ethereum

vienna

Advanced Workshop
From Idea to Contract

Workshops

Workshop #1: Contract Development for Beginners

- Requirements: Basic Understanding of Ethereum
- Solidity Basics

Workshop #2: From Idea to Contract

- **Requirements: Basic Understanding of Solidity**
- **Advanced Solidity: Inheritance, Libraries**
- **Truffle, Testing EthPM**

Workshop #3: From Contract to DApp

- Requirements: Basic Understanding of Solidity, HTML/JS, node.js
- Interfacing with Ethereum using web3.js
- Auxiliary Technologies: IPFS, Whisper and Swarm

Agenda

1. Marketplace Example
2. Mapping concepts to solidity
3. Truffle & EthPM
4. Advanced Solidity
5. Solidity Testing
6. JS Testing
7. Finishing the Marketplace

Warnings

This workshop does **not** make you a contract developer

many small but important differences to other languages

=> many possible bugs (stuck contract, stolen funds, etc.)

If you ever intend to make a real world contract

- read the solidity documentation **in its entirety**
- tests ,tests, tests

Marketplace

We're building on top of the Marketplace from the beginner example

Seller can add an offer with product string and price

Buyer can take the offer by sending the right amount and offer id

Buyer can confirm delivery and release escrow

=> Seller does not get anything if buyer does not confirm!

"Advanced" Marketplace

Offers also have an arbiter

Arbiter can send ether either to creator or taker

ERC-20 compliant Reputation Token

- issued on successful trade to creator
- potentially burned if returned to taker (transfer needs to be blocked during active offer)

To avoid sybil-attacks it will be converted to a permissioned marketplace

Mapping concepts to Solidity

Contracts

Isolated from the outside world

Data can only ever enter the system through transactions

Only really "trustless" data:

- Data where the contract is authoritative
  - Ether / Token balances
  - Key / Value stores like ENS
- Mathematically verifiable facts

External Data

In most applications some trust is required

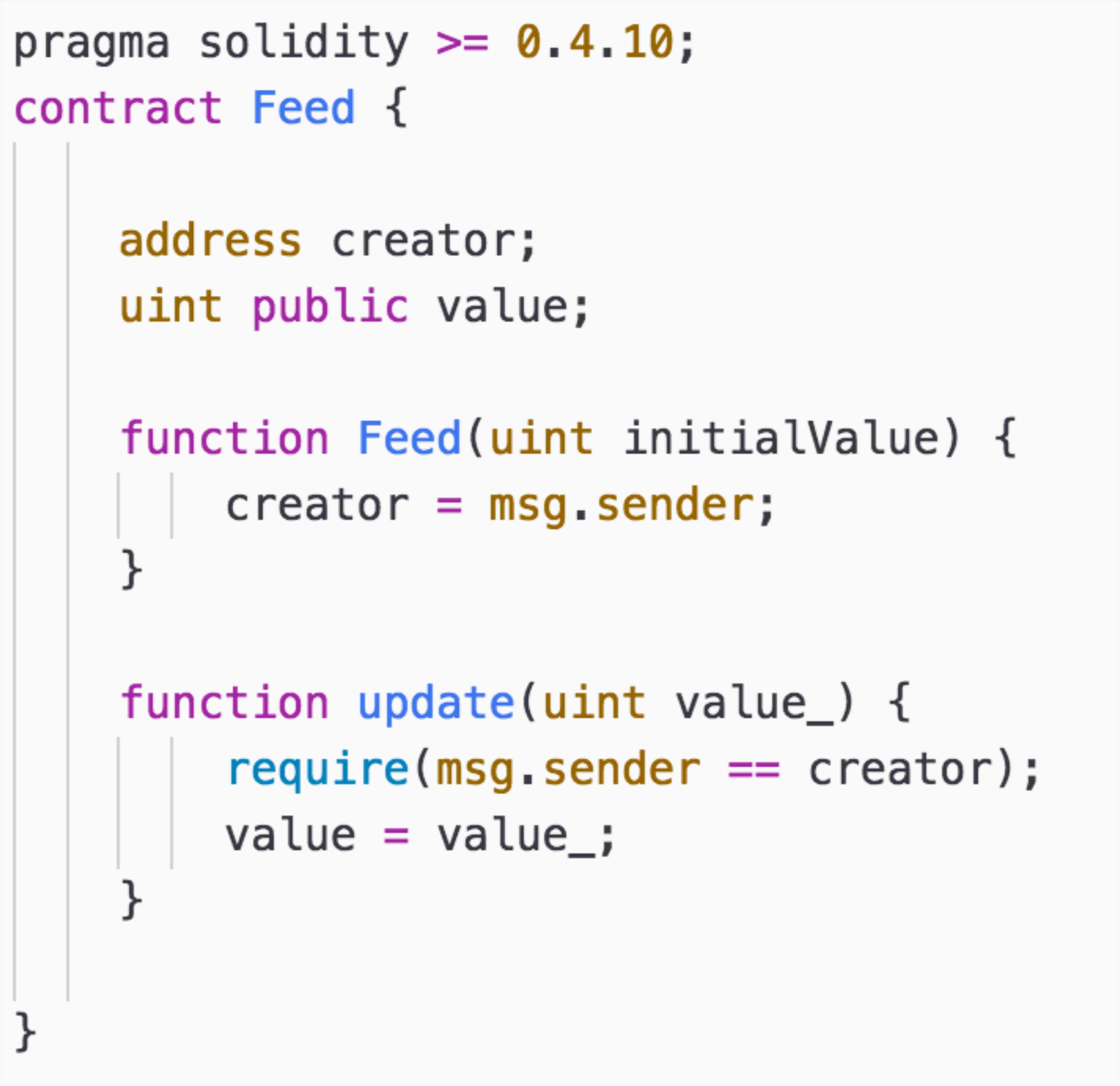
Goal: minimise required trust!

