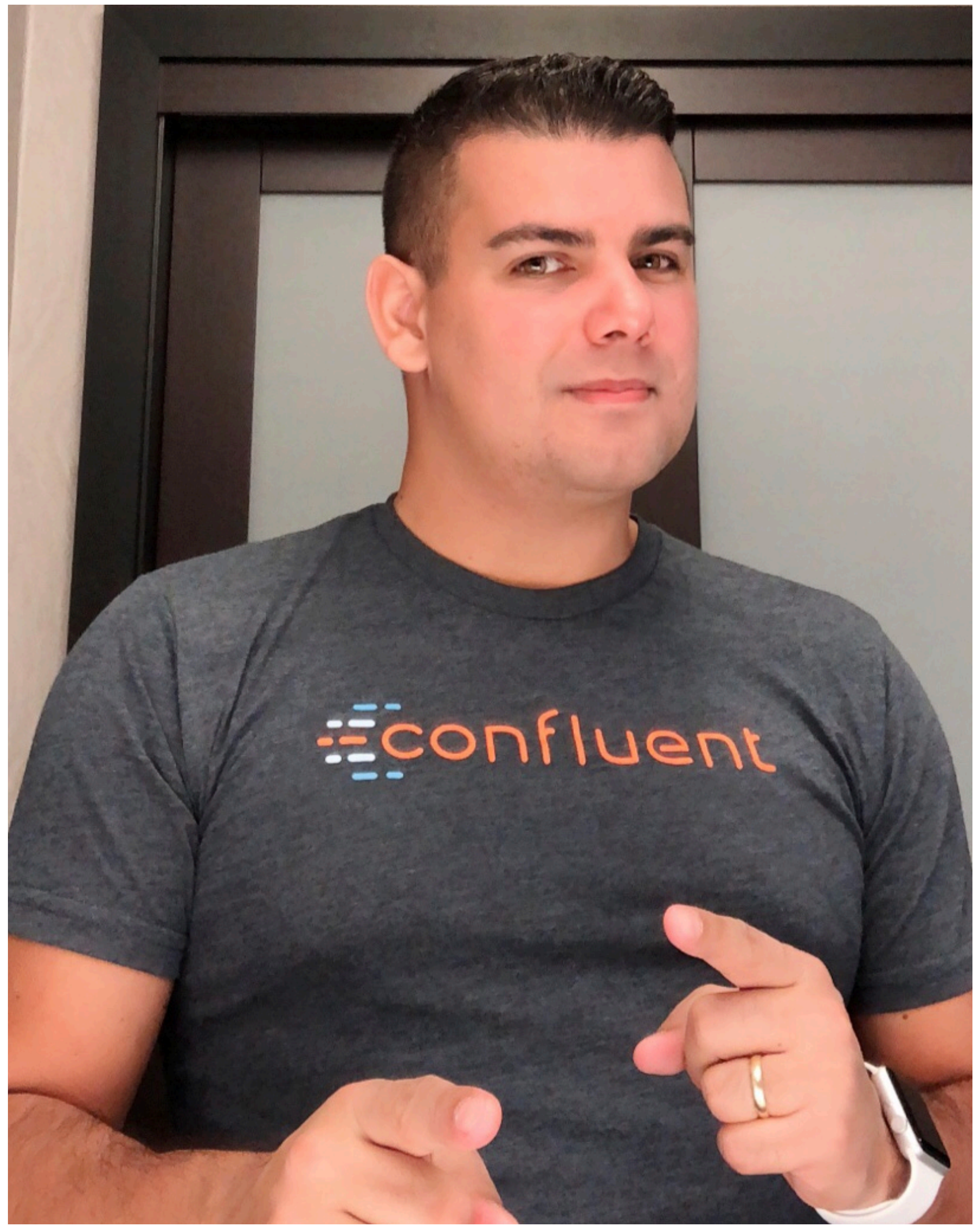
Crossing the Streams: Rethinking Stream Processing with Kafka Streams and KSQL

@gamussa @riferrei @confluentinc



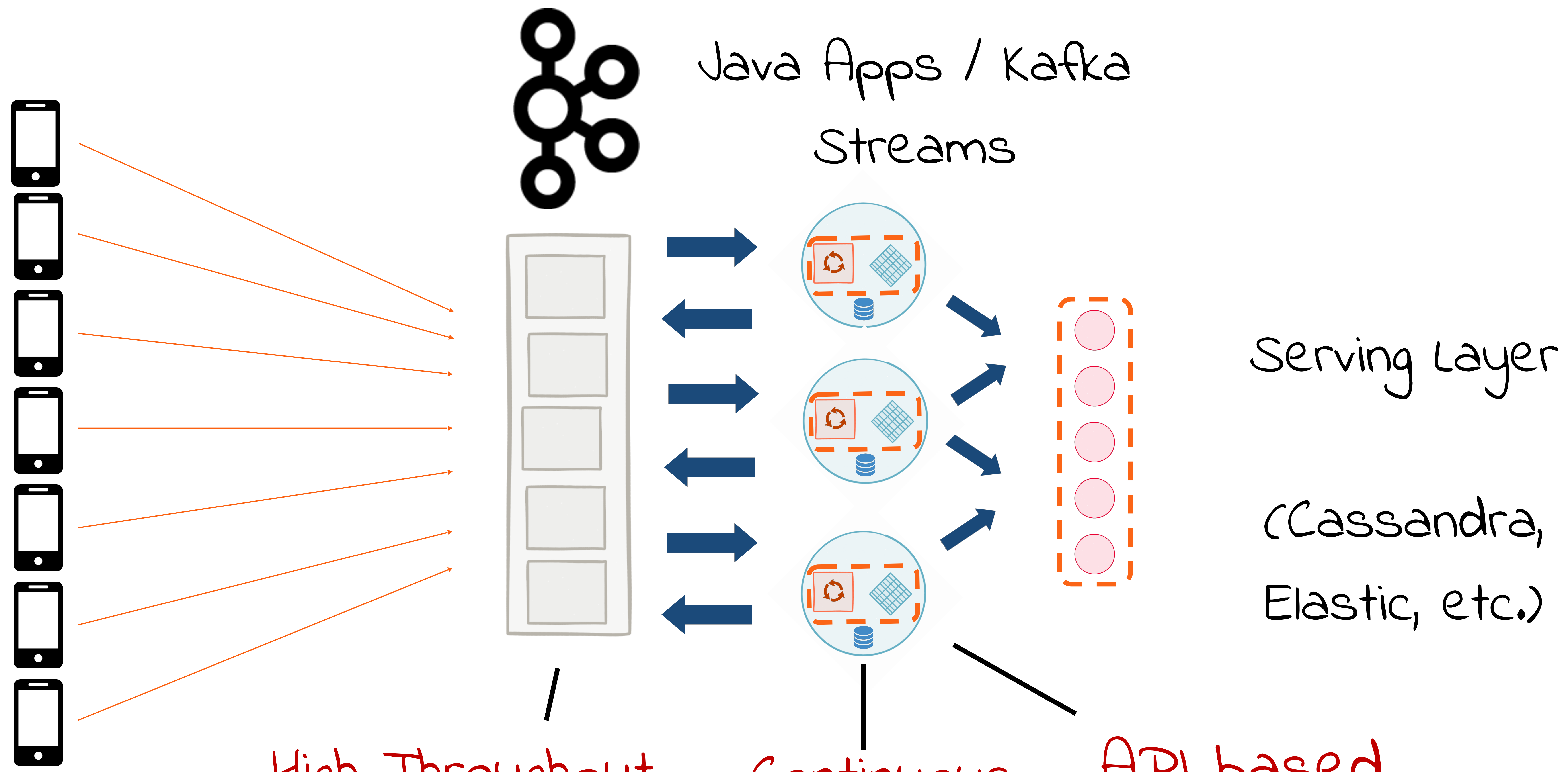
confluent @gamussa @riferrei @confluentinc

confluent

https://cnfl.io/streams-movie-workshop

Preface

Streaming is the toolset for dealing with events as they move!

