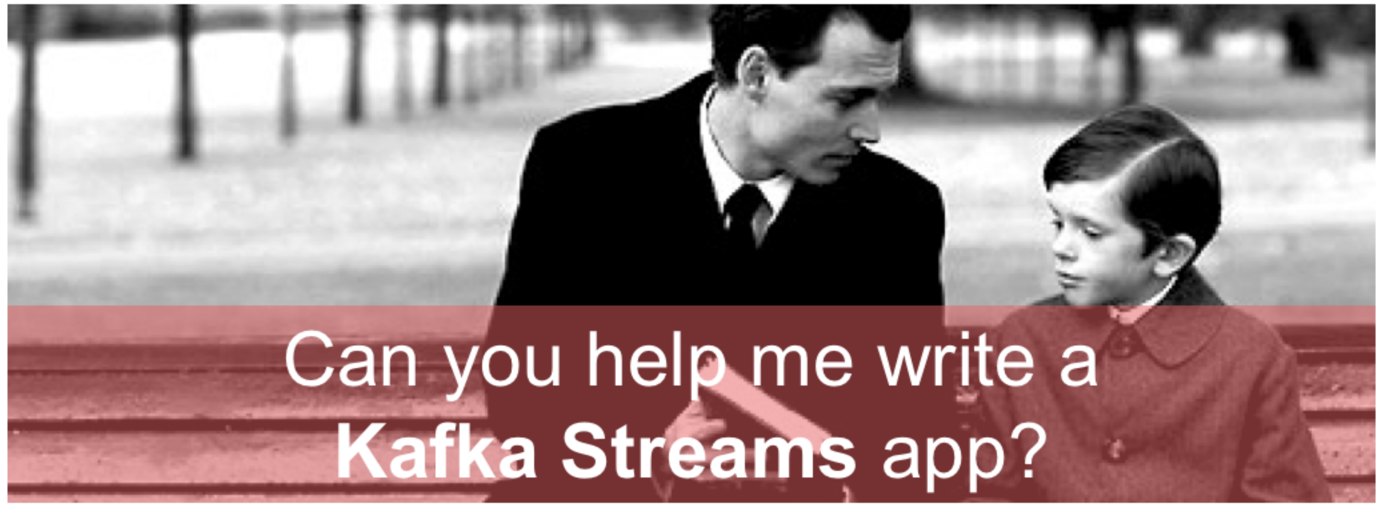
# From ksqlDB with ♥ LOVE ♥

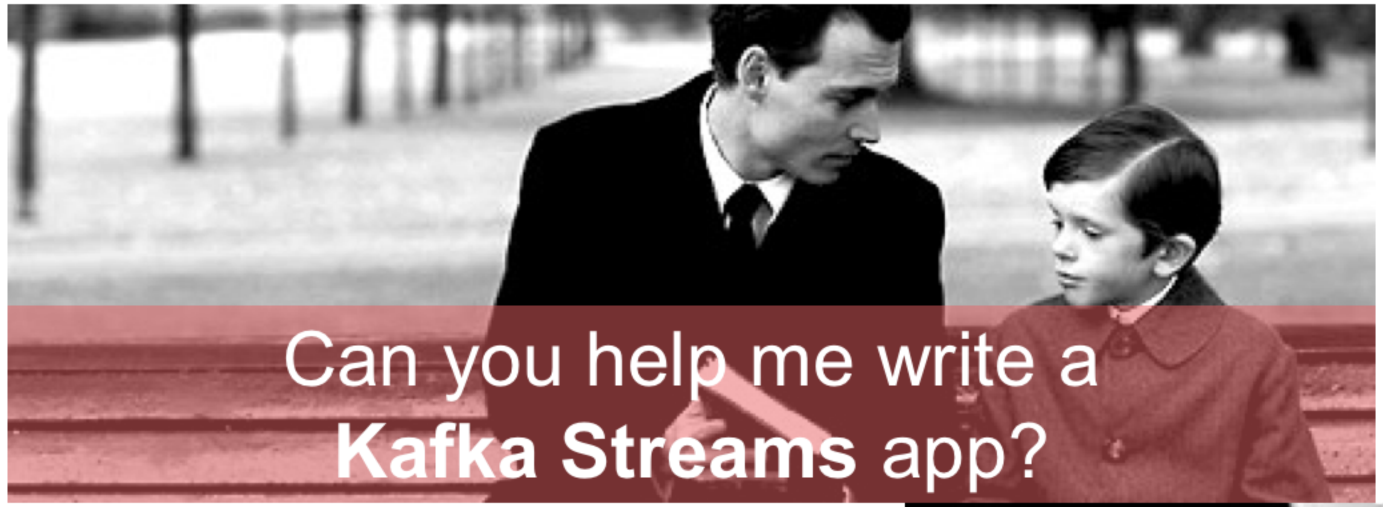
## Detecting 007 with a dash of machine learning

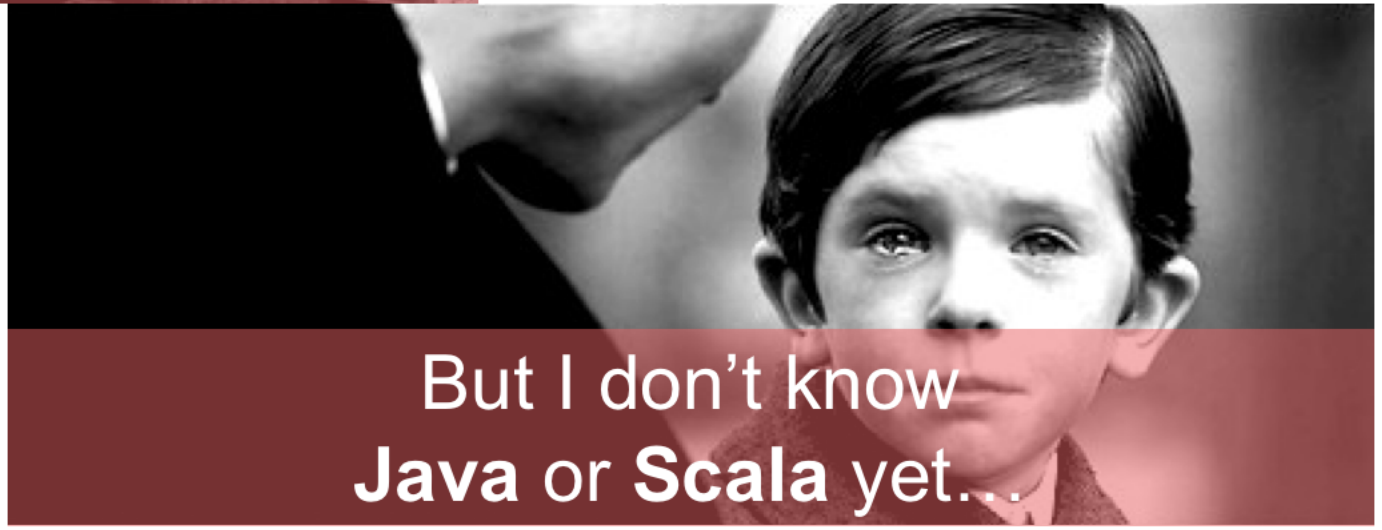
# Hans-Peter Grahsl

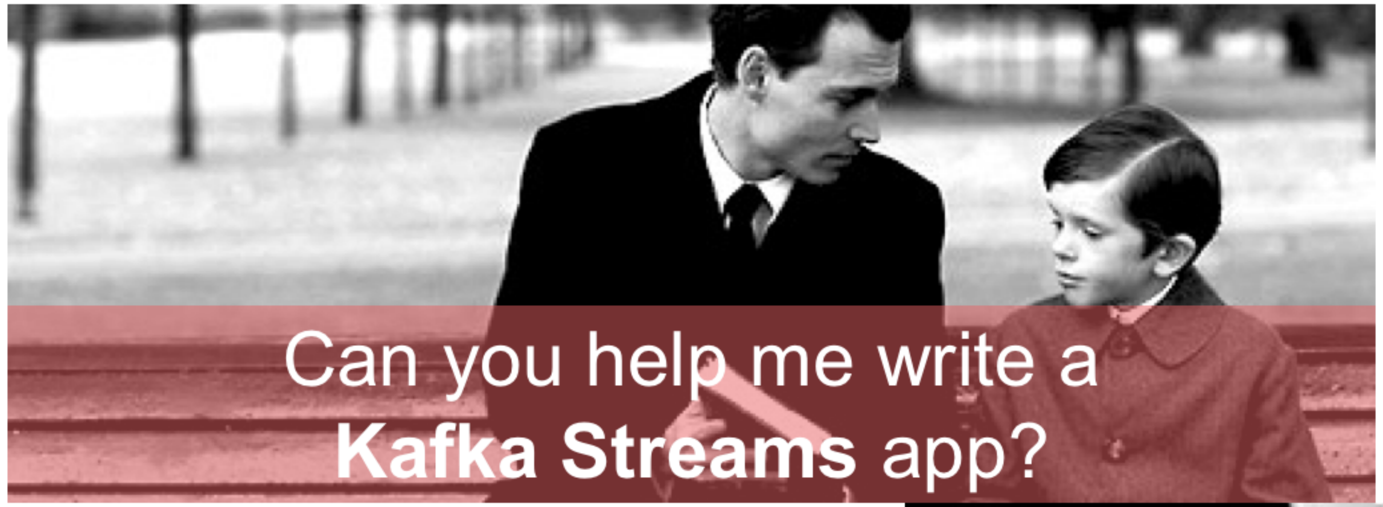
- technical trainer
- independent engineer & consultant
- associate lecturer
- occasional conference speaker

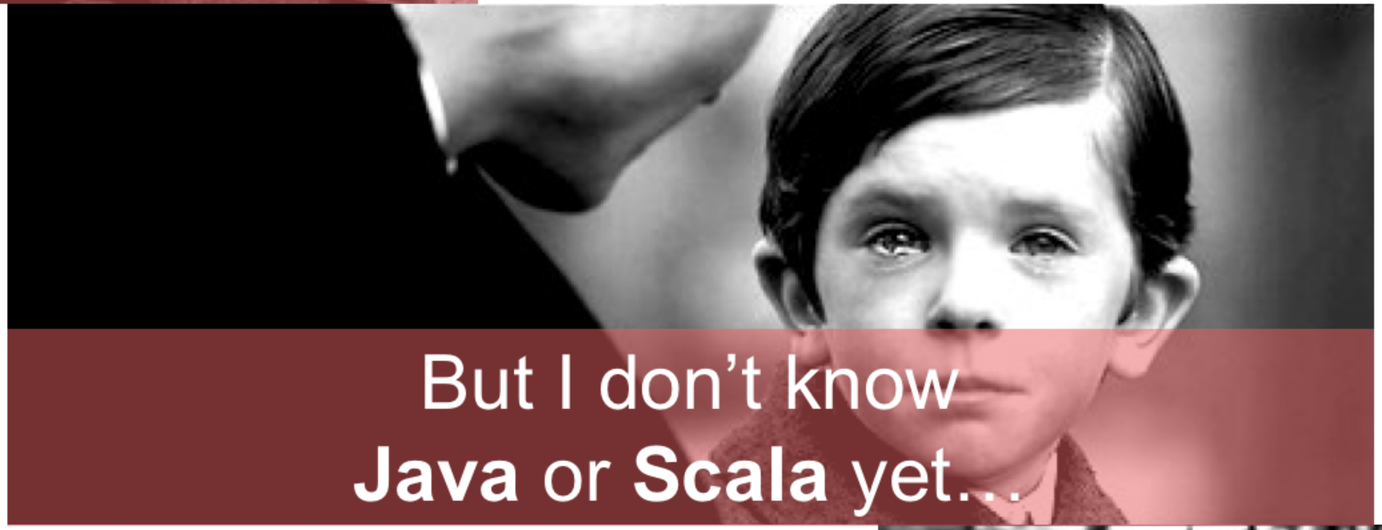

Where did come from?


Can you help me write a
Kafka Streams app?


Can you help me write a
**Kafka Streams** app?


But I don't know
**Java** or **Scala** yet...


Can you help me write a
**Kafka Streams** app?


But I don't know
**Java** or **Scala** yet…

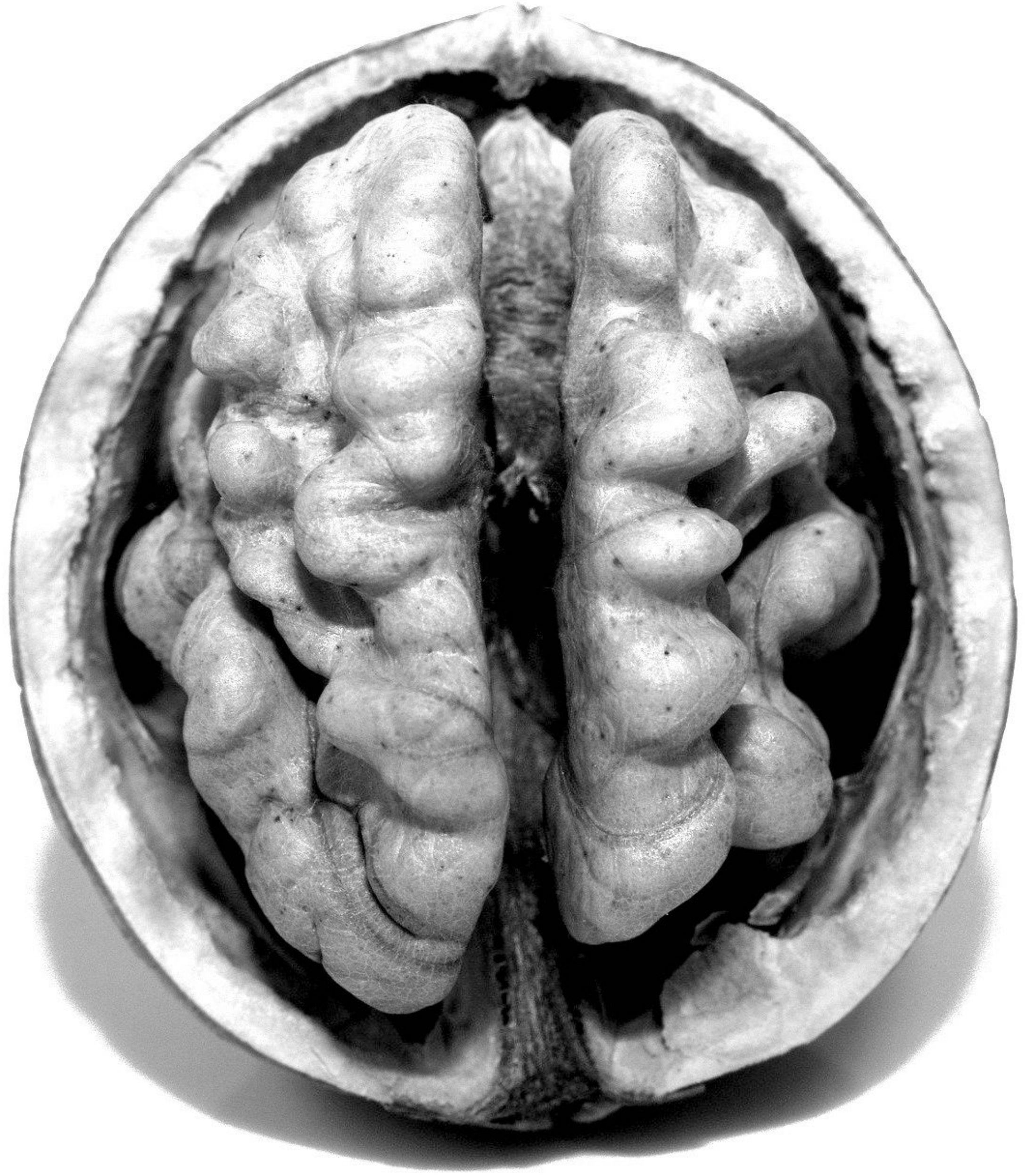

No worries… We will ship
**something declarative** instead!

