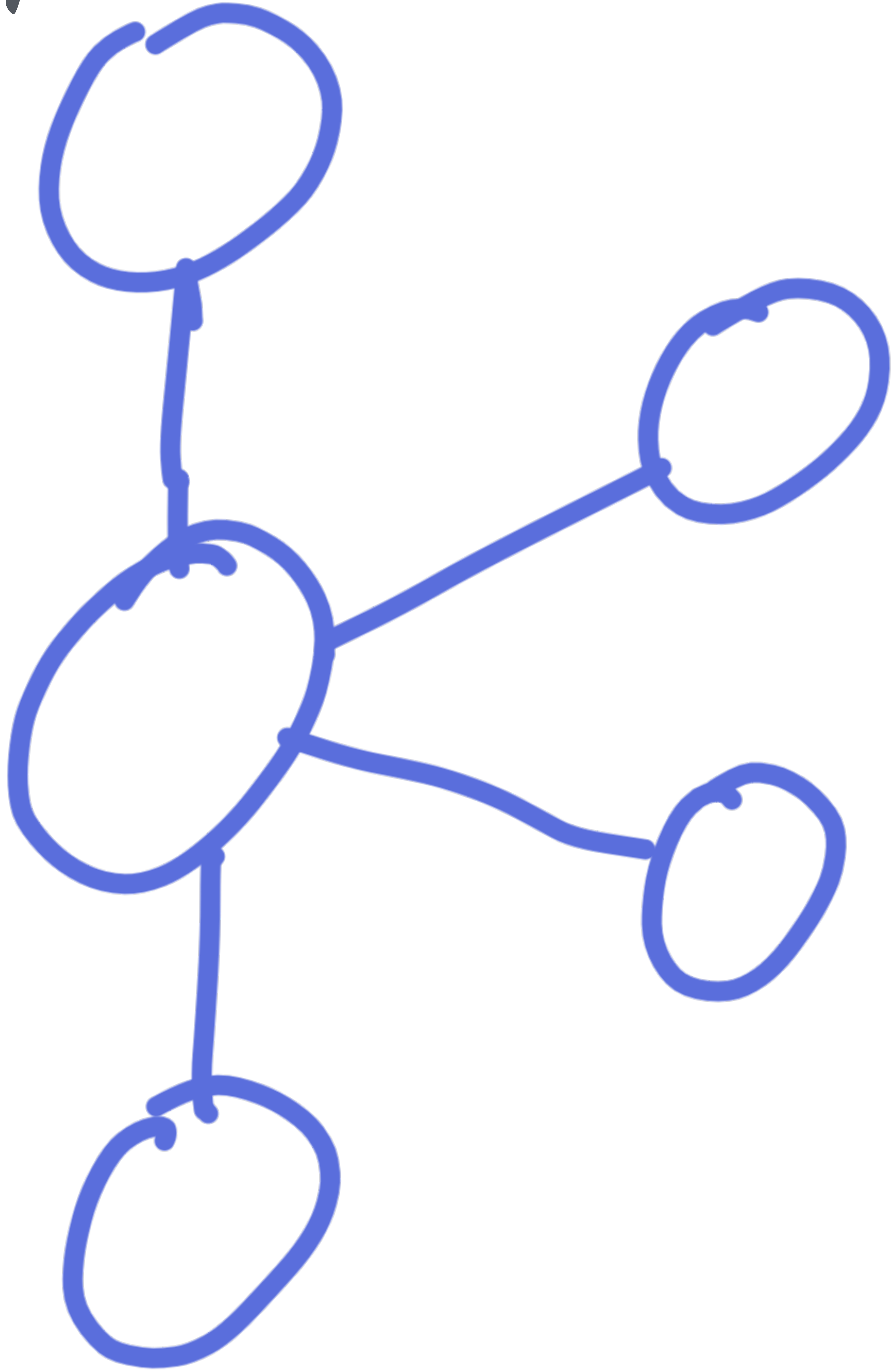
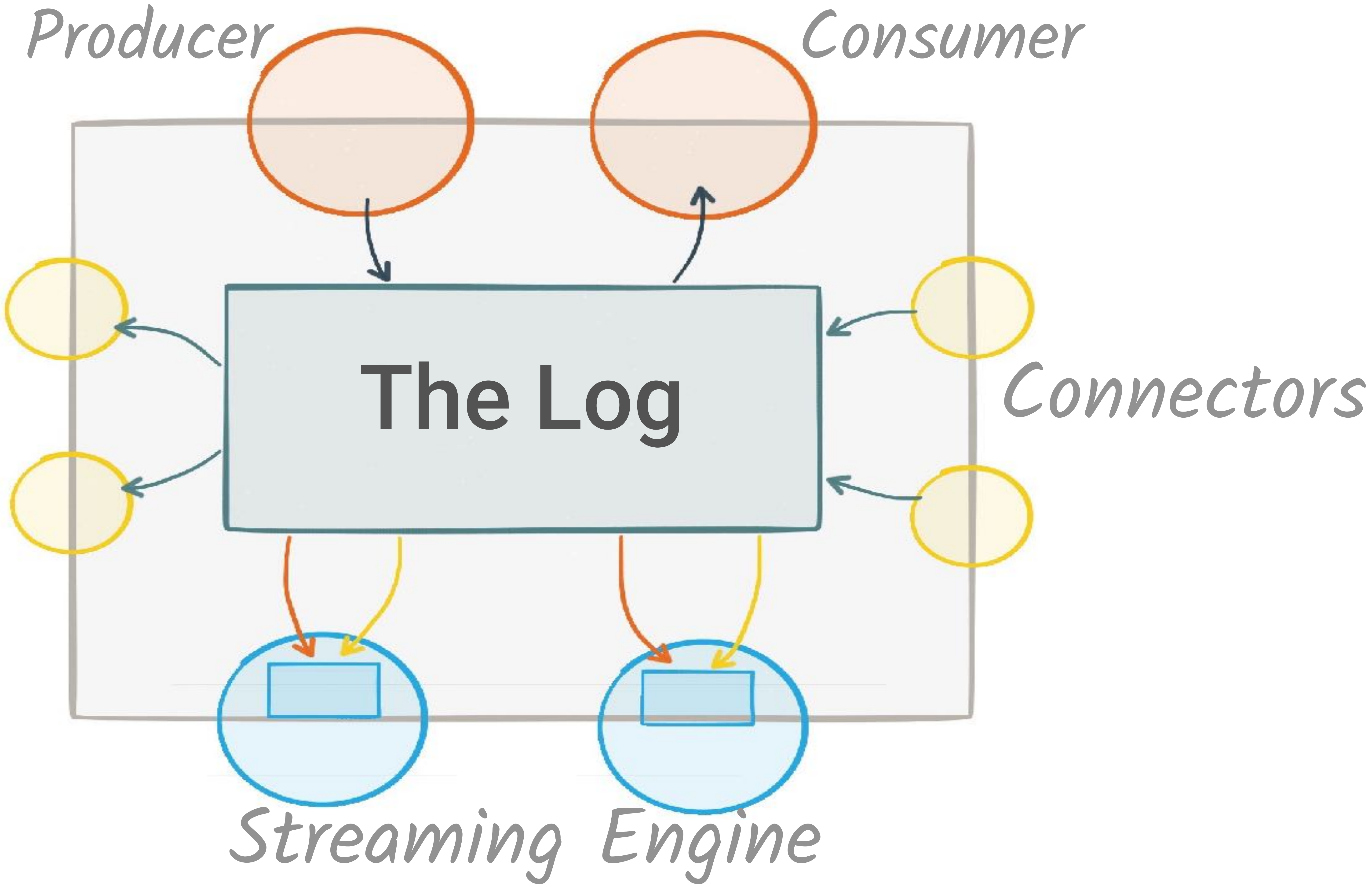
Introduction to ksqlDB

Robin Moffatt

#KafkaMeetup @rmoff

Apache Kafka






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

Streams of events

Stream Processing with ksqlDB

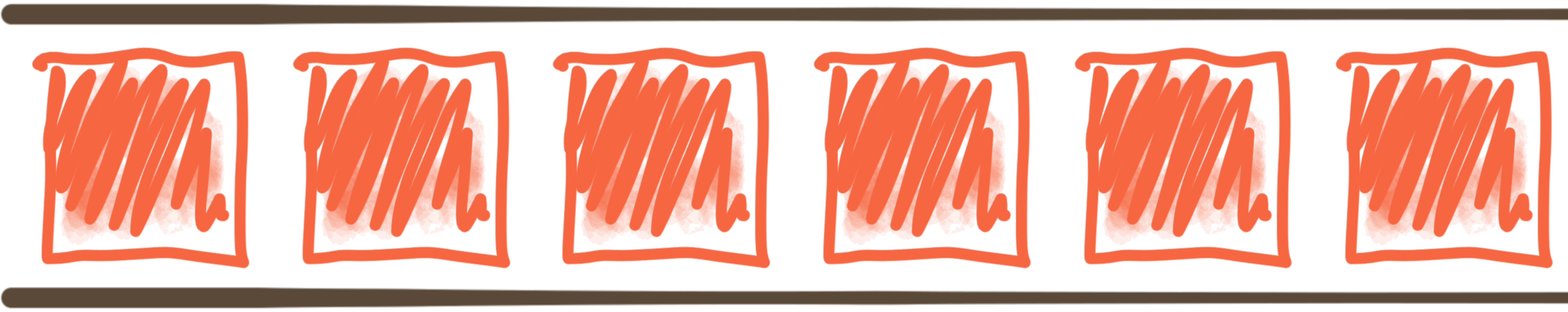
Stream Processing with Kafka Streams

@rmoff #KafkaMeetup

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

Stream Processing with ksqlDB

ksqlDB

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


{
"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

@rmoff #KafkaMeetup

```
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
```