Trusted Data Feed

Creator can manipulate value at will

No cooperation between parties necessary!

=> Full trust in creator



```
pragma solidity >= 0.4.10;
contract Feed {

    address creator;
    uint public value;

    function Feed(uint initialValue) {
        creator = msg.sender;
    }

    function update(uint value_) {
        require(msg.sender == creator);
        value = value_;
    }

}
```

External Data

Multisig Trusted Data Feed

Multiple parties need to agree on a value first

- Multiple actors need to cooperate to cheat
- Still everyone has a direct connection to the contract
- also slower

External Data

Oraclize

Complicated Stuff involving TLSNotary and AWS

- Enables https request to external websites
- Amazon could tamper with the result
- Oraclize cannot (but they could withhold)
- External service can return anything

External Data

Oraclize

Basic Idea:

1. Contract calls Oraclize request
2. External servers listen to request events
3. External servers do the request
4. Proof submitted into contract
5. Proof verified in contract

External Data

Cryptlets

Programs running in Intel secure enclave environment in Azure

Intel might have the private keys

Not available yet

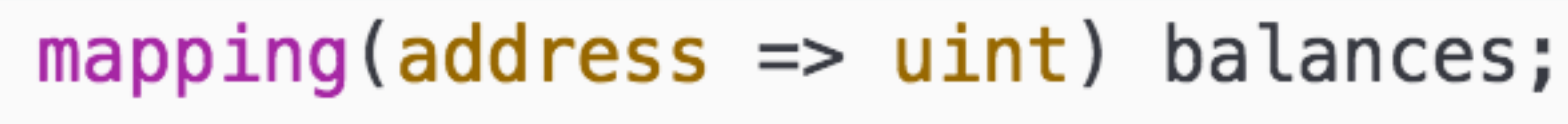
Microsoft Presentation on July 4th @RIAT

External Data

Augur

- Assumption: REP token well distributed
- Large amount of token holders vote on outcome
- Usage of mathematics to determine outliers
- Disincentive false reporting with penalties for outliers

Mappings



```
mapping(address => uint) balances;
```

When something has a non-incrementing identifier

All Fields 0 is either

- the default state for some key
- or an invalid state

cannot be enumerated (on-chain)

