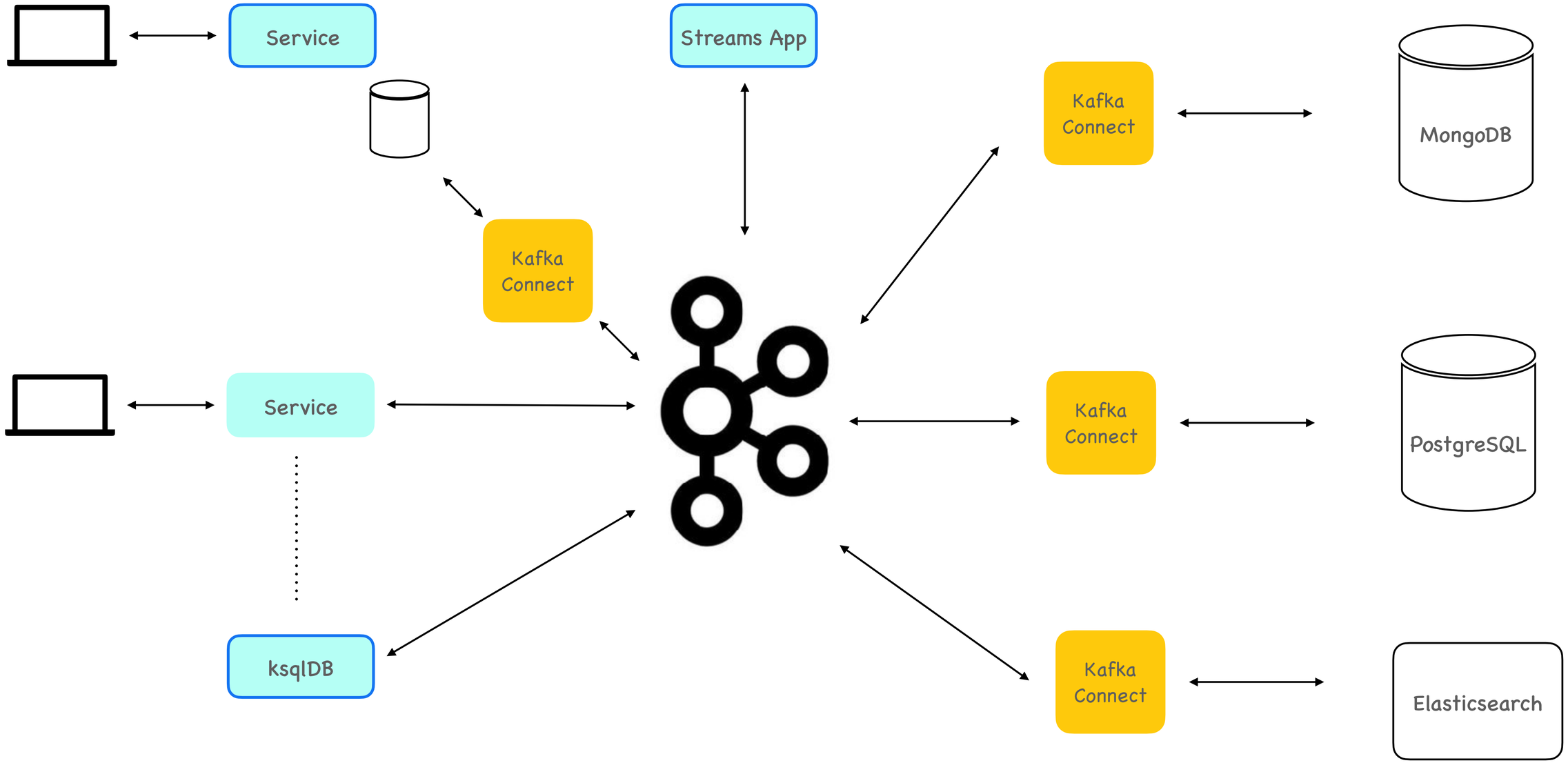
Event Streaming With Apache Kafka

dklein@confluent.io @daveklein linkedin.com/in/daveklein19

What is Apache Kafka?

Event Streaming Platform



Event

Notification

Order Placed

Temperature Read

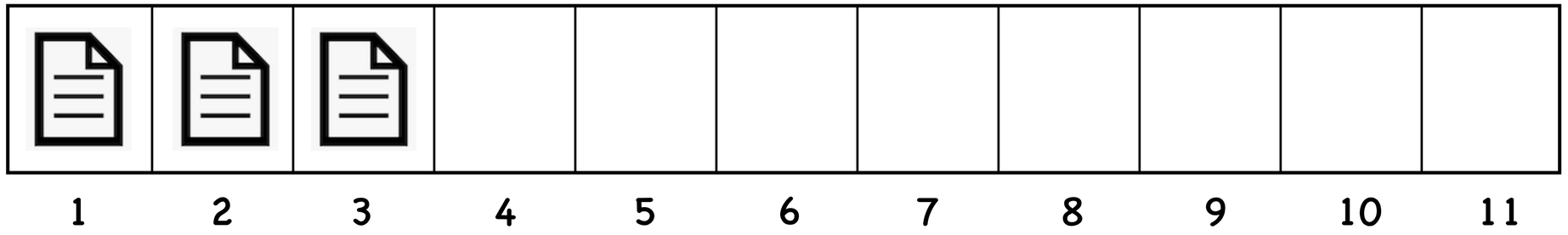
State

{item:123, price: 29.95,
qty: 2}

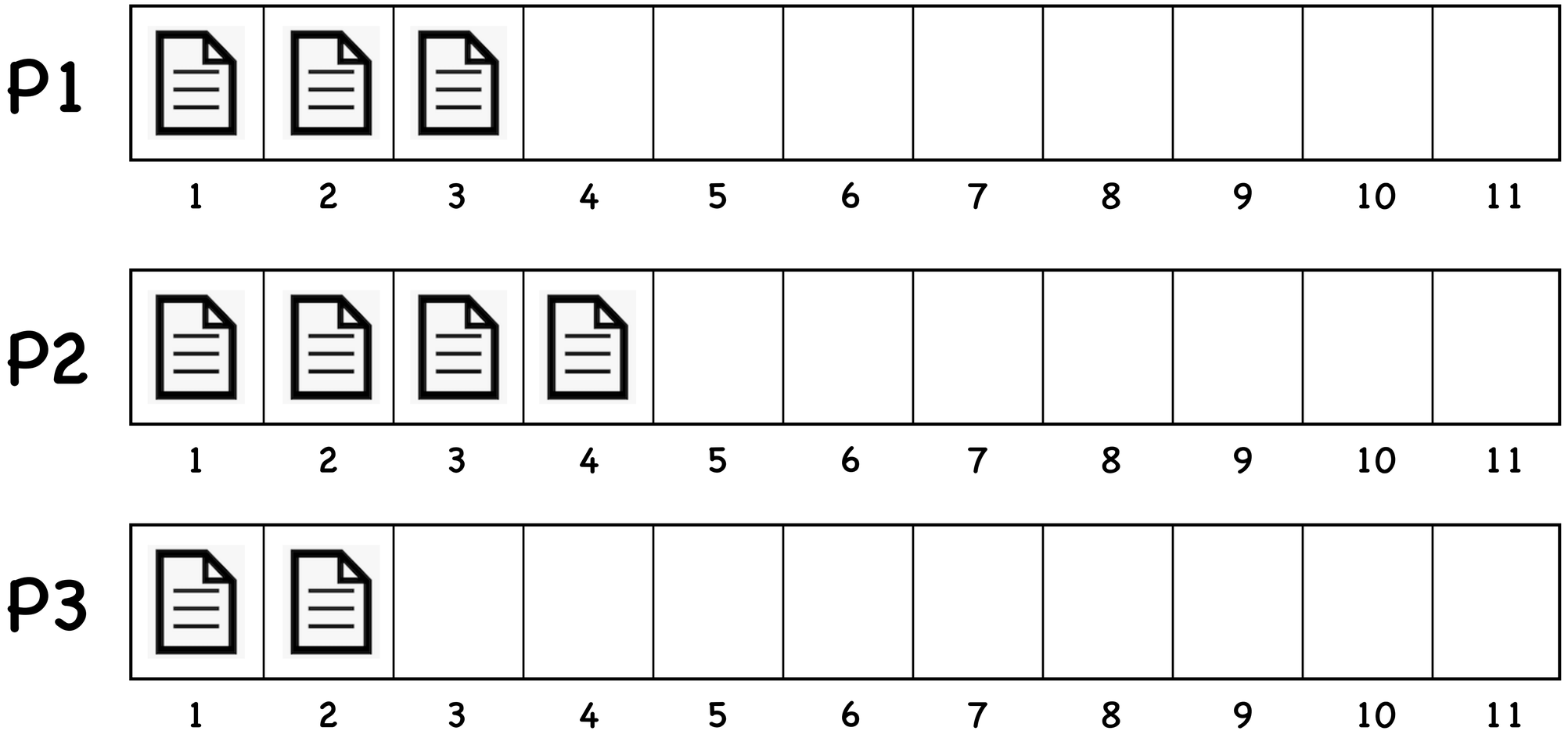
{temp:29, unit: F,
time: 1606949836369}

```
"key": "a1",
"value": {
    "eventType": “added-to-cart",
    "title": "Kafka Streams in Action”,
    "author": "Bill Bejeck",
    "price": 44.99
}
```

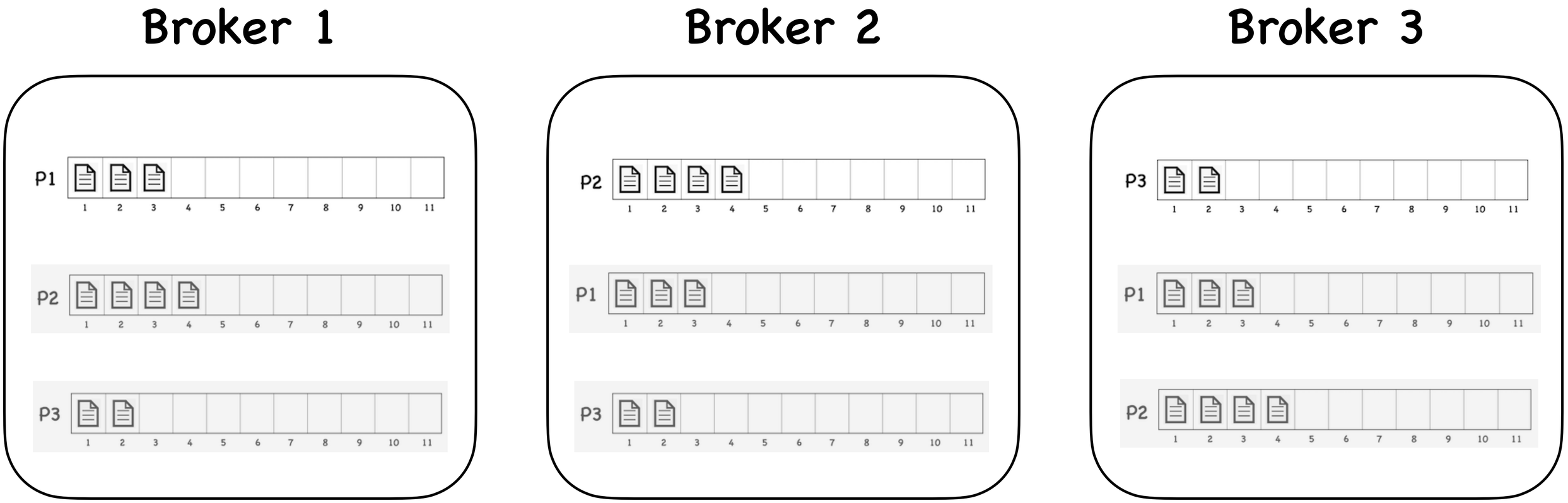
Topic (log)