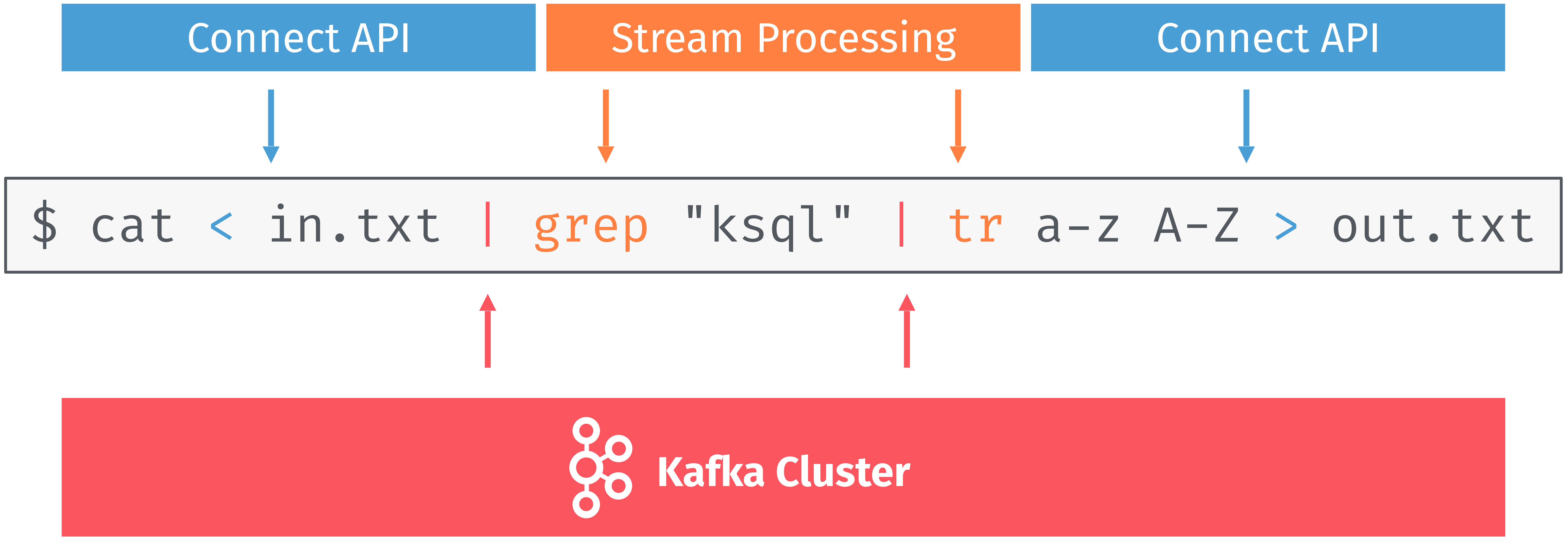


High Throughput Streaming platform

Continuous Computation

API based clustering

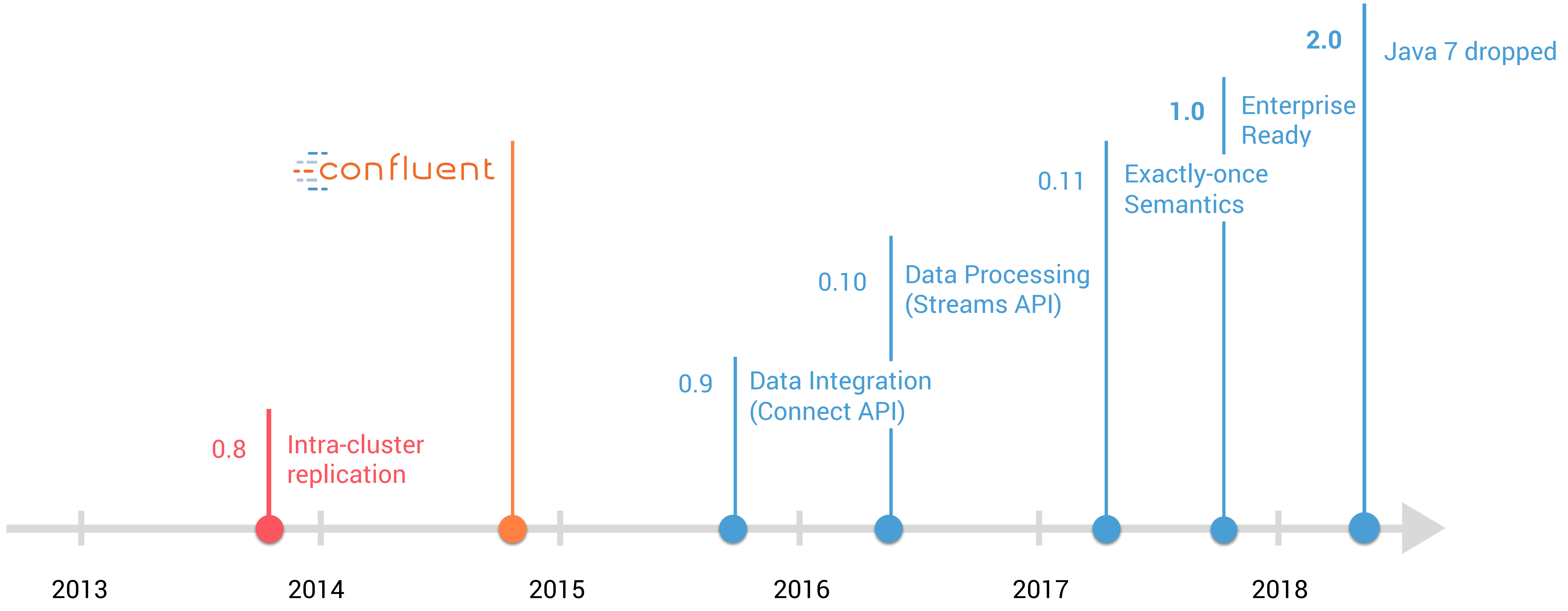
Stream Processing by Analogy



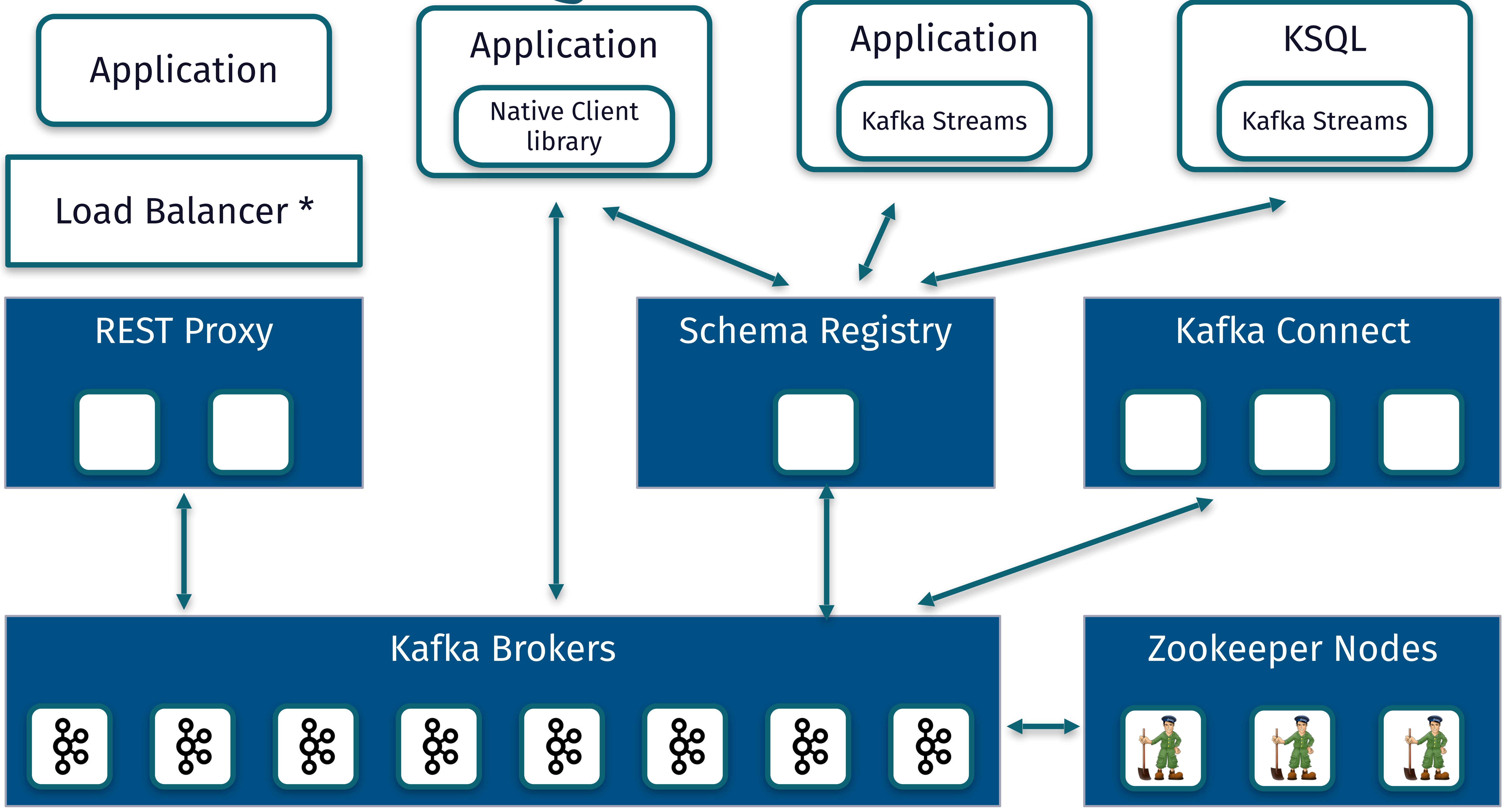
```
$ cat < in.txt | grep "ksql" | tr a-z A-Z > out.txt
```

Apache Kafka Event Streaming Platform 101

Kafka the Streaming Data Platform

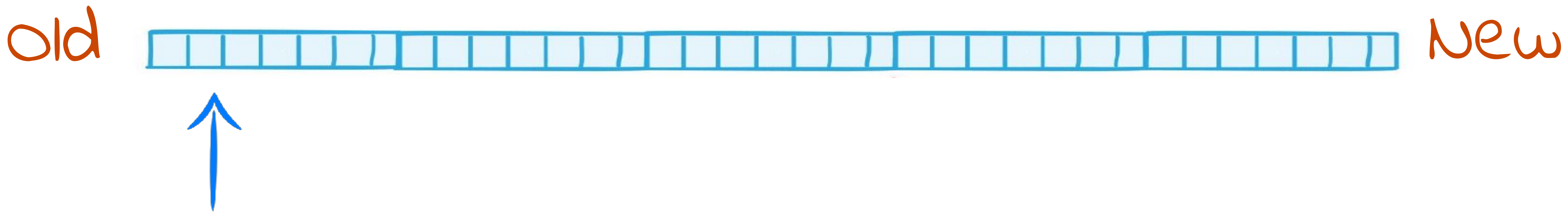


Event Streaming Platform Architecture



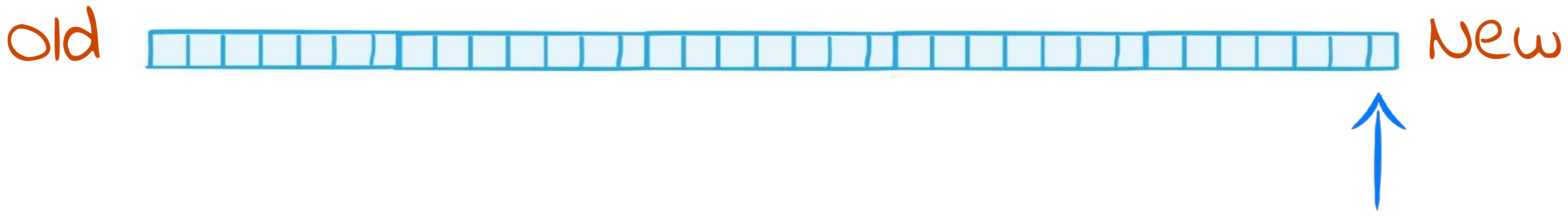
confluent @gamussa @riferrei @confluentinc

The log is a simple idea



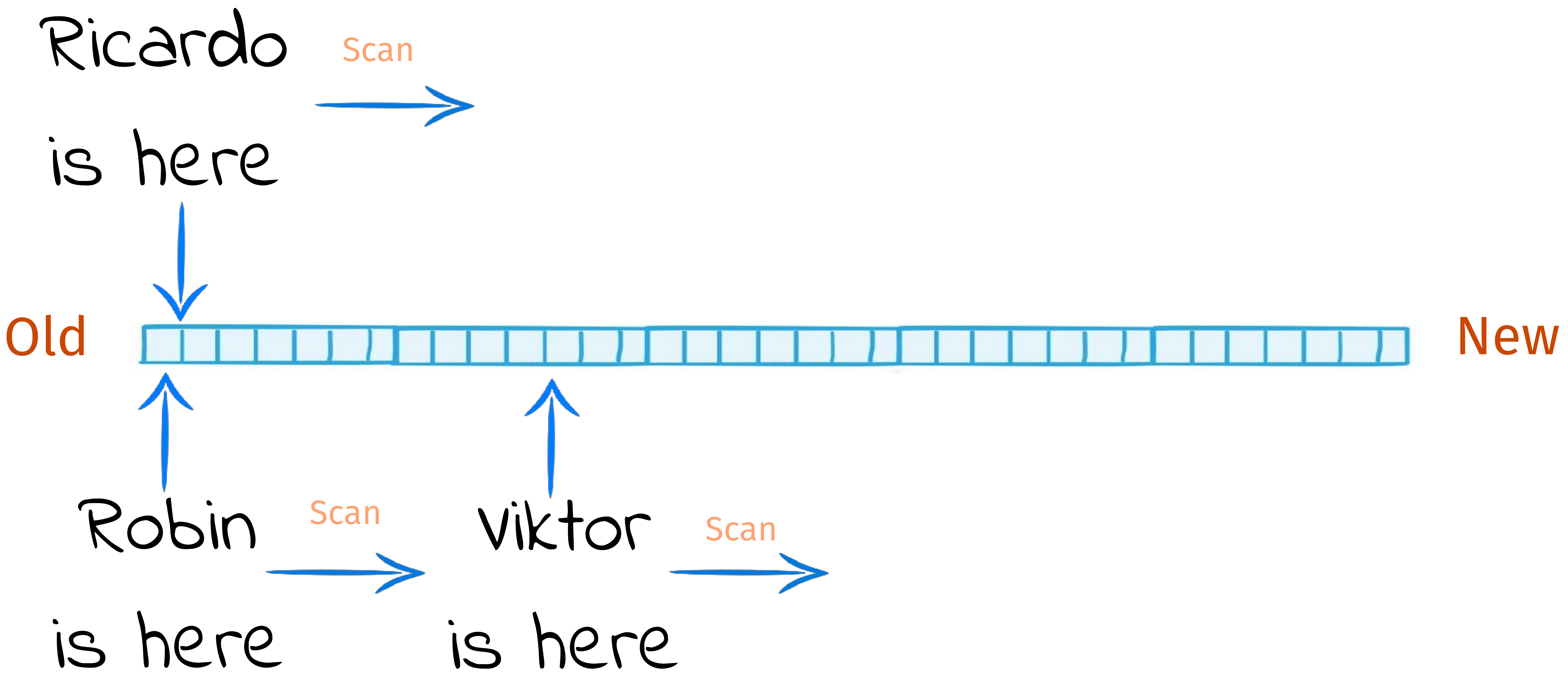
Messages are added at the end of the log

The log is a simple idea

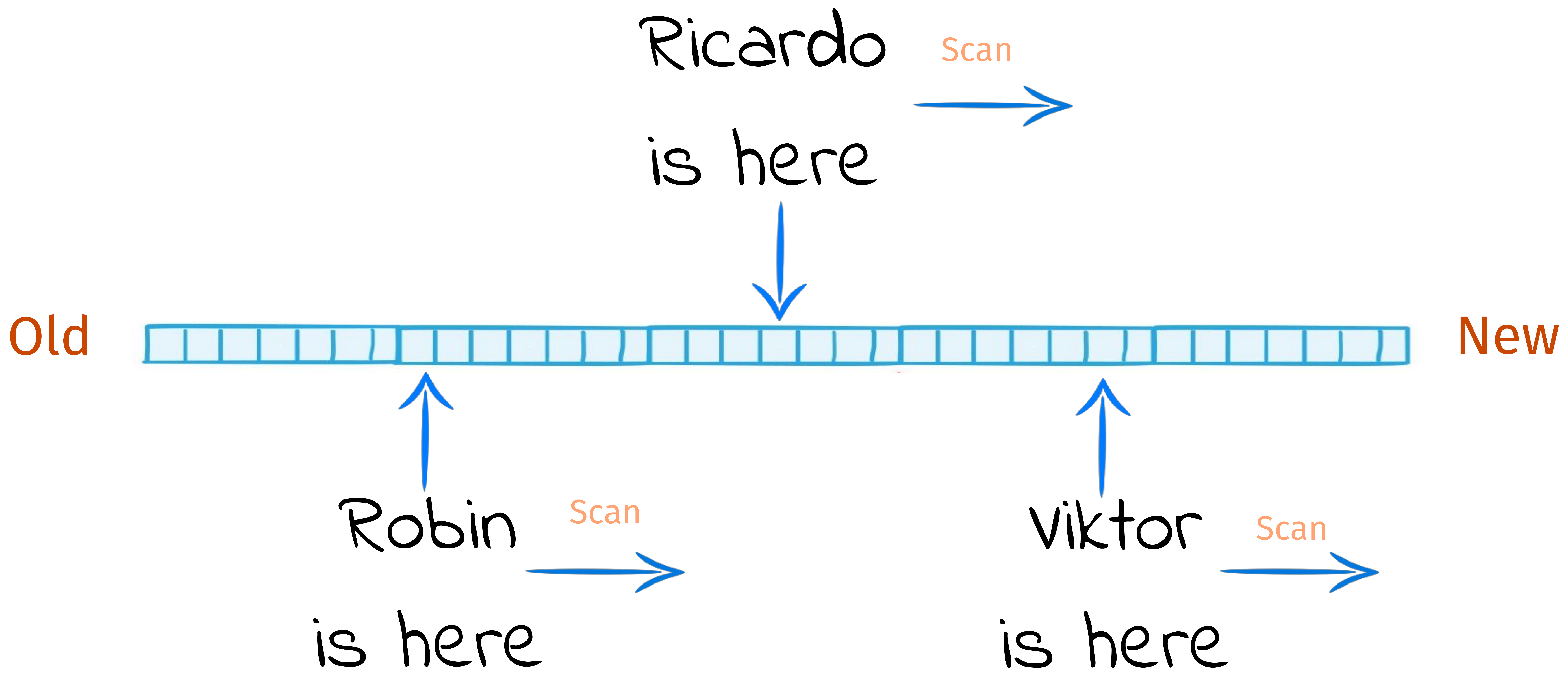


Messages are added at the end of the log

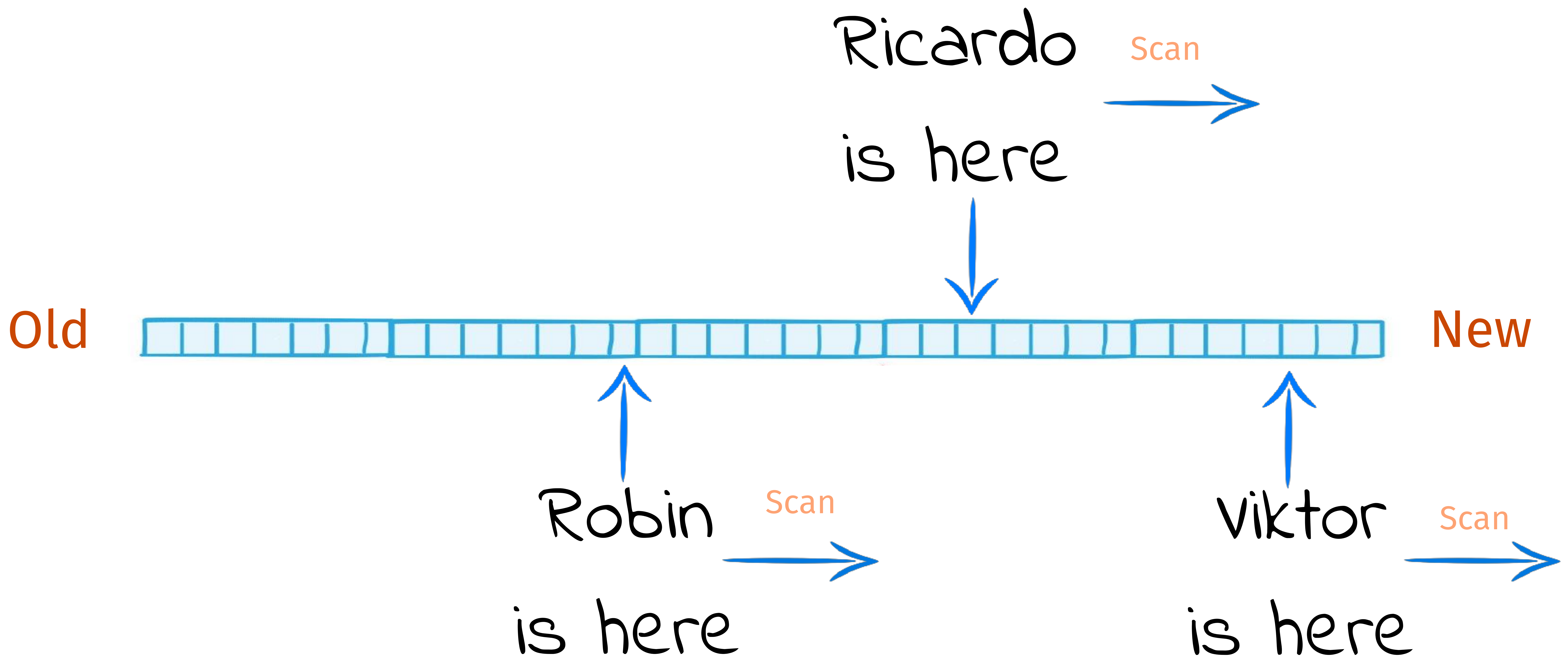
Consumers have a position all of their own



