Big, Fast, Easy Data: Distributed stream processing for everyone with KSQL

The Streaming SQL Engine for Apache Kafka

Michael G. Noll, Confluent
@miguno

confluent

Founded by the creators of **Apache Kafka**

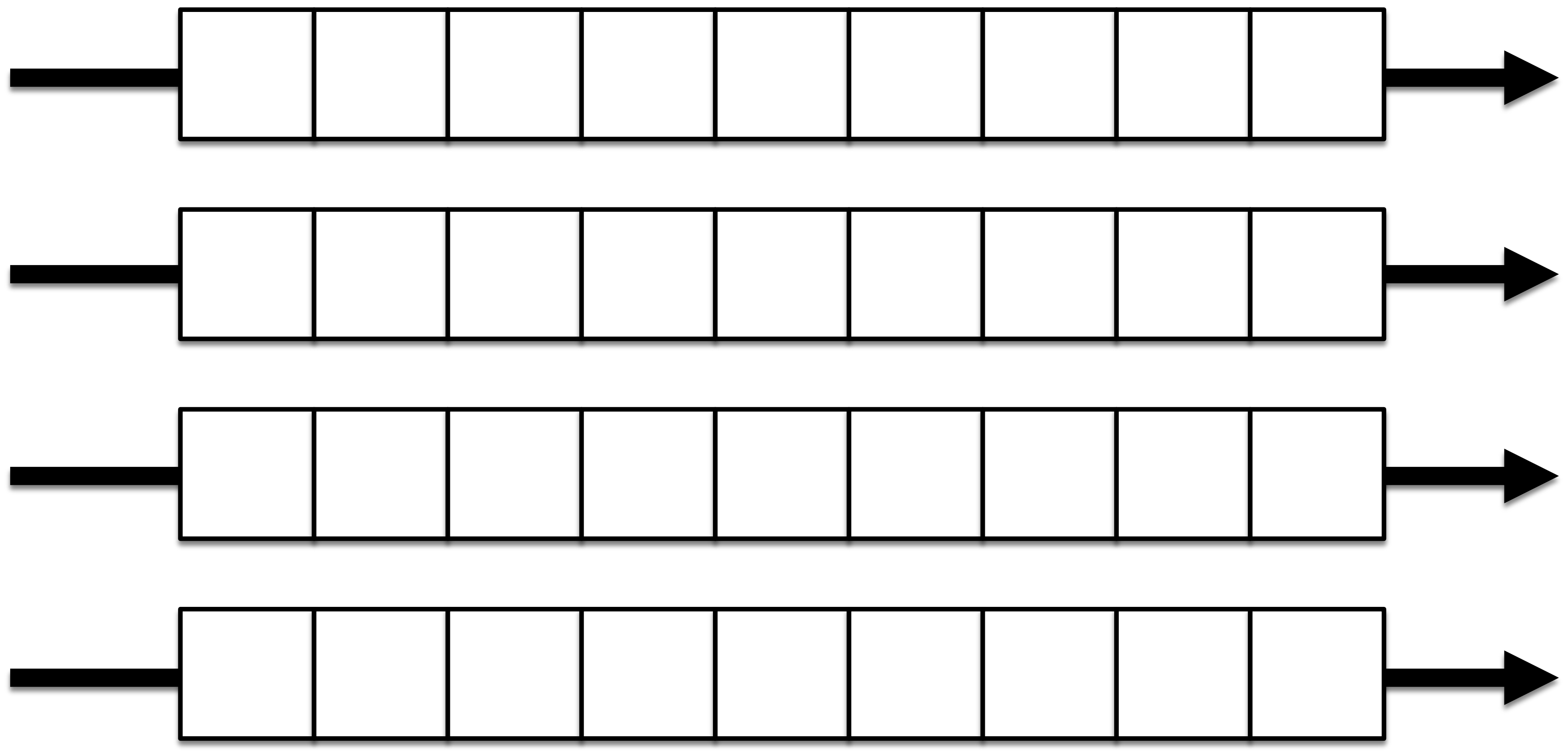
Technology Developed while at **LinkedIn**

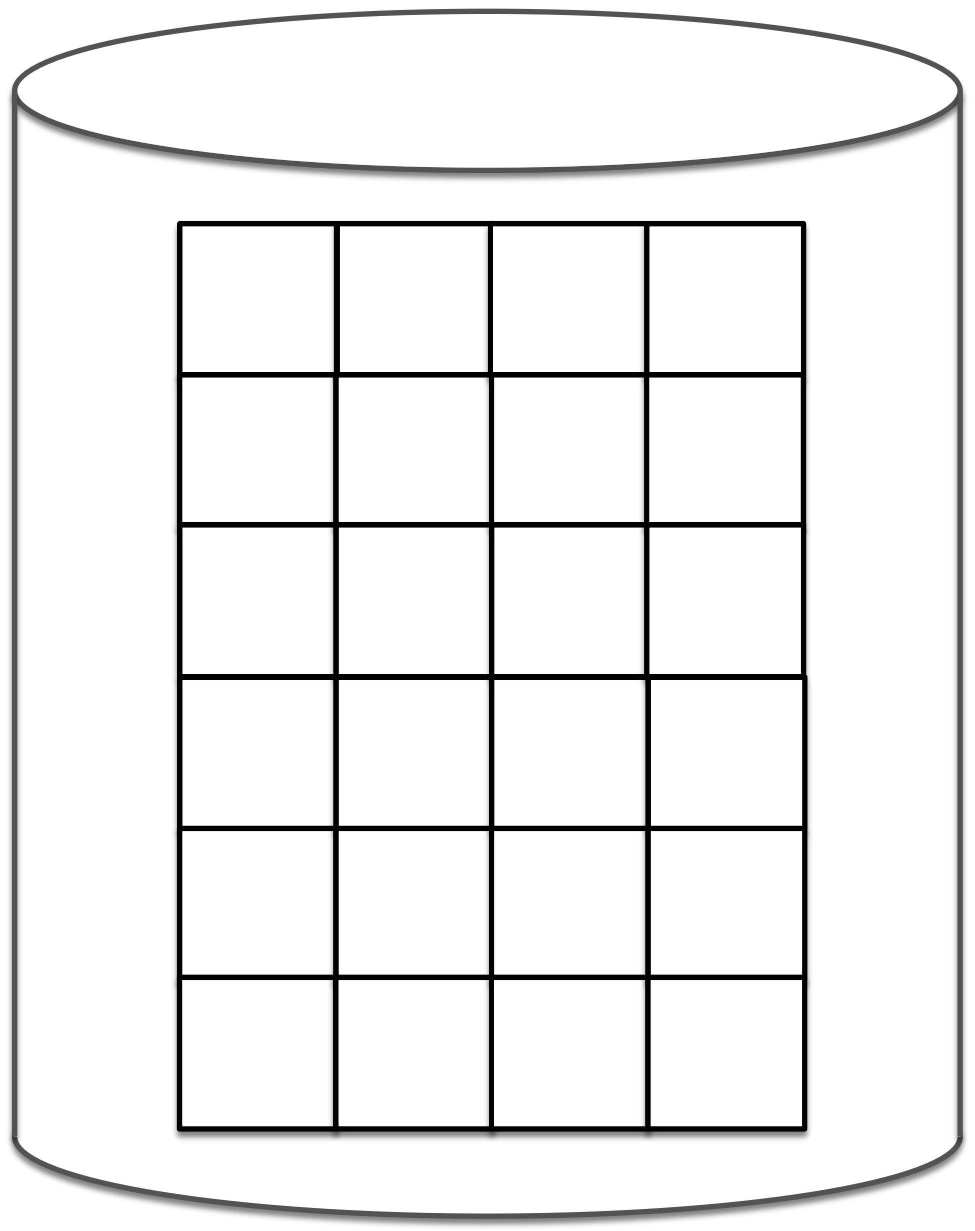
Largest Contributor and tester of Apache Kafka

- Founded in 2014
- Raised $84M from Benchmark, Index, Sequoia
- Transacting in 20 countries
- Commercial entities in US, UK, Germany, Australia

BENCHMARK Index Ventures SEQUOIA

Apache Kafka ♥ Databases





Stream Processing

SQL

Booked hotel, flight

Ordered a taxi

Paid money

Listened to music

Chatted with friends

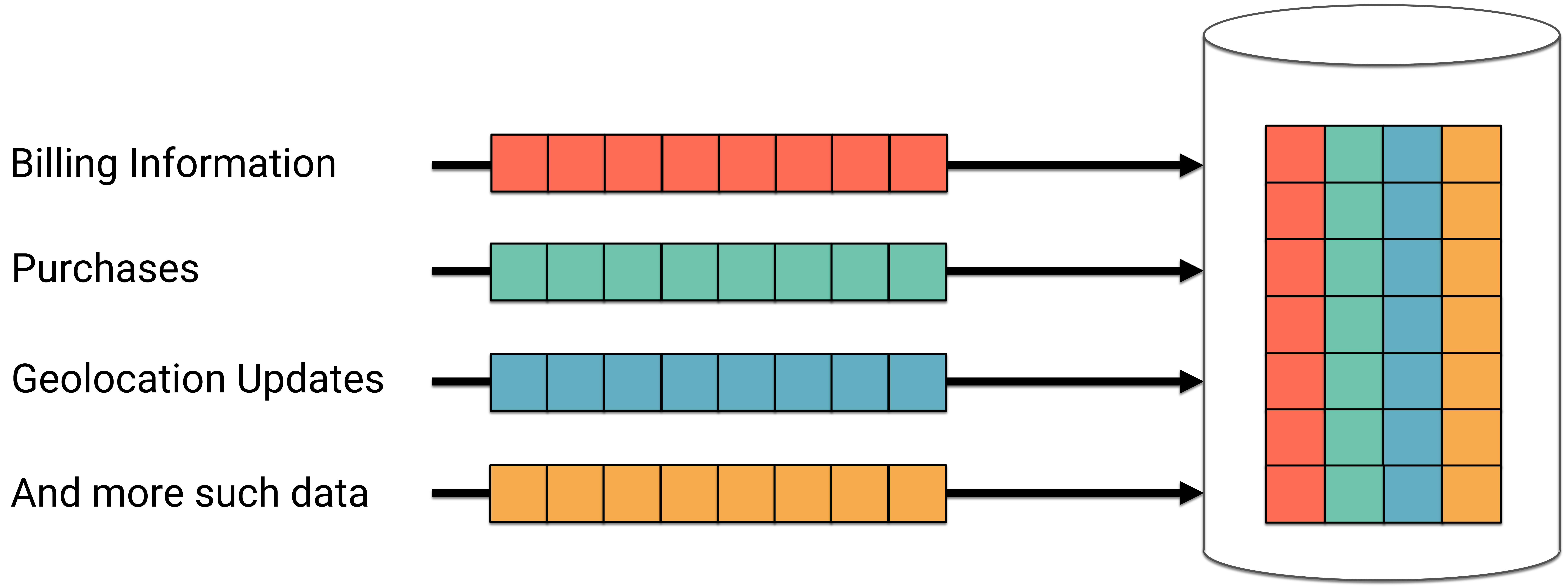
Played a video game

Read a newspaper

<add your example>

confluent

Motivating example



STREAMS of customer data (continuously flowing)

TABLE of customer profiles (continuously updated)

KSQL

is the

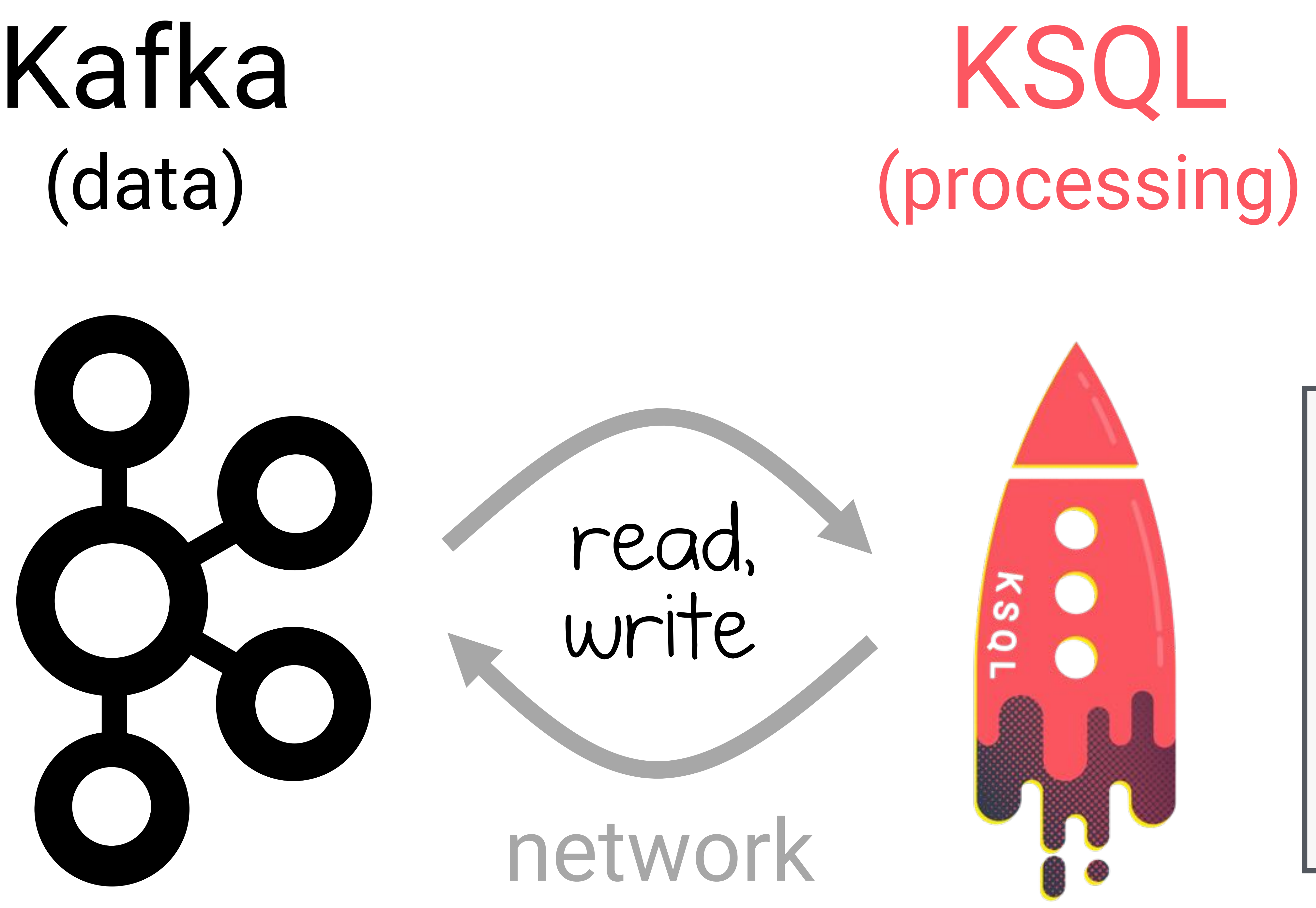
Streaming SQL Engine

for

Apache Kafka

confluent

KSQL is the Easiest Way to Process with Kafka



```
CREATE STREAM
CREATE TABLE
SELECT
…and more…
```

All you need is Kafka – no complex deployments of bespoke systems for stream processing!

