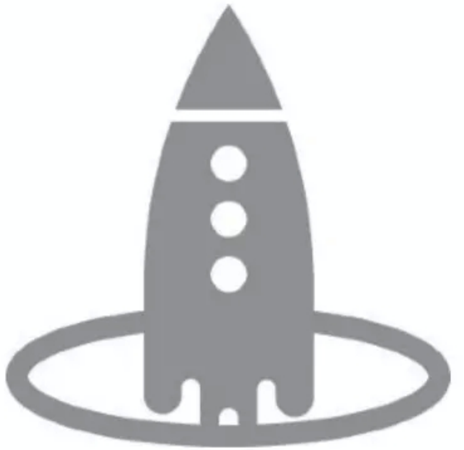
confluent

KSQL in Practice

Almog Gavra
Engineer, KSQL

- **Intro**
- Develop
- Deploy
- Operate
- Common Mistakes
- Q&A



Why KSQL?

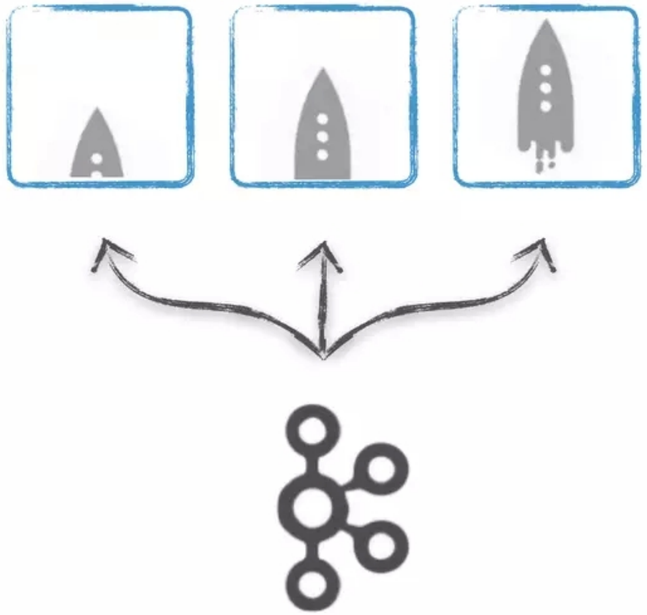
Why KSQL?

Performance and Robustness from Apache Kafka®

Why KSQL?

Simplified Stream Processing

Performance and Robustness from Apache Kafka®

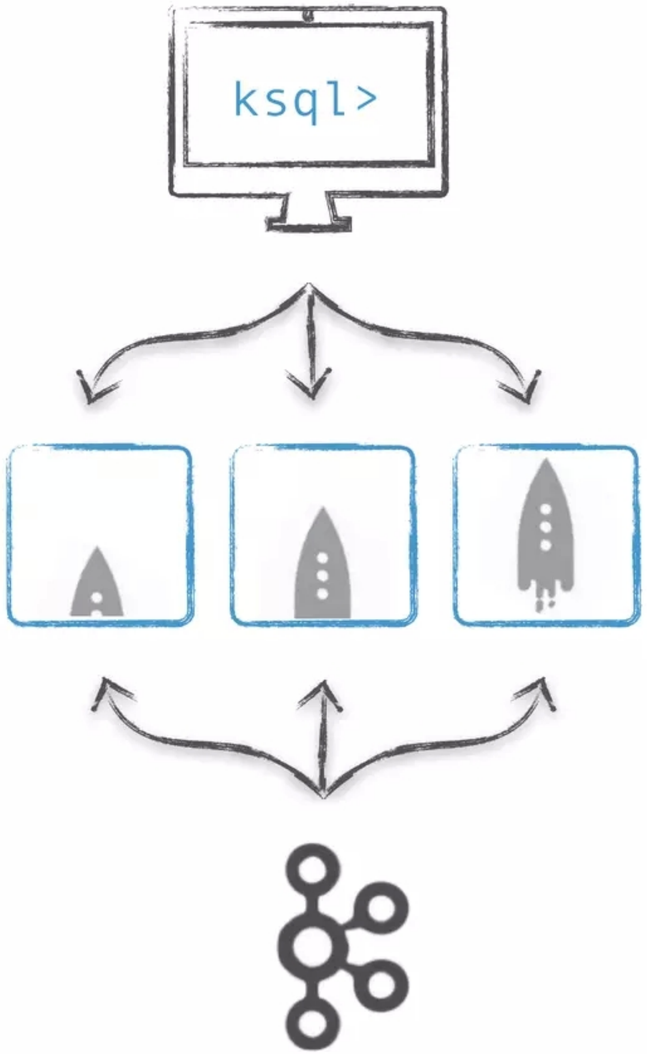


Why KSQL?

Familiar & Expressive SQL-Like Language

Simplified Stream Processing

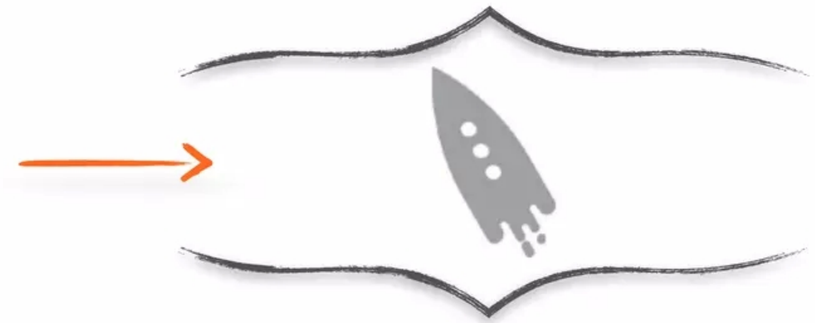
Performance and Robustness from Apache Kafka®



```
SELECT * FROM authorizations
WHERE card_number = 123;
```

User submits SQL

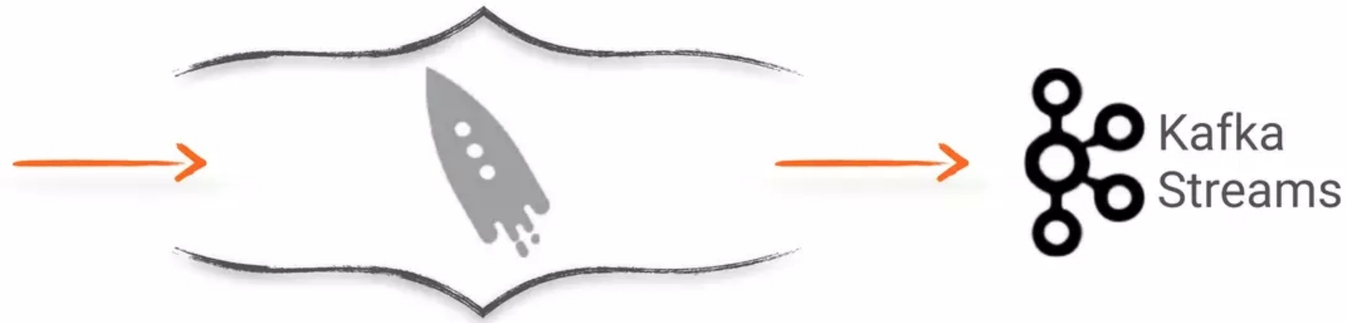
```
SELECT * FROM authorizations
WHERE card_number = 123;
```



User submits SQL

KSQL builds
KafkaStreams topology

```
SELECT * FROM authorizations
WHERE card_number = 123;
```



User submits SQL

KSQL builds
KafkaStreams topology

KafkaStreams
executes the topology

Fraud Detection via KSQL

“find all card numbers that had more than 3 authorization attempts within 5 seconds”

Fraud Detection via KSQL

```
CREATE TABLE possible_frauds AS
SELECT card_number, count(*)
FROM authorizations
GROUP BY card_number
HAVING count(*) > 3;
```

“find all card numbers that had more than 3 authorization attempts within 5 seconds”

Fraud Detection via KSQL

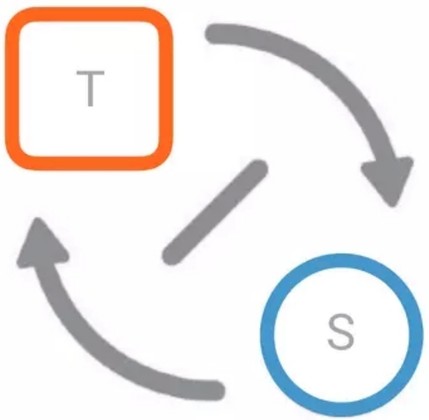
```
CREATE TABLE possible_frauds AS
SELECT card_number, count(*)
FROM authorizations

WINDOW TUMBLING (SIZE 5 SECONDS)

GROUP BY card_number
HAVING count(*) > 3;
```

“find all card numbers that had more than 3 authorization attempts within 5 seconds”

- Intro
- **Develop**
- Deploy
- Operate
- Common Mistakes
- Q&A

but first…

but first... **Stream/Table Duality**




A
key
value


T


topic
A
X
B
Y
C


S


A
key
value


T


A


topic
A
X
B
Y
C


S

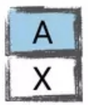

A


A
key
value


T


topic
A
X
B
Y
C


S


A
X

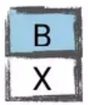

A
X


A
key
value


T


topic
A
X
B
Y
C


S


B
X

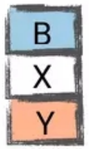

A
X
B


A
key
value


T


topic
A
X
B
Y
C


S


B
X
Y

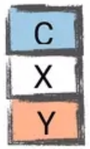

A
X
B
Y

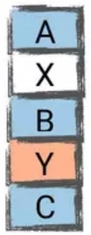

A
key
value


T

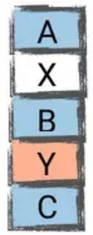

topic
A
X
B
Y
C


S

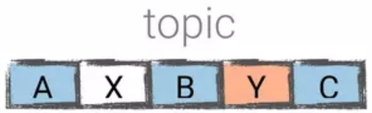

C
X
Y


A
X
B
Y
C

- a **table** is the *current* state of your data
- the topic is a *changelog*



T



S

- a **table** is the *current* state of your data
- the topic is a *changelog*

- a **stream** is the *historical* state of your data
- the topic is a *sequence*

Development Lifecycle

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

Kafka Connect

```
INSERT INTO foo VALUES
```

*

```
./ksql-datagen
```

* Feature Coming in CP 5.3

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

```
CREATE TABLE/STREAM name (schema …)
WITH (kafka_topic=“topic”);
```

DDL

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

```
CREATE TABLE/STREAM name (schema …)
WITH (kafka_topic="topic");
```

DDL

```
CREATE TABLE/STREAM name (schema …)
AS SELECT * FROM source;
```

DML

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

UDF

UDAF

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

UDF

```
SELECT amount, OBFUSCATE(card_number) AS id
FROM AUTHORIZATIONS;
```

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

UDAF

```
SELECT card_number, SUM(cost) AS total_cost
FROM AUTHORIZATIONS WINDOW TUMBLING (SIZE 7 DAYS)
GROUP BY card_number;
```

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

UDAF

```
SELECT card_number, SUM(cost) AS total_cost
FROM AUTHORIZATIONS WINDOW TUMBLING (SIZE 7 DAYS)
GROUP BY card_number;
```

