Developing Applications Using Kinesis Client Library

Ivan Mushketyk

@mushketyk brewing.codes

What Are We Going to Implement

Tweets

mushketyk 13:45

The weather is nice!

anonymous 13:46

@mushketyk Stop procrastinating!

anonymous 13:46

@mushketyk Go record a course already!

language	time	count
en	13:45	1
en	13:46	2

How Are We Going to Implement It

Kinesis Producer Library

Why KPL?

Simple API

Better performance

Lower cost

Errors handling

Monitoring

Writing Data to Kinesis

```
KinesisProducerConfiguration config = new
KinesisProducerConfiguration();

KinesisProducer = new KinesisProducer(config);

kinesisProducer
    .addUserRecord(
          "stream",
          partitionKey,
          recordBytes);
```

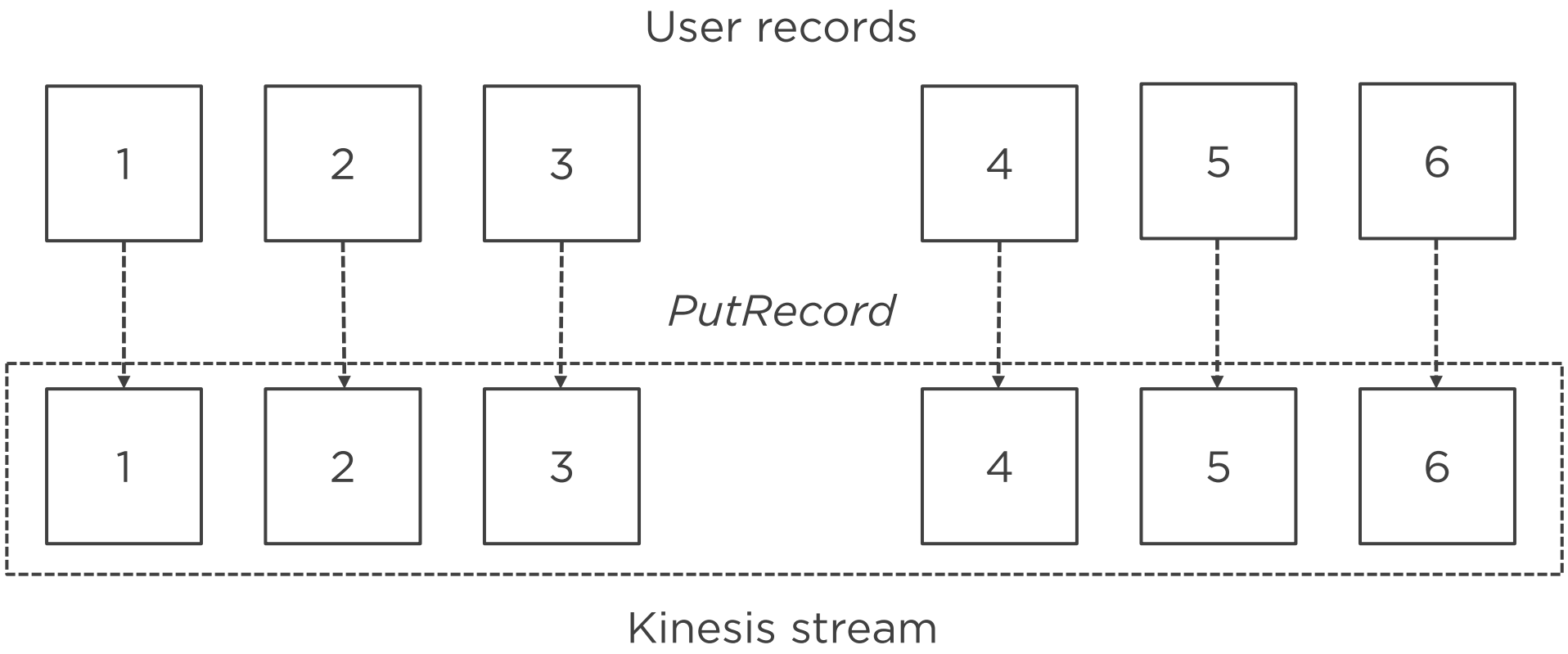
Writing Data to Kinesis

```
KinesisProducerConfiguration config = new
KinesisProducerConfiguration();

KinesisProducer = new KinesisProducer(config)

ListenableFuture<UserRecordResult> f = kinesisProducer
    .addUserRecord(
          "stream",
          partitionKey,
          recordBytes);
```

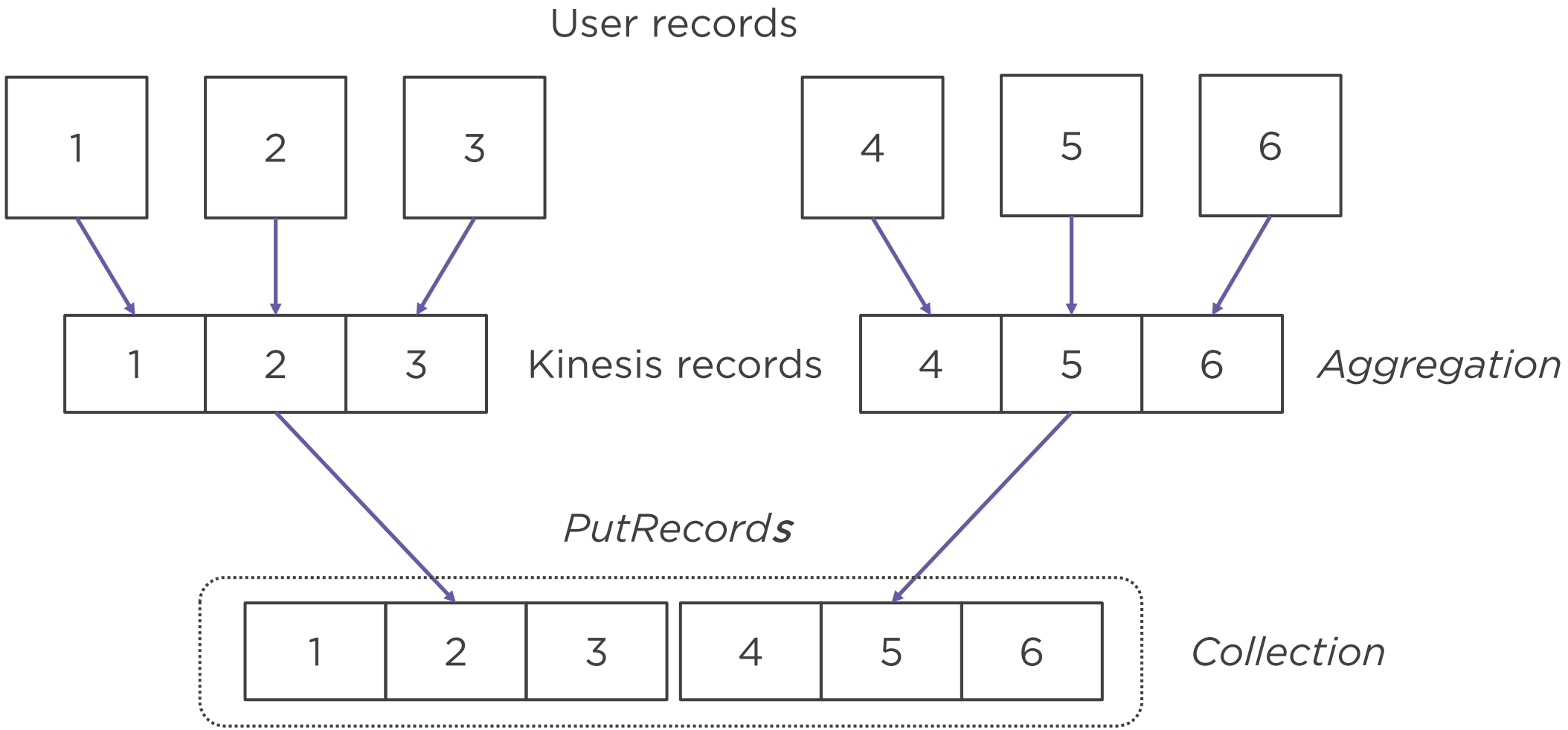
Process Future

```
ListenableFuture<UserRecordResult> f = …
// Synchronously
UserRecordResult result = f.get();
// Asynchronously
Futures.addCallback(f, new FutureCallback<UserRecordResult>() {
    public void onFailure(Throwable t) {…}
    public void onSuccess(UserRecordResult result) {…}
});
```

Storing Records



Storing Records



Records Aggregation

Up to 1000 records per shard per second

1M PutRecord calls costs $0.014

Can be tricky

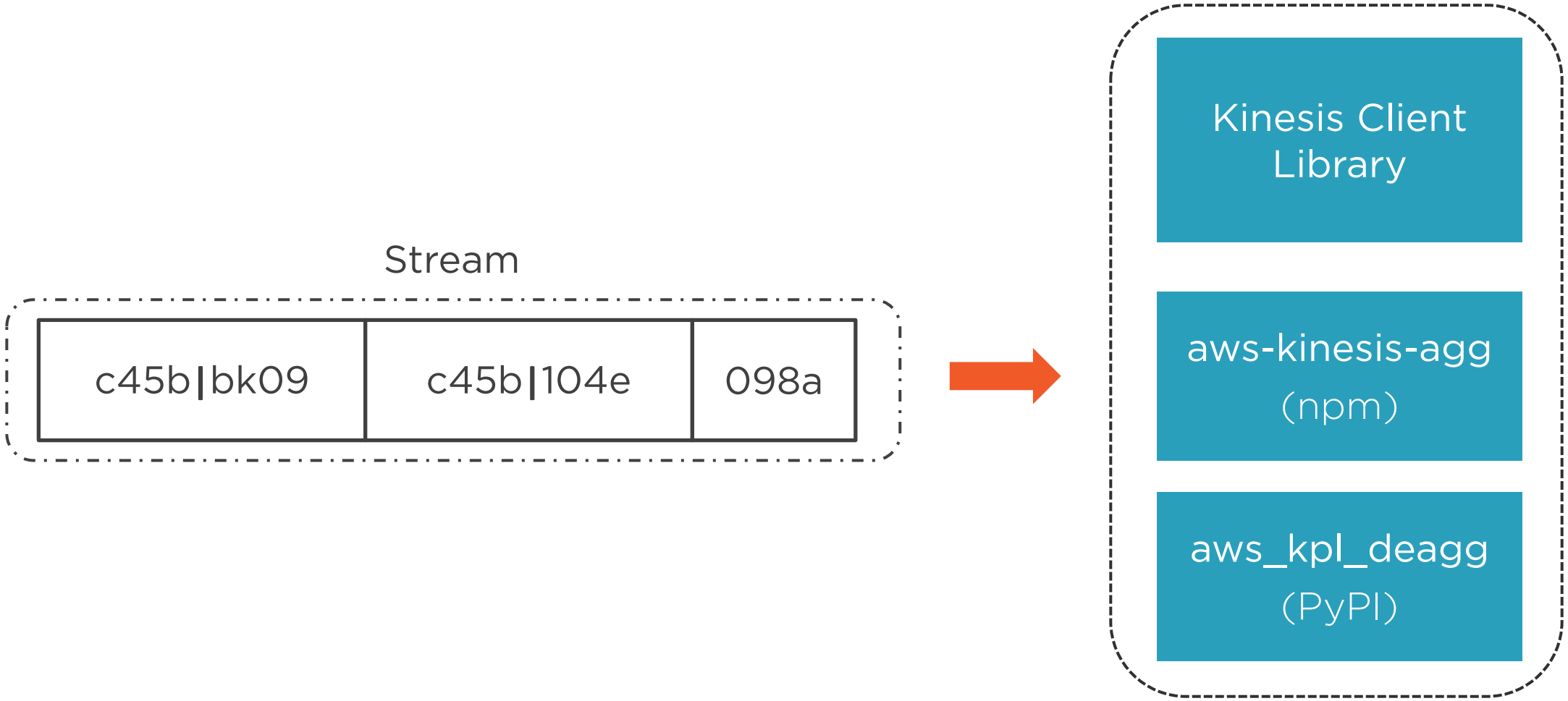
- Consumer should have access to same records
- Limit latency

