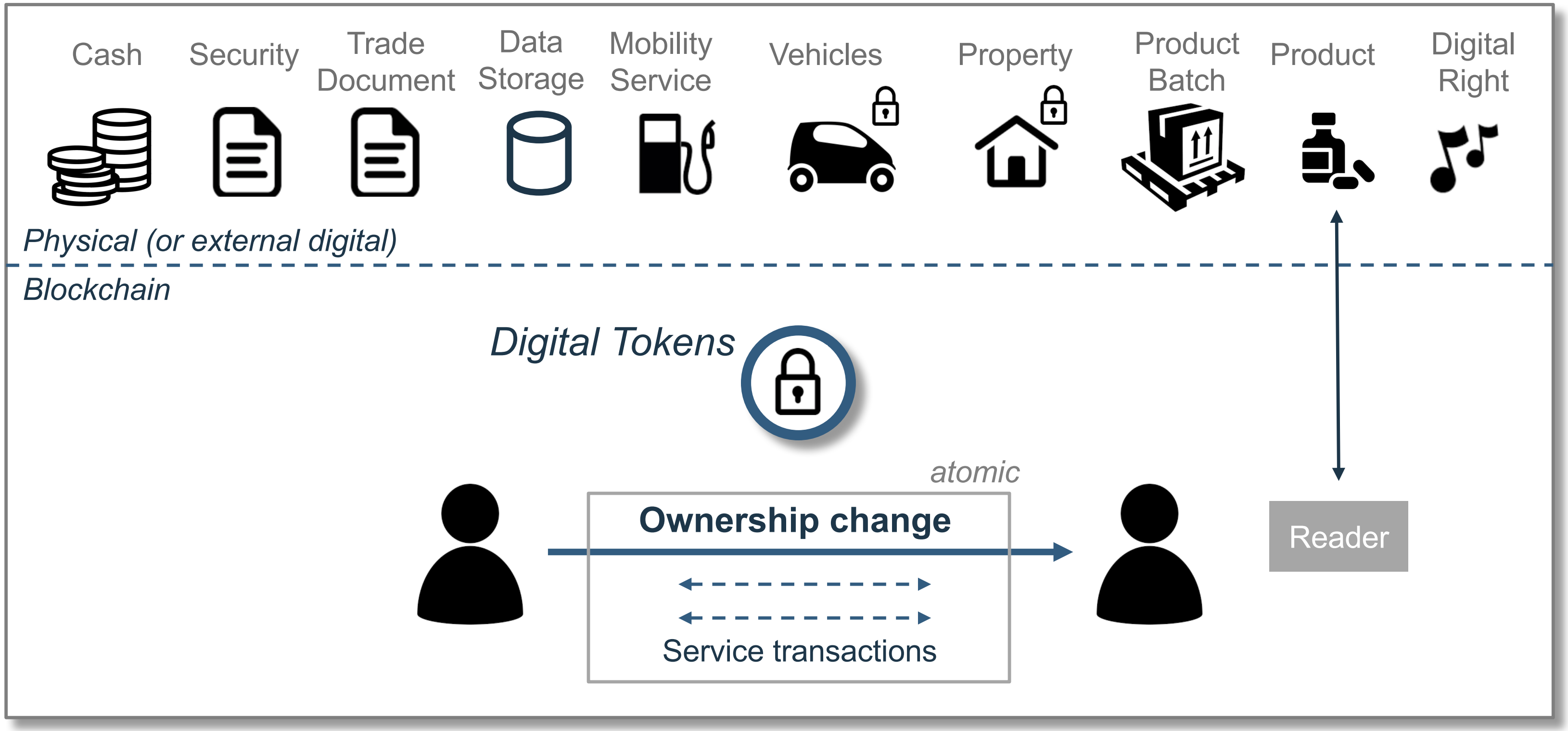
Token management in Fabric

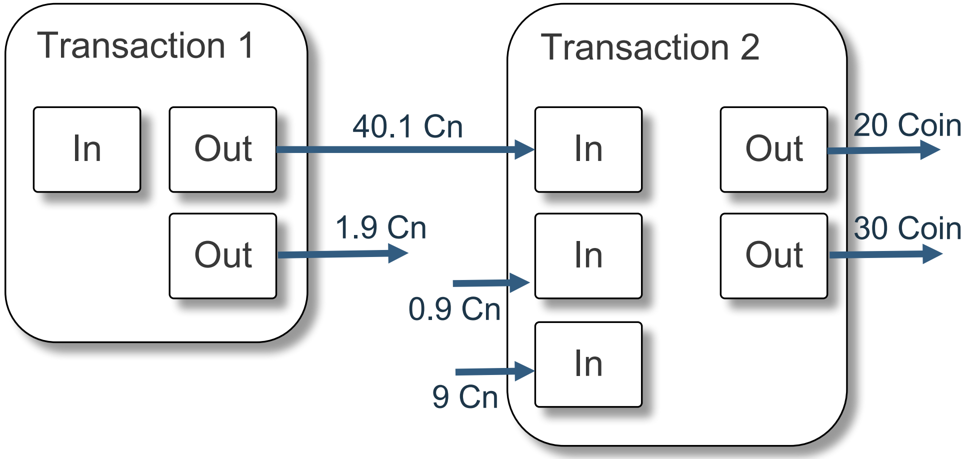
Assets can be conveniently represented with digital tokens



Use cases

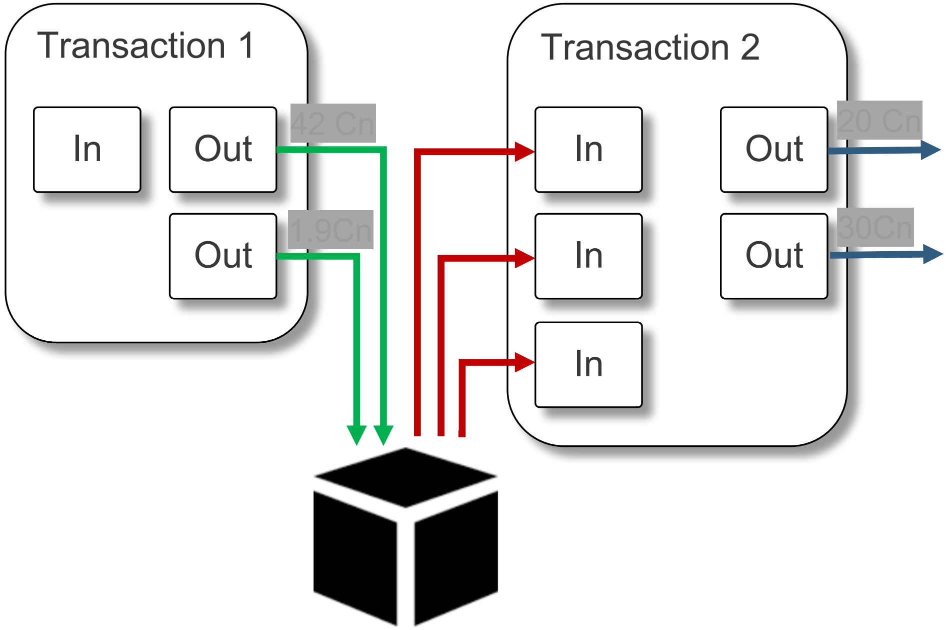
- Securities trading
- Asset transfer
- Digital currency
- Supply chain
- Provenance
- …

Unspent Transaction Output (UTXO) token ownership model



Standard case:

- In a transaction, the sum of the values of all the inputs must be greater or equal to the sum of the values of all outputs
- Only *unspent* outputs of previous transactions can be used as inputs to a new transaction
- With a new transaction, inputs are deleted and new outputs are created that may be consumed in future transactions



Privacy-preserving case:

- Inputs of a valid transaction make respective outputs in the UTXO pool cryptographically unspendable
- Correctness of payments cryptographically enforced

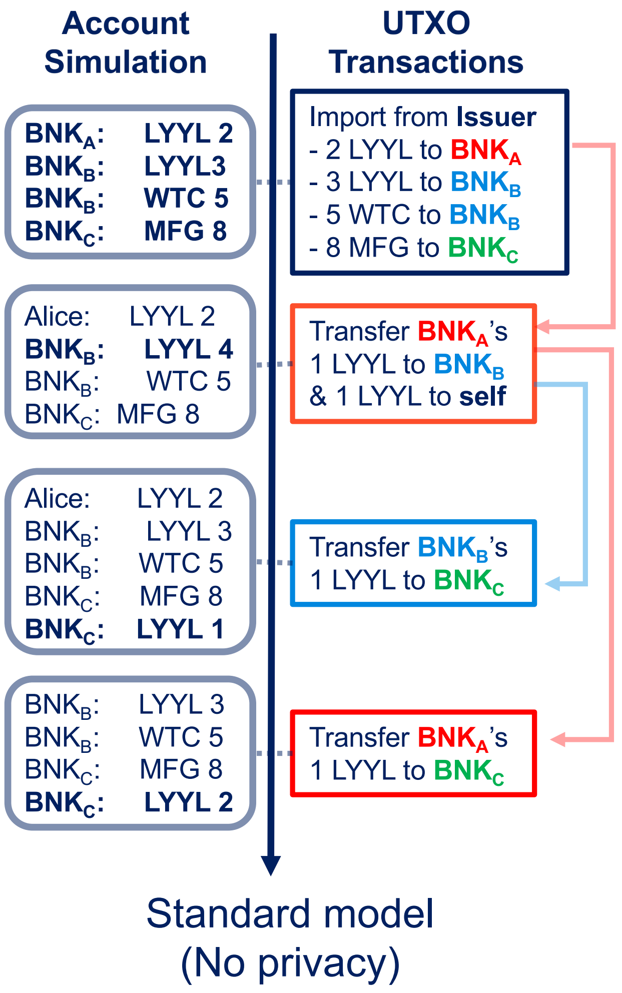
Account model for token ownership

- Single account per system user
- Transactions carry transfer requests, and validation updates corresponding user-accounts
- To some extend and on the client side, can be simulated on top of UTXO model

- Do not support privacy-preserving transactions -> conversion to UTXO is needed

- Support a variety of transfer extensions (e.g, `transferFrom`/`approve`)

Privacy is a key requirement in token management



Account Simulation

UTXO Transactions

BNK$_A$: LYYL 2
BNK$_B$: LYYL3
BNK$_B$: WTC 5
BNK$_C$: MFG 8

Import from **Issuer**
- 2 LYYL to **BNK$_A$**
- 3 LYYL to **BNK$_B$**
- 5 WTC to **BNK$_B$**
- 8 MFG to **BNK$_C$**

Alice: LYYL 2
BNK$_B$: LYYL 4
BNK$_B$: WTC 5
BNK$_C$: MFG 8

Transfer **BNK$_A$**'s 1 LYYL to **BNK$_B$** & 1 LYYL to **self**

Alice: LYYL 2
BNK$_B$: LYYL 3
BNK$_B$: WTC 5
BNK$_C$: MFG 8
BNK$_C$: LYYL 1

Transfer **BNK$_B$**'s 1 LYYL to **BNK$_C$**

BNK$_B$: LYYL 3
BNK$_B$: WTC 5
BNK$_C$: MFG 8
BNK$_C$: LYYL 2

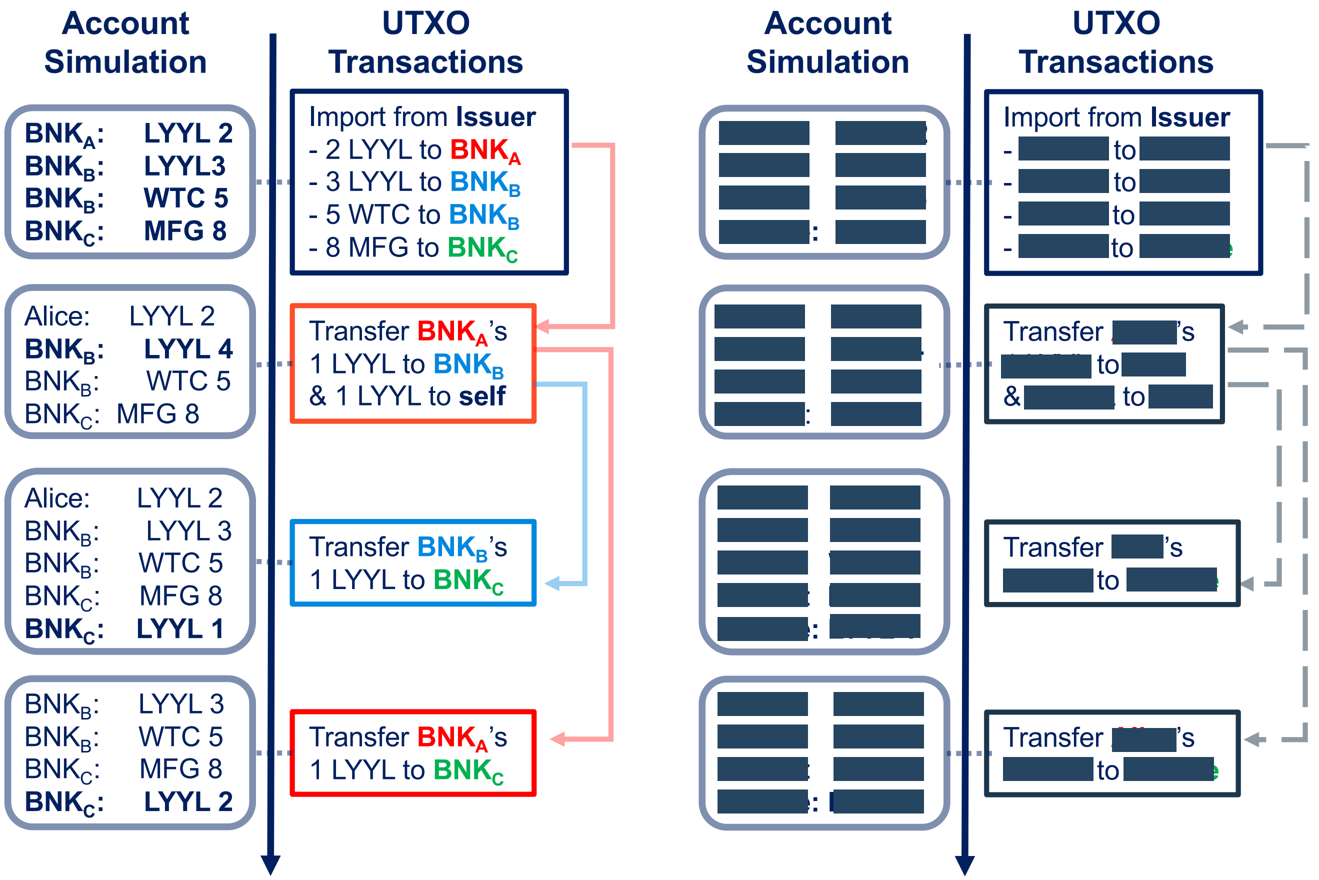
Transfer **BNK$_A$**'s 1 LYYL to **BNK$_C$**

Standard model
(No privacy)

Participants
- Bank A: **BNK$_A$**
- Bank B: **BNK$_B$**
- Bank C: **BNK$_C$**

Token units
- LYYL loyyal
- WTC Water Canary
- MFG MUFG

Privacy is a key requirement in token management



Account Simulation

UTXO Transactions

BNK$_A$: LYYL 2
BNK$_B$: LYYL3
BNK$_B$: WTC 5
BNK$_C$: MFG 8

Import from **Issuer**
- 2 LYYL to **BNK$_A$**
- 3 LYYL to **BNK$_B$**
- 5 WTC to **BNK$_B$**
- 8 MFG to **BNK$_C$**

