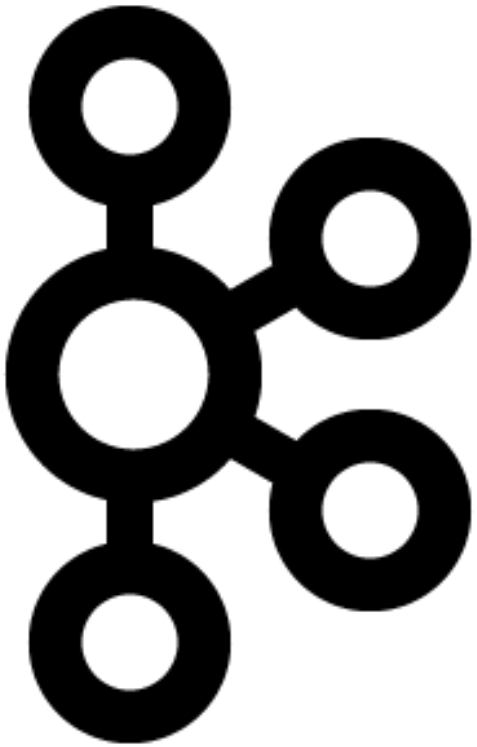


APACHE kafka®

A distributed streaming platform

Need of Messaging Systems

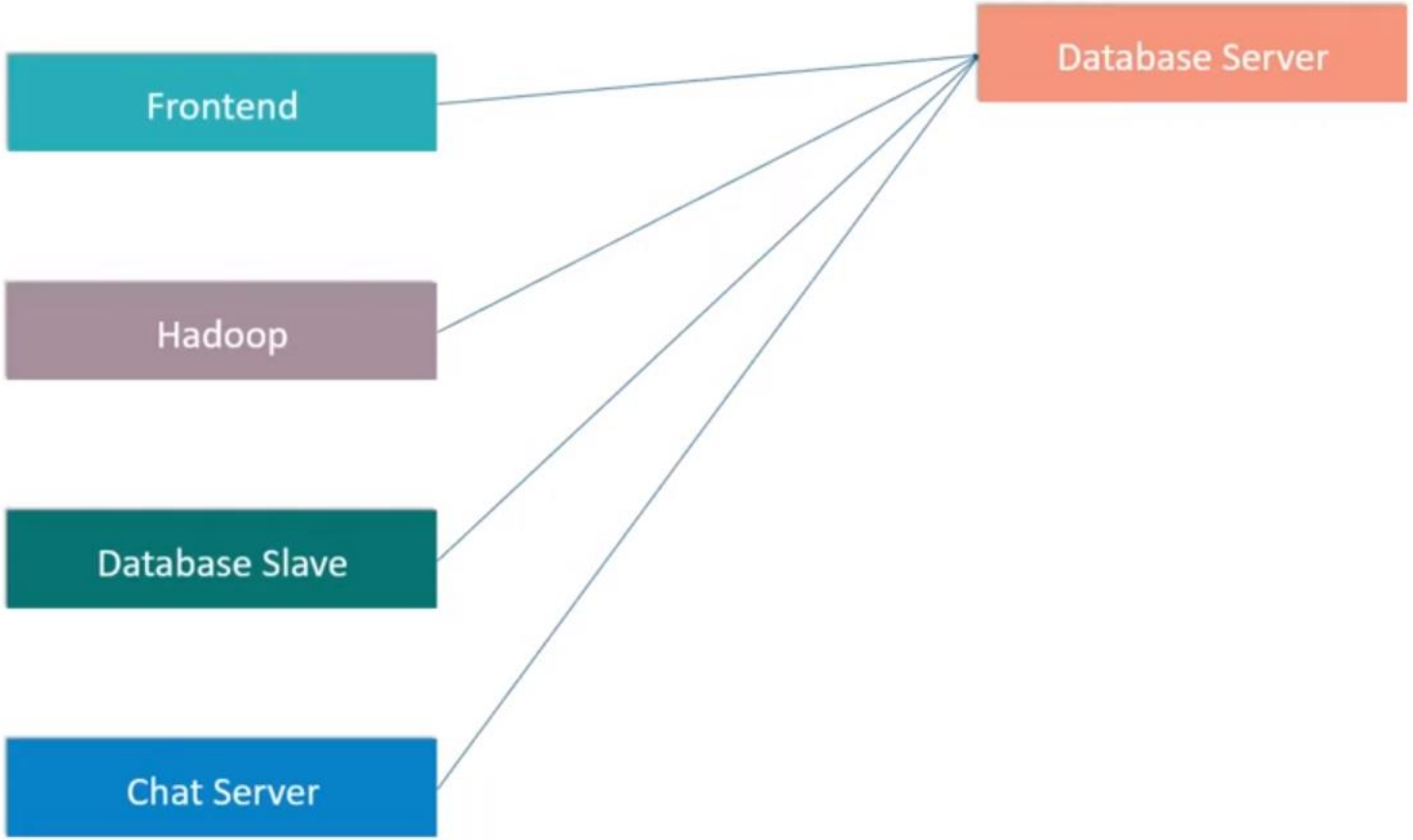
Data Pipelines

Communication is required between different systems in the real-time scenario, which is done by using data pipelines.

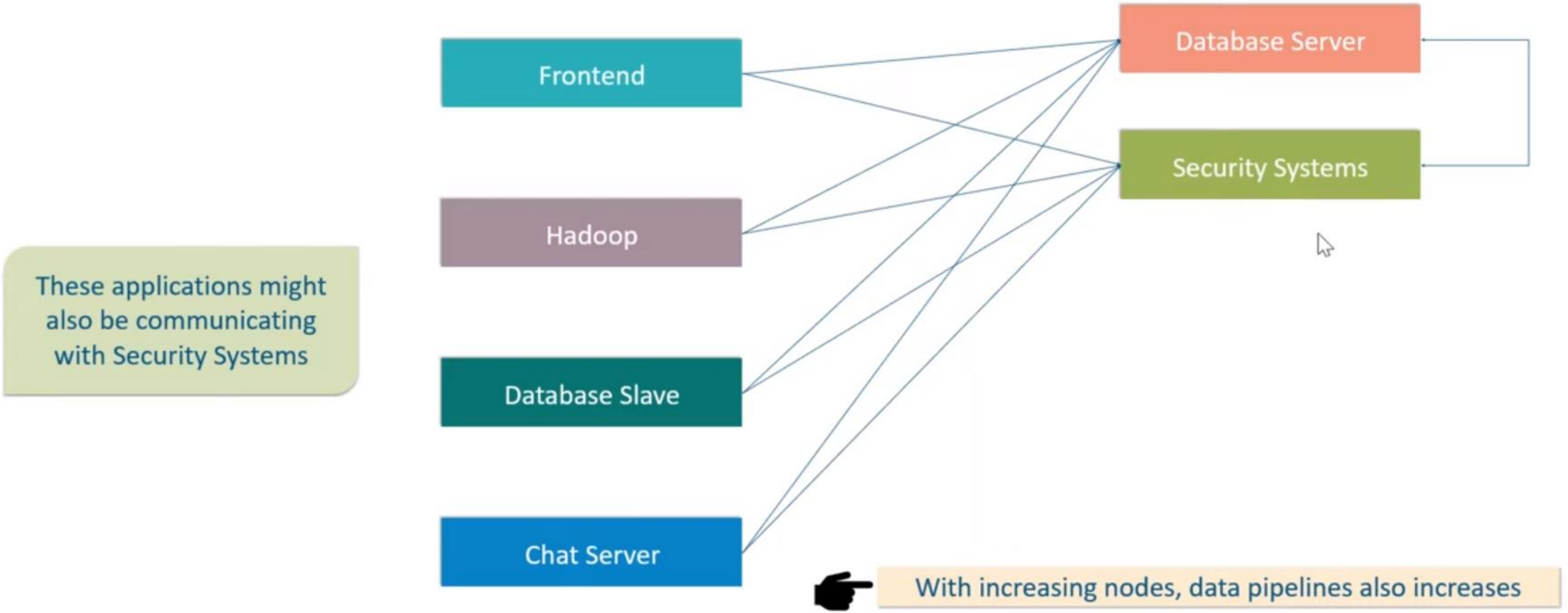
For Example: Chat Server needs to communicate with Database Server for storing messages

Increase in number of Nodes

Similarly, there may be many applications wanting to access the Database Server

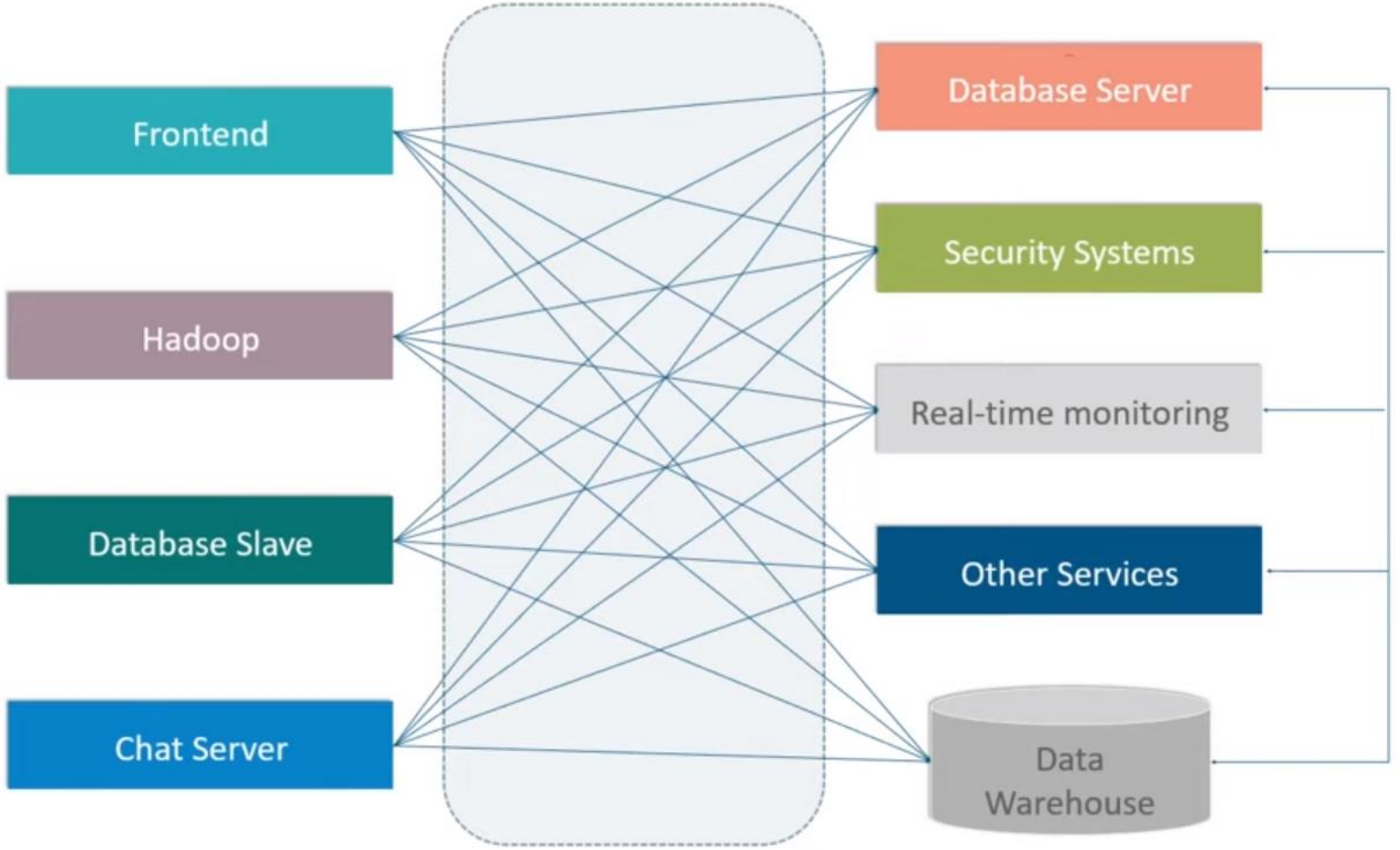


Increase in number of Nodes



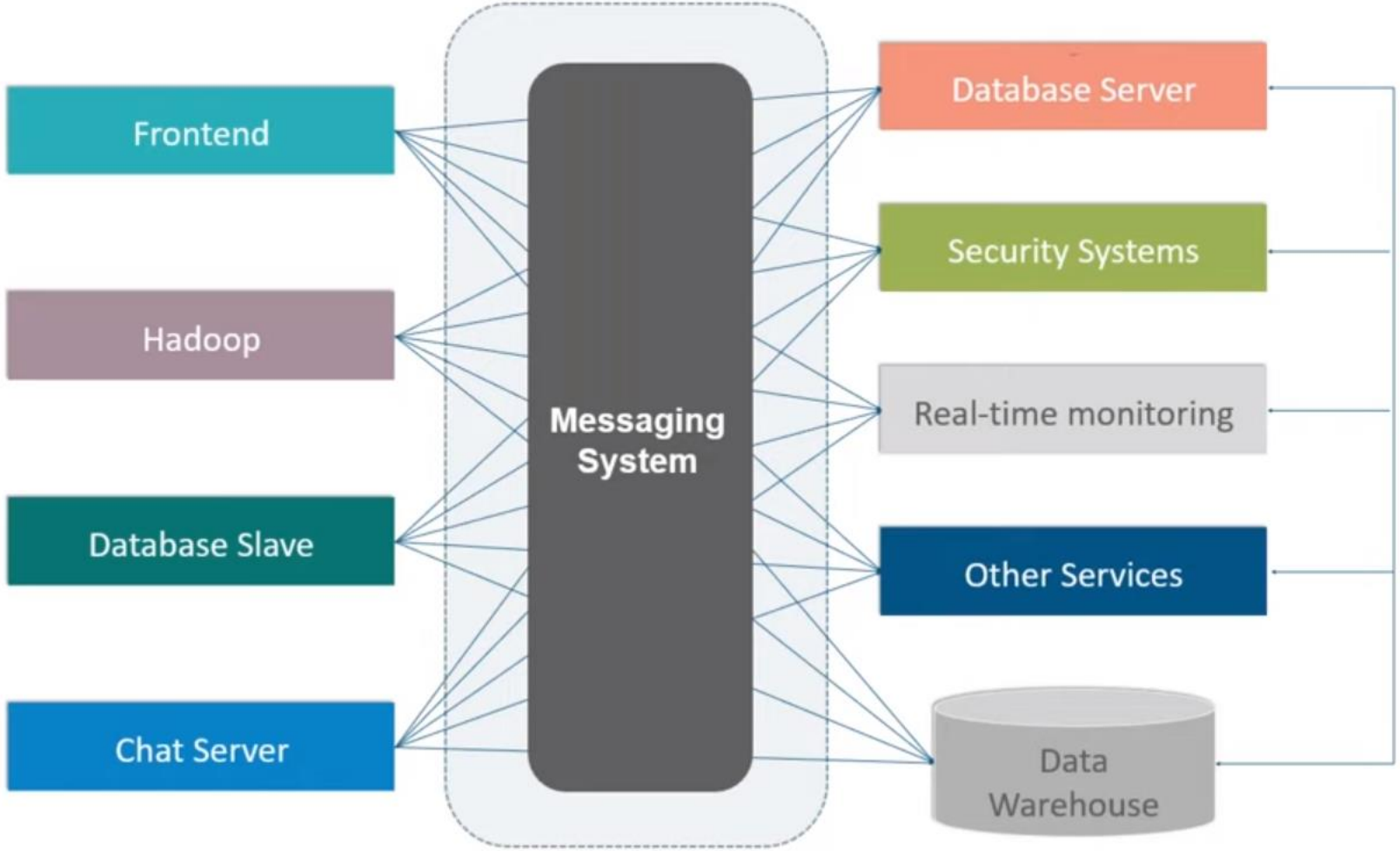
Complex Data Pipelines

Similarly, applications may also be communicating with Real-time monitoring and Other services in real-time scenario



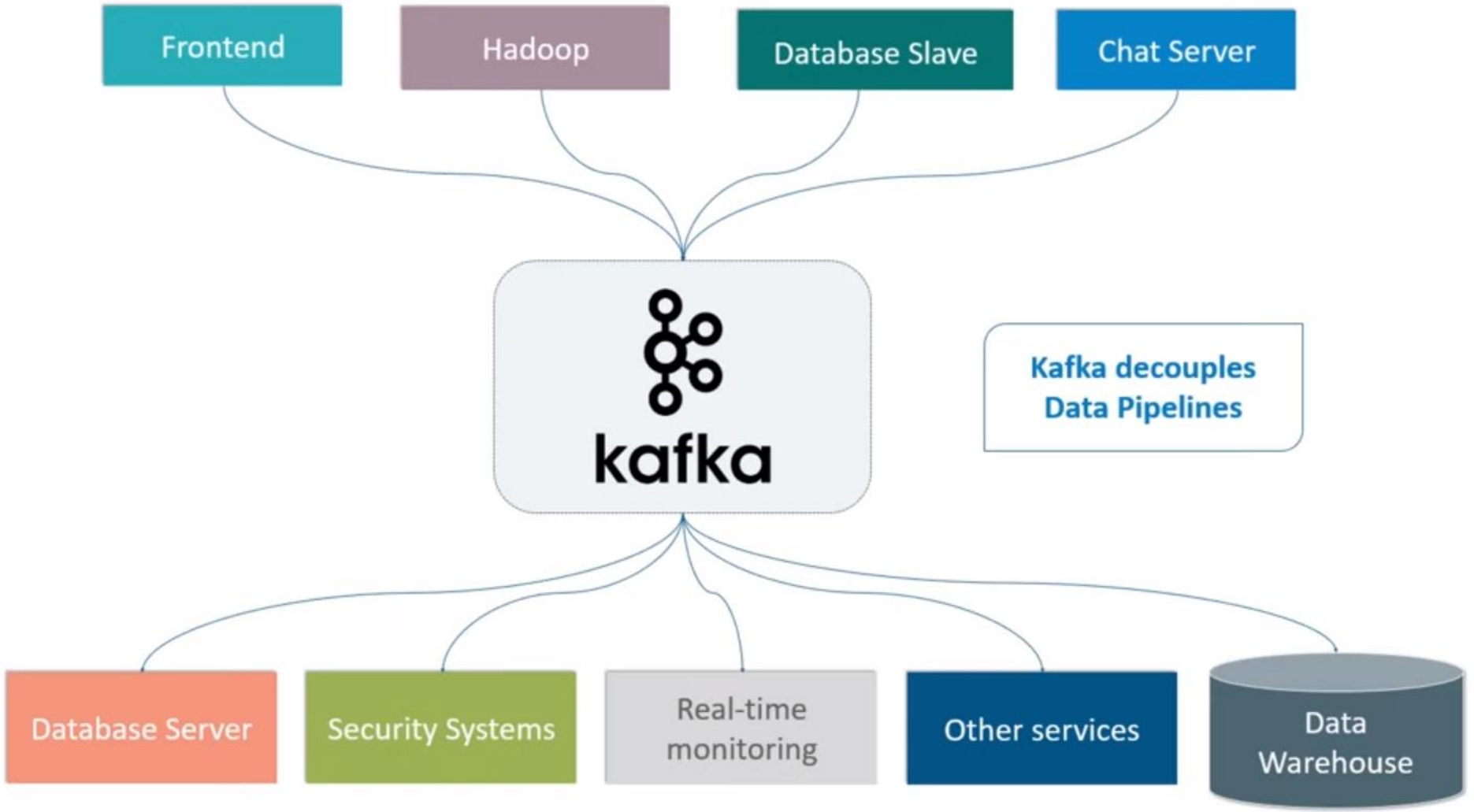
Solution to the Complex Data Pipelines

Messaging Systems helps managing the complexity of the pipelines



Let’s See How Kafka Solves the Problem

Kafka Decouples Data Pipelines



What is Kafka?

- **Apache Kafka** is a distributed *publish-subscribe* messaging system
- It was originally developed at LinkedIn and later on became a part of Apache Project
- Kafka is fast, scalable, durable, fault-tolerant and distributed by design

Kafka @LinkedIn

- *1100+ commodity machines*
- *31,000+ topics*
- *350,000+ partitions*

- *675 billion messages/day*
- *150 TB/day in*
- *580 TB/day out*

Peak Load

- *10.5 million messages/sec*
- *18.5 GB/sec Inbound*
- *70.5 GB/sec Outbound*

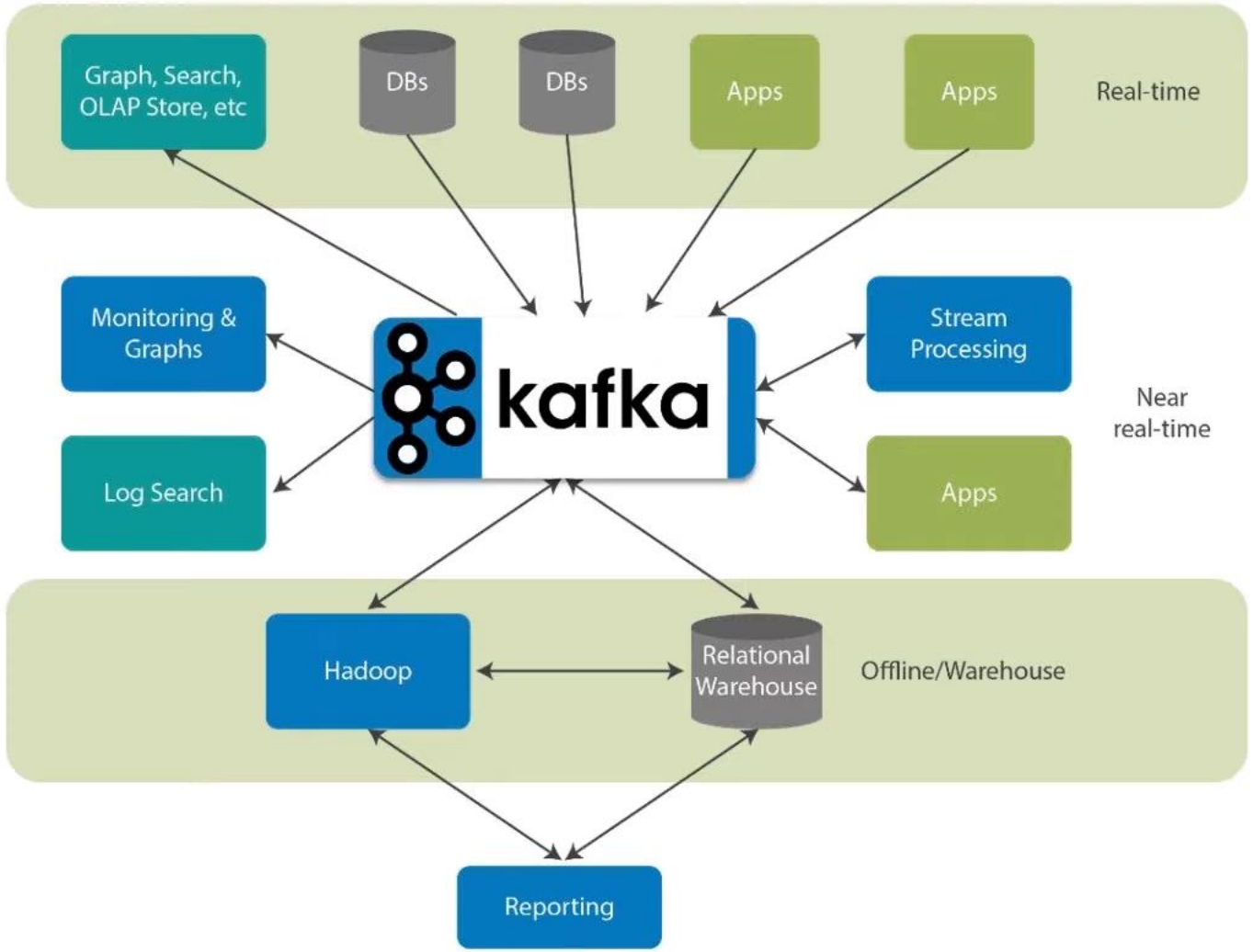


Fig: A modern stream-centric data architecture built around Kafka

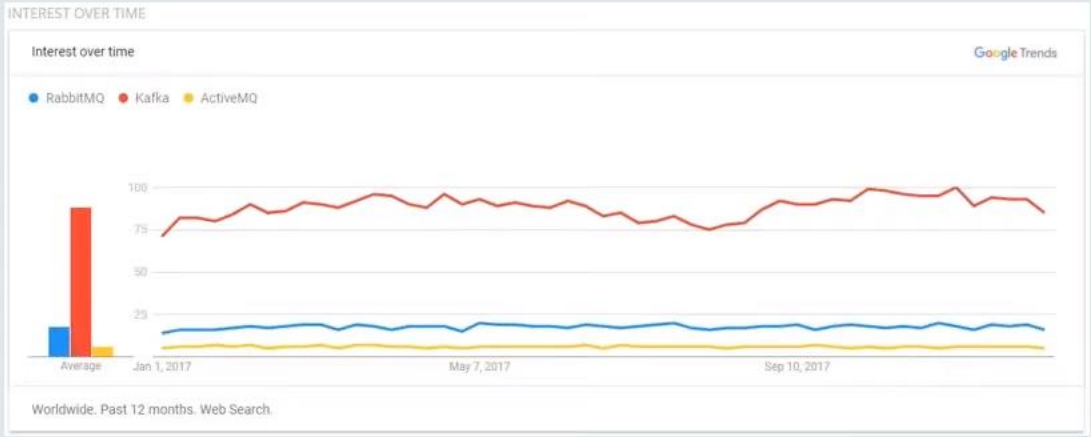
Kafka Growth Exploding

- More than **1/3** of all Fortune **500** companies use **Kafka**.
- These companies includes the top ten travel companies, **7** of top ten banks, **8** of top ten insurance companies, **9** of top ten telecom companies.
- **LinkedIn**, **Microsoft** and **Netflix** process billions of messages a day with Kafka (1,000,000,000,000).
- **Kafka** is used for **real-time streams** of data & used to collect big data for **real time analysis**.

86% of respondents reported that the number of their systems that use Kafka is increasing

20% reported that the number is "growing a lot!"

52% of organizations have at least 6 systems running Kafka



Source: Google Trends

Kafka Concepts

Kafka Terminologies

Producer

A ***producer*** can be any application who can publish messages to a topic

Consumer

A ***consumer*** can be any application that subscribes to a topic and consume the messages

Partition

Topics are *broken* up into *ordered commit logs* called ***partitions***

Broker

Kafka cluster is a set of servers, each of which is called a ***broker***

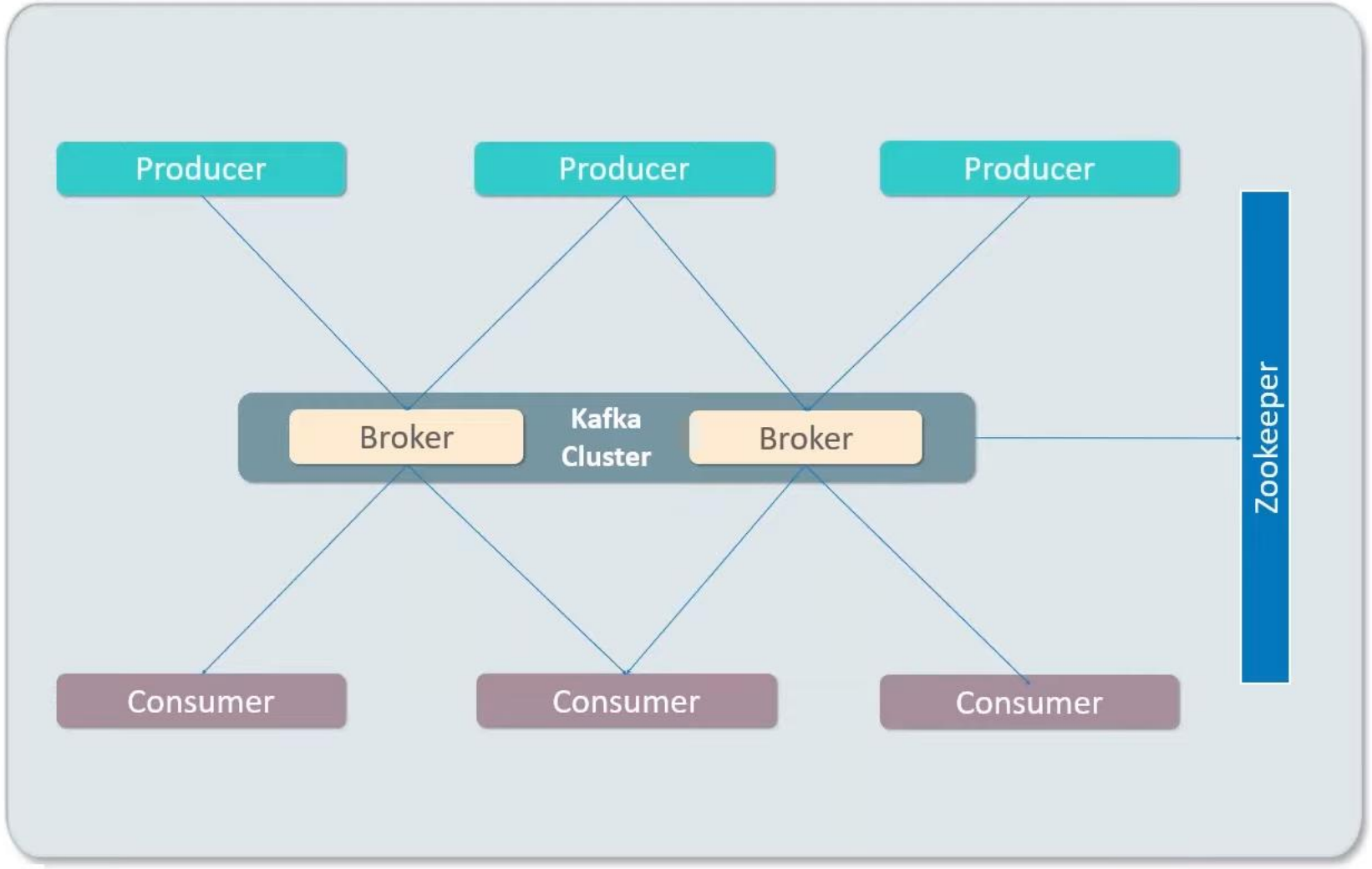
Topic

A ***topic*** is a category or *feed name* to which *records are published*

Zookeeper

ZooKeeper is used for managing and coordinating Kafka broker

Kafka Cluster

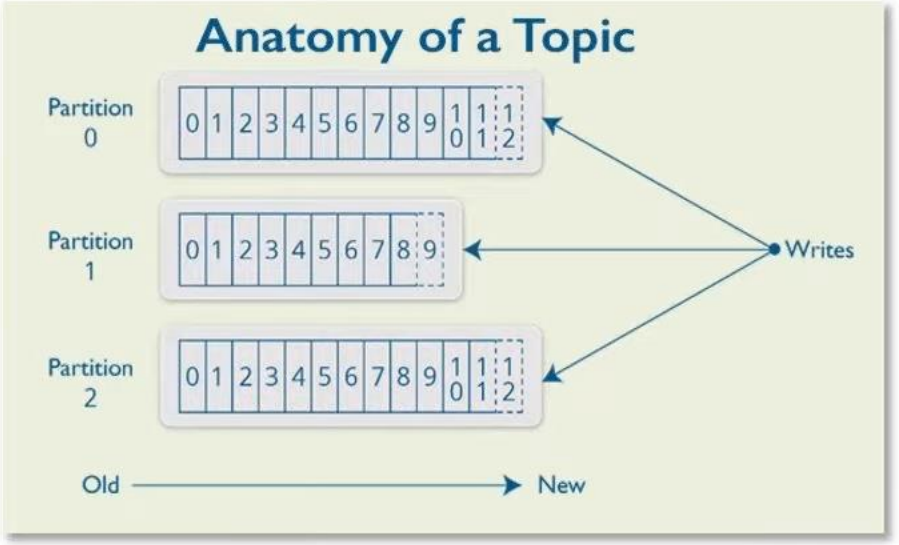


Kafka Features

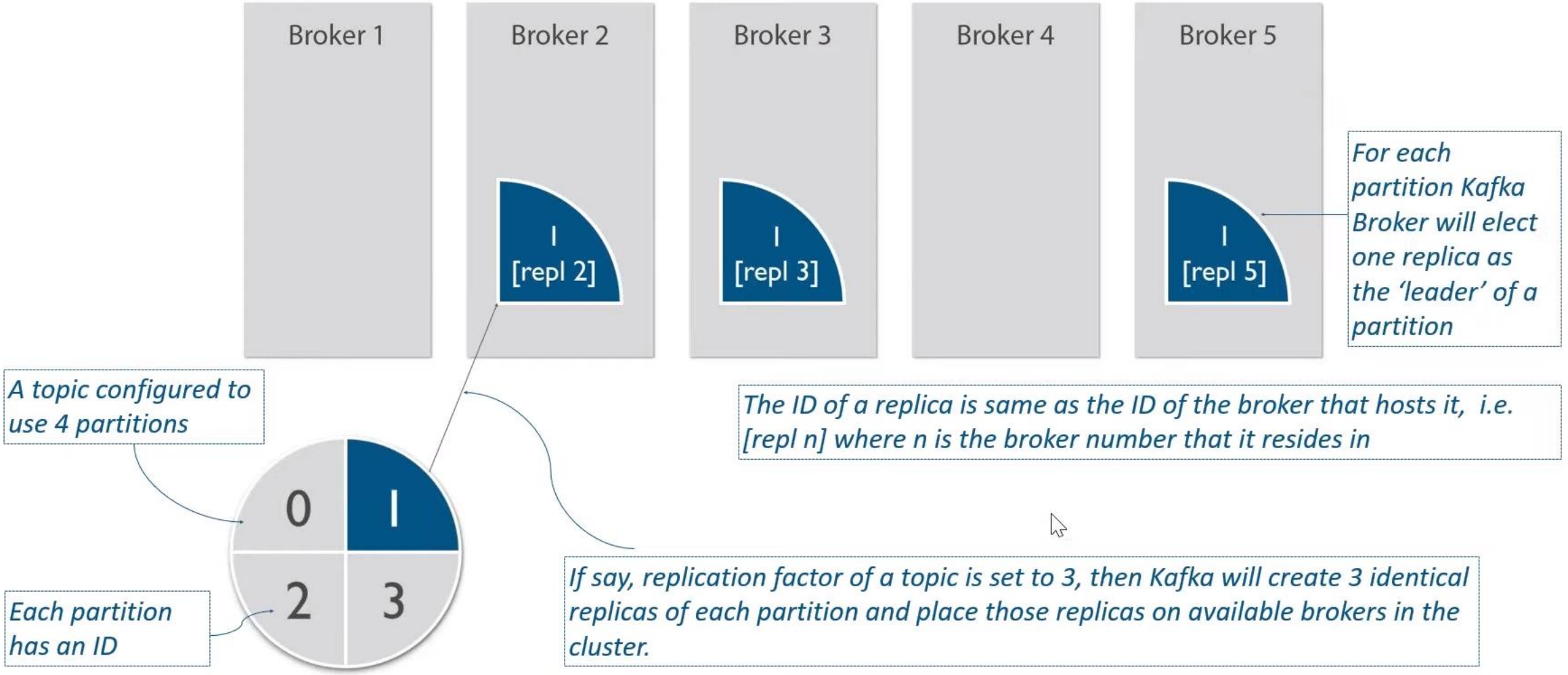
- **High Throughput:** Provides Support for Hundreds of thousands of messages with modest hardware
- **Scalability:** Highly scalable distributed systems with no downtime
- **Data Loss:** Kafka ensures no data loss once configured properly
- **Stream Processing:** Kafka can be used along with real time streaming applications like Spark and Storm
- **Durability:** Provides support to persisting messages on disk
- **Replication:** Messages can be replicated across clusters, which supports multiple subscribers