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

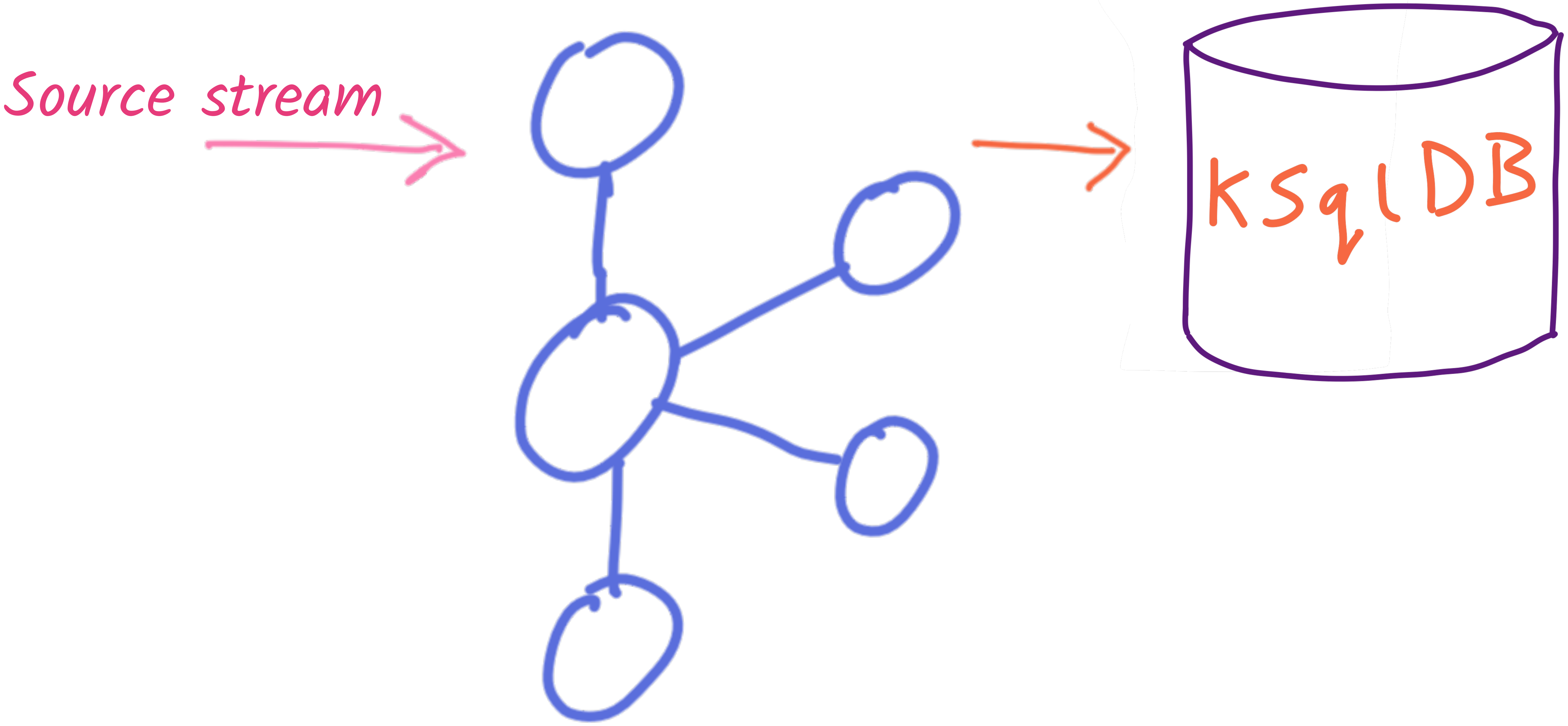

Object store,
data warehouse,
RDBMS

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

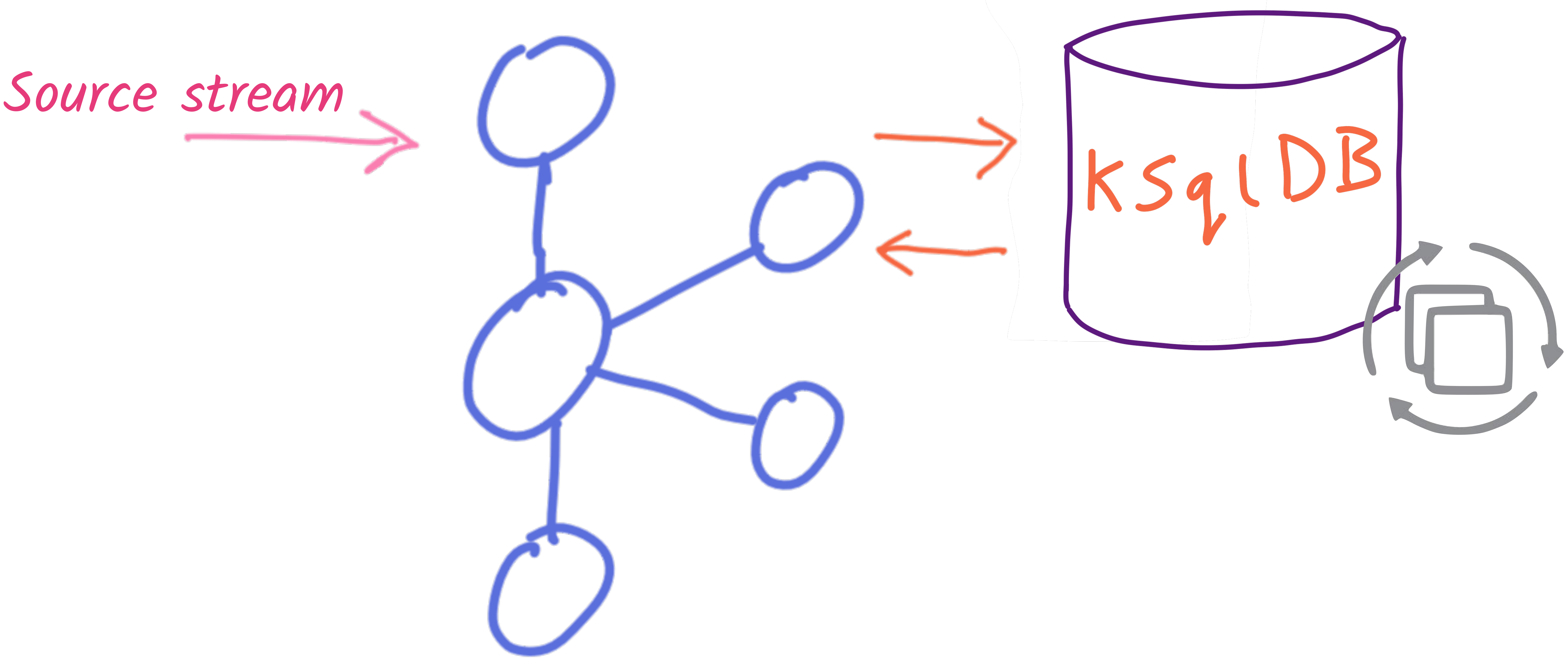
ksqlDB

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

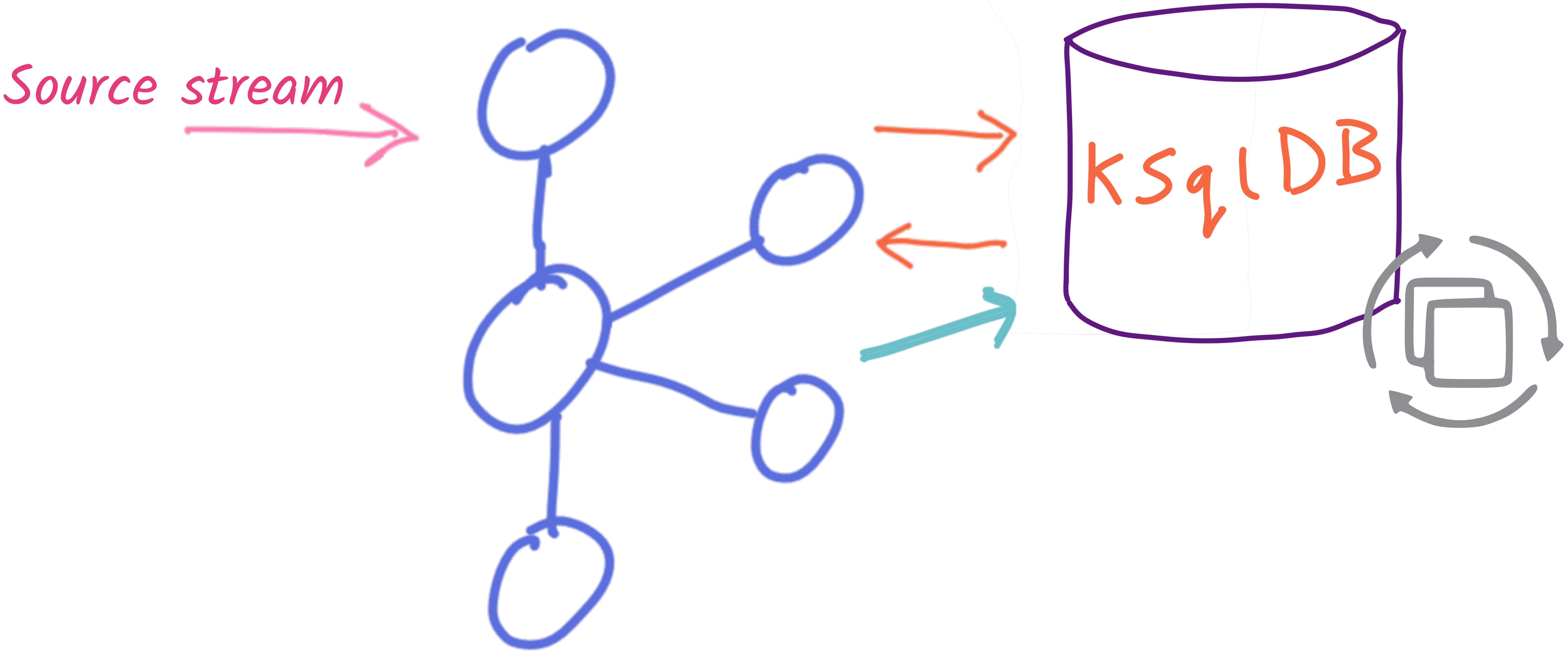
Stream Processing with ksqlDB



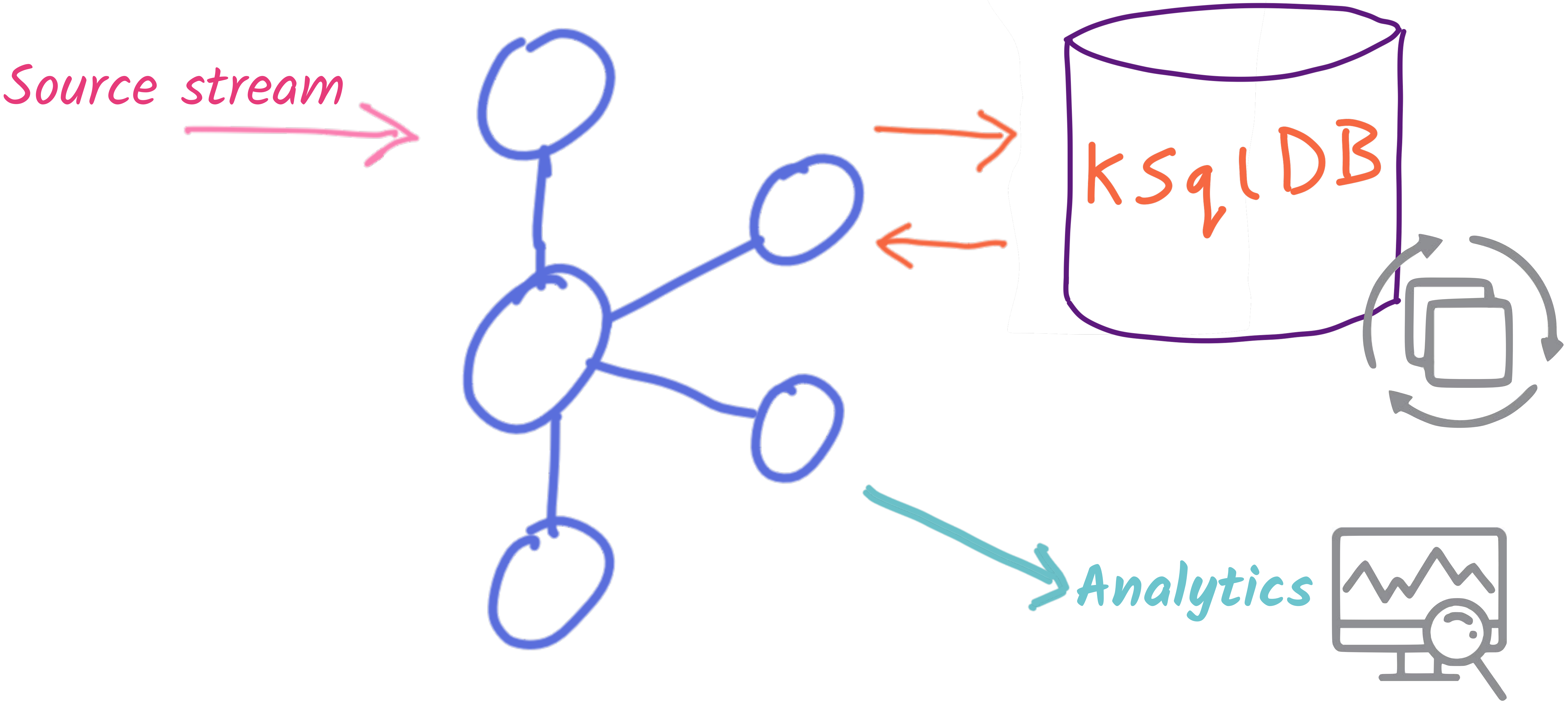
Stream Processing with ksqlDB



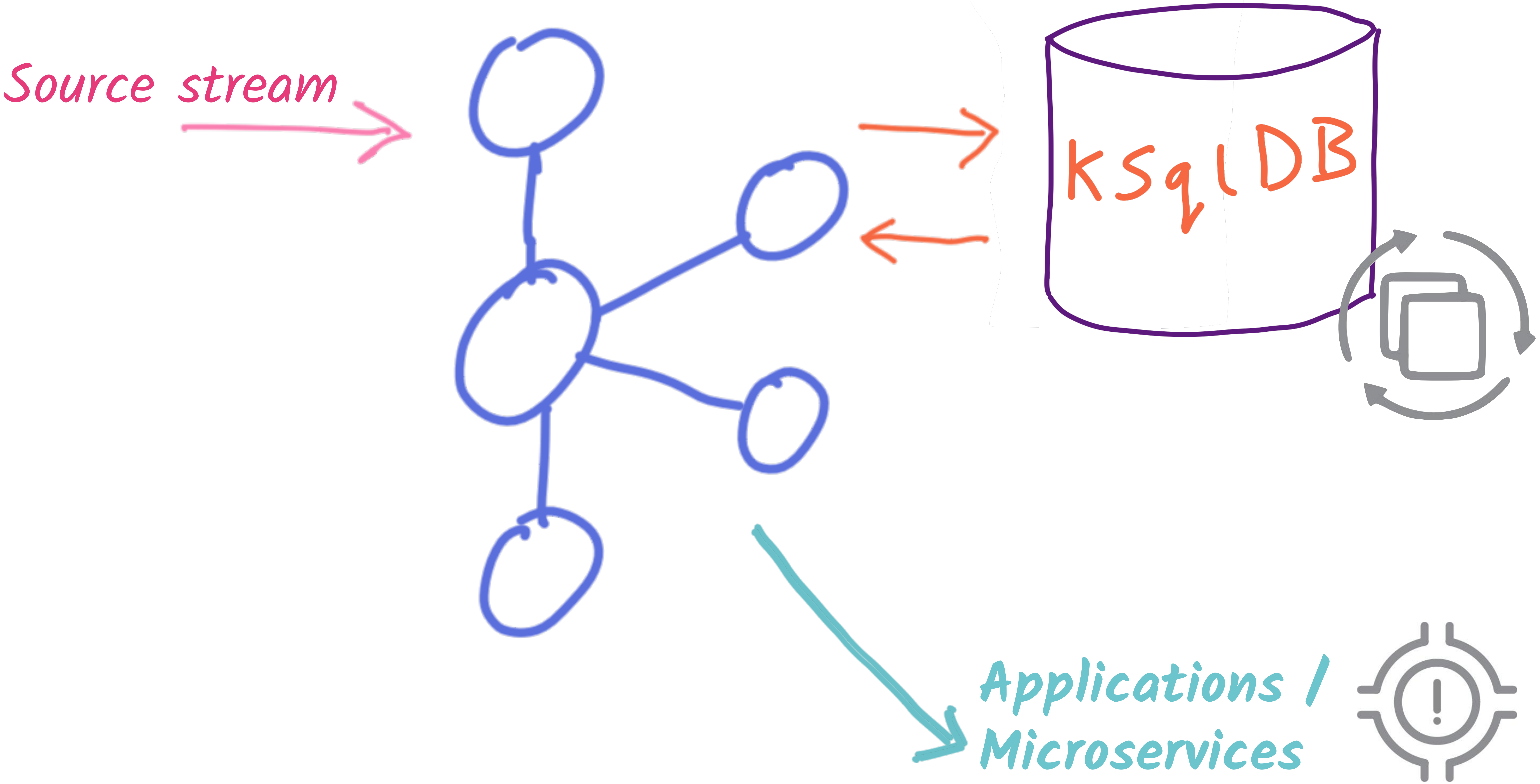
Stream Processing with ksqlDB



Stream Processing with ksqlDB



Stream Processing with ksqlDB



Stream Processing with ksqlDB

DEMO

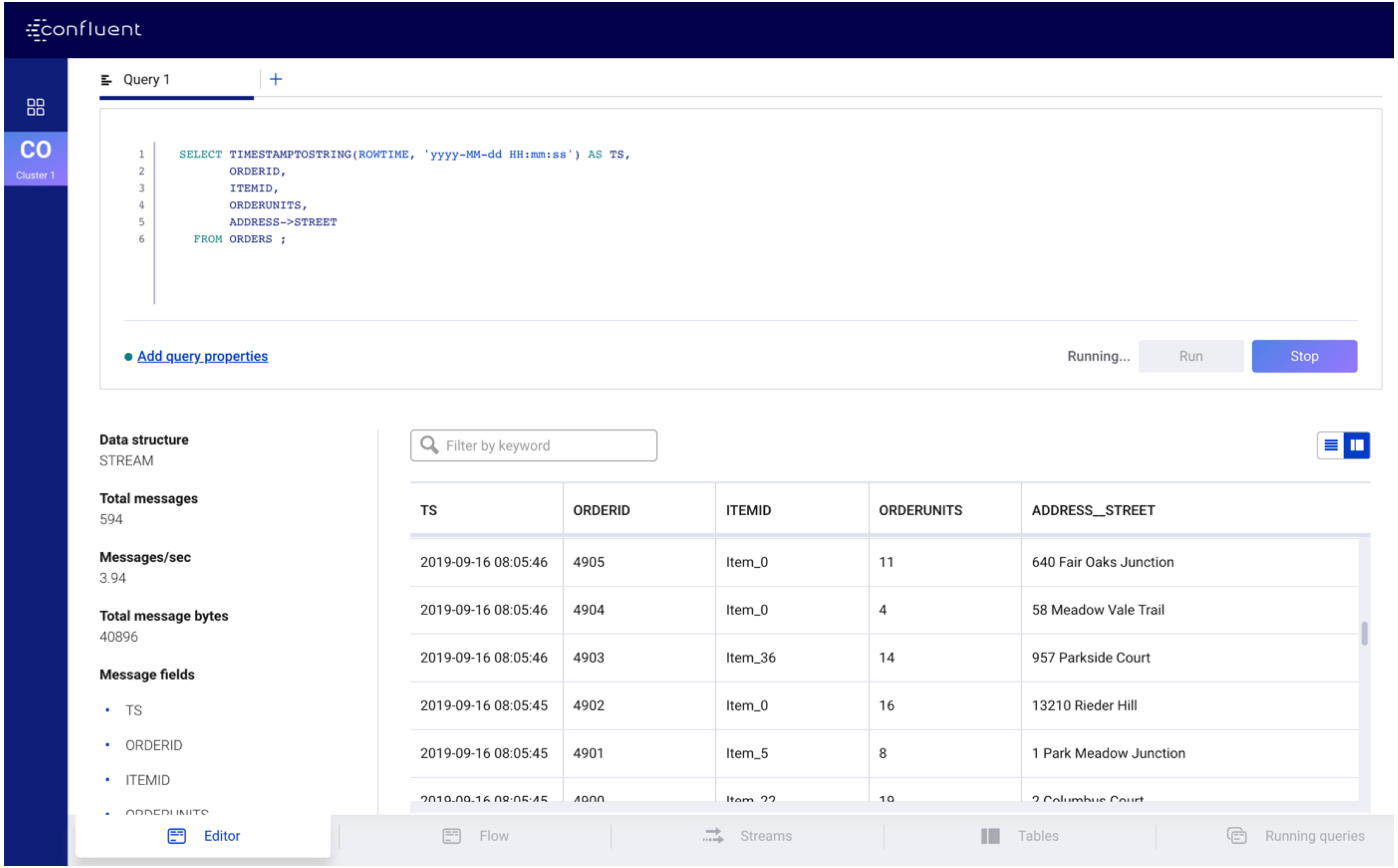
https://rmoff.dev/ksqldb-demo

Interacting with ksqlDB

Photo by Tim Mossholder on Unsplash

ksqlDB - Confluent Control Center

@rmoff #KafkaMeetup



ksqlDB - REST API

@rmoff #KafkaMeetup

```
$ echo '{"ksql":"SELECT ORDERID, ITEMID, ADDRESS FROM ORDERS LIMIT 5;", "streamsProperties":
        "ksql.streams.auto.offset.reset": "earliest"}}' | \
  http http://localhost:8088/query
HTTP/1.1 200 OK
Content-Encoding: gzip
Content-Type: application/json
Date: Tue, 02 Jul 2019 12:46:25 GMT
Server: Jetty(9.4.14.v20181114)
Transfer-Encoding: chunked
Vary: Accept-Encoding, User-Agent

{"row":{"columns":[0,"Item_0",{"STREET":"377 Maryland Place","CITY":"Beaumont","STATE":"Texas
l,"terminal":false}
{"row":{"columns":[0,"Item_0",{"STREET":"072 Butternut Lane","CITY":"Grand Junction","STATE":
sage":null,"terminal":false}
{"row":{"columns":[1,"Item_0",{"STREET":"703 Hoffman Place","CITY":"Mountain View","STATE":"C
sage":null,"terminal":false}
{"row":{"columns":[2,"Item_0",{"STREET":"0 Dorton Circle","CITY":"Brooklyn","STATE":"New York
l,"terminal":false}
{"row":{"columns":[3,"Item_0",{"STREET":"404 Mayer Park","CITY":"Lubbock","STATE":"Texas"}]},
rminal":false}
{"row":null,"errorMessage":null,"finalMessage":"Limit Reached","terminal":true}
```

ksqlDB - Native client (coming soon)

Branch: master ksql / design-proposals / klip-15-new-api-and-client.md Find file Copy path

purplefox docs: Bring KLIP-15 up to date (#4499) ddf0701 7 hours ago

1 contributor

331 lines (235 sloc) 14.9 KB Raw Blame History

KLIP 15 - ksqlDB Client and New Server API

Author: Tim Fox (purplefox) | **Release Target**: ? | **Status**: *In Discussion* | **Discussion**:

Please read the initial discussion on the ksqlDB developer group first:

https://groups.google.com/forum/#!topic/ksql-dev/yxcRlsOsNmo

And there is a partially working prototype for this work described here:

https://groups.google.com/forum/#!topic/ksql-dev/5mLKvtZFs4Y

For ksqlDB to be a successful project we need to provide an awesome out of the box developer experience and it should be super easy and fun to write powerful event streaming applications. Currently this is somewhat diffficult to do, and it requires the use of multiple clients and multiple moving parts in order to build a simple event sourcing / CQRS style application. We should provide a new ksqlDB client that provides that currently missing single interaction point with the event streaming platform so that we can improve the application development process. To support the client we should provide an updated HTTP2 based API that allows streaming use cases to be handled better.