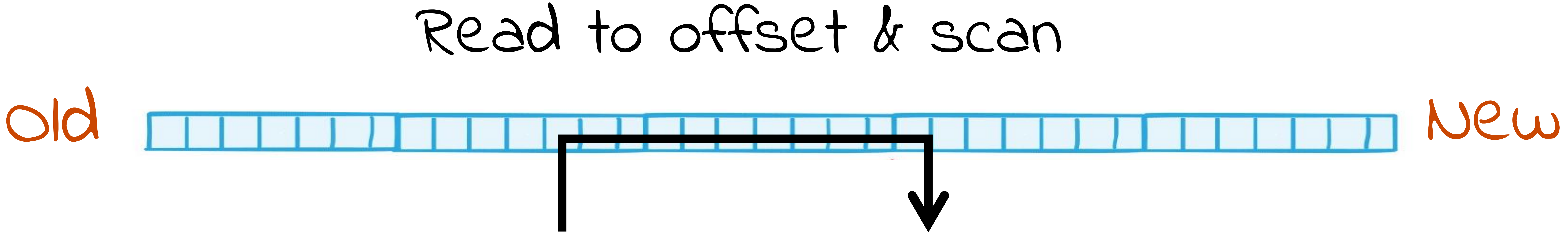
Consumers have a position all of their own



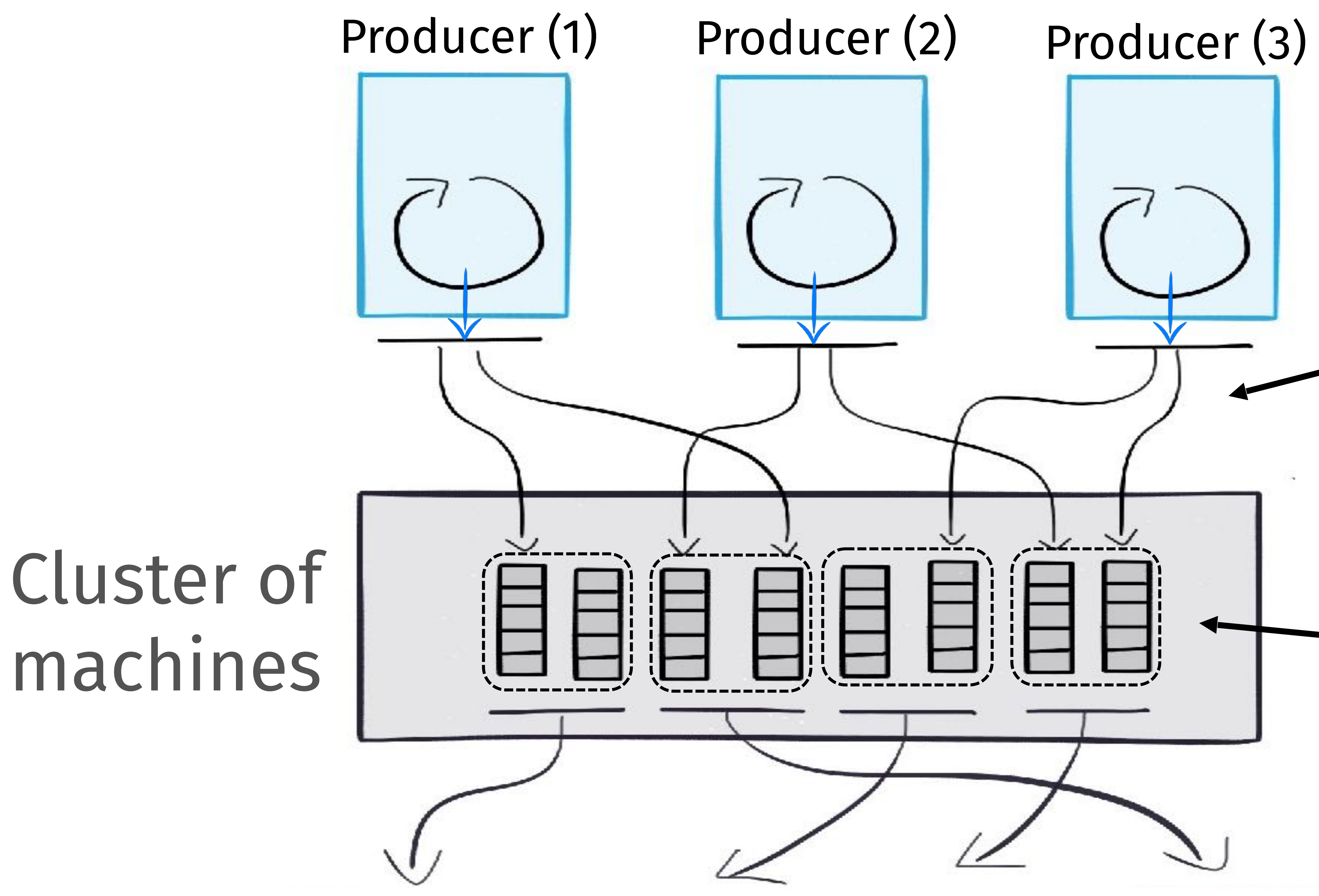
Consumers have a position all of their own



Only Sequential Access



Shard data to get scalability



Messages are sent to different partitions

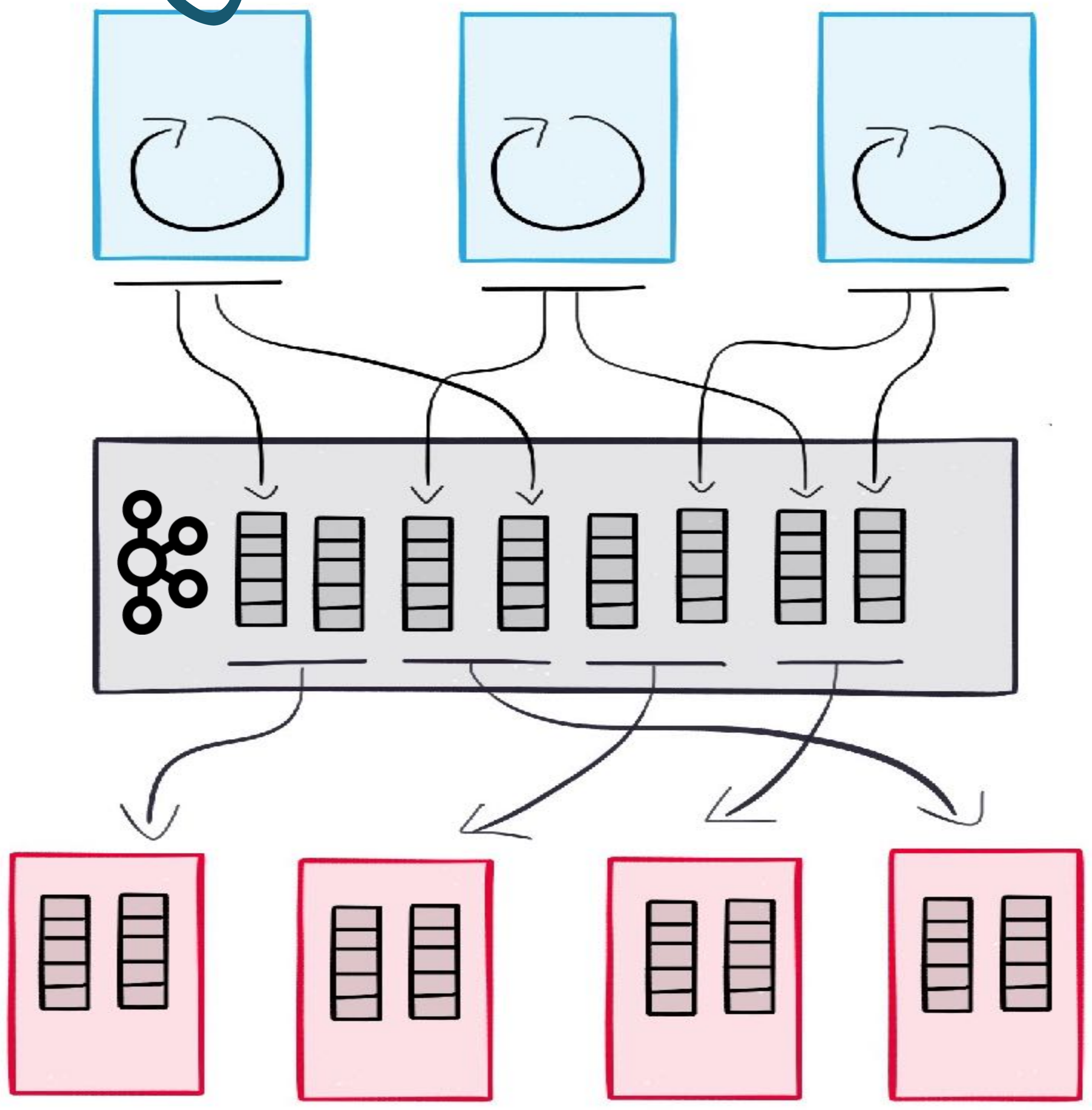
Partitions live on different machines

CONSUMER GROUP COORDINATOR
CONSUMERS
CONSUMER GROUP

Linearly Scalable Architecture

Producers

Consumers

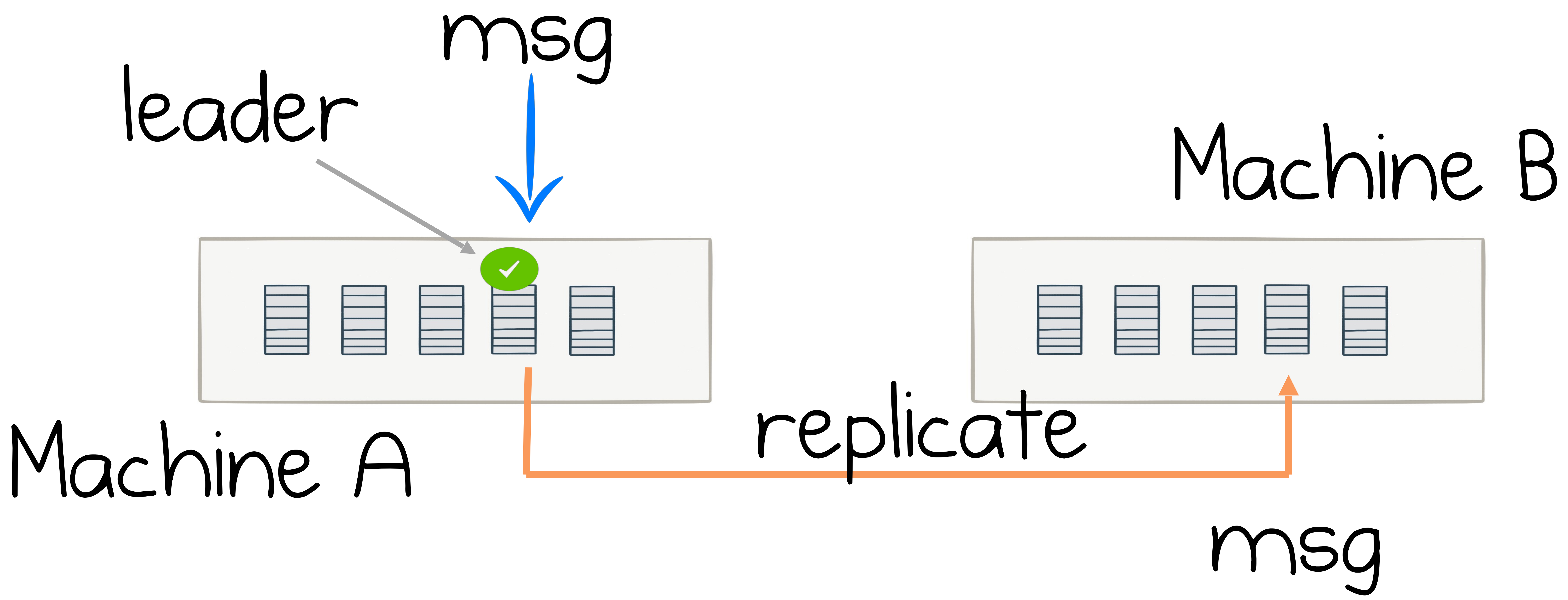


Single topic:

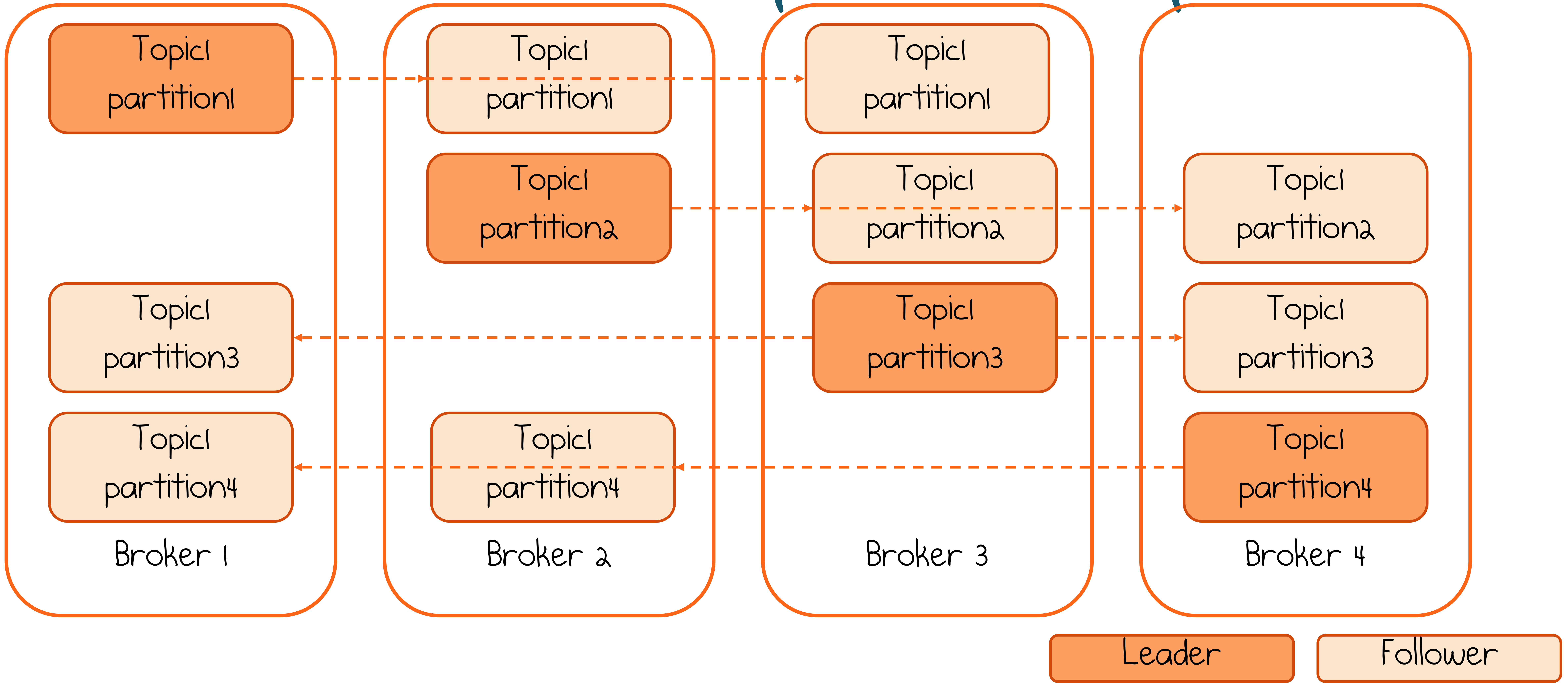
- Many producers machines
- Many consumer machines
- Many Broker machines

No Bottleneck!!

