



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



ethereum 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



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

Logs the transfer



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)

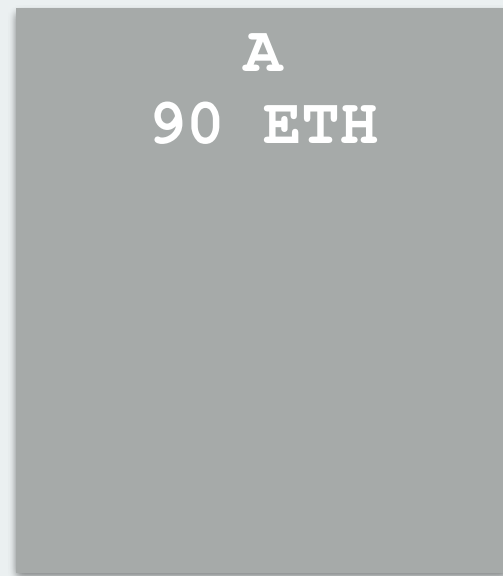
Gaslimit (the maximal gas usage local to this message)

Executes either completely or not at all

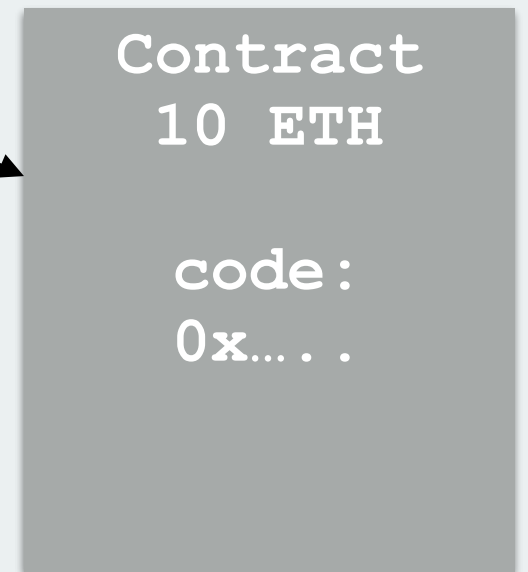


ethereum

Message



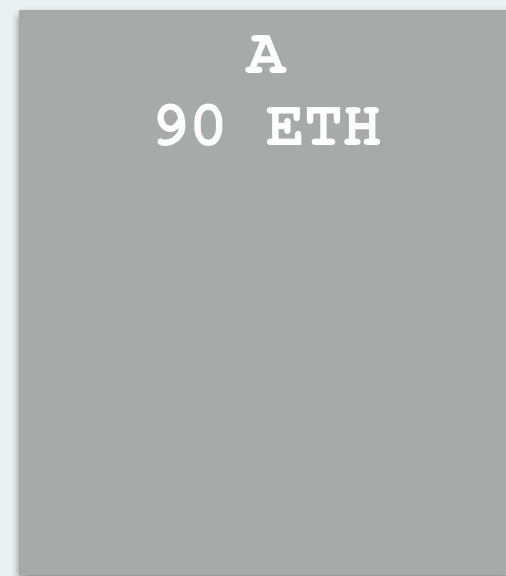
from: A
to: Contract
value: 10 ETH
data: forward(B)



