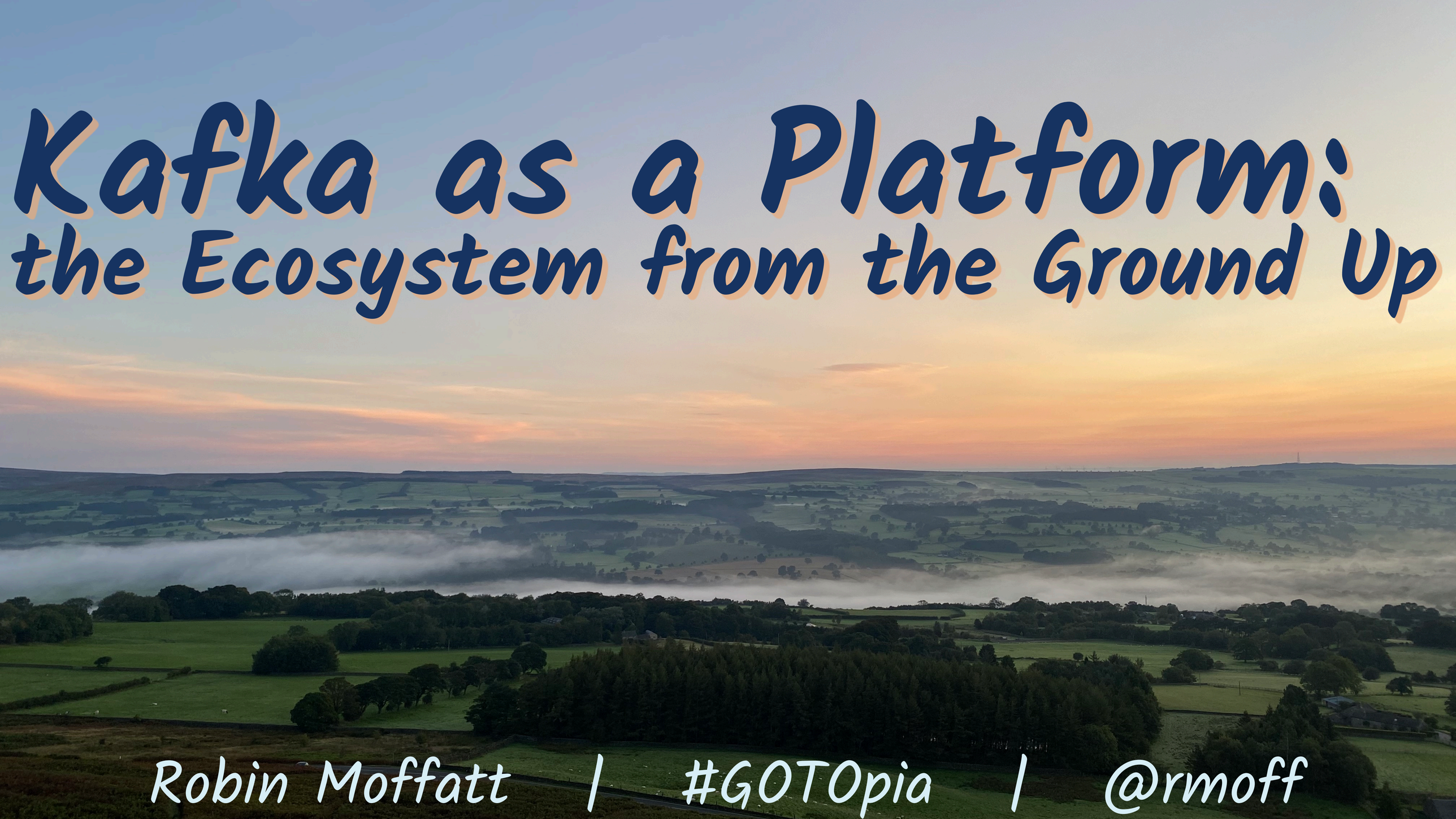
Kafka as a Platform:
the Ecosystem from the Ground Up



Robin Moffatt | #GOTOpia | @rmoff

$ whoami

- Robin Moffatt (@rmoff)
- Senior Developer Advocate at Confluent

  (Apache Kafka, not Wikis 😉)

- Working in data & analytics since 2001
-  Oracle ACE Director (Alumnus)



http://rmoff.dev/**talks** • http://rmoff.dev/**blog** • http://rmoff.dev/**youtube**



EVENTS

@rmoff | #GOTOpia | @confluentinc

EVENTS

EVENTS

- Something happened
- What happened

Human generated events

A Sale



A Stock movement

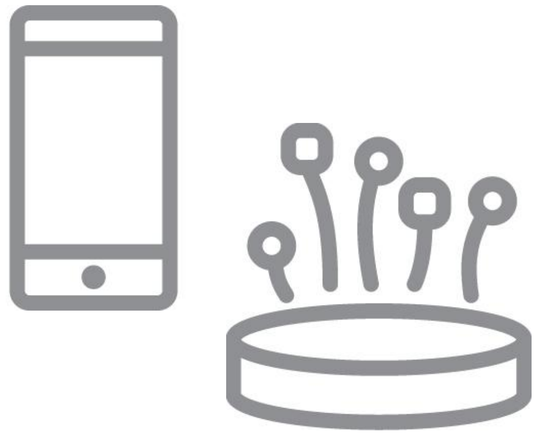


Machine generated events

Networking



IoT



Applications



EVENTS

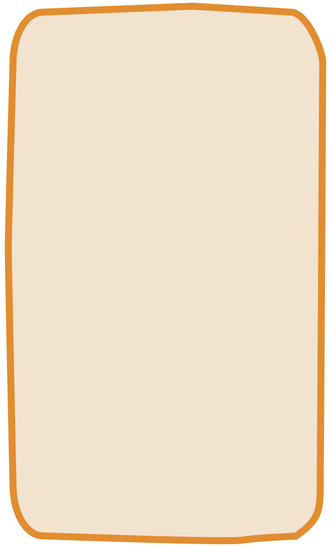
are

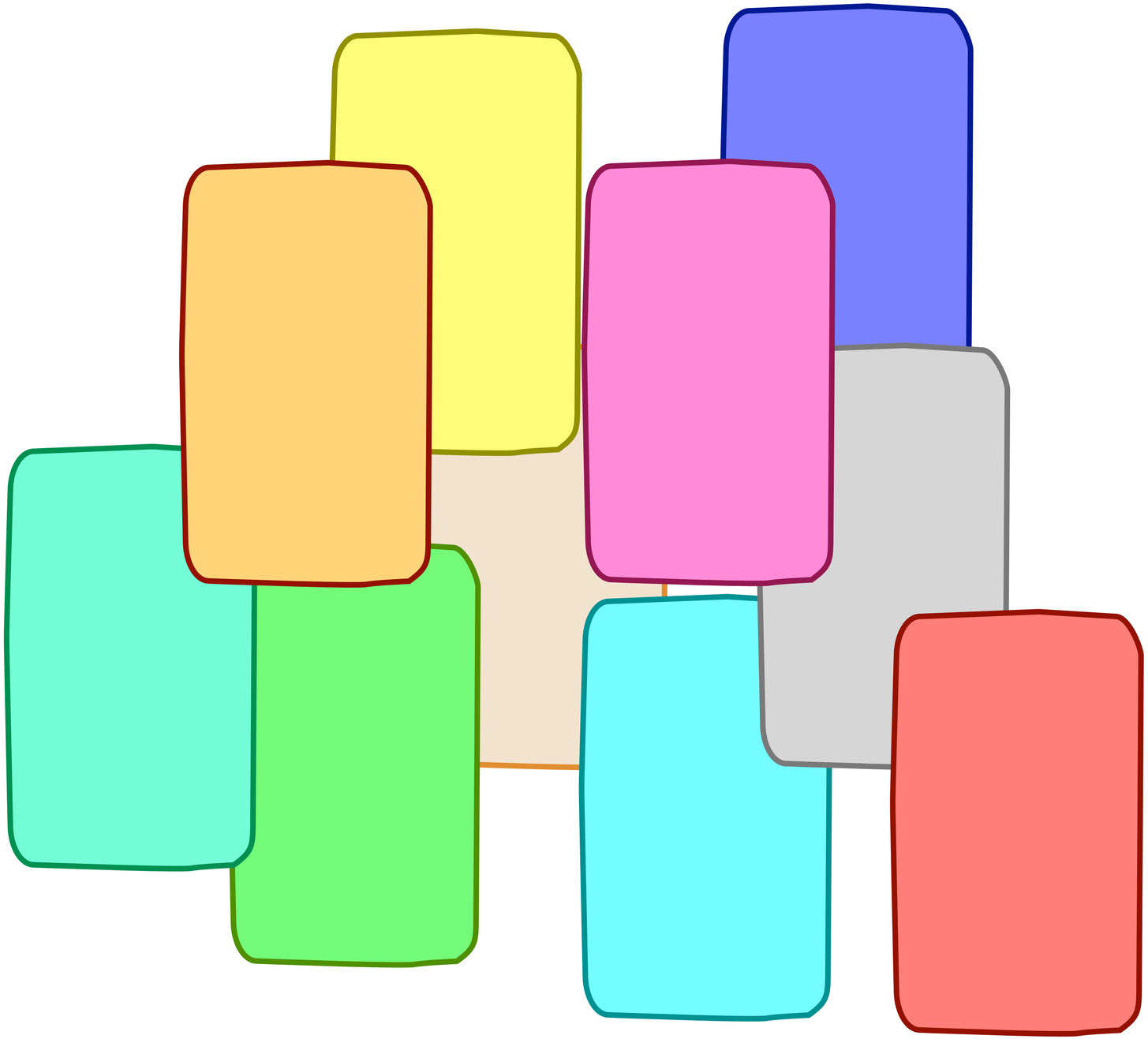
EVERYWHERE

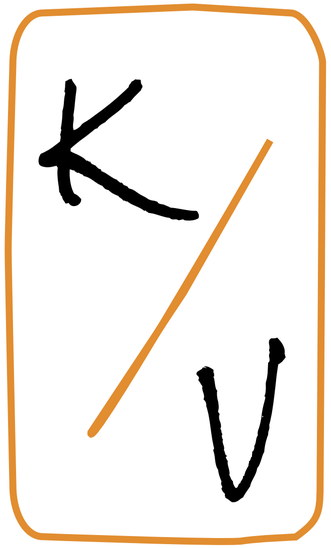
EVENTS

are

very POWERFUL






K
V



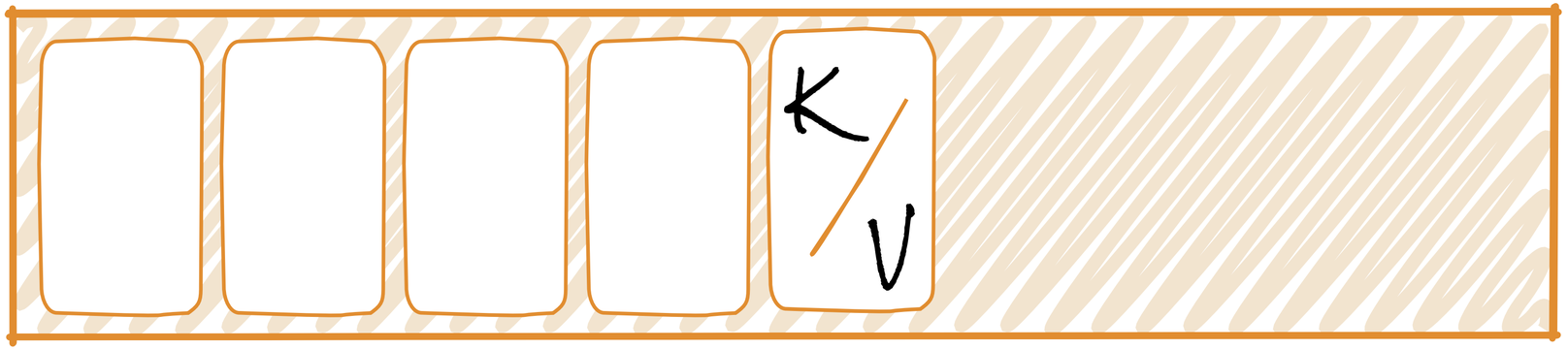
LOG

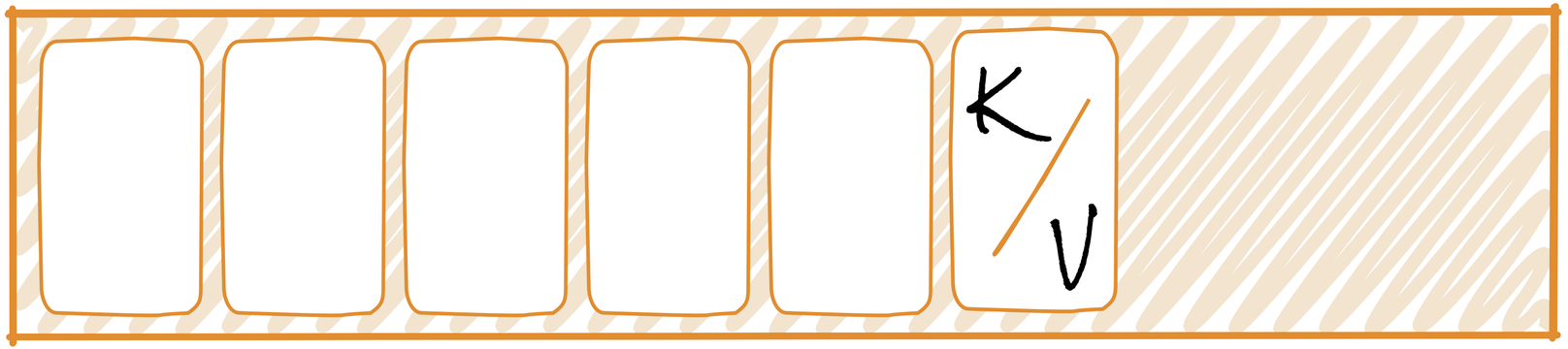

K
V

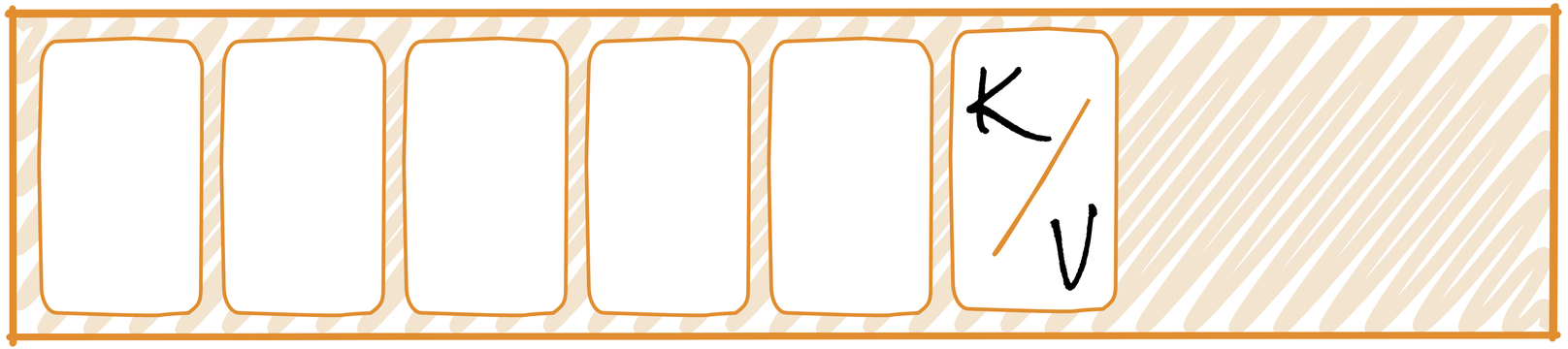

K
V


K
V

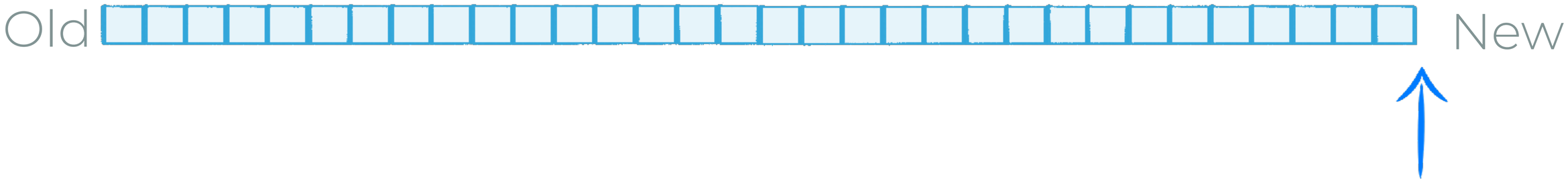

K
V


K
V


K
V


K
V

Immutable Event Log

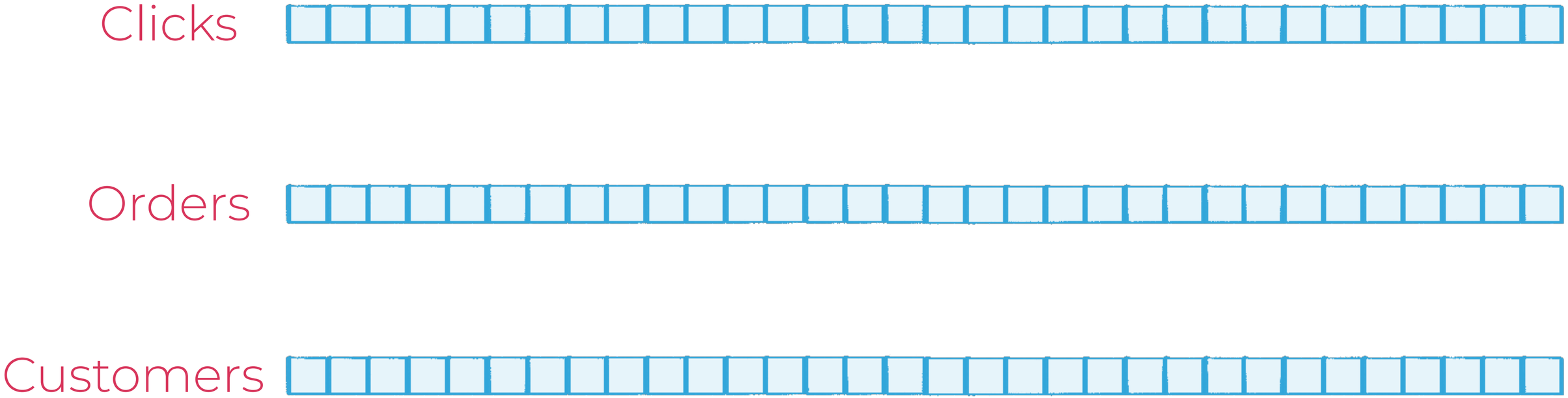


Events are added at the end of the log



TOPICS

Topics

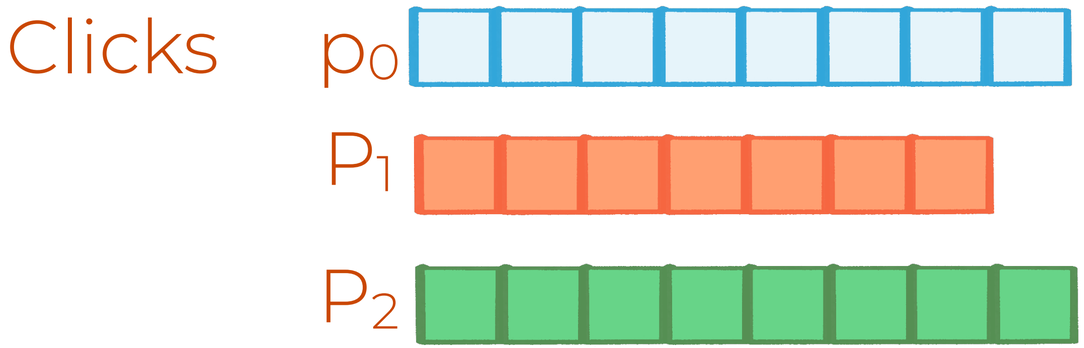


Topics are similar in concept to tables in a database



PARTITIONS

Partitions



Messages are guaranteed to be strictly ordered within a partition



PUB / SUB

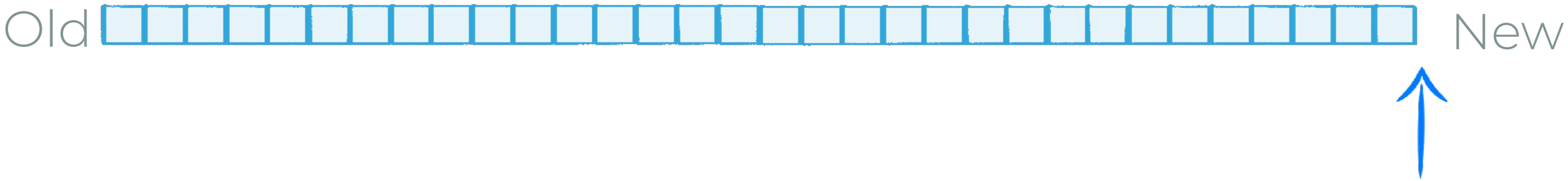
@rmoff | #GOTOpia | @confluentinc



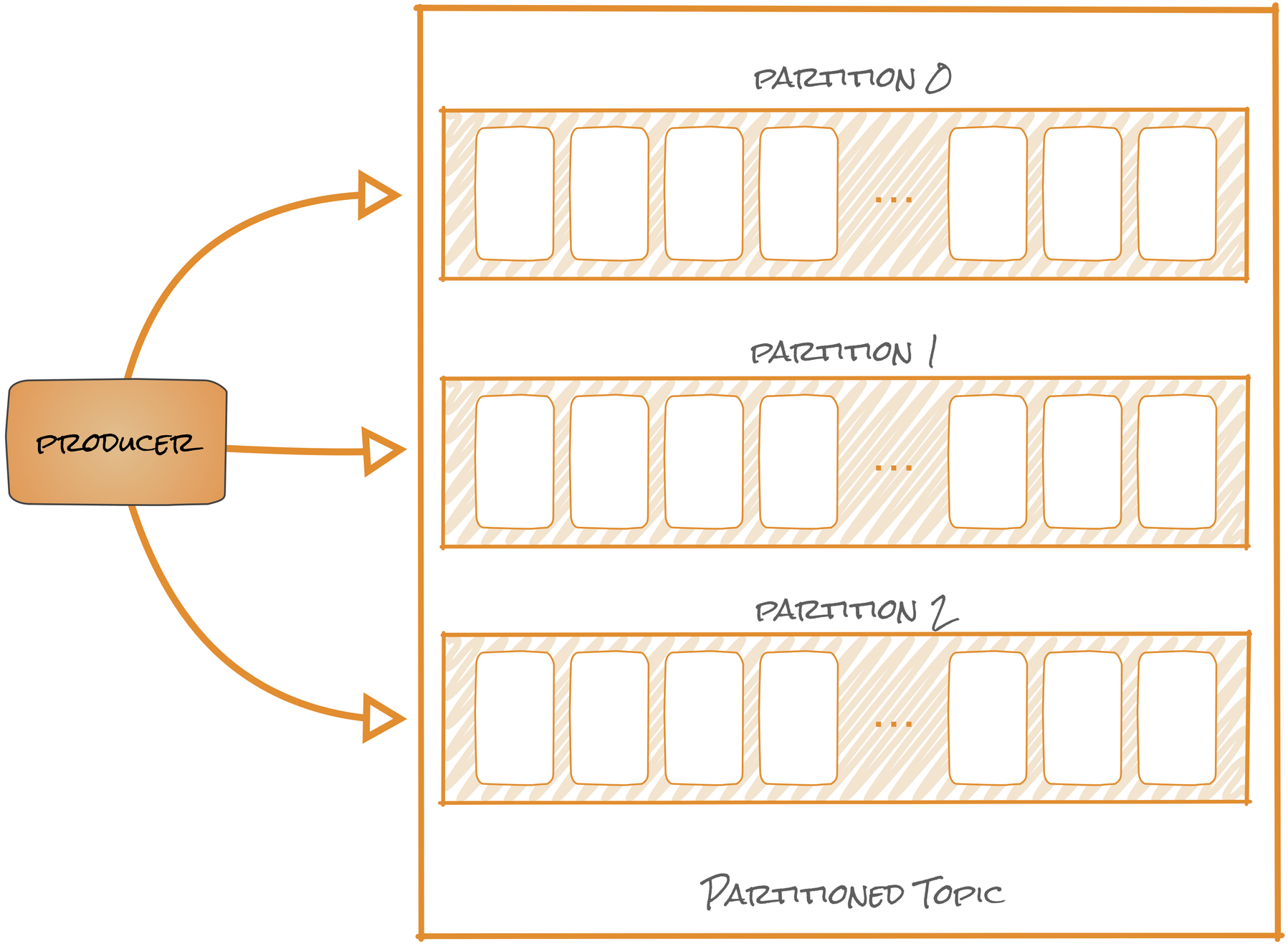
PUB / SUB

@rmoff | #GOTOpia | @confluentinc

Producing data



Messages are added at the end of the log


PRODUCER
PARTITION 0
PARTITION 1
PARTITION 2
PARTITIONED TOPIC

```
try (KafkaProducer<String, Payment> producer = new KafkaProducer<String, Payment>(props)) {

    for (long i = 0; i < 10; i++) {
        final String orderId = "id" + Long.toString(i);
        final Payment payment = new Payment(orderId, 1000.00d);
        final ProducerRecord<String, Payment> record =
           new ProducerRecord<String, Payment>("transactions",
                                               payment.getId().toString(),
                                               payment);
        producer.send(record);
    }
} catch (final InterruptedException e) {
    e.printStackTrace();
}
```

```go
package main

import (
    "gopkg.in/confluentinc/confluent-kafka-go.v1/kafka"
)

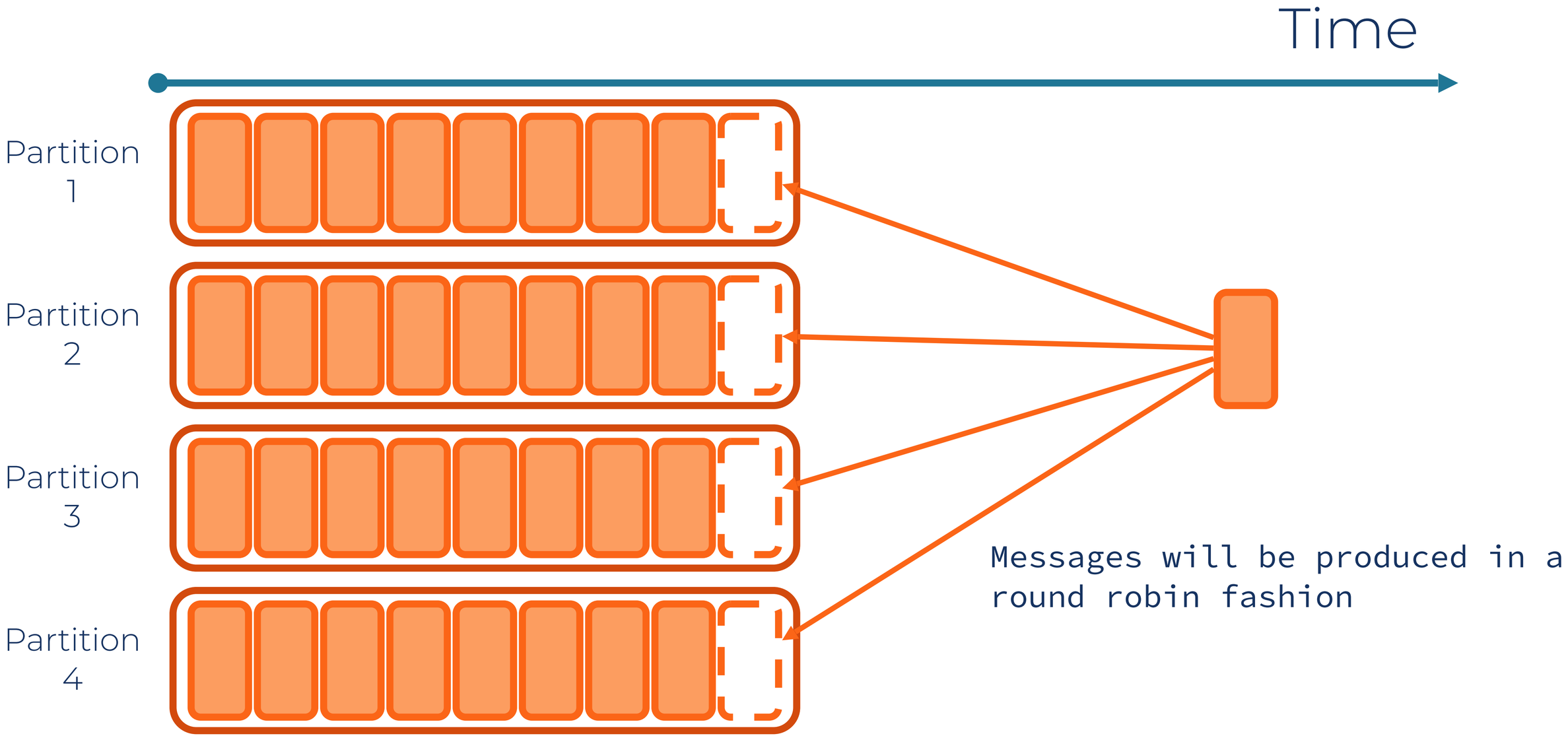
func main() {

    topic := "test_topic"
    p, _ := kafka.NewProducer(&kafka.ConfigMap{
        "bootstrap.servers": "localhost:9092"})
    defer p.Close()

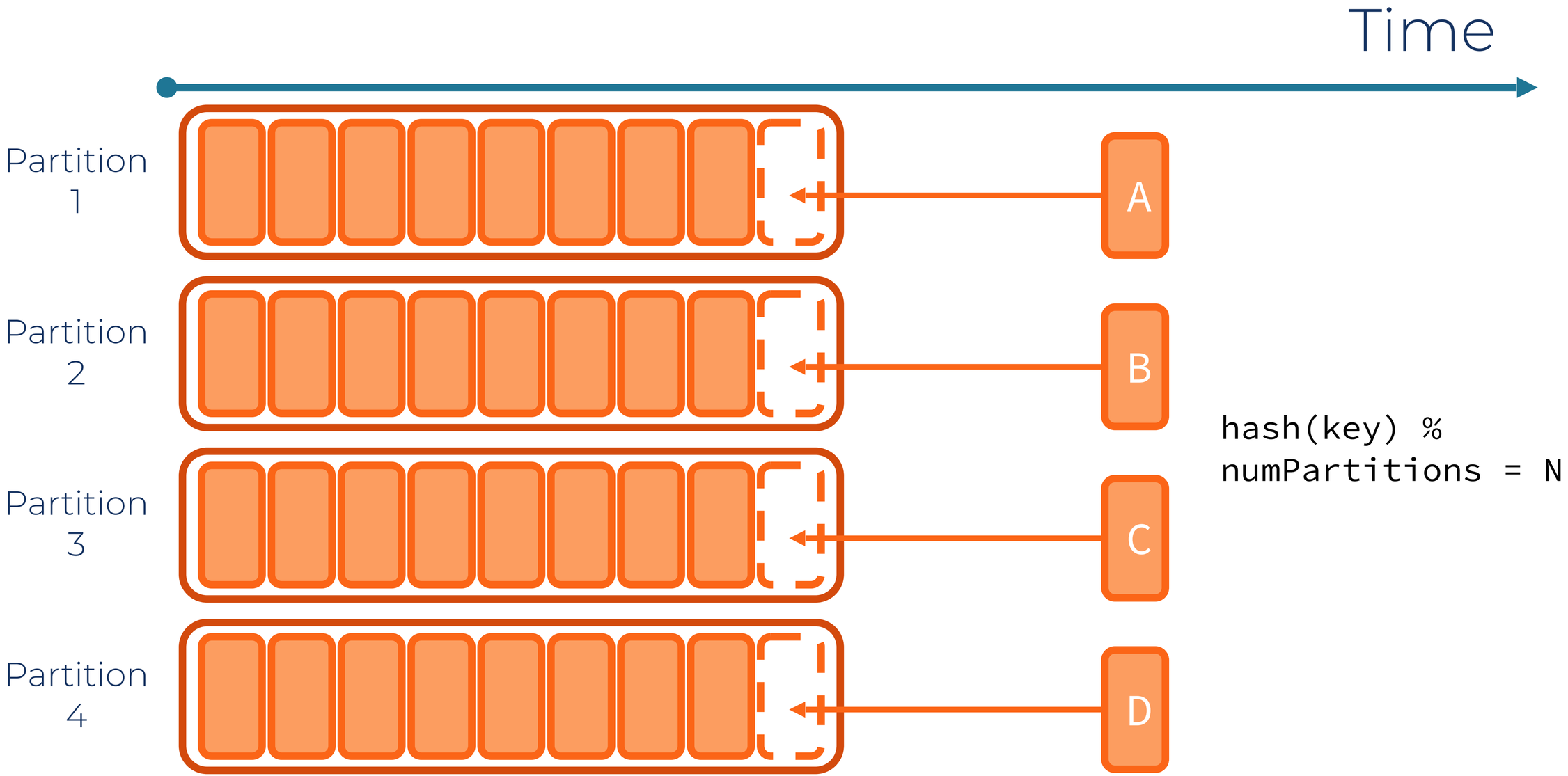
    p.Produce(&kafka.Message{
        TopicPartition: kafka.TopicPartition{Topic: &topic,
            Partition: 0},
        Value: []byte("Hello world")}, nil)

}
```