# Kafka's streaming SQL engine

# declarative
# stream
# processing
# language

# KSQL in a Nutshell

- ANSI **SQL inspired**
  - familiar syntax & semantics
  - concise & expressive
- built on top of **Kafka Streams**
- **NO(!) coding** skills required
- entry barrier? **"none"**



# KSQL in a Nutshell

- usual suspects OOTB:
  - projections, filters
  - joins, aggregations
  - windowing

# Query Examples

```
SELECT u.name,r.movieid,r.stars
FROM ratings r
JOIN users u ON r.userid = u.id
WHERE stars > 3;

SELECT userid,COUNT(*) num_ratings
FROM ratings
GROUP BY userid
HAVING COUNT(*) > 42;
```

```
SELECT r.movieid movieid,
       MIN(stars) worst_rating,
       MAX(stars) best_rating,
       COUNT(*) num_ratings,
       AVG(stars) avg_rating
FROM ratings r
JOIN movies m ON r.movieid = m.movieid
WHERE m.genre = 'Action' OR m.genre = 'Thriller'
GROUP BY r.movieid;
```

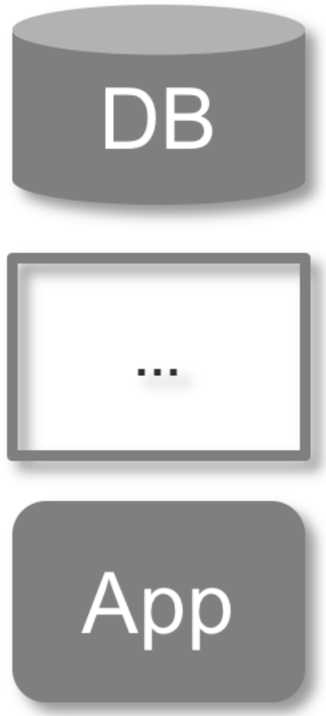
# Criteria 1 - Learning Curve

- SQL == widespread & successful 4GL
- algorithms & data structures handled for us

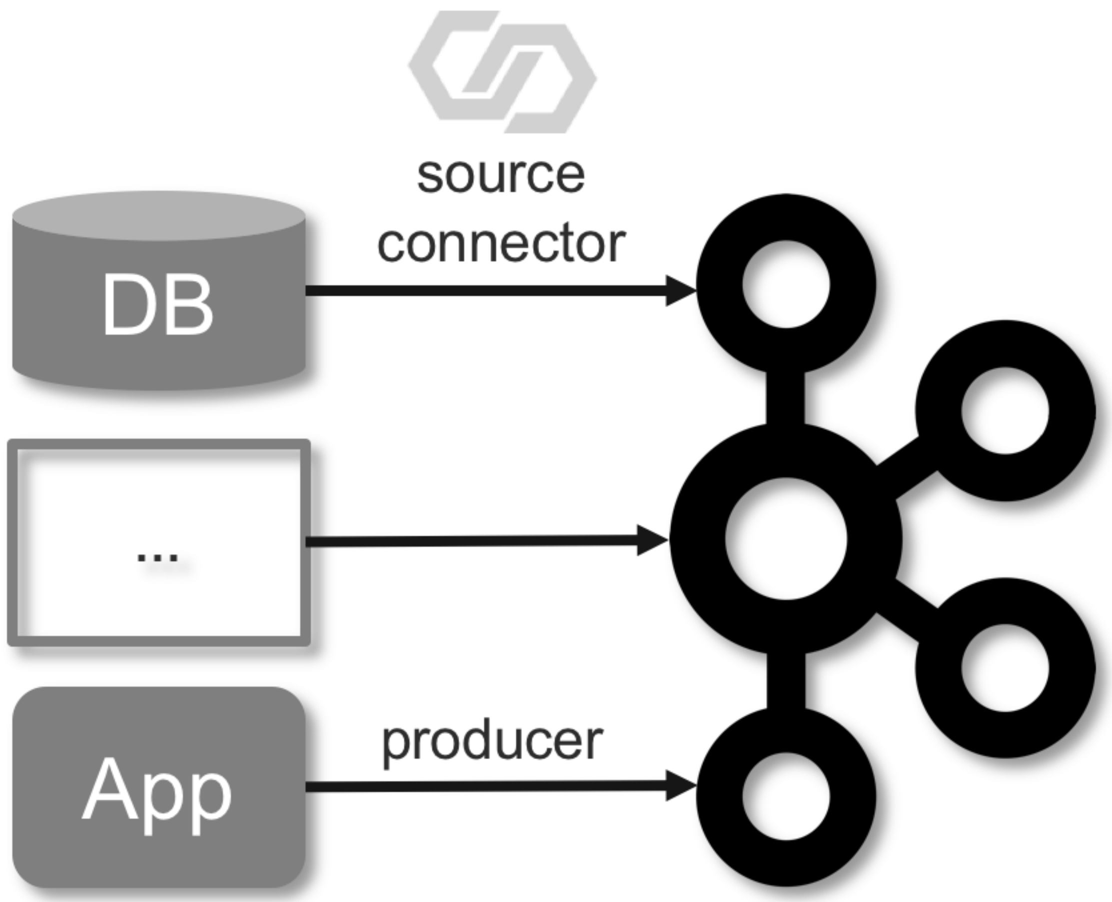
- KSQL
  - very easy to learn
  - quick implementation cycles
  - productivity-wise hard to beat

