DeFi Design Patterns

Prepared by Kirill Sizov

EIP

Ethereum Improvement Proposals (EIPs) describe standards for the Ethereum platform, including core protocol specifications, client APIs, and contract standards.

EIP Process

1. **Proposal**: Idea submission.
2. **Draft**: Formal proposal writing.
3. **Review**: Community review and feedback.
4. **Last Call**: Final review.
5. **Accepted/Final**: Standard adoption.

ERC

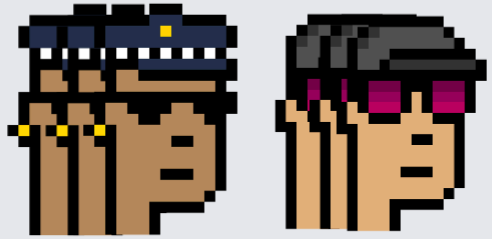
Ethereum Request for Comments (ERC) is a form of EIP that focuses on application standards and conventions.

Token standards

- **ERC-20**: Standard for fungible tokens.
- **ERC-721**: Standard for non-fungible tokens (NFTs).
- **ERC-1155**: Standard for semi-fungible tokens.

Fungible

Non-Fungible



Semi-Fungible

ERC-20

- EIP link
- Openzeppelin implementation

FUNCTIONS

```
totalSupply()
balanceOf(account)
transfer(recipient, amount)
allowance(owner, spender)
approve(spender, amount)
transferFrom(sender, recipient, amount)
```

EVENTS

```
Transfer(from, to, value)
Approval(owner, spender, value)
```

ERC-721

- EIP link
- Openzeppelin implementation

FUNCTIONS

```
balanceOf(owner)
ownerOf(tokenId)
safeTransferFrom(from, to, tokenId)
transferFrom(from, to, tokenId)
approve(to, tokenId)
getApproved(tokenId)
setApprovalForAll(operator, _approved)
isApprovedForAll(owner, operator)
safeTransferFrom(from, to, tokenId, data)
```

`supportsInterface(interfaceId)` IERC165

EVENTS

```
Transfer(from, to, tokenId)
Approval(owner, approved, tokenId)
ApprovalForAll(owner, operator, approved)
```

ERC-1155

- EIP link
- Openzeppelin implementation

FUNCTIONS

```
balanceOf(account, id)
balanceOfBatch(accounts, ids)
setApprovalForAll(operator, approved)
isApprovedForAll(account, operator)
safeTransferFrom(from, to, id, amount, data)
safeBatchTransferFrom(from, to, ids, amounts, data)
```

```
supportsInterface(interfaceId)                    IERC165
```

EVENTS

```
TransferSingle(operator, from, to, id, value)
TransferBatch(operator, from, to, ids, values)
ApprovalForAll(account, operator, approved)
URI(value, id)
```

Access Control

Ownable

- For contracts that have a single administrative user.
- Owner can be another contract, such as multisig or DAO.
- OZ implementation

```solidity
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

contract MyContract is Ownable {
    constructor(address initialOwner) Ownable(initialOwner) {}

    function normalThing() public {
        // anyone can call this normalThing()
    }

    function specialThing() public onlyOwner {
        // only the owner can call specialThing()!
    }
}
```