Ideal when you need to map from addresses or hashes to something

Arrays

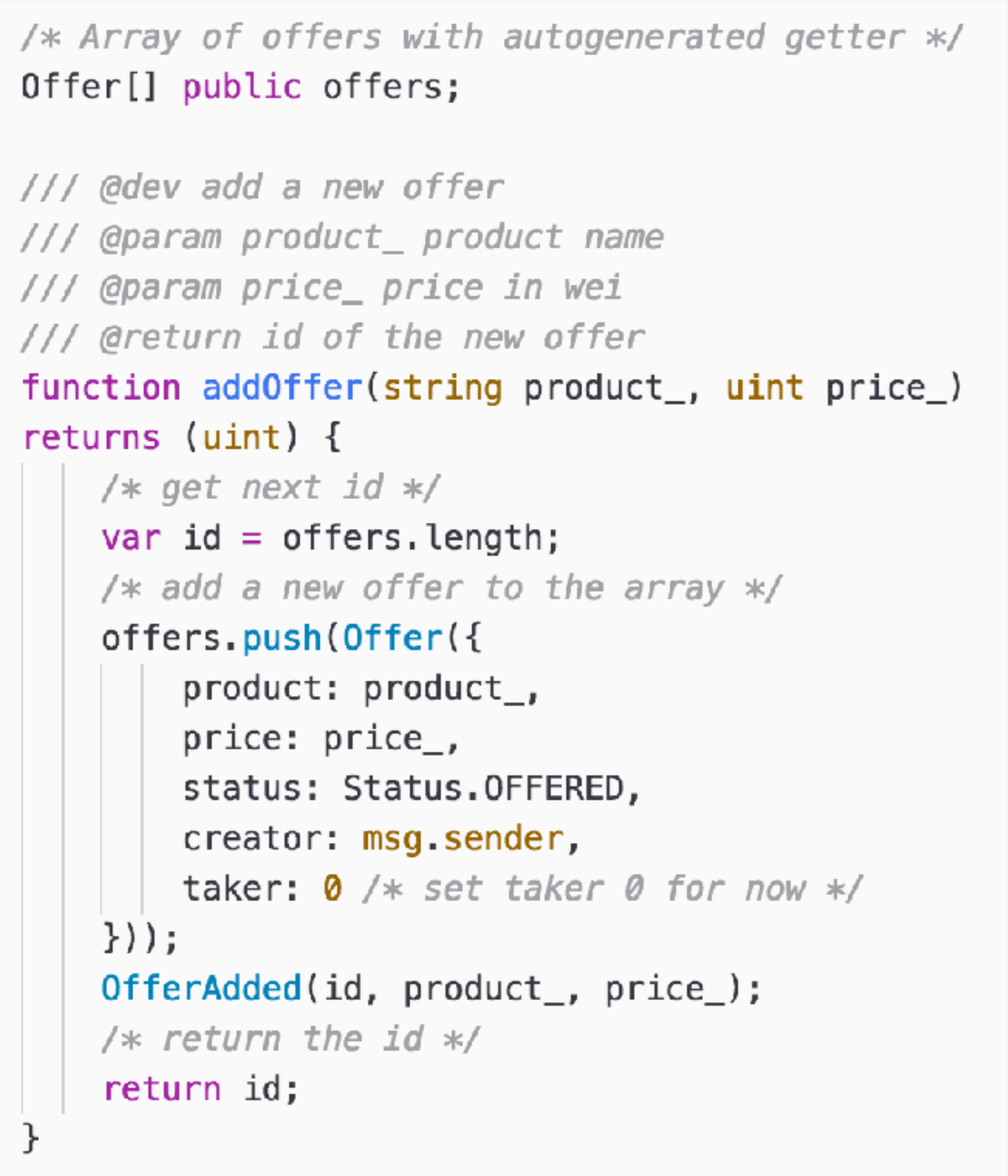
When the index can be derived from the id

Incrementing id

When you need to enumerate a collection

Array and a mapping to the same data can be useful

Mapping for quick lookup, Array for enumeration



```
/* Array of offers with autogenerated getter */
Offer[] public offers;

/// @dev add a new offer
/// @param product_ product name
/// @param price_ price in wei
/// @return id of the new offer
function addOffer(string product_, uint price_)
returns (uint) {
    /* get next id */
    var id = offers.length;
    /* add a new offer to the array */
    offers.push(Offer({
        product: product_,
        price: price_,
        status: Status.OFFERED,
        creator: msg.sender,
        taker: 0 /* set taker 0 for now */
    }));
    OfferAdded(id, product_, price_);
    /* return the id */
    return id;
}
```

