Designing future-proof smart contract systems

Jorge Izquierdo
Devcon3

Aragon

- **Decentralized organizations** platform built on Ethereum.
- Usable by **non-technical** users.
- Allow **extendability** of the system with third party on-chain modules.

Future-proof smart contracts

- **Dumb contracts** are the best smart contracts
- EVM is expensive, optimize for **gas savings** (deploy and usage)
- Contracts need to be **upgraded**

- **Goal:** cheap, upgradeable, yet very simple contracts

The case against upgradeability

- Changing the rules on a live contract
- **Trust** required on the entity that can upgrade
- **Front-running** with an upgrade

Why upgradeable contracts

- Extremely young technology
- Solving **unanticipated bugs** that can result in irreversible loss of funds at the contract level
- Need to add **new features** based on user feedback

Doing upgrades right

- Not rely on just one entity
- Time delayed to allow for **vetting** and **auditing**
- **Governance** process

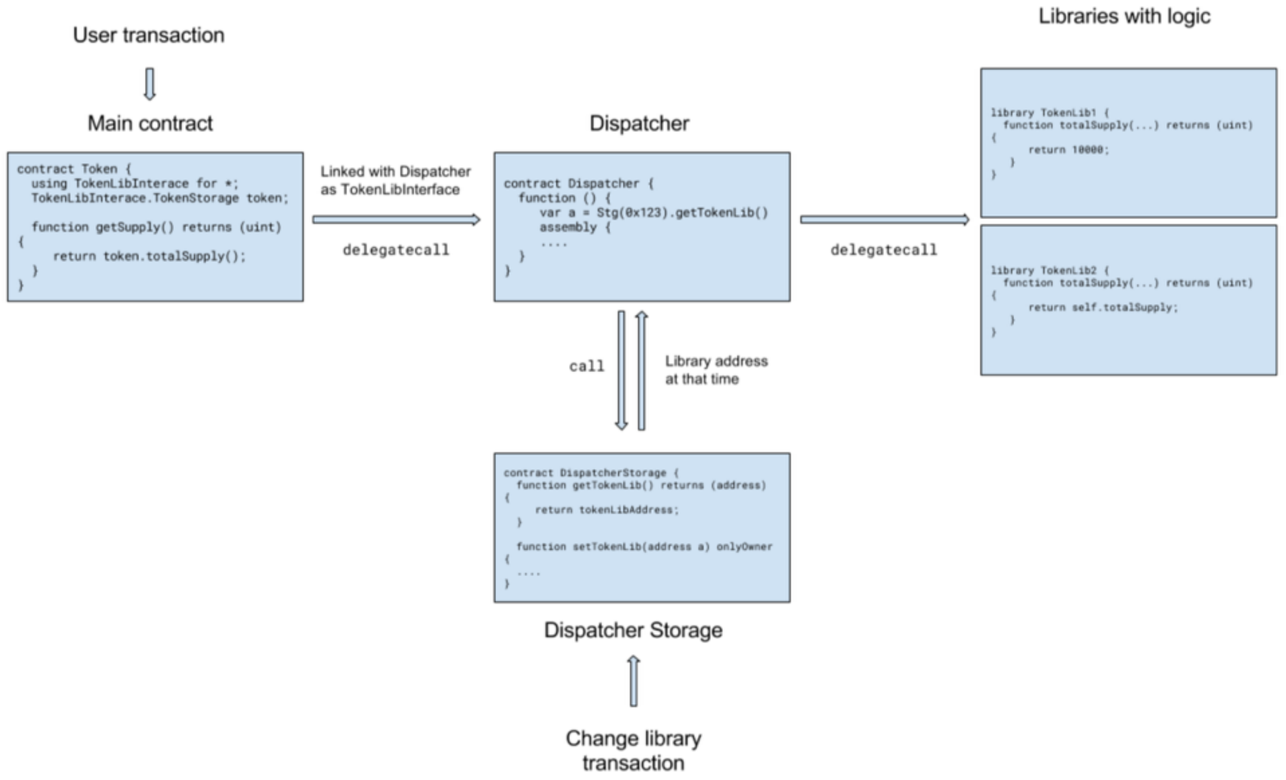
Solidity libraries

- `delegatecall` under the hood to linked library
- `using` semantics simulates calling **methods** on an **object**
- Library is deployed once and securely used by many contracts
- Separation of logic domains in a contract
- Allows for bigger contracts

```
library CounterLib {
    struct Counter { uint i; }

    function incremented(Counter storage self) returns (uint) {
