KSQL

Open-source streaming for Apache Kafka

KSQL and Kafka Streams in 3 minutes

In a nutshell

KSQL

The streaming SQL engine
for Apache Kafka® to write
real-time applications in SQL

kafka
streams

Apache Kafka® library to write
real-time applications and
microservices in Java and Scala

KSQL

is the

Streaming

SQL Engine

for

Apache Kafka

Hello, Streaming World

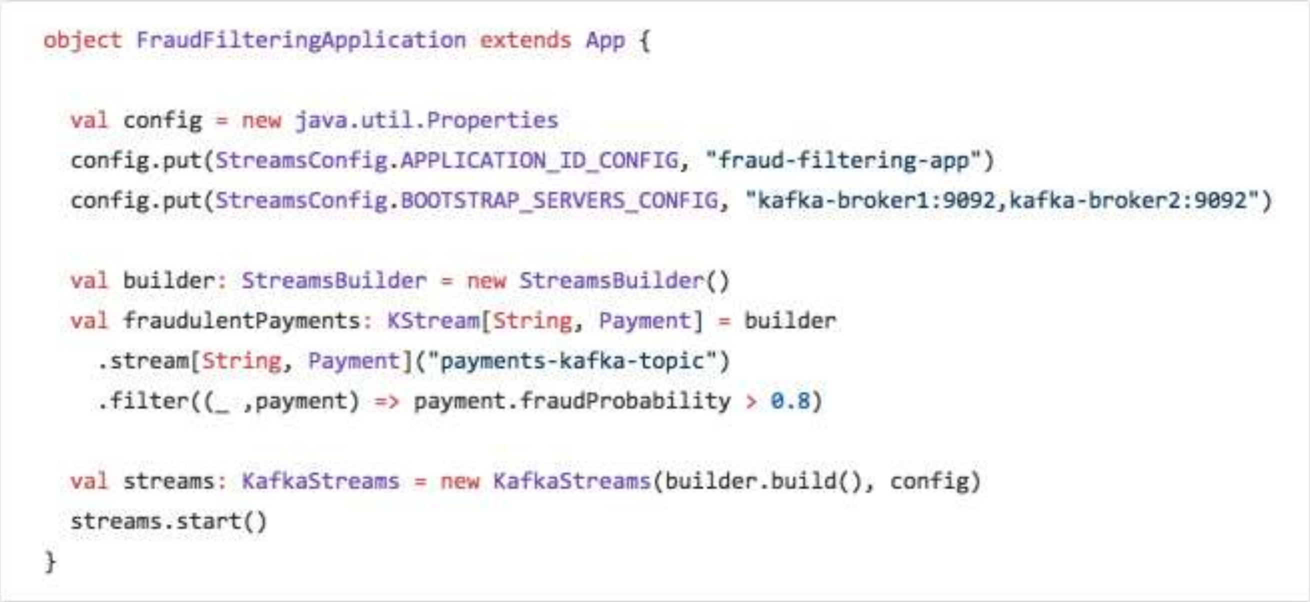
KSQL

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

You write ***only*** SQL. No Java, Python, or other boilerplate to wrap around it!

But you can create KSQL User Defined Functions in Java, if you want to.

kafka streams



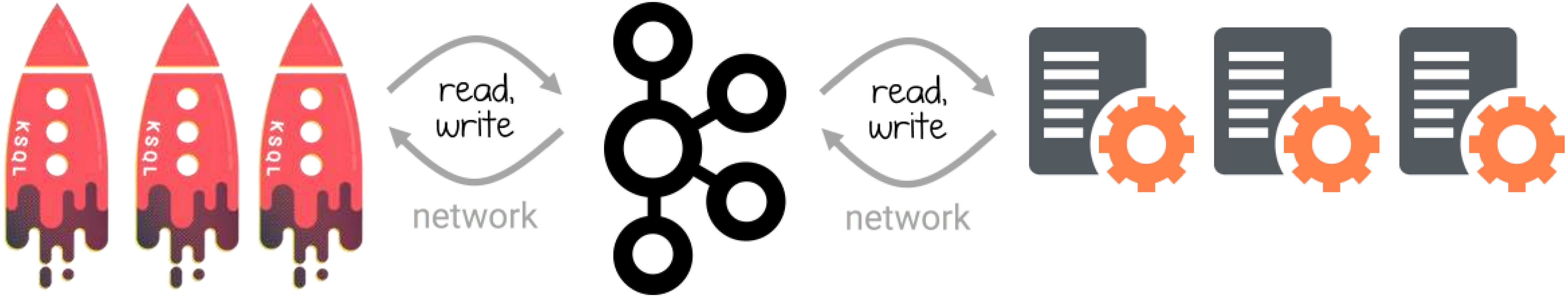
```
object FraudFilteringApplication extends App {

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

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

Interaction with Kafka

KSQL
(processing)

Kafka
(data)

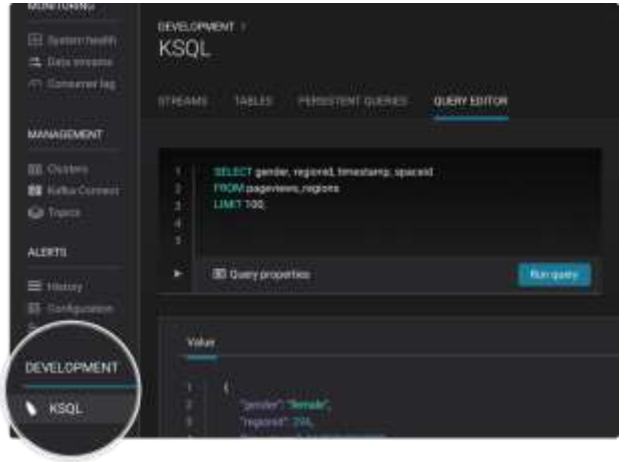
JVM application
with Kafka Streams (processing)



Does not run on
Kafka brokers

Does not run on
Kafka brokers

KSQL can be used interactively + programmatically



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews, regions
LIMIT 100;
```

```
{
  "gender": "female",
  "regionid": 29L,
```