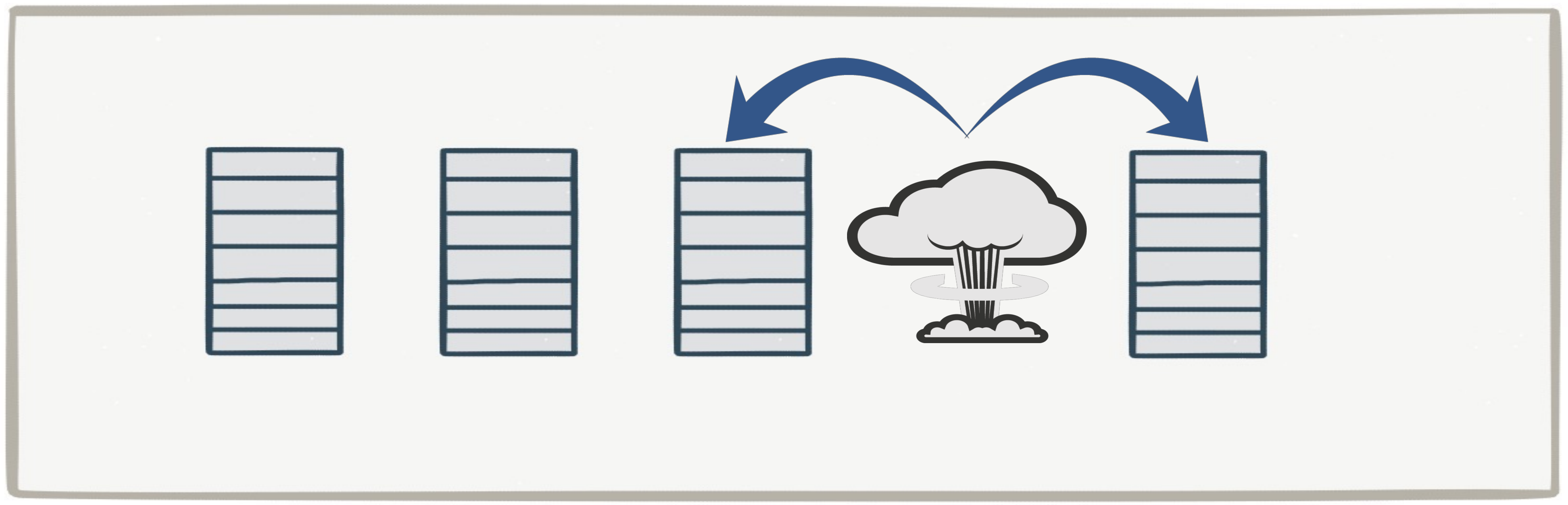
Replicate to get fault tolerance



Partition Leadership and Replication

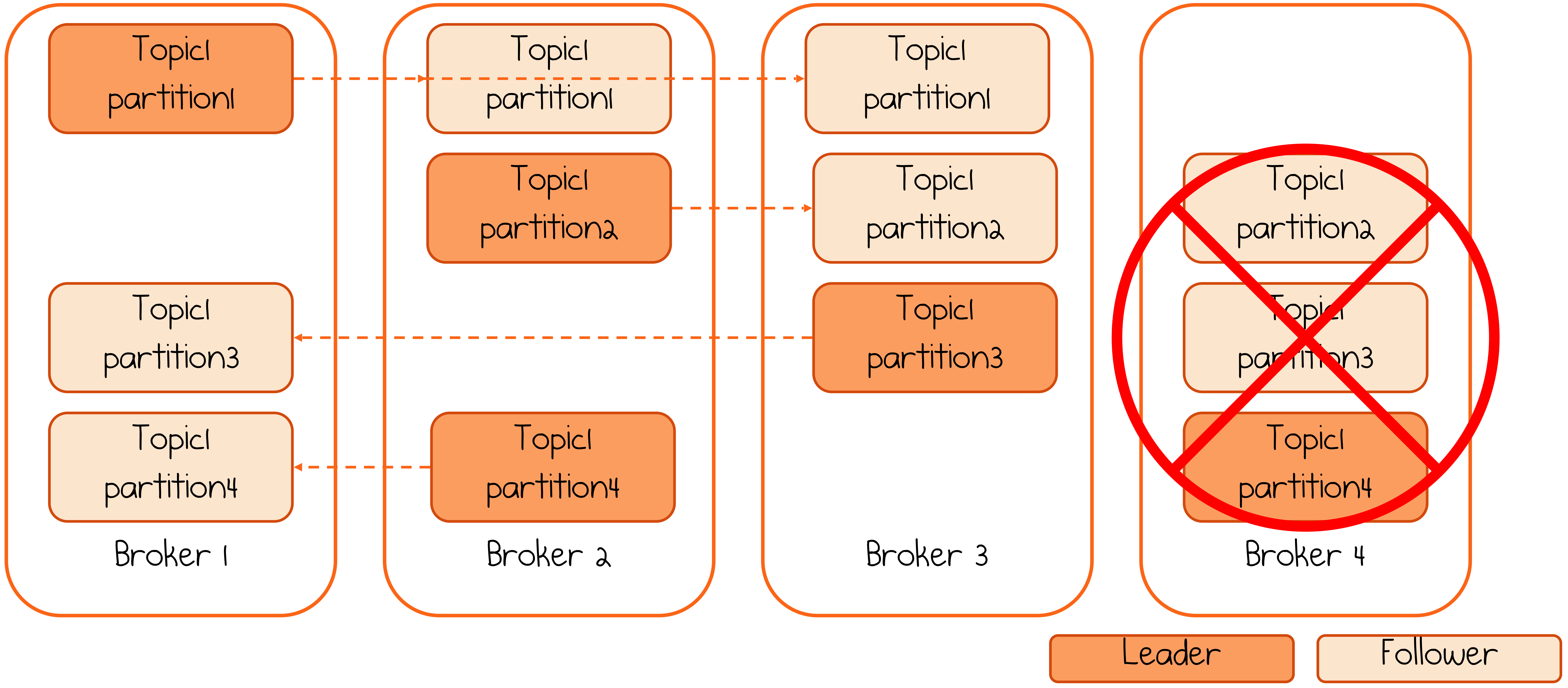


Replication provides resiliency



A 'replica' takes over on machine failure

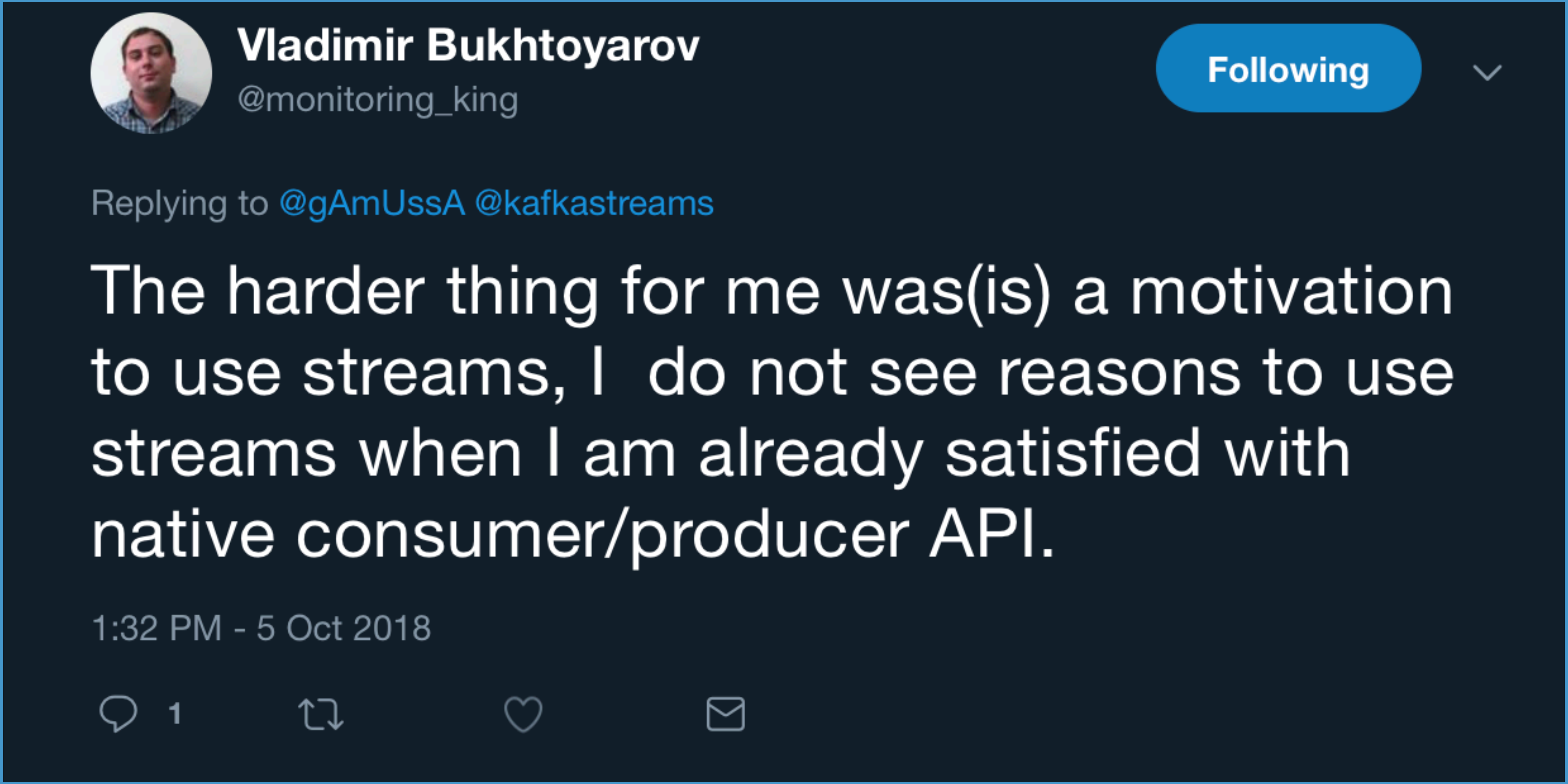
Partition Leadership and Replication - node failure



confluent @gamussa @riferrei @confluentinc

Lab 0: Confluent Cloud

confluent


Vladimir Bukhtoyarov
@monitoring_king

Following

Replying to @gAmUssA @kafkastreams

The harder thing for me was(is) a motivation to use streams, I do not see reasons to use streams when I am already satisfied with native consumer/producer API.

1:32 PM - 5 Oct 2018

1

https://twitter.com/monitoring_king/status/1048264580743479296

```java
// in-memory store, not persistent
Map<String, Integer> groupByCounts = new HashMap<>();

try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerProperties());
     KafkaProducer<String, Integer> producer = new KafkaProducer<>(producerProperties())) {

  consumer.subscribe(Arrays.asList("A", "B"));

  while (true) { // consumer poll loop
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(5));
    for (ConsumerRecord<String, String> record : records) {

      String key = record.key();
      Integer count = groupByCounts.get(key);

      if (count == null) {
        count = 0;
      }
      count += 1;

      groupByCounts.put(key, count);
    }
}
```

```java
// in-memory store, not persistent
Map<String, Integer> groupByCounts = new HashMap<>();

try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerProperties());
     KafkaProducer<String, Integer> producer = new KafkaProducer<>(producerProperties())) {

  consumer.subscribe(Arrays.asList("A", "B"));

  while (true) { // consumer poll loop
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(5));
    for (ConsumerRecord<String, String> record : records) {

      String key = record.key();
      Integer count = groupByCounts.get(key);

      if (count == null) {
        count = 0;
      }
      count += 1;

      groupByCounts.put(key, count);
    }
}
```

```
// in-memory store, not persistent
Map<String, Integer> groupByCounts = new HashMap<>();

try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerProperties());
     KafkaProducer<String, Integer> producer = new KafkaProducer<>(producerProperties())) {

  consumer.subscribe(Arrays.asList("A", "B"));

  while (true) { // consumer poll loop
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(5));
    for (ConsumerRecord<String, String> record : records) {

      String key = record.key();
      Integer count = groupByCounts.get(key);

      if (count == null) {
        count = 0;
      }
      count += 1;

      groupByCounts.put(key, count);
    }
}
```

```
// in-memory store, not persistent
Map<String, Integer> groupByCounts = new HashMap<>();

try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerProperties());
     KafkaProducer<String, Integer> producer = new KafkaProducer<>(producerProperties())) {

  consumer.subscribe(Arrays.asList("A", "B"));

  while (true) { // consumer poll loop
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(5));
    for (ConsumerRecord<String, String> record : records) {

      String key = record.key();
      Integer count = groupByCounts.get(key);

      if (count == null) {
        count = 0;
      }
      count += 1;

      groupByCounts.put(key, count);
    }
}
```

```
  while (true) { // consumer poll loop
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(5));
    for (ConsumerRecord<String, String> record : records) {

      String key = record.key();
      Integer count = groupByCounts.get(key);

      if (count == null) {
        count = 0;
      }
      count += 1;

      groupByCounts.put(key, count);
    }
}
```

```
  while (true) { // consumer poll loop
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(5));
    for (ConsumerRecord<String, String> record : records) {

      String key = record.key();
      Integer count = groupByCounts.get(key);

      if (count == null) {
        count = 0;
      }
      count += 1; // actually doing something useful

      groupByCounts.put(key, count);
    }
}
```

```java
    if (counter++ % sendInterval == 0) {
      for (Map.Entry<String, Integer> groupedEntry : groupByCounts.entrySet()) {

        ProducerRecord<String, Integer> producerRecord =
            new ProducerRecord<>("group-by-counts", groupedEntry.getKey(), groupedEntry.getValue());
        producer.send(producerRecord);
      }
      consumer.commitSync();
    }
  }
}
```

```java
    if (counter++ % sendInterval == 0) {
      for (Map.Entry<String, Integer> groupedEntry : groupByCounts.entrySet()) {

        ProducerRecord<String, Integer> producerRecord =
            new ProducerRecord<>("group-by-counts", groupedEntry.getKey(), groupedEntry.getValue());
        producer.send(producerRecord);
      }
      consumer.commitSync();
    }
  }
}
```

```
    if (counter++ % sendInterval == 0) {
      for (Map.Entry<String, Integer> groupedEntry : groupByCounts.entrySet()) {

        ProducerRecord<String, Integer> producerRecord =
            new ProducerRecord<>("group-by-counts", groupedEntry.getKey(), groupedEntry.getValue());
        producer.send(producerRecord);
      }
      consumer.commitSync();
    }
  }
}
```