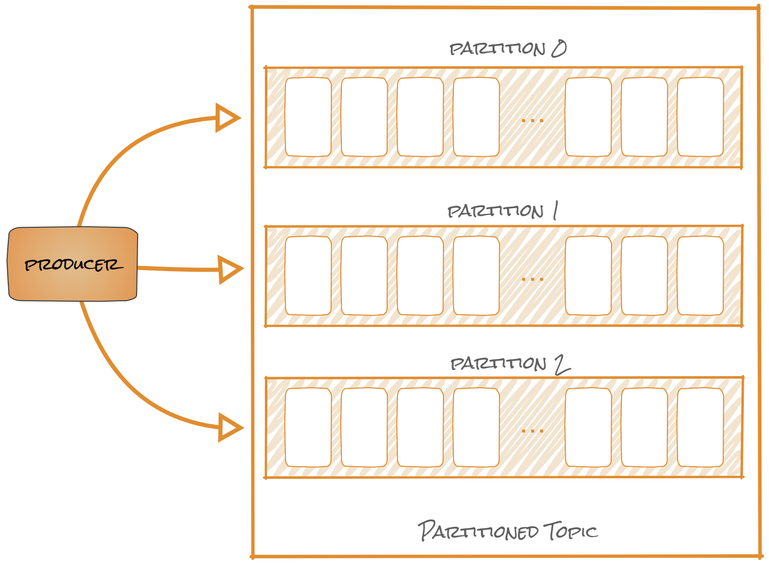
Producing to Kafka - No Key



Producing to Kafka - With Key



Producers

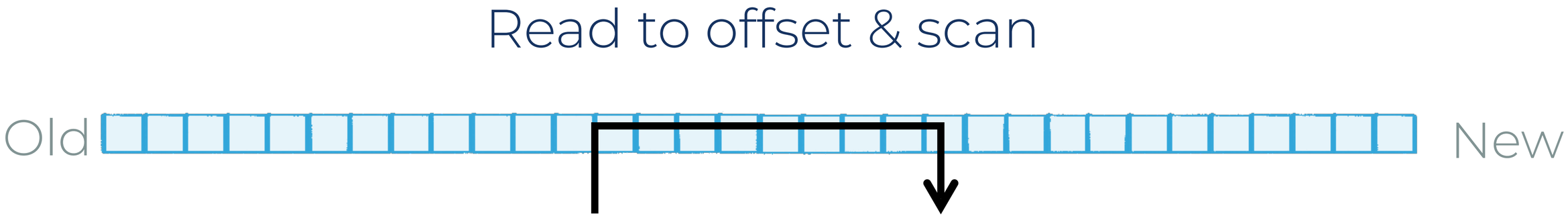


- **A client application**
- **Puts messages into topics**
- **Handles partitioning, network protocol**
- **Java, Go, .NET, C/C++, Python**
- **Also every other language**
  Plus REST proxy if not

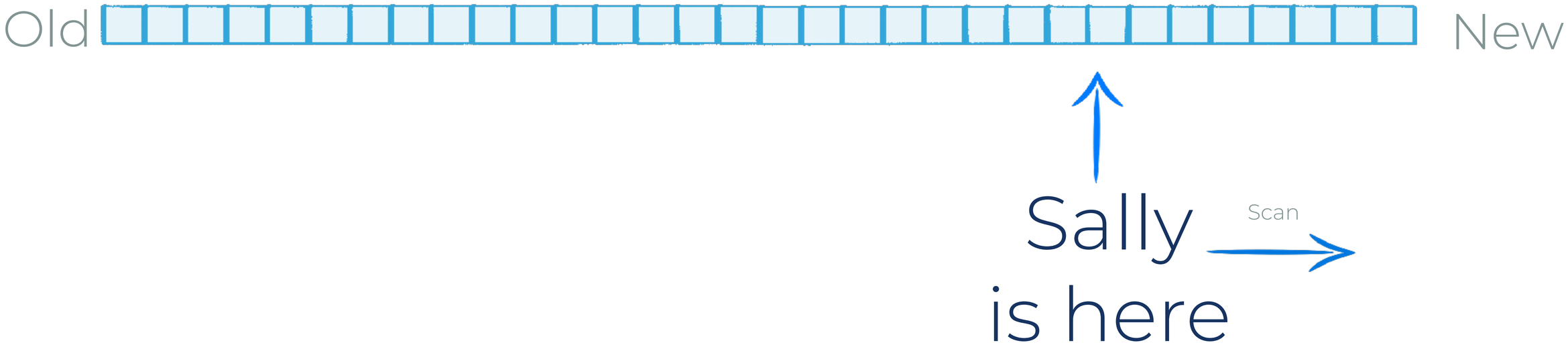


PUB / SUB

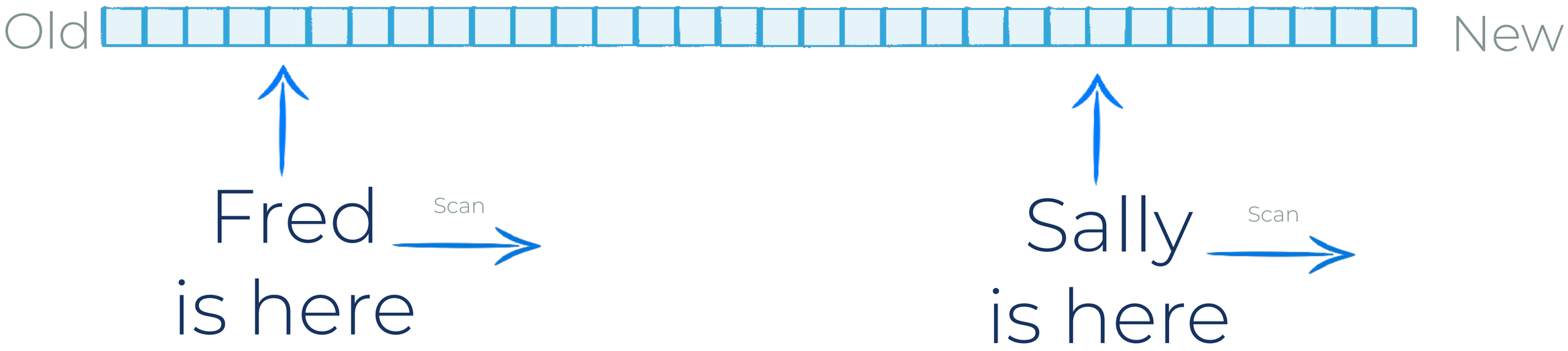
Consuming data - access is only sequential



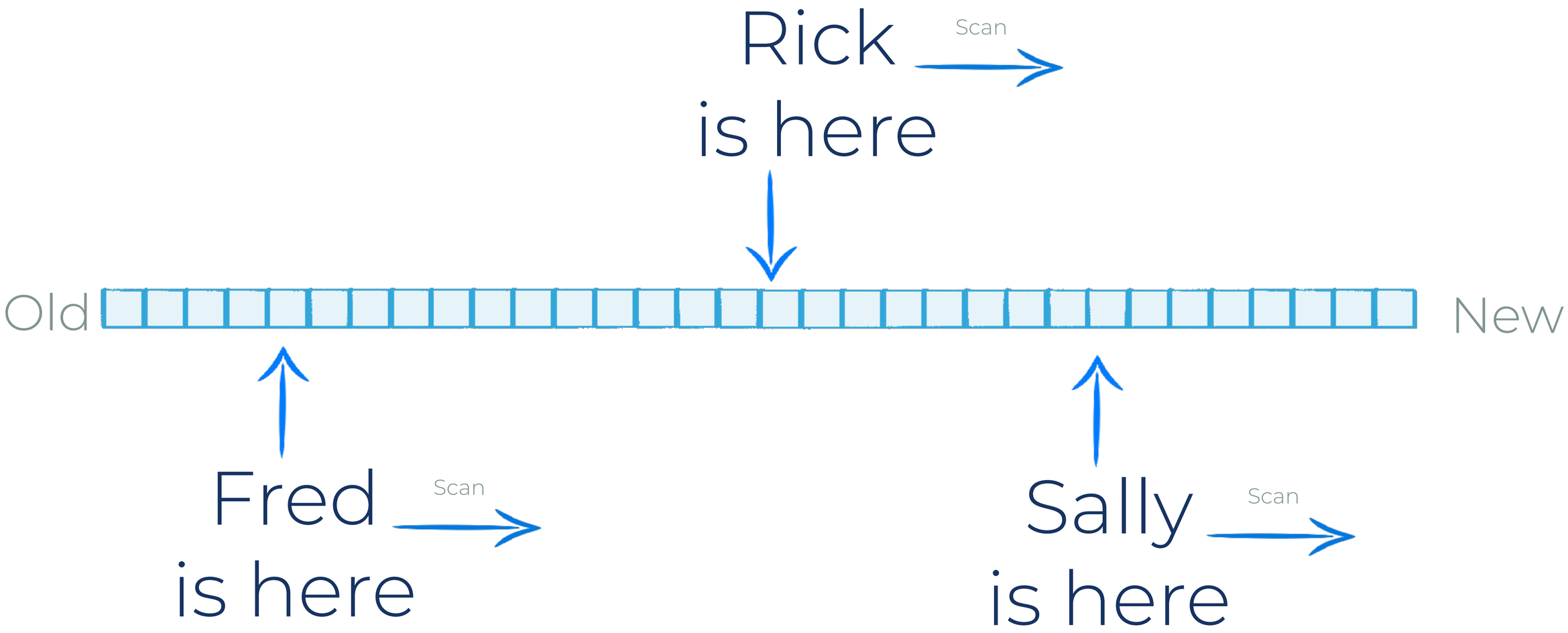
Consumers have a position of their own



Consumers have a position of their own



Consumers have a position of their own



```
try (final KafkaConsumer<String, Payment> consumer = new KafkaConsumer<>(props)) {
    consumer.subscribe(Collections.singletonList(TOPIC));

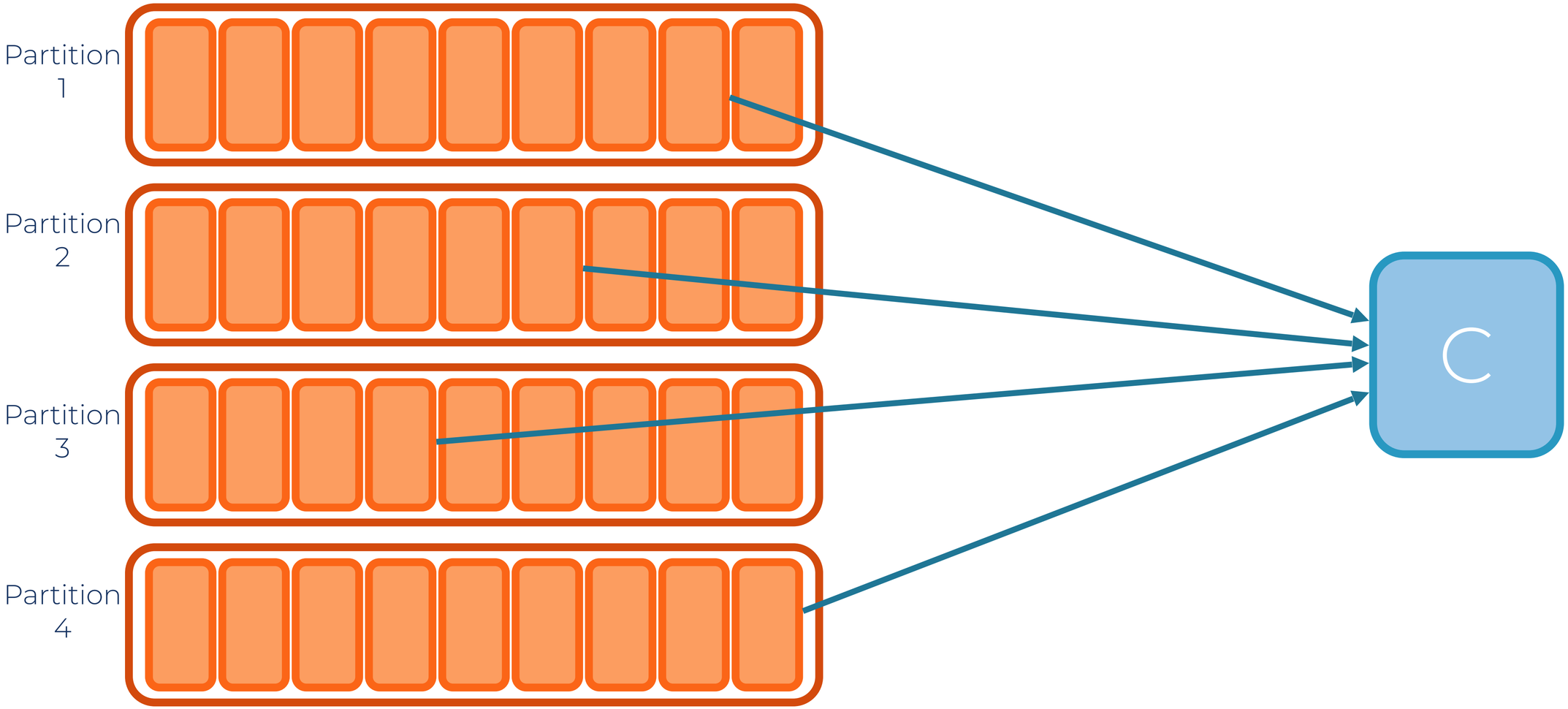
    while (true) {
        ConsumerRecords<String, Payment> records = consumer.poll(100);
        for (ConsumerRecord<String, Payment> record : records) {
            String key = record.key();
            Payment value = record.value();
            System.out.printf("key = %s, value = %s%n", key, value);
        }
    }
}
```

```
c, _ := kafka.NewConsumer(&cm)
defer c.Close()
c.Subscribe(topic, nil)

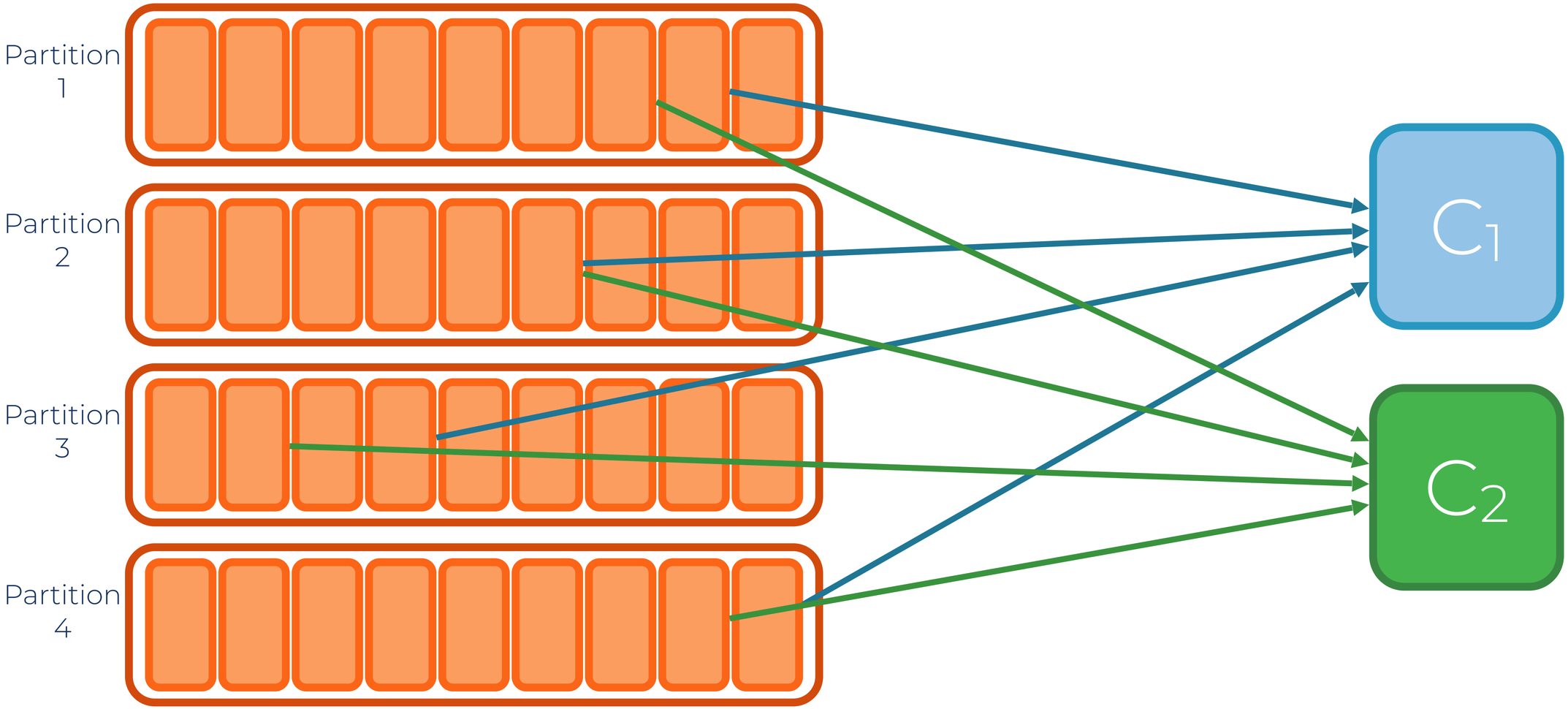
for {
    select {
    case ev := <-c.Events():
        switch ev.(type) {

        case *kafka.Message:
            km := ev.(*kafka.Message)
            fmt.Printf("✅ Message '%v' received from topic
'%v'\n", string(km.Value), string(*km.TopicPartition.Topic))
        }
    }
}
```

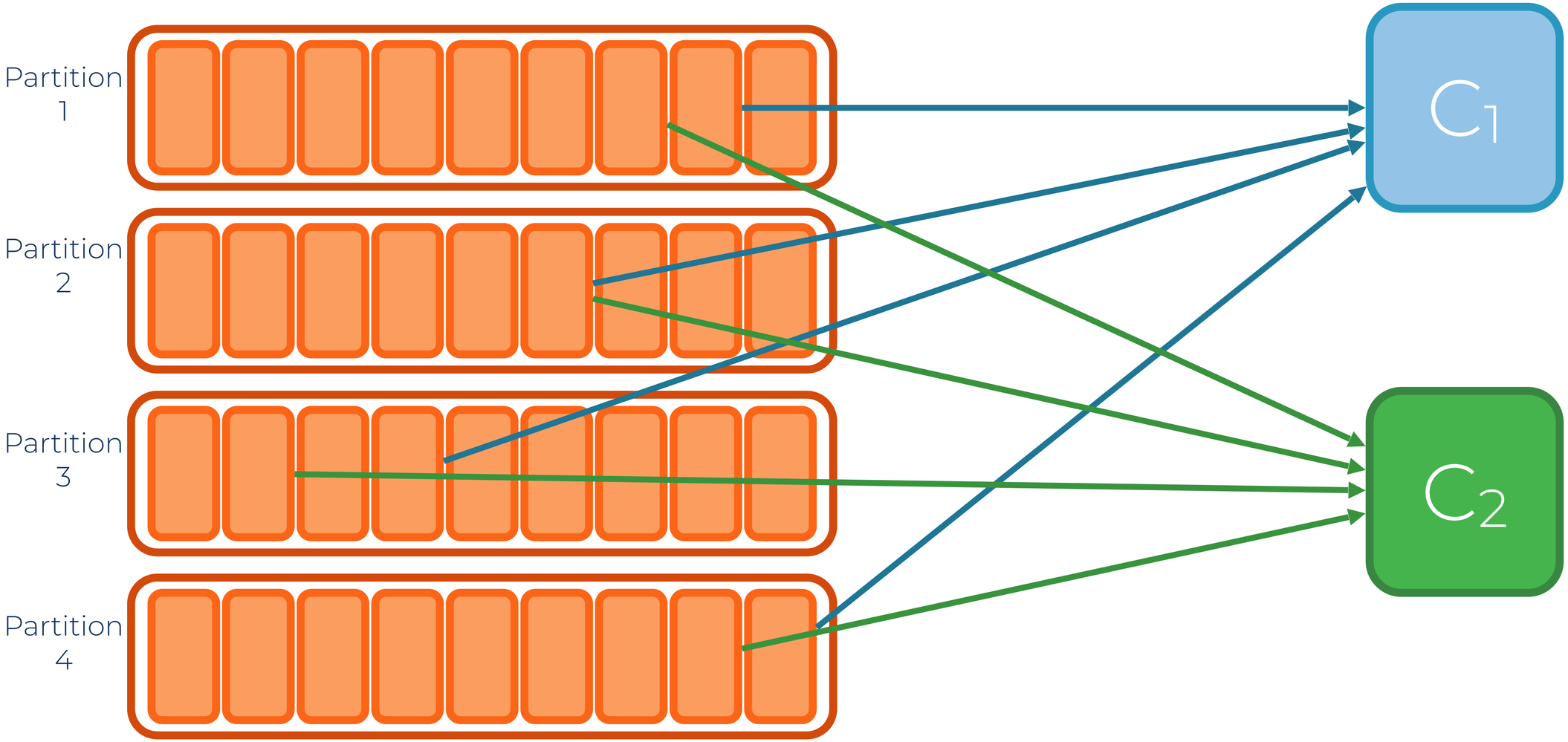
Consuming From Kafka - Single Consumer



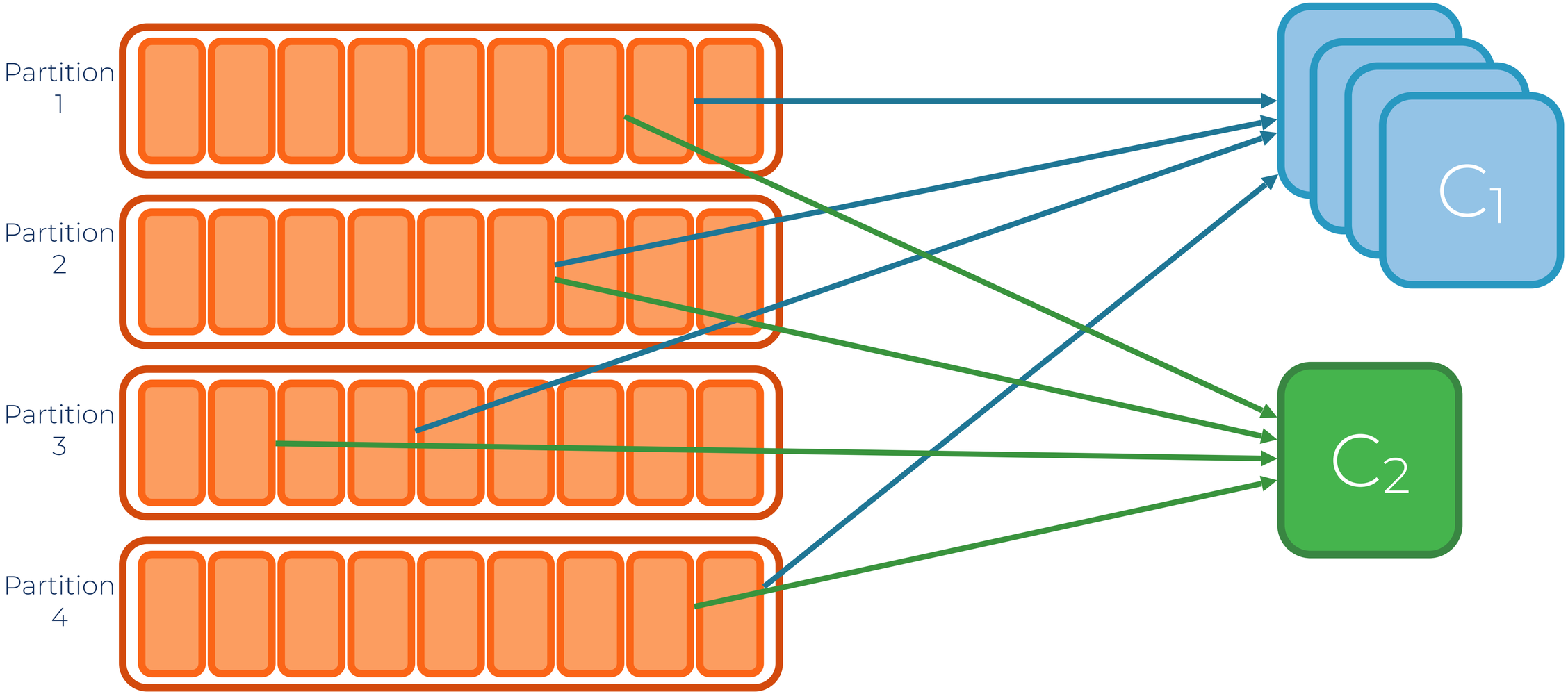
Consuming From Kafka - Multiple Consumers



