The Next 700 EVM Languages

Francisco Giordano
𝕏 @frangio_

Background

OpenZeppelin Contracts

Repository of secure code

Library of abstractions

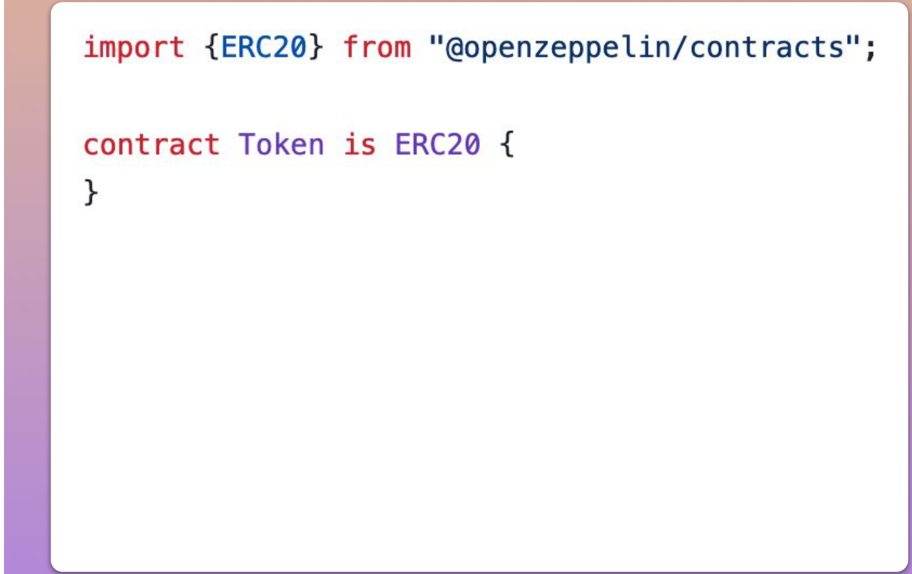
Safe abstractions

Example

ERC20 Mint

Properties & Invariants

1. Transfer only up to balance
2. Sum of balances = Total supply
3. Events must track state



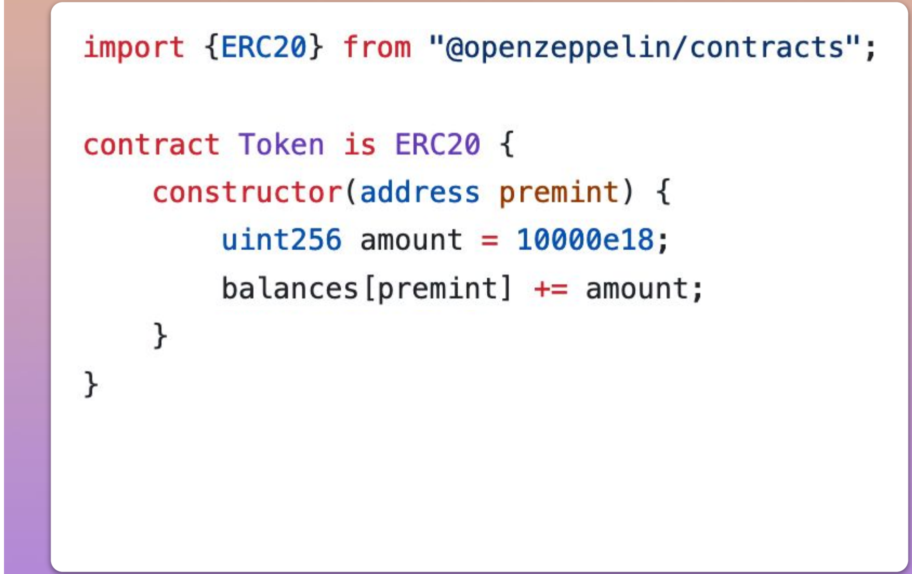
```
import {ERC20} from "@openzeppelin/contracts";

contract Token is ERC20 {
}
```

