Mojaloop TigerBeetle Integration

PI-18 OSS Community Meeting

26 April 2022

Jason Bruwer, Matseliso Thabane

Agenda

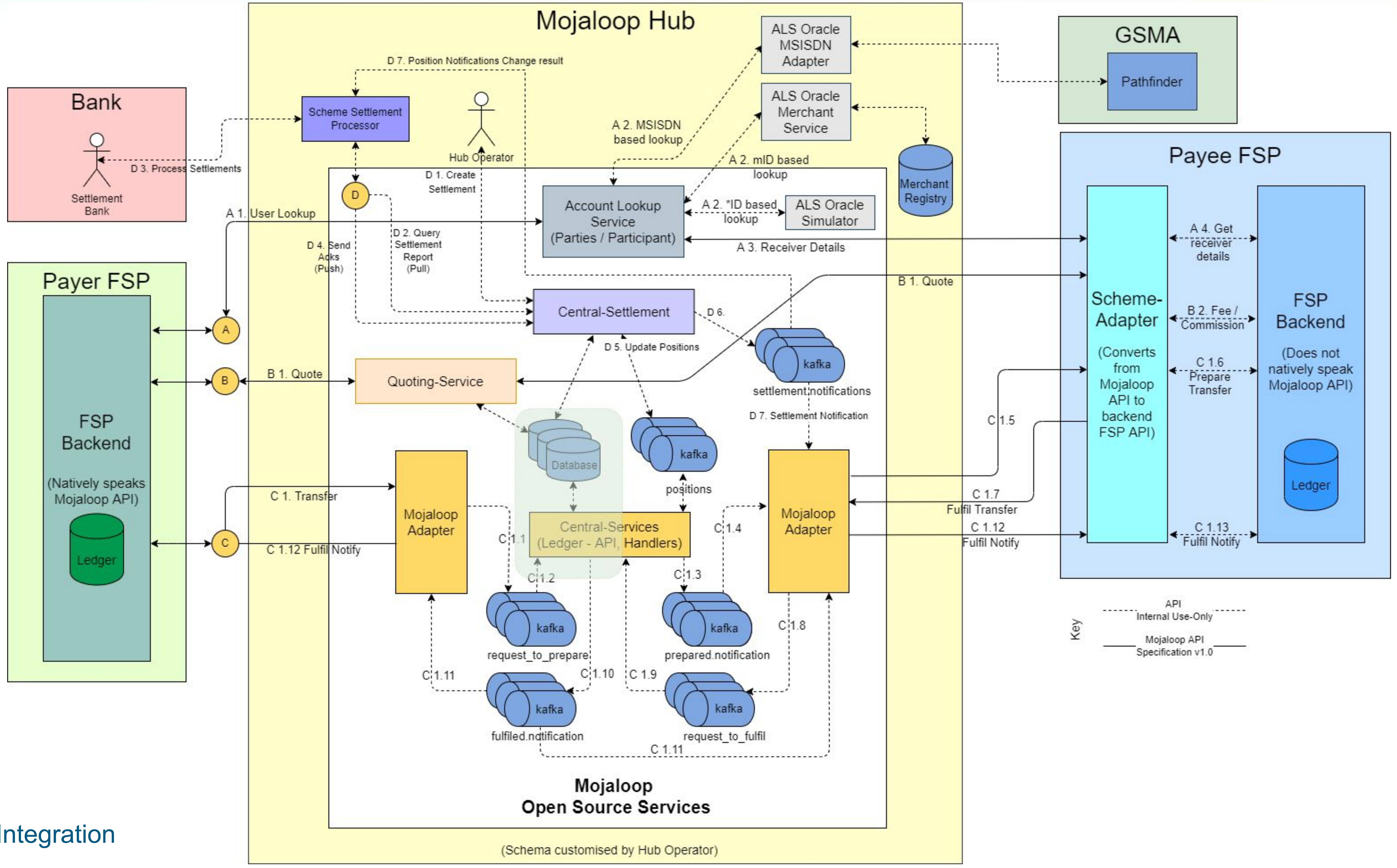
1. Progress Update
2. Hub Architecture
3. Solution Design
4. Components
5. Interaction
6. Data Migration

Progress Update

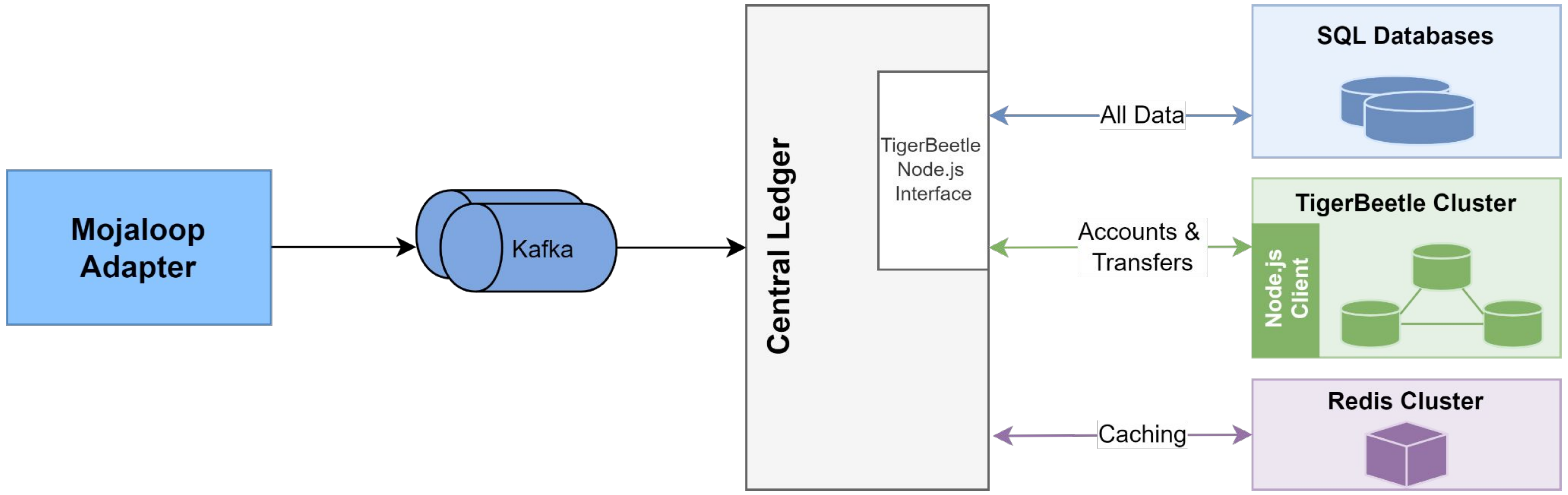
Key objectives	● Implement & test integration into Central-Ledger. ● Complete the design documentation.

Progress	**Documentation** ● Chart of Accounts for use-case subset [draft ready] ● Solution Design Document [80%] **Development** ● Implement & test the NodeJS Interface [75%] ● Integrate & test NodeJS client [75%] ● Finalise performance test suite [95%]

Hub Architecture - Example



Solution Design



Components

TigerBeetle Clients

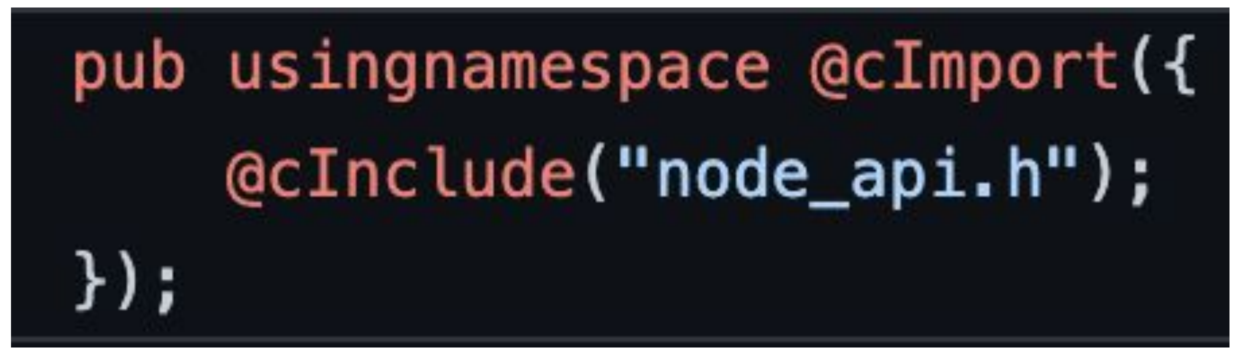
- Native client (`Zig`)
- C used to integrate language platform mappings
- Clients in NodeJS & Golang