Consuming From Kafka - Multiple Consumers

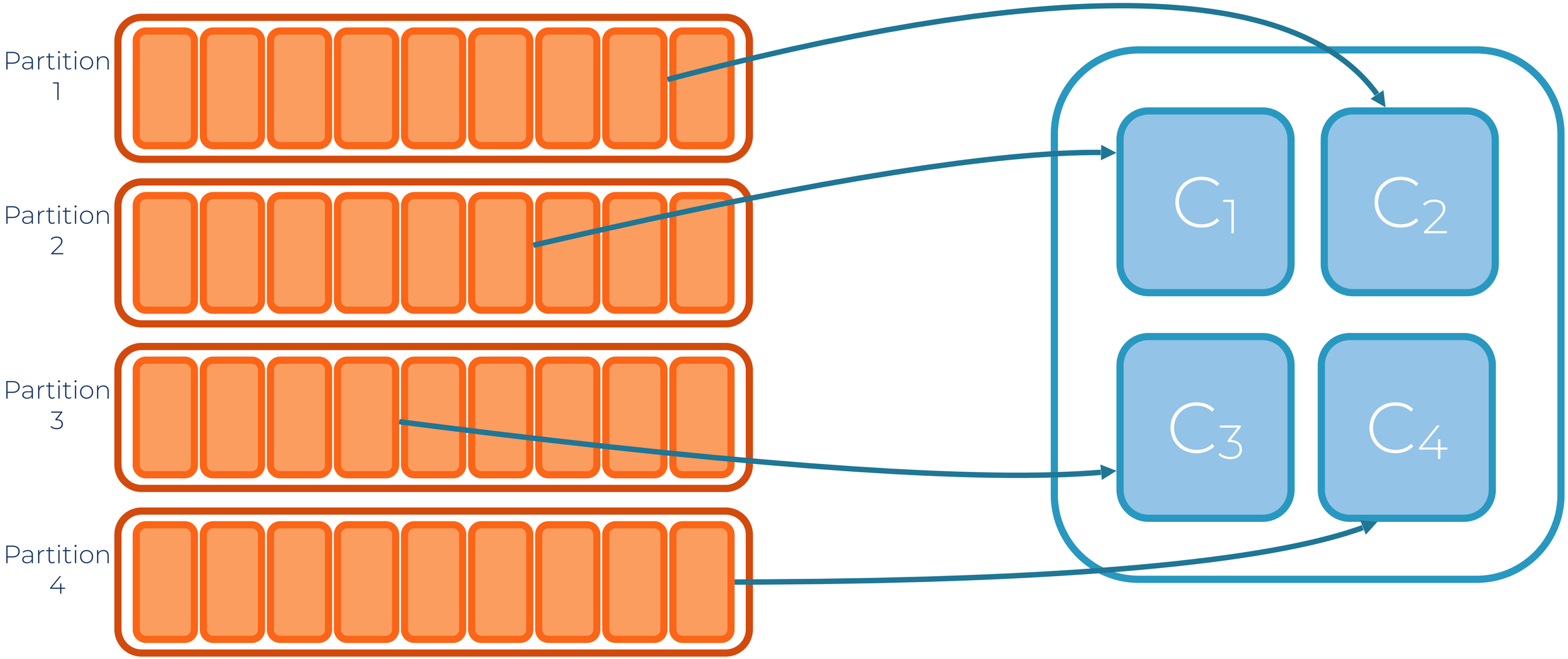


Consuming From Kafka - Grouped Consumers

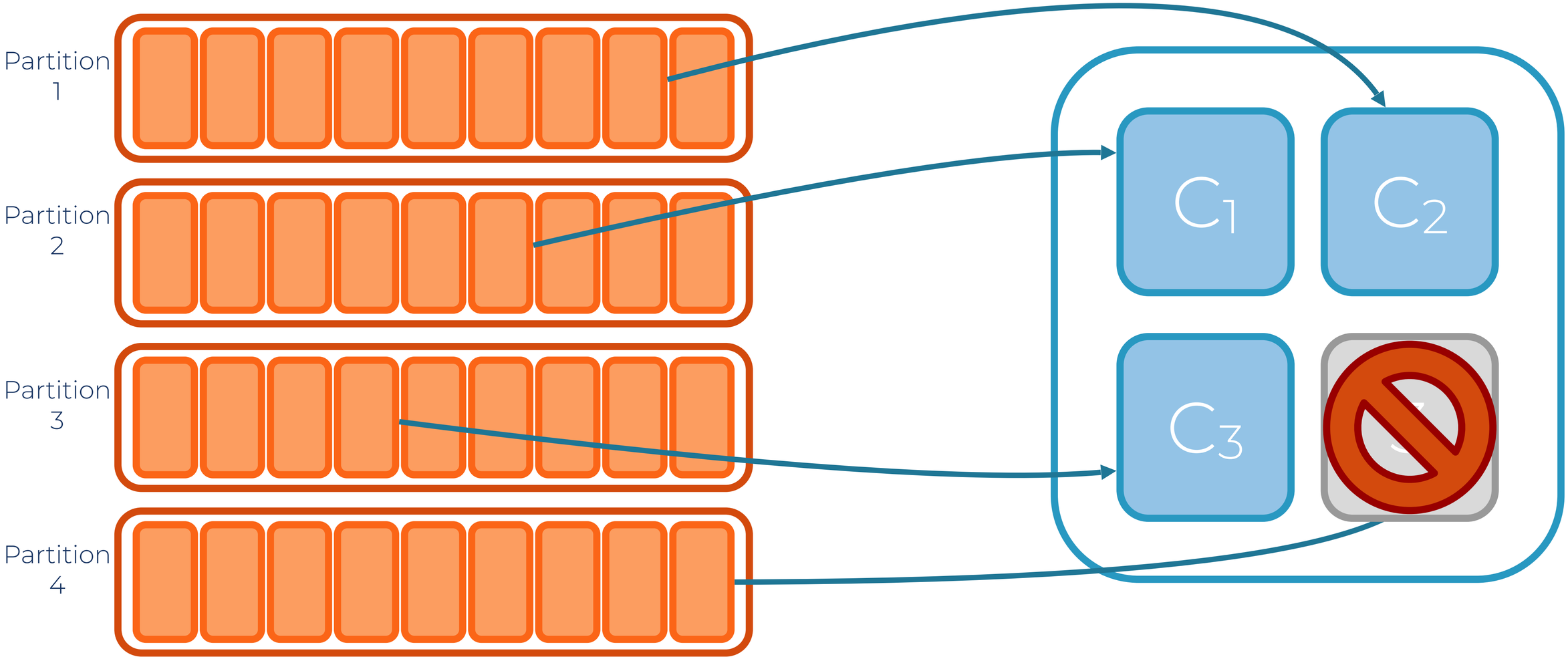



CONSUMER GROUP COORDINATOR
CONSUMERS
CONSUMER GROUP

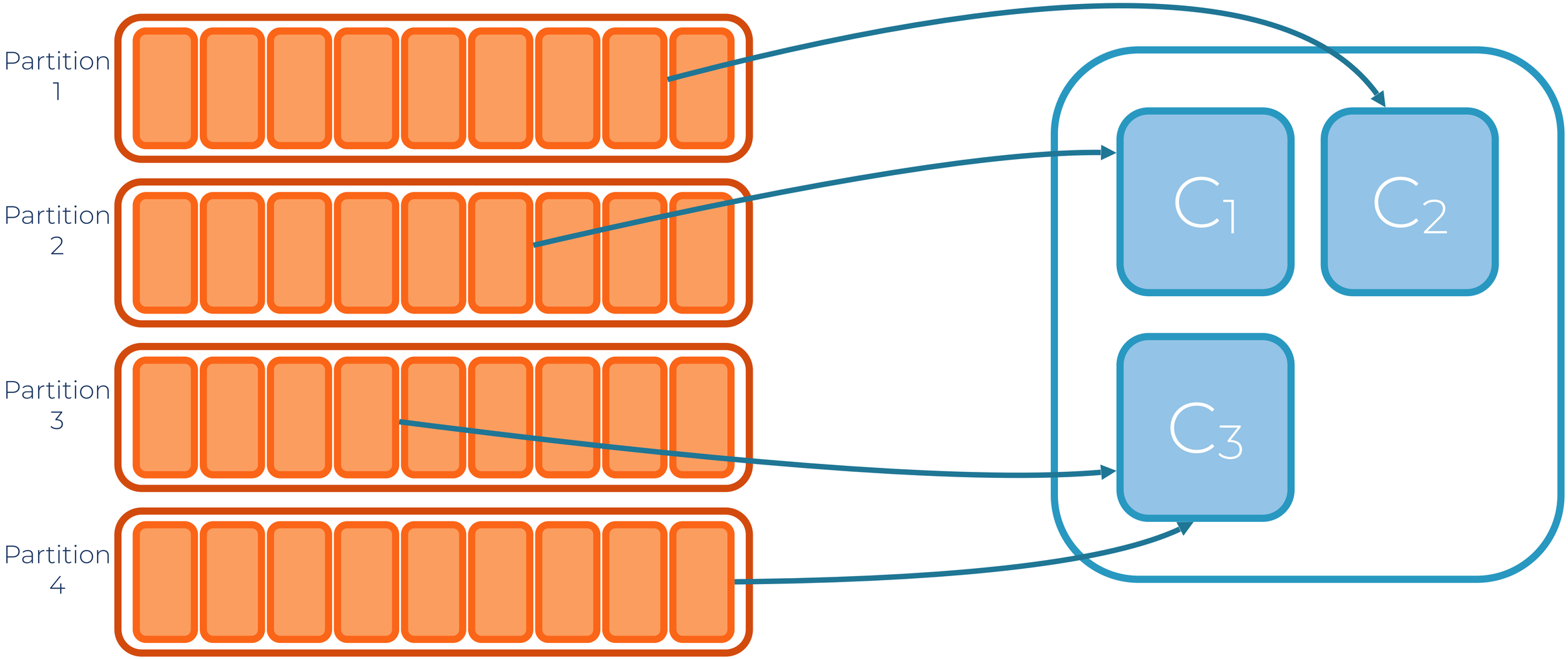
Consuming From Kafka - Grouped Consumers



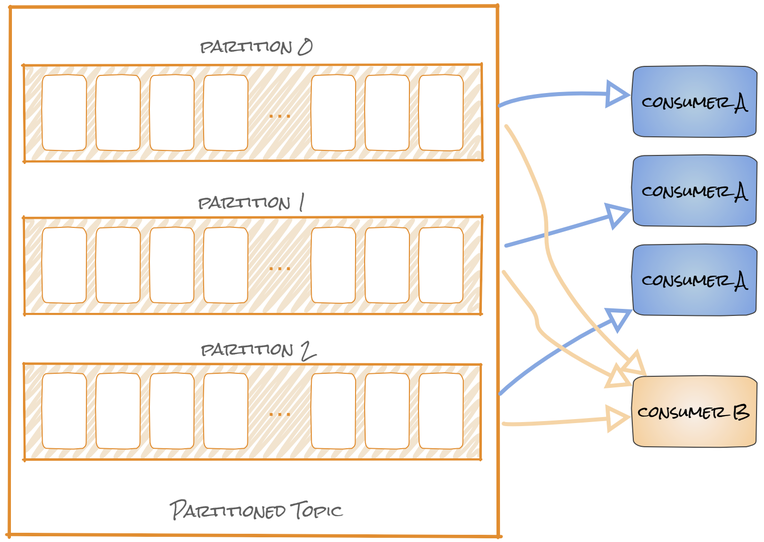
Consuming From Kafka - Grouped Consumers



Consuming From Kafka - Grouped Consumers



Consumers



- **A client application**
- **Reads messages from topics**
- **Horizontally, elastically scalable (if stateless)**
- **Java, Go, .NET, C/C++, Python, everything else**
  Plus REST proxy if not

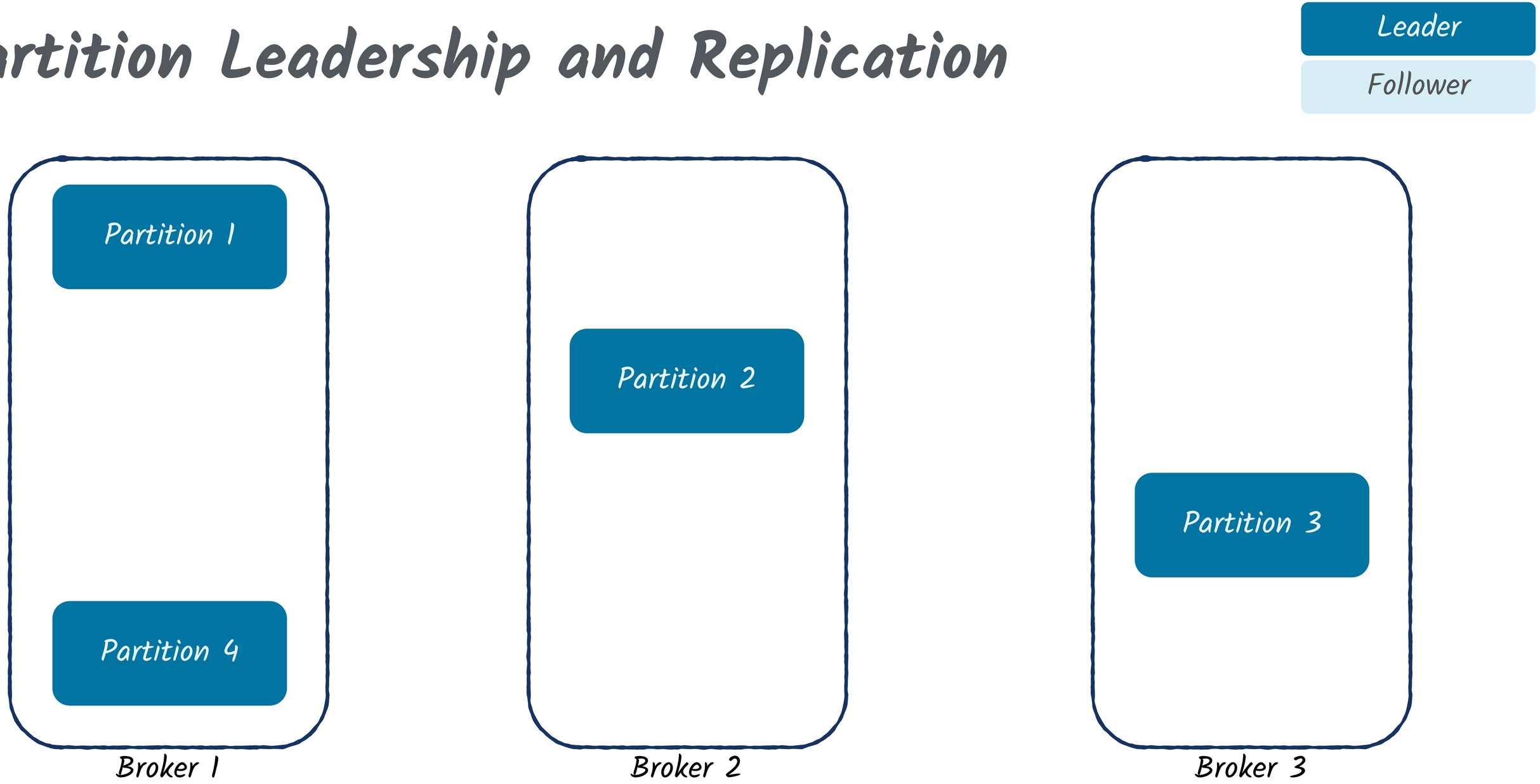


BROKERS

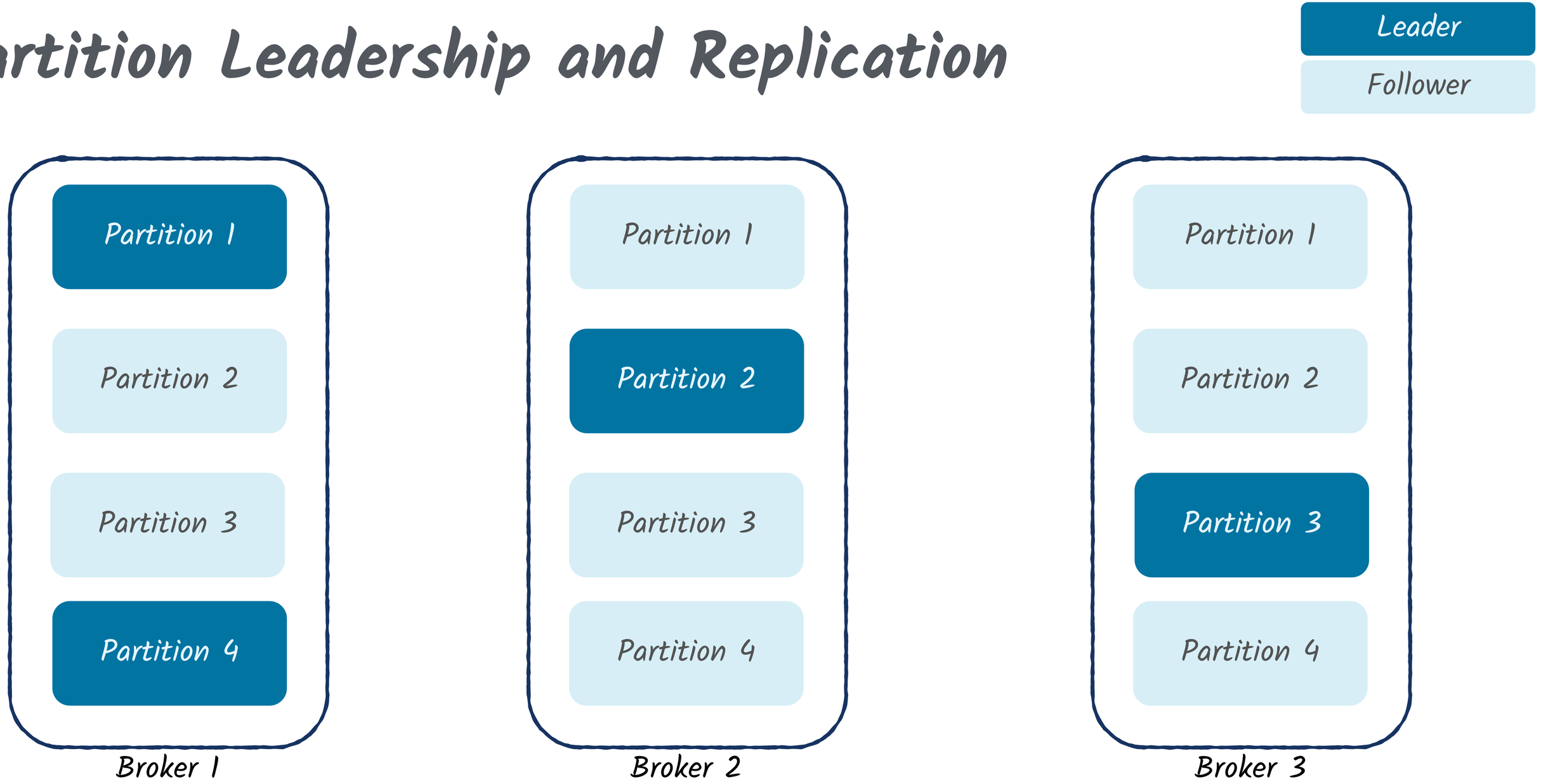
and REPLICATION

@rmoff | #GOTOpia | @confluentinc

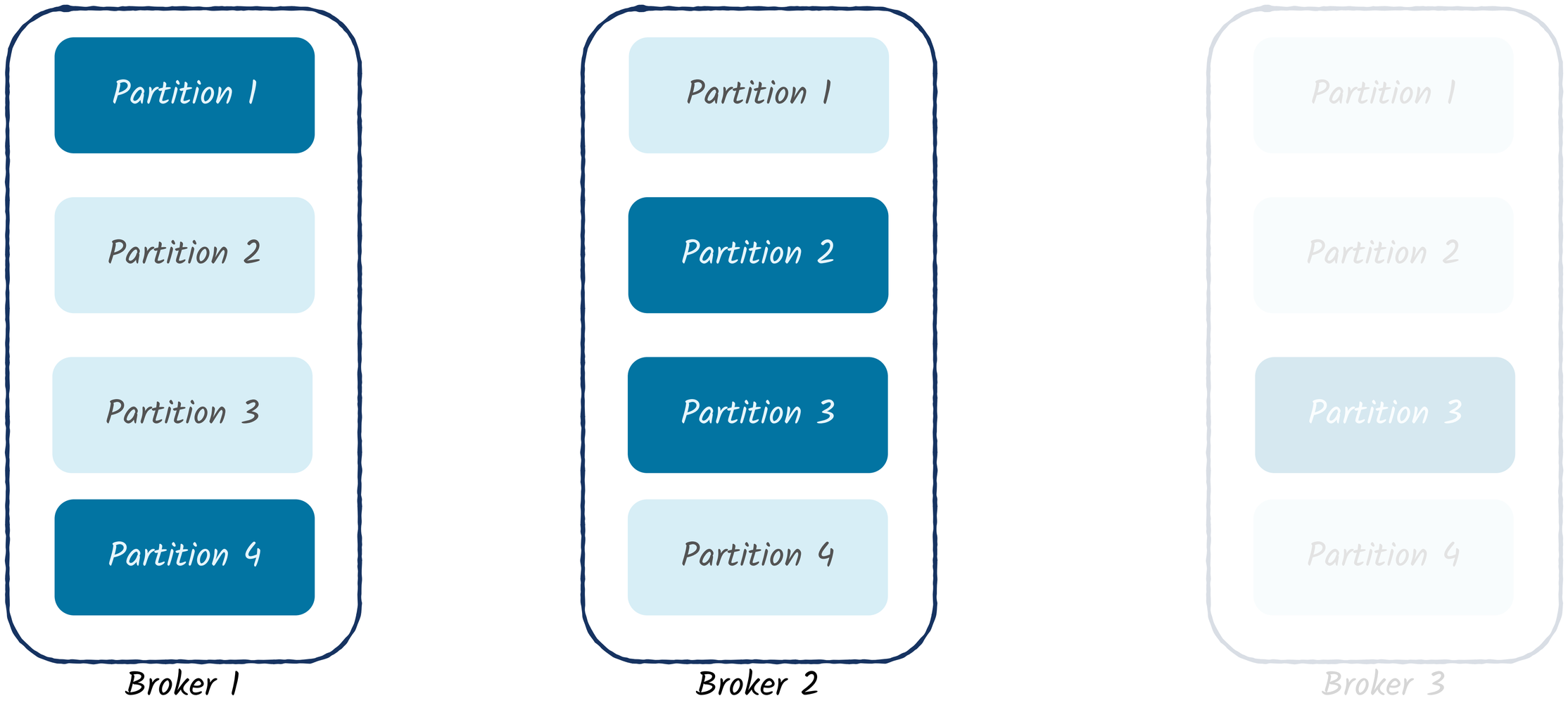
Partition Leadership and Replication



Partition Leadership and Replication



Partition Leadership and Replication





CONFLUENT

DEMO



So far, this is

Pretty good

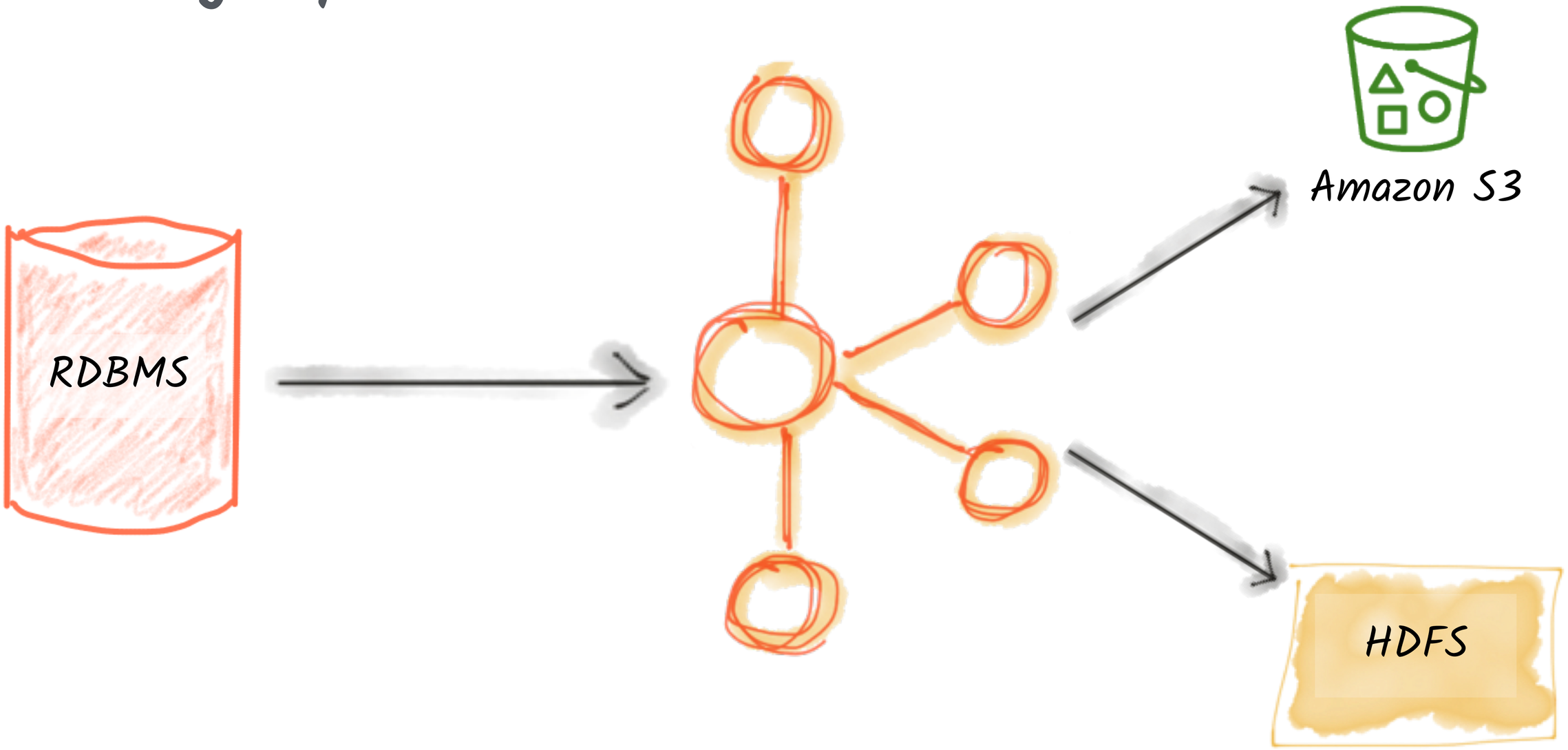


So far, this is

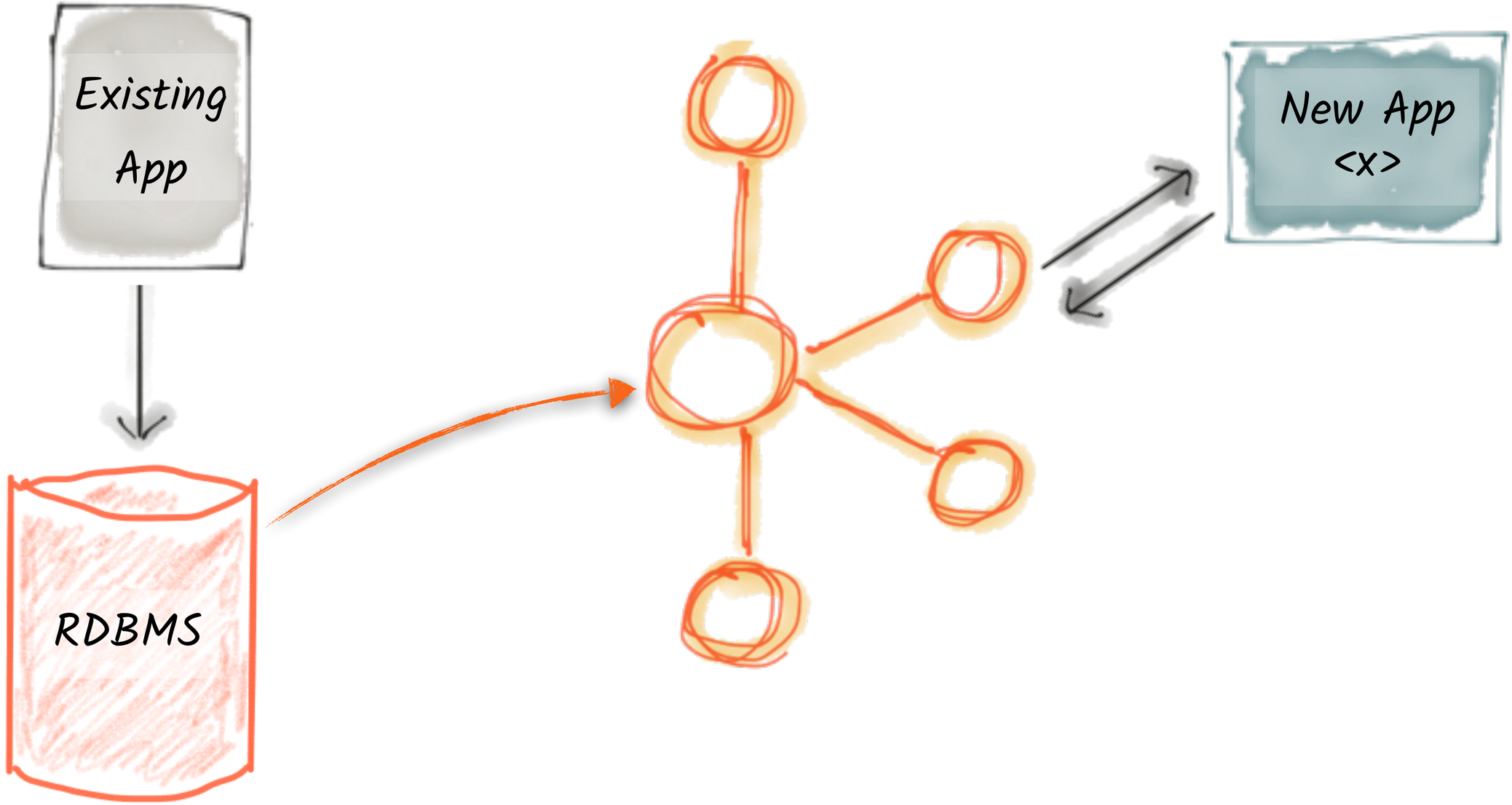
Pretty good

but I've not finished yet...