# "Lingua Franca" effect

# Streaming Architecture



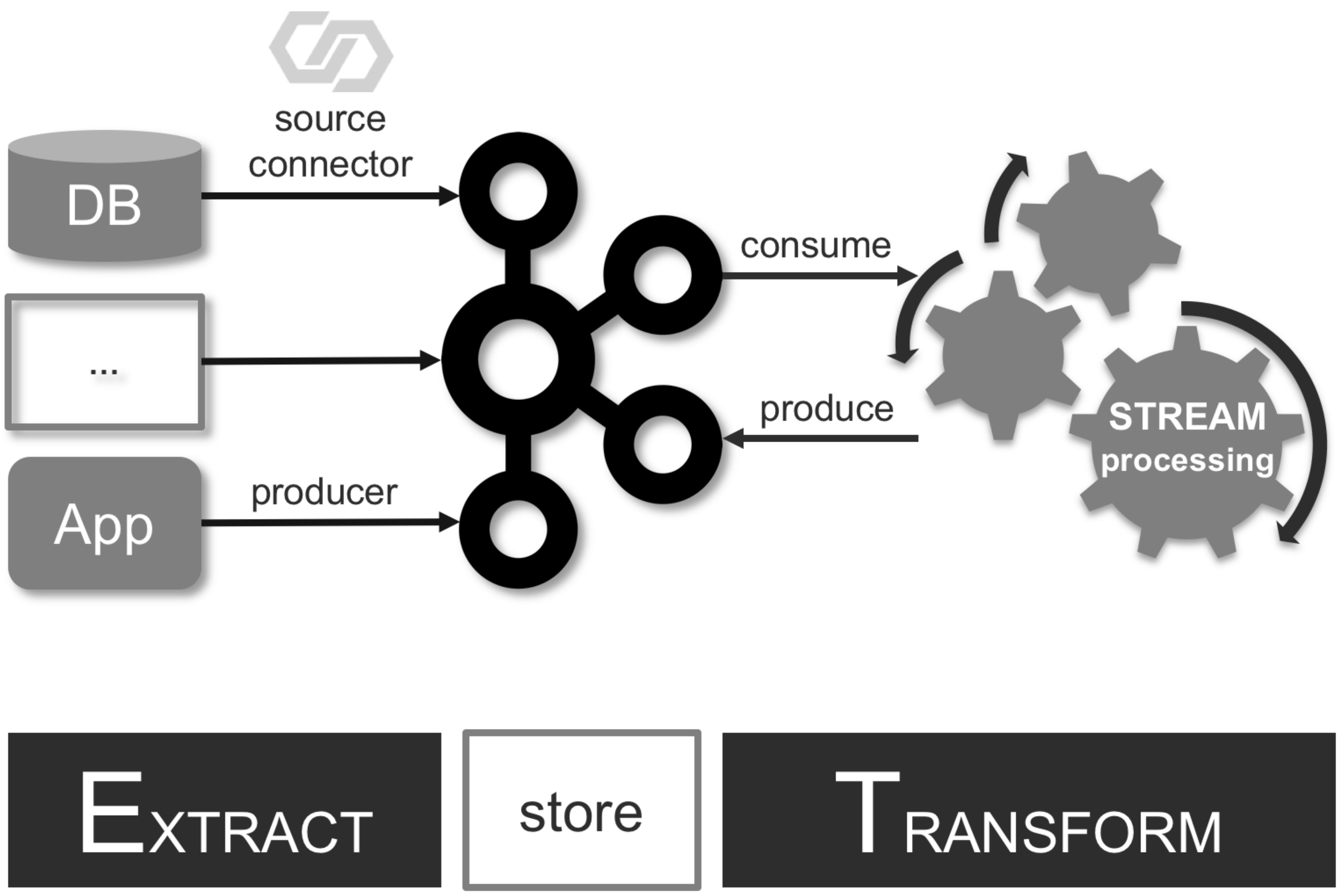
# Streaming Architecture



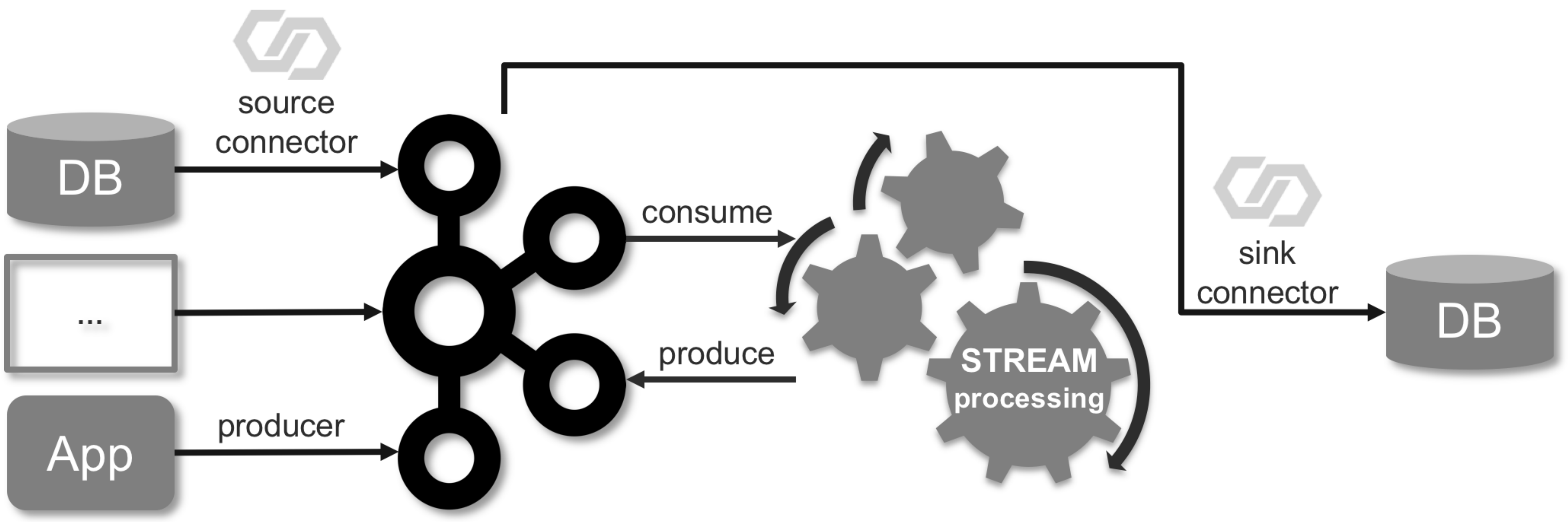

EXTRACT

store

# Streaming Architecture

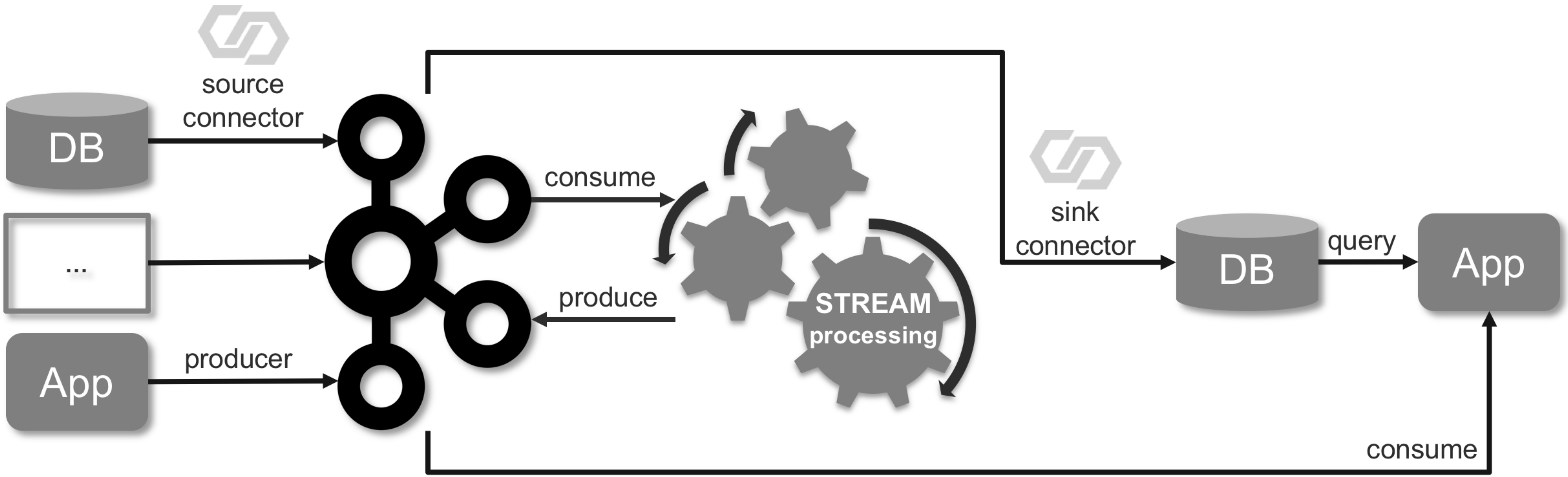


# Streaming Architecture




EXTRACT | store | TRANSFORM | LOAD | store

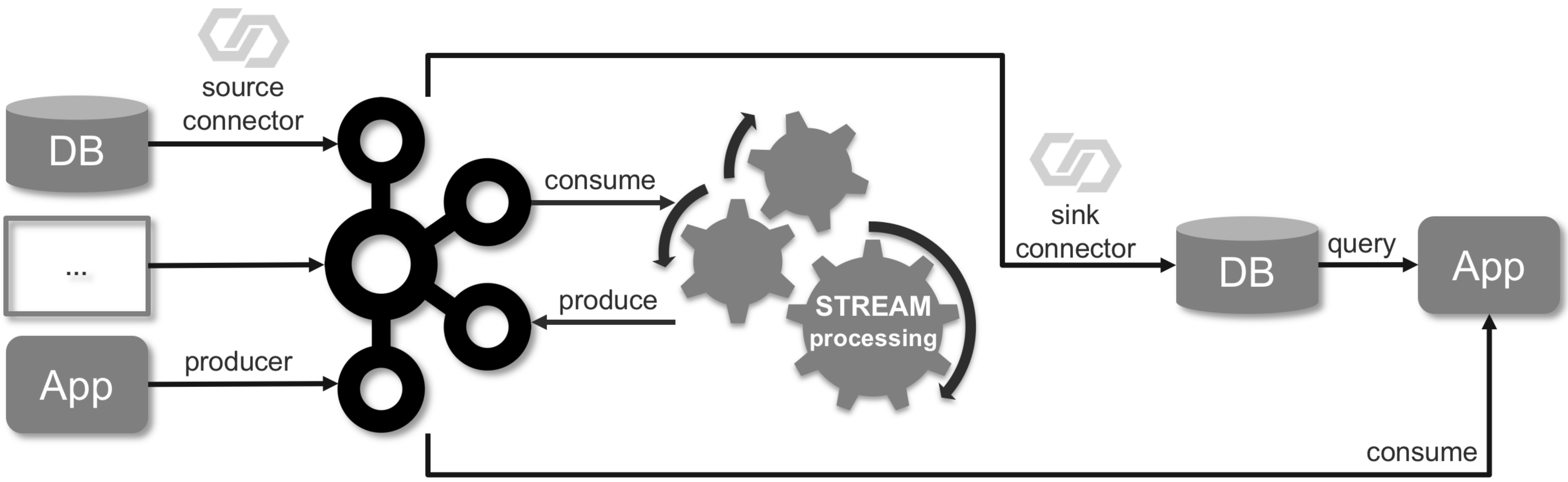
# Streaming Architecture




| | | | | | |
|---|---|---|---|---|---|
| EXTRACT | store | TRANSFORM | LOAD | store | serve |

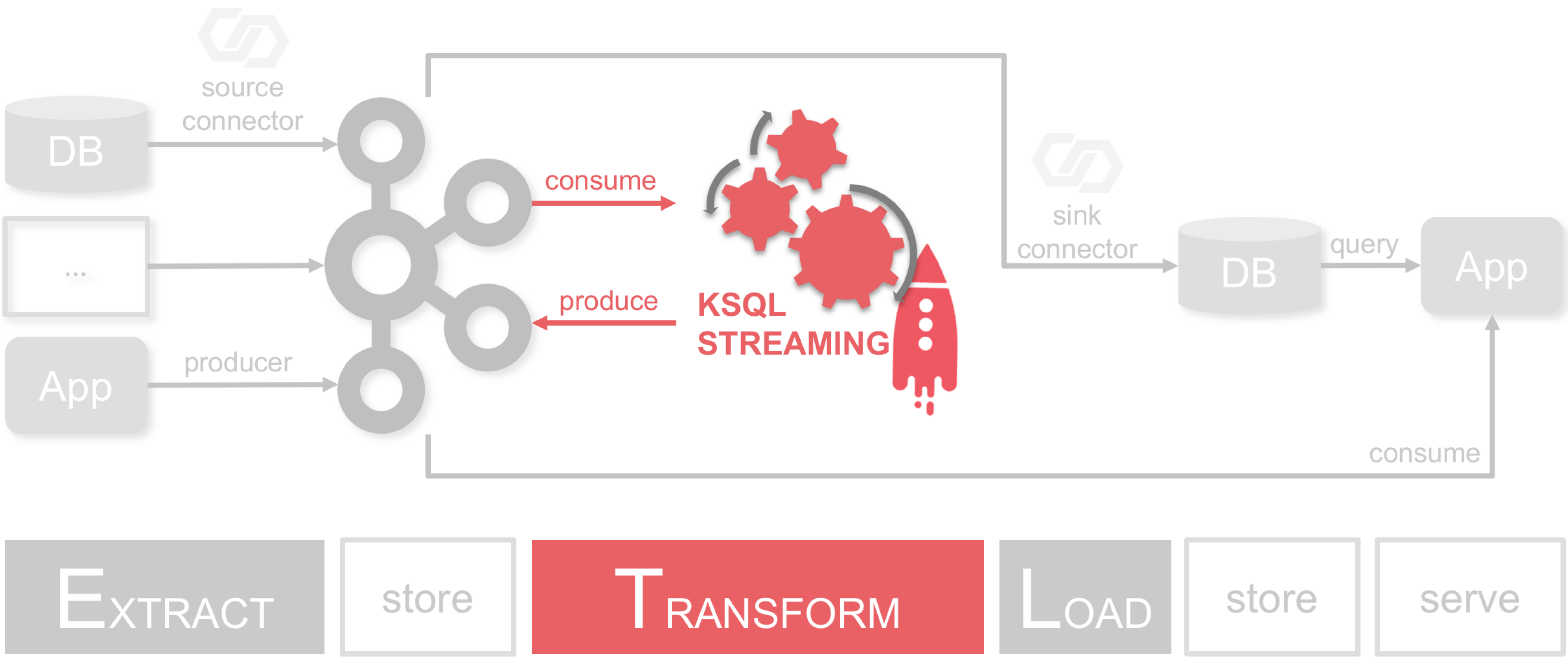
# NOT the ETL of our ancestors

# Complex Streaming Architecture



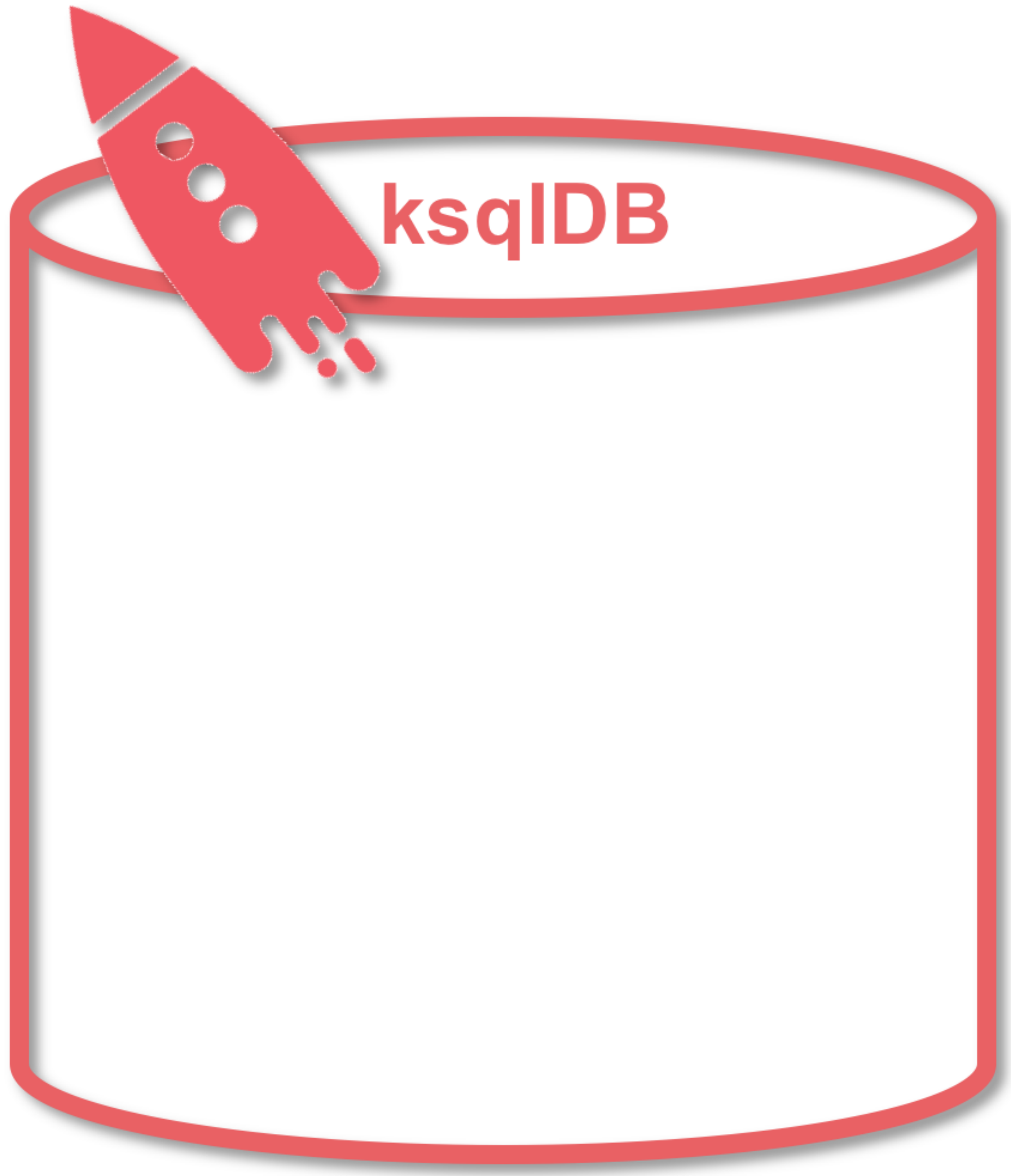

| | | | | | |
|---|---|---|---|---|---|
| EXTRACT | store | TRANSFORM | LOAD | store | serve |

# KSQL → originally the [T] only

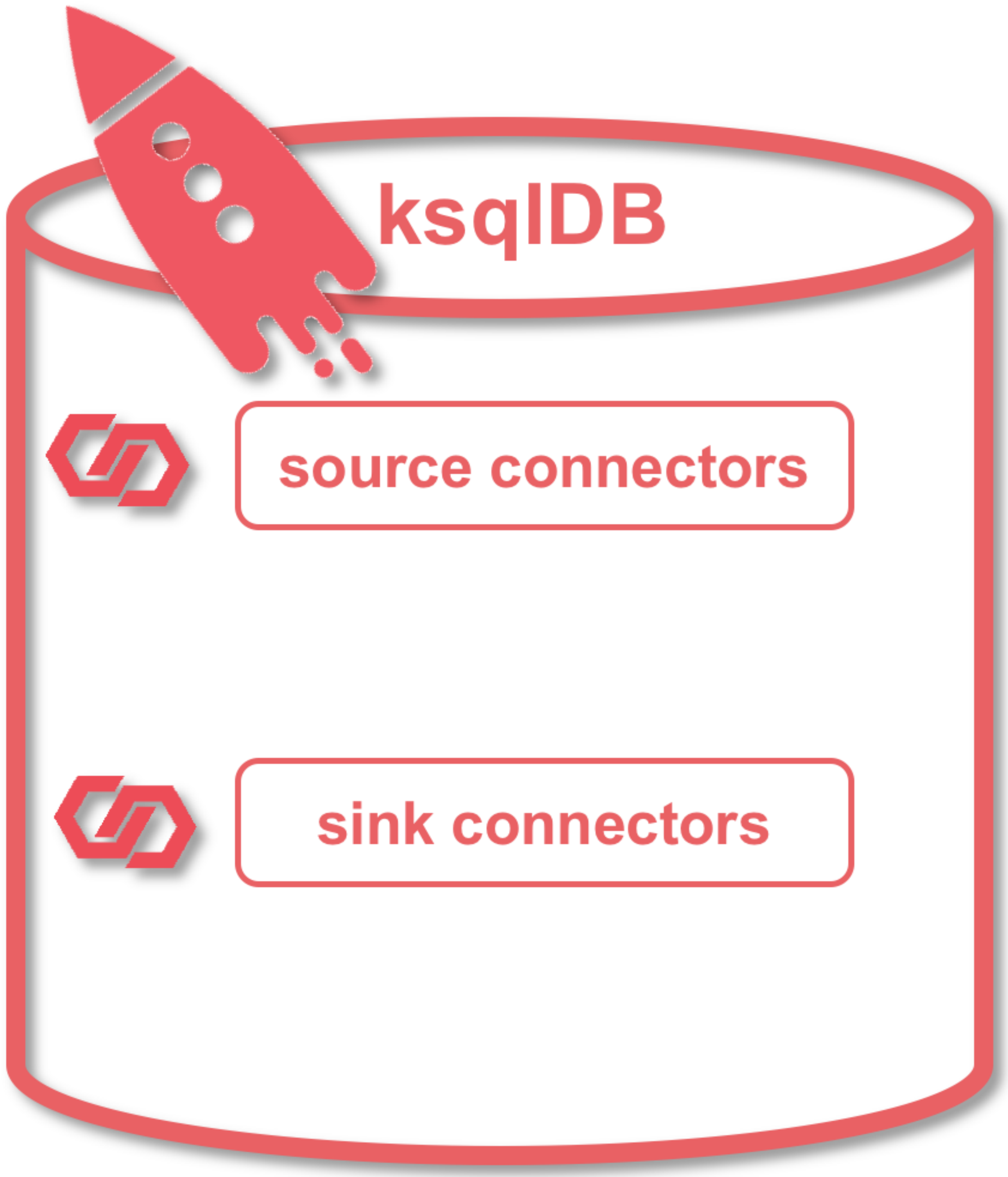


What if I told you it's as **easy to build a STREAMING app** as it is to build a CRUD app?