Kafka Components - Topics and Partitions

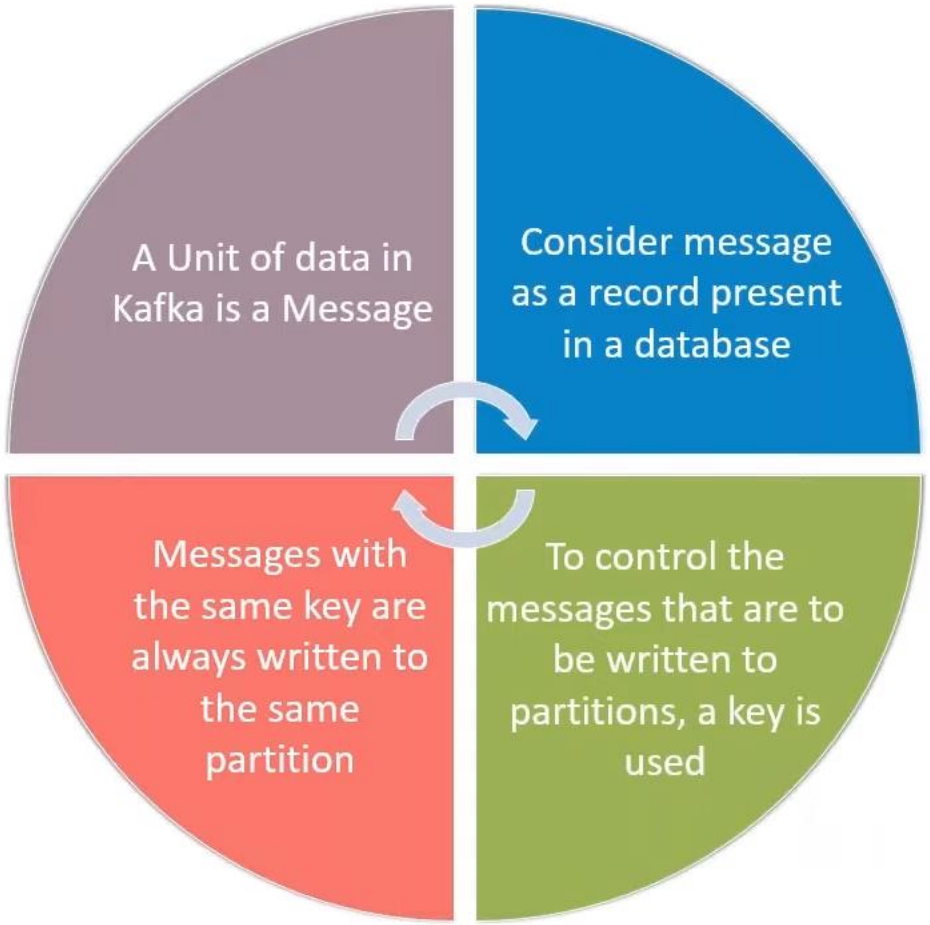
- A *topic* is a category or *feed name* to which *records are published*
- *Topics* are *broken* up into *ordered commit logs* called *partitions*
- Each *message* in a *partition* is assigned a *sequential id* called an *offset*
- Data in a topic is retained for a *configurable period of time*
- *Writes* to a partition *are* generally *sequential* thereby *reducing* the *number of hard disk seeks*
- *Reading messages* can either be from *beginning* & *also* can *rewind* or *skip* to any *point* in partition by *giving* an *offset value*



Kafka Components - Topics, Partitions & Replicas



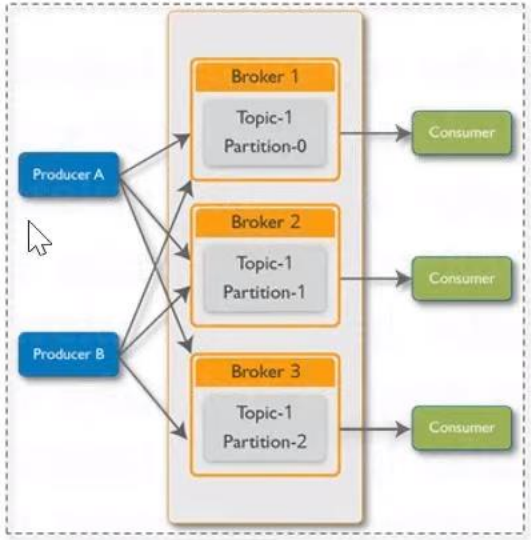
Kafka Components - Messages



Kafka Components - Producer

1

Producer (publisher or writer) publishes a new message to a **specific topic**

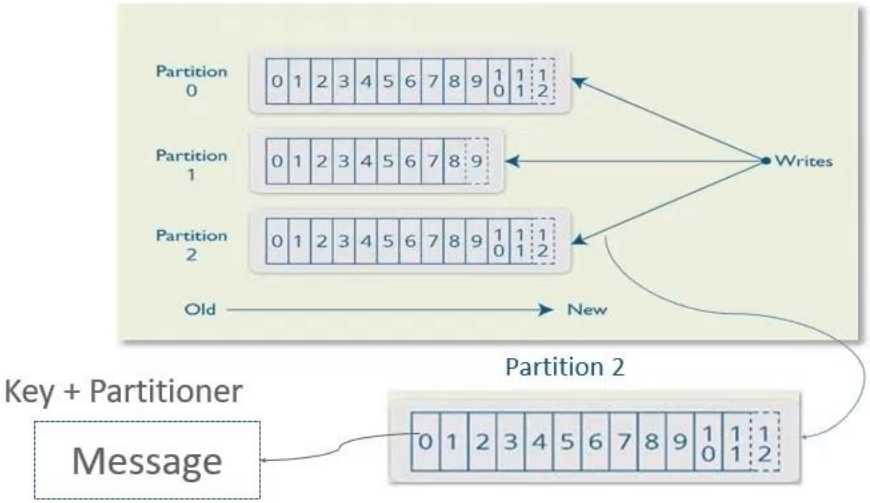


2

The producer does not care what partition a specific message is written to and will balance messages over every partition of a topic evenly

3

Directing messages to a partition is done using the **message key** and a **partitioner**, this will generate a hash of the key and map it to a partition

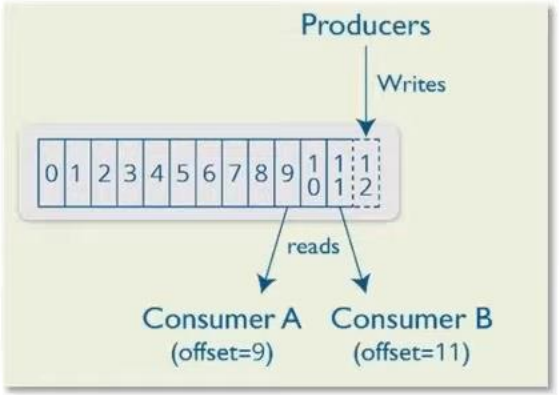


4

Every message a producer publishes in the form of a **key : value** pair

Kafka Components - Consumer

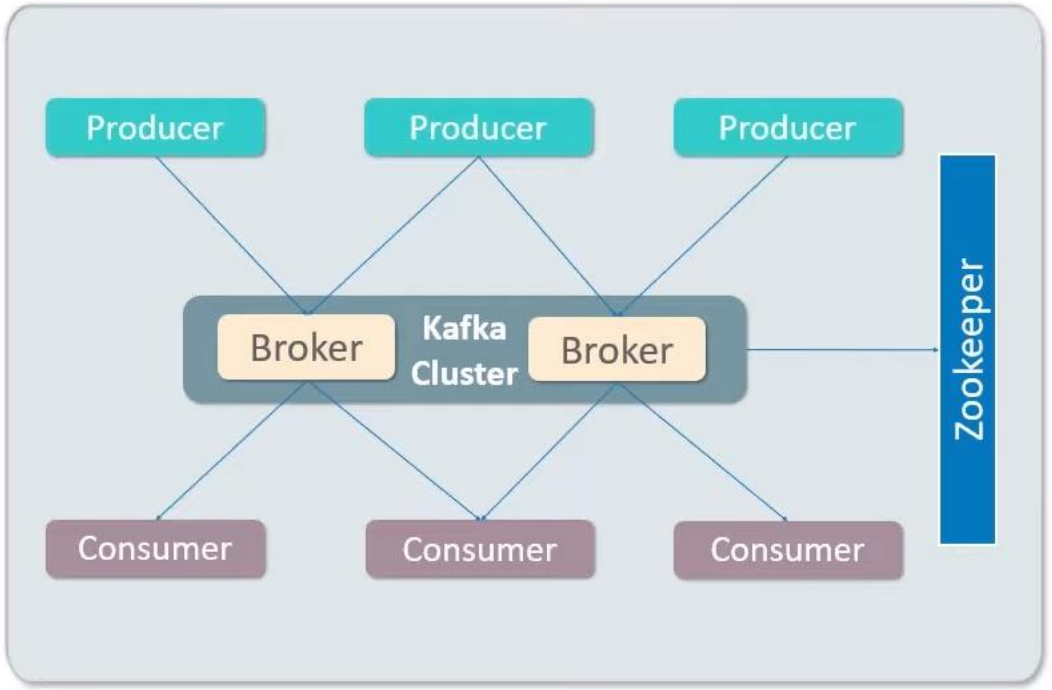
- *Consumers*(subscribers or readers) read messages
- The consumer subscribes to one or more topics and reads the messages sequentially
- The consumer keeps track of the messages it has consumed by keeping track on the offset of messages
- The *offset* is bit of metadata(an integer value that continually increases)that Kafka adds to each message as it is produced
- Each partition has a *unique offset* which is stored
- With the offset of the last consumed message, a consumer can *stop and restart without losing its current state*



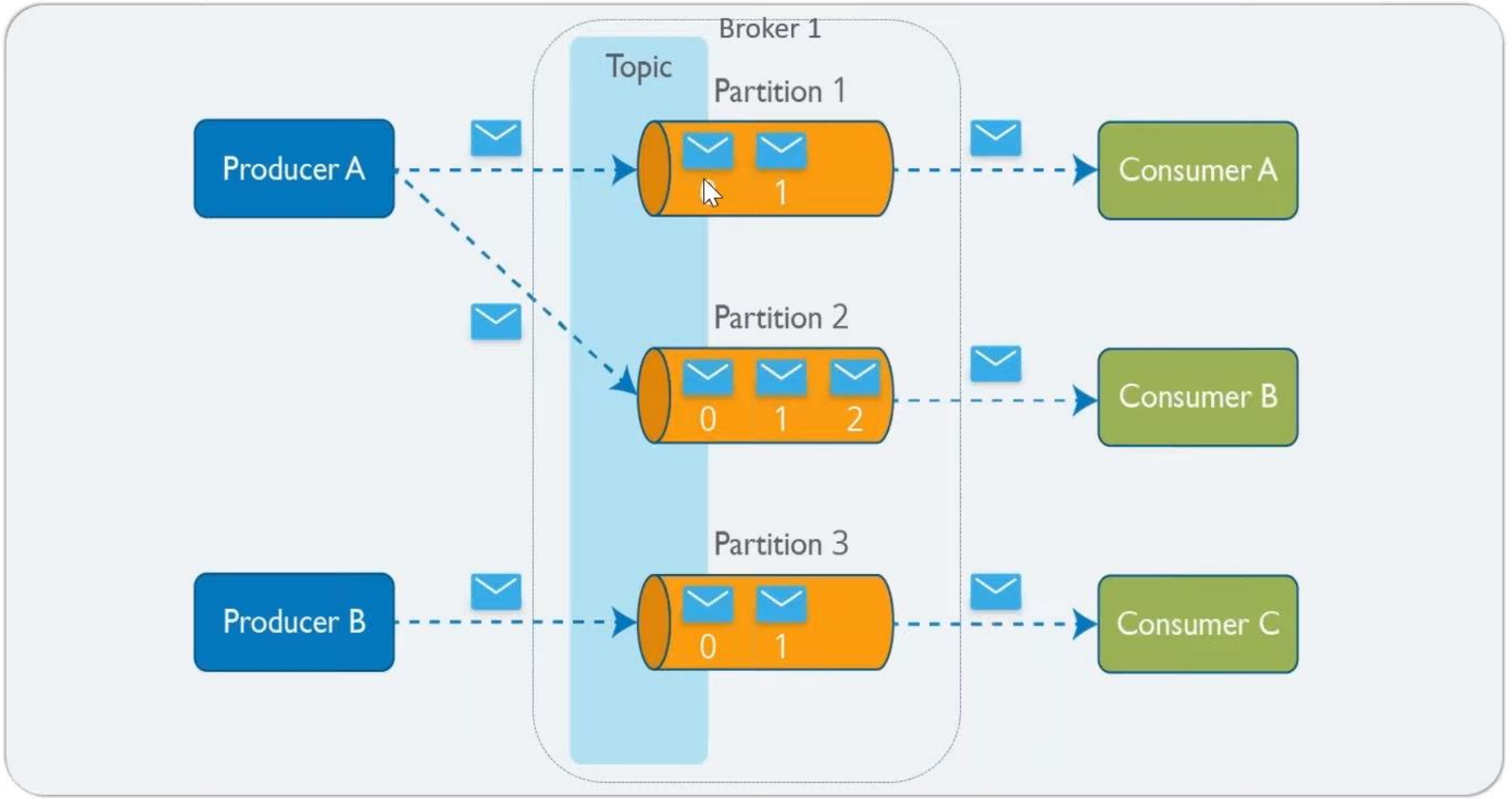
Kafka Components - ZooKeeper

ZooKeeper is used for managing and coordinating Kafka broker

- Zookeeper service is mainly used for co-ordinating between brokers in the Kafka cluster
- Kafka cluster is connected to ZooKeeper to get information about any failure nodes



Kafka Architecture



Let’s see some Use Cases of Kafka

Kafka - Use Cases

- Applications can produce messages using Kafka, without being concerned about the format of the messages
- Messages are sent and handled by a single application that can read all of them consistently, including :
  - A common formatting of messages using a common look
  - Send multiple messages in a single notification
  - Receive messages in a way that meets the users preferences

Kafka - Use Cases

- Originally Kafka was designed at LinkedIn, to track user activity
- When a user interacts with frontend applications, which generates messages regarding actions the user is taking
- Kafka keeps track of simple information like click tracking to complex information like data in a user's profile

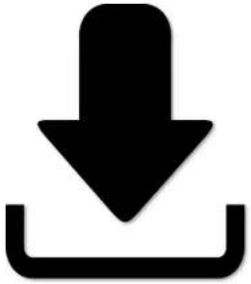
Kafka - Use Cases

- Kafka is also ideal for collecting application's and system metrics and logs
- Applications publish metrics on a regular basis to a Kafka topic, and those metrics can be consumed by systems for monitoring and alerting
- Log messages can be published in the same way and routed to dedicated log search systems like Elasticsearch or security analysis applications

Kafka - Use Cases

- Database changes can be published to Kafka and applications can easily monitor this stream to receive live updates as they happen
- Kafka replicates database updates to a remote system for consolidating changes from multiple applications in a single database view
- Durable retention becomes useful providing a buffer for the changelog, meaning it can be replayed in the event of a failure of the consuming applications
- Log-compacted topics can be used to provide longer retention by only retaining a single change per key

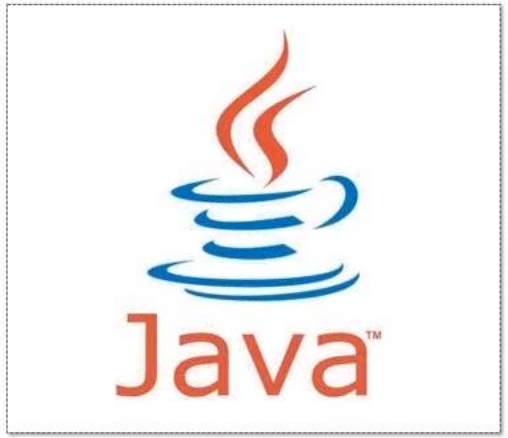
Kafka - Use Cases



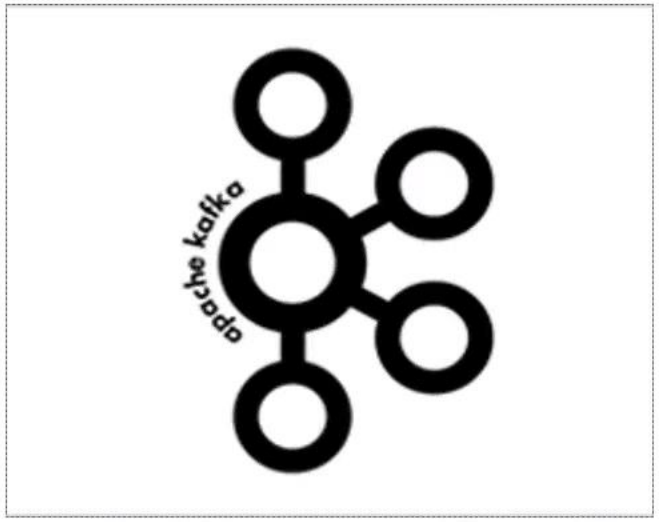
- Stream processing term is typically used to refer applications that provide similar functionality to map/reduce processing in Hadoop
- Stream processing operates on data in real-time, as quickly as messages are produced :
  - Write small applications to operate on Kafka messages,
  - Performing tasks such as counting metrics
  - Partitioning messages for efficient processing by other applications

Getting Started with Kafka

- **Prerequisites :**



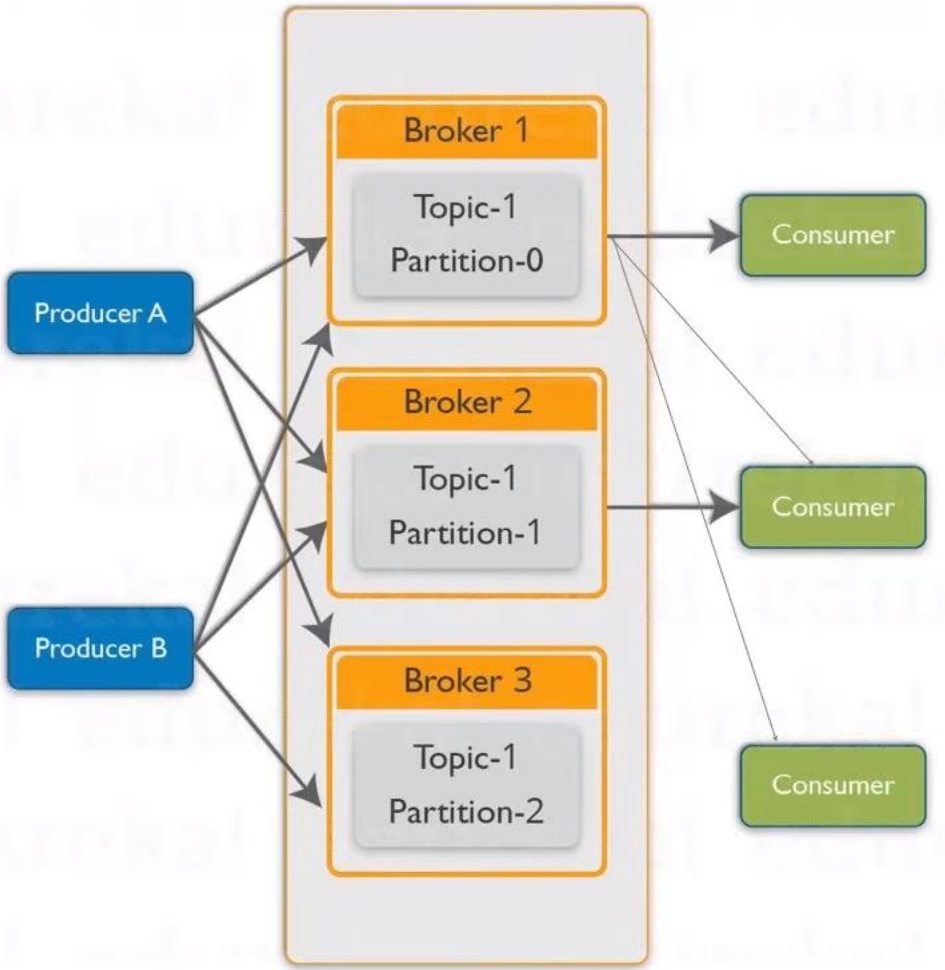
- **Components :**



Let’s Classify Different Types of Clusters in Kafka

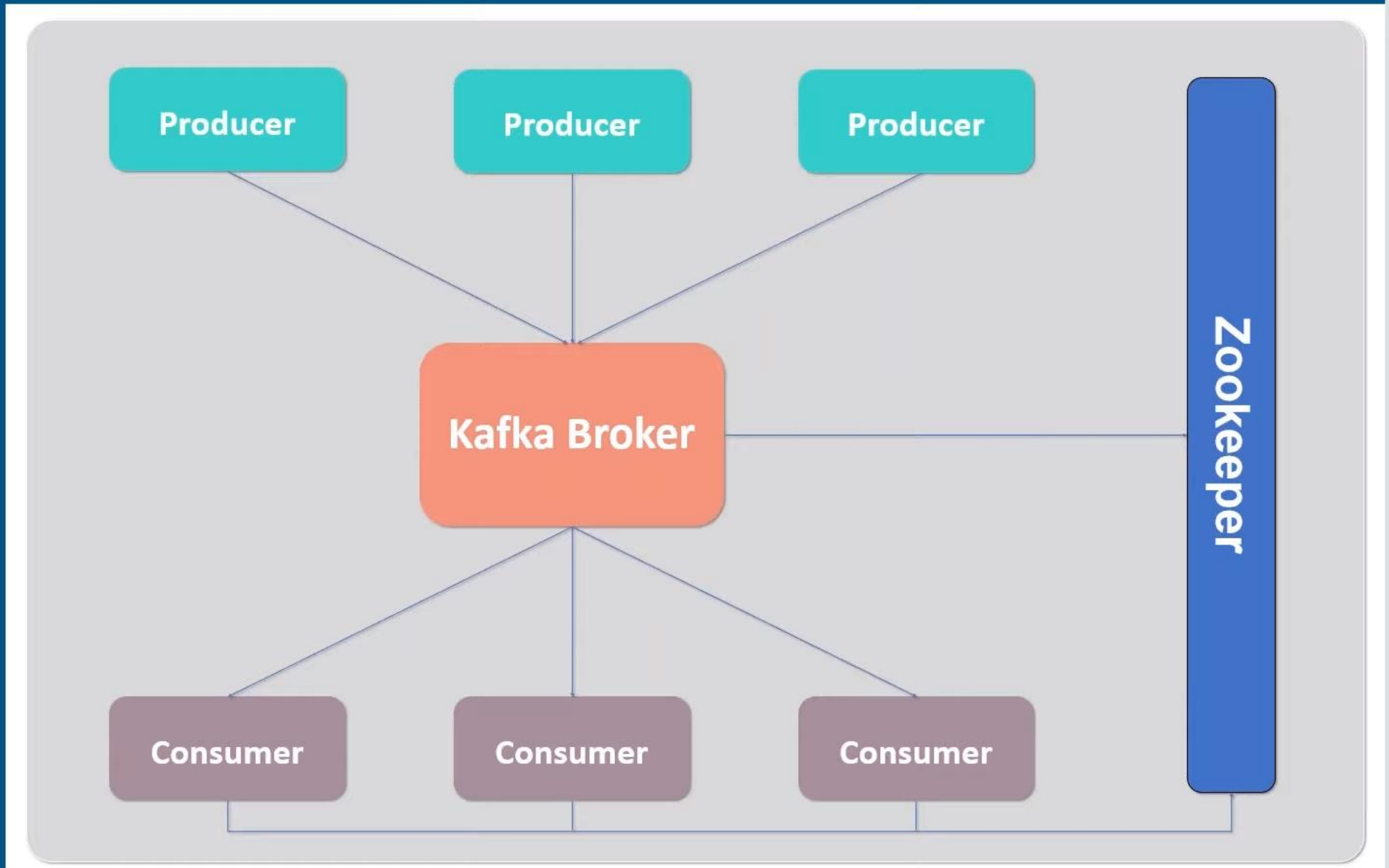
Kafka Cluster

- Kafka brokers are designed to operate as part of a cluster
- One broker will also function as the cluster controller
- Controller is responsible for administrative operations, like
  - Assigning partitions to brokers
  - Monitoring for broker failures in a cluster
- A particular partition is owned by a broker, and that broker is called the leader of the partition
- All consumers and producers operating on that partition must connect to the leader



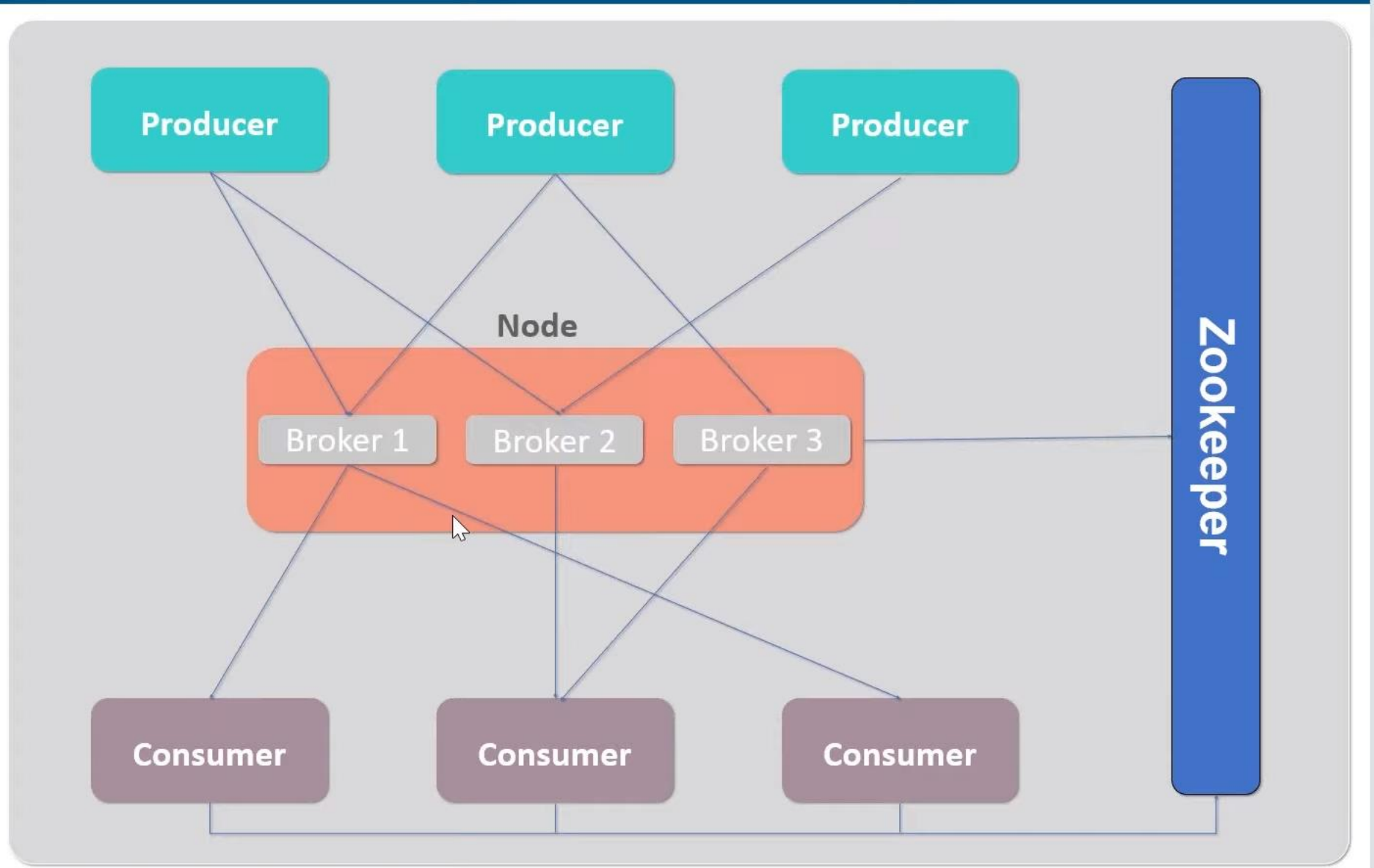
Type of Kafka Clusters

1. **Single Node-Single Broker Cluster**
2. Single Node-Multiple Broker Cluster
3. Multiple Nodes-Multiple Broker Cluster



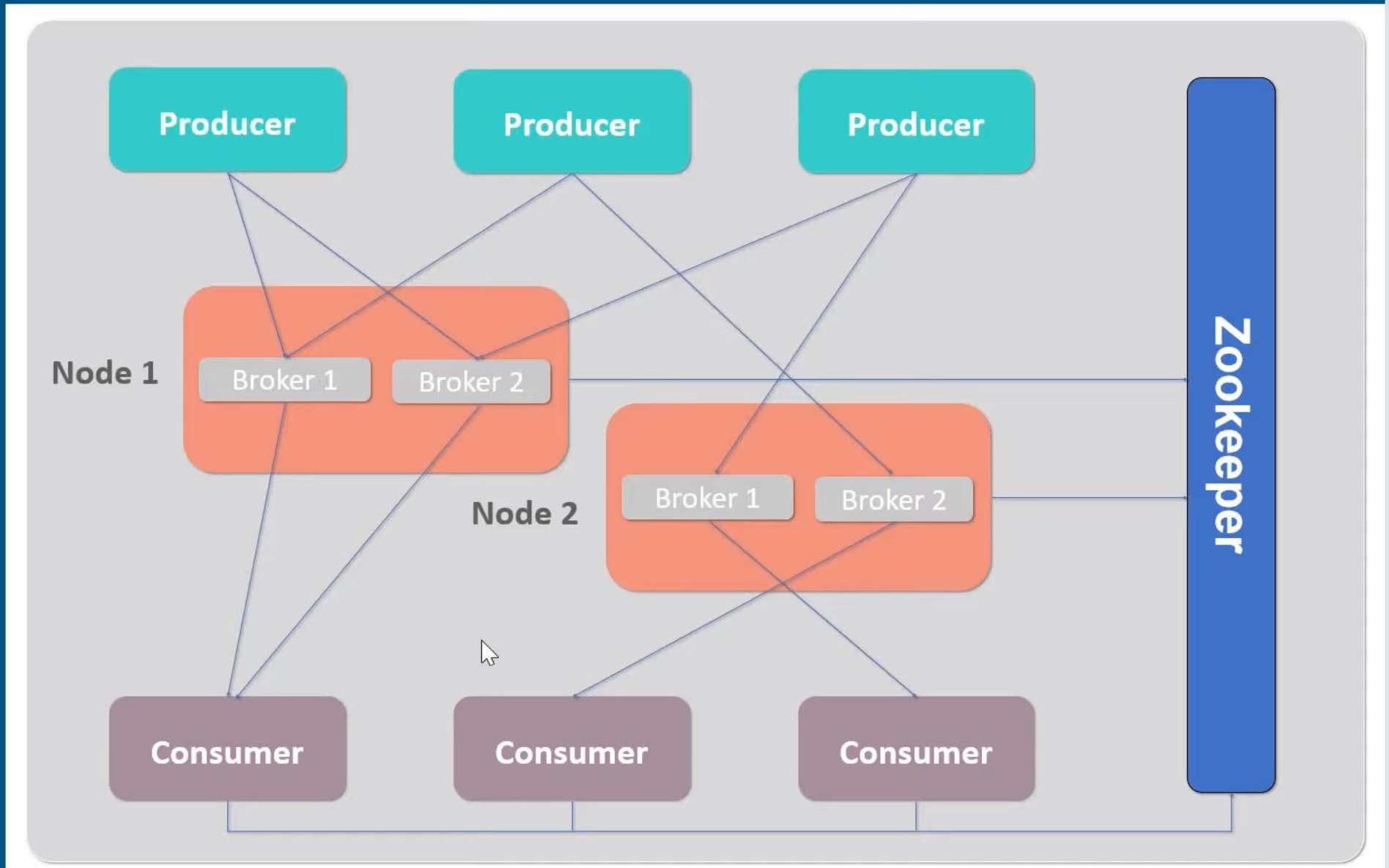
Type of Kafka Clusters

1. Single Node-Single Broker Cluster
2. **Single Node-Multiple Broker Cluster**
3. Multiple Nodes-Multiple Broker Cluster



