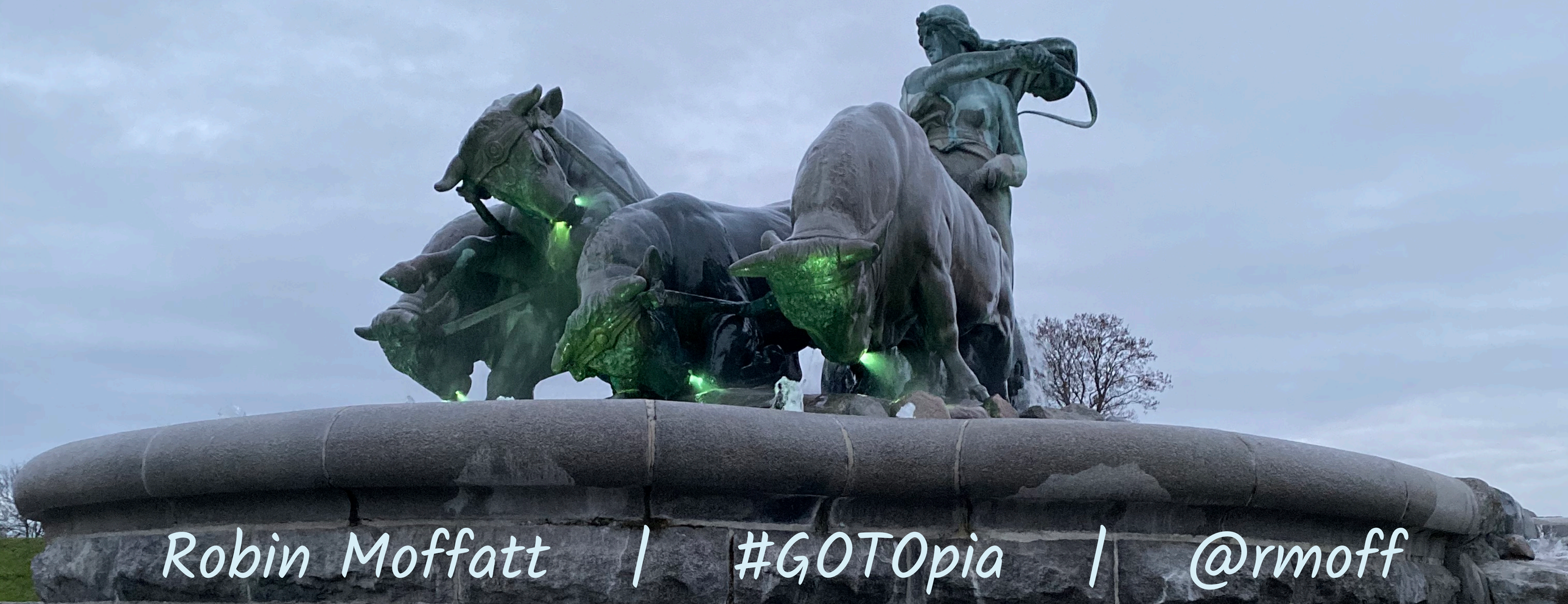
Kafka as a Platform:
the Ecosystem from the Ground Up