Events

Everything something in the RW needs to react to

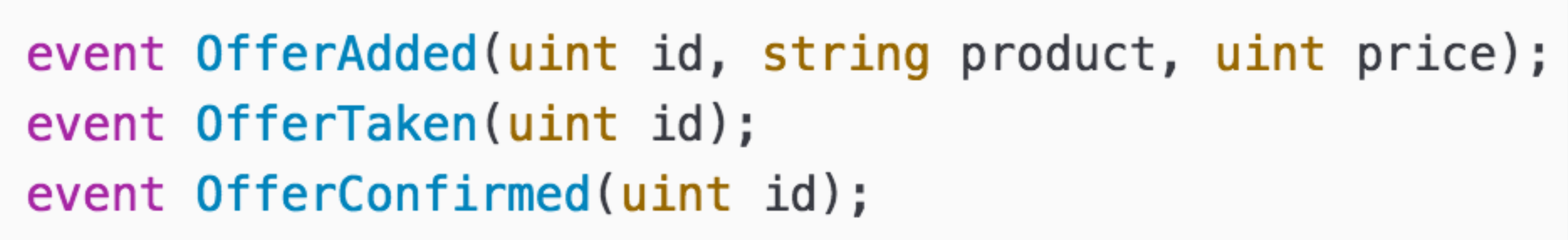
Only way to be light client friendly

If something only needs to be enumerated in RW

events can replace array (no storage cost)

=> RW actor iterates over events instead

Events



```
event OfferAdded(uint id, string product, uint price);
event OfferTaken(uint id);
event OfferConfirmed(uint id);
```

External client can get a list of offers

by iterating of OfferAdded events

by iterating over offers array

Contracts can only do the latter

Truffle

TestRPC

Virtual Ethereum node

Simulates a blockchain but without mining

Can reduce test time from minutes to seconds

Not entirely consensus-compliant

=> Also test with real clients before deployment

TestRPC

Provides a number of pre-filled accounts

Efficient snapshotting and restoring

=> used by truffle if TestRPC is detected

Special api for time travel

=> great if time needs to occur between transactions (e.g. ENS)

Truffle

Javascript-based development framework for Contracts

Most popular

Features include

- Network-dependent Deployment
- Solidity-based testing
- JS-based testing
- EthPM integration

Truffle Migrations

Basically "deployment scripts"

