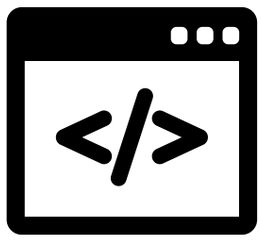
Bitcoin and Smart Contracts

Bitcoin and Smart Contracts



Bitcoin and Smart Contracts

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”

Smart Contract

- Arbitrary code executed by all participants
- Global consensus over execution
- Automated **verification** and **enforcement** of contracts
- Allow transfer of funds

Bitcoin and Smart Contracts

Alice will reveal to Bob a value x such that SHA-256(x) = 0x1b...

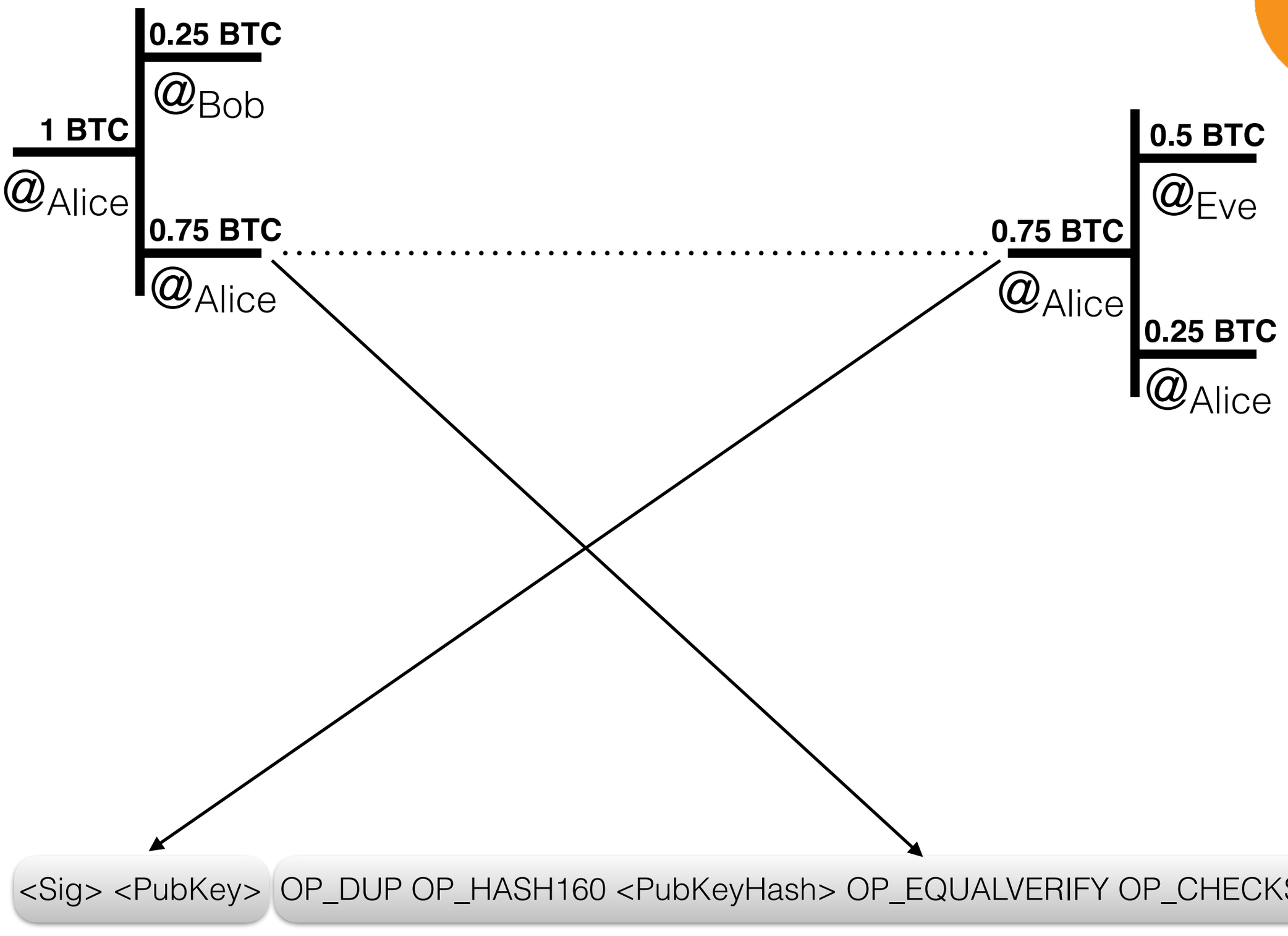
In exchange, Bob will pay USD 10.

If Alice does not reveal by July 1, 2032, then she will pay a penalty of USD 1 per day that she is late, up to USD 100.

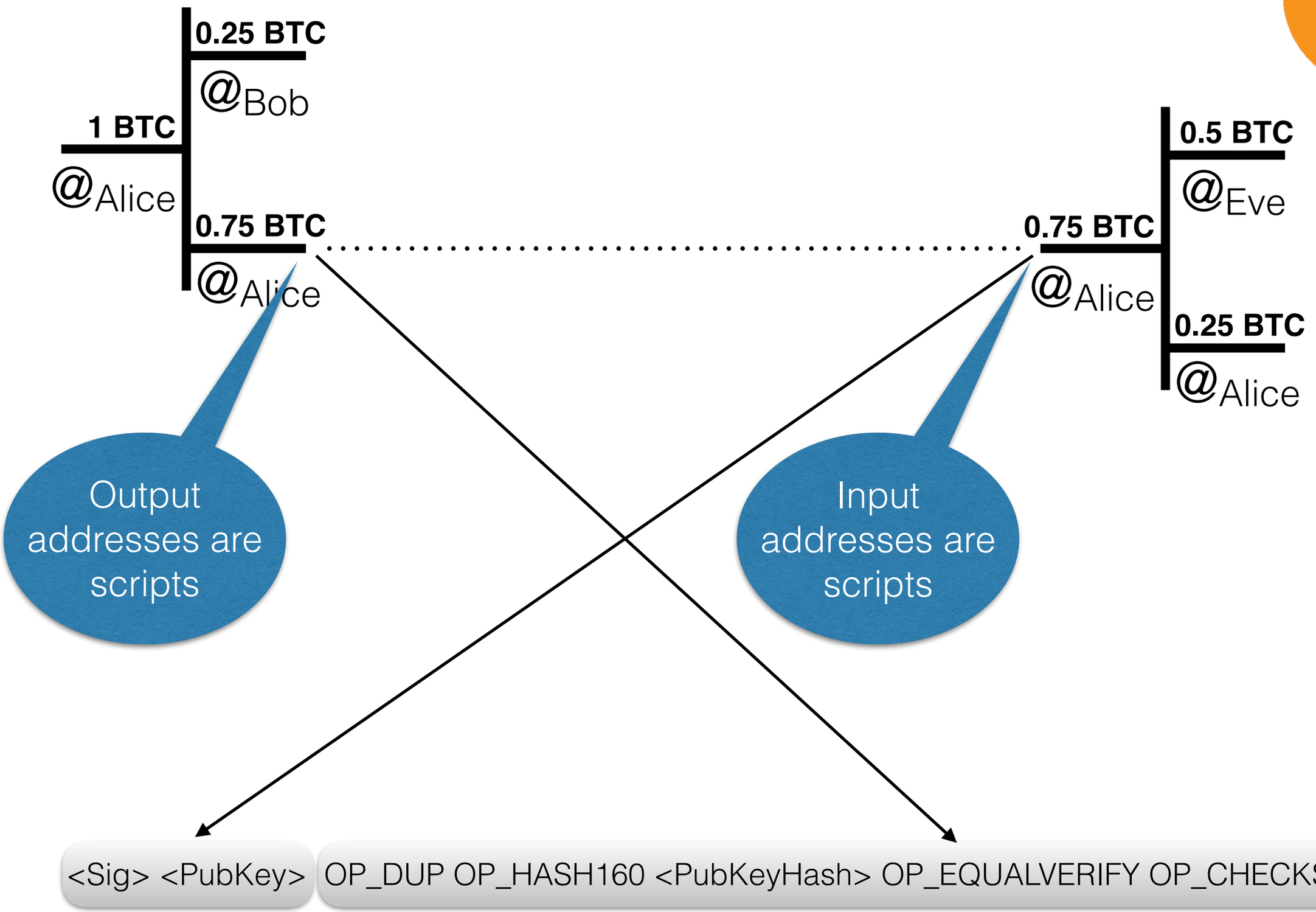
Smart Contracts vs Traditional Contracts

	Traditional	Smart
Specification	Natural language	Code
Identity & consensus	Written Signatures	Digital signatures
Dispute resolution	Judges	Decentralized platform
Nullification	Judges	?
Payment	Legally enforced	Built-in
Escrow	Trusted Third Party	Built-in

Bitcoin Script



Bitcoin Script



Bitcoin Script

- 256 opcodes total
  - 15 disabled (security!), 75 reserved
- Can handle
  - If/then
  - Arithmetic
  - Data handling
  - Crypto
    - Hashes
    - Signature verification
    - Multi-signature verification

Bitcoin Script

- Design goals
  - Simple, compact
  - Stack-based
  - No support for loops —> not Turing complete
  - Execution time/memory bound by program size

Bitcoin Script

- Design goals
  - Simple, compact
  - Stack-based
  - No support for loops —> not Turing complete
  - Execution time/memory bound by program size

 image via Jessie St. Amand

Bitcoin Script

- Design goals
  - Simple, compact
  - Stack-based
  - No support for loops —> not Turing complete
  - Execution time/memory bound by program size

Not impressed

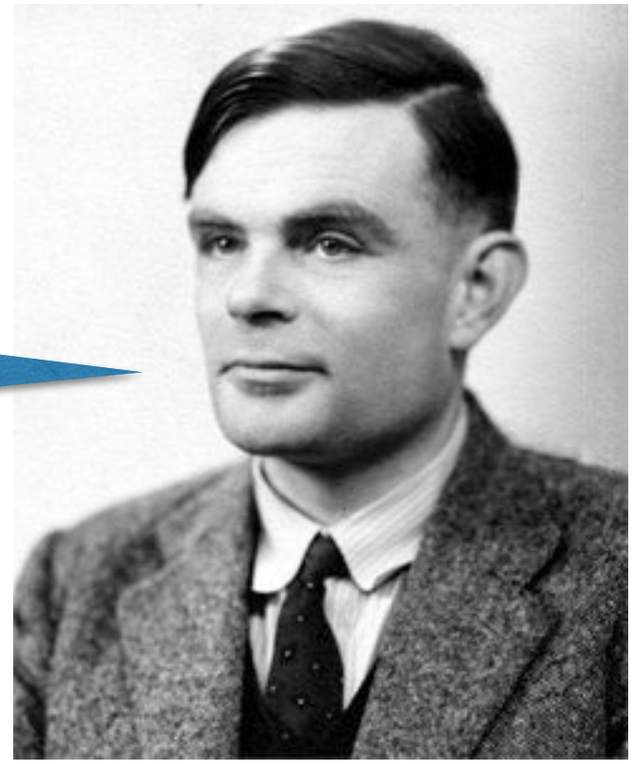


 image via Jessie St. Amand

Bitcoin Script Applications

- Proof of Burn
- Multisignature addresses
- Pay-for-hash preimage
  - Multiparty lotteries
  - Atomic cross-chain swap
- Micropayment channels
  - Use of OP_CHECKLOCKTIME

Extending Bitcoin Script

- Distributed Name Reservation (Namecoin)
- Prediction Markets (Augur, Futurecoin)
- Decentralized Markets (OpenBazaar)
- Financial Instruments (MasterCoin)

Extending Bitcoin Script

- Distributed Name Reservation (Namecoin)
- Prediction Markets (Augur, Futurecoin)
- Decentralized Markets (OpenBazaar)
- Financial Instruments (MasterCoin)

Why not a more flexible and open language?

Namecoin

Namecoin

- Similar to DNS
- Allows anyone to register a name
- Decentralized, no trusted third party
- First fork of Bitcoin!

Namecoin

- Extension to Bitcoin script with 3 new opcodes
  - NAME_NEW
  - NAME_FIRST_UPDATE
  - NAME_UPDATE

Namecoin

- Extension to Bitcoin script with 3 new opcodes
  - NAME_NEW
  - NAME_FIRST_UPDATE
  - NAME_UPDATE

NAME_NEW: Hash(random_nonce, “agervais”)

Time

Namecoin

- Extension to Bitcoin script with 3 new opcodes
  - NAME_NEW
  - NAME_FIRST_UPDATE
  - NAME_UPDATE

NAME_NEW: Hash(random_nonce, “agervais”)

NAME_FIRST_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.23”}

Time

Namecoin

- Extension to Bitcoin script with 3 new opcodes
  - NAME_NEW
  - NAME_FIRST_UPDATE
  - NAME_UPDATE

NAME_NEW: Hash(random_nonce, “agervais”)

NAME_FIRST_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.23”}

NAME_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.24”}

Time

Namecoin

- Extension to Bitcoin script with 3 new opcodes
  - NAME_NEW
  - NAME_FIRST_UPDATE
  - NAME_UPDATE

NAME_NEW: Hash(random_nonce, “agervais”)

Is there something to consider?

NAME_FIRST_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.23”}

NAME_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.24”}

Time

Namecoin

- Extension to Bitcoin script with 3 new opcodes
  - NAME_NEW
  - NAME_FIRST_UPDATE
  - NAME_UPDATE

NAME_NEW: Hash(random_nonce, “agervais”)

Is there something to consider?

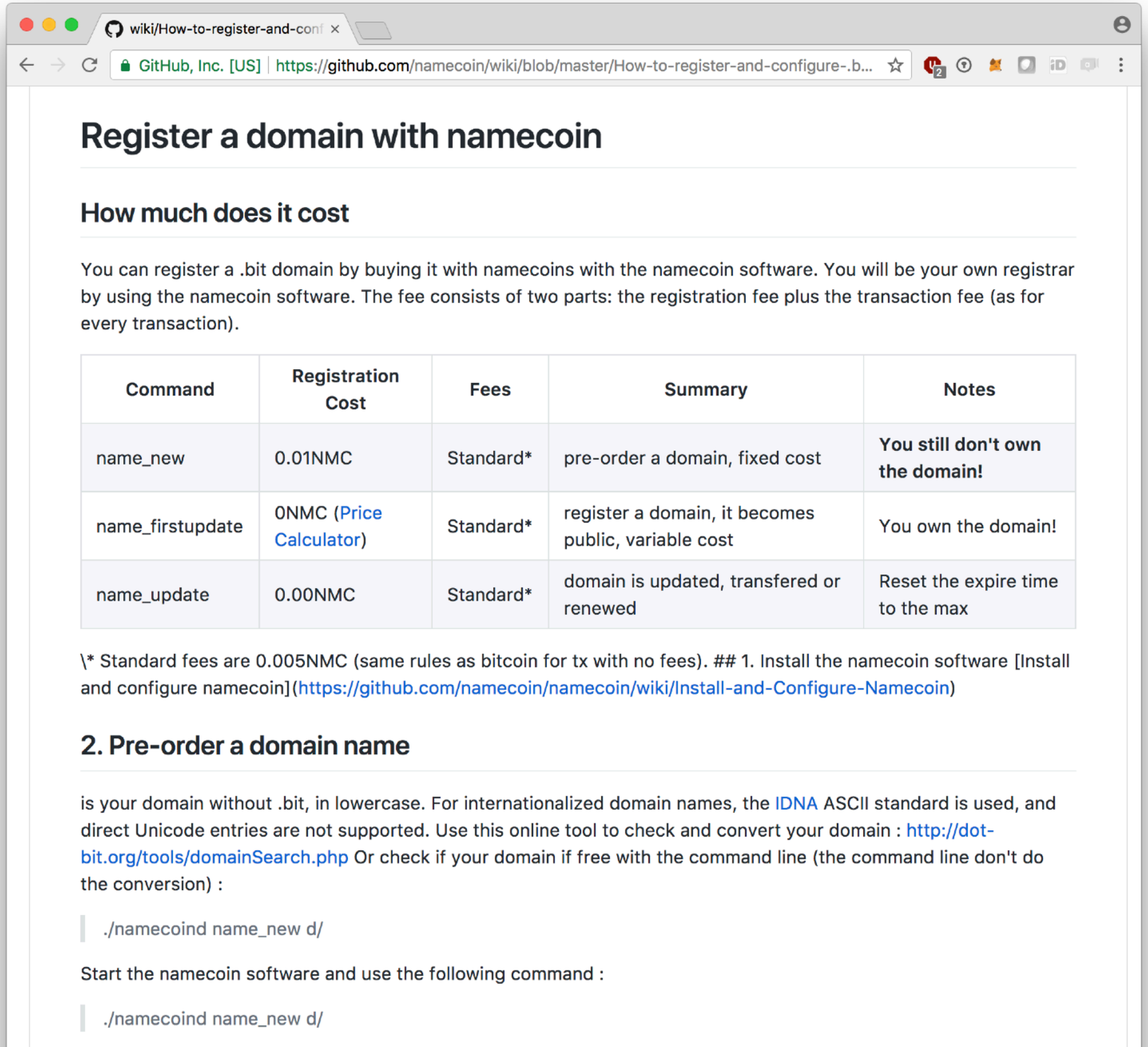
Wait for 12 blocks, preventing frontrunning!