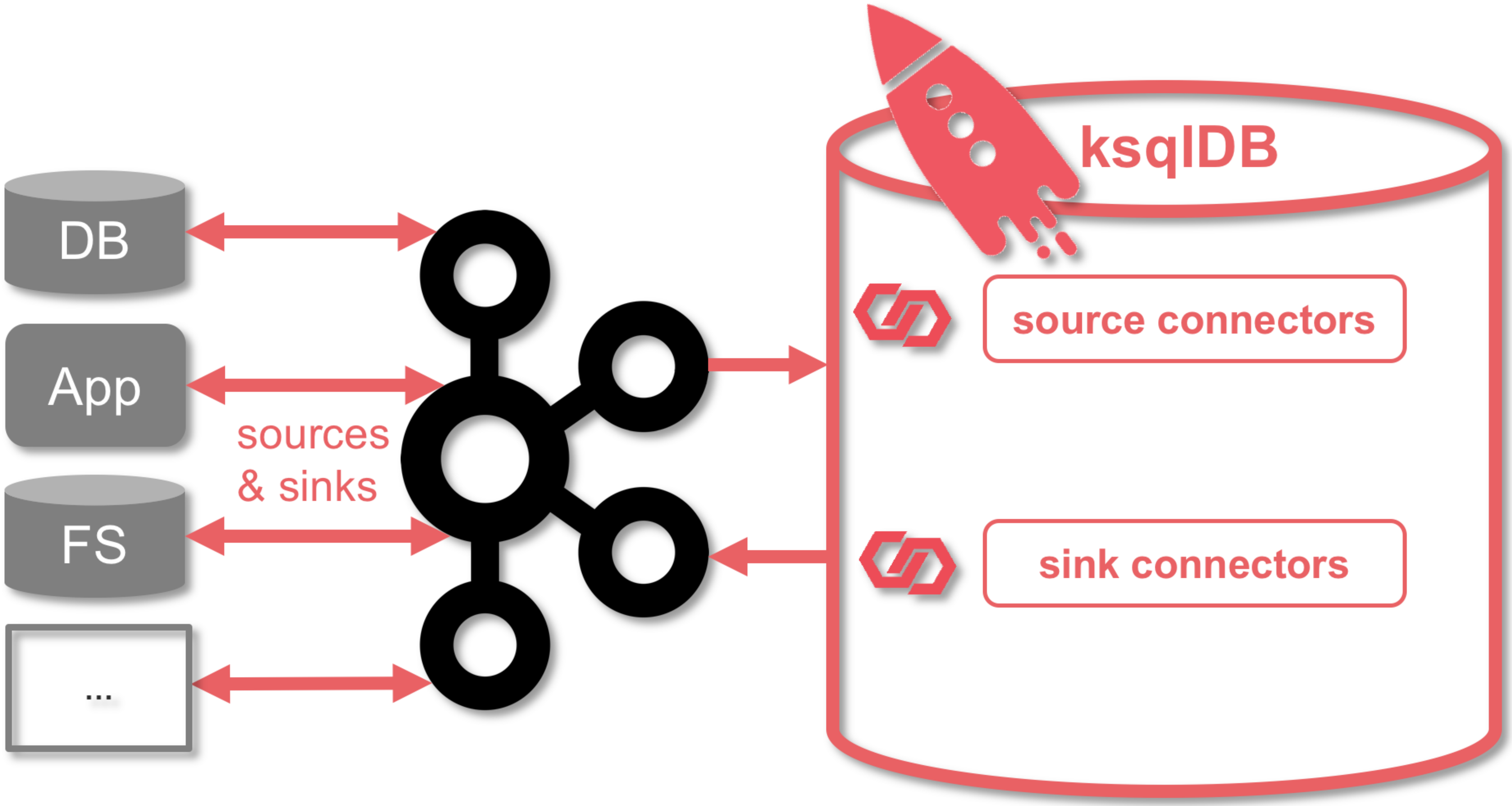
# Unified Streaming Architecture



# Unified Streaming Architecture



# Unified Streaming Architecture



# Connecting with Sources

```sql
CREATE SOURCE CONNECTOR `mysql-src-demo` WITH (
        'connector.class'='i.c.c.j.JdbcSourceConnector',
        'value.converter'='o.a.k.c.j.JsonConverter',
        'key.converter'='o.a.k.c.j.JsonConverter',
        'connection.url'='jdbc:mysql://localhost:3306/mydb',
        'connection.user'='...',
        'connection.password'='...',
        'table.whitelist'='...',
        'transforms'='...'
);
```

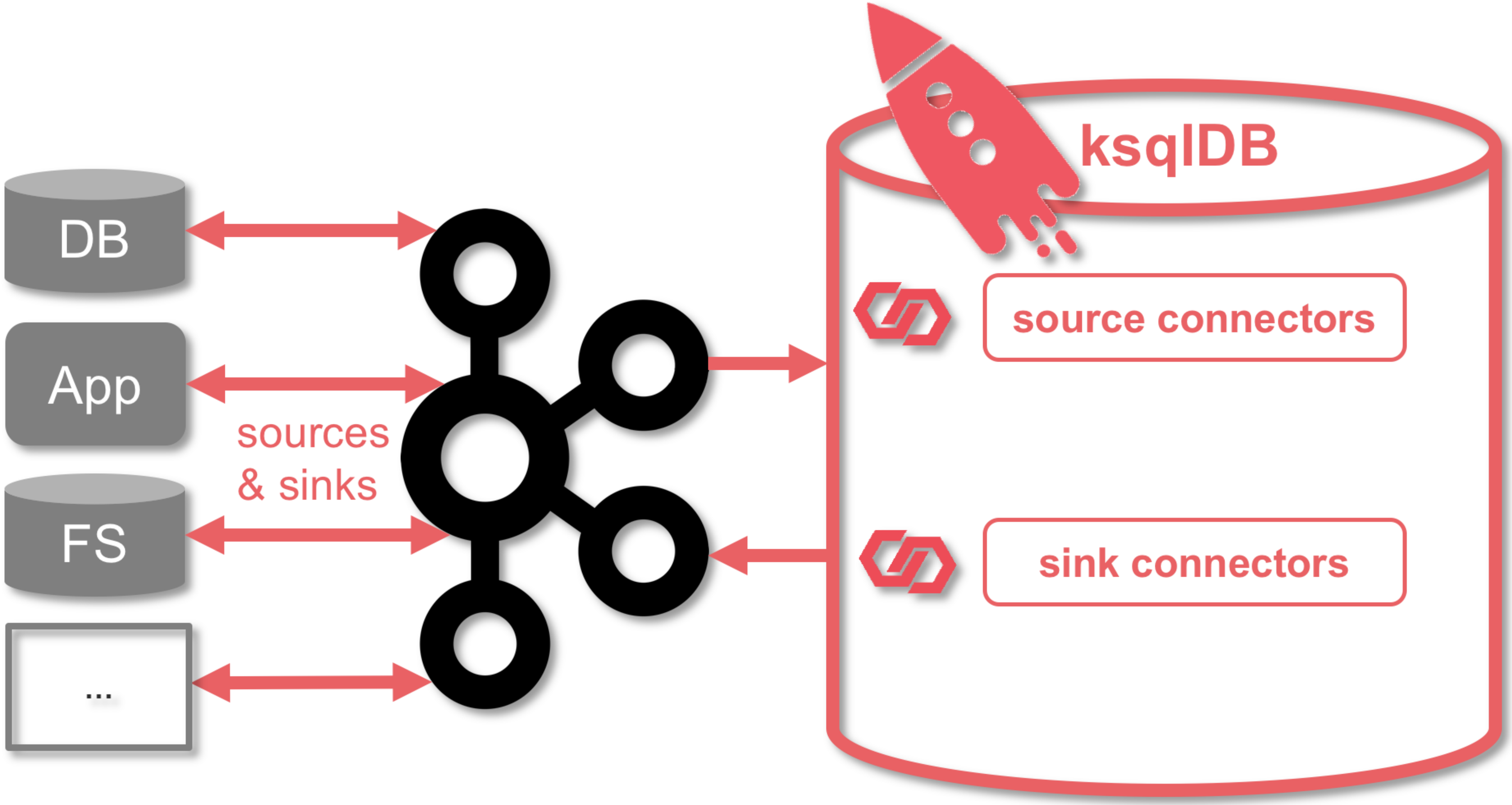
HINT: connector examples shown in demo later

# Connecting with Sinks

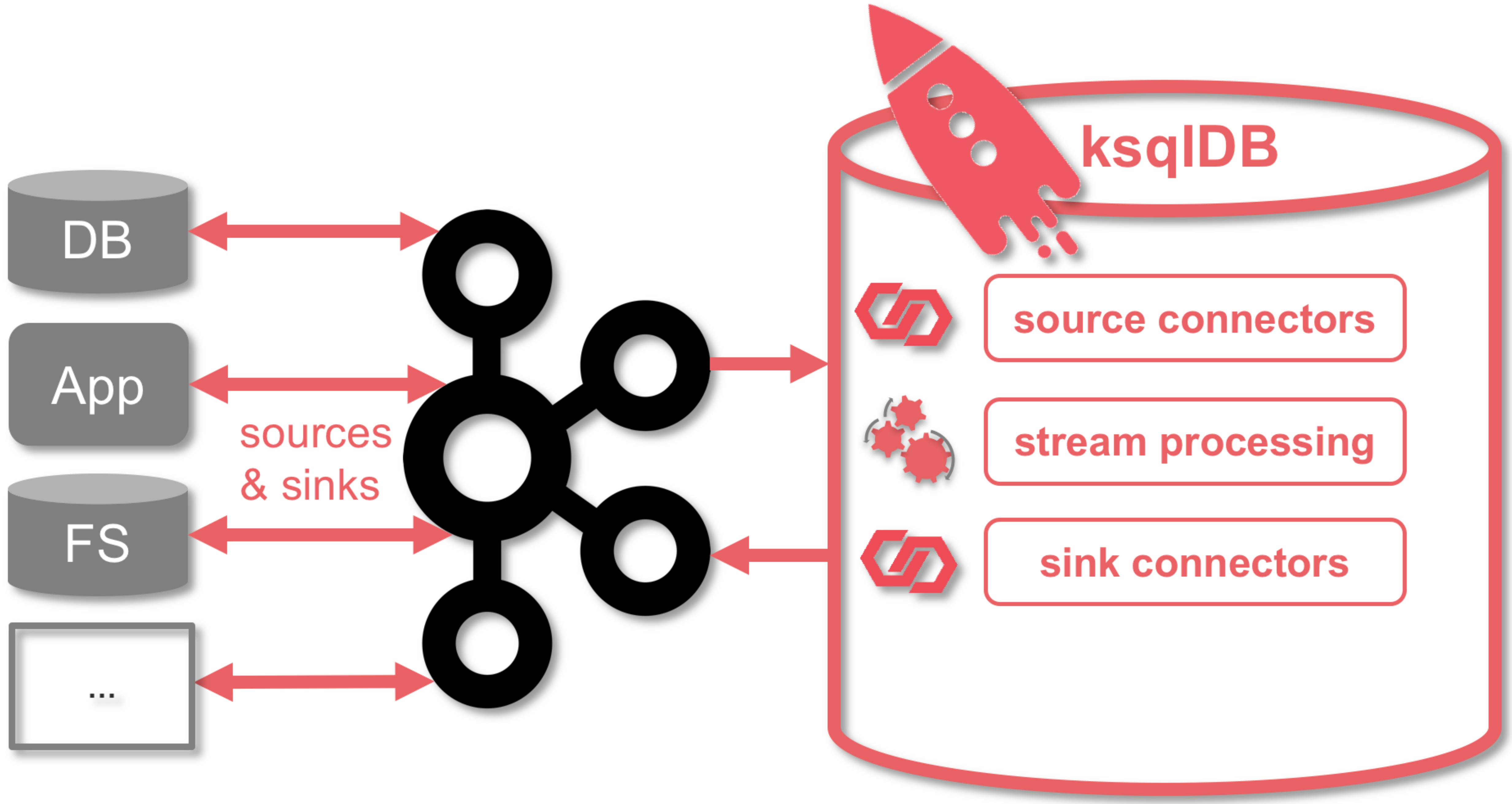
```
CREATE SINK CONNECTOR `mongodb-sink-demo` WITH (
      'topics'='...',
      'connector.class'='c.m.k.c.MongoSinkConnector',
      'key.converter'='o.a.k.c.s.StringConverter',
      'value.converter'='o.a.k.c.j.JsonConverter',
      'connection.uri'= 'mongodb://localhost:27017/kafka',
      'transforms'='...',
);
```

