HERACLITUS TEACHES

Kafka Streams

(AND LEARNS TO STOP CRYING!)

@tlberglund

HERACLITUS TEACHES

Kafka Streams

(AND LEARNS TO STOP CRYING!)

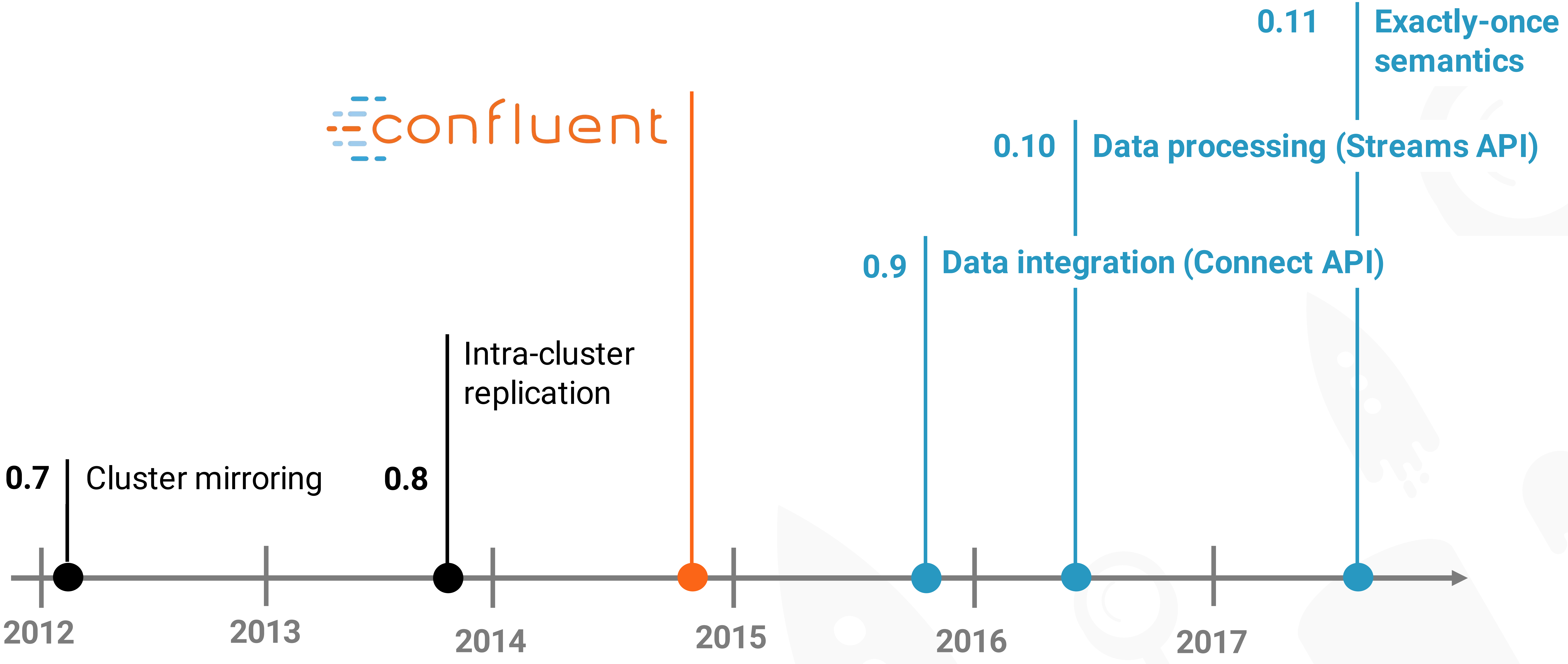
@tlberglund

Heraclitus

- Lived 535-475 BC in Ephesus
- Wrestled with problems of metaphysics
- Struggled with depression
- Probably did not use Kafka Streams

Heraclitus

- Tension of opposites
- Fire
- All things change
- *“No one steps into the same river twice.”*



0.7 Cluster mirroring

0.8 Intra-cluster replication

confluent

0.9 Data integration (Connect API)

0.10 Data processing (Streams API)

0.11 Exactly-once semantics

2012 2013 2014 2015 2016 2017

As developers, we want to build

APPS

not

INFRASTRUCTURE

We want our apps to be:

- Scalable
- Elastic
- Fault-tolerant
- Stateful
- Distributed

Where do I put my compute?

Where do I put my state?

The actual question is

Where is my code?