Type of Kafka Clusters

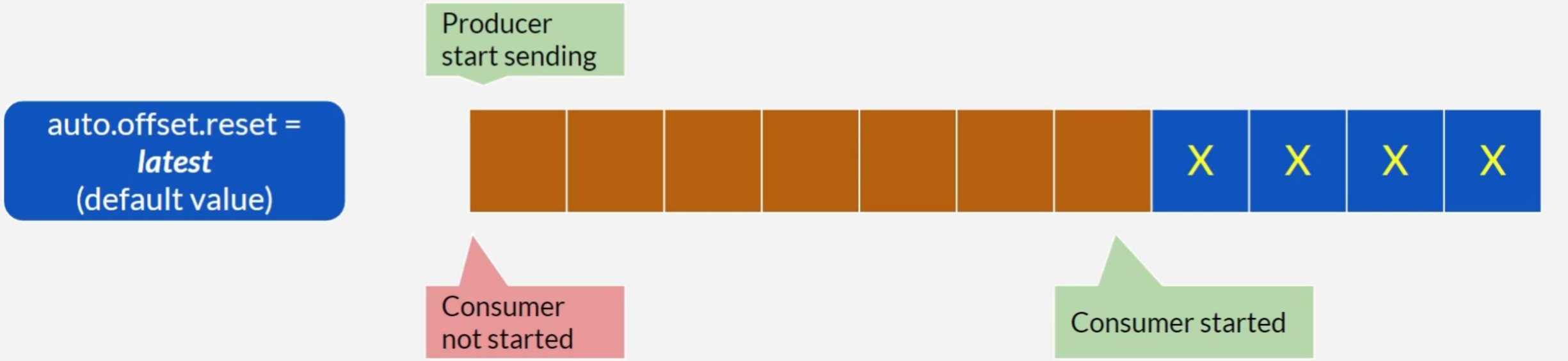
1. Single Node-Single Broker Cluster
2. Single Node-Multiple Broker Cluster
3. **Multiple Nodes-Multiple Broker Cluster**

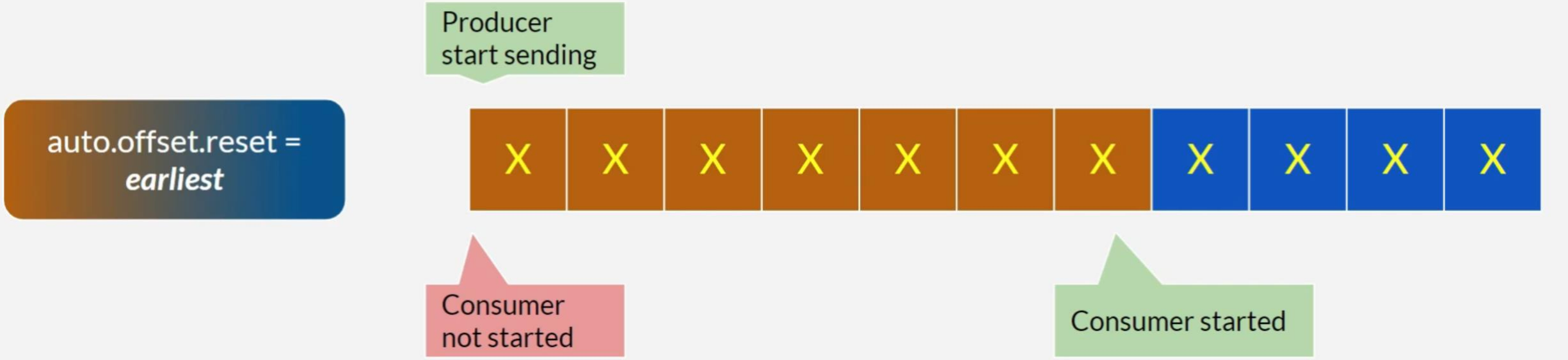


Getting Started with KAFKA & Spring Boot

Producer & Consumer

Consumer Offset on First Run



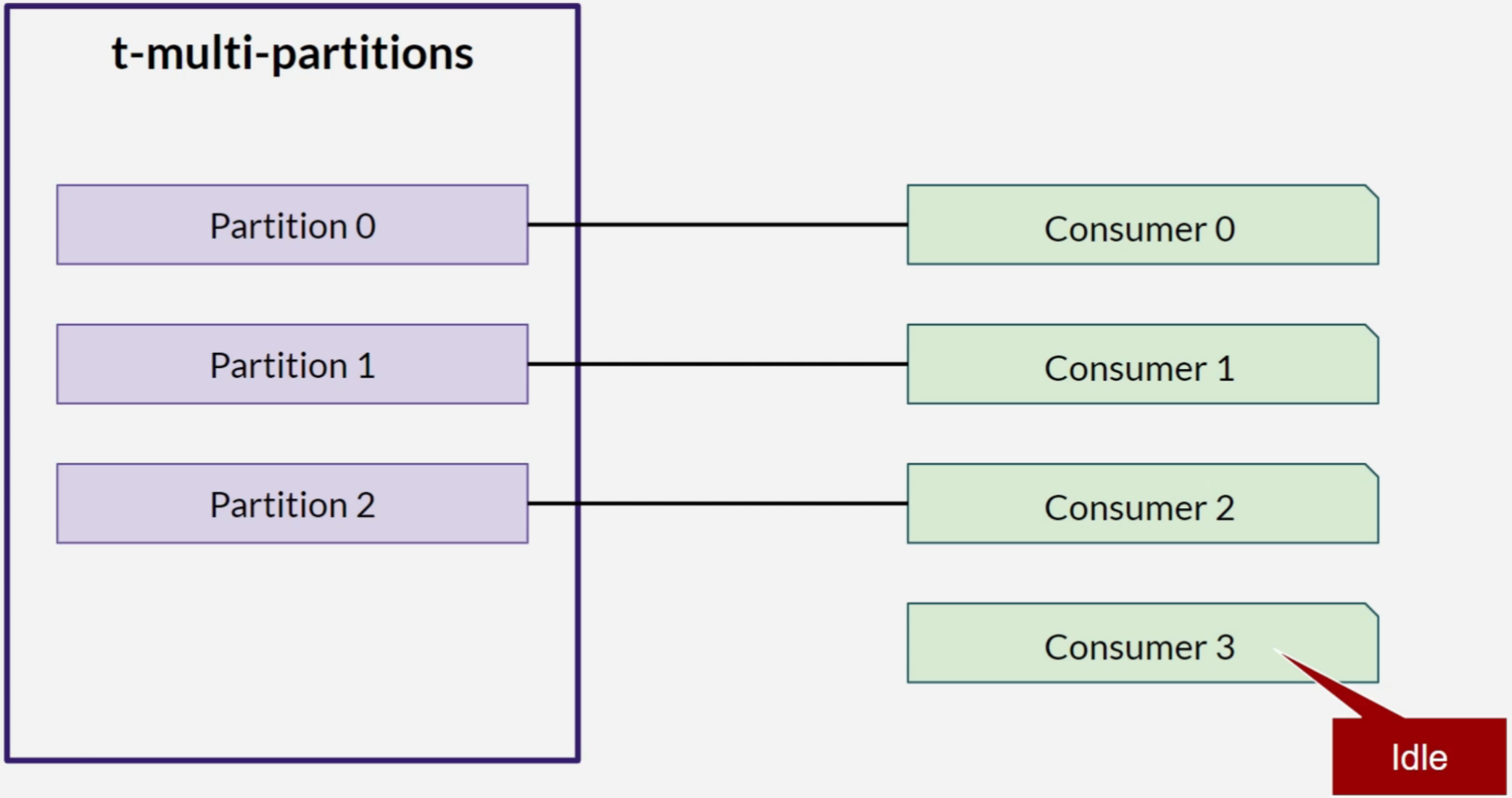


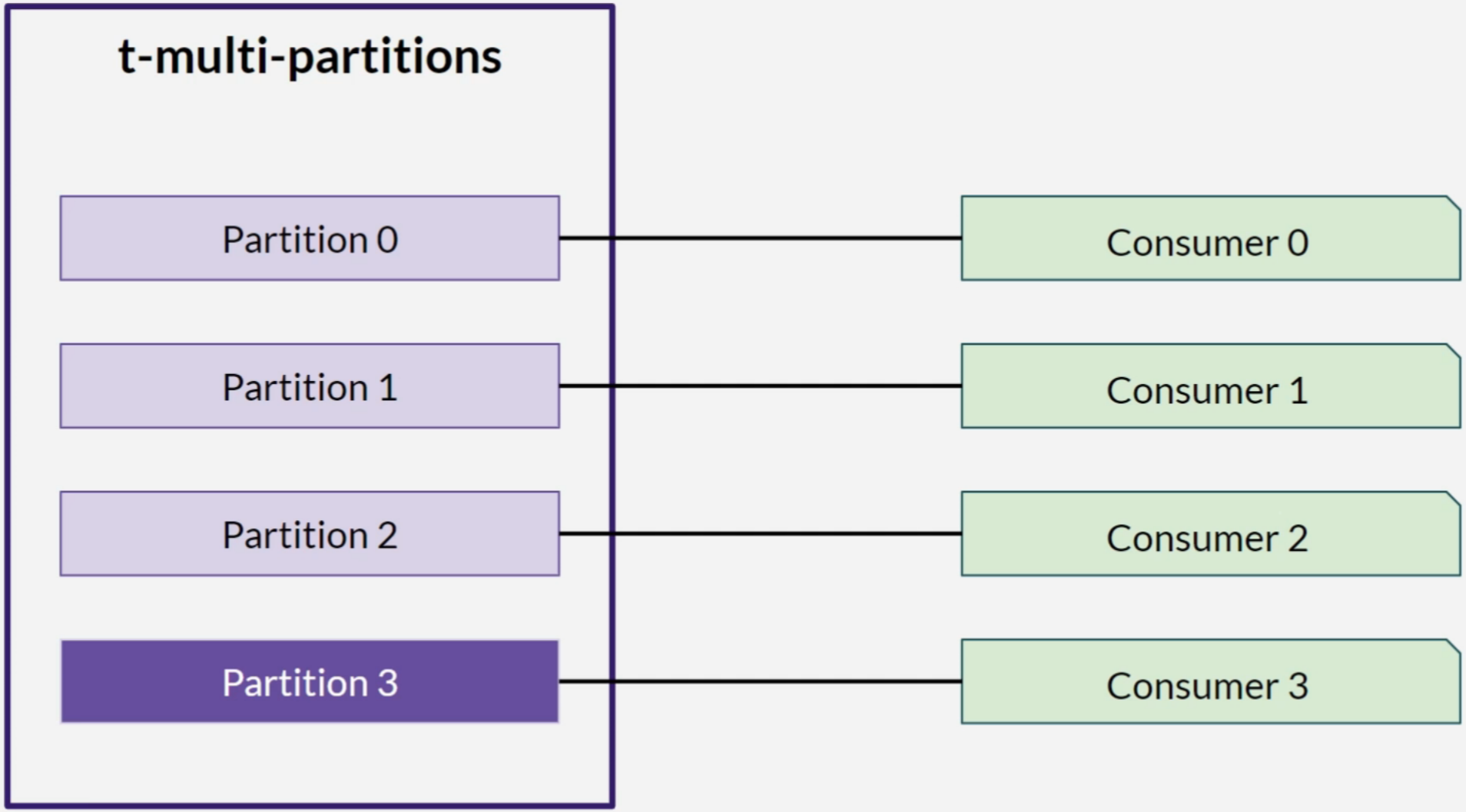
Message with key

- Same key goes to same partition
- As long as we don't change the partition number

Multiple Consumers, Do we need it?

- Publisher (producer) works faster than consumer
- Subscriber (consumer) bottleneck

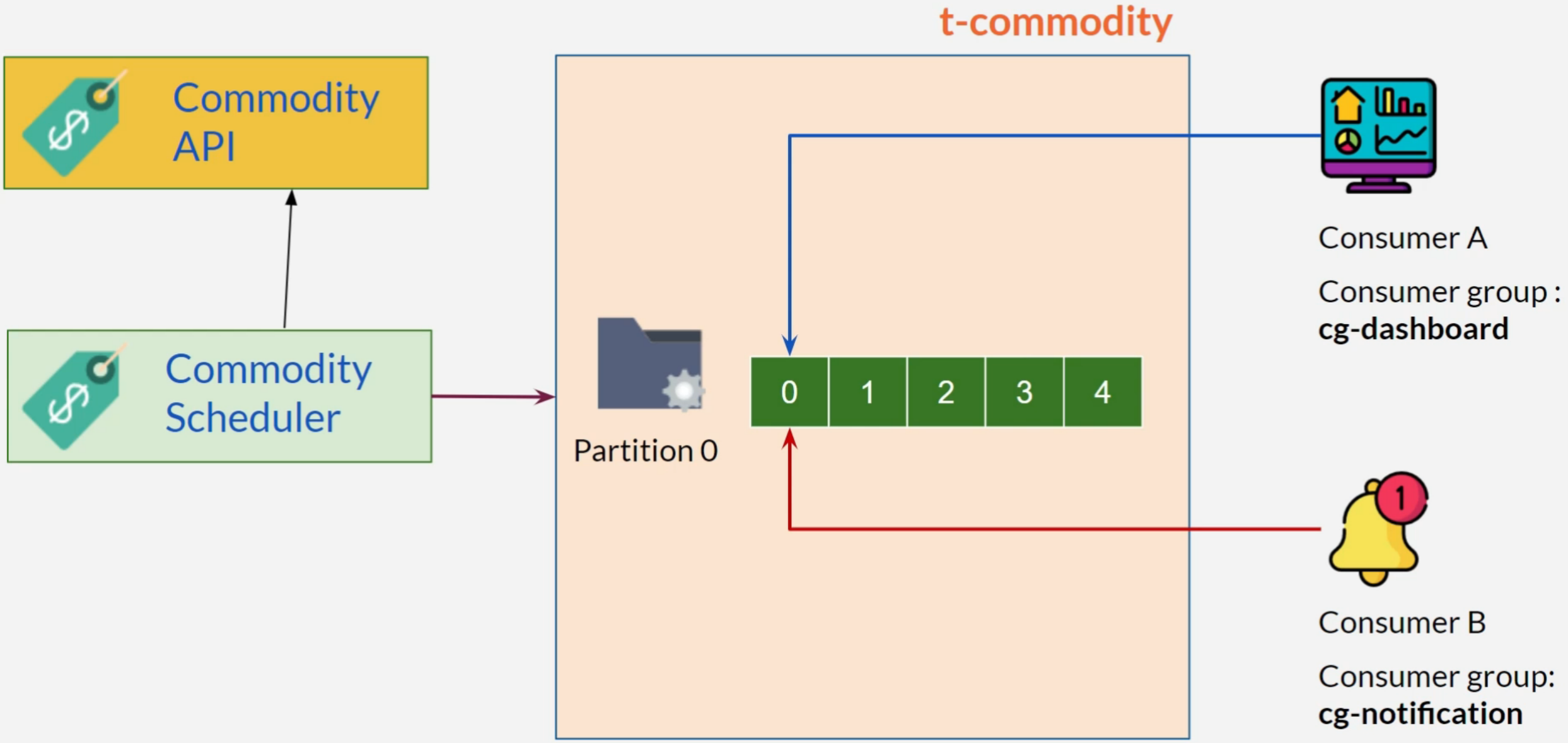

t-multi-partitions
Partition 0
Partition 1
Partition 2
Consumer 0
Consumer 1
Consumer 2
Consumer 3
Idle

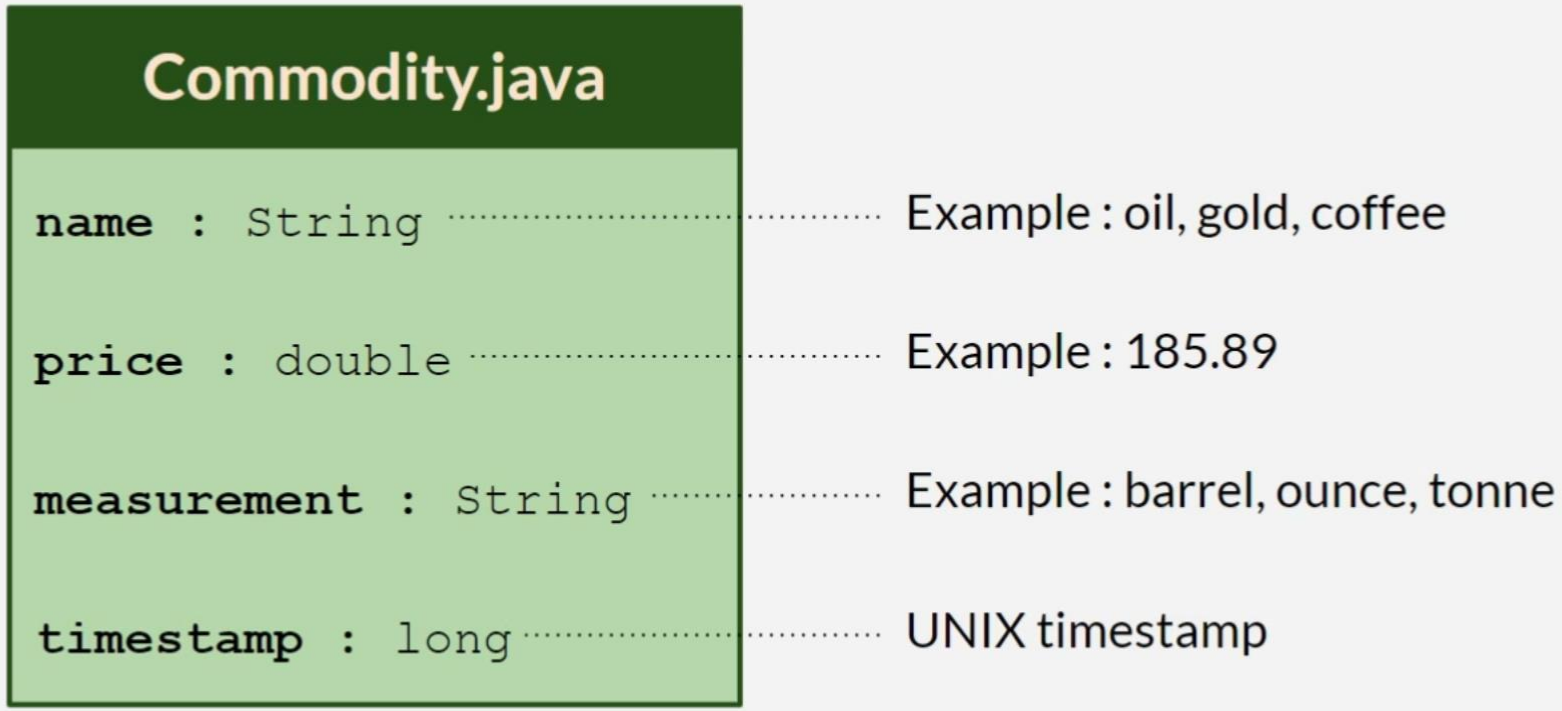

t-multi-partitions
Partition 0
Partition 1
Partition 2
Partition 3
Consumer 0
Consumer 1
Consumer 2
Consumer 3

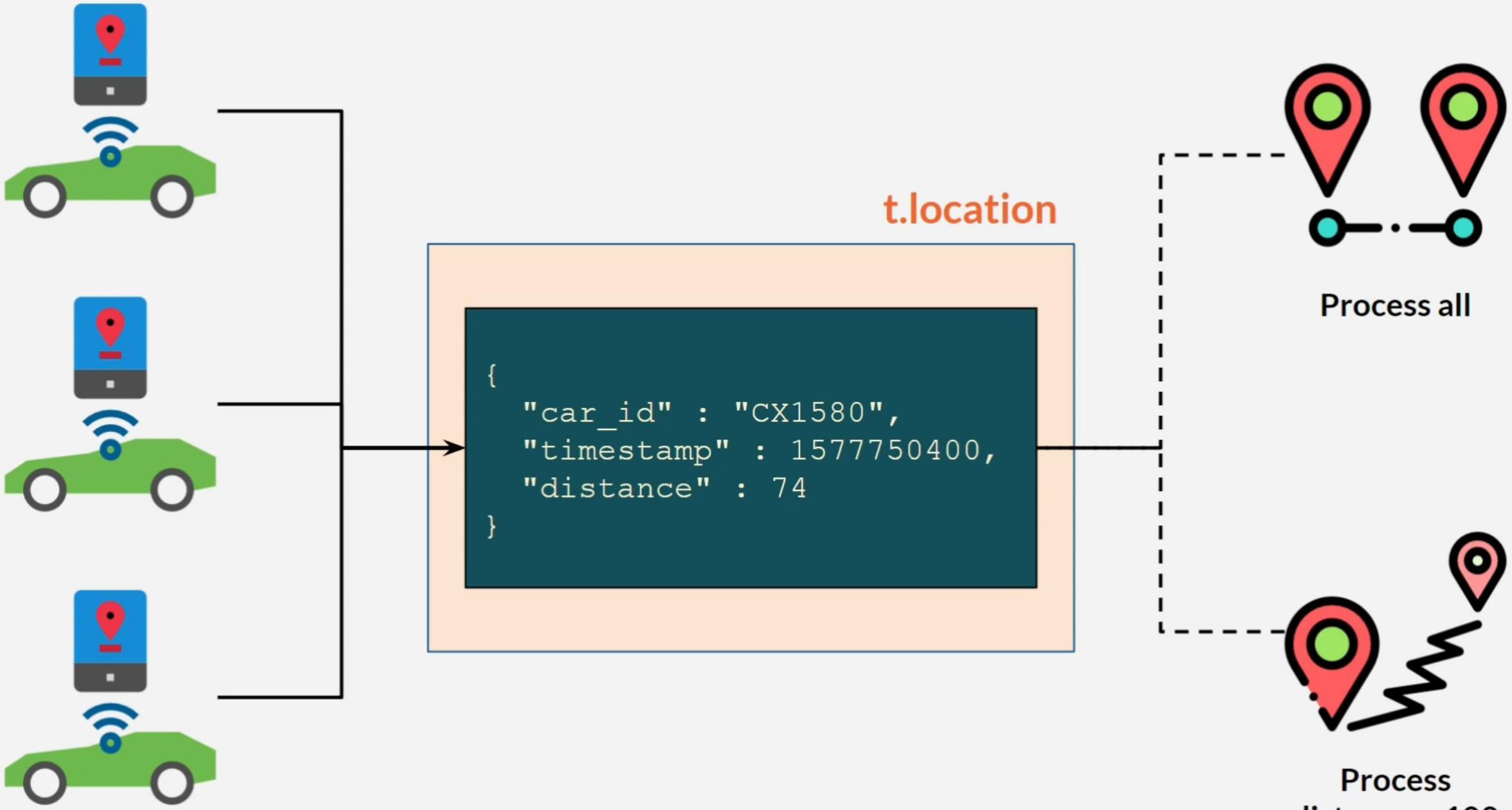
What about deleting partition?

- Delete topic is OK
- Can't delete partition
- Can cause data loss
- Wrong key distribution
- Decrease partition > delete & recreate topic
- Real life usually use kafka native linux

Kafka Schema




Commodity.java
name : String
Example : oil, gold, coffee
price : double
Example : 185.89
measurement : String
Example : barrel, ounce, tonne
timestamp : long
UNIX timestamp


t.location
{
"car_id" : "CX1580",
"timestamp" : 1577750400,
"distance" : 74
}
Process all
Process
distance > 100

Idempotent Consumer

- Duplicate message is OK
  - Outcome of processing message always same even for duplicate messages
  - ex:- update search engine index
- duplicate Message is dangerous
  - Duplicate transaction
  - ex:- create (duplicate) payment
  - Filter out duplicate messages

How to duplicate?

- Use database for permanent unique values
- Use cache for temporary unique values
- Cache
  - Better performance
  - Automatically remove data after certain time
  - Example: Redis
- Database
  - Might publish duplicate after longer period
  - Virtually unlimited storage

id : primary key database (surrogate)

po number : natural key (business)

Event	PR Event ID	PR number
Budget reserve	5551	PR-One
Approval workflow	5552	PR-One
Push notification	5553	PR-One

Kafka record key using ID

Partition 0

Offset 0 : 5551	Offset 1 : 5552

Partition 1

Offset 0 : 5553

5551, 5553, 5552
5553, 5551, 5552

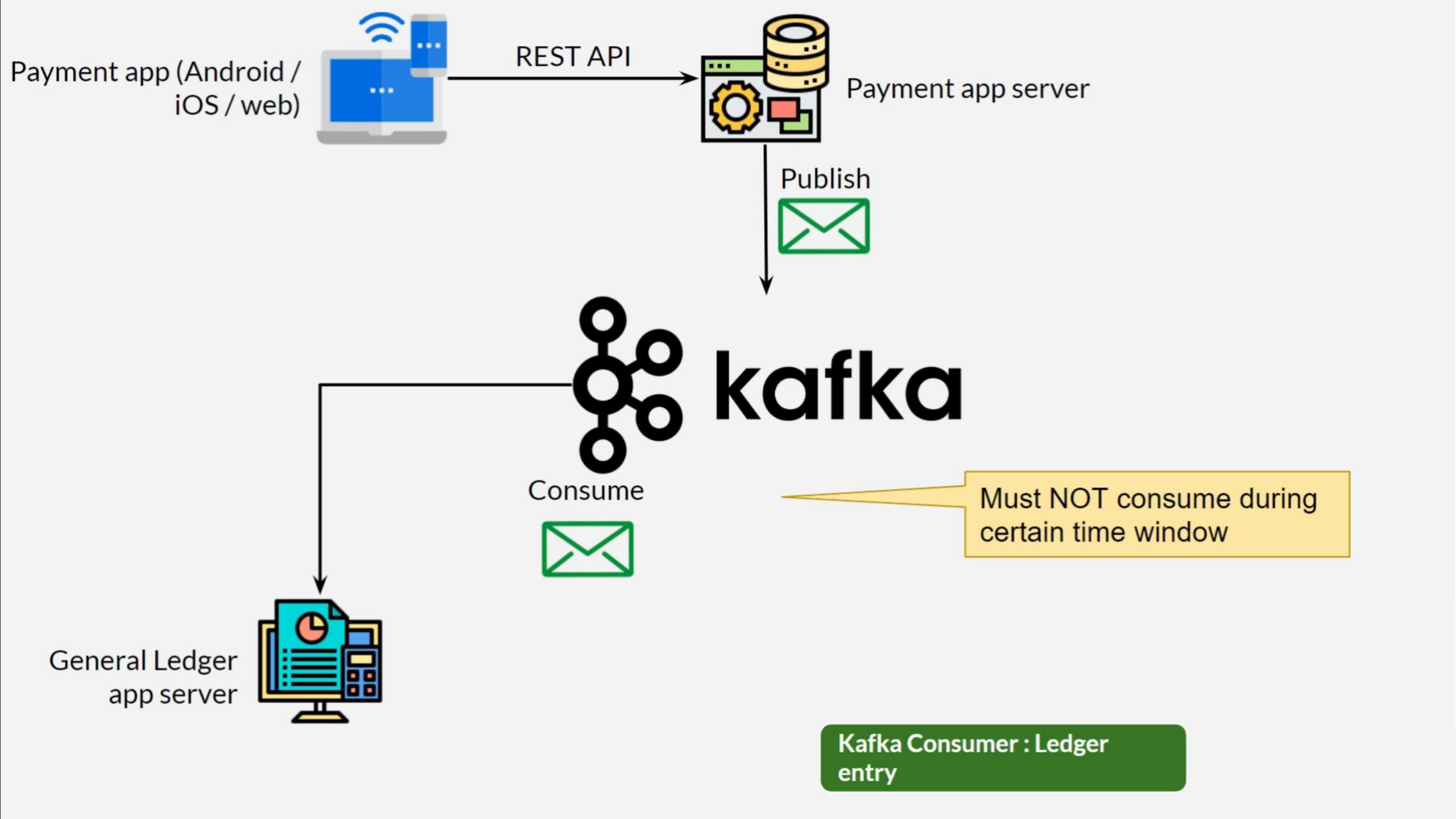
Kafka record key using PR Number

Partition 0

Offset 0 : PR-One 5551	Offset 1 : PR-One 5552	Offset 2: PR-One 5553

Partition 1

Scheduling Consumer


Payment app (Android / iOS / web)
REST API
Payment app server
Publish
kafka
Consume
Must NOT consume during certain time window
General Ledger app server
Kafka Consumer : Ledger entry

Payment app (Android / iOS / web)

REST API

Payment app server

Must live 24/7

Must live 24/7

Publish

kafka

Consume

Must NOT consume during certain time window

General Ledger app server

Scheduler : summarize daily ledger

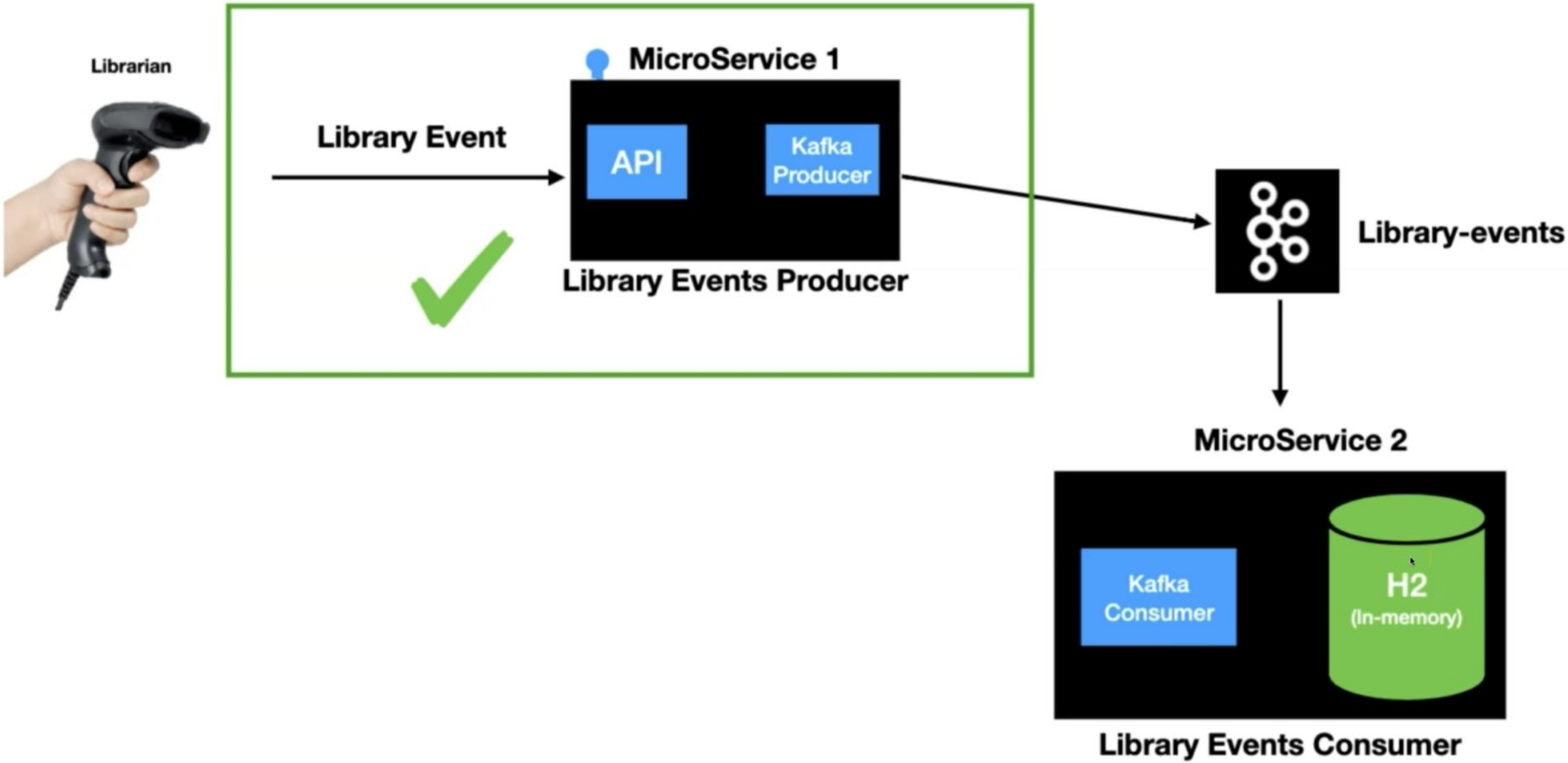
REST API

Auto off / on

Scheduler / on demand : report

Kafka Consumer : Ledger entry (turn off)

