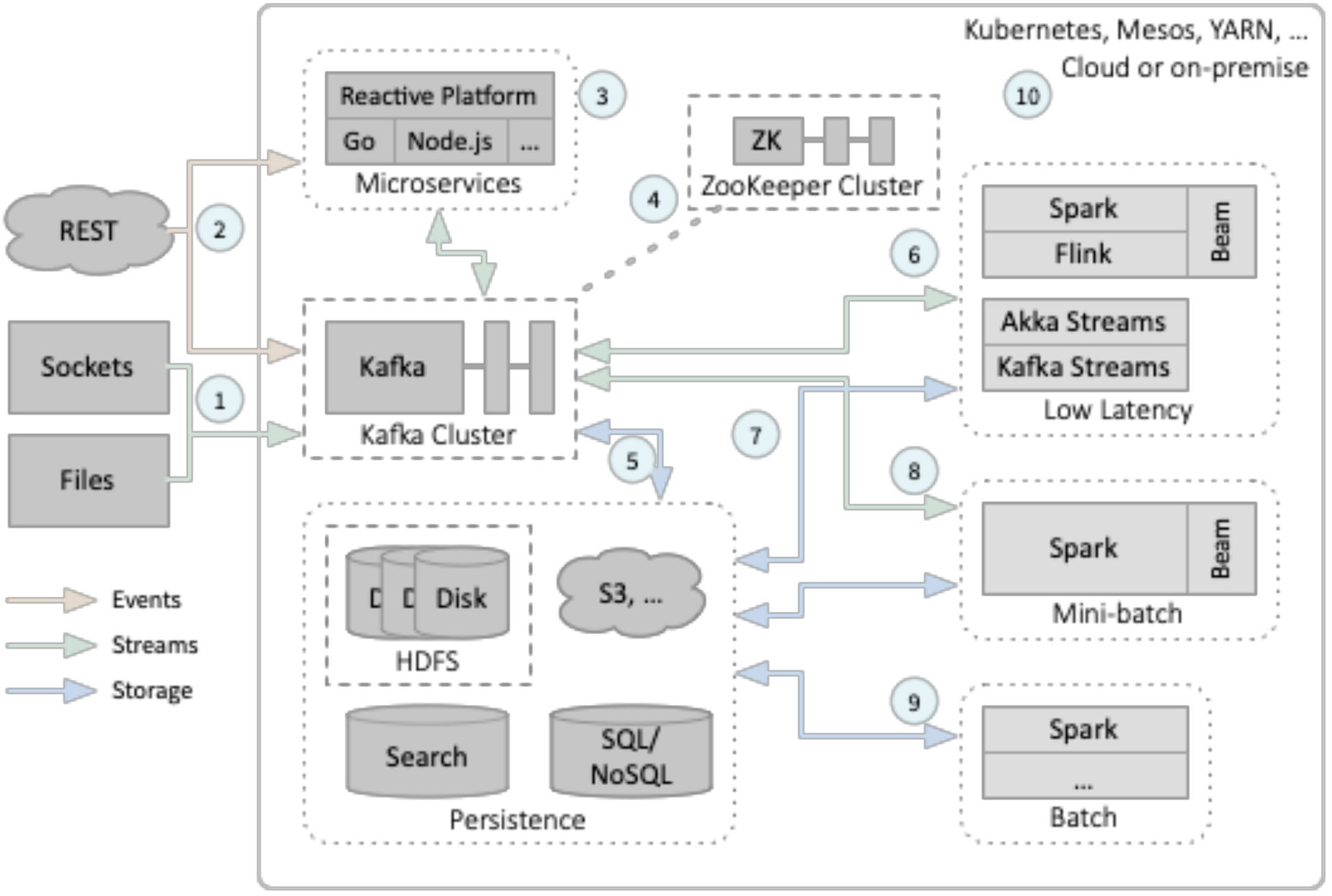
In Depth Kafka Streams

Daniel Hinojosa

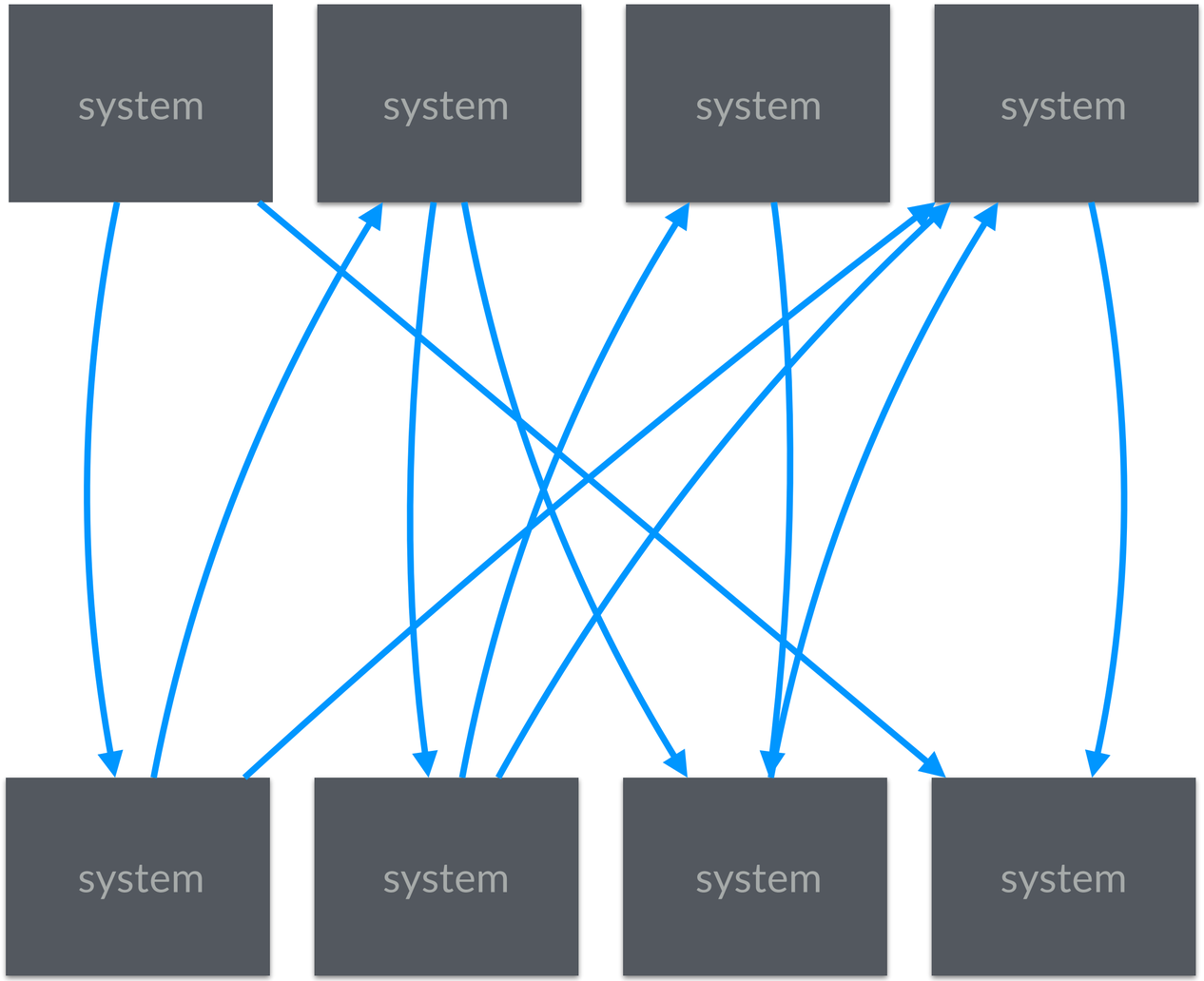
About Me...

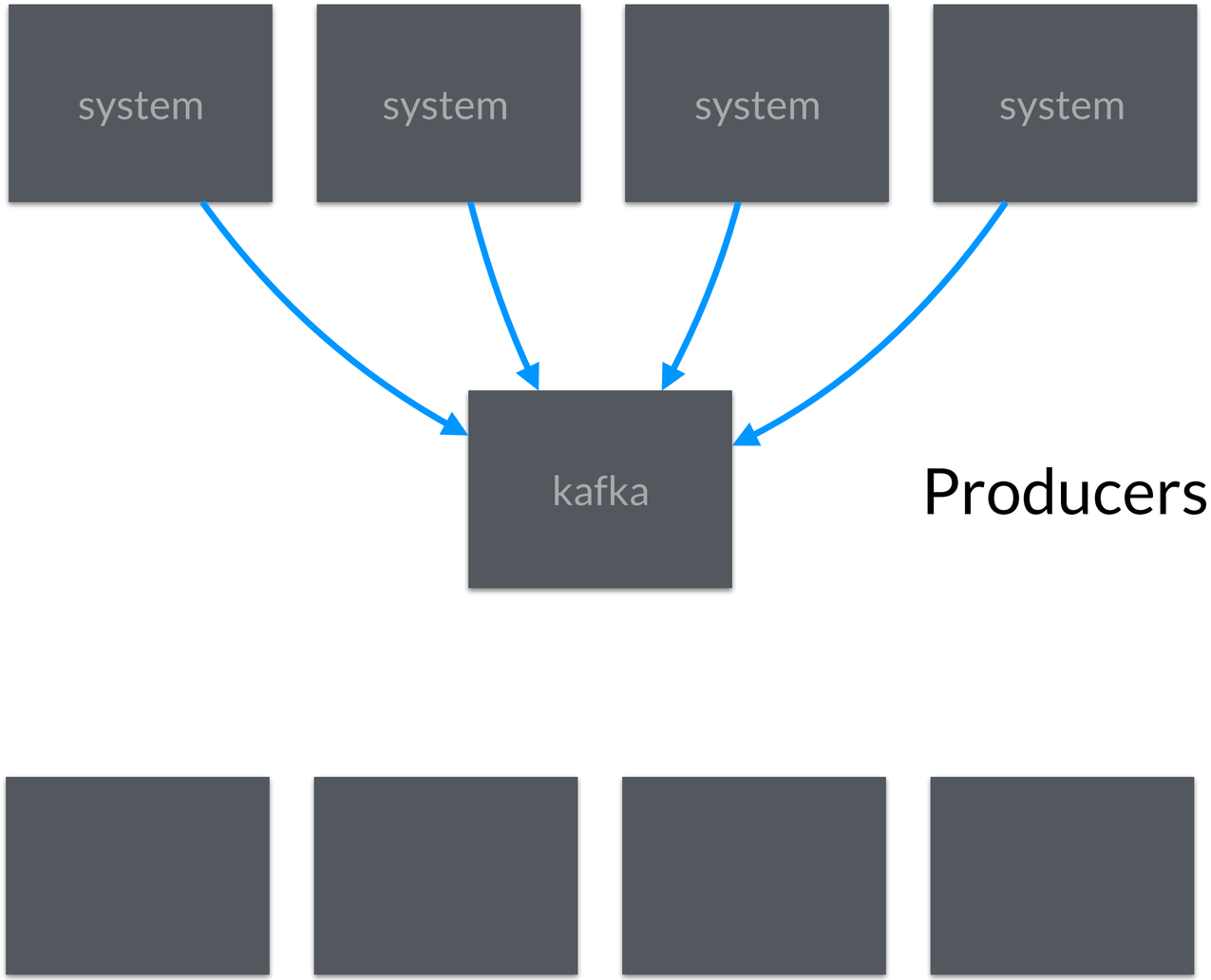
Daniel Hinojosa
Programmer, Consultant, Trainer

