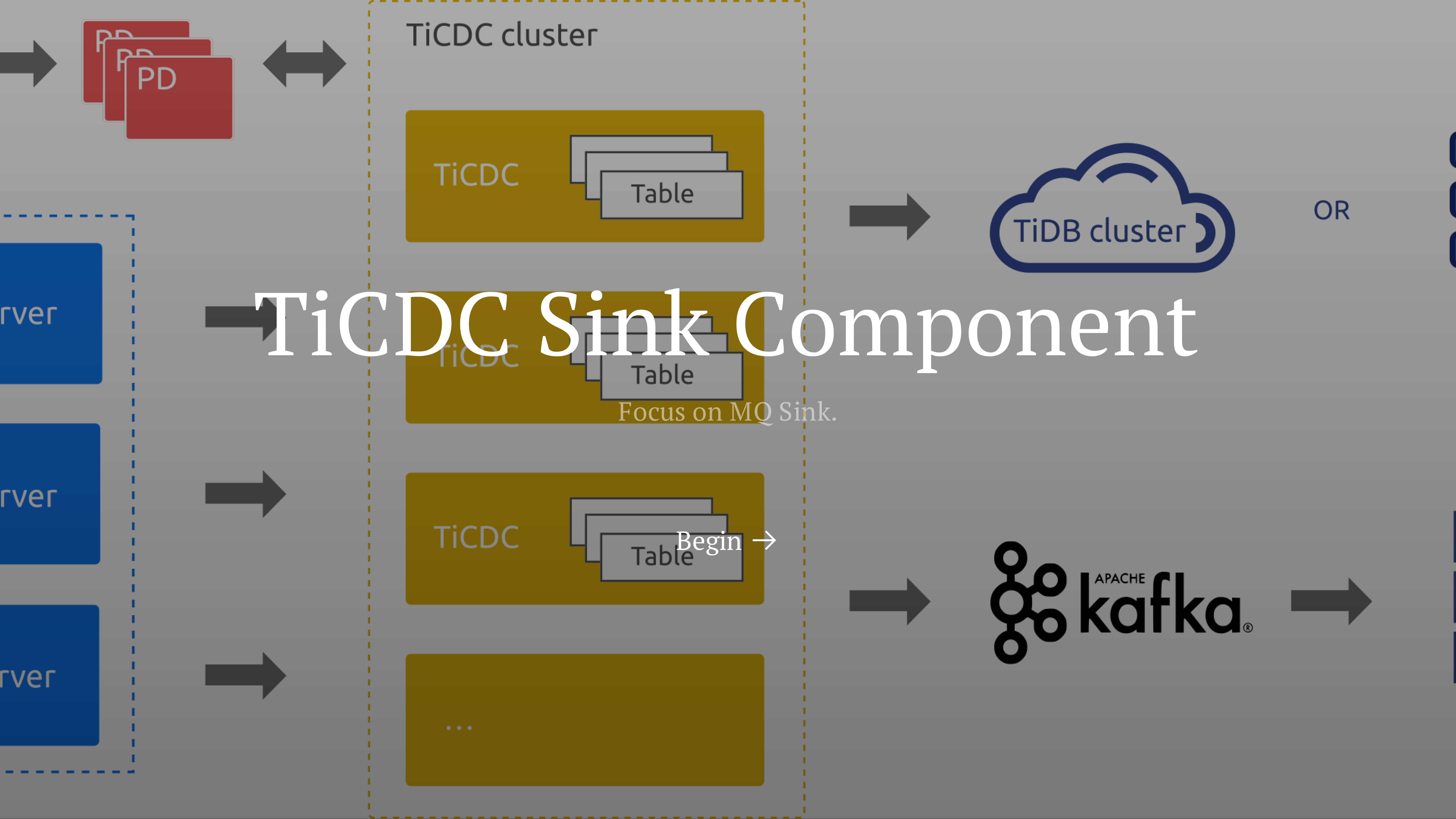


TiCDC Sink Component

Focus on MQ Sink.

Begin →

Architecture

- A TiCDC cluster has only one owner.
- A capture will have multiple processors.
- A processor can only process one changefeed.
- A changefeed can synchronize multiple tables.