NAME_FIRST_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.23”}

NAME_UPDATE: random_nonce, “agervais”, {“ip”:”234.22.34.24”}

Time

Namecoin Fees



Register a domain with namecoin

How much does it cost

You can register a .bit domain by buying it with namecoins with the namecoin software. You will be your own registrar by using the namecoin software. The fee consists of two parts: the registration fee plus the transaction fee (as for every transaction).

Command	Registration Cost	Fees	Summary	Notes
name_new	0.01NMC	Standard*	pre-order a domain, fixed cost	**You still don't own the domain!**
name_firstupdate	0NMC (Price Calculator)	Standard*	register a domain, it becomes public, variable cost	You own the domain!
name_update	0.00NMC	Standard*	domain is updated, transfered or renewed	Reset the expire time to the max

* Standard fees are 0.005NMC (same rules as bitcoin for tx with no fees). ## 1. Install the namecoin software [Install and configure namecoin](https://github.com/namecoin/namecoin/wiki/Install-and-Configure-Namecoin)

2. Pre-order a domain name

is your domain without .bit, in lowercase. For internationalized domain names, the IDNA ASCII standard is used, and direct Unicode entries are not supported. Use this online tool to check and convert your domain : http://dot-bit.org/tools/domainSearch.php Or check if your domain if free with the command line (the command line don't do the conversion) :

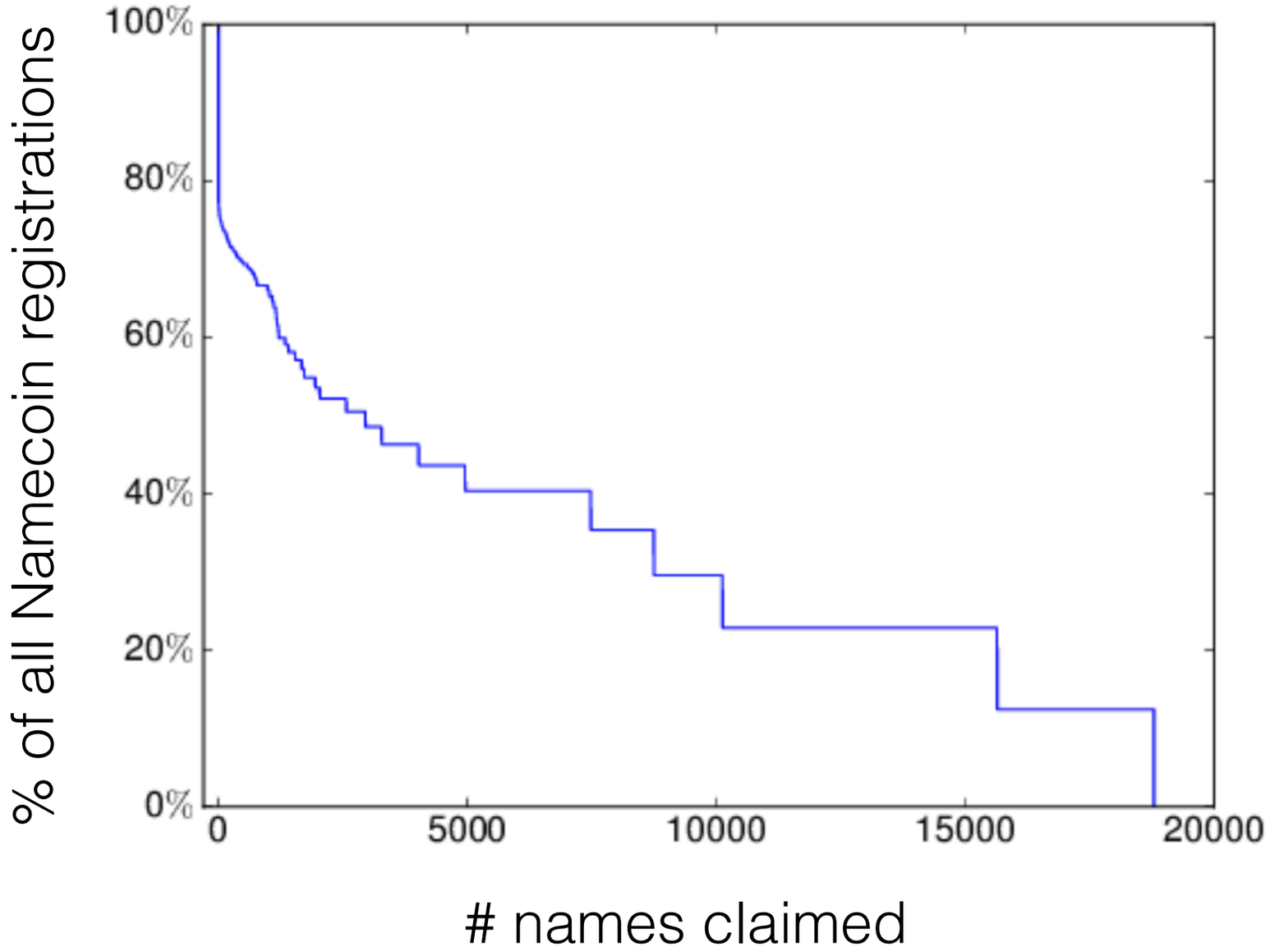
> ./namecoind name_new d/

Start the namecoin software and use the following command :

> ./namecoind name_new d/

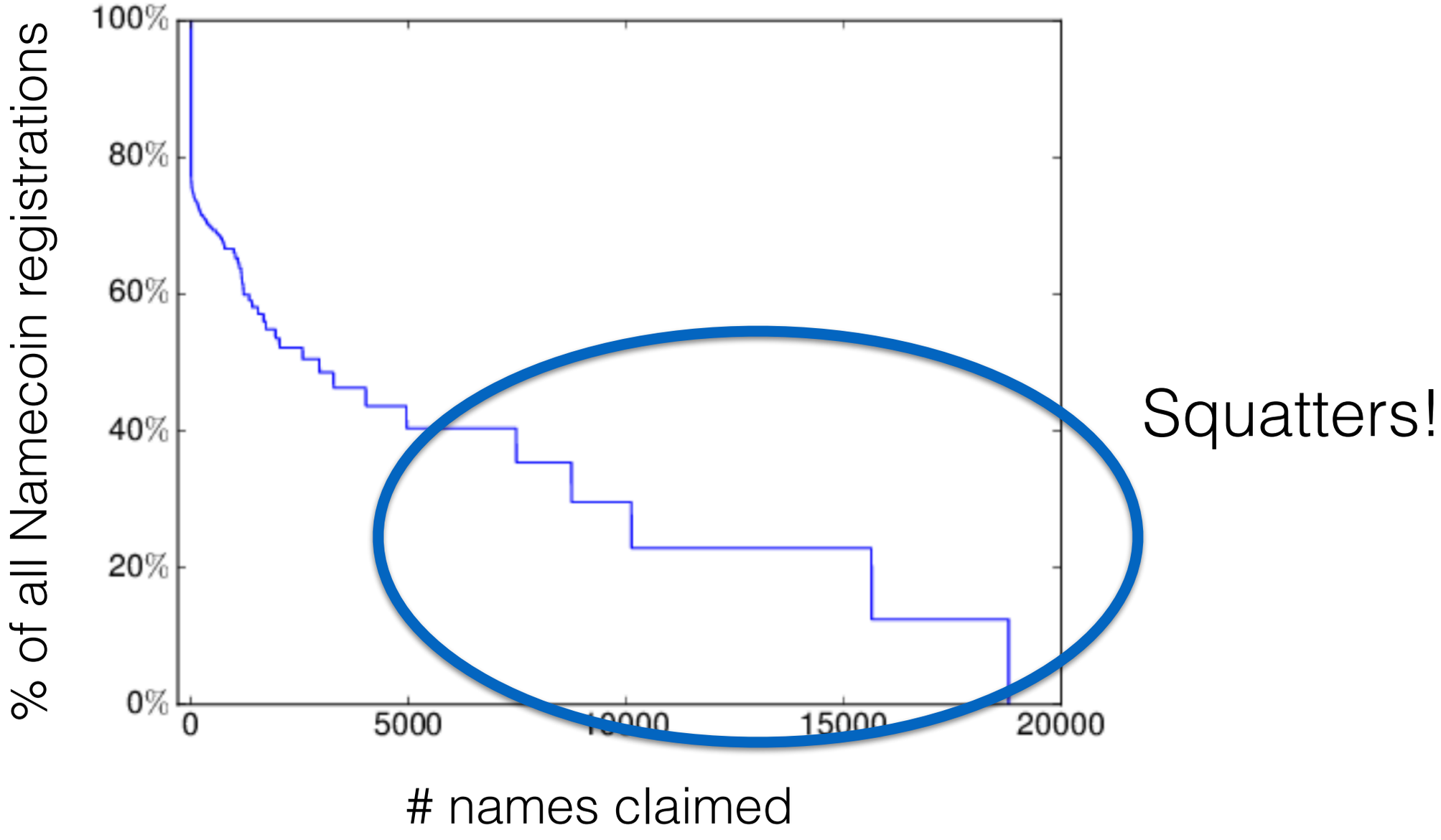
The output will looks like this :

Namecoin's Bad Incentive Structure



An empirical study of Namecoin and lessons for decentralized namespace design, Kalodner et al., WEIS 2015

Namecoin's Bad Incentive Structure



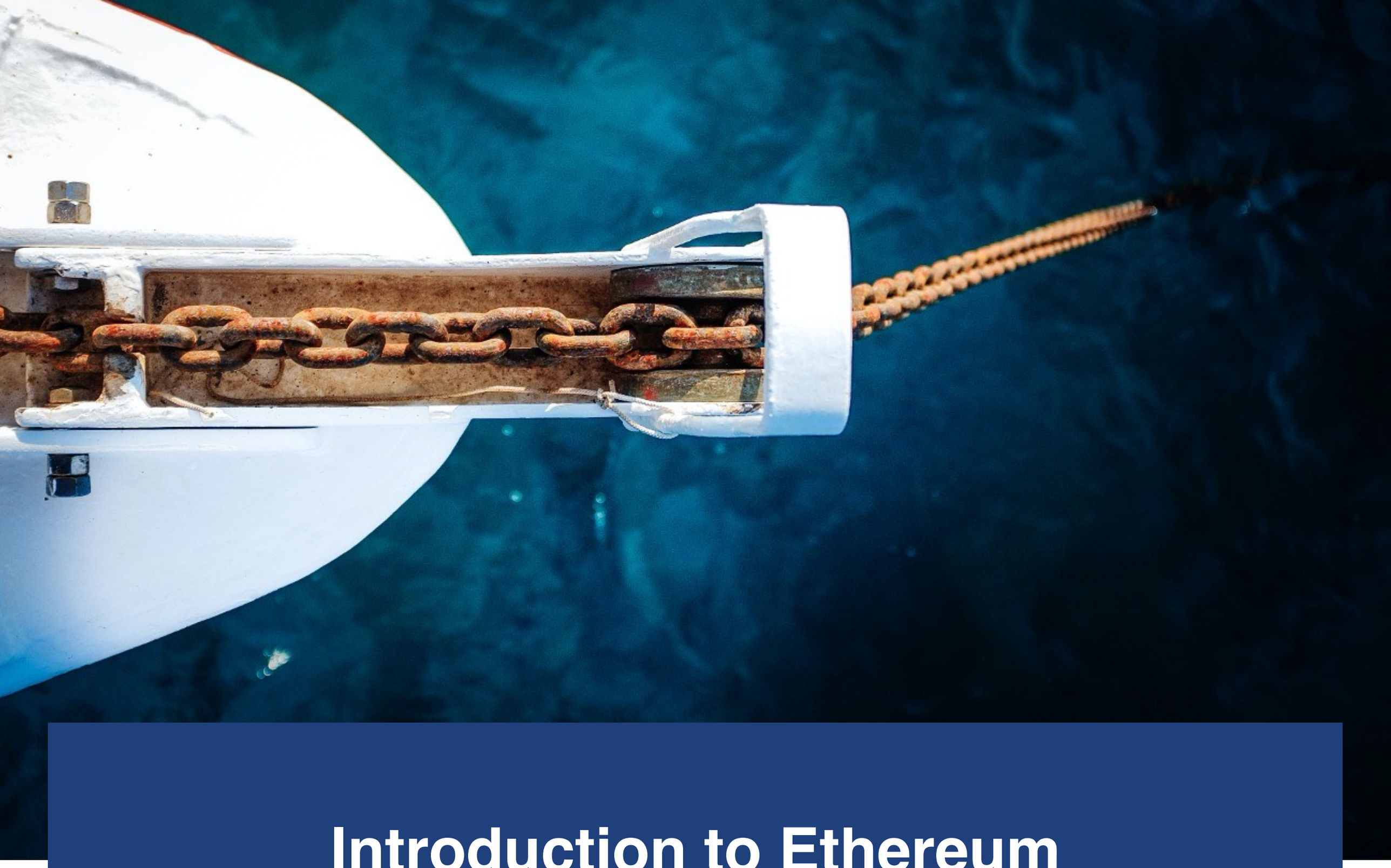
An empirical study of Namecoin and lessons for decentralized namespace design, Kalodner et al., WEIS 2015

New Functionality for Bitcoin

new addition	Purpose	In Namecoin	In OpenBazaar/ Futurecoin
Global State	Track application data	Register name	merchants, ask and bid
Opcodes	Write to global state	NAME_NEW, etc.	OPEN_MARKET, etc.
Fees	Limit spam, computation and storage read/write	Registration fee to limit squatting, expiry	Transaction fees for trades

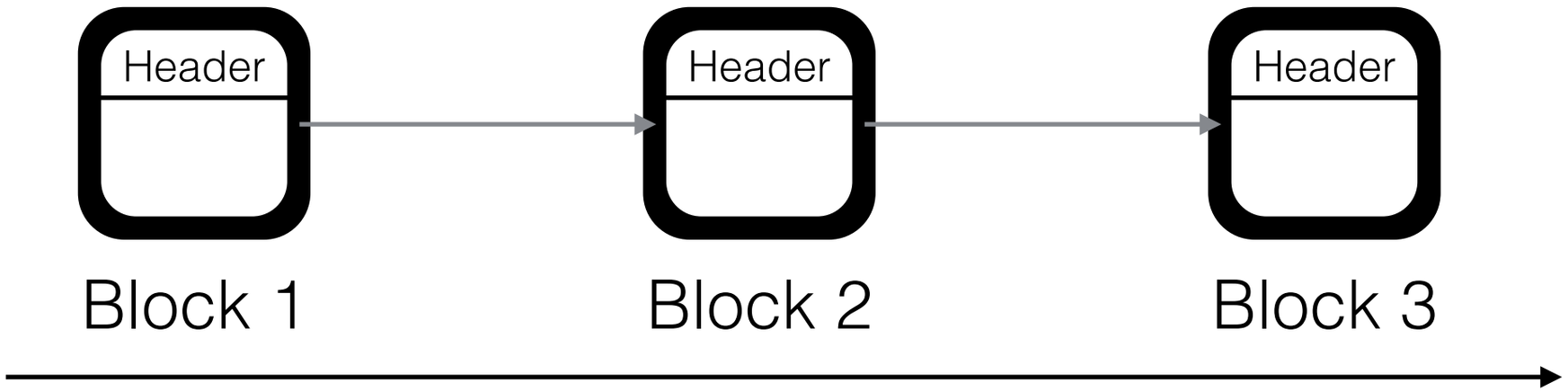
New Functionality for Bitcoin

new addition	Purpose	In Bitcoin
Global State	Track application data	**UTXO set**
Opcodes	Write to global state	**Transactions**
Fees	Limit spam, computation and storage read/write	**Implicit with transaction fees**



