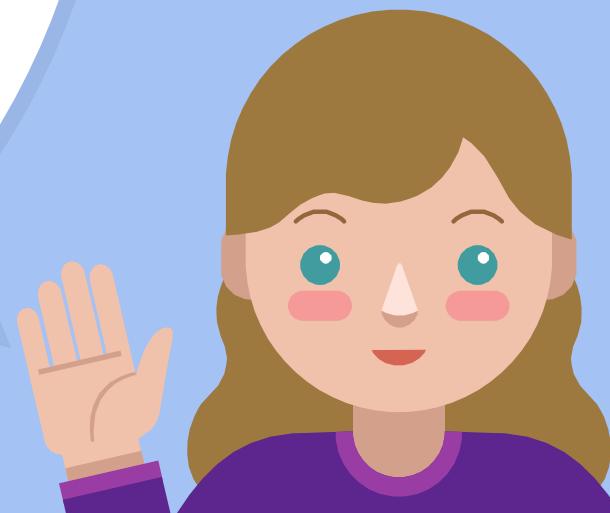
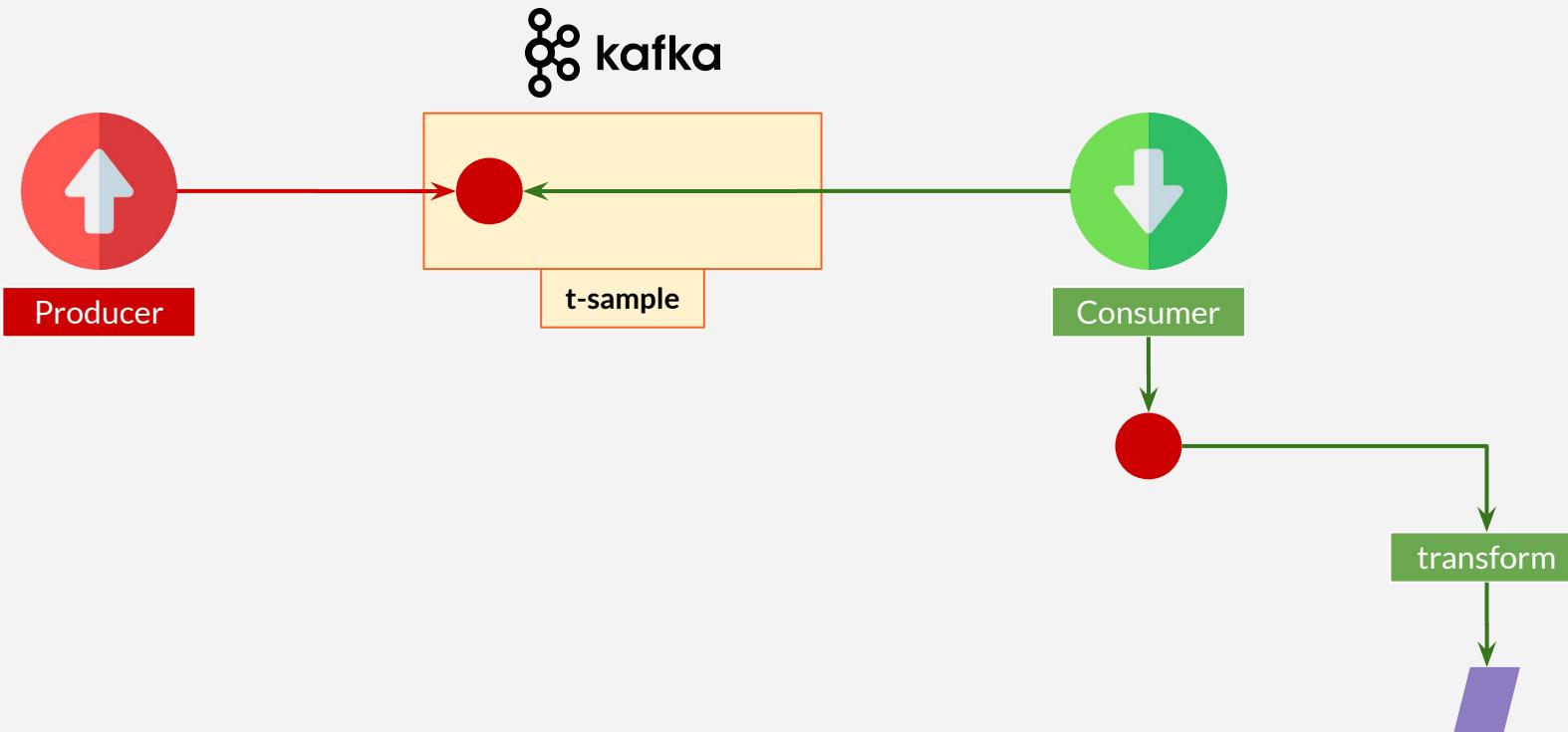


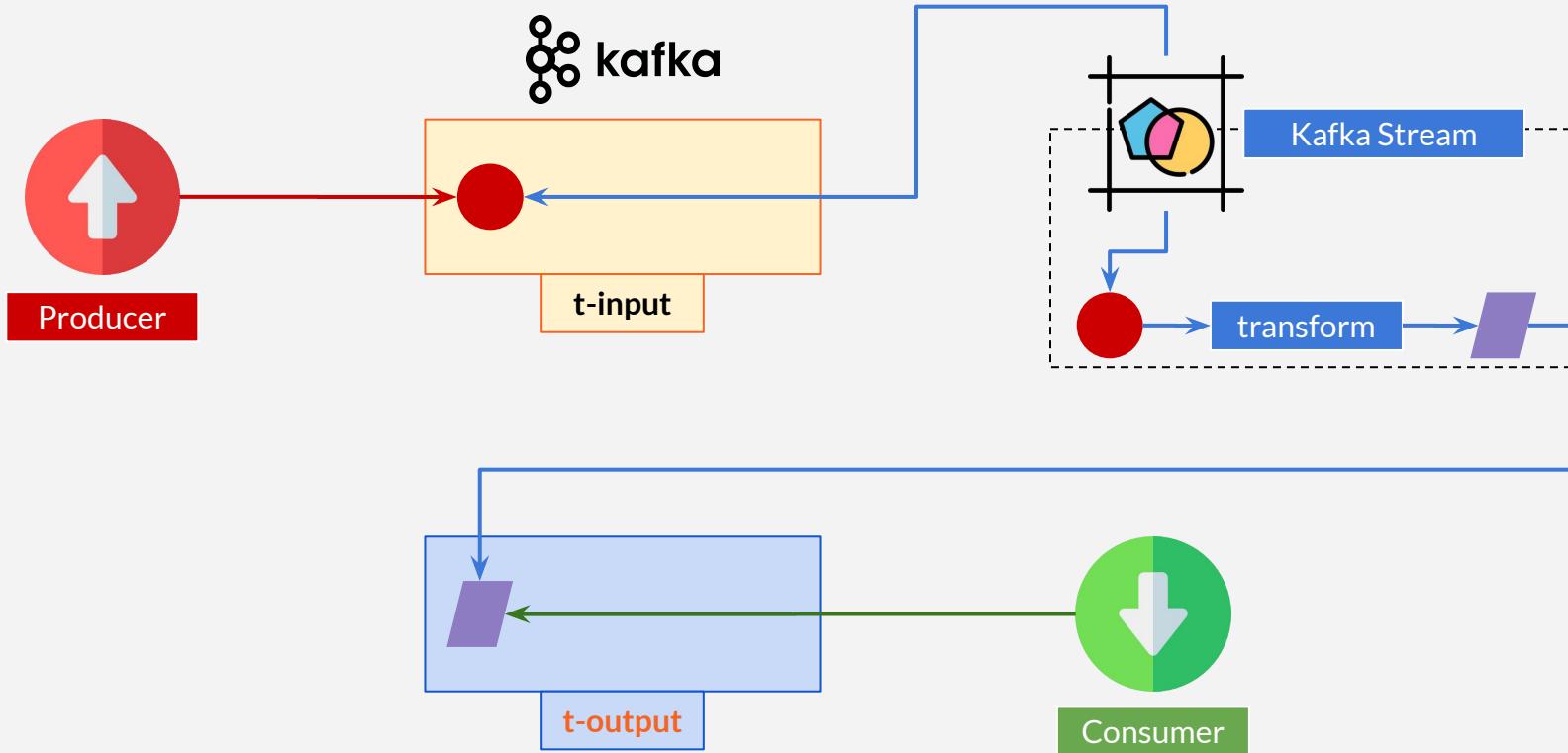
# Introducing Kafka Stream

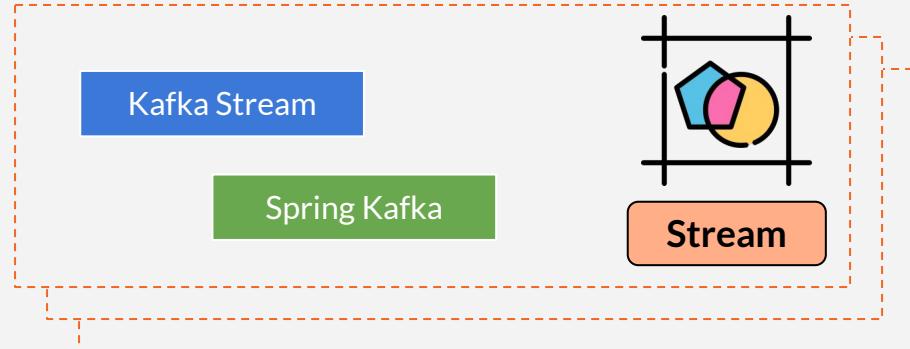
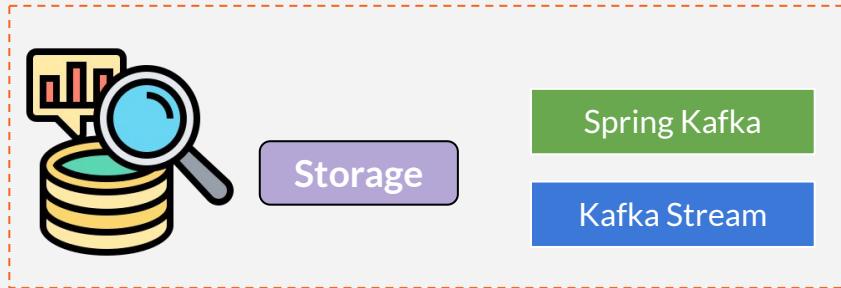
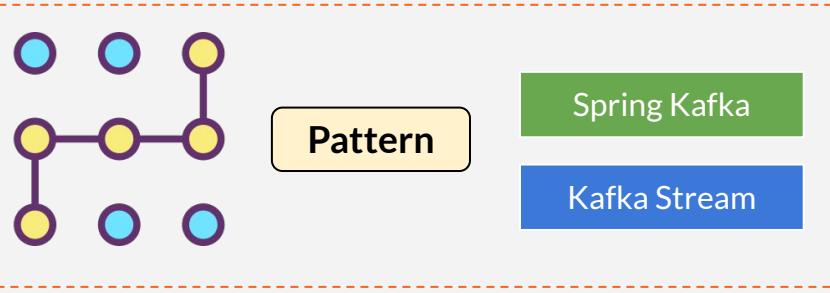


# Data Transformation



# Kafka Stream





# Kafka Stream

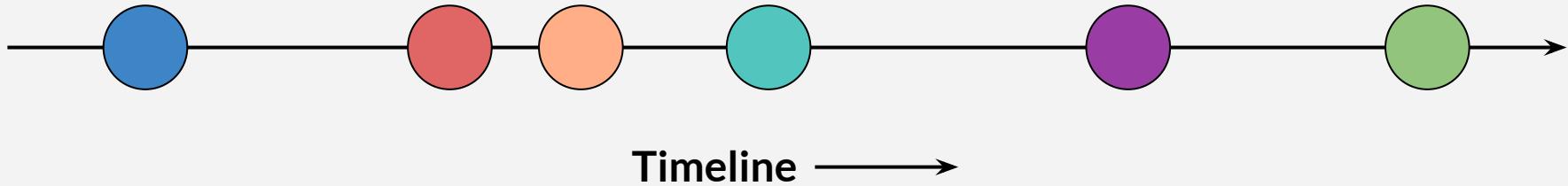
- × Stream processing framework
- × Released on 2017
- × Alternative for Apache Spark, Nifi, or Flink
- × Stream & stream processing?



# Stream Processing

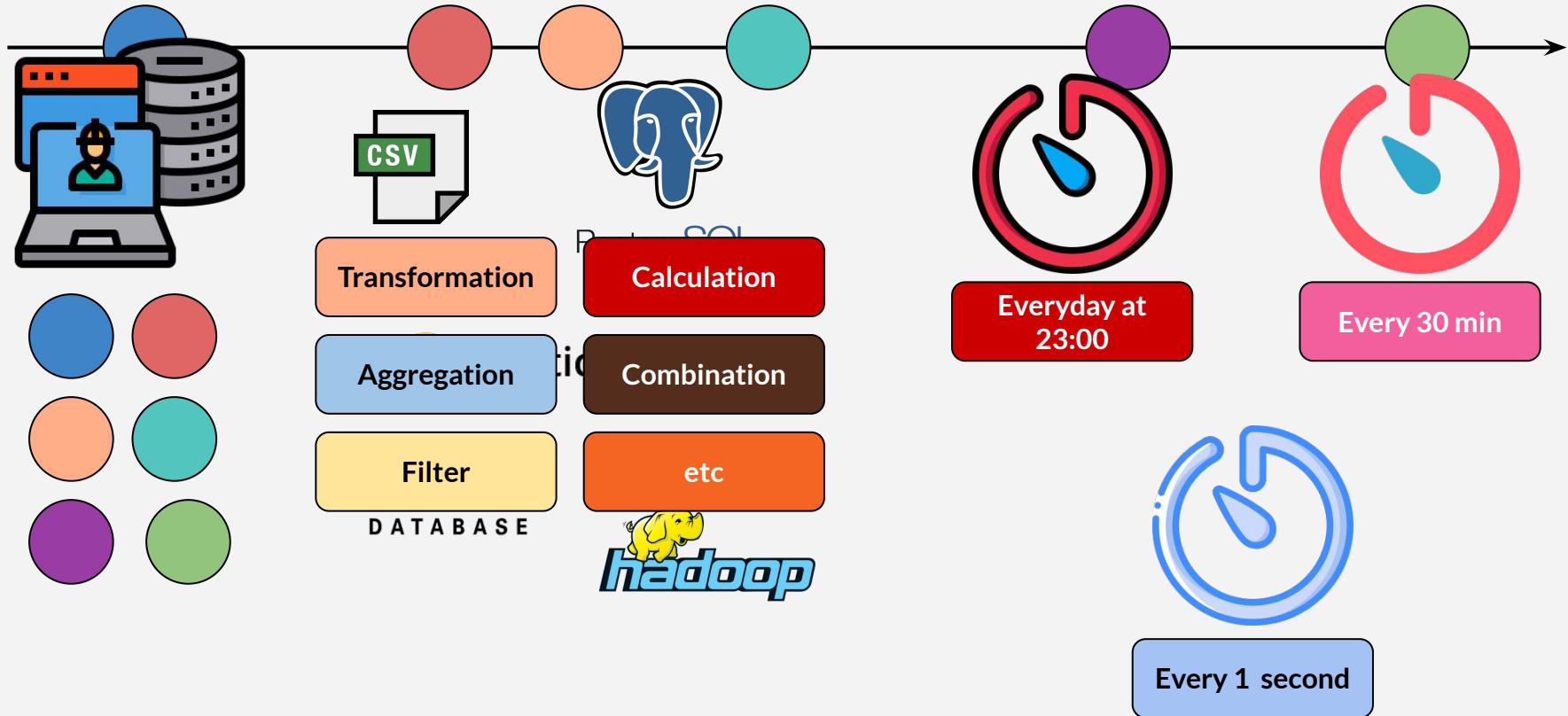


# Data Stream



Each circle represents a data  
Endless  
Data (event) is immutable  
Can be replayed  
Think of Kafka topic as stream

# Data Processing



# Micro Batching



# Micro Batching

Sunday  
23:00

Monday  
23:00

Tuesday  
23:00



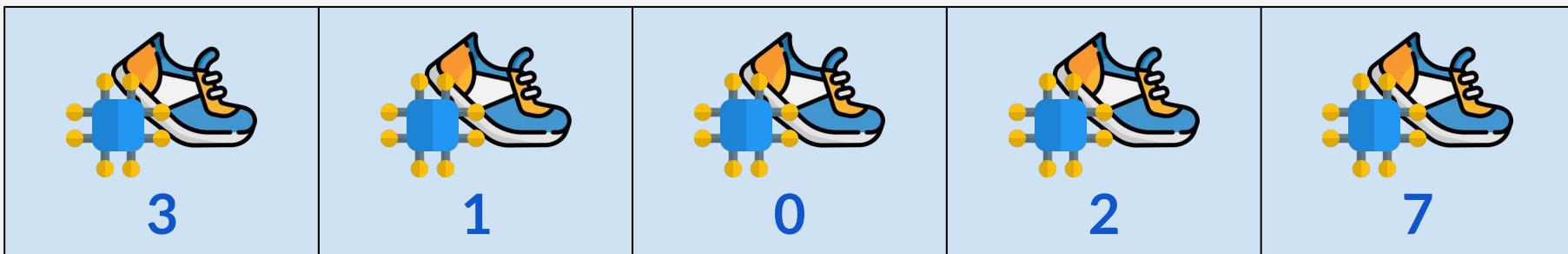
00:01

00:02

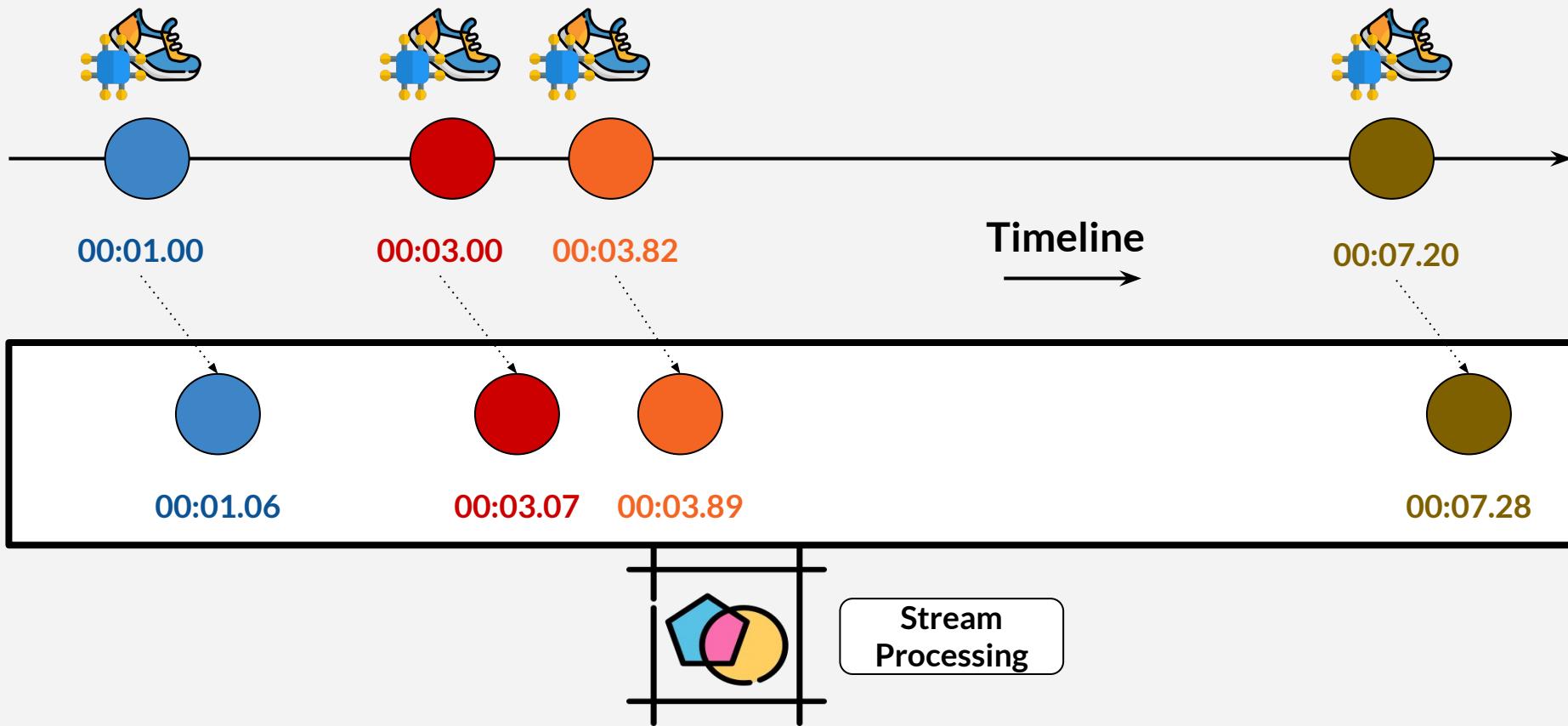
00:03

00:04

00:05



# 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



# To Stream or Not To Stream?

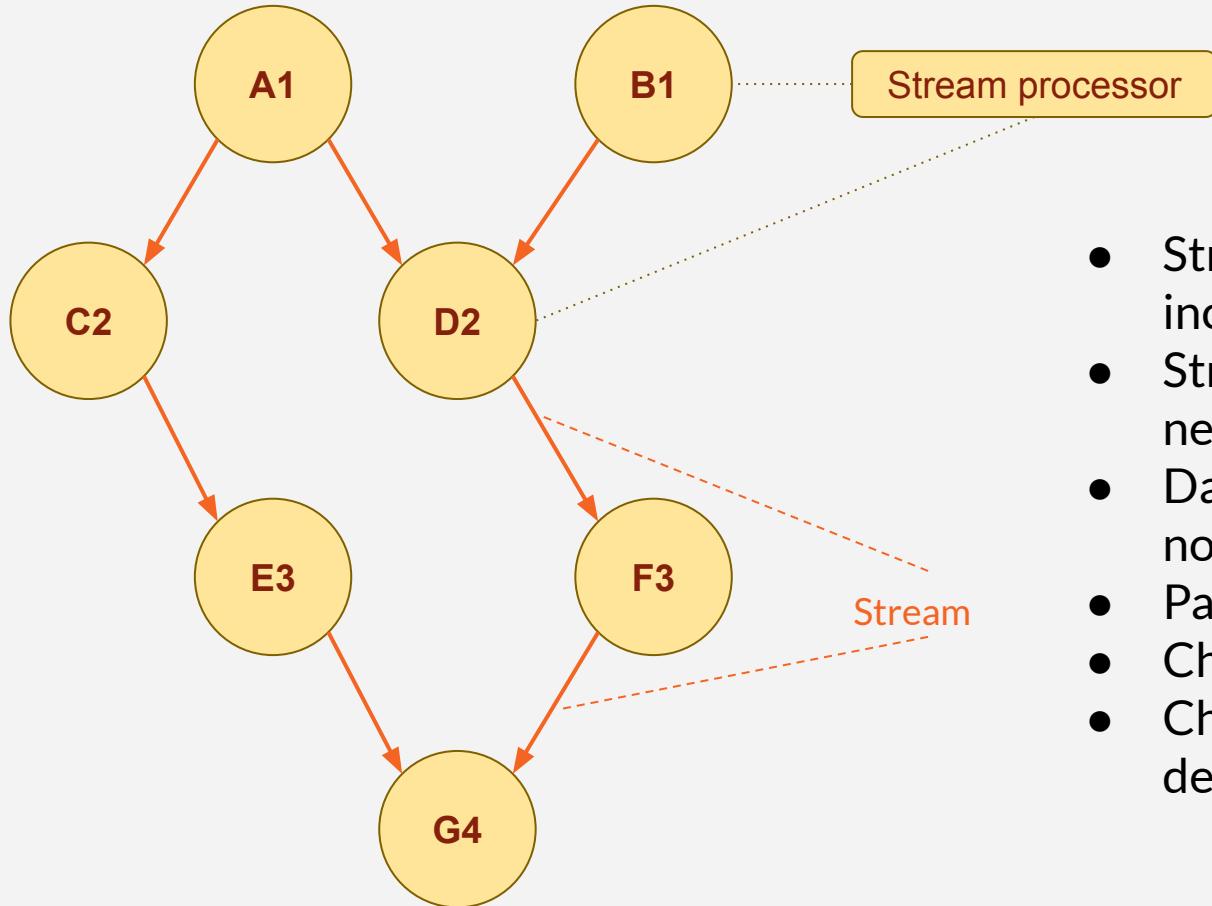
- ✗ Sample : check supply must be above threshold
- ✗ Depends on Service Level Agreement
- ✗ 30 minutes is good : batch
- ✗ Near real time : stream processing



# Kafka Stream Concept

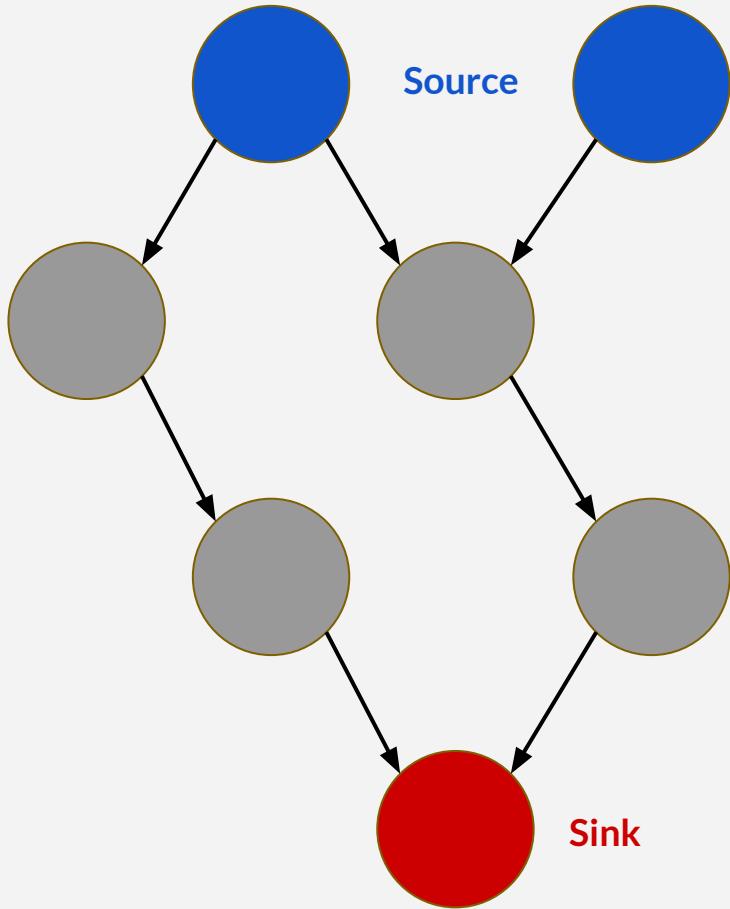


# Topology / DAG



- 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

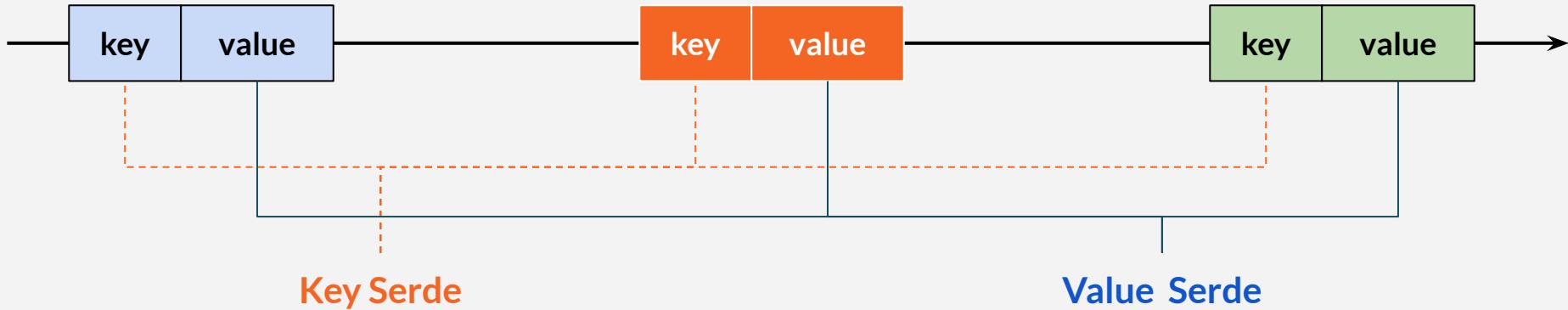
- Does not have downstream
- Receive data from upstream
- Send data to specified kafka topic

# What We Will Learn

- × Low Level Processor API
- × **This Course : Kafka Stream DSL (Domain Specific Language)**
  - × Simple
  - × Provides common data transformation



# Serde (Serializer / Deserializer)



```
Serdes.String()
```

```
Serdes.Long()
```

```
Serdes.ByteArray()
```

```
...
```

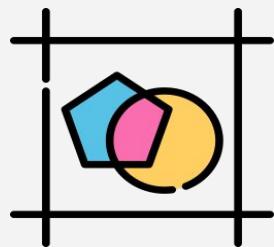
```
new JSONSerde<T>()
```

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

# Kafka Stream Preparation



# What We Will Have



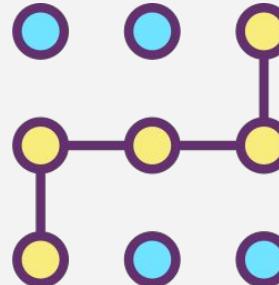
Stream

Order



Storage

Reward



Pattern

# Preparation

- ✗ Delete Kafka docker
- ✗ Re-create kafka containers
- ✗ Download & execute kafka stream scripts on last section (**Resources & References**)
- ✗ Less detail on kafka console
- ✗ Code style with Java lambda & functional (not in course scope)



# Download Source Code

- × Available on last section of the course
- × Source code, postman, kafka script



# 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**
- × Import the project



# Additional Projects

- × Use projects from lesson kafka & microservice
- × Names will be **kafka-stream-xxxxx**
- × Same codebase with previous
- × Might have additional dummy transactions
- × Copy all java files under broker.message from kafka-stream-order to kafka-stream-sample



# AI Assistant

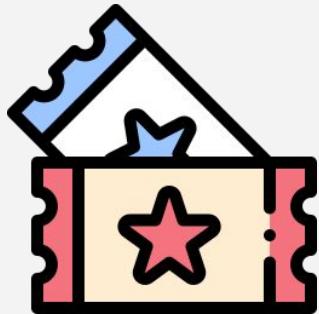
- × Use Github Copilot as AI assistant
- × Non-deterministic
- × Less reliable due to increased complexity
- × Use from chat panel / inline chat / source code comment / semi-transparent autocomplete



# Hello Kafka Stream



# Promotion



PrOmoTlOn7



PROMOTION7

# Kafka Stream Basic Usage

build.gradle

```
implementation 'org.apache.kafka:kafka-streams:{$version}'
```

```
// define configuration
Properties props = new Properties();
props.put(...)
```

```
// create topology
KStreamBuilder builder = ...;
KStream stream = ...;
...
KafkaStreams stream = new KafkaStreams(builder, props);
```

```
// start stream
stream.start();
```

```
// stop stream
Runtime.getRuntime().addShutdownHook(new Thread(stream::close));
```

SpringKafkaStreamApp.java

# Kafka Stream With Spring

build.gradle

```
implementation 'org.springframework.kafka:spring-kafka'  
implementation 'org.apache.kafka:kafka-streams'
```

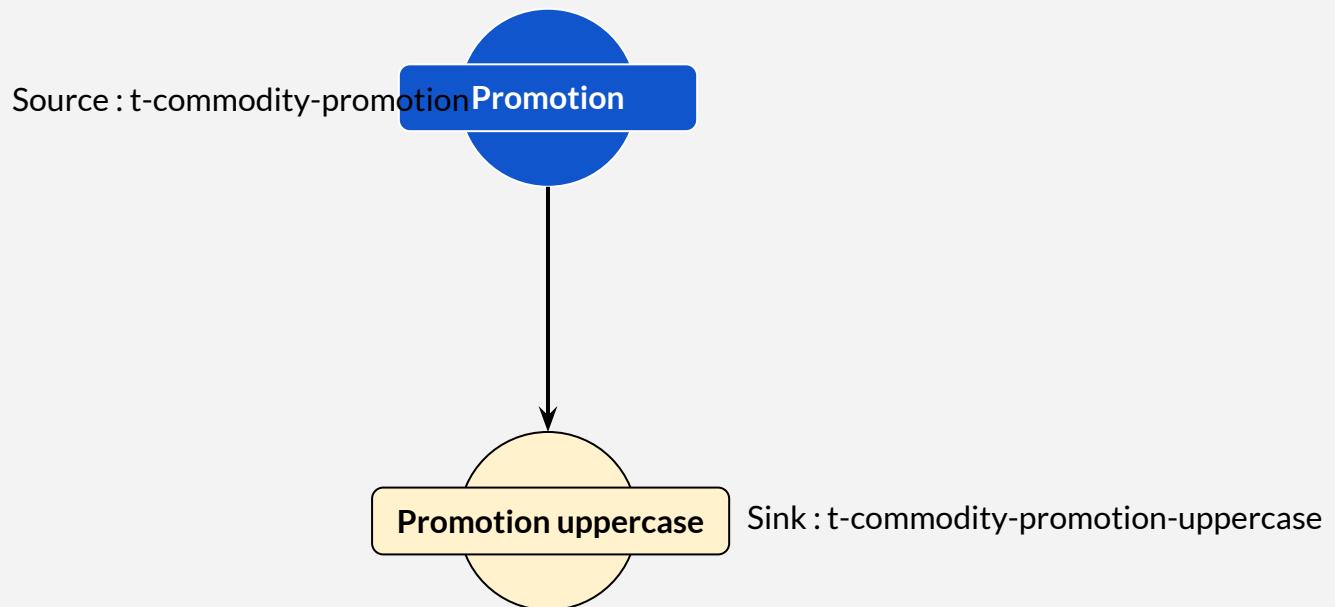
```
@Configuration  
@EnableKafkaStreams  
public class KafkaStreamConfig {  
    // define configuration  
}
```

KafkaStreamConfig.java

```
@Configuration  
public class MyStream {  
  
    @Bean  
    public KStream<String, String> kStream(StreamsBuilder builder) {  
        // define topology  
    }  
}
```

MyStream.java

# Topology



# Uppercase?

null	{"promotionCode":"rabbit35"}
------	------------------------------



# Hello Kafka Stream

## Alternative Coding Style



```
@Configuration
public class MyStreamStyleOne {

    @Bean
    KStream<?, ?> streamMethod(StreamsBuilder builder) {
        var sourceStream = builder.stream(...);

        // ... process the stream

        return sourceStream;
    }
}
```

```
@Component
public class MyStreamStyleTwo {

    @Autowired
    void streamMethod(StreamsBuilder builder) {
        var sourceStream = builder.stream(...);

        // ... process the stream
    }
}
```

# Coding Style

- × The course will use both styles
- × AI assistant can suggest using either style
- × Either style will work



# Promotion Uppercase Stream Using String Serde



# Promotion Uppercase Stream Using JSON Serde



# Promotion Uppercase Stream With AI Assistant



# Custom JSON Serde



# Kafka Message Format

- × Custom data format : comma separated, tab separated, ||| separated, etc
- × Take message as string & convert
- × Custom serde
- × Serializer + Deserializer + implements Serde<T>
- × Example : Custom JSON serde



# Segment



## Topic

### *Partition 0*

Segment 0  
Offset 0-498

Segment 1  
Offset 499-1026

Segment 2  
Offset 1026-1792

Segment 3 (active)  
Offset 1793...

### *Partition n*

Segment 0  
Offset 0-510

Segment 1  
Offset 511-992

Segment 2 (active)  
Offset 993...

- Data writes happen on the active segment
- Each partition only has one active segment

# Segment Configuration

- ✗ **log.segment.bytes**
  - ✗ Maximum size of single segment (in bytes)
  - ✗ Default 1 GB
  - ✗ Closed when exceed 1 GB, new segment created
- ✗ **log.segment.ms**
  - ✗ How long before a segment is closed
  - ✗ Default 1 week
  - ✗ Closed when exceed one week (albeit size is less than configured segment bytes), new segment created



# Segment

- × Kafka log is series of segments
- × Why care?
- × Material for the next lesson about Log Cleanup

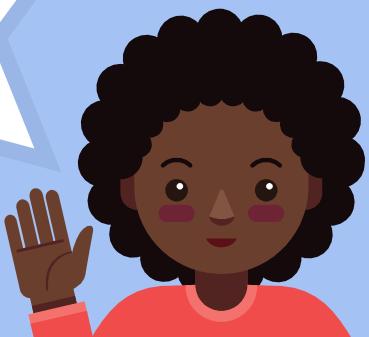


# Log Cleanup



# Log

- × Append-only data sequence stored as segments
- × All published data stored in Kafka log
- × Retain messages (records) for certain period
- × Stored at disk, size can grow
- × Immutable
- × High throughput & fault tolerance



# Log Cleanup

- × Managing data retention & size
- × Removing outdated / unnecessary data on predefined policies
- × Policies
  - × *delete* (removes data)
  - × *compact* (retain only latest value per key)
- × The importance of log cleanup
- × Maintain health & performance of Kafka clusters



# Log Cleanup Policy : *delete*



# Policy : *delete*

- × Default policy for user topics
- × Data will be deleted based on log retention period
  - × Default retention is 168 hours (1 week)
- × Data can also be deleted by the maximum segment size
  - × Default maximum size is -1 (infinite)



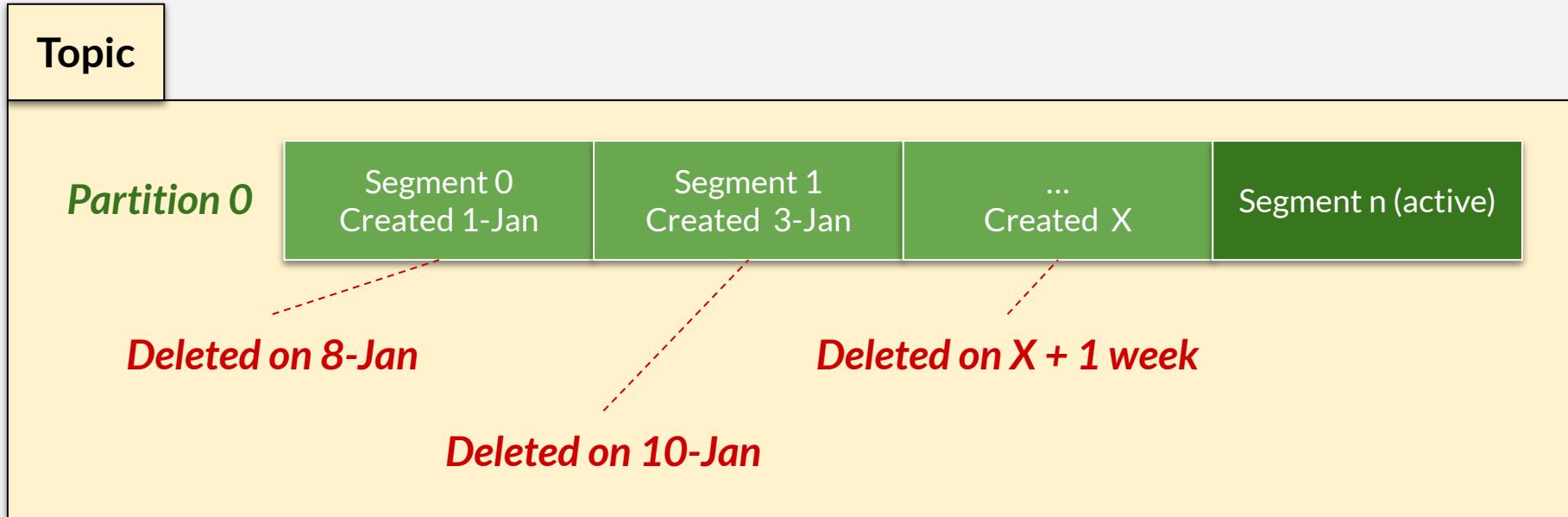
# Policy : *delete*

- × Configurable
- × **log.retention.hours**
  - × How long to keep the data
  - × Default is 168 hours (1 week)
  - × Impact on adjusting the value
  - × **log.retention.minutes** or  
**log.retention.ms** also available
- × **log.retention.bytes**
  - × Maximum size (in bytes) for each topic
  - × Default is -1 (infinite)
  - × Adjust to limit topic log's size

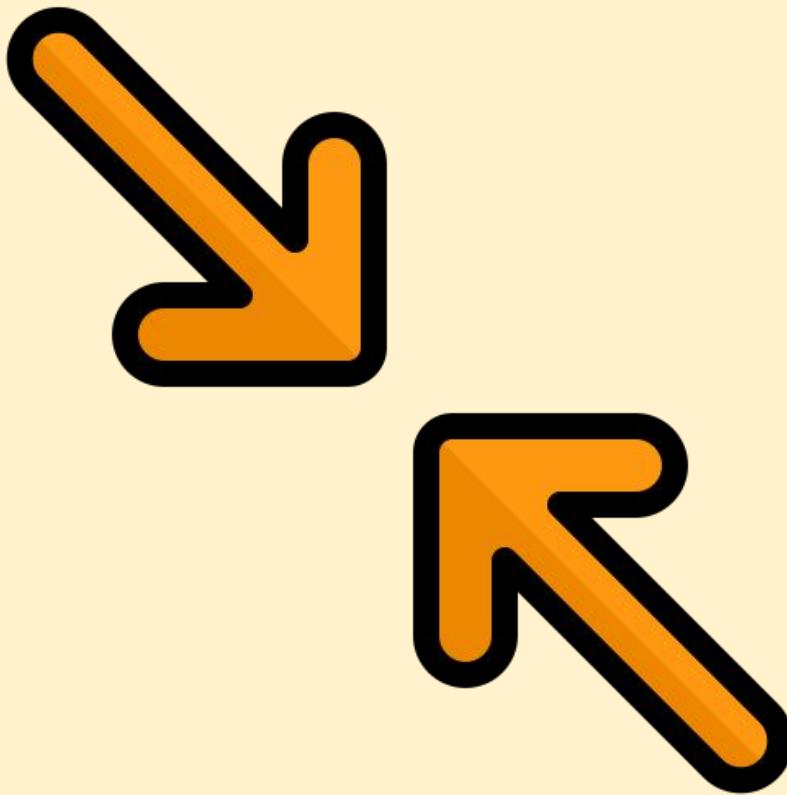


# Log Deletion

168 hours retention period & infinite retention size



# Log Cleanup Policy : *compact*



# Policy : *compact*

- × Retain latest value of unique key within partition
- × Default policy for Kafka internal topic
- × When to use
- × Basic working process
- × Discard redundant records and maintain compact data representation



# 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	40	21	242

# Policy : *compact*

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



# Policy : *compact*

- × Configurable
- × **segment.ms**
  - × Maximum time to wait to close active segment
  - × Default is 1 week
- × **segment.bytes**
  - × Maximum time to wait to close active segment
  - × Default is 1 Gigabyte
- × **min.compaction.lag.ms**
  - × Default is 0
  - × How long to wait before a message can be compacted



# Policy : *compact*

- ✗ **delete.retention.ms**
  - ✗ Default is 24 hours
  - ✗ How long to wait before deleting data marked for compaction
- ✗ **min.cleanable.dirty.ratio**
  - ✗ Default is 0.5
  - ✗ Higher value means less frequent cleaning
  - ✗ Lower value means more frequent cleaning



# Policy : *compact*

- × Compaction frequency affected by traffic, **segment.ms**, and **segment.bytes**
- × Log compaction background process will open segment (file)
- × Can cause error "too many open files" if too many segments to be compacted



# Settings

**segment.ms** = 3600000 (one hour)

**segment.bytes** = 1000000000 (roughly one Gigabyte)

	<i>Case 1 (high throughput)</i>	<i>Case 2 (low throughput)</i>
<b>[A] Throughput</b>	1 million messages / second	10k messages / second
<b>[B] Message size</b>	100 bytes / message	100 bytes / message
<b>[C] Total byte per second = [A] * [B]</b>	100 megabytes (0.1 GB) / second	1 megabyte (0.001 GB) / second
<b>[D] Time to produce 1 GB data = 1 GB / [C]</b>	10 seconds	1000 seconds
<b>Segment per hour = 3600 / [D]</b>	360 segments / hour	3.6 segments / hour

# Policy : *compact*

- × Compaction in case 1 (high throughput) happen more frequently compared to case 2 (low throughput)
- × Affect log compaction frequency
- × Might need configuration adjustment