Can be disabled

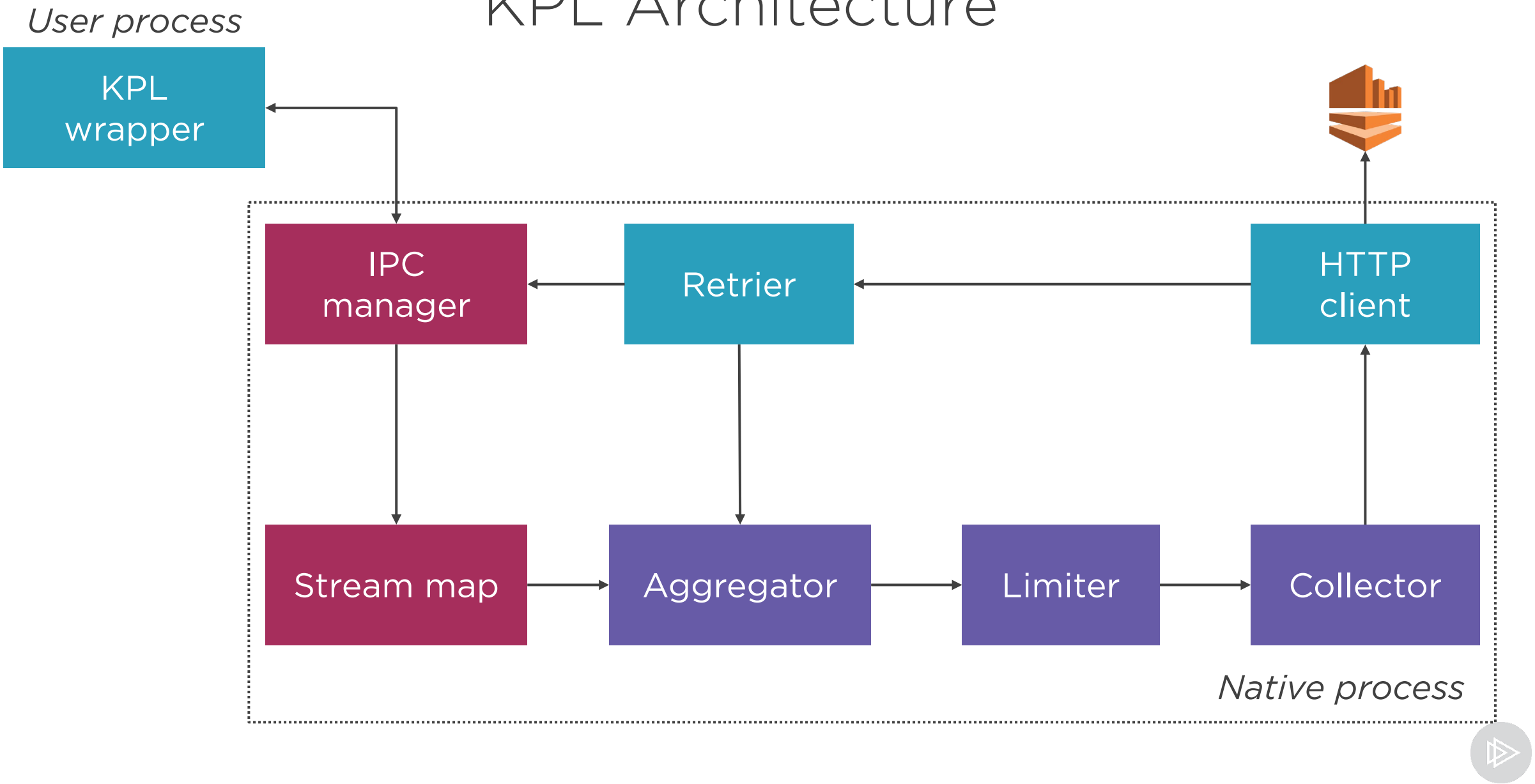
```
config
.setRecordMaxBufferedTime(3000)


.setMaxConnections(1)


.setRequestTimeout(60000)


.setRegion("us-west-1");
```

- How long an user record should be buffered, higher value results in more records being aggregated
- Degree of parallelism for HTTP requests
- Timeout for a single HTTP request. Higher timeout for slow connections
- AWS region

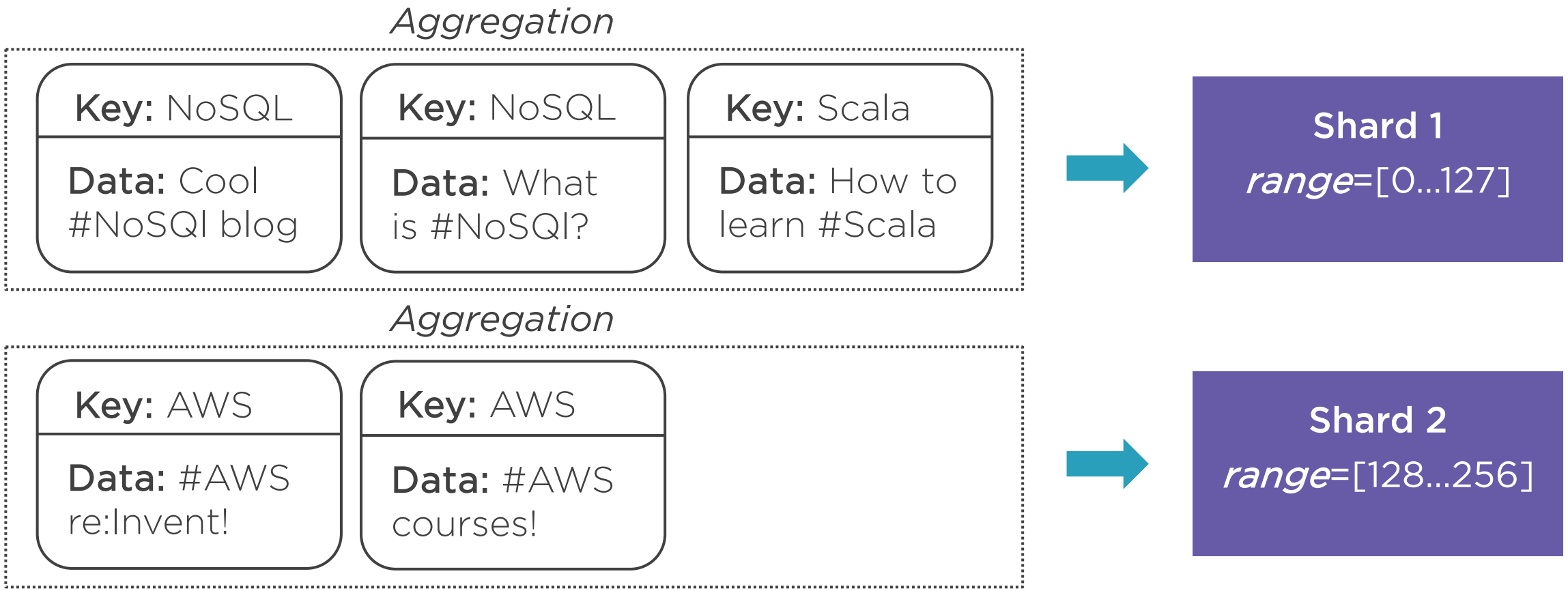
Decoding KPL Records



KPL Architecture



Aggregating with Multiple Shards




```
PutRecord {
    "ExplicitHashKey": "string"
}
```

Detailed Metrics

User Record Received

User Records Pending

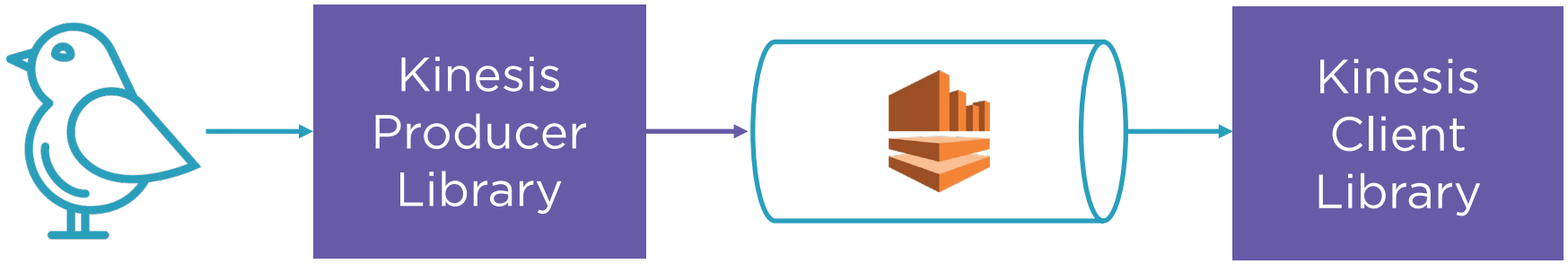
Kinesis Records Put

Buffering Time

User Records per Kinesis Record

Errors by Code

Using Kinesis Client Library



Why KCL?