What else can ksqlDB do?

Photo by Sereja Ris on Unsplash

Message transformation with ksqlDB

ORDERS


```
{ "ordertime": 1560070133853,
  "orderid": 7,
  "itemid": "Item_9",
  "orderunits": 5,
  "address": {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```


Convert this timestamp to human-readable format

Drop these address fields

Message transformation with ksqlDB

ORDERS


```json
{ "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address": {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
```


ksqlDB

```sql
CREATE STREAM ORDERS_NO_ADDRESS_DATA AS
  SELECT ORDERTIME, ORDERID, ITEMID, ORDERUNITS
    FROM ORDERS;
```

Message transformation with ksqlDB

@rmoff #KafkaMeetup

ORDERS

```
{ "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address": {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
```

ksqlDB

```sql
CREATE STREAM ORDERS_NO_ADDRESS_DATA AS
  SELECT TIMESTAMPTOSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss')
         AS ORDER_TIMESTAMP,
         ORDERID, ITEMID, ORDERUNITS
    FROM ORDERS;
```

ORDERS_NO_ADDRESS_DATA

```
{ "order_ts": "2020-02-14 15:10:58",
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5
}
```

Lookups and Joins with ksqlDB

ORDERS


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

Lookups and Joins with ksqlDB

@rmoff #KafkaMeetup

Lookup

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

@rmoff #KafkaMeetup

ITEMS

ORDERS

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

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

```sql
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 ;
```

Lookups and Joins with ksqlDB

@rmoff #KafkaMeetup

Lookup

ITEMS

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

ORDERS

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

ksqlDB

```
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
}
```

@rmoff #KafkaMeetup

Connecting ksqlDB to other systems

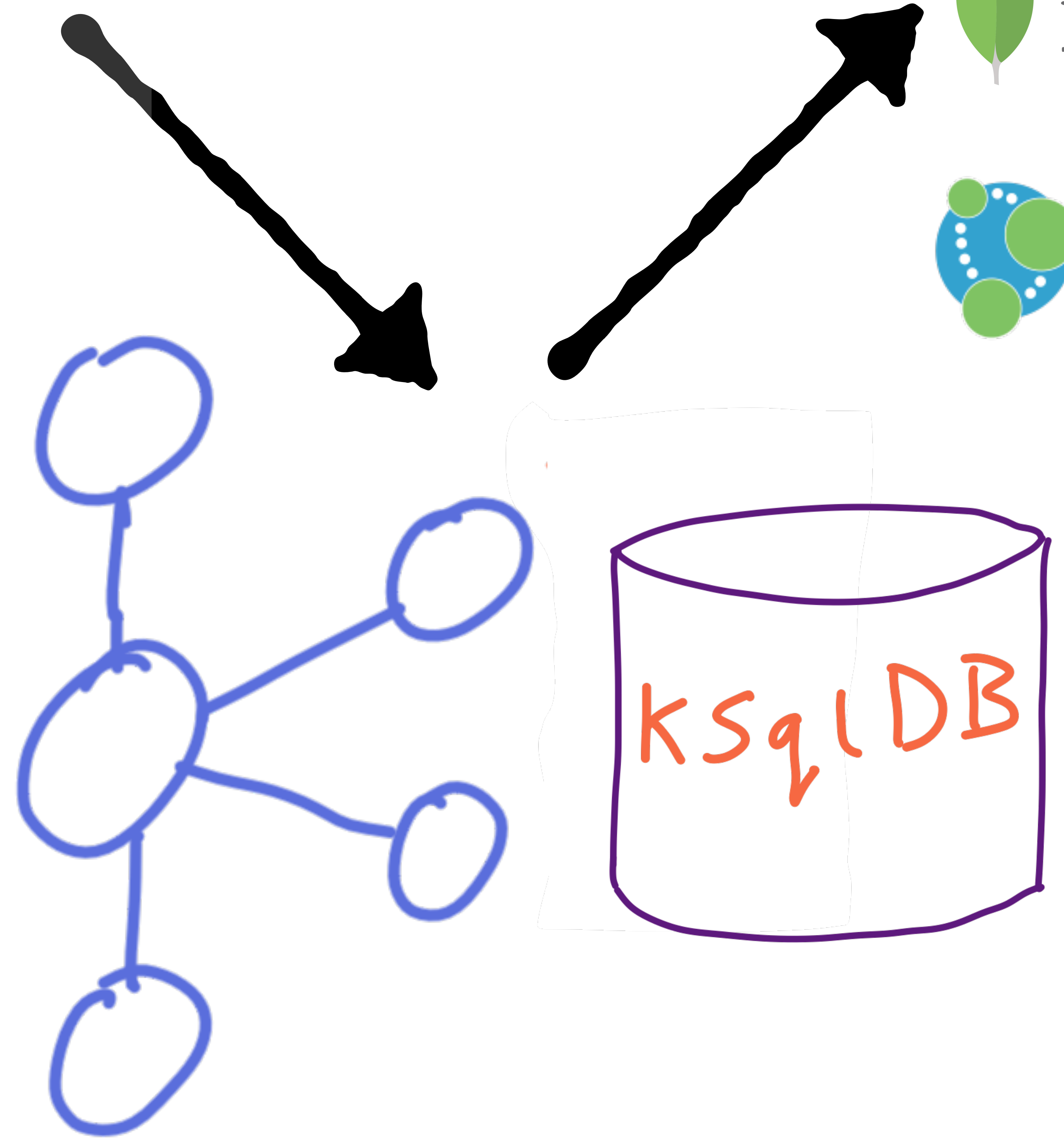
Photo by Mak on Unsplash

confluent

Connecting ksqlDB to other systems

syslog

http://



Google BigQuery

http://

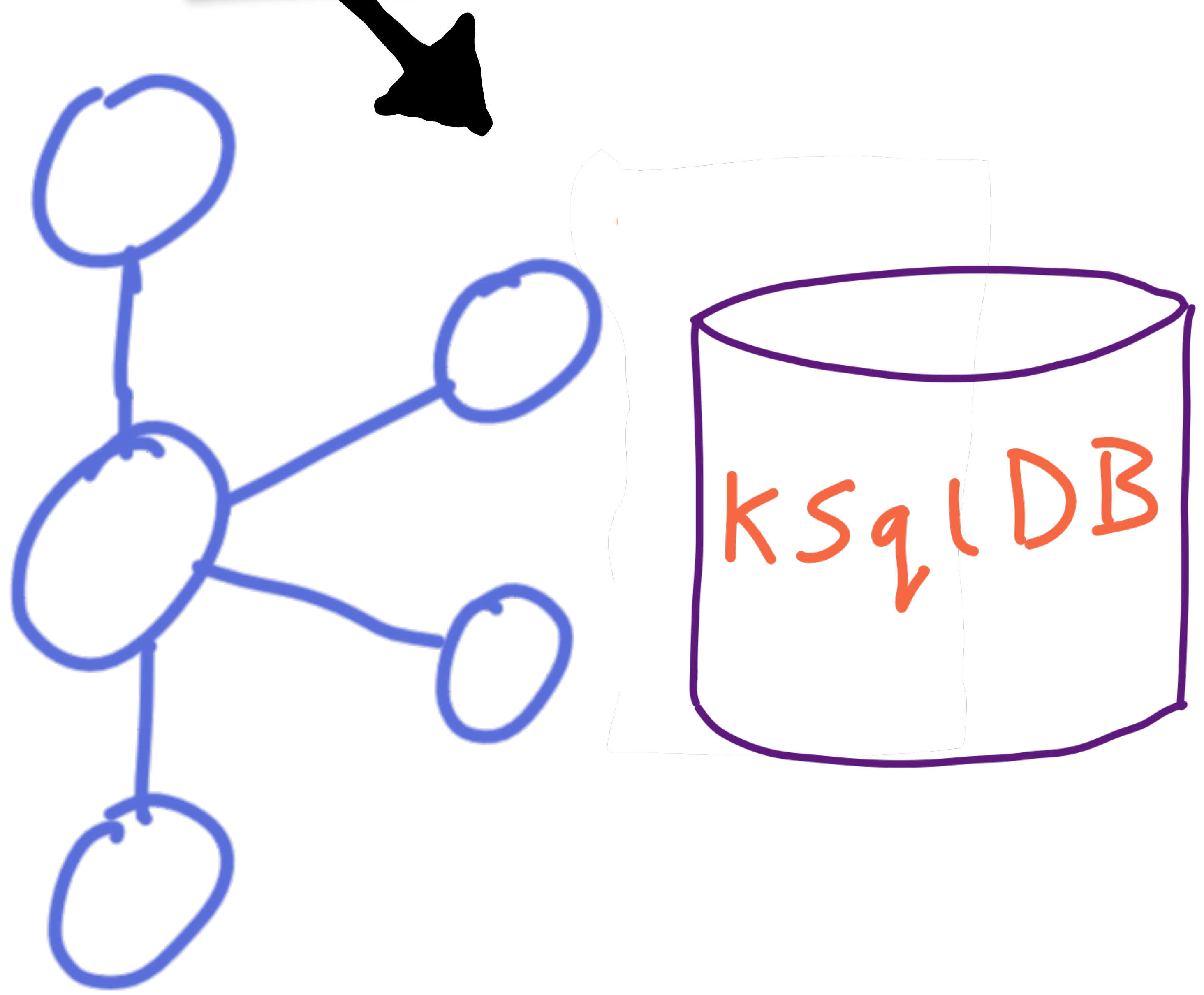
Amazon S3

Connecting ksqlDB to other systems

syslog

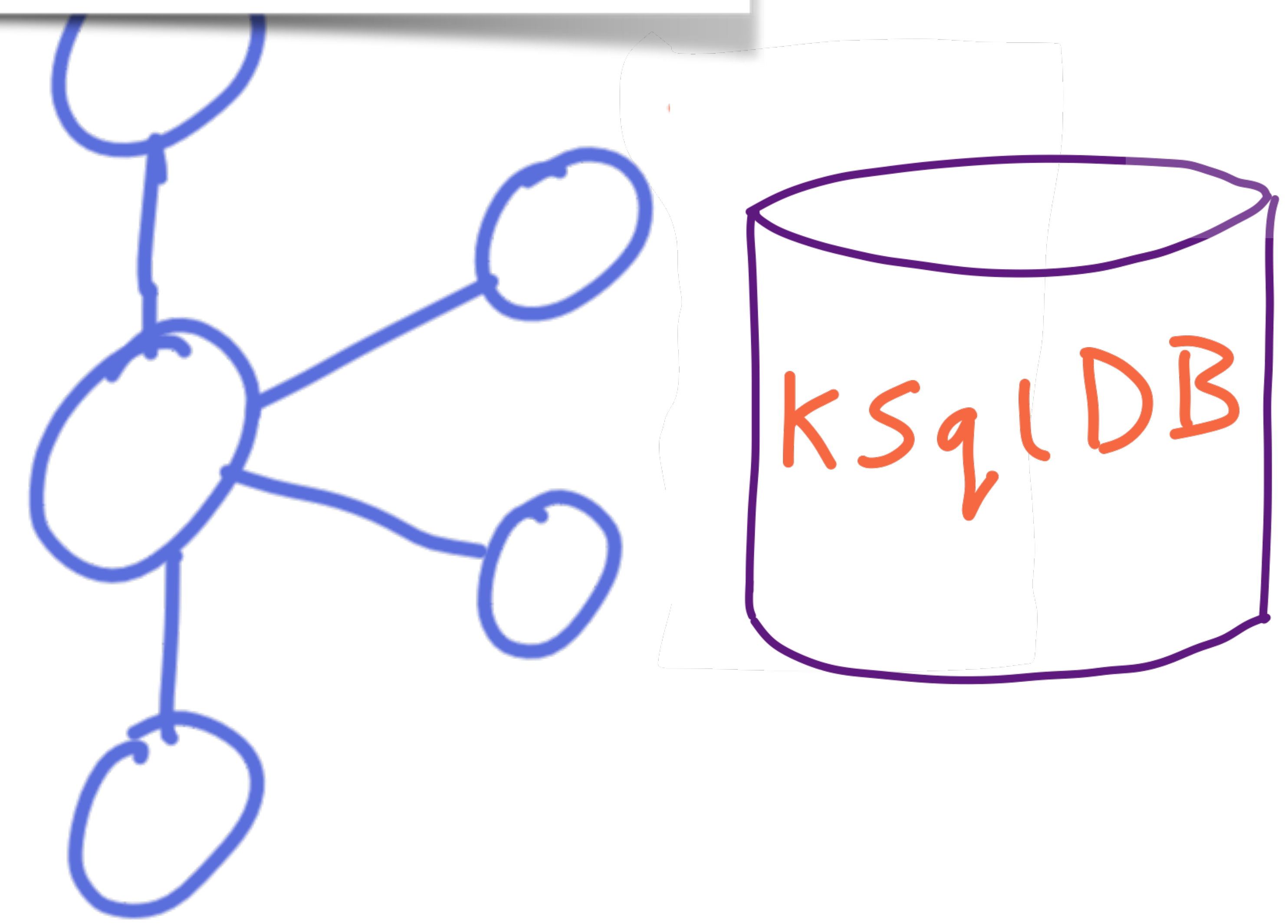
http://

```
CREATE SOURCE CONNECTOR
SOURCE_MYSQL_01 WITH (
 'connector.class' =
'i.d.c.mysql.MySqlConnector',
 'database.hostname' = 'mysql',
 'table.whitelist' = 'demo.customers');
```



Connecting ksqlDB to other systems

```
CREATE SINK CONNECTOR
SINK_ELASTIC_01 WITH (
 'connector.class' =
'…ElasticsearchSinkConnector',
 'connection.url' =
'http://elasticsearch:9200',
 'topics' = 'orders');
```



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

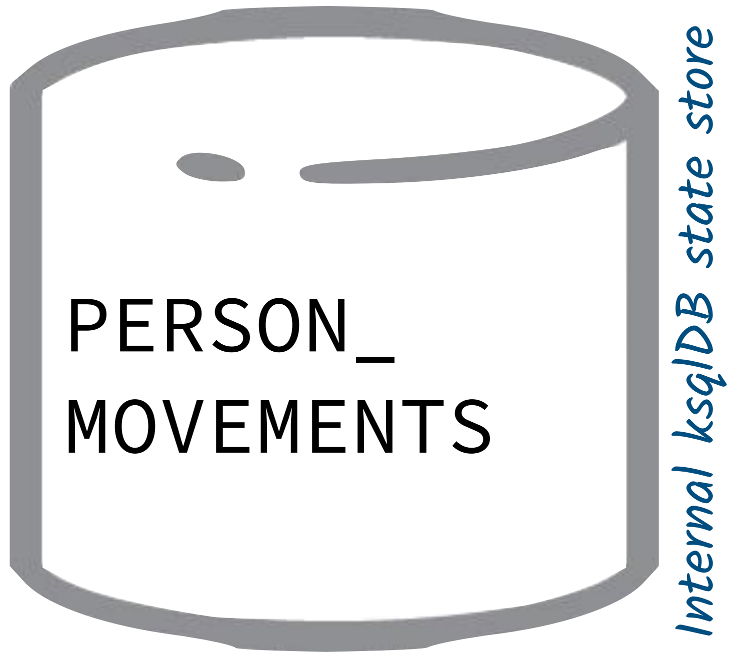
```
{ "event_ts": "2020-02-17T15:22:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

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

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

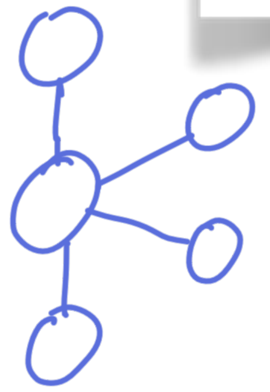
```
{ "event_ts": "2020-02-18T09:00:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

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



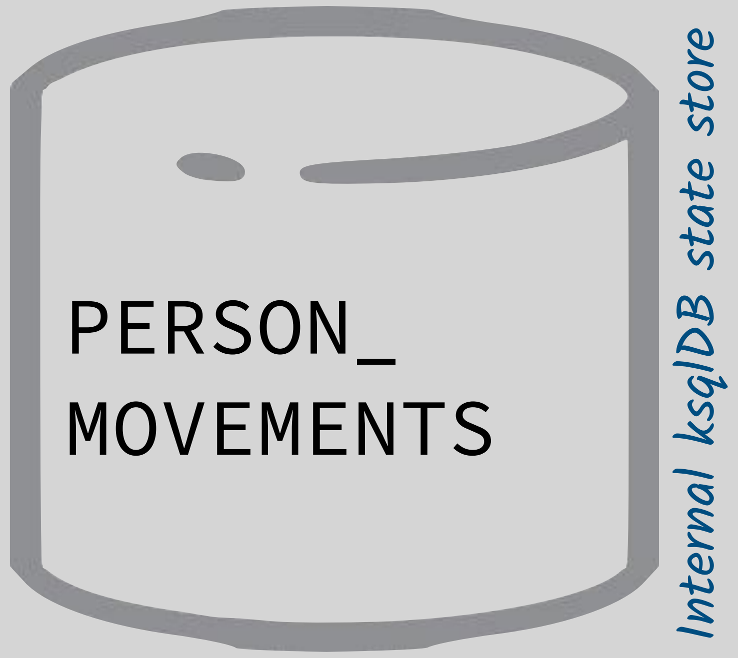
Pull and Push queries in ksqlDB

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;
```




```
ksql> SELECT LOCATION_CHANGES, UNIQUE_LOCATIONS
      FROM PERSON_MOVEMENTS
      WHERE ROWKEY='robin';
+-----------------+-----------------+
|LOCATION_CHANGES |UNIQUE_LOCATIONS |
+-----------------+-----------------+
|3                |4                |
Query terminated
ksql>
```

```
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
```