        return ++self.i;
    }
}

contract CounterContract {
    using CounterLib for CounterLib.Counter;

    CounterLib.Counter counter;

    function increment() returns (uint) {
        return counter.incremented();
    }
}
```

Aragon Blog: **Library Driven Development**

Upgradeable libraries



github.com/maraoz/lib

Zeppelin + Aragon

Upgradeable libraries

Pros:

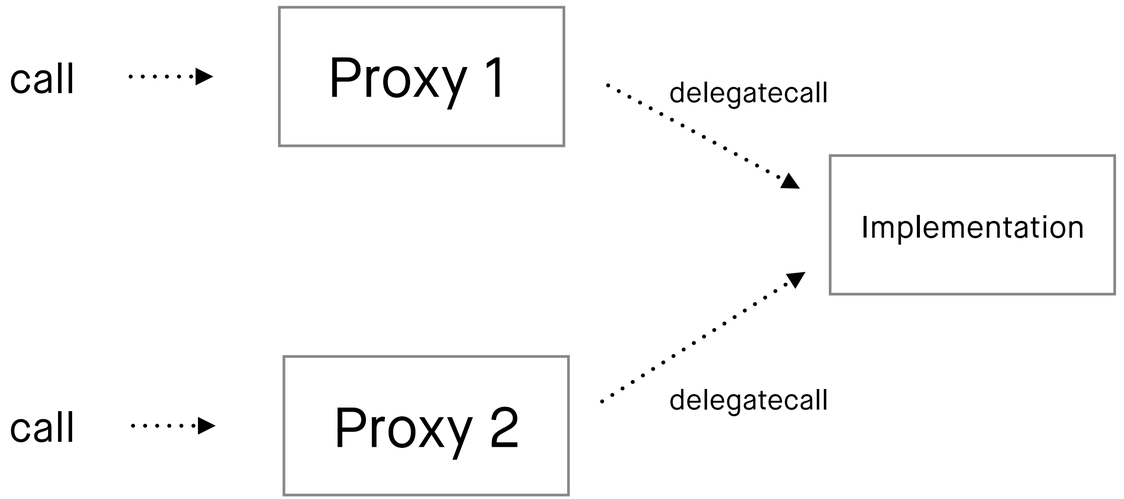
- Transparent for developer
- Allows to 'modify' the linked library

Cons:

- Main contract ABI cannot be modified
- Data structures are fixed

Delegate Proxy

- Run another contract's logic in the context of a contract.



Implementation (EIP 211)

```
contract DelegateProxy {
    function delegatedFwd(address _dst, bytes _calldata) internal {
        assembly {
            switch extcodesize(_dst) case 0 { revert(0, 0) }

            let result := delegatecall(gas, _dst, add(_calldata, 0x20), mload(_calldata), 0, 0)
            let size := returndatasize

            let ptr := mload(0x40)
            returndatacopy(ptr, 0, size)

            switch result case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

aragon-core: common/DelegateProxy.sol

Delegate Proxy flavors

- Static delegate proxy: **forwarders**
- **Upgradeable** proxies

Static forwarders

- Very cheap to deploy 'clone contracts'
- Useful for contracts with a low number of interactions