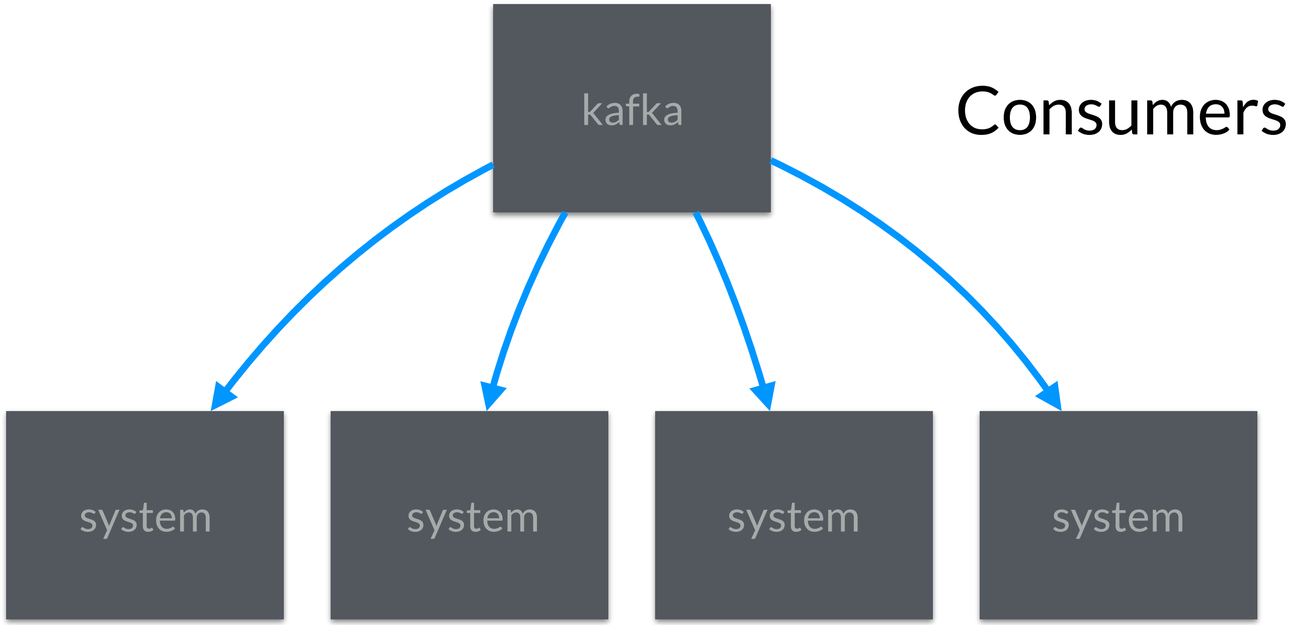
dhinojosa@evolutionnext.com
@dhinojosa

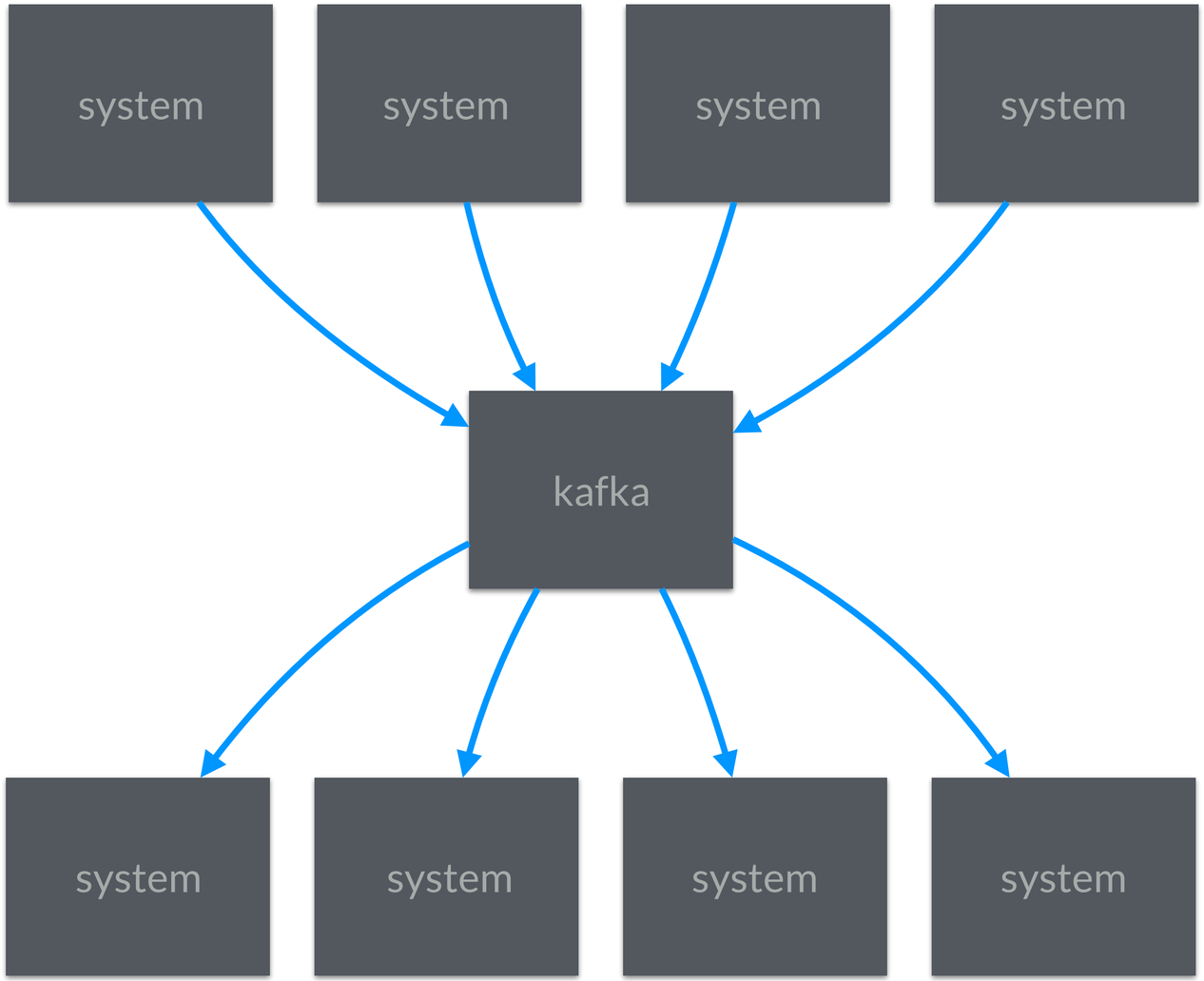

Kubernetes, Mesos, YARN, ...
Cloud or on-premise
10
Reactive Platform
Go
Node.js
...
Microservices
3
ZK
ZooKeeper Cluster
4
REST
2
Sockets
1
Files
Kafka
Kafka Cluster
5
6
Spark
Flink
Beam
Akka Streams
Kafka Streams
Low Latency
7
8
Spark
Beam
Mini-batch
Disk
HDFS
S3, ...
Search
SQL/
NoSQL
Persistence
9
Spark
...
Batch
Events
Streams
Storage

