samza

The Team

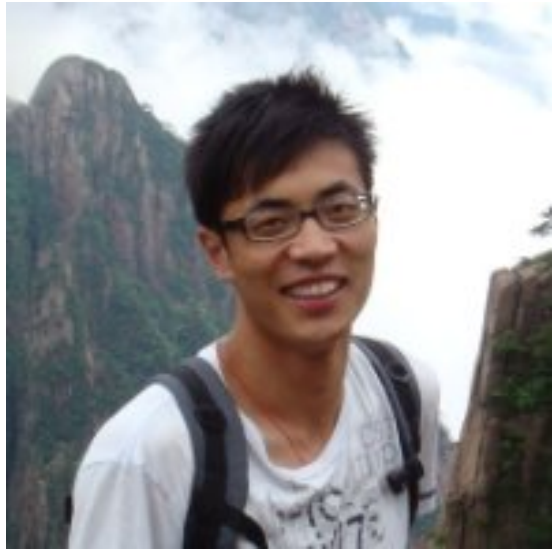
Renato Marroquín

- PhD student at:
- Interested in:
  - Stream processing
  - Distributed data management
- Apache contributor
  - Apache Gora, Giraph, Nutch, Samza
- rmarroquin [at] apache [dot] org

The Team

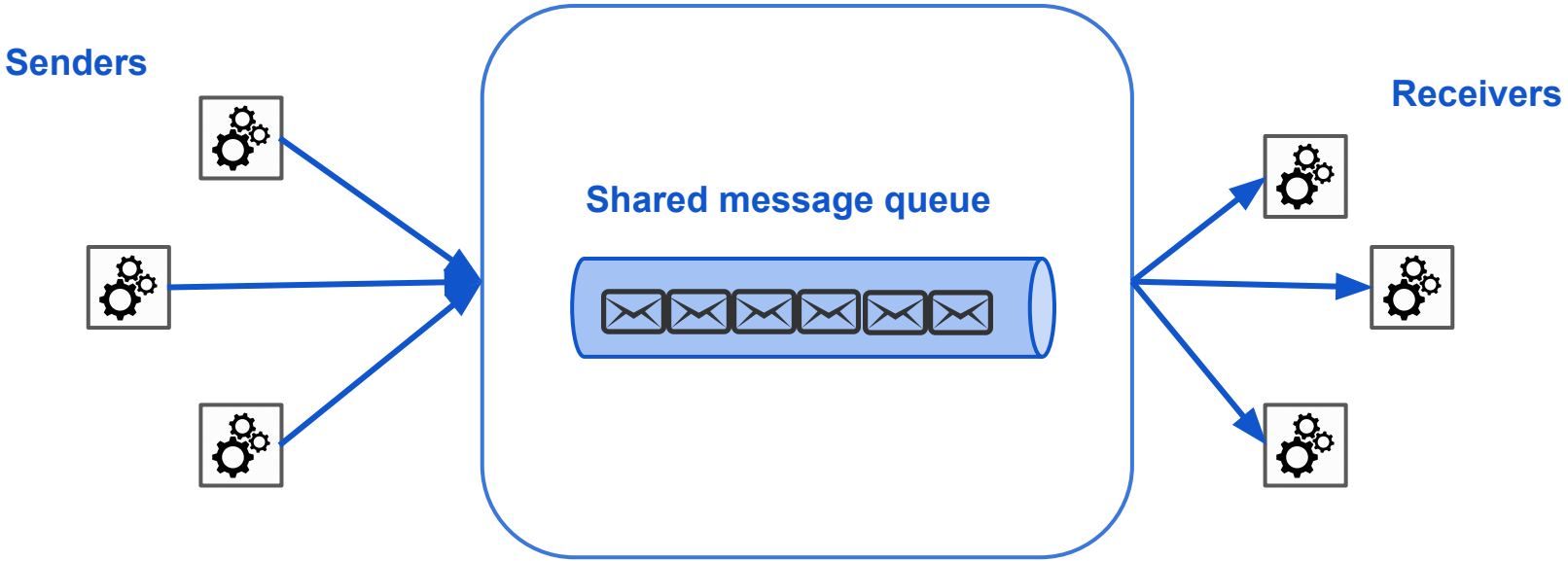
Yan Fang

- Software engineer at: ORACLE
- Interested in:
  - Stream processing
  - Information retrieval
  - Natural language processing
- Committer and PMC for Apache Samza
- yfang [at] apache [dot] org



Background

- Messaging systems



Background

- Messaging systems

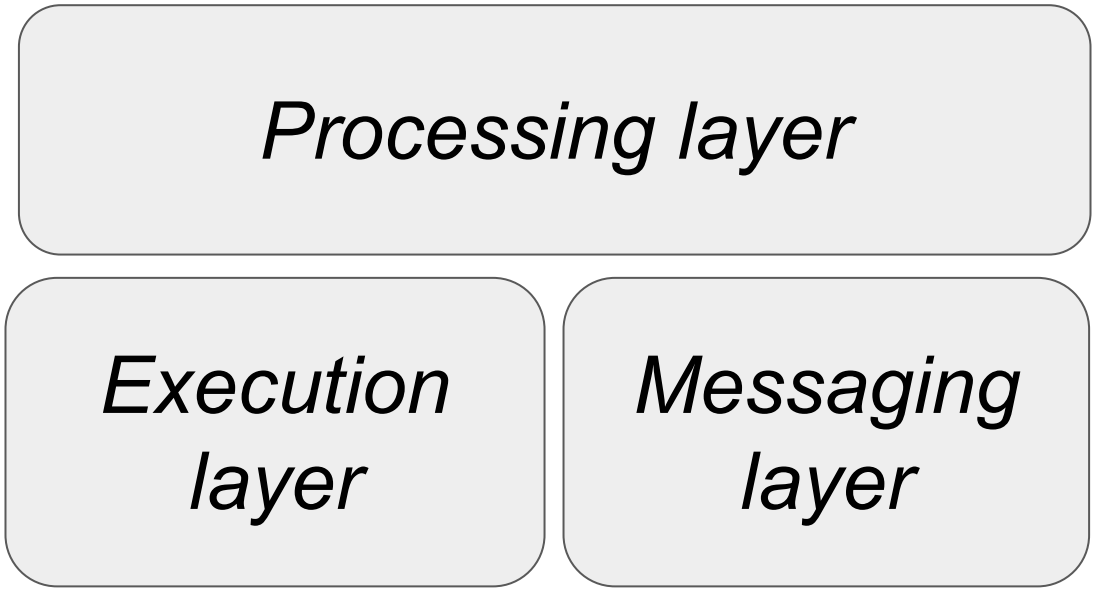
Background

- Stream processing
  - Message producers and consumers are not so trivial
  - Partitioning
  - State
  - Failure semantics
  - Reprocessing
  - Joins to services or databases

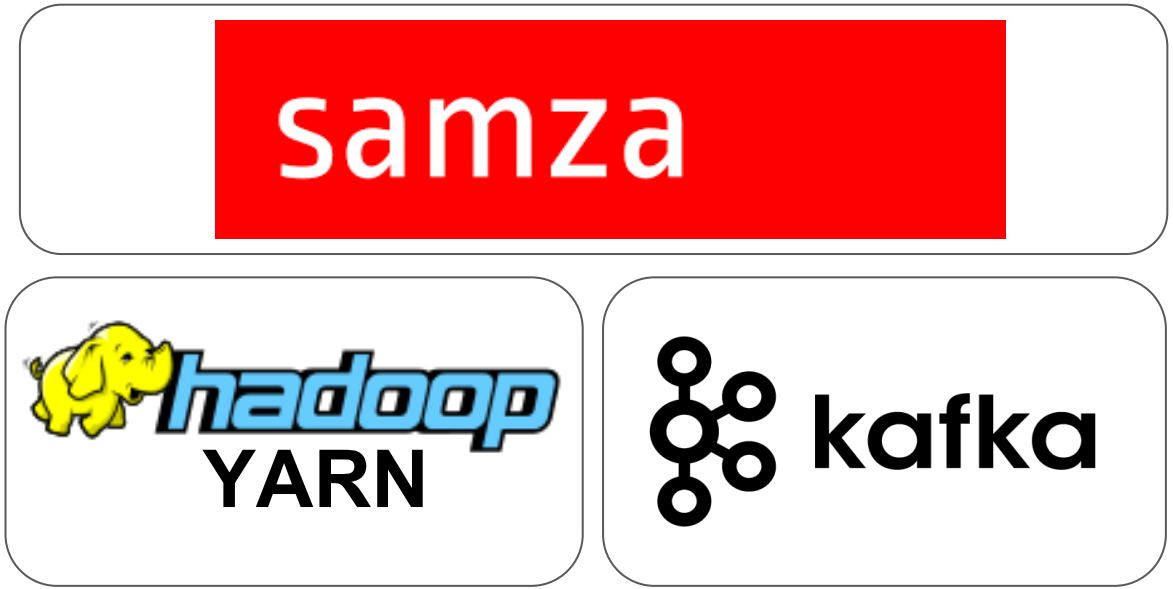
samza

- Distributed stream processing framework
- Developed @ LinkedIn
- Open sourced ~ 2013
- Used by
  - LinkedIn
  - Uber
  - Tivo
  - Nextel
  - Metamarkers
  - ...

Apache Samza's architecture



Apache Samza's architecture



Apache Samza's architecture

Apache Samza

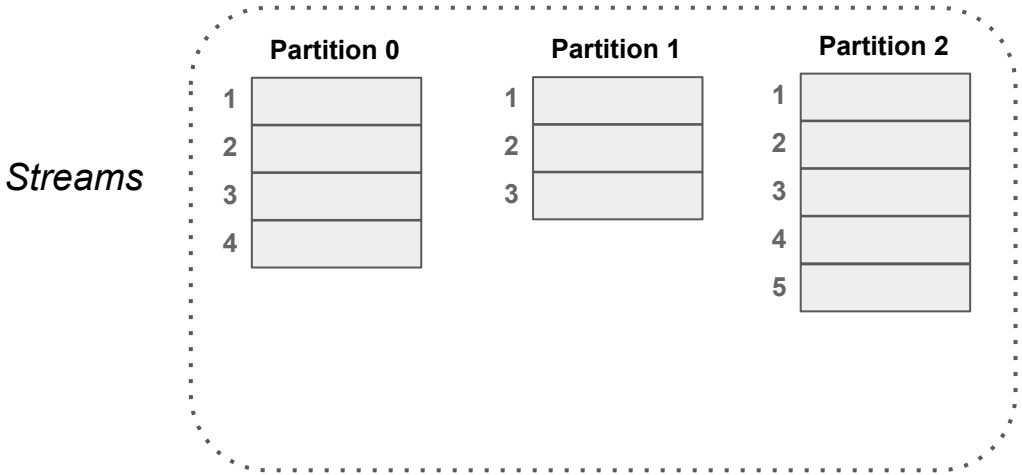
How it works?

- ***Streams***
- Tasks
- Execution

Apache Samza

How it works?