KSQL is the Easiest Way to Process with Kafka

Free and Open Source

Zero Programming in Java, Scala

Elastic, Scalable, Fault-Tolerant, Distributed, S/M/L/XL

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

Runs Everywhere

Supports Streams and Tables

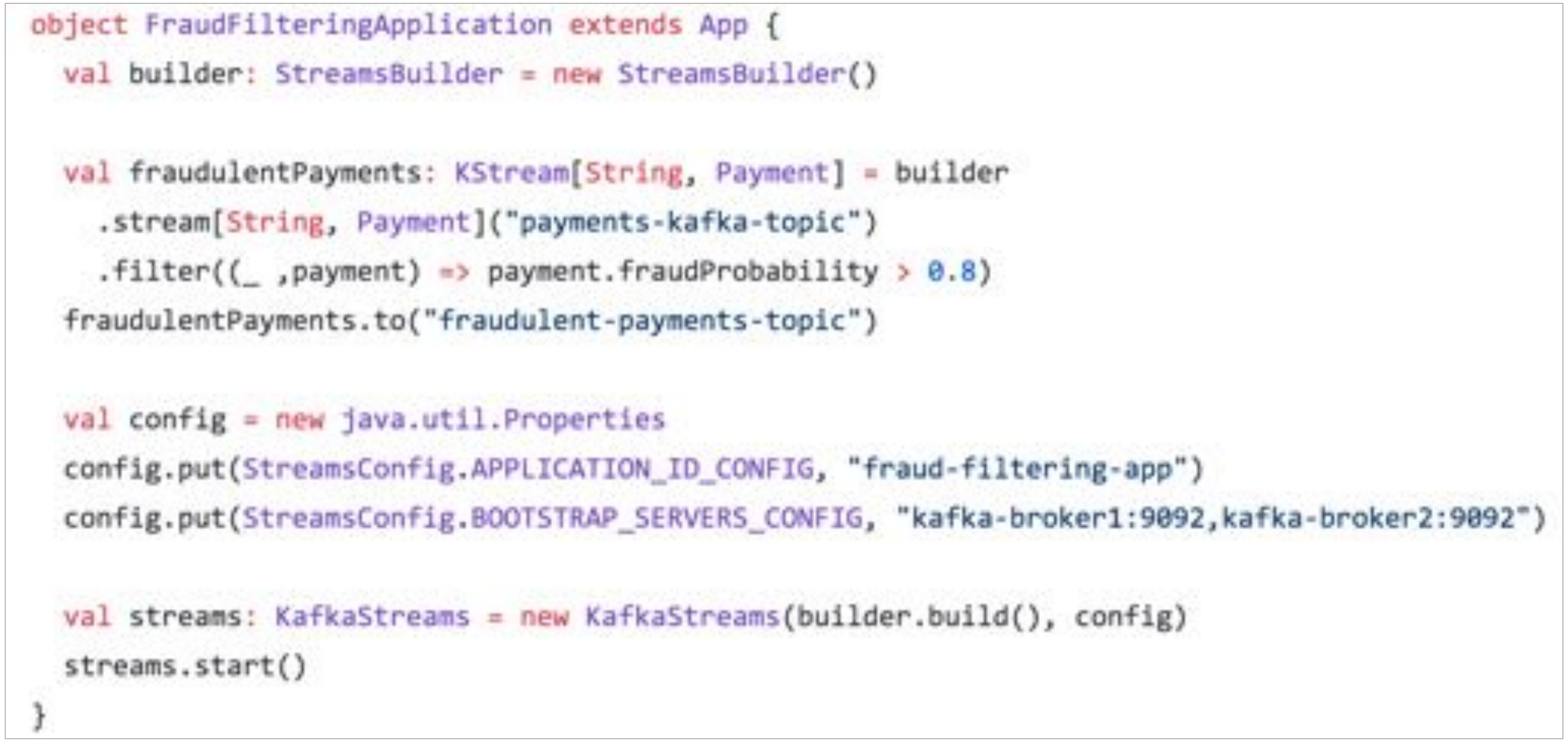
Exactly-Once Processing

Event-Time Processing

Kafka Security Integration

Stream processing with Kafka

Main Logic



```
object FraudFilteringApplication extends App {
  val builder: StreamsBuilder = new StreamsBuilder()

  val fraudulentPayments: KStream[String, Payment] = builder
    .stream[String, Payment]("payments-kafka-topic")
    .filter((_ ,payment) => payment.fraudProbability > 0.8)
  fraudulentPayments.to("fraudulent-payments-topic")

  val config = new java.util.Properties
  config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filtering-app")
  config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092,kafka-broker2:9092")

  val streams: KafkaStreams = new KafkaStreams(builder.build(), config)
  streams.start()
}
```

Example: Using **Kafka's Streams API** for writing elastic, scalable, fault-tolerant Java and Scala applications

Stream processing with Kafka

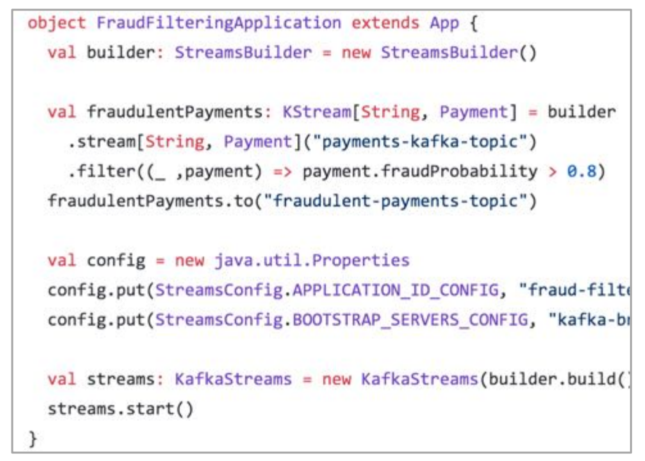
```
CREATE STREAM fraudulent_payments AS
  SELECT * FROM payments
  WHERE fraudProbability > 0.8;
```

Same example, now with **KSQL**.
Not a single line of Java or Scala code needed.

Easier, faster workflow

Kafka Streams API

write code in Java or Scala



```
object FraudFilteringApplication extends App {
  val builder: StreamsBuilder = new StreamsBuilder()

  val fraudulentPayments: KStream[String, Payment] = builder
    .stream[String, Payment]("payments-kafka-topic")
    .filter((_ ,payment) => payment.fraudProbability > 0.8)
  fraudulentPayments.to("fraudulent-payments-topic")

  val config = new java.util.Properties
  config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filt
  config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-b
  
  val streams: KafkaStreams = new KafkaStreams(builder.build(
  streams.start()
}
```

package app

run app

(1 or many instances)

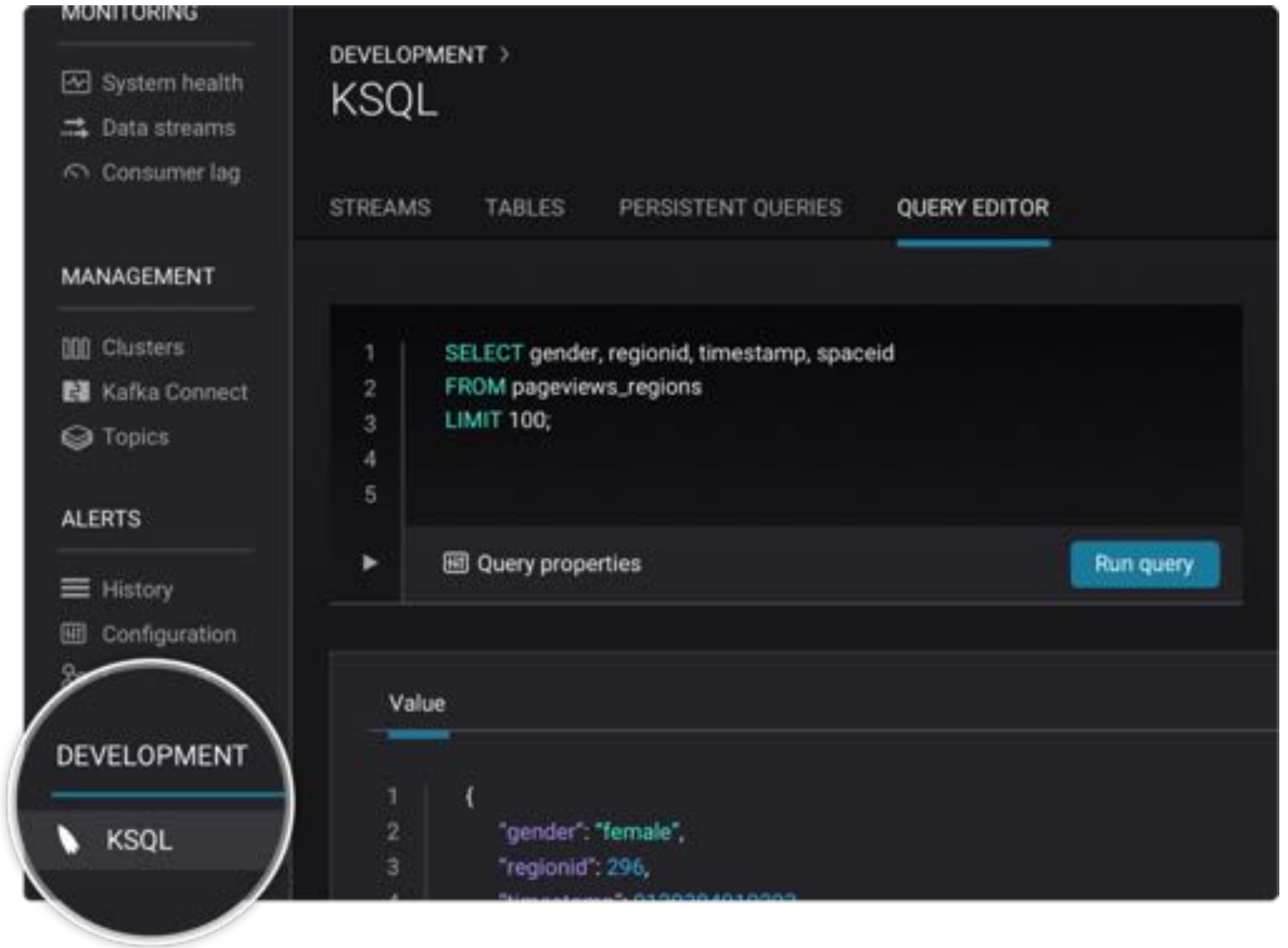
KSQL

```
ksql>
```

write (K)SQL

Interactive KSQL usage

```
ksql>
```



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews_regions
LIMIT 100;
```

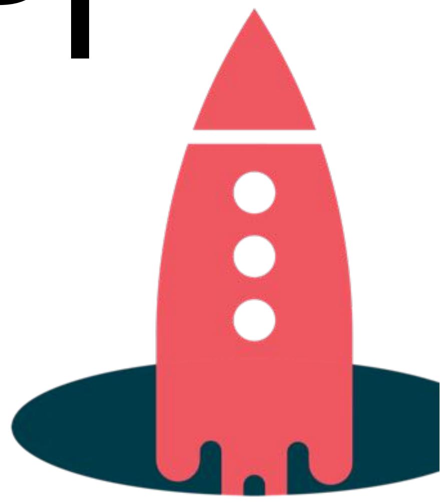
```
{
  "gender": "female",
  "regionid": 296,
```

```
POST /query
```

1 CLI

2 UI

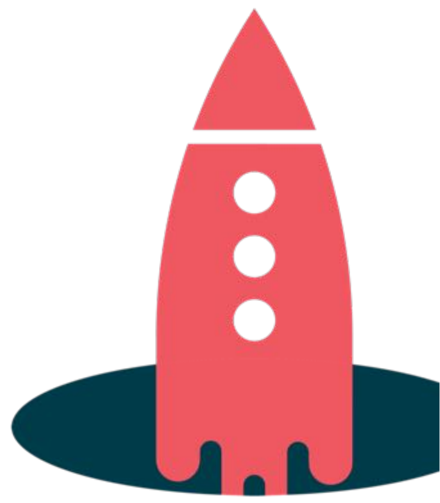
3 REST API



KSQL REST API example

```
POST /query HTTP/1.1

{
  "ksql": "SELECT * FROM users WHERE name LIKE ‘a%’;"
  "streamsProperties": {
    "your.custom.setting": "value"
  }
}
```

Here: run a query and stream back the results



what are some KSQL use cases?

KSQL for Data Exploration

An easy way to inspect data in Kafka

```
SHOW TOPICS;

PRINT 'my-topic' FROM BEGINNING;
```

```
SELECT page, user_id, status, bytes
FROM clickstream
WHERE user_agent LIKE 'Mozilla%';
```

KSQL for Data Enrichment

Join data from a variety of sources to see the full picture

```
CREATE STREAM enriched_payments AS
  SELECT payment_id, u.country, total
  FROM payments_stream p
  LEFT JOIN users_table u
         ON p.user_id = u.user_id;
```

1 Stream-table join

confluent

KSQL for Streaming ETL

Filter, cleanse, process data while it is moving

```
CREATE STREAM clicks_from_vip_users AS
  SELECT user_id, u.country, page, action
  FROM clickstream c
  LEFT JOIN users u ON c.user_id = u.user_id
  WHERE u.level ='Platinum';
```

KSQL for Anomaly Detection

Aggregate data to identify patterns or anomalies in real-time

