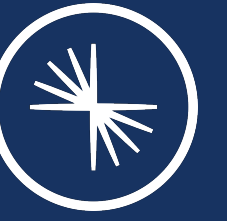


Kafka as a Platform: the Ecosystem from the Ground Up

Robin Moffatt

@rmoff



EVENTS

EVENTS

EVENTS

- Something happened
- What happened

Human generated events

A Sale



*A Stock
movement*



Machine generated events

Networking



IoT



Applications



EVENTS

are

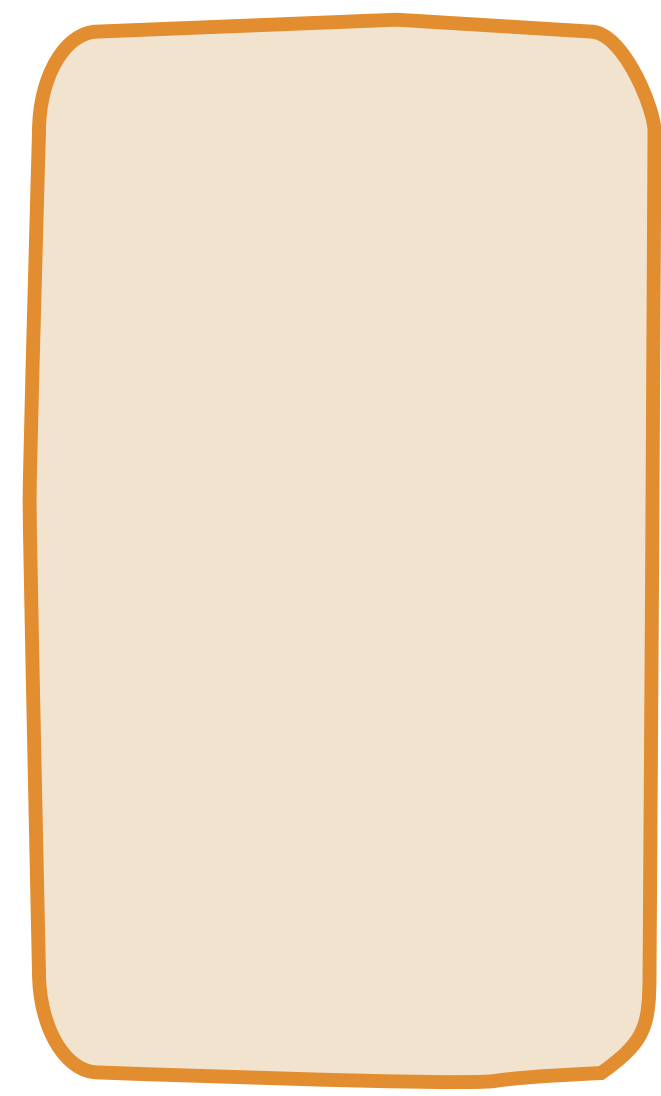
EVERYWHERE

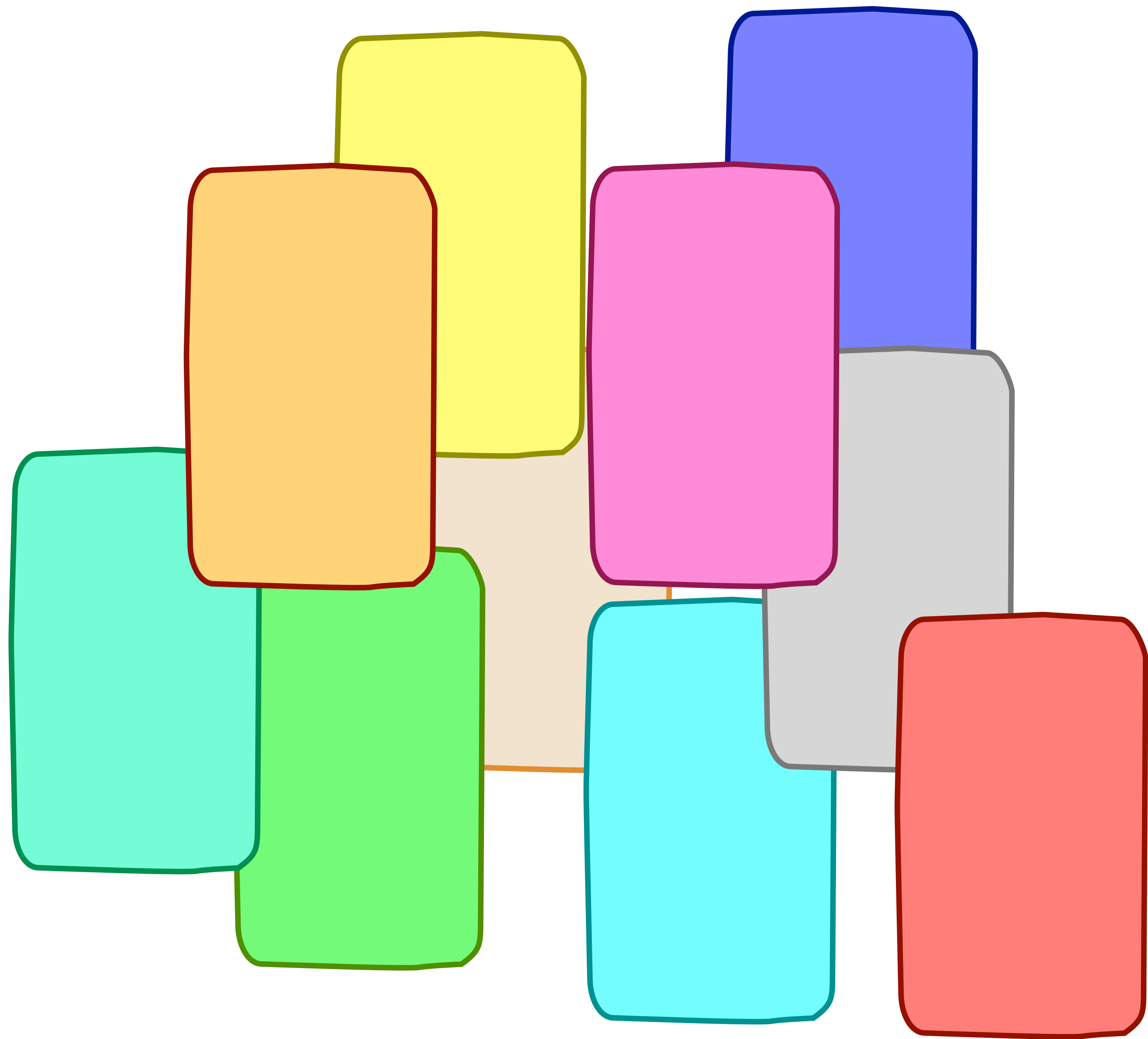
EVENTS

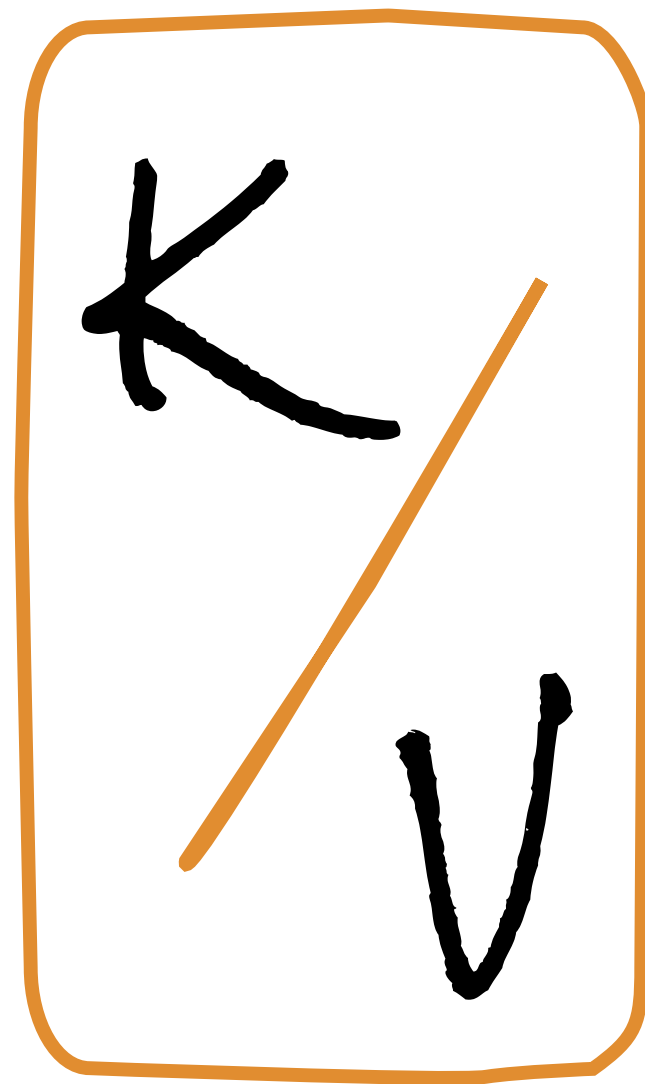
are

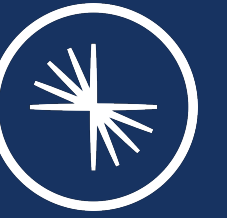
very
^

POWERFUL

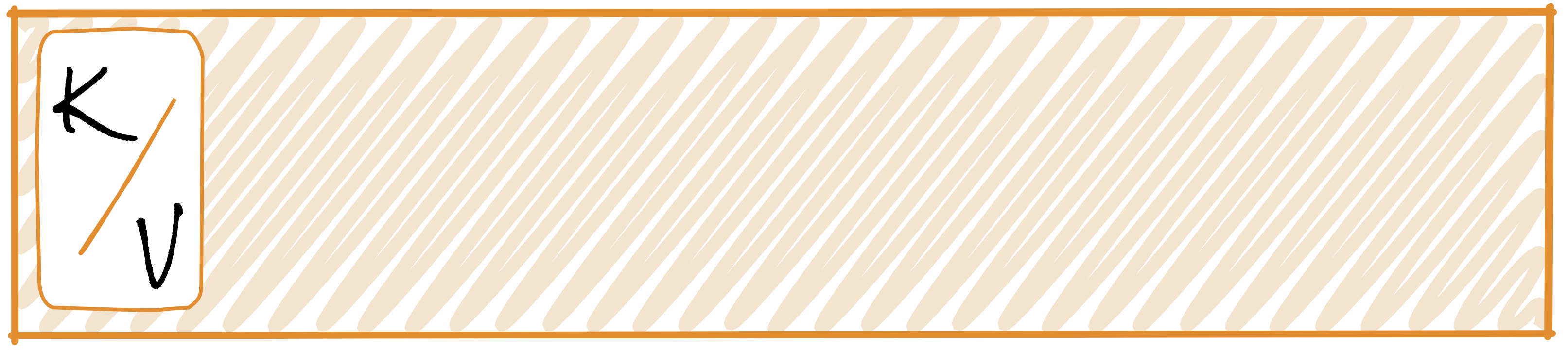




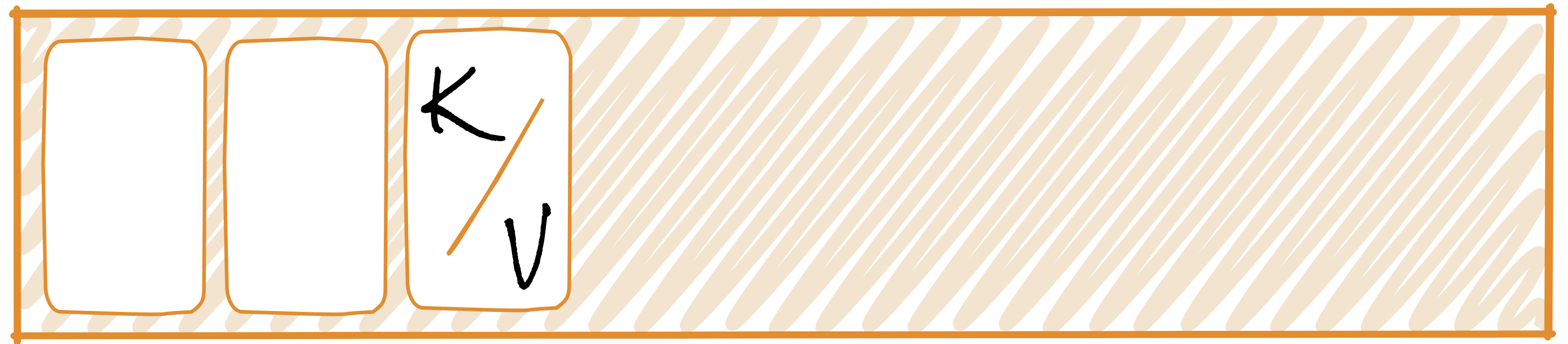




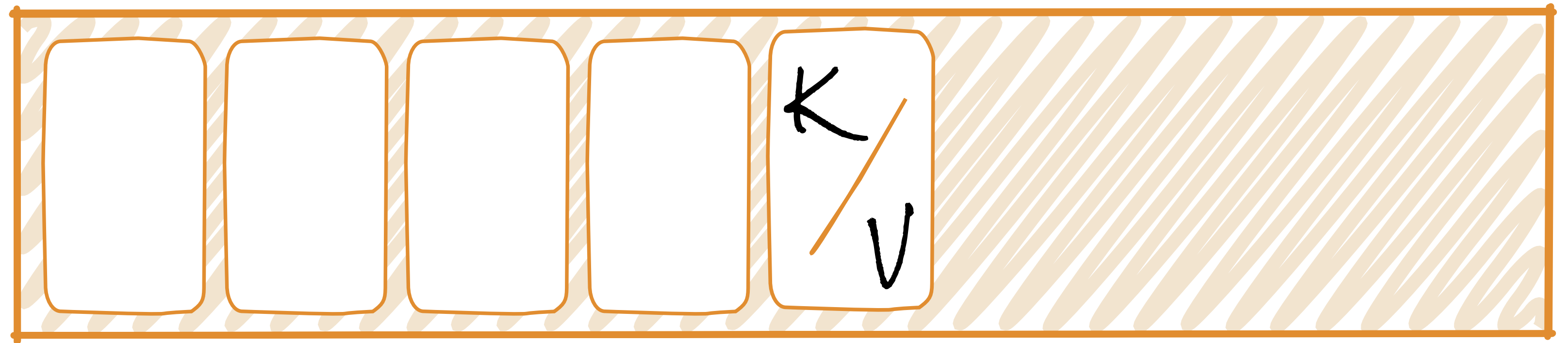
LOG

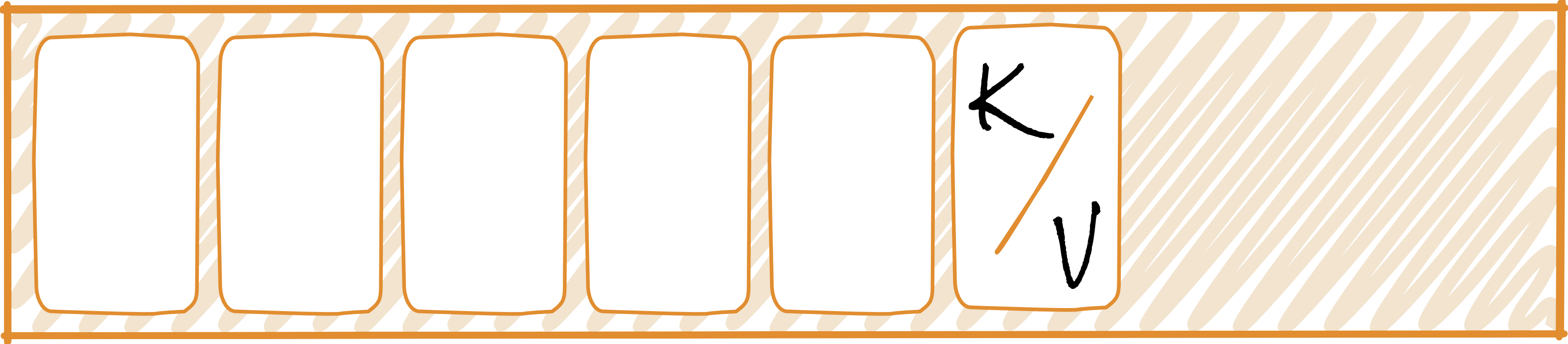


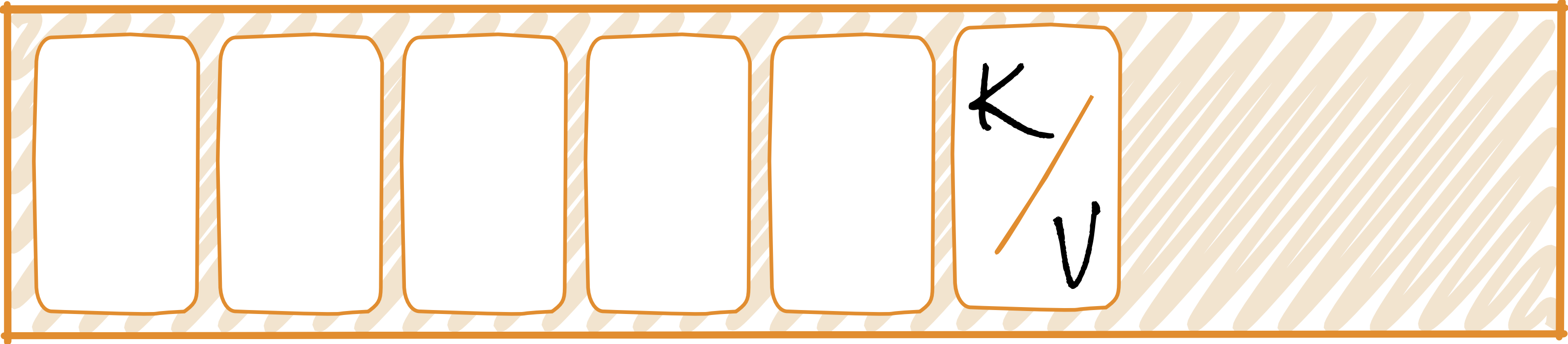




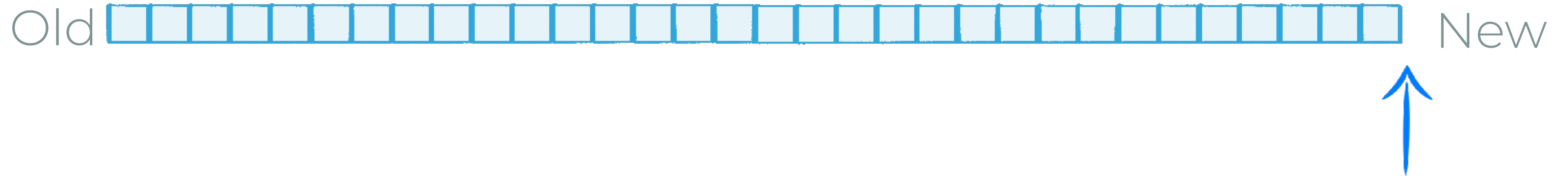




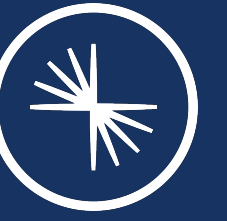




Immutable Event Log



Events are added at the end of the log



TOPICS

Topics

Clicks



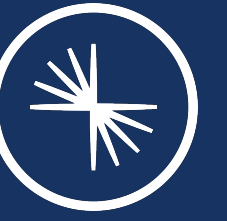
Orders



Customers



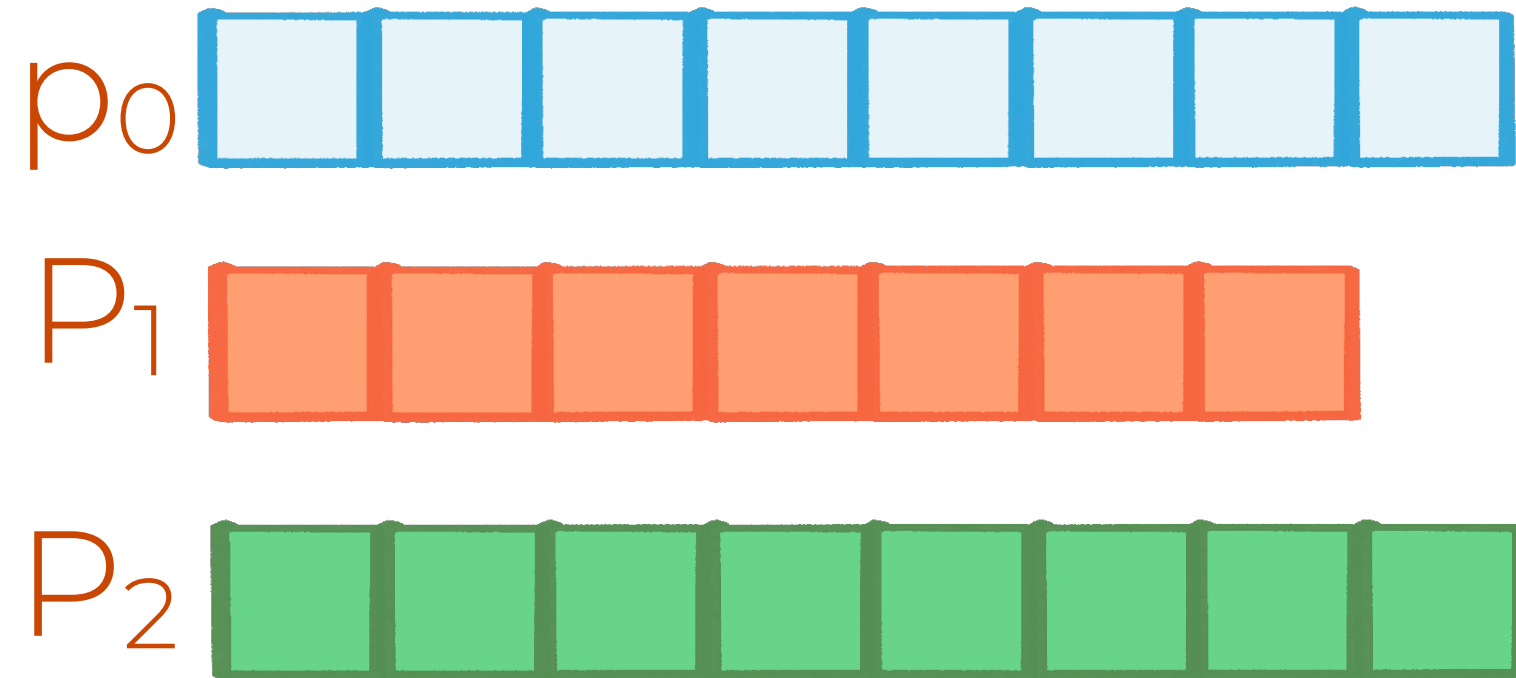
Topics are similar in
concept to tables in a
database



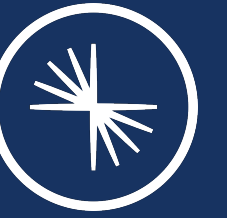
PARTITIONS

Partitions

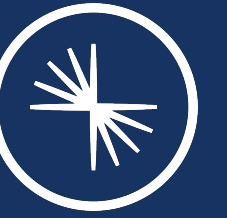
Clicks



Messages are guaranteed to
be strictly ordered within a
partition

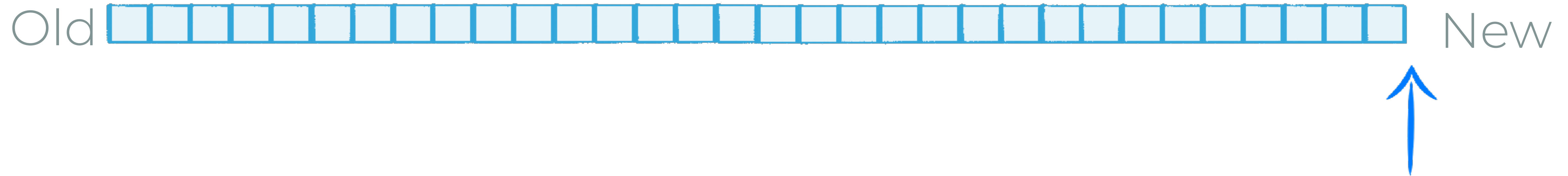


PUB / SUB

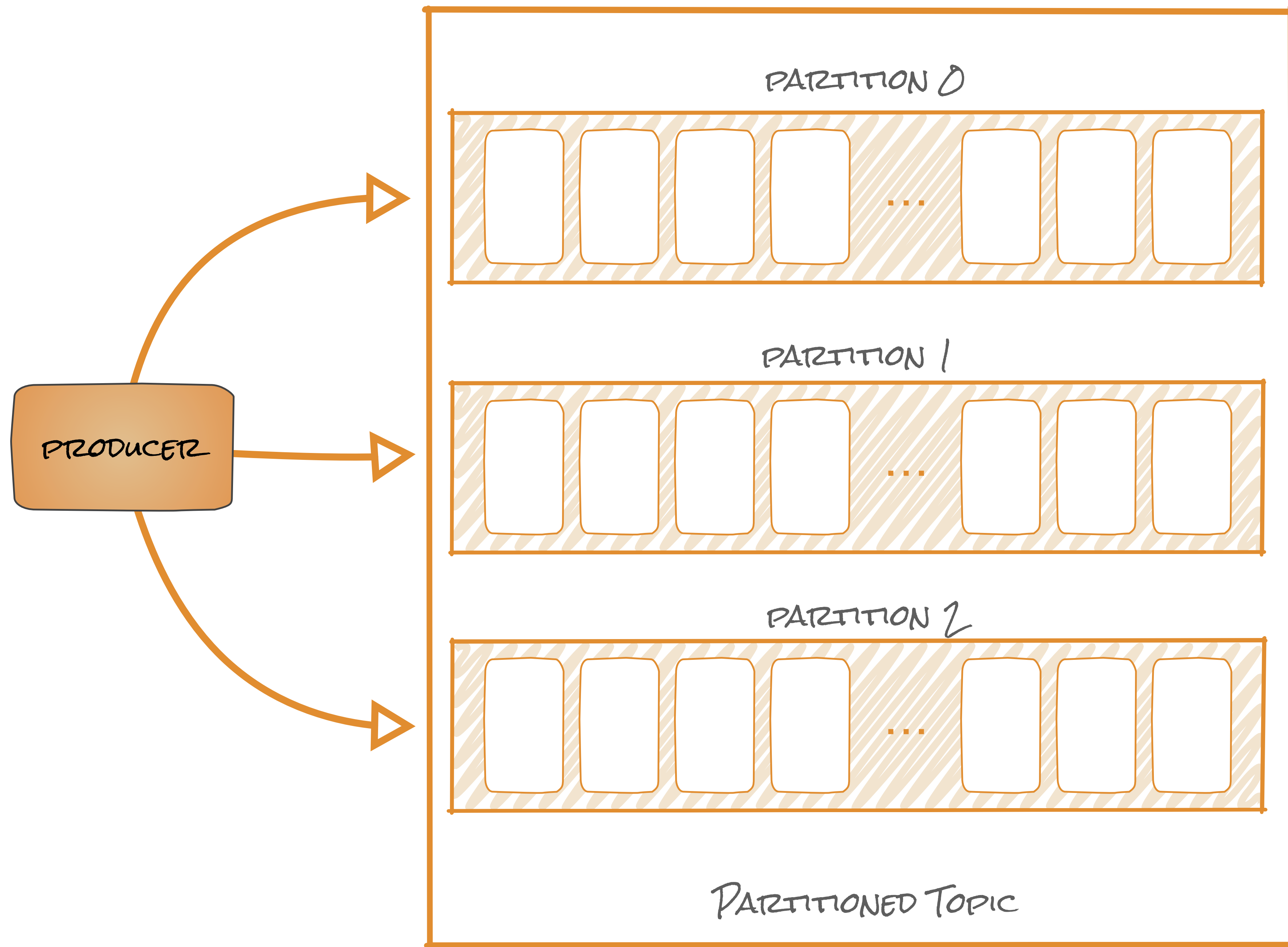


PUB / SUB

Producing data



Messages are added at the end of the log



```
package main

import (
    "gopkg.in/confluentinc/confluent-kafka-go.v1/kafka"
)

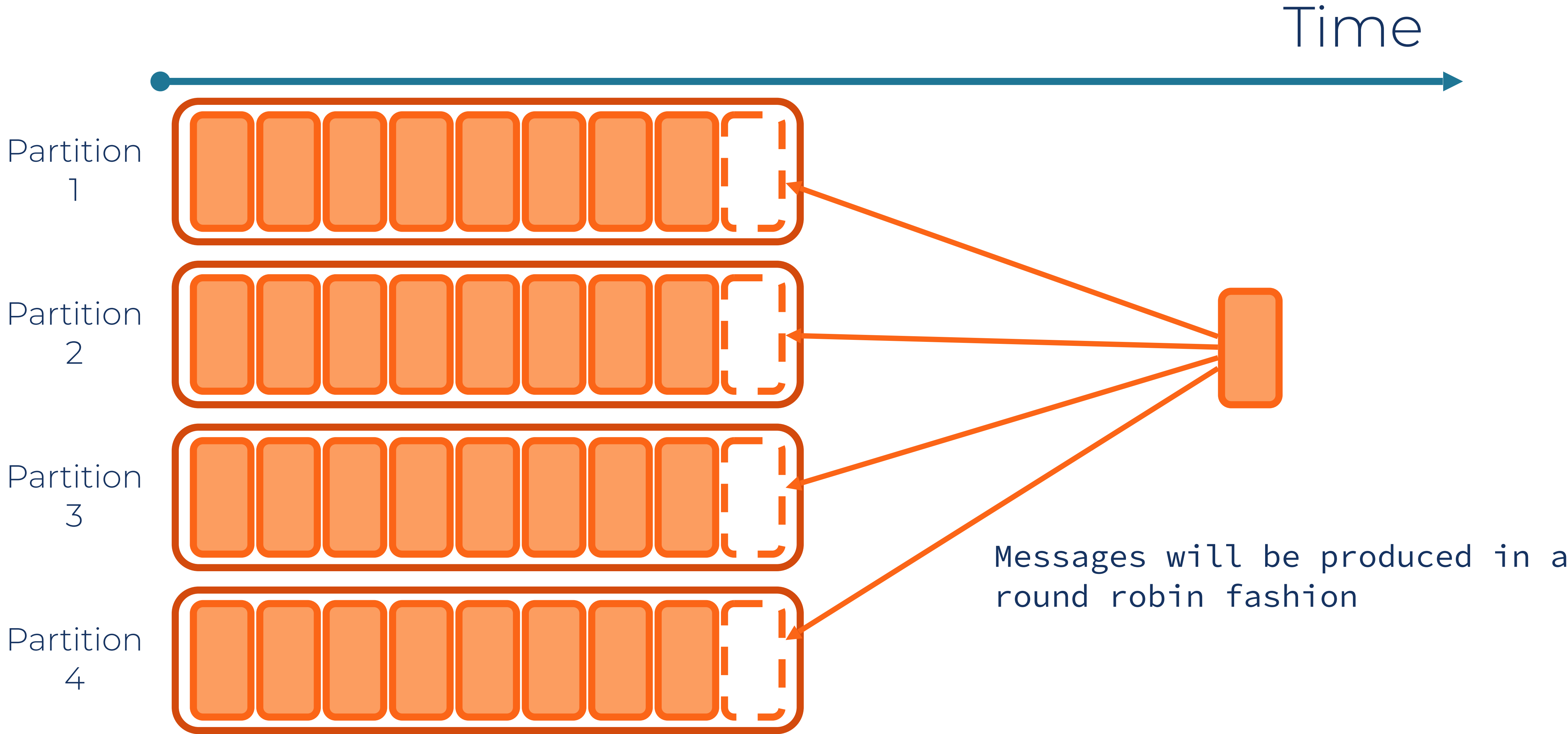
func main() {

    topic := "test_topic"
    p, _ := kafka.NewProducer(&kafka.ConfigMap{
        "bootstrap.servers": "localhost:9092"})
    defer p.Close()

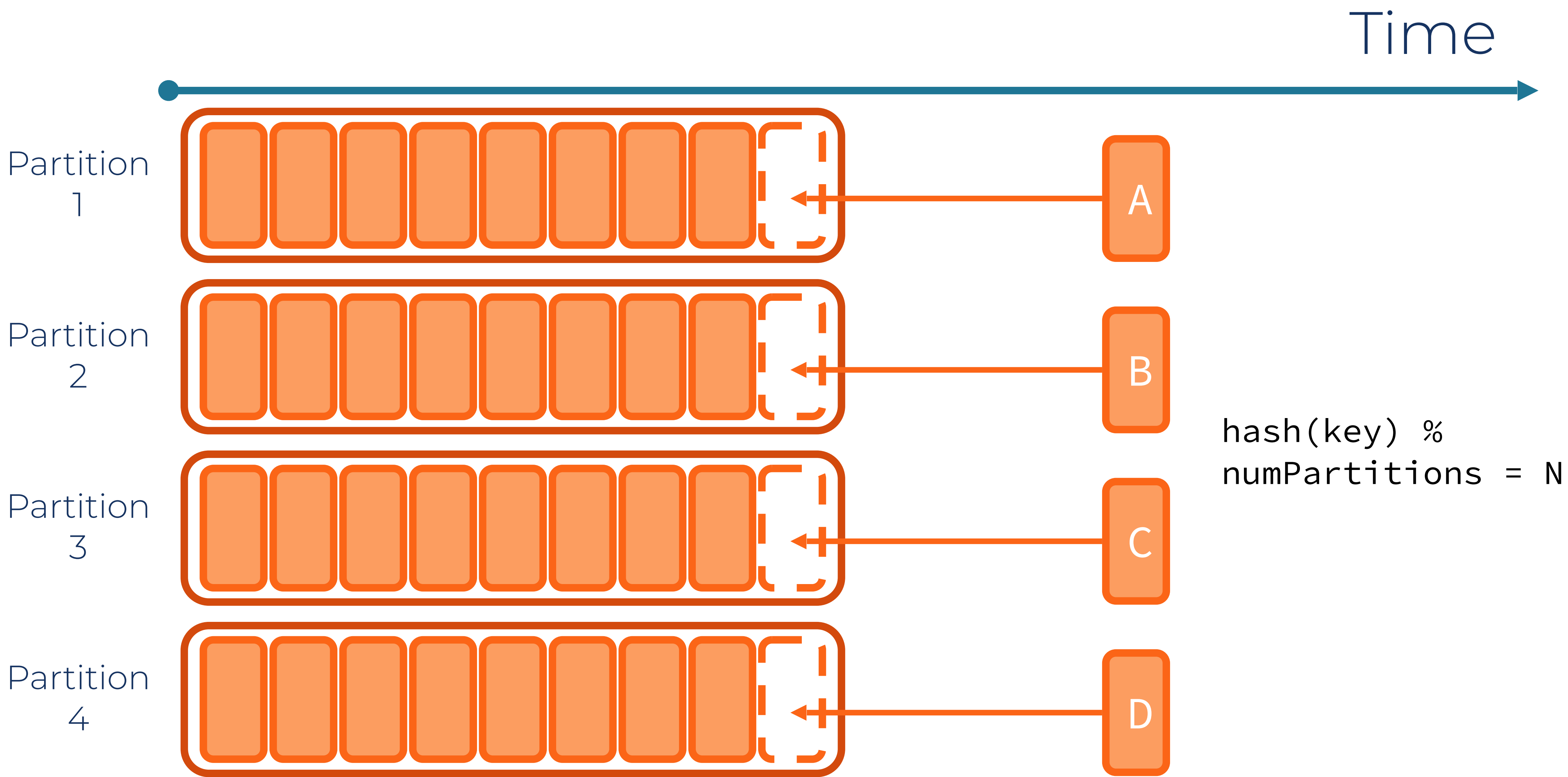
    p.Produce(&kafka.Message{
        TopicPartition: kafka.TopicPartition{Topic: &topic,
            Partition: 0},
        Value: []byte("Hello world")}, nil)

}
```