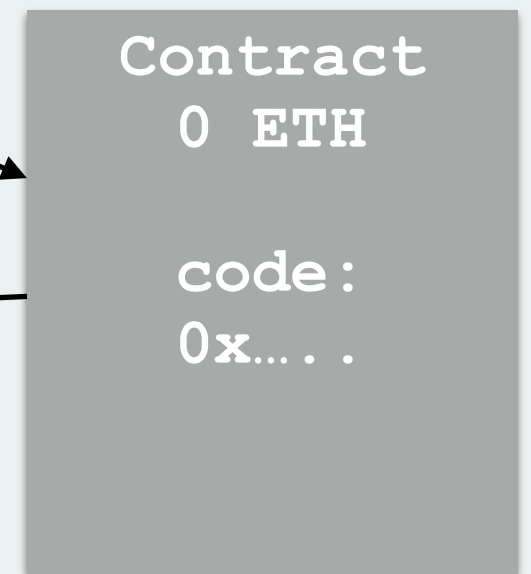
causes contract to be executed



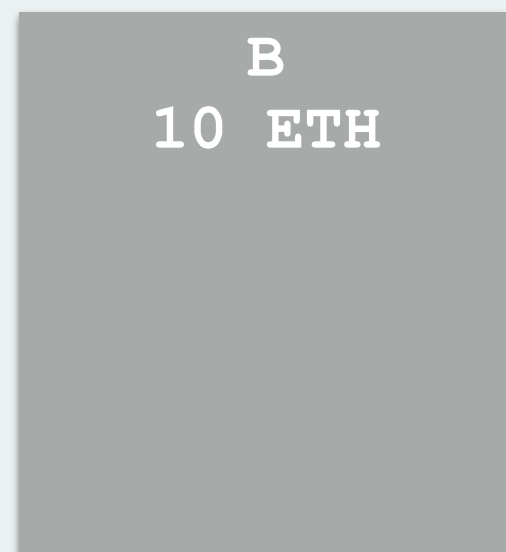
Message



from: A
to: Contract
value: 10 ETH
data: forward(B)



execution can cause new messages



from: Contract
to: B
value: 10 ETH

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

gas: 0

Transaction start

A
100 ETH

Transaction	
to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Contract
0 ETH