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

`ksql-test-framework`*

```
SELECT * FROM stream LIMIT 3
```

* Feature Coming in CP 5.3

Import Data

Create Stream/Table

Write KSQL Queries

Verify Output

ksql-test-framework*

```
{
 "statements": [
    "CREATE STREAM TEST (source VARCHAR) WITH (kafka_topic='test_topic');",
    "CREATE STREAM O AS SELECT CONCAT('prefix-', source) AS C FROM TEST;"],
  "inputs": [
    {"topic": "test_topic", "value": {"source": "s1"}},
    {"topic": "test_topic", "value": {"source": "s2"}],
  "outputs": [
        {"topic": "O", "value": {"C":"prefix-s1"}},
        {"topic": "O", "value": {“C":"prefix-s2"}}]
}
```

* Feature Coming in CP 5.3

* Feature Coming in CP 5.3

- Intro
- Develop
- **Deploy**
- Operate
- Common Mistakes
- Q&A

Deployment Options

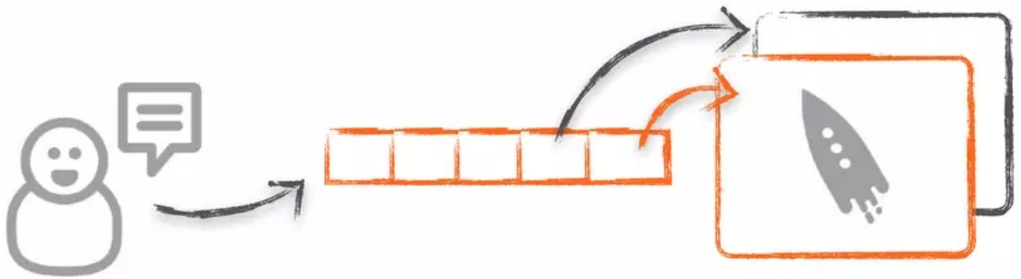
Deployment Options

Interactive Mode

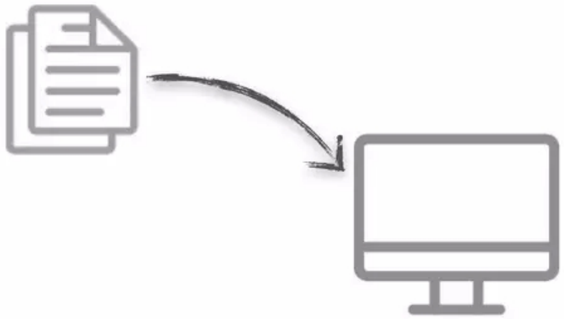
Deployment Options

Interactive Mode

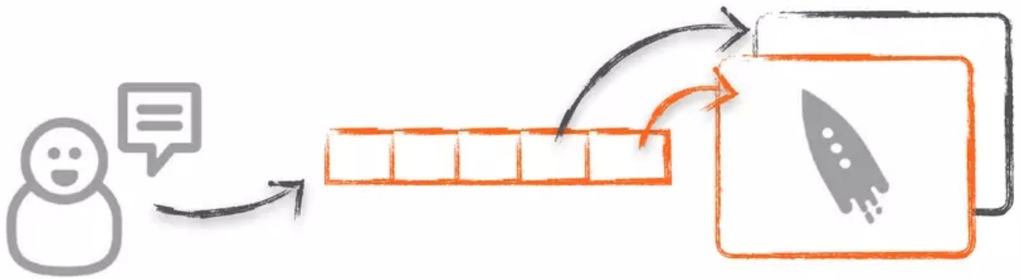
Headless Mode



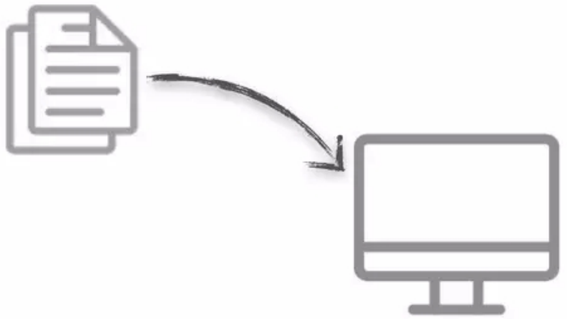
Interactive Mode



Headless Mode

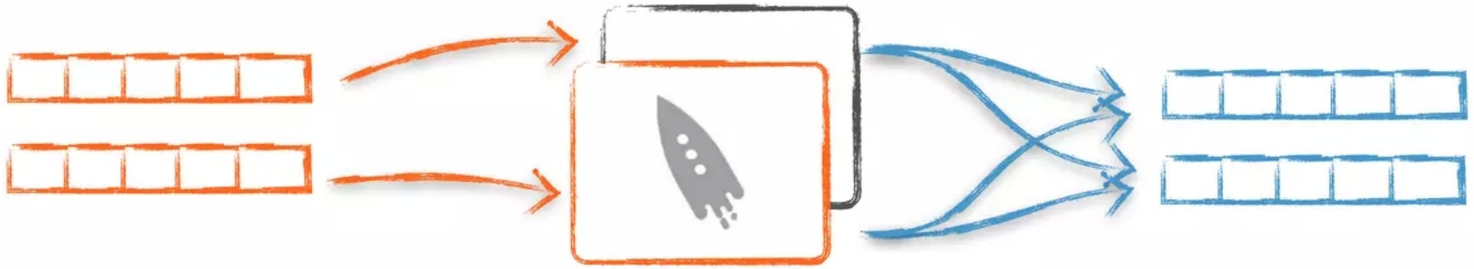


Interactive Mode

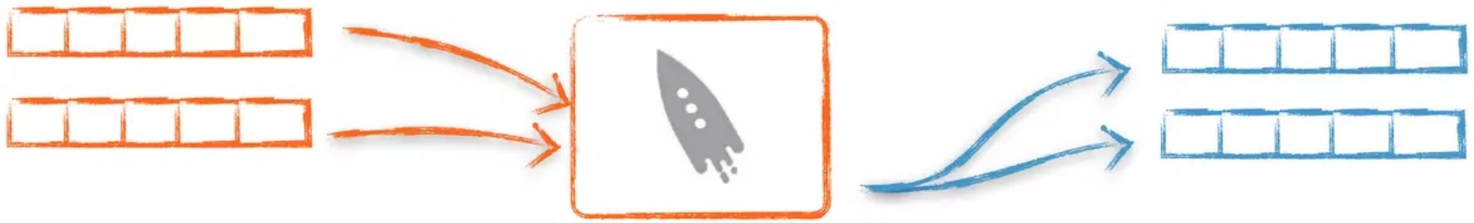


Headless Mode

KSQL scales relative
to partitions in Kafka



KSQL scales relative
to partitions in Kafka



… and rebalances if a
replica is lost

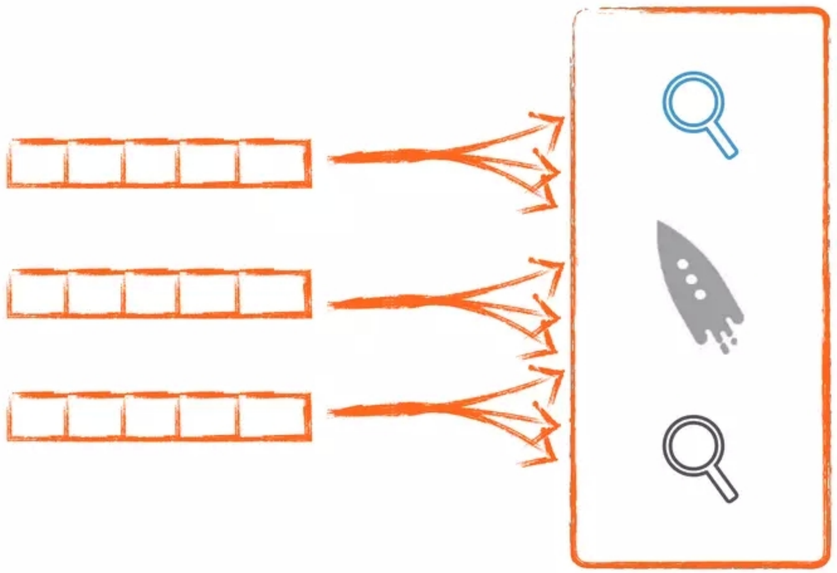
KSQL scales relative
to partitions in Kafka

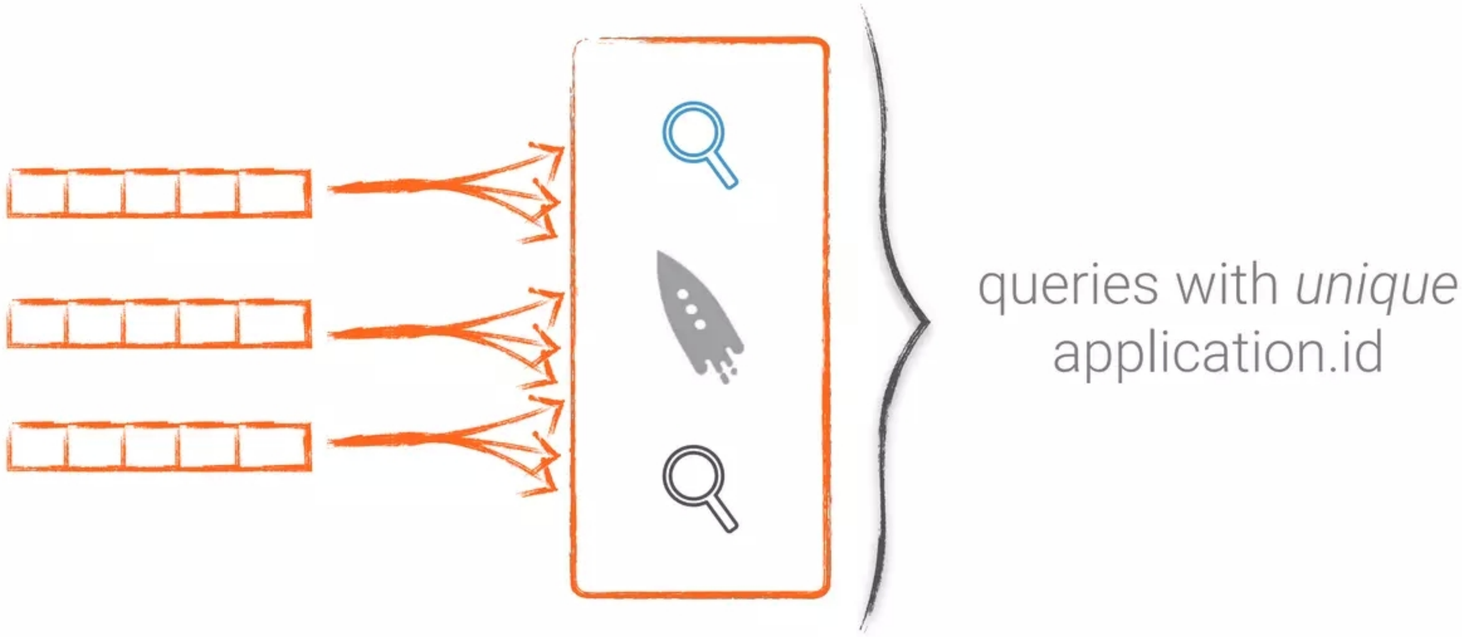
... and rebalances if a
replica is lost

✓ **tip:** initially over partitioning your input topics can help scale!