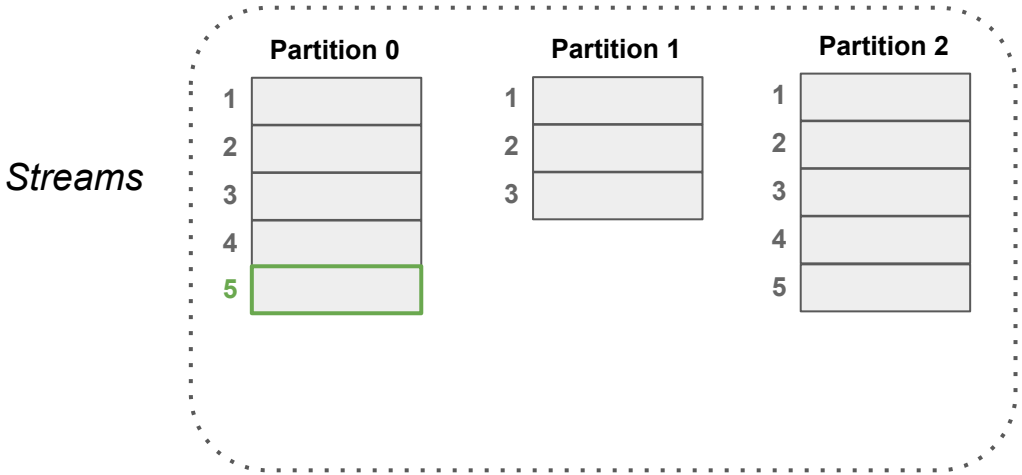
- ***Streams***
- Tasks
- Execution



Apache Samza

How it works?

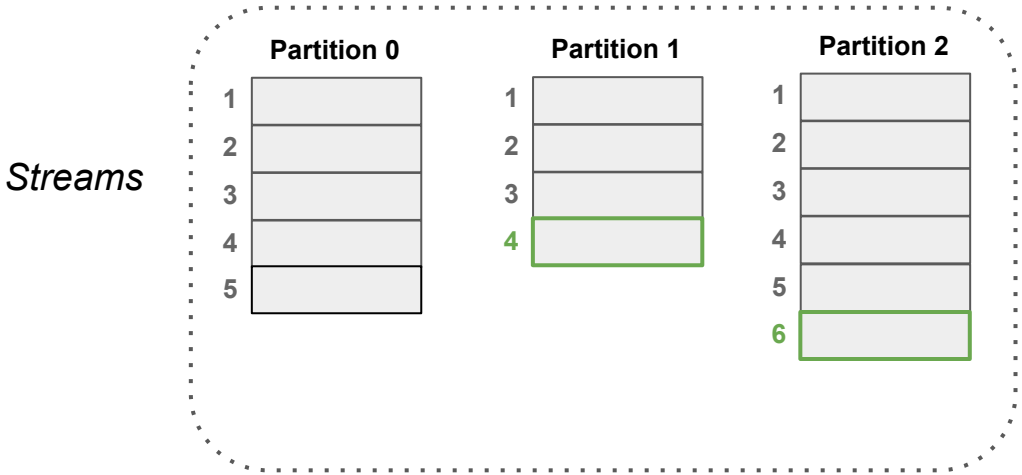
- ***Streams***
- Tasks
- Execution



Apache Samza

How it works?

- ***Streams***
- Tasks
- Execution

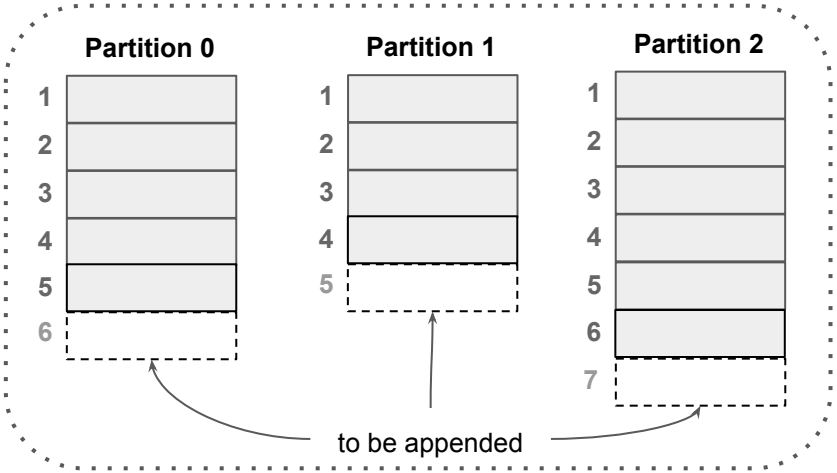


Apache Samza

How it works?

- ***Streams***
- Tasks
- Execution

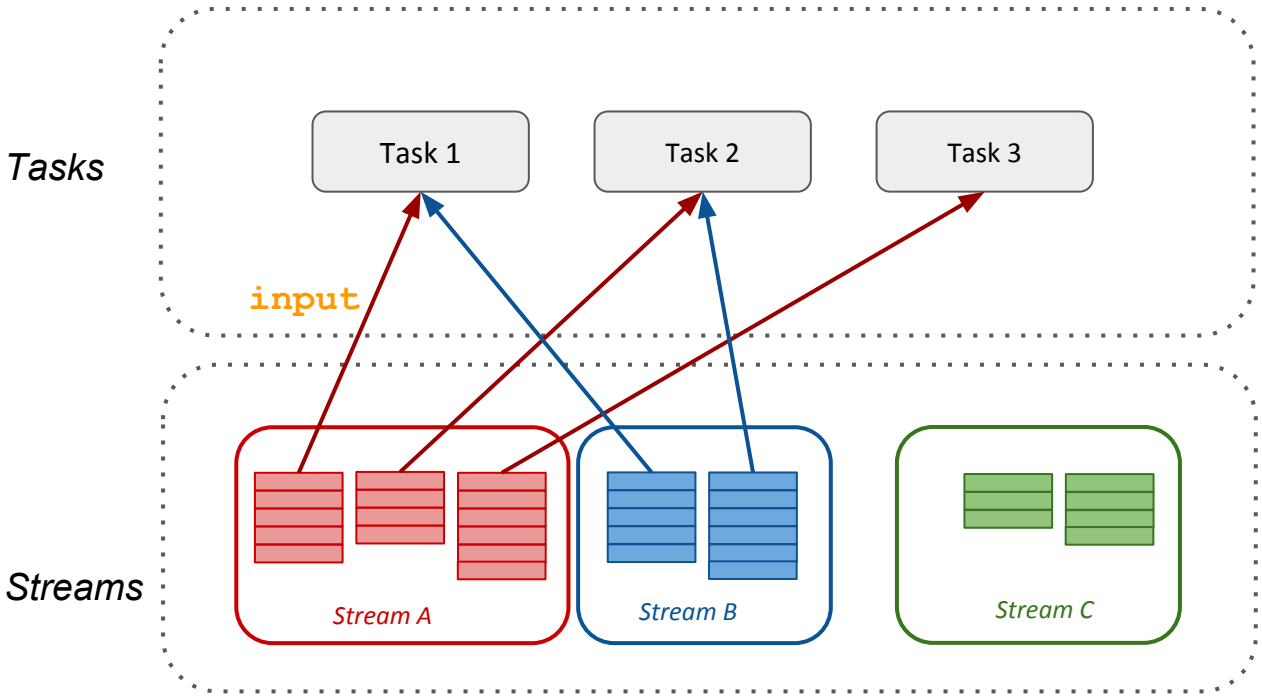
Streams



Apache Samza

How it works?

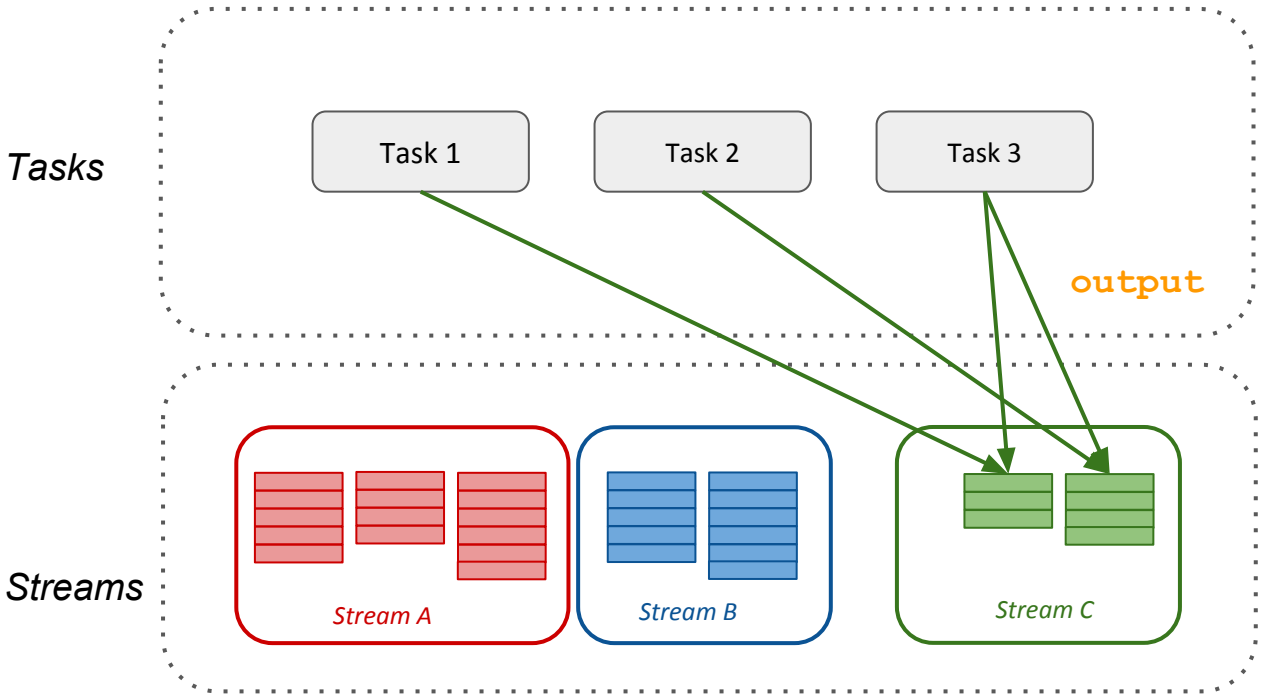
- Streams
- ***Tasks***
- Execution



Apache Samza

How it works?

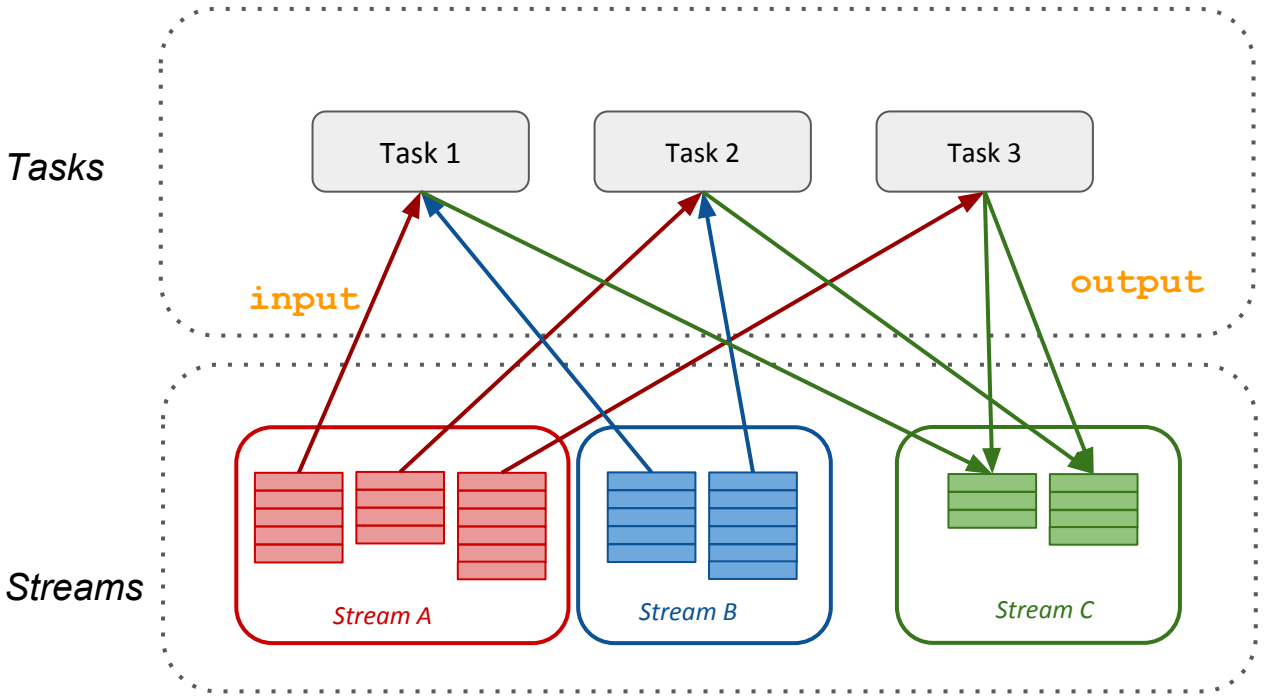
- Streams
- ***Tasks***
- Execution



Apache Samza

How it works?

