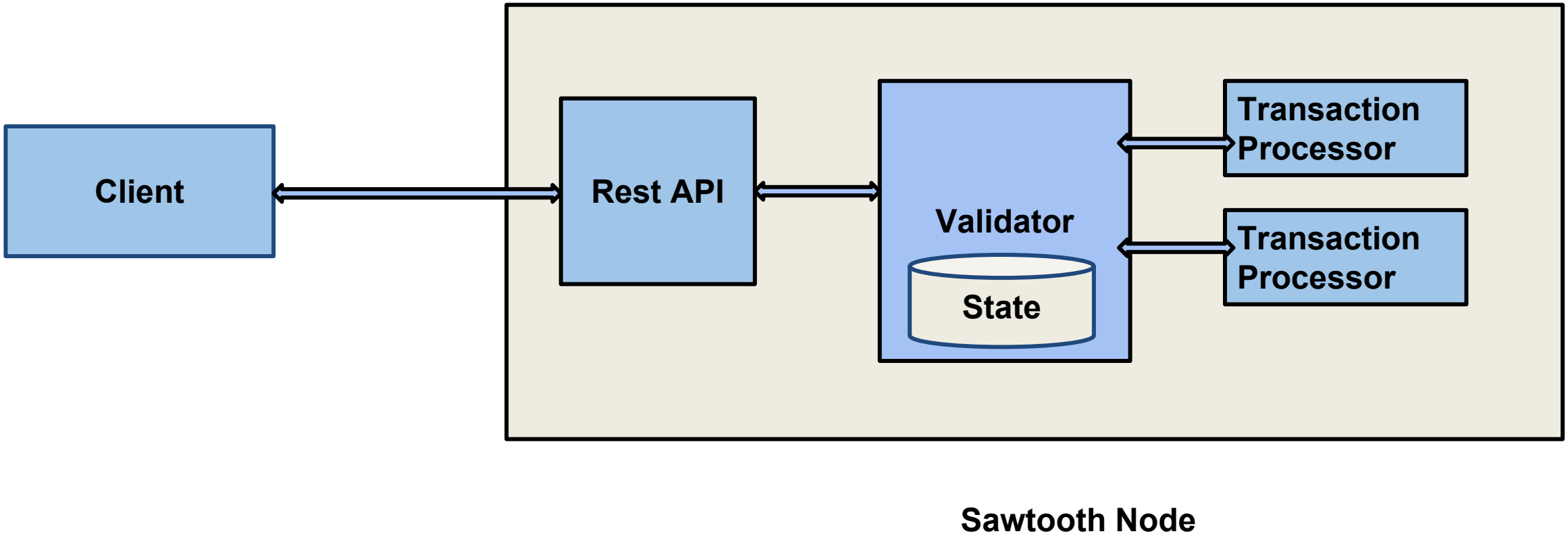
Implementing a Transaction Processor

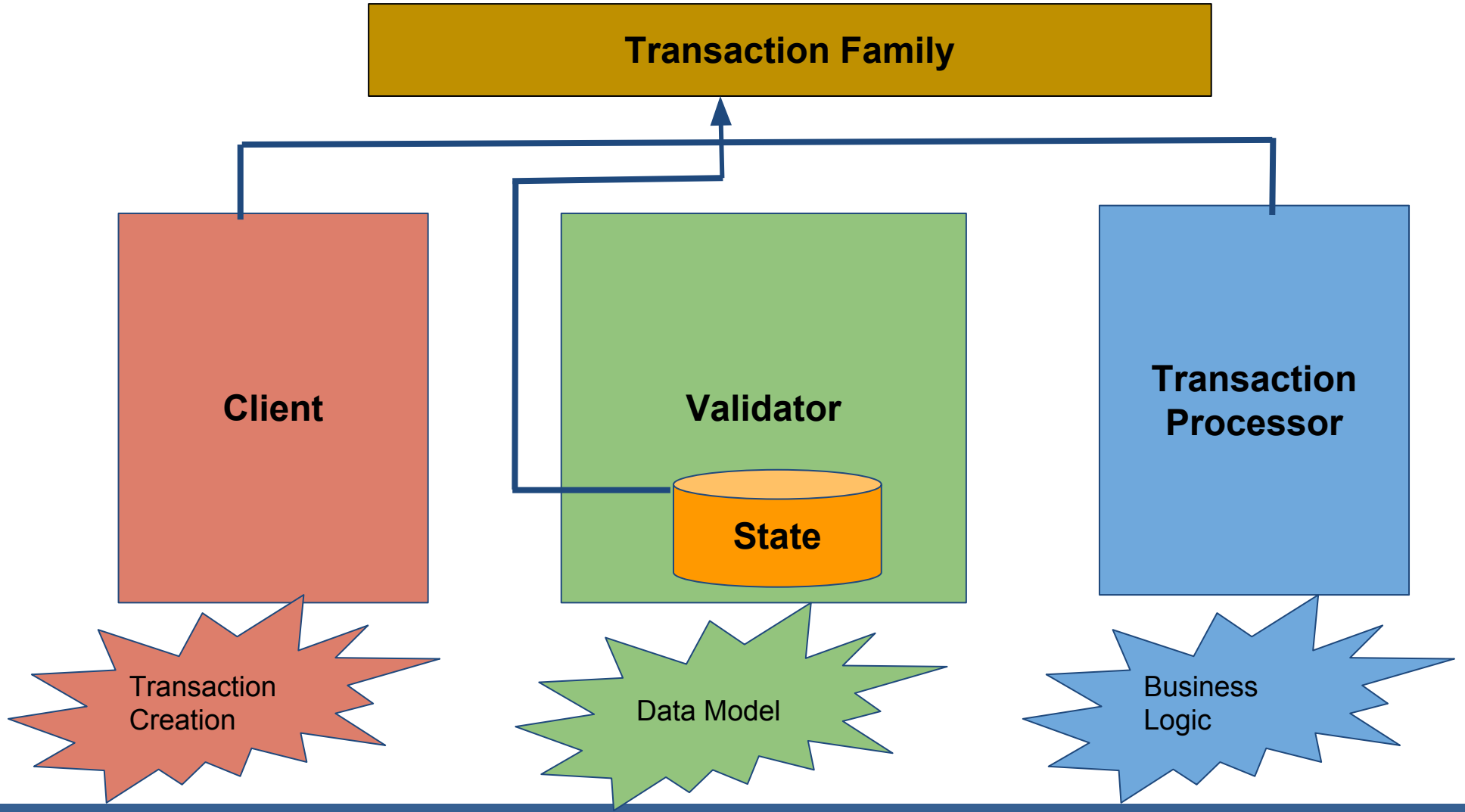
Sawtooth Architecture