Streaming Pipelines



Evolve processing from old systems to new

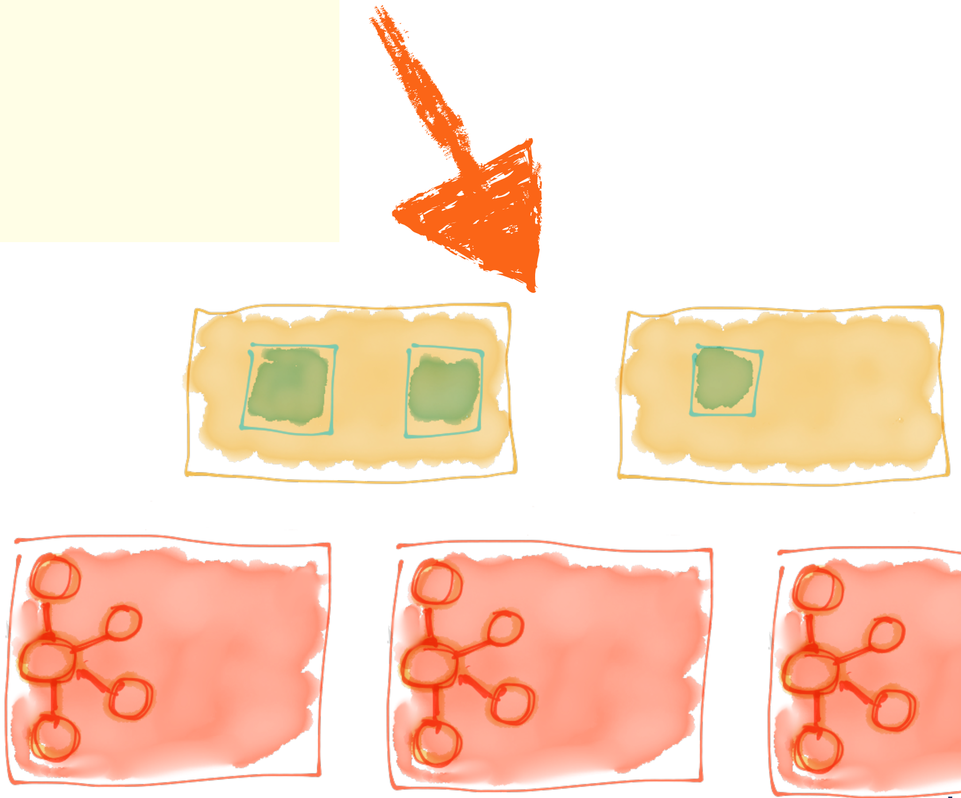




Streaming Integration with Kafka Connect



Sources



Kafka Connect

Kafka Brokers

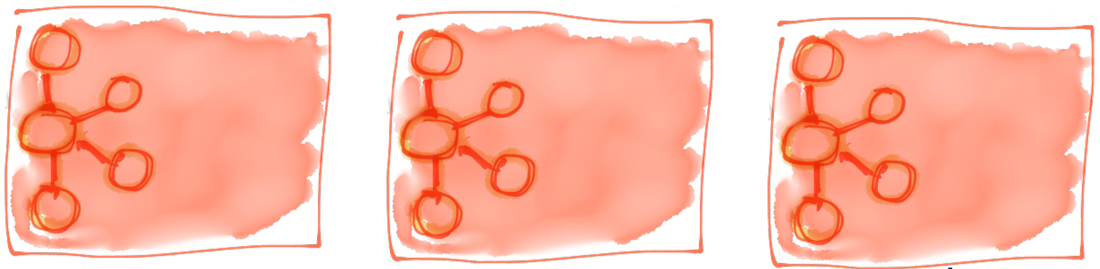
Streaming Integration with Kafka Connect

Sinks





Kafka Connect



Kafka Brokers

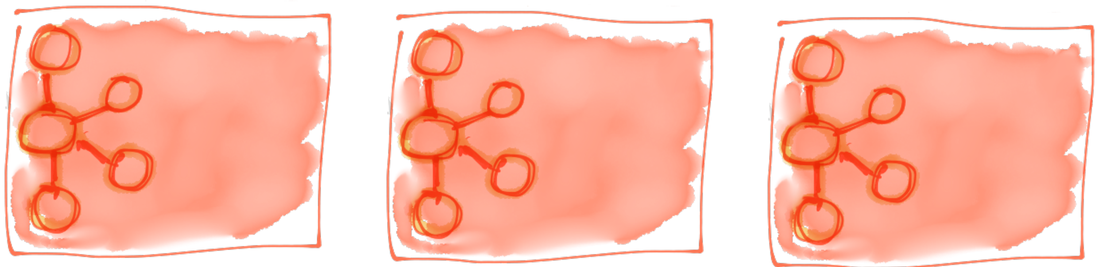
Streaming Integration with Kafka Connect







Kafka Connect



Kafka Brokers

Look Ma, No Code!

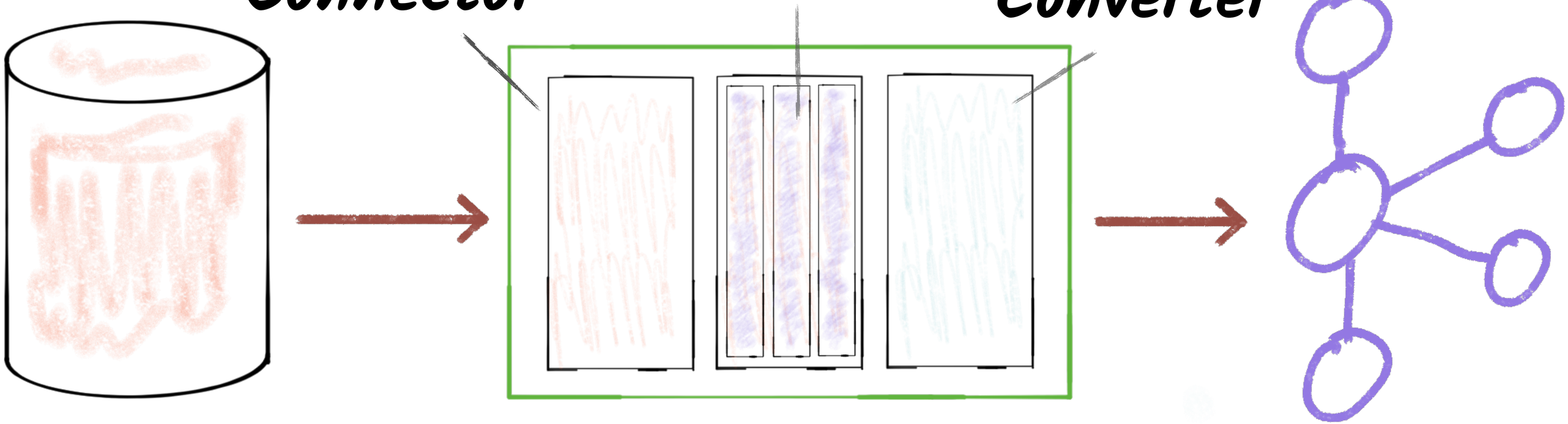
```
{
  "connector.class":
    "io.confluent.connect.jdbc.JdbcSourceConnector",
  "connection.url":
    "jdbc:mysql://asgard:3306/demo",
  "table.whitelist":
    "sales,orders,customers"
}
```

Extensible

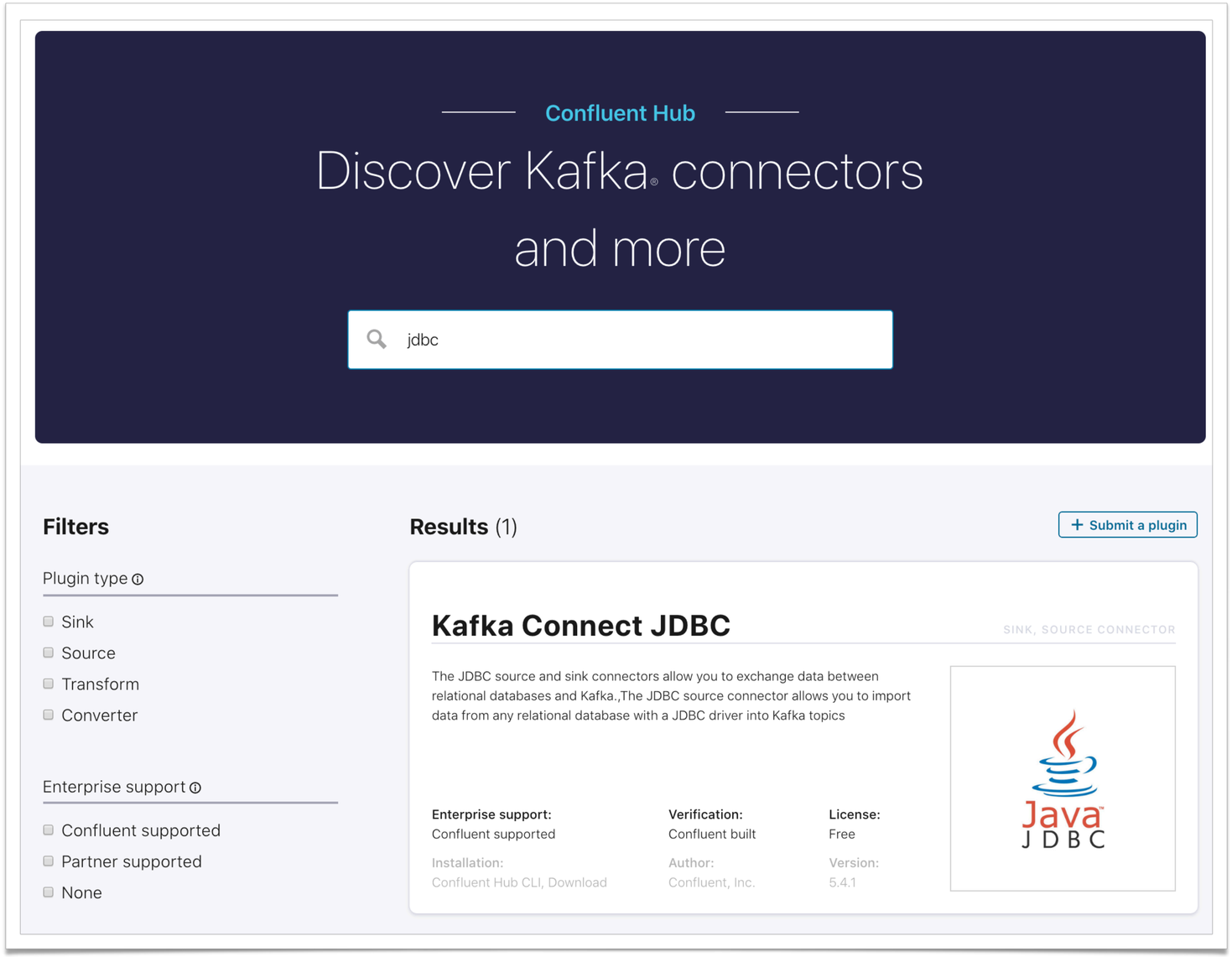
Connector

Transform(s)

Converter

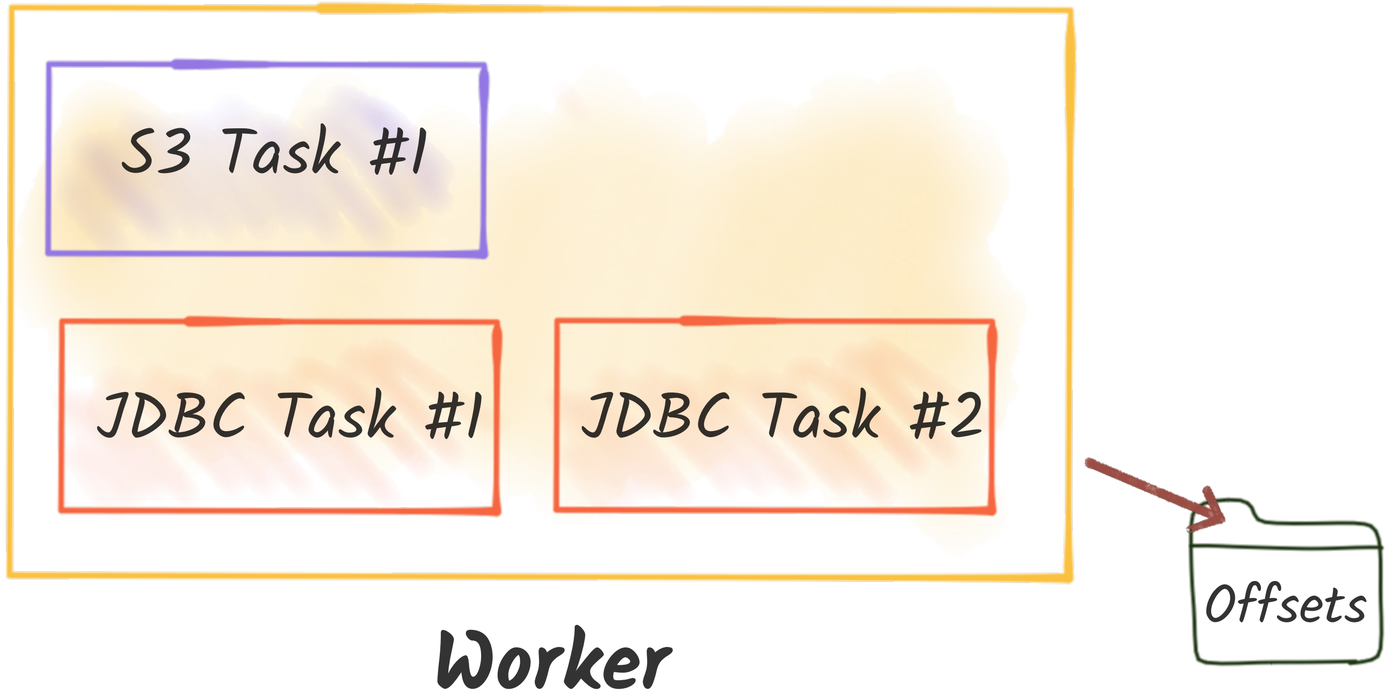


hub.confluent.io



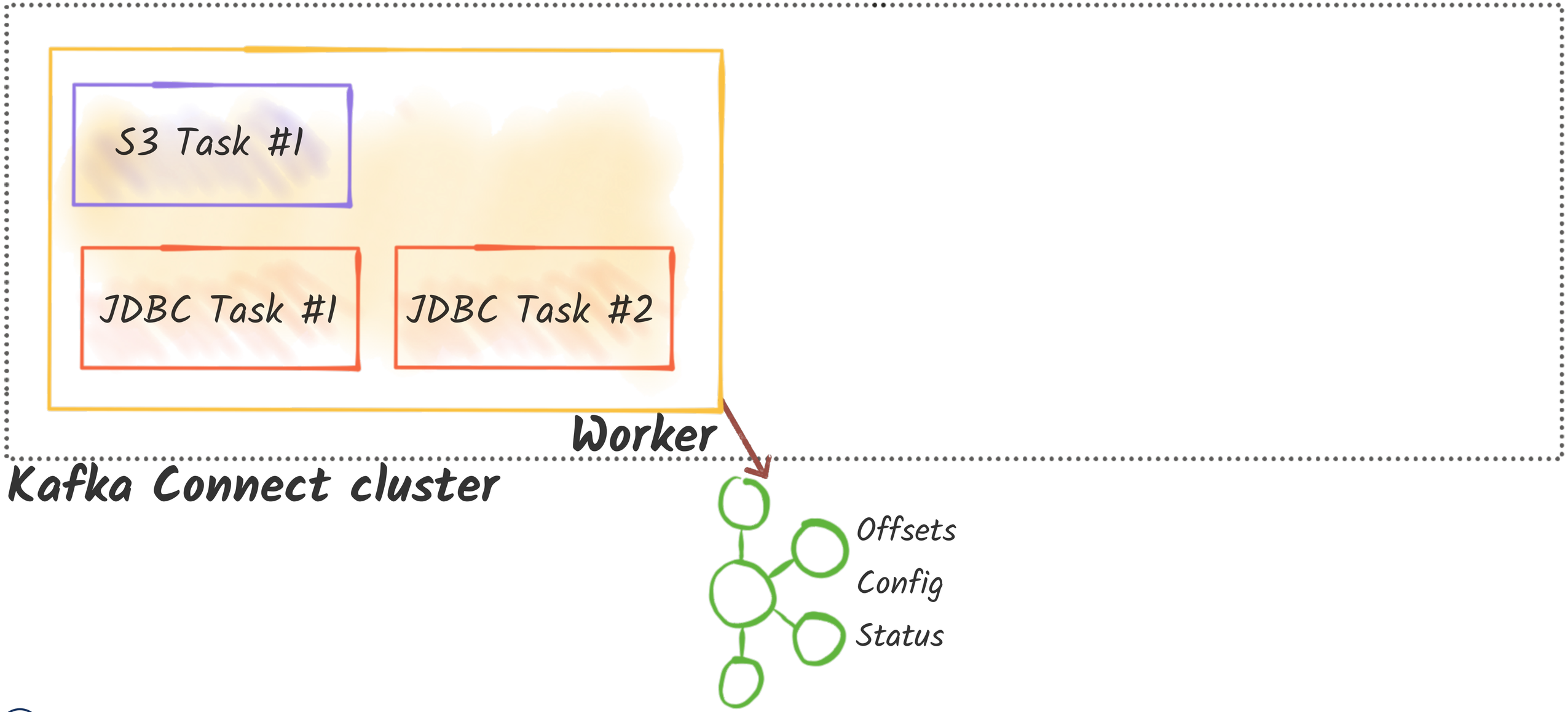
Kafka Connect Standalone Worker

Fault-tolerant? **Nope.**



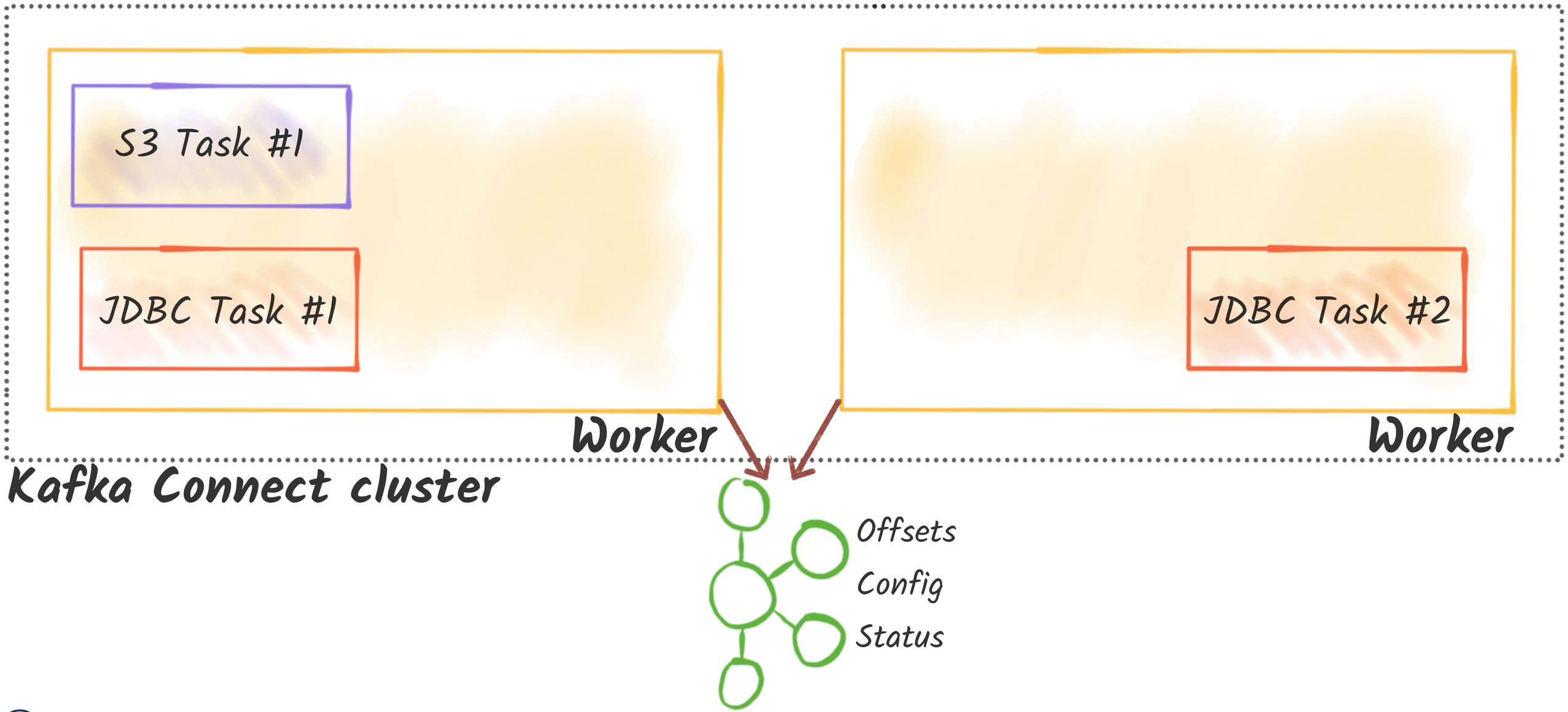
Kafka Connect Distributed Worker

Fault-tolerant? Yeah!



Scaling the Distributed Worker

Fault-tolerant? Yeah!



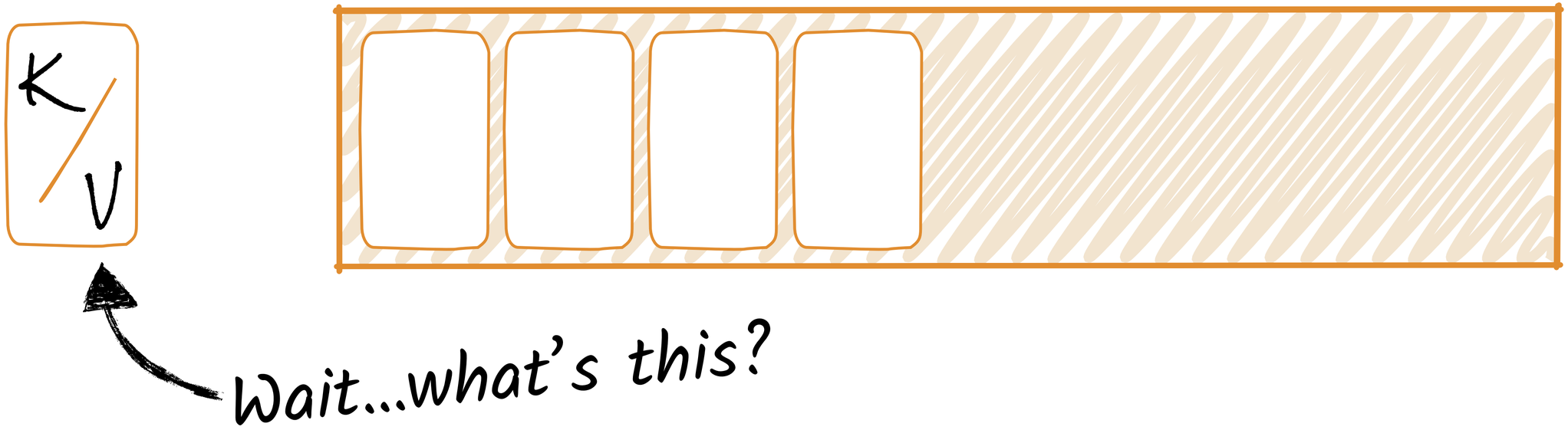
Schema Registry


K
V


K
V


K
V

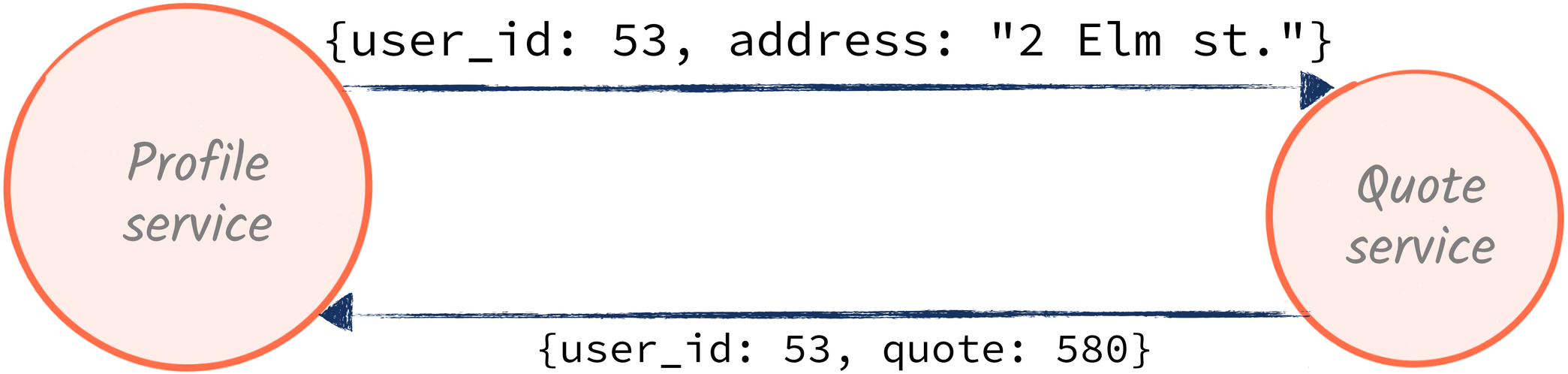

K
V


K
V
Wait...what's this?

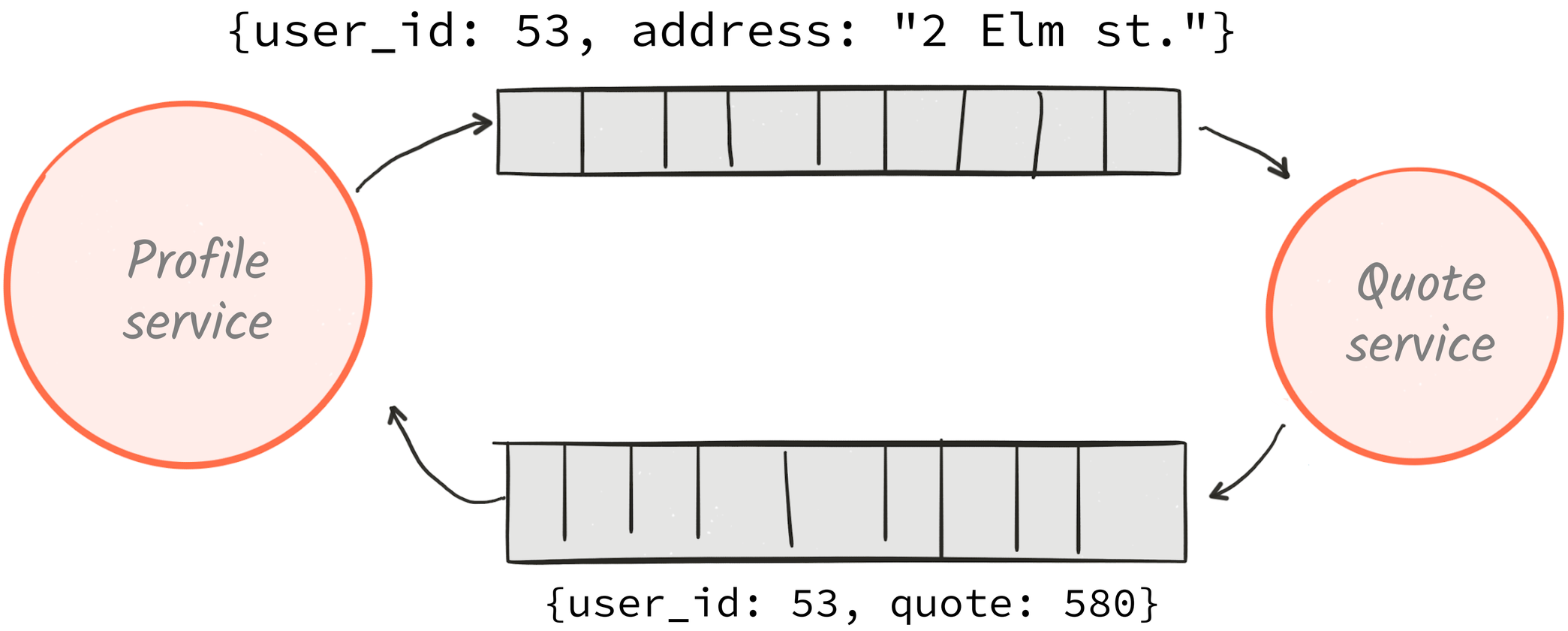
How do you serialise your data?