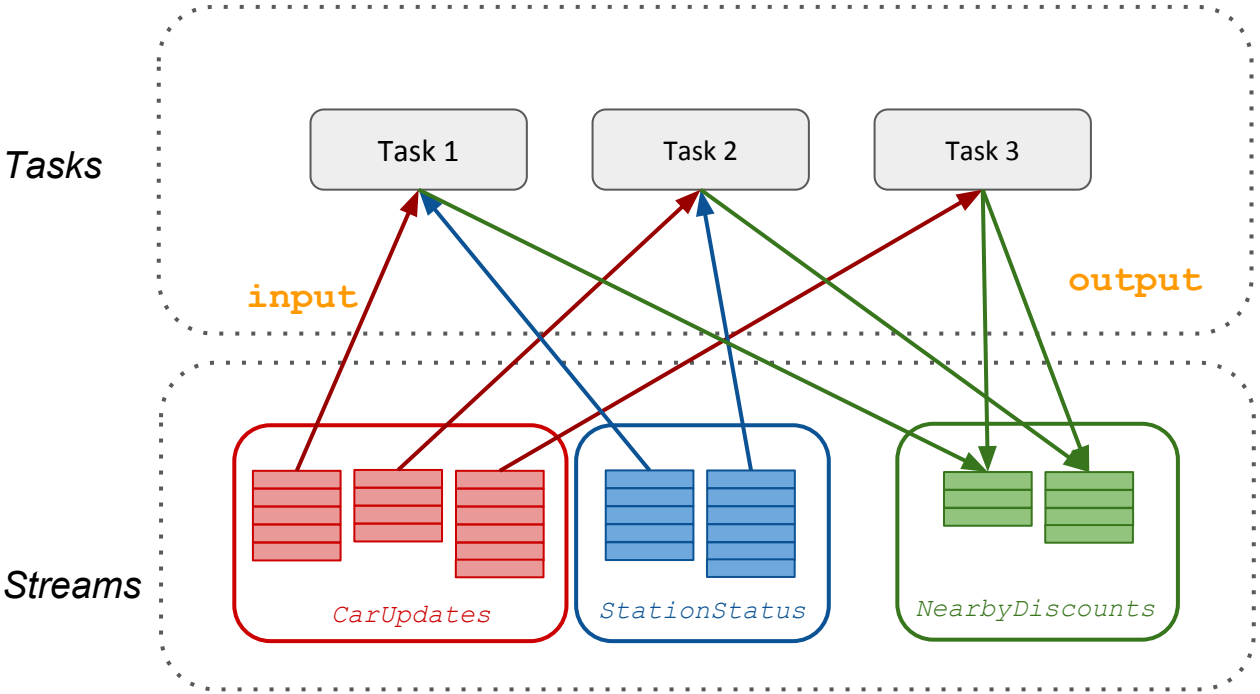
- Streams
- ***Tasks***
- Execution



Apache Samza

How it works?

- Streams
- ***Tasks***
- Execution



Apache Samza

How it works?

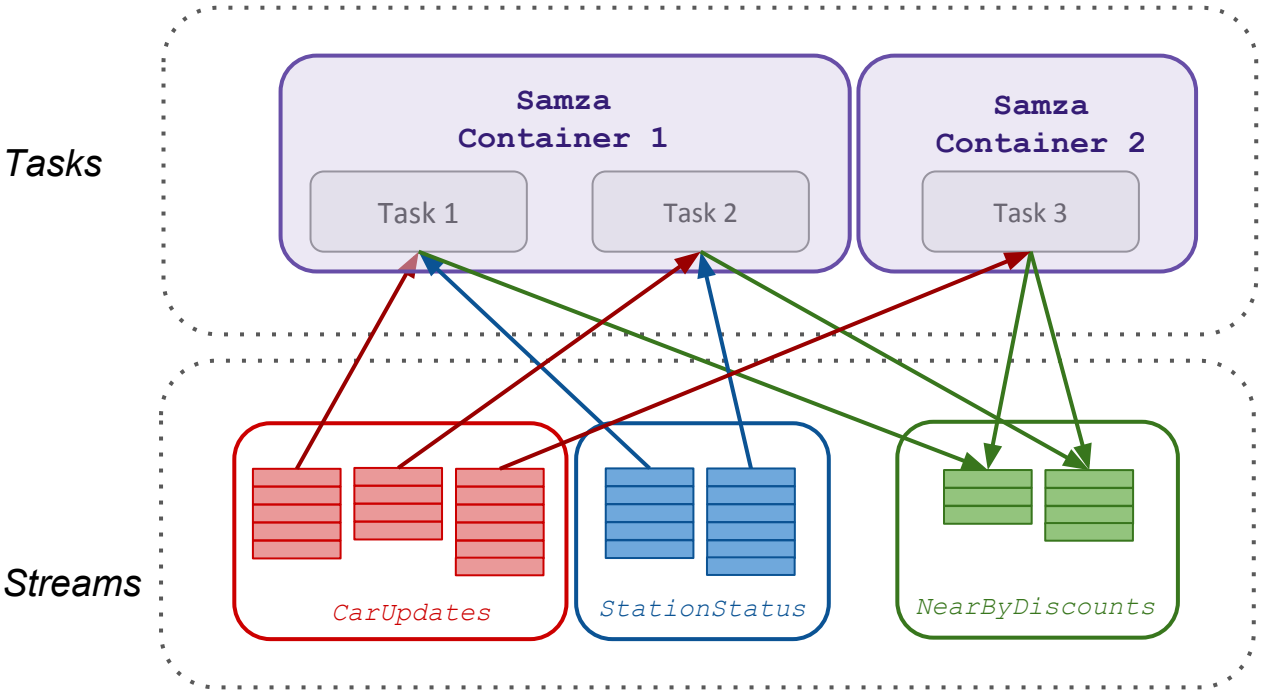
- Streams
- ***Tasks***
- Execution

```
Class MyExampleTask implements StreamTask {
      public void process (IncomingMessageEnvelope env,
                           MessageCollector col,
                           TaskCoordinator coord) {
           final TextMessage msg = (TextMessage) envelope.getMessage();
            Map<String, Object> outMap = new HashMap<>(){{
                  put(msg.getMessageID(), msg.getText());
            }};
           collector.send(new OutgoingMessageEnvelope(outStream, outMap));
      }
}
```

Apache Samza

How it works?

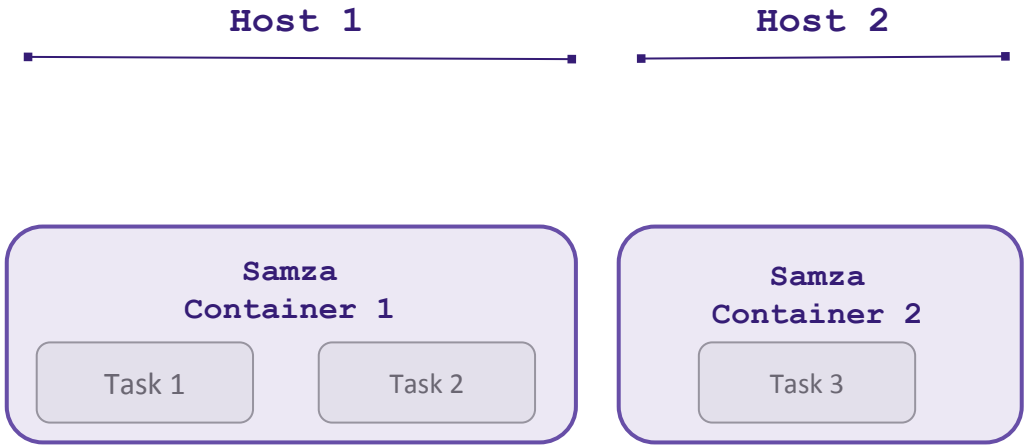
- Streams
- Tasks
- ***Execution***



Apache Samza

How it works?

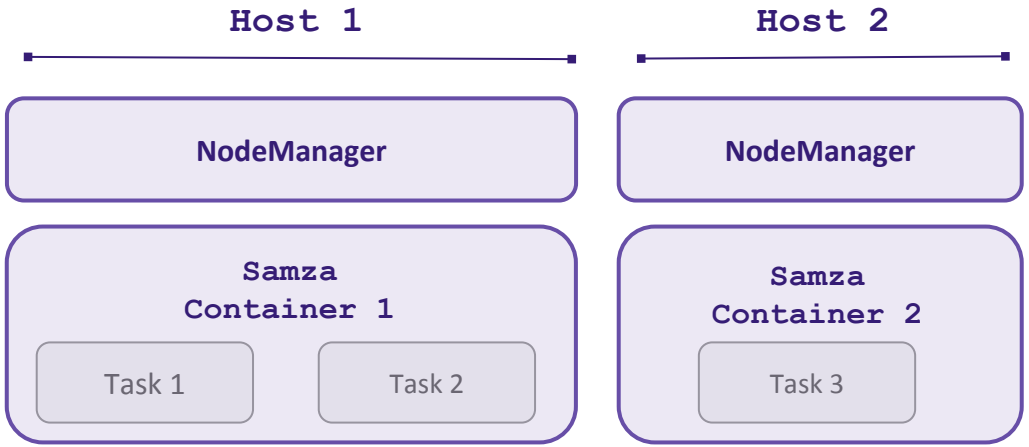
- Streams
- Tasks
- ***Execution***



Apache Samza

How it works?