# Stream & Table



# Stream & Table

**KStream**

Ordered sequence of messages

Unbounded

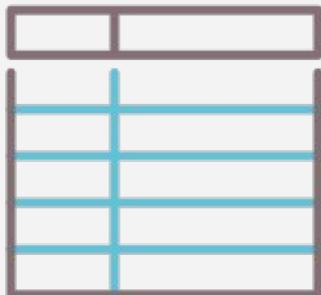
**Inserts** data



**Stream**

---

**Table**



**KTable**

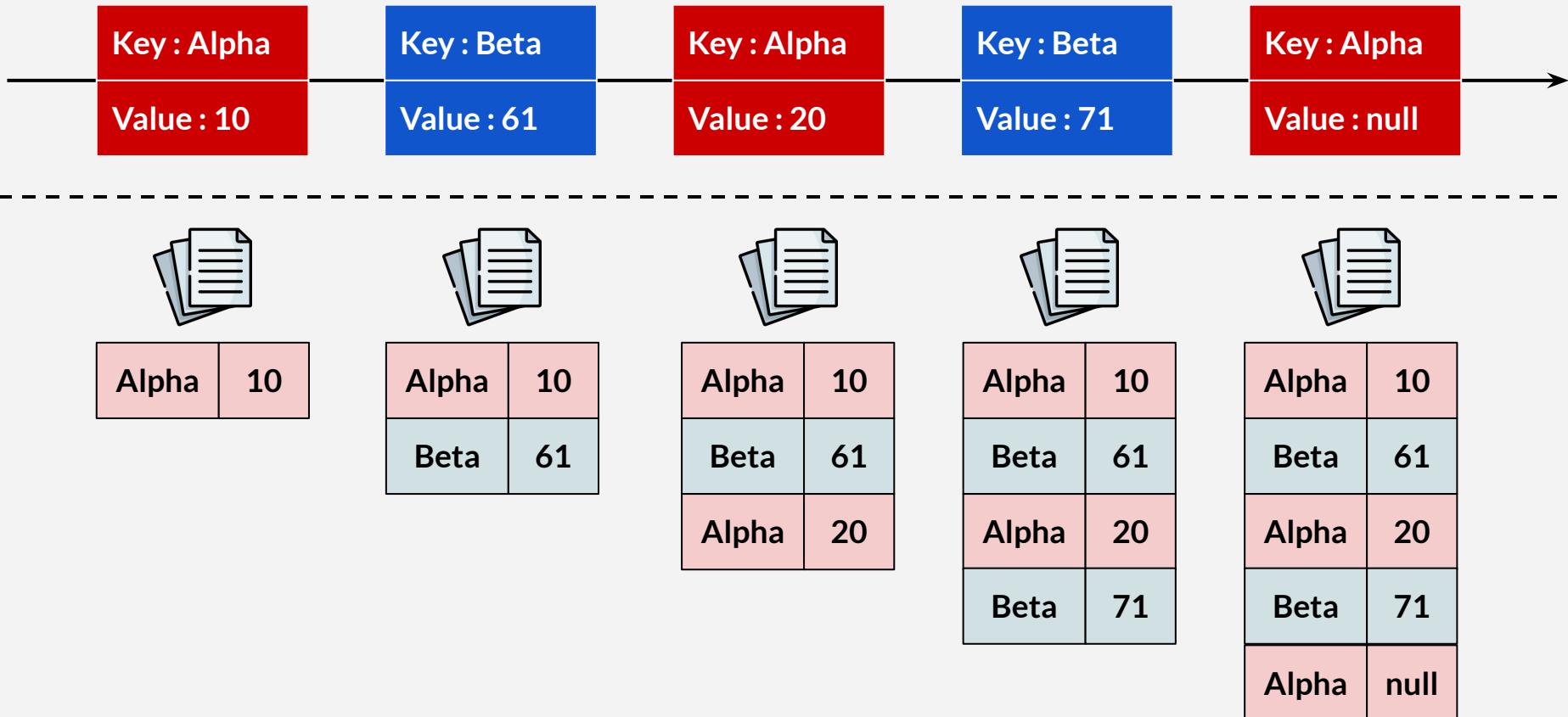
Unbounded

**Upserts** data : insert or update based on key

Delete on null value

Analogy : database table

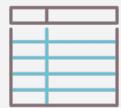
# KStream : Inserts Data



# KTable : Upserts / Delete Data

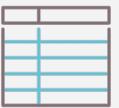


Insert Alpha



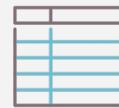
Alpha	10
-------	----

Insert Beta



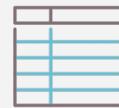
Alpha	10
Beta	61

Update Alpha



Alpha	20
Beta	61

Update Beta



Alpha	20
Beta	71

Delete Alpha



Beta	71
------	----

# When to Use KStream / KTable

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



# Available Operations Theory

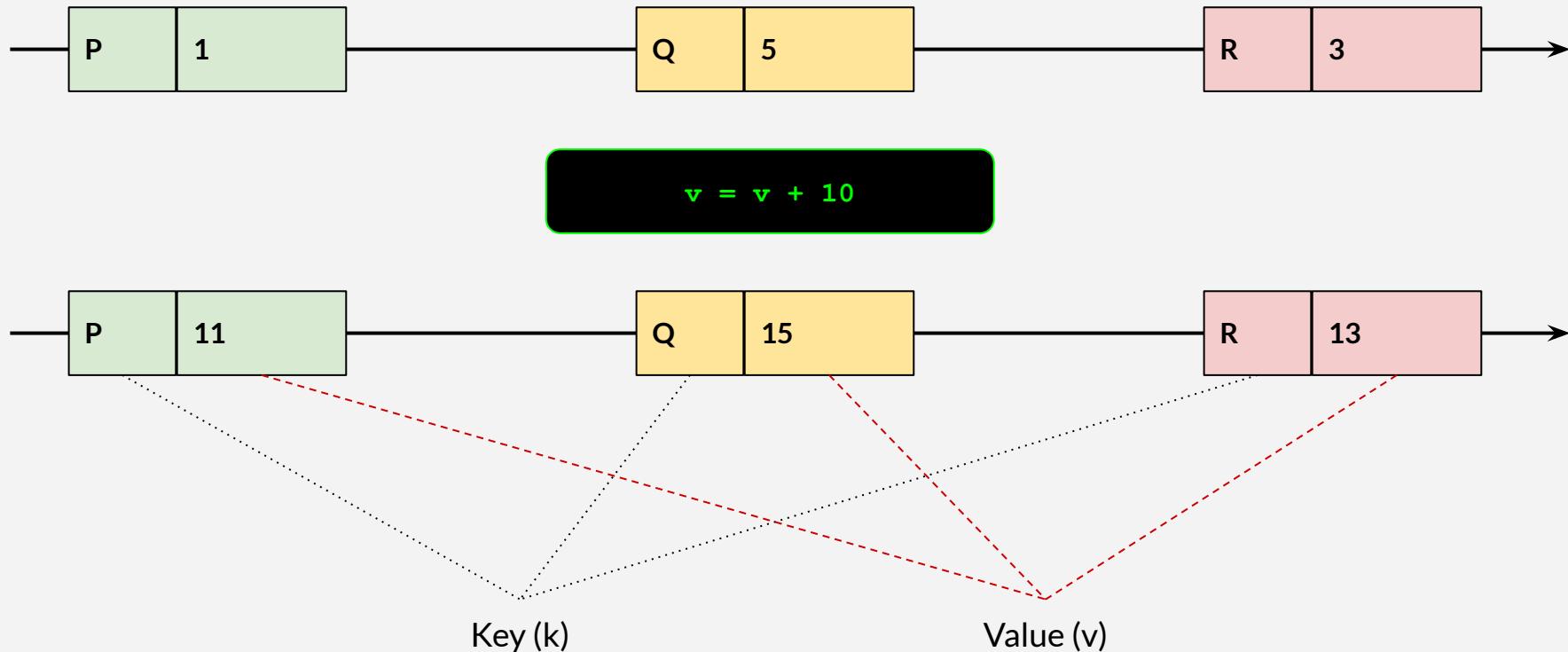


# About This Lesson

- ✗ Kafka streams operations
- ✗ This lesson, then code
- ✗ Code, then this lesson



# Diagram



# Intermediate & Terminal Operation

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



# Stateless & Stateful

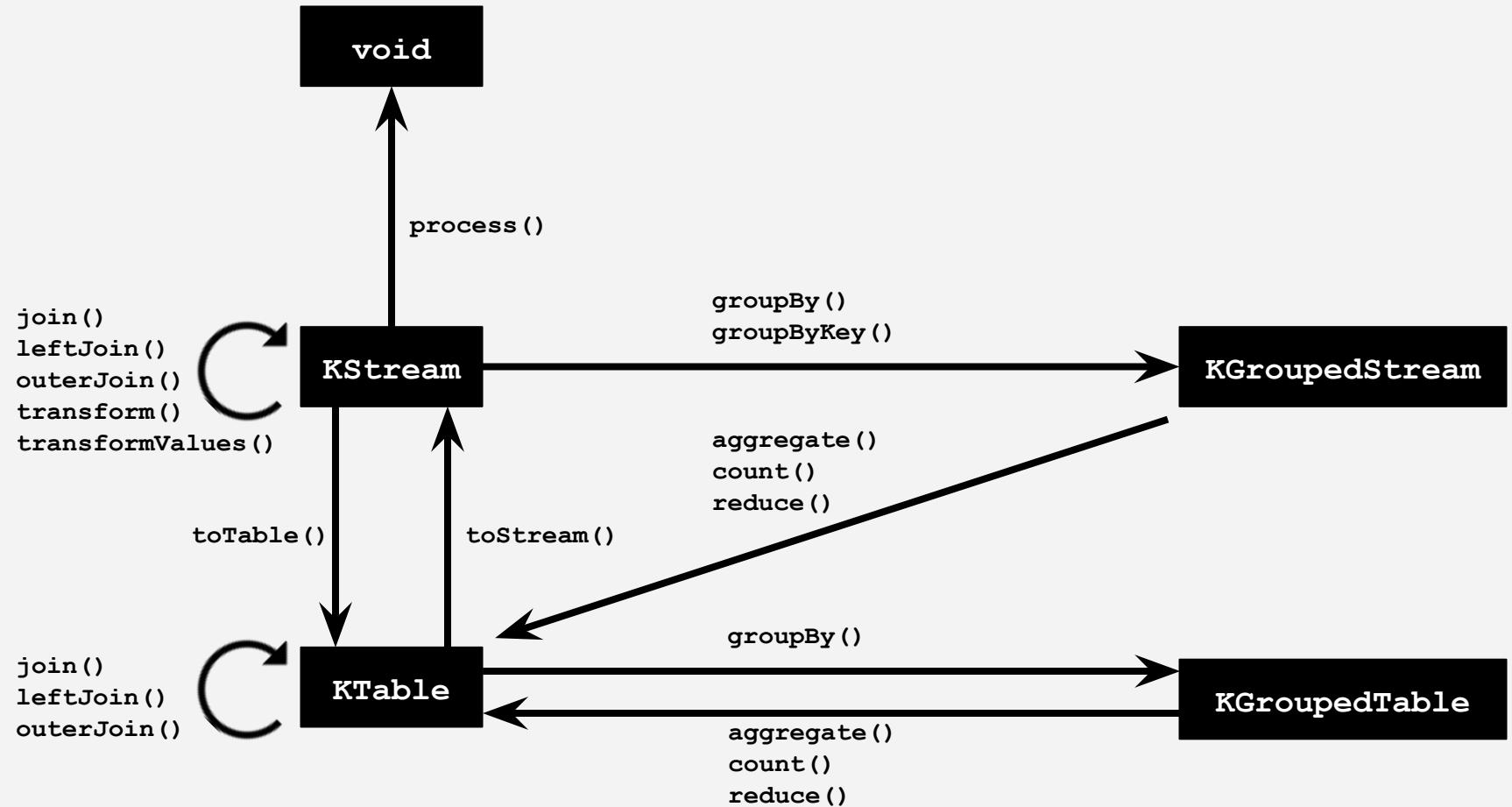
- × Stateless : process each record independently
  - × Ideal use cases?
- × Stateful : maintains state across multiple records
  - × Ideal use cases?
- × Summary



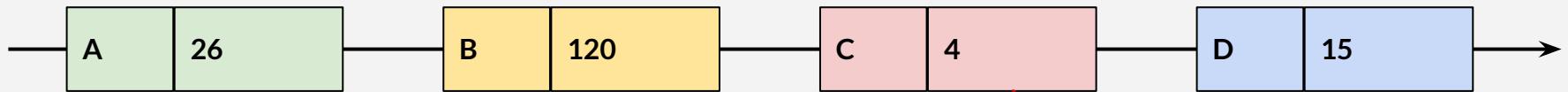
# Reminder : Key & Partition

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



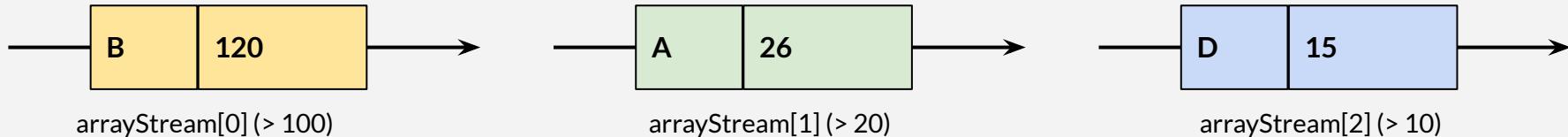


# branch



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

Dropped, no match



- 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

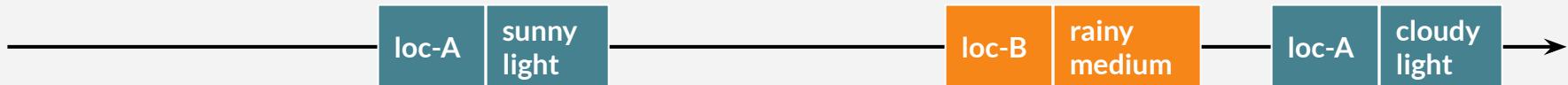
*weather*

# 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));
```



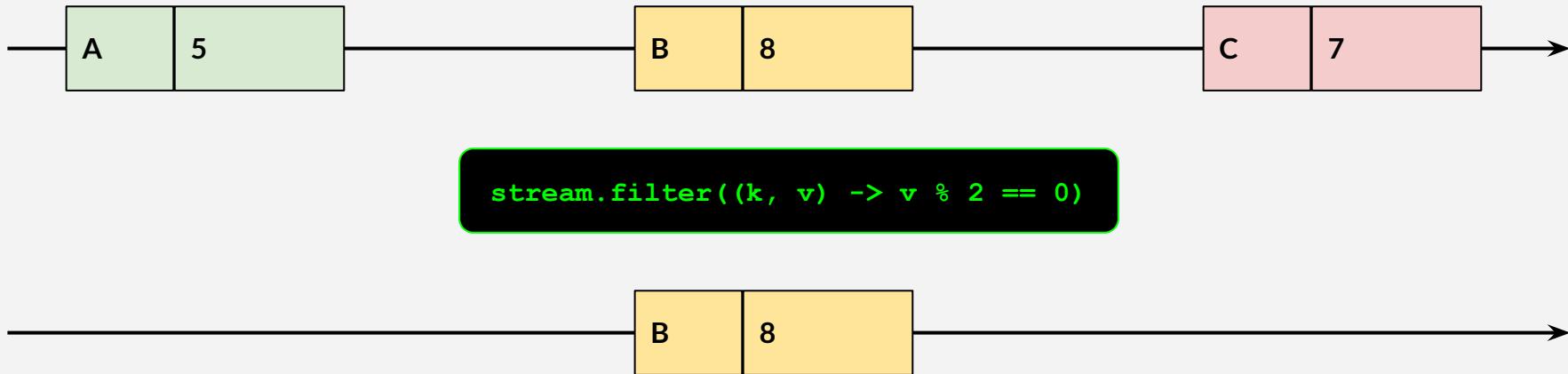
- Intermediate operation
- KStream
- Need aggregator for each cogroup
- This sample : String key, JSON value
- Difference with merge()

# 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;  
    }  
};
```

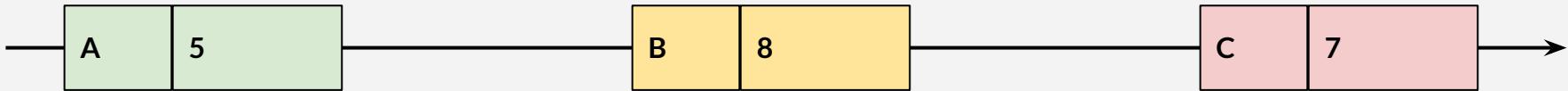
```
Aggregator<String, String, Location> TRAFFIC_AGGREGATOR = new Aggregator<String, String, Location>() {  
  
    @Override  
    public Location apply(String key, String value, Location aggregate) {  
        aggregate.setTraffic(value);  
        return aggregate;  
    }  
};
```

# filter

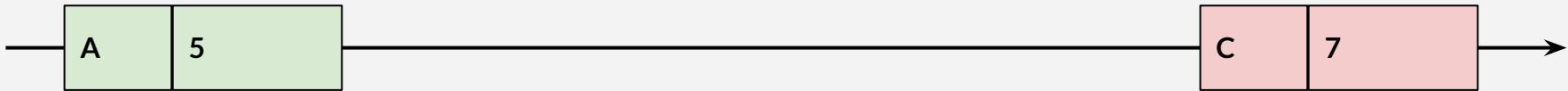


- Takes one record, produces one or zero record
- Produce record that match condition
- Does not change key or value
- Not trigger repartition
- Intermediate operation
- KStream & Ktable

# 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

# flatMap

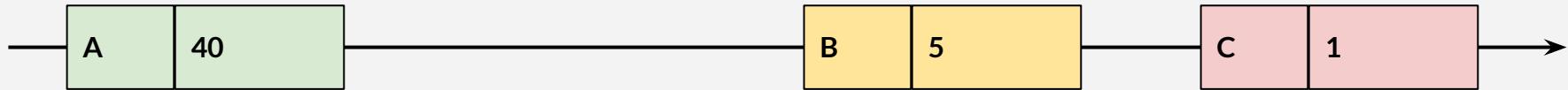


```
stream.flatMap(listPrimeFactorsAndAppendKey())
```

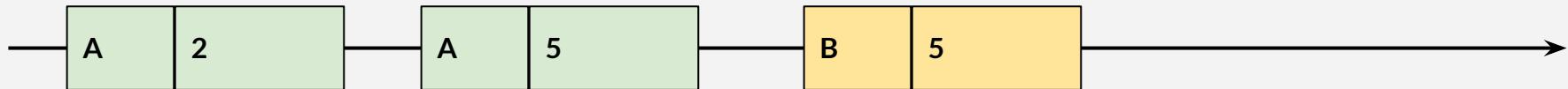


- Takes one record, produces zero or more record
- Change key
- Change value
- Trigger repartition
- Intermediate operation
- KStream

# flatMapValues

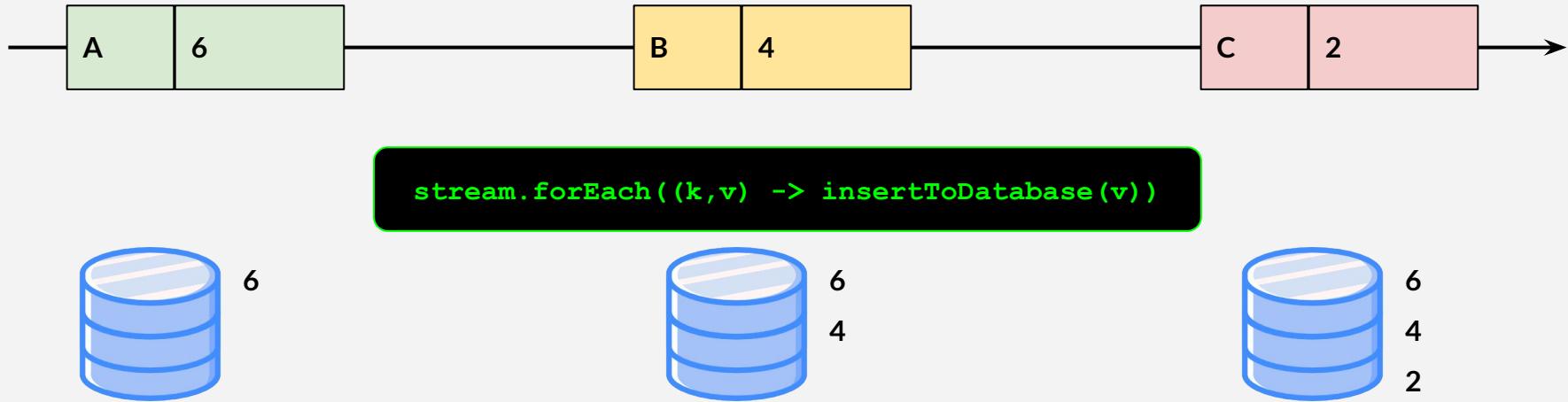


```
stream.flatMapValues(listPrimeFactors())
```



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

# foreach

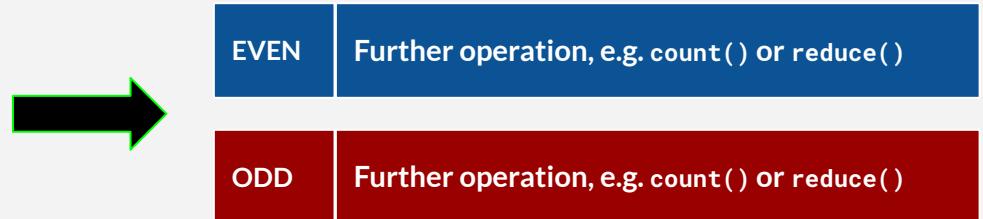


- Terminal operation
- Takes one record, produces none
- Produces side effect
- Side effect not tracked by kafka
- KStream & KTable

# groupBy

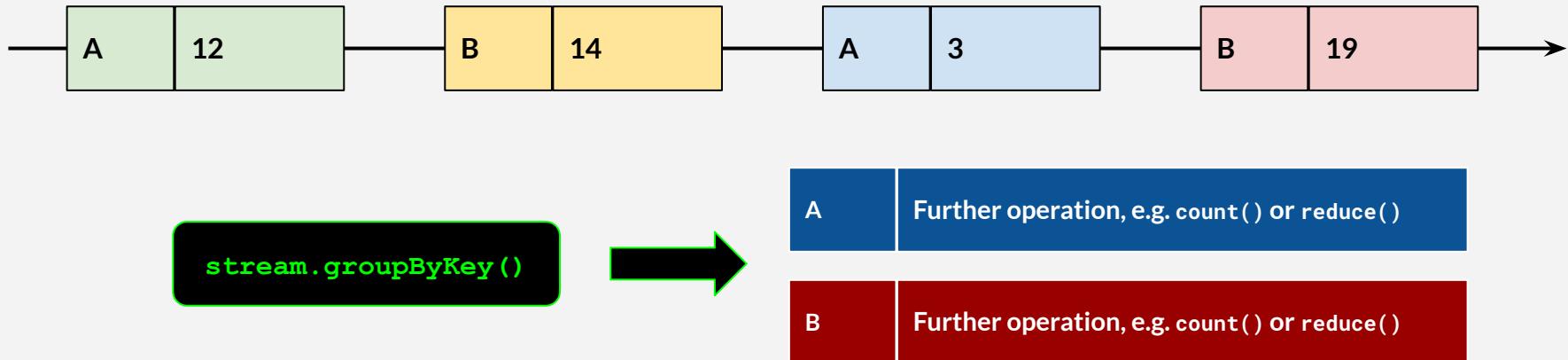


```
stream.groupBy((k, v) -> v % 2 == 0?  
    "EVEN" : "ODD")
```



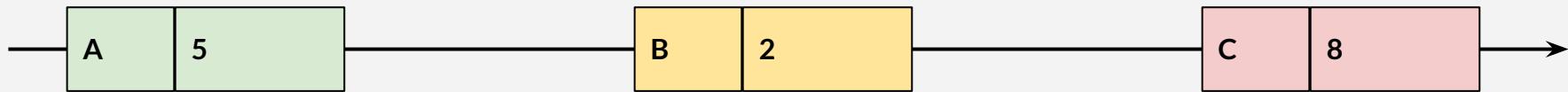
- Intermediate operation
- Group records by new key
- Process further
- KStream & KTable

# groupByKey

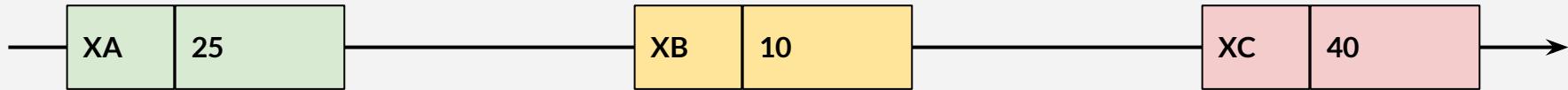


- Intermediate operation
- Group records by existing key
- KStream

# map

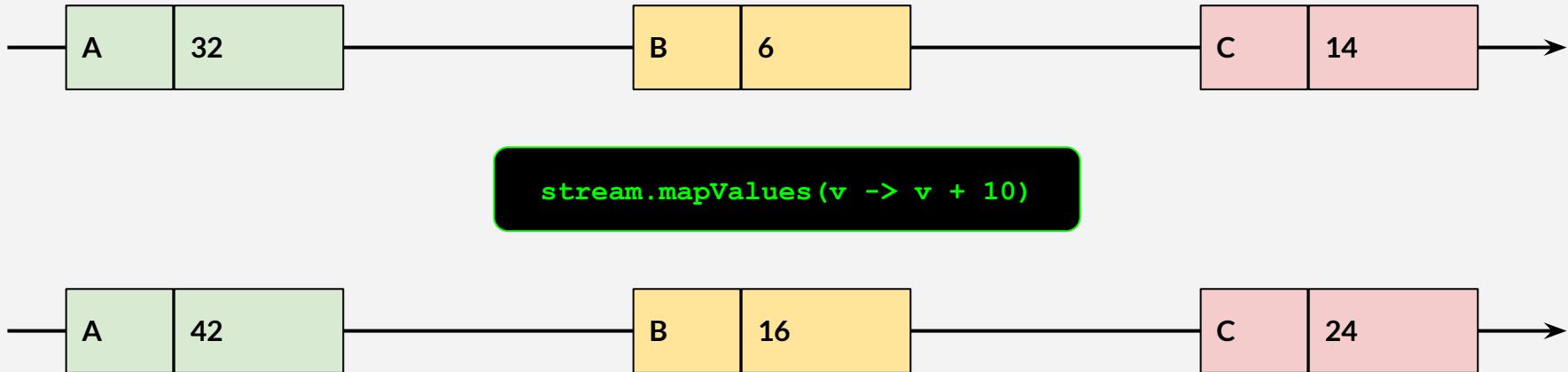


```
stream.map( (k, v) -> KeyValue.pair("X" + k, v * 5) )
```



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

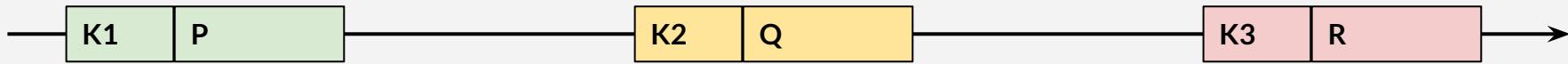
# mapValues



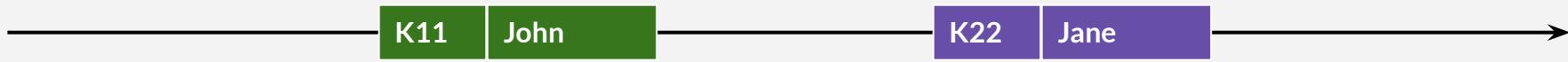
- Takes one record, produces one record
- Does not change key
- Affect only value
- Not trigger repartition
- Intermediate operation
- KStream & Ktable

# merge

*alphabetStream*



*nameStream*

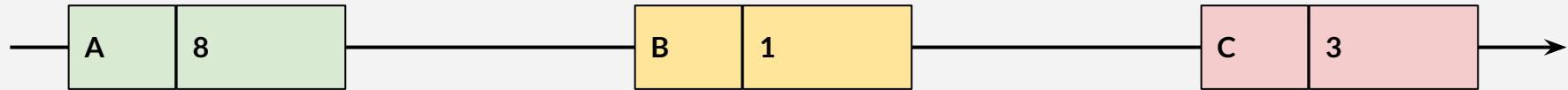


```
var mergedStream = alphabetStream.merge(nameStream)
```



- Merge two streams into one new stream
- Both stream data types must be the same
- No ordering guarantee on resulting stream
- Intermediate operation
- KStream

# peek



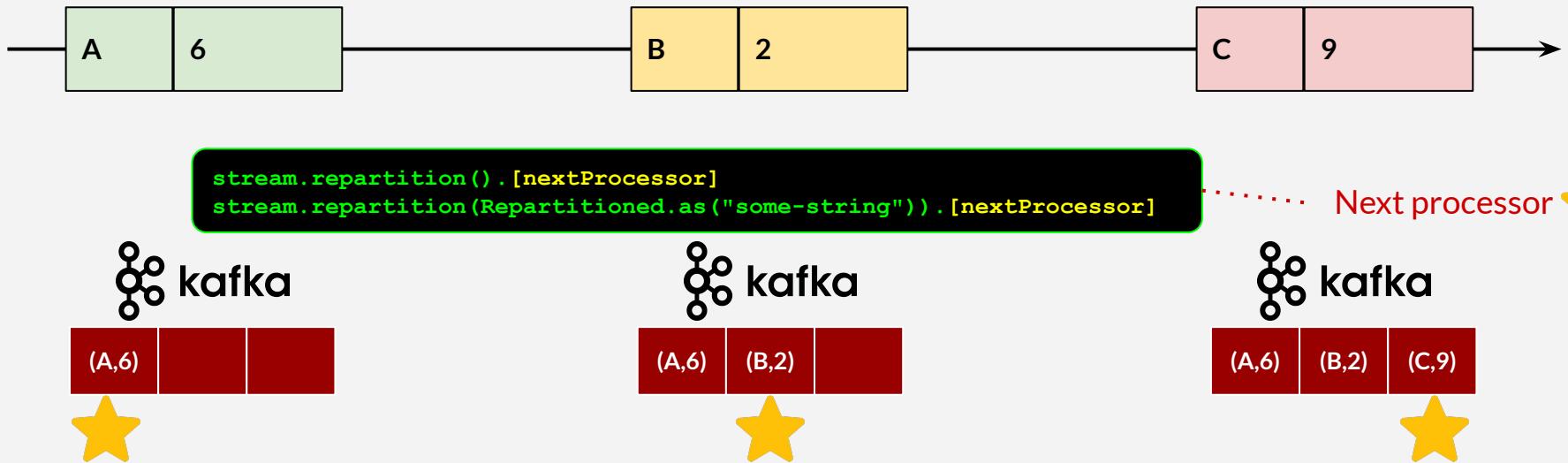
```
stream.peek( (k, v) -> insertToDatabase(v) ) .[nextProcessor]
```

..... Next processor



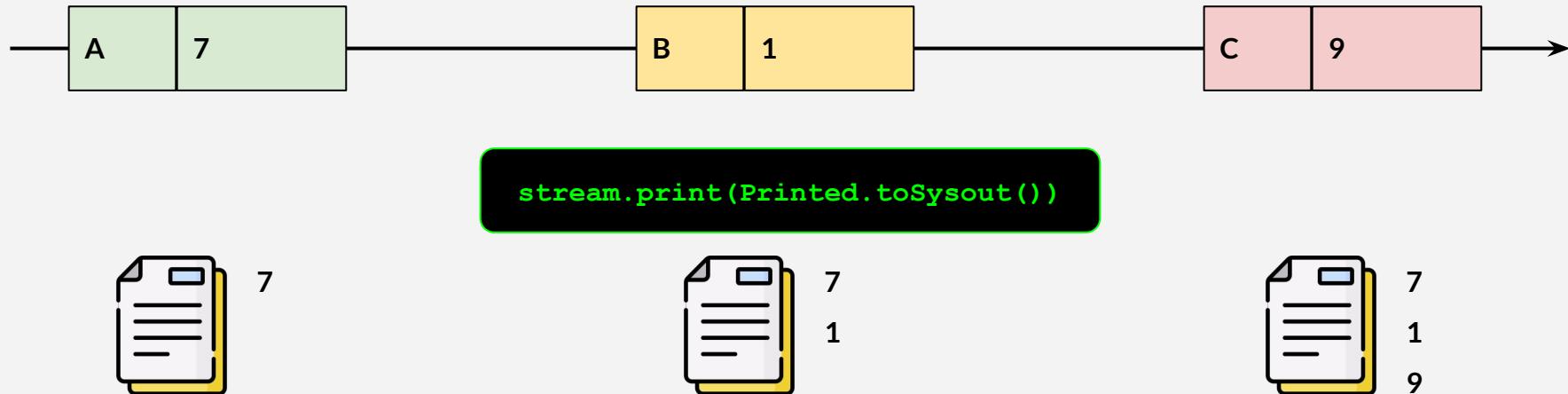
- Produces unchanged stream
- Produces side effect
- Side effect not tracked by kafka
- Result stream can be processed further
- Intermediate operation
- KStream

# repartition



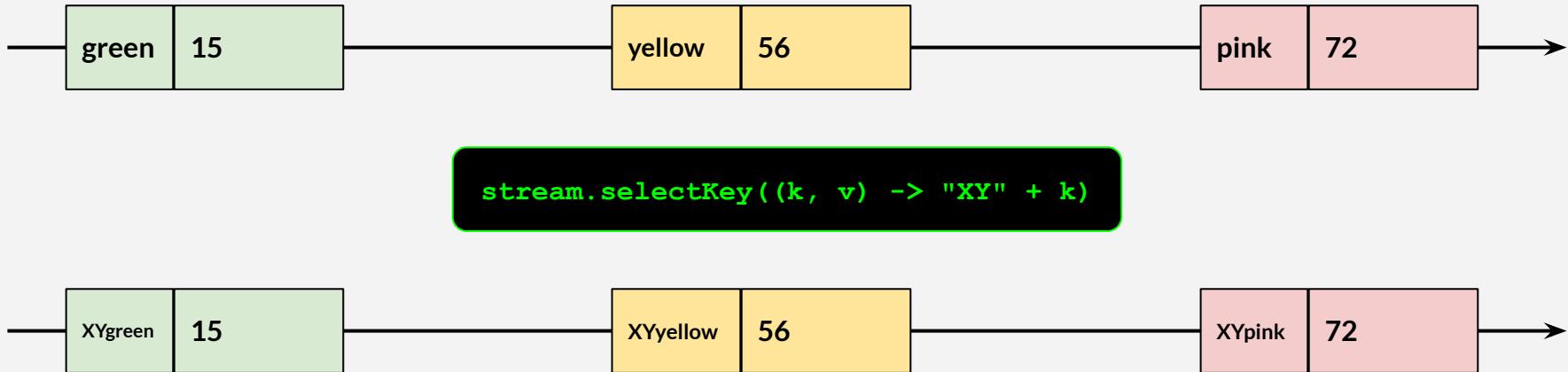
- Intermediate operation
- Write stream to destination topic
- Continue record processing
- repartition() output-topic name is fixed
- Topic only for kafka internal use
- Use through for more control on topic name (risk on using deprecated)

# print



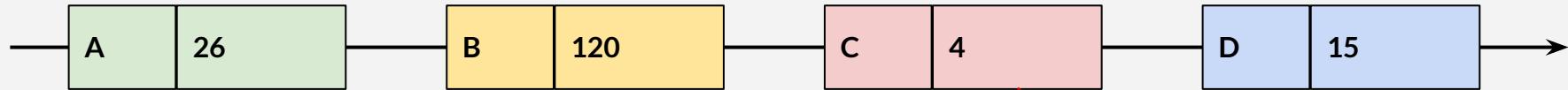
- Terminal operation
- Print each record
- Something like kafka console consumer
- Print to file or console
- KStream

# selectKey



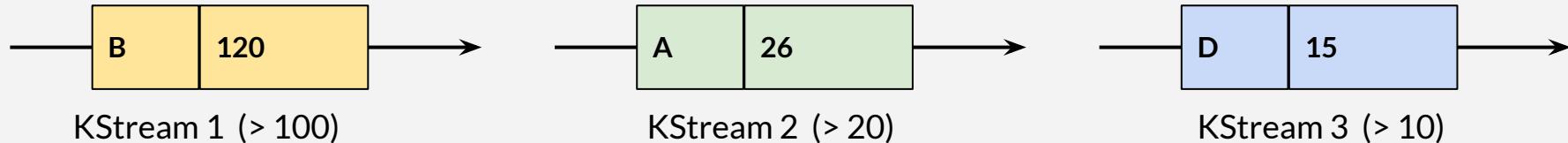
- Takes one record, produces one record
- Set / replace record key
- Possible to change key data type
- Trigger repartitioning
- Value not change
- Intermediate operation
- KStream

# split



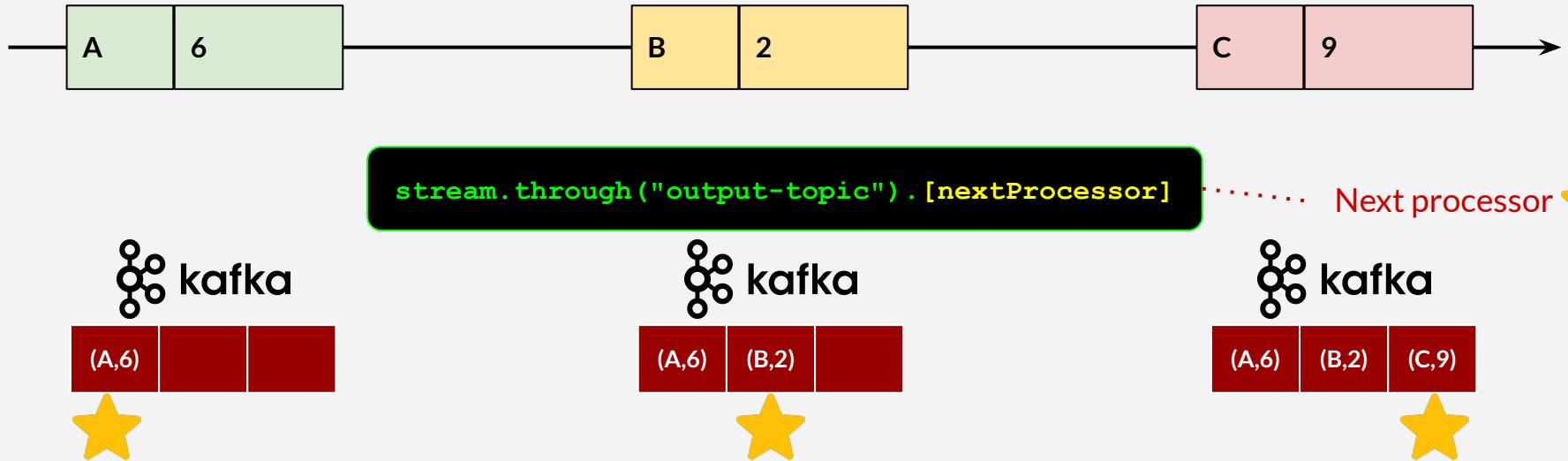
```
stream.split().  
    branch((k, v) -> v > 100, Branched.withConsumer(ks -> ks.to("t-x"))  
    .branch((k, v) -> v > 20, Branched.withConsumer(ks -> ks.to("t-y"))  
    .branch((k, v) -> v > 10, Branched.withConsumer(ks -> ks.to("t-z"))  
)
```

Dropped, no match



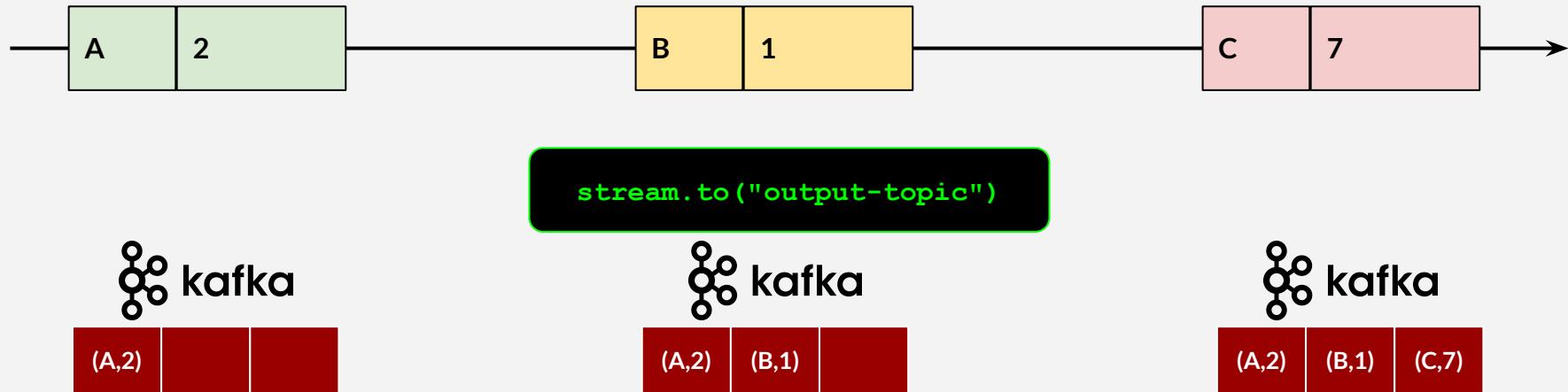
- Split stream based on predicates
- Evaluate predicate in order
- Record only placed once on first match, drop unmatched record
- Get KStream for each branch
- `split()` returns final BranchedKStream
- Each branch returns KStream to be processed further

# through



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

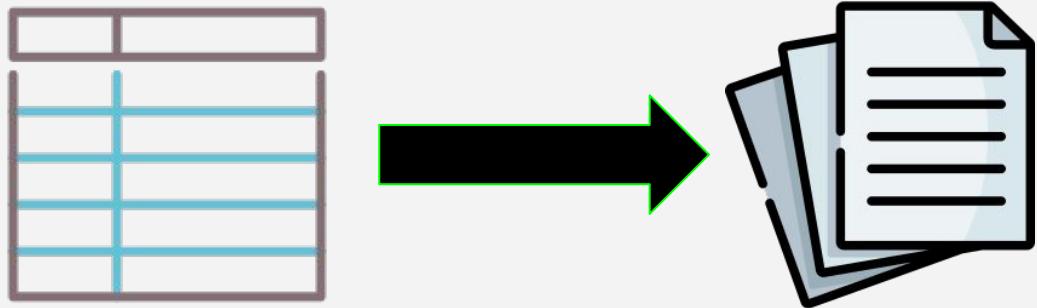
# to



- Terminal operation
- Write stream to destination topic
- KStream

# toStream

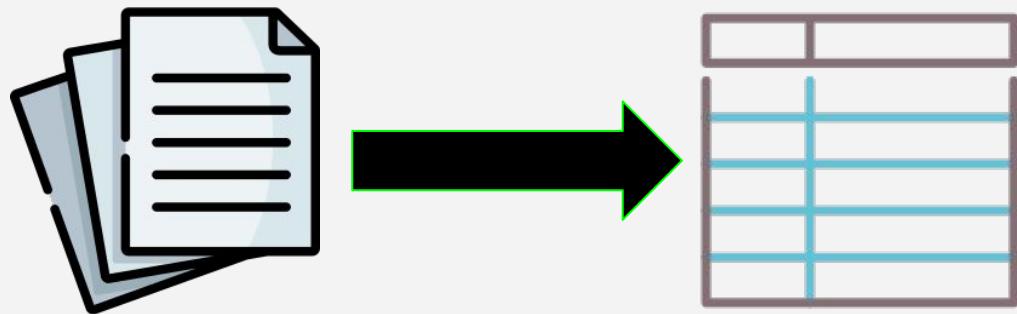
```
table.toStream()
```



- KTable
- Intermediate operation
- Convert KTable to KStream

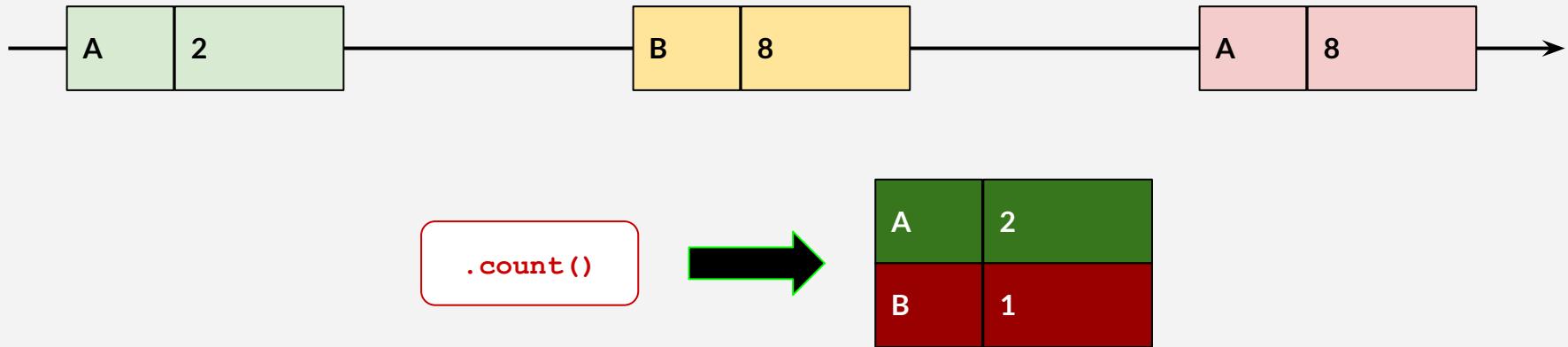
# toTable

```
stream.toTable()
```



- KStream
- Intermediate operation
- Convert KStream to KTable

# count



- Counts number of record based on key
- Ignore null keys or values
- KGroupedStream