Avro Protobuf JSON Schema JSON CSV

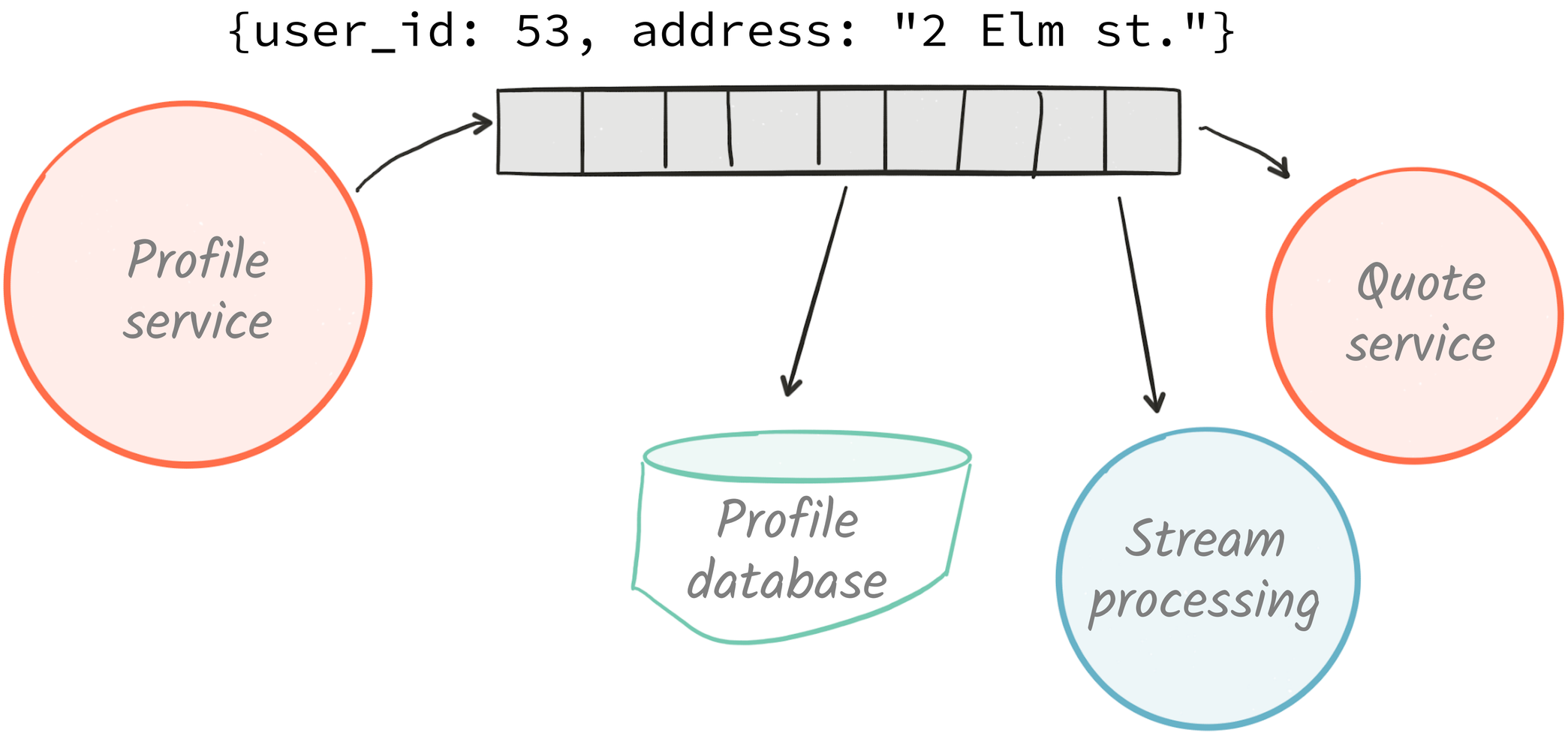
APIs are contracts between services



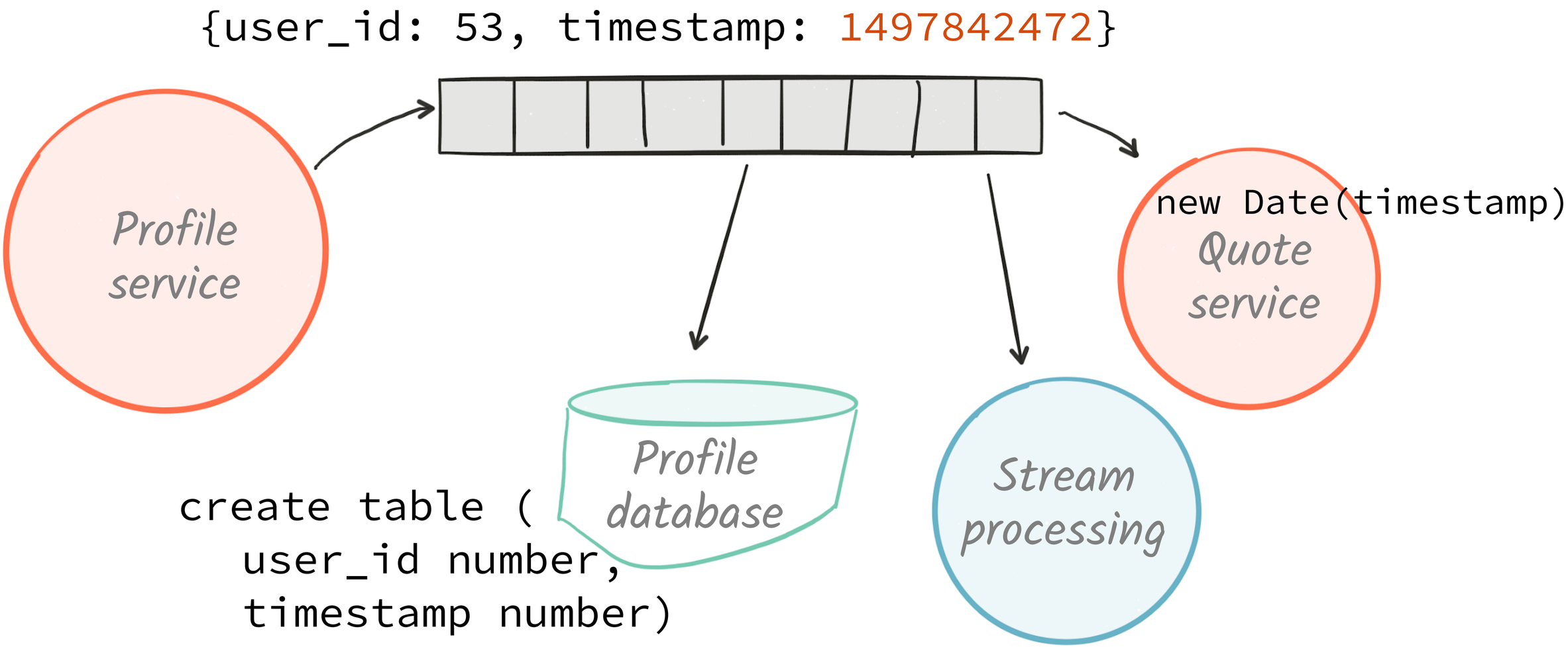
But not all services



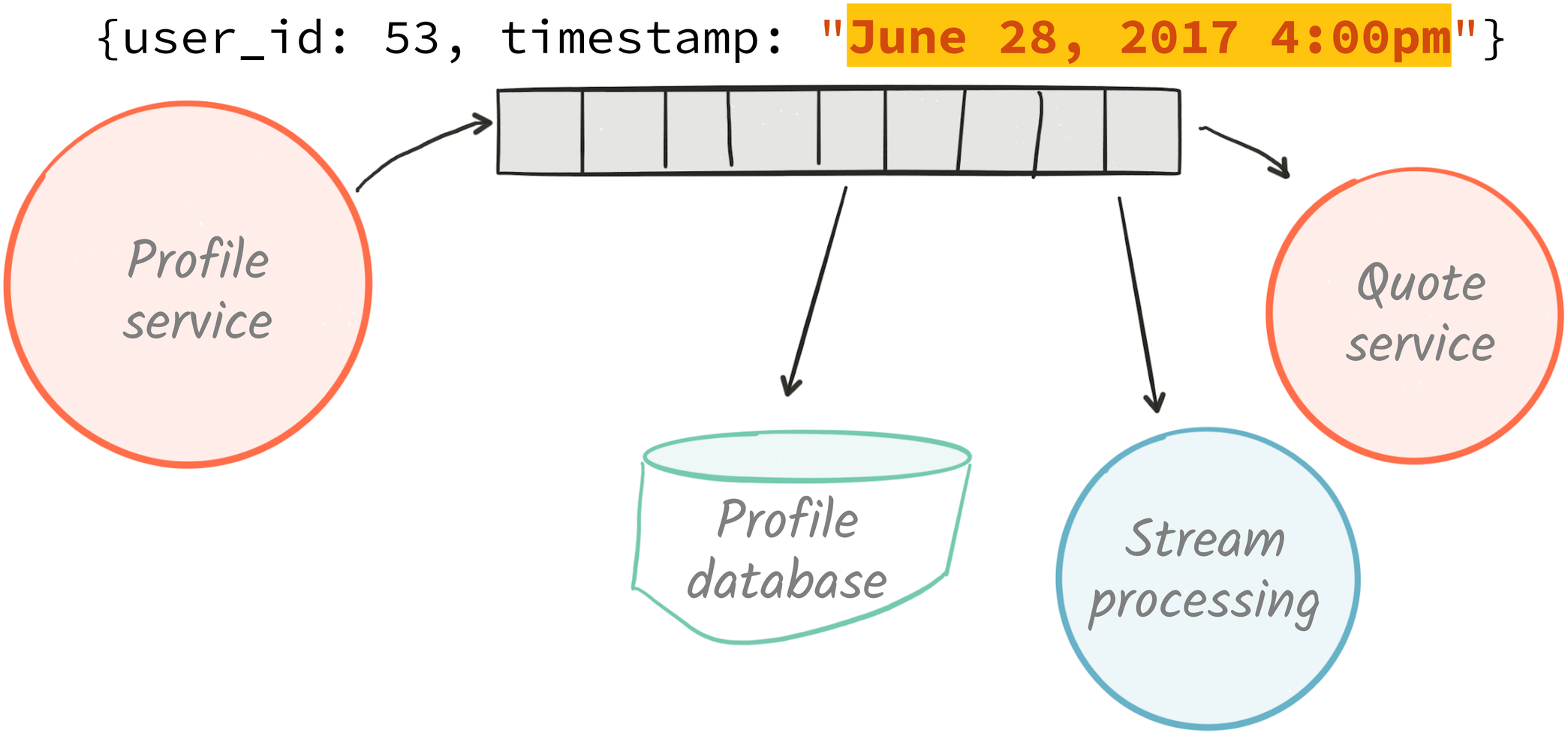
And naturally...



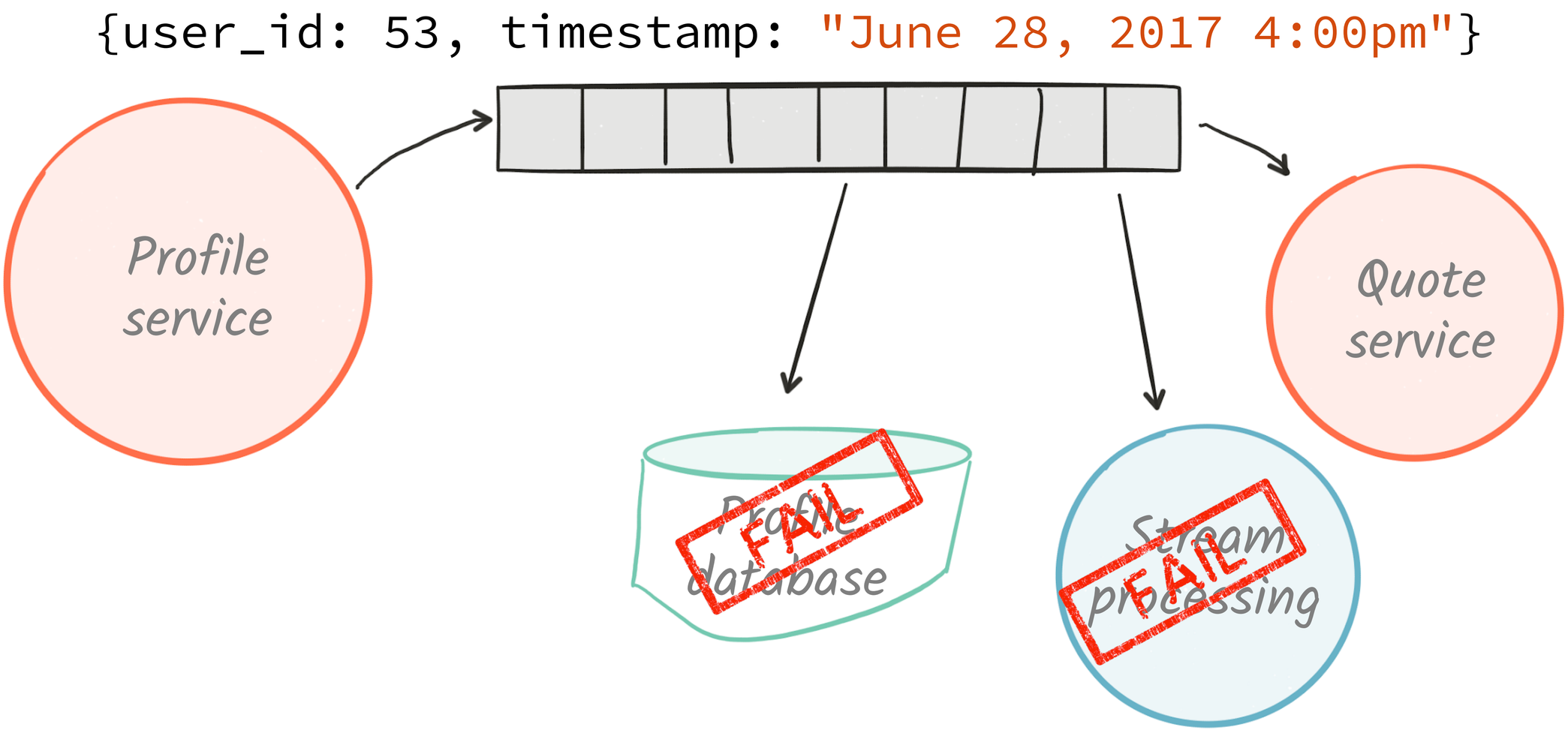
Schemas are about how teams work together



Things change...



Moving fast and breaking things



Lack of schemas - Coupling teams and services

```
2001 2001 Citrus Heights-Sunrise Blvd
Citrus_Hghts 60670001  3400293 34 SAC
Sacramento SV Sacramento Valley SAC
Sacramento County APCD SMA8 Sacramento
Metropolitan Area CA 6920 Sacramento 28 6920
13588 7400 Sunrise Blvd9561038 41 56
38.6988889 121  16 15.98999977
-121.271111  10 4284781 650345  52
```

Serialisation & Schemas

Avro Protobuf JSON Schema JSON CSV

Serialisation & Schemas

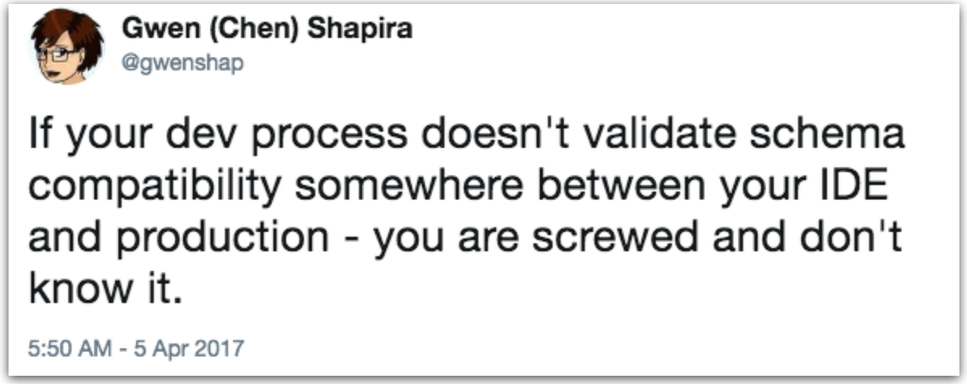
Avro 👍

Protobuf 👍

JSON Schema 👍

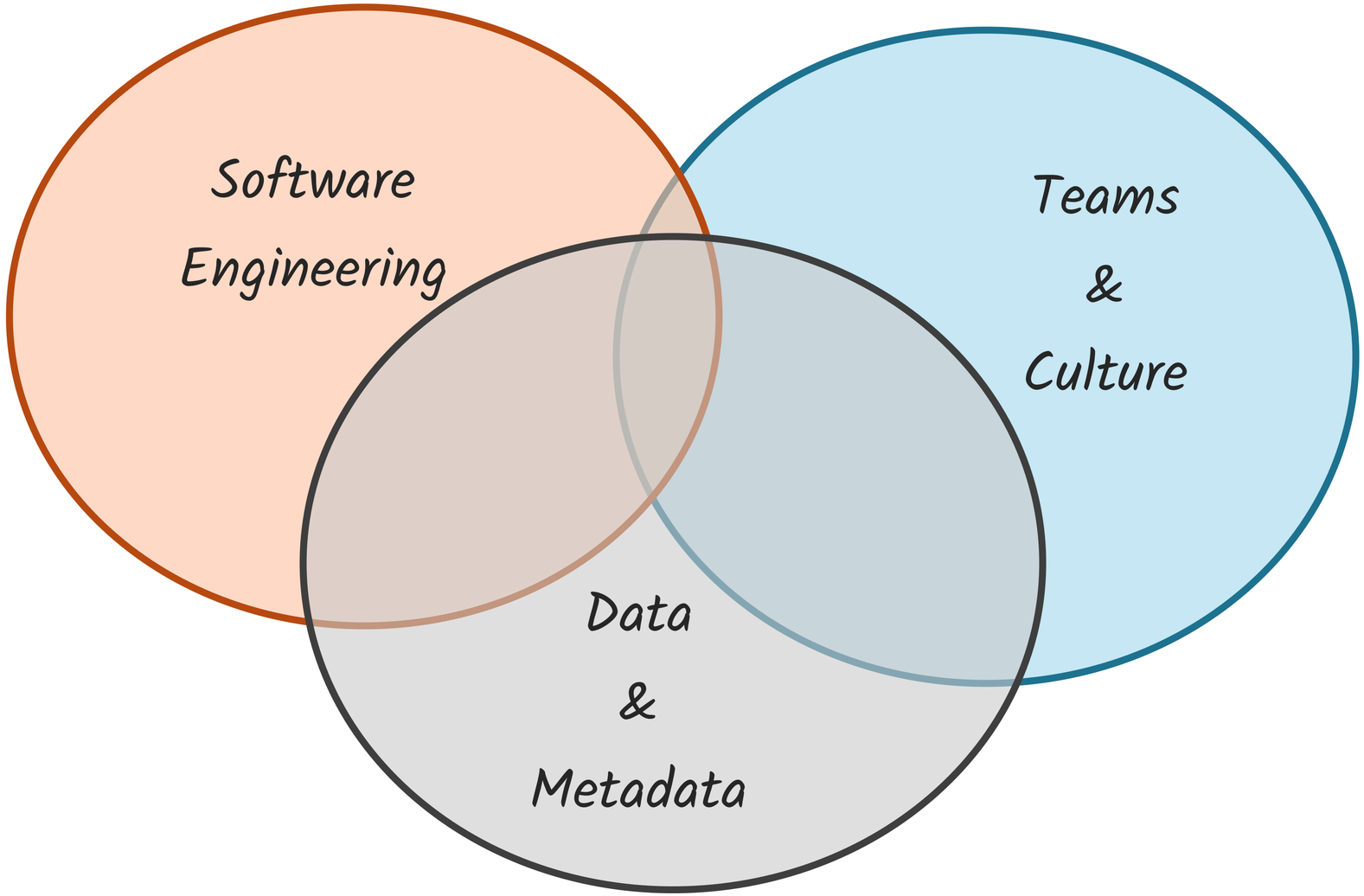
JSON 😬

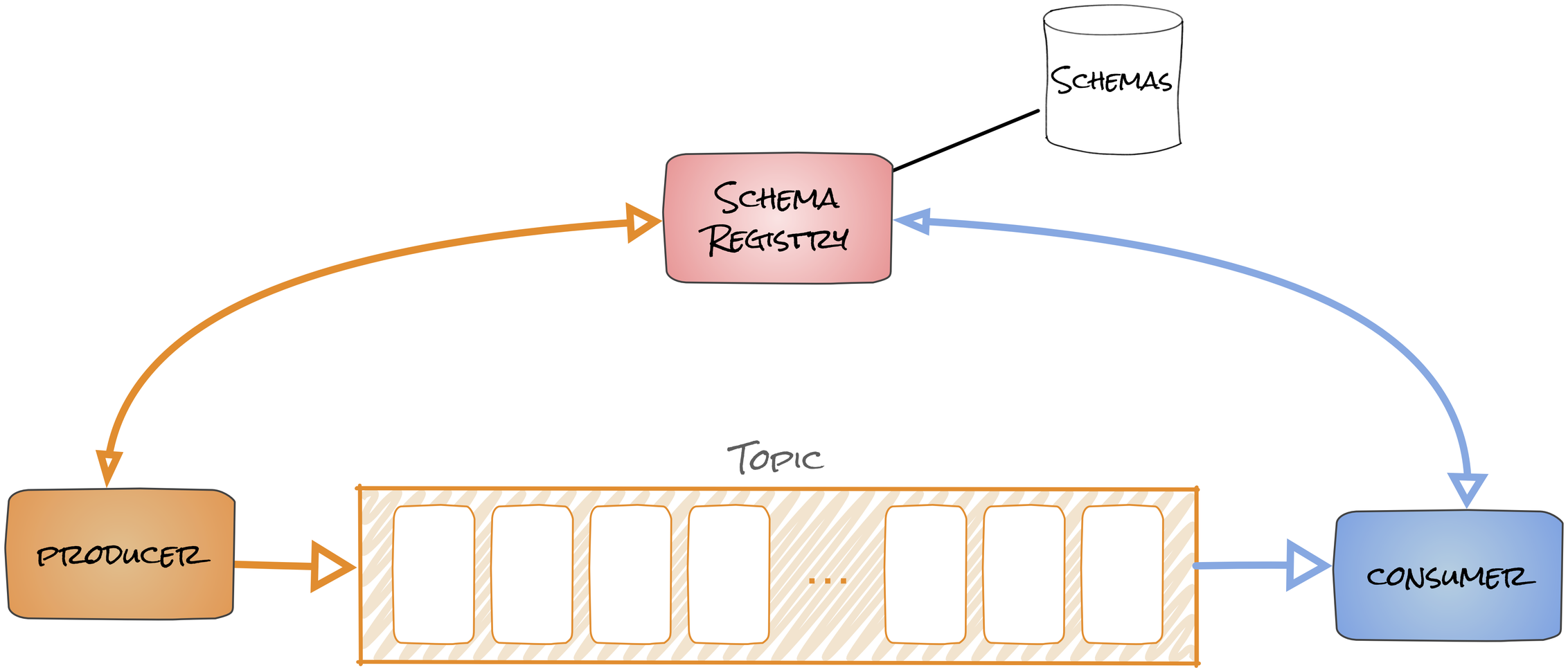
CSV 

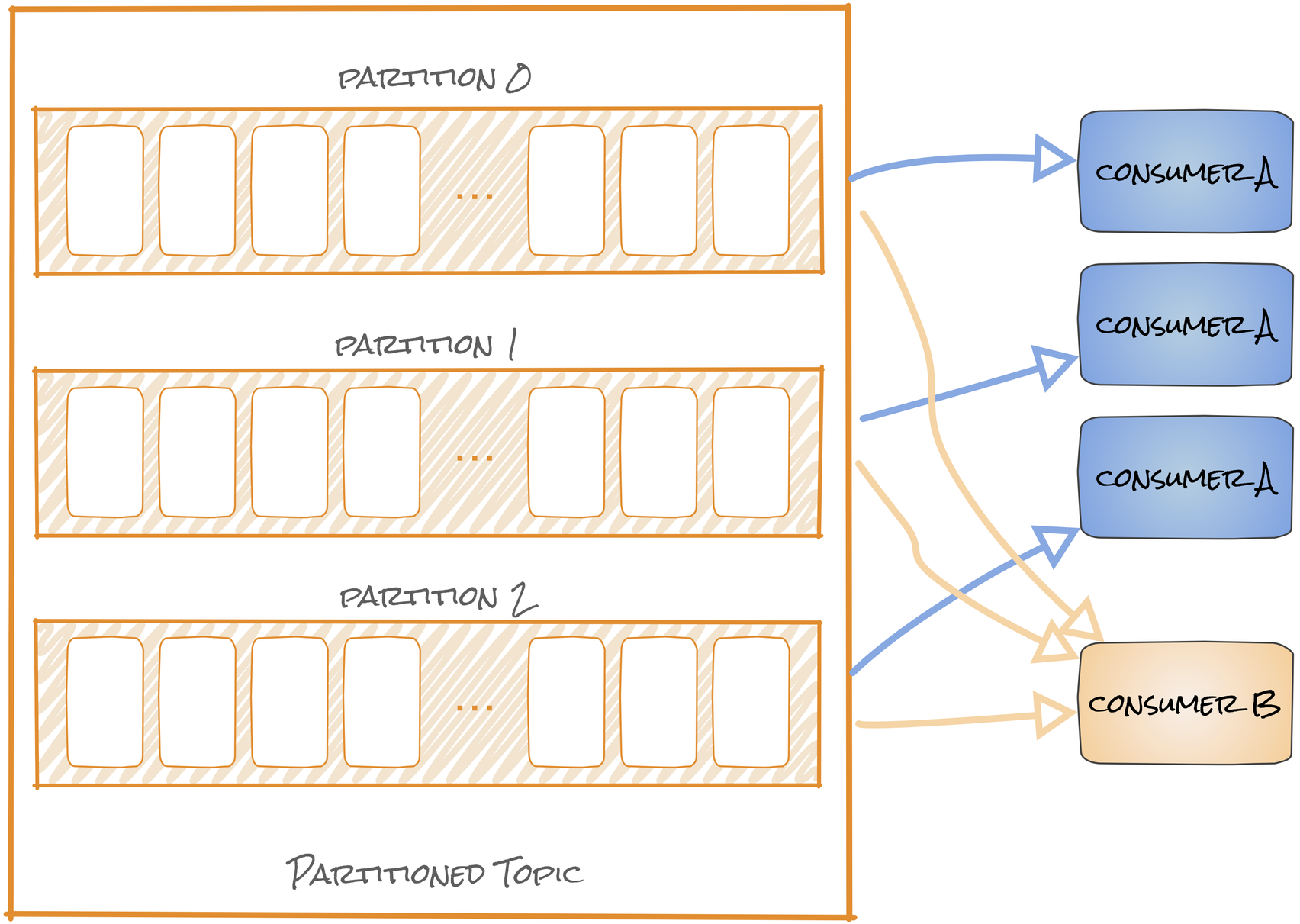


https://rmoff.dev/qcon-schemas

It isn't just about the services




SCHEMAS
SCHEMA REGISTRY
TOPIC
PRODUCER
CONSUMER


PARTITION 0
PARTITION 1
PARTITION 2
PARTITIONED TOPIC
CONSUMER A
CONSUMER A
CONSUMER A
CONSUMER B

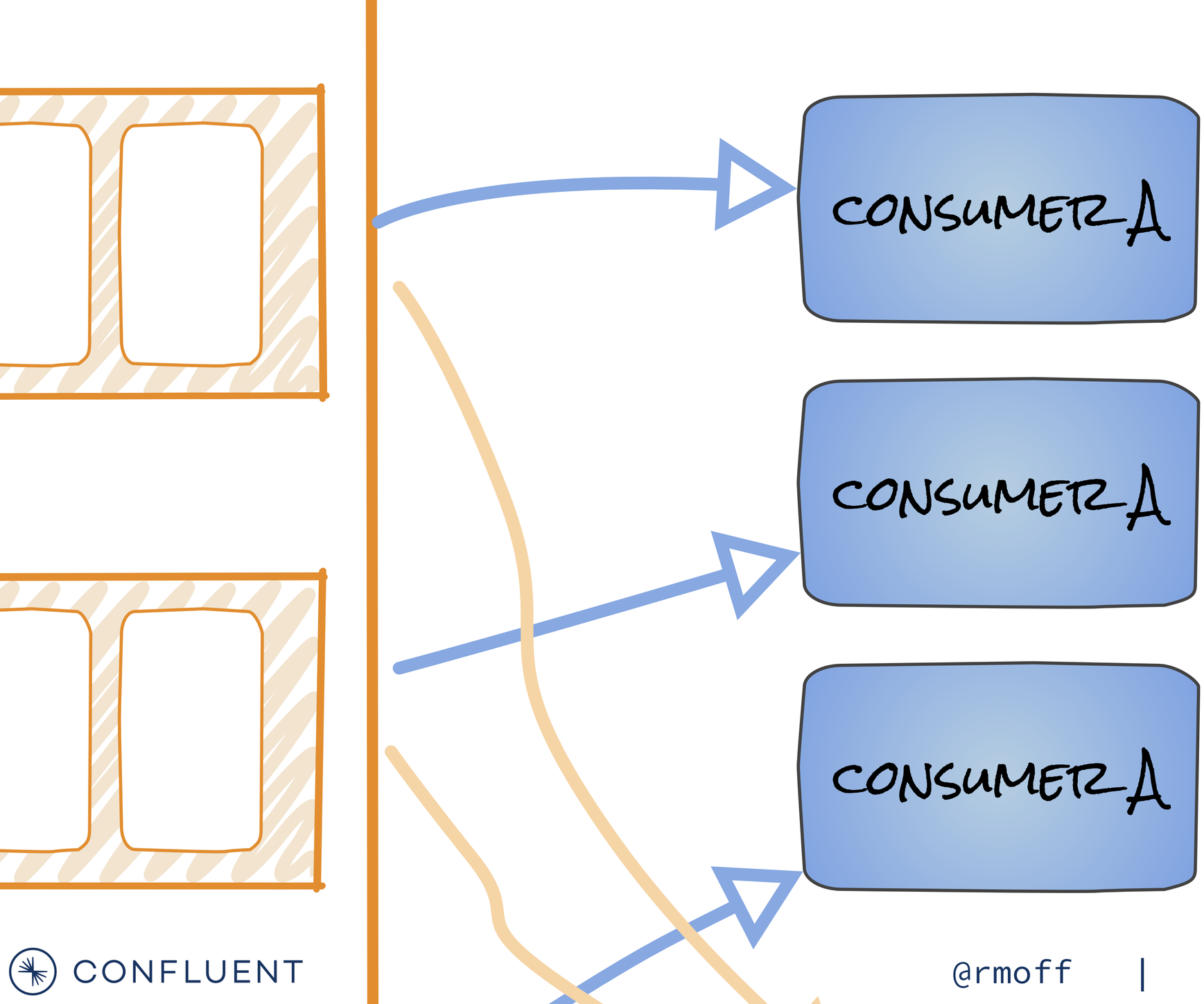
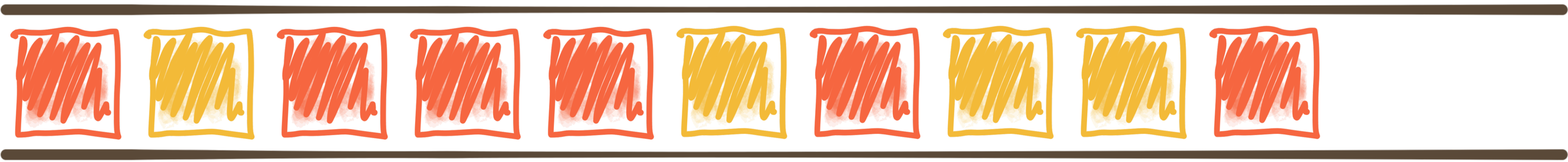