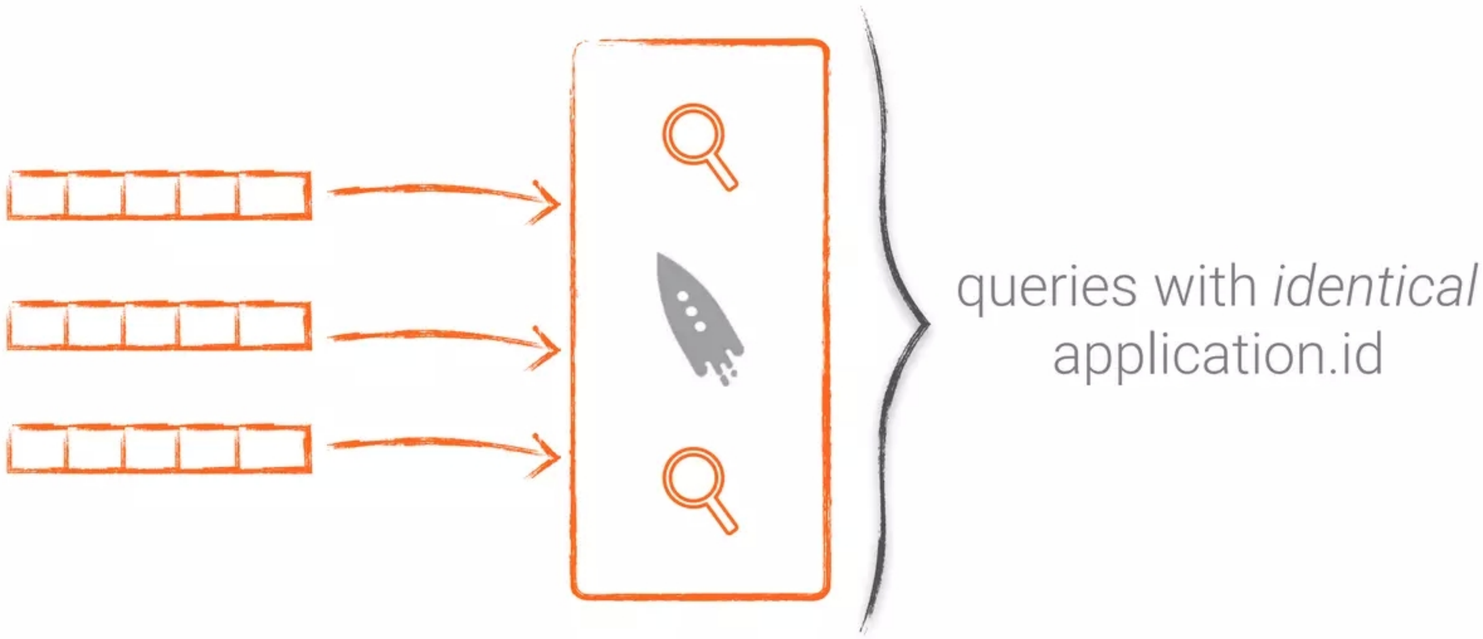
application.id

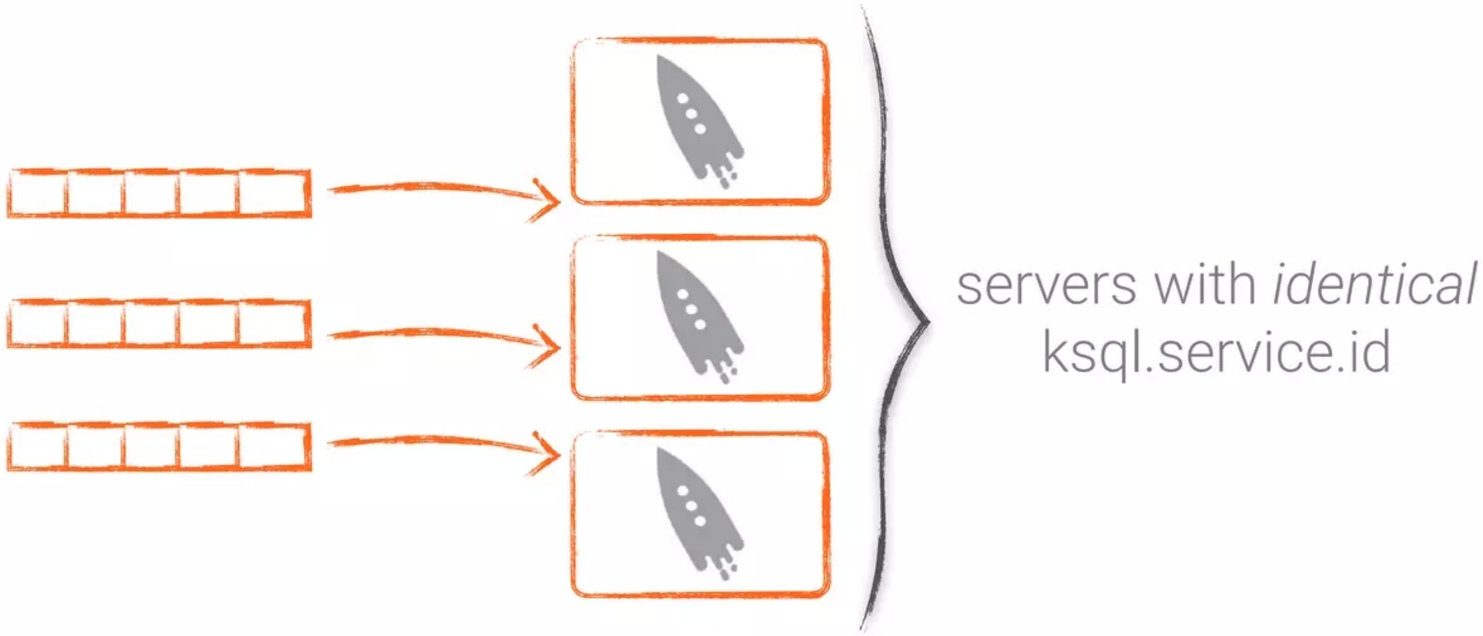
ksql.service.id

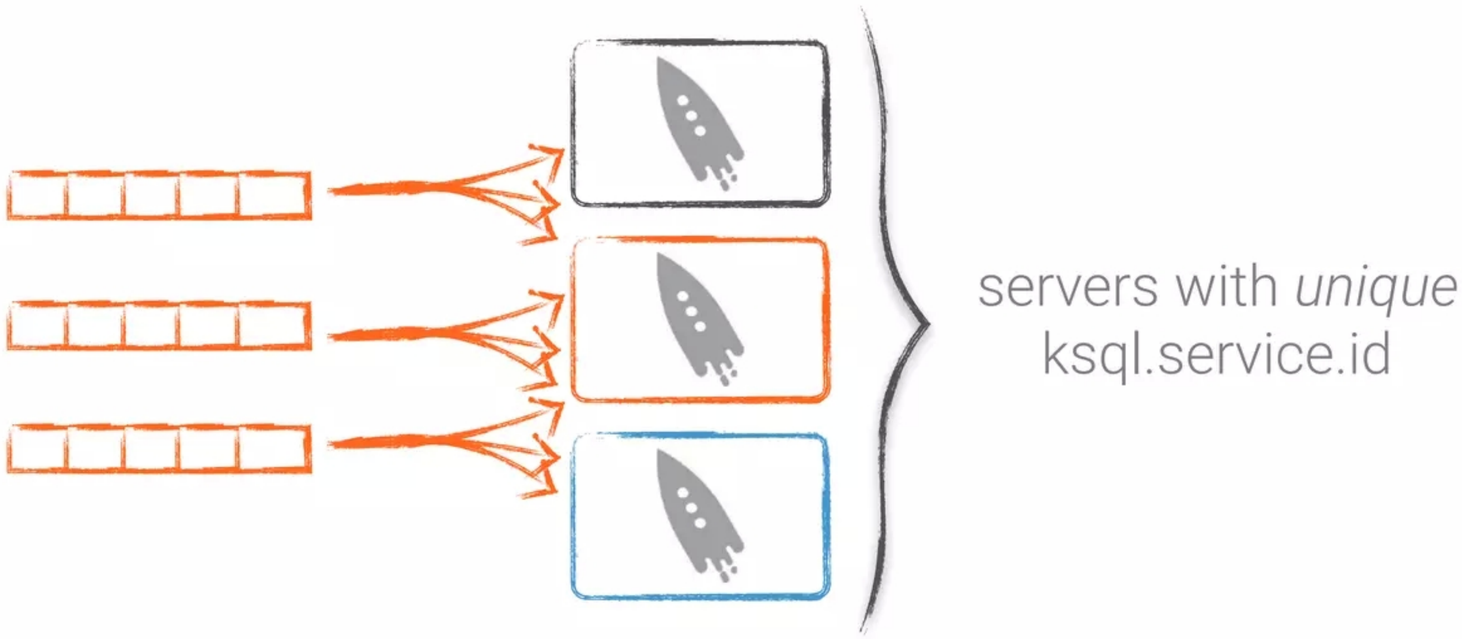




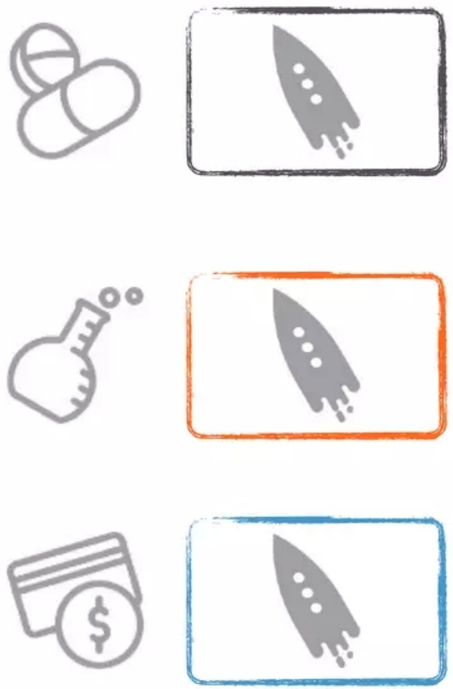
queries with *unique* application.id


queries with *identical*
application.id

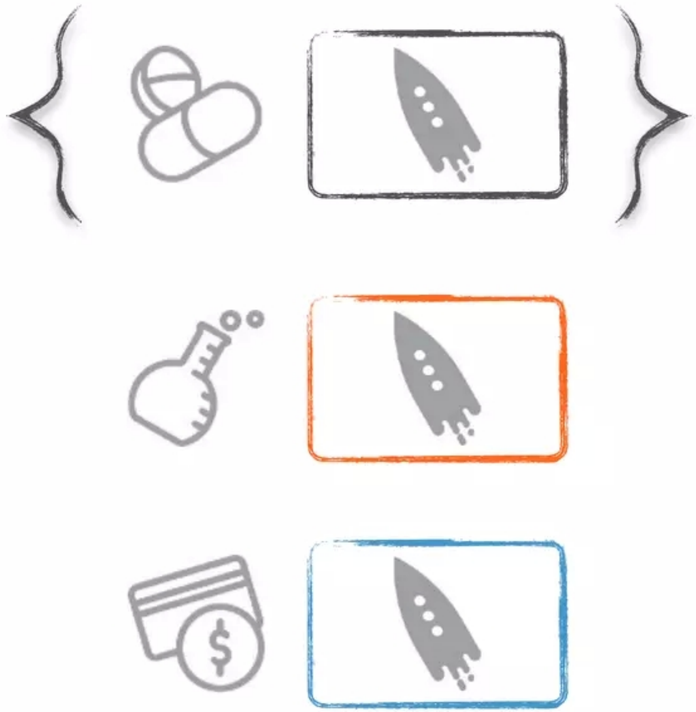

servers with *identical*
ksql.service.id


servers with *unique*
ksql.service.id

Recommended Deployment Topology

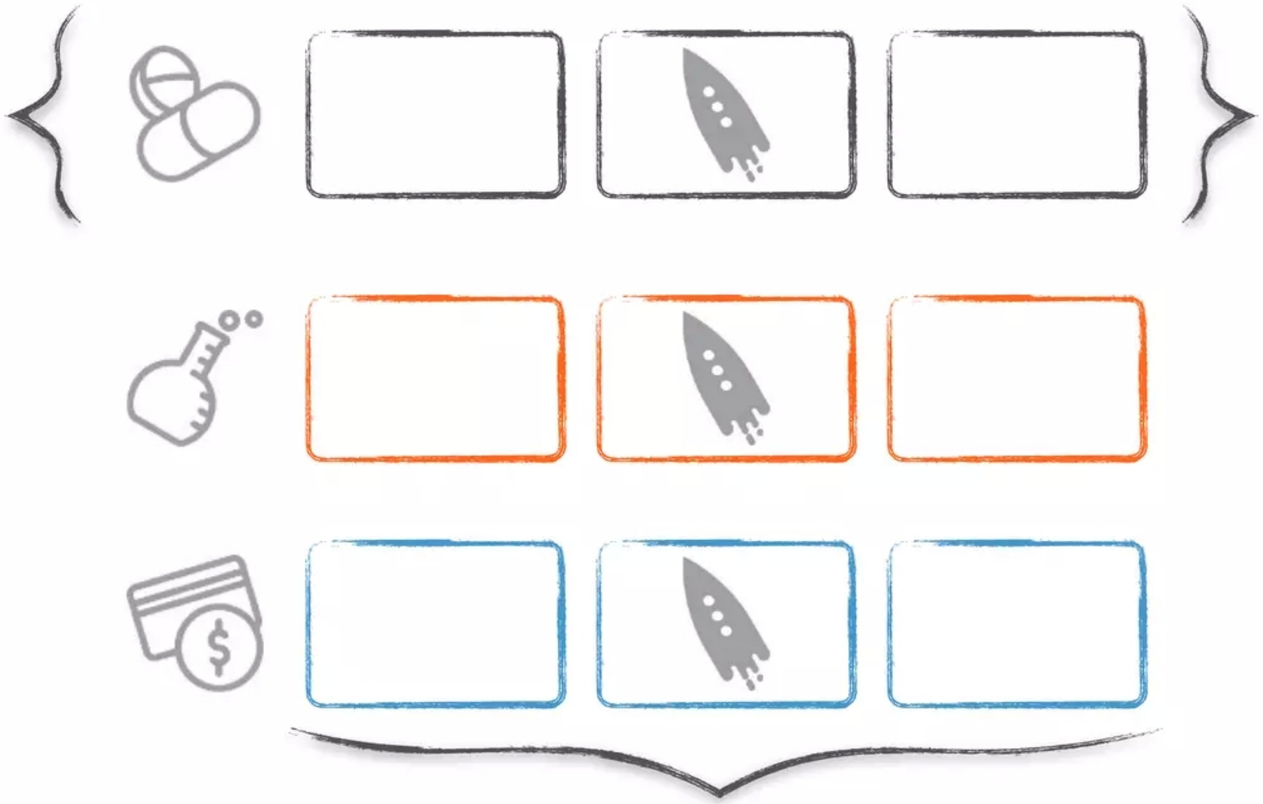


Recommended Deployment Topology



group queries by use-case using service.id