DECODE ARGS
SEND
LOG
RETURN

Signed by A

gas: 100000
+100000

Purchase of gas

A

99 ETH

-1 ETH

gas * gasprice

Transaction

to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Signed by A

Contract

0 ETH

DECODE ARGS

SEND

LOG

RETURN

gas: 79000
-21000

Base transaction cost

A
99 ETH

Transaction	
to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

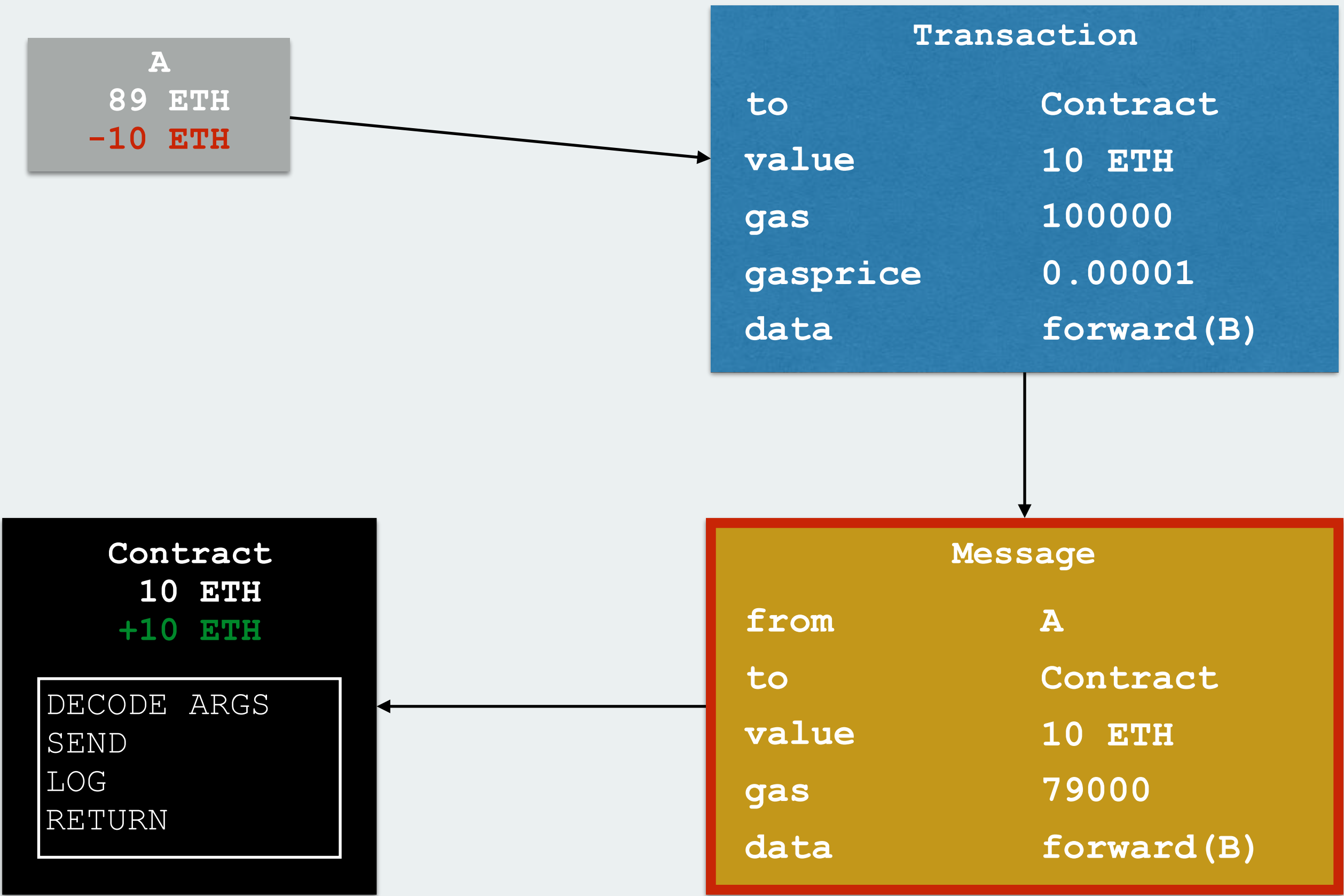
Contract
0 ETH

DECODE ARGS
SEND
LOG
RETURN

Signed by A

gas: 79000

Execution of Tx message



gas: 79000

Contract starts executing

A

89 ETH

Transaction

to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Contract

10 ETH

DECODE ARGS

SEND

LOG

RETURN

Message

from	A
to	Contract
value	10 ETH
gas	79000
data	forward(B)

gas: 78000
-1000

Decode input parameters

A
89 ETH

Transaction	
to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Contract
10 ETH

DECODE ARGS

SEND

LOG

RETURN

Message	
from	A
to	Contract
value	10 ETH
gas	78000
data	forward(B)

gas: 68300
-9700

Send 10 eth to B

A

89 ETH

Transaction

to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Contract

10 ETH

DECODE ARGS

SEND

LOG

RETURN

Message

from	A
to	Contract
value	10 ETH
gas	68300
data	forward(B)

gas: 68300

Send 10 eth to B



gas: 68300

Send 10 eth to B



gas: 2300

Execute new message

Contract

0 ETH

-10 ETH

DECODE ARGS

SEND

LOG

RETURN

Message

from

A

to

Contract

value

10 ETH

gas

68300

data

forward(B)

Message

from

Contract

to

B

value

10 ETH

gas

2300 (+2300)

