Lisk Interoperability

Cross-chain interactions between homogeneous state machines

By Alessandro Ricottone

Lisk

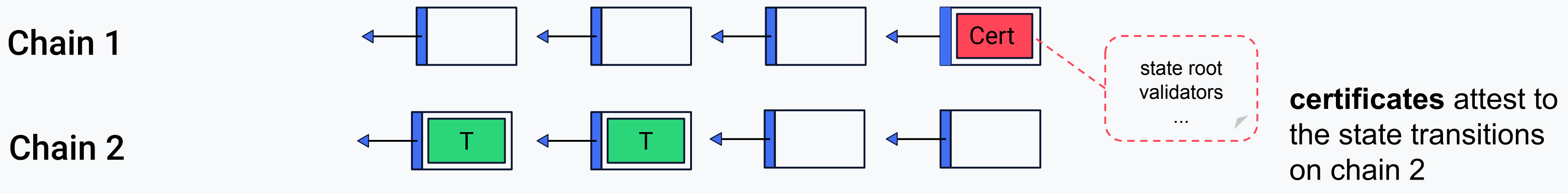
Outline

- **Cross-chain certification**
  - General overview and comparison to industry
- **The Lisk ecosystem**
  - Connecting to the ecosystem
  - State model of a Lisk chain
- **Cross-chain updates**
- **Cross-chain messages**

10000 Feet View of Interoperability

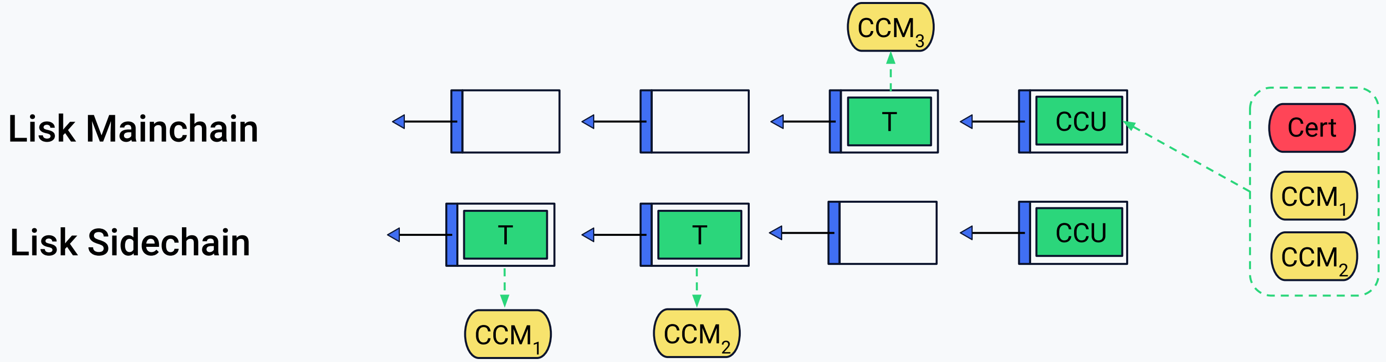
- Multi-shards architecture
  - all chains share same state which is partitioned among them
  - shards submit blocks to mainchain validators that verify all state transitions
  - the whole system reorganizes together
  - example: Polkadot, Ethereum
- Independent-chains architecture
  - each chain has a separate state and state transition functions
  - chains have an independent set of validators that verify and finalize blocks
  - each chain can reorganize independently
  - example: Cosmos, Lisk

Cross-chain Certification



- Chain 1 trusts information from chain 2 because of **certificates**
  - sent by relayers
  - authenticate **state** of chain 2
  - contain **signature** of active validators
  - update about **future validators**

