Ethereum and Smart Contracts

Wenchao Wang
Original slides by Joseph Bonneau

Replicated State Machines

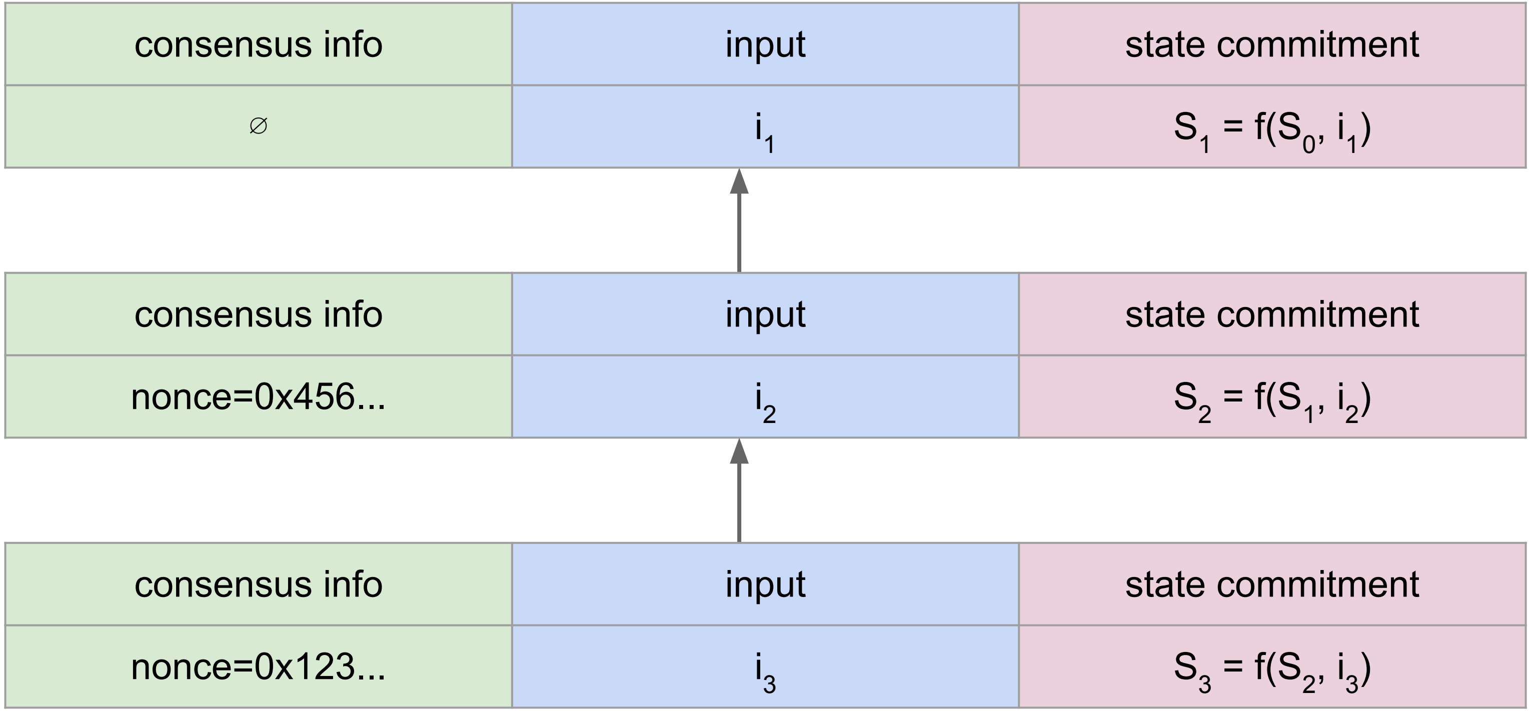
Recap: *Replicated state machines*

- Set of possible states **S**
- Set of possible inputs **I**
- Set of possible outputs **O**

- Transition function f: **S** × **I** → **S** × **O**

- Start state s ∈ **S**   (genesis block)

Blockchains capture an ordered list of inputs



consensus info	input	state commitment
∅	i_1	$S_1 = f(S_0, i_1)$

consensus info	input	state commitment
nonce=0x456...	i_2	$S_2 = f(S_1, i_2)$

consensus info	input	state commitment
nonce=0x123...	i_3	$S_3 = f(S_2, i_3)$

Explicit state commitments offer many advantages

consensus info	input	state commitment
nonce=0x123...	B→C 11 signed(Bob)	{A: 33, B:6, C: 11}

- Inconsistencies surface immediately
- Light clients can quickly get current state
- Can efficiently verify sequence between any two blocks

Ethereum in one slide

- States **S** = a map from *addresses* to *state*

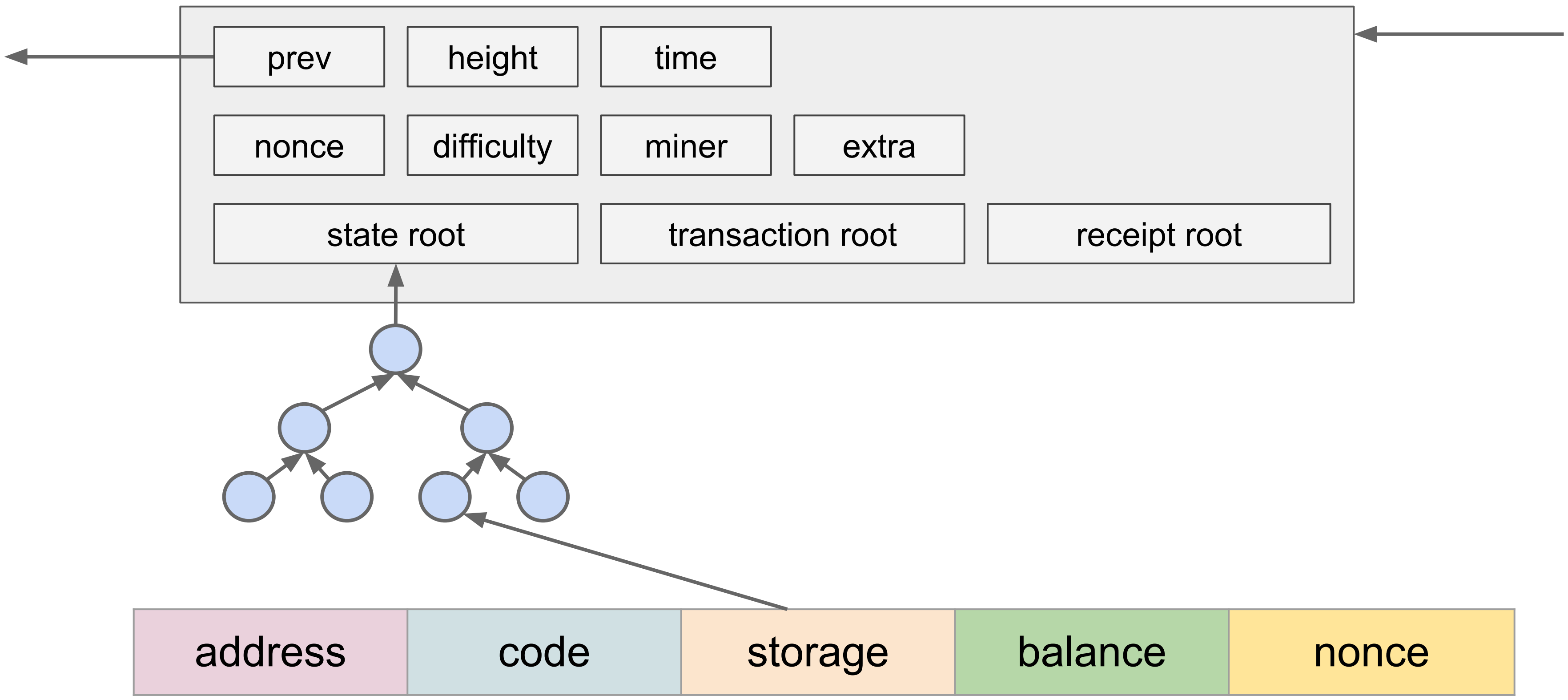
address	code	storage	balance	nonce

- Inputs **I** (transactions)

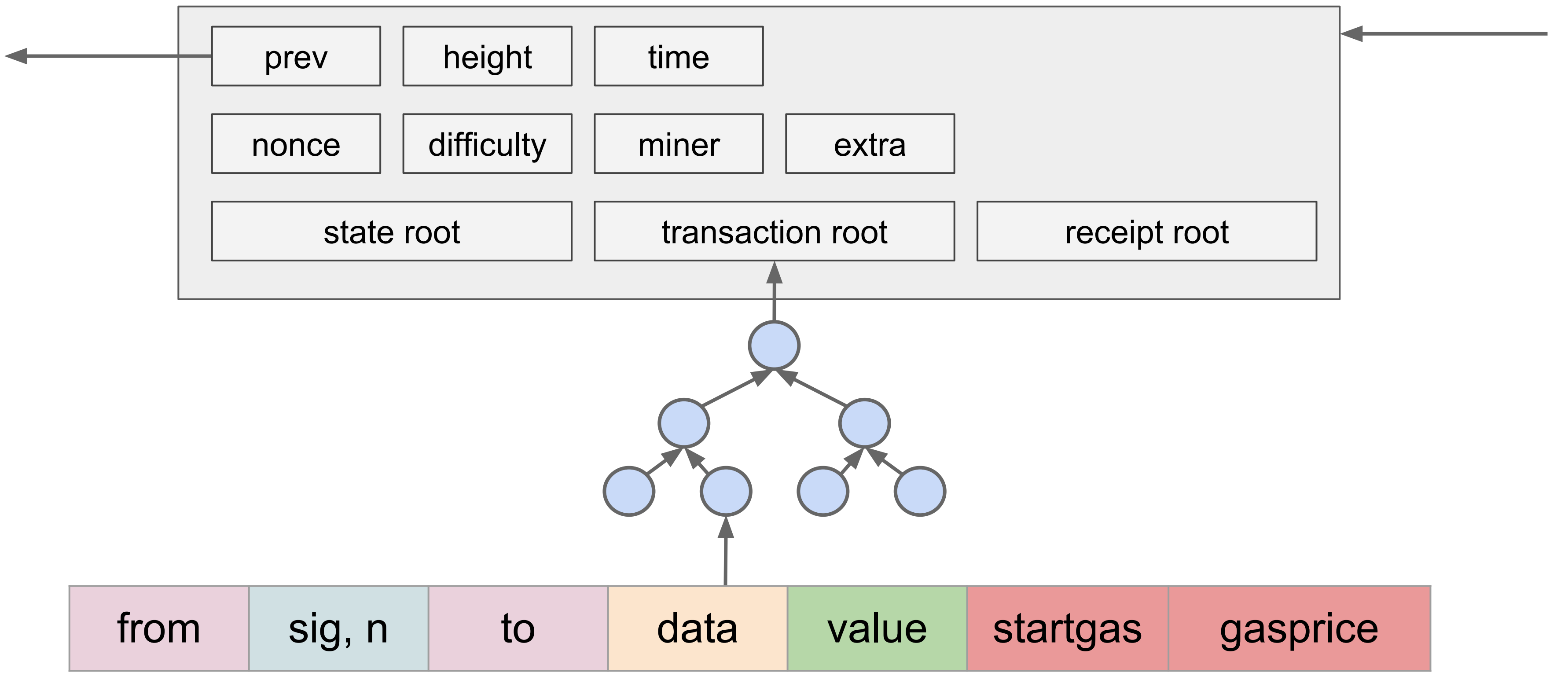
from	sig, n	to	data	value	startgas	gasprice

- Transition **f:**
    - ○ validate signature
    - ○ run to.code(from, data, value, startgas, gasprice)
- Start state: ∅

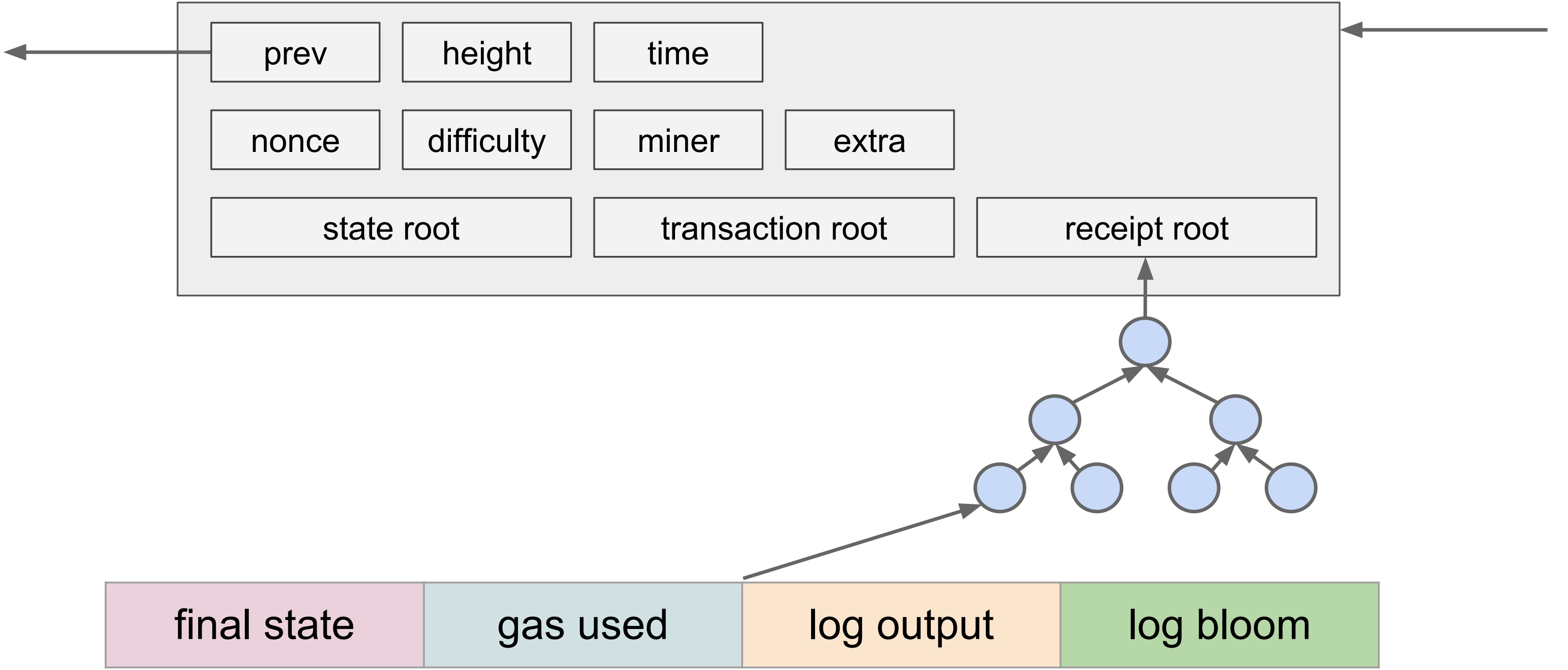
The full* Ethereum blockchain structure



The full* Ethereum blockchain structure



The full* Ethereum blockchain structure



Ethereum addresses can be *accounts* or *contracts*

address	code	storage	balance	nonce

	account	**contract**
address	H(pub_key)	H(creator, nonce)
code	∅	EVM code
storage	∅	Merkle storage root
balance	ETH balance	
nonce	#transaction sent	

Volatile fields

Ethereum
Virtual
Machine

EVM is stack-based, like BTC script

```
PUSH1 0
CALLDATALOAD
SLOAD
NOT
PUSH1 9
JUMPI
STOP
JUMPDEST
PUSH1 32
CALLDATALOAD
PUSH1 0
CALLDATALOAD
SSTORE
```

Features

- 1024-depth stack
- 32-byte words
- Accelerated crypto
    - SHA-3
    - Big num multiply
    - GF-256 operations

EVM provides basic API for I/O

Input:

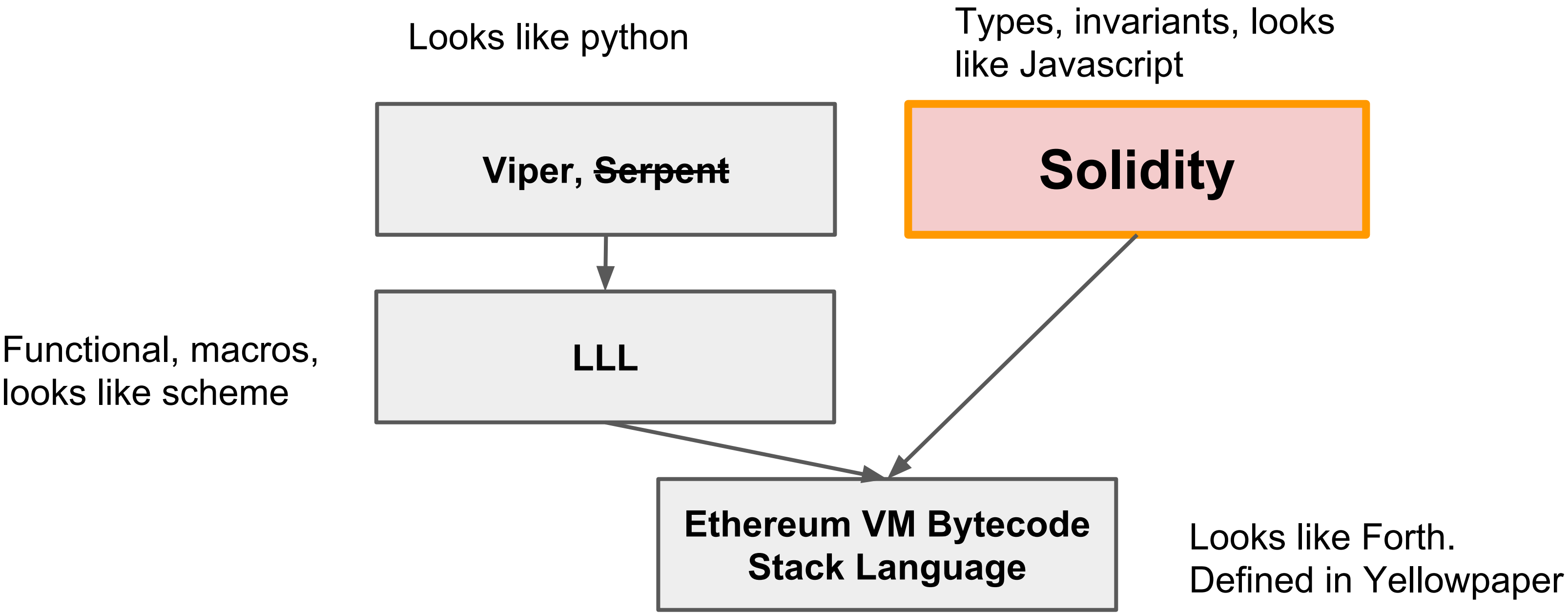
- tx info: sender, value, gas limit
- resource use: gas remaining, memory used
- block info: depth, timestamp, miner, hash

Output:

- send messages (call other contracts and/or send money)
- write to logs
- self destruct

Solidity

Ethereum code written in Solidity, compiled to EVM



Looks like python

Viper, ~~Serpent~~

Functional, macros, looks like scheme

LLL

Types, invariants, looks like Javascript

Solidity

Ethereum VM Bytecode Stack Language

Looks like Forth. Defined in Yellowpaper

Solidity should look familiar

- Syntax looks like C++, JavaScript etc.

- Contracts look like classes/objects
  - Can mark functions `internal`

- Static typing
  - Most types can be cast e.g. `bool(x)`

Solidity types

- `bool, uint8, uint16, ... uint256, int8, ... int256`
- `address`
- `string`
- `byte[]`
- `mapping(keyType ==> valueType)`

EVM memory model offers a *lot* of space

Storage: $\{0,1\}^{256} \rightarrow \{0,1\}^{256}$ map (persistent)

Memory: $\{0,1\}^{256} \rightarrow \{0,1\}^{256}$ map (volatile between tx)

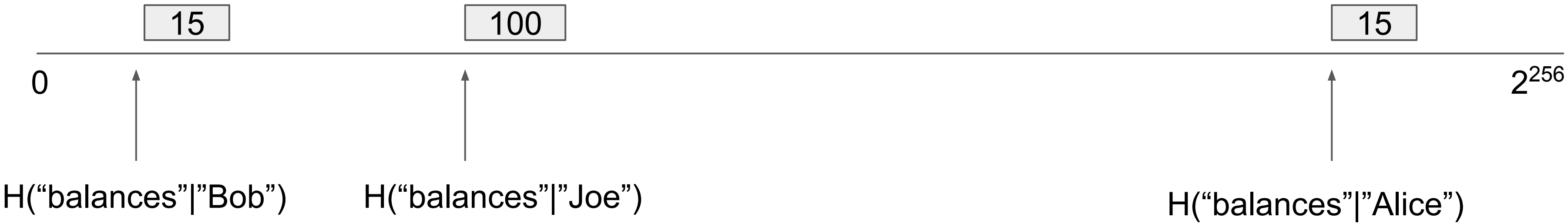
- in other words, both can represent 2^{264} bits!
- arranged in 256-bit words
- all memory is zero-initialized

Storage in Ethereum is **very** expensive. Limiting memory use is critical

Clever implementation of maps in Solidity

```
mapping(string => uint256) balances;
```

Alice	15
Bob	15
Joe	100



- every item requires at least one 256-bit word
- balances[“Andrew”] is 0 if “Andrew” doesn’t exist or if “Andrew” has 0 balance
- to delete a key, set balances[“Andrew’] = 0
- Cannot delete an entire map!

Polite contracts call `throw` on errors

```
uint8 numCandidates;
uint32 votingFee;
mapping(address => bool) hasVoted;
mapping(uint8 => uint32) numVotes;

/// Cast a vote for a designated candidate
function castVote(uint8 candidate) {
    if (msg.value < votingFee)
        return;
    if (hasVoted[msg.sender])
        throw;

    hasVoted[msg.sender] = true;
    numVotes[candidate] += 1;
}
```

Throw ensures no effects persisted except gas consumption

throw: 0xfe invalid opcode
revert: 0xfd REVERT (Byzantium fork)

Modifiers ease repetitive safety checks

```
address public owner;
uint public electionEnd;

modifier onlyBy(address _account){
    require(msg.sender == _account);
    _;
}

modifier onlyAfter(uint _block) {
    require(block.blocknumber >= _block);
    _;
}

function endElection()
    onlyBy(owner) onlyAfter(electionEnd){
```

Solidity gotchas

- Member variables `public` by default
    - Setters, getters automatically provided
- Functions must be marked `payable` to accept funds
- Member variables go to storage by default
    - Method variables go to memory
- Fallback `function()`
    - Called if no function specified (e.g. send)
    - Called if non-existent function called
- msg.sender vs. tx.origin

https://solidity.readthedocs.io/en/develop/solidity-in-depth.html

Solidity and EVM may outgrow Ethereum itself

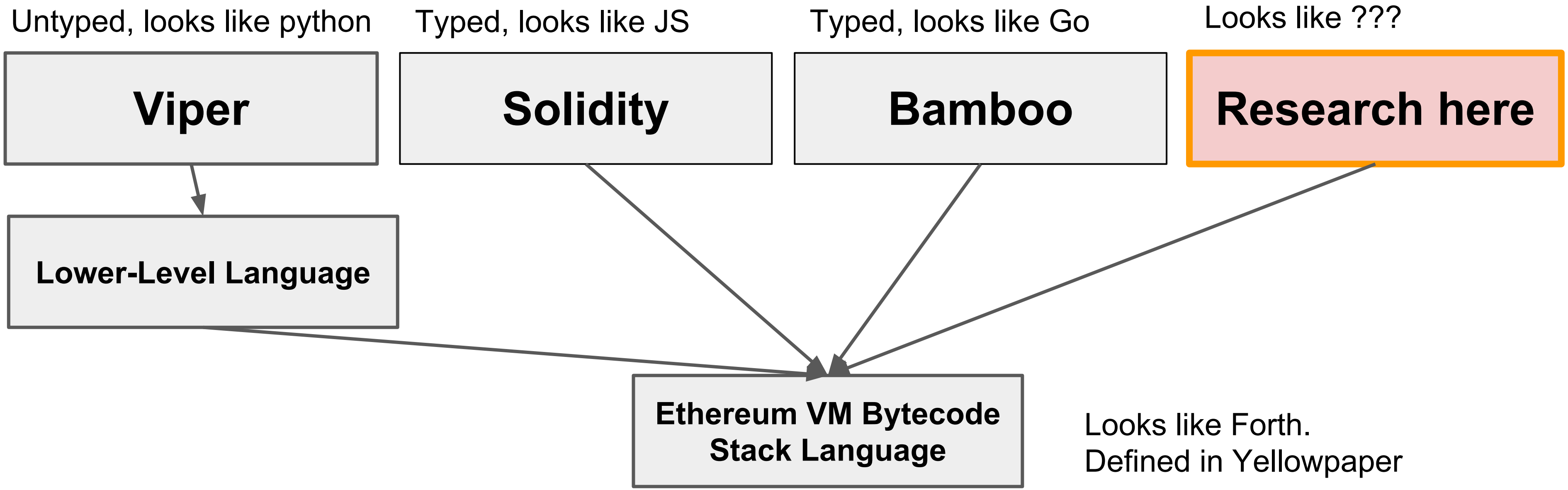
ENTERPRISE
ETHEREUM
ALLIANCE

- Enterprise Ethereum Alliance, still in infancy (Announced Feb 28)

-Goal: support EVM, Solidity and tools for private blockchains

    - maintain compatibility with Ethereum network

Don’t like Solidity? Write your own language!



Untyped, looks like python

Viper

Lower-Level Language

Typed, looks like JS

Solidity

Typed, looks like Go

Bamboo

Looks like ???

Research here

Ethereum VM Bytecode
Stack Language

Looks like Forth.
Defined in Yellowpaper

Solidity by example: Rock-paper-scissors

Warmup: Rock Paper Scissors in Ethereum

1. `function add_player() payable;`

   Takes player’s deposit of 1 ETH.

2. `function input(uint choice);`

   Records player’s choice (0 or 1 or 2)

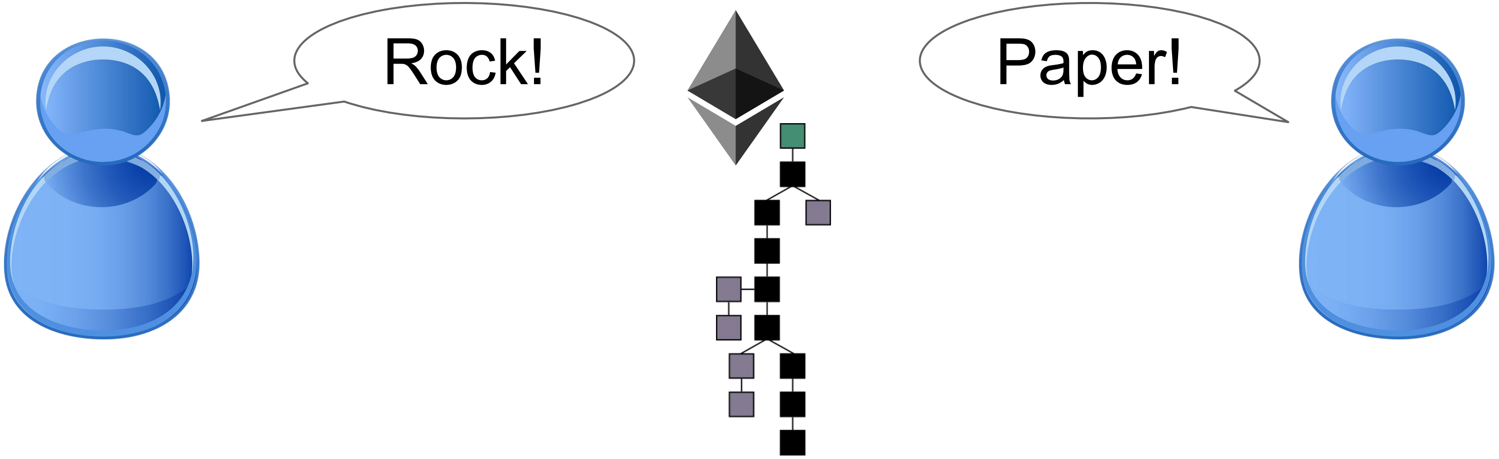
3. `function check_winner();`

   Decides who wins, pays the winner

```
struct Player {
    uint choice;
    uint addr;
}

function add_player() payable {
  assert(num_players < 2);
  assert(msg.value >= 2000 wei);
  reward += msg.value;
  player[num_players].addr = msg.sender;
  num_players++;
}

function input(uint choice, uint idx) {
   assert(msg.sender == player[idx].addr);
   player[idx].choice = choice;
}
```

```
uint num_players = 0;
uint reward = 0;
mapping (uint => Player) player;

function check_winner() returns(int) {
  var p0_choice = player[0].choice;
  var p1_choice = player[1].choice;
  if (p0_choice - p1_choice % 3 == 1)
      // Player 0 wins
      player[0].addr.send(reward);
  else
  if (p0_choice - p1_choice % 3 == 2)
      // Player 1 wins
      player[1].addr.send(reward);
  else {
      player[0].addr.send(reward/2);
      player[1].addr.send(reward/2);
  }
}
```

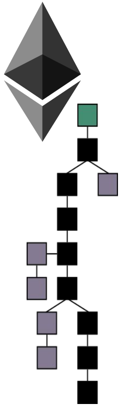
One problem: Front Running



Seconds go by....

Avoid Front-Running with Commitments

nonce0 ← Random String
H0 = hash(nonce0, “Rock”)



nonce1 ← Random String
H1 = hash(nonce1, “Scissors”)

Commit to H0

Commit to H1

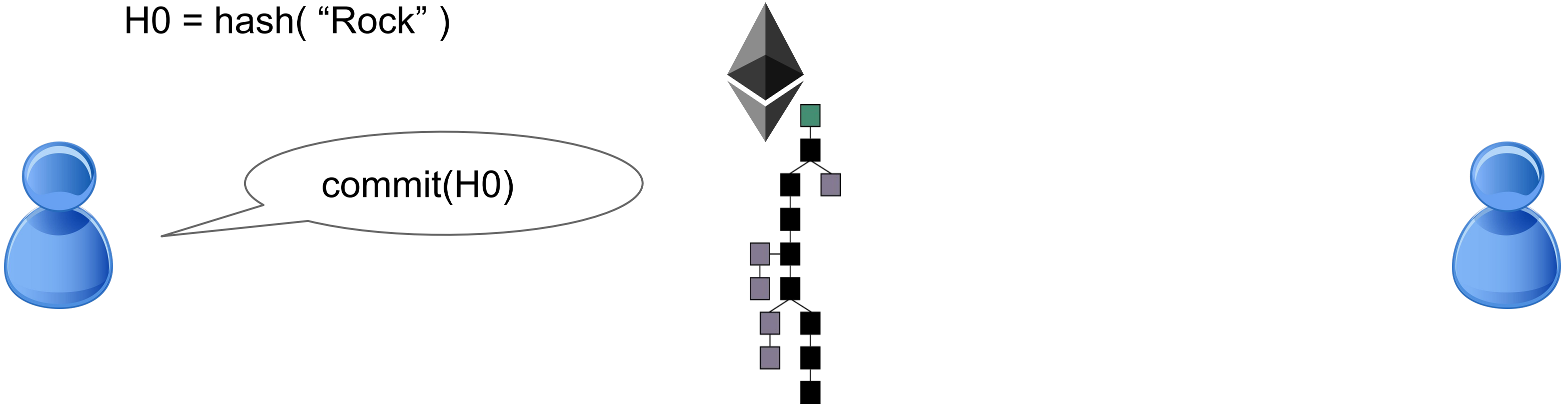
Commitment Deadline

Reveal
nonce0, “Rock”

Reveal
nonce1, “Scissors”

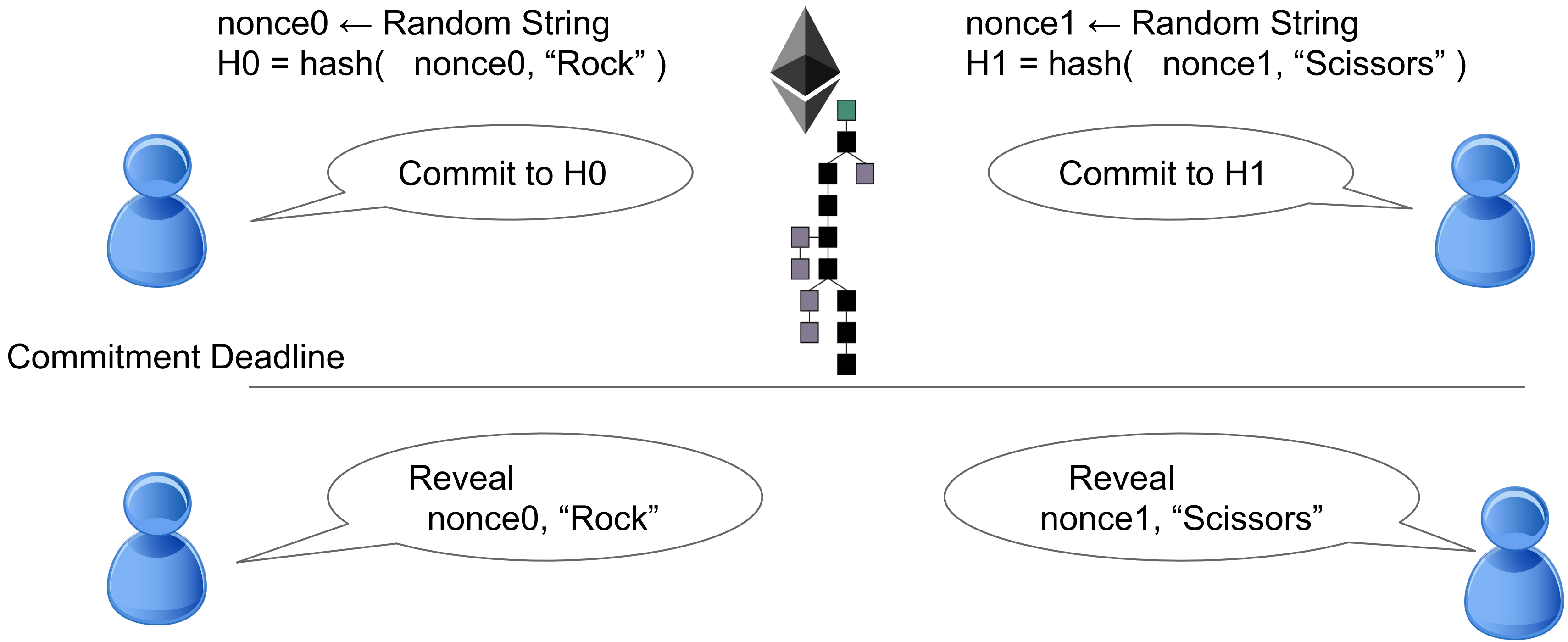
Why are the “nonces” necessary?