data

B

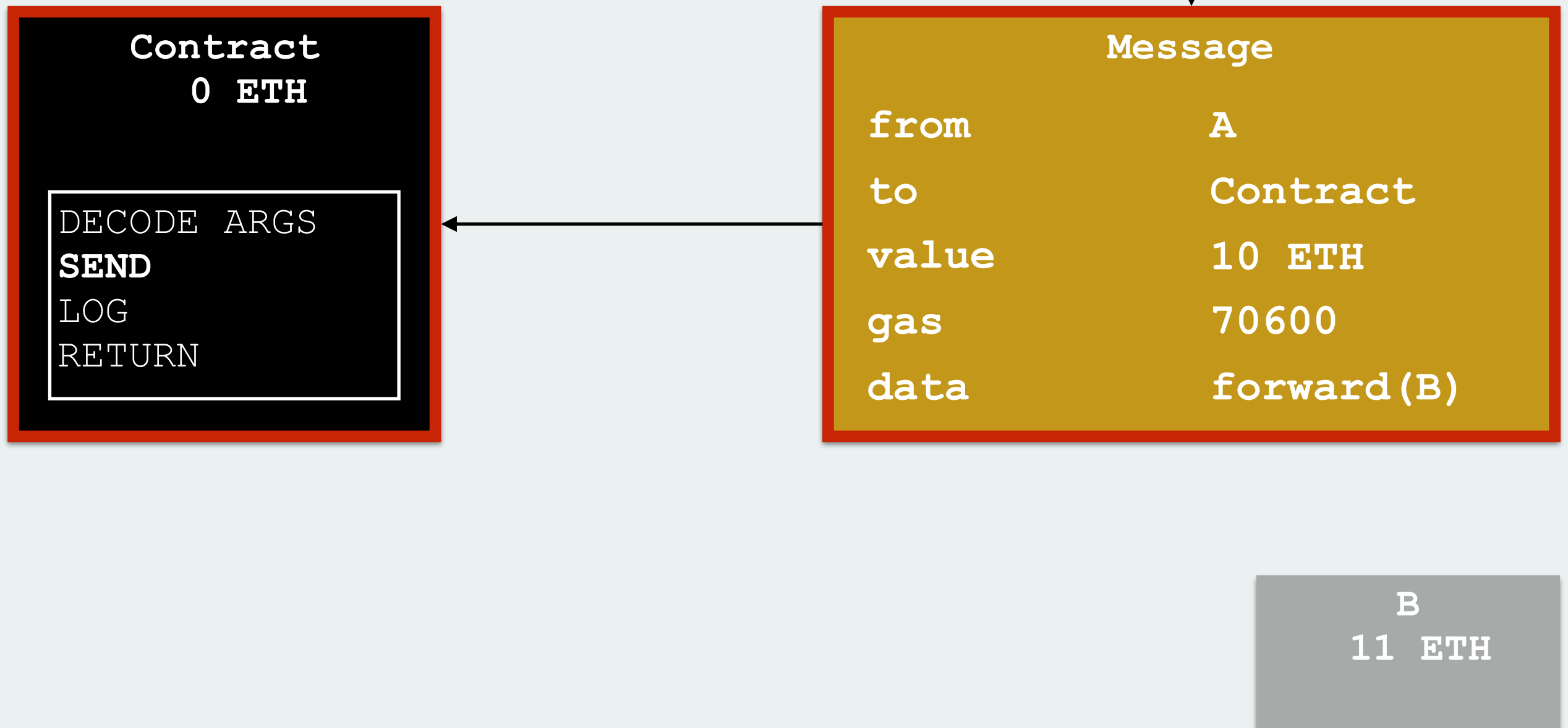
11 ETH

+10 ETH

stipend: 2300 "free" gas

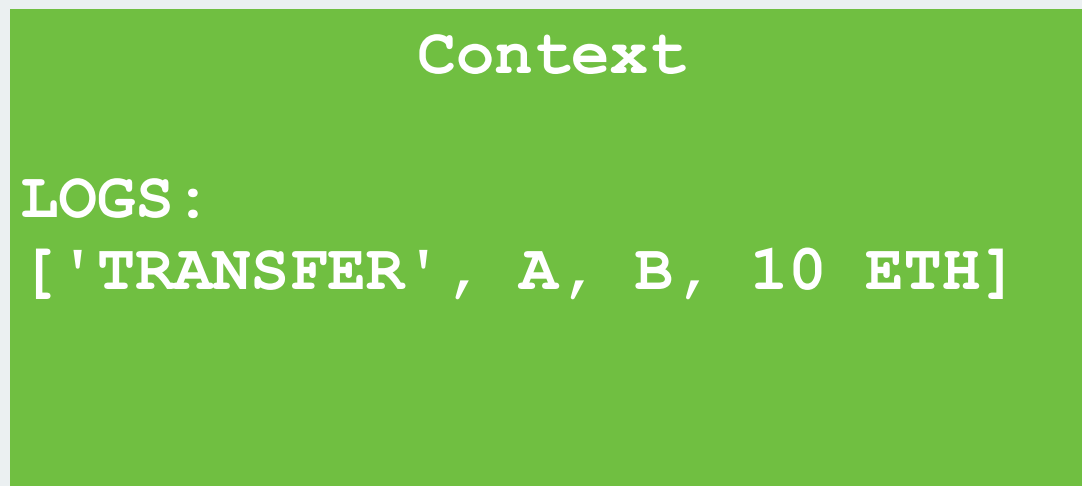
gas: 70600

Parent message continues



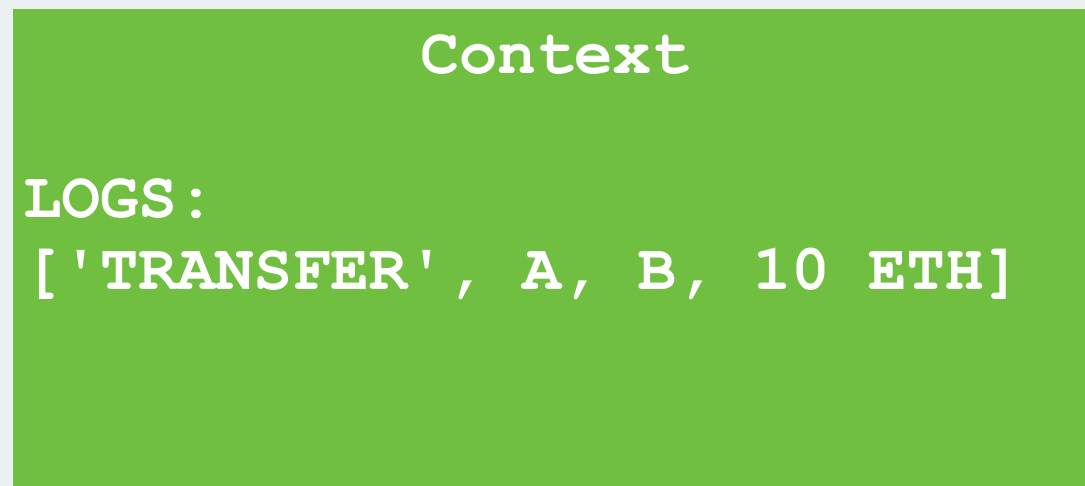
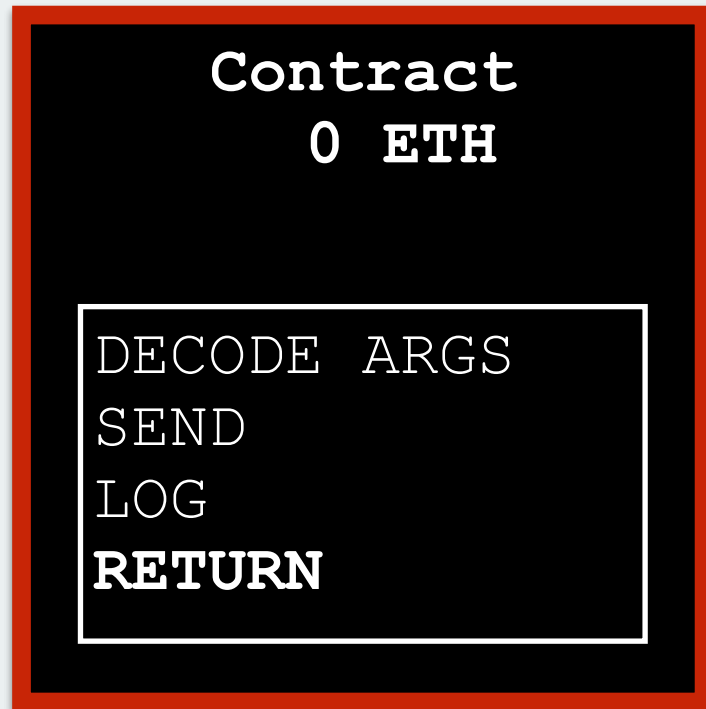
gas: 69600
-1000

Log the transfer



gas: 69000
-600

Get to end of control flow



gas: 69000

End of Tx Message

Contract
0 ETH

```
DECODE ARGS  
SEND  
LOG  
RETURN
```

Context

LOGS:
['TRANSFER', A, B, 10 ETH]

B
11 ETH

A
89 ETH



Transaction	
to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Contract
0 ETH

DECODE ARGS
SEND
LOG
RETURN

Context

LOGS:
['TRANSFER', A, B, 10 ETH]

gas: 0

Refund remaining gas

A
89.69 ETH
+0.69 ETH

gas * gasprice



Transaction	
to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Contract
0 ETH