# aggregate

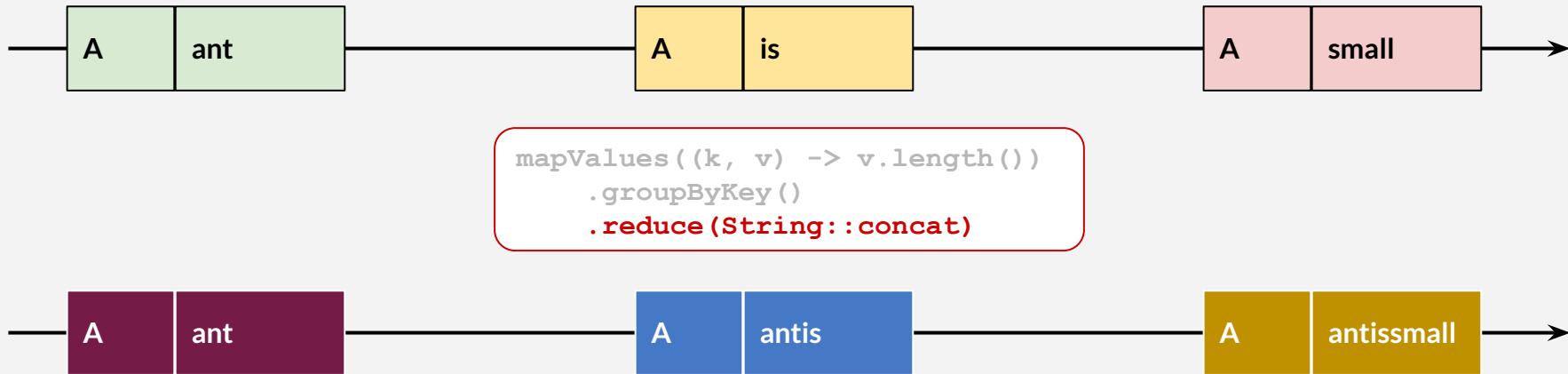


```
mapValues((k, v) -> v.length())
    .groupByKey()
    .aggregate(() -> 0, (aggKey, newValue, aggValue) -> aggValue + newValue)
```



- Aggregate record based on key
- Need initializer and adder
  - Initializer on example : () -> 0
  - Adder on example : (aggKey, newValue, aggValue) -> aggValue + newValue
- Aggregation result can be different type with input
- Ignore null keys
- On first each non-null key : call initializer, then call adder
- On non-null value : call adder
- KGroupedStream

# reduce



- Specialized form / shorter syntax for aggregate
- Result and input type not change
- Need reducer
  - In example: `String.concat()`
- KGroupedStream

# Available Operations Sample Code



# Example Code

- × Example source code for available operations on  
*Resources & References*
- × For simple operations (more real use case later)
  - × *broker.stream.operation.OperationDemoStream*
- × How to use the demo



# Example Code

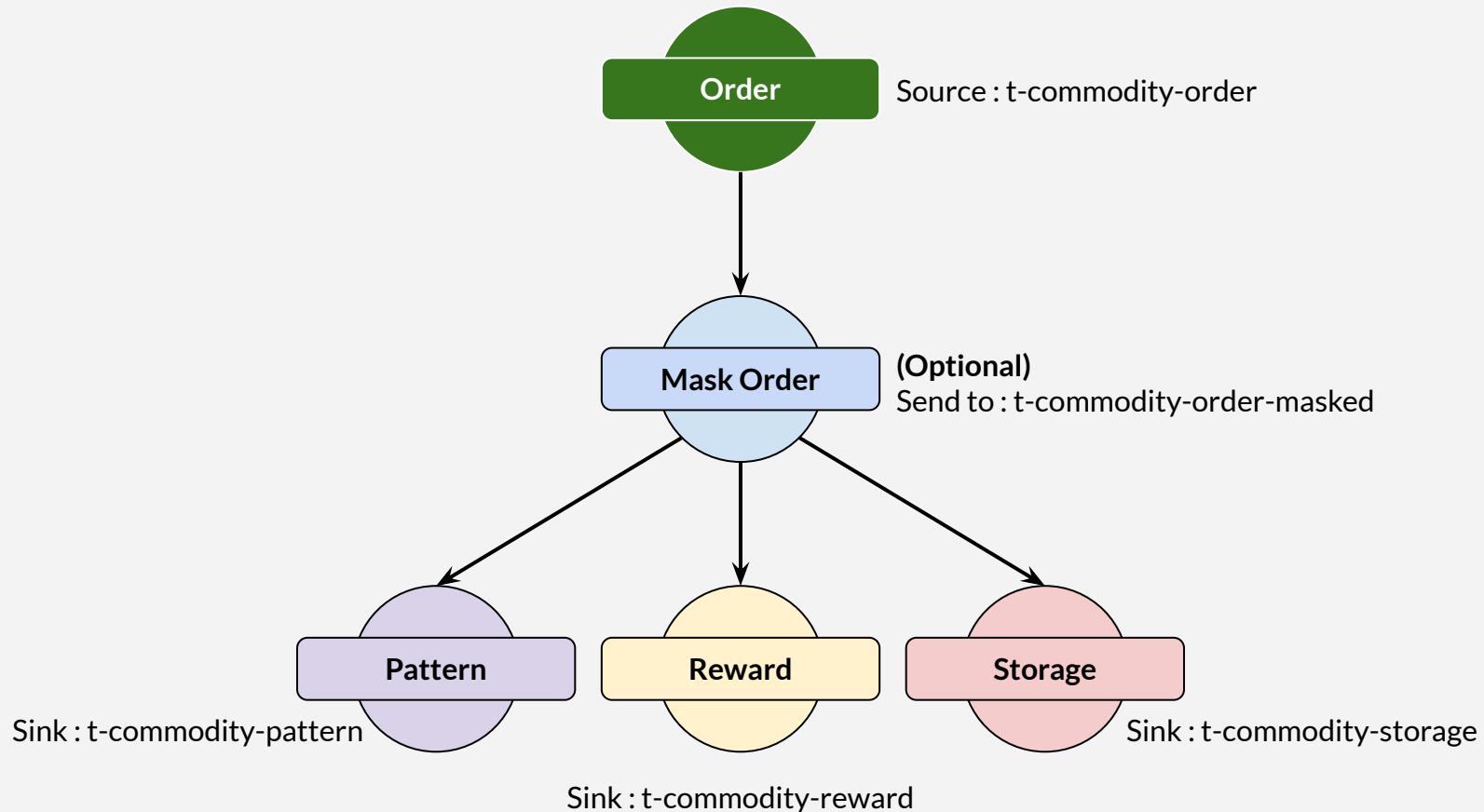
- × Examples for **KStream**
- × Concept is the same for **KTable**
- × **KTable** operations usually involve join
  - × More about this later



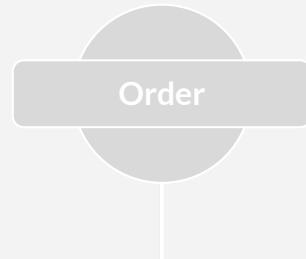
# Commodity Stream Topology



# Topology



# Topology



Key : A123XY  
Value : { JSON }

```
$> kafka-console-consumer.sh ... --property print.key=true
```

**A123XY** { JSON }

Key : A123XY  
Value : { JSON }



Key : A123XY  
Value : { JSON }

Key : A123XY | Value : { JSON }

# Commodity Stream

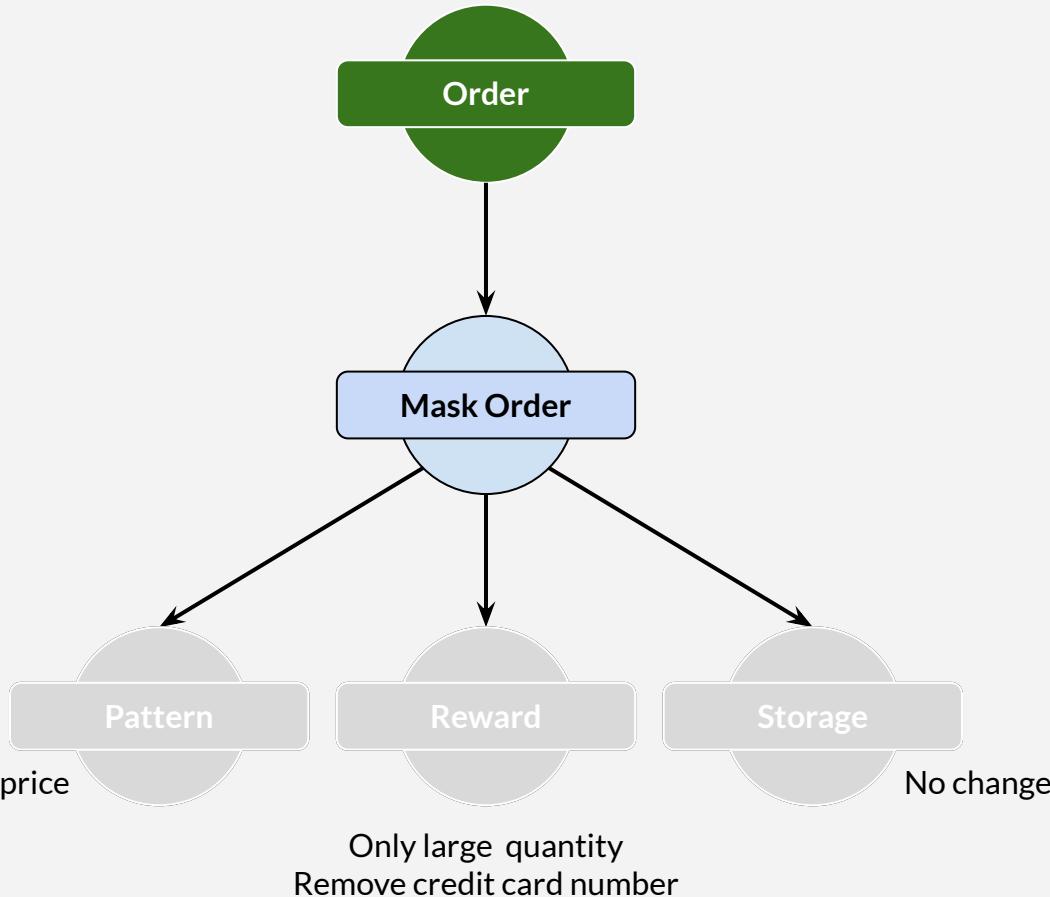
## First Step



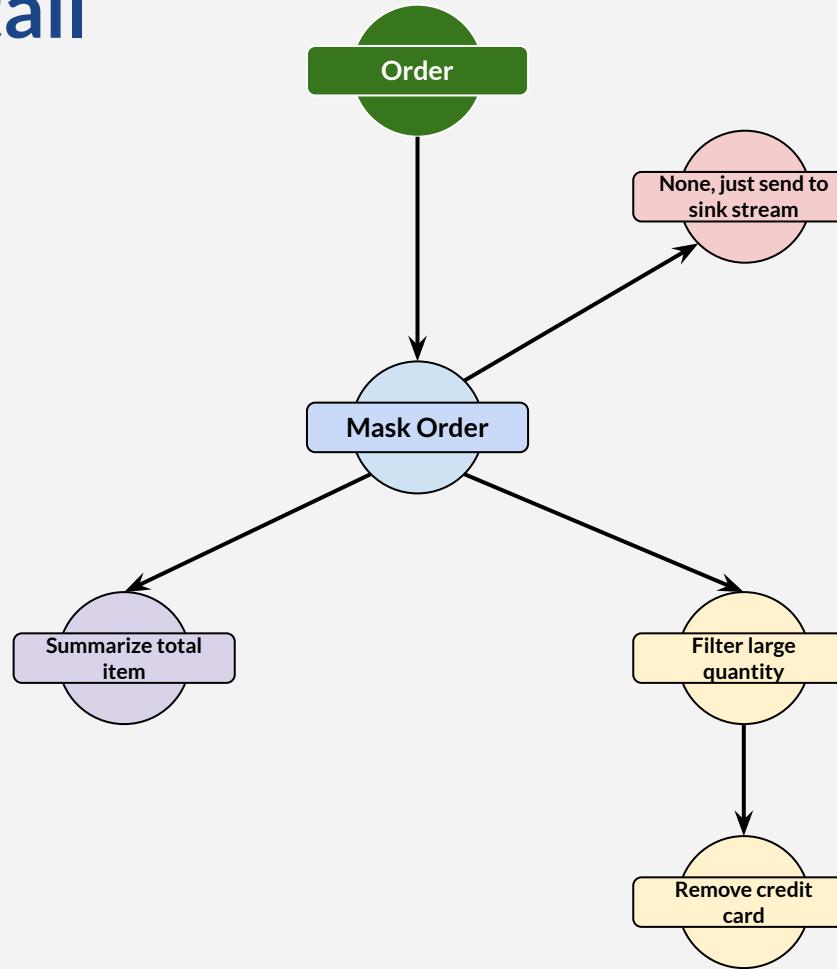
# Commodity Stream Sink Processors



# High Level Topology



# Topology Detail

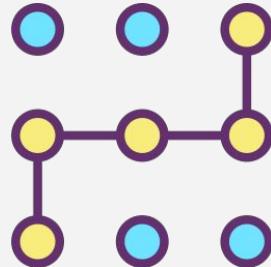


# **Commodity Stream**

## **Additional Requirements**



# 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

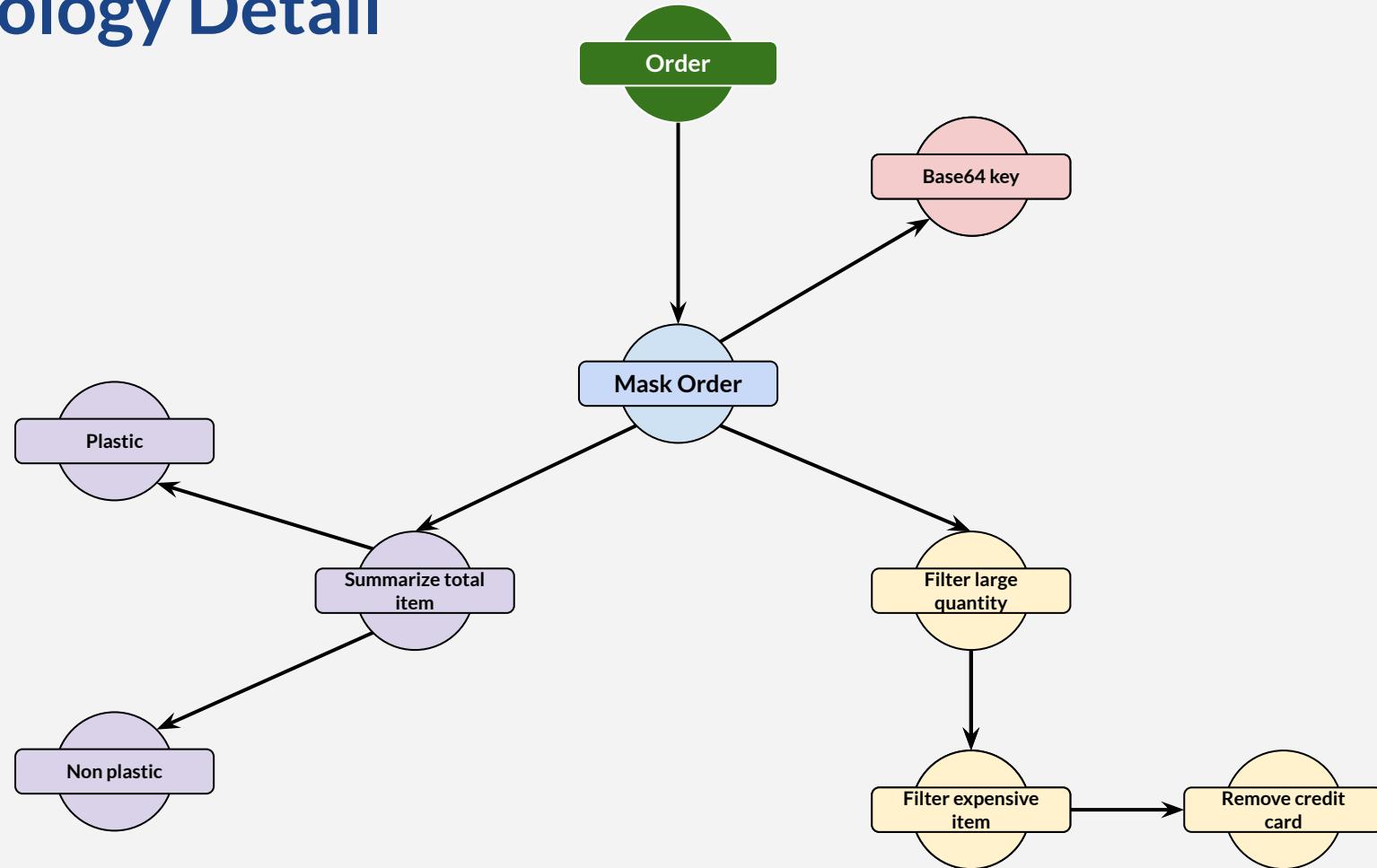


Storage

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

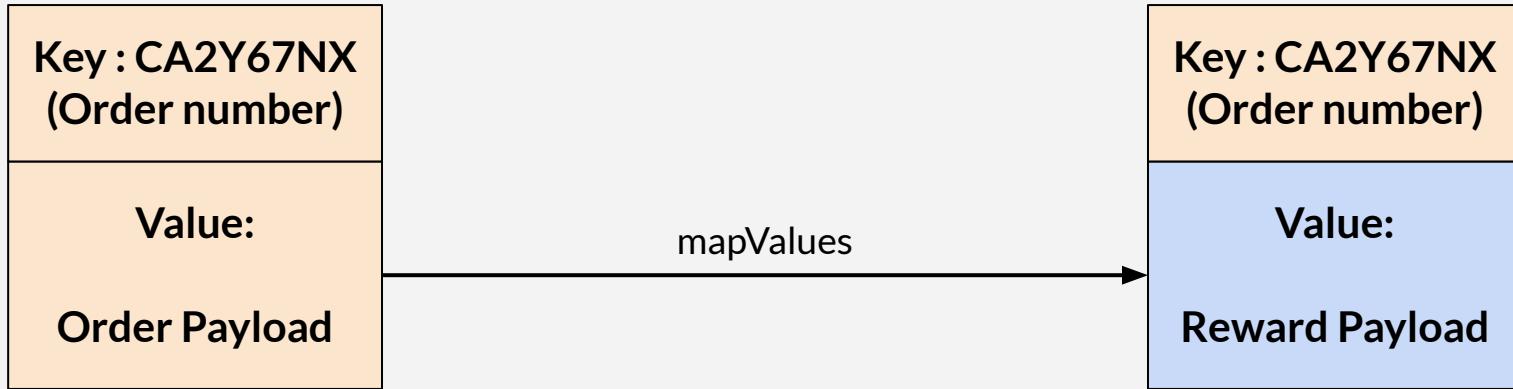
Stream (Kafka Sink Topic)	Data
<b>Pattern - plastic</b>	Plastic Cat
<b>Pattern - not plastic</b>	Cotton Dog, Wooden Horse, Steel Pig
<b>Reward</b>	Plastic Cat, Steel Pig
<b>Storage</b>	Plastic Cat, Cotton Dog, Wooden Horse, Steel Pig

# **Commodity Stream**

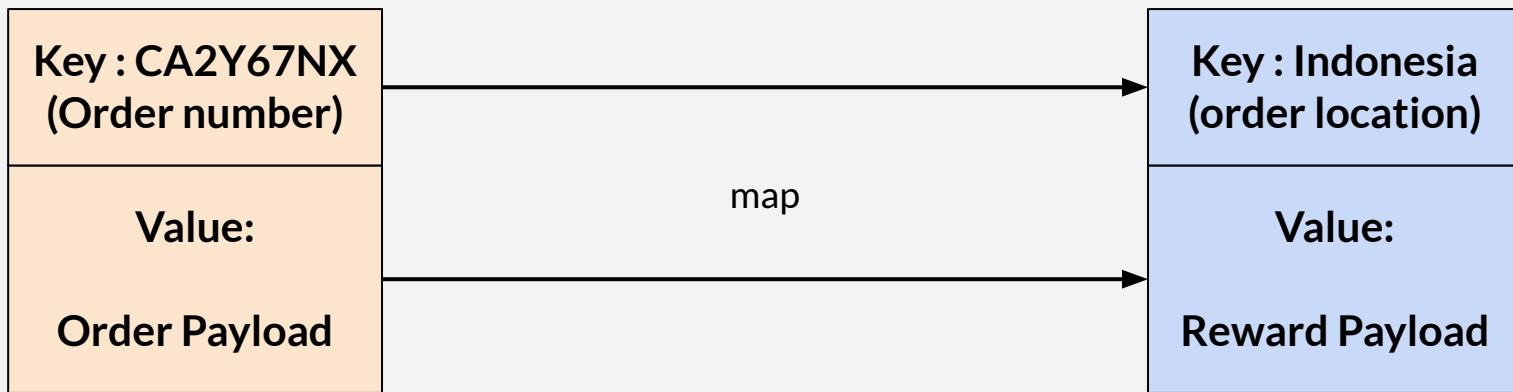
## **Reward Each Location**



# Reward Message



# Reward Message

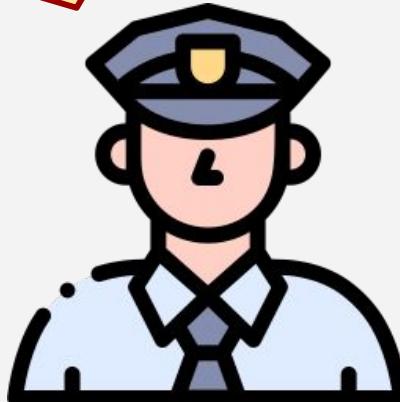


# Commodity Stream

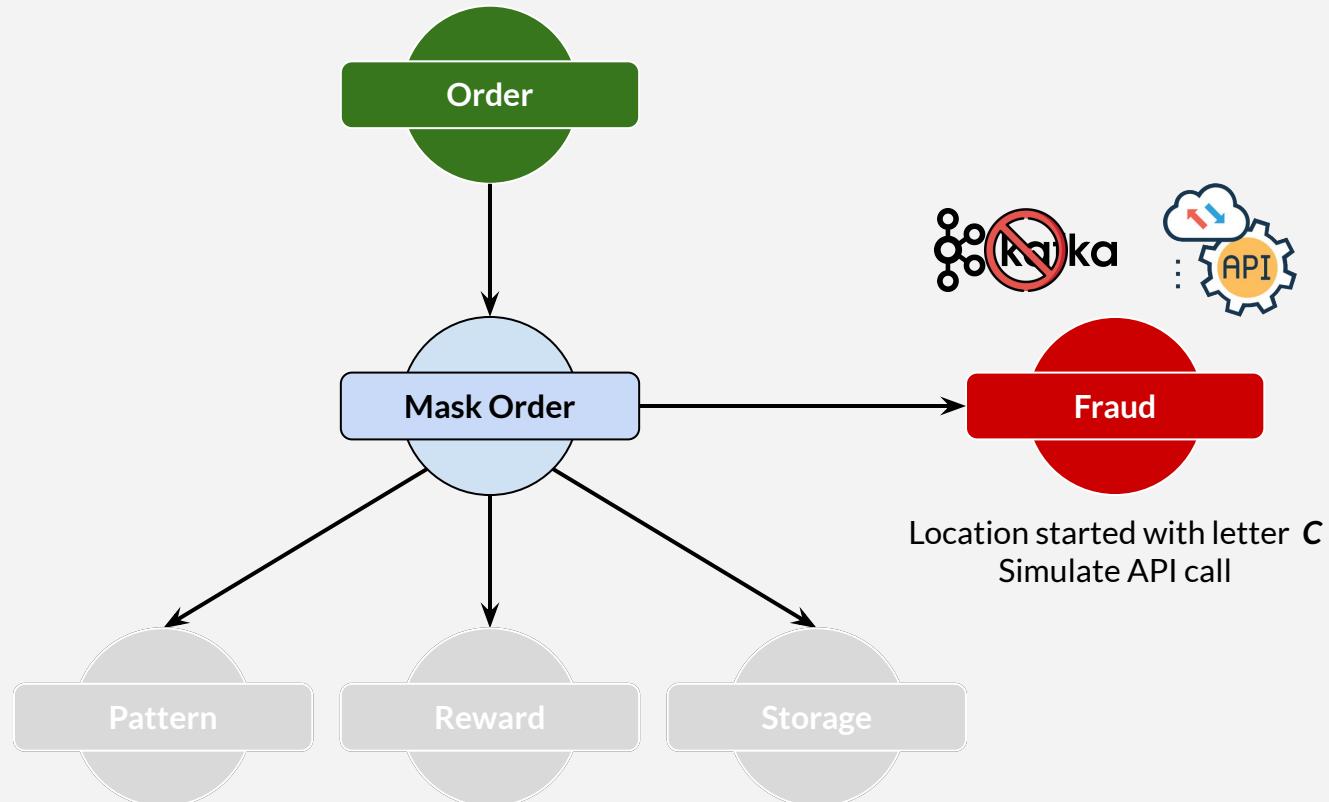
## Calling API or Other Process



# Something Suspicious



# High Level Topology

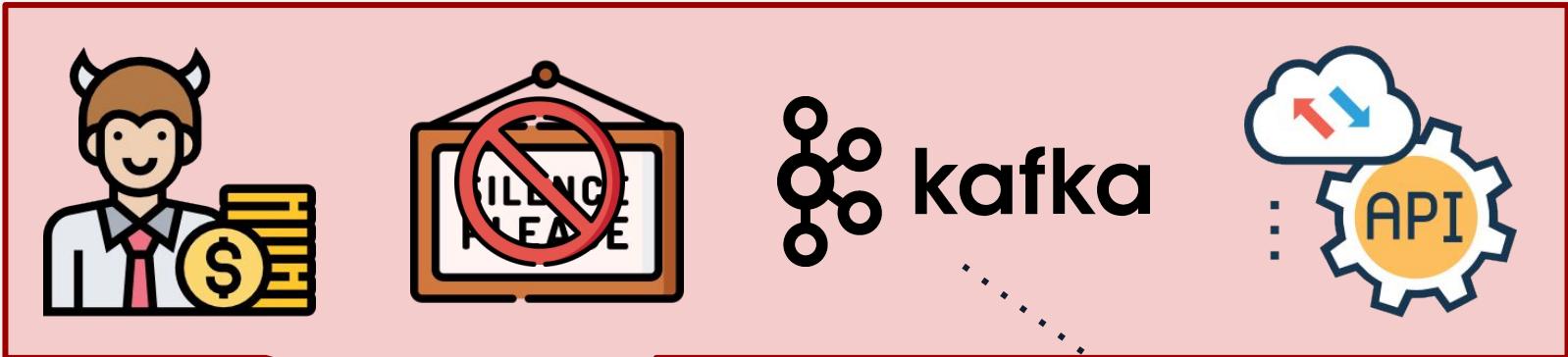


# Commodity Stream

## Further Fraud Processing



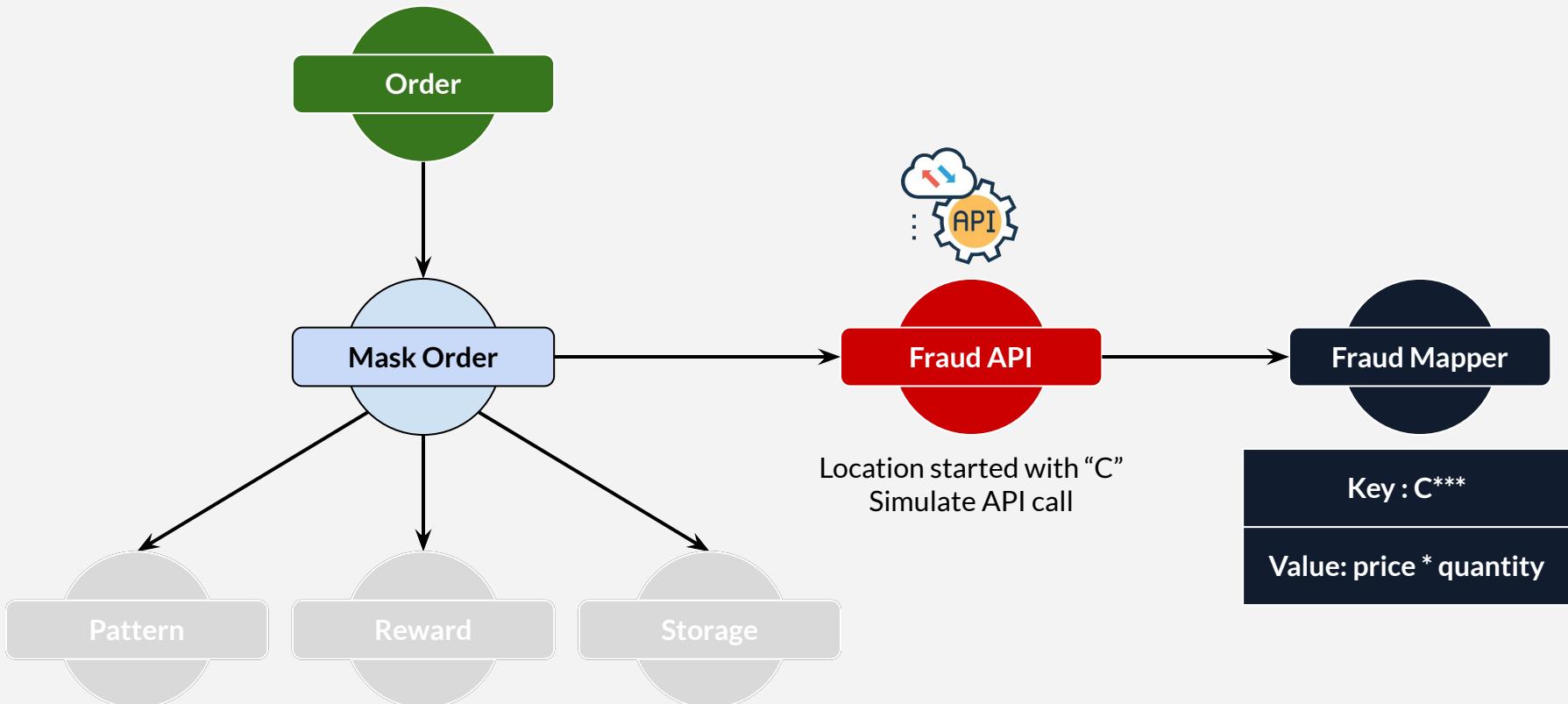
# Something Suspicious



Key : C\*\*\*

Value: price \* quantity

# High Level Topology



# Where Is The AI Assistant?



# AI Assistant

- × Why rarely uses chat feature?
- × Uses autocomplete extensively
- × Functionality is not very complex
  - × Detailed prompt (maybe longer than the code)
  - × Autocomplete is more efficient
- × AI assistant is **non-deterministic**
  - × Different code might cause concept not clear
  - × Different code style can lead to confusion



# AI Assistant

- ✗ Not always the case
- ✗ Example: API performance testing with AI  
**Assistants (see Bonus on last section of the course for discount)**
- ✗ AI assistant is to aid
- ✗ AI assistant is not decision maker

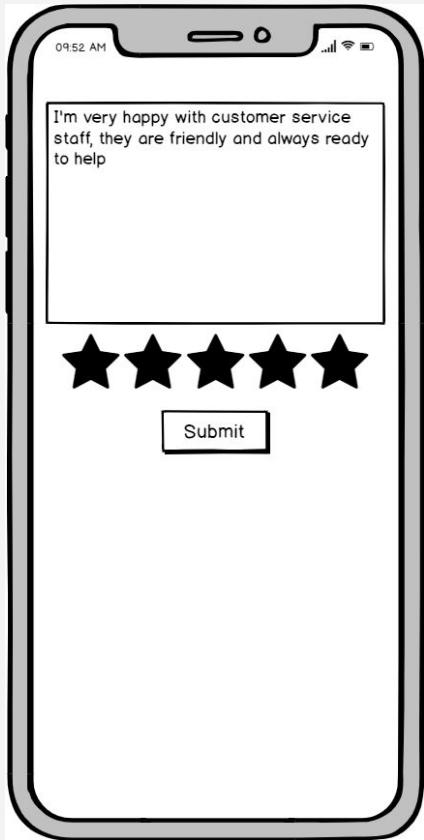


# Feedback Stream

## Are We Good Enough?

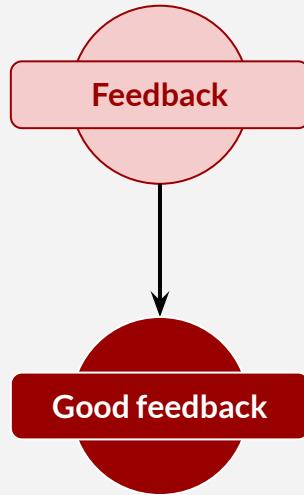


# Customer Feedback



**happy, good, helpful, etc**

# High Level Topology



# Source Code for Feedback

- ✗ **Feedback\*.java**
- ✗ **Package com.course.kafka**
  - ✗ **api.request**
  - ✗ **api.response**
  - ✗ **api.server**
  - ✗ **broker.message**
  - ✗ **broker.producer**
  - ✗ **command.action**
  - ✗ **command.service**

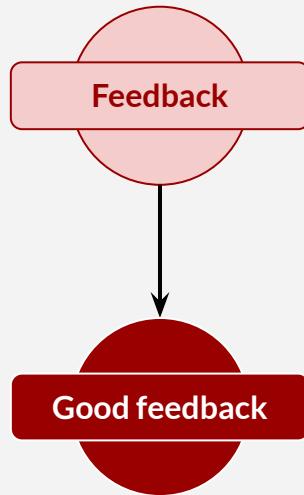


# Feedback Stream

Who Owns This Feedback?



# High Level Topology



Key : [branch location]

Value: [good word]

# Java Stream API

- × Java stream API is not kafka stream
- × Same method names:
  - × **filter**
  - × **flatMap**
  - × **forEach**
  - × **map**
  - × **peek**



# Feedback Stream

## Good Feedback or Bad?

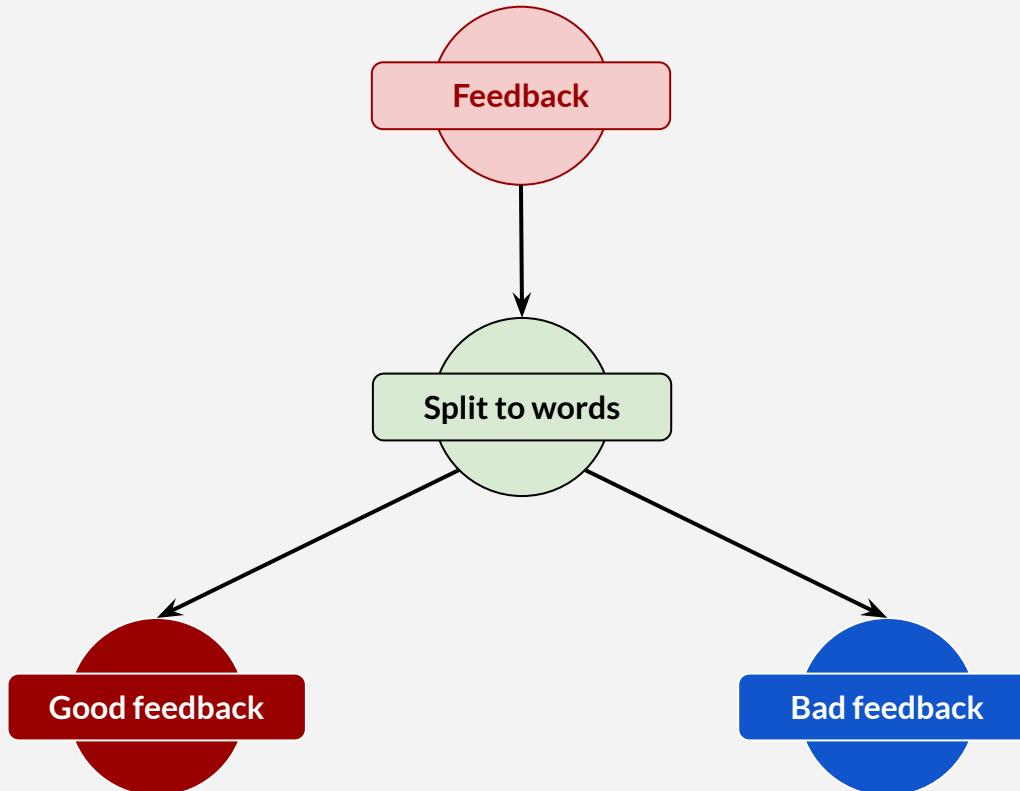


# Bad Feedback?

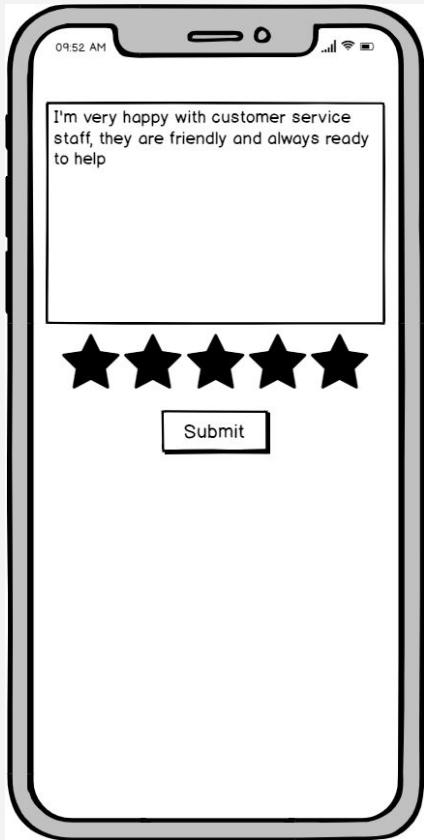
- ✗ Analyze bad feedback
- ✗ Stream to analyze feedback : good or bad?
- ✗ Pause the video & practice
- ✗ Topology
- ✗ Resume the video for answer



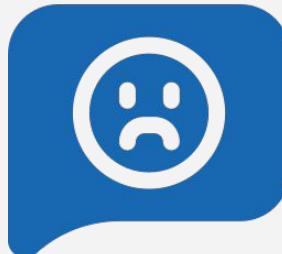
# High Level Topology



# Customer Feedback



**happy, good, helpful, etc**



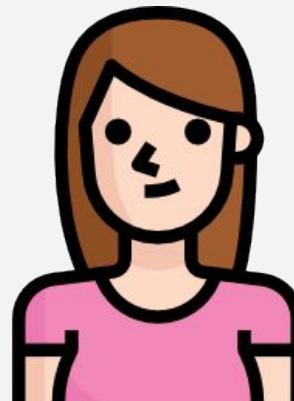
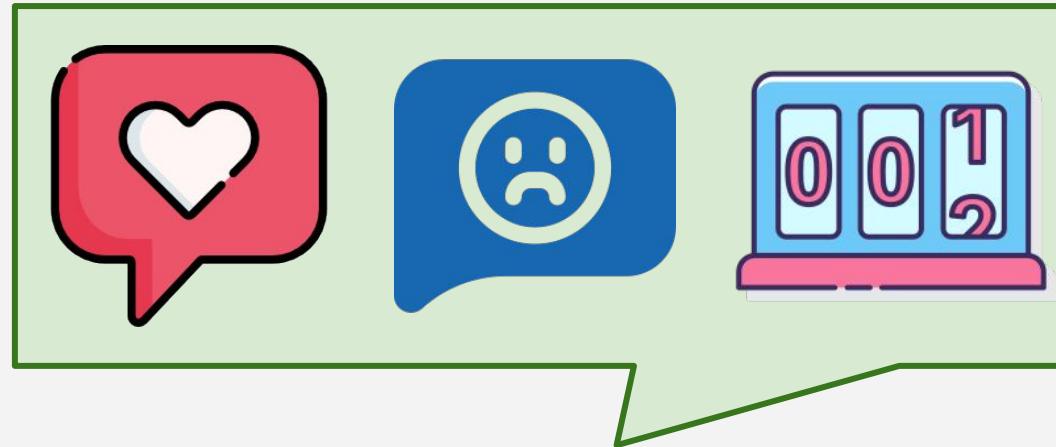
**angry, sad, bad, etc**

# Feedback Stream

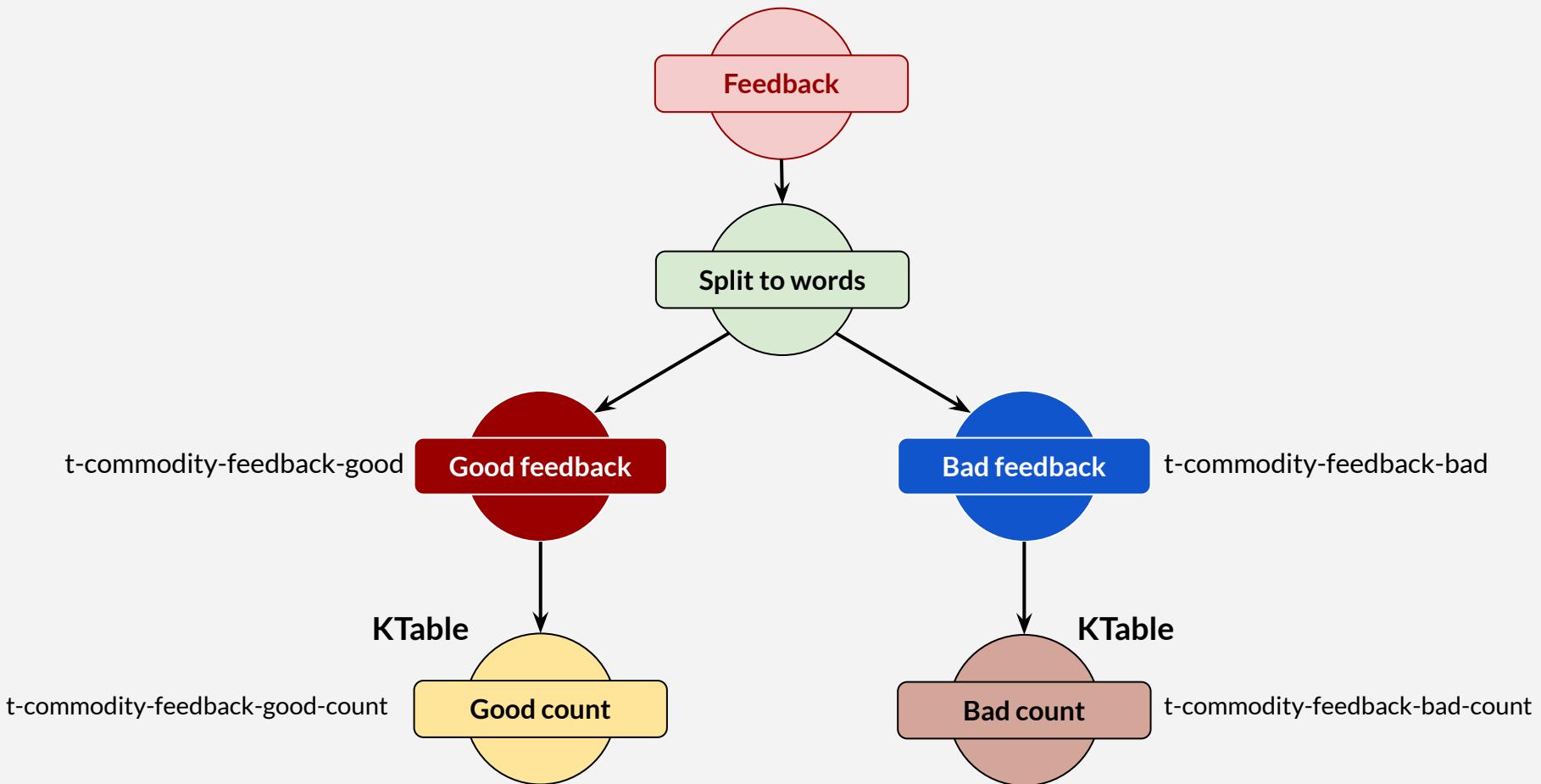
## Group Using Table



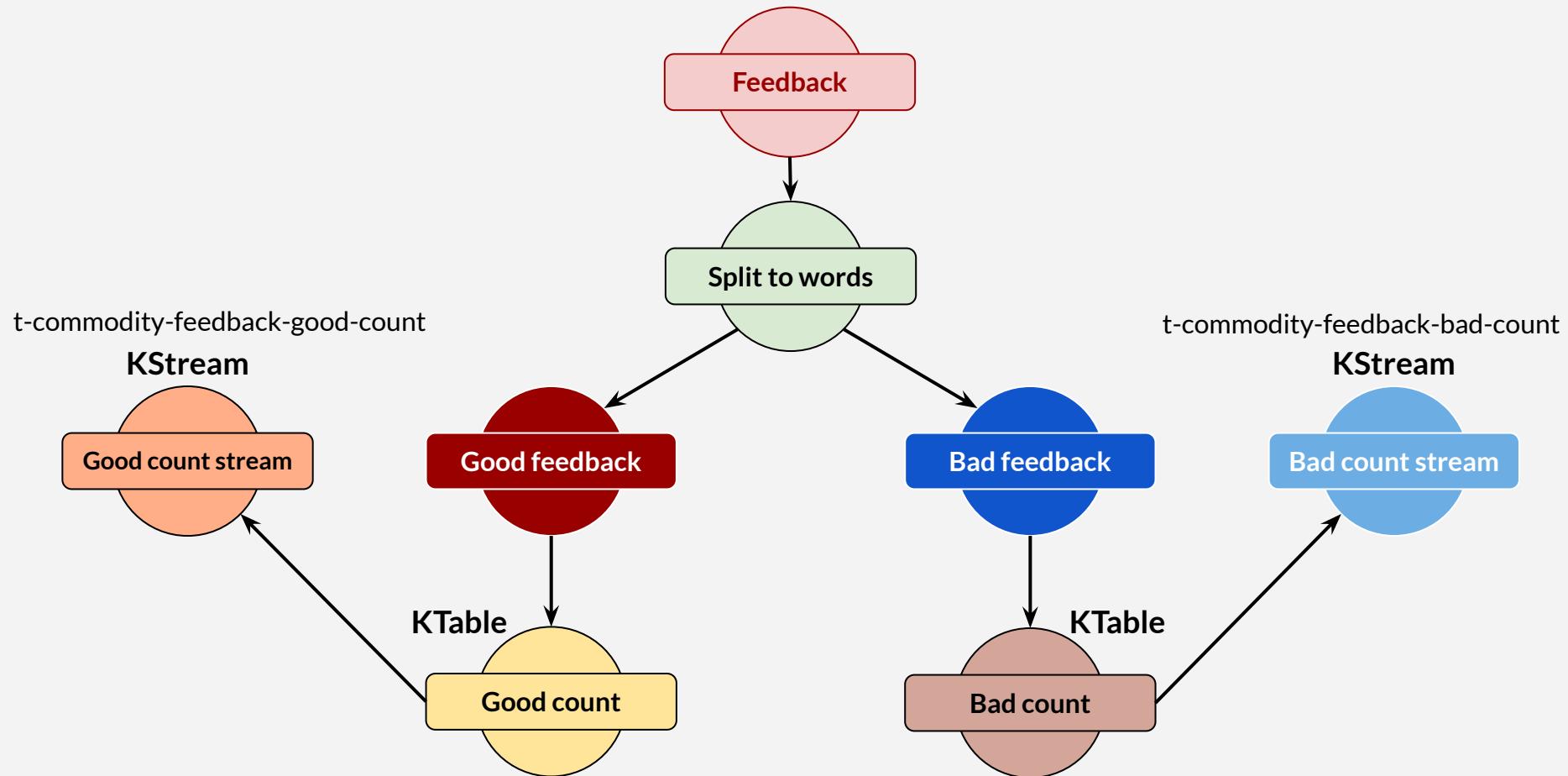
# Count The Word



# High Level Topology



# High Level Topology



# Feedback Stream

## Delay on Table



# Kafka Stream Configuration

- × Default configuration : 30 seconds
- × Cache and send
- × Kafka property : `commit.interval.ms`
- × Adjust configuration if needed



