ЗАЧЕМ НАМ БД? НАМ БАЗА НЕ НУЖНА...

MAY, 2019 / MOSCOW, RUSSIA

@GAMUSSA | #JUGMSK | @CONFLUENTINC

@GAMUSSA | #JUGMSK | @CONFLUENTINC

confluent

FRIENDLY REMINDER

- Follow @gamussa @confluentinc
- 📸🖼️🏋️
- Tag @gamussa
- With #jugmsk

SPECIAL THANKS!

@MATTHIASJSAX

@BBEJECK

@GAMUSSA | #JUGMSK | @CONFLUENTINC

confluent

HTTPS://CNFL.IO/STREAMS-MOVIE-DEMO

@GAMUSSA | #JUGMSK | @CONFLUENTINC

AGENDA

Kafka Streams 101

What is state and why stateful stream processing?

Technical Deep Dive on Interactive Queries

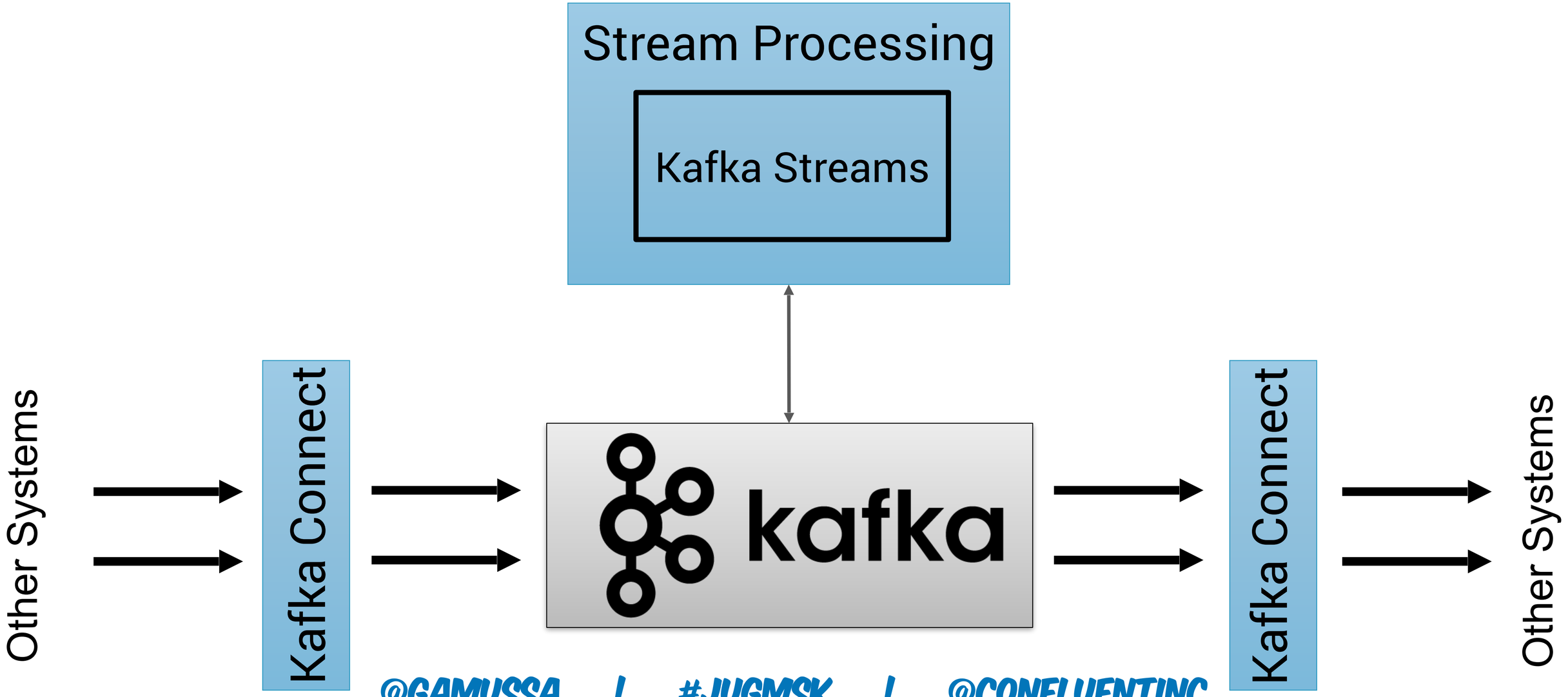
- What does IQ provide?
- State handling with Kafka Streams
- How to use IQ?

