Immutable**
  - *Variables whose values cannot be changed after initialization*
  - **Immutable** variables are **initialized during contract deployment**, in the contract constructor (see later)

    ```
    Examples:
    uint immutable InitialBalance;
    address immutable OwnerAddress;
    ```

  - Saved in the **bytecode**, they are useful for storing values such as addresses, hashes, and other **data that do not need to change over time** but are not known at compile time

Solidity: variable scope

- **Global** scope
  - **State variables**, stored in contract **storage**, are accessible from all functions in the contract
  - Special **built-in variables** are global (accessible from all contracts)
    - msg.sender
    - address.balance
    - block.timestamp

- **Local** scope
  - Variable defined into functions, stored in **memory**

Solidity: constructor

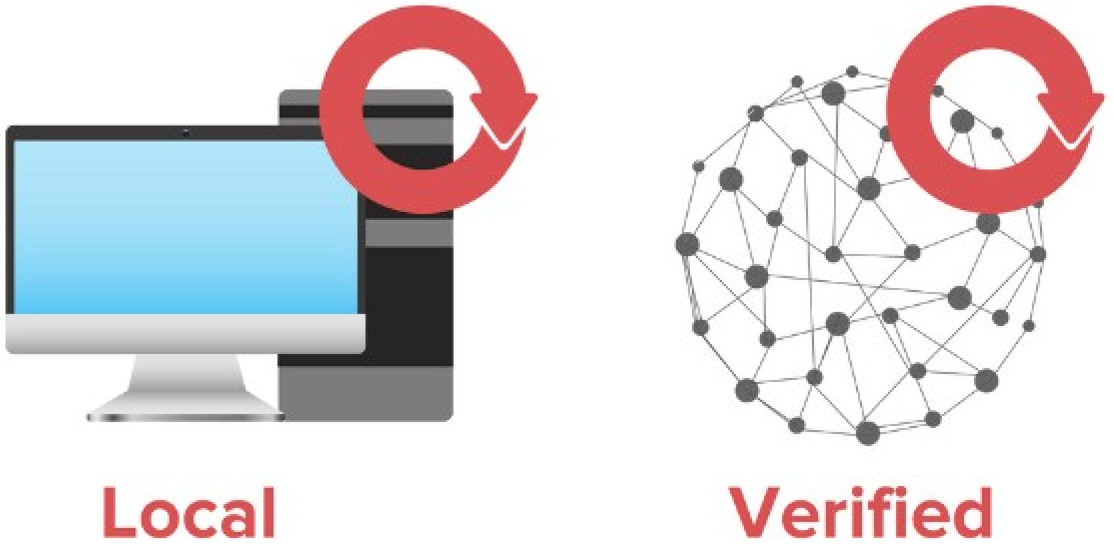
- **Constructor**
  - optional function that is executed upon contract creation

```
contract DummyContract {
    address public immutable OwnerAddress;
    constructor() {
        OwnerAddress = msg.sender;
     }
    …
}
```