Robin Moffatt | #GOTOpia | @rmoff

EVENTS

@rmoff | #GOTOpia | @confluentinc

EVENTS

EVENTS

- Something happened
- What happened

Human generated events

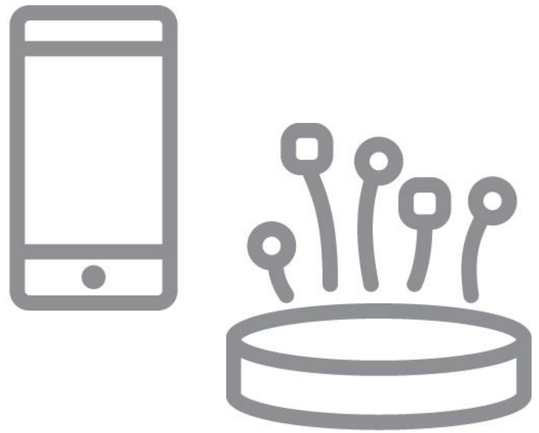
A Sale

A Stock movement

Machine generated events

Networking

IoT



Applications

EVENTS

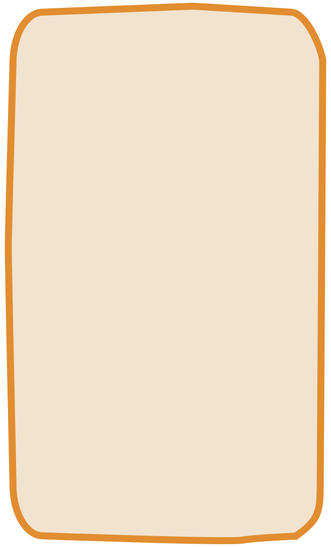
are

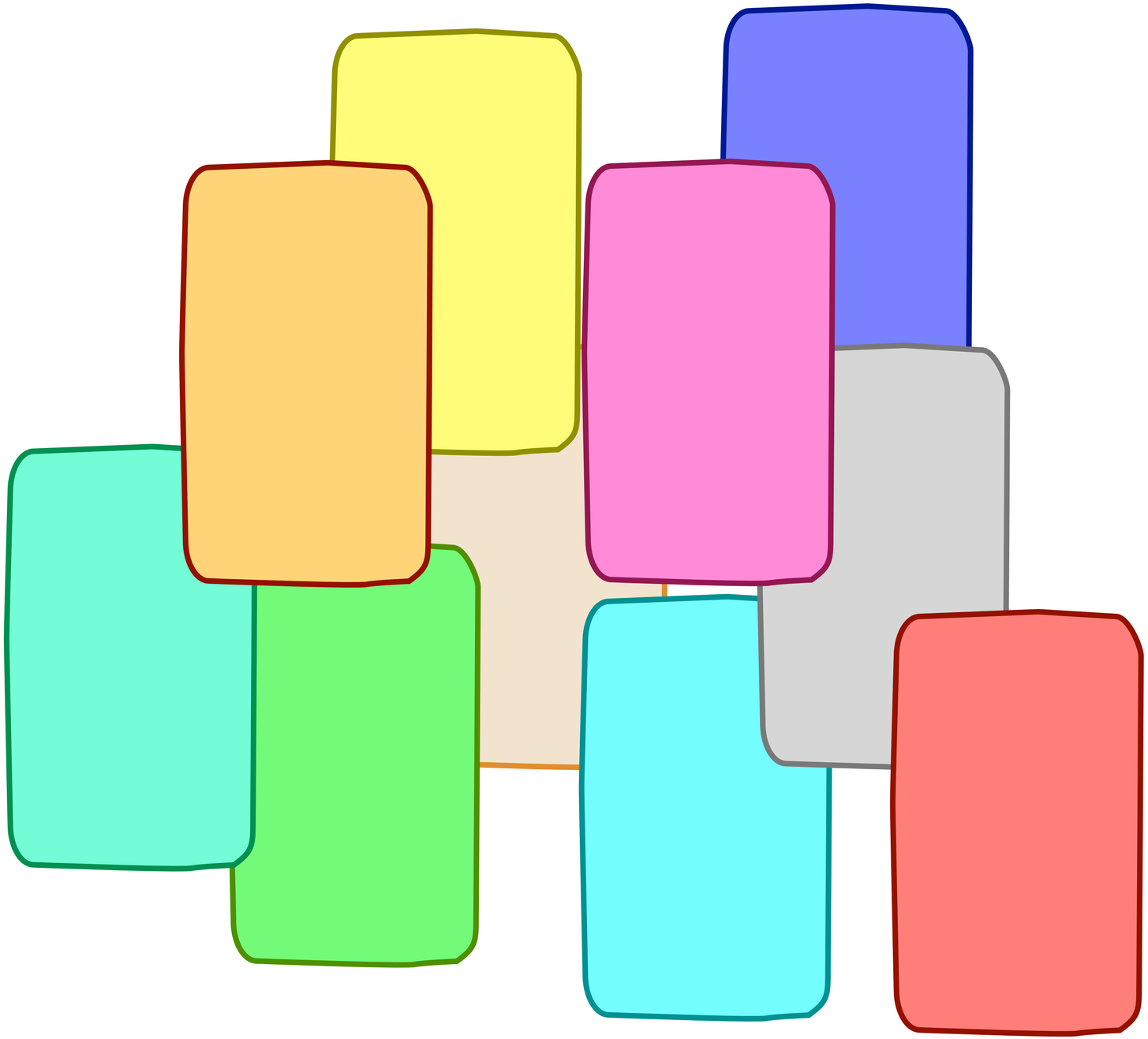
EVERYWHERE

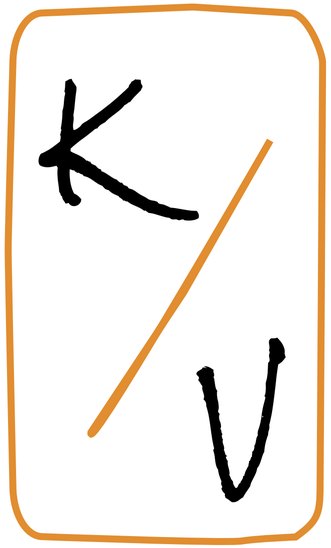
EVENTS

are

very POWERFUL






K
V

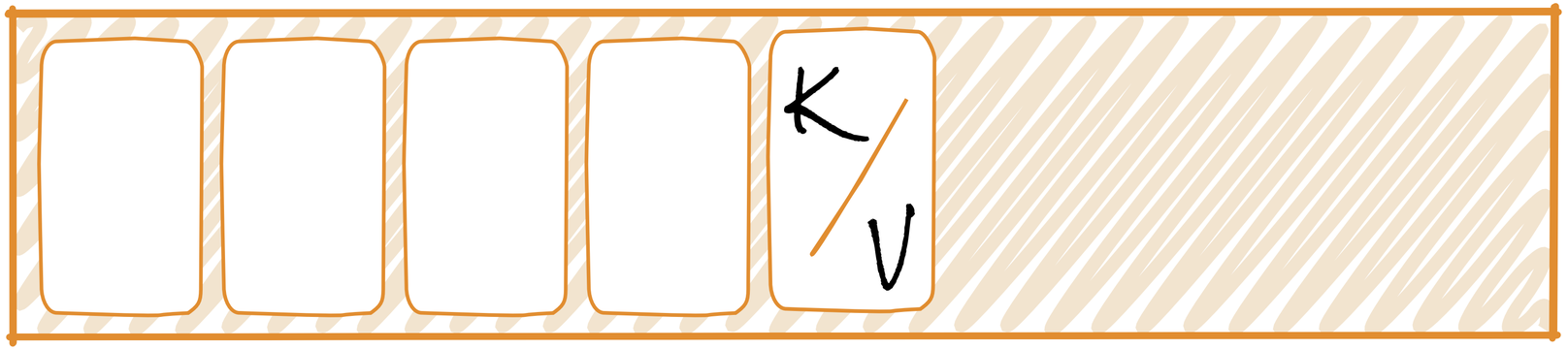
LOG

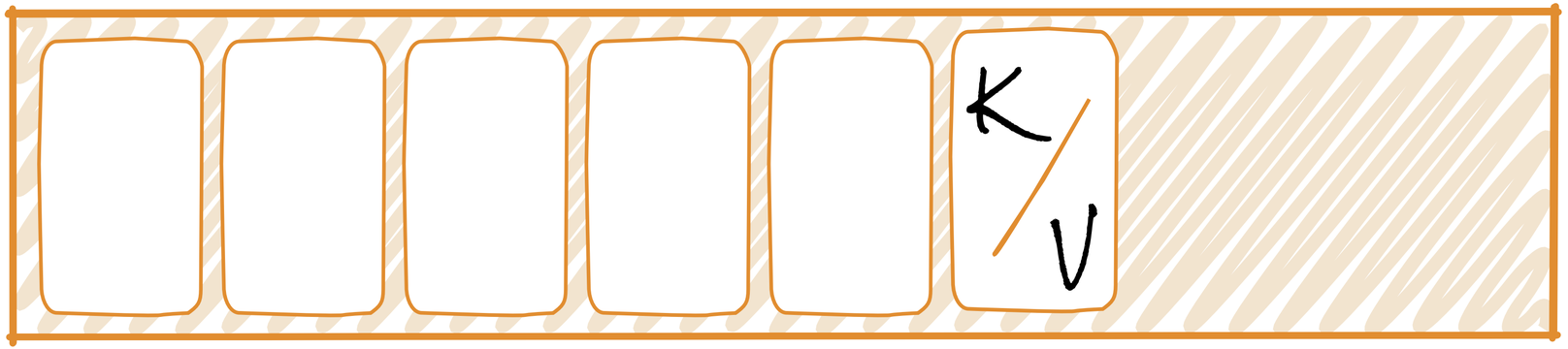

K
V

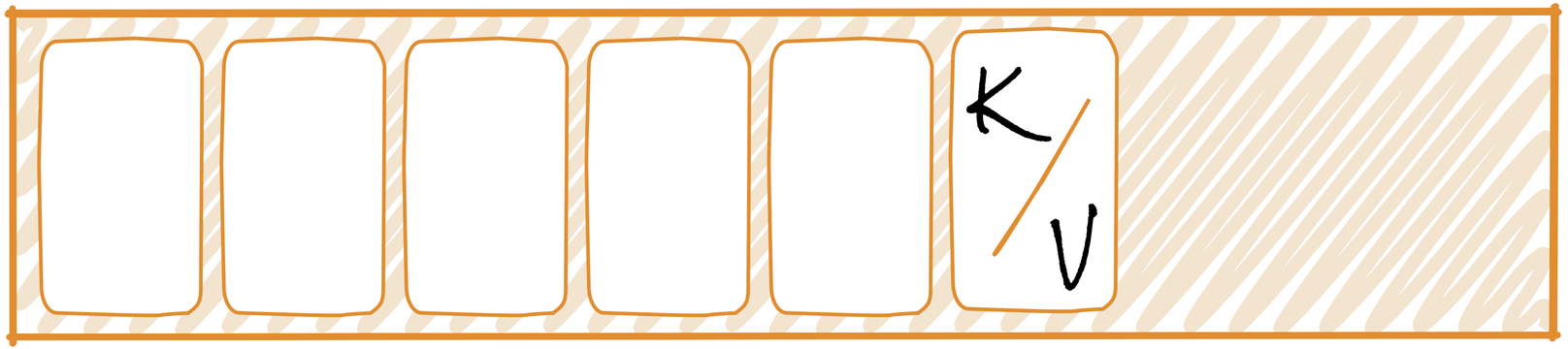

K
V


K
V

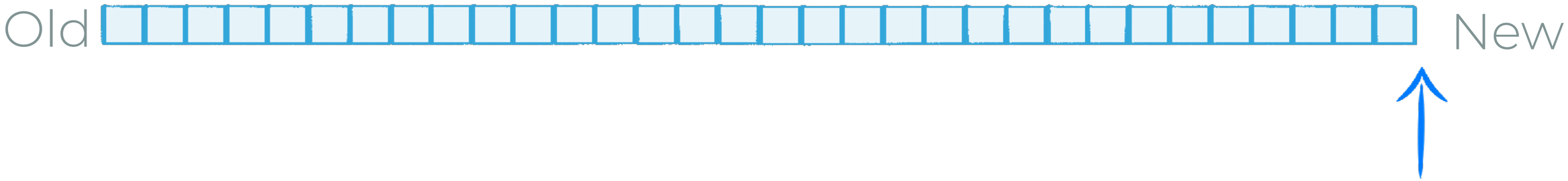

K
V


K
V


K
V


K
V

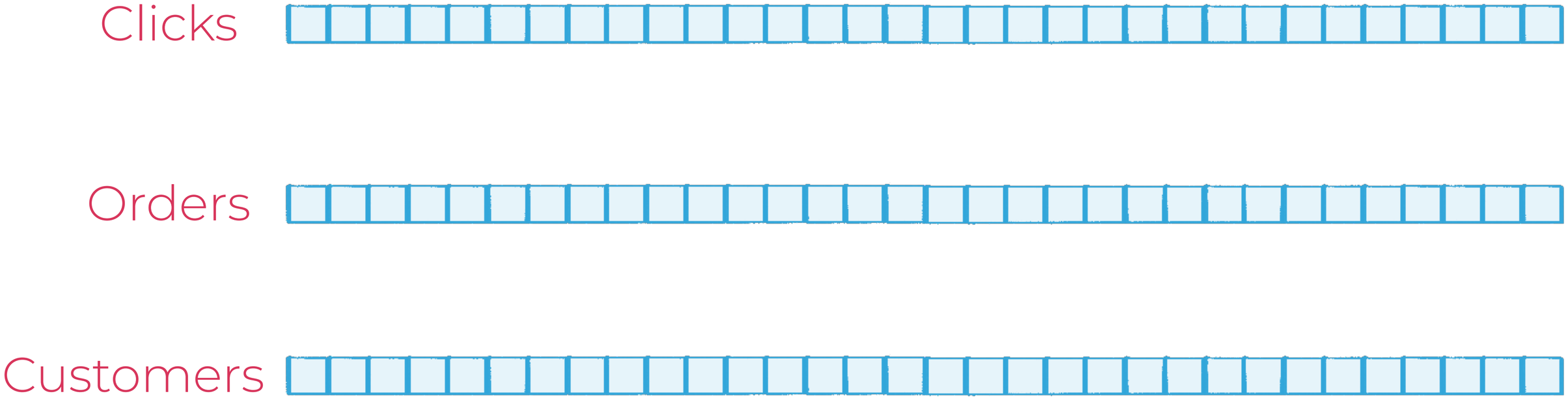
Immutable Event Log



Events are added at the end of the log

TOPICS

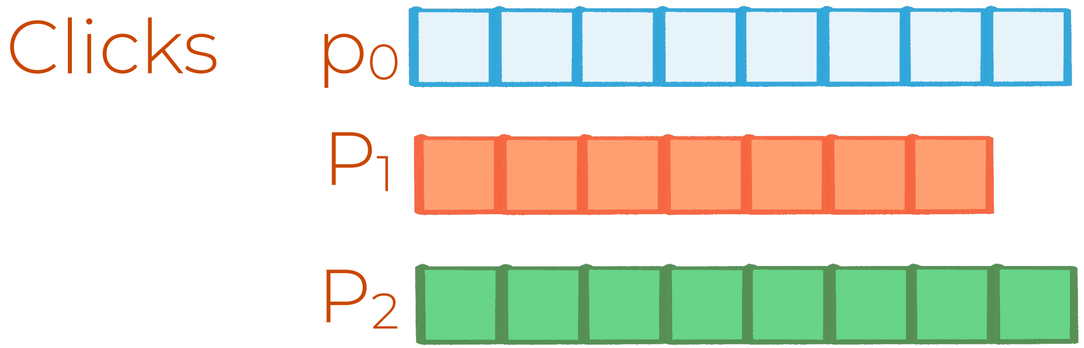
Topics



Topics are similar in concept to tables in a database

PARTITIONS

Partitions



Messages are guaranteed to be strictly ordered within a partition