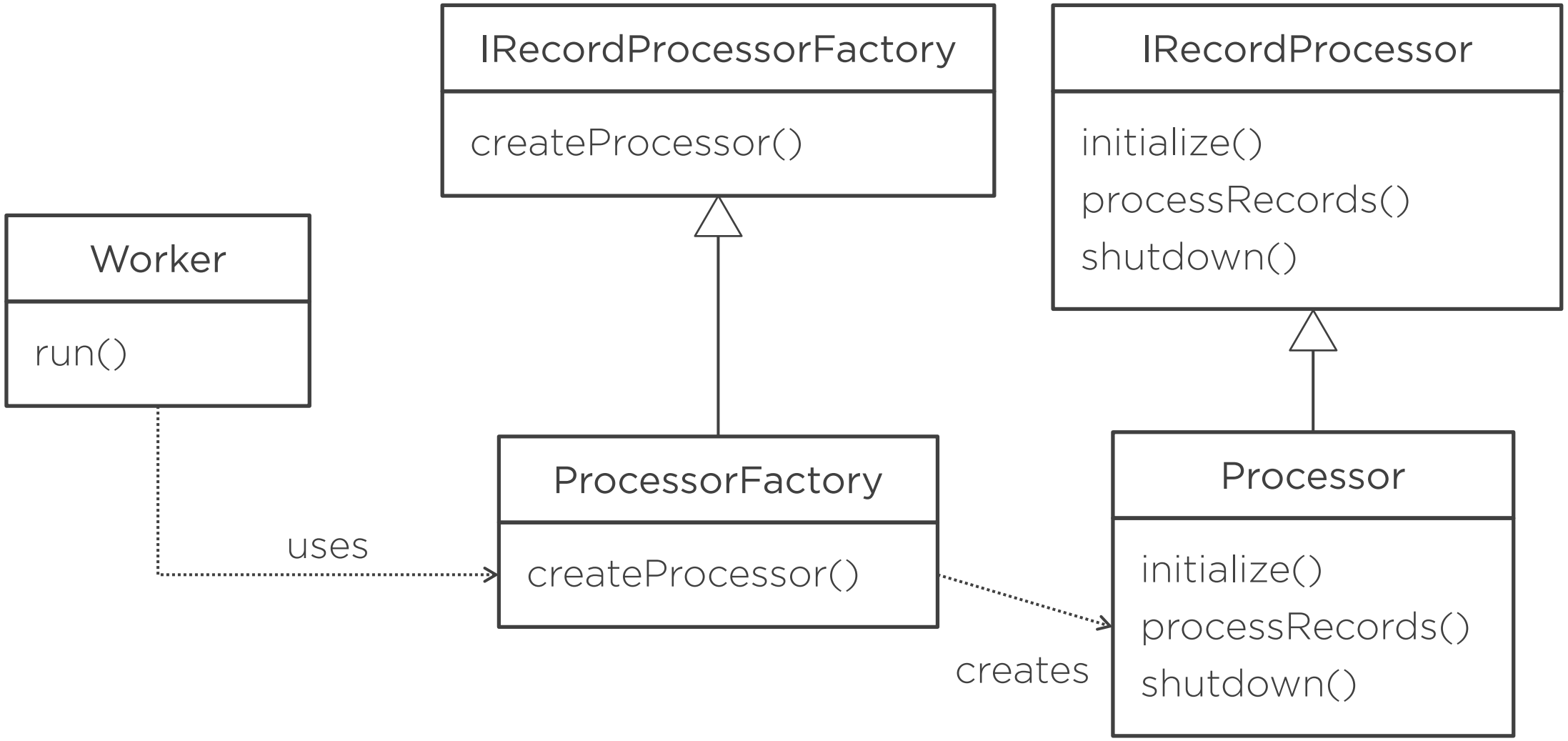
Simple API

Handles resharding

Errors handling

Monitoring

Kinesis Client Library



Using Worker

```
KinesisClientLibConfiguration config = new
KinesisClientLibConfiguration(
        "tweets-processor",  // Application name
        "tweets-stream",     // Stream name
        new DefaultAWSCredentialsProviderChain(),
        "worker-1”           // Name of the worker instance
);

IRecordProcessorFactory recordProcessorFactory = …

Worker worker = new Worker.Builder()
    .config(config)
    .recordProcessorFactory(recordProcessorFactory)
    .build();
```

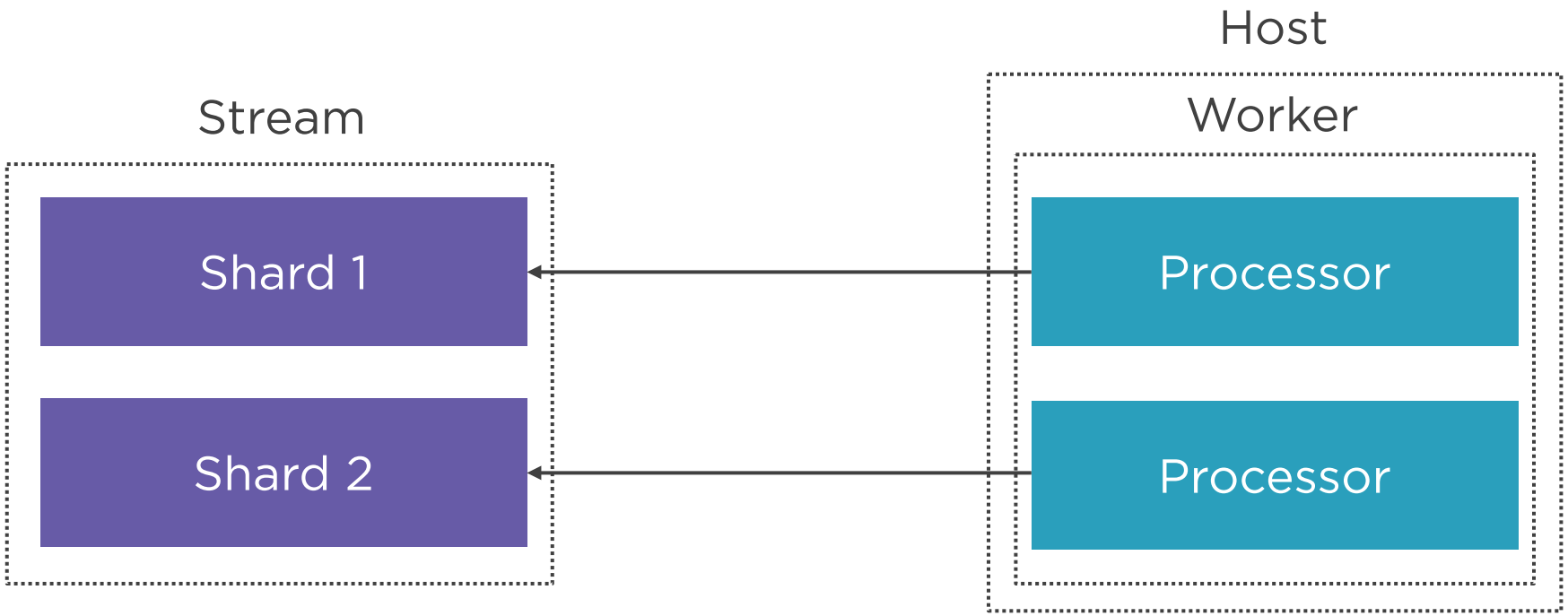
Configuring Worker

```
KinesisClientLibConfiguration config = …

// Starting position in a stream
// Can be: LATEST, AT_TIMESTAMP, TRIM_HORIZON
config.withInitialPositionInStream(
    InitialPositionInStream.LATEST);

// How often to query for new data
config.withIdleTimeBetweenReadsInMillis(200);
```

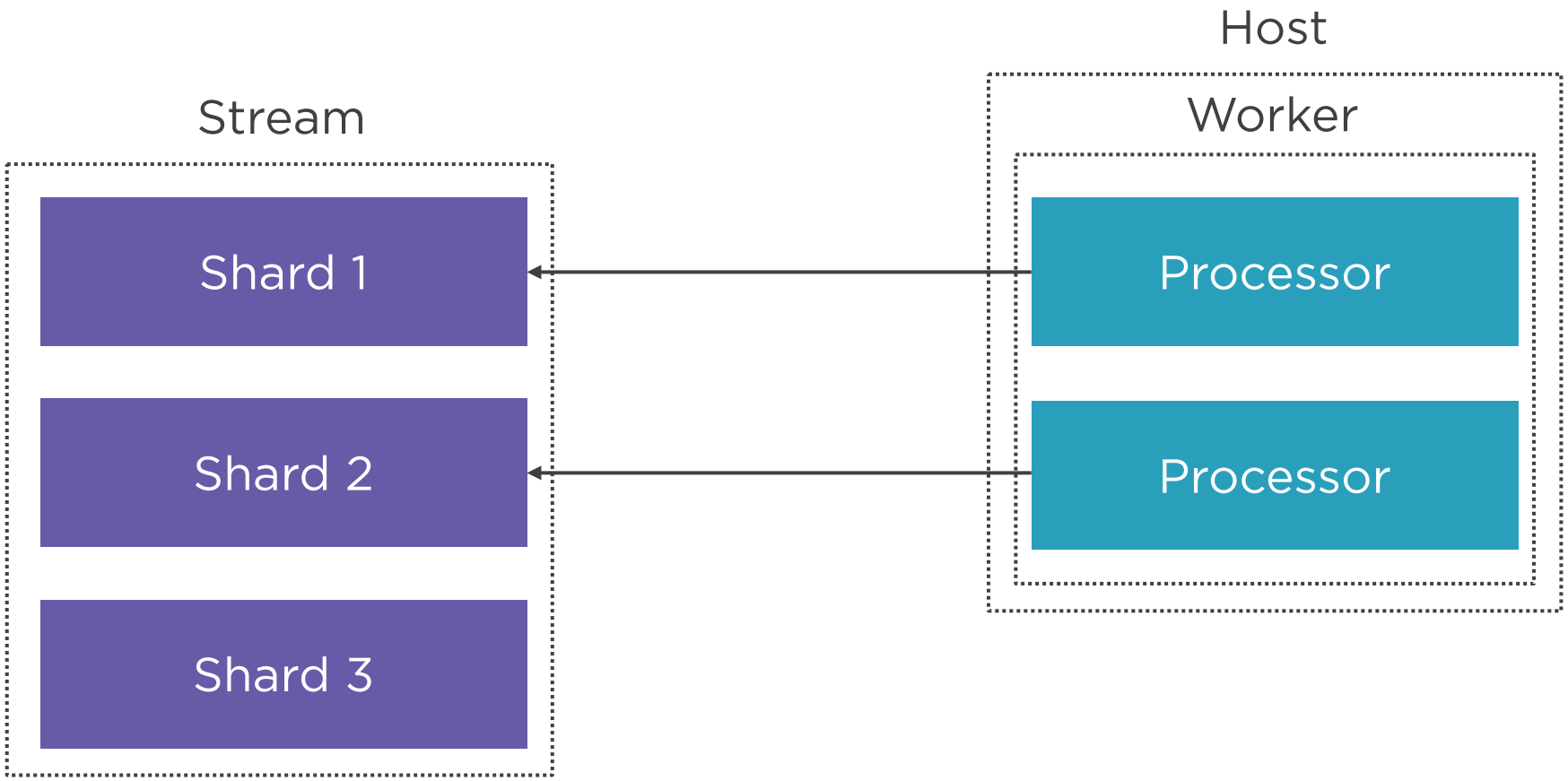
Processing Records

```
public void processRecords(ProcessRecordsInput input) {

  for (Record record : processRecordsInput.getRecords()) {

    String partKey = record.getPartitionKey();
    ByteBuffer data = record.getData();
    String seq = record.getSequenceNumber();

    if (record instanceof UserRecord)
      String subSeq = record.getSubSequenceNumber();

  }

}
```