TigerBeetle NodeJS Interface

- Configuration file (`default.json`)
- Data Mapping
- Protocol Translation
- Orchestrating interactions with TigerBeetle

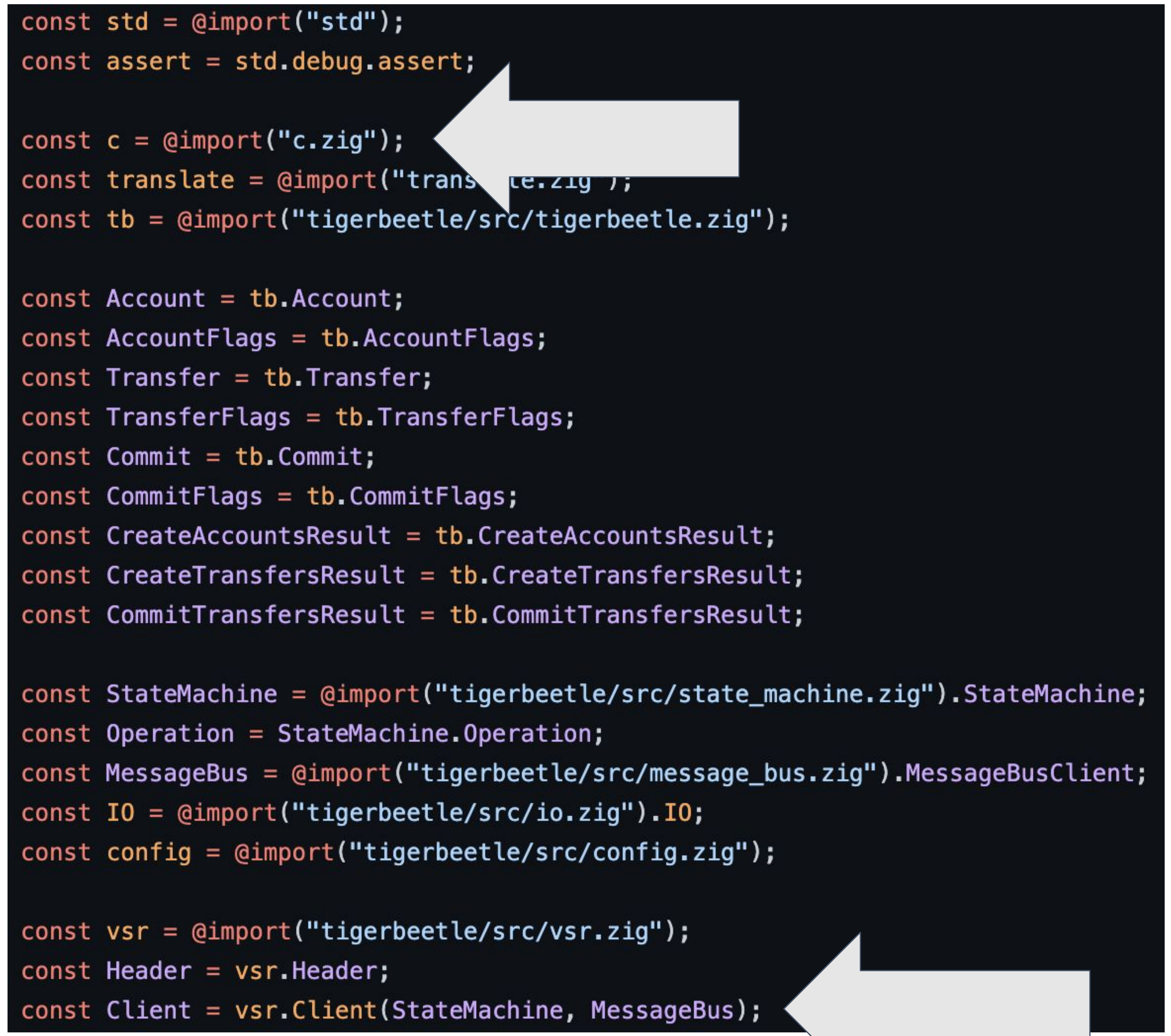
TigerBeetle NodeJS Client

Import Node C header into Zig (`c.zig`):



```
pub usingnamespace @cImport({
    @cInclude("node_api.h");
});
```

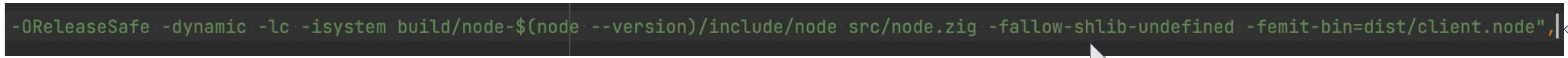
`node.zig` invoking the TigerBeetle Zig native client:



```
const std = @import("std");
const assert = std.debug.assert;

const c = @import("c.zig");
const translate = @import("trans   te.zig");
const tb = @import("tigerbeetle/src/tigerbeetle.zig");

const Account = tb.Account;
const AccountFlags = tb.AccountFlags;
const Transfer = tb.Transfer;
const TransferFlags = tb.TransferFlags;
const Commit = tb.Commit;
const CommitFlags = tb.CommitFlags;
const CreateAccountsResult = tb.CreateAccountsResult;
const CreateTransfersResult = tb.CreateTransfersResult;
const CommitTransfersResult = tb.CommitTransfersResult;

const StateMachine = @import("tigerbeetle/src/state_machine.zig").StateMachine;
const Operation = StateMachine.Operation;
const MessageBus = @import("tigerbeetle/src/message_bus.zig").MessageBusClient;
const IO = @import("tigerbeetle/src/io.zig").IO;
const config = @import("tigerbeetle/src/config.zig");

const vsr = @import("tigerbeetle/src/vsr.zig");
const Header = vsr.Header;
const Client = vsr.Client(StateMachine, MessageBus);
```

Build Process of the Client

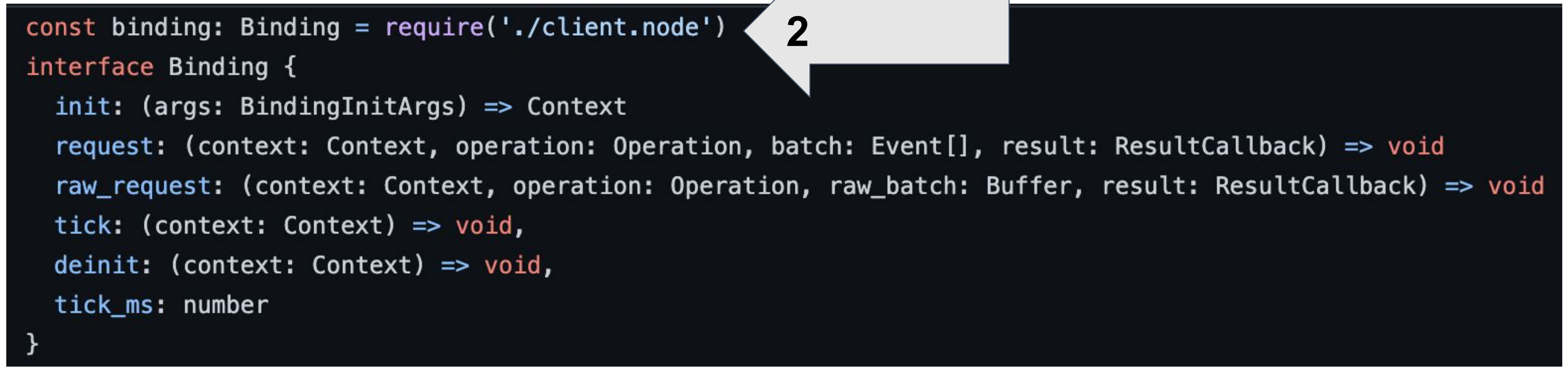
The `build:zig` script for `package.json` in tigerbeetle-node repo:



```
-OReleaseSafe -dynamic -lc -isystem build/node-$(node --version)/include/node src/node.zig -fallow-shlib-undefined -femit-bin=dist/client.node",
```

1

The binding for `client.node` in `index.ts`:

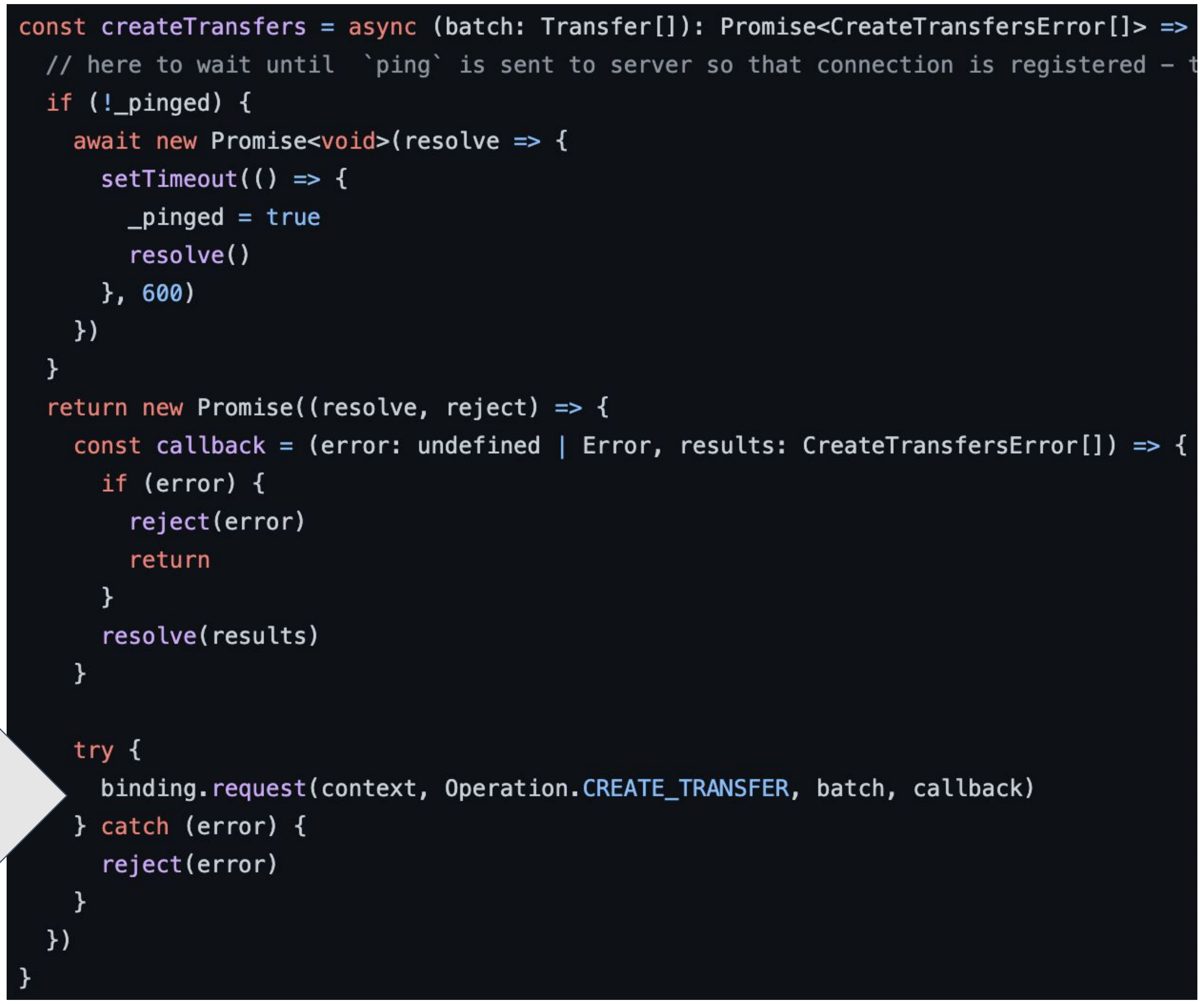


```
const binding: Binding = require('./client.node')
interface Binding {
  init: (args: BindingInitArgs) => Context
  request: (context: Context, operation: Operation, batch: Event[], result: ResultCallback) => void
  raw_request: (context: Context, operation: Operation, raw_batch: Buffer, result: ResultCallback) => void
  tick: (context: Context) => void,
  deinit: (context: Context) => void,
  tick_ms: number
}
```

2

3

Function example `createTransfers`:



```
const createTransfers = async (batch: Transfer[]): Promise<CreateTransfersError[]> =>
  // here to wait until  `ping` is sent to server so that connection is registered - t
  if (!_pinged) {
    await new Promise<void>(resolve => {
      setTimeout(() => {
        _pinged = true
        resolve()
      }, 600)
    })
  }
  return new Promise((resolve, reject) => {
    const callback = (error: undefined | Error, results: CreateTransfersError[]) => {
      if (error) {
        reject(error)
        return
      }
      resolve(results)
    }

    try {
      binding.request(context, Operation.CREATE_TRANSFER, batch, callback)
    } catch (error) {
      reject(error)
    }
  })
}
```

4

Components

TigerBeetle Clients

- Native client (`Zig`)
- C used to integrate language platform mappings
- Clients in NodeJS & Golang

TigerBeetle NodeJS Interface

- Configuration file (`default.json`)
- Data Mapping
- Protocol Translation
- Orchestrating interactions with TigerBeetle

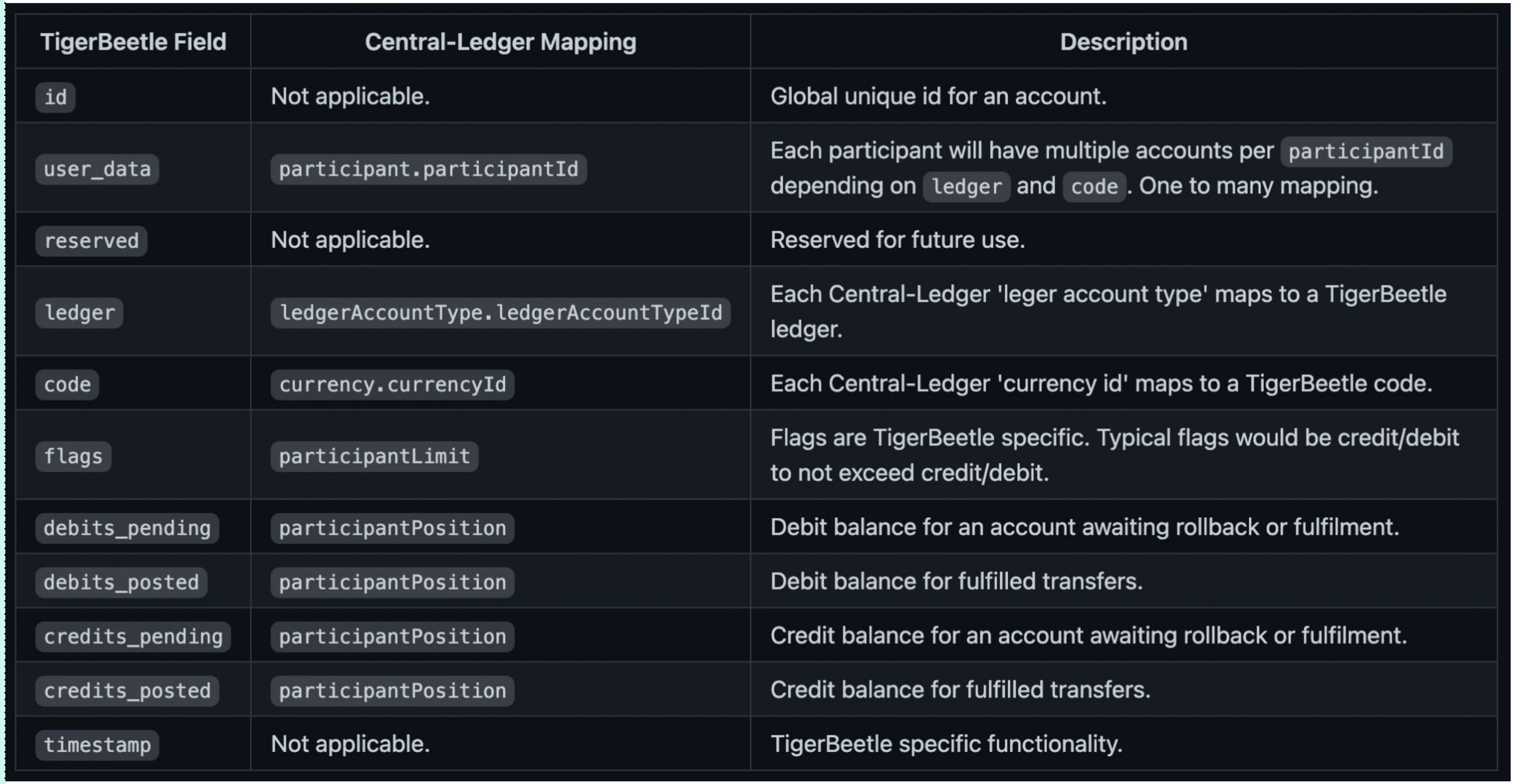
Configuration file

Update the `default.json` configuration file to enable/disable TigerBeetle:

```
"TIGERBEETLE" : {
  "ENABLED" : true,
  "ENABLE_BATCHING" : false,
  "DISABLE_SQL" : false,
  "BATCH_MAX_SIZE" : 2048,
  "CLUSTER" : 1,
  "REPLICA_ENDPOINT_01" : "localhost:5001",
  "REPLICA_ENDPOINT_02" : "localhost:5002",
  "REPLICA_ENDPOINT_03" : "localhost:5003"
},
```

