



# Kafka Ecosystem in 75 Minutes

Daniel Hinojosa

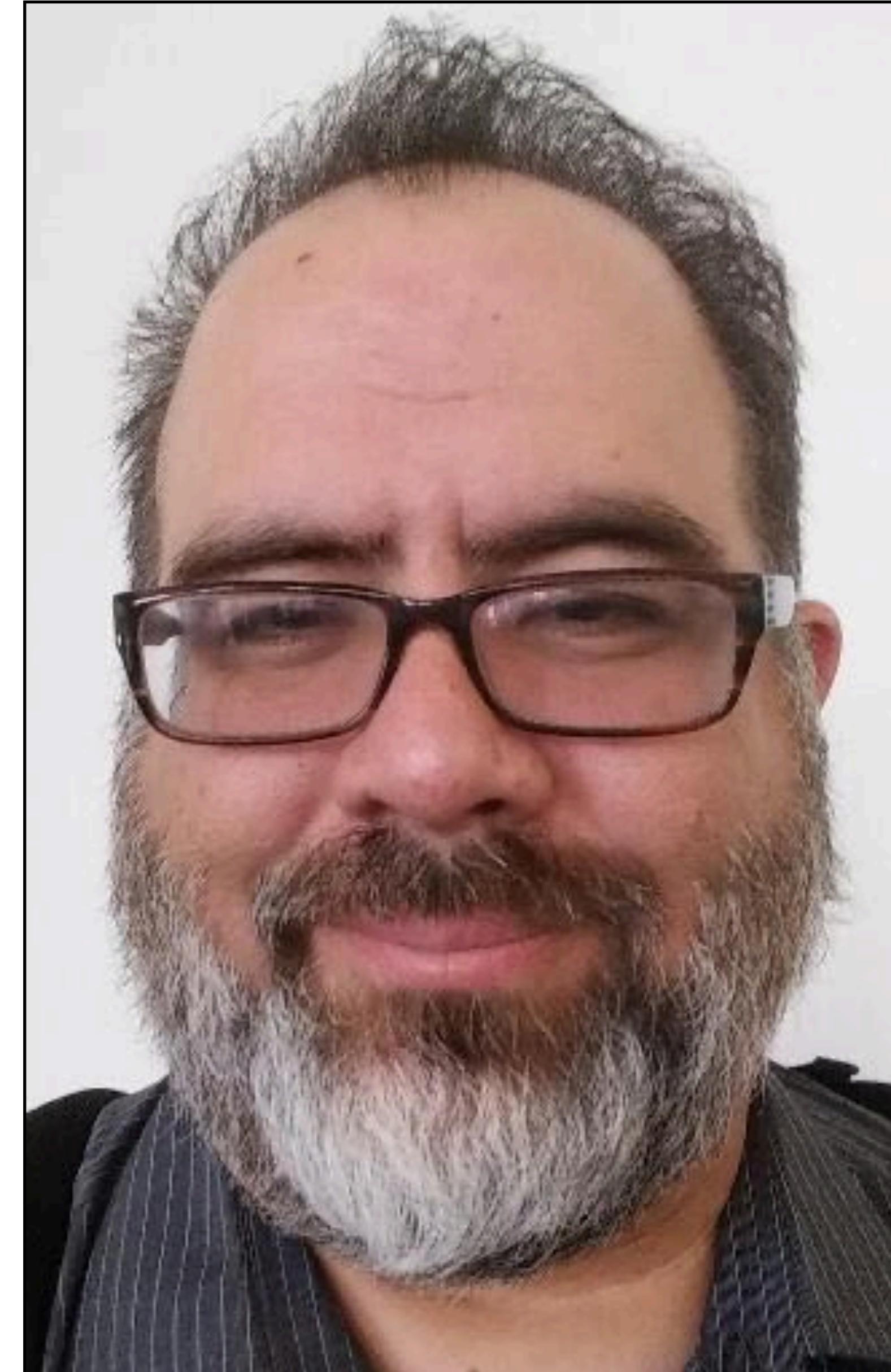
# About Me...

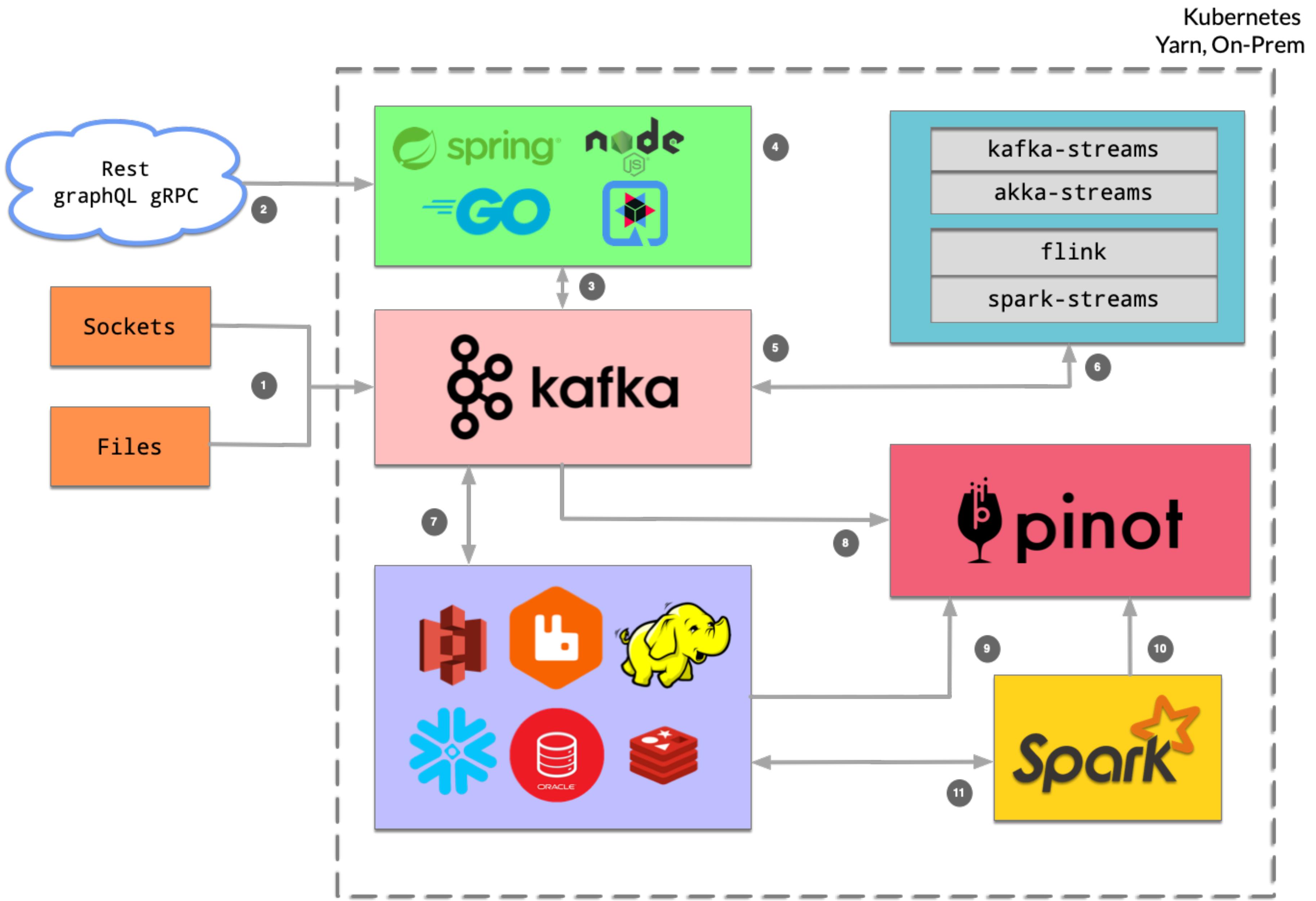
**Daniel Hinojosa**  
**Programmer, Consultant,**  
**Trainer**

[dhinojosa@evolutionnext.com](mailto:dhinojosa@evolutionnext.com)  
[@dhinojosa](https://twitter.com/dhinojosa)

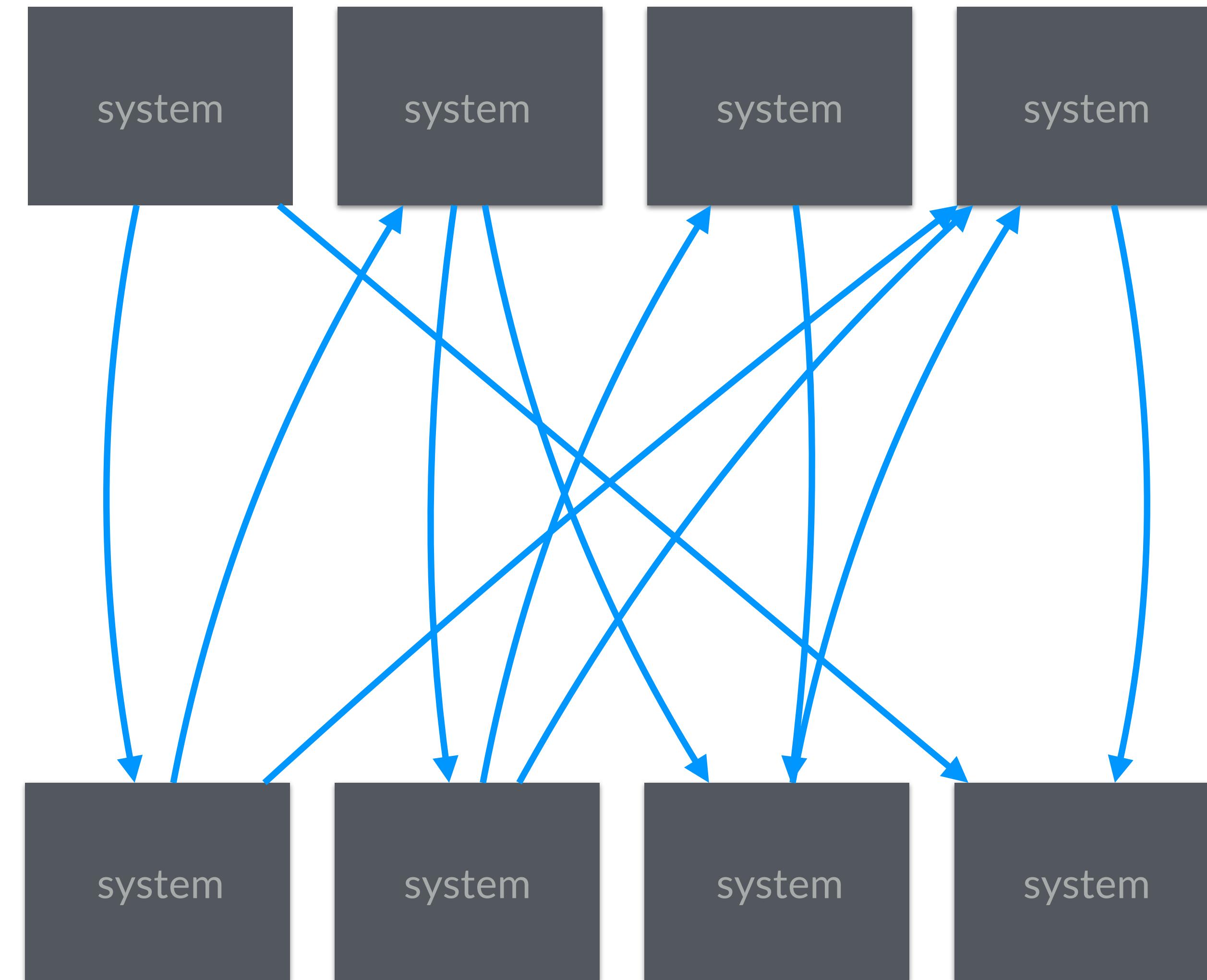
[https://mastodon.social/  
@dhinojosa](https://mastodon.social/@dhinojosa)

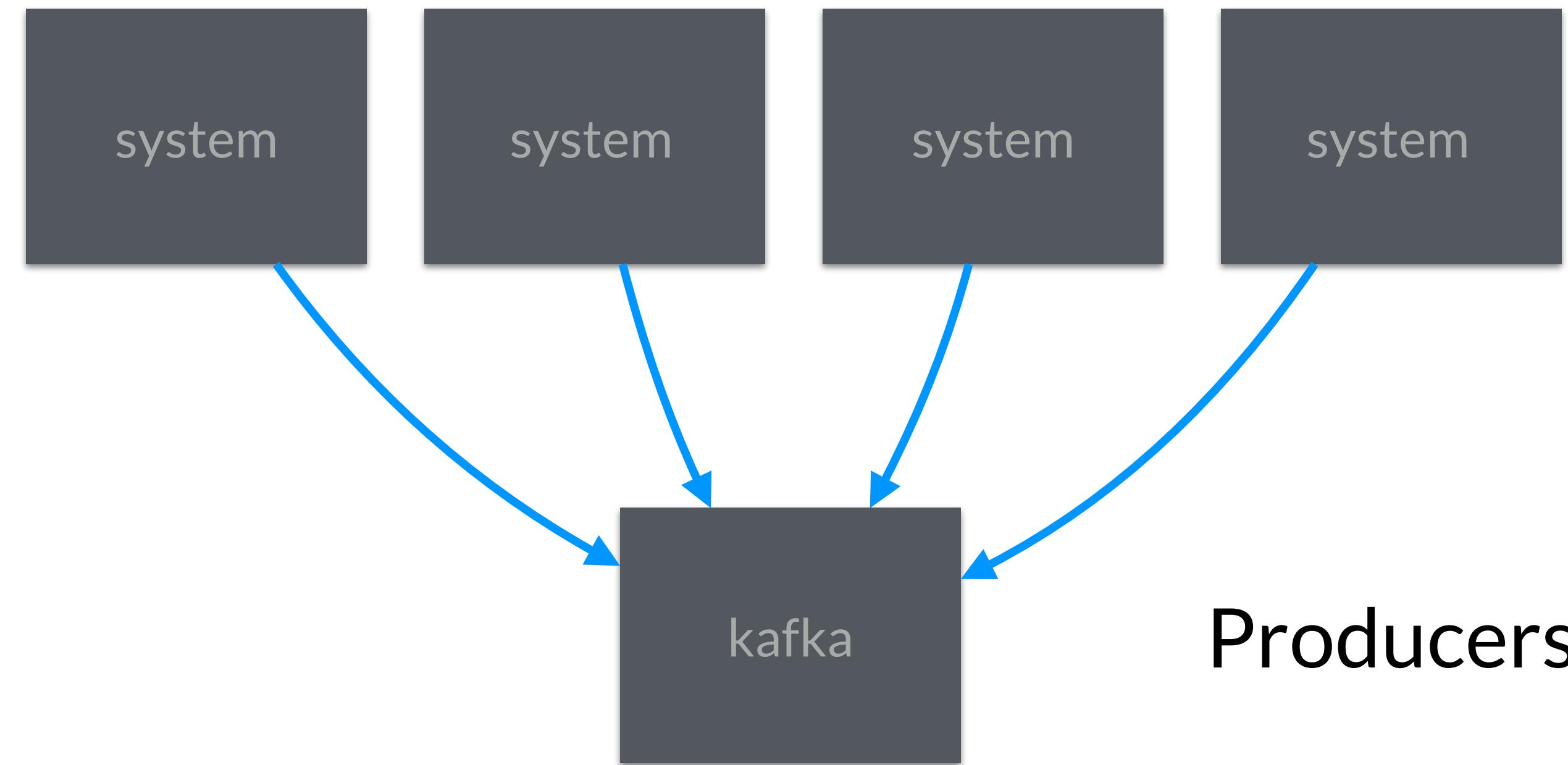
[https://www.linkedin.com/in/  
dhevolutionnext/](https://www.linkedin.com/in/dhevolutionnext/)

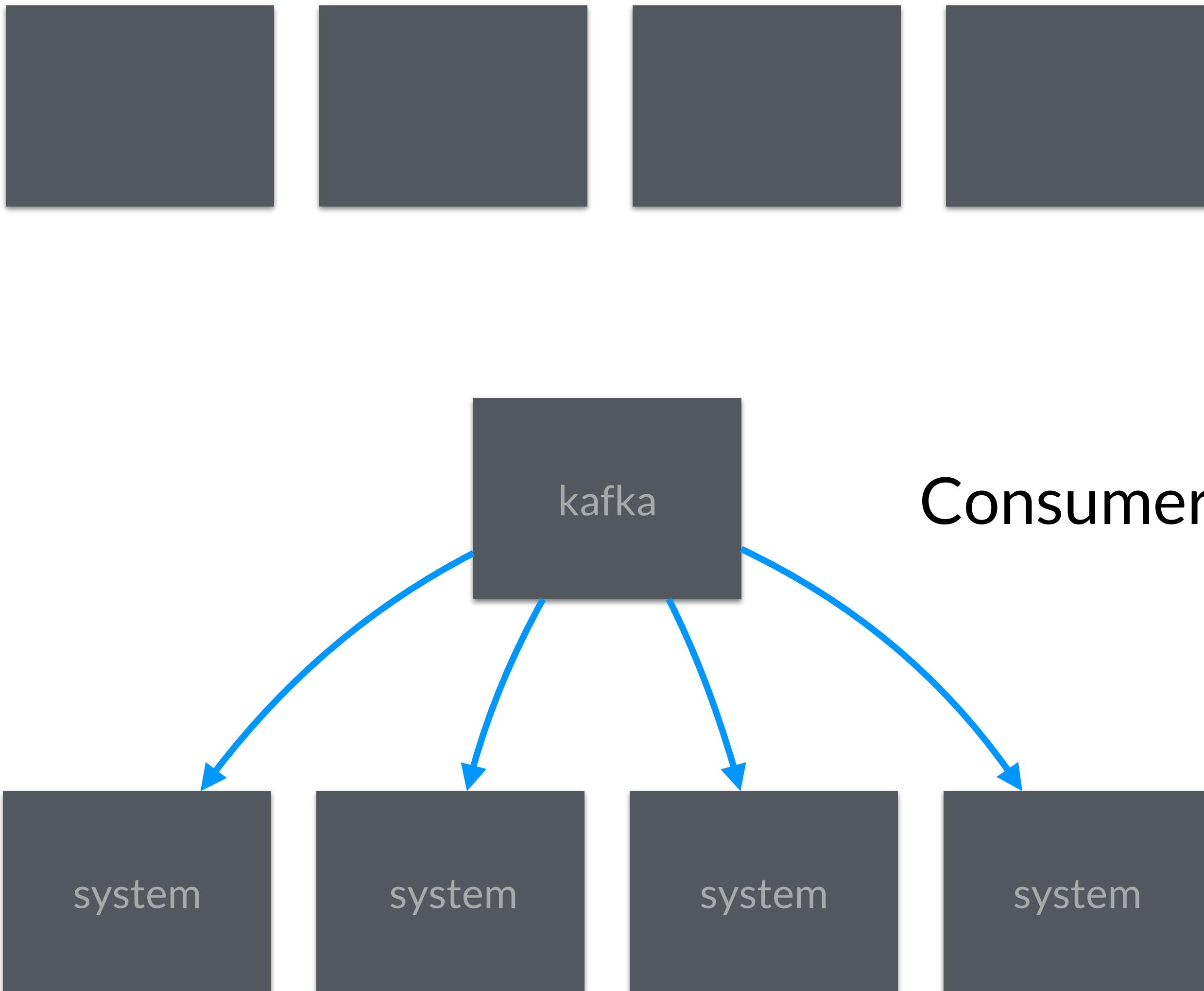


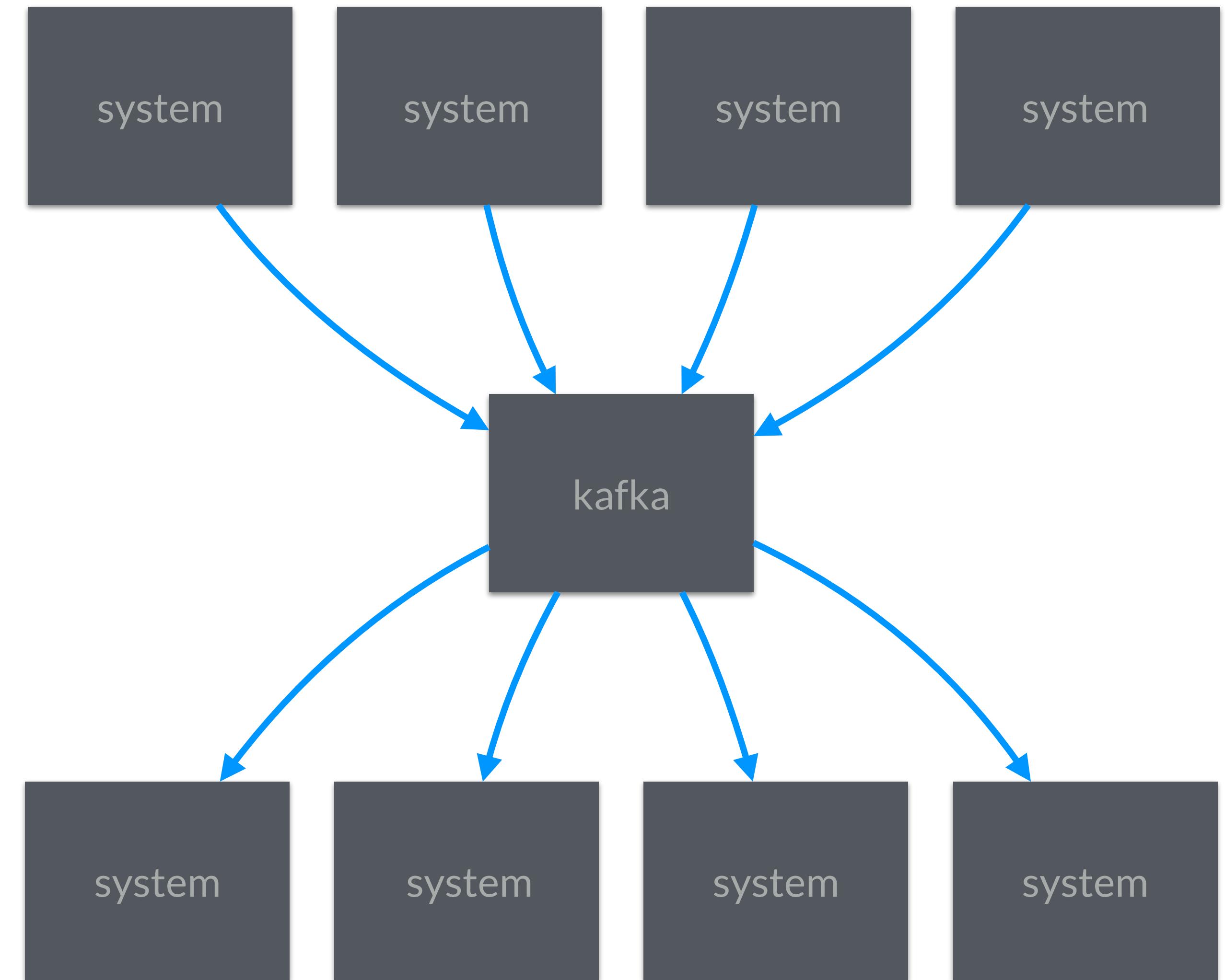


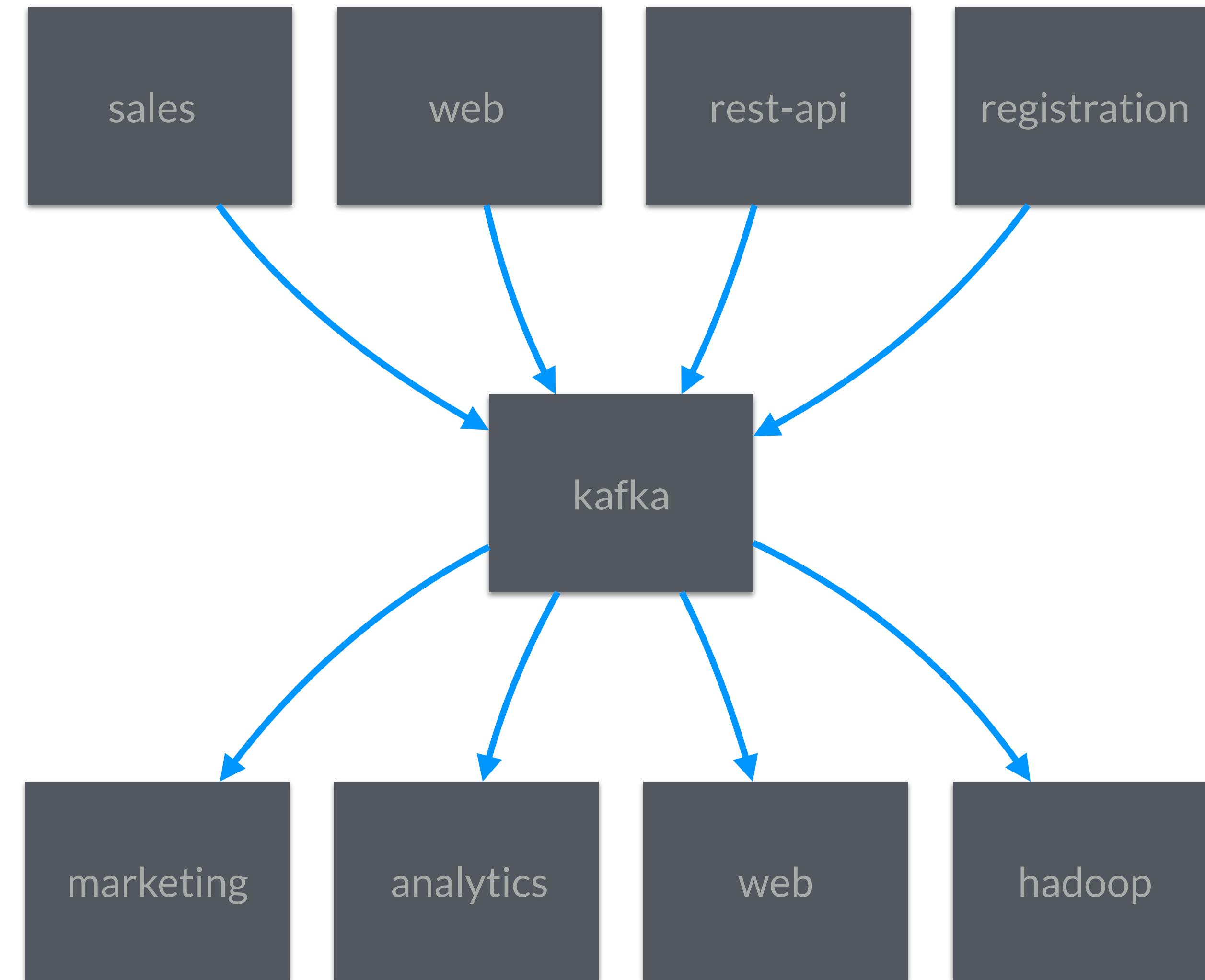
# Kafka Introduction & Review











Oversimplified: A Producer can be a Consumer

# About Kafka

Handles millions of messages per second: high throughput, high volume

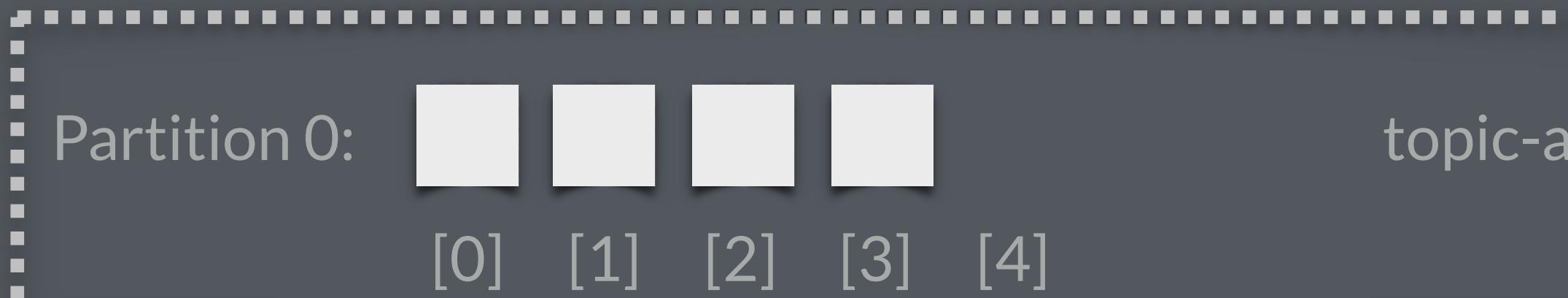
Distributed and Replicated Commit-log

Real Time Data Processing

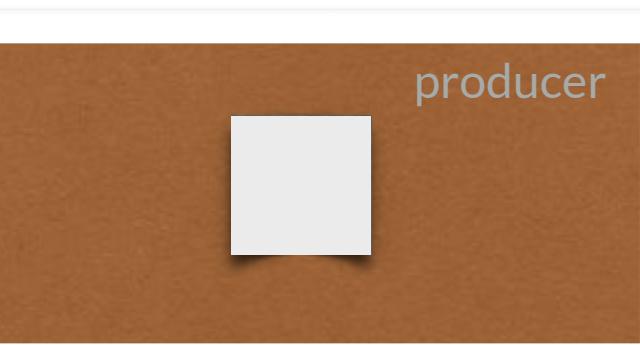
Stream processing

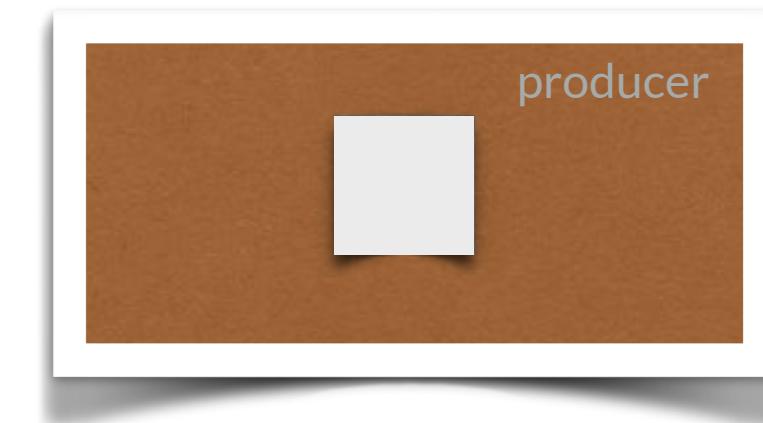
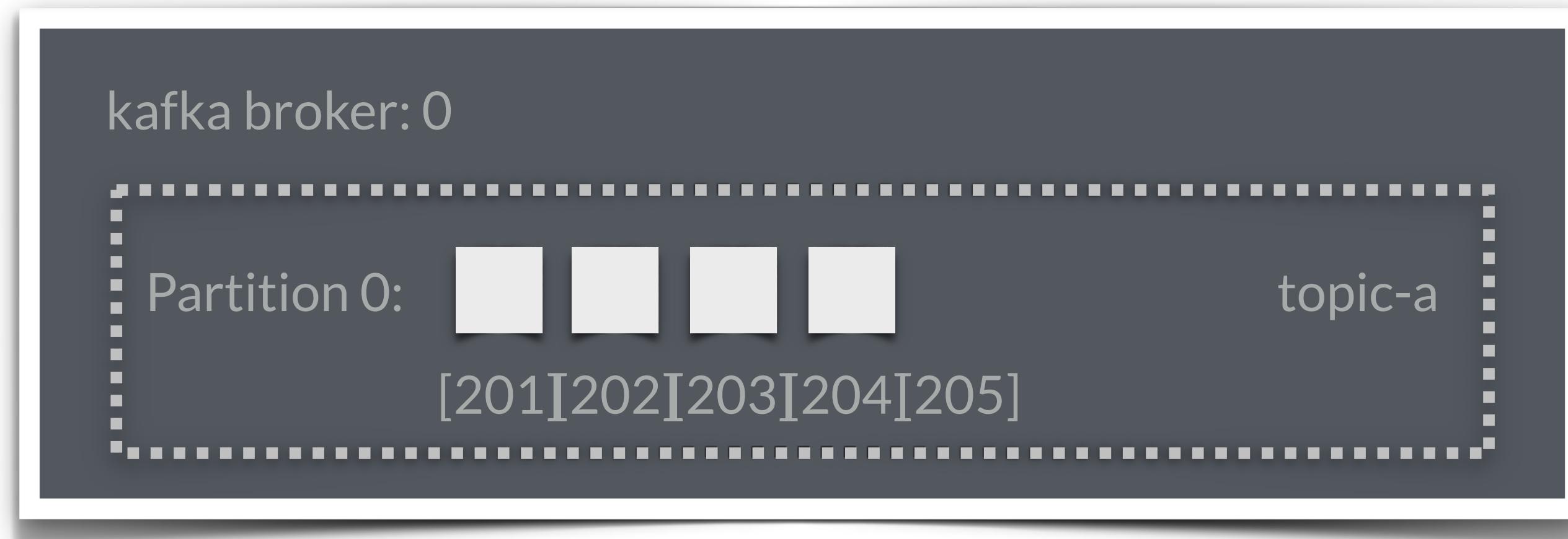
How messages are  
produced