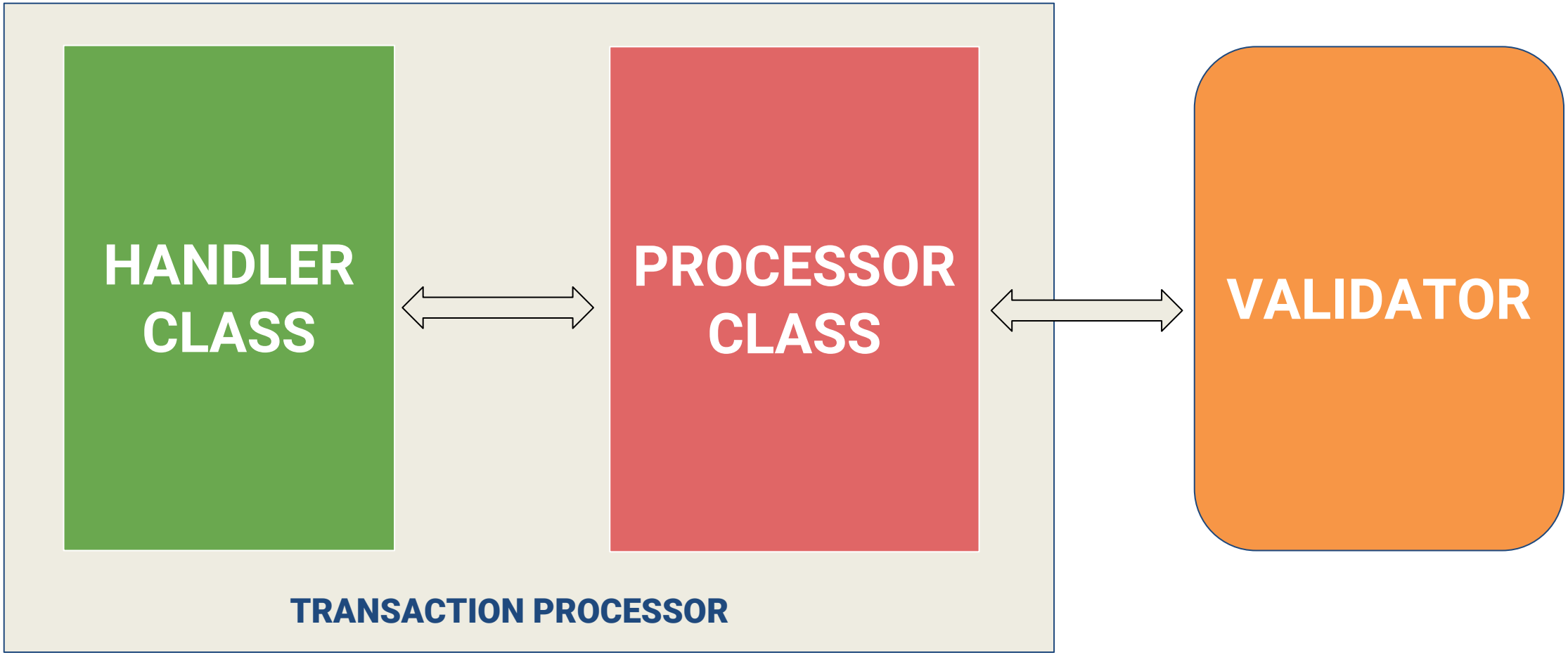
Implementing a TP

- Transaction Family
- What is a Transaction Processor?
- Components of a Transaction Processor
- Handler class
- Apply Method
- TransactionProcessRequest
- Transaction
- Context
- Processor class

