sawtooth

Transaction Processing Architecture

April 19, 2017

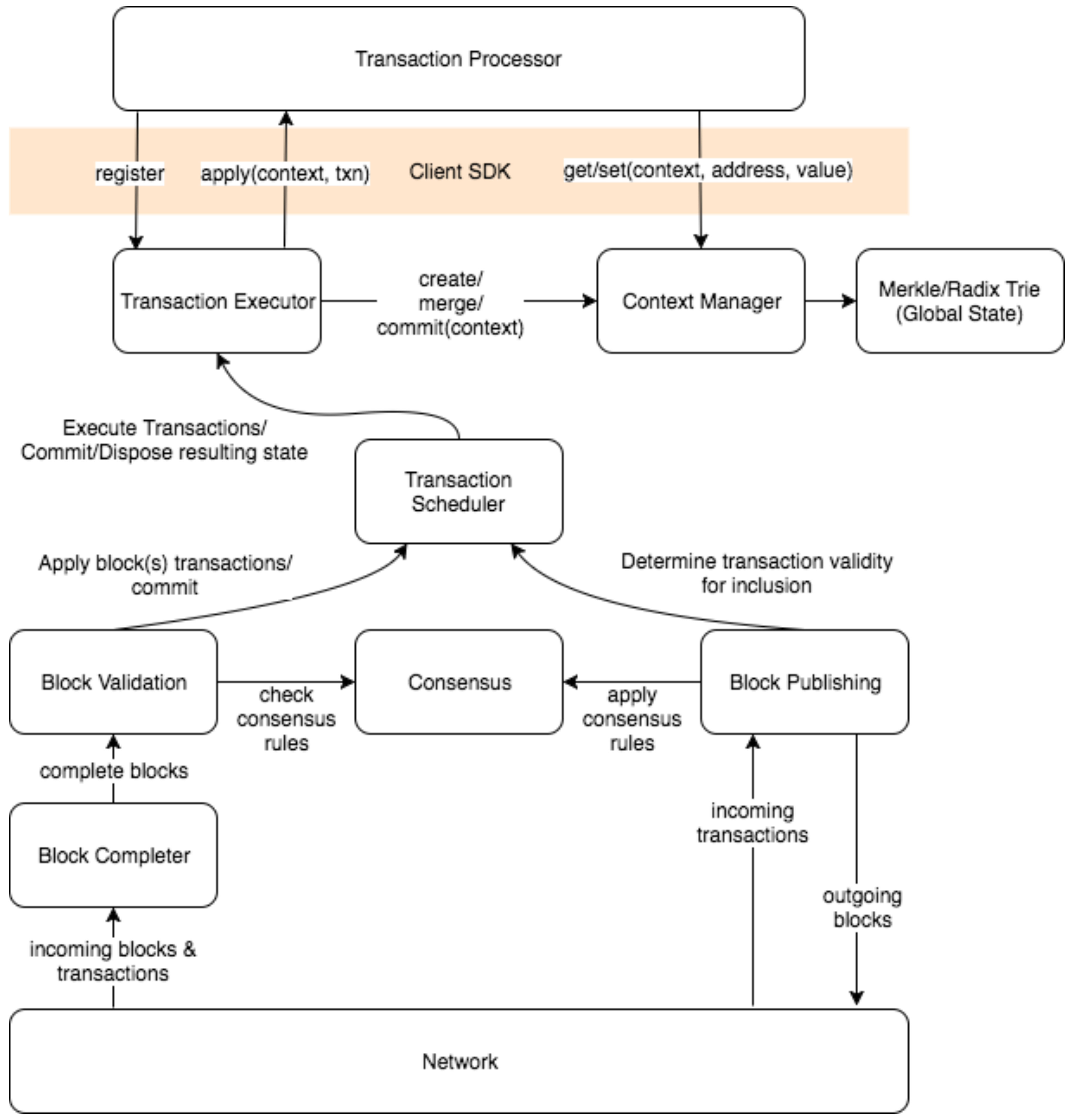
Modular Design

- Sawtooth Lake is a ***platform*** for developing distributed ledgers.
- Takes *no position* on fundamental types/data structures (no assets or currency defined in the core).
- Clearly defined interfaces for pluggable consensus and transaction processing allow for different configurations for different use cases, for example:
    - Permissioned network with domain-specific non-Turing complete execution support via a business rules transaction processor.
    - Open network with a combination of ‘static’ transaction formats and on-chain bytecode using a ‘virtual machine’ implementation (like EVM).