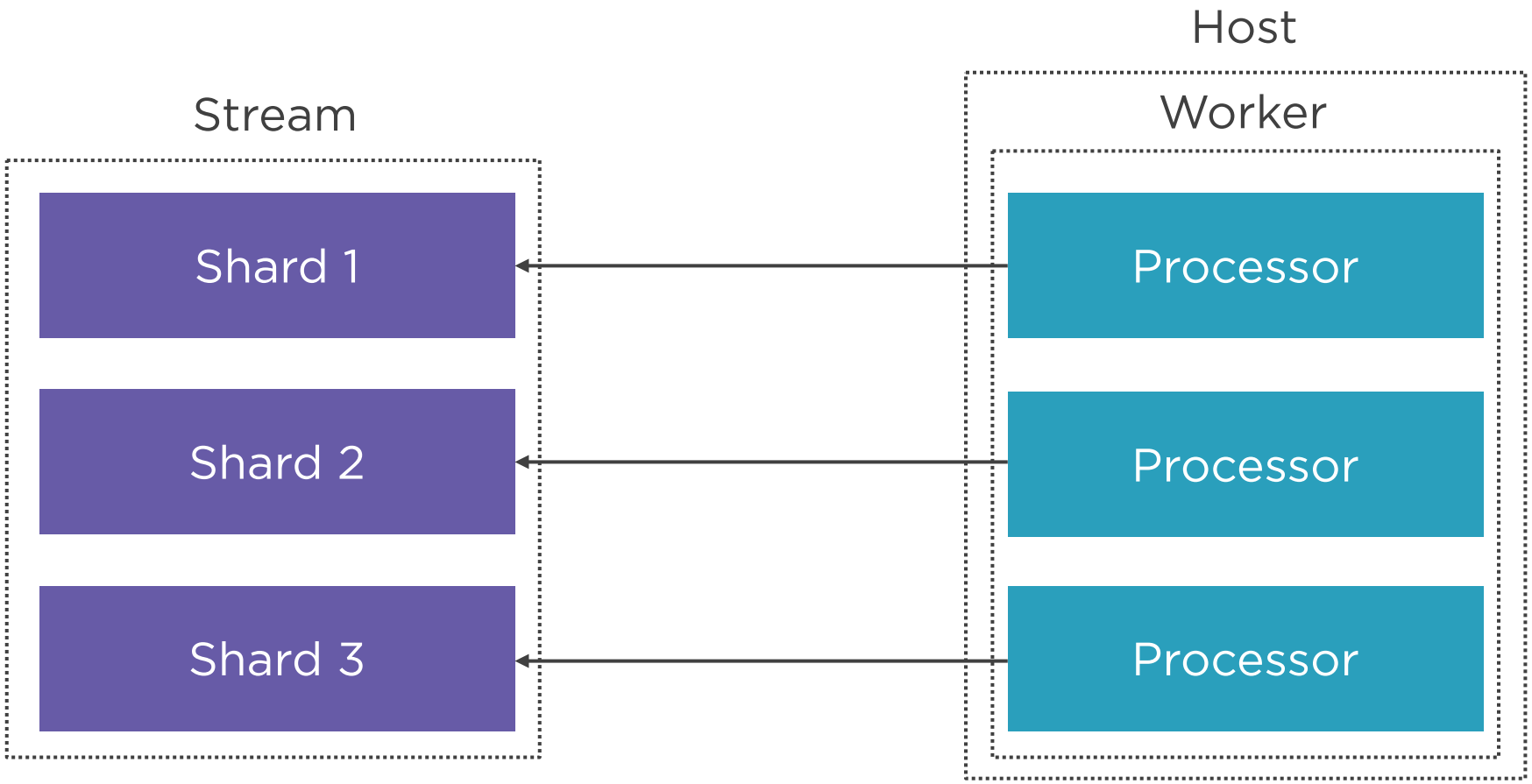
Scaling with KCL



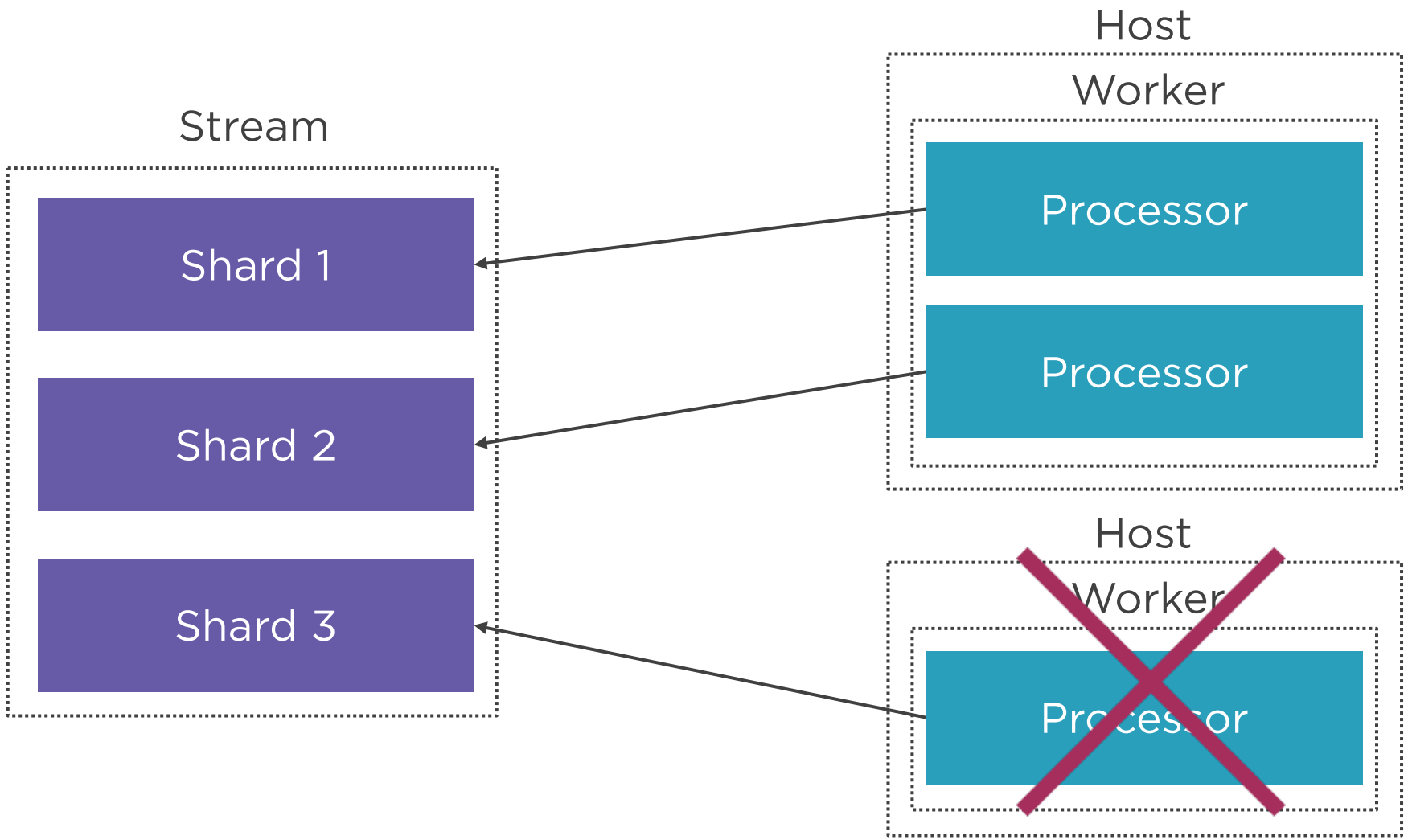
Scaling with KCL



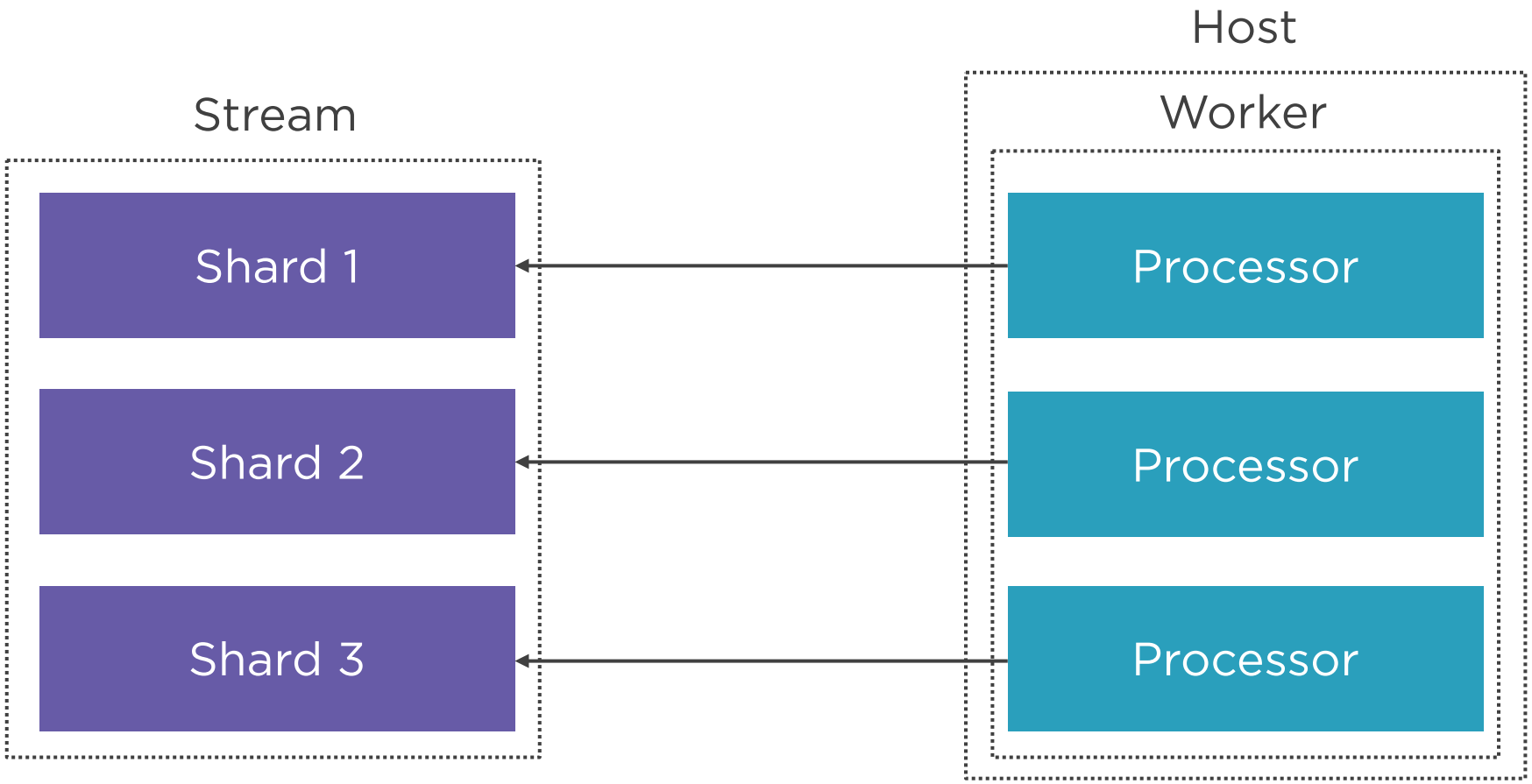
Scaling with KCL



Scaling with KCL

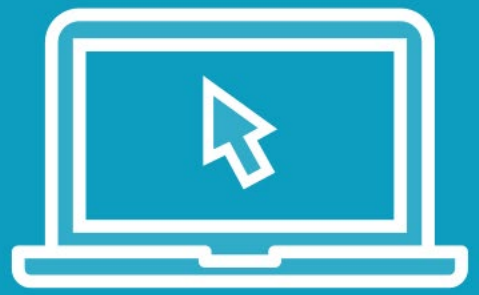


Scaling with KCL



Kinesis Client Library does not launches additional hosts. It only starts new processors.