the

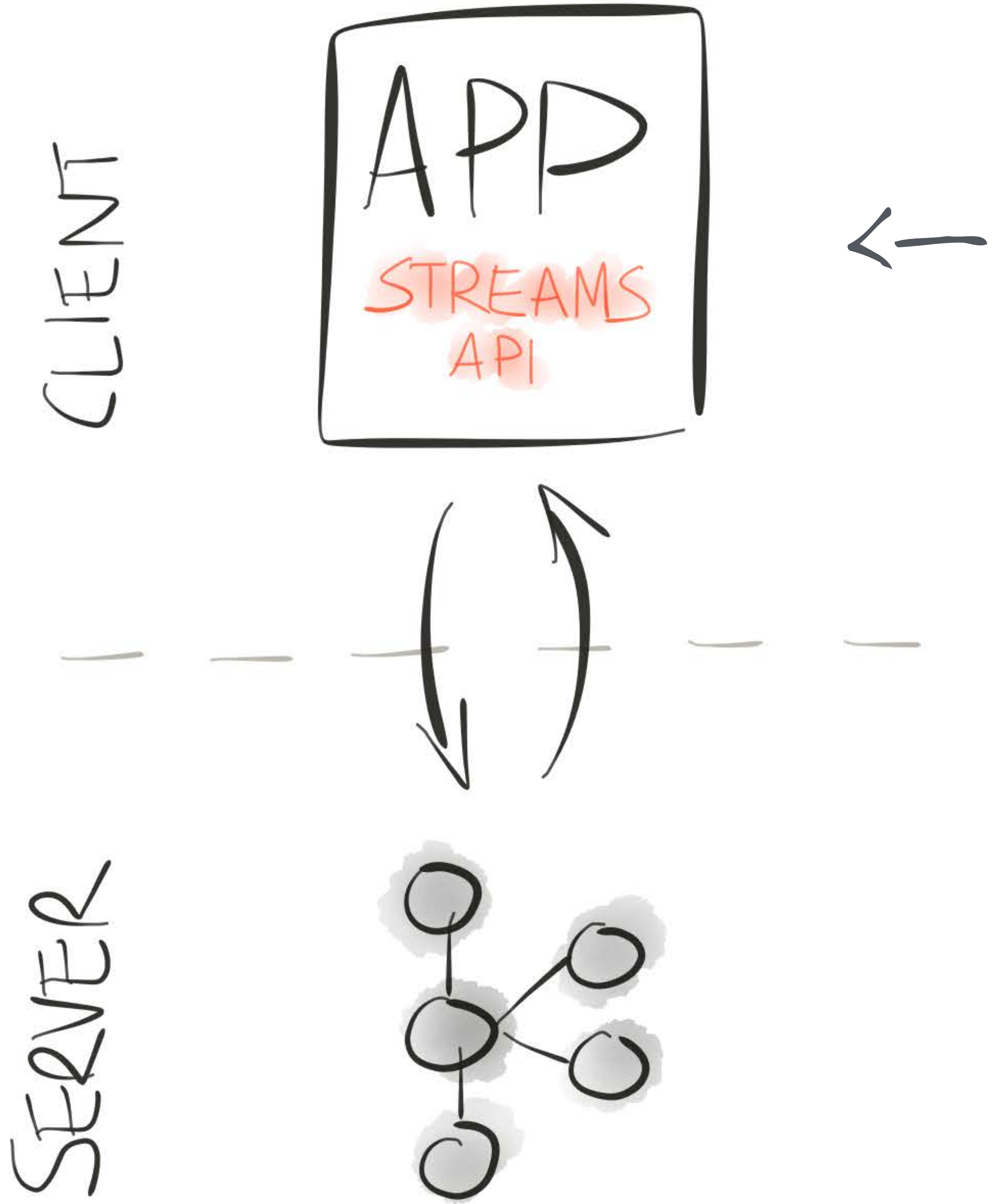
KAFKA STREAMS API

is a

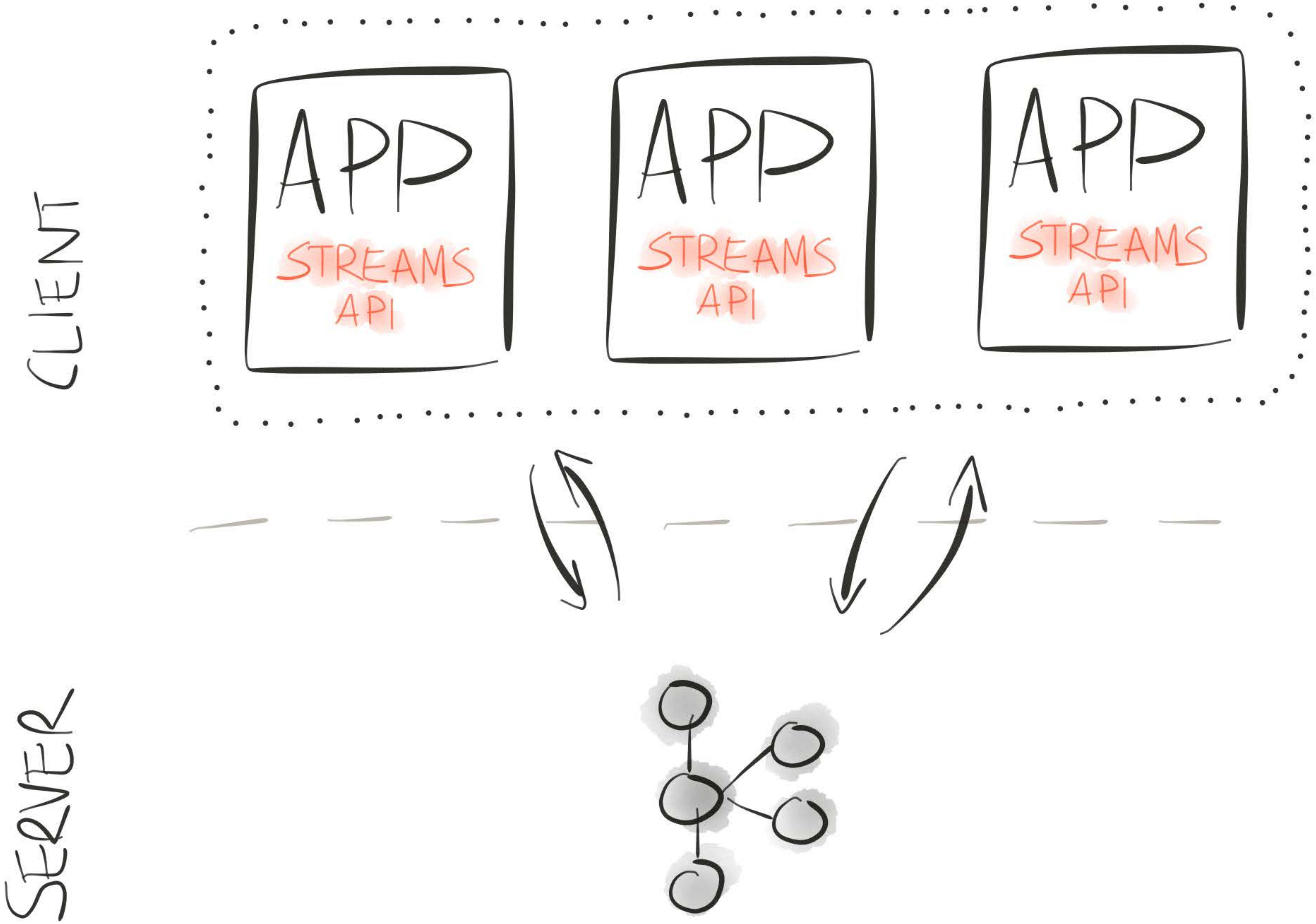
JAVA API

to build real-time applications

