ethereum vienna

Workshop
Contract Development for Beginners

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

Mapping the real world to ethereum concepts

Advanced Solidity / Testing

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

Resources

workshop.zip

- **resources.txt**: useful links
- **specifications**: detailed description of exercises
- **scaffolding**: optional starting point for exercises
- **solutions**: self-explanatory
- **presentation.pdf**: this presentation
- **crowdfund.sol**: (bad) example contract
- **remix-intro.pdf**: "guide" to Remix IDE

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 (!!!)

audits (multiple!!!)

Agenda

1. EVM Fundamentals
2. Remix IDE
3. Intro to Solidity
4. **First Exercise: Trusted Data Feed**
5. More Solidity
6. **Exercise: Advanced Feed**
7. Example: Subscription
8. Solidity Data Structures
9. **Final Exercise: Implementing a marketplace**

EVM Fundamentals

Fundamentals

Accounts

- 160 bit addresses
- hold ether
- hold state
- can have controlling private key
- or EVM bytecode => contract

ethereum Fundamentals

Contract

Runs at every received message

Has a persistent 256-to-256 bit storage

private (to other contracts, public to external actors)

but expensive

Can spawn new messages during execution

(to send ether or just call other contracts)

Fundamentals

Example Crowdfunding:

Storage used to store:

contribution information

campaign info

campaign progress

```
address public beneficiary;
uint public fundingGoal;
uint public amountRaised; uint public deadline;
```

Fundamentals

Example Crowdfunding:

New messages sent for:

paying back funders

paying out the funds

manage token

```
if (!beneficiary.send(amountRaised)) throw;
```

```
tokenReward.sendCoin(msg.sender, amount / price);
```

Example

Forwarder

Simply forwards received funds to another address

Scenario:

A sends 10 ether to Forwarder

encodes argument "forward(B)"

Forwarder forwards 10 ether to B

ethereum

Accounts

A
100 ETH

B
0 ETH

Contract
0 ETH

code:
0x…..

Fundamentals

Message (or "internal Transaction")

- Sender (where the ether is sent from)
- Recipient (e.g. the executing contract)
- Value (can be 0)
- Data (used to encode the function call)
- Return Value (used to retrieve the result of a computation)
- Gaslimit (the maximal gas usage local to this message)

Executes either completely or not at all

ethereum

Message

A
99 ETH

B
0 ETH

from: A
to: Contract
value: 1ETH
data: forward(B)

Contract
1 ETH

code:
0x…..

causes contract to be executed

ethereum

Message

A
99 ETH

from: A
to: Contract
value: 1ETH
data: forward(B)

Contract
0 ETH

code:
0x…..

execution can cause new messages

B
1 ETH

from: Contract
to: B
value: 1ETH

causes contract to be executed

Fundamentals

Transaction

wraps a message

signed by a private key

only transactions appear in chain

sets gasprice for all contained messages

sets a global gaslimit

ethereum

Blockchain

A
100 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Signed by A

Message
30000 gas

Contract
0 ETH

B
0 ETH

Message
10000 gas

ethereum

Blockchain

A
90 ETH

-10 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Message
30000 gas

Contract
0 ETH

B
0 ETH

Message
10000 gas

ethereum

Blockchain

A
89 ETH

-10 ETH
-1 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Gas: 100000

Message
30000 gas

Contract
0 ETH

100000 * 0.00001 = 1

B
0 ETH

Message
10000 gas

ethereum

Blockchain

A
89 ETH

-10 ETH
-1 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Message
30000 gas

Contract
0 ETH

Gas: 79000

B
0 ETH

Message
10000 gas

ethereum

Blockchain

A
89 ETH

-10 ETH
-1 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Gas: 59000

Message
20000 / 30000 gas

Contract
10 ETH

+10 ETH

B
0 ETH

Message
10000 gas

ethereum

Blockchain

A
89 ETH

-10 ETH
-1 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Message
20000 / 30000 gas

Gas: 49000

Contract
0 ETH

+10 ETH
-10 ETH

B
10 ETH

+10 ETH

Message
10000 gas

ethereum

Blockchain

A
89 ETH

-10 ETH
-1 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Gas: 39000

Message
10000/30000 gas

Contract
0 ETH

+10 ETH
-10 ETH

Message
10000 gas

B
10 ETH

+10 ETH

ethereum

Blockchain

A
89 ETH

-10 ETH
-1 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Message
30000 gas