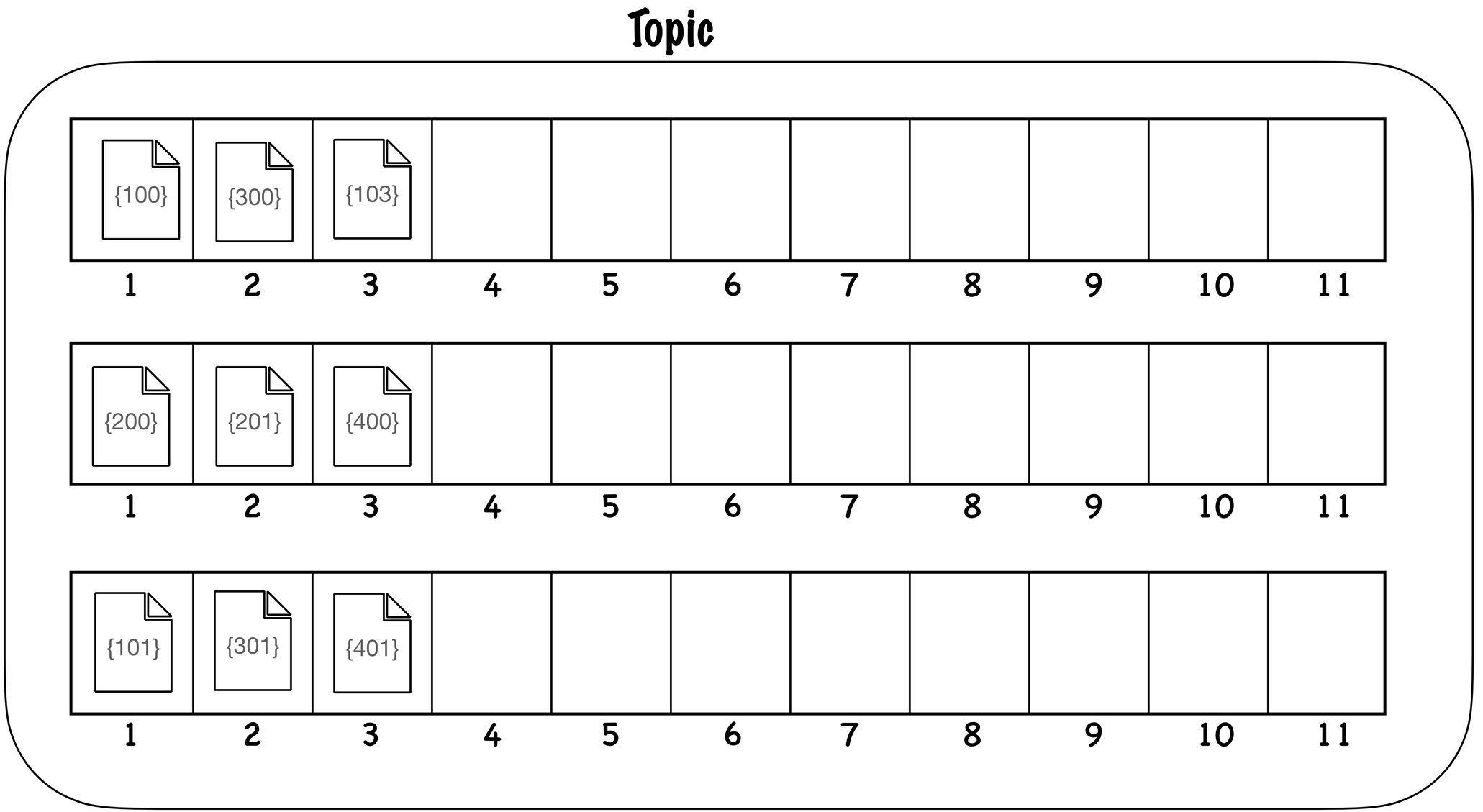


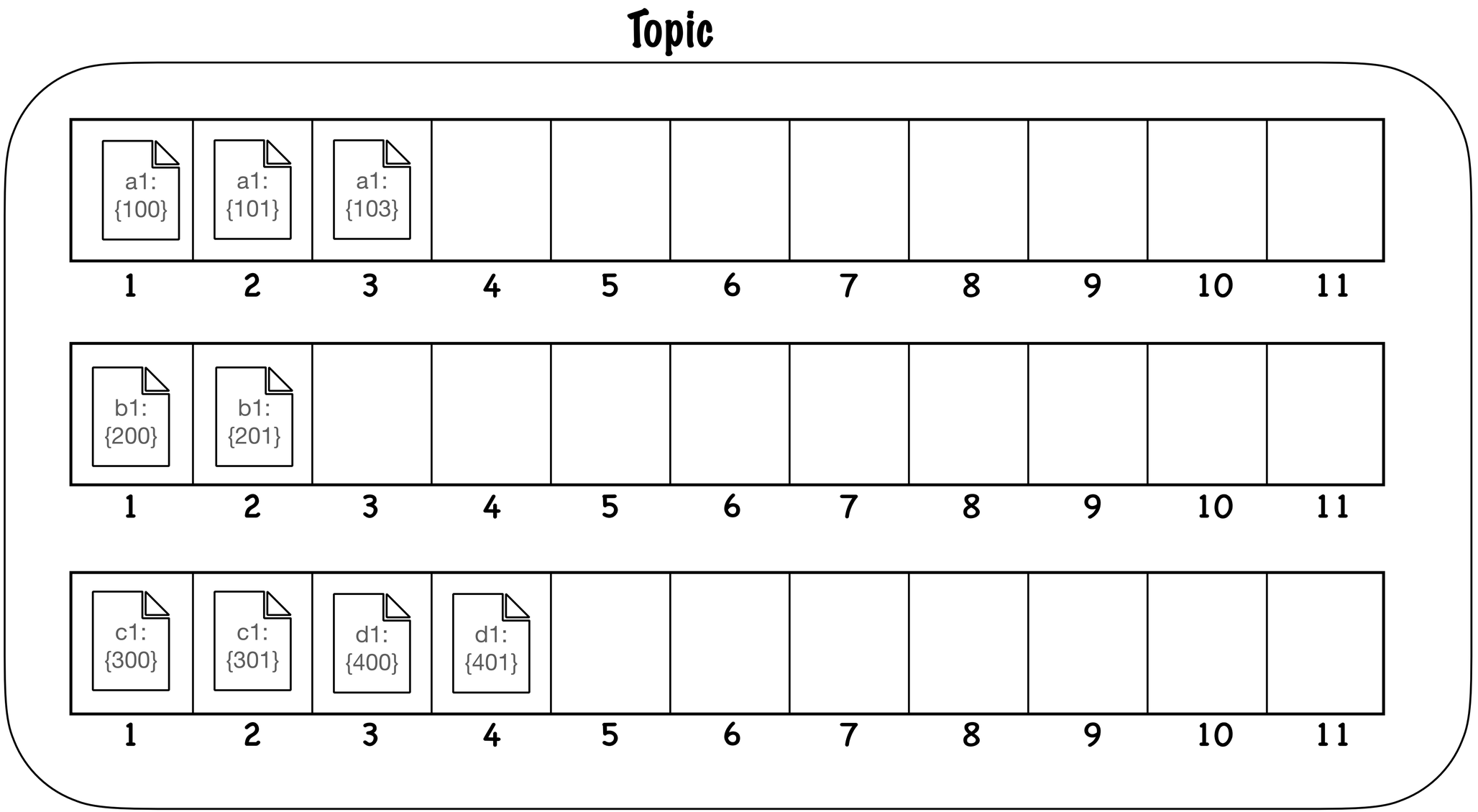
Topic Partitions

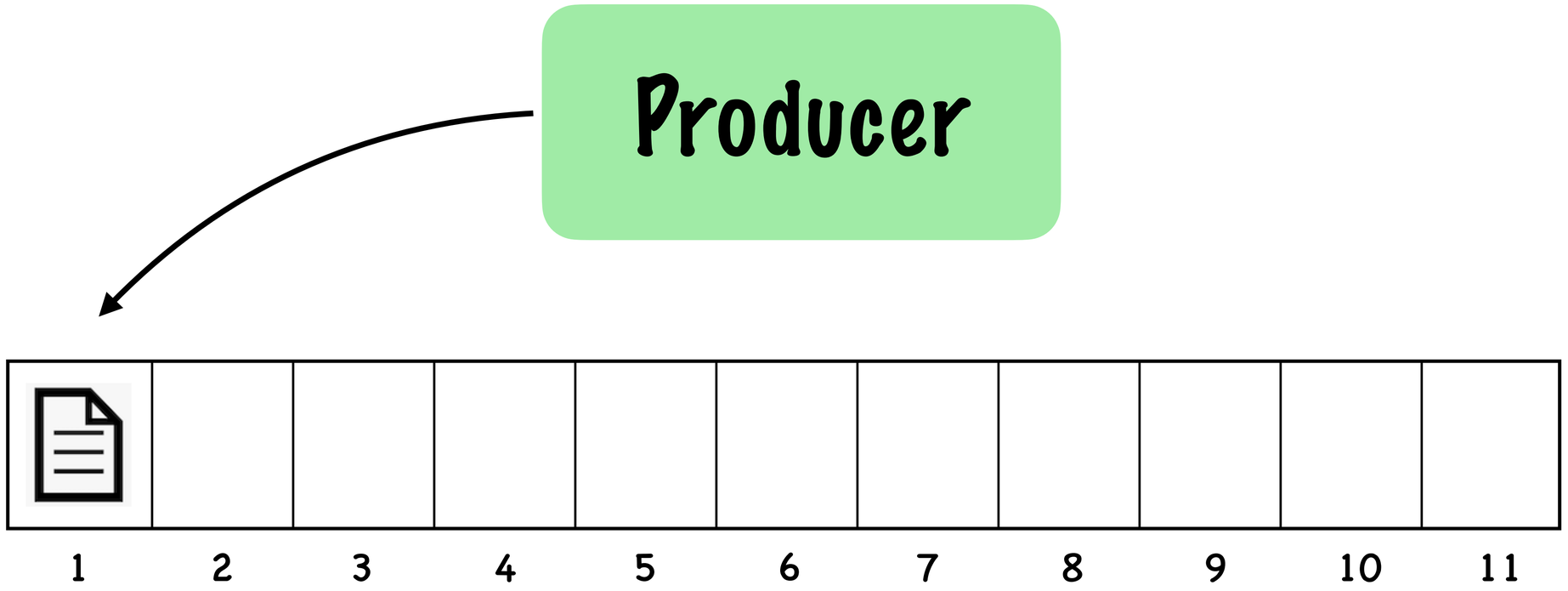


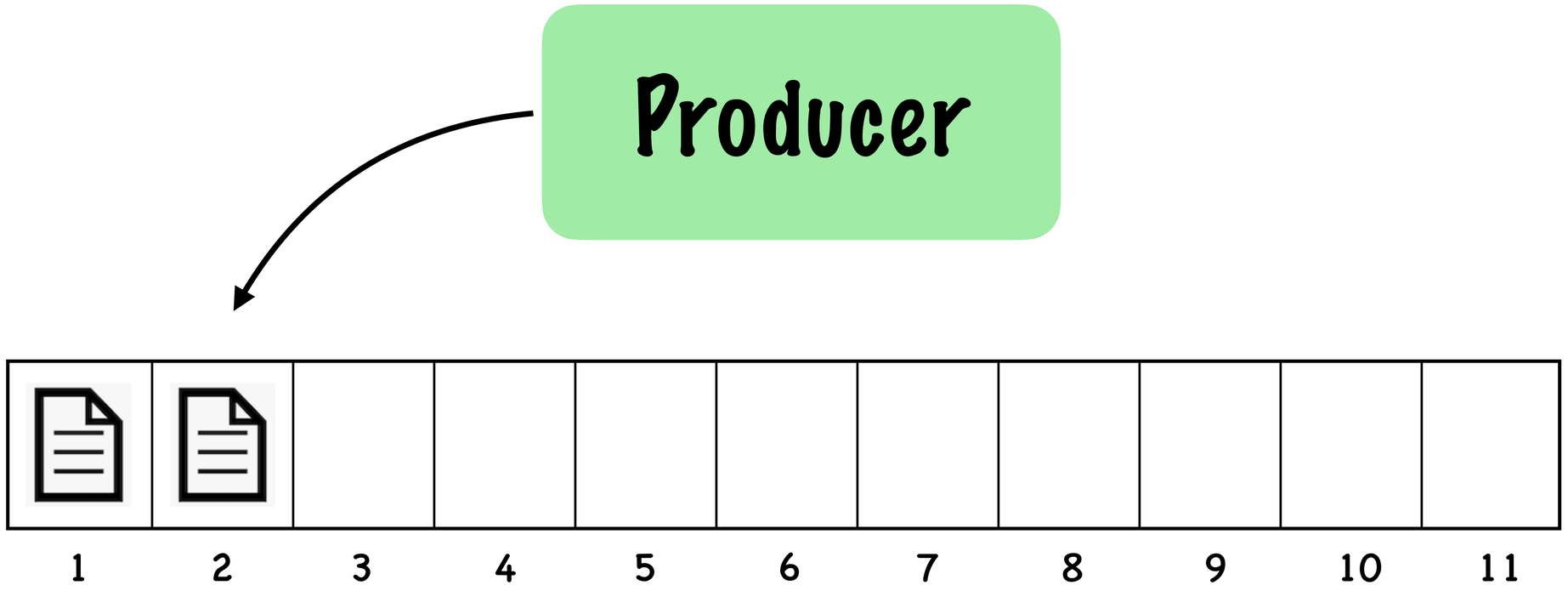
Brokers and Replication