1 UI

```
ksql>
```

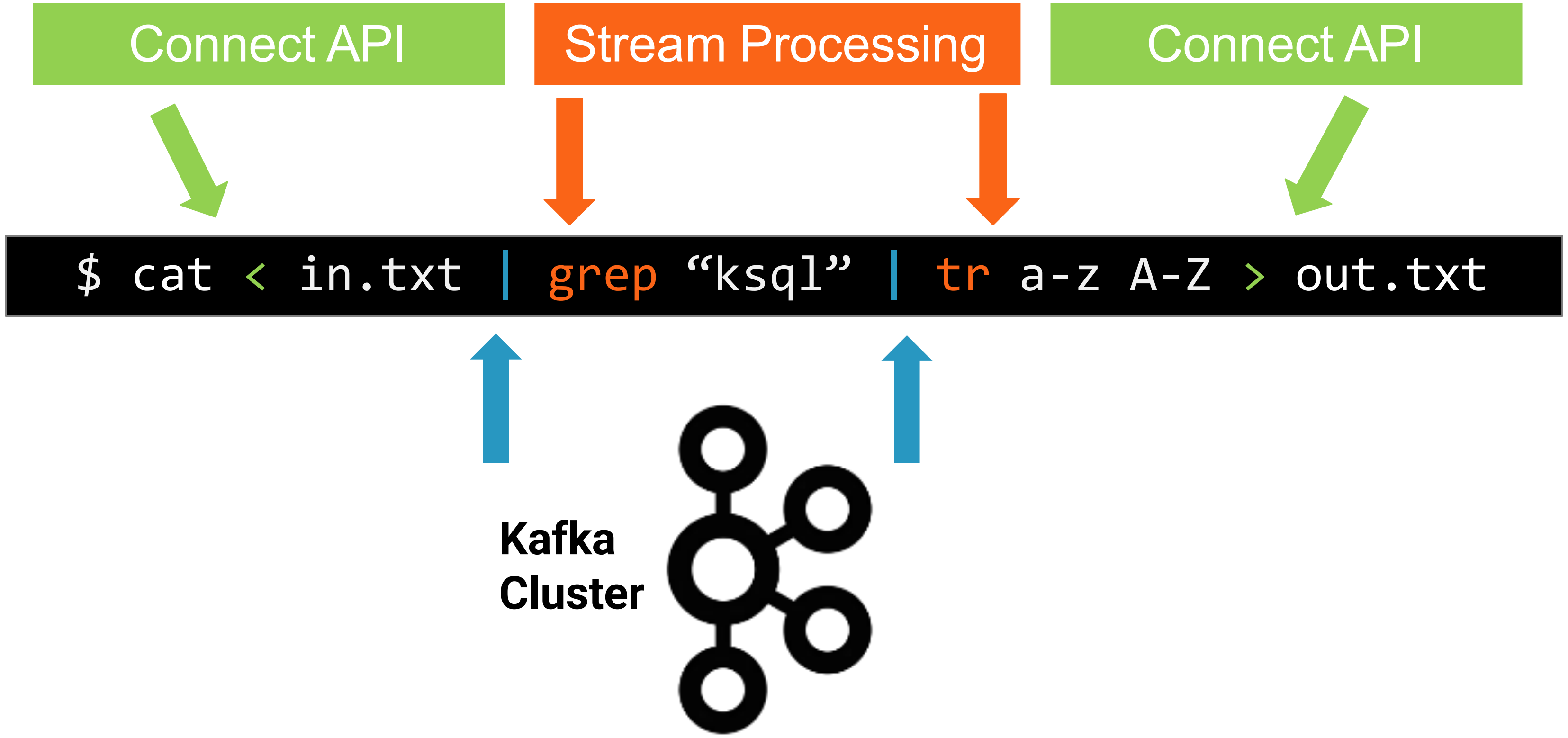
2 CLI

```
POST /query
```

3 REST

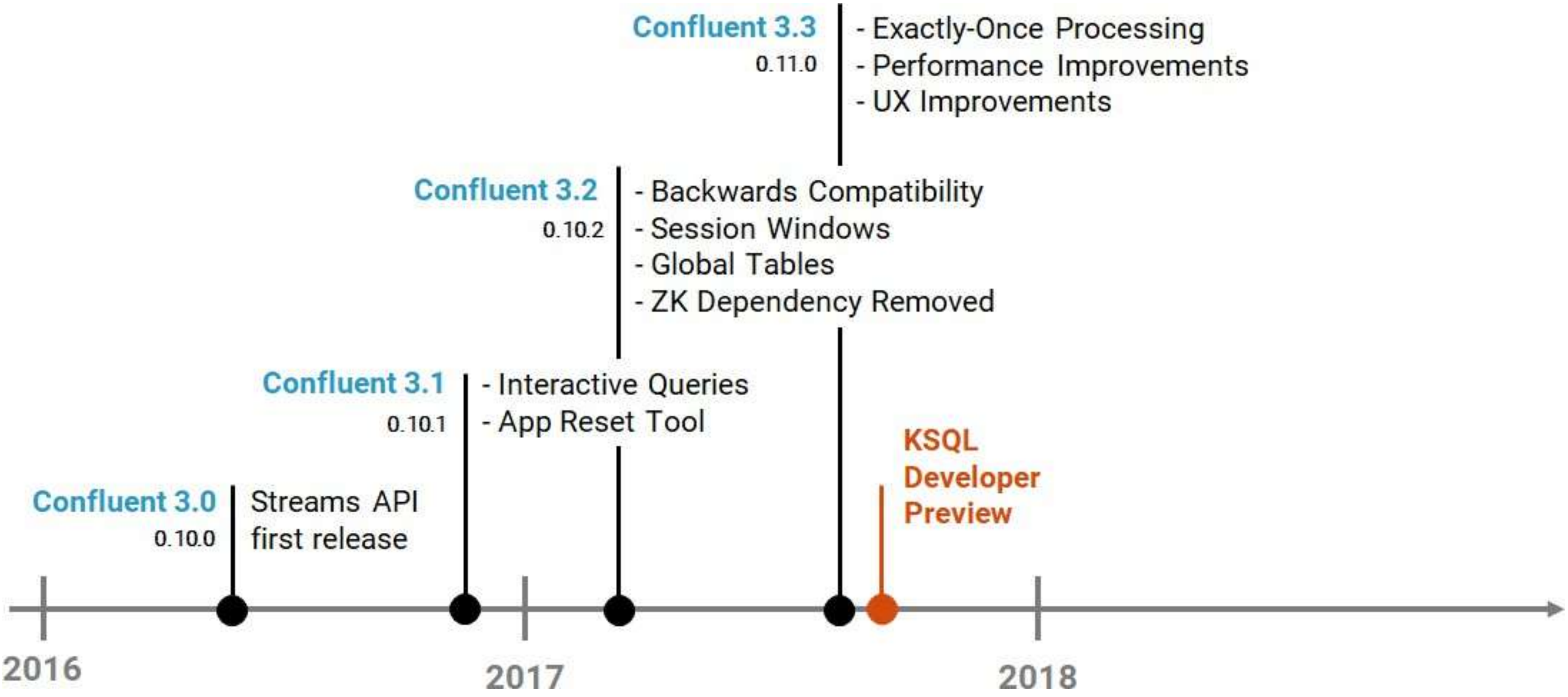
4 Headless

Stream Processing by Analogy



```
$ cat < in.txt | grep “ksql” | tr a-z A-Z > out.txt
```

Kafka Stream Processing Evolution



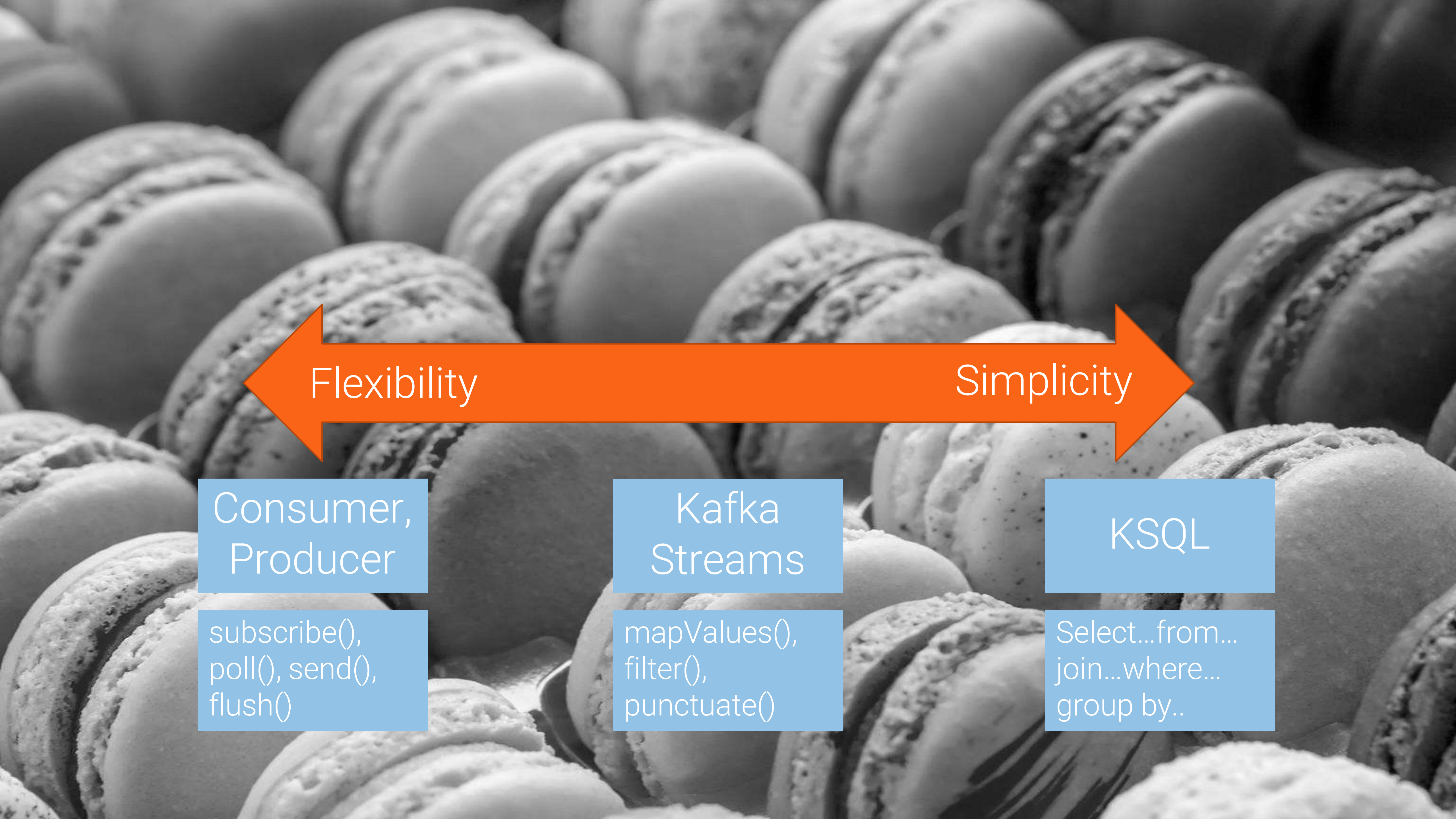
Confluent 3.0
0.10.0
Streams API first release