confluent

KAFKA STREAMS 101

@GAMUSSA | #JUGMSK | @CONFLUENTINC

KAFKA STREAMS - 101



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

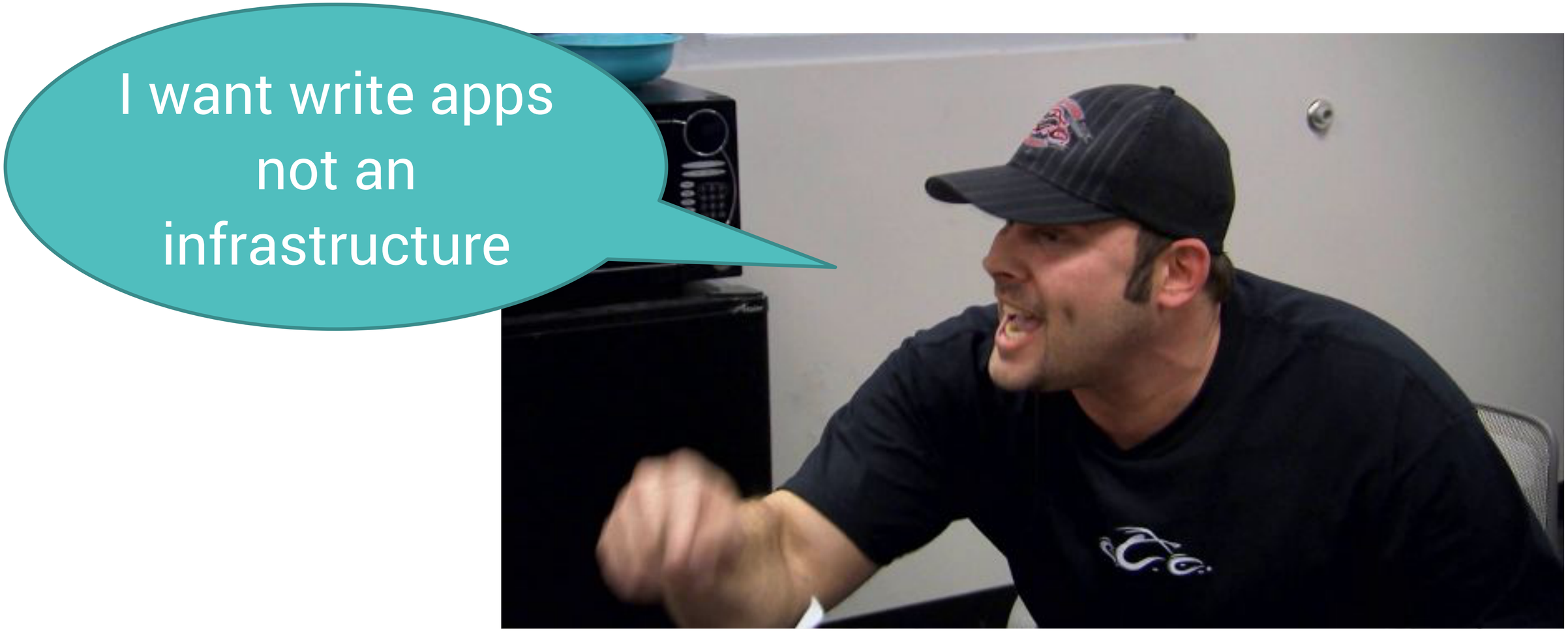
1

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

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

SO LET ME SHOW KAFKA STREAMS
THAT WAY IT MIGHT BE REALLY GOOD

We need a processing cluster like Spark or Flink

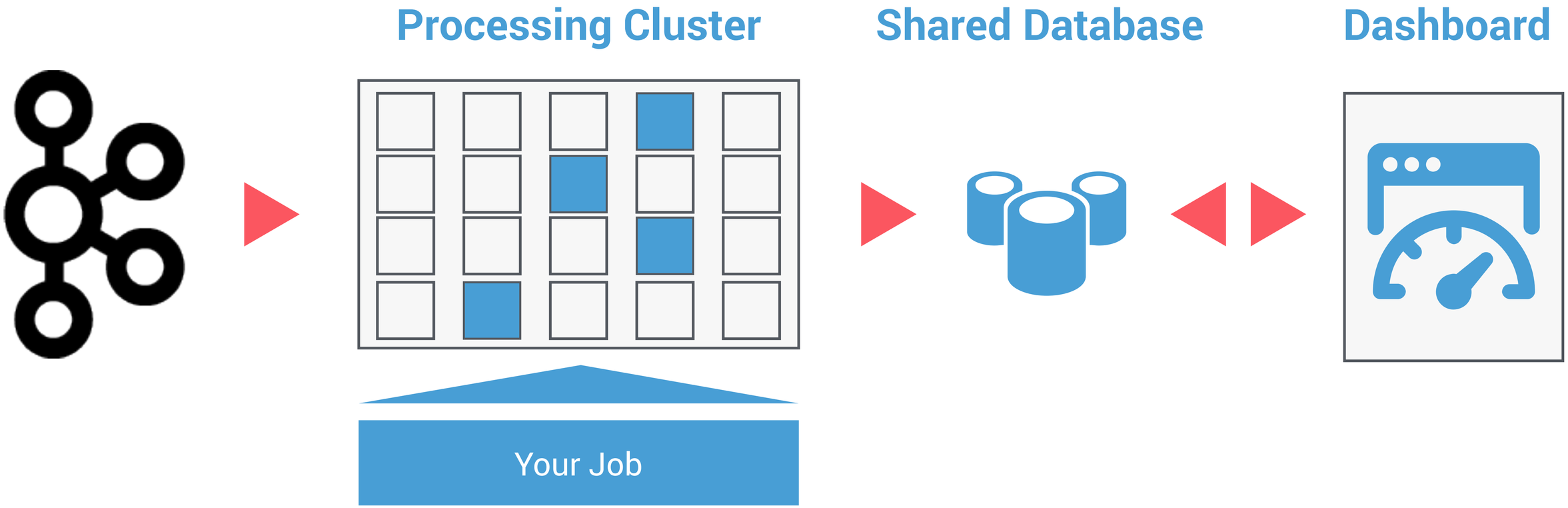

I want write apps
not an
infrastructure

You can’t do distributed stream processing without cluster

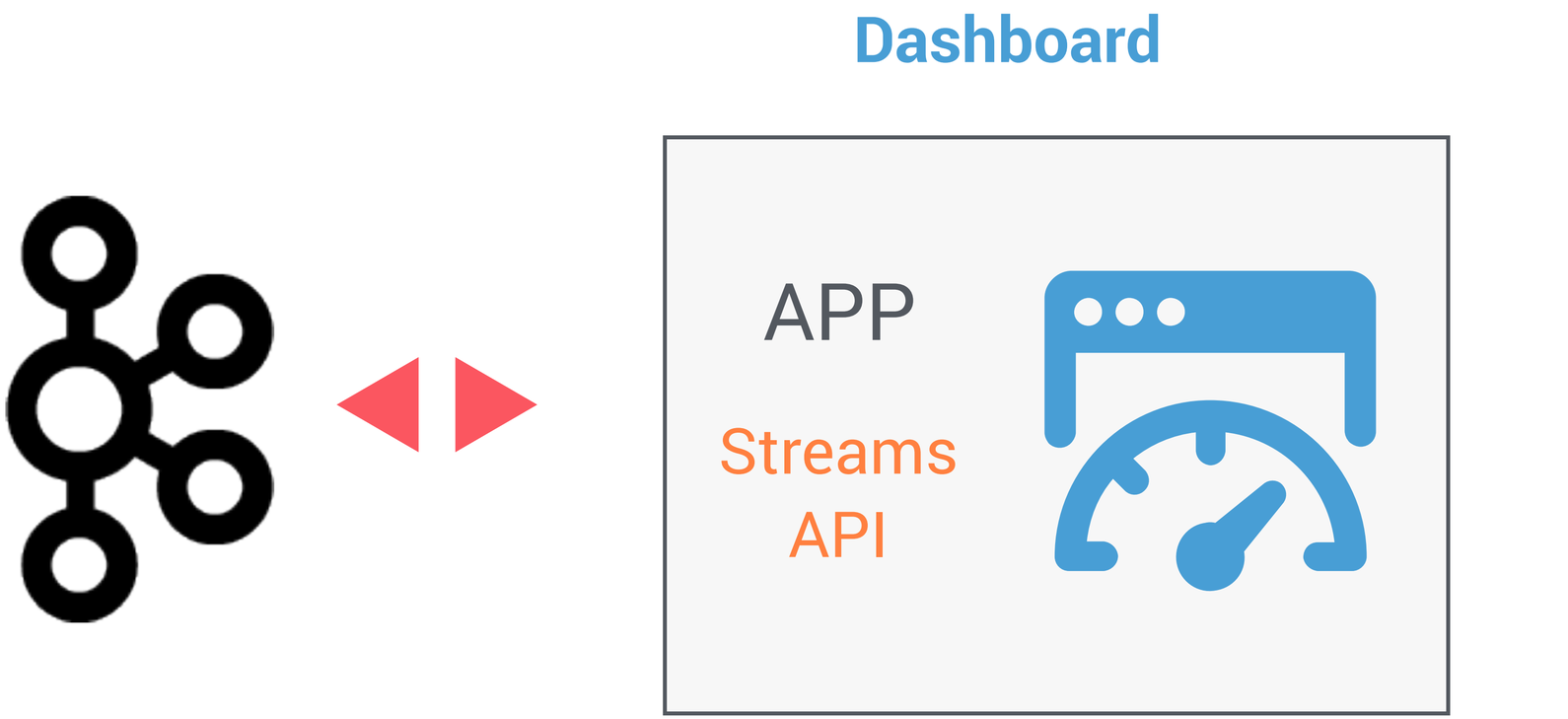
@GAMUSSA | #JUGMSK | @CONFLUENTINC

Stay and write your Java apps, I'm going to write ETL jobs

BEFORE



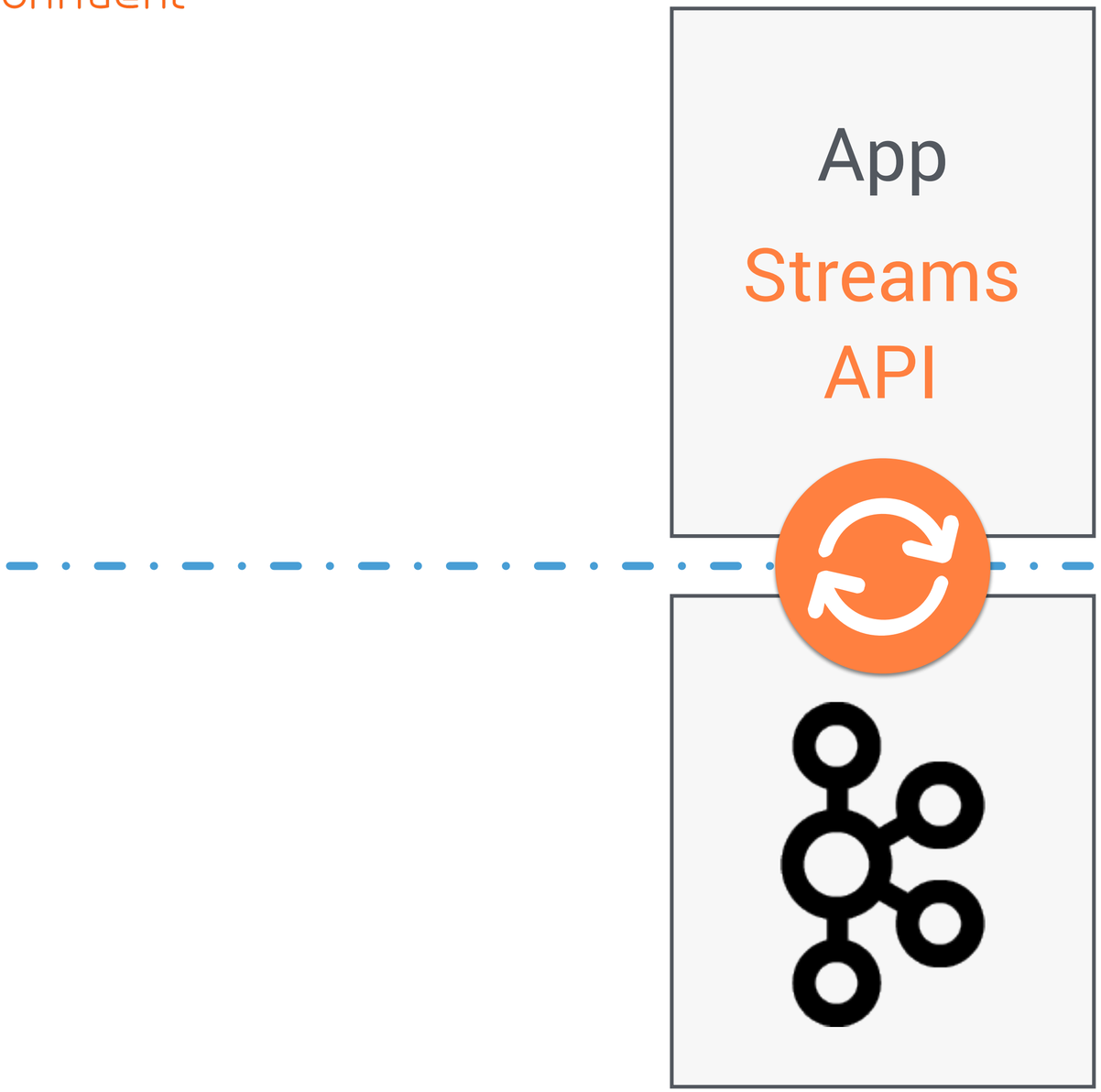
AFTER



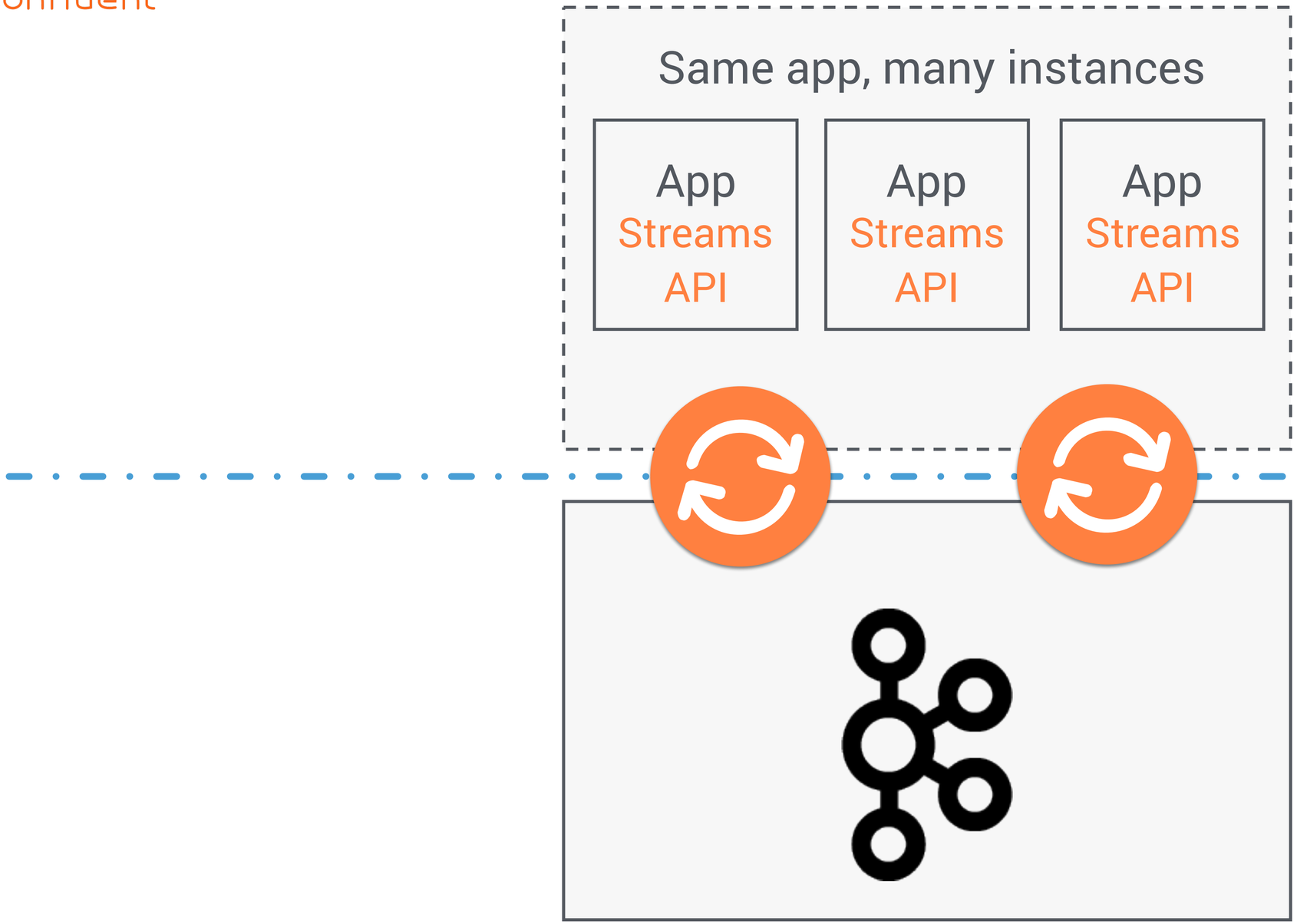
the KAFKA STREAMS **API** is a

JAVA API to

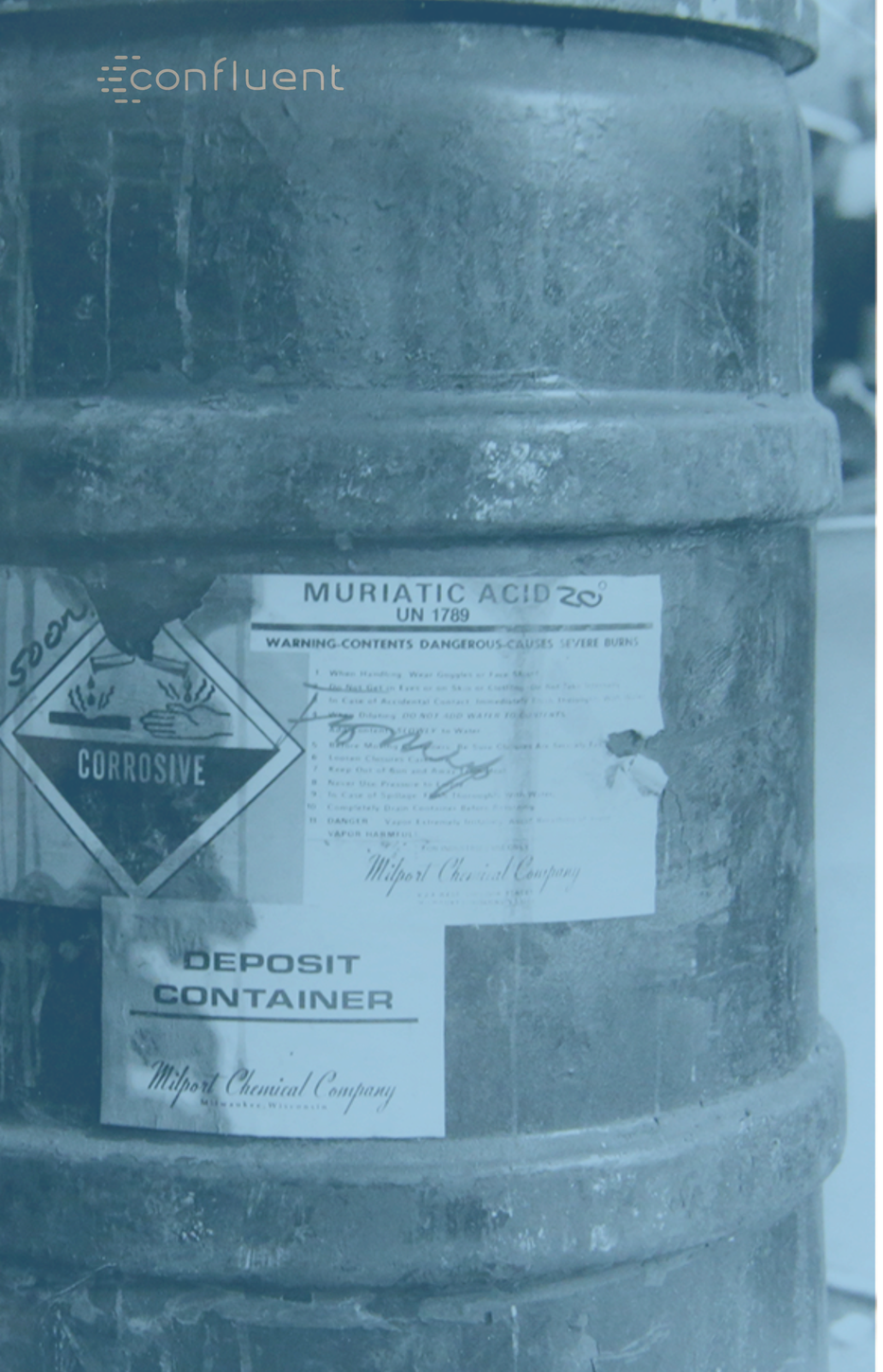
BUILD REAL-TIME **APPLICATIONS**



Not running inside brokers!



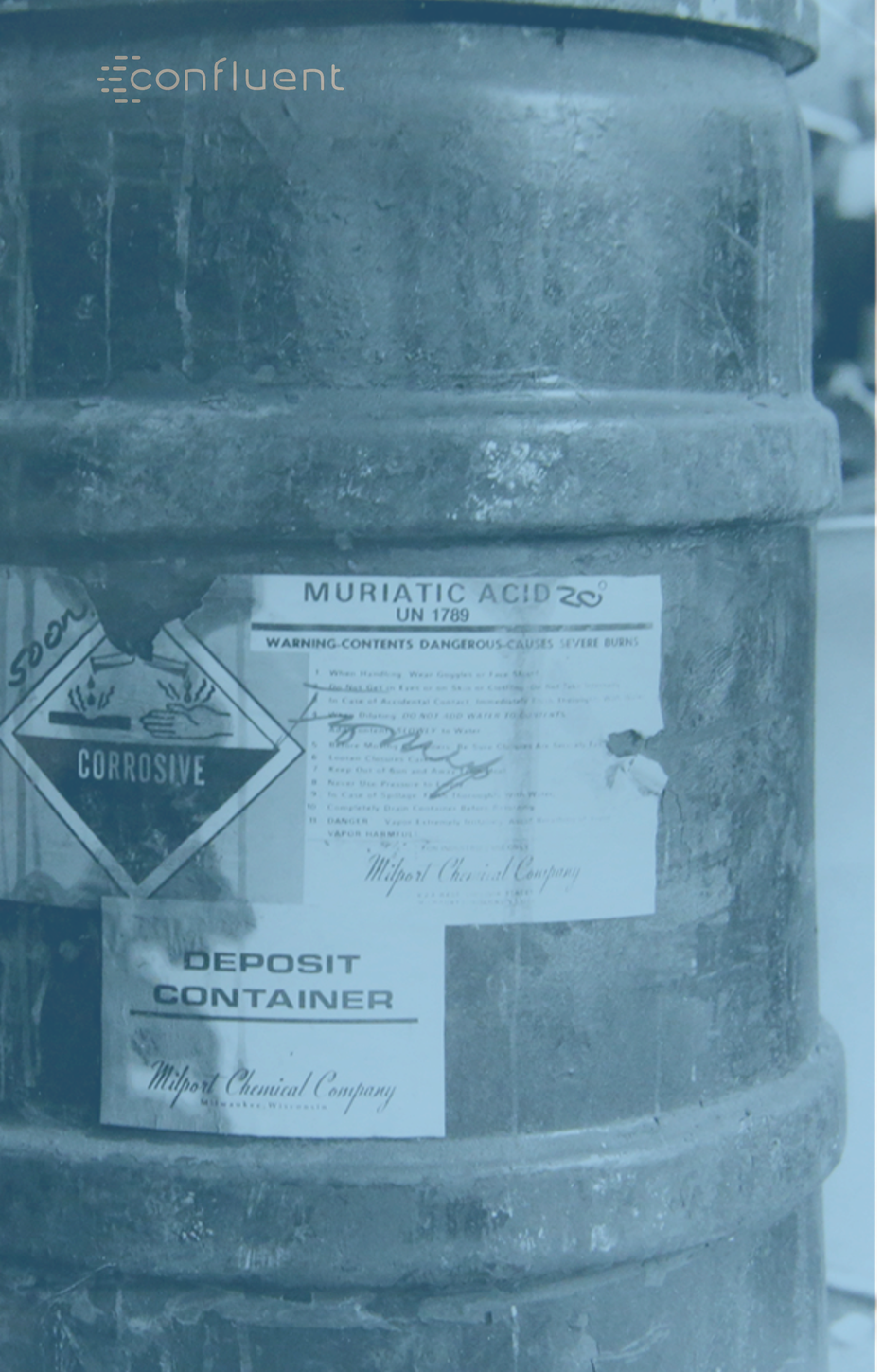
Brokers? Nope!



STATEFUL STREAM PROCESSING

What is State?

- Anything your application needs to "**remember**" beyond the scope of a single record



STATEFUL STREAM PROCESSING

Stateless operators:

- `filter, map, flatMap, foreach`

Stateful operator:

- Aggregations
- Any window operation
- Joins
- CEP

STOCK TRADE STATS EXAMPLE

```
KStream<String, Trade> source = builder.stream(STOCK_TOPIC);

KStream<Windowed<String>, TradeStats> stats = source
    .groupByKey()
    .windowedBy(TimeWindows.of(5000).advanceBy(1000))
    .aggregate(TradeStats::new,
               (k, v, tradestats) -> tradestats.add(v),
               Materialized.<~>as("trade-aggregates")
                   .withValueSerde(new TradeStatsSerde()))
    .toStream()
    .mapValues(TradeStats::computeAvgPrice);

stats.to(STATS_OUT_TOPIC,