# Feedback Stream

## Send and Continue



# Send and Continue

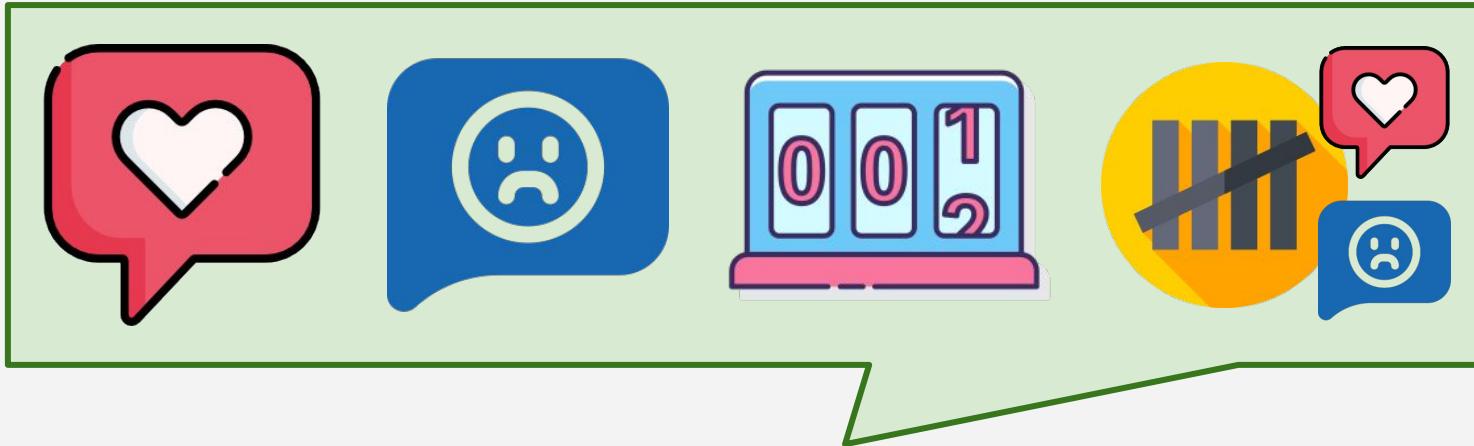
```
streams.to("first-output-topic");
streams.groupByKey().....to("second-output-topic");
```

# Feedback Stream

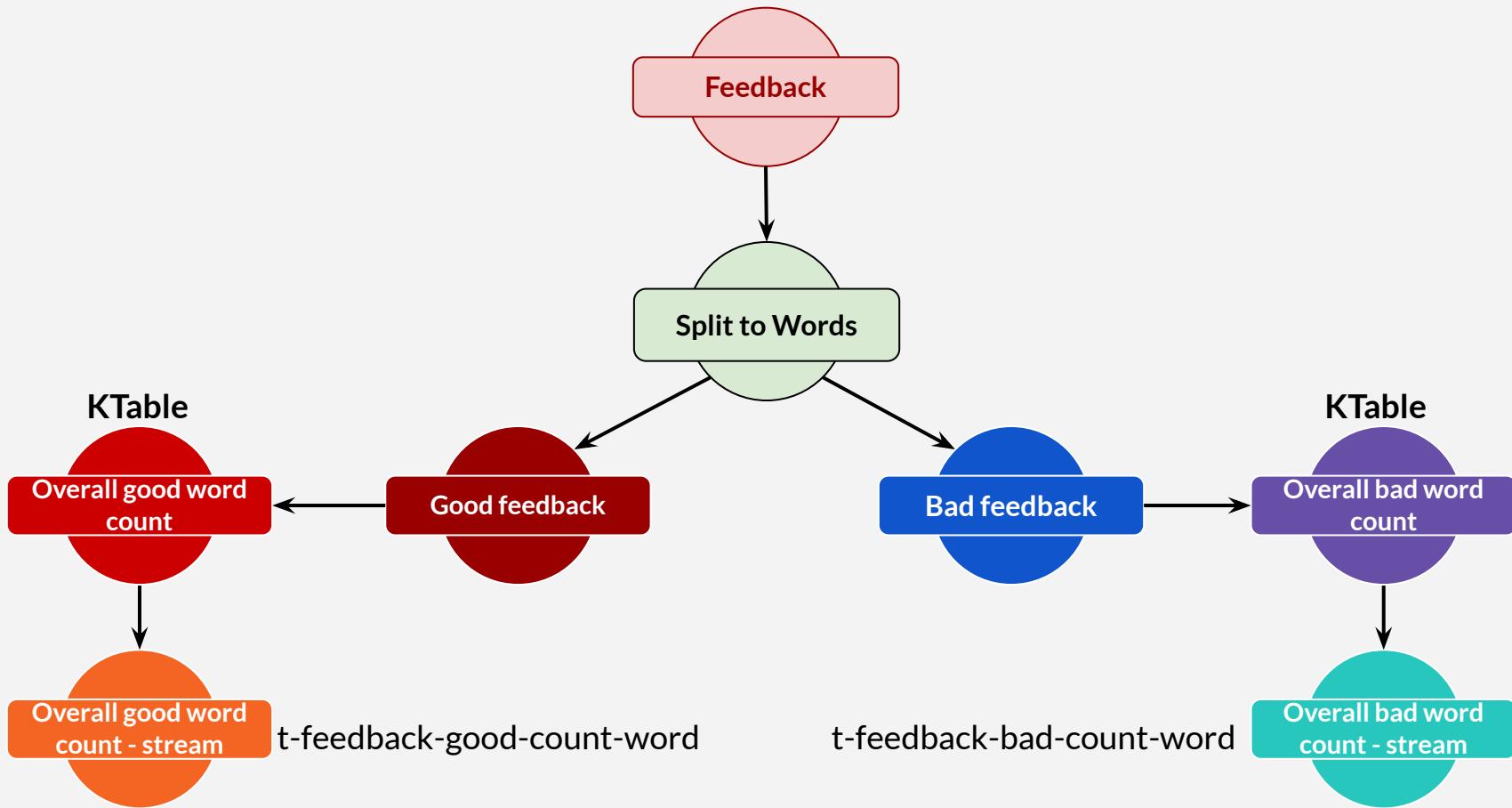
## Overall Good (or Bad)



# Count The Word



# High Level Topology

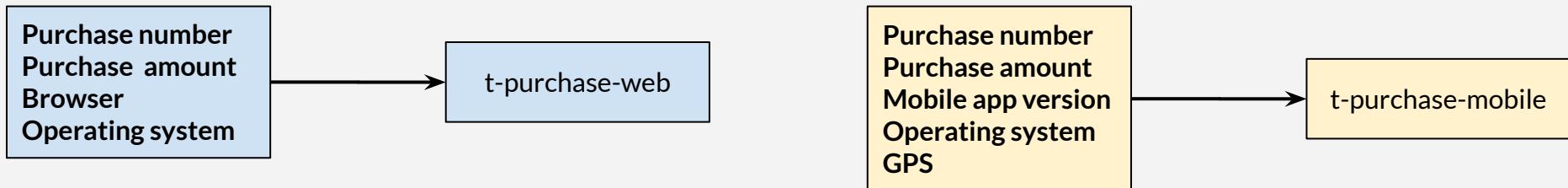


# **Customer Stream**

## **Web & Mobile**



# Different Device, Different Data



t-purchase-all

Purchase Web 1

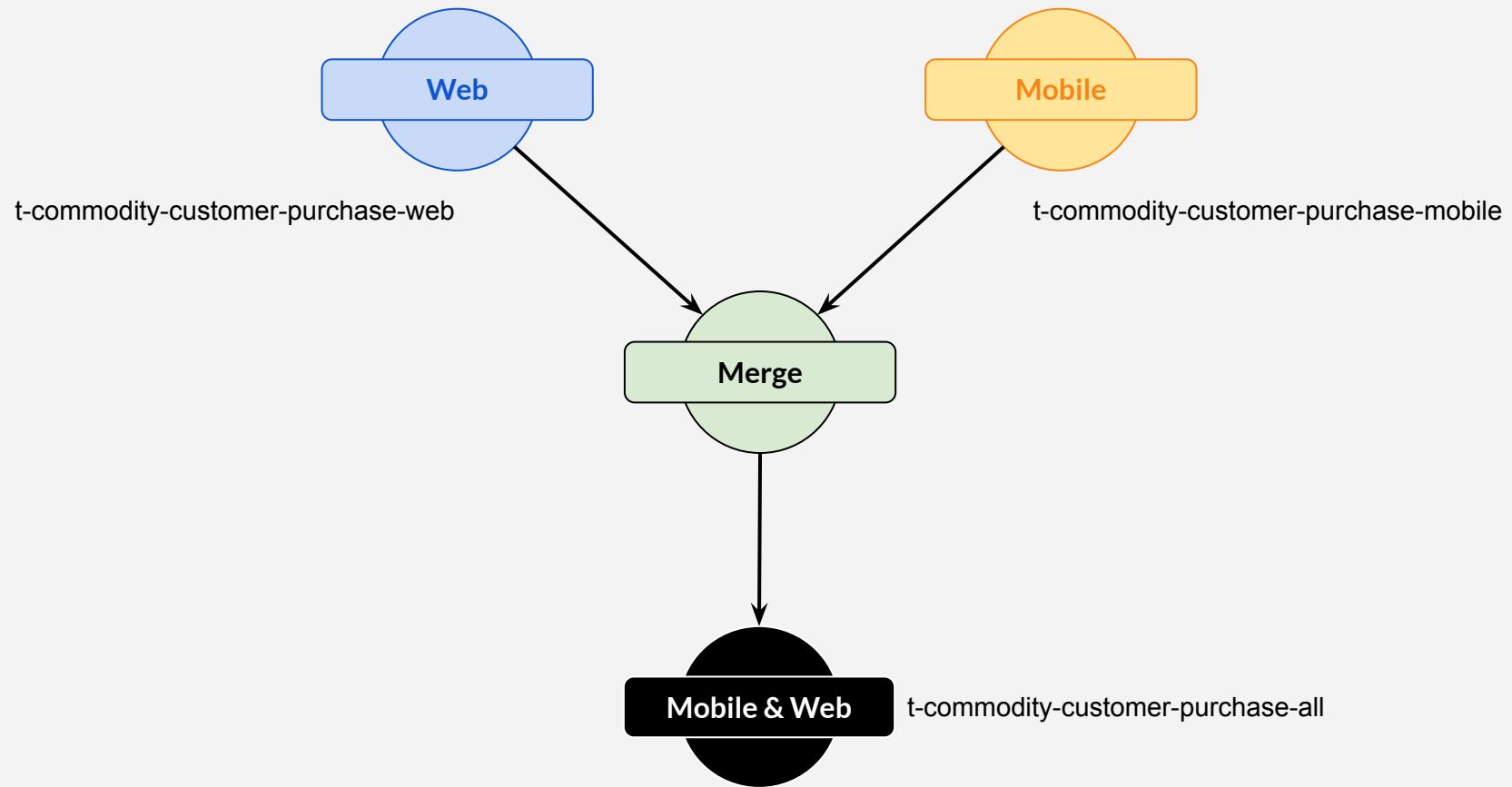
Purchase Mobile  
1

Purchase Mobile  
2

Purchase Web 2

Purchase Mobile  
3

# High Level Topology



# Source Code for Customer Purchase

- ✗ `CustomerPurchase*.java`
- ✗ Package `com.course.kafka`
  - ✗ `api.request`
  - ✗ `api.server`
  - ✗ `broker.message`
  - ✗ `broker.producer`
  - ✗ `command.action`
  - ✗ `command.service`

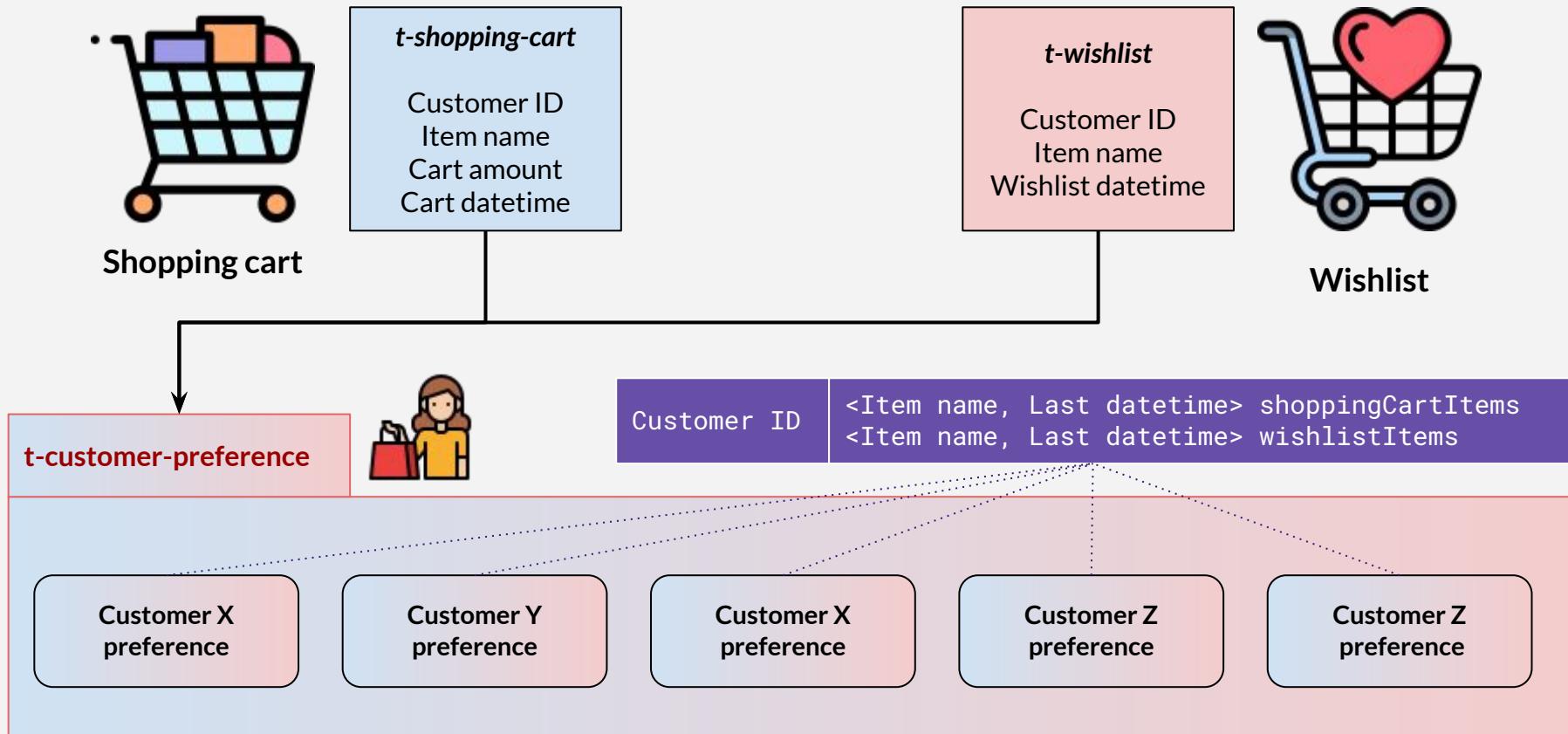


# **Customer Stream**

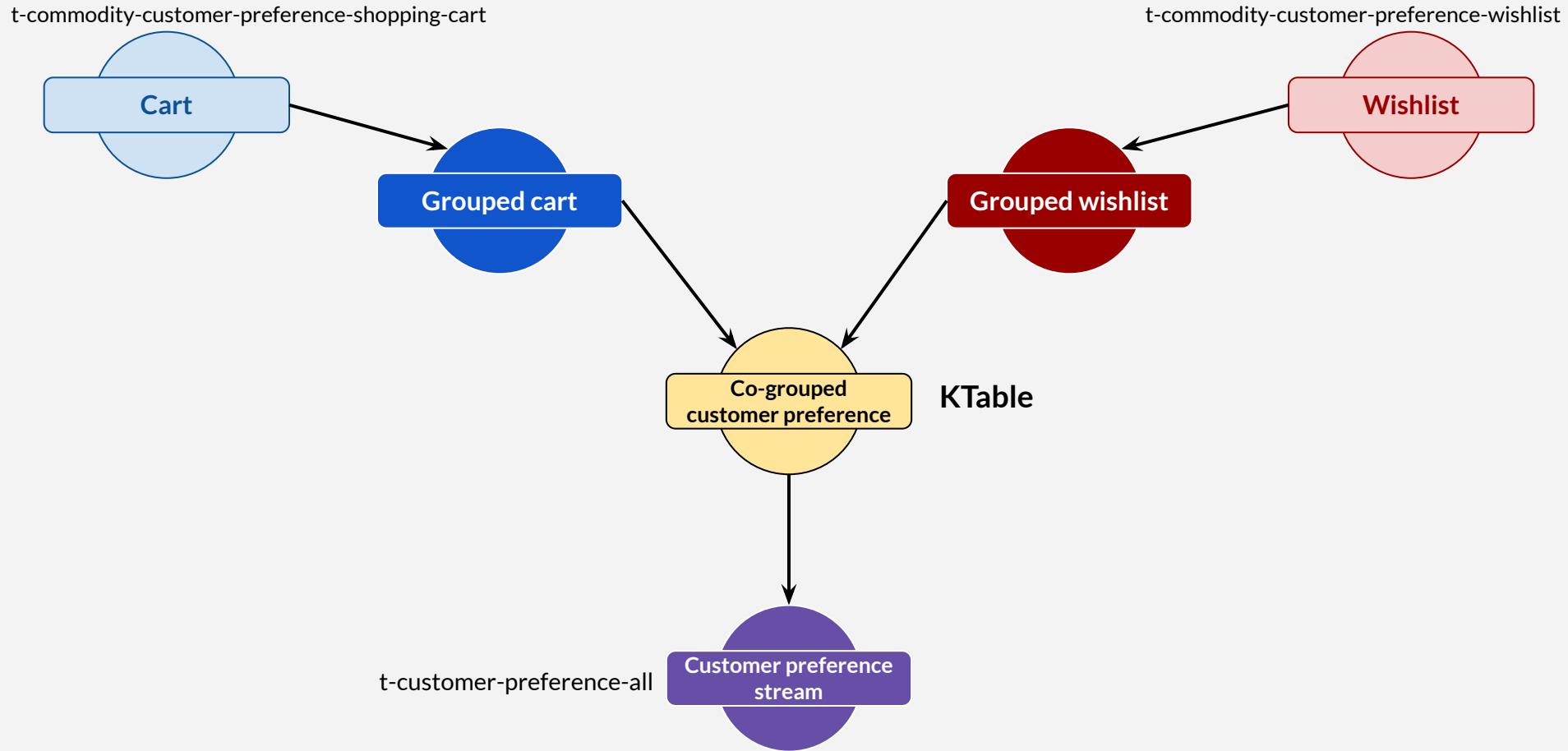
## **Cart & Wishlist**



# Two Things, One Customer Preference



# High Level Topology



# Source Code for Customer Purchase

- ✗ `CustomerPreference*.java`
- ✗ Package `com.course.kafka`
  - ✗ `api.request`
  - ✗ `api.server`
  - ✗ `broker.message`
  - ✗ `broker.producer`
  - ✗ `command.action`
  - ✗ `command.service`



## data

---

Eve  
Shopping cart  
Apple

Eve  
Shopping cart  
Banana

Adam  
Wishlist  
Tomato

Eve  
Wishlist  
Cherry

Adam  
Shopping cart  
Garlic

Eve  
Shopping cart  
Apple

## aggregate

---

Eve  
*Shopping cart*  
- Apple, T1  
  
*Wishlist*

Eve  
*Shopping cart*  
- Apple, T1  
- Banana, T2  
  
*Wishlist*

Adam  
*Shopping cart*  
  
*Wishlist*  
- Tomato, T3

Eve  
*Shopping cart*  
- Apple, T1  
- Banana, T2  
  
*Wishlist*  
- Cherry, T4

Adam  
*Shopping cart*  
- Garlic, T5  
  
*Wishlist*  
- Tomato, T3

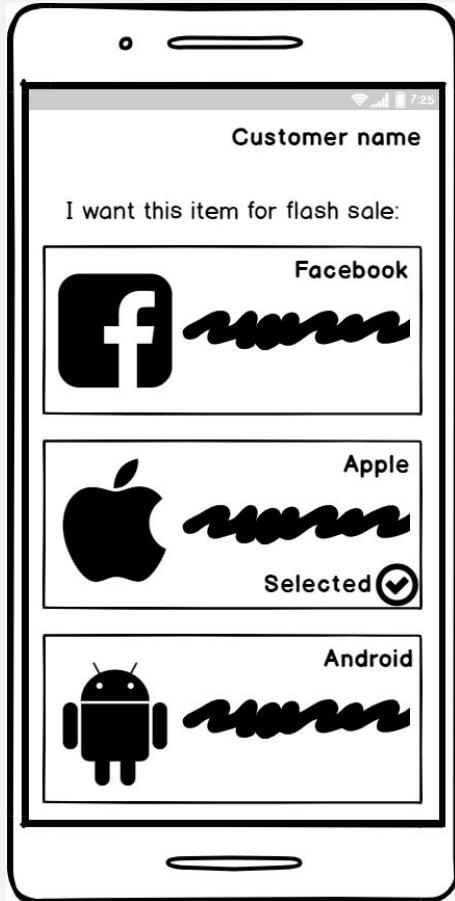
Eve  
*Shopping cart*  
- Apple, T6  
- Banana, T2  
  
  
*Wishlist*  
- Cherry, T4

# **Flash Sale Stream**

## **Most Recent Data Feed**



# Flash Sale by Customer



- Customer can vote flash sale candidate
- Next flash sale will be most voted items
- One customer, one candidate
- Can change selection

# Stream or Table?

- × Track latest selected item per customer
- × Kafka Stream table
- × Record
  - × Key : Customer ID
  - × Value : Flash sale candidate
- × Upsert (update if exist / insert if not exist)

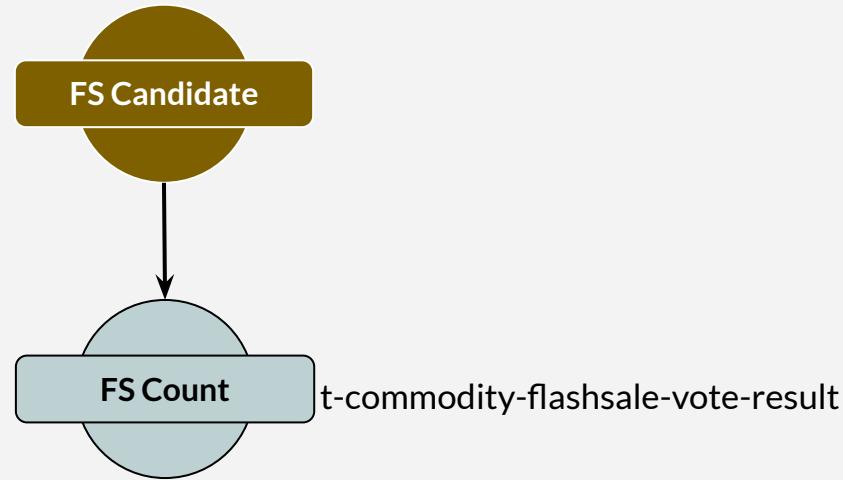


# Source Code for Flash Sale Vote

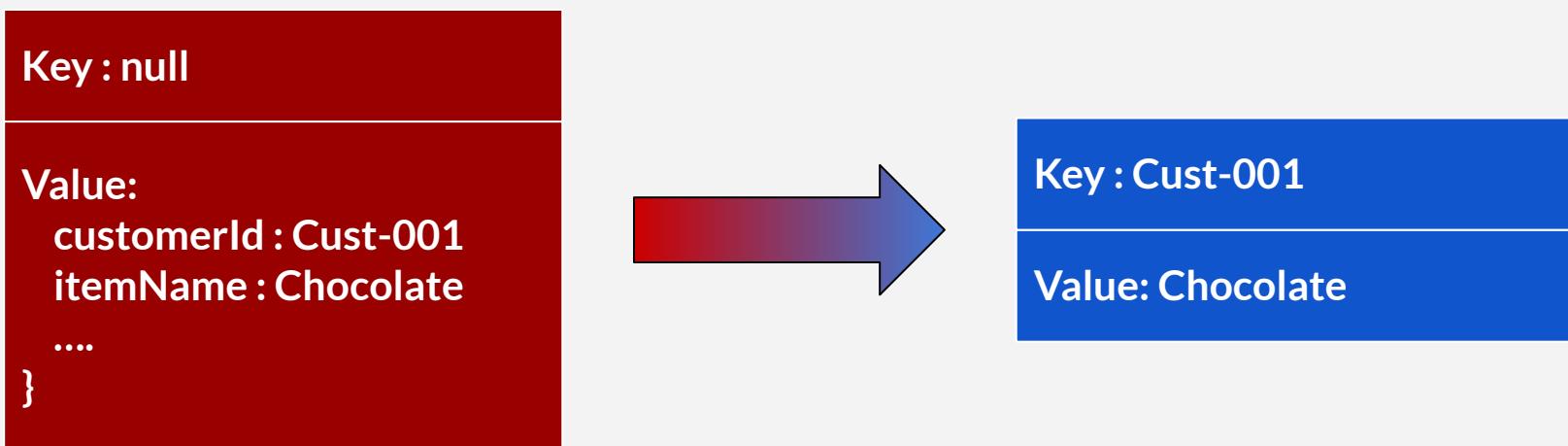
- ✗ `FlashSaleVote*.java`
- ✗ Package `com.course.kafka`
  - ✗ `api.request`
  - ✗ `api.server`
  - ✗ `broker.message`
  - ✗ `broker.producer`
  - ✗ `command.action`
  - ✗ `command.service`



# High Level Topology



# Processing The Message



# How Vote Works?

Anna	Cookies	
------	---------	---

Olaf	Cookies	
------	---------	---

Olaf	Cake	
------	------	---

Anna	Cake	
------	------	---

Elsa	Cookies	
------	---------	---

	Cookies	1
---	---------	---

	Cookies	2
---	---------	---

	Cookies	1		Cake	1
---	---------	---	---	------	---

	Cookies	0		Cake	2
---	---------	---	---	------	---

	Cookies	1		Cake	2
---	---------	---	---	------	---

timeline



# Flash Sale Stream

## Stream & State



# State in Kafka Stream

- × Care about current data
- × No need to know previous data
- × Flash sale example : need to know current / previous choice
- × "User choice" is kafka stream **state**
- × Stateless operations (commodity & feedback)
- × Stateful operations (flash sale vote)

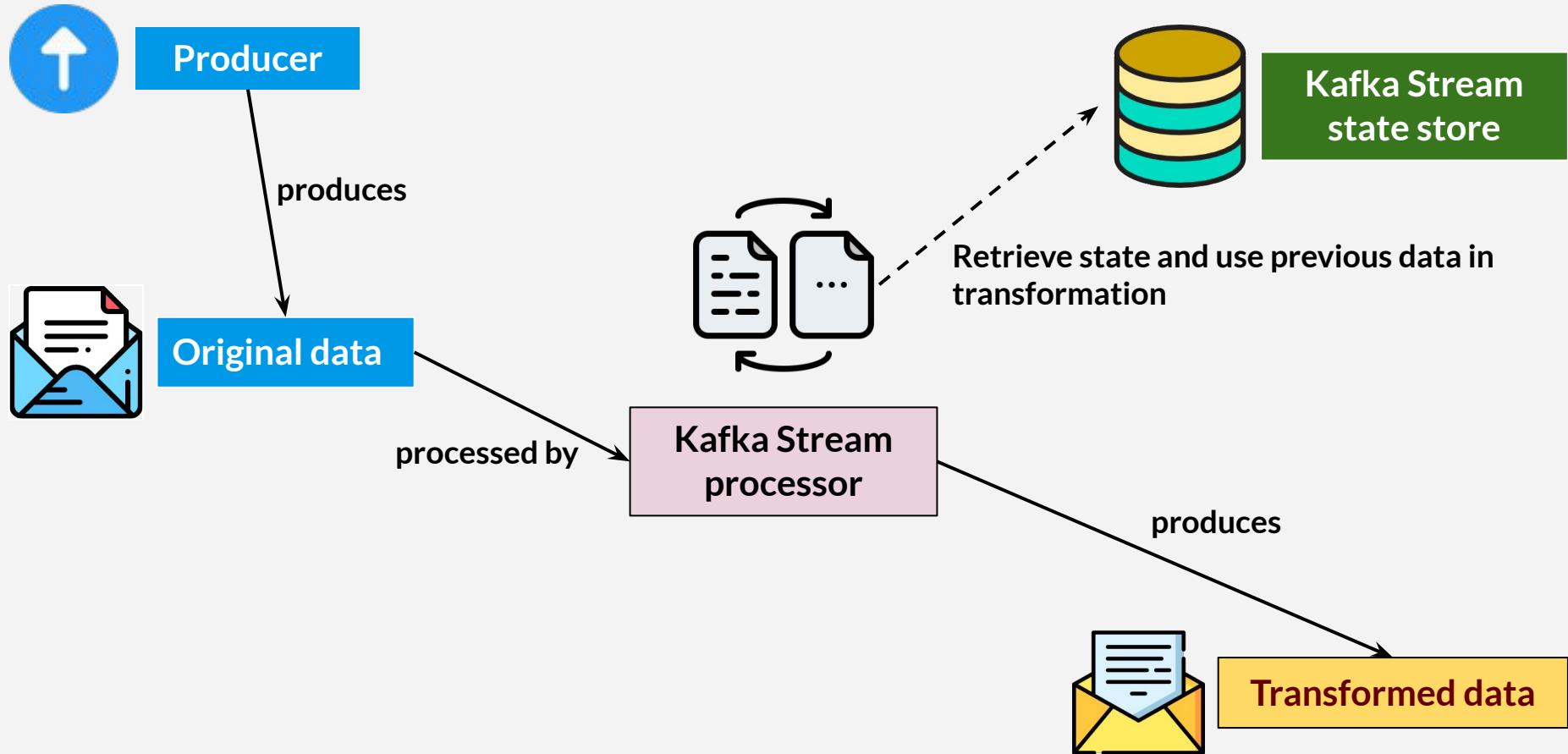


# State Store

- × See existing information & connect it
- × Kept in state store
- × Key-value data storage
- × Accessed from processor
- × Kafka stream state stores:
  - × In-memory
  - × Persistent (disk based)



# State Store



# Important Aspects

- × Data locality
  - × Same machine with processing node
  - × No network overhead
  - × No sharing store
- × Fault tolerance
  - × Recover quickly in case application failure
  - × Use changelog topic

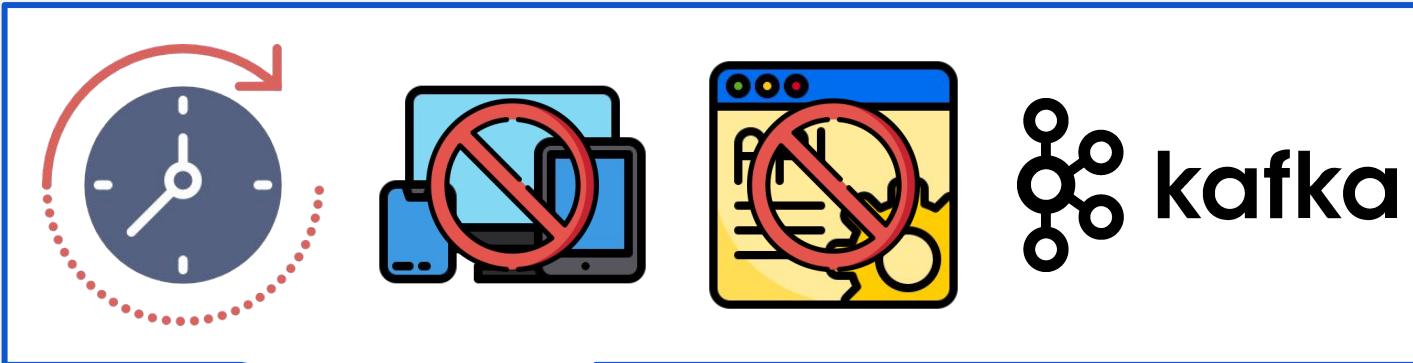


# Flash Sale Stream

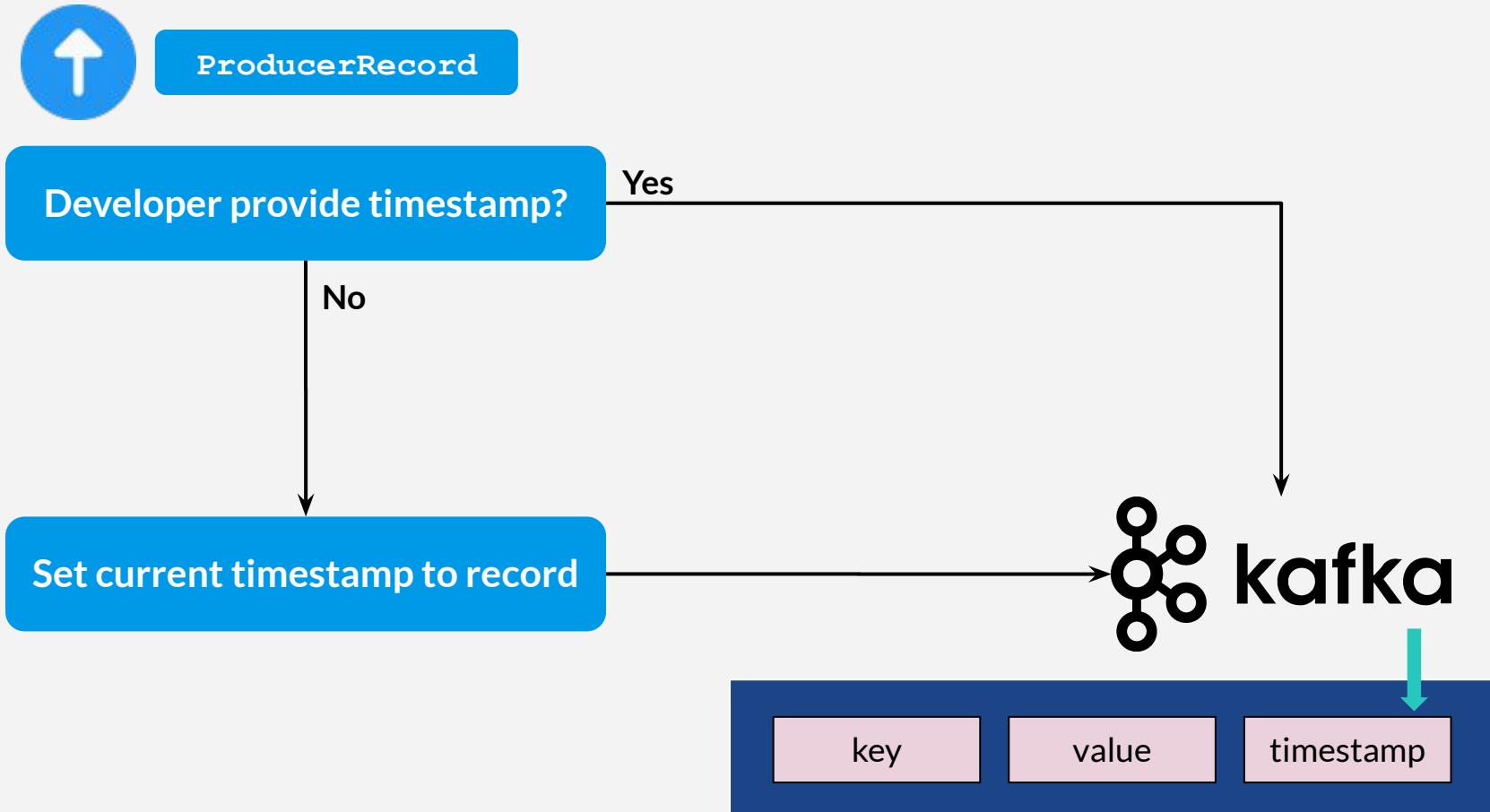
## Timestamp



# Flash Sale by Customer



# Kafka Timestamp



# Processor API

- × Use processor API
- × Extract timestamp
- × Useful not only for timestamp
- × Learn basic processor API



# Value Transformer

```
stream.mapValues()
```



```
stream.transformValues()
```



```
class X implements ValueTransformer<Value, TransformedValue>  
  
.transformValues(() -> new X())
```

```
class Y implements ValueTransformerSupplier<Value, TransformedValue> {  
  
    public ValueTransformer<Value, TransformedValue> get() {...}  
  
}  
  
.transformValues(new Y())
```

# OffsetDateTime to Timestamp

- ✗ Epoch time millisecond vs Java OffsetDateTime
- ✗ `OffsetDateTime.toInstant().toEpochMilli()`



# processValues

- ✗ `transformValues` is deprecated
- ✗ Use `processValues`
- ✗ Requires `FixedKeyProcessor<keyInput, valueInput, valueOutput>`
- ✗ `valueInput` can be different from `valueOutput`



# Feedback Rating

## Average Rating



# Feedback Dashboard

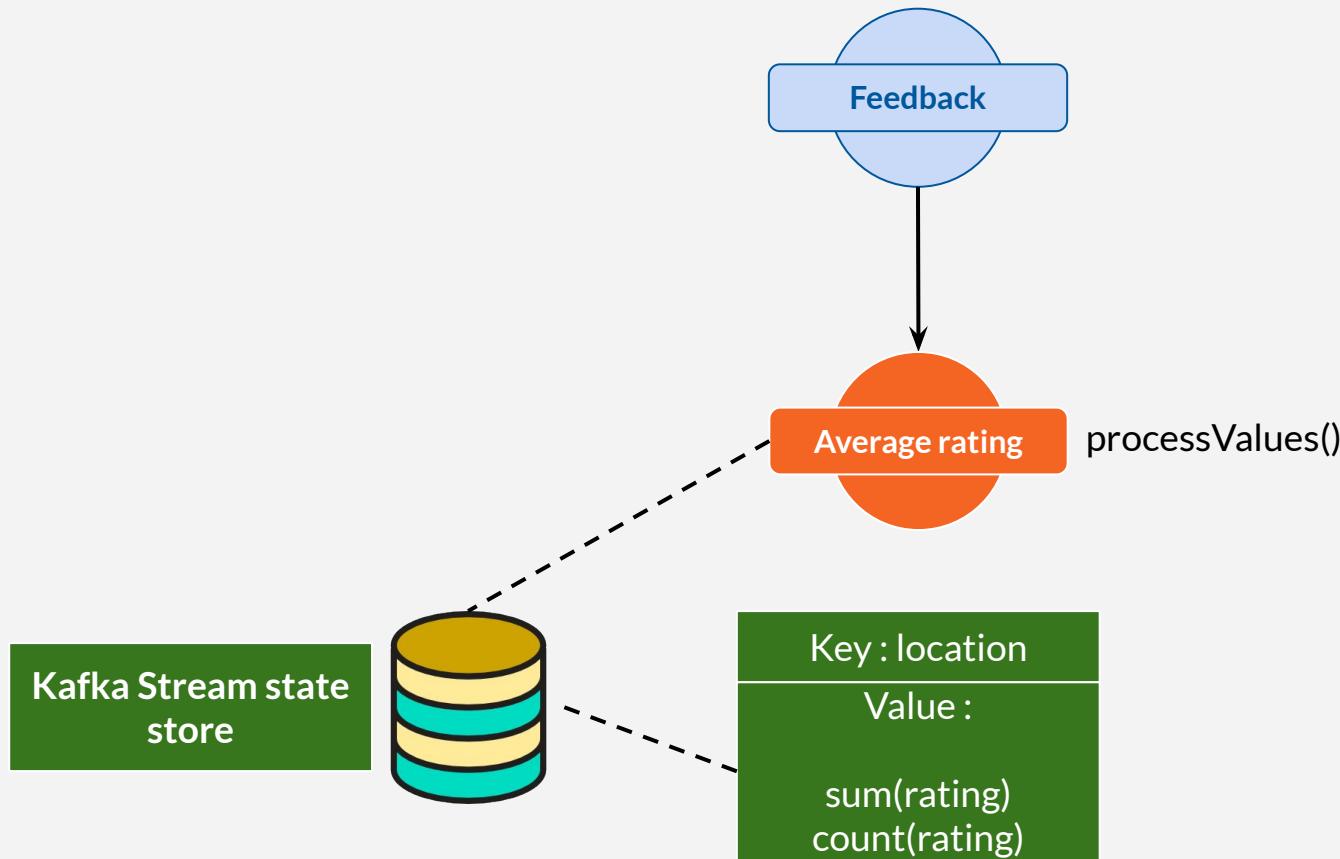


# Average Rating

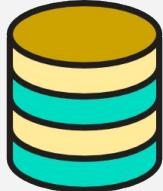
- × average = sum(ratings) / count(ratings)
- × Need to know all ratings
- × Use processor API & state store
- × State store?