Example

ERC20 Mint

Properties & Invariants

1. Transfer only up to balance
2. Sum of balances = Total supply
3. Events must track state



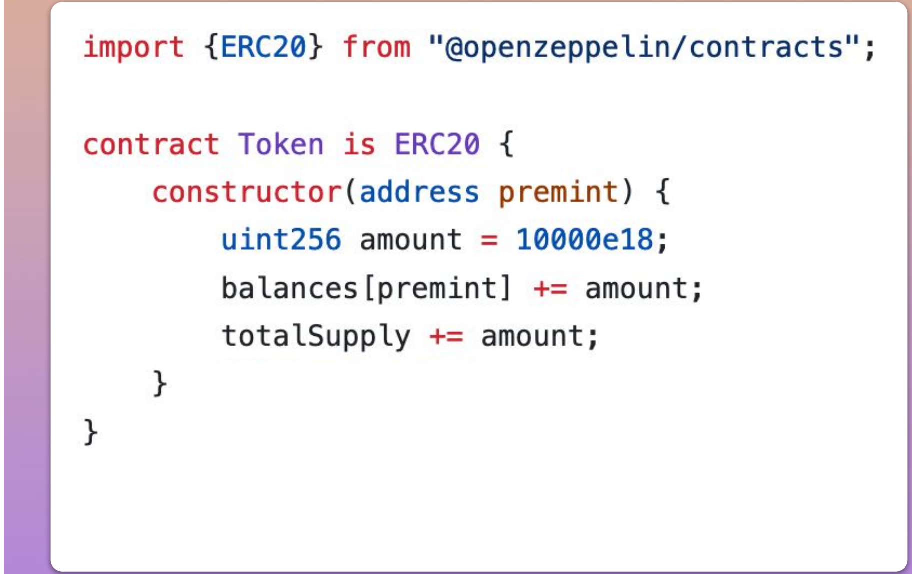
```
import {ERC20} from "@openzeppelin/contracts";

contract Token is ERC20 {
    constructor(address premint) {
        uint256 amount = 10000e18;
        balances[premint] += amount;
    }
}
```

Example

ERC20 Mint

Properties & Invariants

1. Transfer only up to balance
2. Sum of balances = Total supply
3. Events must track state



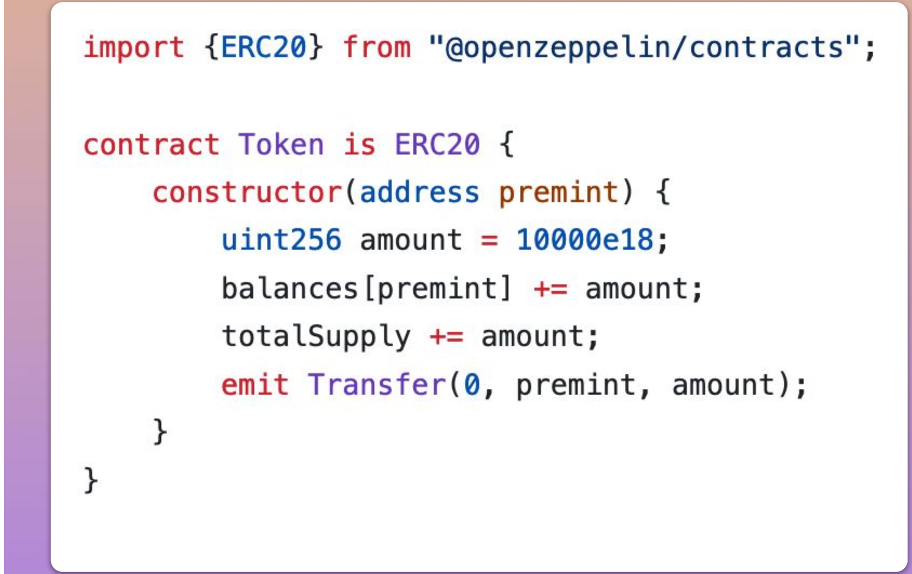
```
import {ERC20} from "@openzeppelin/contracts";

contract Token is ERC20 {
    constructor(address premint) {
        uint256 amount = 10000e18;
        balances[premint] += amount;
        totalSupply += amount;
    }
}
```