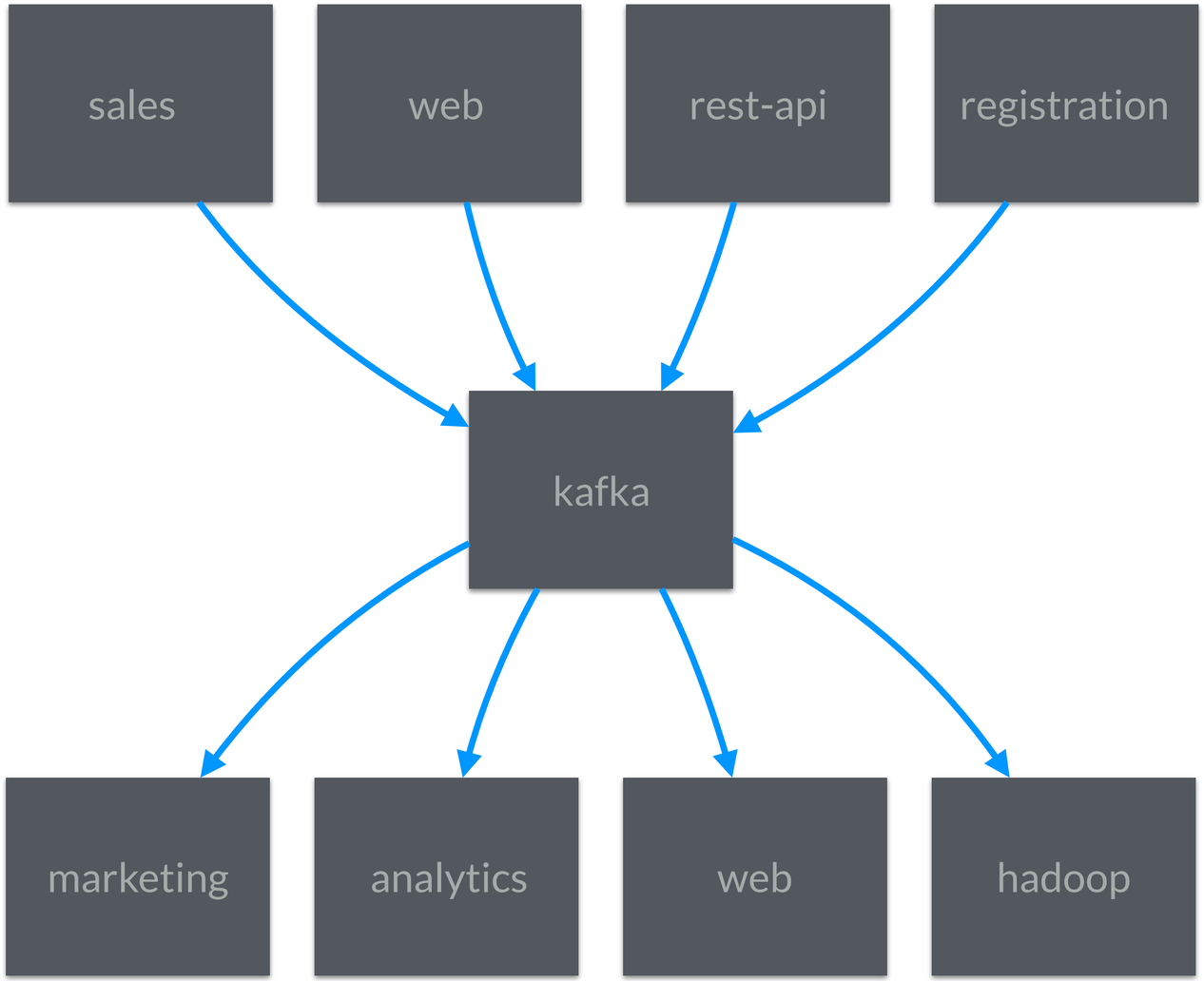
Kafka Introduction


system
system
system
system
system
system
system
system


system
system
system
system
kafka
Producers


kafka
Consumers
system
system
system
system


system
system
system
system
kafka
system
system
system
system



Oversimplied: A Producer can be a Consumer

About Kafka

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

Distributed and Replicated Commit-log

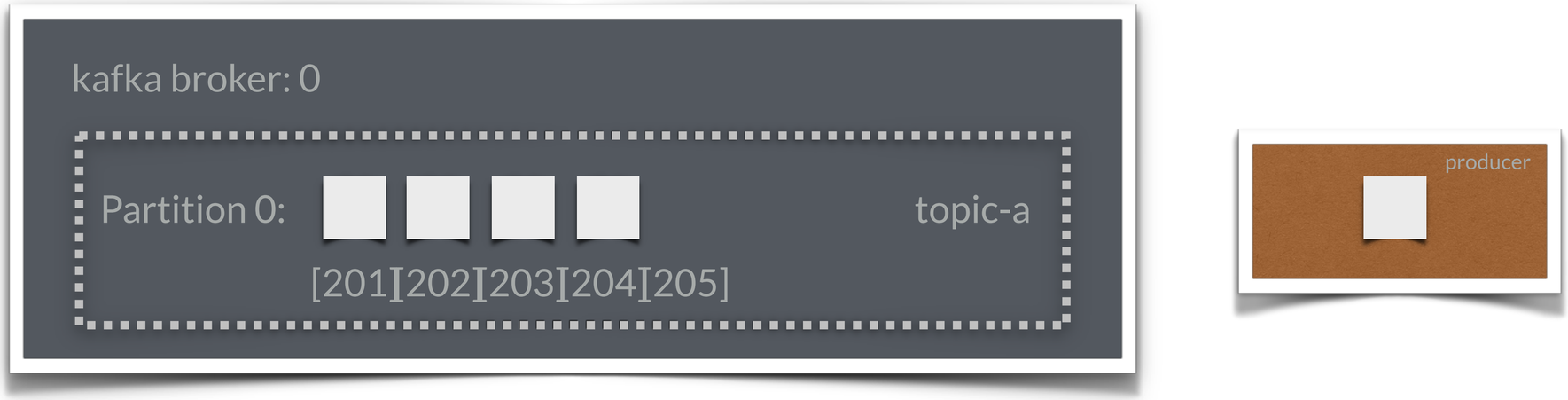
Real Time Data Processing

Stream processing

How messages are produced

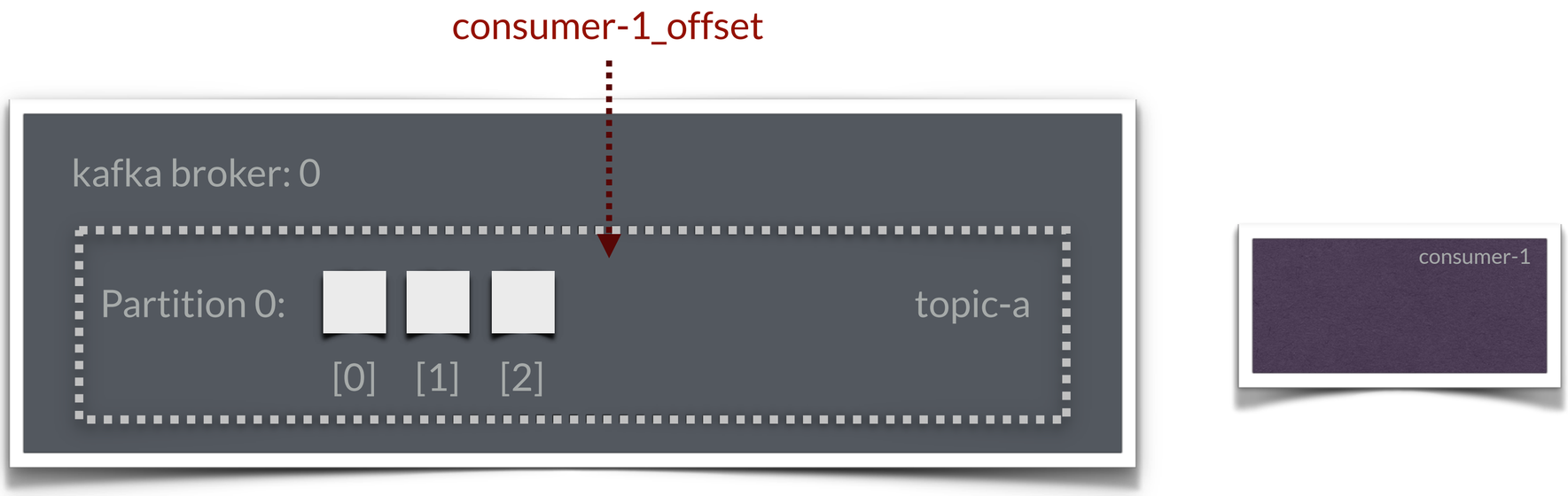

kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]