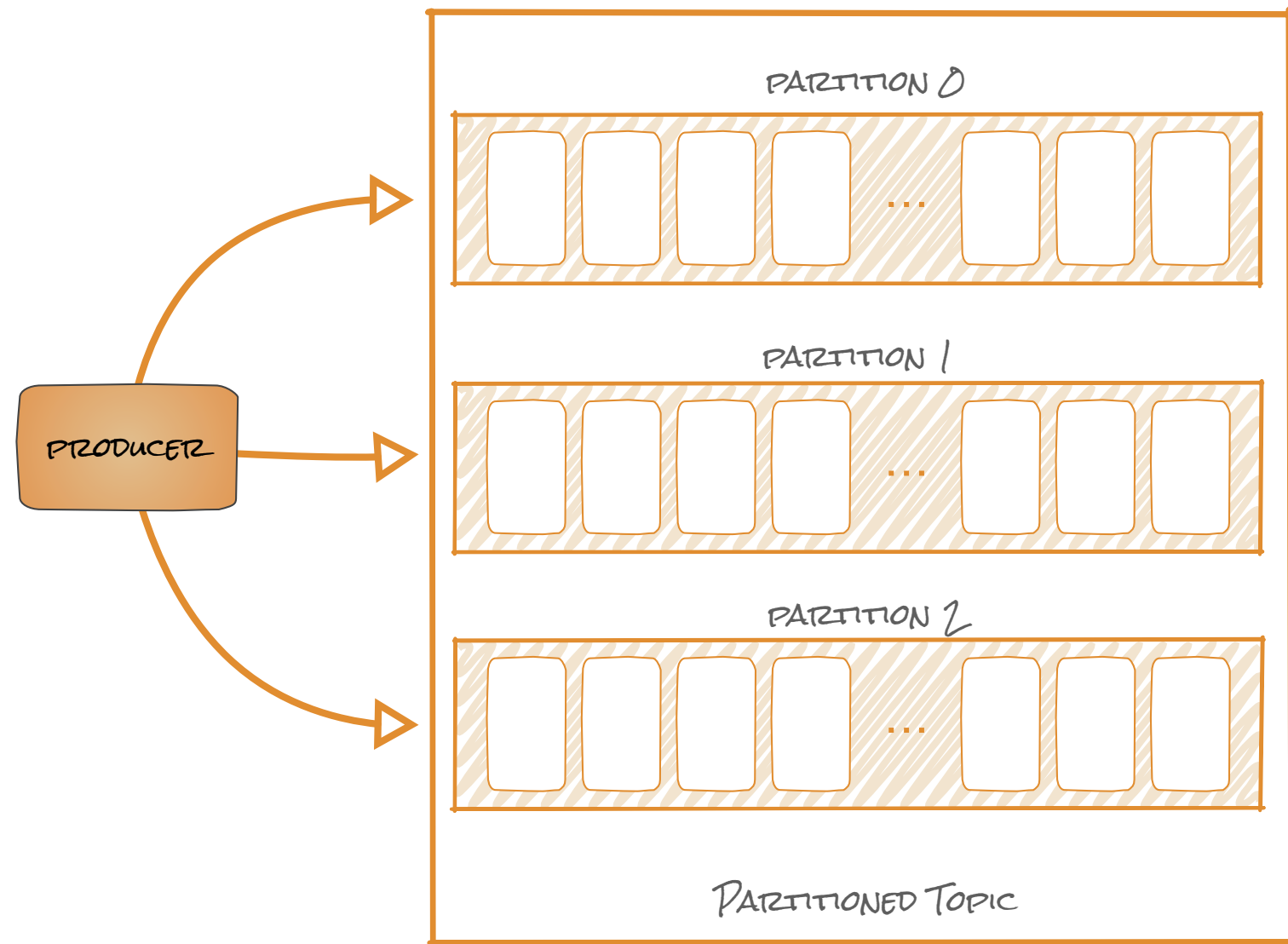

Producing to Kafka - No Key



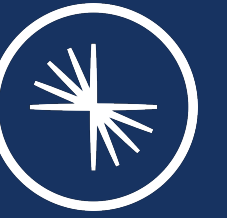
Producing to Kafka - With Key



Producers



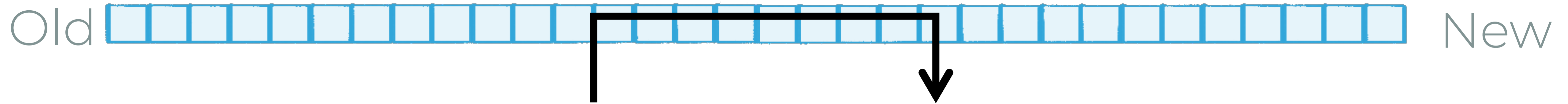
- **A client application**
- **Puts messages into topics**
- **Handles partitioning, network protocol**
- **Java, Go, .NET, C/C++, Python**
- **Also every other language**



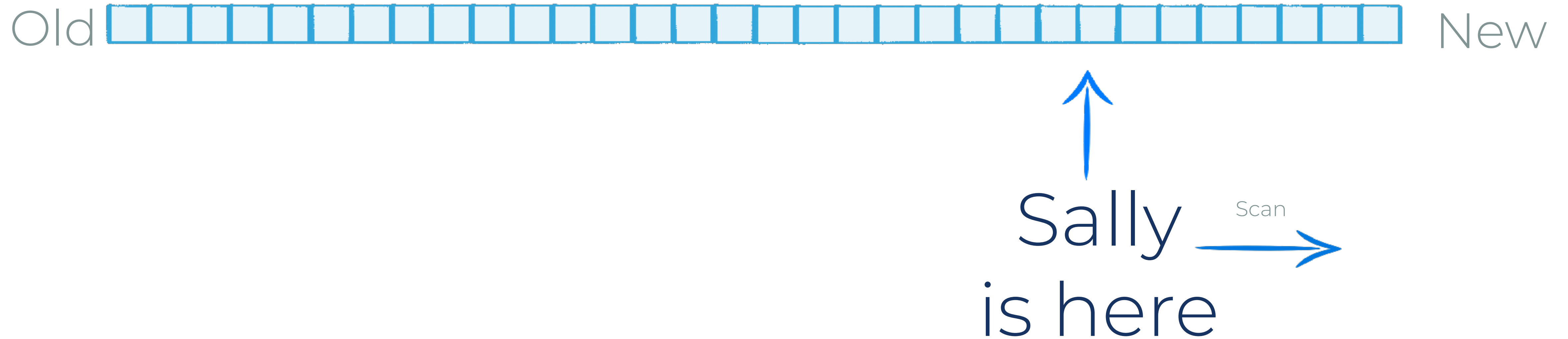
PUB / SUB

Consuming data - access is only sequential

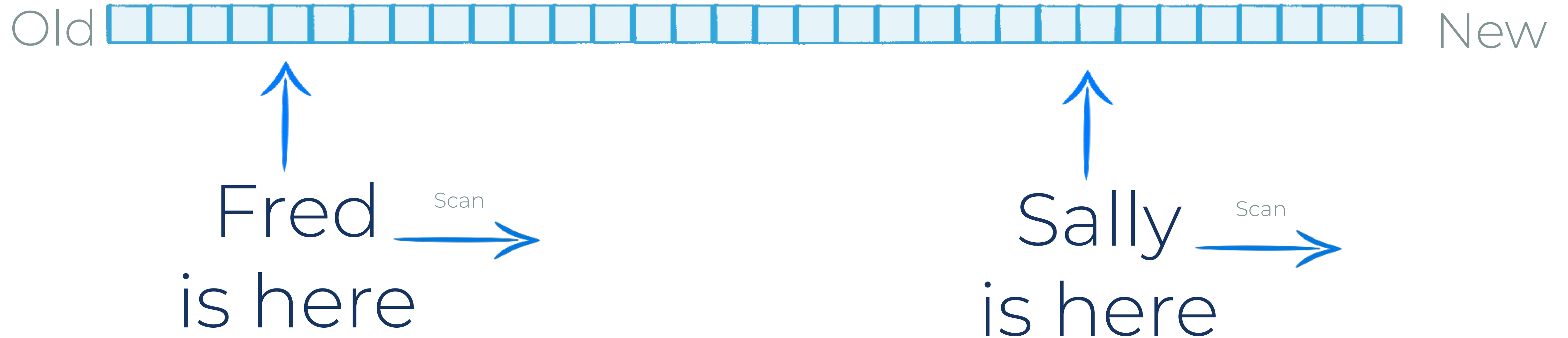
Read to offset & scan



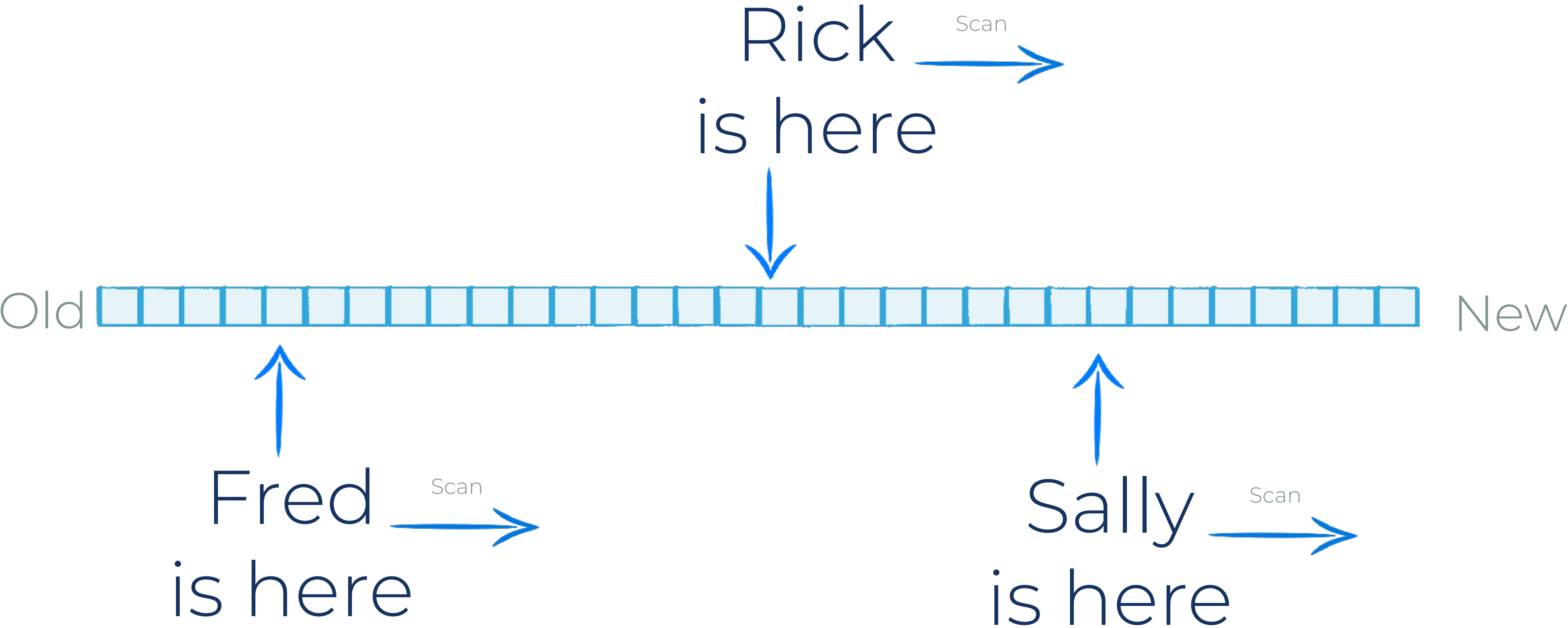
Consumers have a position of their own



Consumers have a position of their own



Consumers have a position of their own

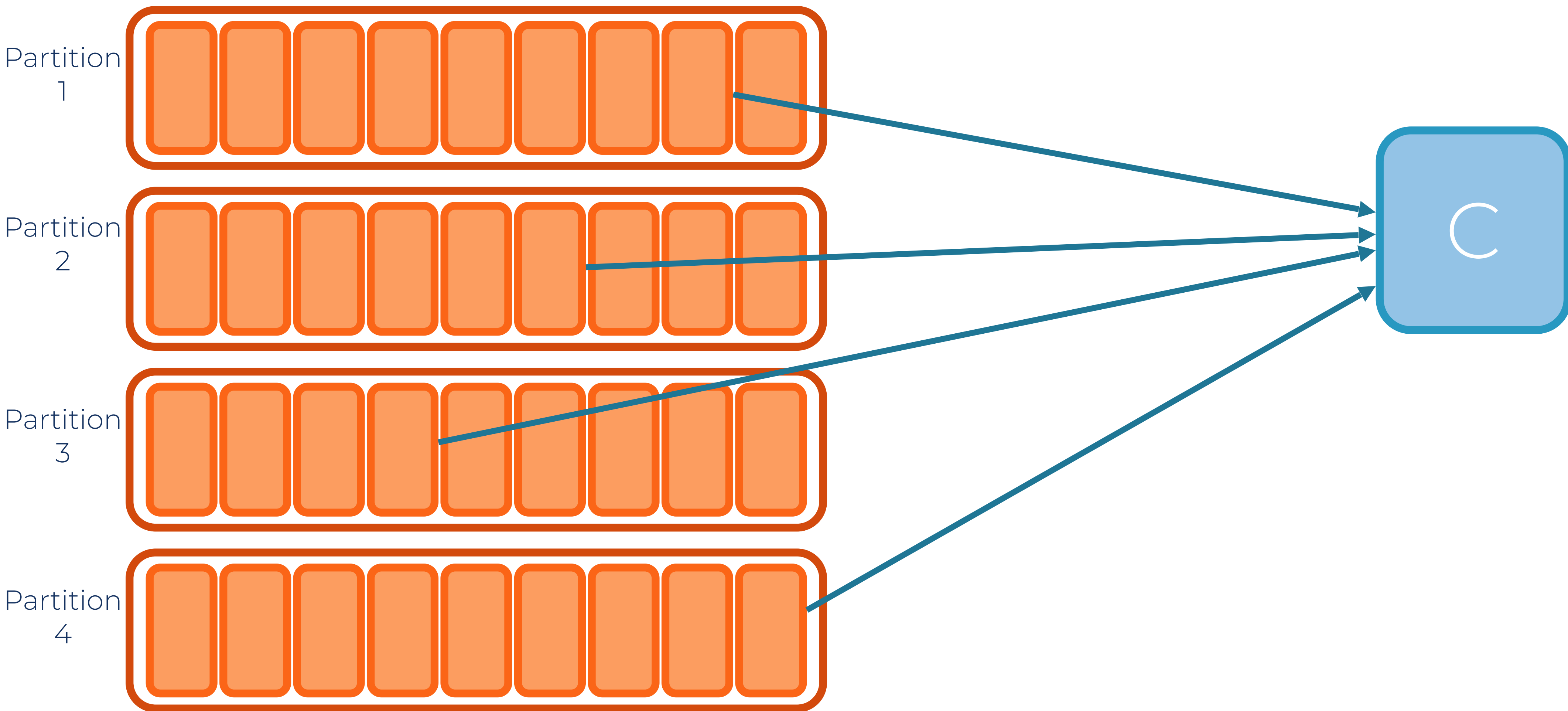


```
c, _ := kafka.NewConsumer(&cm)
defer c.Close()
c.Subscribe(topic, nil)

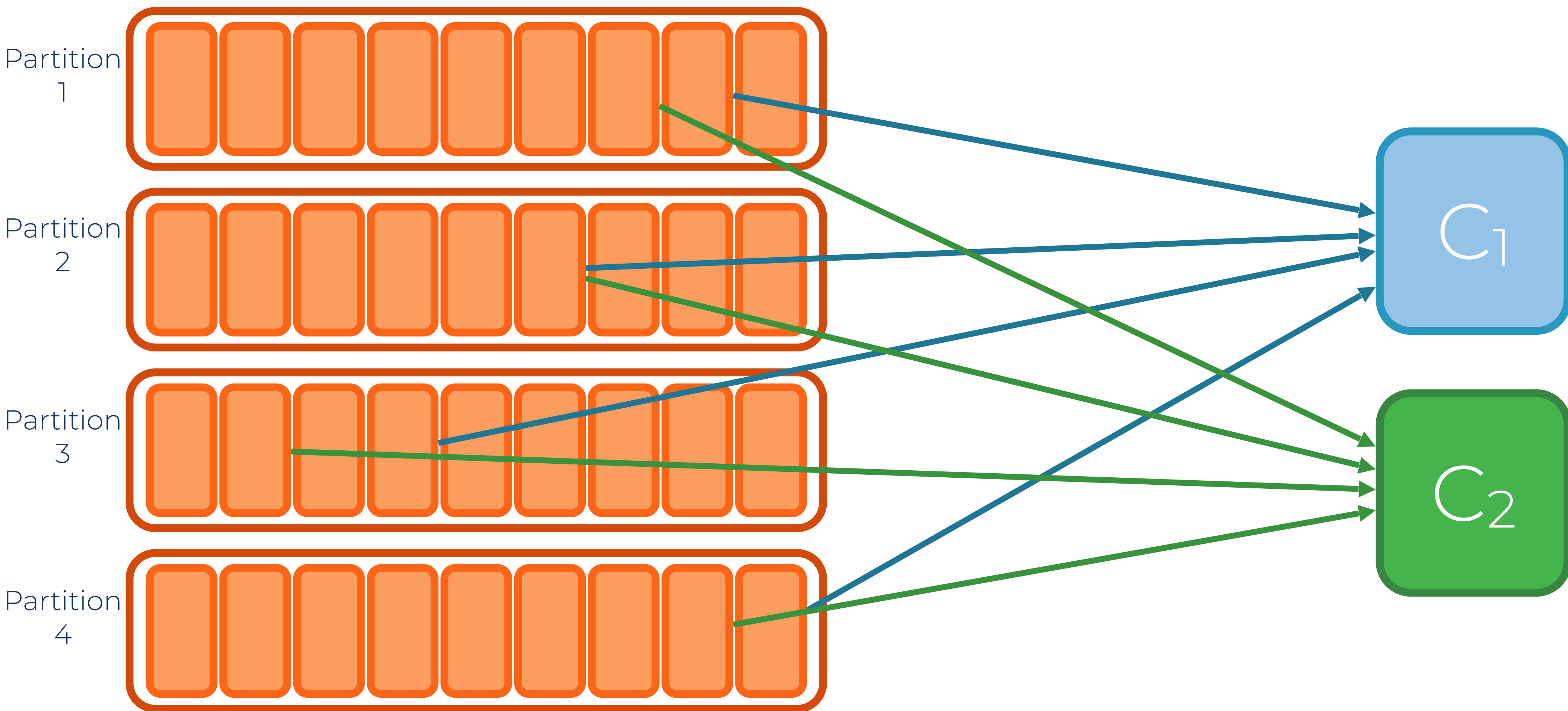
for {
    select {
    case ev := <-c.Events():
        switch ev.(type) {

        case *kafka.Message:
            km := ev.(*kafka.Message)
            fmt.Printf("✅ Message '%v' received from topic '%v'\n", string(km.Value), string(*km.TopicPartition.Topic))
        }
    }
}
```

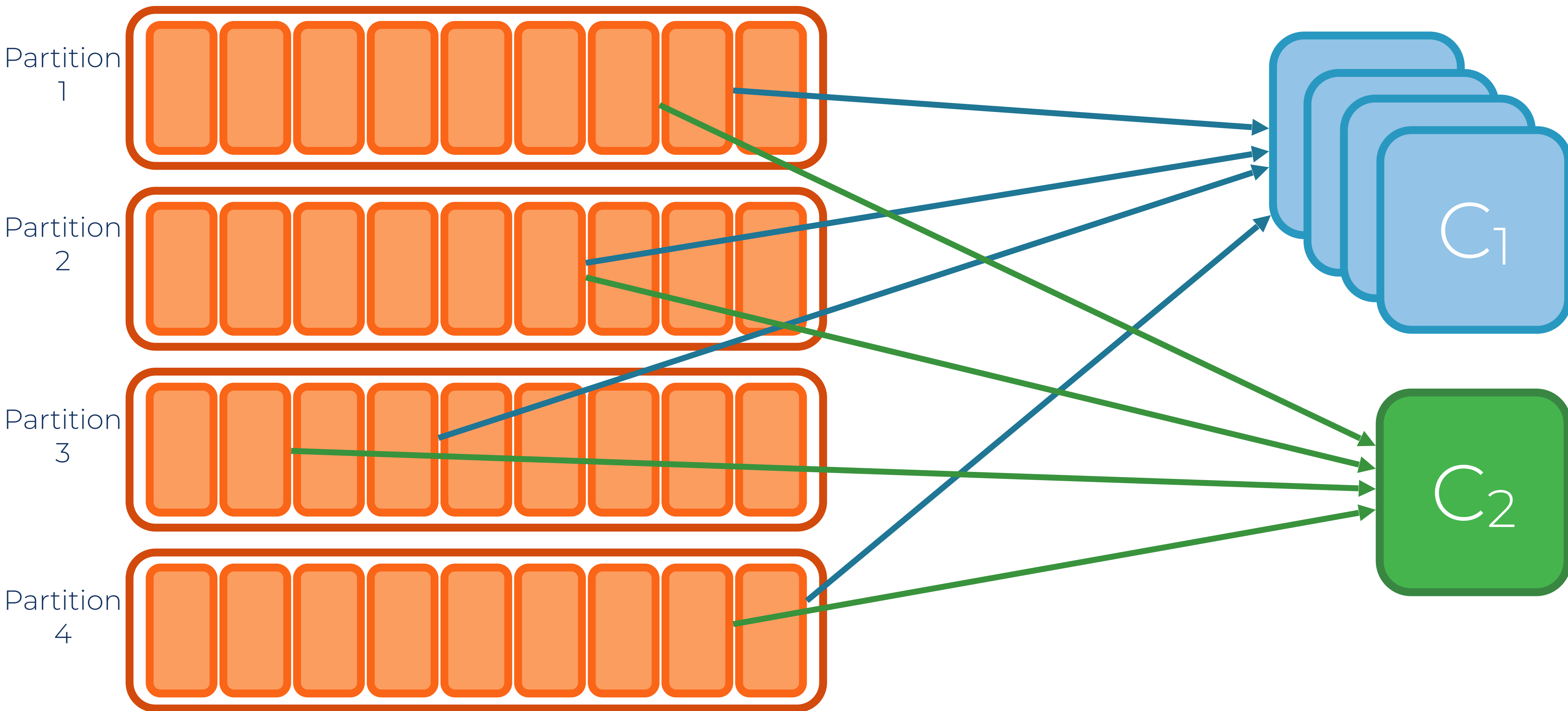

Consuming From Kafka - Single Consumer



Consuming From Kafka - Multiple Consumers



Consuming From Kafka - Grouped Consumers



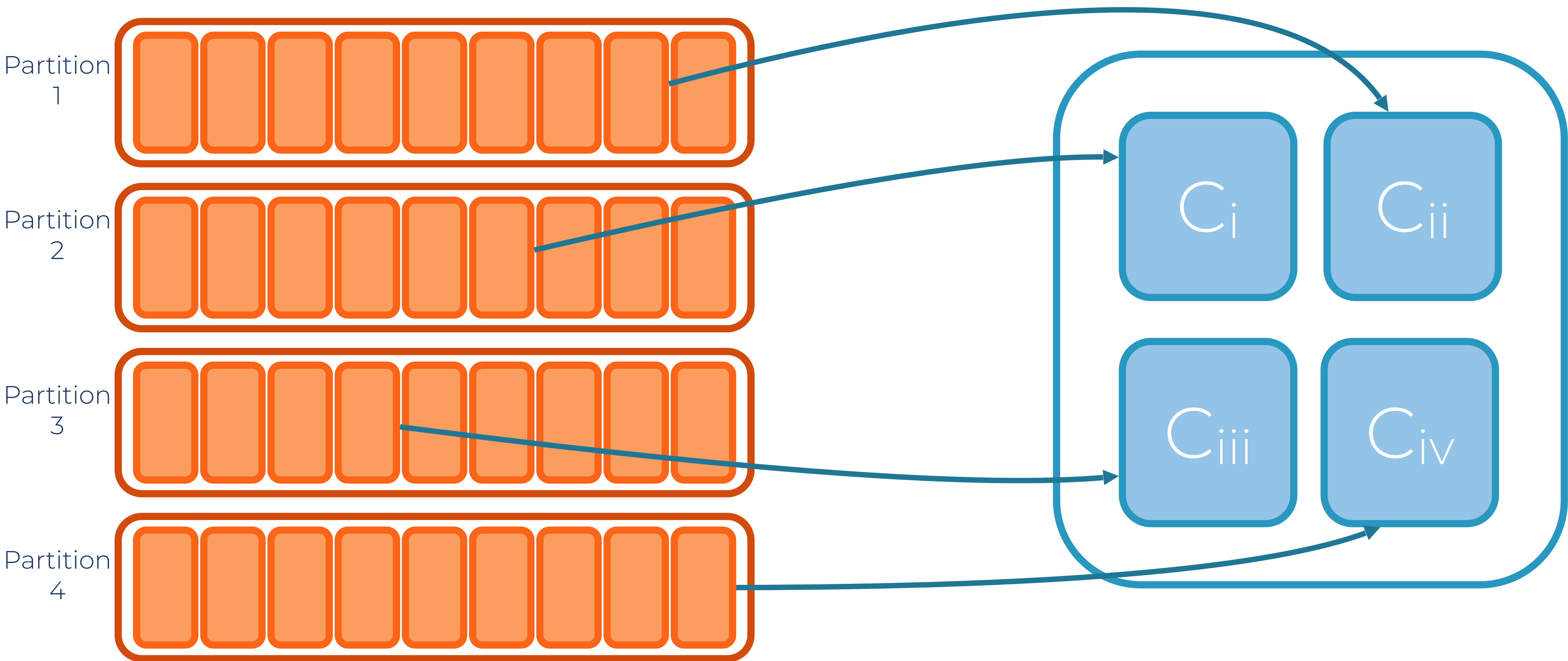


CONSUMERS

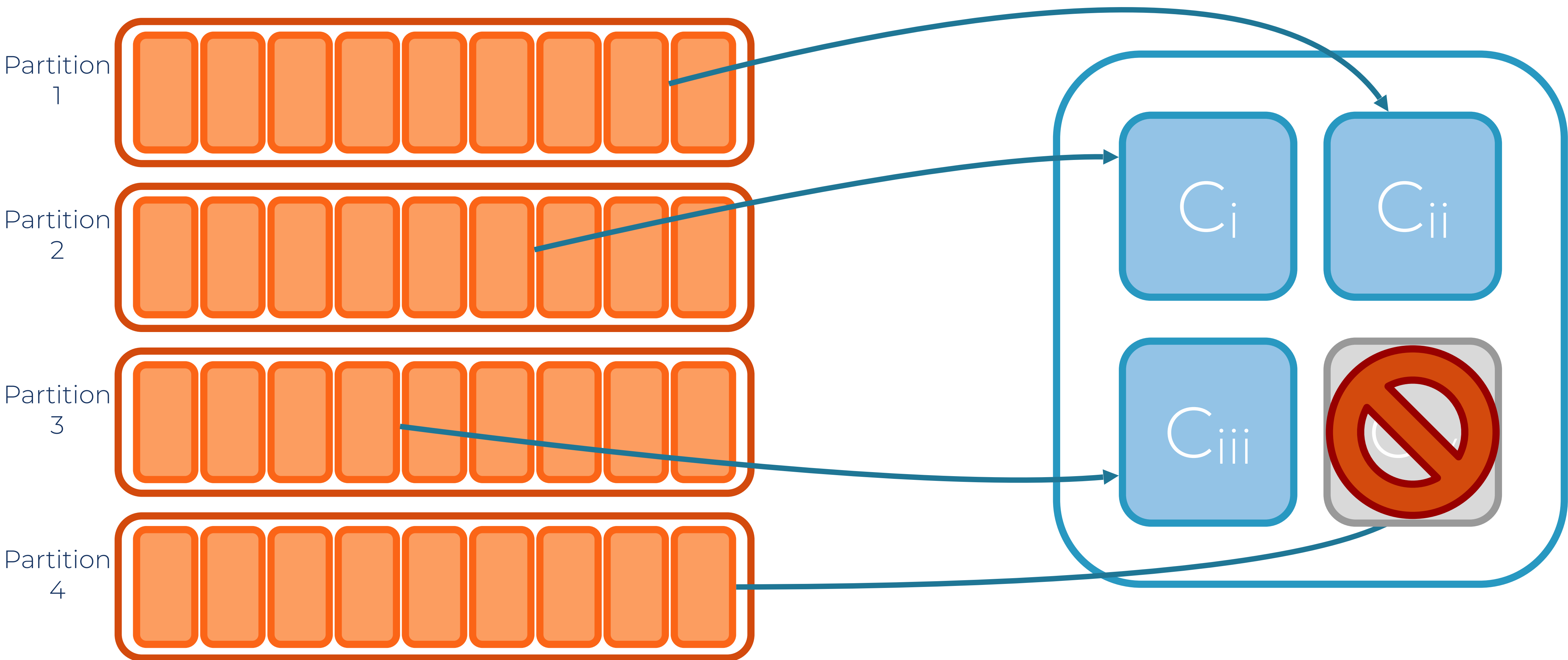
**CONSUMER GROUP
COORDINATOR**

CONSUMER GROUP

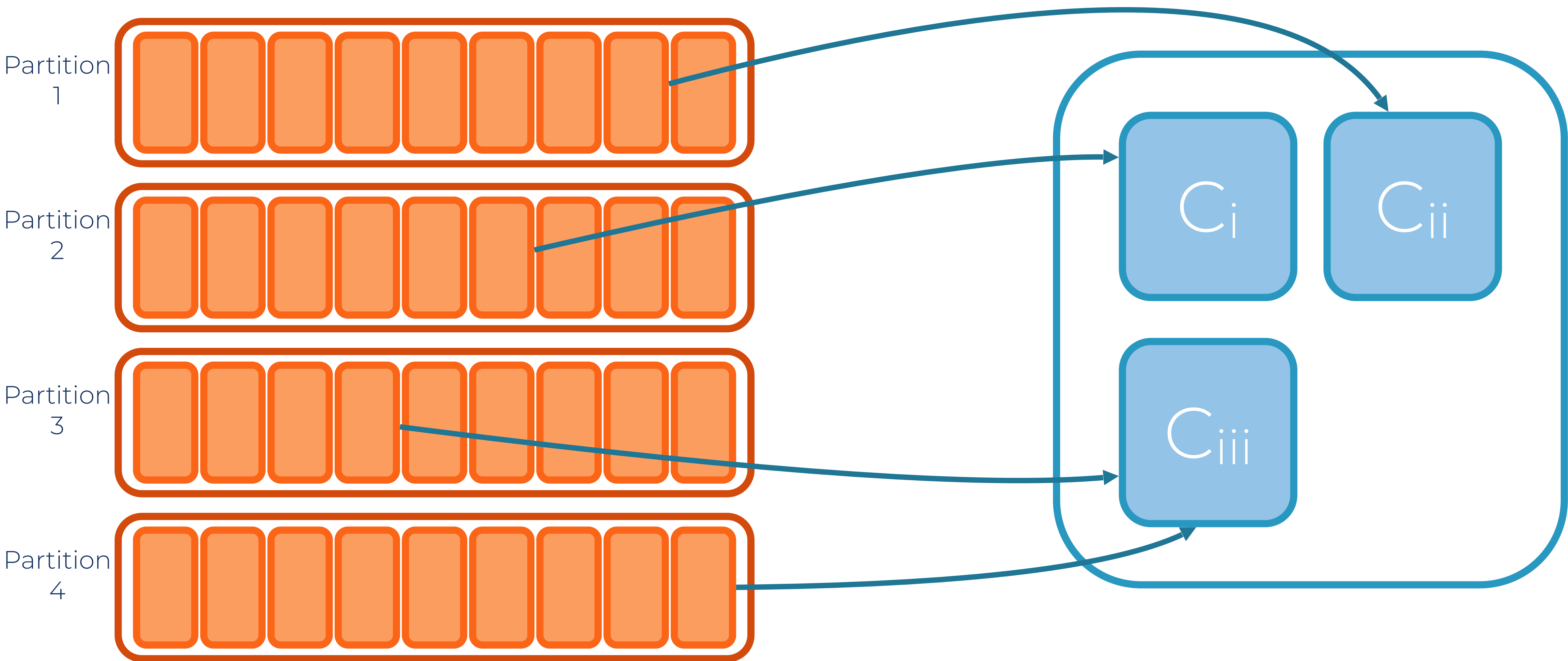
Consuming From Kafka - Grouped Consumers