KAFKA PROJECT


Librarian
MicroService 1
Library Event
API
Kafka Producer
Library Events Producer
Library-events
MicroService 2
Kafka Consumer
H2
(In-memory)
Library Events Consumer


Basic Usage Kafka + Spring Boot
Kafka in Microservice + Kafka Stream
Kafka Schema Registry + Rest Proxy
Persisting data in MongoDB

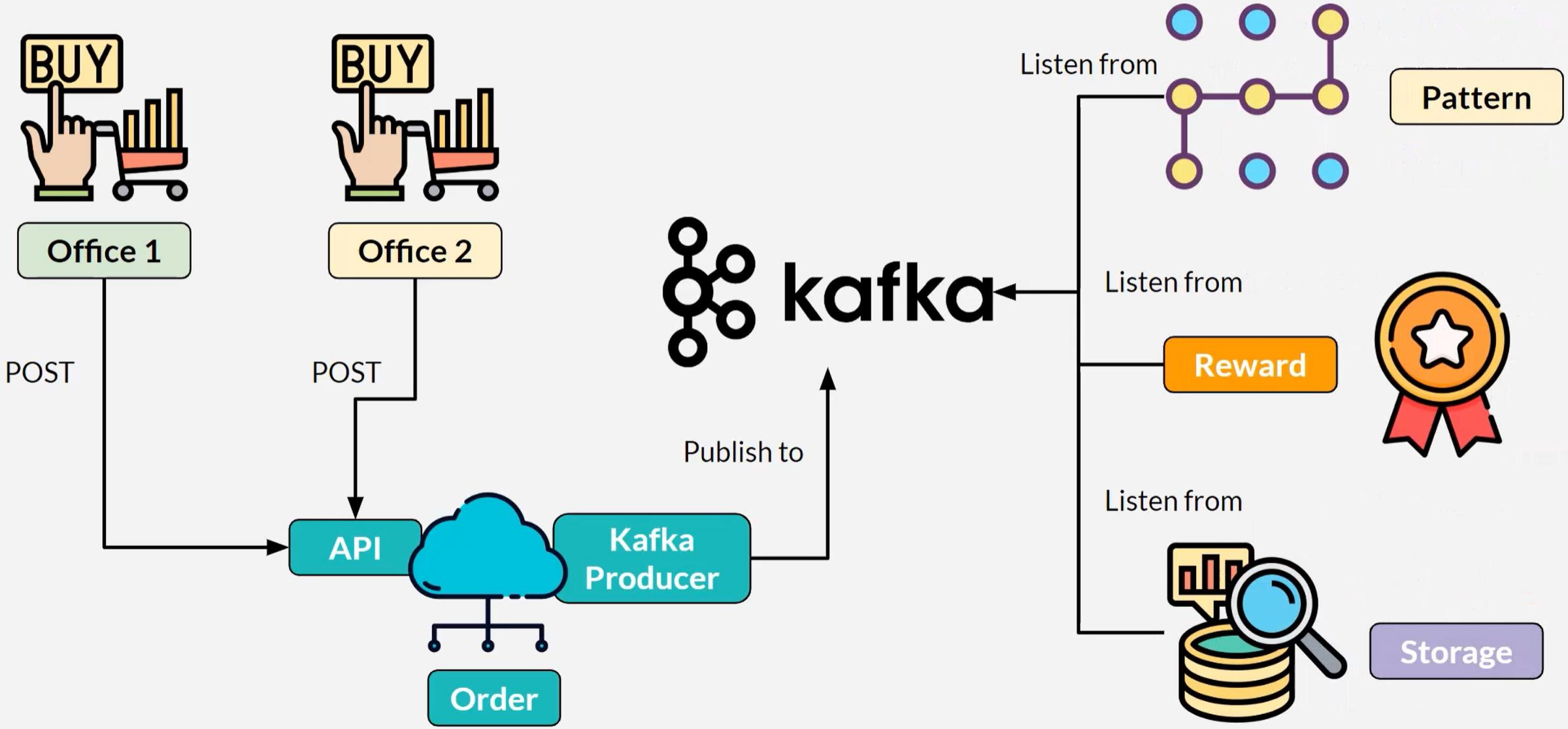
Application Overview

- Oversimplified application
- Kafka usage on real life
- Microservice architecture & pattern

Use Case

- Commodity trading company with multiple branches
- Branch Submit purchase order to head office
- After Process
  - Pattern Analysis
  - Reward
  - Bigdata storage/Persist data
- Speed is the key
- Branch office submit each order once
- Head Office process using kafka

Application Architecture

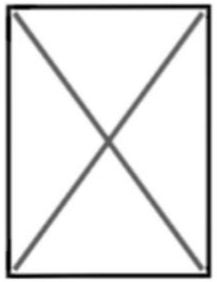


Order

Order location: Indonesia

Credit Card Number: 1958 2850 6094 3758

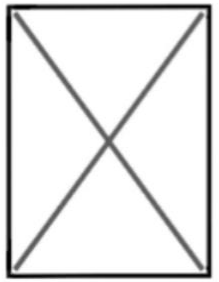
Items



A beautiful book

$14

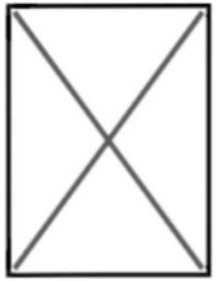
Quantity: 3



An exotic fruit

$3

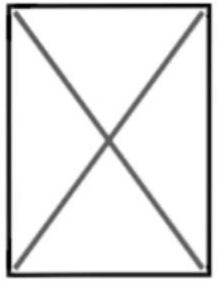
Quantity: 3



A mysterious box

$14

Quantity: 1



A luxury dress

$26

Quantity: 2

Submit Order

The need of organizing code

- Many codebase
- Applications
- Easy to work with
- Easy to future changes
- Easy transfer knowledge & employee onboarding
- Multiple coding & source code organizations
- patterns for code structure & organization

Hexagonal Architecture

User Interface
(API Consumer)

<<inbound>>
REST API

Other Service
API

Message broker

<<inbound>>
Listener

<<outbound>>
API client

Business Logic

Message broker

<<outbound>>
Publisher

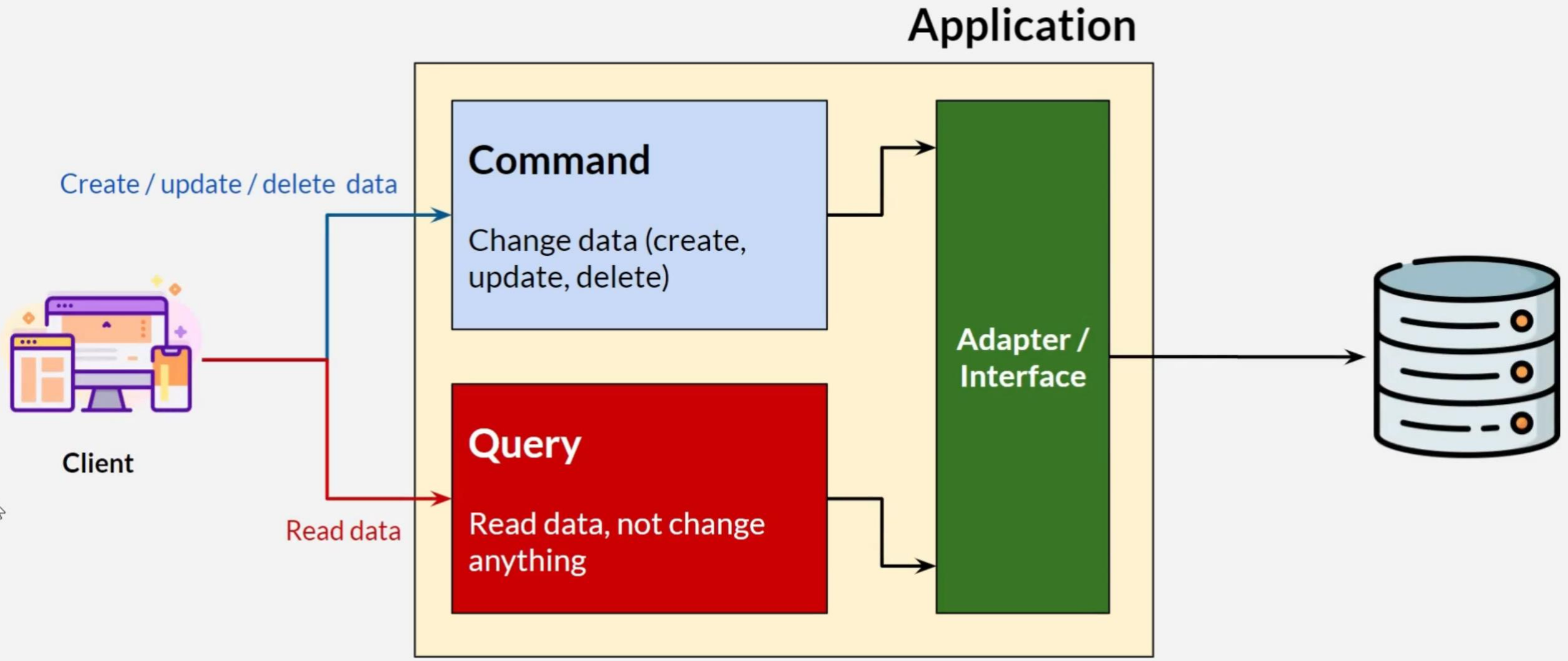
<<outbound>>
Database repository

Database

Hexagonal Architecture

- Benefit: decouple business logic with data access
- Easier to test or change
- Communication using multiple adapters

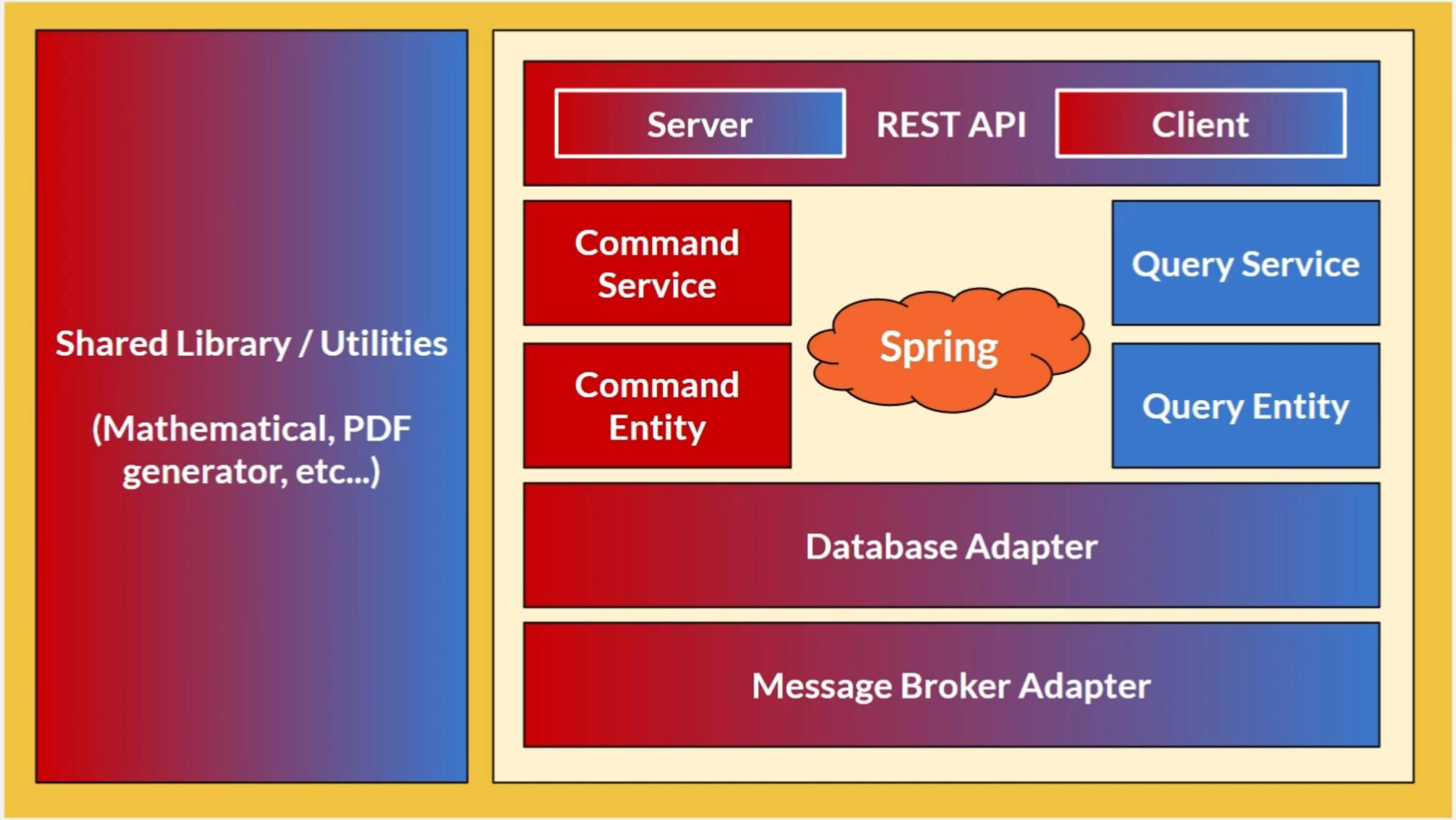
Command Query Separation (CQS)



Command Query Separation

- Command / transaction / modifier / mutator
- Query / view
- Separate, don't mix
- Easier maintenance and change

Application source code



Generate Projects

Use Spring initializr / IDE

- Create 4 projects
  - Group: com.virtusa.kafka
  - Artifact:
    - kafka-ms-order
    - kafka-ms-pattern
    - kafka-ms-reward
    - kafka-ms-storage
  - Package name: com.virtusa.kafka (remove any suffix)
- Spring boot 2.X + Java 11/17

Dependencies

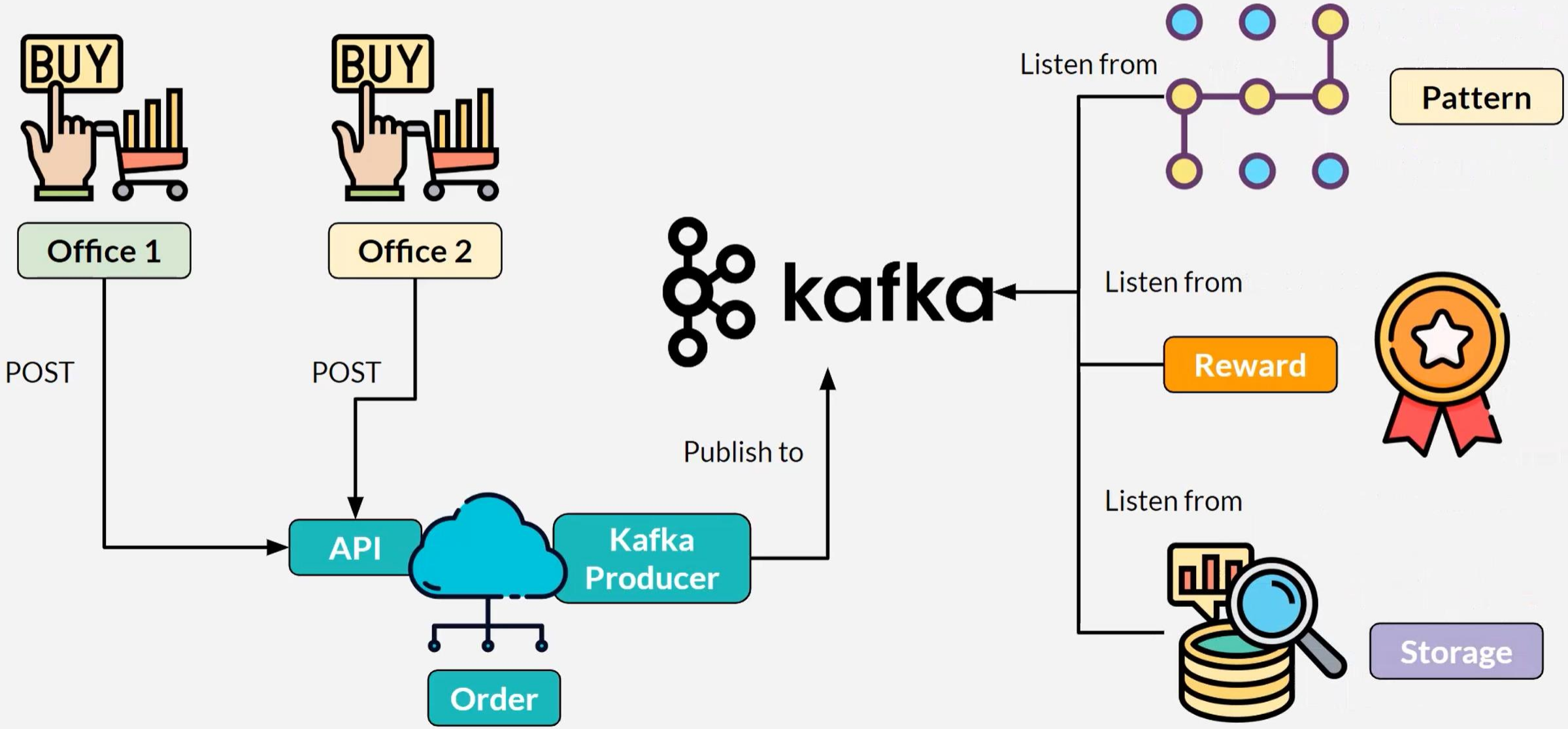
Kafka-ms-order

- Web
- Spring Kafka
- Dev tools
- JPA
- H2 database

Other 3 projects

- Spring Kafka
- Dev tools

Application Architecture



Order - Kafka Message

- ★ order number
- ★ location
- ★ order date & time
- ★ credit card number

Item 1

Item 2

Item 3

Each item has

- name
- price
- quantity

flattened

Kafka Message

Each message has

- order number
- location
- order date & time
- credit card number
- name
- price
- quantity

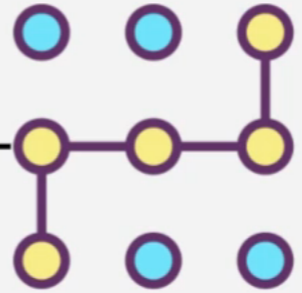
Order - Kafka Message

Kafka Message

Each message has

- order number
- location
- order date & time
- credit card number
- name
- price
- quantity

Aggregate by item name

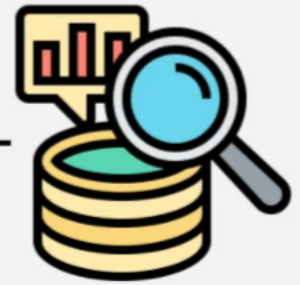


Pattern

Aggregate by location

Reward

No aggregation

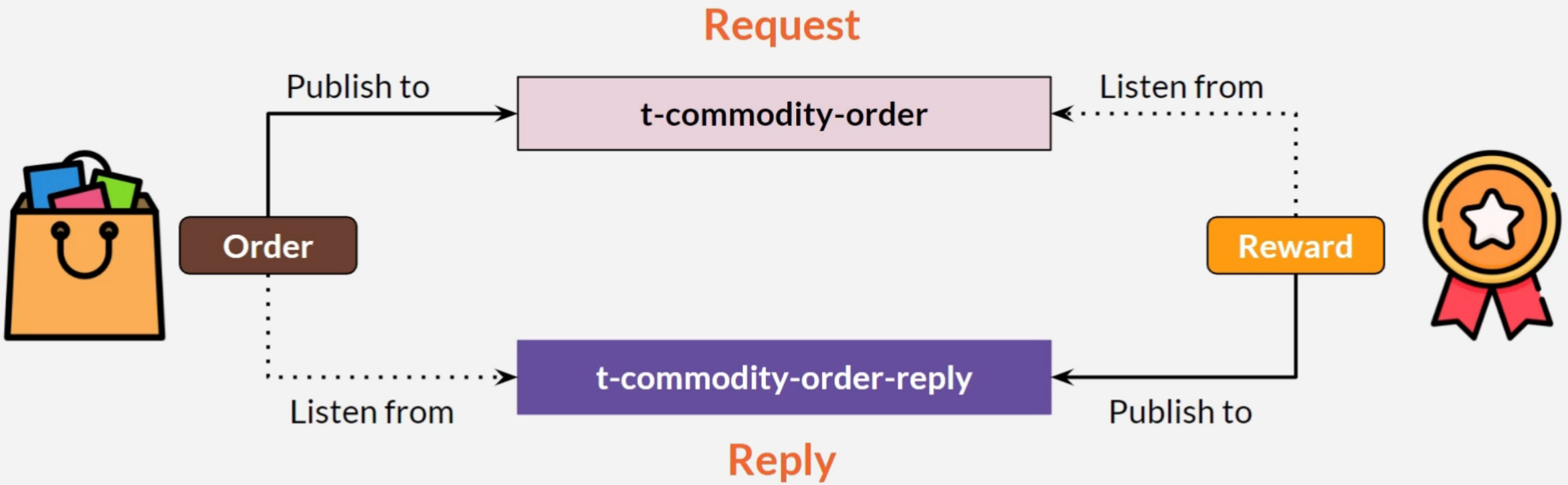


Storage

Order Team Says

- Reward team, please confirm order received
- We provide kafka topic for confirmation

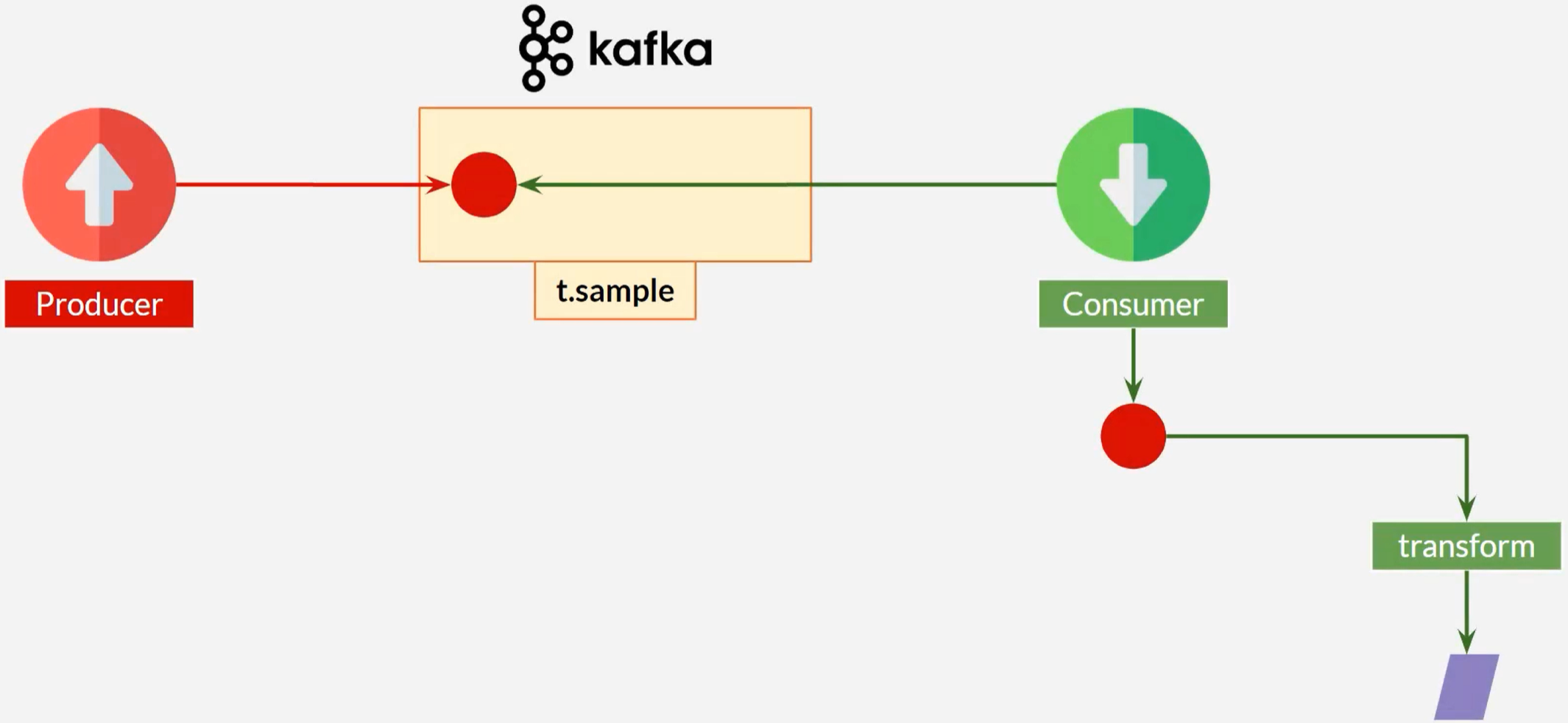
Asynchronous Request / Reply



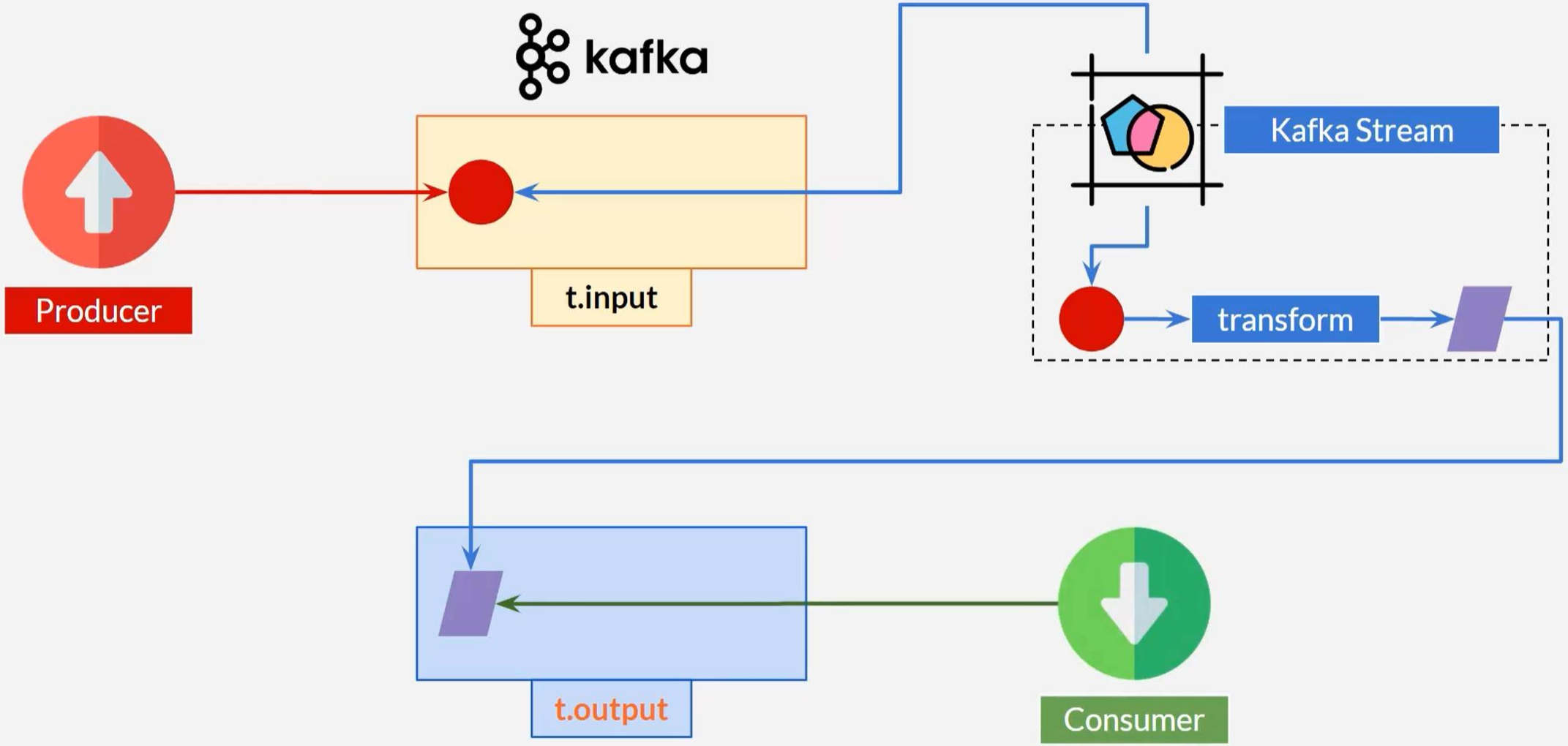
Asynchronous Request / Reply

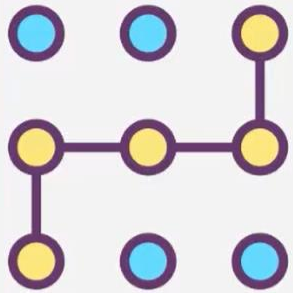
```
public class ConsumerOnReward {

    @KafkaListener(topics = "t-commodity-order")
    @SendTo("t-commodity-order-reply")
    public OrderReplyMessage listen(OrderMessage requestMessage) {
        // do some process for request
        // …

        OrderReplyMessage replyMessage = new OrderReplyMessage(...);
        return replyMessage;
    }

}
```

Data Transformation



Kafka Stream





Pattern

Spring Kafka

Kafka Stream

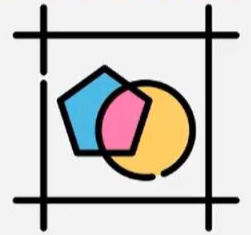
Reward

Spring Kafka

Kafka Stream

Kafka Stream

Spring Kafka



Stream

Storage

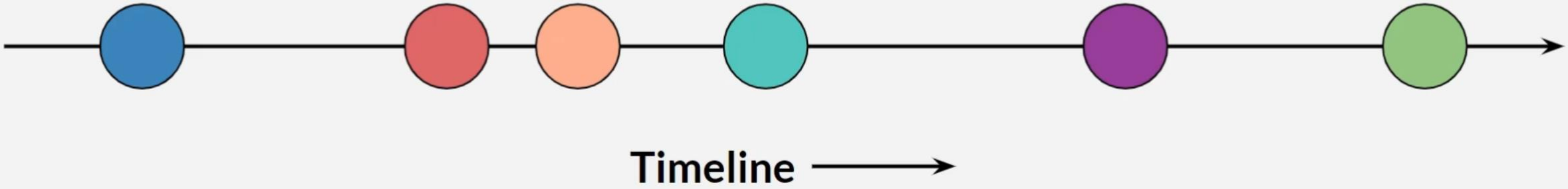
Spring Kafka

Kafka Stream

Kafka Stream

- Stream processing framework
- Released on 2017
- Alternative for Apache Spark, NIFI or Flink
- Stream & stream processing?