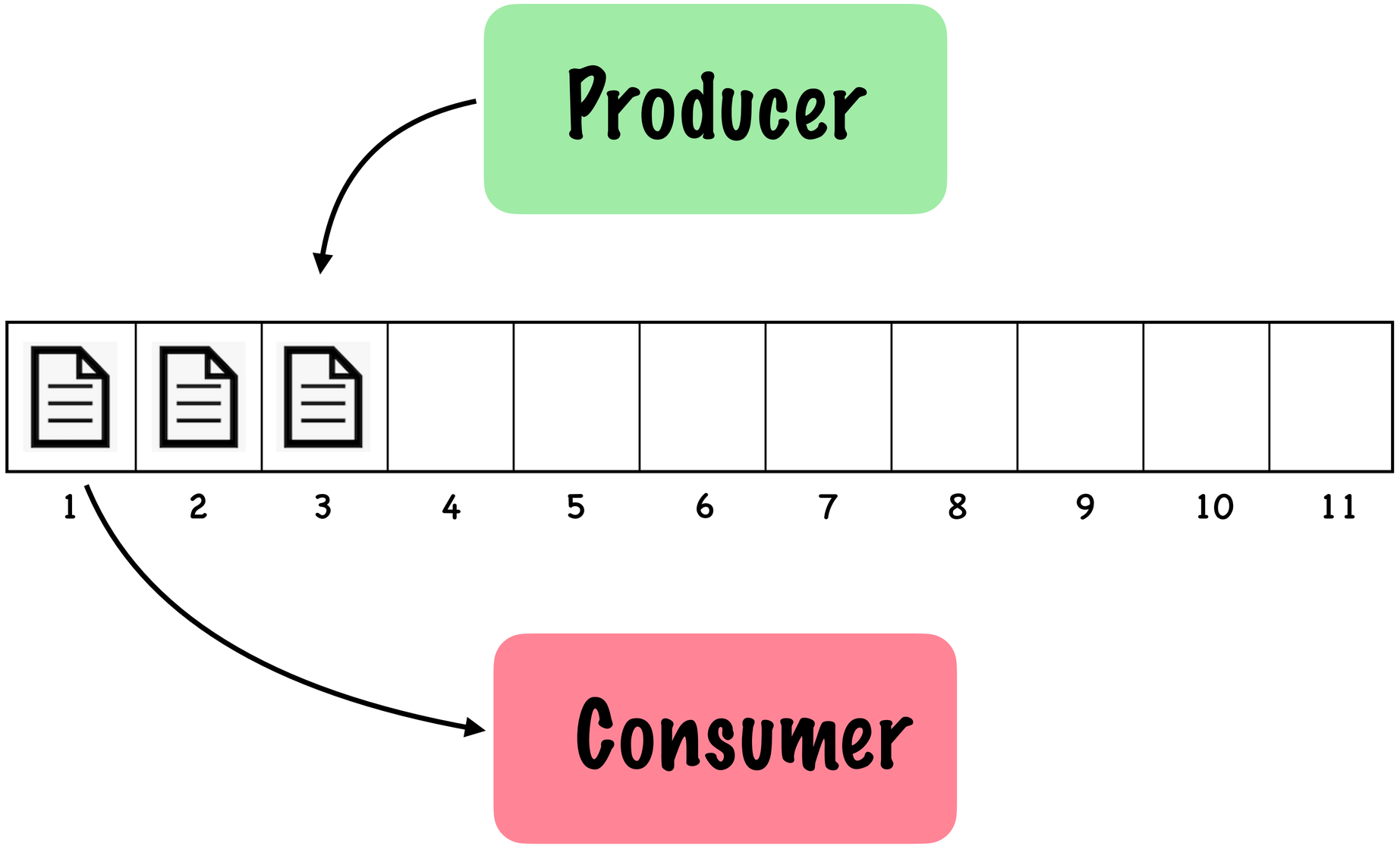


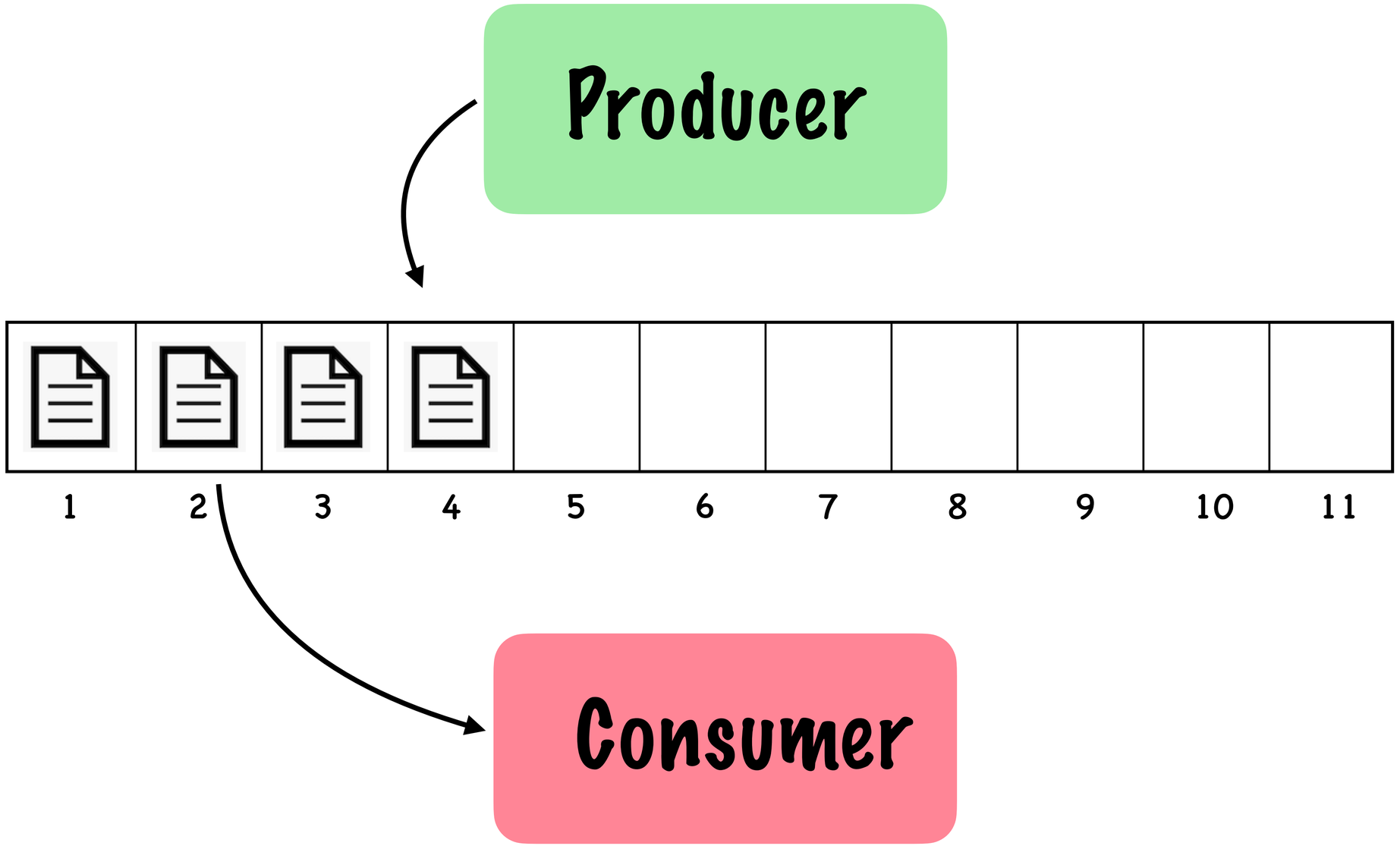

Topic
{100}
{300}
{103}
1
2
3
4
5
6
7
8
9
10
11
{200}
{201}
{400}
1
2
3
4
5
6
7
8
9
10
11
{101}
{301}
{401}
1
2
3
4
5
6
7
8
9
10
11