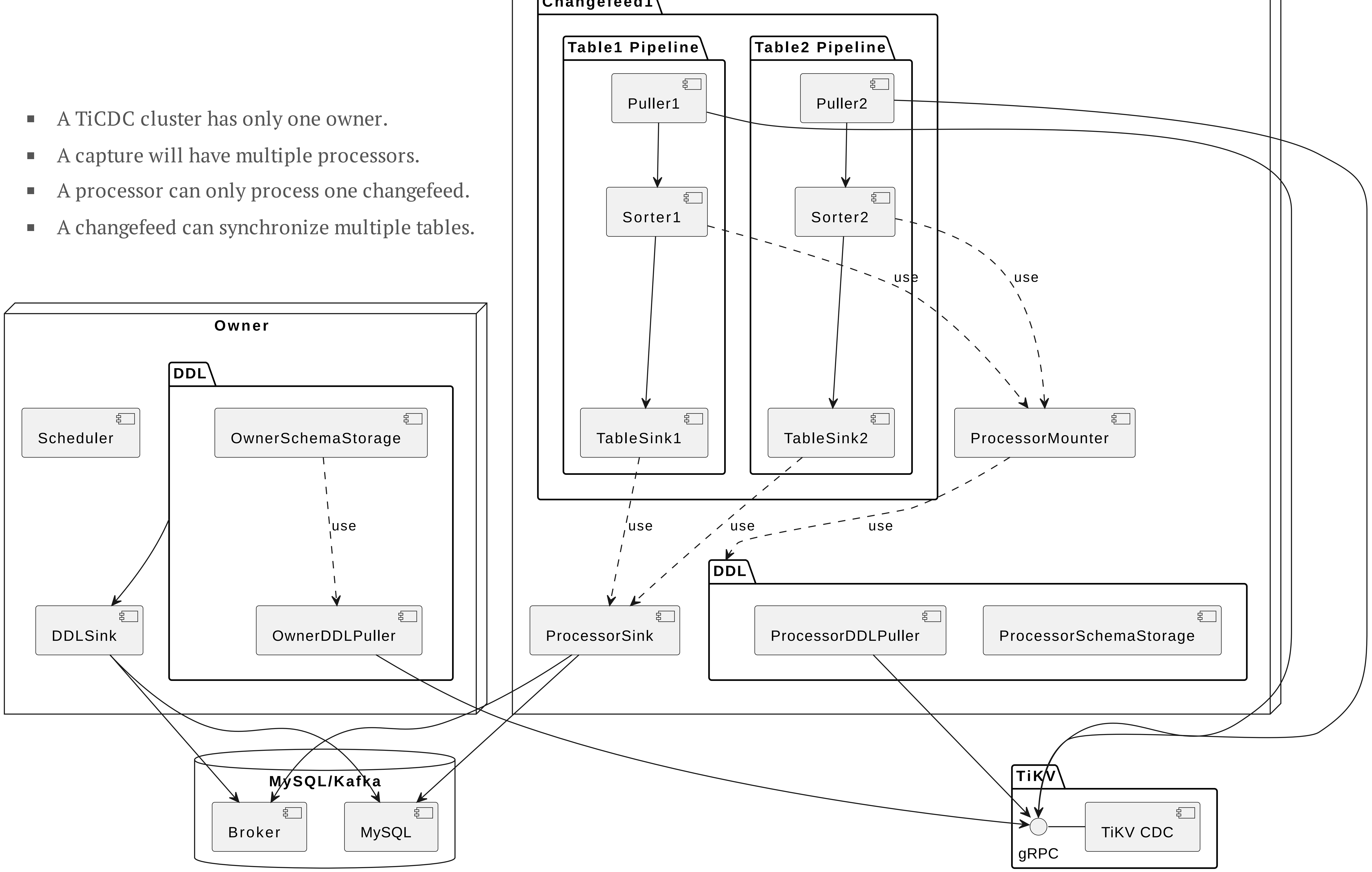
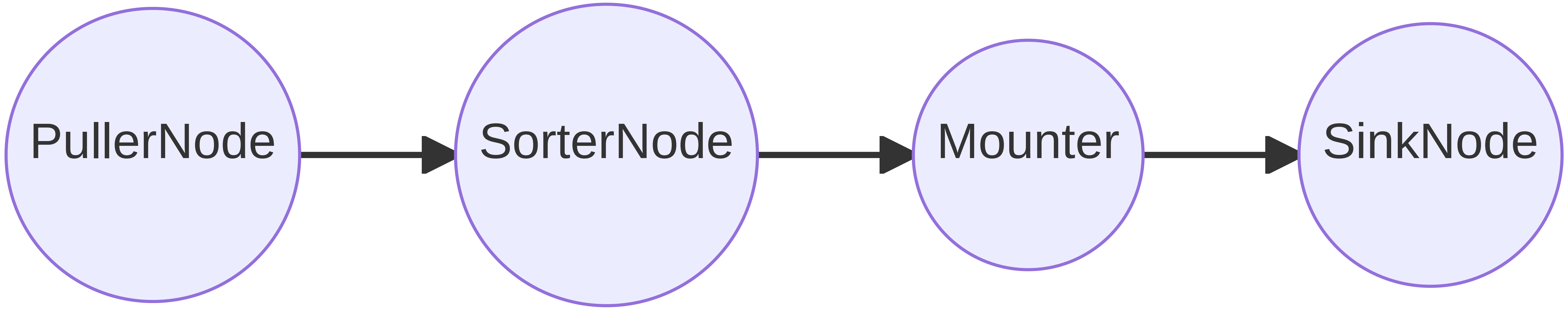


Table Pipeline

Each changefeed creates a processor, and each processor maintains multiple table pipelines.

Pipeline



Puller

Pull DDL and Row Change data from TiKV.