Account Data Mapping

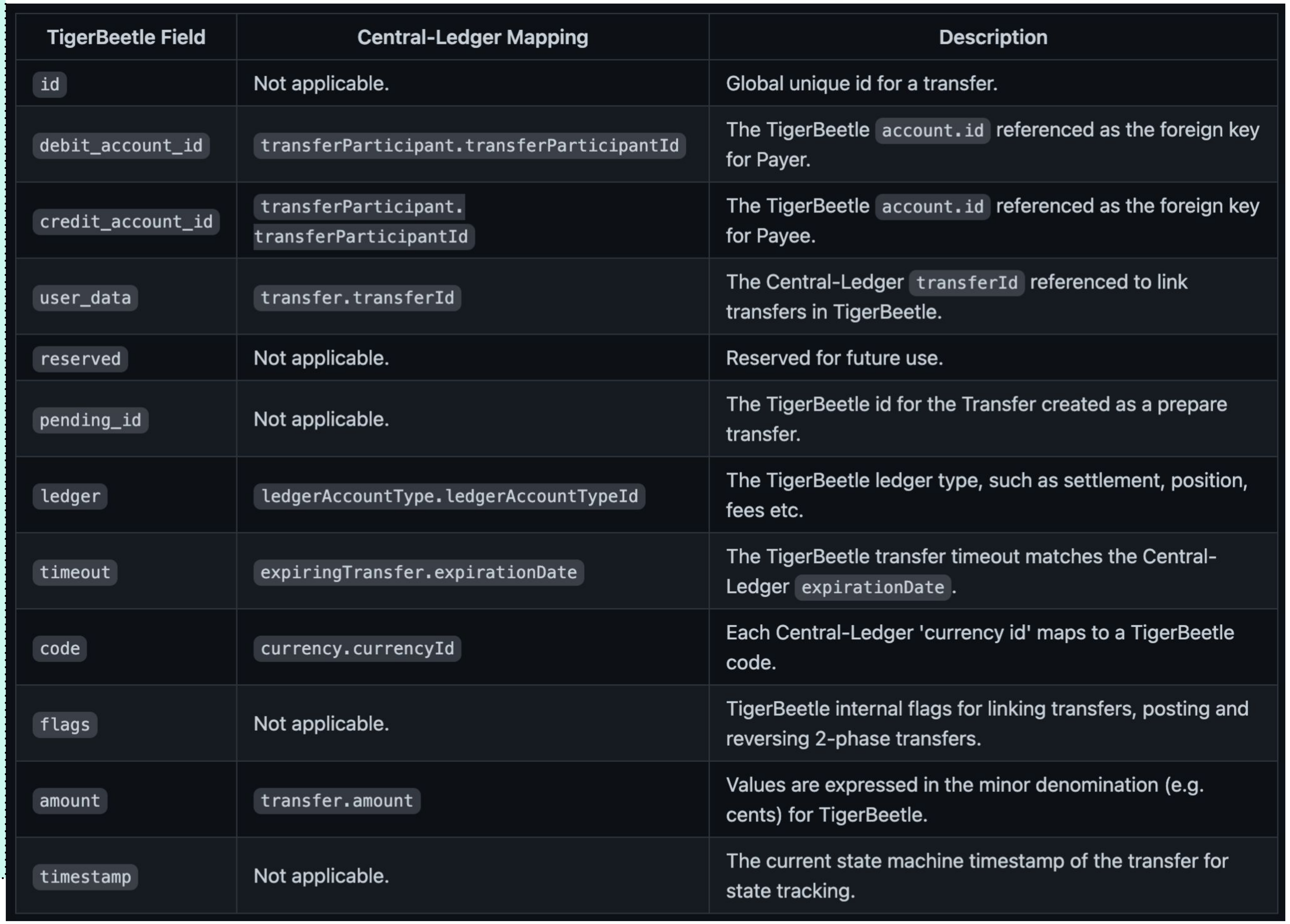
TigerBeetle NodeJS Interface



TigerBeetle Field	Central-Ledger Mapping	Description
`id`	Not applicable.	Global unique id for an account.
`user_data`	`participant.participantId`	Each participant will have multiple accounts per `participantId` depending on `ledger` and `code`. One to many mapping.
`reserved`	Not applicable.	Reserved for future use.
`ledger`	`ledgerAccountType.ledgerAccountTypeId`	Each Central-Ledger 'leger account type' maps to a TigerBeetle ledger.
`code`	`currency.currencyId`	Each Central-Ledger 'currency id' maps to a TigerBeetle code.
`flags`	`participantLimit`	Flags are TigerBeetle specific. Typical flags would be credit/debit to not exceed credit/debit.
`debits_pending`	`participantPosition`	Debit balance for an account awaiting rollback or fulfilment.
`debits_posted`	`participantPosition`	Debit balance for fulfilled transfers.
`credits_pending`	`participantPosition`	Credit balance for an account awaiting rollback or fulfilment.
`credits_posted`	`participantPosition`	Credit balance for fulfilled transfers.
`timestamp`	Not applicable.	TigerBeetle specific functionality.

Transfer Data Mapping

TigerBeetle NodeJS Interface



TigerBeetle Field	Central-Ledger Mapping	Description
`id`	Not applicable.	Global unique id for a transfer.
`debit_account_id`	`transferParticipant.transferParticipantId`	The TigerBeetle `account.id` referenced as the foreign key for Payer.
`credit_account_id`	`transferParticipant.transferParticipantId`	The TigerBeetle `account.id` referenced as the foreign key for Payee.
`user_data`	`transfer.transferId`	The Central-Ledger `transferId` referenced to link transfers in TigerBeetle.
`reserved`	Not applicable.	Reserved for future use.
`pending_id`	Not applicable.	The TigerBeetle id for the Transfer created as a prepare transfer.
`ledger`	`ledgerAccountType.ledgerAccountTypeId`	The TigerBeetle ledger type, such as settlement, position, fees etc.
`timeout`	`expiringTransfer.expirationDate`	The TigerBeetle transfer timeout matches the Central-Ledger `expirationDate`.
`code`	`currency.currencyId`	Each Central-Ledger 'currency id' maps to a TigerBeetle code.
`flags`	Not applicable.	TigerBeetle internal flags for linking transfers, posting and reversing 2-phase transfers.
`amount`	`transfer.amount`	Values are expressed in the minor denomination (e.g. cents) for TigerBeetle.
`timestamp`	Not applicable.	The current state machine timestamp of the transfer for state tracking.