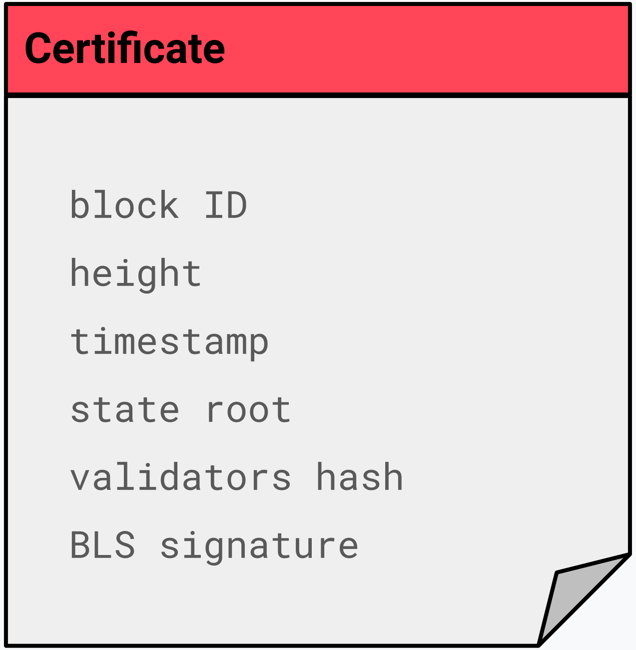
Lisk Cross-chain Certification



cross-chain updates contain certificates and **cross-chain messages**

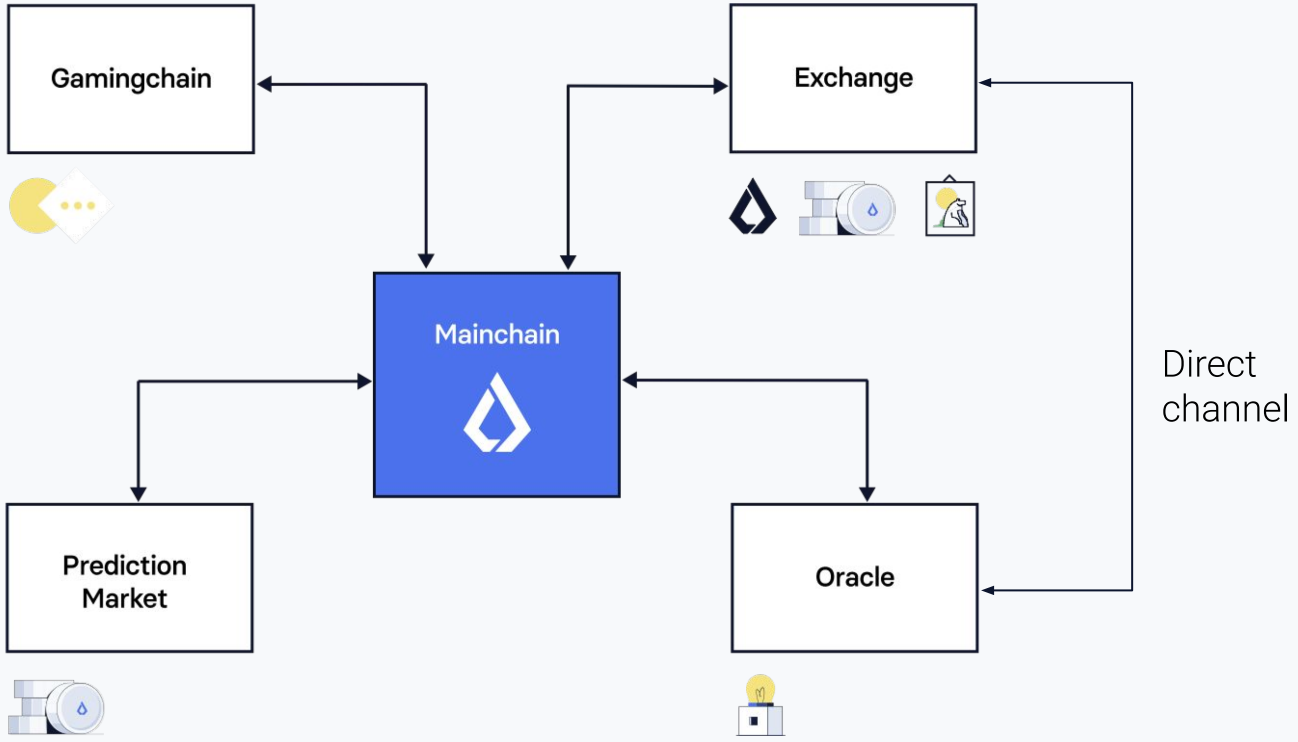
- Mainchain trusts information from sidechain because of **certificates**
    - created from block headers
    - certificates are included in **cross-chain updates** together with **cross-chain messages**
    - contain **BLS signature** of active validators
    - this interaction is symmetric