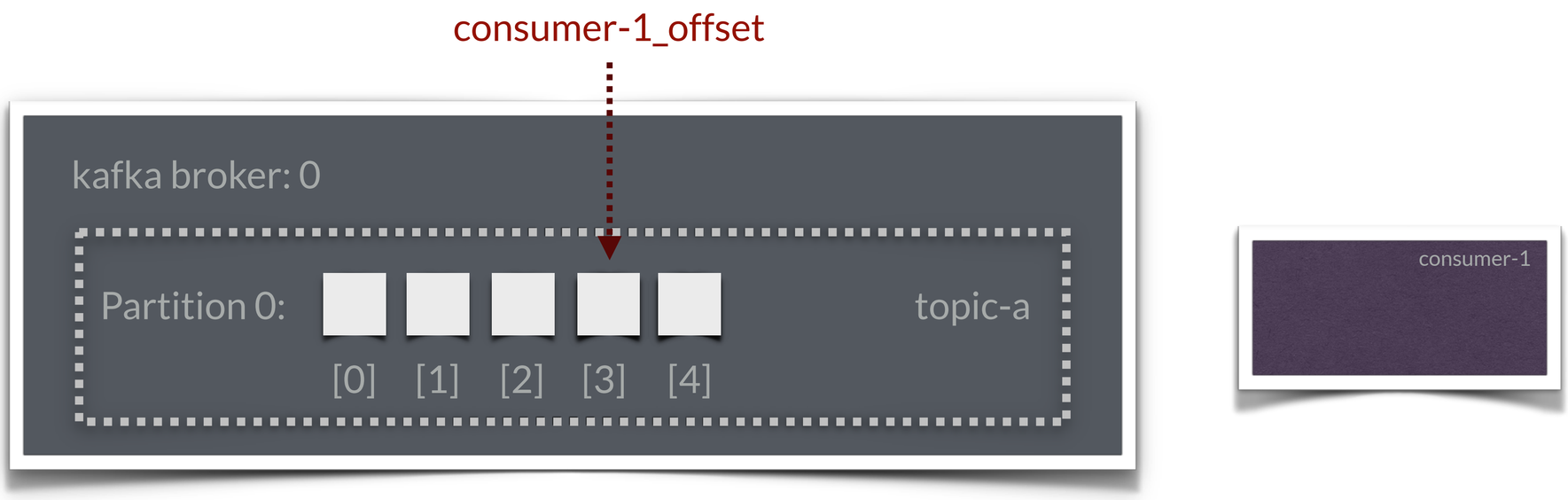
producer



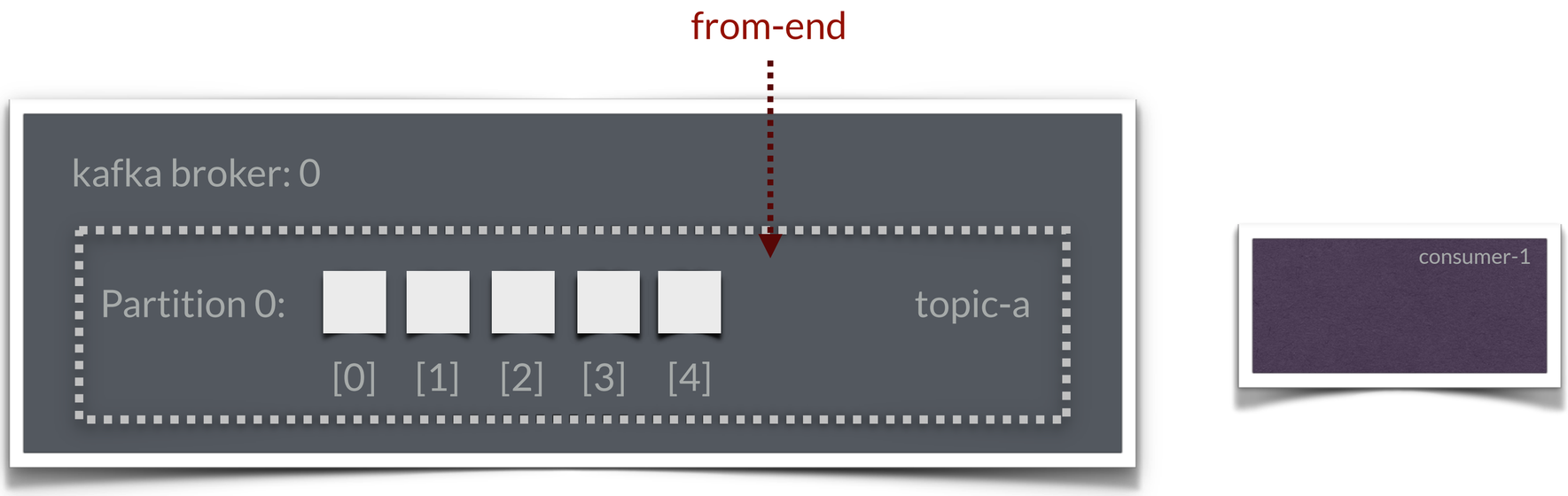
Retention: The **data is temporary**

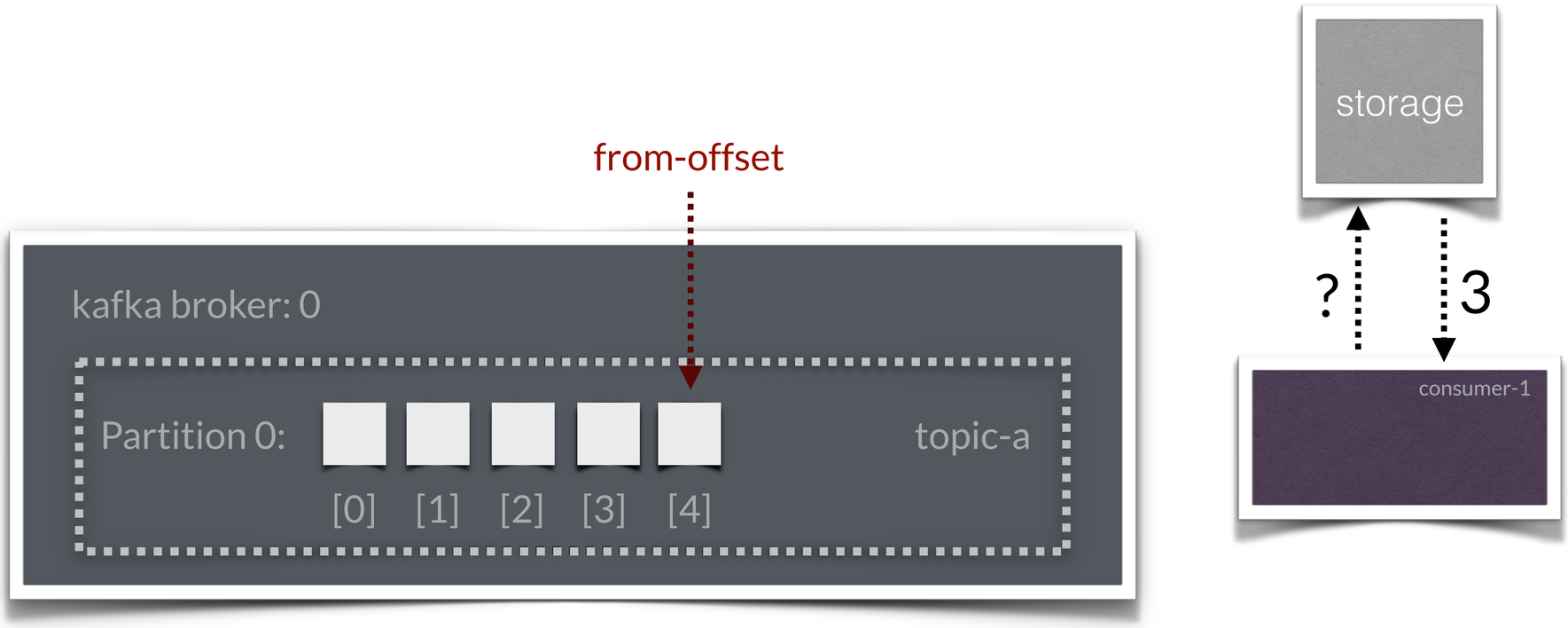
How messages are consumed


consumer-1_offset
kafka broker: 0
Partition 0:
[0]
[1]
[2]
topic-a
consumer-1

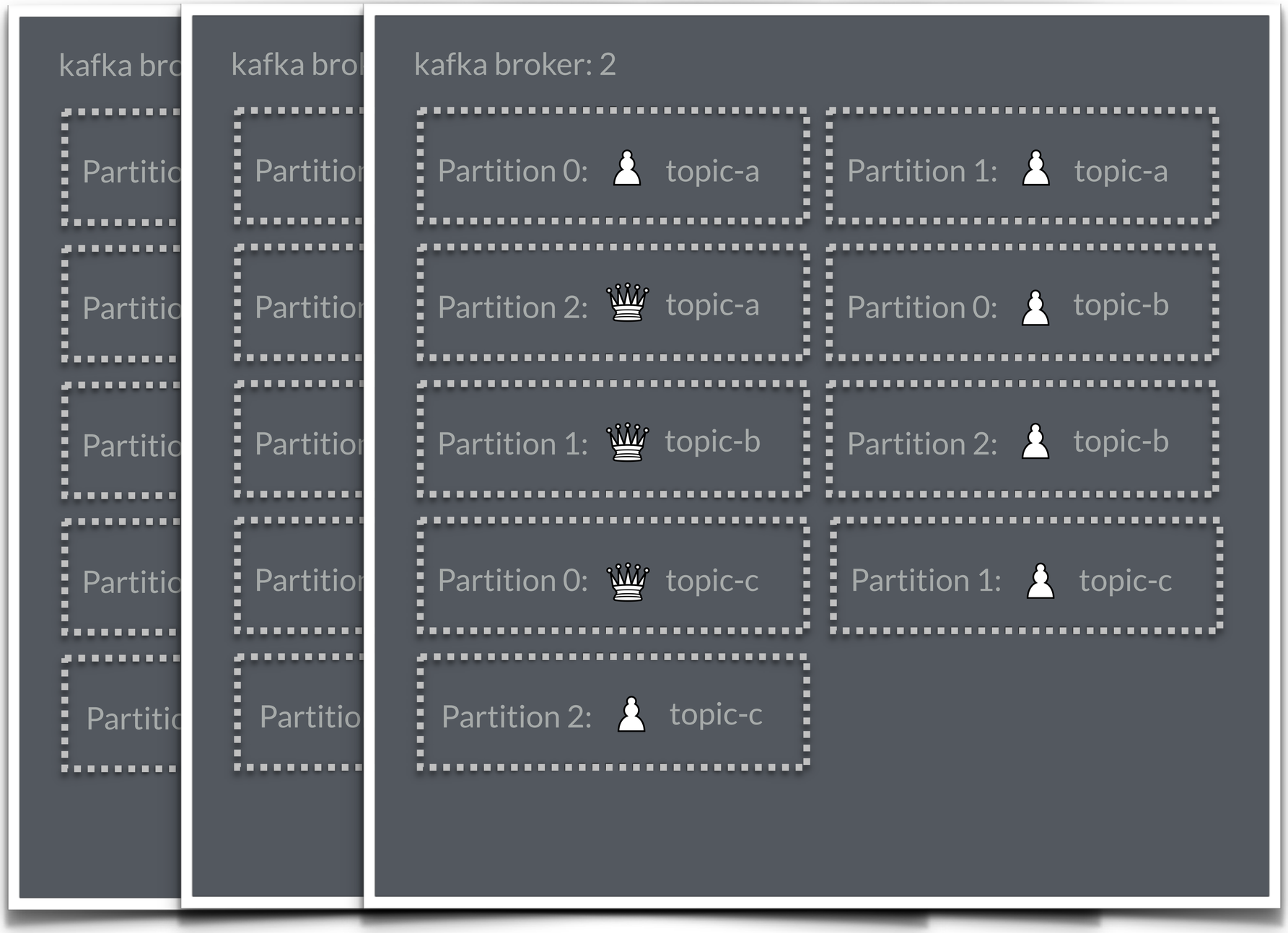

consumer-1_offset
kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]
consumer-1


from-beginning
kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]
consumer-1


from-end
kafka broker: 0
Partition 0:
[0] [1] [2] [3] [4]
topic-a
consumer-1


from-offset
kafka broker: 0
Partition 0:
[0]
[1]
[2]
[3]
[4]
topic-a
storage
?
3
consumer-1

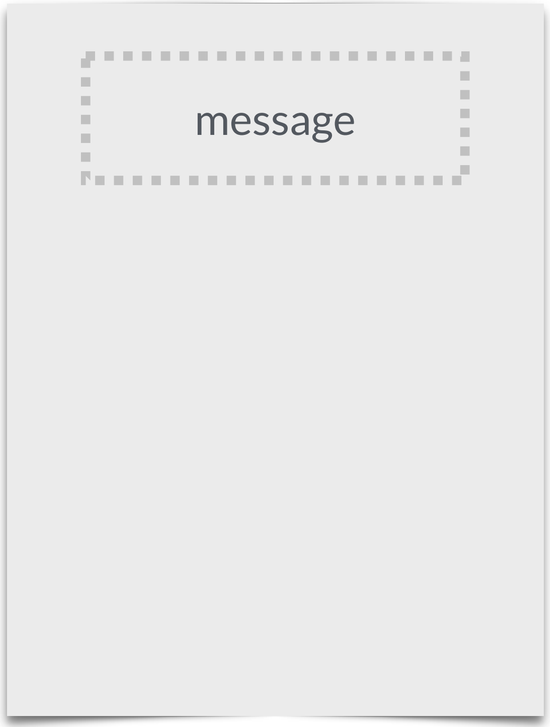
Each partition is on a different broker,
therefore a single topic is scaled


kafka broker: 2
Partition 0: topic-a
Partition 1: topic-a
Partition 2: topic-a
Partition 0: topic-b
Partition 1: topic-b
Partition 2: topic-b
Partition 0: topic-c
Partition 1: topic-c
Partition 2: topic-c

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 Guarantees

Kafka Guarantees

Messages sent by a producer to a particular topic partition will be appended in the order they are sent.

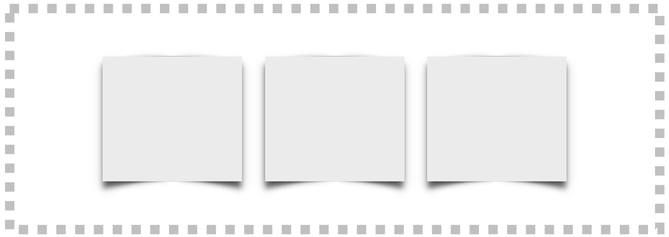
A consumer instance sees records in the order they are stored in the log.

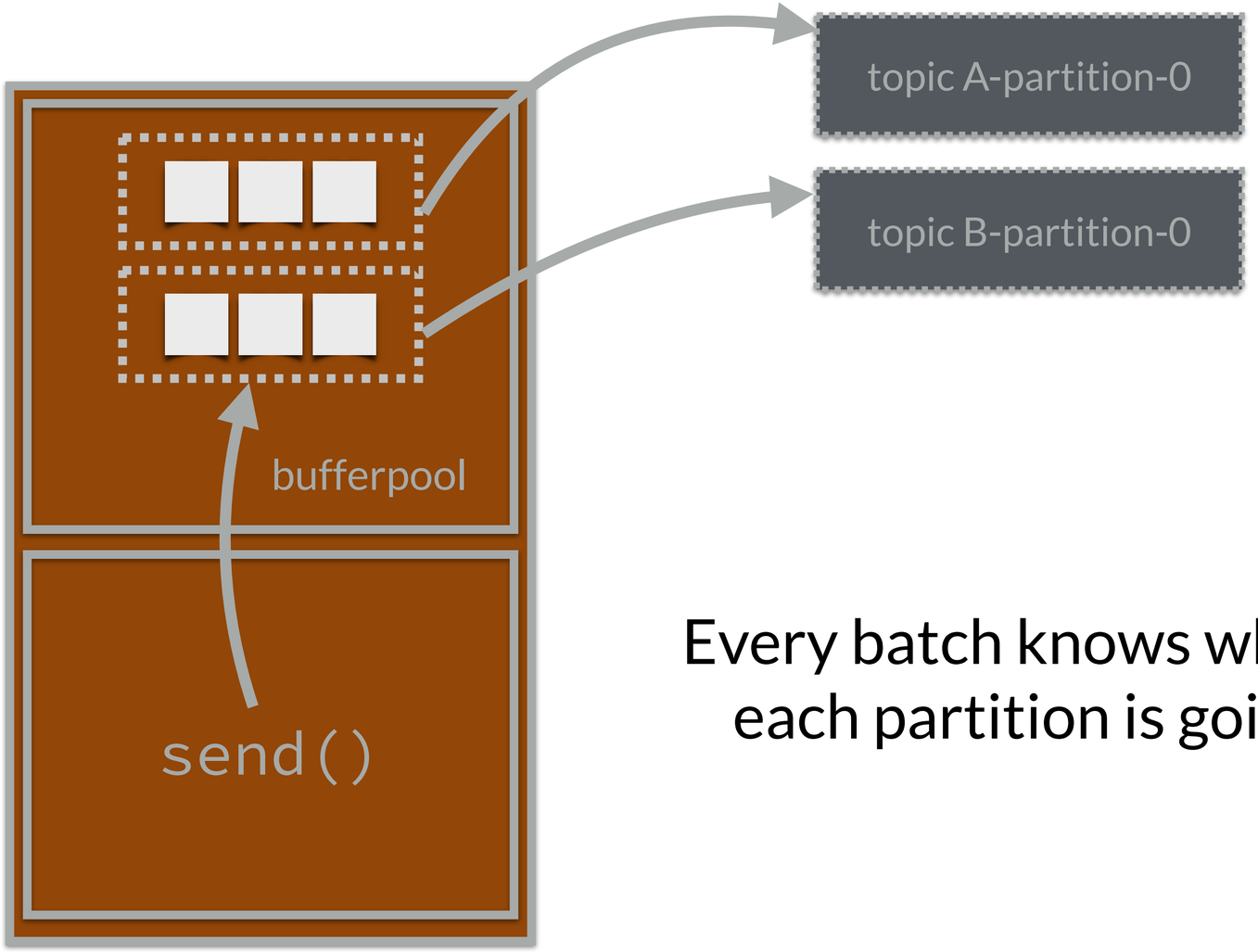
For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

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

```
murmur2(bytes) % number partitions
```

Acks

Acks

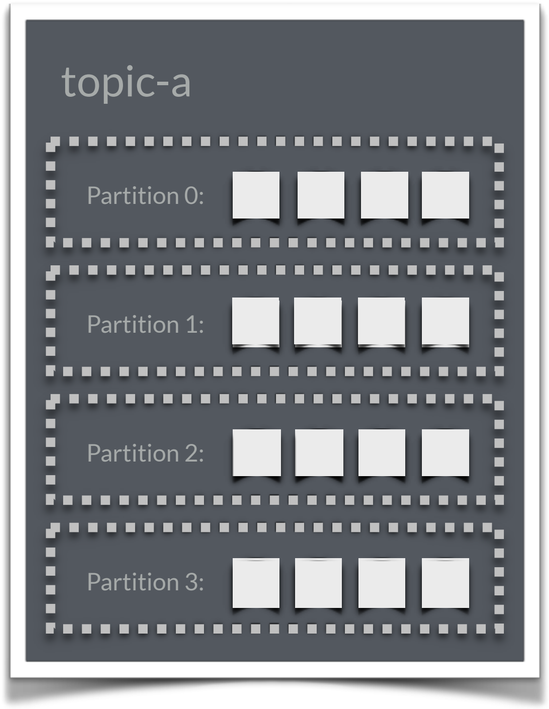
`acks` controls how many partition replicas must receive the record before the write is considered a success.

