Privacy in Cryptocurrency

Mixing and more advanced technologies

Leander Jehl

Privacy

Anonymity

Definition

Anonymity requires two properties:

- *Pseudonymity*
  You can interact without revealing your identify.

- *Unlinkability*
  An attacker is unable to connect transactions from the same user.

Creating *user profiles* will eventually allow to de-anonymize users.

Anonymity

Why

- Cryptocurrency is used for many illegitimate activities.
- Anonymity focused cryptocurrencies are associated with crime.

But:

- Tainted coins pose a problem (1$ is not 1$?)
- Deanonymization creates targets for criminal activity

Anonymity

Bitcoin

- Bitcoin uses Pseudonyms (addresses)
- UTXO favors unlikability:
  - Can use new address for every received coin (without extra cost)

More anonymous solutions usually build on UTXO.

Anonymity

Bitcoin - Regulations

- Due to regulations all exchanges for cryptocurrencies require identification and keep logs.
- Same counts for law-compliant services.
- Some privacy focused chains have special tools to disclose information to trusted parties.

Even if we get anonymity on chain,
exchange into and out of cryptocurrency are subject to regulations.

Anonymity

Linking Bitcoin transactions

- Link multiple addresses used for inputs into one transaction.
- Link address used for input and address used for change in transaction.
  - **Problem: Identify which output is change.**
- Identify regular money flows between users.
- Identify time of day
- Use network analysis to identify users IP or Location

Anonymity

Anonymity set

The *anonymity set* is the a set of users or transactions, such that an attacker is unable to identify which item in the set if yours.

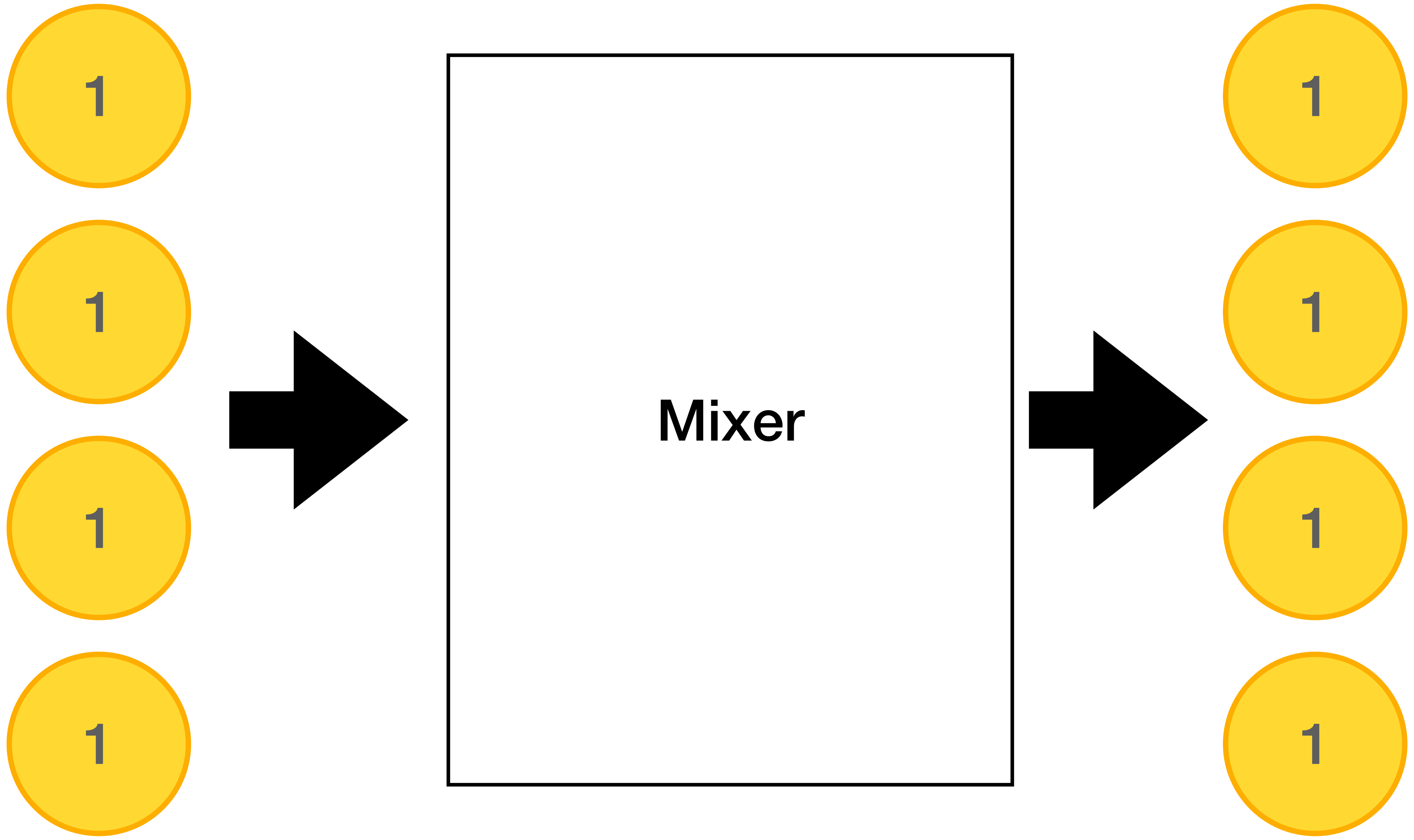
- Anonymity set is limitted to users/transactions using a certain feature/system.
- Large anonymity set is preferrable.

Mixing services

Anonymity

Mixing

- Mixing is an external service.
- Can send bitcoin to the service.
- Service shuffles coins.
- User recieves back a coin.