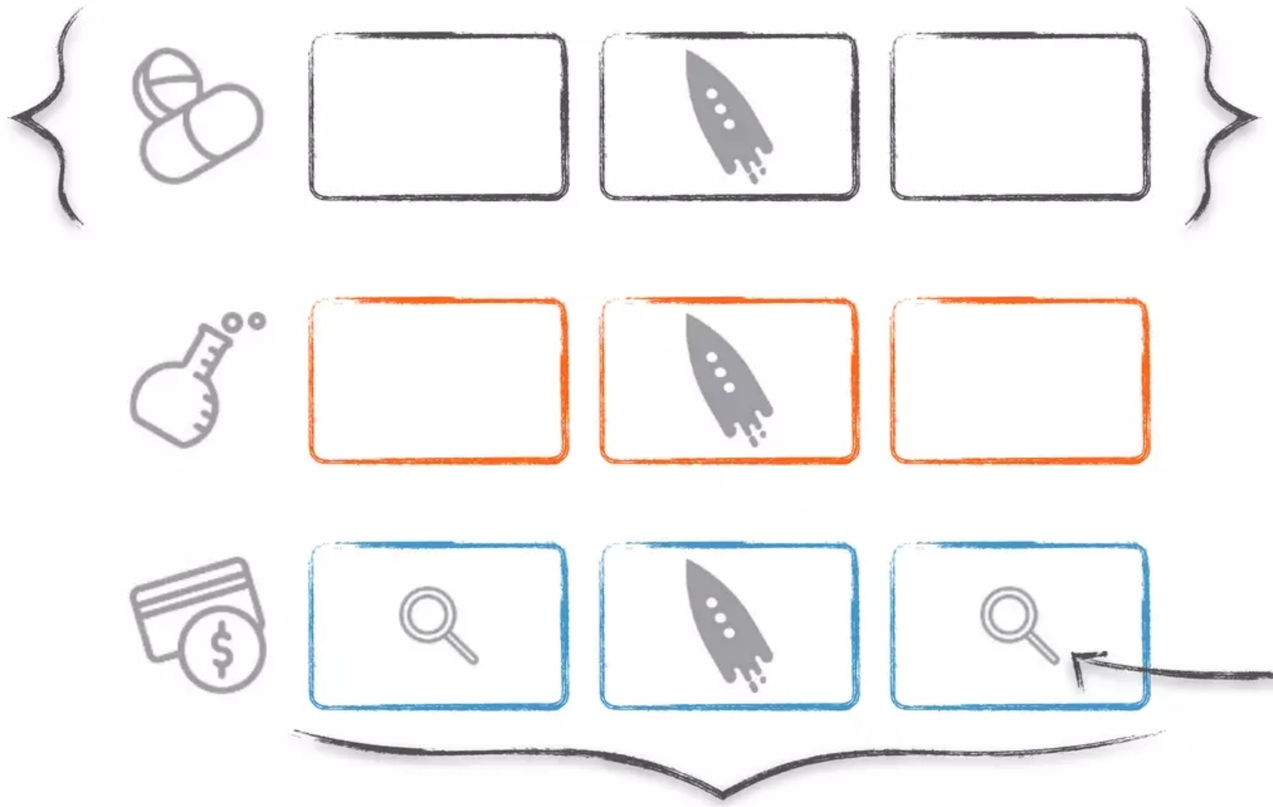
Recommended Deployment Topology



group queries by use-case using service.id

scale by increasing replicas

Recommended Deployment Topology

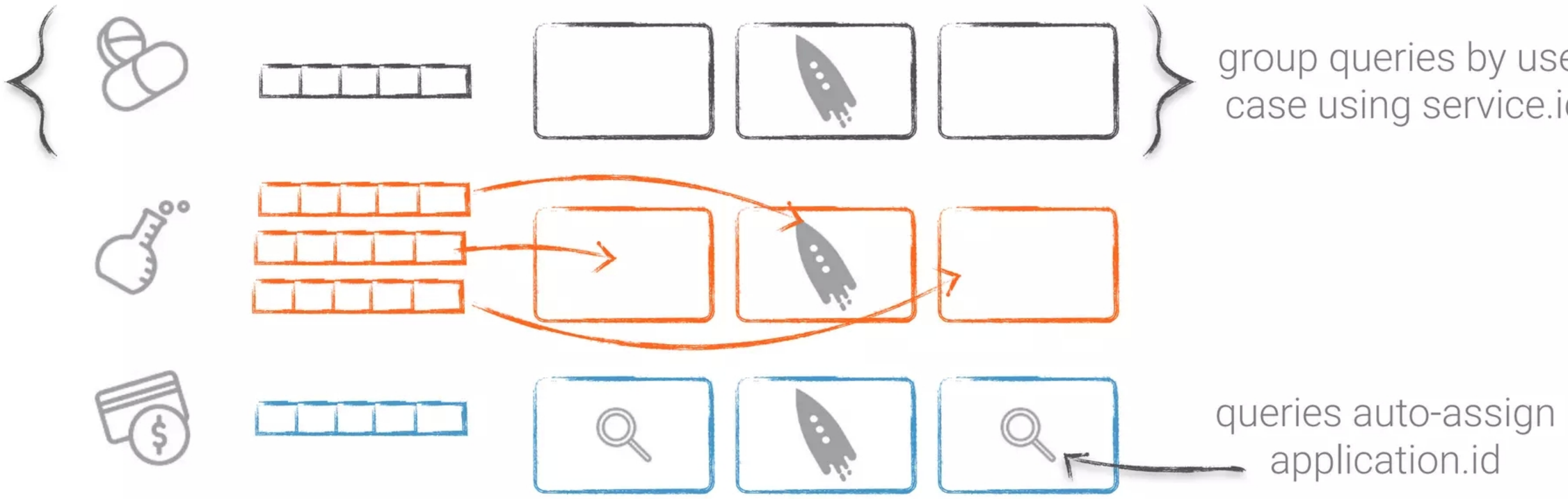


group queries by use-case using service.id

queries auto-assign application.id

scale by increasing replicas

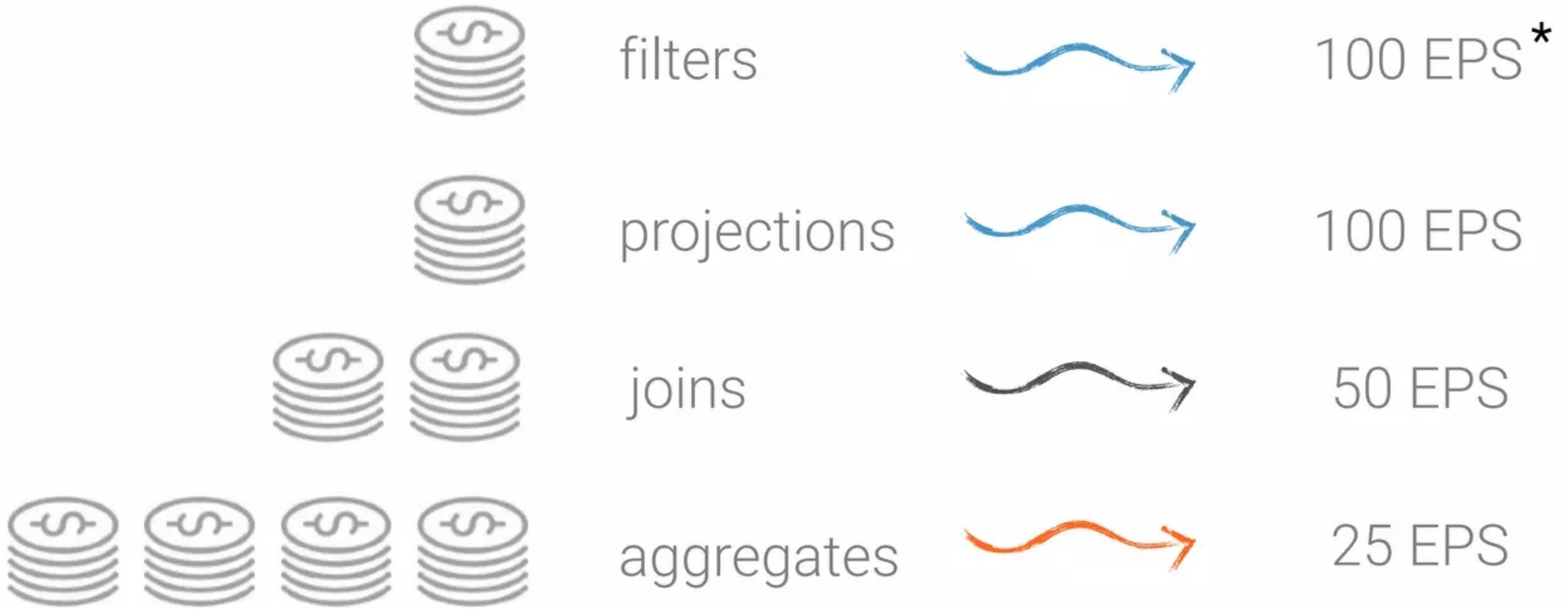
Recommended Deployment Topology



application.id = cluster-foo_123

application.id = { cluster-foo_123 | ksql.service.id + query_prefix + query_id }


filters
projections