Alice: LYYL 2
BNK$_B$: LYYL 4
BNK$_B$: WTC 5
BNK$_C$: MFG 8

Transfer **BNK$_A$**'s
1 LYYL to **BNK$_B$**
& 1 LYYL to **self**

Alice: LYYL 2
BNK$_B$: LYYL 3
BNK$_B$: WTC 5
BNK$_C$: MFG 8
BNK$_C$: LYYL 1

Transfer **BNK$_B$**'s
1 LYYL to **BNK$_C$**

BNK$_B$: LYYL 3
BNK$_B$: WTC 5
BNK$_C$: MFG 8
BNK$_C$: LYYL 2

Transfer **BNK$_A$**'s
1 LYYL to **BNK$_C$**

Account Simulation

UTXO Transactions

[redacted]

Import from **Issuer**
- [redacted] to [redacted]
- [redacted] to [redacted]
- [redacted] to [redacted]
- [redacted] to [redacted]

[redacted]

Transfer [redacted]'s
[redacted] to [redacted]
& [redacted] to [redacted]

[redacted]

Transfer [redacted]'s
[redacted] to [redacted]

[redacted]

Transfer [redacted]'s
[redacted] to [redacted]

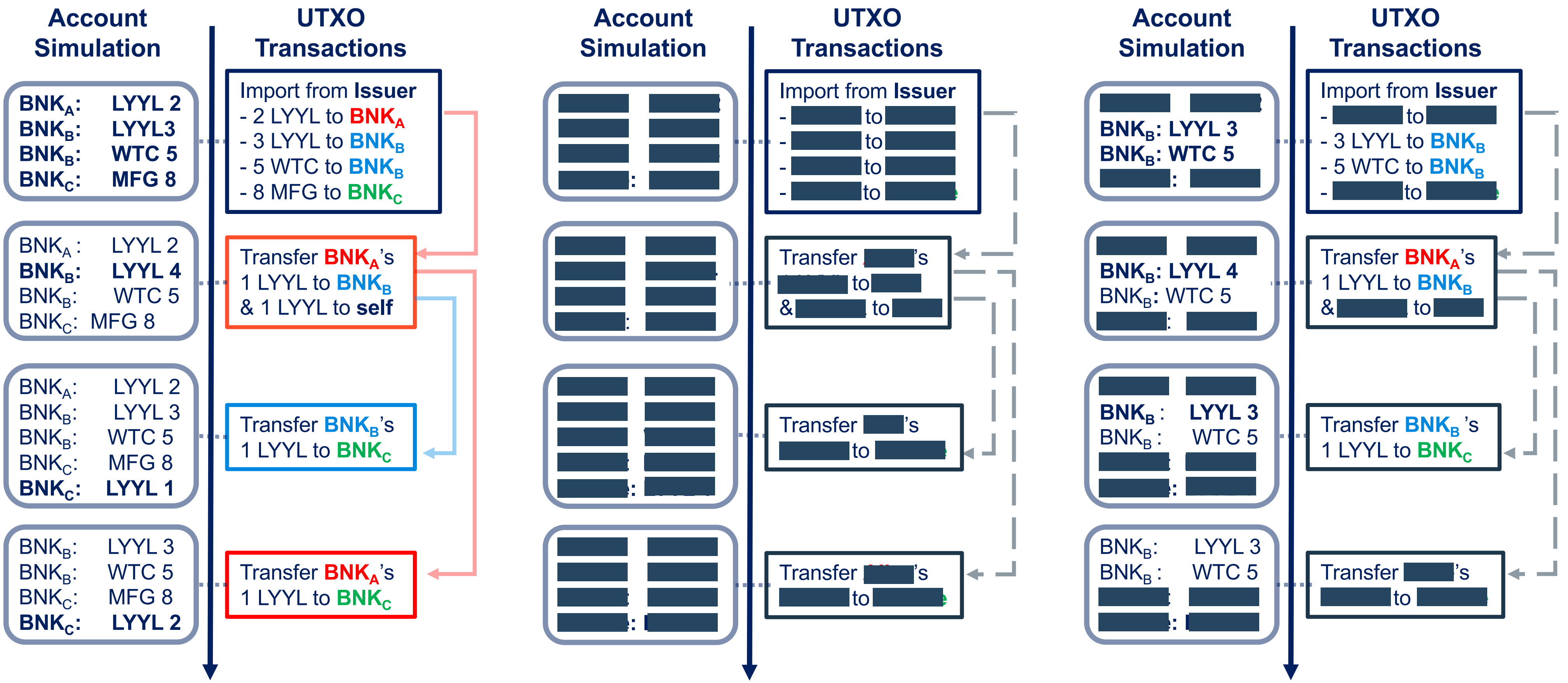
Standard model
(No privacy)

Hyperledger Fabric
(Identities, assets concealed)

Participants
- Bank A: **BNK$_A$**
- Bank B: **BNK$_B$**
- Bank C: **BNK$_C$**

Token units
- LYYL loyyal
- WTC Water Canary
- MFG MUFG

Privacy is a key requirement in token management



Account Simulation | UTXO Transactions

BNK_A: LYYL 2
BNK_B: LYYL3
BNK_B: WTC 5
BNK_C: MFG 8

Import from Issuer
- 2 LYYL to BNK_A
- 3 LYYL to BNK_B
- 5 WTC to BNK_B
- 8 MFG to BNK_C

BNK_A : LYYL 2
BNK_B: LYYL 4
BNK_B: WTC 5
BNK_C: MFG 8

Transfer BNK_A's 1 LYYL to BNK_B & 1 LYYL to self

BNK_A: LYYL 2
BNK_B: LYYL 3
BNK_B: WTC 5
BNK_C: MFG 8
BNK_C: LYYL 1

Transfer BNK_B's 1 LYYL to BNK_C

BNK_B: LYYL 3
BNK_B: WTC 5
BNK_C: MFG 8
BNK_C: LYYL 2

Transfer BNK_A's 1 LYYL to BNK_C

Standard model
(No privacy)

Account Simulation | UTXO Transactions

[redacted]

Import from Issuer
- [redacted] to [redacted]
- [redacted] to [redacted]
- [redacted] to [redacted]
- [redacted] to [redacted]

[redacted]

Transfer [redacted]'s [redacted] to [redacted] & [redacted] to [redacted]

[redacted]

Transfer [redacted]'s [redacted] to [redacted]

[redacted]

Transfer [redacted]'s [redacted] to [redacted]

Hyperledger Fabric
(Identities, assets concealed)

Account Simulation | UTXO Transactions

[redacted]
BNK_B: LYYL 3
BNK_B: WTC 5
[redacted]

Import from Issuer
- [redacted] to [redacted]
- 3 LYYL to BNK_B
- 5 WTC to BNK_B
- [redacted] to [redacted]

[redacted]
BNK_B: LYYL 4
BNK_B: WTC 5
[redacted]

Transfer BNK_A's 1 LYYL to BNK_B & [redacted] to [redacted]

[redacted]
BNK_B : LYYL 3
BNK_B : WTC 5
[redacted]

Transfer BNK_B 's 1 LYYL to BNK_C

BNK_B: LYYL 3
BNK_B : WTC 5
[redacted]

Transfer [redacted]'s [redacted] to [redacted]

Hyperledger Fabric
(e.g., view of auditor of BNK_B)

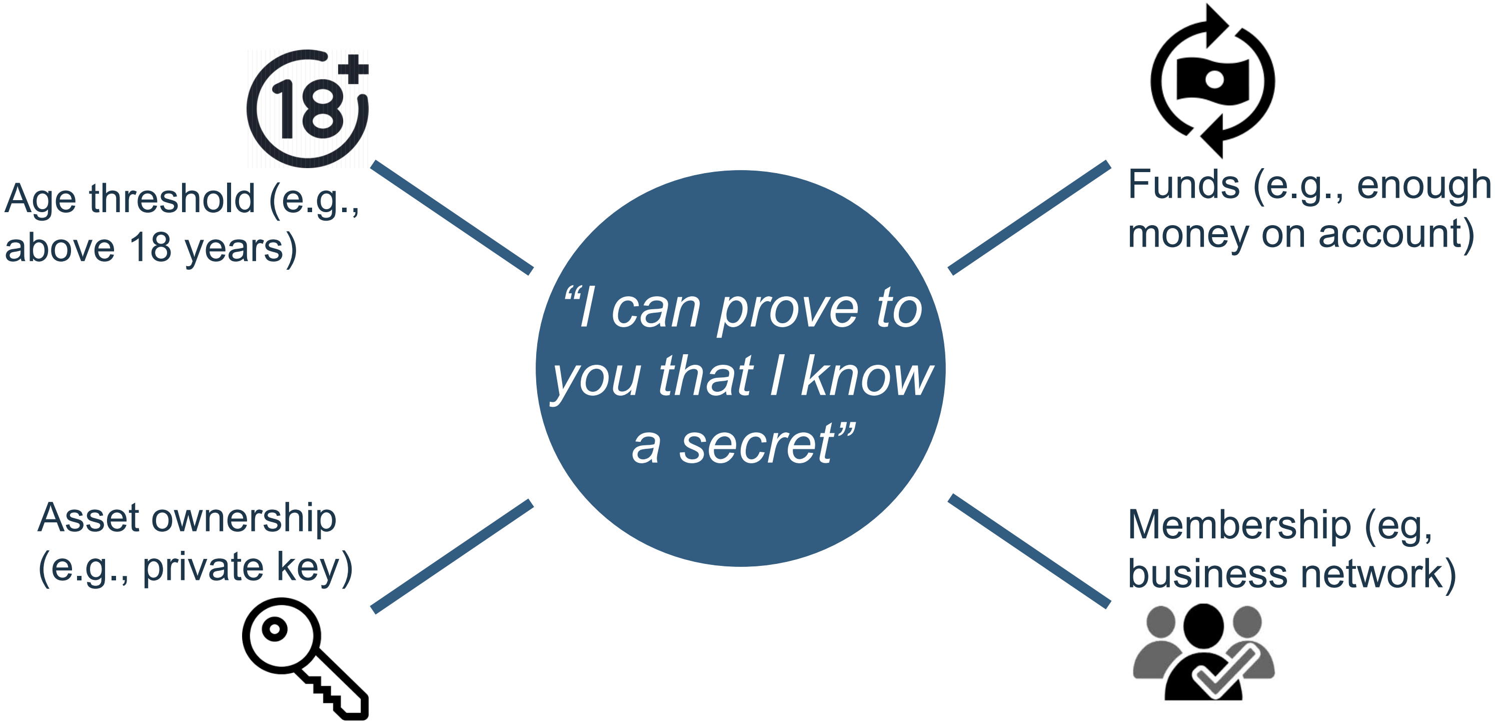
FabToken in a nutshell

- Fabric enablement for ***direct*** or ***as-a-service*** token management using ***UTXO***
- ***Modular*** architecture to accommodate a variety of implementations addressing different privacy, performance requirements & regulatory restrictions
- Compatible and ***integrate-able*** with other UTXO based token systems
- Easily ***extensible*** to support a variety of financial services operations

Zero-Knowledge Asset Transfer is a leading technology to privacy-preserving asset management on permissioned Blockchains

Strong identity management	- Users associated to long term identities that they cannot deny use of; provided by the Identity Mixer Technology
Accountability Non-repudiation	
Privacy	- User anonymity - Transferred token confidentiality (type, value)
Audit support	- On a per user-level: auditors bound to a user are guaranteed unconditional access to that user’s transaction details
Performance	- Lightweight (trusted) setup, easily decentralized - Lightweight transfer request computation
Technical Foundation	- Standard cryptographic assumptions

How to combine public verifiability with privacy? Using Zero-Knowledge (ZK) proofs!



“I can prove to you that I know a secret”

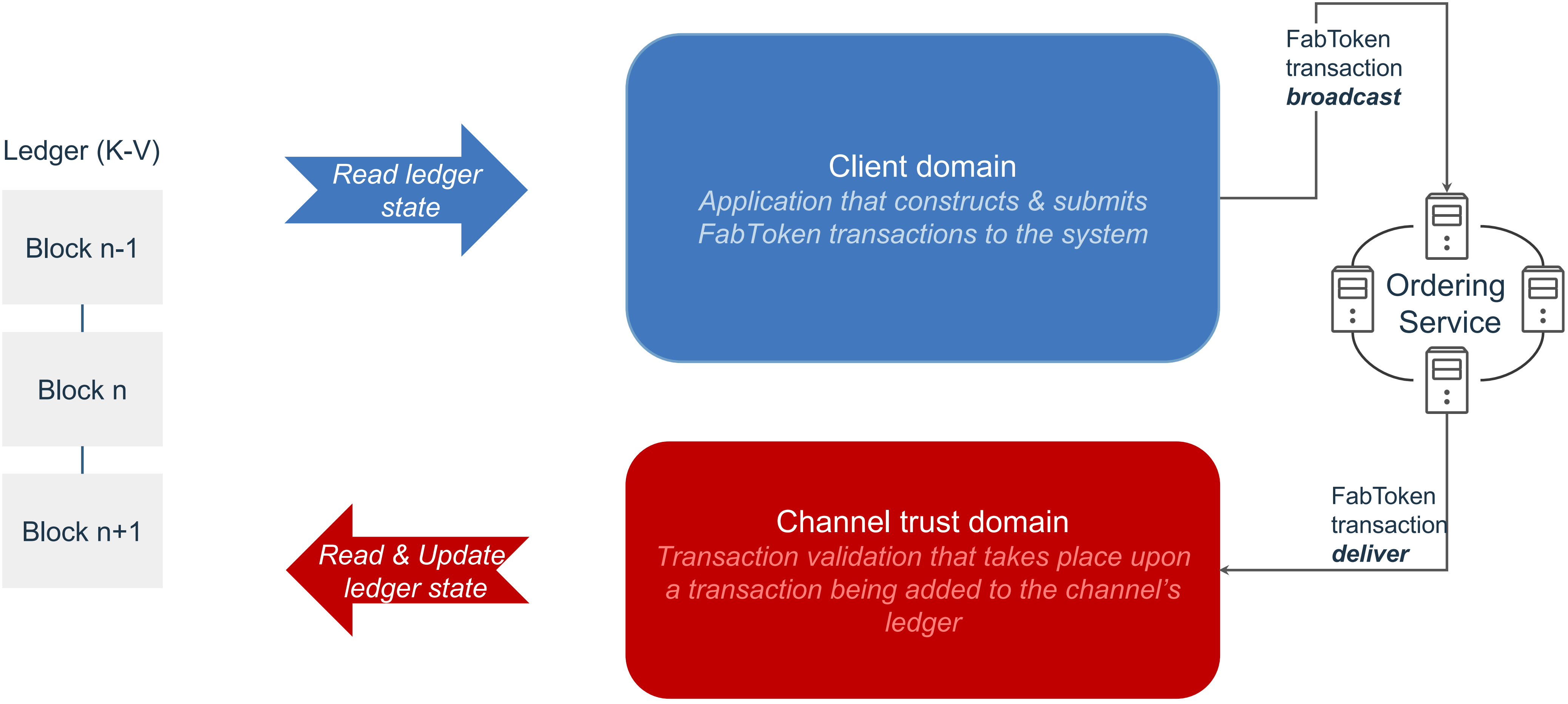
Age threshold (e.g., above 18 years)

Funds (e.g., enough money on account)

Asset ownership (e.g., private key)

Membership (eg, business network)