Region1	Region2
	ts1: C -> 2
ts2: A -> 6	ts1: Resolved
ts1: B -> 4	
ts1: Resolved	
ts2: B -> 3	ts2: C ->3
ts2: Resolved	
ts3: A -> 7	

Output Chan
ts1: C -> 2
ts2: A -> 6
ts1: B -> 4
ts1: Resolved
ts2: B ->3
ts2: C ->3
ts2: Resolved
ts3: A -> 7

Sorter

To Sort
ts1: C -> 2
ts2: A -> 6
ts1: B -> 4
ts1: Resolved
ts2: B ->3
ts2: C ->3
ts2: Resolved
ts3: A -> 7

Output Chan
ts1: C -> 2
ts1: B -> 4
ts1: Resolved
ts2: A -> 6
ts2: B ->3
ts2: C ->3
ts2: Resolved

Mounter

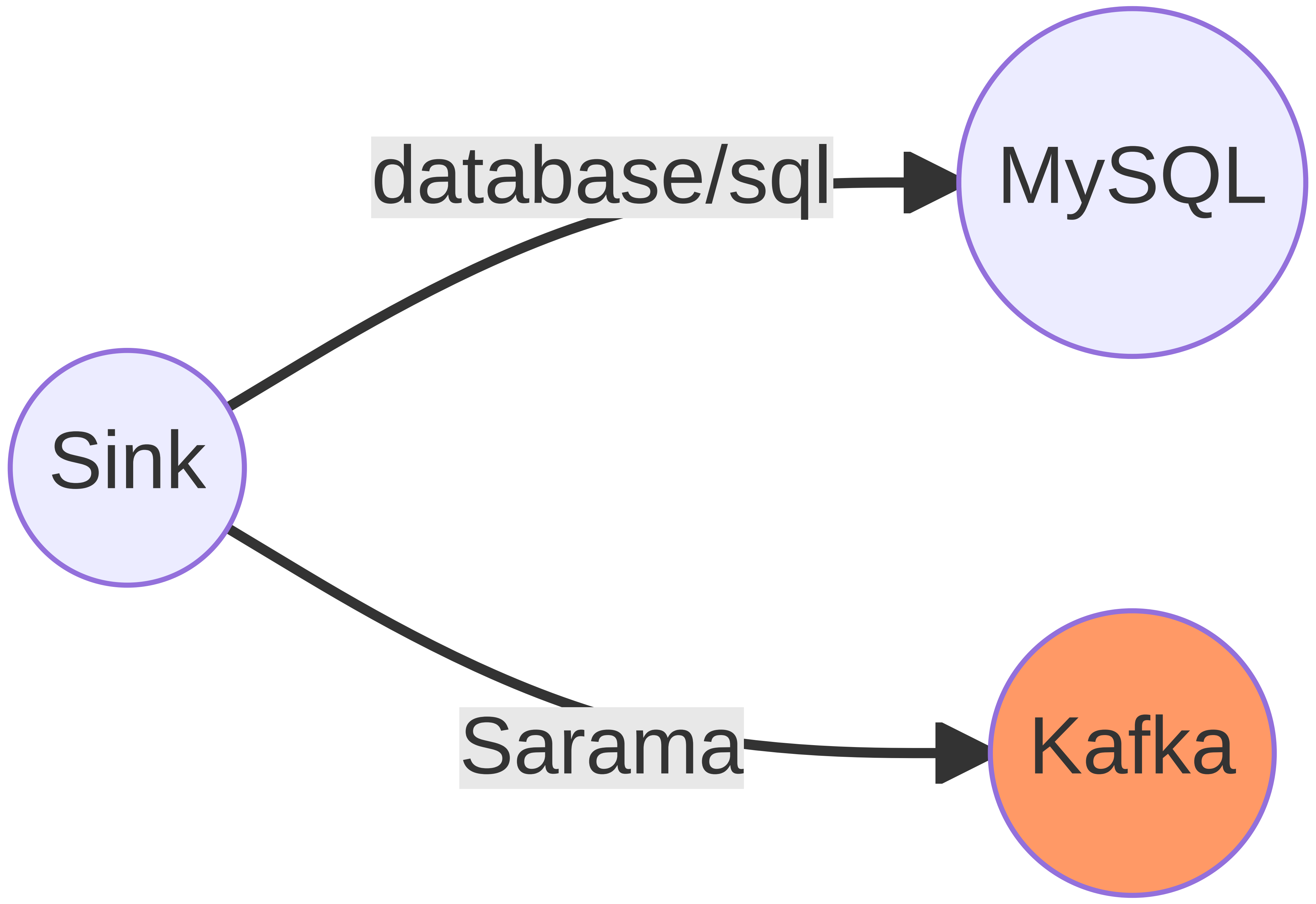
Mounter will use the schema information to convert the row kv into row changes that TiCDC can handle.

```go
type RawKVEntry struct {
    OpType OpType
    Key    []byte
    // nil for delete type
    Value []byte
    // nil for insert type
    OldValue []byte
    StartTs  uint64
    // Commit or resolved TS
    CRTs uint64
    // Additional debug info
    RegionID uint64
}
```

```go
type RowChangedEvent struct {
    StartTs  uint64
    CommitTs uint64
    RowID int64
    Table    *TableName
    ColInfos []rowcodec.ColInfo
    TableInfoVersion uint64
    ReplicaID    uint64
    Columns      []*Column
    PreColumns   []*Column
    IndexColumns [][]int
    ApproximateDataSize int64
}
```

Sink

Sink is responsible for sending data to MySQL or Kafka.



