

# Introduction to Ethereum

Joseph Bonneau

FOSAD 2017

Bertinoro, Italy

# Schedule

## **Monday: Intro to Ethereum**

- Ethereum system design
- Overview of solidity
- Overview of applications

## **Tuesday: Practical Exercise**

- Solidity coding tutorial

## **Wednesday: Security issues for Ethereum Development**

- Secure smart contract programming
- Classic security bugs: reentrancy, unchecked sends
- DAO case study

# Smart contracts & Bitcoin

# “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  
 $\text{SHA-256}(x) = 0x2a\dots$

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

# Recall: BTC contains a simple scripting language

```
"tx_out":[  
  {  
    "value":"10.12287097",  
    "scriptPubKey":"OP_DUP OP_HASH160 69e...3d42e  
OP_EQUALVERIFY OP_CHECKSIG"  
  },  
  ...  
]
```

Output “addresses” are really *scripts*

```
OP_DUP  
OP_HASH160  
<69e02e18...>  
OP_EQUALVERIFY  
OP_CHECKSIG
```



# Input “addresses” are *a/so* scripts



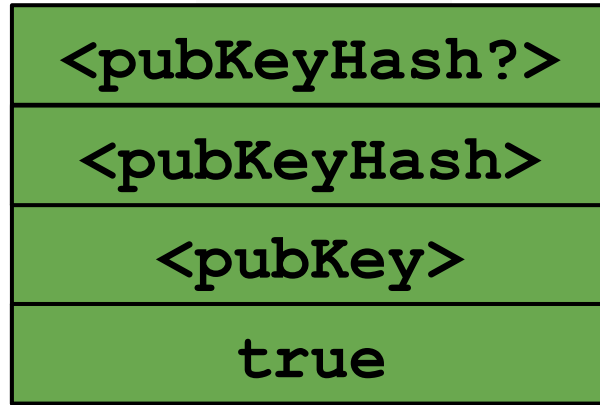
TO VERIFY: Concatenated script must execute completely with no errors

# Bitcoin script instructions

256 opcodes total (15 disabled, 75 reserved)

- Arithmetic
- If/then
- Logic/data handling
- Crypto!
  - Hashes
  - Signature verification
  - Multi-signature verification

# Bitcoin script execution example



```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG
```

# Bitcoin scripting language (“Script”)

## Design goals

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Support for cryptography
- Stack-based
- No looping
  - Not Turing-complete
- Time/memory usage bound by program size

I am not impressed

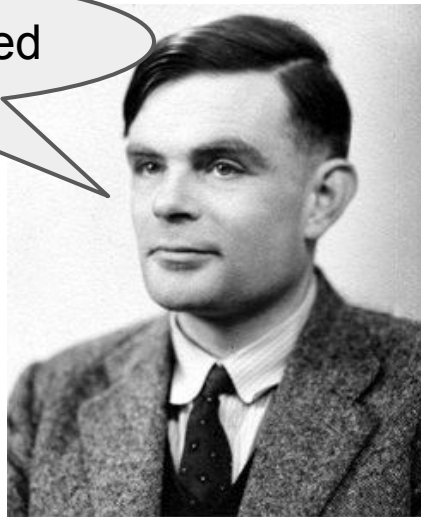


image via Jessie St. Amand

# Some useful contracts can be done in Bitcoin

- Proof-of-burn
- MULTISIG/access control trees
- Pay-for-hash-preimage
  - Multi-party lotteries
  - Atomic cross-chain currency exchange
- Micropayment/payment channels
  - Greatly improved with OP\_CHECKLOCKTIME

# Extending Bitcoin functionality

# Bitcoin script left developers wanting more

By adding a few opcodes to Bitcoin script, what if we could support:

- Distributed naming (Namecoin)
- Options, financial derivatives (OpenBazaar, MasterCoin)
- Prediction markets (Futurecoin)
- Open-ended, user-defined functionality?

# Namecoin was the first fork of Bitcoin

Goal: distributed naming, similar functionality to DNS

3 new opcodes:

- NAME\_NEW
- NAME\_FIRST\_UPDATE
- NAME\_UPDATE



# Namecoin introduces three new opcodes

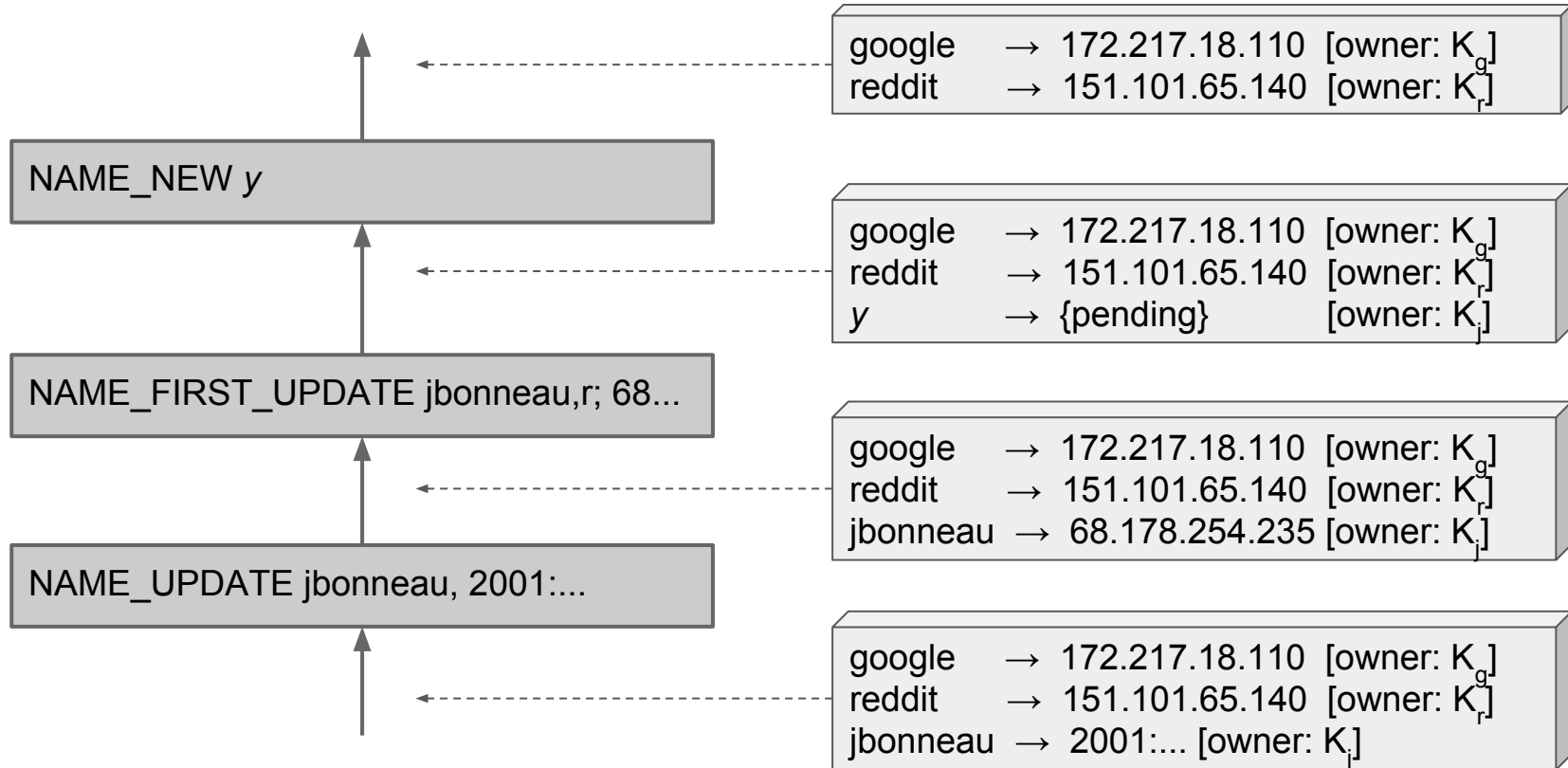
NAME\_NEW: H(r, "jbonneau")

} 12 block delay (frontrunning)