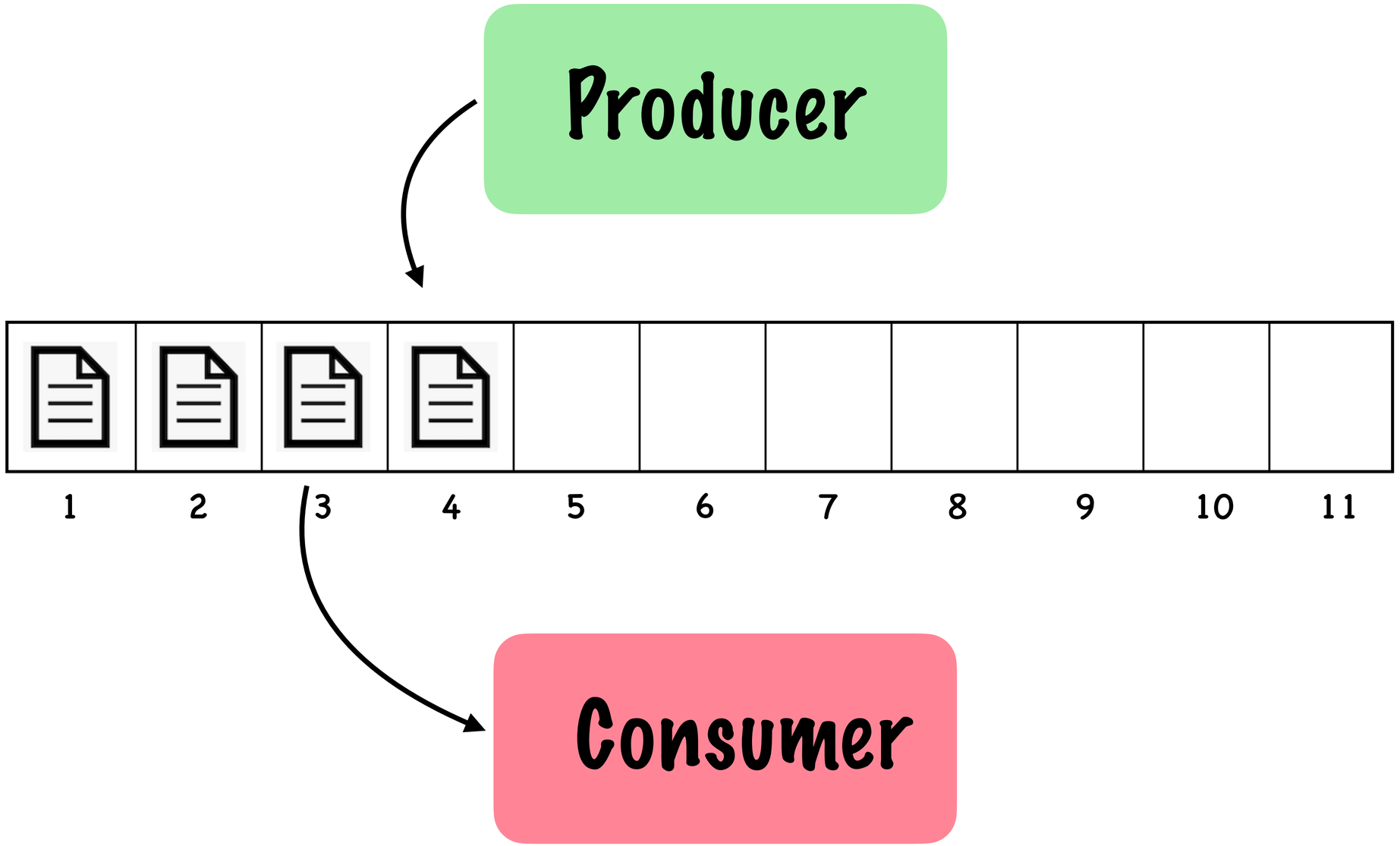

Topic
a1: {100}
a1: {101}
a1: {103}
1 2 3 4 5 6 7 8 9 10 11
b1: {200}
b1: {201}
1 2 3 4 5 6 7 8 9 10 11
c1: {300}
c1: {301}
d1: {400}
d1: {401}
1 2 3 4 5 6 7 8 9 10 11