DECODE ARGS
SEND
LOG
RETURN

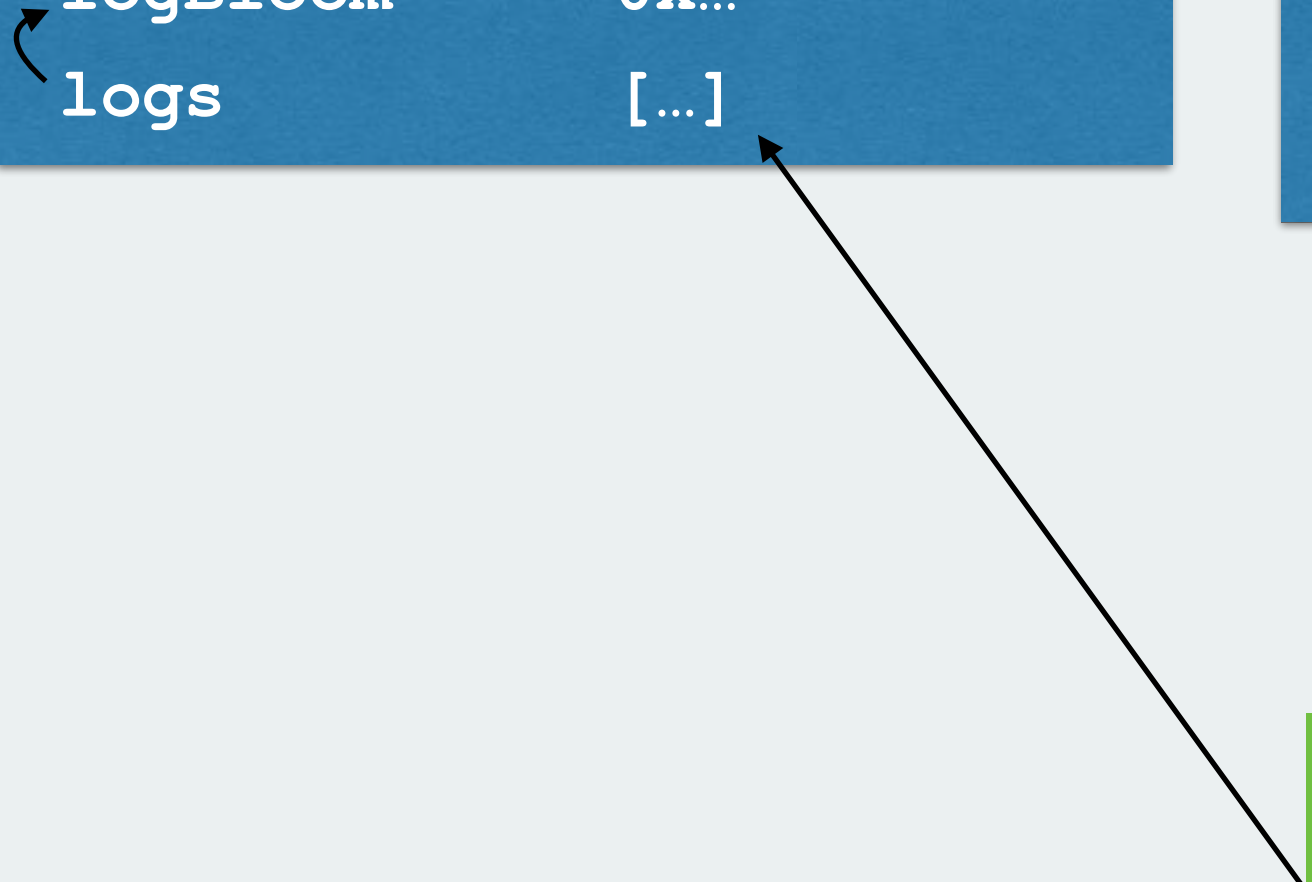
Context

LOGS:
['TRANSFER', A, B, 10 ETH]

End of transaction

Transaction Receipt

gasUsed	33000
stateRoot	0x...
logBloom	0x...
logs	[...]



Transaction

to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Context

LOGS :
['TRANSFER', A, B, 10 ETH]

Block inclusion

Combined with all other
transactions into a
Patricia Merkle Trie

transactionsRoot

Transaction

to	Contract
value	10 ETH
gas	100000
gasprice	0.00001
data	forward(B)