HINT: connector examples shown in demo later

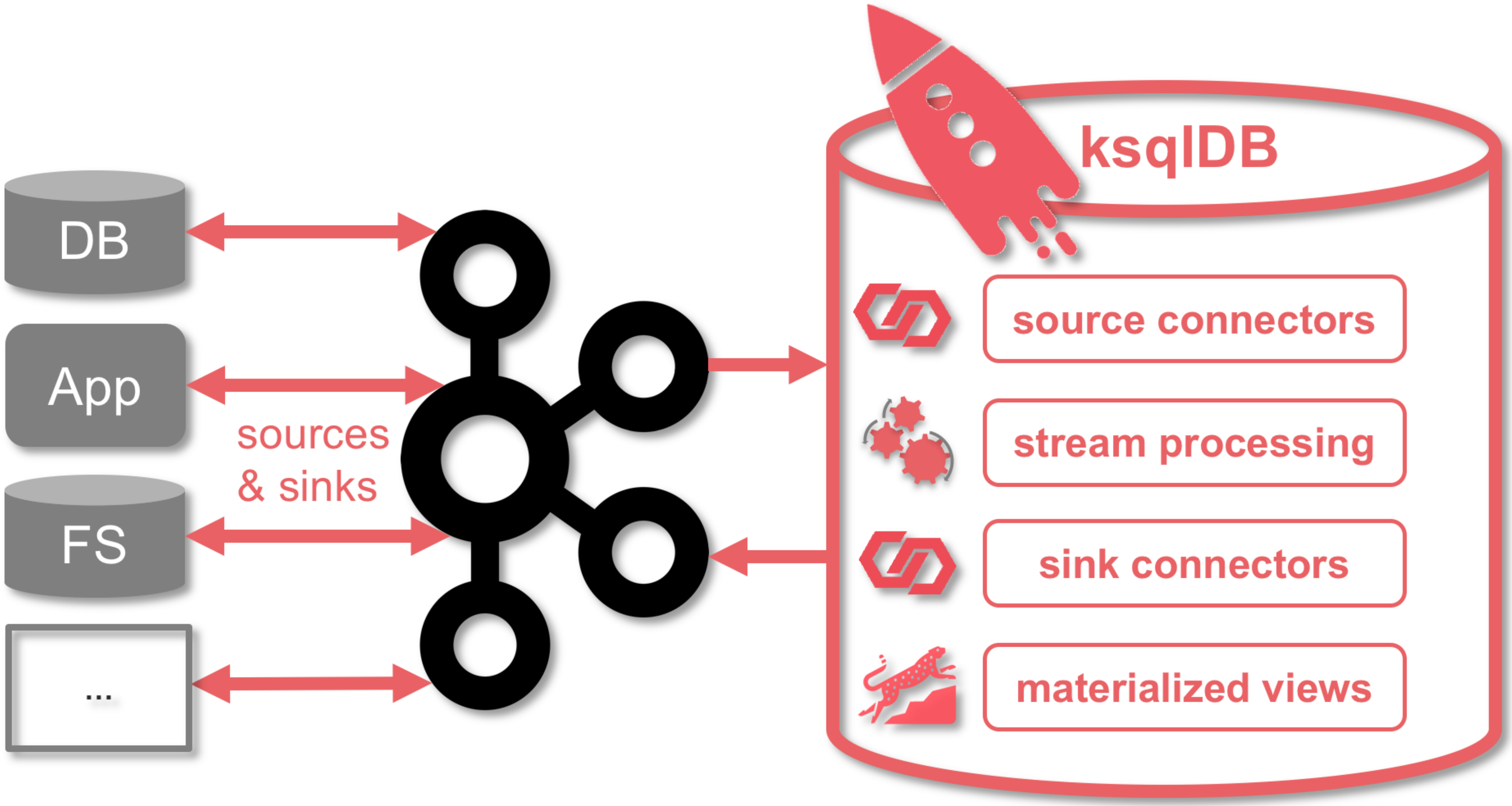
# Unified Streaming Architecture



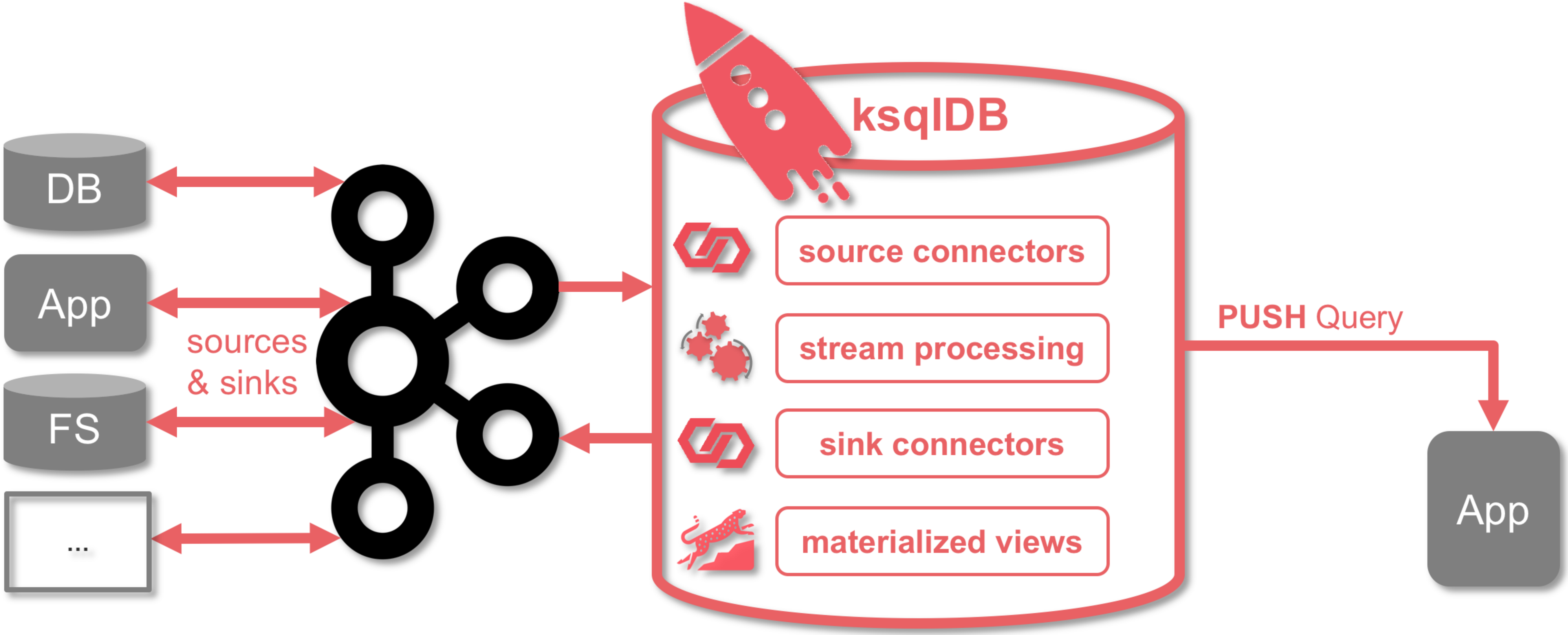
# Unified Streaming Architecture



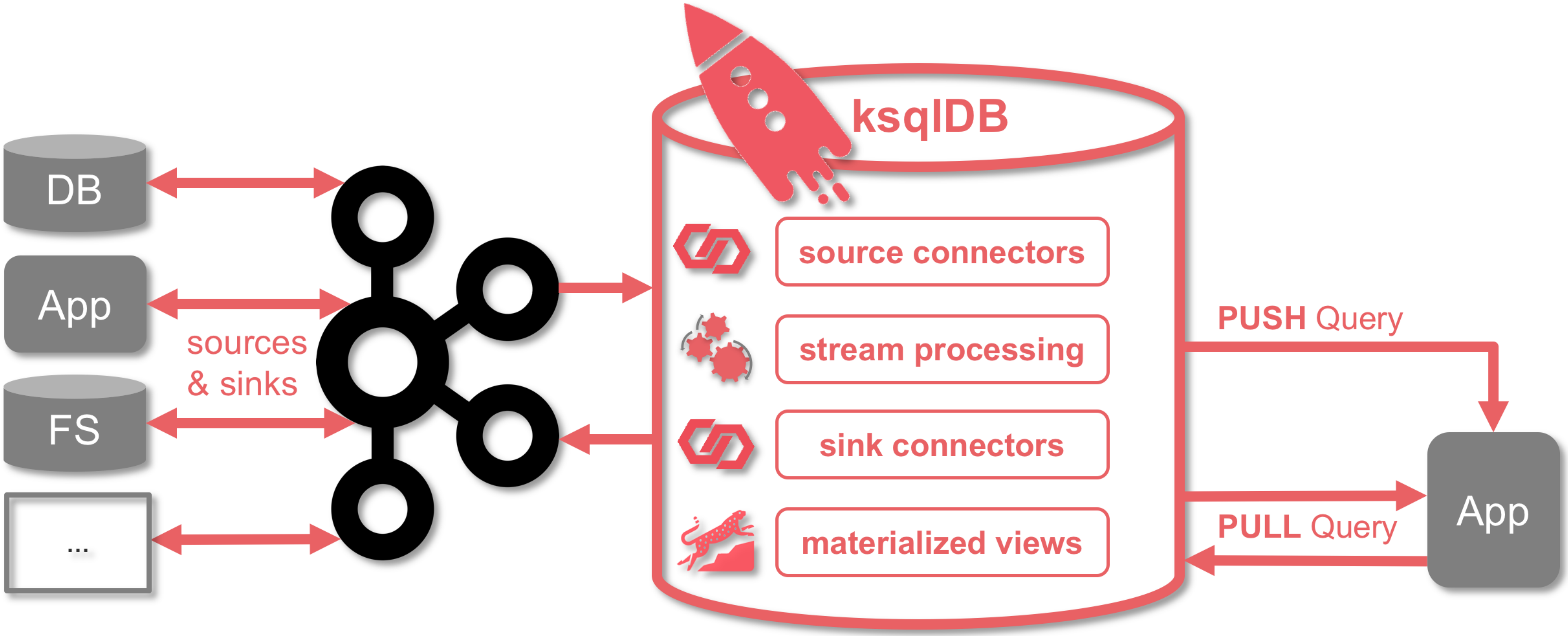
# Unified Streaming Architecture

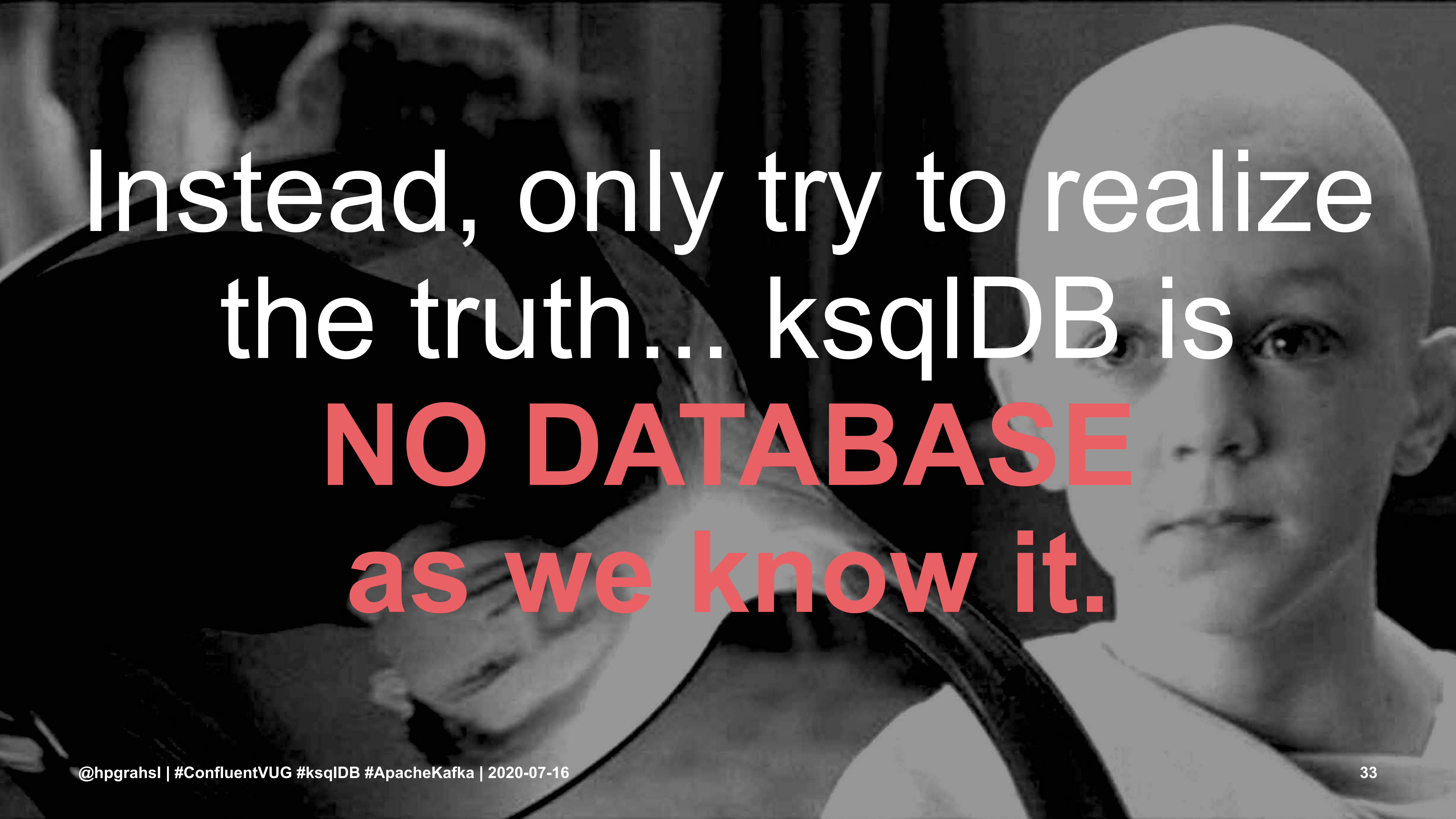


# Unified Streaming Architecture



# Unified Streaming Architecture





# Instead, only try to realize the truth... ksqlDB is

# **NO DATABASE as we know it.**

# Data Concepts in ksqlDB

## STREAM

- immutable append-only sequence
- captures events representing a series of facts

# Data Concepts in ksqlDB

```
CREATE STREAM ratings
    (userid INTEGER KEY,movieid INTEGER,stars INTEGER)
WITH (kafka_topic='ratings',value_format='JSON');
```

# Data Concepts in ksqlDB

## TABLE

- mutable collection of events
- holds the last known value for each key
- also result from stateful operations

# Data Concepts in ksqlDB

```
CREATE TABLE movies
    (movieid INTEGER PRIMARY KEY,title VARCHAR)
WITH (kafka_topic='movies',value_format='JSON');
```

# Data Concepts in ksqlDB

```
CREATE TABLE user_stats AS
    SELECT userid, MIN(stars) worst_rating, MAX(stars) best_rating,
        COUNT(*) num_ratings, AVG(stars) avg_rating
    FROM ratings
    GROUP BY userid;
```

# ksqlDB: PUSH Queries

- act as **subscription** to query results
- fit **asynchronous & reactive** data flows
- run **indefinitely**
- new data causes **continuous updates**

# ksqlDB: PUSH Query

```
ksql> SELECT
>       movieid, worst_rating, best_rating, num_ratings, avg_rating
>     FROM
>       movie_stats
>     EMIT CHANGES;

+-----------------+-----------------+-----------------+-----------------+-----------------+
|MOVIEID          |WORST_RATING     |BEST_RATING      |NUM_RATINGS      |AVG_RATING       |
+-----------------+-----------------+-----------------+-----------------+-----------------+
|123              |4                |5                |2                |4.5              |
|234              |3                |5                |2                |4.0              |
|345              |1                |5                |2                |3.0              |

Press CTRL-C to interrupt
```

NOT finished… more result rows as new movie ratings arrive!