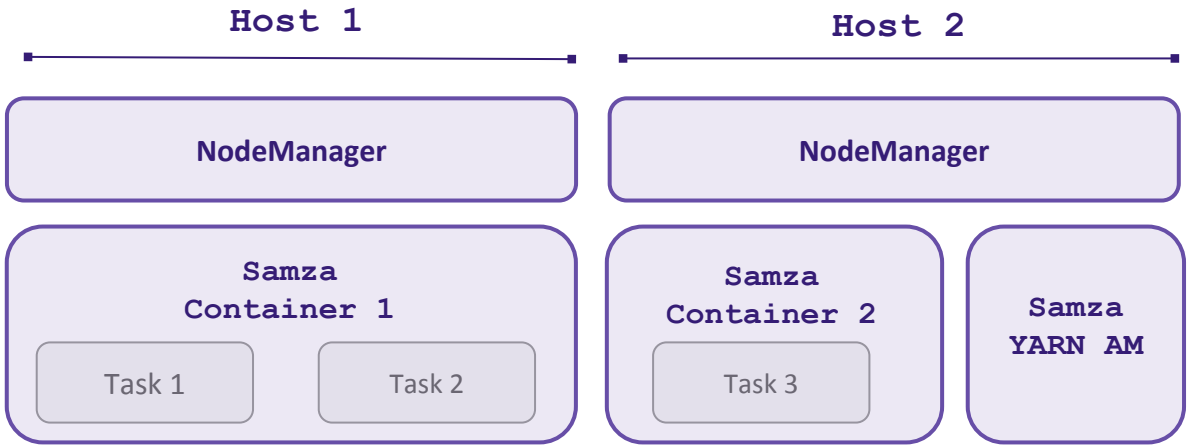
- Streams
- Tasks
- ***Execution***



Apache Samza

How it works?

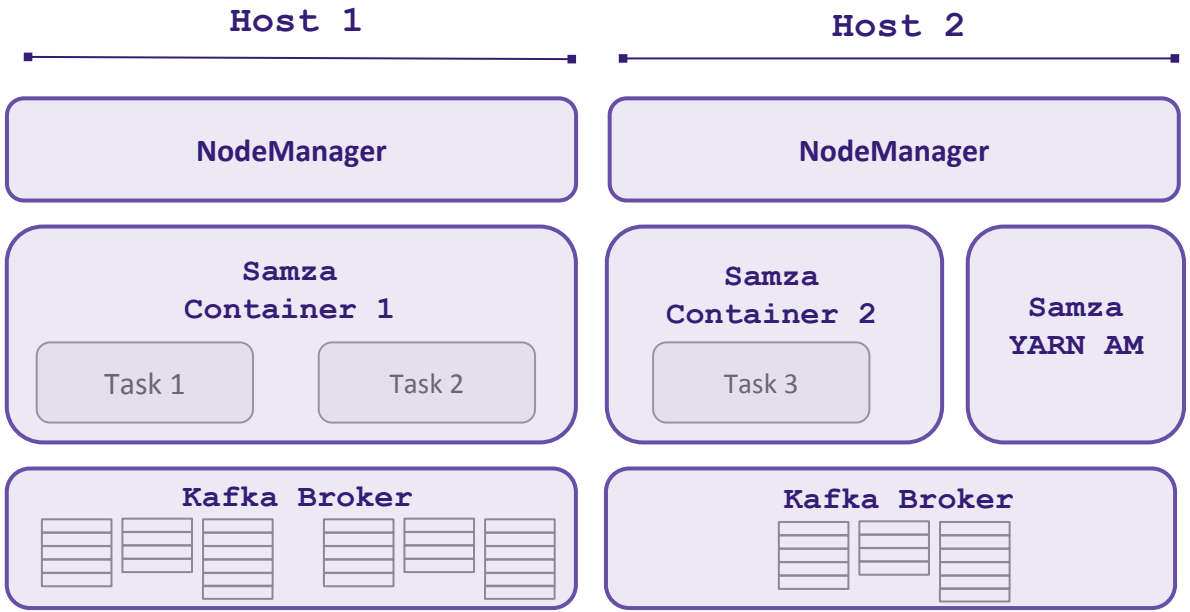
- Streams
- Tasks
- ***Execution***



Apache Samza

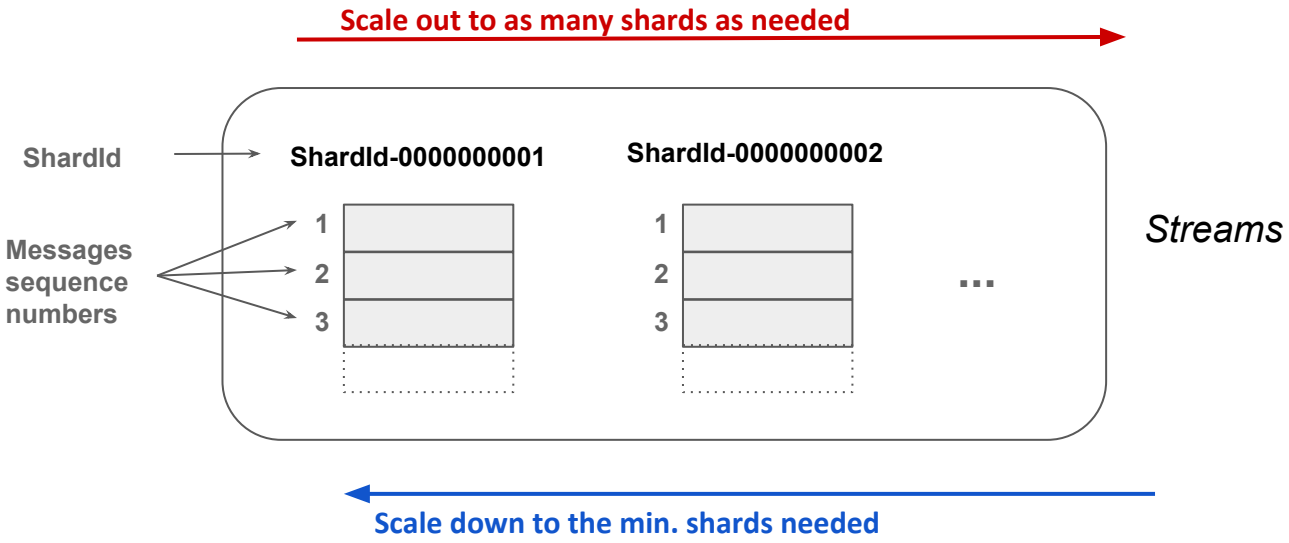
How it works?

- Streams
- Tasks
- ***Execution***



Amazon Kinesis

Introduction



Amazon Kinesis

- Two types of API
  - ***Amazon KCL***
  - Amazon Kinesis API

Amazon Kinesis

- Two types of API
  - ***Amazon KCL***
    - Specification for handling records: `IRecordProcessor` interface
    - Assigns a shard to an `IRecordProcessor`
    - Create a KCL Worker to consumer data
    - If many workers, they coordinate through Amazon Dynamo

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

  - Amazon Kinesis API

Amazon Kinesis

- Two types of API
  - ***Amazon KCL***
    - Specification for handling records: `IRecordProcessor` interface
    - Assigns a shard to an `IRecordProcessor`
    - Create a KCL Worker to consumer data
    - If many workers, they coordinate through Amazon Dynamo

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

Record processor

  - Amazon Kinesis API

Amazon Kinesis

- Two types of API
  - ***Amazon KCL***
    - Specification for handling records: `IRecordProcessor` interface
    - Assigns a shard to an `IRecordProcessor`
    - Create a KCL Worker to consumer data
    - If many workers, they coordinate through Amazon Dynamo

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

KCL worker

  - Amazon Kinesis API

Amazon Kinesis

- Two types of API
  - Amazon KCL
  - ***Amazon Kinesis API***

Amazon Kinesis

- Two types of API
  - Amazon KCL
  - ***Amazon Kinesis API***
    - Get data from Kinesis shards
    - To iterate over shards: `getNextShardIterator`
    - To get shard iterator: `getShardIterator`

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);
```

Configure Kinesis request

```
GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Amazon Kinesis

- Two types of API
  - Amazon KCL
  - ***Amazon Kinesis API***
    - Get data from Kinesis shards
    - To iterate over shards: `getNextShardIterator`
    - To get shard iterator: `getShardIterator`

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Executing the request

Integration story

```
public interface SystemConsumer {
  void start();

  void stop();

  void register( SystemStreamPartition systemStreamPartition, String lastReadOffset);

  List<IncomingMessageEnvelope> poll(
      Map<SystemStreamPartition, Integer> systemStreamPartitions,
      long timeout) throws InterruptedException;
}
```

Integration story

```
public interface SystemProducer {
  void start();

  void stop();

  void register(String source);

  void send(String source, OutgoingMessageEnvelope envelope);

  void flush(String source);
}
```

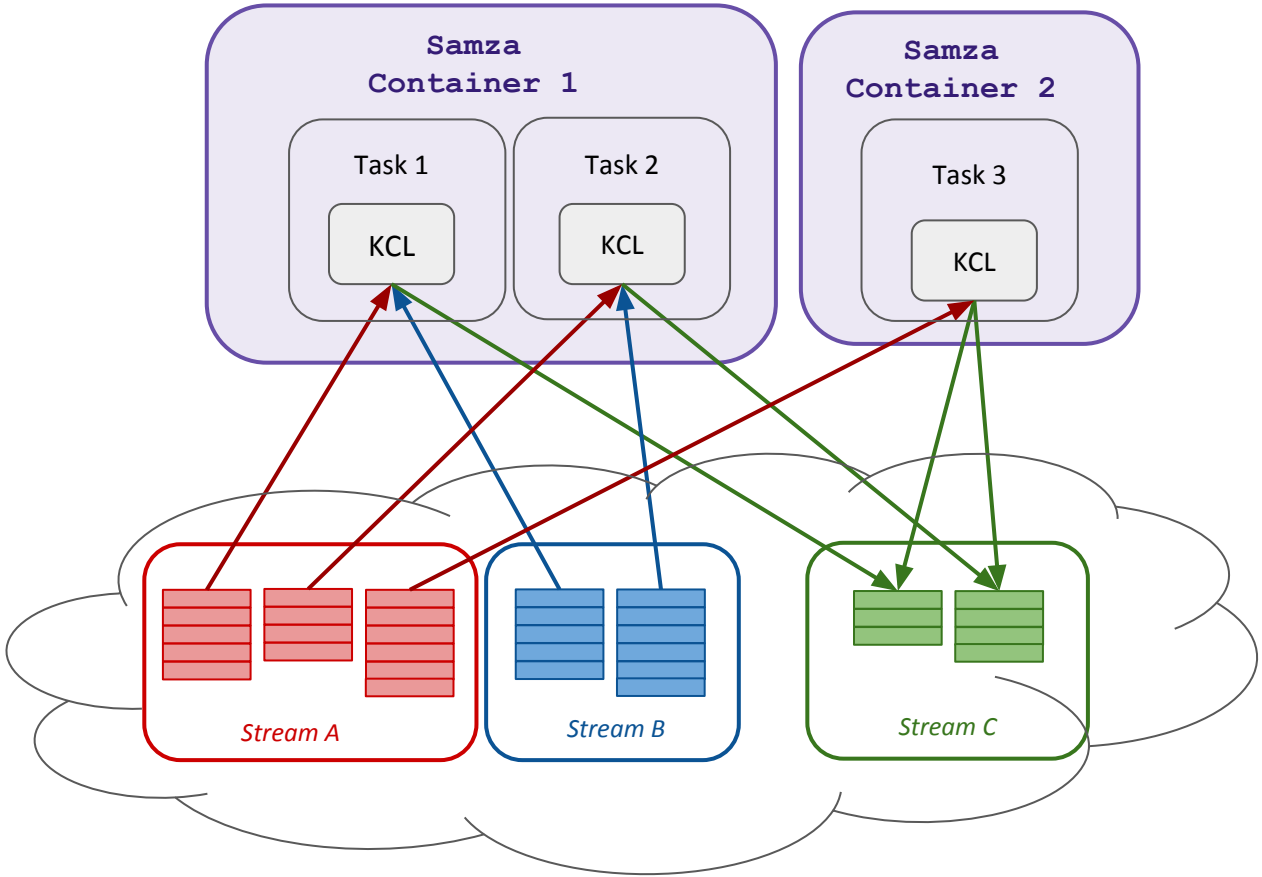
Integration story

- Map each Kinesis shard → Samza's logical partition
- Remember → Messages coming from a specific partition go to:
  - Particular task
  - Inside a specific container
- Approach 1:
  - Auto-scaling
  - Load balancing
- Approach 2:
  - Correctness
  - At-least once guarantees