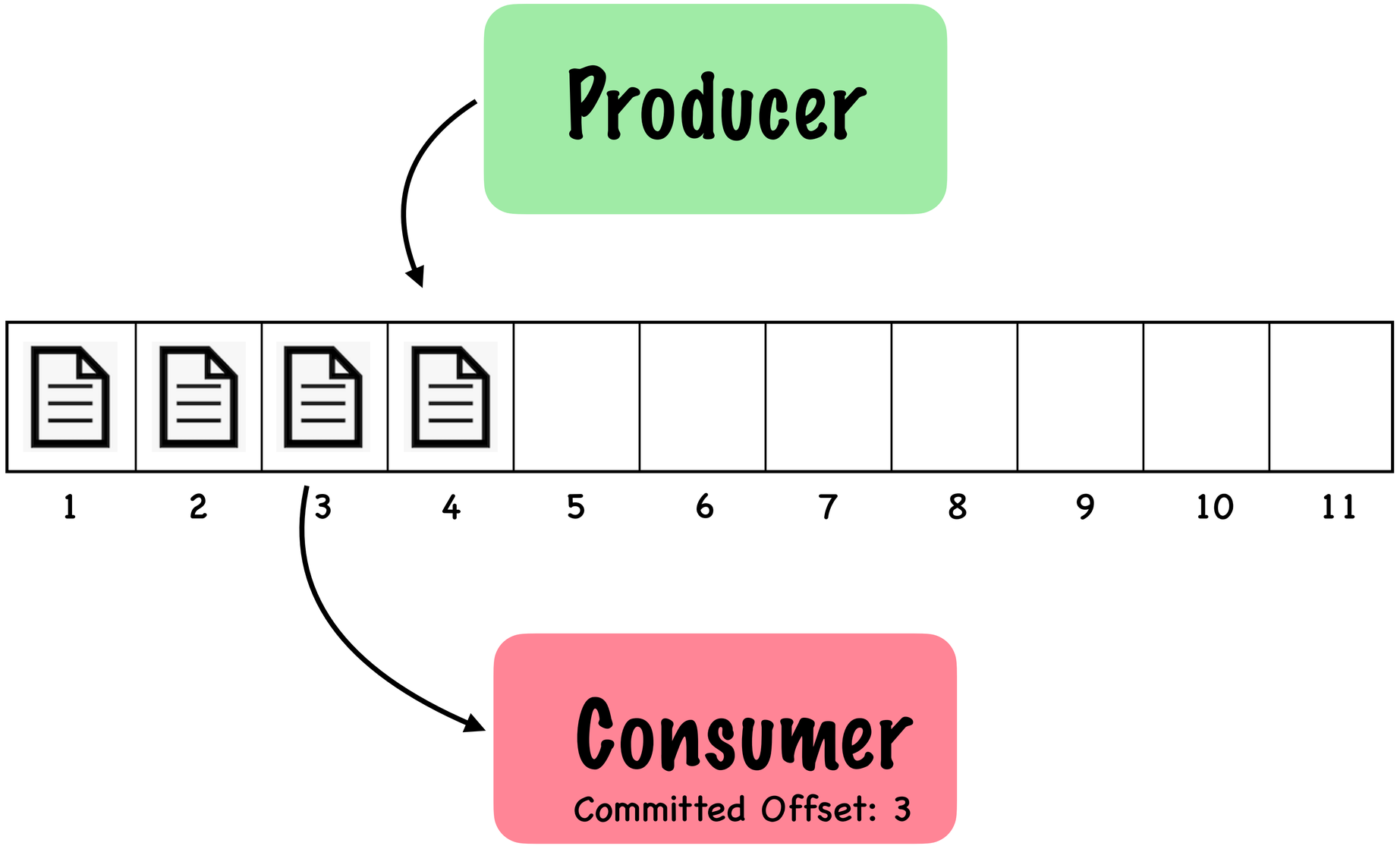

Producer
1
2
3
4
5
6
7
8
9
10
11

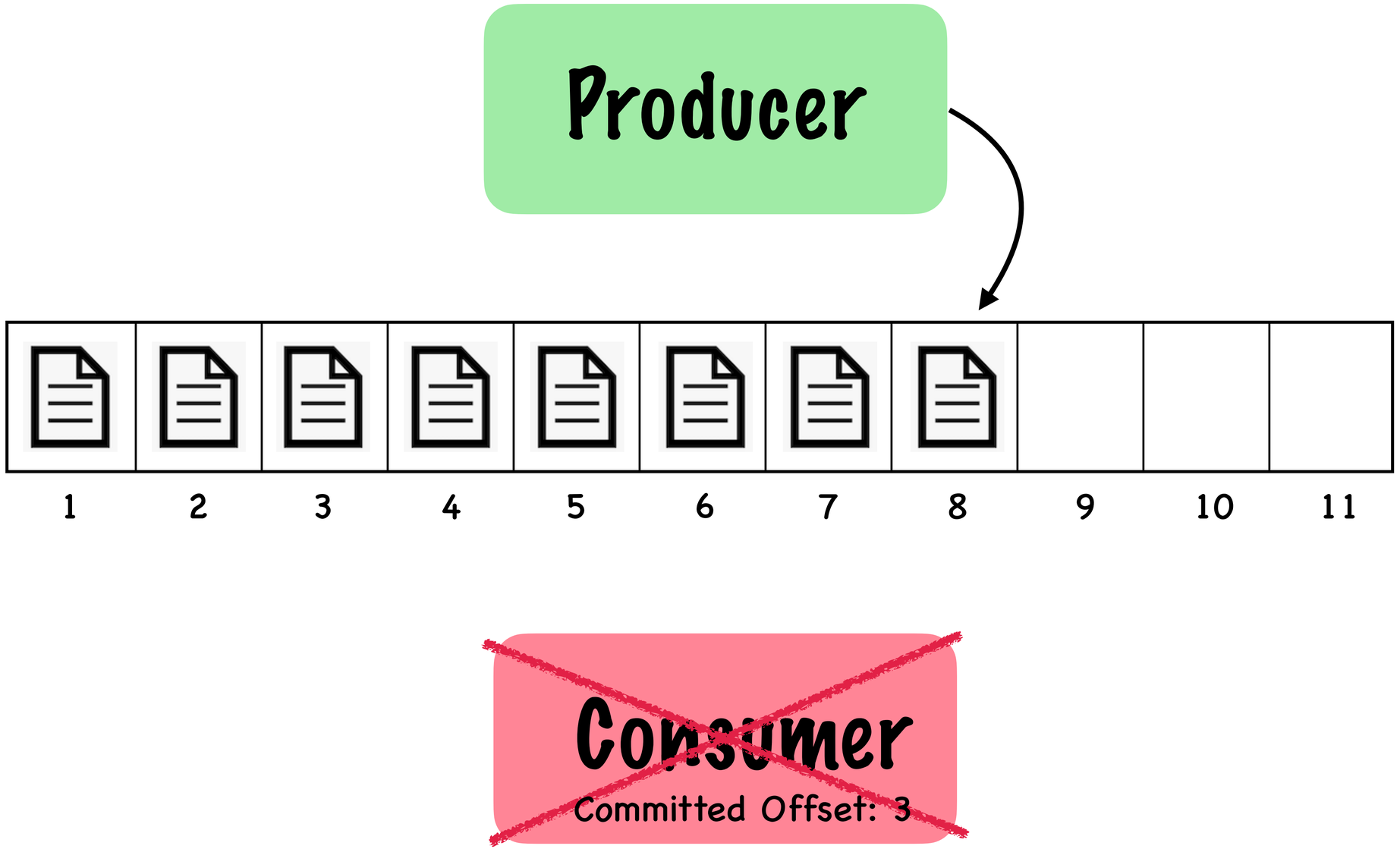

Producer
1
2
3
4
5
6
7
8
9
10
11

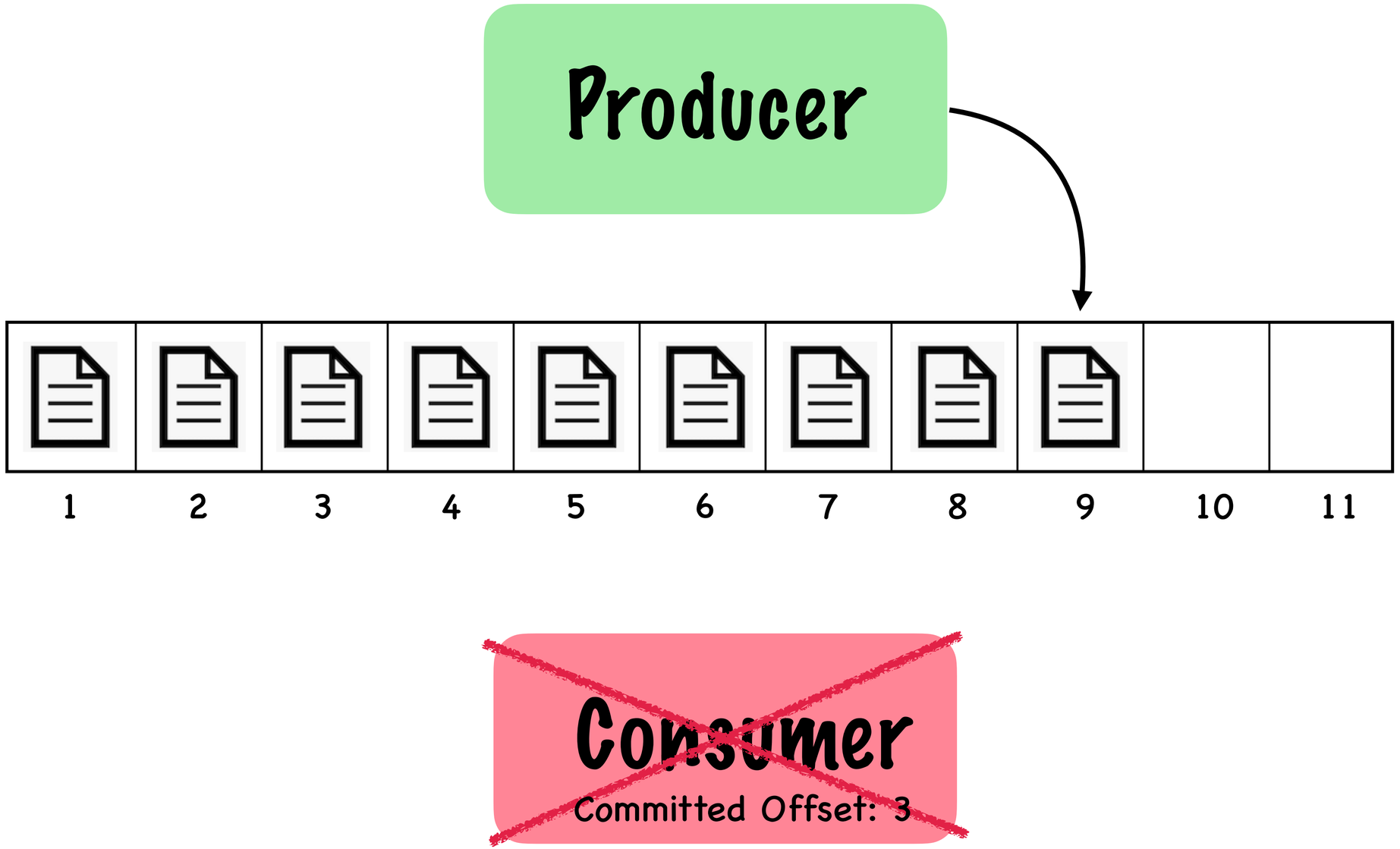

Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer

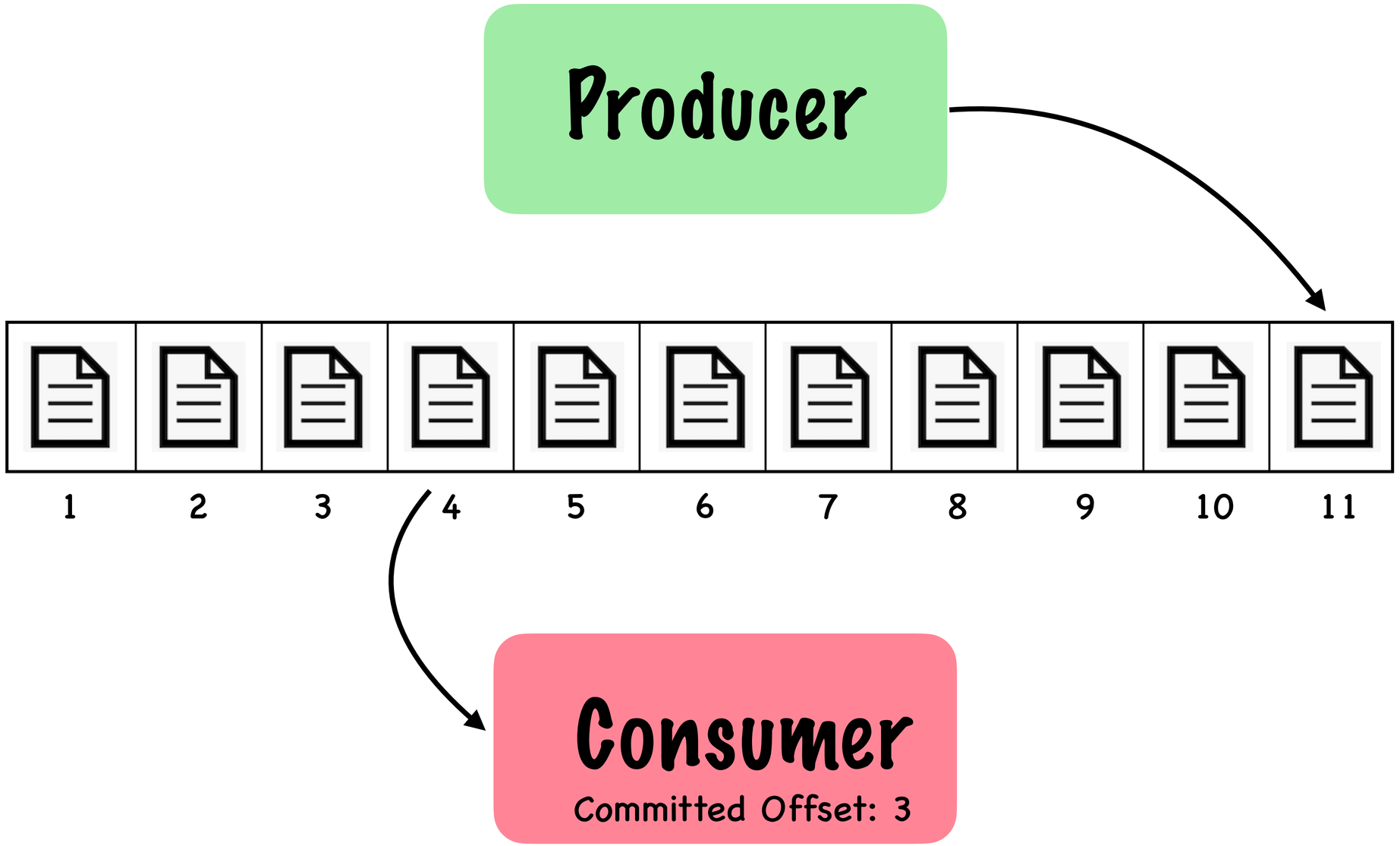

Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer


Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer


Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer
Committed Offset: 3


Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer
Committed Offset: 3


Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer
Committed Offset: 3

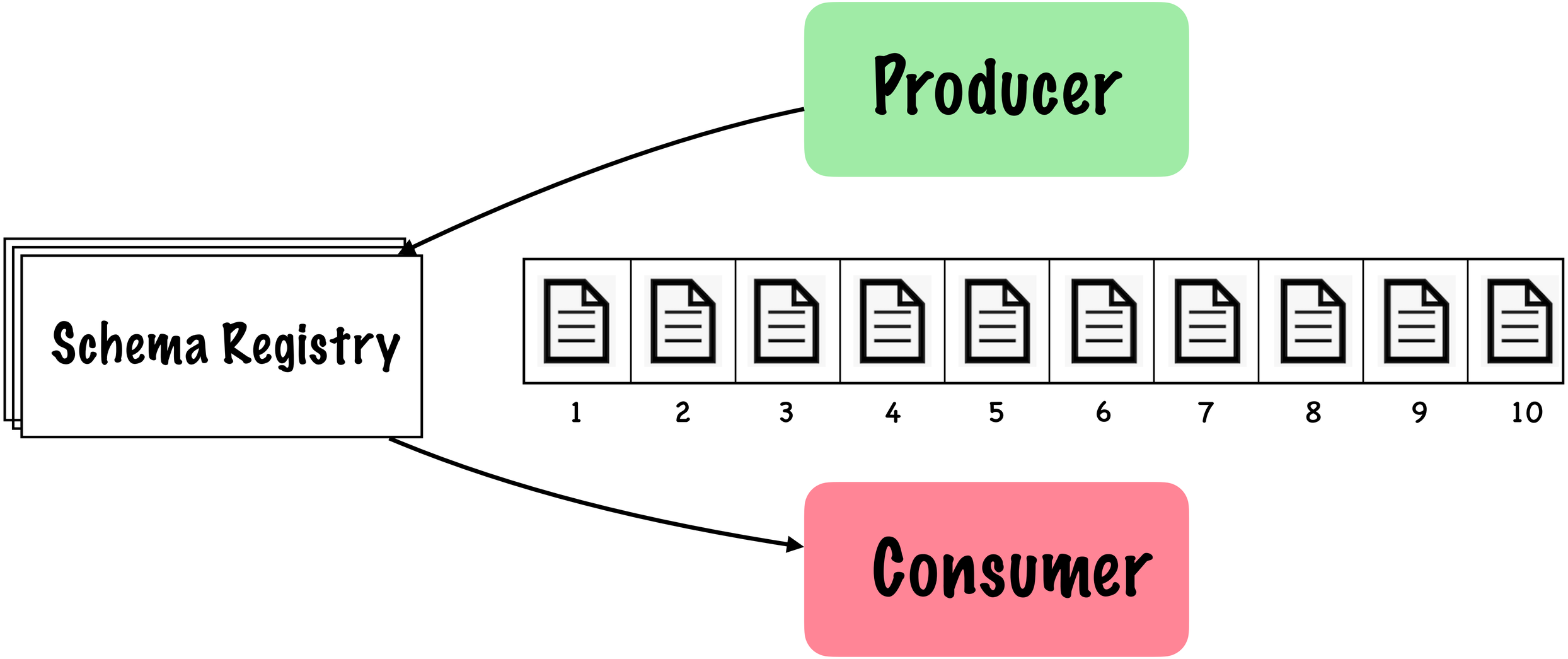

Producer
1
2
3
4
5
6
7
8
9
10
11
Consumer
Committed Offset: 3

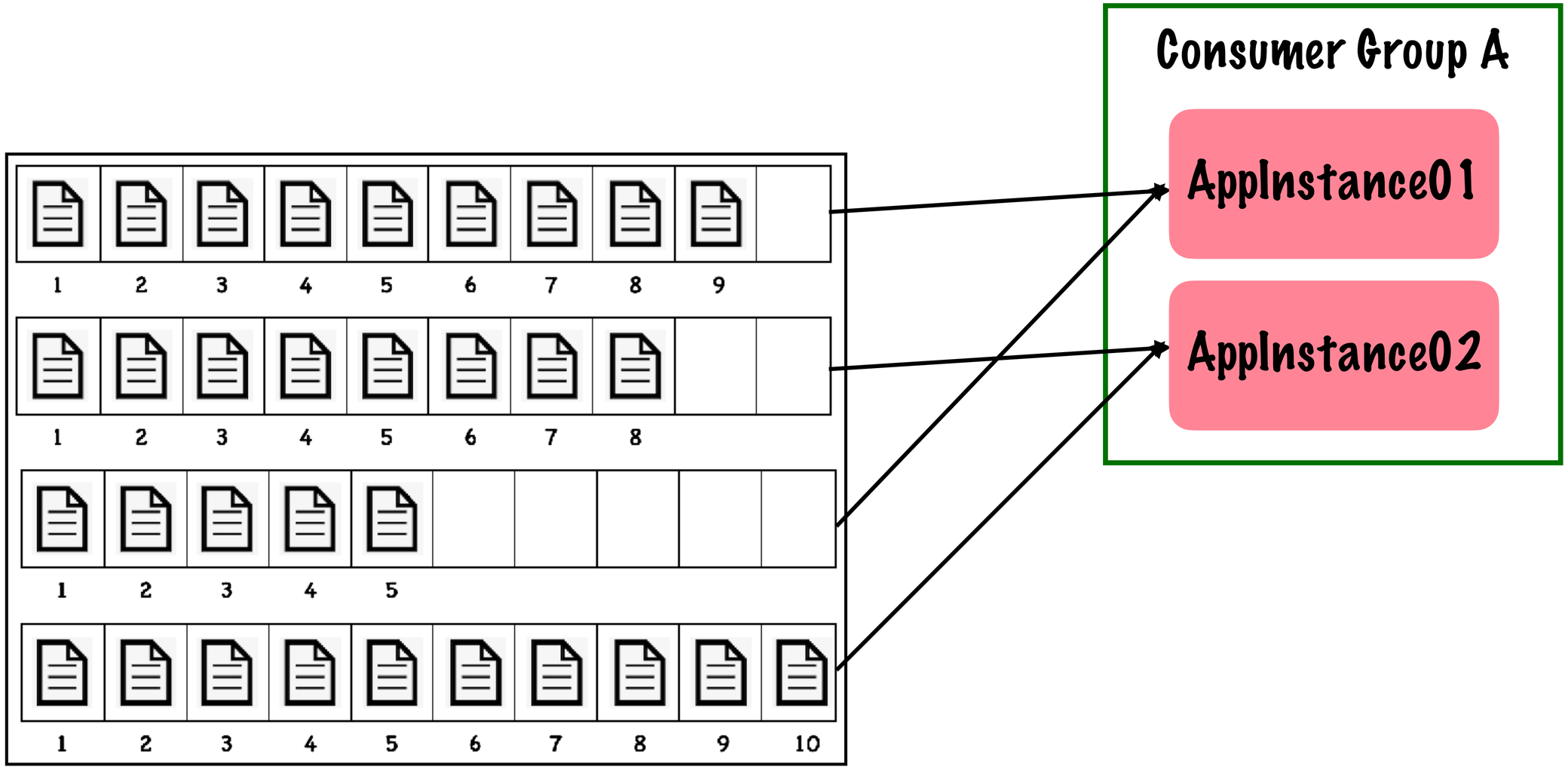
Sample Kafka Producer

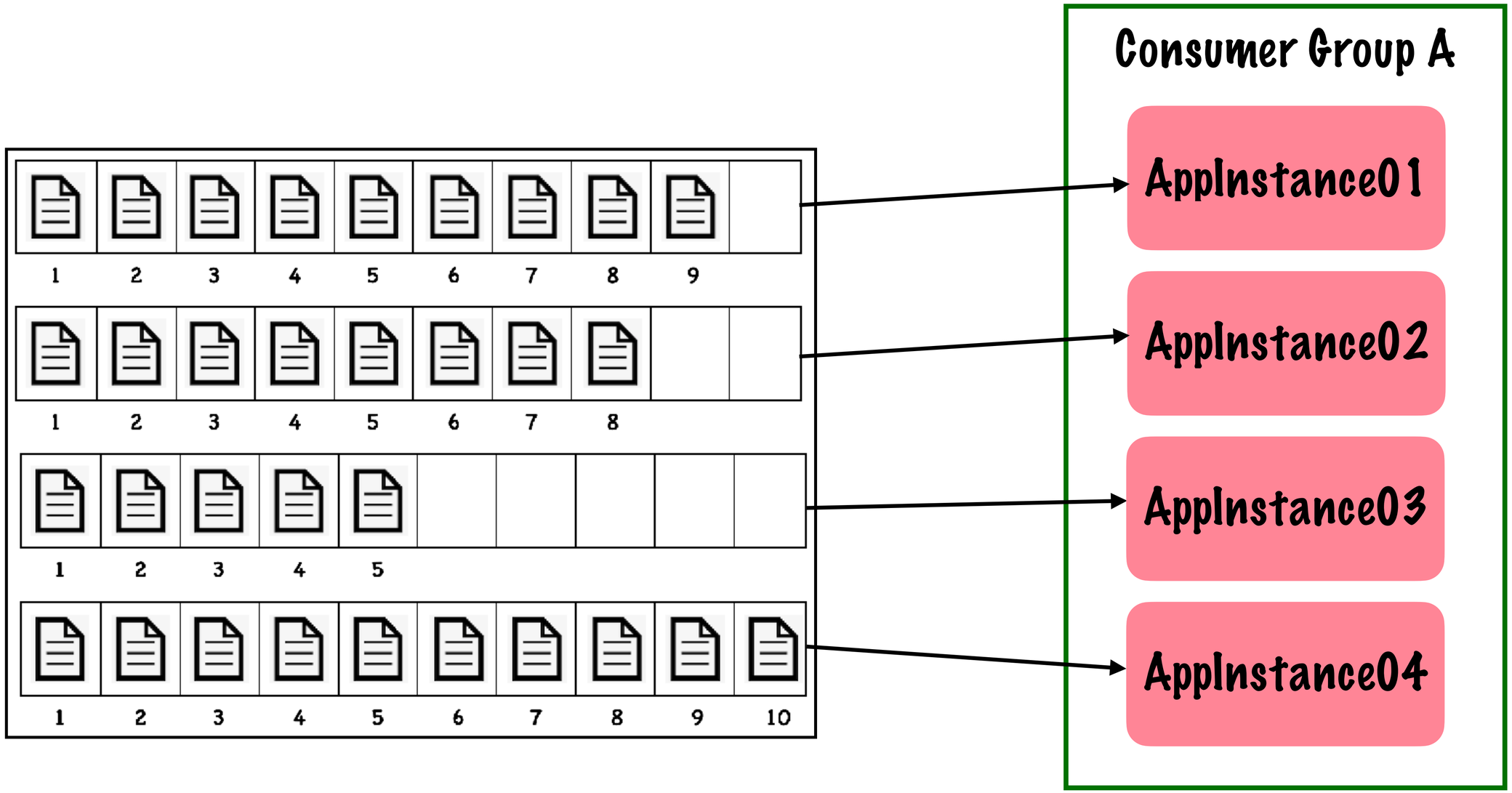
```
Producer<String, String> producer = new KafkaProducer<String, String>(props);

String key = "acme";