         Produced.keySerde(WindowedSerdes.timeWindowedSerdeFrom(String.class)));
```

STOCK TRADE STATS EXAMPLE

```
KStream<String, Trade> source = builder.stream(STOCK_TOPIC);

KStream<Windowed<String>, TradeStats> stats = source
    .groupByKey()
    .windowedBy(TimeWindows.of(5000).advanceBy(1000))
    .aggregate(TradeStats::new,
               (k, v, tradestats) -> tradestats.add(v),
               Materialized.<~>as("trade-aggregates")
                   .withValueSerde(new TradeStatsSerde()))
    .toStream()
    .mapValues(TradeStats::computeAvgPrice);

stats.to(STATS_OUT_TOPIC,
         Produced.keySerde(WindowedSerdes.timeWindowedSerdeFrom(String.class)));
```

STOCK TRADE STATS EXAMPLE

```
KStream<String, Trade> source = builder.stream(STOCK_TOPIC);

KStream<Windowed<String>, TradeStats> stats = source
    .groupByKey()
    .windowedBy(TimeWindows.of(5000).advanceBy(1000))
    .aggregate(TradeStats::new,
               (k, v, tradestats) -> tradestats.add(v),
               Materialized.<~>as("trade-aggregates")
                   .withValueSerde(new TradeStatsSerde()))
    .toStream()
    .mapValues(TradeStats::computeAvgPrice);

stats.to(STATS_OUT_TOPIC,
         Produced.keySerde(WindowedSerdes.timeWindowedSerdeFrom(String.class)));
```

STOCK TRADE STATS EXAMPLE

```
KStream<String, Trade> source = builder.stream(STOCK_TOPIC);

KStream<Windowed<String>, TradeStats> stats = source
    .groupByKey()
    .windowedBy(TimeWindows.of(5000).advanceBy(1000))
    .aggregate(TradeStats::new,
               (k, v, tradestats) -> tradestats.add(v),
               Materialized.<~>as("trade-aggregates")
                   .withValueSerde(new TradeStatsSerde()))
    .toStream()
    .mapValues(TradeStats::computeAvgPrice);

stats.to(STATS_OUT_TOPIC,

Produced.keySerde(WindowedSerdes.timeWindowedSerdeFrom(String.class)));
```