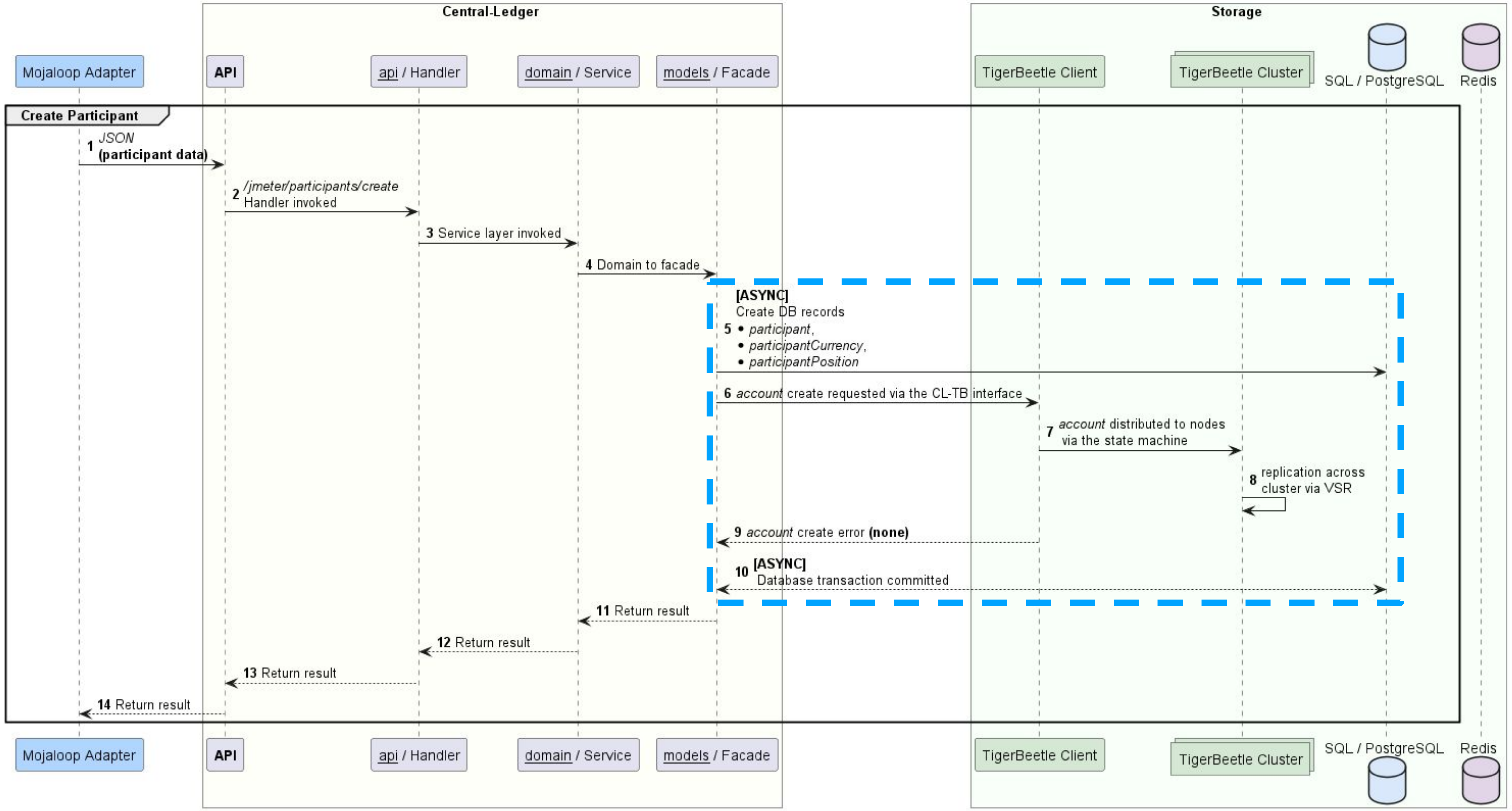
TigerBeetle NodeJS Interface

Protocol Translation Library

The `tb.js` interface transforms the Central-Ledger & TigerBeetle data models. Some of the main exposed functions are:

- **tbCreateAccount** → Create participant, currency, position and other related data
- **tbLookupAccount** → Lookup using participantId or participantCurrencyid
- **tbTransfer** → Create a transfer
- **tbPrepareTransfer** → Create a 2-phase transfer prepare (with a timeout)
- **tbFulfilTransfer** → Fulfil a 2-phase prepared transfer
- **tbRollbackTransfer** → Rollback a prepared transfer
- **tbLookupTransfer** → Lookup an existing transfer
- **tbTransferTimedOut** → Verify whether a 2-phase transfer has expired

Interaction - Create Participant



1 *JSON* **(participant data)**

2 */jmeter/participants/create* Handler invoked

3 Service layer invoked

4 Domain to facade

5 **[ASYNC]** Create DB records
- *participant*,
- *participantCurrency*,
- *participantPosition*