Cross-chain Certificates

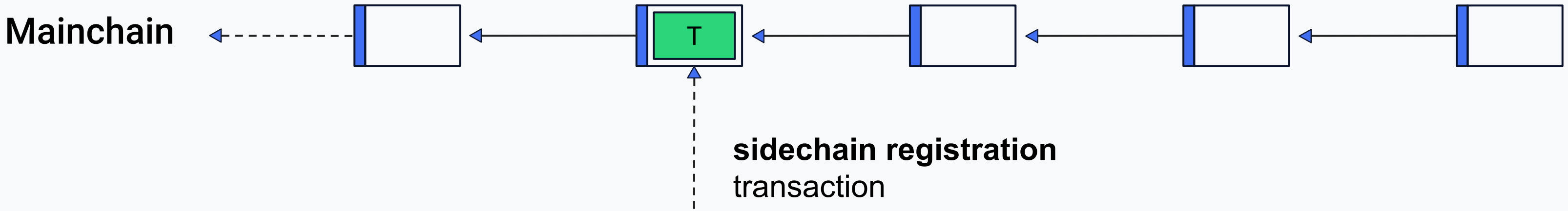


- **Block ID**: identify block header
- **Height**: used to validate certificate
- **Timestamp**: used to validate certificate
- **State root**: authenticates state of sending chain
- **Validators hash**: authenticates set of future validators
- **BLS signature**: aggregated signature of current validators

Lisk Ecosystem



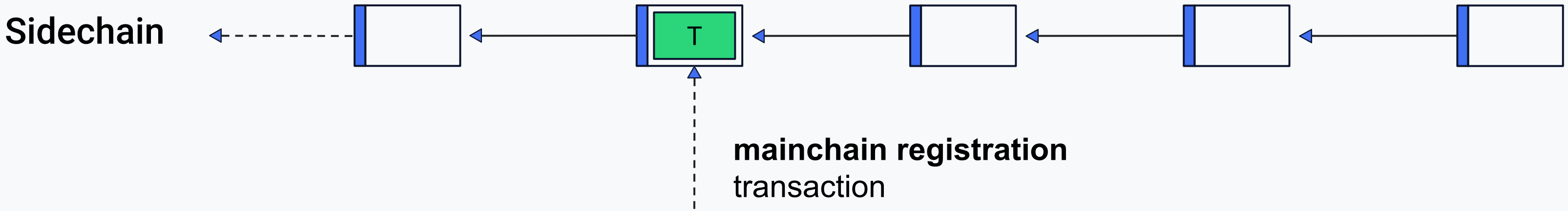
Sidechain Registration



All sidechains register on the mainchain

- **Name** of the sidechain
- **ID of the genesis block** of the sidechain
  - Used to compute the **sidechain network ID**
- **Sidechain validators** to sign first cross-chain update from sidechain

Mainchain Registration



Afterwards mainchain registers on the sidechain

- The sidechain own **chain ID** and **name** computed during the sidechain registration
- **Mainchain validators** to sign first cross-chain update from mainchain

Registration Process

Chain account

chain ID

name

network ID

last certificate
- height
- timestamp
- state root
- validators hash

status

Validators

[**active validators**]
- bls key
- bft weight

certificate threshold

Channel

inbox
- root
- size
- append path

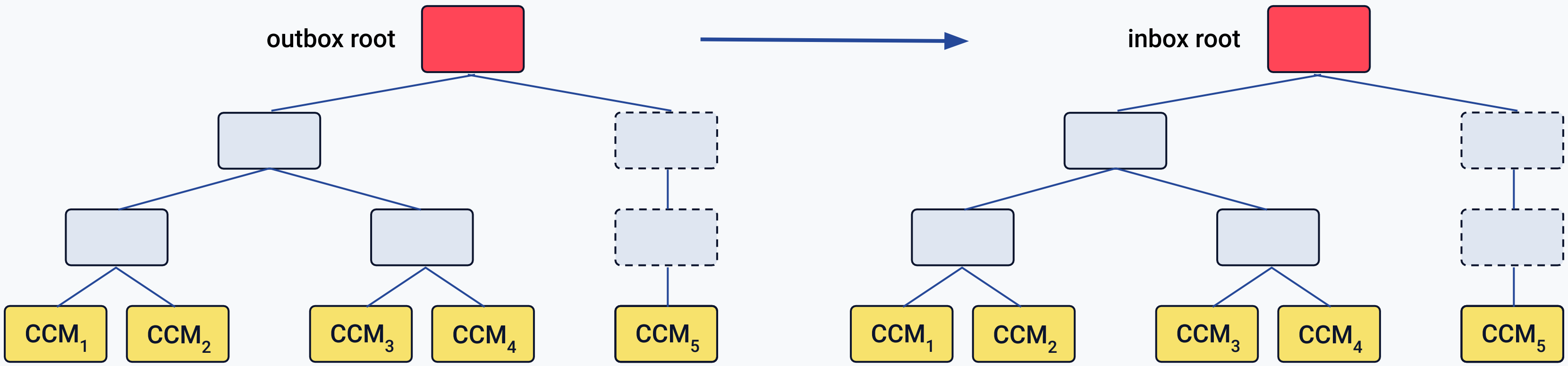
outbox
- root (also stored separately)
- size
- append path