# High Level Topology



# State Store



Indonesia
Sum : 12
Count : 4

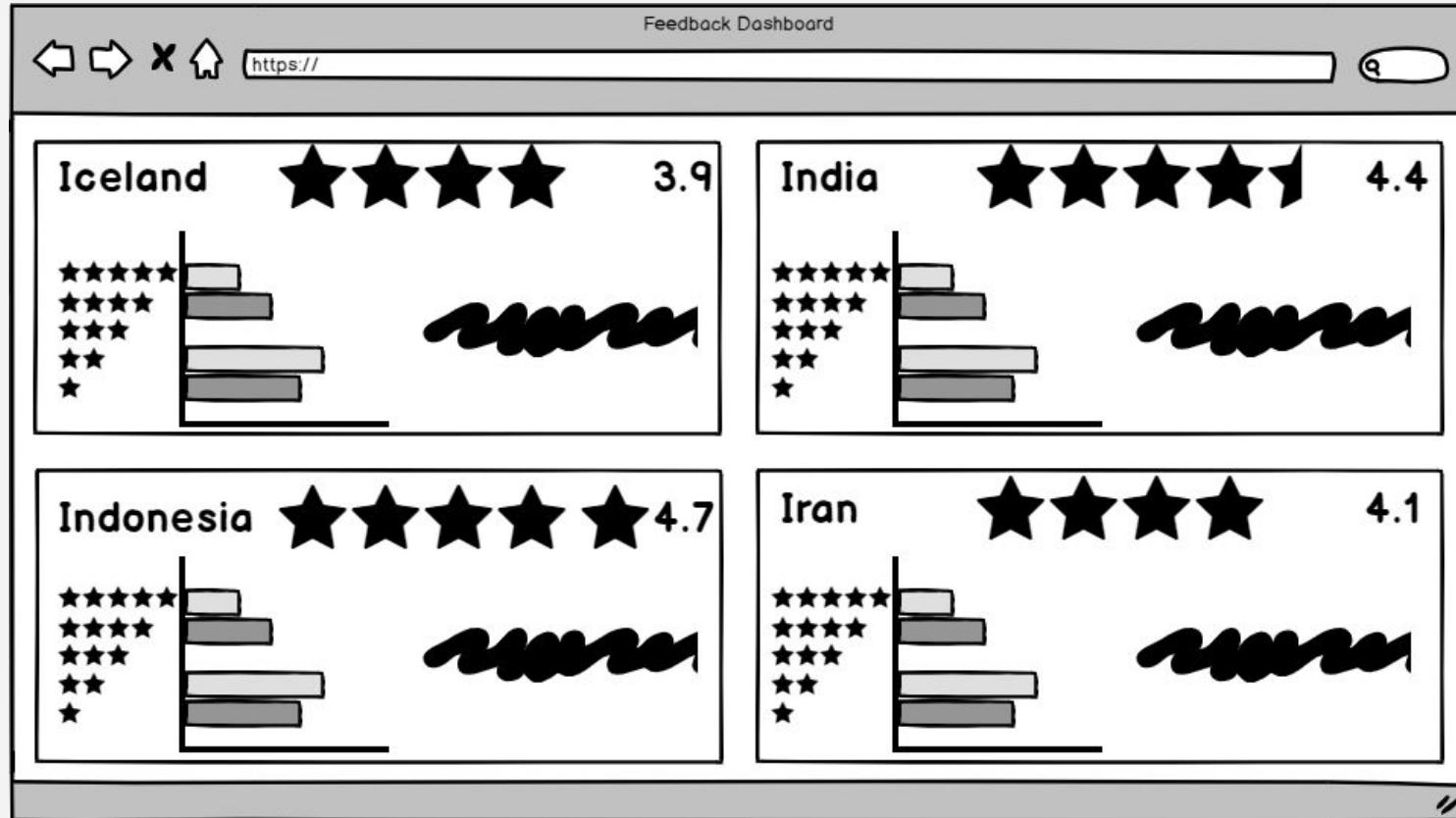
Singapore
Sum : 8
Count : 2

# Feedback Rating

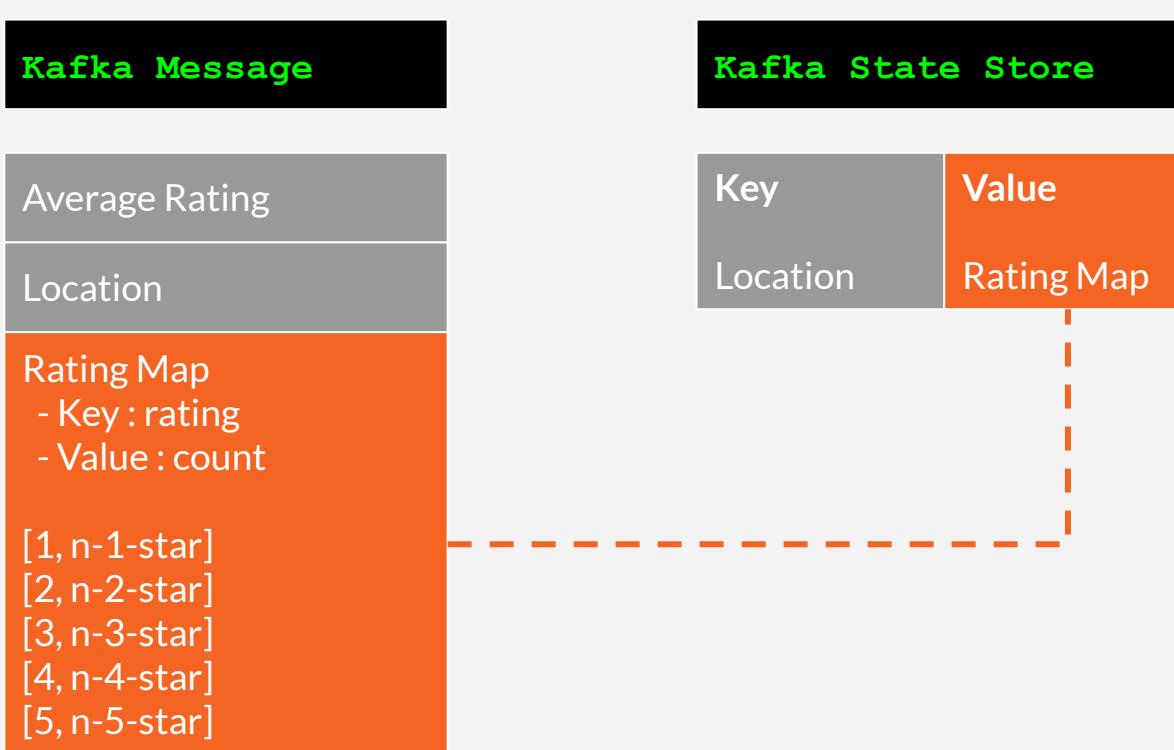
## Detailed Rating



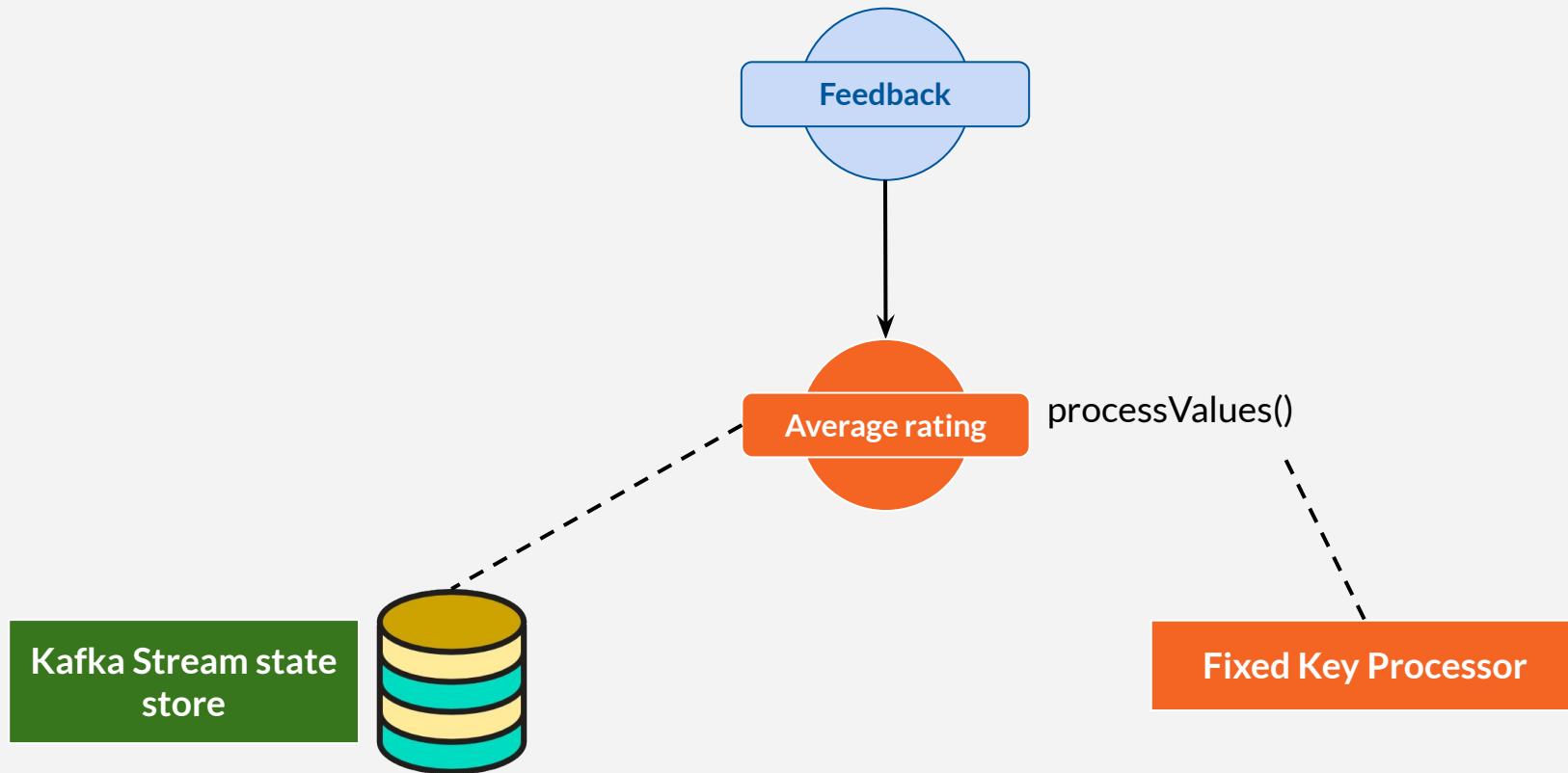
# Feedback Dashboard



# Data Structure



# High Level Topology

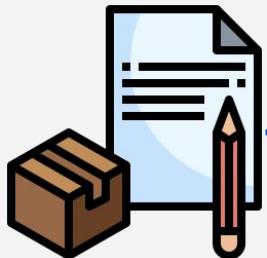


# Inventory Summing Records

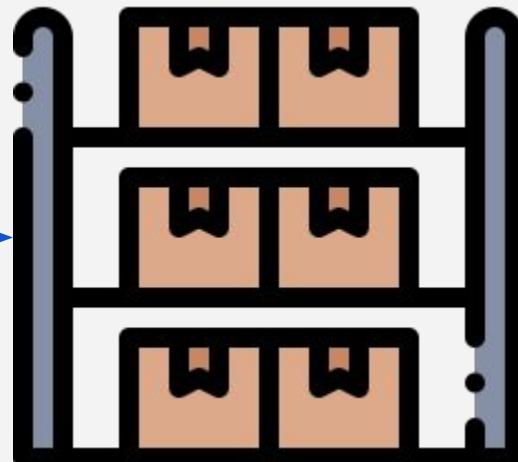


# Inventory

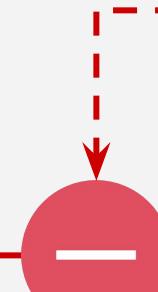
Supplier delivery



Item refund, etc

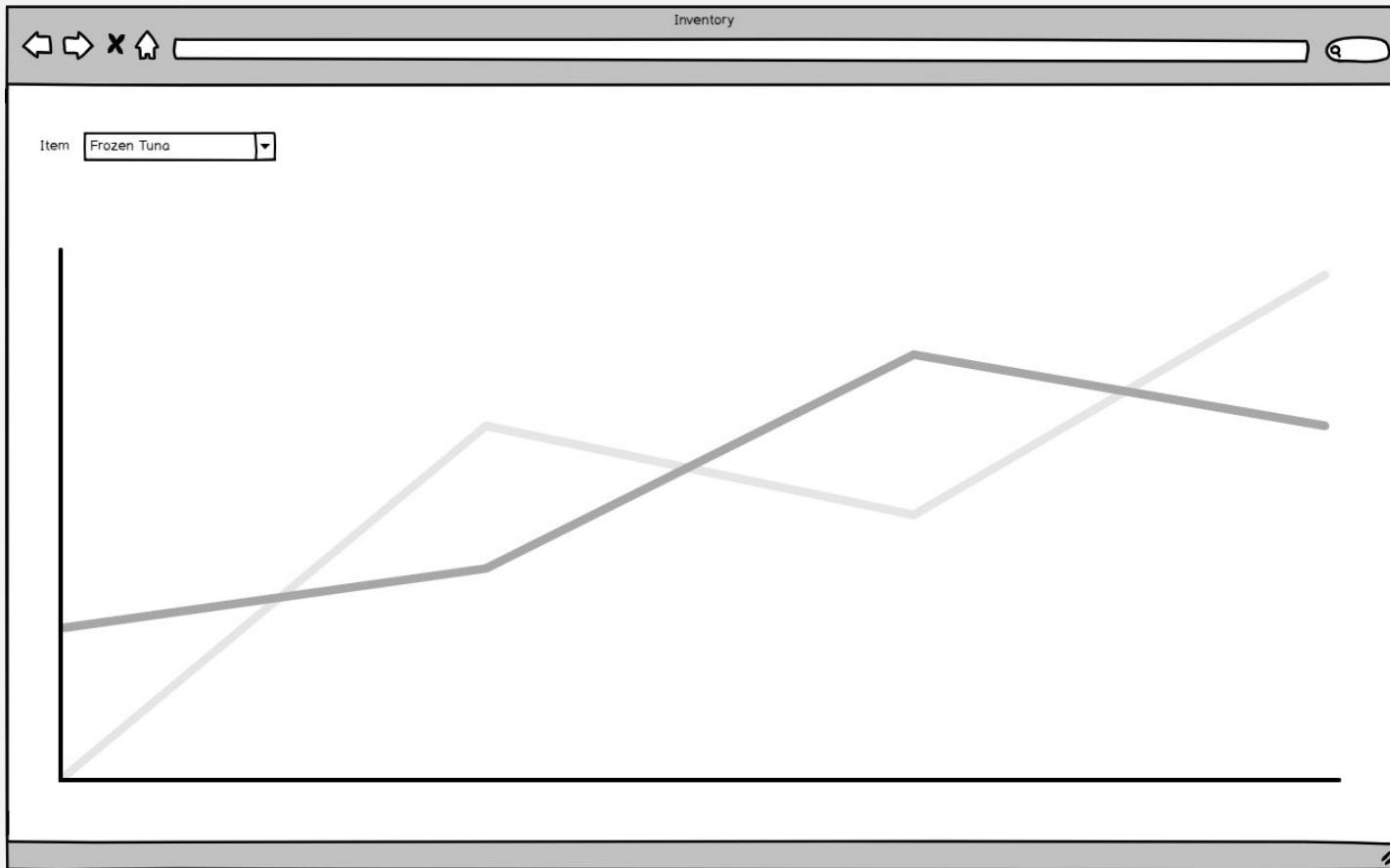


Item sold



Item moved to store display, etc

# Inventory Chart



# Source Code for Inventory

- ✗ **Inventory\*.java**
- ✗ **Package com.course.kafka**
  - ✗ **api.request**
  - ✗ **api.server**
  - ✗ **broker.message**
  - ✗ **broker.producer**
  - ✗ **command.action**
  - ✗ **command.service**

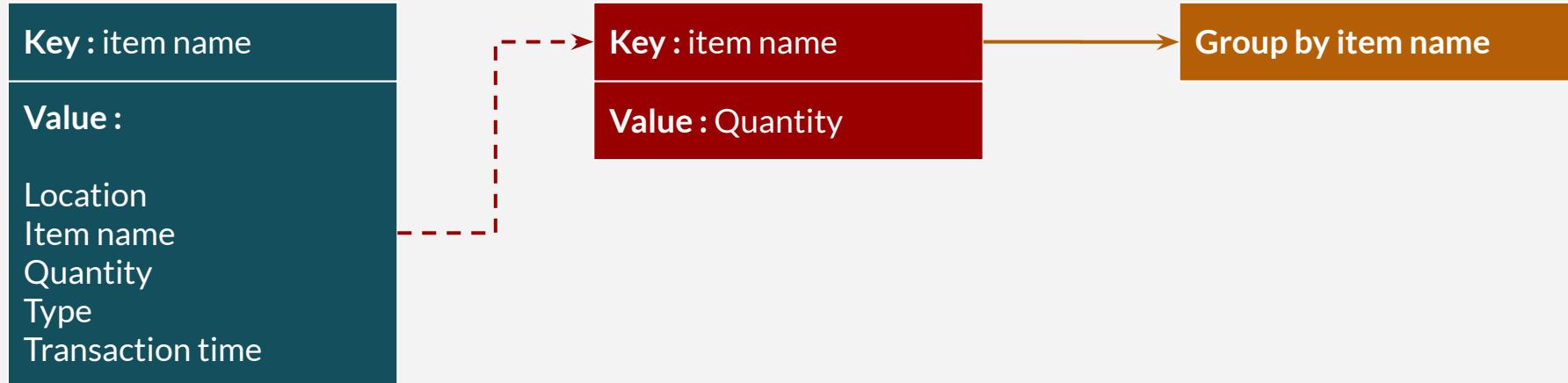


# In This Section

- × Kafka stream aggregation (sum)
- × Inventory message sent
  - × Key : item name
  - × Value : item name, location, quantity, type
  - × Value (optional) : transaction time
- × No validation, e.g :
  - × Negative amount
  - × Inventory quantity still enough for removal
  - × etc



# Grouping & Aggregate



# Grouping & Aggregate

Record #	Record after map values	Group by key	KGroupedStream aggregate		Aggregated
			Initializer	Adder	
1	(candy, 4)	(candy, 4)	0 (for candy)	(candy, 0 + 4)	(candy, 4)
2	(apple, 2)	(apple, 2)	0 (for apple)	(apple, 0 + 2)	(apple, 2)
3	(candy, 3)	(candy, 3)		(candy, 4 + 3)	(candy, 7)
4	(candy, 1)	(candy, 1)		(candy, 7 + 1)	(candy, 8)
5	(apple, 4)	(apple, 4)		(apple, 2 + 4)	(apple, 6)
6	(apple, 5)	(apple, 5)		(apple, 6 + 5)	(apple, 11)

# Inventory Subtracting Value



# Grouping & Aggregate



# Inventory

## Using Reduce



# Inventory Timestamp Extractor



# Timestamp

Key : item name

Value :

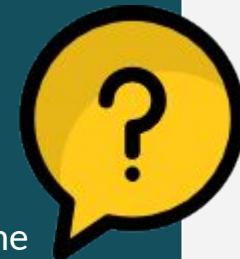
Location

Item name

Quantity

Type

Transaction time



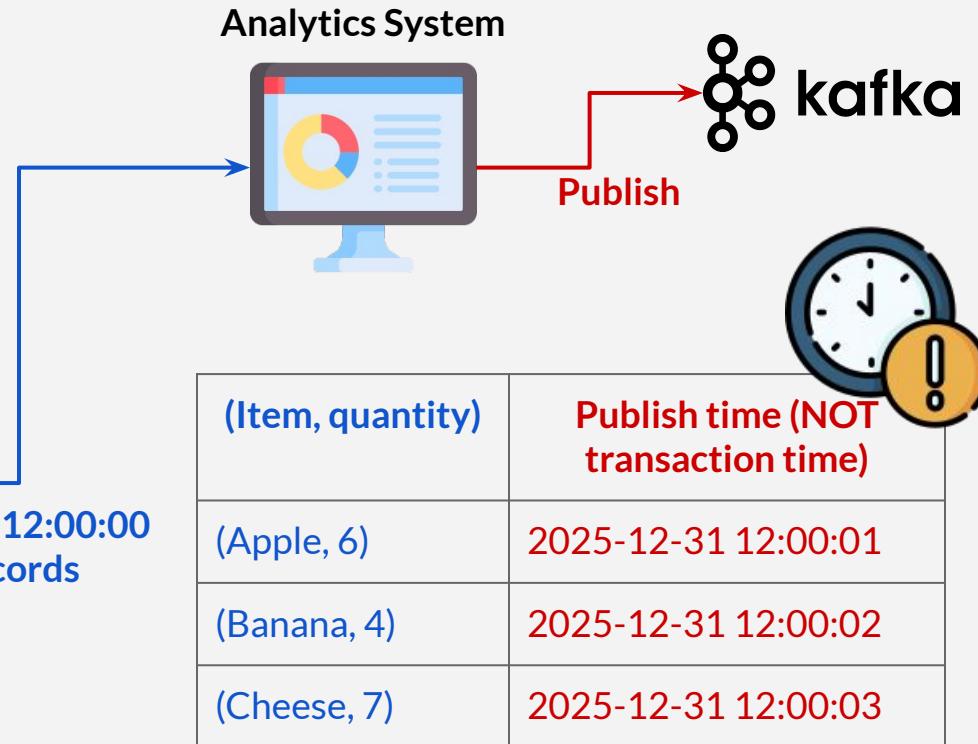
# Timestamp

(Item, quantity)	Transaction time
(Apple, 6)	2025-12-31 08:56:00
(Banana, 4)	2025-12-31 10:24:00
(Cheese, 7)	2025-12-31 11:37:00



Warehouse Inventory  
System

Hit API start at 12:00:00  
For all 3 records



# Timestamp Extractor

- × Extract transaction time from payload
- × Use it as record timestamp
- × Built-in timestamp extractor (FYI)
  - × FailOnInvalidTimestamp
  - × LogAndSkipOnInvalidTimestamp
  - × UsePreviousTimeOnInvalidTimestamp
  - × WallclockTimestampExtractor



# Windowing



# Windowing

- × Group records for same key
- × Time-based & session based
- × Example : How many click advertisement X in 15 minutes?
- × Need to know the accumulative result based on 15 minutes window
  - × The 1st 15 minutes : 2000
  - × The 2nd 15 minutes : 3000
  - × Same advertisement, different grouping



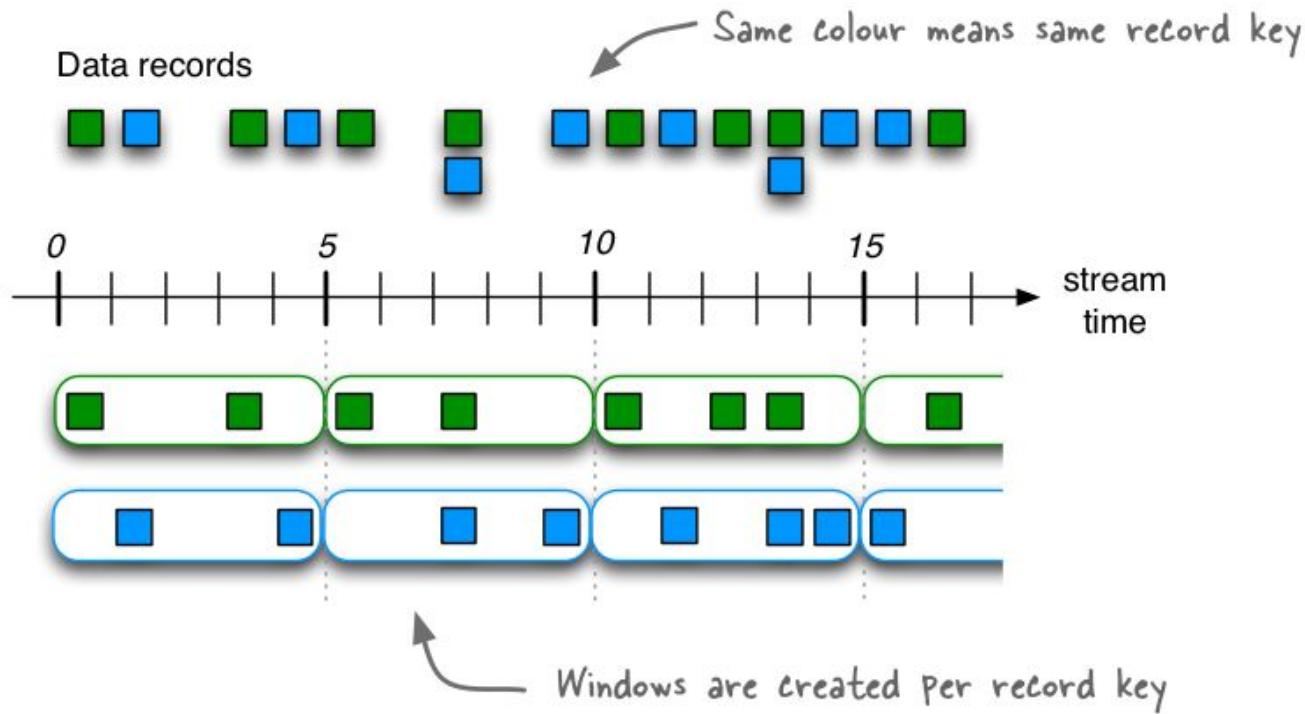
# Windowing

- ✗ Tumbling Time Window
- ✗ Hopping Time Window
- ✗ Session Based Window



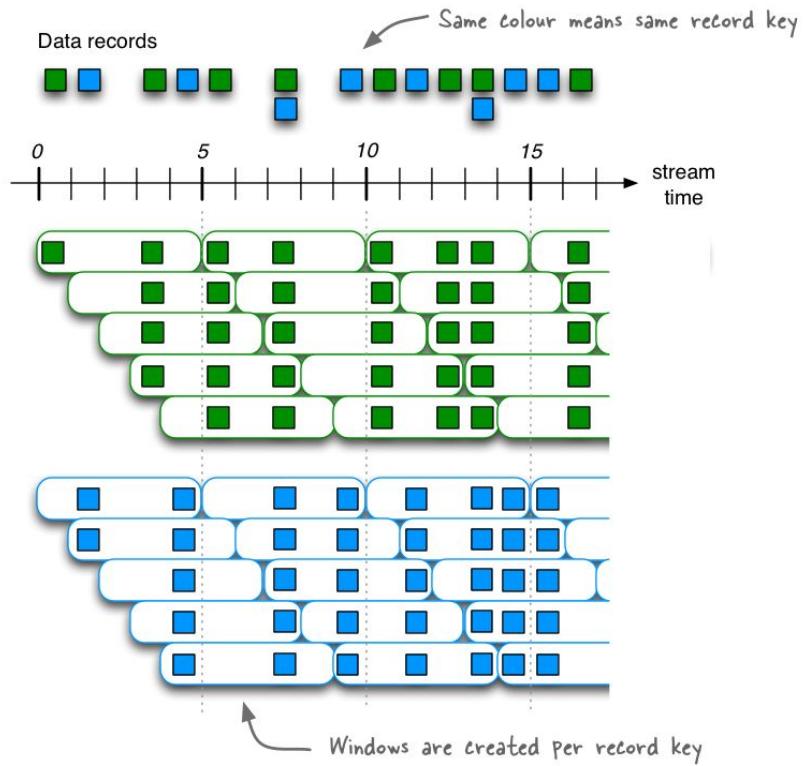
# Tumbling Time Window

## A 5-min Tumbling Window



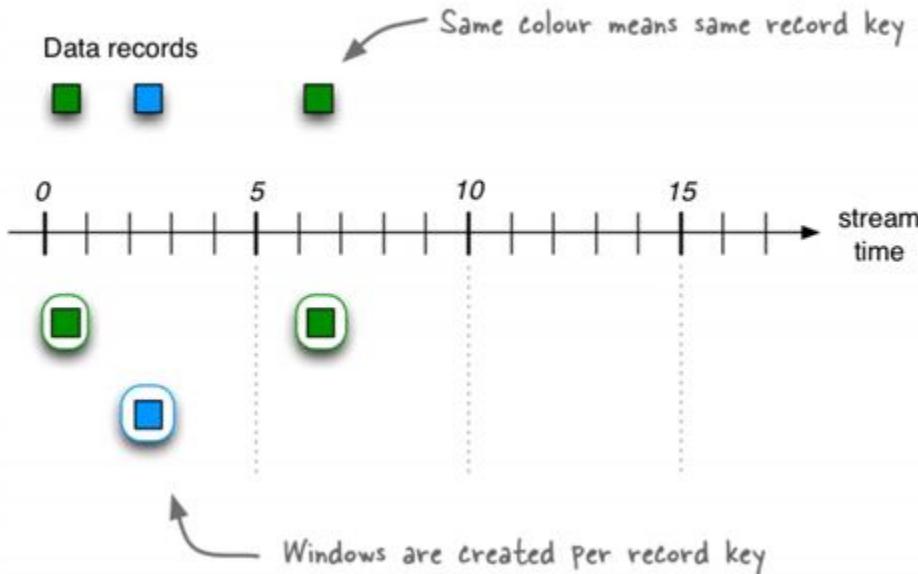
# Hopping Time Window

A 5-min Hopping Window with a 1-min "hop"



# Session Window

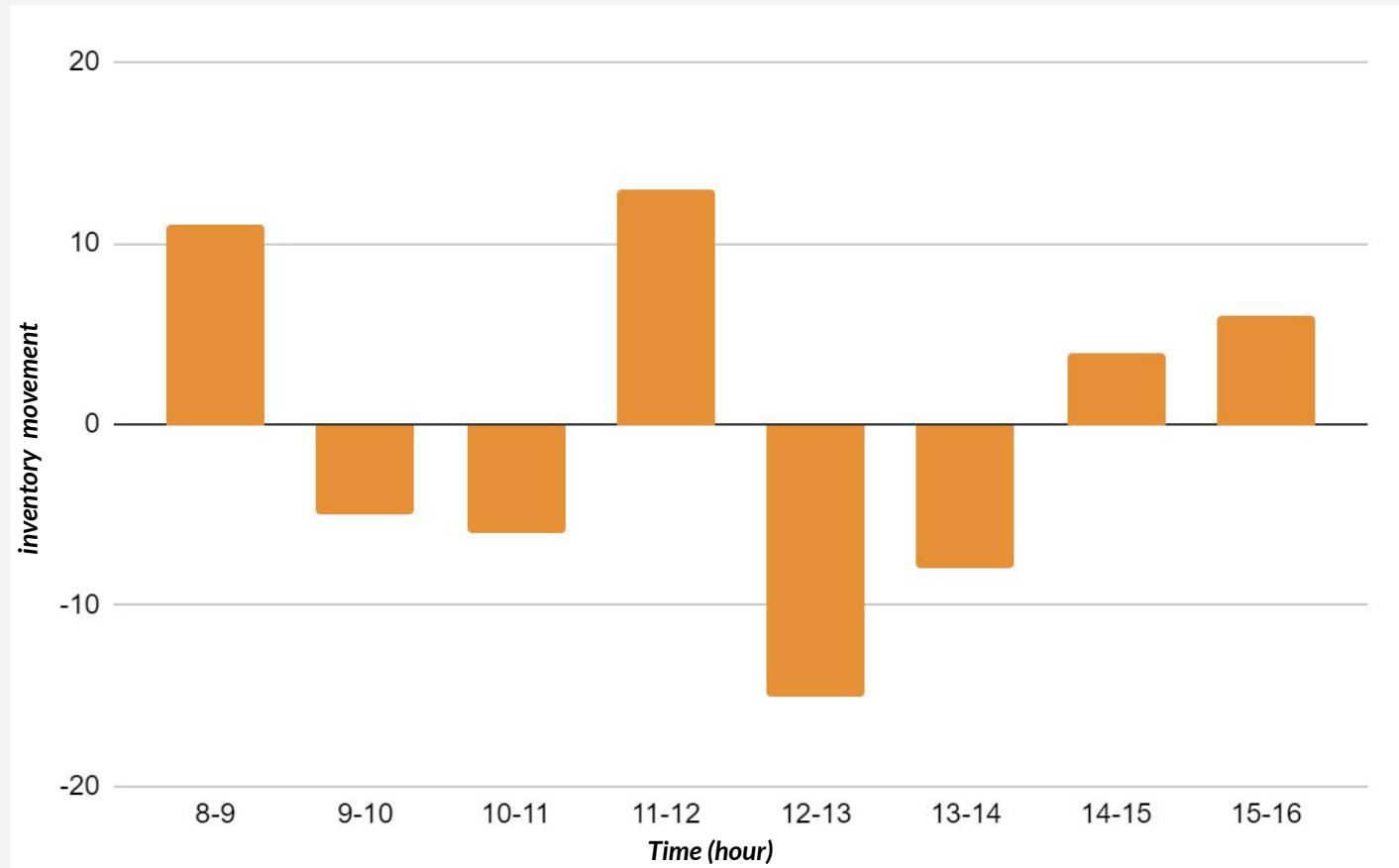
## A Session Window with a 5-min inactivity gap



# Inventory Tumbling Time Window



# Inventory Movement



# Inventory Transaction for Item X

Time Range : 08:00-09:00	
Time	Transaction
8:00:00	50
8:10:00	15
8:20:00	-8
8:30:00	7
8:40:00	-12
8:50:00	11

Aggregated Sum : 63

Time Range : 09:00-10:00	
Time	Transaction
9:00:00	9
9:10:00	4
9:20:00	-6
9:30:00	-3
9:40:00	10
9:50:00	-6

Aggregated Sum : 8

Time Range : 10:00-11:00	
Time	Transaction
10:00:00	3
10:10:00	-4
10:20:00	-17

Aggregated Sum : -18

# How We Do It

- × Extract transaction time
- × Create tumbling time window



# Log Output

Group 1:  
08:00-09:00

[Apple@2024-12-31T08:00Z/2024-12-31T09:00Z], 64  
[Apple@2024-12-31T08:00Z/2024-12-31T09:00Z], 63

Group 3:  
10:00-11:00

[Apple@2024-12-31T09:00Z/2024-12-31T10:00Z], 7  
[Apple@2024-12-31T09:00Z/2024-12-31T10:00Z], 8  
[Apple@2024-12-31T10:00Z/2024-12-31T11:00Z], -1  
[Apple@2024-12-31T10:00Z/2024-12-31T11:00Z], -18

Group 2:  
09:00-10:00

[Apple@2024-12-31T08:00Z/2024-12-31T09:00Z], 63

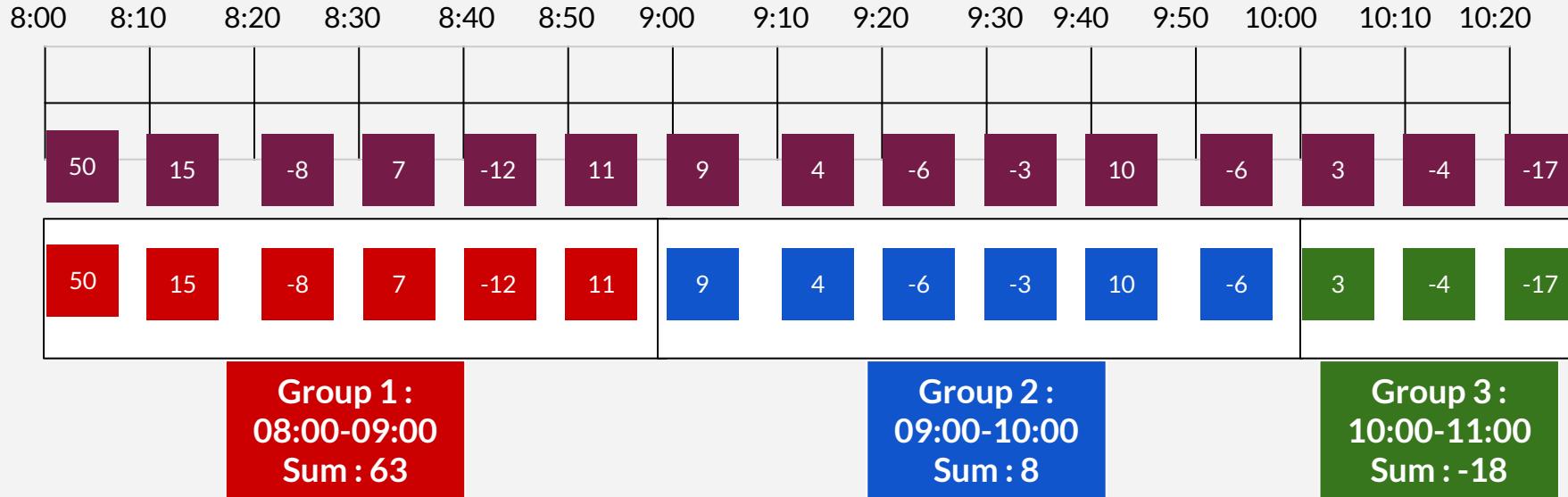
Original key

Window start

Window end

Value

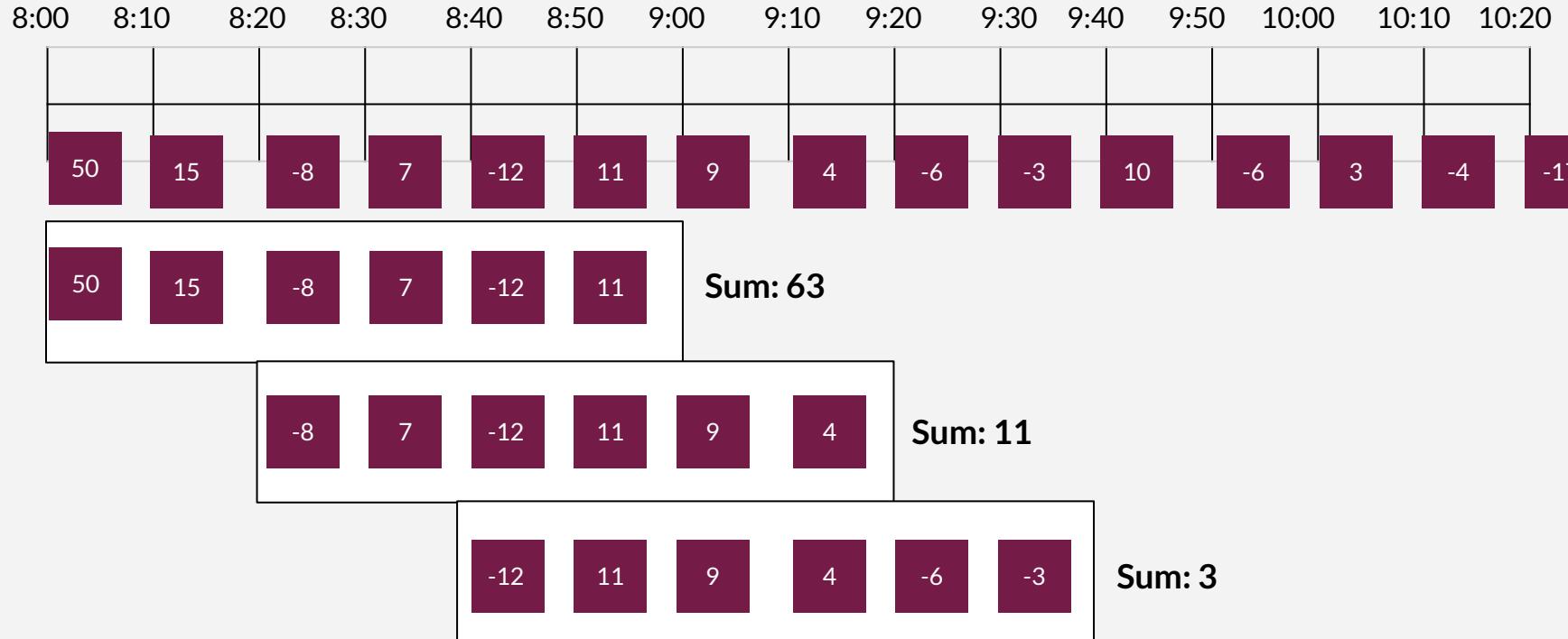
# Tumbling Time Window



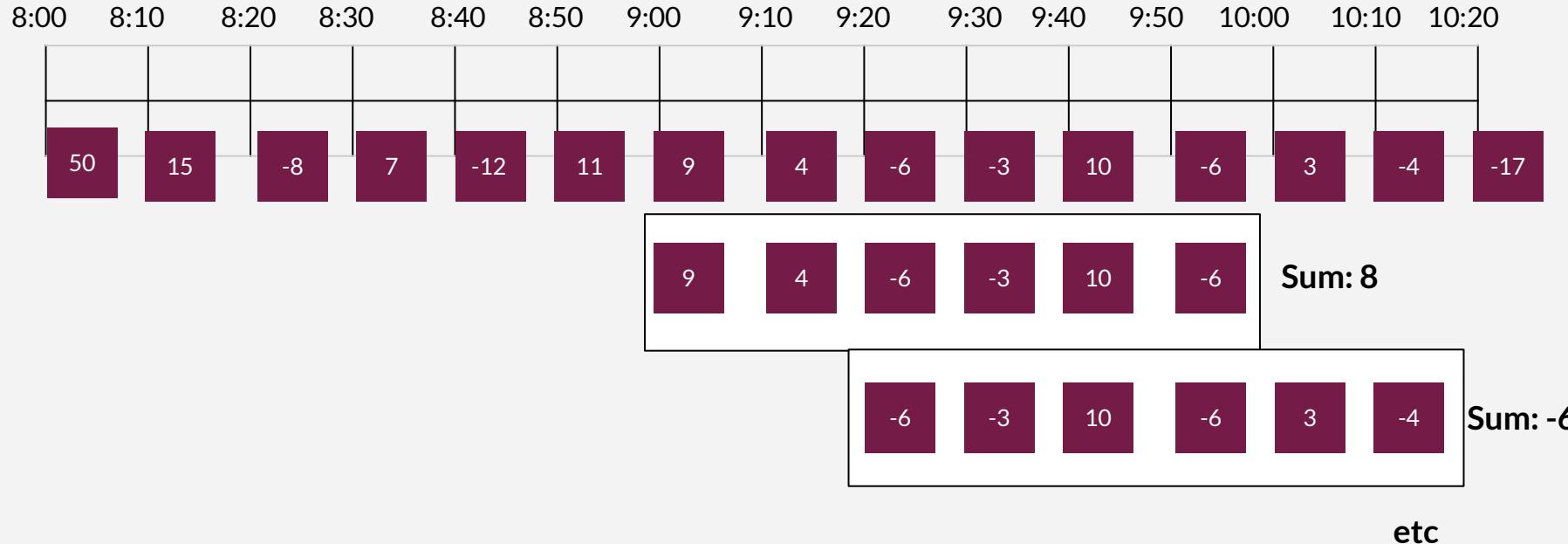
# Inventory Hopping Time Window



# Hopping Time Window



# Hopping Time Window - continued

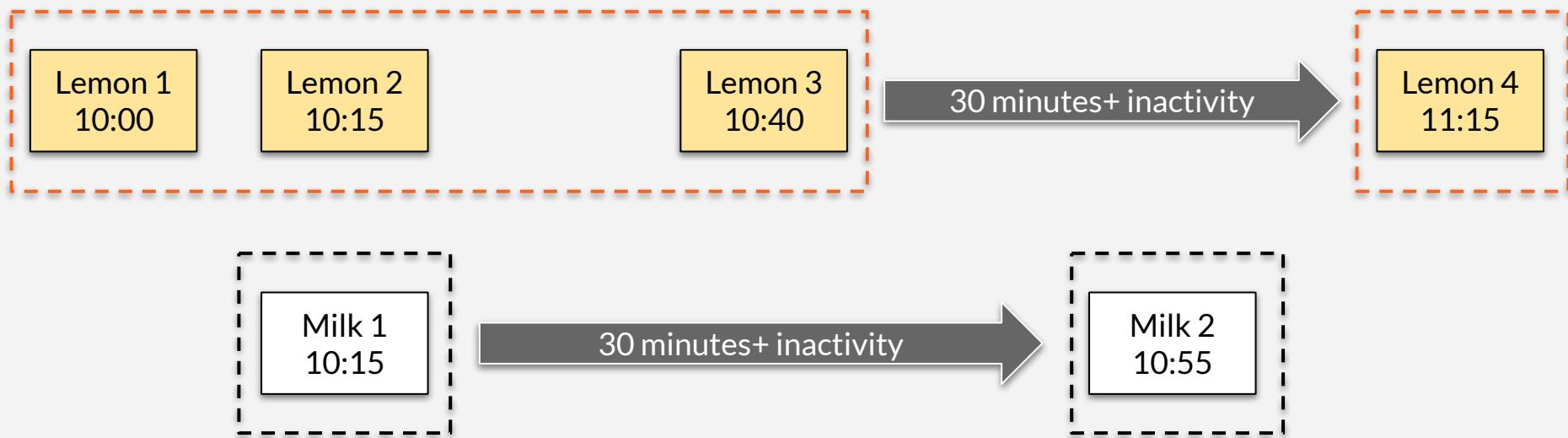


# Inventory Session Window



# Event

*Inactivity gap : 30 minutes*



# Join Theory



# Kafka Stream Join

- × Taking two streams / tables and make new stream / table out of them
- × Join based on record key
- × Join on:
  - × KStream & KStream
  - × KTable & KTable
  - × KStream & KTable
  - × KStream & GlobalKTable
- × Join types : inner, left, outer



# Join Combinations

Left Type	Right Type	Co-Partition	Windowed	Join Types		
				Inner Join	Left Join	Outer Join
KStream	KStream	Required	Yes	Supported	Supported	Supported
KTable	KTable	Required	No	Supported	Supported	Supported
KStream	KTable	Required	No	Supported	Supported	Not Available
KStream	GlobalKTable	Not required	No	Supported	Supported	Not Available

# Co-Partition

- × Combinations:
  - × KStream & KStream
  - × KTable & KTable
  - × KStream & KTable
- × Must be co-partitioned
- × Otherwise, **runtime error**



# Co-partition

- × Left type must have same number of partition with right type
  - × Left : x & Right : x = co-partitioned
  - × Left : x & Right : y = Not co-partitioned
- × Same partitioning strategy
  - × Key with same value must go same partition
  - × If use custom logic to send to partition n , both must be the same



# GlobalKTable

- × No need co-partition
- × GlobalKTable:
  - × Fetch / copy all data from all partitions of certain topic
  - × Live in kafka stream instance
  - × Consumes memory / disk on kafka stream instance
  - × Relatively small data & almost static
- × Can join stream to GlobalKTable, although different partition number

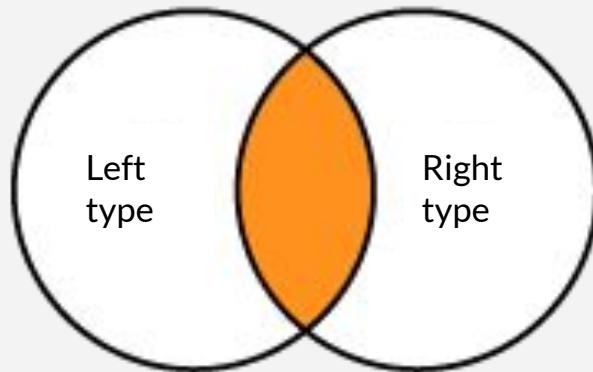


# Windowed KStream-KStream

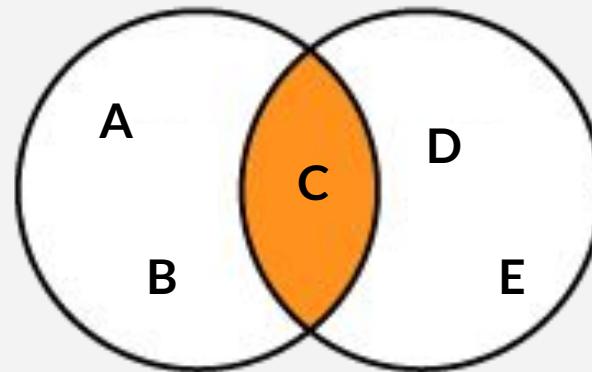
- ✗ KStream : unbounded, unlimited data
- ✗ KStream / KStream join need constraints
- ✗ Otherwise both KStream will be scanned when new data arrives -> **huge performance cost**
- ✗ Use window for constraint