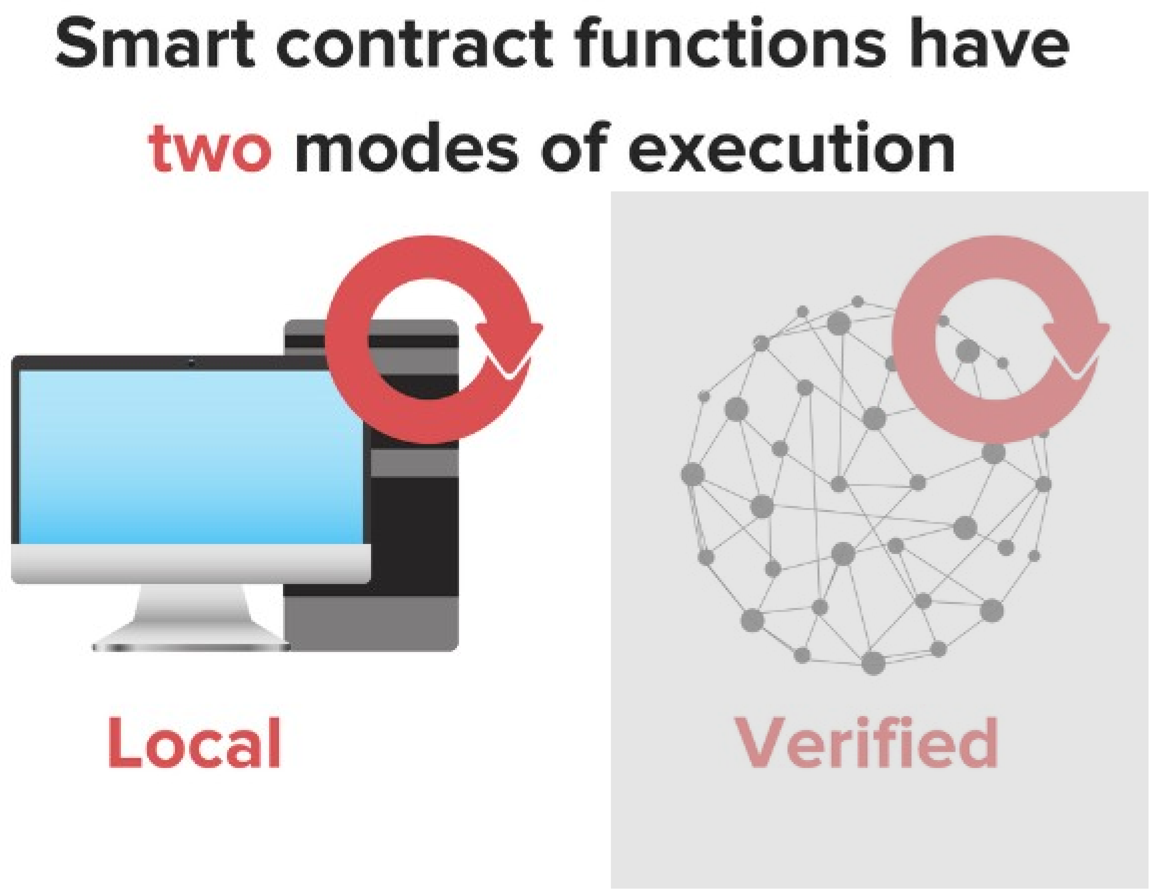
Known only at deploy time

Solidity: execution

Smart contract functions have two modes of execution

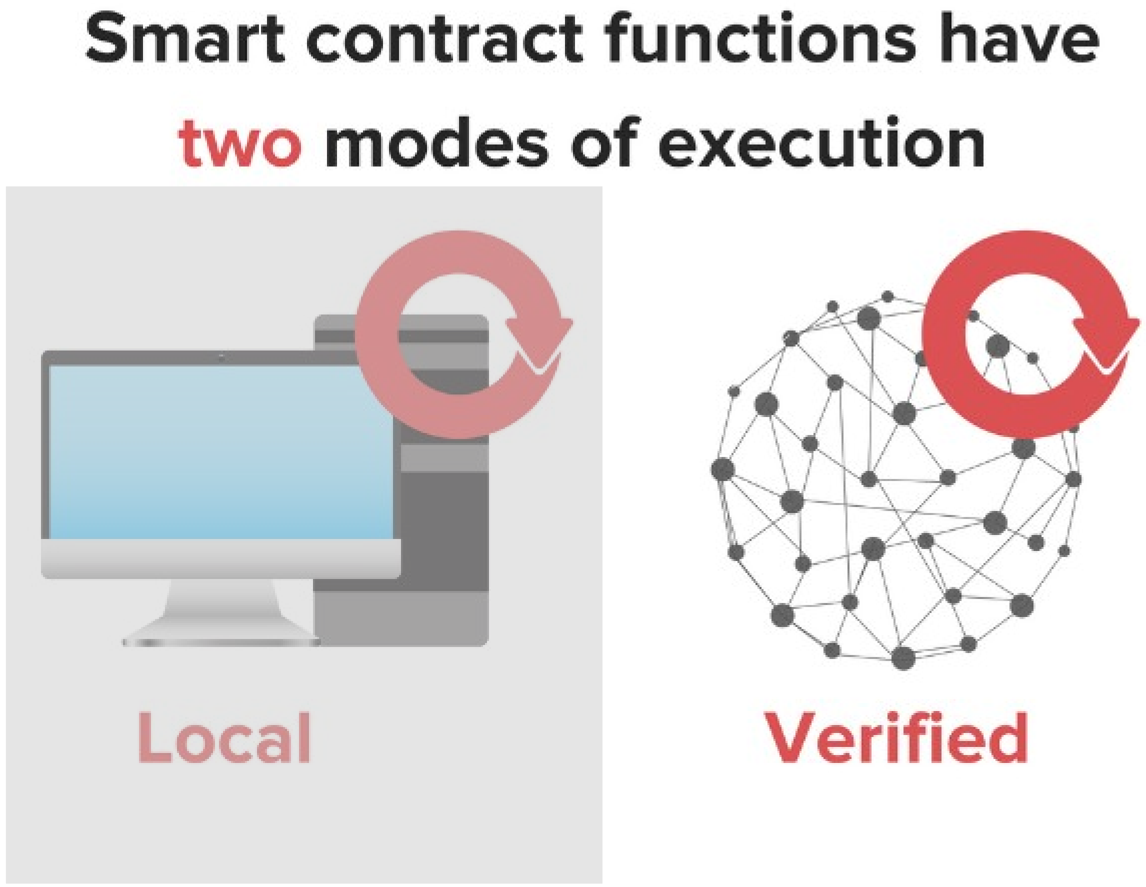


Solidity: call



- A **call** is a local invocation of a function that **does not broadcast or publish anything on the blockchain**
- **Read-only** operation, does **not consume any gas**
- It is **synchronous** and the return value is sent back immediately
- Used for view and pure functions
- Its underlying JSON-RPC is eth_call

Solidity: transaction