joins
aggregates

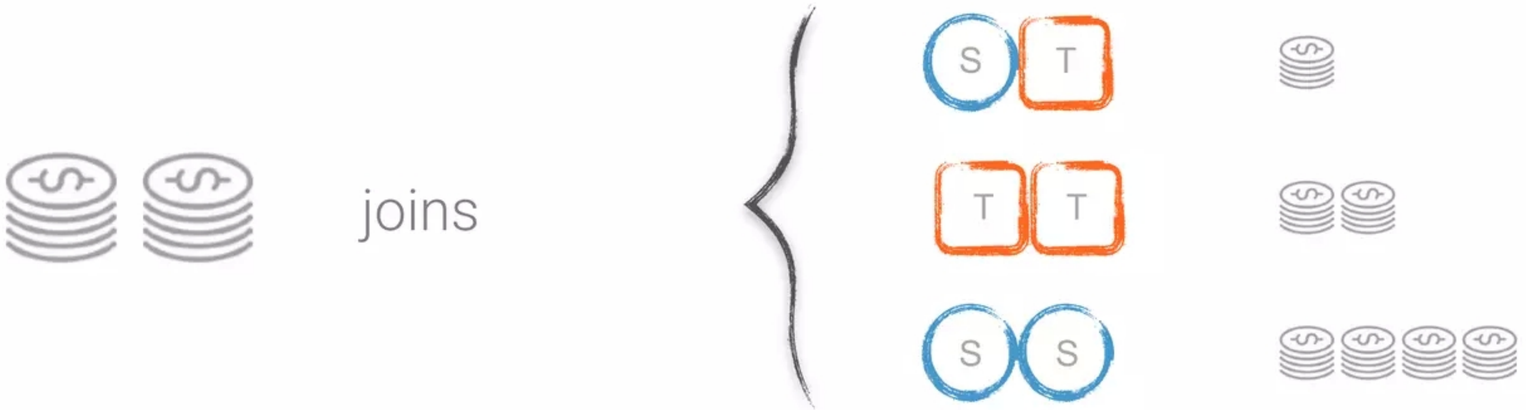


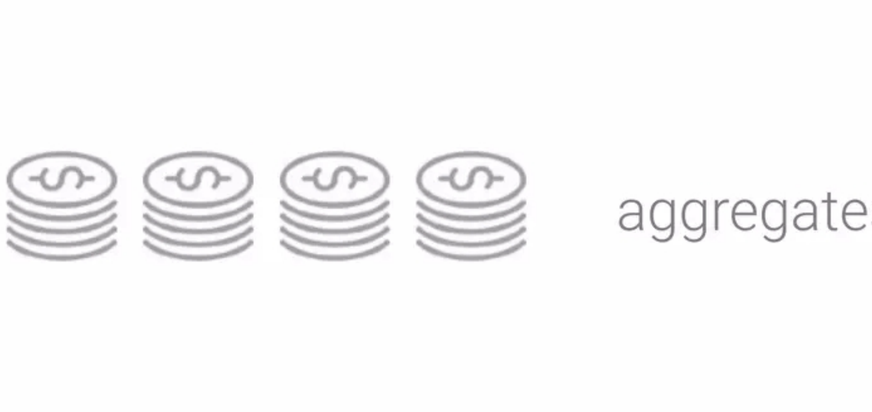
* assume that your max throughput is 100 EPS (it is likely you can support orders of magnitude more)

joins

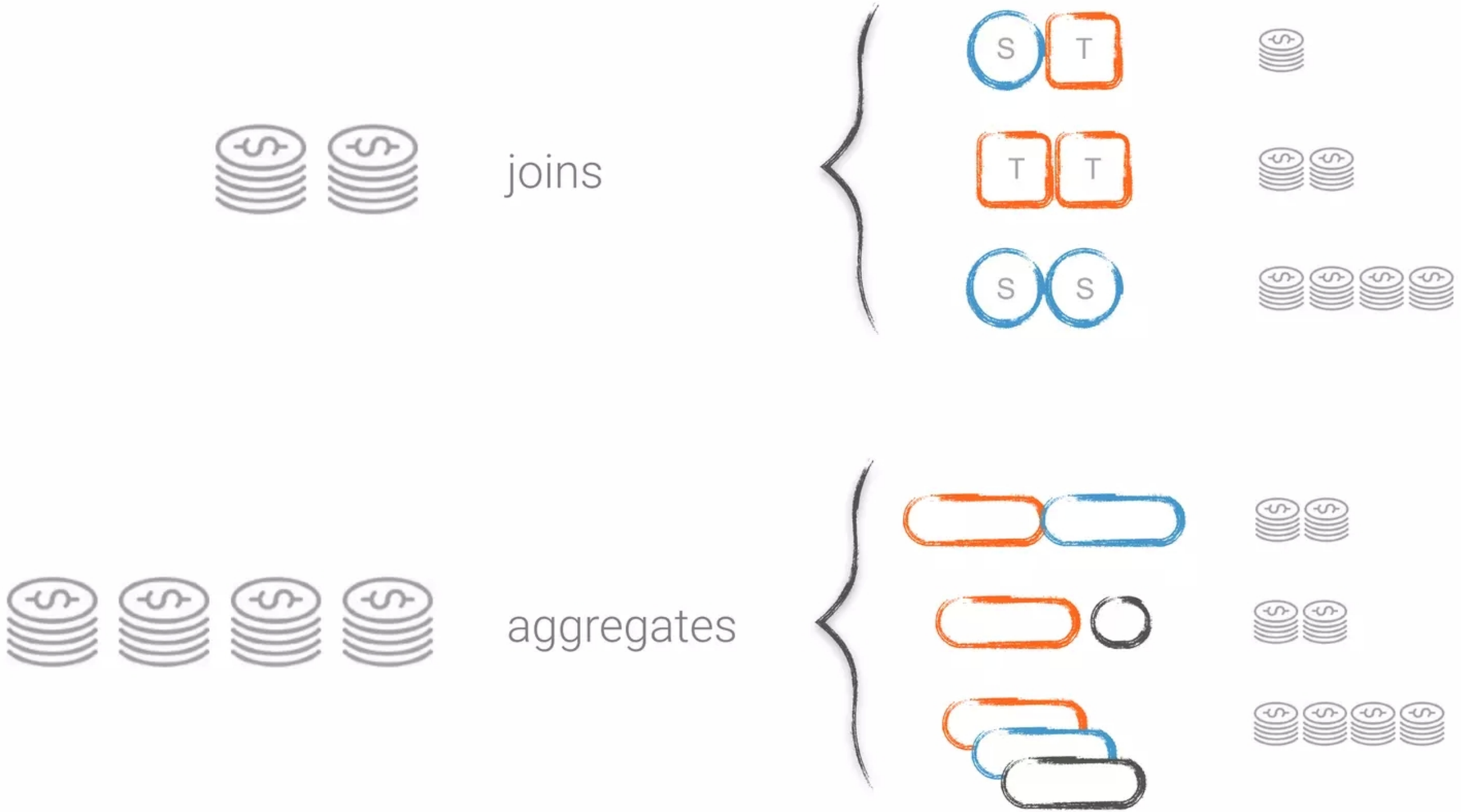
aggregates

see: https://docs.confluent.io/current/streams/developer-guide/dsl-api.html#stateful-transformations



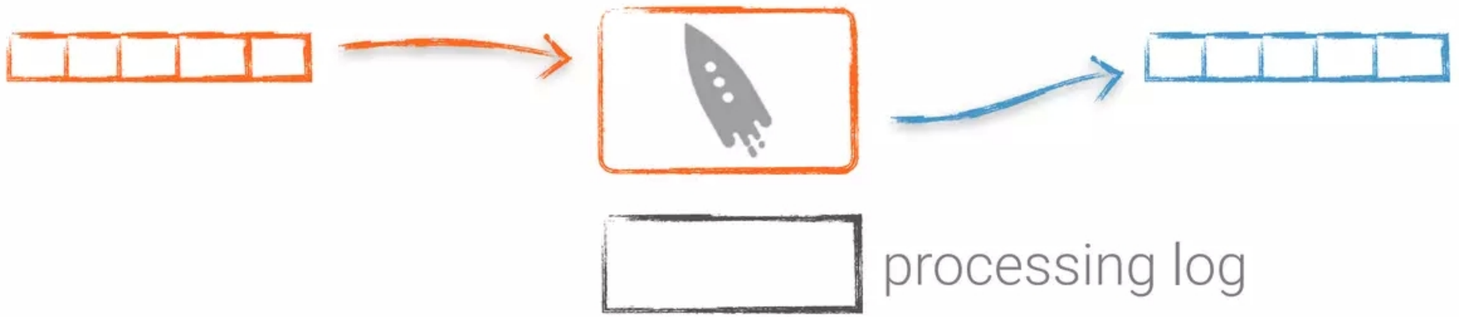


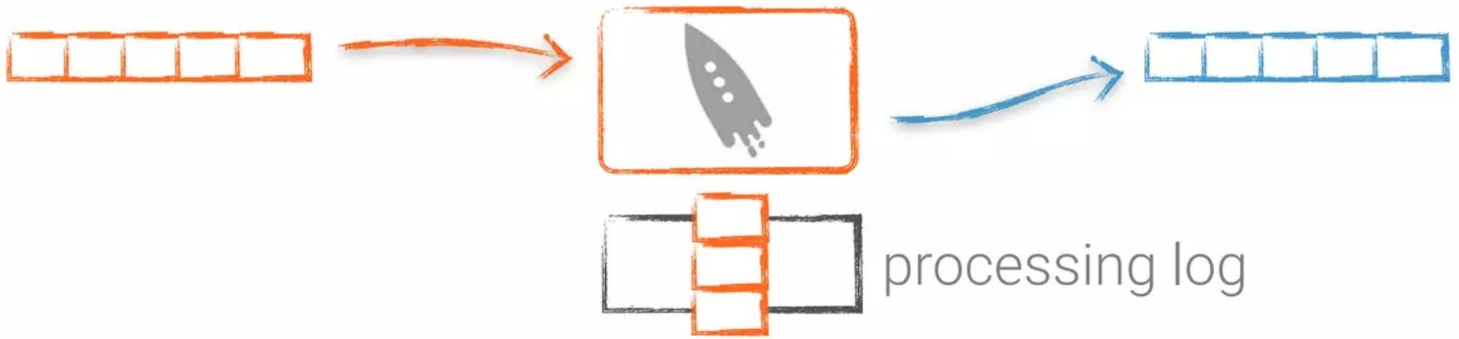
see: https://docs.confluent.io/current/streams/developer-guide/dsl-api.html#stateful-transformations