6 *account* create requested via the CL-TB interface

7 *account* distributed to nodes via the state machine

8 replication across cluster via VSR

9 *account* create error **(none)**

10 **[ASYNC]** Database transaction committed

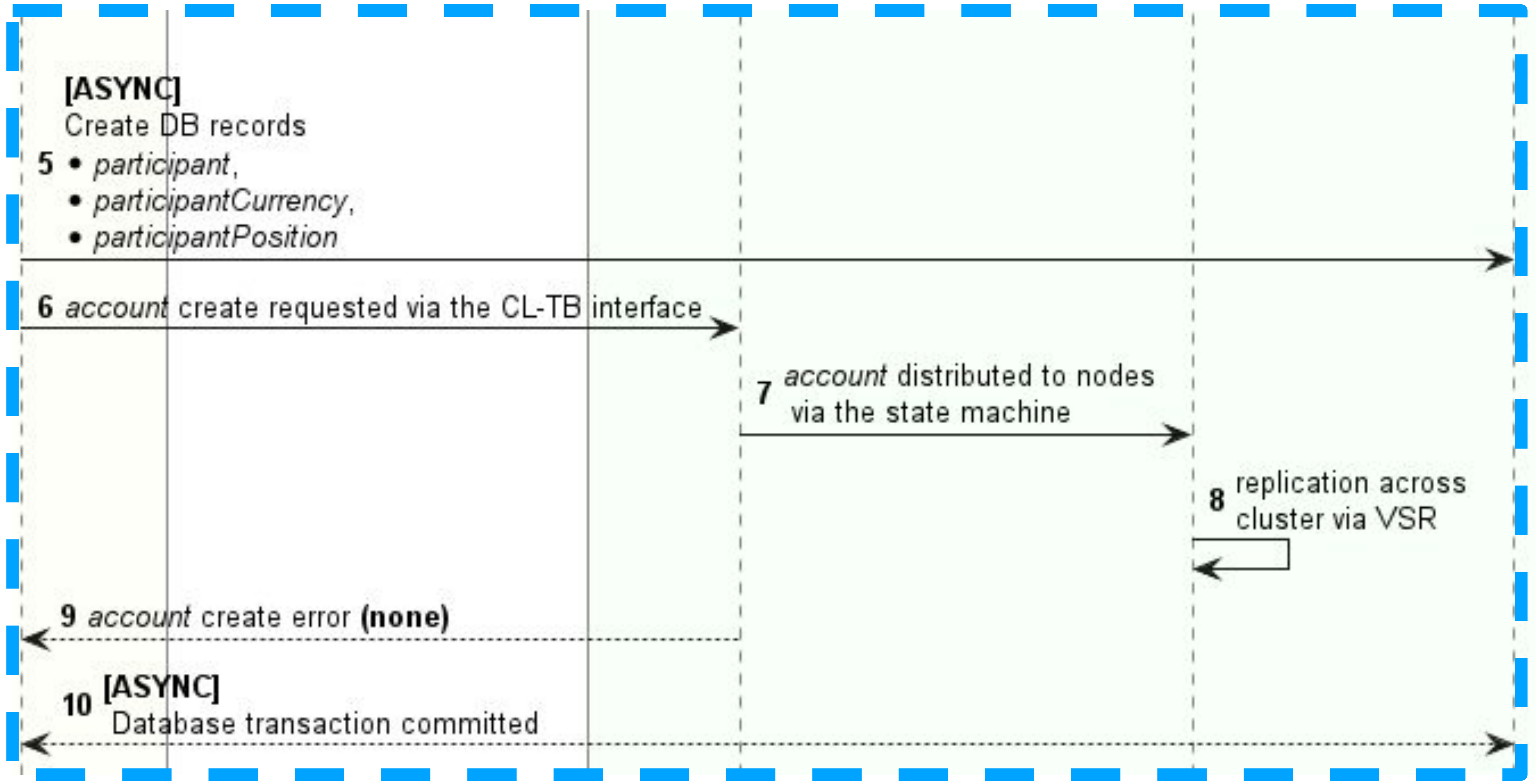
11 Return result

12 Return result

13 Return result

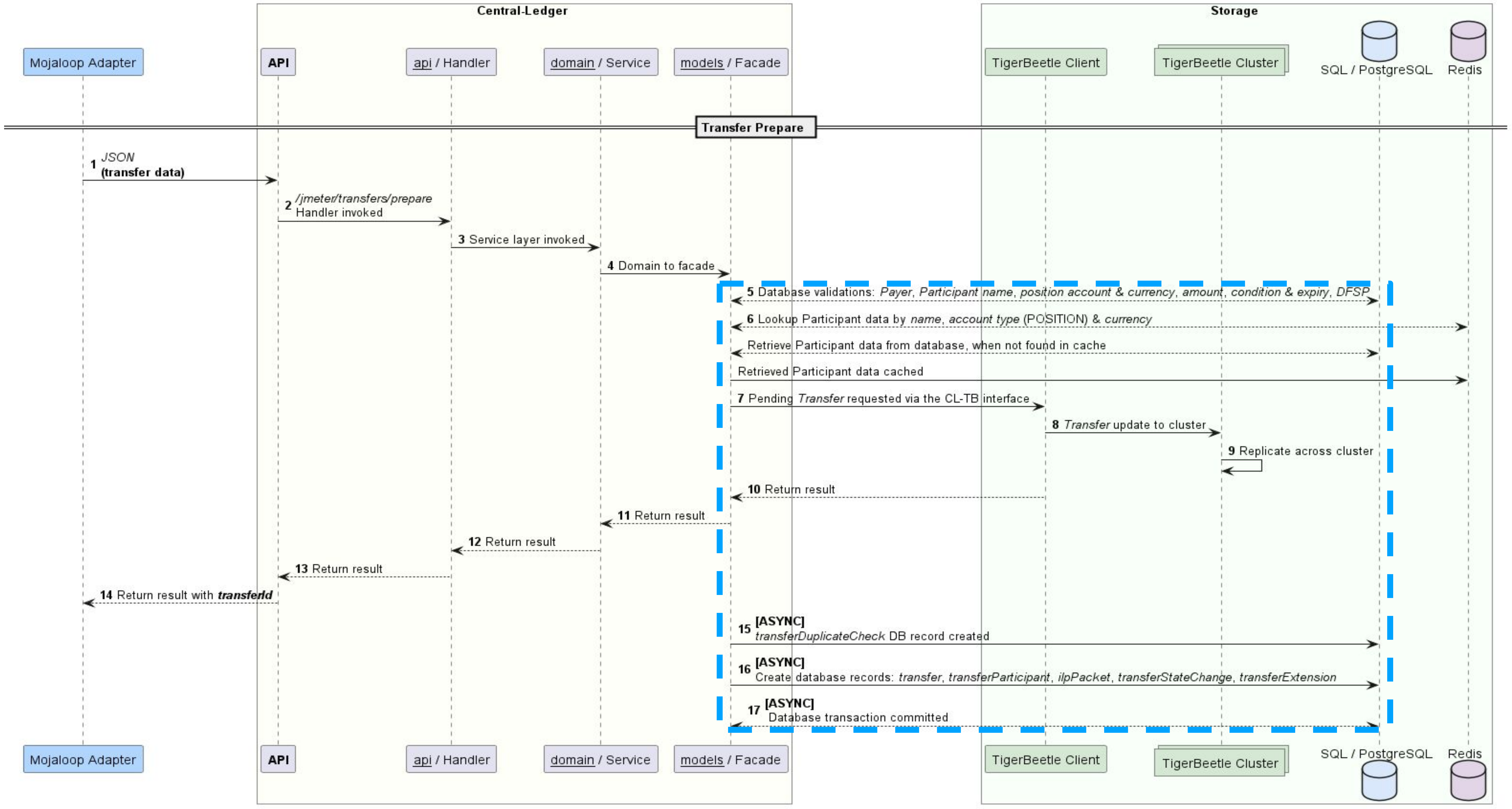
14 Return result

Interaction Detail - Create Participant



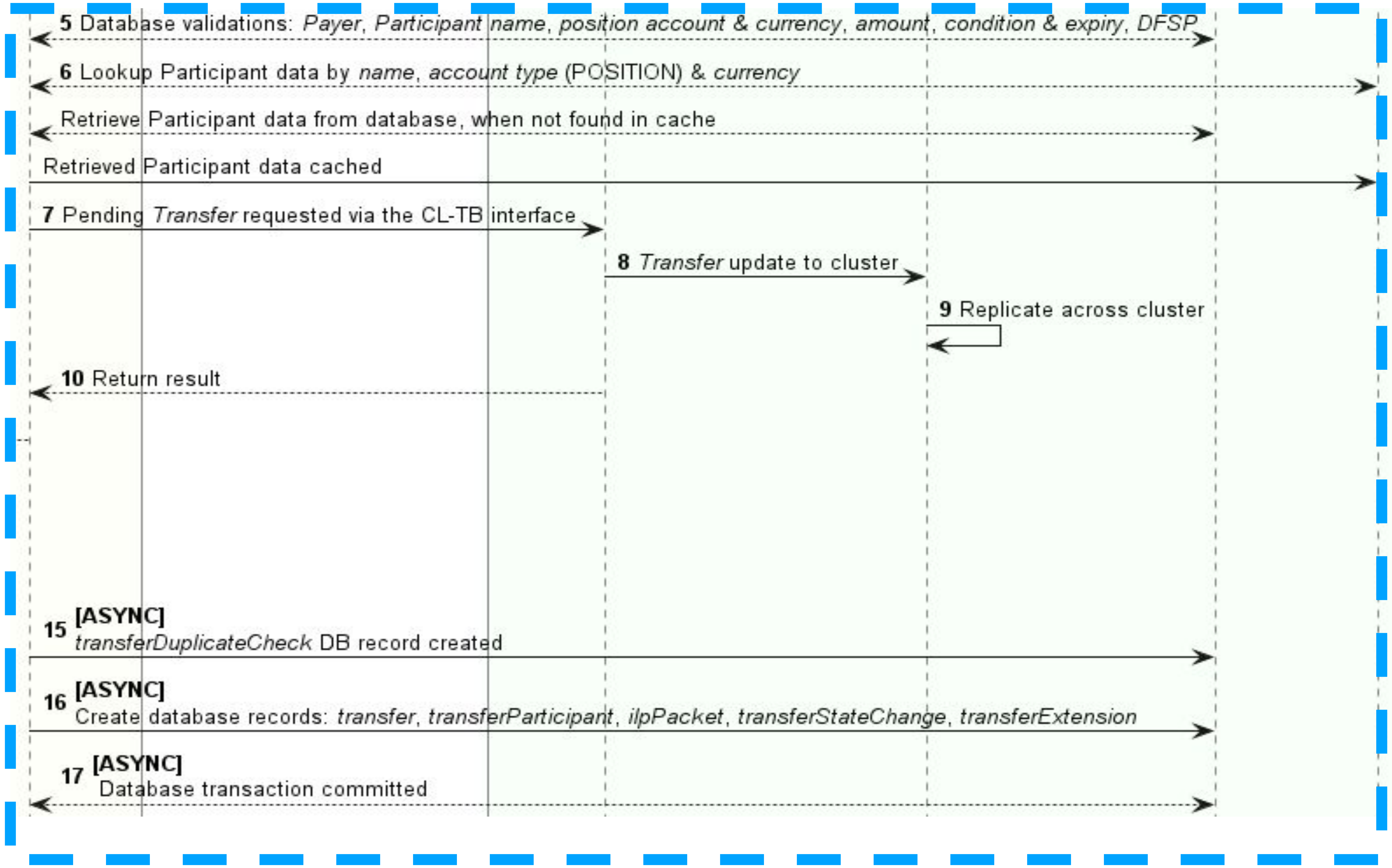
5 [ASYNC]
Create DB records
• *participant*,
• *participantCurrency*,
• *participantPosition*
6 *account* create requested via the CL-TB interface
7 *account* distributed to nodes via the state machine
8 replication across cluster via VSR
9 *account* create error **(none)**
10 [ASYNC]
Database transaction committed