Integration story

Attempt 1: KCL

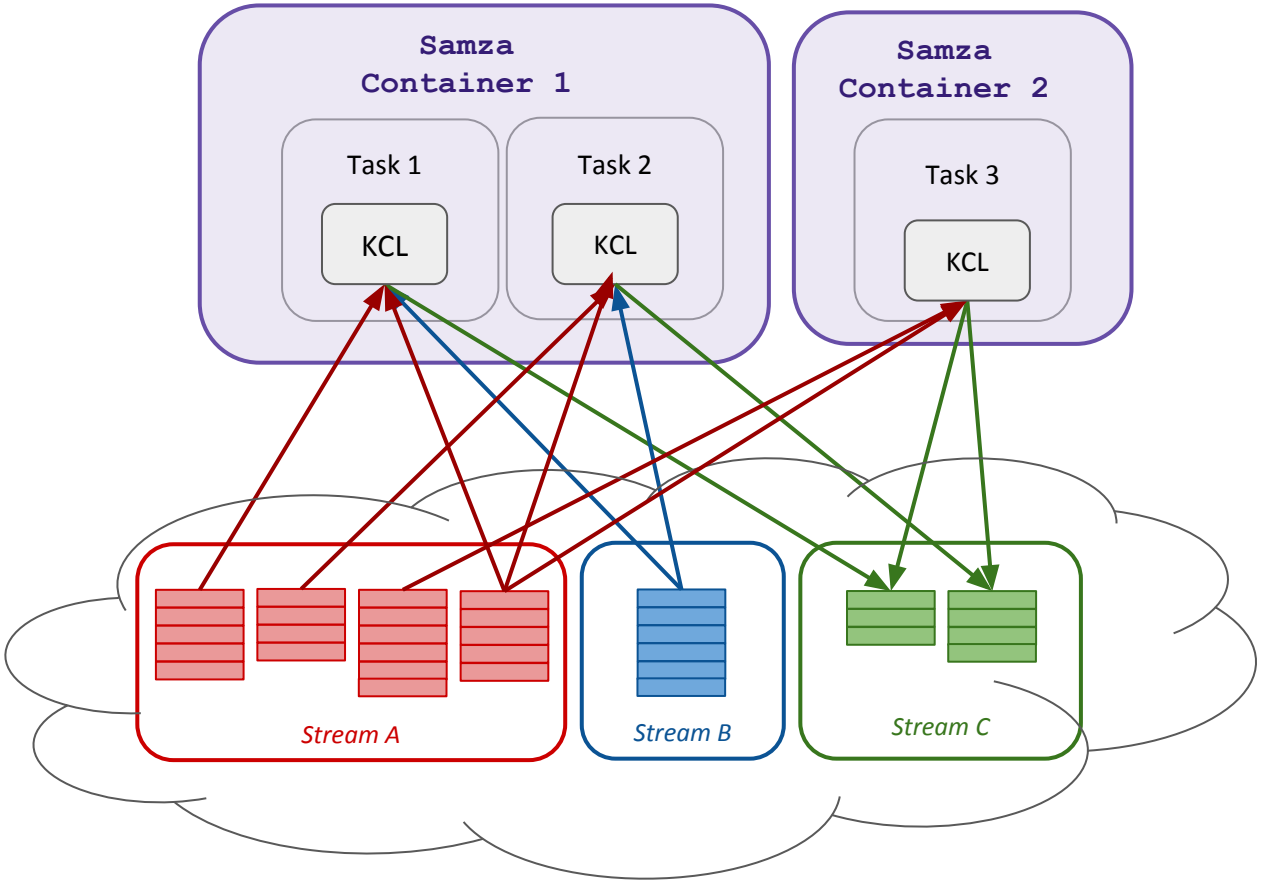
- Every task uses its own KCL



Integration story

Attempt 1: KCL

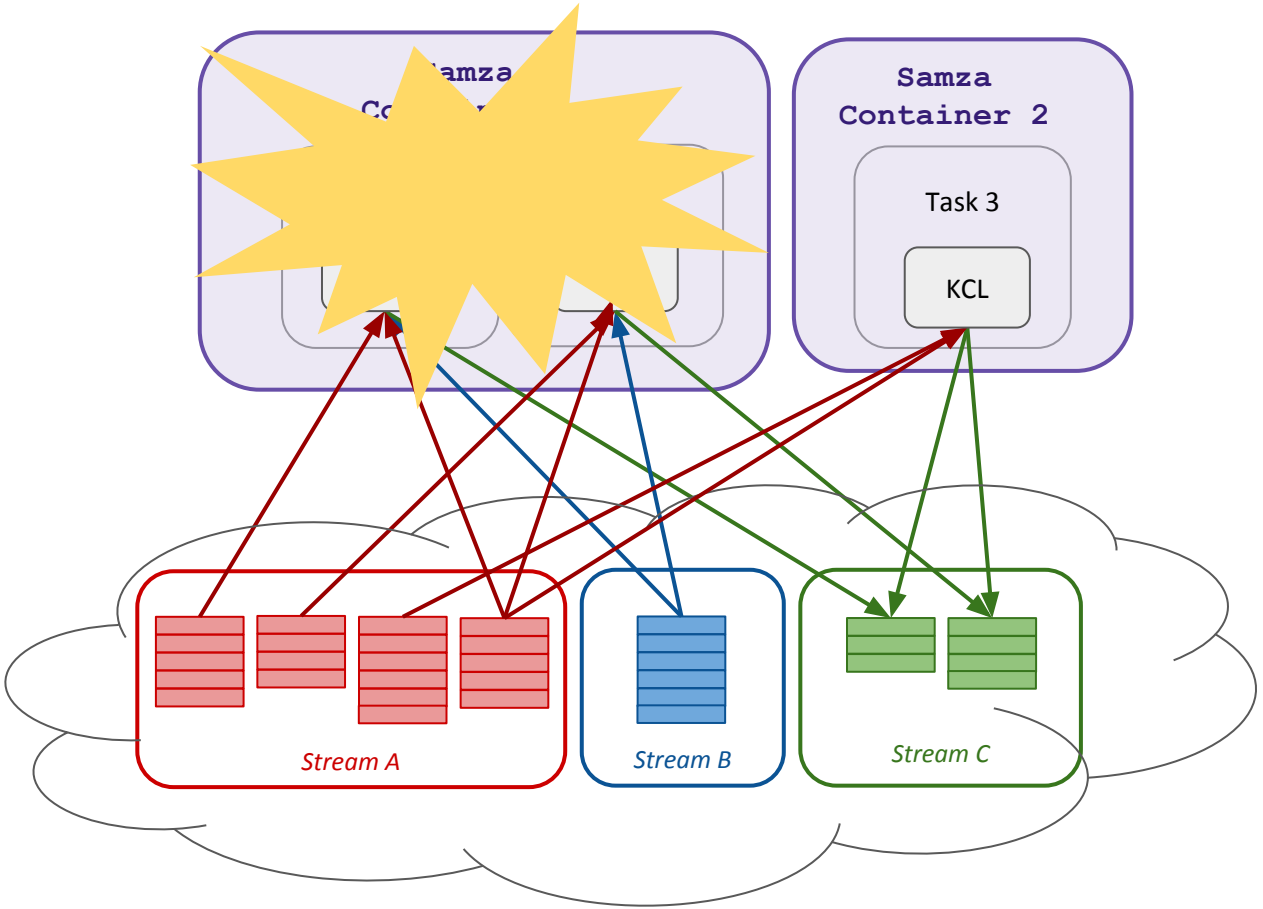
- Every task uses its own KCL
- KCL is in charge of *auto-scaling* and fault-tolerance



Integration story

Attempt 1: KCL

- Every task uses its own KCL
- KCL is in charge of auto-scaling and *fault-tolerance*

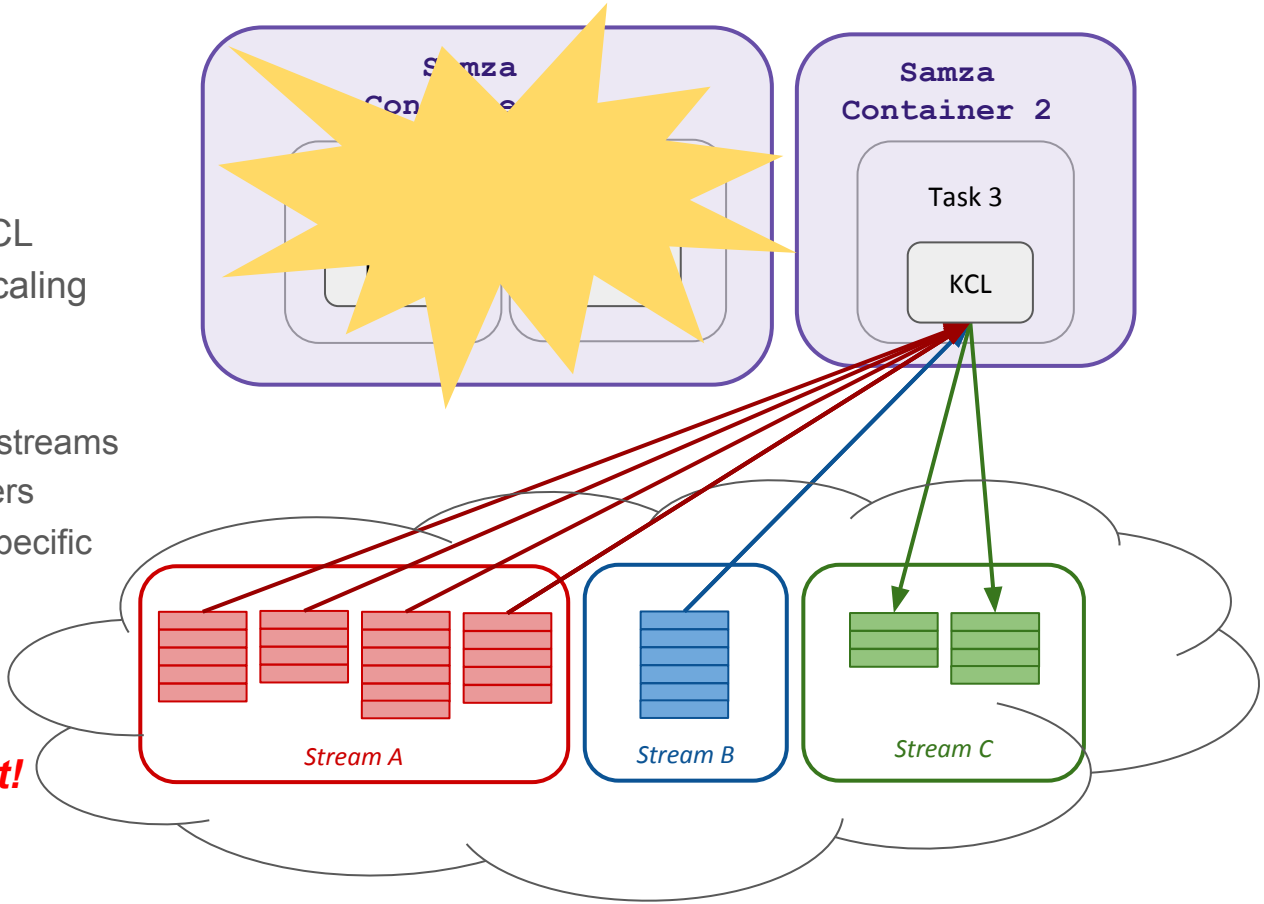


Integration story

Attempt 1: KCL

- Every task uses its own KCL
- KCL is in charge of auto-scaling and *fault-tolerance*
- But
  - Fix mapping between streams partitions and containers
  - Msgs should go to a specific container