Example

ERC20 Mint

Properties & Invariants

1. Transfer only up to balance
2. Sum of balances = Total supply
3. Events must track state



```
import {ERC20} from "@openzeppelin/contracts";

contract Token is ERC20 {
    constructor(address premint) {
        uint256 amount = 10000e18;
        balances[premint] += amount;
        totalSupply += amount;
        emit Transfer(0, premint, amount);
    }
}
```

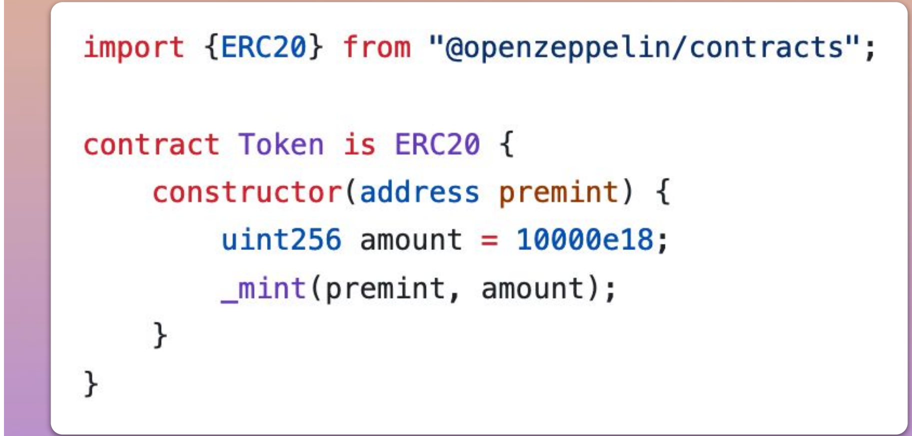
Example

ERC20 Mint

Properties & Invariants

1. Transfer only up to balance
2. Sum of balances = Total supply
3. Events must track state

Safe abstraction (2 & 3): `_mint`



```
import {ERC20} from "@openzeppelin/contracts";

contract Token is ERC20 {
    constructor(address premint) {
        uint256 amount = 10000e18;
        _mint(premint, amount);
    }
}
```

Example

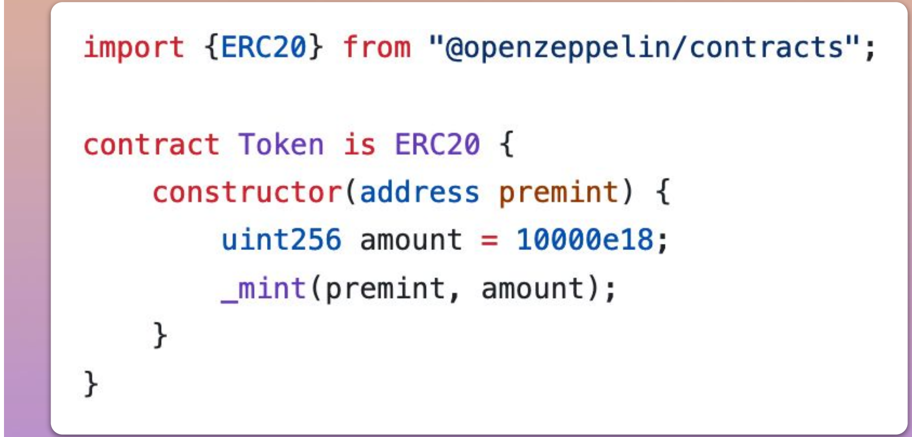
ERC20 Mint

Properties & Invariants

1. Transfer only up to balance
2. Sum of balances = Total supply
3. Events must track state

Safe abstraction (2 & 3): `_mint`

Real consequences: sDAI



```
import {ERC20} from "@openzeppelin/contracts";

contract Token is ERC20 {
    constructor(address premint) {
        uint256 amount = 10000e18;
        _mint(premint, amount);
    }
}
```