Pull query

Push query

ksqlDB REST API

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;
```

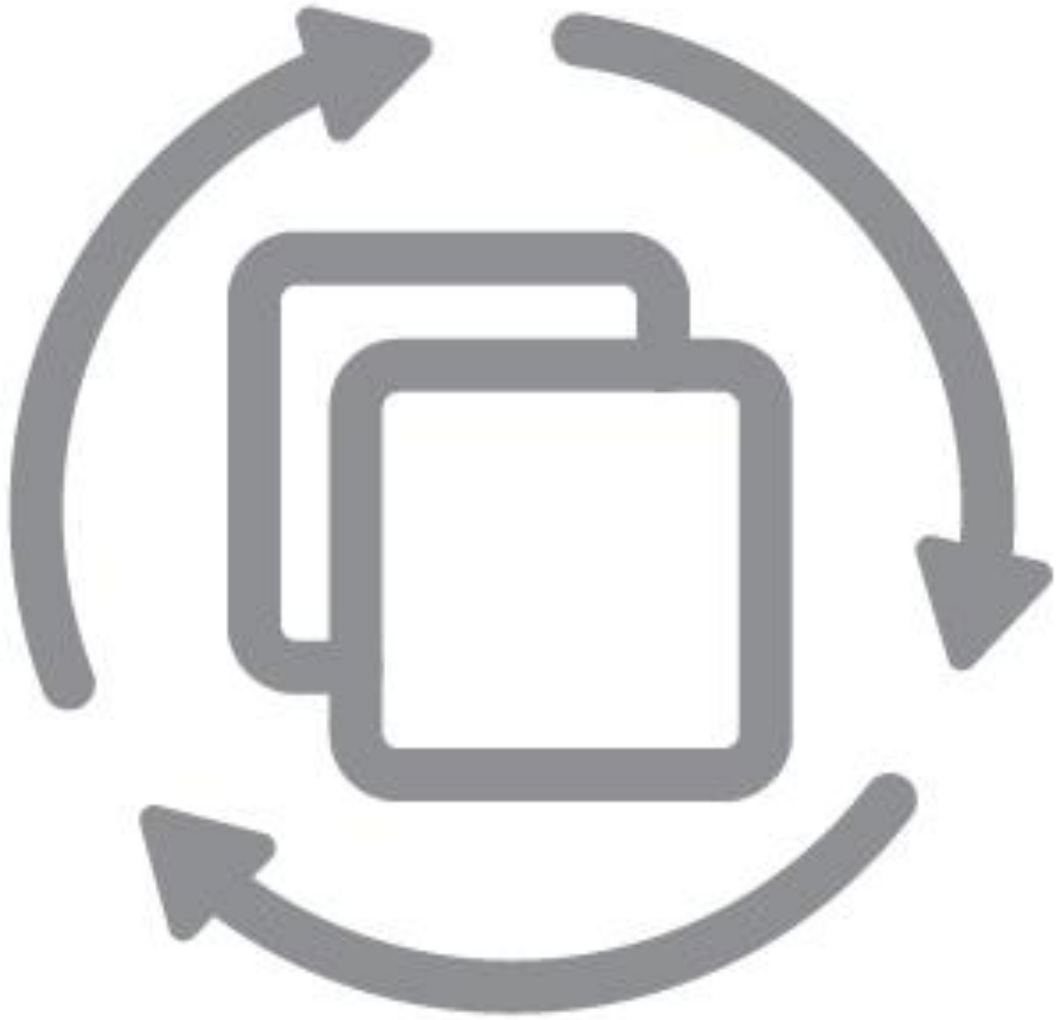
PERSON_
MOVEMENTS

Internal ksqlDB state store