- ***Messages could be lost!***

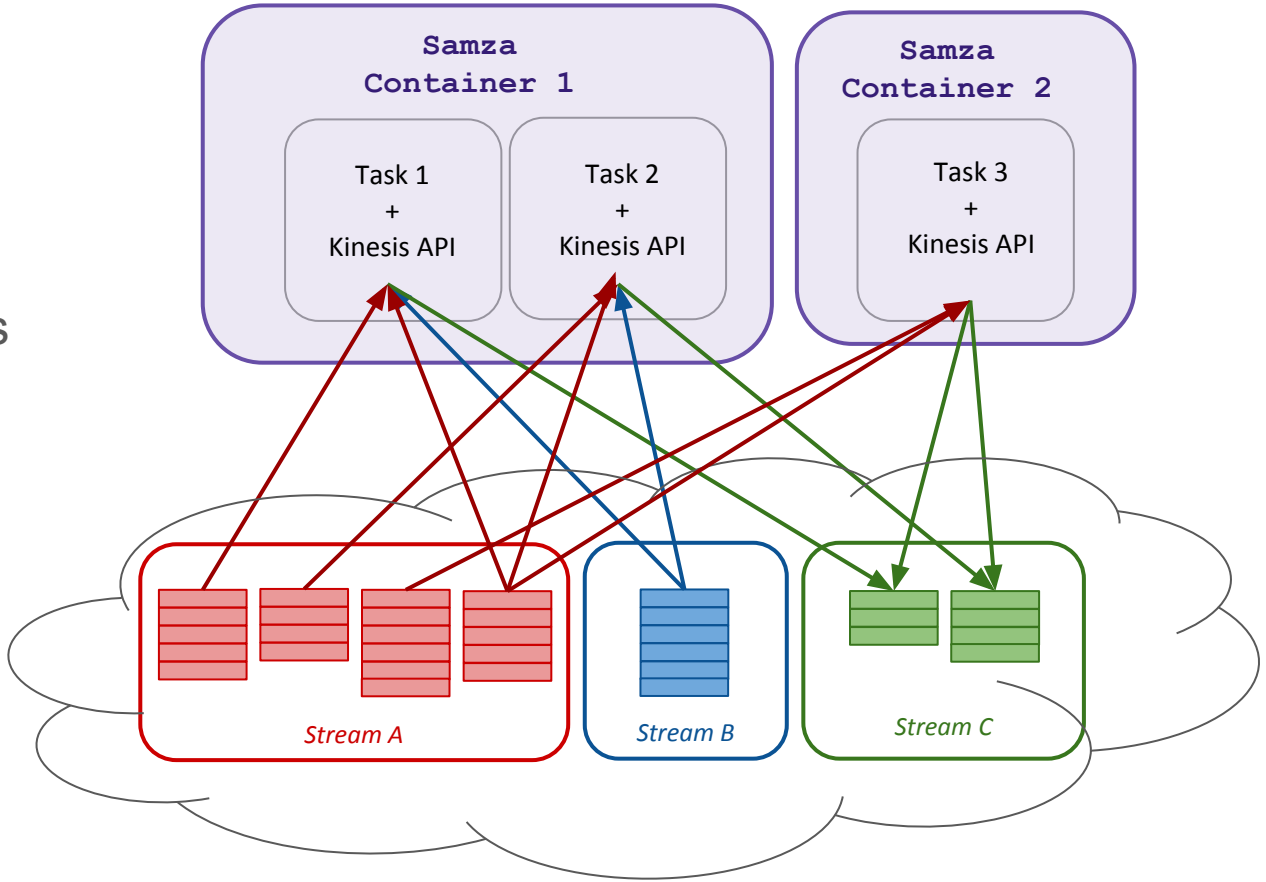


Integration story

Attempt 2:

Amazon Kinesis API

- Assuring correctness
- Letting Samza deal with fault-tolerance and load balancing



Integration story

Attempt 2: Amazon Kinesis API

- Register each Kinesis shard as Samza's partitions.
  - Involves creating a **`shardIterator`** per partition
- Create a **`GetShardIteratorRequest`** per partition to fetch data
  - From the beginning of the stream
  - From a specific sequence number
- Keep track of each record received from Kinesis
- Checkpoint in Samza

Integration story

```
 // Create a new getRecordsRequest with an existing shardIterator
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
GetRecordsResult result = kClient.getRecords(getRecordsRequest);

// Put the result into record list.
for (Record record : result.getRecords()) {
   IncomingMessageEnvelope envelope = new IncomingMessageEnvelope(ssp,
                                                  record.getSequenceNumber(),
                                                  record.getPartitionKey(),
                                                  record.getData());
   put(ssp, envelope);
   trackDeliveries(ssp.getName(), envelope);
}
```

Integration story

```
 // Create a new getRecordsRequest with an existing shardIterator
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
GetRecordsResult result = kClient.getRecords(getRecordsRequest);

// Put the result into record list.
for (Record record : result.getRecords()) {
   IncomingMessageEnvelope envelope = new IncomingMessageEnvelope(ssp,
                                                  record.getSequenceNumber(),
                                                  record.getPartitionKey(),
                                                  record.getData());
   put(ssp, envelope);
   trackDeliveries(ssp.getName(), envelope);
}
```

Creating a handle to Kinesis shards

Integration story

```
 // Create a new getRecordsRequest with an existing shardIterator
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
GetRecordsResult result = kClient.getRecords(getRecordsRequest);

// Put the result into record list.
for (Record record : result.getRecords()) {
   IncomingMessageEnvelope envelope = new IncomingMessageEnvelope(ssp,
                                                  record.getSequenceNumber(),
                                                  record.getPartitionKey(),
                                                  record.getData());
   put(ssp, envelope);
   trackDeliveries(ssp.getName(), envelope);
}
```

Executing the request

Integration story

```
 // Create a new getRecordsRequest with an existing shardIterator
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
GetRecordsResult result = kClient.getRecords(getRecordsRequest);

// Put the result into record list.
for (Record record : result.getRecords()) {
   IncomingMessageEnvelope envelope = new IncomingMessageEnvelope(ssp,
                                                  record.getSequenceNumber(),
                                                  record.getPartitionKey(),
                                                  record.getData());
   put(ssp, envelope);
   trackDeliveries(ssp.getName(), envelope);
}
```