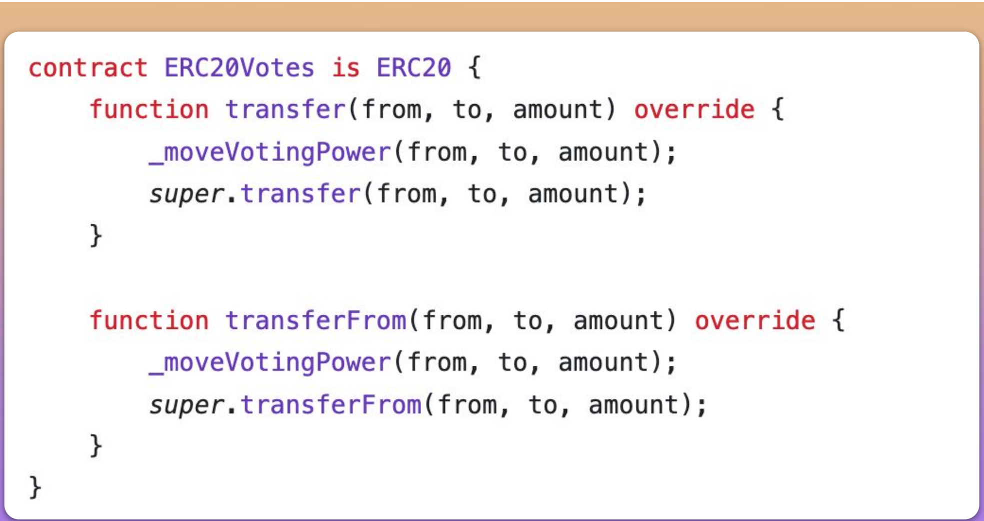
Extensibility

Modularity

Example

Transfer Hook

Goal: Consistent extension



```
contract ERC20Votes is ERC20 {
    function transfer(from, to, amount) override {
        _moveVotingPower(from, to, amount);
        super.transfer(from, to, amount);
    }

    function transferFrom(from, to, amount) override {
        _moveVotingPower(from, to, amount);
        super.transferFrom(from, to, amount);
    }
}
```

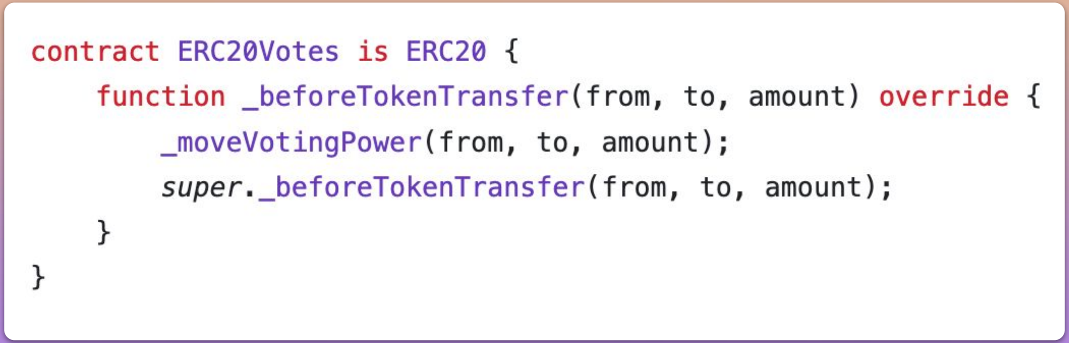
Example

Transfer Hook

Goal: Consistent extension

Abstraction: `_beforeTokenTransfer`

Abstraction leak: `super`



```
contract ERC20Votes is ERC20 {
    function _beforeTokenTransfer(from, to, amount) override {
        _moveVotingPower(from, to, amount);
        super._beforeTokenTransfer(from, to, amount);
    }
}
```