```
CREATE TABLE possible_fraud AS
  SELECT card_number, COUNT(*)
  FROM authorization_attempts
  WINDOW TUMBLING (SIZE 30 SECONDS)
  GROUP BY card_number
  HAVING COUNT(*) > 3;
```

1 Aggregate data

2 … per 30sec windows

confluent

KSQL for Real-Time Monitoring

Derive insights from events (IoT, sensors, etc.) and turn them into actions

```
CREATE TABLE failing_vehicles AS
  SELECT vehicle, COUNT(*)
  FROM vehicle_telemetry_stream
  WINDOW TUMBLING (SIZE 1 MINUTE)
  WHERE event_type = 'ERROR’
  GROUP BY vehicle
  HAVING COUNT(*) >= 3;
```

KSQL for Data Transformation

Quickly make derivations of existing data in Kafka

```
CREATE STREAM clicks_by_user_id
  WITH (PARTITIONS=6,
        TIMESTAMP='view_time’
        VALUE_FORMAT='JSON') AS
  SELECT * FROM clickstream
  PARTITION BY user_id;
```

1 Re-partition the data

2 Convert data to JSON

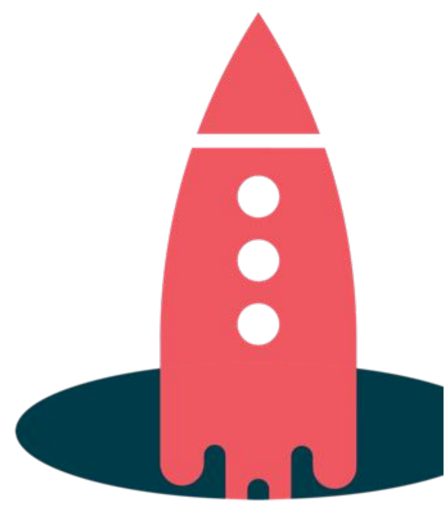
Where is KSQL not such a great fit?

Ad-hoc queries

- Because no indexes to facilitate efficient random lookups on arbitrary record fields

BI reports

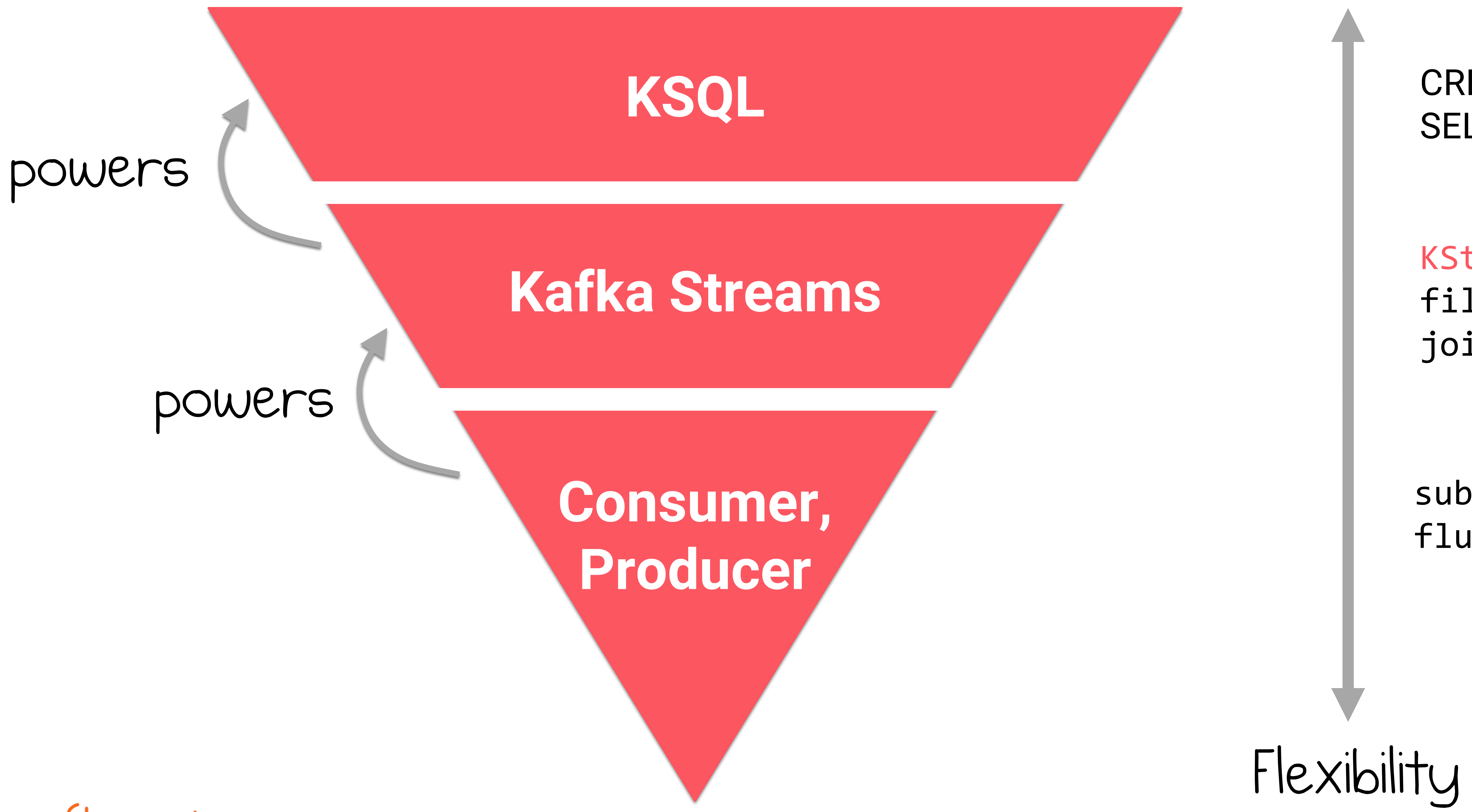
- Because no indexes
- No JDBC (most BI tools are not good with continuous results!)



How does KSQL work?

confluent

Shoulders of Streaming Giants



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

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

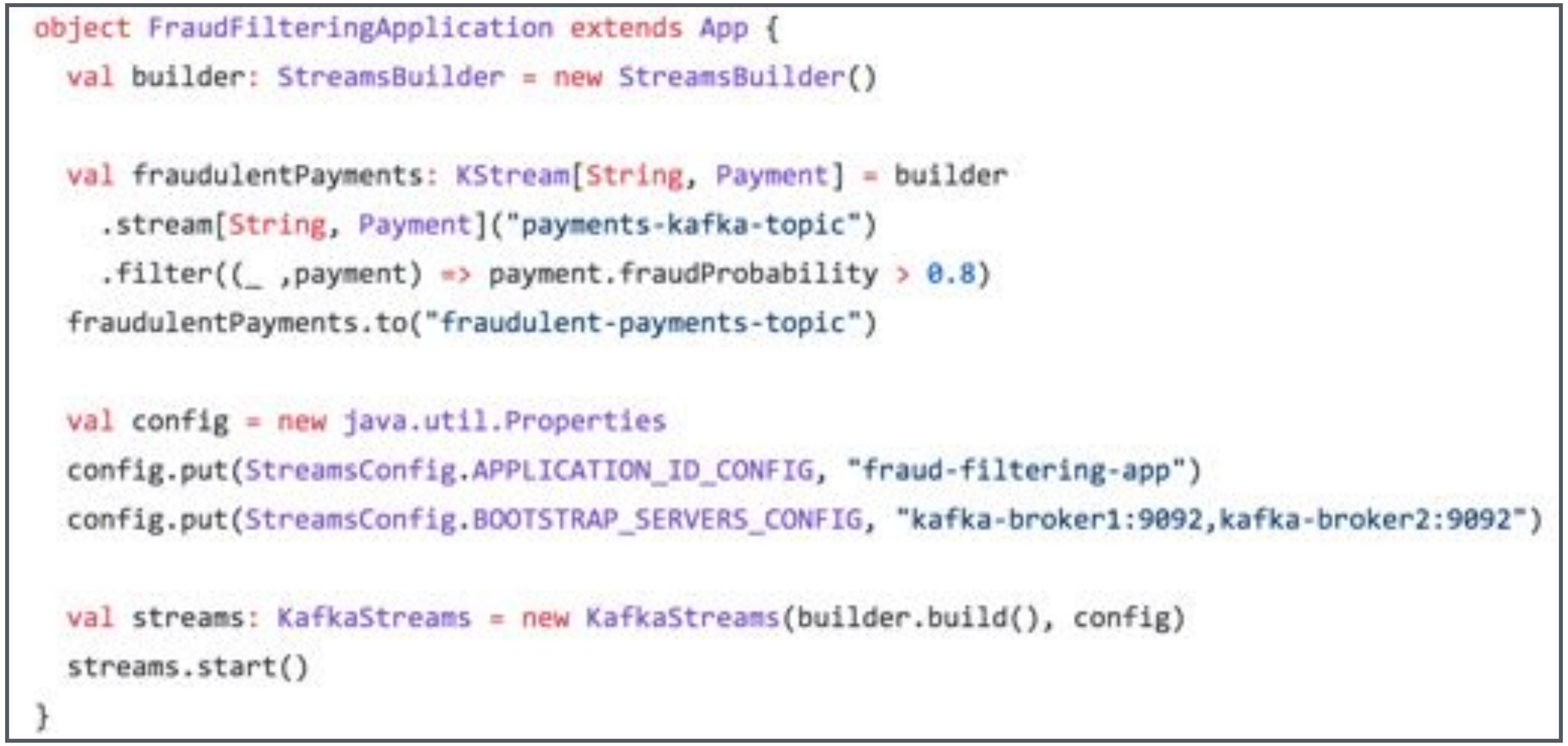
```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

Shoulders of Streaming Giants

KSQL

```
CREATE STREAM fraudulent_payments AS
  SELECT * FROM payments
  WHERE fraudProbability > 0.8;
```

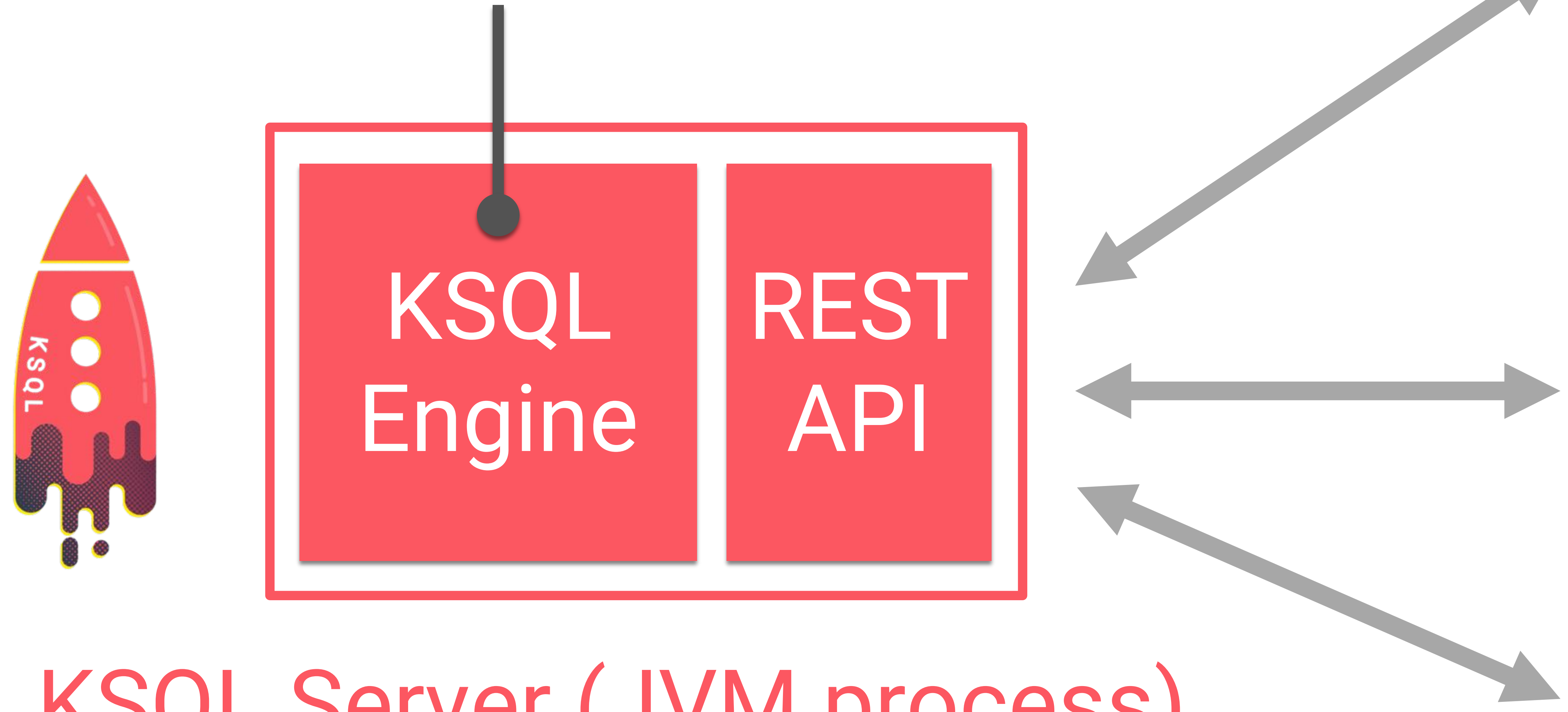
Kafka Streams



```
object FraudFilteringApplication extends App {
  val builder: StreamsBuilder = new StreamsBuilder()

  val fraudulentPayments: KStream[String, Payment] = builder
    .stream[String, Payment]("payments-kafka-topic")
    .filter((_ ,payment) => payment.fraudProbability > 0.8)
  fraudulentPayments.to("fraudulent-payments-topic")

  val config = new java.util.Properties
  config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filtering-app")
  config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092,kafka-broker2:9092")

  val streams: KafkaStreams = new KafkaStreams(builder.build(), config)
  streams.start()
}
```