Confluent 3.1
0.10.1
- Interactive Queries
- App Reset Tool

Confluent 3.2
0.10.2
- Backwards Compatibility
- Session Windows
- Global Tables
- ZK Dependency Removed

Confluent 3.3
0.11.0
- Exactly-Once Processing
- Performance Improvements
- UX Improvements

KSQL Developer Preview



Flexibility

Simplicity

Consumer, Producer

subscribe(), poll(), send(), flush()

Kafka Streams

mapValues(), filter(), punctuate()

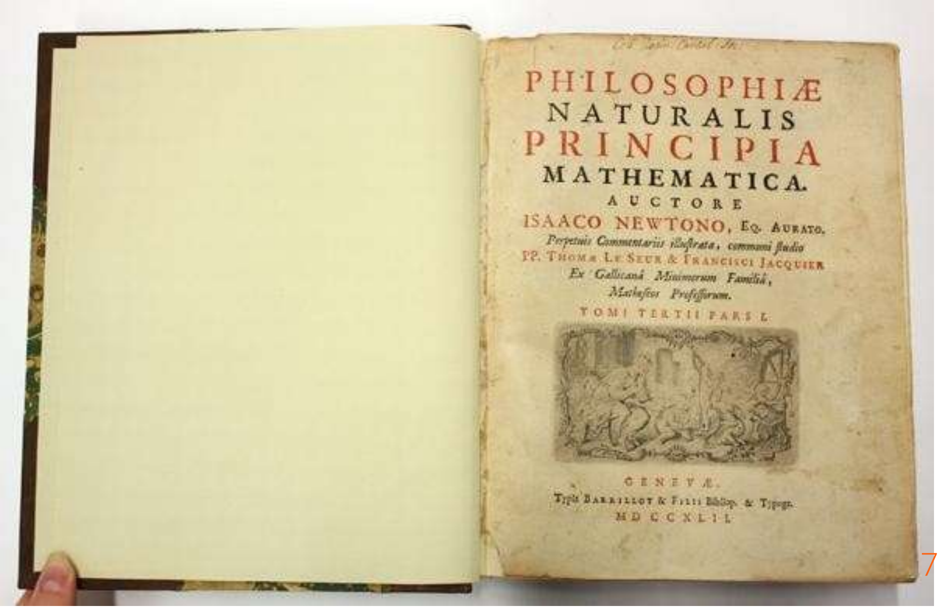
KSQL

Select…from… join…where… group by..

On the Shoulders of (Streaming) Giants

- Native, 100%-compatible Kafka integration
- Secure stream processing using Kafka’s security features
- Elastic and highly scalable
- Fault-tolerant
- Stateful and stateless computations
- Interactive queries

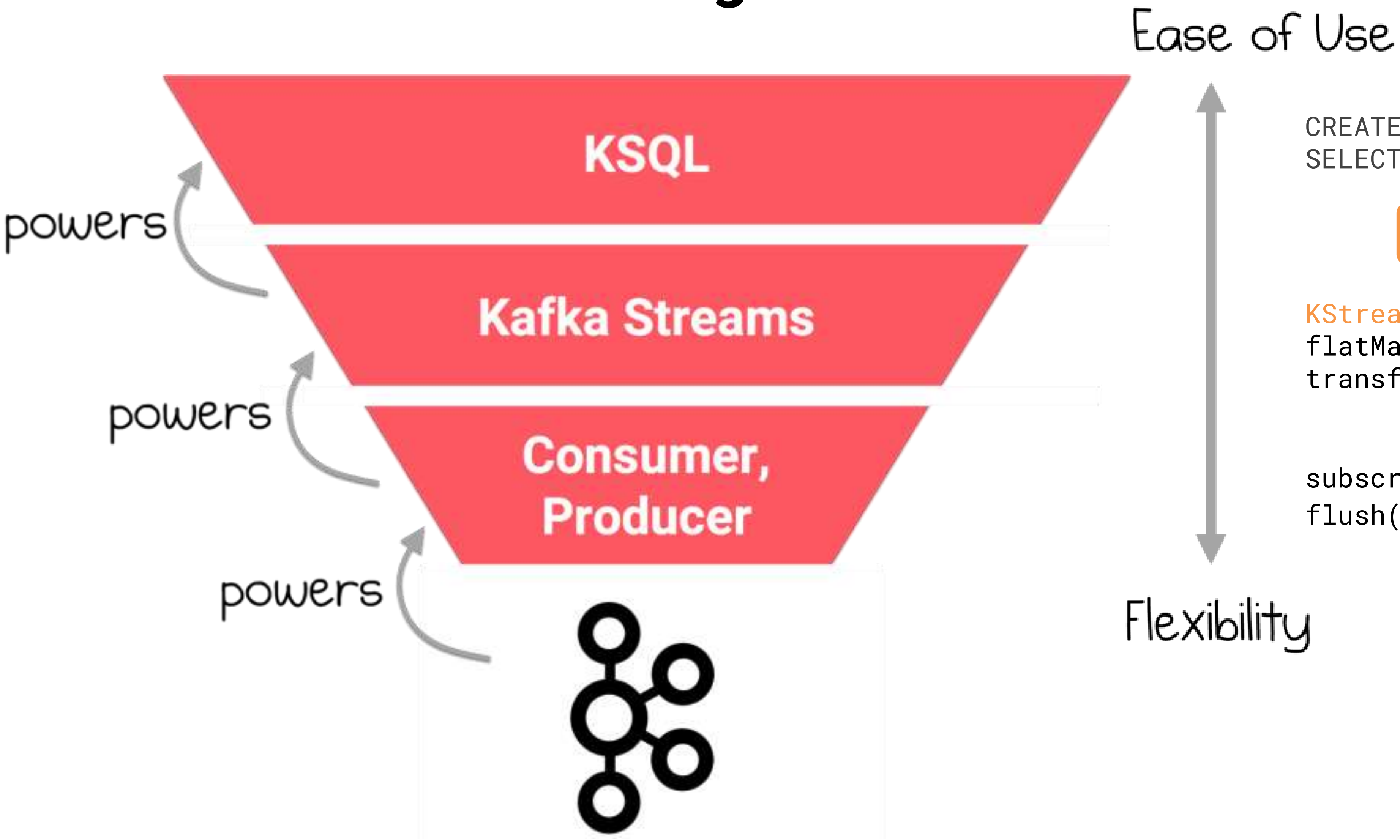
- Time model
- Supports late-arriving and out-of-order data
- Windowing
- Millisecond processing latency, no micro-batching
- At-least-once and exactly-once processing guarantees



PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.
AUCTORE
ISAACO NEWTONO, EQ. AURATO.
Perpetuis Commentariis illuſtrata, communi ſtudio
PP. THOMÆ LE SEUR & FRANCISCI JACQUIER
Ex Gallicanâ Minimorum Familiâ,
Matheſeos Profeſſorum.
TOMI TERTII PARS I.