As developers,
we want to build APPS
not INFRASTRUCTURE

LET'S TALK ABOUT THIS FRAMEWORK
OF YOURS.
I THINK ITS GOOD, EXCEPT IT SUCKS

SO LET ME WRITE THE FRAMEWORK
THAT'S WHY IT MIGHT BE REALLY GOOD

Every framework wants to be when it grows up

Scalable

Elastic

Fault-tolerant

Stateful

Distributed

```java
final StreamsBuilder streamsBuilder = new StreamsBuilder();
final KStream<String, Long> stream = streamsBuilder.stream(Arrays.asList("A", "B"));


stream.groupByKey()
    .count()
    .toStream()
    .to("group-by-counts",
        Produced.with(Serdes.String(), Serdes.Long()));

final Topology topology = streamsBuilder.build();
final KafkaStreams kafkaStreams = new KafkaStreams(topology, streamsProperties());
kafkaStreams.start();
```

```java
final StreamsBuilder streamsBuilder = new StreamsBuilder();
final KStream<String, Long> stream = streamsBuilder.stream(Arrays.asList("A", "B"));

// actual work
stream.groupByKey()
    .count()
    .toStream()
    .to("group-by-counts",
        Produced.with(Serdes.String(), Serdes.Long()));

final Topology topology = streamsBuilder.build();
final KafkaStreams kafkaStreams = new KafkaStreams(topology, streamsProperties());
kafkaStreams.start();
```

```
final StreamsBuilder streamsBuilder = new StreamsBuilder();
final KStream<String, Long> stream = streamsBuilder.stream(Arrays.asList("A", "B"));

// actual work
stream.groupByKey()
    .count()
    .toStream()
    .to("group-by-counts",
        Produced.with(Serdes.String(), Serdes.Long()));

final Topology topology = streamsBuilder.build();
final KafkaStreams kafkaStreams = new KafkaStreams(topology, streamsProperties());
kafkaStreams.start();
```

```
final StreamsBuilder streamsBuilder = new StreamsBuilder();
final KStream<String, Long> stream = streamsBuilder.stream(Arrays.asList("A", "B"));

// actual work
stream.groupByKey()
    .count()
    .toStream()
    .to("group-by-counts",
        Produced.with(Serdes.String(), Serdes.Long()));

final Topology topology = streamsBuilder.build();
final KafkaStreams kafkaStreams = new KafkaStreams(topology, streamsProperties());
kafkaStreams.start();
```

Where do I put my compute?

Where do I put my state?

The actual question is

Where is my code?

HIVE
STORM
APACHE Spark
hadoop Map Reduce
kafka
ZOO

the KAFKA STREAMS **API** is a

JAVA API to

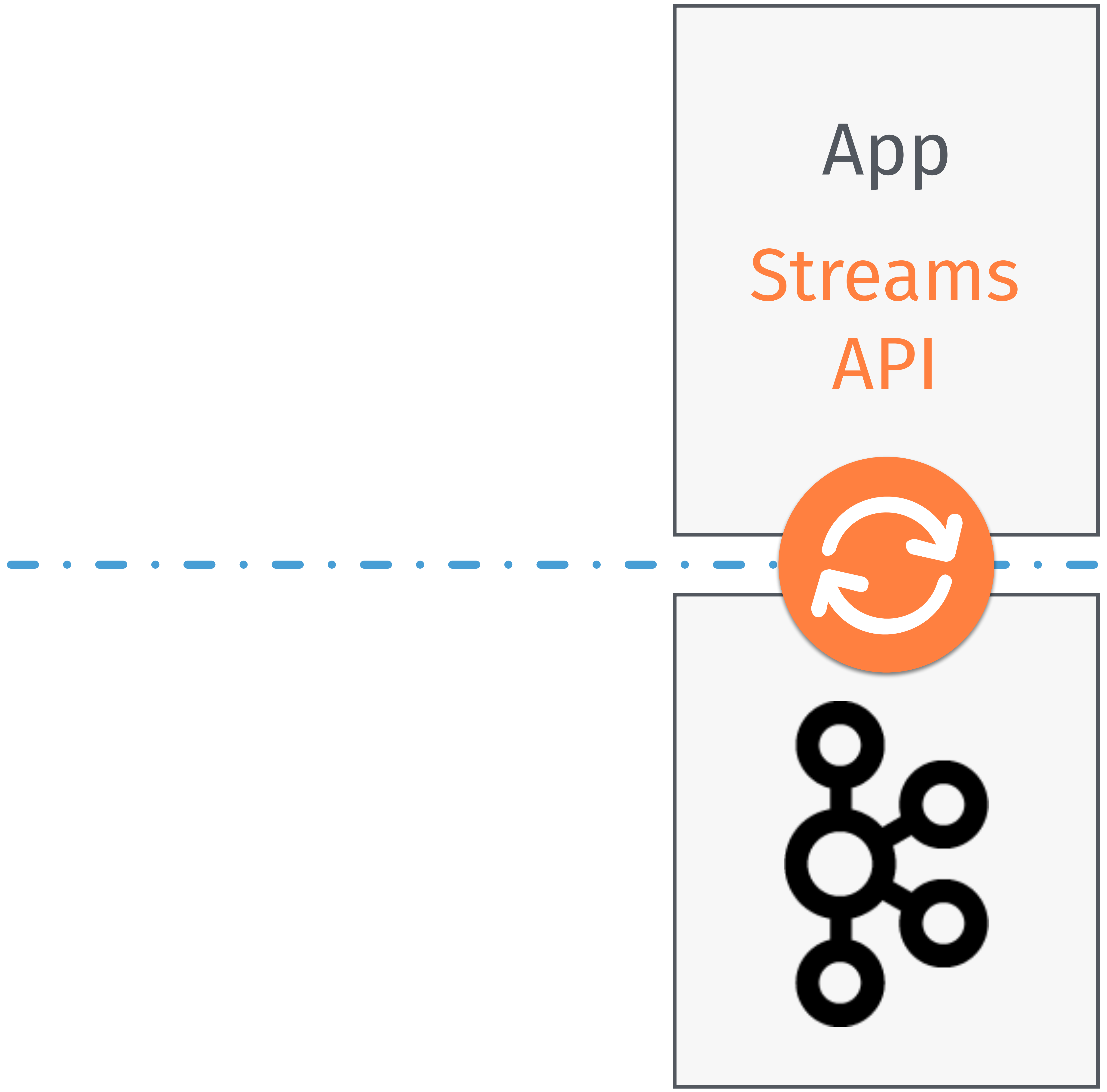
BUILD REAL-TIME **APPLICATIONS**

confluent

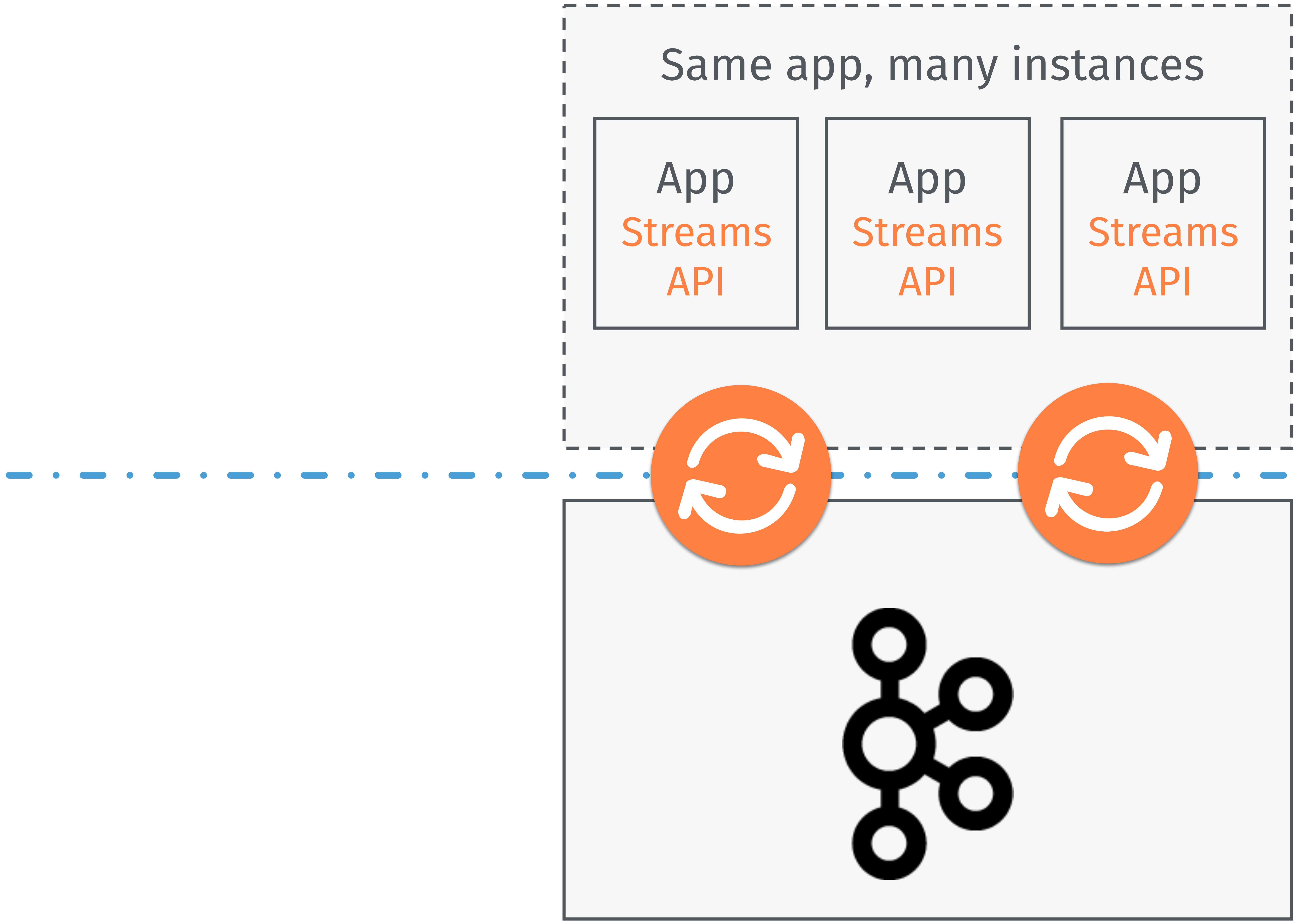
@gamussa

@riferrei

@confluentinc



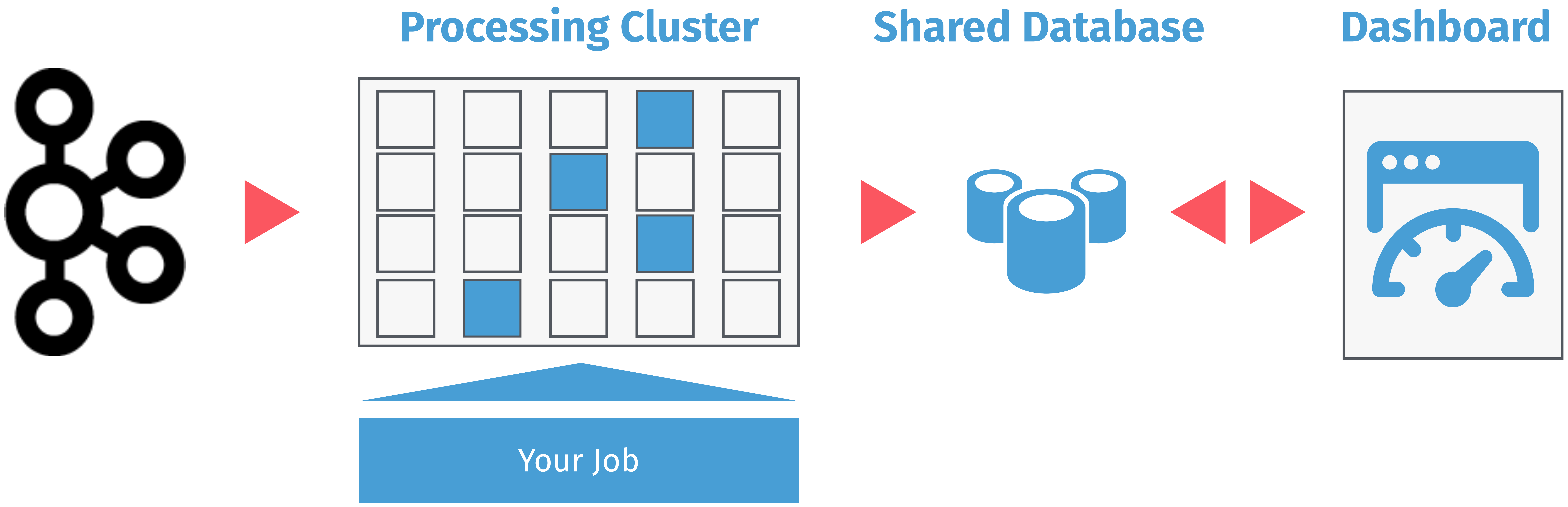
Not running inside brokers!