KSQL Architecture

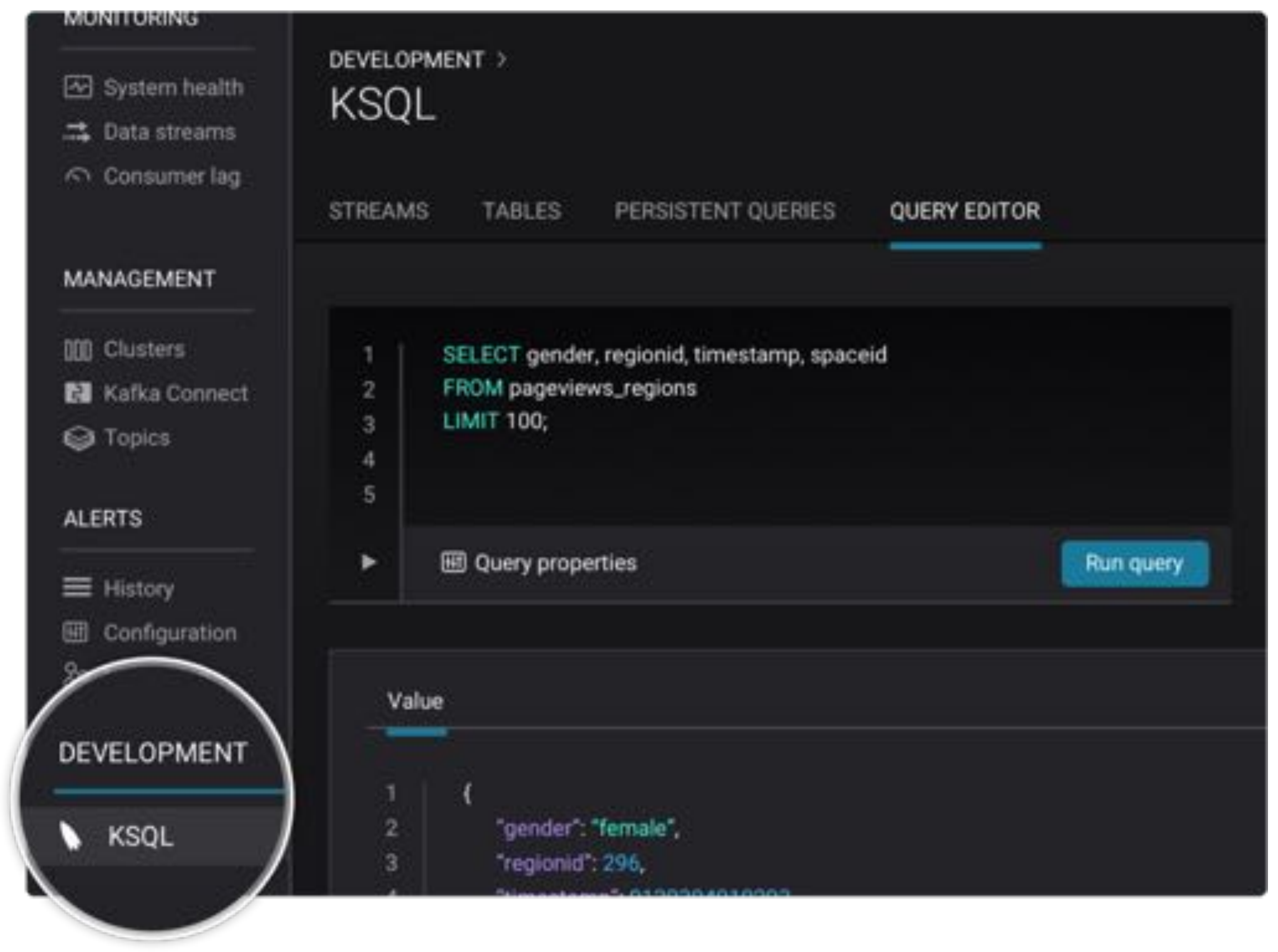
Processing happens here, powered by Kafka Streams



KSQL Server (JVM process)

```
$ ksql-server-start
```

Physical docker amazon web services ...



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews_regions
LIMIT 100;
```

```
{
    "gender": "female",
    "regionid": 296,
```

UI

```
ksql>
```

CLI

Programmatic access from Go, Python, .NET, Java, JavaScript, ...

confluent

Runs Everywhere, Viable for S/M/L/XL Use Cases

confluent cloud

amazon web services

Google Cloud Platform

Microsoft Azure

vmware

Physical

docker

kubernetes

openstack.

MESOS

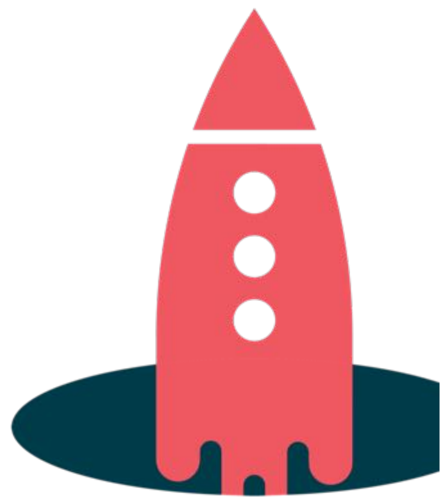
TERRAFORM

VAGRANT

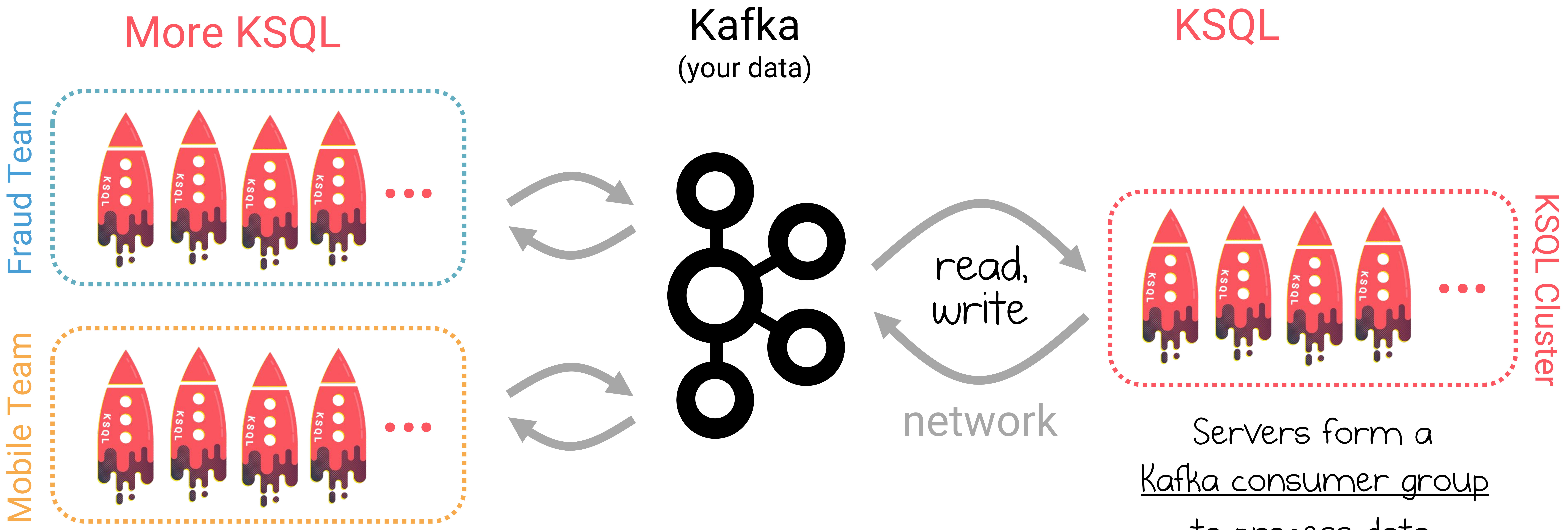
ANSIBLE

puppet labs

...and many more...



KSQL Architecture

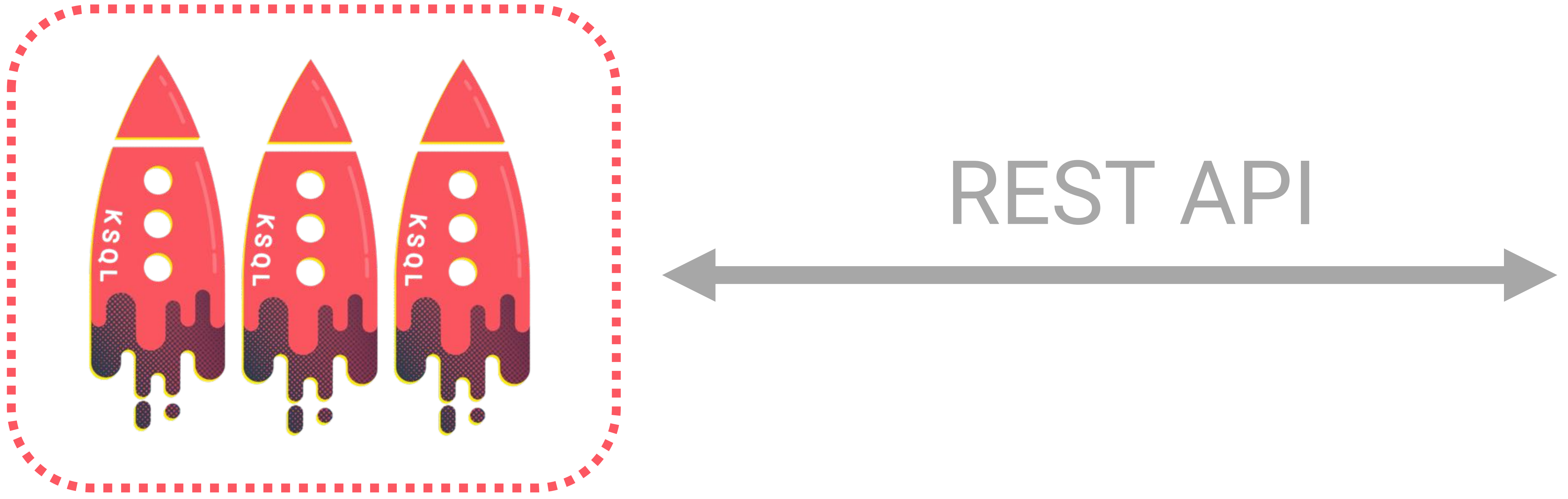


Servers form a <u>Kafka consumer group</u> to process data collaboratively

KSQL Interactive Usage

Start 1+ KSQL servers

Interact with
KSQL CLI, UI, etc.

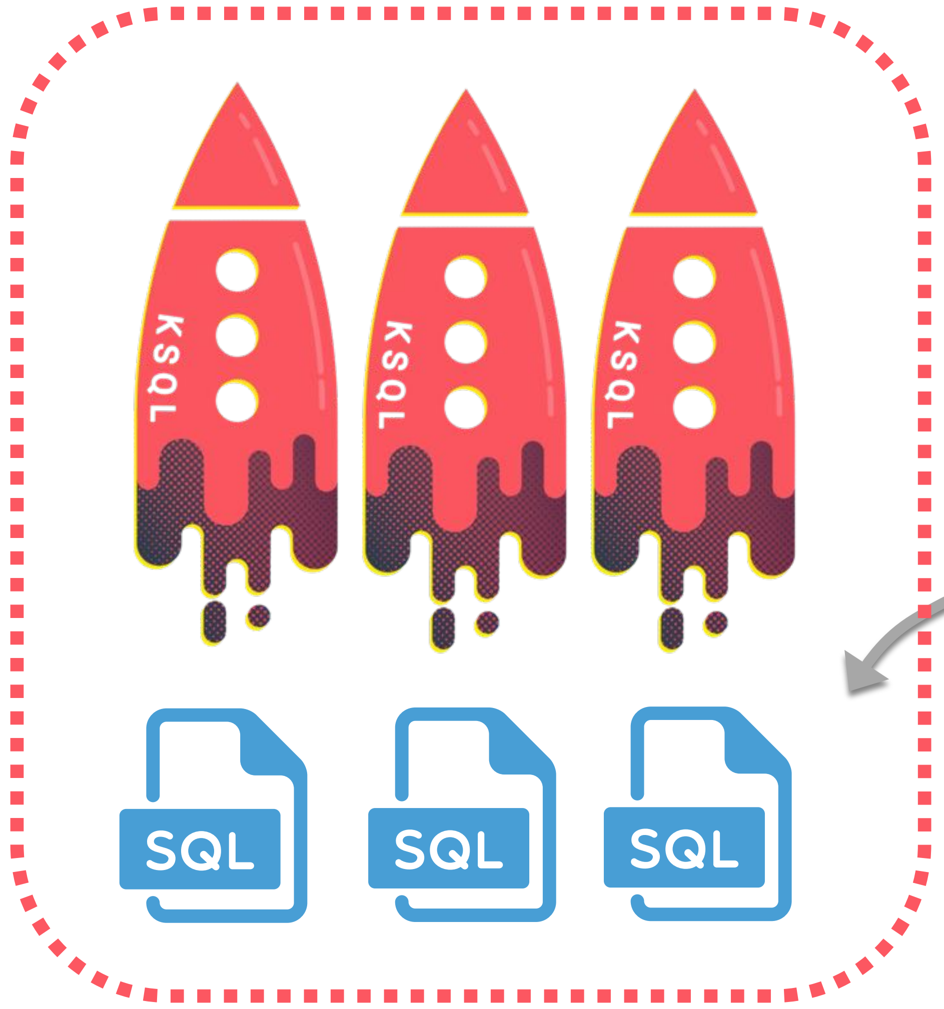