H0 = hash(“Rock”)



commit(H0)

Avoid Front-Running with Commitments



nonce0 ← Random String
H0 = hash(nonce0, “Rock”)

nonce1 ← Random String
H1 = hash(nonce1, “Scissors”)

Commit to H0

Commit to H1

Commitment Deadline

Reveal
nonce0, “Rock”

Reveal
nonce1, “Scissors”

What remaining problems are there?

Inter-contract calls

Three levels of contract call in Ethereum

Original message:

from	sig, n	to	data	value	startgas	gasprice
A	sig	B	d	x	S	g

Results of a call to C:

	CALL	**CALLCODE**	**DELEGATE CALL**
msg.sender	B		A
value	x’≤ B.balance		
data	(as specified)		
startgas	s’ ≤ gas remaining		
storage updated	C	B	

Solidity syntax for calling other contracts

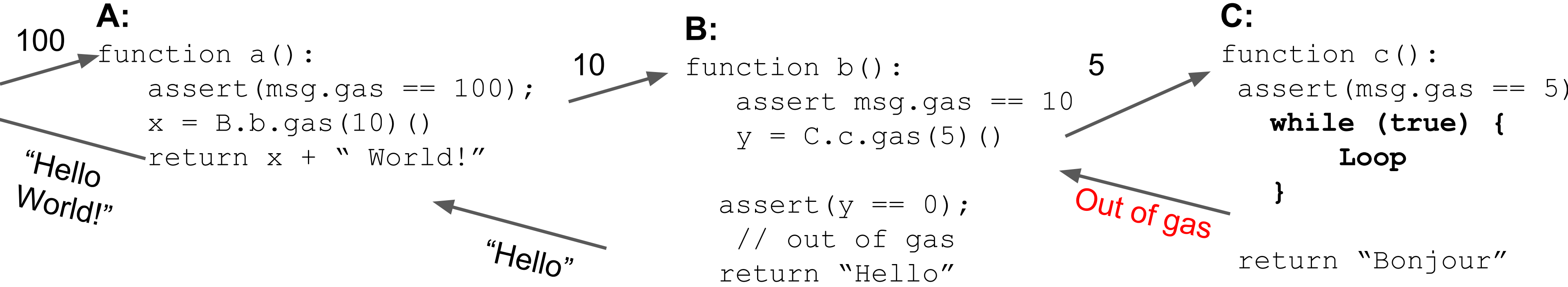
- `a.send(x)` sends x to address a
  - returns 0 if this fails due to call stack

- `foo.call.value(3).gas(20764)( bytes4(sha3("bar()")));`
  - also `callcode, delegatecall`
  - default is 0 value, all available gas

- `new` constructor deploys a new contract
  - Careful, it's expensive!

Remember:
Smart contracts code is fixed *forever*.
Calls required to update functionality

Callers can choose how much gas to send



100

A:

```
function a():
   assert(msg.gas == 100);
   x = B.b.gas(10)()
   return x + " World!"
```

"Hello World!"

10

"Hello"

B:

```
function b():
   assert msg.gas == 10
   y = C.c.gas(5)()

  assert(y == 0);
   // out of gas
  return "Hello"
```

5

Out of gas

C:

```
function c():
 assert(msg.gas == 5)
   while (true) {
       Loop
   }

 return "Bonjour"
```

Subtleties to contract calls

- **Data:** unlimited params/return values
  - Direct mapped to memory address + size

- **Exceptions:** out of gas, bad jump, etc.
  - No state changes persisted
  - Control returns to caller

- **Call stack limit:** 1024
  - Calls from 1024th frame will fail

Many idioms for calling functions

100	Amount in Ether
25	Amount in gas
55	Data argument

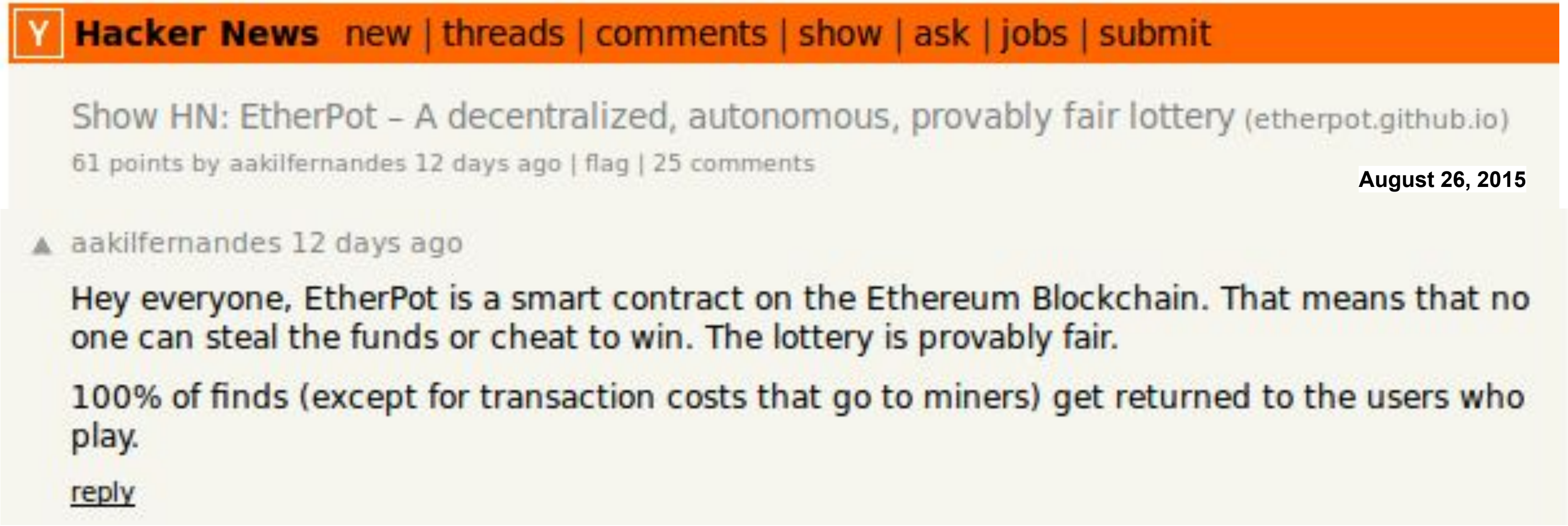
Solidity:

recipient.**send**(100)	returns 0	**2300**
recipient.**transfer**(100)	exception	**2300**
recipient.**call**.value(100).gas(25)()	returns 0	**25**
recipient.foo.value(100)(55)	exception	all of it
recipient.foo(55)	exception	all of it

Safe transfer:
Introduced in 2017 in Solidity after various incidents

Unchecked send and other problems

The EtherPot Story



Show HN: EtherPot – A decentralized, autonomous, provably fair lottery (etherpot.github.io)

61 points by aakilfernandes 12 days ago | flag | 25 comments

August 26, 2015

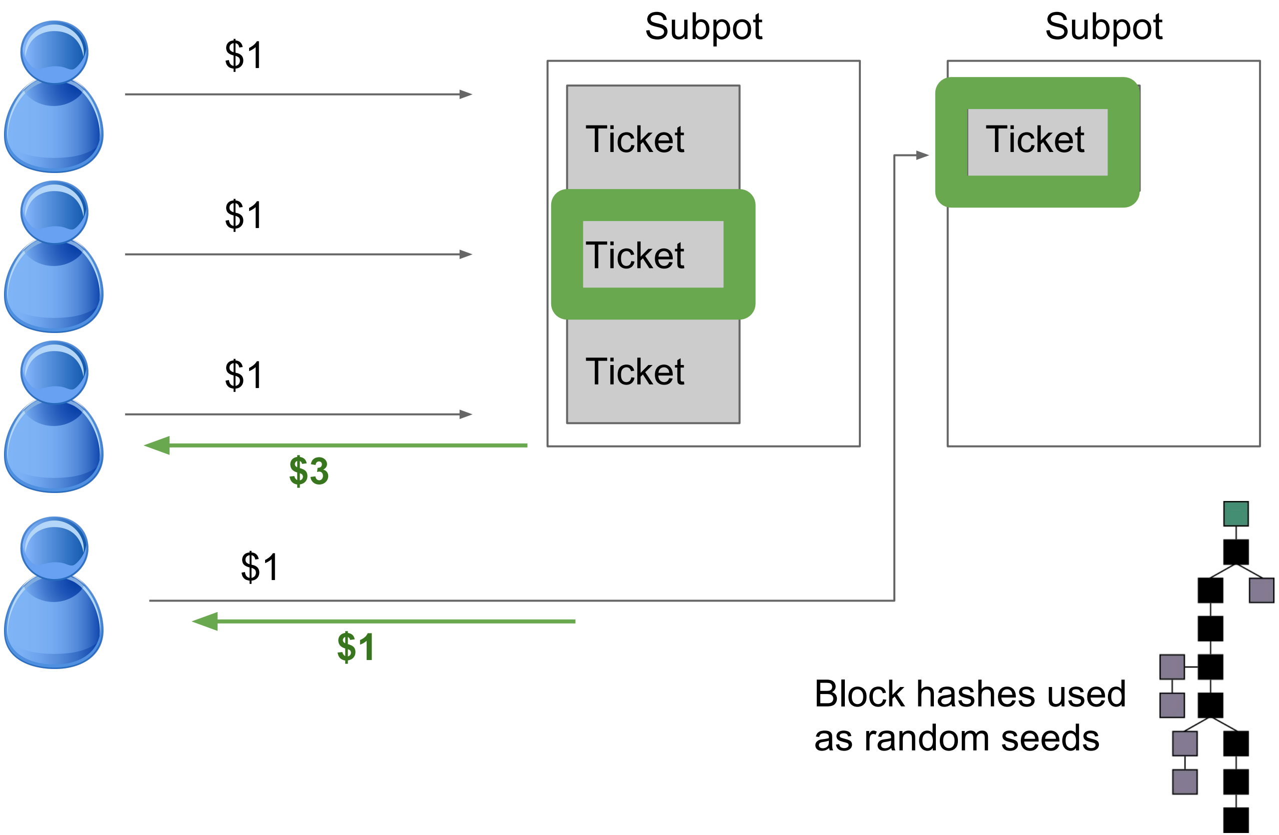
▲ aakilfernandes 12 days ago

Hey everyone, EtherPot is a smart contract on the Ethereum Blockchain. That means that no one can steal the funds or cheat to win. The lottery is provably fair.

100% of finds (except for transaction costs that go to miners) get returned to the users who play.

reply

How Etherpot Works



Block hashes used as random seeds

1. Each round lasts 1 day. (Everything is reset after a round.)

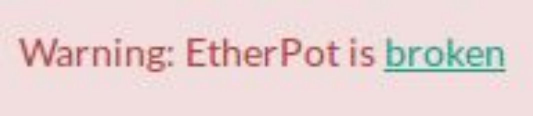
2. Users deposit money to purchase Tickets at a fixed price.

Each "Subpot" holds a fixed number of tickets.

3. After the round ends, the next *N* block hashes are used as random seeds to determine the winners of N subpots.
(1 block for each subpot)

Bugs in EtherPot

Within days, hundreds of dollars of Eth paid out to the wrong recipient.

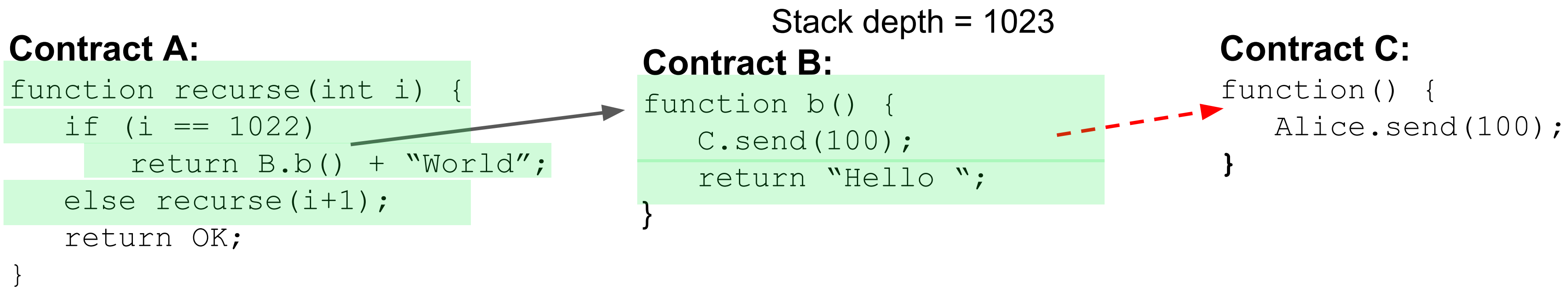


Warning: EtherPot is broken

Multiple more bugs were found.

Call stack hazard:
Maximum call stack depth is 1023

Suppose we call A.recurse(0). Does Alice get 100?