kafka broker: 0



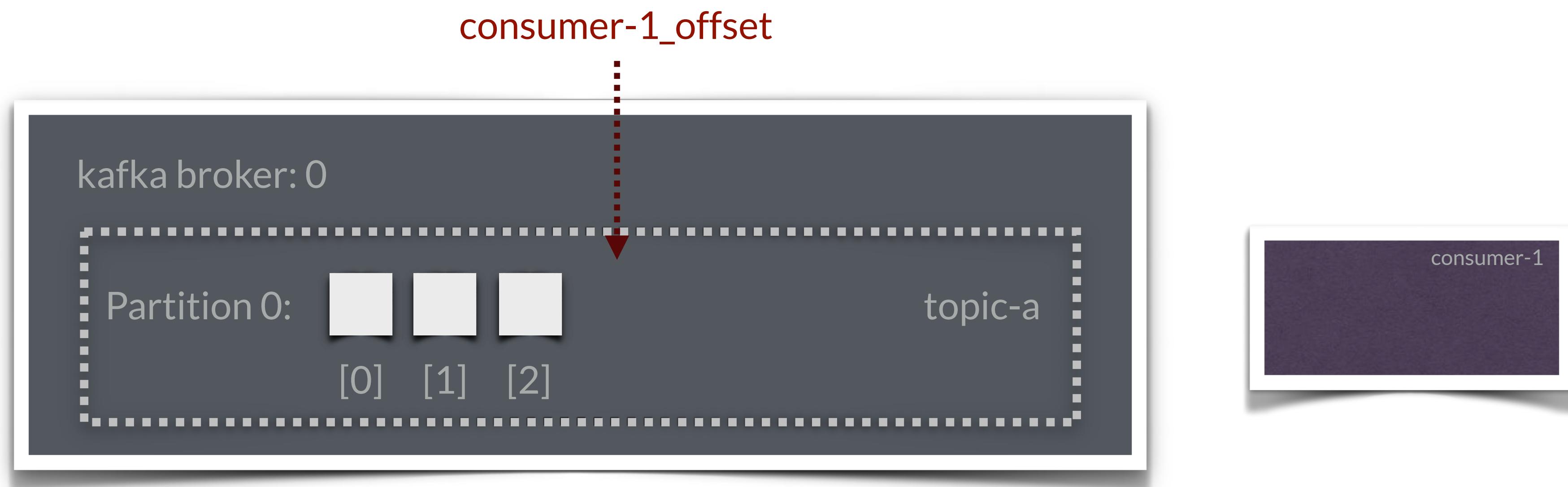
topic-a

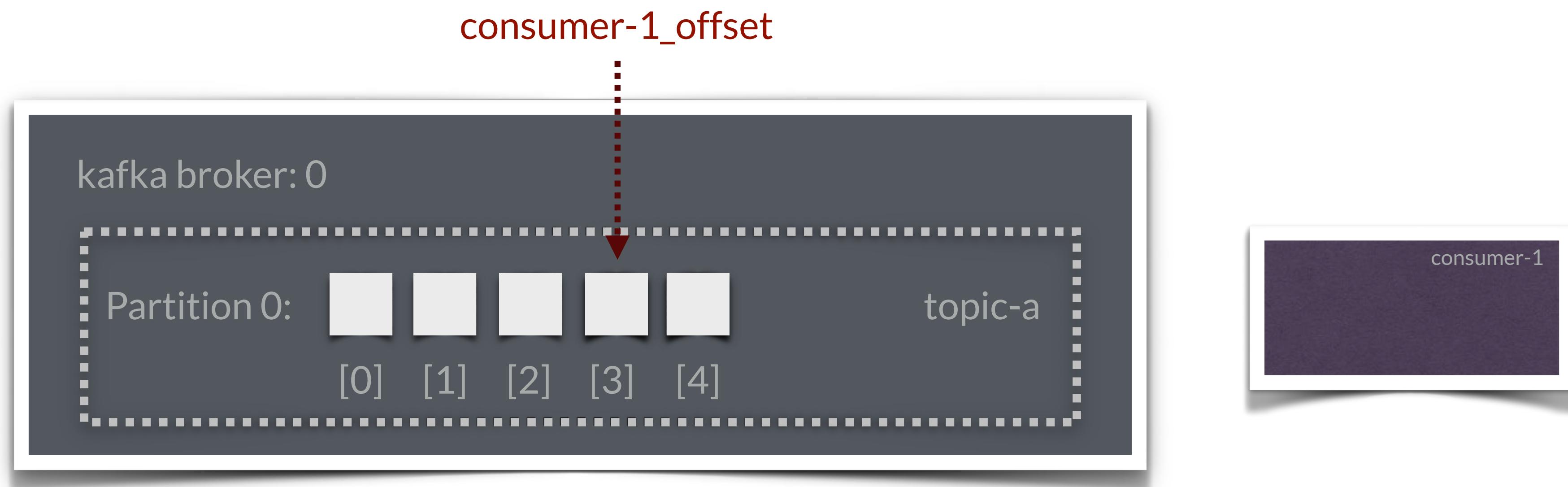




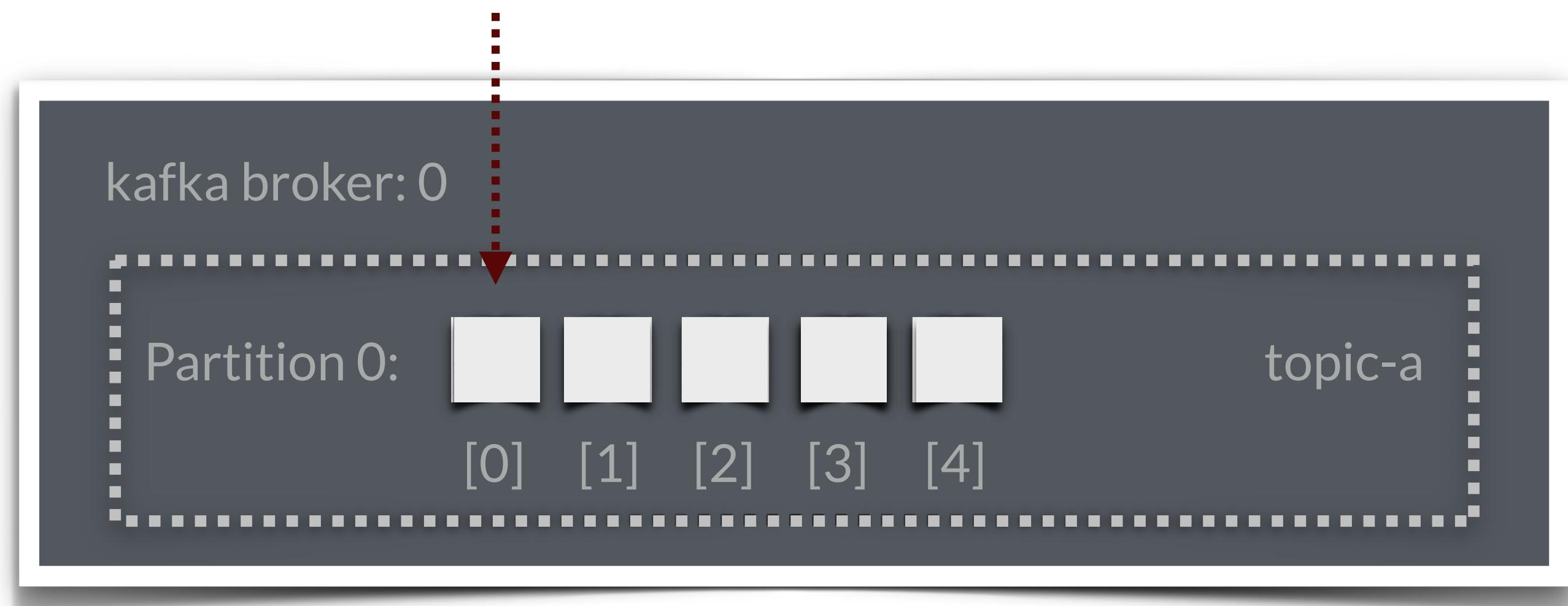
**Retention: The data is temporary**

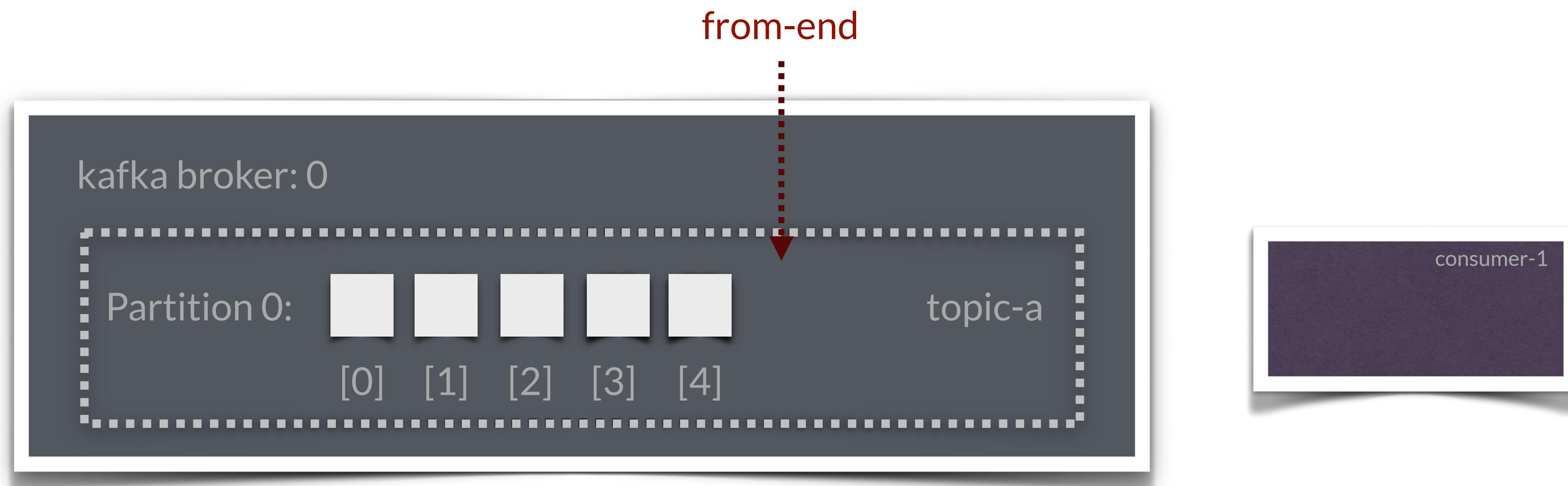
# How messages are consumed

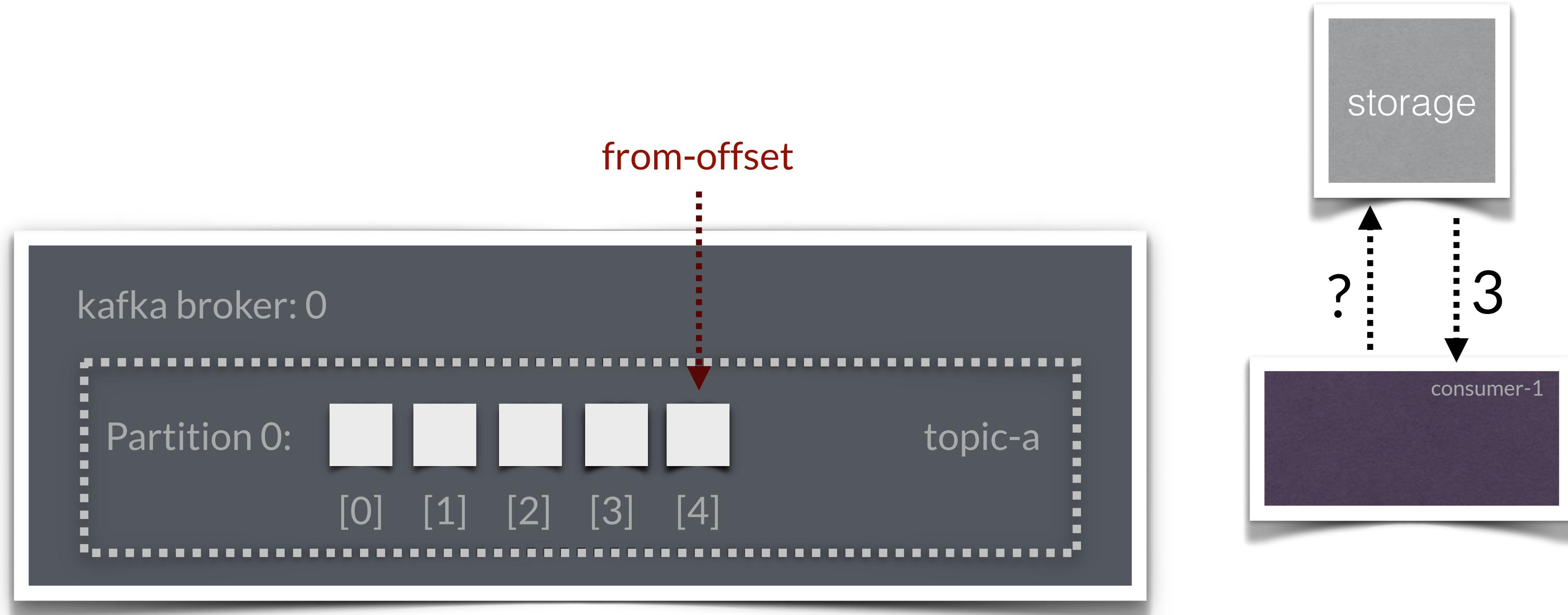


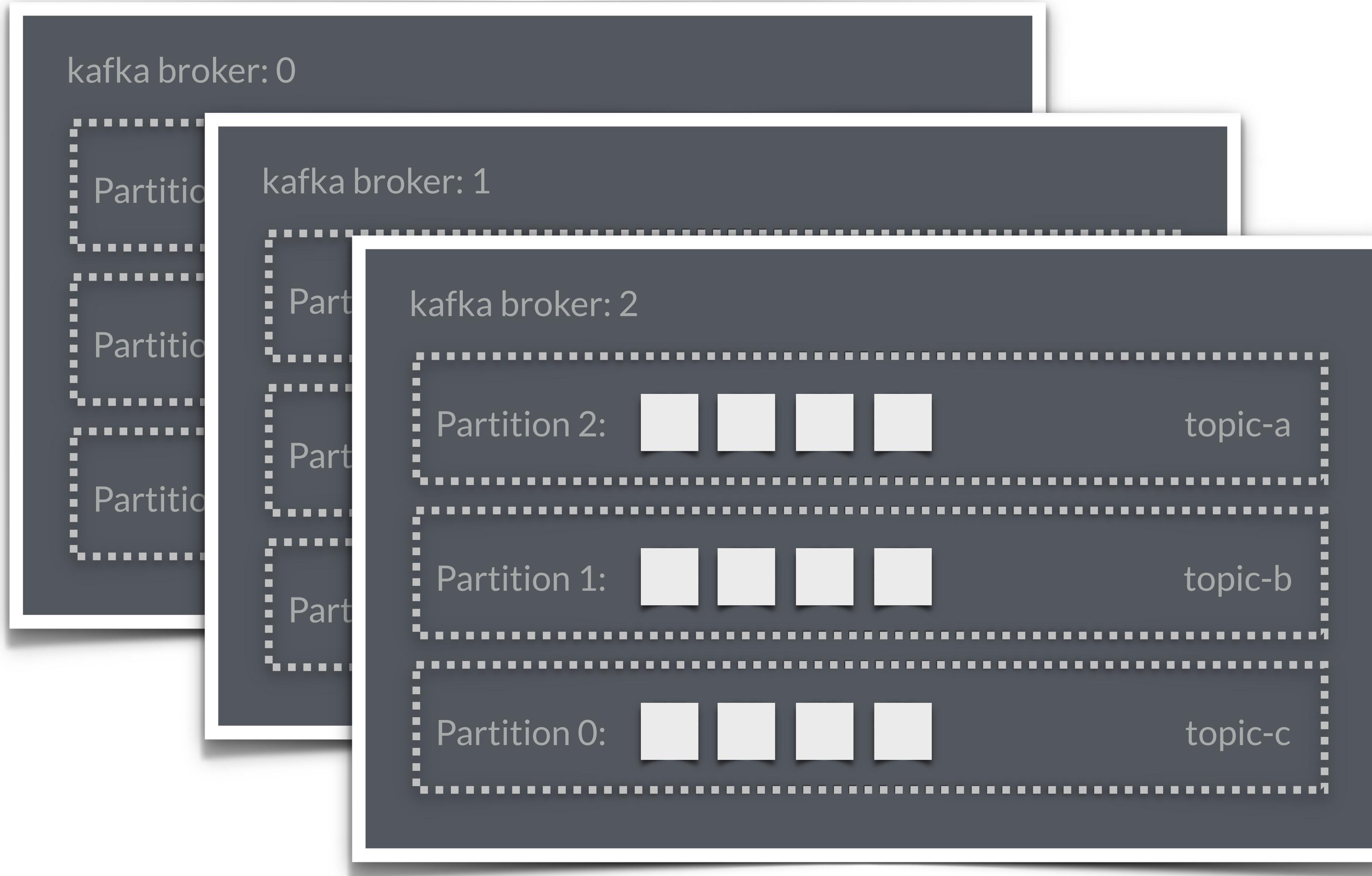


from-beginning









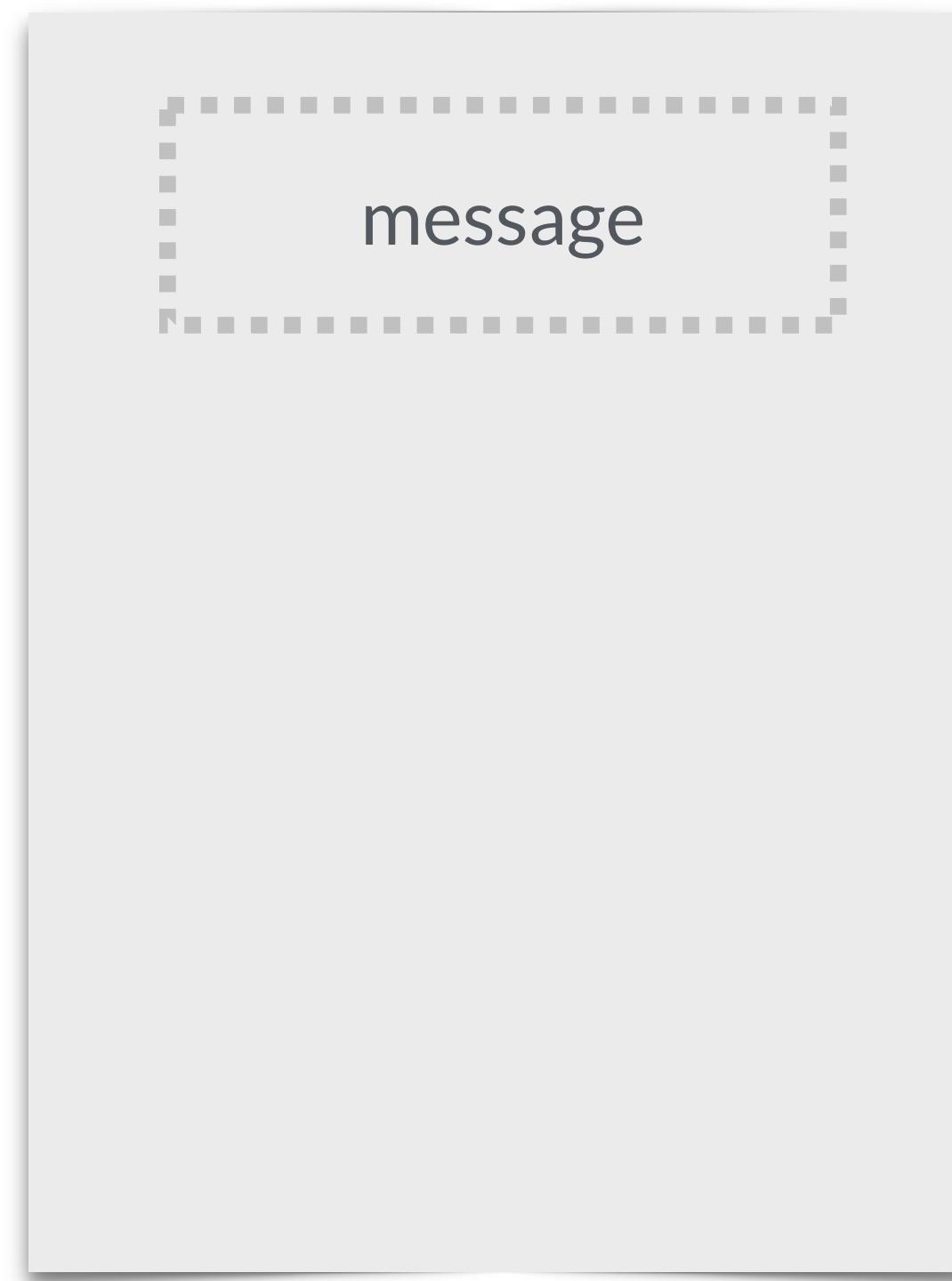
Each partition is on a different broker,  
therefore a single topic is scaled



# What is a message?

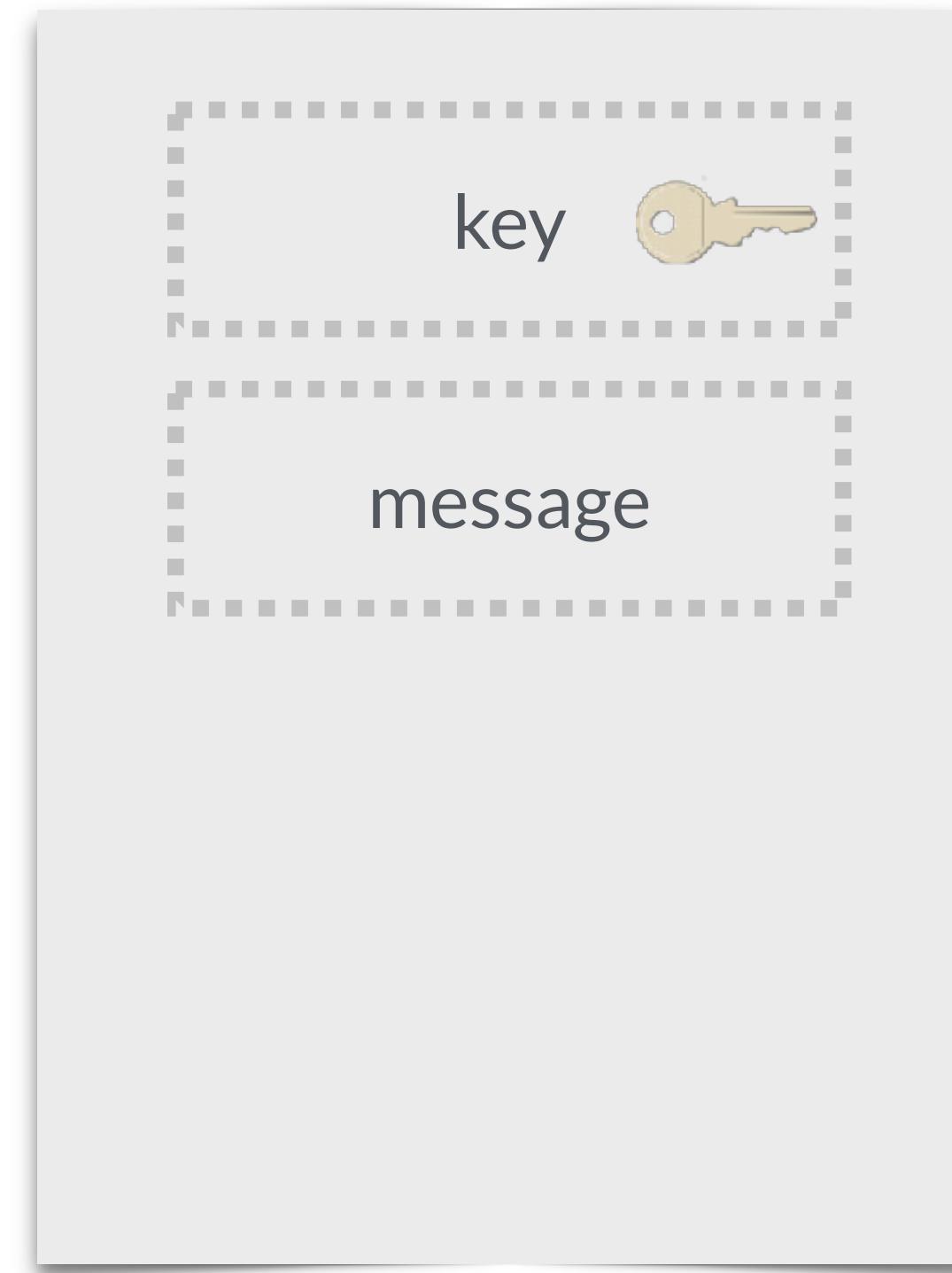
# A Kafka Message

- Similar to a *row* or a *record*
- Message is an array of bytes
- No special serialization, that is done at the producer or consumer