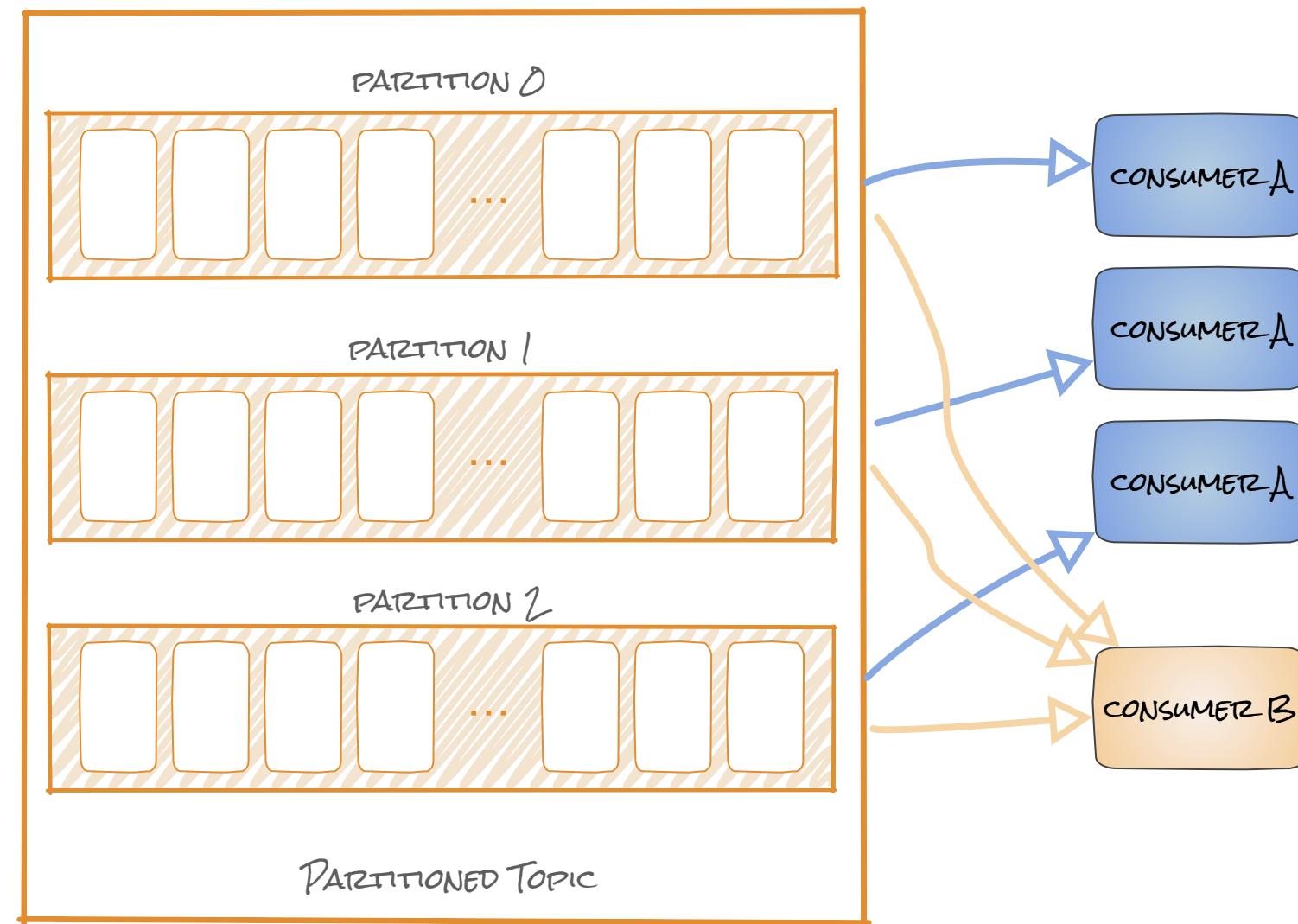
Consuming From Kafka - Grouped Consumers



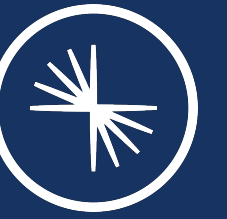
Consuming From Kafka - Grouped Consumers



Consumers



- **A client application**
- **Reads messages from topics**
- **Horizontally, elastically scalable (if stateless)**
- **Java, Go, .NET, C/C++, Python, everything else**



BROKERS

and REPLICATION

Partition Leadership and Replication

Leader

Follower

Partition 1

Partition 2

Partition 3

Partition 4

Broker 1

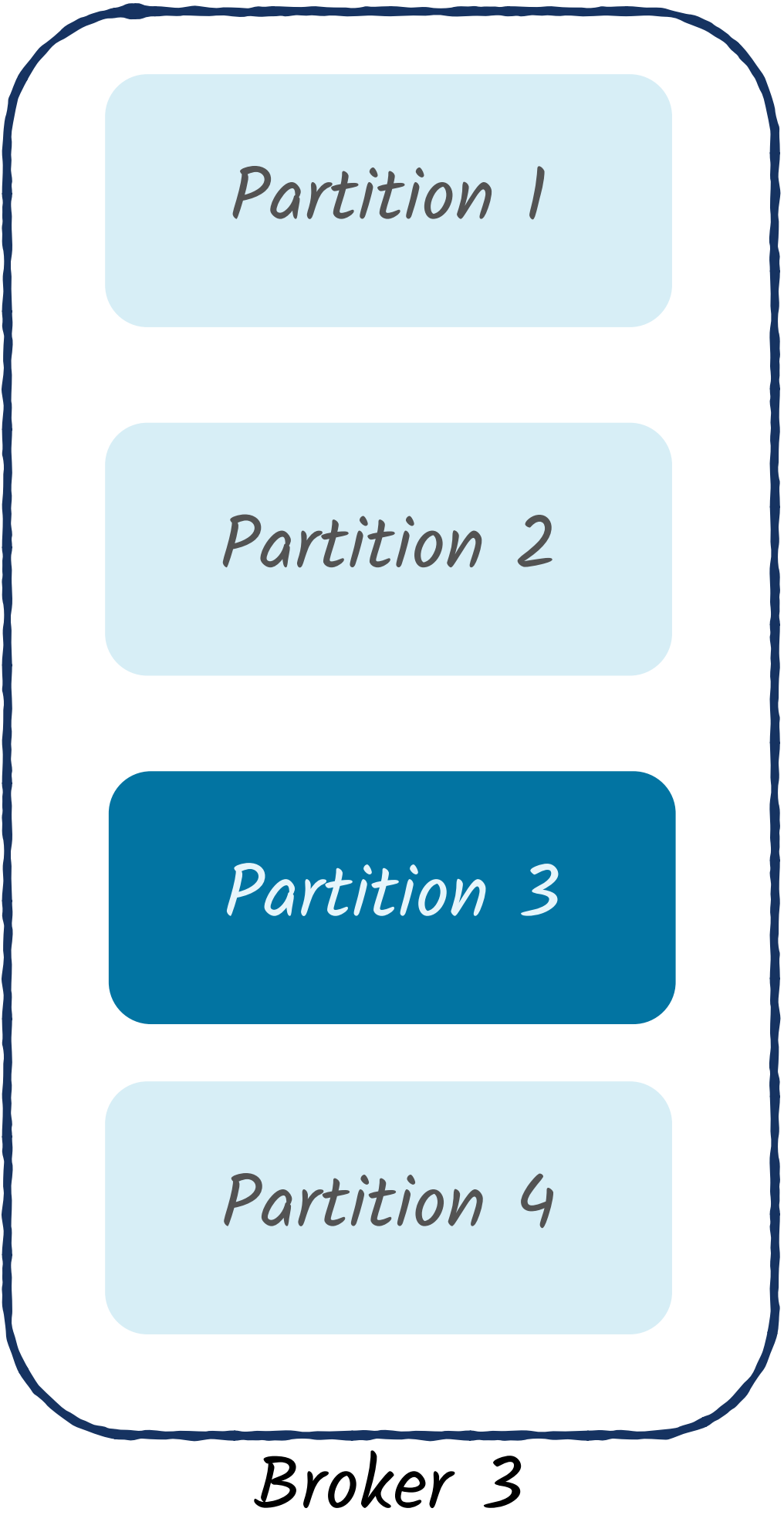
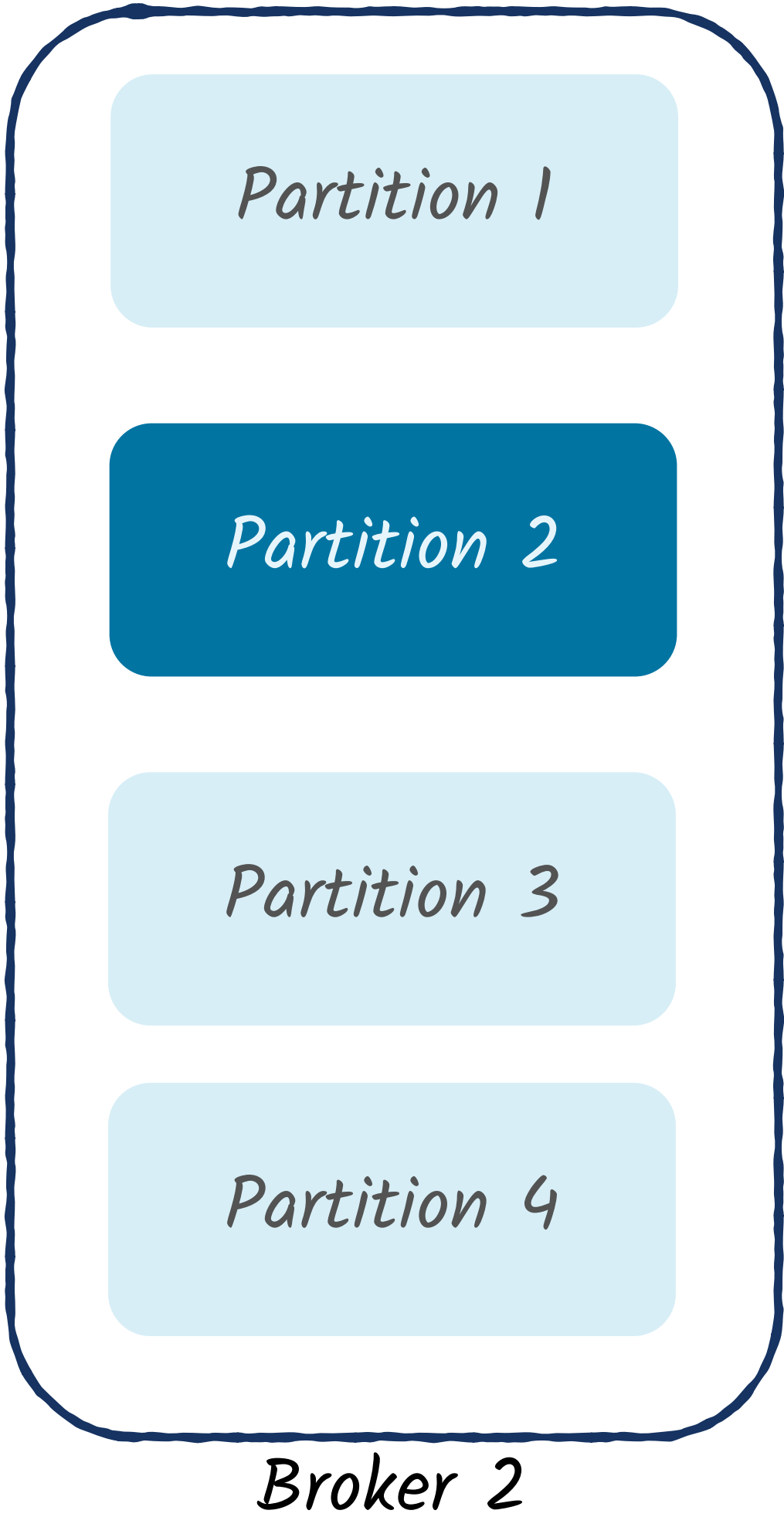
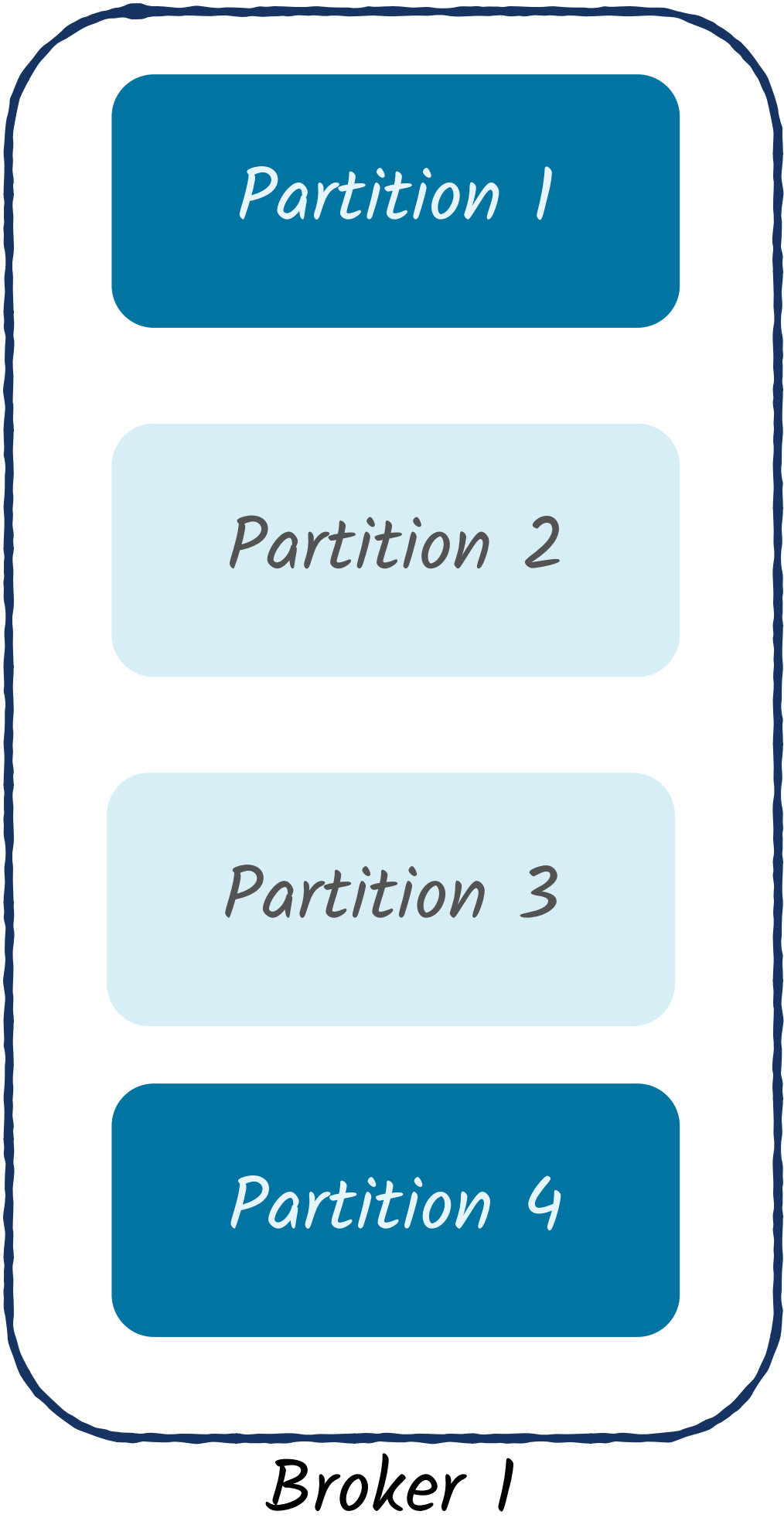
Broker 2

Broker 3

Partition Leadership and Replication

Leader

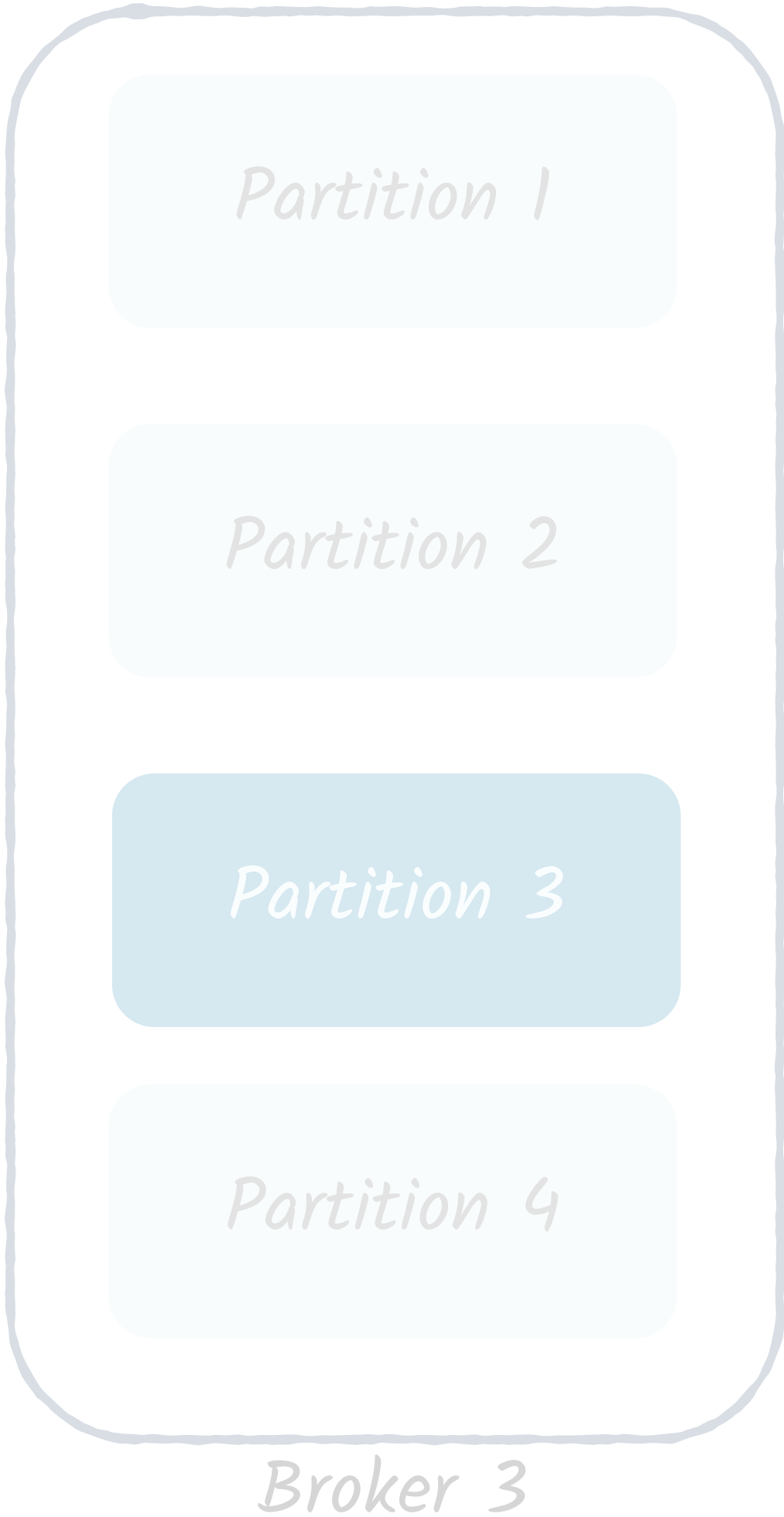
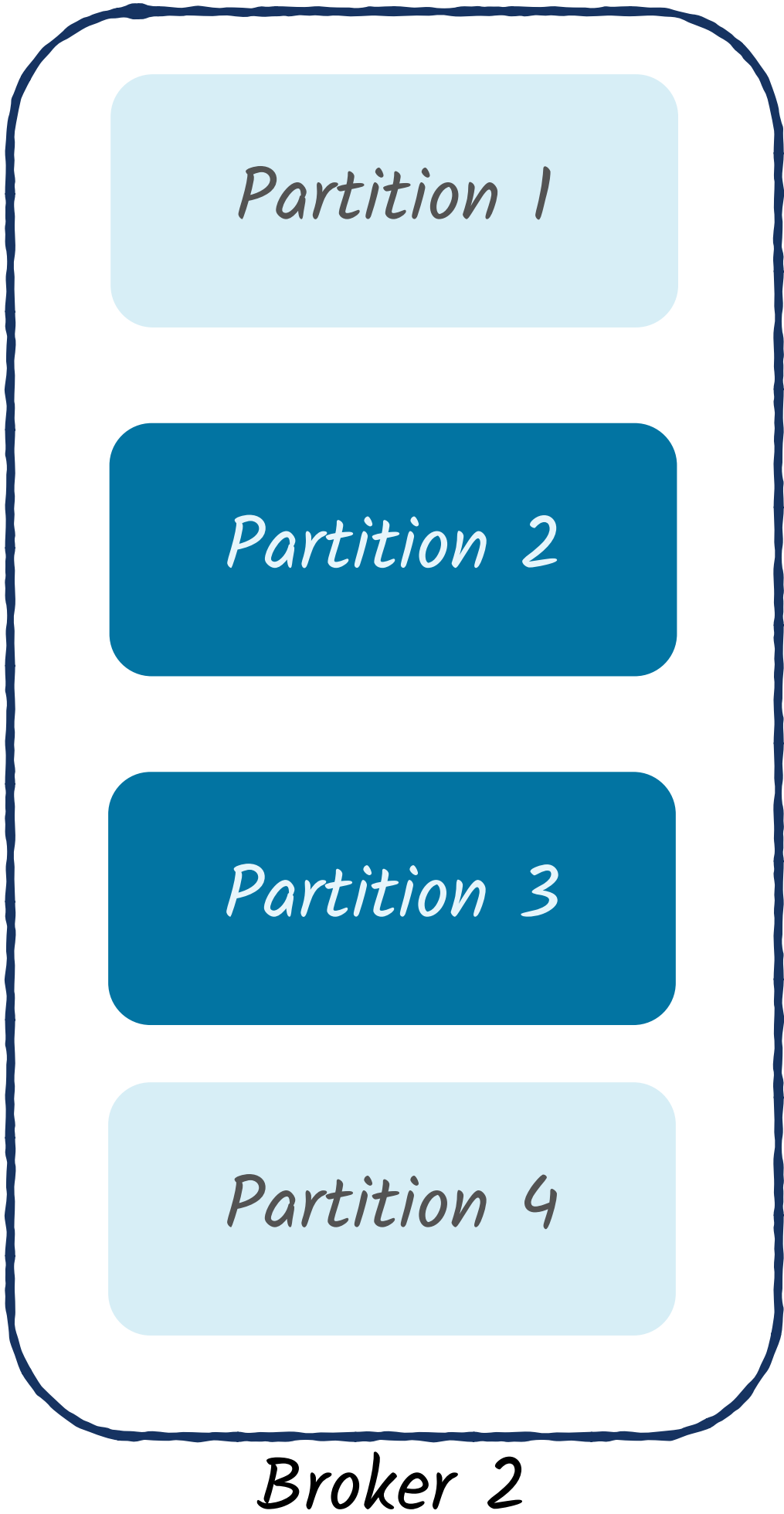
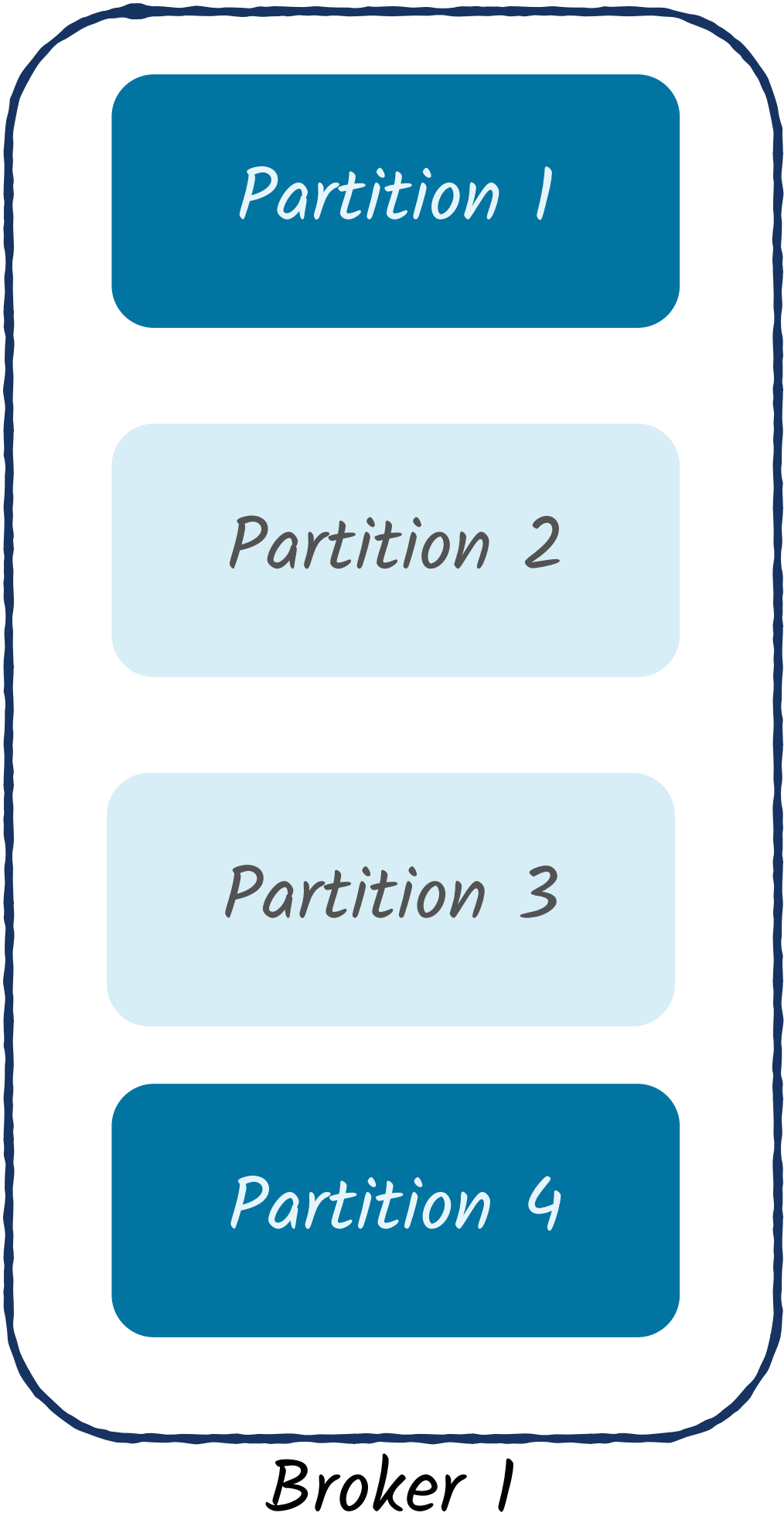
Follower



Partition Leadership and Replication

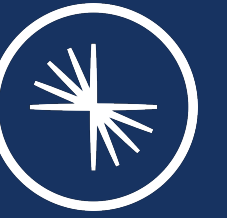
Leader

Follower

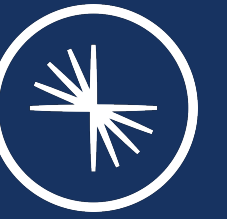




DEMO

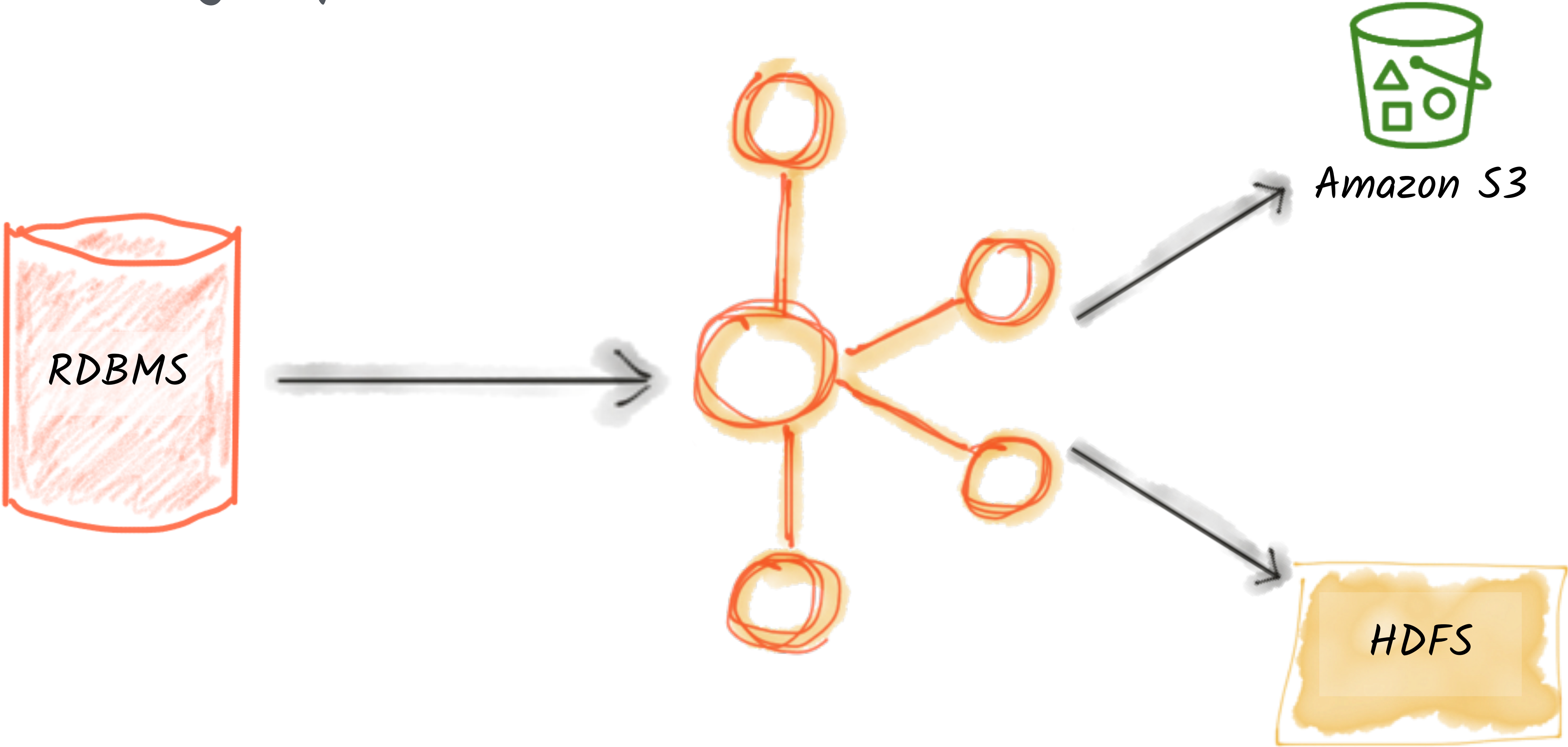


So far, this is
Pretty good

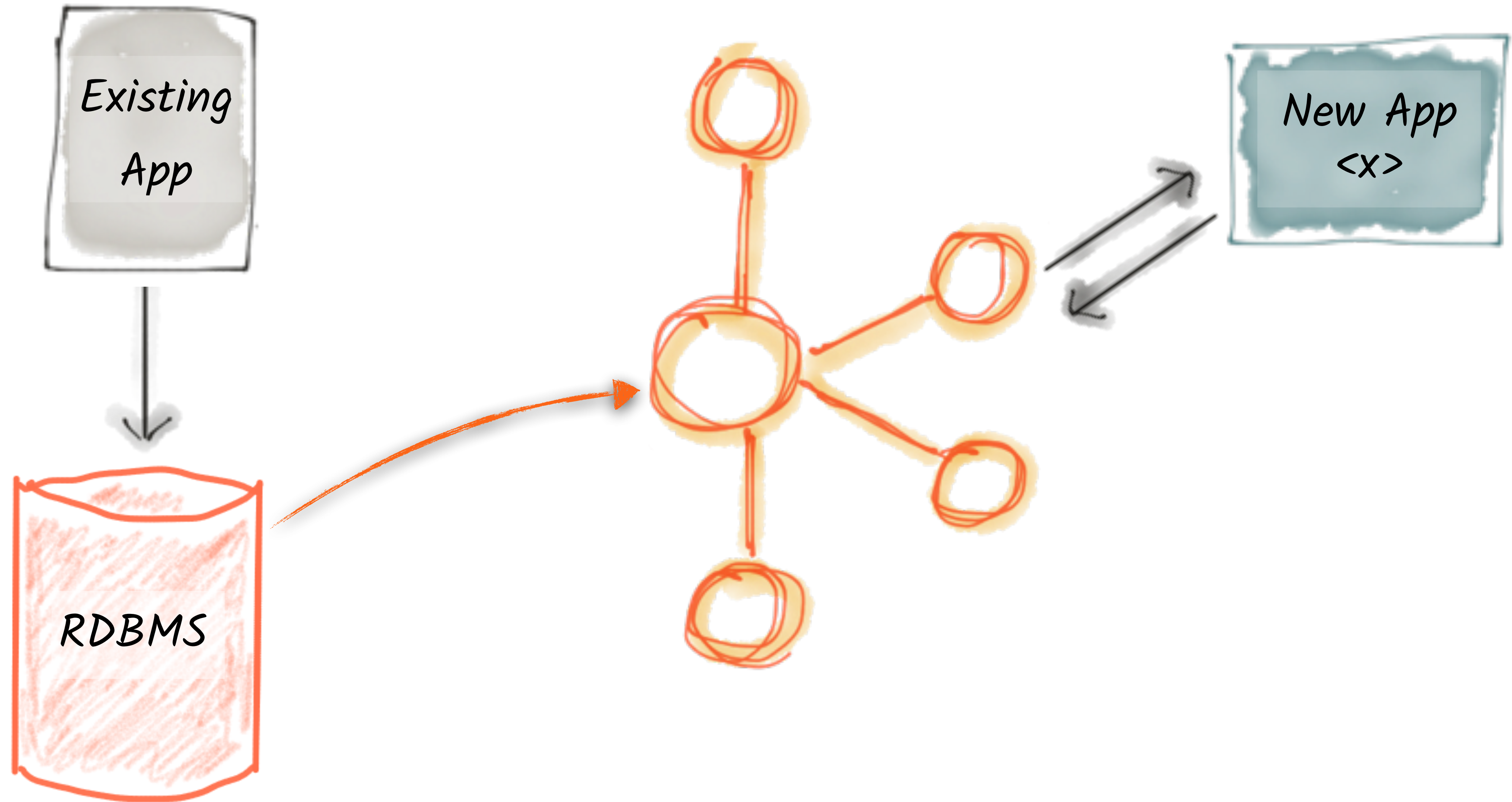


So far, this is
Pretty good
but I've not finished yet...