Contracts loaded with artifacts.require

Deployed with deployer.deploy

Also supports linking for libraries

Can also access information about the network / available accounts

EthPM

Package Manager for Ethereum Contracts

Fully decentralised

- Packages retrieved via IPFS
- Package index managed by ethereum contract

Supported by various development frameworks

Only very few packages available at this time

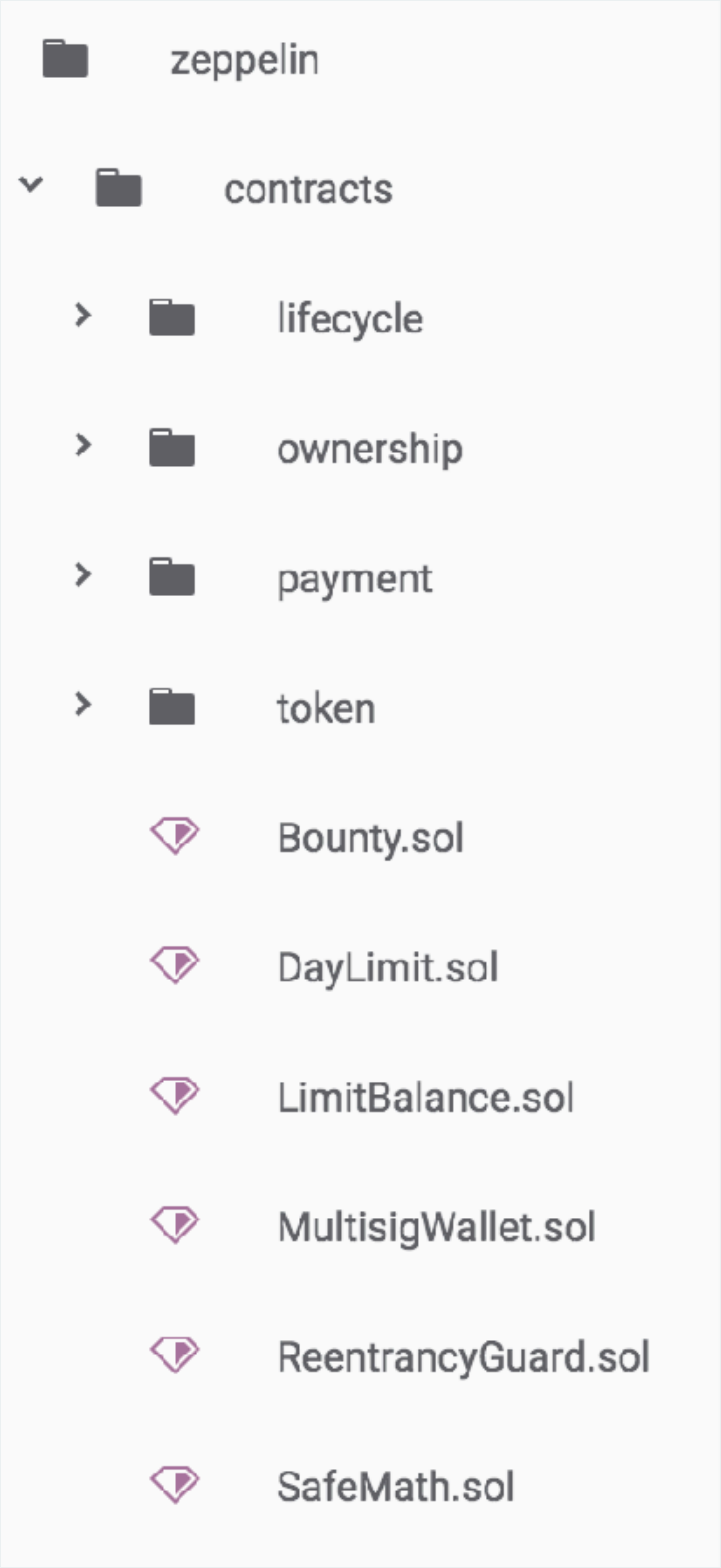
Zeppelin

Contract package developed by DCG

All Contracts come with extensive test suites

Includes base contracts for things like

- Lifecycle
- Ownership
- Tokens
- SafeMath



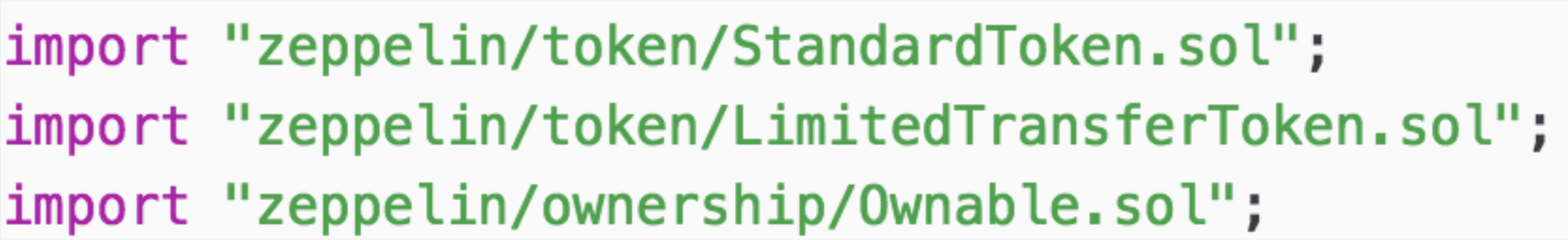
```
zeppelin
  contracts
    lifecycle
    ownership
    payment
    token
    Bounty.sol
    DayLimit.sol
    LimitBalance.sol
    MultisigWallet.sol
    ReentrancyGuard.sol
    SafeMath.sol
```

Solidity

Modules

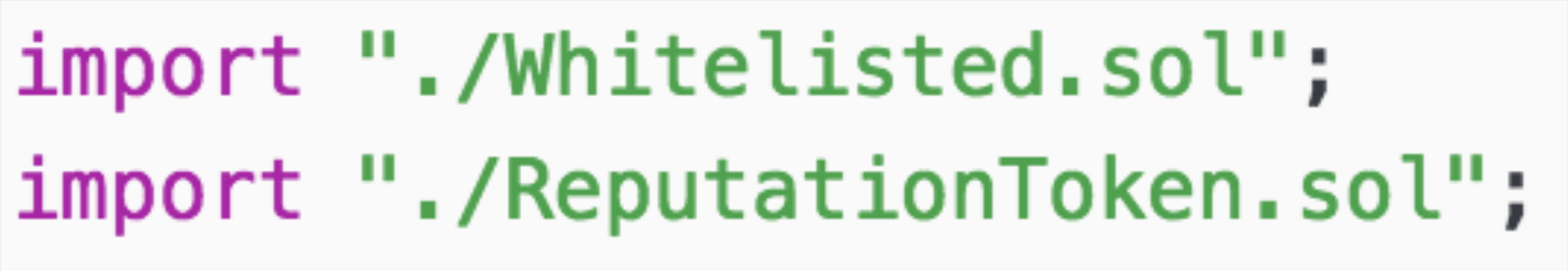
Solidity has a sophisticated module system