# Inner Join



example →

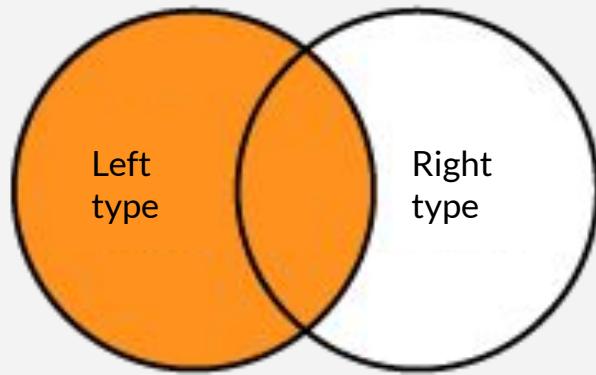


Left (Primary)	Right (Secondary)
KStream	KStream
KTable	KTable
KStream	KStream
KStream	GlobalKTable

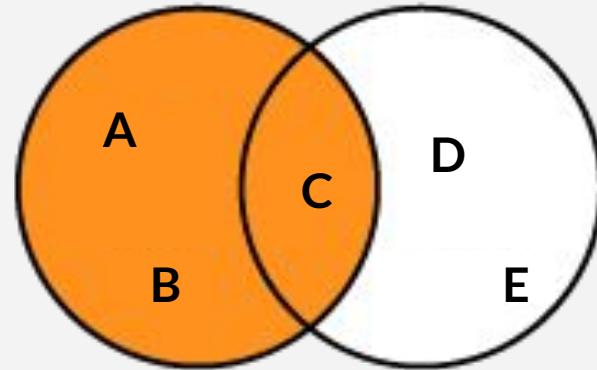
(left value, right value)

Result is:  
(C, C)

# Left Join



example →

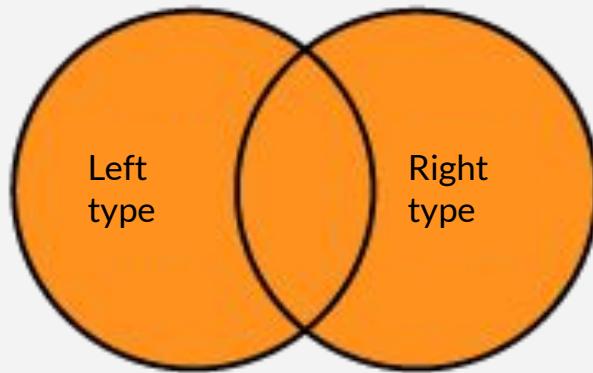


Left (Primary)	Right (Secondary)
KStream	KStream
KTable	KTable
KStream	KStream
KStream	GlobalKTable

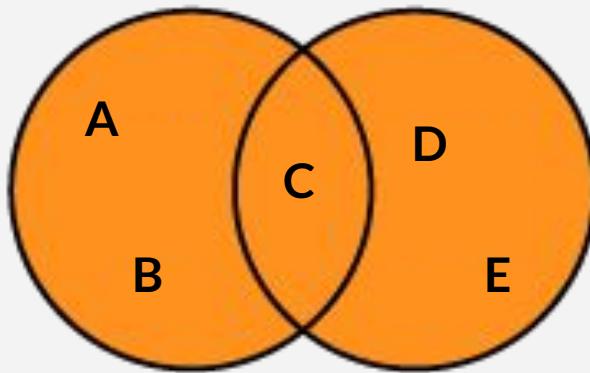
(left value, right value)

Result are:  
(A, null)  
(B, null)  
(C, C)

# Outer Join



example →



Left (Primary)	Right (Secondary)
KStream	KStream
KTable	KTable

(right value, left value)

Result are:  
(A, null)  
(B, null)  
(C, C)  
(null , D)  
(null, E)

# Join Stream / Stream

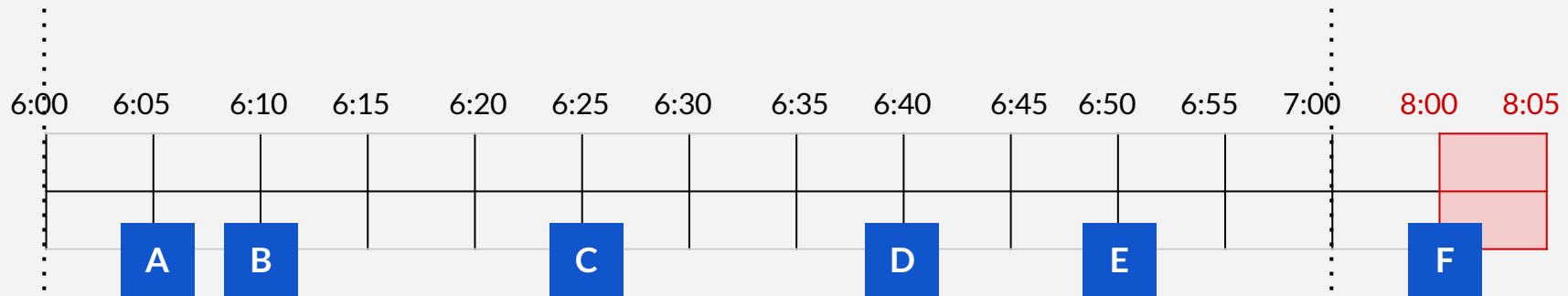


# Join Window

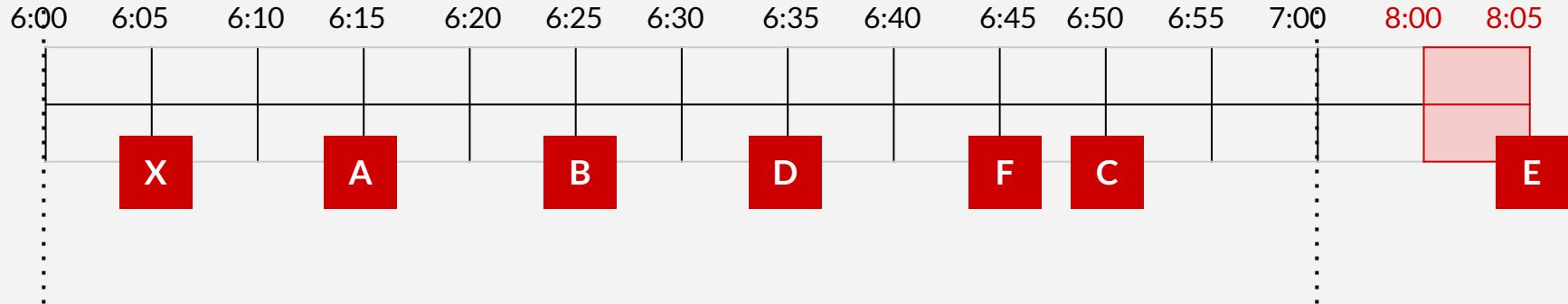
- × Class `JoinWindows`
- × Define maximum time difference
- × Record key X arrives on left stream at 7:15
  - × Matching record on right stream between 7:15 - 8:15
  - × 7:16 : match
  - × 7:50 : match
  - × 8:12 : match
  - × 8:17 : not match



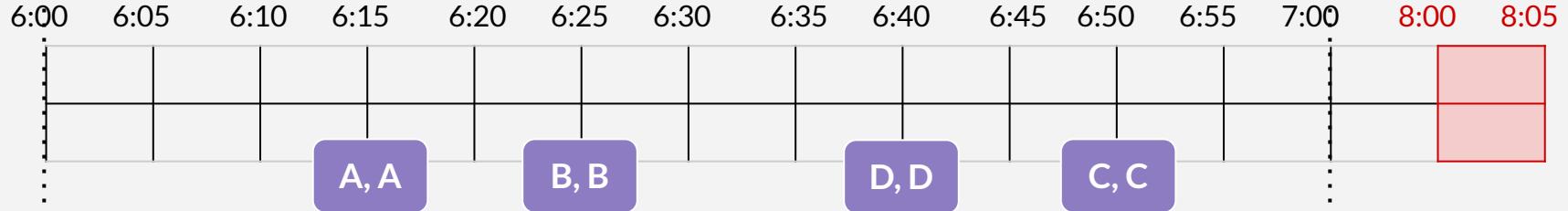
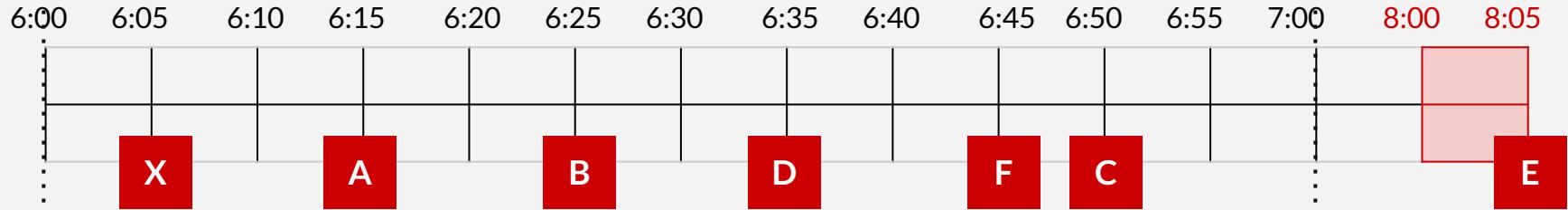
# Input Streams



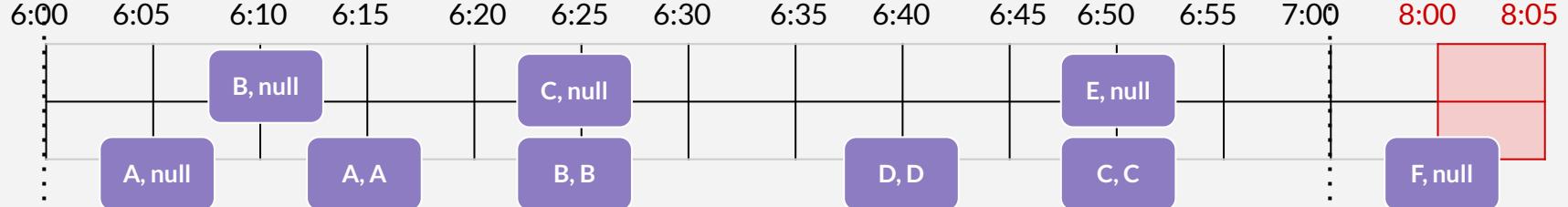
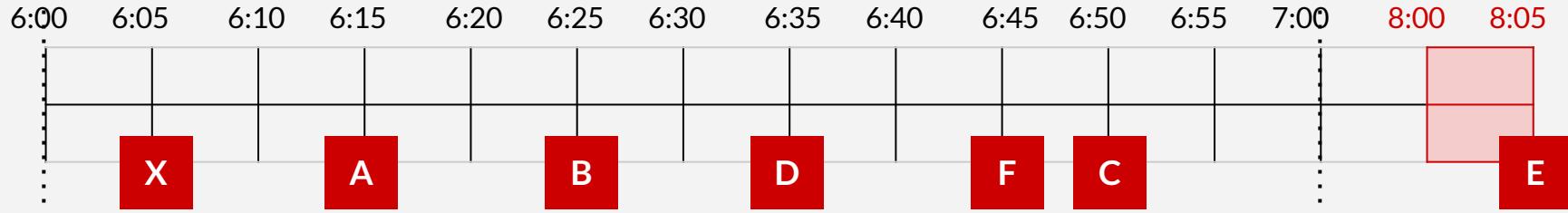
Time window : 1 hour



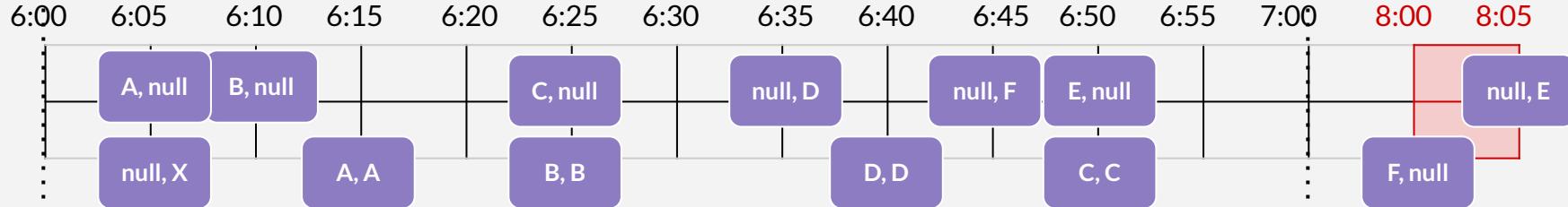
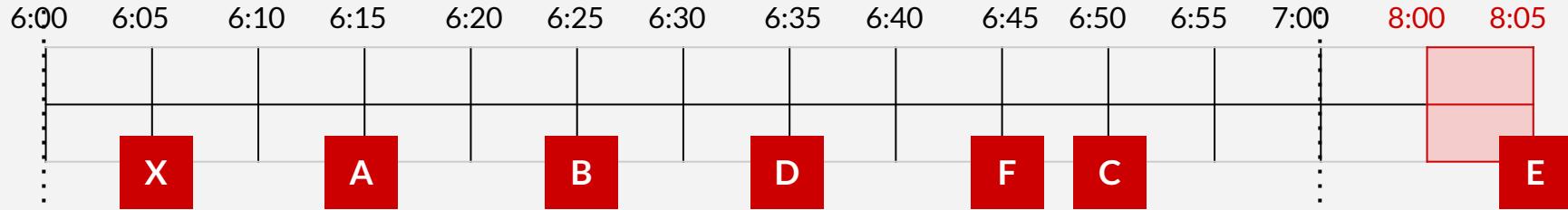
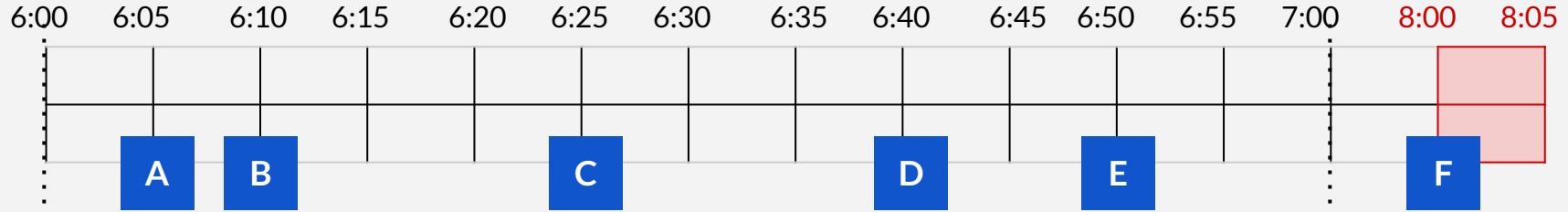
# Inner Join Stream-Stream



# Left Join Stream-Stream



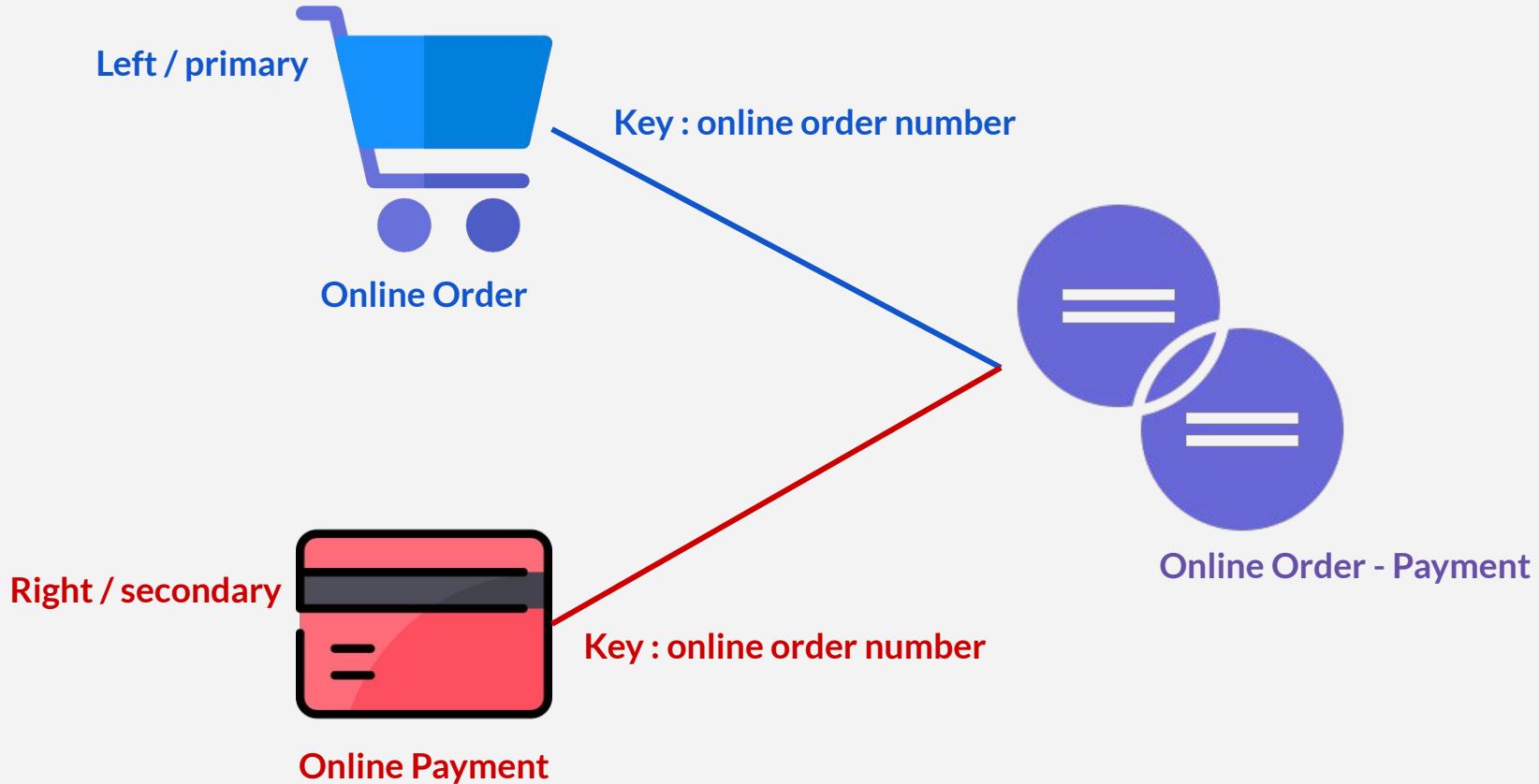
# Outer Join Stream-Stream



# Join Inner Join Stream / Stream



# Online Order & Payment



# Source Code for Online Order / Payment

- ✗ `OnlineOrder*.java`
- ✗ `OnlinePayment*.java`
- ✗ Package `com.course.kafka`
  - ✗ `api.request`
  - ✗ `api.server`
  - ✗ `broker.message`
  - ✗ `broker.producer`
  - ✗ `command.action`
  - ✗ `command.service`



# Join Class

`OnlineOrderMessage.java`

1st class : left source stream

```
int a  
Int b
```



`OnlinePaymentMessage.java`

2nd class : right source stream

```
double x
```

`OnlineOrderPaymentMessage.java`

3rd class : join class

```
int a  
int b  
double x
```

join class example 2

```
int a  
double x
```

join class example 3

```
int a  
String someField  
double aPlusX()
```

# Join Syntax

```
leftStream.join(rightStream, joiner, windows, joined)
leftStream.leftJoin(rightStream, joiner, windows, joined)
leftStream.outerJoin(rightStream, joiner, windows, joined)
```

```
// joiner

private JoinClass joiner(LeftClass left, RightClass right) { ... }
```

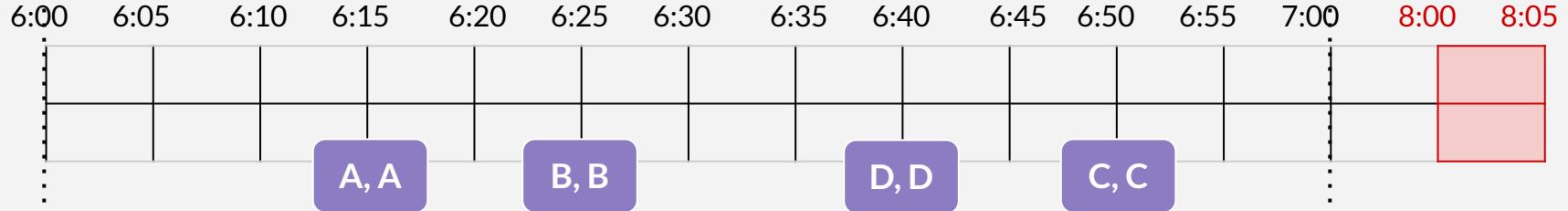
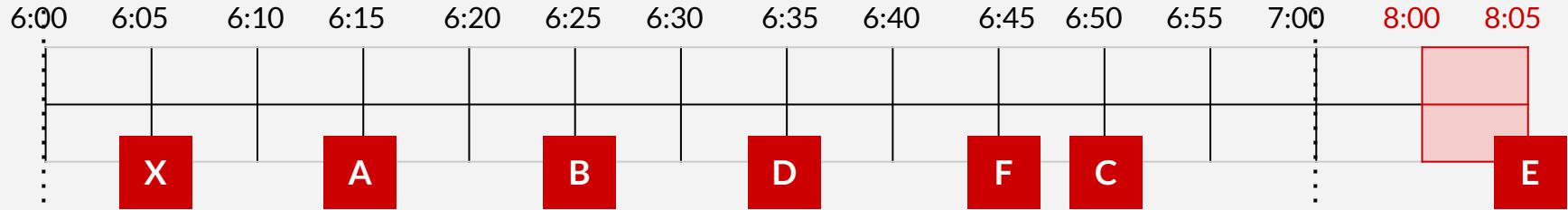
```
// windows

JoinWindows.of(...)
```

```
// joined

Joined.with(keySerde, leftSerde, rightSerde)
```

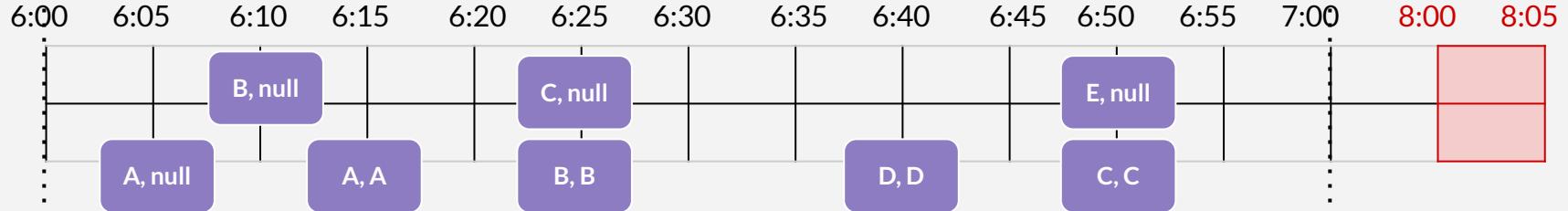
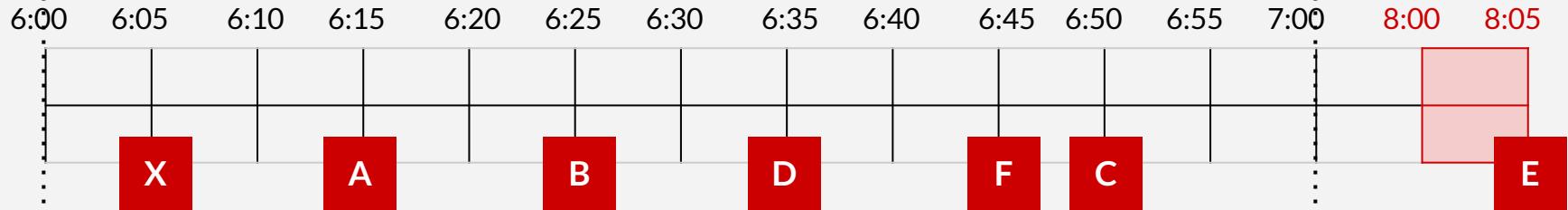
# Inner Join Stream-Stream



# Join Left Join Stream / Stream



# Left Join Stream-Stream

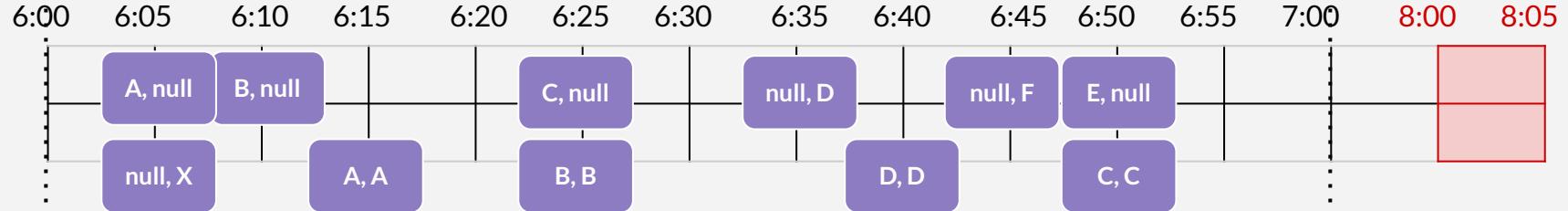
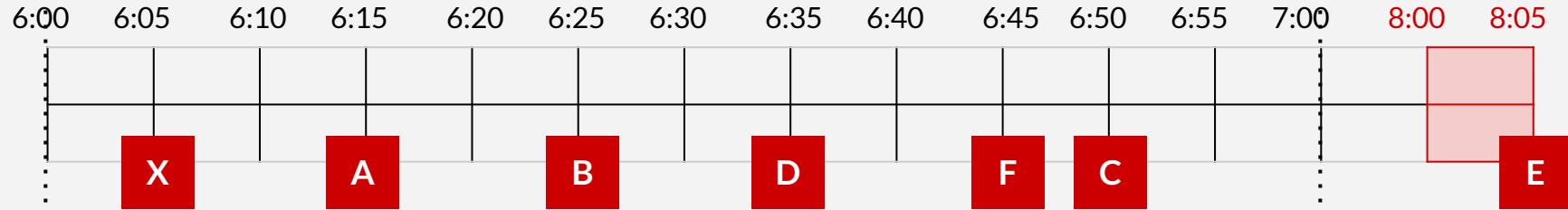
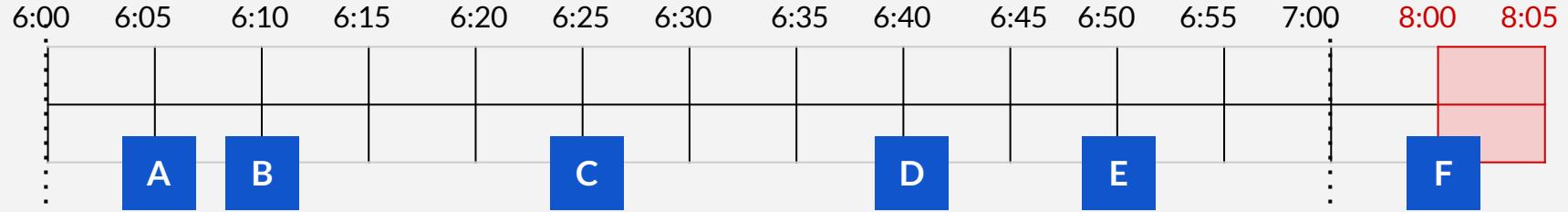


# Join

## Outer Join Stream / Stream



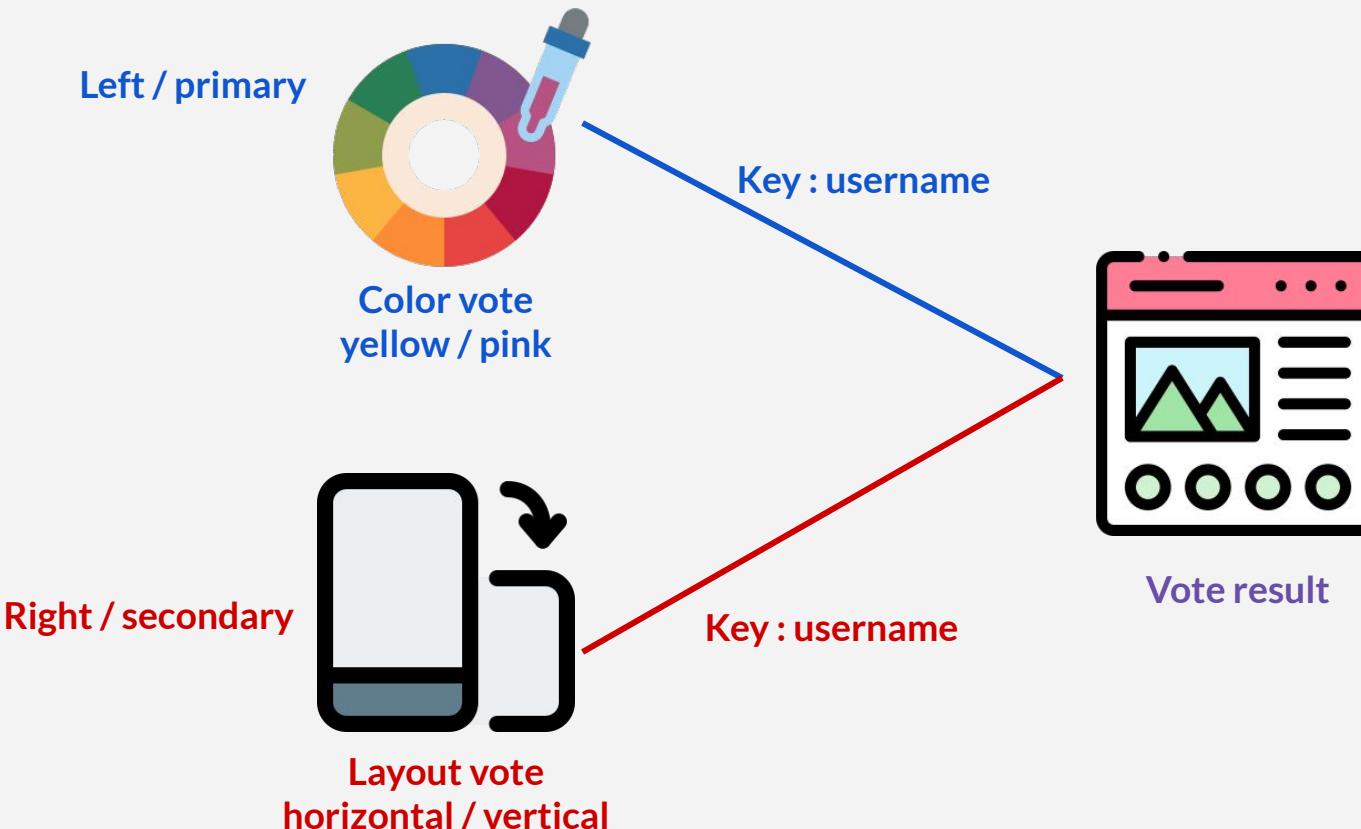
# Outer Join Stream-Stream



# Join Inner Join Table / Table



# Web Color & Layout Vote



# Source Code for Color / Layout Vote

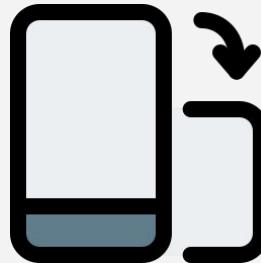
- ✗ `WebColorVote*.java`
- ✗ `WebLayoutVote*.java`
- ✗ Package `com.course.kafka`
  - ✗ `api.request`
  - ✗ `api.server`
  - ✗ `broker.message`
  - ✗ `broker.producer`
  - ✗ `command.action`
  - ✗ `command.service`



# Inner Join



Color vote  
yellow / pink



Layout vote  
horizontal / vertical



Mandatory!



Mandatory!

# Left Join



Color vote  
yellow / pink



Layout vote  
horizontal / vertical



Mandatory!

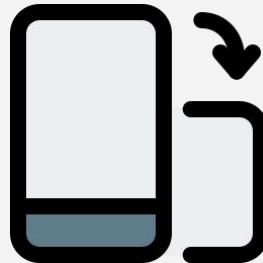


Optional

# Outer Join



Color vote  
yellow / pink



Layout vote  
horizontal / vertical



Optional



Optional



8:05	piglet	pink	
8:25	eyore	yellow	
8:40	piglet		vertical
9:15	pooh	yellow	
9:30	tigger		vertical
9:50	piglet		horizontal
10:00	tigger	pink	
10:05	pooh		vertical
10:40	rabbit		horizontal
11:20	tigger	yellow	

timeline

piglet	pink	vertical	pink : 1, yellow : 0	hor : 0, ver : 1
tigger	pink	horizontal	pink : 1, yellow : 0	hor : 1, ver : 0
tigger	yellow	vertical	pink : 2, yellow : 0	hor : 1, ver : 1
pooh	yellow	vertical	pink : 2, yellow : 1	hor : 1, ver : 2
tigger	yellow	vertical	pink : 1, yellow : 2	hor : 1, ver : 2

# Join Left Join Table / Table





8:05	piglet	pink	
8:25	eyore	yellow	
8:40	piglet		vertical
9:15	pooh	yellow	
9:30	tigger		vertical
9:50	piglet		horizontal
10:00	tigger	pink	
10:05	pooh		vertical
10:40	rabbit		horizontal
11:20	tigger	yellow	

timeline

piglet	pink		pink : 1, yellow : 0	hor : 0, ver : 0
eyore	yellow		pink : 1, yellow : 1	hor : 0, ver : 0
piglet	pink	vertical	pink : 1, yellow : 1	hor : 0, ver : 1
pooh	yellow		pink : 1, yellow : 2	hor : 0, ver : 1
piglet	pink	horizontal	pink : 1, yellow : 2	hor : 1, ver : 0
tigger	pink	vertical	pink : 2, yellow : 2	hor : 1, ver : 1
pooh	yellow	vertical	pink : 2, yellow : 2	hor : 1, ver : 2
tigger	yellow	vertical	pink : 1, yellow : 3	hor : 1, ver : 2

# Join

## Outer Join Table / Table





8:05	piglet	pink	
8:25	eyore	yellow	
8:40	piglet		vertical
9:15	pooh	yellow	
9:30	tigger		vertical
9:50	piglet		horizontal
10:00	tigger	pink	
10:05	pooh		vertical
10:40	rabbit		horizontal
11:20	tigger	yellow	

timeline



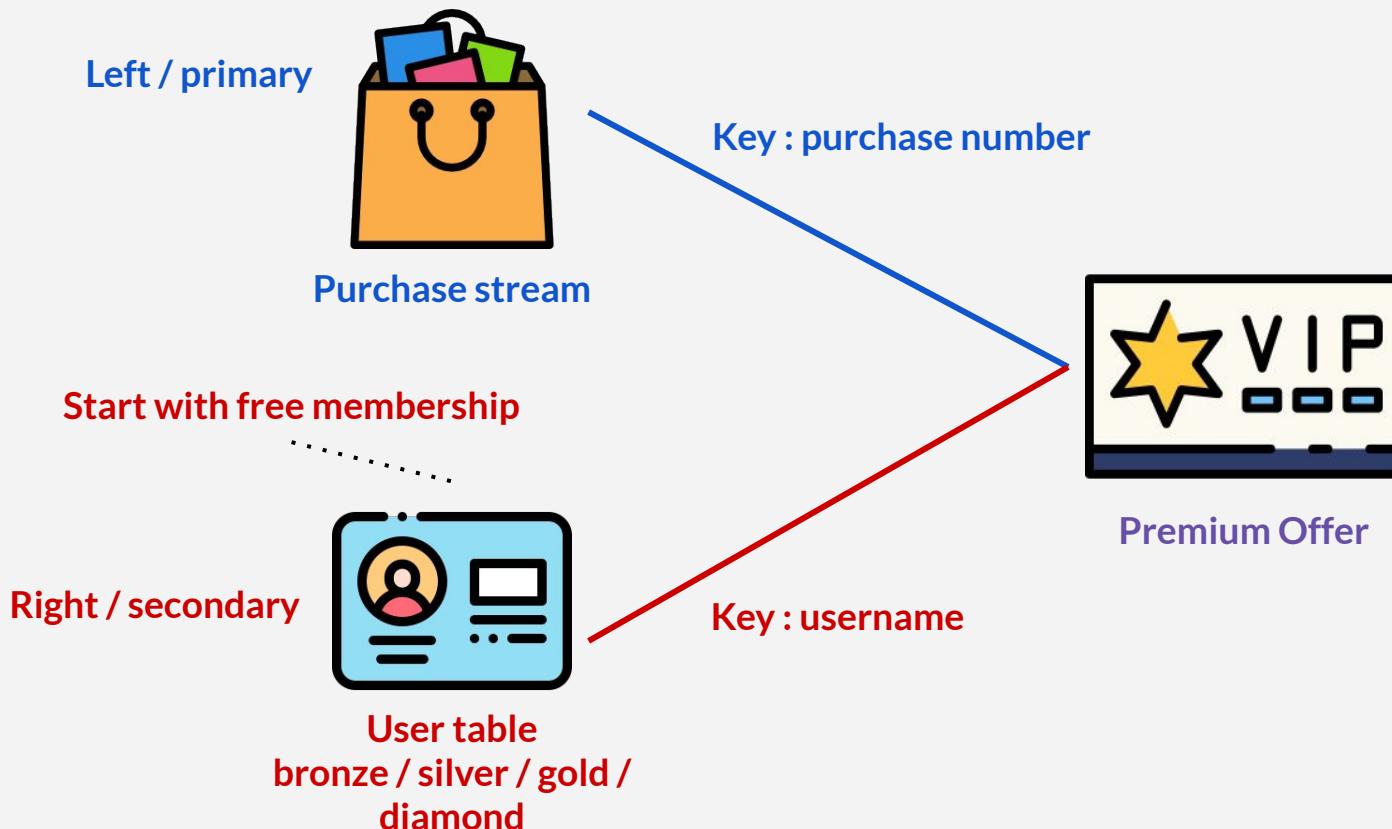
piglet	pink		pink : 1, yellow : 0	hor : 0, ver : 0
eyore	yellow		pink : 1, yellow : 1	hor : 0, ver : 0
piglet	pink	vertical	pink : 1, yellow : 1	hor : 0, ver : 1
pooh	yellow		pink : 1, yellow : 2	hor : 0, ver : 1
tigger		vertical	pink : 1, yellow : 2	hor : 0, ver : 2
piglet	pink	horizontal	pink : 1, yellow : 2	hor : 1, ver : 1
tigger	pink	vertical	pink : 2, yellow : 2	hor : 1, ver : 1
pooh	yellow	vertical	pink : 2, yellow : 2	hor : 1, ver : 2
rabbit		horizontal	pink : 1, yellow : 2	hor : 2, ver : 2
tigger	yellow	vertical	pink : 1, yellow : 3	hor : 2, ver : 2

# Join

## Inner Join Stream / Table



# Premium Purchase & User



# Source Code for Premium

- ✗ **Premium\*.java**
- ✗ Package **com.course.kafka**
  - ✗ **api.request**
  - ✗ **api.server**
  - ✗ **broker.message**
  - ✗ **broker.producer**
  - ✗ **command.action**
  - ✗ **command.service**



# Stream-Table Join Characteristic

- × Stream-Stream / Table-Table is **symmetric**
- × Stream-Table is **asymmetric**
  - × Only left input trigger join
  - × Right input only update table value



1	2	3	4	5	6	7	8	9	10	11	12	13	14
B 21	C 31			B 22	A 11			C 32	B 23		B 24	C 33	C 34

1	2	3	4	5	6	7	8	9	10	11	12	13	14
B, bronze C, gold		B, gold			C, silver			B, silver			A, gold C, diamond		
A, B, bronze C, gold		A, B, gold C, gold			A, B, gold C, silver			A, B, silver C, silver			A, gold B, silver C, diamond		

1	2	3	4	5	6	7	8	9	10	11	12	13	14
		C 31 gold		B 22 gold				B 23 gold				C 34 diamond	

# Join

## Left Join Stream / Table



1	2	3	4	5	6	7	8	9	10	11	12	13	14
	B 21	C 31		B 22	A 11		C 32	B 23		B 24	C 33		C 34

1	2	3	4	5	6	7	8	9	10	11	12	13	14

1	2	3	4	5	6	7	8	9	10	11	12	13	14
	B 21	C 31 gold		B 22 gold	A 11		C 32 silver	B 23 gold		B 24 silver	C 33 silver		C 34 diamond

# Join Stream / GlobalKTable



# Join Combinations

Left Type	Right Type	Co-Partition	Windowed	Join Types		
				Inner Join	Left Join	Outer Join
KStream	KStream	Required	Yes	Supported	Supported	Supported
KTable	KTable	Required	No	Supported	Supported	Supported
KStream	KTable	Required	No	Supported	Supported	Not Available
KStream	GlobalKTable	Not required	No	Supported	Supported	Not Available

# KTable vs GlobalKTable

## KTable

```
var table = builder.table("topic-input").filter(...)
```

## GlobalKTable

```
// create intermediary stream
builder.stream("topic-input").filter(...).to("topic-intermediary");

// build global table
var globalTable = builder.globalTable("topic-intermediary");
```



# KTable vs GlobalKTable

## KTable

```
stream.join(table, joiner)
```

## GlobalKTable

```
stream.join(globalTable, keySelector, joiner)
```

A horizontal timeline consisting of 14 numbered boxes. Boxes 1, 2, 5, 6, 8, 9, 10, 11, 12, and 13 are empty. Boxes 3, 4, 7, 14, and 15 are filled blue and contain labels B21, C31, B22, A11, C32, B23, B24, C33, and C34 respectively.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
		B 21	C 31		B 22	A 11		C 32	B 23		B 24	C 33	C 34

1	2	3	4	5	6	7	8	9	10	11	12	13	14
		C 31 gold		B 22 gold			C 32 gold	B 23 gold		B 24 gold	C 33 gold		C 34 diamond

1	2	3	4	5	6	7	8	9	10	11	12	13	14
	B 21	C 31		B 22	A 11		C 32	B 23		B 24	C 33		C 34

1	2	3	4	5	6	7	8	9	10	11	12	13	14
B, bronze C, gold		B, gold			C, silver			B, silver			A, gold C, diamond		
A, B, C, gold		A, B, gold C, gold			A, B, gold C, gold			A, B, gold C, gold			A, gold B, gold C, diamond		

1	2	3	4	5	6	7	8	9	10	11	12	13	14
		C 31 gold		B 22 gold			C 32 gold	B 23 gold		B 24 gold	C 33 gold		C 34 diamond

# GlobalKTable

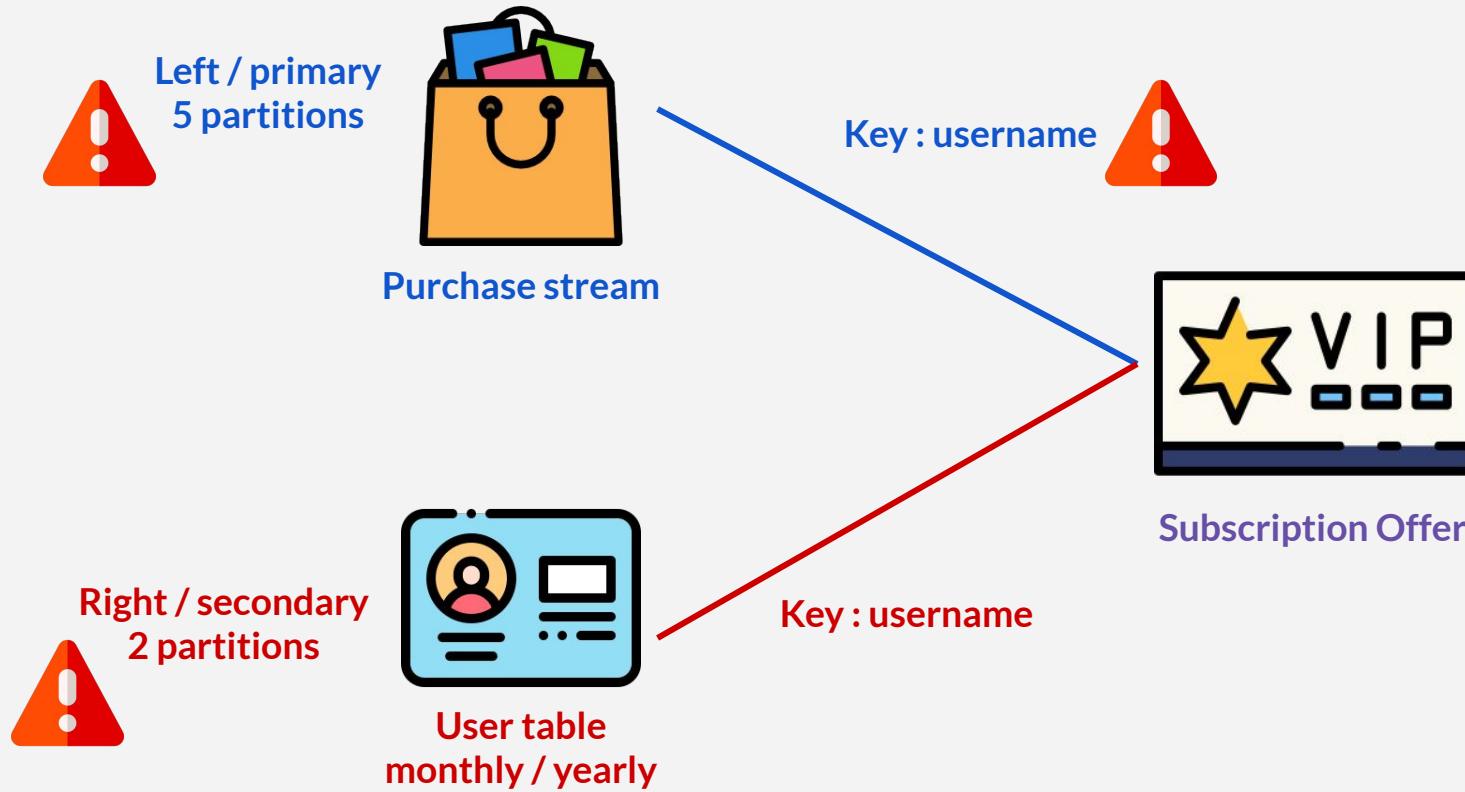
- ✗ Kafka global table is NOT database table
- ✗ Can't filter like SQL:
  - ✗ `SELECT key, value FROM kafka-topic WHERE ...`
- ✗ Latency compared to table
- ✗ Not suitable for frequent updates & a lot of data
- ✗ Frequent? A lot?
  - ✗ Many factors
  - ✗ Configure kafka correctly