Introduction to Ethereum

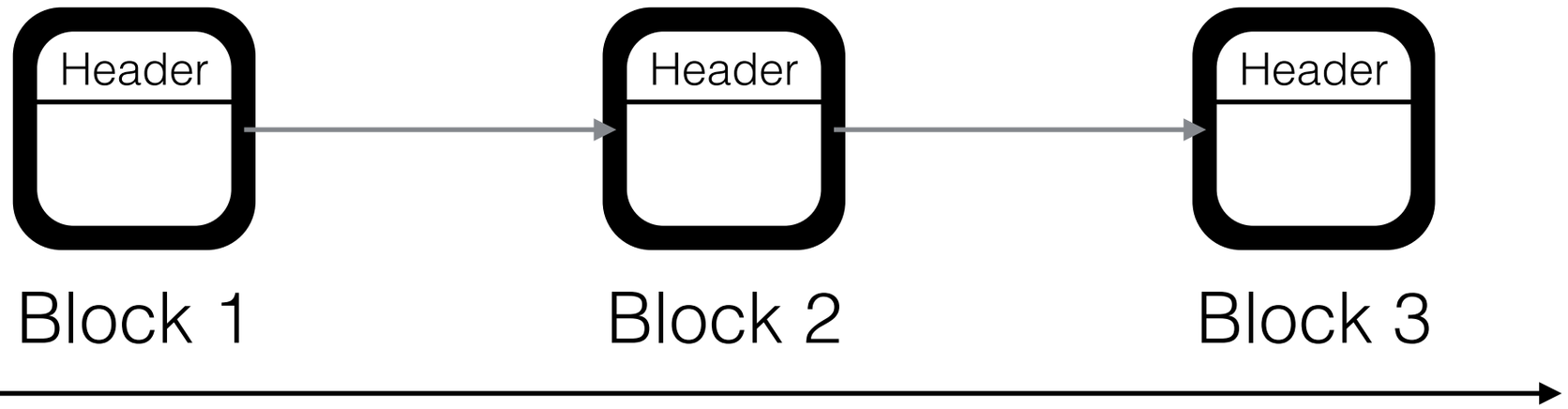
State changes in Blockchain



Consensus (nonce): ∅

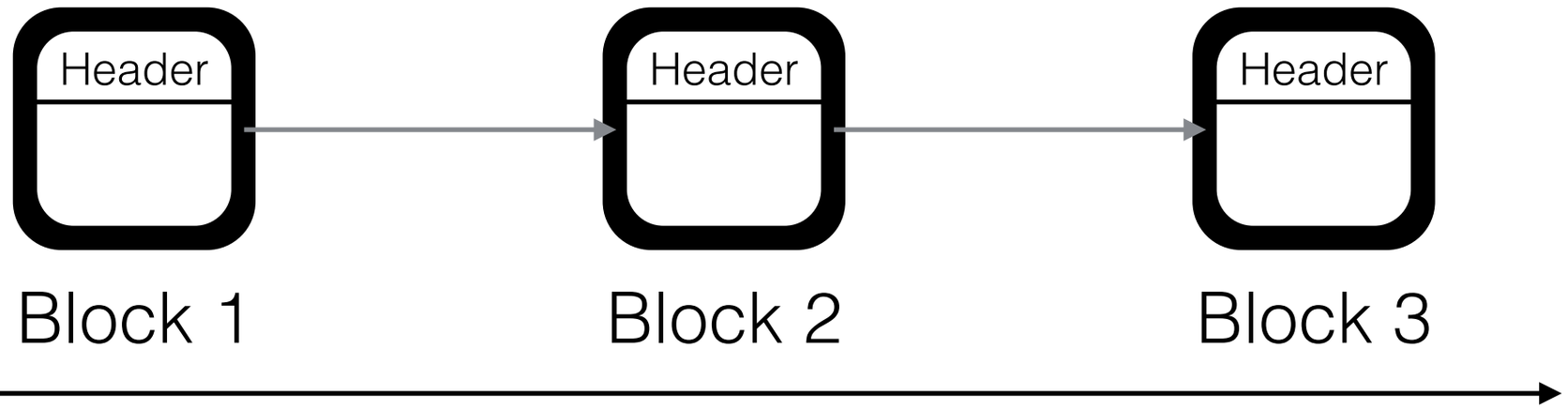
State change: ∅

State changes in Blockchain



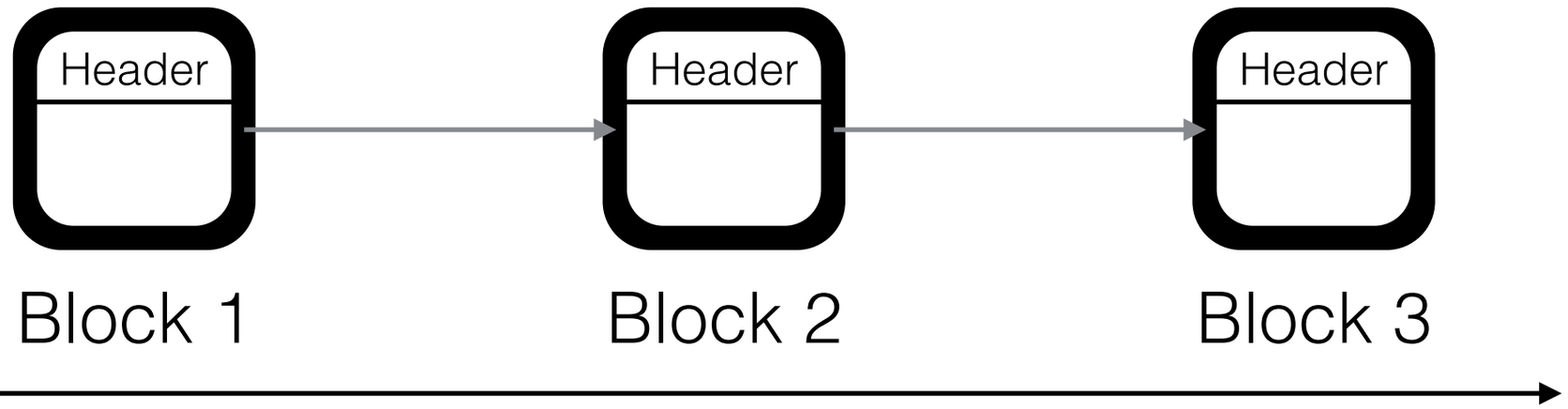
Consensus (nonce):	∅	0xab
State change:	∅	Transaction 1 A —> B, 3

State changes in Blockchain



Consensus (nonce):	∅	0xab	0xbv
State change:	∅	Transaction 1 A —> B, 3	Transaction 2 B —> C, 2

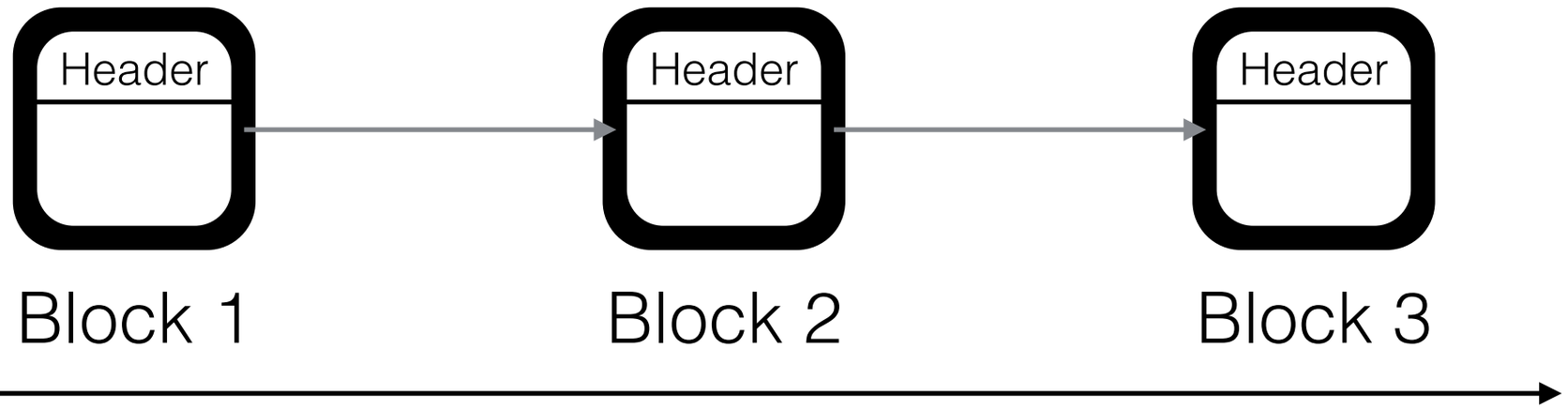
State changes in Blockchain



Consensus (nonce):	∅	0xab	0xbv
State change:	∅	Transaction 1 A —> B, 3	Transaction 2 B —> C, 2

Transaction 3
C —> D, 1

State changes in Blockchain

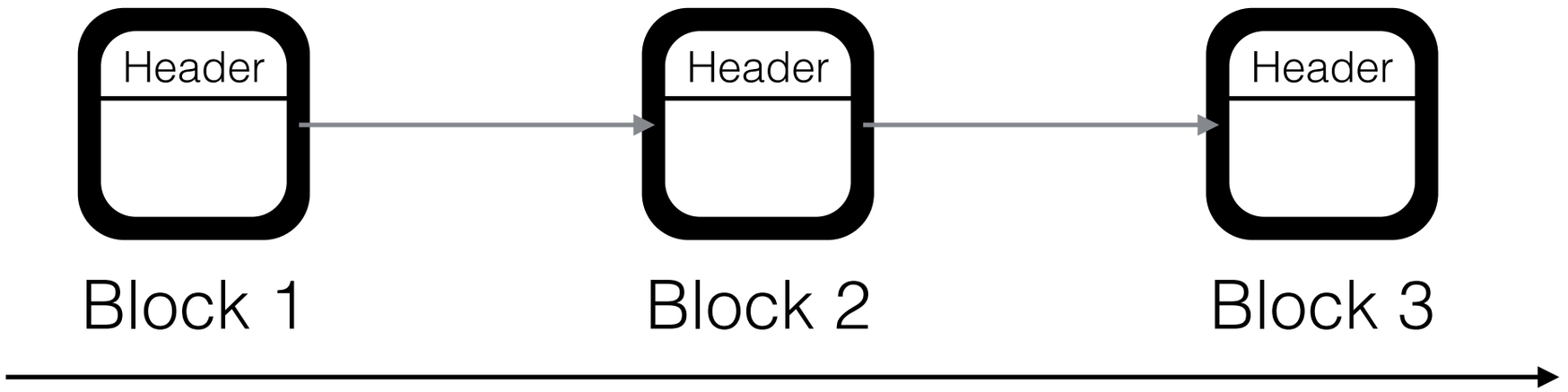


Consensus (nonce):	∅	0xab	0xbv
State change:	∅	Transaction 1 A —> B, 3	Transaction 2 B —> C, 2

Transaction 3
C —> D, 1

Is this transaction 3 valid?

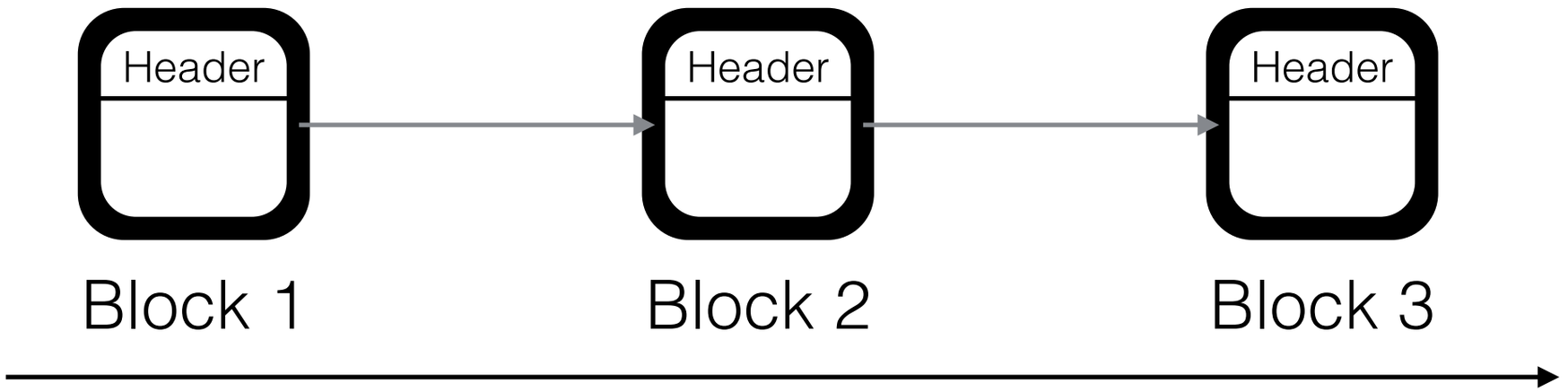
Store explicit state



Consensus (nonce):

State change:

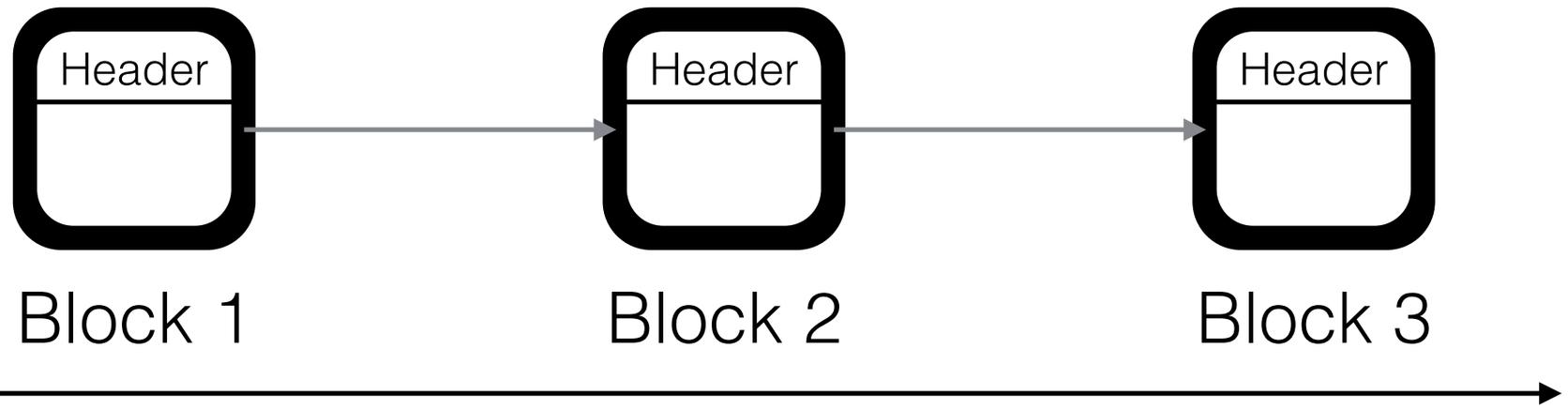
Store explicit state



Consensus (nonce): 0xab

State change: Transaction 1
A —> B, 3

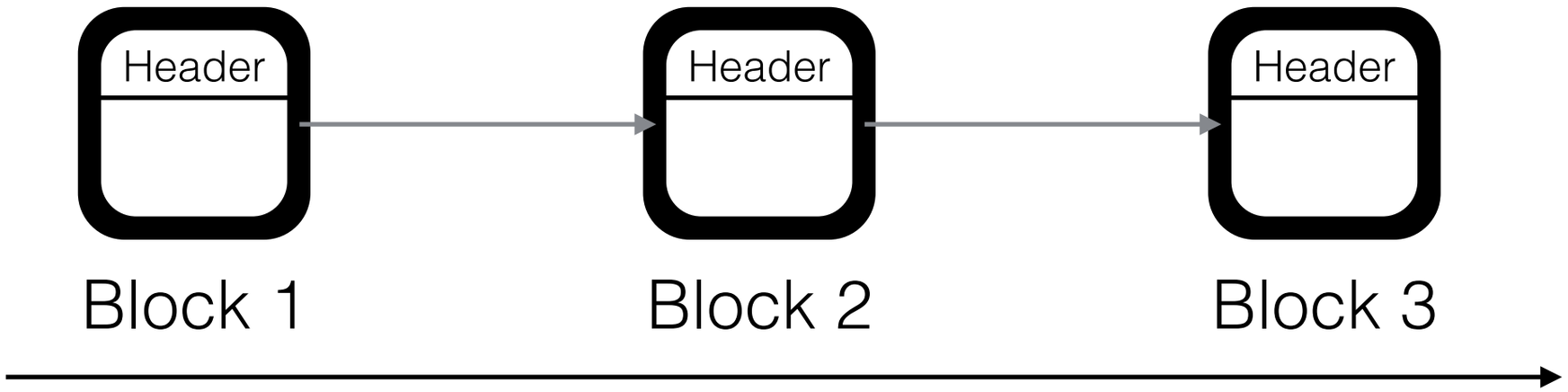
Store explicit state



Consensus (nonce): 0xab 0xbv

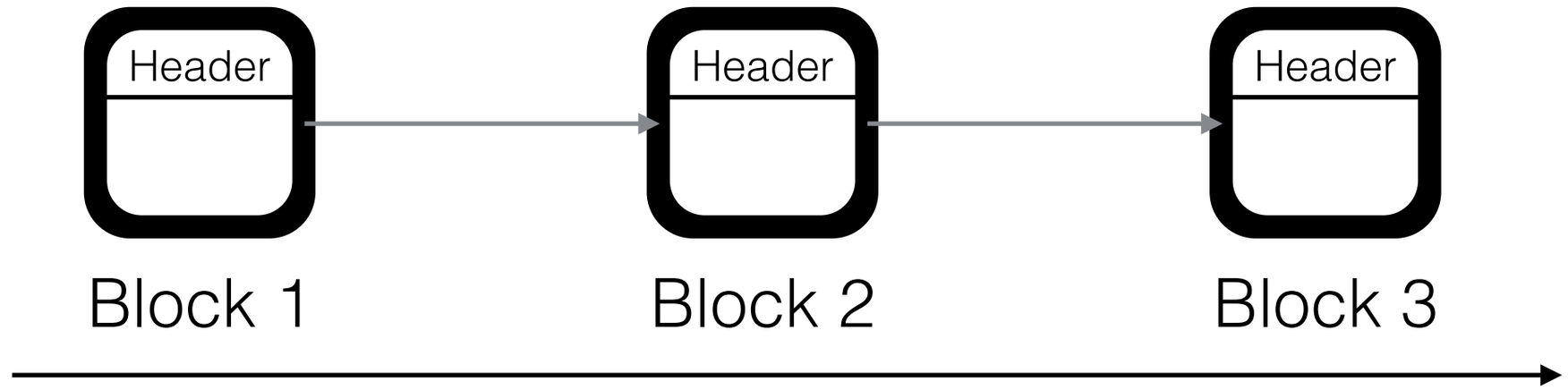
State change: Transaction 1 A —> B, 3 Transaction 2 B —> C, 2