Sink Interface

```go
type Sink interface {
    EmitRowChangedEvents(ctx context.Context, rows ...*model.RowChangedEvent) error

    EmitDDLEvent(ctx context.Context, ddl *model.DDLEvent) error

    FlushRowChangedEvents(ctx context.Context, tableID model.TableID, resolvedTs uint64) (uint64, error)

    // Only for MQ Sink.
    EmitCheckpointTs(ctx context.Context, ts uint64, tables []model.TableName) error

    Close(ctx context.Context) error

    // Only for MySQL Sink.
    Barrier(ctx context.Context, tableID model.TableID) error
}
```

Sink Implement

Owner Level Sink

- DDL Sink: Sync DDL

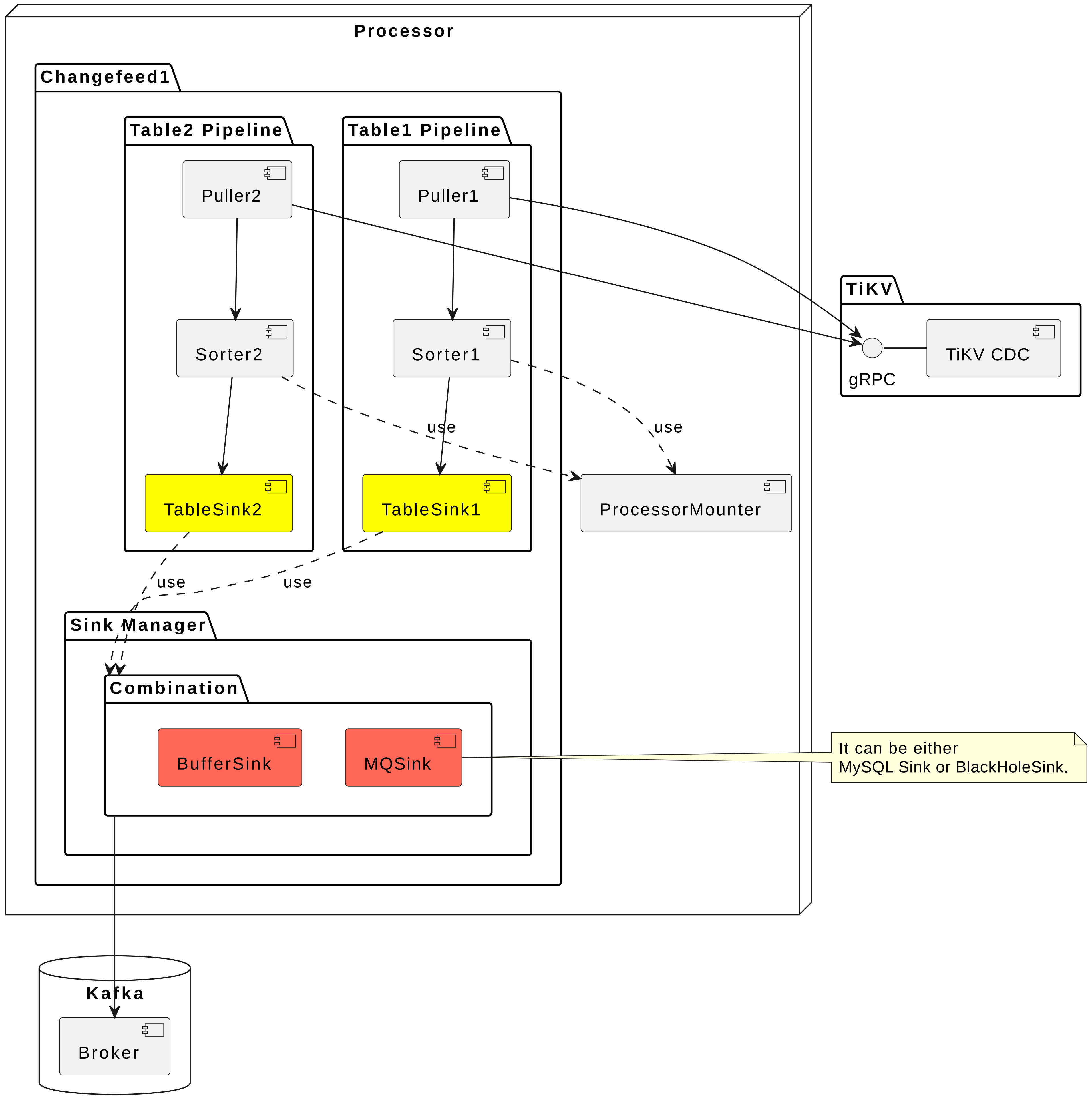
Processor Level Sink

- BlackHole Sink: Do nothing
- MQSink: For MQ
- MySQLSink: For MySQL
- Buffer Sink: Buffer + Asynchronously