Stack depth = 1023

Contract A:

```
function recurse(int i) {
   if (i == 1022)
      return B.b() + “World”;
   else recurse(i+1);
   return OK;
}
```

Contract B:

```
function b() {
   C.send(100);
   return “Hello “;
}
```

Contract C:

```
function() {
   Alice.send(100);
}
```

A.recurse(0)	Stack depth = 0
A.recurse(1)	Stack depth = 1
….	
A.recurse(1022)	Stack depth = 1022

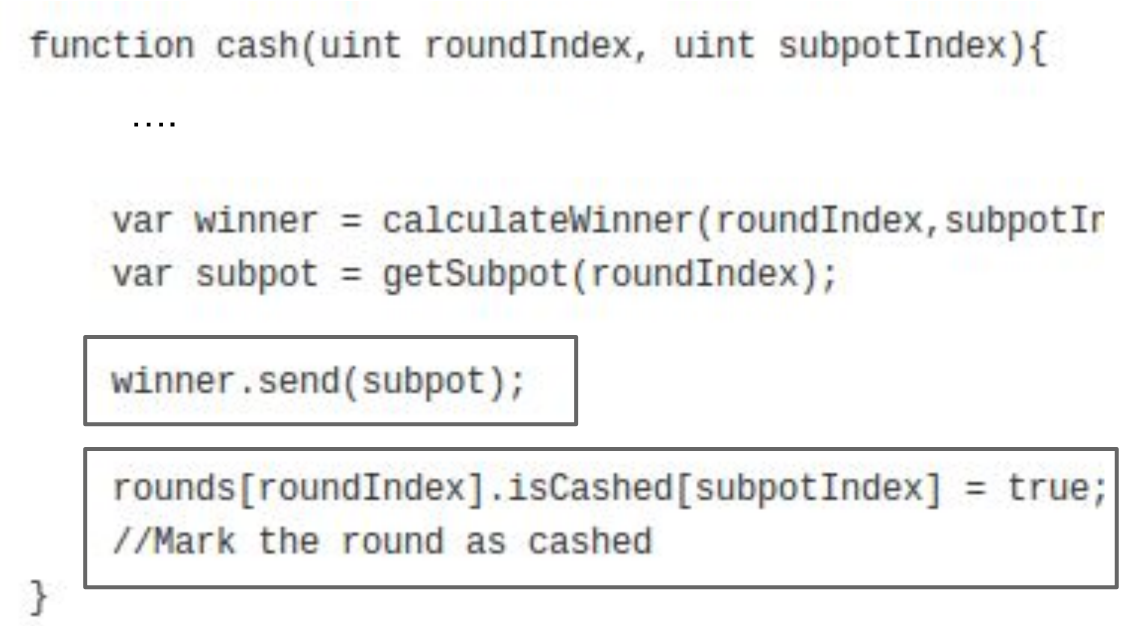
Returns “Hello World!”

https://gist.github.com/amiller/665cc46970f2c0684d2a

The Callstack hazard in Etherpot

Attack Contract:

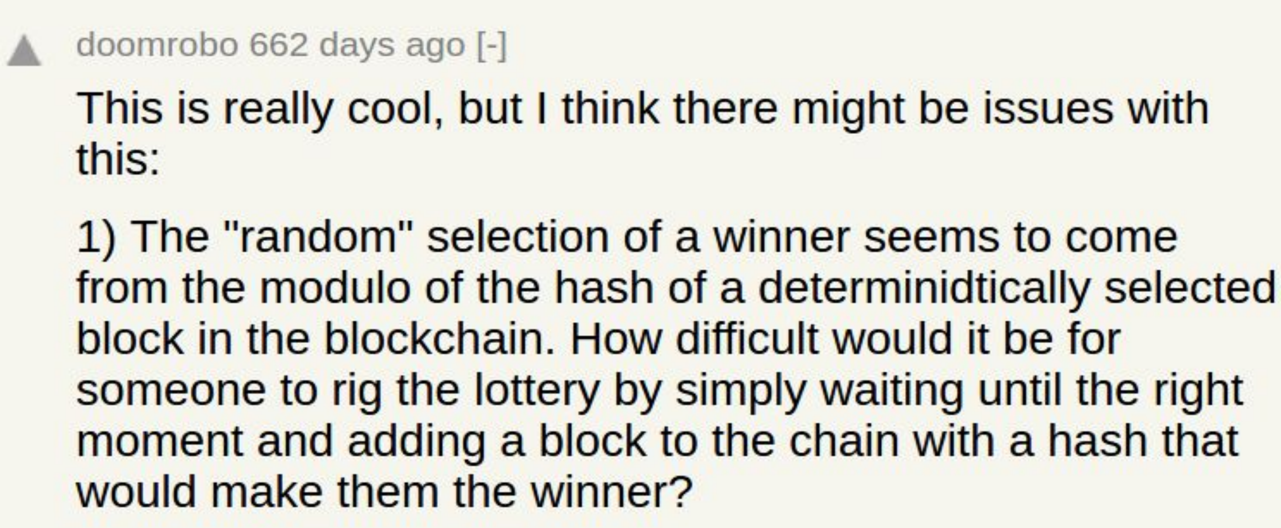
```
function recurse(int i) {
   if (i == 1022)
      Etherpot.cash(r,idx)
   else recurse(i+1);
   return OK;
}
```



```
function cash(uint roundIndex, uint subpotIndex){

        ....

    var winner = calculateWinner(roundIndex,subpotIr
    var subpot = getSubpot(roundIndex);

    winner.send(subpot);

    rounds[roundIndex].isCashed[subpotIndex] = true;
    //Mark the round as cashed
}
```

Result: ***attacker can destroy all the funds in the contract***

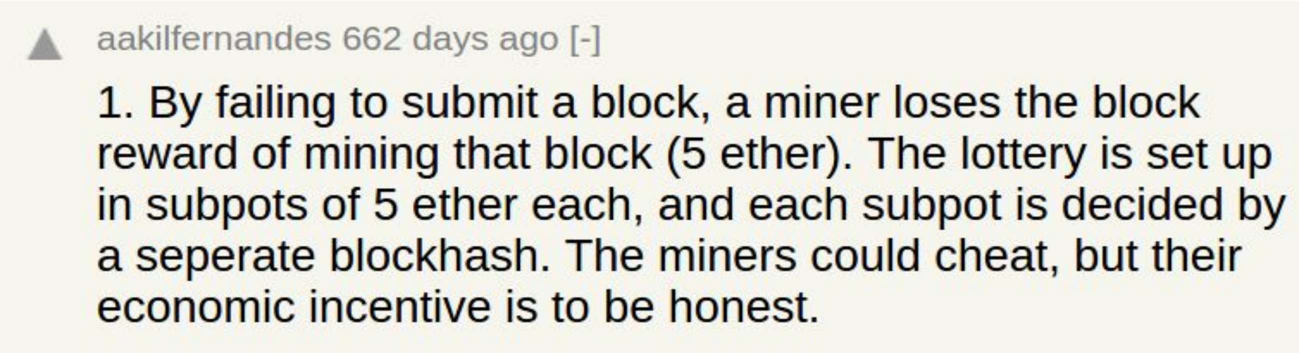
EtherPot's incentive mechanism



doomrobo 662 days ago [-]

This is really cool, but I think there might be issues with this:

1) The "random" selection of a winner seems to come from the modulo of the hash of a determinidtically selected block in the blockchain. How difficult would it be for someone to rig the lottery by simply waiting until the right moment and adding a block to the chain with a hash that would make them the winner?



aakilfernandes 662 days ago [-]

1. By failing to submit a block, a miner loses the block reward of mining that block (5 ether). The lottery is set up in subpots of 5 ether each, and each subpot is decided by a seperate blockhash. The miners could cheat, but their economic incentive is to be honest.

EtherPot’s incentive mechanism

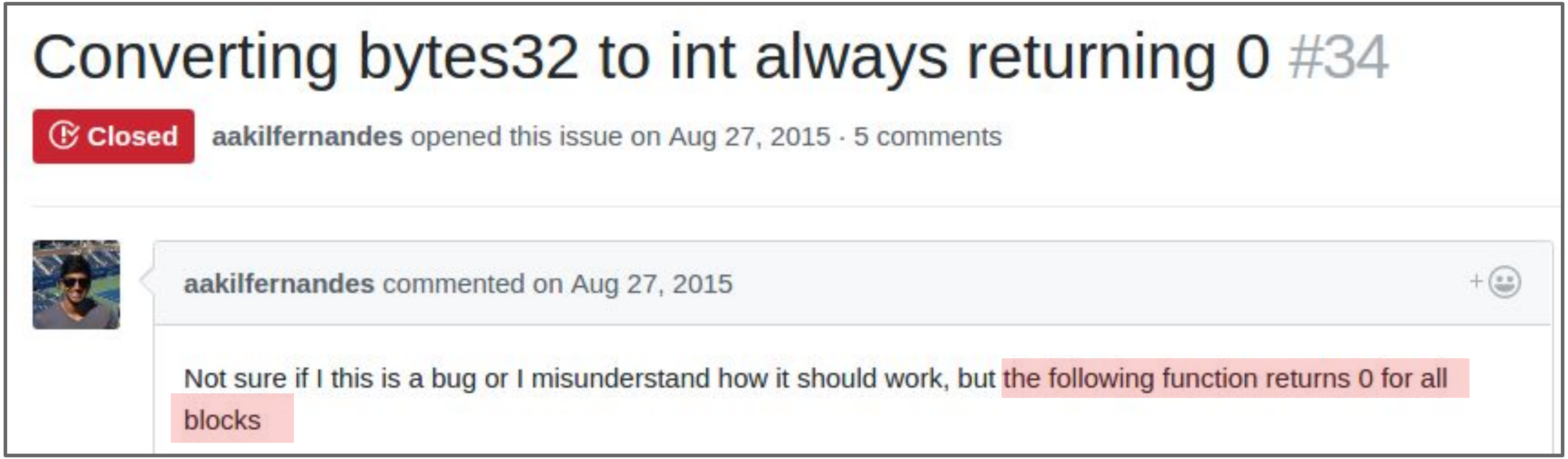
Can miners influence the outcome of lottery?

Yes - by withholding blocks

Solution: “subpots” smaller than block reward

Problem: GHOST

Withheld blocks can still get 88% reward if revealed in the next round



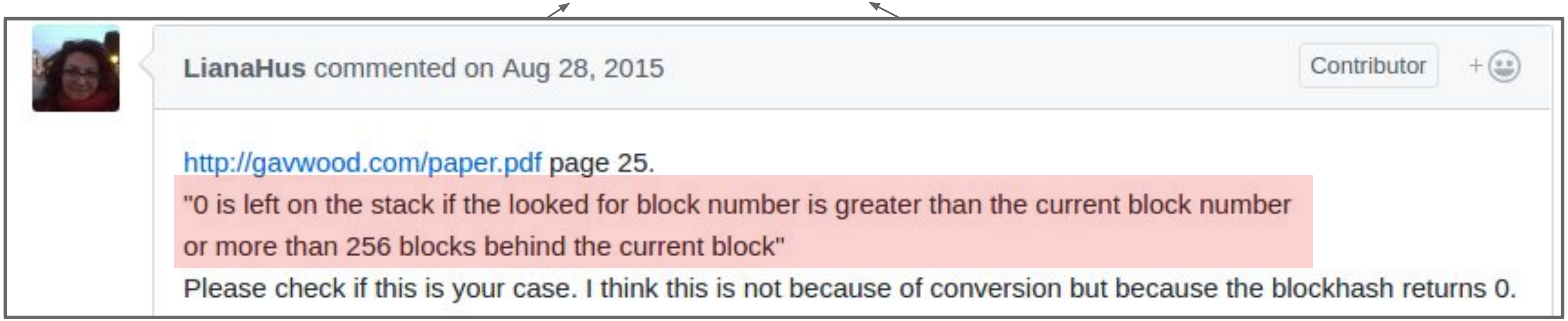
Converting bytes32 to int always returning 0 #34

Closed **aakilfernandes** opened this issue on Aug 27, 2015 · 5 comments

aakilfernandes commented on Aug 27, 2015

Not sure if I this is a bug or I misunderstand how it should work, but the following function returns 0 for all blocks

```
function getHashOfBlock(uint blockIndex) constant returns(uint){

    return uint(block.blockhash(blockIndex));
```



LianaHus commented on Aug 28, 2015

Contributor

http://gavwood.com/paper.pdf page 25.
"0 is left on the stack if the looked for block number is greater than the current block number or more than 256 blocks behind the current block"
Please check if this is your case. I think this is not because of conversion but because the blockhash returns 0.

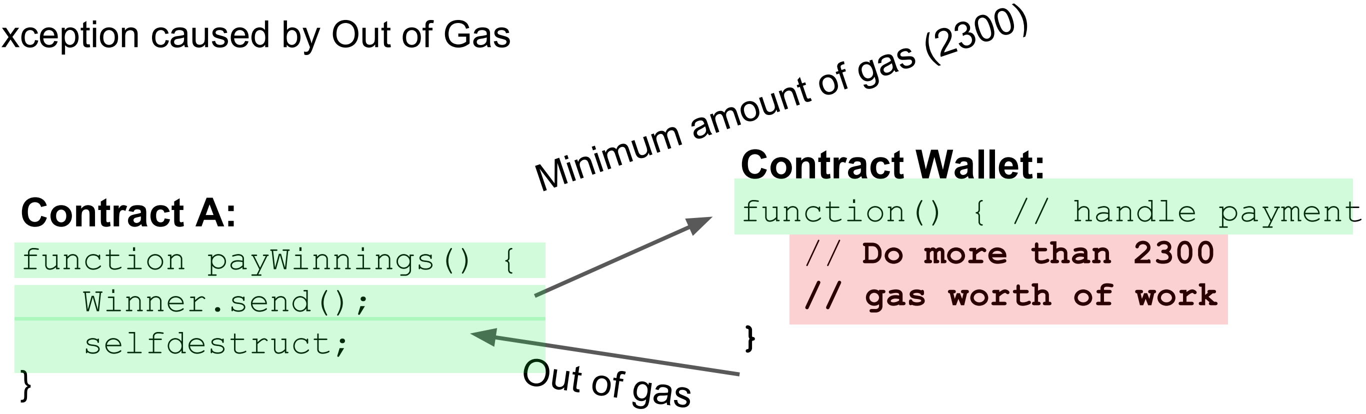
King of the Ether Throne

Post-Mortem Investigation (Feb 2016)

During the 'Turbulent Age' (06 Feb 2016 to 08 Feb 2016) of the King of the Ether Throne, a serious issue caused some monarch compensation payments and over/under payment refunds to fail to be sent. This web page explains the issue, the causes, the response, and the recommended solutions. It is currently in FINAL form.

Call stack hazard:
Exceptions are not propagated (for default function)

Example: an exception caused by Out of Gas



Minimum amount of gas (2300)

Out of gas

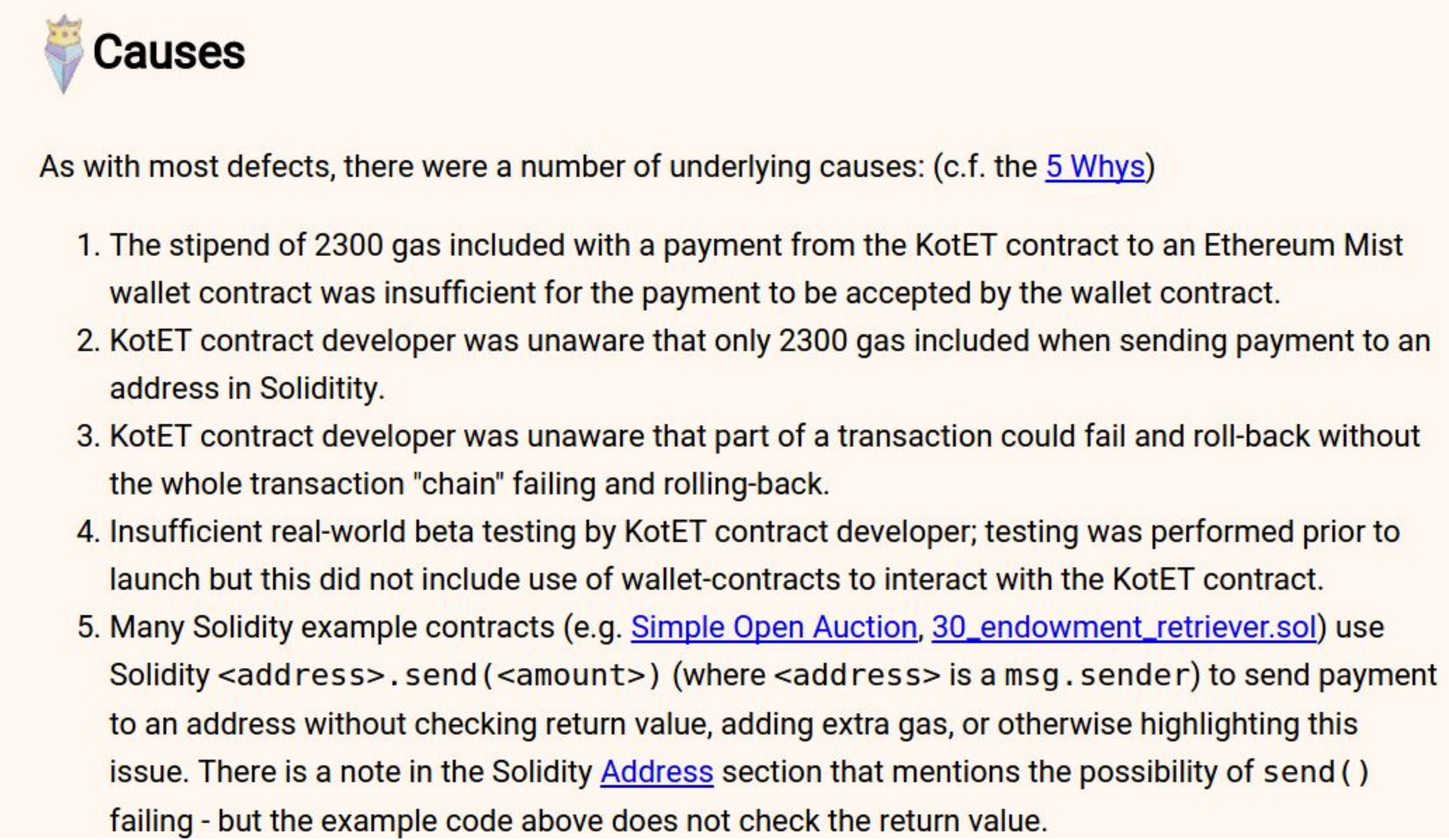
Contract A:

```
function payWinnings() {
    Winner.send();
    selfdestruct;
}
```

Contract Wallet:

```
function() { // handle payment
    // Do more than 2300
    // gas worth of work
}
```

Result: If you played King of Ether Throne using a "Wallet Contract", your winnings would be destroyed forever.