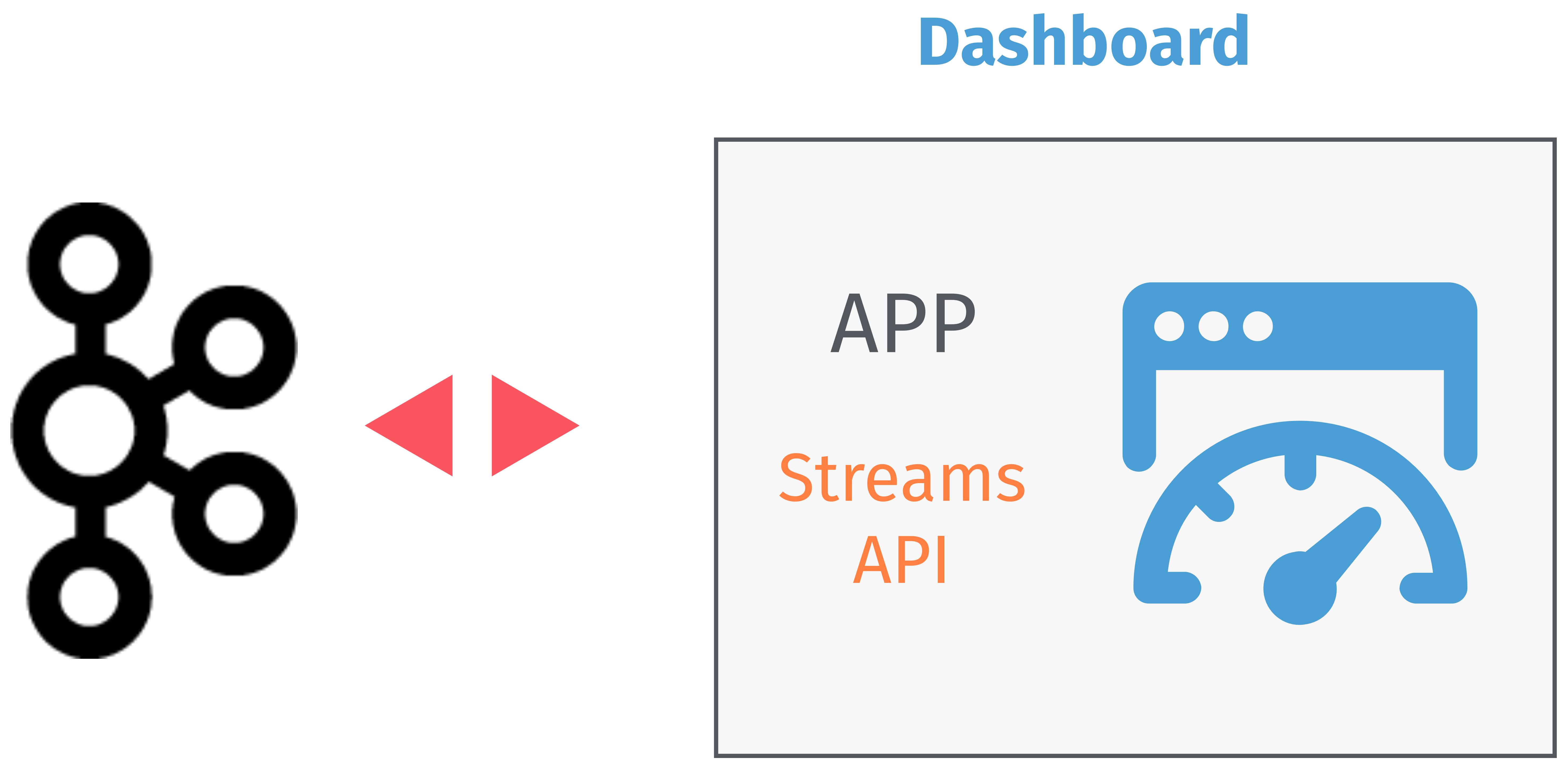
Brokers?

Nope!

Before



After



this means you can

DEPLOY your app **ANYWHERE** using

WHATEVER TECHNOLOGY YOU WANT

So many places to run you app!

amazon web services

Google Cloud Platform

Microsoft Azure

docker

kubernetes

Physical

openstack.

MESOS

vmware®

VAGRANT

TERRAFORM

ANSIBLE

puppet labs

Jenkins

...and many more...

Things Kafka Stream Does

Enterprise Support

Open Source

Runs Everywhere

Elastic, Scalable, Fault-tolerant

Kafka Security Integration

Powerful Processing incl. Filters, Transforms, Joins, Aggregations, Windowing

Supports Streams and Tables

Exactly-Once Processing

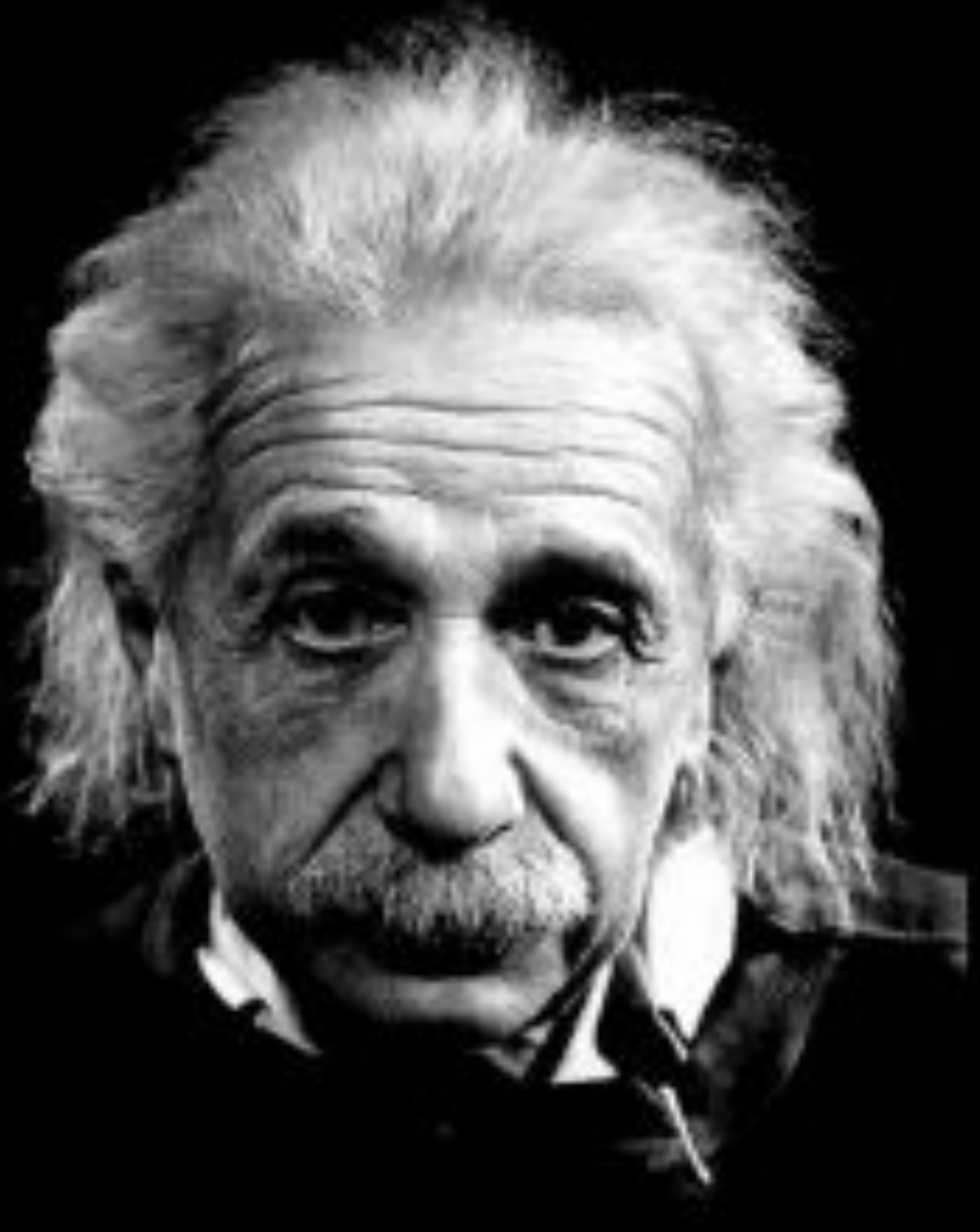
Event-Time Processing

Talk is cheap!

Show me code!

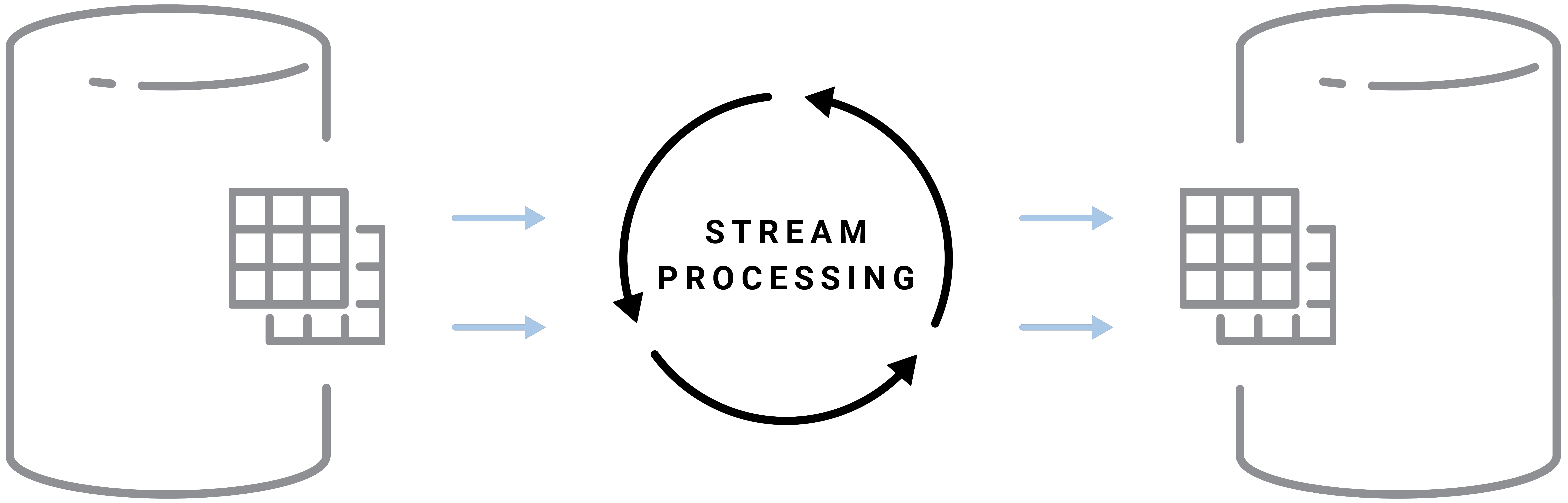
Lab 1: Streams Movie with Kafka Streams

SPECIAL RELATIVITY



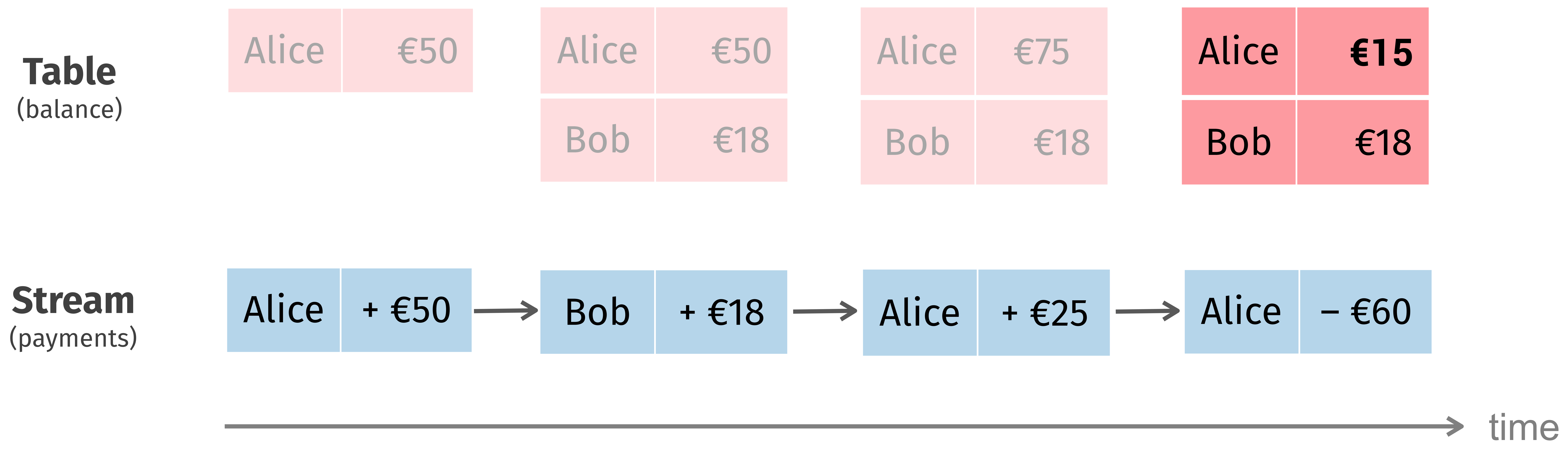
OF STREAMS AND TABLES

Table-Stream Duality

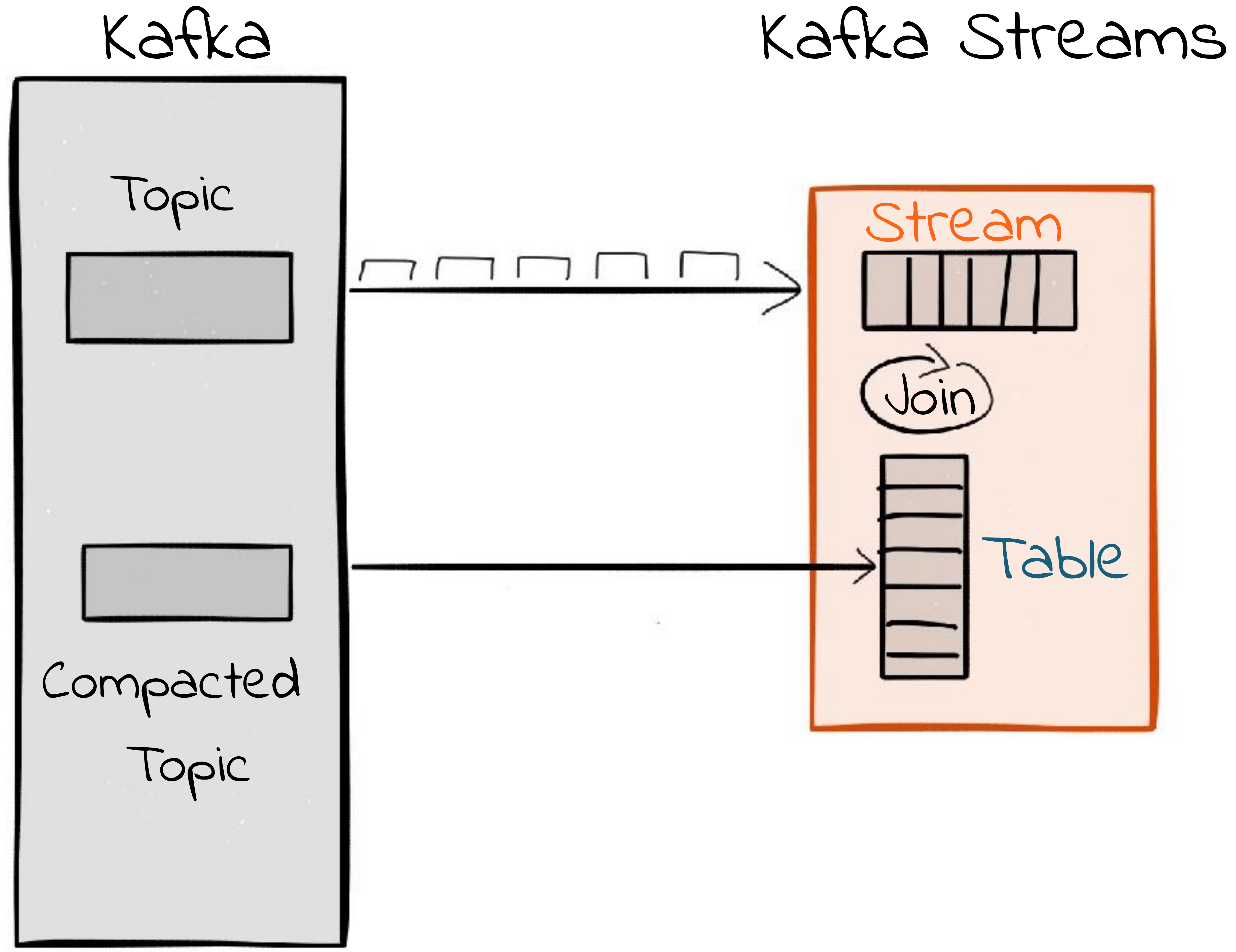


Do you think that’s a **table** you are querying ?