Data Stream



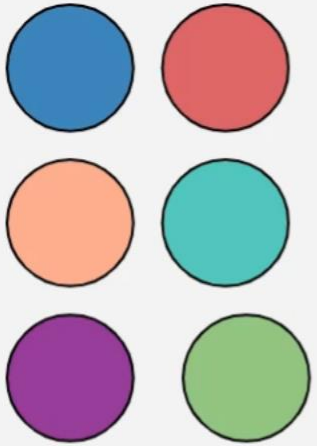
Each circle represents a data

Endless

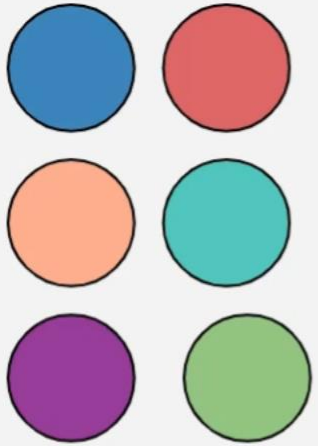
Data (event) is immutable

Can be replayed

Data Processing



Data Processing



- Transformation
- Calculation
- Aggregation
- Combination
- Filter
- etc

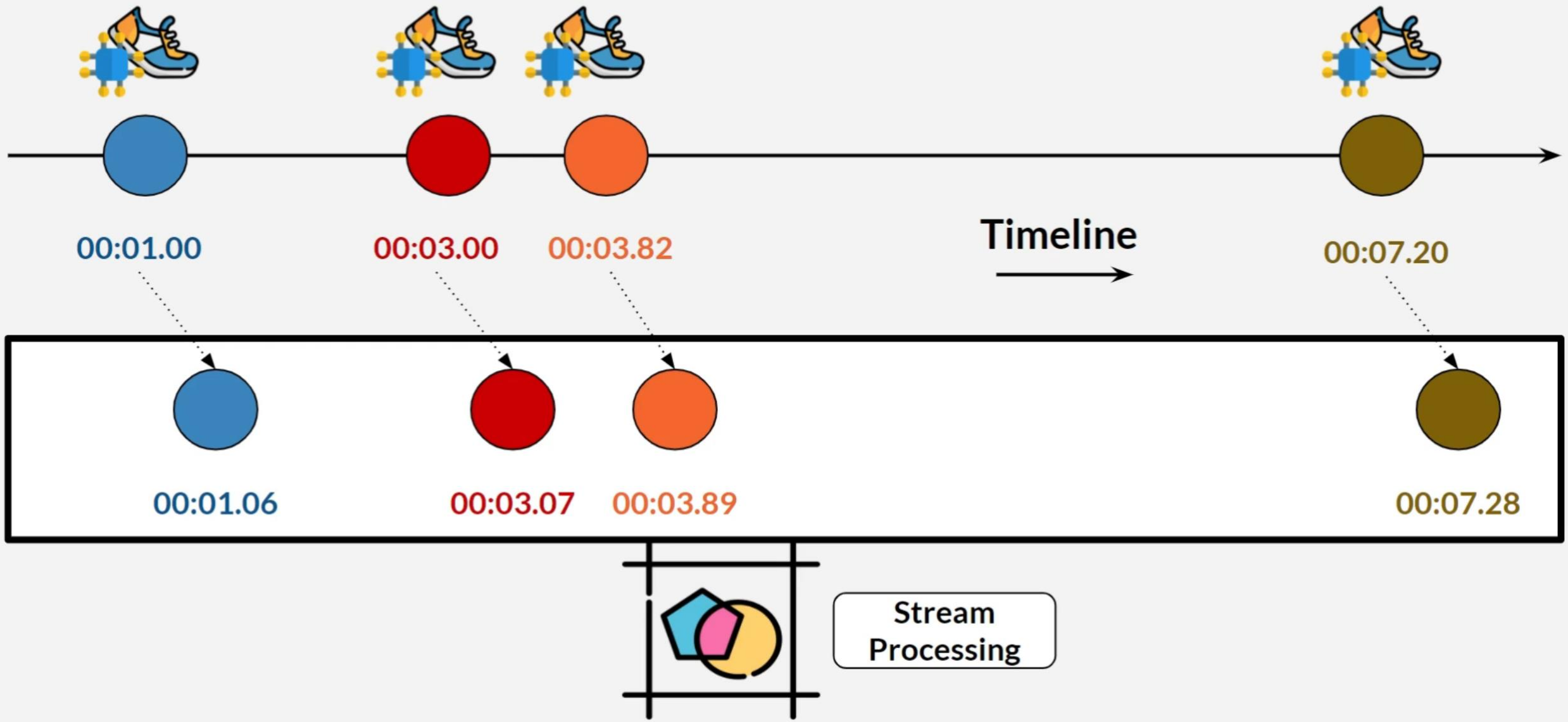
Everyday at 23:00

Every 30 min

Every 1 second

Micro Batching

Stream Processing



Yes To Stream Processing

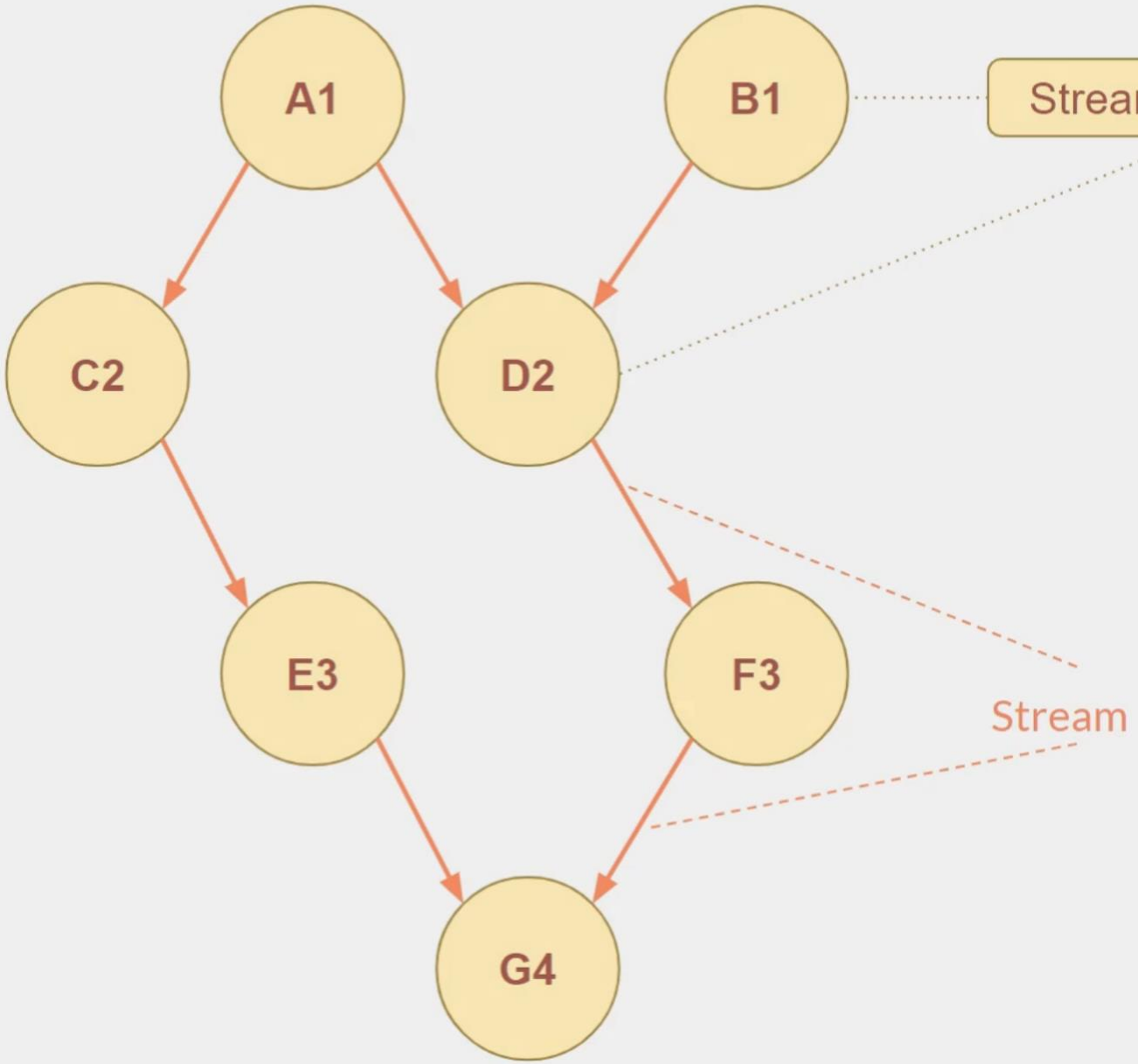
- Yes, When:
  - (relatively) fast data flow
  - Application need to response quick to most recent data

- Example
  - Marathon
  - Credit card fraud
  - Stock trading
  - Log analysis

No To Stream Processing

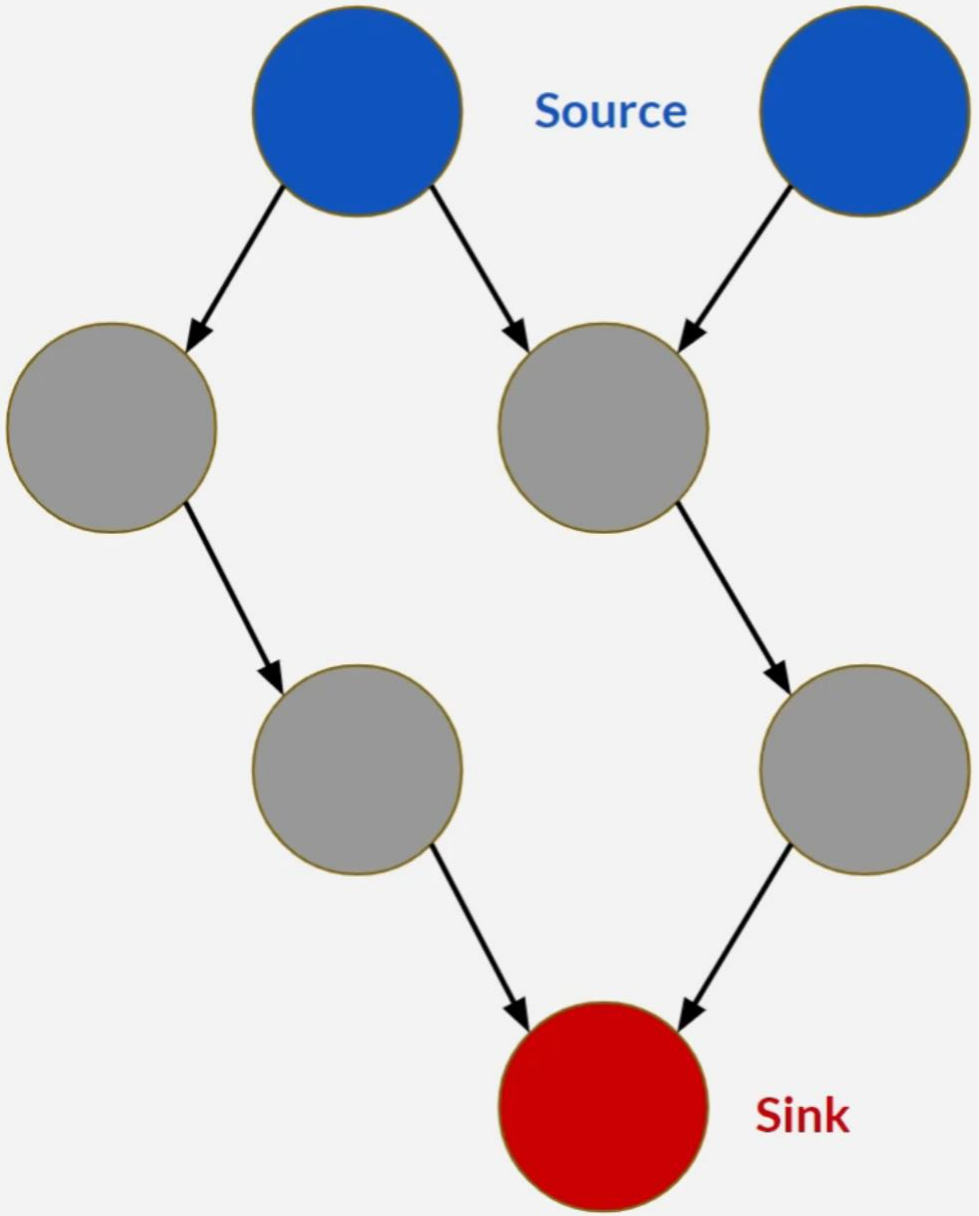
- Example
  - Daily Interest
  - Forecasting

Topology / DAG



- Stream processor process incoming data stream
- Stream processor can create new output stream
- Data flows from parent to child, not vice versa
- Parent = upstream
- Child = downstream
- Child stream processor can define another child(ren)

Kafka Stream Topology



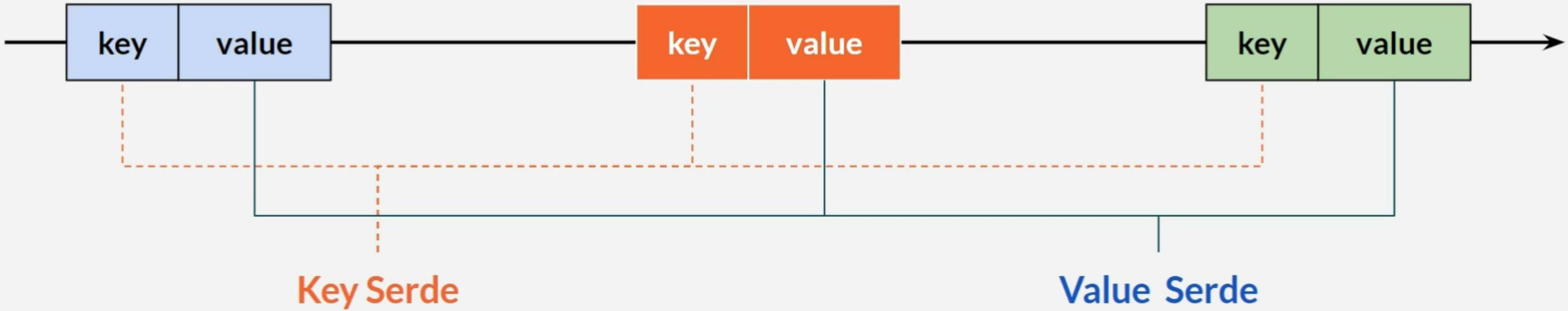
Source Processor

- Does not have upstream
- Consumes from one or more kafka topics
- Forwarding data to downstream

Sink Processor

- Does not have downstream
- Receive data from upstream
- Send data to specified kafka topic

Serde (Serializer / Deserializer)




```
Serdes.String()
Serdes.Long()
Serdes.ByteArray()
...
```

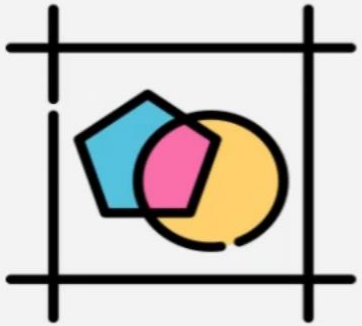
```
new JSONSerde<T>()
```

```
class CsvSerde<T> implements Serde<T>
```

What We Will Have

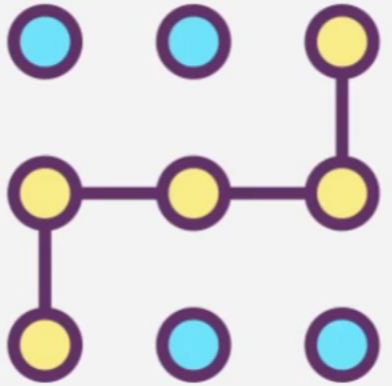
Order

Storage



Stream

Reward



Pattern

Spring Initializr

- Generate 1 java / gradle project from start.spring.io
  - Group : **`com.course.kafka`**
  - Artifact : **`kafka-stream-sample`**
  - Package name : **`com.course.kafka`** (remove any suffix)
  - Dependency : **`Spring for Apache Kafka, Spring for Apache Kafka Streams, Spring Boot Devtools`**
- Spring boot 2.x

Stream & Table

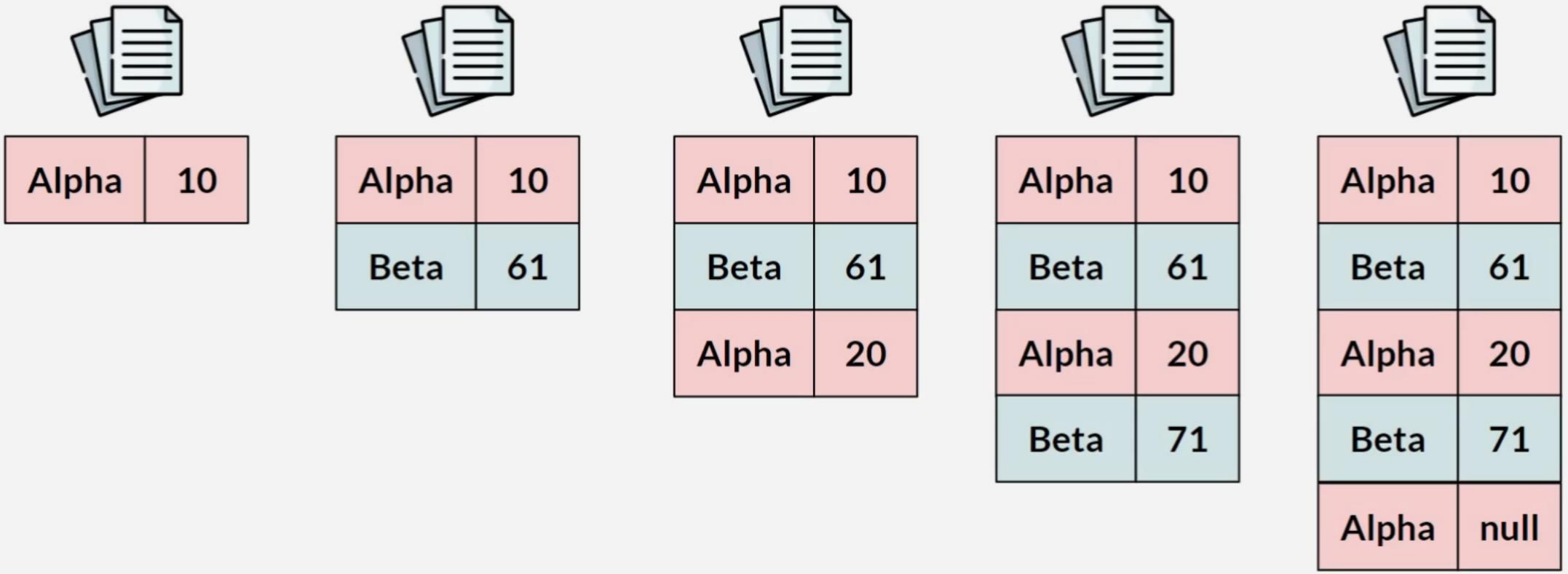
`KStream`
Ordered sequence of messages
Unbounded
Inserts data

Stream

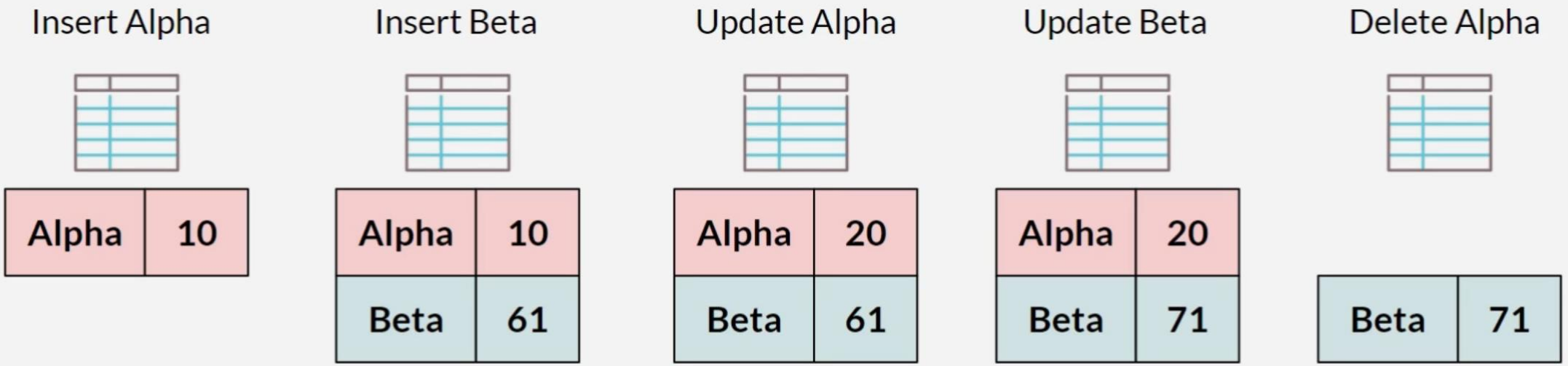
Table

`KTable`
Unbounded
Upserts data : insert or update based on key
Delete on null value
Analogy : database table

KStream : Inserts Data



KTable : Upserts / Delete Data



When to Use KStream / KTable?

- KStream
  - Topic not log-compacted
  - Data is partial information
- KTable
  - Topic is log-compacted
  - Data is self sufficient

Log Compaction

- Kafka admin process
- Keep at least latest value & delete the older
- Based on record key
- Useful if we need latest snapshot
- Configure when creating topic

Log Compaction

Topic : T, partition 0

Offset	0	1	2	3	4	5	6	7	8	9	10
Key	Alpha	Sigma	Beta	Alpha	Omega	Alpha	Delta	Delta	Beta	Omega	Omega
Value	10	180	20	11	240	12	40	40	21	241	242

Log compaction

Topic : T, partition 0

Offset	1	5	7	8	10
Key	Sigma	Alpha	Delta	Beta	Omega
Value	180	12	41	21	242

Log Compaction

Topic : T, partition 0

Offset	0	1	2	3	4	5	6	7	8	9	10
Key	Alpha	Sigma	Beta	Alpha	Omega	Alpha	Delta	Delta	Beta	Omega	Omega
Value	10	180	20	11	240	12	40	40	21	241	242

Log compaction

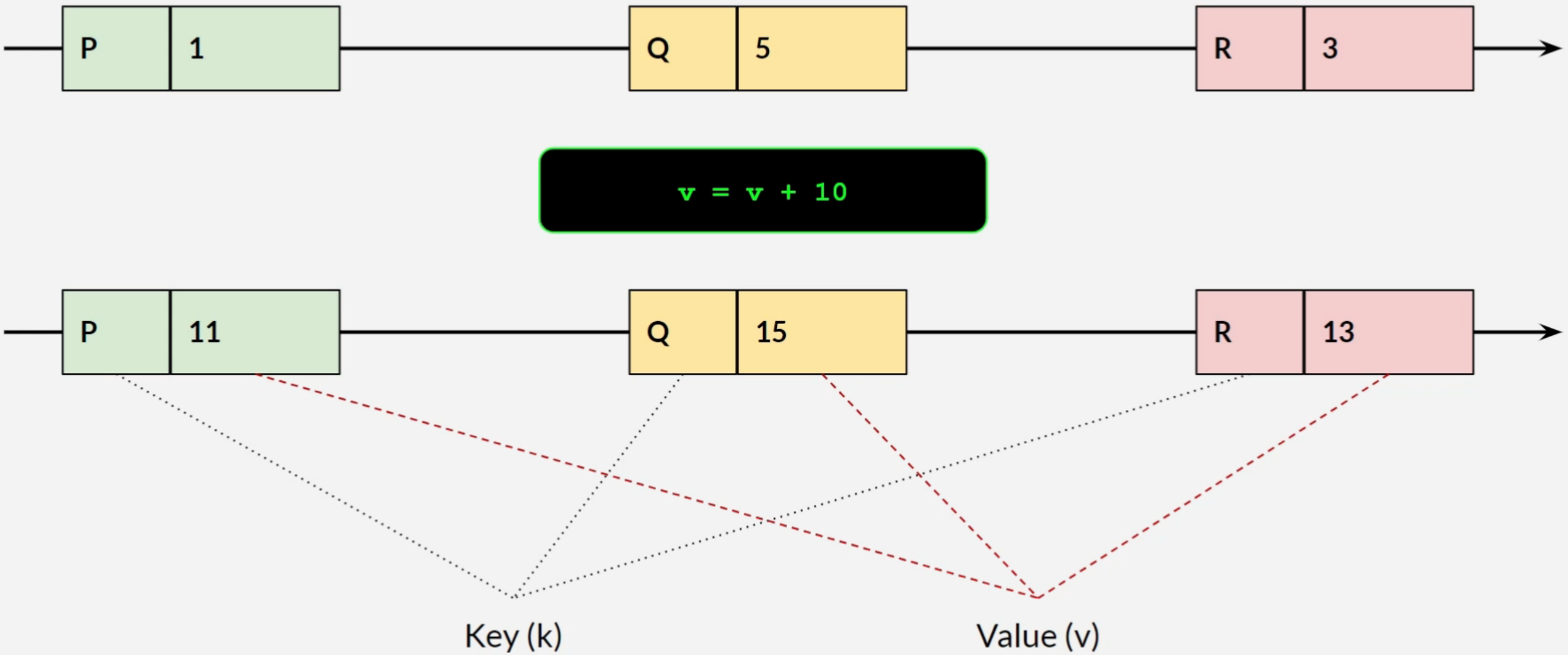
Topic : T, partition 0

Offset	1	5	7	8	9	10
Key	Sigma	Alpha	Delta	Beta	Omega	Omega
Value	180	12	40	21	241	242

Log Compaction

- Keep the order
- Not change offset
- Not duplication validator
- Can fail

Diagram



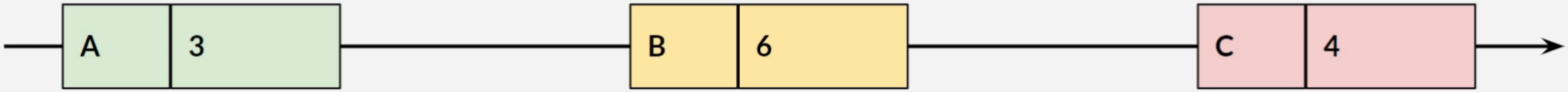
Intermediate & Terminal Operation

- Intermediate
  - KStream -> KStream
  - KTable -> Ktable
- Terminal
  - KStream -> void
  - KTable -> void
  - “Final” operation

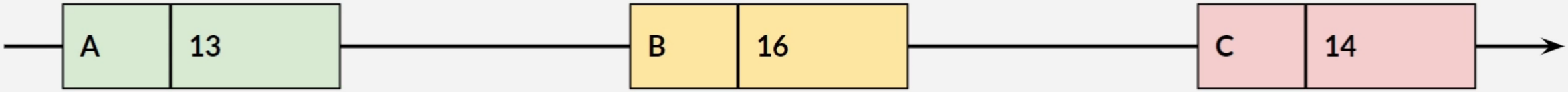
Reminder : Key & Partition

- Key & partition is related
- Partition according to key
- Repartition
  - From partition A to partition X

mapValues

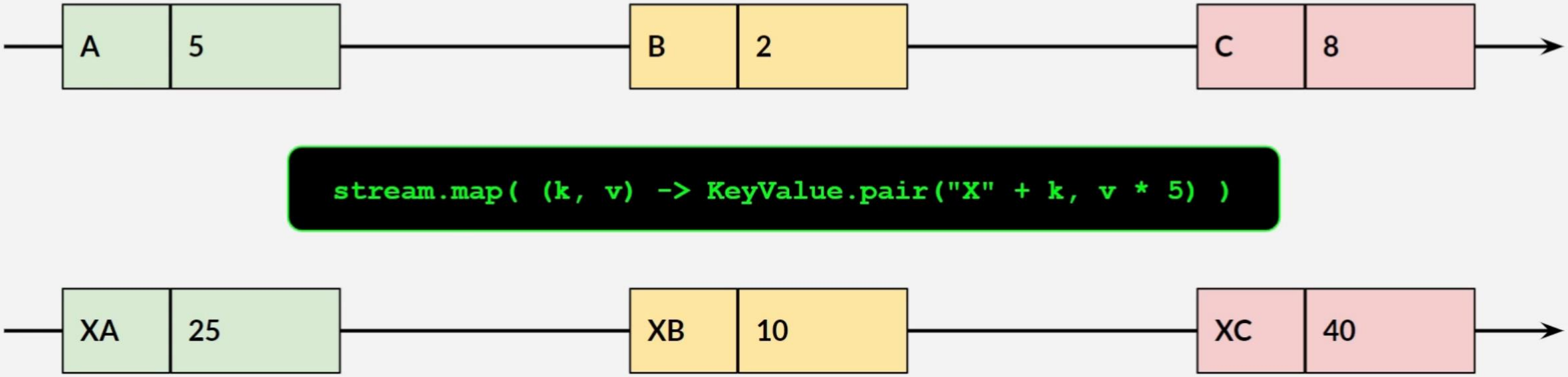



```
stream.mapValues(v -> v + 10)
```



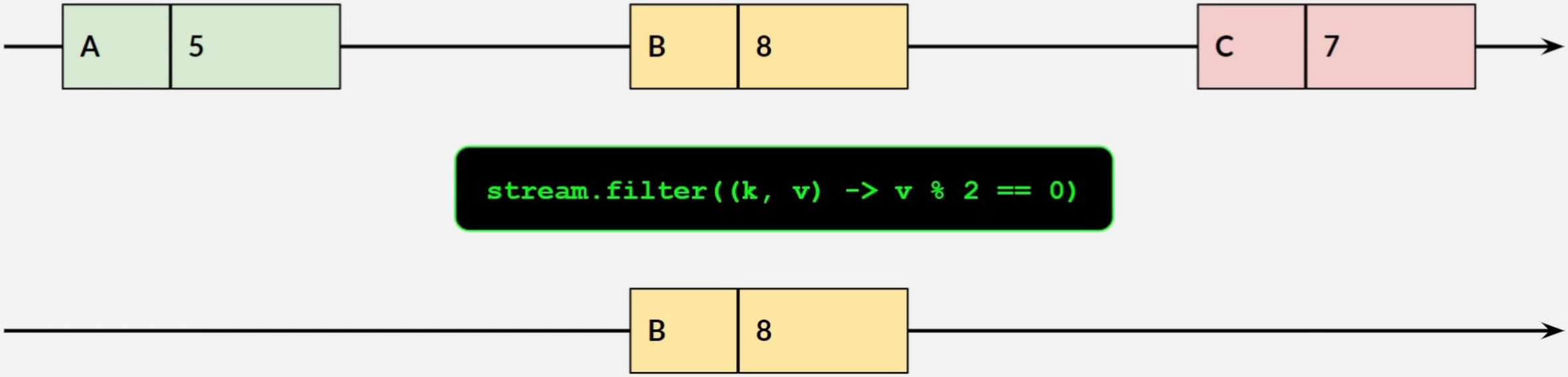
- Takes one record, produces one record
- Does not change key
- Affect only value
- Not trigger repartition
- Intermediate operation
- KStream & Ktable

map



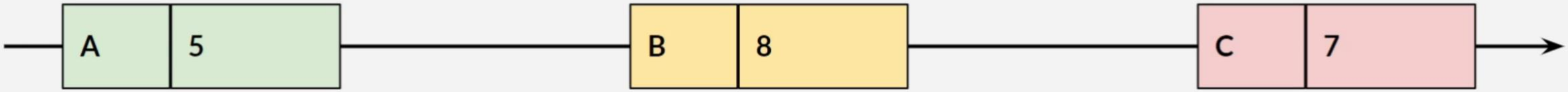
- Takes one record, produces one record
- Change key
- Change value
- Trigger repartition
- Intermediate operation
- KStream

filter

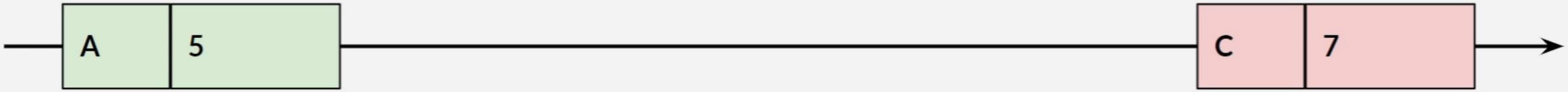


- Takes one record, produces one or zero record
- Produce record that match condition
- Does not change key or value
- Not trigger repartition
- Intermediate operation
- KStream & Ktable

filterNot

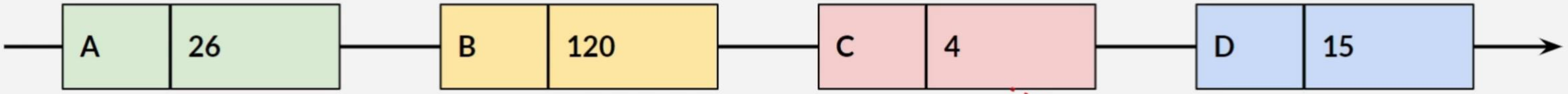



```
stream.filterNot((k, v) -> v % 2 == 0)
```



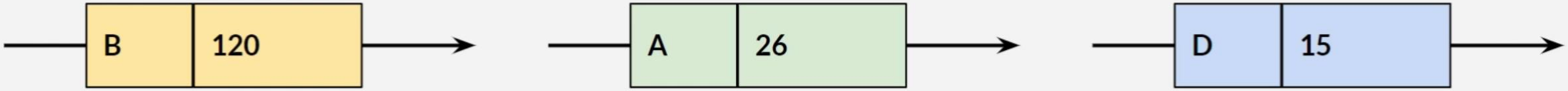
- Takes one record, produces one or zero record
- Produce record that NOT match condition
- Does not change key or value
- Not trigger repartition
- Intermediate operation
- KStream & KTable

branch




```
var arrayStream = stream.branch(
    (k, v) -> v > 100,
    (k, v) -> v > 20,
    (k, v) -> v > 10
)
```

Dropped, no match



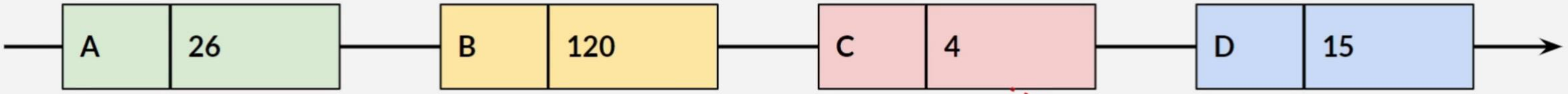
arrayStream[0] (> 100)

arrayStream[1] (> 20)

arrayStream[2] (> 10)