sawtooth

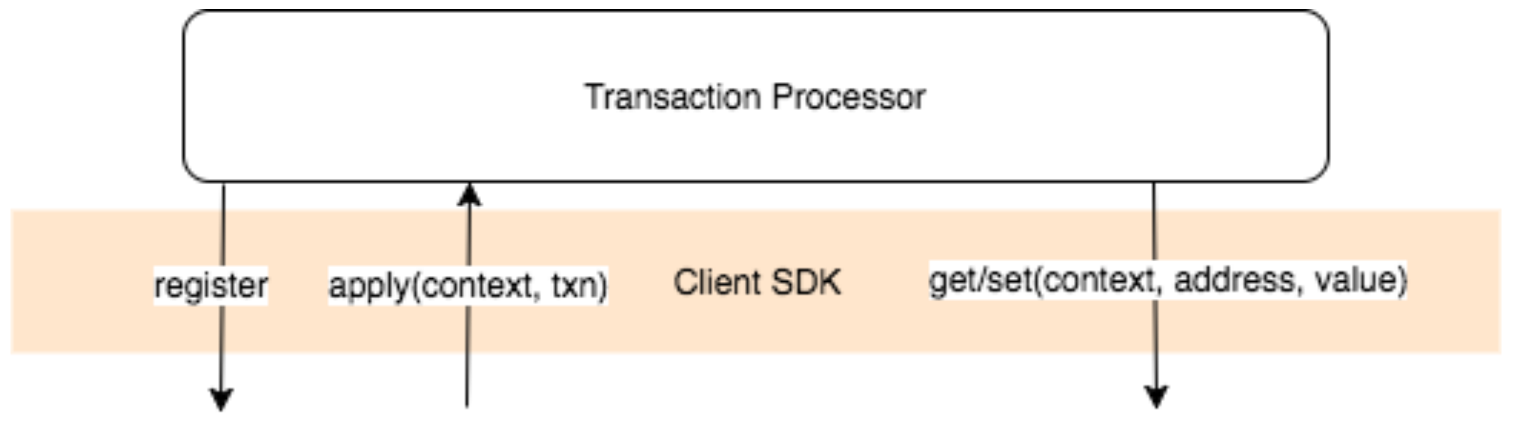
0.8 Architecture

- The ***core*** is responsible for:
  - Message handling
  - Block publishing/validation
  - Consensus
  - Global state management
- ***Pluggable*** consensus
- Parallel Transaction Processing