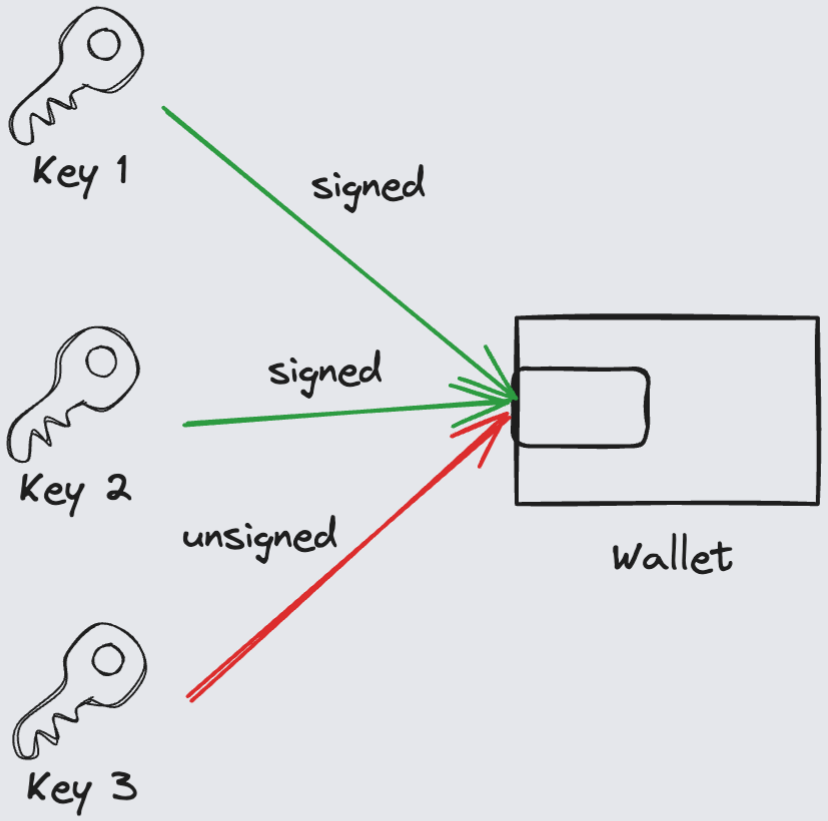
Role-based access

- More granular levels of permission may be implemented than were possible with the simpler ownership.
- OZ implementation

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import {AccessControl} from "@openzeppelin/contracts/access/AccessControl.sol";
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract AccessControlERC20Mint is ERC20, AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

    constructor(address minter, address burner) ERC20("MyToken", "TKN") {
        _grantRole(MINTER_ROLE, minter);
        _grantRole(BURNER_ROLE, burner);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }

    function burn(address from, uint256 amount) public onlyRole(BURNER_ROLE) {
        _burn(from, amount);
    }
}
```

Multisig

- A multisignature wallet requires multiple keys to authorize a transaction.
- A designated minimum number of keys must approve a transaction (e.g., 2-of-3, 3-of-5)
- Example implementation



DAO

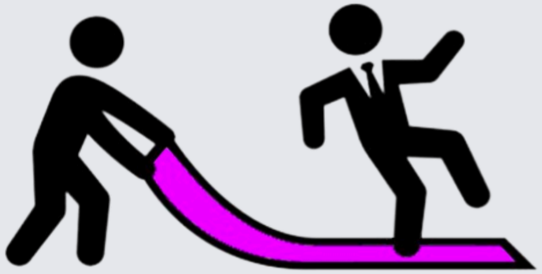
DAOs are member-owned communities without centralized leadership.

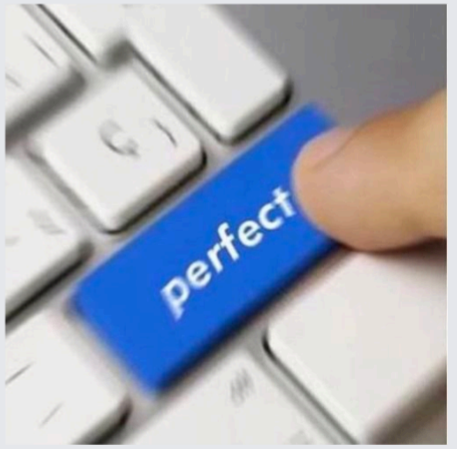
Governance token

- Tokens that grant voting power in DAO governance.
- Can be earned, purchased, or distributed at inception.
- Token holders vote on organizational decisions.

Upgradable contracts

DeFi project launched and raised the funds





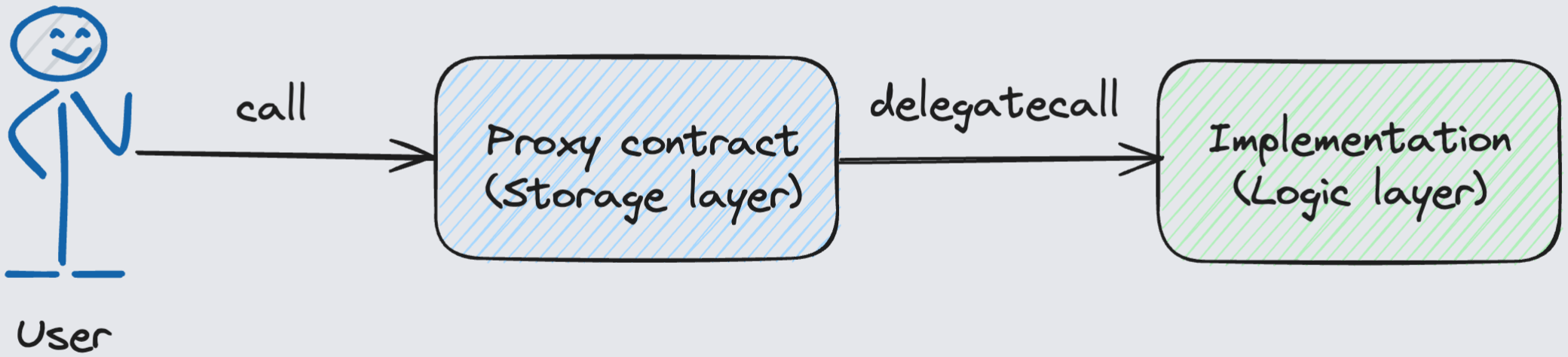
Delegatecall

`delegatecall` is a low level function similar to call.

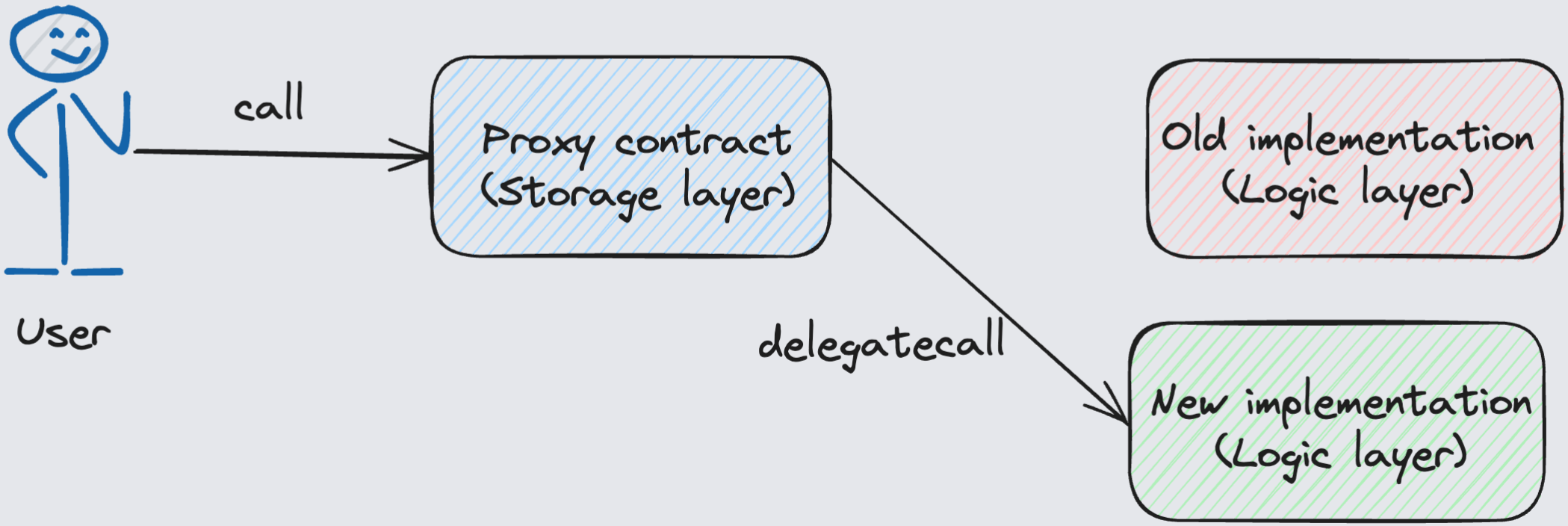
When contract `A` executes `delegatecall` to contract `B`, `B`'s code is executed
with contract `A`'s storage, `msg.sender` and `msg.value`.

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

// NOTE: Deploy this contract first
contract B {
    // NOTE: storage layout must be the same as contract A
    uint public num;
    address public sender;
    uint public value;

    function setVars(uint _num) public payable {
        num = _num;
        sender = msg.sender;
        value = msg.value;
    }
}

contract A {
    uint public num;
    address public sender;
    uint public value;

    function setVars(address _contract, uint _num) public payable {
        // A's storage is set, B is not modified.
        (bool success, bytes memory data) = _contract.delegatecall(
            abi.encodeWithSignature("setVars(uint256)", _num)
        );
    }
}
```