Streaming Pipelines



Evolve processing from old systems to new

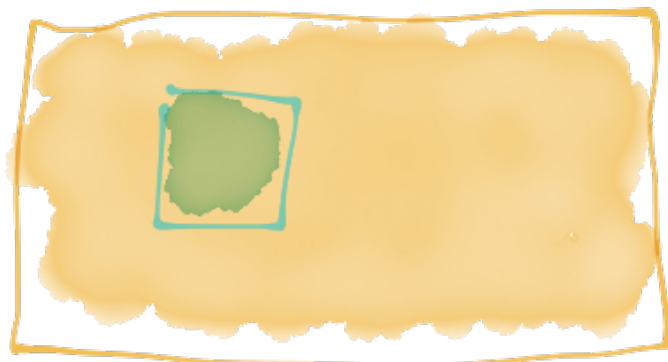
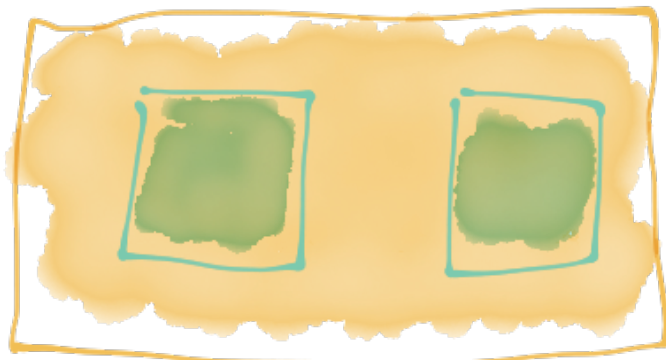




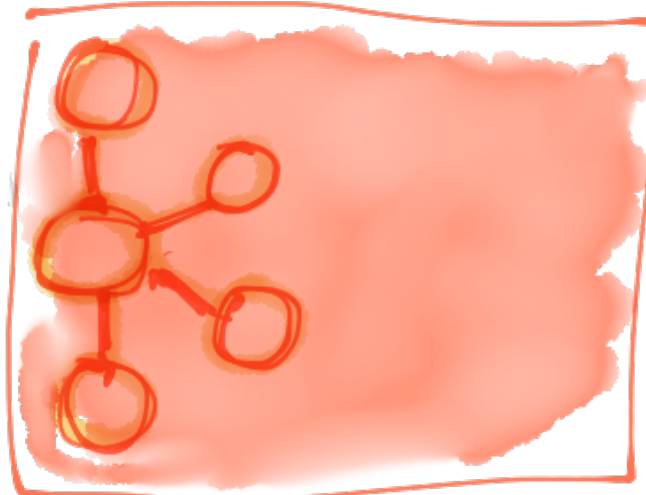
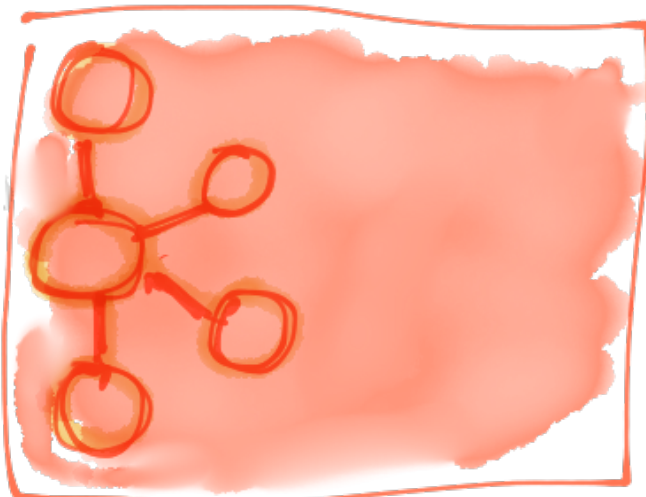
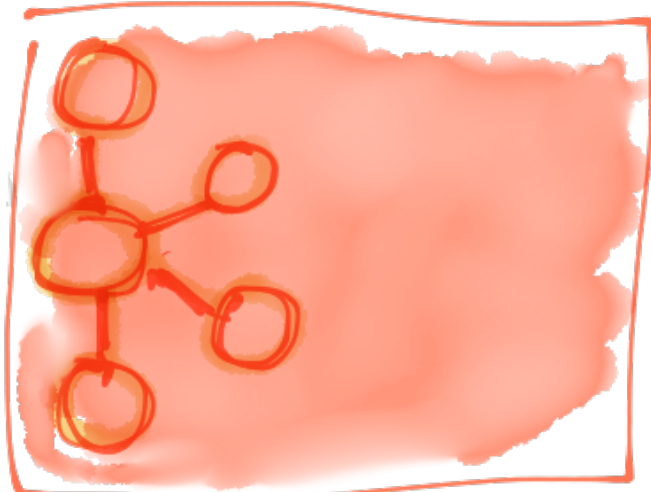
Streaming Integration with Kafka Connect



Sources



Kafka Connect



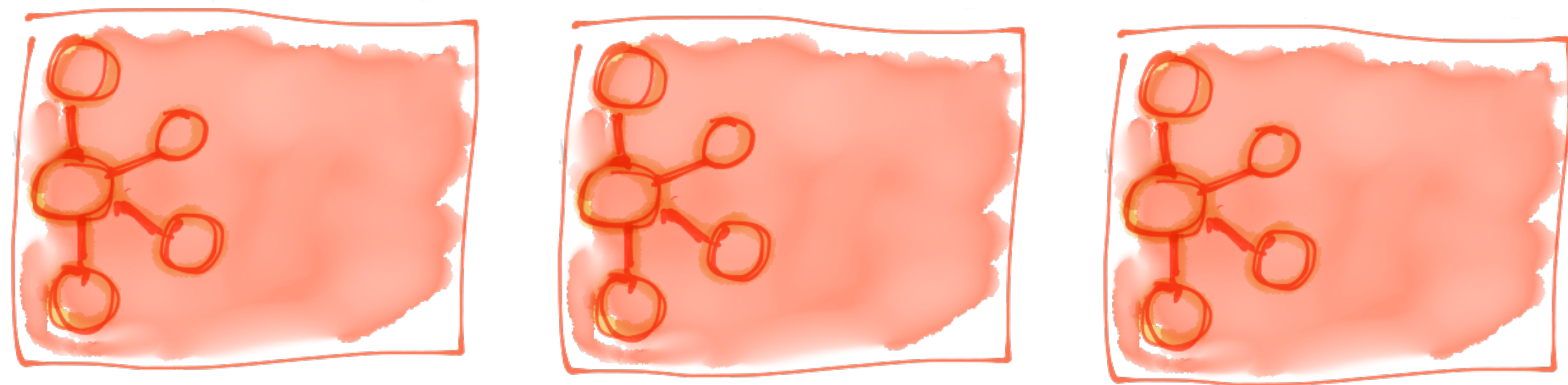
Kafka Brokers

Streaming Integration with Kafka Connect

Sinks



Kafka Connect



Kafka Brokers

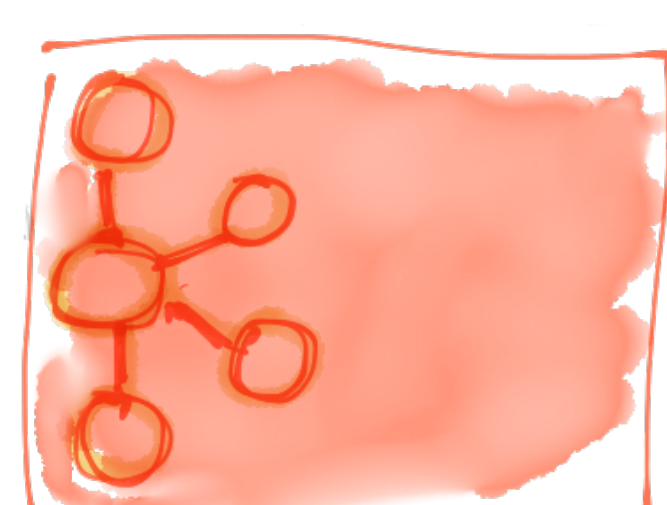
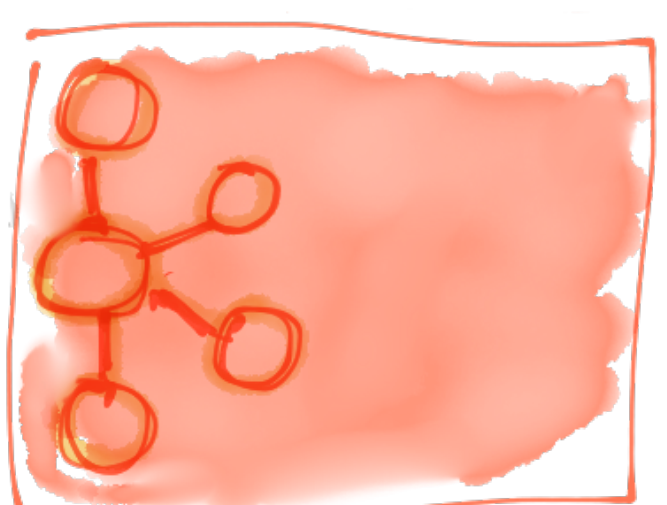
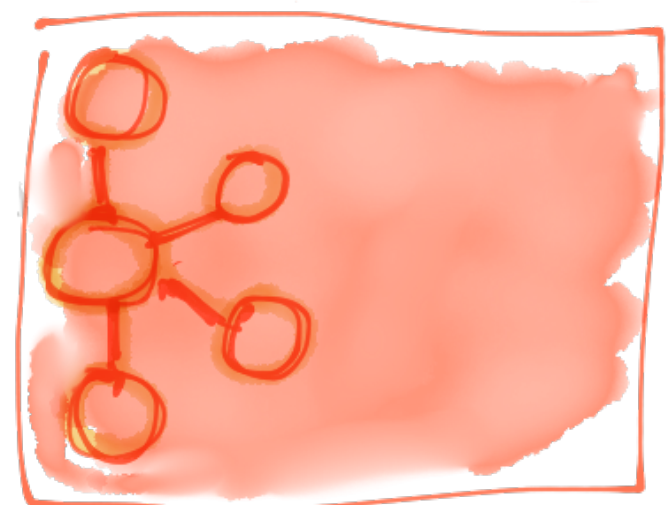
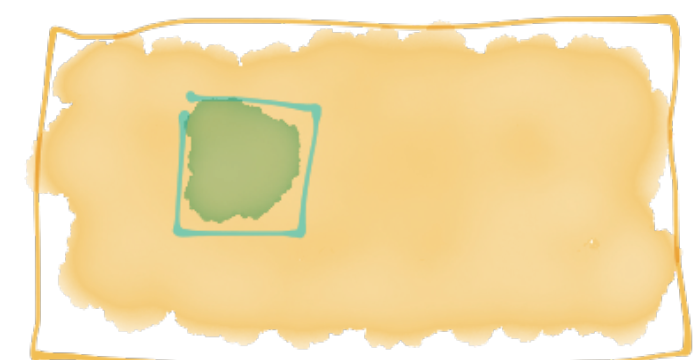
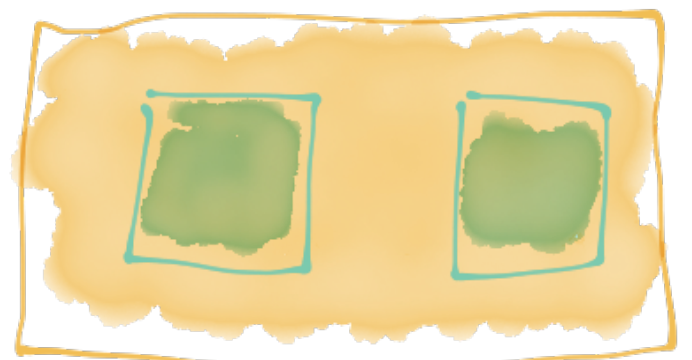
Streaming Integration with Kafka Connect



A collection of logos for various data sources and sinks. Log sources include syslog, http://, CSV, and Microsoft SQL Server. Sinks include JSON, MQTT, MySQL, ORACLE, and Salesforce. An elephant icon is also present.



A collection of logos for various data stores and destinations. Stores include mongoDB, elasticsearch, influxdb, neo4j, and splunk. Destinations include Amazon, snowflake, IBM MQ, Google, ORACLE, hadoop HDFS, and MQTT. Other logos include Java JDBC, Salesforce, and http://.



Kafka Connect

Kafka Brokers

Look Ma, No Code!

{

"connector.class":

"io.confluent.connect.jdbc.JdbcSourceConnector",

"connection.url":

"jdbc:mysql://asgard:3306/demo",

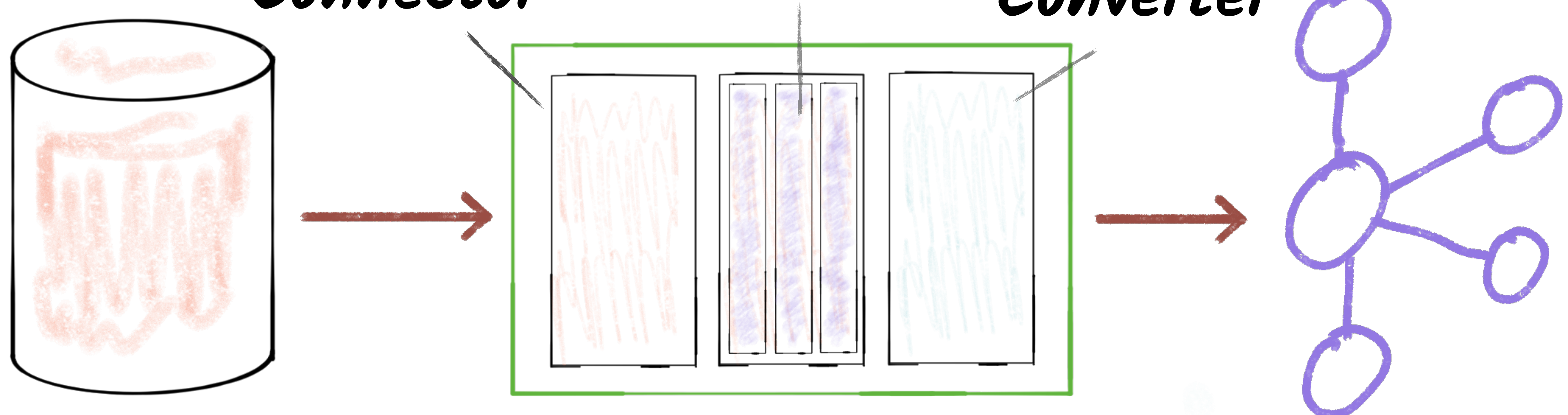
"table.whitelist":

"sales,orders,customers"

}

Extensible

Connector Transform(s) Converter



hub.confluent.io

Confluent Hub

Discover Kafka® connectors
and more

🔍 jdbc

Filters

Plugin type ⓘ

☐ Sink

☐ Source

☐ Transform

☐ Converter

Enterprise support ⓘ

☐ Confluent supported

☐ Partner supported

☐ None

Results (1)

+ Submit a plugin

Kafka Connect JDBC

SINK, SOURCE CONNECTOR

The JDBC source and sink connectors allow you to exchange data between relational databases and Kafka. The JDBC source connector allows you to import data from any relational database with a JDBC driver into Kafka topics

Enterprise support:
Confluent supported

Installation:
Confluent Hub CLI, Download

Verification:
Confluent built

Author:
Confluent, Inc.

License:
Free

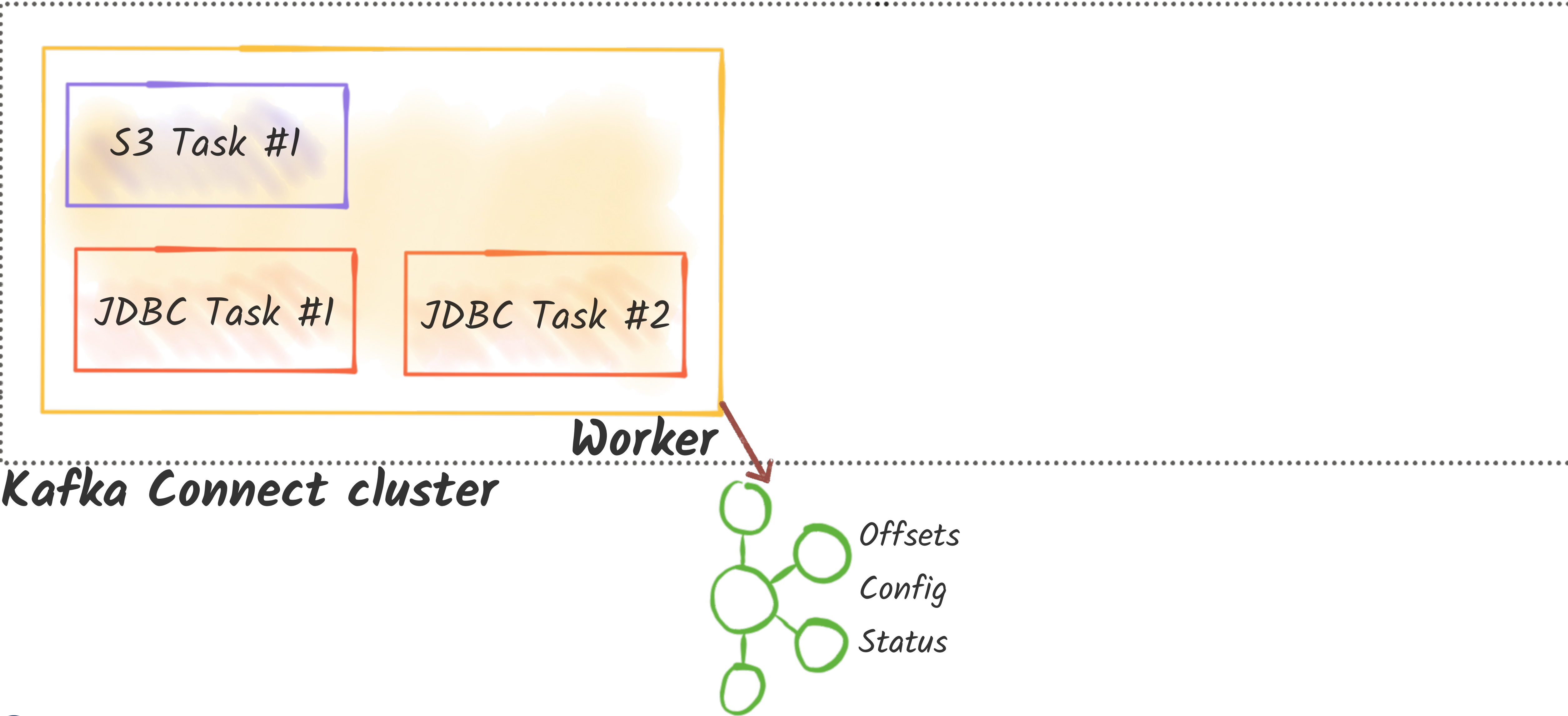
Version:
5.4.1



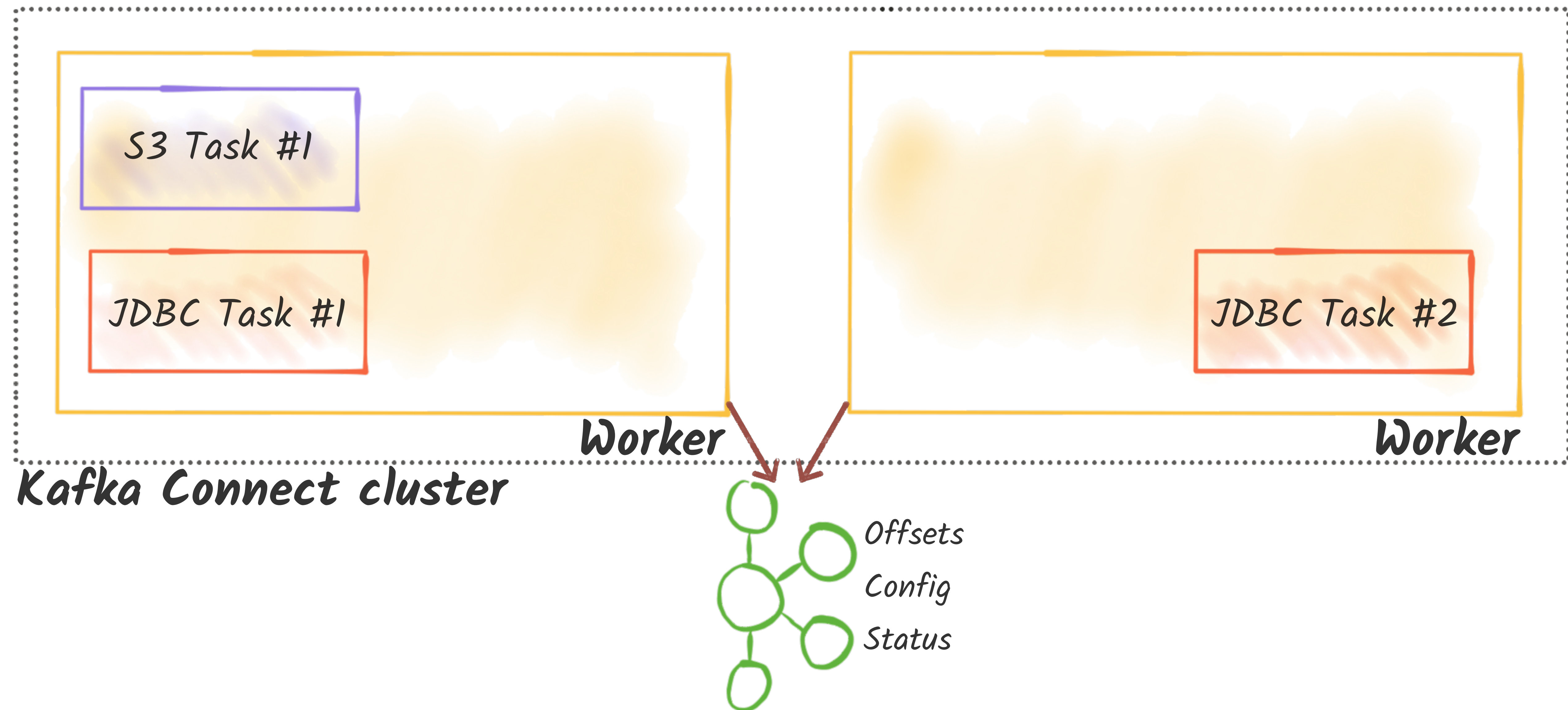
 CONFLUENT

@rmoff

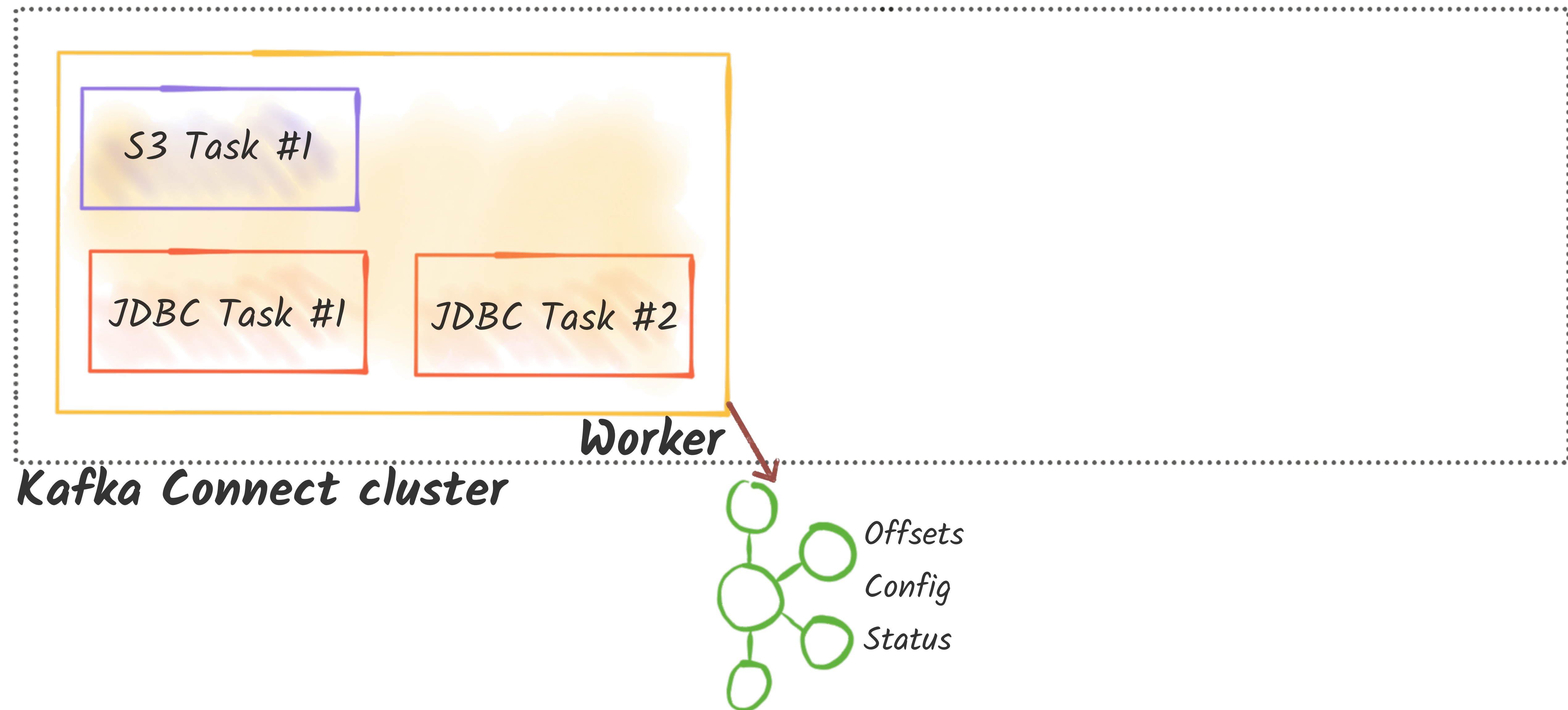
Single Kafka Connect node

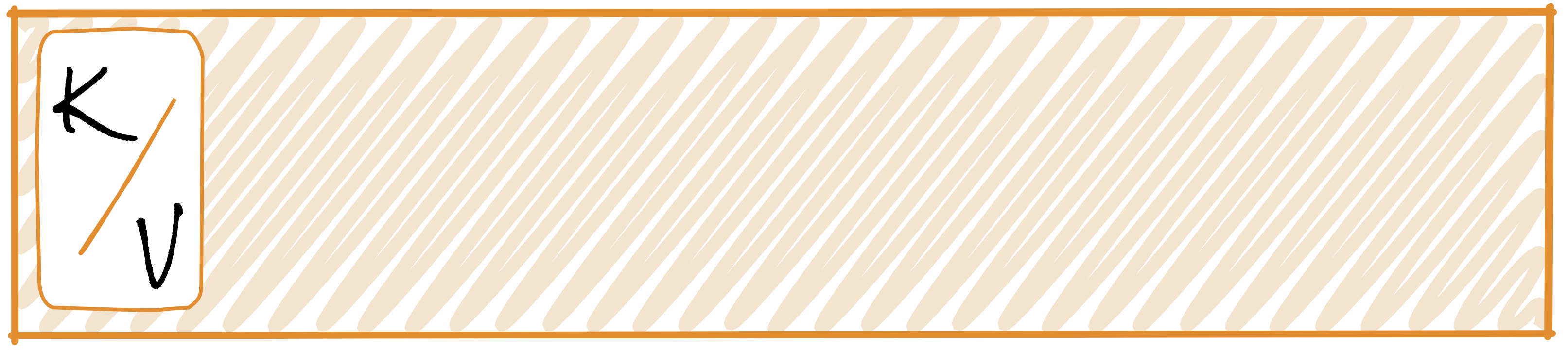


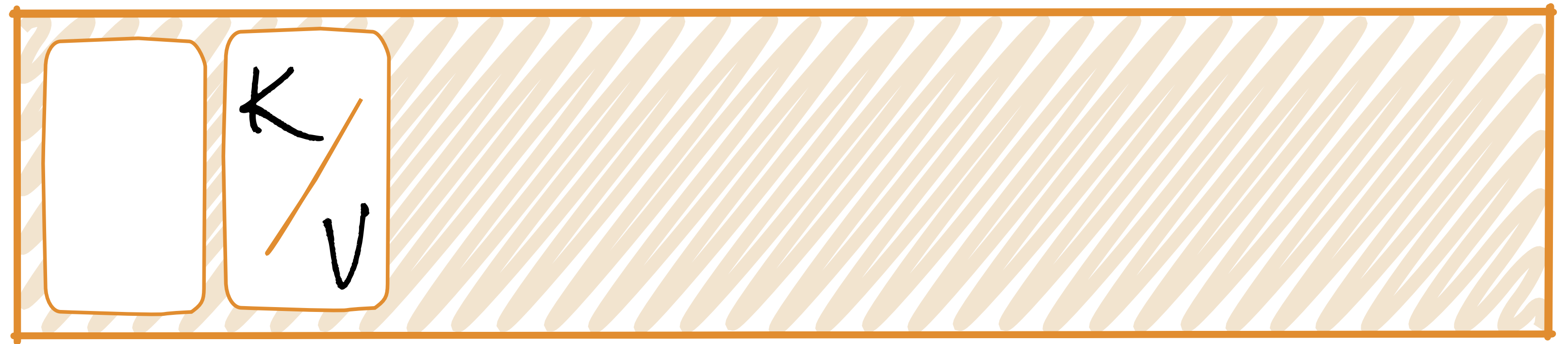
Kafka Connect - scalable and fault-tolerant