# A Kafka Message Key

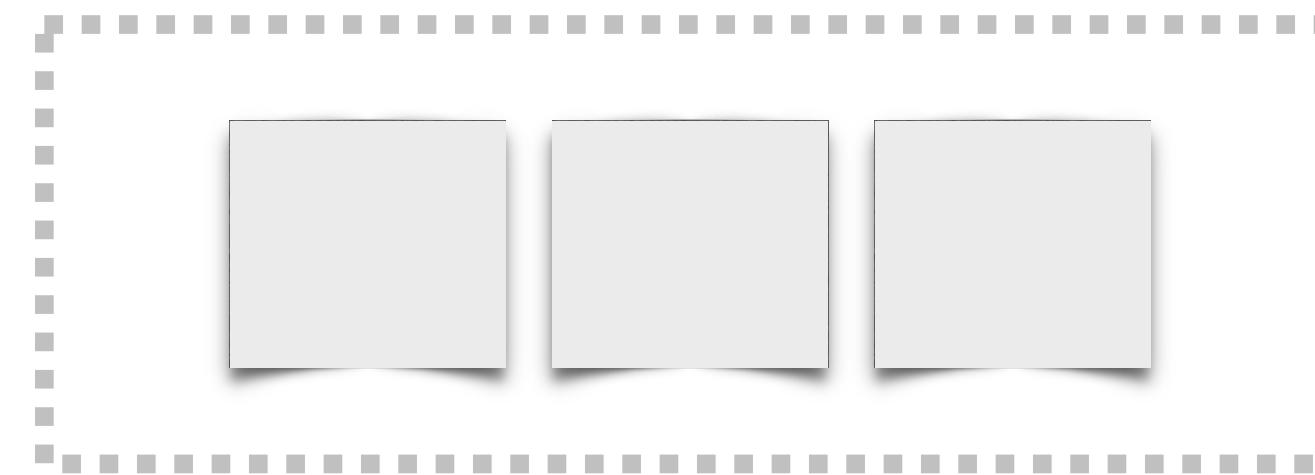
- Message may contain a *key* for better distribution to partitions
  - The *key* is also an array of bytes
  - If a *key* is provided, a partitioner will hash the key and map it to a single partition
  - Therefore it is the only time that something is guaranteed to be in order

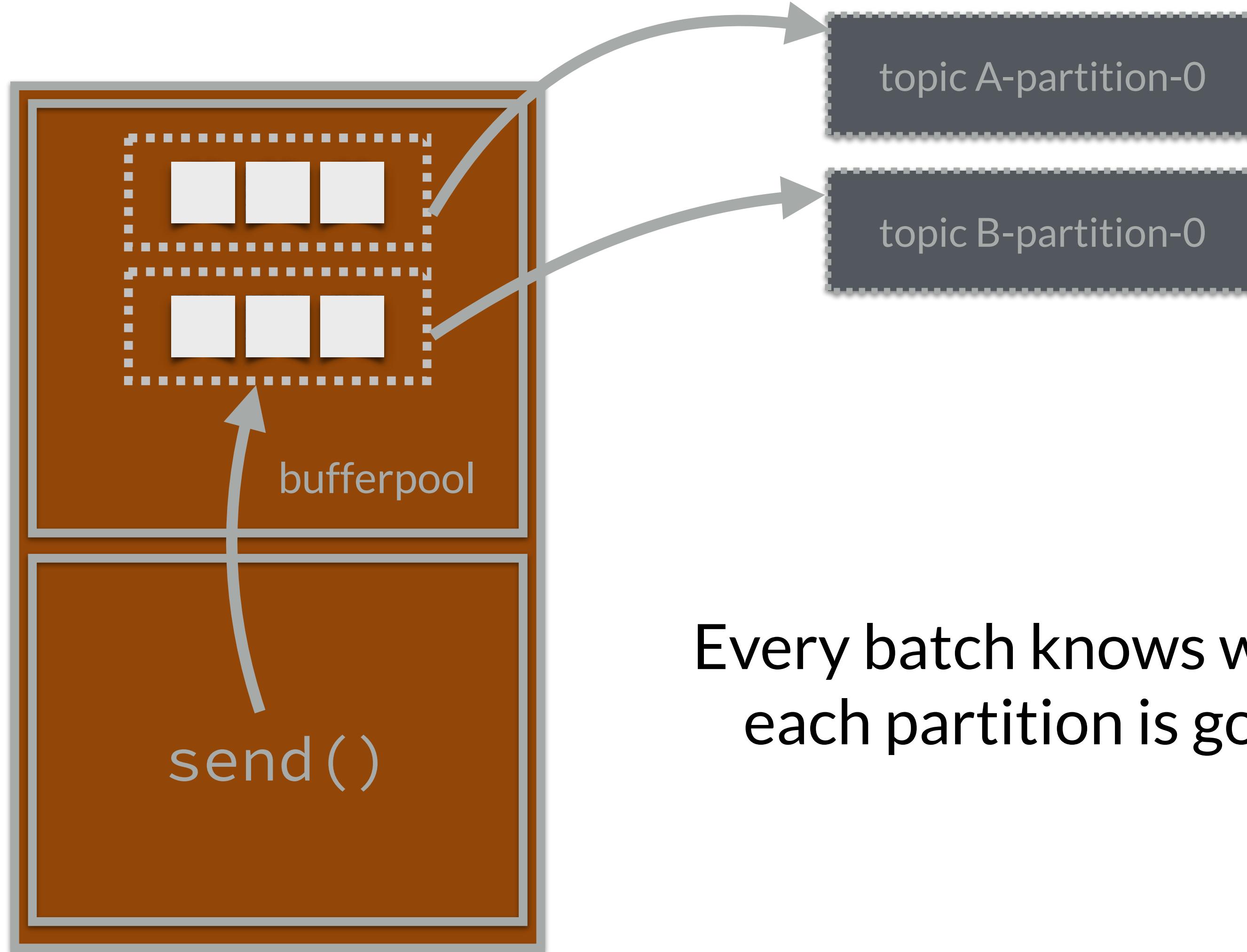


# Kafka Programming Producers

# Kafka Batch

- A collection of messages, that is sent, configured in *bytes*
- Sent to the same *topic* and the same *partition*
- *Avoids overhead of sending multiple message over the wire*





Every batch knows where  
each partition is going

$\text{murmur2(bytes)} \% \text{ number partitions}$

# Acks

# Acks

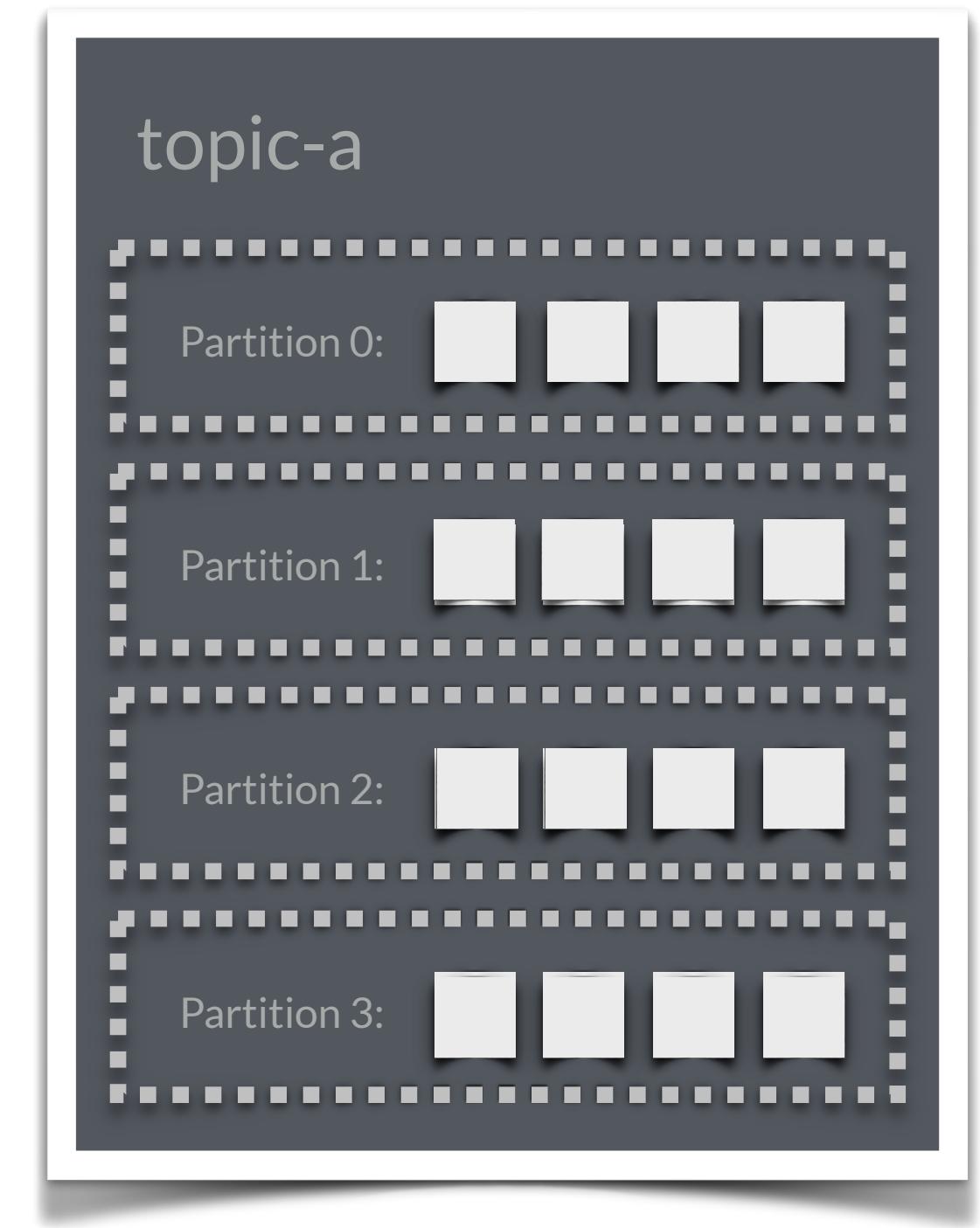
acks controls how many partition replicas must receive the record before the write is considered a success.

# Acks

acks	description
0	No acknowledgment, assume all is well
1	At least one replica will Producer will receive a success response, error if unsuccessful, up to client to hand
all	All replicas must acknowledge. Higher latency, safest

# Demo: Producers

# Kafka Programming Consumers & Groups

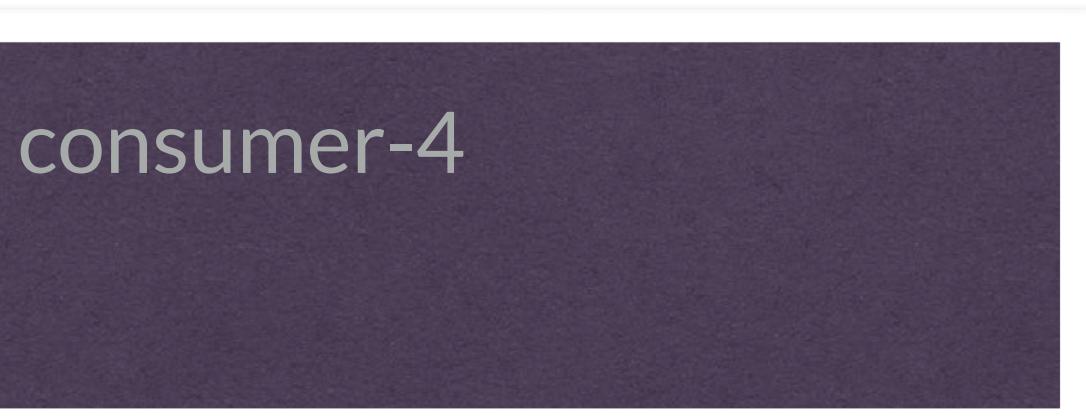
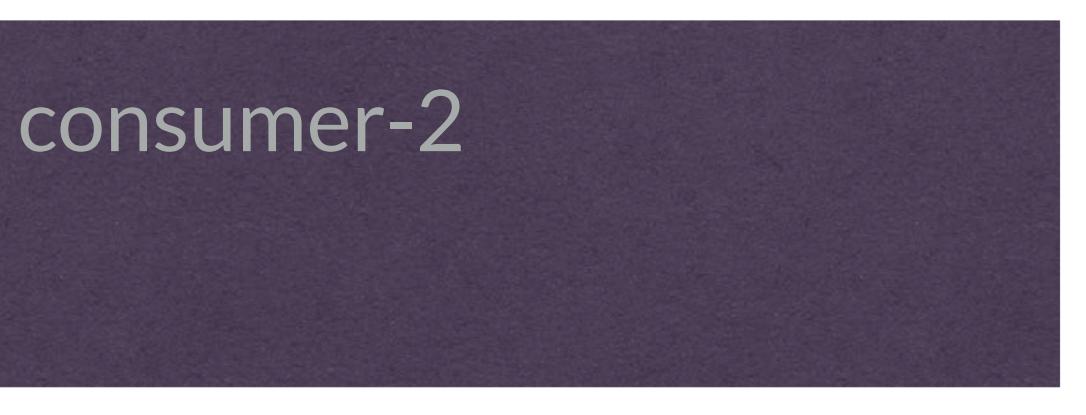
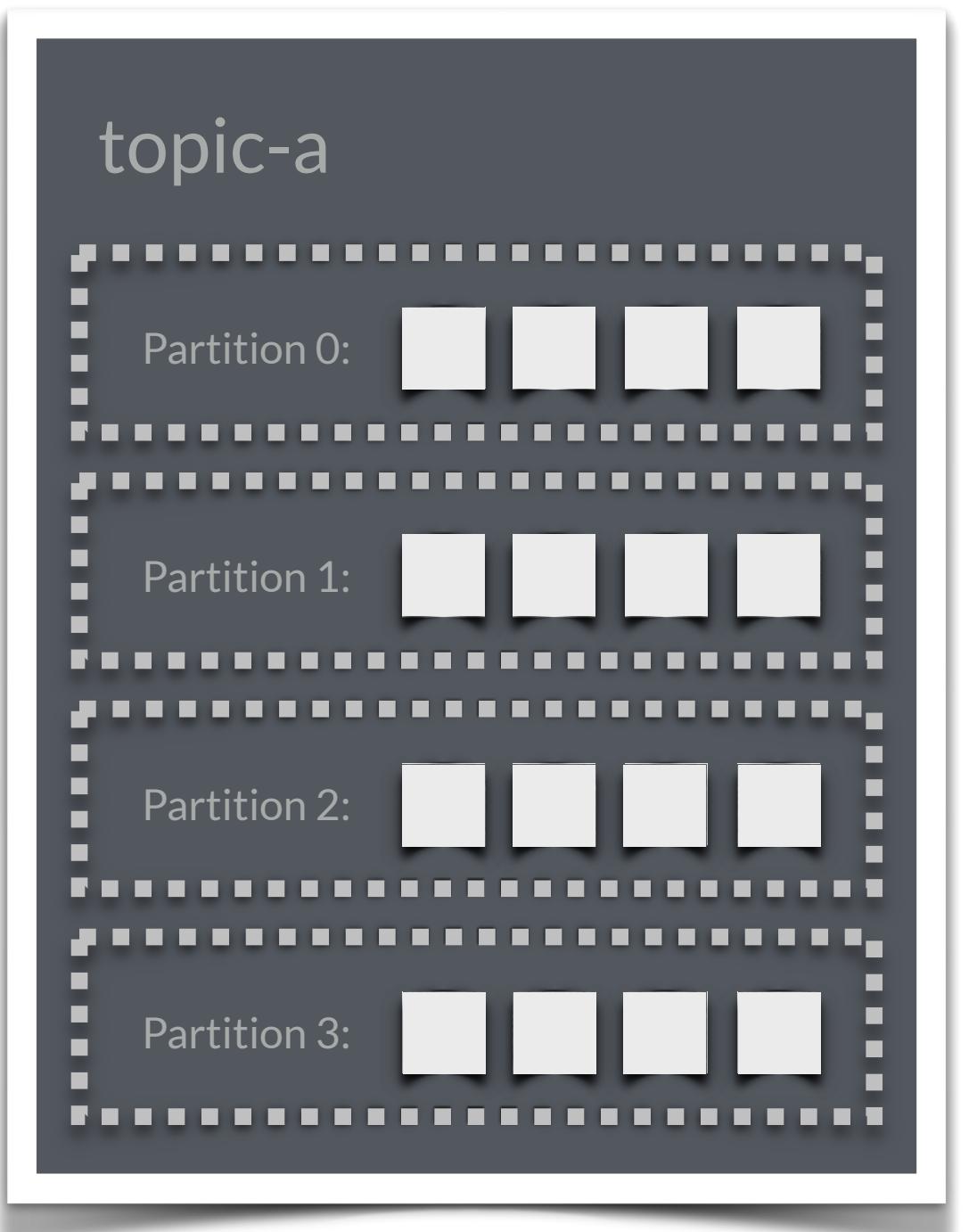


# Kafka Consumer Groups

Consumers are typically done as a group

A single consumer will end up inefficient with large amounts of data

A consumer may never catch up



# Kafka's Goal

Kafka scales to large amount of different consumers  
without affecting performance

# Kafka Consumer Threading

There are no multiple consumers that belong to the same group in one thread.

**One consumer per one thread.**

There are not multiple threads running one consumer,  
**One consumer per one thread.**

# consumer\_offsets

Topic on Kafka brokers that contain information about consumer and their offsets, stored in Kafka's data directory

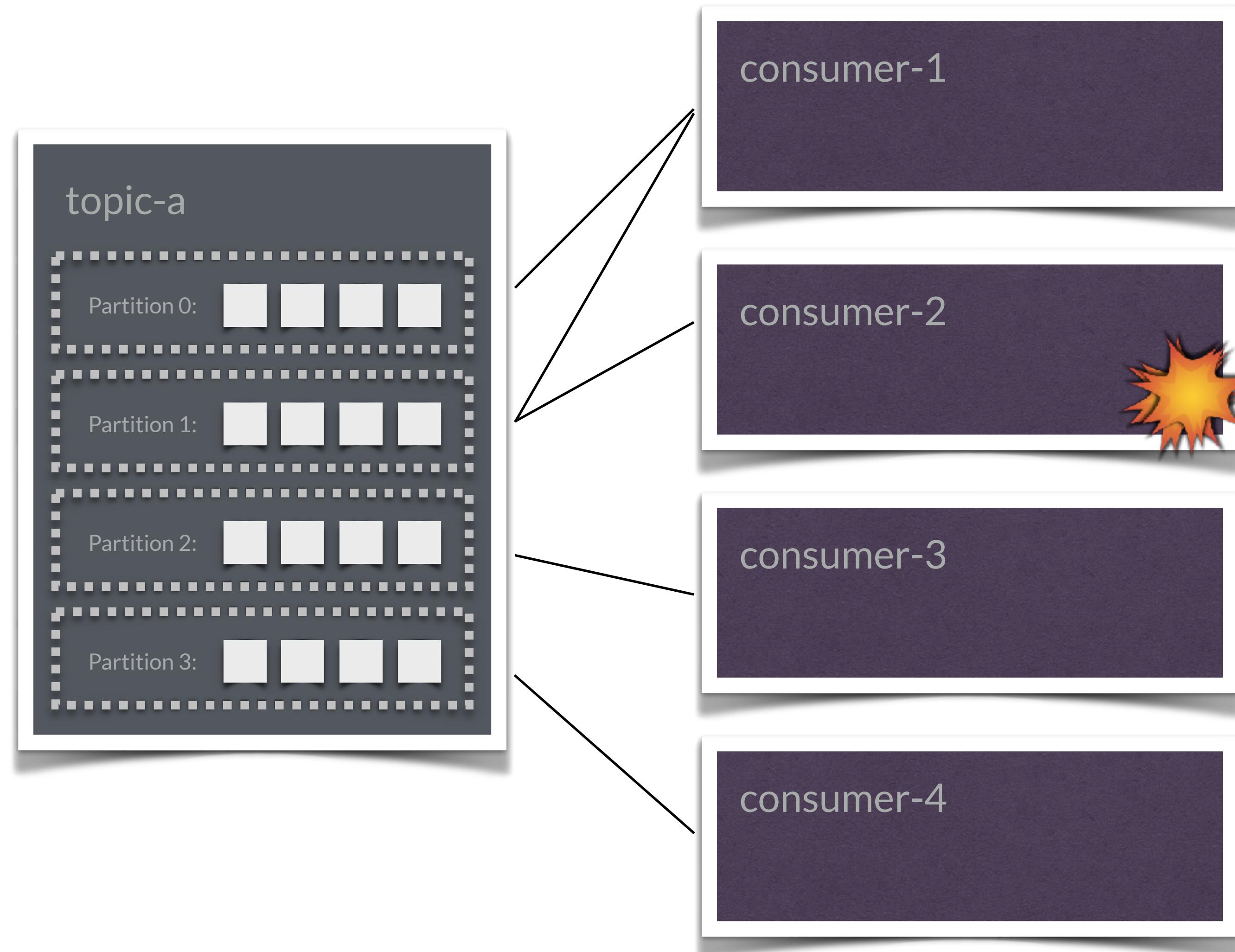
# Consumer Rebalancing

# Consumer Rebalance

When one partition is moved from one consumer to another, this is known as a *rebalance*.

A way to mitigate when either consumers go down, or when consumers are added.

Although unavoidable, this will cause an unfortunate pause, and it *will lose state*.



# Groups and Heartbeats

# Groups and Heartbeats

At regular intervals, consumers will send heartbeats to the broker, to let it know it is alive and still reading data

Heartbeats are sent to a Kafka broker called the *group coordinator*

They are sent when the consumer polls and consumes a record

# Leaving the Group

When a consumer leaves the group, it lets the *group coordinator* know it is done

The Kafka broker then triggers a rebalance of the group.

# Demo: Consumers

# Compaction

# Compaction

A form of retention where messages of the same key where only the latest message will be retained.

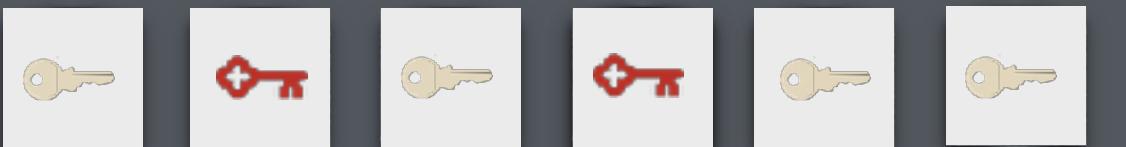
Compaction is performed by a *cleaner thread*.

*This is particularly important when it comes to a notion of tables!*

*Tables and their corresponding topics are compacted!*

kafka broker: 0

Partition 0:

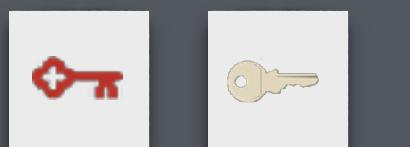


[0] [1] [2] [3] [4] [5]



kafka broker: 0

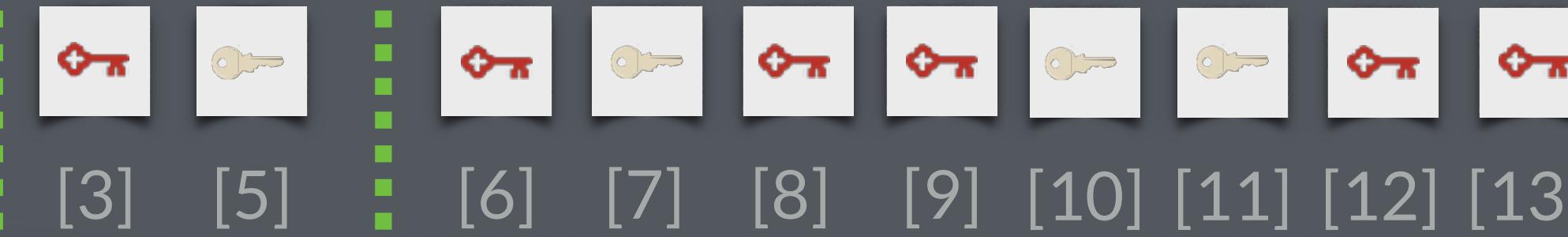
Partition 0:



[3] [5]

kafka broker: 0

Partition 0:



clean

dirty

Kafka will start compacting when 50% of the topic contains dirty records

# Streaming



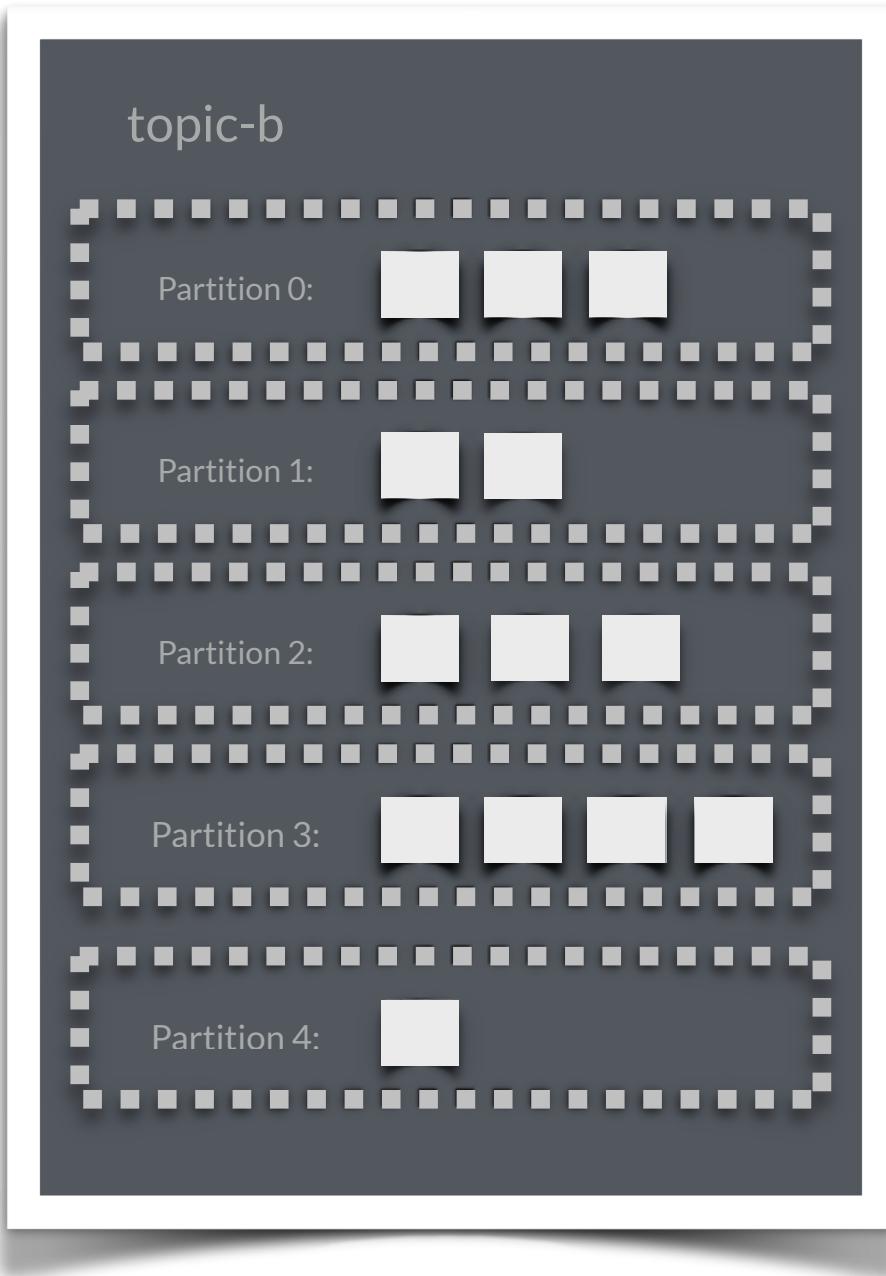
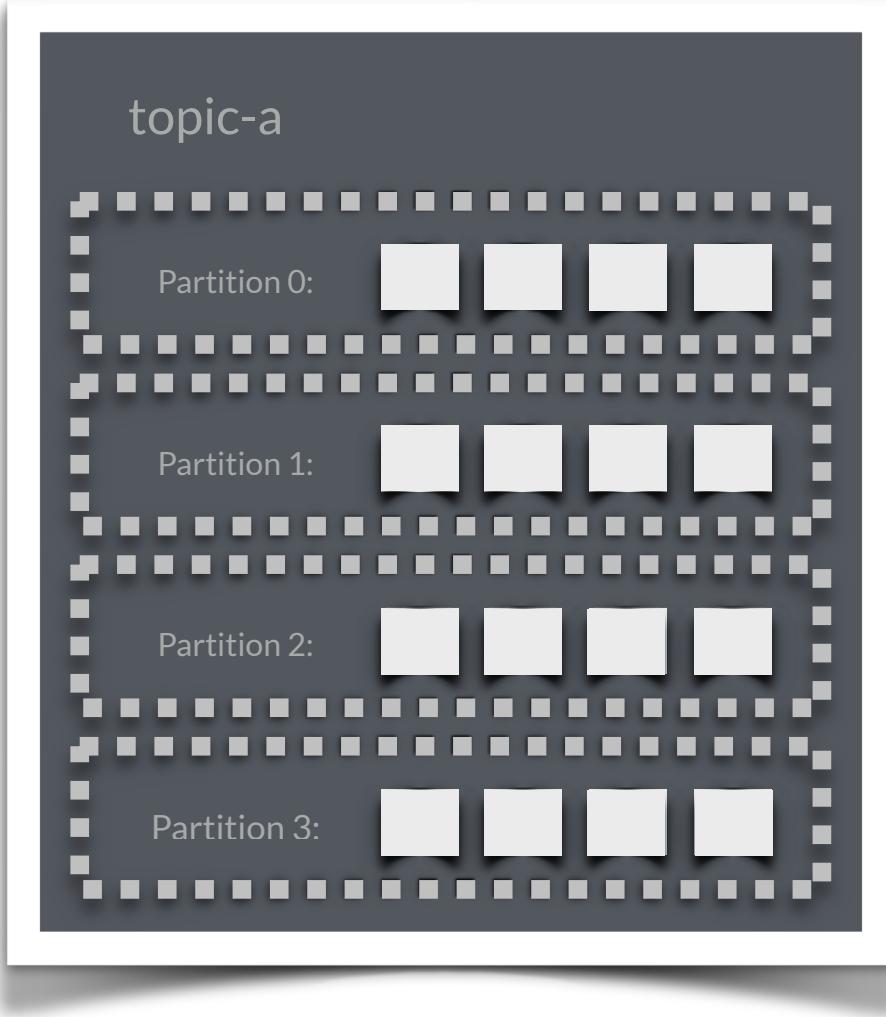
# Stream Processing