CONSUMER A
CONSUMER A
CONSUMER A



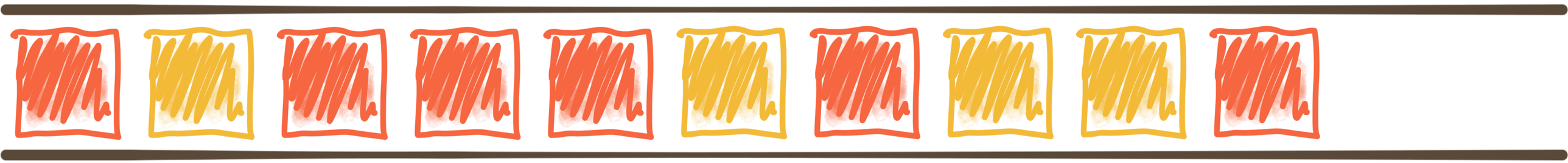
Photo by Franck V. on Unsplash

Streams of events



Time

Stream Processing



Stream: widgets



Stream: widgets_red

Stream Processing with Kafka Streams

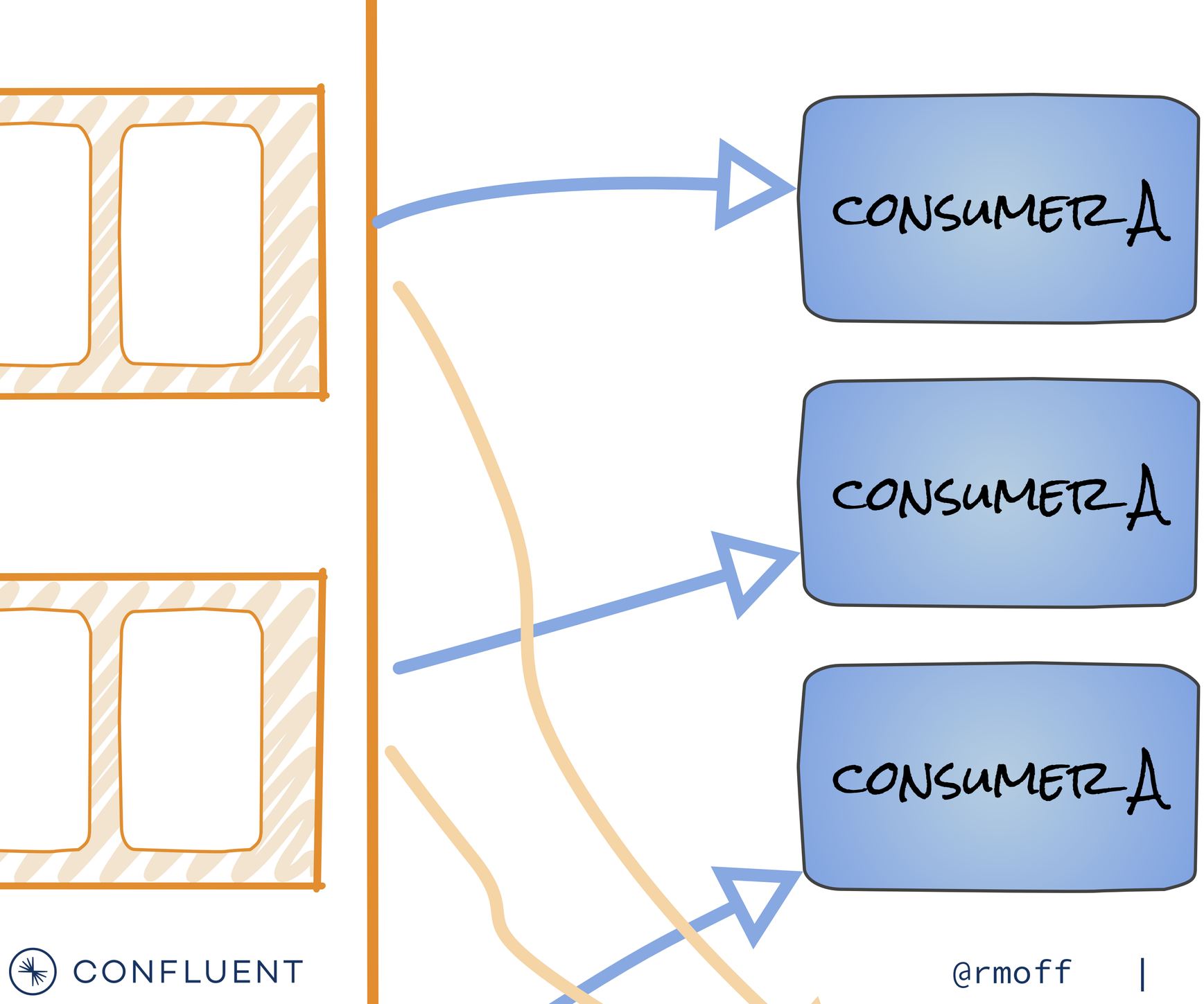


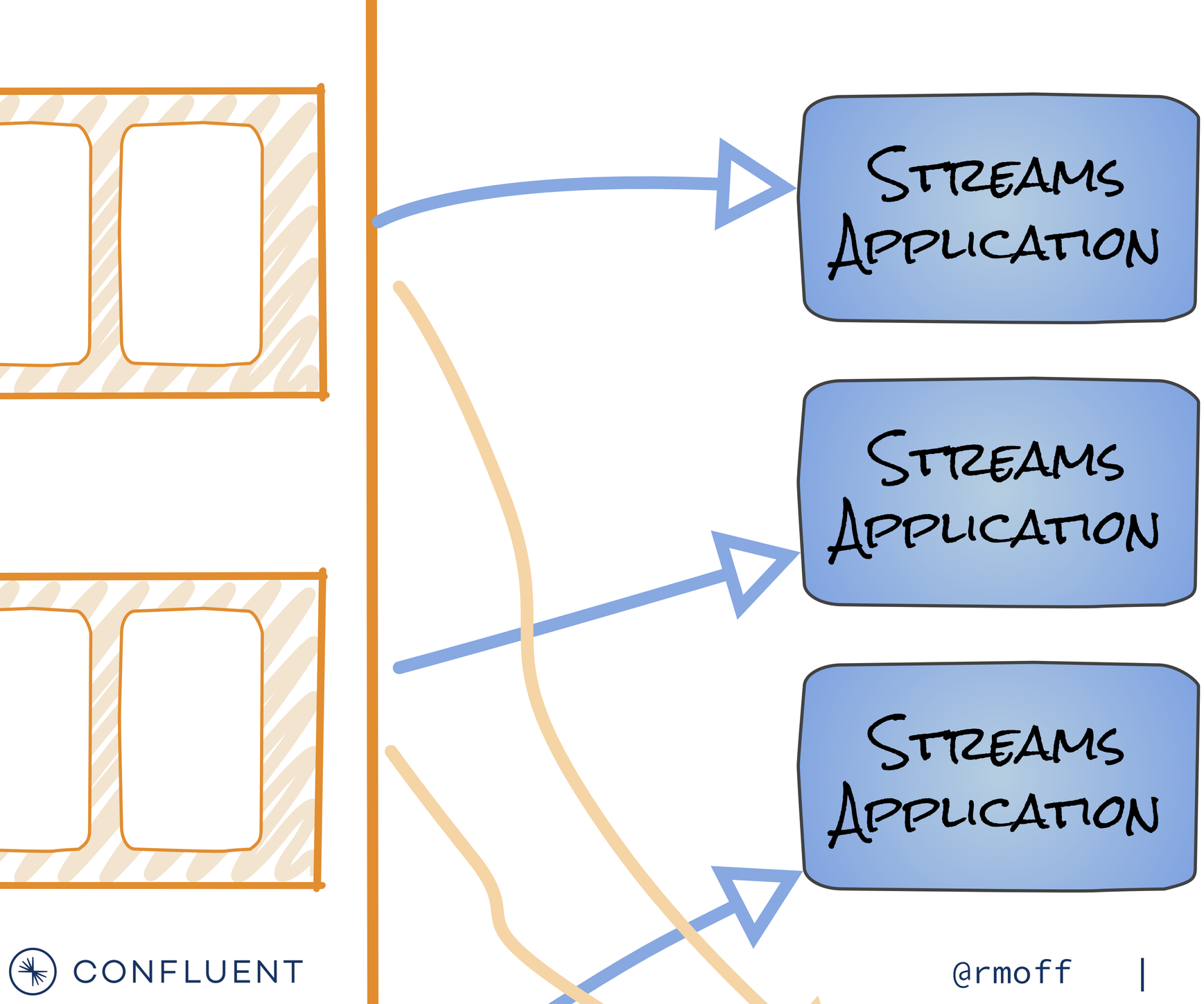
Stream:
widgets

```
final StreamsBuilder builder = new StreamsBuilder()
    .stream("widgets", Consumed.with(stringSerde, widgetsSerde))
    .filter( (key, widget) -> widget.getColour().equals("RED") )
    .to("widgets_red", Produced.with(stringSerde, widgetsSerde));
```



Stream:
widgets_red


CONSUMER A
CONSUMER A
CONSUMER A


Streams Application
Streams Application
Streams Application

```
  Properties streamsConfiguration = getProperties(SCHEMA_REGISTRY_URL);

  final Map<String, String> serdeConfig =
      Collections.singletonMap(AbstractKafkaAvroSerDeConfig.SCHEMA_REGISTRY_URL_CONFIG,
                               SCHEMA_REGISTRY_URL);

  final SpecificAvroSerde<Movie> movieSerde = getMovieAvroSerde(serdeConfig);

  final SpecificAvroSerde<Rating> ratingSerde = getRatingAvroSerde(serdeConfig);

  final SpecificAvroSerde<RatedMovie> ratedMovieSerde = new SpecificAvroSerde<>();
  ratingSerde.configure(serdeConfig, false);

  StreamsBuilder builder = new StreamsBuilder();

  KTable<Long, Double> ratingAverage = getRatingAverageTable(builder);

  getRatedMoviesTable(builder, ratingAverage, movieSerde);

  Topology topology = builder.build();
  KafkaStreams streams = new KafkaStreams(topology, streamsConfiguration);
  Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
  streams.start();
}

private static SpecificAvroSerde<Rating> getRatingAvroSerde(Map<String, String> serdeConfig) {
  final SpecificAvroSerde<Rating> ratingSerde = new SpecificAvroSerde<>();
  ratingSerde.configure(serdeConfig, false);
  return ratingSerde;
```

```
  final SpecificAvroSerde<Movie> movieSerde = new SpecificAvroSerde<>();
  movieSerde.configure(serdeConfig, false);
  return movieSerde;
}

public static KTable<Long, String> getRatedMoviesTable(StreamsBuilder builder,
                                                       KTable<Long, Double> ratingAverage,
                                                       SpecificAvroSerde<Movie> movieSerde) {

  builder.stream("raw-movies", Consumed.with(Serdes.Long(), Serdes.String()))
      .mapValues(Parser::parseMovie)
      .map((key, movie) -> new KeyValue<>(movie.getMovieId(), movie))
      .to("movies", Produced.with(Serdes.Long(), movieSerde));

  KTable<Long, Movie> movies = builder.table("movies",
                                             Materialized
                                                 .<Long, Movie, KeyValueStore<Bytes, byte[]>>as(
                                                     "movies-store")
                                                 .withValueSerde(movieSerde)
                                                 .withKeySerde(Serdes.Long())

  );

  KTable<Long, String> ratedMovies = ratingAverage
      .join(movies, (avg, movie) -> movie.getTitle() + "=" + avg);

  ratedMovies.toStream().to("rated-movies", Produced.with(Serdes.Long(), Serdes.String()));
  return ratedMovies;
}
```

```
        .join(movies, (avg, movie) -> movie.getTitle() + "=" + avg);

    ratedMovies.toStream().to("rated-movies", Produced.with(Serdes.Long(), Serdes.String()));
    return ratedMovies;
  }

  public static KTable<Long, Double> getRatingAverageTable(StreamsBuilder builder) {
    KStream<Long, String> rawRatings = builder.stream("raw-ratings",
                                                      Consumed.with(Serdes.Long(),
                                                                    Serdes.String()));
    KStream<Long, Rating> ratings = rawRatings.mapValues(Parser::parseRating)
        .map((key, rating) -> new KeyValue<>(rating.getMovieId(), rating));

    KStream<Long, Double> numericRatings = ratings.mapValues(Rating::getRating);

    KGroupedStream<Long, Double> ratingsById = numericRatings.groupByKey();

    KTable<Long, Long> ratingCounts = ratingsById.count();
    KTable<Long, Double> ratingSums = ratingsById.reduce((v1, v2) -> v1 + v2);

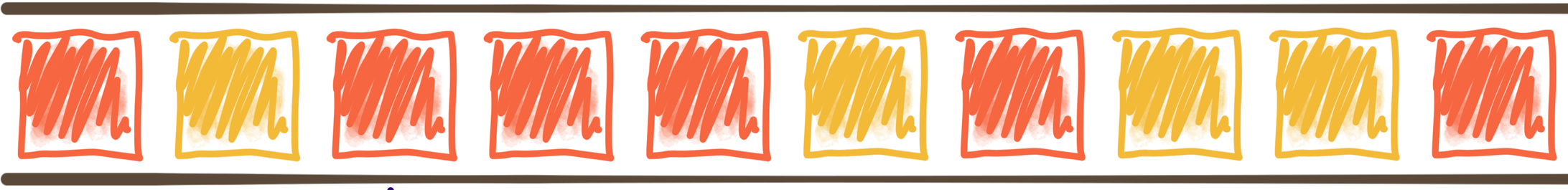
    KTable<Long, Double> ratingAverage = ratingSums.join(ratingCounts,
                                                         (sum, count) -> sum / count.doubleValue(),
                                                         Materialized.as("average-ratings"));

    ratingAverage.toStream()
        /*.peek((key, value) -> { // debug only
          System.out.println("key = " + key + ", value = " + value);
        })*/
        .to("average-ratings");
    return ratingAverage;
  }
```

```
KTable<Long, Movie> movies =
builder.table("movies",
Materialized.
<Long, Movie,KeyValueStore<
Bytes, byte[]>>
as("movies-store")
.withValueSerde(movieSerde)
.withKeySerde(Serdes.Long())
);
```

- **Java API**
- **Filter, join, aggregate, etc.**
- **Locates stream processing with your application**
- **Scales like a Consumer Group (but better!)**

Stream Processing with ksqlDB



Stream:
widgets

ksqlDB

```
CREATE STREAM widgets_red AS
SELECT * FROM widgets
WHERE colour='RED';
```



Stream:
widgets_red



Photo by Franck V. on Unsplash



```
SELECT *
FROM WIDGETS
WHERE WEIGHT_G > 120
```



```
SELECT COUNT(*)
FROM WIDGETS
GROUP BY PRODUCTION_LINE
```



```
SELECT AVG(TEMP_CELCIUS) AS TEMP
FROM WIDGETS
GROUP BY SENSOR_ID
HAVING TEMP>20
```



Object store, data warehouse, RDBMS

```
CREATE SINK CONNECTOR dw WITH (
'connector.class' = 'S3Connector',
'topics' = 'widgets'
…);
```


```
{
    "reading_ts": "2020-02-14T12:19:27Z",
    "sensor_id": "aa-101",
    "production_line": "w01",
    "widget_type": "acme94",
    "temp_celcius": 23,
    "widget_weight_g": 100
}
```

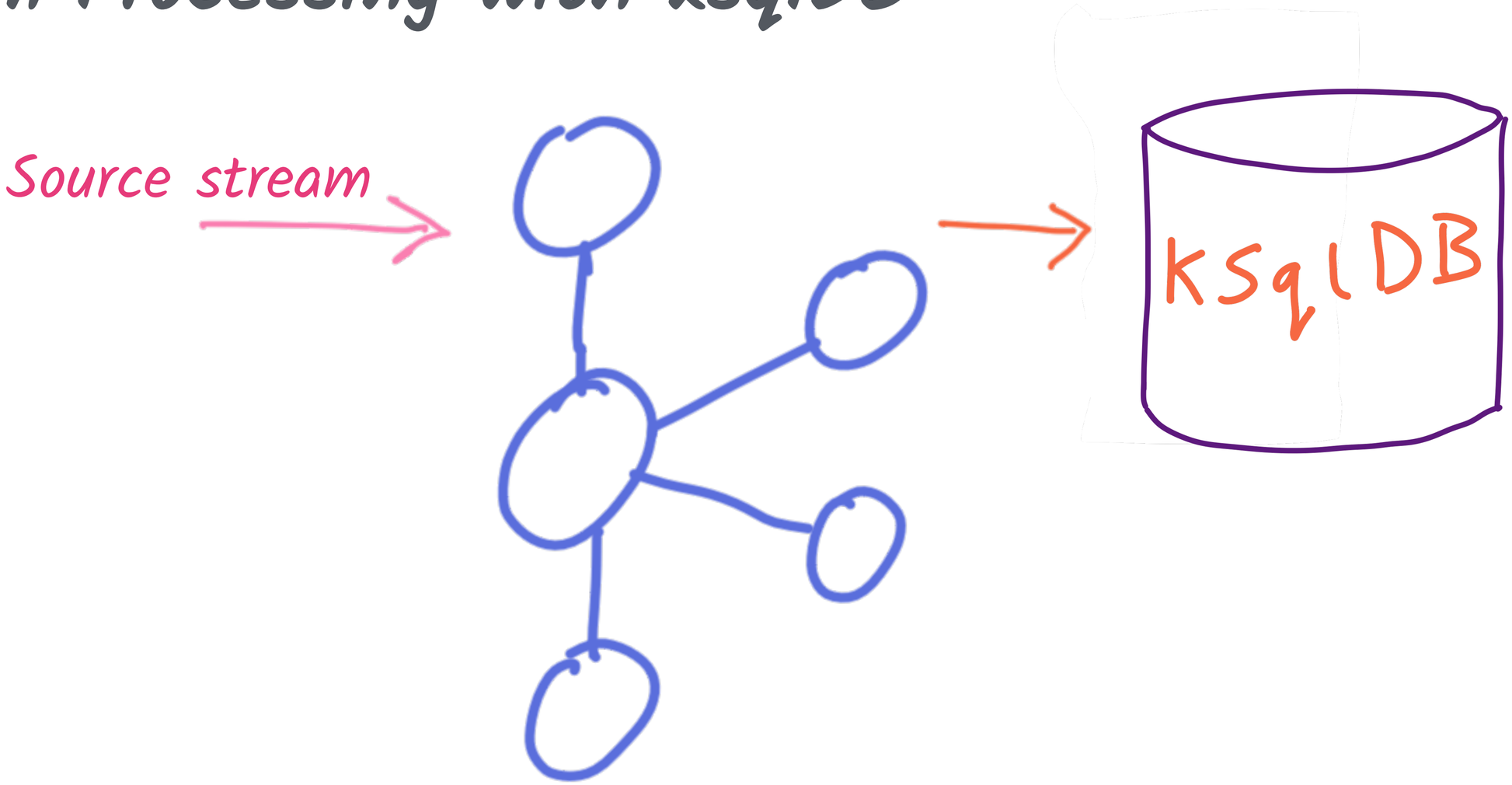

Photo by Franck V. on Unsplash

ksqlDB

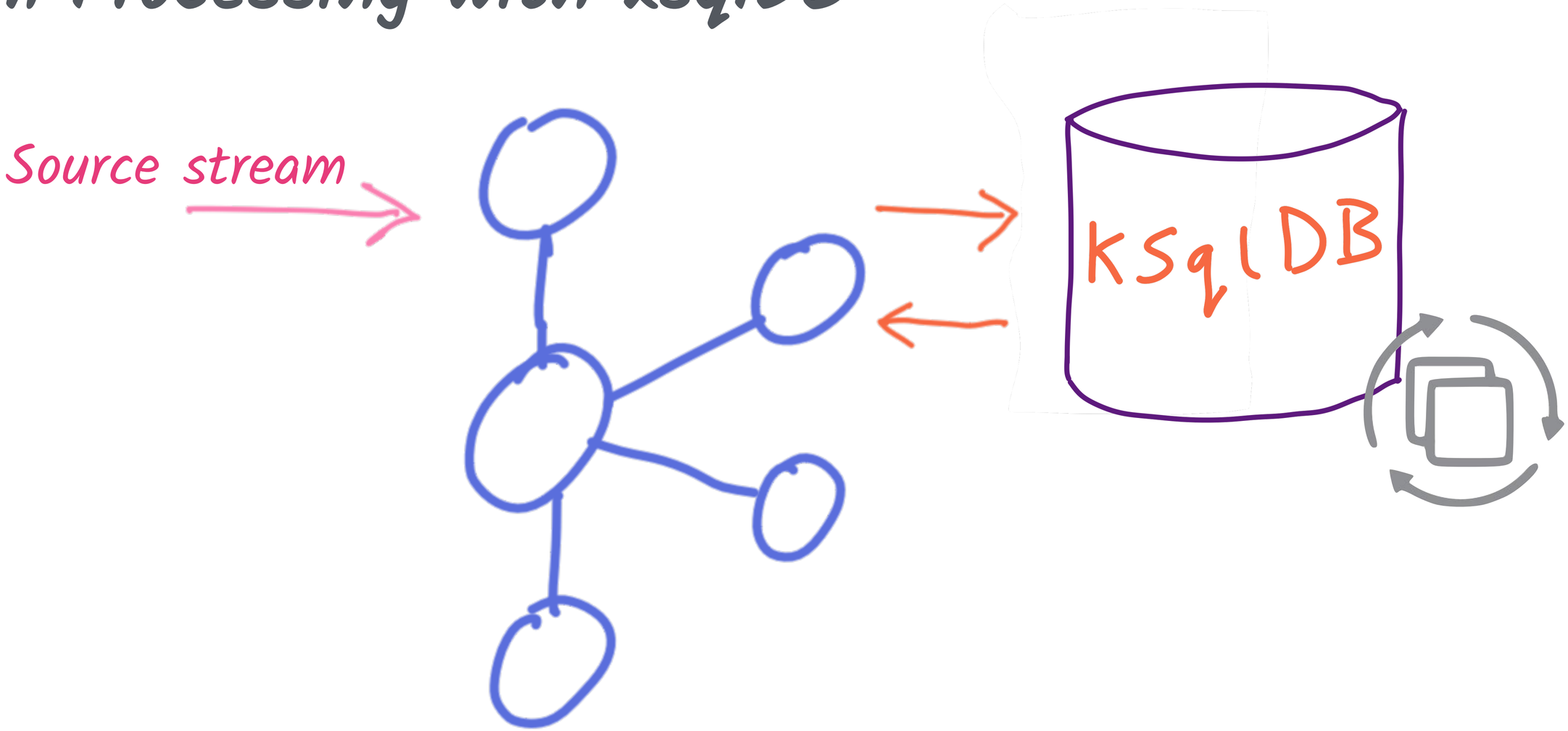


The event streaming database purpose-built for stream processing applications.