partner chain outbox root

message fee token ID

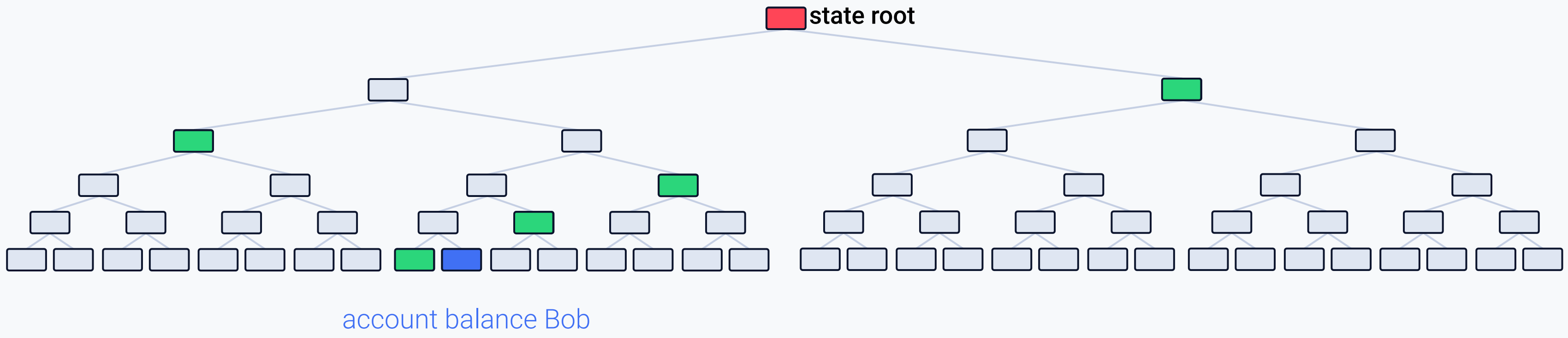
Inbox and Outbox

- **Inbox** and **outbox** are append-only Merkle trees
- Cross-chain messages sent to a chain are appended to the outbox
- The same message is appended to corresponding inbox when received



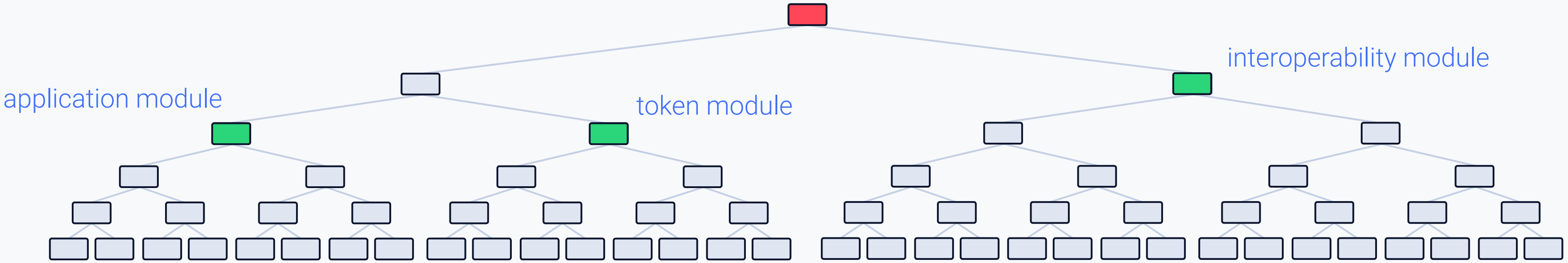
State Tree

- Every chain uses a sparse Merkle tree for its state, called **state tree**
- The root of this tree is called **state root**
- SMTs allow to prove efficiently that an entry has a certain value