Store explicit state



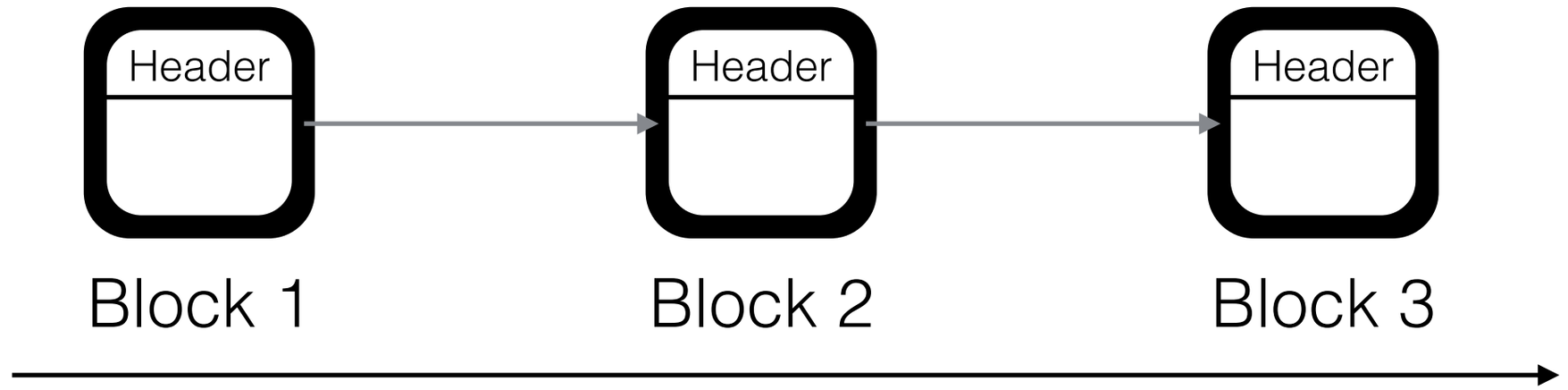
Consensus (nonce):		0xab	0xbv
State change:		Transaction 1 A —> B, 3	Transaction 2 B —> C, 2
State commitment:	{A:50}		

Store explicit state



Consensus (nonce):		0xab	0xbv
State change:		Transaction 1 A —> B, 3	Transaction 2 B —> C, 2
State commitment:	{A:50}	{A:47, B:3}	

Store explicit state



Consensus (nonce):		0xab	0xbv
State change:		Transaction 1 A —> B, 3	Transaction 2 B —> C, 2
State commitment:	{A:50}	{A:47, B:3}	{A:47, B:1, C:2}

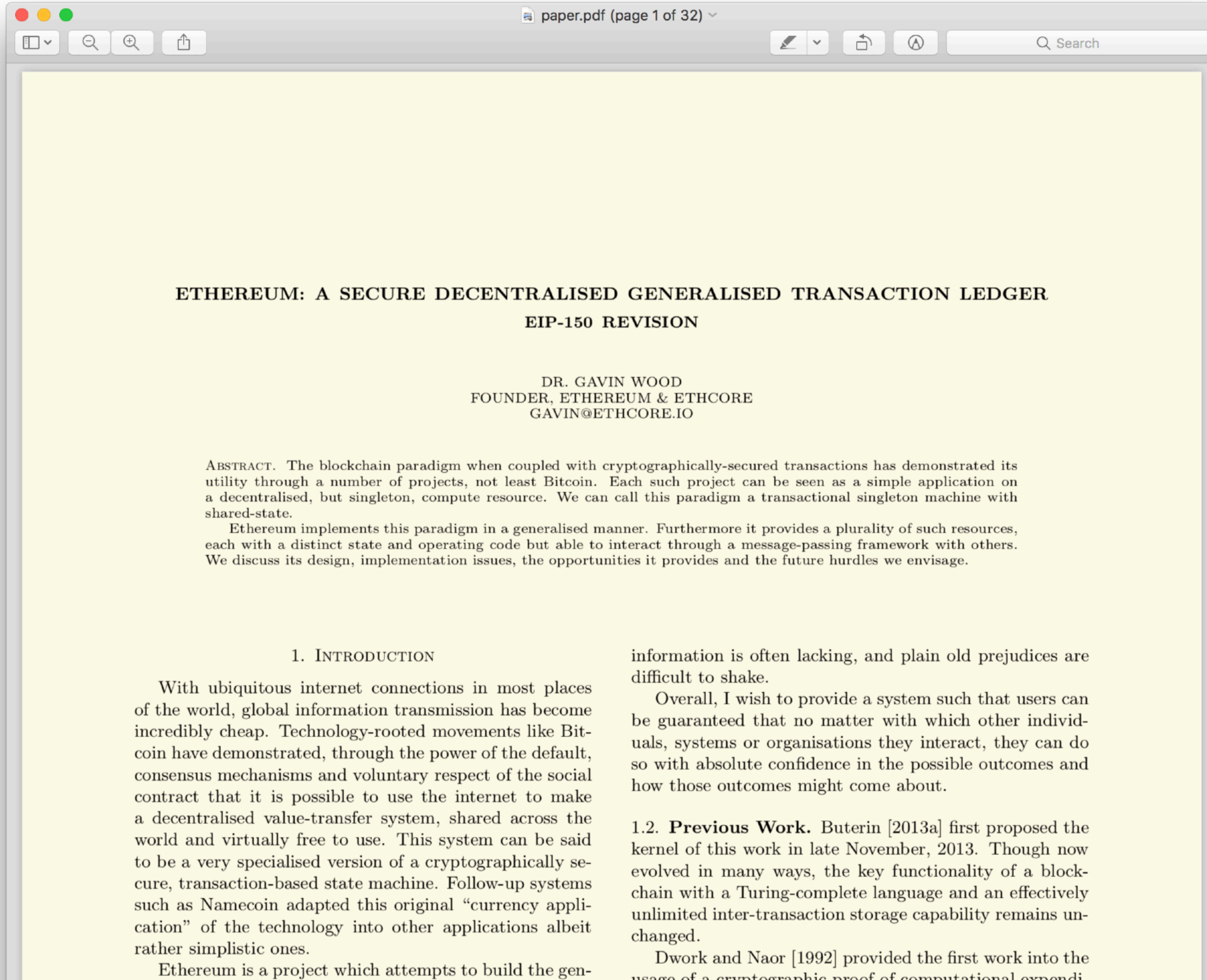
Advantages of explicit state storage

State commitment: {A:50} {A:47, B:3} {A:47, B:1, C:2}

- No need to go through whole history
- Sequence between any two blocks can be verified
- Light clients can sync up quickly

Ethereum - A universal Replicated State Machine

Ethereum - A universal Replicated State Machine



ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
EIP-150 REVISION

DR. GAVIN WOOD
FOUNDER, ETHEREUM & ETHCORE
GAVIN@ETHCORE.IO

Abstract. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. Introduction

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated, through the power of the default, consensus mechanisms and voluntary respect of the social contract that it is possible to use the internet to make a decentralised value-transfer system, shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original "currency application" of the technology into other applications albeit rather simplistic ones.

Ethereum is a project which attempts to build the gen-

information is often lacking, and plain old prejudices are difficult to shake.

Overall, I wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. **Previous Work.** Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now evolved in many ways, the key functionality of a blockchain with a Turing-complete language and an effectively unlimited inter-transaction storage capability remains unchanged.

Dwork and Naor [1992] provided the first work into the usage of a cryptographic proof of computational expendi-

Ethereum

- A (slow and expensive) world computer
- Consensus among all nodes about the execution
- More transaction type flexibility than Bitcoin
- Quasi-Turing complete language

Replicated State Machine

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

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

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

Replicated State Machine

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

Arbitrary programs

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

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

Replicated State Machine

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

Arbitrary programs

Execute programs

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

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

Ethereum

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

address	code	storage	balance	nonce

- **Inputs I** (transactions)

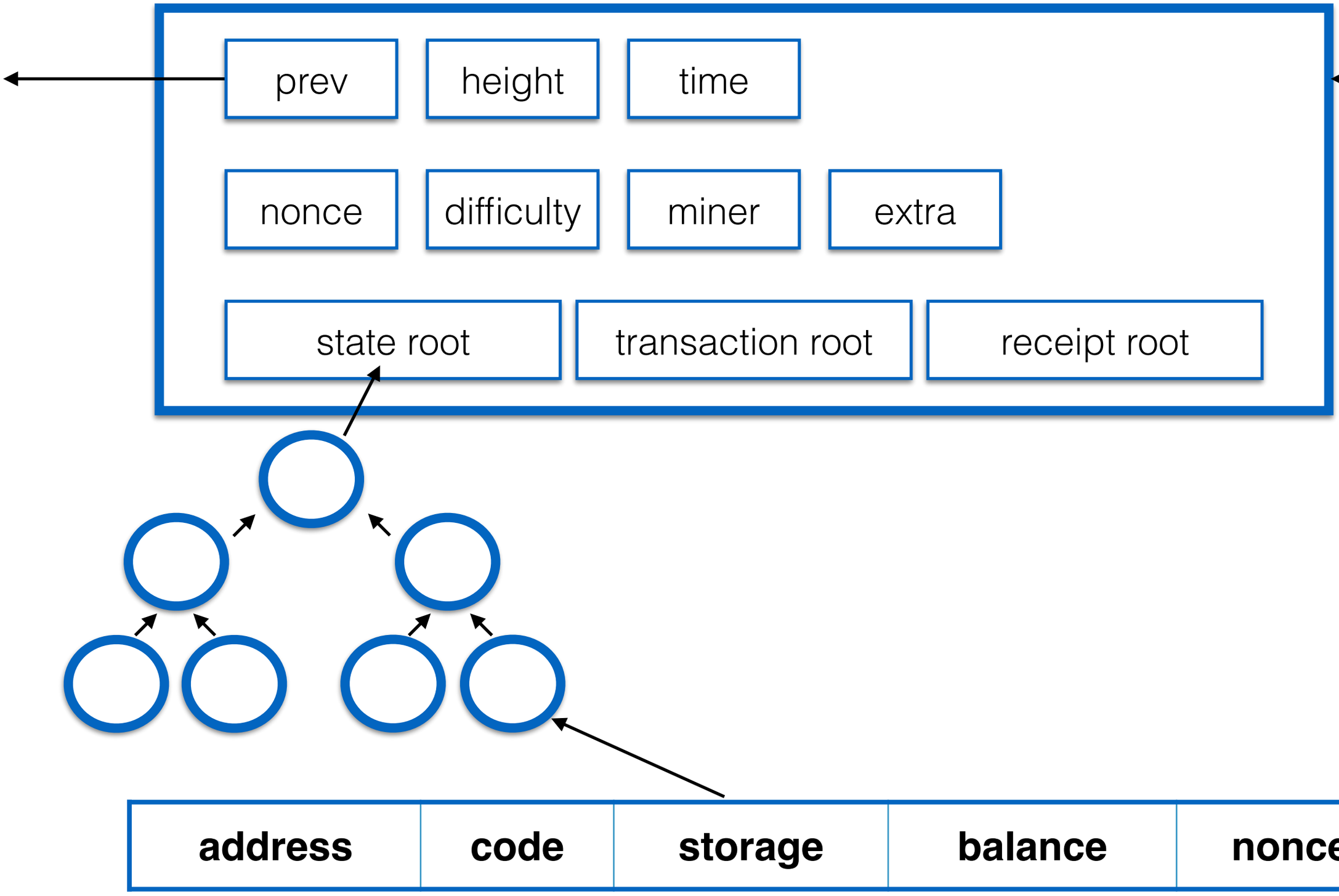
from	sig	nonce	to	data	value	gaslimit	gasprice

- **Transition f**:
  - Validate signature, nonce
  - Execute code (from, data, value, gaslimit, gasprice)