GENEVÆ.
Typis BARRILLOT & FILII Bibliop. & Typogr.
MDCCXLII.

Shoulders of Streaming Giants



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

KSQL UDFs

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

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

Example Use Cases

(focus on KSQL)

What is it for ?

- Streaming ETL
  - Kafka is popular for data pipelines.
  - KSQL enables easy transformations of data within the pipe

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

What is it for ?

- Anomaly Detection
  - Identifying patterns or anomalies in real-time data, surfaced in milliseconds

```
CREATE TABLE possible_fraud AS
SELECT card_number, count(*)
FROM authorization_attempts
WINDOW TUMBLING (SIZE 5 SECONDS)
GROUP BY card_number
HAVING count(*) > 3;
```

What is it for ?

- Real Time Monitoring
  - Log data monitoring, tracking and alerting
  - Sensor / IoT data

```
CREATE TABLE error_counts AS
SELECT error_code, count(*)
FROM monitoring_stream
WINDOW TUMBLING (SIZE 1 MINUTE)
WHERE type = 'ERROR'
GROUP BY error_code;
```

What is it for ?

- Simple Derivations of Existing Topics
  - One-liner to re-partition and/or re-key a topic for new uses

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

KSQL for Data Exploration

An easy way to inspect your data in Kafka

```
SHOW TOPICS;
```

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

```
SELECT page, user_id, status, bytes
 FROM clickstream
 WHERE user_agent LIKE 'Mozilla/5.0%';
```

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. Change number of partitions
2. Convert data to JSON
3. Repartition the data

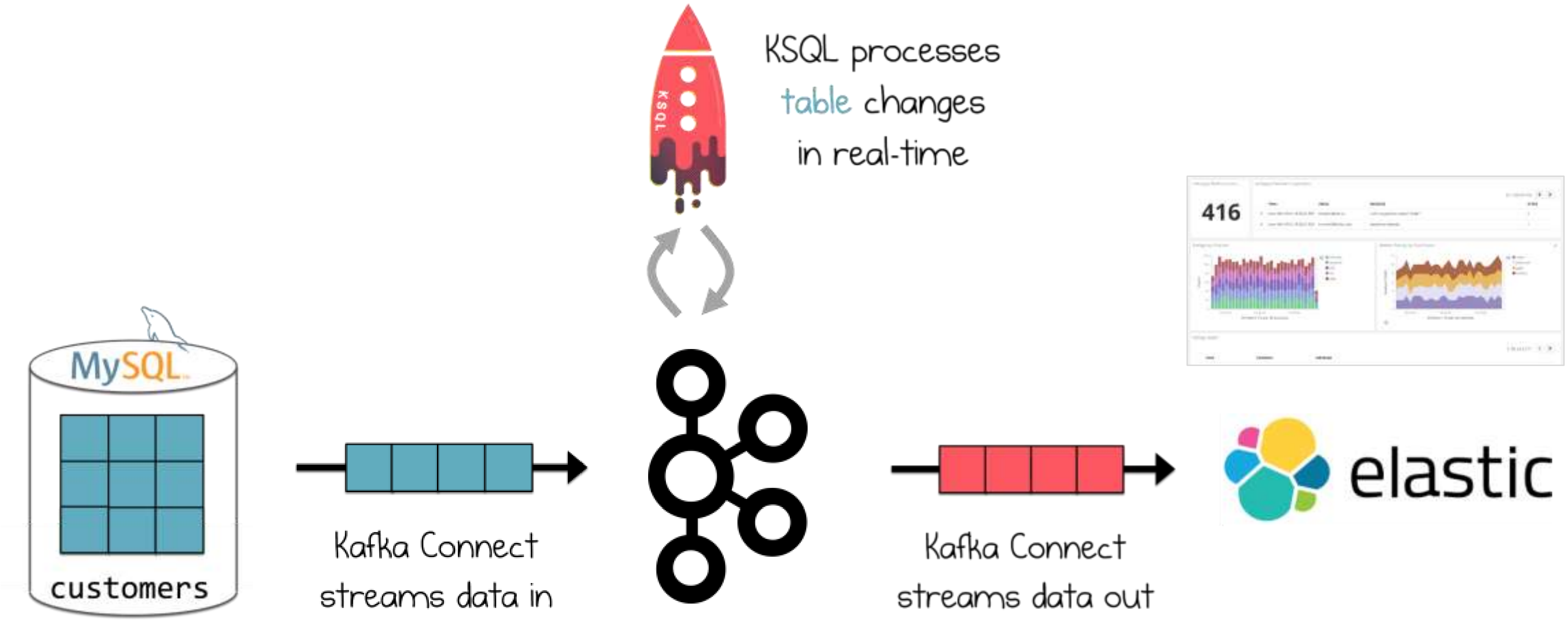
KSQL for Real-Time, Streaming ETL

Filter, cleanse, process data while it is in motion

```
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';
```