```
$ ksql-server-start
```

```
$ ksql http://ksql-server:8088
```

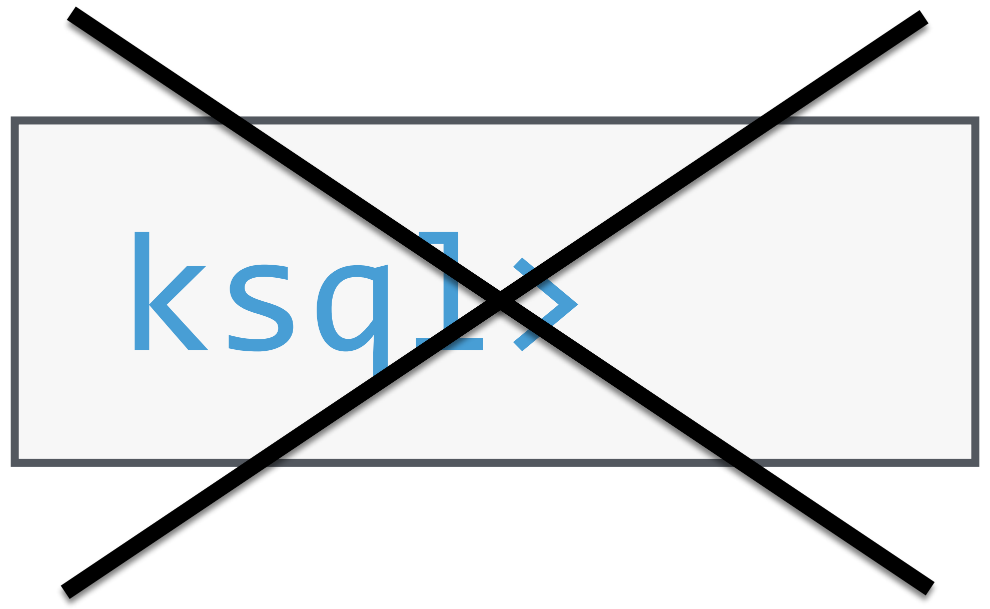
KSQL Headless, Non-Interactive Usage

Start 1+ KSQL servers with .sql file containing pre-defined queries.



Typically version controlled for auditing, rollbacks, etc.

REST API disabled

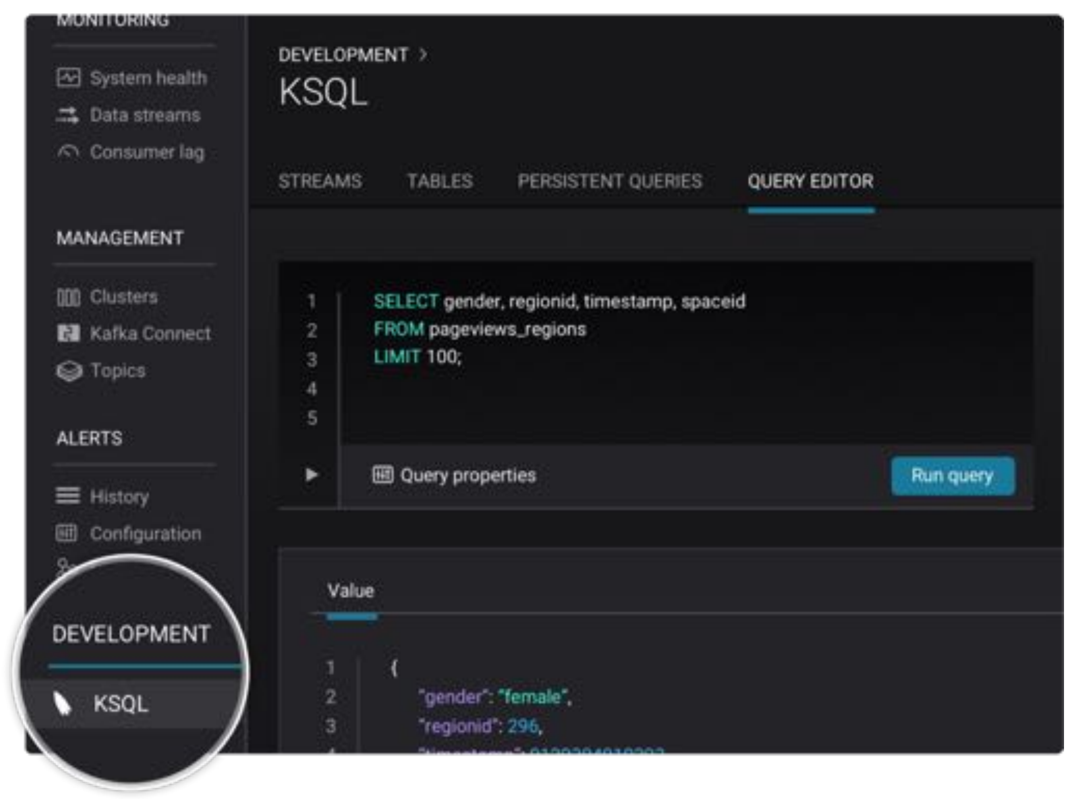


```
$ ksql-server-start --queries-file application.sql
```

Example Journey from Idea to Production

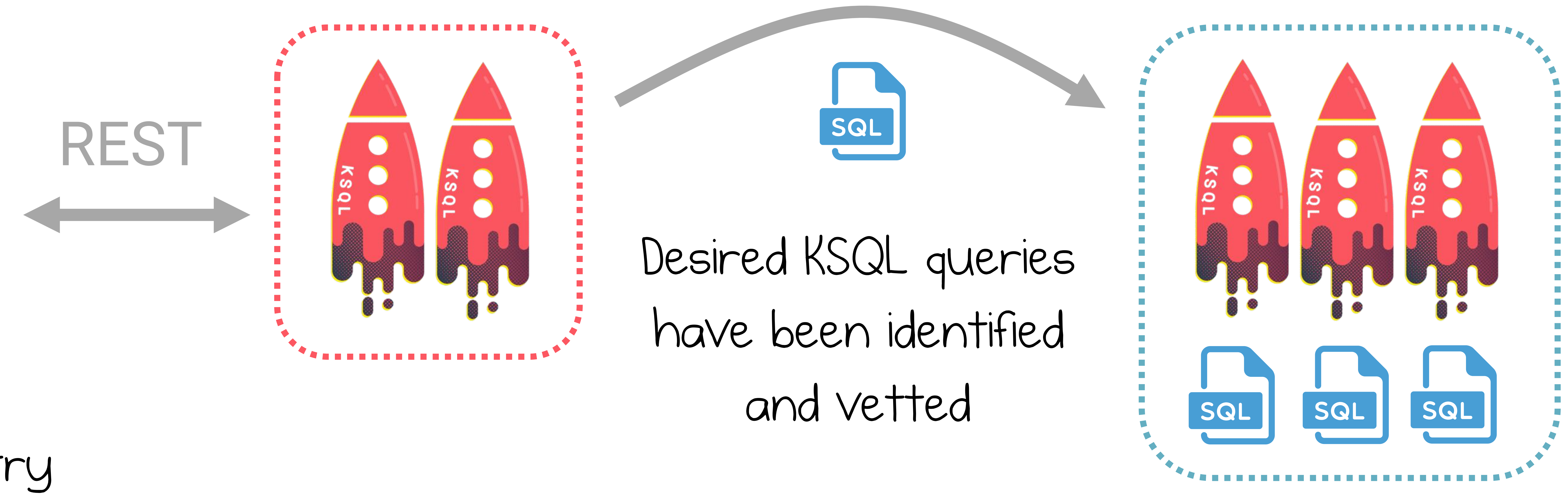
Interactive KSQL
for development and testing

Headless KSQL
for Production



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews_regions
LIMIT 100;
```

```
{
    "gender": "female",
    "regionid": 296,
```



Desired KSQL queries have been identified and vetted

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

The Stream-Table Duality

confluent

Stream-Table Duality

Stream

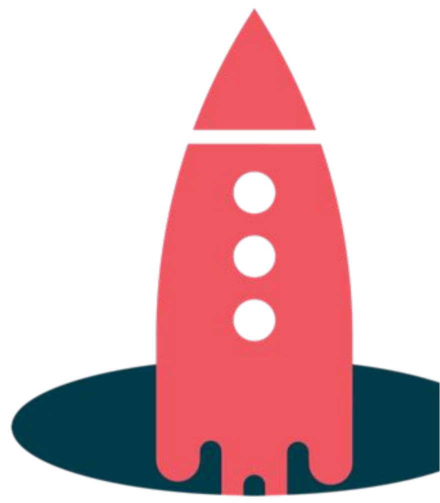
Table

```
CREATE STREAM enriched_payments AS
  SELECT payment_id, u.country, total
  FROM payments_stream p
  LEFT JOIN users_table u
         ON p.user_id = u.user_id;
```

```
CREATE TABLE failing_vehicles AS
  SELECT vehicle, COUNT(*)
  FROM vehicle_monitoring_stream
  WINDOW TUMBLING (SIZE 1 MINUTE)
  WHERE event_type = 'ERROR’
  GROUP BY vehicle
  HAVING COUNT(*) >= 3;
```

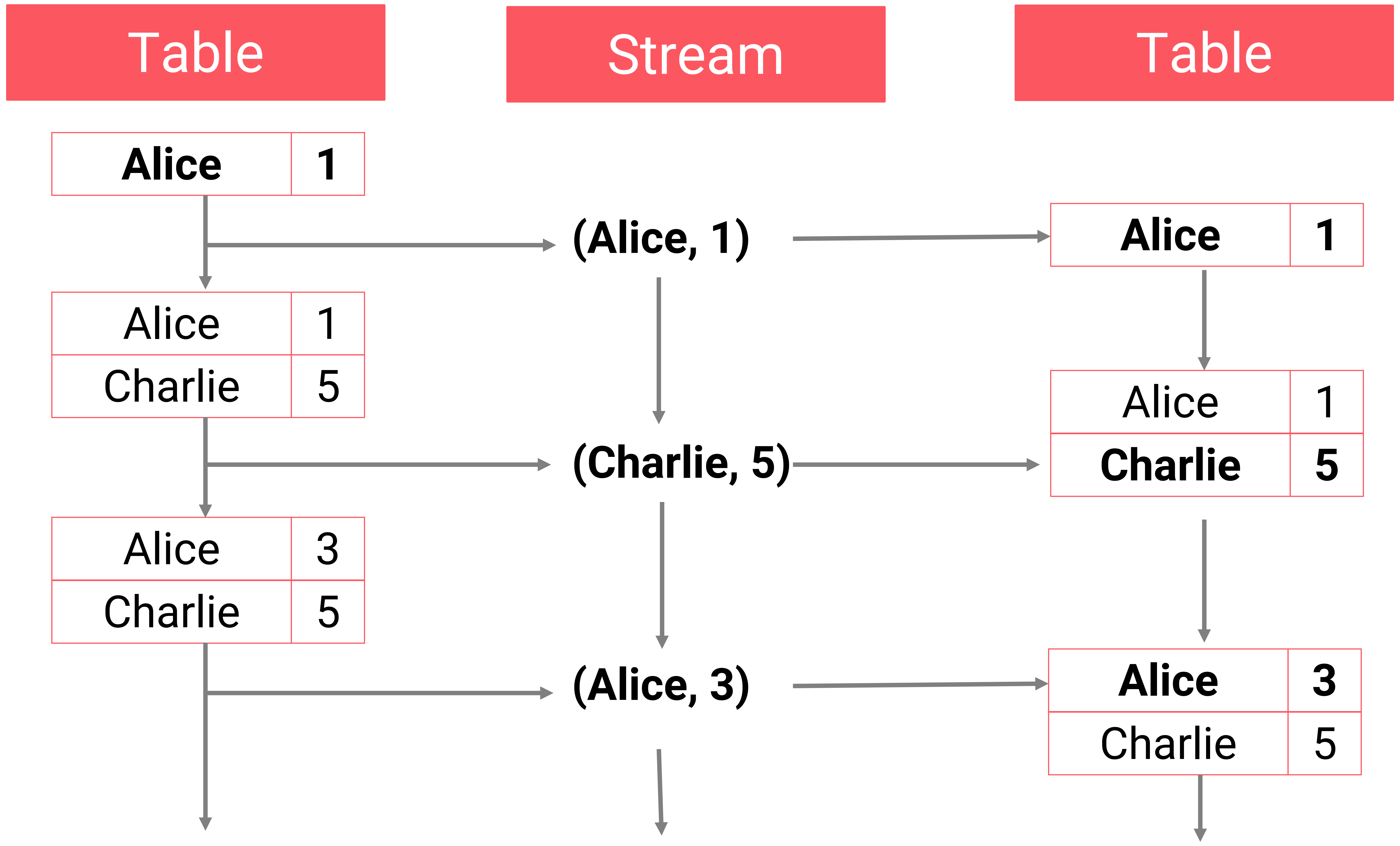
(from previous slides)