Token information flow in Fabric



Ledger (K-V)

Block n-1

Block n

Block n+1

Read ledger state

Client domain

Application that constructs & submits FabToken transactions to the system

FabToken transaction ***broadcast***

Ordering Service

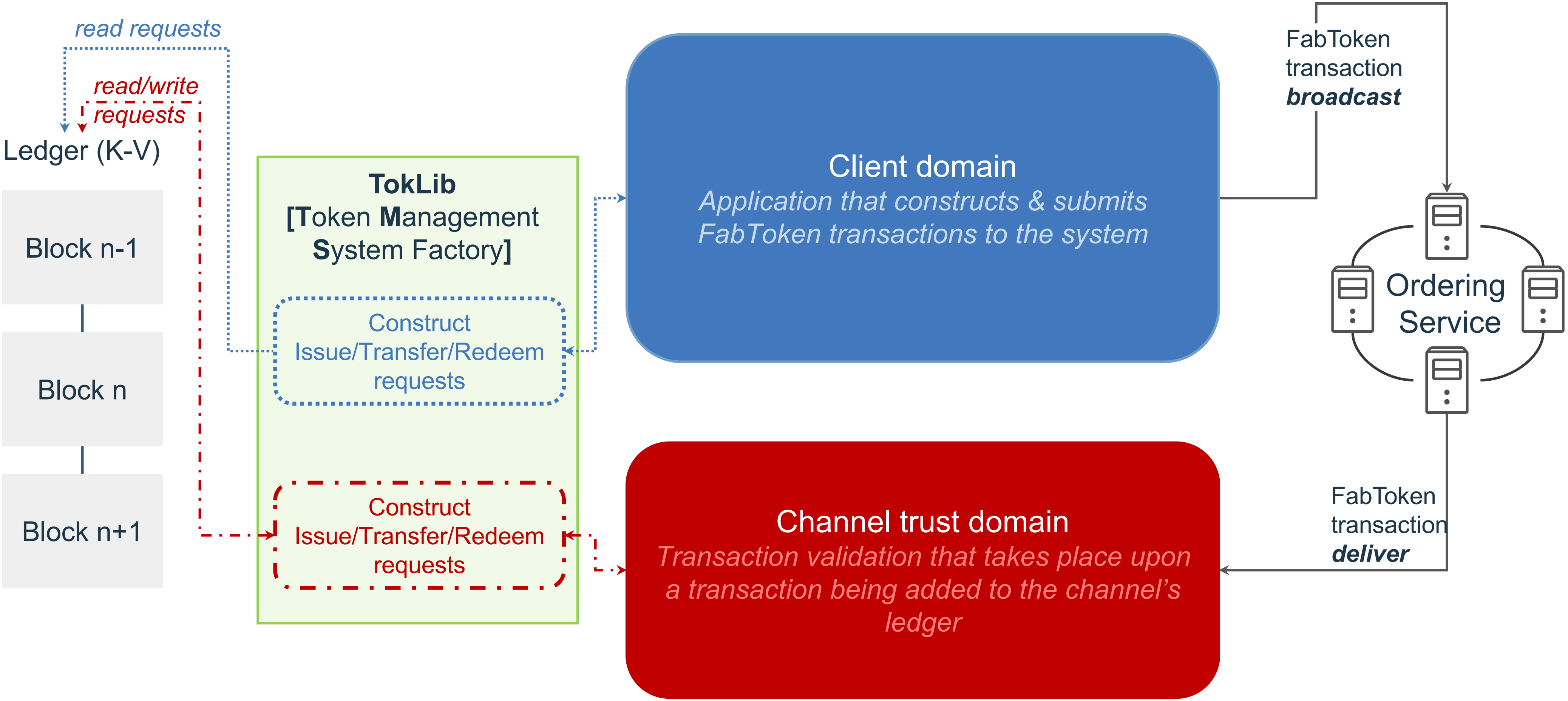
FabToken transaction ***deliver***

Channel trust domain

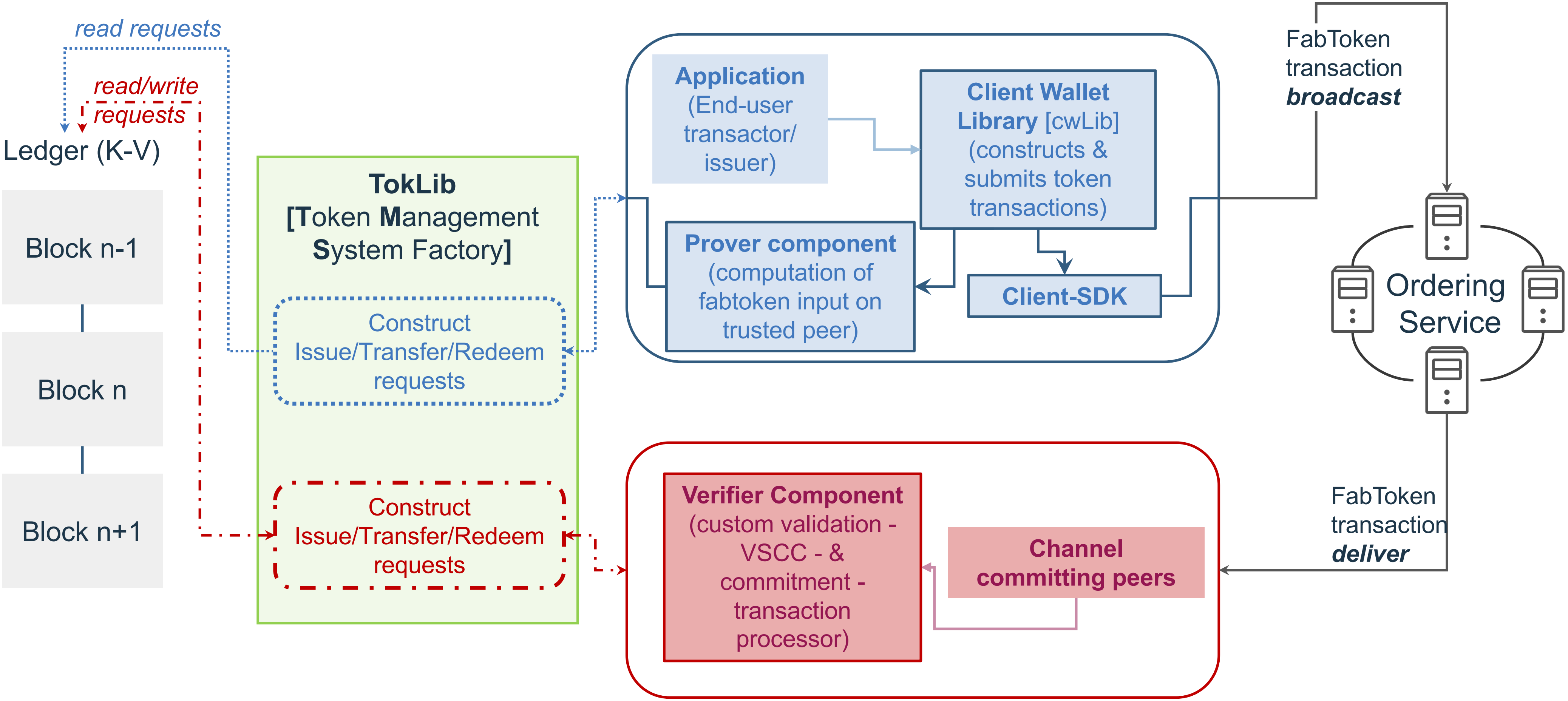
Transaction validation that takes place upon a transaction being added to the channel's ledger

Read & Update ledger state

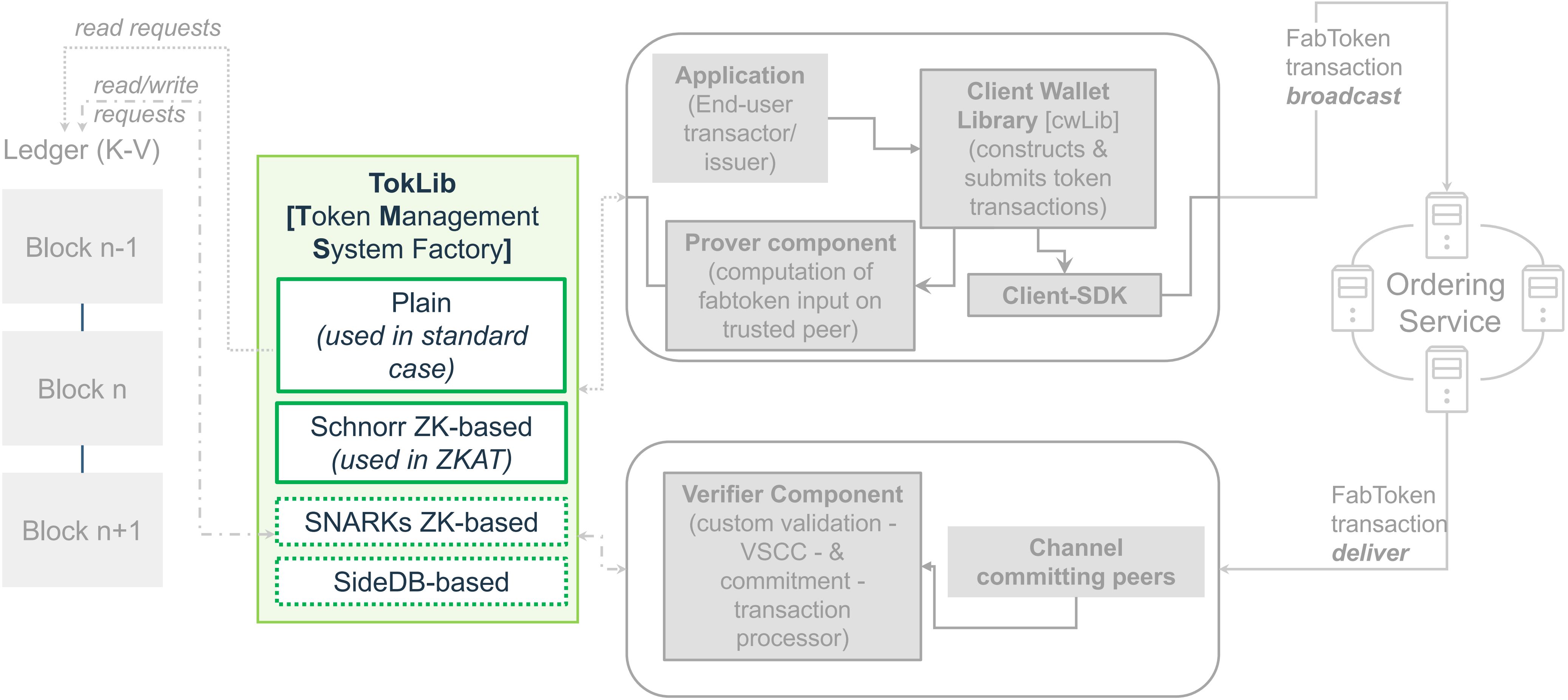
FabToken exhibits a modular architecture


read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib
[Token Management System Factory]
Construct Issue/Transfer/Redeem requests
Construct Issue/Transfer/Redeem requests
Client domain
Application that constructs & submits FabToken transactions to the system
Channel trust domain
Transaction validation that takes place upon a transaction being added to the channel's ledger
FabToken transaction broadcast
Ordering Service
FabToken transaction deliver

FabToken exhibits a modular architecture

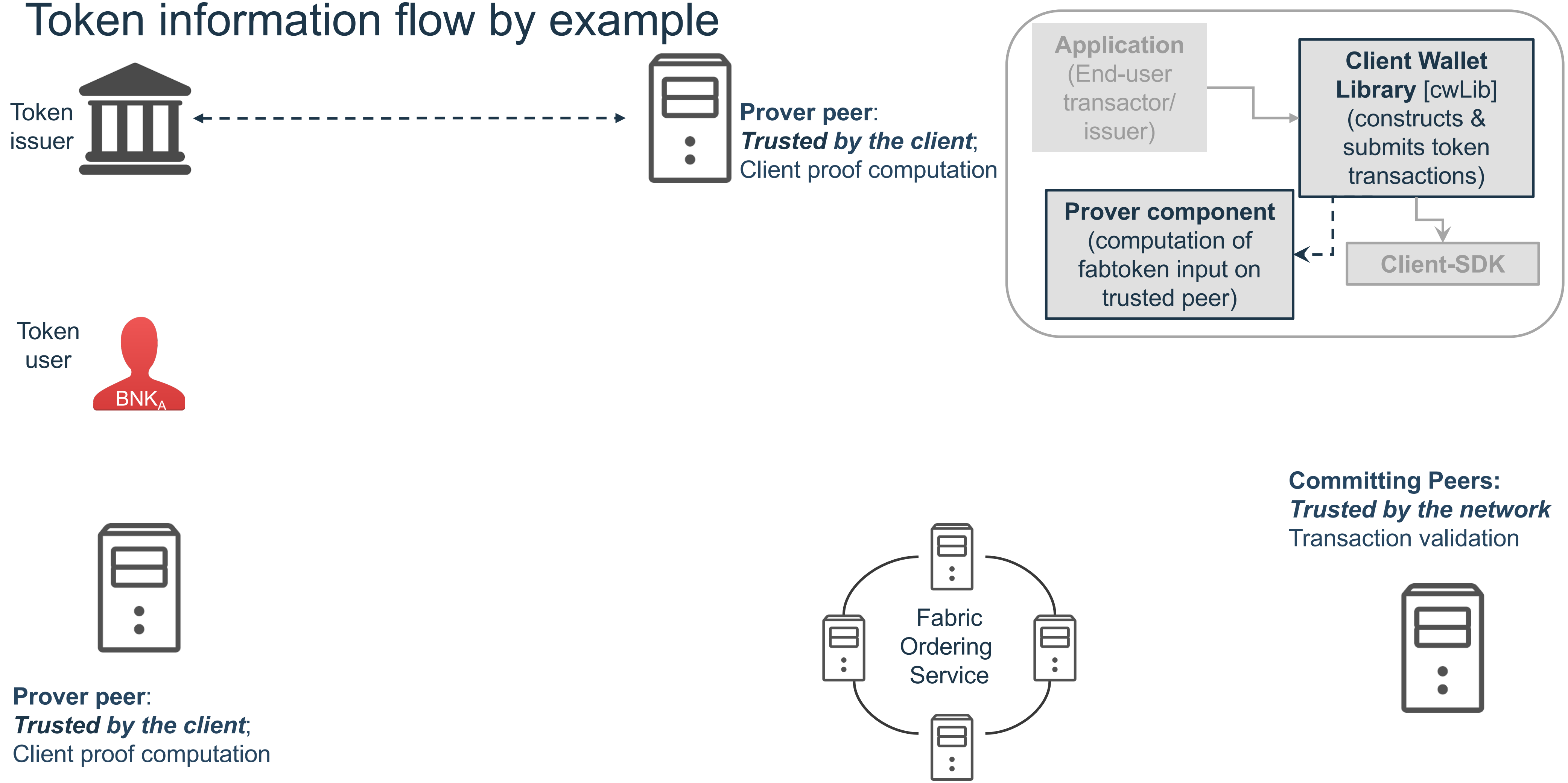

read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib
[Token Management System Factory]
Construct Issue/Transfer/Redeem requests
Construct Issue/Transfer/Redeem requests
Application (End-user transactor/ issuer)
Client Wallet Library [cwLib] (constructs & submits token transactions)
Prover component (computation of fabtoken input on trusted peer)
Client-SDK
FabToken transaction *broadcast*
Ordering Service
FabToken transaction *deliver*
Verifier Component (custom validation - VSCC - & commitment - transaction processor)
Channel committing peers