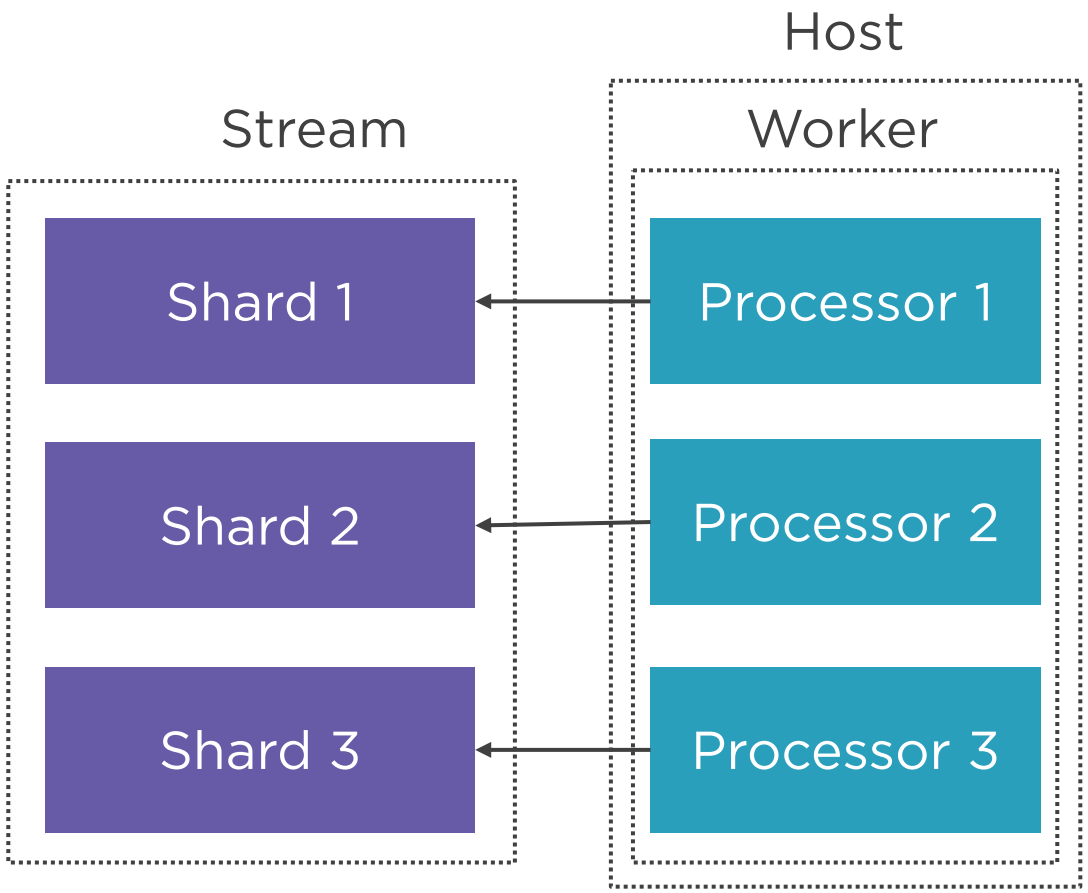
Demo



Implement simple application with KCL

Read tweets written with KPL

Fault Tolerance with KCL

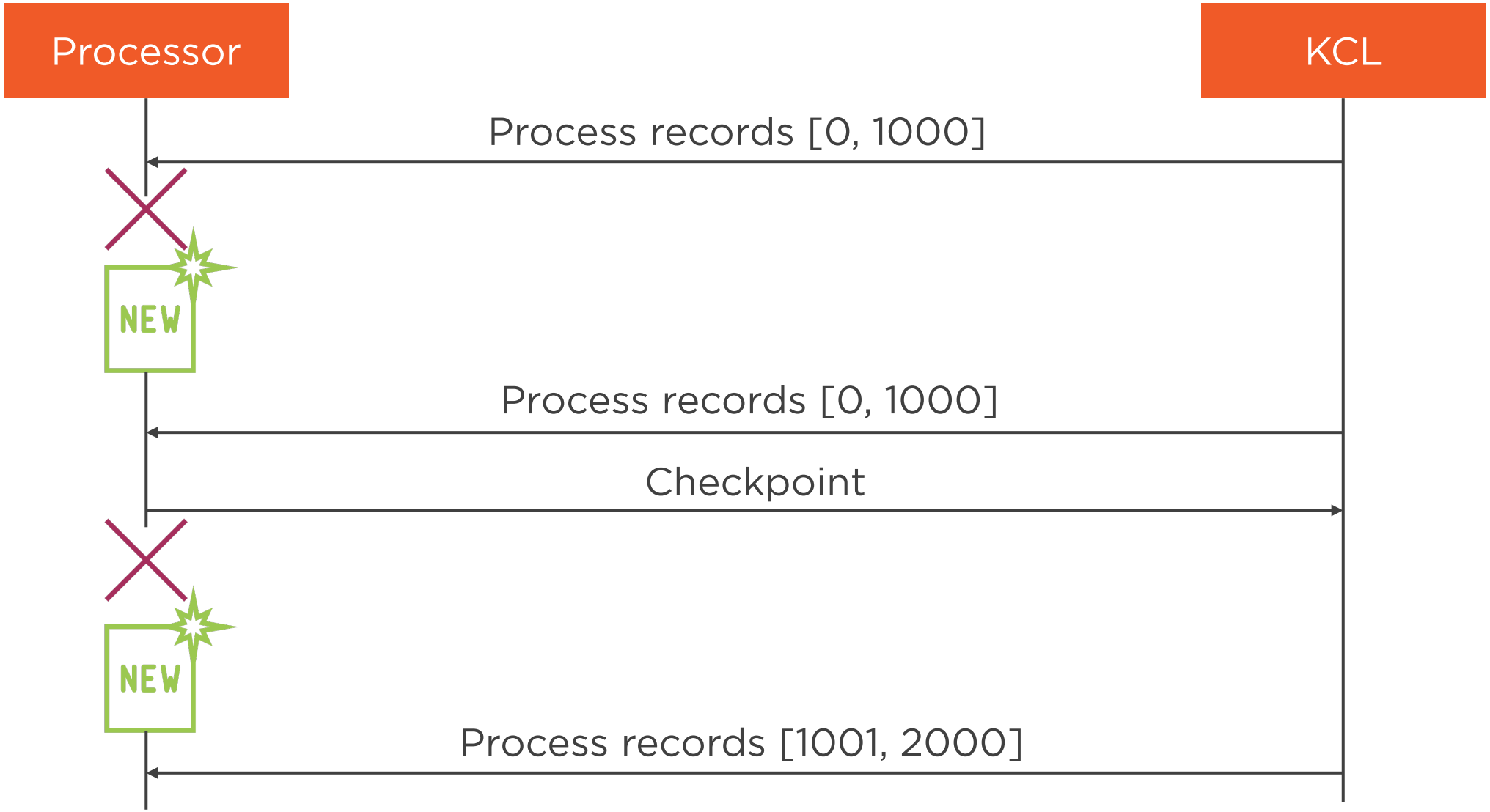


shard	seq	subSeq	owner
1	100	1	processor_1
2	200	0	processor_2
3	134	2	processor_3

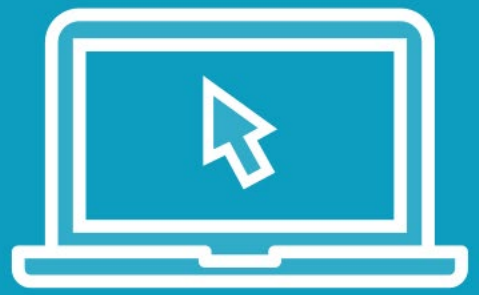
Checkpointing

```
public void processRecords(ProcessRecordsInput input) {
    IRecordProcessorCheckpointer checkpointer =
        input.getCheckpointer();
}

// Checkpoint last record
checkpointer.checkpoint();
// Checkpoint specified record
checkpointer.checkpoint(record);
// Checkpoint sequence number
checkpointer.checkpoint(sequenceNumber);
//Checkpoint aggregated record from KPL
checkpointer.checkpoint(sequenceNumber, subSequenceNumber);
```