1 Pick only VIP users

Example: CDC from DB via Kafka to Elastic



KSQL processes table changes in real-time

Kafka Connect streams data in

Kafka Connect streams data out

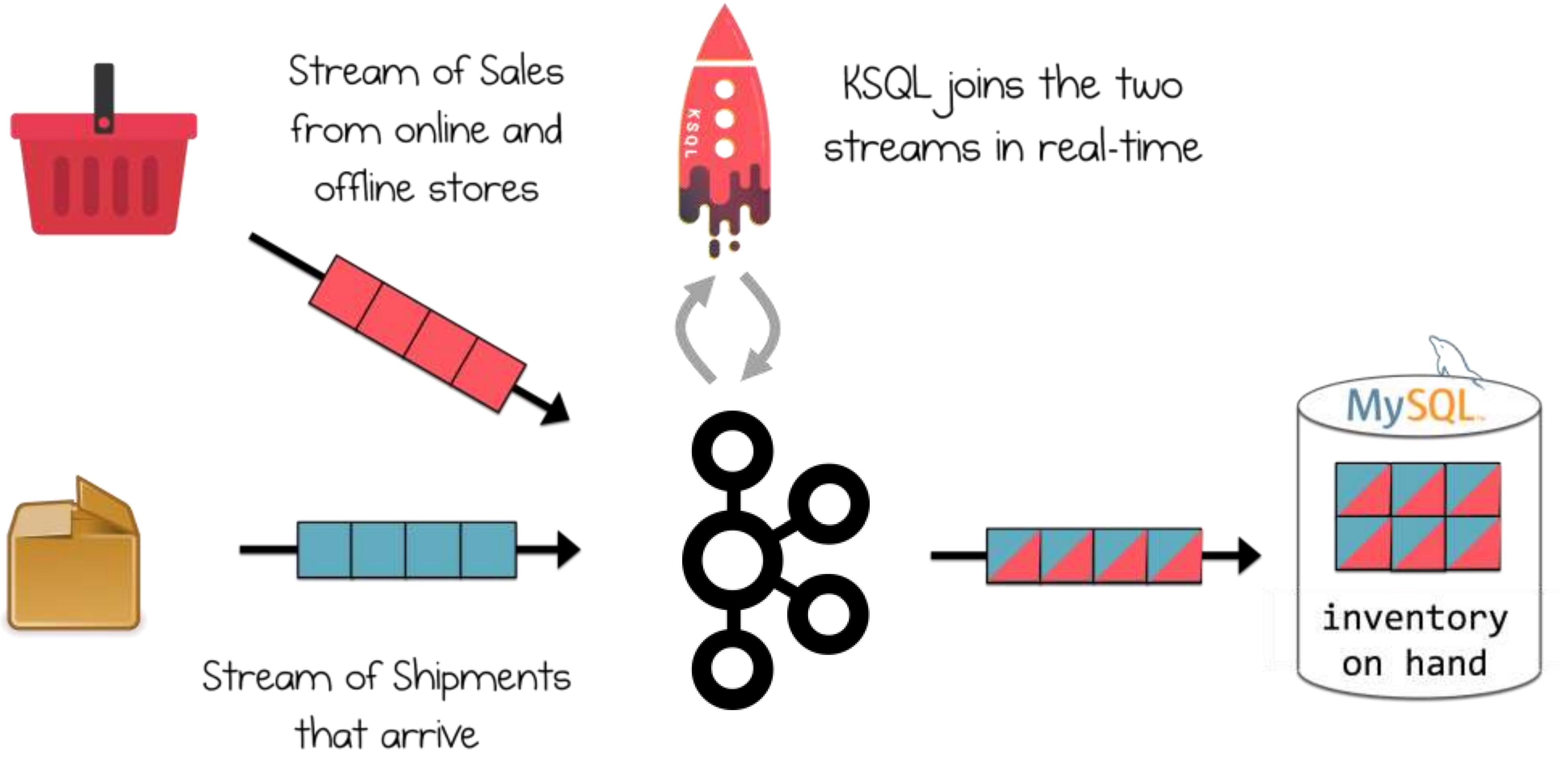
KSQL for Real-time Data Enrichment

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

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

1 Stream-Table Join

Example: Retail



Stream of Sales from online and offline stores

KSQL joins the two streams in real-time

Stream of Shipments that arrive

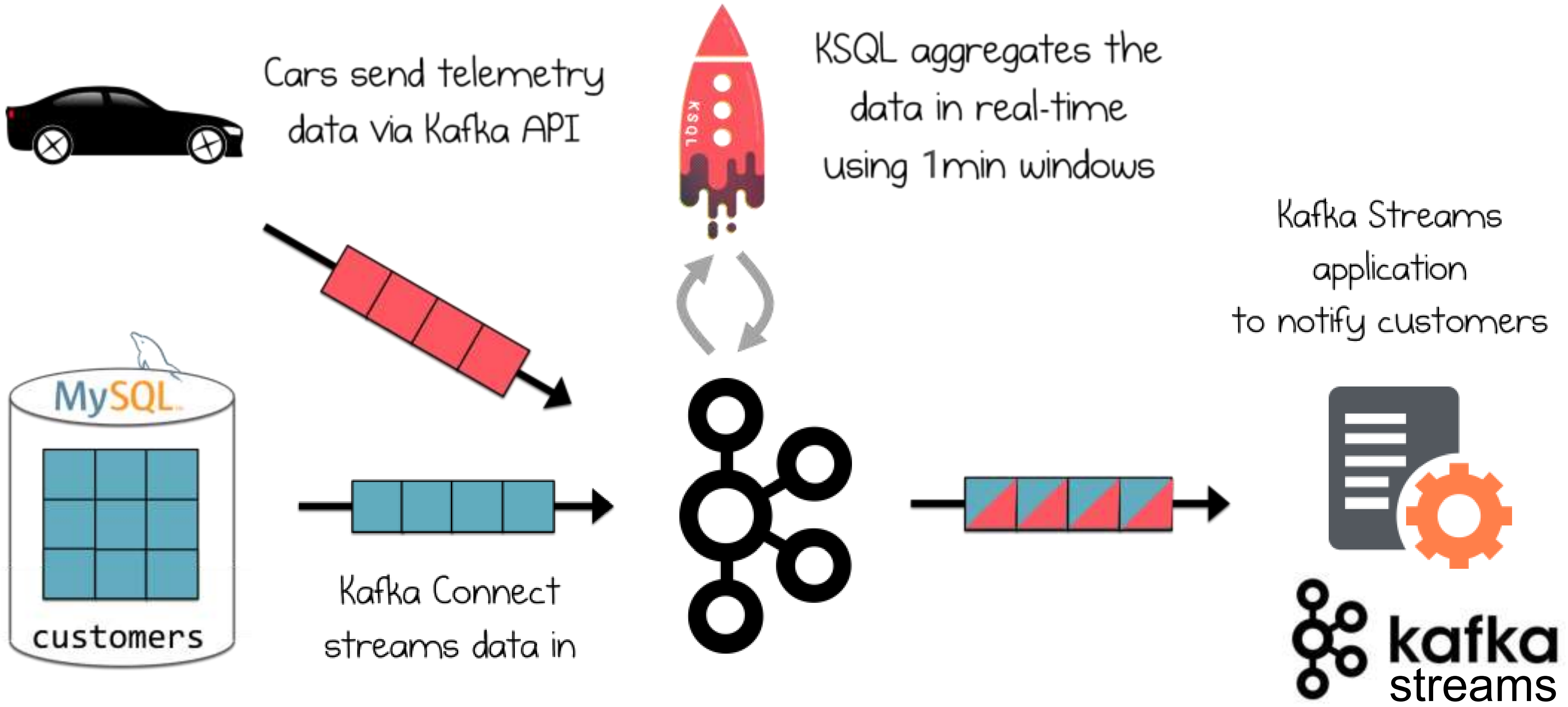
KSQL for Real-Time Monitoring

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

```
CREATE TABLE failing_vehicles AS
   SELECT vehicle, COUNT(*)
   FROM vehicle_monitoring_stream
   WINDOW TUMBLING (SIZE 1 MINUTE)
   WHERE event_type = 'ERROR'
   GROUP BY vehicle
   HAVING COUNT(*) >= 5;
```

1 Now we know to alert, and whom

Example: IoT, Automotive, Connected Cars



Cars send telemetry data via Kafka API

KSQL aggregates the data in real-time using 1min windows

Kafka Streams application to notify customers

Kafka Connect streams data in

KSQL for Anomaly Detection

Aggregate data to identify patterns and 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 30-sec windows

Where is KSQL not such a great fit ?

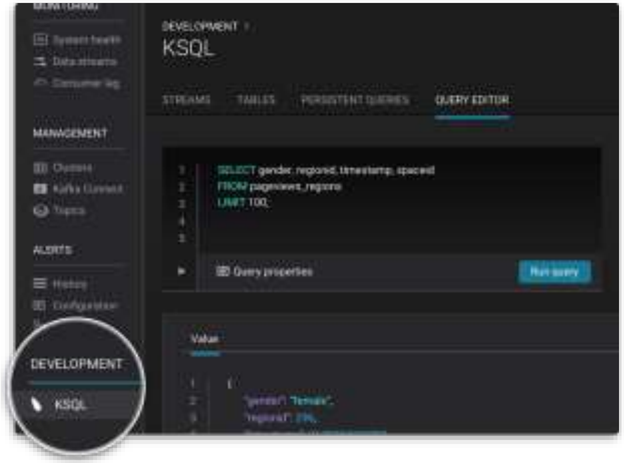
- Ad-hoc query
    - Limited span of time usually retained in Kafka
    - No indexes for random point lookups
- BI reports (Tableau etc.)
    - No indexes
    - No JDBC (most BI tools are not good with continuous results!)