Endless supply of data

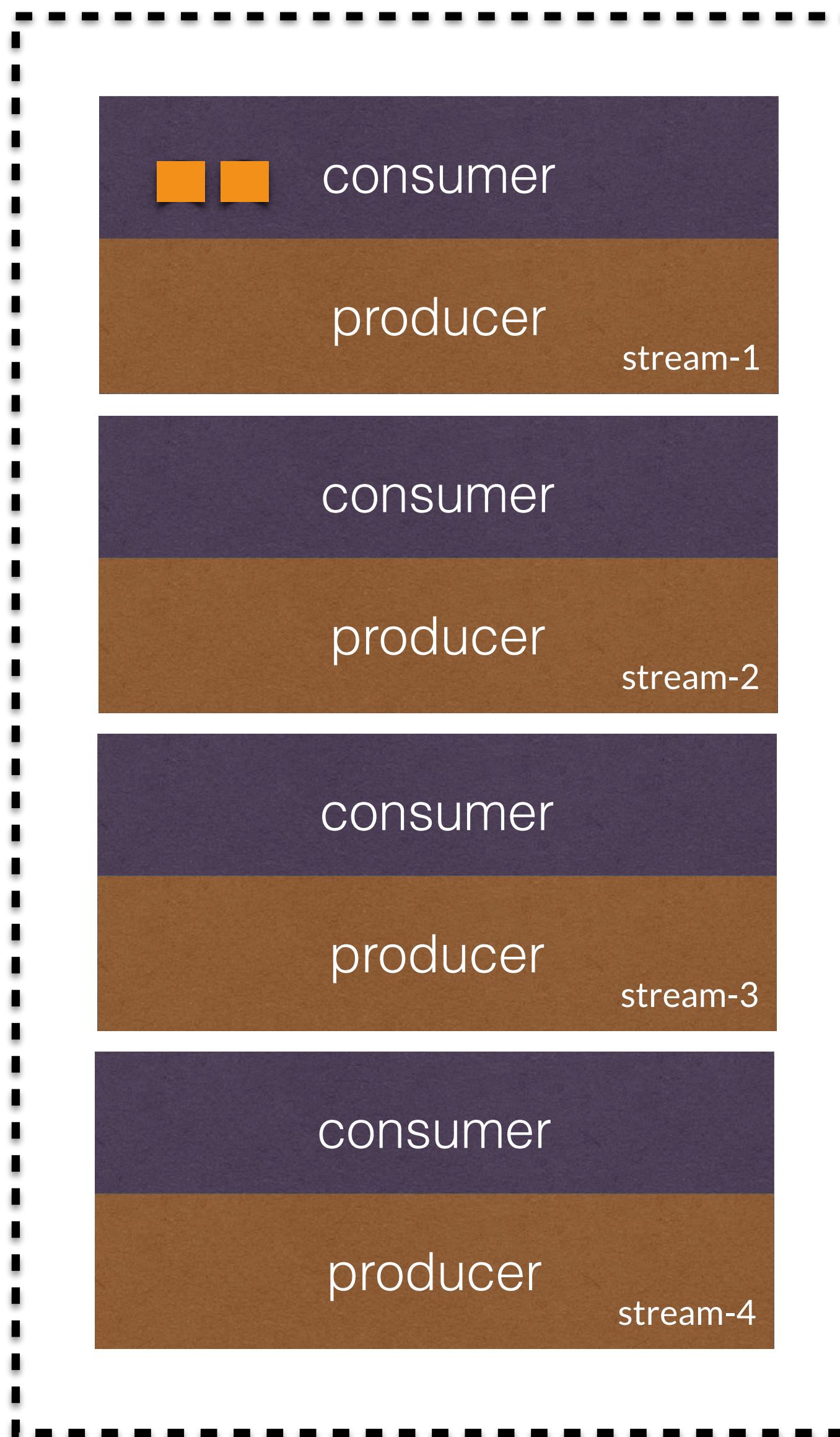
"Replayable"

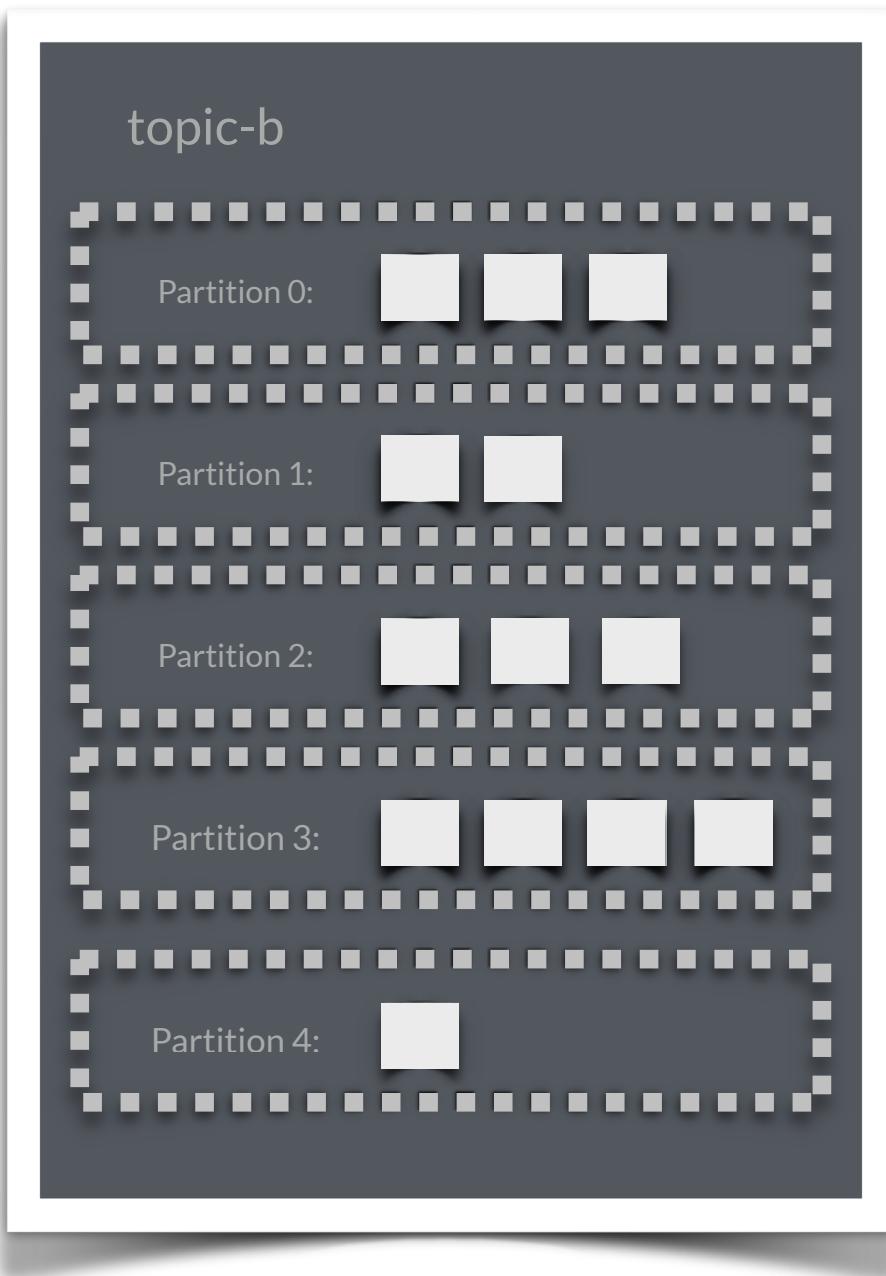
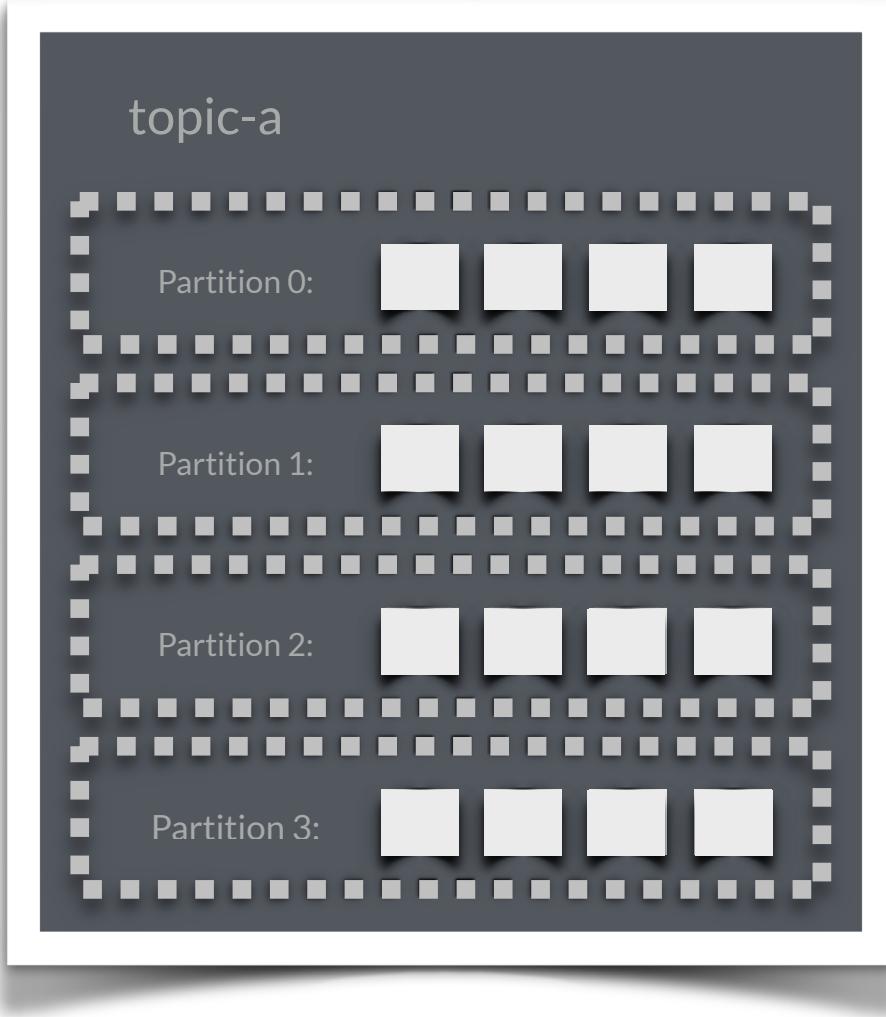
Real Time Processing for Fraud Detection, High End Sales Detection, Internet of Things, and More.

Will require built-in accumulator table to perform real time data processing, like *counting*, and *grouping*

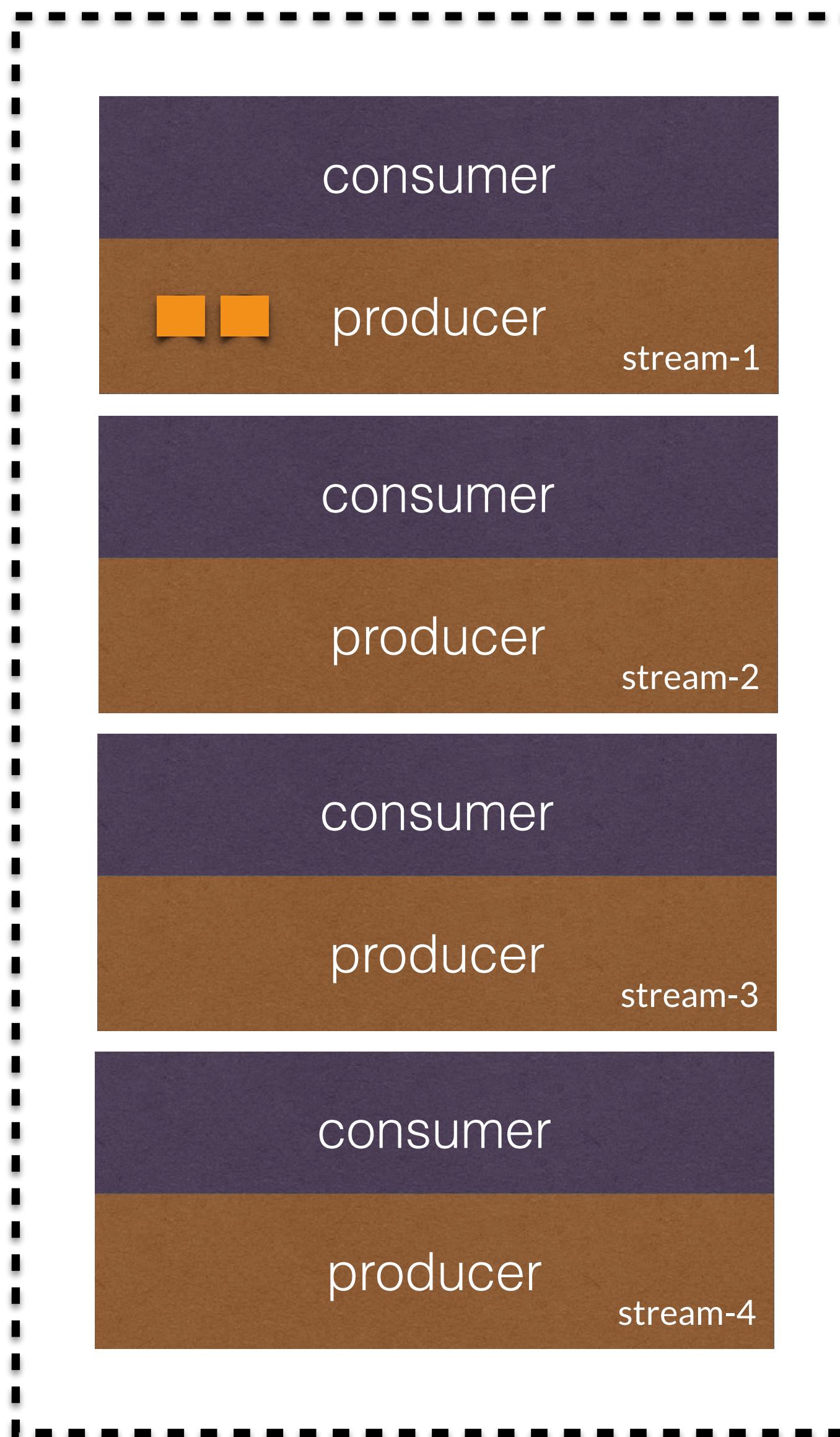


application.id = “my-stream”





application.id = “my-stream”

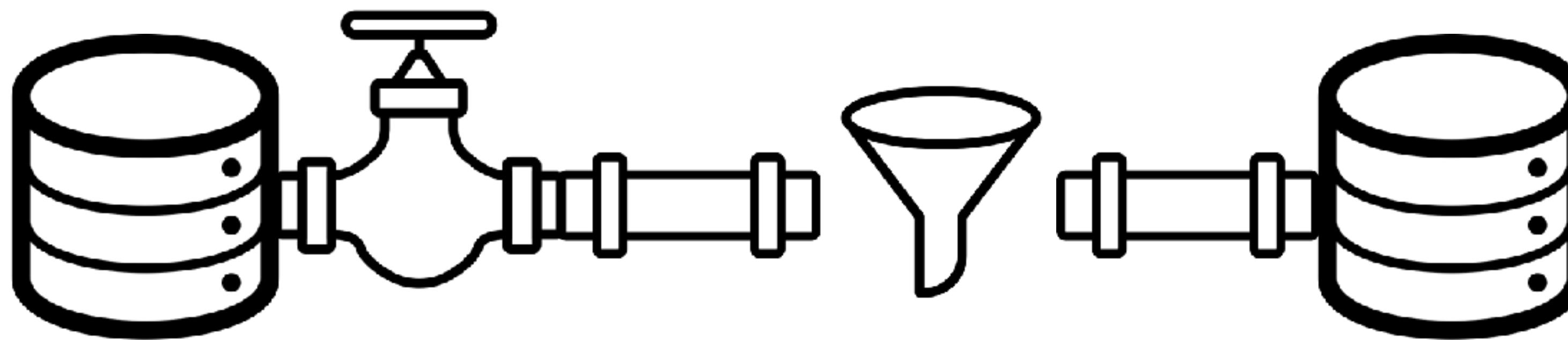


NY

100.00

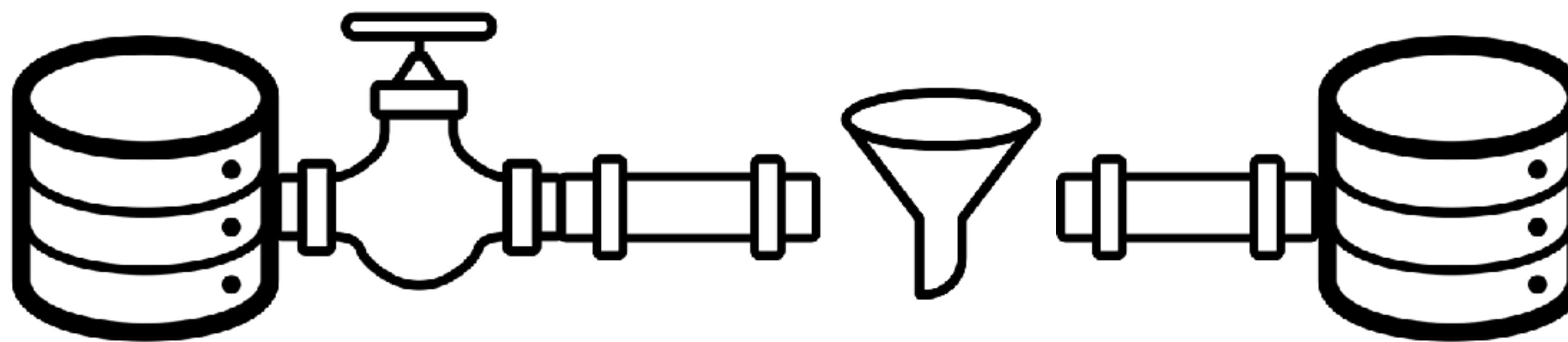
# Filtering

NY  
.....  
100.00



value > 10000

OH  
.....  
20000

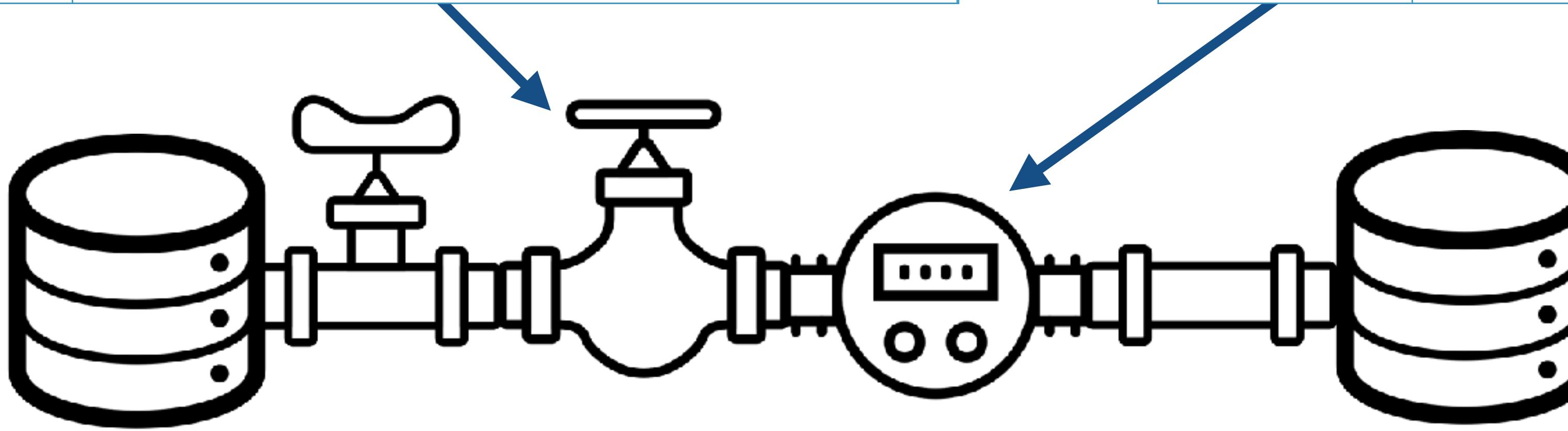


value > 10000

# Aggregating

Key	Value
<b>OH</b>	100.0

Key	Value
<b>OH</b>	100.0

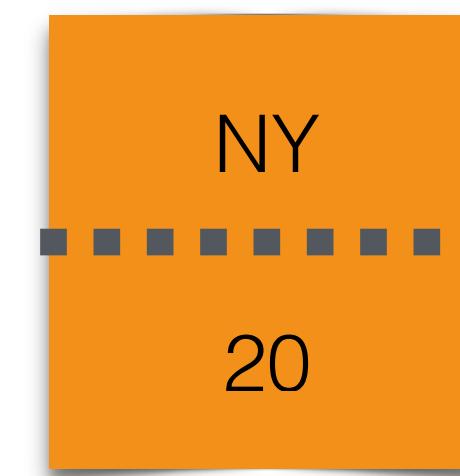
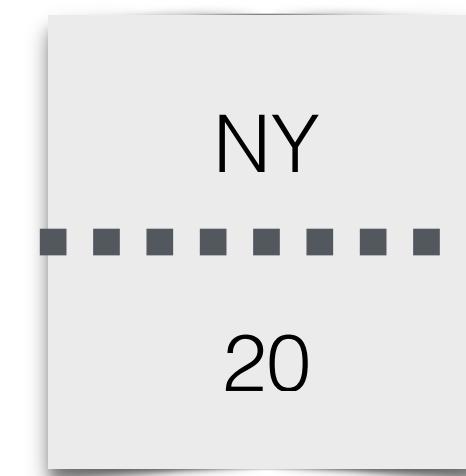
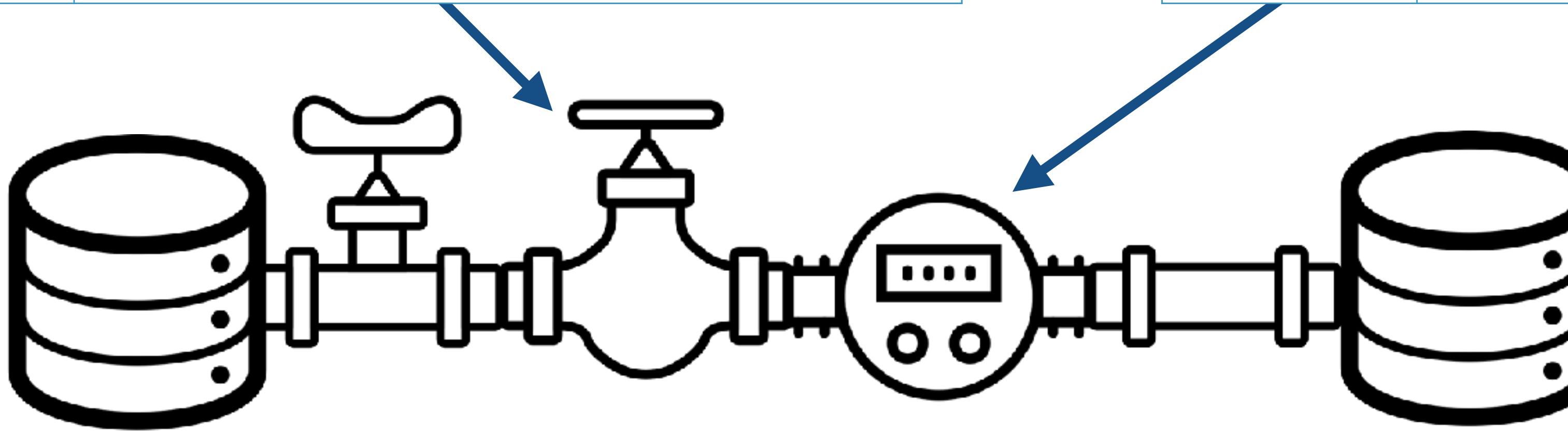


OH  
.....  
100.00

OH  
.....  
100.00

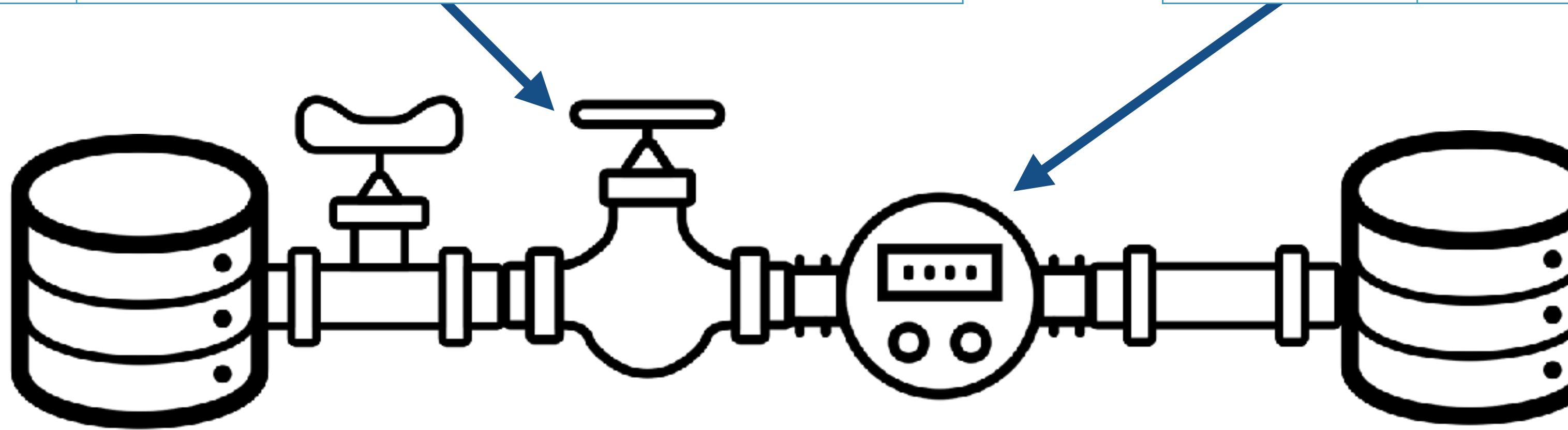
Key	Value
<b>OH</b>	100.0
<b>NY</b>	20.0

Key	Value
<b>OH</b>	100.0
<b>NY</b>	20.0



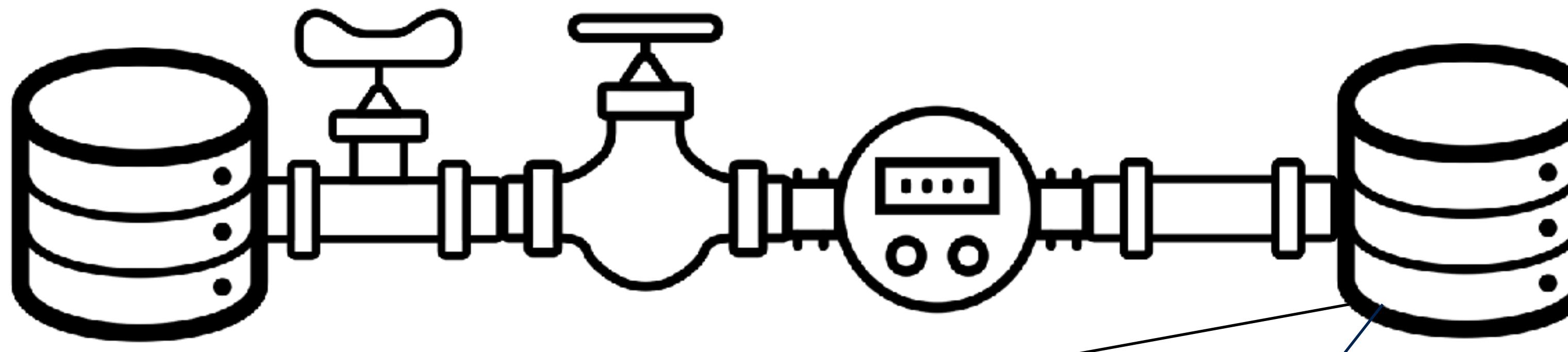
Key	Value
<b>OH</b>	100.0
<b>NY</b>	20.0, 60.0