Table Level Sink

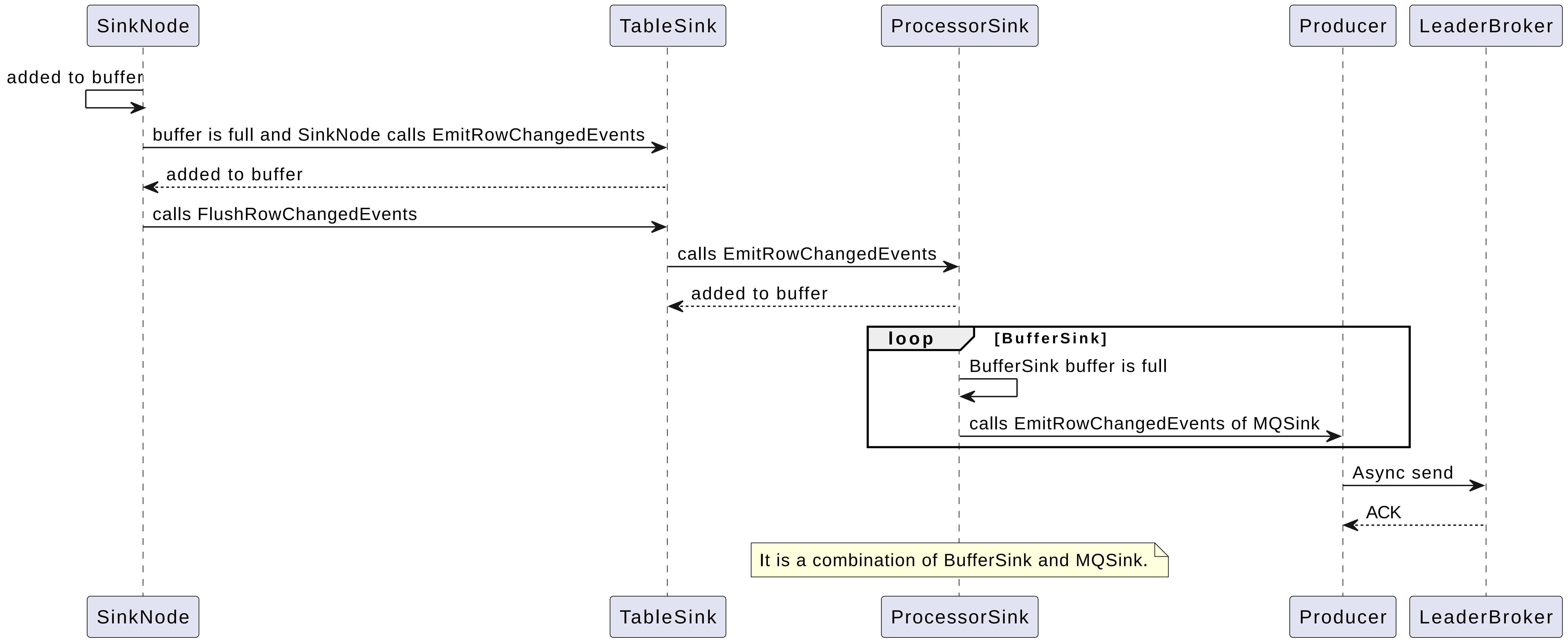
- Table Sink: Sink Minimum Management Unit

Relations


Processor
Changefeed1
Table2 Pipeline
Table1 Pipeline
Puller2
Puller1
Sorter2
Sorter1
TableSink2
TableSink1
use
use
use
use
ProcessorMounter
TiKV
gRPC
TiKV CDC
Sink Manager
Combination
BufferSink
MQSink
It can be either
MySQL Sink or BlackHoleSink.
Kafka
Broker