confluent



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

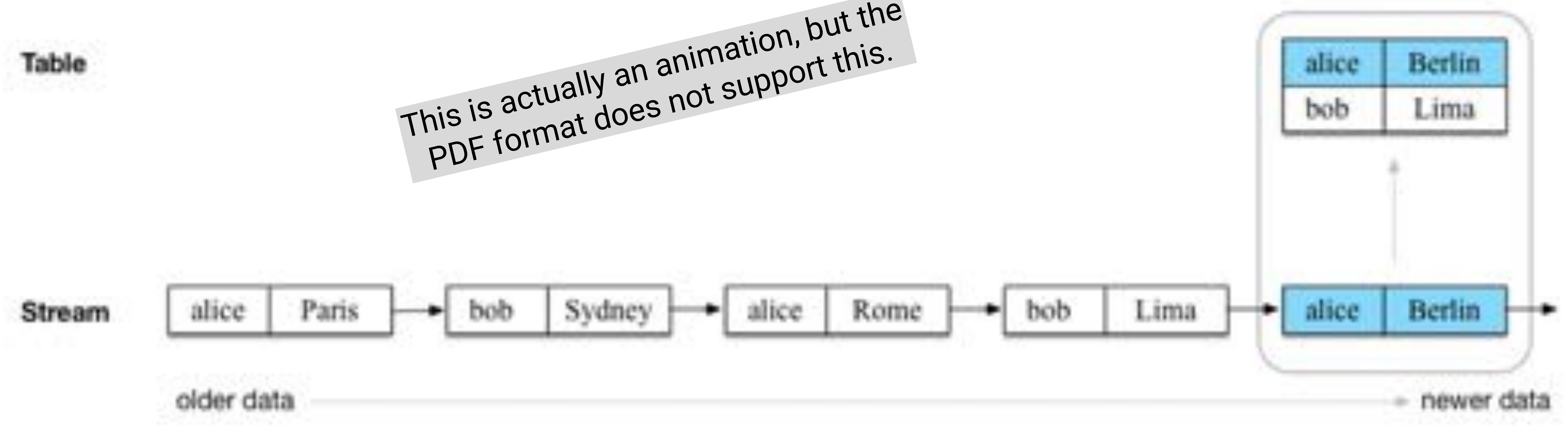
Stream-Table Duality



https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/

https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

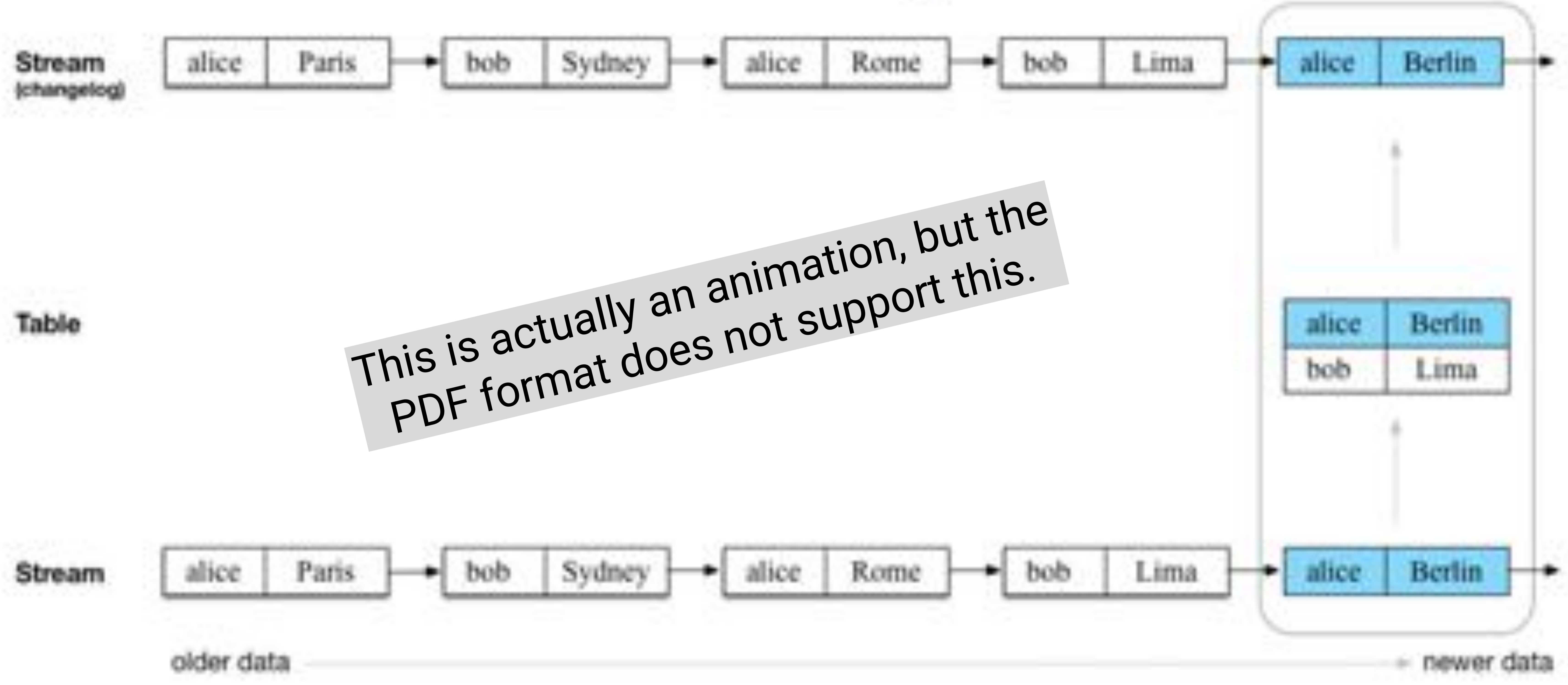
Stream-Table Duality



This is actually an animation, but the PDF format does not support this.

```
CREATE TABLE current_location_per_user
  WITH (KAFKA_TOPIC='input-topic’, ...);
```

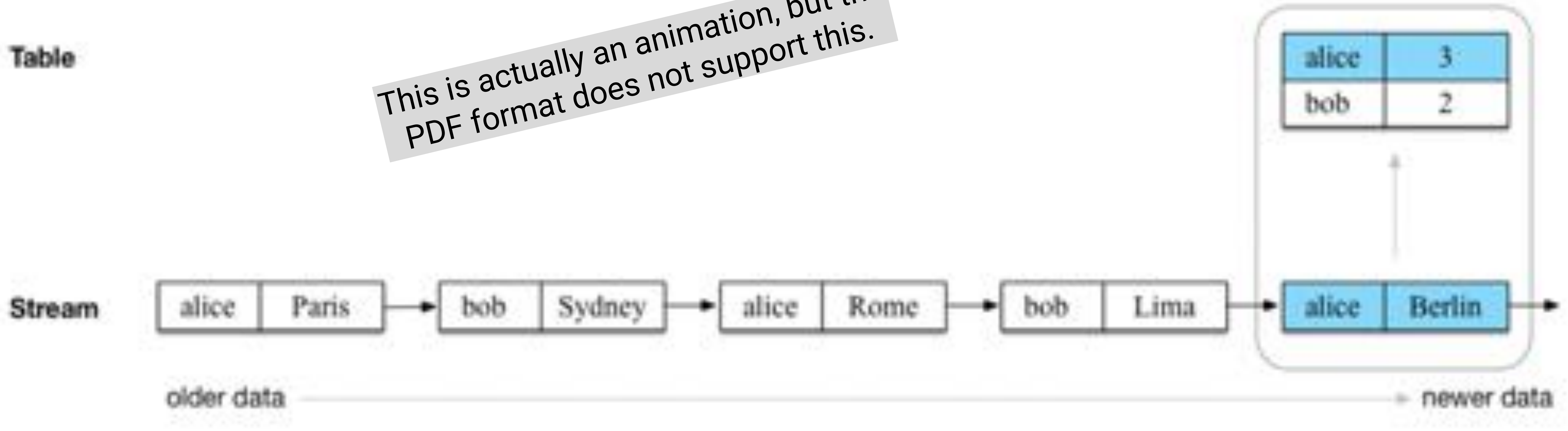
Stream-Table Duality



This is actually an animation, but the PDF format does not support this.

```
CREATE TABLE current_location_per_user
  WITH (KAFKA_TOPIC='input-topic’, ...);
```

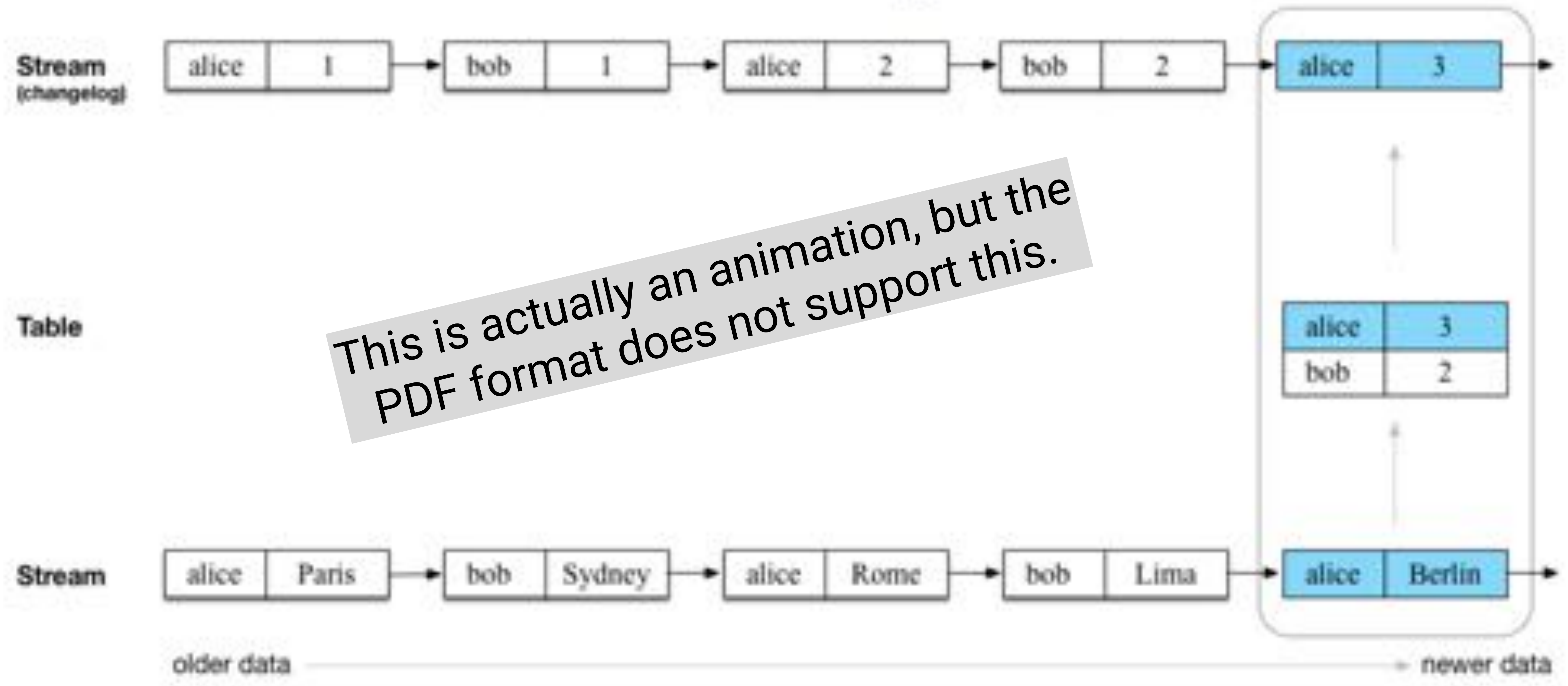
Stream-Table Duality



This is actually an animation, but the PDF format does not support this.

```
CREATE TABLE visited_locations_per_user AS
  SELECT username, COUNT(*)
  FROM location_updates
  GROUP BY username;
```

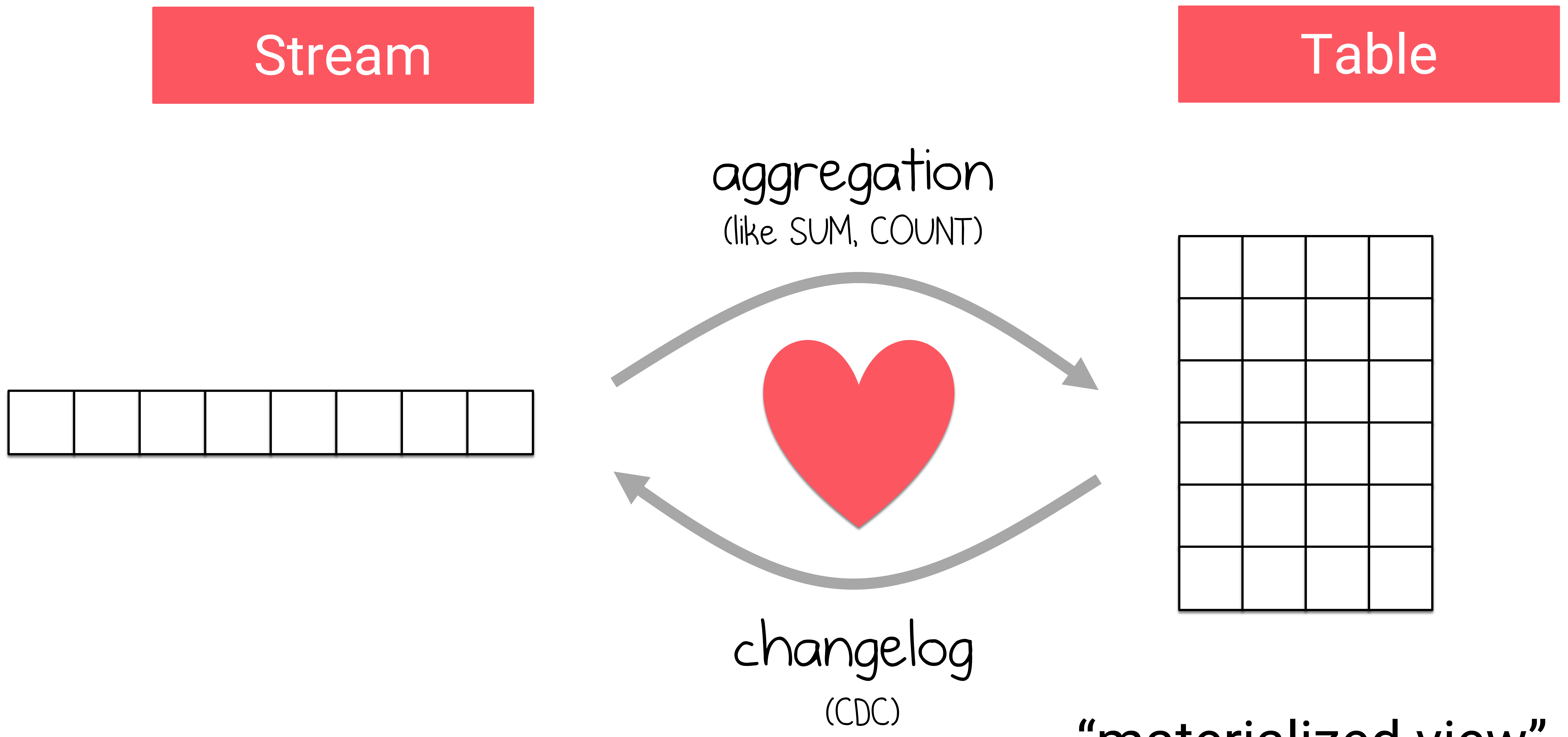
Stream-Table Duality



This is actually an animation, but the PDF format does not support this.

```
CREATE TABLE visited_locations_per_user AS
  SELECT username, COUNT(*)
  FROM location_updates
  GROUP BY username;
```