TO POWER THE BUSINESS

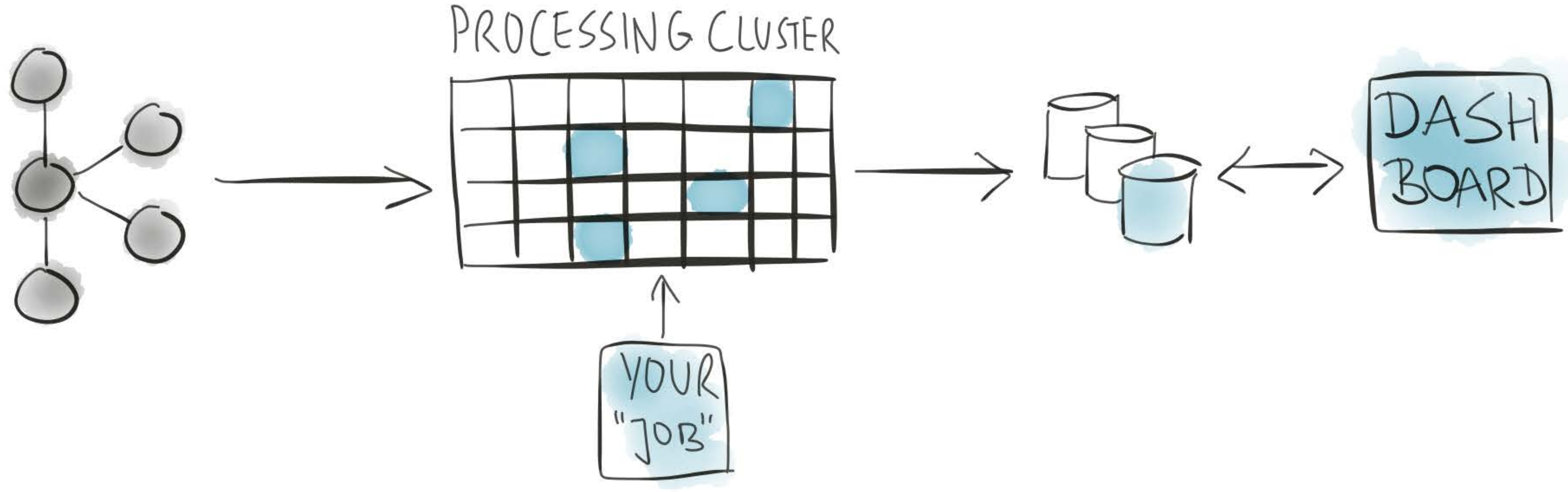


← Not running inside brokers!

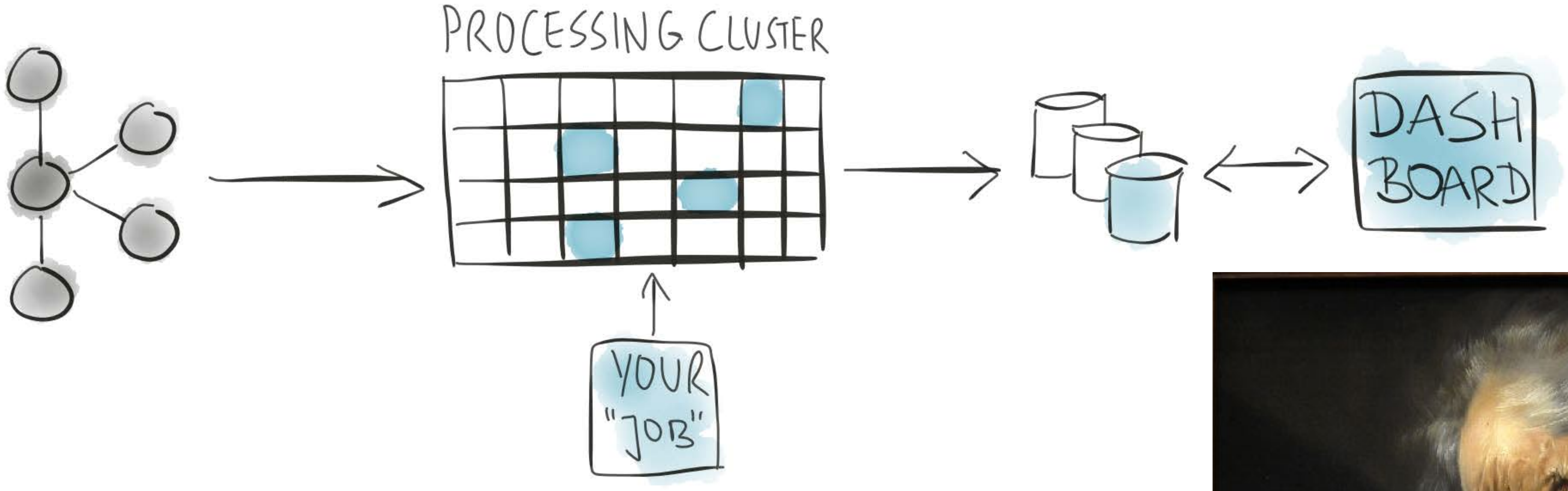


Brokers?
Still nope!