Core idea


call
Proxy contract
(Storage layer)
delegatecall
Implementation
(Logic layer)
User

Upgrading


call
Proxy contract
(Storage layer)
Old implementation
(Logic layer)
User
delegatecall
New implementation
(Logic layer)

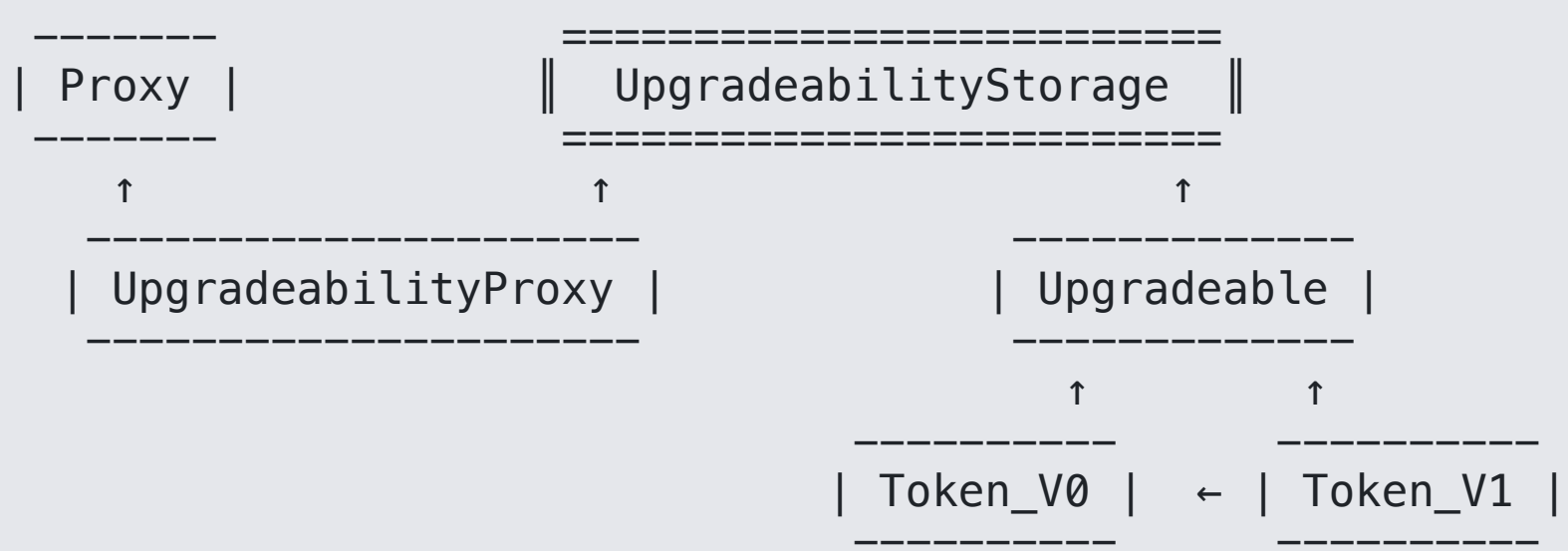
Proxy contract by OZ

- Proxy: Abstract contract implementing the core delegation functionality.
- ERC1967Utils: Internal functions to get and set the slots defined in EIP1967.
- ERC1967Proxy: A proxy using EIP1967 storage slots.