# ksqlDB: PULL Queries

- fetch **point-in-time** results
- fit **request / response** data flows
- terminate **immediately**
- **lookup current state** of materialized views

# ksqlDB: PULL Query

```
ksql> SELECT
>       movieid,worst_rating,best_rating,num_ratings,avg_rating
>     FROM
>       movie_stats WHERE movieid = 234;

+-----------------+-----------------+-----------------+-----------------+-----------------+
|MOVIEID          |WORST_RATING     |BEST_RATING      |NUM_RATINGS      |AVG_RATING       |
+-----------------+-----------------+-----------------+-----------------+-----------------+
|234              |3                |5                |2                |4.0              |

Query terminated
```

finished! this is the result based on all available ratings right NOW!

```
ksql>
```

# ksqlDB **Functions**

- choose from three categories
  - **scalar** ➔ UDF
  - **aggregation** ➔ UDAF
  - **table** ➔ UDTF


EXPLODE
SUM
AVG
MIN
MAX
COUNT
COLLECT_LIST
COLLECT_SET
EARLIEST_BY_OFFSET
LN
ABS
CEIL
HISTOGRAM
CUBE_EXPLODE
COUNT_DISTINCT
EXP
TOPK
RANDOM
ROUND
FLOOR
SQRT
GEO_DISTANCE
ARRAY_MAX
ARRAY_CONTAINS
ARRAY
MASK
ARRAY_LENGTH
UCASE
REPLACE
COALESCE
CONCAT_WS
MAP_VALUES
SPLIT
ELT
AS_VALUE
LATEST_BY_OFFSET
ARRAY_SORT
MASK_LEFT
INITCAP
MAP
ARRAY_DISTINCT
TOPKDISTINCT
ARRAY_UNION
IFNULL
ARRAY_JOIN
LEN
GENERATE_SERIES
MAP_UNION
SUBSTRING
FIELD
RPAD
UUID
EXTRACTJSONFIELD
ARRAY_MIN
LPAD
STRINGTODATE
CHR
ENTRIES
SLICE
CONCAT
MAP_KEYS
AS_MAP
INSTR
MASK_KEEP_LEFT
MASK_RIGHT
ARRAY_INTERSECT
LCASE
UNIX_DATE
ENCODE
SIGN
TRIM
ARRAY_EXCEPT
REGEXP_EXTRACT
REGEXP_REPLACE
DATETOSTRING
SPLIT_TO_MAP

# Criteria 2: Extensibility Options

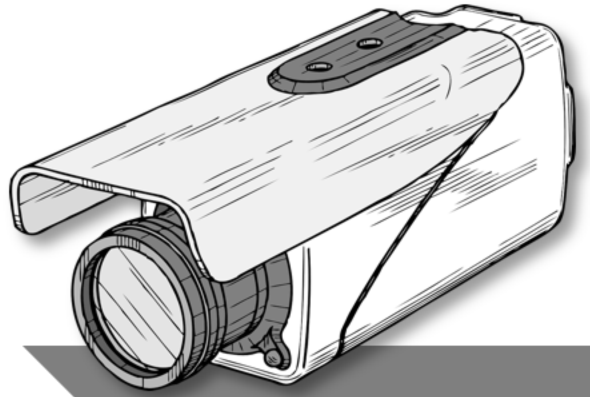
- custom functions (UDFs, UDFAs & UDTFs)
- enable flexbile & powerful capabilities
- but Java code needed