String value = "some info about acme, probably in JSON”;
ProducerRecord record = new ProducerRecord<String, String>(topic, key, value);

producer.send(record, new Callback(){
    @Override
    public void onCompletion(RecordMetadata m, Exception e) {
        if (e != null) {
            e.printStackTrace();
        } else {
            System.out.printf("Produced record to topic %s partition [%d]",
                m.topic(), m.partition());
        }
    }
});
```

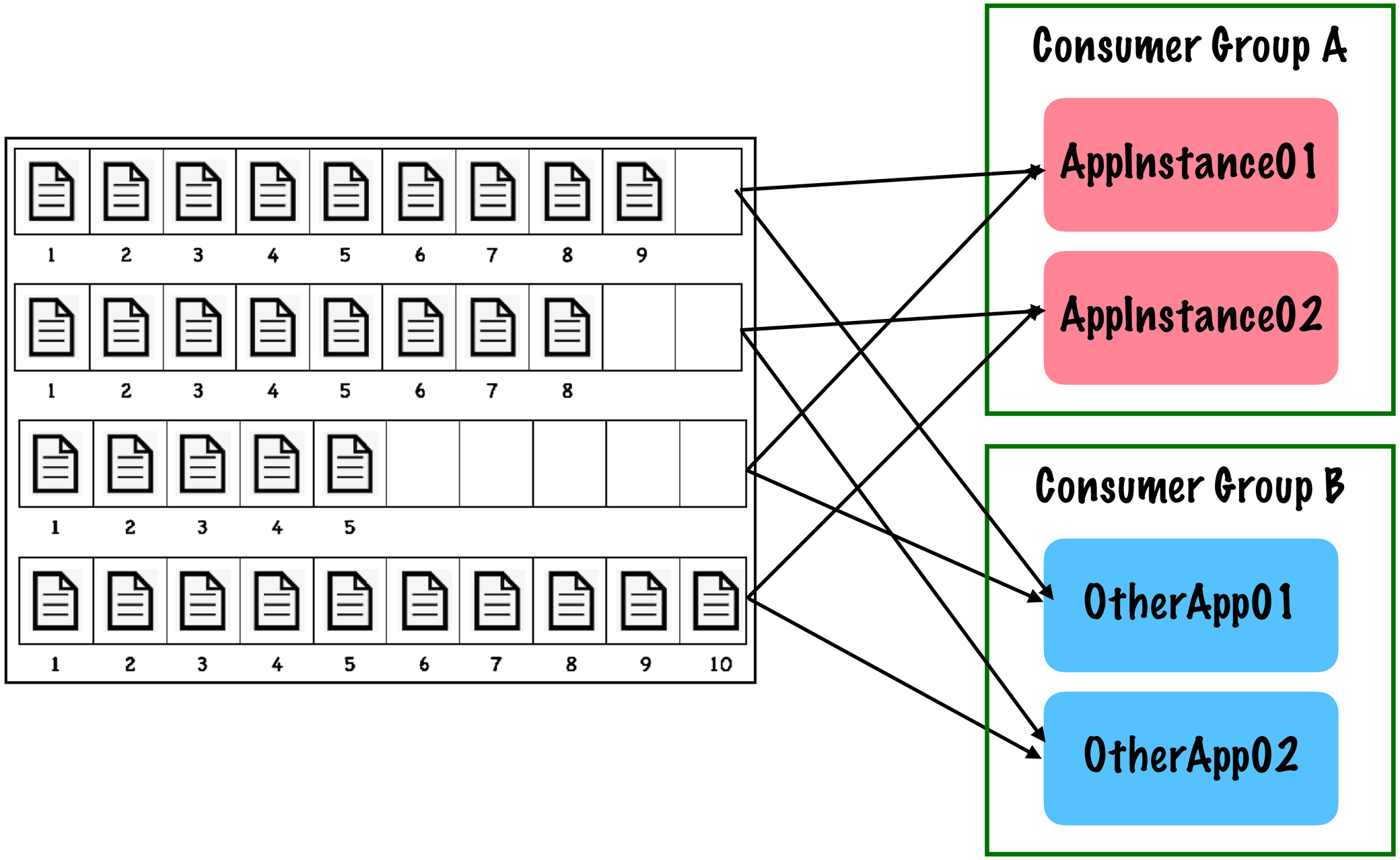
Sample Kafka Consumer

```
Consumer<String, String> consumer = new KafkaConsumer<String, String>(props);
consumer.subscribe(Arrays.asList(topic));

try {
  while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records) {
      String key = record.key();
      String value = record.value();
      System.out.printf("Consumed record - key: %s and value: %s", key, value);
    }
  }
} finally {
  consumer.close();
}
```