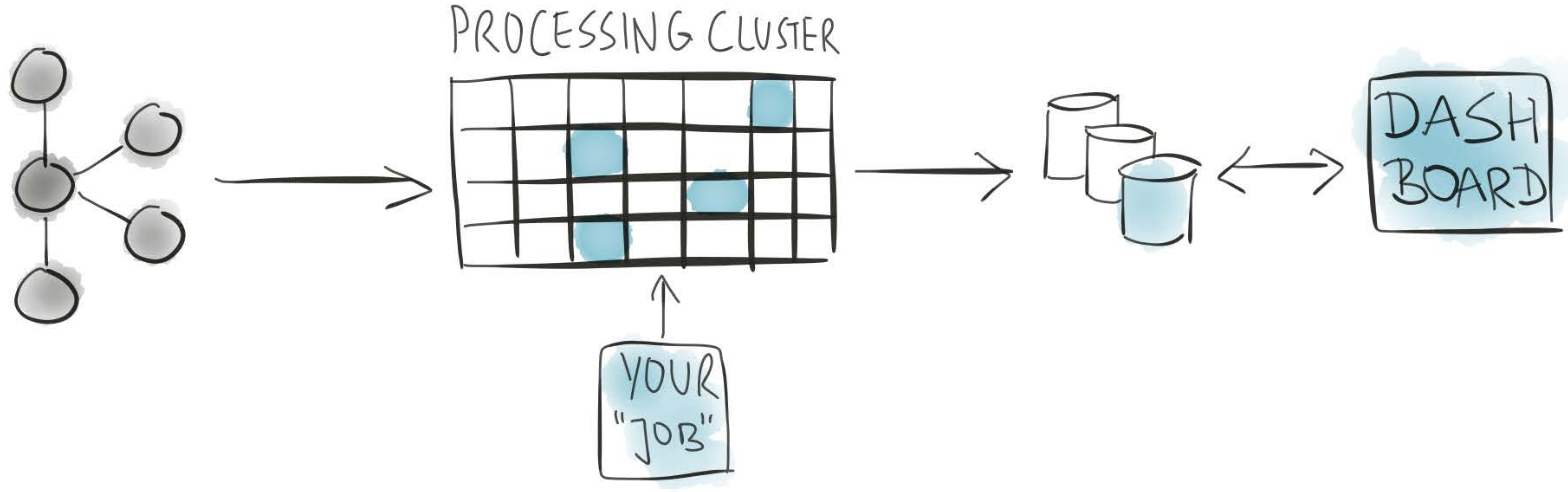
Before



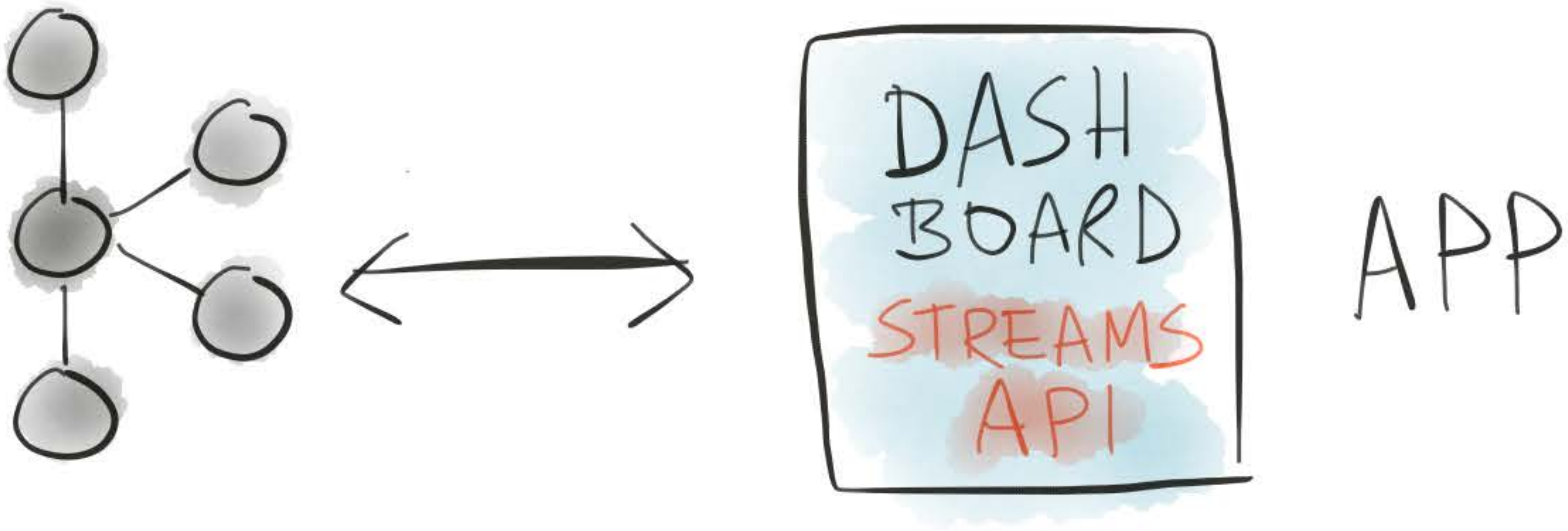
Before



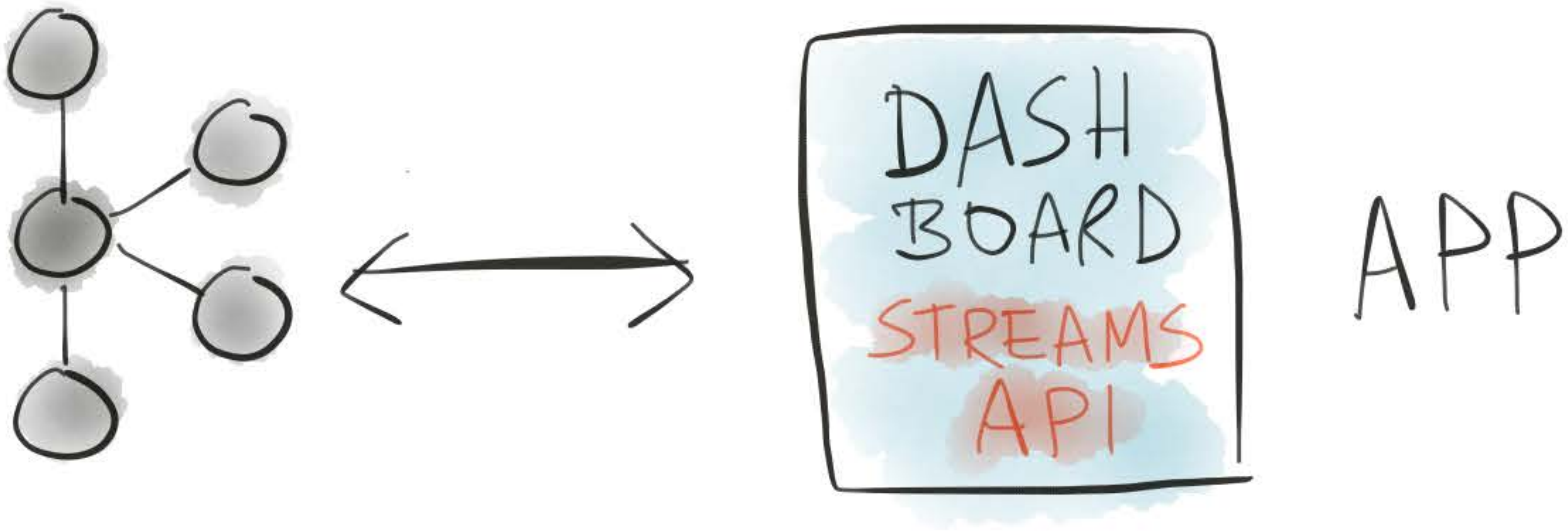
Before



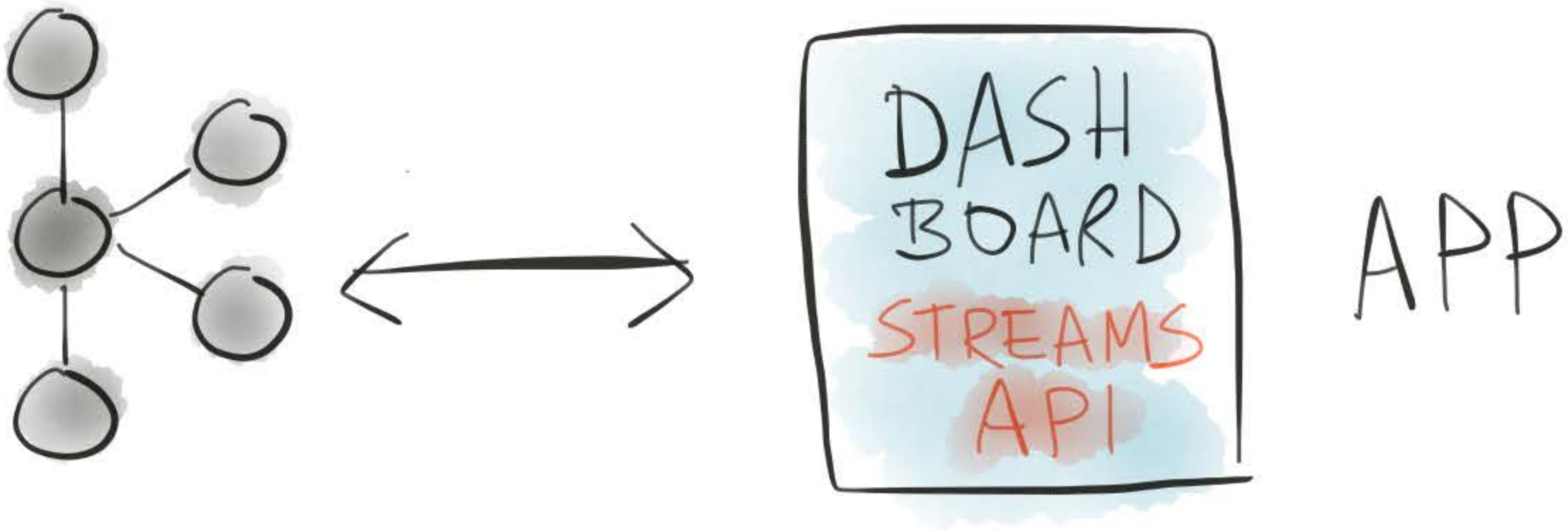
After



After



After



confluent

this means you can

DEPLOY

your app

ANYWHERE