NAME\_FIRST\_UPDATE: r, "jbonneau", {"ip" : "68.178.254.235"}

NAME\_UPDATE: r, "jbonneau", {"ip6" : "2001:4860:0:1001::68"}

# Namecoin introduces new global state



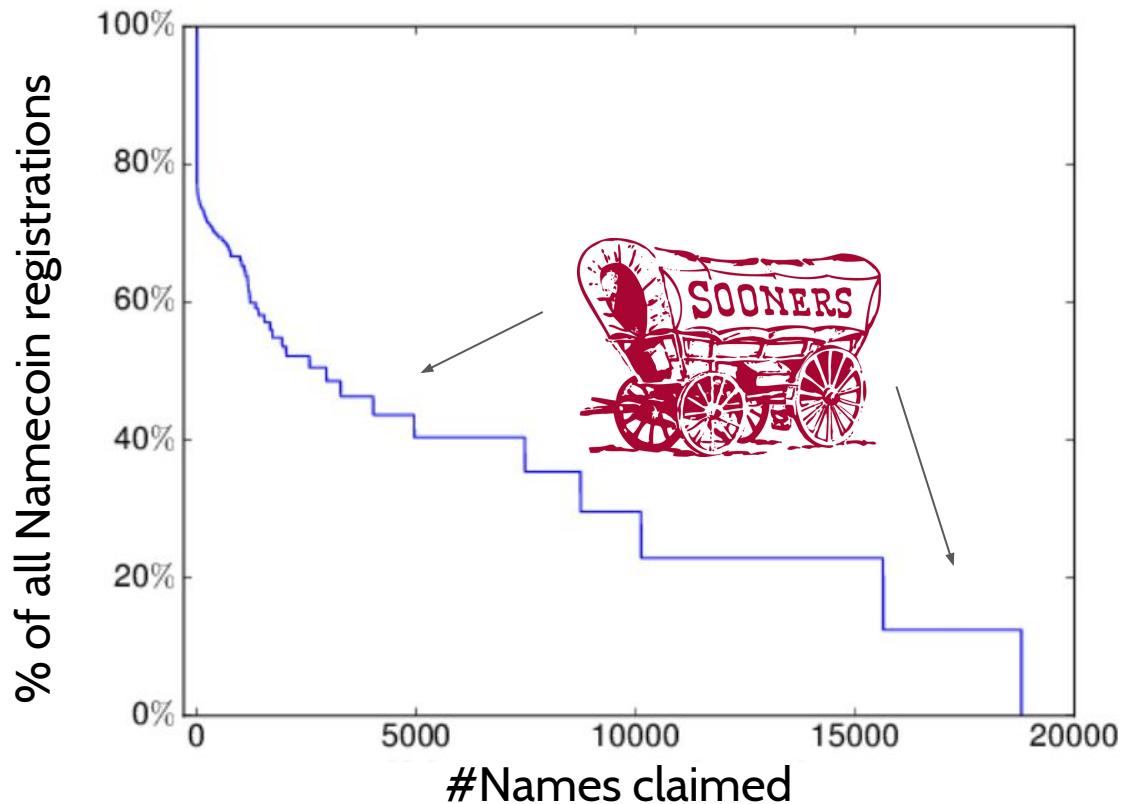
# Namecoin introduces new fees, incentives

## How much does it cost

You can register a .bit domain for 0.02 NMC using the Namecoin software *namecoind* or *Namecoin-Qt*. The fee consists of two parts: a) the registration fee, and b) the transaction fees for the `name_new` and the `name_firstupdate` transactions.

Command	Registration fee	Transaction fee	Summary	Notes
<code>name_new</code>	0.01 NMC	0.005 NMC	Pre-order a domain name	<b>You still don't own the domain!</b>
<code>name_firstupdate</code>	0.00 NMC	0.005 NMC	Finalize the registration. The name becomes public.	You own the domain during the next 36000 blocks (six months, approx.)
<code>name_update</code>	0.00 NMC	0.005 NMC	Renew, update, or transfer a name	Gives you another ~6 months of ownership

# Side note: Namecoin got the incentives badly wrong



**An empirical study of Namecoin and lessons for decentralized namespace design**

Harry Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau and Arvind Narayanan. *WEIS 2015*

# Recap: several requirements for new functionality

new addition	purpose	in Namecoin	in Futurecoin
Global state	Track app-specific data	name → value map	list of markets, bets in each market
Opcodes	Express updates to global state	NAME_NEW etc.	OPEN_MARKET etc.
Fees	Limit computation & reads/writes to global state	Registration fees to limit squatting, maintenance fees	transaction fees per open market, exchange

# Replicated State Machines

# Recap: Bitcoin itself implicitly defines state

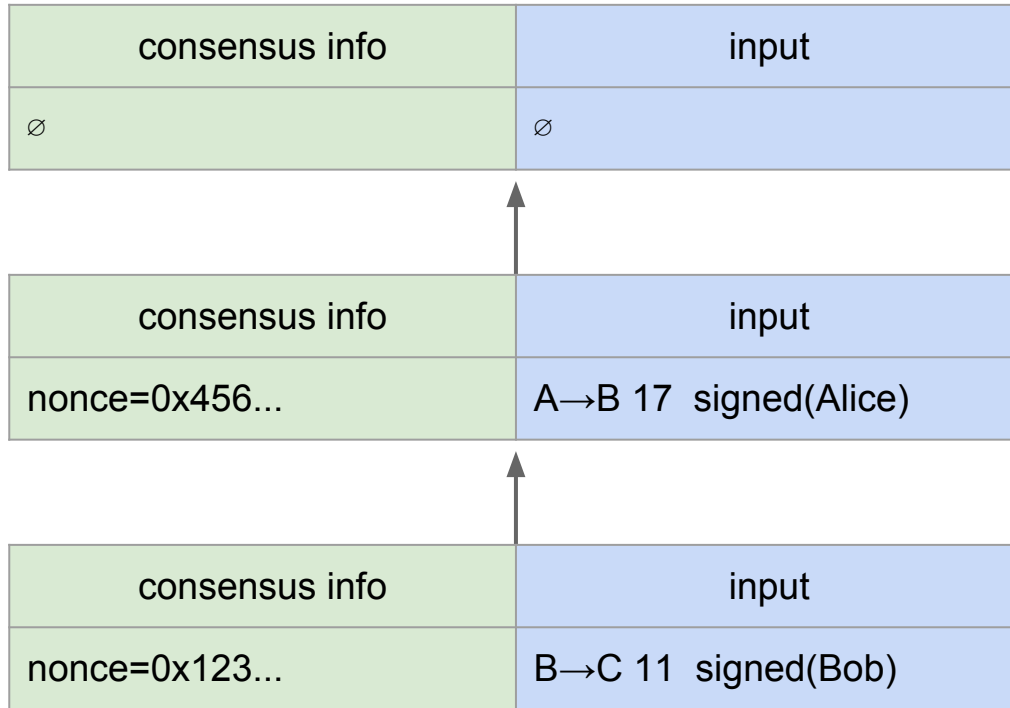
new addition	purpose	in Bitcoin	
Global state	Track app-specific data	UTXO set	This state is implicit
Opcodes	Express updates to global state	transactions	Bitcoin scripts only succeed/fail. No side effects on global state
Fees	Limit computation & reads/writes to global state	not required	Miners can produce blocks which are very costly to verify

*Replicated state machines* are the classic abstraction

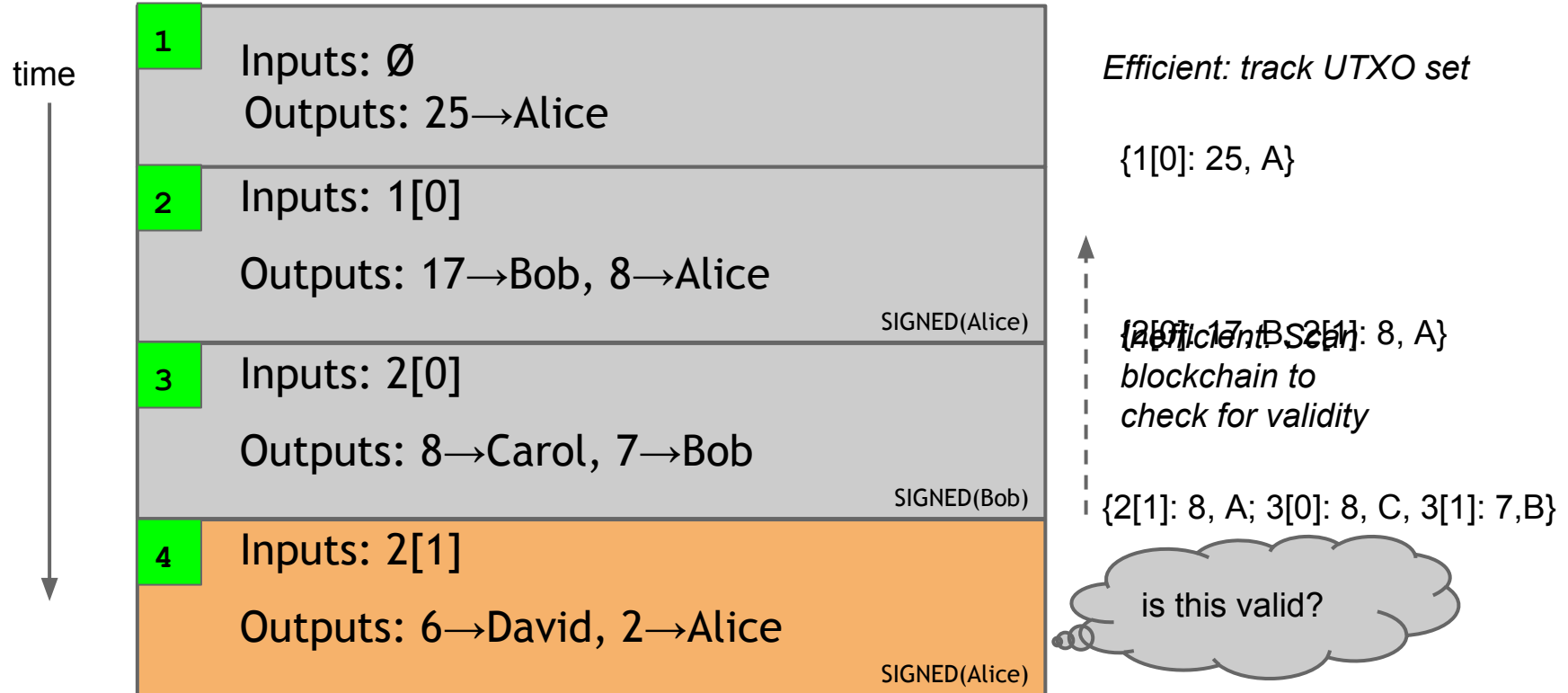
- Set of possible states  $\mathbf{S}$
- Set of possible inputs  $\mathbf{I}$
- Set of possible outputs  $\mathbf{O}$
- Transition function  $f: \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S} \times \mathbf{O}$
- Start state  $s \in \mathbf{S}$  (genesis block)