HINT: custom UDF example shown in demo later

# Fictional Use Case

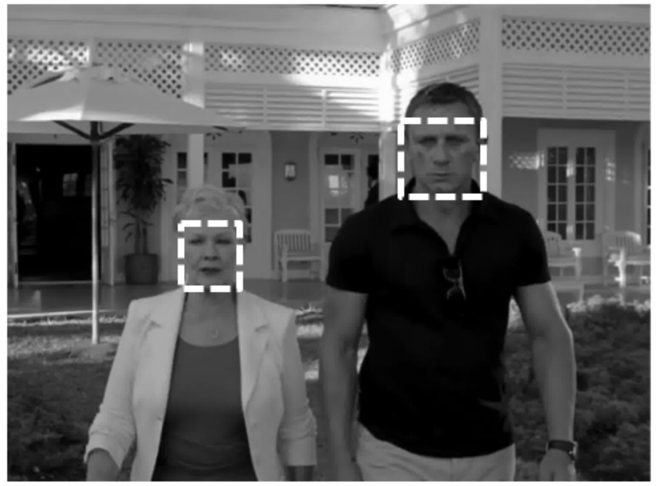
# ksqlDB Use Case


# ksqlDB Use Case

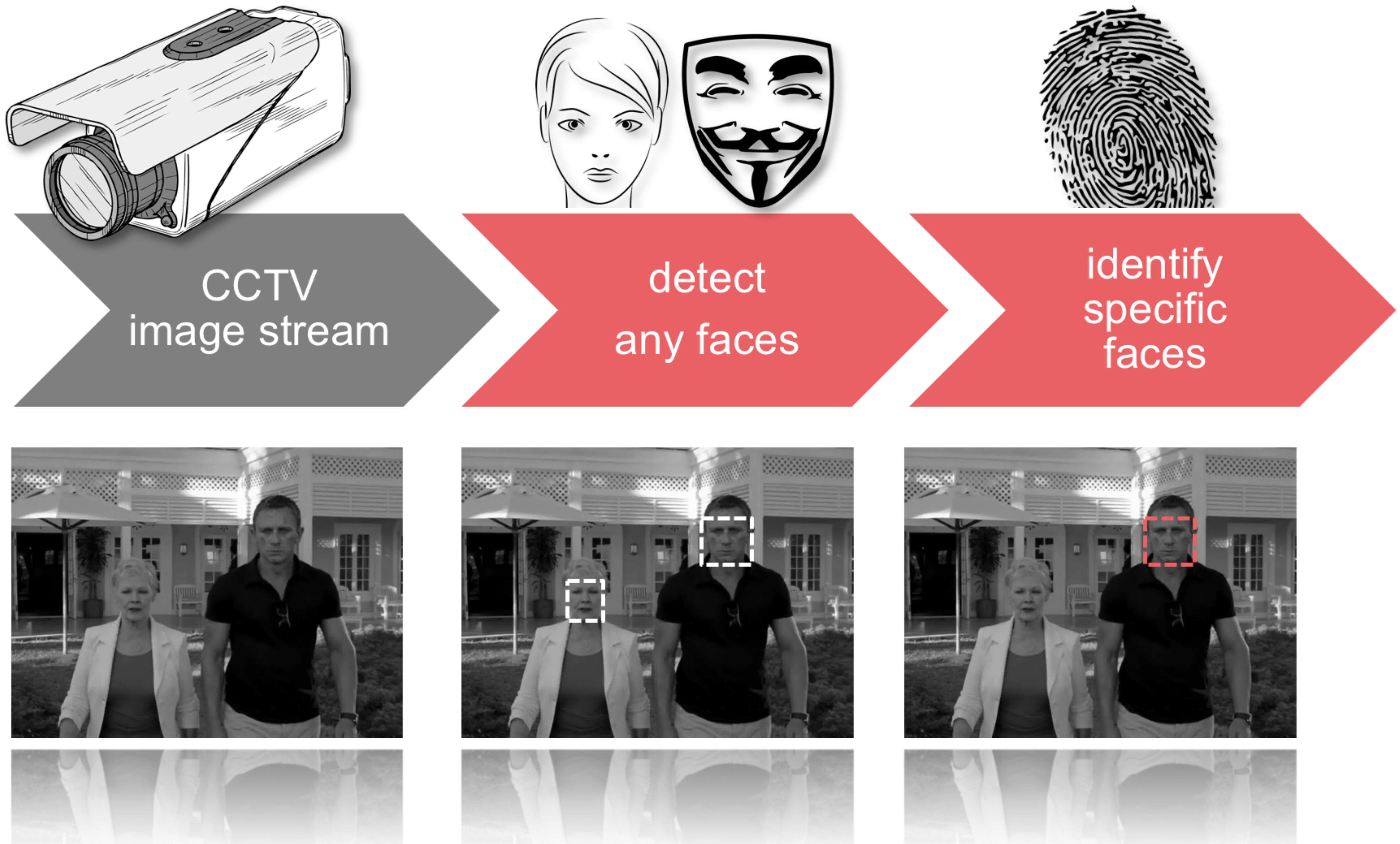


CCTV
image stream

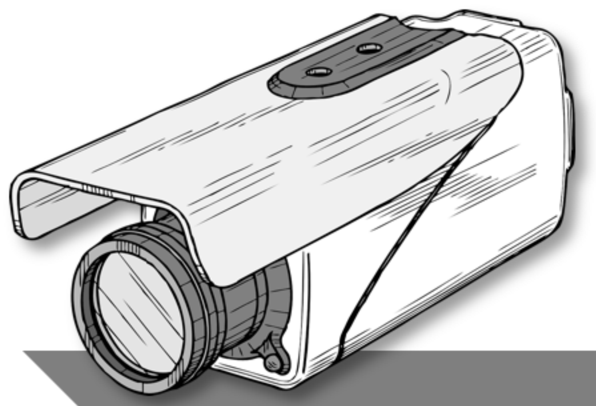
detect
any faces

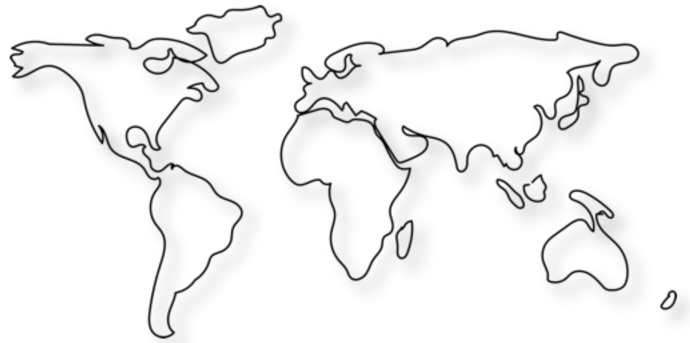


# ksqlDB Use Case



# ksqlDB Use Case



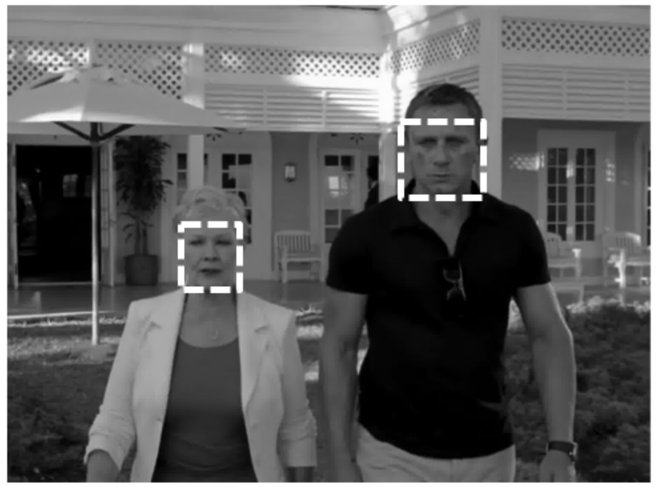


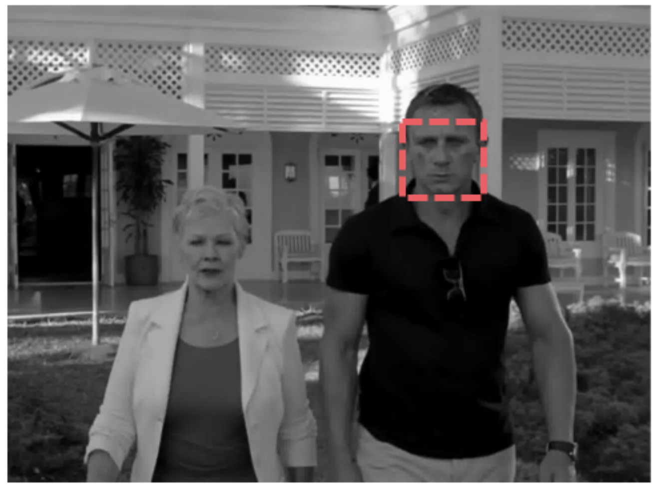
CCTV image stream

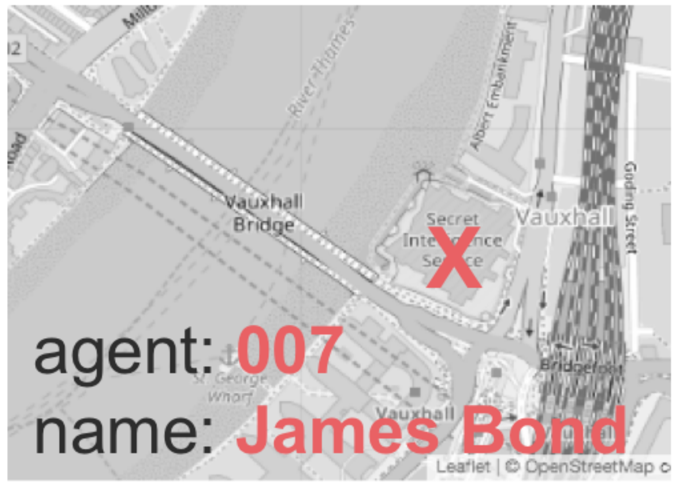
detect any faces

identify specific faces

enrich with location & personal data

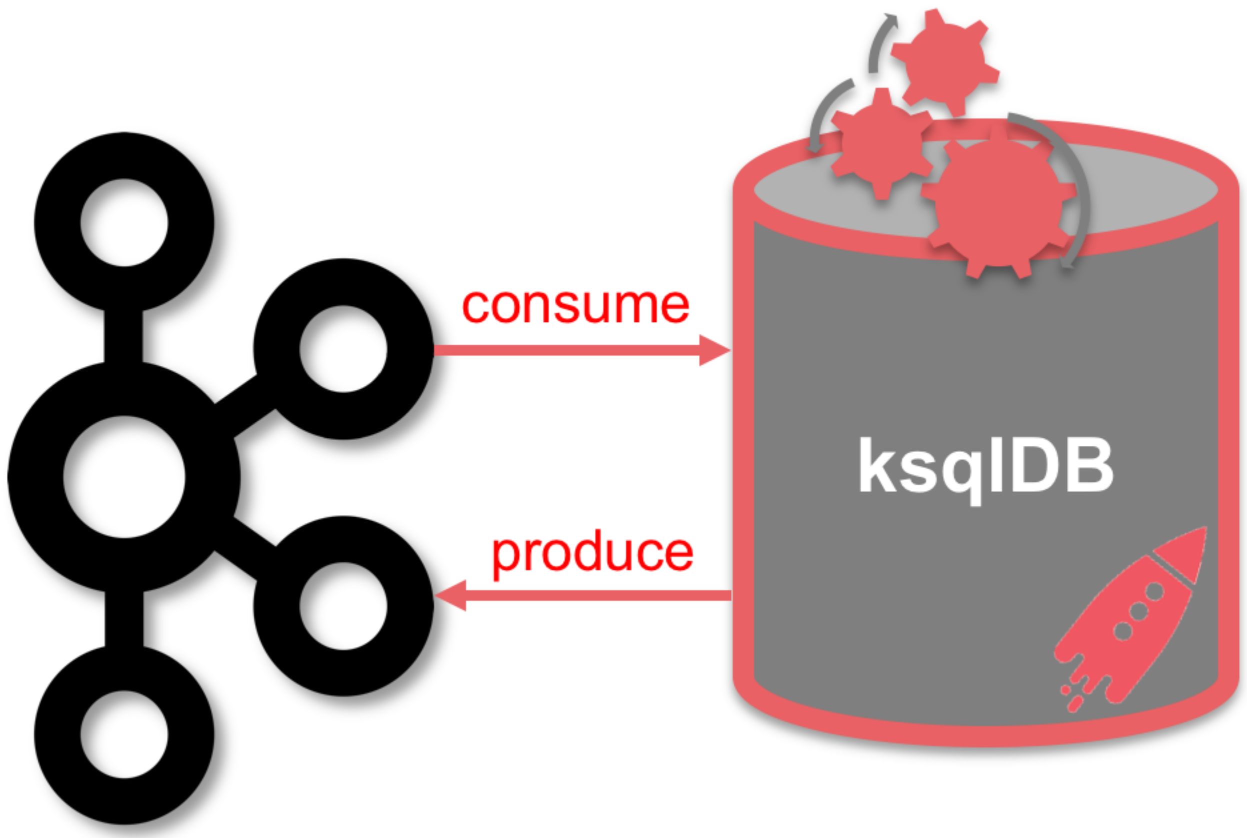




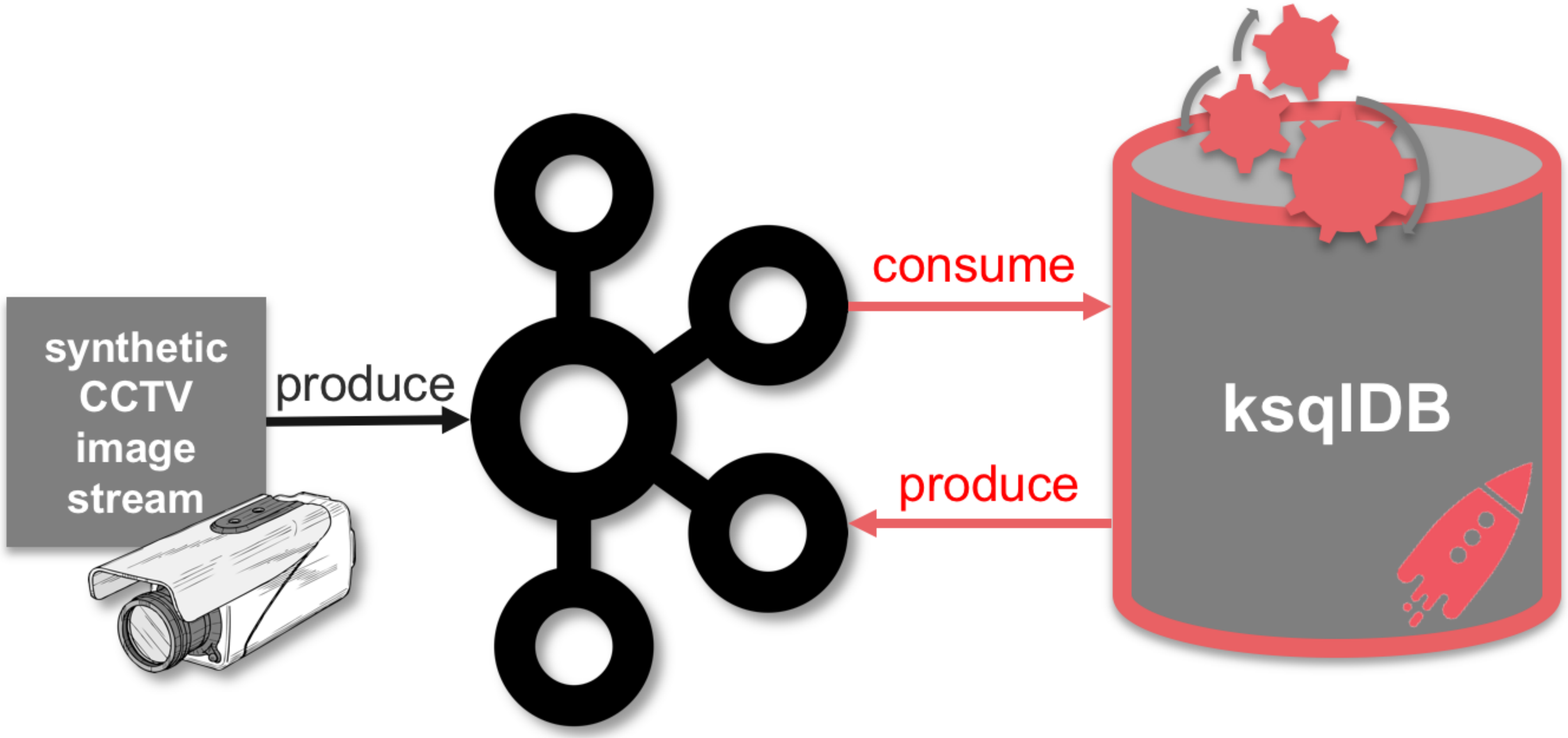


# Data Architecture

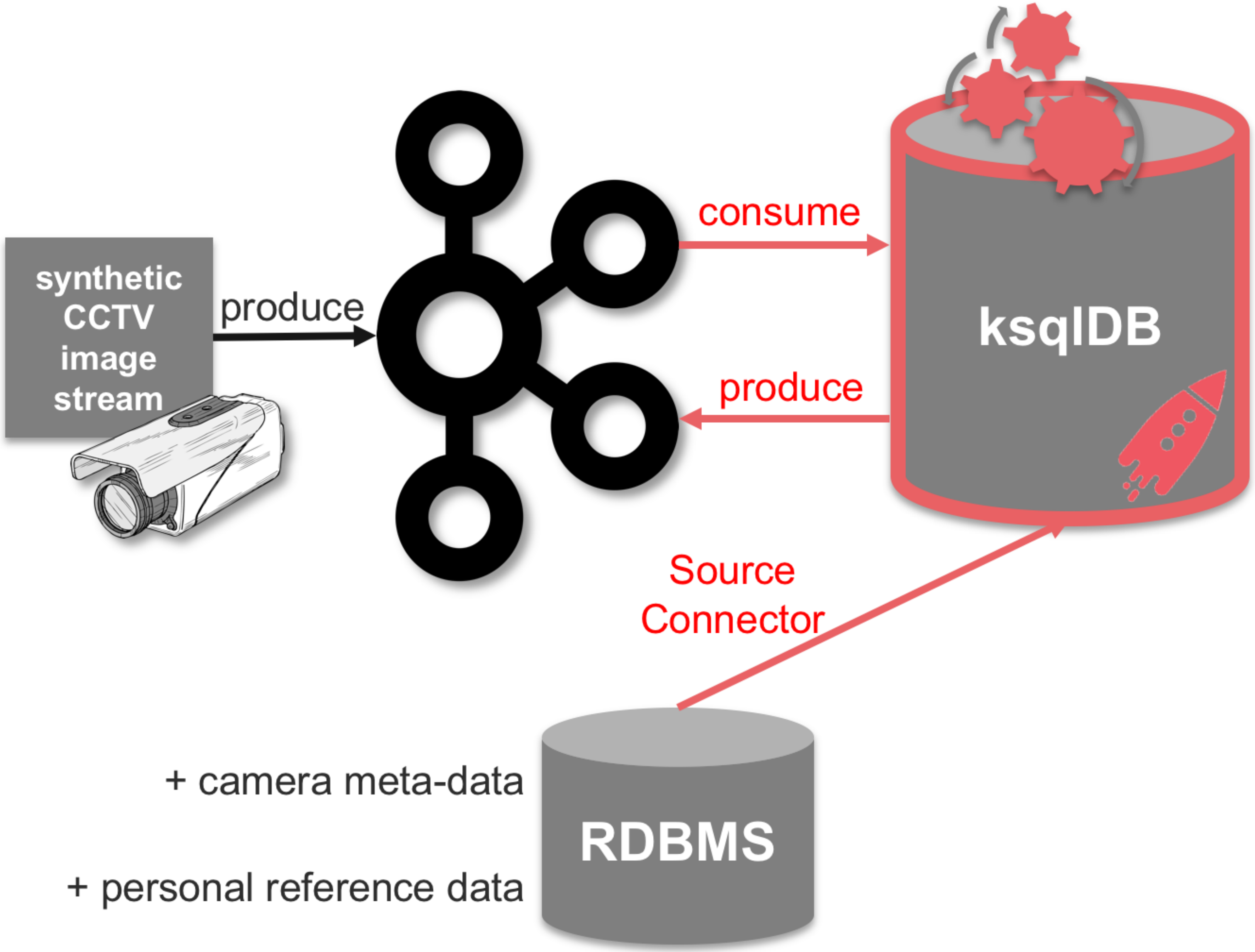
# Data Architecture



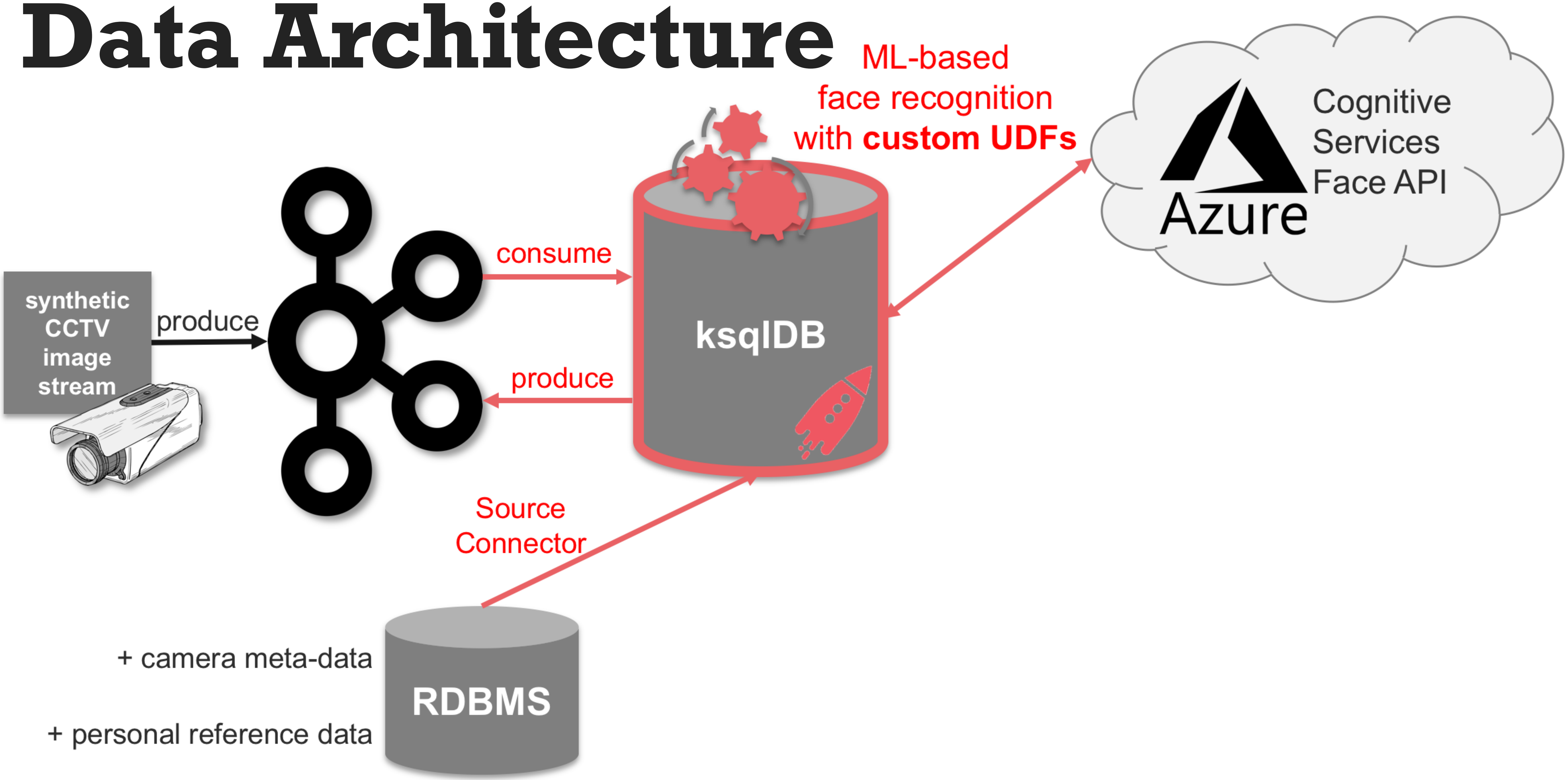
# Data Architecture



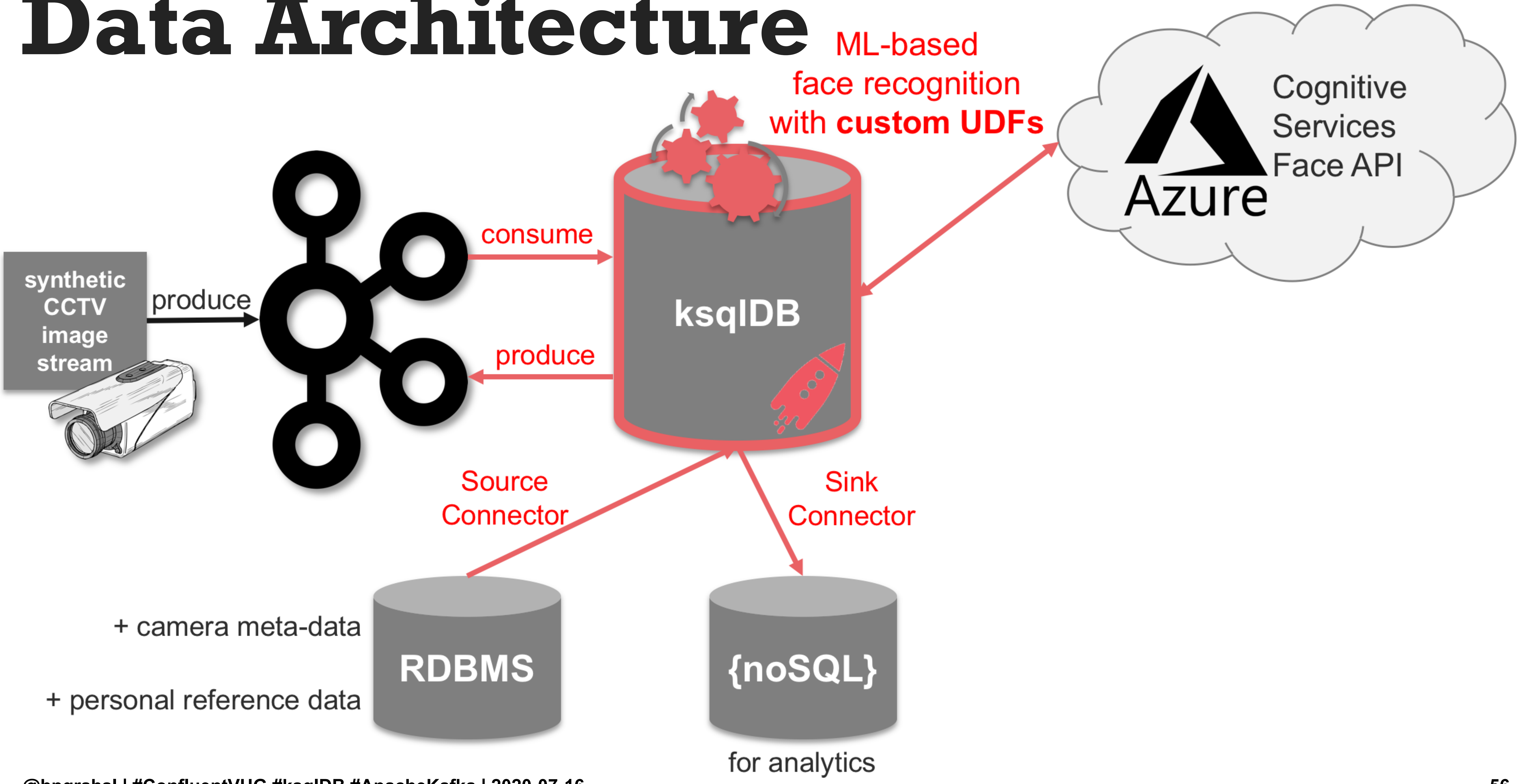
# Data Architecture



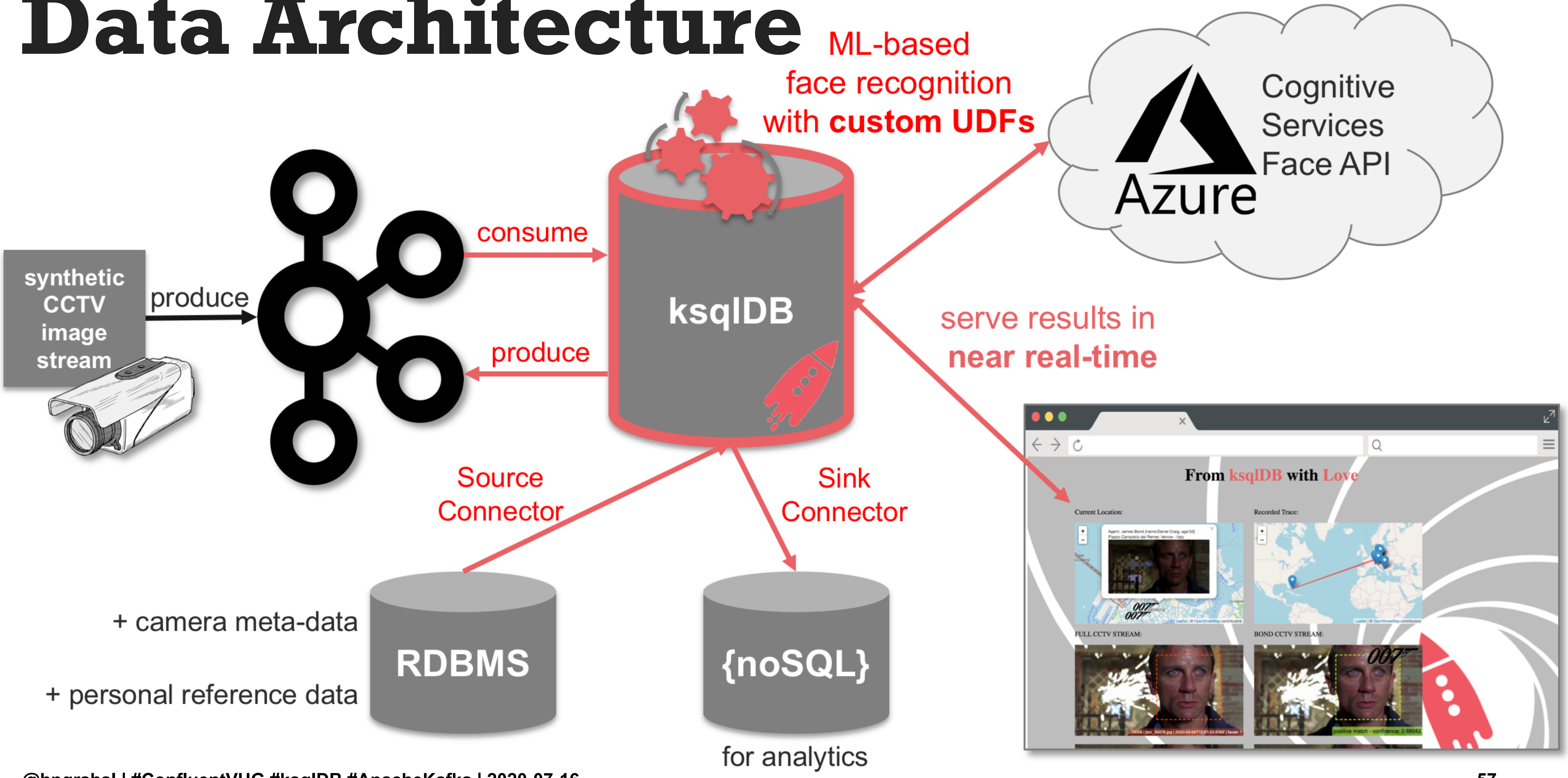
# Data Architecture



# Data Architecture



# Data Architecture



# Criteria 3: ML Integration Paths

- call **fully-managed ML services** (external)
- run your **own model server** (co-located)
- package **home-brewed model** into UDF (embedded)