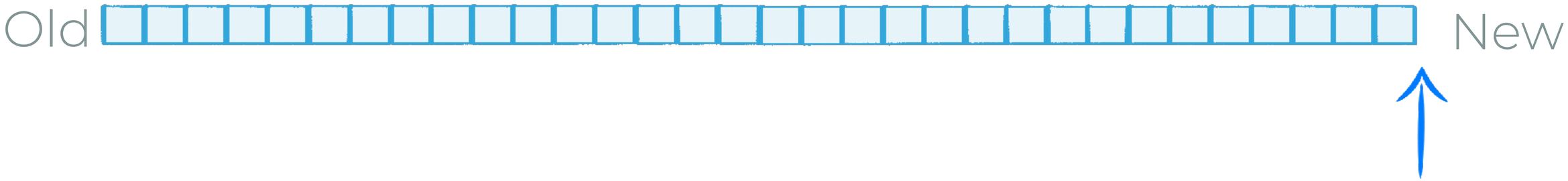
PUB / SUB

@rmoff | #GOTOpia | @confluentinc

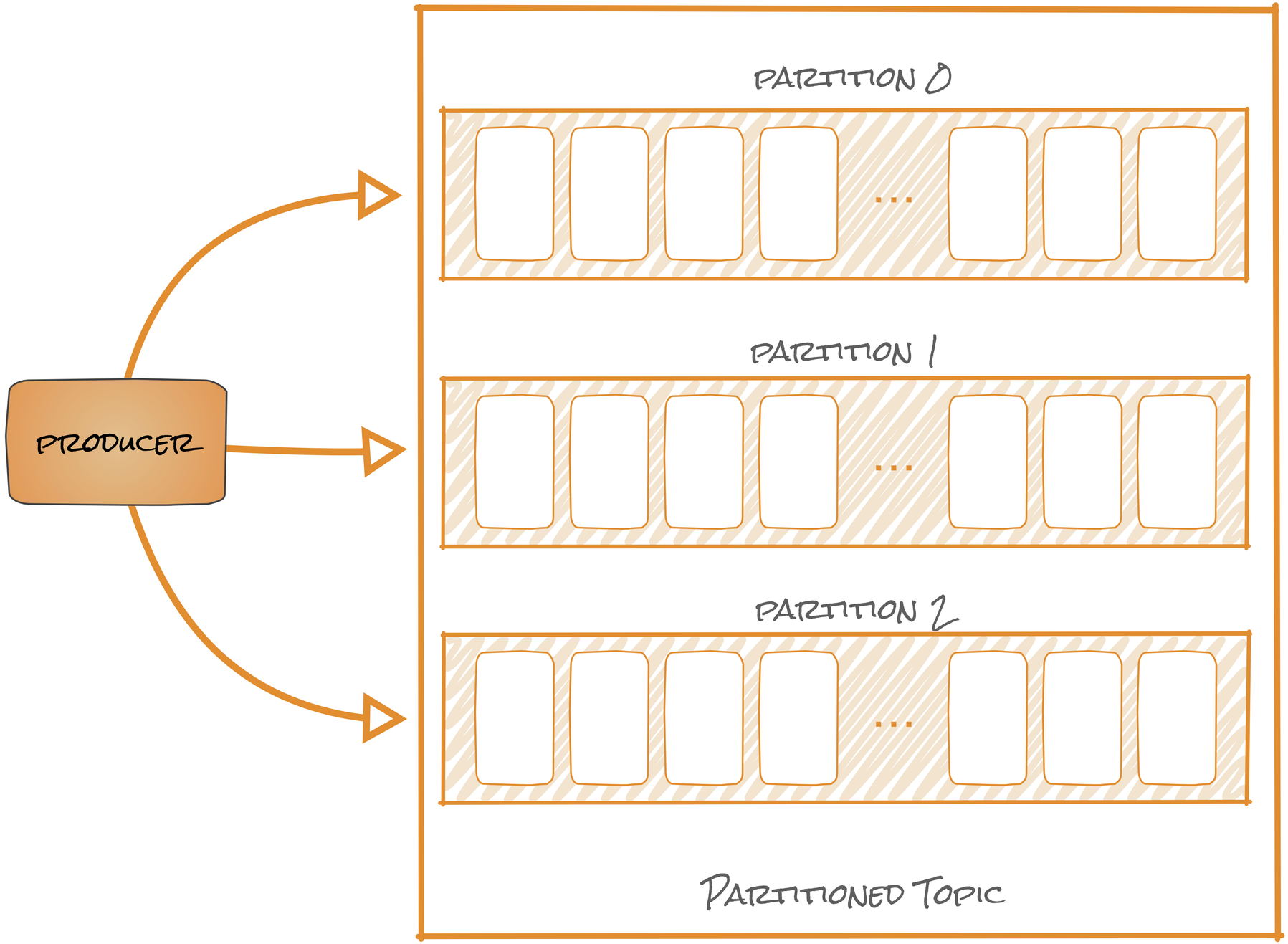
PUB / SUB

@rmoff | #GOTOpia | @confluentinc

Producing data

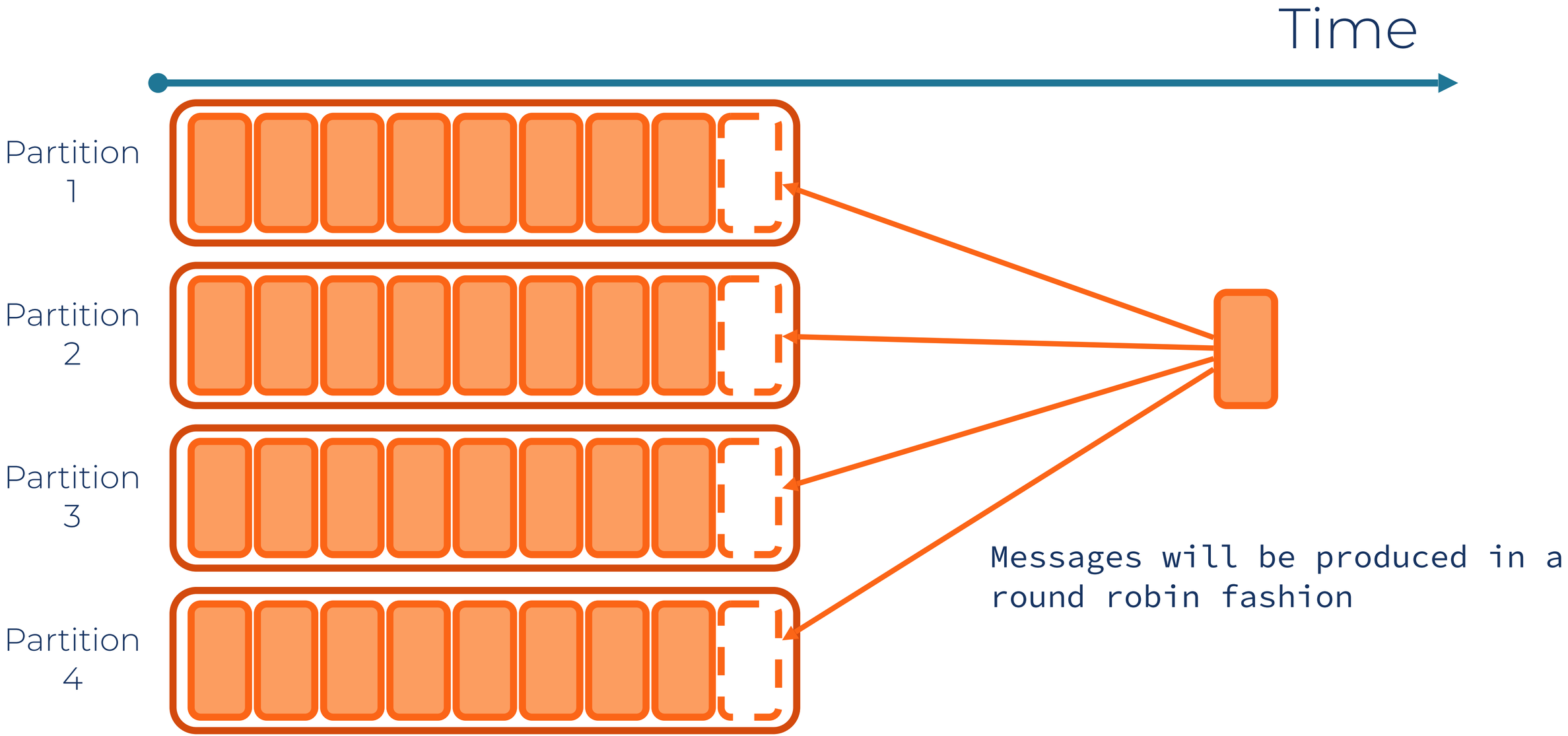


Messages are added at the end of the log

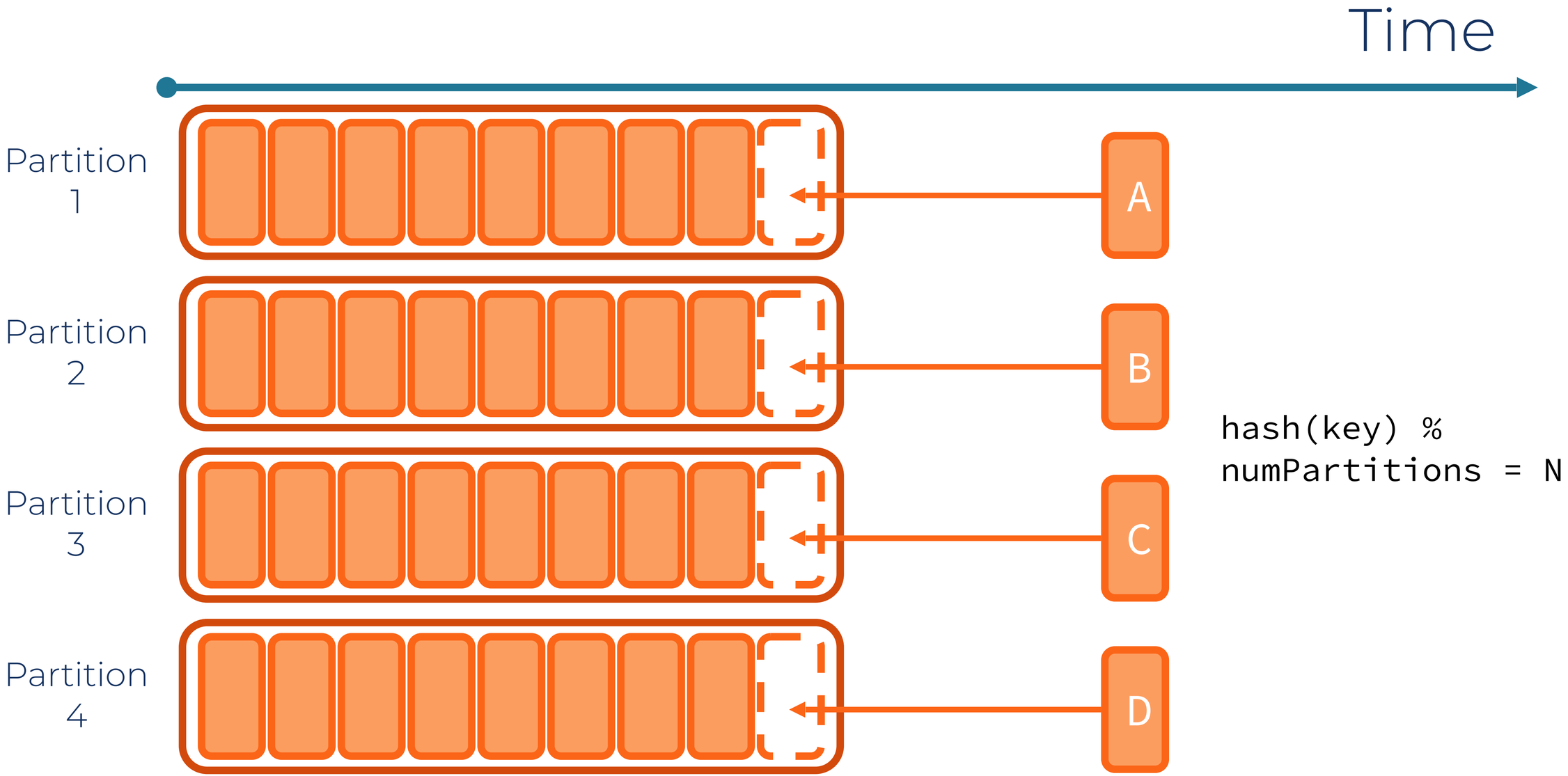

PRODUCER
PARTITION 0
PARTITION 1
PARTITION 2
PARTITIONED TOPIC

```go
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)

}
```

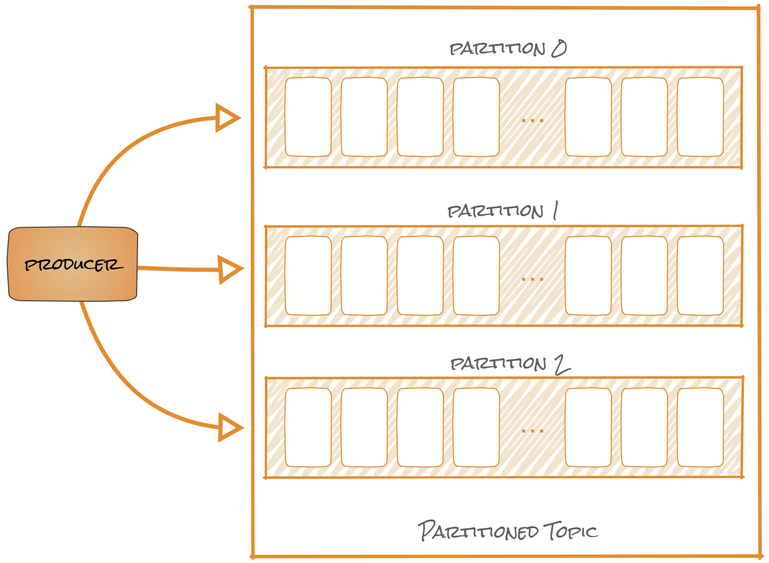
Producing to Kafka - No Key



Producing to Kafka - With Key



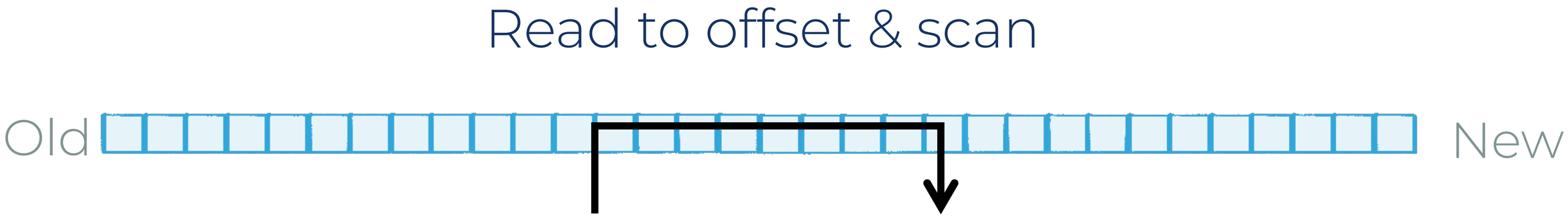
Producers



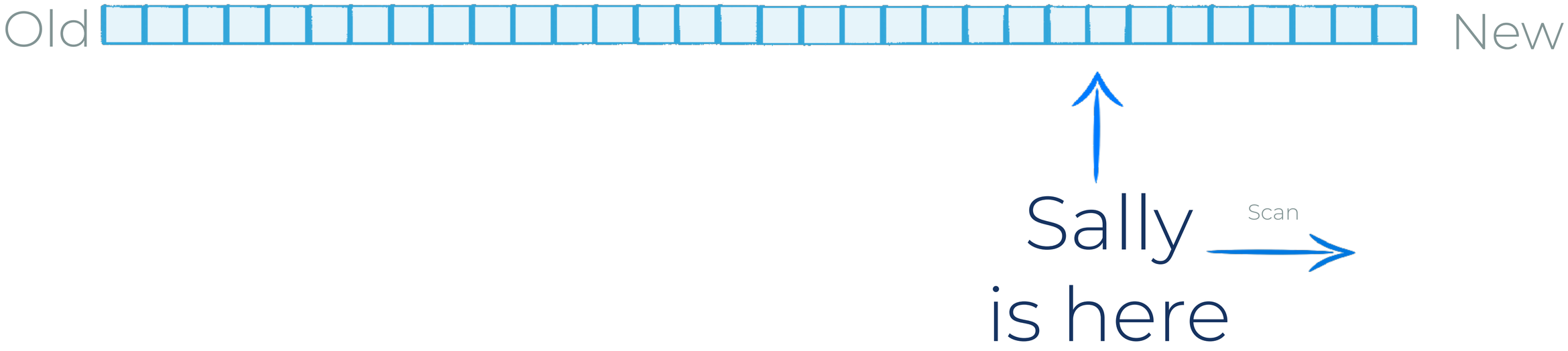
- **A client application**
- **Puts messages into topics**
- **Handles partitioning, network protocol**
- **Java, Go, .NET, C/C++, Python**
- **Also every other language**
  Plus REST proxy if not

PUB / SUB

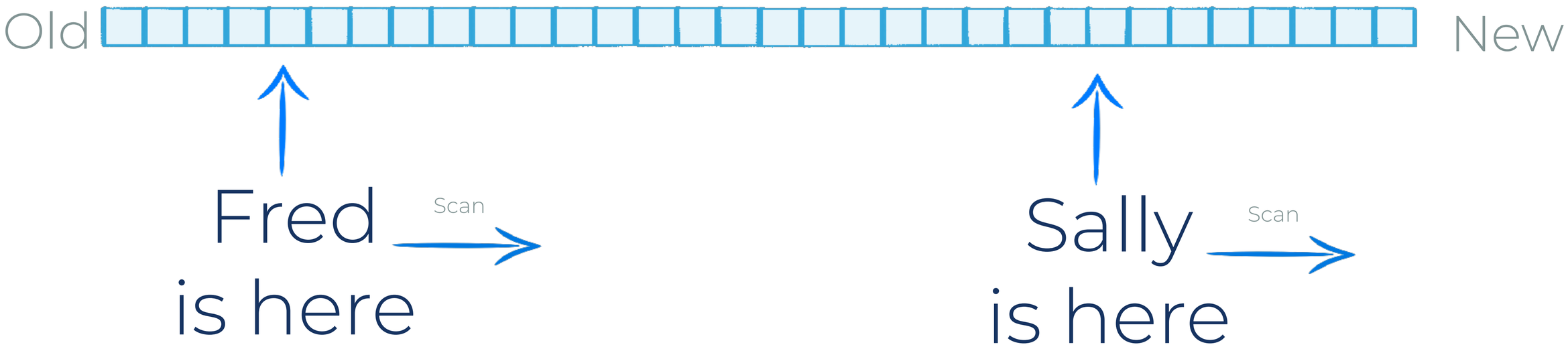
Consuming data - access is only sequential