KCL Checkpointing



Demo

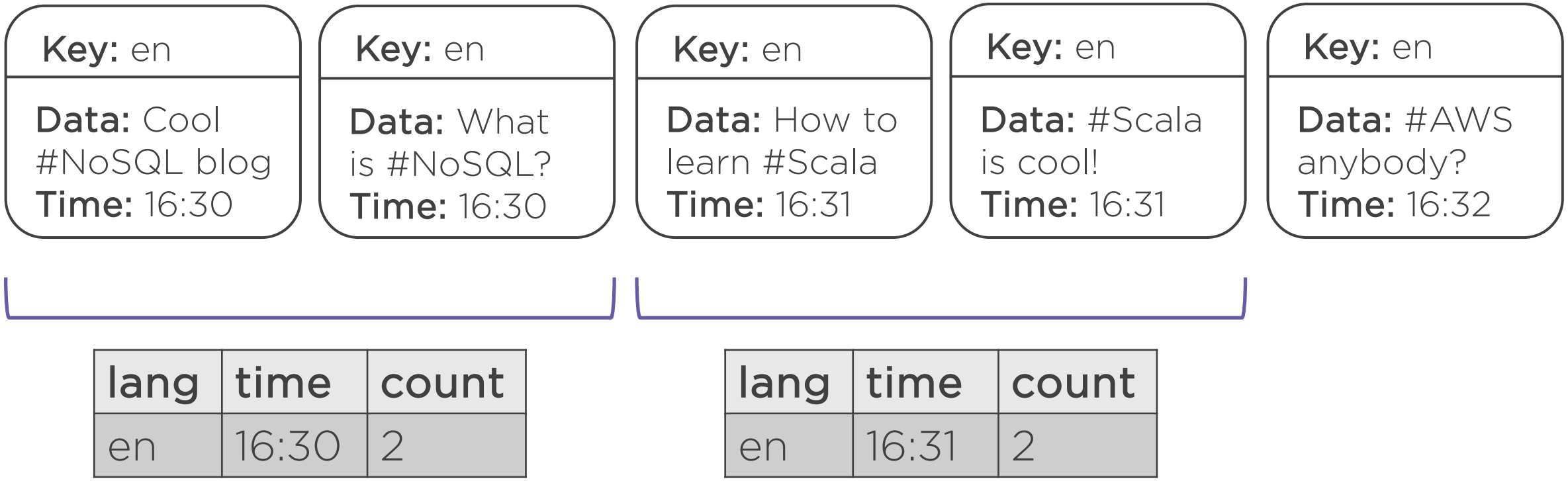


Add fault tolerance

Use checkpointing correctly

Tweet-counting Application

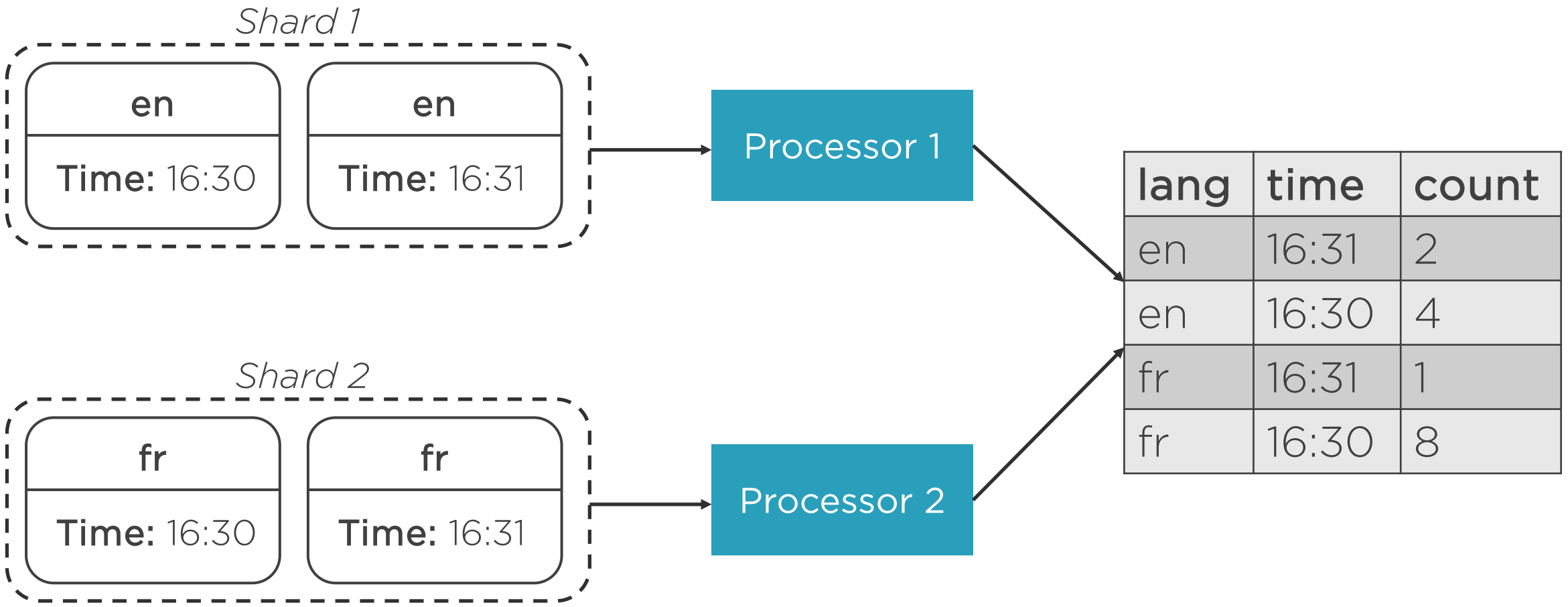
Counting the Number of Tweets



lang	time	count
en	16:30	2

lang	time	count
en	16:31	2

Processing Multiple Shards



Event Time

Processing time

System time

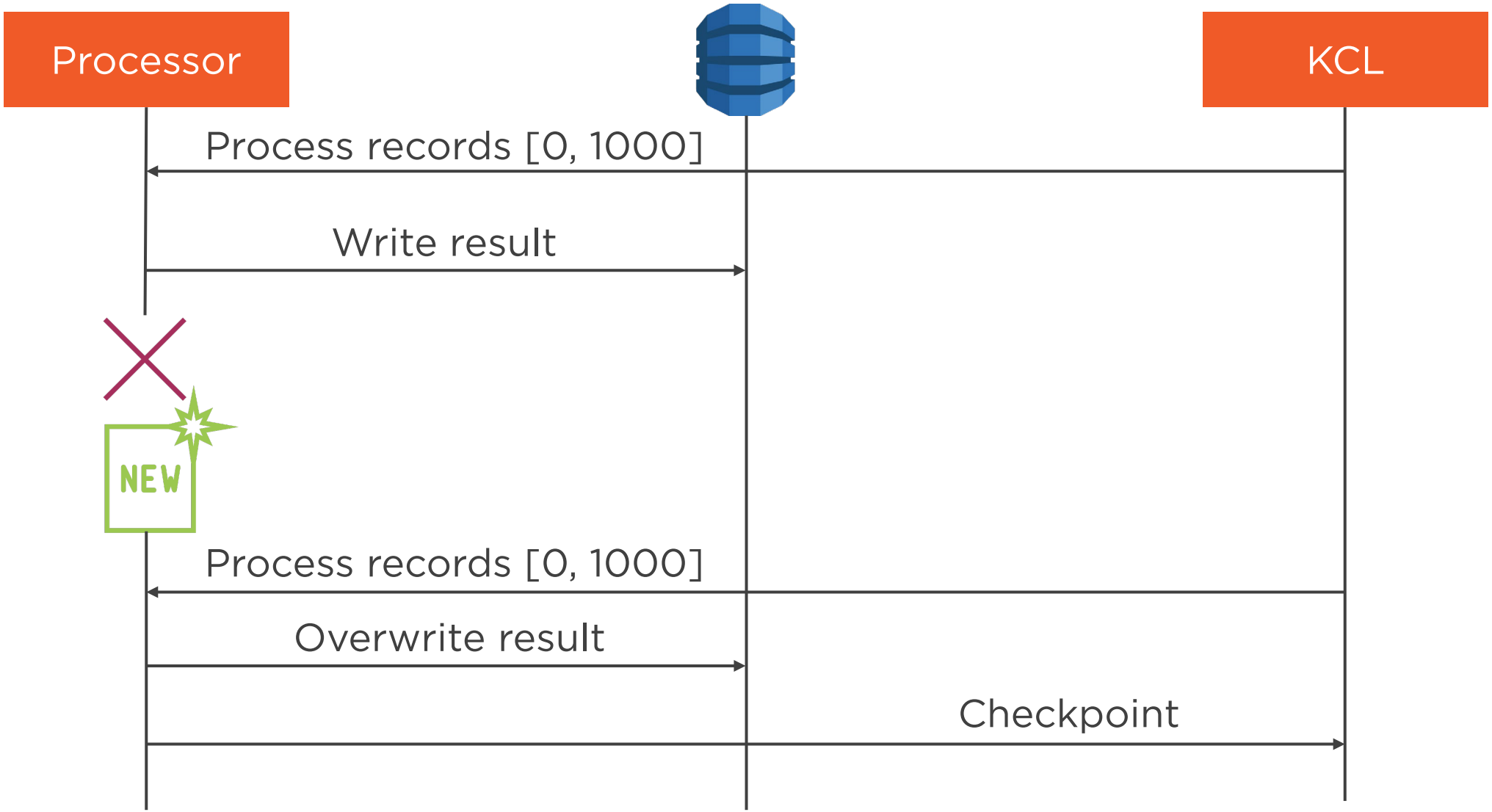
Event time

Time when an event was created

Idempotence

The property of certain operations that can be applied multiple times without changing the result

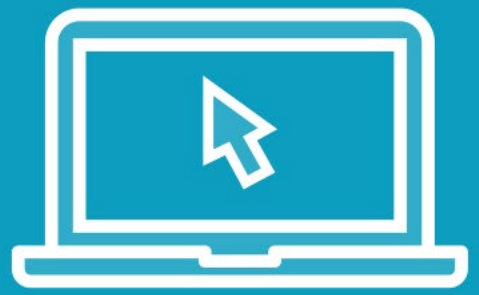
Idempotent Log Processing



Prepare Checkpointing

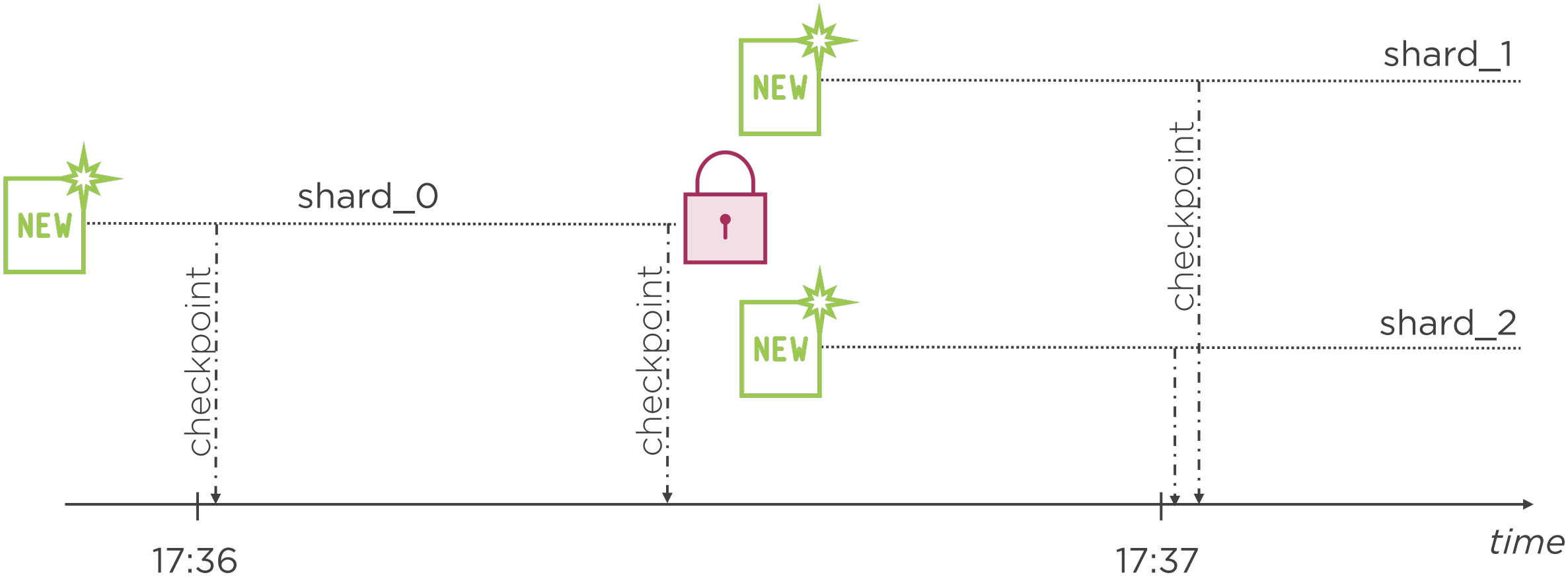
```
void initialize(InitializationInput input) {
    input.getPendingCheckpointSequenceNumber();
}
// Prepare checkpoint
checkpointer.prepareCheckpoint();
checkpointer.prepareCheckpoint(record);
checkpointer.prepareCheckpoint(sequenceNumber);
checkpointer.prepareCheckpoint(
    sequenceNumber,
    subSequenceNumber);
```

Demo



Implement tweet-counting application

Checkpointing with Resharding

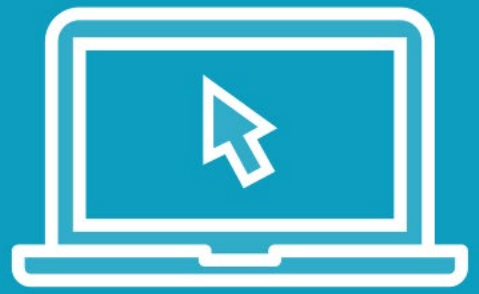


Recording Shard Name

lang	time	shard	count
en	16:30	0	2

lang	time	shard	count
en	16:31	0	2

Demo



Fix tweet-counting application

Is There a Better Way?

What are some problems with our approach?

Why KCL is not enough?