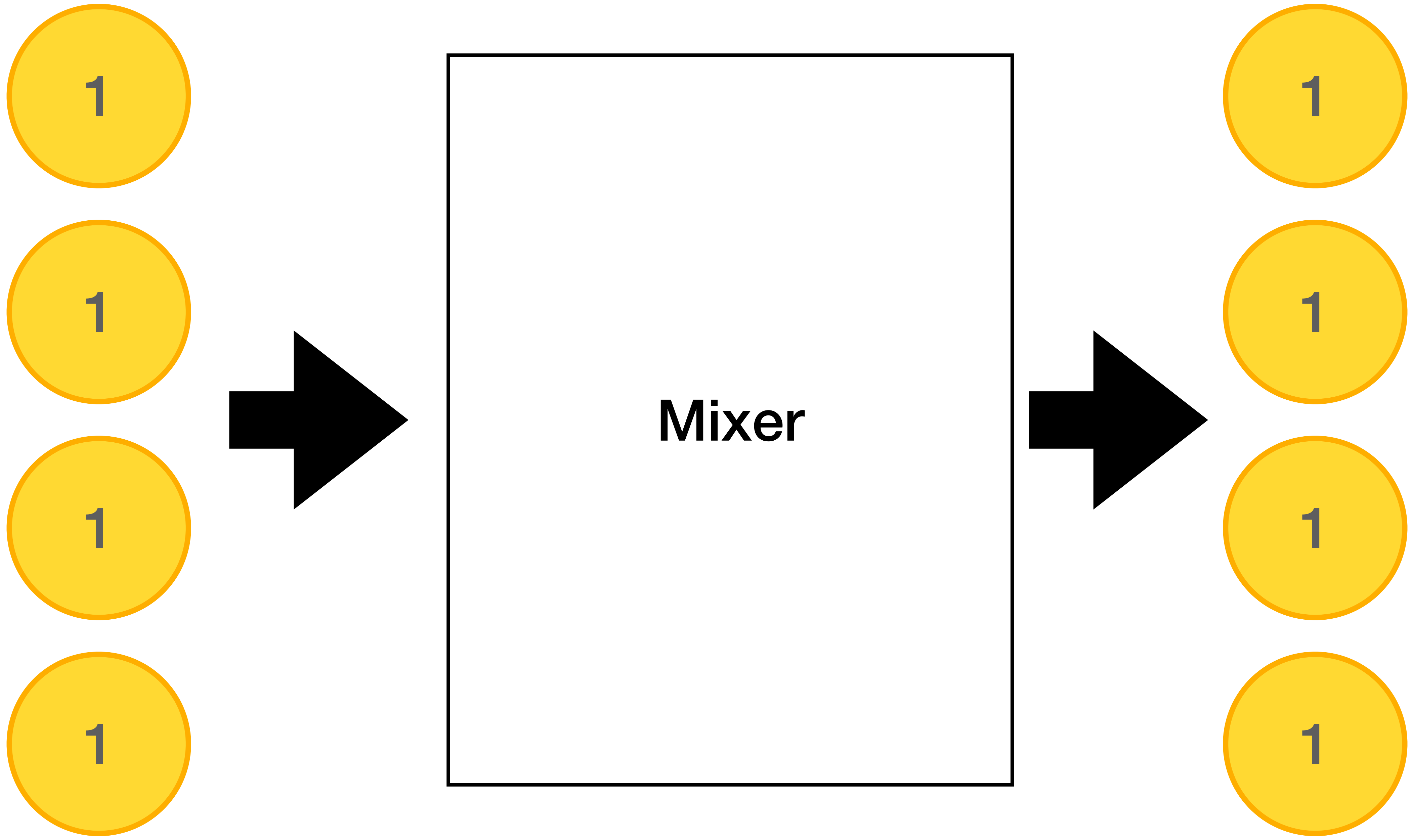
- Attacker cannot link outputs to specific inputs.

Anonymity

Mixing

Centralized mixer:

- Need to trust mixing service
- Service might get hacked
- Fees
- Few people are using it, and most do not have good intentions

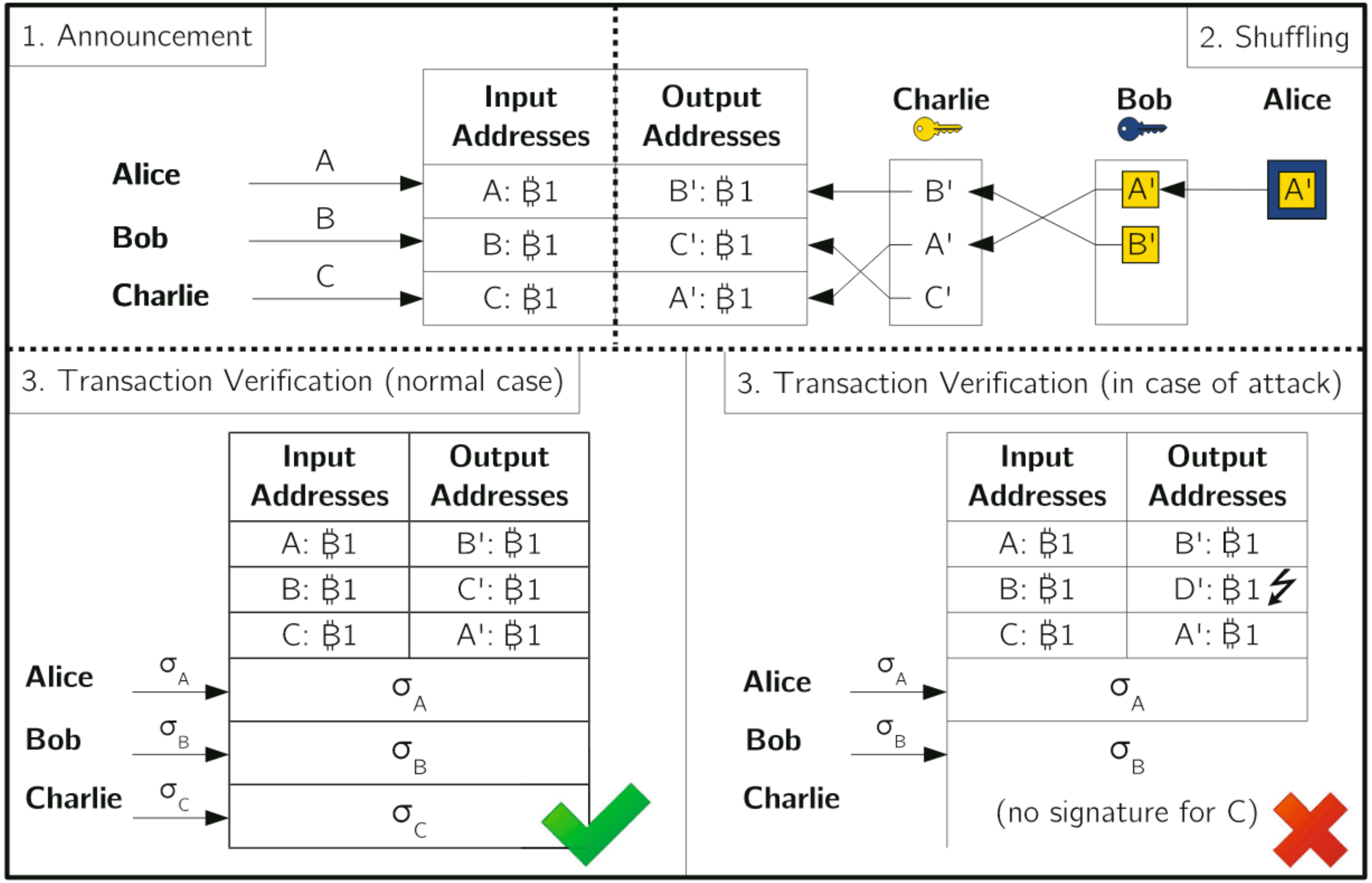


Anonymity

Dezentralized mixer (Coinshuffle)

Participants create mixing transaction through offchain interaction.