Combined with all other
receipts into a
Patricia Merkle Trie

receiptsRoot

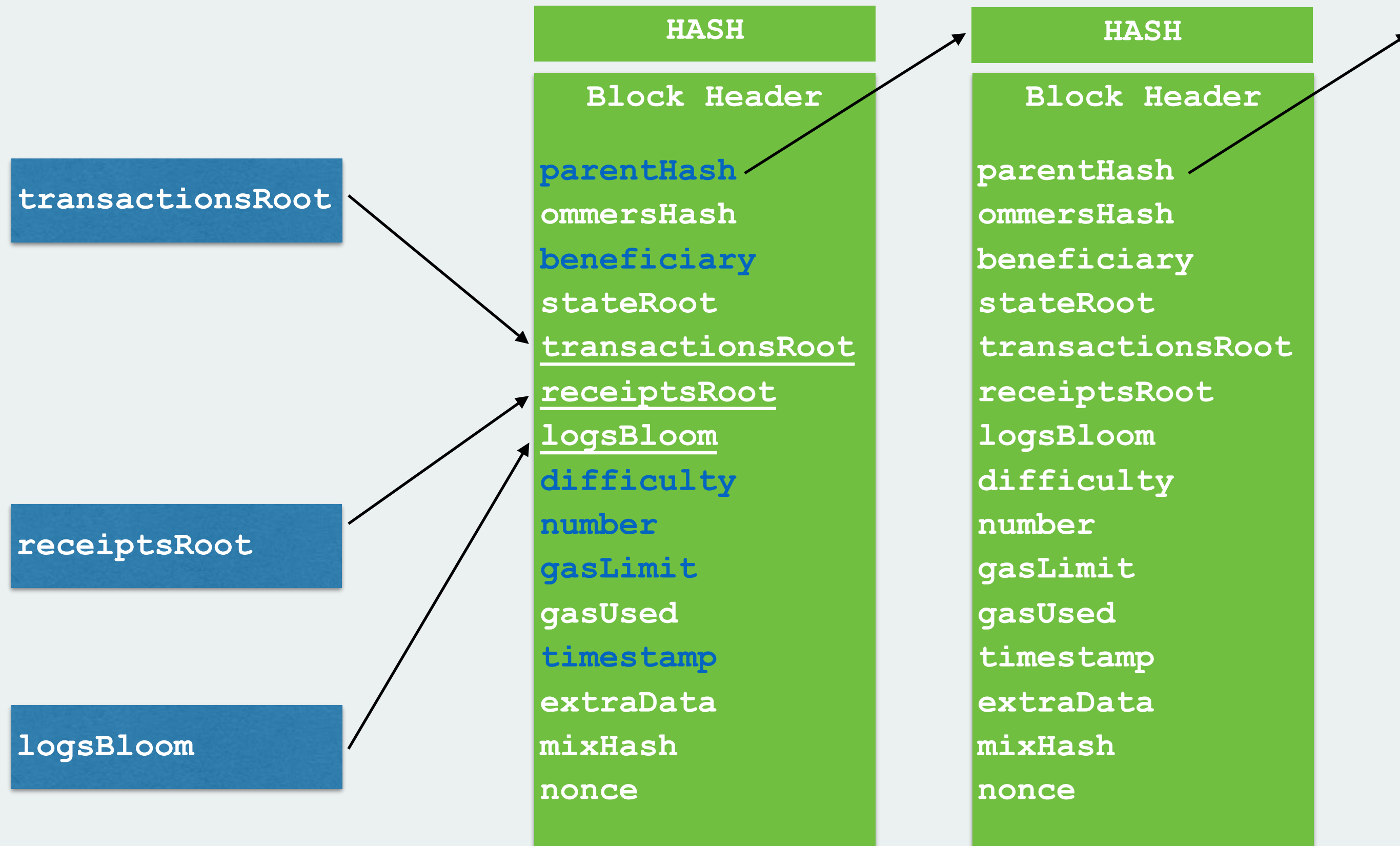
Transaction Receipt

gasUsed	33000
stateRoot	0x...
logBloom	0x...
logs	[...]

logsBloom

Combined with all other
bloom filters

Block inclusion





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



EVM

Storage

- expensive

- persistent

Memory (only during execution)

- cheaper

- byte-level access

Stack (only during execution)

- inaccessible in solidity (except assembly)



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

can also be caused manually (for exceptions)

parent message runs afterwards (but might also be oog)



EVM

Logs

for UIs

Light Clients

Fills a bloom filter in block header

Enables a quick check whether an event might have happened



IDEs



Ethereum Studio

[illegible]



ethereum

Remix IDE

Crowdfund.sol

Feed.sol

FeedAdvanced.sol

Forwarder.sol

Market.sol

Sample.sol

Sample2.sol

Sample3.sol

Subscription.sol

Test.sol

Crowdfund.sol

Feed.sol

FeedAdvanced.sol

Forwarder.sol

Market.sol

Sample.sol

Sample2.sol

Sample3.sol

Subscription.sol

Test.sol

```
1 pragma solidity >= 0.4.10;
2 contract Sample {
3
4     uint public value;
5
6     function Sample(uint initial) {
7         set(initial);
8     }
9
10    function set(uint v) {
11        value = v;
12    }
13
14    function get() returns (uint) {
15        return value;
16    }
17
18    function() {
19
20    }
21
22 }
```

Settings

Files

Contract

Debugger

Analysis

Docs

remix

Your current Solidity version is
0.4.11+commit.68ef5810.Emscripten.clang

Select new compiler version

☐ Text Wrap

☐ Enable Optimization

☒ Auto Compile

Compile



Solidity



C/Java/JS-like syntax

Compiler: solc

Available debuggers: Remix, Ethereum Studio

This workshop: solc 0.4.11 (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;  compiler version  
contract Sample {  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 initial