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


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:

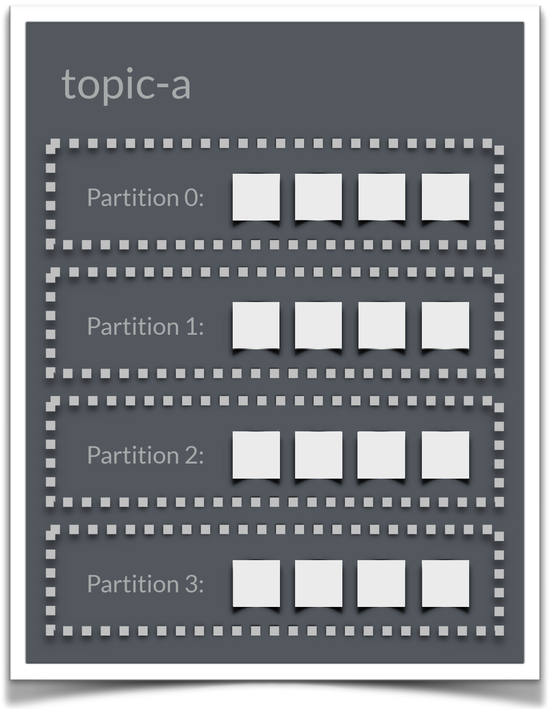

consumer-1

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



consumer-1

consumer-2

consumer-3

consumer-4

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

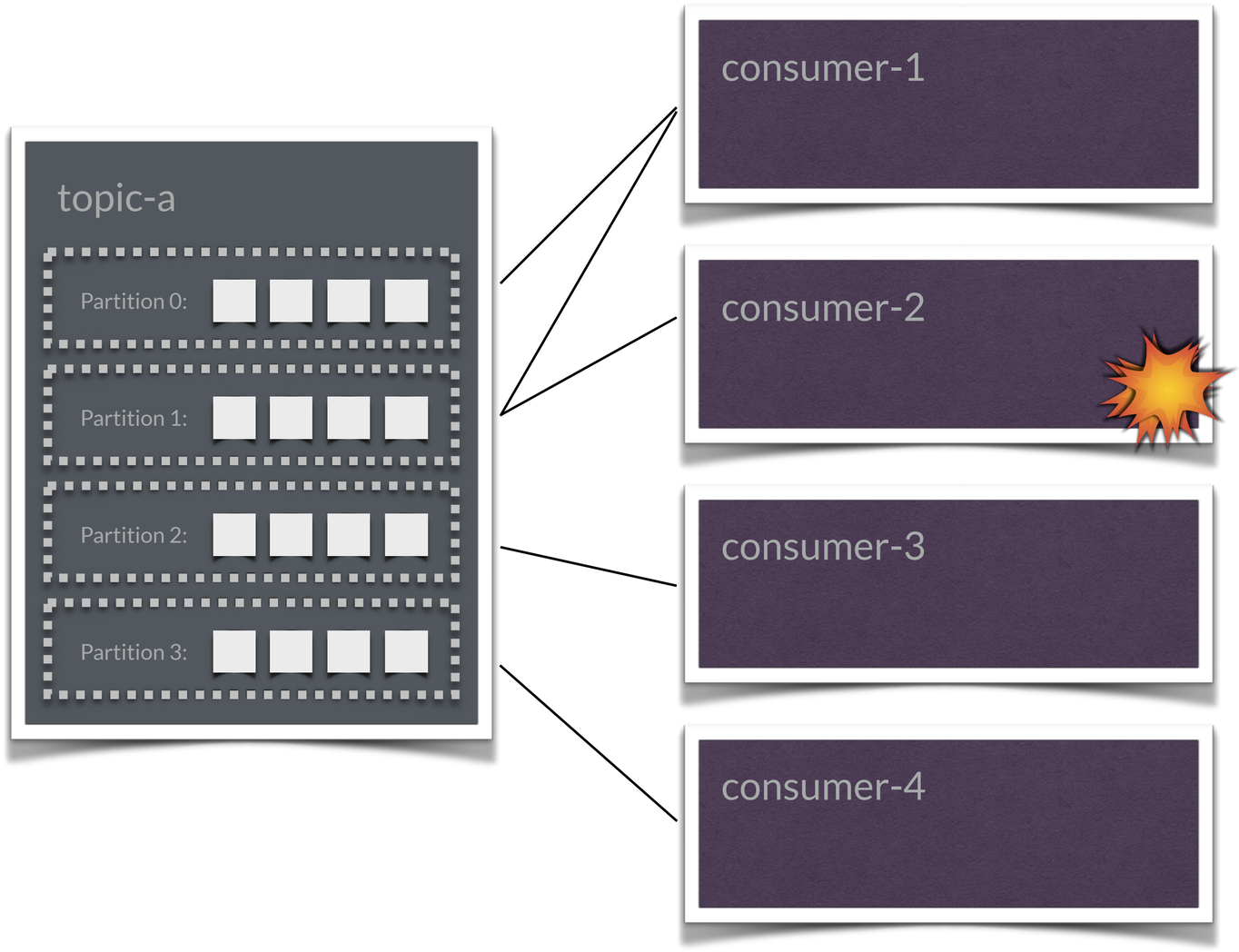
Consumer Rebalancing

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 or added.

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


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:
consumer-1
consumer-2
consumer-3
consumer-4

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

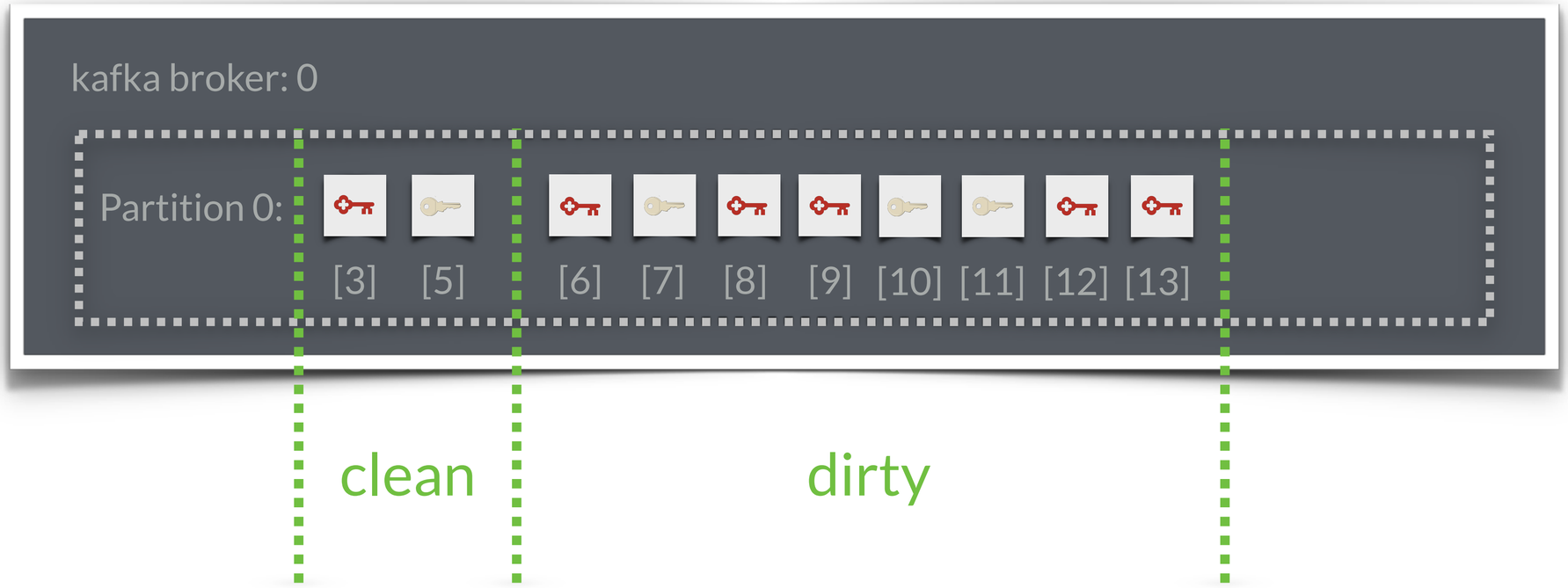
Compaction

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*.


kafka broker: 0
Partition 0:
[0]
[1]
[2]
[3]
[4]
[5]
kafka broker: 0
Partition 0:
[3]
[5]



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

Streaming


TK45
115m
ÚNIK.C.

Stream Processing

Endless supply of data

“Replayable"

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

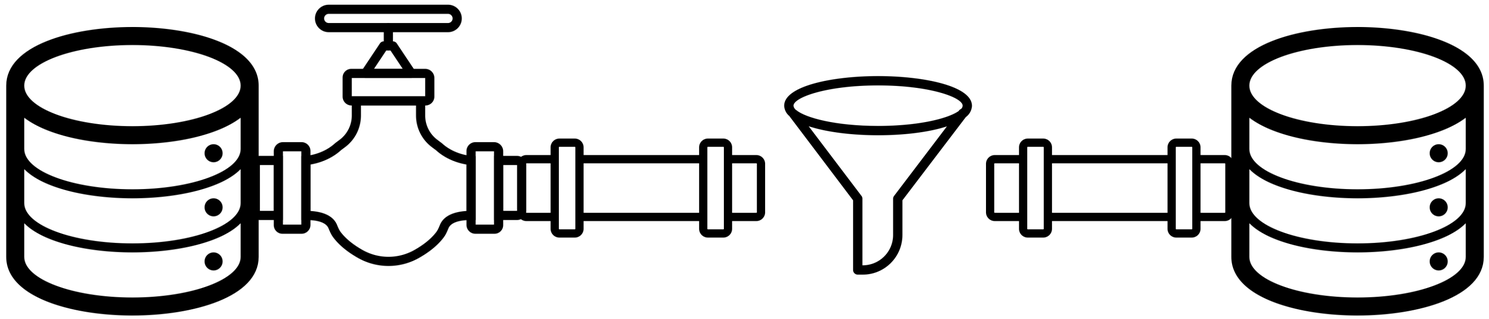
Will require built-in accumulator table to perform real time data processing, like *counting*, and *grouping*