Gas: 39000
~ 0.39 ETH

Contract
0 ETH

+10 ETH
-10 ETH

B
10 ETH

+10 ETH

Message
10000 gas

ethereum

Blockchain

A
89.39 ETH

-10 ETH
-1 ETH
+0.39 ETH

Gasprice:
0.00001

Gaslimit:
100000

Transaction
21000 gas
(61000)
(Eth Cost: 2.1)

Message
30000 gas

Gas: 0
~ 0 ETH

Contract
0 ETH

+10 ETH
-10 ETH

B
10 ETH

+10 ETH

Message
10000 gas

ethereum

EVM

Stack machine

256 bit words

Has all the usual instructions plus

block data, tx data, msg data, contract data access

cryptographic functions

message sending

ethereum EVM

Storage

- expensive
- persistent

Memory (only during execution)

- cheaper
- byte-level access

Stack (only during execution)

- inaccessible in solidity (except assembly)

ethereum EVM

Out of gas exception:

If a message runs out of gas

- all state changes are reversed
- includes transfers, storage modifications, events

gas is not reversed (and neither are selfdestructs(!))

parent message runs afterwards (but might also be oog)

ethereum

EVM

Logs

for UIs

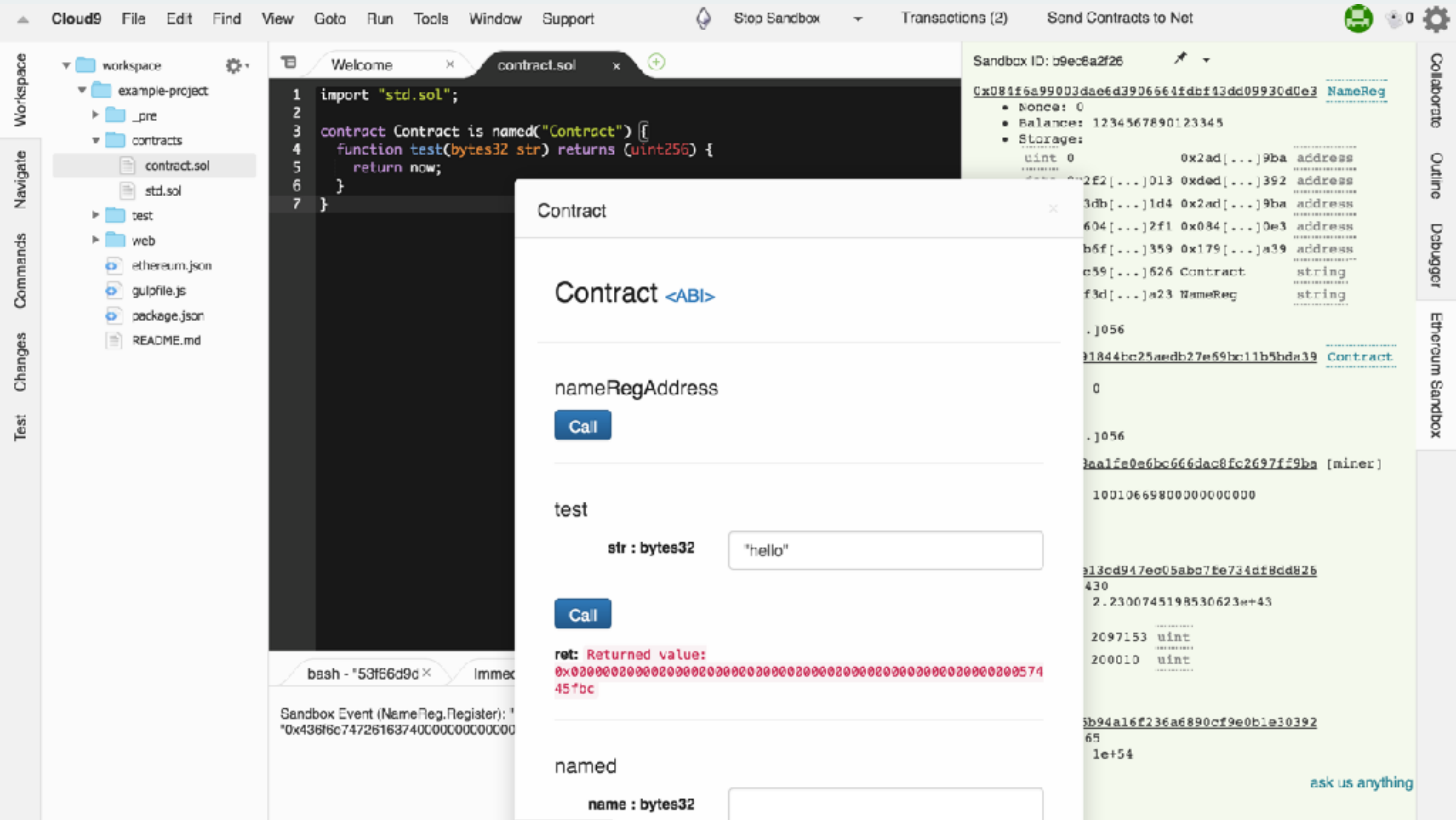
Light Clients

Logging

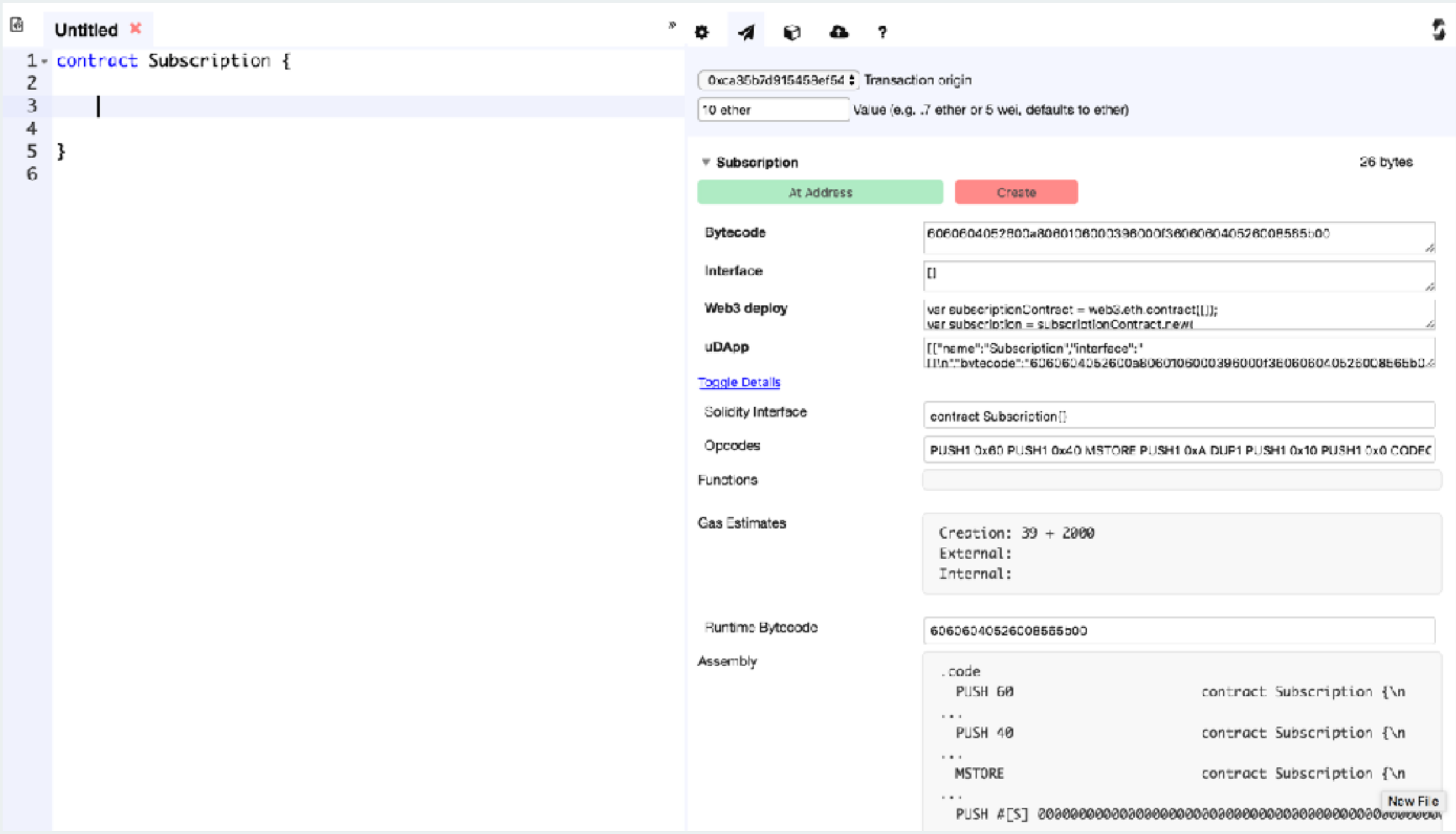
Fill a bloom filter in block header

IDEs

ethereum Ethereum Studio



ethereum Remix IDE



Solidity

ethereum

Solidity

C/Java/JS-like syntax

Compiler: solc

Available debuggers: Remix, Ethereum Studio

This workshop: solc 0.4.10 (latest)