- No central service.
- But still limitted to few users.
- How to find users?



1. Announcement

2. Shuffling

Input Addresses	Output Addresses
A: ₿1	B': ₿1
B: ₿1	C': ₿1
C: ₿1	A': ₿1

3. Transaction Verification (normal case)

Input Addresses	Output Addresses
A: ₿1	B': ₿1
B: ₿1	C': ₿1
C: ₿1	A': ₿1
σ_A	
σ_B	
σ_C	

3. Transaction Verification (in case of attack)

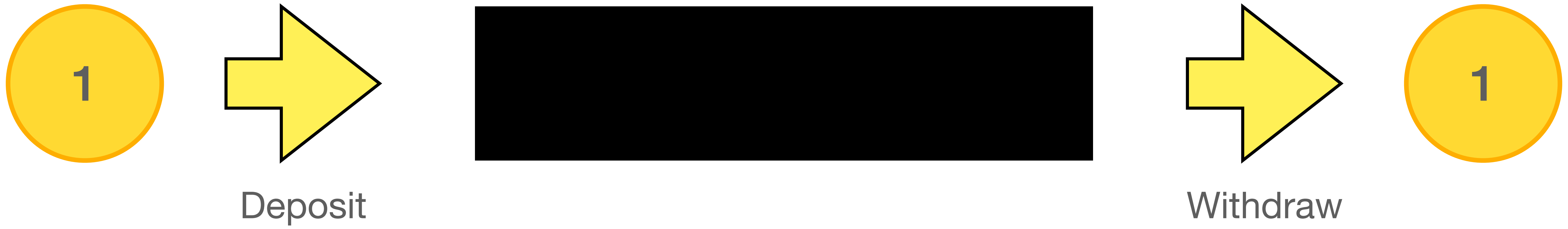
Input Addresses	Output Addresses
A: ₿1	B': ₿1
B: ₿1	D': ₿1 ⚡
C: ₿1	A': ₿1
σ_A	
σ_B	
(no signature for C)	

Altcoins

Altcoins

ZeroCoin

- Can change coin from Base currency to mixed currency (deposit) and back (withdraw).
- Cannot be tracked, i.e. impossible to identify which deposit is withdrawn.
- Anonymity set is: All deposits every made (with the same value).



Altcoins

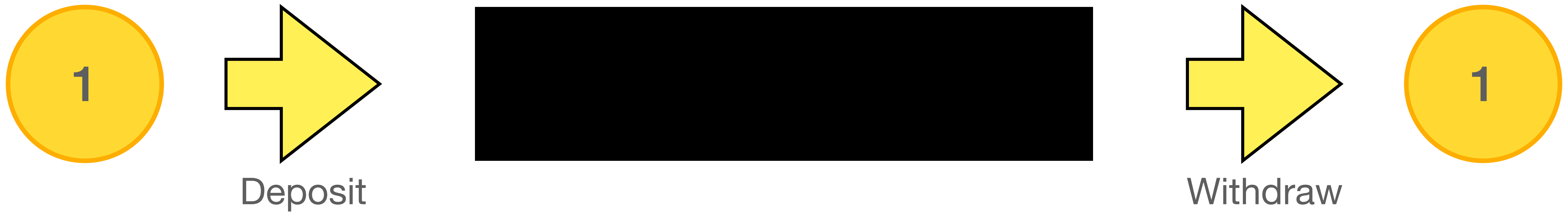
ZeroCoin

Deposit:

- Create sequence number *Sn* and secret *x.*
- Publish *Commit(Sn,x)* while burning 1 coin.

Withdraw:

- Publish *Sn* and *Zero-knowledger* proof of:
    - *I know x, such that Commit(Sn,x) is one of the commitments published on the chain.*



Altcoins

ZeroCoin

Deposit:

- Create sequence number *Sn* and secret *x.*
- Publish *Commit(Sn,x)* while burning 1 coin.

Withdraw:

- Publish *Sn* and *Zero-knowledge* proof of:
  - *I know x, such that Commit(Sn,x) is one of the commitments published on the chain.*

- *Sn* is published to prevent double spending.
- *x* is kept secret, to prevent linking.
- Problem: zero-knowledge proofs take space and are expensive to compute.

1
Deposit
Withdraw
1

Altcoins

Zero knowledge proof of knowledge

Given a function *f(x),* it is possible to create a proof machinery such that:

- Given a value x', and $y' = f(x')$ we can create a proof: (π, y') that shows, that:
    - I know x' such that $y' = f(x')$.
- This reveals nothing about x' , than what can be deduced from y'.
- f must be representable as a NP-circuit.

ZeroCoin:
Zero knowledge proof of set membership

Altcoins

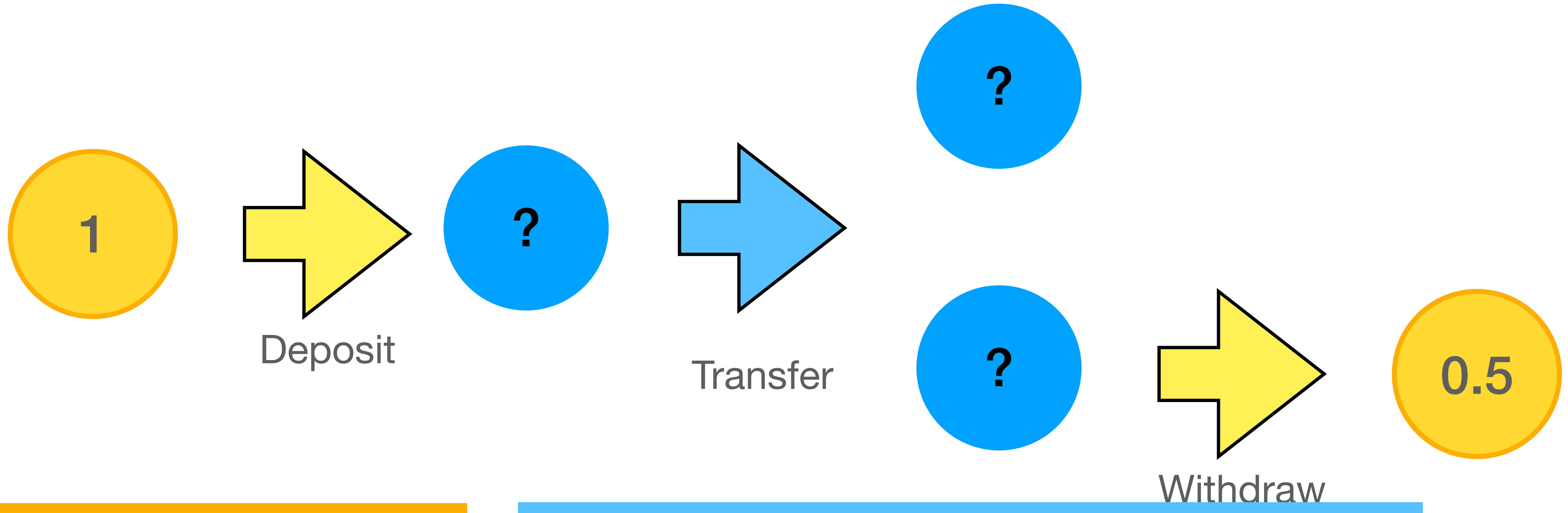
ZeroCash

Similar to ZeroCoin but can transfer deposited coins.