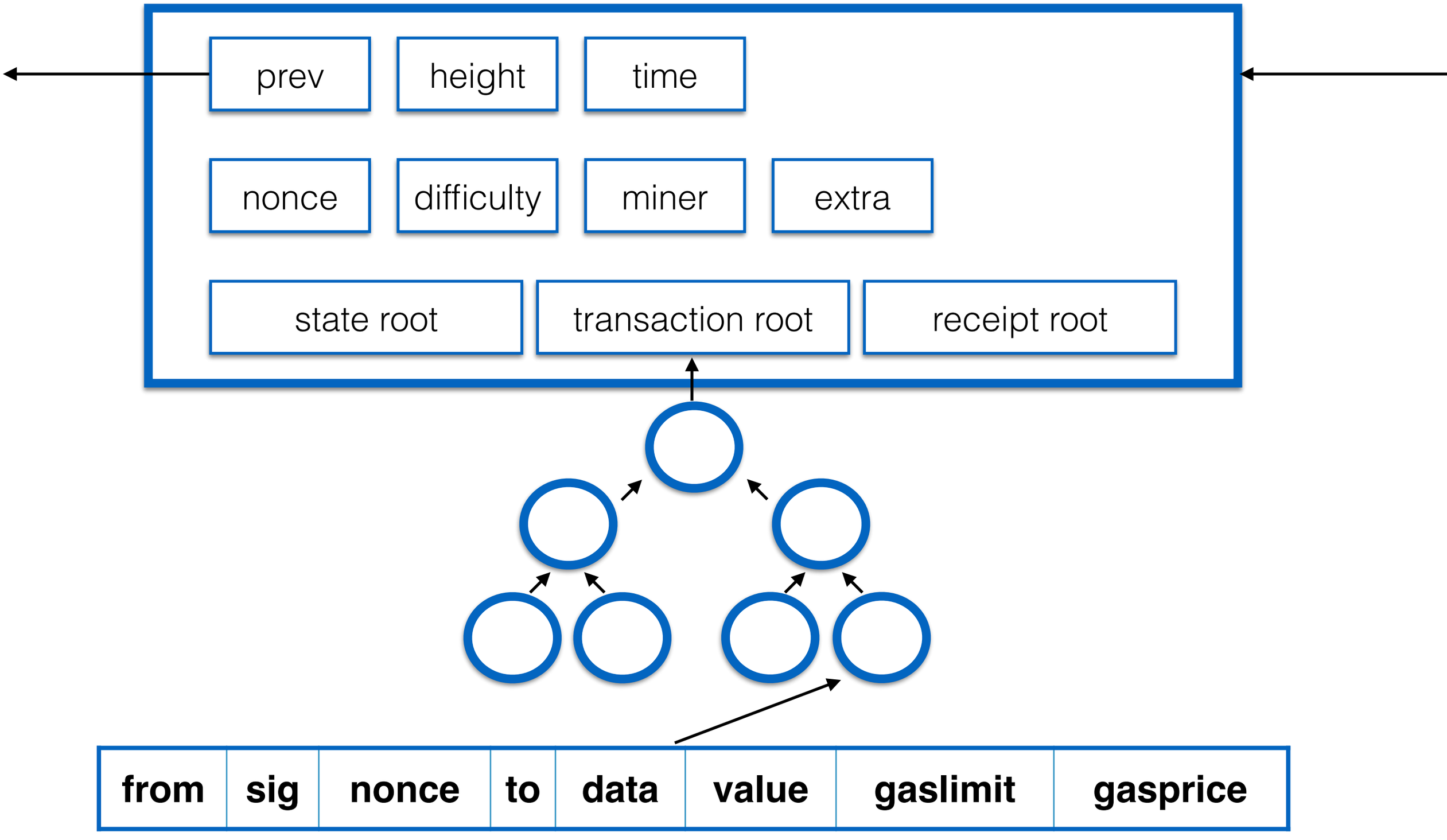
- Start state: ∅

Ethereum Blockchain Structure

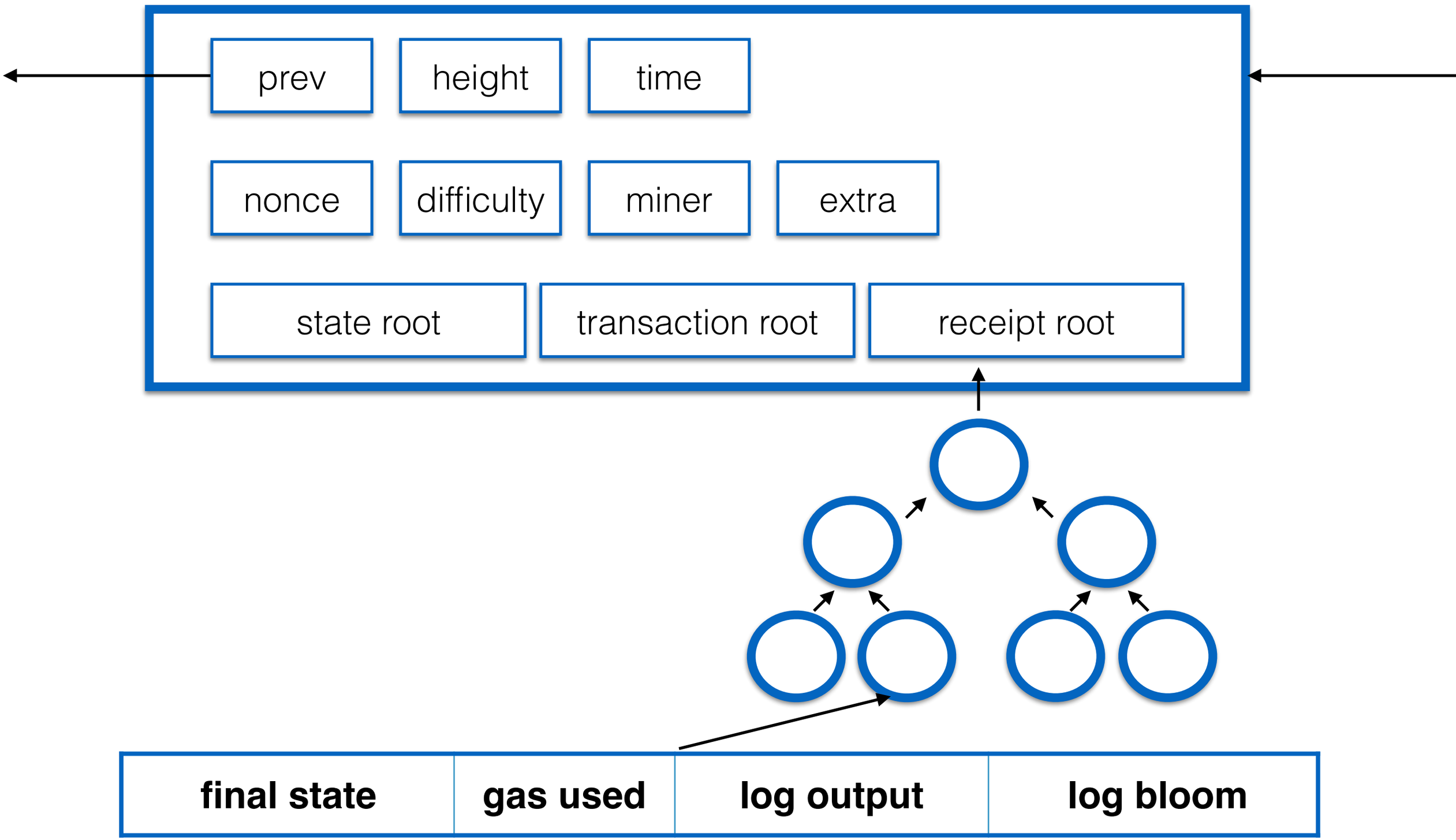
Ethereum Blockchain Structure



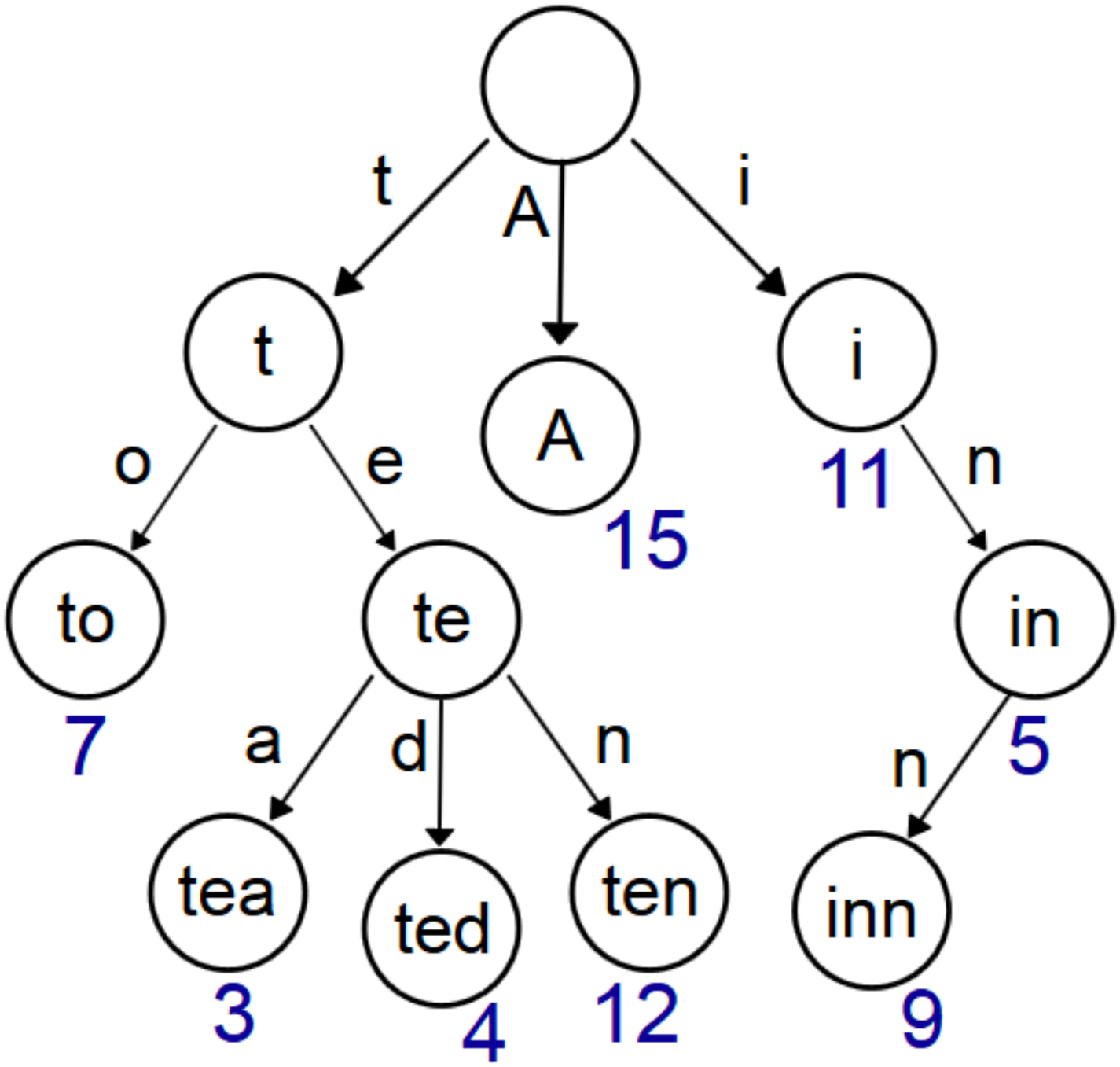
Ethereum Blockchain Structure

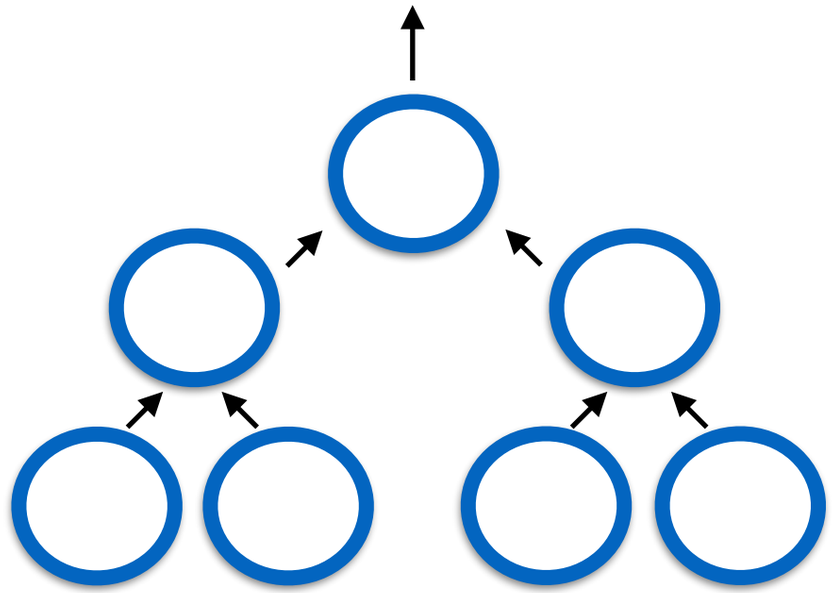


Ethereum Blockchain Structure

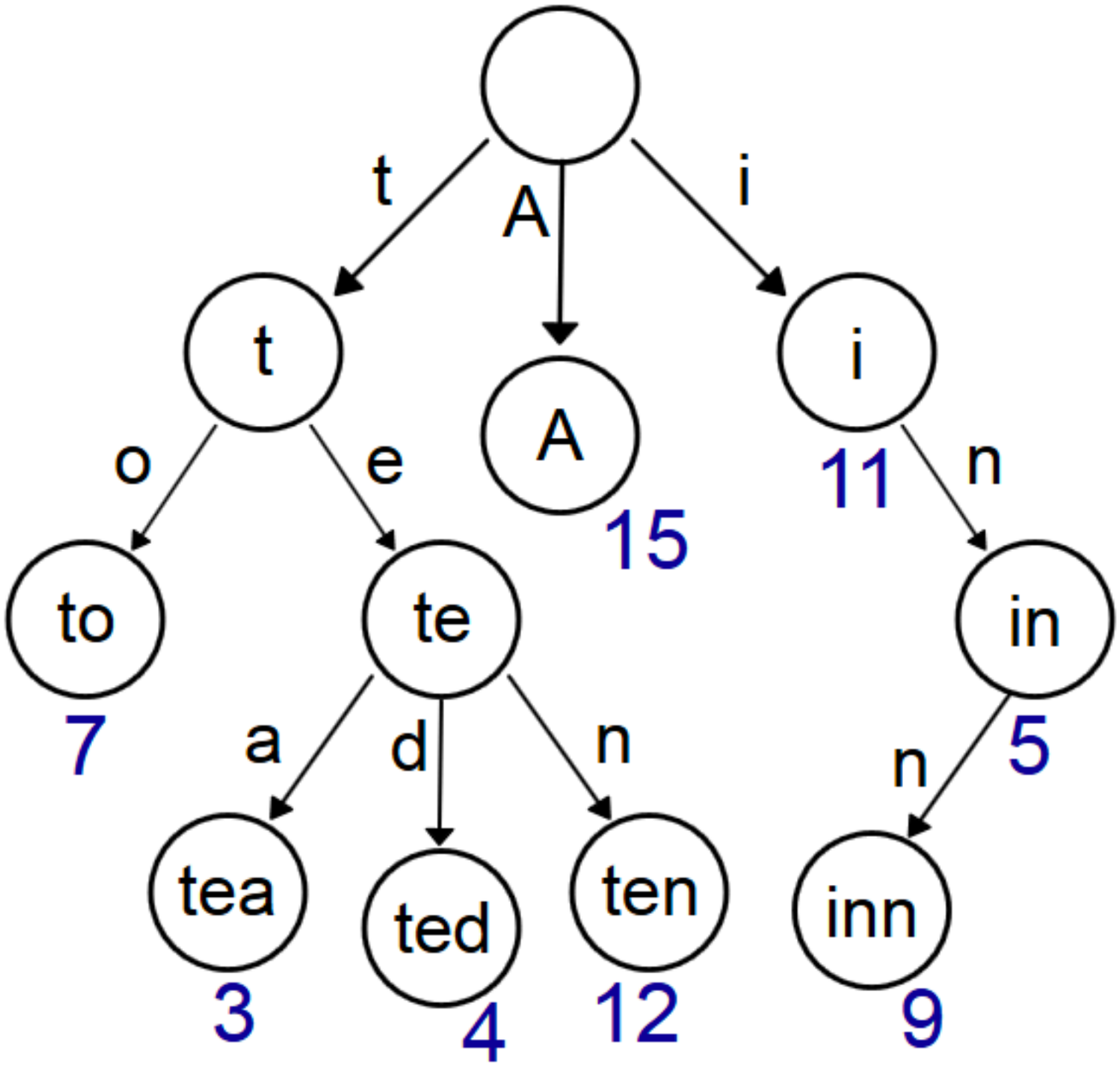


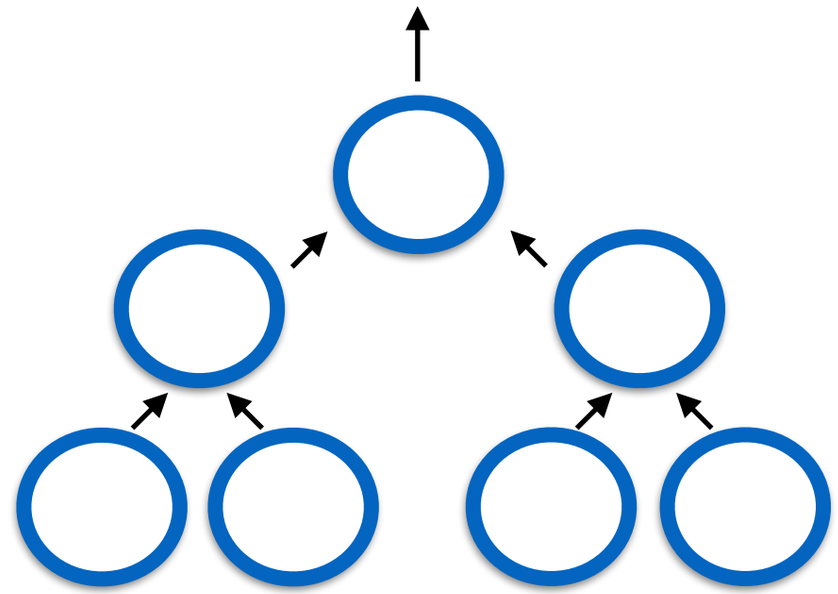
Ethereum Merkle Patricia Tree





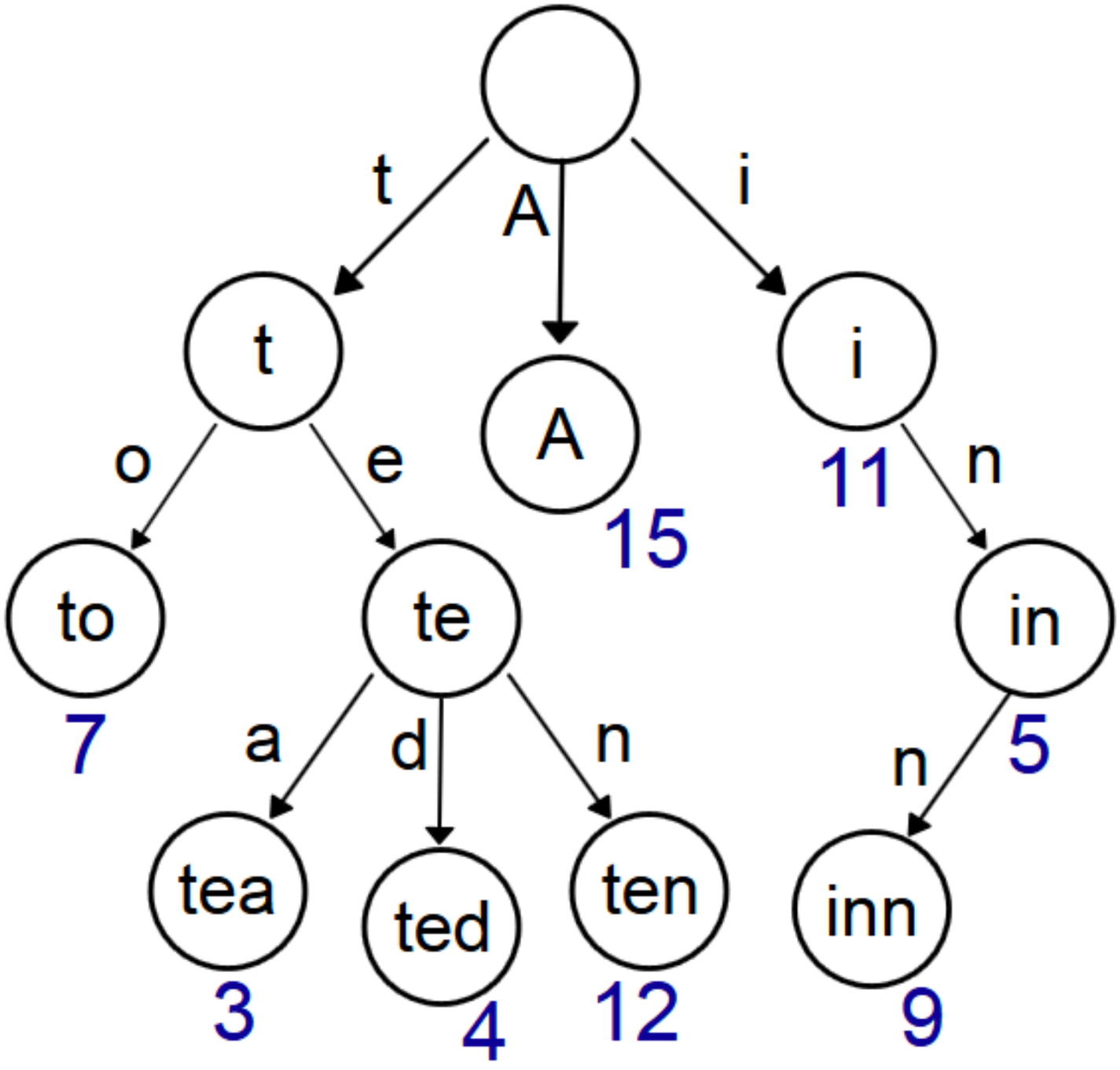
Ethereum Merkle Patricia Tree

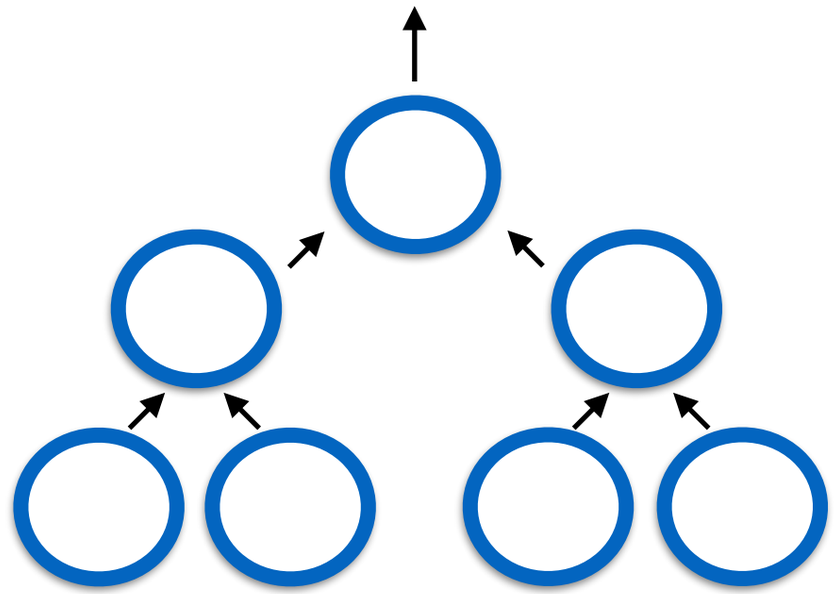


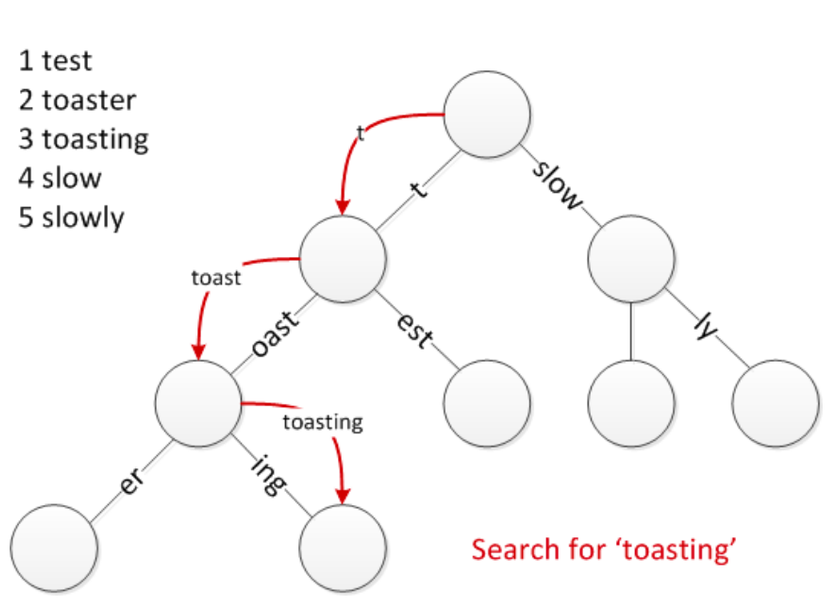


A trie for keys "A","to", "tea", "ted", "ten", "i", "in", and "inn".

Ethereum Merkle Patricia Tree







A trie for keys "A","to", "tea", "ted", "ten", "i", "in", and "inn".

Ethereum Merkle Patricia Tree

Ethereum Modified Merkle-Paricia-Trie System

An interpretation of the Ethereum Project Yellow Paper

G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", 2014.

Lee Thomas

Ver 0.0 2016-06-23

Block Header, H or B_H

stateRoot, H_r

Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied

Hash function:

KECCAK256()

Simplified World State, σ

Keys							Values
a	7	1	1	3	5	5	45.0 ETH
a	7	7	d	3	3	7	1.00 WEI
a	7	f	9	3	6	5	1.1 ETH
a	7	7	d	3	9	7	0.12 ETH

World State Trie

ROOT: Extension Node

prefix	*shared nibble(s)*	*next node*
0	a7	

Branch Node

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*a*	*b*	*c*	*d*	*e*	*f*	*value*

Leaf Node

prefix	*key-end*	*value*
2	1355	45.0ETH

Extension Node

prefix	*shared nibble(s)*	*next node*
0	d3	

Leaf Node

prefix	*key-end*	*value*
2	9365	1.1ETH

Prefixes

0 – Extension Node, even number of nibbles

1□ – Extension Node, odd number of nibbles,

2 – Leaf Node, even number of nibbles

3□ – Leaf Node, odd number of nibbles

□ = 1st nibble

1 nibble = 4 bits

Branch Node

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*a*	*b*	*c*	*d*	*e*	*f*	*value*

Leaf Node

prefix	*key-end*	*value*
3□	7	1.00WEI

Leaf Node

prefix	*key-end*	*value*
3□	7	0.12ETH

Ethereum Accounts and Transactions

Ethereum addresses are accounts or contracts

address	code	storage	balance	nonce

	account	contract
address	Hash(pub_key)	H(creator, nonce)
code	∅	EVM code
storage	∅	Merkle storage root
balance	ETH balance	
nonce	Number of transactions sent	

Three types of transactions in Ethereum

type	from	sig	nonce	to	data	value	gaslimit	gasprice

Three types of transactions in Ethereum

type	from	sig	nonce	to	data	value	gaslimit	gasprice

send	sender	sig	nonce	receiver	∅	amount	?	?

Three types of transactions in Ethereum

type	from	sig	nonce	to	data	value	gaslimit	gasprice

send	sender	sig	nonce	receiver	∅	amount	?	?

create	creator	sig	nonce	∅	code	start_bal	?	?

Three types of transactions in Ethereum

type	from	sig	nonce	to	data	value	gaslimit	gasprice

send	sender	sig	nonce	receiver	∅	amount	?	?

create	creator	sig	nonce	∅	code	start_bal	?	?

call	caller	sig	nonce	contract	f, args	amount	?	?

Ethereum Virtual Machine

Ethereum Virtual Machine

EVM code

```
.code
  PUSH 60             contract Ballot {\n
    struct...
  PUSH 40             contract Ballot {\n
    struct...
  MSTORE              contract Ballot {\n
    struct...
  CALLVALUE                function Ballot(uint8 _numProp...
  ISZERO              function Ballot(uint8 _numProp...
  PUSH [tag] 1             function Ballot(uint8 _numProp...
  JUMPI               function Ballot(uint8 _numProp...
  PUSH 0              function Ballot(uint8 _numProp...
  DUP1                function Ballot(uint8 _numProp...
  REVERT              function Ballot(uint8 _numProp...
tag 1                 function Ballot(uint8 _numProp...
  JUMPDEST                 function Ballot(uint8 _numProp...
  PUSH 40             function Ballot(uint8 _numProp...
  MLOAD               function Ballot(uint8 _numProp...
  PUSH 20             function Ballot(uint8 _numProp...
  DUP1                function Ballot(uint8 _numProp...
  PUSHSIZE                 function Ballot(uint8 _numProp...
  DUP4                function Ballot(uint8 _numProp...
  CODECOPY                 function Ballot(uint8 _numProp...
  DUP2                function Ballot(uint8 _numProp...
  ADD                 function Ballot(uint8 _numProp...
  PUSH 40             function Ballot(uint8 _numProp...
  MSTORE              function Ballot(uint8 _numProp...
  DUP1                function Ballot(uint8 _numProp...
  DUP1                function Ballot(uint8 _numProp...
```

EVM Features

- Stack of max depth of 1024
- 32-byte words
- Dedicated crypto opcodes
  - SHA-3
- Big num multiply
- GF-256 operators

Ethereum Memory

Storage: $\{0,1\}^{256}$ —> $\{0,1\}^{256}$ map (permanent)

Memory: $\{0,1\}^{256}$ —> $\{0,1\}^{256}$ map (volatile)

- Memory is zero initialized
- Memory is arranged in 256-bit words
- Storage is very **expensive**

Ethereum Memory

Storage: $\{0,1\}^{256}$ —> $\{0,1\}^{256}$ map (permanent)

Memory: $\{0,1\}^{256}$ —> $\{0,1\}^{256}$ map (volatile)

- Memory is zero initialized
- Memory is arranged in 256-bit words
- Storage is very **expensive**

Yellowpaper --> fee of 20k gas to store a 256 bit word
Gas Price = 10 Gwei = 10^10 Wei = 10^(-8) ETH
1 kilobyte --> 640k gas --> 0.0064 ETH = 6.4 USD

The cost of storing 1 kb is currently 6.4 USD

EVM provides an API for programmer

Input