Interaction - Prepare Transfer



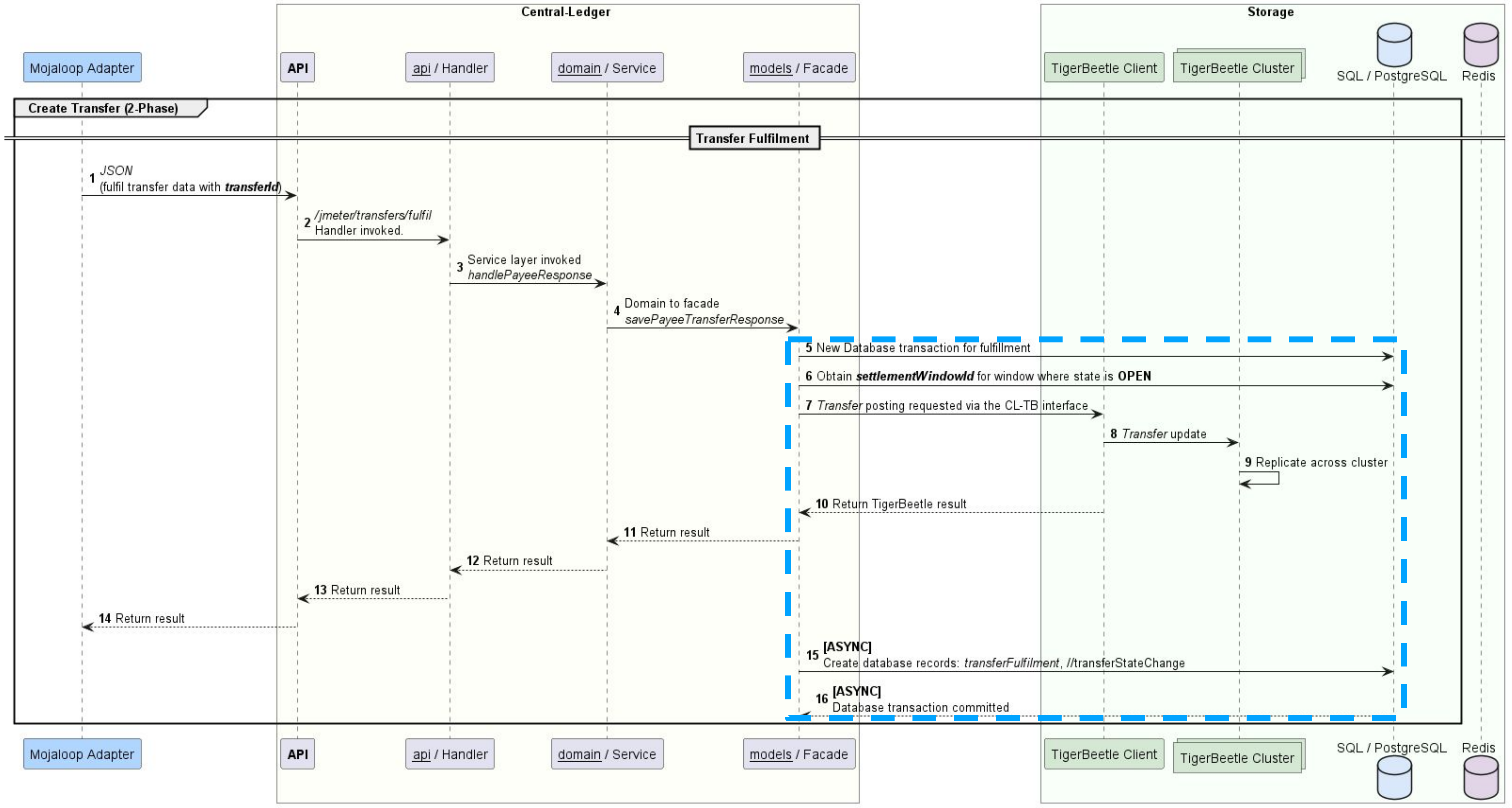
Central-Ledger
Storage
Mojaloop Adapter
API
api / Handler
domain / Service
models / Facade
TigerBeetle Client
TigerBeetle Cluster
SQL / PostgreSQL
Redis
Transfer Prepare
1 JSON (transfer data)
2 /jmeter/transfers/prepare Handler invoked
3 Service layer invoked
4 Domain to facade
5 Database validations: Payer, Participant name, position account & currency, amount, condition & expiry, DFSP
6 Lookup Participant data by name, account type (POSITION) & currency
Retrieve Participant data from database, when not found in cache
Retrieved Participant data cached
7 Pending Transfer requested via the CL-TB interface
8 Transfer update to cluster
9 Replicate across cluster
10 Return result
11 Return result
12 Return result
13 Return result
14 Return result with transferId
15 [ASYNC] transferDuplicateCheck DB record created
16 [ASYNC] Create database records: transfer, transferParticipant, ilpPacket, transferStateChange, transferExtension
17 [ASYNC] Database transaction committed

Interaction Detail - Prepare Transfer



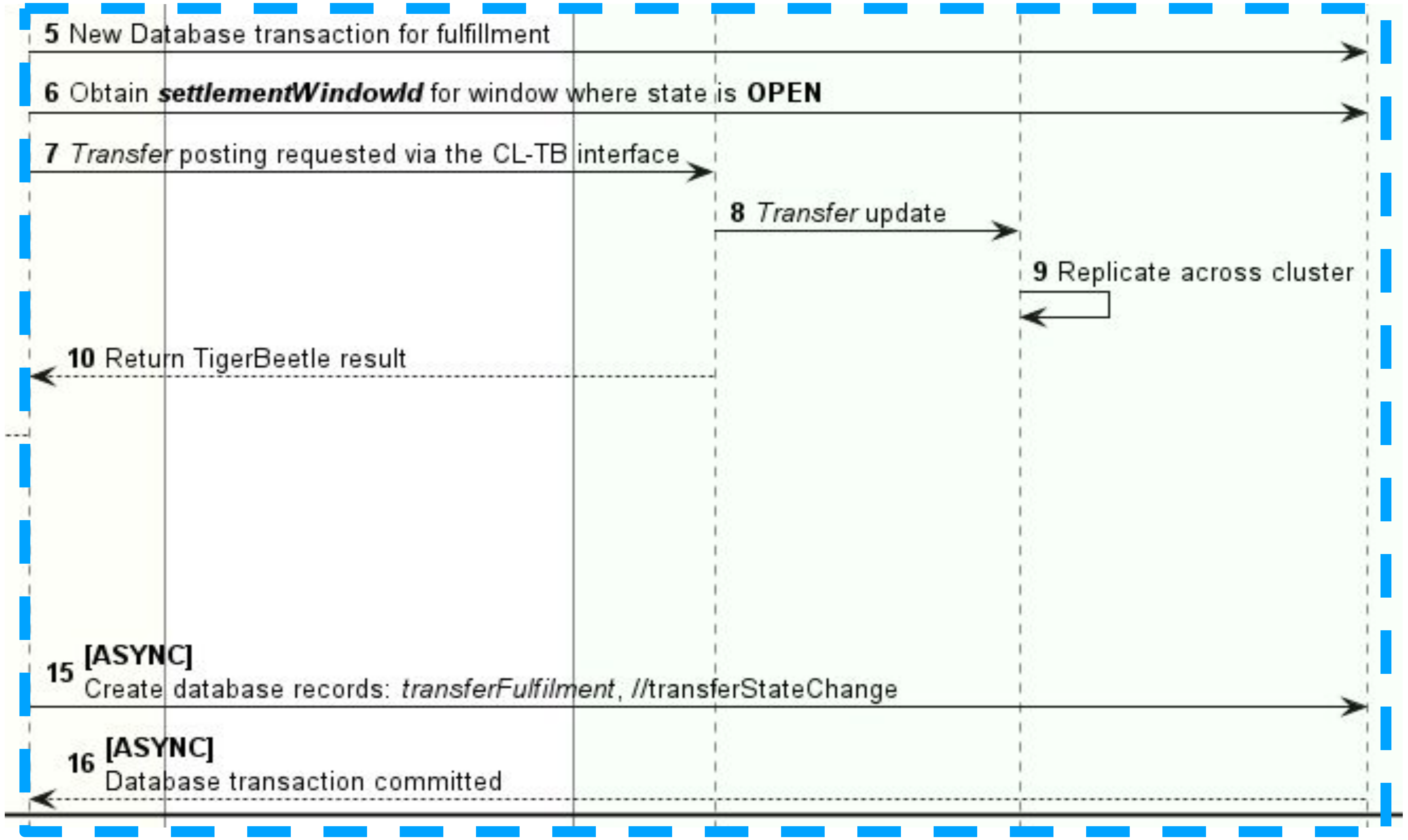
5 Database validations: *Payer, Participant name, position account & currency, amount, condition & expiry, DFSP*
6 Lookup Participant data by *name, account type* (POSITION) & *currency*
Retrieve Participant data from database, when not found in cache
Retrieved Participant data cached
7 Pending *Transfer* requested via the CL-TB interface
8 *Transfer* update to cluster
9 Replicate across cluster
10 Return result
15 [ASYNC] *transferDuplicateCheck* DB record created
16 [ASYNC] Create database records: *transfer, transferParticipant, ilpPacket, transferStateChange, transferExtension*
17 [ASYNC] Database transaction committed