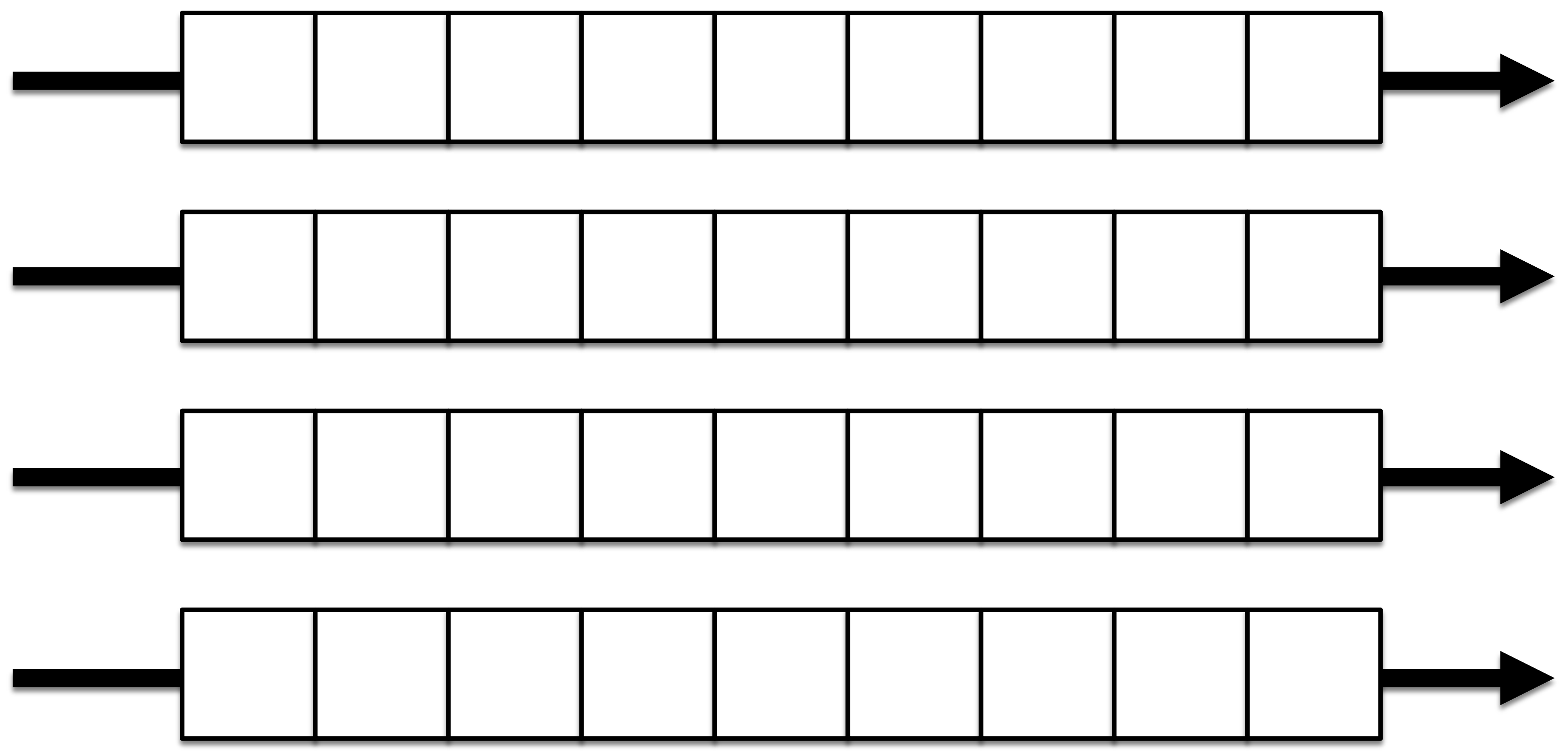
Stream-Table Duality

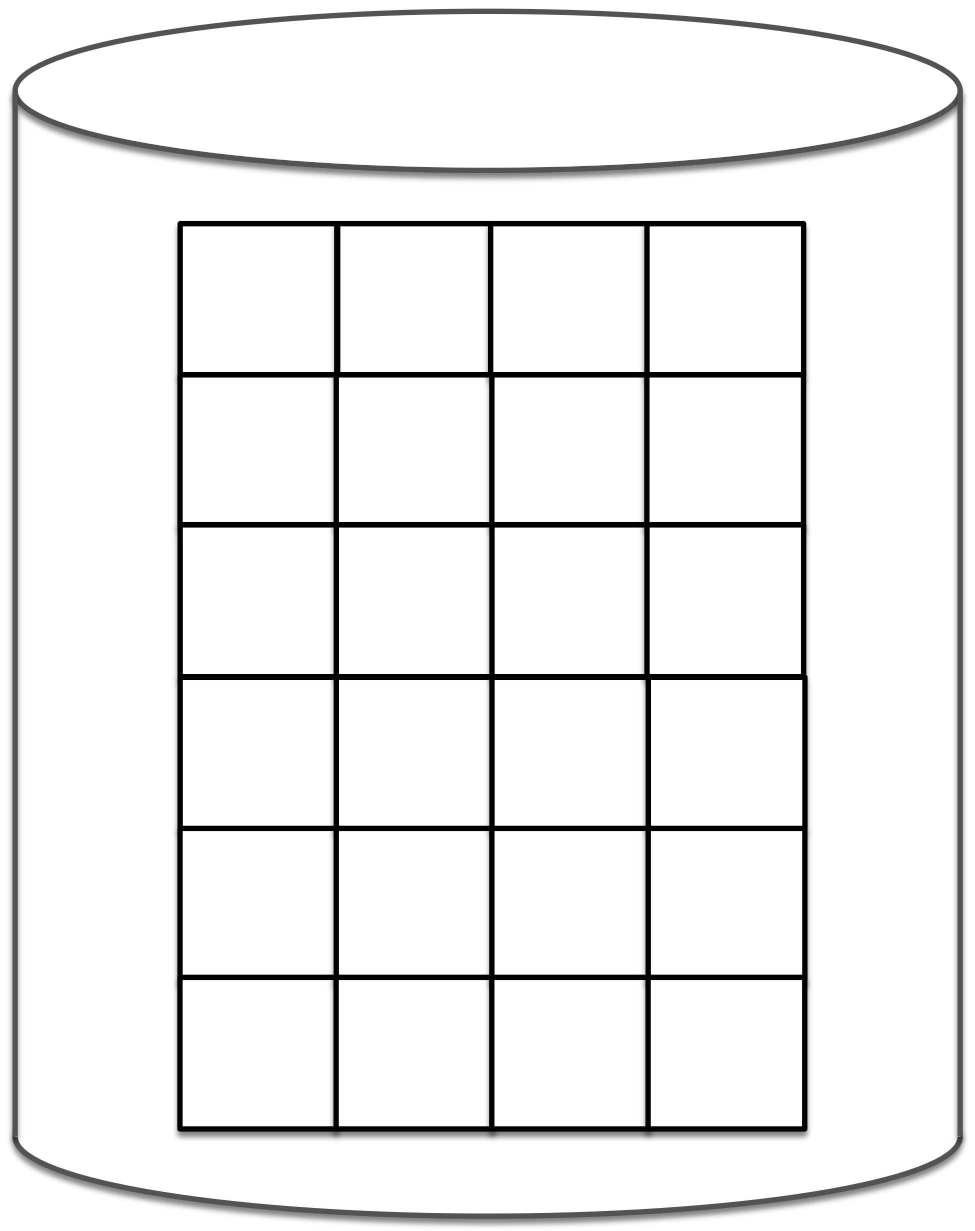


"materialized view"
of the stream

Stream-Table Duality

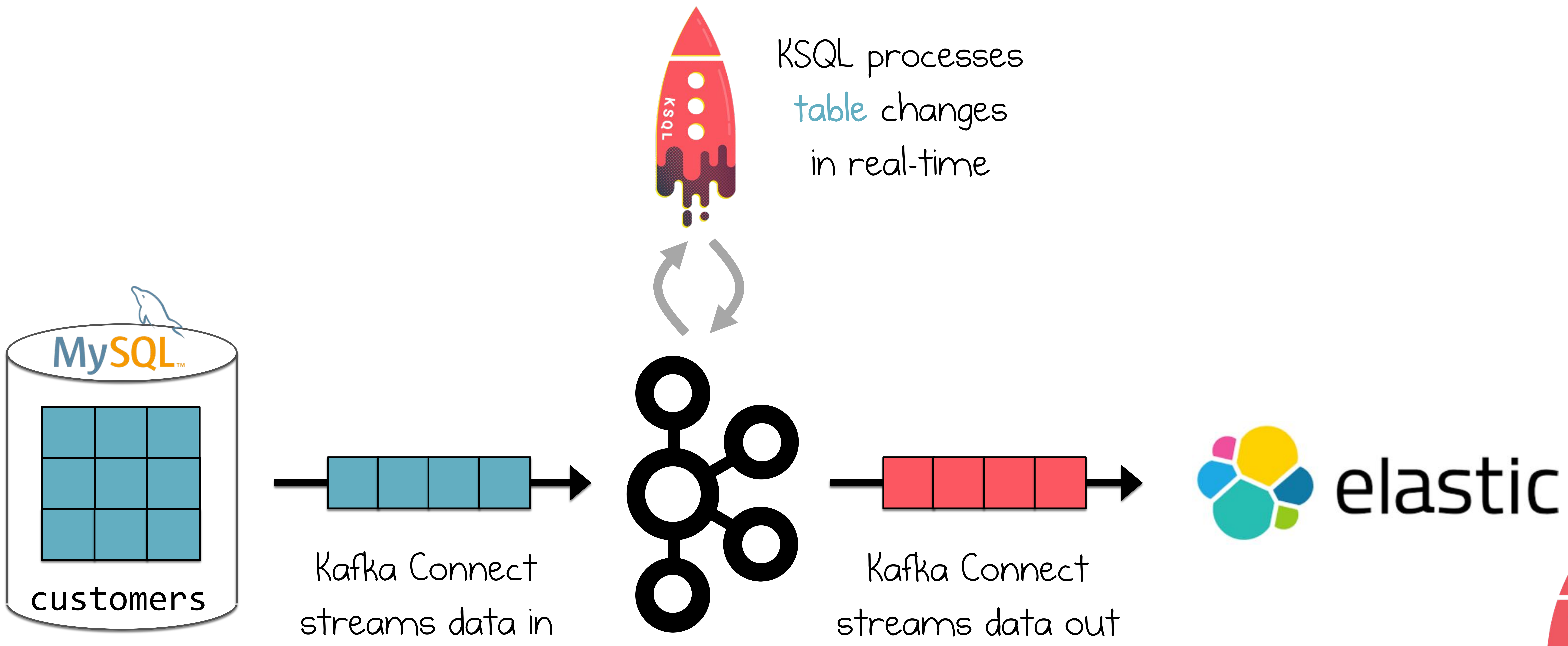
Apache Kafka ❤ Databases





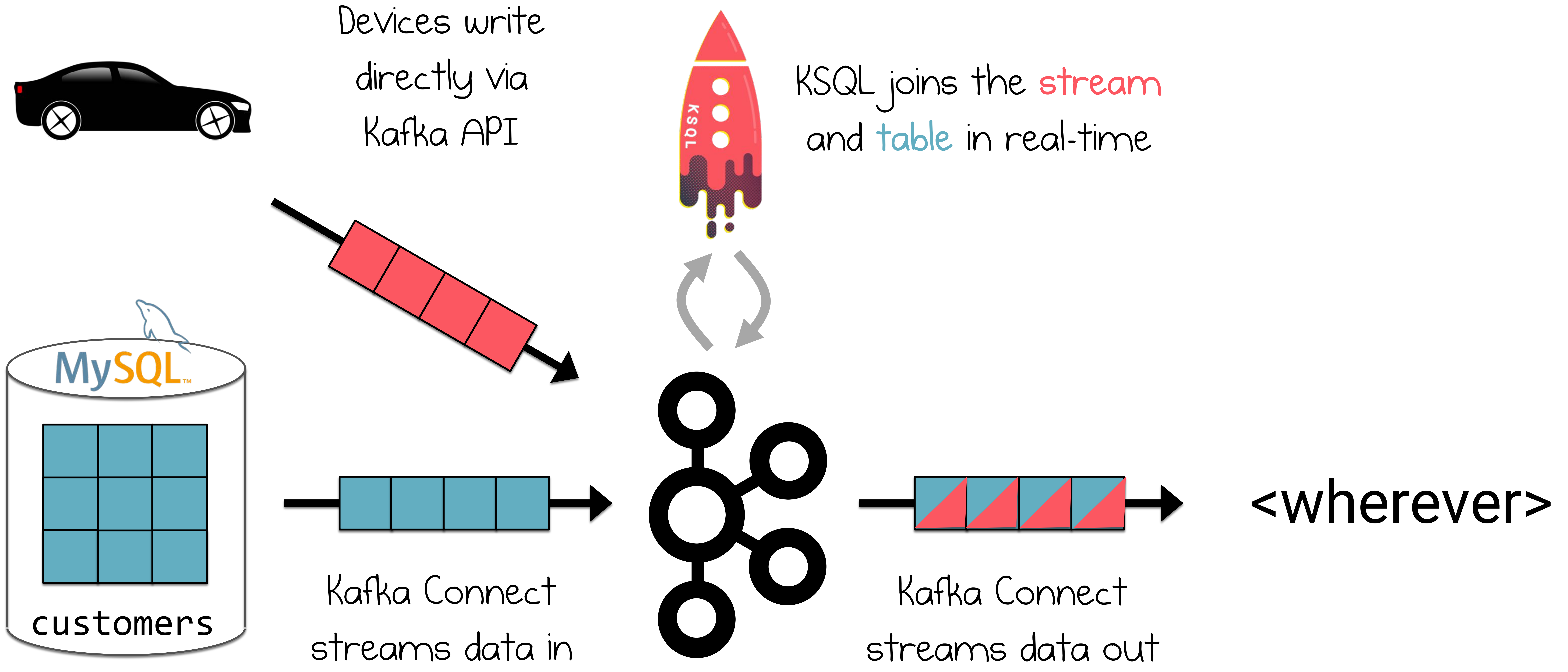
How you benefit from this as a **KSQL user**.