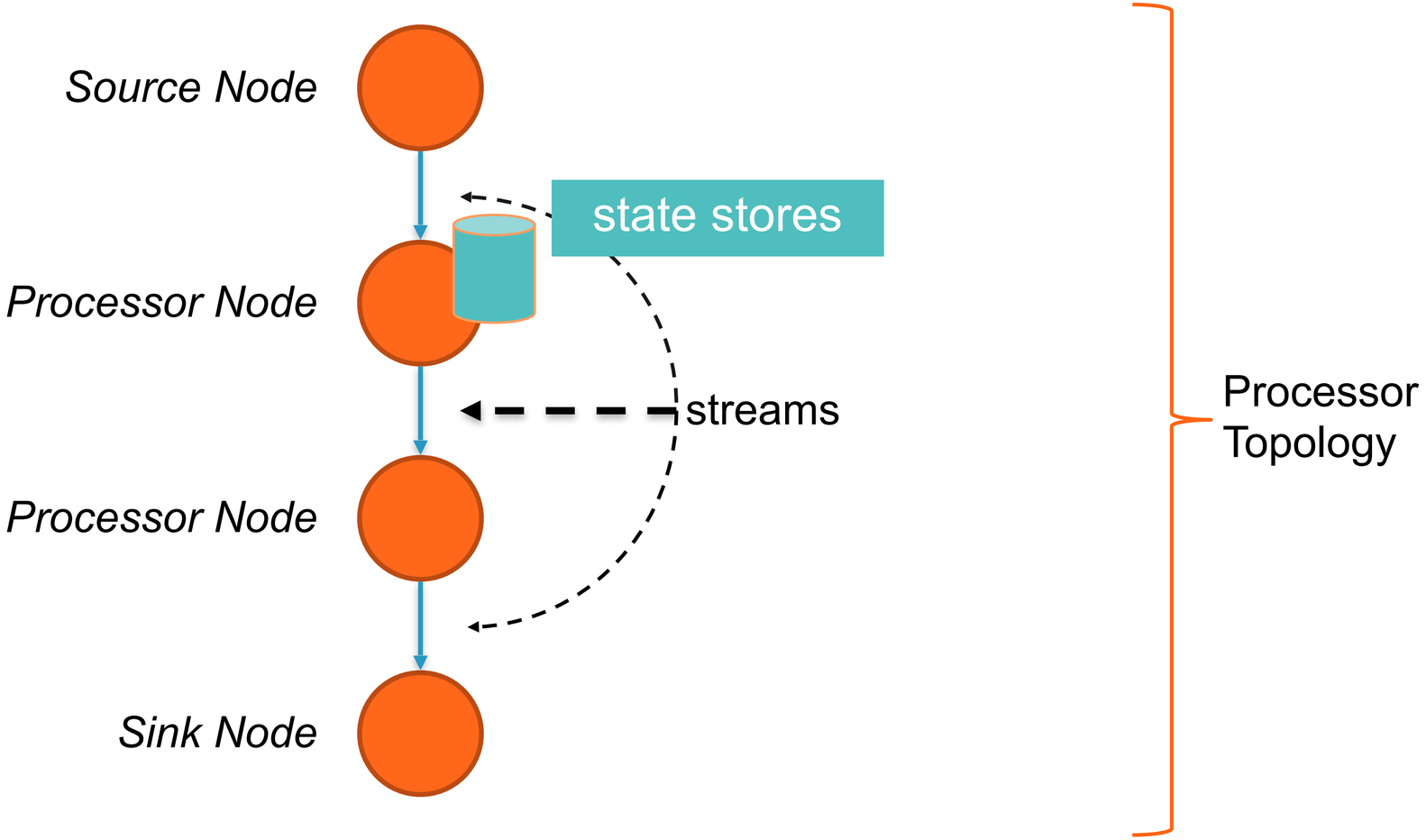
TOPOLOGIES

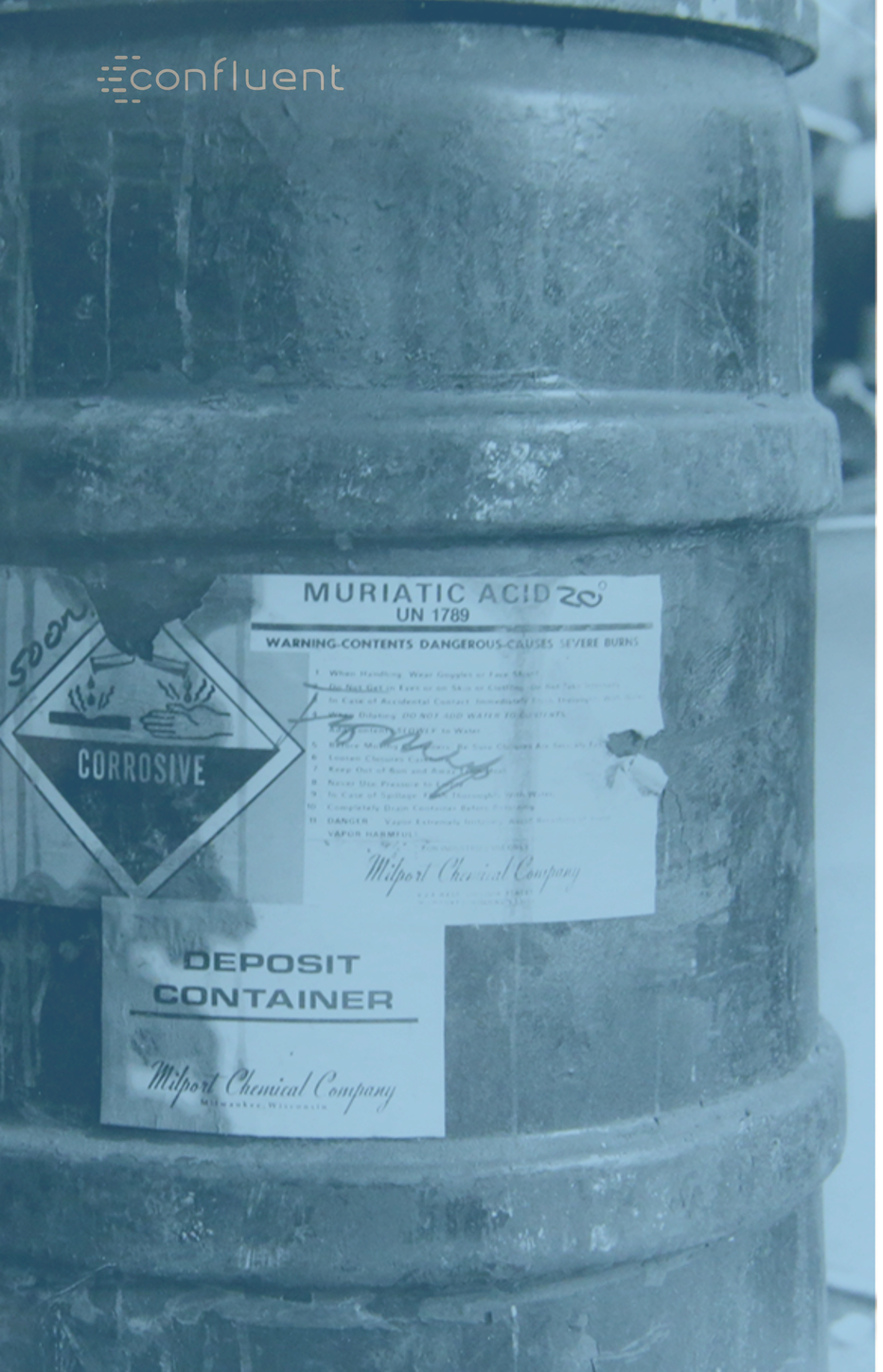
```
builder.stream

source.groupByKey
        .windowedBy(…)
        .aggregate(…)

mapValues()

to(…)
```





ACCESSING APPLICATION STATE

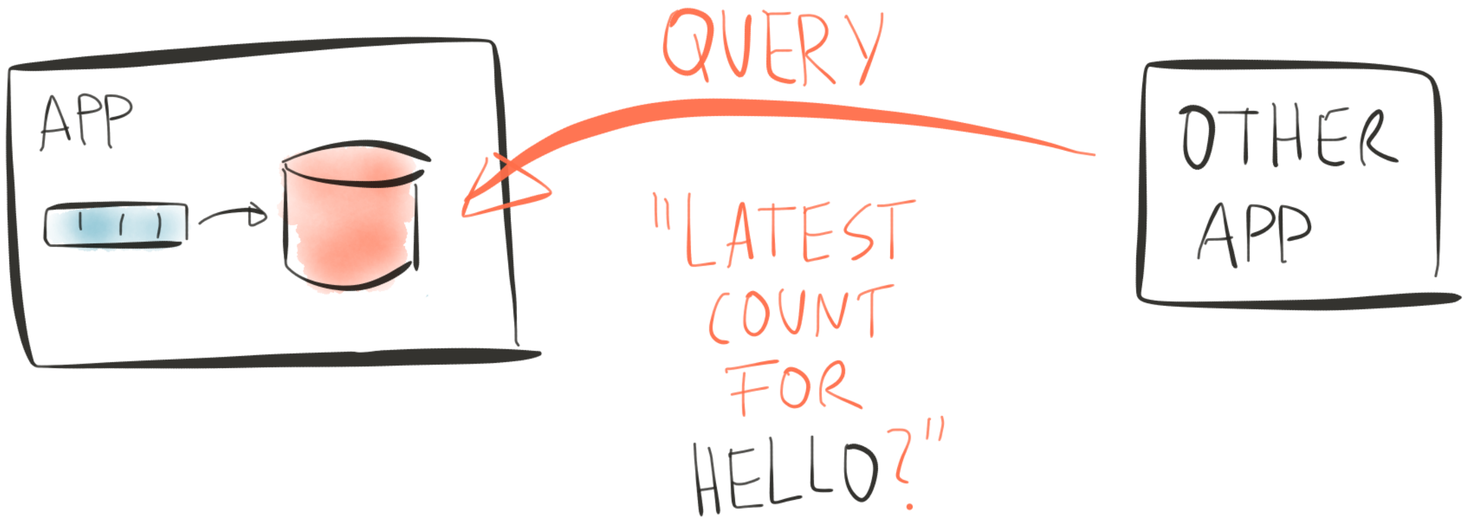
External DB: query database

Local State: Interactive Queries

- Allows to query the local DB instances
- Kafka Streams application is the database
- Read-Only access (no “external” state updates)
- Works for DSL or Processor API

ACCESSING APPLICATION STATE – "DB TO GO"

Interactive Queries turn Kafka Streams application into queryable light-weight database