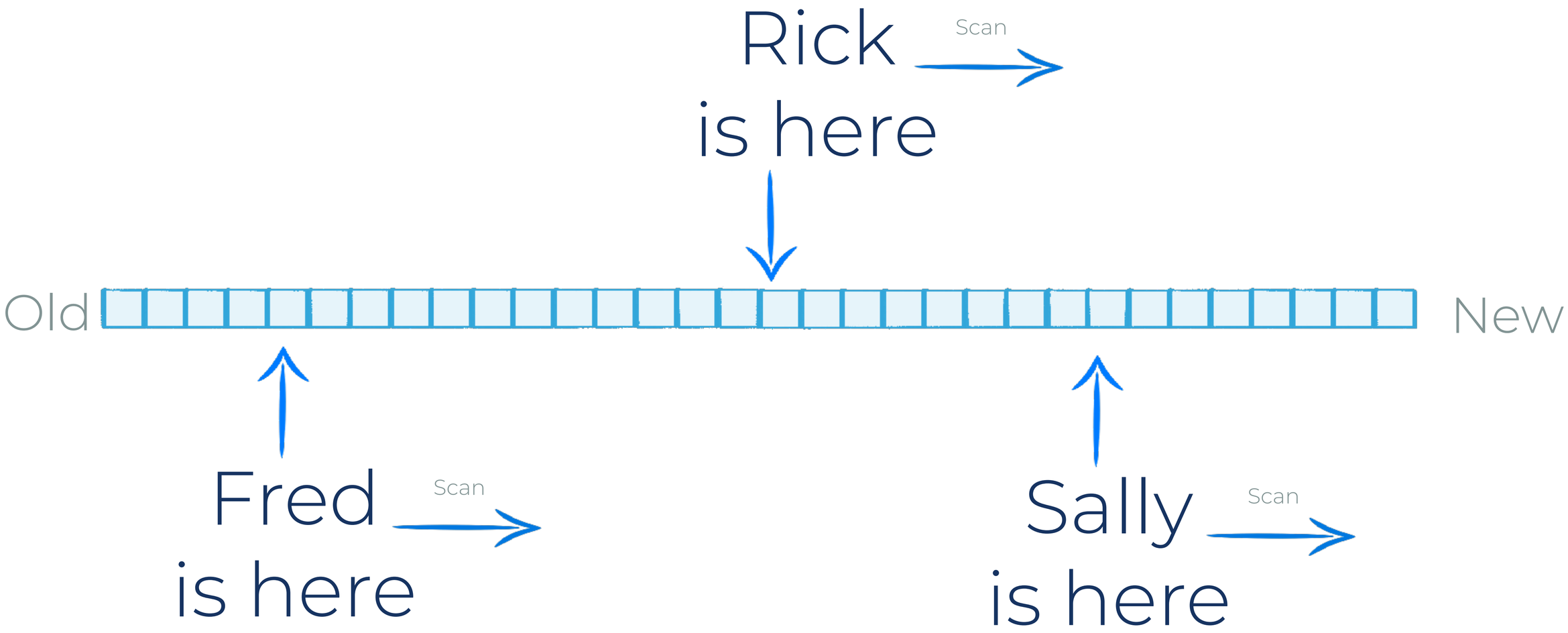
Consumers have a position of their own



Consumers have a position of their own

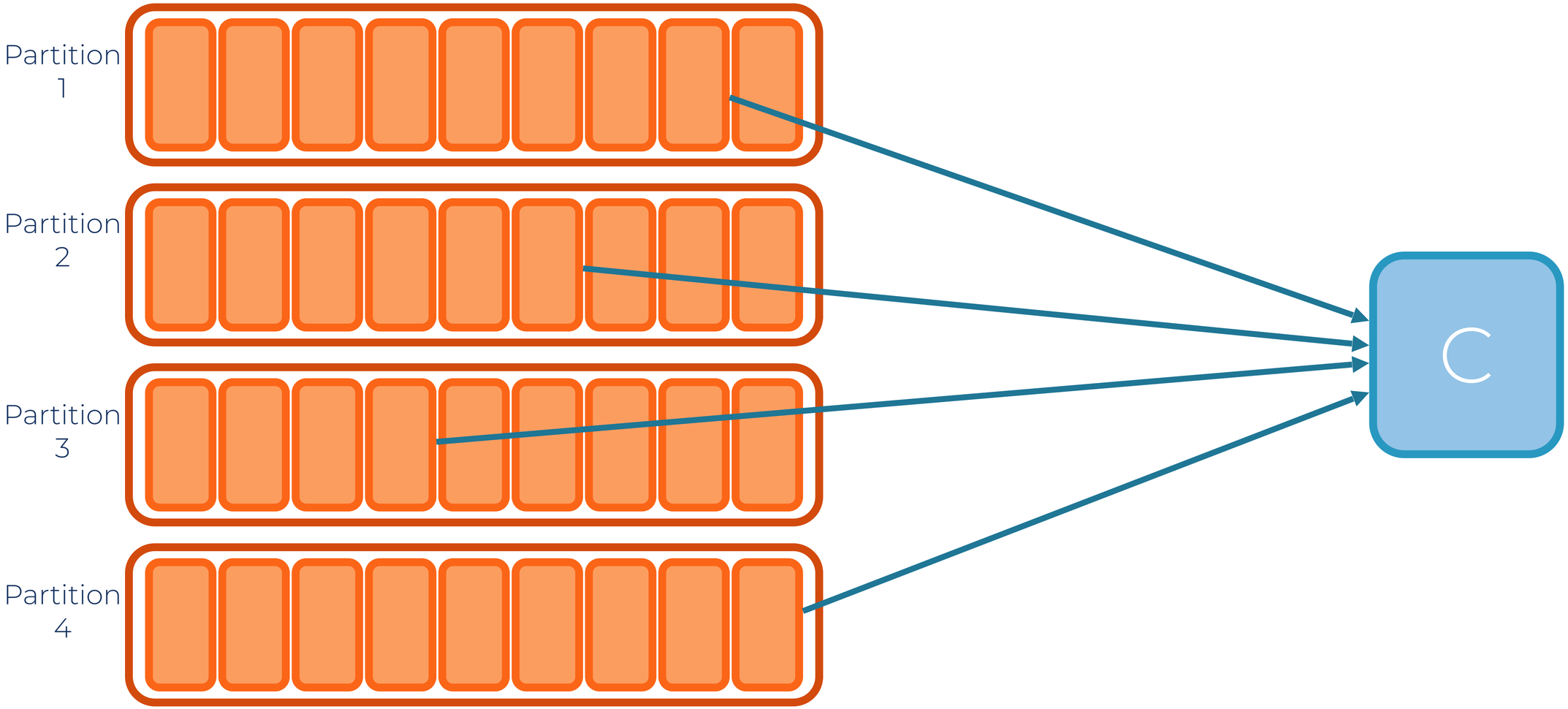


Consumers have a position of their own

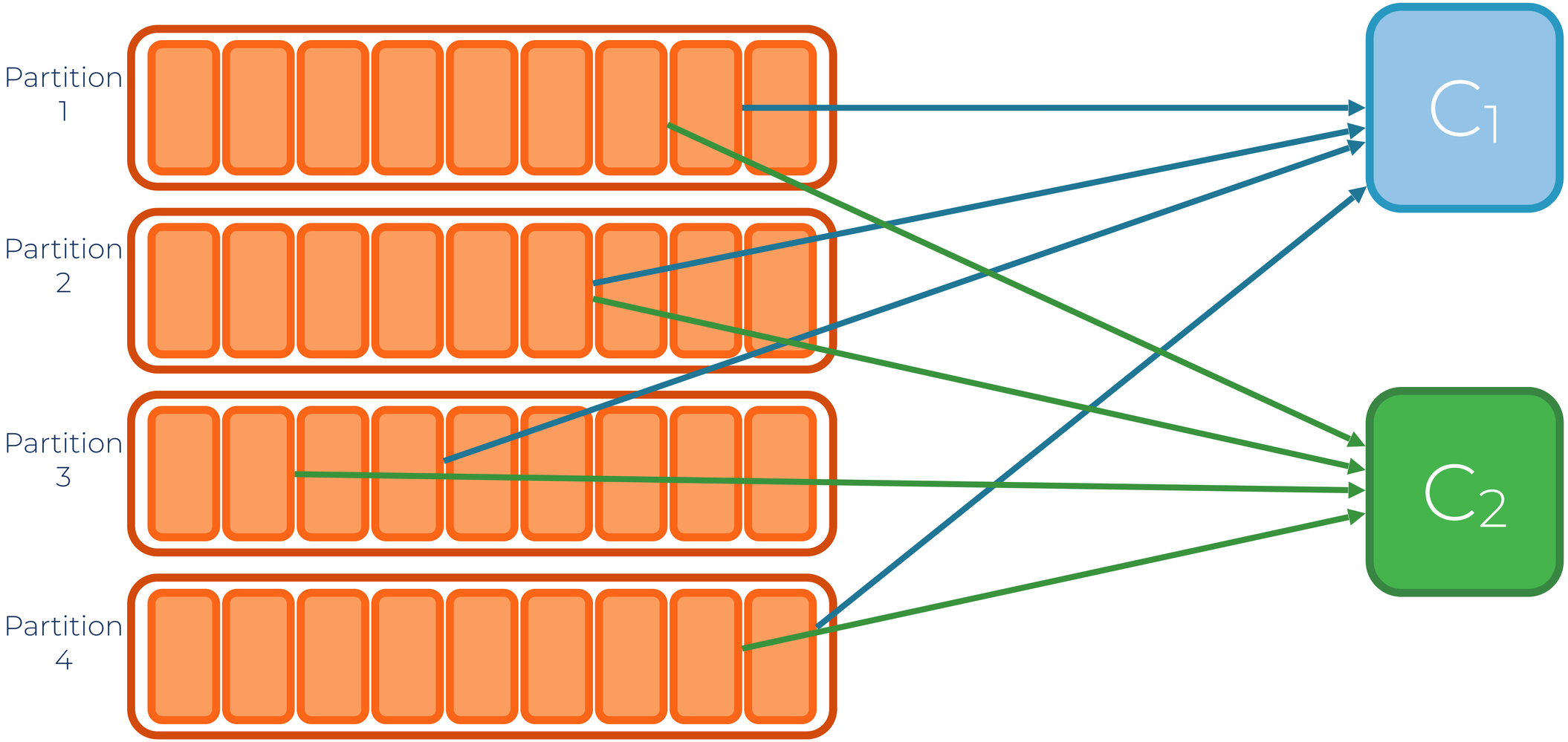


```
c, _ := kafka.NewConsumer(&cm)
defer c.Close()
c.Subscribe(topic, nil)

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))
        }
    }
}
```

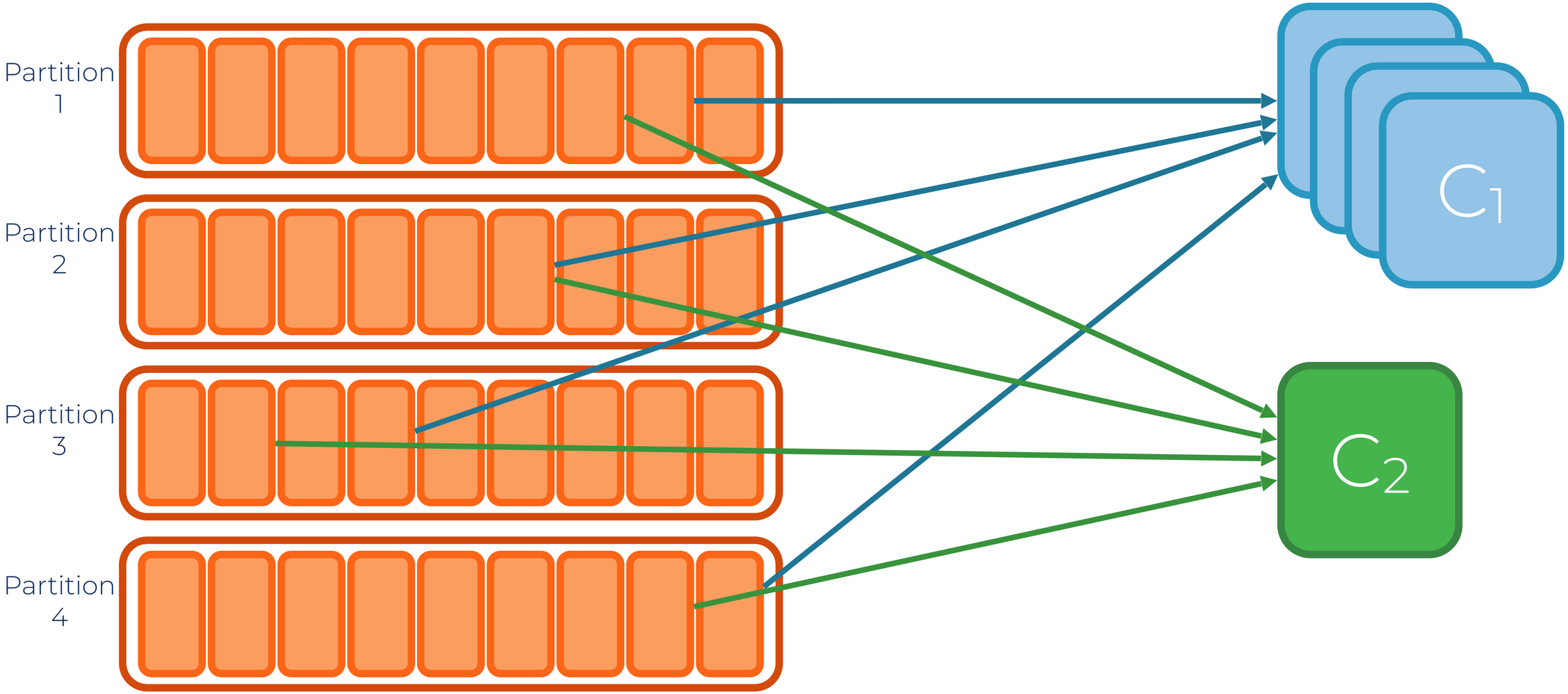
Consuming From Kafka - Single Consumer



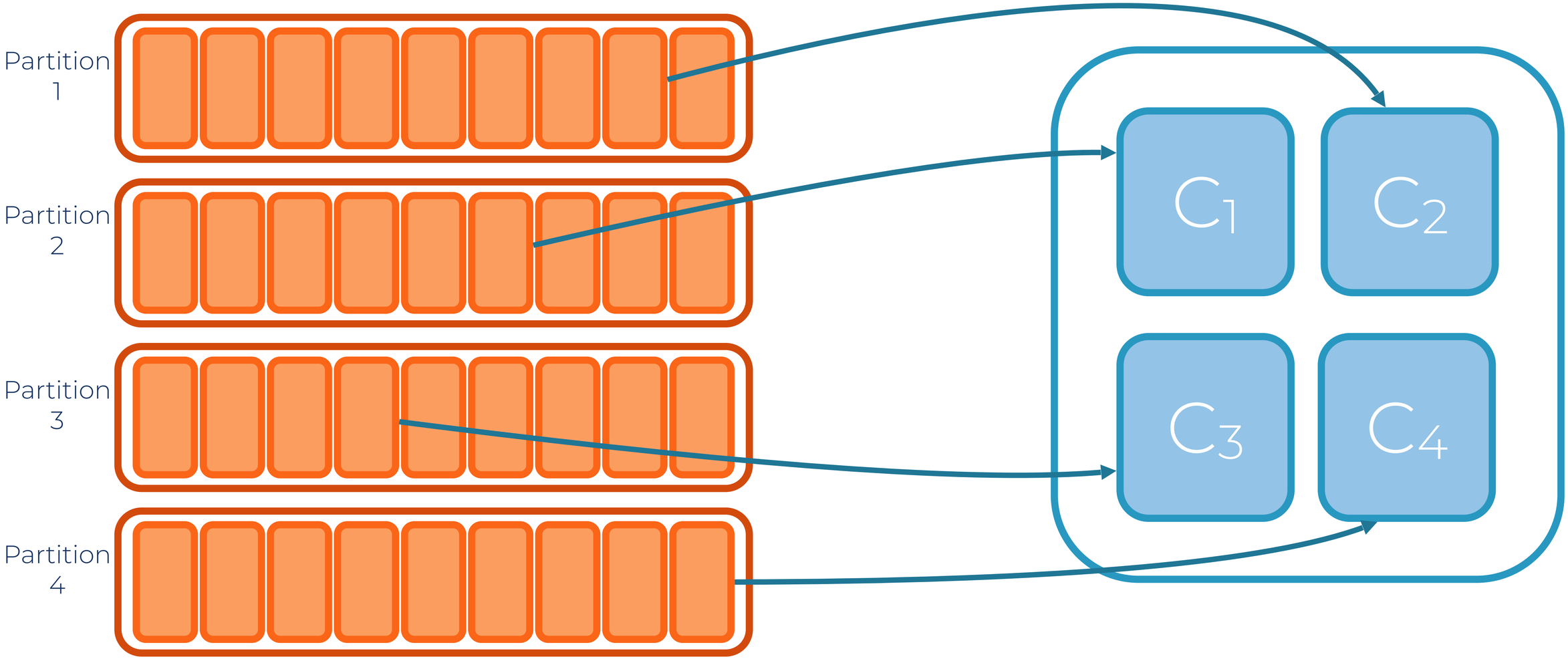
Consuming From Kafka - Multiple Consumers