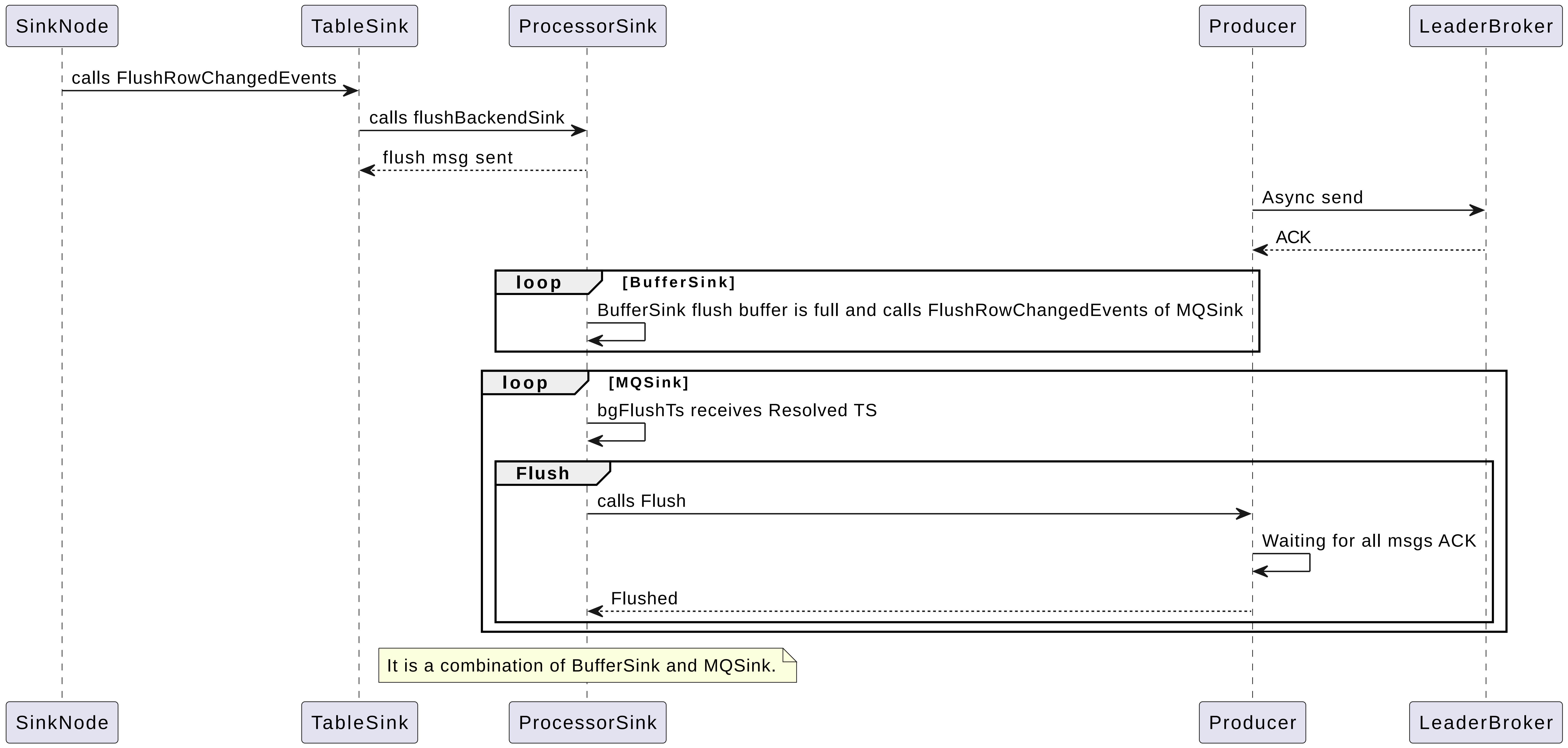
Data Sequence

Row Change Data Sequence

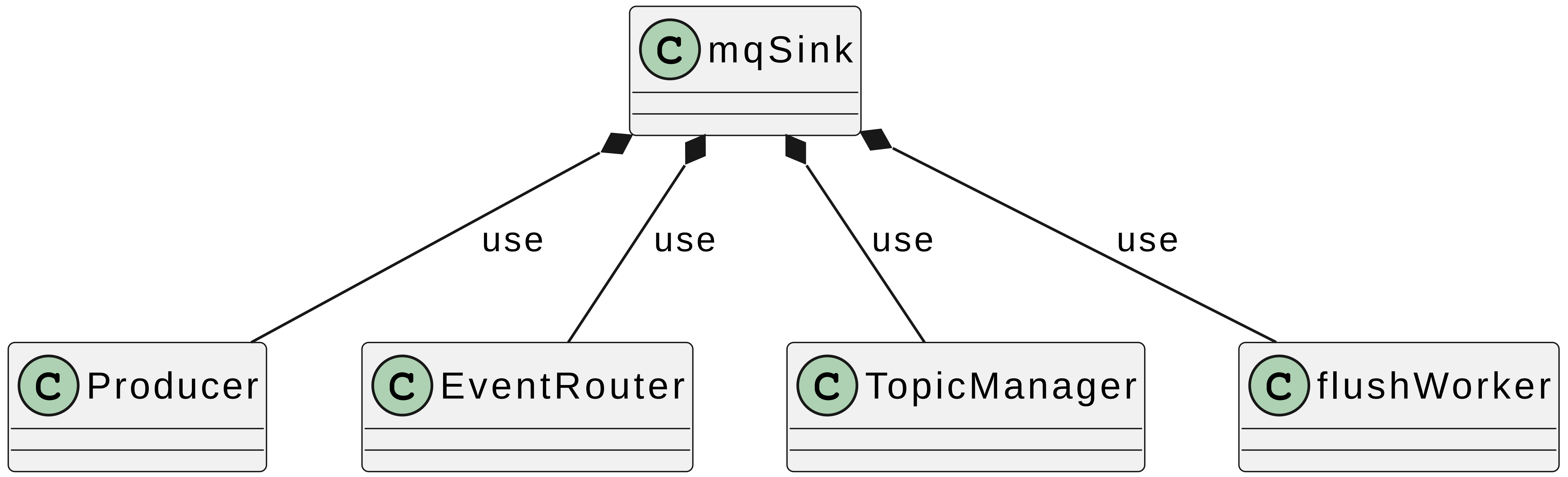

SinkNode
TableSink
ProcessorSink
Producer
LeaderBroker
added to buffer
buffer is full and SinkNode calls EmitRowChangedEvents
added to buffer
calls FlushRowChangedEvents
calls EmitRowChangedEvents
added to buffer
loop
[BufferSink]
BufferSink buffer is full
calls EmitRowChangedEvents of MQSink
Async send
ACK
It is a combination of BufferSink and MQSink.
SinkNode
TableSink
ProcessorSink
Producer
LeaderBroker