# Join Stream / Table Co-Partition



# Subscription Purchase & User



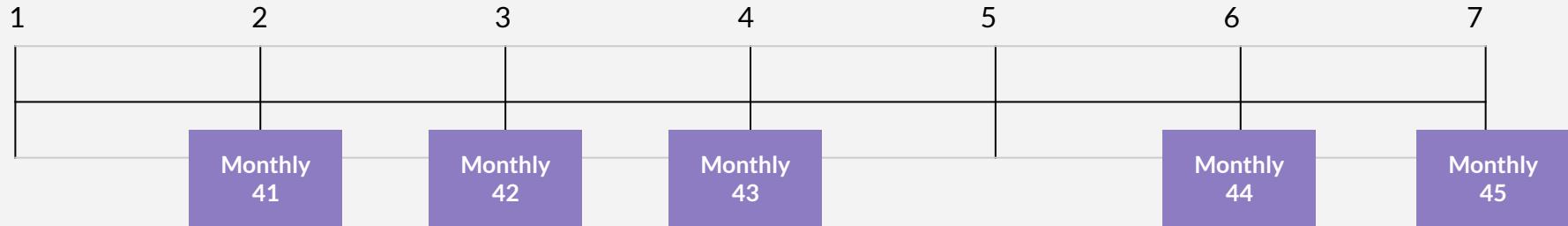
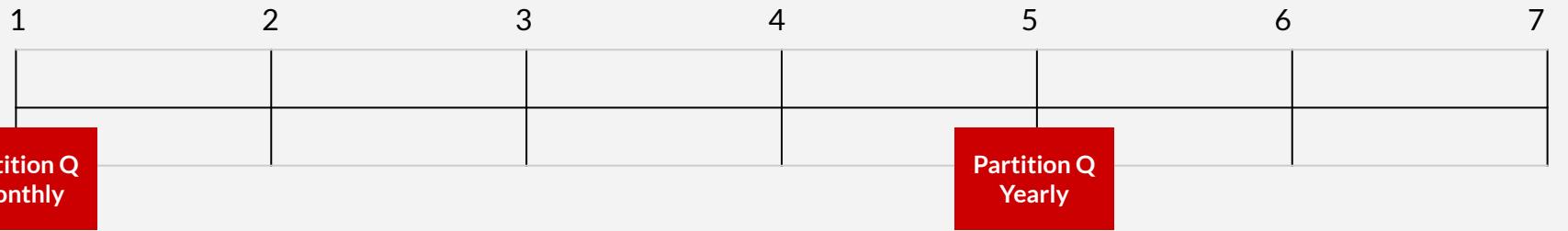
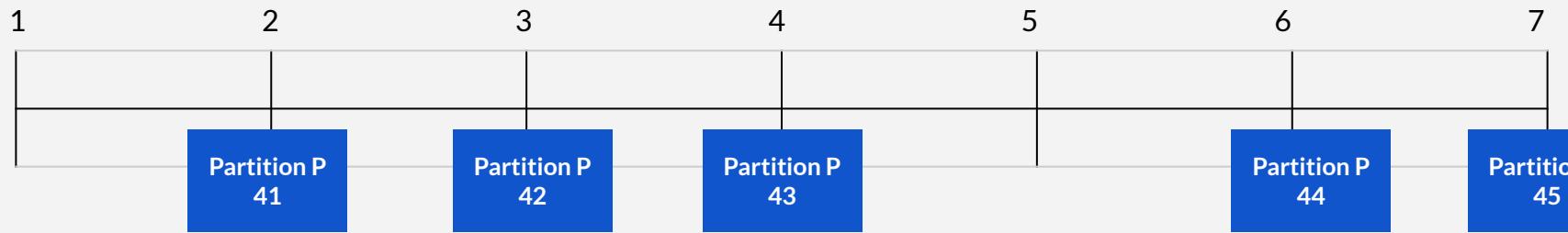
# Source Code for Subscription

- ✗ `Subscription*.java`
- ✗ Package `com.course.kafka`
  - ✗ `api.request`
  - ✗ `api.server`
  - ✗ `broker.message`
  - ✗ `broker.producer`
  - ✗ `command.action`
  - ✗ `command.service`



# Join Stream / GlobalKTable 2





# Exactly Once Semantic



# Delivery Semantic

- × At-most-once, at-least-once, exactly-once
- × Kafka producer or consumer
  - × Configure producer / consumer properties
- × Currently not in course
- × Overview on exactly-once

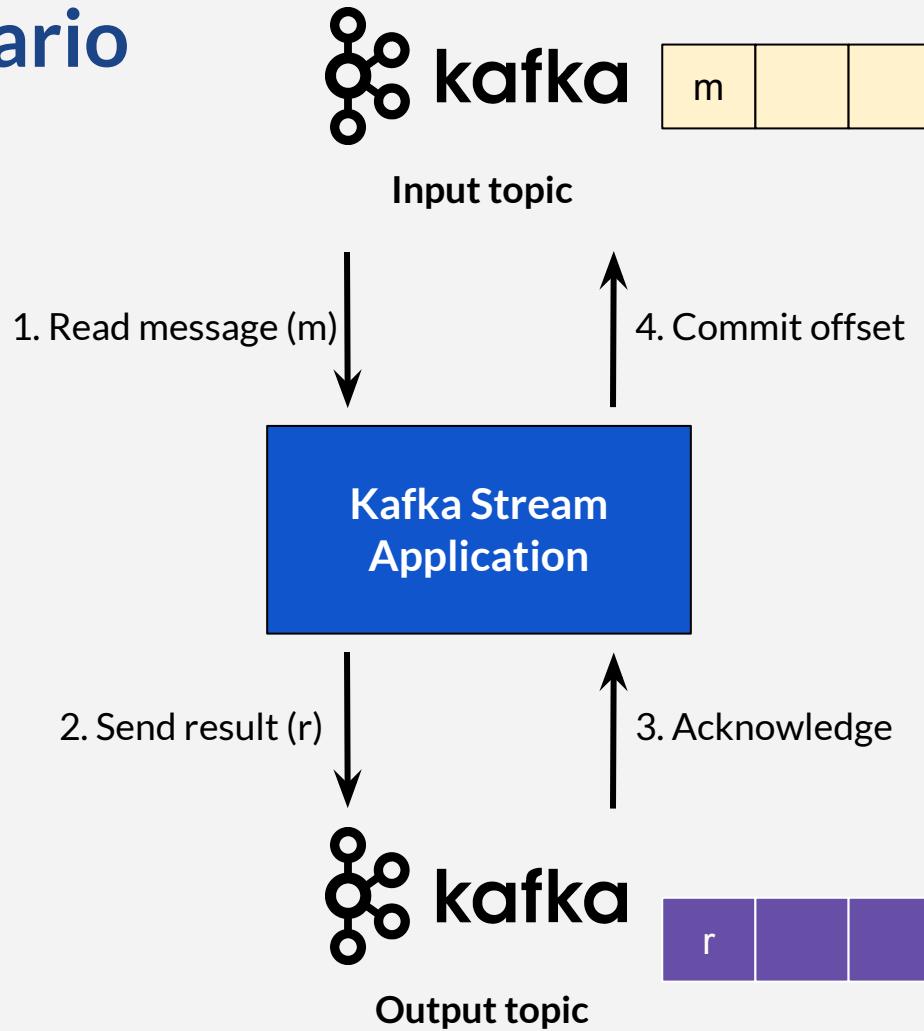


# Exactly Once Semantic

- × Produced only one copy, and consume only once
- × Retry on producer (send multiple times) can still be happened
- × Kafka will deduplicate messages
- × Kafka as input and output, without involving external system (database, text file, etc)
- × Match for kafka stream



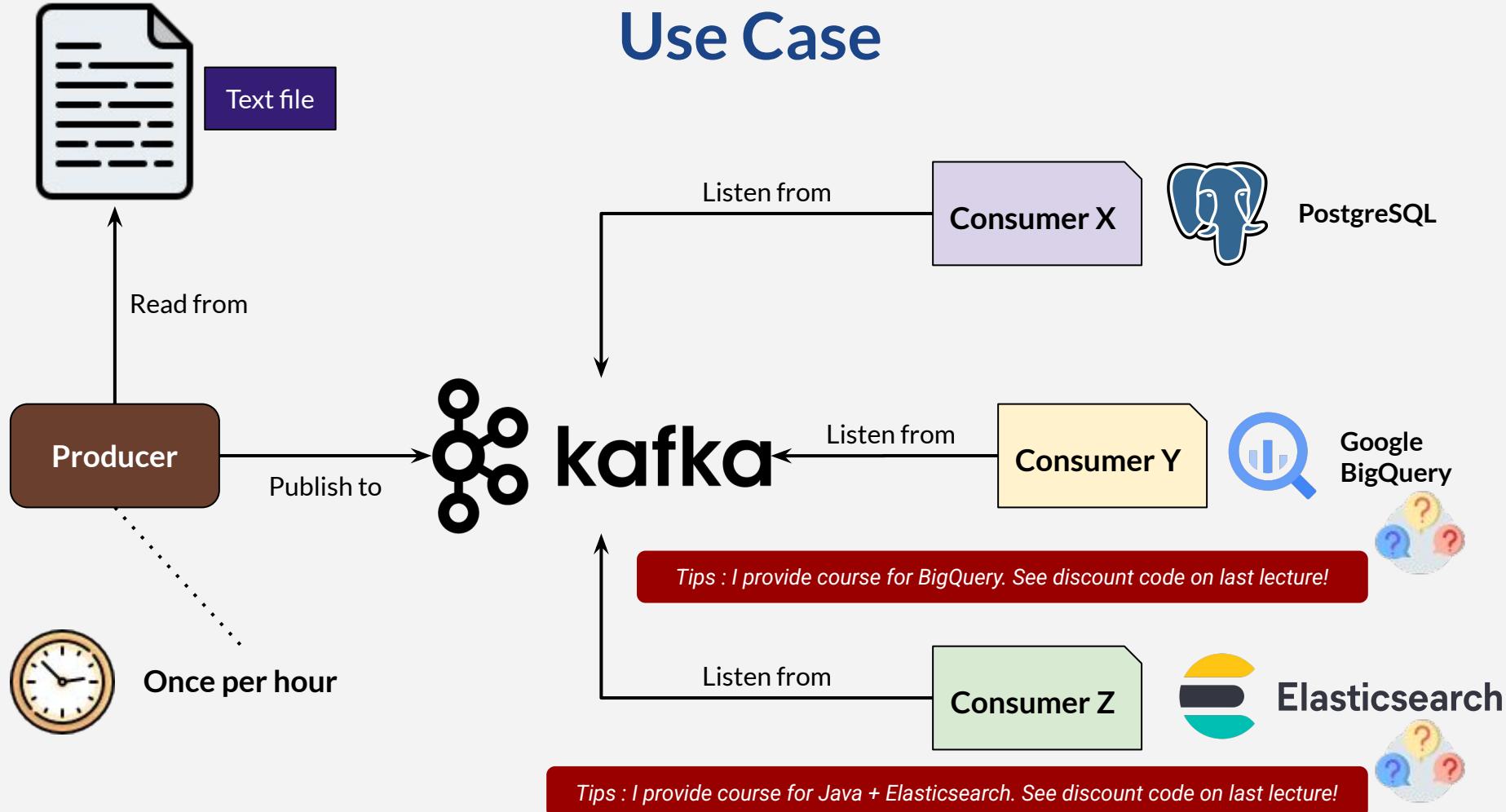
# Happy Scenario



# Message In, Message Out



# Use Case



# Connectors

- × Java jar file
- × Plugin for kafka connect
- × Interface between kafka and non-kafka
- × **Source** connector : read (ingest) into kafka (**producer**)
- × **Sink** connector : write from kafka to non-kafka (**consumer**)
- × Install connector for specific need
- × Write configuration (json)
- × Declarative configuration
- × Fast & reliable



# Connectors

- × A lot of source / sink connectors
- × Shorten time and effort

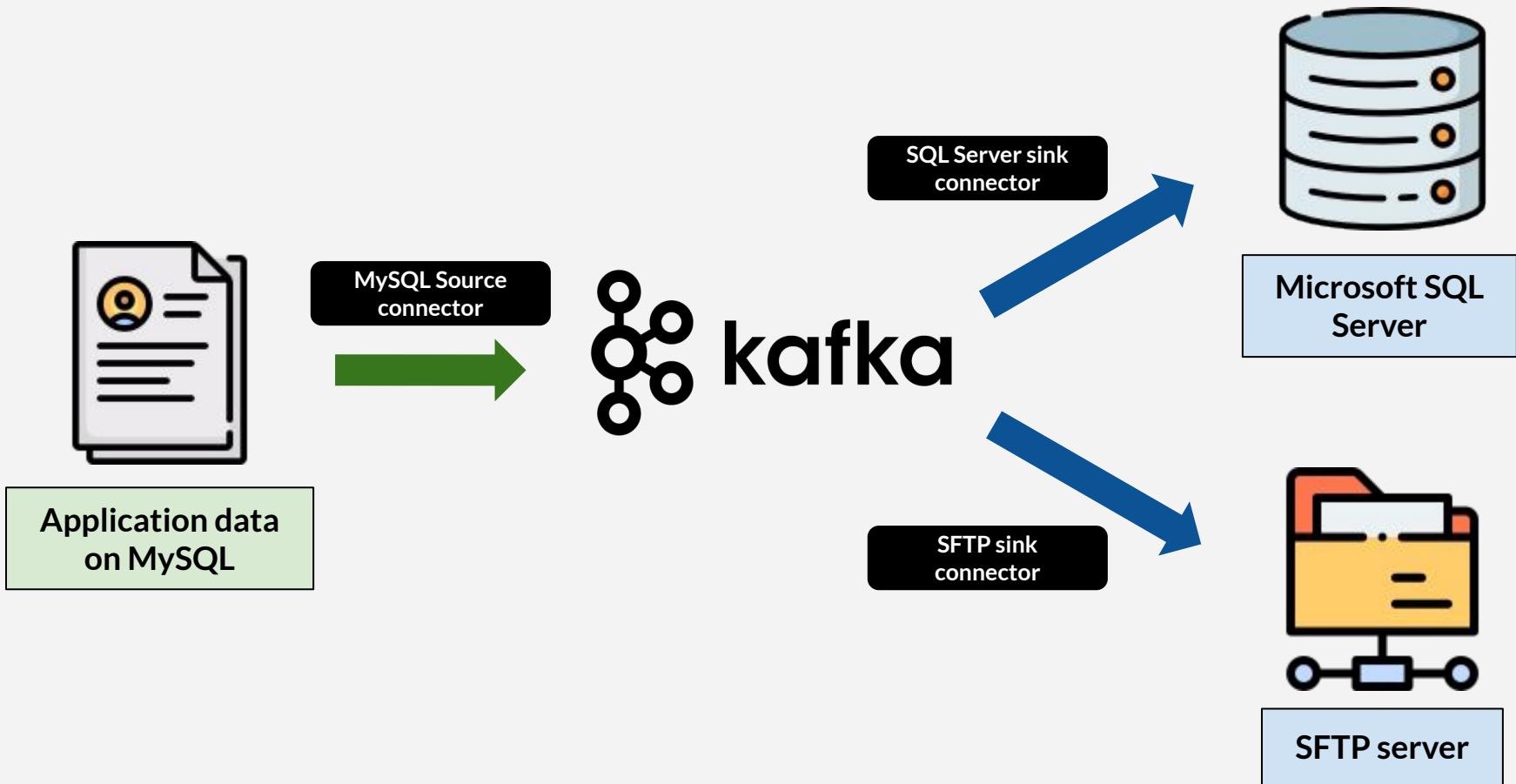


# How to Get Connectors?

- × Curated list on [confluent.io/hub](https://www.confluent.io/hub)
- × Google : *kafka source / sink connector for xxx*
- × Build your own



# Use Case : Write to Data Stores

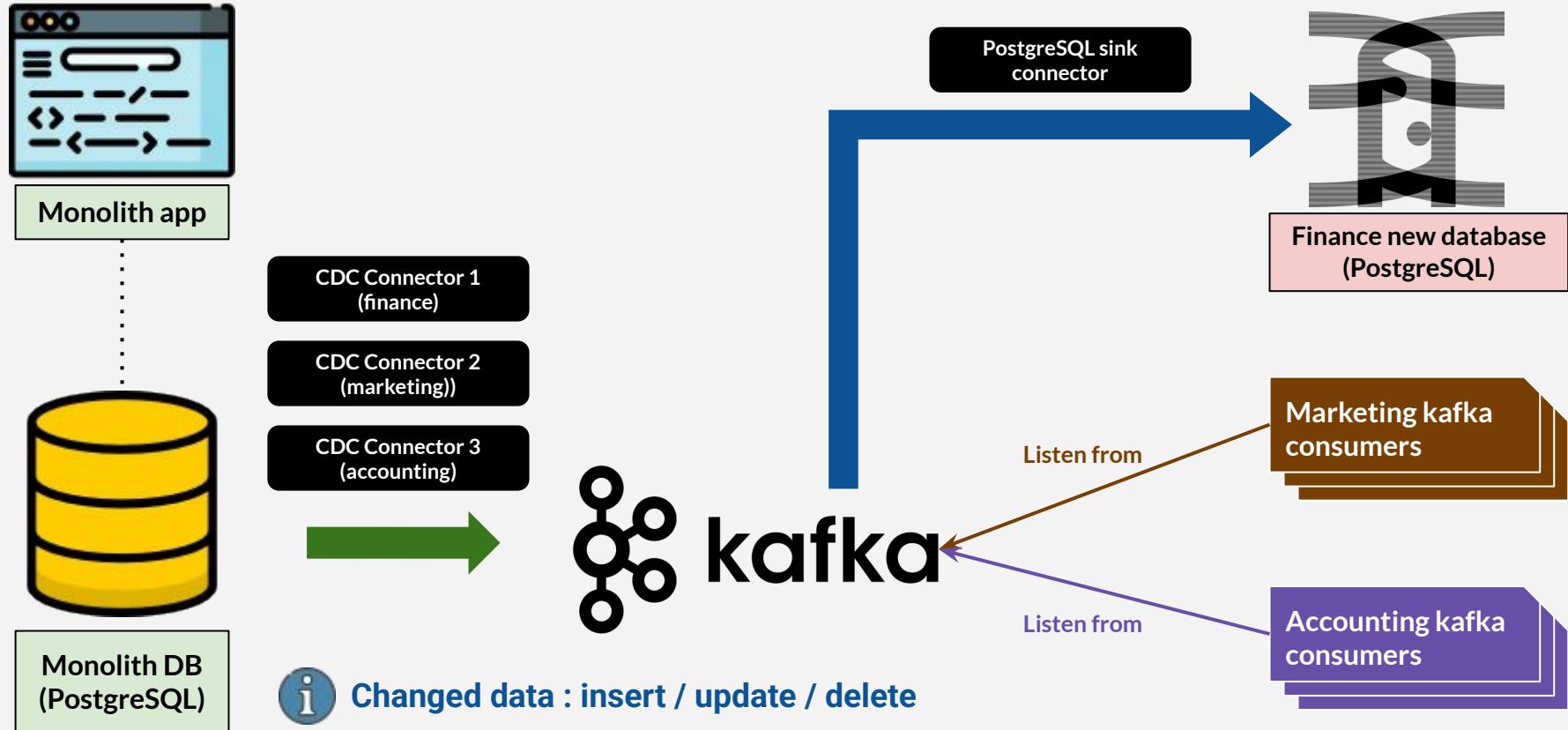


# Use Case : Modernize Legacy System

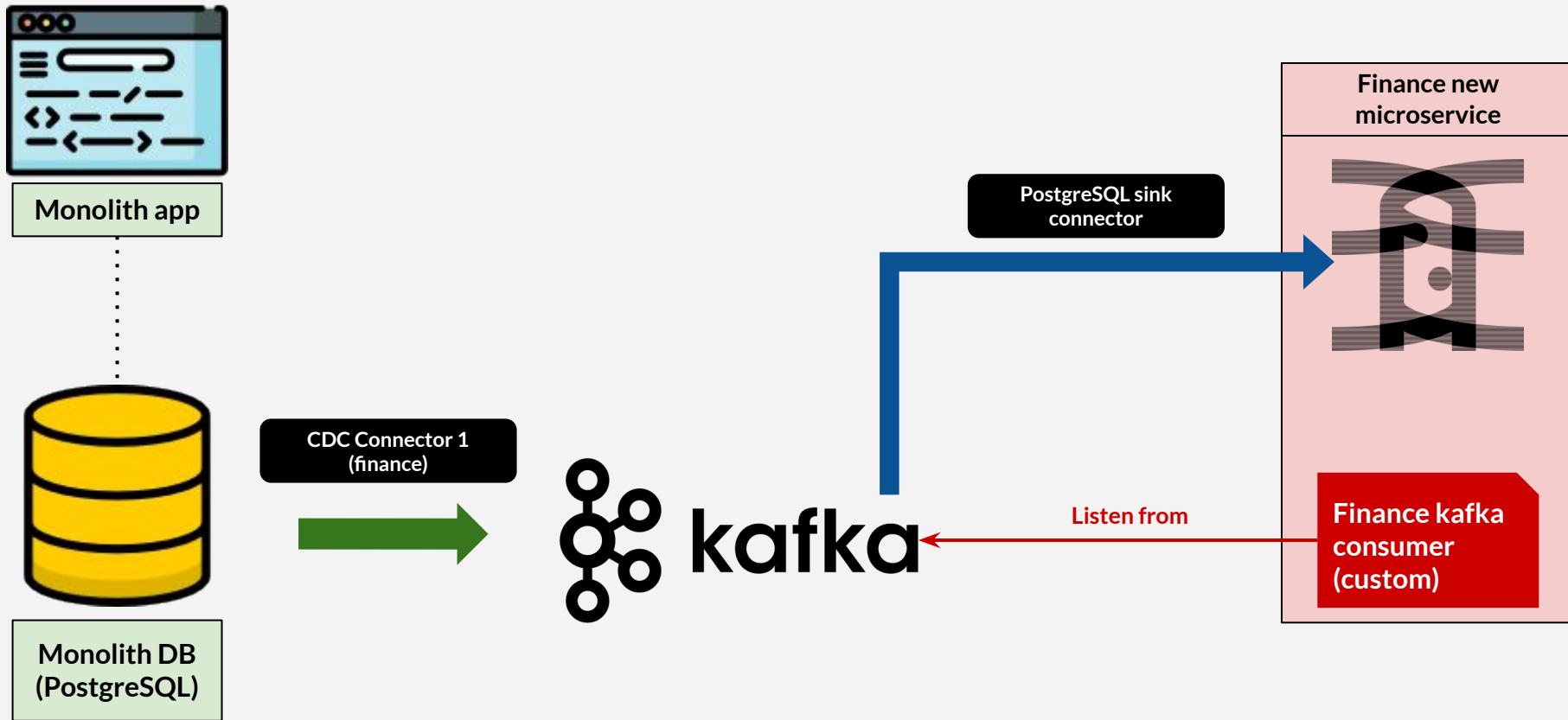
- × Modernize legacy monolith into microservices
- × Modernization is hard and long
- × Legacy system still needs to run during modernization
- × Modernize functionalities part by part
- × Use kafka connect CDC (Change Data Capture) connectors
- × *Microservice Architecture & Pattern course (and discount) available on last lecture of this course*



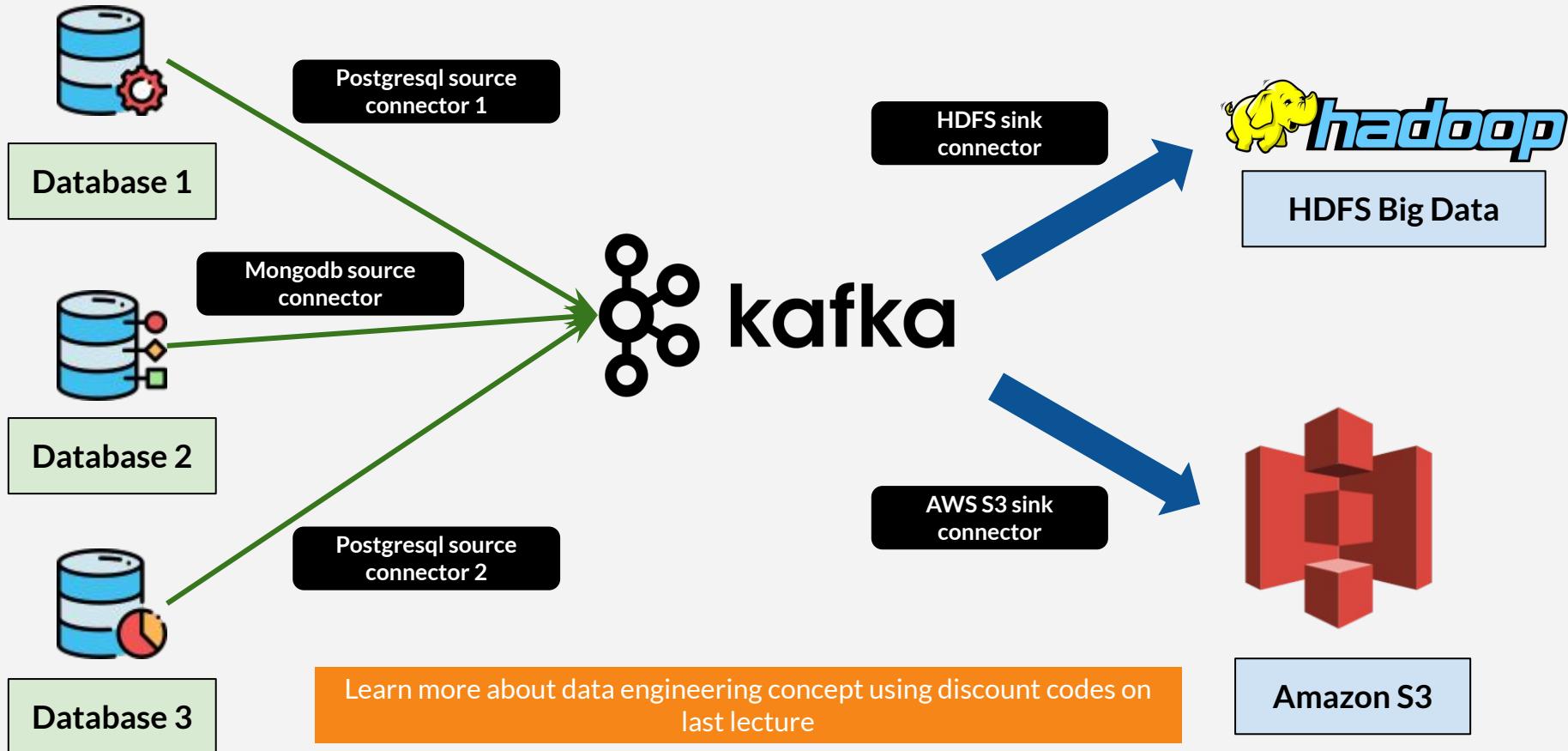
# Use Case : Modernize Legacy System



# Use Case : Modernize Legacy System



# Use Case : Data Engineering ETL Pipeline



# Kafka Connect

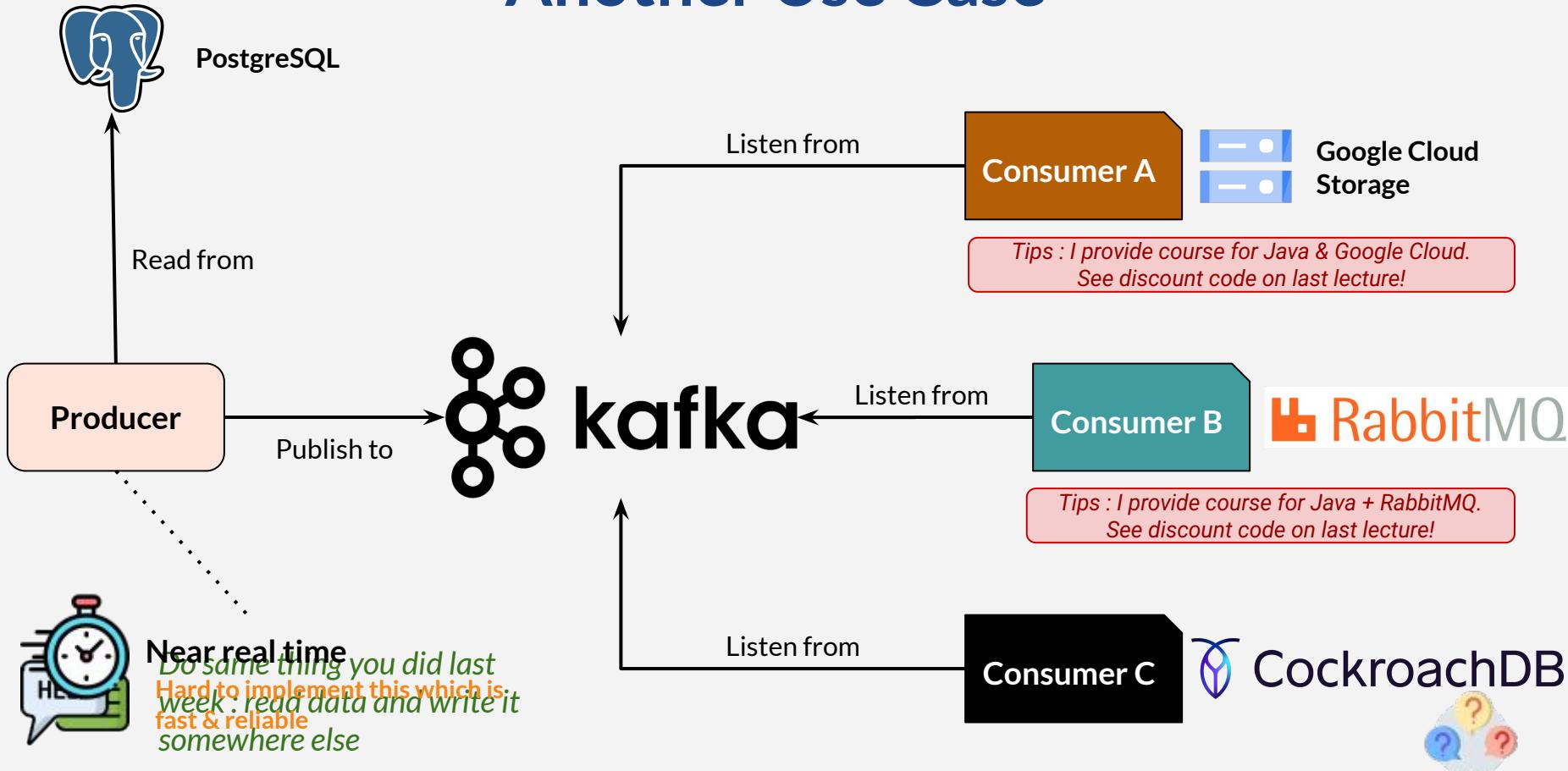
- × Cluster can has one or more workers (servers)
- × Connector + user configuration = **task**
- × A user configuration might spawn one or more task
- × Worker will execute task



# Kafka Connect on Docker



# Another Use Case



```
#> docker-compose -f [script-file] -p [project] down
```

Part	What to run (in sequence)
1 - core kafka	#> docker-compose -f docker-compose-core.yml -p core up -d
2 - kafka connect	#> docker-compose -f docker-compose-core.yml -p core down #> docker-compose -f docker-compose-connect.yml -p connect up -d #> docker-compose -f docker-compose-connect-sample.yml -p connect-sample up -d
3 - kafka full	#> docker-compose -f docker-compose-connect.yml -p connect down #> docker-compose -f docker-compose-connect-sample.yml -p connect-sample down #> docker-compose -f docker-compose-full.yml -p full up -d #> docker-compose -f docker-compose-full-sample.yml -p full-sample up -d

# Sample Use Cases



# Use Cases

- × Sample use cases
  - × Basic connector usage
  - × Legacy modernization with CDC
  - × Data engineering
- × **Note**
  - × Need at least 4 GB free memory for docker
  - × Simple use cases only
  - × Data validity is not a concern
  - × Temporary data (stored on docker containers)



# Sample Data

- × SFTP tool in this course : *Filezilla*
- × PostgreSQL tool in this course : *Dbeaver*
- × Feel free to use your own tools
- × Available on Resource & Reference
- × Sample data generated using [mockaroo.com](https://mockaroo.com)
- × Mockaroo.com schema available on Resource & Reference



# Use Case : Basic Connector File Source





CSV source  
connector



kafka

employee-\* .csv

```
employee_id,first_name,last_name,email,gender,birth_date,salary
79-8035819,Hanny,Stubs,hstubs0@github.com,F,1996-02-02,7061
24-9676514,Val,Bonnick,vbonnick1@icq.com,M,1991-09-11,2657
61-3900911,Juanita,Lanfranconi,jlanfranconi2@rediff.com,F,1992-07-08,7241
28-0495338,Elsie,Brownlea,ebrownlea3@theglobeandmail.com,F,1985-11-03,8026
13-9980006,Chris,Kettlestringes,ckettlestringes4@indiegogo.com,M,1994-12-14,1015
```

```
{  
  "schema": {  
    "type": "data type",  
    "fields": {  
  
    }  
  },  
  "payload": {  
    "field1": "value 1",  
    "field2": "value 2"  
  }  
}
```

**Embedded schema**  
Automatically generated by source connector  
Required by most sink connector

# Schema

- × Contract
- × Set of rules for message, to be obeyed
- × Field name, data type, mandatory / optional
- × Kafka connect able to work with dynamic data structure
- × Need to inform data structure (schema)
- × Kafka source connector auto generate schema



# Use Case : Basic Connector Database Sink





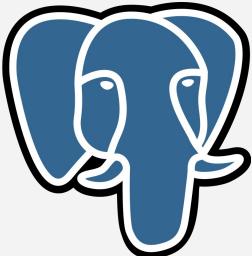
employee-\* .csv

CSV source  
connector



kafka

PostgreSQL sink  
connector



PostgreSQL

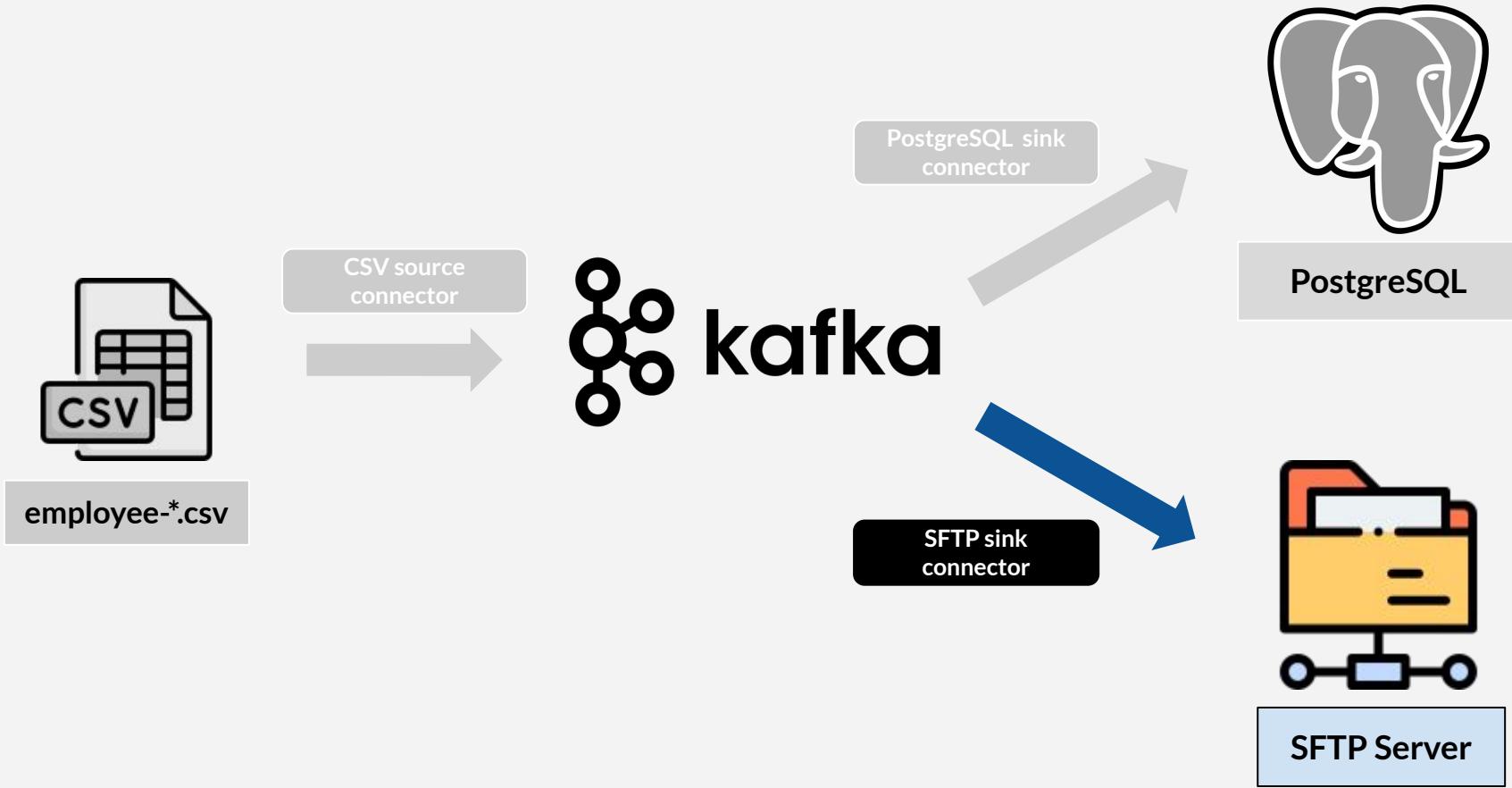
# Message In, Message Out

- × Happens in real life
- × Today we have multiple data sources (**in**)
  - × text file, relational databases
  - × Non relational database, email, cloud, social media, specific software, etc
  - × Can be more than 100
- × Multiple target data store (**out**)
  - × local text file, FTP, relational / non-relational databases, big data, cloud, etc



# Use Case : Basic Connector **SFTP Sink**





# Use Case : Legacy Modernization Change Data Capture



# Legacy Modernization

- ✗ Existing (legacy) application without kafka
- ✗ Rebuild into microservice with kafka
- ✗ Two systems (legacy & new) running together
- ✗ Synchronize data change (new, update, delete)
- ✗ Data source is from legacy application
- ✗ How?