Key	Value
<b>OH</b>	100.0
<b>NY</b>	80.0



NY  
.....  
60

NY  
.....  
80.0

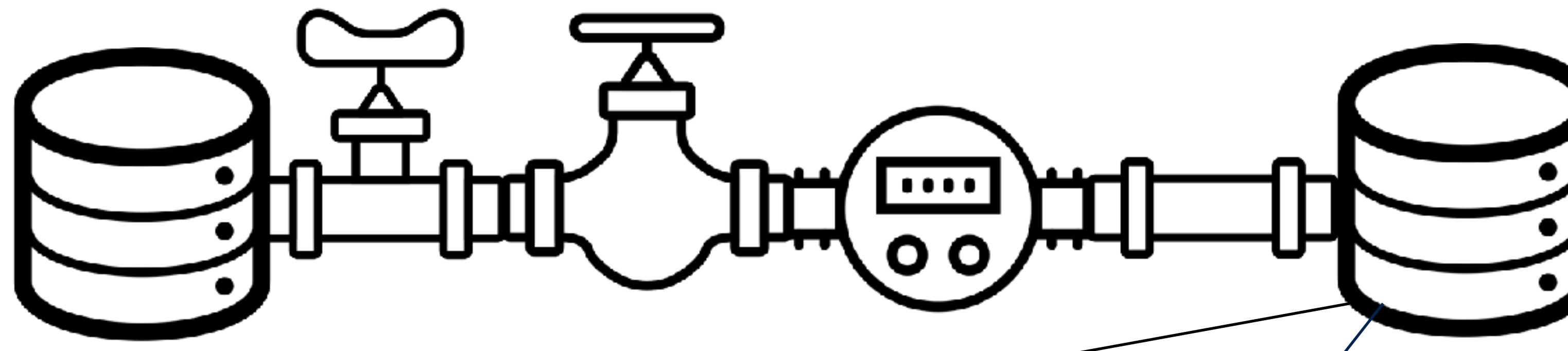


Offset	Key	Value
<b>0</b>	<b>OH</b>	100.0

Partition 0

Offset	Key	Value
<b>0</b>	<b>NY</b>	20.0
<b>1</b>	<b>NY</b>	80.0

Partition 1



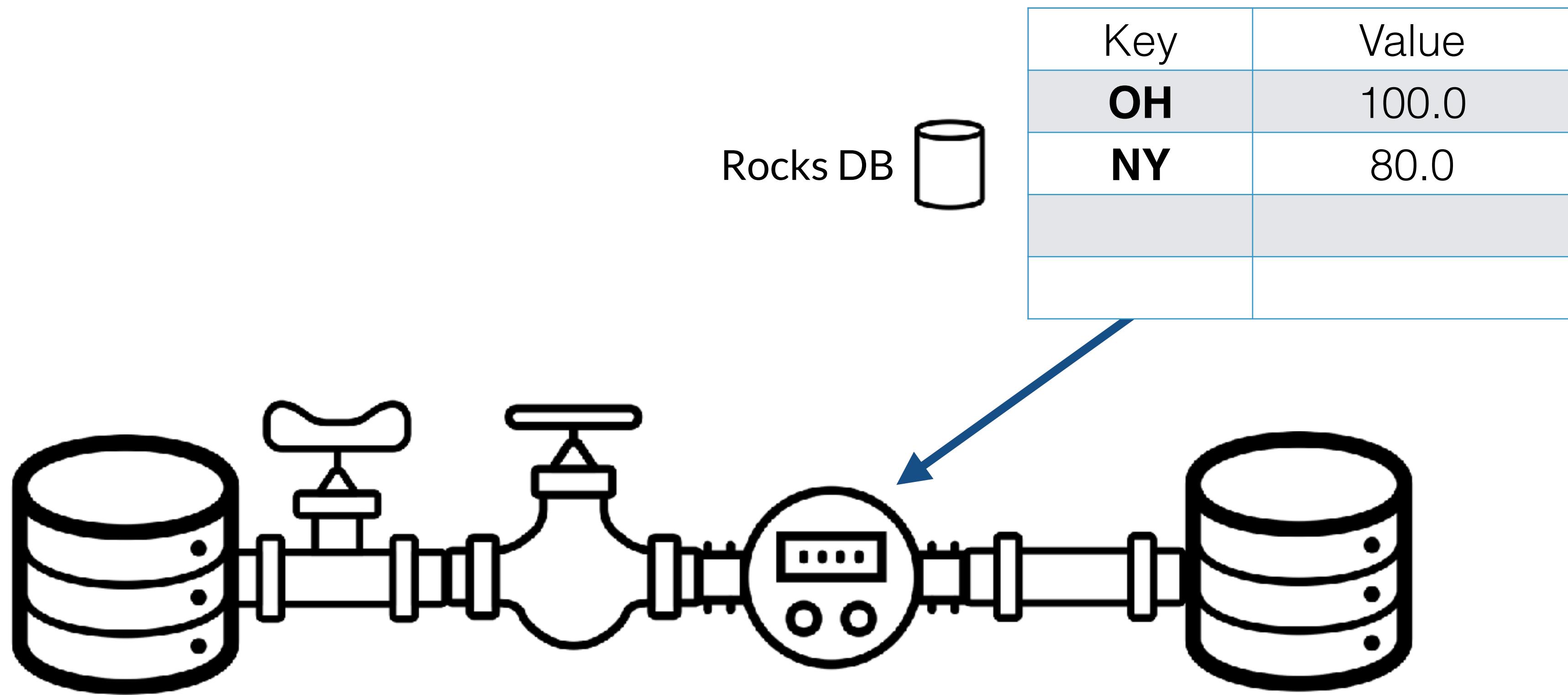
Offset	Key	Value
<b>0</b>	<b>OH</b>	100.0

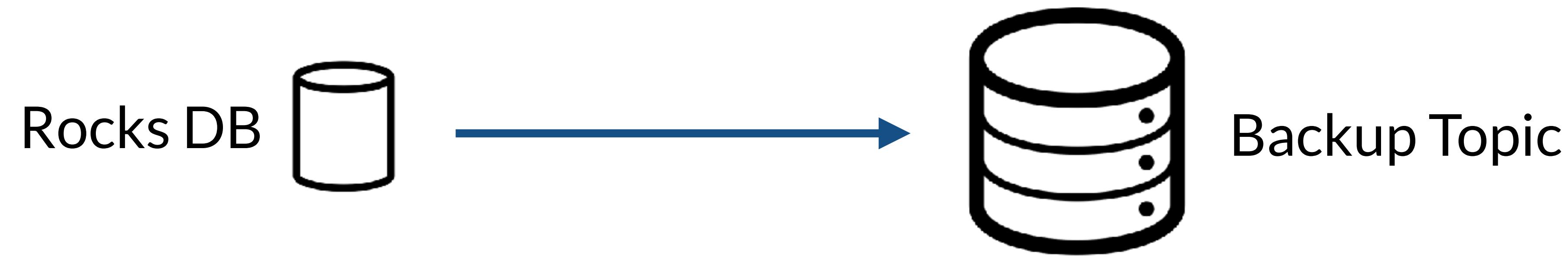
Partition 0

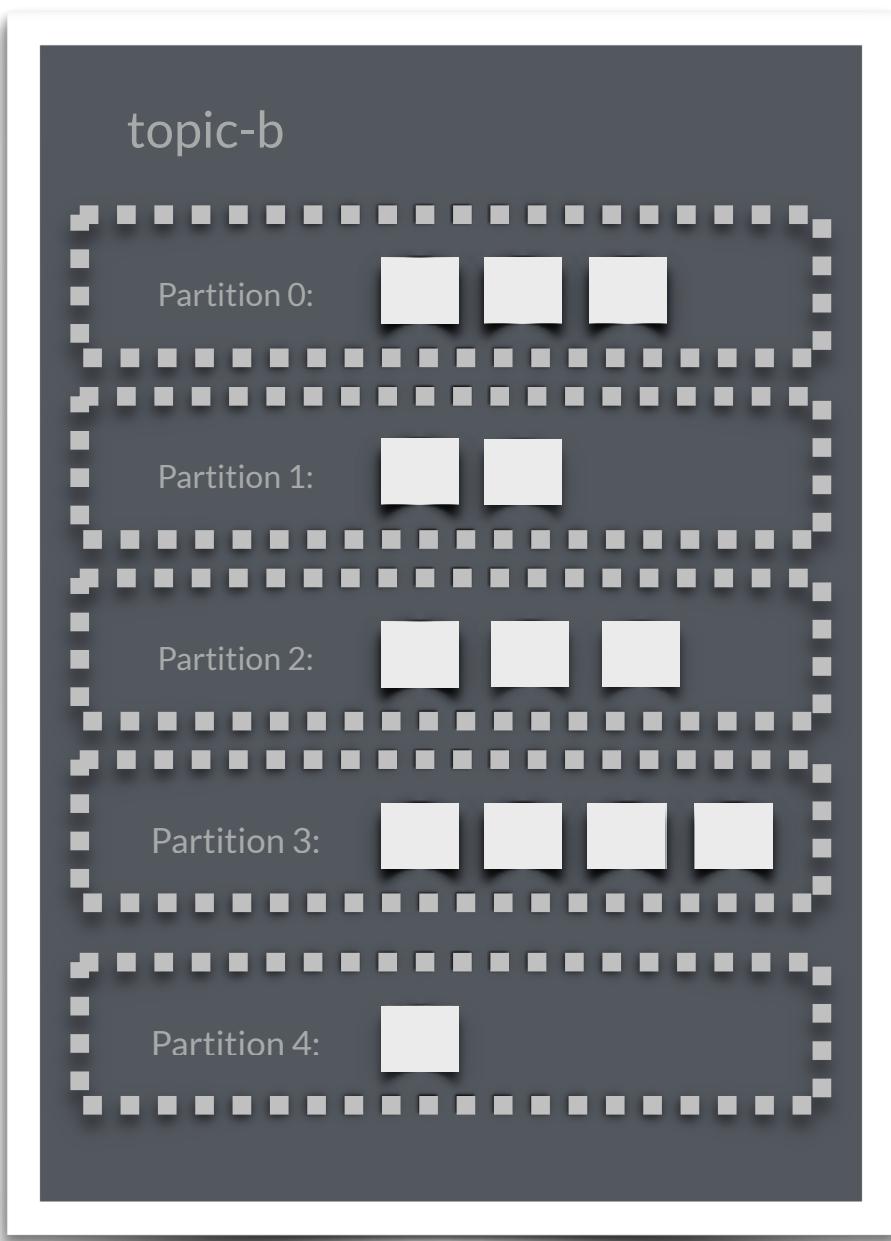
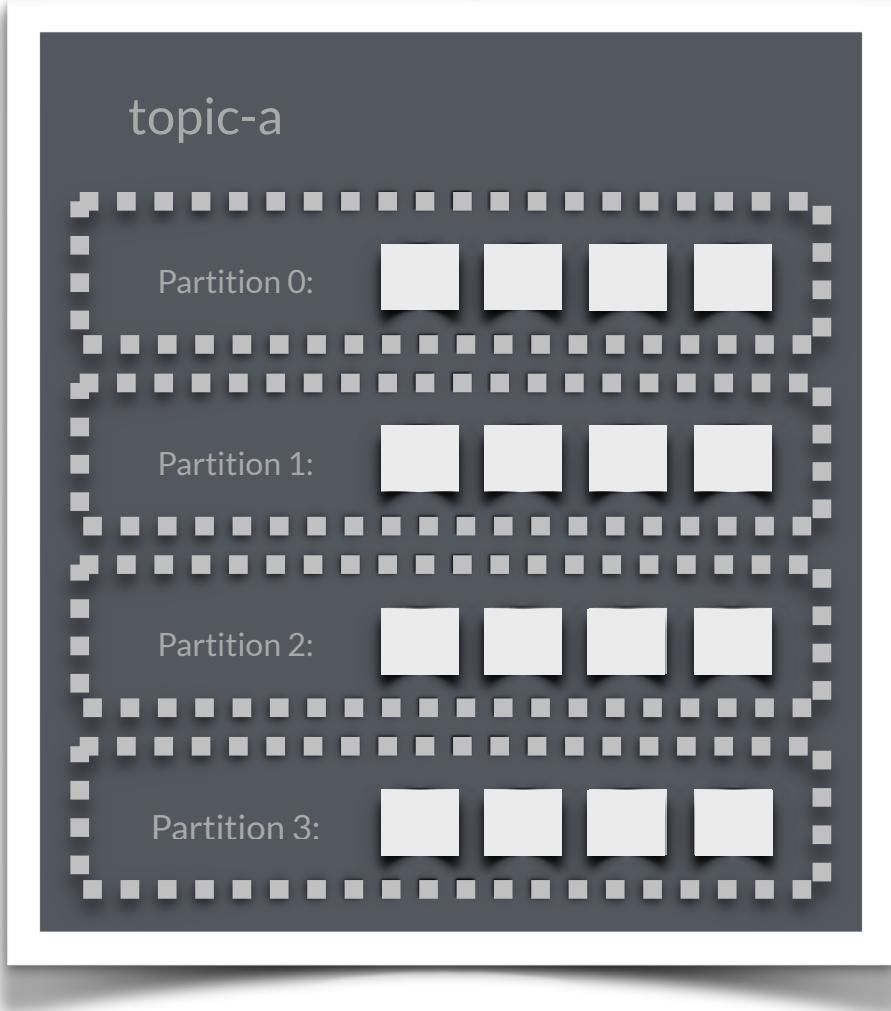
Offset	Key	Value
<b>1</b>	<b>NY</b>	80.0

Partition 1

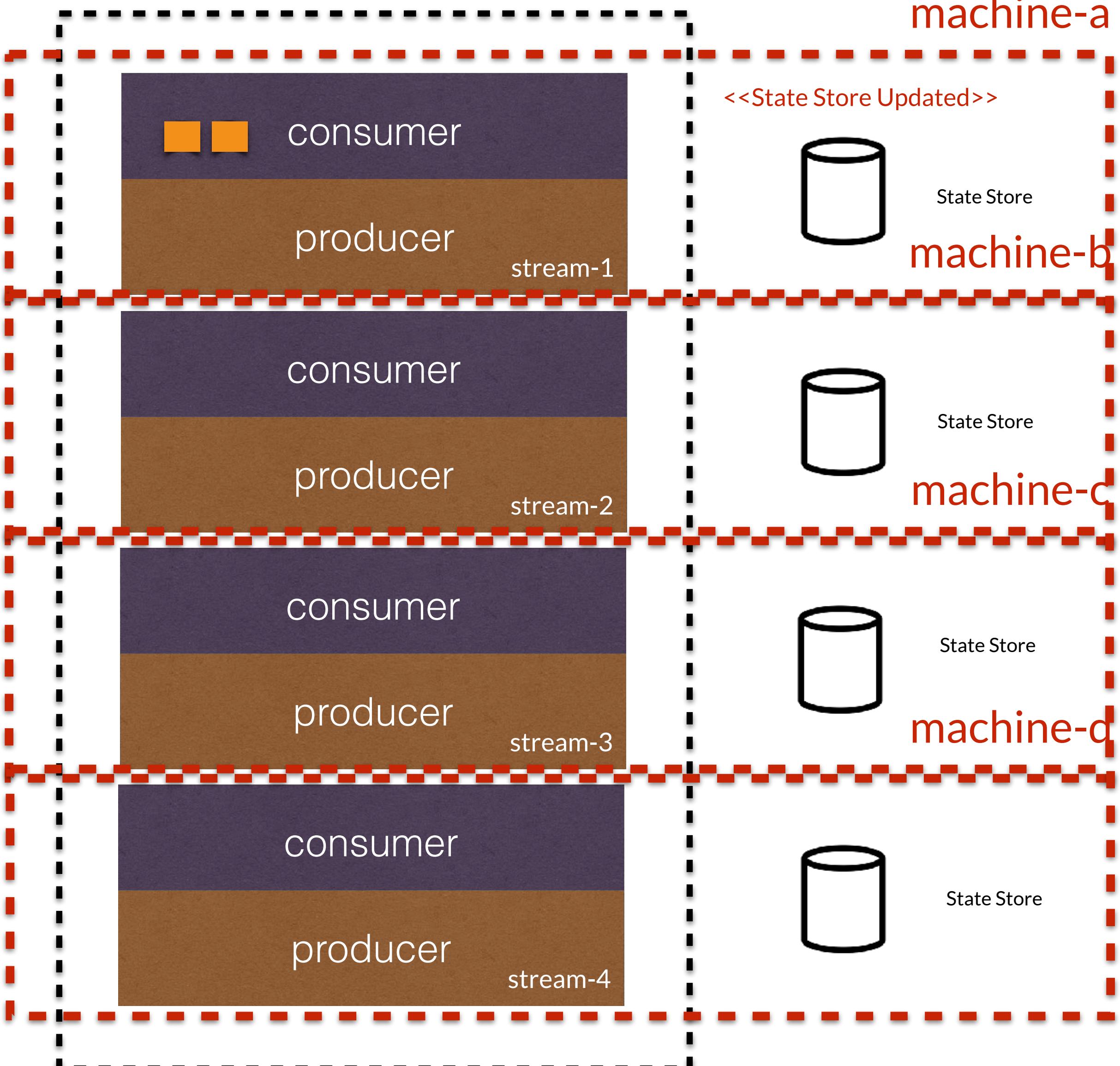
After Compaction



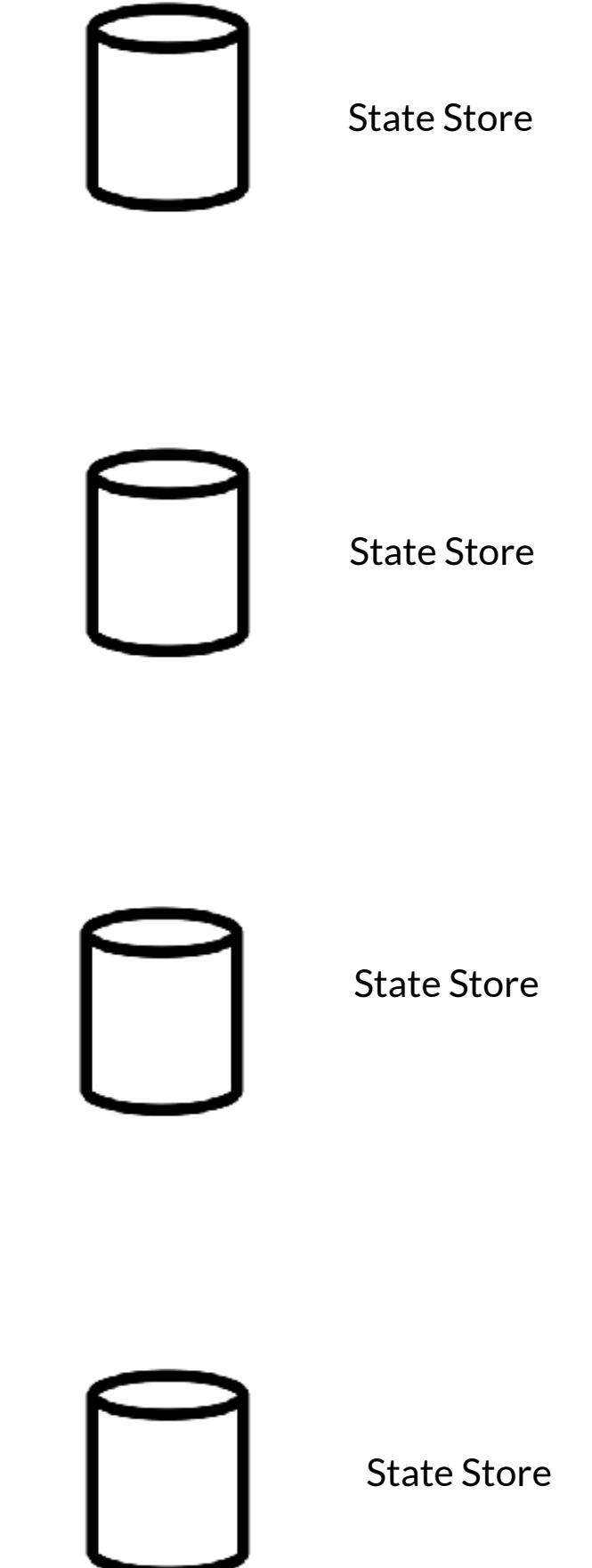
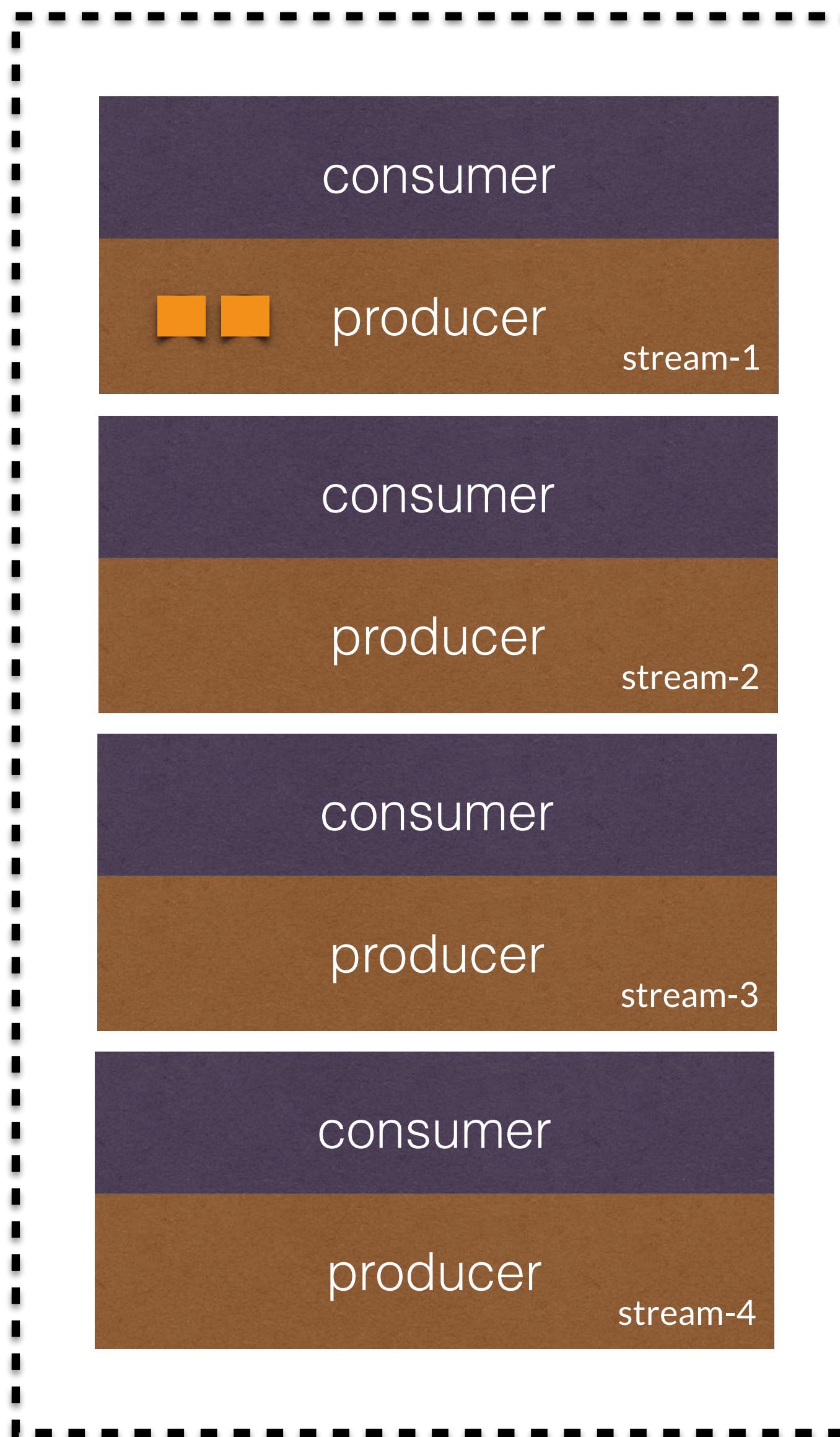
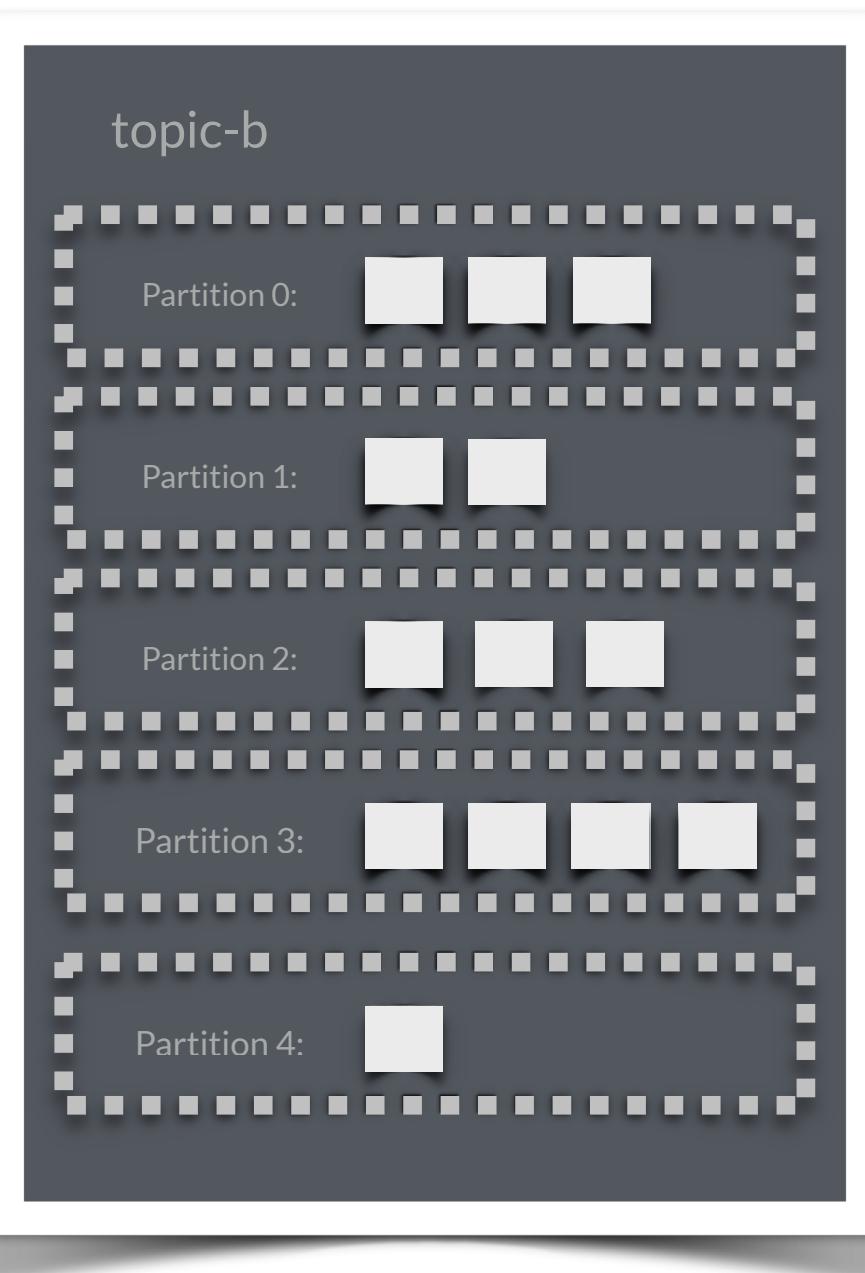
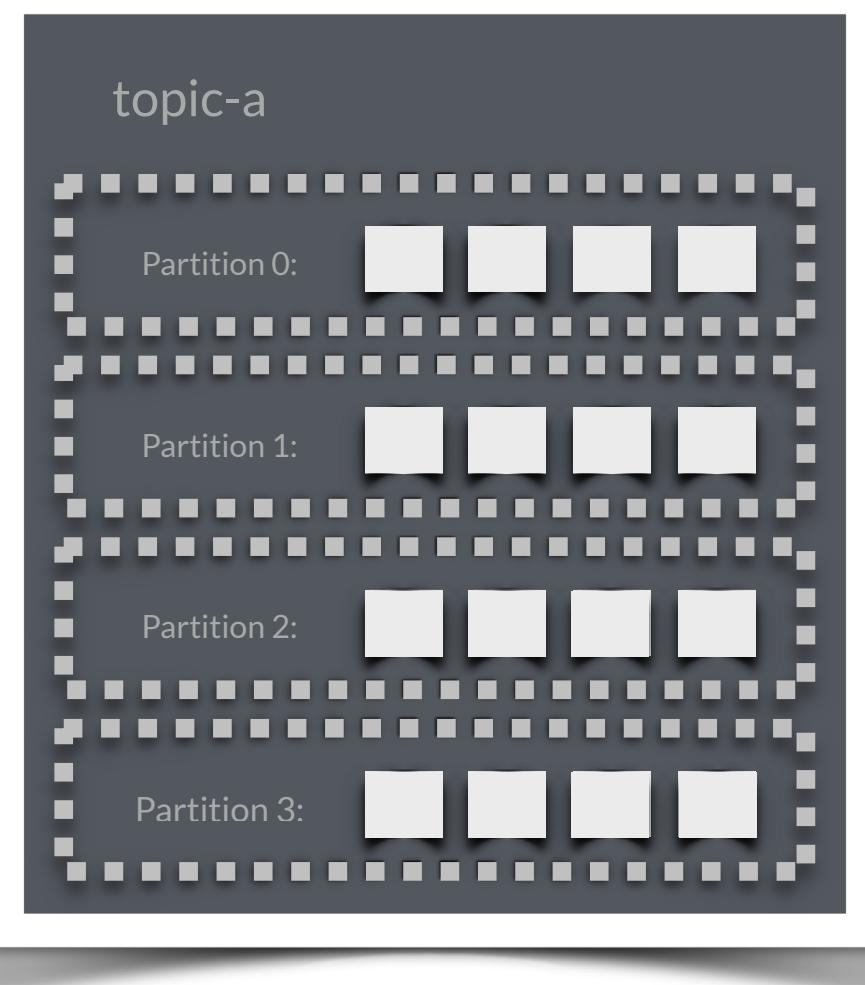




application.id = “my-stream”