FabToken exhibits a modular architecture to accommodate various privacy levels



read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib
[Token **M**anagement **S**ystem Factory**]**
Plain
(used in standard case)
Schnorr ZK-based
(used in ZKAT)
SNARKs ZK-based
SideDB-based
Application
(End-user transactor/ issuer)
Client Wallet Library [cwLib]
(constructs & submits token transactions)
Prover component
(computation of fabtoken input on trusted peer)
Client-SDK
Verifier Component
(custom validation - VSCC - & commitment - transaction processor)
Channel committing peers
FabToken transaction ***broadcast***
Ordering Service
FabToken transaction ***deliver***

Token information flow by example



Token issuer

Prover peer:
Trusted by the client;
Client proof computation

Application (End-user transactor/ issuer)

Client Wallet Library [cwLib] (constructs & submits token transactions)

Prover component (computation of fabtoken input on trusted peer)

Client-SDK

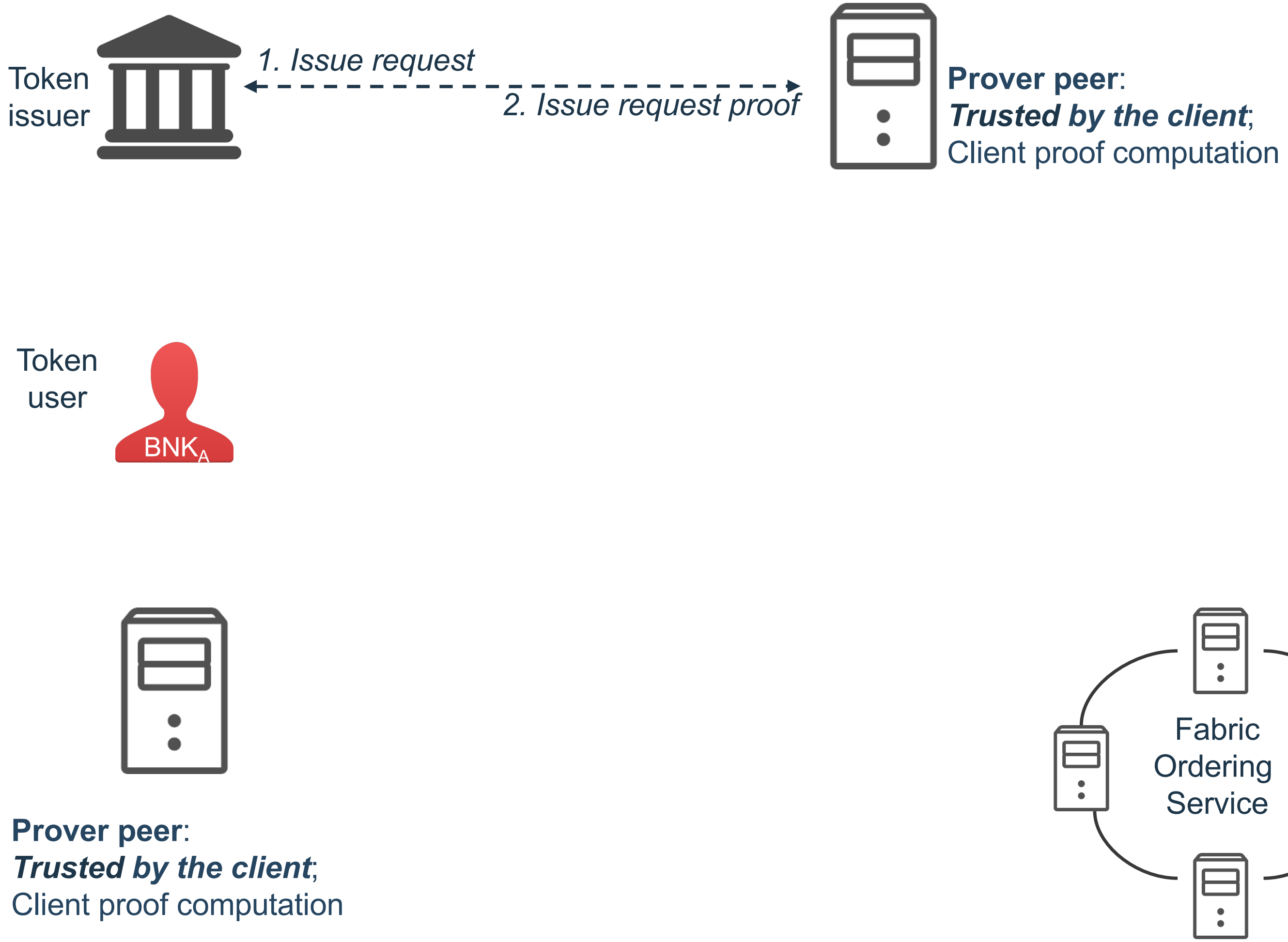
Token user

Committing Peers:
Trusted by the network
Transaction validation

Fabric Ordering Service

Prover peer:
Trusted by the client;
Client proof computation

Token information flow by example



Token issuer

1. Issue request

2. Issue request proof

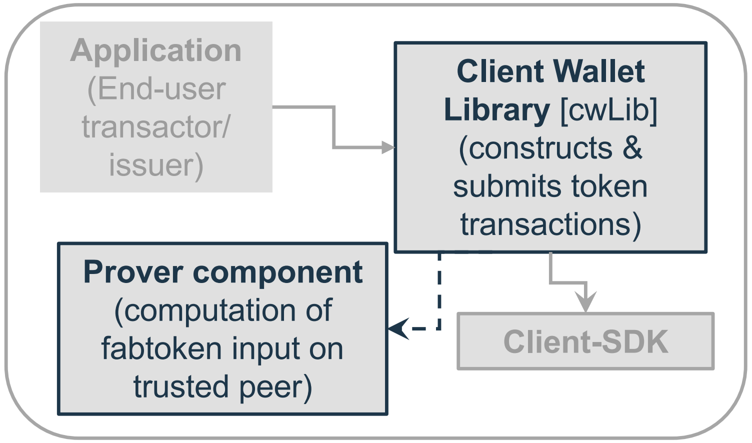
Prover peer:
Trusted by the client;
Client proof computation

Token user

BNK$_A$

Prover peer:
Trusted by the client;
Client proof computation

Fabric Ordering Service

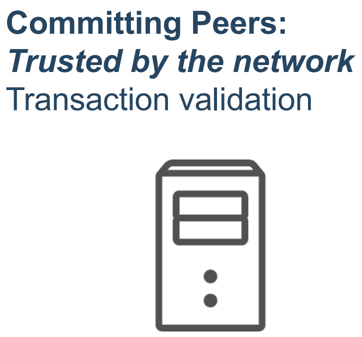


Application (End-user transactor/ issuer)

Client Wallet Library [cwLib] (constructs & submits token transactions)

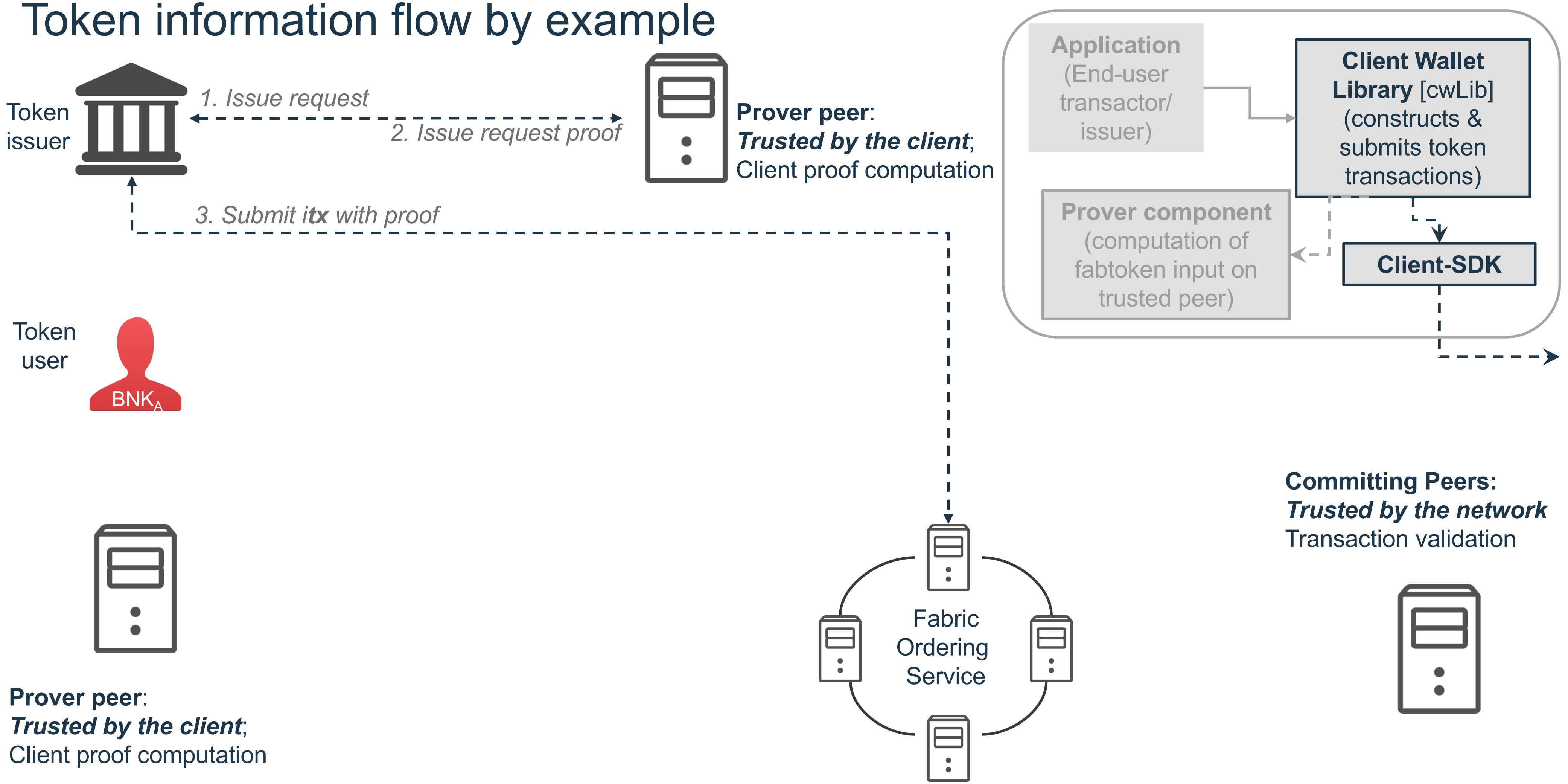
Prover component (computation of fabtoken input on trusted peer)

Client-SDK



Committing Peers:
Trusted by the network
Transaction validation

Token information flow by example



Token issuer

1. Issue request

2. Issue request proof

Prover peer:
Trusted by the client;
Client proof computation

*3. Submit i**tx** with proof*

Token user

BNK$_A$

Prover peer:
Trusted by the client;
Client proof computation

Fabric Ordering Service

Committing Peers:
Trusted by the network
Transaction validation

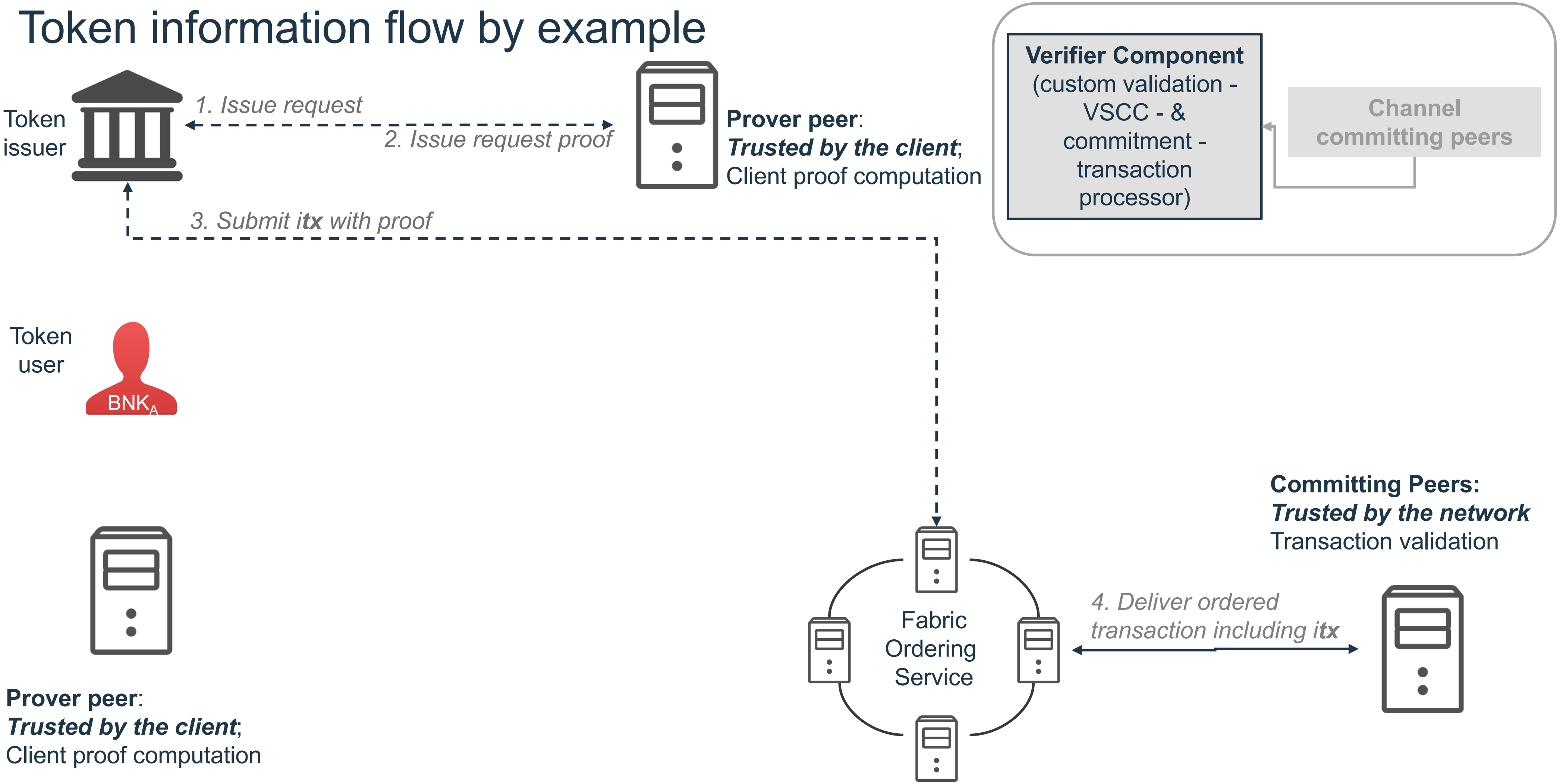
Application (End-user transactor/ issuer)

Client Wallet Library [cwLib] (constructs & submits token transactions)