What Is Missing – Too Much Code

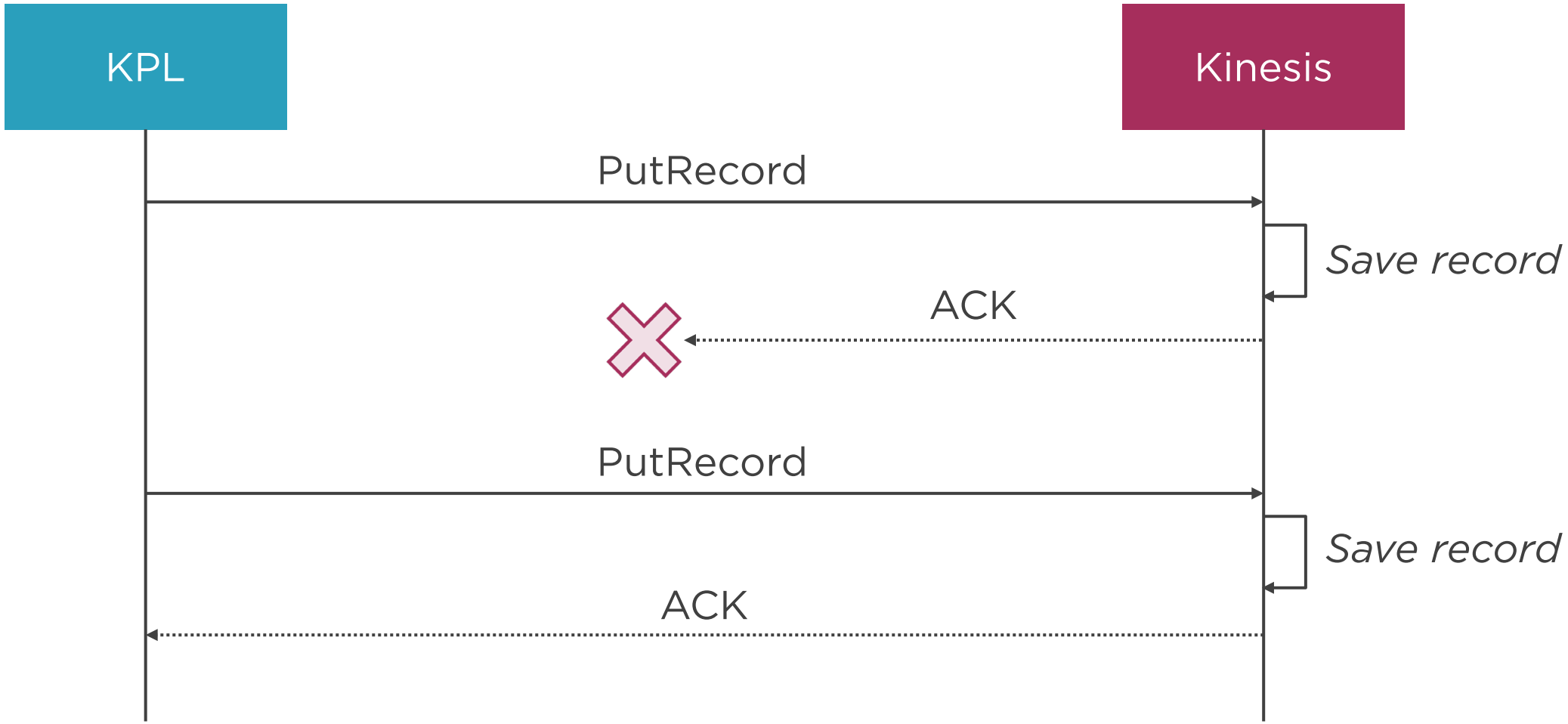
Implemented simple example

Still quite low-level

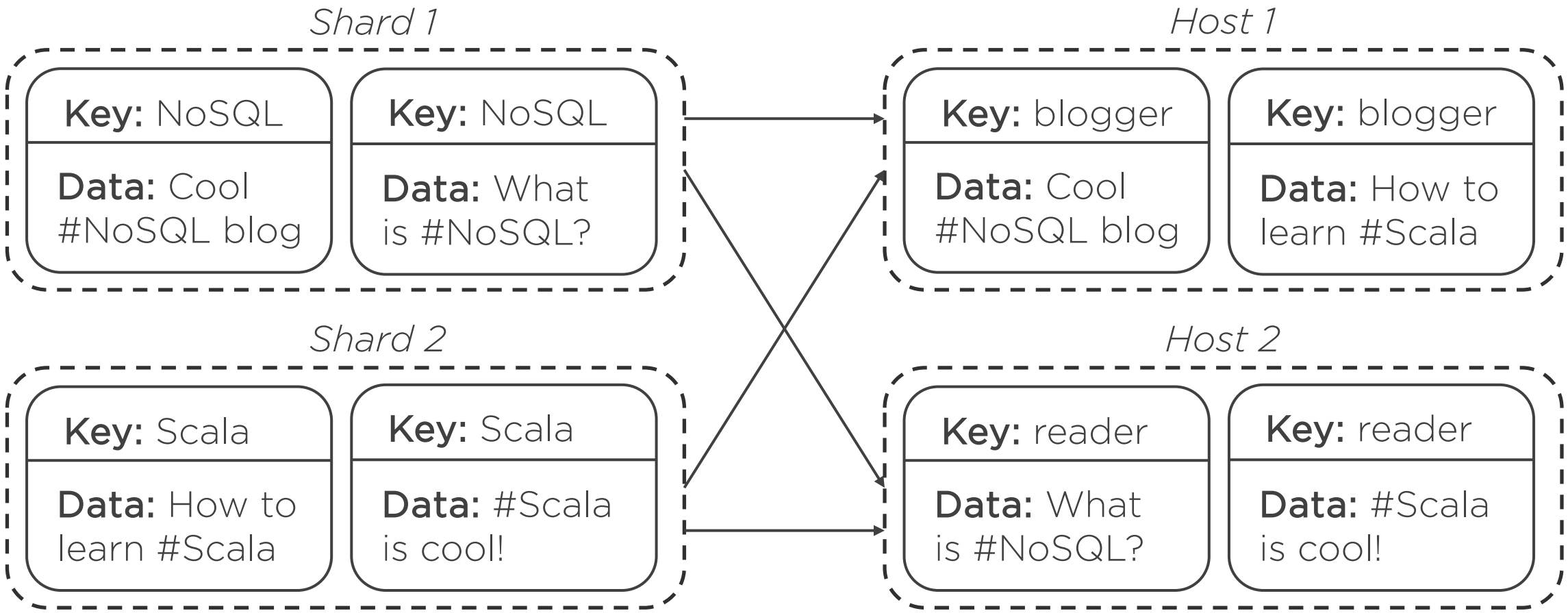
More complex use cases?