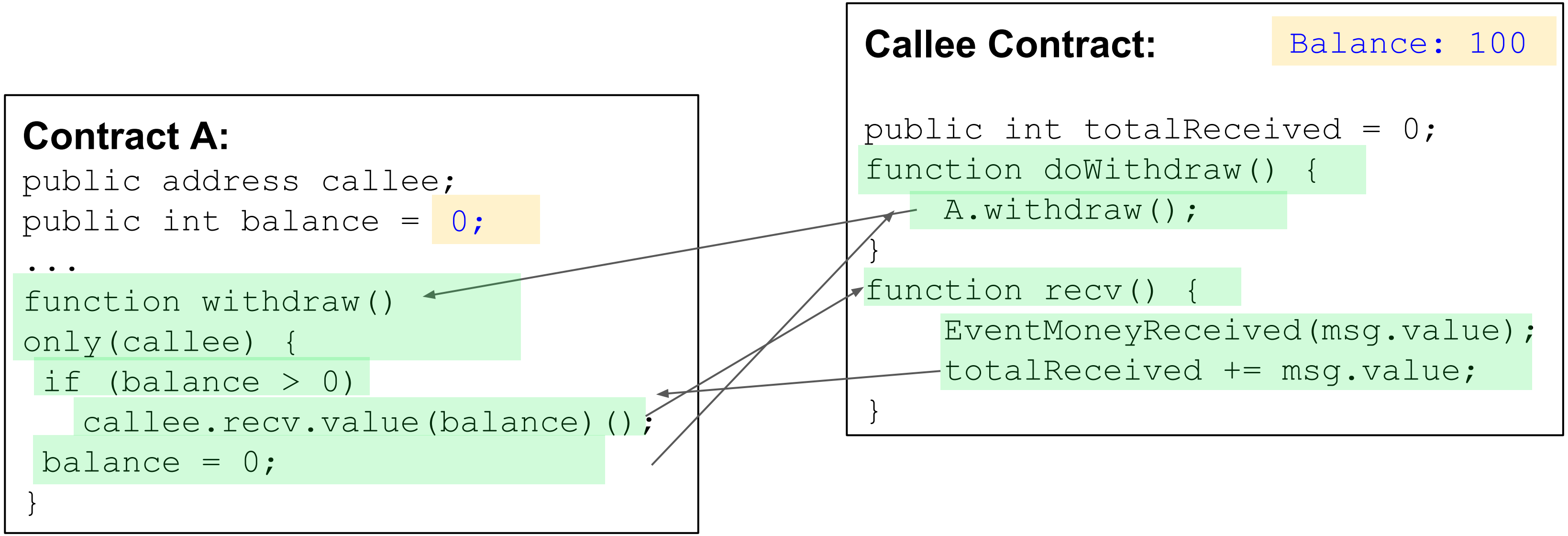
Causes

As with most defects, there were a number of underlying causes: (c.f. the 5 Whys)

1. The stipend of 2300 gas included with a payment from the KotET contract to an Ethereum Mist wallet contract was insufficient for the payment to be accepted by the wallet contract.
2. KotET contract developer was unaware that only 2300 gas included when sending payment to an address in Soliditity.
3. KotET contract developer was unaware that part of a transaction could fail and roll-back without the whole transaction "chain" failing and rolling-back.
4. Insufficient real-world beta testing by KotET contract developer; testing was performed prior to launch but this did not include use of wallet-contracts to interact with the KotET contract.
5. Many Solidity example contracts (e.g. Simple Open Auction, 30_endowment_retriever.sol) use Solidity `<address>.send(<amount>)` (where `<address>` is a `msg.sender`) to send payment to an address without checking return value, adding extra gas, or otherwise highlighting this issue. There is a note in the Solidity Address section that mentions the possibility of `send()` failing - but the example code above does not check the return value.

Re-entrancy

Re-entrancy hazards in Ethereum



Contract A:

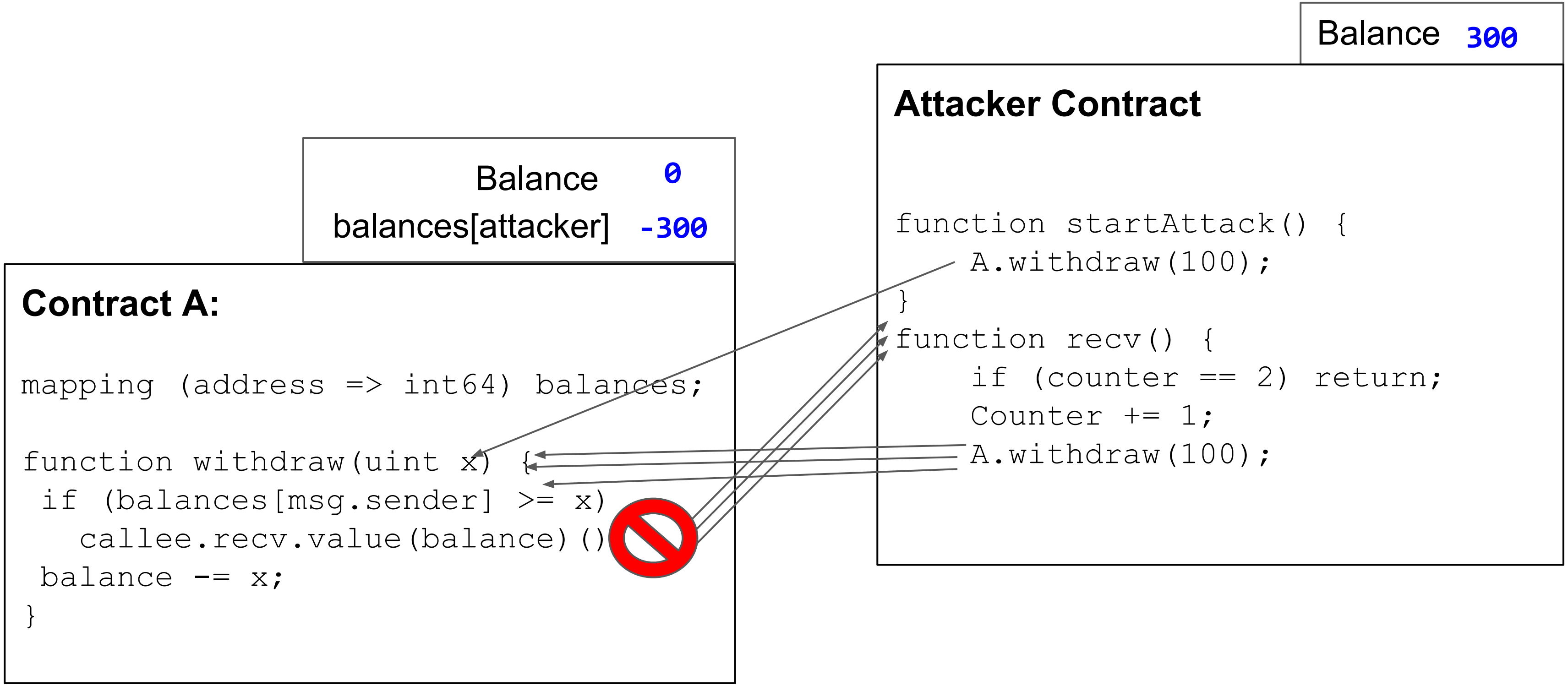
```
public address callee;
public int balance = 0;
...
function withdraw()
only(callee) {
 if (balance > 0)
   callee.recv.value(balance)();
 balance = 0;
}
```

Callee Contract: Balance: 100

```
public int totalReceived = 0;
function doWithdraw() {
    A.withdraw();
}
function recv() {
    EventMoneyReceived(msg.value);
    totalReceived += msg.value;
}
```

Done!   Callee withdraws 100.

Re-entrancy hazards in Ethereum



Balance 0

balances[attacker] -300

Contract A:

```
mapping (address => int64) balances;

function withdraw(uint x)  {
 if (balances[msg.sender] >= x)
   callee.recv.value(balance)()
 balance -= x;
}
```

Balance 300

Attacker Contract

```
function startAttack() {
    A.withdraw(100);
}
function recv() {
    if (counter == 2) return;
    Counter += 1;
    A.withdraw(100);
```

Fixes to re-entrancy

- only use send() or transfer() to limit gas
- Modifiers:

```
bool reentrantLock;

modifier noReentrancy{

    if (!reentranLock){

        reentrantLock = true;

        _;

    }
```

The “Checks / Effects / Interactions” paradigm

A Best Practice guideline for safe smart contract behavior

When receiving a message, do the following in order:

1. Perform all input validation and checks on current state. Discard the message if validation fails.

2. Update local state.

3. Finally, pass on interactions to trigger other contracts.

Contract A:

```
public address callee;
public int balance = 100;
…



function withdraw()
only(callee) {
 if (balance <= 0) return;
 var toSend = balance;
 balance = 0;
 callee.recv.value(toSend)();
}
```

Ethereum project

Ethereum is “run” by the Ethereum Foundation

Compatible “alt-clients” exist (e.g. Parity, Consensys)

Ethereum blockchain is different than Bitcoins

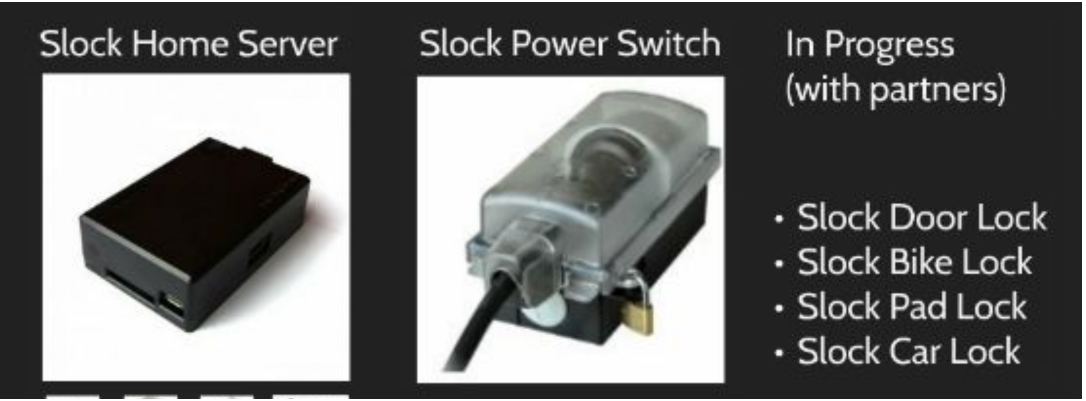
	Ethereum	Bitcoin
Target time between blocks	14.5 seconds	10 minutes
Proof of work	Equihash	SHA-256^2
Stale block rewards	Uncle rewards	none

Hard Forks are planned in Ethereum

release	date
Frontier	July 2015
Homestead	March 2016
DAO hard fork	July 2016
Byzantium	October 2017
Constantinople	2019?

The DAO

slock.it   a Blockchain + IoT company



Slock Home Server

Slock Power Switch

In Progress (with partners)

- Slock Door Lock
- Slock Bike Lock
- Slock Pad Lock
- Slock Car Lock

Example use case:

1. AirBnB user submits payment to the Ethereum blockchain

2. Slock Home Server (Ethereum client) receives the transaction

3. Power switch connected to Home Server receives “unlock” command, unlocks the door

slock.it built The DAO as a custom fundraising tool

“DAO”: Decentralized Autonomous Organization (coined by Vitalik in 2013)

Built by slock.it to raise funds for their company

Main idea: A decentralized hedge fund

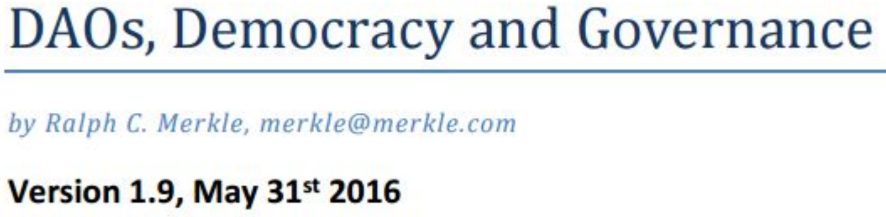
Investors contribute funds, receive ownership “tokens”

Investors jointly decide how to spend funds, by voting in proportion to tokens

Many additional mechanisms:

“Splitting” to prevent hostile takeover

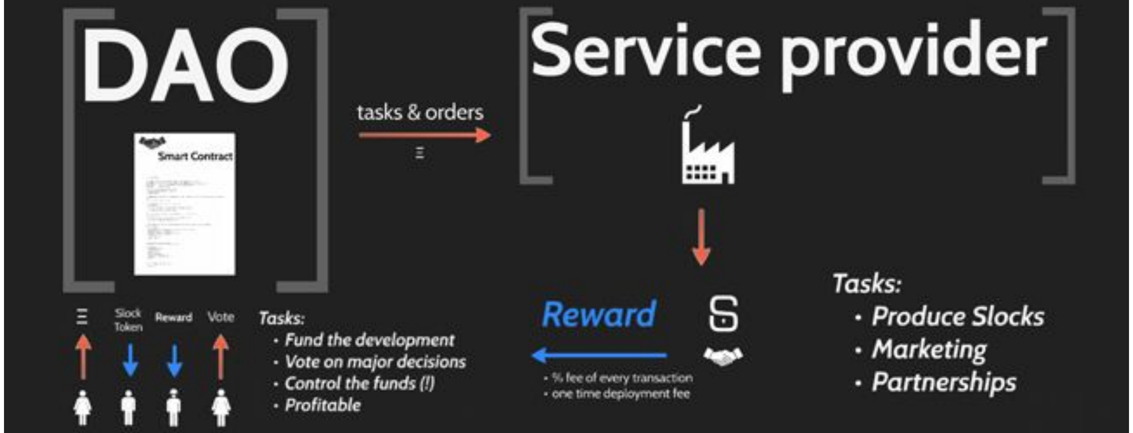
Reward disbursing



DAOs, Democracy and Governance

by Ralph C. Merkle, merkle@merkle.com

Version 1.9, May 31st 2016



DAO

Smart Contract

tasks & orders

Ξ

Service provider

Ξ

Slock Token

Reward

Vote

Tasks:

- *Fund the development*
- *Vote on major decisions*
- *Control the funds (!)*
- *Profitable*

Reward

- % fee of every transaction
- one time deployment fee

Tasks:

- ***Produce Slocks***
- ***Marketing***
- ***Partnerships***

Slock Home Server

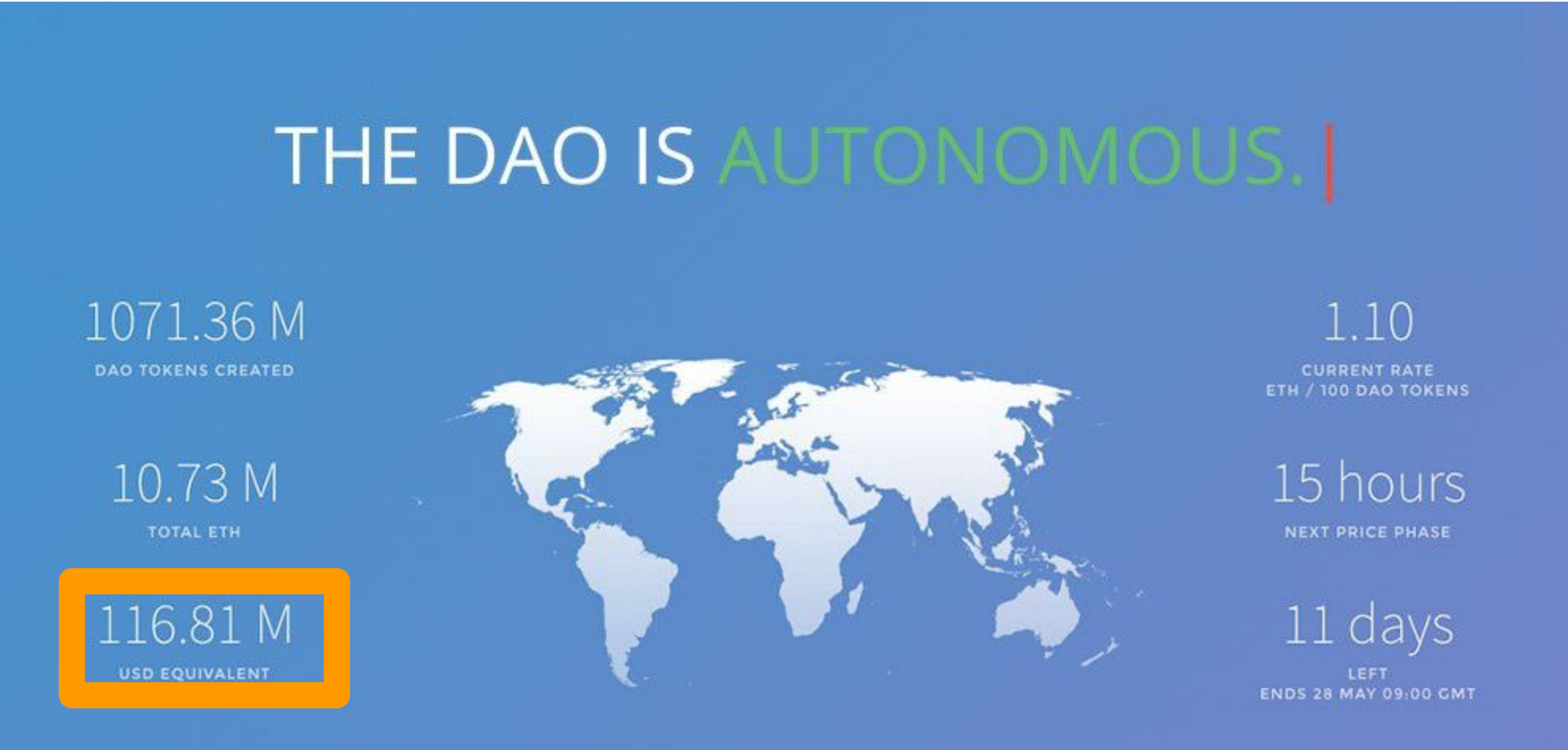
supports:

- Z-Wave
- Zigbee
- Bluetooth LE

Slock Power Switch

In Progress (with partners)

- Slock Door Lock
- Slock Bike Lock
- Slock Pad Lock
- Slock Car Lock



THE DAO IS AUTONOMOUS.