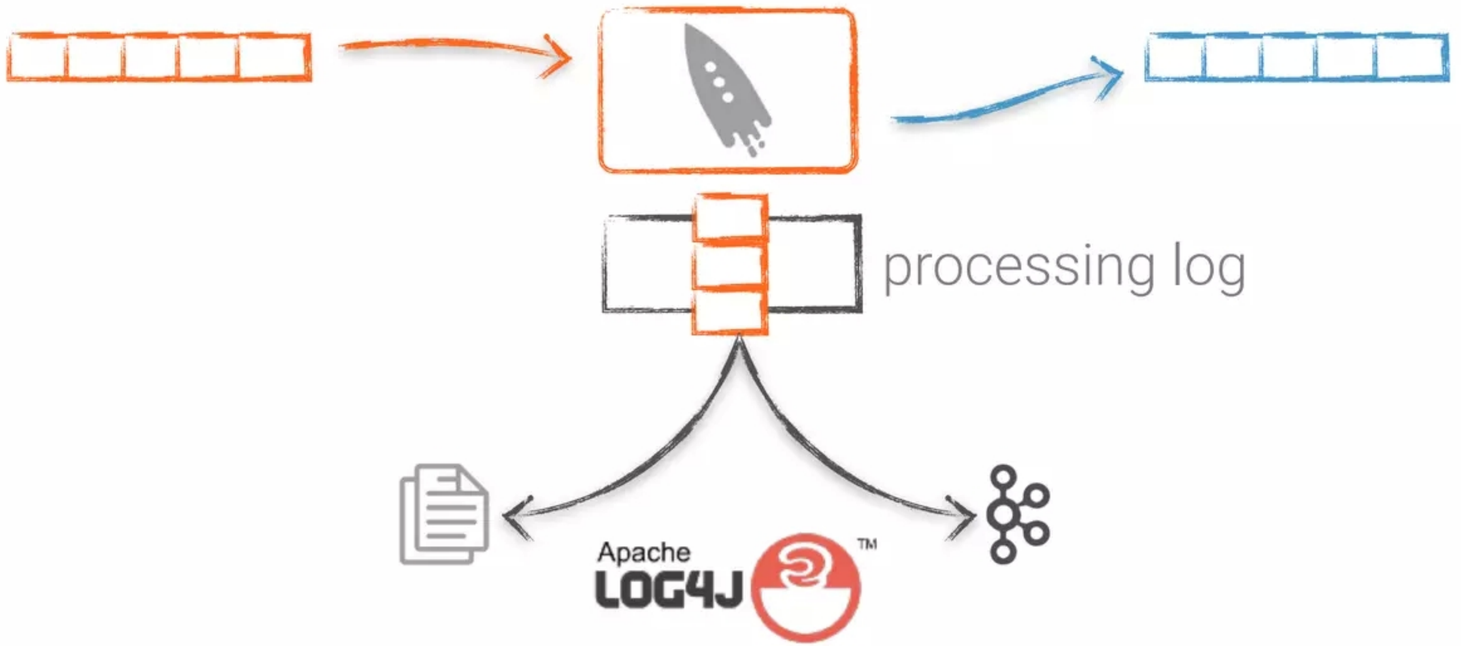


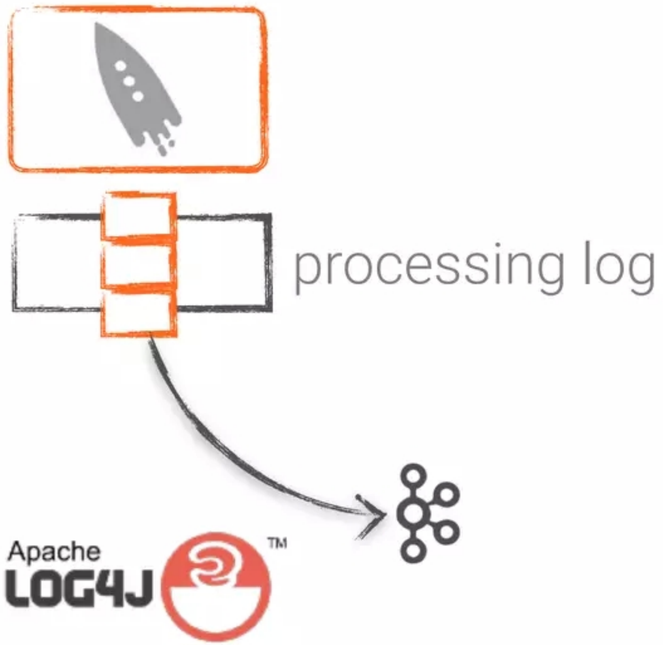
see: https://docs.confluent.io/current/streams/developer-guide/dsl-api.html#stateful-transformations

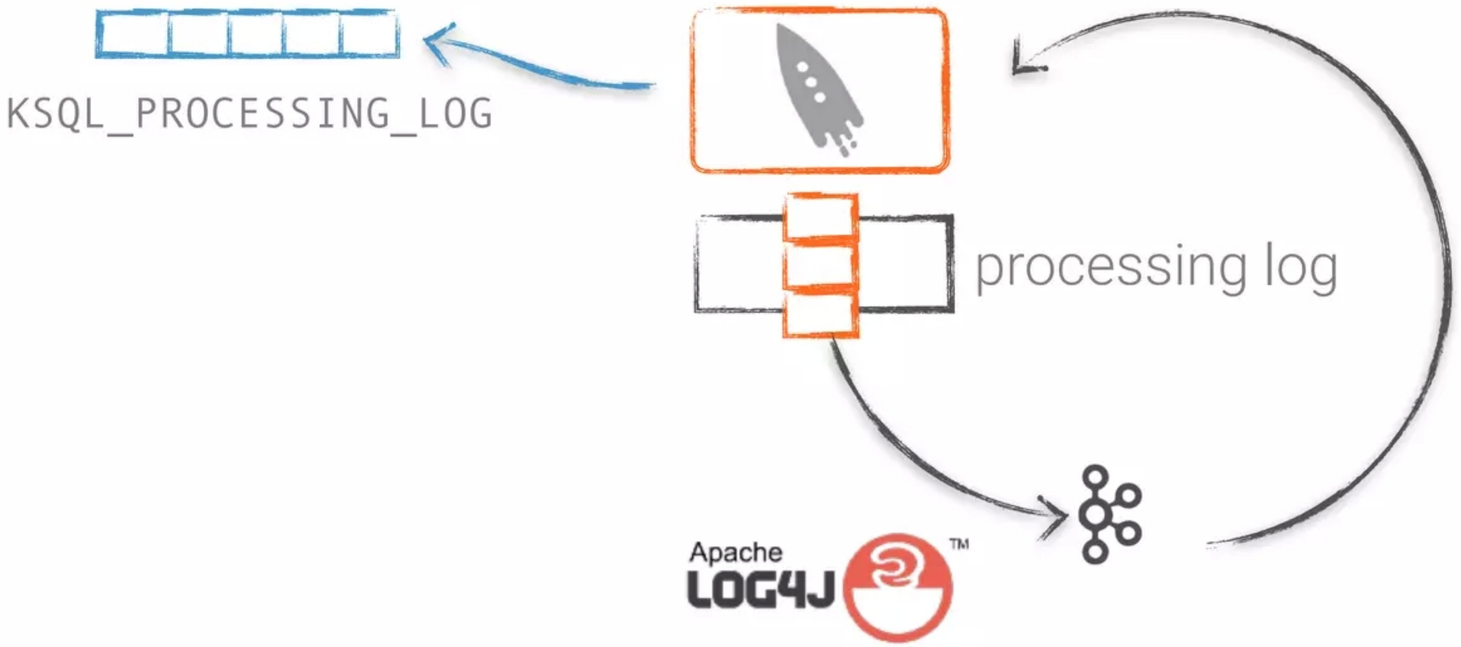
- Intro
- Develop
- Deploy
- **Operate**
- Common Mistakes
- Q&A

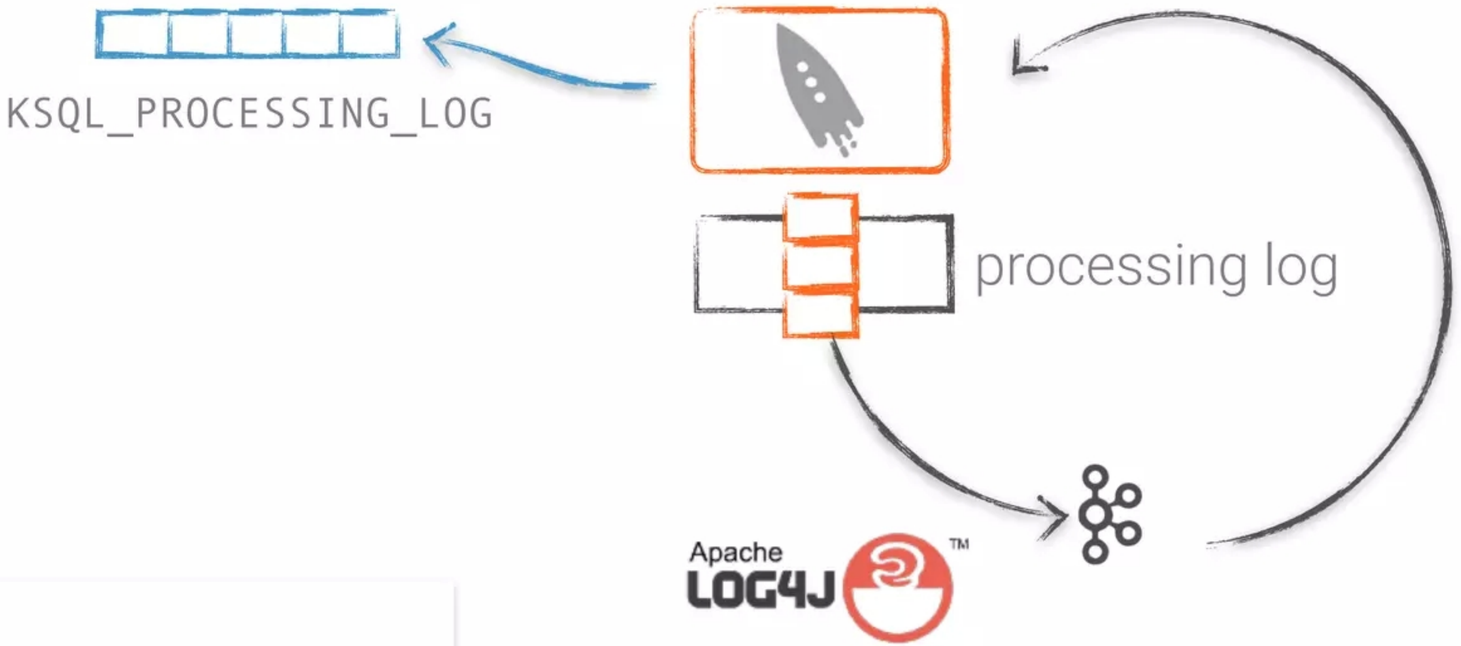

processing log


processing log

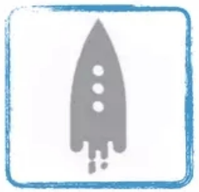

processing log
Apache
LOG4J
™

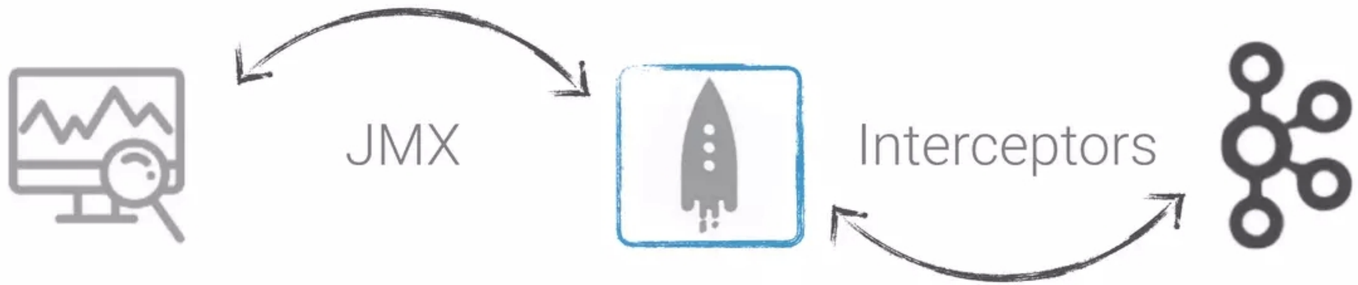

processing log
Apache
LOG4J
™


KSQL_PROCESSING_LOG
processing log
Apache
LOG4J



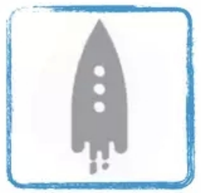
✓ **tip:** the best way to debug streaming is by streaming