Transfer:
Similar to

- withdraw for used output,
- Deposit for new outputs
- Plus zk proof showing that

$$\sum input\ values = \sum output\ values$$



ZeroCash:
Uses more advance ZK; zkSNARK

Problem:
Requires complex trusted setup.

Use transparent setup: zkSTARK

Altcoins

Monero

Similar to bitcoin (PoW, UTXO). Privacy focused.

Outputs have encoded value.

Then issuing a transaction:

- Pick one of your outputs and 10 other ones.
- Create ring signature using your private key and public keys from all the inputs.
- Proof that outputs and input values are equal using novel zk-proofs.

SmartContracts

Private Smart Contracts

What to hide?

- Code
- Inputs
- State

State stored in Ethereum blockchain

Bitcoin

Block structure

Header:

```
PrevBlockhash
Nonce
Timestamp
```

Transaction data

```
Merkle tree
```

Merkle tree allows to easily proof that a transaction is included in a block.

Blockchain contains transactions
State (UTXO) stored in memory

Ethereum

Block structure

Header:

```
PrevBlockhash
Nonce
Timestamp

State root hash
Receipts root hash
```

Transaction data

```
Merkle tree
```

Blockchain contains also root of state

Ethereum

Block structure

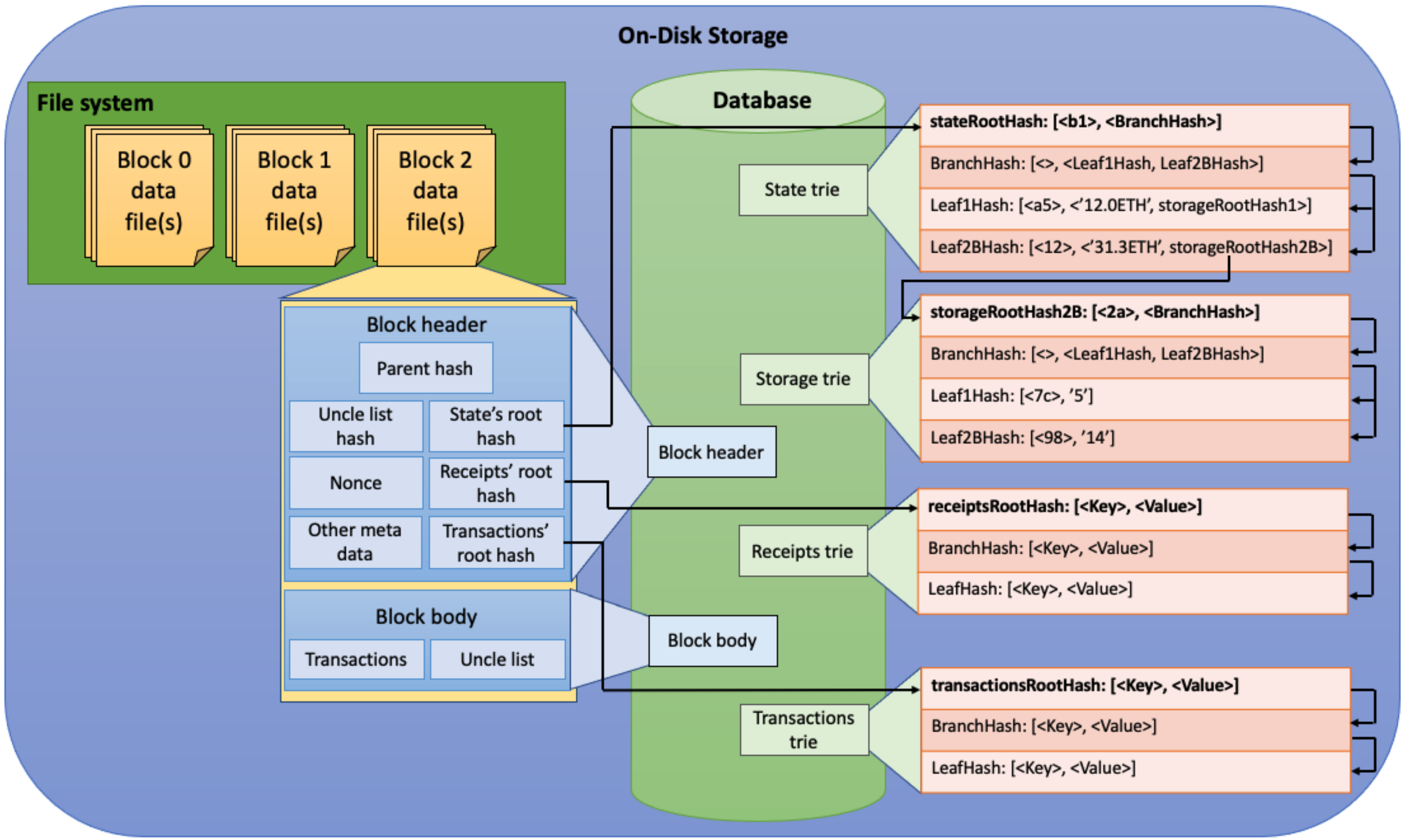
Header:

PrevBlockhash
Nonce
Timestamp

State root hash
Receipts root hash

Transaction data

Merkle tree


On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Block header
Parent hash
Uncle list hash
State's root hash
Nonce
Receipts' root hash
Other meta data
Transactions' root hash
Block body
Transactions
Uncle list
Database
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <'12.0ETH', storageRootHash1>]
Leaf2BHash: [<12>, <'31.3ETH', storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, '5']
Leaf2BHash: [<98>, '14']
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]