Reference at: https://solidity.readthedocs.io/

ethereum Solidity

Developer writes contract with functions

Compiler generates

init code

dispatcher

At deployment the

contract constructor

is executed

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

    uint value;

    function Sample(uint initial) {
        set(initial);
    }

    function set(uint v) {
        value = v;
    }

    function get() returns (uint) {
        return value;
    }

}
```

Solidity

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

compiler version

starts a contract block

contract name

unsigned int 256 bit

type

```
    uint value;
```

variable in contract storage

initialised to 0 by default

variable name

ethereum Solidity

function name argument type name

```
function Sample(uint initial) {
    set(initial);
}
```

function call of set with argument inital

Function with same name as contract = constructor
Runs once at deployment

ethereum Solidity

```
function set(uint v) {
    value = v;
}
```

sets the storage of the variable value to v

return value type

```
function get() returns (uint) {
    return value;
}
```

terminates function and returns value

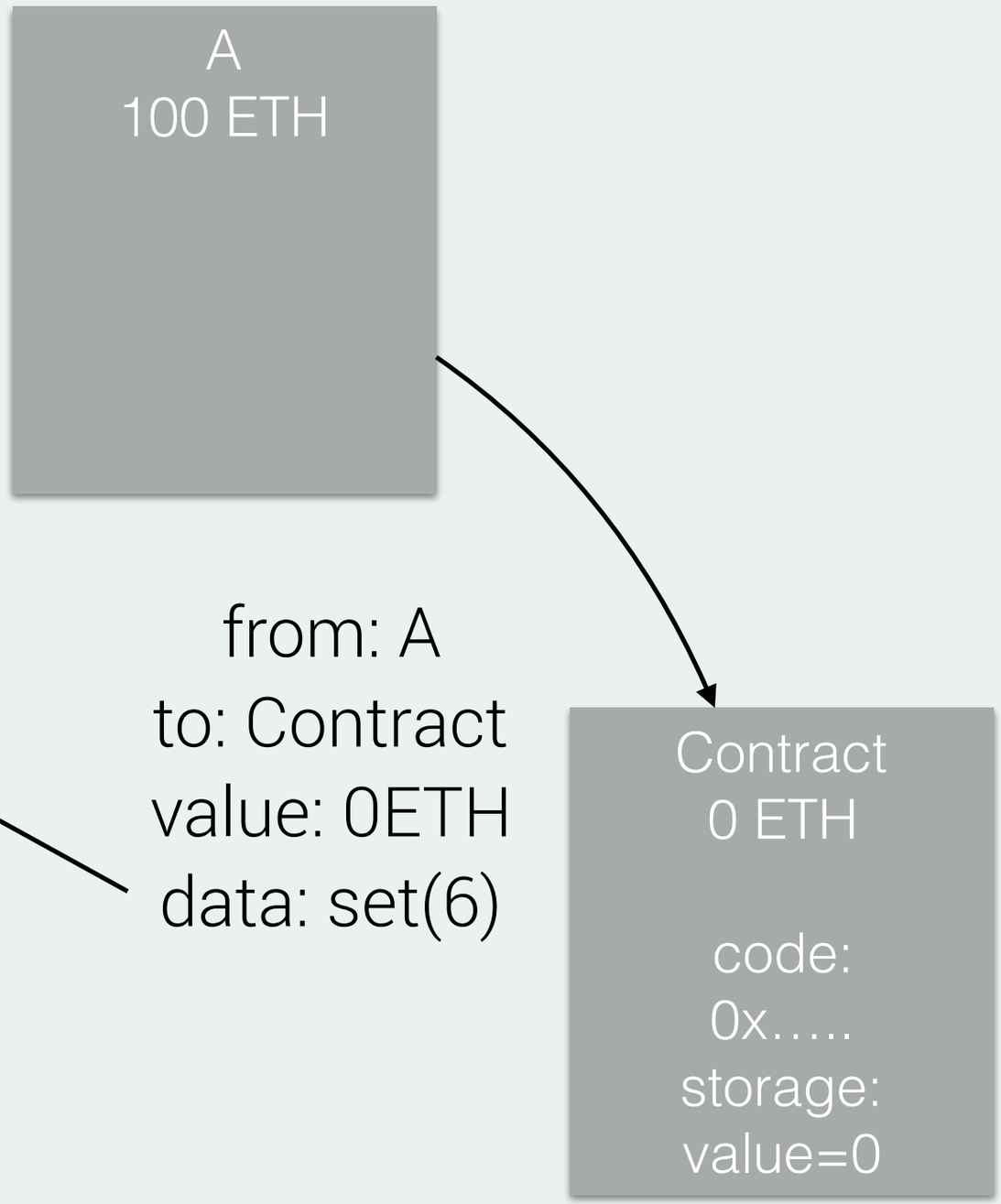
modifier code might still run (!)