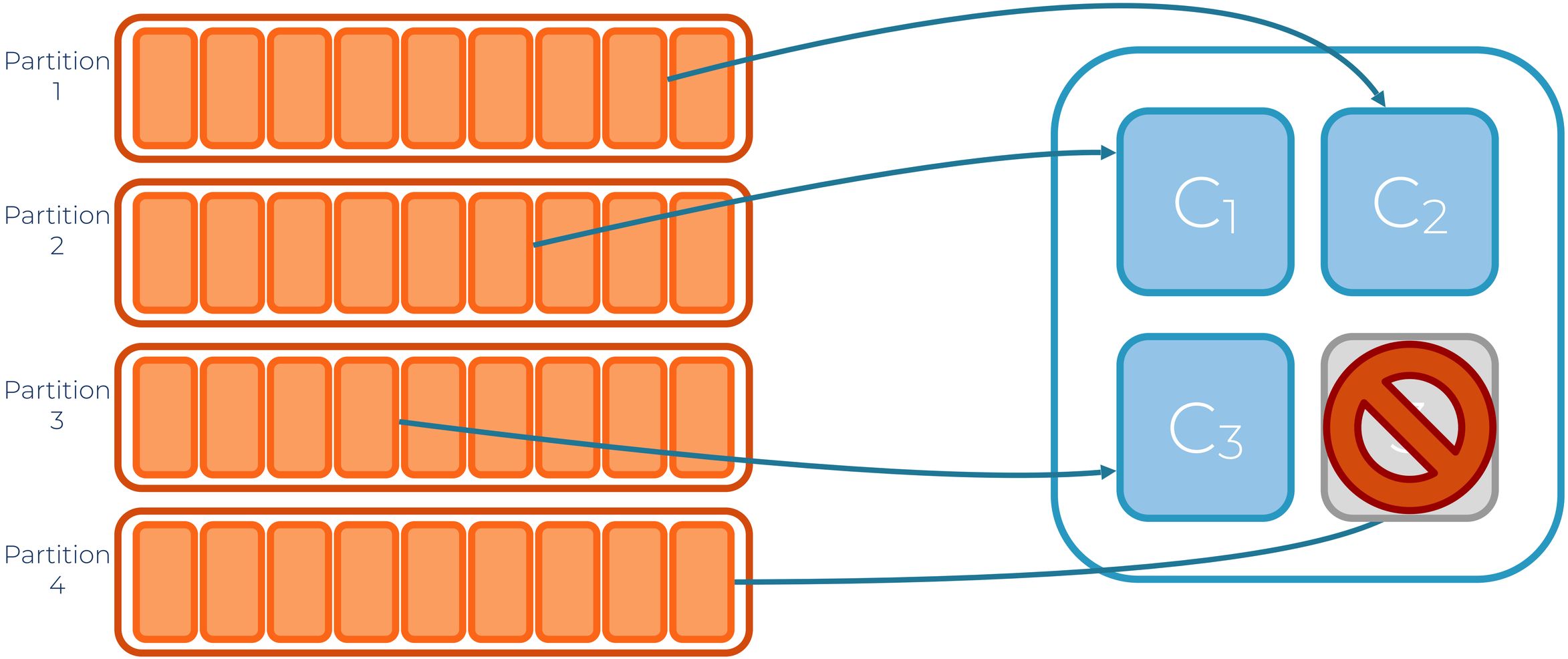
Consuming From Kafka - Grouped Consumers



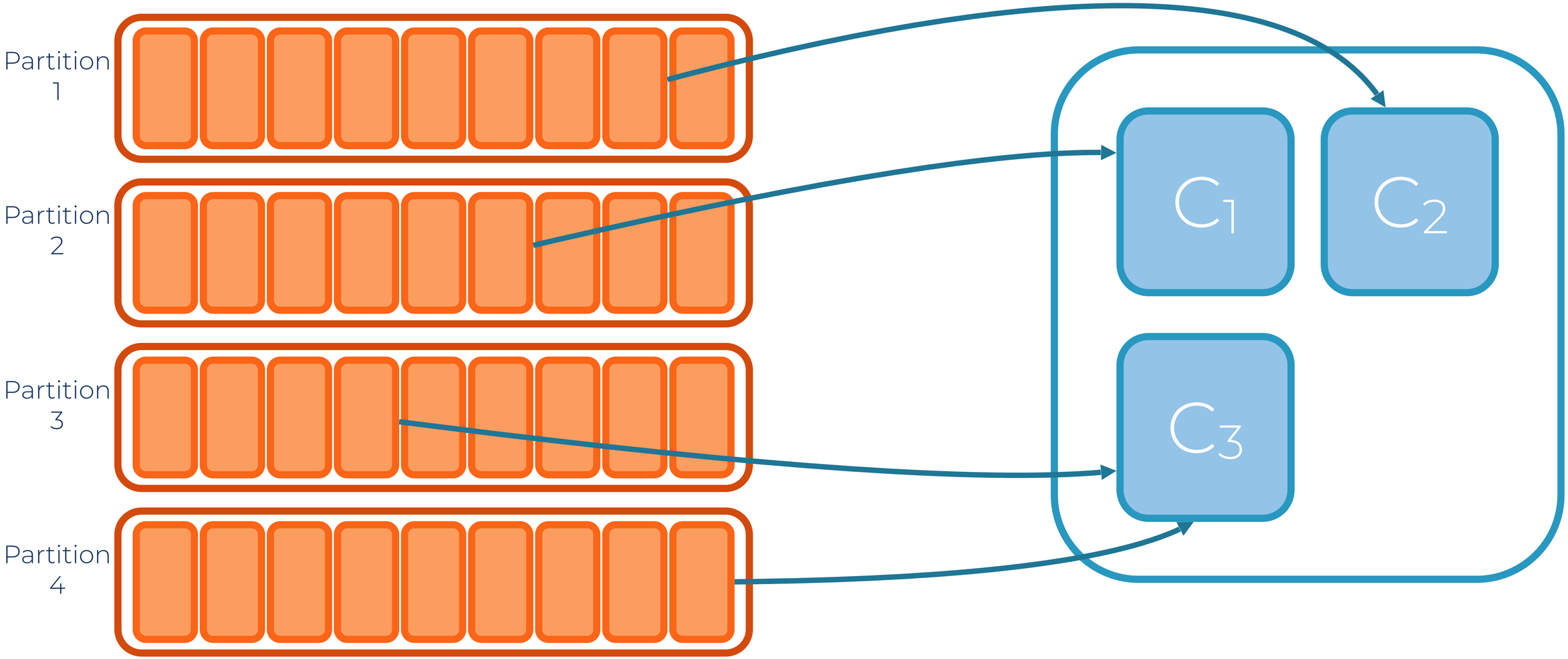
Consuming From Kafka - Grouped Consumers



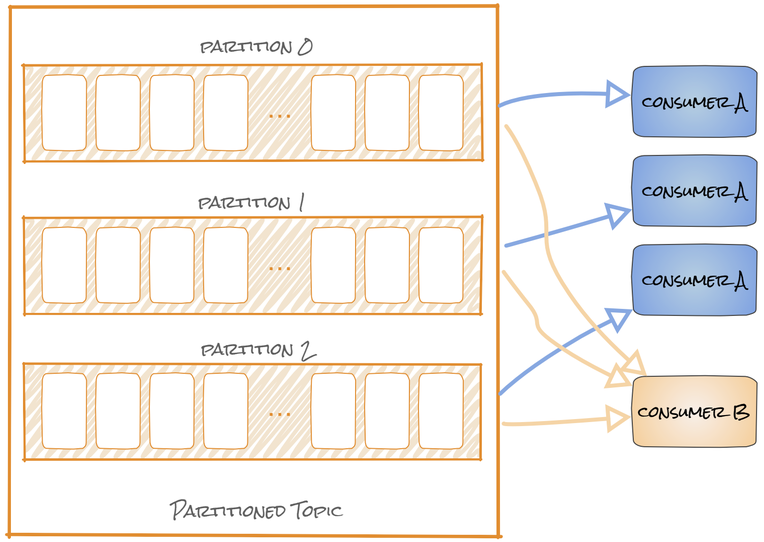
Consuming From Kafka - Grouped Consumers



Consuming From Kafka - Grouped Consumers



Consumers



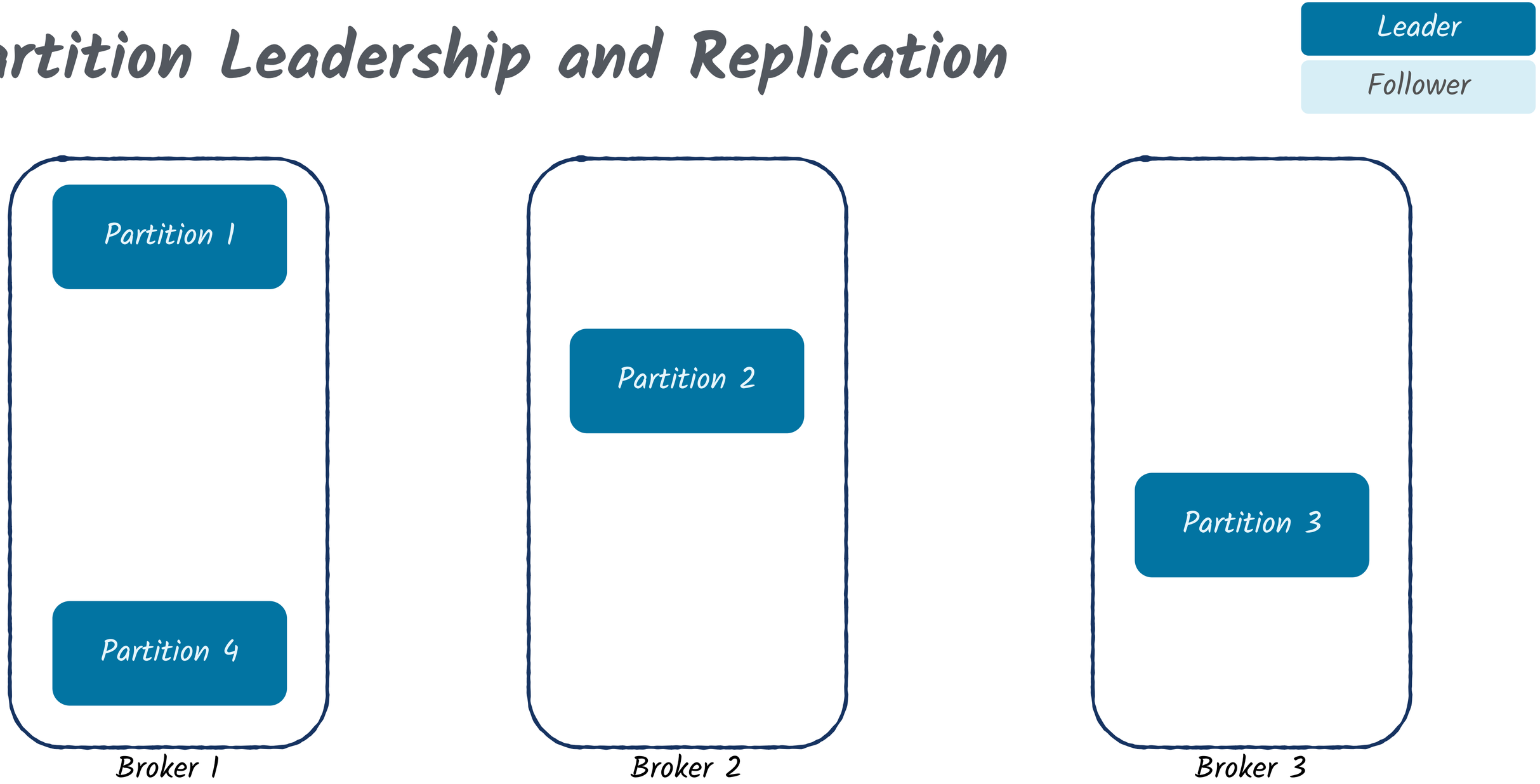
- **A client application**
- **Reads messages from topics**
- **Horizontally, elastically scalable (if stateless)**
- **Java, Go, .NET, C/C++, Python, everything else**
  Plus REST proxy if not