- Transaction information: sender, value, gas limit
- Resource usage: gas remaining, memory used
- Block info: depth, timestamp, miner, hash

Output

- Sent messages
- Write to logs
- Self destruct

Ethereum Transaction Fees

Gas

Gas is the equivalent of transaction fees in Ethereum

- Each opcode of the EVM has a certain gas cost
- Limits computational cost and DoS attacks
- Miners choose transactions based on GAS_PRICE
- Overpriced opcodes might be preferred
- Maximum GAS_LIMIT per block

Like in Bitcoin

- All miners needs to
  - Execute all transactions
  - Store code

Unlike Bitcoin

- GAS_LIMIT may halt execution abruptly

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

1.0 gas costs - Google Sheets

Secure | https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs/edit#gid=0

1.0 gas costs

File Edit View Insert Format Data Tools Add-ons Help

arthur.gervais@chainsolutions.com

Share

100% View only

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		Approximations						Gas price						
2	Param	Compute (µs)	History (bytes)	State (bytes)	Bandwidth	Bloom topic	Mem quad	Computed	**Actual**					Coefficient
3	DUP	3						3	**3**	FASTESTSTEP				
4	SWAP	3						3	**3**	FASTESTSTEP				
5	PUSH	3						3	**3**	FASTESTSTEP		Max execution time (us)	3141592	
6												Max history growth per day (MB)	3217	6.7(
7	ADD	3						3	**3**	FASTESTSTEP		Max state growth per day (MB)	113	190
8	MUL	5						5	**5**	FASTSTEP		Max block size	50.25	6
9	SUB	3						3	**3**	FASTESTSTEP		Max bloom topics per block	12566	250
10	DIV	5						5	**5**	FASTSTEP		Max memory (MB)	39	0.00196
11	SDIV	5						5	**5**	FASTSTEP				
12	MOD	5						5	**5**	FASTSTEP				
13	SMOD	5						5	**5**	FASTSTEP		Gas limit	3141592	
14	ADDMOD	8						8	**8**	MIDSTEP				
15	MULMOD	8						8	**8**	MIDSTEP				
16	EXPBASE	10						10	**10**	SLOWSTEP				
17	EXPBYTE	10						10	**10**					
18	SIGNEXTEND	5						5	**5**	FASTSTEP				
19	LT	3						3	**3**	FASTESTSTEP				
20	GT	3						3	**3**	FASTESTSTEP				
21	SLT	3						3	**3**	FASTESTSTEP				
22	SGT	3						3	**3**	FASTESTSTEP				
23	EQ	3						3	**3**	FASTESTSTEP				
24	ISZERO	3						3	**3**	FASTESTSTEP				
25	AND	3						3	**3**	FASTESTSTEP				
26	OR	3						3	**3**	FASTESTSTEP				
27	XOR	3						3	**3**	FASTESTSTEP				
28	NOT	3						3	**3**	FASTESTSTEP				
29	BYTE	3						3	**3**	FASTESTSTEP				
30	SHA3BASE	30						30	**30**					
31	SHA3WORD	6						6	**6**					
32	ECRECOVER	3000						3000	**3000**					

Sheet1 Sheet2

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Transaction Gas

Transaction creator specifies

from	sig	nonce	to	data	value	**gaslimit**	**gasprice**

If GAS_LIMIT x GAS_PRICE > accounts[from].balance, halt

Update the balance

accounts[from].balance -= value + gas x gasprice
accounts[to].balance += value
execute(code)
accounts[from] += unusedGas x gasprice

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Out-of-gas Exception

If remaining GAS is 0 before termination, throw out-of-gas

- State reverts to previous state
- GAS_LIMIT x GAS_PRICE is deducted to pay miners

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

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

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

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

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

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

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

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

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

10 gas

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

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

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

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

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

10 gas

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

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

```
Contract C {
function c():
 assert(msg.gas == 5);
   while (true) {
       Loop
   }
 return "Bonjour"
```

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

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

10 gas

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

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

5 gas

```
Contract C {
function c():
 assert(msg.gas == 5);
   while (true) {
       Loop
   }
 return "Bonjour"
```

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

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

10 gas

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

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

5 gas

```
Contract C {
function c():
 assert(msg.gas == 5);
   while (true) {
       Loop
   }
 return "Bonjour"
```

Out of Gas Exception

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

Caller determines the sent gas

100 gas

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

10 gas

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

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

5 gas

```
Contract C {
function c():
 assert(msg.gas == 5);
   while (true) {
       Loop
   }
 return "Bonjour"
```

returns “Hello World”

Out of Gas Exception

Gas costs: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs

How do you think miners order transactions in a block?