```
def mk_forwarder(address):
    code = b'\x36\x60\x00\x60\x00\x37' # CALLDATACOPY 0 0 (CALLDATASIZE)
    code += b'\x61\x10\x00\x60\x00\x36\x60\x00' # 4096 0 CALLDATASIZE 0
    code += b'\x73' + utils.normalize_address(address) + b'\x5a' # address gas
    code += b'\xf4' # delegatecall
    code += b'\x61\x10\x00\x60\x00\xf3' # 4096 0 RETURN
    return code

def mk_wrapper(code):
    lencodepush = b'\x60' + utils.encode_int(len(code)) # length of code
    returner = lencodepush +  b'\x60\x0c\x60\x00' # start from 12 in code, 0 in memory
    returner += b'\x39' # CODECOPY
    returner += lencodepush + b'\x60\x00' + b'\xf3' # return code
    assert len(returner) == 12
    return returner + code
```

Original idea: Vitalik's DELEGATECALL forwarders

Solidity forwarders

```
import "./DelegateProxy.sol";

contract Forwarder is DelegateProxy {
    address constant target = 0xbeef;

    function () payable {
        delegatedFwd(target, msg.data);
    }
}

contract ForwarderFactory {
    function clone() returns (address) {
        return address(new Forwarder());
    }
}
```

Gas overhead

- Added gas per call
  - 700 **delegatecall**
  - 3 + 3 * (returndatasize / 32) **returndatacopy**

- Deploy gas
  - ~66k gas **deploy** for barebones version
  - ~97k gas **deploy** Solidity (~87k using factory)

- 1M gas for contract deploy, ~1.3k calls break even

ENS Deed case study

- Deed create + parametrization = 620,741 gas
- Forwarder create (solidity) + setup call = 173,697 gas
- 340,565 found deed contracts
- Total gas saved = 152,247,539,860 gas
- Average 20 gwei gas price = 3044.95 ETH

Upgradeable Proxies

```
contract ProxyStorage {
    address target;
}

contract UpgradeableProxy is ProxyStorage, DelegateProxy {
    function UpgradeableProxy(address _target) {
        target = _target;
    }

    function () payable {
        delegatedFwd(target, msg.data);
    }
}

contract UpgradeableContract is ProxyStorage {
    function upgrade(address _newCode) {
        // do some checks here
        target = _newCode;
    }

    function foo() {
        // interesting upgradeable logic
    }
}
```

Original idea: Nick Johnson's upgradeable.sol

Storage in Delegate Proxies

- Storage layout must respect Proxy contract's layout.
  - Inherit Proxy's storage
- Data structures must be designed thinking on upgradeability

Solidity Storage slots review

- Storage counter starts at 0
- Reverse inheritance graph order
- Solidity packs contiguous smaller types to 32 bytes
- Structs are stored as inline types

TODO: addresses slots

```
contract Storage1 {
    uint256 a;  // slot 0
    uint128 b;  // slot 1 (bytes 17 to 32)
    uint64  c;  // slot 1 (bytes 9 to 16)
    bool    d;  // slot 1 (bit 64)
}

contract Storage2 is Storage1 {
    address x;  // slot 2
    Y y;
    address z;  // slot 5

    struct Y {
       uint256 i; // slot 3
       uint256 b; // slot 4
    }
}
```

Arrays

- Static length arrays behave just like normal types
  - `uint256[2] == uint256, uint256`
- Dynamic length arrays:
  - Store length in `p`
  - Array item position at `sha3(p) + arrayP`
  - Structs stored as inline items, adding to `arrayP`
  - Arrays in arrays follow same property

```
contract StorageArray {
    uint256[3] a; // slots 0, 1, 2
    uint256[] b;  // slot 3 (array length)

    function storeUint() {
        b.length = 10; // slot 3 = 10

        b[0] = 1;  // slot sha3(3) = 1
        b[5] = 2;  // slot sha3(3) + 5 = 2
    }

    struct C {
        uint256 a;
        uint256 b;
        uint256[] c;
    }

    C[] cs;

    function storeStruct() {
        cs.length = 2;       // slot 4 = 2

        cs[0].a = 1;         // slot sha3(4) = 1;

        cs[1].a = 1;         // slot sha3(4) + 3 = 1
        cs[1].b = 2;         // slot sha3(4) + 4 = 2
        cs[1].c.length = 1;  // slot sha3(4) + 5 = 1
        cs[1].c[0] = 2;      // slot sha3(sha3(4) + 5) = 2
    }
}
```

Mappings

- Values stored `sha3(key, p)`
- Nested mappings: `p` is where the mapping would have been stored if it was a normal value