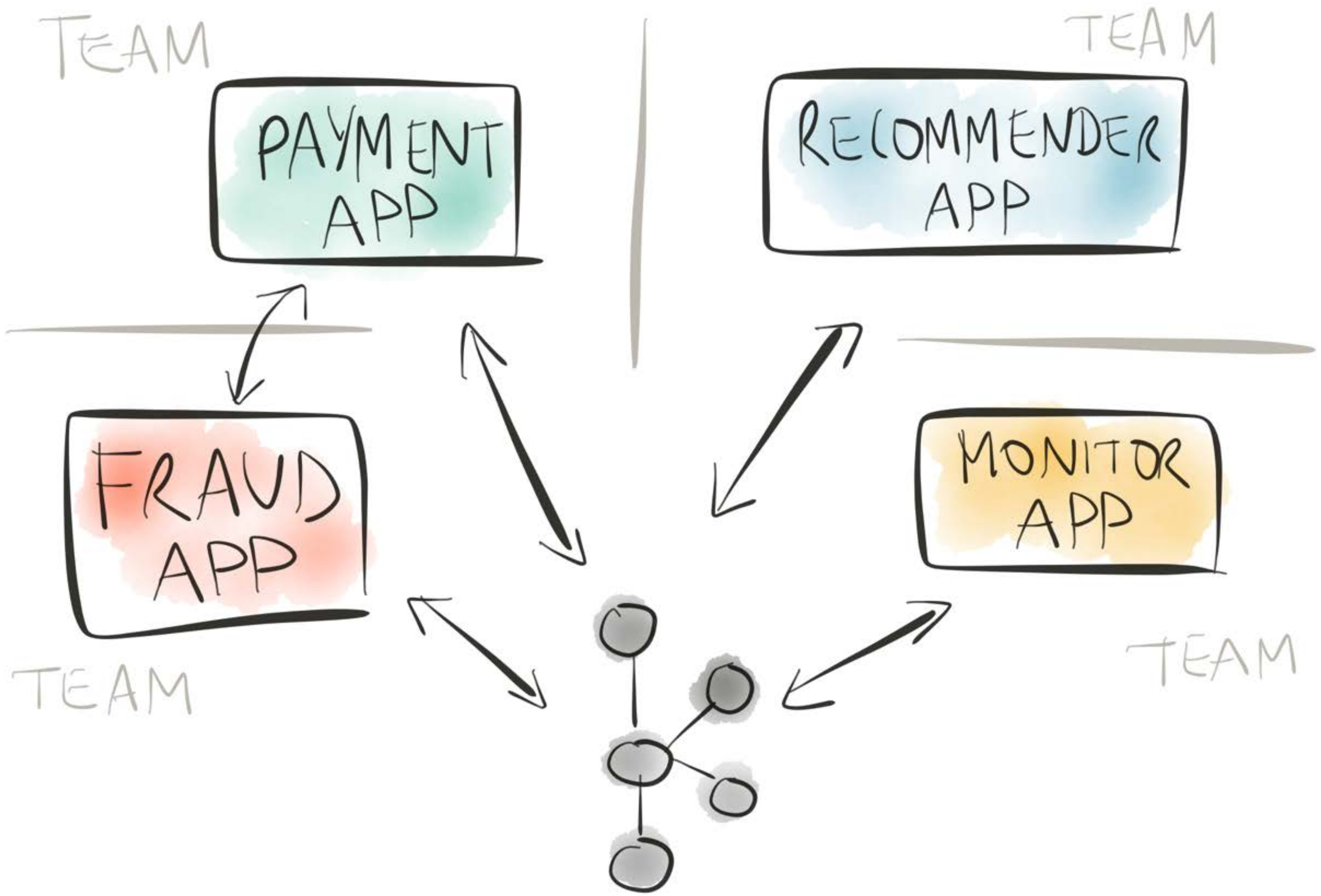
using whatever technology

YOU WANT

Things Kafka Streams Does

- Runs everywhere
- Clustering done for you
- Exactly-once processing
- Event-time processing
- Integrated database
- Joins, windowing, aggregation
- S/M/L/XL/XXL/XXXL sizes

An integration story?