- A **transaction** is **broadcast to the network**, processed by validators
- If **valid,** it is **added to a block**
- **Write-operation** that updates the state of the blockchain and **consumes gas**
- It is **asynchronous**
- The **immediate return value** is always the **tx's hash**
- Its underlying JSON-RPC is eth_sendTransaction

Solidity: compiler

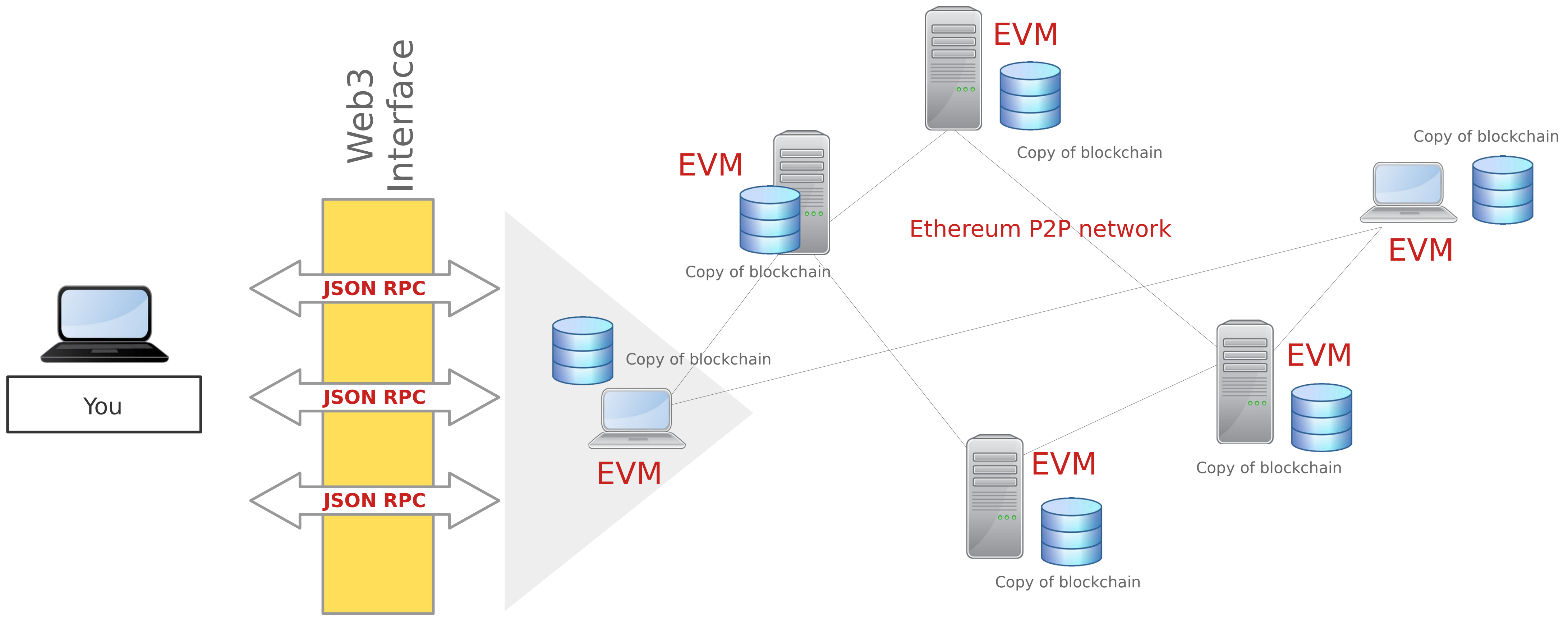
- Produces
  - the **bytecode**, e.g., the executable code on the EVM
  - the Application Binary Interface (**ABI**), e.g., an interface to interact with the EVM bytecode (JSON-like syntax)