Data Sequence

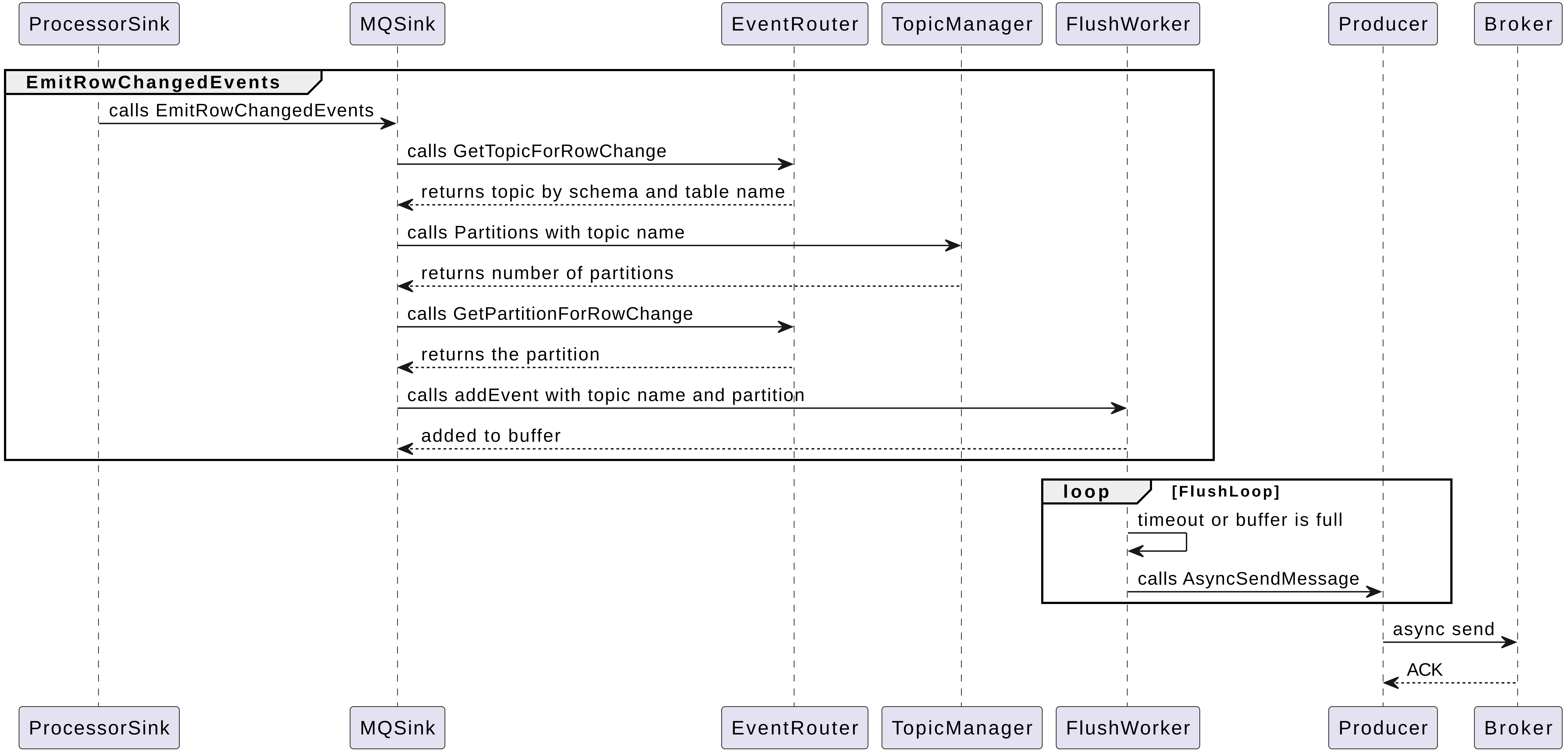
Resolved TS Flush Sequence



MQ Sink

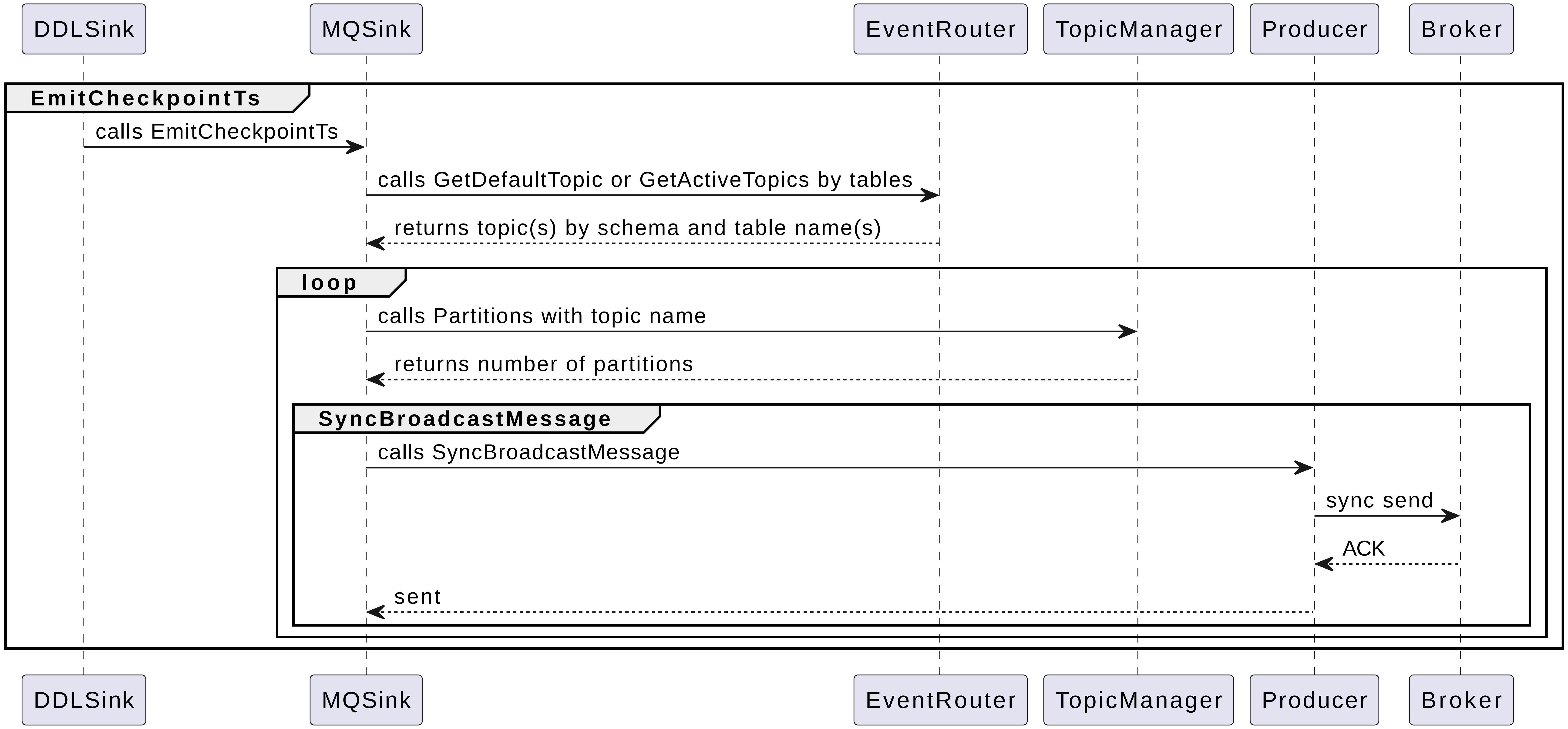

mqSink
use
use
use
use
Producer
EventRouter
TopicManager
flushWorker

Row Change Data Sequence


ProcessorSink
MQSink
EventRouter
TopicManager
FlushWorker
Producer
Broker
EmitRowChangedEvents
calls EmitRowChangedEvents
calls GetTopicForRowChange
returns topic by schema and table name
calls Partitions with topic name
returns number of partitions
calls GetPartitionForRowChange
returns the partition
calls addEvent with topic name and partition
added to buffer
loop
[FlushLoop]
timeout or buffer is full
calls AsyncSendMessage
async send
ACK
ProcessorSink
MQSink
EventRouter
TopicManager
FlushWorker
Producer
Broker

Resolved TS Sequence

Checkpoint TS Sequence


DDLSink
MQSink
EventRouter
TopicManager
Producer
Broker
EmitCheckpointTs
calls EmitCheckpointTs
calls GetDefaultTopic or GetActiveTopics by tables
returns topic(s) by schema and table name(s)
loop
calls Partitions with topic name
returns number of partitions
SyncBroadcastMessage
calls SyncBroadcastMessage
sync send
ACK
sent
DDLSink
MQSink
EventRouter
TopicManager
Producer
Broker

Code View

Reference

- TiCDC Architecture
- TiCDC multi topic support spec
- Kafka Producer topic support design