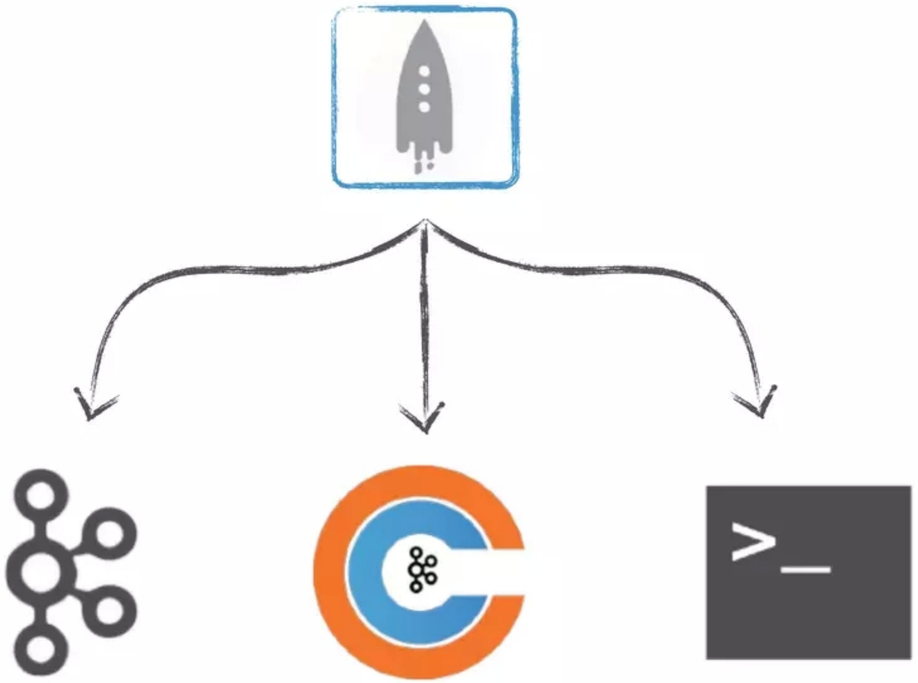


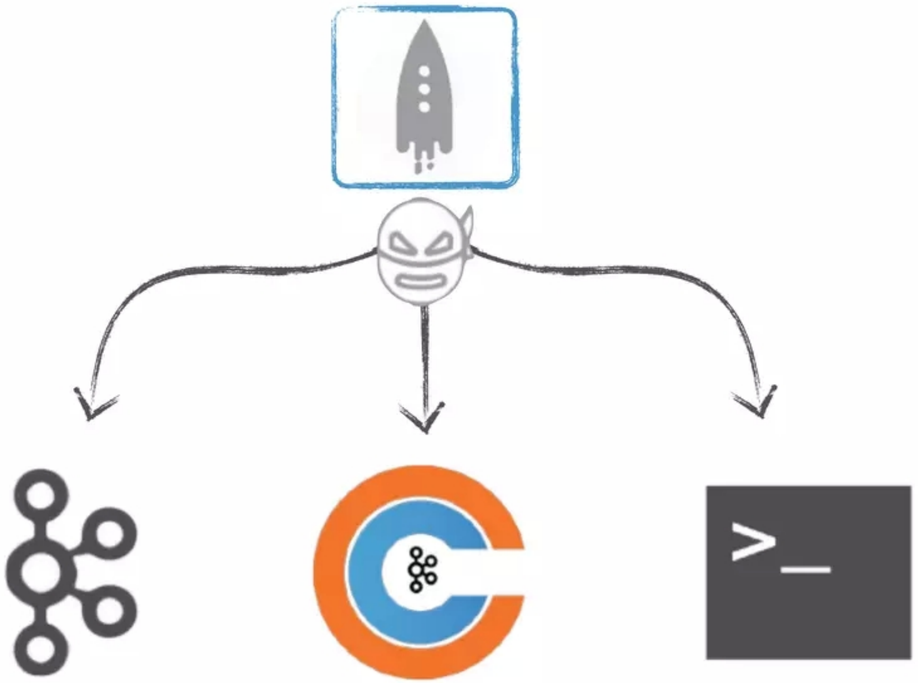

JMX
Interceptors

```
error-rate: 0.0
num-persistent-queries: 2.0
num-active-queries: 2.0
messages-consumed-per-sec: 193024.78294586178
messages-produced-per-sec: 193025.4730374501
messages-consumed-max: 103397.81191436431
```

JMX

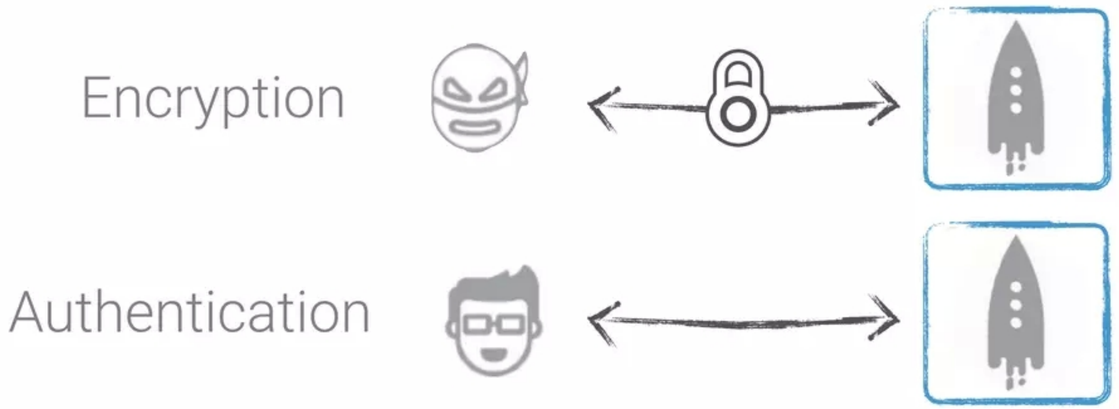




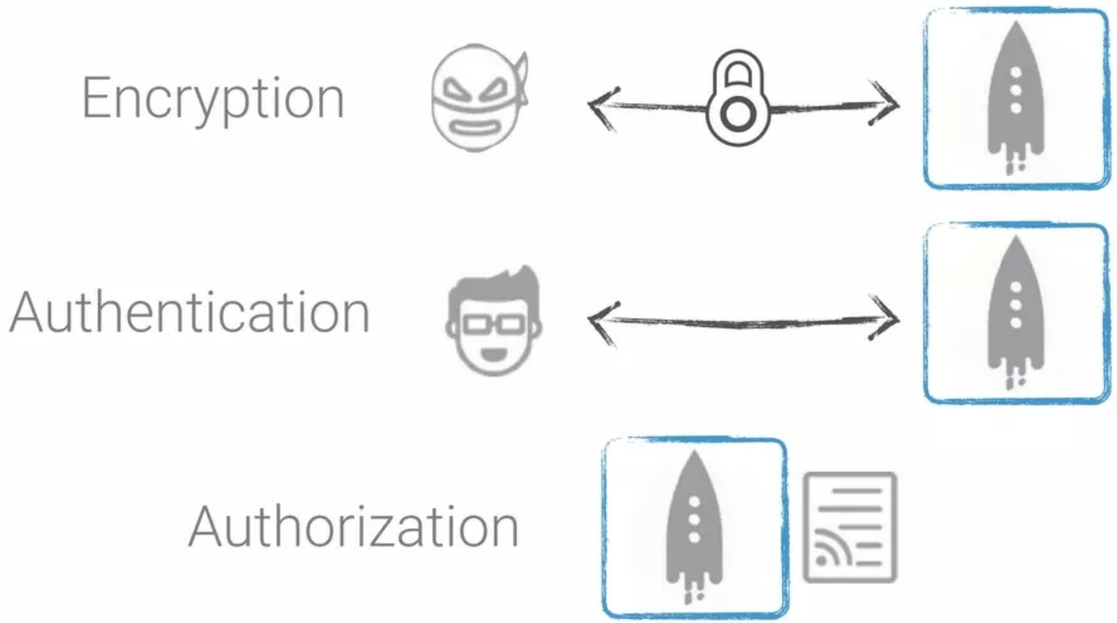


Encryption

see: https://www.confluent.io/kafka-summit-ny19/ksql-and-security



see: https://www.confluent.io/kafka-summit-ny19/ksql-and-security



see: https://www.confluent.io/kafka-summit-ny19/ksql-and-security

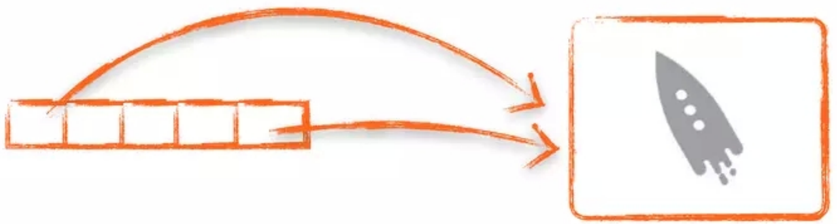
- Intro
- Develop
- Deploy
- Operate
- **Common Mistakes**
- Q&A