Istanbul

Congestion

2020

Gas efficiency

Assembly

Zero-cost Abstractions

> “when you use it,
> you get at least
> as good performance
> as if you had
> handcoded it”

Rust

Cairo

Noir

Move

Sway

Stylus

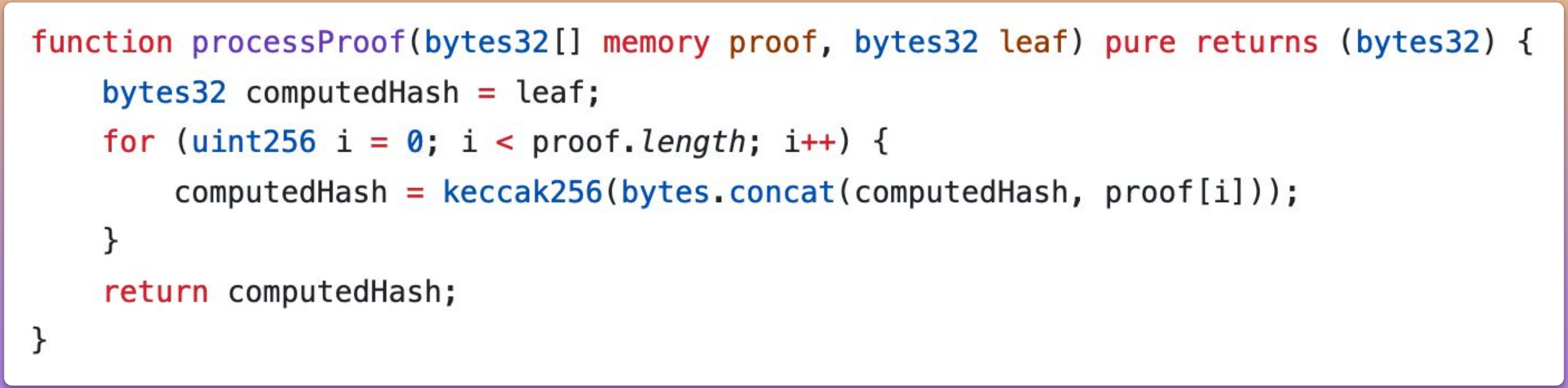
Solana

Example

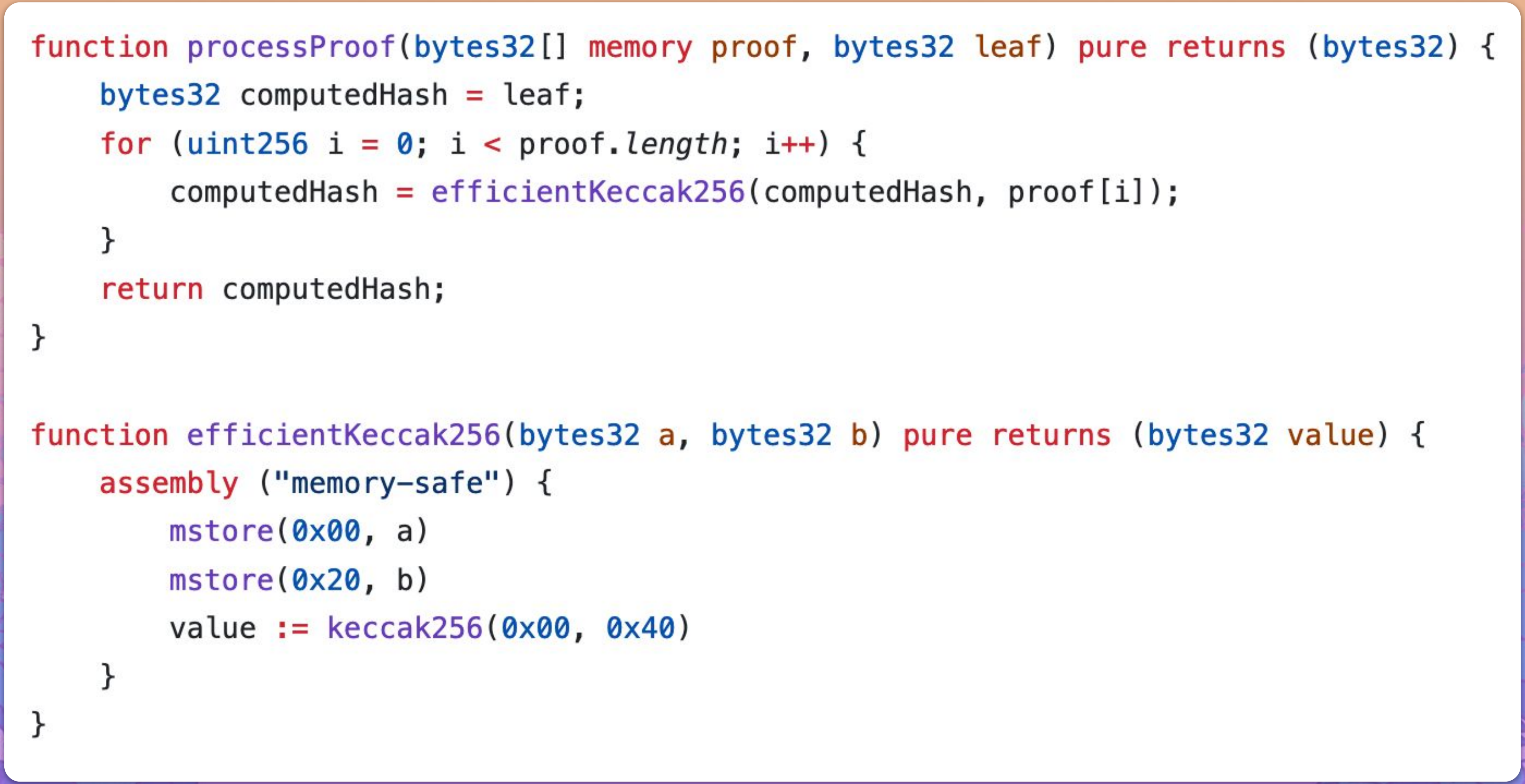
Merkle Proof Hash

High Level Abstraction: `bytes.concat`

Penalty: Overuse of memory



```
function processProof(bytes32[] memory proof, bytes32 leaf) pure returns (bytes32) {
    bytes32 computedHash = leaf;
    for (uint256 i = 0; i < proof.length; i++) {
        computedHash = keccak256(bytes.concat(computedHash, proof[i]));
    }
    return computedHash;
}
```



```
function processProof(bytes32[] memory proof, bytes32 leaf) pure returns (bytes32) {
    bytes32 computedHash = leaf;
    for (uint256 i = 0; i < proof.length; i++) {
        computedHash = efficientKeccak256(computedHash, proof[i]);
    }
    return computedHash;
}

function efficientKeccak256(bytes32 a, bytes32 b) pure returns (bytes32 value) {
    assembly ("memory-safe") {
        mstore(0x00, a)
        mstore(0x20, b)
        value := keccak256(0x00, 0x40)
    }
}
```

High Level Goals

Low Level Details

Compiler Risk

Still

Work Underway

Solidity

IR Pipeline

“Experimental” Solidity

Alternative compilers

Vyper

Security processes

Optimizations

Venom IR

Modules

EVML

First-class functions

Algebraic data types

Expressive type system

}

Swiss army knife

Zero-cost (*TBC*)

The Next 700 EVM Languages

Thank you!

Francisco Giordano

✉️ fg@frang.io

𝕏 @frangio_