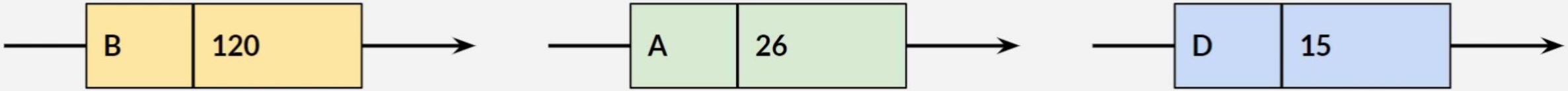
- Split stream based on predicates
- Evaluate predicate in order
- Record only placed once on first match, drop unmatched record
- Returns array of stream
- Intermediate operation
- KStream

split & branch




```
stream.split().
    branch((k, v) -> v > 100, Branched.withConsumer(ks -> ks.to("t-x"))
    .branch((k, v) -> v > 20, Branched.withConsumer(ks -> ks.to("t-y"))
    .branch((k, v) -> v > 10, Branched.withConsumer(ks -> ks.to("t-z"))
)
```

Dropped, no match



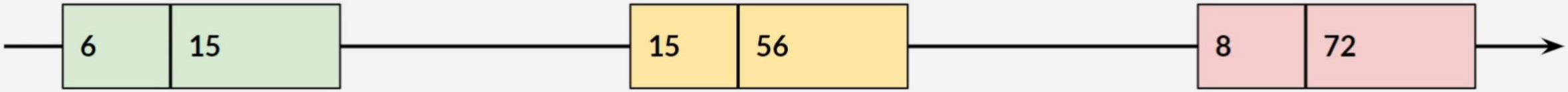
KStream 1 (> 100)

KStream 2 (> 20)

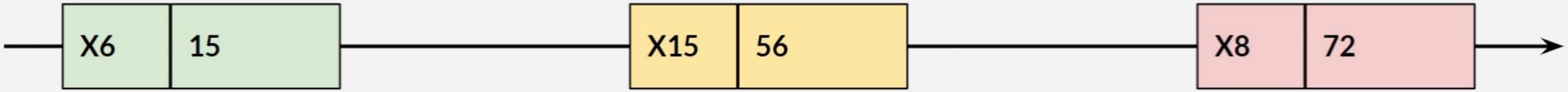
KStream 3 (> 10)

- Split stream based on predicates
- Evaluate predicate in order
- Record only placed once on first match, drop unmatched record
- Get KStream for each branch
- split() returns final BranchedKStream
- Each branch returns KStream to be processed further

selectKey

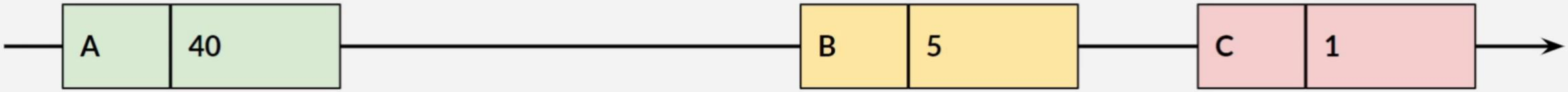



```
stream.selectKey((k, v) -> "X" + k)
```

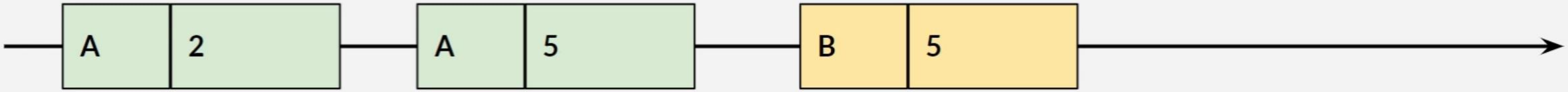


- Takes one record, produces one record
- Set / replace record key
- Possible to change key data type
- Trigger repartitioning
- Value not change
- Intermediate operation
- KStream

flatMapValues

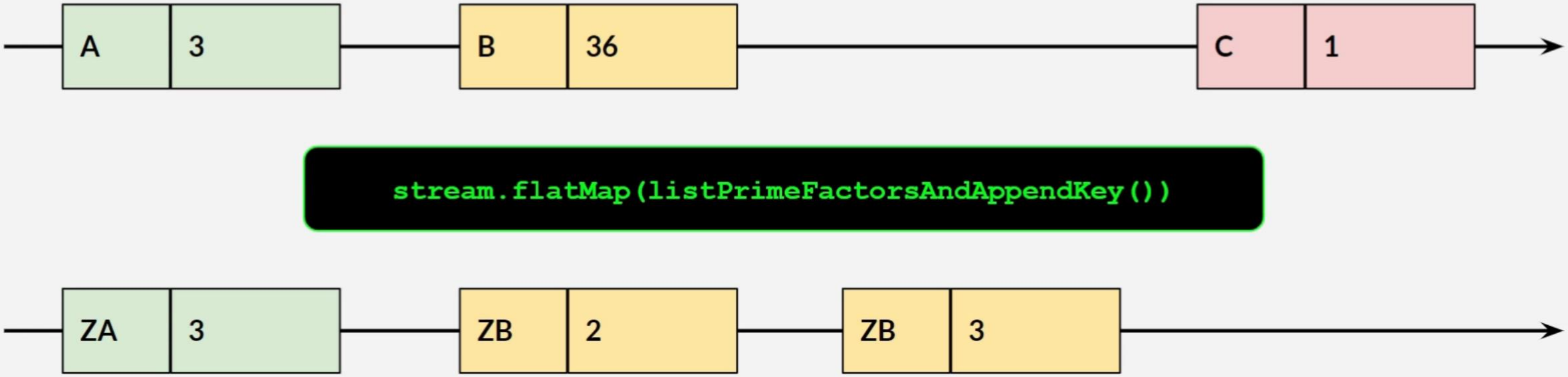



```
stream.flatMapValues(listPrimeFactors())
```



- Takes one record, produces zero or more record
- Does not change key
- Affect only value
- Not trigger repartition
- Intermediate operation
- KStream

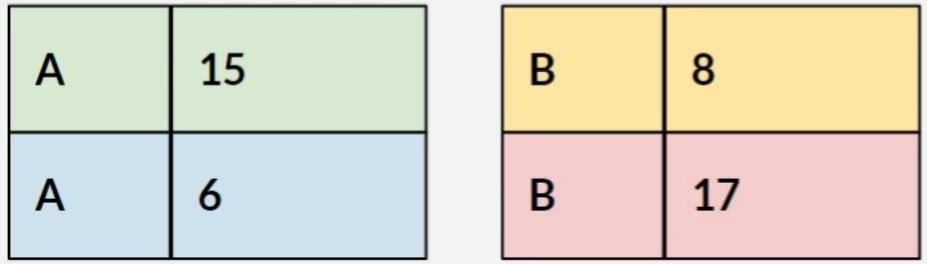
flatMap



- Takes one record, produces zero or more record
- Change key
- Change value
- Trigger repartition
- Intermediate operation
- KStream

groupByKey

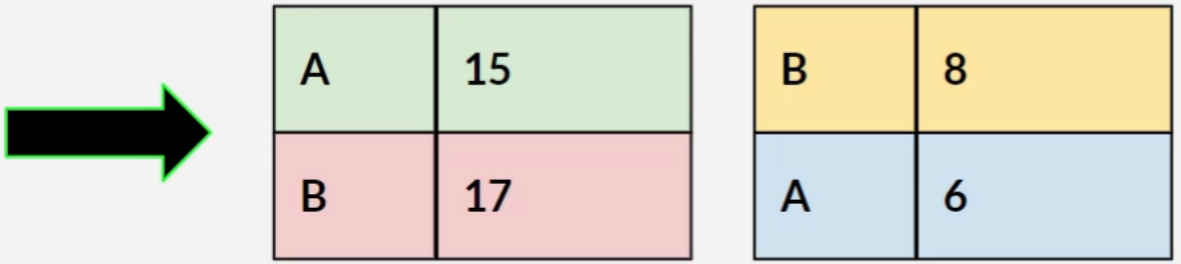

```
stream.groupByKey()
```



- Intermediate operation
- Group records by existing key
- KStream

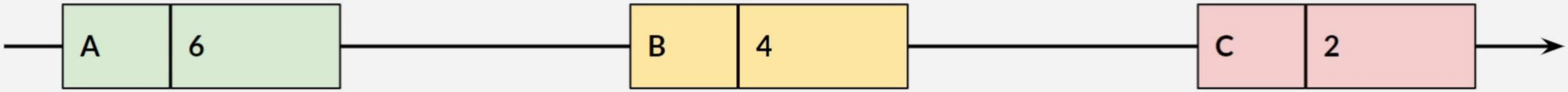
groupBy


```
stream.groupBy((k, v) -> v % 2)
```



- Intermediate operation
- Group records by new key
- KStream & KTable

forEach




```
stream.forEach((k,v) -> insertToDatabase(v))
```

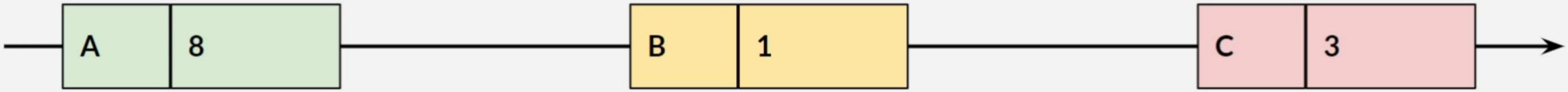

6


6
4


6
4
2

- Terminal operation
- Takes one record, produces none
- Produces side effect
- Side effect not tracked by kafka
- KStream & KTable

peek




```
stream.peek((k,v) -> insertToDatabase(v)).[nextProcessor]
```

Next processor

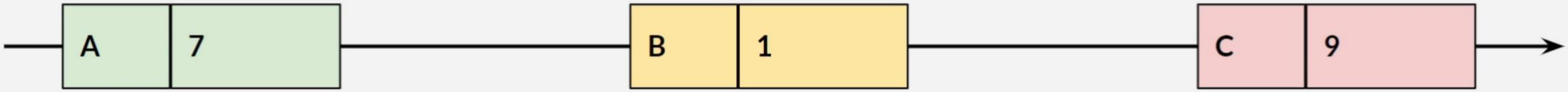
8

8
1

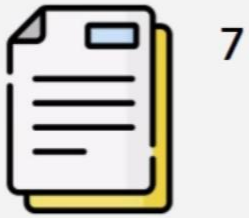
8
1
3

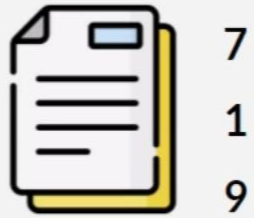
- Produces unchanged stream
- Produces side effect
- Side effect not tracked by kafka
- Result stream can be processed further
- Intermediate operation
- KStream

print



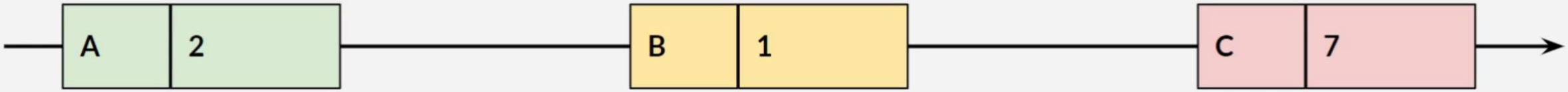

```
stream.print(Printed.toSysout())
```





- Terminal operation
- Print each record
- Something like kafka console consumer
- Print to file or console
- KStream

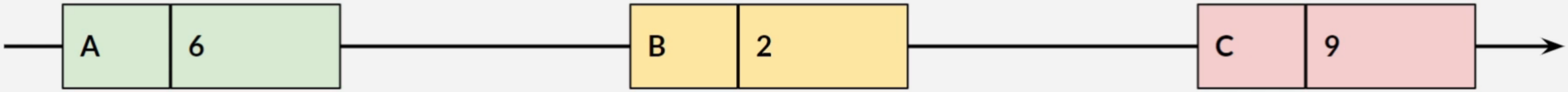
to




```
stream.to("output-topic")
```

- Terminal operation
- Write stream to destination topic
- KStream

through

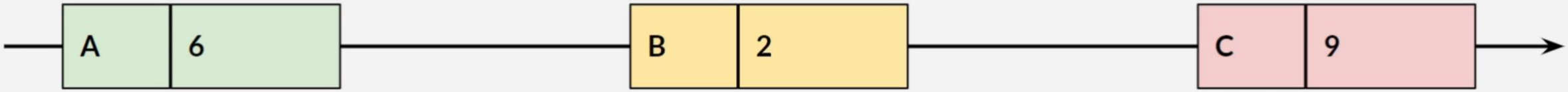



```
stream.through("output-topic").[nextProcessor]
```

Next processor

- Intermediate operation
- Write stream to destination topic
- Continue record processing
- KStream

repartition



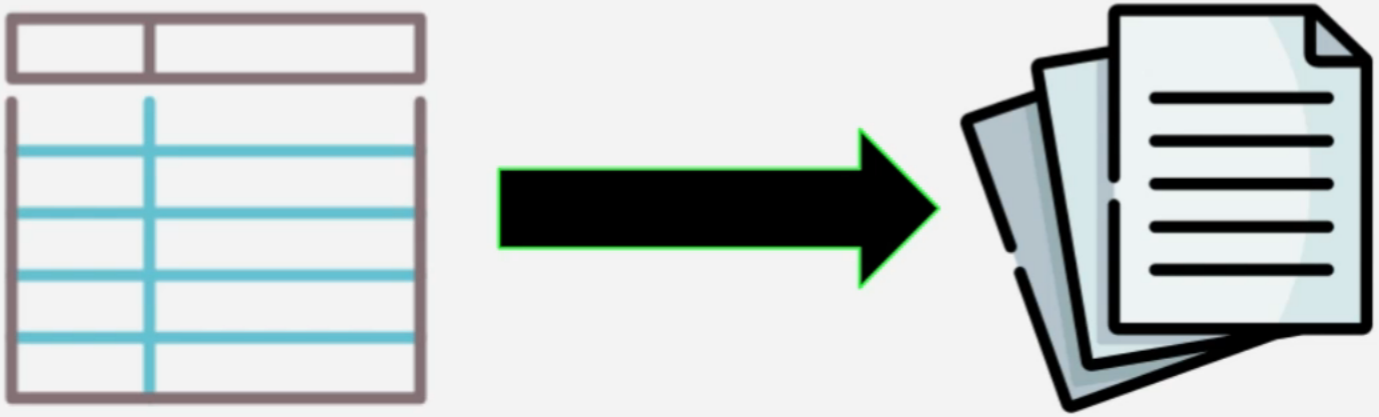

```
stream.repartition().[nextProcessor]
stream.repartition(Repartitioned.as("output-topic")).[nextProcessor]
```

Next processor

- Intermediate operation
- Write stream to destination topic
- Continue record processing
- repartition() output-topic name is fixed
- Topic only for kafka internal use

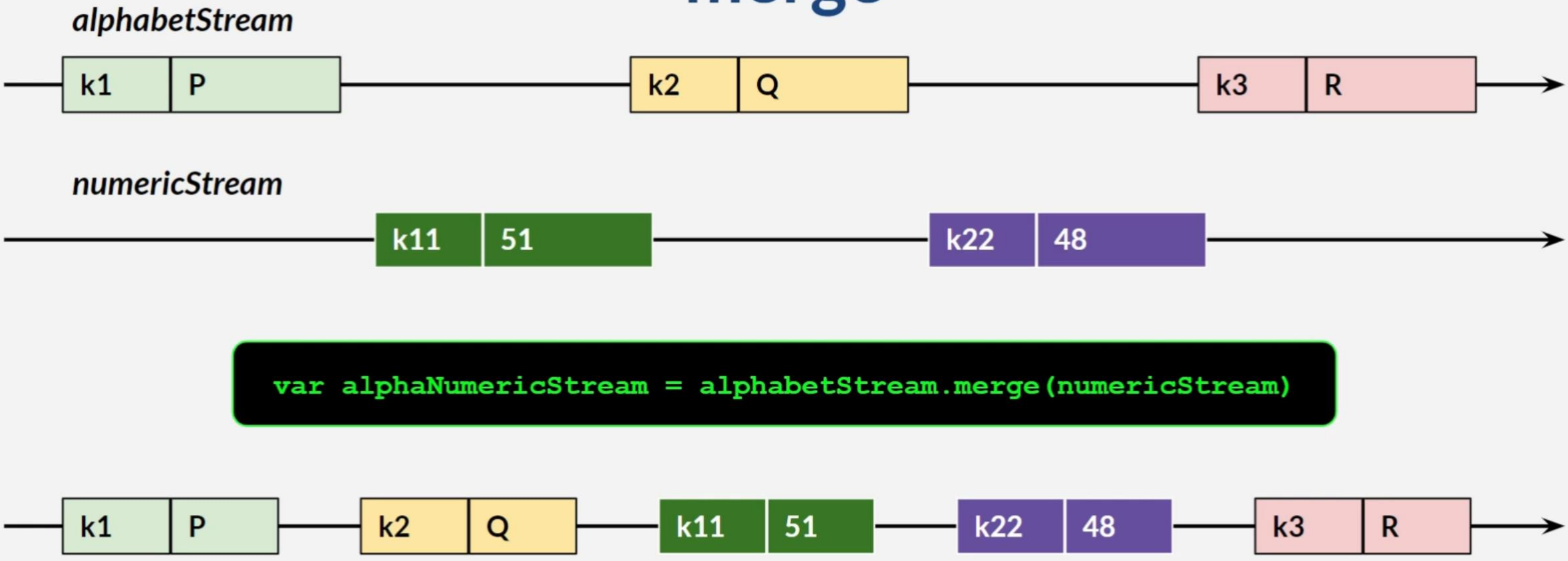
toStream

```
table.toStream()
```



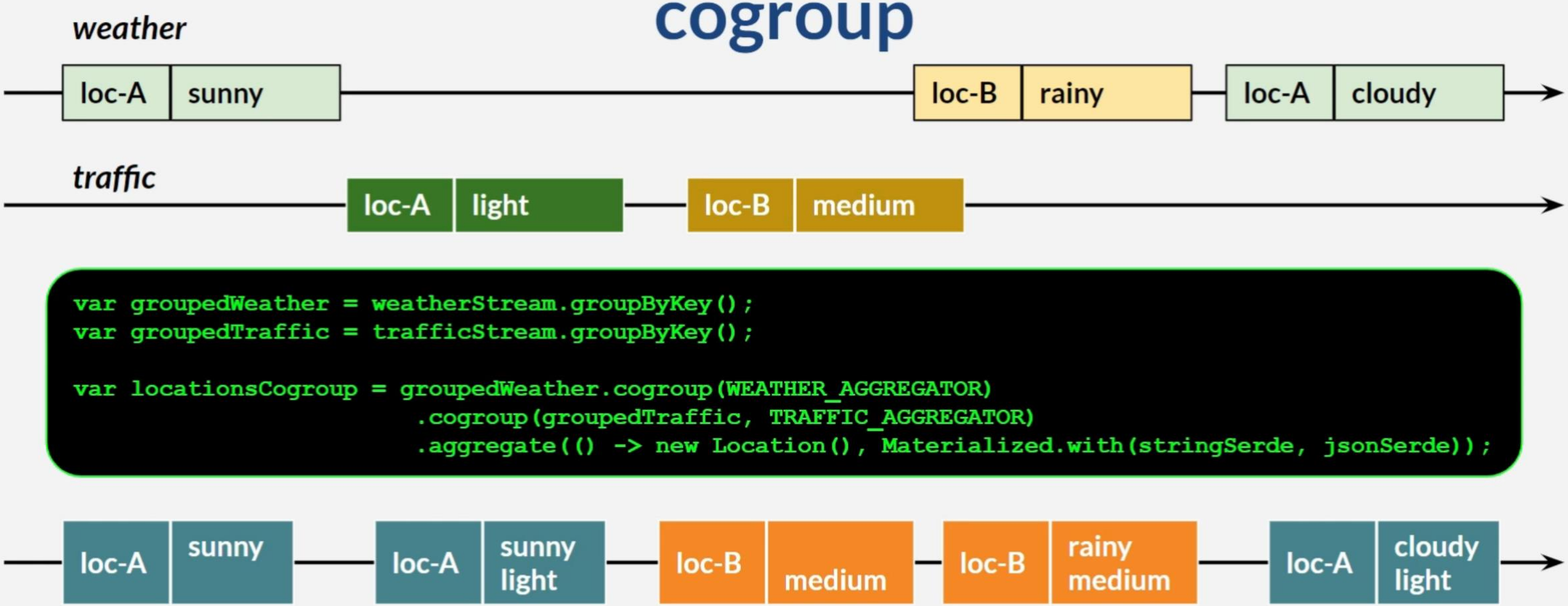
- KTable
- Intermediate operation
- Convert KTable to KStream

merge



- Merge two streams into one new stream
- No ordering guarantee on resulting stream
- Intermediate operation
- KStream

cogroup




```
var groupedWeather = weatherStream.groupByKey();
var groupedTraffic = trafficStream.groupByKey();

var locationsCogroup = groupedWeather.cogroup(WEATHER_AGGREGATOR)
                      .cogroup(groupedTraffic, TRAFFIC_AGGREGATOR)
                      .aggregate(() -> new Location(), Materialized.with(stringSerde, jsonSerde));
```


- Intermediate operation
- KStream
- Need aggregator for each cogroup
- This sample : String key, JSON value
- Difference with merge()

cogroup - Aggregator

```
Aggregator<String, String, Location> WEATHER_AGGREGATOR = new Aggregator<String, String, Location>() {

        @Override
        public Location apply(String key, String value, Location aggregate) {
            aggregate.setWeather(value);
            return aggregate;
        }
    };
```

```
Aggregator<String, String, Location> TRAFFIC_AGGREGATOR = new Aggregator<String, String, Location>() {

        @Override
        public Location apply(String key, String value, Location aggregate) {
            aggregate.setTraffic(value);
            return aggregate;
        }
    };
```

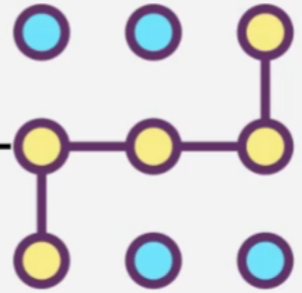
Order - Kafka Message

Kafka Message

Each message has

- order number
- location
- order date & time
- credit card number
- name
- price
- quantity

Aggregate by item name

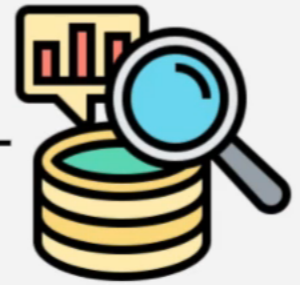


Pattern

Aggregate by location

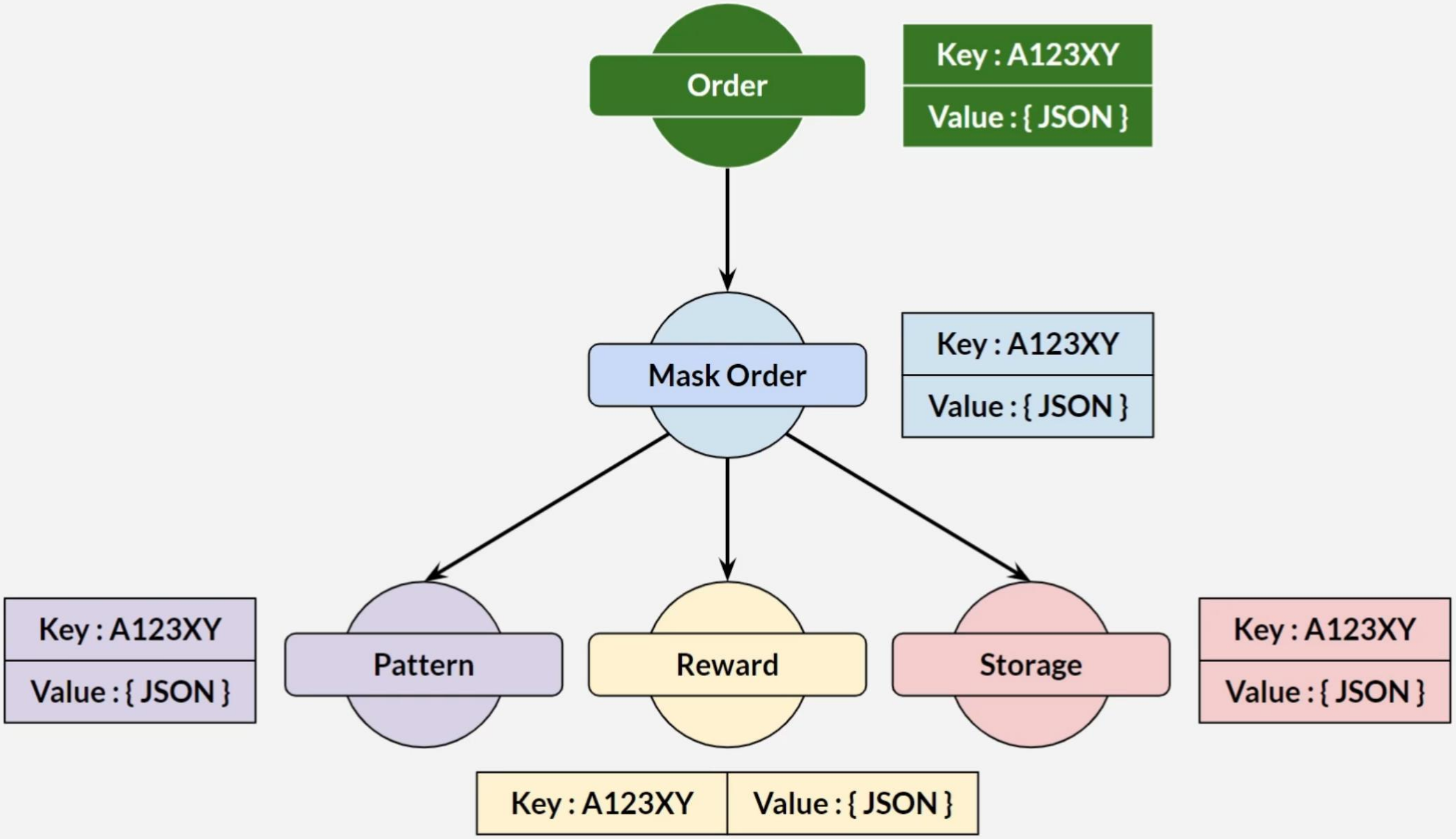
Reward

No aggregation

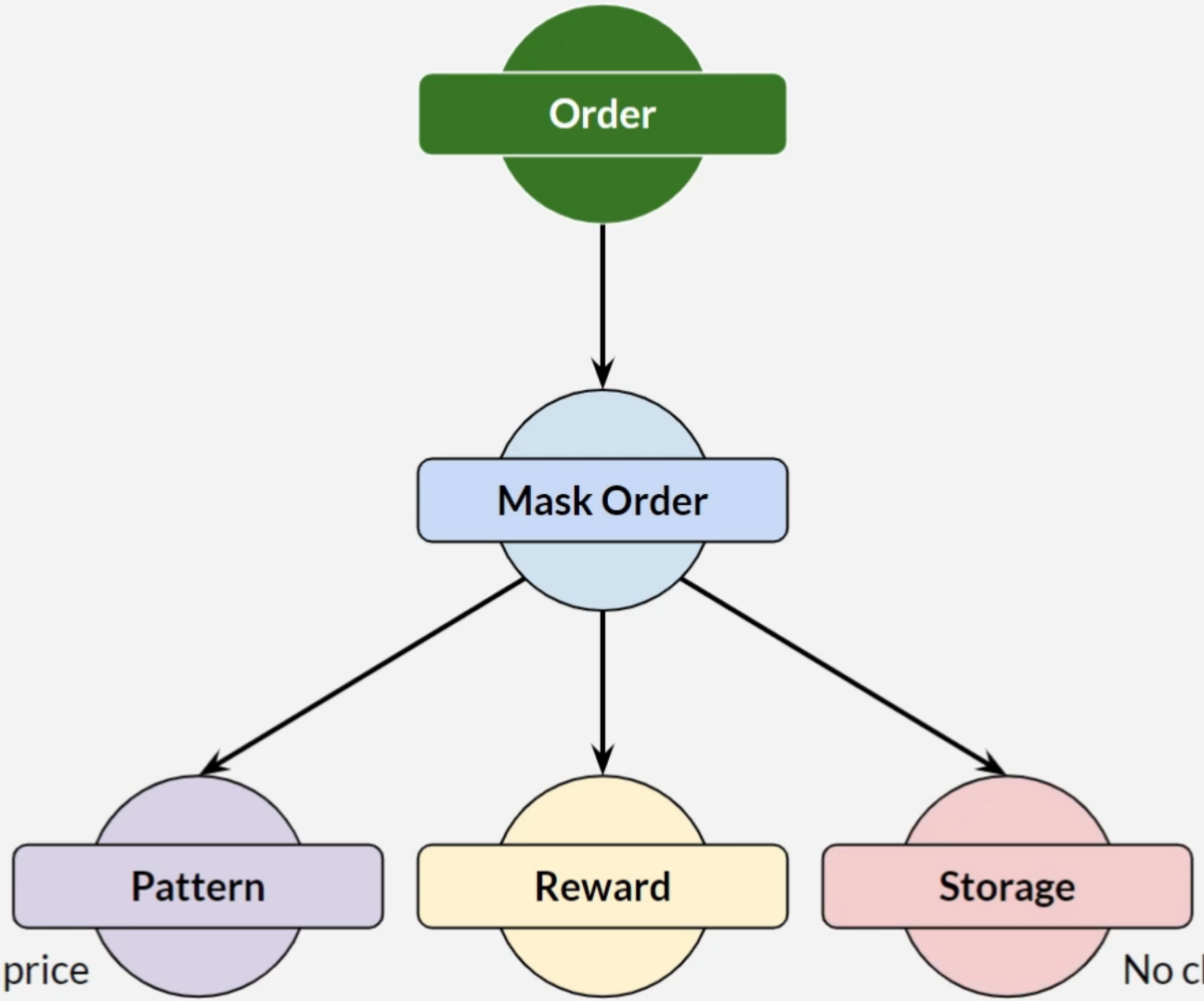


Storage

Topology


Order
Key : A123XY
Value : { JSON }
Mask Order
Key : A123XY
Value : { JSON }
Pattern
Key : A123XY
Value : { JSON }
Reward
Key : A123XY
Value : { JSON }
Storage
Key : A123XY
Value : { JSON }
Key : A123XY
Value : { JSON }

High Level Topology

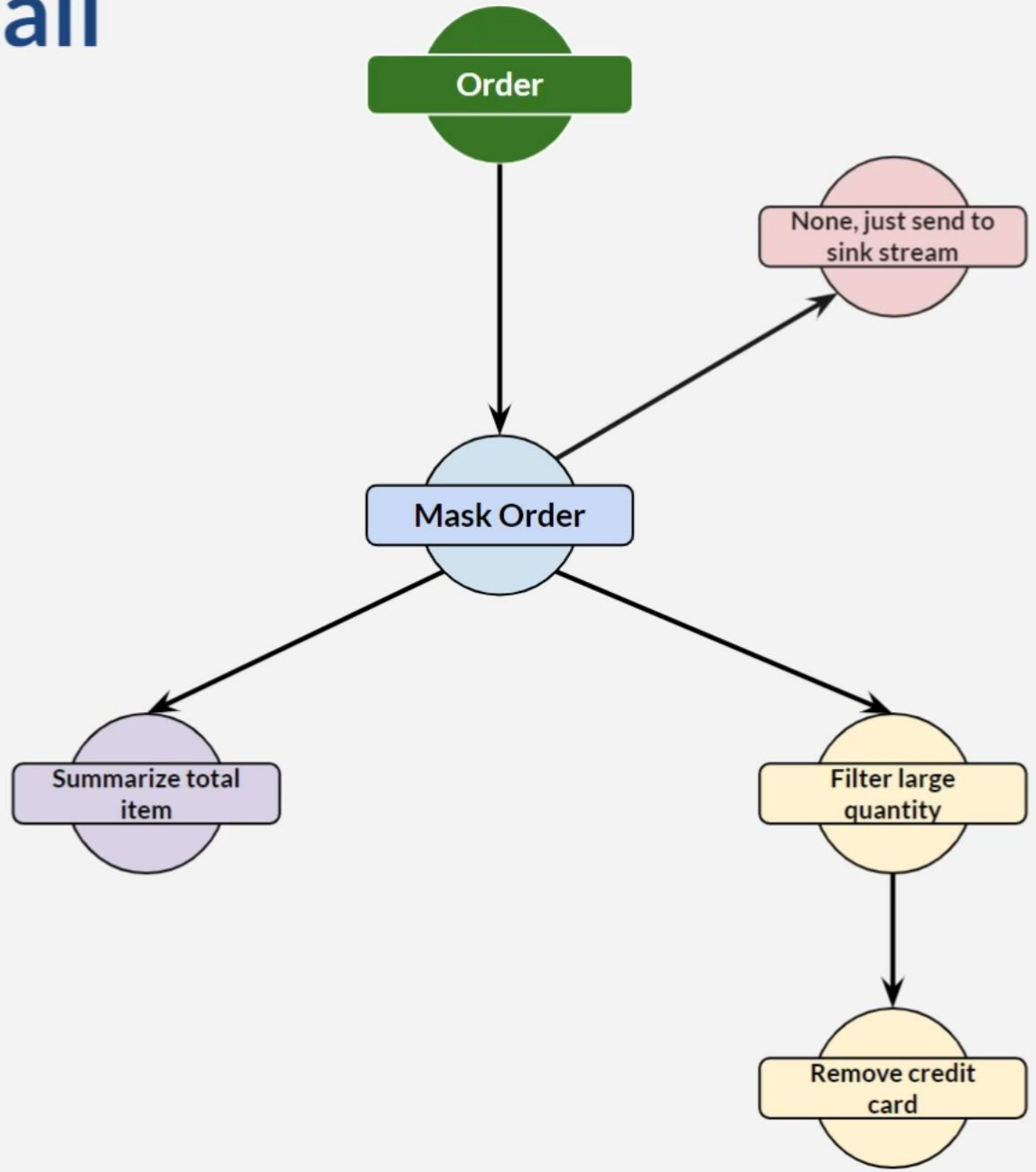


total item amount = quantity * price

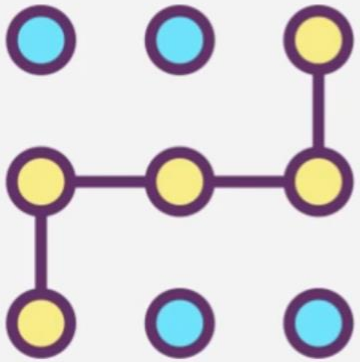
Only large quantity
Remove credit card number