```
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'\'';"}'
```

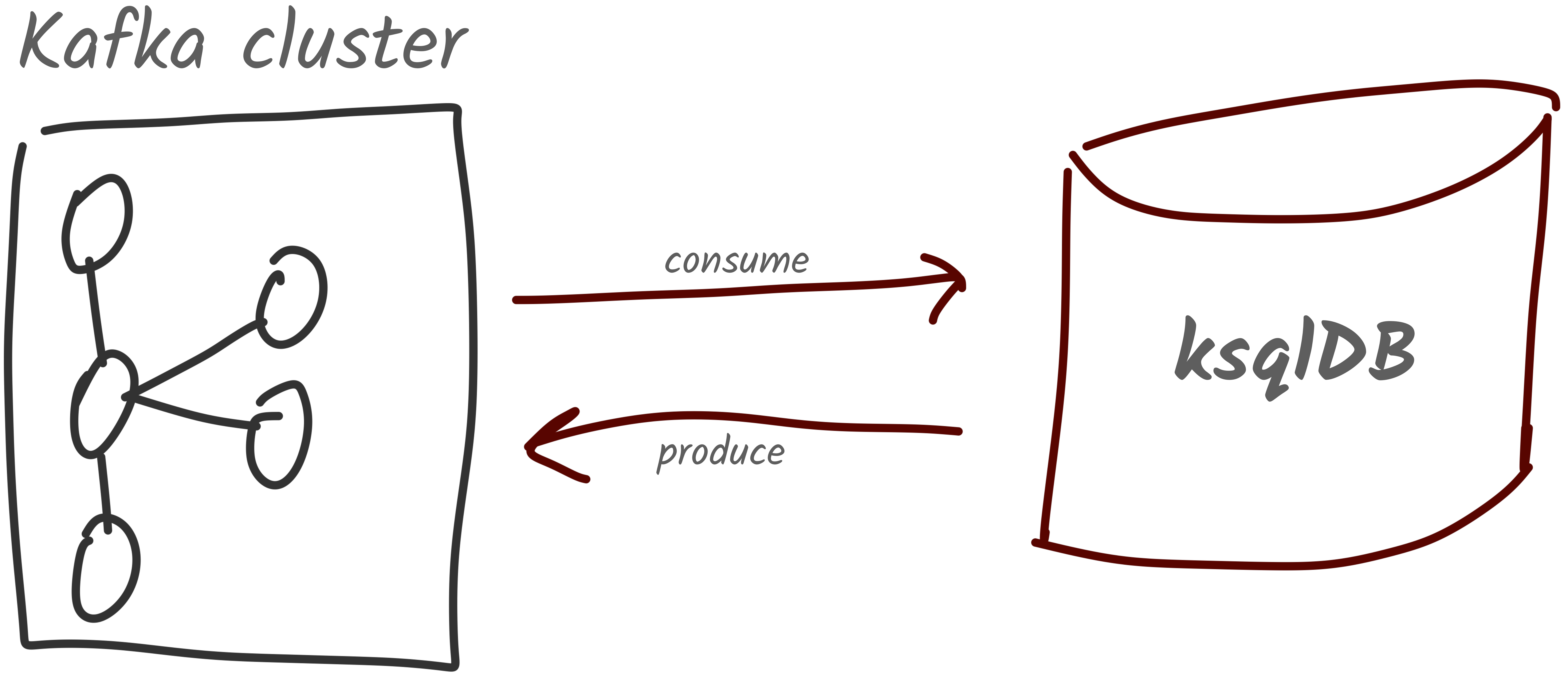
```
{"value":"3"}
```

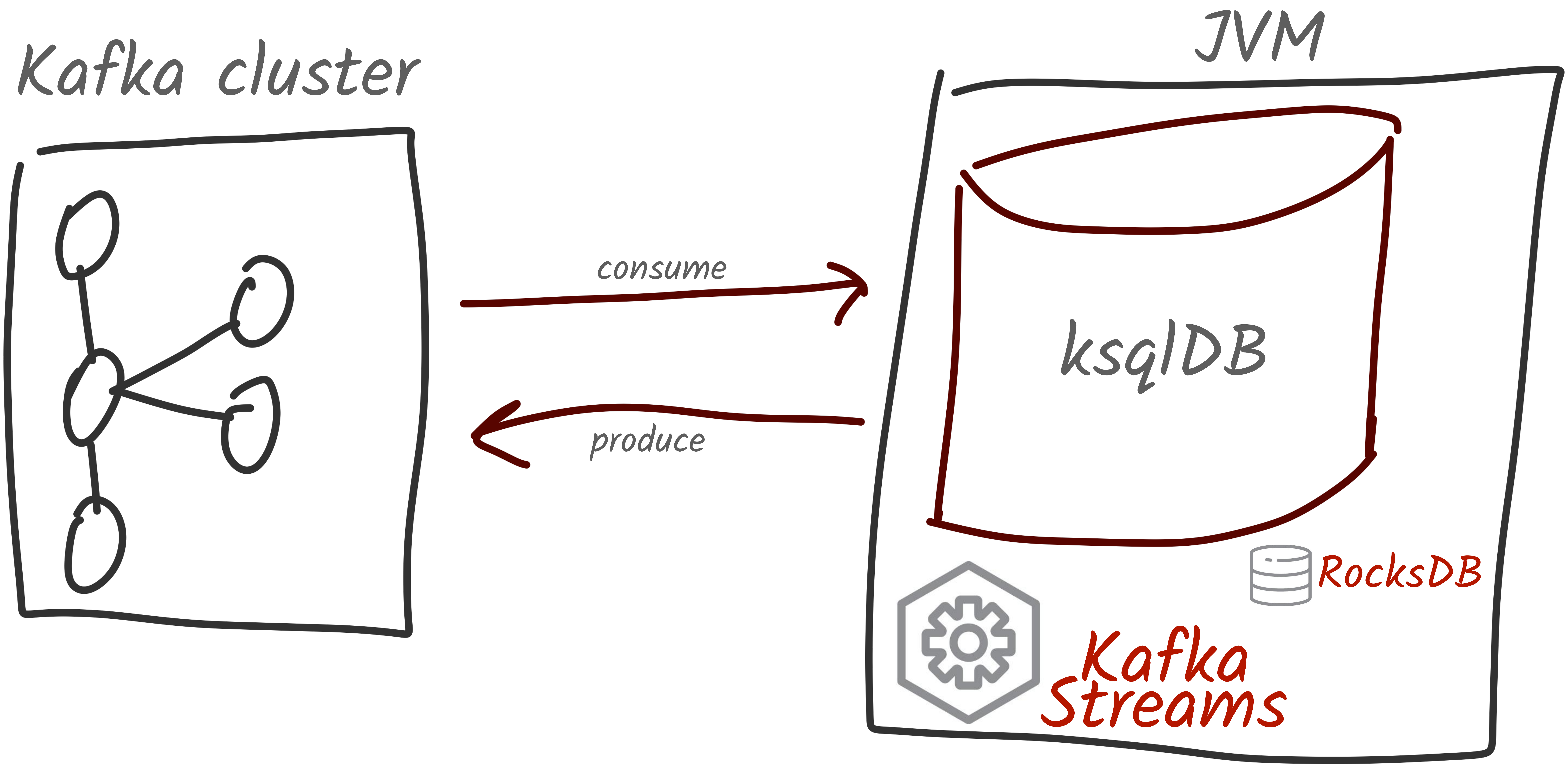
Pull and Push queries in ksqlDB

	Pull query	Push query
		
Tells you:	Point in time value	All value changes
Exits:	Immediately	Never

Under the covers of ksqlDB

Photo by Vinicius de Moraes on Unsplash


Kafka cluster
consume
produce
ksqlDB


Kafka cluster
JVM
consume
produce
ksqlDB
RocksDB
Kafka Streams


WHERE CAN YOU RUN KSQLDB?
ANYWHERE!
imgflip.com

confluent cloud

Fully Managed Kafka & KSQL as a Service

Running ksqlDB - self-managed

ksqlDB Server
(JVM process)

DEB, RPM, ZIP, TAR downloads
http://confluent.io/download

Docker images
confluentinc/ksqldb-server

Physical

docker

kubernetes

openstack

vmware

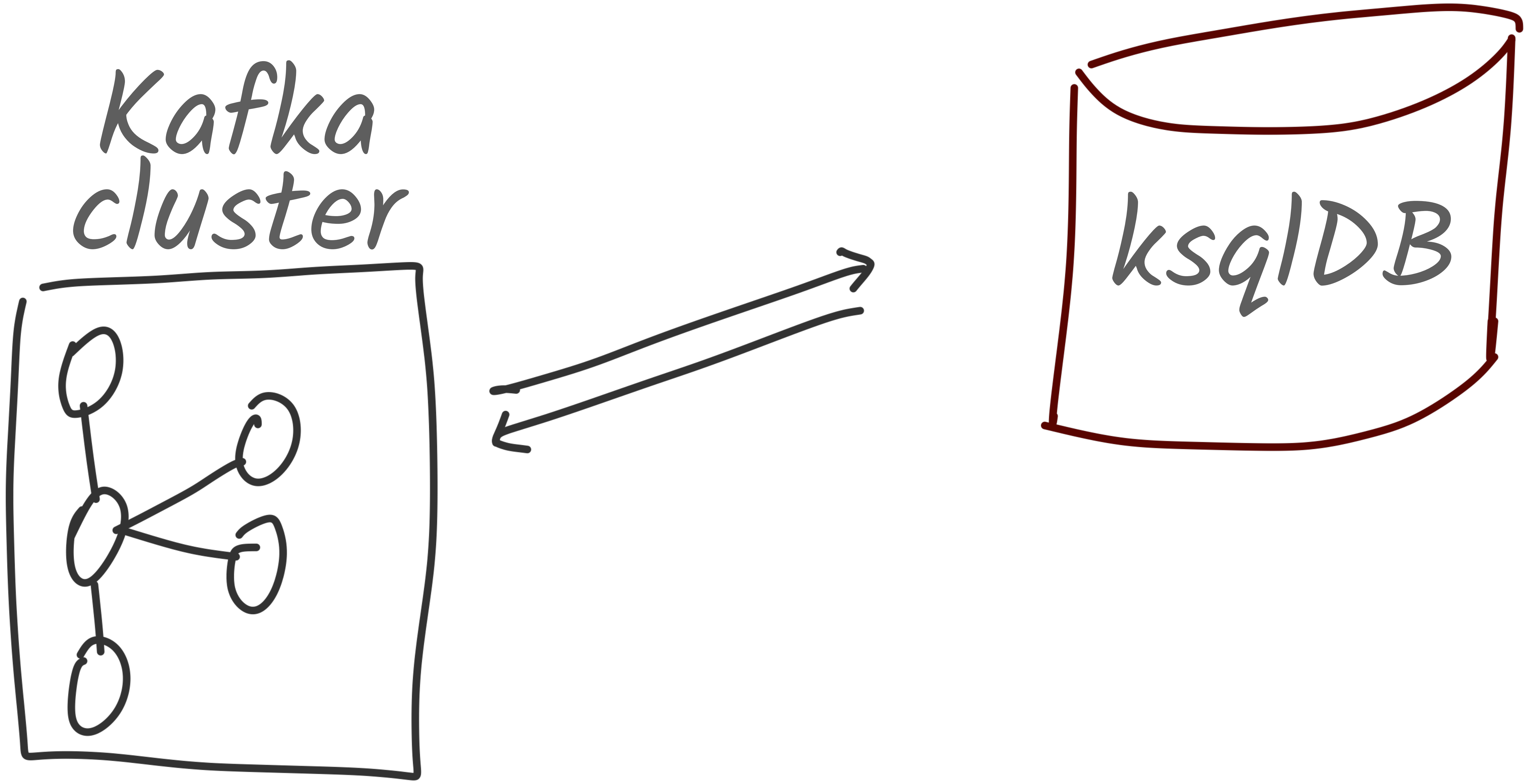
Microsoft Azure

Google Cloud Platform

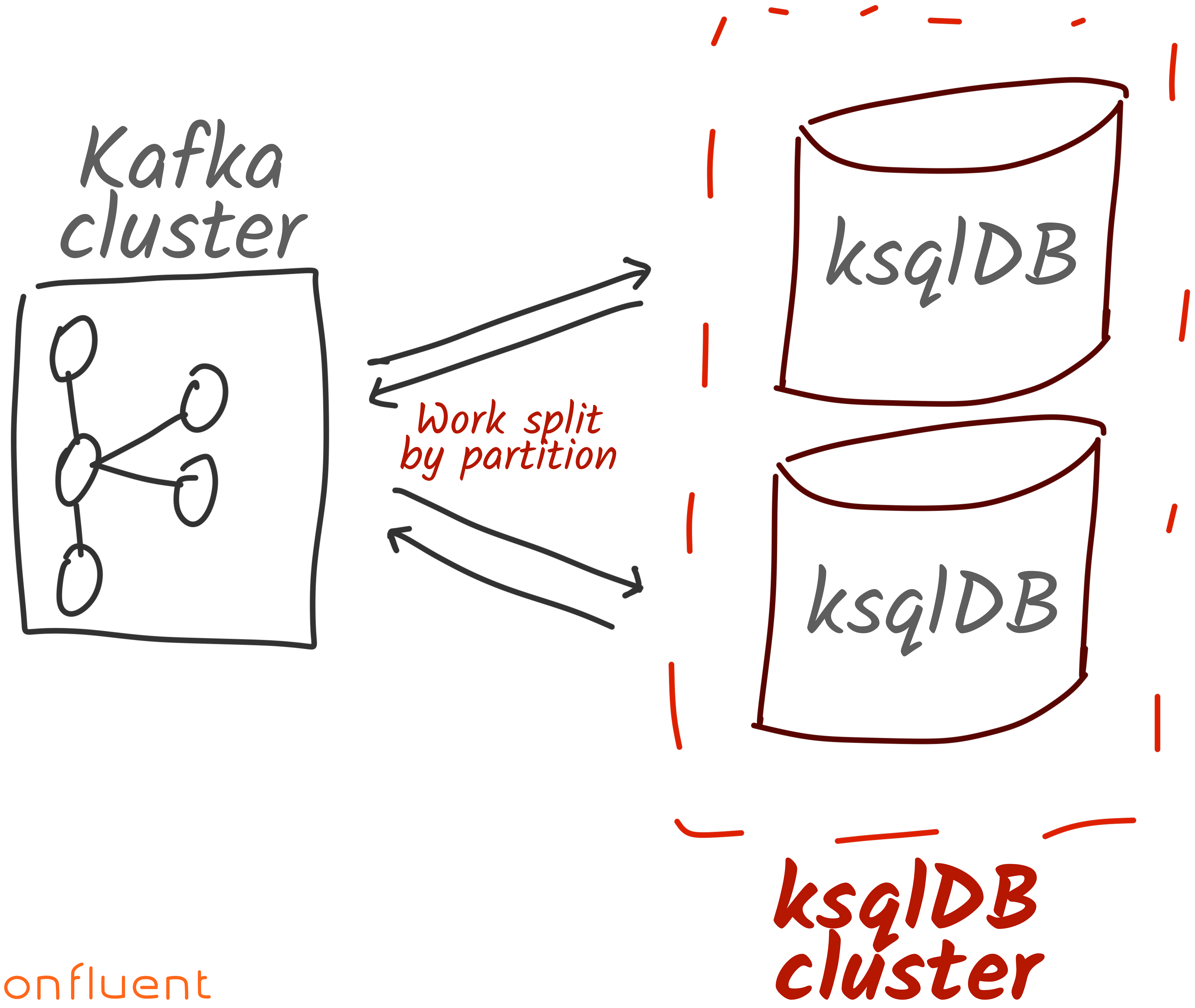
amazon web services

…and many more…

Scaling ksqlDB

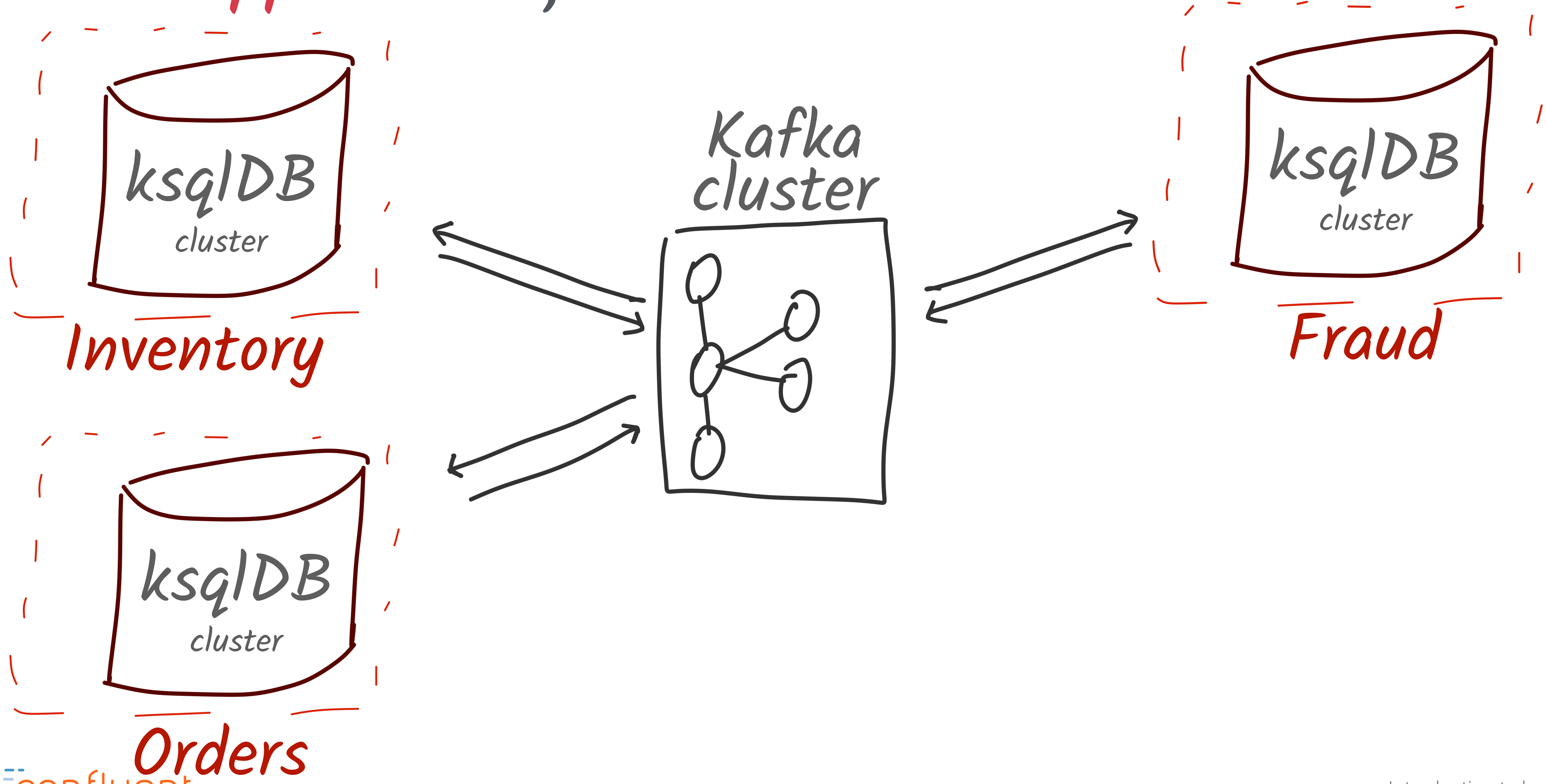


Scaling ksqlDB



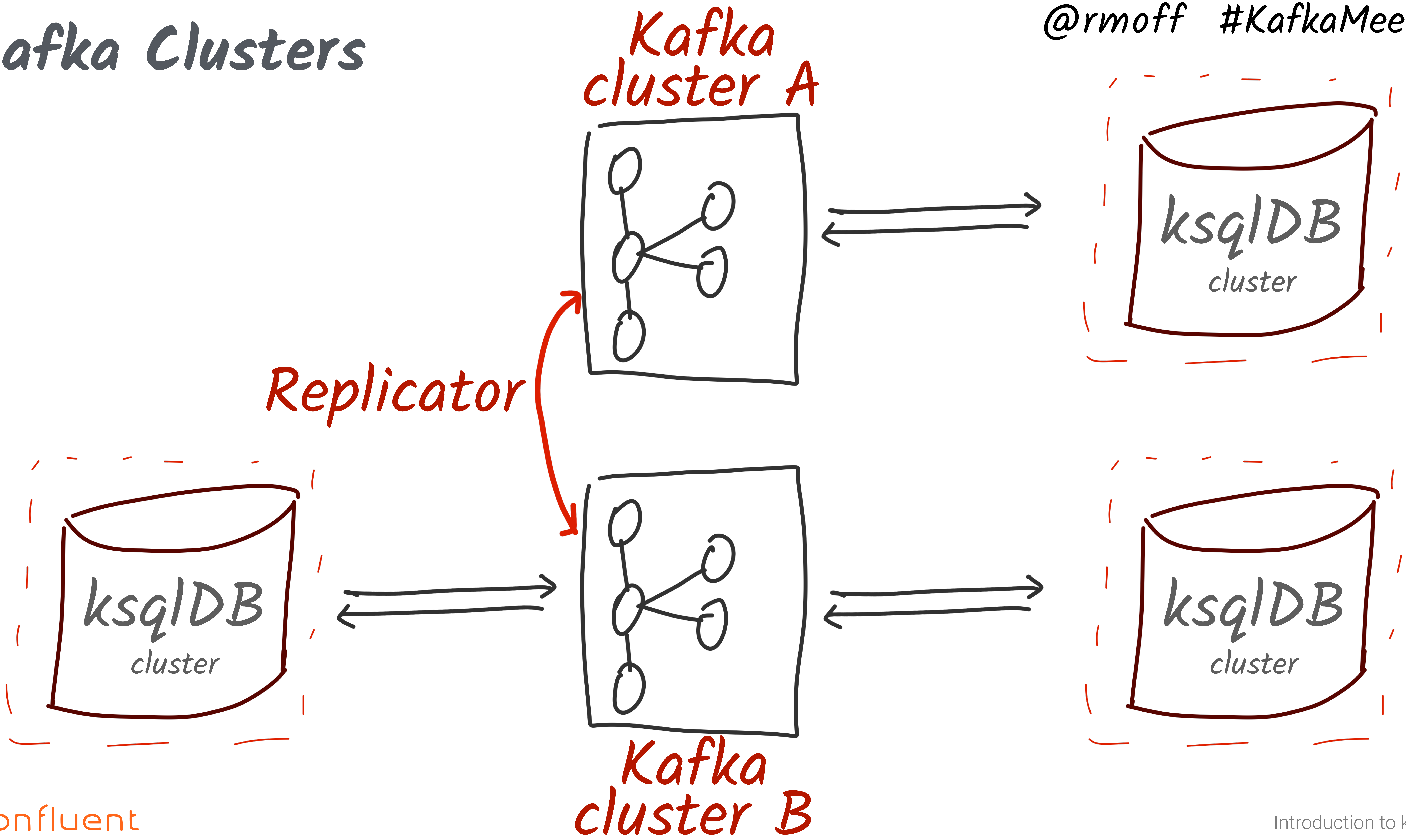

ONE DOES NOT SIMPLY RUN
ONE SINGLE KSQLDB CLUSTER

Think Applications, not database instances



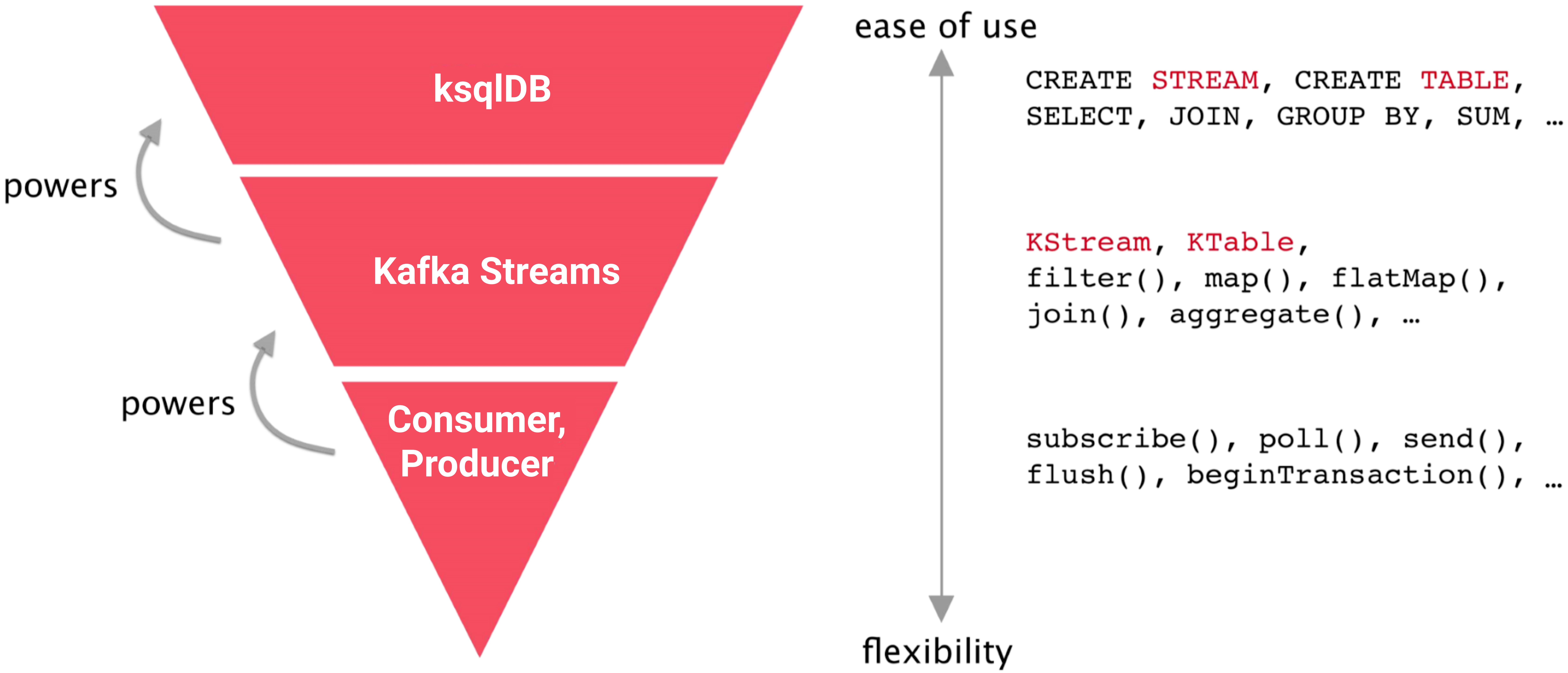
Kafka Clusters

@rmoff #KafkaMeetup



ksqlDB or Kafka Streams?

ksqlDB Builds on Streams



ksqlDB supports UDF, UDAF, UDTF