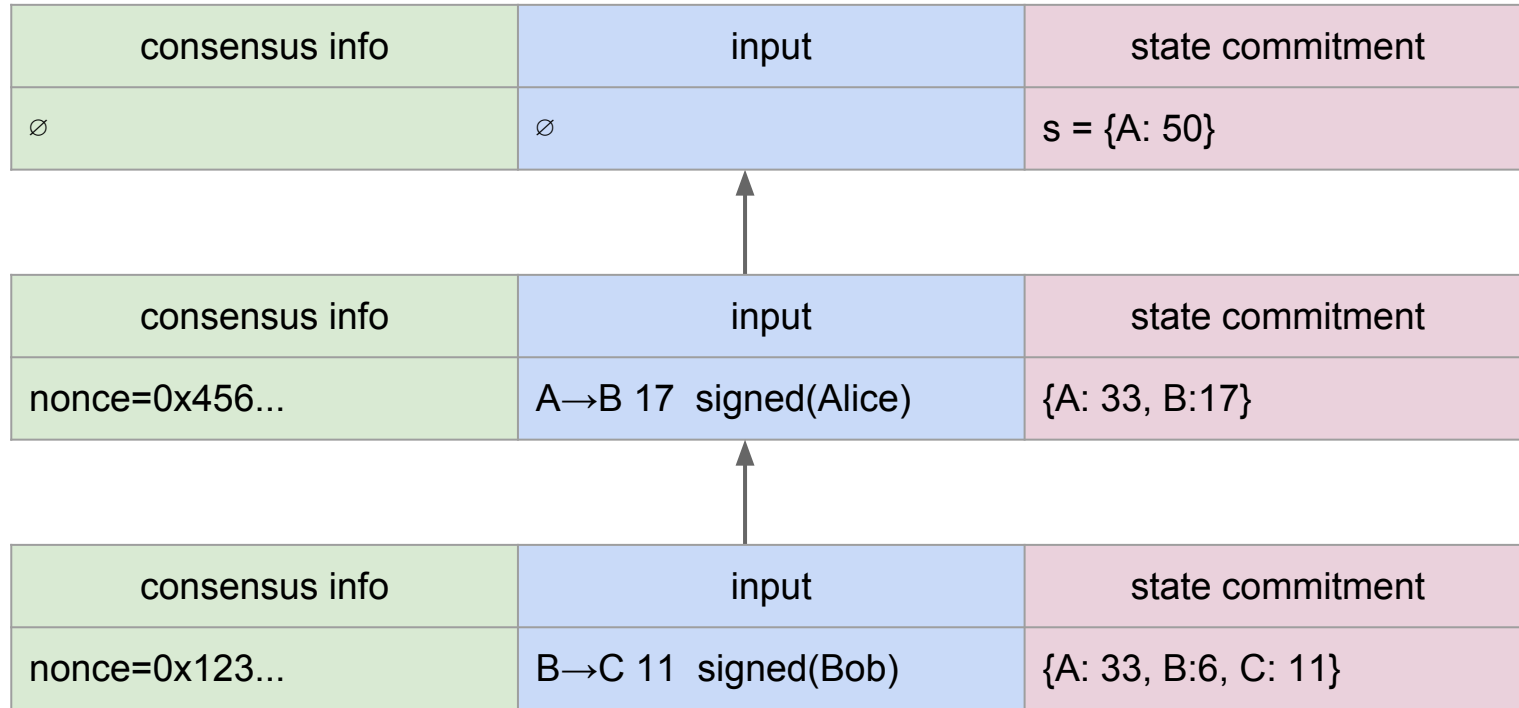
# “Blockchain” is an ordered list of inputs w/consensus



# “State” is really just a compression of history



# Blockchains may include explicit *state commitments*



# Explicit state commitments offer many advantages

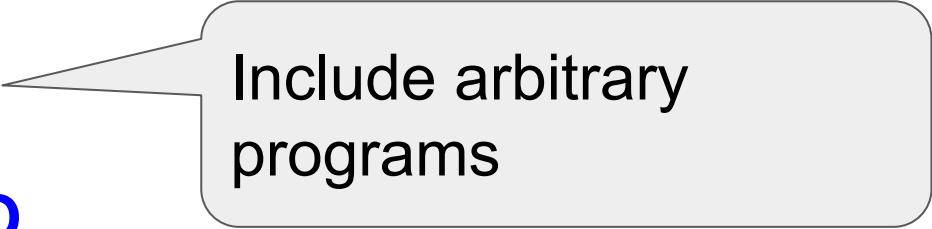
consensus info	input	state commitment
nonce=0x123...	B→C 11 signed(Bob)	{A: 33, B:6, C: 11}

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

Ethereum: *A* universal RSM

# To get Turing-completeness:

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



Include arbitrary programs

- Transition function  $f: \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S} \times \mathbf{O}$

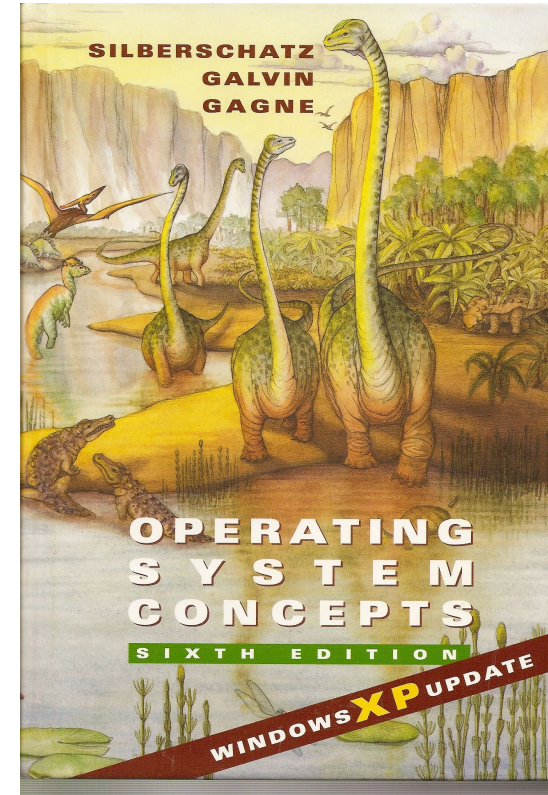


Interpret programs

- Start state  $s \in \mathbf{S}$  (genesis block)

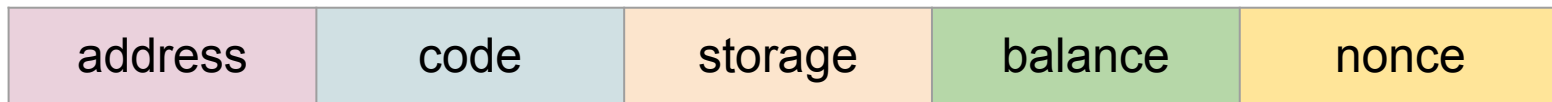
# Universality brings on classic OS problems

- What state can a tx change?
  - *memory protection*
- How many resources can a contract use?
  - *resource contention*