Producer
Schema Registry
1
2
3
4
5
6
7
8
9
10
Consumer


Consumer Group A
AppInstance01
AppInstance02
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8
1 2 3 4 5
1 2 3 4 5 6 7 8 9 10


Consumer Group A
AppInstance01
AppInstance02
AppInstance03
AppInstance04
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8
1 2 3 4 5
1 2 3 4 5 6 7 8 9 10


Consumer Group A
AppInstance01
AppInstance02
Consumer Group B
OtherApp01
OtherApp02
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8
1 2 3 4 5
1 2 3 4 5 6 7 8 9 10

Exercises Part 1

https://github.com/daveklein/kafka-workshop

Kafka Streams

What is it?

What is it Good For?

Customer 360

Recommendation Engines

Factory Automation

Inventory Management

Fraud Detection

Customer Loyalty Programs

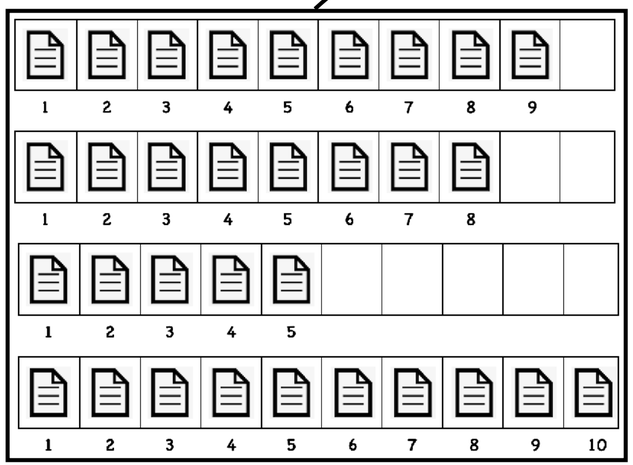
Fleet Management

Real-time Payments

Kafka Streams DSL

- Java Library (use with any JVM language)
- Define a processor topology (DAG) using a fluent interface
- Stateless operations (Branch, Filter, Map)
- Stateful operations (Count, Aggregate, Reduce)
- Manages state using RocksDB, backed by Kafka topics
- Supports Streams and Tables
- Join Stream -> Stream, Table -> Table, or Stream -> Table

Kafka Streams



Input Topic

Consumer

stream(input topic)

.filter()

.map()

.flatMap()

.branch()

.reduce()

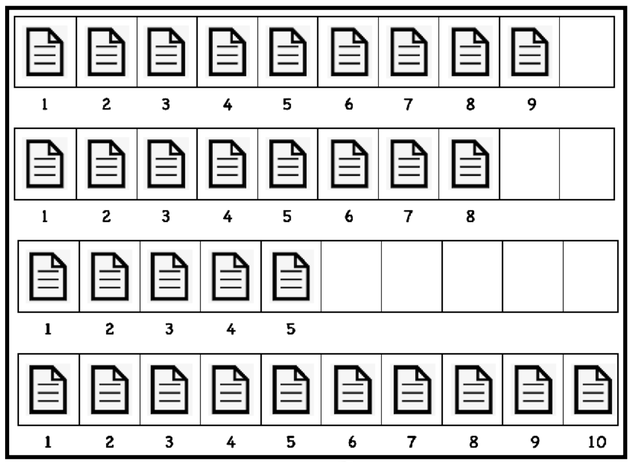
.groupByKey()

.peek()

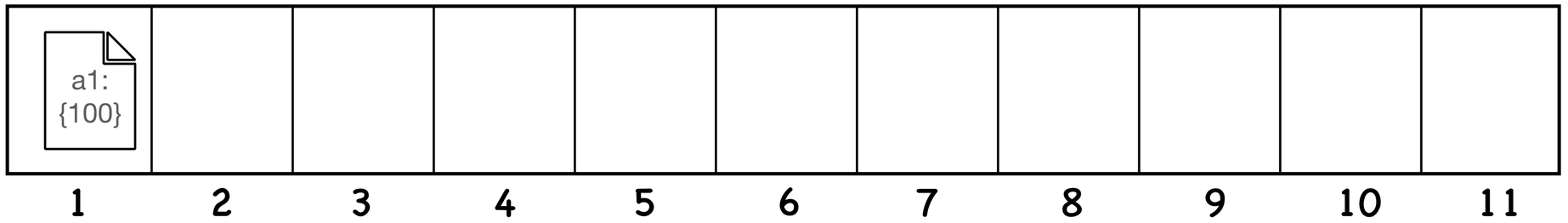
.to(output topic)