@rmoff #KafkaMeetup

ksqlDB

Search

confluentinc/ksql
3.2k Stars · 602 Forks

Overview Concepts Developer Guide Reference Operate and Deploy Tutorials

Developer Guide

- Develop ksqlDB Applications
- Create a Stream
- Create a Table
- Aggregate Streaming Events
- Implement a User-defined Function (UDF, UDAF, and UDTF)
- Query With Arrays and Maps
- Query With Structured Data
- Transform a Stream
- Example Queries
- Test and Debug

Implement a User-defined Function (UDF, UDAF, and UDTF)

Prerequisites

- Apache Maven
- Confluent Platform installed locally
- Internet connectivity for downloading Confluent POM files

Create a user-defined function (UDF), a user-defined aggregation function (UDAF), or a user-defined table function (UDTF) by following these steps:

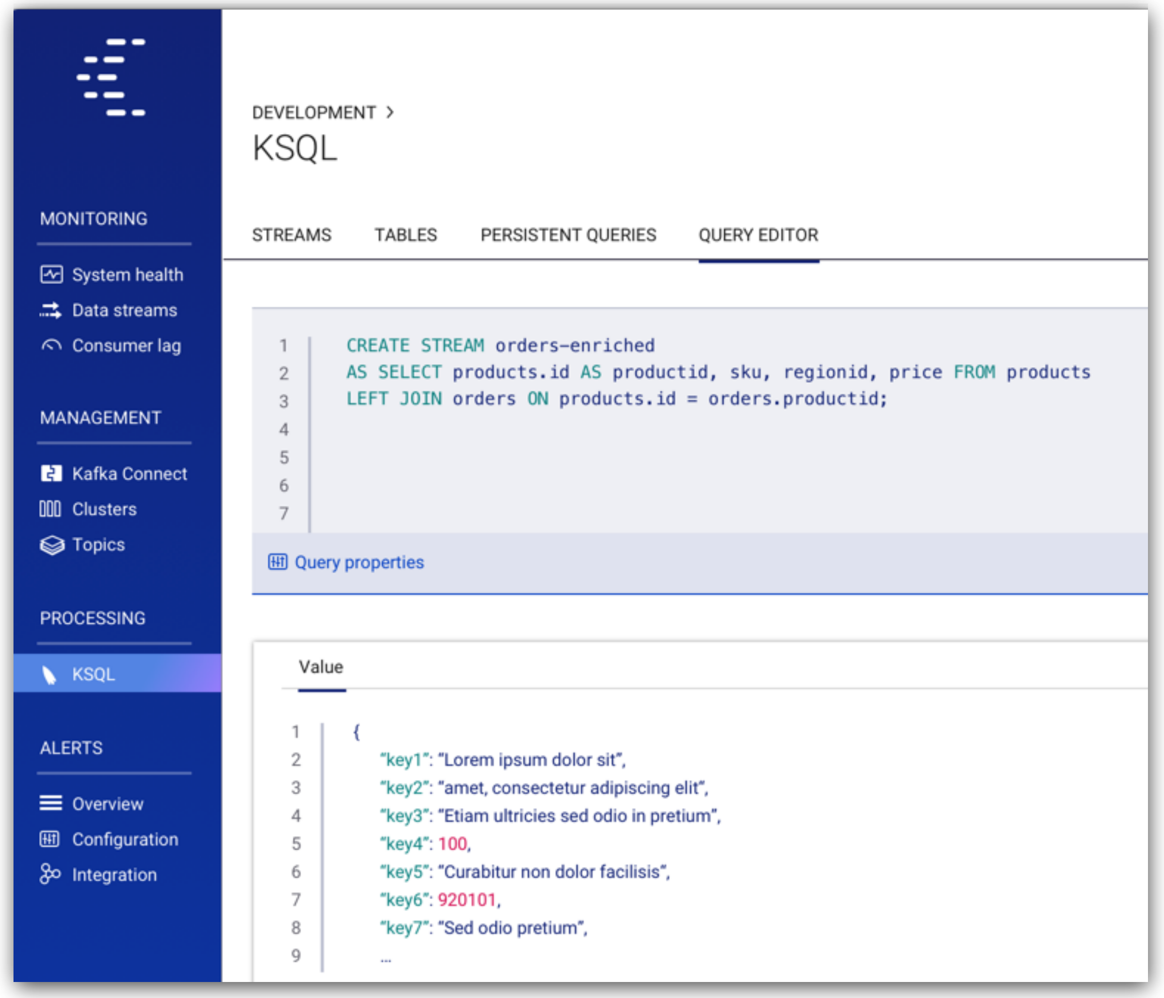
1. Create the ksql extensions directory that contains your packages.

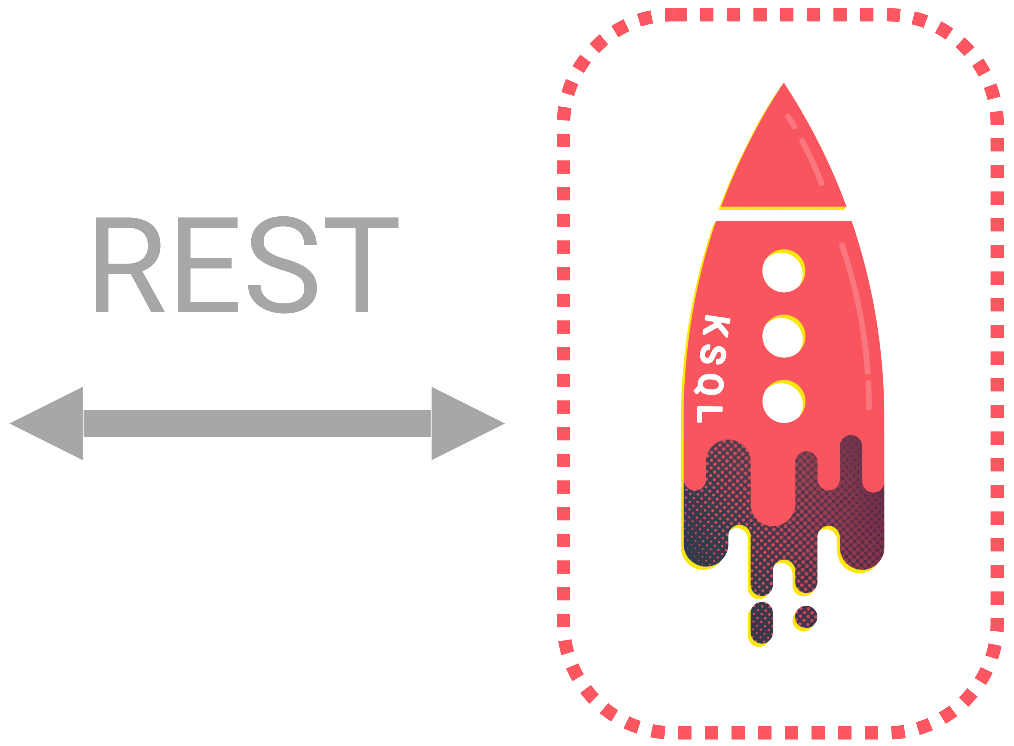
Table of contents

ksqlDB code lifecycle

Web UI and CLI
for development and testing

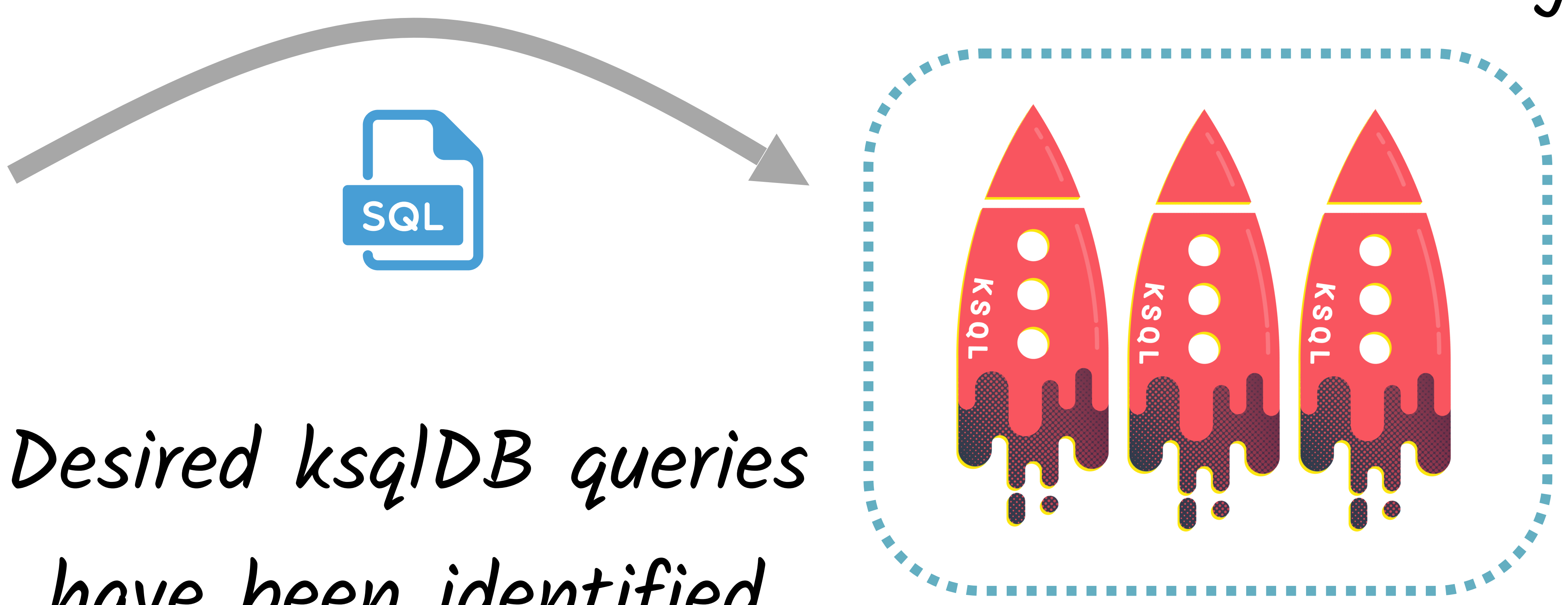
Single ksqlDB node



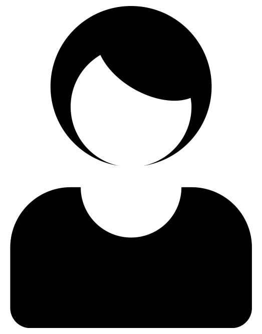


REST API
to deploy code for Production

ksqlDB clustered for scale and availability



Desired ksqlDB queries have been identified

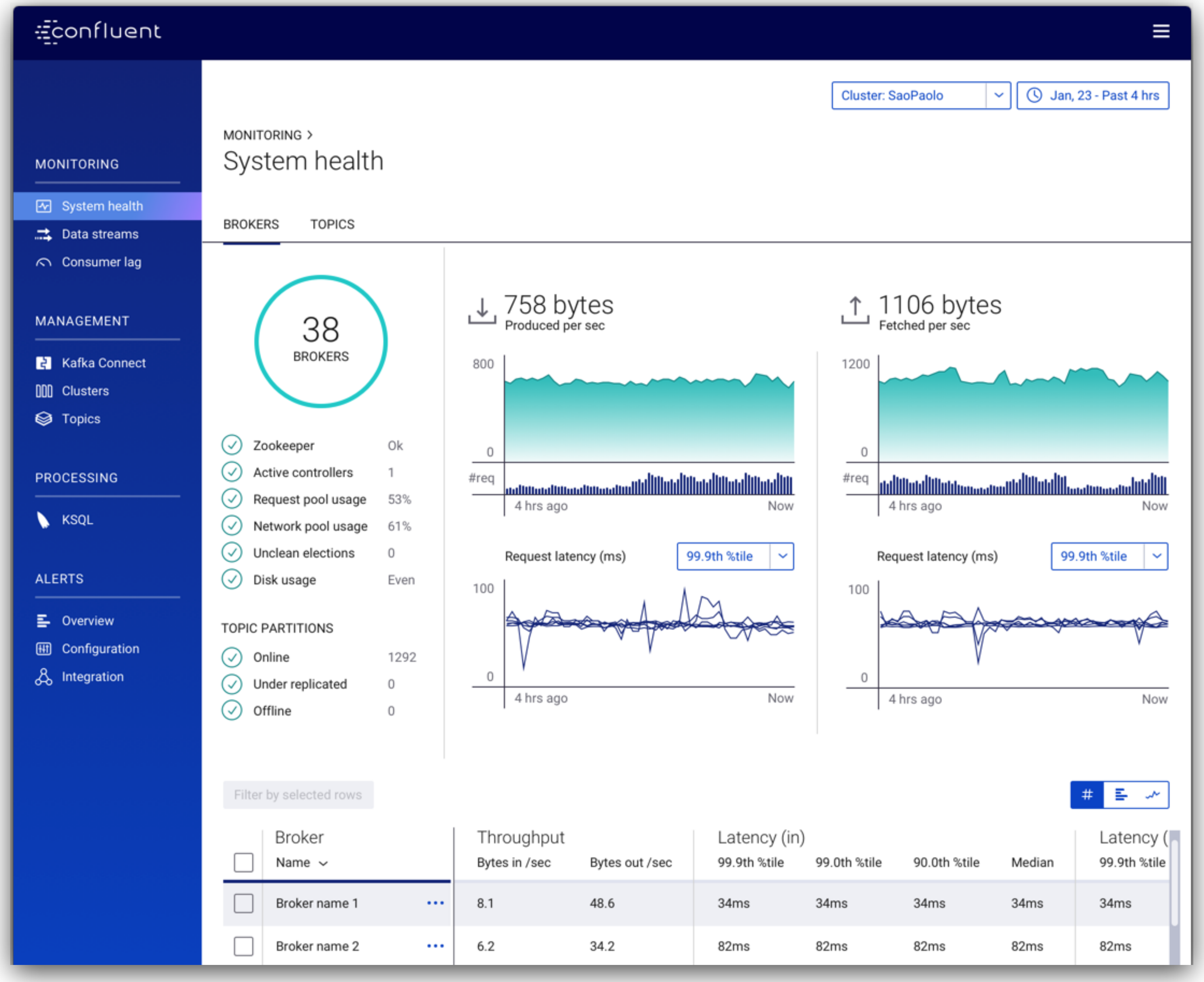


"Hmm, let me try out this idea..."