Queries Are Continuous

Queries Are Continuous

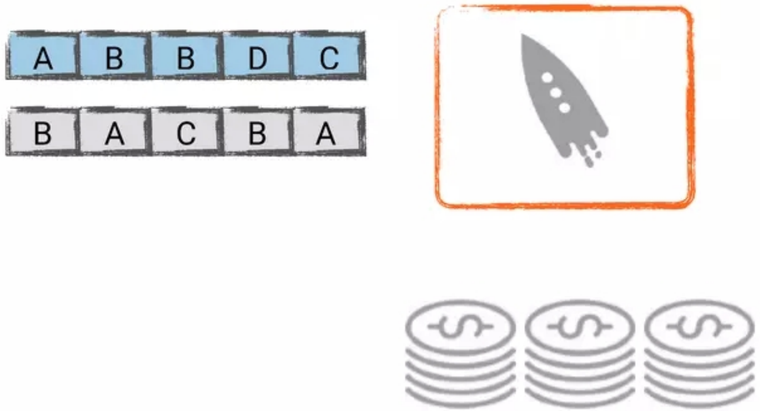
new queries read from most recent by default

new queries read from most recent by default



```
SET 'auto.offset.reset' = 'earliest'
```

joins may cause repartition



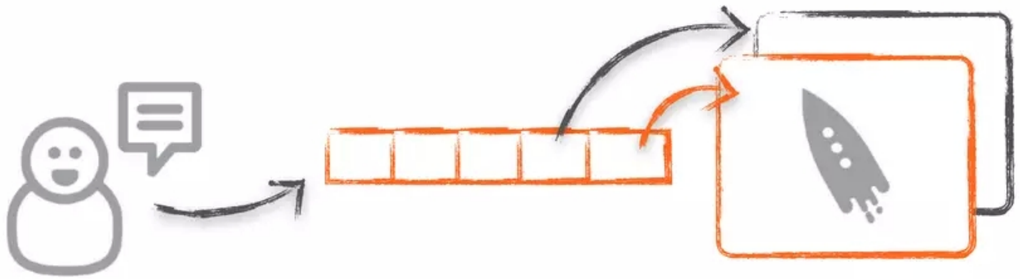
joins may cause
repartition

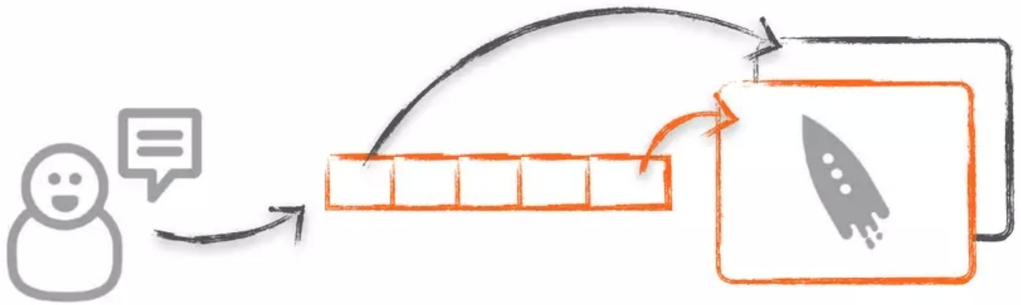
repartition does not preserve order across partitions…

repartition does not preserve order across partitions…

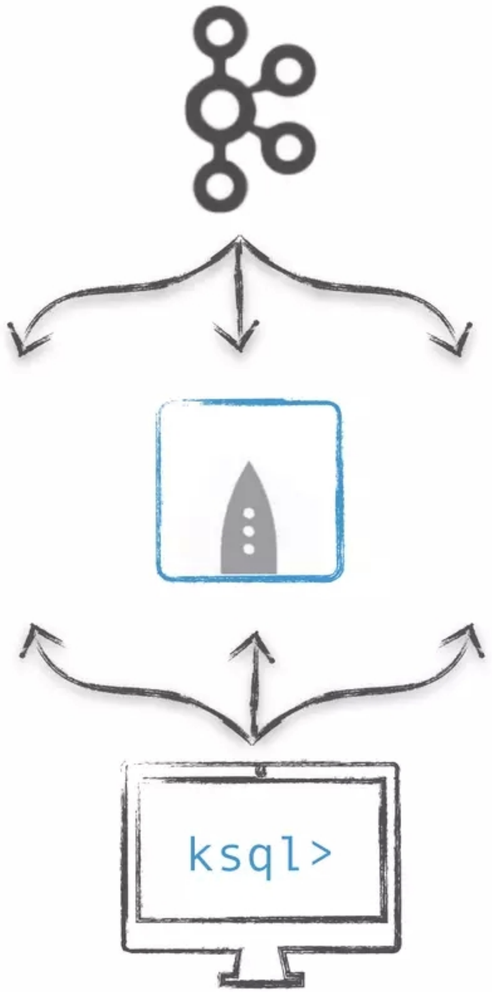
repartition does not preserve order across partitions...

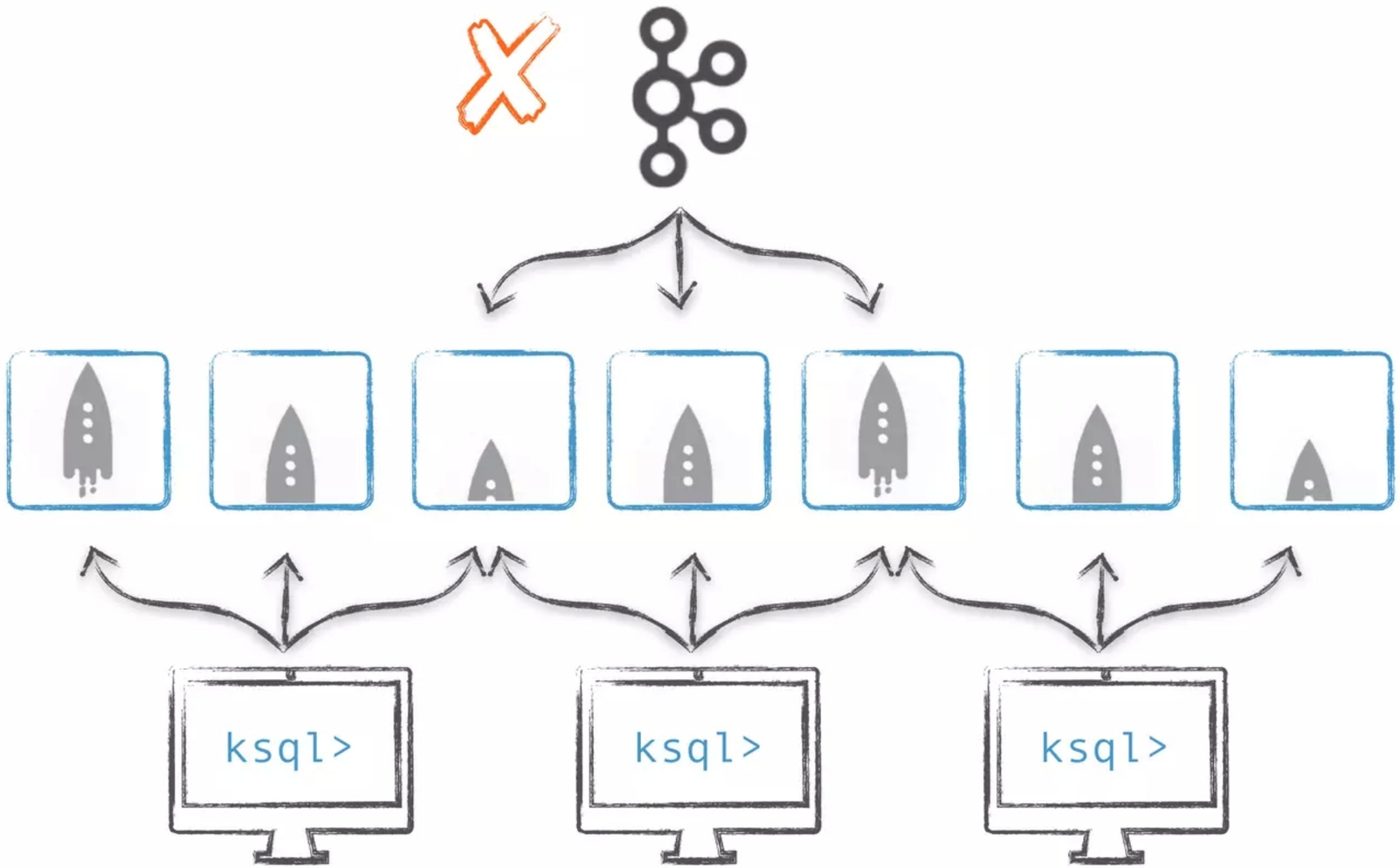
so tables cannot be rekeyed

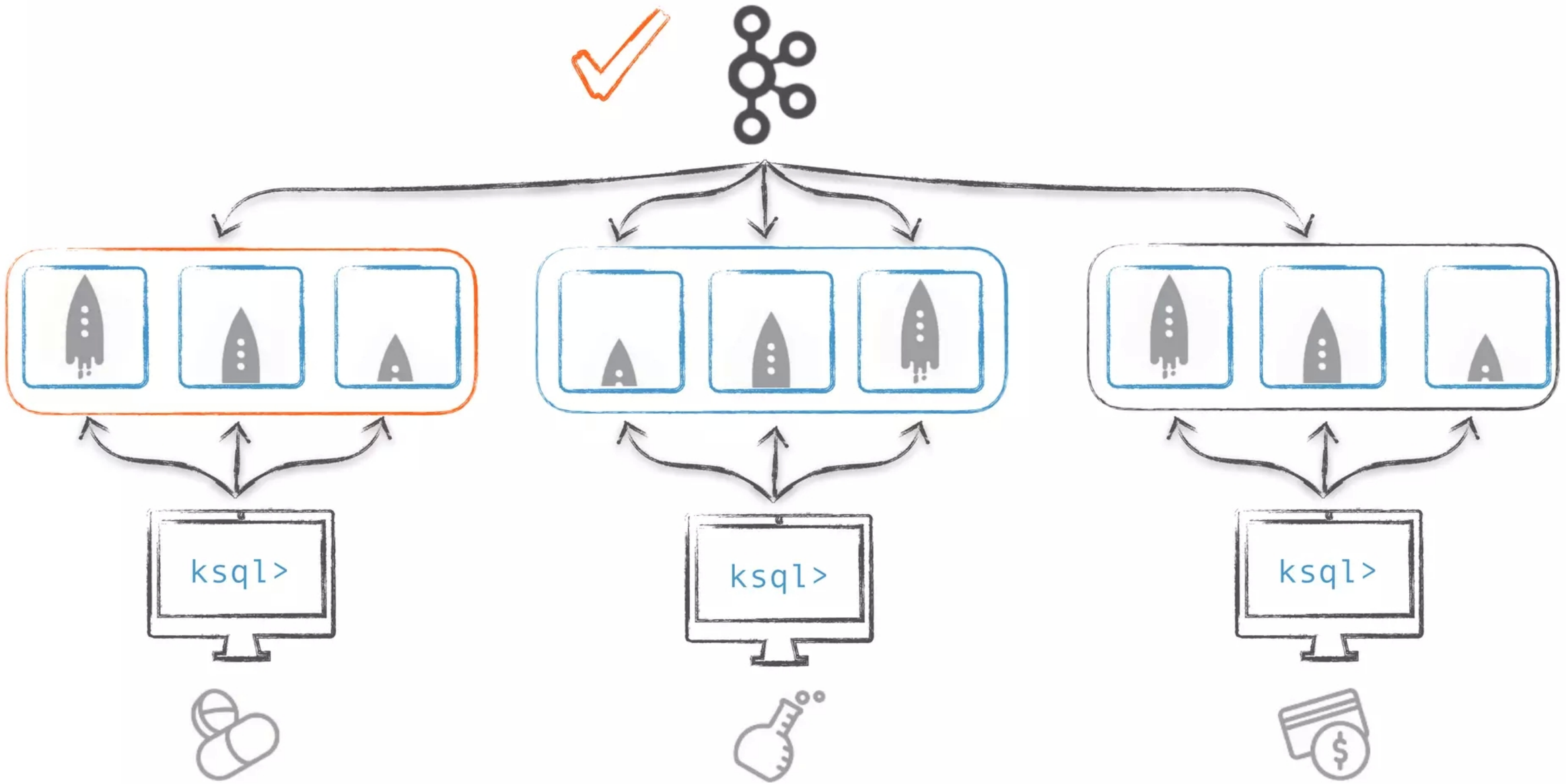




crashes replay (live query) state


ksql>


ksql>
ksql>
ksql>


ksql>
ksql>
ksql>

- Intro
- Develop
- Deploy
- Operate
- Common Mistakes
- **Q&A**

in/agavra almog@confluent.io