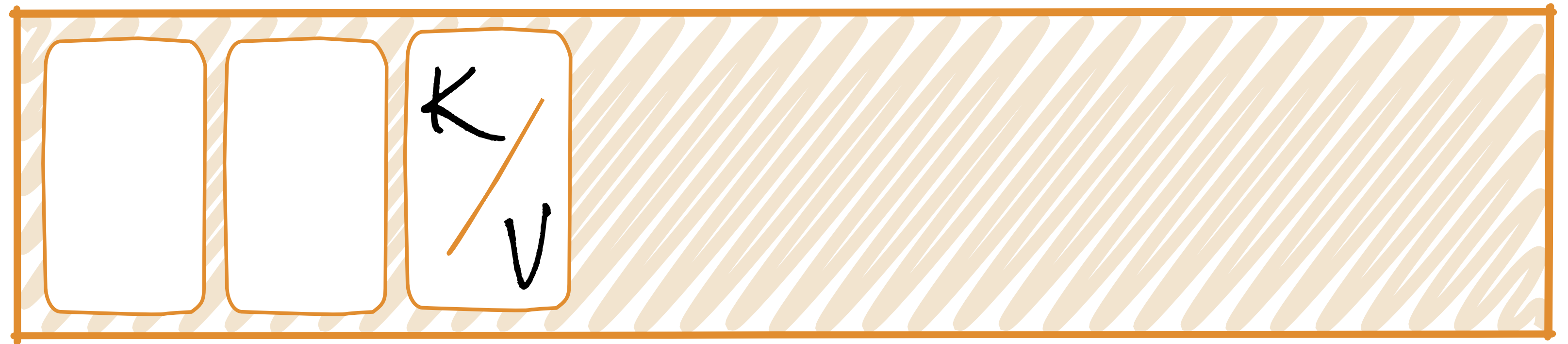


Automatic fault tolerance

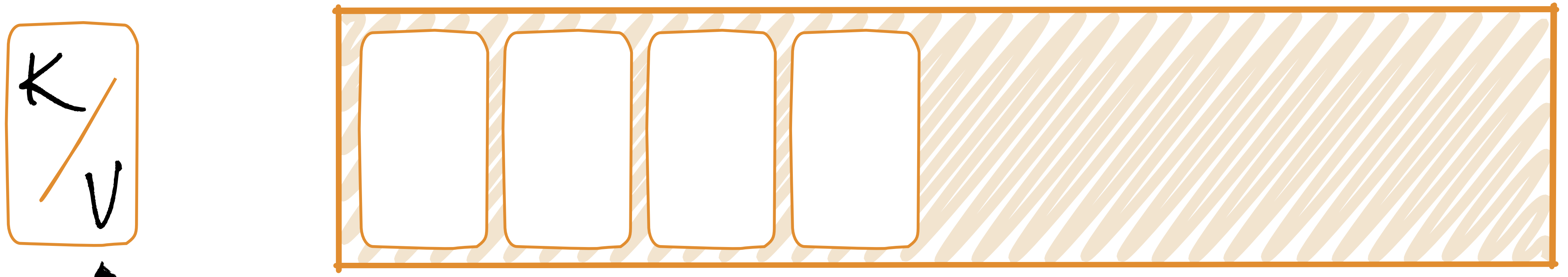










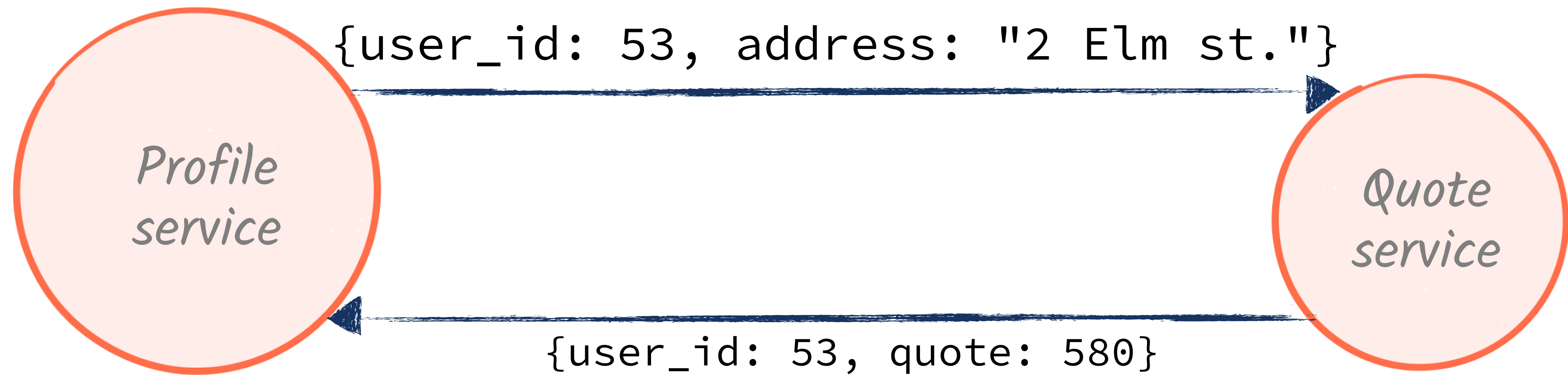


Wait...what's this?

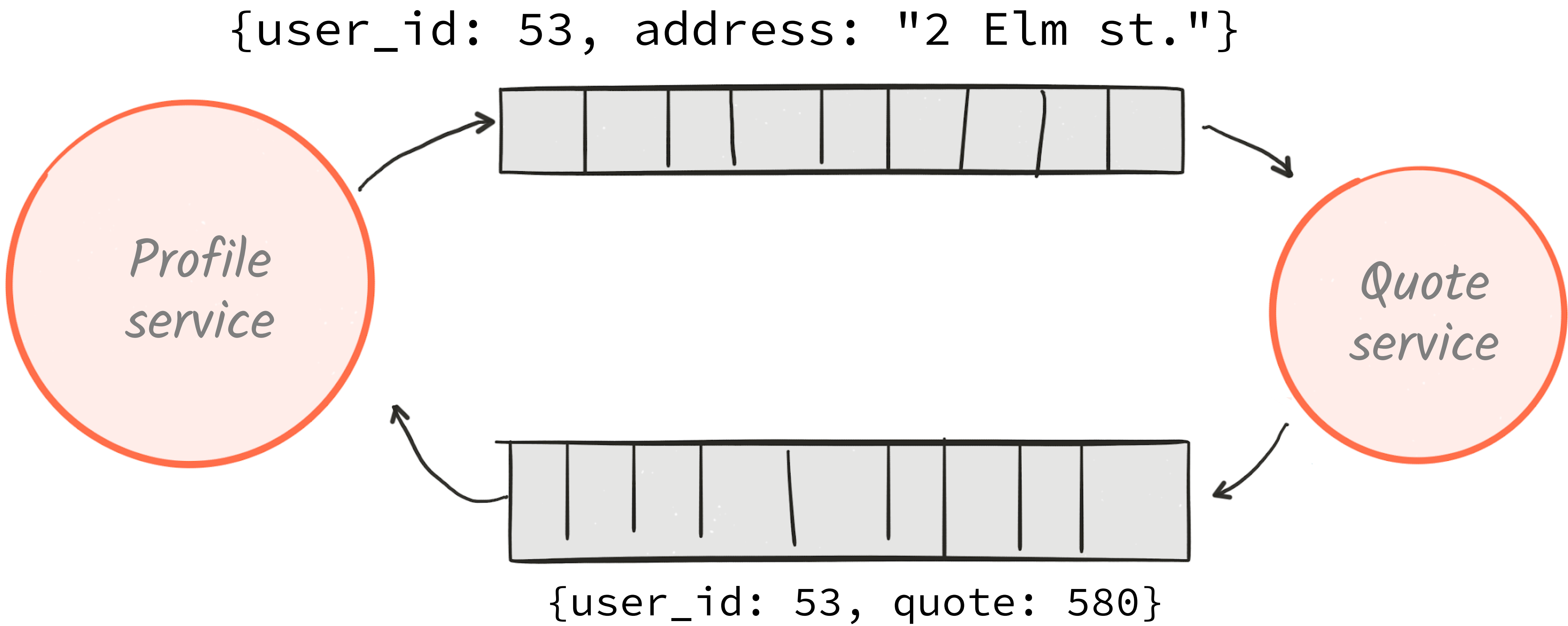
How do you serialise your data?

Avro Protobuf JSON
Schema JSON CSV

APIs are contracts between services

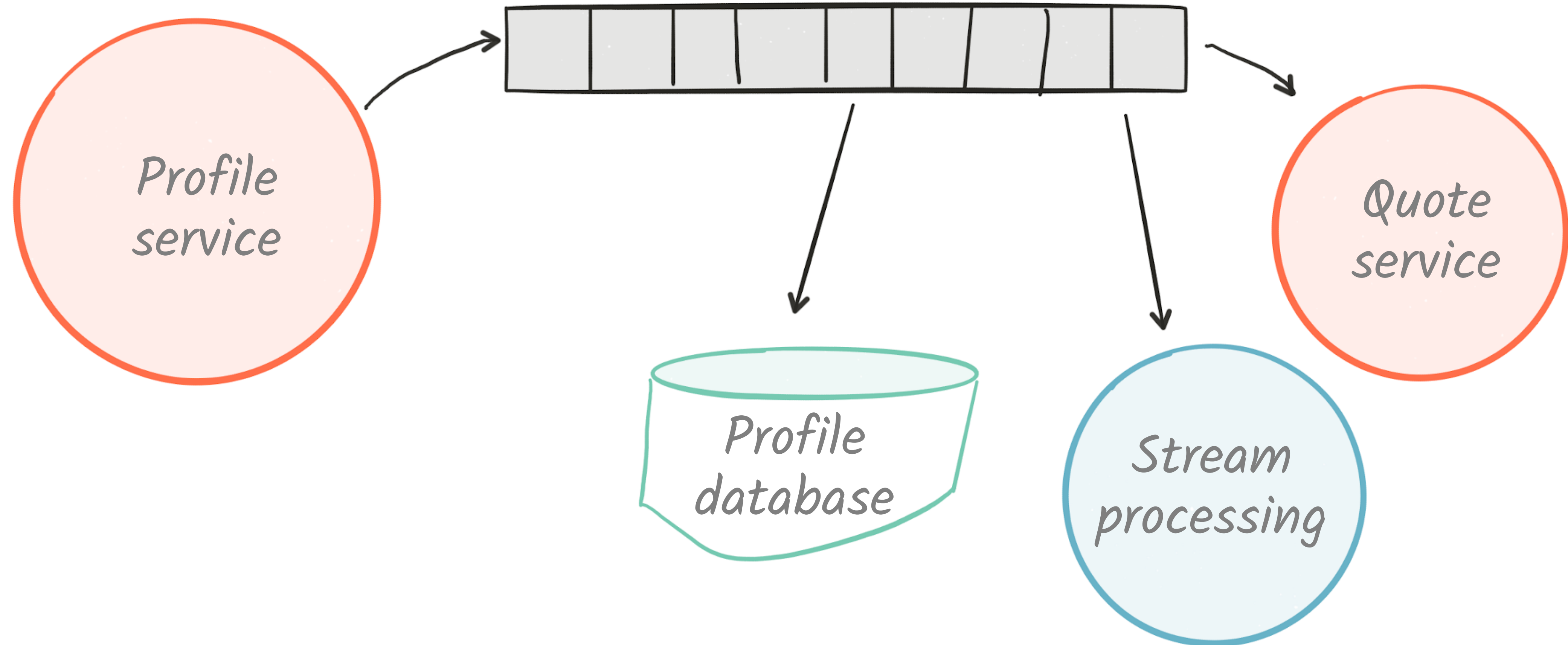


But not all services

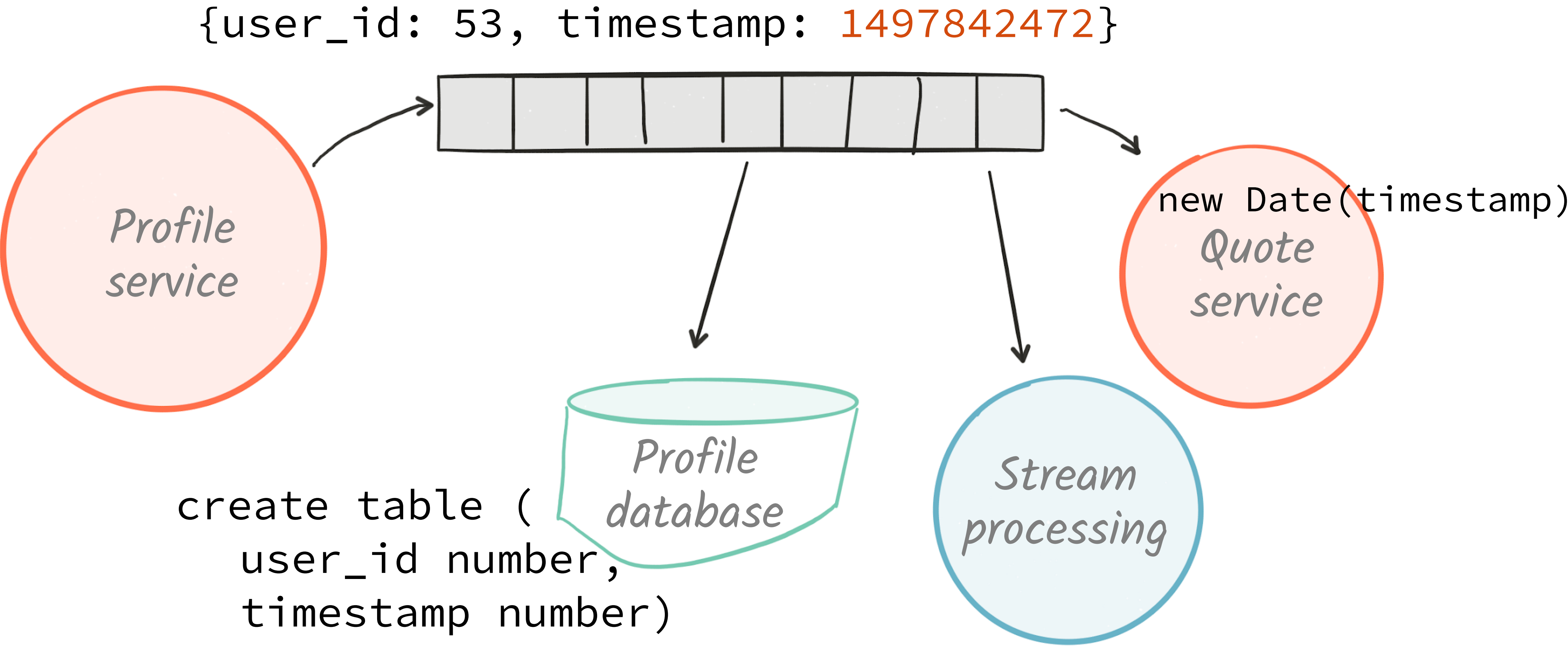


And naturally...

`{user_id: 53, address: "2 Elm st."}`

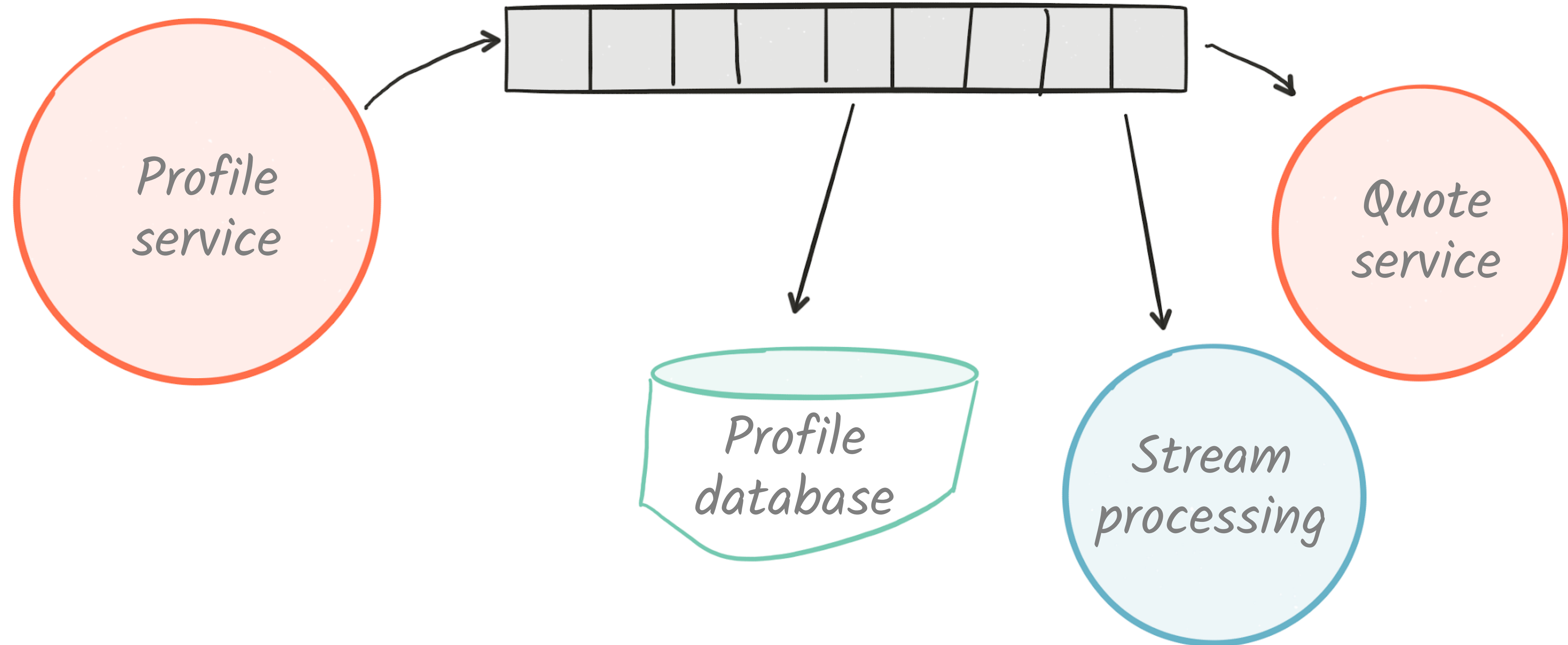


Schemas are about how teams work together



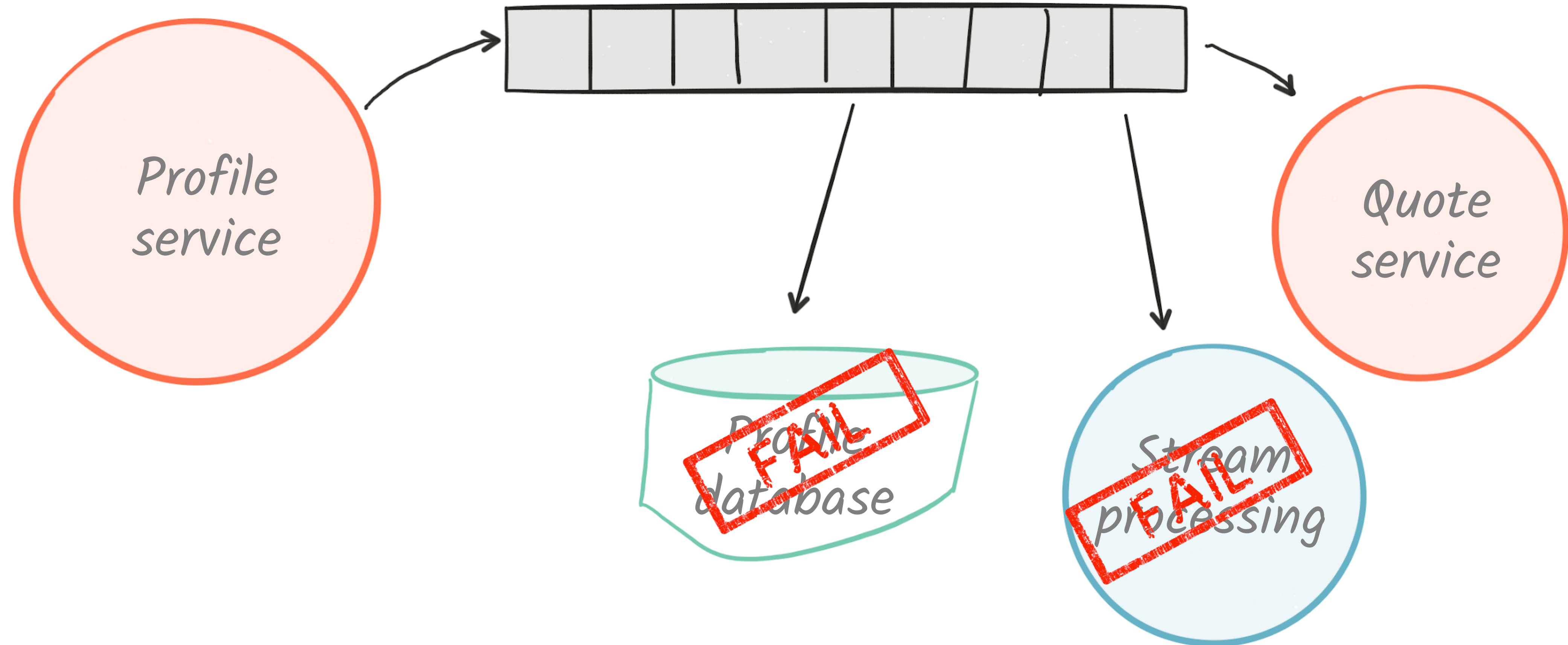
Things change...

{user_id: 53, timestamp: "June 28, 2017 4:00pm"}



Moving fast and breaking things

{user_id: 53, timestamp: "June 28, 2017 4:00pm"}



Lack of schemas – Coupling teams and services

2001 2001 Citrus Heights–Sunrise Blvd
Citrus_Hghts 60670001 3400293 34 SAC
Sacramento SV Sacramento Valley SAC
Sacramento County APCD SMA8 Sacramento
Metropolitan Area CA 6920 Sacramento 28 6920
13588 7400 Sunrise Blvd 95610 38 41 56
38.6988889 121 16 15.98999977
–121.271111 10 4284781 650345 52

Serialisation & Schemas

Avro Protobuf JSON Schema JSON CSV

Serialisation & Schemas

Avro



Protobuf



JSON
Schema



JSON



CSV

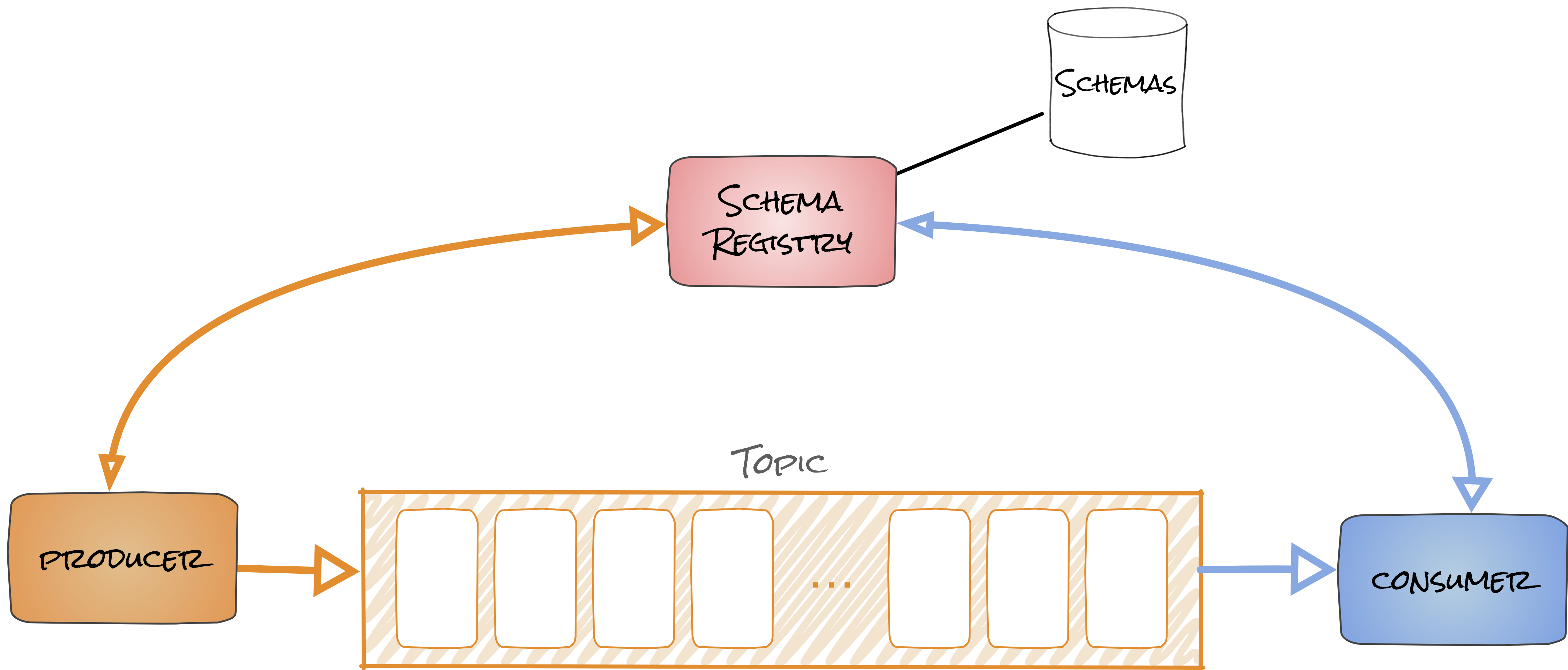


Gwen (Chen) Shapira
@gwenshap

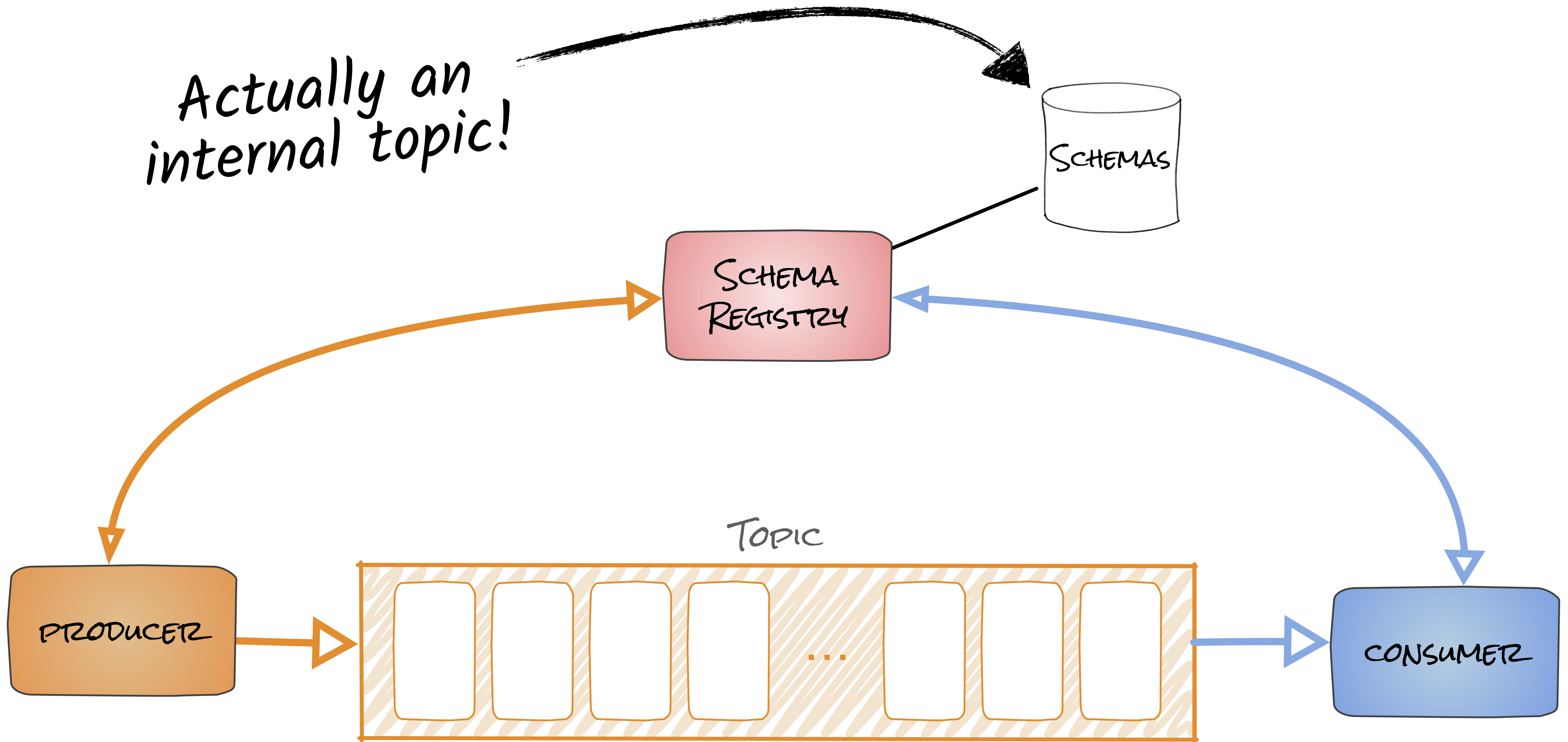
If your dev process doesn't validate schema compatibility somewhere between your IDE and production - you are screwed and don't know it.