Workflow Comparison

Typical developer interaction

KSQL

kafka streams



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

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

write KSQL queries

view results in real-time

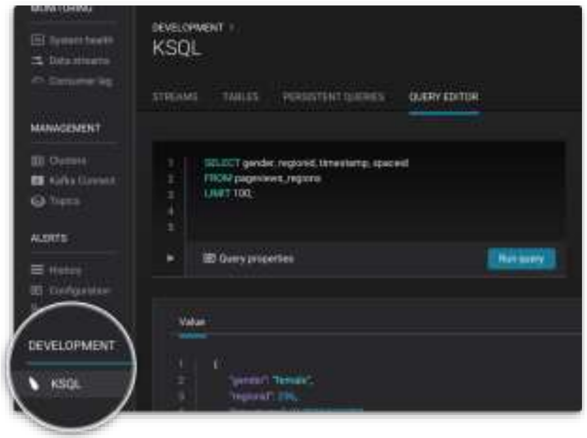
write code in Java or Scala

recompile, then run/test your app

KSQL: typical workflow from development to production

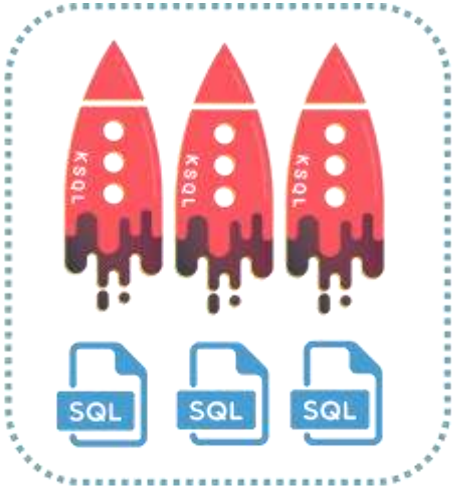
Interactive KSQL
for development

Headless KSQL
in production



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

```
{
  "gender": "female",
  "regionid": [illegible]
```



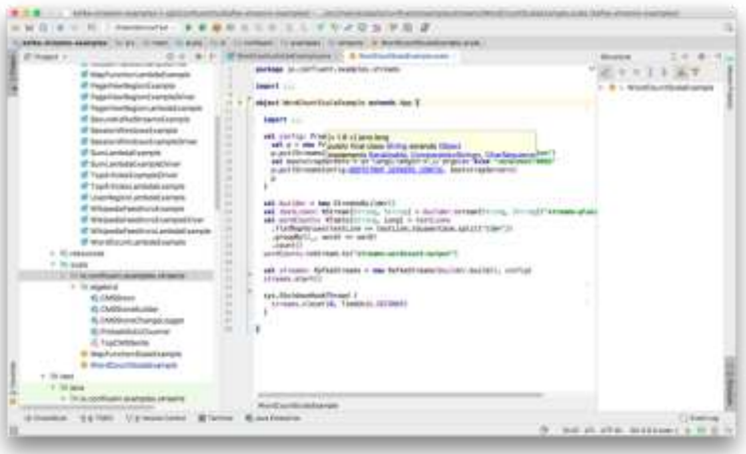
develop your application
and its queries

deploy & run application

Kafka Streams: typical workflow from development to production

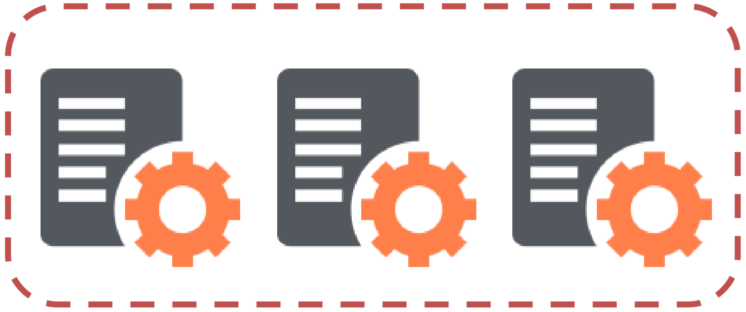
Local development and testing with Java/Scala IDE

Production



develop your application

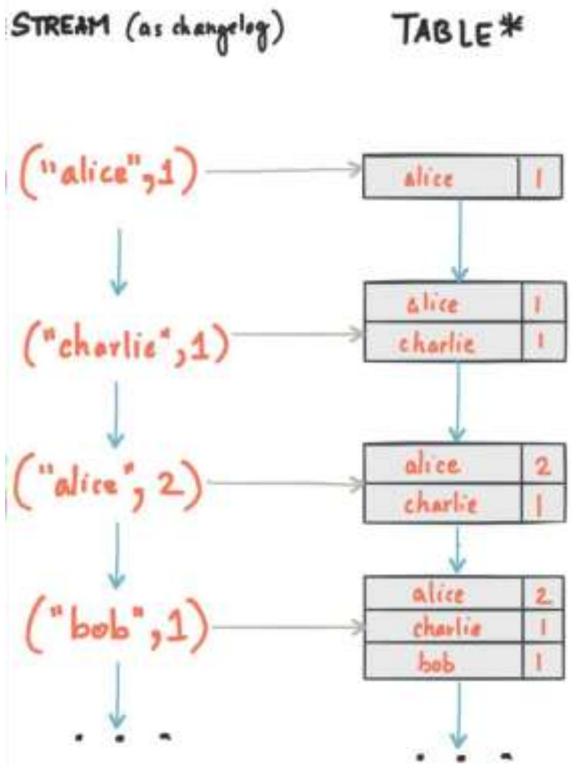
build & package the Java/Scala application



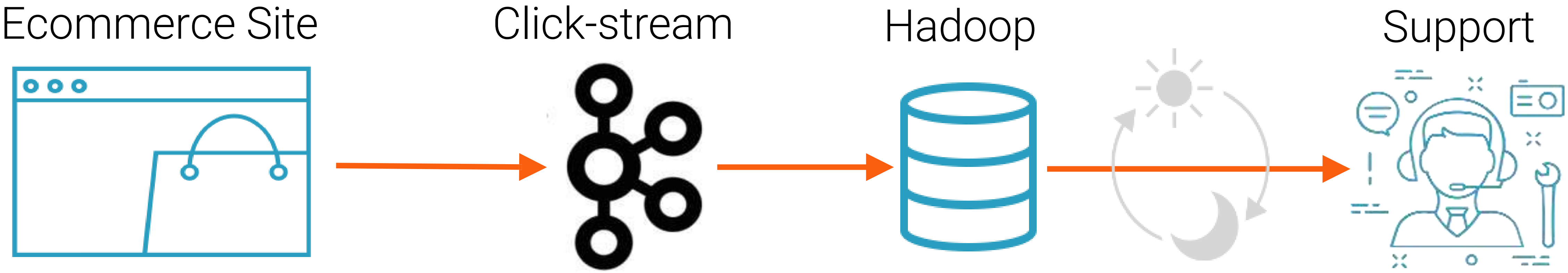
deploy & run application

Streams & Tables

- STREAM and TABLE as first-class citizens
- Interpretations of topic content
- STREAM - data in motion
- TABLE - collected state of a stream
  - One record per key (per window)
  - Current values (compacted topic) ← Not yet in KSQL
  - Changelog
- STREAM – TABLE Joins