1071.36 M
DAO TOKENS CREATED

10.73 M
TOTAL ETH

116.81 M
USD EQUIVALENT

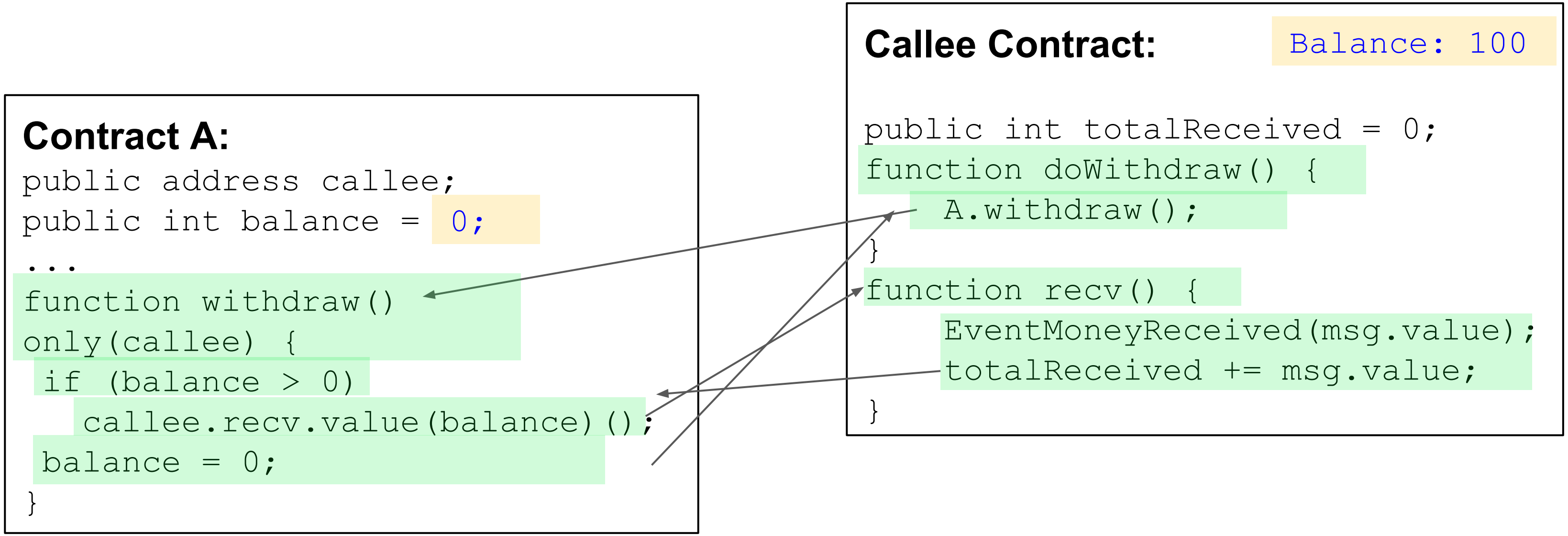
1.10
CURRENT RATE
ETH / 100 DAO TOKENS

15 hours
NEXT PRICE PHASE

11 days
LEFT
ENDS 28 MAY 09:00 GMT

Raised ~150 million dollars in ~ 1 month

Re-entrancy hazards in Ethereum



Contract A:

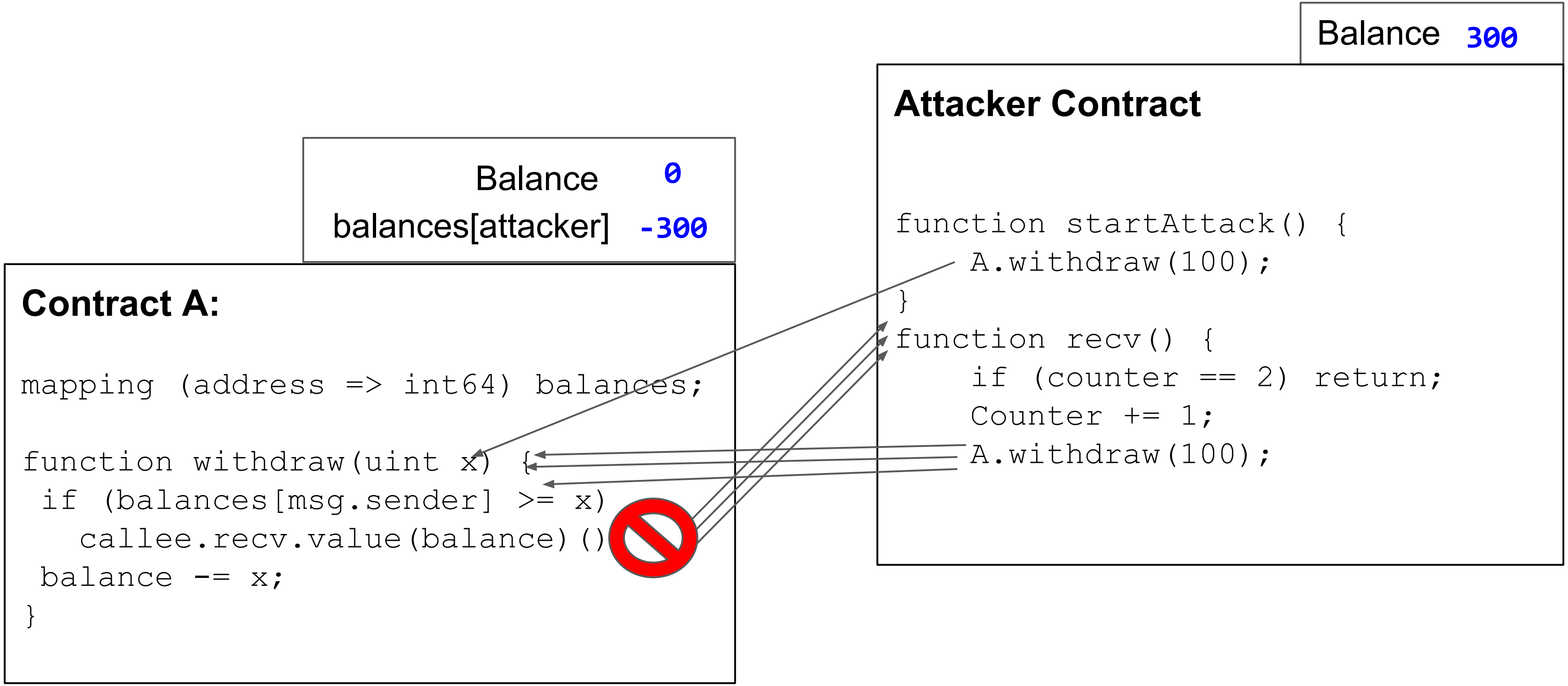
```
public address callee;
public int balance = 0;
...
function withdraw()
only(callee) {
 if (balance > 0)
   callee.recv.value(balance)();
 balance = 0;
}
```

Callee Contract: Balance: 100

```
public int totalReceived = 0;
function doWithdraw() {
   A.withdraw();
}
function recv() {
     EventMoneyReceived(msg.value);
     totalReceived += msg.value;
}
```

Done!   Callee withdraws 100.

Re-entrancy hazards in Ethereum



Balance 0

balances[attacker] -300

Contract A:

```
mapping (address => int64) balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x)
   callee.recv.value(balance)()
 balance -= x;
}
```

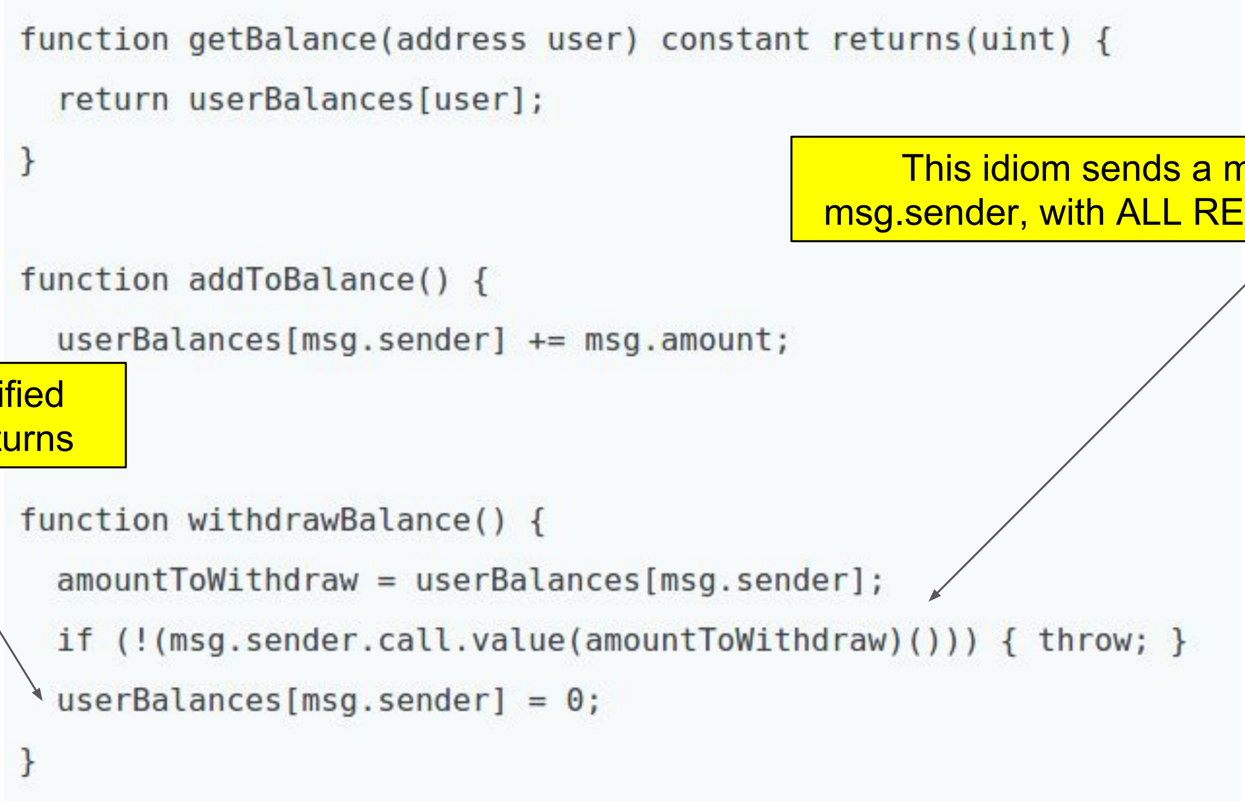
Balance 300

Attacker Contract

```
function startAttack() {
    A.withdraw(100);
}
function recv() {
    if (counter == 2) return;
    Counter += 1;
    A.withdraw(100);
```

http://vessenes.com/more-ethereum-attacks-race-to-empty-is-the-real-deal/

The “reentrancy” hazard in Ethereum

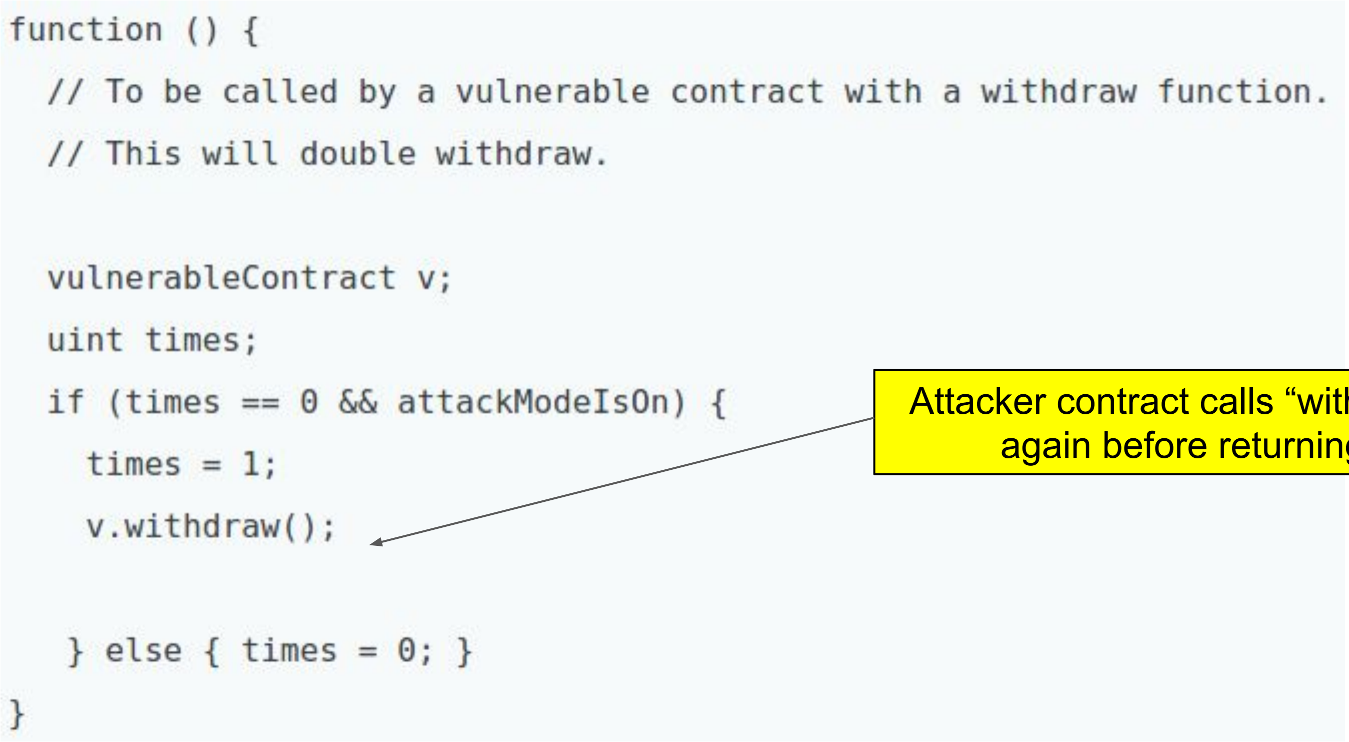


```
function getBalance(address user) constant returns(uint) {
  return userBalances[user];
}

function addToBalance() {
  userBalances[msg.sender] += msg.amount;
}

function withdrawBalance() {
  amountToWithdraw = userBalances[msg.sender];
  if (!(msg.sender.call.value(amountToWithdraw)())) { throw; }
  userBalances[msg.sender] = 0;
}
```

This idiom sends a message to msg.sender, with ALL REMAINING GAS

Storage is modified *after* callee returns

The attacker built a contract to drain the DAO



```
function () {
  // To be called by a vulnerable contract with a withdraw function.
  // This will double withdraw.

  vulnerableContract v;
  uint times;
  if (times == 0 && attackModeIsOn) {
    times = 1;
    v.withdraw();

   } else { times = 0; }
}
```

Attacker contract calls “withdraw” again before returning

Timeline and Aftermath of The DAO

- June 12: slock.it developers announce that the bug is found, but no funds at risk
- June 17 (Morning): attacker drains ⅓ of the DAO’s Ether ($50M) over 24 hrs

  Attacker’s funds were trapped in a subcontract for 40 days (July 27)
- June 17 (Evening): Eth Foundation proposes a “Soft Fork” to freeze the funds
- June 28: Cornell freshmen identify a flaw in the Soft Fork Proposal
- July 15 (Morning): Eth Foundation proposes a “Hard Fork” to recover funds
- July 15 (Evening): “Ethereum Classic” manifesto published on github
- July 19: “Hard Fork” moves funds from attacker’s contract to recovery contract

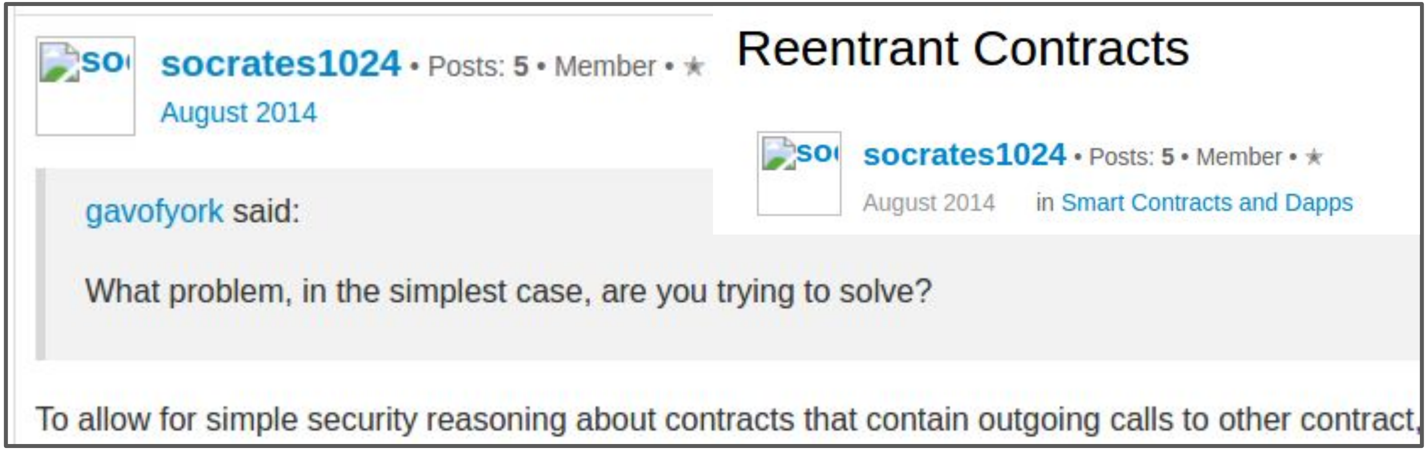
  Ethereum Classic blockchain survives and is traded on exchanges

Both Ethereum and Ethereum Classic are both around, reached new peaks

Reentrancy was known before the DAO

2014: Forum post on re-entrancy hazards

- Suggested mitigations at the language level



Reentrant Contracts

socrates1024 • Posts: 5 • Member • ★

August 2014 in Smart Contracts and Dapps

socrates1024 • Posts: 5 • Member • ★

August 2014

> gavofyork said:
>
> What problem, in the simplest case, are you trying to solve?