Transaction Family



Transaction Processor



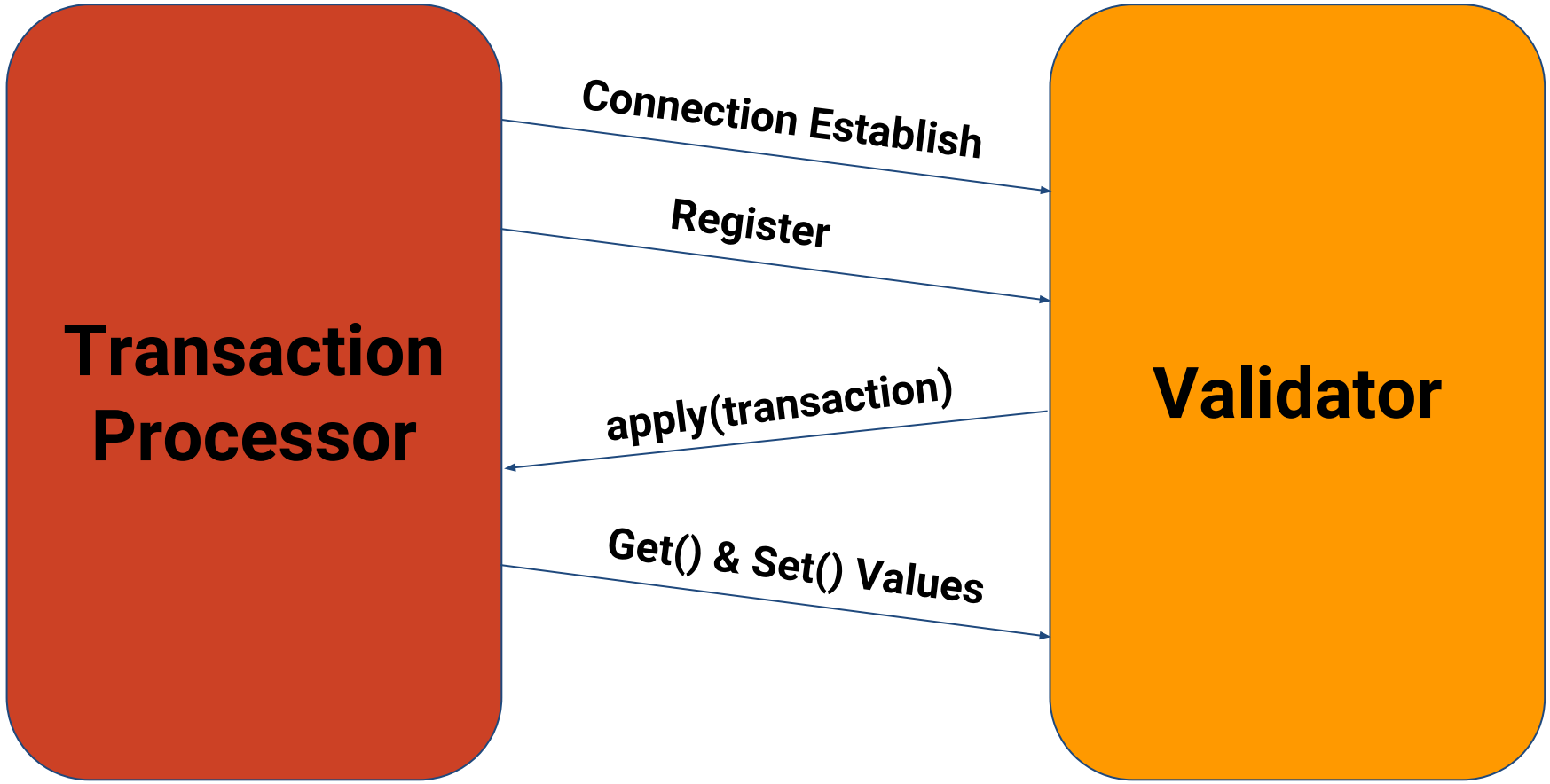
Components of a TP

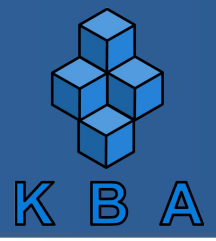
- There are two top-level components of a transaction processor.

1. **Processor class**
2. **Handler Class**

- The SDK provides a general-purpose processor class.

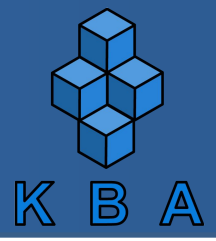
Transaction Processor





Handler Class

- Handler is the interface that defines the business logic for a new transaction family.
- This is the only interface that needs to be implemented to create a new transaction family.

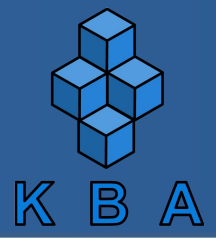


Handler Class

Handler class contains:

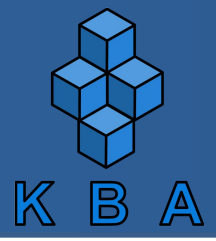
1. A constructor

2. An apply method



The Handler class

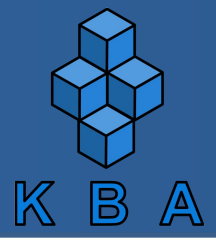
- The constructor uses “super” keyword to register a handler class with a validator by sending it information about what kinds of transactions it can handle.



Handler Class

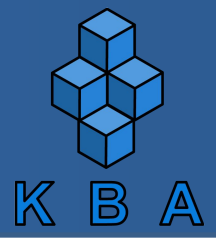
```
class Handler extends TransactionHandler {
  constructor () {
    super(FAMILY_NAME, '1.0', [NAMESPACE])
  }

  apply (transactionProcessRequest, context) {
    //
  }
}
```



Apply Method

- Apply is the single method that defines all the business logic for a transaction family.
- The method will be called by the transaction processor upon receiving a TpProcessRequest(ie; transaction).