No change

Topology Detail



Additional Requirement



Pattern

Current : summarize item price * quantity

+ ADDITIONAL : split plastic & non plastic items

Reward

Current : give reward only for item with quantity > xxx

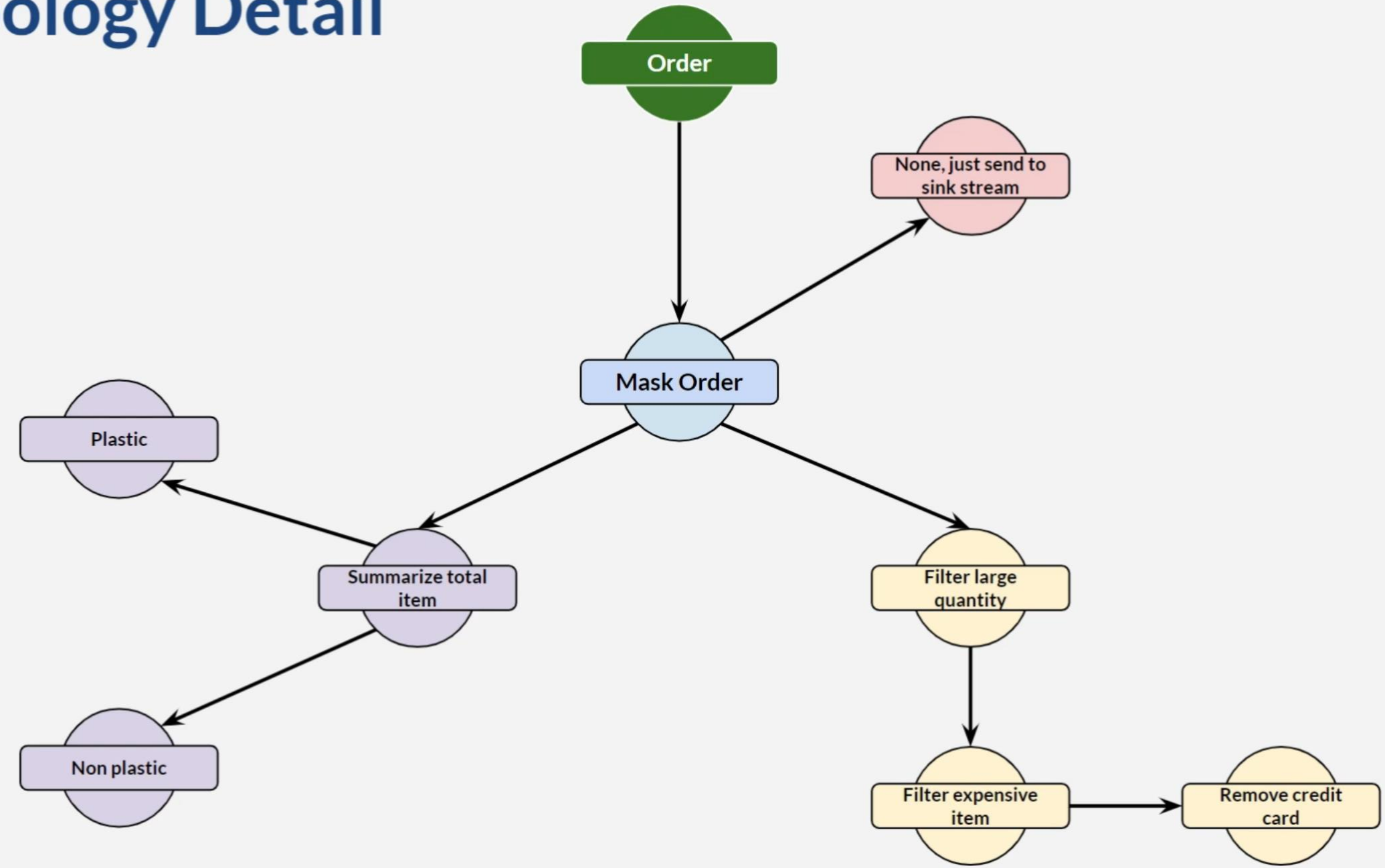
+ ADDITIONAL : give reward only for item that is not cheap

Storage

Current : -

+ ADDITIONAL : key is base64(order number)

Topology Detail



Sample Data

Cotton Dog

Price : 80
Qty : 250

Plastic Cat

Price : 400
Qty : 500

Wooden Horse

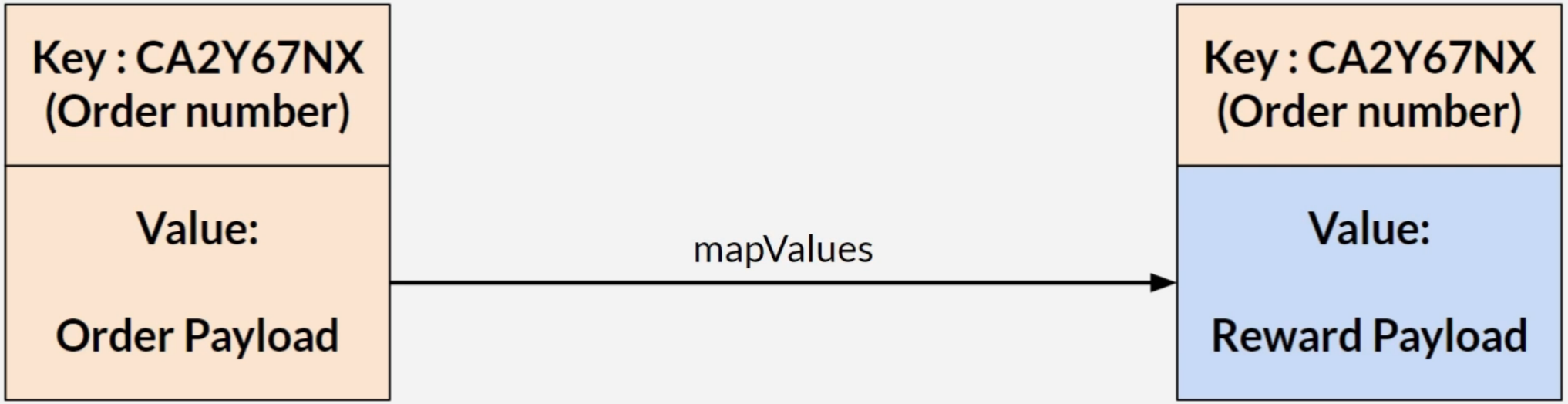
Price : 700
Qty : 90

Steel Pig

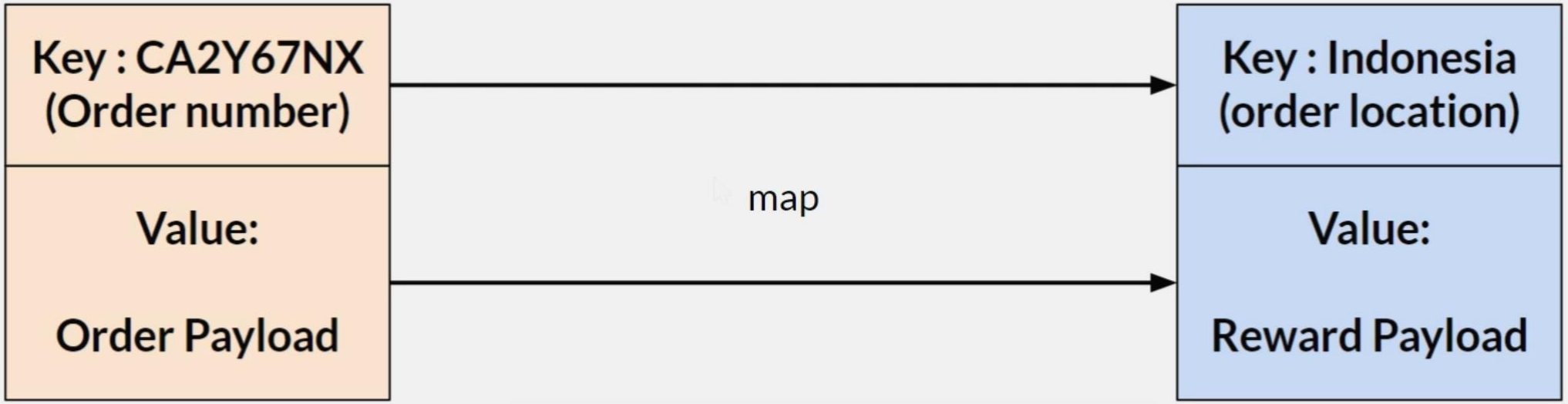
Price : 350
Qty : 270

Stream (Kafka Sink Topic)	Data
Pattern - plastic	Plastic Cat
Pattern - not plastic	Cotton Dog, Wooden Horse, Steel Pig
Reward	Plastic Cat, Steel Pig
Storage	Plastic Cat, Cotton Dog, Wooden Horse, Steel Pig

Reward Message

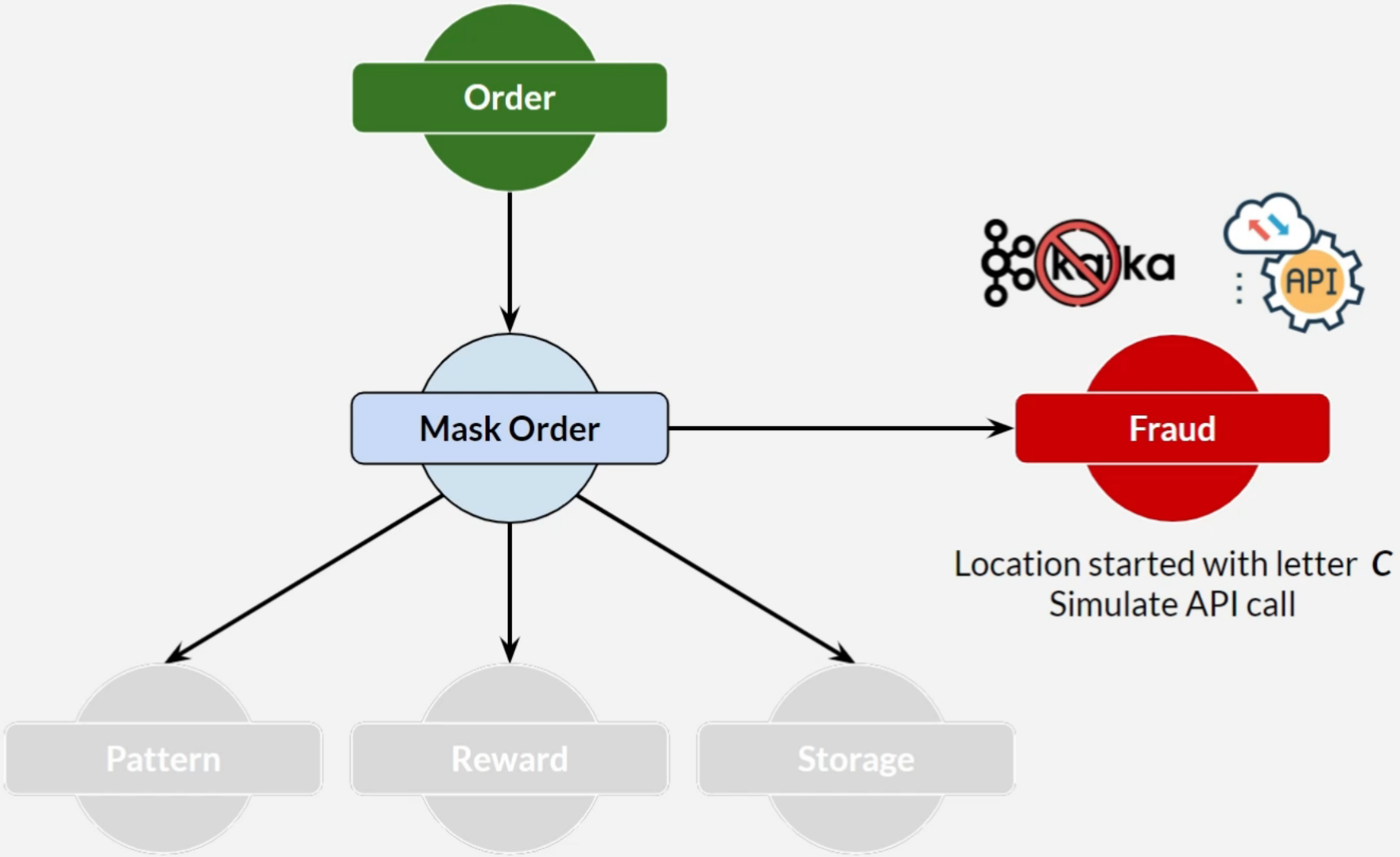


Reward Message



Something Suspicious

High Level Topology



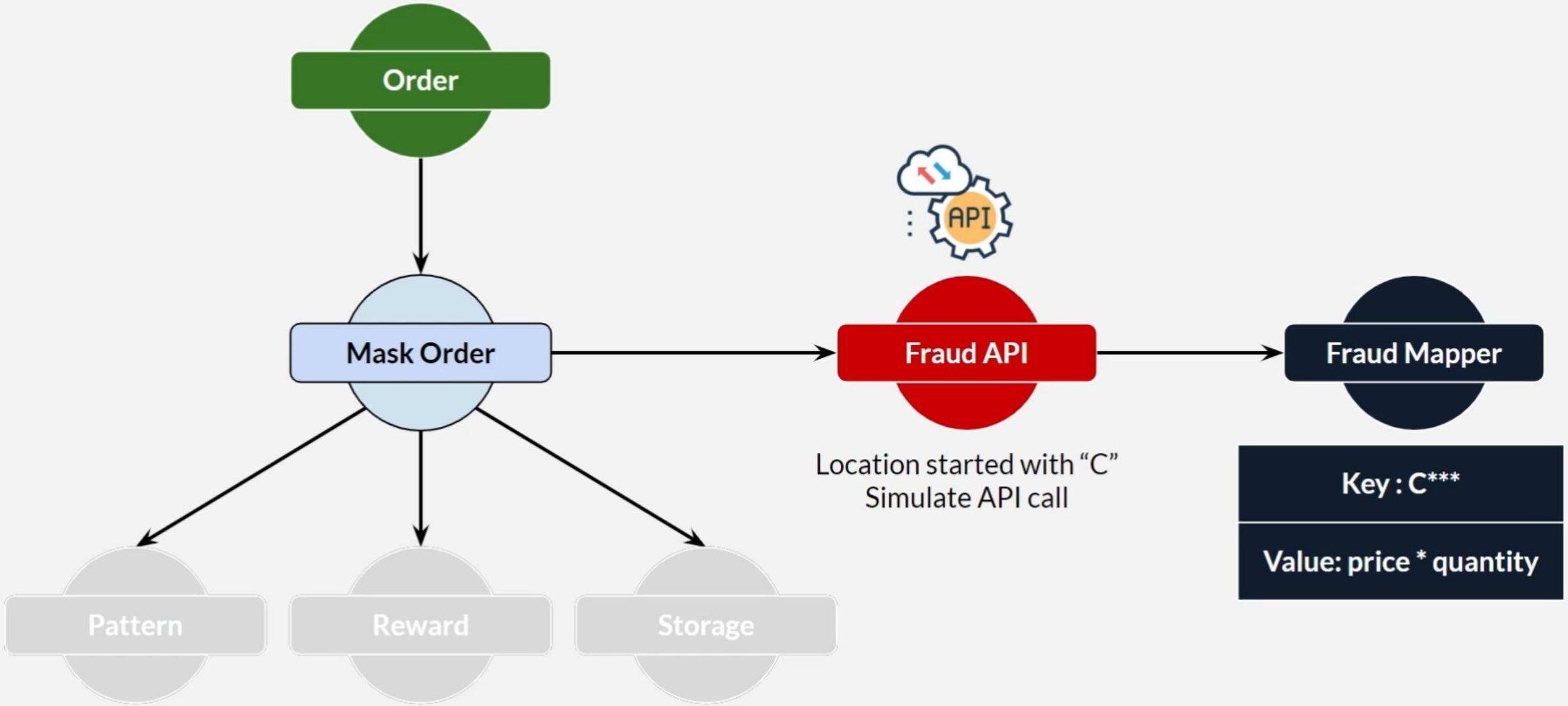
Something Suspicious

kafka

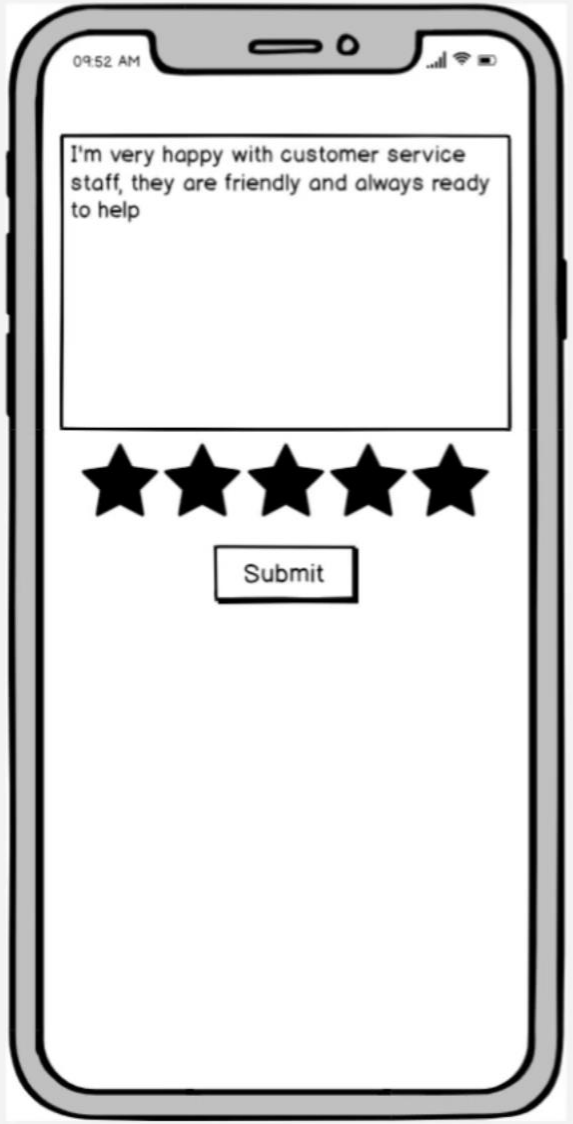
Key : C***

Value: price * quantity

High Level Topology

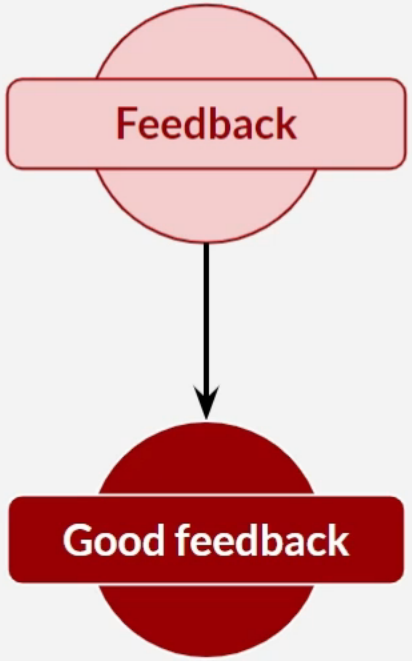


Customer Feedback



happy, good, helpful, etc

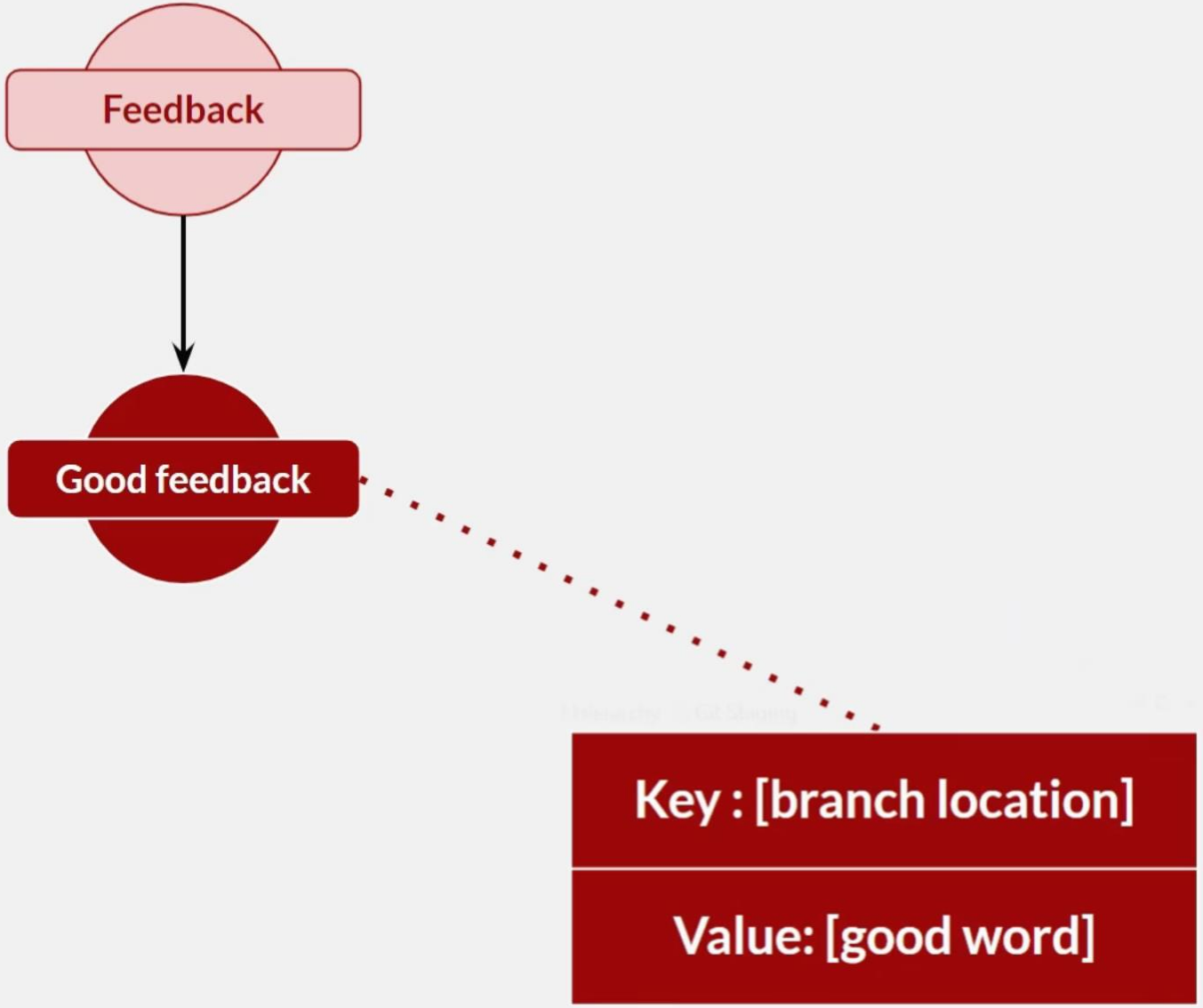
High Level Topology



Create project for feedback

- Feedback*.java
- Package: **com.virtusa.kafka**
  - api.request
  - api.server
  - broker.message
  - broker.producer
  - command.action
  - command.service

High Level Topology



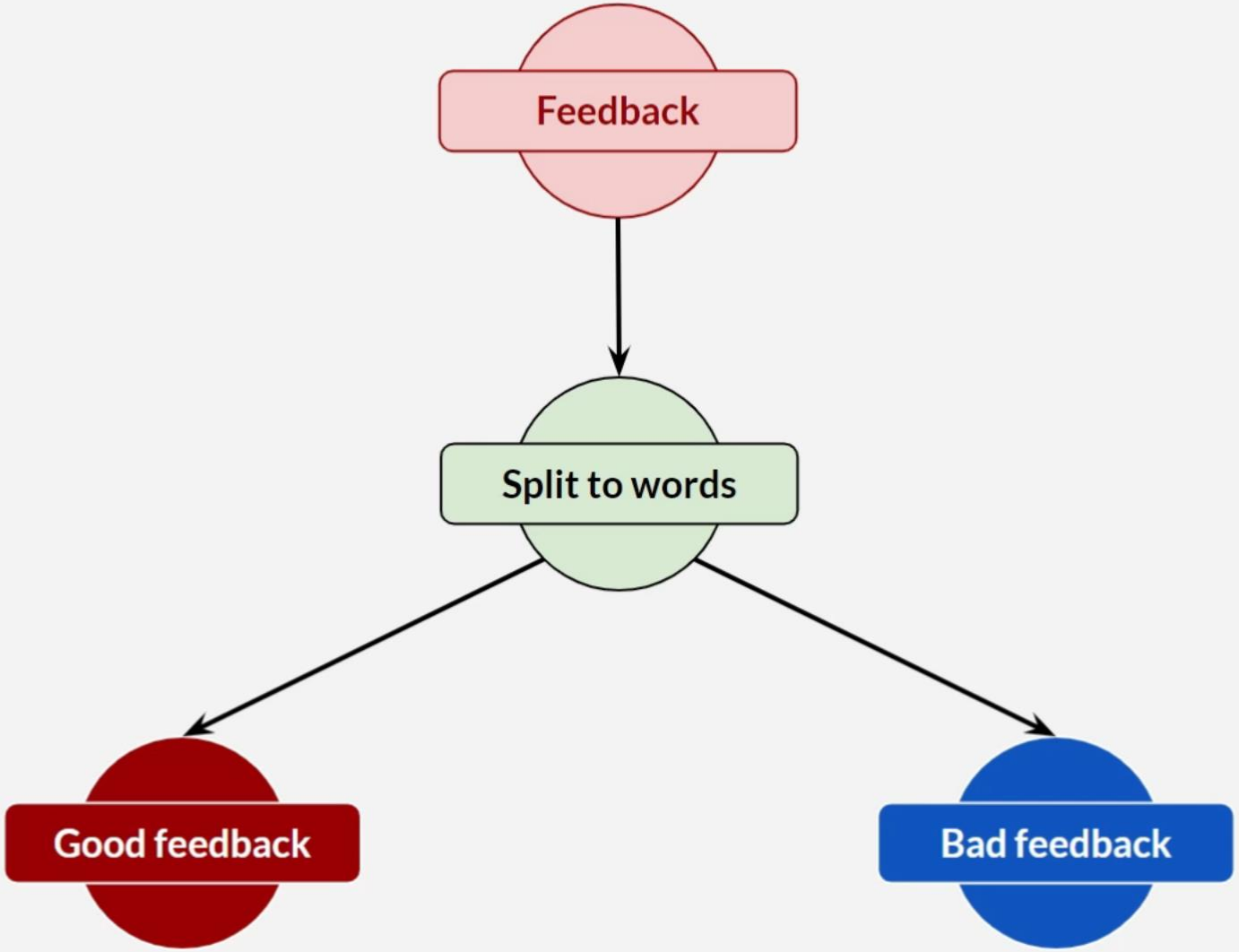
Java Stream API

- Since Java 8
- Not kafka stream
- Some method names:
  - filter
  - filterMap
  - filterEach
  - map
  - peek

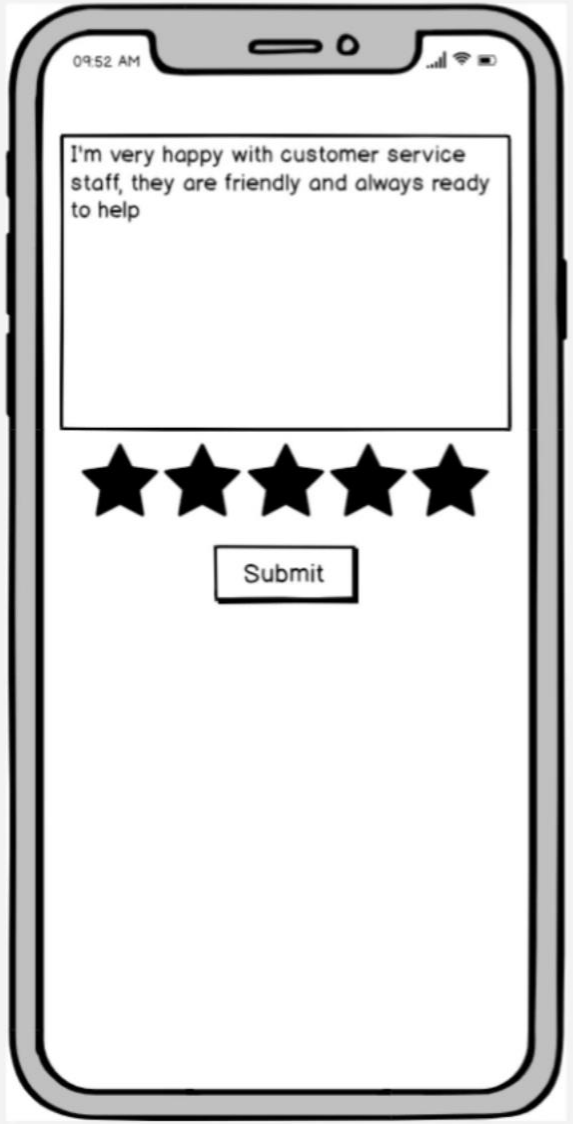
Bad Feedback?