To allow for simple security reasoning about contracts that contain outgoing calls to other contract,

Reentrancy was known before the DAO

2014: Forum post on re-entrancy hazards

- Suggested mitigations at the language level

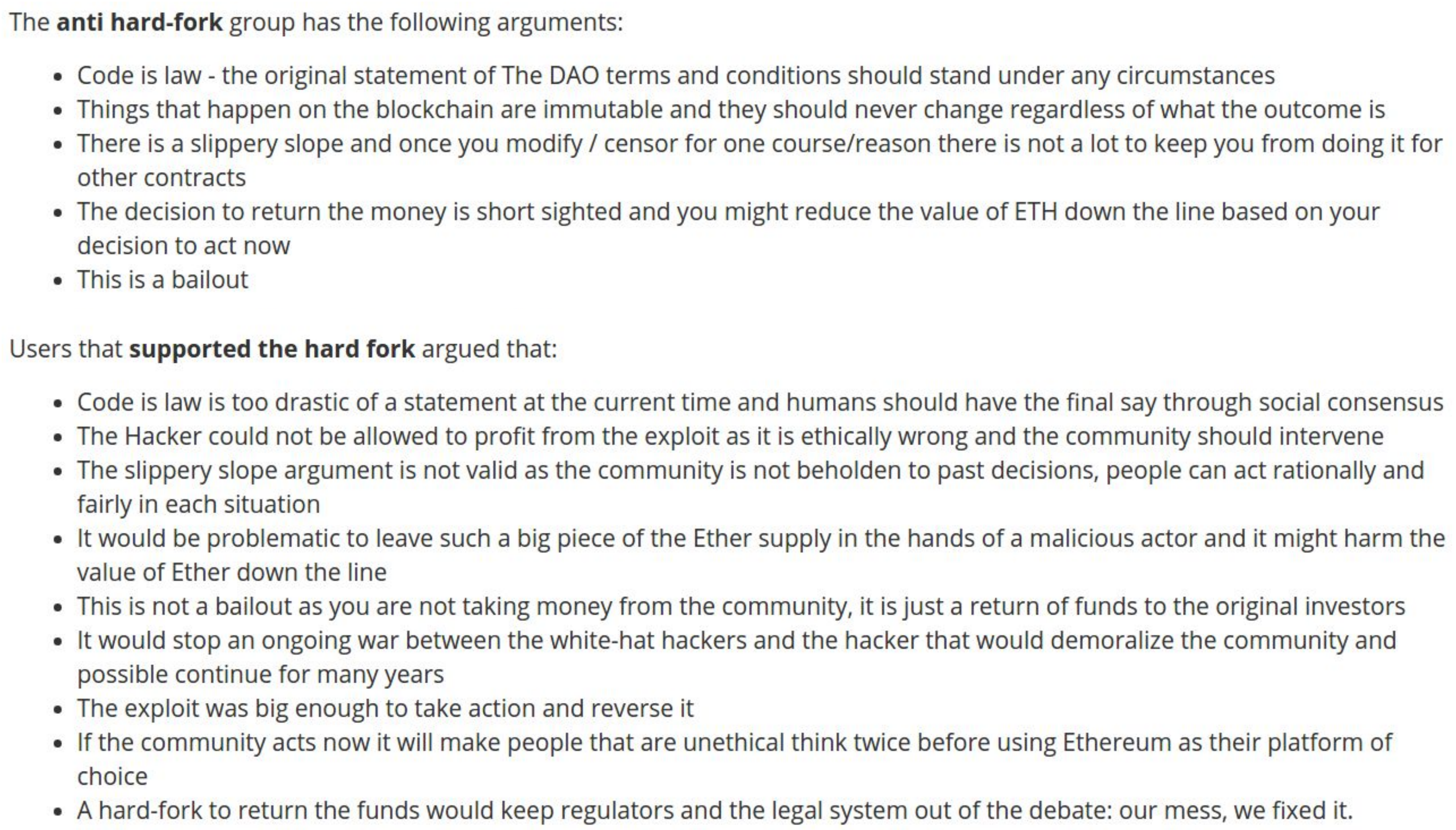
2015: ETH-commissioned report on EVM security

- Official ETH examples (crowdfund.se) also exhibit this flaw

(they happen not to be exploitable, but without showing why)

“the refund callback could make a new donation, triggering another refund cycle, potentially double-refunding the earlier contributions, or failing to refund later ones”

2016: The DAO happens anyway



The **anti hard-fork** group has the following arguments:

- Code is law - the original statement of The DAO terms and conditions should stand under any circumstances
- Things that happen on the blockchain are immutable and they should never change regardless of what the outcome is
- There is a slippery slope and once you modify / censor for one course/reason there is not a lot to keep you from doing it for other contracts
- The decision to return the money is short sighted and you might reduce the value of ETH down the line based on your decision to act now
- This is a bailout

Users that **supported the hard fork** argued that:

- Code is law is too drastic of a statement at the current time and humans should have the final say through social consensus
- The Hacker could not be allowed to profit from the exploit as it is ethically wrong and the community should intervene
- The slippery slope argument is not valid as the community is not beholden to past decisions, people can act rationally and fairly in each situation
- It would be problematic to leave such a big piece of the Ether supply in the hands of a malicious actor and it might harm the value of Ether down the line
- This is not a bailout as you are not taking money from the community, it is just a return of funds to the original investors
- It would stop an ongoing war between the white-hat hackers and the hacker that would demoralize the community and possible continue for many years
- The exploit was big enough to take action and reverse it
- If the community acts now it will make people that are unethical think twice before using Ethereum as their platform of choice
- A hard-fork to return the funds would keep regulators and the legal system out of the debate: our mess, we fixed it.

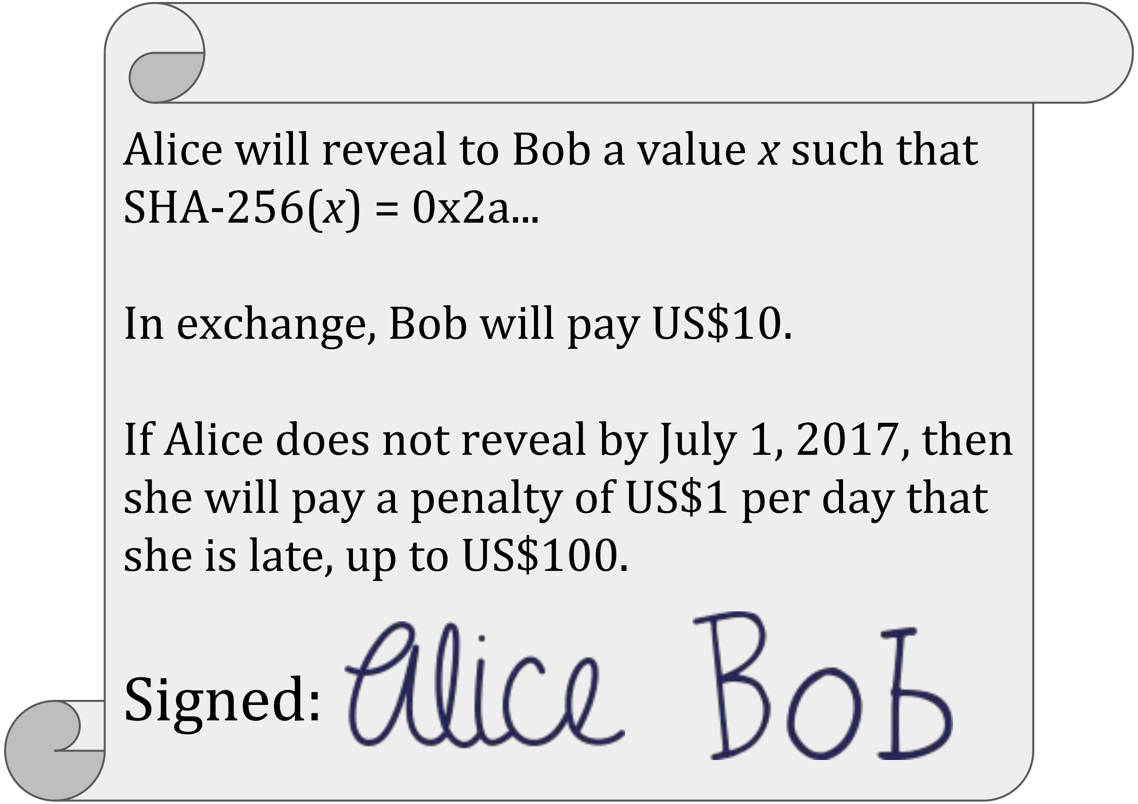
Smart contracts

“Smart contracts” conceptualized by Szabo in 1994

A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs.

-Nick Szabo “The Idea of Smart Contracts”

A “dumb contract” example: pay for a hash pre-image



Alice will reveal to Bob a value x such that SHA-256(x) = 0x2a...

In exchange, Bob will pay US$10.

If Alice does not reveal by July 1, 2017, then she will pay a penalty of US$1 per day that she is late, up to US$100.

Signed: Alice Bob

Traditional contracts vs. smart contracts

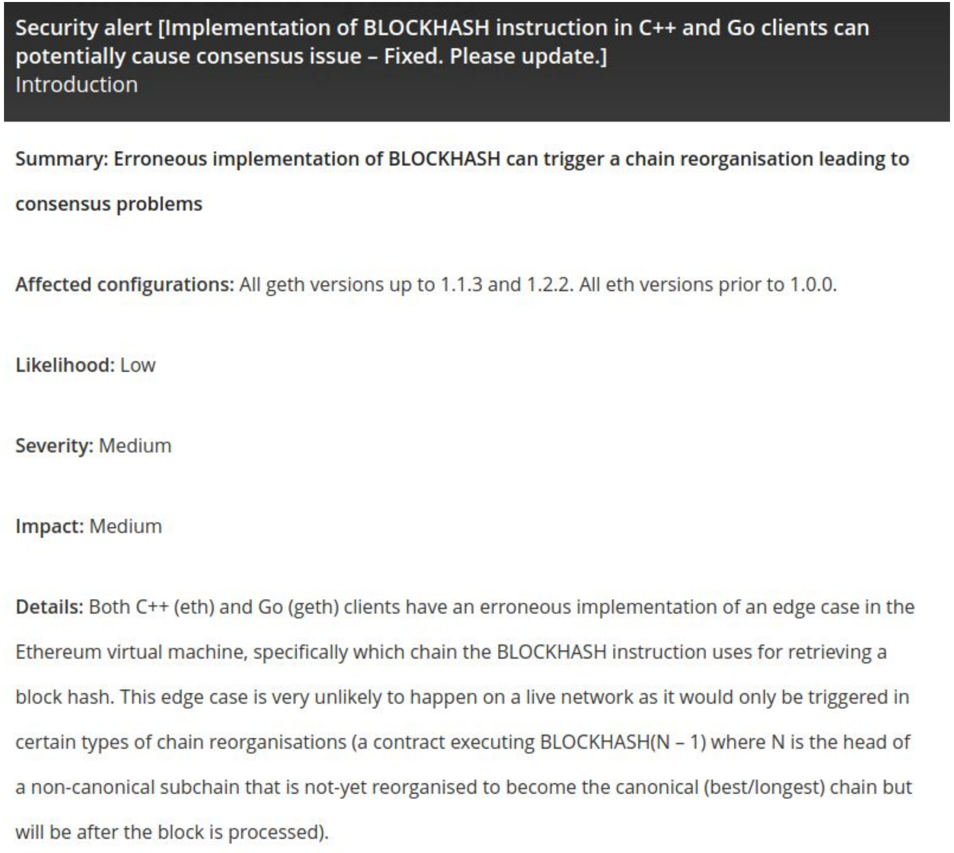
	Traditional	Smart
specification	Natural language + “legalese”	Code
assent	Signatures	Digital signatures
dispute resolution	Judges, arbitrators	Decentralized platform
nullification	By judges	????
payment	As specified	built-in
escrow	Trusted third party	built-in

Research challenges

Ethereum makes all data public

- Proposals:
  - Project Alchemy-exchange Eth for Zcash
  - SNARKs for token-issuing contracts
    - Acceleration within EVM?
  - Hawk: The blockchain model of cryptography and privacy-preserving smart contracts [Khosba et al. 2016]

Verifying consistency of Ethereum implementations



Security alert [Implementation of BLOCKHASH instruction in C++ and Go clients can potentially cause consensus issue – Fixed. Please update.]
Introduction

Summary: Erroneous implementation of BLOCKHASH can trigger a chain reorganisation leading to consensus problems

Affected configurations: All geth versions up to 1.1.3 and 1.2.2. All eth versions prior to 1.0.0.