```
curl -s -X "POST" "http://localhost:8088/ksql" \
     -H "Content-Type: application/vnd.ksql.v1+json; charset=utf-8" \
     -d '{
            "ksql":"CREATE STREAM LONDON AS SELECT * FROM MOVEMENTS WHERE LOCATION='\''london'\'';",
            "streamsProperties": {
                "ksql.streams.auto.offset.reset": "earliest"
            }
        }'
```

Monitoring ksqlDB

Confluent Control Center



JMX

https://www.confluent.io/blog/troubleshooting-ksql-part-2

@rmoff #KafkaMeetup

Want to learn more?

confluent

CTAs, not CATs
(sorry, not sorry)

kafka-summit.org

KAFKA SUMMIT 2020

LONDON

APRIL 27 & 28

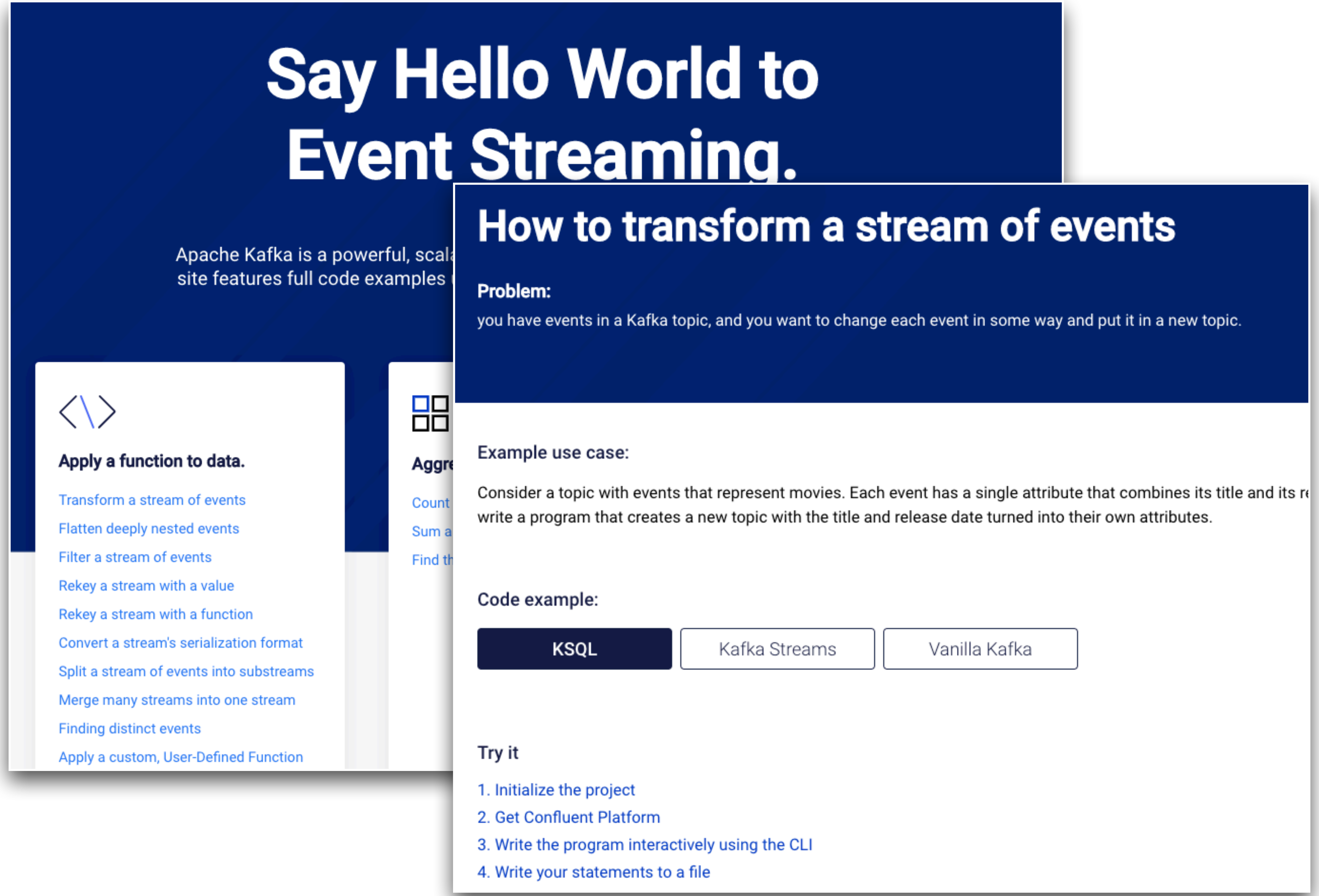
Moffatt30

*30% OFF**

*Standard Priced Conference pass

Kafka Tutorials

@rmoff #KafkaMeetup



https://kafka-tutorials.confluent.io/

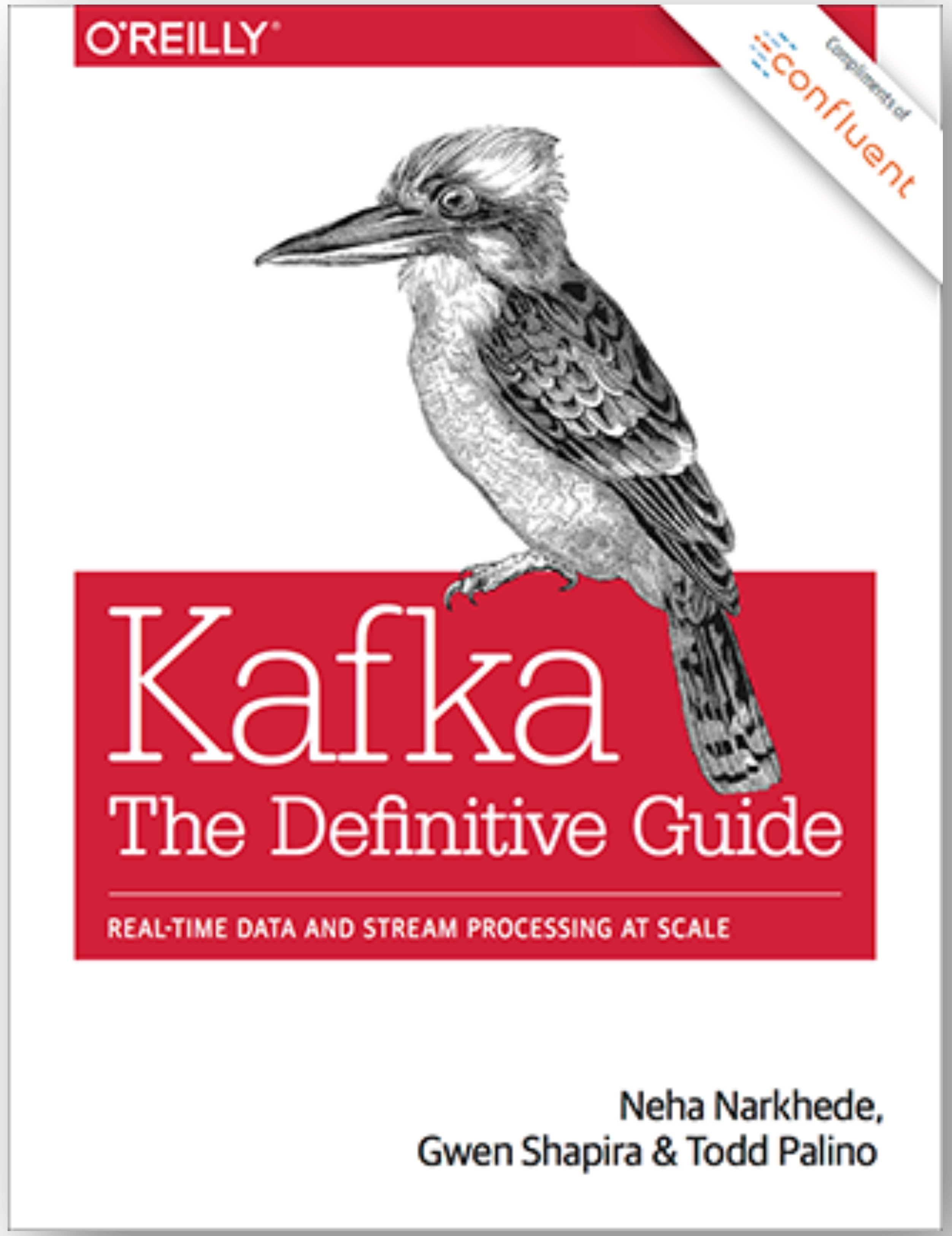
Confluent Community Slack group

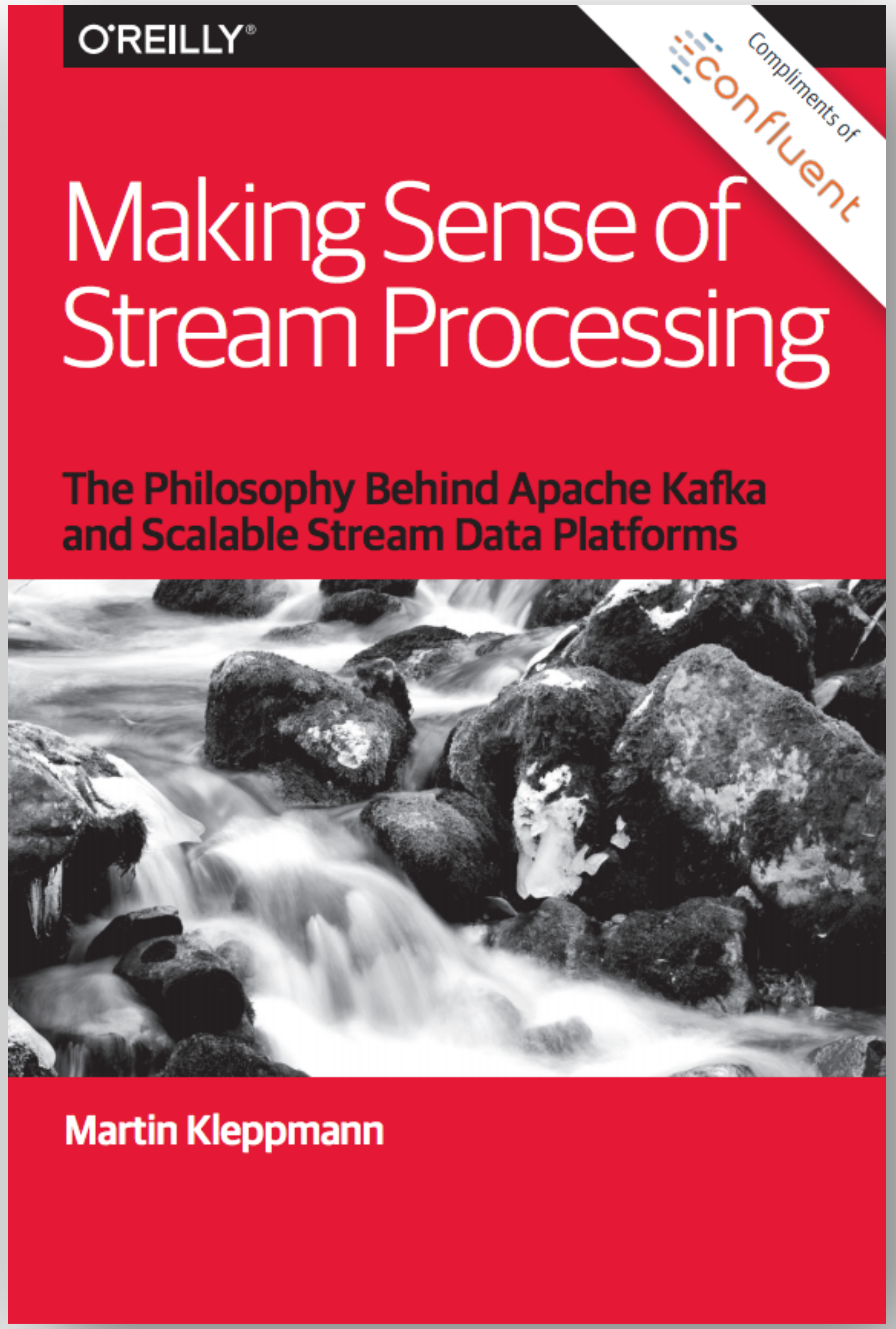
cnfl.io/slack

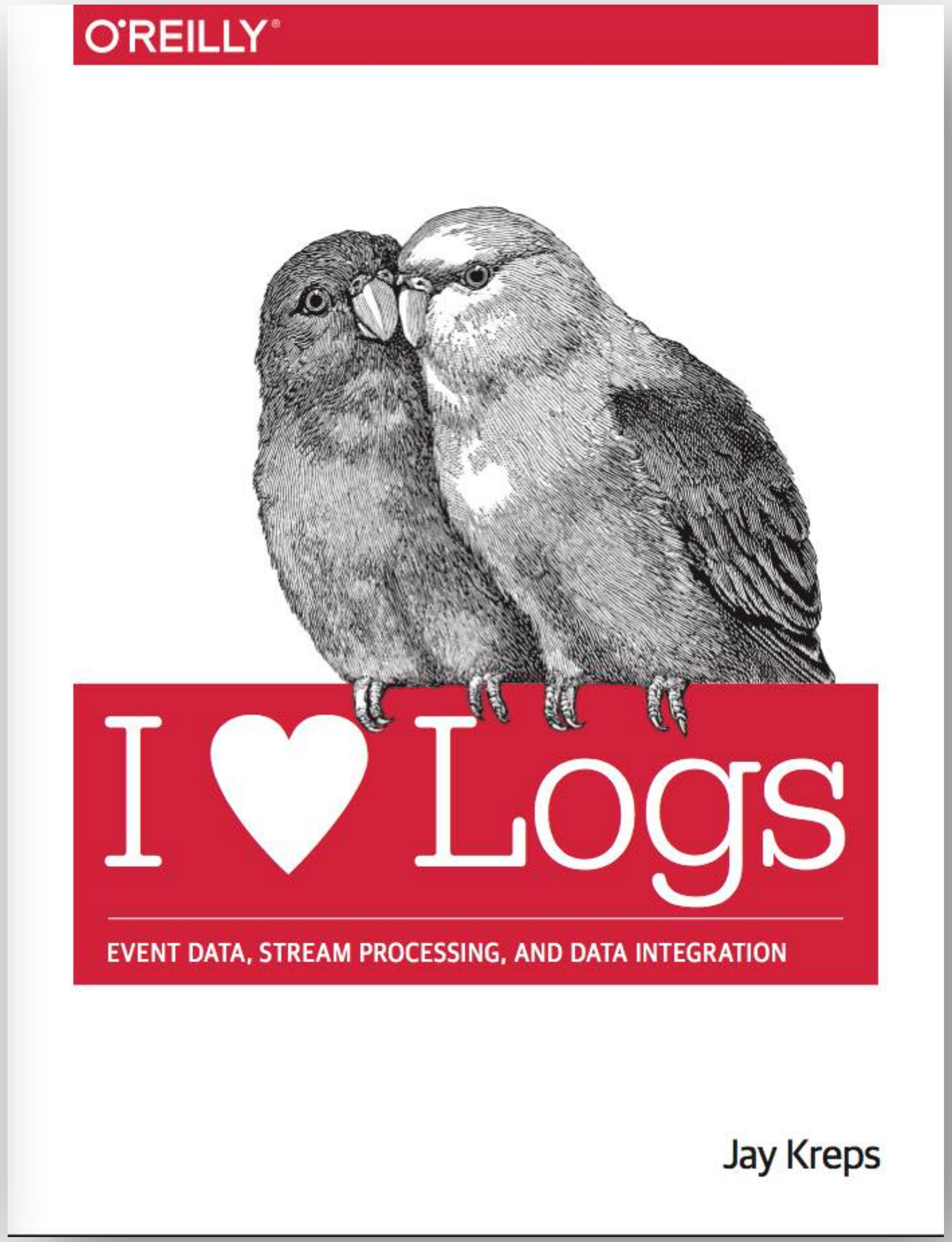
Free Books!

@rmoff

https://rmoff.dev/books-krk20







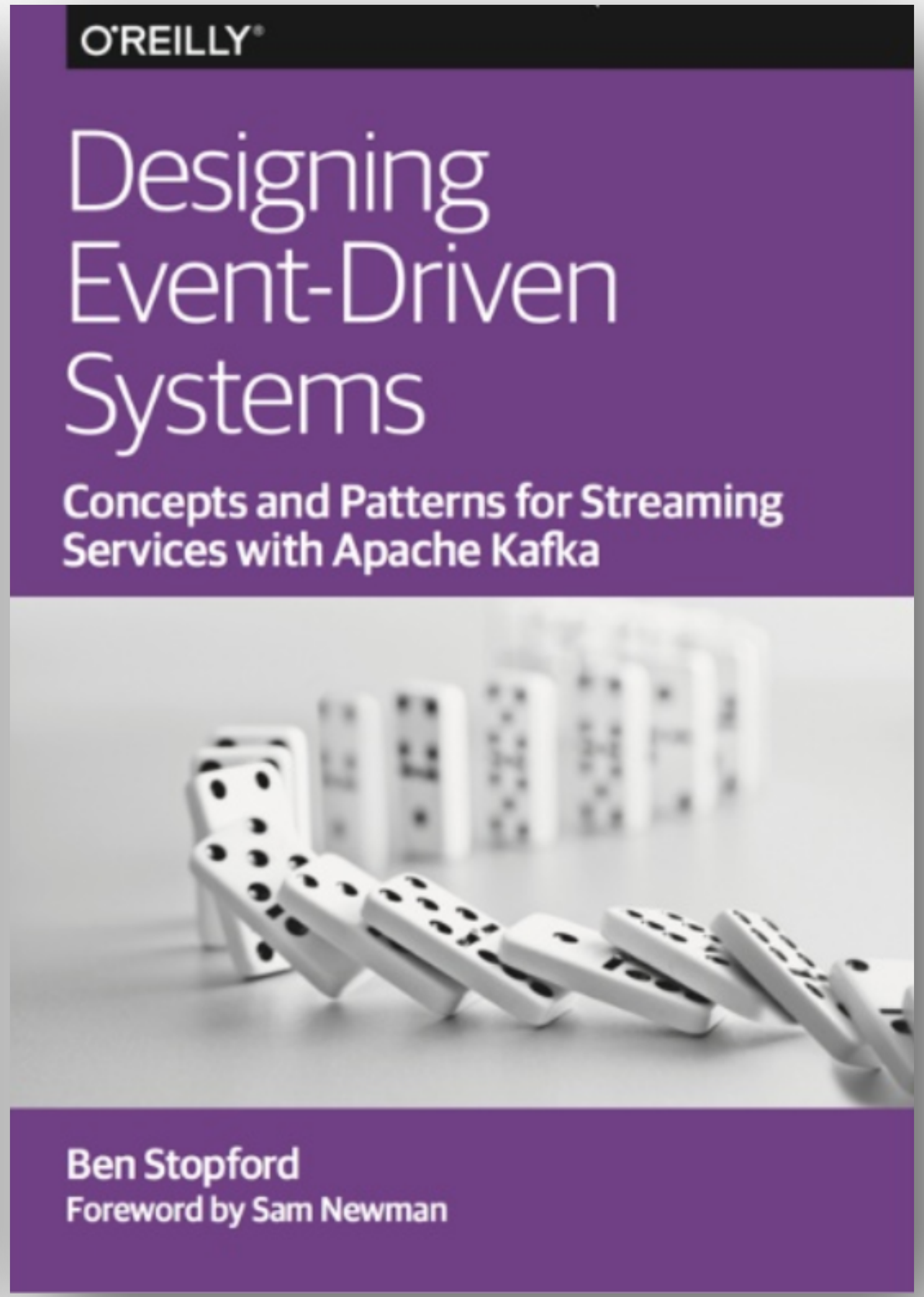


Photo by Tengyart on Unsplash

Filtering with ksqlDB

ORDERS

Filtering with ksqlDB

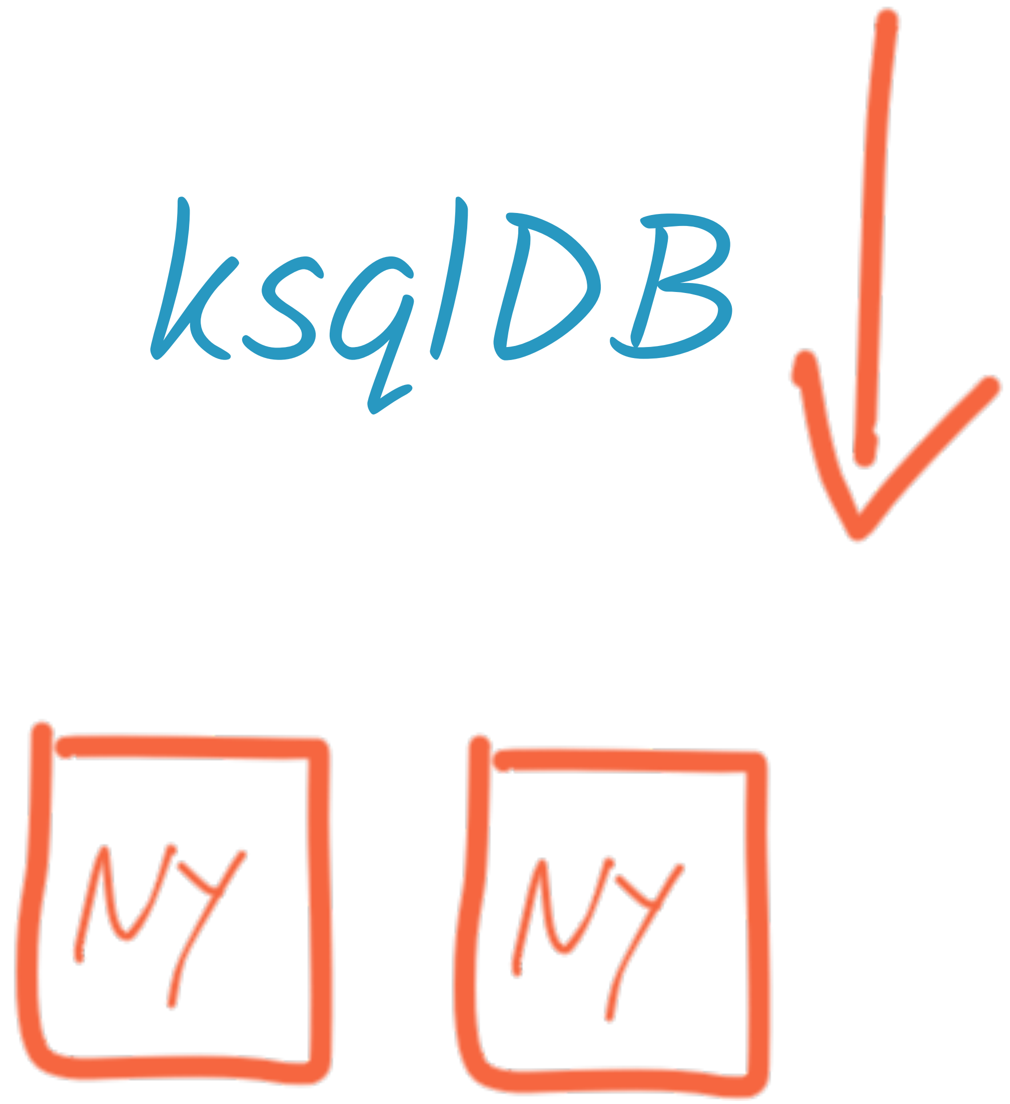
ORDERS

ksqlDB