Importing from a package:



```
import "zeppelin/token/StandardToken.sol";
import "zeppelin/token/LimitedTransferToken.sol";
import "zeppelin/ownership/Ownable.sol";
```

Relative Import:



```
import "./Whitelisted.sol";
import "./ReputationToken.sol";
```

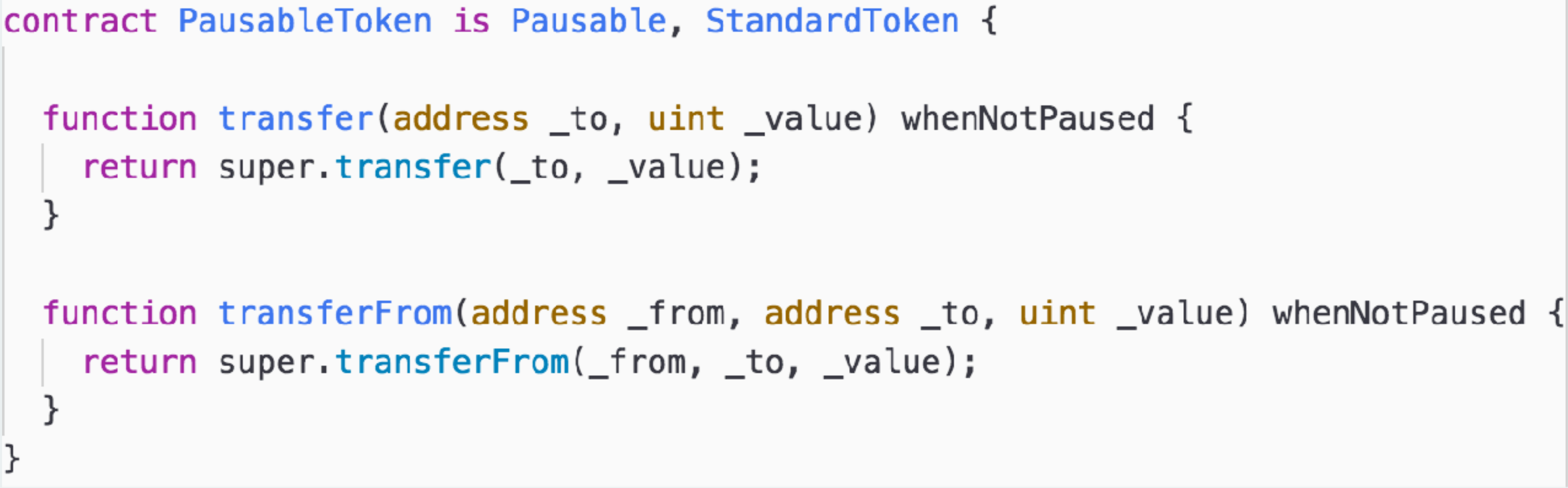
Many different ways of importing, see the official documentation!

Inheritance

Contract can inherit from another contract

Code of the ancestor contract copied into child

Multiple inheritance also possible!



```
contract PausableToken is Pausable, StandardToken {

  function transfer(address _to, uint _value) whenNotPaused {
    return super.transfer(_to, _value);
  }

  function transferFrom(address _from, address _to, uint _value) whenNotPaused {
    return super.transferFrom(_from, _to, _value);
  }
}
```

ERC-20 Tokens

transfer	Sends tokens to another accounts
balanceOf	gets balance of some account
totalSupply	gets total supply of token
approve	Approve that another account may spend some amount of tokens
transferFrom	Sends tokens from another accounts to another account (if approved)

Other standards

ERC-20 is the only "real" standard

Other proposed standards (with some usage)

- ENS Resolvers
- Token Registry

Libraries

Reusable pieces of code

Can have internal functions which are copied into a contract (like inheritance)

Can have external functions, which live in a different contract

=> Deployment time linking necessary!