```
}
```

ethereum

Message

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

    uint value;

    function Sample(uint initial) {
        set(initial);
    }

    function set(uint v) {
        value = v;
    }

    function get() returns (uint) {
        return value;
    }

}
```

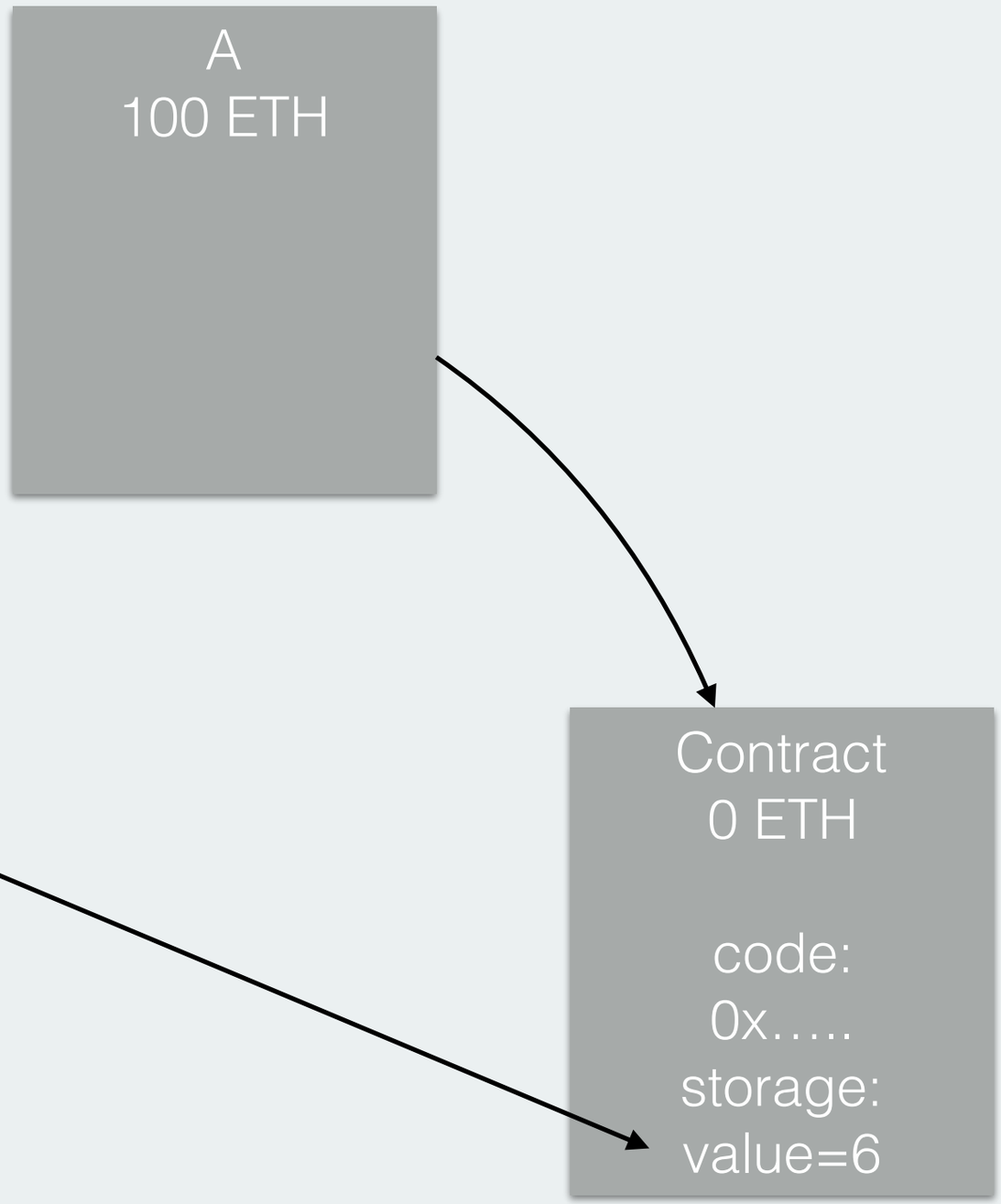


* gas cost omitted for simplicity

ethereum

Message

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

    uint value;