5:50 AM - 5 Apr 2017

<https://rmoff.dev/qcon-schemas>

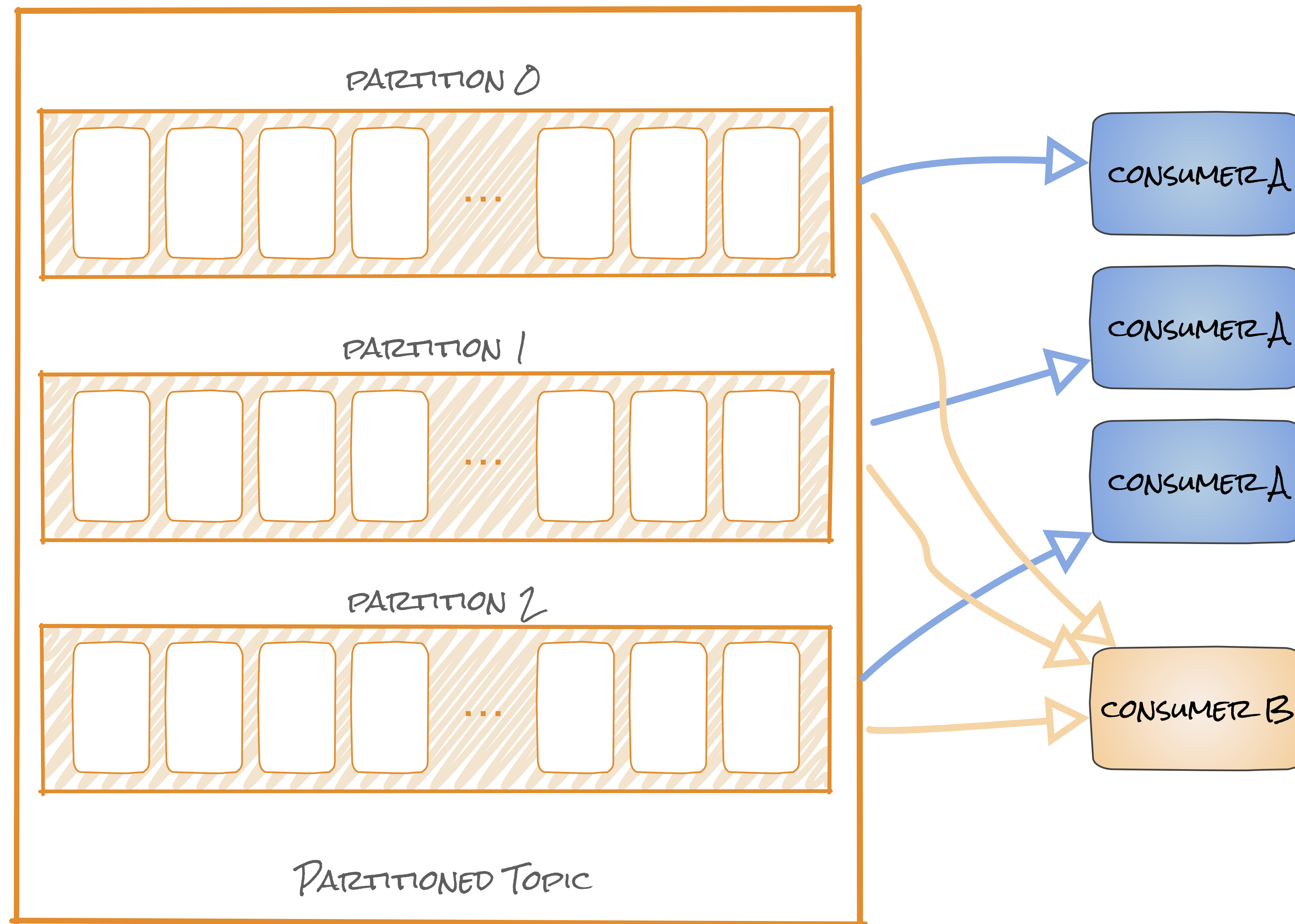


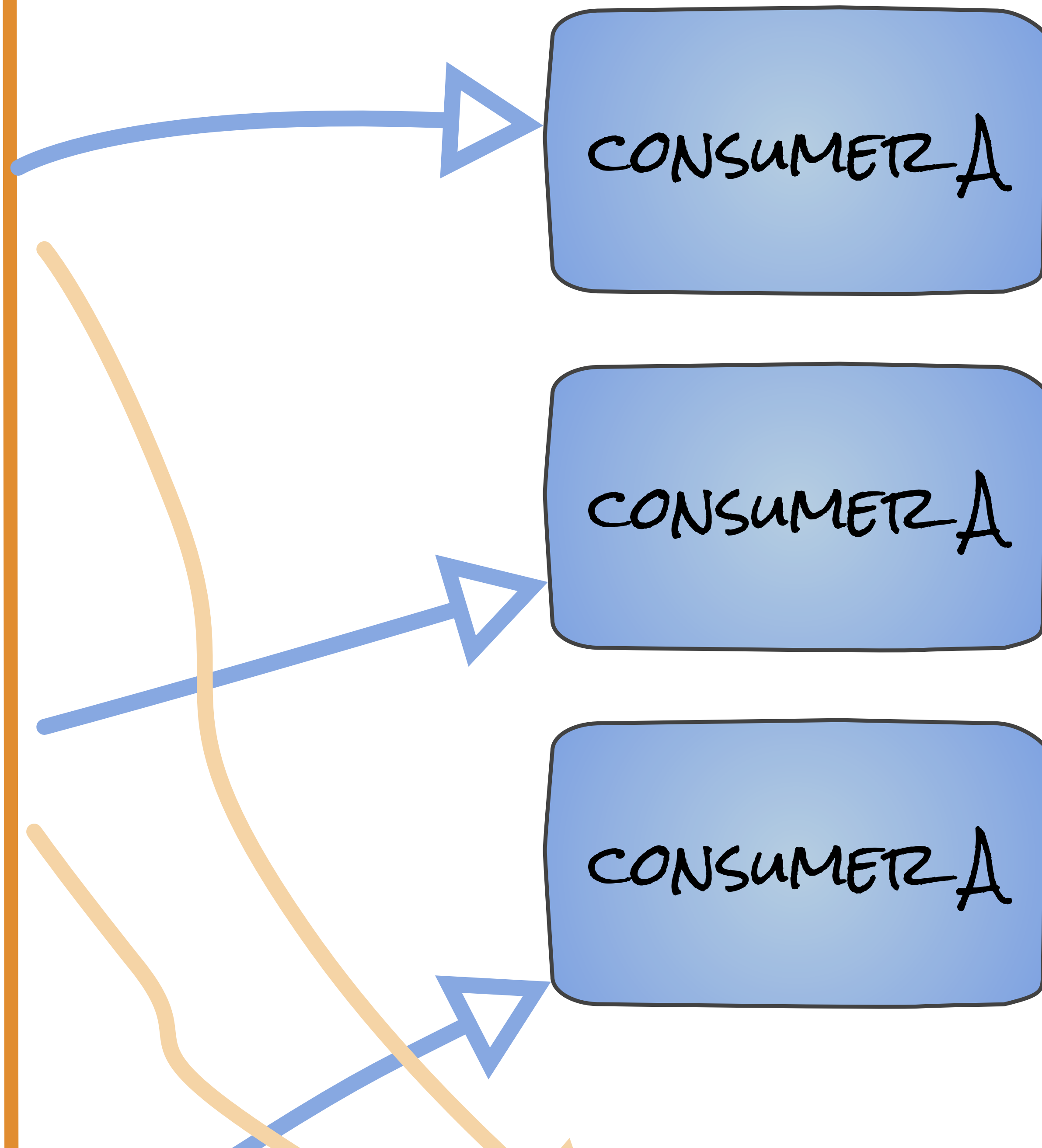
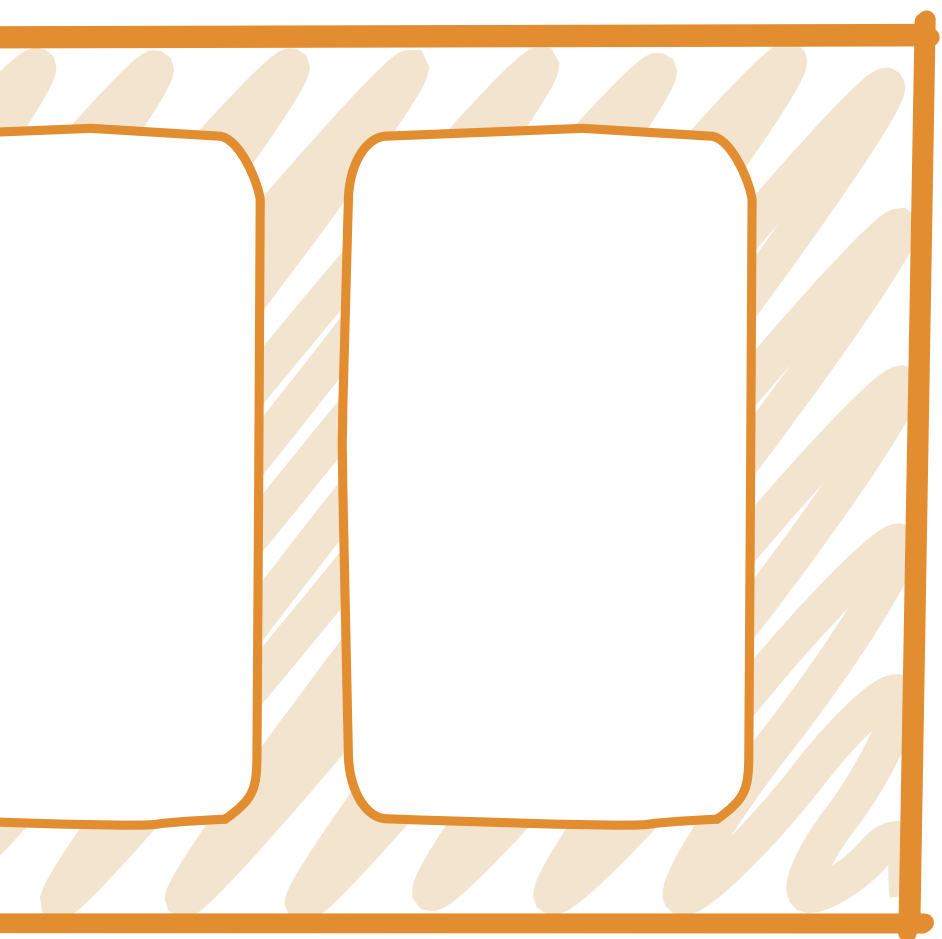
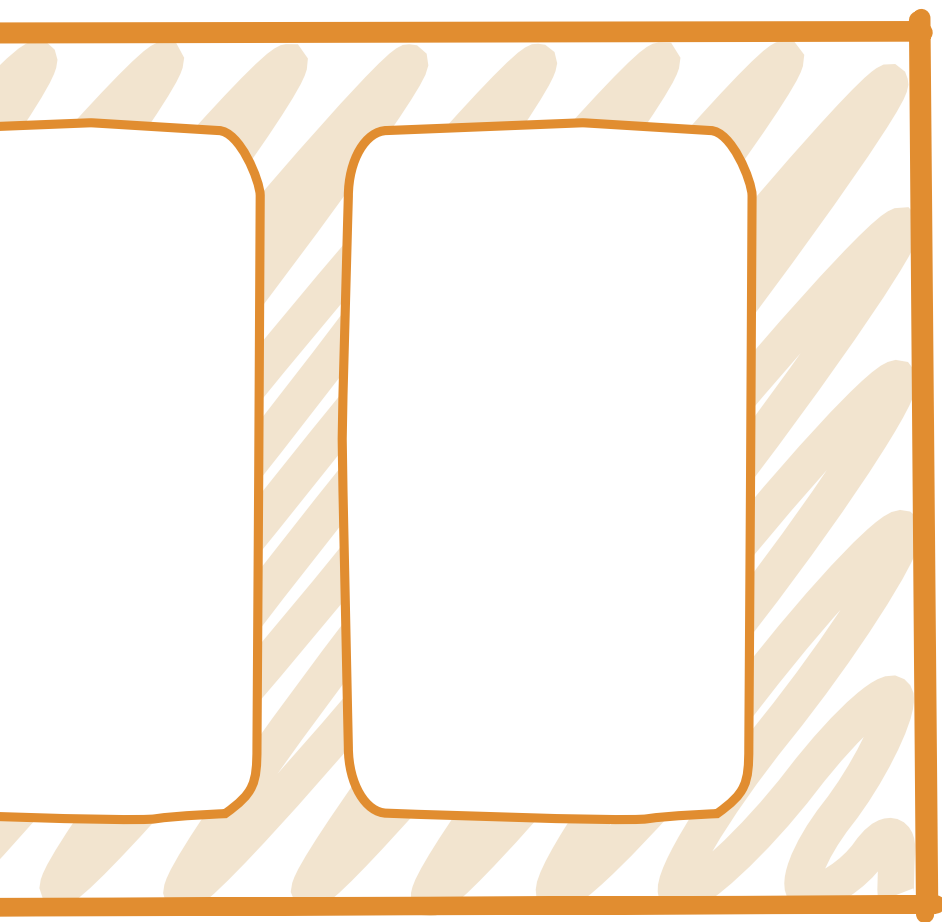
Actually an
internal topic!



Producers contain serializers

```
props.put("key.serializer",  
    "org.apache.kafka.serializers.StringSerializer");  
props.put("value.serializer",  
    "io.confluent.kafka.serializers.KafkaAvroSerializer");  
props.put("schema.registry.url",  
    "http://schema-registry:8081");  
...  
producer<String, LogLine> producer = new  
    KafkaProducer<String, LogLine>(props);
```

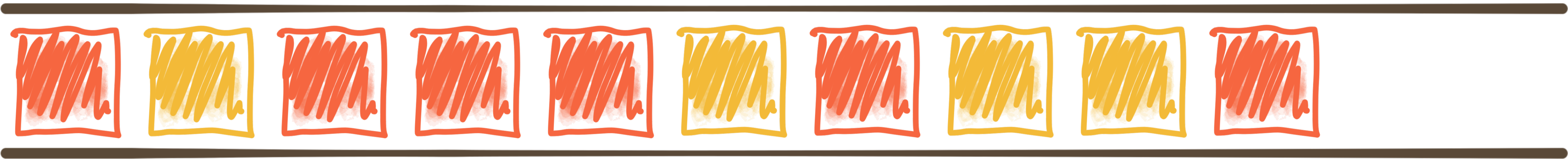






```
{  
  "reading_ts": "2020-02-14T12:19:27Z",  
  "sensor_id": "aa-101",  
  "production_line": "w01",  
  "widget_type": "acme94",  
  "temp_celcius": 23,  
  "widget_weight_g": 100  
}
```

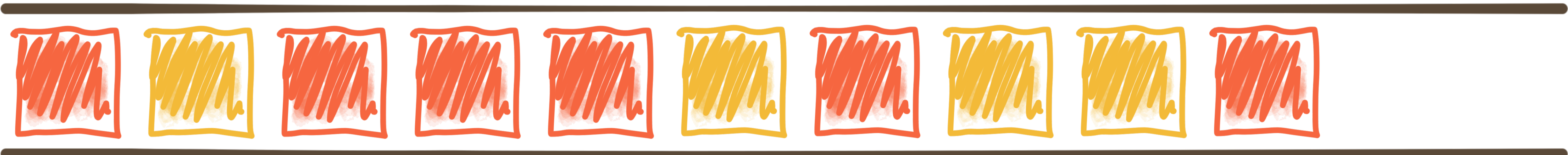


Streams of events



Time →

Stream Processing



Stream: widgets



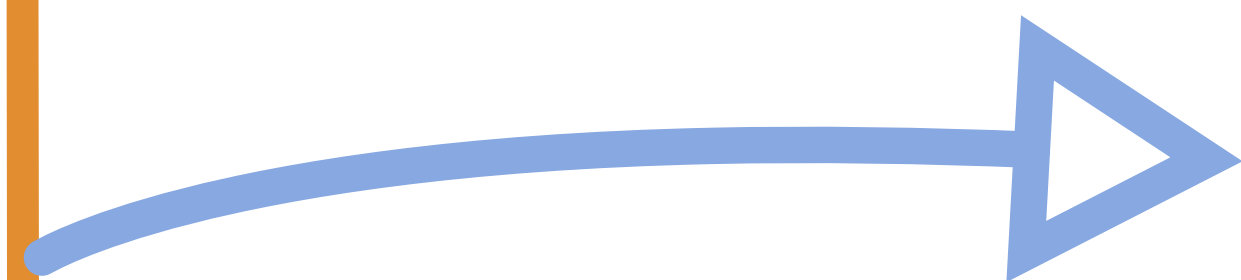
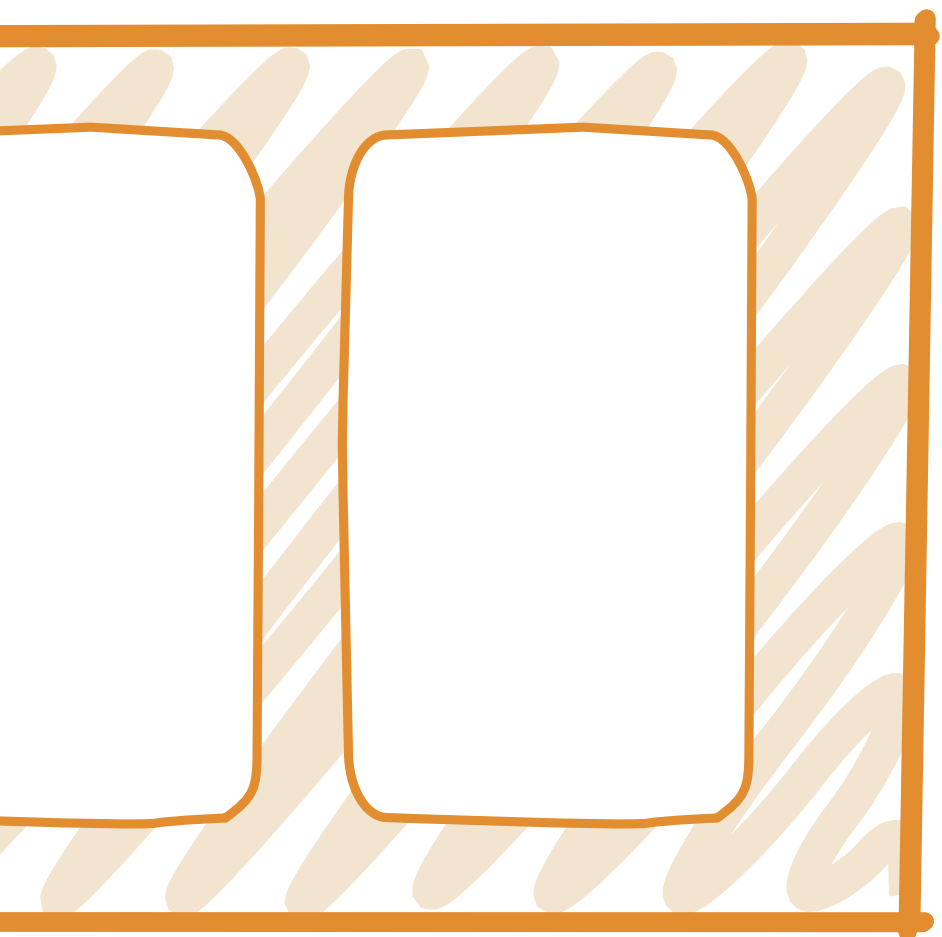
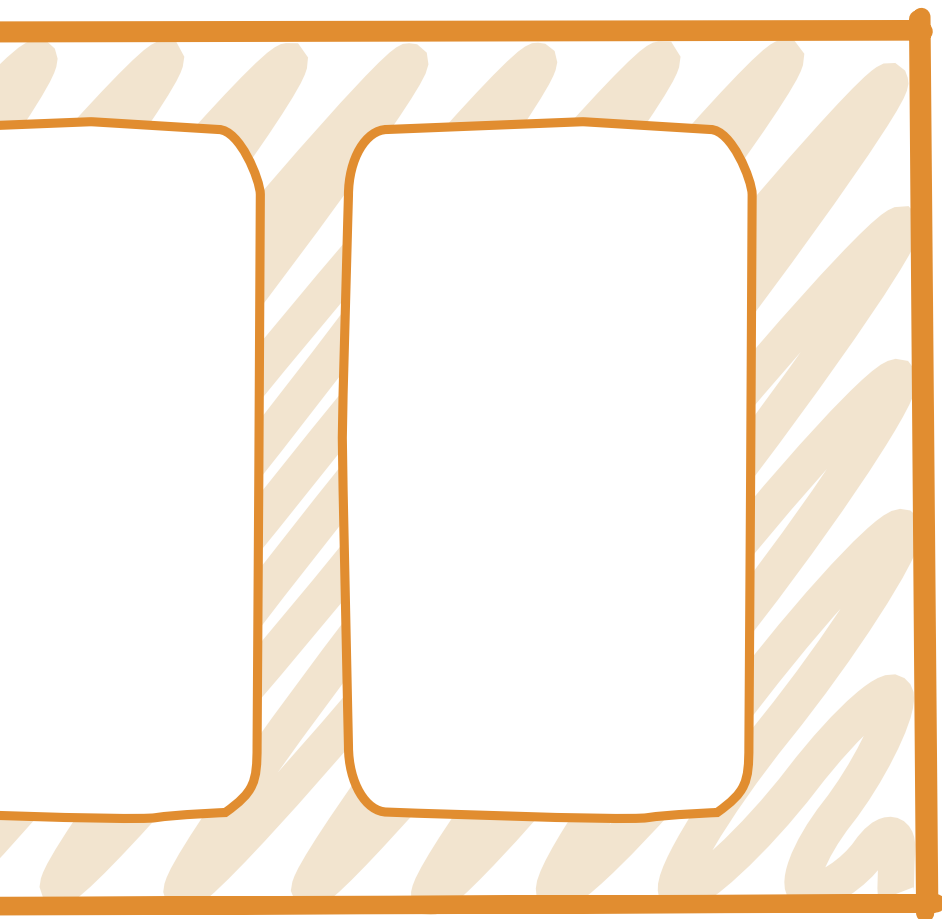
Stream: widgets_red

Stream Processing with Kafka Streams

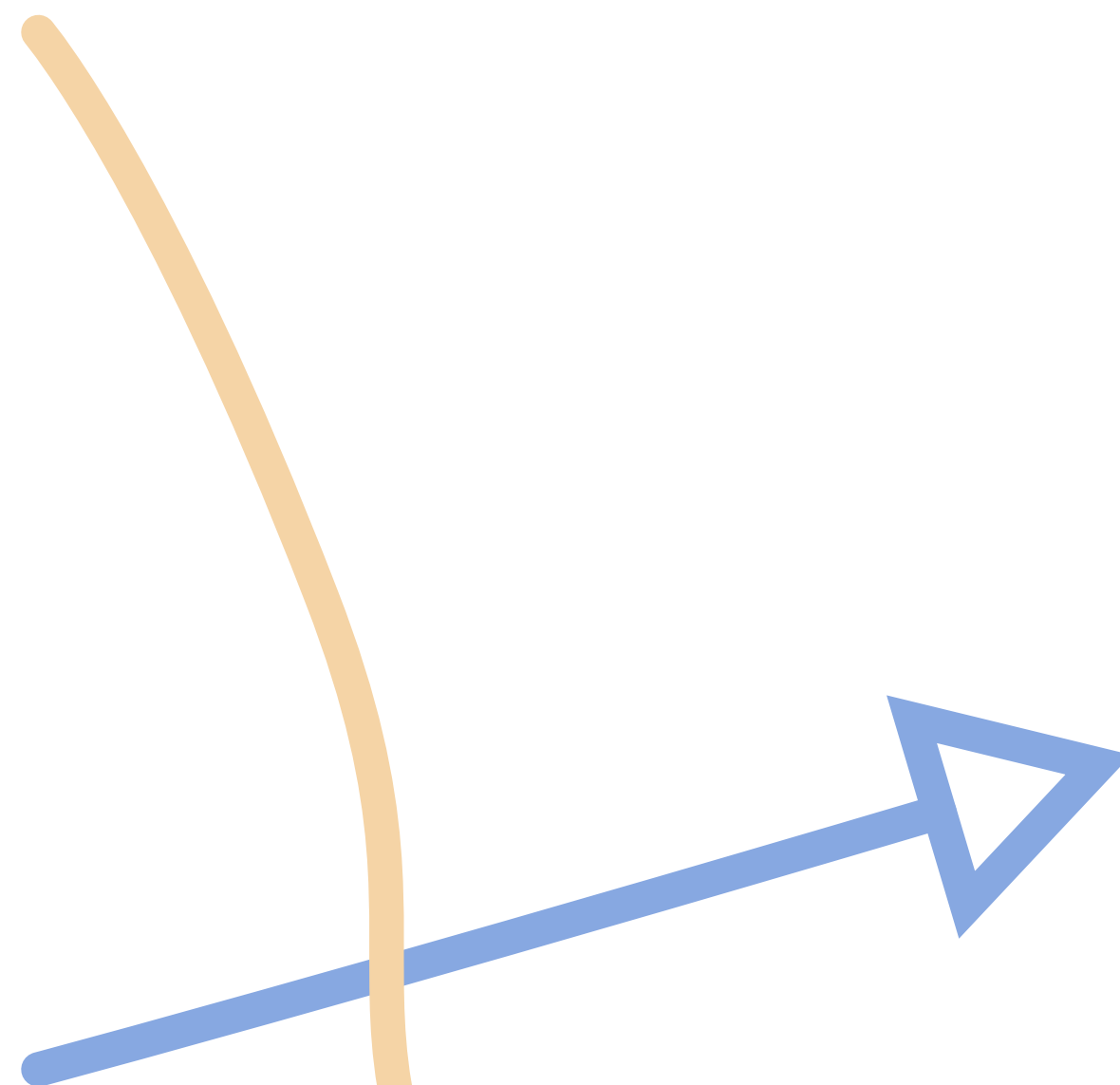


```
final StreamsBuilder builder = new StreamsBuilder()
    .stream("widgets", Consumed.with(stringSerde, widgetsSerde))
    .filter( (key, widget) -> widget.getColour().equals("RED") )
    .to("widgets_red", Produced.with(stringSerde, widgetsSerde));
```





STREAMS
APPLICATION



STREAMS
APPLICATION



STREAMS
APPLICATION

Stream Processing with ksqlDB



ksqlDB

```
CREATE STREAM widgets_red AS
SELECT * FROM widgets
WHERE colour='RED';
```




```
{  
  "reading_ts": "2020-02-14T12:19:27Z",  
  "sensor_id": "aa-101",  
  "production_line": "w01",  
  "widget_type": "acme94",  
  "temp_celcius": 23,  
  "widget_weight_g": 100  
}
```





```
SELECT *  
FROM WIDGETS  
WHERE WEIGHT_G > 120
```



```
SELECT COUNT(*)  
FROM WIDGETS  
GROUP BY PRODUCTION_LINE
```



```
SELECT AVG(TEMP_CELCIUS) AS TEMP  
FROM WIDGETS  
GROUP BY SENSOR_ID  
HAVING TEMP>20
```

```
{  
  "reading_ts": "2020-02-14T12:19:27Z",  
  "sensor_id": "aa-101",  
  "production_line": "w01",  
  "widget_type": "acme94",  
  "temp_celcius": 23,  
  "widget_weight_g": 100  
}
```

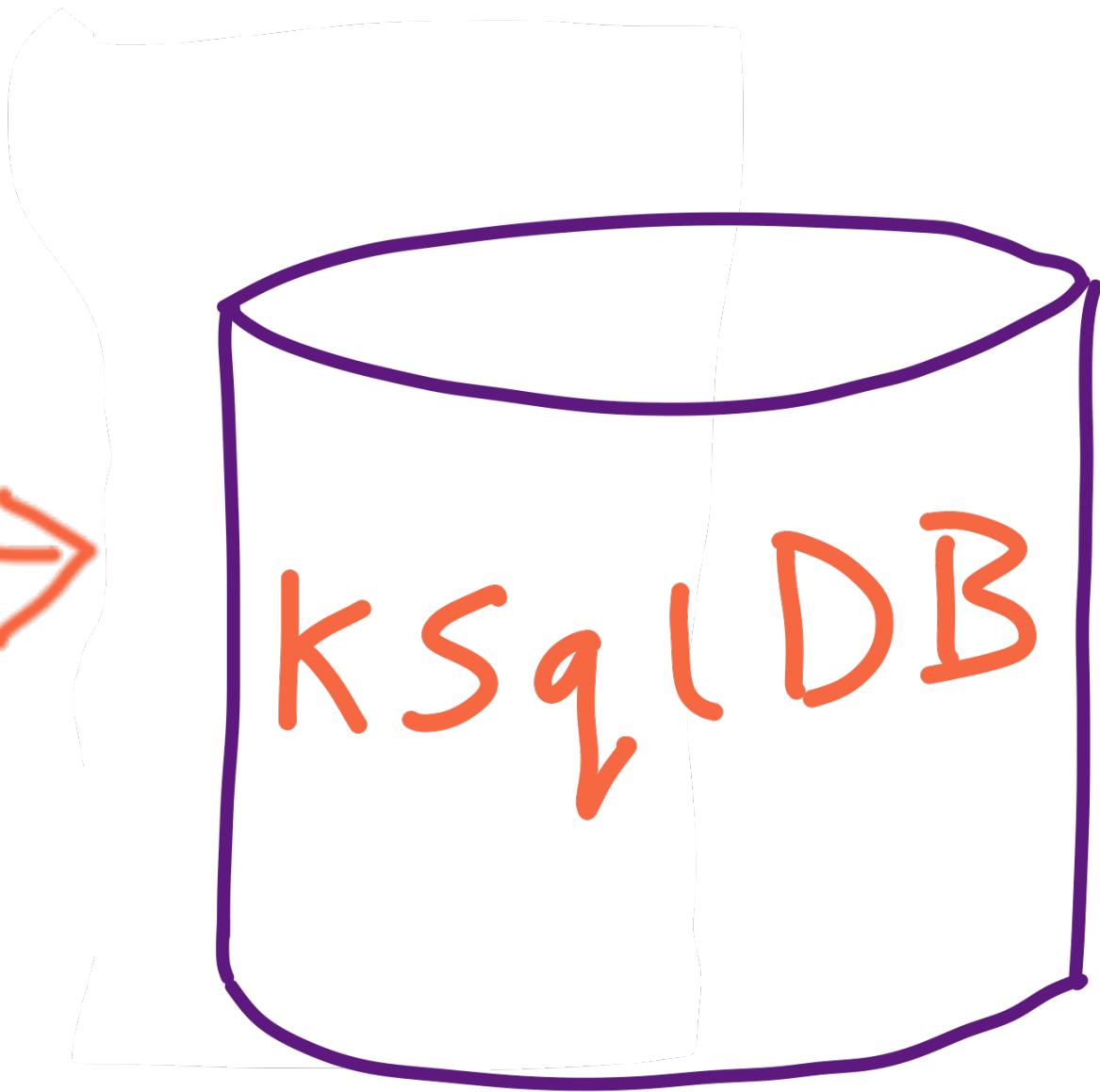
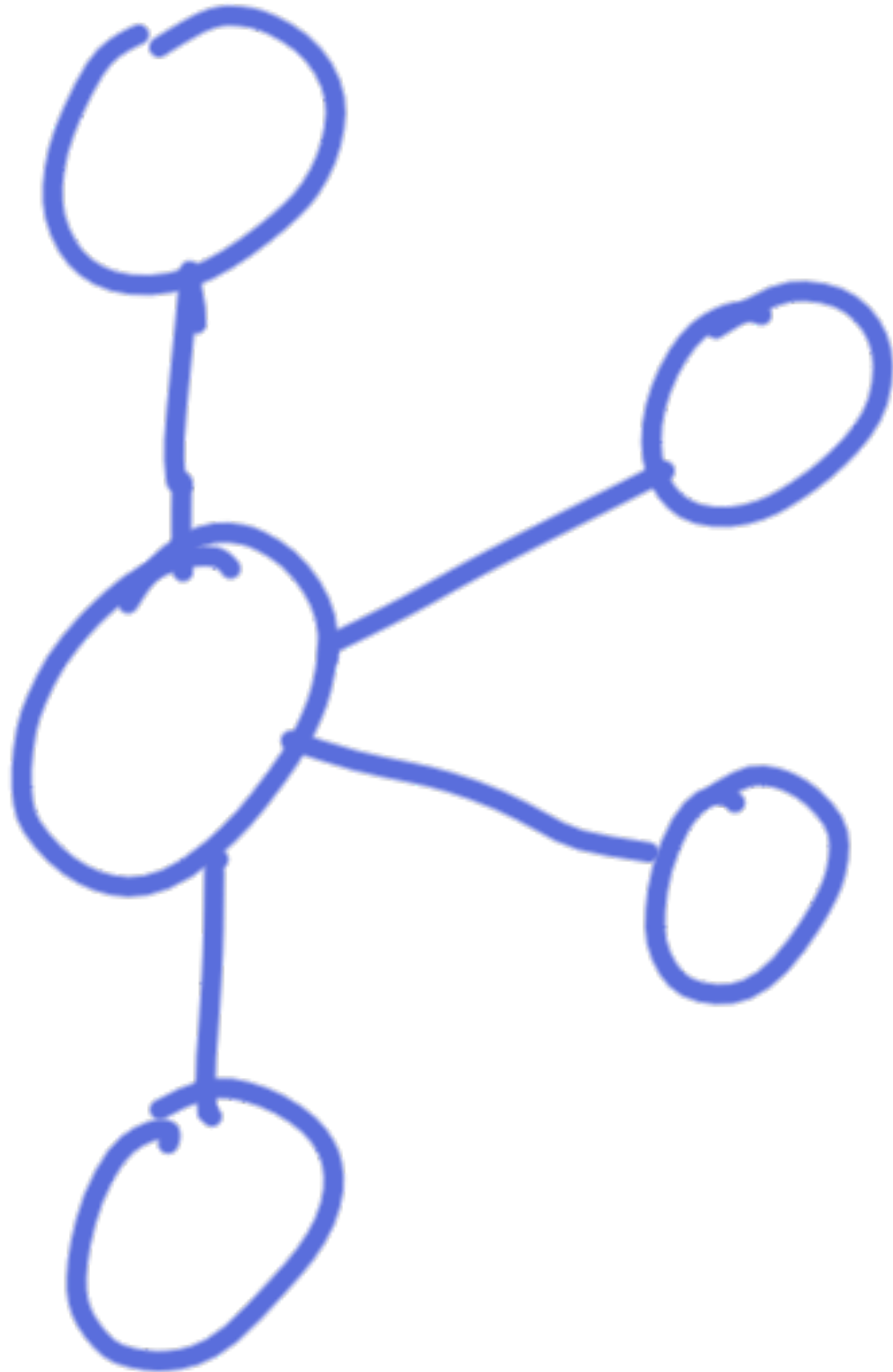


*Object store,
data warehouse,
RDBMS*

```
CREATE SINK CONNECTOR dw WITH (  
  'connector.class' = 'S3Connector',  
  'topics' = 'widgets'  
...);
```