Interaction - Fulfil Transfer



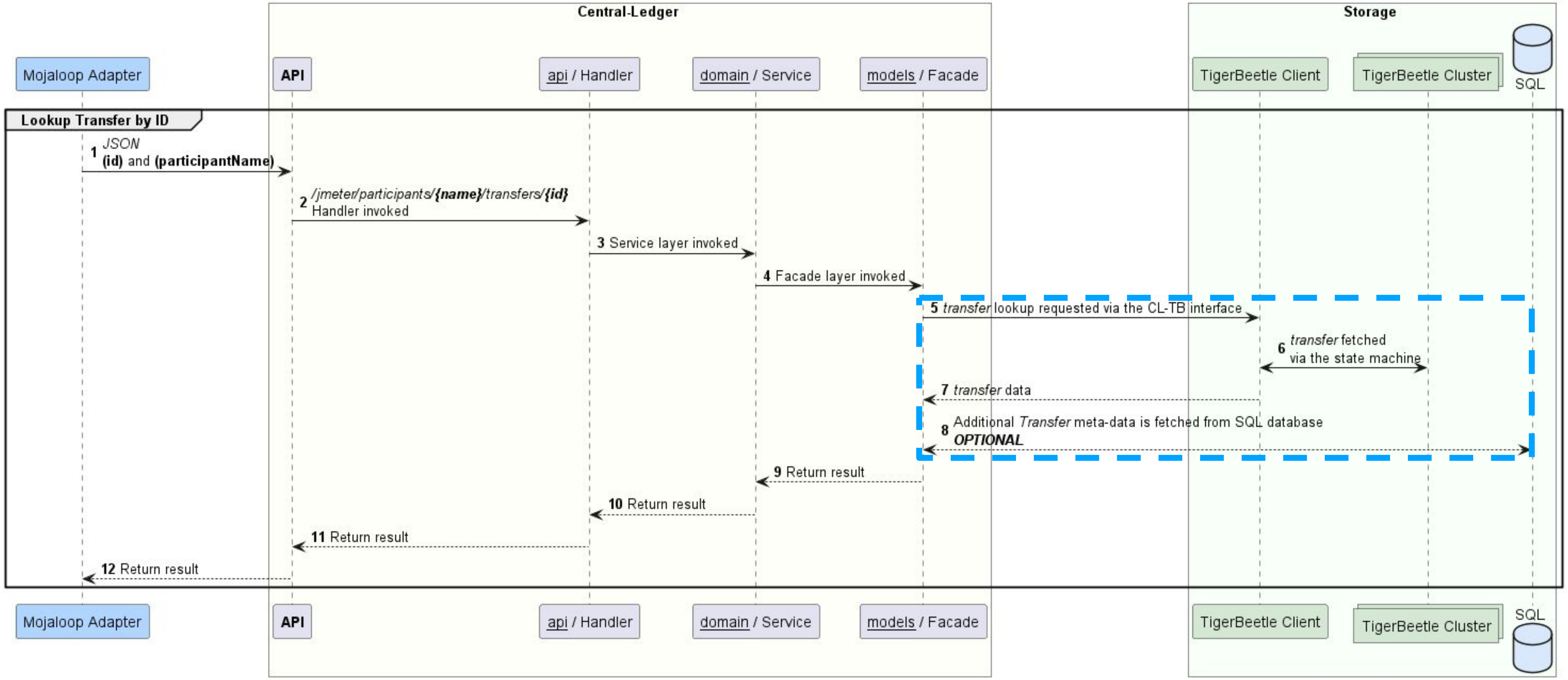
Central-Ledger
Storage
Mojaloop Adapter
API
api / Handler
domain / Service
models / Facade
TigerBeetle Client
TigerBeetle Cluster
SQL / PostgreSQL
Redis
Create Transfer (2-Phase)
Transfer Fulfilment
1 JSON (fulfil transfer data with ***transferId***)
2 */jmeter/transfers/fulfil* Handler invoked.
3 Service layer invoked *handlePayeeResponse*
4 Domain to facade *savePayeeTransferResponse*
5 New Database transaction for fulfillment
6 Obtain ***settlementWindowId*** for window where state is **OPEN**
7 *Transfer* posting requested via the CL-TB interface
8 *Transfer* update
9 Replicate across cluster
10 Return TigerBeetle result
11 Return result
12 Return result
13 Return result
14 Return result
15 **[ASYNC]** Create database records: *transferFulfilment*, //transferStateChange
16 **[ASYNC]** Database transaction committed

Interaction Detail - Fulfil Transfer

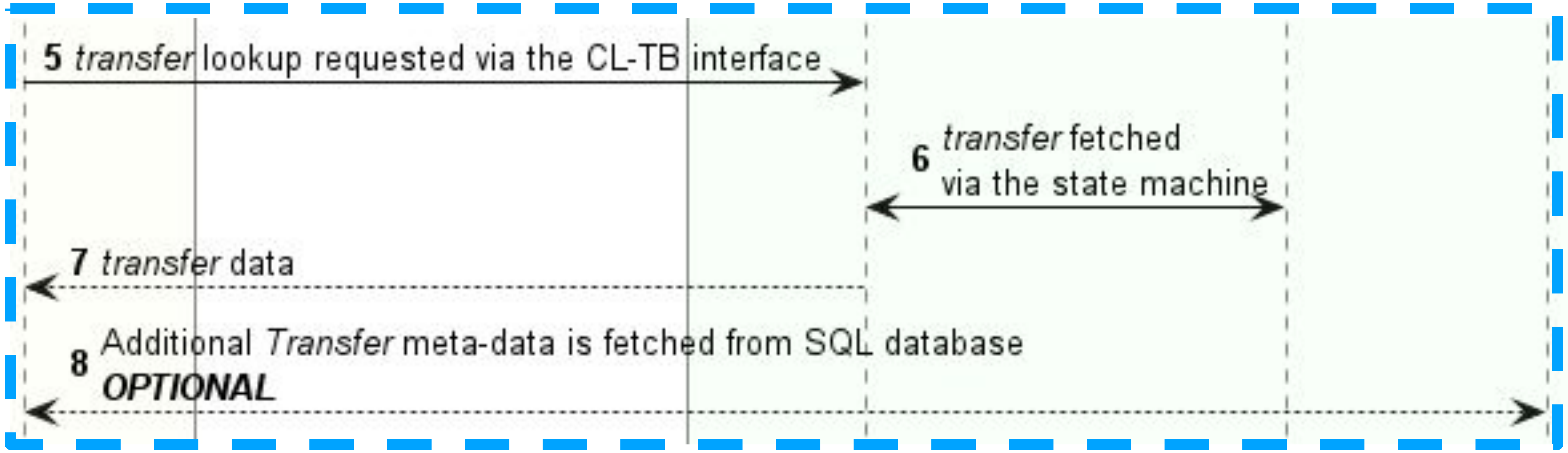


5 New Database transaction for fulfillment
6 Obtain ***settlementWindowId*** for window where state is **OPEN**
7 *Transfer* posting requested via the CL-TB interface
8 *Transfer* update
9 Replicate across cluster
10 Return TigerBeetle result
15 [ASYNC] Create database records: *transferFulfilment*, //transferStateChange
16 [ASYNC] Database transaction committed

Interaction - Lookup Transfer



Interaction Detail - Lookup Transfer



5 *transfer* lookup requested via the CL-TB interface
6 *transfer* fetched via the state machine
7 *transfer* data
8 Additional *Transfer* meta-data is fetched from SQL database ***OPTIONAL***

Data Migration

1. Default scripts: summary; migrate; verify
2. Process:
    a. Execute data summary scripts - accounts, balances & transfers
    b. Configure & execute migration scripts
    c. Execute verification scripts

Upcoming focus

For the next cycle	1. Design documentation community input → update & finalise 2. Implement & showcase Central-Settlement integration 3. Implement rich queries (batch, date range, entities) 4. Release JMeter Central-Ledger test suite 5. Demo integration

Questions

COMMUNITY
mojaloop
MEETING

Thank you.