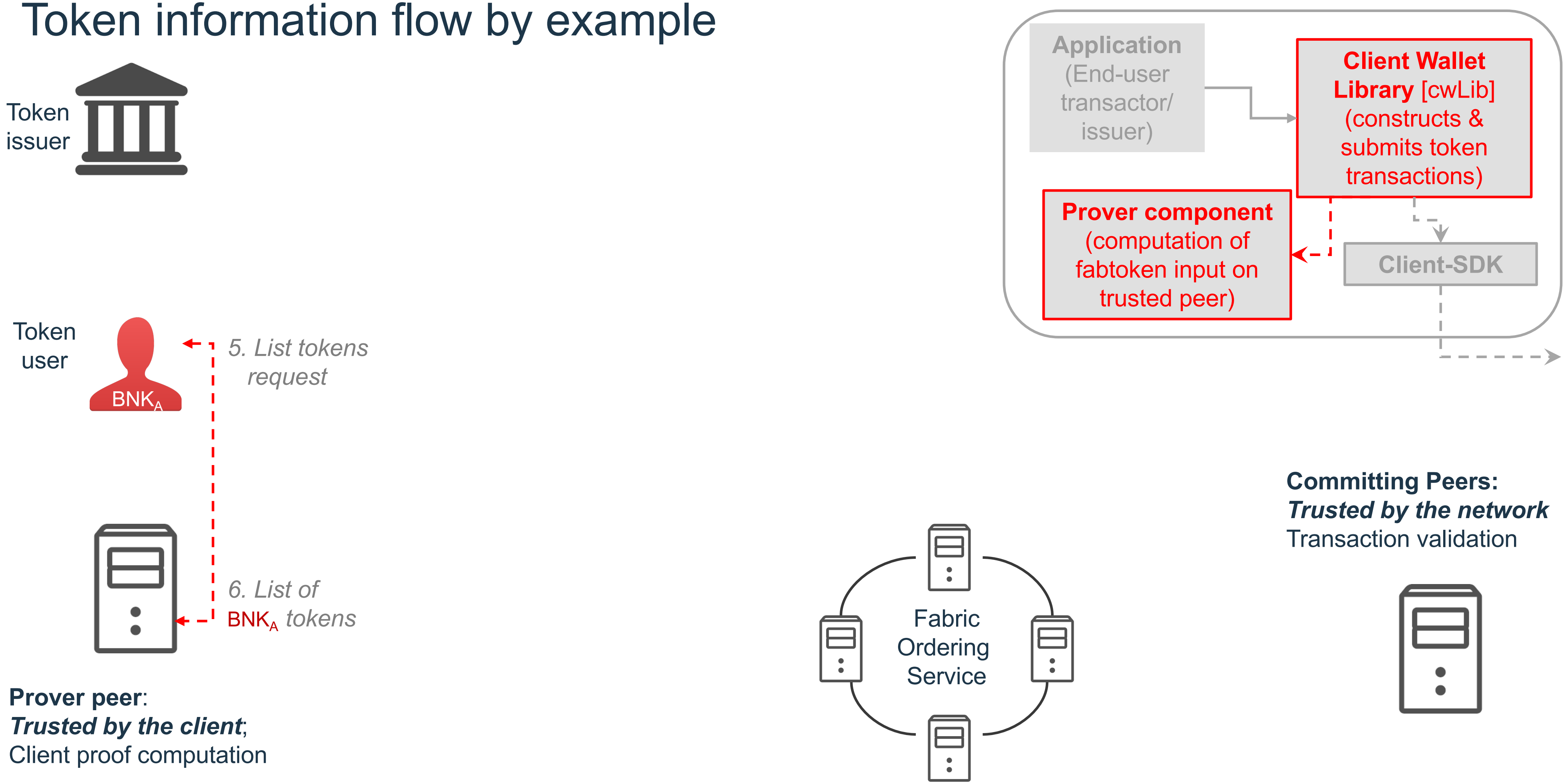
Prover component (computation of fabtoken input on trusted peer)

Client-SDK

Token information flow by example



Token information flow by example



Token issuer

Application (End-user transactor/ issuer)

Client Wallet Library [cwLib] (constructs & submits token transactions)

Prover component (computation of fabtoken input on trusted peer)

Client-SDK

Token user

BNK$_A$

5. List tokens request

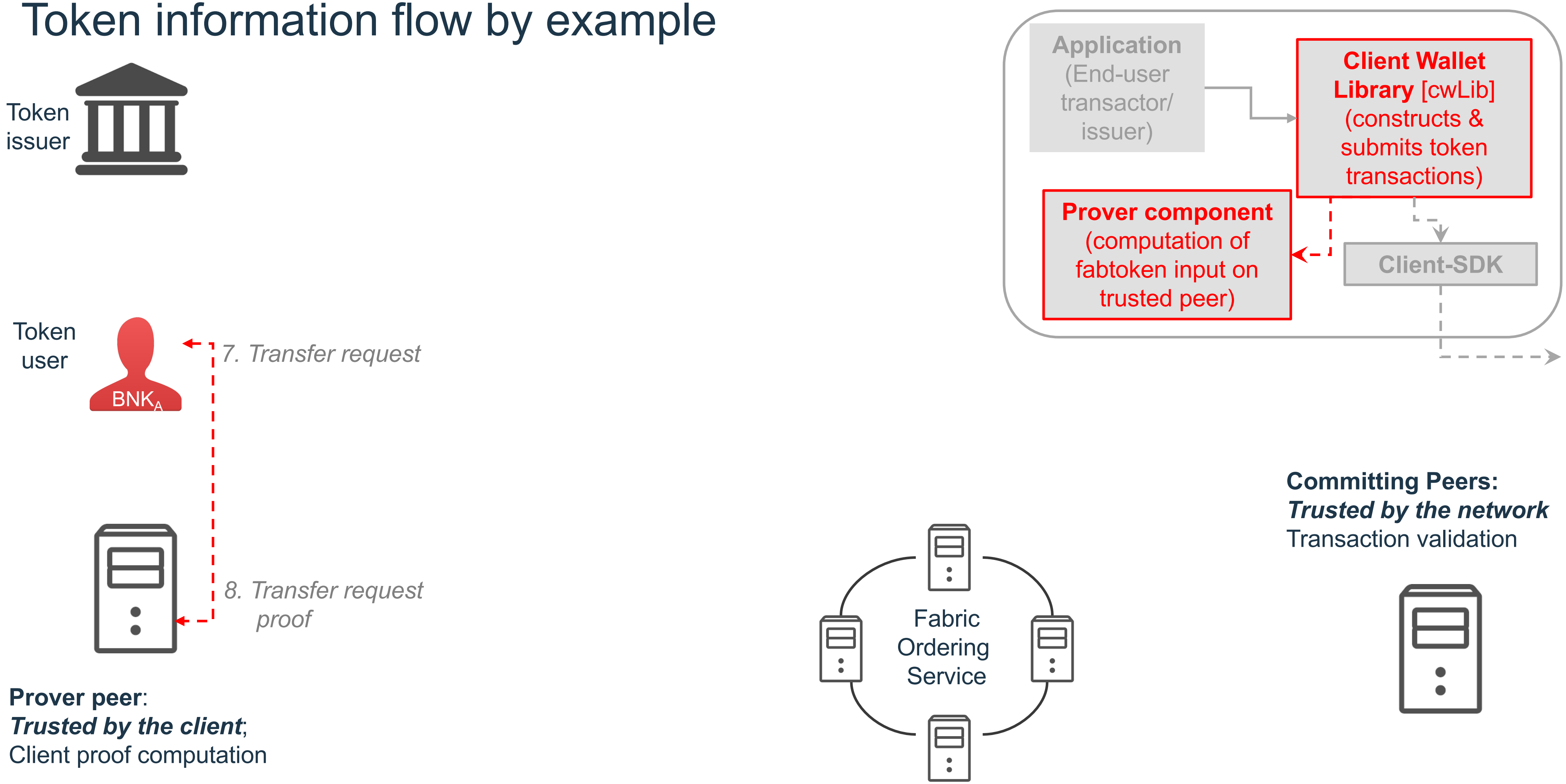
6. List of BNK$_A$ *tokens*

Fabric Ordering Service

Committing Peers:
Trusted by the network
Transaction validation

Prover peer:
Trusted by the client;
Client proof computation

Token information flow by example



Token issuer

Token user

BNK_A

7. Transfer request

8. Transfer request proof

Prover peer:
Trusted by the client;
Client proof computation

Fabric Ordering Service

Committing Peers:
Trusted by the network
Transaction validation

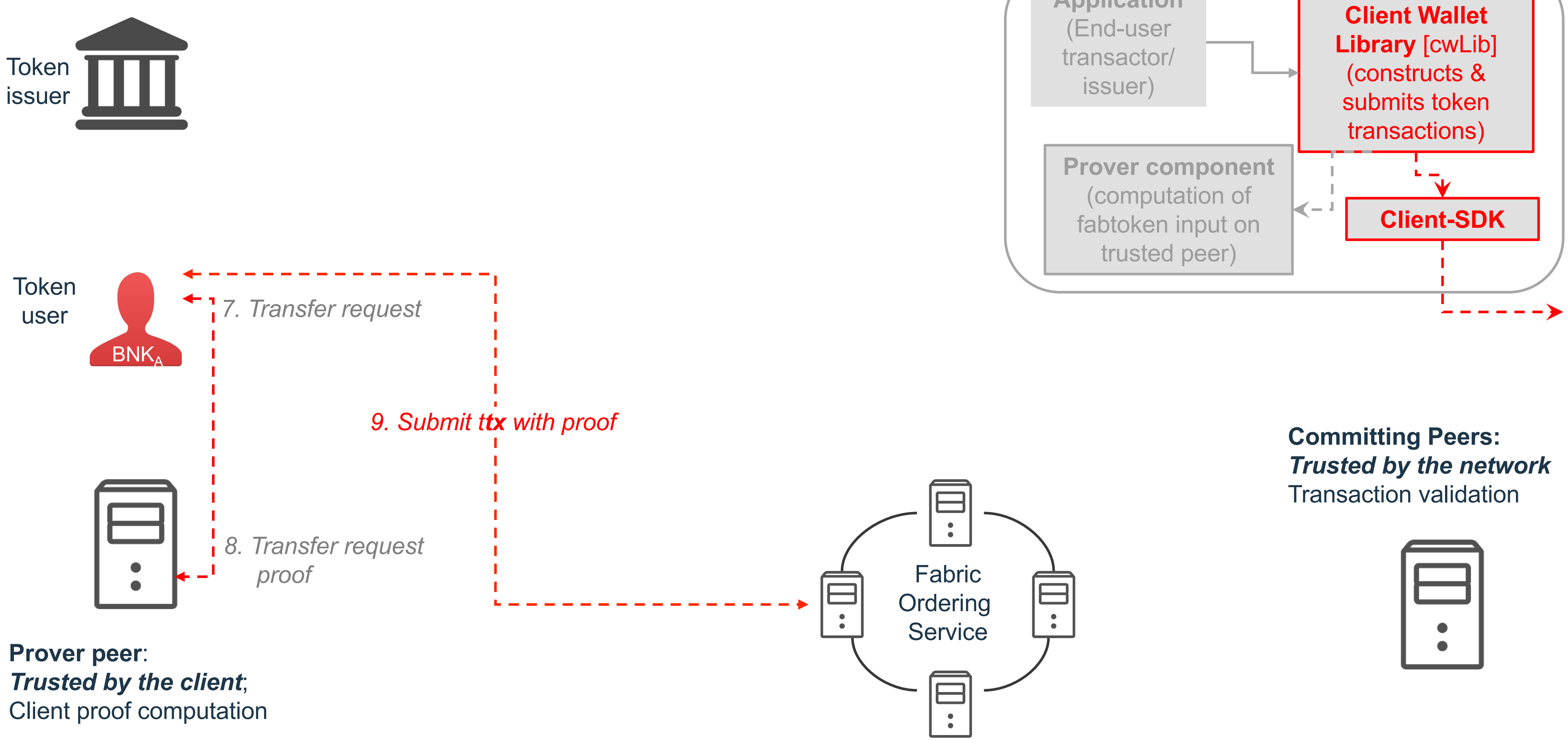
Application (End-user transactor/ issuer)

Client Wallet Library [cwLib] (constructs & submits token transactions)

Prover component (computation of fabtoken input on trusted peer)

Client-SDK

Token information flow by example



Token issuer

Token user

BNK$_A$

7. Transfer request

8. Transfer request proof

*9. Submit t**tx** with proof*

Prover peer:
Trusted by the client;
Client proof computation

Fabric Ordering Service

Committing Peers:
Trusted by the network
Transaction validation

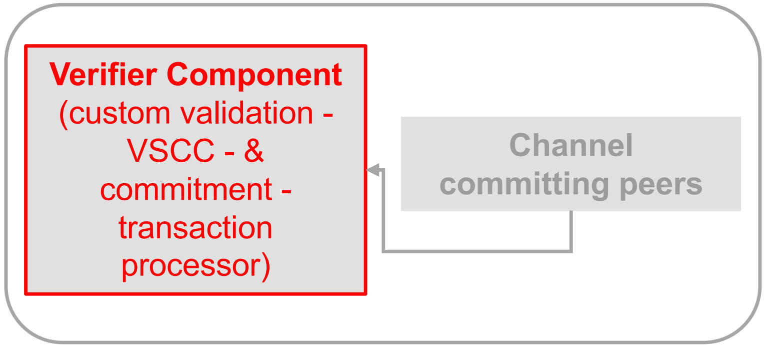
Application (End-user transactor/ issuer)

Client Wallet Library [cwLib] (constructs & submits token transactions)

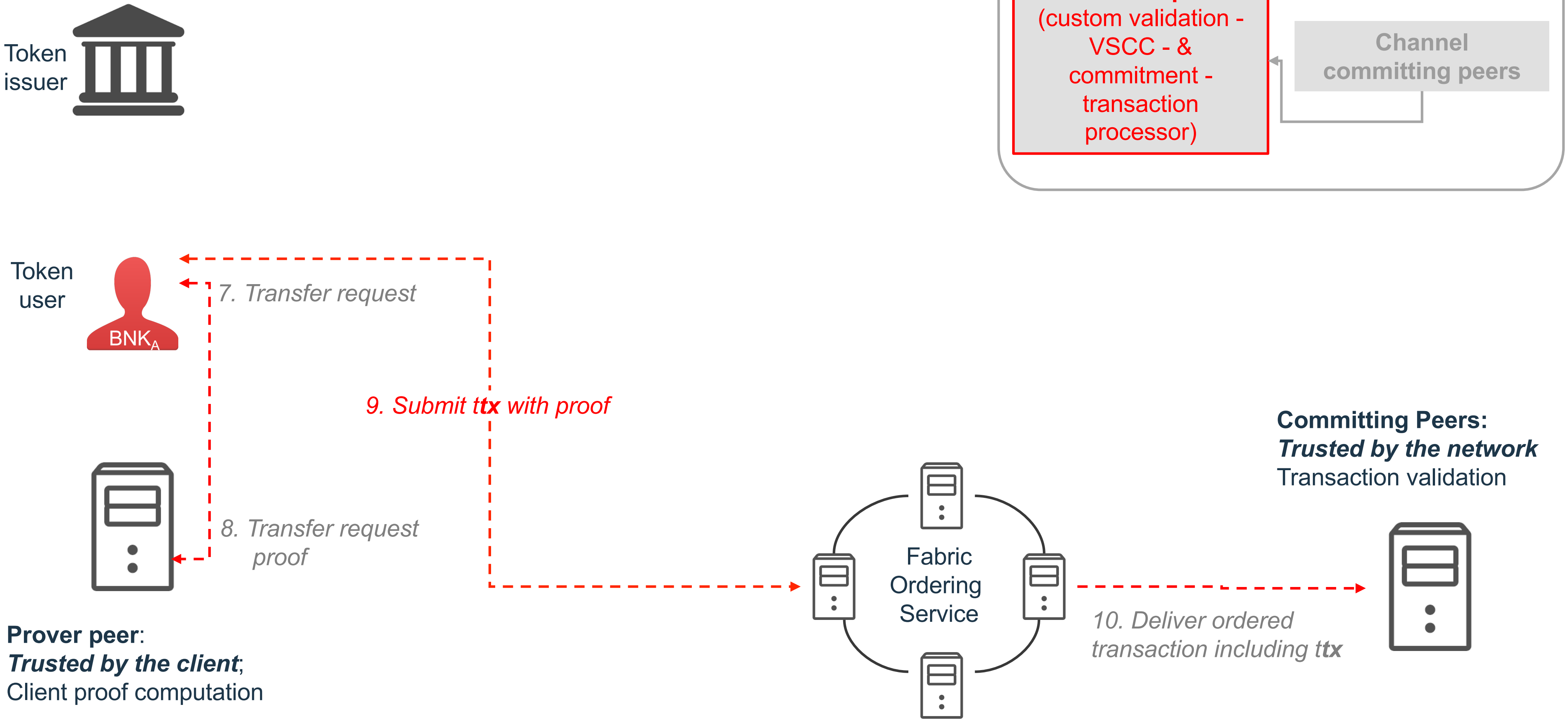
Prover component (computation of fabtoken input on trusted peer)