Running Example



Ecommerce Site

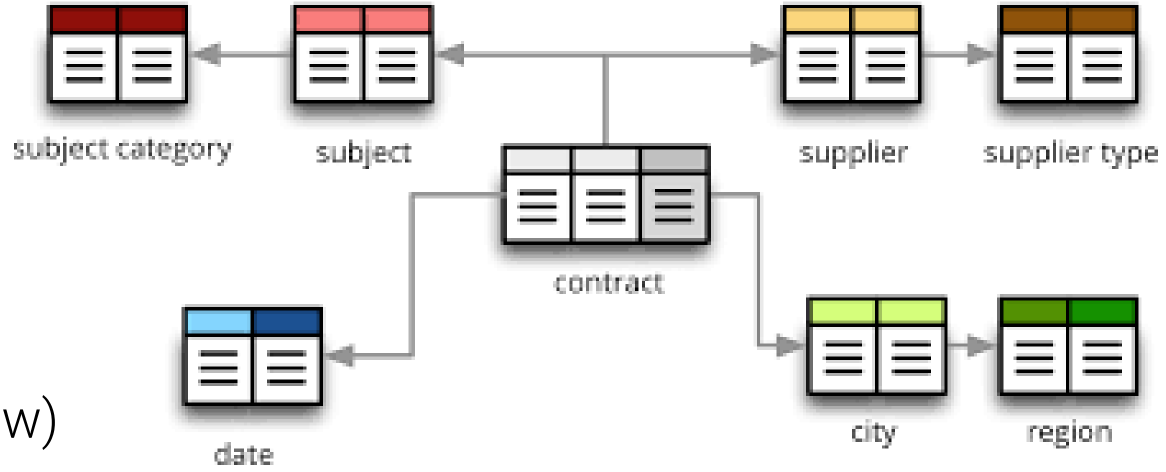
Click-stream

Hadoop

Support

Schema and Format

- A Kafka broker knows how to move <sets of bytes>
  - Technically a message is (ts, byte[], byte[])
- SQL-like queries require a richer structure
- Start with message (value) format
  - JSON
  - DELIMITED (comma-separated in this preview)
  - AVRO – requires that you supply a .avsc schema-file
- Pseudo-columns are automatically generated
  - ROWKEY, ROWTIME



Schema & Datatypes

- varchar / string
- boolean / bool
- integer / int
- bigint / long
- double
- array(*of_type*)
  - *of-type* must be primitive (no nested Array or Map yet)
- map(*key_type, value_type*)
  - *key-type* must be string, *value-type* must be primitive

Define a Stream

```
CREATE STREAM ratings (
   rating_id long,
   user_id int,
   stars int,
   route_id int,
   rating_time long,
   channel varchar,
   message varchar)
WITH (
value_format=‘JSON',
kafka_topic=‘ratings');
```

SELECTing from the Stream

Let’s test our new stream definition by finding all the low-scoring ratings from our iPhone app

```
SELECT *
FROM ratings
WHERE stars <= 2
AND lcase(channel) LIKE ‘%ios%'
AND user_id > 0
LIMIT 10;
```

SELECTing from the Stream

And set this to run as a continuous transformation, with results being saved into a new topic

```
CREATE STREAM poor_ratings AS
SELECT *
FROM ratings
WHERE stars <= 2
AND lcase(channel) LIKE ‘%ios%’;
```

Bring in reference tables

Define reference tables

```
CREATE TABLE users (
   uid int,
   name varchar,
   elite varchar)
WITH (
   key=‘uid’,
   value_format=‘JSON’,
   kafka_topic=‘mysql-users');
```

Joins for Enrichment

Enrich the ‘poor_ratings’ stream with data about each user, and derive a stream of low quality ratings posted only by our Platinum Elite users

```
CREATE STREAM vip_poor_ratings AS
SELECT uid, name, elite,
  stars, route_id, rating_time, message
FROM poor_ratings r
LEFT JOIN users u ON r.user_id = u.uid
WHERE u.elite = 'P';
```

Aggregates and Windowing

- COUNT, SUM, MIN, MAX
- Windowing - Not strictly ANSI SQL ☺
- Three window types supported:
  - TUMBLING
  - HOPPING (aka ‘sliding’)
  - SESSION

```
SELECT uid, name, count(*) as rating_count
FROM vip_poor_ratings
WINDOW TUMBLING(size 2 minutes)
GROUP BY uid, name;
```

Continuous Aggregates

Save the results of our aggregation to a TABLE

```
CREATE TABLE sad_vips AS
SELECT uid, name, count(*) as rating_count
FROM vip_poor_ratings
WINDOW TUMBLING(size 1 minute)
GROUP BY uid, name
HAVING count(*) > 2;
```

Session Variables

- Just as in MySQL, ORCL etc. there are settings to control how your CLI behaves
- Set any property the Kafka Streams consumers/producers will understand
- Defaults can be set in the *ksql.properties* file
- To see a list of currently set or default variable values:
    - `ksql> show properties;`
- Useful examples:
    - `num.stream.threads=4`
    - `commit.interval.ms=1000`
    - `cache.max.bytes.buffering=2000000`
- TIP! - Your new best friend for testing and development is:
    - `ksql> set ‘auto.offset.reset’ = ‘earliest’;`

KSQL Components

- CLI
  - Designed to be familiar to users of MySQL, Postgres, etc
- Engine
  - Actually runs the Kafka Streams topologies
- REST Server
  - HTTP interface allows an Engine to receive instructions from the CLI

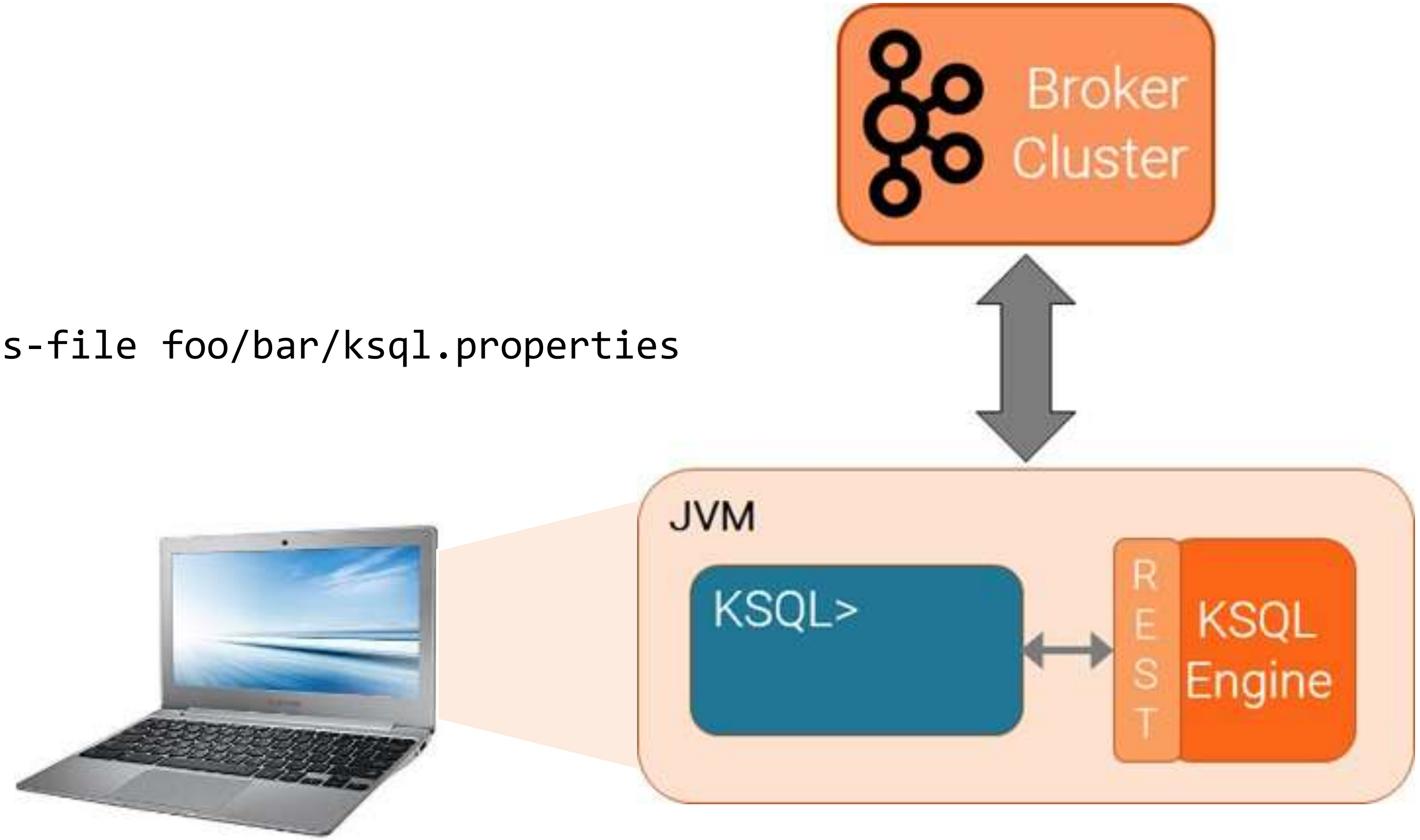
How to run KSQL - #1 Stand-alone aka ‘local mode’