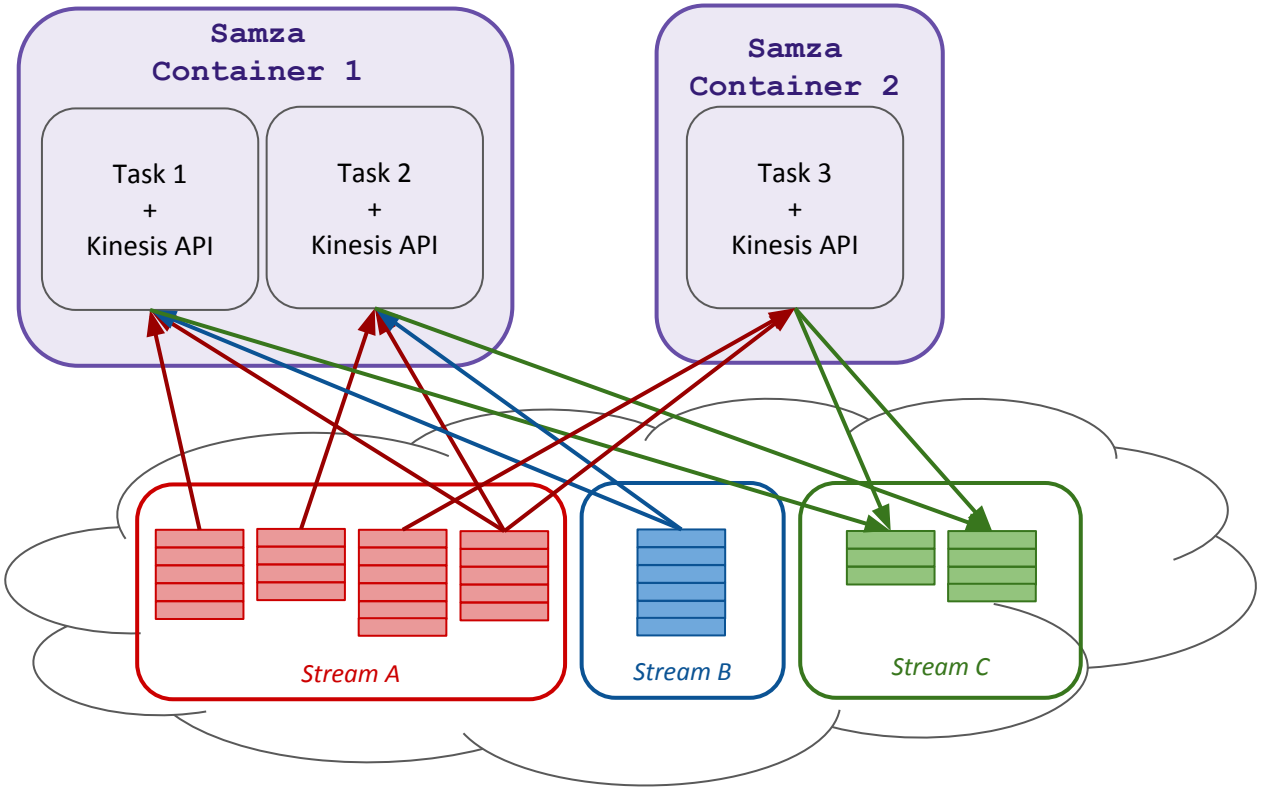
Passing messages to Samza

Integration story

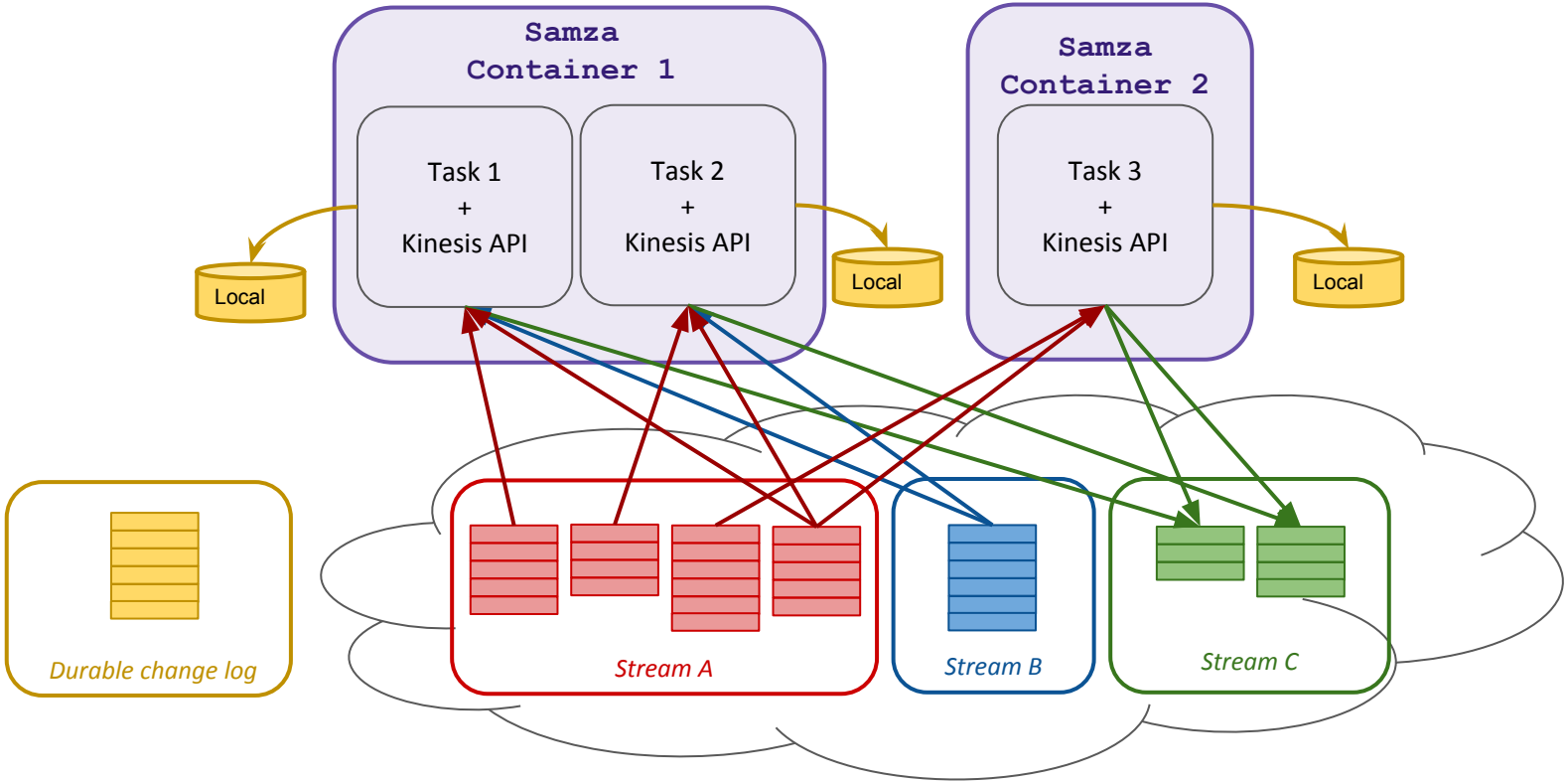
```
 // Create a new getRecordsRequest with an existing shardIterator
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
GetRecordsResult result = kClient.getRecords(getRecordsRequest);

// Put the result into record list.
for (Record record : result.getRecords()) {
   IncomingMessageEnvelope envelope = new IncomingMessageEnvelope(ssp,
                                                  record.getSequenceNumber(),
                                                  record.getPartitionKey(),
                                                  record.getData());

   put(ssp, envelope);
   trackDeliveries(ssp.getName(), envelope);
}
```

Keeping track of messages received

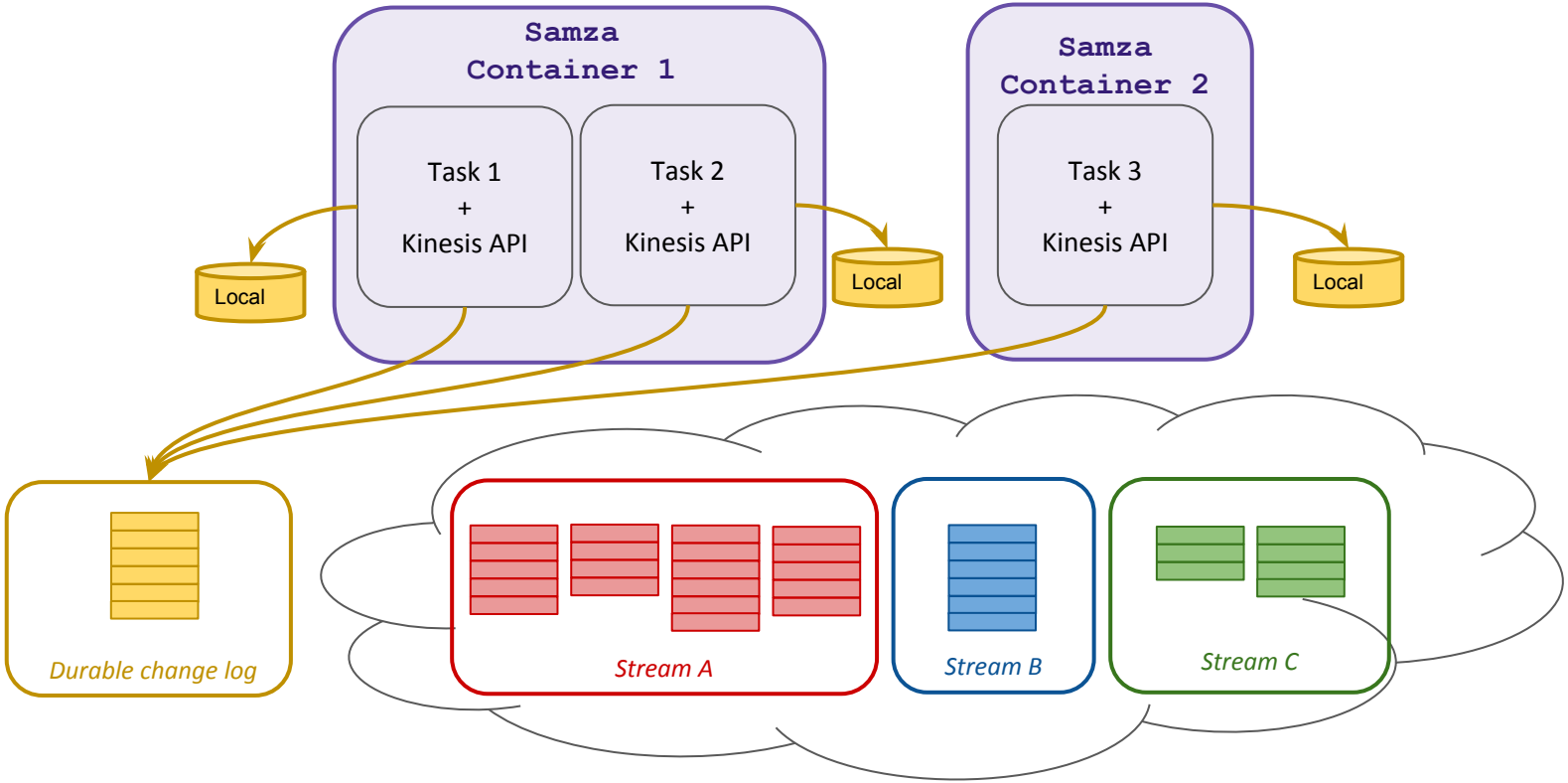
Integration story: Samza's fault tolerance



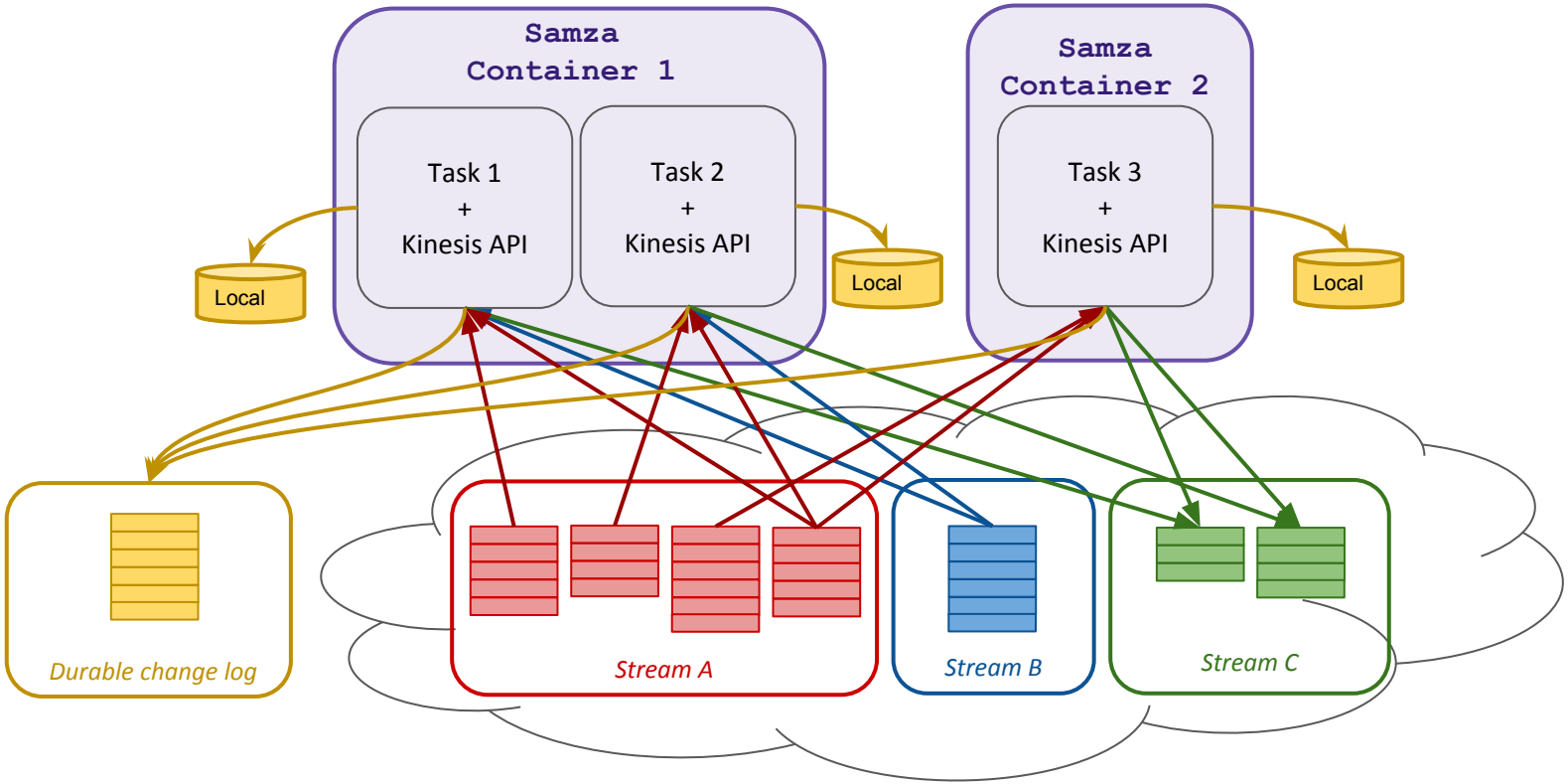
Integration story: Samza's fault tolerance