BROKERS

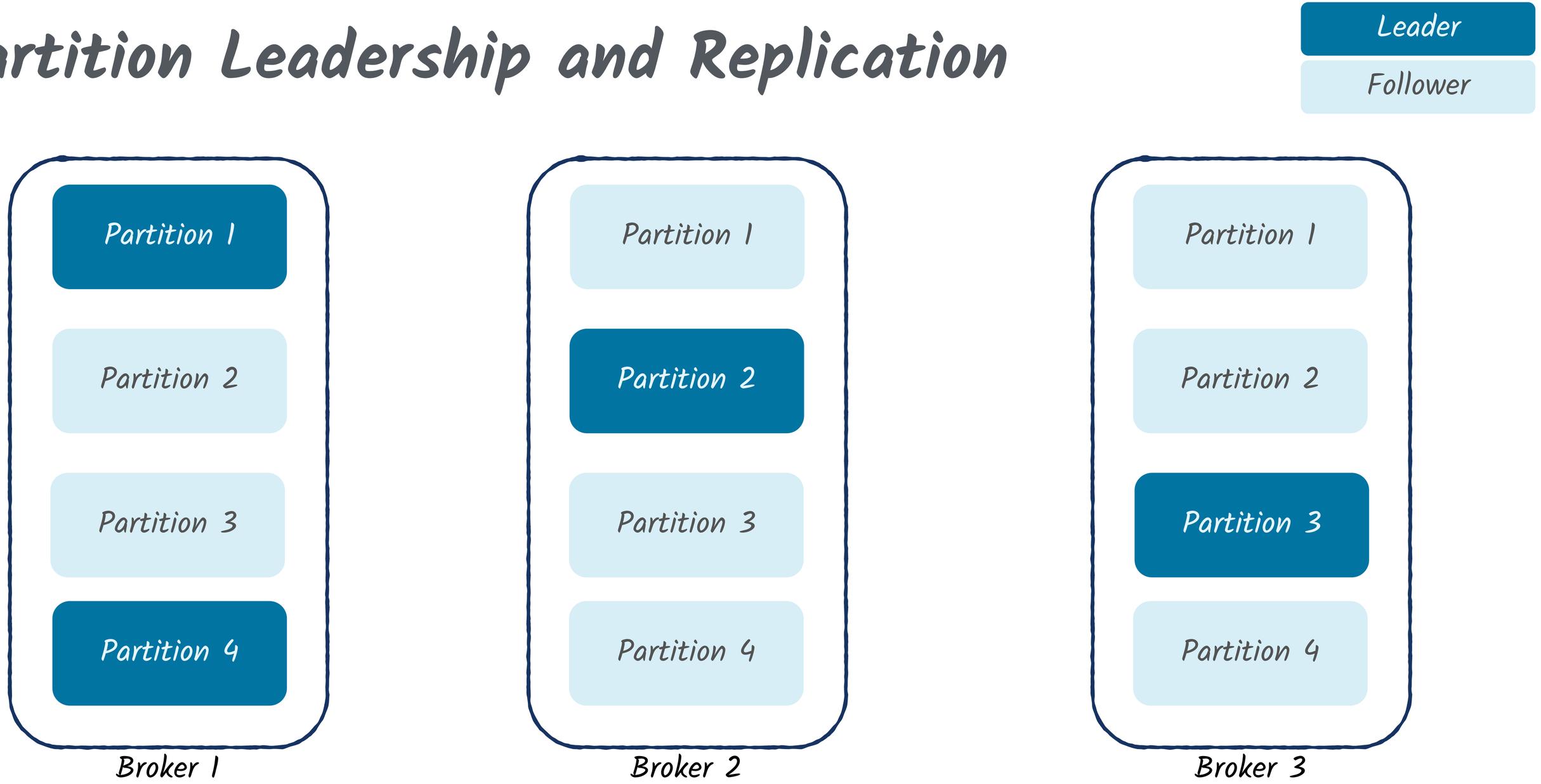
and REPLICATION

@rmoff | #GOTOpia | @confluentinc

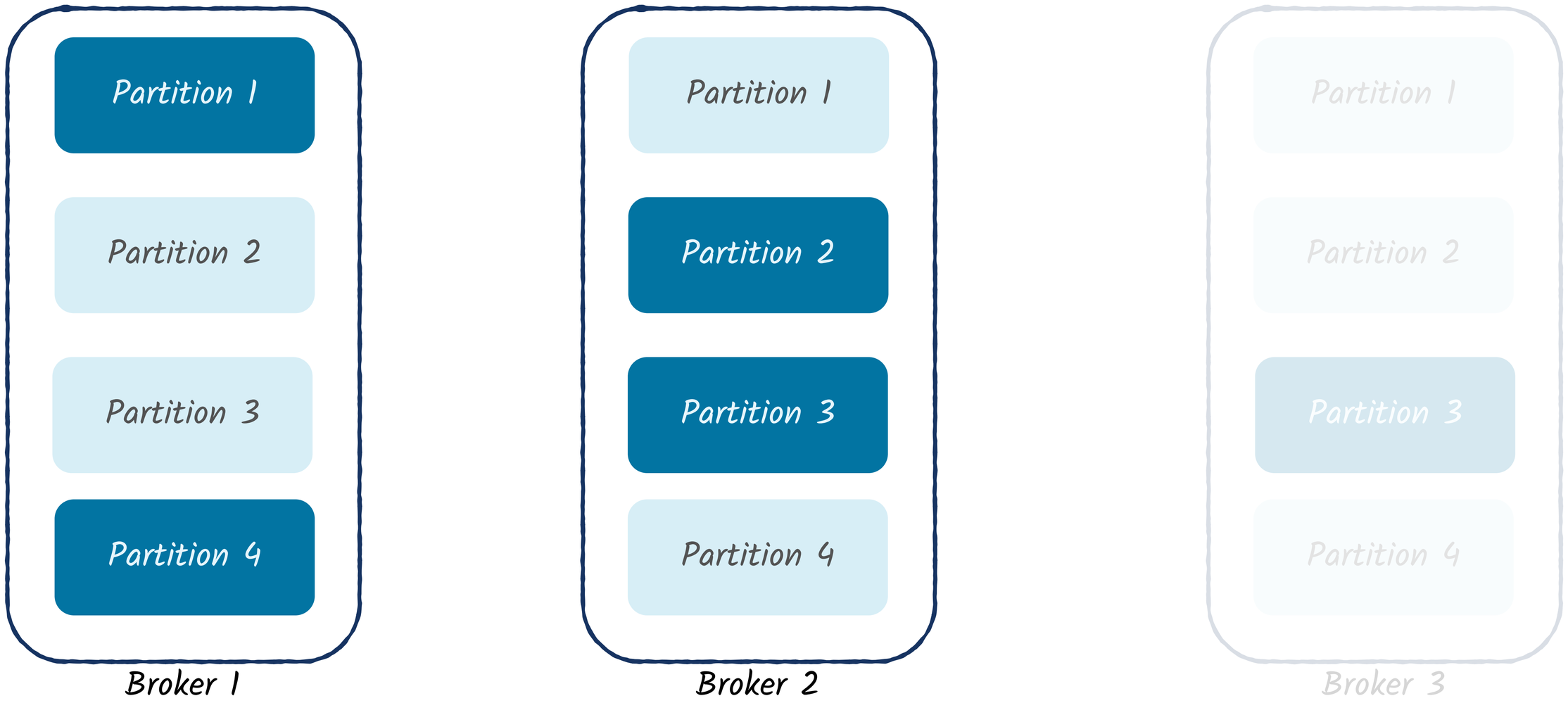
Partition Leadership and Replication



Partition Leadership and Replication



Partition Leadership and Replication



So far, this is

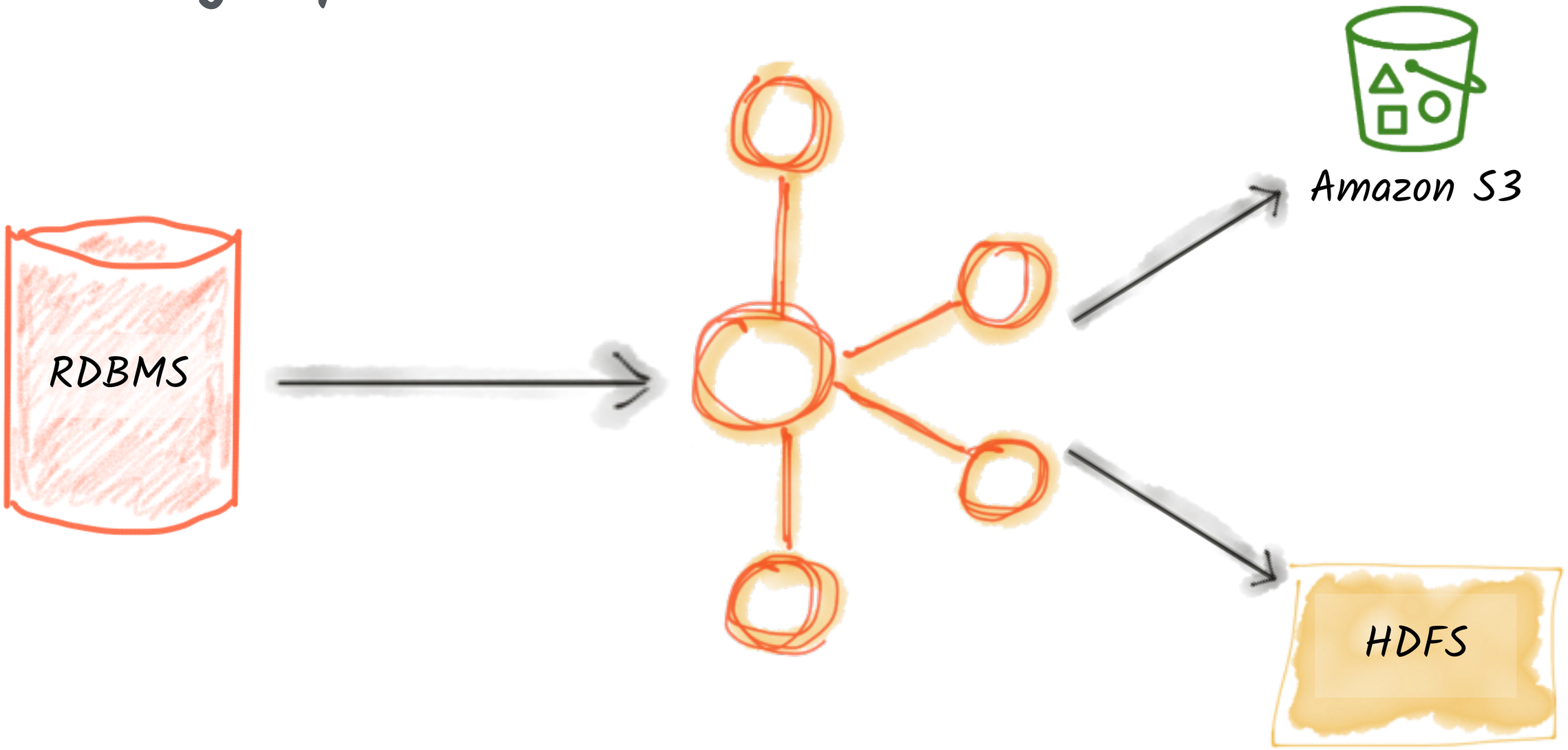
Pretty good

So far, this is

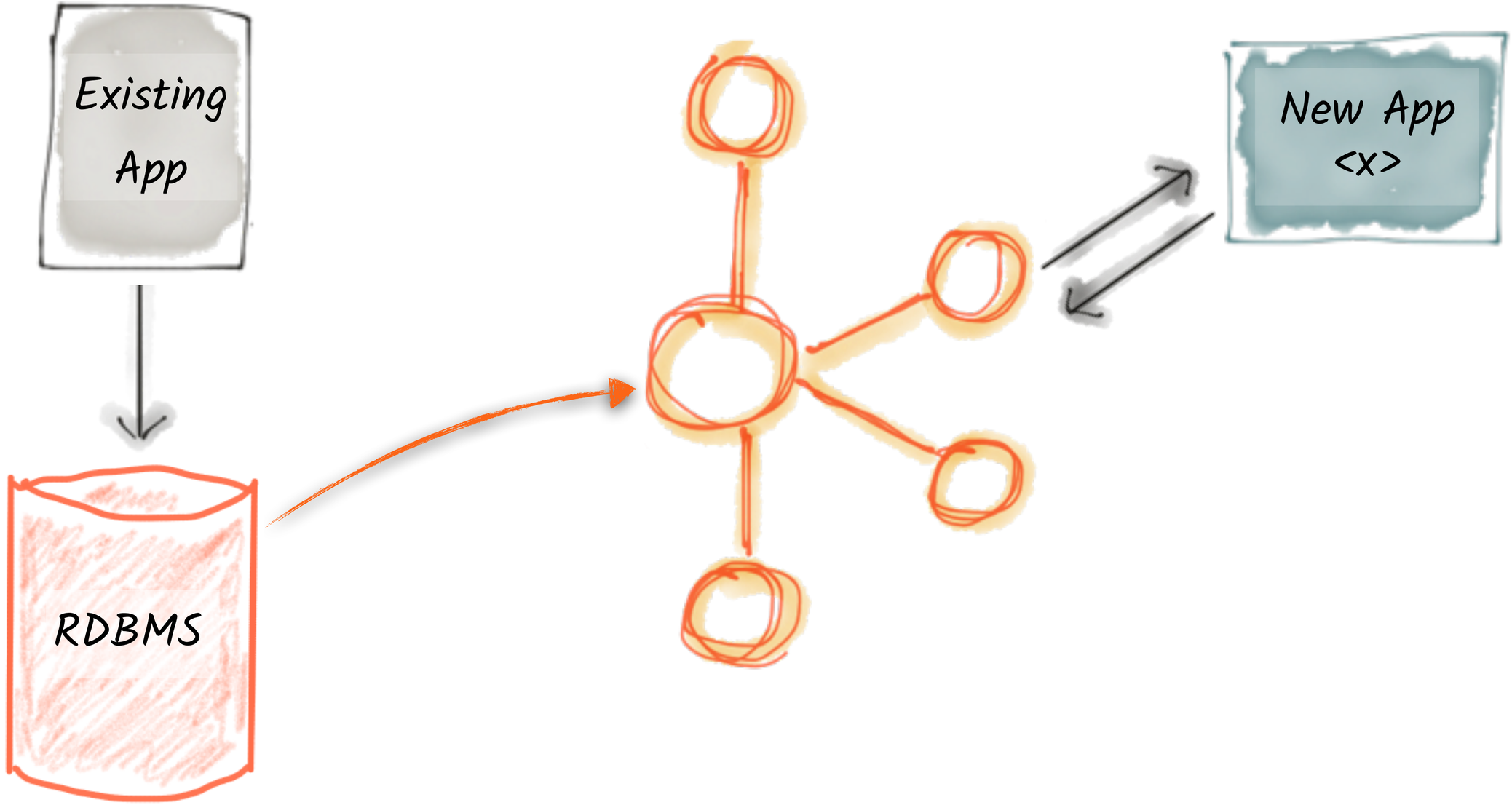
Pretty good

but I've not finished yet...

Streaming Pipelines

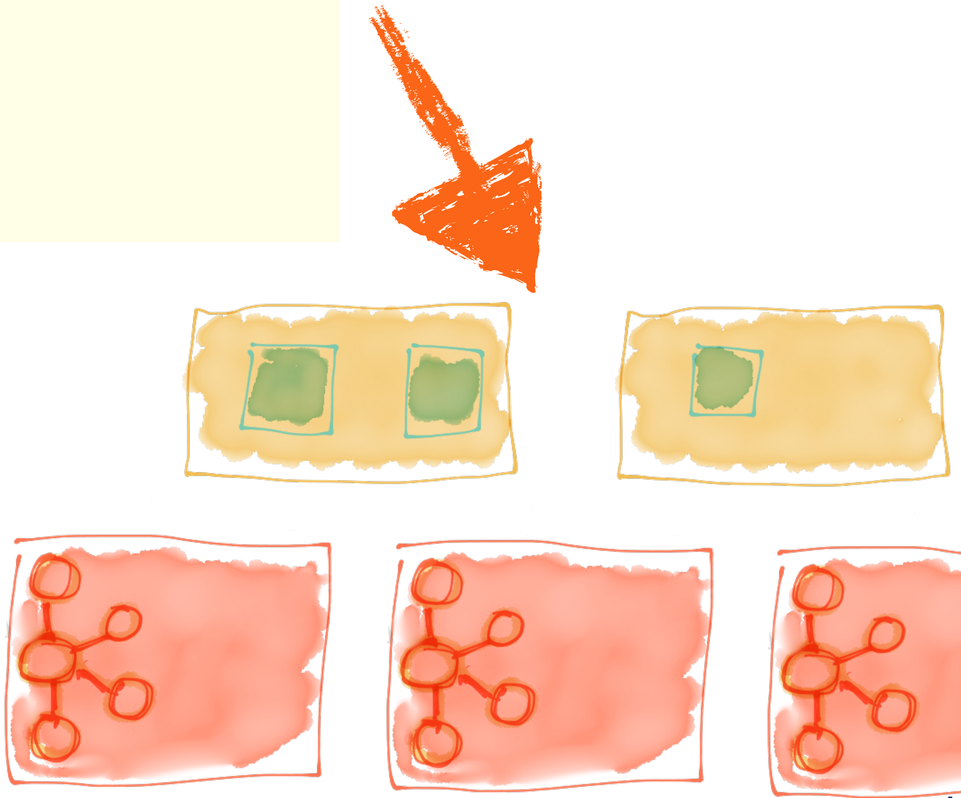


Evolve processing from old systems to new



Streaming Integration with Kafka Connect

Sources



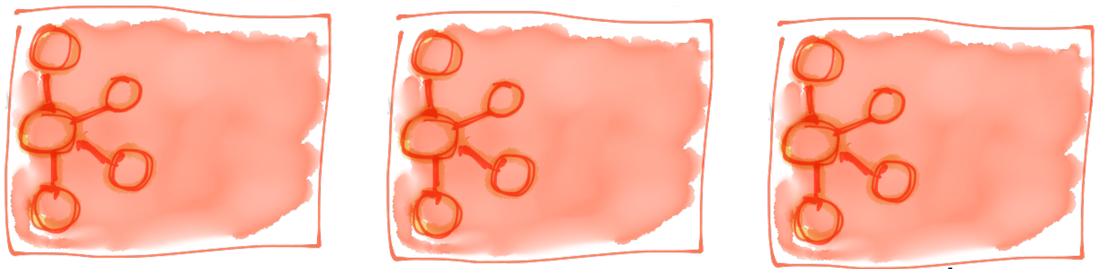
Kafka Connect

Kafka Brokers

Streaming Integration with Kafka Connect

Sinks

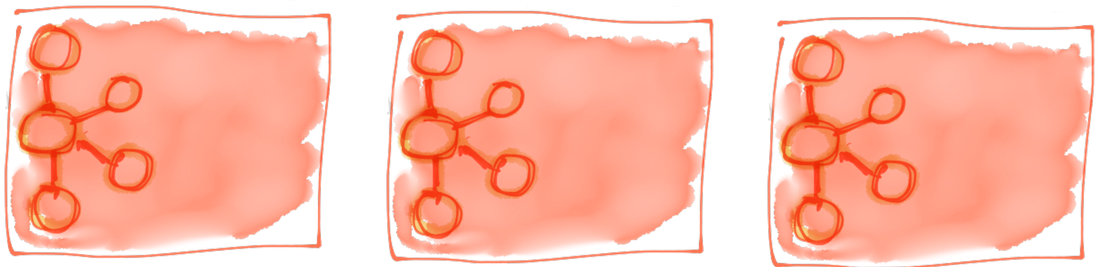
Kafka Connect



Kafka Brokers

Streaming Integration with Kafka Connect

Kafka Connect



Kafka Brokers

Look Ma, No Code!

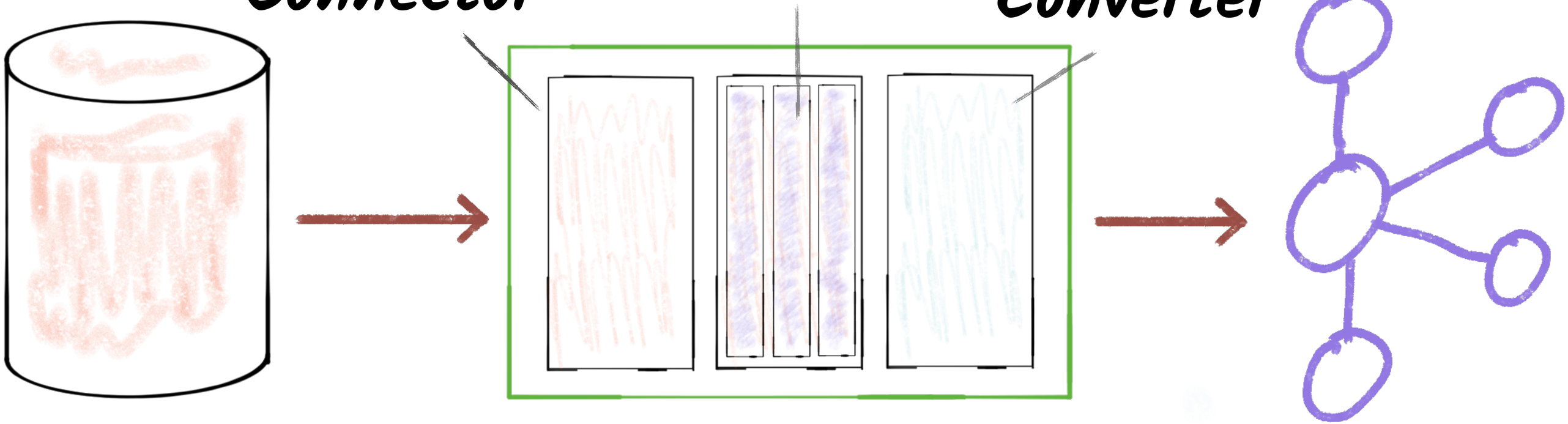
```
{
  "connector.class":
    "io.confluent.connect.jdbc.JdbcSourceConnector",
  "connection.url":
    "jdbc:mysql://asgard:3306/demo",
  "table.whitelist":
    "sales,orders,customers"
}
```

Extensible

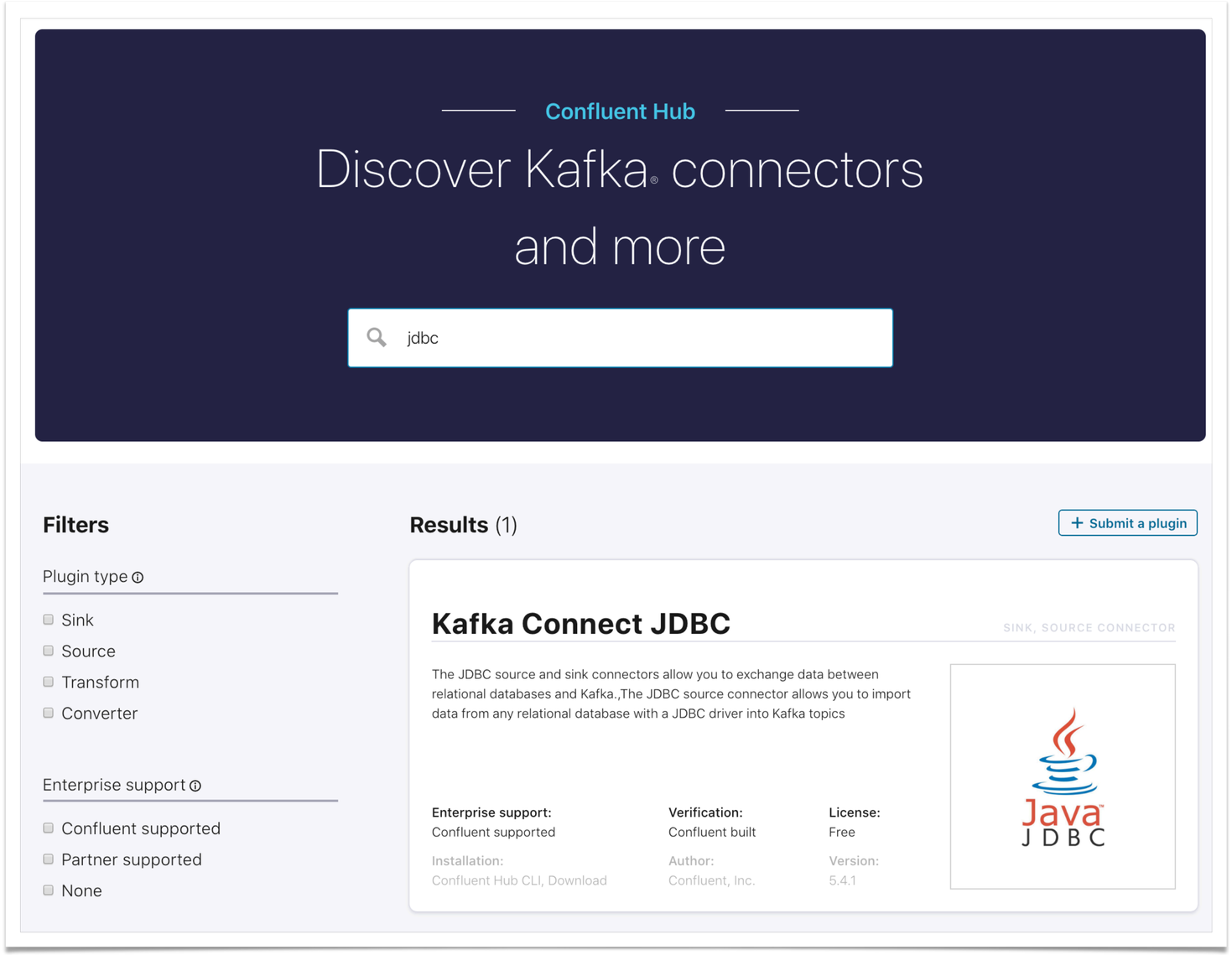
Connector

Transform(s)