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 (!)

```
}
```




Message

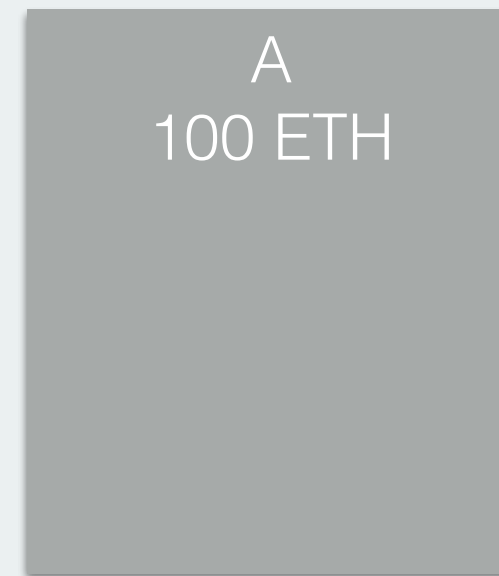
```
pragma solidity >= 0.4.10;
contract Sample {

    uint value;

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

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

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



from: A
to: Contract
value: 0ETH
data: set(6)



* gas cost omitted for simplicity



Message

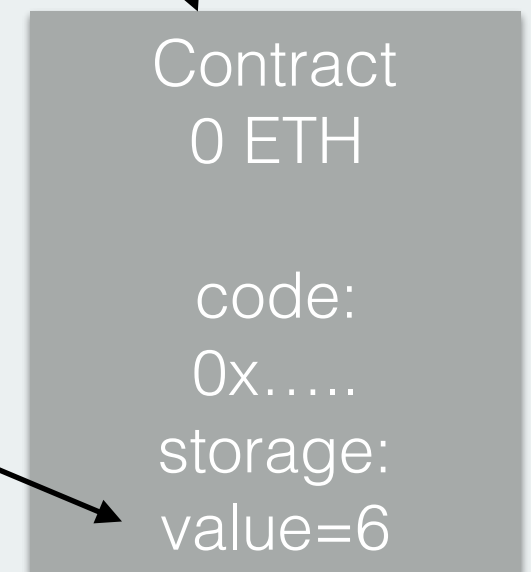
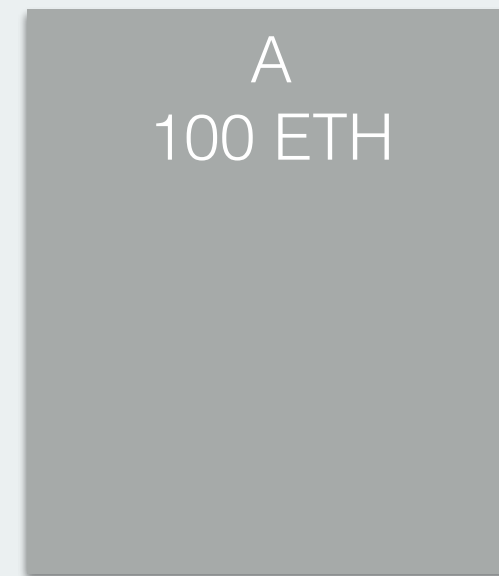
```
pragma solidity >= 0.4.10;
contract Sample {

    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



Message

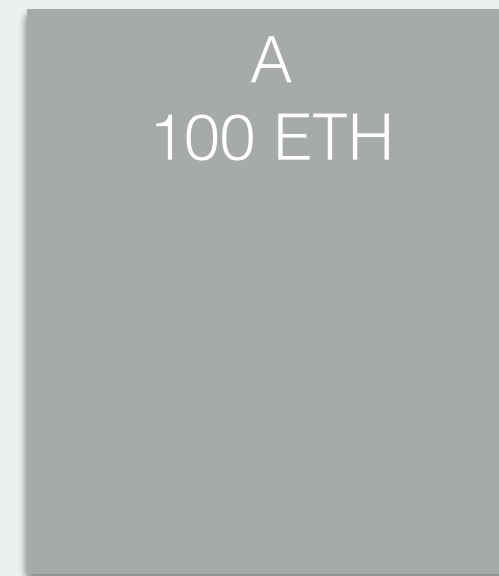
```
pragma solidity >= 0.4.10;
contract Sample {

    uint value;

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

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

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



from: A
to: Contract
value: 0 ETH
data: get()
return: 6



* 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



ethereum 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 = 0x3049280948ffa0afafafffffffaafffff9789372;  
    mapping (address => uint) balances;  
    Sample otherContract;  
}
```



ethereum

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;  
}
```




ethereum

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); up to 25k gas  
    FundTransfer(funder.addr, funder.amount, false);  
}
```

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



`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



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



block.

coinbase: miner of the block

difficulty

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

blockhash: function to get hashes of past blocks

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;  
    }  
}
```



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



ethereum

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!



ethereum

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**

ethereum 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;
```

ethereum 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);
```



ethereum

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);  
}
```



ethereum

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);  
}
```




ethereum

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!



ethereum

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);  
}
```



ethereum

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);
```


ethereum 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
```



ethereum 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();  
}
```



Visibility

External

Can only be called by a message

Public (default)

Can be called by anyone

Private

Can only be called by the contract itself

Internal

Cannot be called by a message

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



ethereum

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;
```



EPM - Ethereum Package Manager

Many dev frameworks (e.g. truffle)

JS / solidity based unit testing

Standards



ethereum

"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)



1N7wgGE2eeMpiDS6LFbSxypsVbHo4H3Spv

The End

0xe9b0d93a7514d619b5eea66b7bacf665a69320e6
riatspace.eth