Ethereum

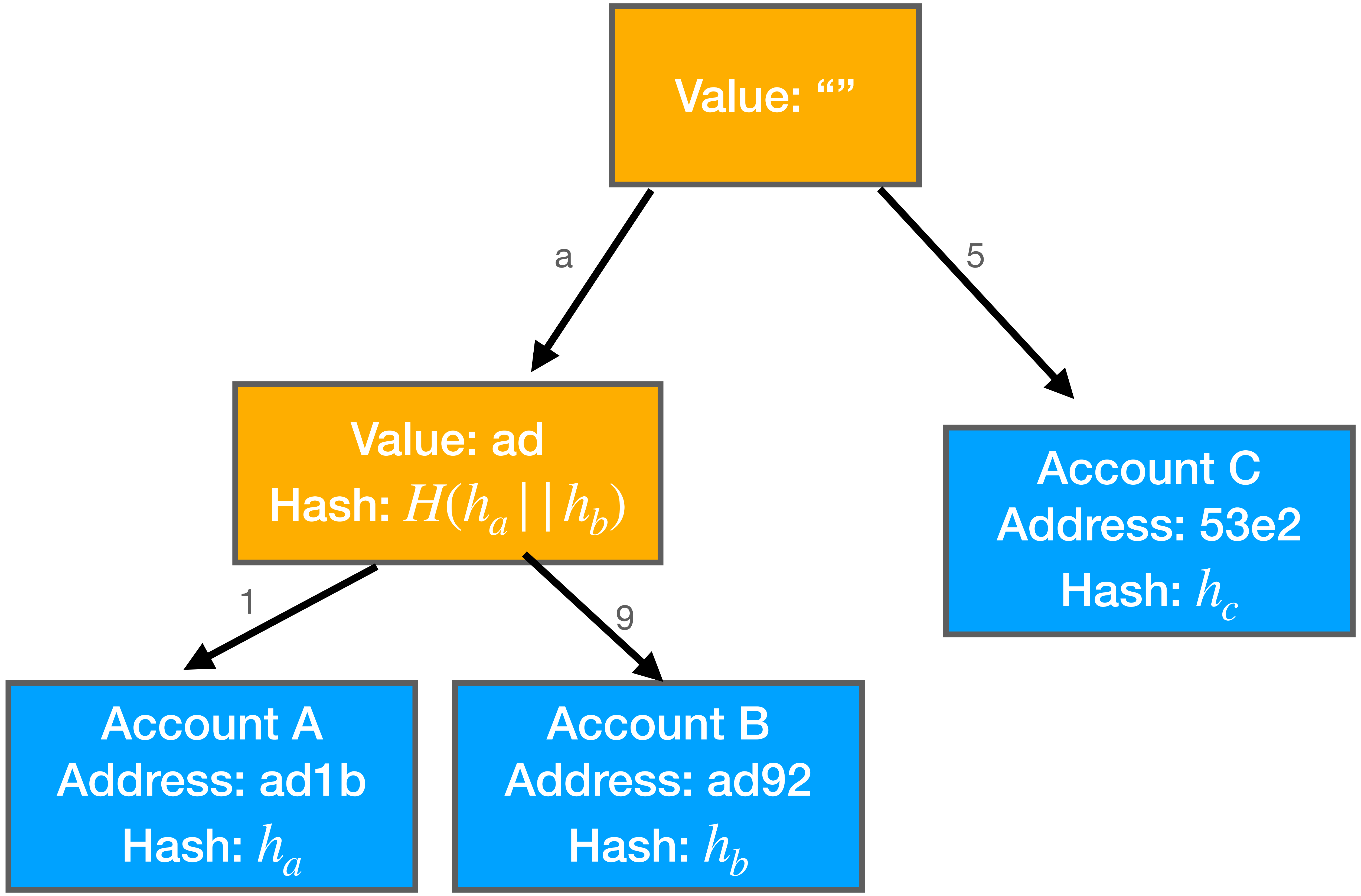
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



On new block, only changed nodes get added.

Ethereum

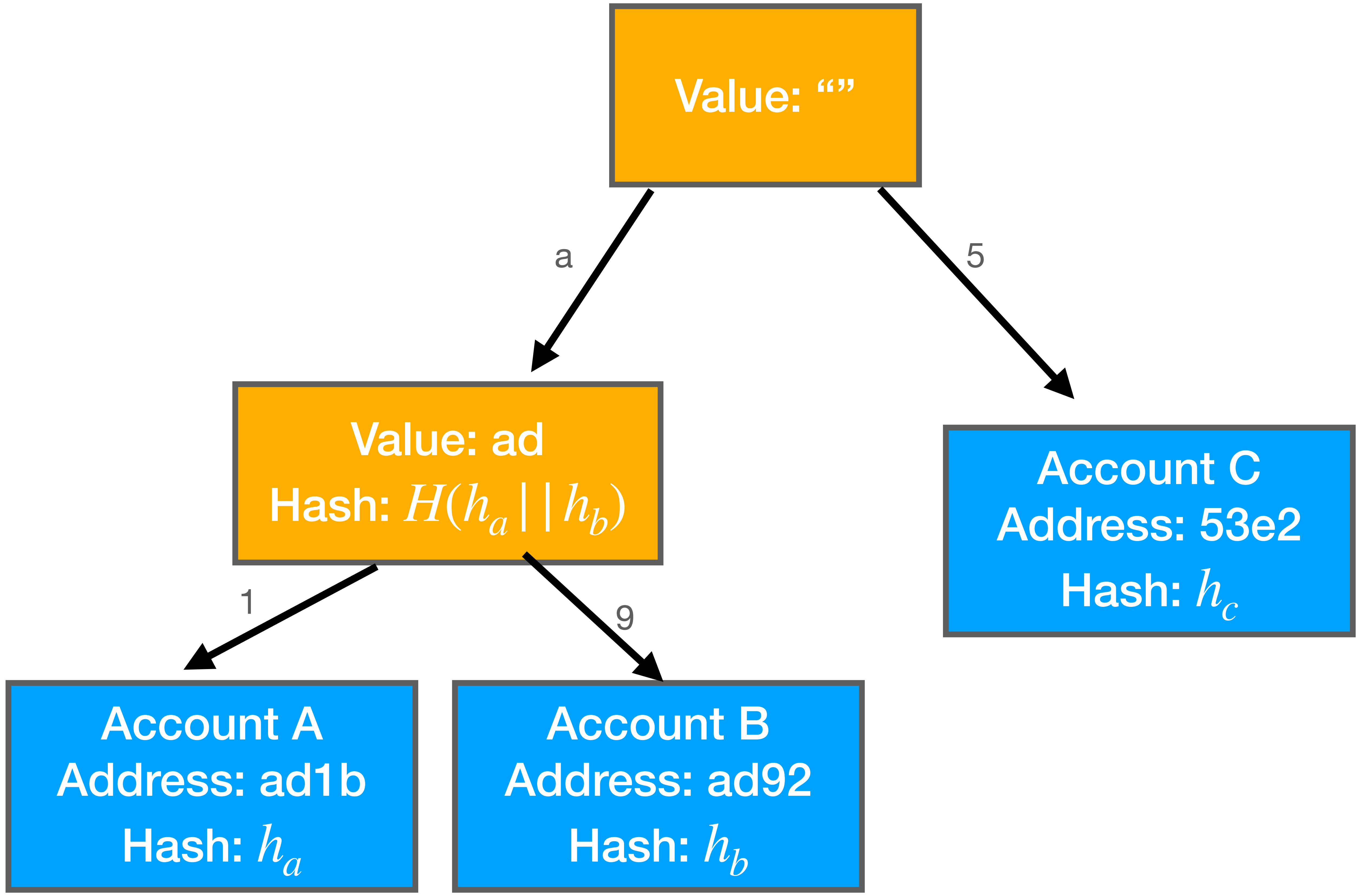
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



On new block, only changed nodes get added.