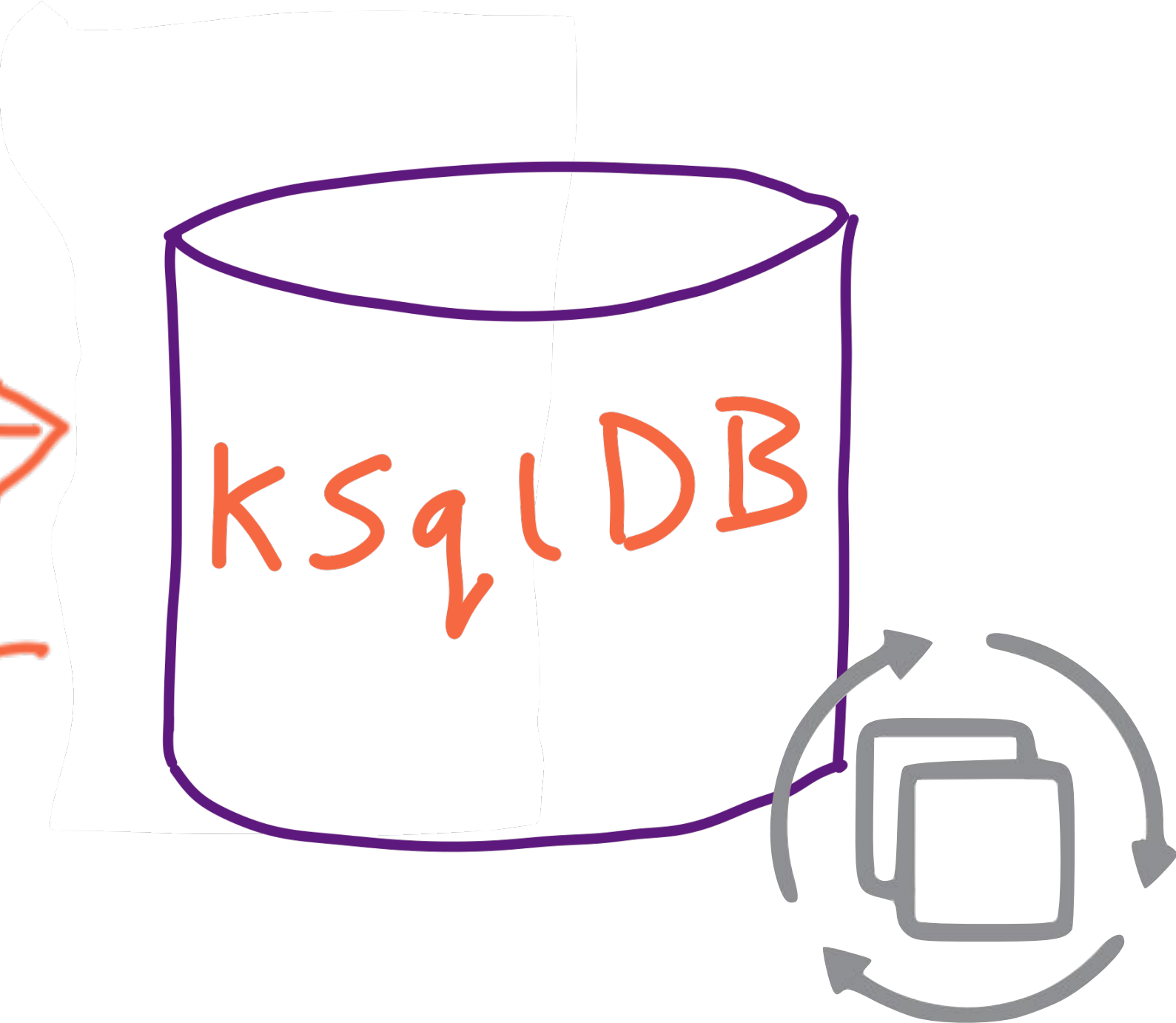
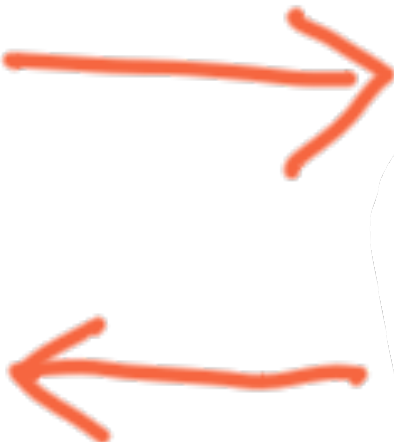
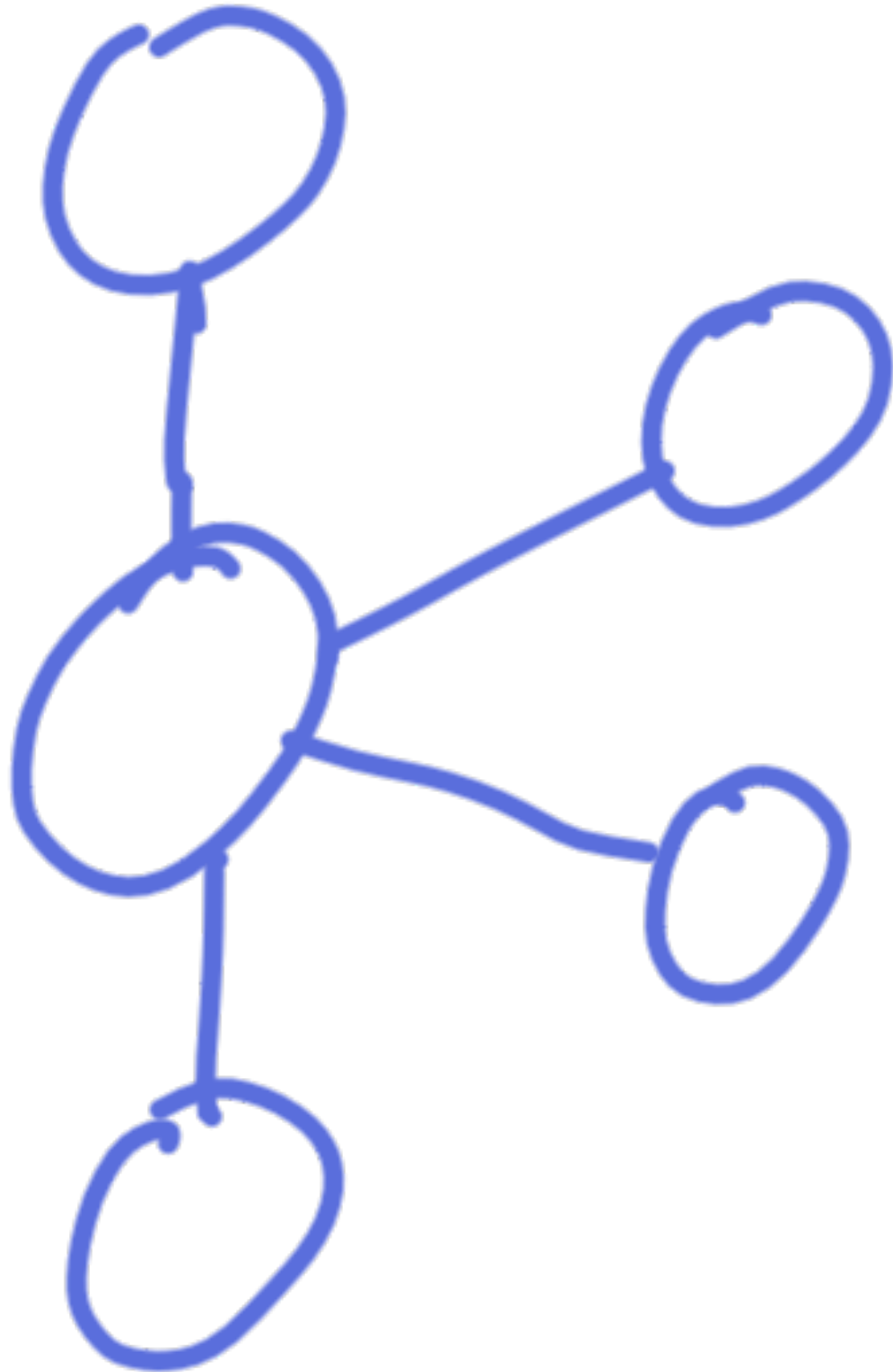
Stream Processing with ksqlDB

Source stream



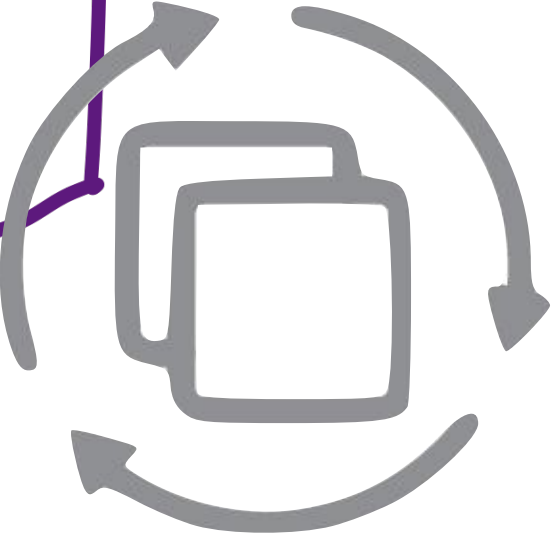
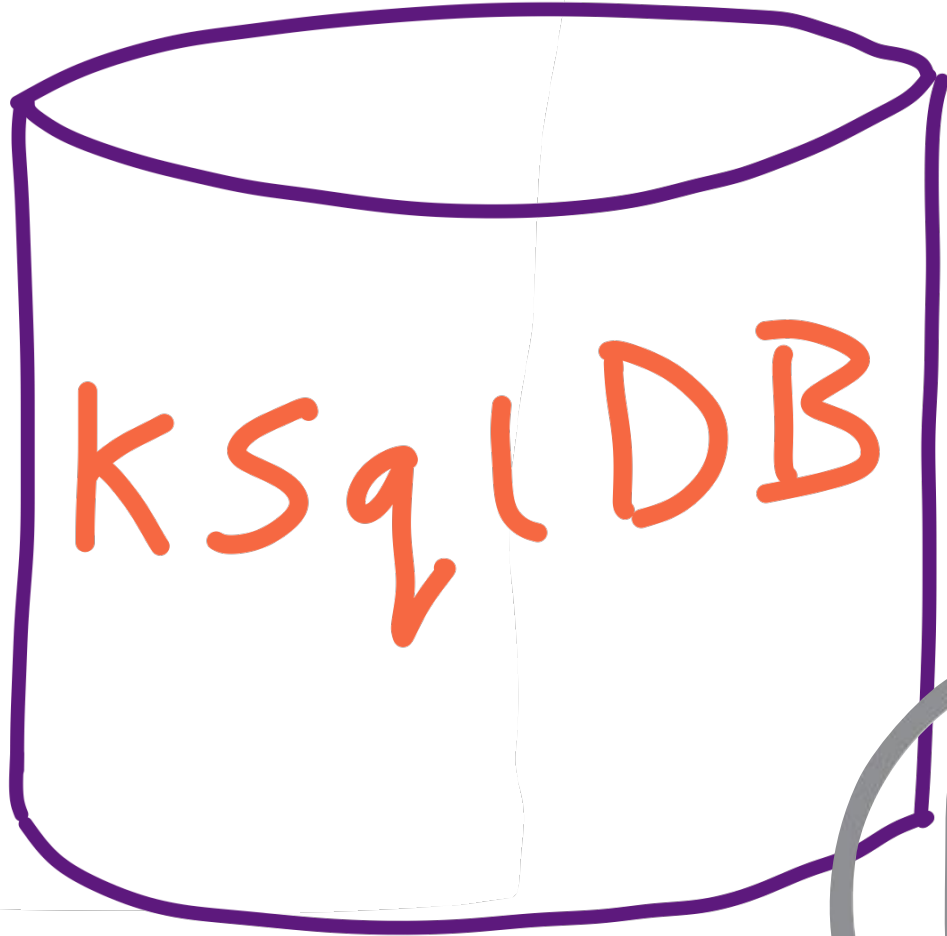
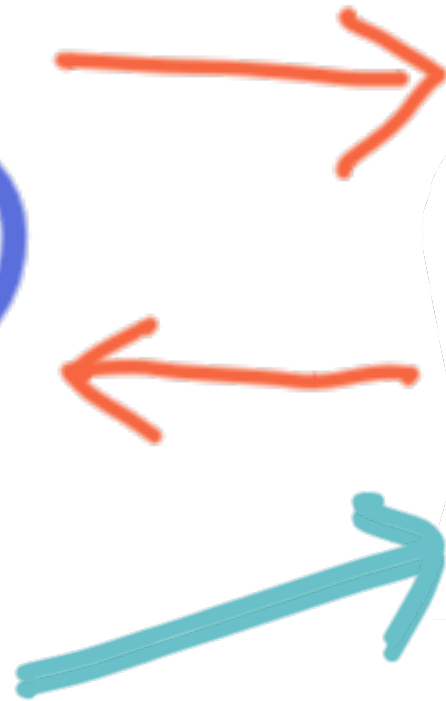
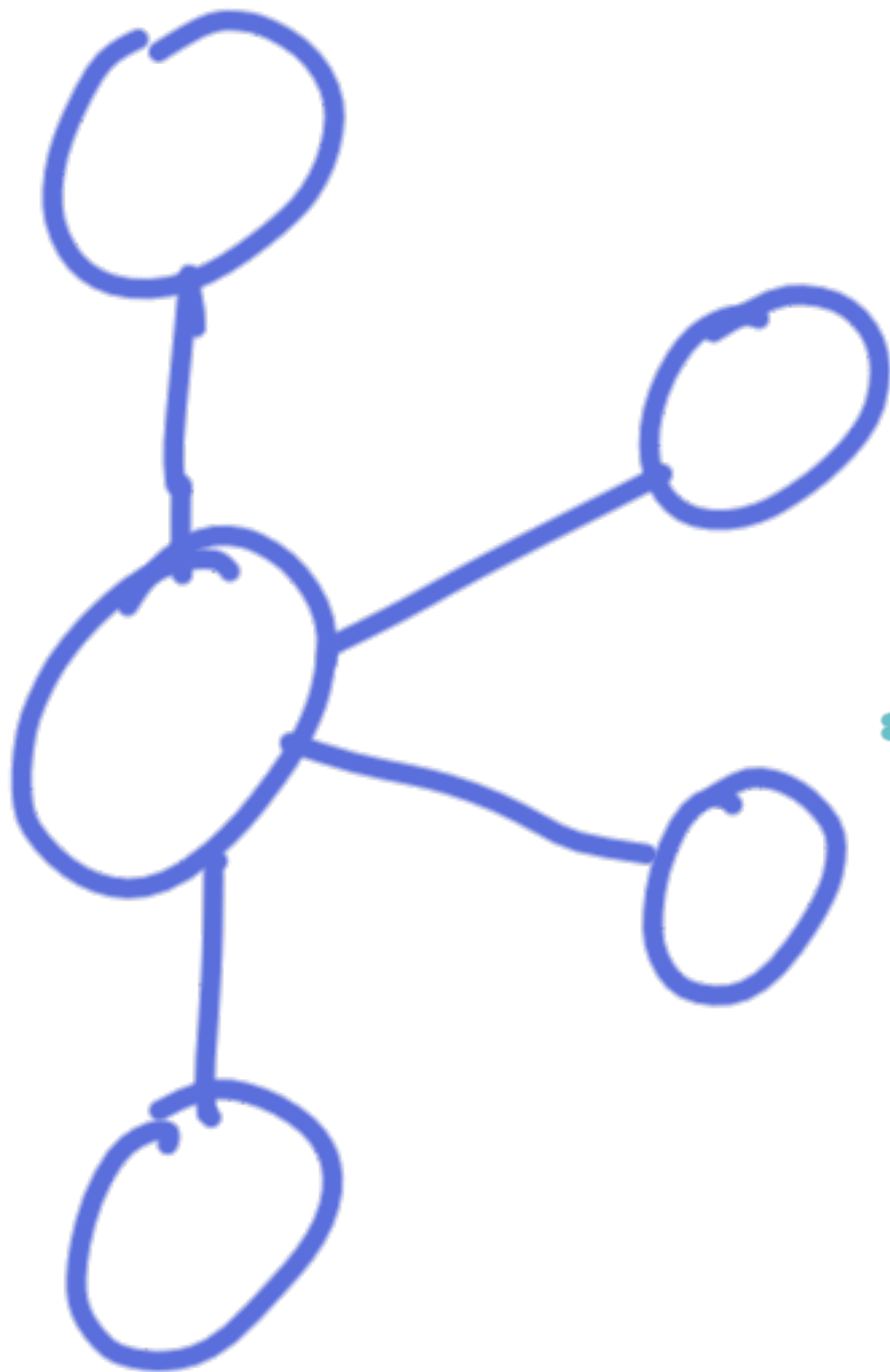
Stream Processing with ksqlDB

Source stream



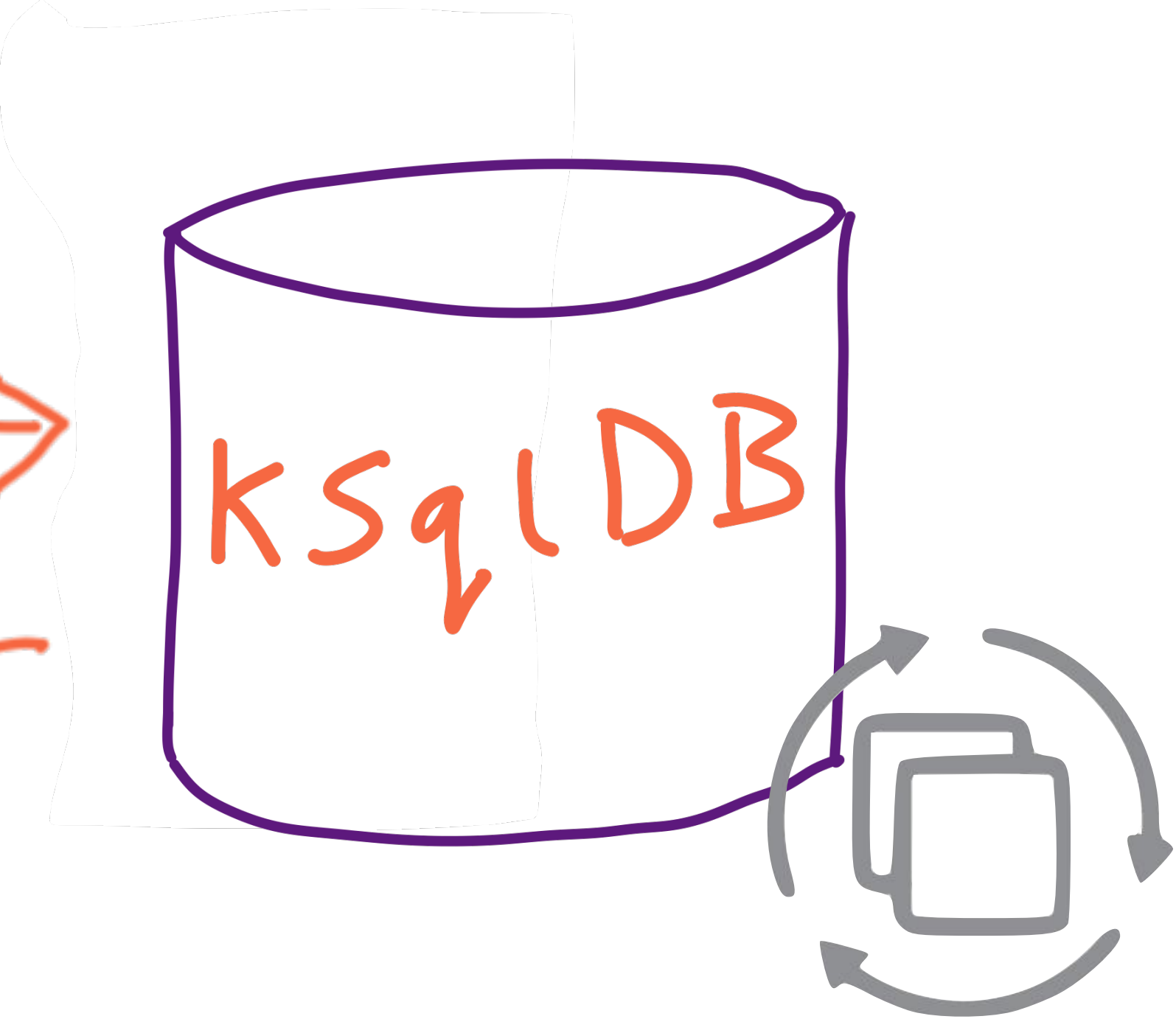
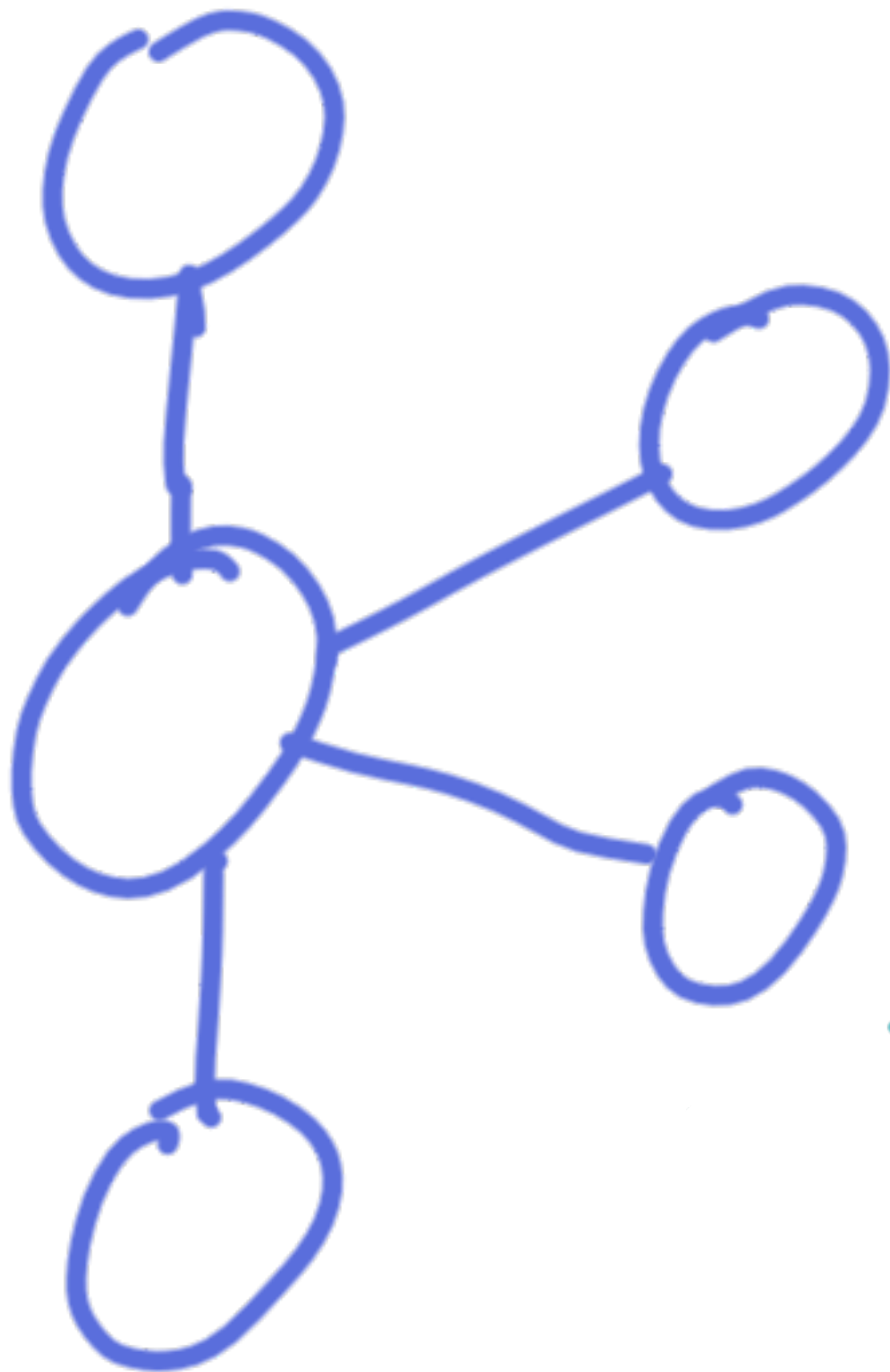
Stream Processing with ksqlDB