```
contract StorageMapping {
    mapping (address => uint256) m;
    mapping (address => mapping (address => uint256)) mm;

    struct Map {
        mapping (address => uint256) m;
    }
    Map[] maps;

    function StorageMapping() {
        m[0x12] = 10;          // slot sha3(0x12, 0) = 10
        mm[0x12][0x34] = 11;   // slot sha3(0x34, sha3(0x12, 1)) = 11
        maps.length = 1;
        maps[0].m[0x12] = 12; // slot sha3(0x12, sha3(2) + 0) = 12
    }
}
```

Mappings

- Values stored `sha3(key, p)`
- Nested mappings: `p` is where the mapping would have been stored if it was a normal value

```
contract StorageMapping {
    mapping (address => uint256) m;
    mapping (address => mapping (address => uint256)) mm;

    struct Map {
        mapping (address => uint256) m;
    }
    Map[] maps;

    function StorageMapping() {
        m[0x12] = 10;          // slot sha3(0x12, 0) = 10
        mm[0x12][0x34] = 11;   // slot sha3(0x34, sha3(0x12, 1)) = 11
        maps.length = 1;
        maps[0].m[0x12] = 12; // slot sha3(0x12, sha3(2) + 0) = 12
    }
}
```

Warnings

- Adding just one storage value anywhere will increase p and break all storage.
- Failure will be silent, storage will be randomized.
- Always append new storage at the end.

Upgrade example

```
contract Payroll is ProxyStorage {
    struct Employee {
        uint256 salary;
    }

    Employee[] public employees;

    function newEmployee(uint256 salary) { employees.push(Employee(salary)); }
}

contract Payroll2 is ProxyStorage {
    struct Employee {
        uint256 salary;
        uint256 joinDate;
    }

    Employee[] public employees;

    function newEmployee(uint256 salary) { employees.push(Employee(salary, now)); }
}
```

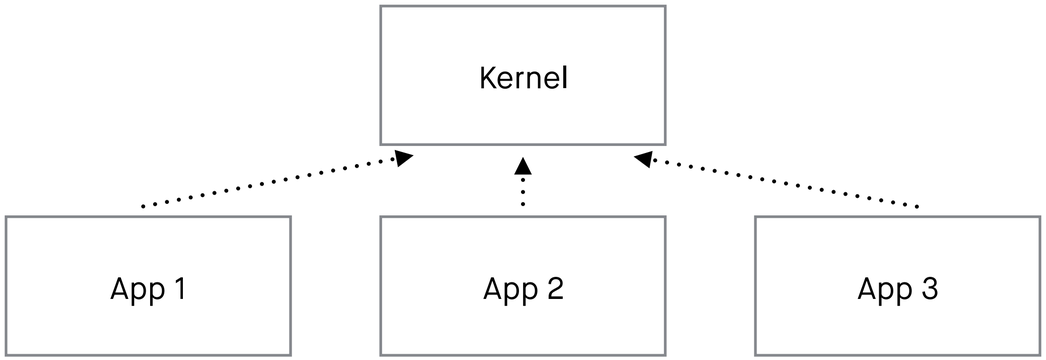
- Deploy Payroll
- Create 2 employees
- Upgrade to Payroll2
- Employee 1 joinDate = Employee2 salary
- Employee2 salary = 0

Upgrade example