Ethereum

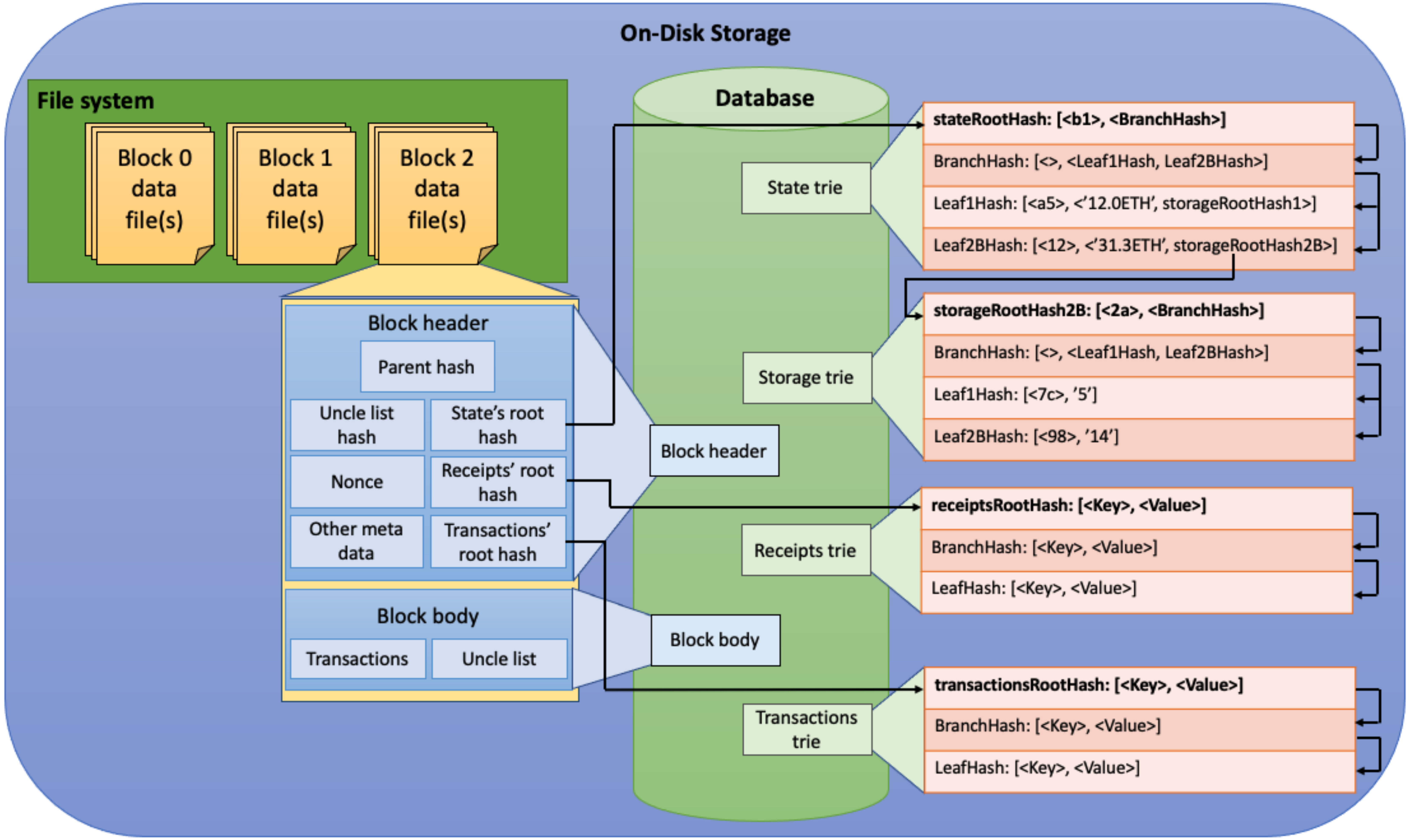
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```