Converter



hub.confluent.io

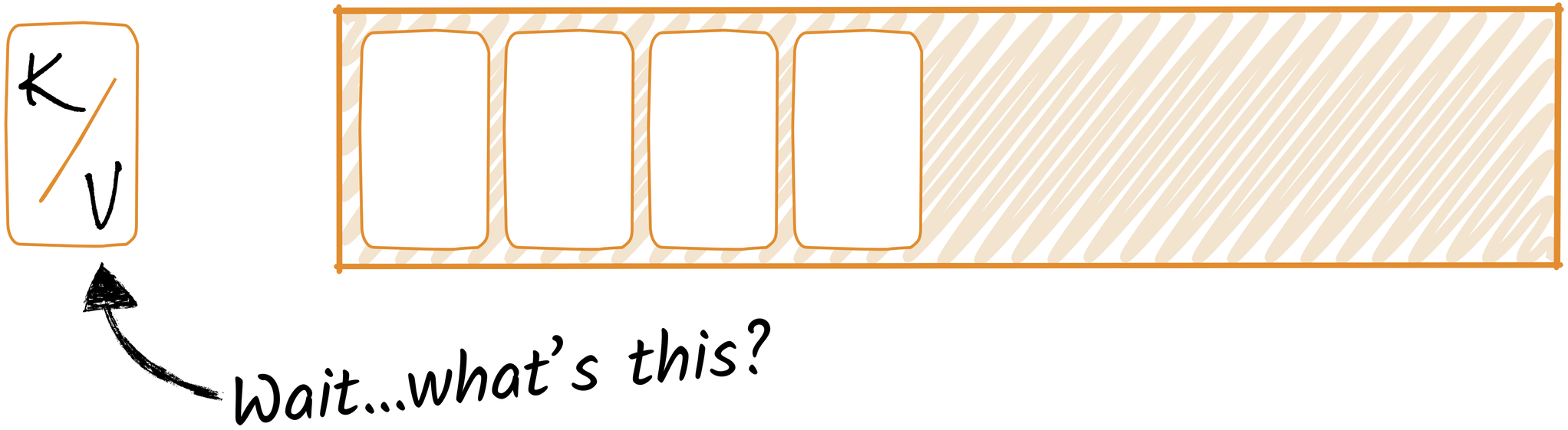



K
V


K
V


K
V


K
V


K
V
Wait...what's this?

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 Blvd9561038 41 56
38.6988889 121  16 15.98999977
-121.271111  10 4284781 650345  52
```

Serialisation & Schemas

Avro Protobuf JSON Schema JSON CSV

Serialisation & Schemas

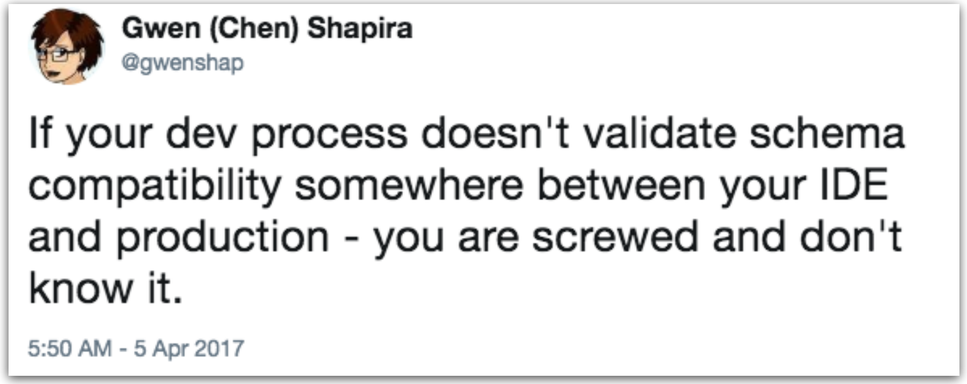
Avro 👍

Protobuf 👍

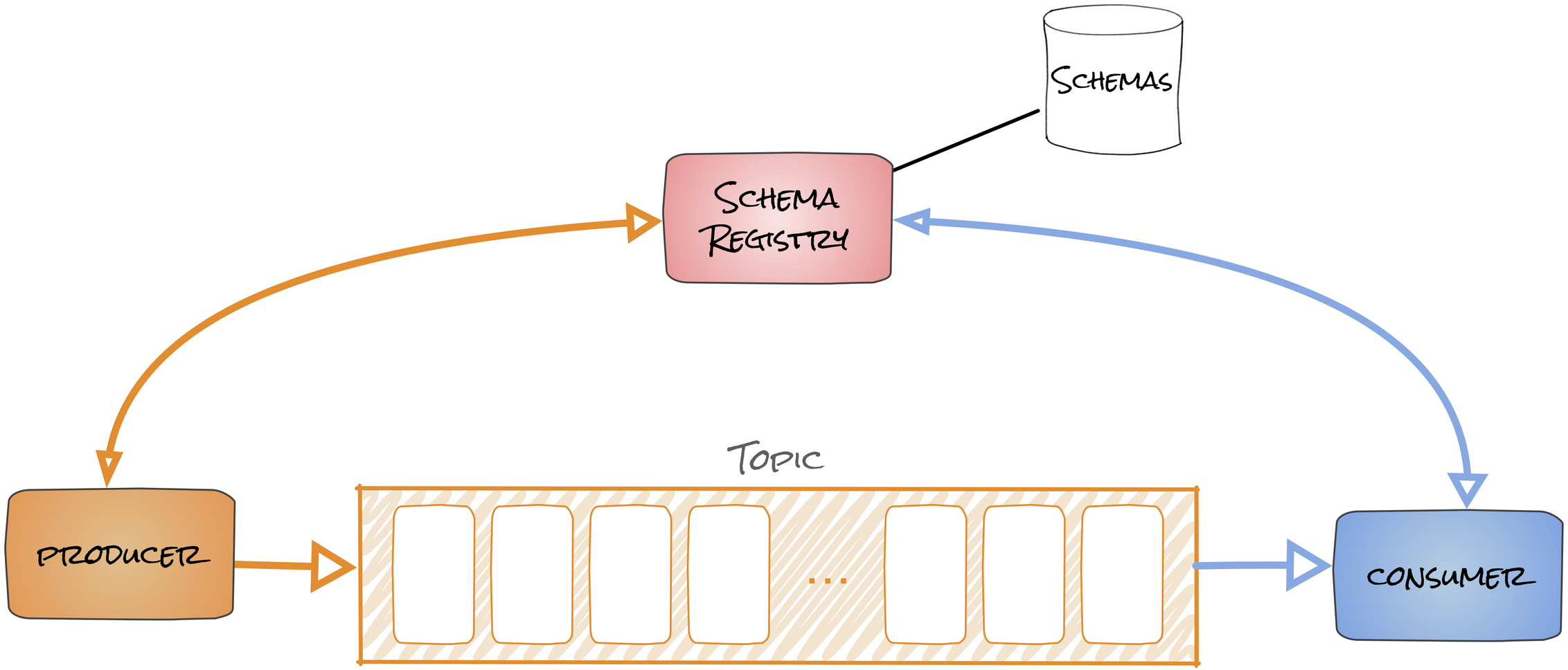
JSON Schema 👍

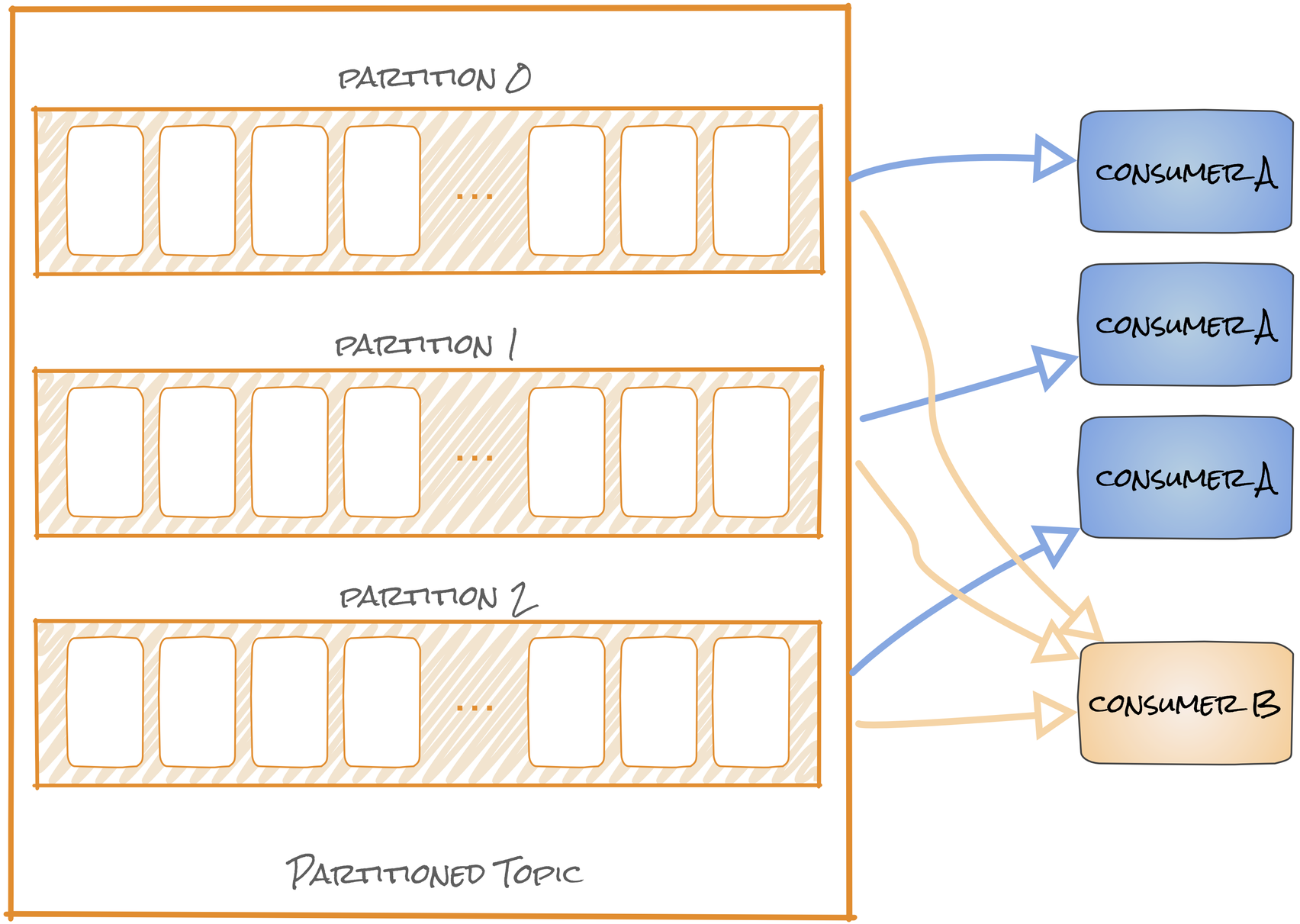
JSON 😬

CSV 



https://rmoff.dev/qcon-schemas


SCHEMAS
SCHEMA REGISTRY
TOPIC
PRODUCER
CONSUMER


PARTITION 0
PARTITION 1
PARTITION 2
PARTITIONED TOPIC
CONSUMER A
CONSUMER A
CONSUMER A
CONSUMER B

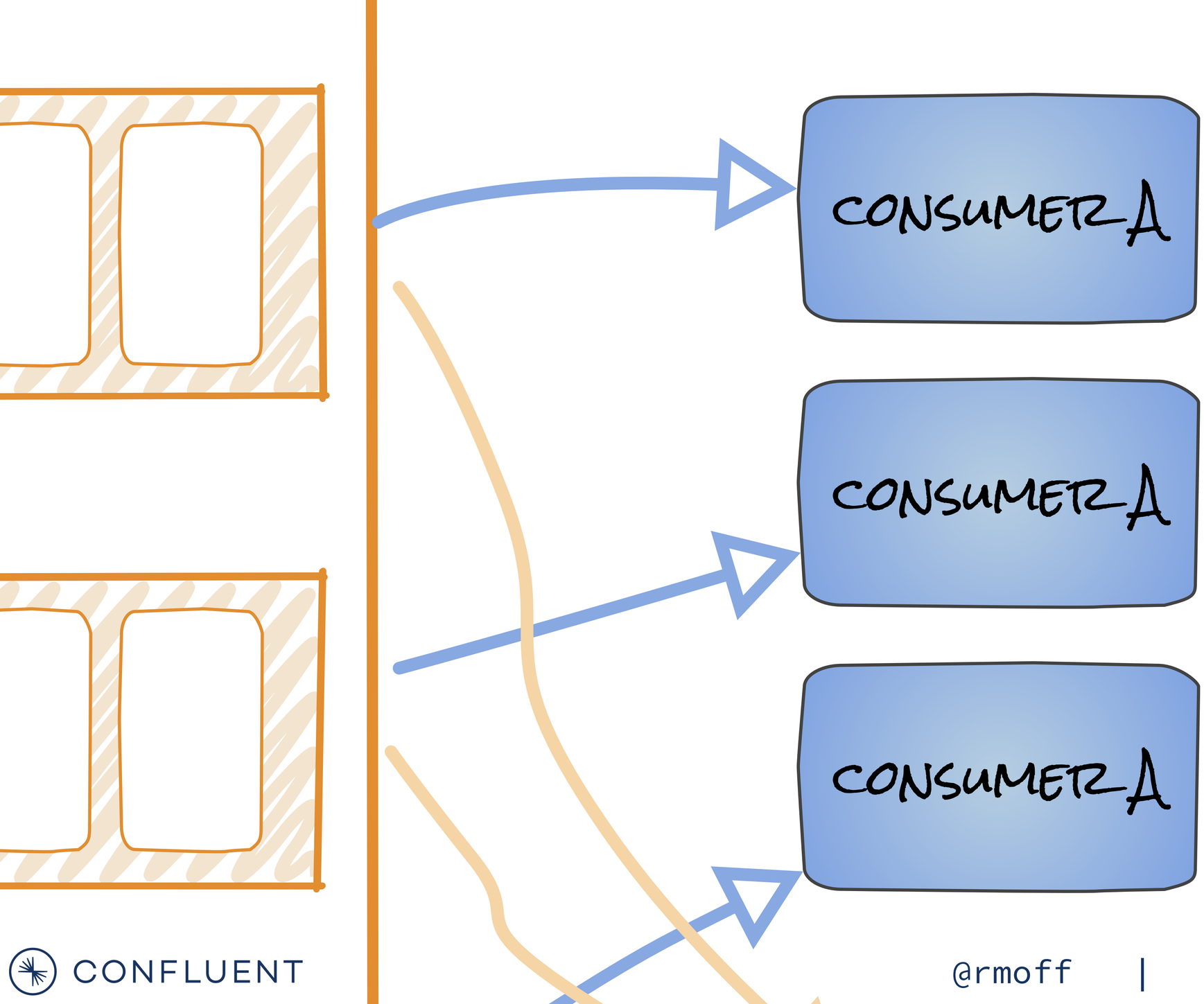
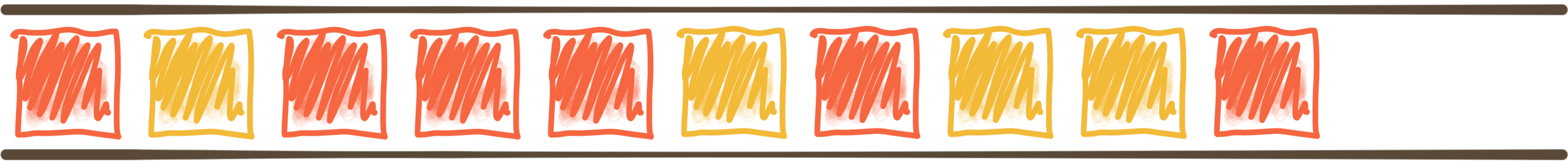

CONSUMER A
CONSUMER A
CONSUMER A

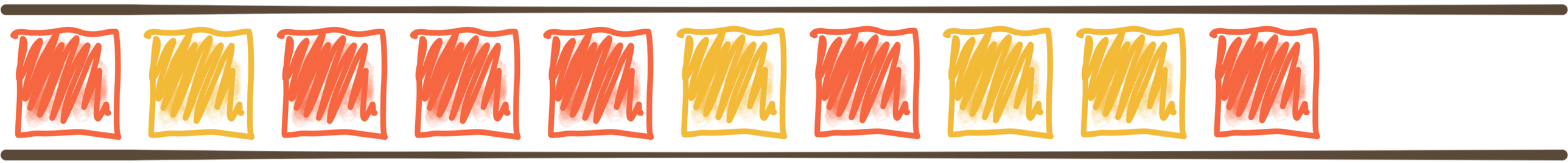
Photo by Franck V. on Unsplash

Streams of events



Time

Stream Processing



Stream: widgets

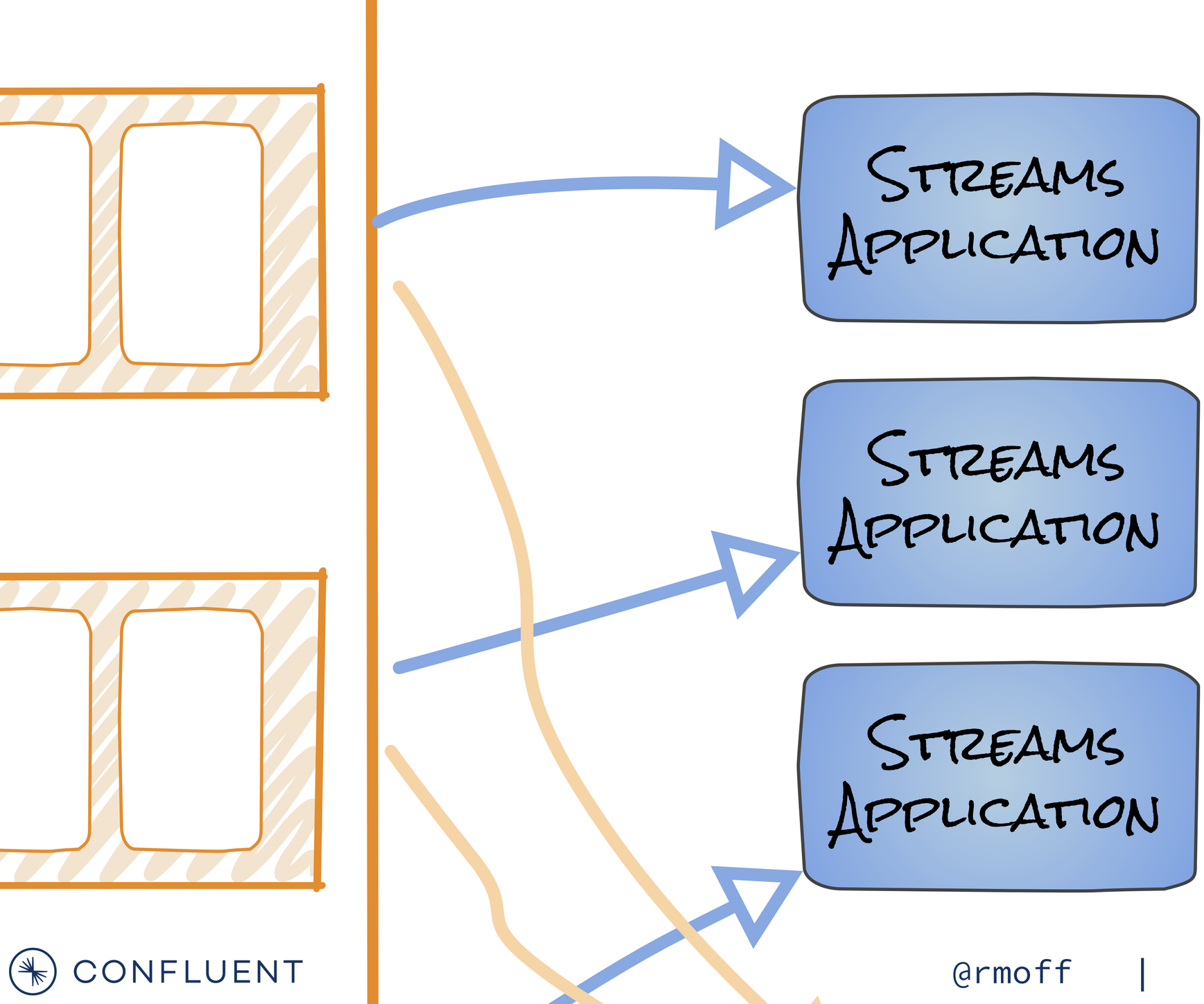
Stream: widgets_red

Stream Processing with Kafka Streams

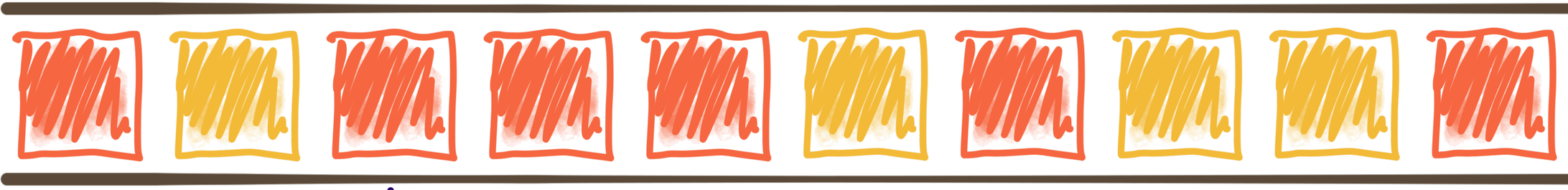
Stream:
widgets