The Stream-Table Duality



Join Streams and Tables

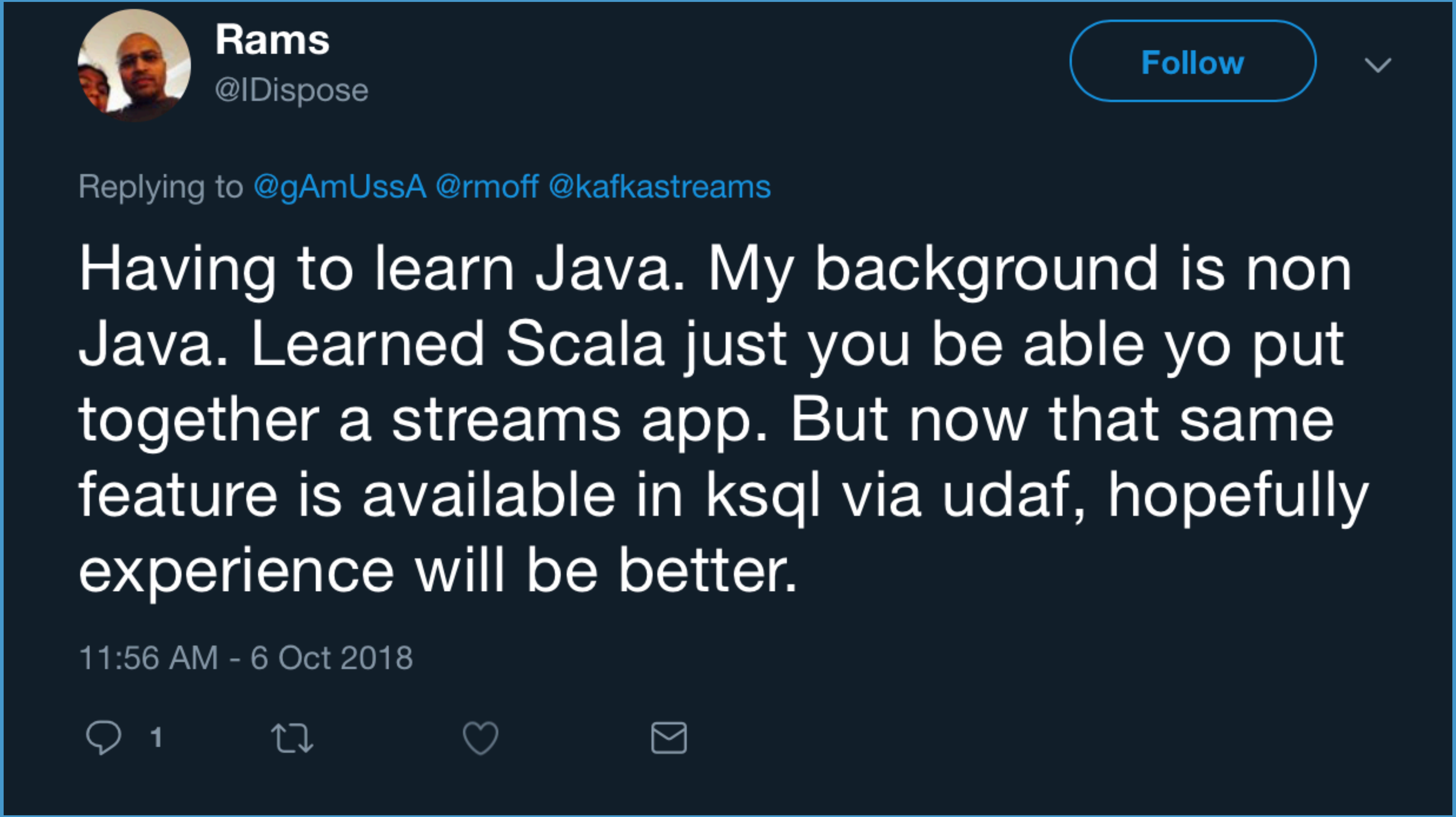


Talk is cheap!

Show me code!

What’s next?

confluent



Rams
@IDispose

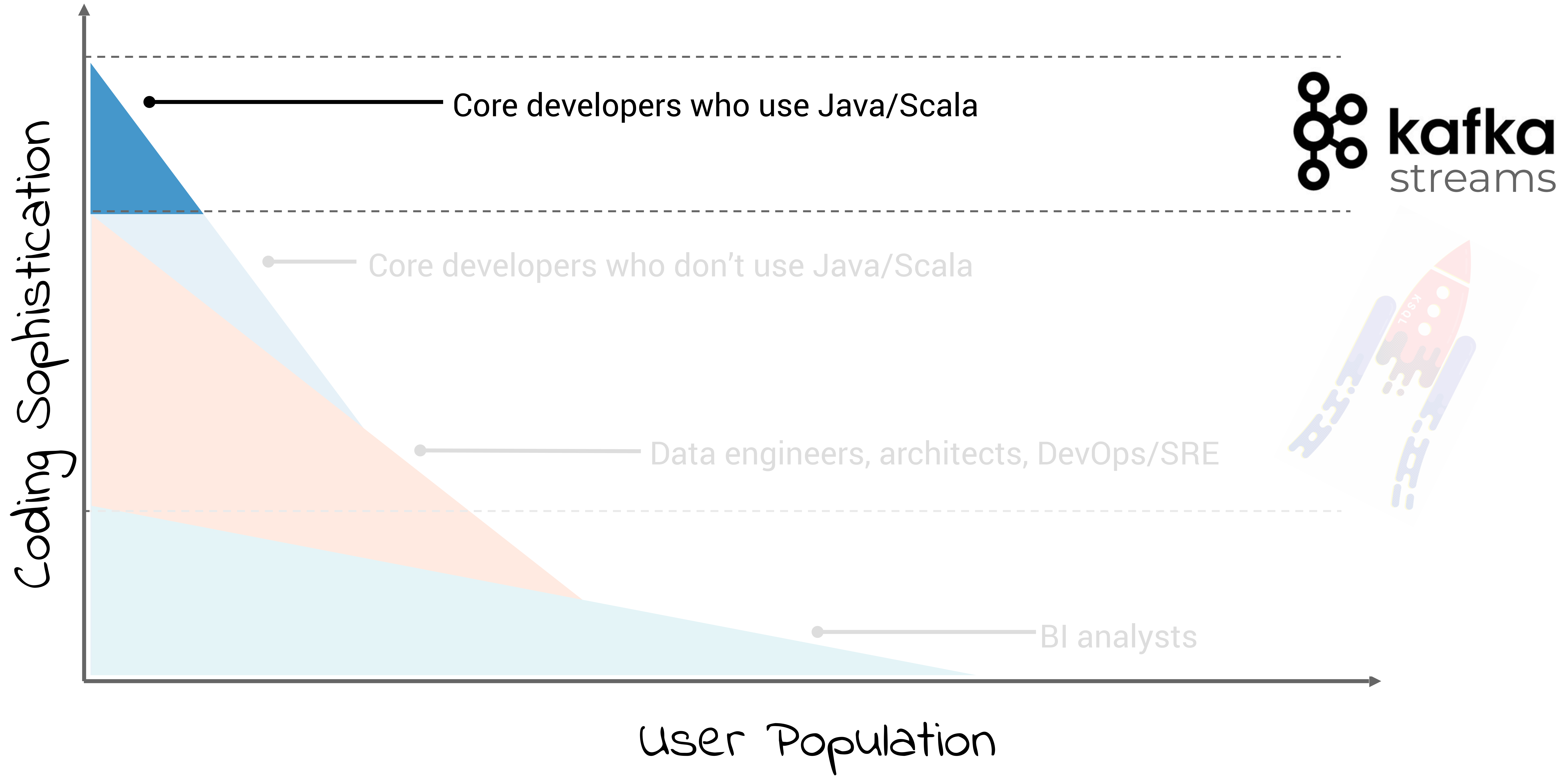
Replying to @gAmUssA @rmoff @kafkastreams

Having to learn Java. My background is non Java. Learned Scala just you be able yo put together a streams app. But now that same feature is available in ksql via udaf, hopefully experience will be better.

11:56 AM - 6 Oct 2018

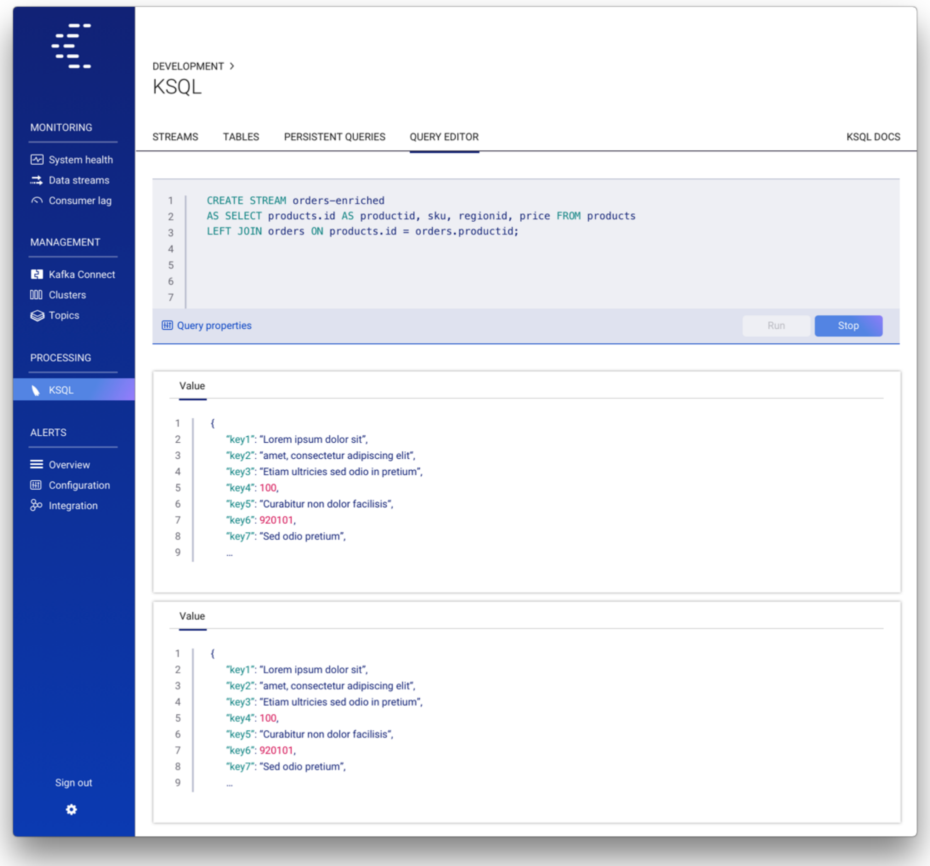
https://twitter.com/IDispose/status/1048602857191170054

Lower the bar to enter the world of streaming



confluent @gamussa @riferrei @confluentinc

KSQL #FTW



```
CREATE STREAM orders-enriched
AS SELECT products.id AS productid, sku, regionid, price FROM products
LEFT JOIN orders ON products.id = orders.productid;
```

```
{
  "key1": "Lorem ipsum dolor sit",
  "key2": "amet, consectetur adipiscing elit",
  "key3": "Etiam ultricies sed odio in pretium",
  "key4": 100,
  "key5": "Curabitur non dolor facilisis",
  "key6": 920101,
  "key7": "Sed odio pretium",
  ...
```

```
{
  "key1": "Lorem ipsum dolor sit",
  "key2": "amet, consectetur adipiscing elit",
  "key3": "Etiam ultricies sed odio in pretium",
  "key4": 100,
  "key5": "Curabitur non dolor facilisis",
  "key6": 920101,
  "key7": "Sed odio pretium",
  ...
```

```
ksql>
```

```
POST /query
```