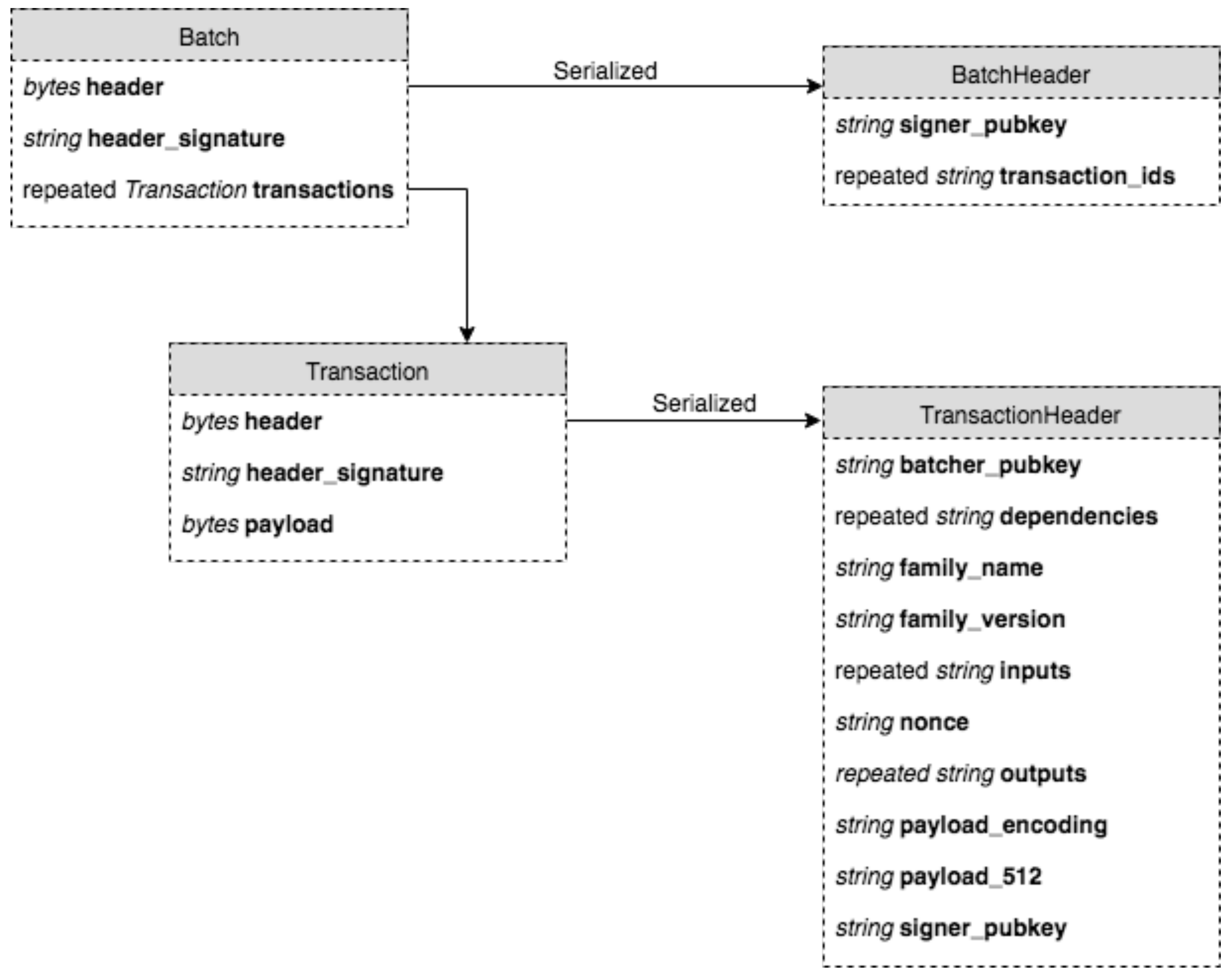


sawtooth

Transaction Processing



- The Transaction Processor interface defined by the Sawtooth Lake SDK is very lightweight.
- A transaction is executed in a ***context*** which is built by the validator. The context governs the starting version of state, and the state addresses which will be read from and written to as part of the transaction's execution.
- The validator calls the apply() method of the transaction handler and provides the context and the opaque payload of the transaction.
- The Transaction Processor is responsible for deserializing the payload and performing the necessary state transitions through get() and set() calls against the context.
- If the Transaction Processor attempts to reference state addresses outside of the declared inputs and outputs, an error will be returned by the validator.
- The validator is responsible for managing the isolation of transaction execution via explicit dependency ordering and declared inputs and outputs and for aggregating the contexts into block-level state transitions for publishing (state root hash calculation) and verification/application.