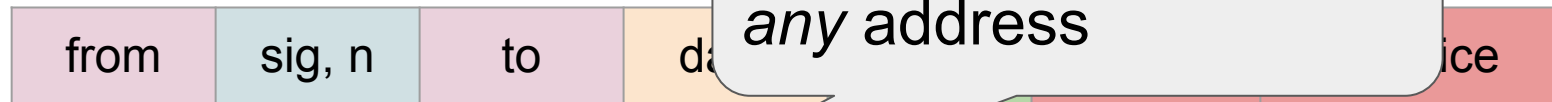


# Ethereum in one slide

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



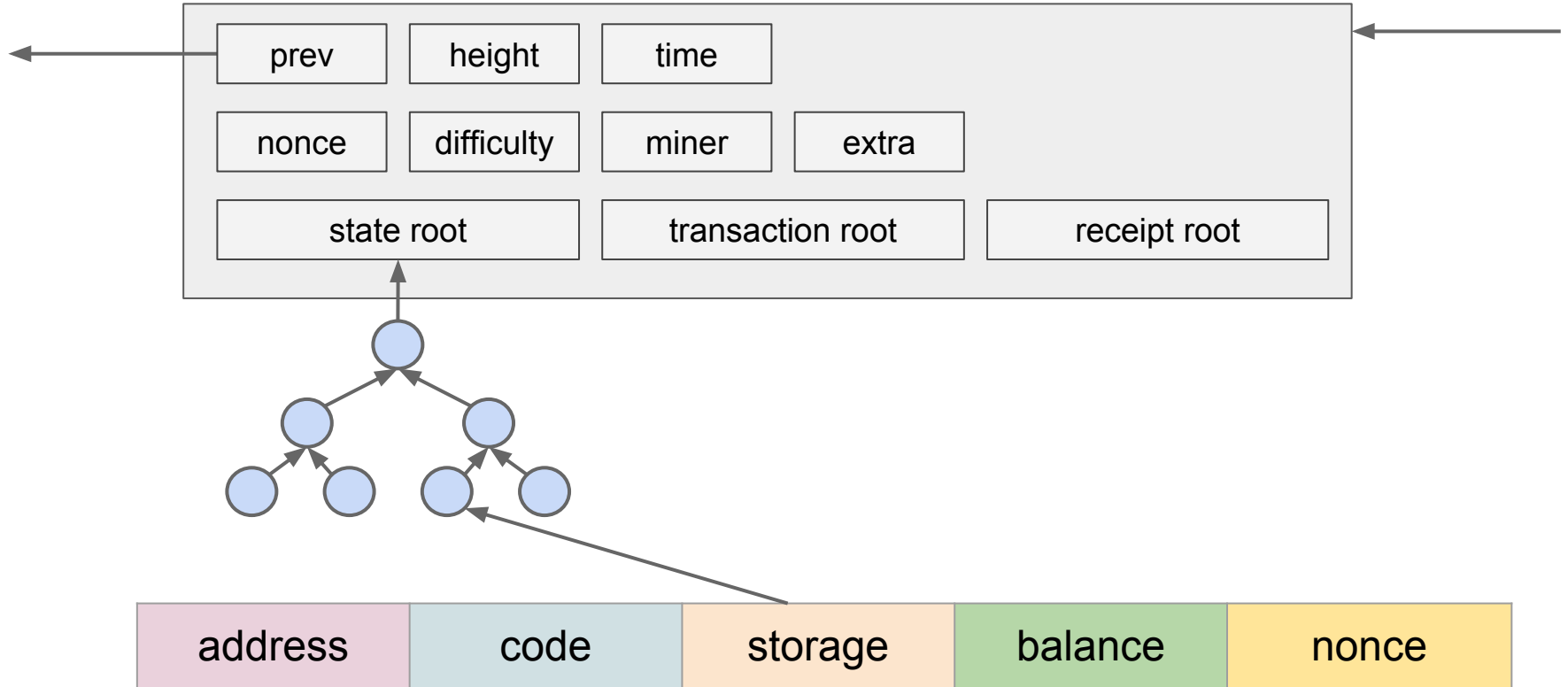
- Inputs **I** (transactions)



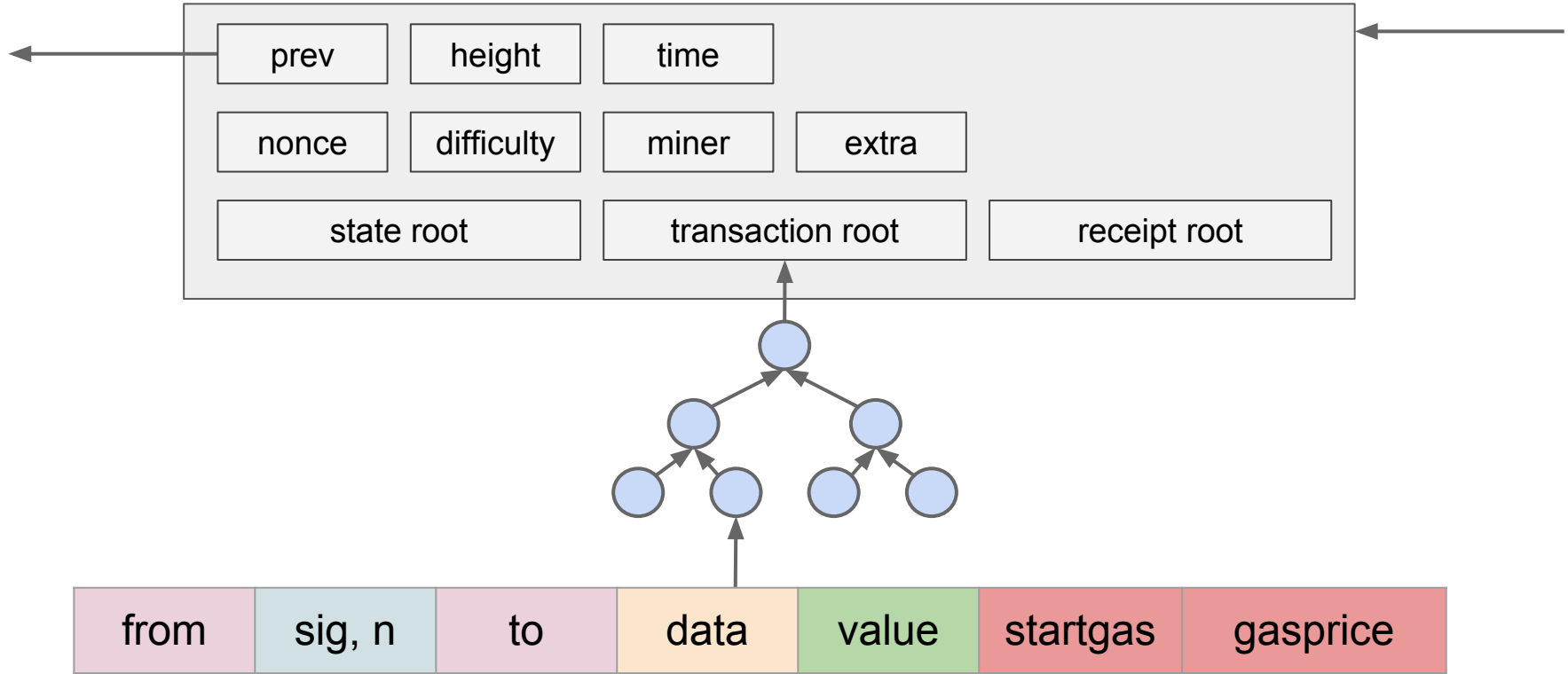
- Transition **f**:
  - validate signature
  - run `to.code(from, data, value, startgas, gasprice)`
- Start state:  $\emptyset$



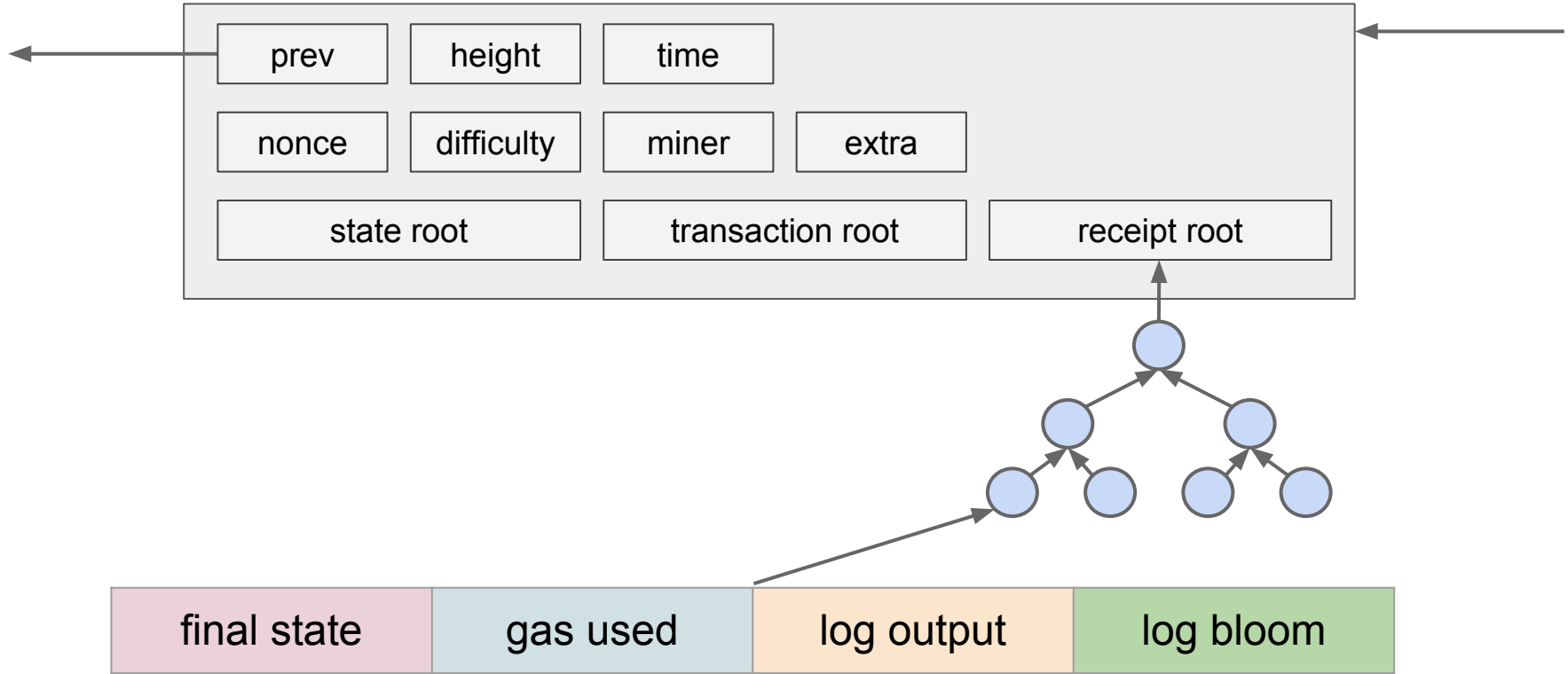
# The full\* Ethereum blockchain structure



# The full\* Ethereum blockchain structure



# The full\* Ethereum blockchain structure



# Ethereum addresses can be *accounts* or *contracts*

address	code	storage	balance	nonce
---------	------	---------	---------	-------

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

Note: no UTXOs in Ethereum

**Volatile fields**

# Three\* types of transaction in Ethereum

type	from	sig, n	to	data	value	startgas	gasprice
create	creator	sig, n	∅	code	start_bal	53000	?