Can be attached to types with the "using for" syntax

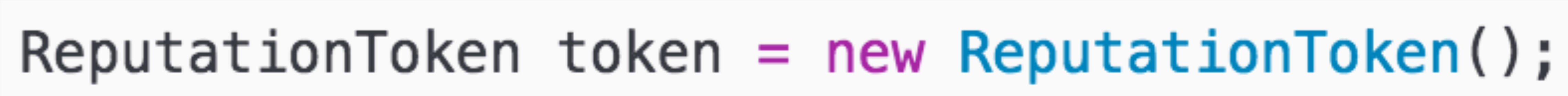
Address Interaction

call	sends a message with gas (.call in solidity, also used for cross contract function calls)
send	call with 0 gas (stipend only, used for address.send and address.transfer)
callcode	call code in the current context (effectively useless)
delegatecall	like callcode, but preserves sender and value (used for libraries)
create	creates contract and runs init code ("new" keyword in solidity)

Contract Interaction

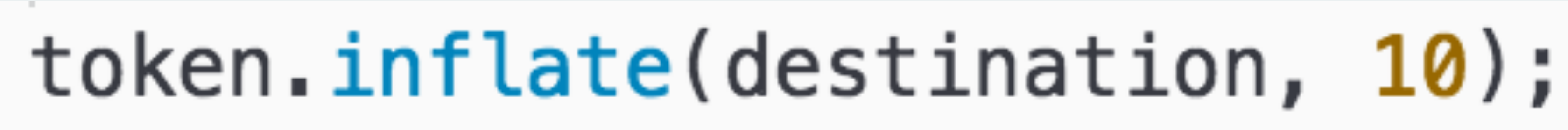
Solidity type for every contract type that's in scope

Create new contracts with new:



```
ReputationToken token = new ReputationToken();
```

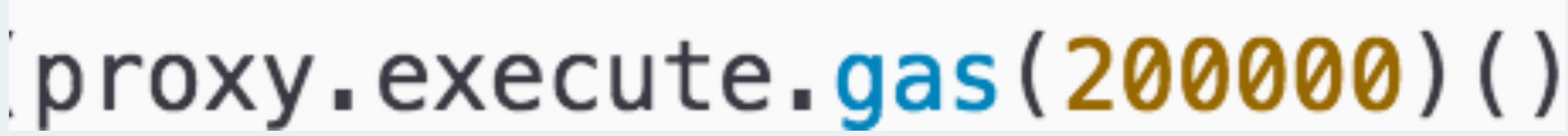
Call function on another contract:



```
token.inflate(destination, 10);
```

=> if inflate runs out of gas, so does the calling contract!

Call function with custom gas:



```
proxy.execute.gas(200000)()
```

Testing

Test Scenarios

All the regular stuff:

- All use cases
- Unauthorised access
- Wrong usage

Different gas limits (if you use explicit gas limits somewhere)

Test Scenarios

Contract Reentrance

During **any untrusted call** you contract can be called again (unless low gaslimit)

After **any untrusted call**, all assumptions about the state of the contract invalid

=> If you use a prevention, make sure it actually works!

Solidity Testing

Every test suite is a contract

Individual tests are solidity functions

Truffle comes with an assertion library

You can either deploy contracts in the test

Or access the ones deployed by the migrations

Solidity Testing Limitations

Current solidity testing has some issues:

- Functions with dynamic length return arguments can't be tested (e.g. strings) *
- Testing function for throws is very clumsy *
- Sending from different accounts is not ideal either
- Events cannot be tested
- Tests that need different timestamps, block.number, tx.origin, etc.

* due to EVM limitations which will be resolved in Metropolis

Solidity Testing with Dapple

Dapple is another development framework

Also has solidity based unit tests

- Can test for events (kind of)
- Can test for throws (but also with some drawbacks)
- Has some inbuilt support for easy proxies

JS Testing

Based on mocha + chai

Uses the truffle contract abstractions for a Promise-based API

exposes "contract" function: like describe but with redeploy

artifacts.require to load contracts and their information

JS async / await

Even with Promises tests are mostly clutter!

Recent versions of node ship with async/await by default

=> makes tests much more readable

Coding Part

1N7wgGE2eeMpiDS6LFbSxypsVbHo4H3Spv

The End

0xe9b0d93a7514d619b5eea66b7bacf665a69320e6
riatspace.eth