NY
100.00

Filtering

NY

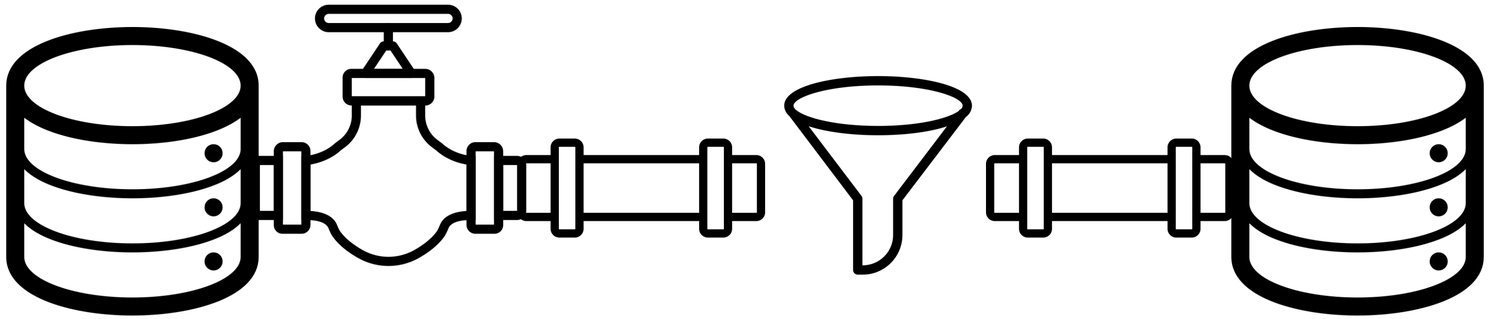
100.00



`value > 10000`

OH

20000

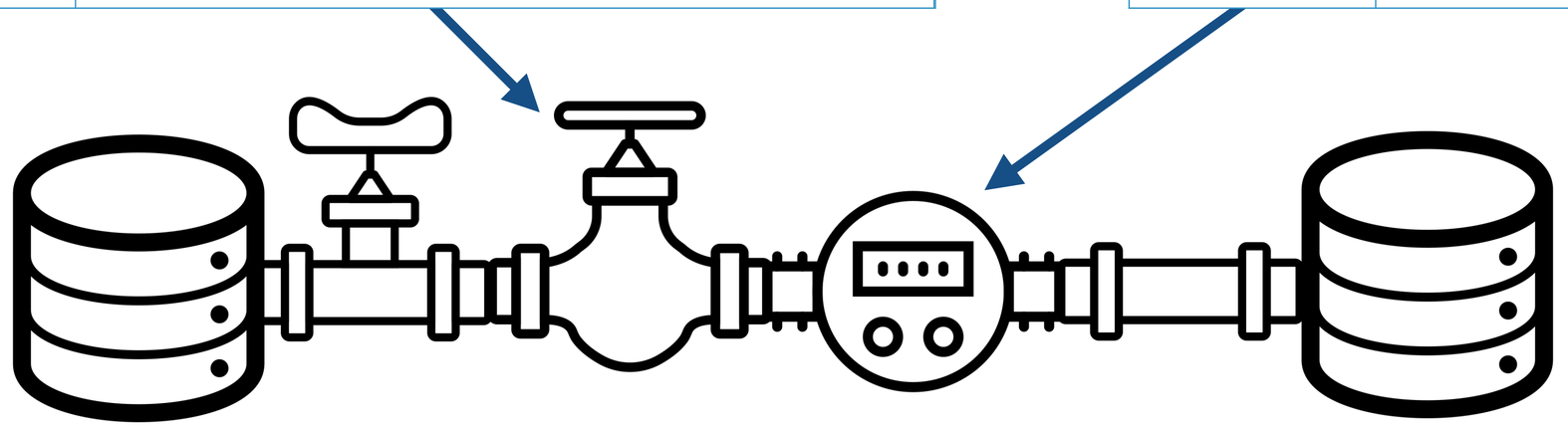


`value > 10000`

Aggregating

Key	Value
OH	100.0

Key	Value
OH	100.0



OH

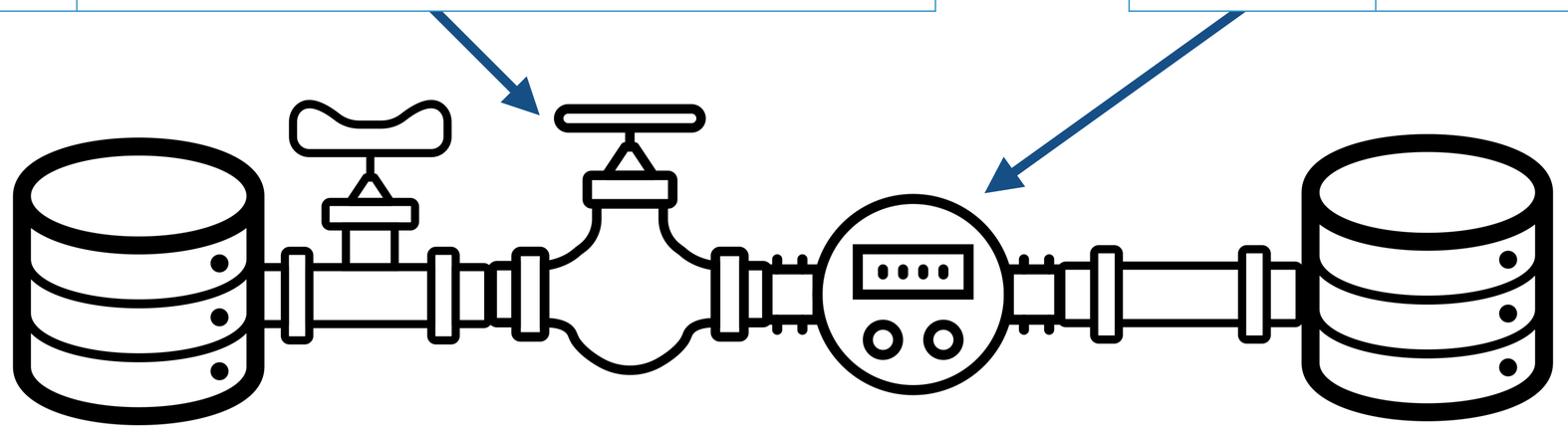
100.00

OH

100.00

Key	Value
OH	100.0
NY	20.0

Key	Value
OH	100.0
NY	20.0



NY

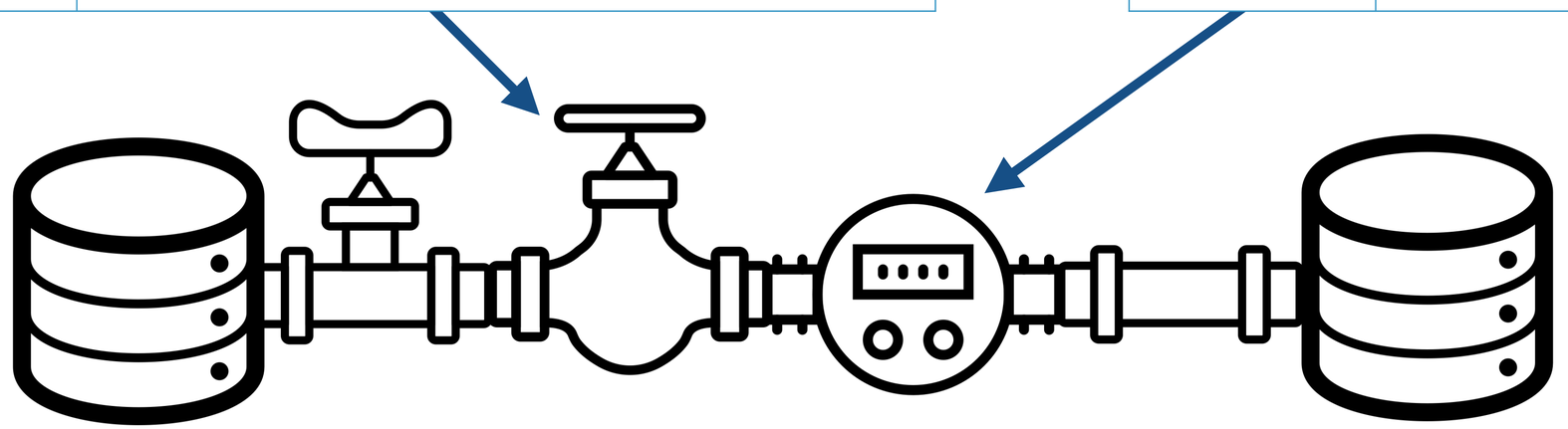
20

NY

20

Key	Value
OH	100.0
NY	20.0, 60.0

Key	Value
OH	100.0
NY	80.0

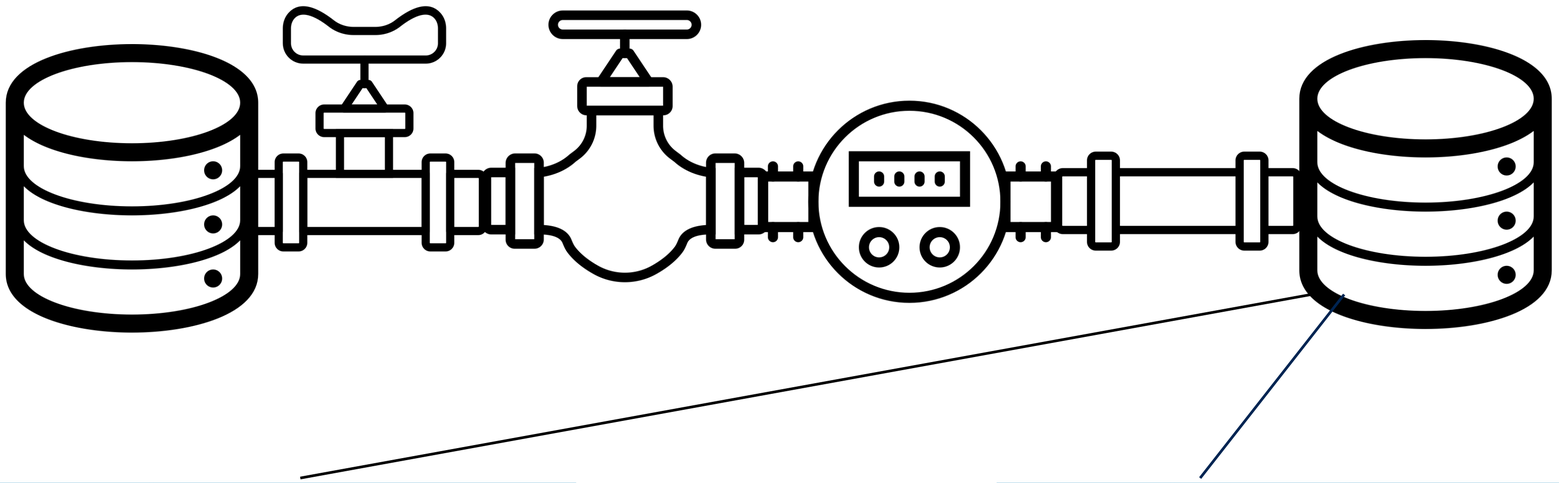


NY

60

NY

80.0

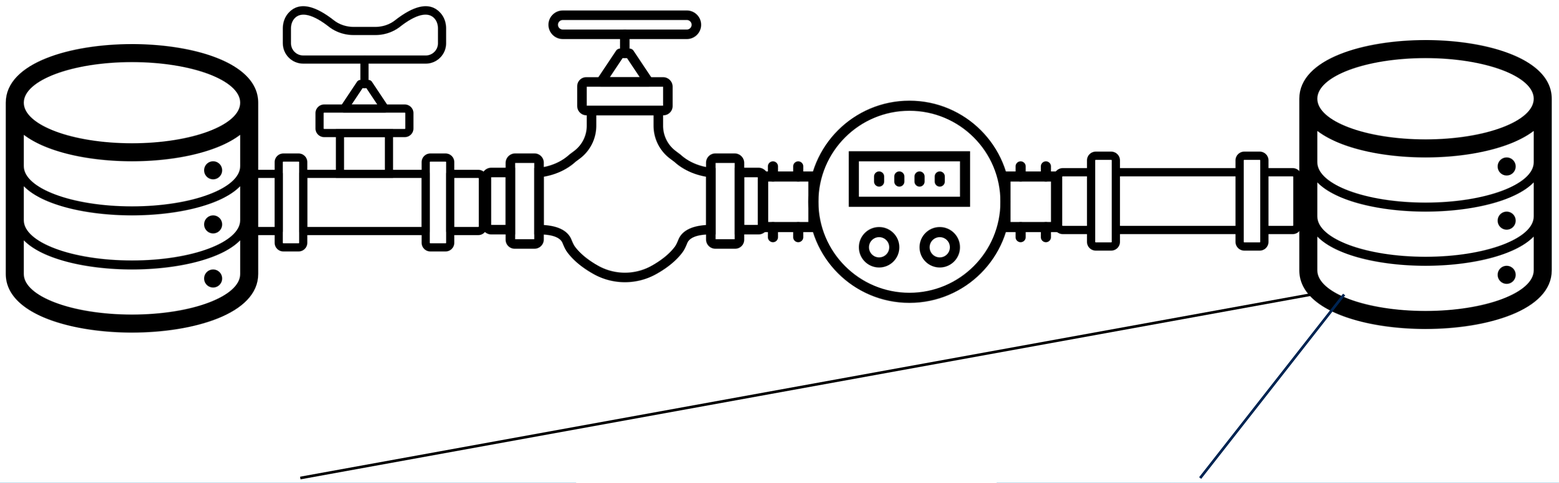


Offset	Key	Value
0	**OH**	100.0

Partition 0

Offset	Key	Value
0	**NY**	20.0
1	**NY**	80.0

Partition 1



Offset	Key	Value
0	**OH**	100.0

Partition 0

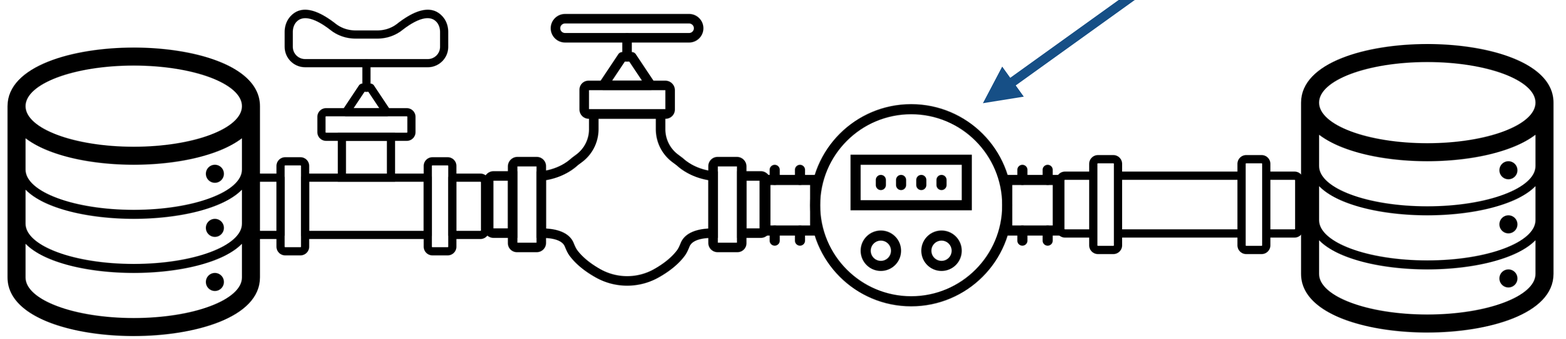
Offset	Key	Value
1	**NY**	80.0

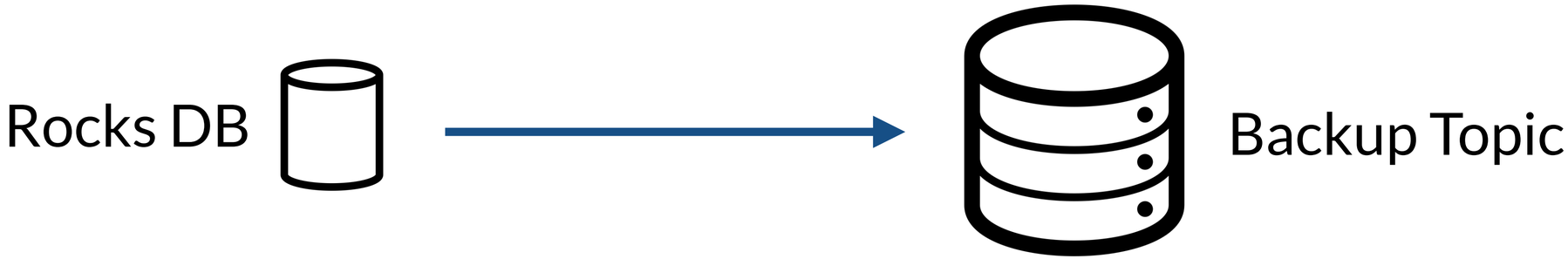
Partition 1

After Compaction

Rocks DB

Key	Value
OH	100.0
NY	80.0




Rocks DB
Backup Topic

Windowing

Tumbling Window

Hopping Window

Session Window

Session Window