Deploy Smart Contract

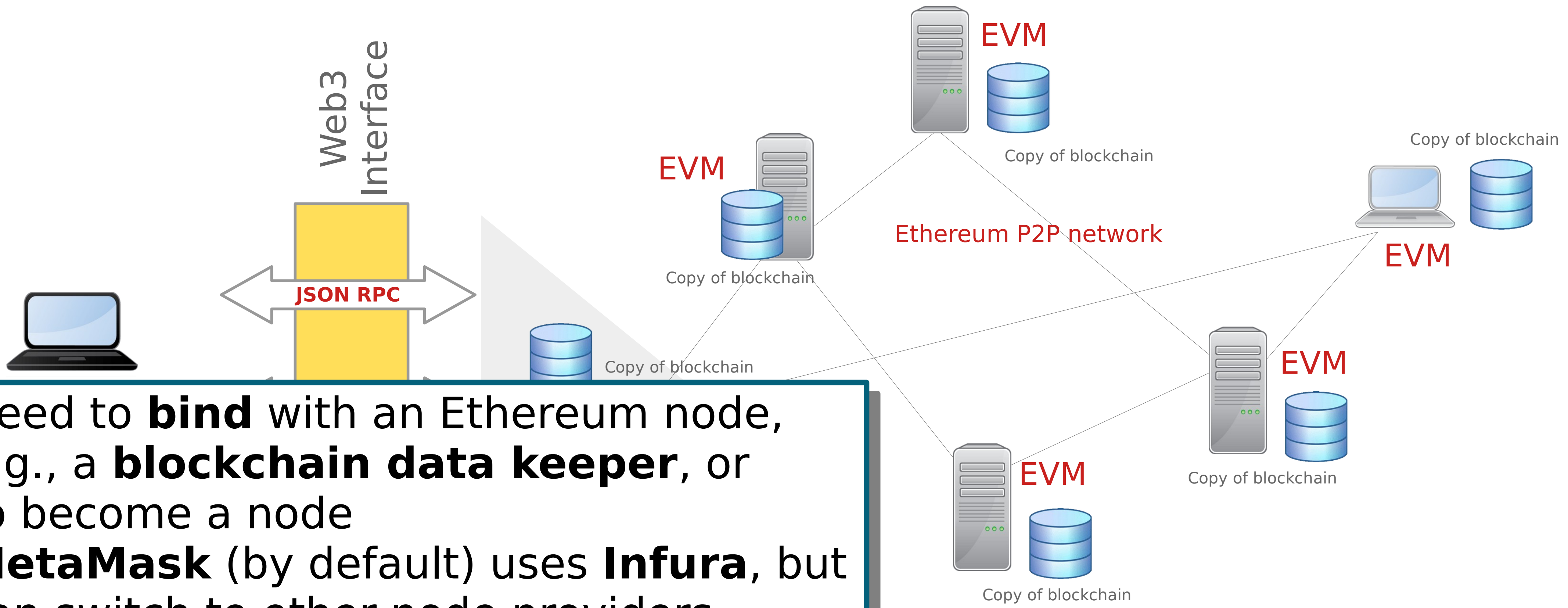
Deploy smart contract

- You need
  - bytecode
  - ABI
  - wallet with a balance
  - infrastructure

Infrastructure



Infrastructure



Need to **bind** with an Ethereum node, e.g., a **blockchain data keeper**, or to become a node
MetaMask (by default) uses **Infura**, but can switch to other node providers

Software

- Huge ecosystem…
- We will start with
  - Remix IDE (to write, compile, deploy demo smart contracts)
  - Metamask (to transact on the testnet)
  - Sepolia testnet (to see what happens)

Remix IDE

- You can access the Remix IDE in two different ways:
  - via a **web browser** (https://remix.ethereum.org/)
  - from a locally installed copy (npm install remix-ide -g)
- It provides various tools to write, compile, and deploy smart contracts

Remix IDE

- Remix Execution Environments
  - **JavaScript VM**: a sandbox blockchain implemented with JavaScript in the browser to emulate a real blockchain
  - **Injected Web3**: a provider that injects web3 such as Metamask, connecting you to your local blockchain or to a test net
  - **Web3 Provider**: a remote node with geth, parity or any Ethereum client. Can be used to connect to the real network, or to your private blockchain directly without MetaMask in the middle

Useful links

- https://solidity-by-example.org/
- https://cryptozombies.io/
- https://ethernaut.openzeppelin.com/

Start
DEMO