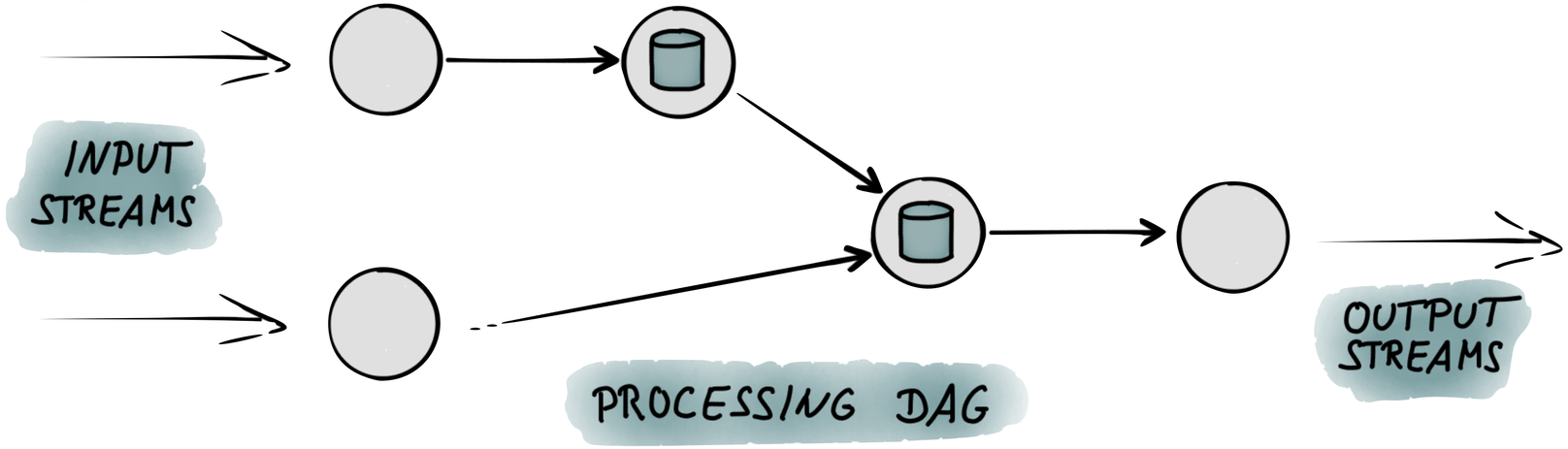


KAFKA STREAMS USES LOCAL STATE

Light-weight embedded DB

Local data access

No RPC, network I/O



SCALING LOCAL STATE

Topics are scaled via partitions

Kafka Streams scales via tasks

- 1:1 mapping from input topic partitions to tasks

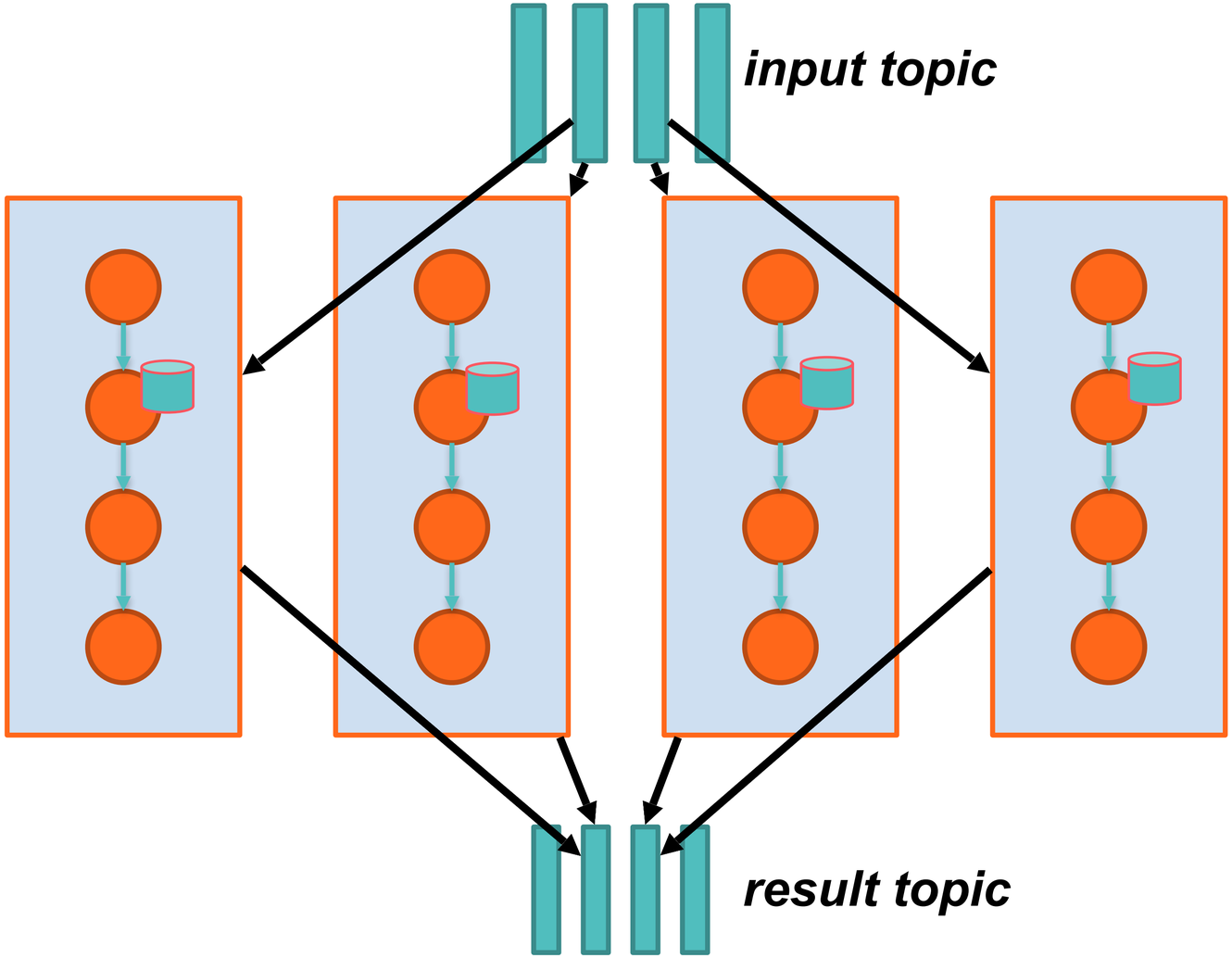
State can be sharded based on partitions

- Local DB for each shard
- 1:1 mapping from state shard to task

PARTITIONS, TASKS, AND CONSUMER GROUPS

Task executes processor topology

4 input topic partitions
=> 4 tasks

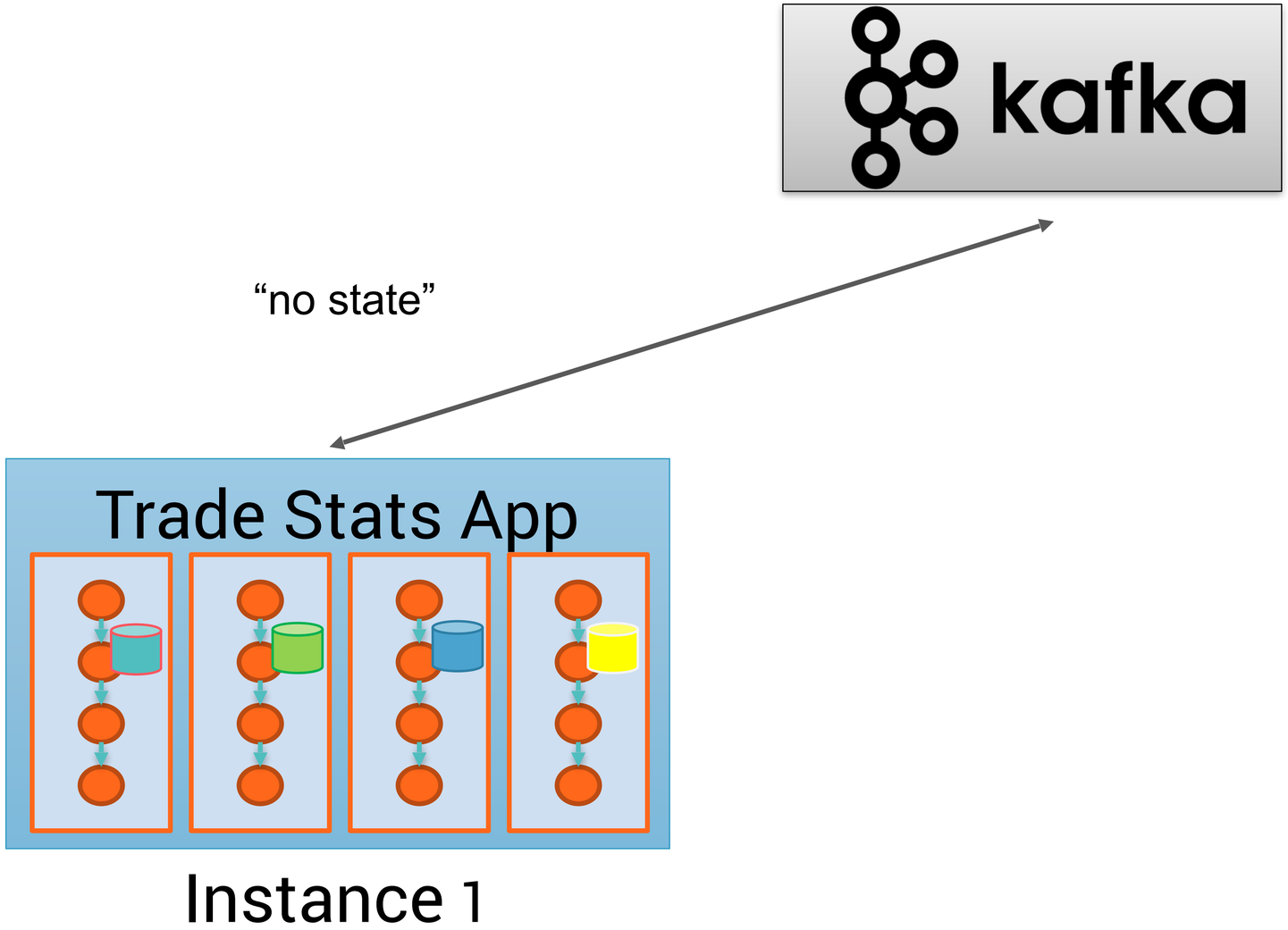


One **consumer group**:

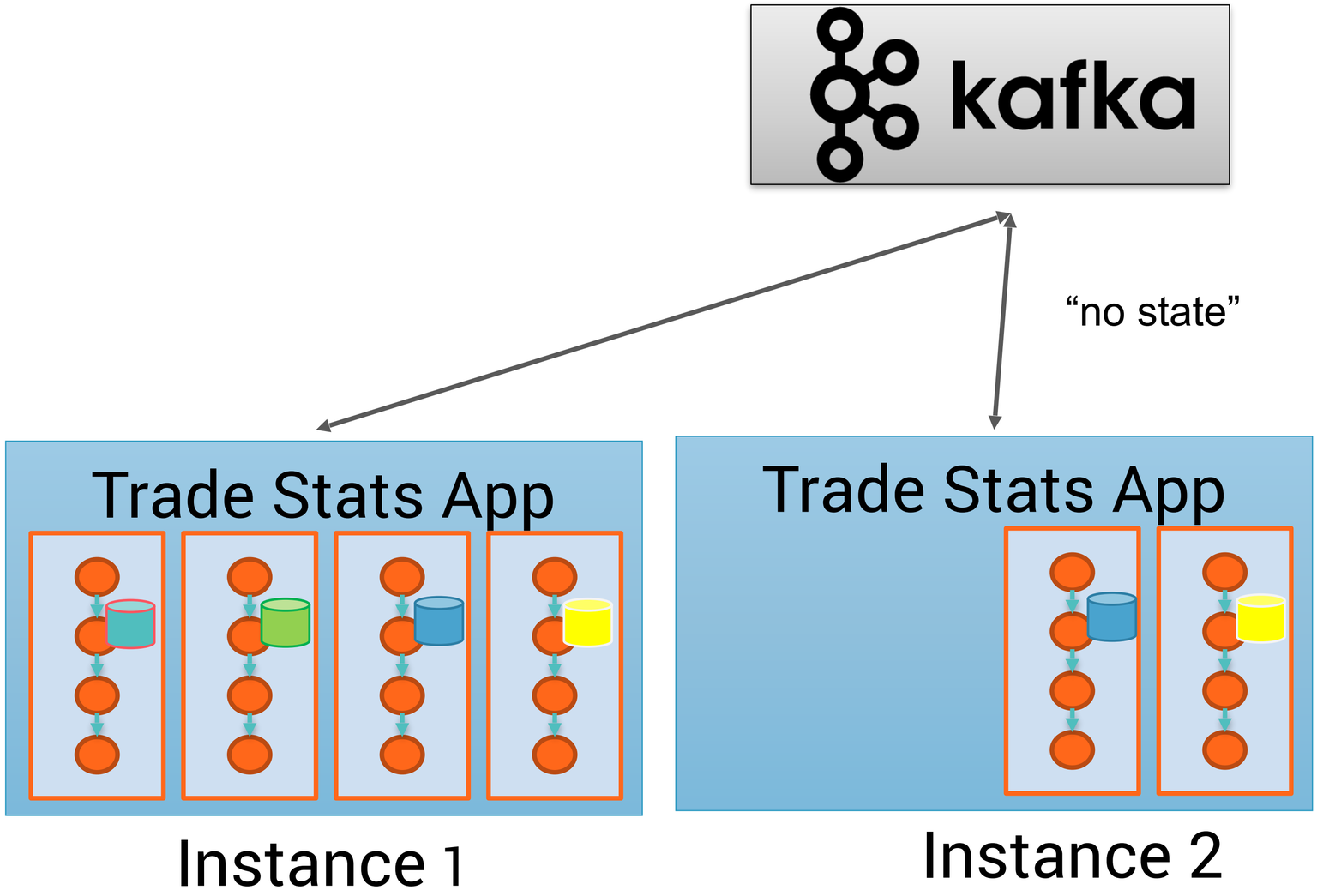
can be executed with

1 - 4 threads on
1 - 4 machines

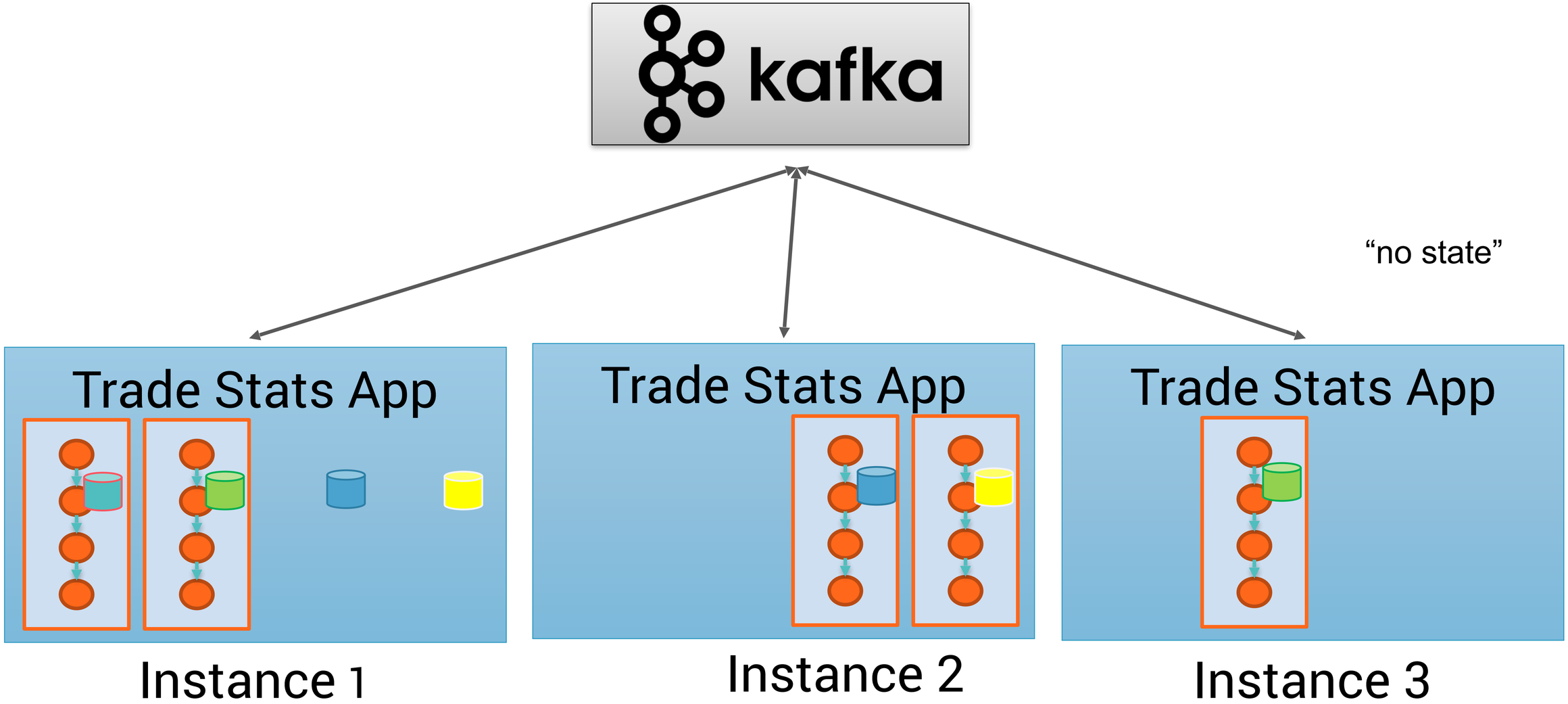
SCALING WITH STATE



Scaling with State



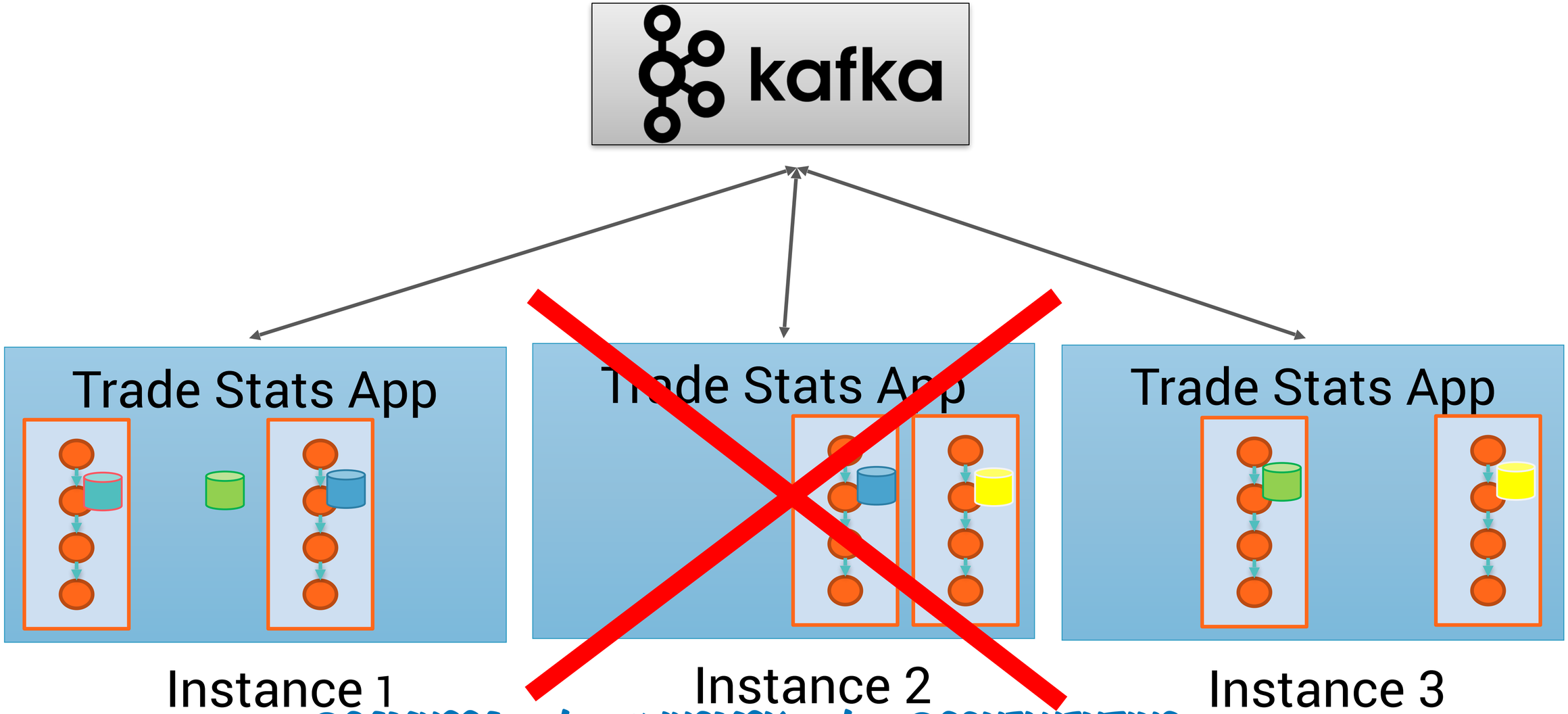
SCALING WITH STATE



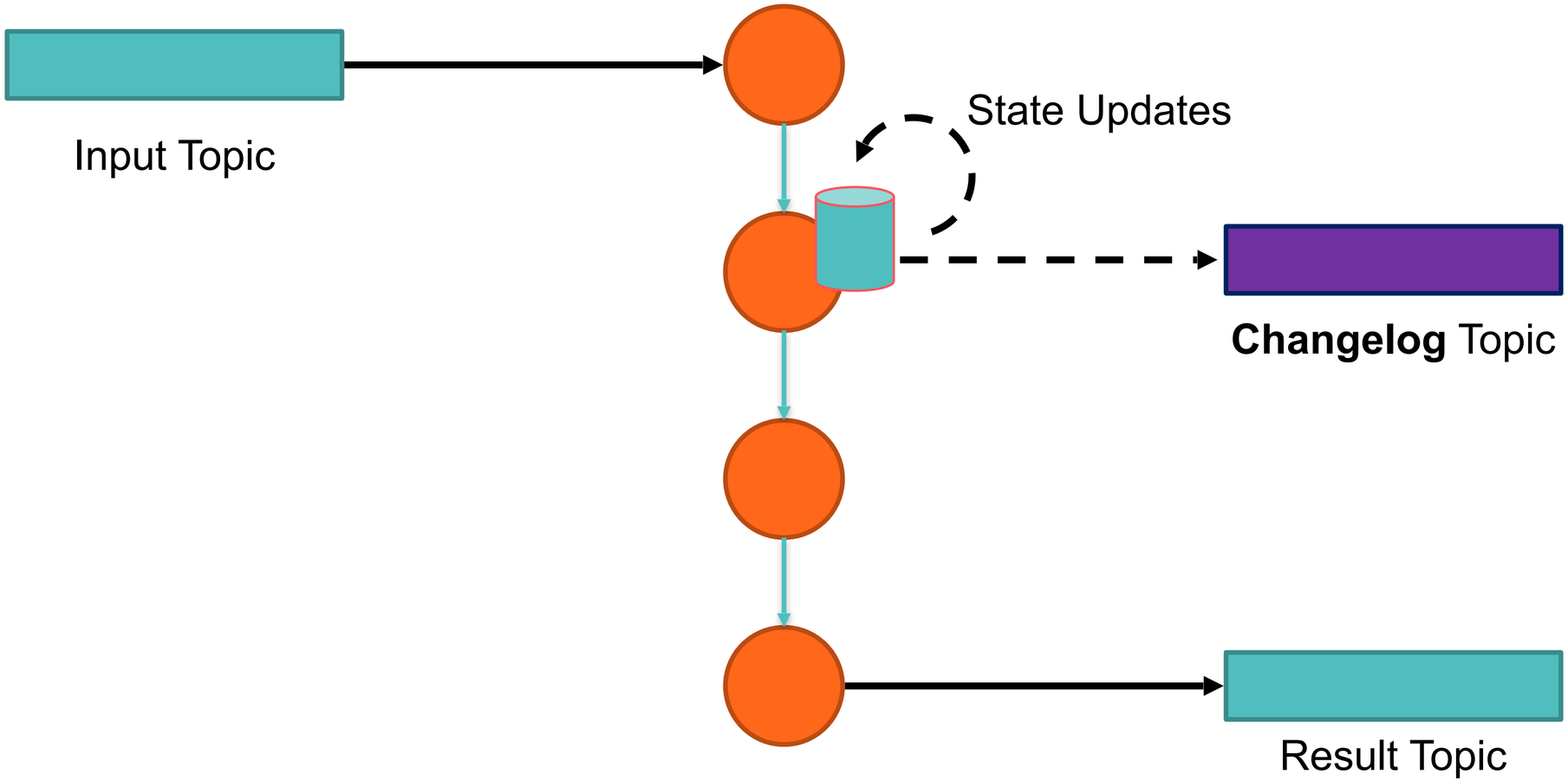
SCALING AND FAULT-TOLERANCE

TWO SIDES OF SAME COIN

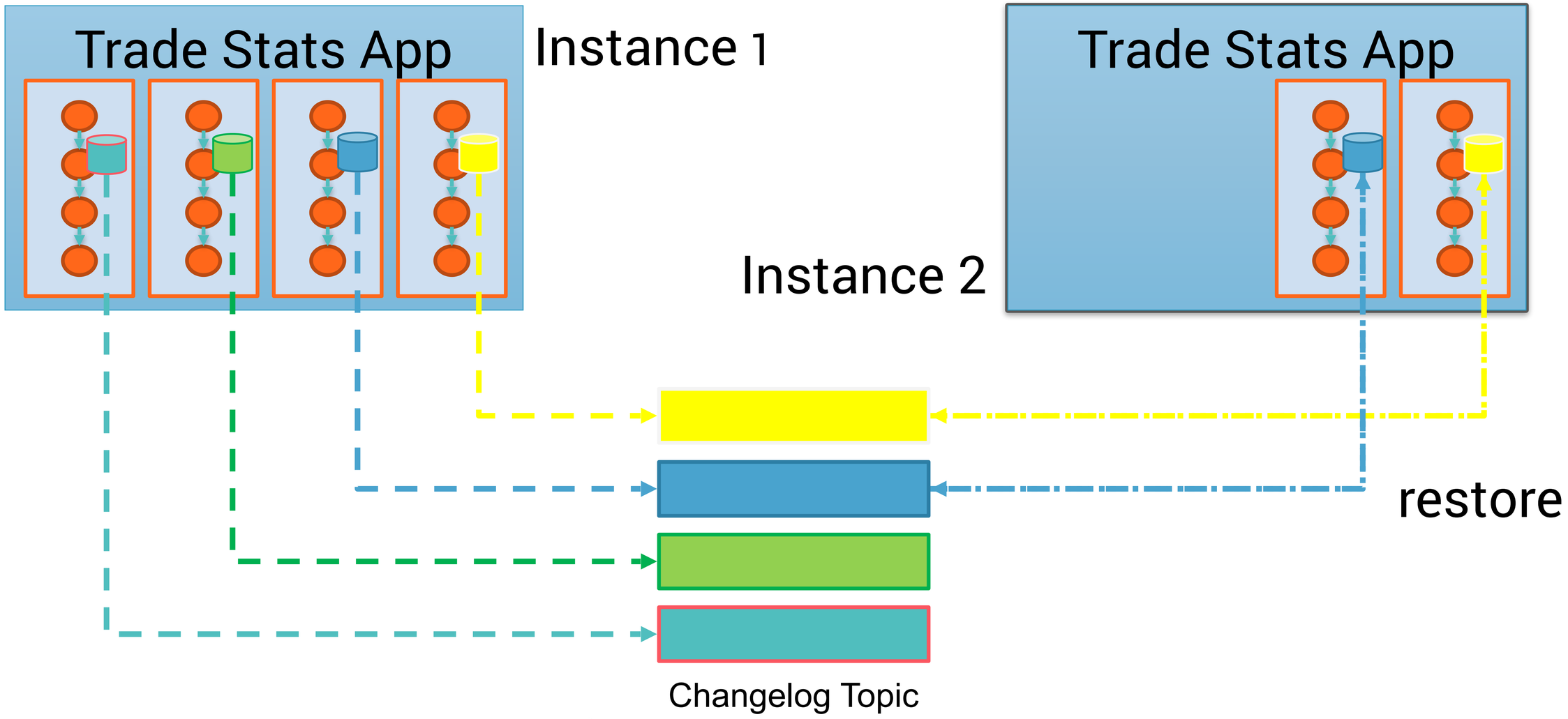
FAULT-TOLERANCE



FAULT-TOLERANT STATE



MIGRATE STATE



RECOVERY TIME

Changelog topics are log compacted

Size of *changelog* topic linear in size of state

Large state implies **high** recovery times

RECOVERY OVERHEAD

Recovery overhead is proportional to