the last element has a occured; now we wait

Session Window

waited 5 minutes = make it a window

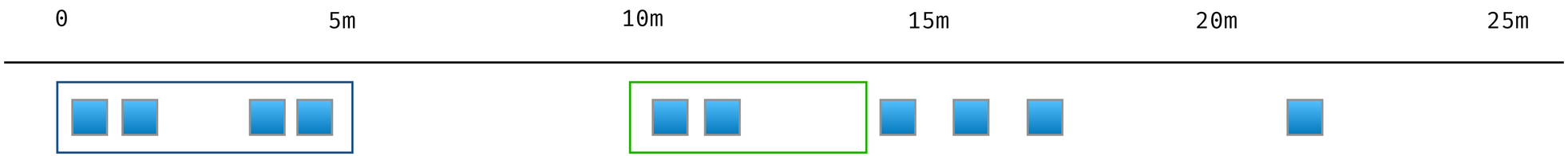
Session Window

we have a new element; we start a new window

Session Window

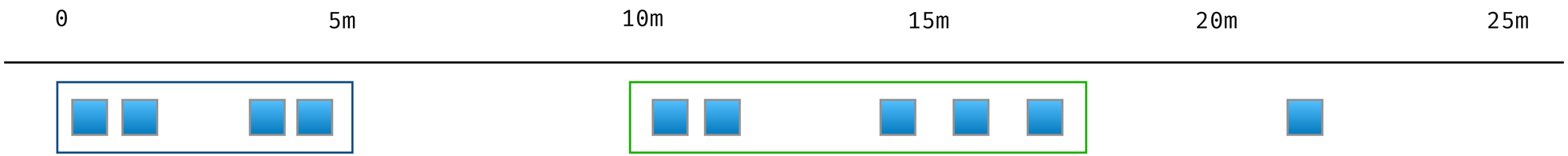
we wait for 5 minutes

Session Window



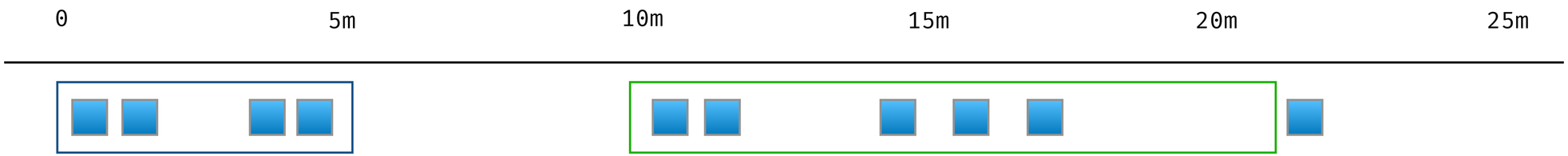
2 minutes elapsed; no windowing yet

Session Window



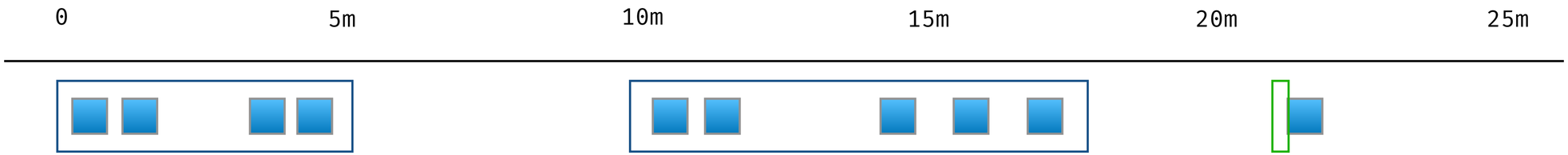
we wait five minutes

Session Window



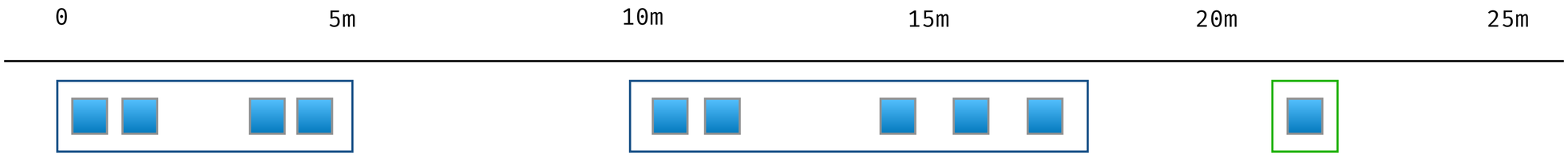
no new elements; we make it a window

Session Window



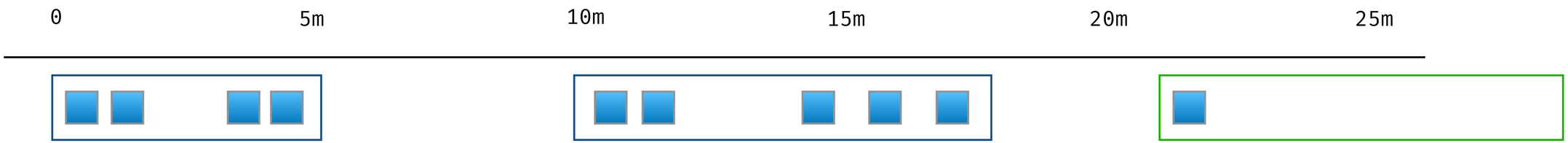
we see something new

Session Window



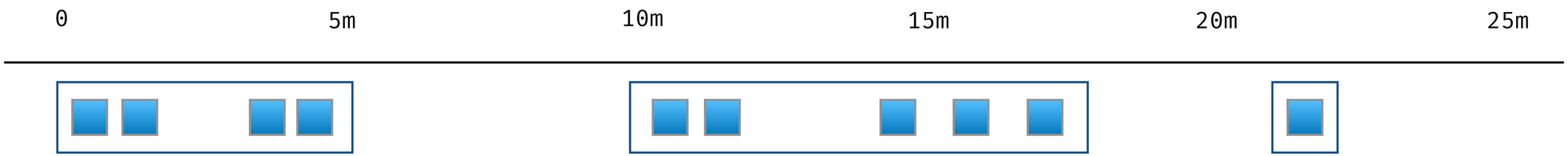
last element; we wait five minutes

Session Window



nothing else; we wait 5 minutes

Session Window



that's a new window

Demo: Streams

KSQL

KSQL

Open source streaming SQL engine for Apache Kafka

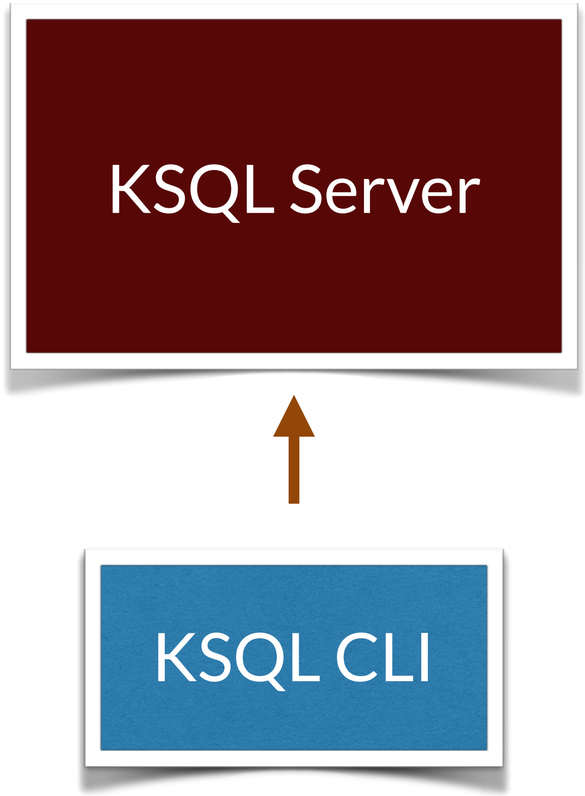
Provides SQL interface for stream processing on Kafka