```
CREATE STREAM ORDERS_NY AS
  SELECT *
    FROM ORDERS
   WHERE ADDRESS->STATE='New York';
```

Filtering with ksqlDB

ORDERS




```
CREATE STREAM ORDERS_NY AS
  SELECT *
    FROM ORDERS
   WHERE ADDRESS->STATE='New York';
```

ORDERS_NY

Transform data with ksqlDB - merge streams

Transform data with ksqlDB - merge streams

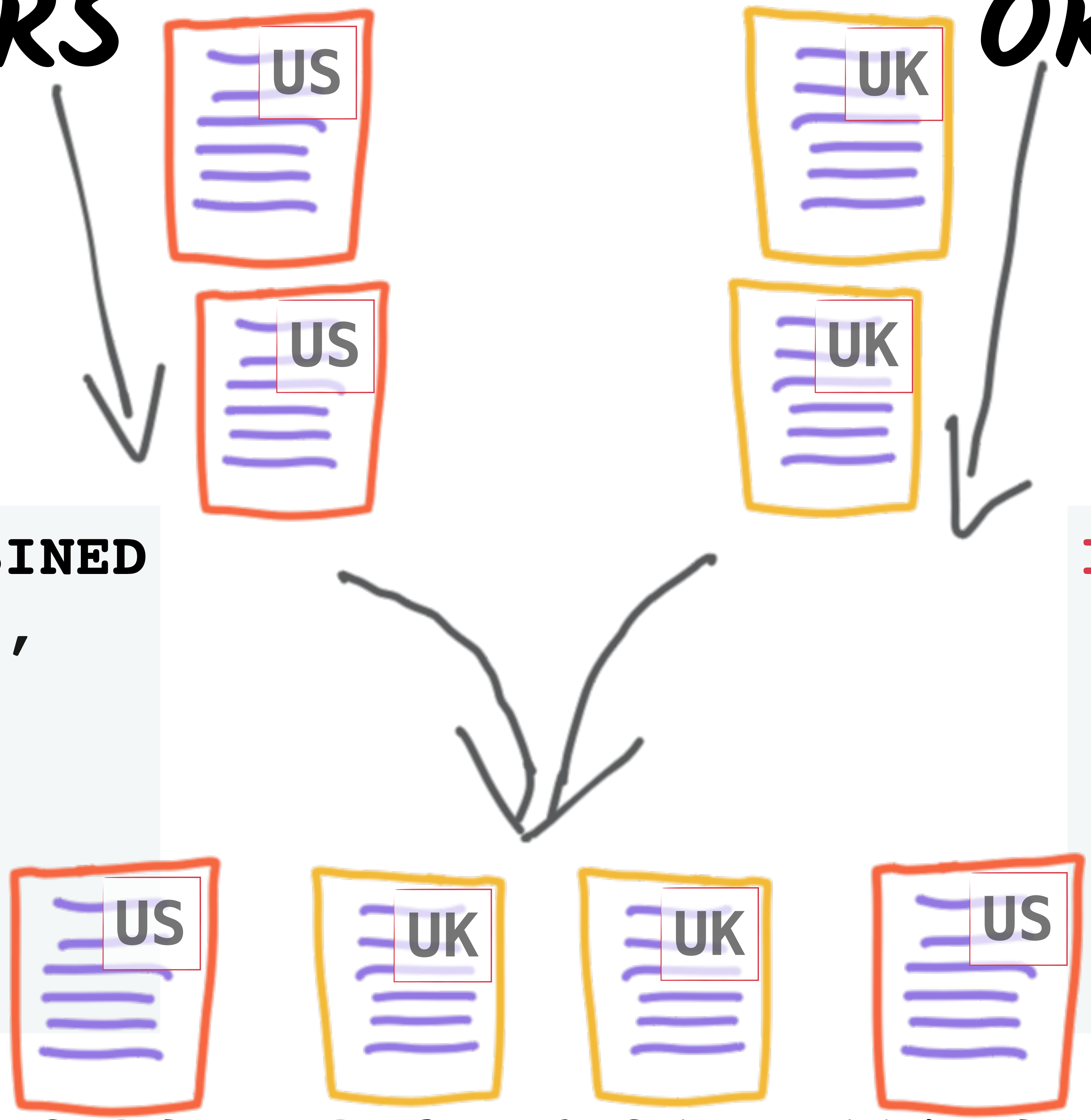
ORDERS

ORDERS_UK


```
INSERT INTO ORDERS_COMBINED
  SELECT 'US' AS SOURCE,
         ORDERTIME,
         ITEMID,
         ORDERUNITS,
         ADDRESS
    FROM ORDERS;
```

```
INSERT INTO ORDERS_COMBINED
  SELECT 'UK' AS SOURCE,
         ORDERTIME,
         ITEMID,
         ORDERUNITS,
         ADDRESS
    FROM ORDERS_UK;
```

Transform data with ksqlDB - merge streams




```
INSERT INTO ORDERS_COMBINED
  SELECT 'US' AS SOURCE,
         ORDERTIME,
         ITEMID,
         ORDERUNITS,
         ADDRESS
    FROM ORDERS;
```

```
INSERT INTO ORDERS_COMBINED
  SELECT 'UK' AS SOURCE,
         ORDERTIME,
         ITEMID,
         ORDERUNITS,
         ADDRESS
    FROM ORDERS_UK;
```

Transform data with ksqlDB - split streams

ORDERS_COMBINED

Transform data with ksqlDB - split streams

@rmoff #KafkaMeetup

ORDERS_COMBINED

```
CREATE STREAM ORDERS_US AS
  SELECT *
    FROM ORDERS_COMBINED
   WHERE SOURCE ='US';
```

```
CREATE STREAM ORDERS_UK AS
  SELECT *
    FROM ORDERS_COMBINED
   WHERE SOURCE ='UK';
```

Transform data with ksqlDB - split streams

ORDERS_COMBINED

```
CREATE STREAM ORDERS_US AS
  SELECT *
    FROM ORDERS_COMBINED
   WHERE SOURCE ='US';
```

```
CREATE STREAM ORDERS_UK AS
  SELECT *
    FROM ORDERS_COMBINED
   WHERE SOURCE ='UK';
```

ORDERS_US   ORDERS_UK

Schema manipulation - flatten records

ORDERS

Nested fields

```json
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address": {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

Schema manipulation with ksqlDB

ORDERS


```json
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address": {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```


ksqlDB

```sql
CREATE STREAM ORDERS_FLAT AS
  SELECT […]
        ADDRESS->STREET AS ADDRESS_STREET,
         ADDRESS->CITY AS ADDRESS_CITY,
         ADDRESS->STATE AS ADDRESS_STATE
    FROM ORDERS;
```

Schema manipulation with ksqlDB

@rmoff #KafkaMeetup

ORDERS


```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address": {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

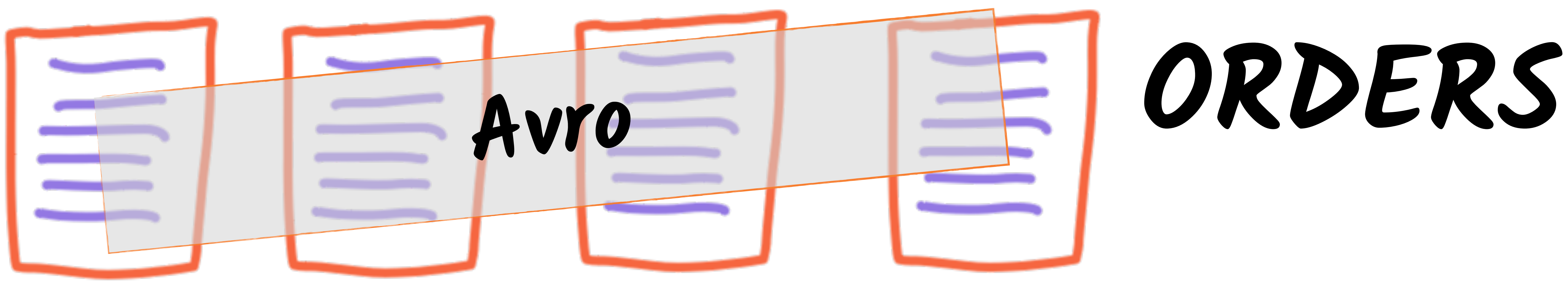


```
CREATE STREAM ORDERS_FLAT AS
  SELECT […]
        ADDRESS->STREET AS ADDRESS_STREET,
         ADDRESS->CITY AS ADDRESS_CITY,
         ADDRESS->STATE AS ADDRESS_STATE
    FROM ORDERS;
```

ORDERS_FLAT

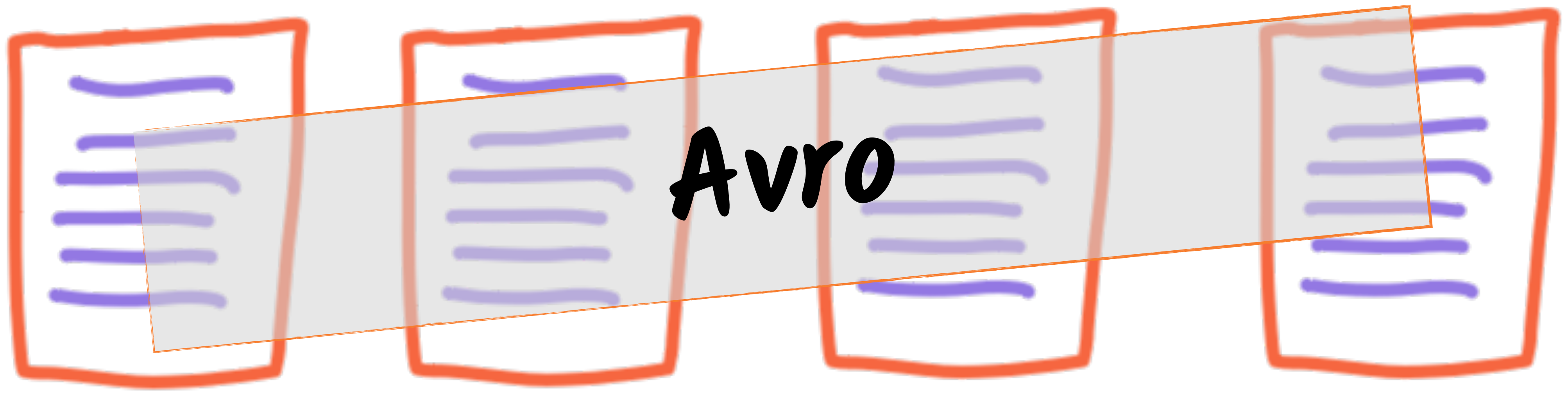

```
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address-street": "243 Utah Way",
  "address-city": "Orange",
  "address-state": "California"}
```

Reserialising data with ksqlDB




```
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address-street": "243 Utah Way",
  "address-city": "Orange",
  "address-state": "California"}
```

Reserialising data with ksqlDB



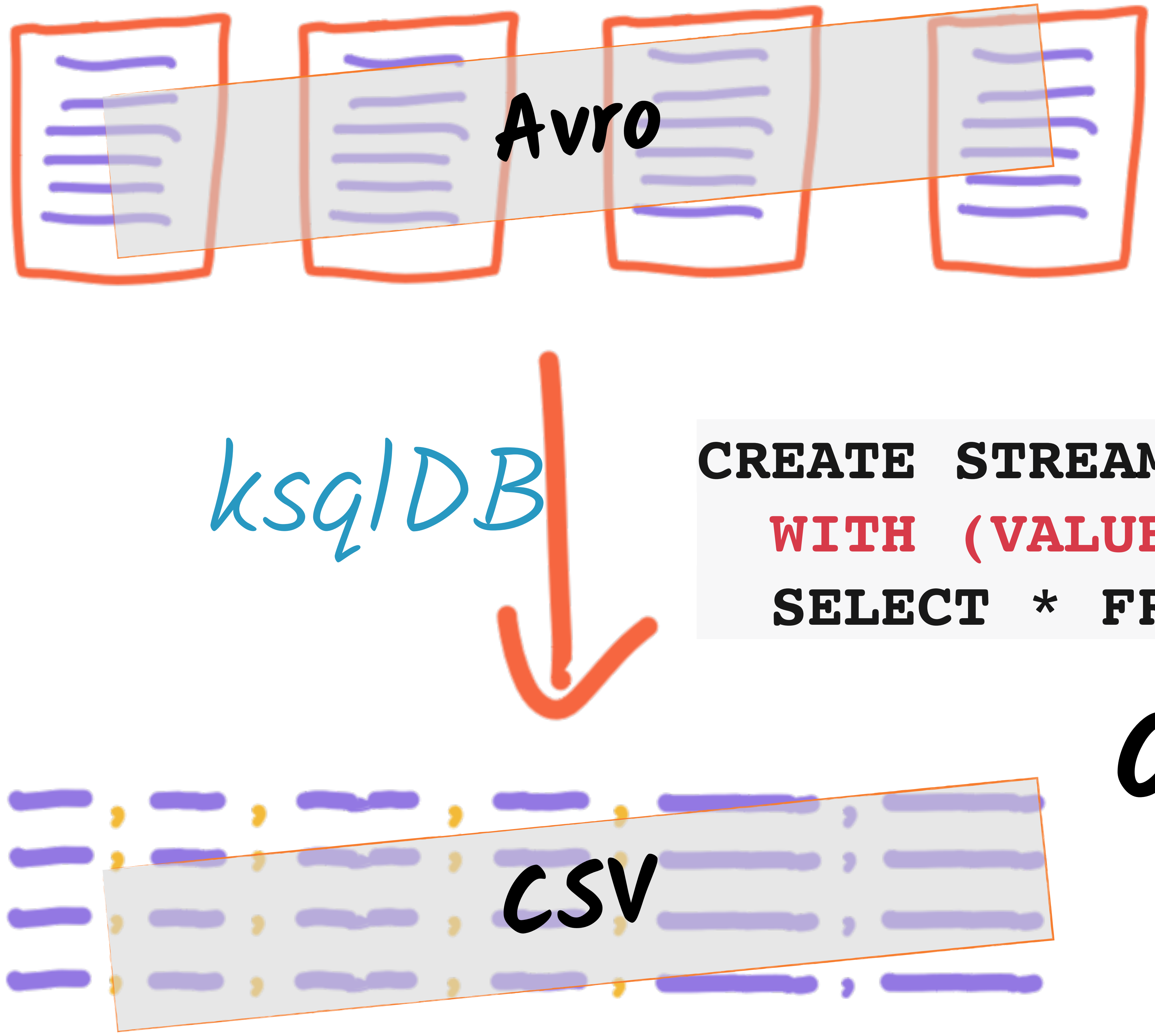
ORDERS


```
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address-street": "243 Utah Way",
  "address-city": "Orange",
  "address-state": "California"}
```


ksqlDB

```
CREATE STREAM ORDERS_CSV
  WITH (VALUE_FORMAT='DELIMITED') AS
  SELECT * FROM ORDERS_FLAT;
```

Reserialising data with ksqlDB



ORDERS


```
{"ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address-street": "243 Utah Way",
  "address-city": "Orange",
  "address-state": "California"}
```


```
CREATE STREAM ORDERS_CSV
  WITH (VALUE_FORMAT='DELIMITED') AS
  SELECT * FROM ORDERS_FLAT;
```

ORDERS_CSV


```
1560045914101,24644,Item_0,1,43078 De
1560047305664,24643,Item_29,3,209 Mon
1560057079799,24642,Item_38,18,3 Autu
1560088652051,24647,Item_6,6,82893 Ar
1560105559145,24648,Item_0,12,45896 W
1560108336441,24646,Item_33,4,272 Hef
1560123862235,24641,Item_15,16,0 Dort
1560124799053,24645,Item_12,1,71 Knut
```