For another time...

first, some

API CONCEPTS

STREAMS

are

EVERYWHERE

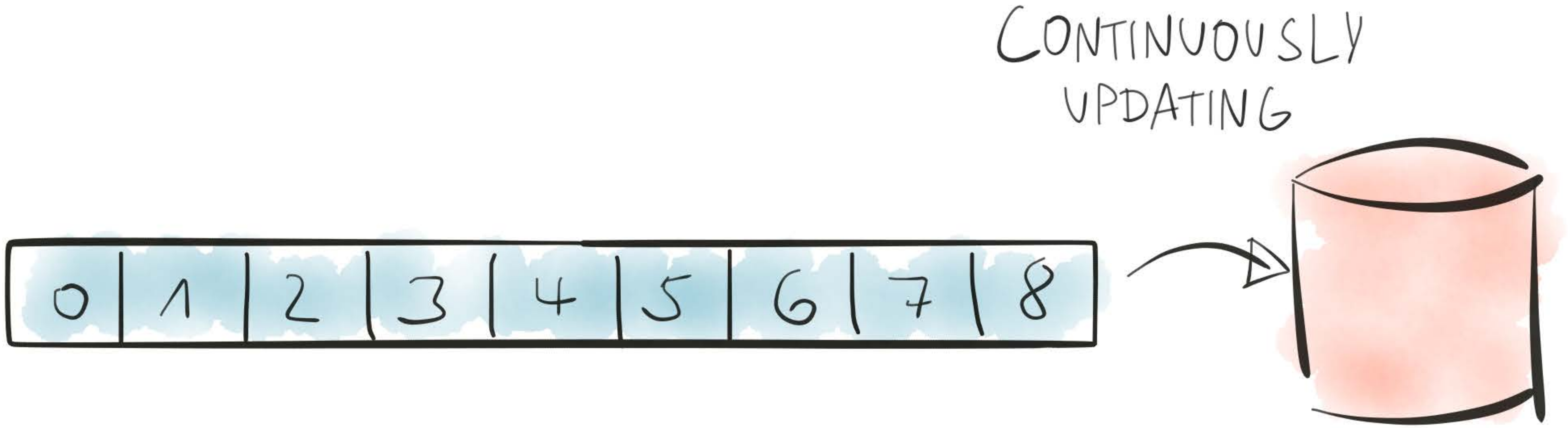
confluent

TABLES

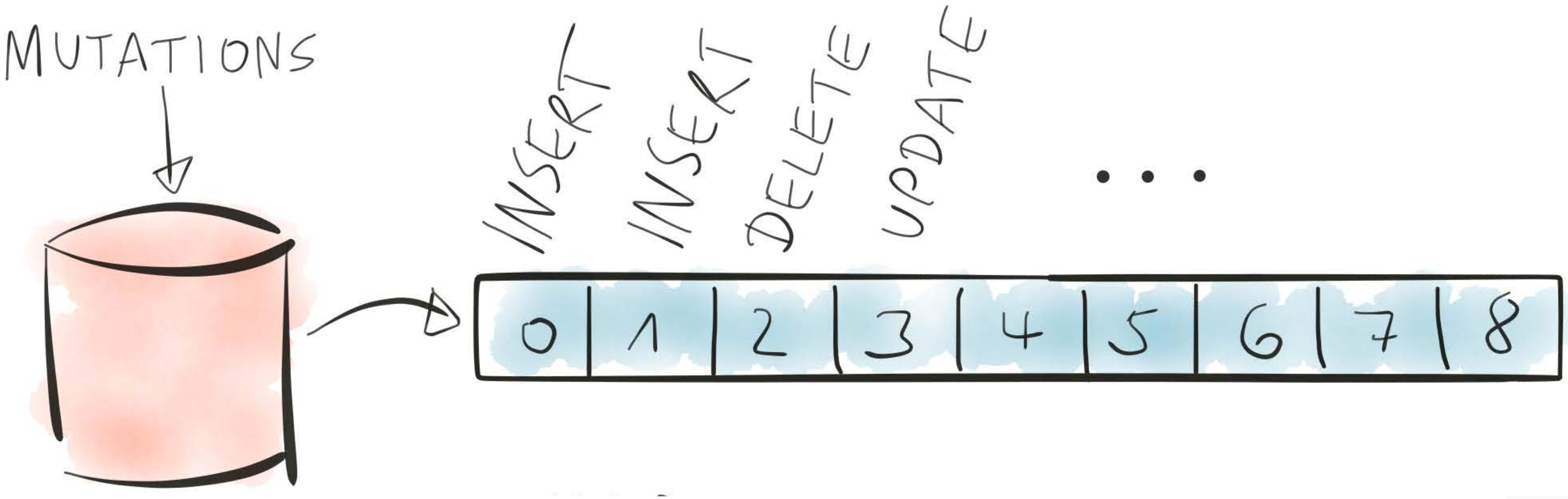
are

EVERYWHERE

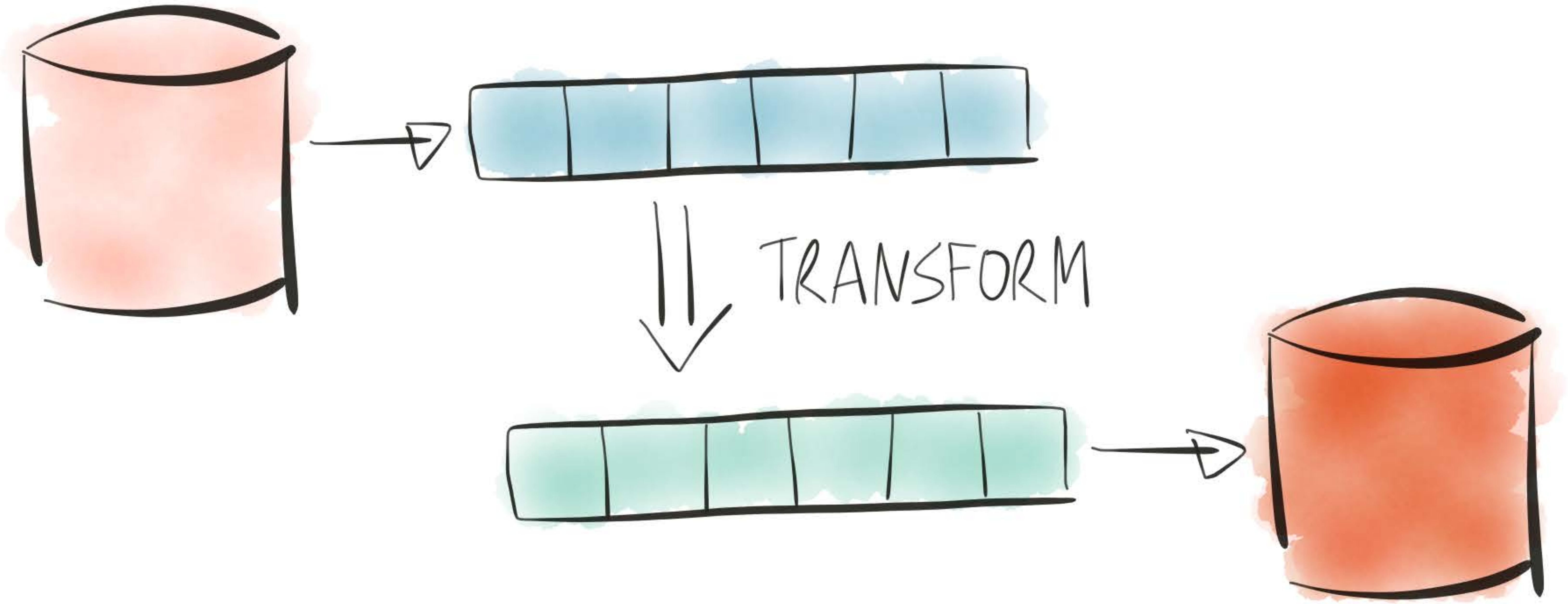
Streams to Tables



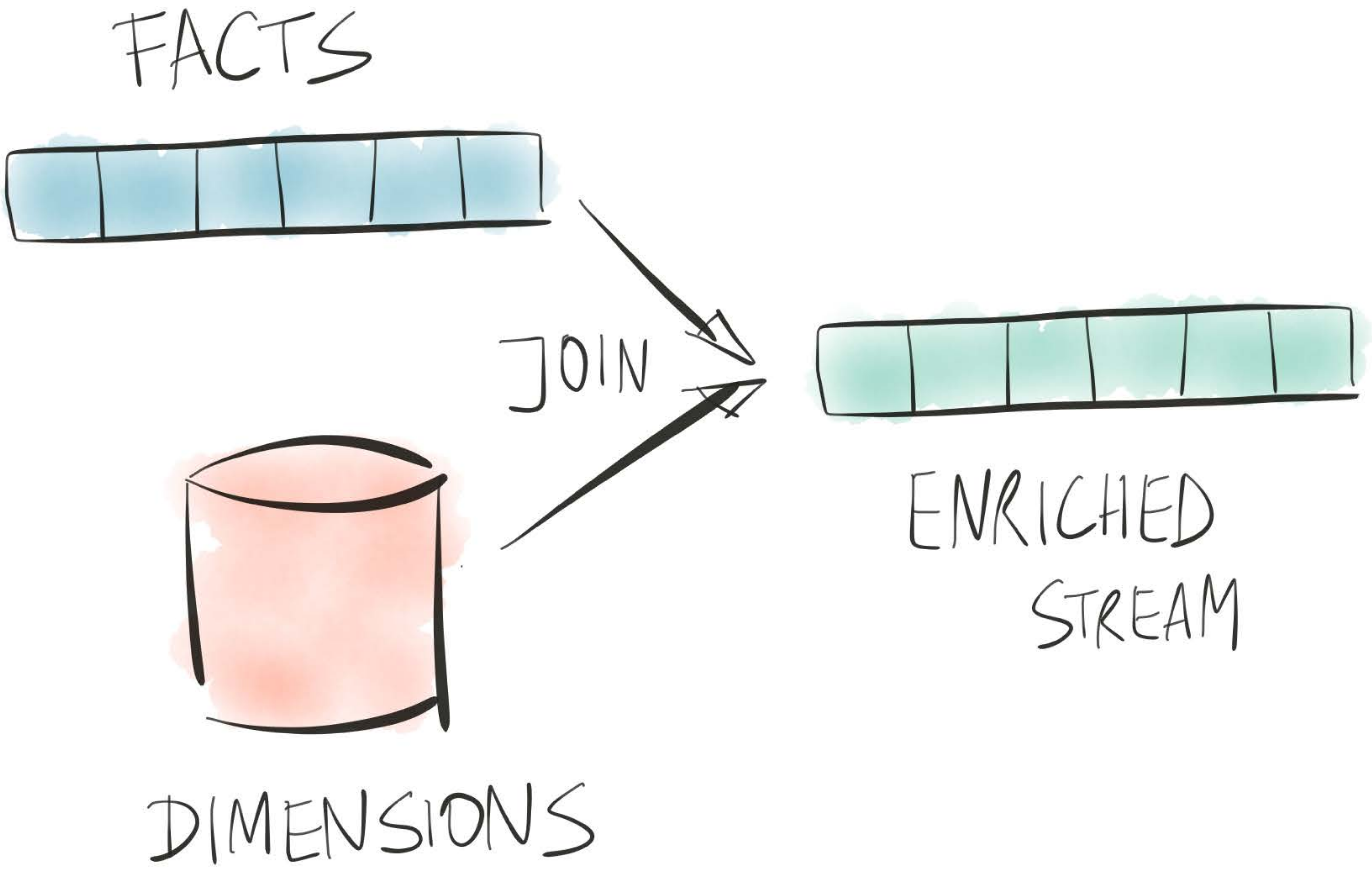
Tables to Streams



Stream/Table Duality



Stream/Table Duality



KStream

```
KStream<Long, String> rawRatings = builder.stream(Serdes.Long(),
                                                  Serdes.String(),
                                                  "raw-ratings");

KStream<Long, Rating> ratings = rawRatings
         .mapValues(text -> Parser.parseRating(text))
         .map((key, rating) -> new KeyValue<Long, Rating>(rating.getMovieId(), rating));
```

KTable

```
KStream<Long, Float> numericalRatings = ratings.mapValues(rating -> rating.getRating());

KGroupedStream<Long, Float> ratingsByMovieId = numericalRatings.groupByKey();

KTable<Long, Long> ratingCount = ratingsByMovieId.count();
KTable<Long, Float> ratingSum = ratingsByMovieId.reduce((r1, r2) -> r1 + r2);
KTable<Long, Float> ratingAvg = ratingSum.join(ratingCount,
        (sum, count) -> sum.floatValue()/count.floatValue());
```