```
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));
```

Stream:
widgets_red


Streams Application
Streams Application
Streams Application

Stream Processing with ksqlDB



Stream:
widgets

ksqlDB

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

Stream:
widgets_red

Photo by Franck V. on Unsplash

```
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

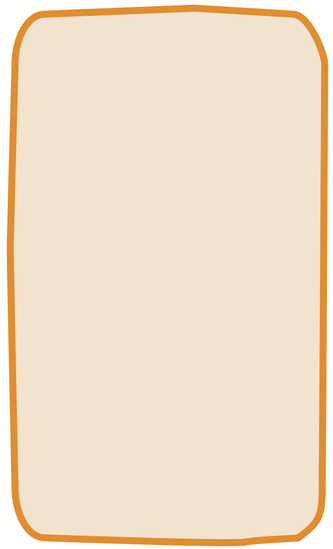
```
CREATE SINK CONNECTOR dw WITH (
'connector.class' = 'S3Connector',
'topics' = 'widgets'
…);
```

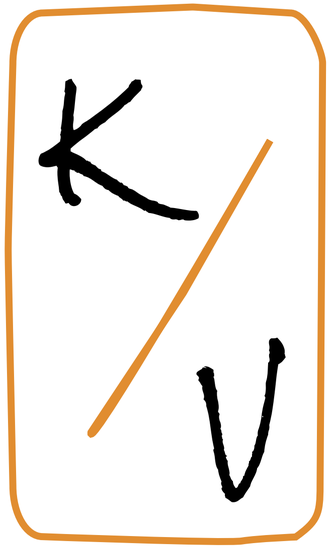
Photo by Franck V. on Unsplash

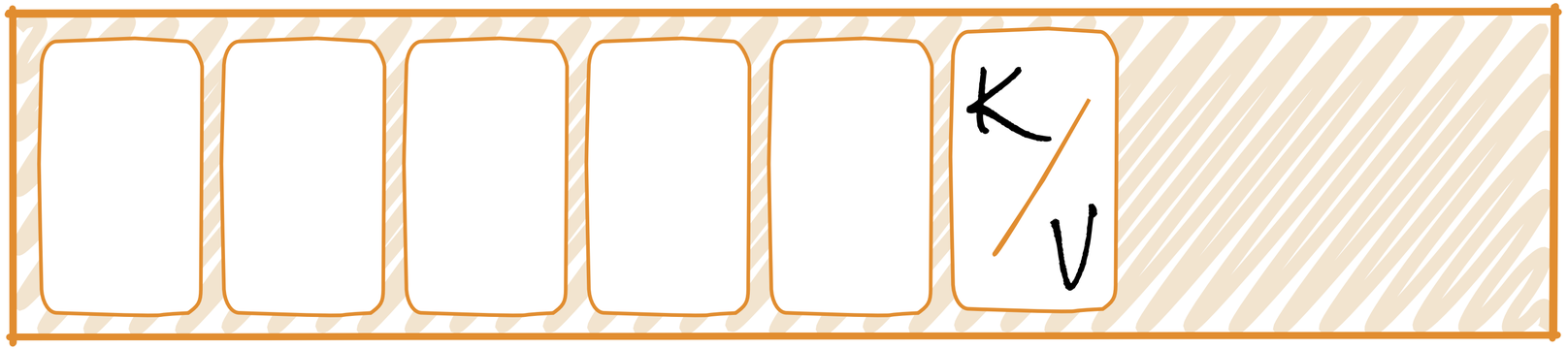
CONFLUENT

DEMO

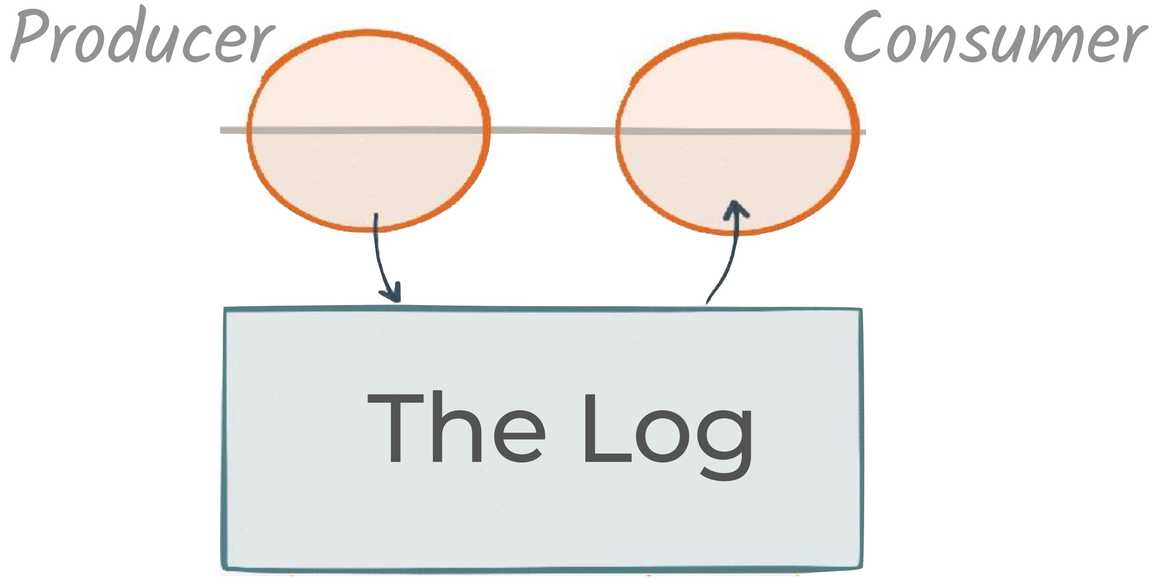
Summary

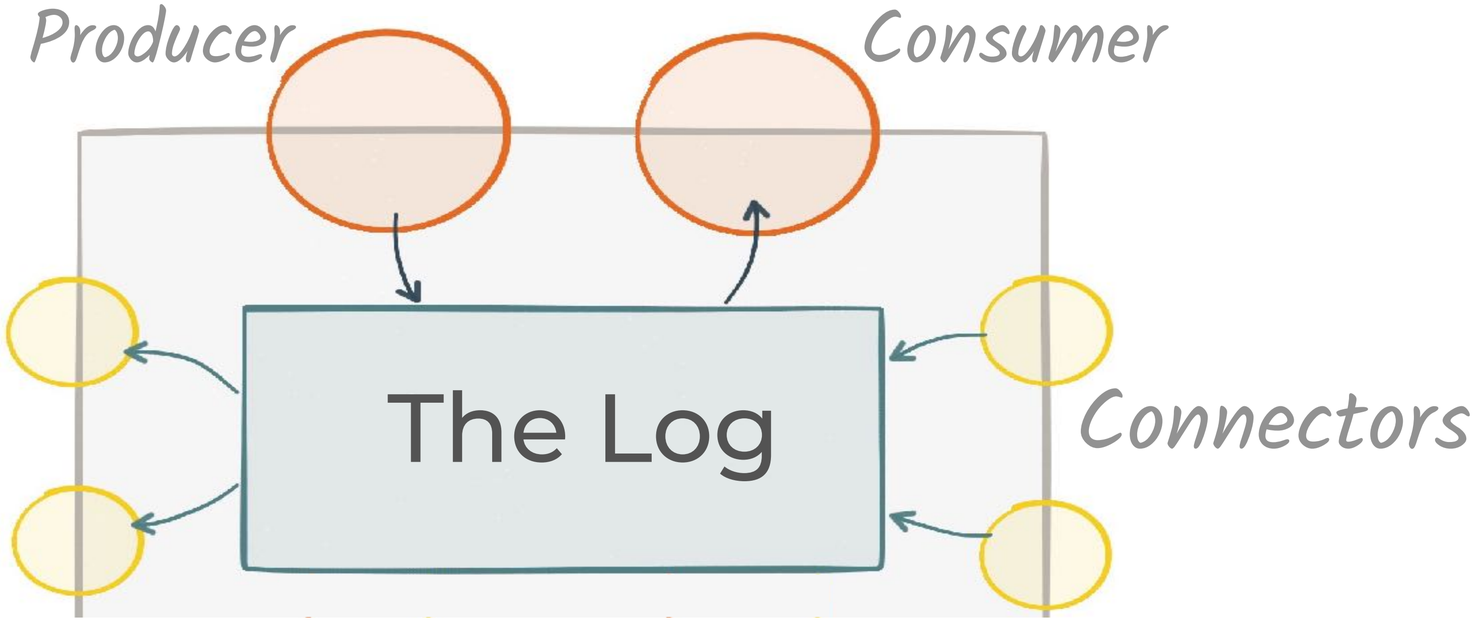


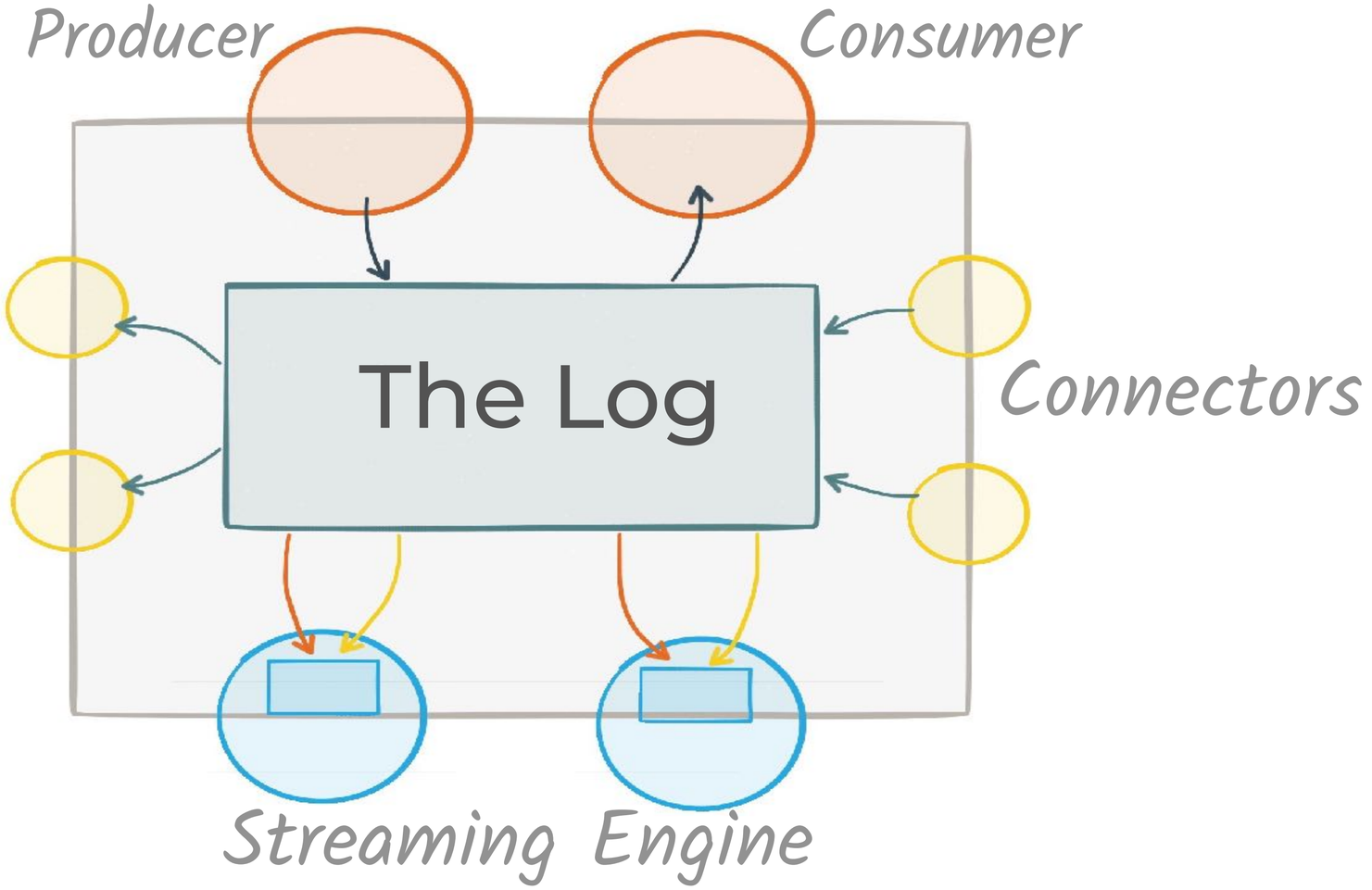

K
V


K
V

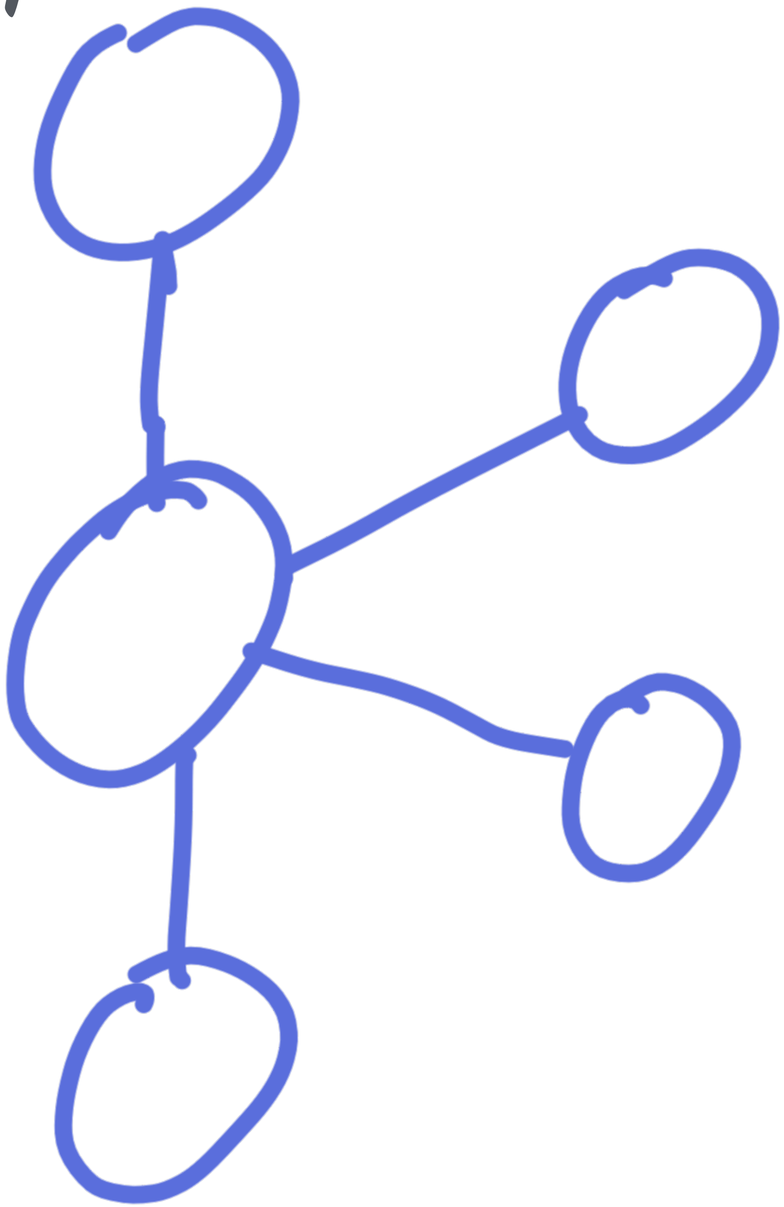
The Log

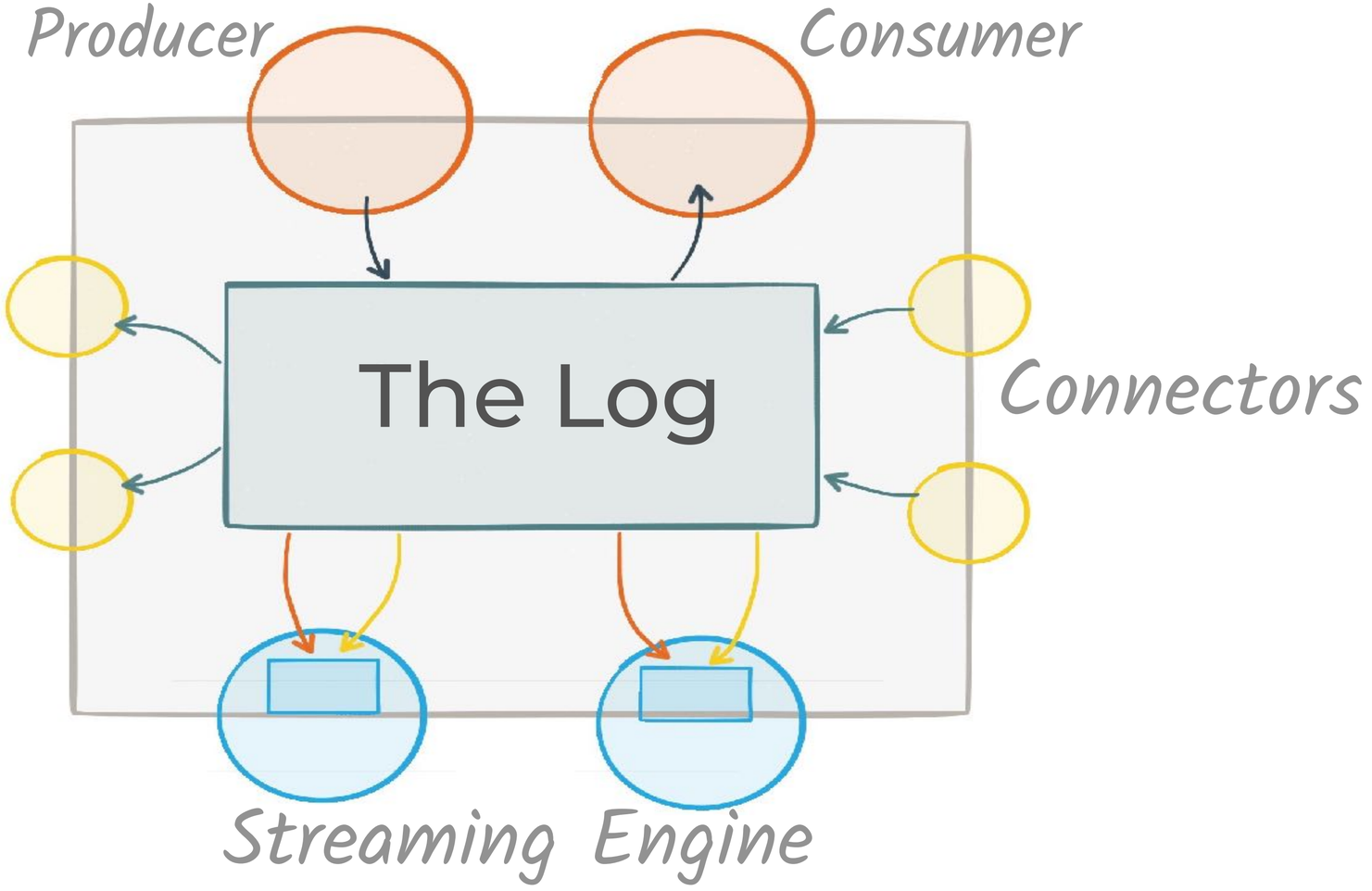

Producer
Consumer
The Log


Producer
Consumer
The Log
Connectors

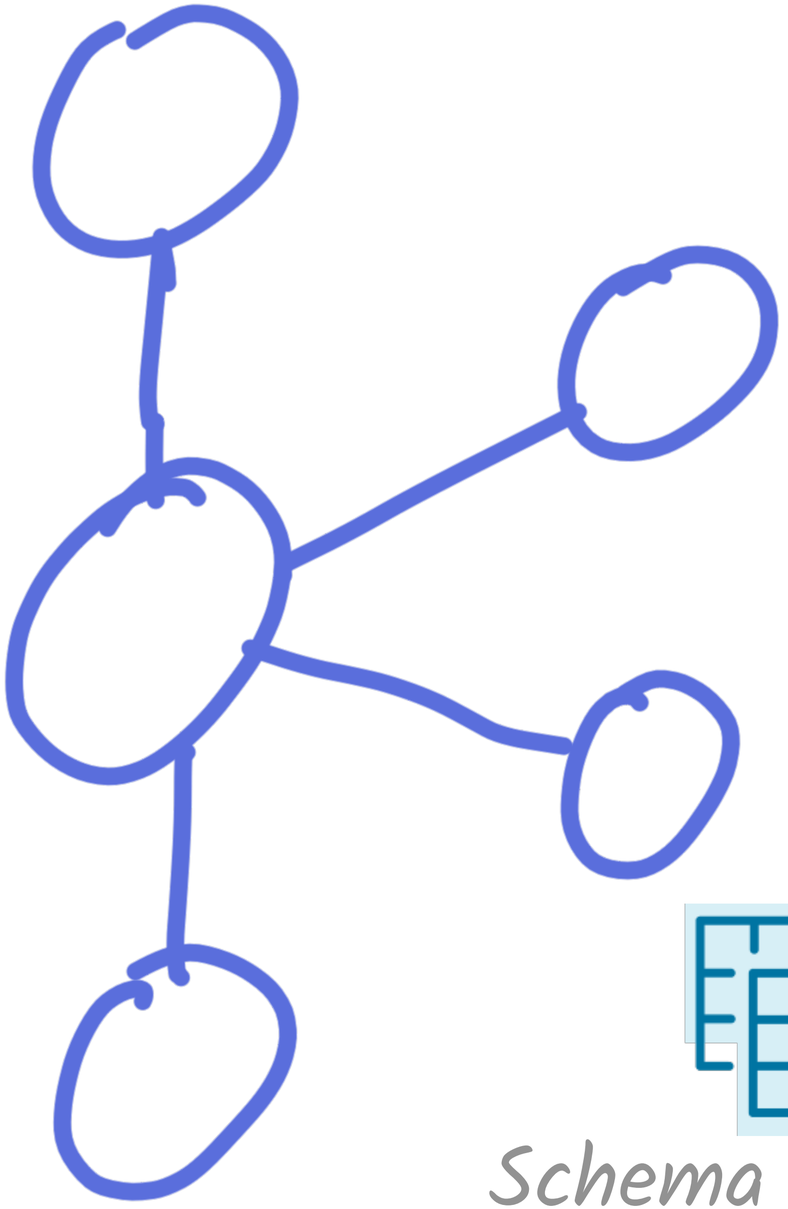

Producer
Consumer
The Log
Connectors
Streaming Engine

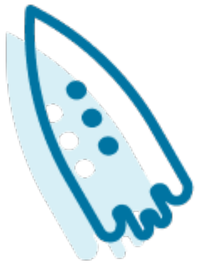
Apache Kafka



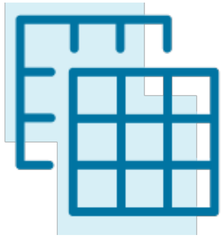


Confluent Platform

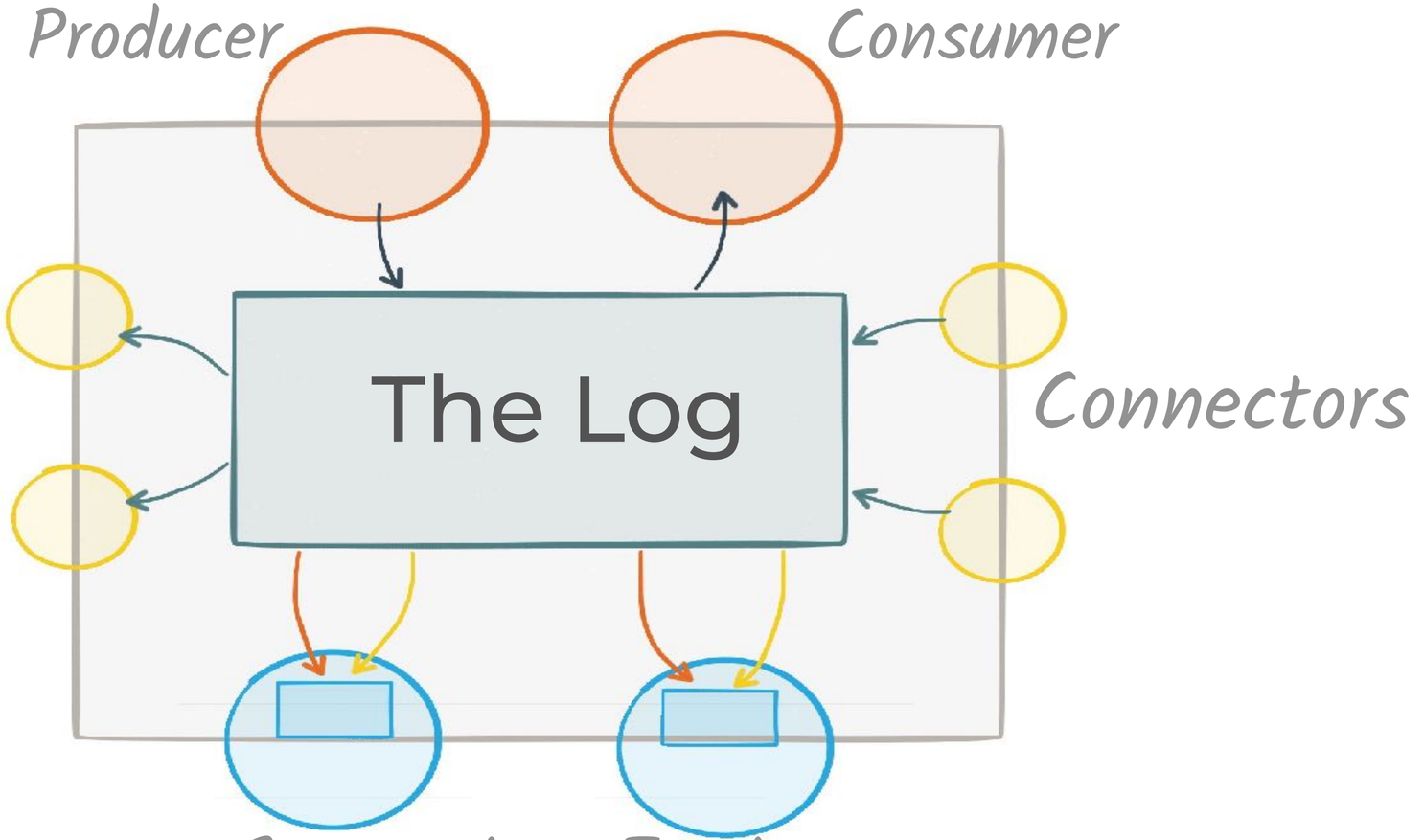




ksqlDB

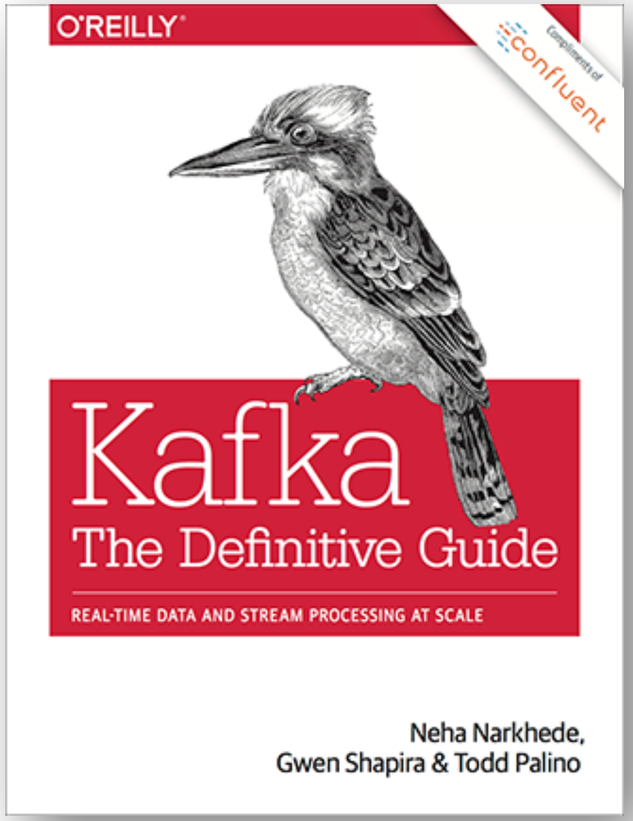


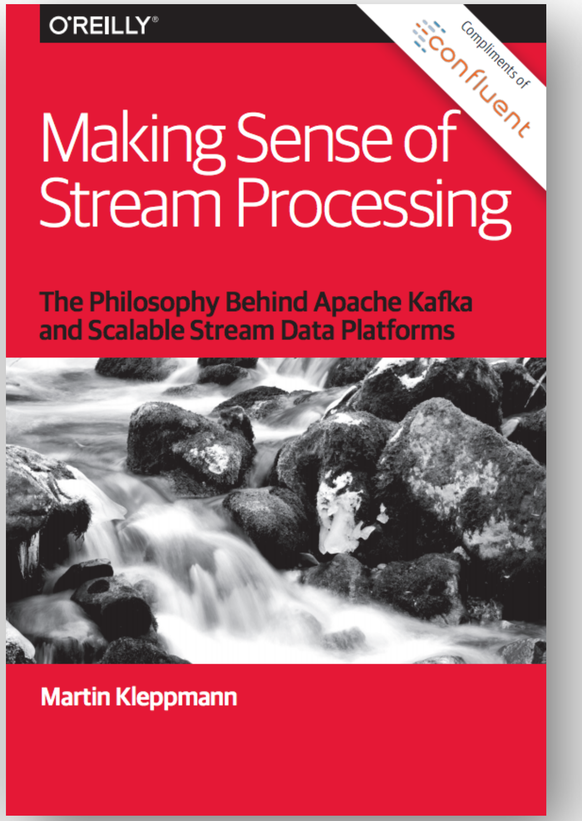
Schema Registry

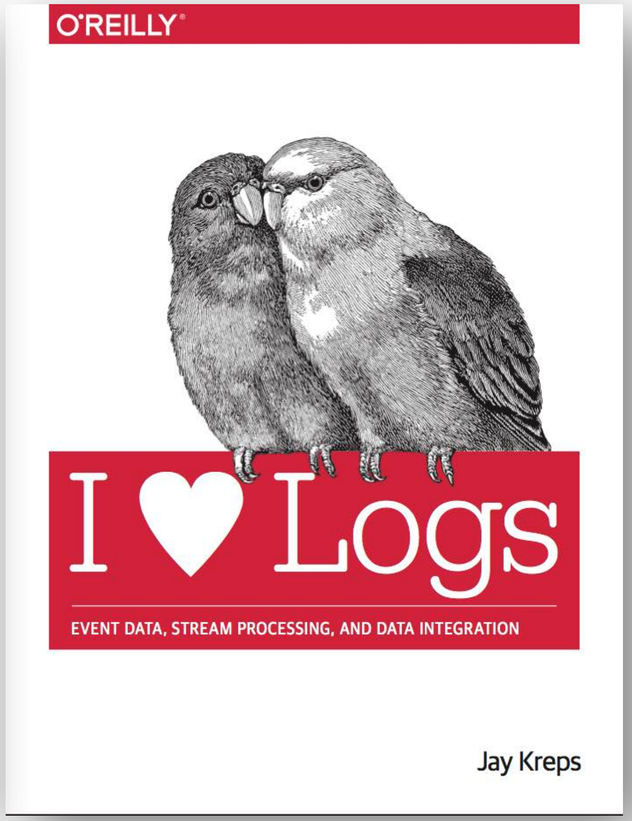


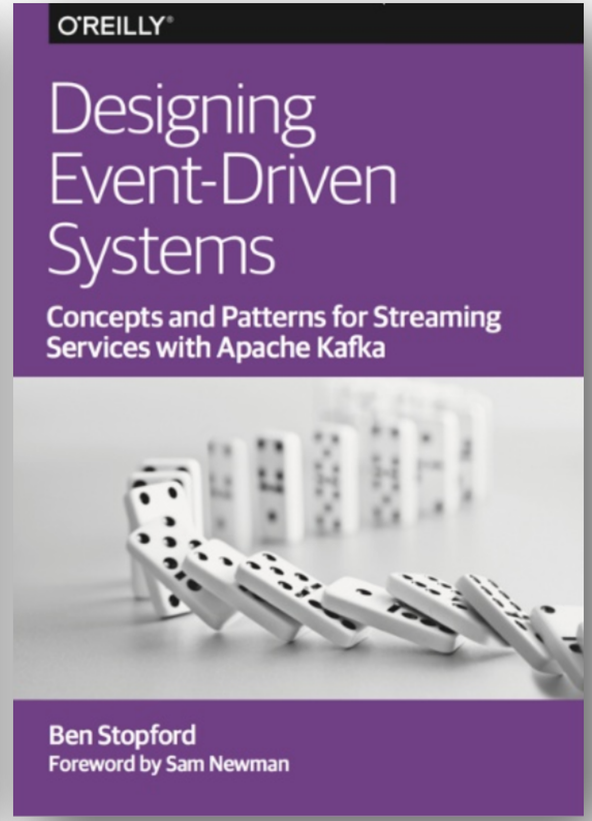
Free Books!

https://rmoff.dev/gotopia









confluent cloud

Fully Managed Kafka as a Service


Learn Kafka.
Start building with
Apache Kafka at
Confluent Developer.
developer.confluent.io

#EOF

@rmoff rmoff.dev/talks youtube.com/rmoff