Solidity

Solidity

- Looks familiar to JavaScript
- Behaves very differently!
- Contracts look similar to classes
- Functions are public by default
- Static typing

- Solidity source code should start with

```
pragma solidity ^0.5.3;
```

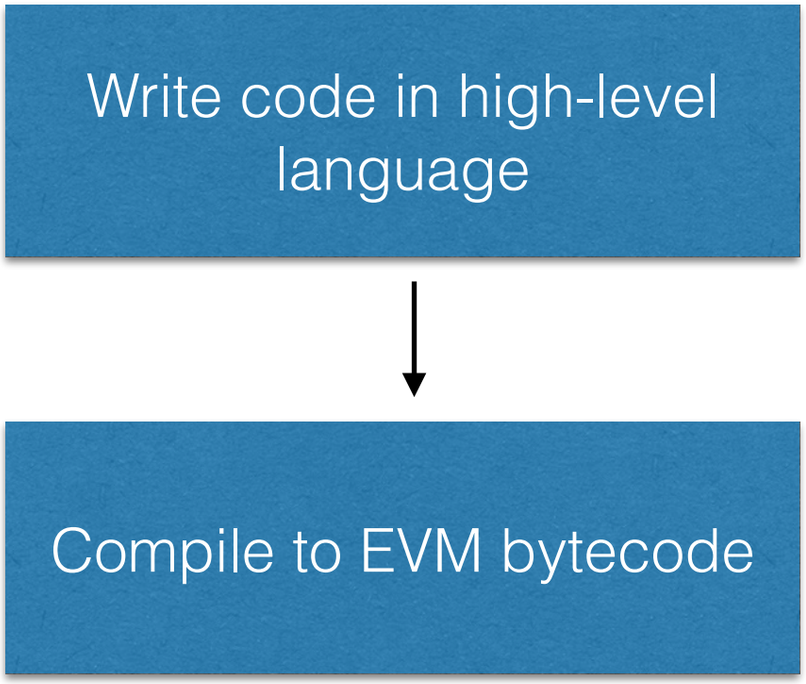
- Good resource

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

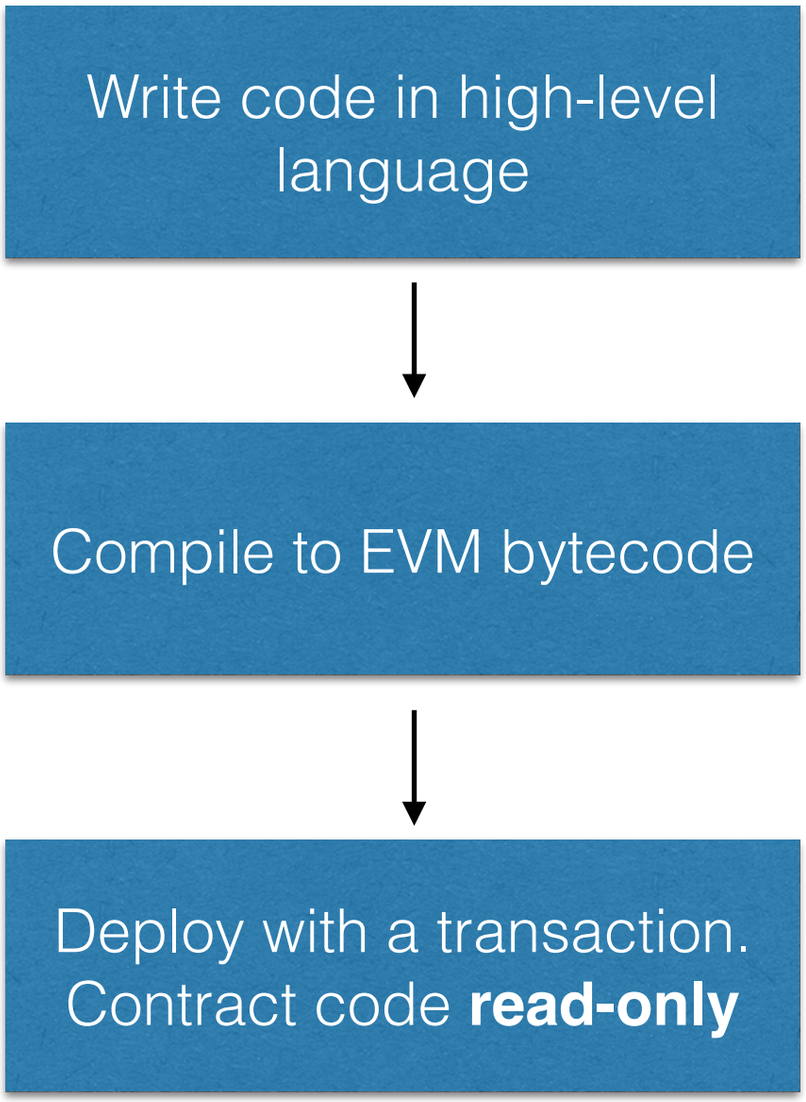
Smart Contract Lifecycle

Write code in high-level language

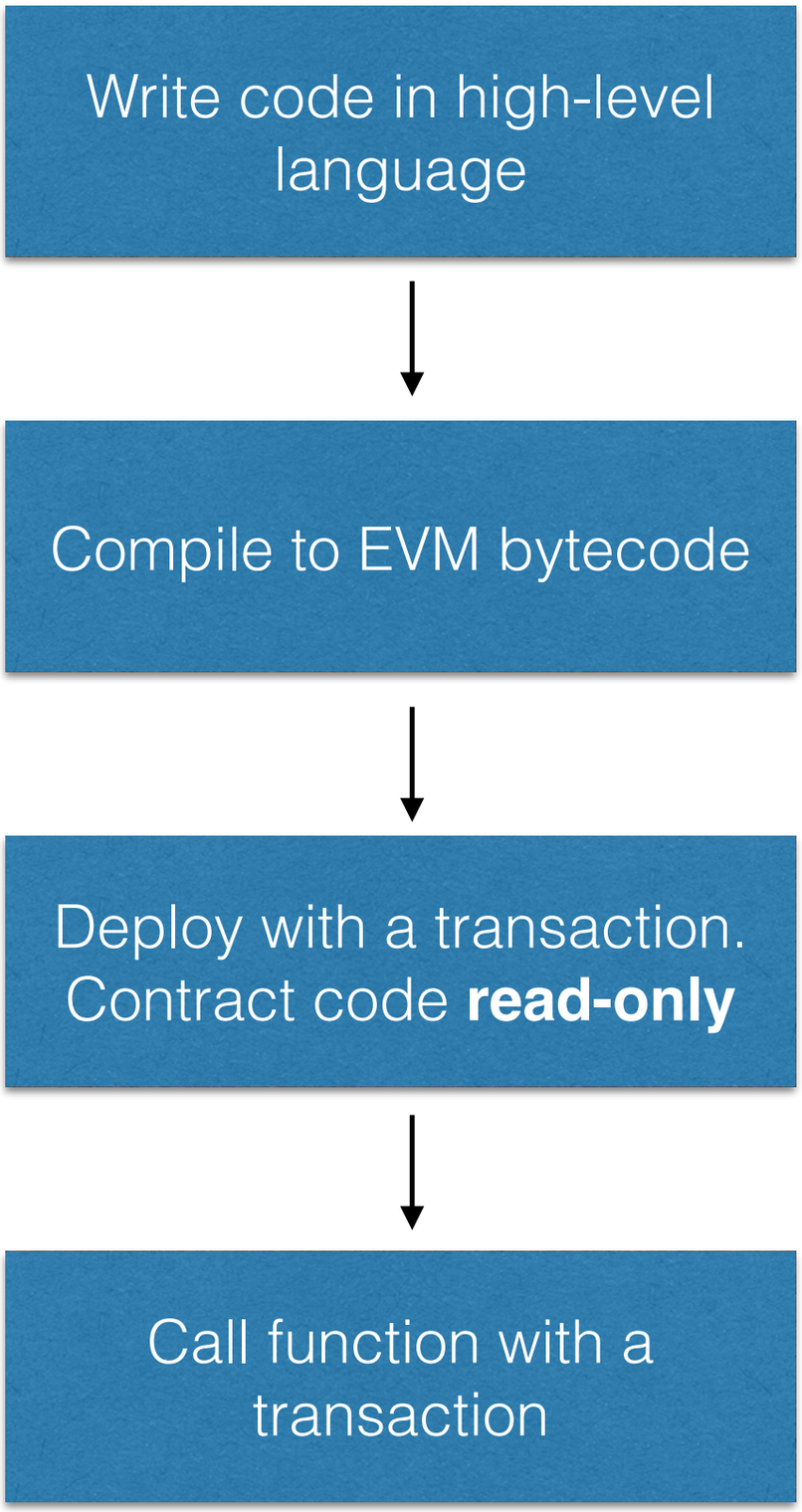
Smart Contract Lifecycle



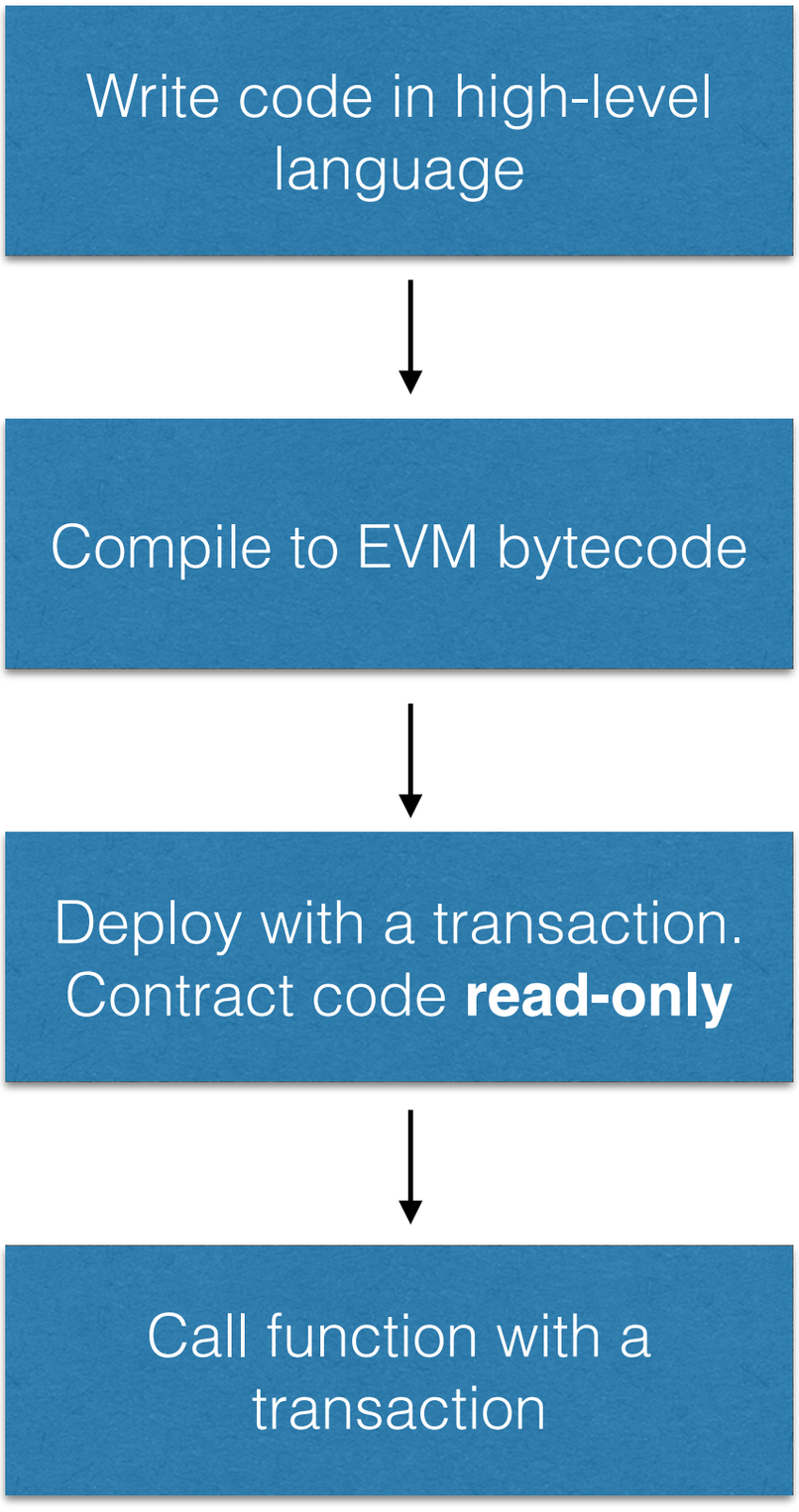
Smart Contract Lifecycle



Smart Contract Lifecycle



Smart Contract Lifecycle



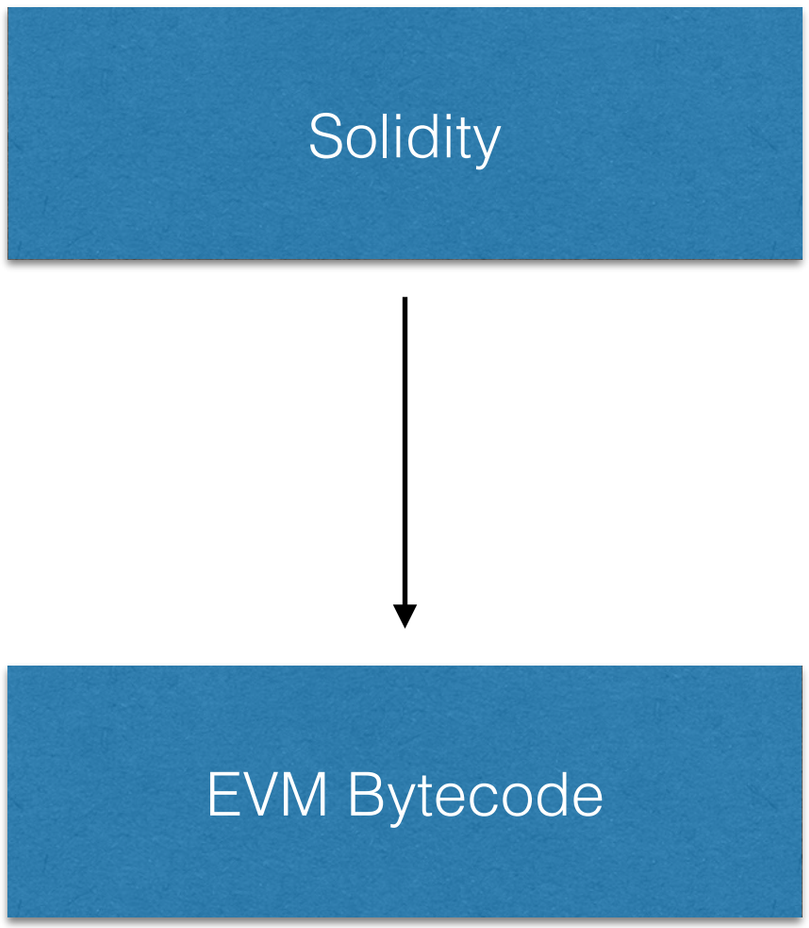
- Lives forever if no destruct defined

  —> no keep-alive cost

- Contracts can call other contracts

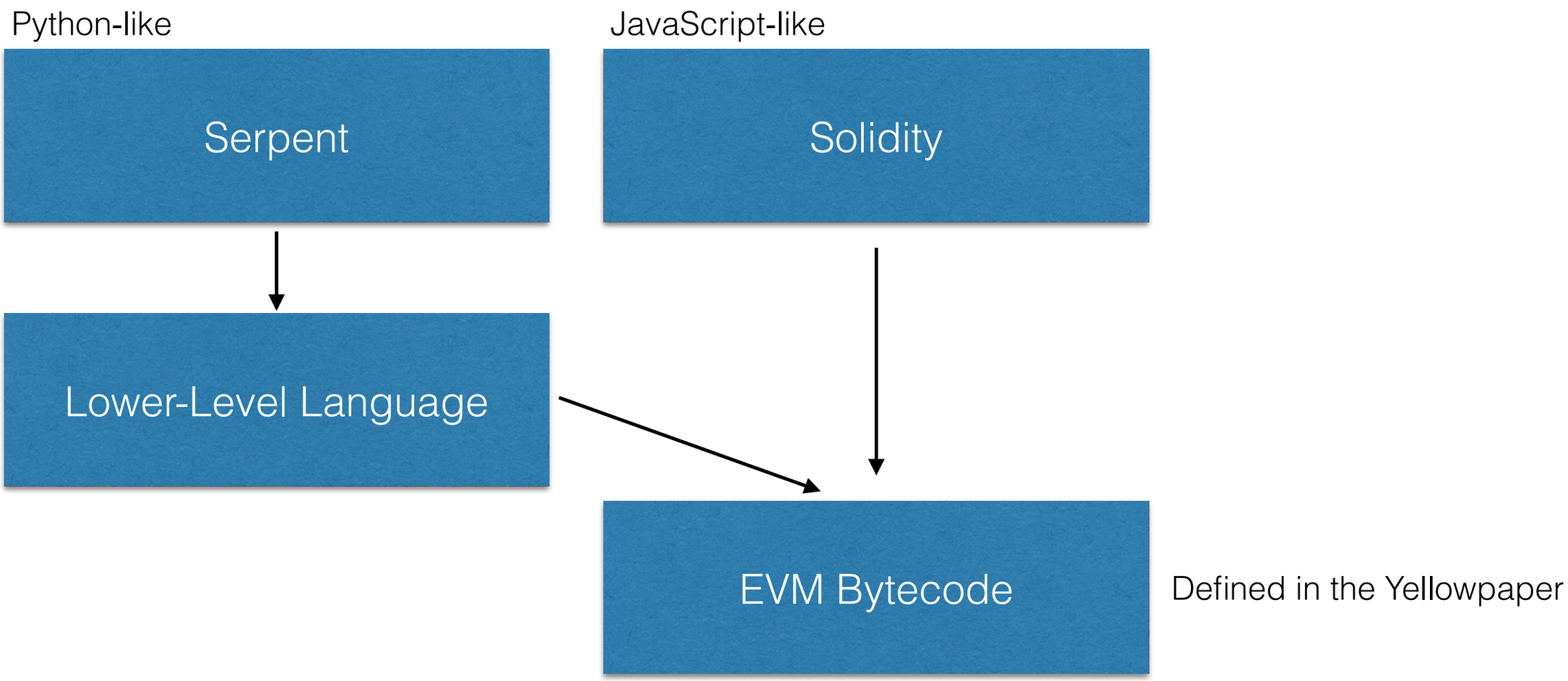
Smart Contract Development

JavaScript-like



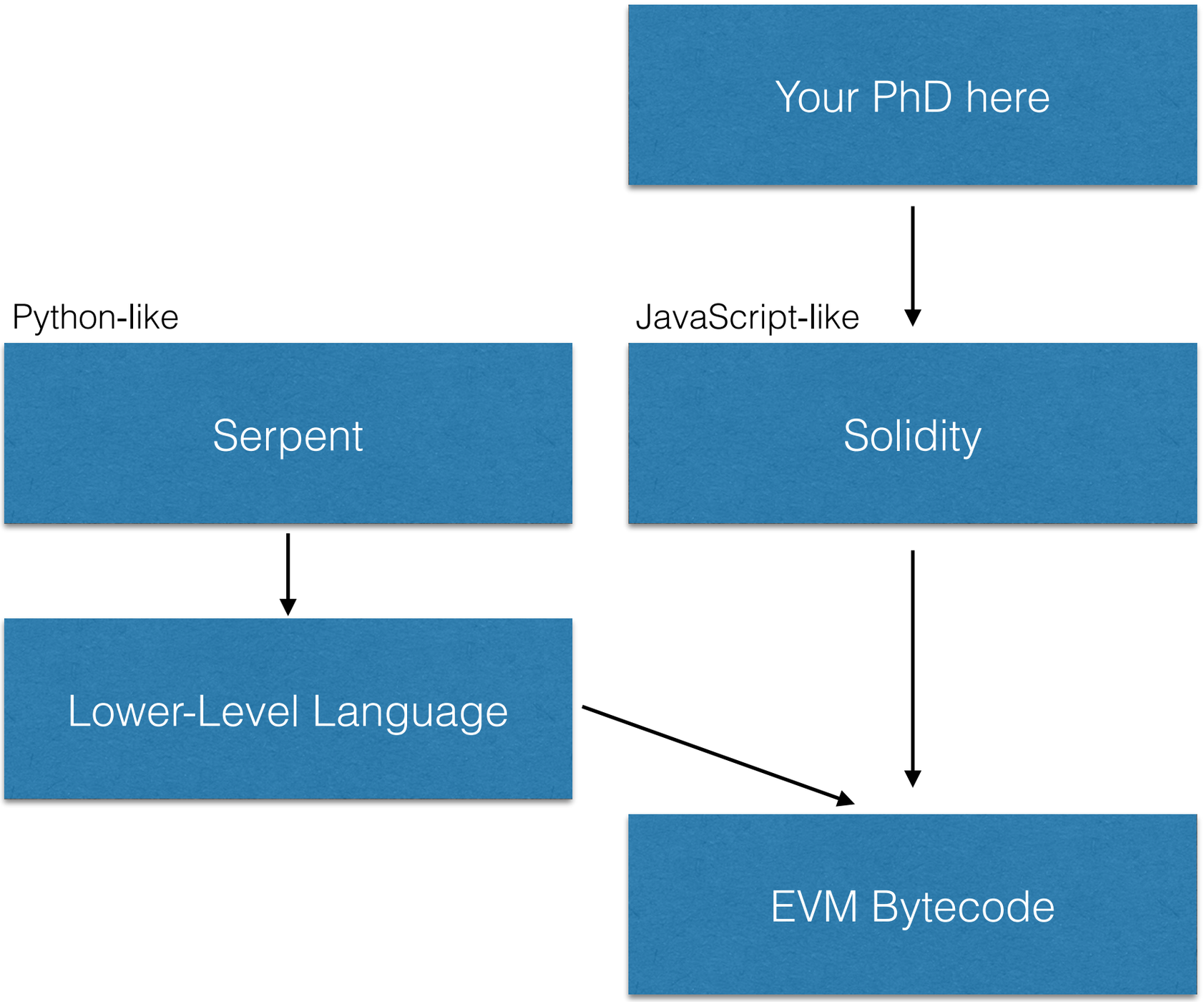
Defined in the Yellowpaper

Smart Contract Development



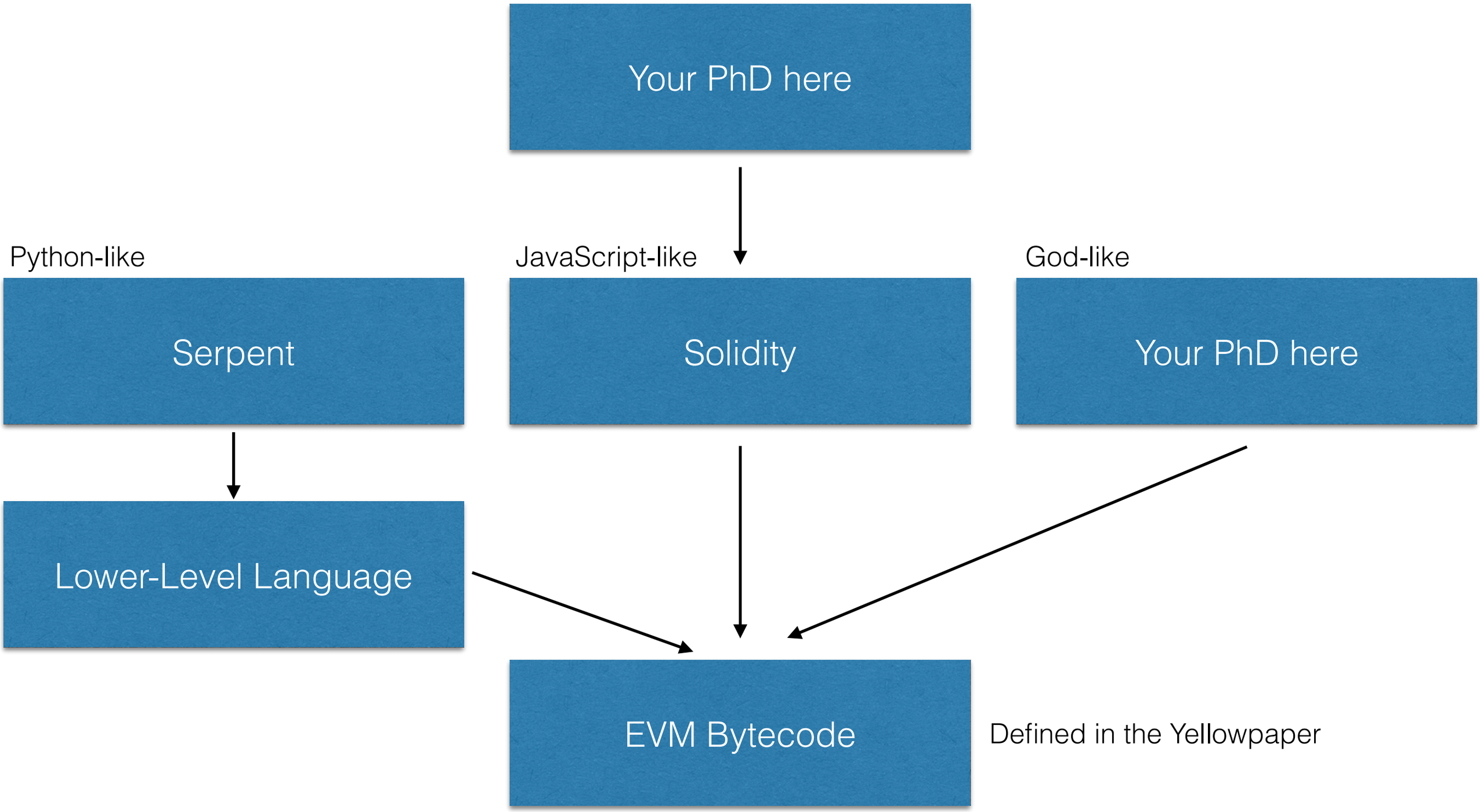
Defined in the Yellowpaper

Smart Contract Development

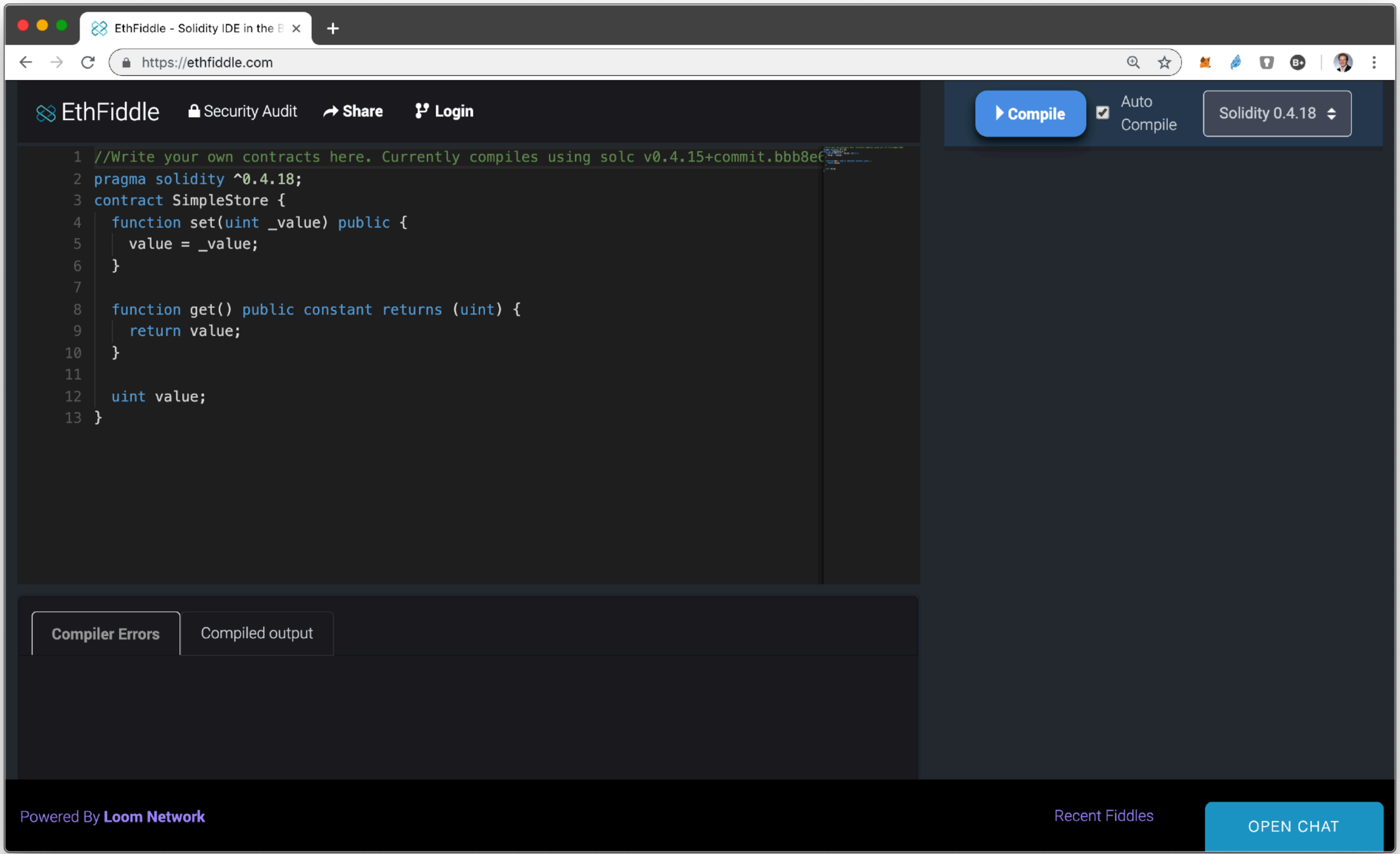


Defined in the Yellowpaper

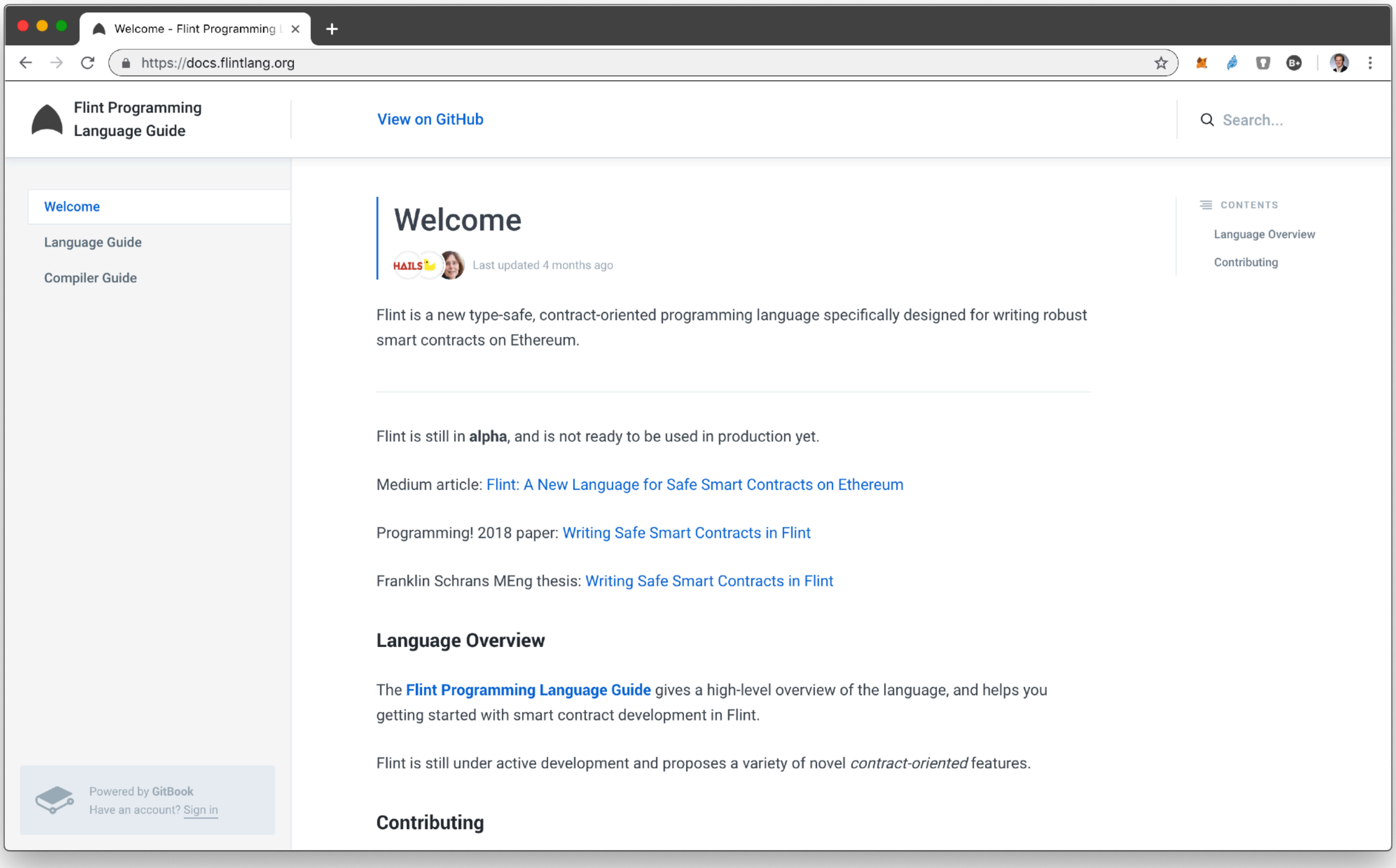
Smart Contract Development



ETHFiddle


EthFiddle - Solidity IDE in the B
https://ethfiddle.com
EthFiddle
Security Audit
Share
Login
Compile
Auto Compile
Solidity 0.4.18
1 //Write your own contracts here. Currently compiles using solc v0.4.15+commit.bbb8e6
2 pragma solidity ^0.4.18;
3 contract SimpleStore {
4 function set(uint _value) public {
5 value = _value;
6 }
7
8 function get() public constant returns (uint) {
9 return value;
10 }
11
12 uint value;
13 }
Compiler Errors
Compiled output
Powered By Loom Network