# Change Data Capture - 1st Way

- × Scheduler(hourly) that query data from legacy database based on *last\_updated\_date* field
- × But
  - × Not every table has *last\_updated\_date* field
  - × One hour is too long, need low latency (e.g. 60 seconds)
  - × Scheduler with short delay might problematic

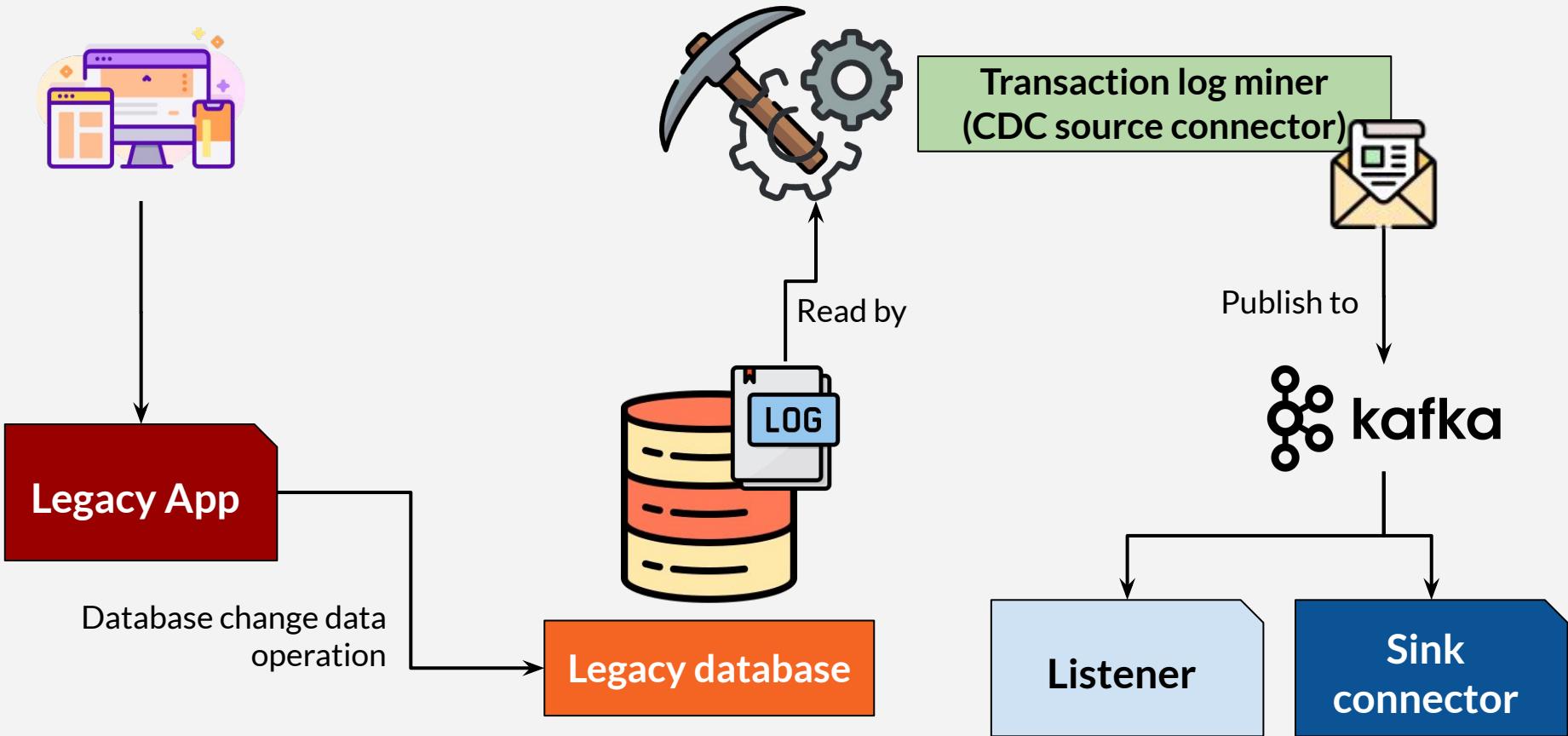


# Change Data Capture - 2nd way

- × Each data change directly sent to kafka
- × Coding kafka integration points in legacy application is painful & not worth the effort
- × Good if we can implement this, but with minimal effort
- × Possible without writing code
- × Database transaction log tailing
- × Kafka connect with CDC (change data capture) source connector



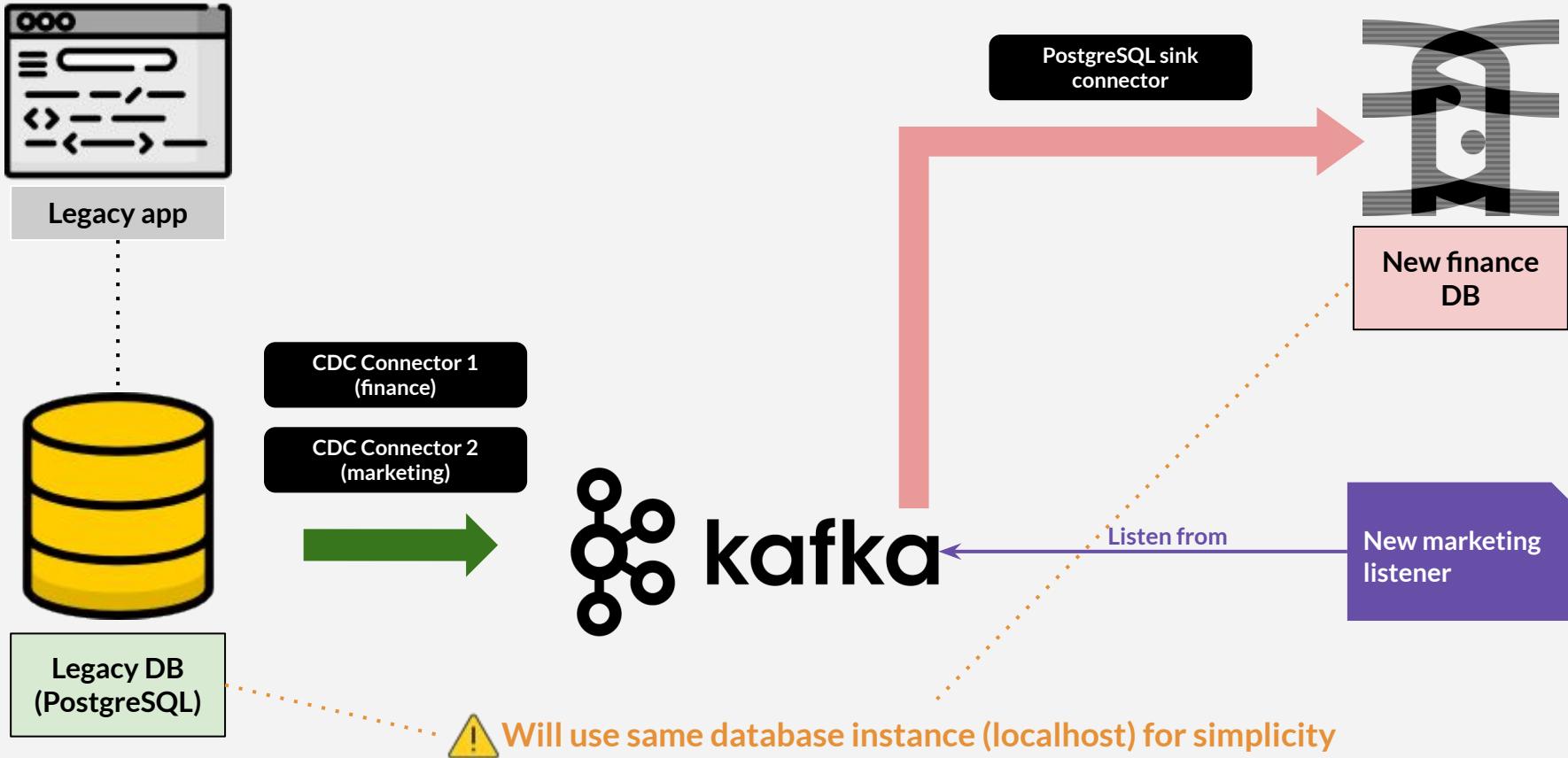
# Transaction Log Tailing



# Microservice Architecture & Pattern

- × Transaction log tailing is just one pattern to ease your life
- × Available in my course **Microservice Architecture & Pattern with Java & Kafka**
- × Grab discount code on last section of this course





# Use Case : Legacy Modernization

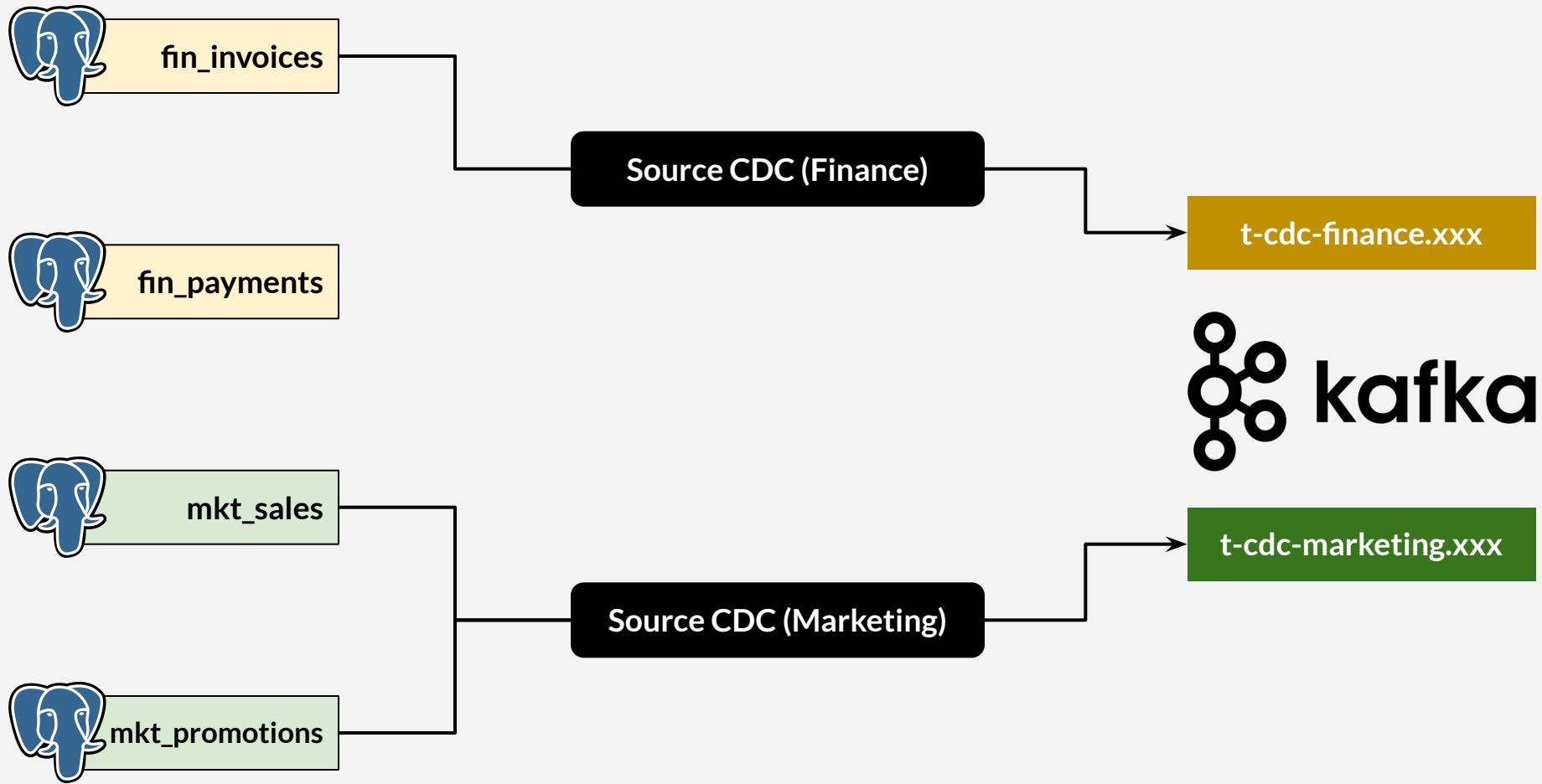
## CDC PostgreSQL Connector



# On Kafka

- × Write a lot of producers and consumers
- × Extra time & effort for performance & reliability
- × Good news : ***you don't have to write your own!***
- × People / companies already wrote them
- × Read plugin : from non-kafka into kafka
- × Write plugin : from kafka into non-kafka
- × **Kafka Connect**





# Use Case : Legacy Modernization

## PostgreSQL Sink Connector



# Use Case : Legacy Modernization Marketing Consumer



# Spring Initializr

- × [start.spring.io](https://start.spring.io)
- × Java project with gradle
- × Group : **com.course.kafka**
- × Artifact : **kafka-connect-sample**
- × Package name : **com.course.kafka**
- × Dependency : **Spring Kafka, Spring Kafka Stream**



# Json Deserializer?

- × Why use default (String deserializer)
- × JSON (De)serializer is part of Spring
- × Spring specific mechanism to (de)serialize
- × Works out-of-the-box if publisher uses Spring JSON serializer
- × CDC connector is **not** part of spring framework
- × Process JSON string manually



# It's not A One-Stop-Solution



# Kafka Connect

- × Basic use case works
- × Complex use case needs more effort
- × In real life:
  - × Kafka connect can helps
  - × But it's not a magical problem solver
- × Need proper Kafka Connect configuration



# Kafka Connect

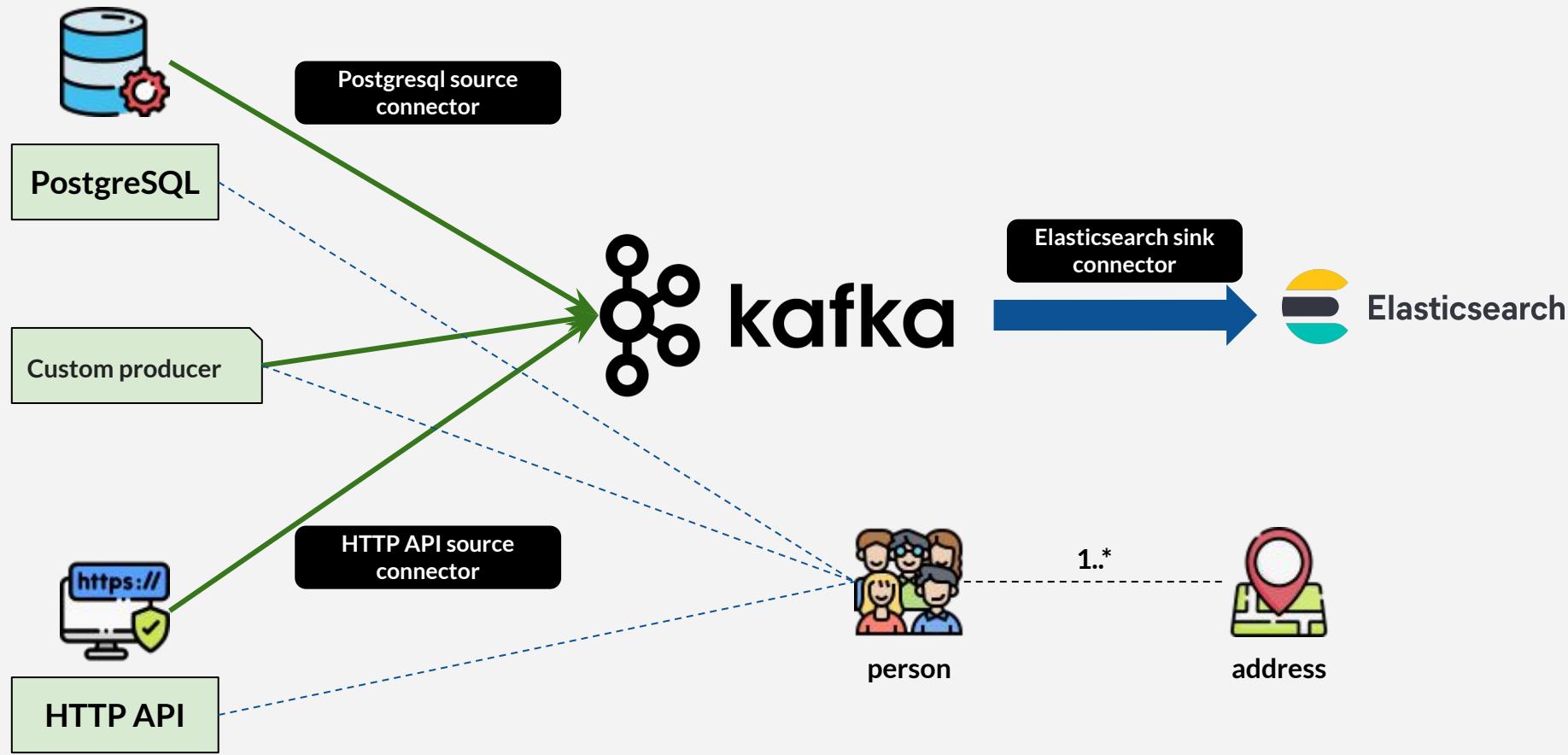
- ✗ Not entirely without cost
- ✗ Connector might need license
- ✗ Example : cloud storage
- ✗ Amazon / Google / Azure



# Use Case : Data Engineering Database Source



# Use Case : Data Engineering ETL Pipeline



# Introducing Kafka Connect



# Use Case : Data Engineering HTTP Source



# Use Case : Data Engineering Custom Source



# Data Format Differences



## Database

```
{  
  "schema":{  
    ...  
  },  
  "payload":{  
    "person_id":2100,  
    "id_card_number":"401-99-8581",  
    "full_name":"Hamlin Kayes",  
    "email":"hkayes2r@bigcartel.com",  
    "address":"0 Westerfield Place",  
    "city":"Sājūr",  
    "postal_code":null  
  }  
}
```

*Schema + payload  
Actual data in payload  
Using snake\_case  
Single address*

## HTTP

```
{  
  "schema":{  
    ...  
  },  
  "payload":{  
    "value":{  
      "id_card_number":"444-25-9069",  
      "full_name":"Rebekkah Toller",  
      "email":"rtoller8@ezinearticles.com",  
      "addresses":{  
        "address_id":6710,  
        "address":"5 Lillian Parkway",  
        "city":"Lyubim",  
        "postal_code":152470},  
        "address_id":7860,  
        "address":86278 Calypso Trail,  
        "city":Gaotieling,  
        "postal_code":null  
      },  
      "key":9d38e4bf-3b1d-32d9-b78b-7a8135233597,  
      "timestamp":1644280143619  
    }  
  }  
}
```

*Schema + payload  
Actual data in payload.value  
Using snake\_case  
Array address*

## Custom

```
{  
  "personId":84527,  
  "fullName":"Joe Auer",  
  "email":"joeauer99@hotmail.com",  
  "addresses": [  
    {  
      "addressId":64288,  
      "address":"4639 Everett Walks",  
      "city":"Stacyfort",  
      "postalCode":null  
    }  
  ]  
}
```

*No schema, no payload  
Represents actual data  
Using camelCase  
Array address*

## Database

```
{  
  "schema": {  
    ...  
  },  
  "payload": 3102  
}
```

## HTTP

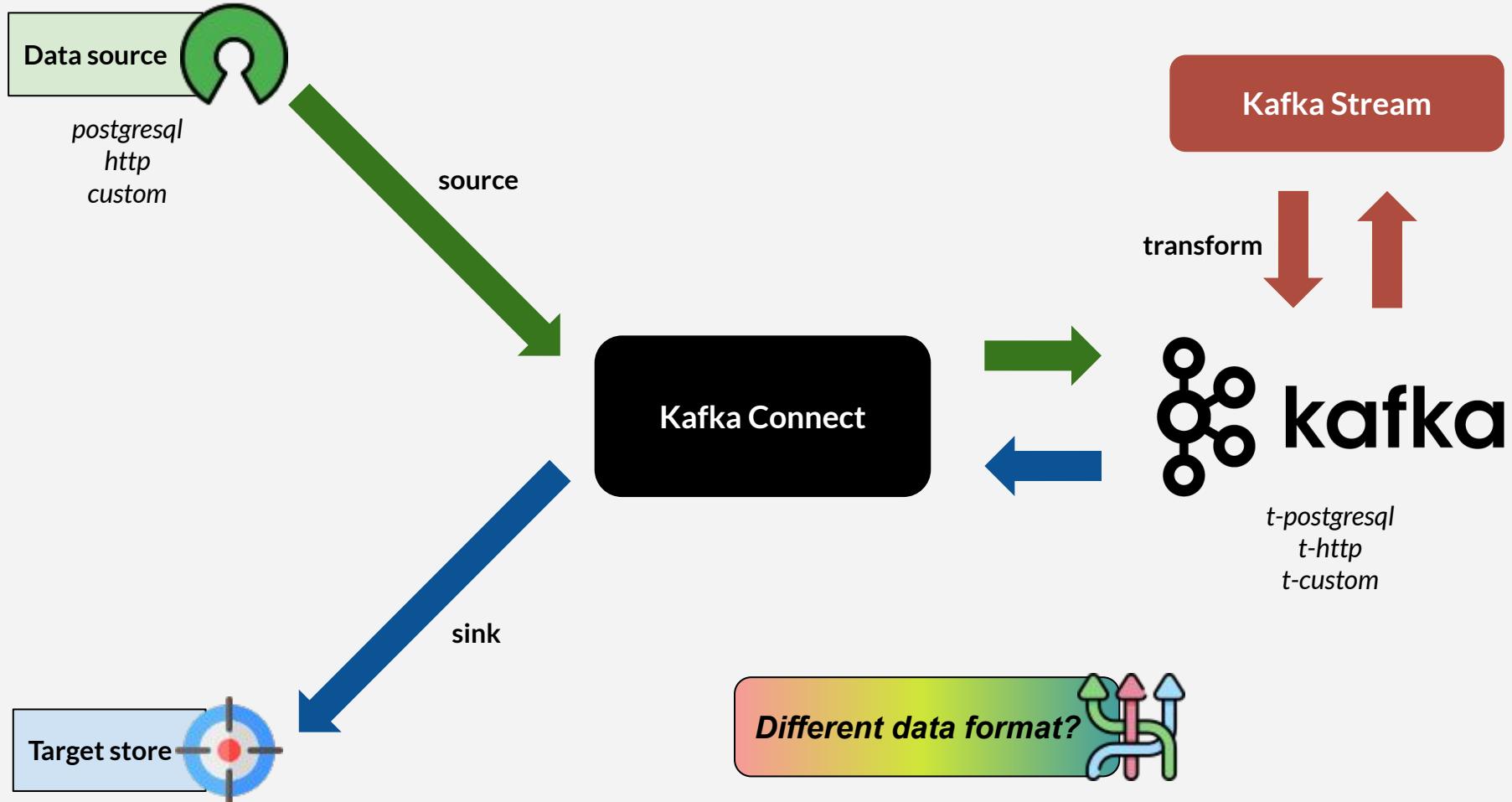
```
{  
  "schema": {  
    ...  
  },  
  "payload": {  
    "key": "62aeb383-539a-3ea5-99cb-a16fb4b3681b"  
  }  
}
```

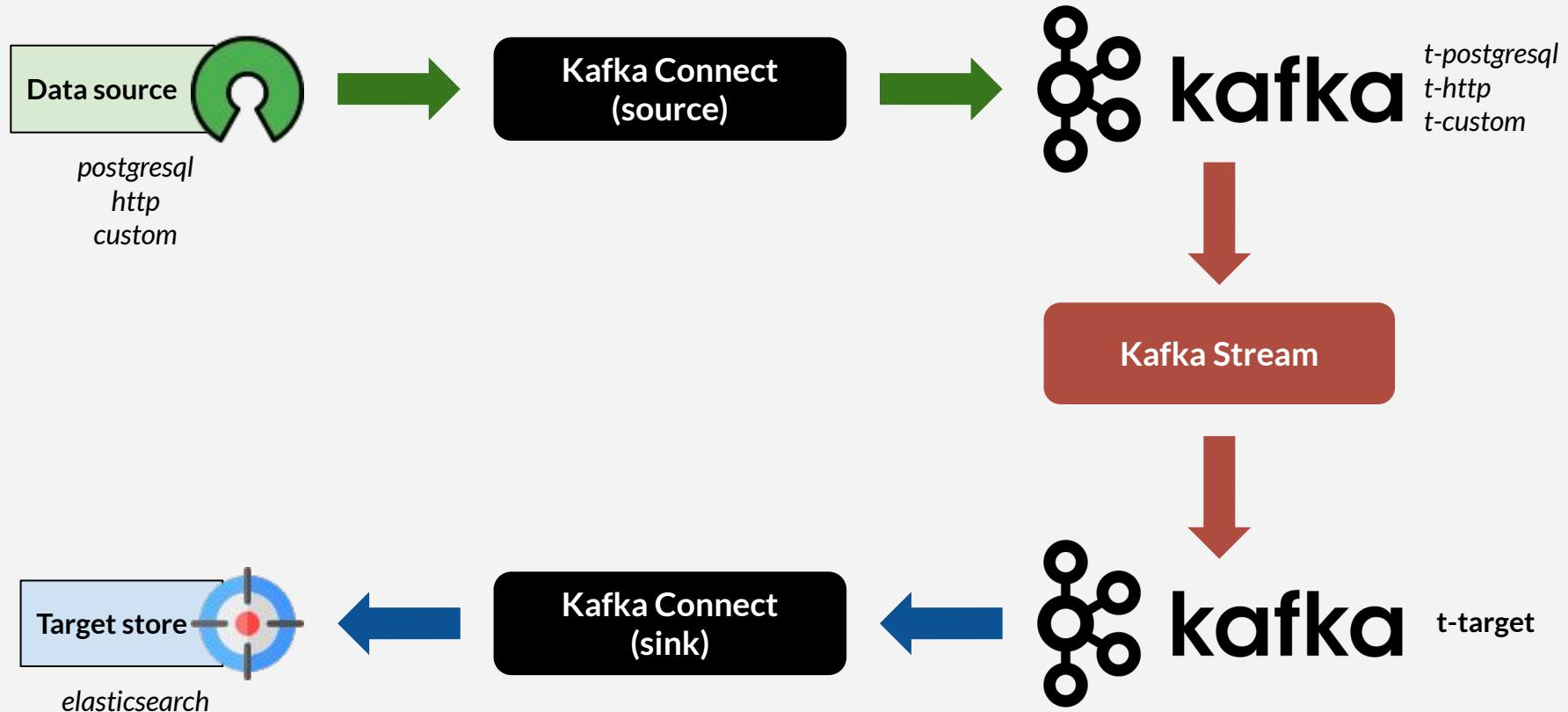
*Schema + payload  
Actual data in payload.key  
Random UUID*

## Custom

*Schema + payload  
Actual data in payload  
Address ID*

*No key*





# Kafka Stream & Kafka Connect Code Overview

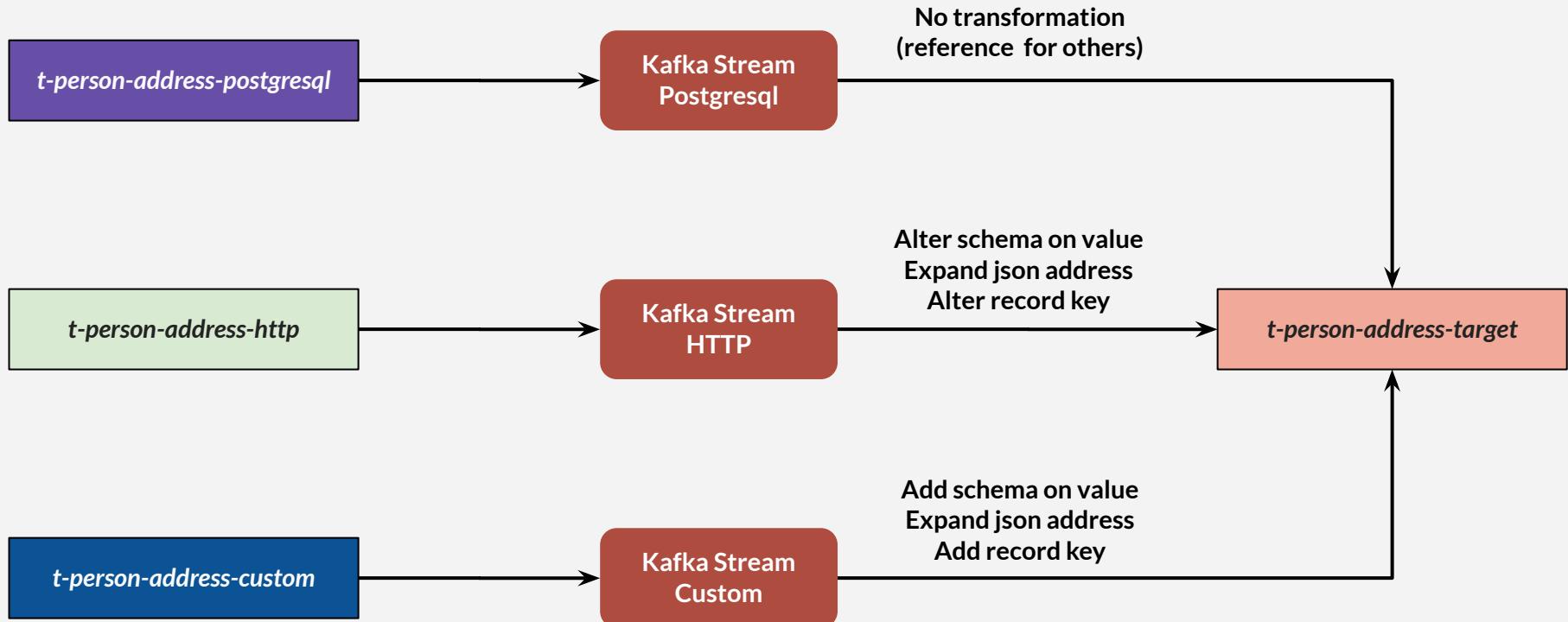


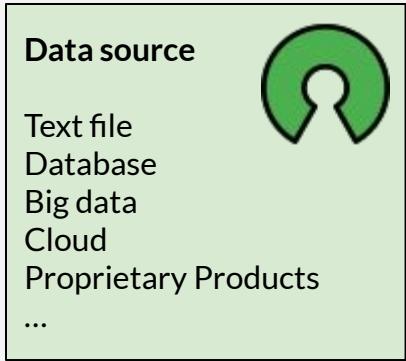
# Kafka Stream & Kafka Connect

- × Source topics (from source connectors)
  - × *t-person-address-postgresql*
  - × *t-person-address-http*
  - × *t-person-address-custom*
- × Target topic (for sink connector)
  - × *t-person-address-target*
- × Use java generic coding practice
- × String serde for key and value



# Kafka Stream Transformation





# All

<i>Topic</i>	<i>Class</i>	<i>Description</i>
all	<b>message.KafkaConnectMessage</b>	Wrapper class for kafka connect message that contains <b>schema</b> and <b>payload</b> . Payload uses java generic.
	<b>schema.KafkaConnectSchema</b>	Represents <b>schema</b> on <b>KafkaConnectMessage</b>

# PostgreSQL Source

Topic	Class	Description
<i>t-person-address-postgresql</i>	-	No need to define class representing message. This is the message format we will use on target topic.
	<code>stream.PersonAddressFromPostgresqlStream</code>	<p>Kafka stream:</p> <ol style="list-style-type: none"><li>1. Take data from <i>t-person-address-postgresql</i></li><li>2. Send as is, to <i>t-person-address-target</i></li></ol> <p>No transformation needed, since the record format (key &amp; value) is what we expect.</p>

# HTTP Source

Topic	Class	Description
<i>t-person-address-http</i>	<code>message.KafkaConnectPersonAddressFromHttpMessage</code>	Represents message retrieved by HTTP source connector (contains full json string on <code>payload.value</code> )
	<code>message.KafkaConnectPersonMessageSnakeCase</code> <code>message.KafkaConnectAddressMessageSnakeCase</code>	Holder for converting json string in <code>payload.value</code> into java object (the actual person & address data). Fields are <code>snake_case</code> .
	<code>stream.PersonAddressFromHttpStream</code>	Kafka stream: <ol style="list-style-type: none"><li>1. Take data from <i>t-person-address-http</i></li><li>2. Transform (expand json <code>addresses</code> and convert each address into one record). Alter schema on record key &amp; value</li><li>3. Send to <i>t-person-address-target</i></li></ol>

# Custom Source

Topic	Class	Description
<i>t-person-address-custom</i>	<code>message.PersonMessage</code> <code>message.PersonAddress</code>	Represents message published by dummy scheduler. Fields are camelCase.
	<code>stream.PersonAddressFromCustomStream</code>	Kafka stream: <ol style="list-style-type: none"><li>1. Take data from <i>t-person-address-custom</i></li><li>2. Transform (expand json <b>addresses</b> and convert each address into one record). Add schema on record key &amp; value</li><li>3. Send to <i>t-person-address-target</i></li></ol>

# Target Sink (From All Sources)

Topic	Class	Description
<i>t-person-address-target</i>	<code>message.KafkaConnectPersonTargetMessage</code>	<p>Represents <b>payload</b> on record value (class <code>KafkaConnectMessage</code>) to be published to <i>t-person-address-target</i>. This actually the same format as data published by JDBC source connector at <i>t-person-address-postgresql</i></p>
	<code>schema.KafkaConnectPersonAddressTargetKeySchema</code>	<p>Singleton for creating <b>schema</b> on record key to be published to <i>t-person-address-target</i>.</p>
	<code>schema.KafkaConnectPersonAddressTargetValueSchema</code>	<p>Singleton for creating <b>schema</b> on record value to be published to <i>t-person-address-target</i>.</p>

# Kafka Stream & Kafka Connect Code Hands on



# Tip : Override Converter



# Use Case : Data Engineering Elasticsearch Sink



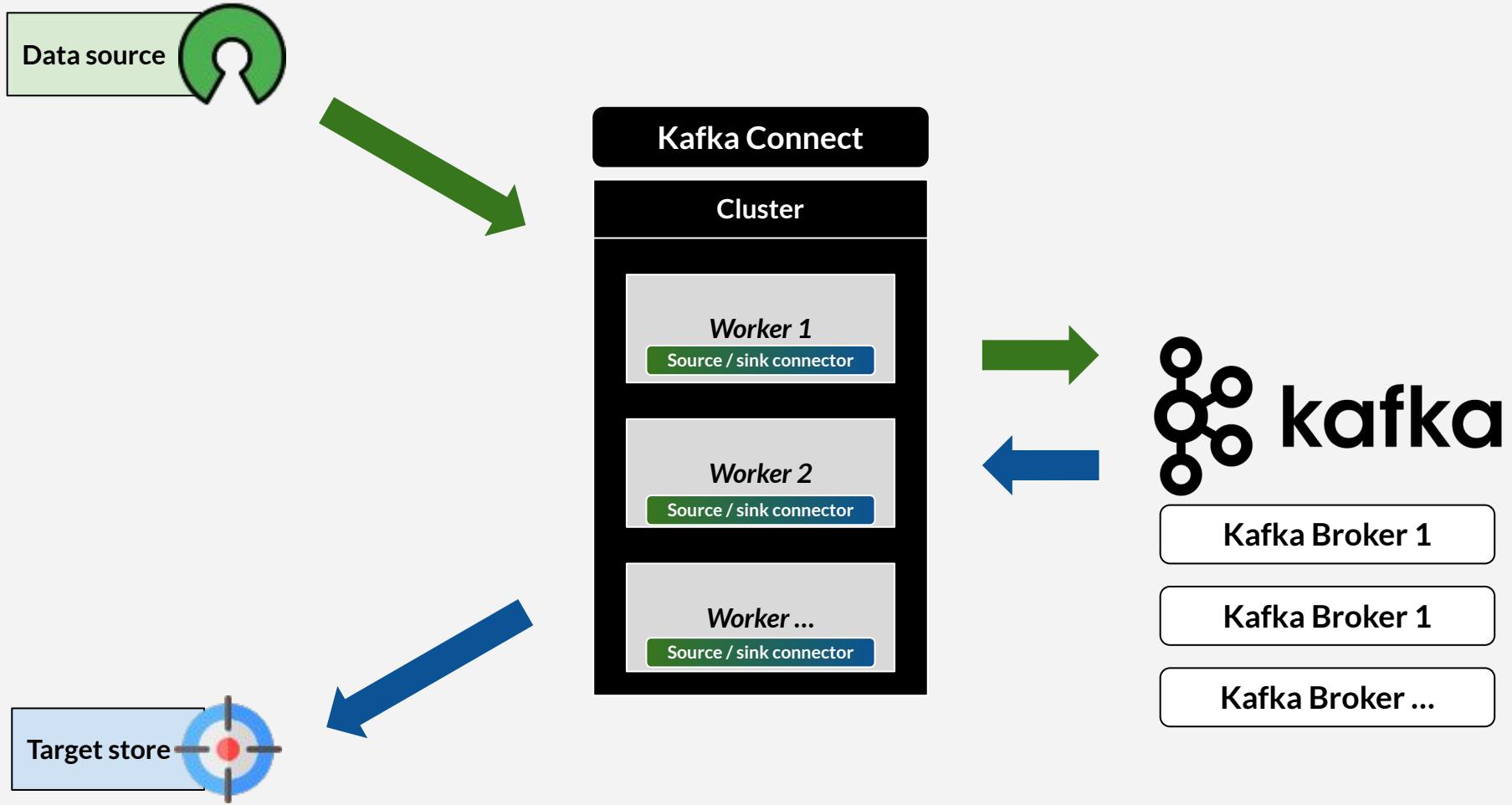
# AI Assistant and Kafka Connect



# AI Assistant & Kafka Connect

- × Can use AI assistant up to a certain point
- × Generate basic Kafka Connect JSON configuration
- × Sometimes fine-tuning by human is better
  - × Limited / outdated AI assistant knowledge base
  - × Might need long & detailed instruction
  - × Might be faster to read & configure the connector based on recent documentation





# AI Assistant & Kafka Connect

- ✗ Not much AI assistant usage in the Kafka Connect lessons
- ✗ Some prompts example provided on **Resources & References**
- ✗ Generate JSON body that resembles connectors in this course
- ✗ Only some examples
- ✗ Difference between AI assistant vs manual
- ✗ AI assistant output is non-deterministic
- ✗ AI assistant generated JSON body will not be executed in this course



# AI Assistant & Kafka Connect

- × Explain existing configuration
- × Ask AI assistant to explain existing JSON configuration



# Kafka User Interface



# Kafka GUI

- ✗ Use GUI instead console
- ✗ Available on market
- ✗ Might need license
- ✗ Check tool's website for pricing & license



# Kafka GUI

- × Confluent Control Center ([confluent.io](https://confluent.io))
- × Kafdrop ([github.com/obsidiandynamics/kafdrop](https://github.com/obsidiandynamics/kafdrop))
- × Kafka-ui ([github.com/provectus/kafka-ui](https://github.com/provectus/kafka-ui))
- × Lenses ([lenses.io](https://lenses.io))
- × Conduktor ([conduktor.io](https://conduktor.io))
  - × We will use this



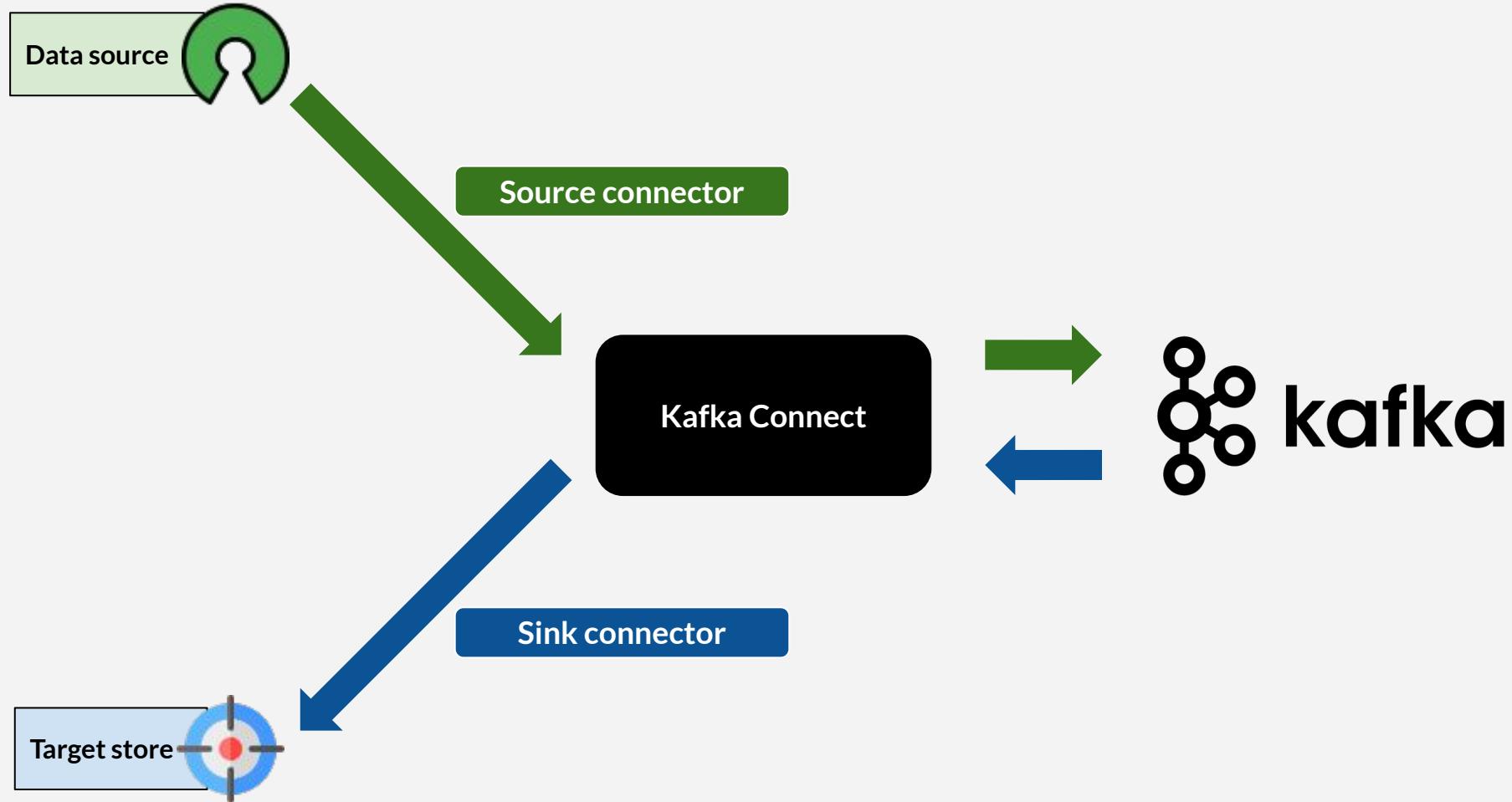
# Conduktor



# Conduktor

- × Using new topics
- × Dummy producer, consumer
- × Dummy kafka stream
- × Source code available on **Resource & Reference**
- × *Note : Conduktor features might different from the course*



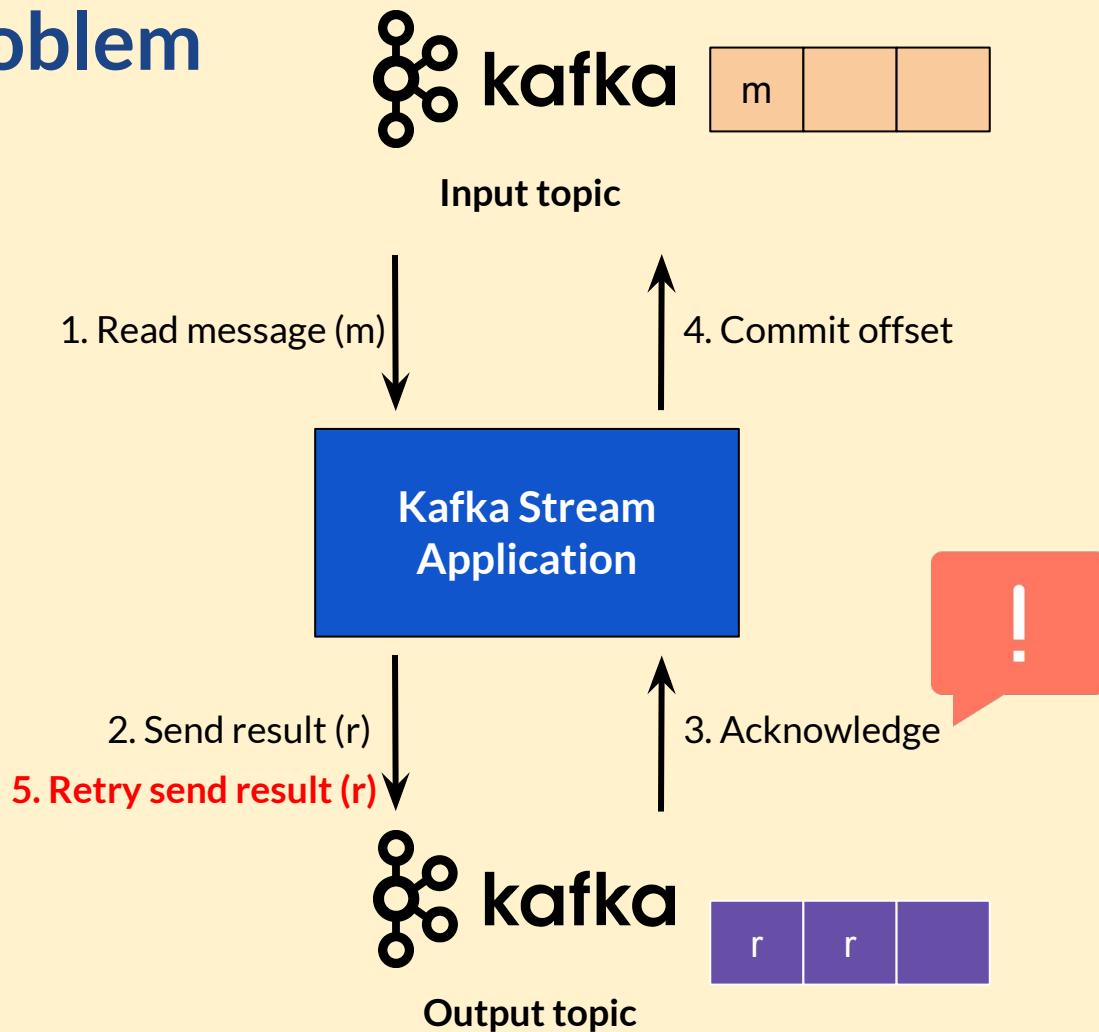


# Kafka Connect

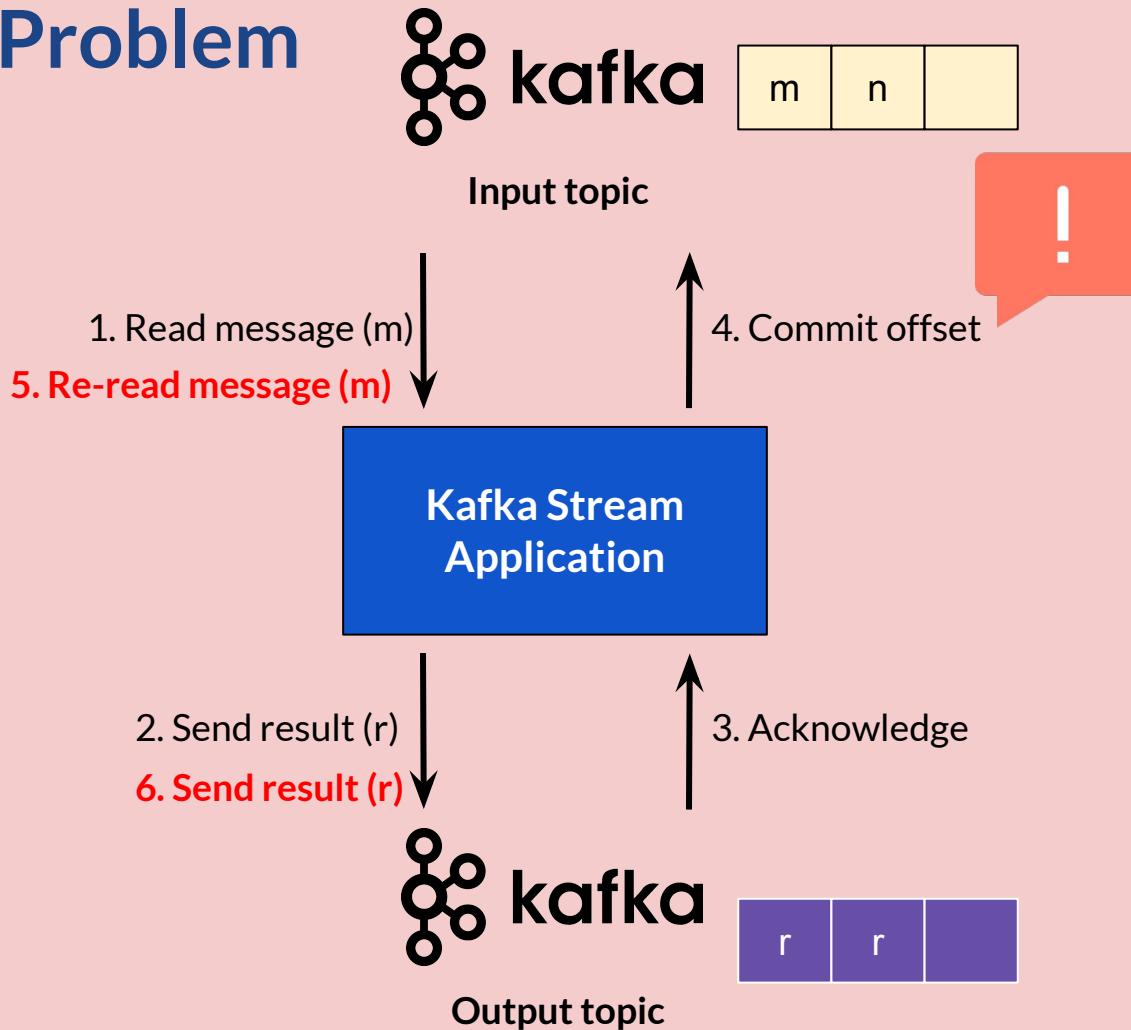
- × Additional platform for kafka
- × Data integration
- × Transfer data between Kafka - non kafka
- × Horizontally scalable & fault tolerant
- × Uses connectors for interact with kafka server



# Publish Problem



# Consume Problem



# Exactly Once in Kafka

- × Idempotent producer
  - × Idempotent?
  - × Kafka producer will make sure only one copy of message exists on topic
- × Atomic transaction
  - × Several operations on kafka stream cycle
  - × Happened all-or-none
  - × Like commit-rollback on database



# Enabling Exactly Once



# Enabling Exactly Once

- × Kafka version  $\geq 0.11$
- × Use library that support exactly once (**Kafka Stream**)
- × Add configuration on properties
- × No additional code