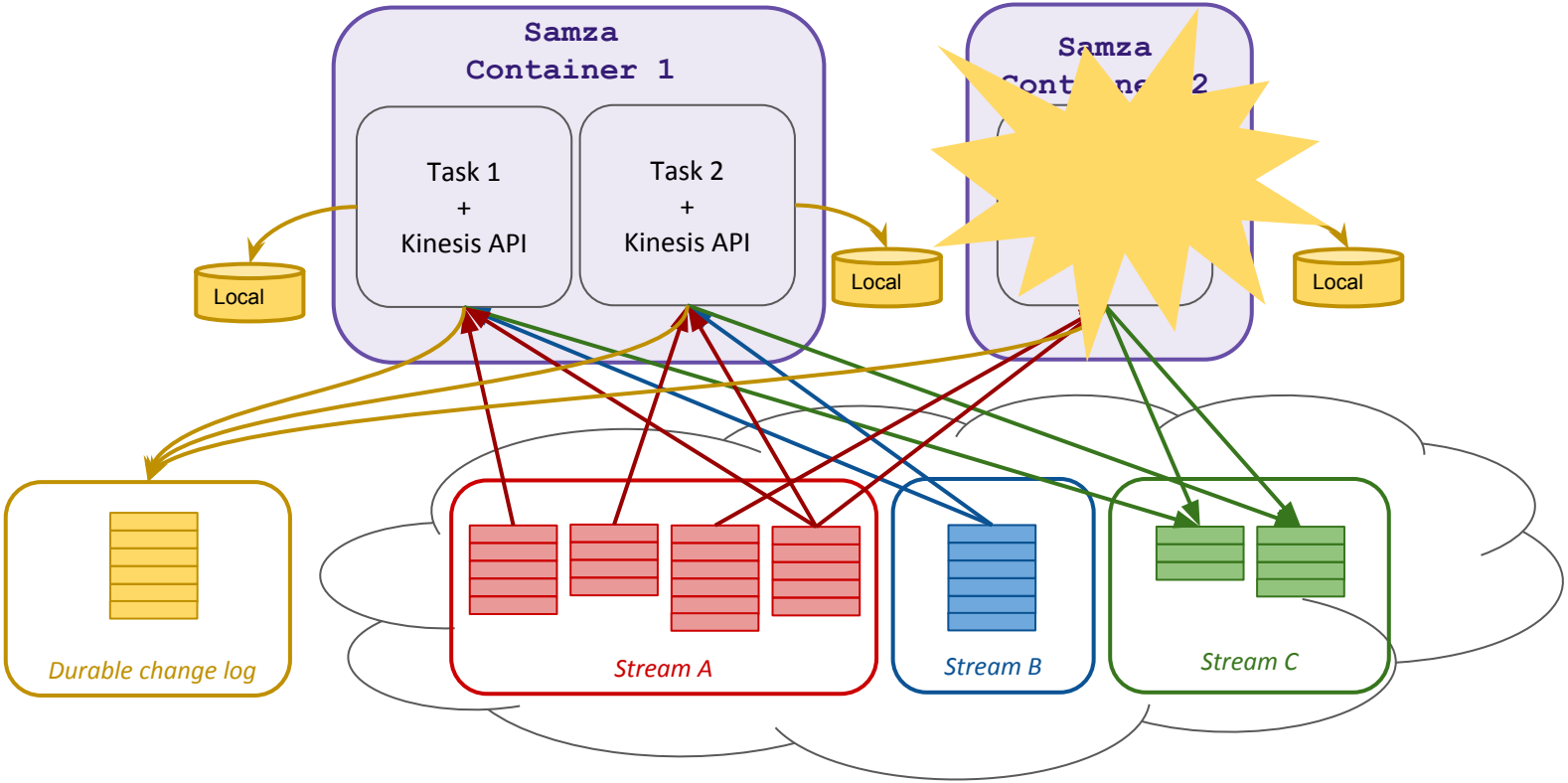
Integration story: Samza's fault tolerance



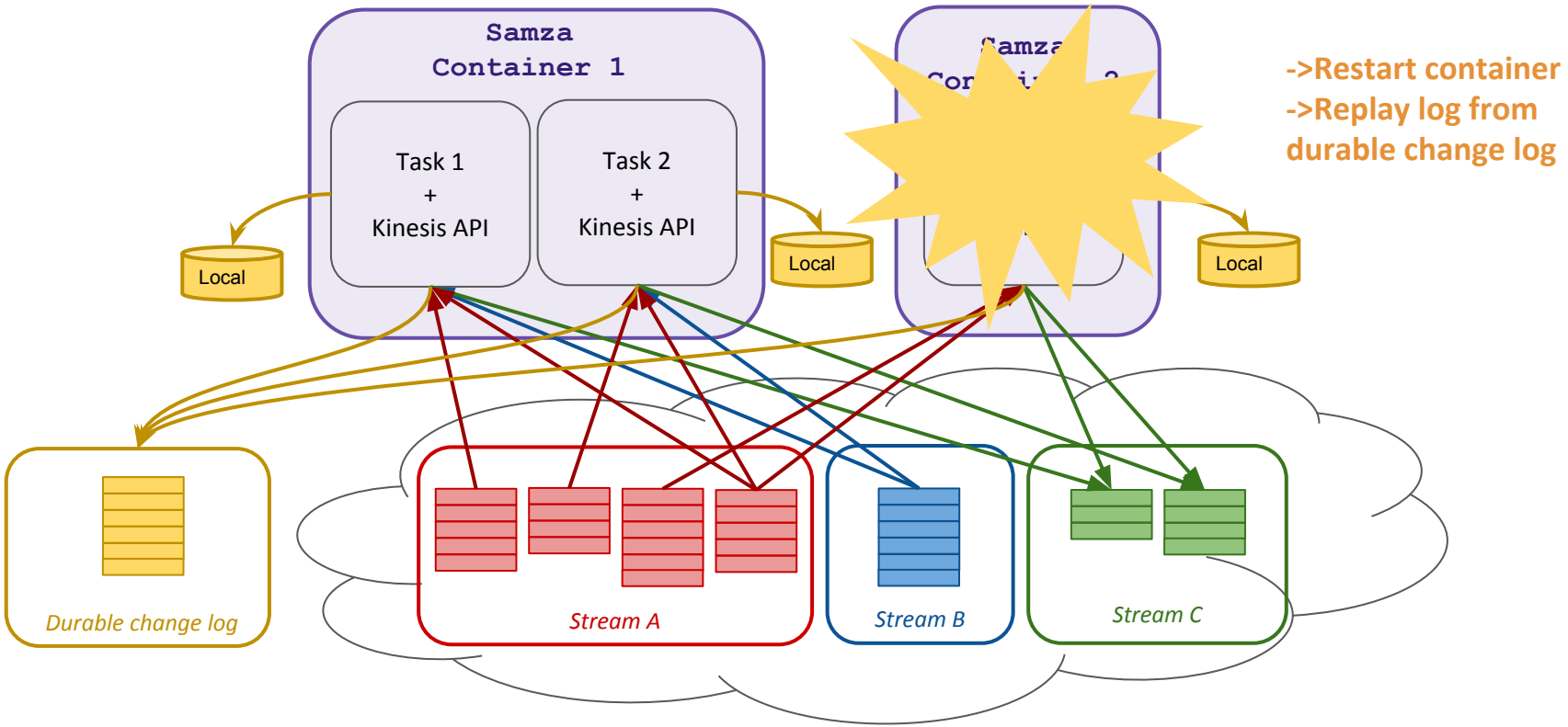
Integration story: Samza's fault tolerance



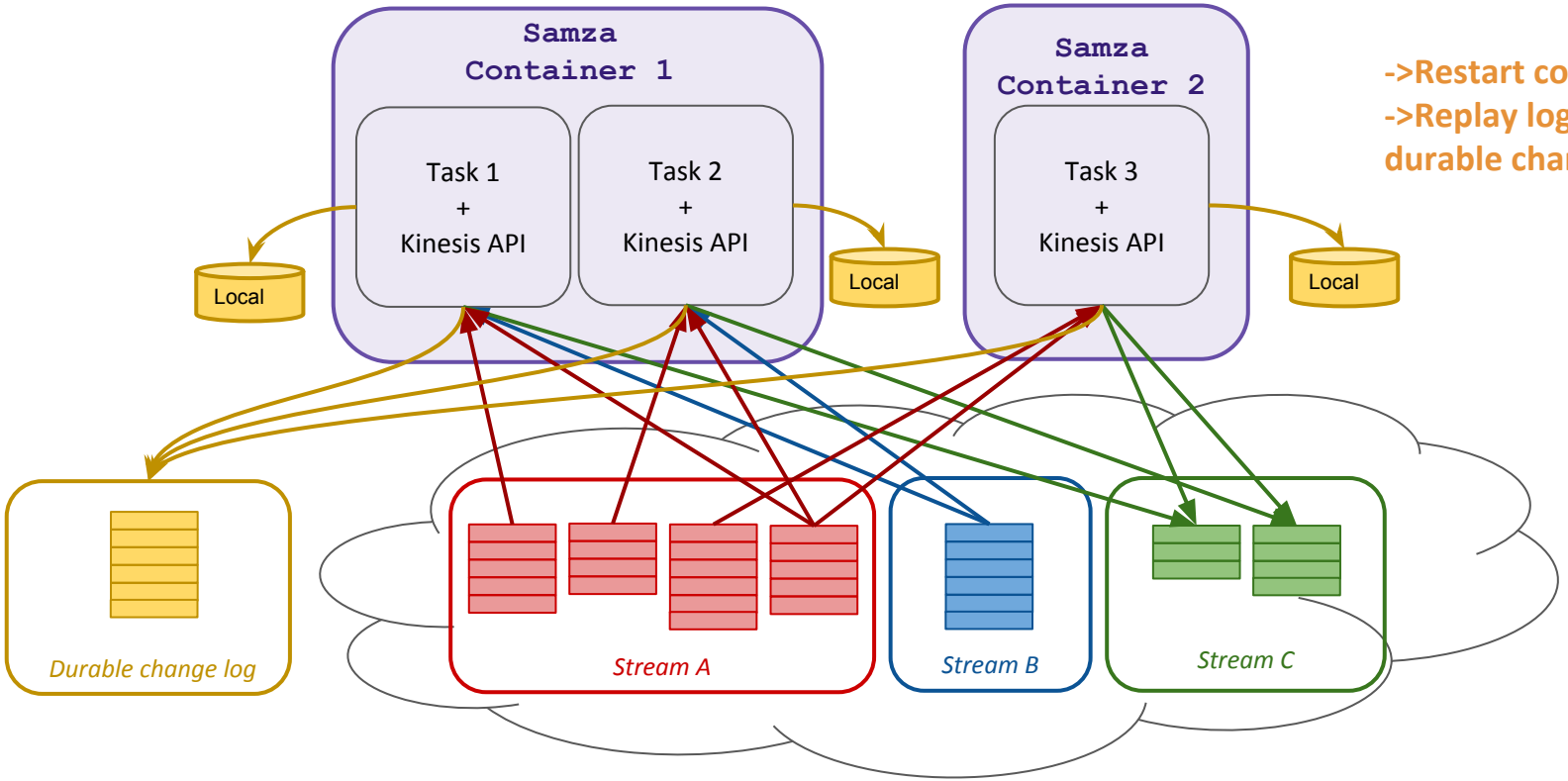
Integration story: Samza's fault tolerance



Integration story: Samza's fault tolerance



Integration story: Samza's fault tolerance



Apache Samza next moves

- Tighter integration with Apache Kafka
- Samza as
  - a *stream processing as a service*
  - the transformation layer for many services
- Pluggable mapping from streams/tasks/containers
- Different execution layers
  - Yarn is not always needed
  - Standalone mode
- Integrate with other systems

Thanks!

@renatomarroquin
rmarroquin [at] apache [dot] org