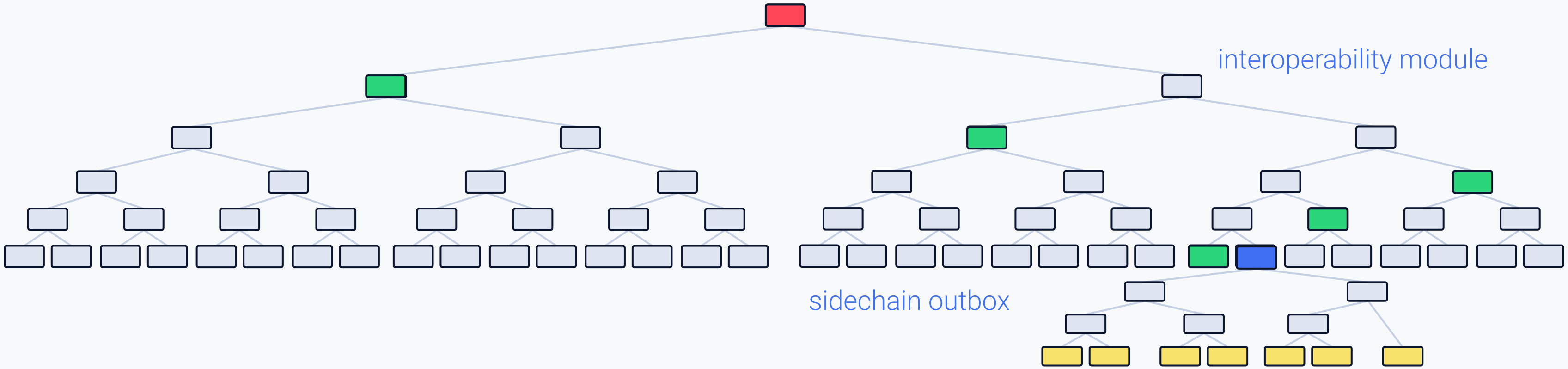
State Tree

- The state tree is split into several **module subtrees**
- Each module defines generic key-value map
- Application modules implement custom logic



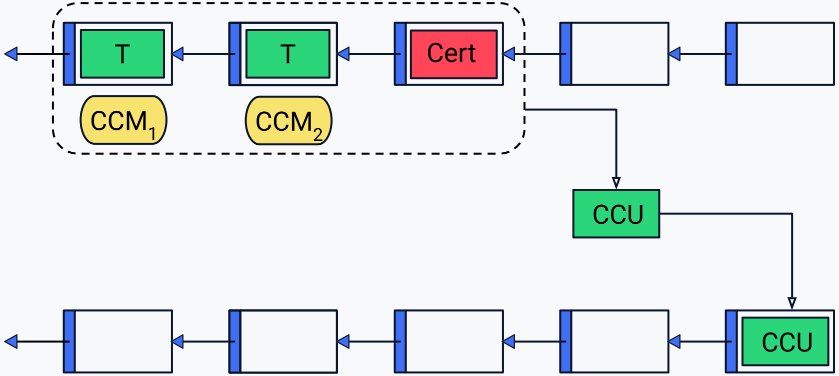
State Tree

- Interoperability module contains the channel state
- Cross-chain messages are validated from the outbox root to the state root