Apply Method

- Apply gets called with two arguments, transactionProcessRequest and context.

- apply() method is called for each transaction.
- apply() is the execution entry point to Transaction processor.

- transactionProcessRequest holds the valid transactions send by the client .

- context stores information about the current state of the Tp .

TransactionProcessRequest

- The transaction contains payload bytes that are not transparent to the validator core, and transaction family specific.
- The transaction consists of a header and a payload.
- The header contains the signer’s public key, which can be used to identify the current transactor and the Address generation.

Payload

- The payload will contain data(application specific).

- If the payload is Invalid, throw an *InvalidTransaction* error to be caught and logged by the sdk.

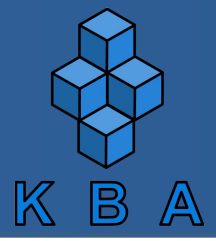
Converting Payload back from Bytes

- The byte converted payload is sent across the network from the client module to the transaction processor.
- This is performed to make sure that the same byte array is sent across network.

```
const {TextDecoder} = require('text-encoding/lib/encoding')

var dec = new TextDecoder('utf8');

var msg = dec.decode(transactionProcessRequest.payload);
```



Context

- Context is an object. It provides the instance of current state.
- Context provides an interface for getting , setting and deleting validators state.
- All validator interactions by a handler should be through a Context instance.

getState()

getState()

- The getState() method takes an array of state addresses and returns an object with the fetched state.

getState(addresses, timeout=None)

Parameters: addresses (list) – the addresses to fetch
timeout – optional timeout, in seconds

setState()

setState()

- The setState() requests that each address in the provided dictionary be set in validator state to its corresponding value.

setState(entries, timeout=None)

Parameters:

entries (dict) – where addresses are the keys and data is the value.

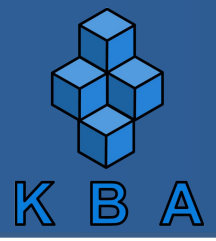
timeout – optional timeout, in seconds

deleteState()

deleteState()

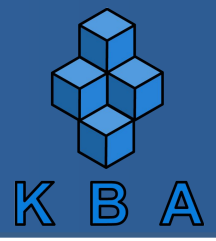
- deleteState() requests that each of the provided addresses be unset in validator state.

deleteState(addresses, timeout=None)



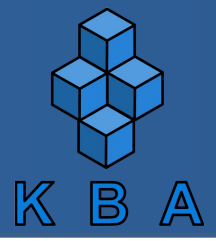
Error Classes

- *InvalidTransaction* thrown for an invalid transaction.
- *InternalError* thrown when an internal error occurs during transaction processing.
- *ValidatorConnectionError* thrown when a connection error occurs between the validator and transaction processor.
- *AuthorizationException* thrown when an authorization error occurs.



Processor Class

- TransactionProcessor is a generic class for communicating with a validator.
- It routes transaction processing requests to a registered handler class.
- Multiple handlers can be connected to an instance of the processor class.

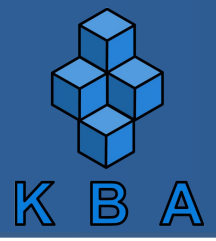


Processor Class

- The Transaction Processor receives the payload, decodes it and modifies the state to write the data.

- *addHandler* method will add the given handler to the Transaction Processor so it can receive specific transaction processing request.

- All handlers must be added before starting the processor.



Processor Class

```
const { TransactionProcessor } = require('sawtooth-sdk/processor')

const handlername = require('./handlername')

const address = 'tcp://validator:4004'

const transactionProcessor = new TransactionProcessor(address);

transactionProcessor.addHandler(new handlername());

transactionProcessor.start();
```

Points to remember

- All validators must run the same transaction processors that are on the network.

- Any number of Transaction processors can run in a sawtooth network.

- Every transaction processor must be registered on every validator in the sawtooth network

THANK YOU