    function Sample(uint initial) {
        set(initial);
    }

    function set(uint v) {
        value = v;
    }

    function get() returns (uint) {
        return value;
    }

}
```

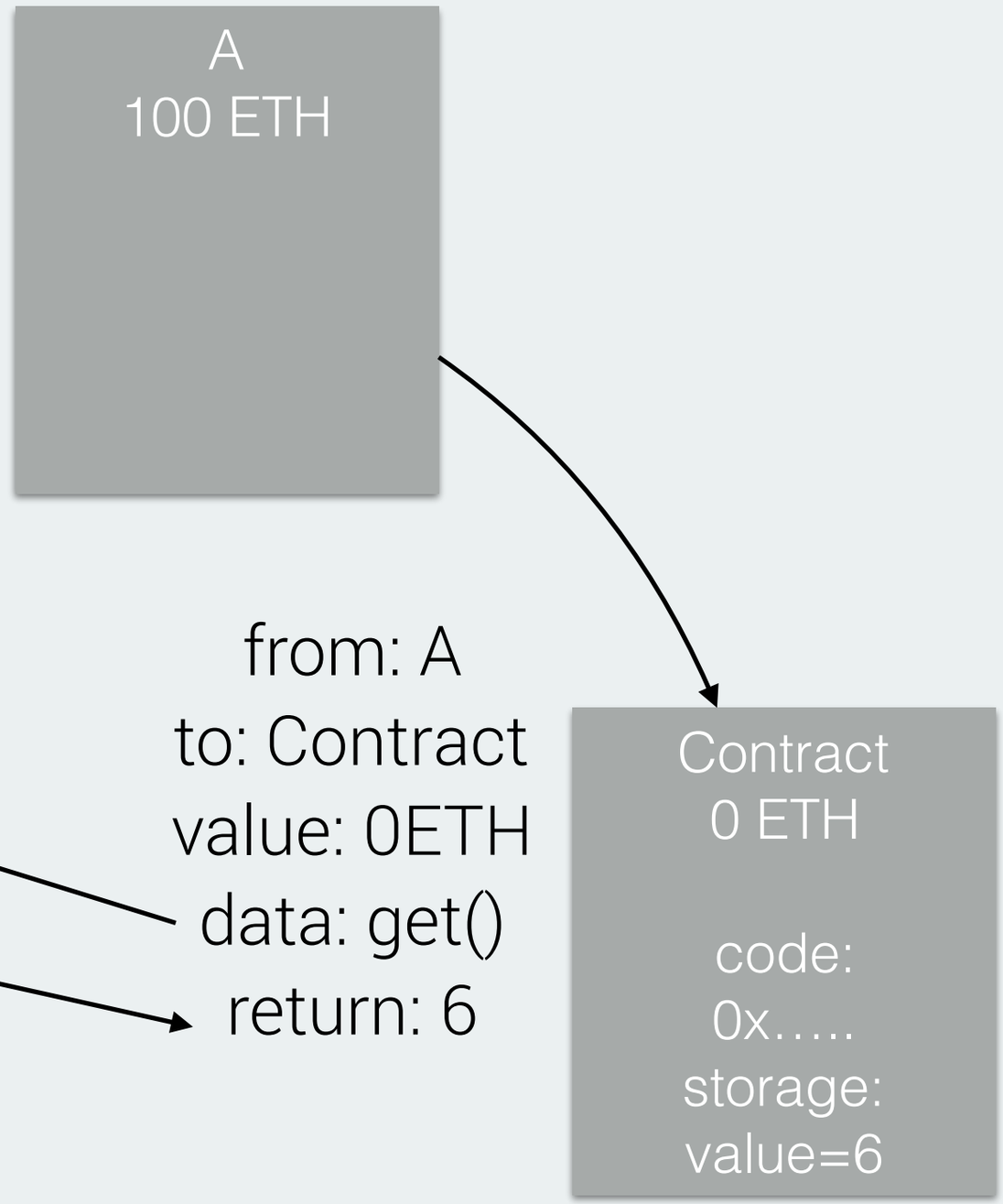


* gas cost omitted for simplicity

ethereum

Message

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

    uint value;

    function Sample(uint initial) {
        set(initial);
    }

    function set(uint v) {
        value = v;
    }

    function get() returns (uint) {
        return value;
    }

}
```



* gas cost omitted for simplicity

Crowdfunding

Simple old contract by Vitalik

modified for solidity 0.4.10

uses almost all features covered in this workshop

has some design issues

=> security issues

=> good example of what not to do

Types

“Standard” types:

bool

int: Signed 256 bit Integer (other sizes available)

uint: Unsigned 256 bit Integer (other sizes available)

Array: Static and Dynamic

String (Unicode)

Enum

ethereum Types

Special types:

Address: 160 bit for ethereum address

Fields: **balance**

Functions: **send**, call, callcode, delegatecall

Mapping (hashtable-like):

maps from one solidity type to another

contains all keys at construction

Contract Types:

Inherits from address

Contract-specific functions

```
contract Sample {
  address a = 0x3049280948ffa0afafaffffffffafffff9789372;
  mapping (address => uint) balances;
  Sample otherContract;
}
```

Control Flow

If

```
function f (uint x) returns (uint) {
    if (x > 5) {
        return 3;
    } else {
        return 4;
    }
}
```

ethereum Control Flow

For (can be very dangerous)

```
for (uint a = 0; a < 99; a++) {

}
```

While / Break

```
while(true) {
  break;
}
```

Control Flow

Crowdfund: Refund in case of failed campaign

```
for (uint i = 0; i < funders.length; ++i) {
    var funder = funders[i];
    funder.addr.transfer(funder.amount);
    FundTransfer(funder.addr, funder.amount, false);
}
```

ethereum Control Flow

Crowdfund: Refund in case of failed campaign

unbounded for-loop

```
for (uint i = 0; i < funders.length; ++i) {
    var funder = funders[i];
    funder.addr.transfer(funder.amount);~21k gas
    FundTransfer(funder.addr, funder.amount, false);
}
```

200 participants = > 4.2m gas, above current global gaslimit

this

this.balance: gets the balance of the **executing** contract

this.function(): calls a function by message

you cannot access storage variables with this!

ethereum Public Variables

declare a variable as public

-> Automatic getter generation (no setter!)

```
contract Sample {

    uint public value;

    function Sample(uint initial) {
        set(initial);
    }

    function set(uint v) {
        value = v;
    }

}
```

generates value() function

Globals

msg.

sender: immediate caller of the function

value: wei sent in the current message

gas: remaining gas available for the current message

tx. (shared by all messages)

origin: original creator of the transaction

gasprice: global gasprice

Globals

block.

coinbase: miner of the block

difficulty

timestamp: in unix time (solidity also has synonym "now")

blockhash

number: number of blocks since genesis

Special cryptographic functions (e.g. sha3)

ethereum Globals

```
contract Sample {

    uint public value;
    uint public timestamp;
    address public setter;
    uint public burn;

    function Sample(uint initial) {
        set(initial);
    }

    function set(uint v) payable {
        value = v;
        timestamp = block.timestamp;
        setter = msg.sender;
        burn = msg.value;
    }

}
```

ethereum

Events for writing to the log (light clients, UIs, etc.)

```
contract Sample {
    event GotWei(uint amount);

    function () payable {
        GotWei(msg.value);
    }

}
```

like functions
but with event keyword
no body

Exercise #1

Trusted data feed

Contains only one readable integer

Can only be changed by the creator

Change Event

Field can be read by other contracts

relevant globals: msg.sender

hint:

many similarities to Sample

creator is sender in constructor

Modifier

Modifiers for code reuse

```
modifier afterDeadline() { if (now >= deadline) _; }

/* checks if the goal or time limit has been reached and ends the campaign */
function checkGoalReached() afterDeadline {
    if (amountRaised >= fundingGoal){
        // sends amountRaised wei to beneficiary account
        if (!beneficiary.send(amountRaised)) throw;
        FundTransfer(beneficiary, amountRaised, false);
    } else {
```

_; replaced by function body

modifier can still run after return!

Modifier

```
contract Sample {

    uint public value;

    modifier mod {
        _;
        value = 10;
    }

    function test() mod returns(bool) {
        value = 5;
        return true;
    }

}
```

after test()
value is 10
not 5

throw

creates an exception (invalid opcode)

execution aborts, state reverts

cannot be caught on contract functions

all gas is used (alternative "revert" coming soon)

```
/* throw if the offer has already been taken */
if(offer.status != Status.OFFERED) throw;
/* throw if the sent value does not match the offer */
if(msg.value != offer.price) throw;
```

require

creates an exception if condition is not met

new in solidity 0.4.10

require(x); = if(!x) throw;

```
/* throw if the offer has already been taken */
require(offer.status == Status.OFFERED);
/* throw if the sent value does not match the offer */
require(msg.value == offer.price);
```

Sending Ether

Every address or contract object

has a send method, takes the amount in wei

```
function doSomething() {
    address recipient = 0x0;
    uint amount = 50 ether;

    var success = recipient.send(amount);
    if(!success) throw;
    if(!recipient.send(amount)) throw;
}
```

returns false if message does not succeed (does not throw!)