- **completely separated:** integrate ML results via Connectors

# Everywhere…

## ksqlDB runs everywhere

# Criteria 4: Deployment Options

- run **however you want**
  - bare metal, VMs, containers
- deploy **wherever you need**
  - on-premises, private / public cloud, hybrid
- something **fully-managed?**

CONFLUENT CLOUD

# Apache Kafka®
# Re-engineered for the Cloud

Focus on building apps and not managing clusters with a scalable, resilient and secure event streaming platform. Event streaming with Kafka made simple on AWS, Azure and GCP clouds.

aws    Microsoft Azure    Google Cloud

TRY FREE

Unfortunately, no one can be told what **ksqlDB** is... You have to **TRY IT** for yourselves!

**https://ksqldb.io**

HOME OVERVIEW QUICKSTART EXAMPLES DOCS COMMUNITY

# ksqlDB

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

GET STARTED GET THE CODE

## Real, real-time

Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.

## Kafka-native

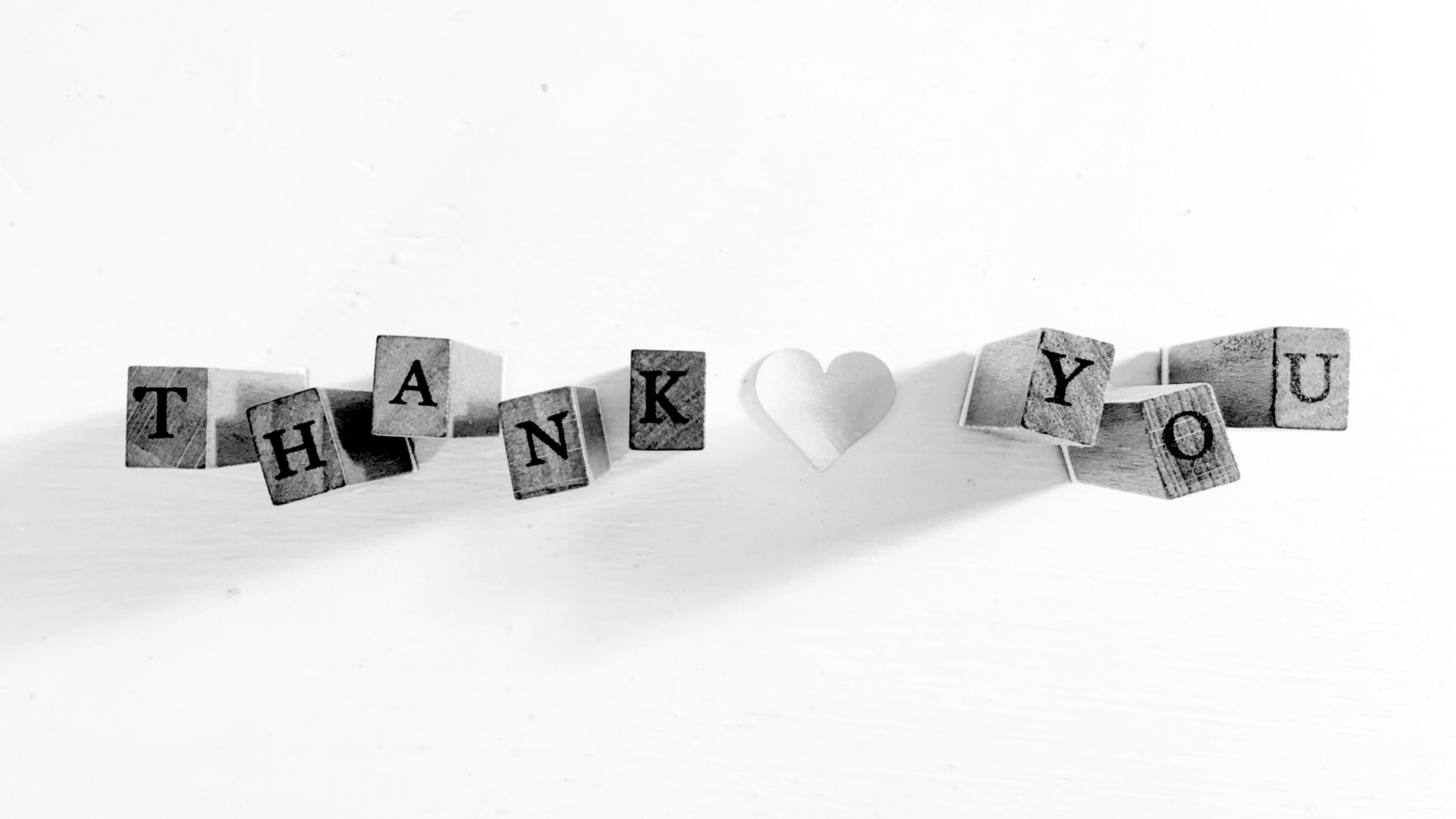
Seamlessly leverage your existing Apache Kafka® infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.

## What, not how

Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

# reach out to me

**@hpgrahsl**


THANK YOU