DEMO

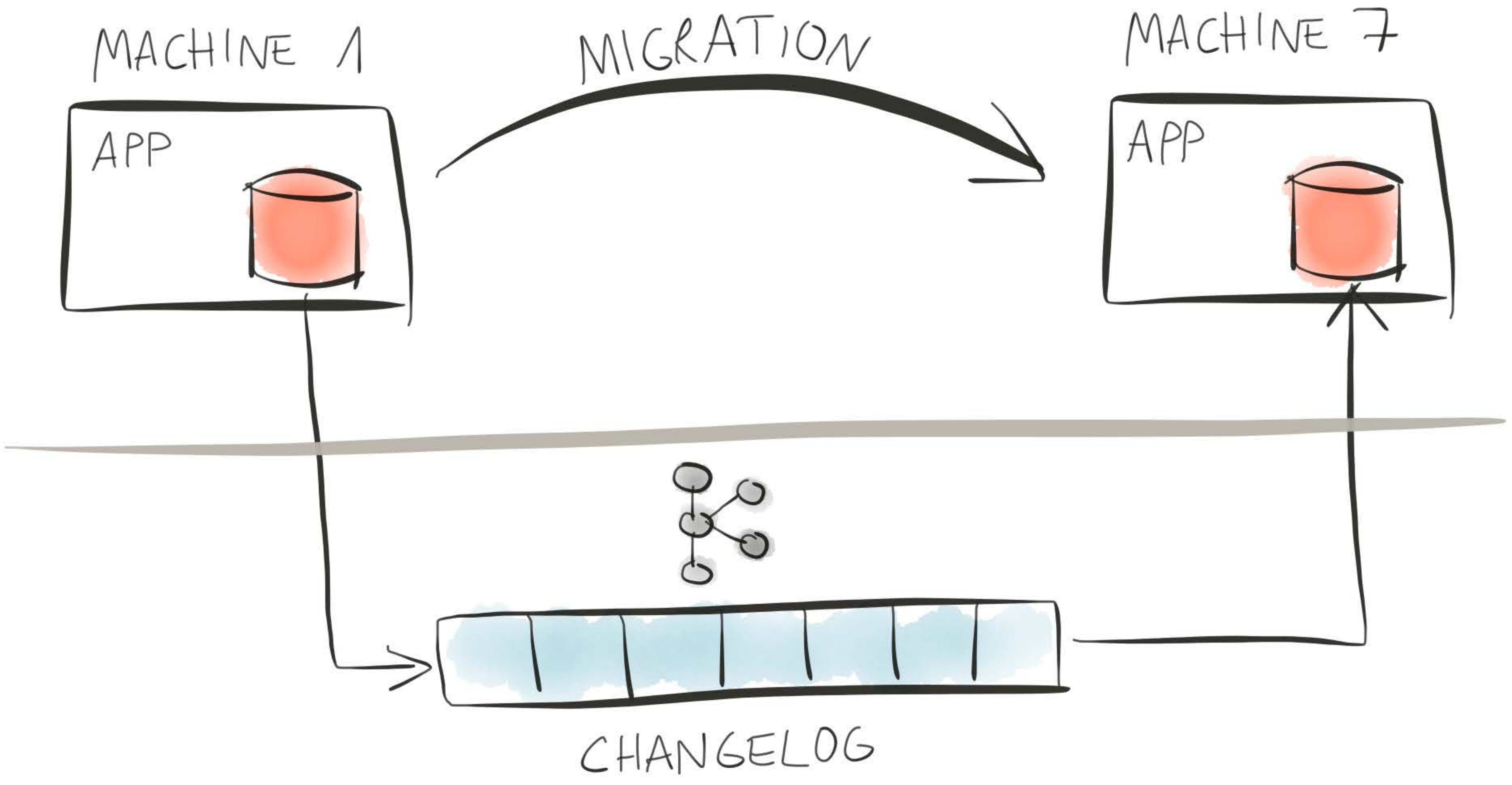
Remember, we want to build

APPS

not

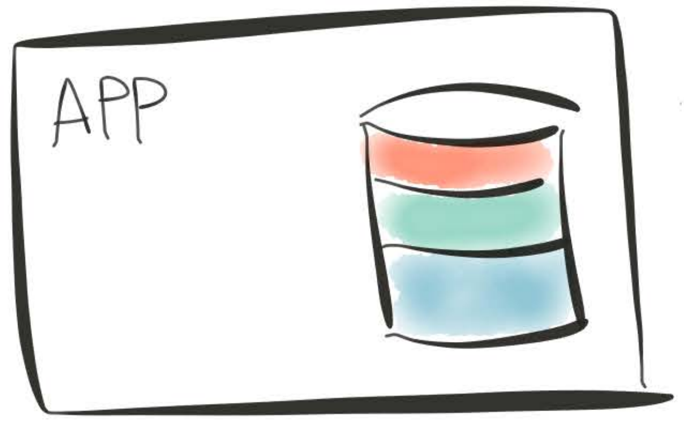
INFRASTRUCTURE

Fault Tolerance

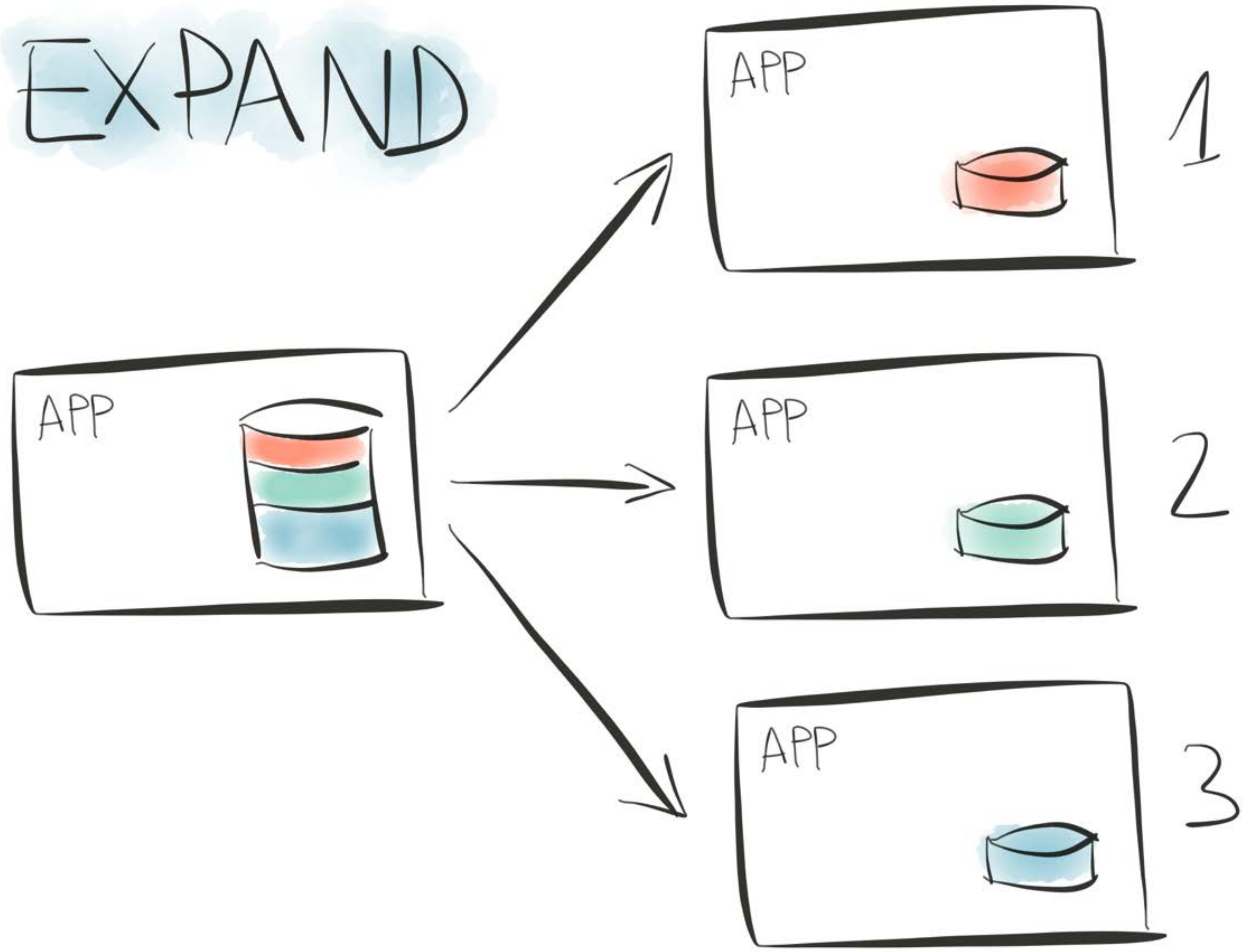


Elasticity

EXPAND



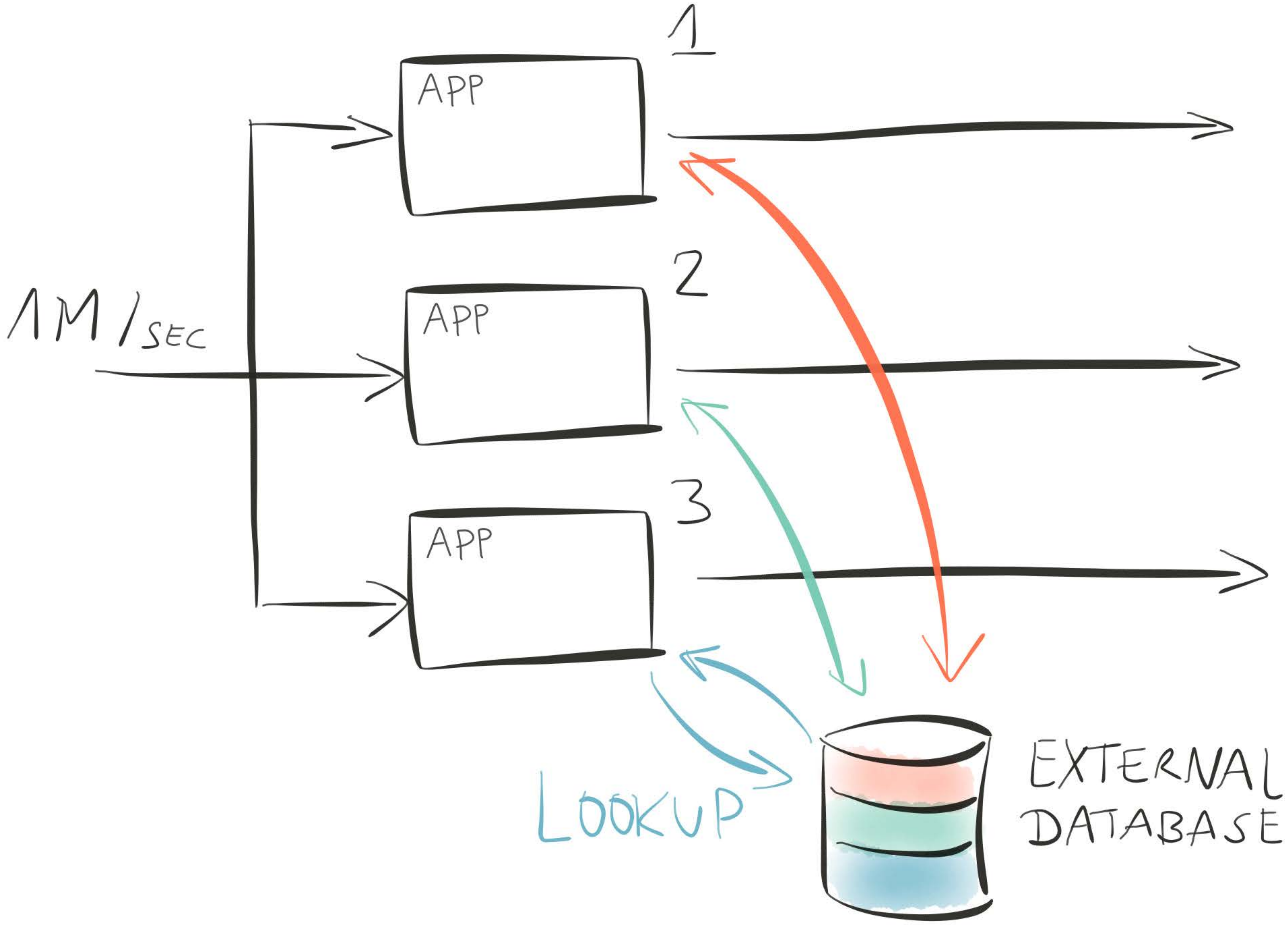
Elasticity