```
contract PayrollM is ProxyStorage {
    struct Employee {
        uint256 salary;
    }

    mapping (uint256 => Employee) public employees;
    uint256 public employeeCounter;

    function newEmployee(uint256 salary) { employees[employeeCounter++] = Employee(salary); }
}


contract PayrollM2 is ProxyStorage {
    struct Employee {
        uint256 salary;
        uint256 joinDate;
    }

    mapping (uint256 => Employee) public employees;
    uint256 public employeeCounter;

    function newEmployee(uint256 salary) { employees[employeeCounter++] = Employee(salary, now); }
}
```

- Deploy PayrollM
- Create 2 employees
- Upgrade to Payroll2M
- Salaries are correct
- Join date before update is 0

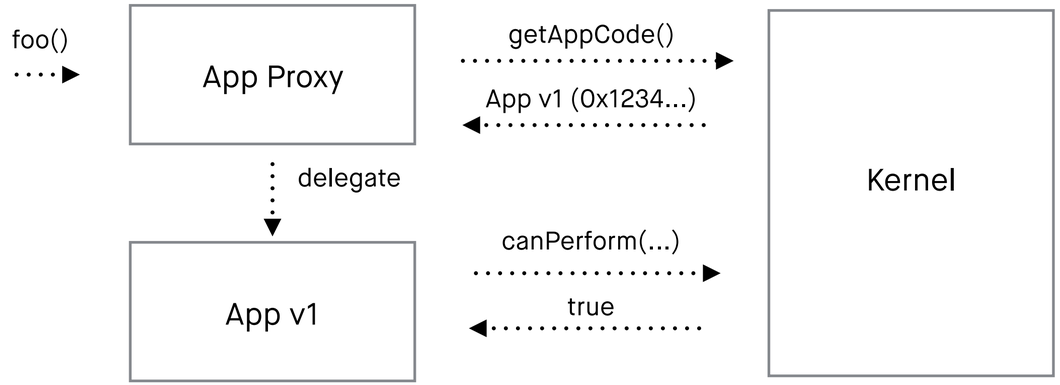
AragonOS



- Tiny kernel
- Upgradeable business logic on edges: apps

Kernel

- Context dependent **ACL**
- **Upgradeable** apps



ACL

- Usability/security balance
  - Can't rely on just one superuser or owner
  - Protect users from destructive actions
  - Different governance mechanism for different actions

- Purely address based whitelist
  - Apps as entities
  - Complex authentication on a second layer
  - New governance systems can be plugged in

ACL

- App defined roles for capabilities
- Granted permissions have an different parent
  - Parent can revoke the permission
  - If grantee is parent, grantee can re-grant it

Example app

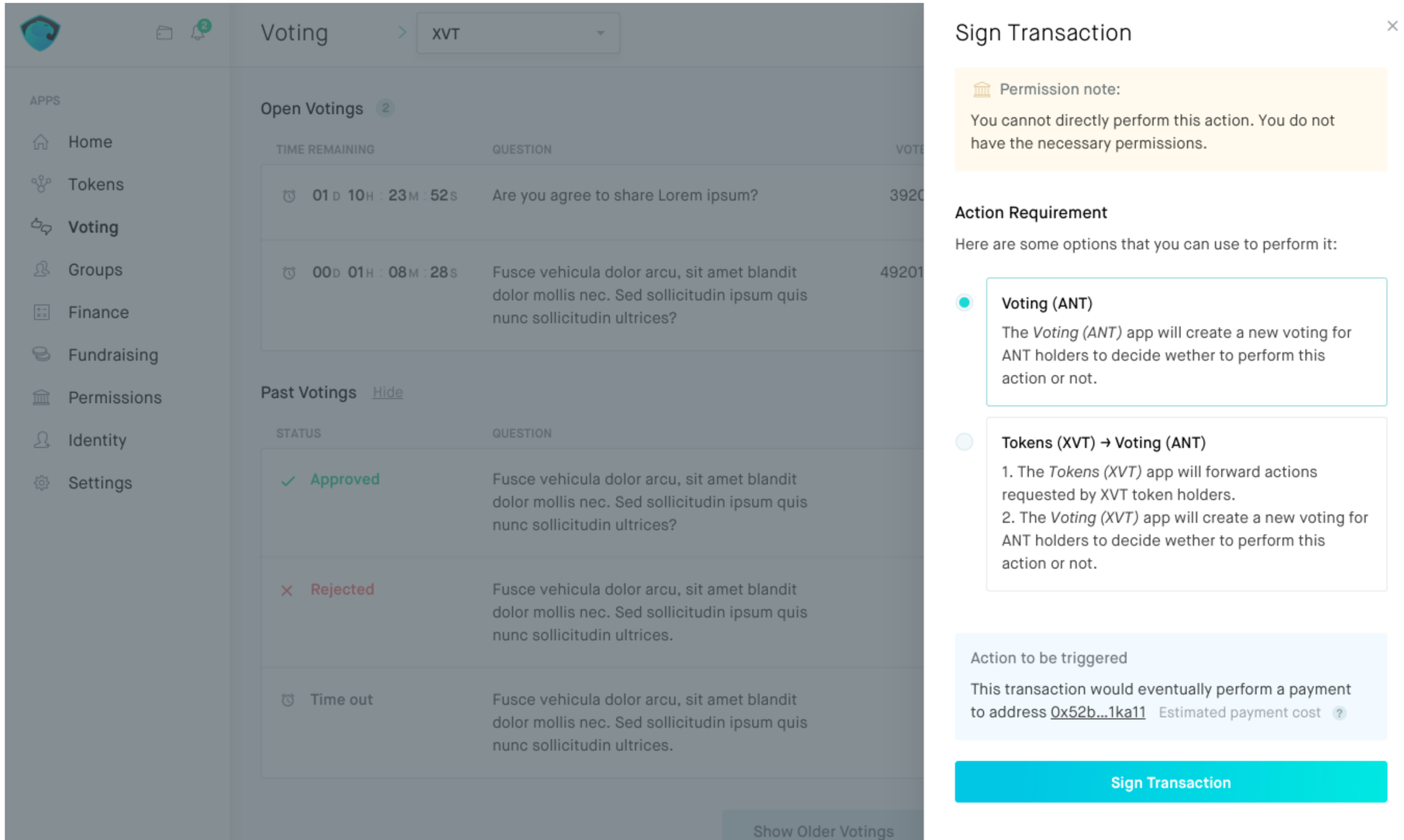
```
import "@aragon/core/contracts/apps/App.sol";