Example: CDC from DB via Kafka to Elastic


KSQL
KSQL processes table changes in real-time
MySQL
customers
Kafka Connect streams data in
Kafka Connect streams data out
elastic

confluent

Example: Real-time Data Enrichment

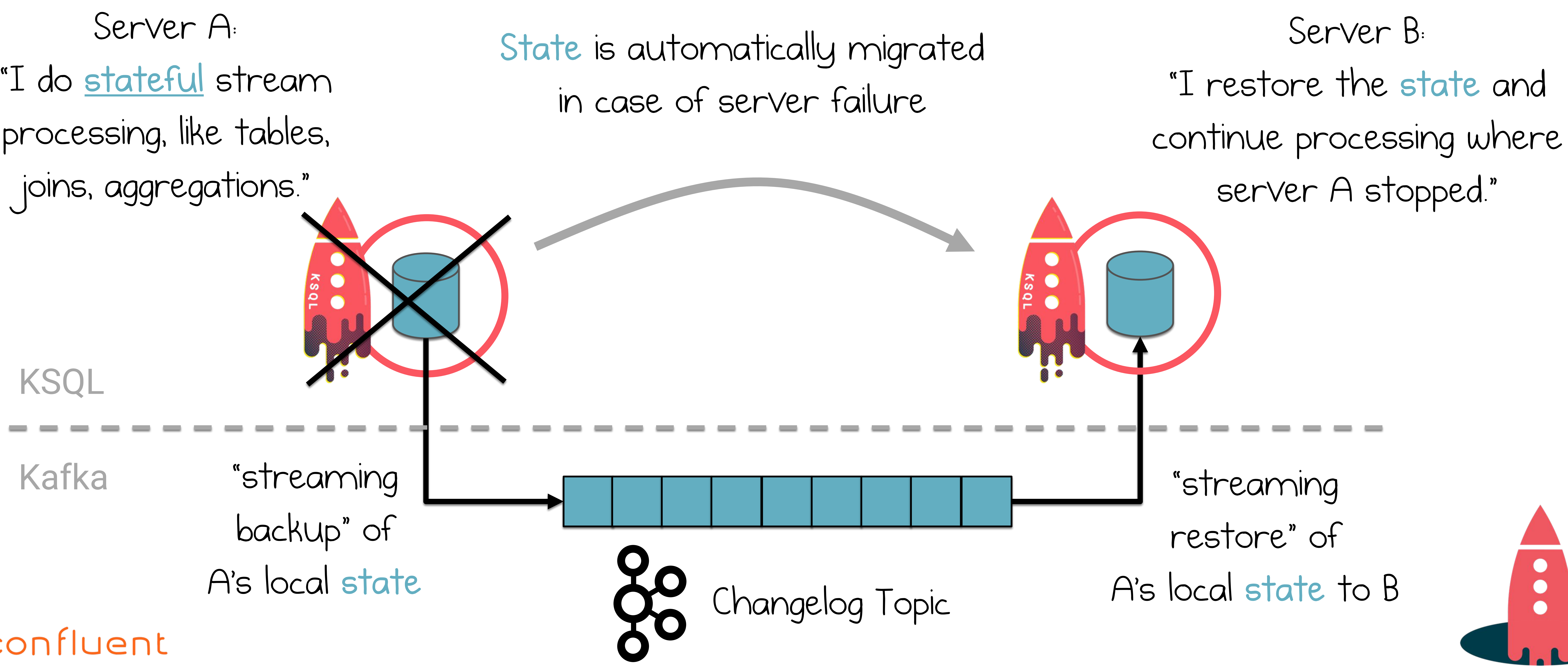


Devices write directly via Kafka API
KSQL
KSQL joins the stream and table in real-time
MySQL
customers
Kafka Connect streams data in
Kafka Connect streams data out
<wherever>

How **KSQL itself** benefits from this – a closer technical look

Fault-Tolerance, powered by Kafka

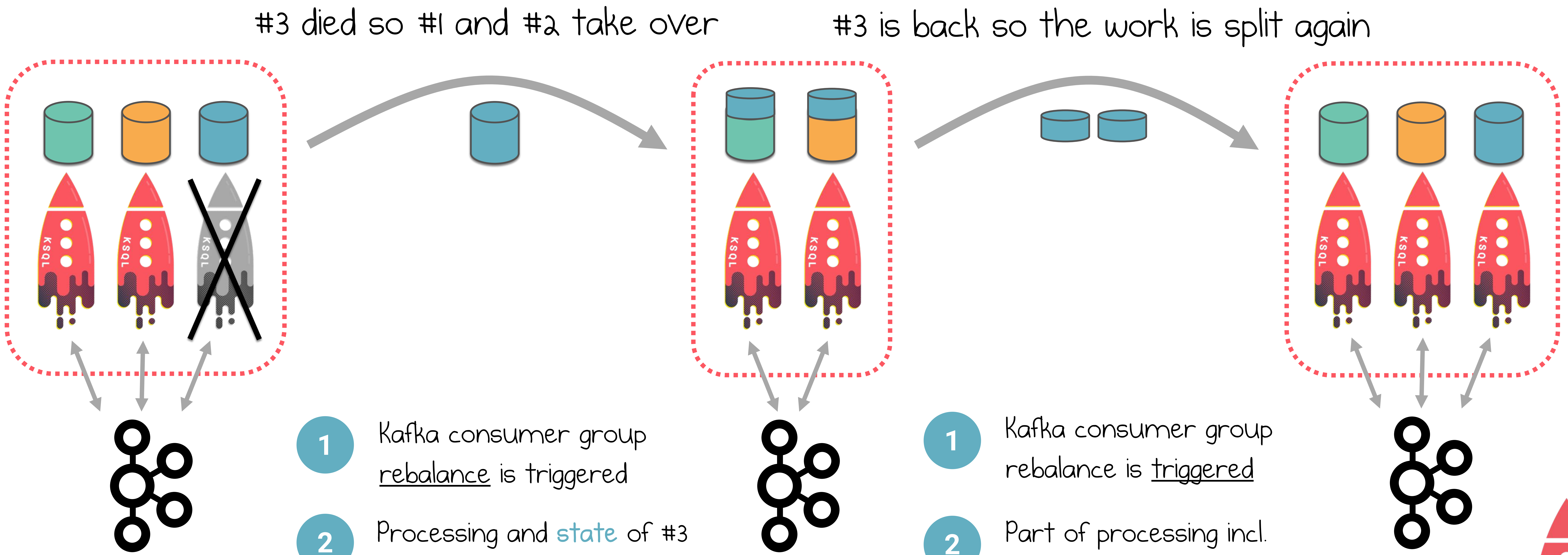
A key challenge of distributed stream processing is fault-tolerant state.


Server A:
"I do stateful stream processing, like tables, joins, aggregations."
State is automatically migrated in case of server failure
Server B:
"I restore the state and continue processing where server A stopped."
KSQL
KSQL
KSQL
Kafka
"streaming backup" of A's local state
Changelog Topic
"streaming restore" of A's local state to B

confluent

Fault-Tolerance, powered by Kafka

Processing fails over automatically, without data loss or miscomputation.


#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

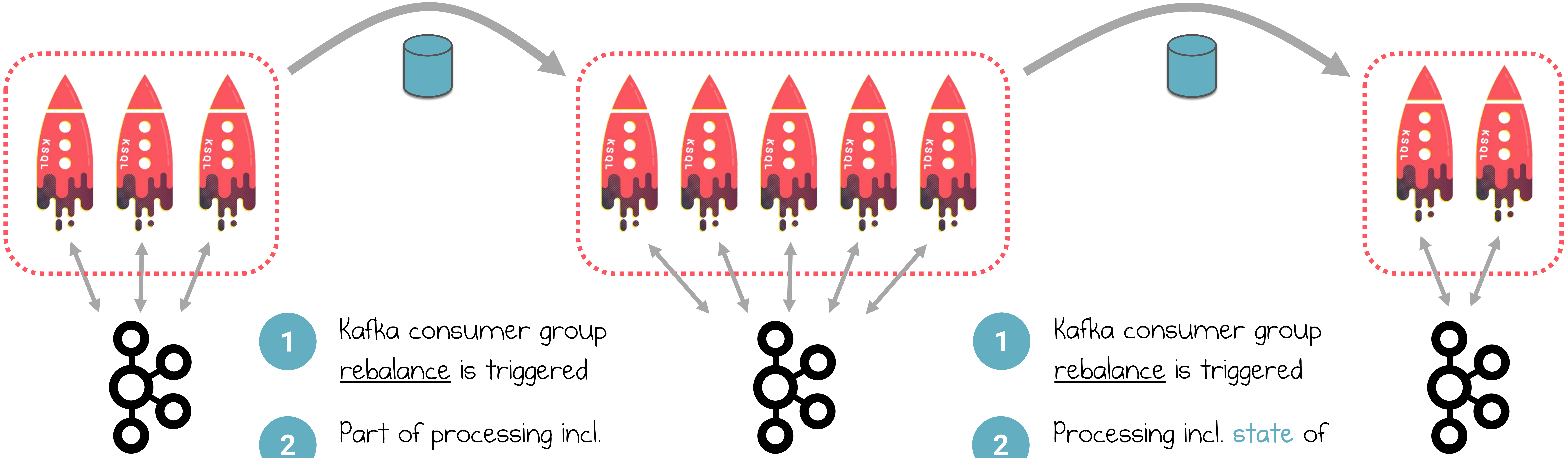
confluent

Elasticity and Scalability, powered by Kafka

You can add, remove, restart servers in KSQL clusters **during live operations.**

"We need more processing power!"

"Ok, we can scale down again."

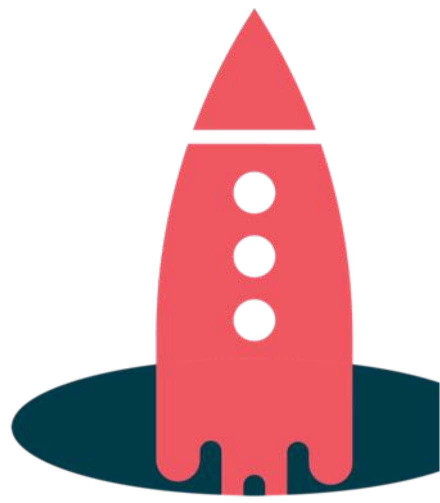


1 Kafka consumer group rebalance is triggered

2 Part of processing incl. state is migrated via Kafka to additional server processes

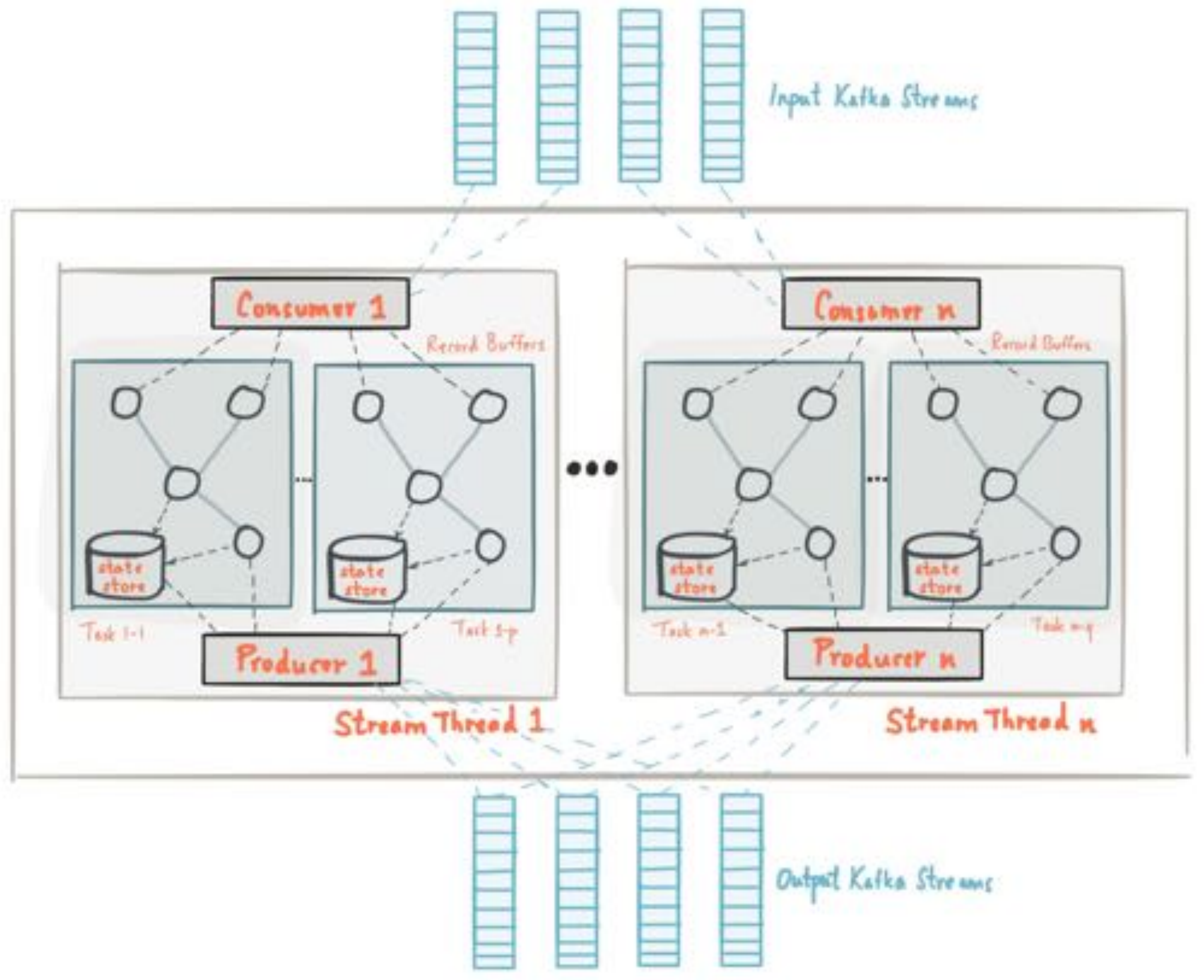
1 Kafka consumer group rebalance is triggered

2 Processing incl. state of stopped servers is migrated via Kafka to remaining servers



Want to take a deeper dive?

KSQL is built on top of Kafka Streams:
Read up on Kafka Streams' architecture including threading model, elasticity, fault-tolerance, state stores for stateful computation, etc. to learn more about how all this works behind the scenes.



https://kafka.apache.org/documentation/streams/architecture

confluent

Wrapping up

KSQL

is the

Streaming SQL Engine

for

Apache Kafka

confluent

KSQL is the Easiest Way to Process with Kafka

Free and Open Source

Zero Programming in Java, Scala

Elastic, Scalable, Fault-Tolerant, Distributed, S/M/L/XL

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

Runs Everywhere

Supports Streams and Tables

Exactly-Once Processing

Event-Time Processing

Kafka Security Integration

confluent

Where to go from here

http://confluent.io/ksql

https://github.com/confluentinc/ksql

https://slackpass.io/confluentcommunity #ksql