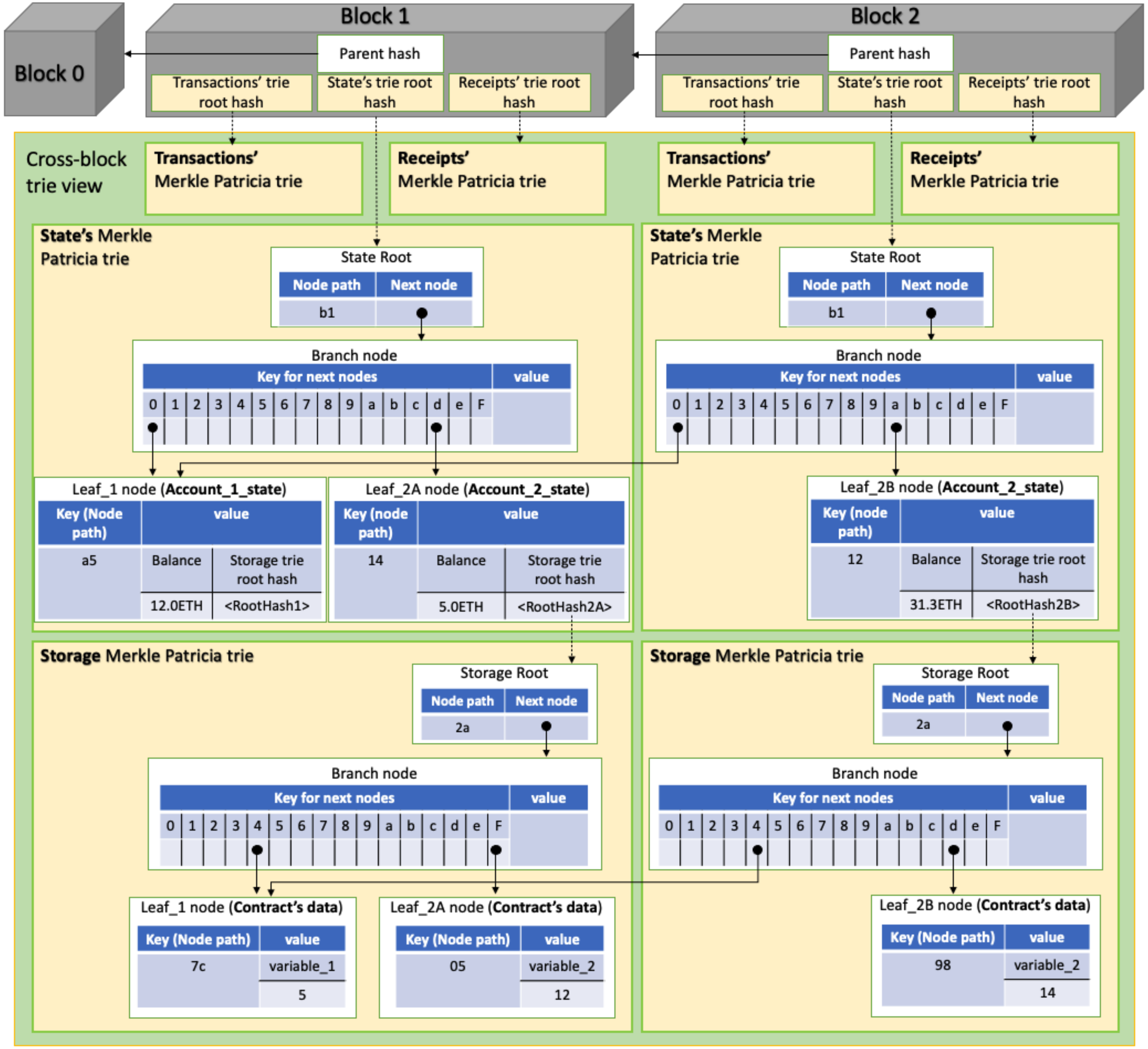


On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Block header
Parent hash
Uncle list hash
State's root hash
Nonce
Receipts' root hash
Other meta data
Transactions' root hash
Block body
Transactions
Uncle list
Database
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <'12.0ETH', storageRootHash1>]
Leaf2BHash: [<12>, <'31.3ETH', storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, '5']
Leaf2BHash: [<98>, '14']
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]

`StorageRoot` is the root of a different trie.

Ethereum

State trie



Block 0

Block 1

Parent hash		
Transactions' trie root hash	State's trie root hash	Receipts' trie root hash

Block 2

Parent hash		
Transactions' trie root hash	State's trie root hash	Receipts' trie root hash

Cross-block trie view

Transactions' Merkle Patricia trie

Receipts' Merkle Patricia trie

Transactions' Merkle Patricia trie

Receipts' Merkle Patricia trie

State's Merkle Patricia trie

State Root

Node path	Next node
b1	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
●													●			

Leaf_1 node (**Account_1_state**)

Key (Node path)	value	
a5	Balance	Storage trie root hash
	12.0ETH	<RootHash1>

Leaf_2A node (**Account_2_state**)

Key (node path)	value	
14	Balance	Storage trie root hash
	5.0ETH	<RootHash2A>

State's Merkle Patricia trie

State Root

Node path	Next node
b1	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
●										●						

Leaf_2B node (**Account_2_state**)

Key (node path)	value	
12	Balance	Storage trie root hash
	31.3ETH	<RootHash2B>

Storage Merkle Patricia trie

Storage Root

Node path	Next node
2a	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
				●											●	

Leaf_1 node (**Contract's data**)

Key (Node path)	value
7c	variable_1
	5

Leaf_2A node (**Contract's data**)

Key (Node path)	value
05	variable_2
	12

Storage Merkle Patricia trie

Storage Root

Node path	Next node
2a	●

Branch node

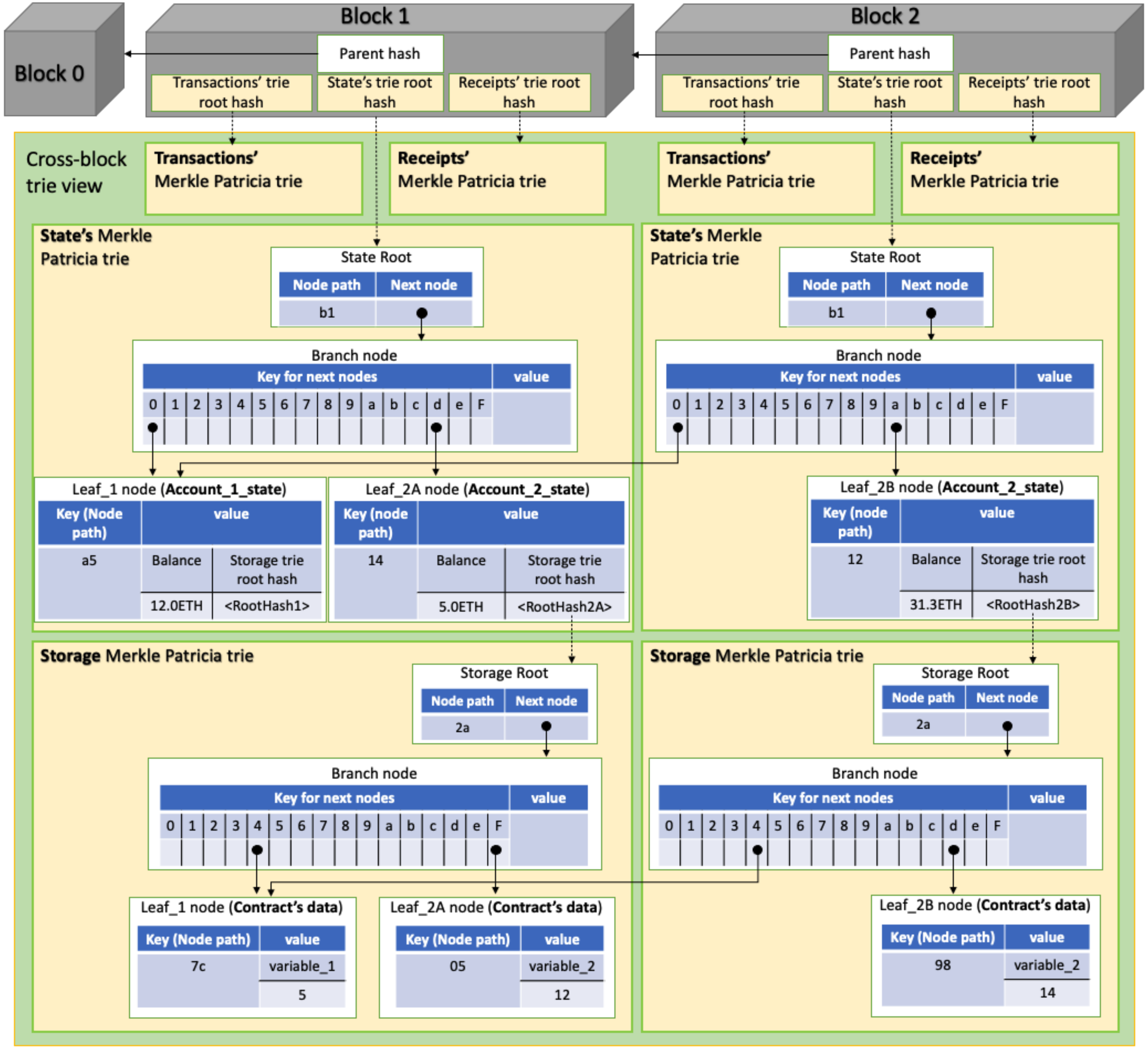
Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
				●									●			