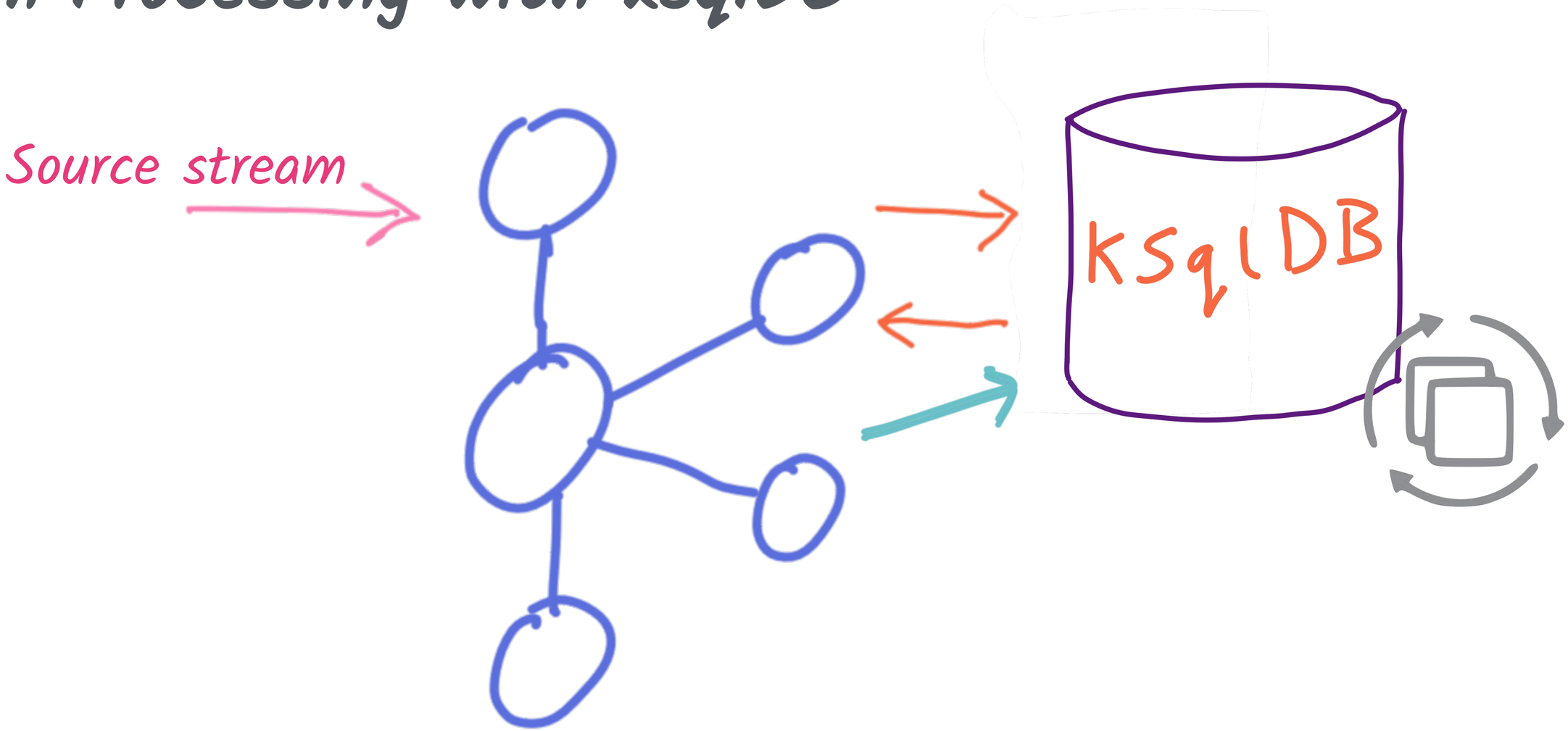
Stream Processing with ksqlDB



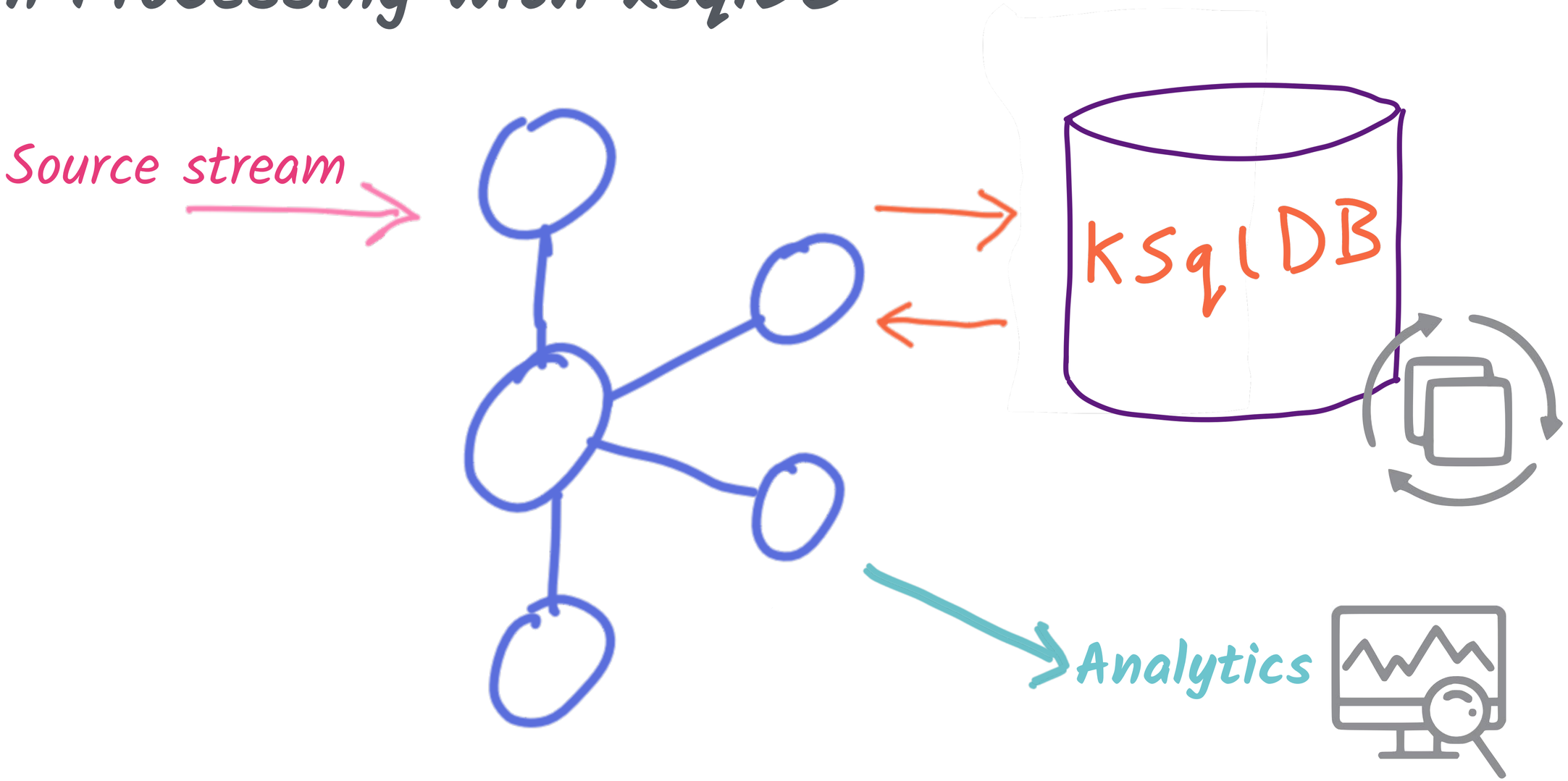
Stream Processing with ksqlDB



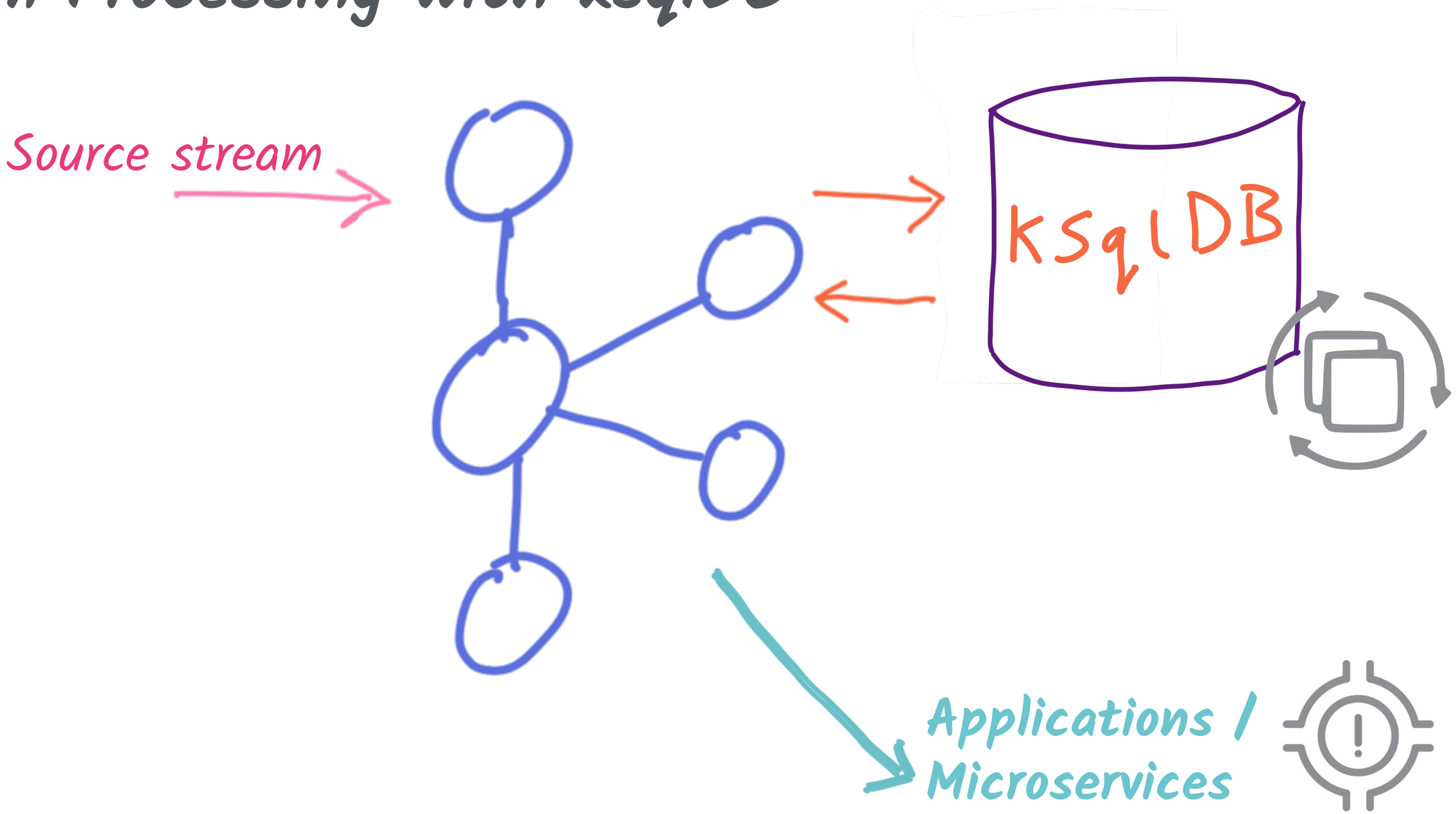
Stream Processing with ksqlDB



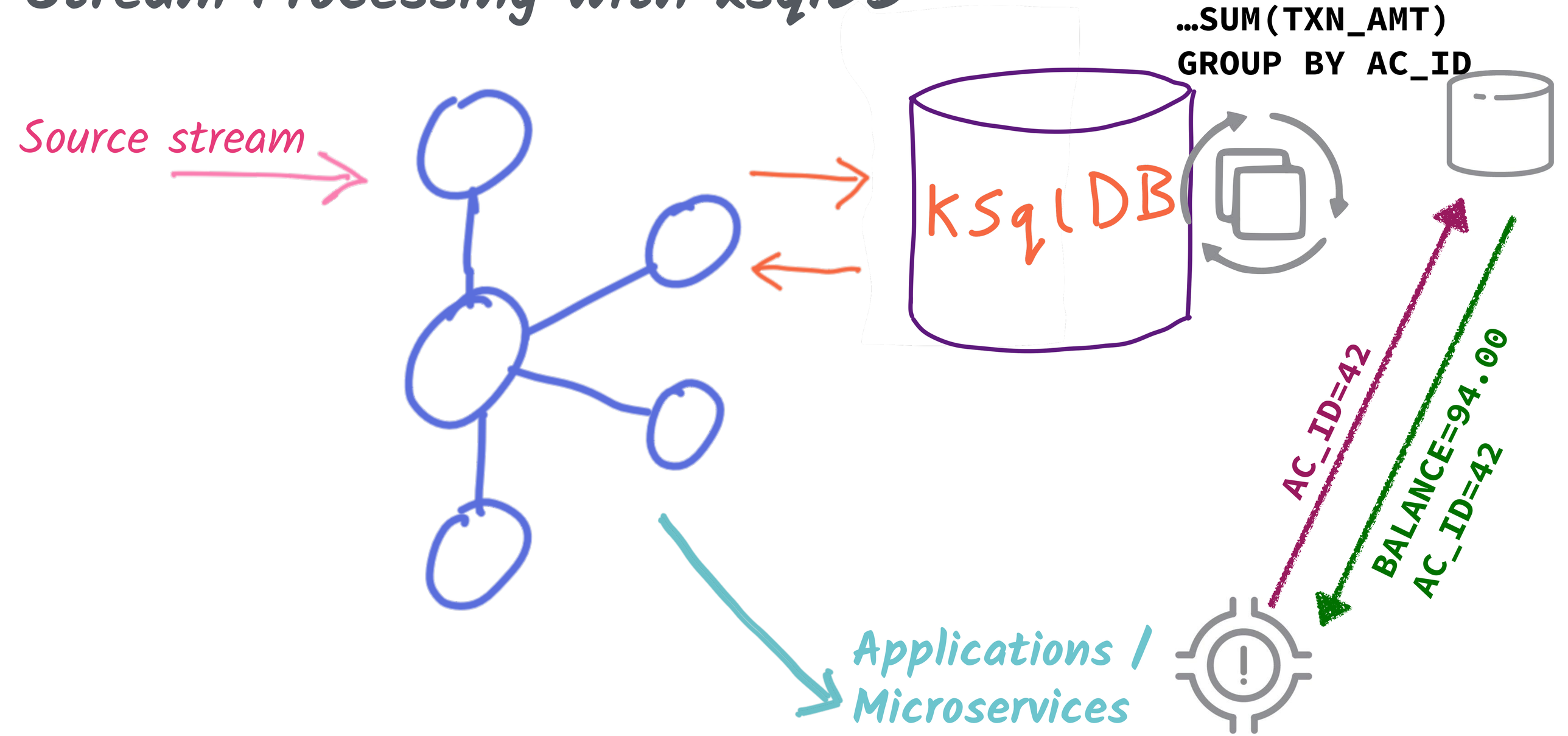
Stream Processing with ksqlDB



Stream Processing with ksqlDB



Stream Processing with ksqlDB



Lookups and Joins with ksqlDB



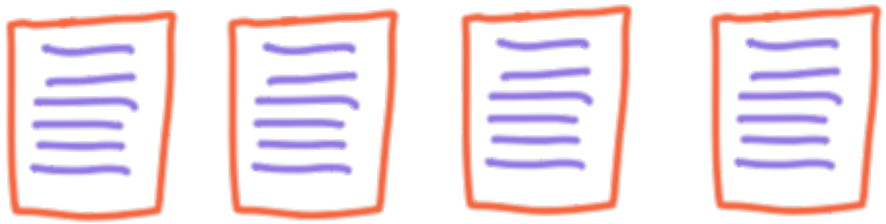
ORDERS


```
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5}
```

Lookups and Joins with ksqlDB



ITEMS



ORDERS


```
{
  "id": "Item_9",
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
 "unit_cost": 19.9
}
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5}
```

Lookups and Joins with ksqlDB



ITEMS



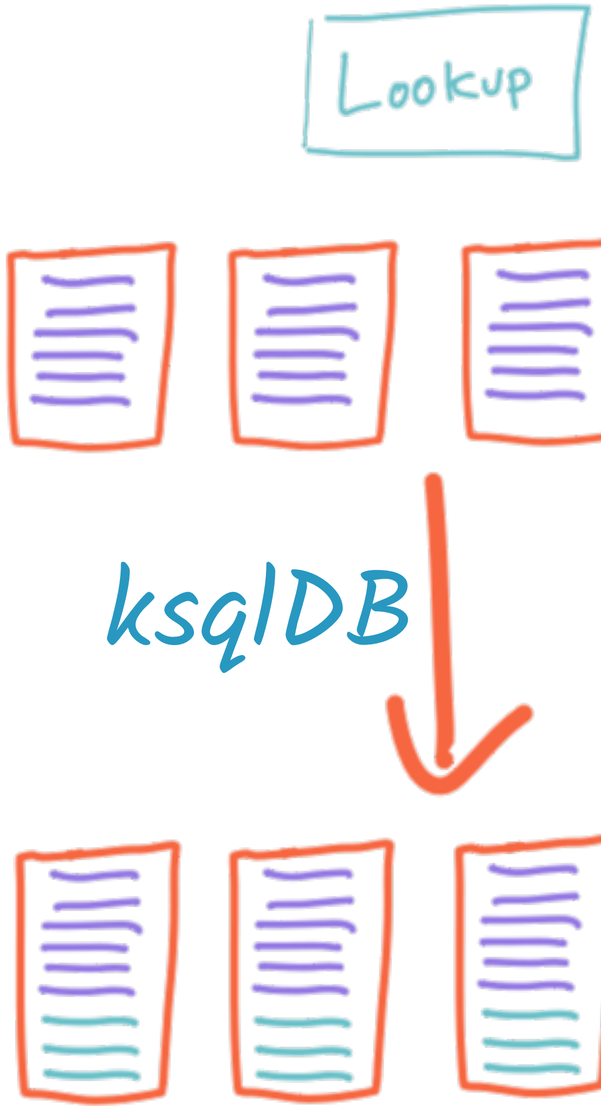
ORDERS




```
CREATE STREAM ORDERS_ENRICHED AS
SELECT O.*, I.*,
  O.ORDERUNITS * I.UNIT_COST
    AS TOTAL_ORDER_VALUE,
  FROM ORDERS O
       INNER JOIN ITEMS I
       ON O.ITEMID = I.ID ;
```


```
{
  "id": "Item_9",
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
 "unit_cost": 19.9
}
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5}
```

Lookups and Joins with ksqlDB



ITEMS

```
{
  "id": "Item_9",
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
 "unit_cost": 19.9
}
```

ORDERS

```
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5}
```

```
CREATE STREAM ORDERS_ENRICHED AS
SELECT O.*, I.*,
  O.ORDERUNITS * I.UNIT_COST
    AS TOTAL_ORDER_VALUE,
  FROM ORDERS O
       INNER JOIN ITEMS I
       ON O.ITEMID = I.ID ;
```

ORDERS_ENRICHED

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
  "unit_cost": 19.9,
  "total_order_value": 99.5
}
```



CONFLUENT

Streams & Tables

Streams and Tables

Kafka topic (k/v bytes)

```
{ "event_ts": "2020-02-17T15:22:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

```
{ "event_ts": "2020-02-17T17:23:00Z",
  "person"  : "robin",
  "location": "London"
}
```

```
{ "event_ts": "2020-02-17T22:23:00Z",
  "person"  : "robin",
  "location": "Wakefield"
}
```

```
{ "event_ts": "2020-02-18T09:00:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

ksqlDB Stream

```
+--------------------+-------+---------+
|EVENT_TS            |PERSON |LOCATION |
+--------------------+-------+---------+
|2020-02-17 15:22:00 |robin  |Leeds    |
|2020-02-17 17:23:00 |robin  |London   |
|2020-02-17 22:23:00 |robin  |Wakefield|
|2020-02-18 09:00:00 |robin  |Leeds    |
```

Stream: Topic + Schema

ksqlDB Table

```
+-------+---------+
|PERSON |LOCATION |
+-------+---------+
|robin  |Leeds    |
```

Table: **state** for given **key**

Topic + Schema

Stateful aggregations in ksqlDB

Kafka topic

```
{ "event_ts": "2020-02-17T15:22:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

```
{ "event_ts": "2020-02-17T17:23:00Z",
  "person"  : "robin",
  "location": "London"
}
```

```
{ "event_ts": "2020-02-17T22:23:00Z",
  "person"  : "robin",
  "location": "Wakefield"
}
```

```
{ "event_ts": "2020-02-18T09:00:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

```
SELECT PERSON,
       COUNT(*)
FROM   MOVEMENTS
GROUP BY PERSON;
```

```
+-------+------------------+
|PERSON | LOCATION_CHANGES |
+-------+------------------+
|robin  | 4                |
```

```
SELECT PERSON,
       COUNT_DISTINCT(LOCATION)
FROM   MOVEMENTS
GROUP BY PERSON;
```

```
+-------+------------------+
|PERSON | UNIQUE_LOCATIONS |
+-------+------------------+
|robin  | 3                |
```



Aggregations can be across the entire input, or windowed (TUMBLING, HOPPING, SESSION)

Stateful aggregations in ksqlDB

Kafka topic

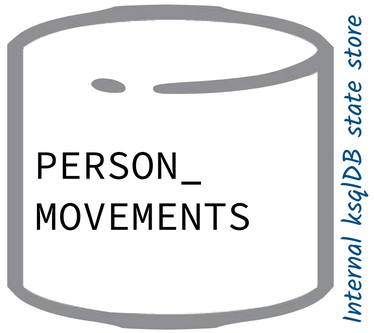
```
{ "event_ts": "2020-02-17T15:22:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

```
{ "event_ts": "2020-02-17T17:23:00Z",
  "person"  : "robin",
  "location": "London"
}
```

```
{ "event_ts": "2020-02-17T22:23:00Z",
  "person"  : "robin",
  "location": "Wakefield"
}
```

```
{ "event_ts": "2020-02-18T09:00:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

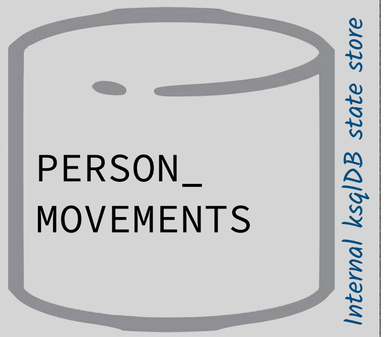
```
CREATE TABLE PERSON_MOVEMENTS AS
 SELECT PERSON,
        COUNT_DISTINCT(LOCATION) AS
          UNIQUE_LOCATIONS,
        COUNT(*) AS LOCATION_CHANGES
   FROM MOVEMENTS
 GROUP BY PERSON;
```



Pull and Push queries in ksqlDB




```
CREATE TABLE PERSON_MOVEMENTS AS
 SELECT PERSON,
        COUNT_DISTINCT(LOCATION) AS
          UNIQUE_LOCATIONS,
        COUNT(*) AS LOCATION_CHANGES
   FROM MOVEMENTS
 GROUP BY PERSON;
```




```
ksql> SELECT LOCATION_CHANGES, UNIQUE_LOCATIONS
      FROM PERSON_MOVEMENTS
      WHERE ROWKEY='robin';
+------------------+-----------------+
|LOCATION_CHANGES  |UNIQUE_LOCATIONS |
+------------------+-----------------+
|3                 |3                |
Query terminated
ksql>
```

Pull query

```
ksql> SELECT LOCATION_CHANGES, UNIQUE_LOCATIONS
      FROM PERSON_MOVEMENTS
      WHERE ROWKEY='robin';
      EMIT CHANGES;
+------------------+-----------------+
|LOCATION_CHANGES  |UNIQUE_LOCATIONS |
+------------------+-----------------+
|1                 |1                |
|2                 |2                |
|3                 |3                |
|4                 |3                |

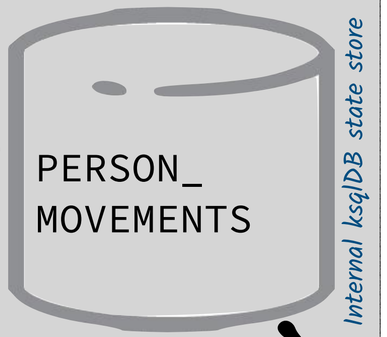
Press CTRL-C to interrupt
```

Push query

ksqlDB REST API




```
CREATE TABLE PERSON_MOVEMENTS AS
 SELECT PERSON,
        COUNT_DISTINCT(LOCATION) AS
          UNIQUE_LOCATIONS,
        COUNT(*) AS LOCATION_CHANGES
   FROM MOVEMENTS
 GROUP BY PERSON;
```




```
curl -s -X "POST" "http://localhost:8088/query" \
-H "Content-Type: application/vnd.ksql.v1+json; charset=utf-8" \
-d '{"ksql":"SELECT UNIQUE_LOCATIONS FROM PERSON_MOVEMENTS WHERE ROWKEY='\''robin'\'';"}'
```



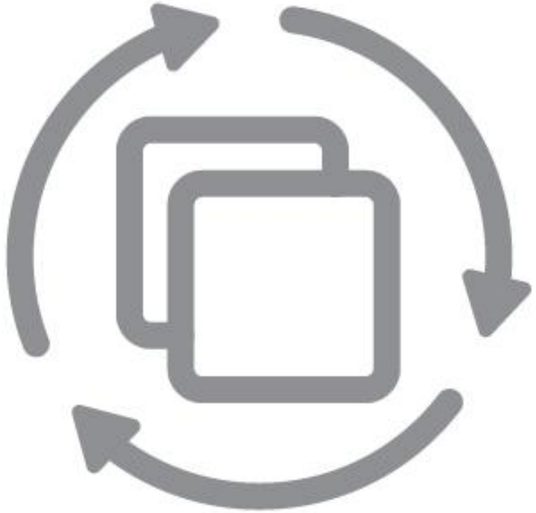
```
{"value":"3"}
```

Pull and Push queries in ksqlDB

Pull query



Push query



Tells you: Point in time value

All value changes

Exits: Immediately

Never

CONFLUENT

ksqlDB

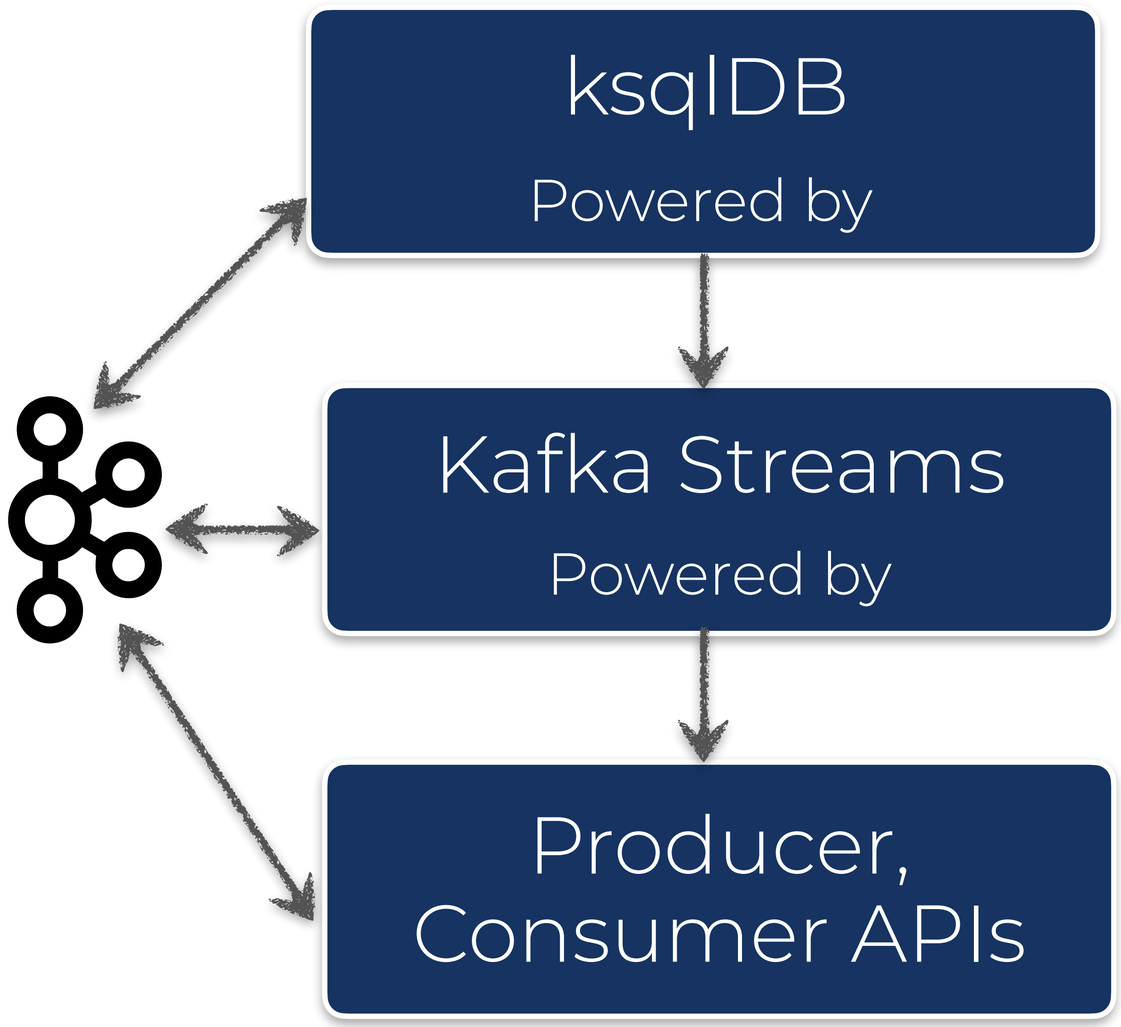
or

Kafka Streams?



Photo by Ramiz Dedaković on Unsplas

Standing on the Shoulders of Streaming Giants



Ease of use

Flexibility

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

ksqlDB UDFs

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

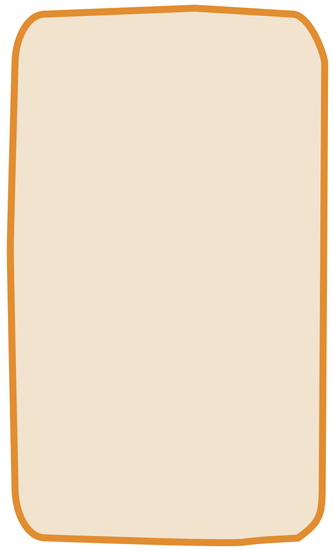
```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