SDKs

- Sawtooth Lake SDKs provide lightweight interfaces for writing standalone Transaction Processors.
- The SDKs provide the 0mq and protobuf message definitions and framework to allow the new Transaction Processor to register with the validator and receive requests to process transactions.

python JS Java COMING SOON COMING SOON COMING SOON

sawtooth

```python
# Copyright 2016 Intel Corporation
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ------------------------------------------------------------------------------

import logging
import hashlib

import cbor

from sawtooth_sdk.processor.state import StateEntry
from sawtooth_sdk.processor.exceptions import InvalidTransaction
from sawtooth_sdk.processor.exceptions import InternalError


LOGGER = logging.getLogger(__name__)


class IntkeyTransactionHandler(object):
    def __init__(self, namespace_prefix):
        self._namespace_prefix = namespace_prefix

    @property
    def family_name(self):
        return 'intkey'

    @property
    def family_versions(self):
        return ['1.0']

    @property
    def encodings(self):
        return ['application/cbor']

    @property
    def namespaces(self):
        return [self._namespace_prefix]
```

```python
    def apply(self, transaction, state):
        content = cbor.loads(transaction.payload)

        (verb, name, value) = \
            (content['Verb'], content['Name'], content['Value'])

        if verb is None or len(verb) < 1:
            raise InvalidTransaction("Verb is required")
        if name is None or len(name) < 1:
            raise InvalidTransaction("Name is required")
        if value is None:
            raise InvalidTransaction("Value is required")

        if verb not in ['set', 'inc', 'dec']:
            raise InvalidTransaction("invalid Verb: '{}'".format(verb))

        address = self._namespace_prefix + hashlib.sha512(
            name.encode()).hexdigest()

        LOGGER.info(
            'processing: Verb=%s Name=%s Value=%s address=%s',
            verb,
            name,
            value,
            address)

        entries_list = state.get([address])
        state_value_rep = entries_list[0].data \
            if len(entries_list) != 0 else None

        if state_value_rep is not None:
            LOGGER.debug(
                'address received: %s=%s',
                address,
                state_value_rep)

        if verb in ['inc', 'dec'] and (state_value_rep is None):
            raise InvalidTransaction("inc/dec require existing value")

        if verb == 'set':
            state_value = None
            if state_value_rep is not None:
                state_value = cbor.loads(state_value_rep)
                if name in state_value:
                    raise InvalidTransaction(
                        "Verb was 'set', but already exists: "
                        "Name: {}, Value {}".format(name,
                                                    state_value.get(name))
                    )
```

```python
            if state_value is None:
                data = {}
            else:
                data = {k: v for k, v in state_value.iteritems()}

            data[name] = value
            addresses = list(state.set(
                [StateEntry(address=address, data=cbor.dumps(data))]
            ))
        elif verb == 'inc':
            state_value = cbor.loads(state_value_rep)
            if name not in state_value:
                raise InvalidTransaction(
                    "Verb was 'inc' but Name, {}, not in state.".format(name)
                )
            if int(value) < 0:
                raise InvalidTransaction(
                    "Verb was 'inc', but Value was negative: {}".format(value))
            state_value[name] = int(state_value[name]) + int(value)
            addresses = list(state.set(
                [StateEntry(address=address,
                            data=cbor.dumps(state_value))]
            ))
        elif verb == 'dec':
            state_value = cbor.loads(state_value_rep)
            if name not in state_value:
                raise InvalidTransaction(
                    "Verb was 'dec', but Name, {}, not in state.".format(name)
                )

            if int(state_value[name]) - int(value) < 0:
                raise InvalidTransaction(
                    "Verb was 'dec', but resulting value would be negative")
            state_value[name] = int(state_value[name]) - int(value)
            addresses = list(state.set(
                [StateEntry(
                    address=address,
                    data=cbor.dumps(state_value))]
            ))
        else:
            # This would be a programming error.
            raise InternalError('unhandled Verb')

        if len(addresses) == 0:
            raise InternalError("State Error.")
```

Key Factors in Transaction Processor Design

- Deterministic across time and space
- Transaction payload format and serialization/deserialization
- Radix address encoding within state namespace
- Data schema and serialization/deserialization for information stored at addresses

sawtooth

Get Involved

https://github.com/hyperledger/sawtooth-core

https://chat.hyperledger.org/channel/sawtoothlake

https://intelledger.github.io/0.8/

sawtooth