Infrastructure maintenance

What Is Missing – Duplicated Records



What Is Missing - Reshuffling

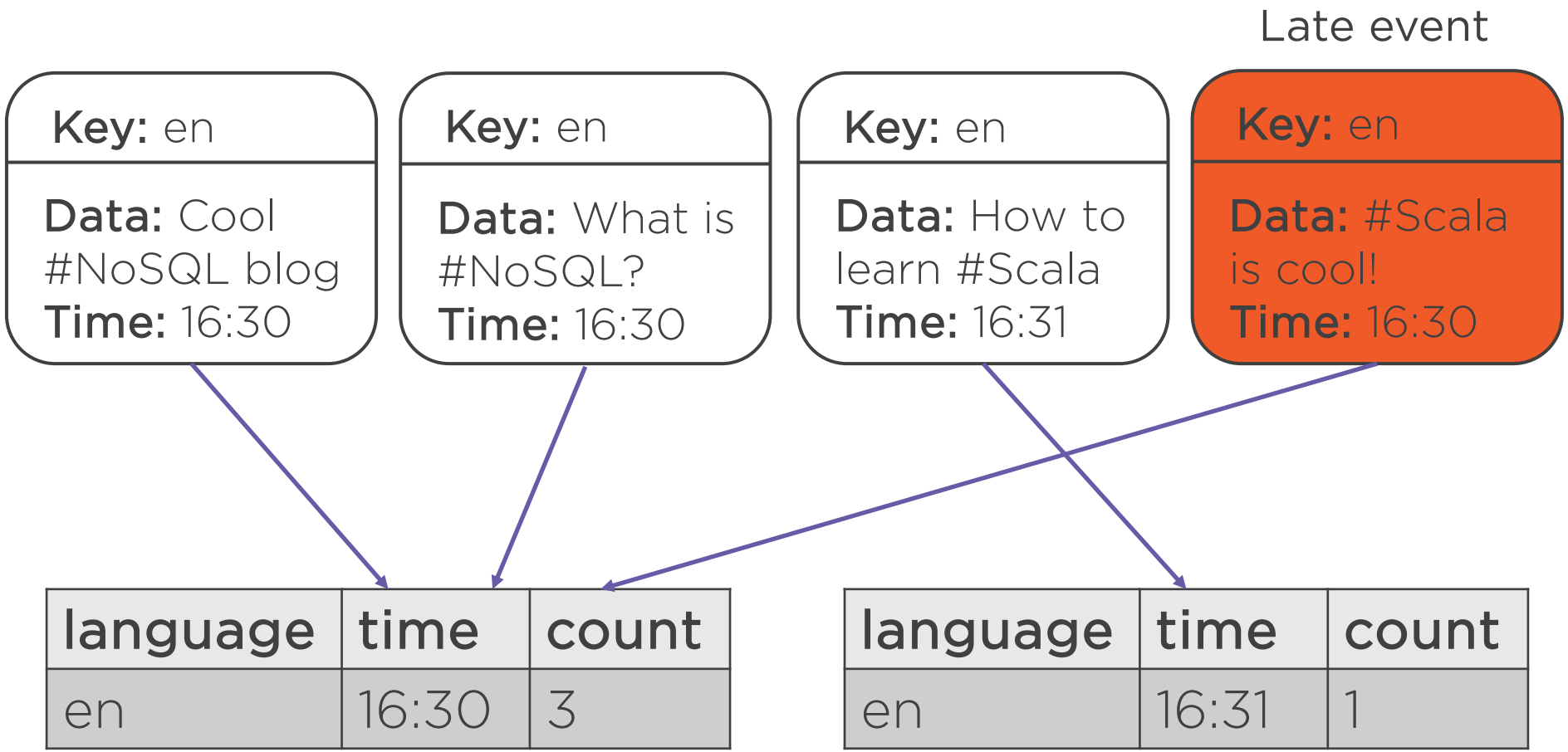


What Is Missing – Late Events

language	time	count
en	16:30	2

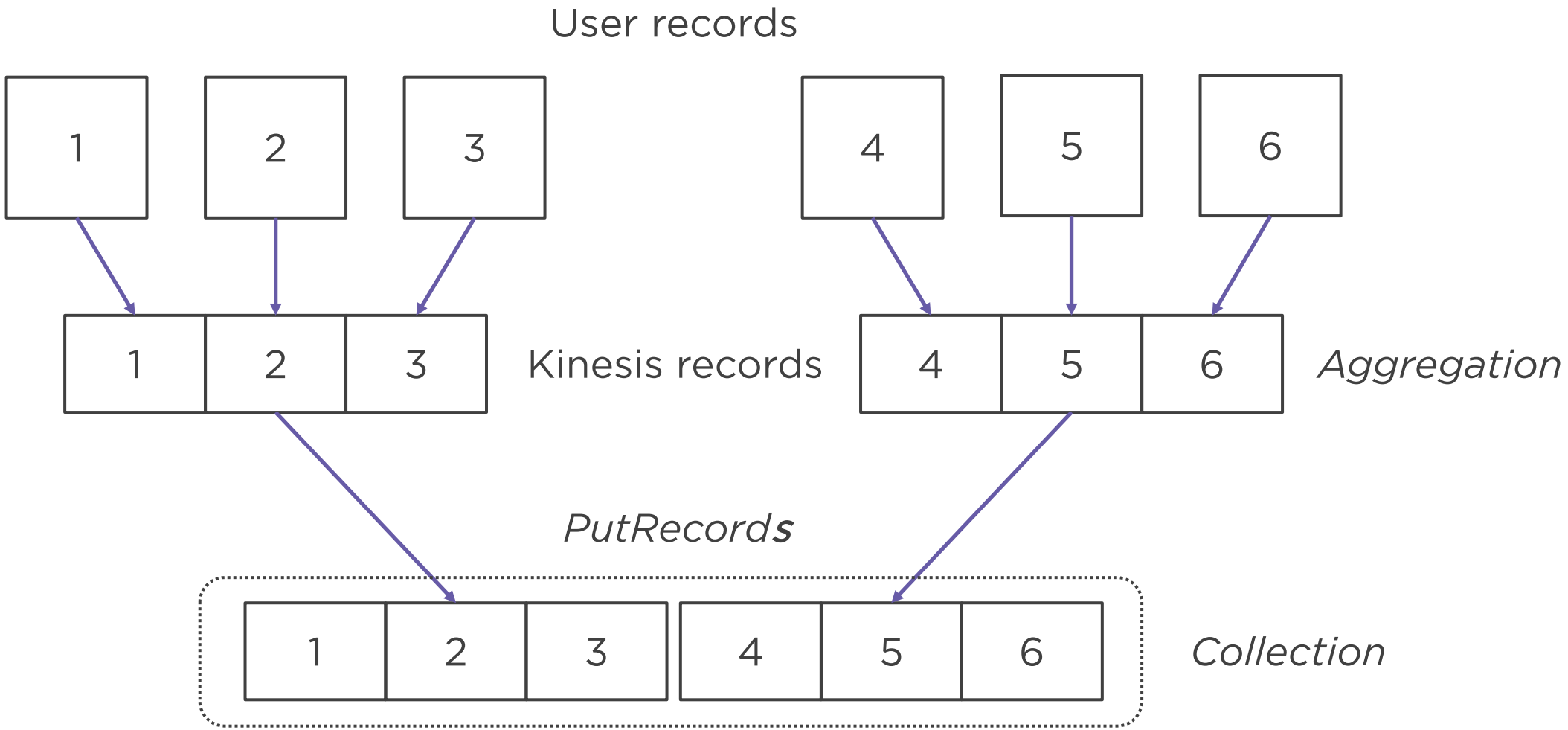
language	time	count
en	16:31	2

What Is Missing – Late Events

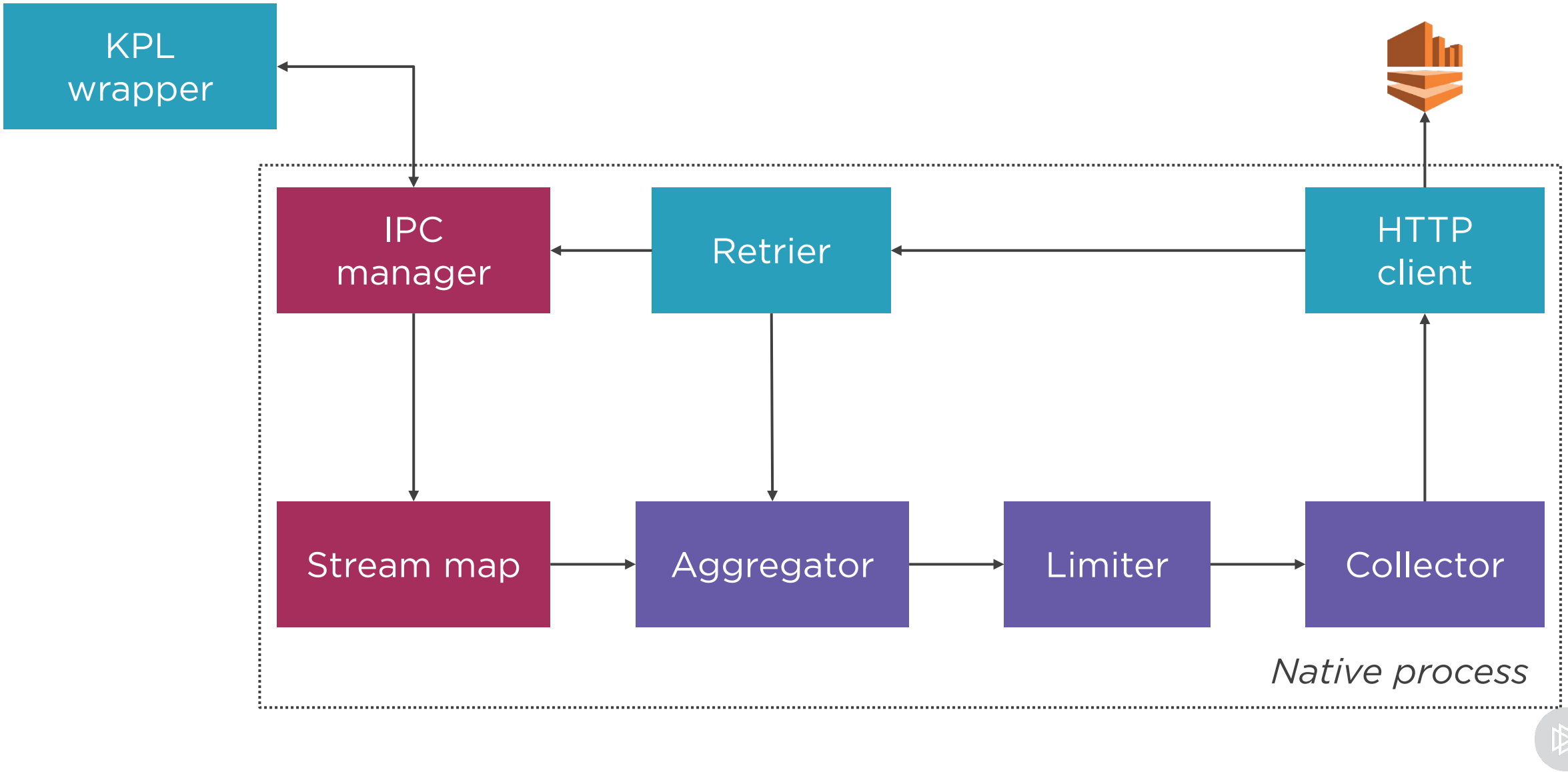


Summary

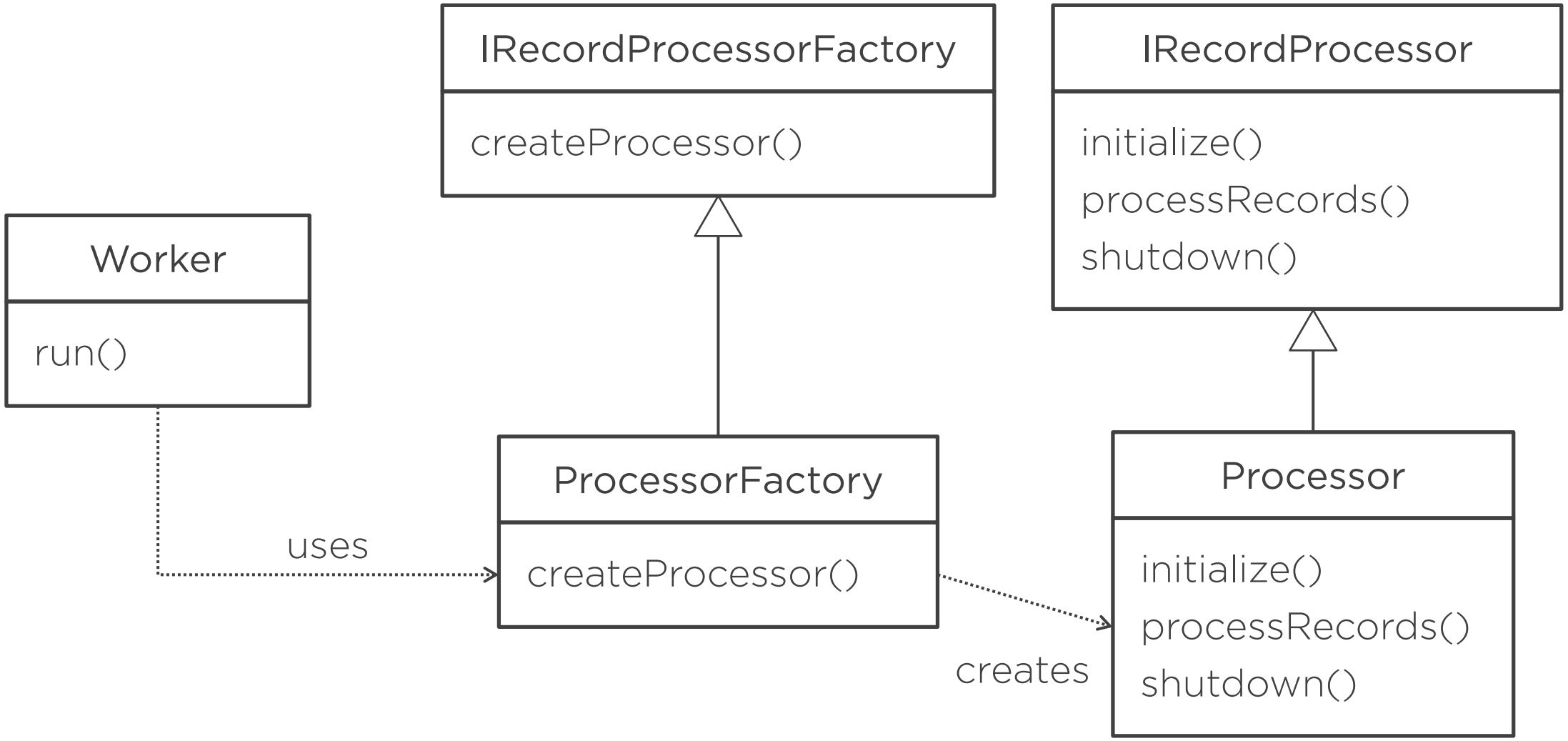
Summary - Aggregation



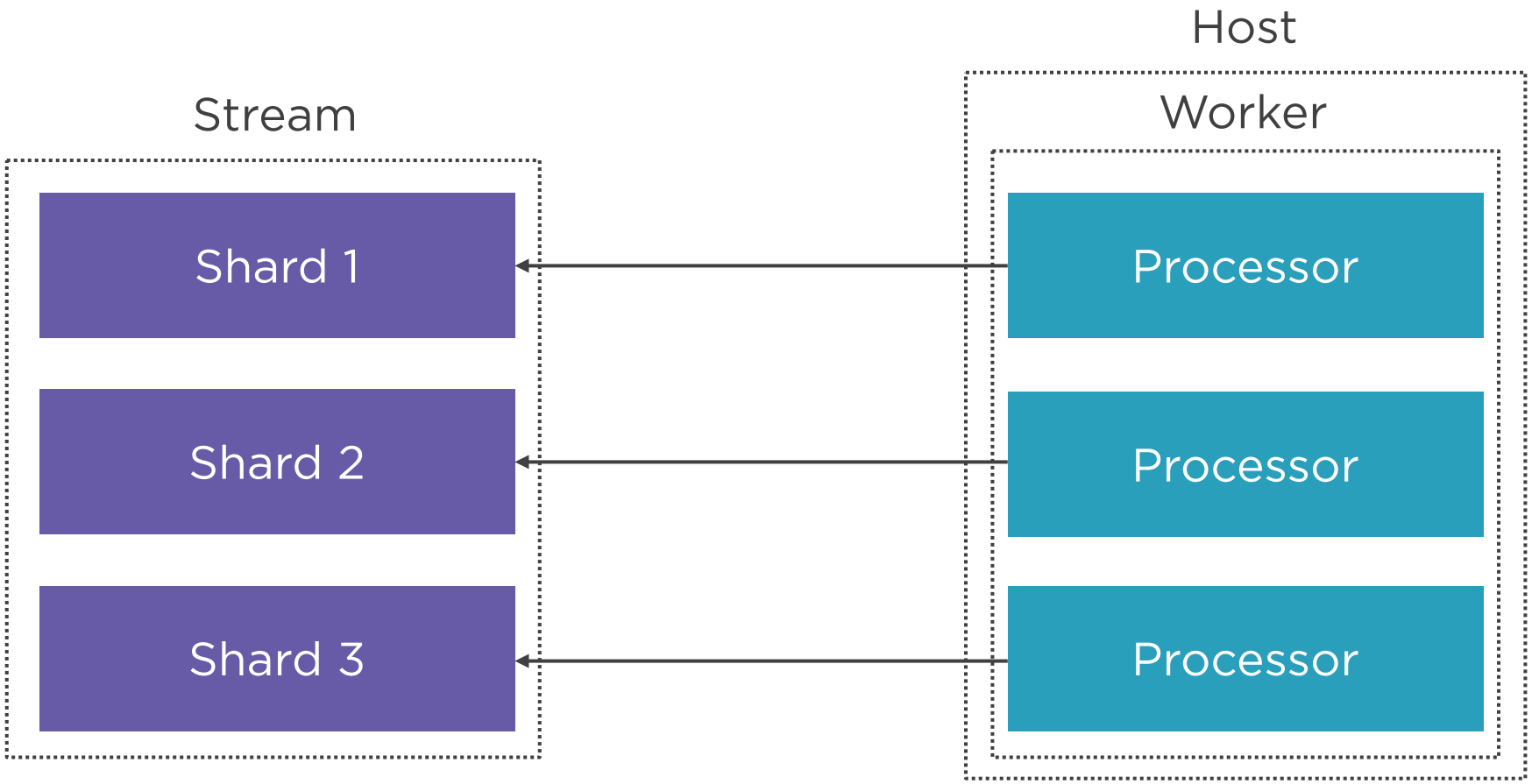
Summary - KPL Architecture



Summary - Kinesis Client Library



Summary - Scaling with KCL



Summary - Checkpointing