Leaf_2B node (**Contract's data**)

Key (Node path)	value
98	variable_2
	14

Ethereum

State trie



Block 0

Block 1

Parent hash

Transactions' trie root hash

State's trie root hash

Receipts' trie root hash

Block 2

Parent hash

Transactions' trie root hash

State's trie root hash

Receipts' trie root hash

Cross-block trie view

Transactions' Merkle Patricia trie

Receipts' Merkle Patricia trie

Transactions' Merkle Patricia trie

Receipts' Merkle Patricia trie

State's Merkle Patricia trie

State Root

Node path	Next node
b1	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
●													●			

Leaf_1 node (**Account_1_state**)

Key (Node path)	value	
a5	Balance	Storage trie root hash
	12.0ETH	<RootHash1>

Leaf_2A node (**Account_2_state**)

Key (node path)	value	
14	Balance	Storage trie root hash
	5.0ETH	<RootHash2A>

State's Merkle Patricia trie

State Root

Node path	Next node
b1	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
●										●						

Leaf_2B node (**Account_2_state**)

Key (node path)	value	
12	Balance	Storage trie root hash
	31.3ETH	<RootHash2B>

Storage Merkle Patricia trie

Storage Root

Node path	Next node
2a	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
				●											●	

Leaf_1 node (**Contract's data**)

Key (Node path)	value
7c	variable_1
	5

Leaf_2A node (**Contract's data**)

Key (Node path)	value
05	variable_2
	12

Storage Merkle Patricia trie

Storage Root

Node path	Next node
2a	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
				●									●			

Leaf_2B node (**Contract's data**)

Key (Node path)	value
98	variable_2
	14

Ethereum

Read contract state

1. ask trusted node
2. receive inclusion proof for

   stateRoot: storageTrie
   account state: stateTrie

   and block header

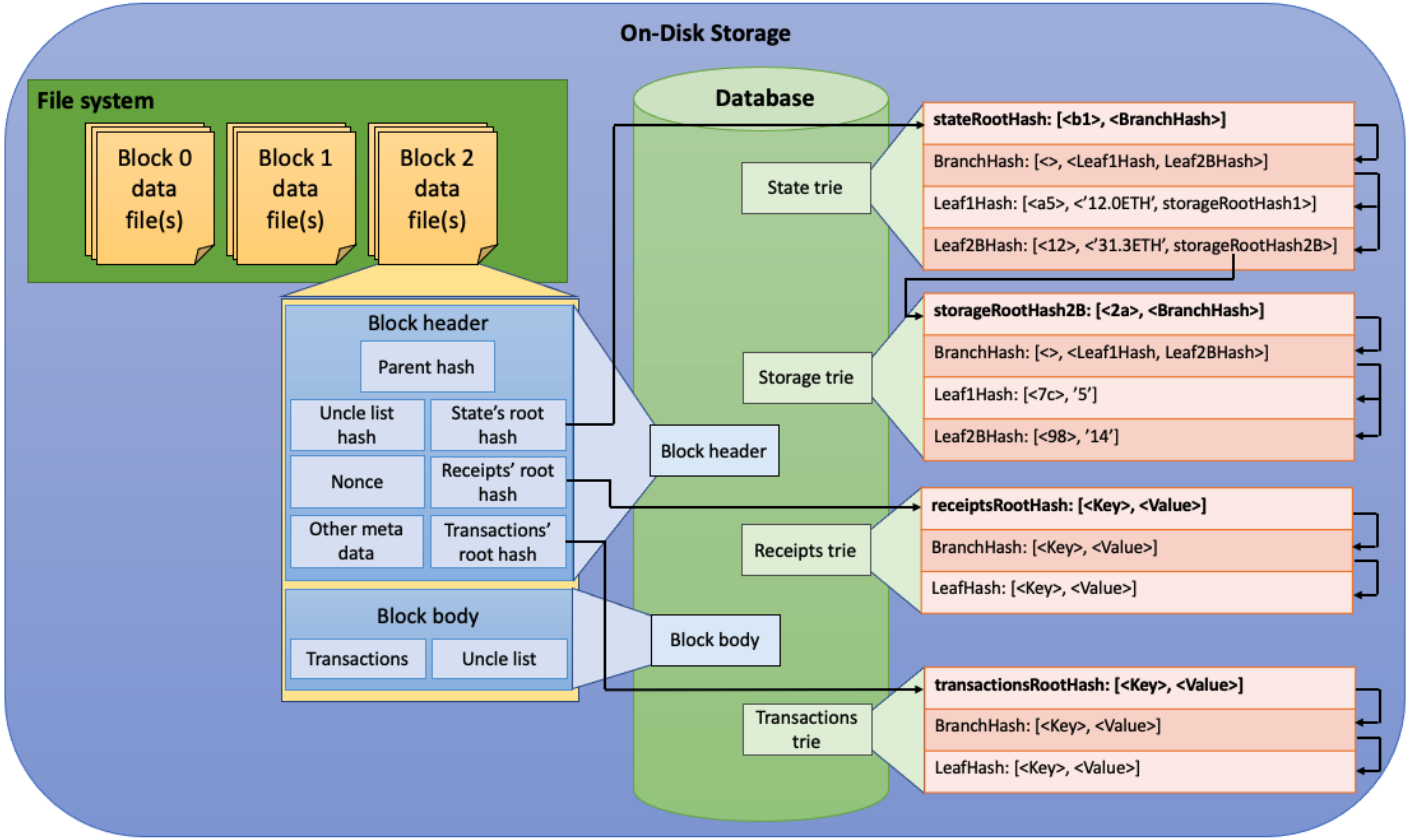
Ethereum

Receipts trie

Stores transaction results:

```
From:  address
To:    address
Status: … // aborted?
Logs: events
ContractAddress address
  // new contract address,
  // if created
```

Return transaction results,
by emitting Events,
which are added to the `logs`.



On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Block header
Parent hash
Uncle list hash
State's root hash
Nonce
Receipts' root hash
Other meta data
Transactions' root hash
Block body
Transactions
Uncle list
Database
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <'12.0ETH', storageRootHash1>]
Leaf2BHash: [<12>, <'31.3ETH', storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, '5']
Leaf2BHash: [<98>, '14']
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]