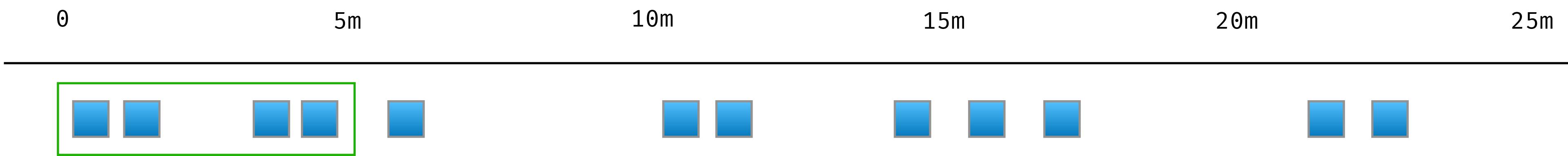


application.id = “my-stream”

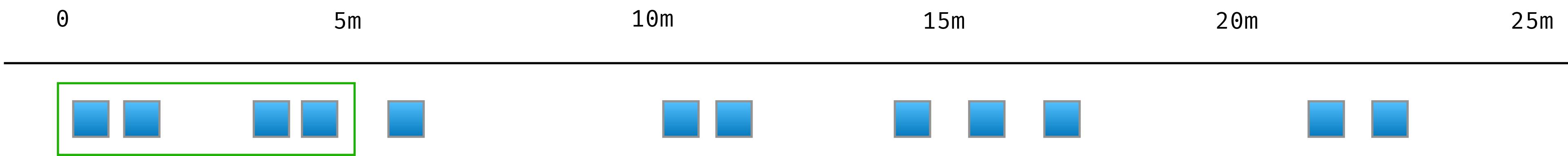


# Windowing

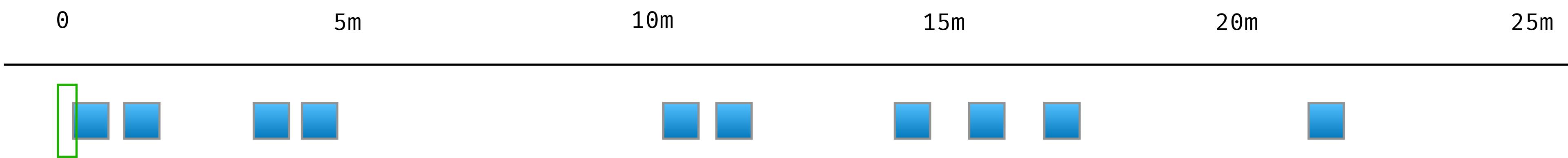
# Tumbling Window



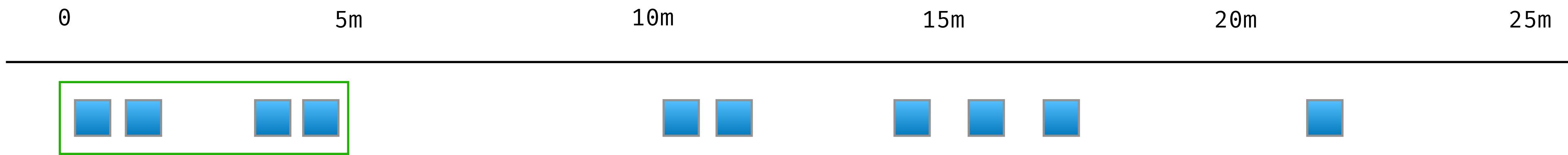
# Hopping Window



# Session Window

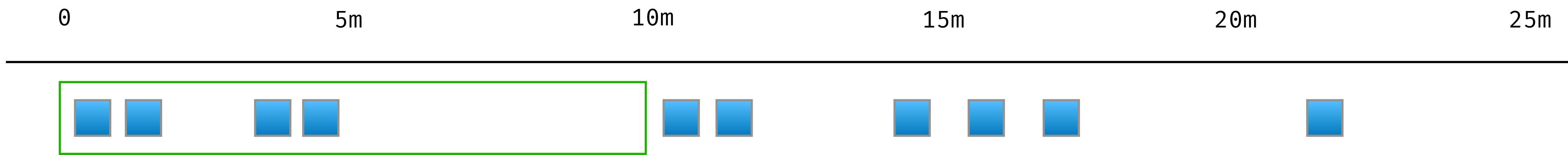


# Session Window



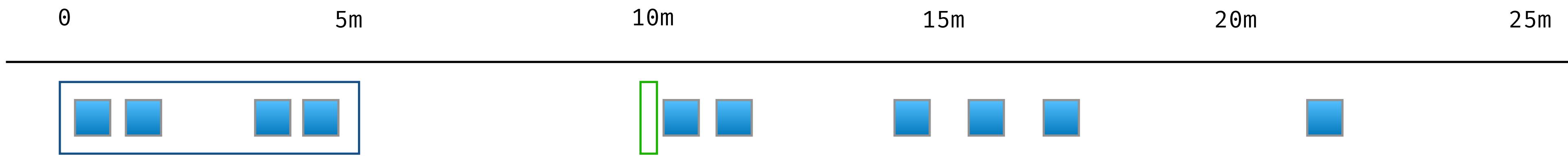
the last element has occurred; now we wait

# Session Window



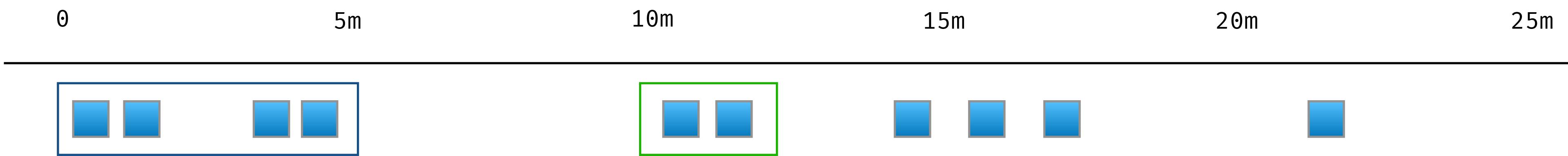
waited 5 minutes = make it a window

# Session Window



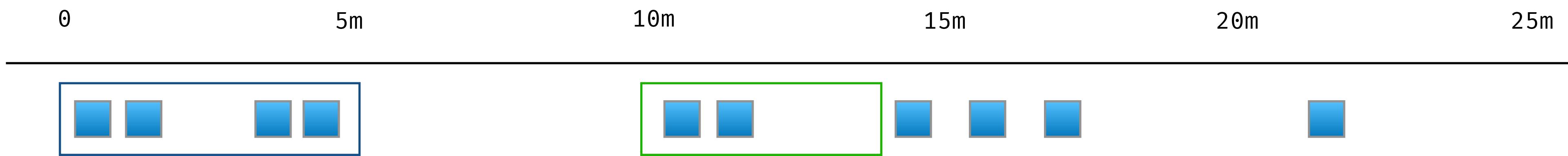
we have a new element; we start a new window

# Session Window



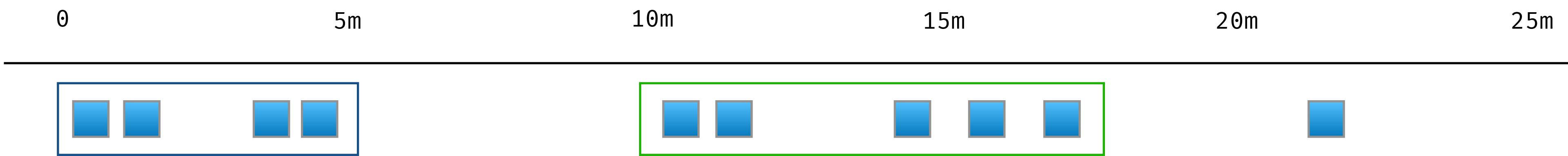
we wait for 5 minutes

# Session Window



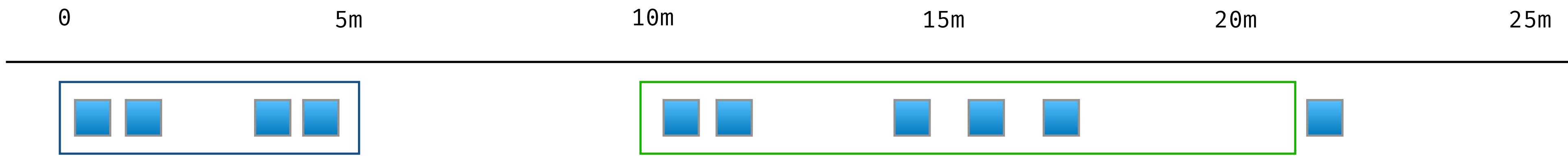
2 minutes elapsed; no windowing yet

# Session Window



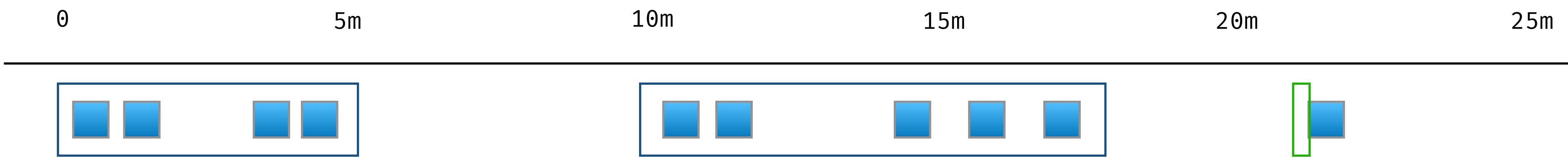
we wait five minutes

# Session Window



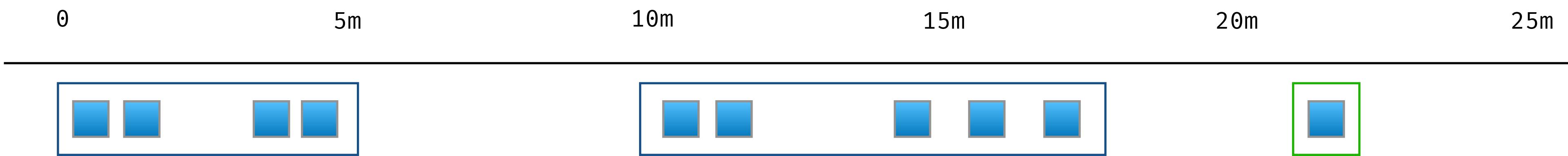
no new elements; we make it a window

# Session Window



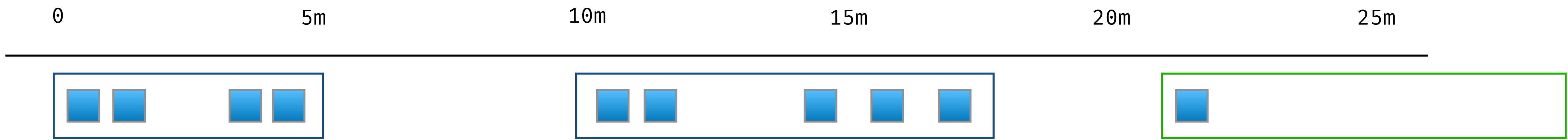
we see something new

# Session Window



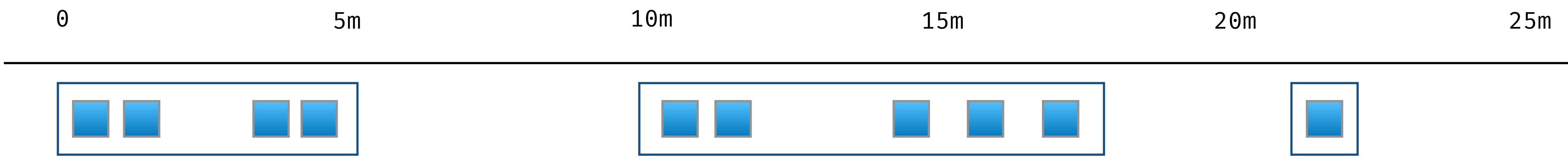
last element; we wait five minutes

# Session Window



nothing else; we wait 5 minutes

# Session Window



that's a new window

# Demo: Streams

# KSQldb

# KSQLDB

Open source streaming SQL engine for Apache Kafka

Provides SQL interface for stream processing on Kafka

Scalable, elastic, fault-tolerant, and real-time.

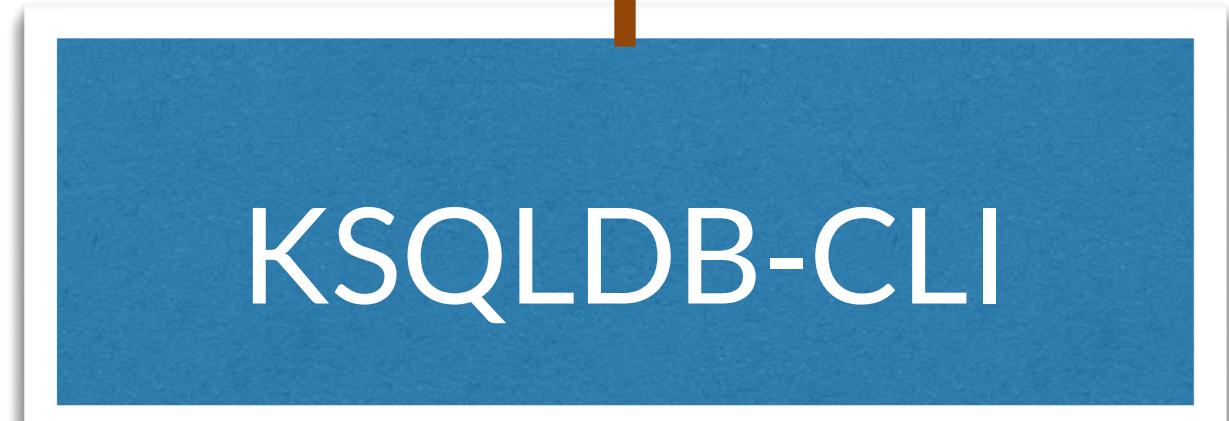
# KSQLDB

Supports a wide range of streaming operations, including:

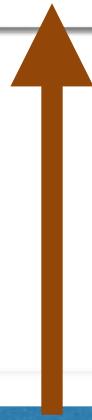
- Data filtering
- Transformations
- Aggregations
- Joins
- Windowing
- Sessionization



KSQLDB Server



KSQLDB-CLI



# KSQLDB Architecture

KSQLDB Server communicates with the broker

KSQLDB servers are run separately from the KSQLDB CLI client and Kafka brokers.

You can deploy servers on remote machines, VMs, or containers

KSQLDB CLI interacts with KSQLDB Servers via a REST API

KSQLDB Server runs in two modes: CLI, Headless

# Before you start...

Ensure that your data is structured and meaningful

That means, at the very least: *Structure your values*

*Recommended: Use Avro, Protobuf, or JSON Schema along with Schema Registry to validate your data*

KSQLDB

Kafka Streams

Producer/Consumer API

# Starting in CLI Mode

```
/bin/ksql http://localhost:8088
```

# Starting in Headless Mode

```
/bin/KSQLDB-start-server \  
server_path/etc/KSQLDB/KSQLDB-server.properties \  
--queries-file /path/to/queries.sql
```



# Tapping a Stream Based Topic

```
CREATE STREAM pageviews \
(viewtime BIGINT, \
userid VARCHAR, \
pageid VARCHAR) \
WITH (KAFKA_TOPIC='pageviews', \
VALUE_FORMAT='DELIMITED');
```

# Tapping a Table Based Topic

```
CREATE TABLE users \
  (registertime BIGINT, \
  userid VARCHAR, \
  gender VARCHAR, \
  regionid VARCHAR) \
WITH (KAFKA_TOPIC = 'users', \
      VALUE_FORMAT='JSON', \
      KEY = 'userid');
```

# Showing the Streams

**SHOW STREAMS;**

Stream Name	Kafka Topic	Format
PAGEVIEWS	pageviews	DELIMITED

# Showing the Tables

**SHOW TABLES;**

Table Name	Kafka Topic	Format
USERS	users	JSON

# Supported SQL Types

- BOOLEAN
- INTEGER
- BIGINT
- DOUBLE
- VARCHAR (or STRING)
- ARRAY<ArrayType> (JSON and AVRO only. Index starts from 0)
- MAP<VARCHAR, ValueType> (JSON and AVRO only)
- STRUCT<FieldName FieldType, ... > (JSON and AVRO only)

# Various Operators

## Scalar Operators:

ABS, ARRAYCONTAINS, CEIL, CONCAT,  
EXTRACTJSONFIELD, FLOOR, GEO\_DISTANCE, IFNULL, LC  
ASE, LEN, MASK, RANDOM, ROUND,  
STRINGTOTIMESTAMP, SUBSTRING, TIMESTAMPTOSTRING,  
TRIM, UCASE

## Aggregator Operators:

COUNT, MAX, MIN, SUM, TOPK, TOPKDISTINCT, WINDOWS  
TART, WINDOWEND

# Displays Information

SHOW TOPICS	Show all topics
SHOW <TOPIC>	Display Topic
SHOW STREAMS	Show Streams
SHOW TABLES	Show Tables
SHOW FUNCTIONS	Show Available Functions
SHOW QUERIES	Show Persistent Queries

# Describing

**DESCRIBE**

List columns in stream or table

**DESCRIBE EXTENDED**

Describe in detail stream or table; runtime statistics, and queries that populate the stream or table

# Non-Persistent Pull Query

```
SELECT * FROM pageviews  
WHERE ROWTIME >= 1510923225000  
AND ROWTIME <= 1510923228000;
```

A **LIMIT** can be used to limit the number of rows returned.  
Once the limit is reached the query will terminate. This is a  
pull query since we are want a one time pull of an exact set

# Push Query

```
SELECT * FROM pageviews EMIT  
CHANGES;
```

The `EMIT CHANGES` is used for Push Queries. When there are new results, they will *pushed* to you. This is the only time, a push exists in the Kafka ecosystem.

# Persistent Query

```
CREATE TABLE users_female AS \  
SELECT userid, gender, regionid FROM users \  
WHERE gender='FEMALE' EMIT CHANGES;
```

What makes this a persistent query is the form

**CREATE (TABLE | STREAM) AS SELECT**

Once executed, it will continuously run, until terminated  
with **TERMINATE** command

# Persistent v. Non Persistent

	<b>Persistent</b>	<b>Non Persistent</b>
Where does data go?	Will store into a topic	Will display on screen
How do I stop it?	Find query id using SHOW QUERIES and TERMINATE <id>	CTRL + C
How to create?	CREATE STREAM AS SELECT ...	SELECT ....

# Selecting Customized Fields

```
CREATE STREAM pageviews_transformed \
    WITH (TIMESTAMP='viewtime', \
          PARTITIONS=5, \
          VALUE_FORMAT='JSON') AS \
    SELECT viewtime, \
           userid, \
           pageid, \
           TIMESTAMPTOSTRING \
               (viewtime, 'yyyy-MM-dd HH:mm:ss.SSS') \
               AS timestamp \
    FROM pageviews \
    PARTITION BY userid \
    EMIT CHANGES;
```

# Start from the Beginning

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

# Tumbling Window

```
SELECT item_id, SUM(quantity)  
FROM orders  
WINDOW TUMBLING (SIZE 20 SECONDS)  
GROUP BY item_id;
```

# Hopping Window

```
SELECT item_id, SUM(quantity)
FROM orders
WINDOW HOPPING (SIZE 20 SECONDS,
                  ADVANCE BY 5 SECONDS)
GROUP BY item_id;
```

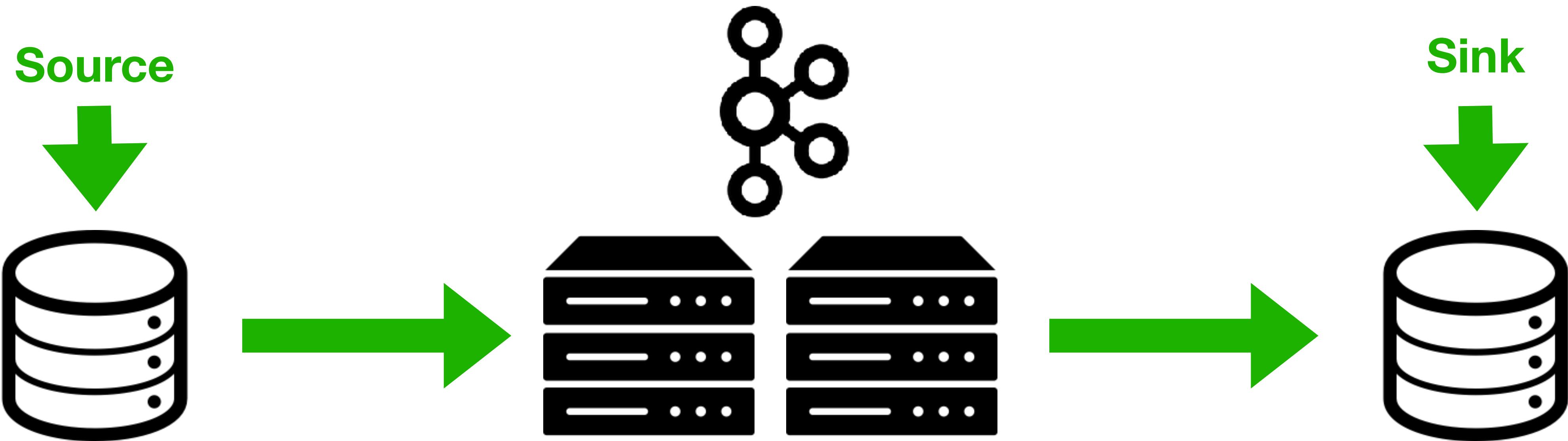
# Session Window

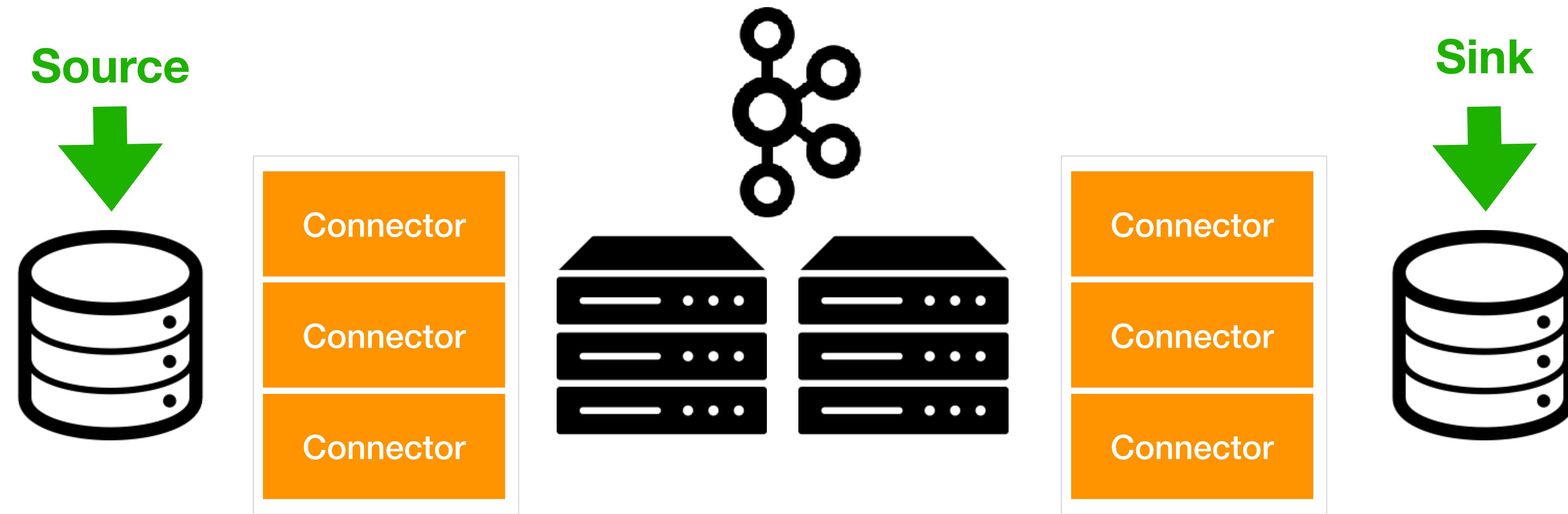
```
SELECT item_id, SUM(quantity)  
FROM orders  
WINDOW SESSION (20 SECONDS)  
GROUP BY item_id;
```

# Kafka Connect

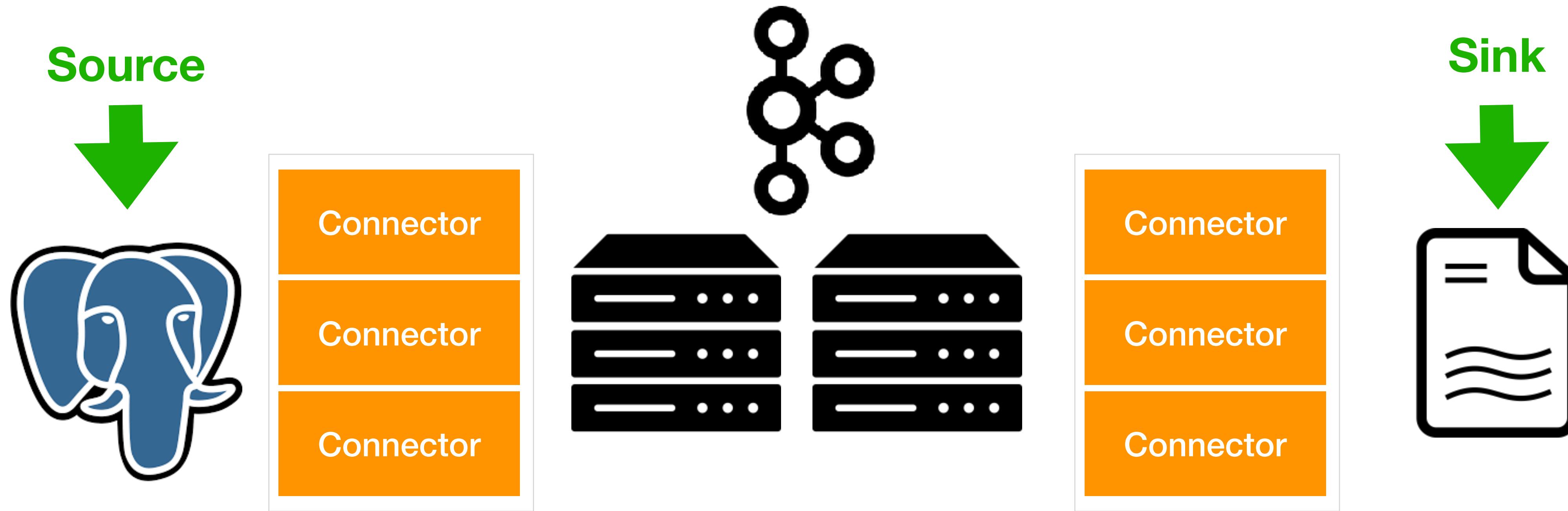
# Kafka Connect

- Automatic Scaling
- Fault Tolerance
- All Configuration
- Pre-Engineered
- Transformations
- Confluent Community License

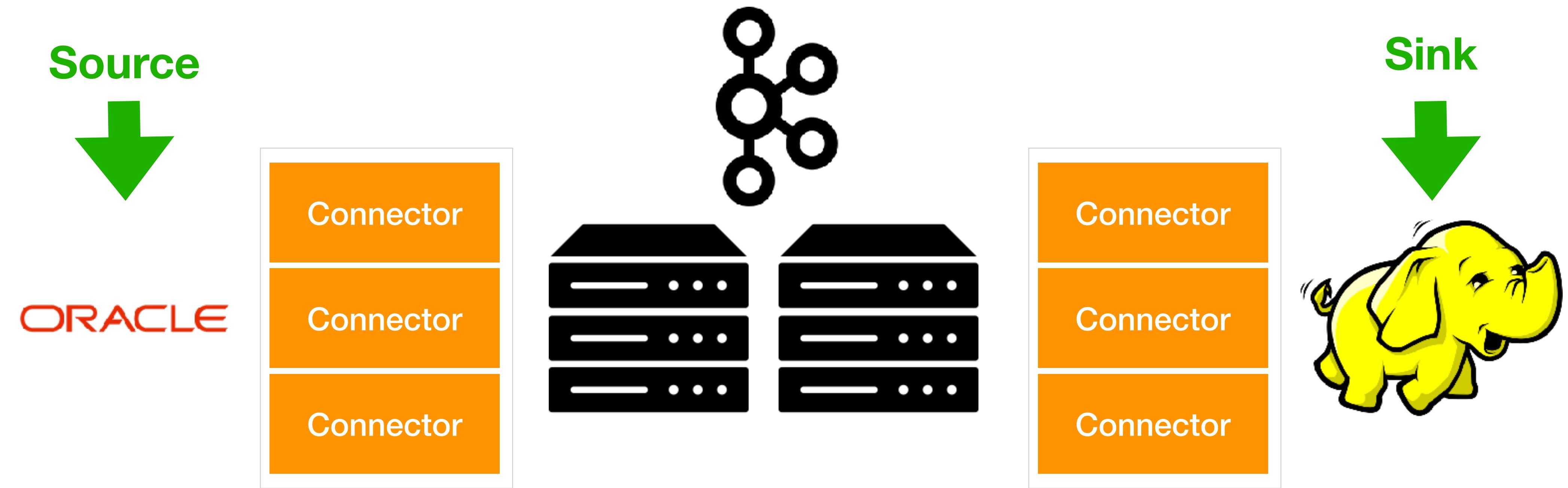




**Kafka Connect is an Open Source Framework to stream data to and from Kafka**



**Connect a PostgreSQL database to a single file**



**Connect an Oracle database to Hadoop**

[Product](#)[Cloud](#)[Developers](#)[Blog](#)[Docs](#)[Download](#)

Verified Standard

**CockroachDB Change  
Data Capture**

Cockroach Labs

[Read More](#)

Verified Standard

**Vertica Analytics  
Platform**

Vertica

[Read More](#)

Confluent Supported

**Debezium MySQL CDC  
Connector**

Debezium Community

[Read More](#)

Confluent Supported

**Debezium PostgreSQL  
CDC Connector**

Debezium Community

[Read More](#)

Confluent Supported

**Debezium SQL Server  
CDC Connector**

Debezium Community

[Read More](#)

Confluent Supported

**Debezium MongoDB  
CDC Connector**

Debezium Community

[Read More](#)

Confluent Supported

**Kafka Connect ActiveMQ  
Source**

Confluent, Inc.