# Three\* types of transaction in Ethereum

type	from	sig, n	to	data	value	startgas	gasprice
create	creator	sig, n	∅	code	start_bal	53000	?
send	sender	sig, n	receiver	∅	transfer_val	30000	?

# Three\* types of transaction in Ethereum

type	from	sig, n	to	data	value	startgas	gasprice
create	creator	sig, n	∅	code	start_bal	53000	?
send	sender	sig, n	receiver	∅	transfer_val	30000	?
call	caller	sig, n	contract	f, args	transfer_val	?	?

**Example:**

NameCoin in Ethereum





```
contract Namespace {

    struct NameEntry {
        address owner;
        bytes32 value;
    }

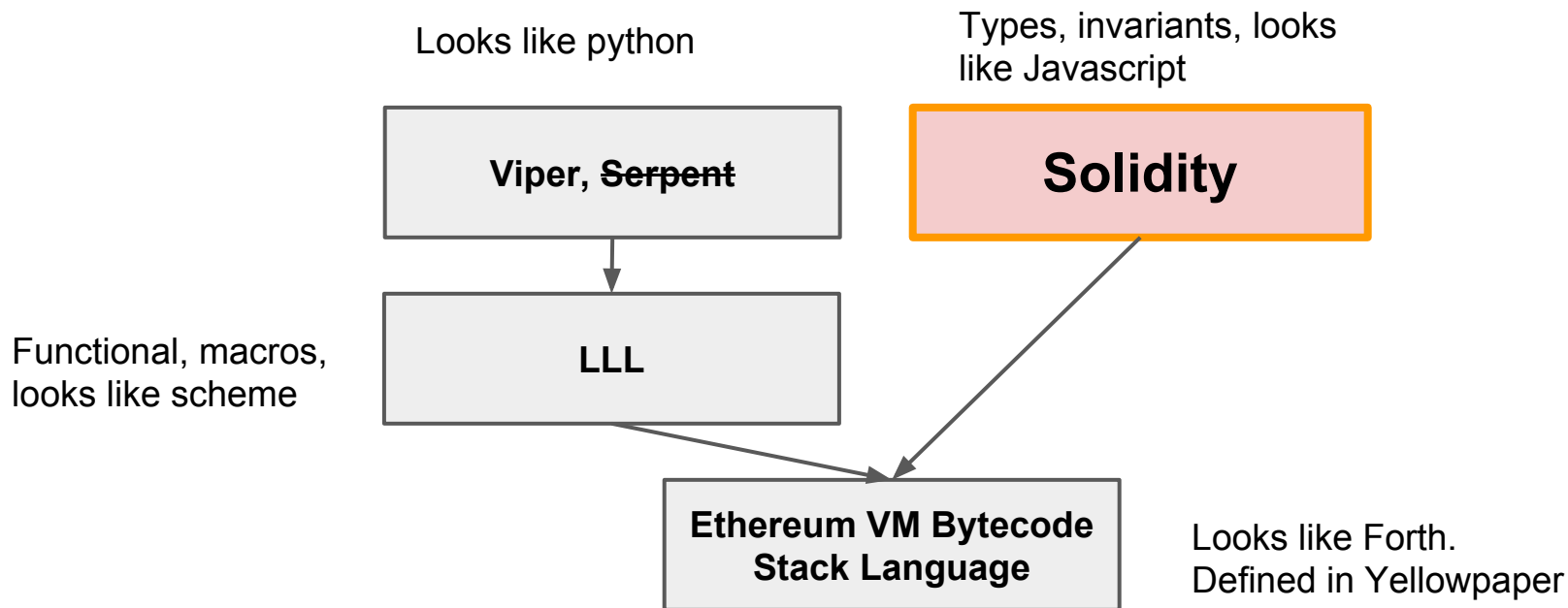
    uint32 constant REGISTRATION_COST = 100;
    uint32 constant UPDATE_COST = 10;
    mapping(bytes32 => NameEntry) data;

    function nameNew(bytes32 hash){
        if (msg.value >= REGISTRATION_COST){
            data[hash].owner = msg.sender;
        }
    }

    function nameUpdate(bytes32 name, bytes32 newValue, address newOwner){
        bytes32 hash = sha3(name);
        if (data[hash].owner == msg.sender && msg.value >= UPDATE_COST){
            data[hash].value = newValue;
            if (newOwner != 0){
                data[hash].owner = newOwner;
            }
        }
    }

    function nameLookup(bytes32 name){
        return data[sha3(name)];
    }
}
```

# Ethereum code written in Solidity, compiled to EVM



# 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 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  $2^{256}$  bits!
- arranged in 256-bit words
- all memory is zero-initialized

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

# 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

# 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	A		B
value	$x' \leq B.\text{balance}$		
data	(as specified)		
startgas	$s' \leq \text{gas remaining}$		
storage updated	C	B	



# 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



# Efficient map commitments

# How can Ethereum commit to so much state?

**Recall:** storage is a  $\{0,1\}^{256} \rightarrow \{0,1\}^{256}$  map

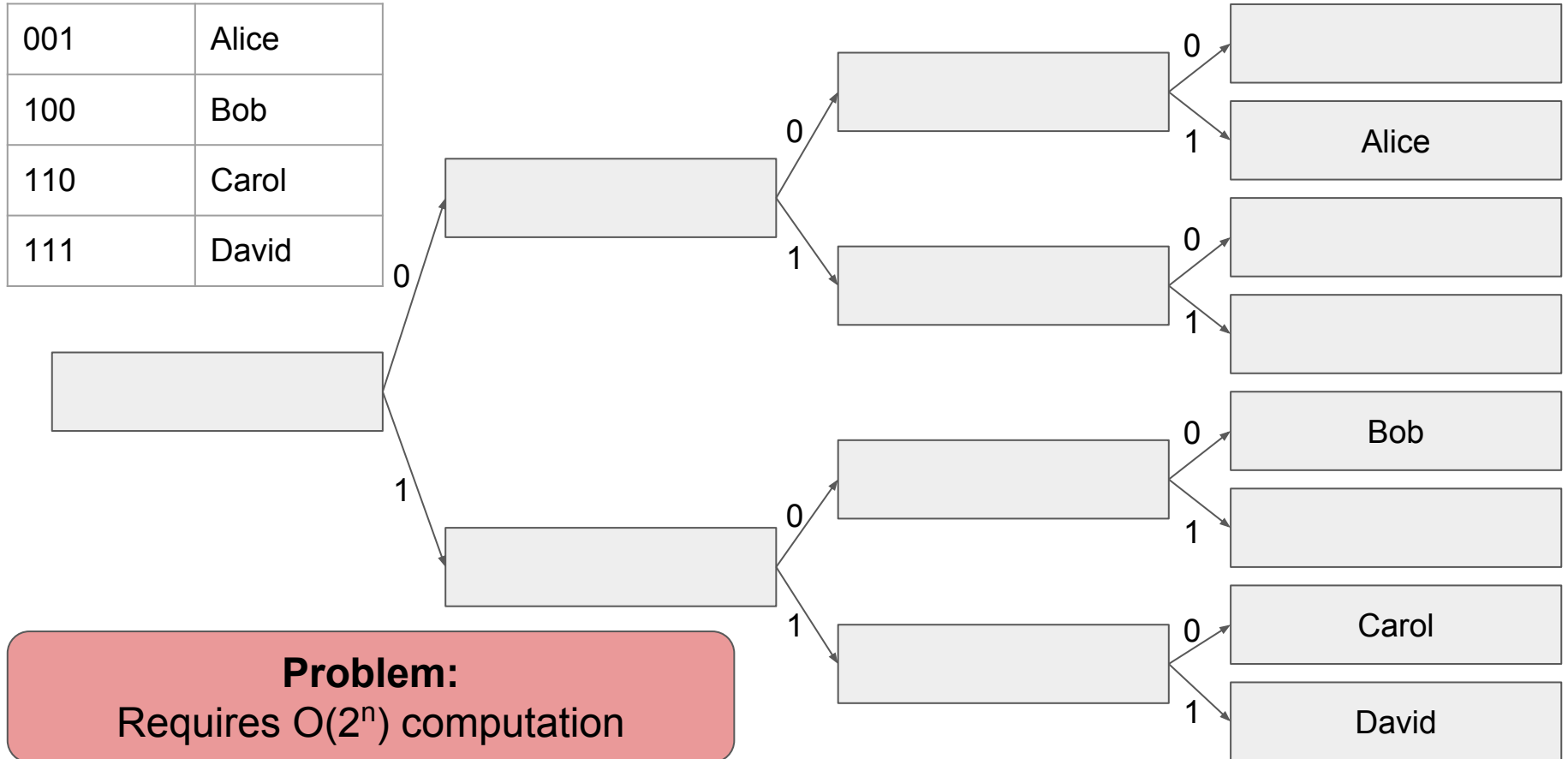
## Requirements:

- $2^n$  keys supported
- $k$  keys with a non-zero value
- $O(1)$  commitment size
- $O(\lg k)$  update cost
- $O(\lg k)$  proofs (even for zero values)

### Key insight:

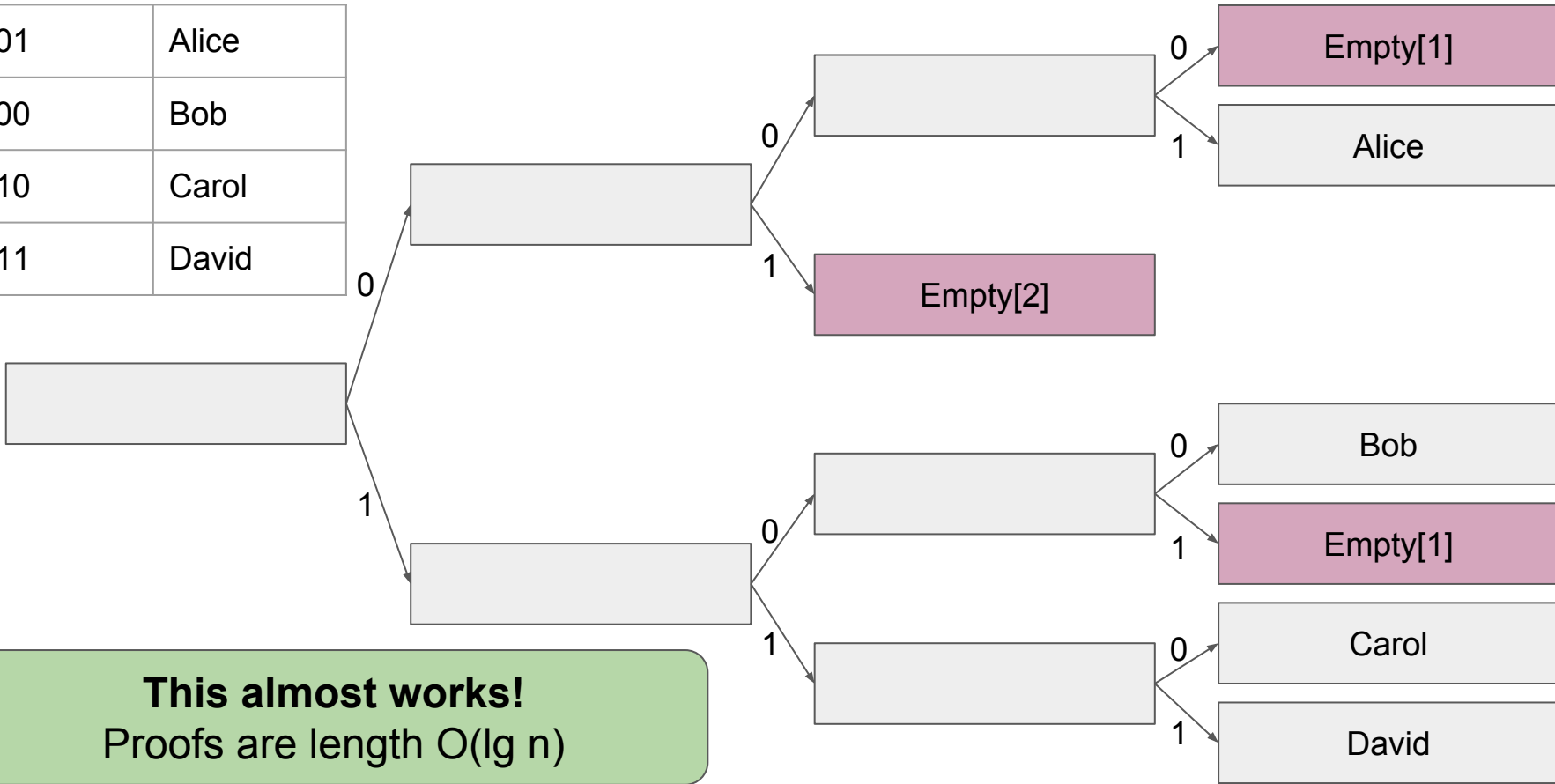
Optimize for efficient storage of zeroized values

# Non-solution: use a binary tree, paths encode address



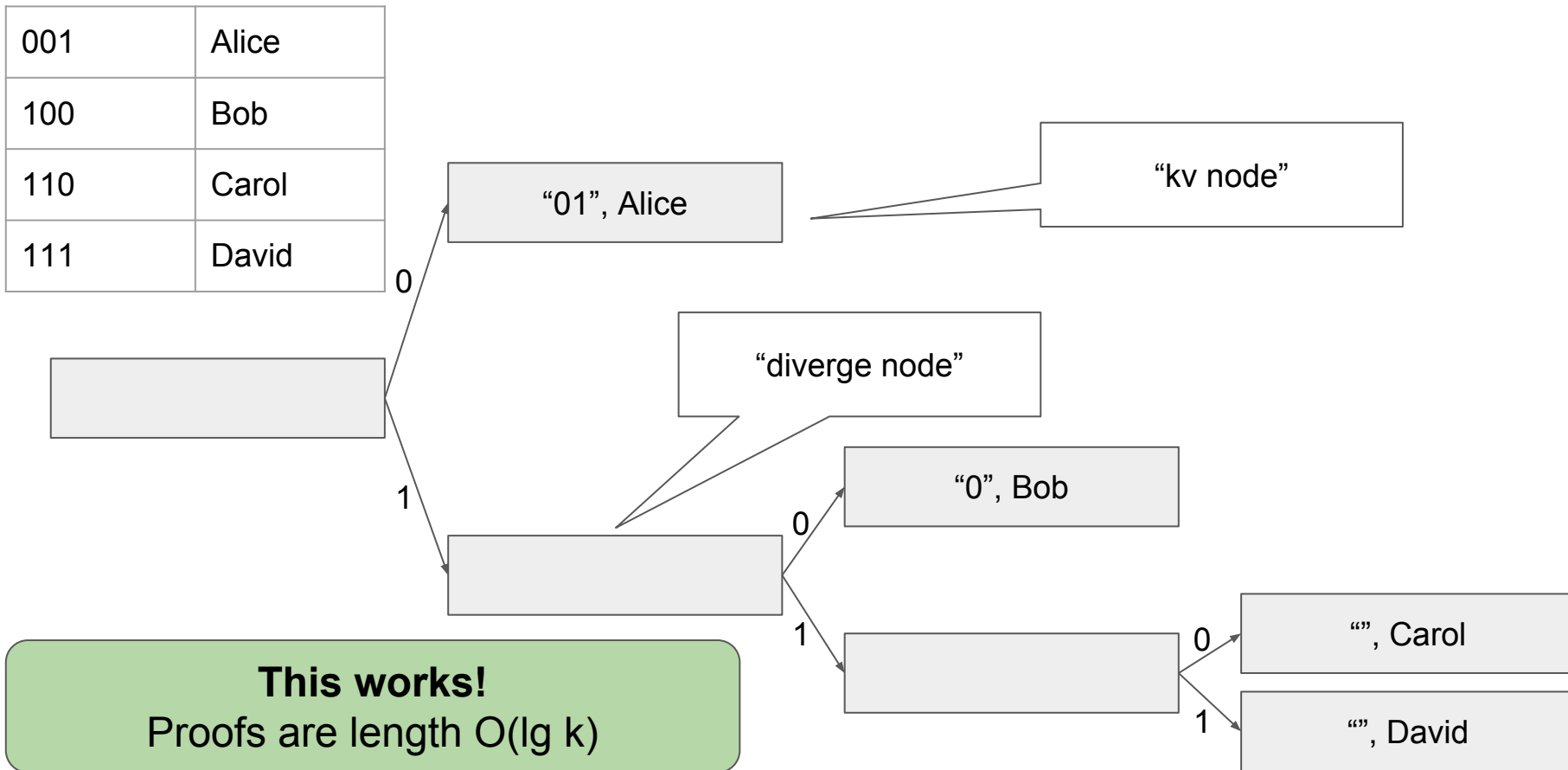
## Optimization #1: pre-compute all sizes of empty tree