ethereum

Transfer Ether

transfer method, takes the amount in wei

new in solidity 0.4.10

addr.transfer(x); = if(!addr.send(x)) throw;

```
function doSomething() {
    address recipient = 0x0;
    uint amount = 50 ether;

    recipient.transfer(amount);
}
```

Transfer Ether

Crowdfund: Refund in case of failed campaign

```
for (uint i = 0; i < funders.length; ++i) {
    var funder = funders[i];
    funder.addr.transfer(funder.amount);
    FundTransfer(funder.addr, funder.amount, false);
}
```

Transfer Ether

Crowdfund: Refund in case of failed campaign

```
for (uint i = 0; i < funders.length; ++i) {
    var funder = funders[i];
    funder.addr.transfer(funder.amount); might throw!
    FundTransfer(funder.addr, funder.amount, false);
}
```

One bad actor can block all refunds!

Transfer Ether

Crowdfund: Refund in case of failed campaign

```
for (uint i = 0; i < funders.length; ++i) {
    var funder = funders[i];
    funder.addr.send(funder.amount);
    FundTransfer(funder.addr, funder.amount, false);
}
```

Transfer Ether

Crowdfund: Refund in case of failed campaign

```
for (uint i = 0; i < funders.length; ++i) {
    var funder = funders[i];
    funder.addr.send(funder.amount); does not throw
    FundTransfer(funder.addr, funder.amount, false);
}
```

Refunds could be lost forever

ethereum Receiving Ether

```
contract Forwarder {
    function forward(address recipient) payable {
        recipient.transfer(msg.value);
    }

    /* default function */
    function() payable {}
}
```

Functions reject ether by default

If a function can be called with ether

explicit modifier **payable** necessary!

Exercise #1.5

Trusted data feed

Contains only one field

Can only be changed by the creator

Change Event

Field can be read by other contracts **(for a fee)**

Fee forwarded to creator

relevant globals: msg.value, throw

Example

Subscription Contract

Manages **one** subscription

Recipient: can withdraw PRICE wei per TIME

Creator: can cancel if there are not outstanding payments

relevant:

address.send(value): send value wei to address

block.timestamp: unix timestamp (in seconds)

ethereum Contract Calls

Coerce address into contract type

Call the function on that

.value() to send wei

.gas() to limit gas

```
token public tokenReward;
Funder[] public funders;
mapping (address => bool) public
```

```
// Coerce an address into a contract type
tokenReward = token(_reward);
```

```
// sends a sendCoin message to the tokenReward contract
tokenReward.sendCoin(msg.sender, amount / price);
```

```
tokenReward.sendCoin.value(10).gas(1000)(msg.sender, amount / price);
```

Warning: Recursion possible!

ethereum Structs

```
/* data structure to hold information about campaign contributors */
struct Funder {
    address addr;
    uint amount;
}
```

```
// push an additional value onto the array
var funder = Funder({addr: msg.sender, amount: amount});
```

```
if (!funder.addr.send(funder.amount)) throw; /* P
FundTransfer(funder.addr, funder.amount, false);
```

Arrays

```
Funder[] public funders;
```

dynamically sized array (starting with index 0)

push: adds a new element to the array

```
funders.push(Funder({addr: msg.sender, amount: amount}));
```

get element at index i

```
var funder = funders[i];
```

number of elements:

`funders.length` == index of the next pushed element

Solidity

functions can have multiple return values

retrieve values by deconstruction

```
function return2Values() returns (uint a, bool b) {
    a = 9;
    b = false;
}

function callThatFunction() {
    var (a,b) = return2Values();
}
```

ethereum Visibility

External

Can only be called by a message

Public (default)

Can be called by anyone

```
function f() private { }
function g() public { }
function h() external { }
function i() internal { }
```

Private

Can only be called by the contract itself

Internal

Cannot be called by a message

Enums

```
/* Status enum for the 3 possible states */
enum Status { OFFERED, TAKEN, CONFIRMED}
```

```
/* set status to confirmed */
offer.status = Status.CONFIRMED;
```

```
/* throw if offer is not taken */
if(offer.status != Status.TAKEN) throw;
```

"Advanced"

EPM - Ethereum Package Manager

Many dev frameworks (e.g. truffle)

JS / solidity based unit testing

selfdestruct - potentially dangerous (!)

"Advanced"

Import other files

Contract inheritance

Code from ancestor copied into child

Still only one contract

and much more...

Exercise #3

Market Contract

Seller can add offers (with name and price)

Buyer can take offers (by sending the right amount)

Buyer can confirm the offer (and release funds)

1CkG85YbRrvahhU8ChigoQHCP9qw98MtY8

The End

0x8f8cf4c20ae44b5ca2c4a0523499b42844a3d28c