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

KSQL

Supports a wide range of streaming operations, including:

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


KSQL Server
KSQL CLI

KSQL Architecture

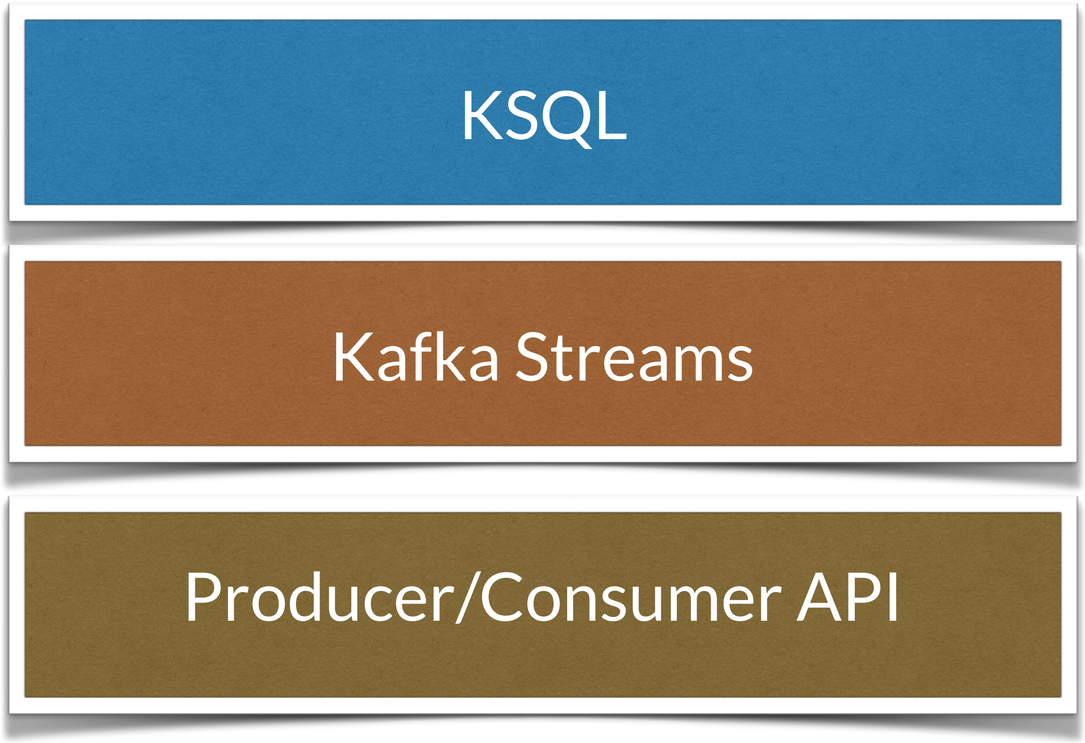
KSQL Server communicates with the broker

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

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

KSQL CLI interacts with KSQL Servers via a REST API

KSQL Server runs in two modes: CLI, Headless


KSQL
Kafka Streams
Producer/Consumer API

Starting in CLI Mode

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

Starting in Headless Mode

```
/bin/ksql-start-server \
server_path/etc/ksql/ksql-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

Persistent Query

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

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

Non-Persistent 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.

Persistent v. Non Persistent

	Persistent	**Non Persistent**
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 timestring \
  FROM pageviews \
  PARTITION BY userid;
```

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

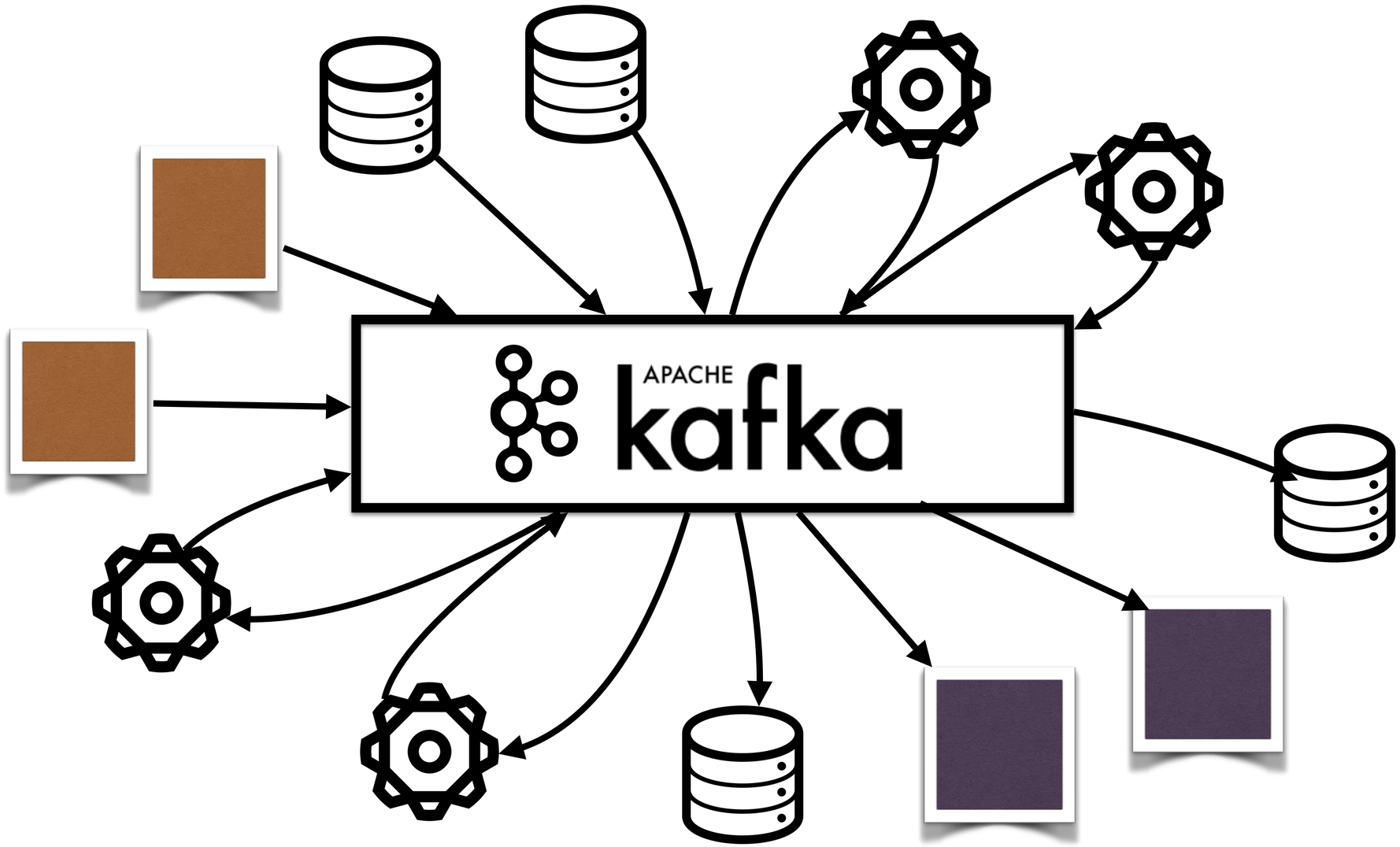
```
SELECT item_id, SUM(quantity)
  FROM orders
  WINDOW HOPPING (SIZE 20 SECONDS,
     ADVANCE BY 5 SECONDS)
  GROUP BY item_id;
```

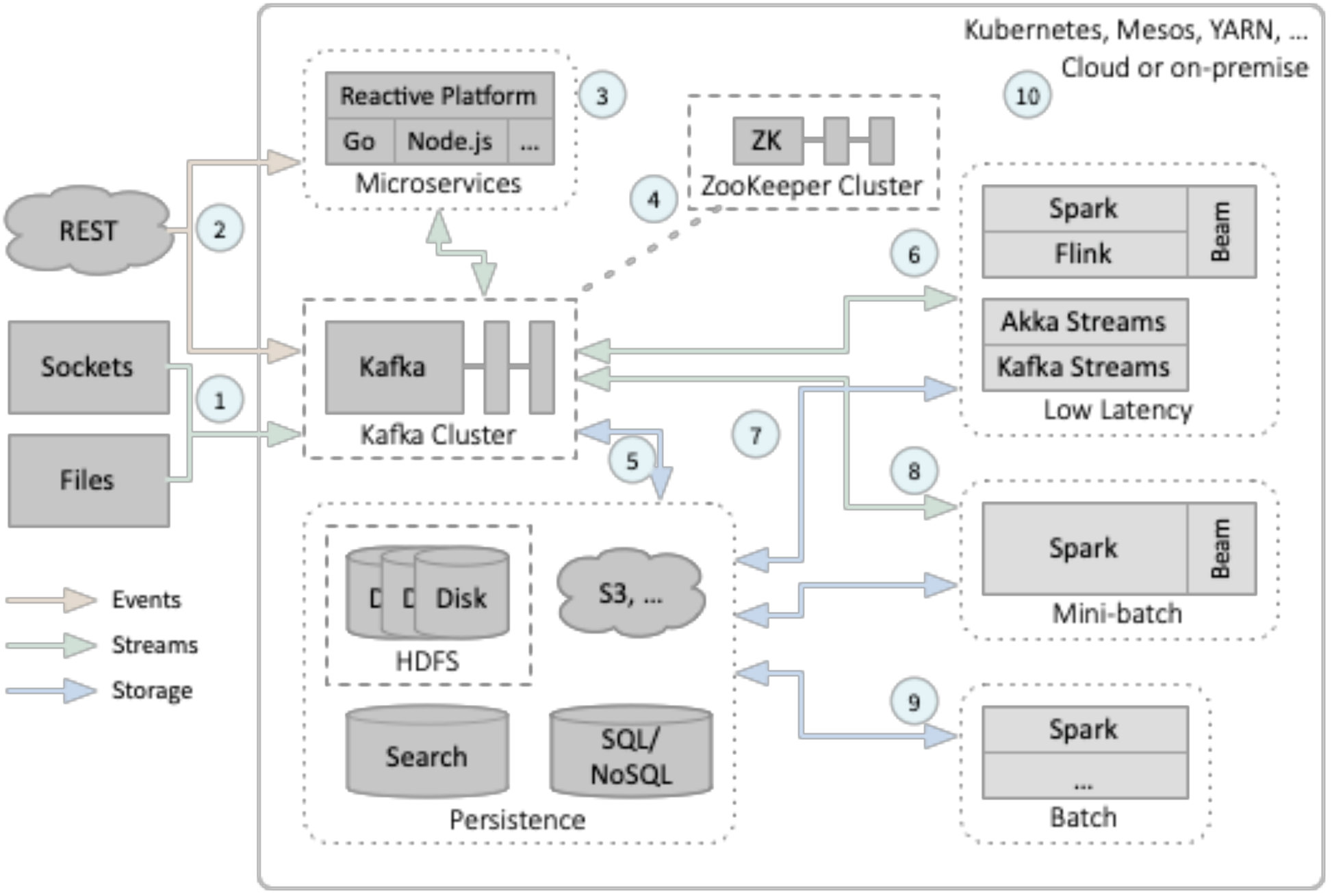
Session Window

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

Demo: KSQL

Conclusion


APACHE
kafka


Kubernetes, Mesos, YARN, ...
Cloud or on-premise
10
Reactive Platform
Go
Node.js
...
Microservices
3
ZK
ZooKeeper Cluster
4
REST
2
Sockets
1
Files
Kafka
Kafka Cluster
5
6
Spark
Flink
Beam
Akka Streams
Kafka Streams
Low Latency
7
8
Spark
Beam
Mini-batch
Disk
HDFS
S3, ...
Search
SQL/
NoSQL
Persistence
9
Spark
...
Batch
Events
Streams
Storage

Thank You

Daniel Hinojosa
Programmer, Consultant, Trainer

dhinojosa@evolutionnext.com
@dhinojosa