- Analyze bad feedback
- Stream to analyze feedback : good or bad
- Topology

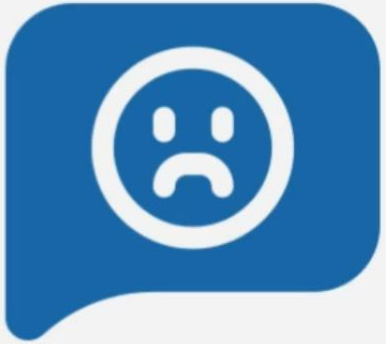
High Level Topology



Customer Feedback

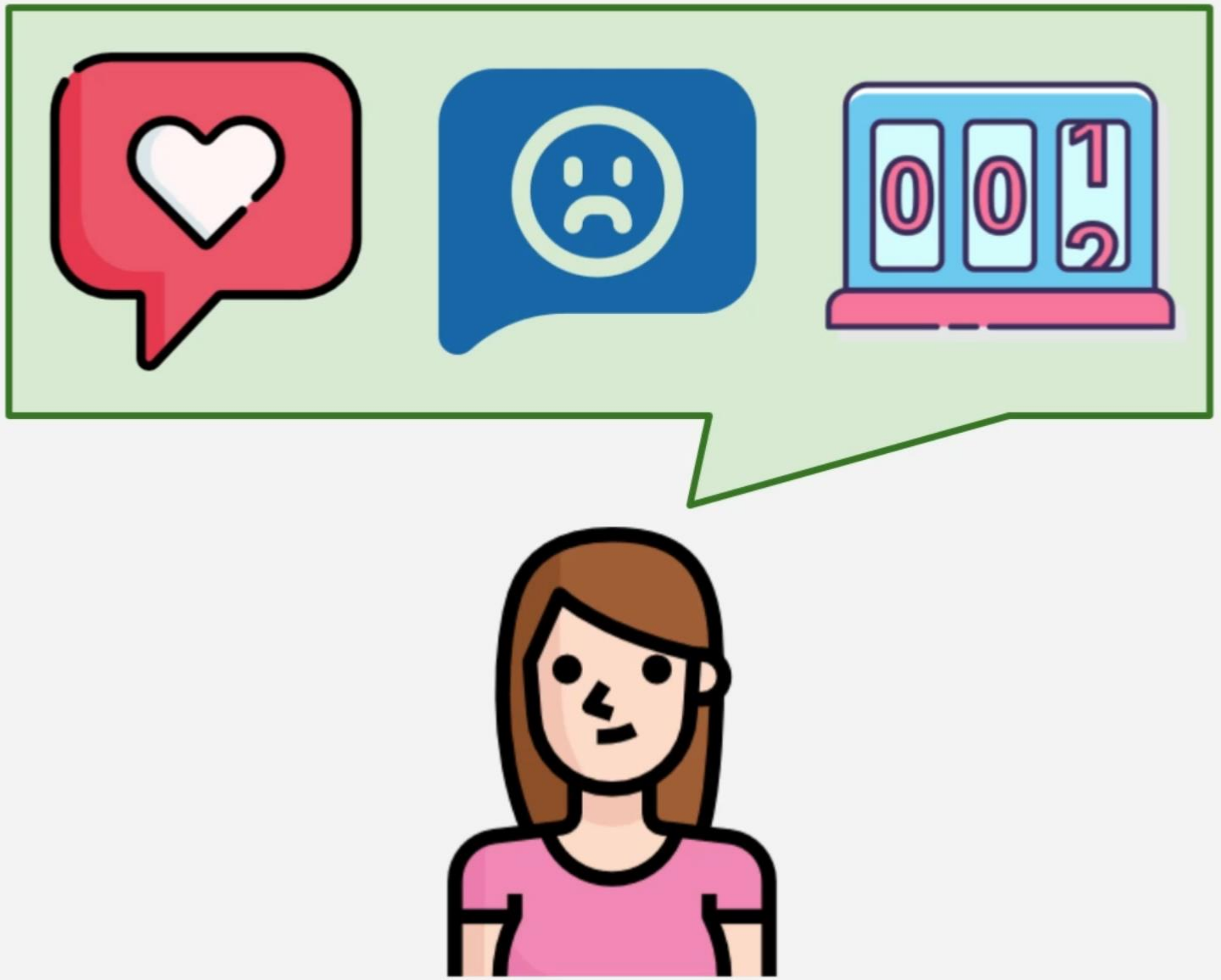


happy, good, helpful, etc

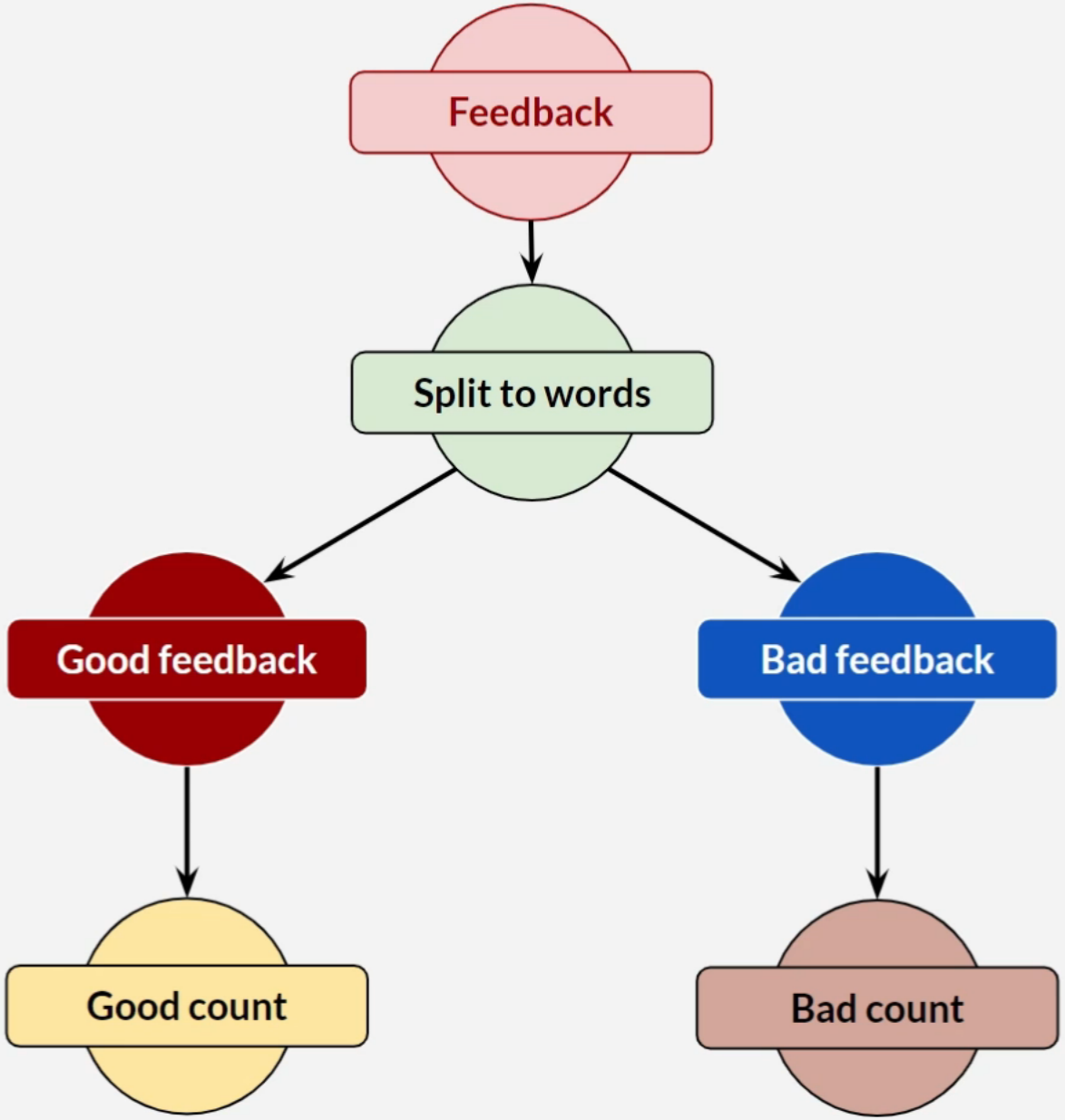


angry, sad, bad, etc

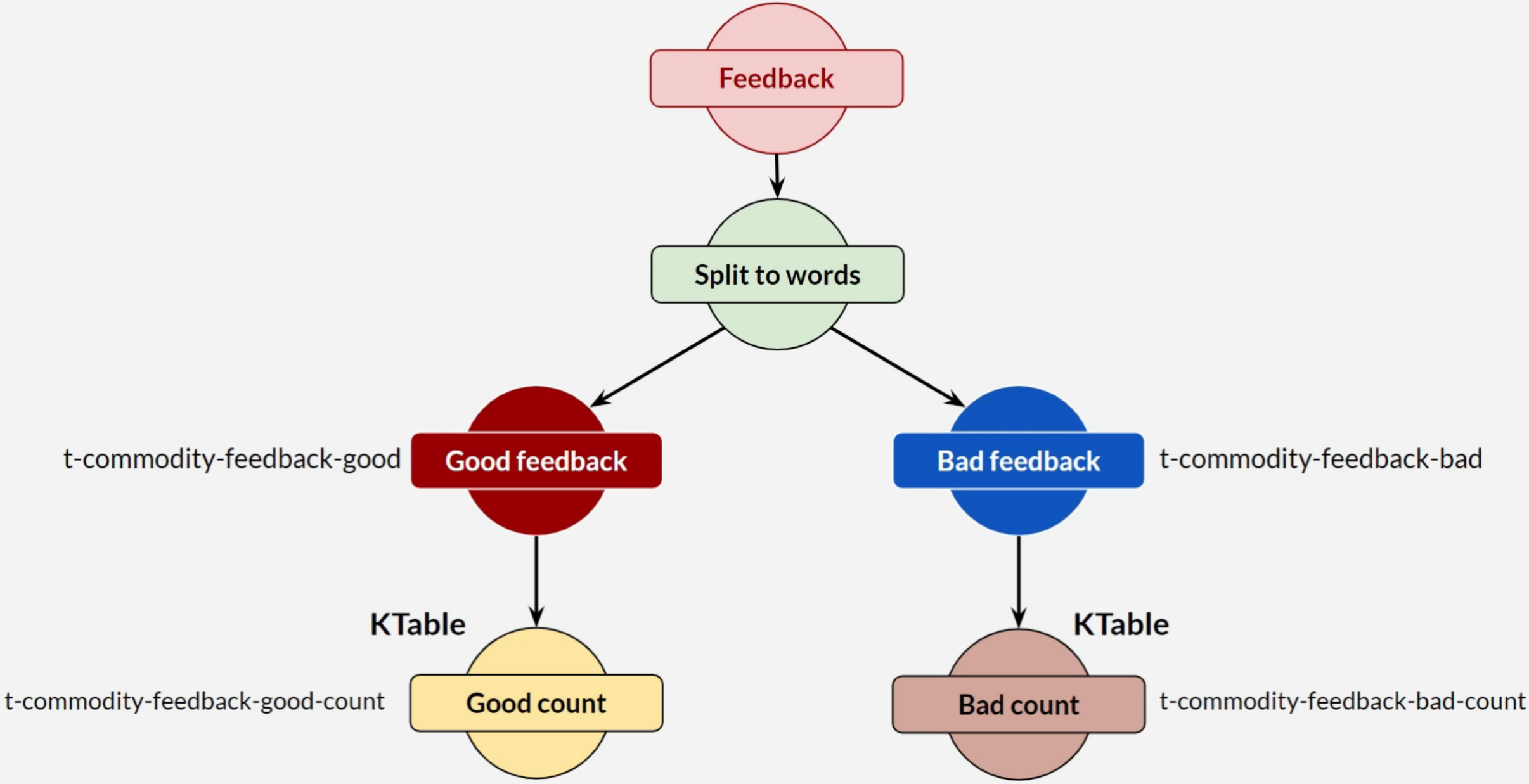
Count The Word



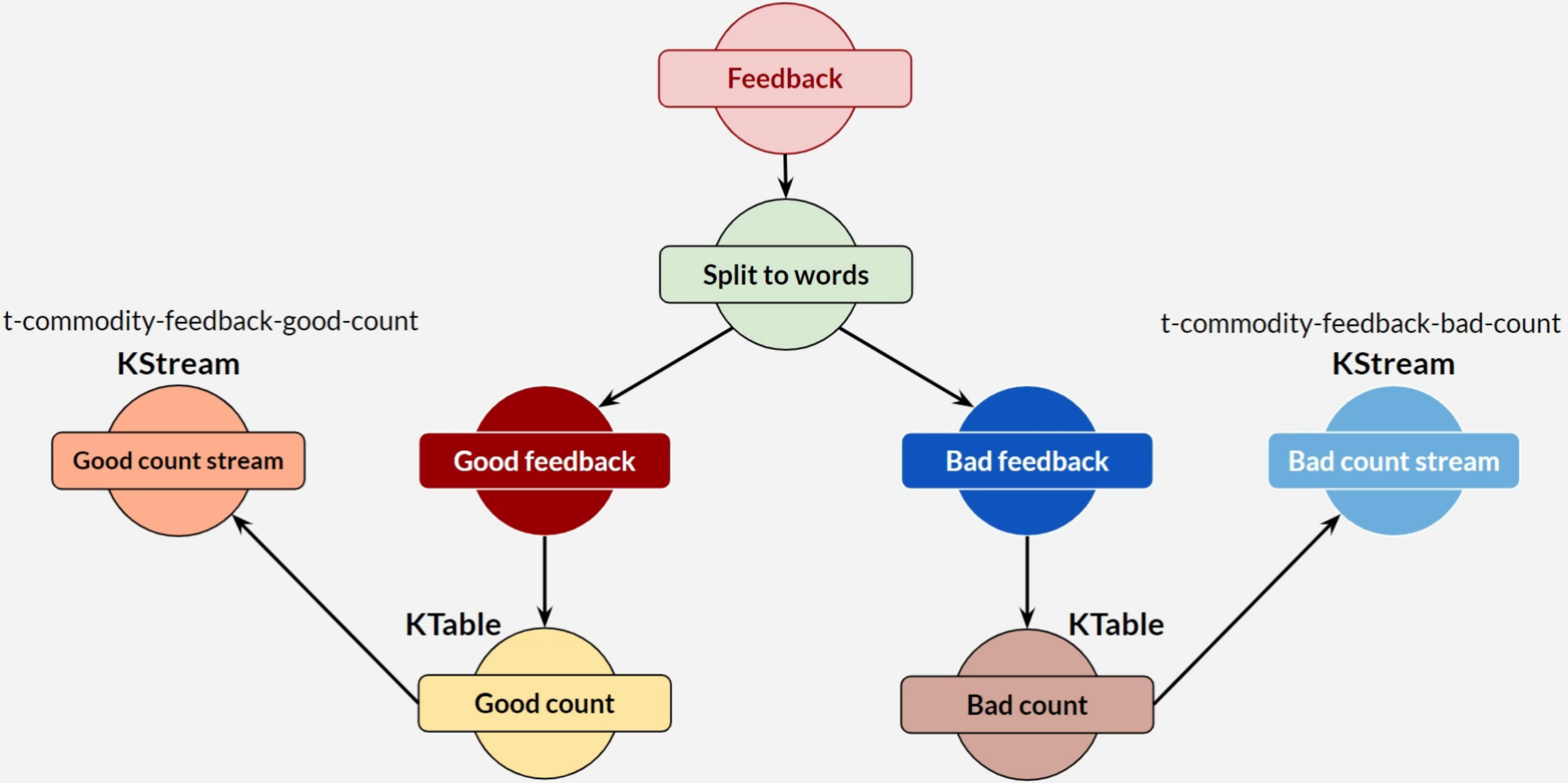
High Level Topology



High Level Topology



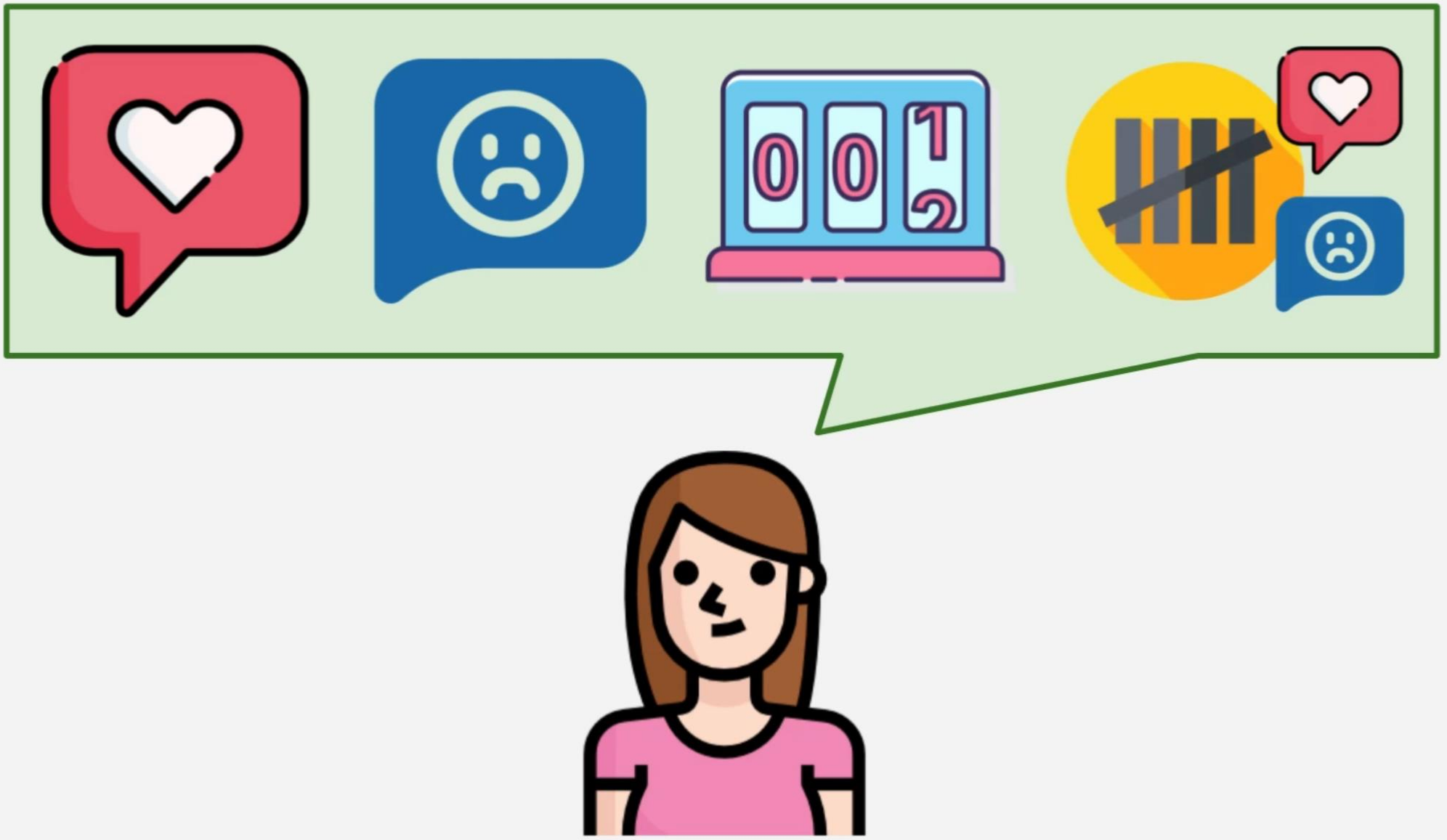
High Level Topology



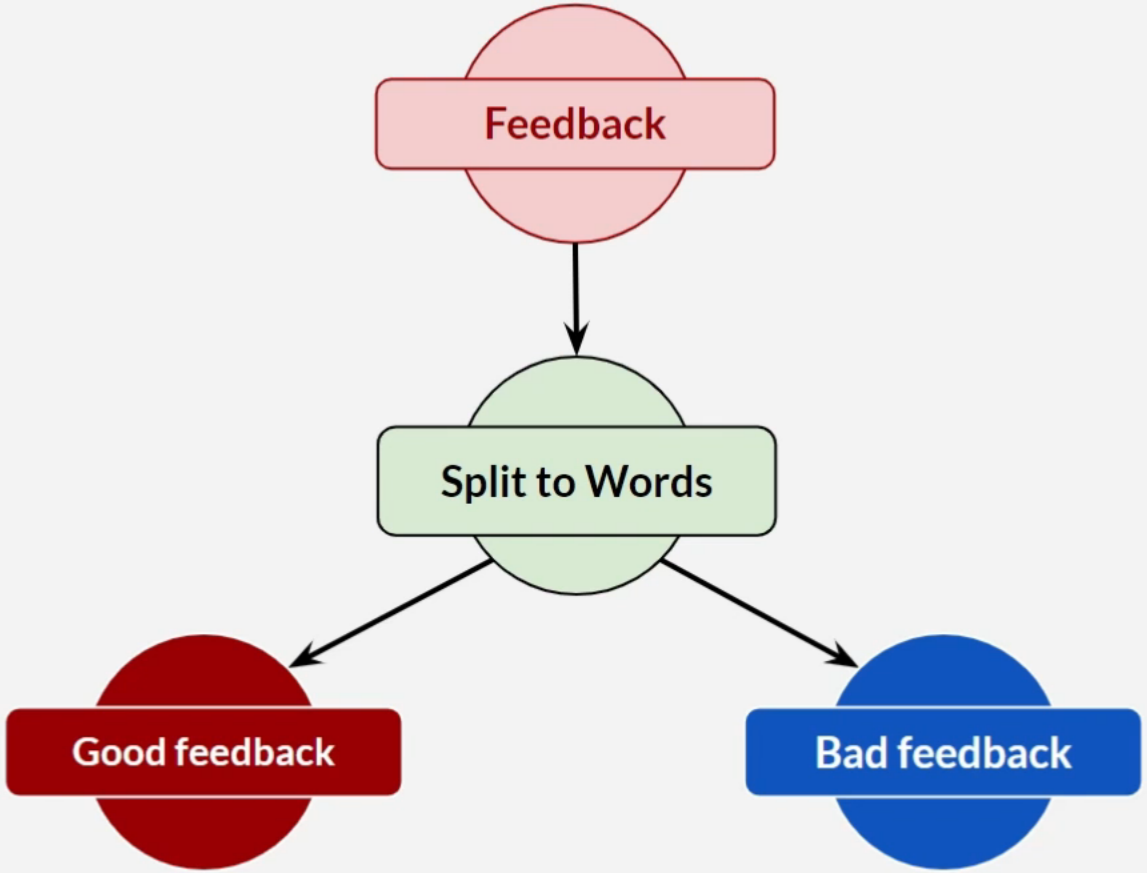
Kafka Stream Configuration

- Default configuration : 30 seconds
- Cache and send
- On `commit.interval.ms`
- Adjust configuration

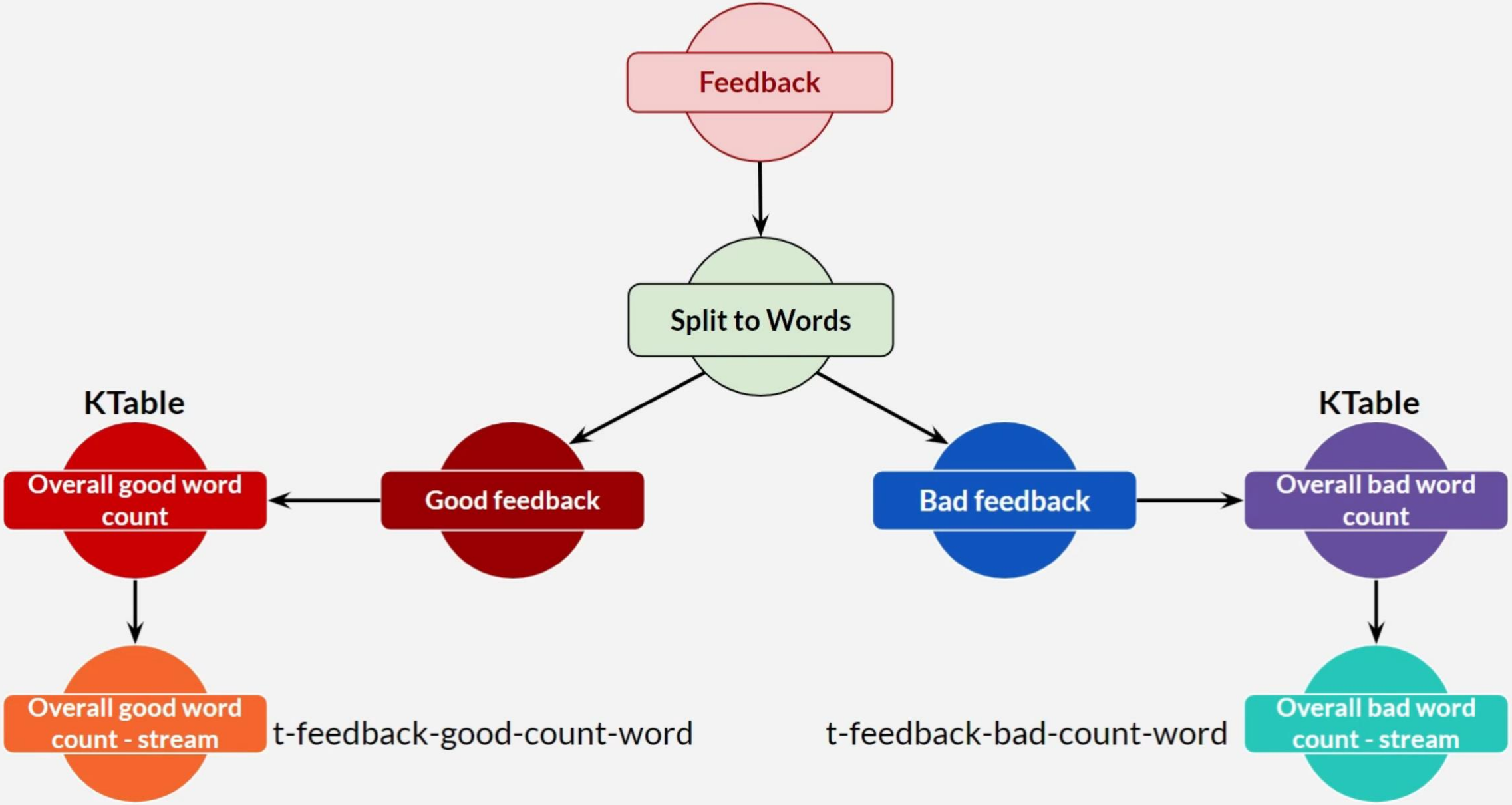
Count The Word



High Level Topology



High Level Topology



Customer Stream

- Mobile & Web -

Different Device, Different Data

Purchase number
Purchase amount
Browser
Operating system

→ t-purchase-web

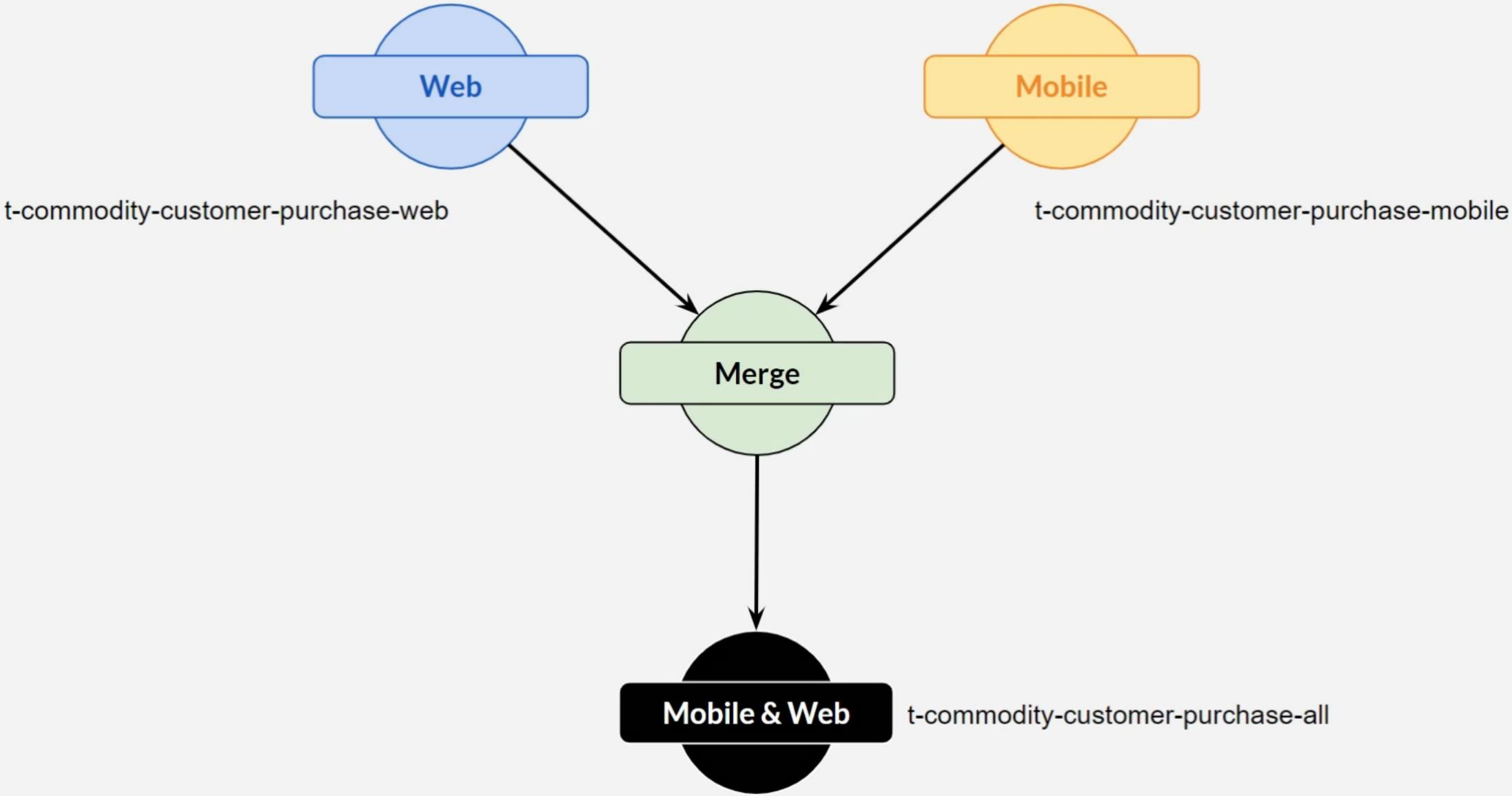
Purchase number
Purchase amount
Mobile app version
Operating system
GPS

→ t-purchase-mobile

t-purchase-all

- Purchase Web 1
- Purchase Mobile 1
- Purchase Mobile 2
- Purchase Web 2
- Purchase Mobile 3

High Level Topology



Create project for Customer Purchase

- CustomerPurchase*.java
- Package: **com.virtusa.kafka**
  - api.request
  - api.server
  - broker.message
  - broker.producer
  - command.action
  - command.service

Two Things, One Customer Preference

Shopping cart

t-shopping-cart

Customer ID
Item name
Cart amount
Cart datetime

t-wishlist

Customer ID
Item name
Wishlist datetime

Wishlist

t-customer-preference

Customer X preference	Customer Y preference	Customer X preference	Customer Z preference	Customer Z preference

Two Things, One Customer Preference

Shopping cart

t-shopping-cart

Customer ID
Item name
Cart amount
Cart datetime

t-wishlist

Customer ID
Item name
Wishlist datetime

Wishlist

t-customer-preference

Customer ID	<Item name, Last datetime> shoppingCartItems <Item name, Last datetime> wishlistItems

Customer X preference

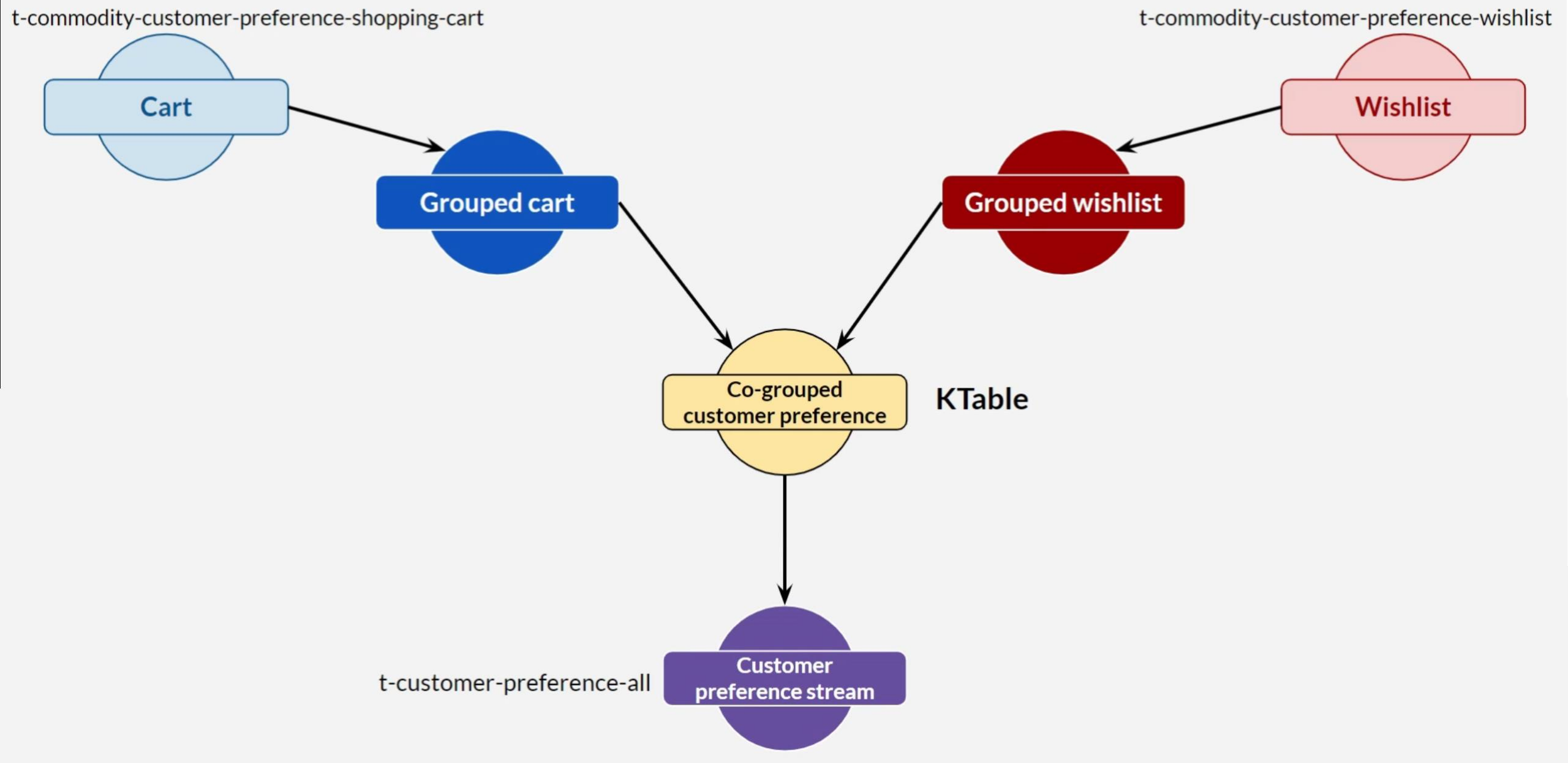
Customer Y preference

Customer X preference

Customer Z preference

Customer Z preference

High Level Topology



Create project for Customer Preference

- CustomerPreference*.java
- Package: **com.virtusa.kafka**
  - api.request
  - api.server
  - broker.message
  - broker.producer
  - command.action
  - command.service

data

aggregate

Venkat
Corporate Trainer & Motivational Speaker