Client-SDK

Token information flow by example



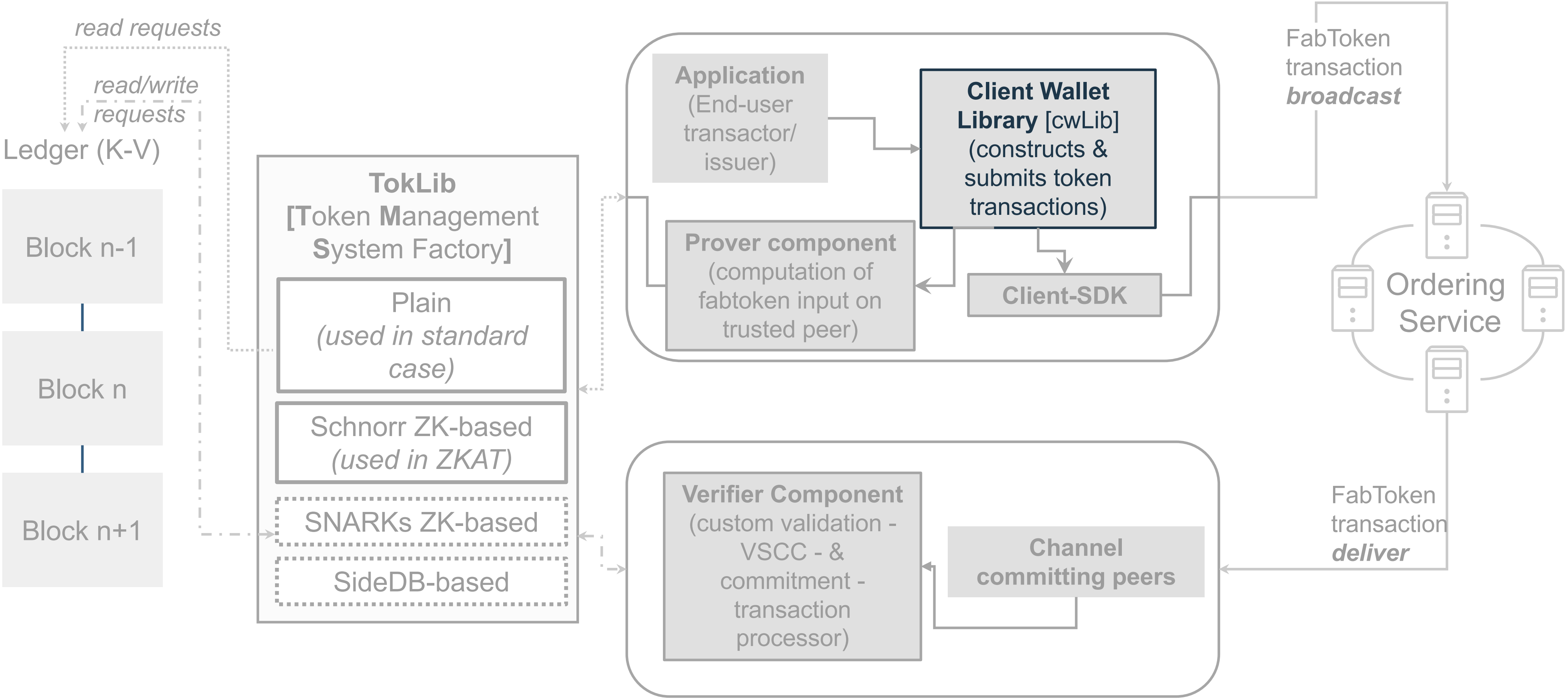
Verifier Component
(custom validation - VSCC - & commitment - transaction processor)
Channel committing peers



Committing Peers:
Trusted by the network
Transaction validation

Prover peer:
Trusted by the client;
Client proof computation

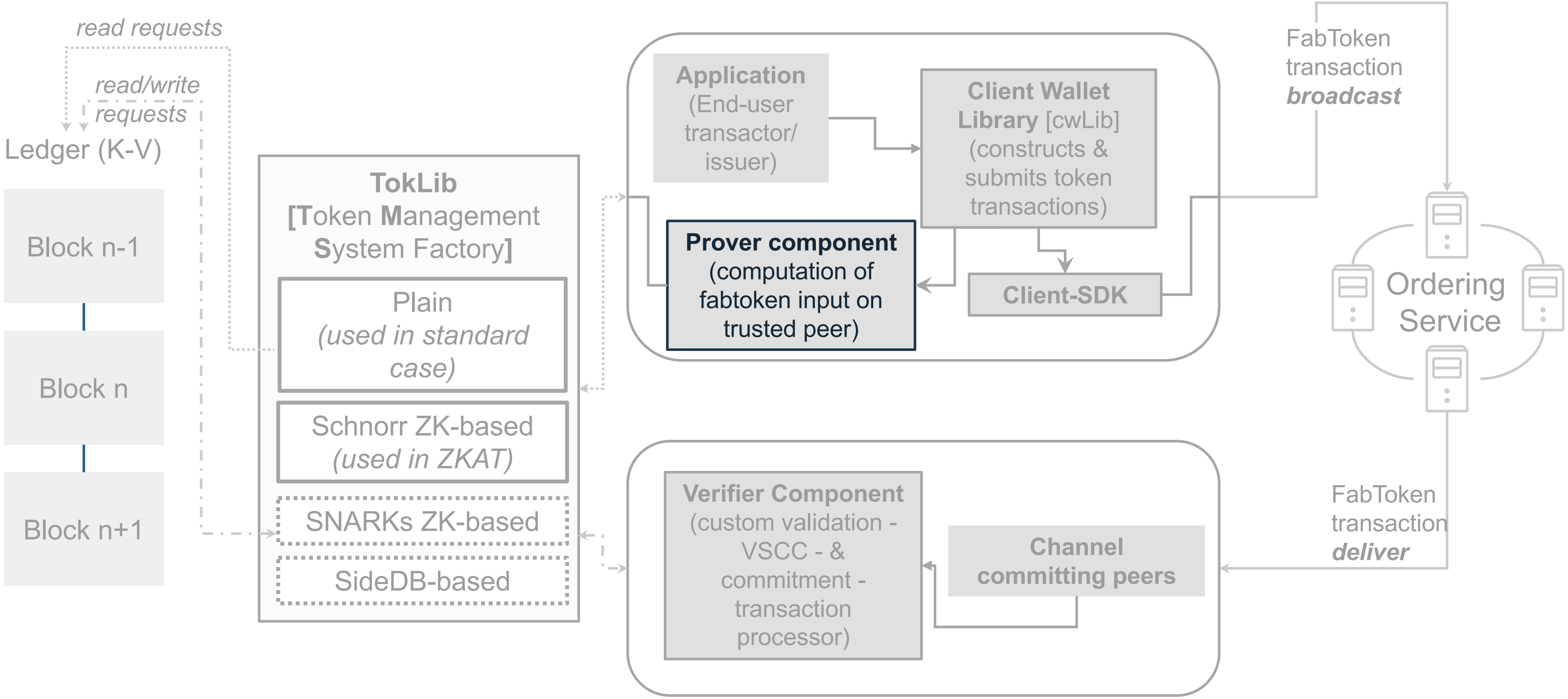
FabToken exhibits a modular architecture to accommodate various privacy levels


read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib [Token Management System Factory]
Plain (used in standard case)
Schnorr ZK-based (used in ZKAT)
SNARKs ZK-based
SideDB-based
Application (End-user transactor/ issuer)
Client Wallet Library [cwLib] (constructs & submits token transactions)
Prover component (computation of fabtoken input on trusted peer)
Client-SDK
FabToken transaction broadcast
Ordering Service
FabToken transaction deliver
Verifier Component (custom validation - VSCC - & commitment - transaction processor)
Channel committing peers

Client wallet library

- A library to expose user-friendly token functionalities to end user/application developer
- https://jira.hyperledger.org/browse/FAB-11153

FabToken exhibits a modular architecture to accommodate various privacy levels



read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib
[Token **M**anagement
System **F**actory**]**
Plain
(used in standard case)
Schnorr ZK-based
(used in ZKAT)
SNARKs ZK-based
SideDB-based
Application
(End-user transactor/ issuer)
Client Wallet Library [cwLib]
(constructs & submits token transactions)
Prover component
(computation of fabtoken input on trusted peer)
Client-SDK
FabToken transaction ***broadcast***
Ordering Service
FabToken transaction ***deliver***
Verifier Component
(custom validation - VSCC - & commitment - transaction processor)
Channel committing peers

Prover peer

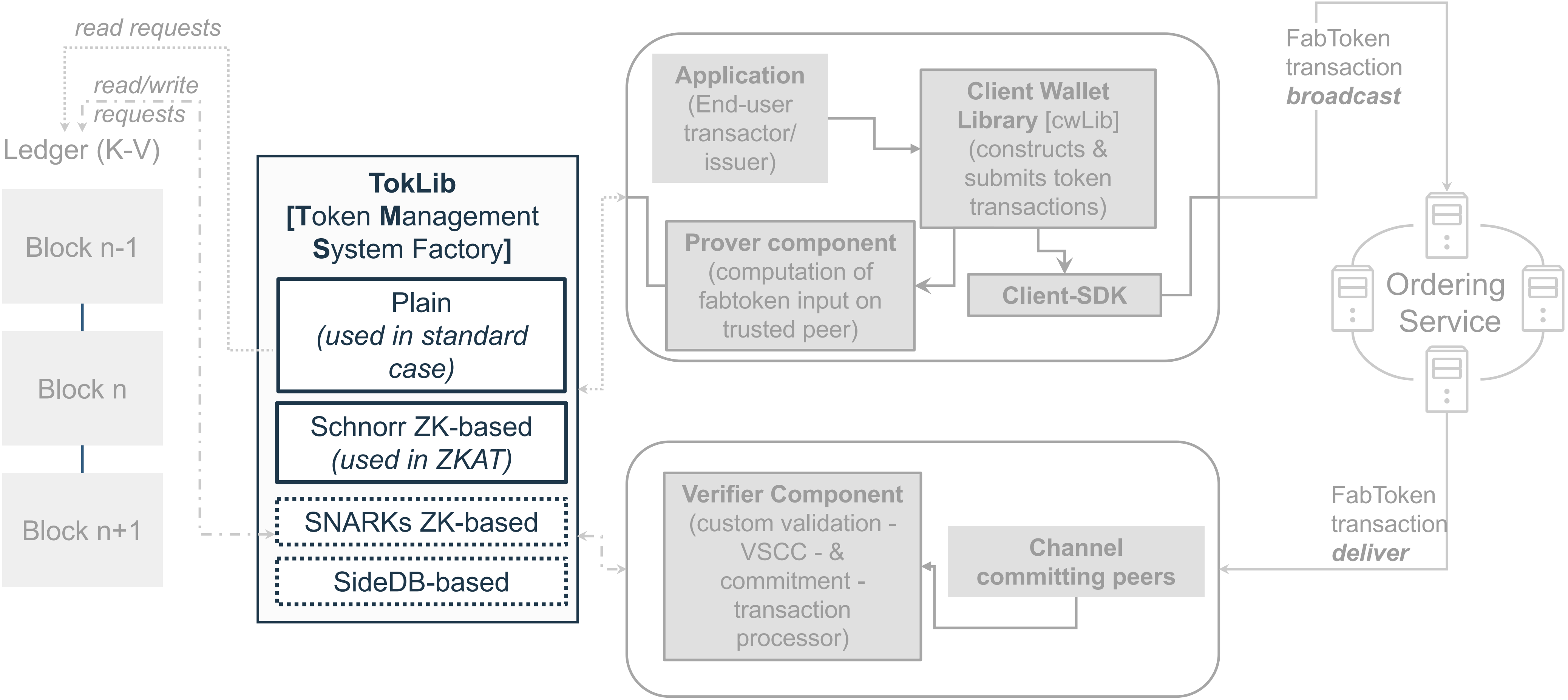
- A peer ***trusted*** by the client to
  - Perform computation on the client's behalf
  - Maintain confidential information on the client's behalf
  - Respond properly to client's ledger queries (status of transactions, list of tokens)

- Implemented as a GRPC service of a peer

- Why do we need it?
  - Client needs ledger access to compute issue, transfer proofs
  - Proof computation (esp. in the privacy-preserving case) often requires heavy computation that we want to offload to a common code base