Source stream

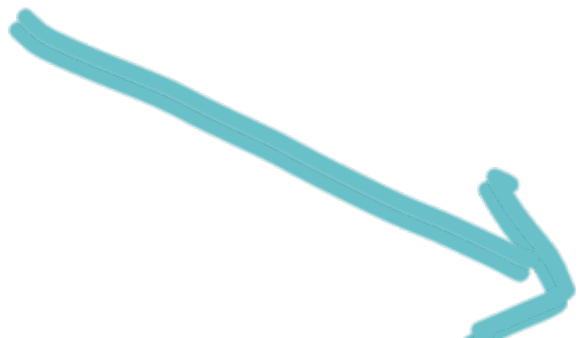


Stream Processing with ksqlDB

Source stream

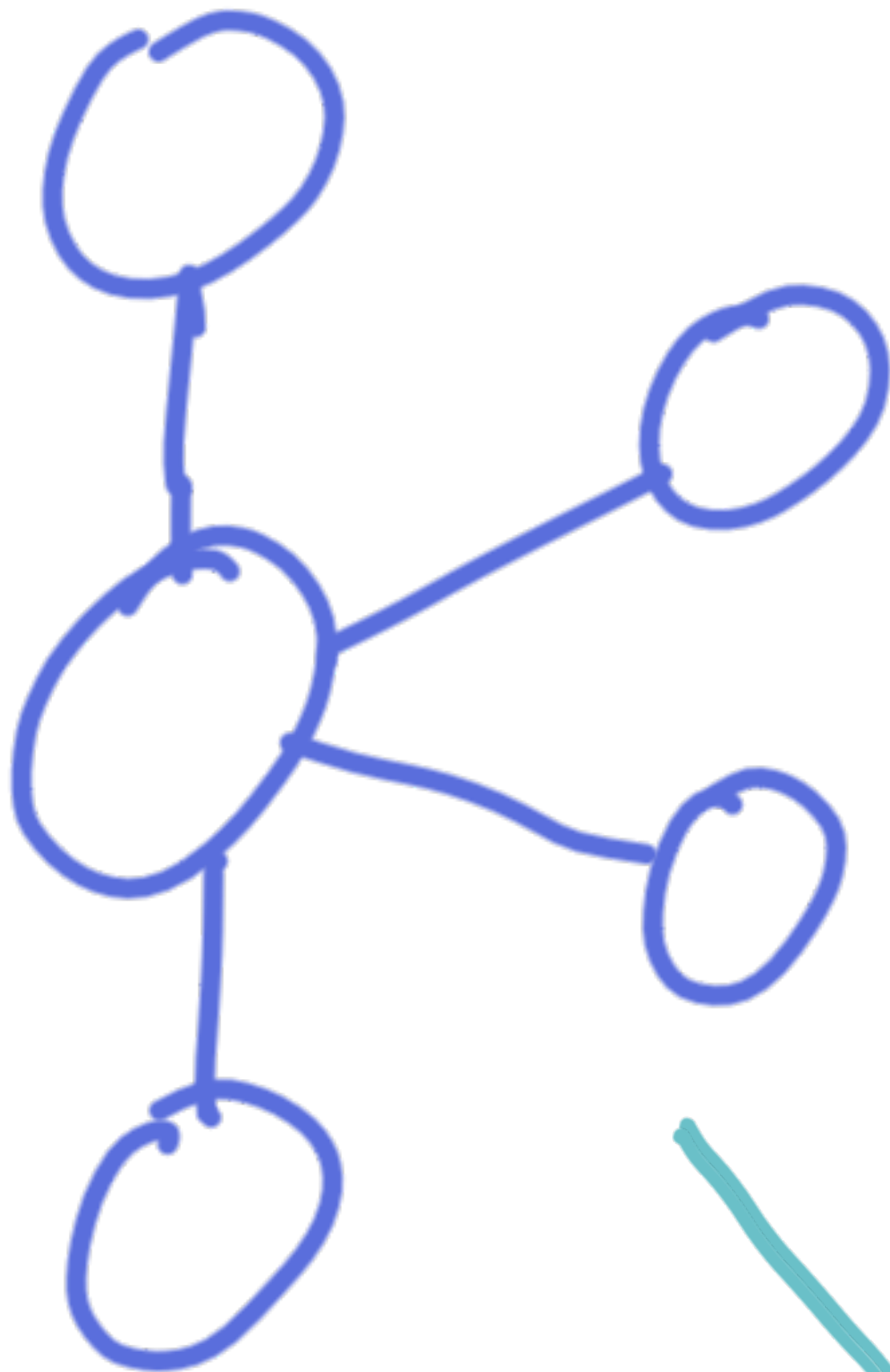


Analytics

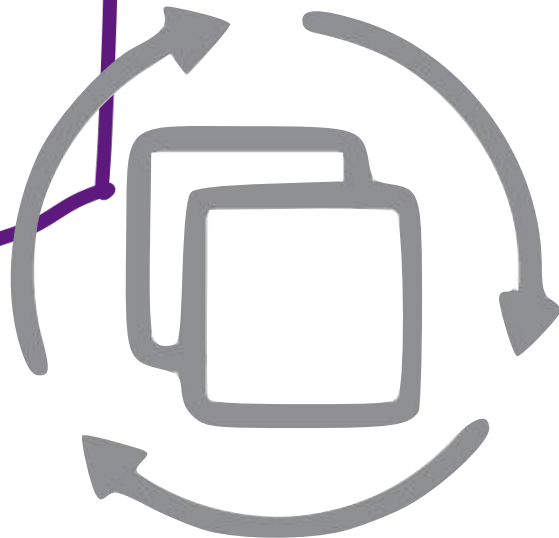


Stream Processing with ksqlDB

Source stream



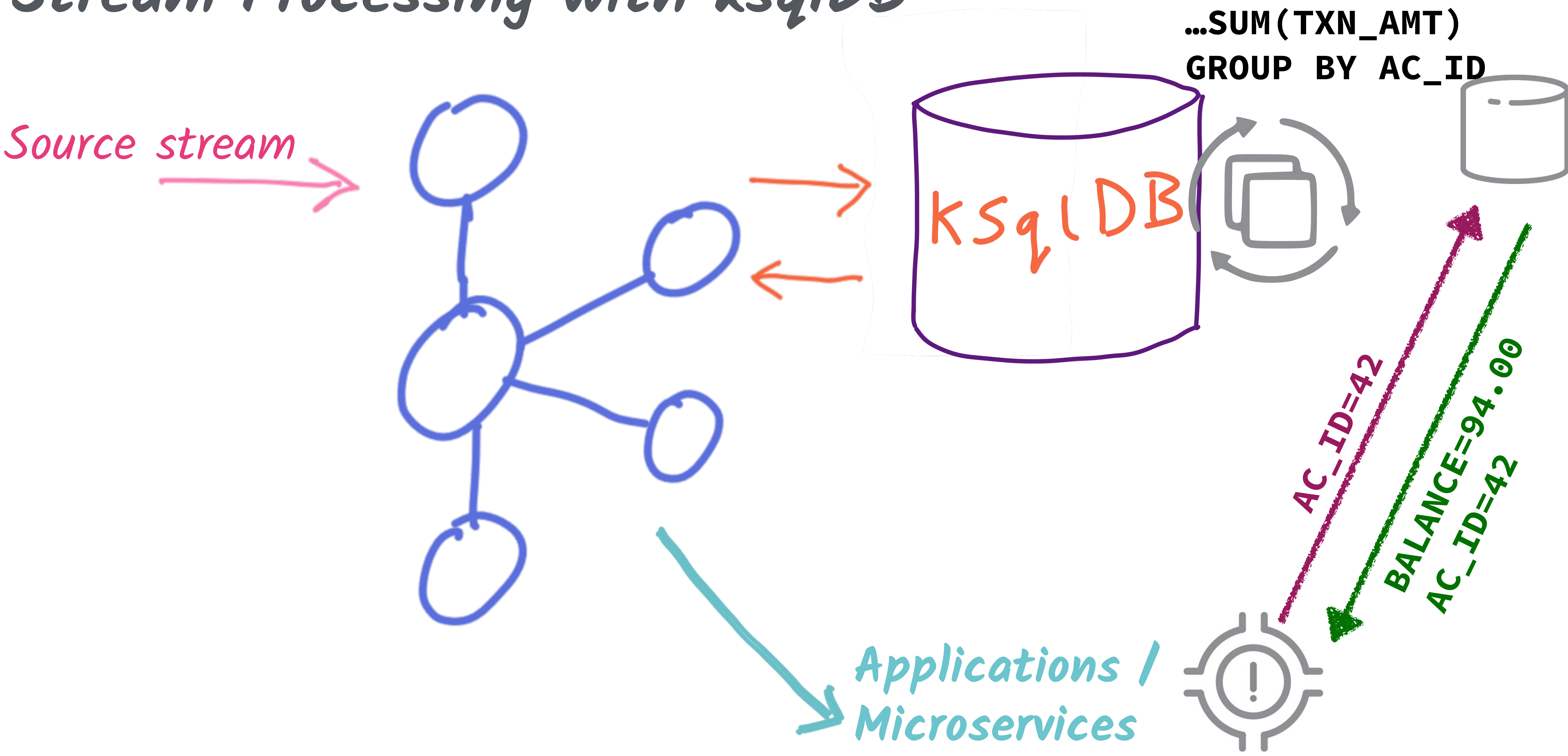
ksqlDB



Applications /
Microservices



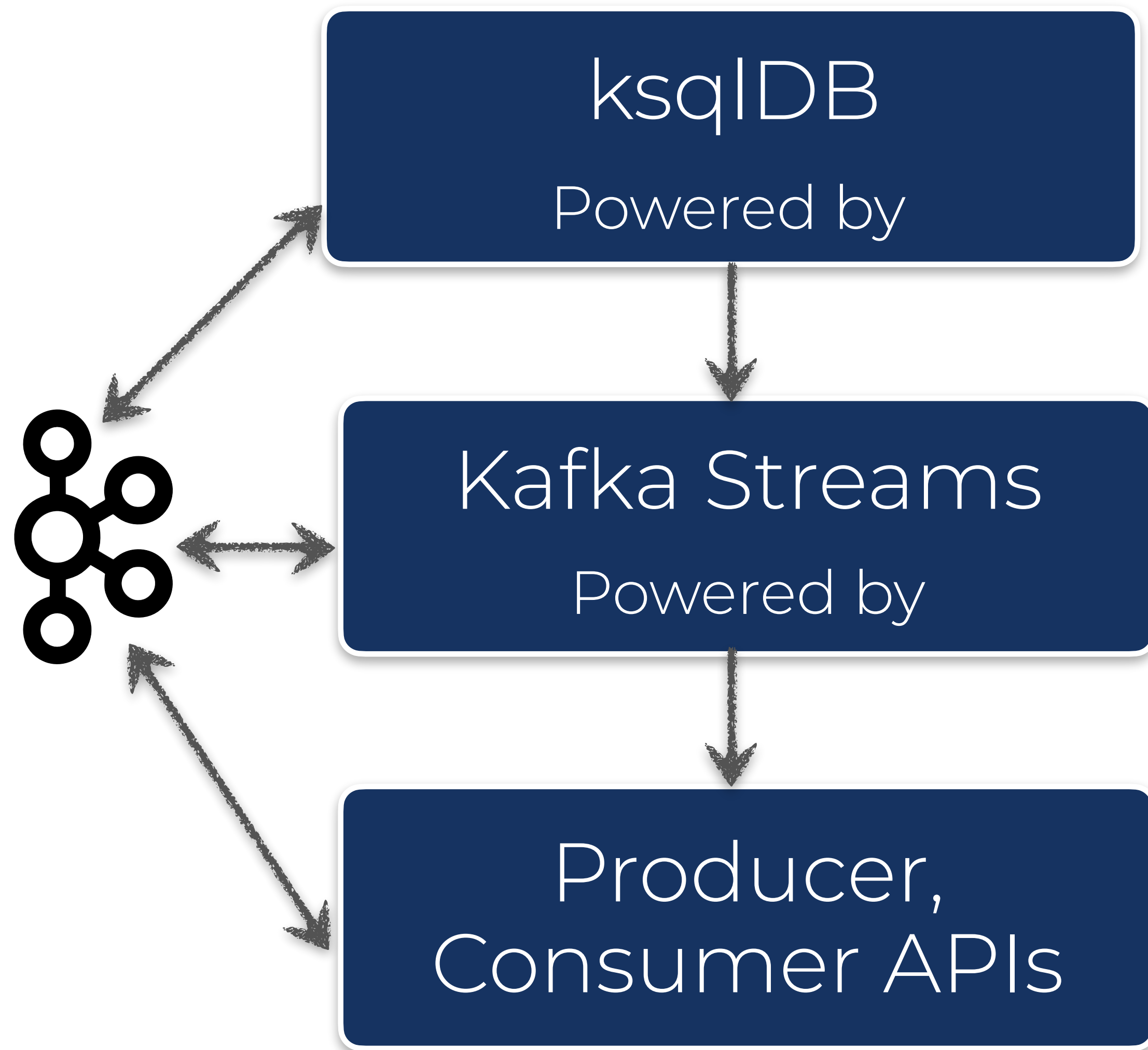
Stream Processing with ksqlDB



ksqlDB *or* *Kafka Streams?*



Standing on the Shoulders of Streaming Giants



Ease of use

Flexibility

```
CREATE STREAM,  
CREATE TABLE, SELECT, JOIN,  
GROUP BY, SUM, ...
```

ksqlDB UDFs

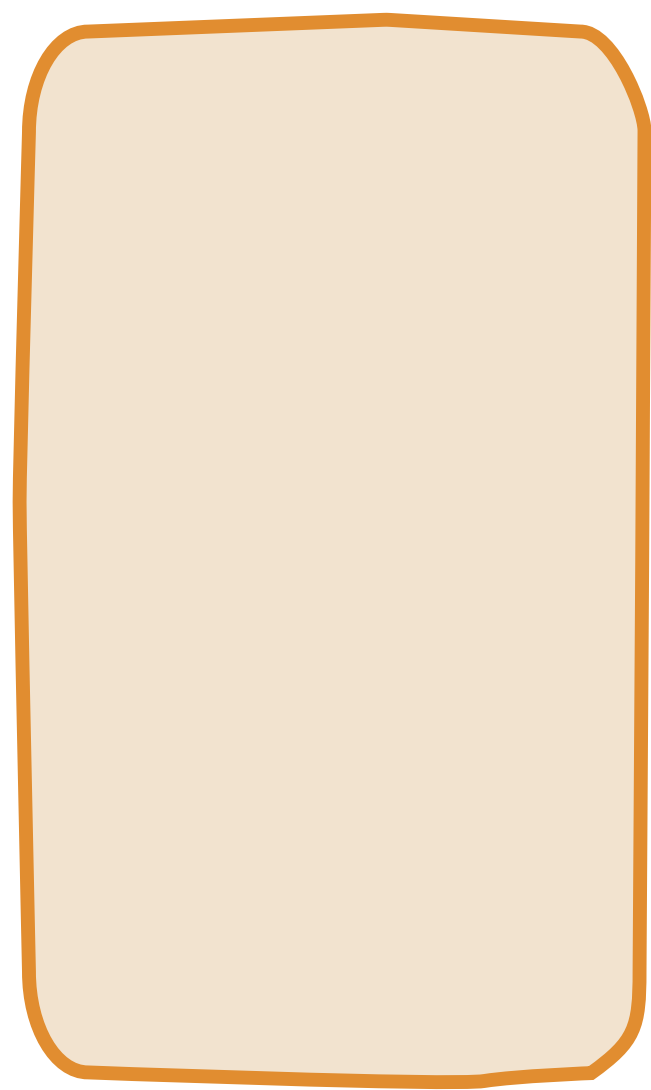
```
KStream<>, KTable<>, filter(), map(),  
flatMap(), join(), aggregate(),  
transform(), ...
```

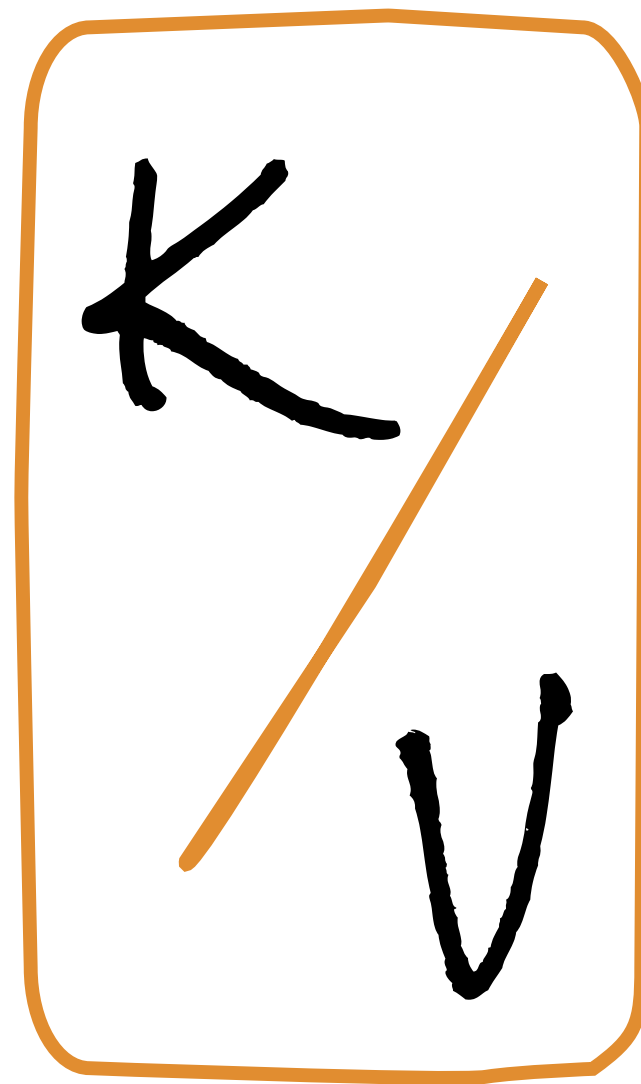
```
subscribe(), poll(), send(),  
flush(), beginTransaction(), ...
```

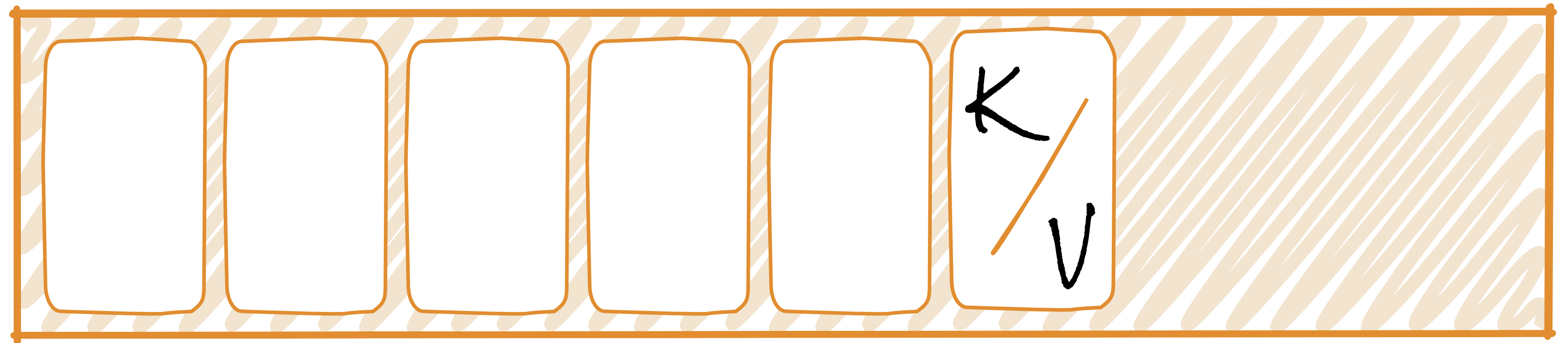



DEMO

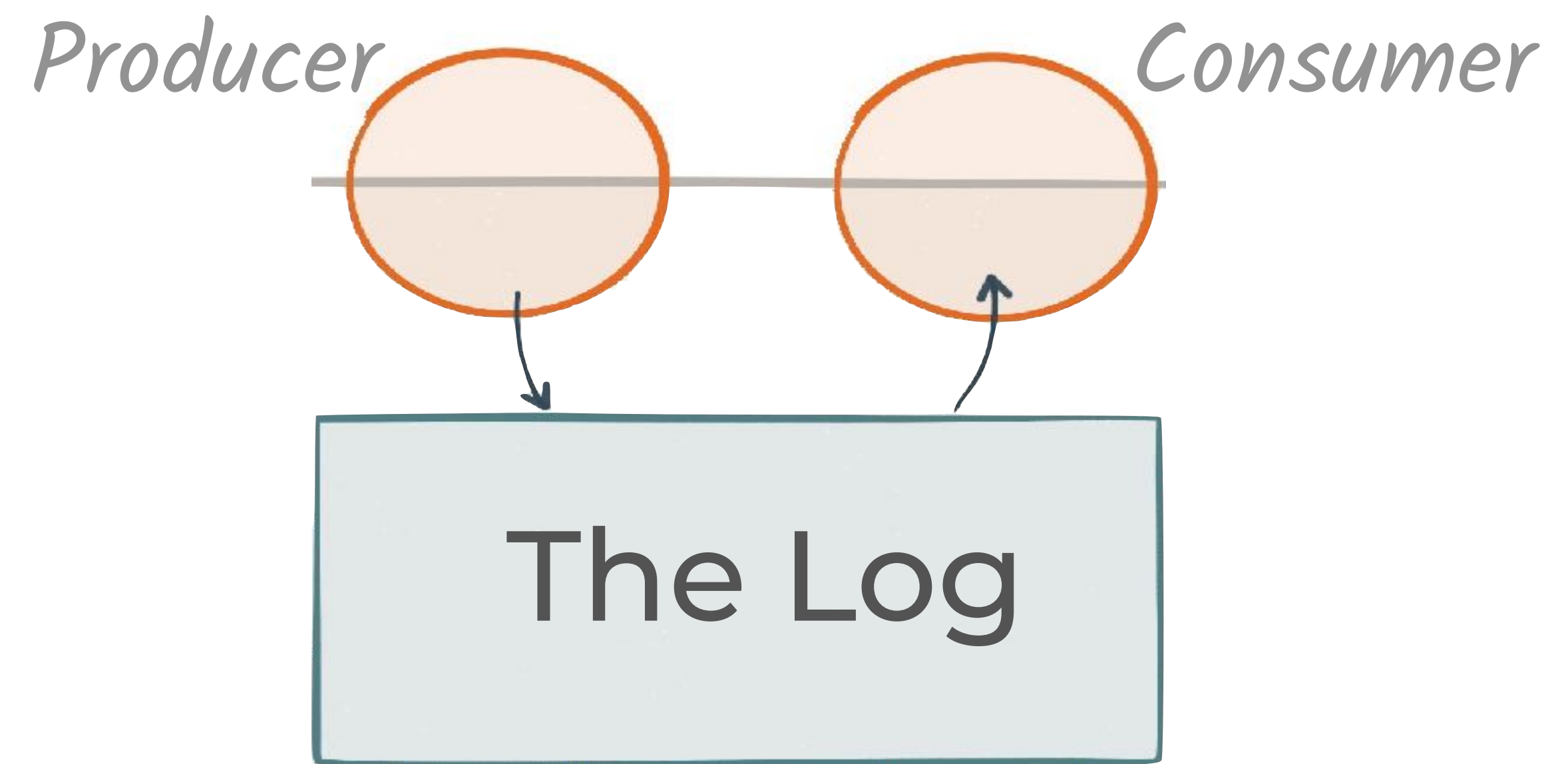
Summary

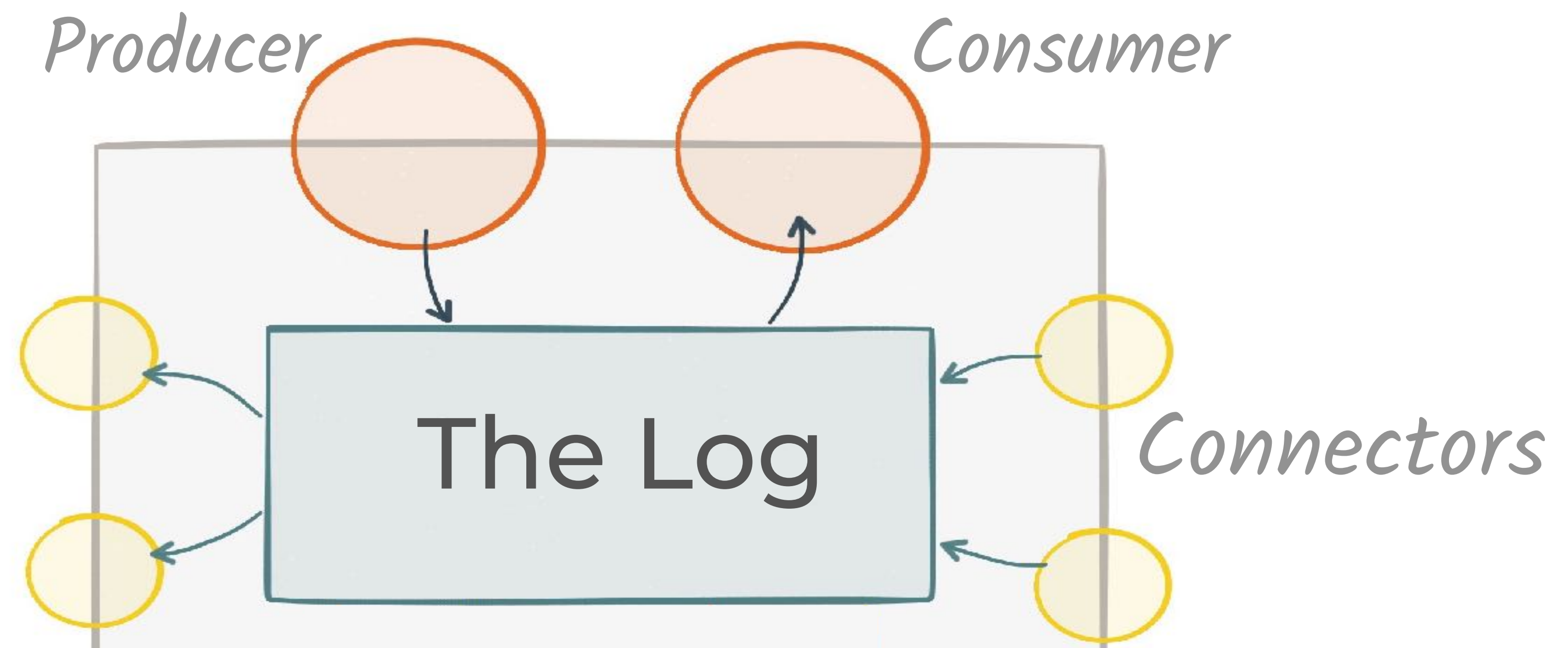


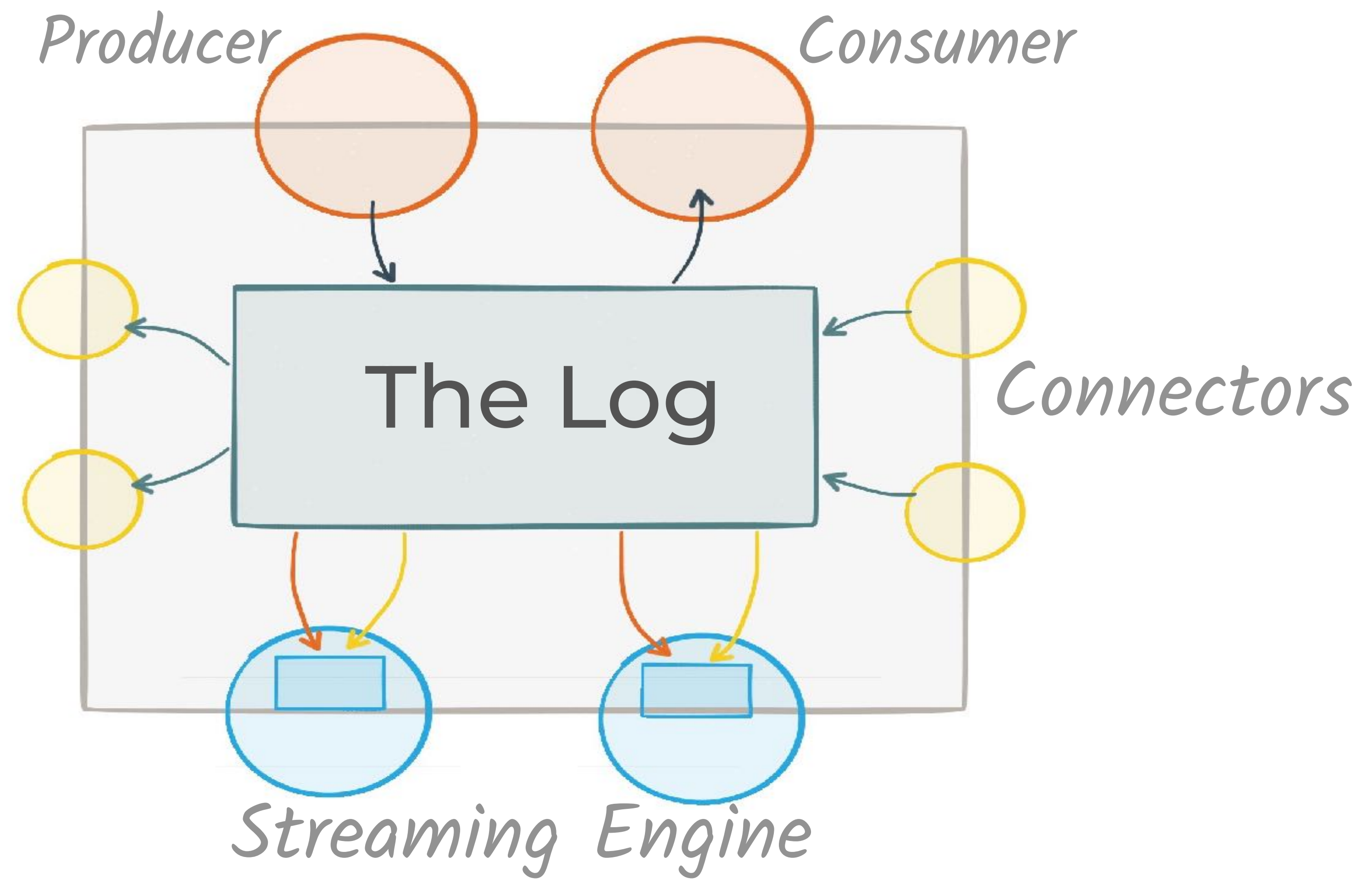




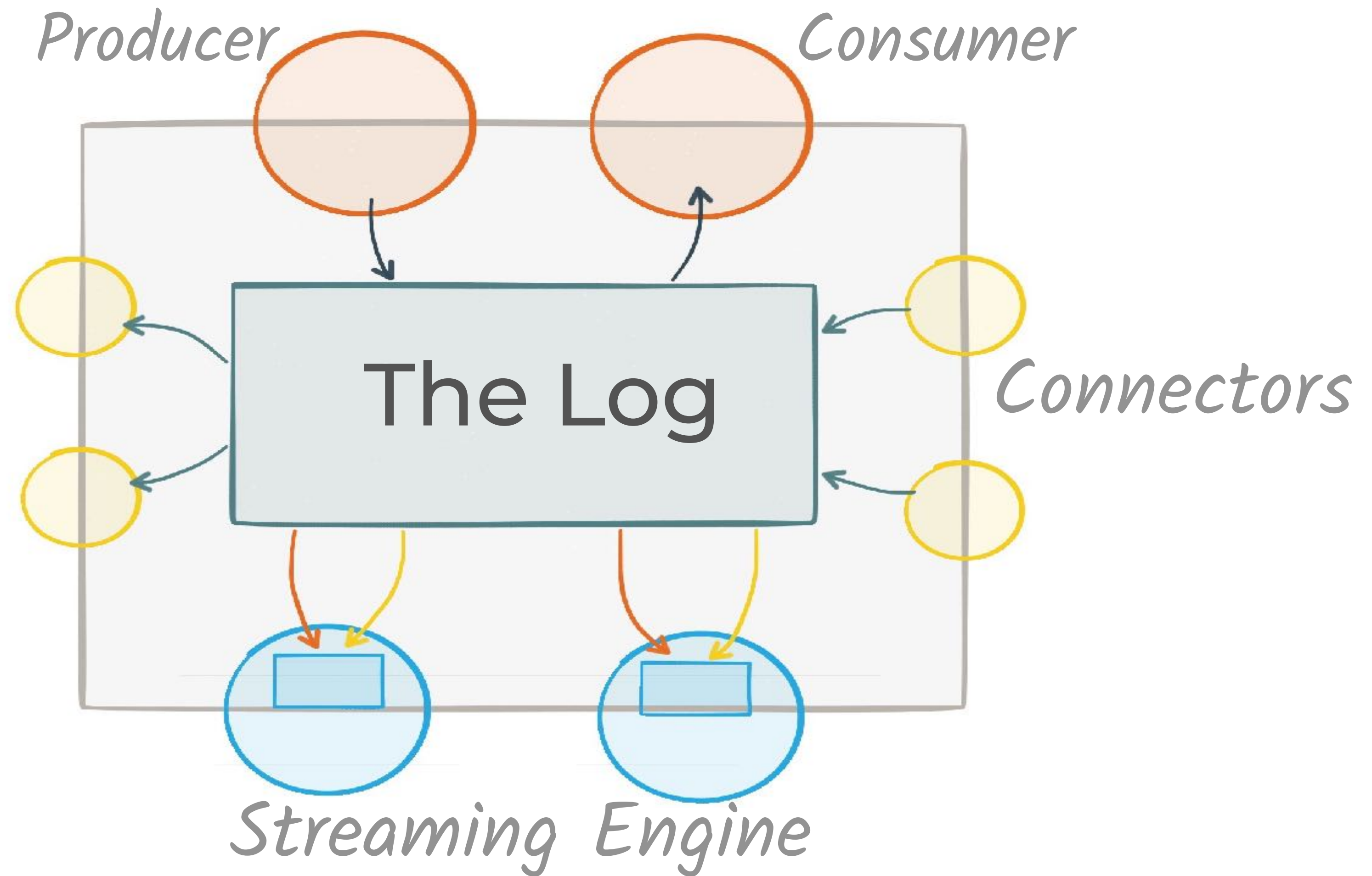
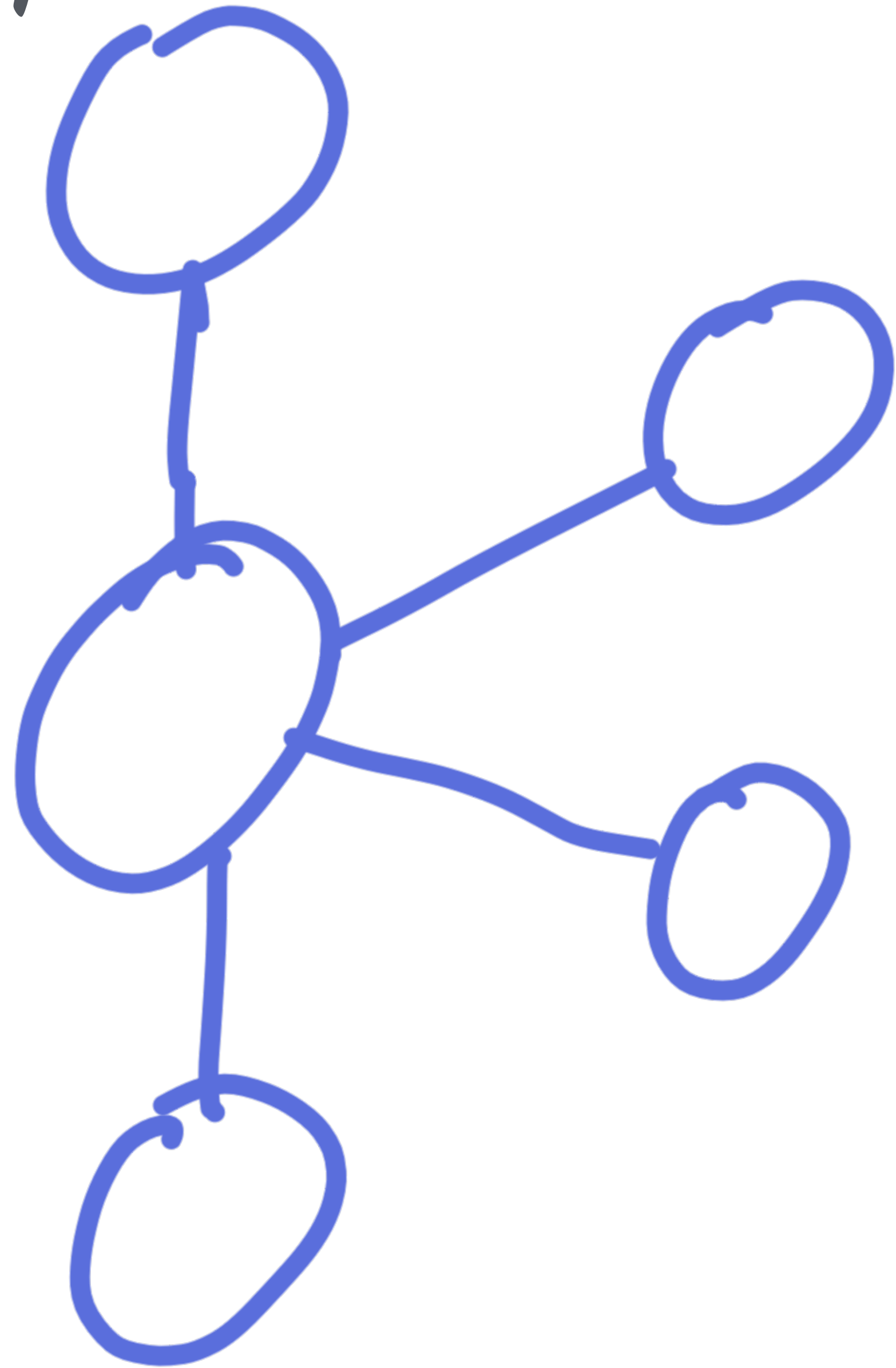
The Log



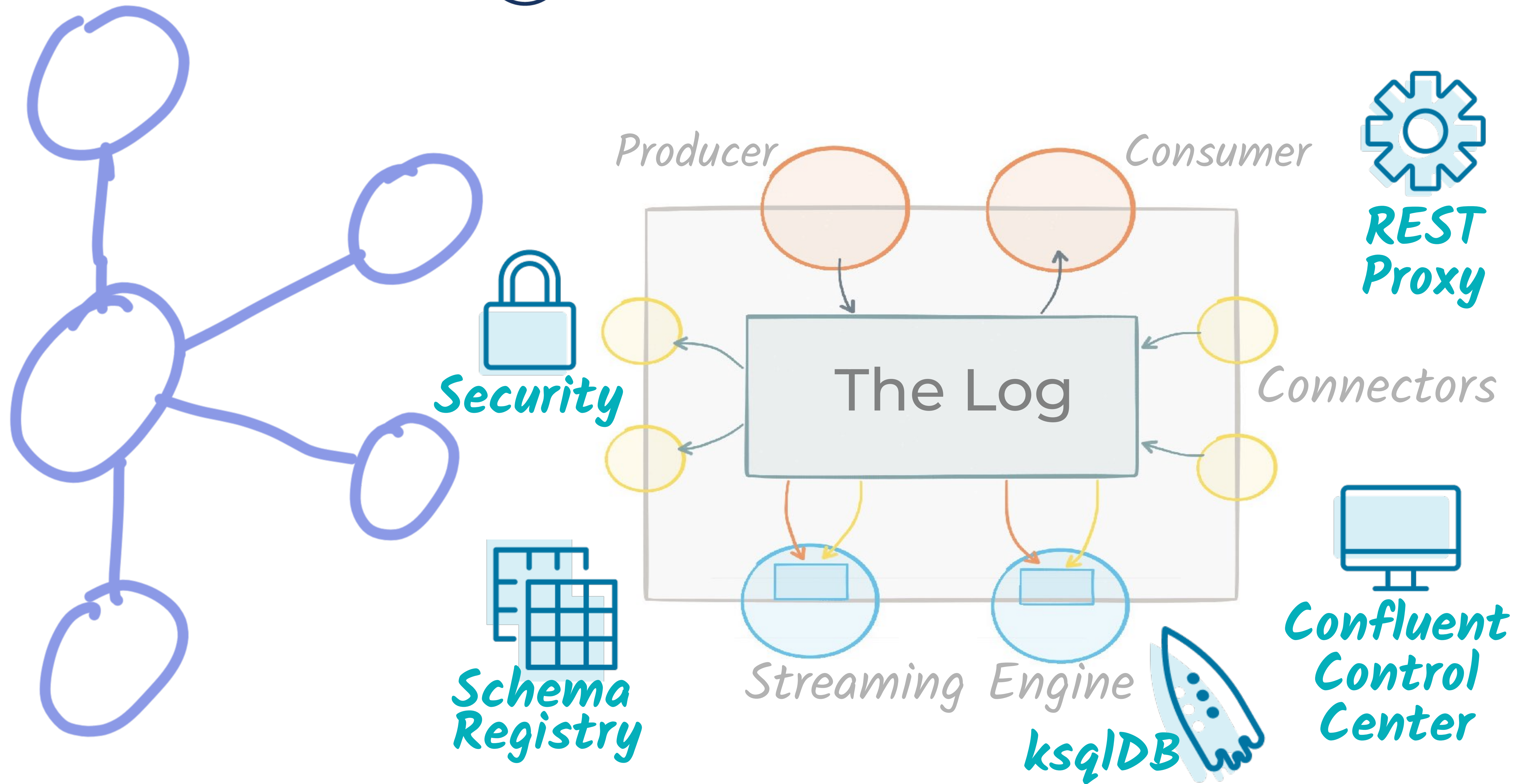




Apache Kafka



CONFLUENT



EVENTS

are

EVERYWHERE

EVENTS

are

very
^

POWERFUL

Learn Kafka.

Start building with
Apache Kafka at
Confluent Developer.



developer.confluent.io

@rmoff

#EOF

rmoff.dev/talks