Proxy patterns

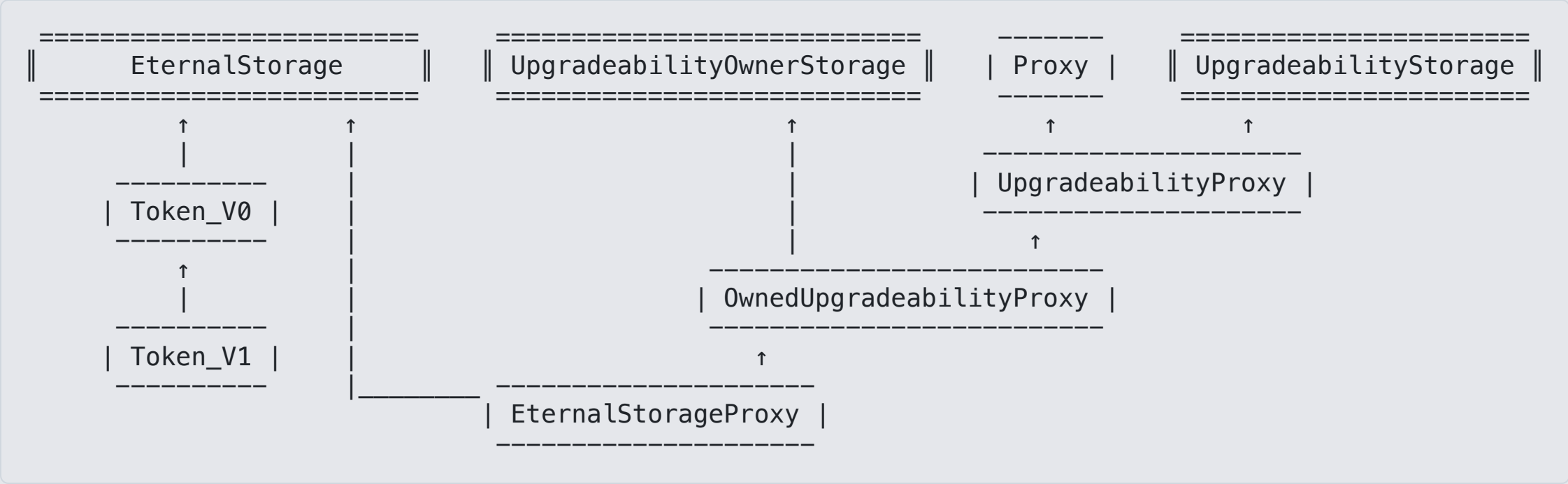
Inherited storage

- Each version will follow the storage structure of the previous one.
- Example of implementation.



Eternal storage

- Store all variables in mappings.
- Example of implementation.
- Not really used much.



Unstructured storage

- Use fixed "random" storage slots to store the required data.
- Example of implementation.