[Read More](#)

Confluent Supported

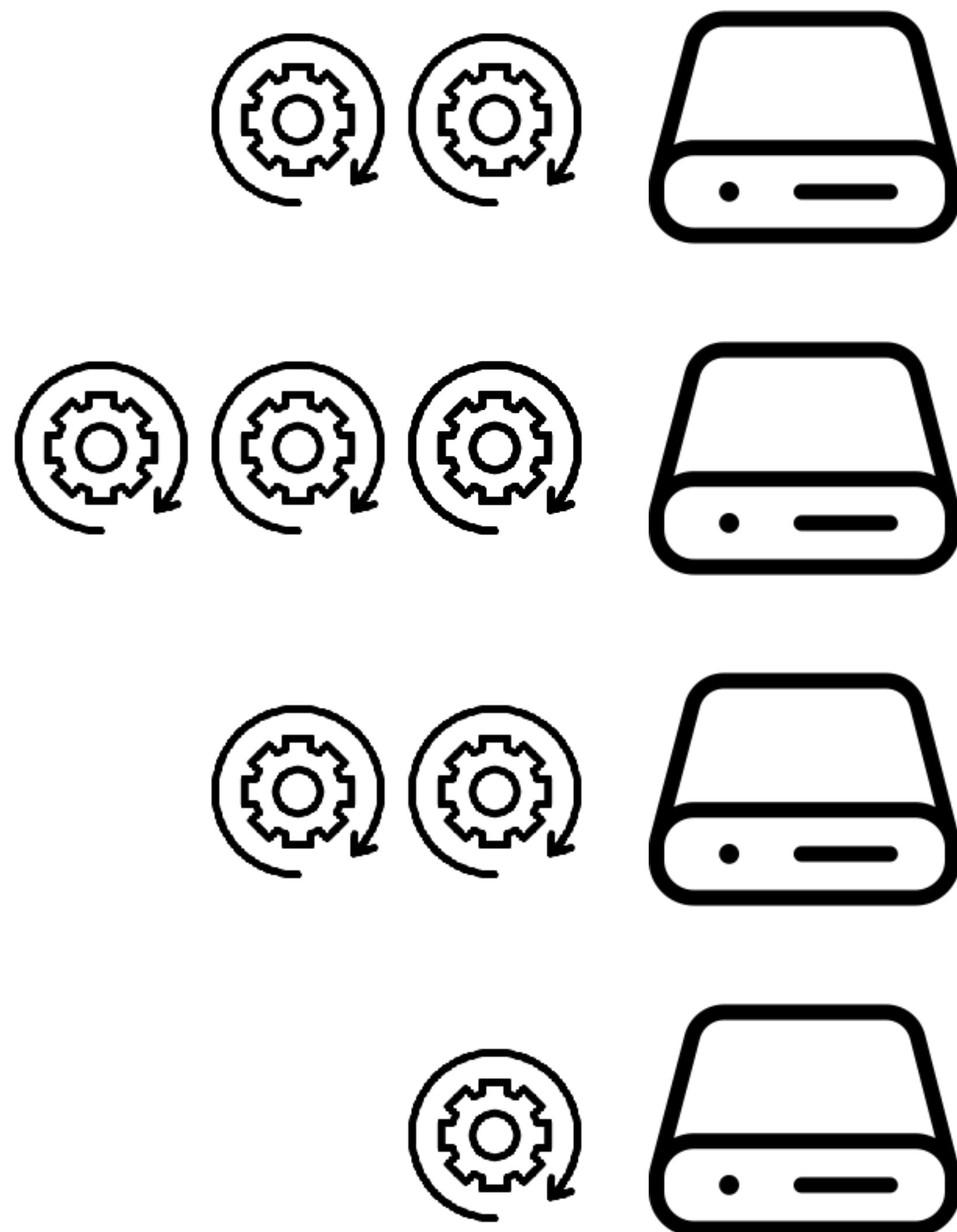
**Kafka Connect AWS  
CloudWatch Logs Source  
Connector**

Confluent, Inc.

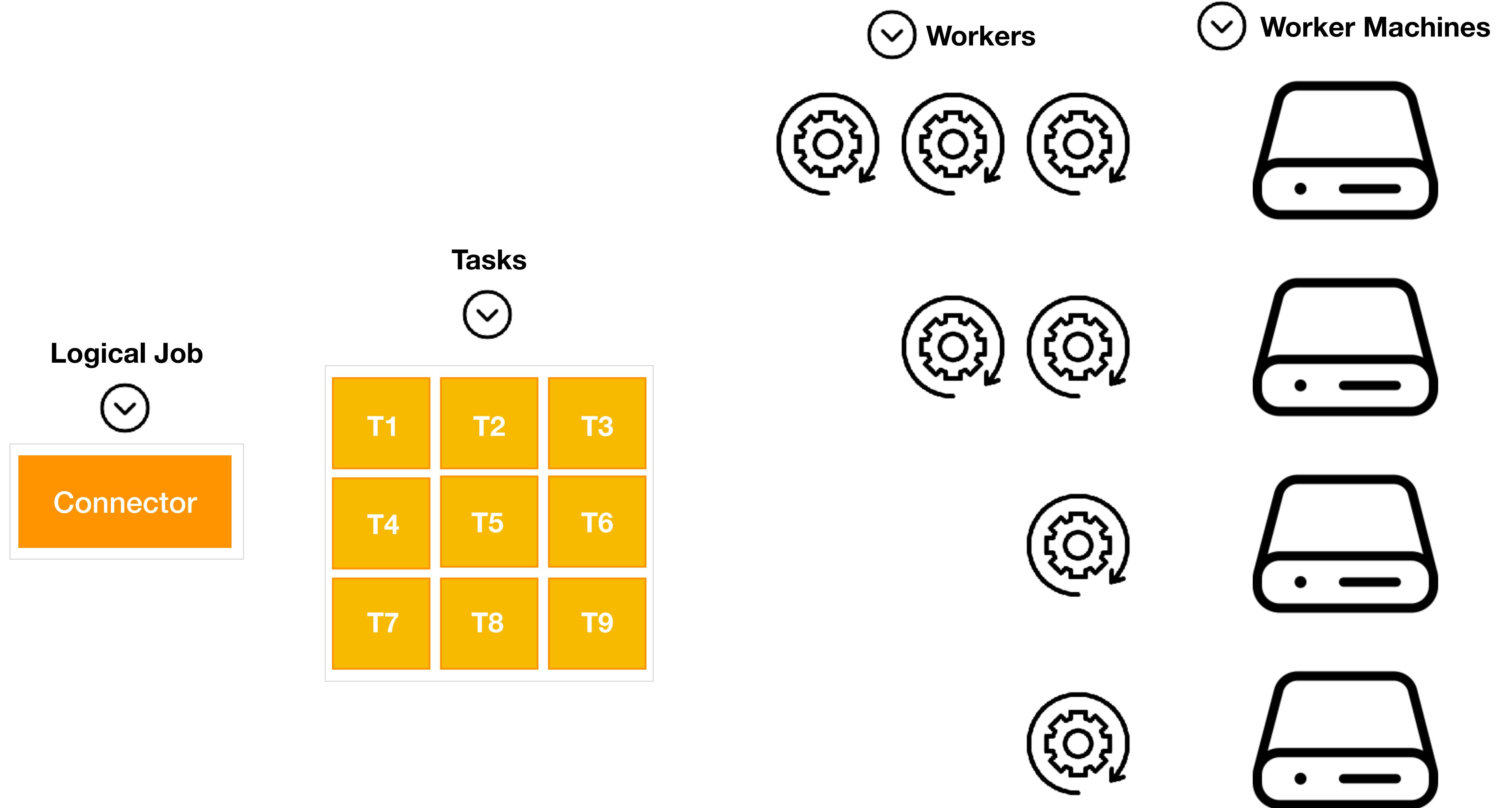
[Read More](#)<https://confluent.io/hub>

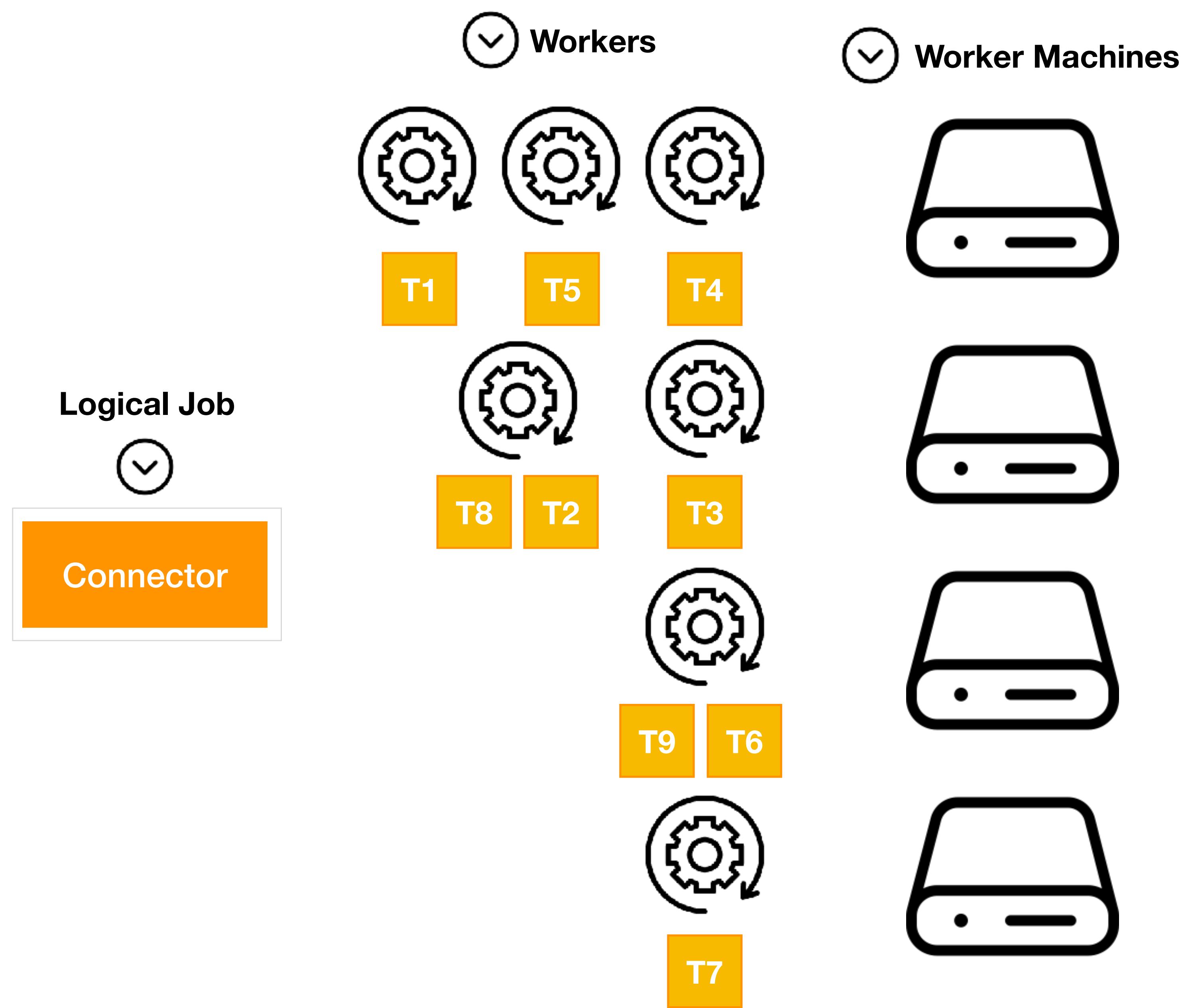
# Architecture

**Workers** >



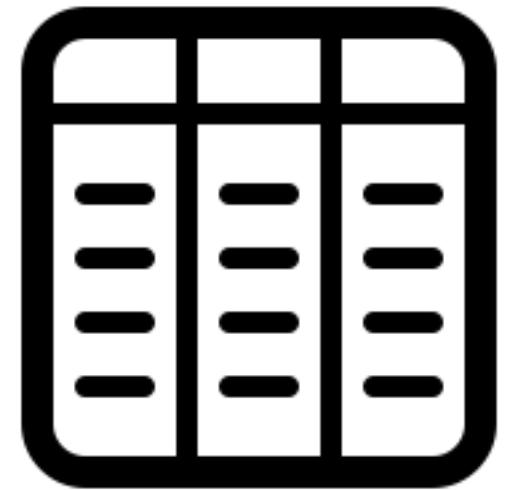
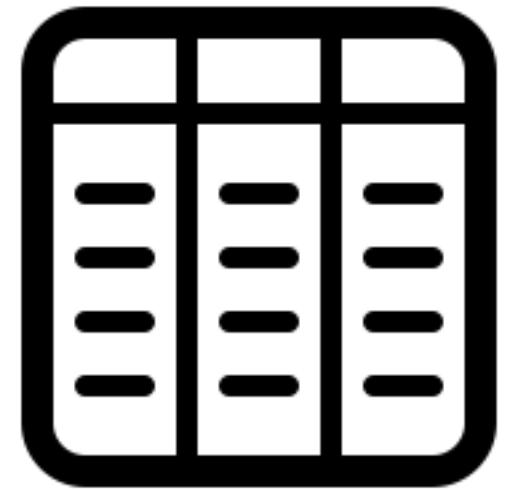
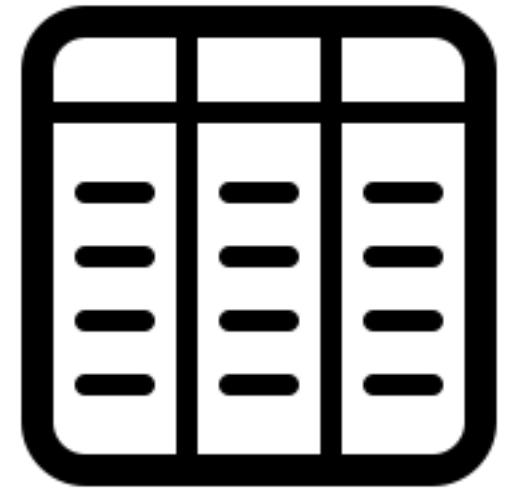
< **Worker Machines**





# Source Architecture

## Database Table to Kafka Topic



Tasks



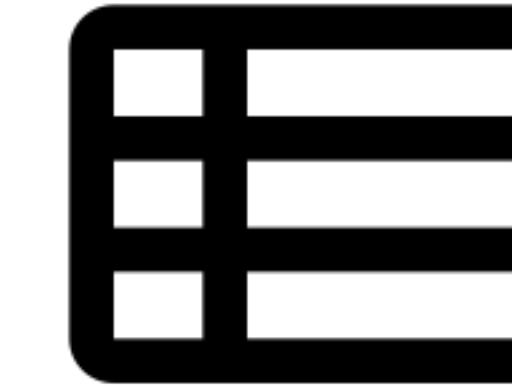
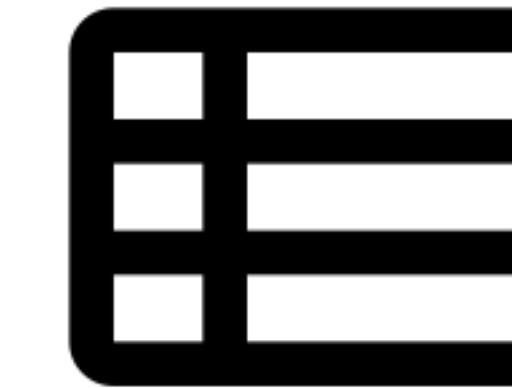
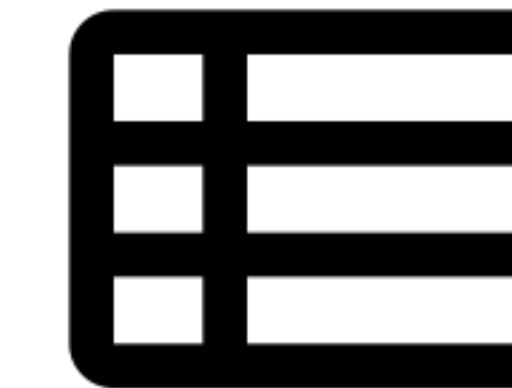
T2