- Currently in https://jira.hyperledger.org/browse/FAB-11149

FabToken exhibits a modular architecture to accommodate various privacy levels

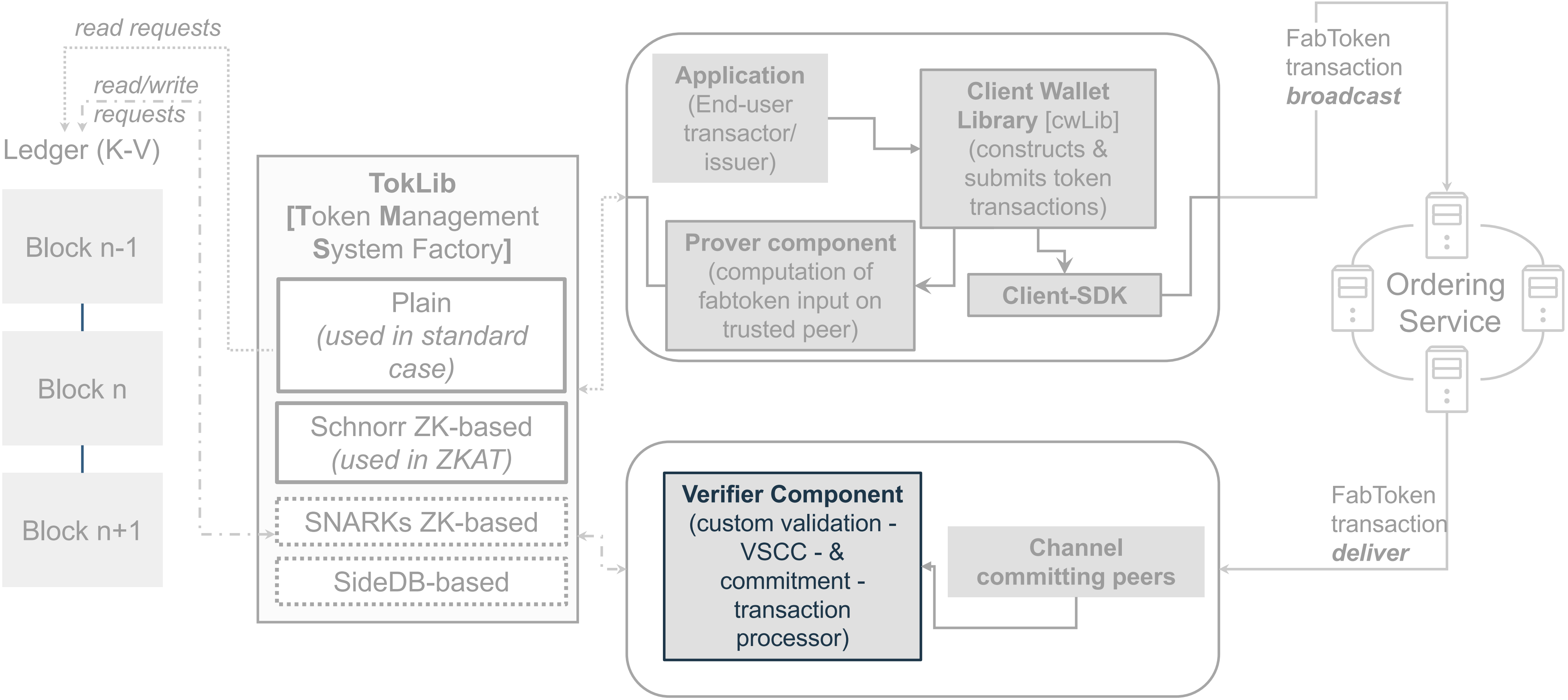

read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib
[Token Management System Factory]
Plain
(used in standard case)
Schnorr ZK-based
(used in ZKAT)
SNARKs ZK-based
SideDB-based
Application
(End-user transactor/ issuer)
Client Wallet Library [cwLib]
(constructs & submits token transactions)
Prover component
(computation of fabtoken input on trusted peer)
Client-SDK
Verifier Component
(custom validation - VSCC - & commitment - transaction processor)
Channel committing peers
FabToken transaction broadcast
Ordering Service
FabToken transaction deliver

Token Management System

- An abstraction to represent token management low-level operations (i.e., proof computation & verification)

- Currently as parts of two epics:
    - https://jira.hyperledger.org/browse/FAB-11149
    - https://jira.hyperledger.org/browse/FAB-11144

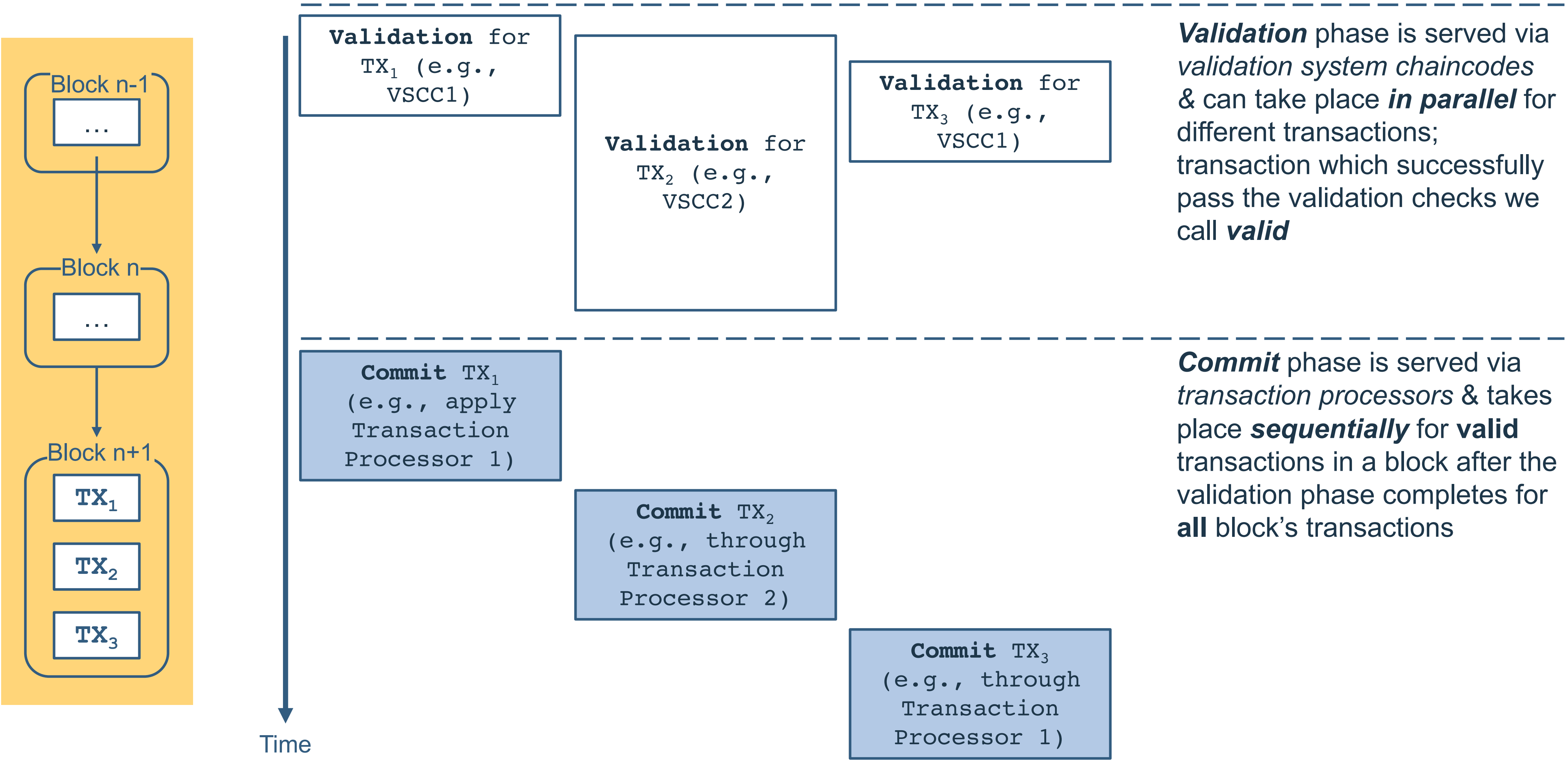
FabToken exhibits a modular architecture to accommodate various privacy levels



read requests
read/write requests
Ledger (K-V)
Block n-1
Block n
Block n+1
TokLib [Token Management System Factory]
Plain *(used in standard case)*
Schnorr ZK-based *(used in ZKAT)*
SNARKs ZK-based
SideDB-based
Application (End-user transactor/ issuer)
Client Wallet Library [cwLib] (constructs & submits token transactions)
Prover component (computation of fabtoken input on trusted peer)
Client-SDK
FabToken transaction ***broadcast***
Ordering Service
Verifier Component (custom validation - VSCC - & commitment - transaction processor)
Channel committing peers
FabToken transaction ***deliver***

Currently in https://jira.hyperledger.org/browse/FAB-11144

Transaction processing flow @Committing peer



Block n-1
…
Block n
…
Block n+1
TX_1
TX_2
TX_3
Time
Validation for TX_1 (e.g., VSCC1)
Validation for TX_2 (e.g., VSCC2)
Validation for TX_3 (e.g., VSCC1)
Commit TX_1 (e.g., apply Transaction Processor 1)
Commit TX_2 (e.g., through Transaction Processor 2)
Commit TX_3 (e.g., through Transaction Processor 1)

Validation phase is served via *validation system chaincodes* & can take place ***in parallel*** for different transactions; transaction which successfully pass the validation checks we call ***valid***

Commit phase is served via *transaction processors* & takes place ***sequentially*** for **valid** transactions in a block after the validation phase completes for **all** block's transactions

More Diagrams

Token system bootstrapping on a given channel

- Token system stakeholders agree on the configuration of the token system & compile this into a config file, **`config`** ⇒ tools can be used to convert **`config`** into protobuf messages

- **`config`** (or protobuf equivalent) is passed to the channel stakeholders that ***deploy*** the token system using chaincode lifecycle operations, i.e.,
  - A namespace would be reserved for the token system & activated
  - **`config`** would serve as the validation parameter for validation of transactions that aim to modify state with the token system's namespace (stored in the LSCC table)

- The peer retrieves **`config`** from the ledger to:
  - serve queries to the client (prover peer) for that channel
  - setup validator/committer components for transaction validation/commit (committing peer)

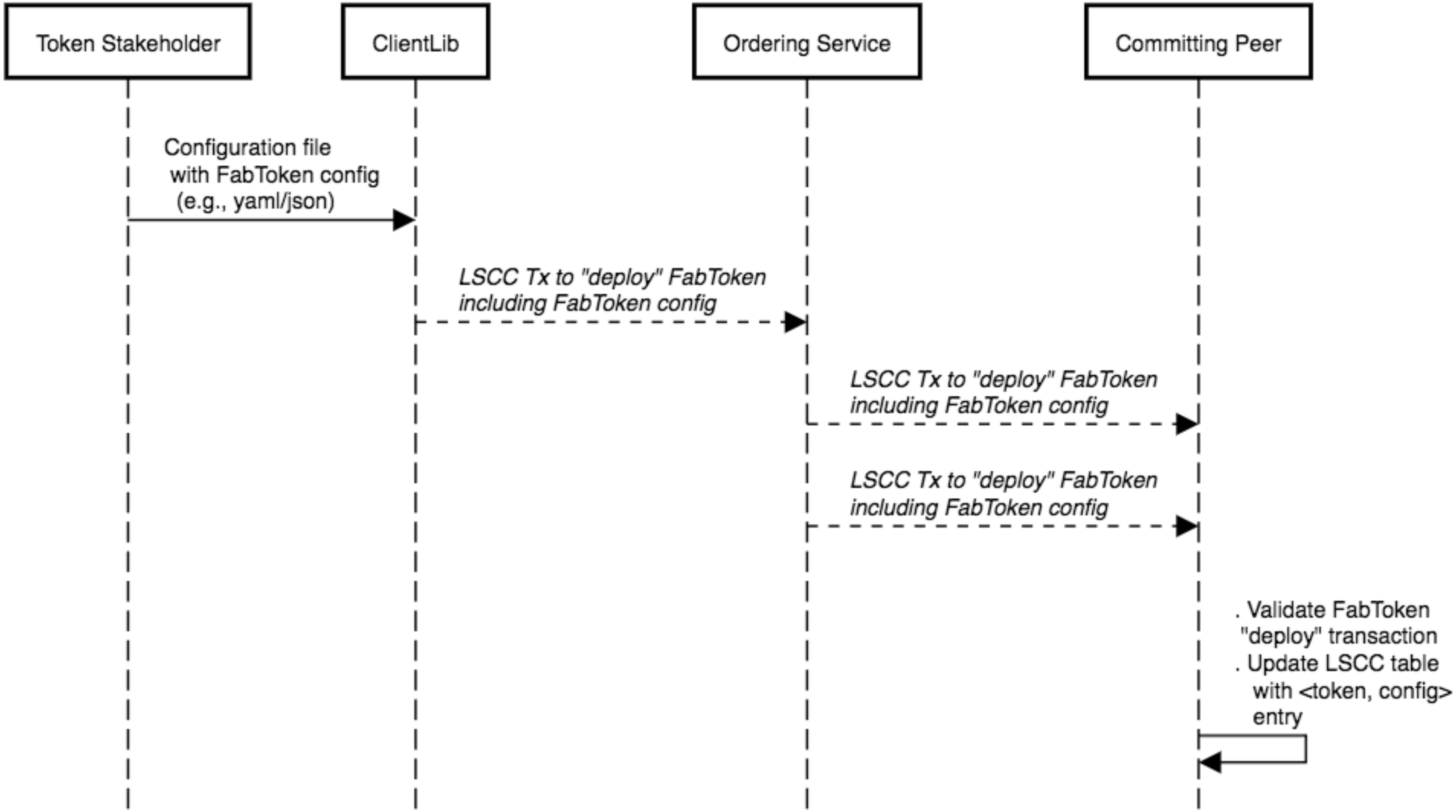
- **Trust assumptions:**
  - Channel stakeholders are trusted to propagate **`config`** for the system's deployment
  - Token stakeholders are responsible for choosing properly parameters in **`config`**
  - Clients trust their prover peers for i) setup, ii) transaction construction, iii) queries on ledger state

Token system bootstrapping on a given channel

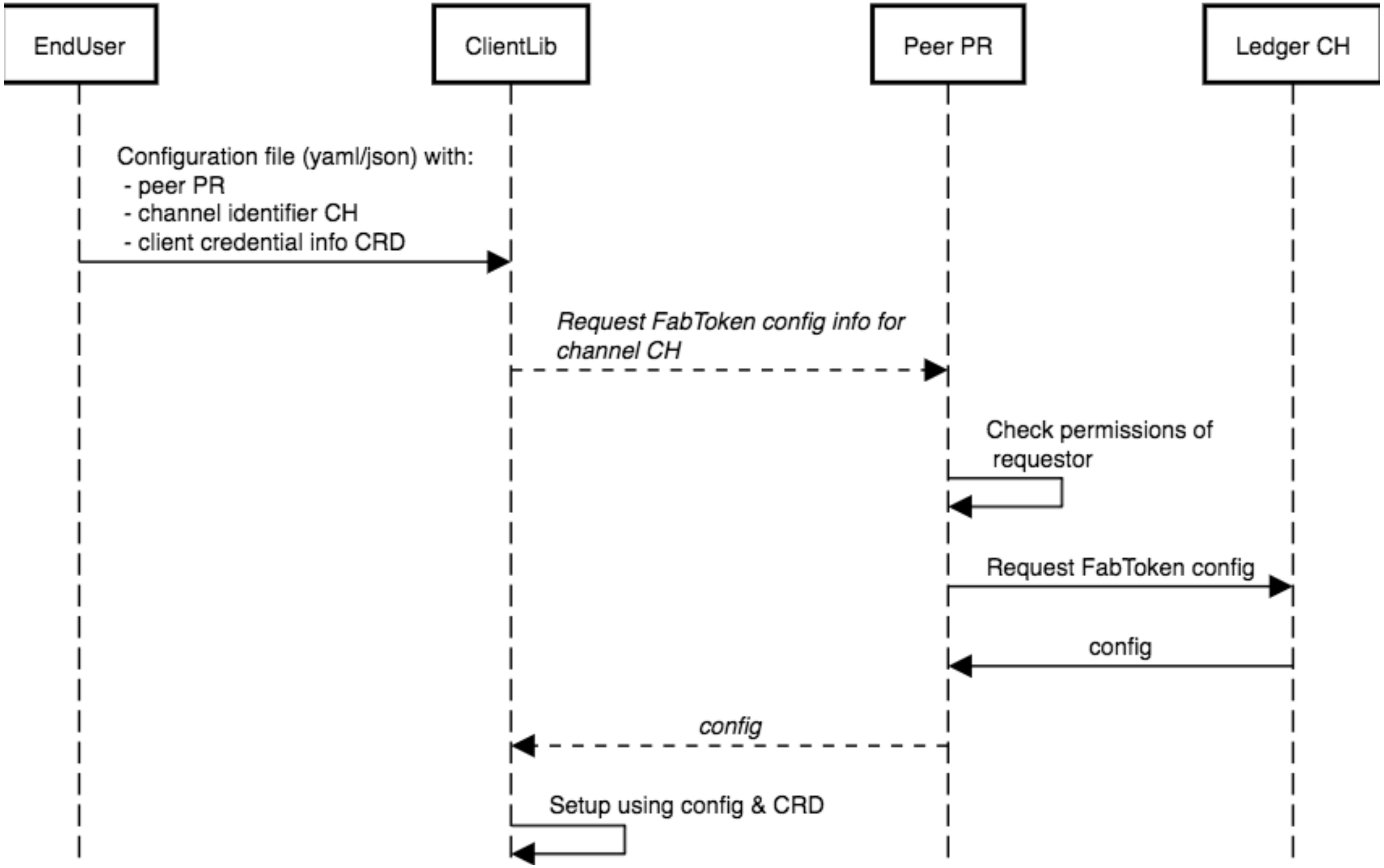
- Related JIRAs for peer setup:
  - https://jira.hyperledger.org/browse/FAB-11285
  - https://jira.hyperledger.org/browse/FAB-11169