- `segment-size / state-size`

Segment-size is *smaller* than state-size => reduced overhead

Update changelog topic segment size accordingly

- topic config: `log.segments.bytes`
- log cleaner interval important, too

confluent

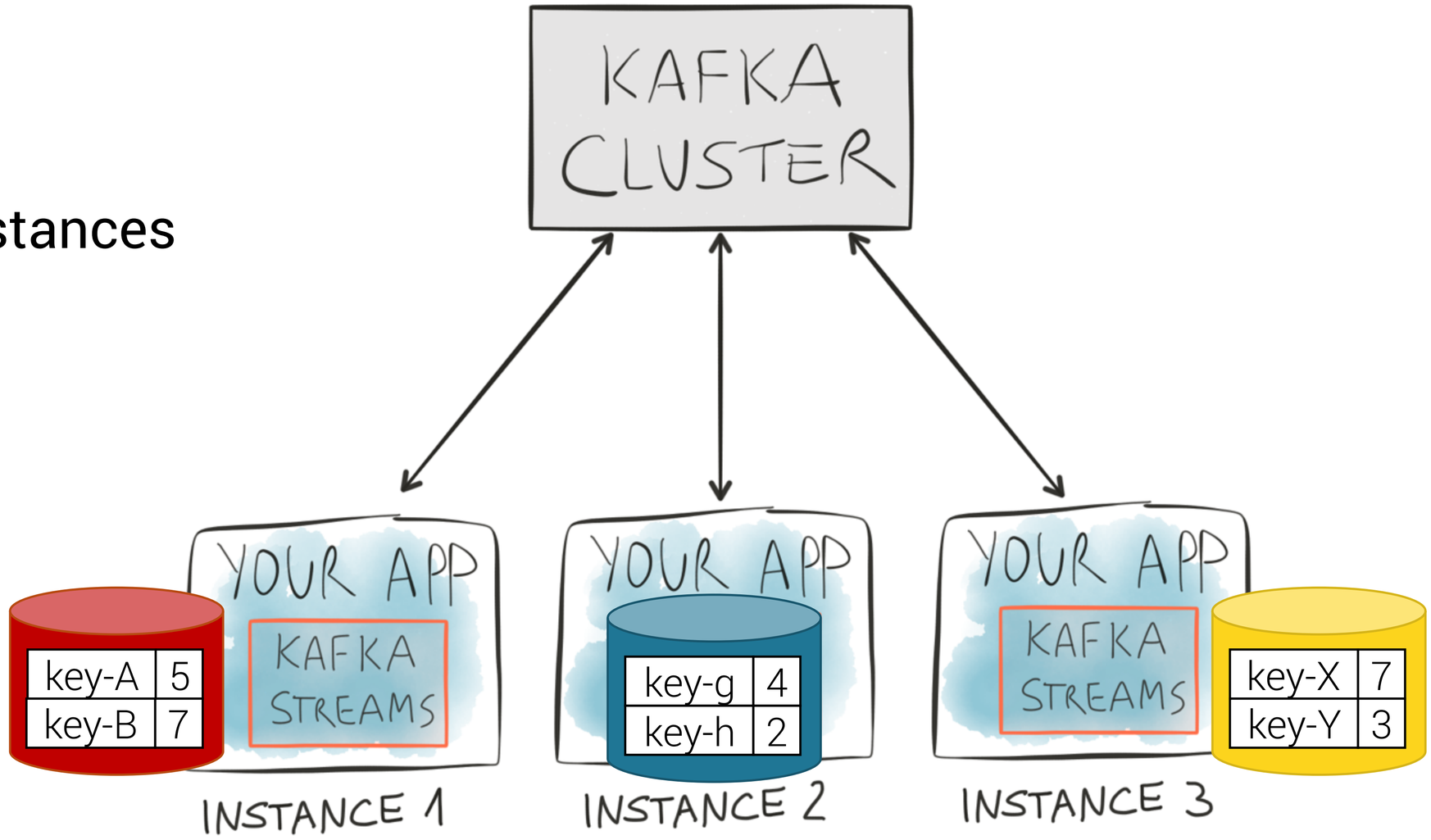
QUERY YOUR APP...
NOT DATABASE

HOW TO USE INTERACTIVE QUERIES

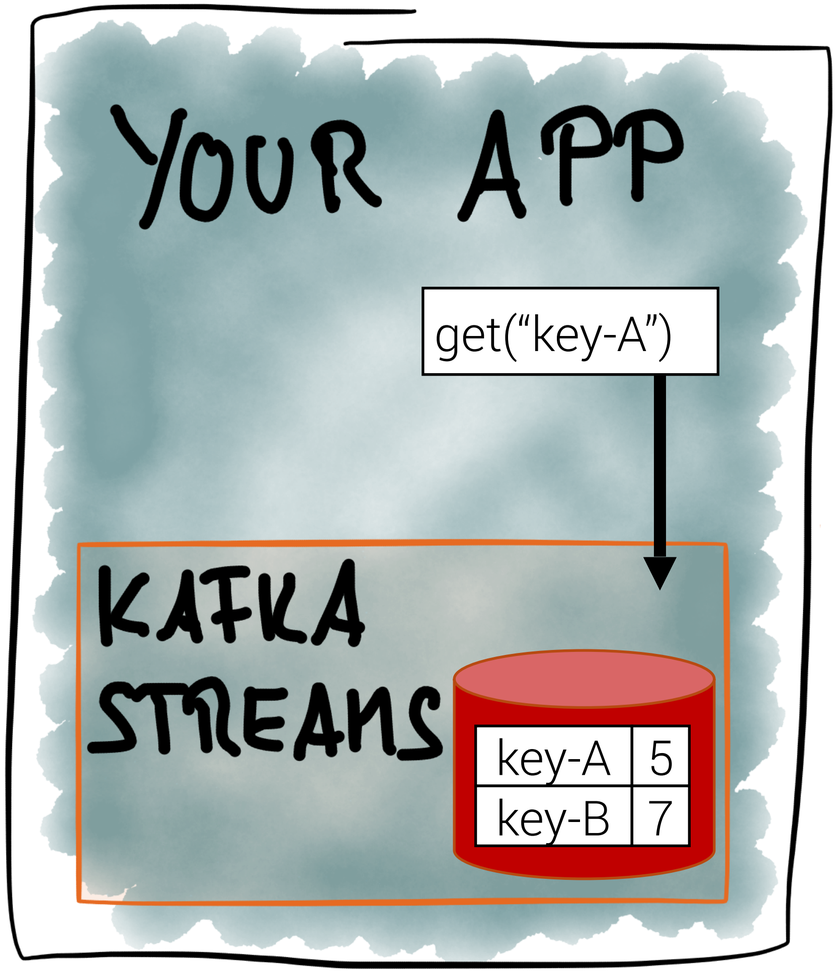
Query local state

Discover remote instances

Query remote state



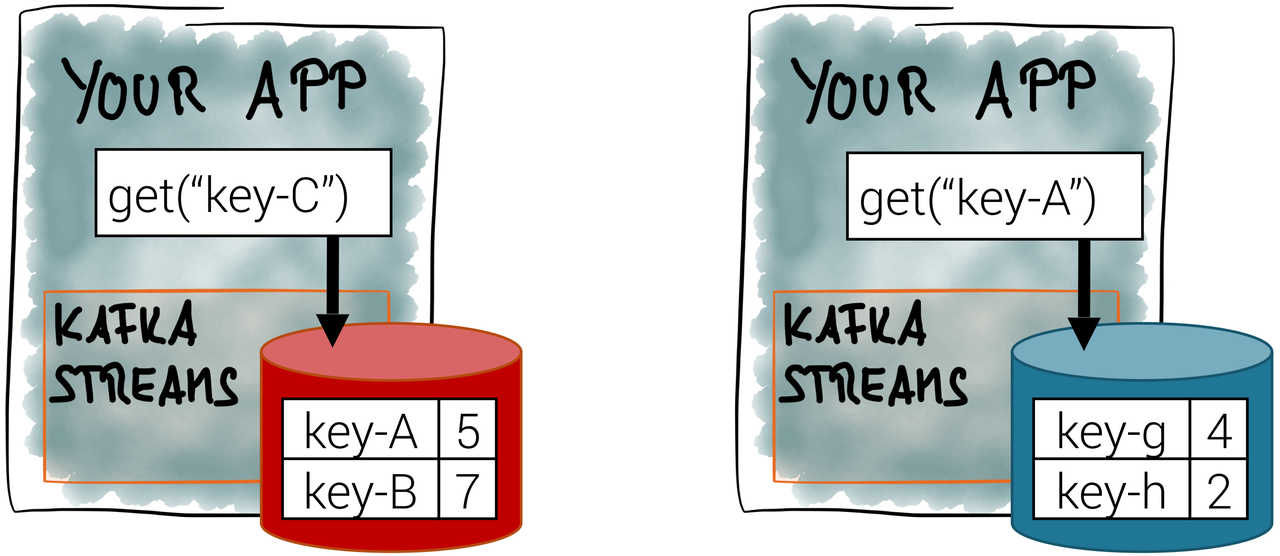
QUERY LOCAL STATE



DISCOVER REMOTE INSTANCES

Kafka Streams application instances are started independently

- What other instances are there?
- Which instances hold which data?