Upgrading pattern

Transparent

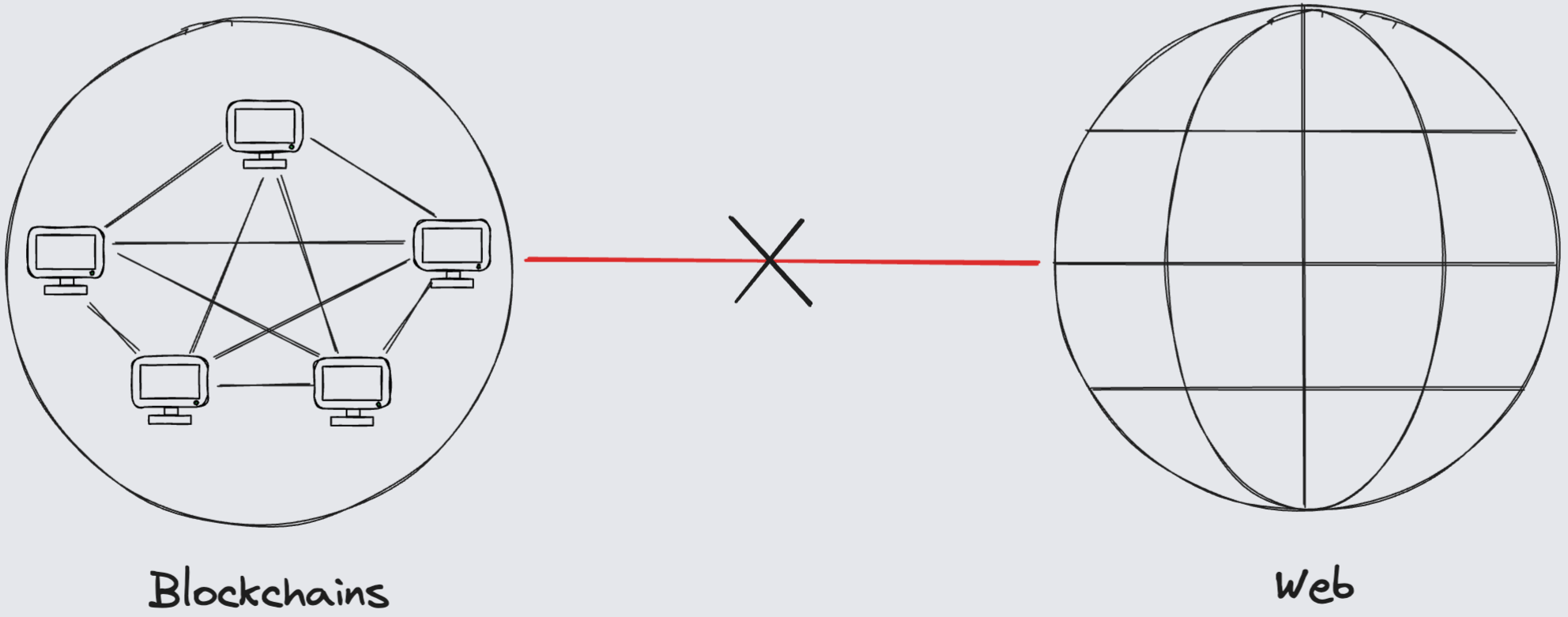
- An upgrade is handled by proxy contract.
- More gas during calls.
- Implementation

UUPS

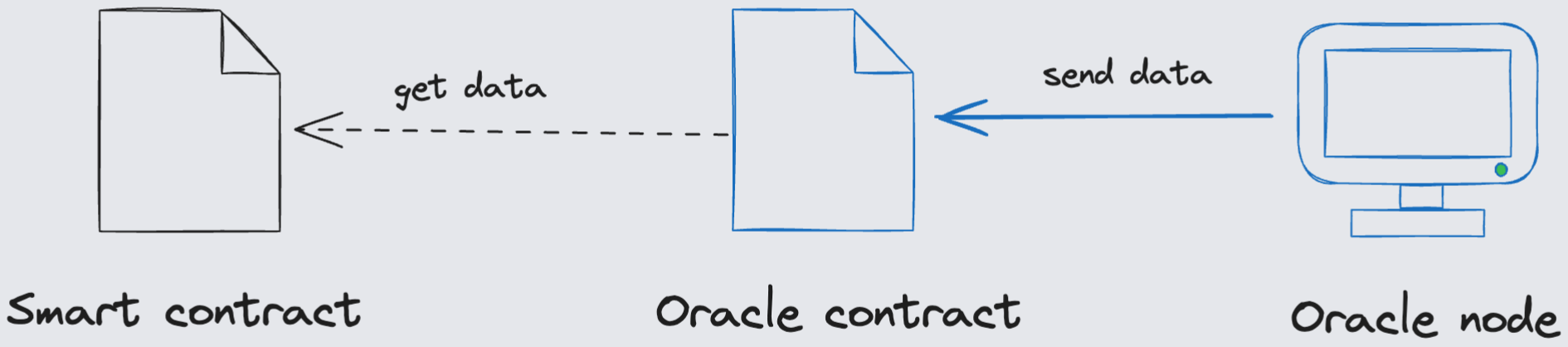
- An upgrade is handled by an implementation contract.
- You have to be more careful.
- Implementation

Oracles

Problem



Architecture



Design patterns

Centralized:

- Single oracle
- Single failure point
- Single attack target
- Fast

CFT resistant:

- Set of nodes controlled by single entity
- No single attack point
- Fast

BFT resistant:

- Set of nodes chosen by consensus algo
- No single attack target
- Not so fast