001	Alice
100	Bob
110	Carol
111	David



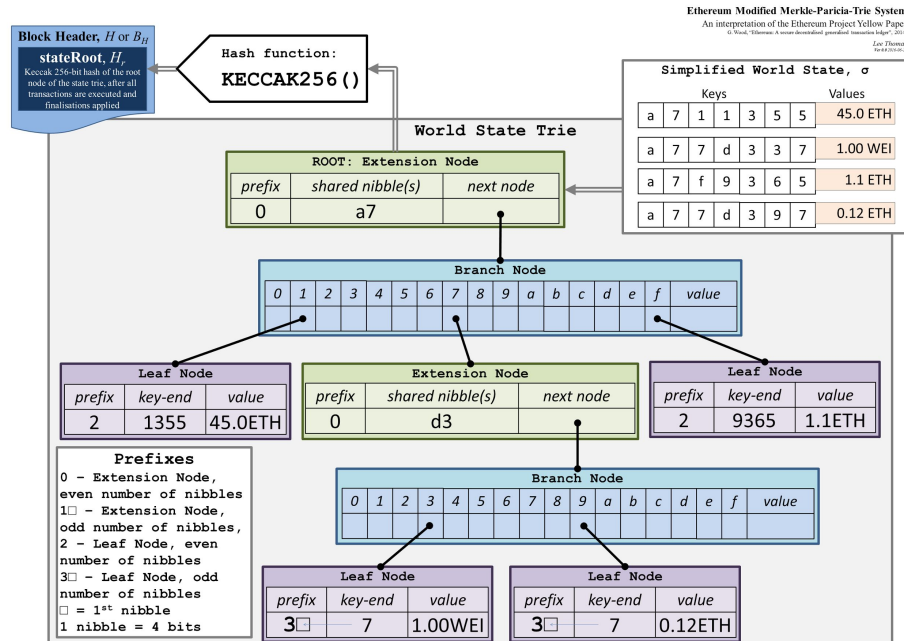
**This almost works!**  
Proofs are length  $O(\lg n)$

# Optimization #2: shrink all single subtrees; store prefix



# Ethereum's “Merkle Patricia tree” has a few quirks

- Used in two main places
  - overall state
  - per-contract storage
- 16-ary
  - proofs ~4x longer
- paths can be < 256 bits
  - “1”, “01” are distinct
  - Really  $2^{257}-1$  addresses



# Gas and transaction limits



# Ethereum is like Ryanair: pay to board, then keep paying

**Table - Optional fees**

Fees subject to VAT on Italian, French, Spanish, Portuguese, German, Greek, Polish & Romanian domestic routes at applicable government rates.

	Online price ⓘ	Airport price ⓘ	More info
<b>Credit card fee</b>	2% of transaction total	2% of transaction total	The credit card fee is based on the total value of your transaction, and is charged at the standard 2%.
<b>Priority with extra legroom seats</b> Row 1, 2 (D,E,F) & 16-17 (incl. Priority Boarding) from* (per flight)	€11.00 / £11.00	€11.00 / £11.00	Please note that an increased charge is applicable for allocated seating & priority boarding on selected routes.
<b>Priority Seats</b> Rows 2 (A,B,C) - 5 (incl. Priority Boarding) from* (per flight)	€7.00 / £7.00	€7.00 / £7.00	Please note that an increased charge is applicable for allocated seating & priority boarding on selected routes.
	€2.00 / £2.00	€3.00 / £3.00	Please note that an increased charge is applicable for allocated seating & priority boarding on selected routes.



# Gas in Ethereum is a necessary evil

- All miners must evaluate all transactions
  - limit computation cost
- All miners must store all state
  - limit storage use
- Short-cut the halting problem
  - Finite GAS\_LIMIT ensures all programs halt



## Observe:

Bitcoin also employs a (crude) means to pay for resources consumed

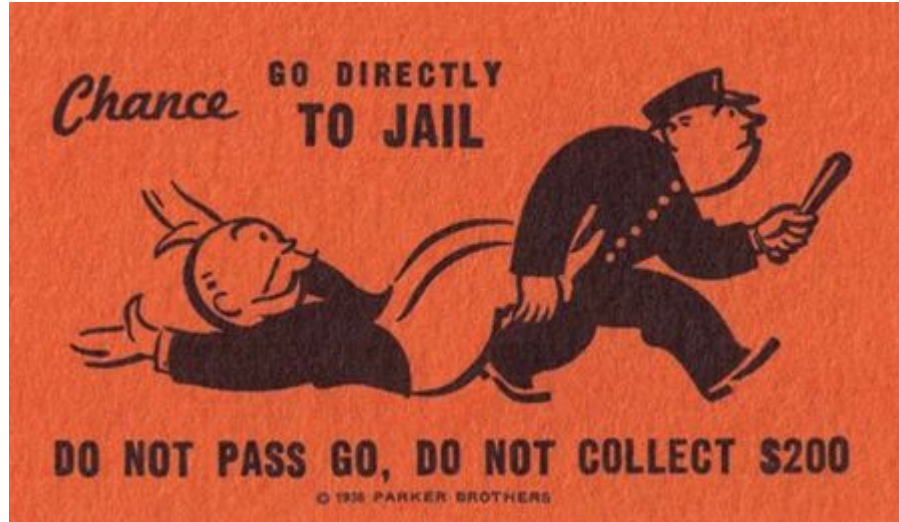
# Every operation has a fixed gas cost

	opcodes	gas cost
Basic operations	ADD, MUL, PUSH, JUMP	2-10
Storage read	SLOAD	200
Storage write	SSTORE	5000
Storage write (from zero)	SSTORE	20000
Storage zeroize	SSTORE	-10000
Contract call	CALL, CODECALL, etc.	700
Transaction overhead	n/a	21000
Contract creation	n/a	32000
Contract destruction	SELFDESTRUCT	-19000

# Gas metering is complex

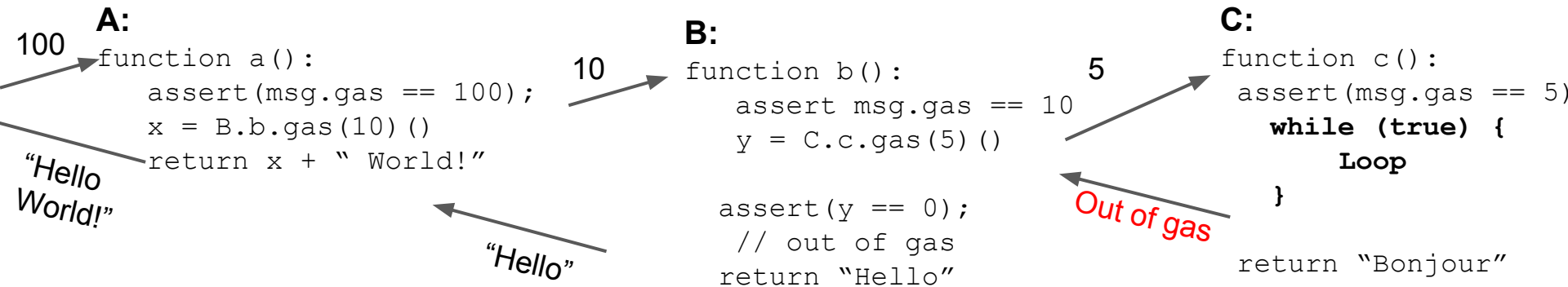
1. Transactions specify `START_GAS`, `GAS_PRICE`
2. If  $\text{START\_GAS} \times \text{GAS\_PRICE} > \text{caller.balance}$ , halt
3. Deduct  $\text{START\_GAS} \times \text{GAS\_PRICE}$  from `caller.balance`
4. Set `GAS = START_GAS`
5. Run code, deducting from `GAS`
6. For negative values, add to `GAS_REFUND`
  - a. `GAS` only decreases
7. After termination, add  $\text{GAS} + \text{GAS\_REFUND}$  to `caller.balance`

# Out-of-gas exceptions are bad news



- State reverts to previous value
  - Except that  $\text{START\_GAS} * \text{GAS\_PRICE}$  is still deducted

# Callers can choose how much gas to send



Gas today is typically  $5 \times 10^{-9}$  ether =  $9 \times 10^{-7}$  USD

	opcodes	gas cost	USD cost [Aug '17]
Basic operations	ADD, MUL, PUSH, JUMP	2-10	$\sim 10^{-5}$
Storage read	SLOAD	200	0.0003
Storage write	SSTORE	5000	0.008
Storage write (from zero)	SSTORE	20000	0.032
Storage zeroize	SSTORE	-10000	-0.016
Contract call	CALL, CODECALL, etc.	700	0.0011
Transaction overhead	n/a	21000	0.033
Contract creation	n/a	32000	0.051
Contract destruction	SELFDESTRUCT	-19000	-0.031

# Economics of gas are similar to transaction fees

- Miners choose transactions based on GAS\_PRICE
- In theory, they should not care which opcodes are used
  - In practice, some “overpriced” opcodes may be preferred
- Maximum gas limit per block
  - Miners can slowly raise it, each block votes