- Related JIRAs for client setup:
  - https://jira.hyperledger.org/browse/FAB-11286

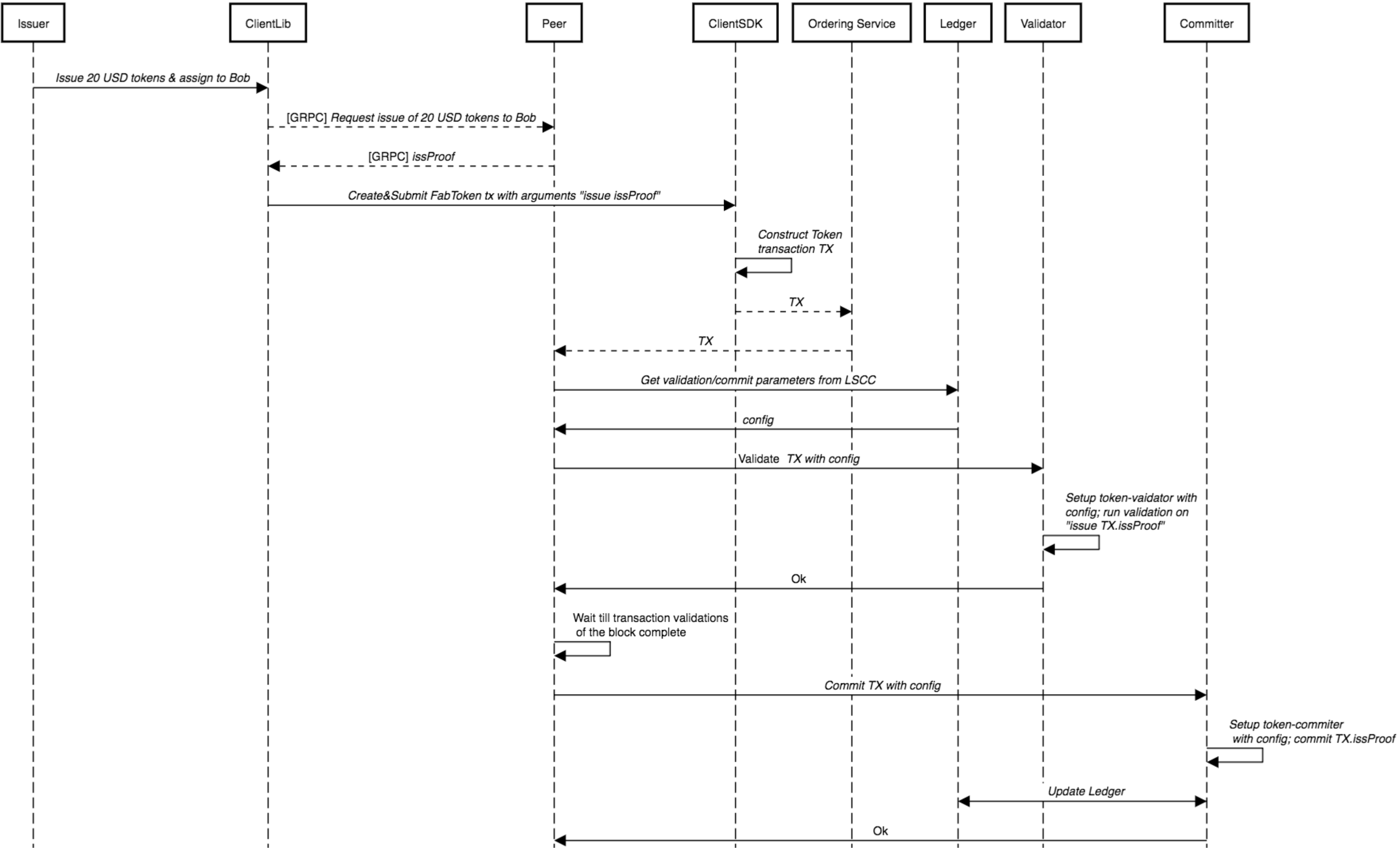
Token system setup


Token Stakeholder
ClientLib
Ordering Service
Committing Peer
Configuration file with FabToken config (e.g., yaml/json)
LSCC Tx to "deploy" FabToken including FabToken config
LSCC Tx to "deploy" FabToken including FabToken config
LSCC Tx to "deploy" FabToken including FabToken config
. Validate FabToken "deploy" transaction
. Update LSCC table with <token, config> entry

Client setup flow


EndUser
ClientLib
Peer PR
Ledger CH
Configuration file (yaml/json) with:
- peer PR
- channel identifier CH
- client credential info CRD
Request FabToken config info for channel CH
Check permissions of requestor
Request FabToken config
config
config
Setup using config & CRD

Token issue



Issuer

ClientLib

Peer

ClientSDK

Ordering Service

Ledger

Validator

Committer

Issue 20 USD tokens & assign to Bob

[GRPC] *Request issue of 20 USD tokens to Bob*

[GRPC] *issProof*

Create&Submit FabToken tx with arguments "issue issProof"

Construct Token transaction TX

TX

TX

Get validation/commit parameters from LSCC

config

Validate *TX with config*

Setup token-vaidator with config; run validation on "issue TX.issProof"

Ok

Wait till transaction validations of the block complete

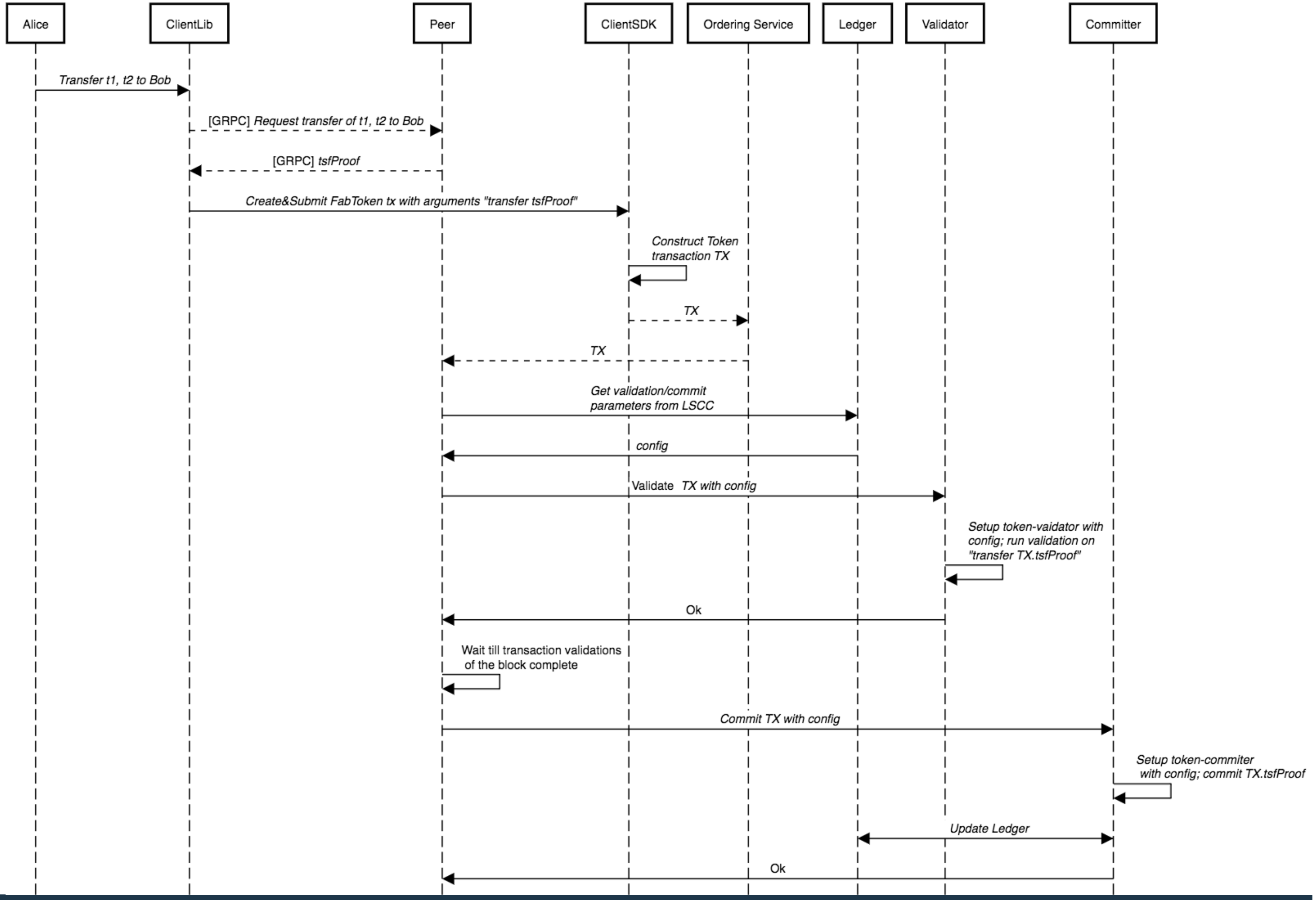
Commit TX with config

Setup token-commiter with config; commit TX.issProof

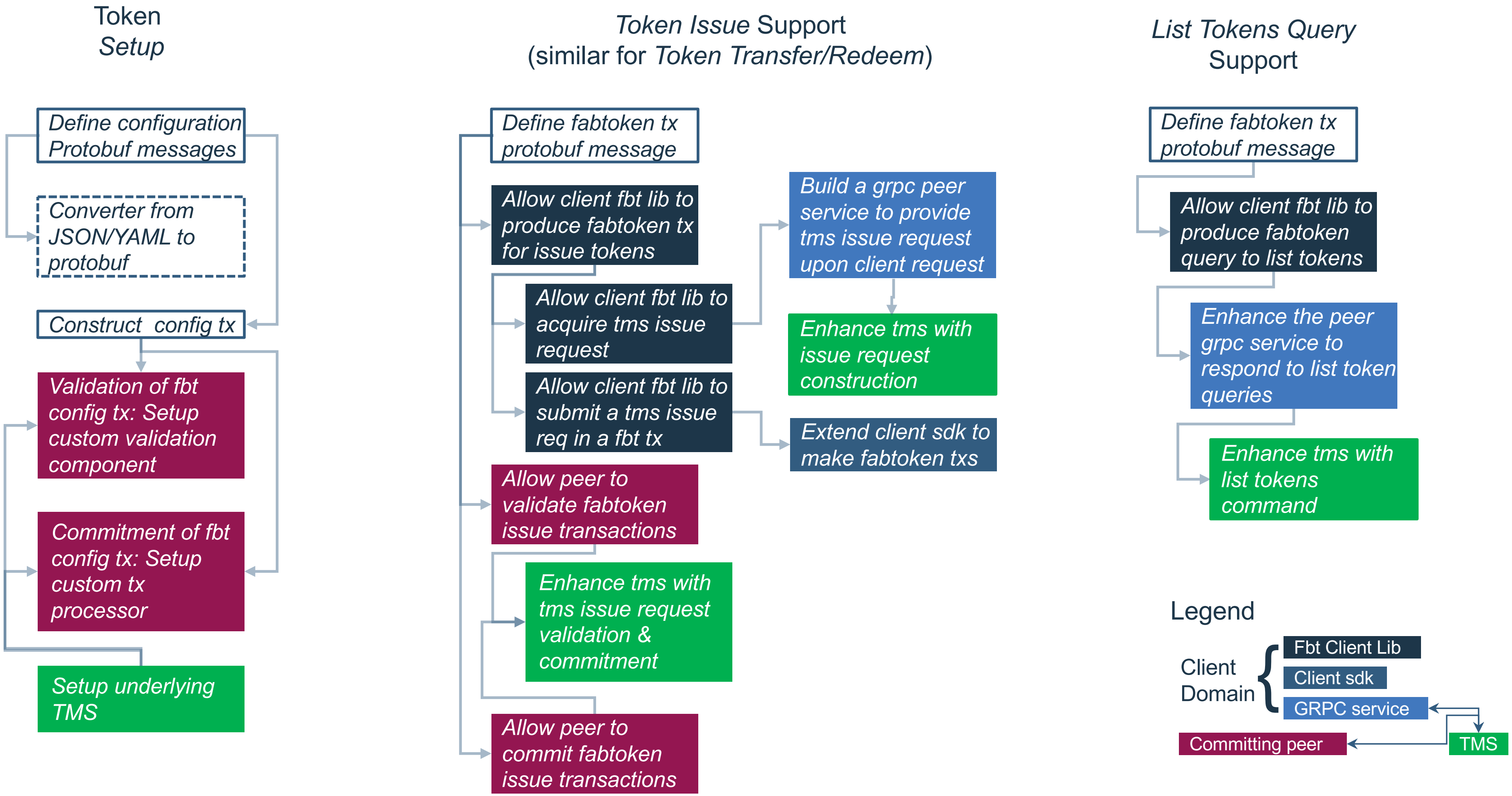
Update Ledger

Ok

Token transfer


Alice
ClientLib
Peer
ClientSDK
Ordering Service
Ledger
Validator
Committer
Transfer t1, t2 to Bob
[GRPC] Request transfer of t1, t2 to Bob
[GRPC] tsfProof
Create&Submit FabToken tx with arguments "transfer tsfProof"
Construct Token transaction TX
TX
TX
Get validation/commit parameters from LSCC
config
Validate TX with config
Setup token-vaidator with config; run validation on "transfer TX.tsfProof"
Ok
Wait till transaction validations of the block complete
Commit TX with config
Setup token-commiter with config; commit TX.tsfProof
Update Ledger
Ok

Abstraction/dependency diagram



Token *Setup*

Define configuration Protobuf messages

Converter from JSON/YAML to protobuf

Construct config tx

Validation of fbt config tx: Setup custom validation component

Commitment of fbt config tx: Setup custom tx processor

Setup underlying TMS

Token Issue Support (similar for *Token Transfer/Redeem*)

Define fabtoken tx protobuf message

Allow client fbt lib to produce fabtoken tx for issue tokens

Allow client fbt lib to acquire tms issue request

Allow client fbt lib to submit a tms issue req in a fbt tx

Build a grpc peer service to provide tms issue request upon client request

Enhance tms with issue request construction

Extend client sdk to make fabtoken txs

Allow peer to validate fabtoken issue transactions

Enhance tms with tms issue request validation & commitment

Allow peer to commit fabtoken issue transactions

List Tokens Query Support

Define fabtoken tx protobuf message

Allow client fbt lib to produce fabtoken query to list tokens

Enhance the peer grpc service to respond to list token queries

Enhance tms with list tokens command

Legend

Client Domain { Fbt Client Lib, Client sdk, GRPC service }

Committing peer

TMS