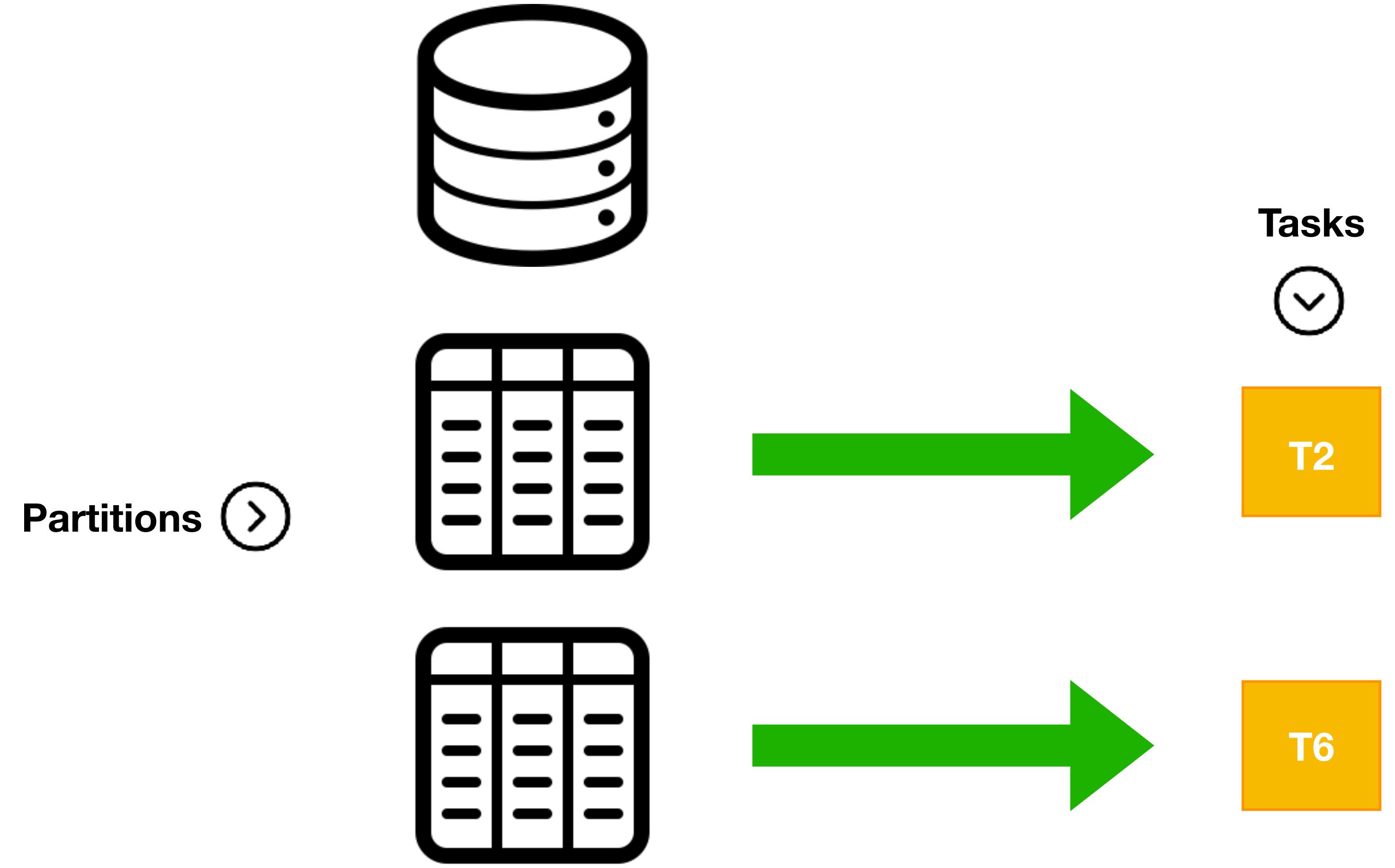
T6



T9



Topics

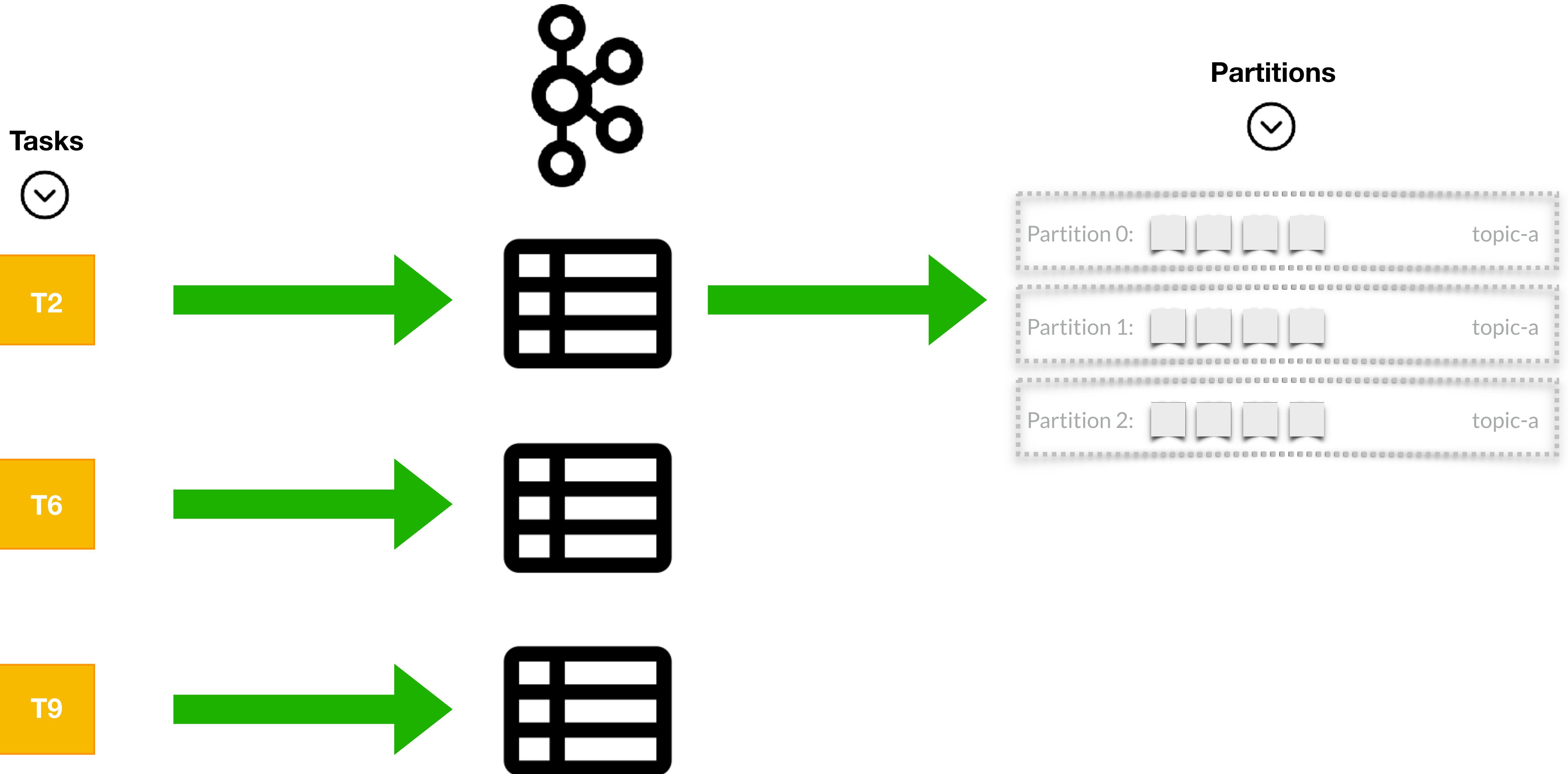


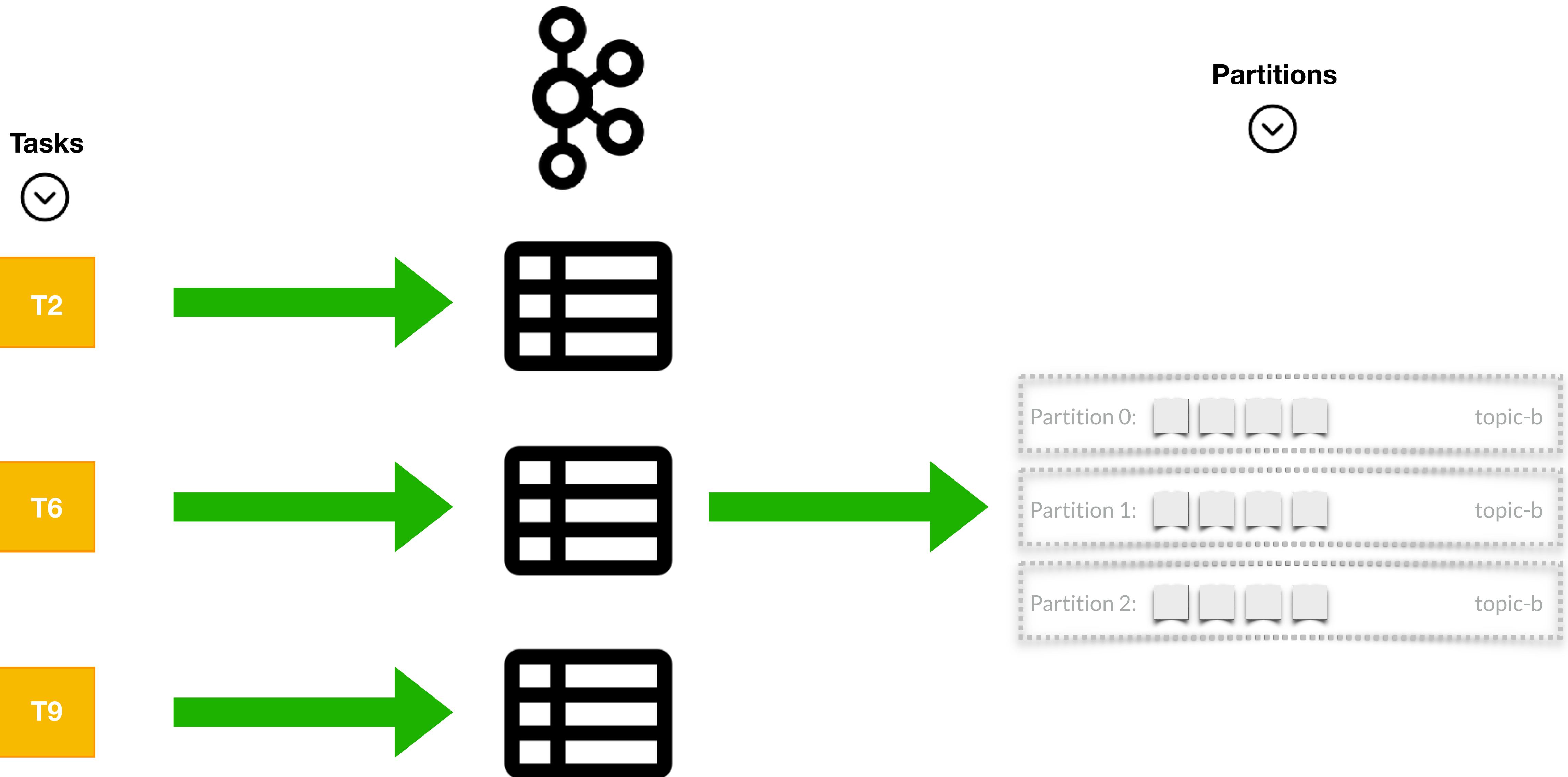
**Partitions** 

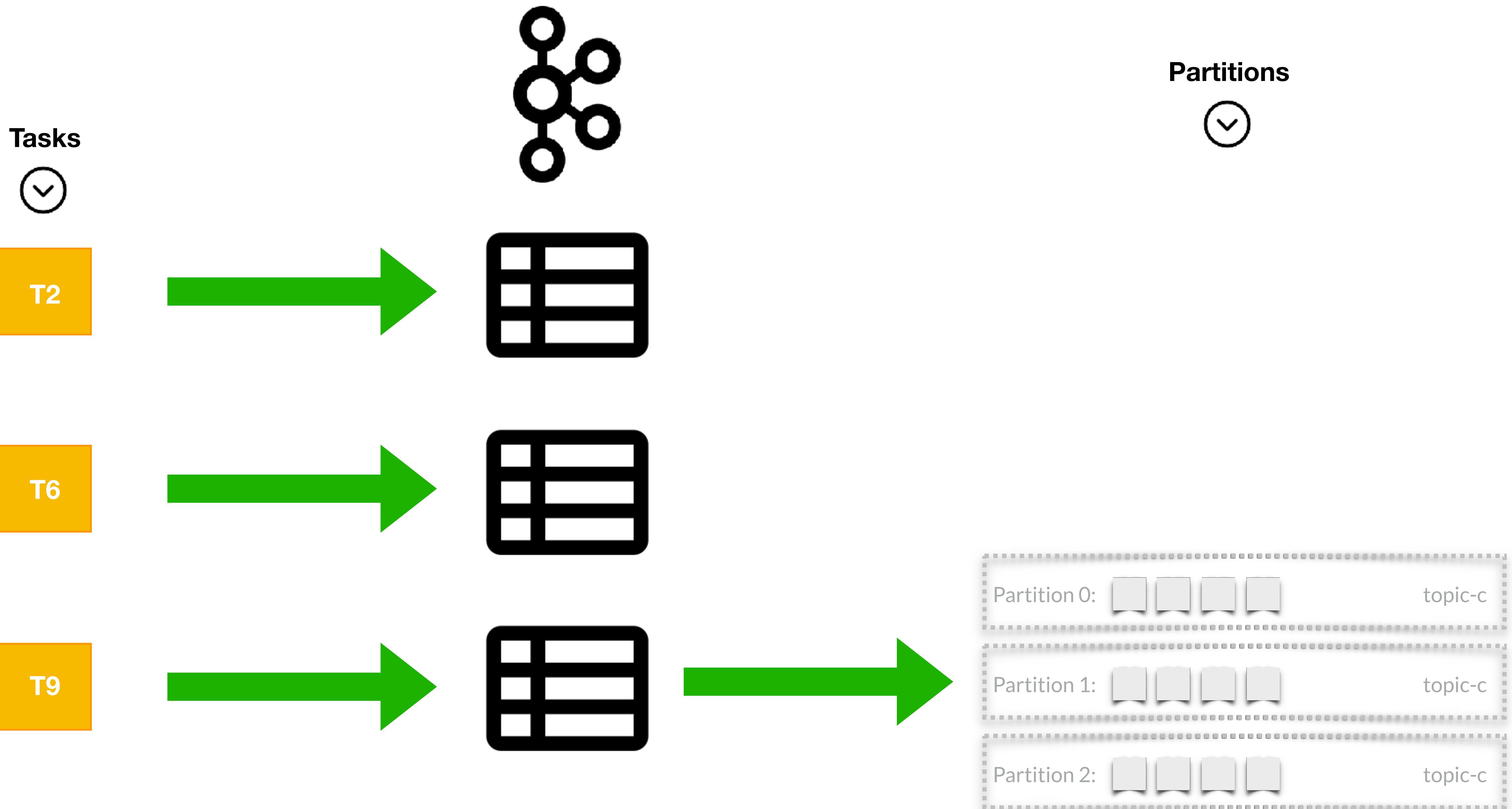


**Tasks**

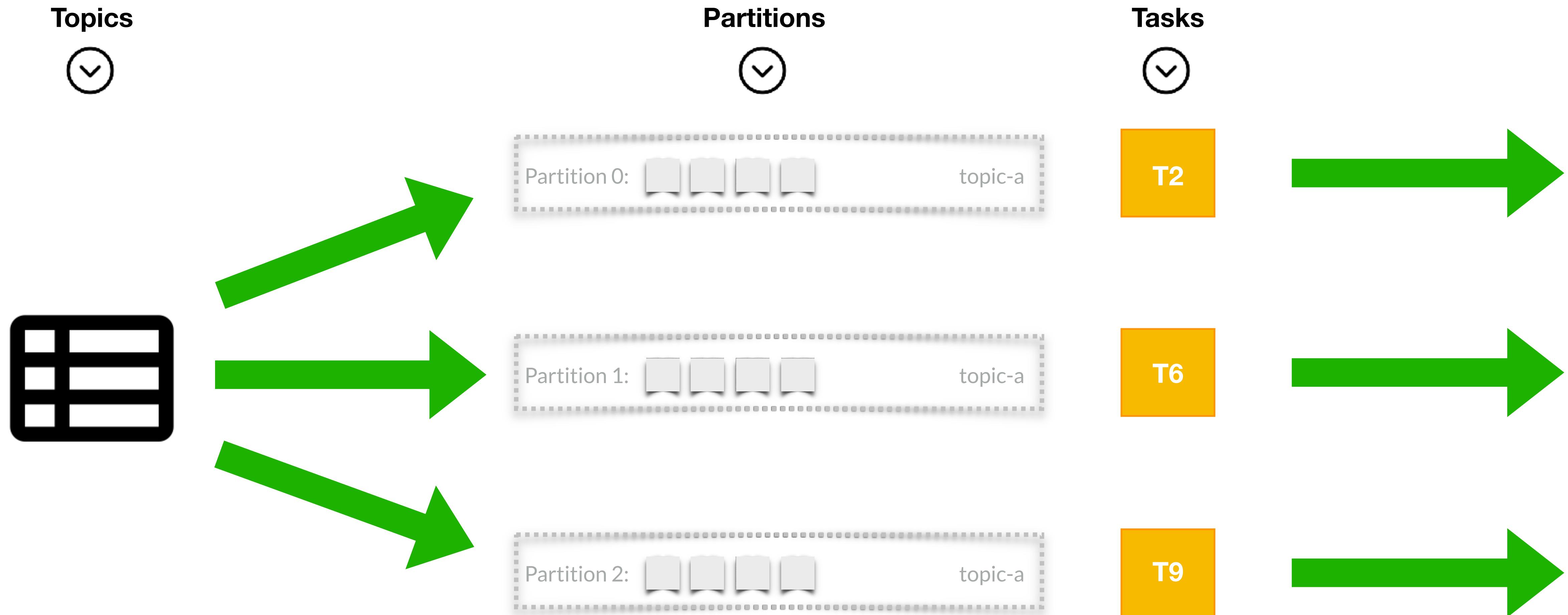


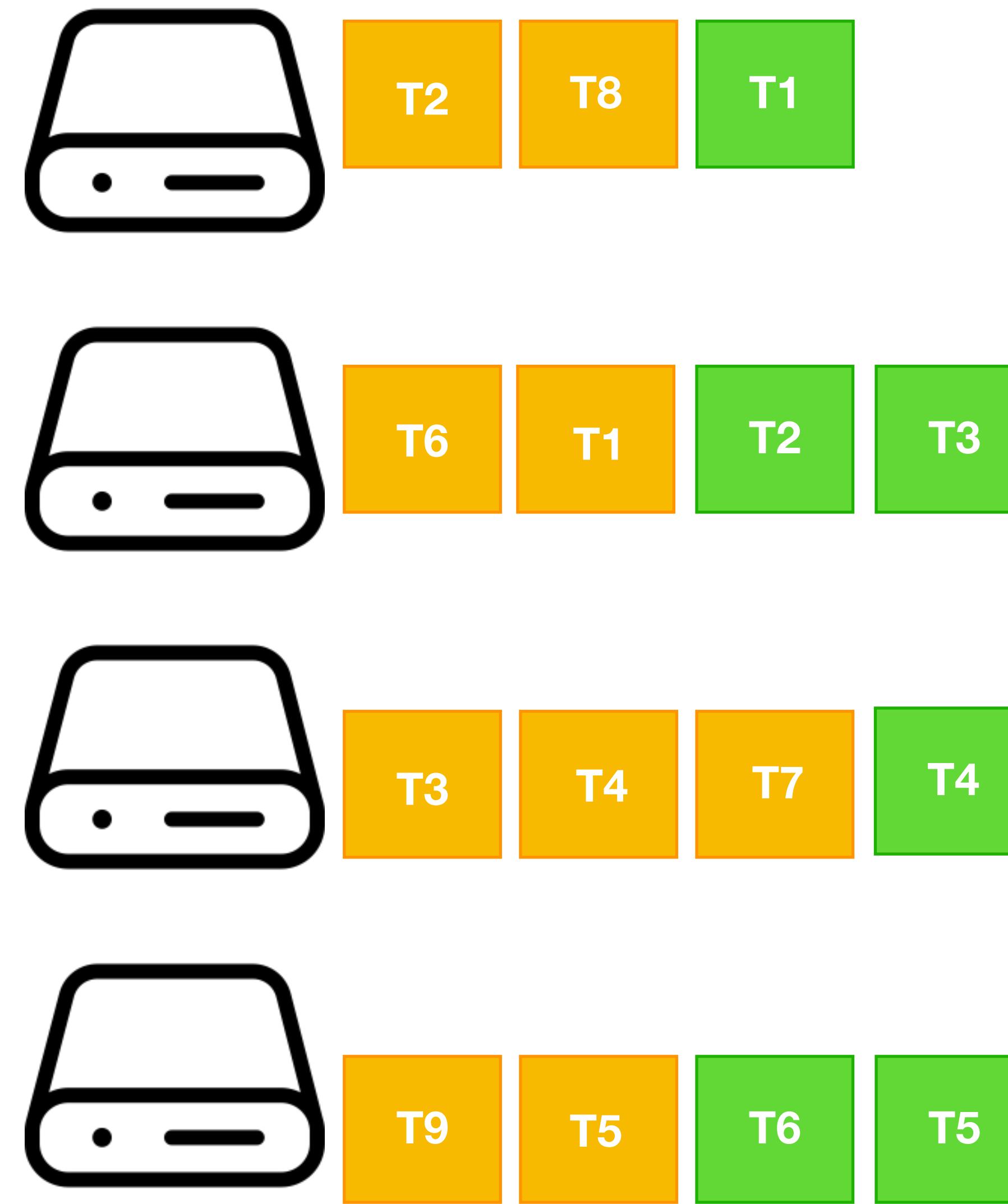






# Sink Architecture





# Demo: Kafka Connect & KSQLDB

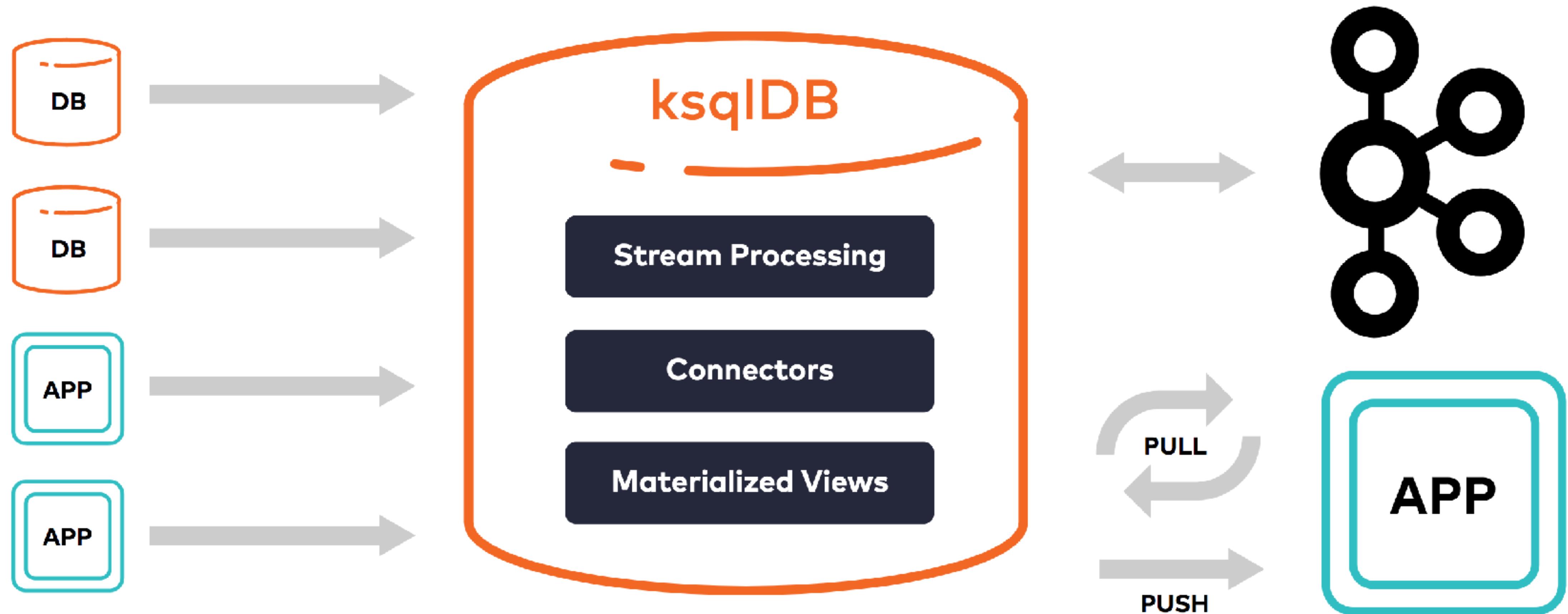
Wait! What about  
the DB in KSQLDB?

# Setting up a Connector through KSQLDB

Connectors (connections to data stores through Kafka Connect) can be established within KSQLDB

This makes it perfectly suitable for Data Engineers and Data Scientists to specify a source data store and a sink data store and start creating stream applications

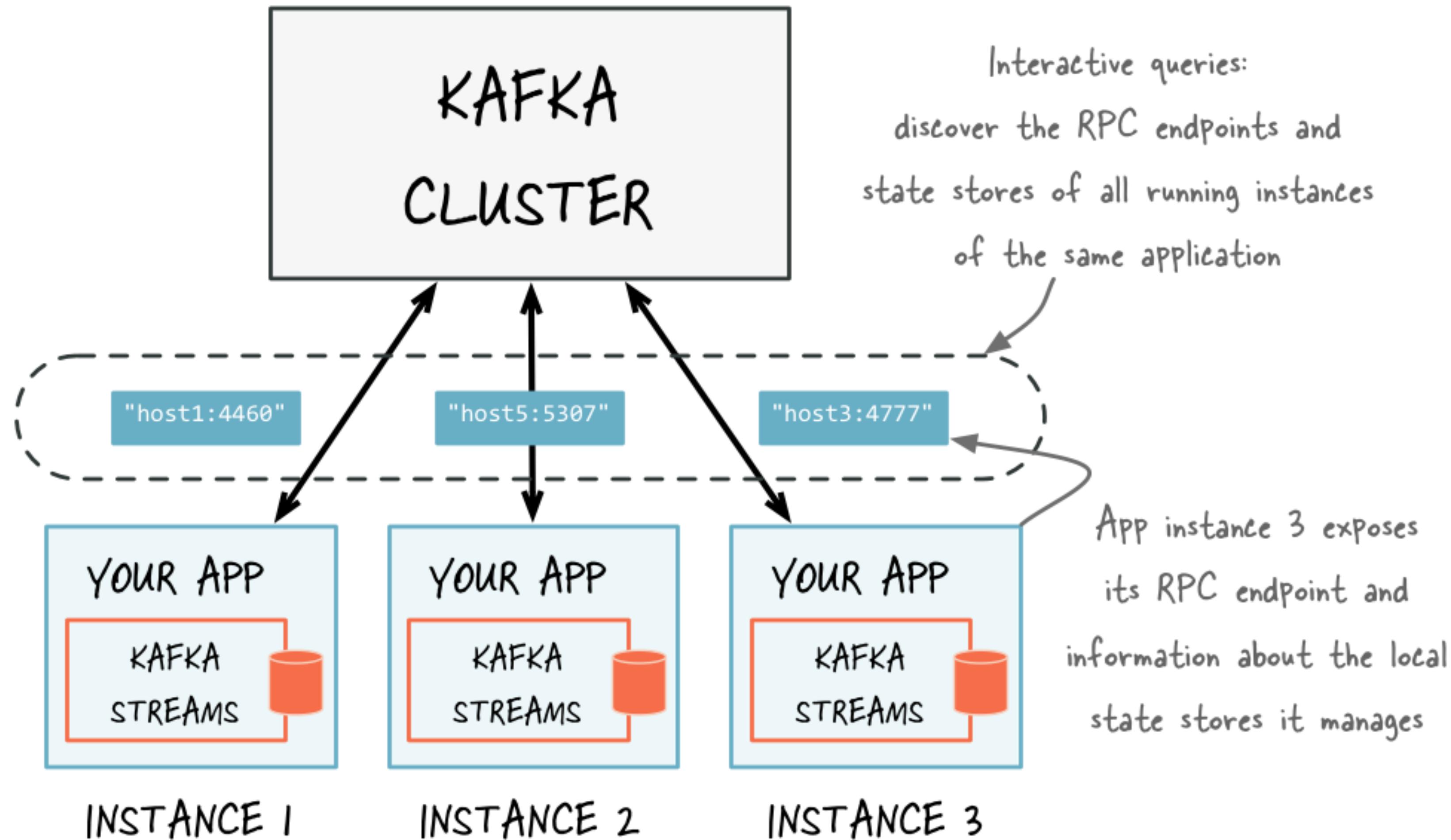
Requires connector instances deployed and configured



<https://ksqldb.io/>

<https://developer.confluent.io/tutorials/discount-promo/confluent.html>

# What are Interactive Queries?



# Conclusion

# These Technologies Solve Multiple Architectural Patterns

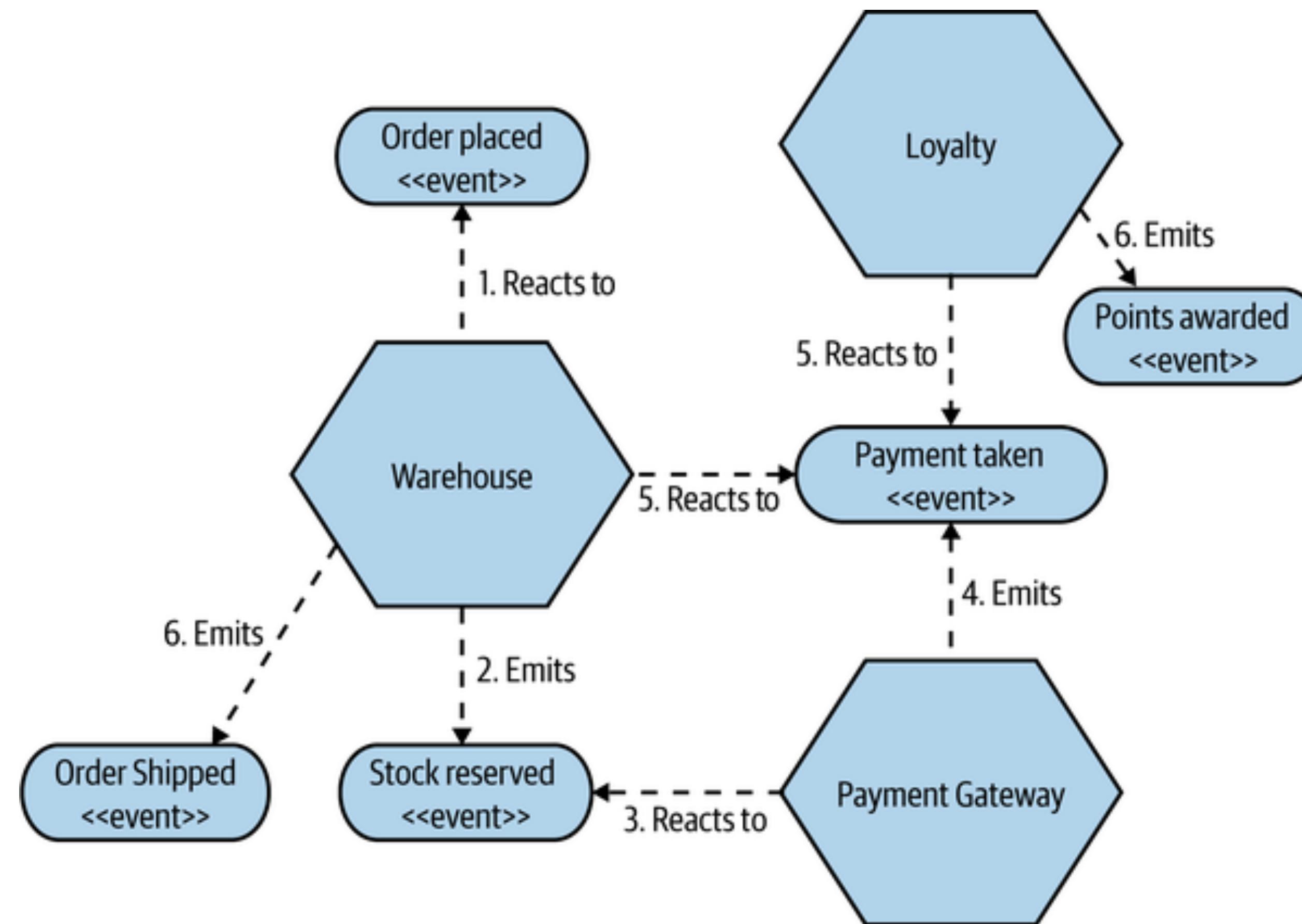
**Saga Pattern** - Streaming or KSQL

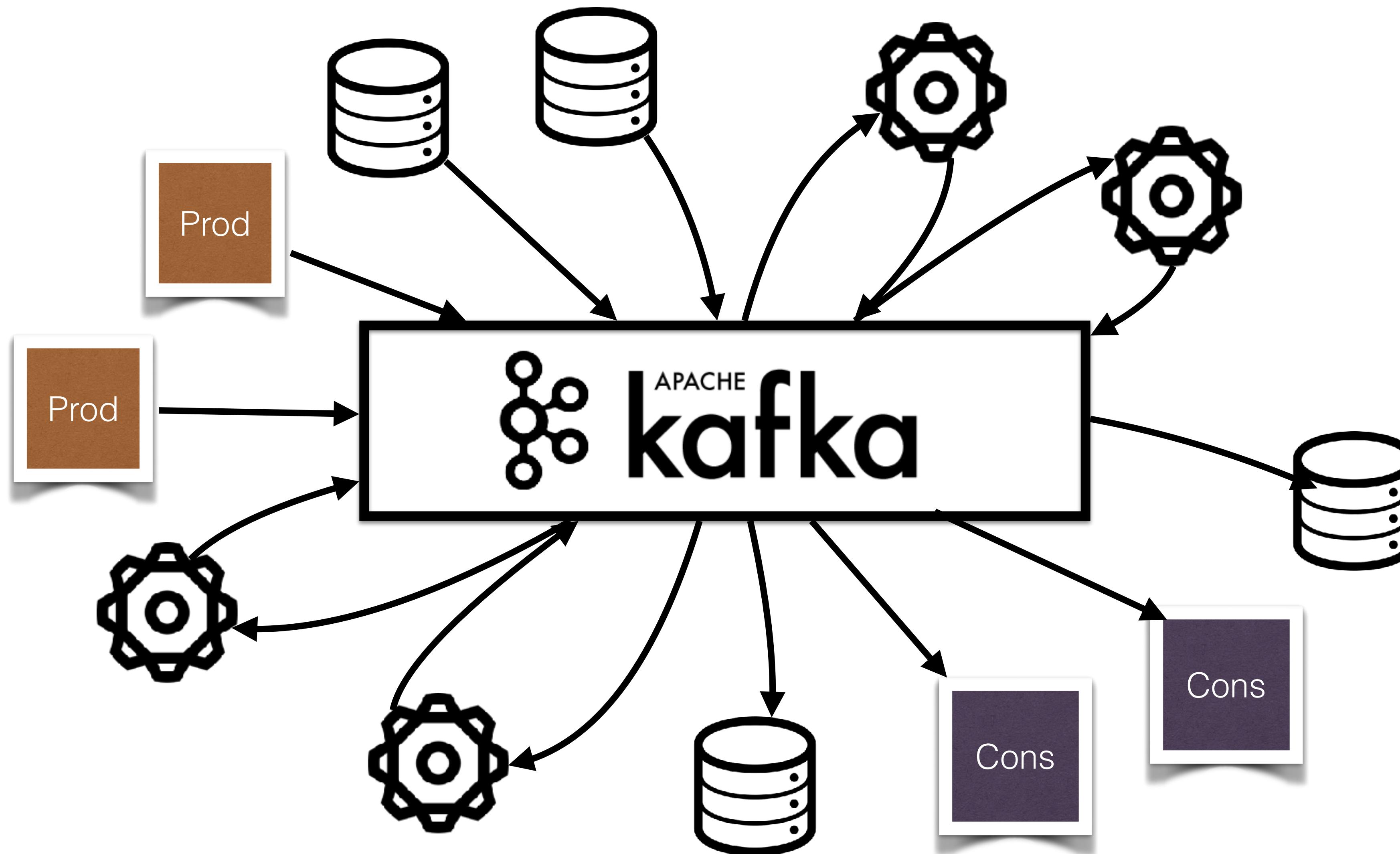
**CQRS** - Connect Source through KSQLDB to Connect Sink

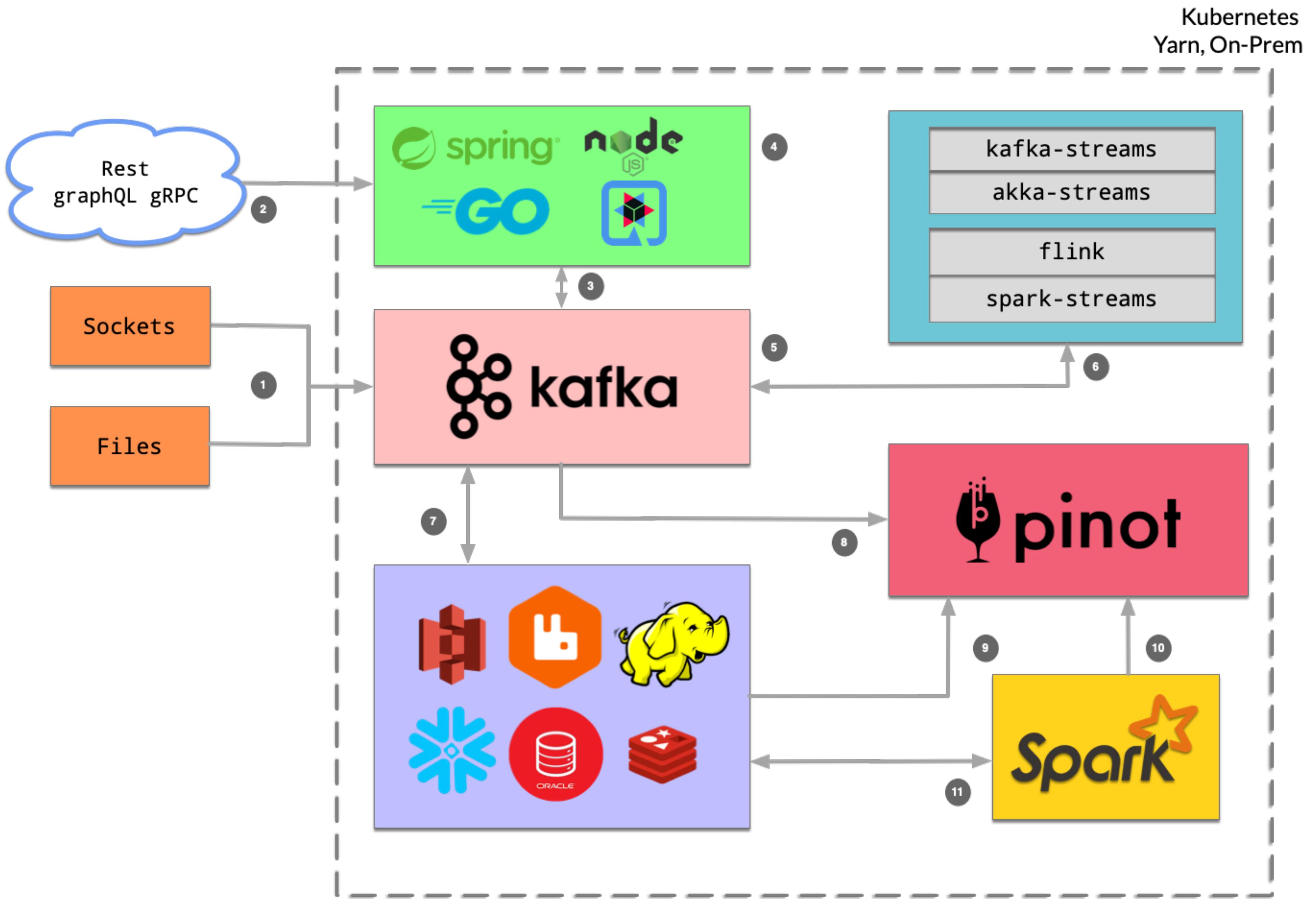
**Outbox Pattern** - Kafka Connect Source to Kafka

**Event Sourcing** - Everything Kafka

# Choreographed Sagas







# Thank You

**Daniel Hinojosa  
Programmer, Consultant,  
Trainer**

[dhinojosa@evolutionnext.com](mailto:dhinojosa@evolutionnext.com)

[https://mastodon.social/  
@dhinojosa](https://mastodon.social/@dhinojosa)

[https://www.linkedin.com/in/  
dhevolutionnext/](https://www.linkedin.com/in/dhevolutionnext/)