DISCOVER REMOTE INSTANCES

Kafka Streams *metadata API* to the rescue:

- `KafkaStreams#allMetadata()`
- `KafkaStreams#allMetadataForStore(String storeName)`
- `KafkaStreams#allMetadataForKey(`
  `String storeName,`
  `K key,`
  `Serializer<K> keySerializer)`

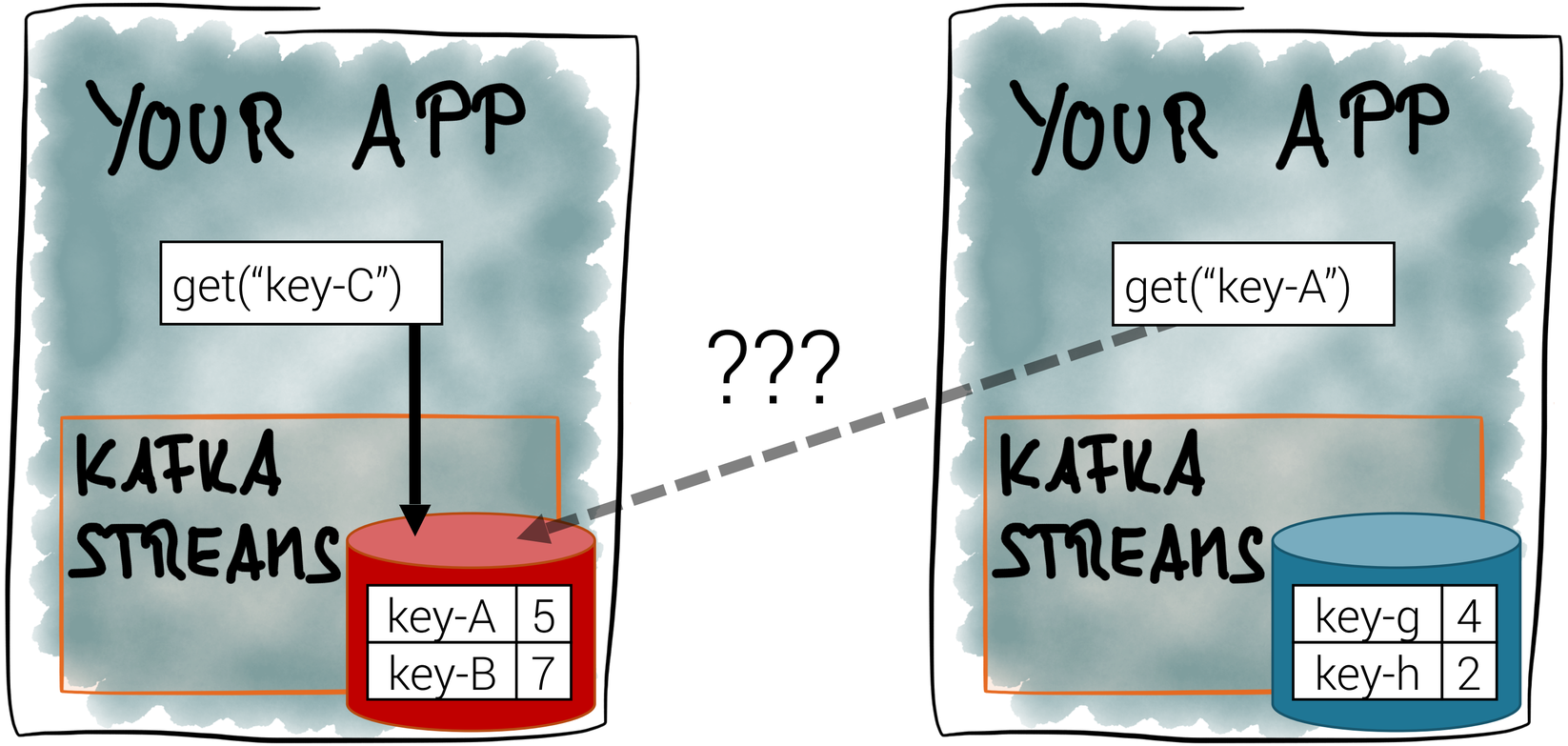
Returns `StreamsMetadata` object

DISCOVER REMOTE INSTANCES

`StreamsMetadata` object

- Host information (server/port) according to `application.server` config
- State store names of hosted stores
- `TopicPartition` mapped to the store (ie, shard)

HOW TO ACCESS DATA OF ANOTHER INSTANCE?

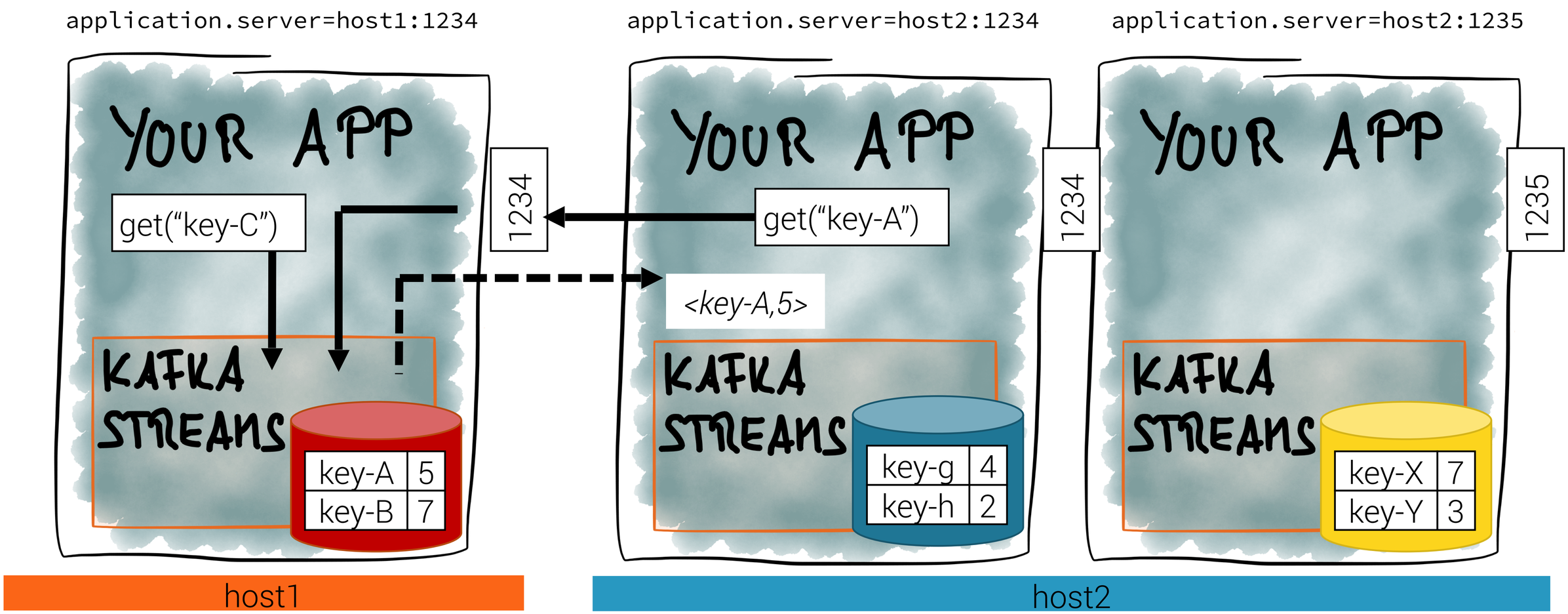


QUERY REMOTE STATE VIA RPC

Choose whatever you like:

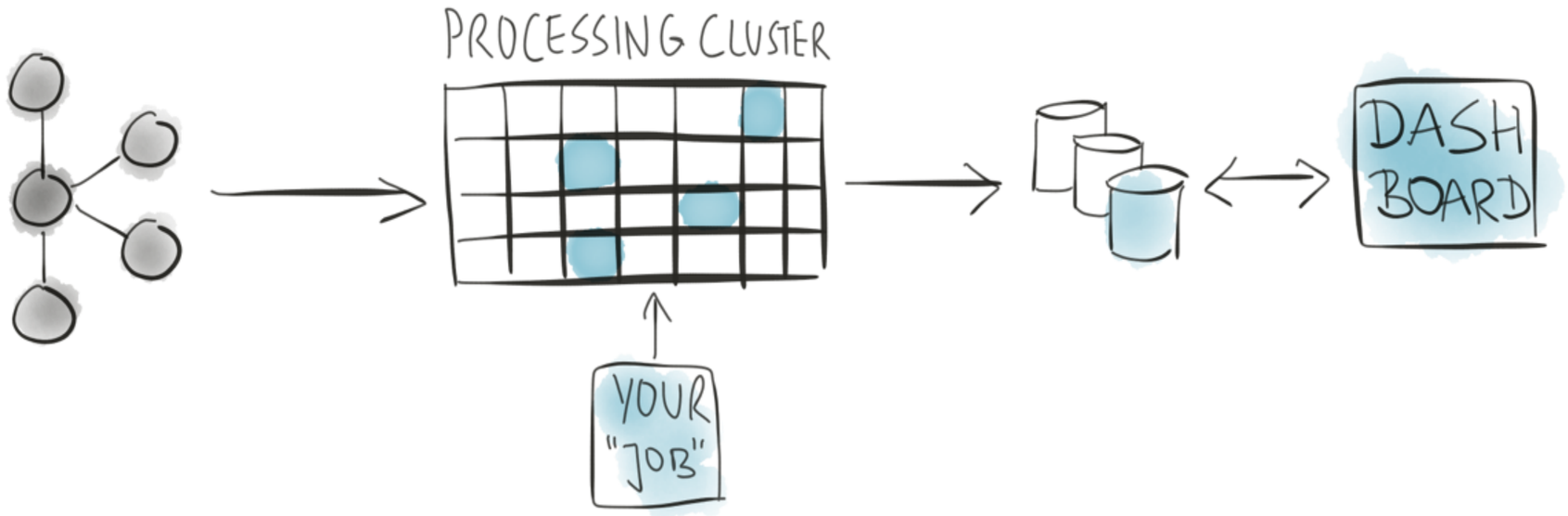
- REST
- gRPC
- Apache Thrift
- XML-RPC
- Java RMI
- SOAP

QUERY REMOTE STATE

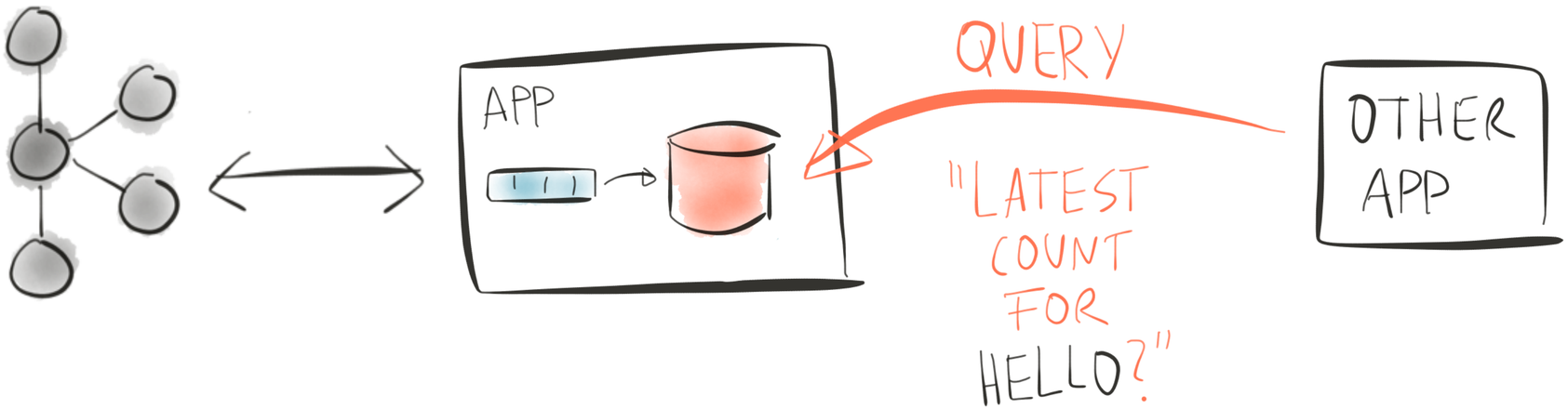


TAKE HOME MESSAGE

Before IQ ☹️



With IQ ☺️



PROS

There are fewer moving pieces; no external database

It enables faster and more efficient use of the application state

It provides better isolation

It allows for flexibility

Mix-and-match local an remote DB

CONS

It may involve moving away from a datastore you know and trust

You might want to scale storage independently of processing

You might need customized queries, specific to some datastores

🛑 STOP!
DEMO TIME!

SUMMARY

Interactive Queries is a powerful abstractions that simplifies stateful stream processing

There are still cases for which external database/storage might be a better fit – with IQ you can choose!

SUMMARY

Music example: https://github.com/confluentinc/examples/blob/master/kafka-streams/src/main/java/io/confluent/examples/streams/interactivequeries/kafkamusic/KafkaMusicExample.java

Streaming Movie Ratings: https://github.com/confluentinc/demo-scene/tree/master/streams-movie-demo

THANKS!

@gamussa
viktor@confluent.io

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

confluent