Recent Fiddles
OPEN CHAT

Flint - a language to replace Solidity



Solidity - Types

```
bool, uint8, uint16, ... uint256, int8, ... int256

address

string

byte[]

mapping(keyType ==> valueType)
```

- State variables reside in storage

Solidity - Throw

```
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 makes sure that only gas is consumed.

Solidity - State Variables

```
pragma solidity ^0.5.3;

contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

- State variables reside in storage

Solidity - Functions

```
pragma solidity ^0.5.3;

contract SimpleAuction {
    function bid() public payable { // Function
        // ...
    }
}
```

- Internal/External function calls

- Visibility
  - External (can be called from other contracts)
  - Public (like external, can also be called internally)
  - Internal (can only be called internally)
  - Private (not visible in derived contracts)

Solidity - Modifiers

```
contract Purchase {
    address public seller;

    modifier onlySeller() { // Modifier
        require(msg.sender == seller);
        _;
    }

    function abort() public onlySeller { // Mod. usage
        // ...
    }
}
```

- Amend the semantics of functions

Solidity - Events

```
contract SimpleAuction {
    event HighestBidIncreased(address bidder, uint
amount); // Event

    function bid() public payable {
        // ...
        HighestBidIncreased(msg.sender, msg.value); //
Triggering event
    }
}
```

- Events interface with EVM's logging capabilities
- Allows to call JavaScript callbacks in a dApp
- Function bid() is marked *payable* to receive funds

Namecoin in Solidity

```
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)];
    }

}
```