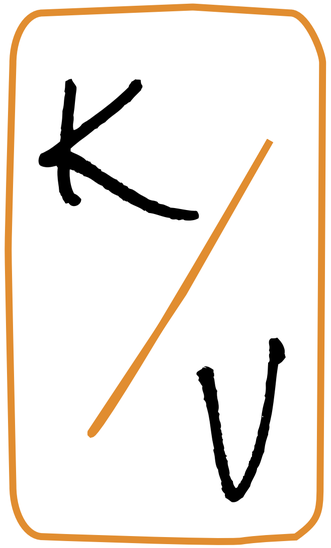


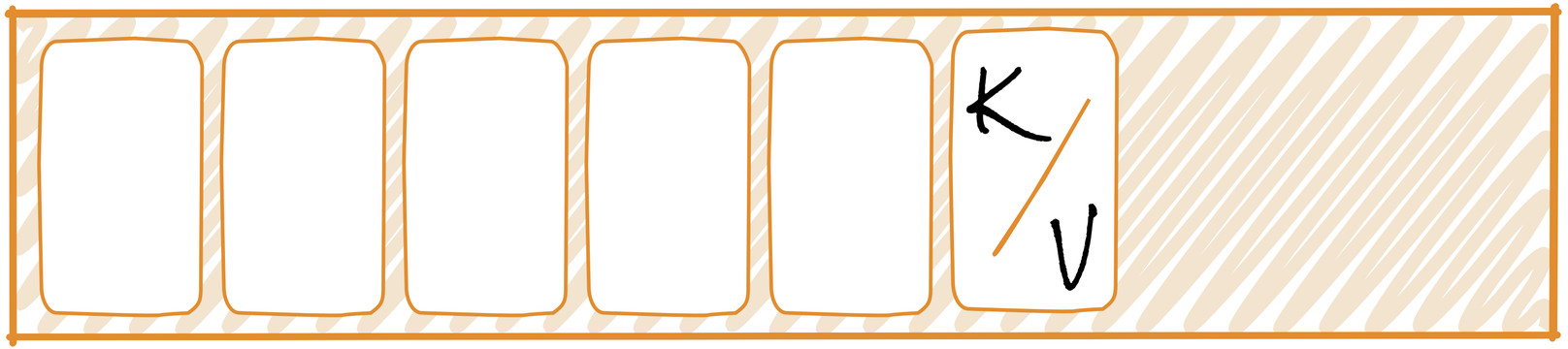
CONFLUENT

DEMO

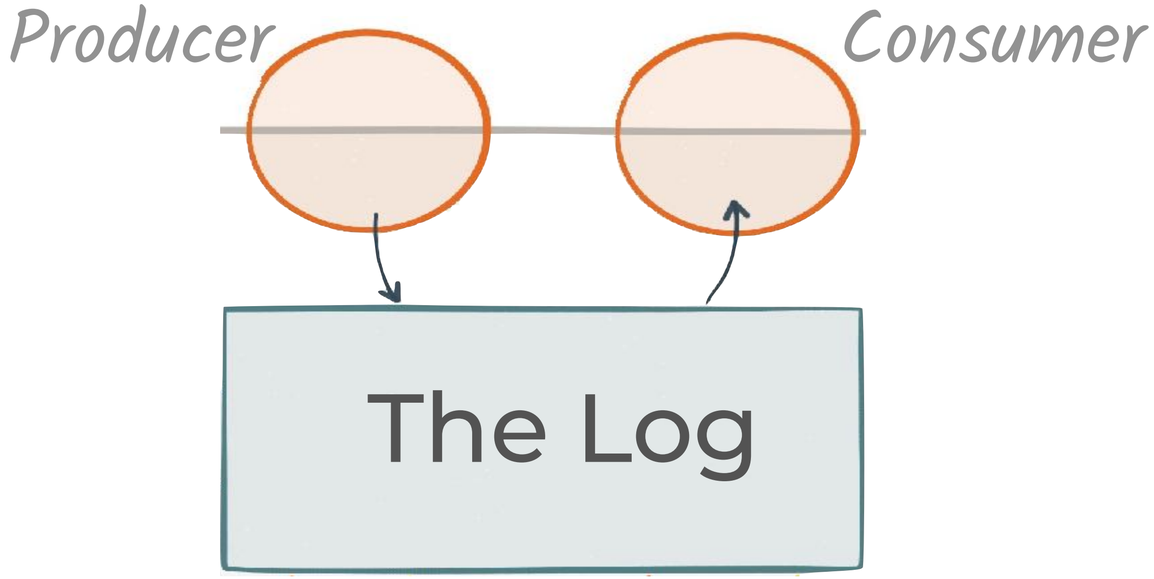
Summary

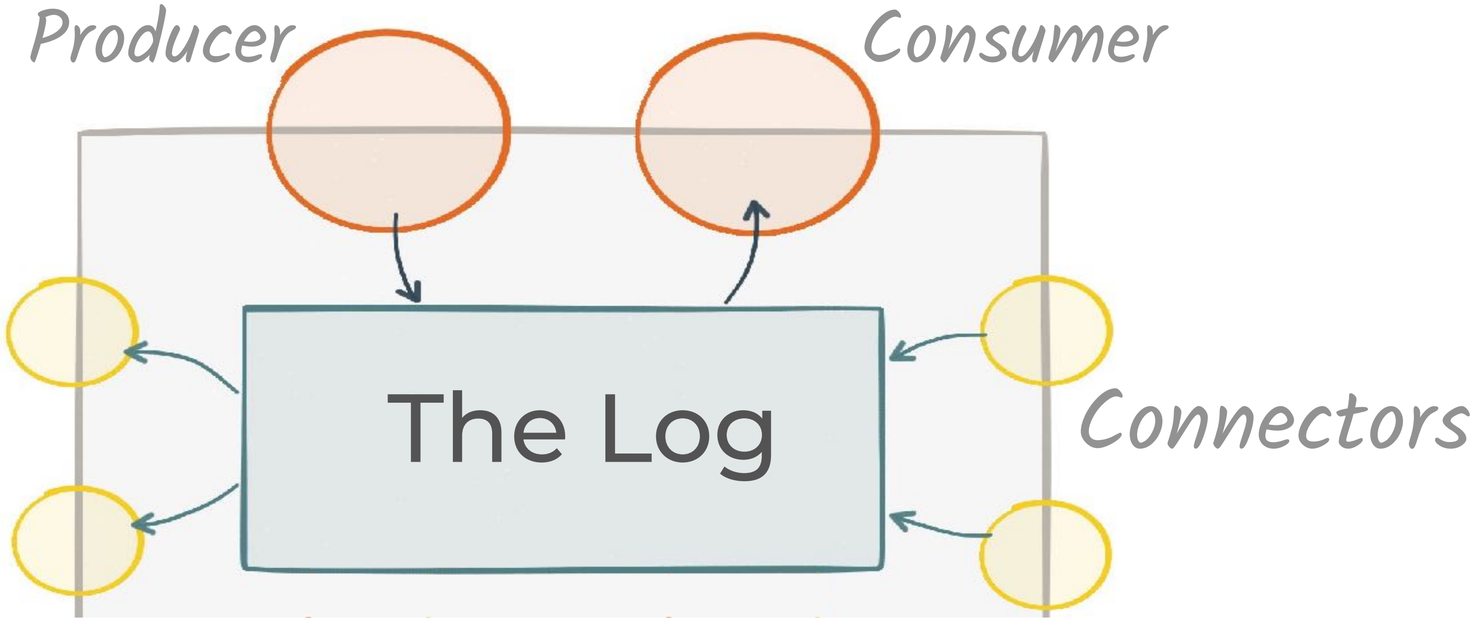


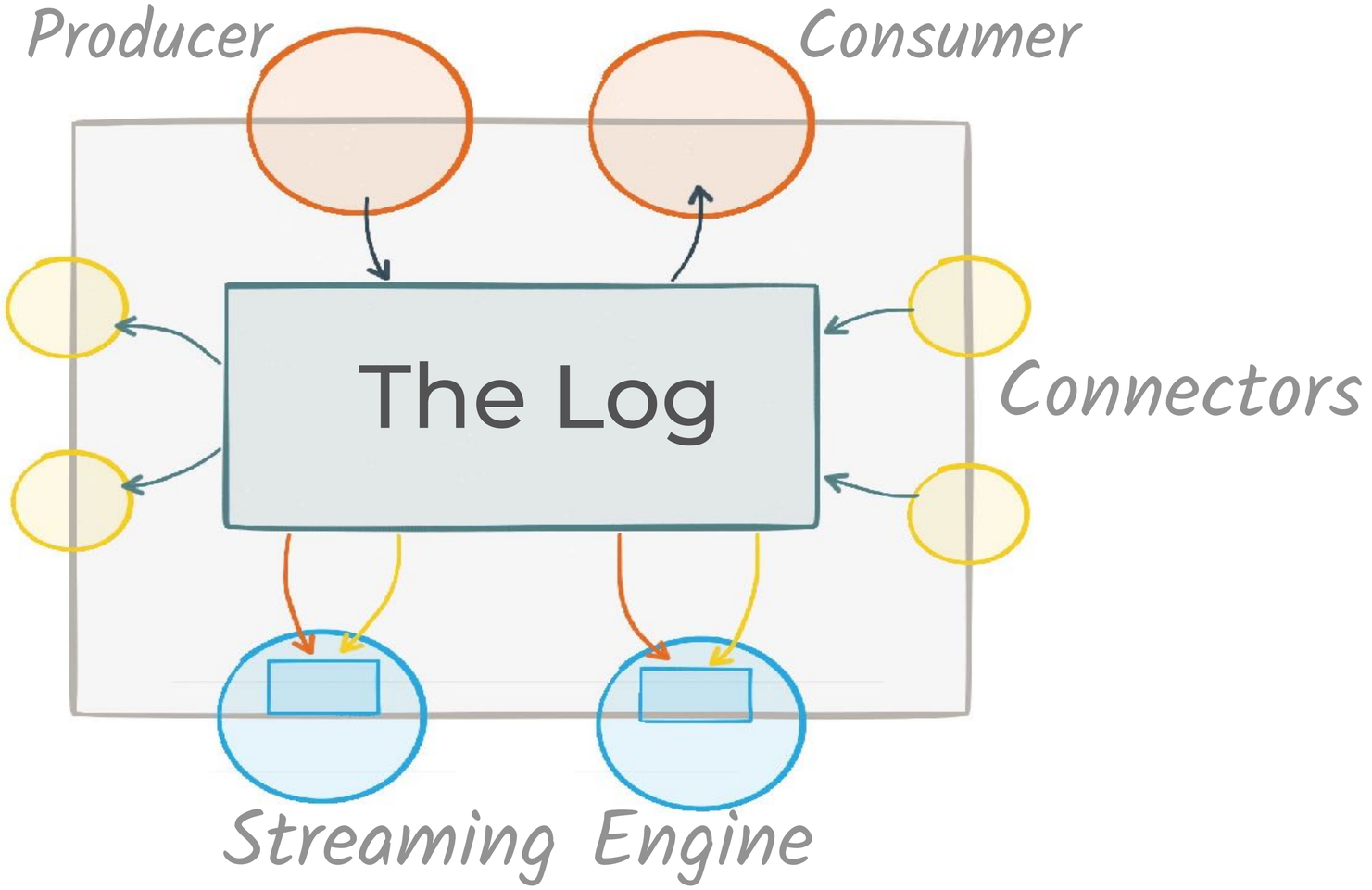

K
V


K
V

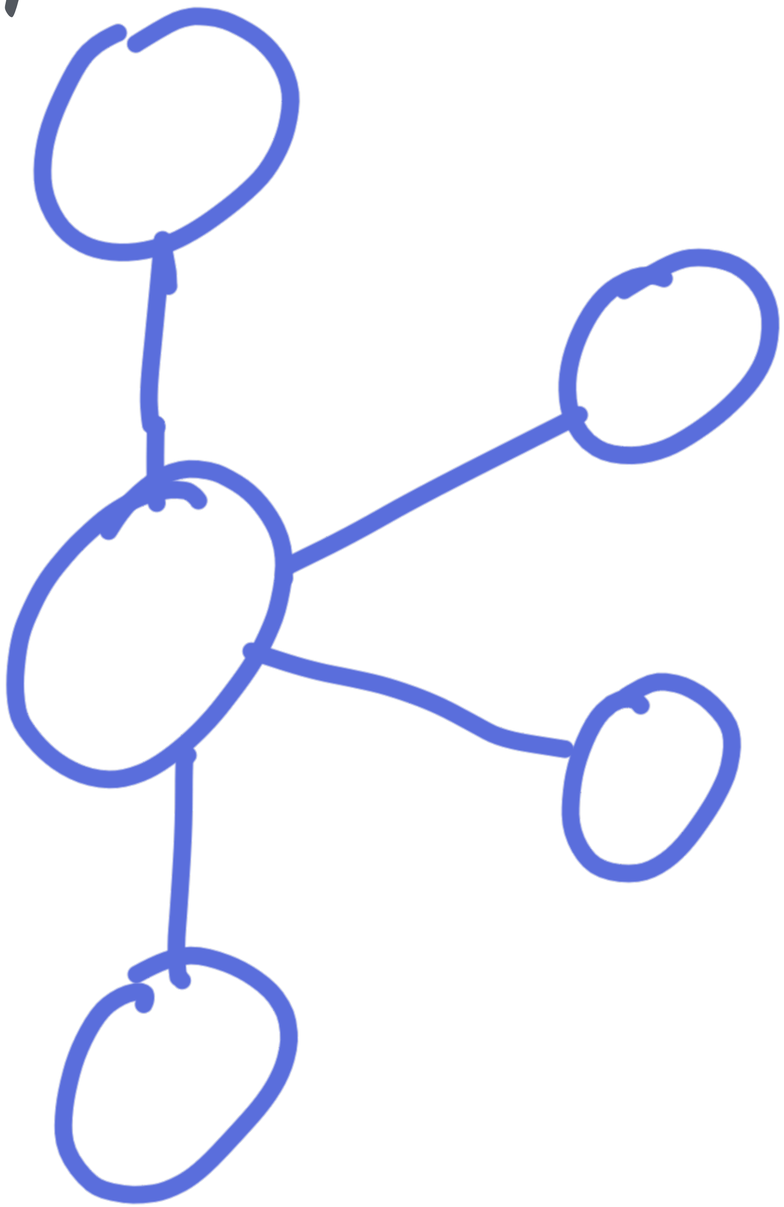
The Log

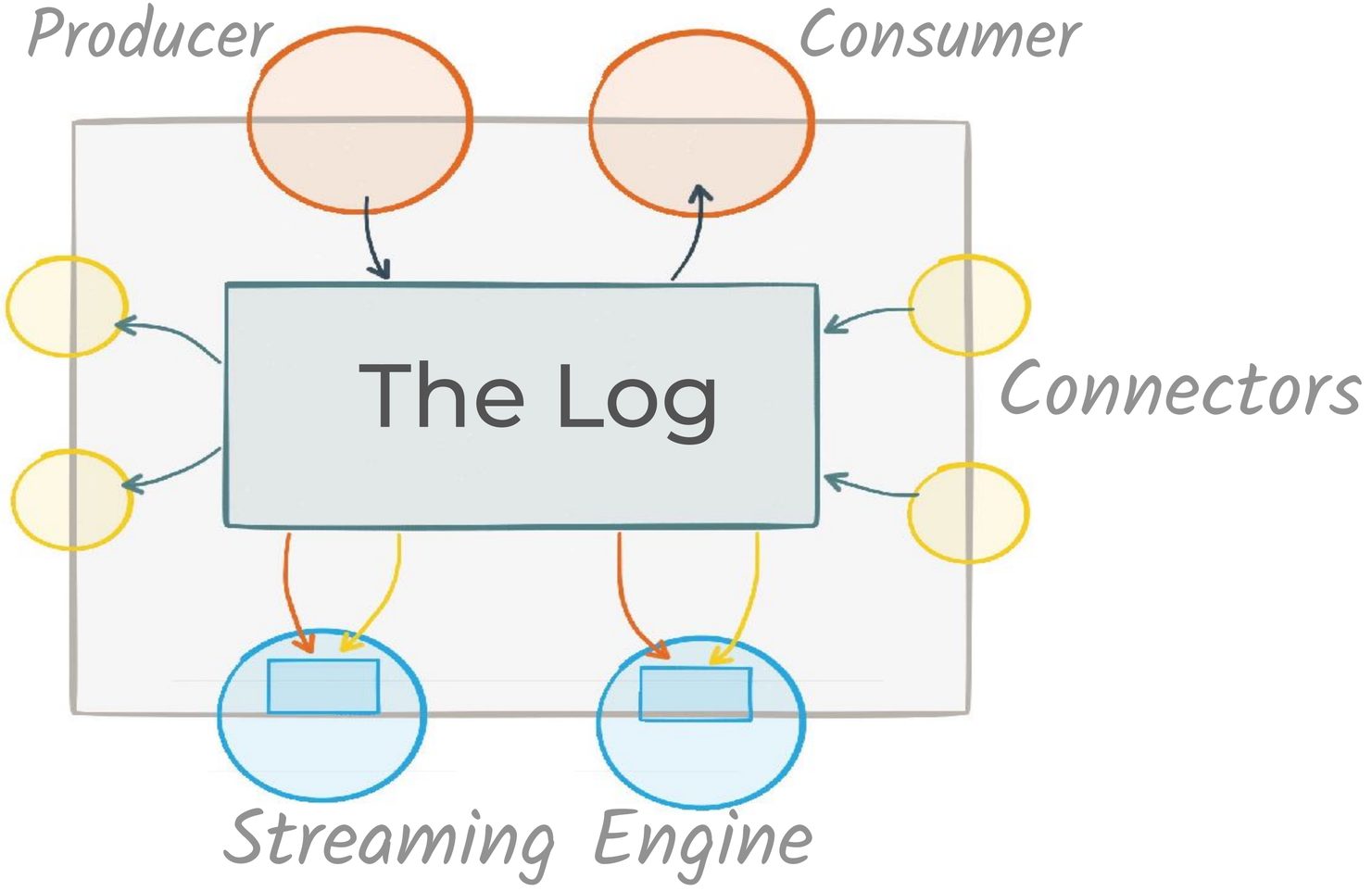

Producer
Consumer
The Log


Producer
Consumer
The Log
Connectors

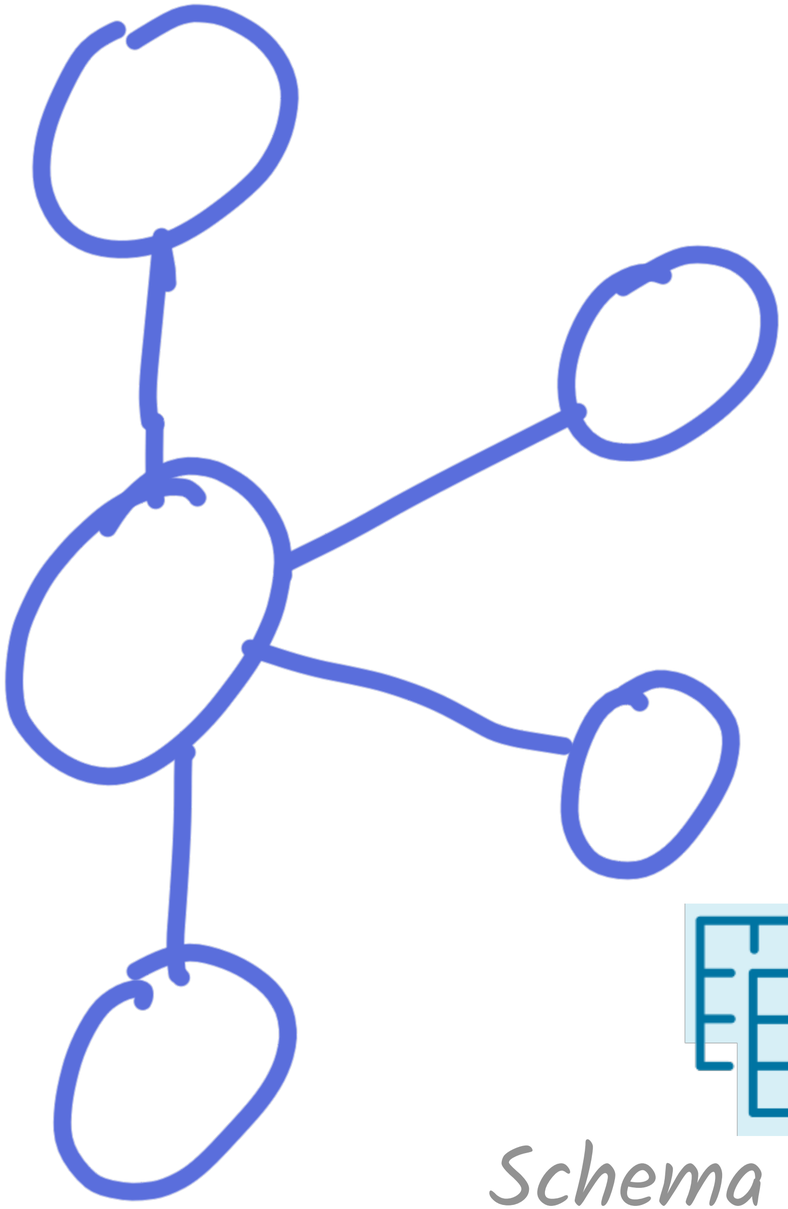

Producer
Consumer
The Log
Connectors
Streaming Engine

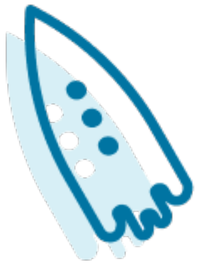
Apache Kafka



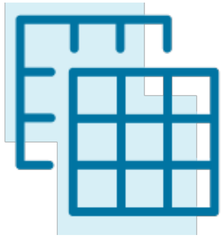


Confluent Platform

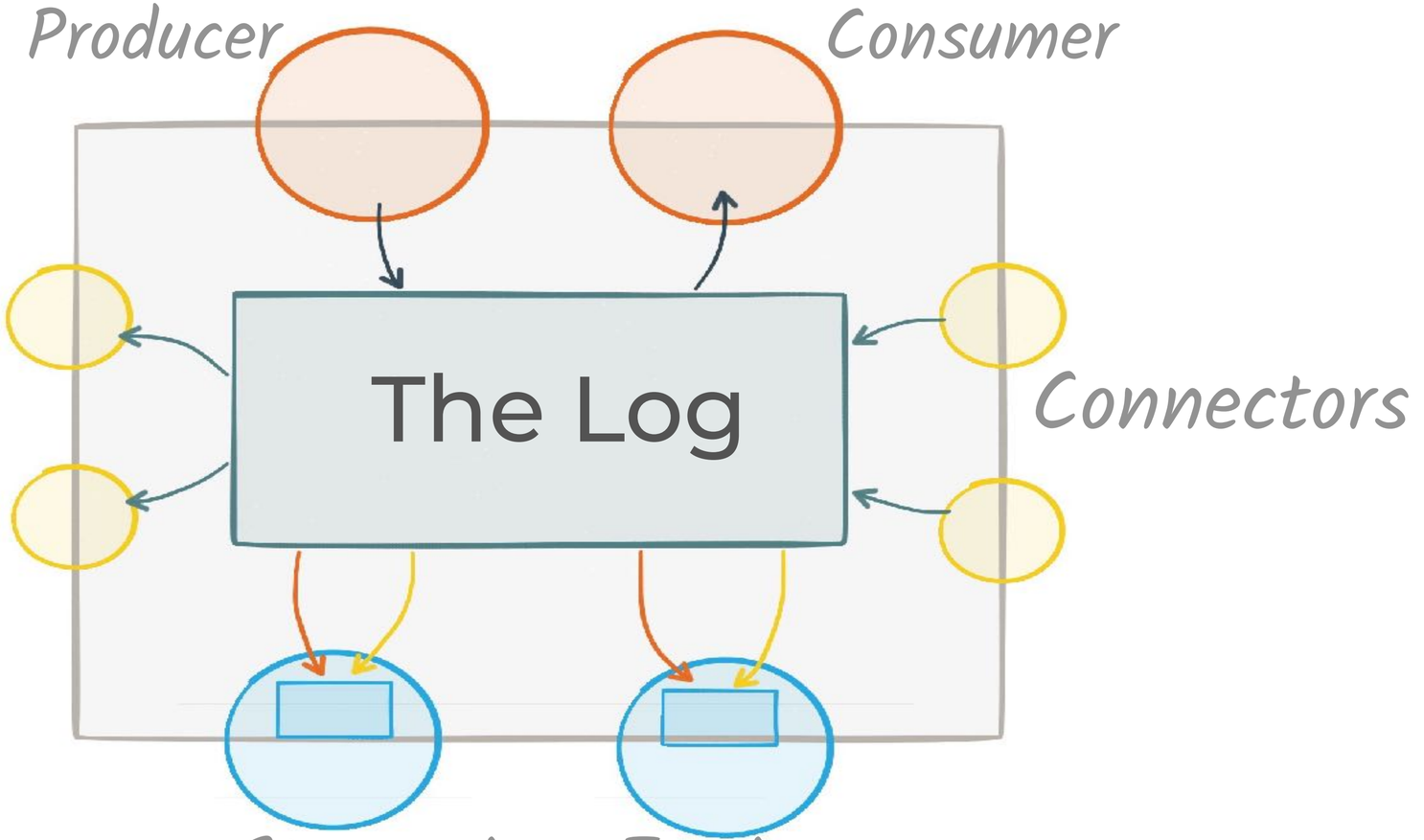




ksqlDB



Schema Registry



EVENTS

are

EVERYWHERE

EVENTS

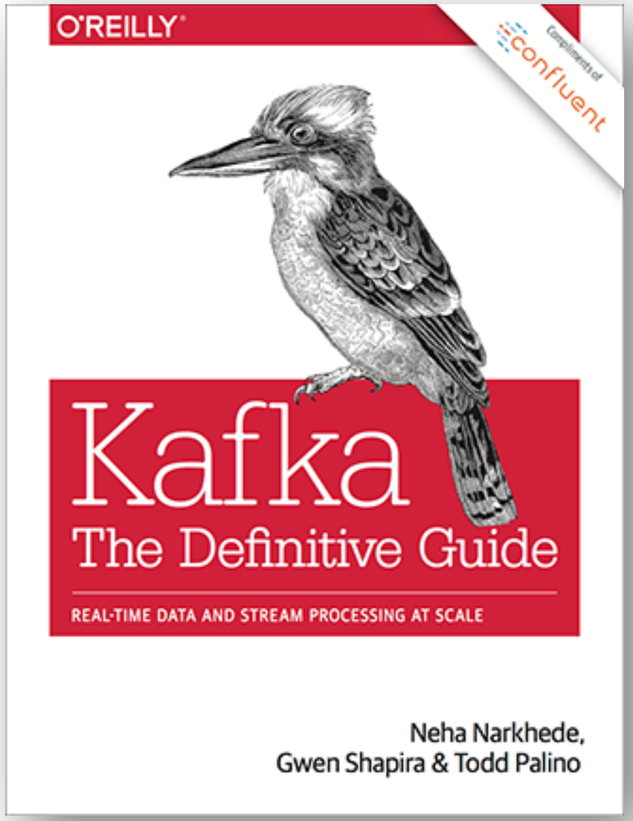
are

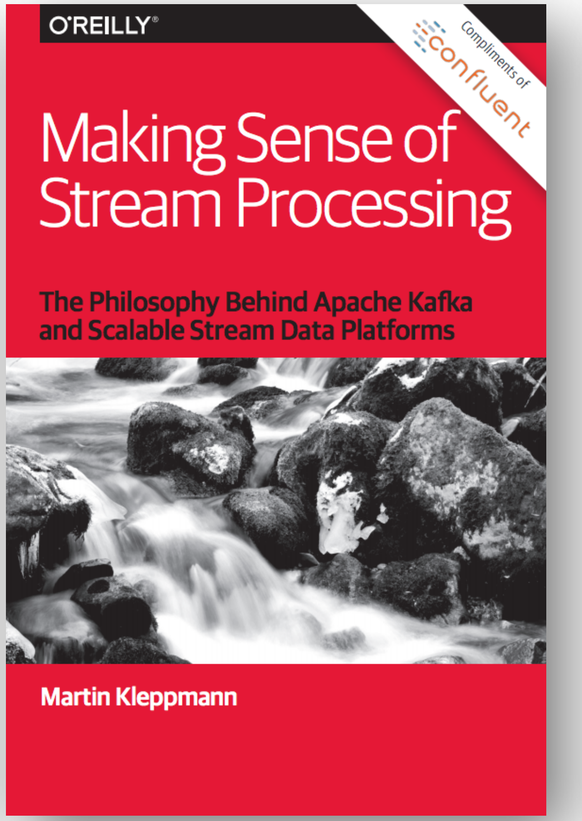
very POWERFUL

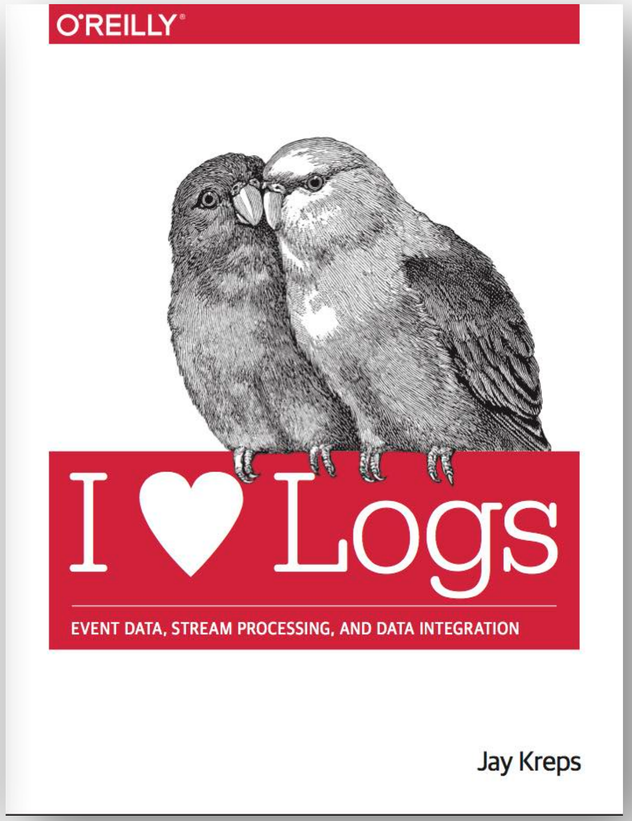
Standby for resource links...

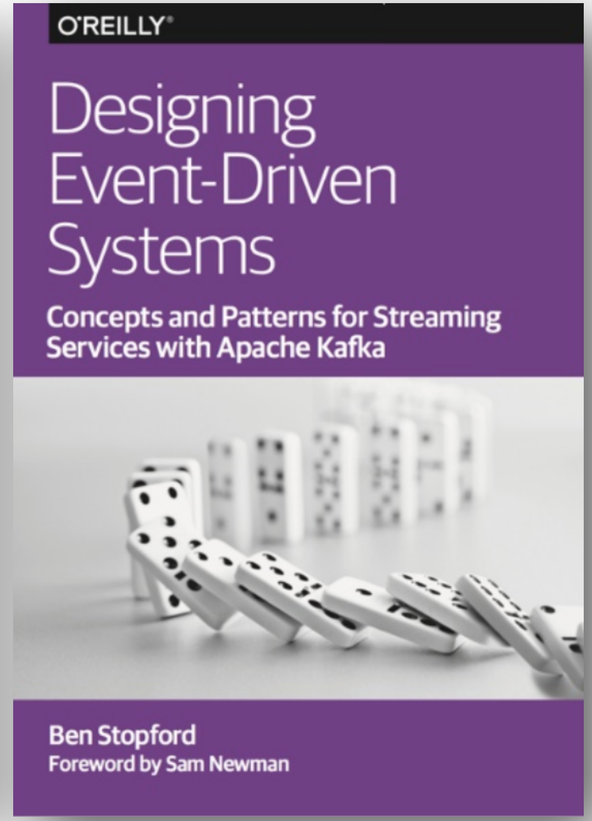
Free Books!

https://rmoff.dev/gotopia









60DEVADV

Free money!
(additional $60 towards your bill 😄)

$50 USD off your bill each calendar month for the first three months when you sign up

https://rmoff.dev/ccloud



confluent cloud

Fully Managed Kafka as a Service

* T&C: https://www.confluent.io/confluent-cloud-promo-disclaimer


Learn Kafka.
Start building with
Apache Kafka at
Confluent Developer.
developer.confluent.io

Confluent Community Slack group



cnfl.io/slack

#EOF

@rmoff rmoff.dev/talks youtube.com/rmoff