# Solidity

# 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)`

# 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

# 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){
```

# Built-in support 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

# Solidity gotchas (many more tomorrow)

- 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`

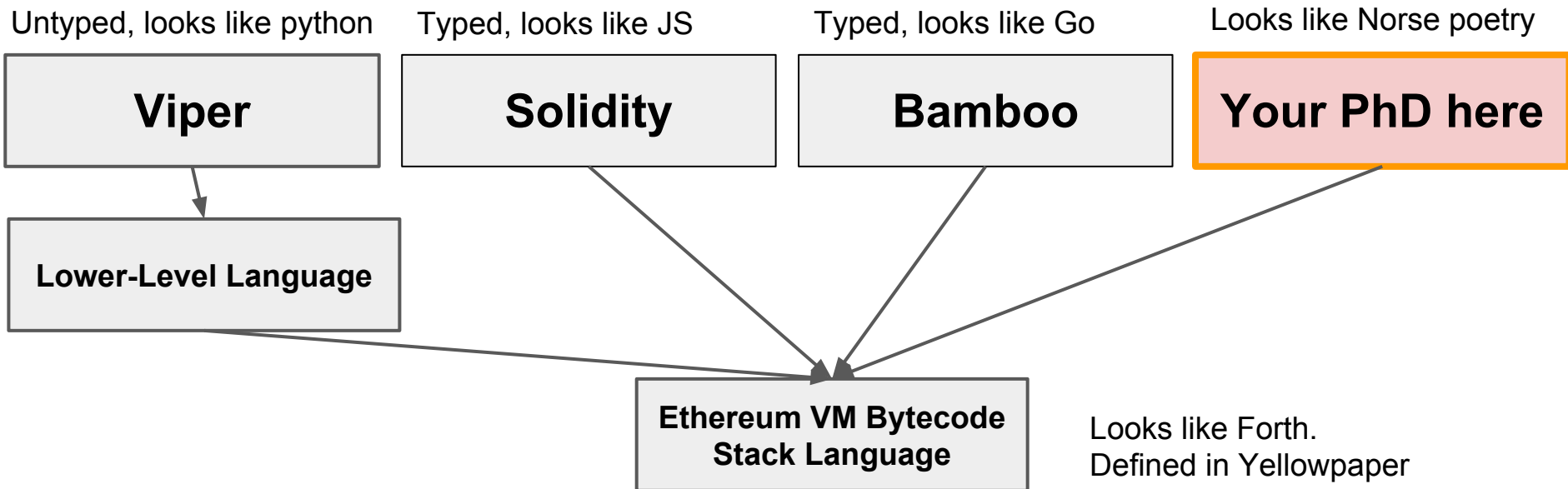


# Solidity and EVM may outgrow Ethereum itself



- 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!



# 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 <sup>2</sup>
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
Metropolis	2017?
Serenity	??

# Ethereum may soon overtake Bitcoin in market cap

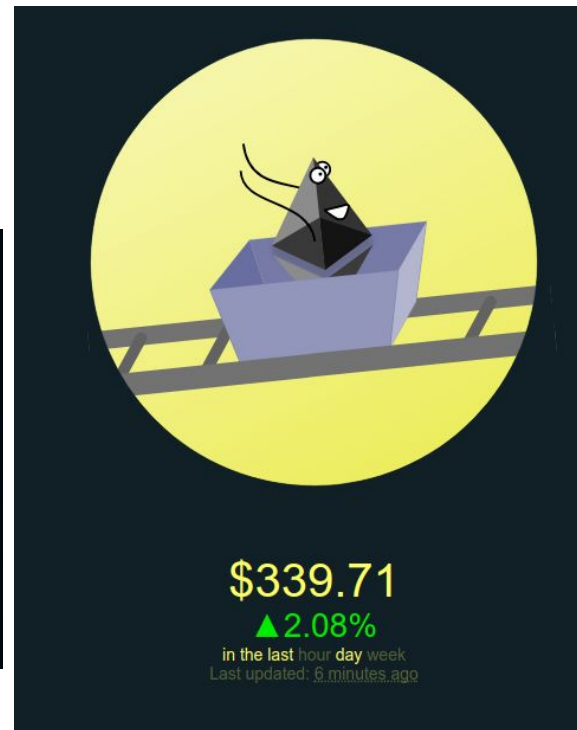


Source:  
Investopedia

# Price has been a wild ride recently



<https://ethereumprice.org/>




<http://eth.rollercoaster.one/>



# Research challenges

# Ethereum makes all data public

- Proposals:

- Project Alchemy-exchange Eth for Zcash  CASH
- 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  $\text{BLOCKHASH}(N - 1)$  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) {
```

...

Can you spot the bug?

```
    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
```

# 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>

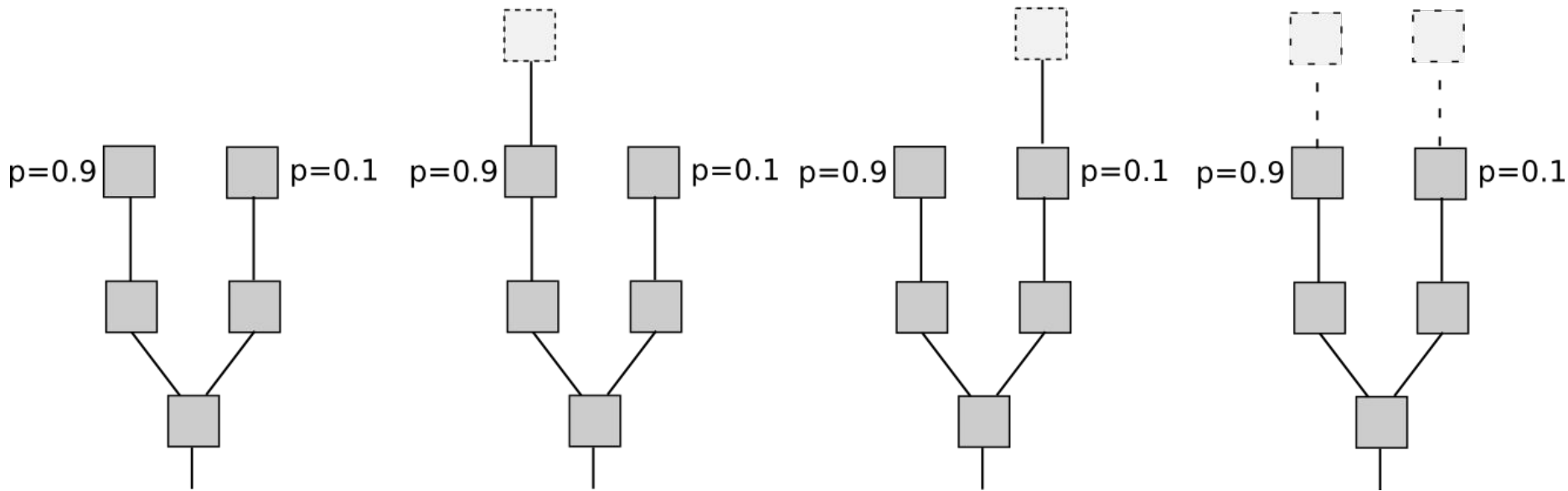
# Ethereum has long held plans to adopt proof-of-stake

Vote on neither  
 $EV = 0$

Vote on A  
 $EV = 0.9$

Vote on B  
 $EV = 0.1 - 0.9 * 5 = -4.4$

Vote on both  
 $EV = 0.1 + 0.9 - 5 = -4$



Explore more!

# Explore the blockchain: <https://etherscan.io>



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%)

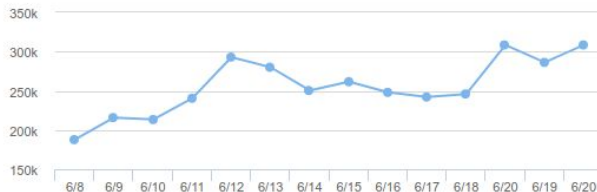
LAST BLOCK  
3907629 (17.41s Avg)

Hash Rate  
50,614.71 GH/s

TRANSACTIONS  
31452919

Network Difficulty  
868.00 TH

14 day Ethereum Transaction History



Blocks

View All

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

View All

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

TX# [0X3F89C2BD16D10FA32AD07EF...](#) > 31 secs ago  
From [0x308593430e6e35...](#) To [0x07db7e8722a04a...](#)  
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 DAPPS

Search



search by tags



503 dapps listed

Sort: Updated

<b>ClimateCoin</b> <b>Dennis Peterson</b> Coins for those who offset carbon  Concept 2017-06-20	<b>Token Creation Ser...</b> <b>Minereum Team</b> Create your own Ethereum Token with just a Minereum Transaction  Live 2017-06-20	<b>HitFin</b> <b>Patrick Salami</b> OTC Derivatives Settlement  Demo 2017-06-20	<b>Seglos</b> <b>Maytham Naei</b> Spend your Ethereum without losing out on the future grow  Working Prototype 2017-06-20	<b>Ether.Camp</b> <b>Roman Mandeleil</b> Blockchain explorer  Live 2017-06-20	<b>TimeBank</b> <b>Isaac Ibiapina</b> Store Ether enforceably with a time lock  Live 2017-06-20
<b>Vevue</b> <b>Thomas Olson</b> Bringing Google Street View to life  Concept 2017-06-20	<b>PowerBall</b> <b>Peter Borah</b> "Powerball"-style lottery  Work In Progress 2017-06-20	<b>cyber•Fund</b> <b>Dima Starodubcev</b> Make digital investments comprehensible, accessible, easy and safe  Concept 2017-06-20	<b>WeiLend</b> <b>Massi Terzi</b> P2P Lending  Work in Progress 2017-06-20	<b>Truffle</b> <b>Tim Coulter</b> Development framework for Ethereum  Working Prototype 2017-06-20	<b>TrustlessPrivacy</b> <b>sam@trustlessprivacy.com</b> Interoperable electronic health records  Work In Progress 2017-06-20

# More this week!

- Tuesday: Practical programming exercise!
- Wednesday: Security issues in smart contract programming

# Verifying consistency of Ethereum implementations

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