- Starts a CLI, an Engine, and a REST server all in the same JVM
- Ideal for laptop development
  - Start with default settings:

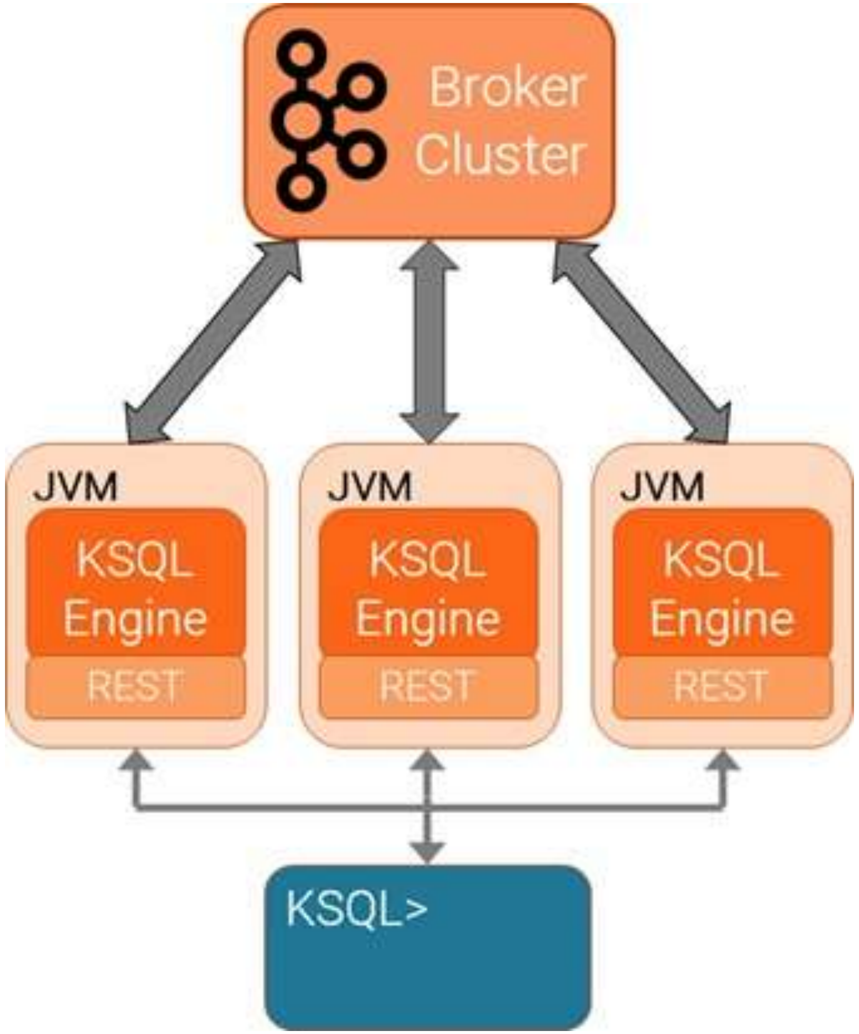
    ```
    > bin/ksql-cli local
    ```

  - Or with customized settings:

    ```
    > bin/ksql-cli local --properties-file foo/bar/ksql.properties
    ```



How to run KSQL - #2 Client-Server



- Start any number of Server nodes
  - `> bin/ksql-server-start`
- Start any number of CLIs and specify ‘remote’ server address
  - `>bin/ksql-cli remote http://myserver:8090`
- All running Engines share the processing load
  - Technically, instances of the same Kafka Streams Applications
  - Scale up/down without restart

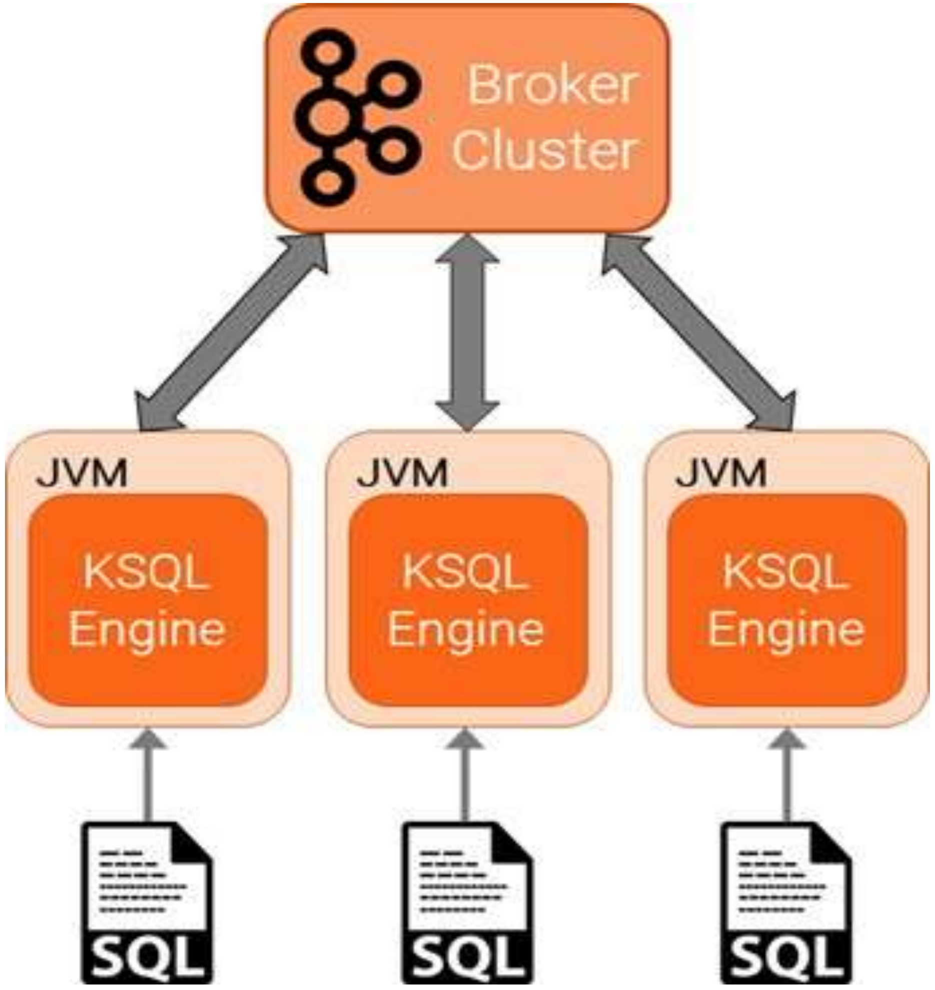
How to run KSQL - #3 as an Application

- Ideal for streaming application deployment
  - Version control your queries and transformations as code
  - Deploy like any other java application
  - Avoid interactive changes to running apps from ‘rogue’ CLI users

- Start any number of Engine instances
  - Pass a file of KSQL statements to execute

```
> bin/ksql-node foo/bar.sql
```

- All running Engines share the processing load
  - Technically, instances of the same Kafka Streams Applications
  - Scale up/down without restart



Resources & Next Steps

Time to get involved !

- Try the Quickstart on Github
- Check out the code
- Play with the examples

The point of 'developer preview' is that we can change things for the better, together

https://github.com/confluentinc/ksql

http://confluent.io/ksql

https://slackpass.io/confluentcommunity #ksql