Econimcal at small and large scale

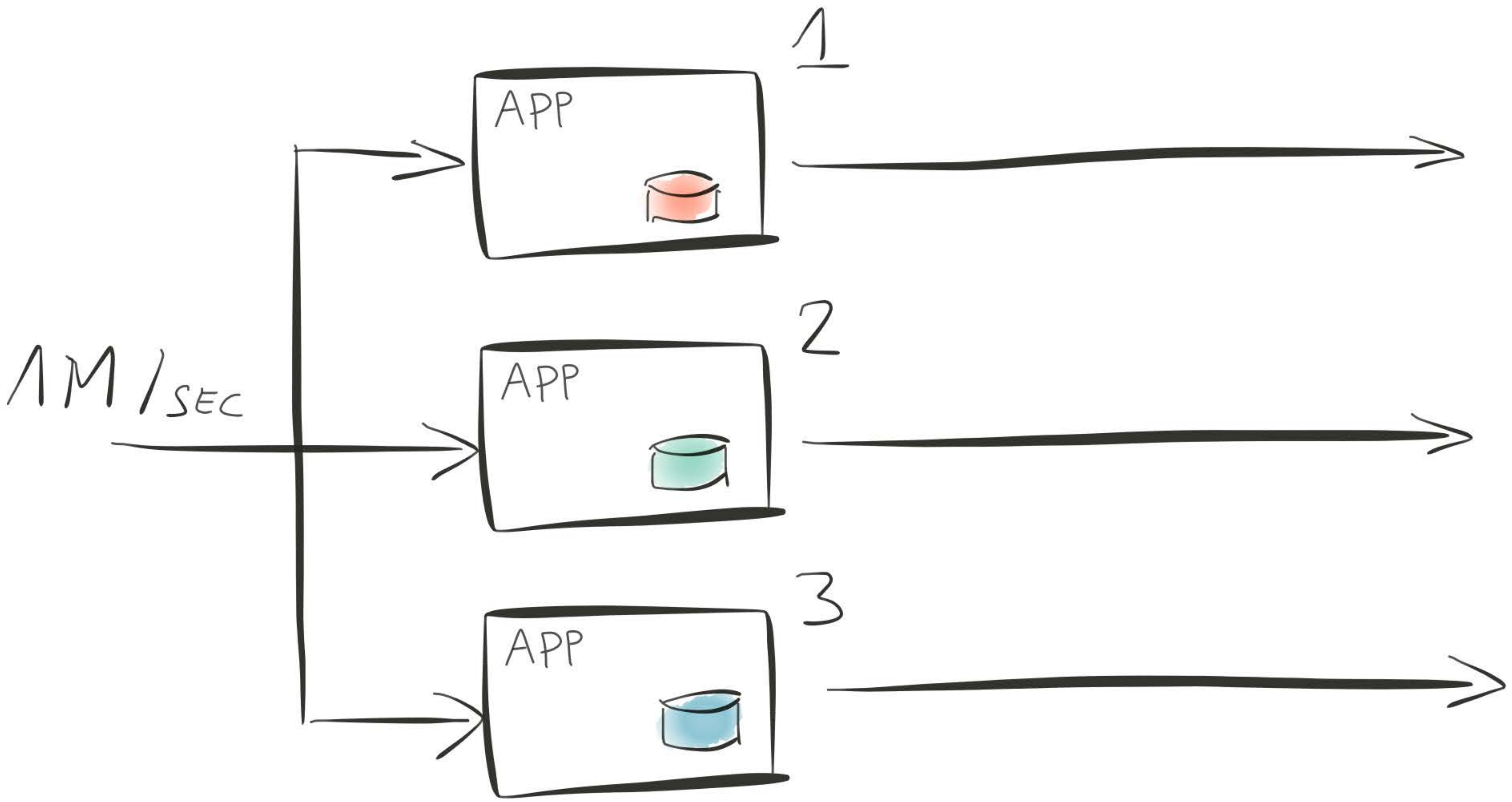
Shared State

Probably failing at life

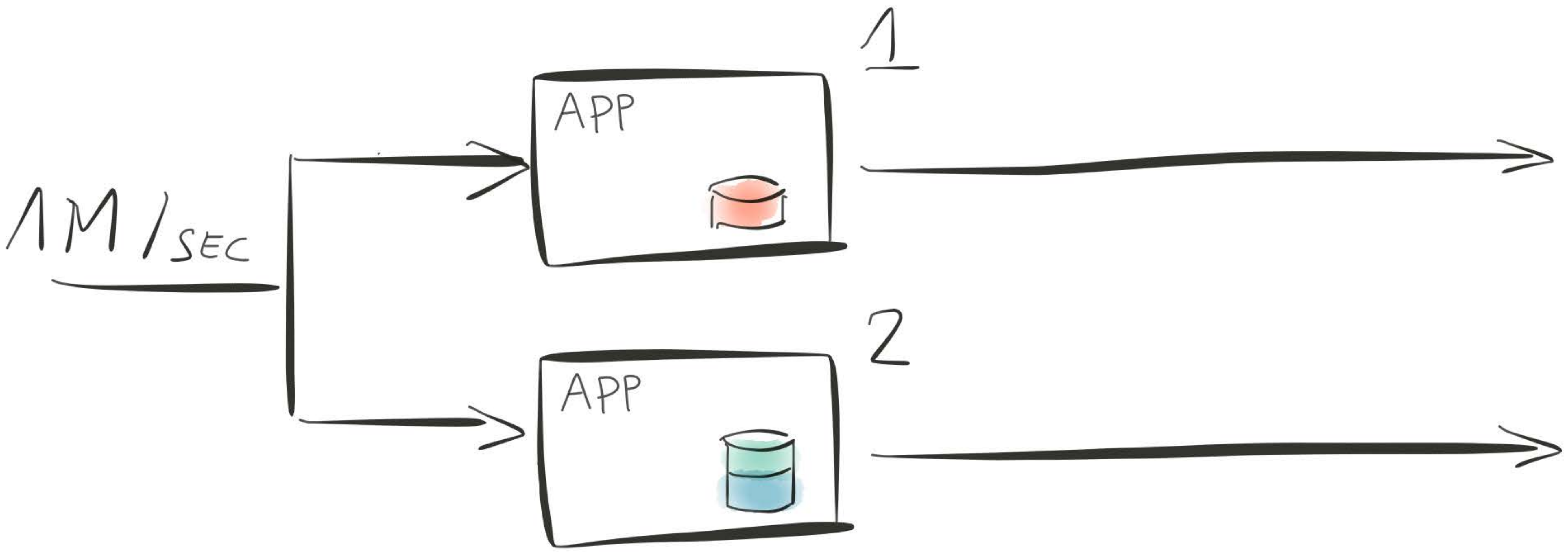


Shared State

Adulation of peers



Shared State



Lower infrastructure costs...

THANK YOU!

confluent

@tlberglund