Likelihood: Low

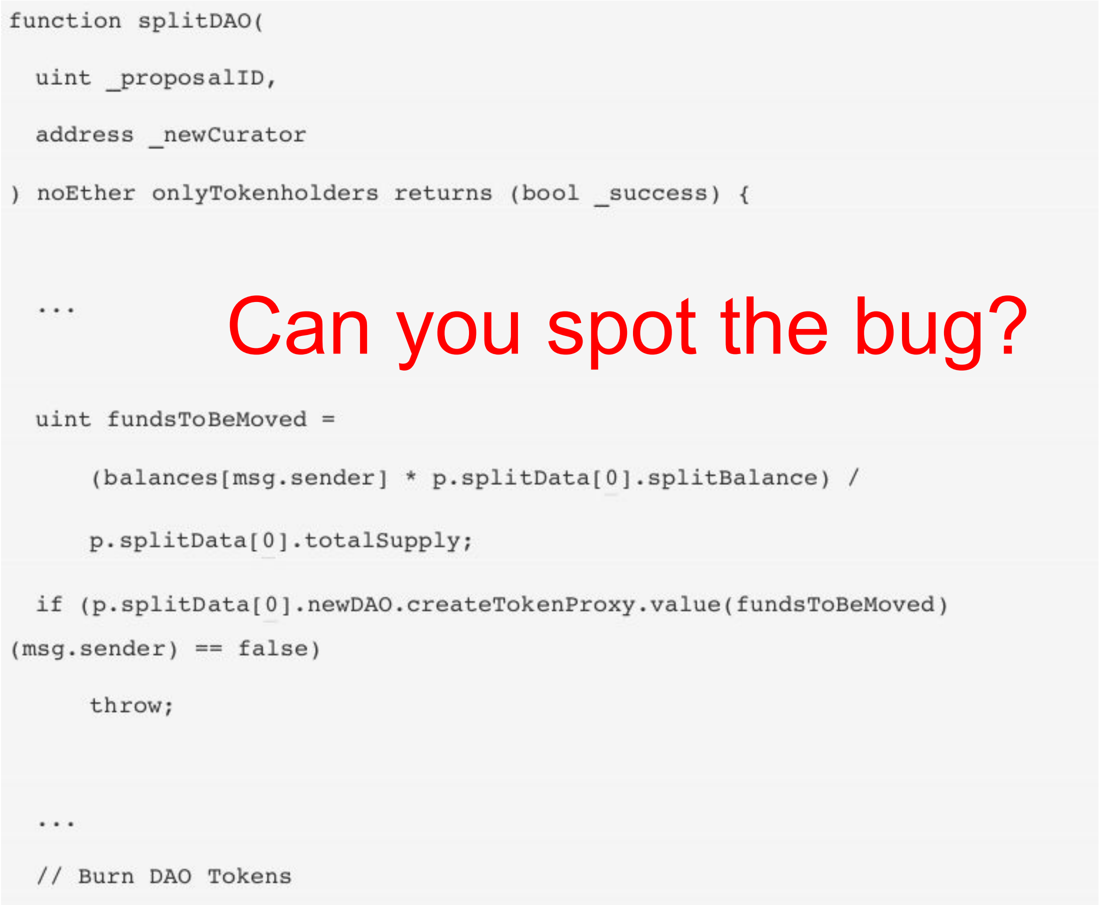
Severity: Medium

Impact: Medium

Details: Both C++ (eth) and Go (geth) clients have an erroneous implementation of an edge case in the Ethereum virtual machine, specifically which chain the BLOCKHASH instruction uses for retrieving a block hash. This edge case is very unlikely to happen on a live network as it would only be triggered in certain types of chain reorganisations (a contract executing BLOCKHASH(N – 1) where N is the head of a non-canonical subchain that is not-yet reorganised to become the canonical (best/longest) chain but will be after the block is processed).

- *at least* 7 EVM implementations
  - C++, Go, Haskell, Java, Python, Ruby, Rust
- Inconsistency can be exploited to cause a hard fork!

Verifying correctness of Ethereum contracts



```
function splitDAO(

  uint _proposalID,

  address _newCurator

) noEther onlyTokenholders returns (bool _success) {


  ...

  uint fundsToBeMoved =

      (balances[msg.sender] * p.splitData[0].splitBalance) /

      p.splitData[0].totalSupply;

  if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)
(msg.sender) == false)

      throw;


  ...

  // Burn DAO Tokens
```

Can you spot the bug?

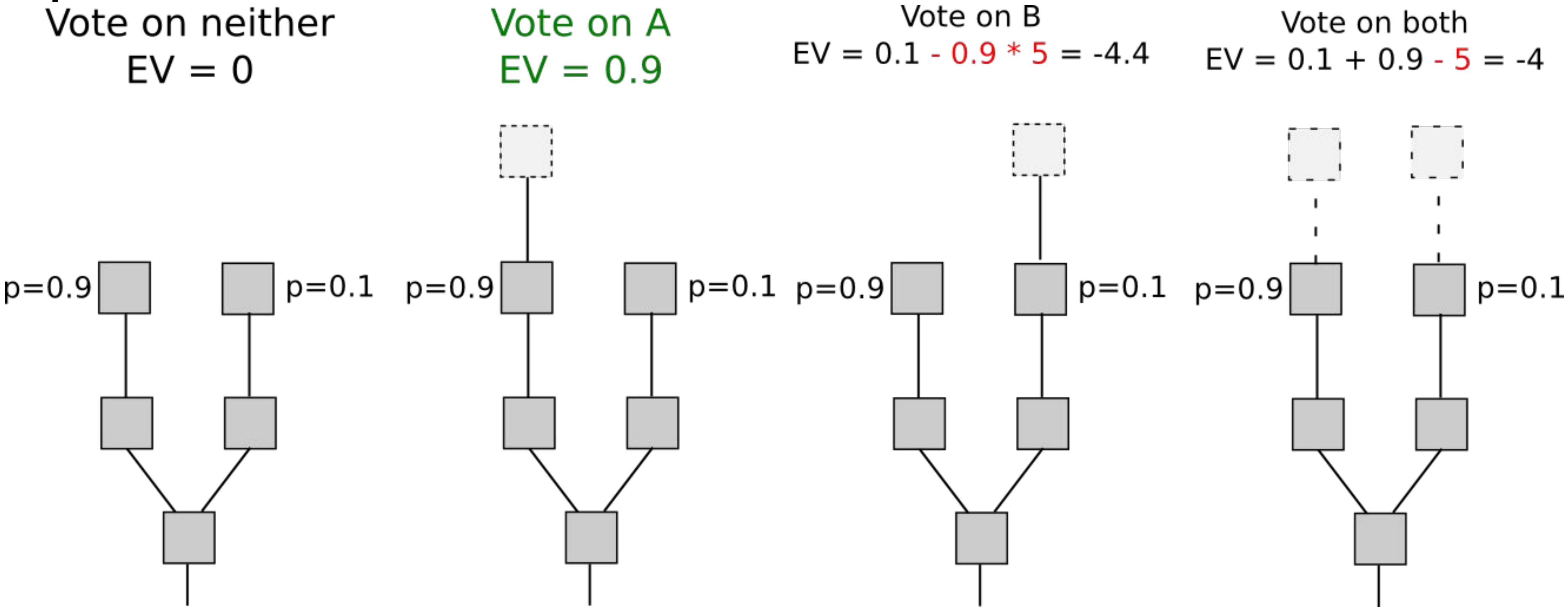
Ethereum scaling limited as nodes verify all contracts

- Can’t always determine which state a tx will change

- Goal is to support *sharding*
  - Most nodes track only a random subset of contracts
  - Super nodes process cross-shard communication
  - Details get complicated... great research topic!

https://github.com/ethereum/wiki/wiki/Sharding-FAQ

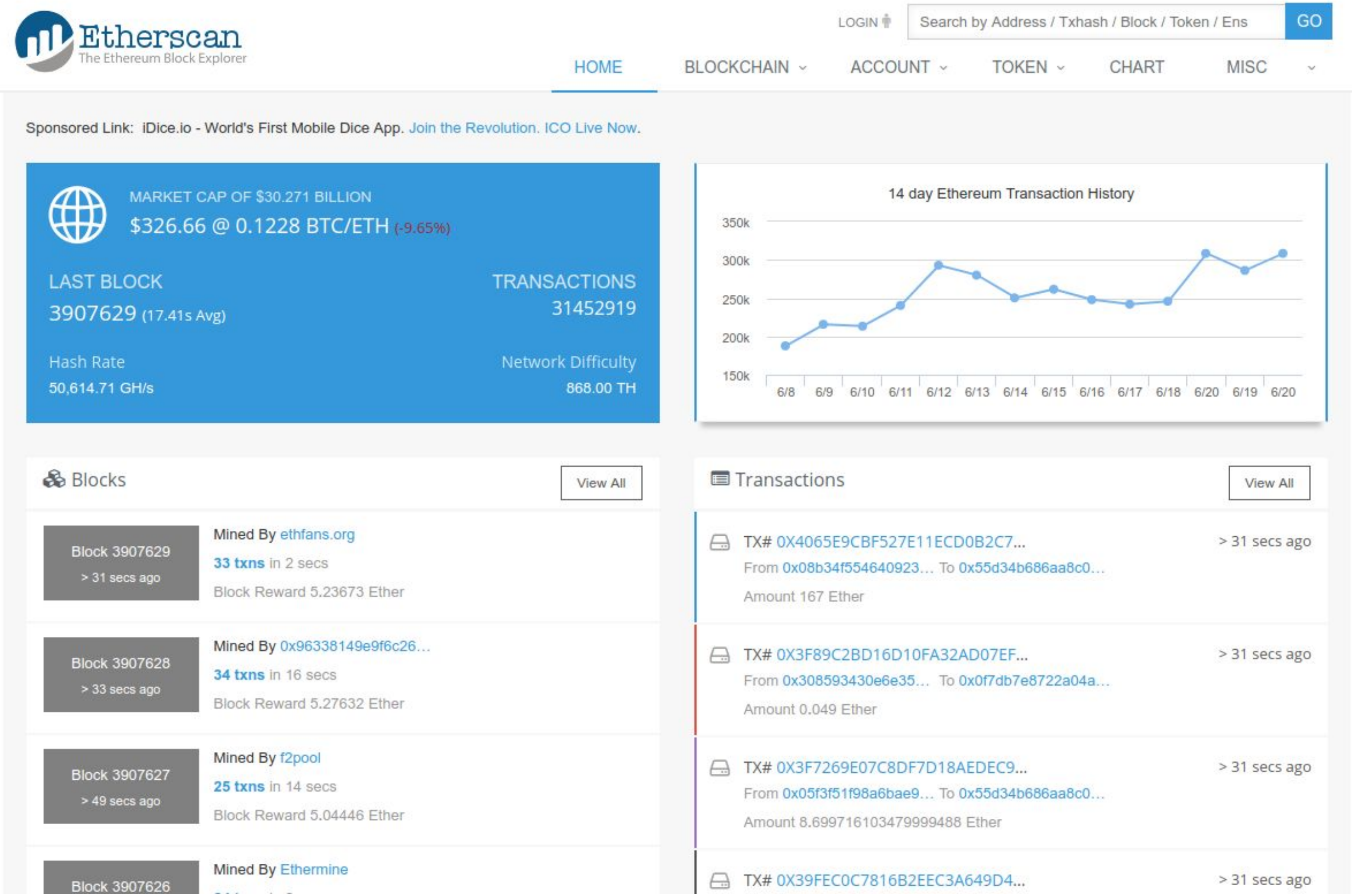
Ethereum has long held plans to adopt proof-of-stake



https://medium.com/@VitalikButerin/safety-under-dynamic-validator-sets-ef0c3bbdf9f6

Explore more!

Explore the blockchain: https://etherscan.io



Etherscan
The Ethereum Block Explorer
LOGIN
Search by Address / Txhash / Block / Token / Ens
GO
HOME
BLOCKCHAIN
ACCOUNT
TOKEN
CHART
MISC

Sponsored Link: iDice.io - World's First Mobile Dice App. Join the Revolution. ICO Live Now.

MARKET CAP OF $30.271 BILLION
$326.66 @ 0.1228 BTC/ETH (-9.65%)

LAST BLOCK
3907629 (17.41s Avg)

TRANSACTIONS
31452919

Hash Rate
50,614.71 GH/s

Network Difficulty
868.00 TH

14 day Ethereum Transaction History

Blocks

Block 3907629
> 31 secs ago
Mined By ethfans.org
33 txns in 2 secs
Block Reward 5.23673 Ether

Block 3907628
> 33 secs ago
Mined By 0x96338149e9f6c26...
34 txns in 16 secs
Block Reward 5.27632 Ether

Block 3907627
> 49 secs ago
Mined By f2pool
25 txns in 14 secs
Block Reward 5.04446 Ether

Block 3907626
Mined By Ethermine

Transactions

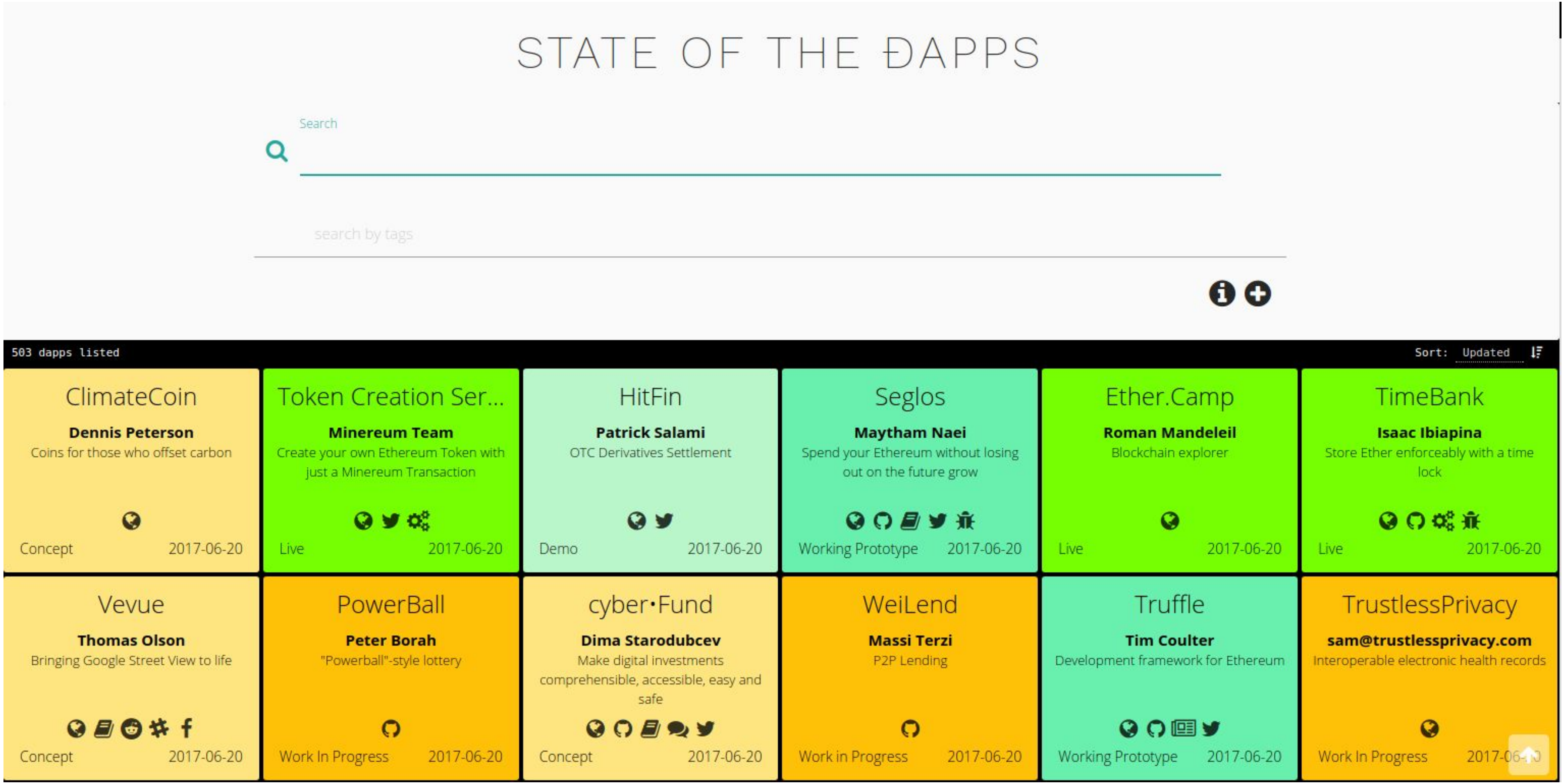
TX# 0X4065E9CBF527E11ECD0B2C7... > 31 secs ago
From 0x08b34f554640923... To 0x55d34b686aa8c0...
Amount 167 Ether

TX# 0X3F89C2BD16D10FA32AD07EF... > 31 secs ago
From 0x308593430e6e35... To 0x0f7db7e8722a04a...
Amount 0.049 Ether

TX# 0X3F7269E07C8DF7D18AEDEC9... > 31 secs ago
From 0x05f3f51f98a6bae9... To 0x55d34b686aa8c0...
Amount 8.699716103479999488 Ether

TX# 0X39FEC0C7816B2EEC3A649D4... > 31 secs ago

State of the Dapps: https://dapps.ethercasts.com/



STATE OF THE ĐAPPS

Search

search by tags

503 dapps listed

Sort: Updated

ClimateCoin
Dennis Peterson
Coins for those who offset carbon
Concept 2017-06-20

Token Creation Ser...
Minereum Team
Create your own Ethereum Token with just a Minereum Transaction
Live 2017-06-20

HitFin
Patrick Salami
OTC Derivatives Settlement
Demo 2017-06-20

Seglos
Maytham Naei
Spend your Ethereum without losing out on the future grow
Working Prototype 2017-06-20

Ether.Camp
Roman Mandeleil
Blockchain explorer
Live 2017-06-20

TimeBank
Isaac Ibiapina
Store Ether enforceably with a time lock
Live 2017-06-20

Vevue
Thomas Olson
Bringing Google Street View to life
Concept 2017-06-20

PowerBall
Peter Borah
"Powerball"-style lottery
Work In Progress 2017-06-20

cyber•Fund
Dima Starodubcev
Make digital investments comprehensible, accessible, easy and safe
Concept 2017-06-20

WeiLend
Massi Terzi
P2P Lending
Work in Progress 2017-06-20

Truffle
Tim Coulter
Development framework for Ethereum
Working Prototype 2017-06-20

TrustlessPrivacy
sam@trustlessprivacy.com
Interoperable electronic health records
Work In Progress 2017-06-20