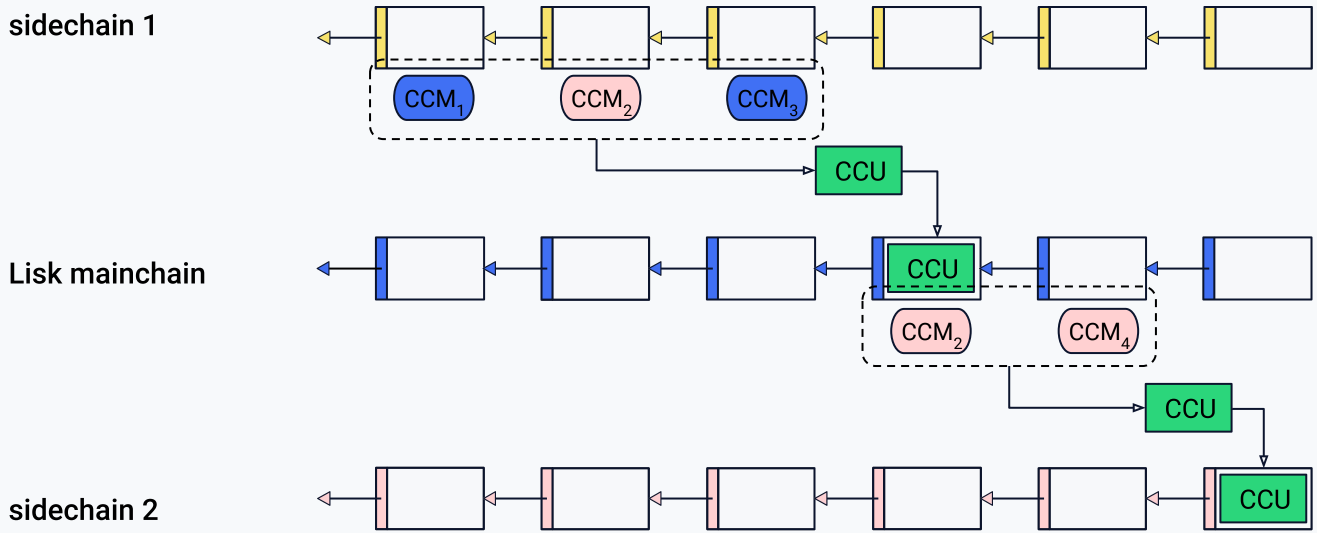
Cross-chain Updates

- Transactions trigger CCMs
- Validators create certificates from finalized blocks
- Relayers collect certificates and CCMs in a CCU
- Relayers post the CCU on the receiving chain
- CCMs are executed directly on receiving chain



Mainchain Routing

- A sidechain will exchange CCUs with the Lisk mainchain
- CCMs targeting other sidechains will be forwarded by the mainchain



Cross-chain Messages

Cross-chain Message

```
nonce

module ID
asset ID

fee

params

sending chain ID
receiving chain ID

status
```

- **Nonce**: identifies CCM in the ecosystem
- **Module ID and asset ID**: identifies transaction
- **Params**: contains transaction parameters
- **Fee**: paid to the relayer
- **Sending chain ID**: identifies sending chain
- **Receiving chain ID**: identifies receiving chain
- **Status**: extra information about CCM

Cross-chain Token Params

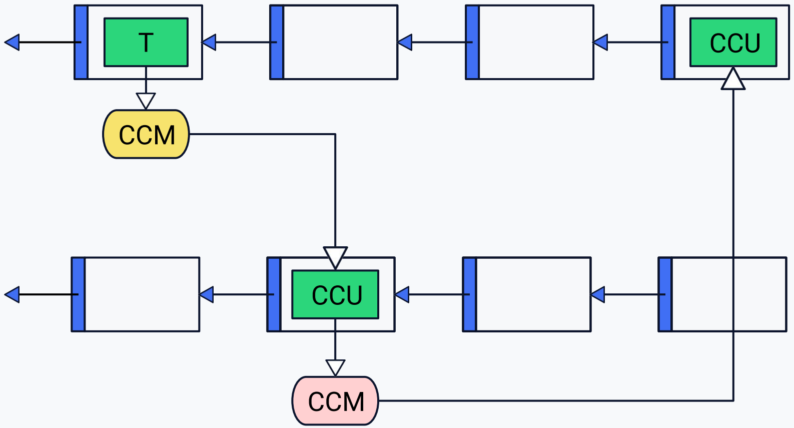
Cross-chain Transfer Params

```
amount
token ID
recipient address
sender address
data
```

- **Amount**: the amount of transferred tokens
- **Token ID**: used to identify the transferred tokens
- **Recipient address**: used for crediting the tokens
- **Sender address**: used for reimbursement if message fails
- **Data**: data field accompanying the transfer

CCM can Trigger a CCM

- Message is sent to receiving chain
- Processing message generates new message
- Example: Message triggers an error



Thank you!

LiskHQ

Lisk

LiskHQ