contract Counter is App {
    /// Events
    event LogIncrement();
    event LogDecrement();

    /// State
    uint256 public value;

    /// ACL
    bytes32 constant public INCREMENT_ROLE = sha3("increment");
    bytes32 constant public DECREMENT_ROLE = sha3("decrement");

    function increment() auth(INCREMENT_ROLE) external {
        value += 1;
        LogIncrement();
    }

    function decrement() auth(DECREMENT_ROLE) external {
        value -= 1;
        LogDecrement();
    }
}
```

Upgrading the app

```
import "@aragon/core/contracts/apps/App.sol";

contract Counter is App {
    /// Events
    event LogIncrement();
    event LogDecrement();

    /// State
    uint256 public value;

    /// ACL
    bytes32 constant public INCREMENT_ROLE = sha3("increment");
    bytes32 constant public DECREMENT_ROLE = sha3("decrement");

    function increment() auth(INCREMENT_ROLE) external {
        value += 1;
        LogIncrement();
    }

    function decrement() auth(DECREMENT_ROLE) external {
        value -= 1;
        LogDecrement();
    }
}
```

```
import "@aragon/core/contracts/apps/App.sol";

contract Counter is App {
    /// Events
    event LogIncrement();
    event LogDecrement();
    event LogReset();

    /// State
    uint256 public value;
    uint256 public lastReset;

    /// ACL
    bytes32 constant public INCREMENT_ROLE = sha3("increment");
    bytes32 constant public DECREMENT_ROLE = sha3("decrement");
    bytes32 constant public RESET_ROLE = sha3("reset");

    function reset() auth(RESET_ROLE) external {
        value = 0;
        lastReset = now;
        LogReset();
    }

    function increment() auth(INCREMENT_ROLE) external {
        value += 1;
        LogIncrement();
    }

    function decrement() auth(DECREMENT_ROLE) external {
        require(value > 0);
        value -= 1;
        LogDecrement();
    }
}
```

Putting everything together



Thanks!

wiki.aragon.one

github.com/aragon

aragon.chat