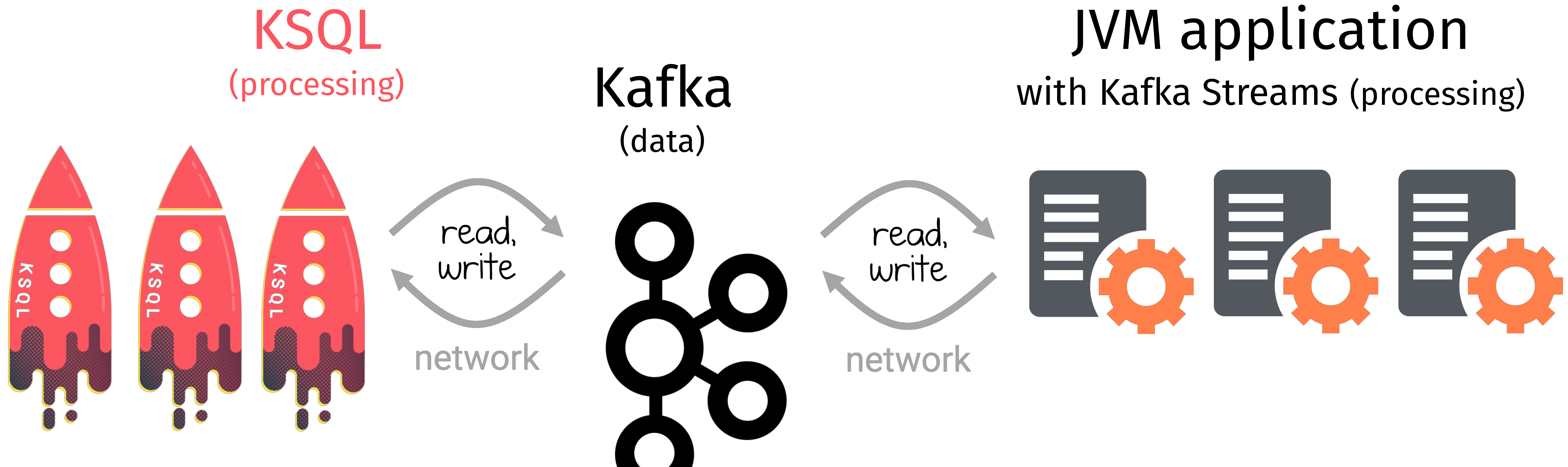
1 UI

2 CLI

3 REST

4 Headless

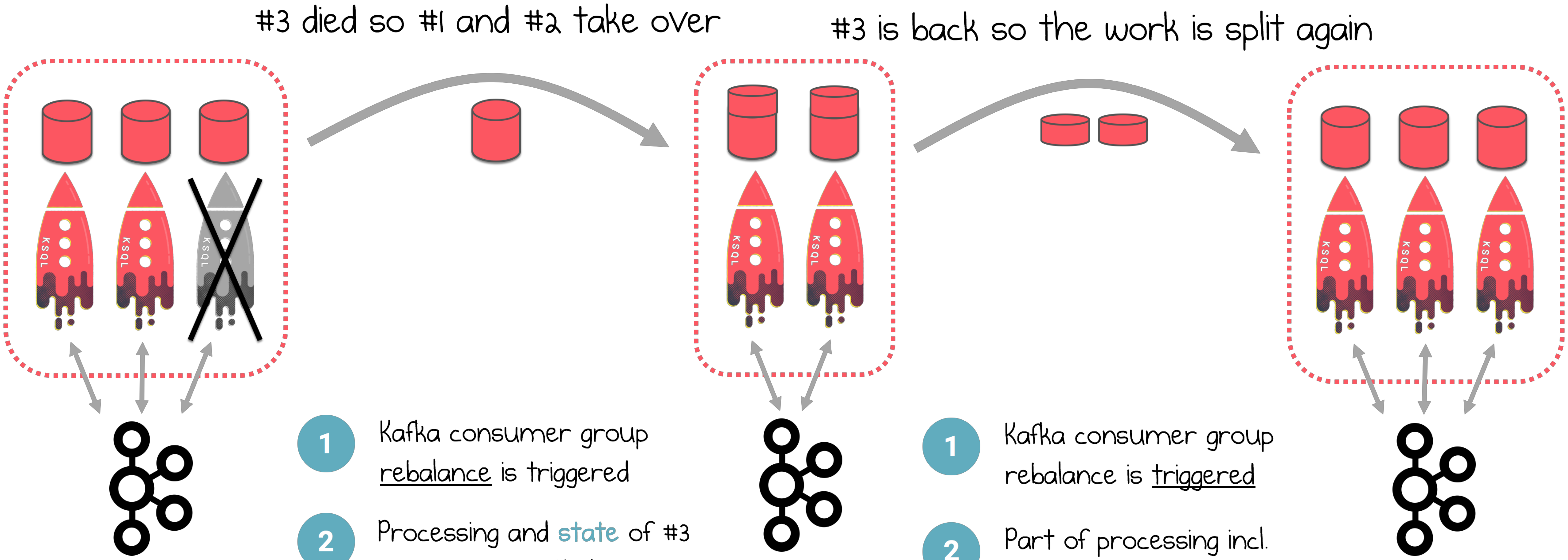
Interaction with Kafka



Does not run on Kafka brokers

Does not run on Kafka brokers

Fault-Tolerance, powered by Kafka


#3 died so #1 and #2 take over
#3 is back so the work is split again
KSQL

1 Kafka consumer group rebalance is triggered

2 Processing and state of #3 is migrated via Kafka to remaining servers #1 + #2

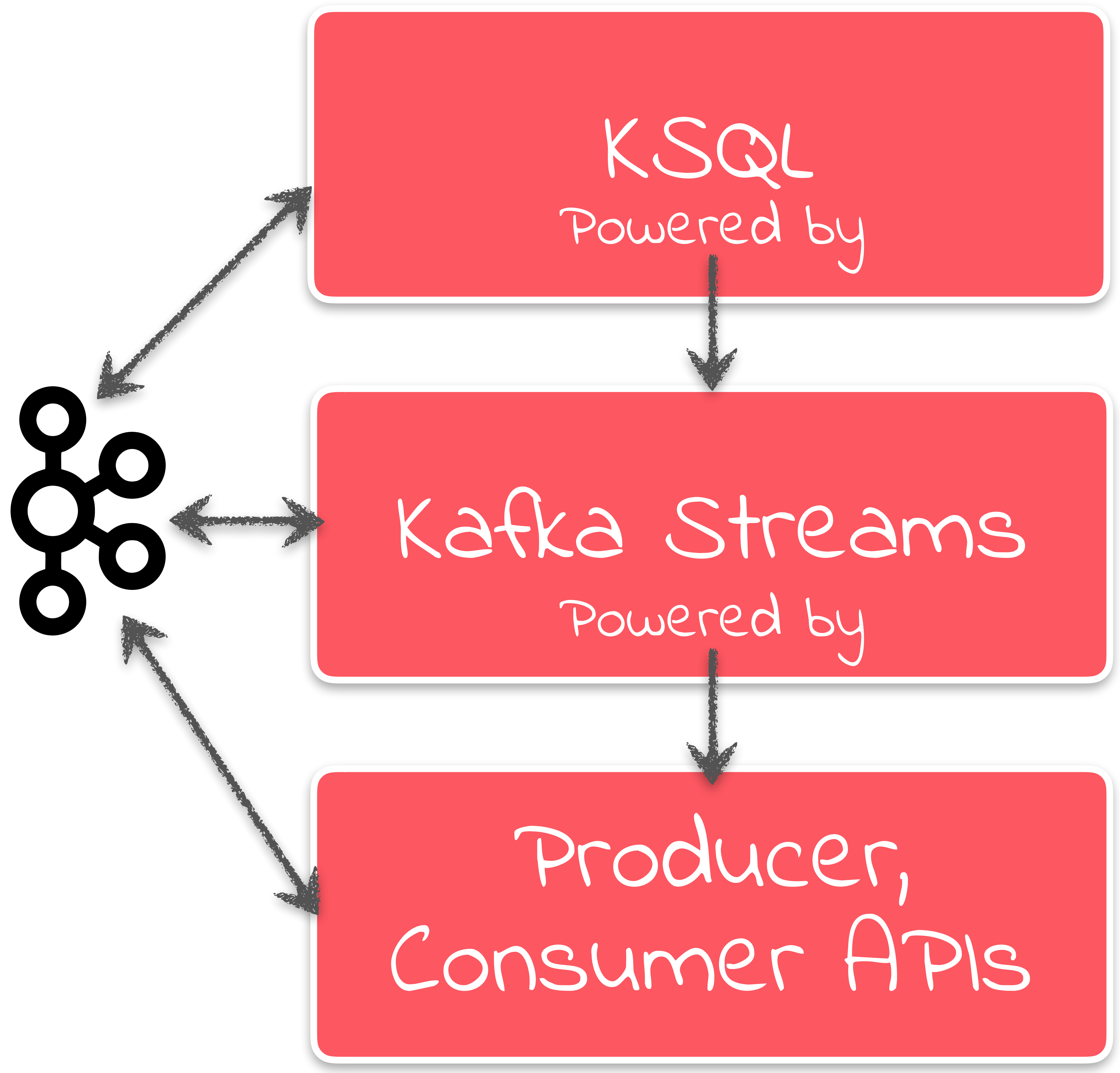
1 Kafka consumer group rebalance is triggered

2 Part of processing incl. state is migrated via Kafka from #1 + #2 to server #3

Differences

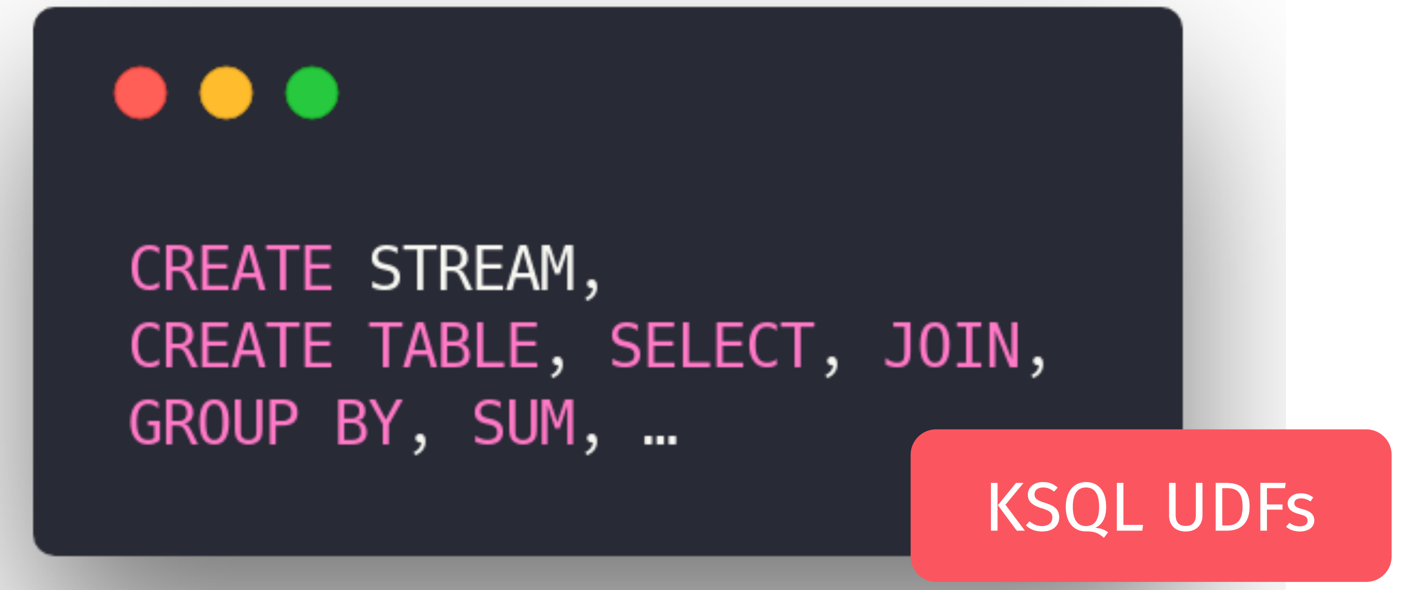
	KSQL	kafka streams
You write...	**KSQL statements**	**JVM applications**
UI included for human interaction	**Yes**, in Confluent Platform	No
CLI included for human interaction	**Yes**	No
Data formats	Avro, JSON, CSV (today)	**Any data format**, including Avro, JSON, CSV, Protobuf, XML
REST API included	**Yes**	No, but you can DIY
Runtime included	**Yes**, the KSQL server	**Not needed**, applications run as standard JVM processes
Queryable state	Not yet	**Yes**

Standing on the shoulders of Streaming Giants



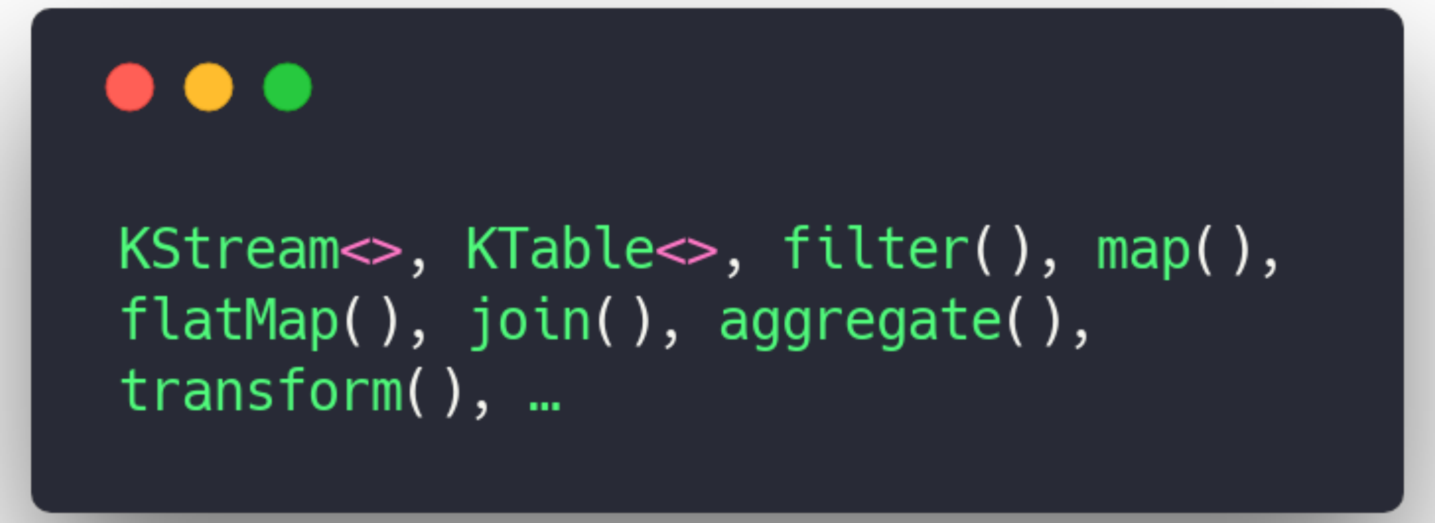
Ease of use

Flexibility

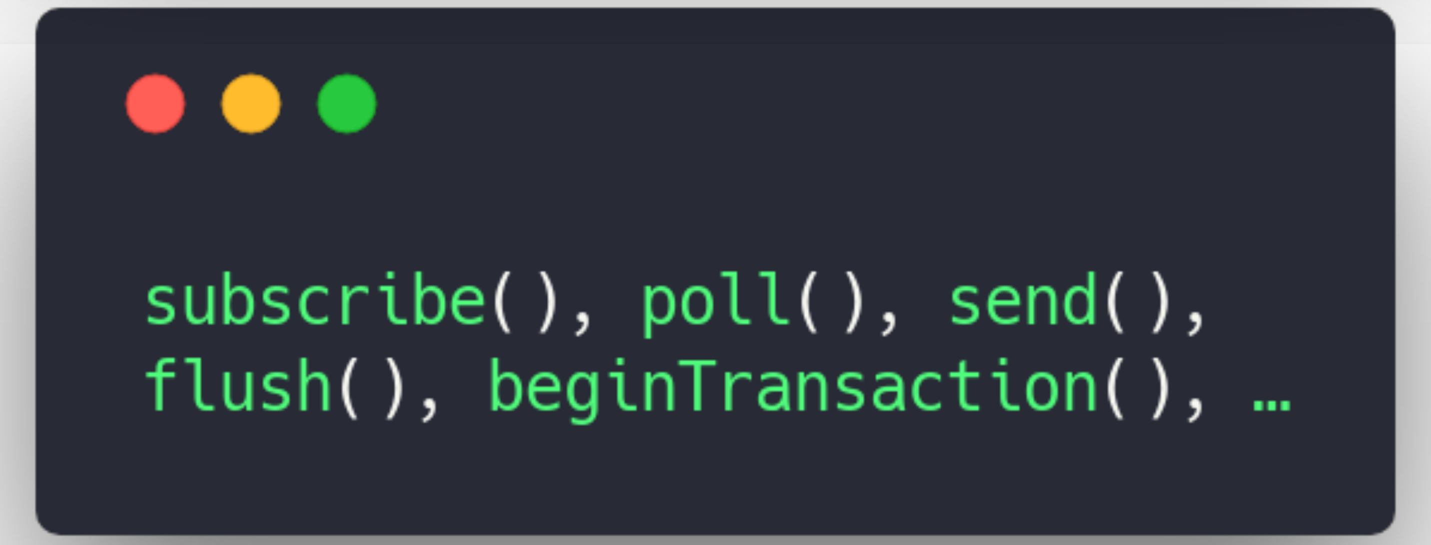


```
CREATE STREAM,
CREATE TABLE, SELECT, JOIN,
GROUP BY, SUM, …
```

KSQL UDFs



```
KStream<>, KTable<>, filter(), map(),
flatMap(), join(), aggregate(),
transform(), …
```



```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

One last thing...

kafka summit

https://kafka-summit.org

Gamov30

APRIL 2, 2019

NEW YORK CITY

THANKS!

@gamussa viktor@confluent.io

@riferrei ricardo@confluent.io

We are hiring!
https://www.confluent.io/careers/