Producer

Output Topic



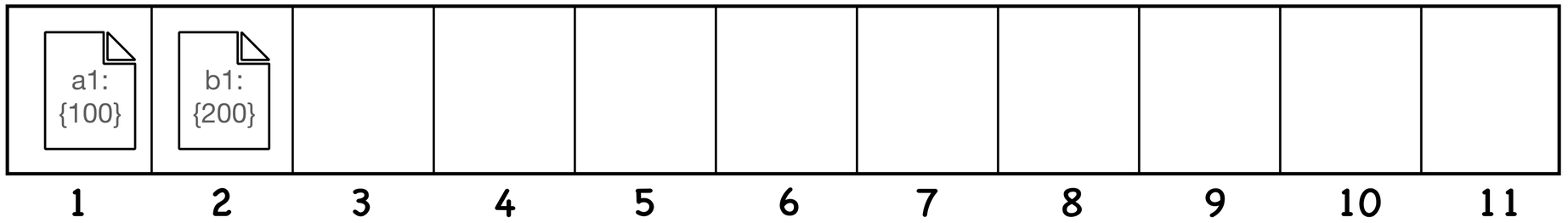
Stream



Table

a1	{100}

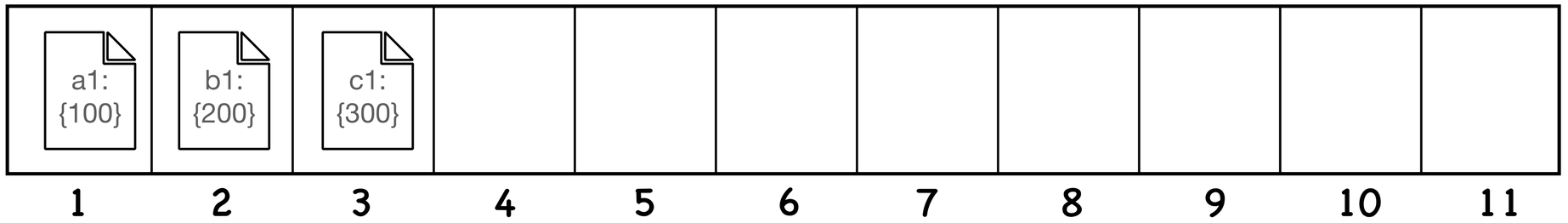
Stream



Table

a1	{100}
b1	{200}

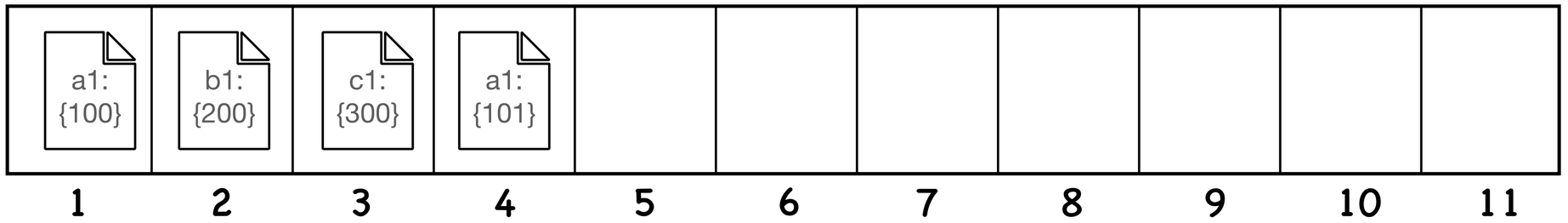
Stream



Table

a1	{100}
b1	{200}
c1	{300}

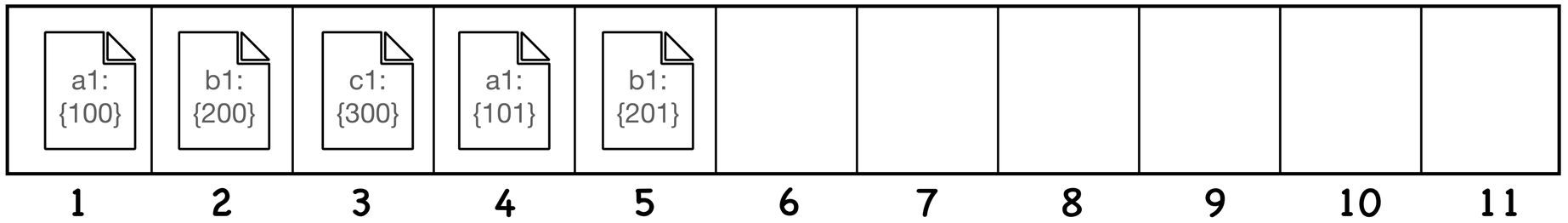
Stream



Table

a1	{101}
b1	{200}
c1	{300}

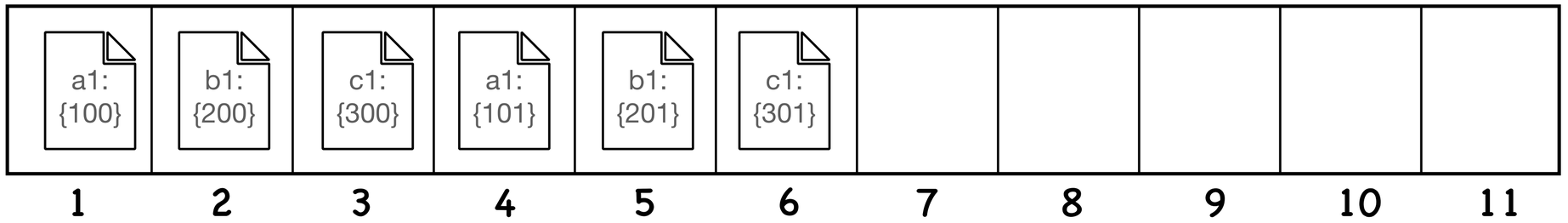
Stream



Table

a1	{101}
b1	{201}
c1	{300}

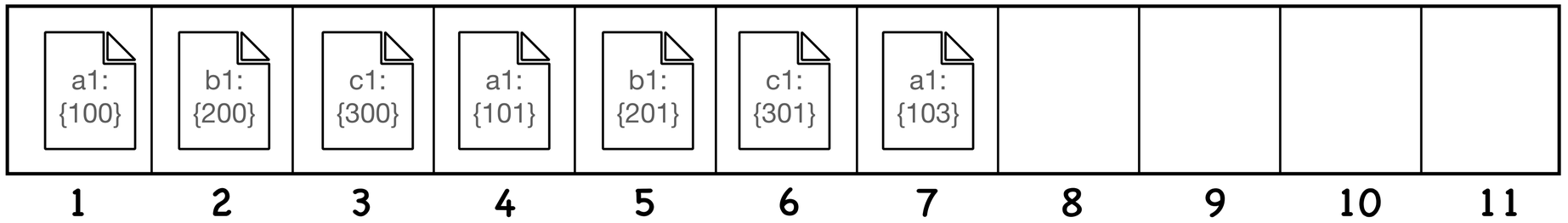
Stream



Table

a1	{101}
b1	{201}
c1	{301}

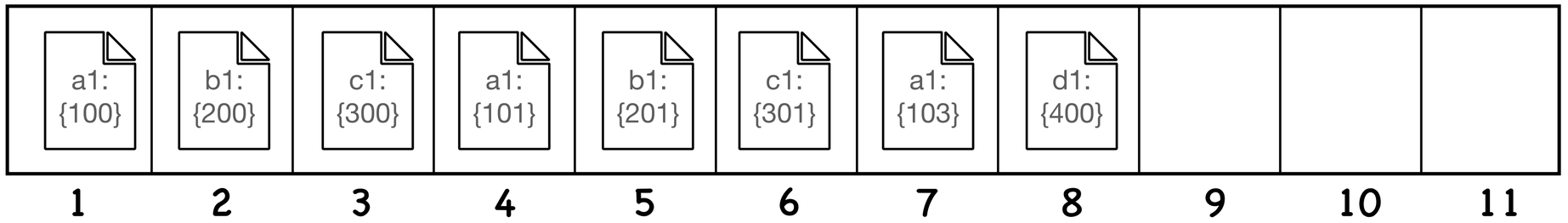
Stream



Table

a1	{103}
b1	{201}
c1	{301}

Stream



Table

a1	{103}
b1	{201}
c1	{301}
d1	{400}

Streams
record history

"The sequence of moves"

Tables
represent state

"The state of the board"

https://www.confluent.io/blog/kafka-streams-tables-part-1-event-streaming

Kafka Streams DSL Key Classes

- KafkaStreams - Encapsulates Producer & Consumer, Manages topology execution
- StreamsBuilder - Provides a fluent interface for defining topologies
- StreamsConfig - Helps to accurately assign configuration properties
- KStream - Represents of an unbounded stream of events (key optional)
- KGroupedStream - A stream of events grouped for aggregation
- KTable - Holds the latest value for each key in an event stream (key required)
- Global KTable - KTable that is replicated for each application instance

Kafka Streams DSL Example

```
final StreamsBuilder builder = new StreamsBuilder();
final Topology topology;

builder.stream(inputTopic, Consumed.with(Serdes.String(), citySerde))
        .filterNot((key, val) -> val.getState() == "NY")
        .map((key, val) -> KeyValue.pair(val.getState(), val.getPopulation()))
        .groupByKey(Grouped.with(Serdes.String(), Serdes.Integer()))
        .reduce((sumVal, val) -> sumVal + val)
        .toStream()
        .to(outputTopic, Produced.with(Serdes.String(), Serdes.Integer()));

topology = builder.build();

System.out.println(topology.describe().toString());

KafkaStreams streams = new KafkaStreams(topology, props);
streams.start();
```

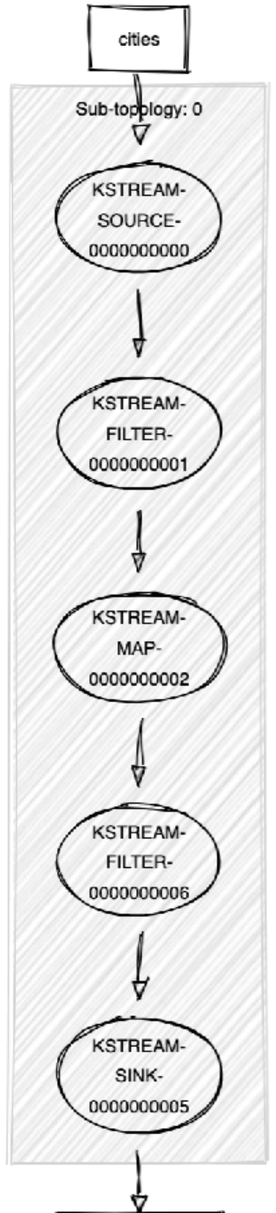
Kafka Streams Topology Visualizer

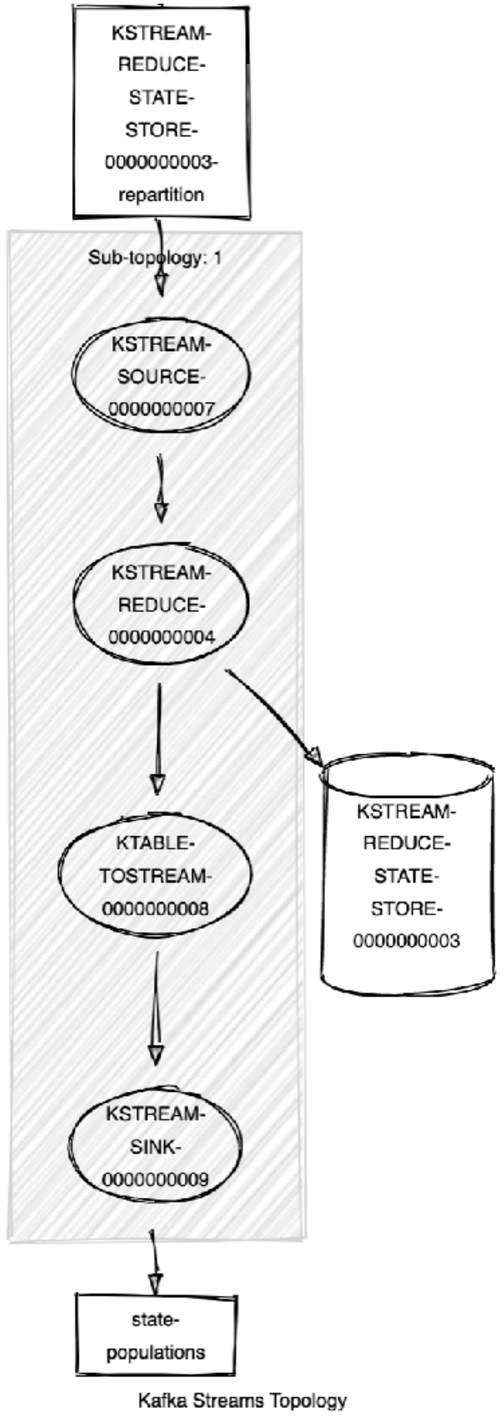
Converts an ASCII Kafka Topology description into a hand drawn diagram. Github link.

Input Kafka Topology update

```
Topologies:
  Sub-topology: 0
   Source: KSTREAM-SOURCE-0000000000 (topics: [cities])
     --> KSTREAM-FILTER-0000000001
   Processor: KSTREAM-FILTER-0000000001 (stores: [])
     --> KSTREAM-MAP-0000000002
     <-- KSTREAM-SOURCE-0000000000
   Processor: KSTREAM-MAP-0000000002 (stores: [])
     --> KSTREAM-FILTER-0000000006
     <-- KSTREAM-FILTER-0000000001
   Processor: KSTREAM-FILTER-0000000006 (stores: [])
     --> KSTREAM-SINK-0000000005
     <-- KSTREAM-MAP-0000000002
   Sink: KSTREAM-SINK-0000000005 (topic: KSTREAM-REDUCE-
STATE-STORE-0000000003-repartition)
     <-- KSTREAM-FILTER-0000000006

  Sub-topology: 1
   Source: KSTREAM-SOURCE-0000000007 (topics: [KSTREAM-
REDUCE-STATE-STORE-0000000003-repartition])
     --> KSTREAM-REDUCE-0000000004
   Processor: KSTREAM-REDUCE-0000000004 (stores: [KSTREAM-
REDUCE-STATE-STORE-0000000003])
     --> KTABLE-TOSTREAM-0000000008
     <-- KSTREAM-SOURCE-0000000007
   Processor: KTABLE-TOSTREAM-0000000008 (stores: [])
```

Output Sketch Diagram





Kafka Streams Topology

https://zz85.github.io/kafka-streams-viz/

Count Events in a Stream

```
builder.stream(inputTopic, Consumed.with(Serdes.String(), ticketSaleSerde))
        .map((k, v) -> new KeyValue<>(v.getTitle(), v))
        .groupByKey(Grouped.with(Serdes.String(), ticketSaleSerde))
        .count()
        .toStream()
        .mapValues(v -> v.toString() + " tickets sold")
        .to(outputTopic, Produced.with(Serdes.String(), Serdes.String()));
```

Split Events Into Multiple Streams

```
KStream<String, ActingEvent>[] branches =
                builder.<String,ActingEvent>stream(inputTopic)
    .branch((key, appearance) -> "drama".equals(appearance.getGenre()),
            (key, appearance) -> "fantasy".equals(appearance.getGenre()),
            (key, appearance) -> true);

branches[0].to(allProps.getProperty("output.drama.topic.name"));
branches[1].to(allProps.getProperty("output.fantasy.topic.name"));
branches[2].to(allProps.getProperty("output.other.topic.name"));
```

Join a Stream and a Table

```
KStream<String, Movie> movieStream = builder.<String, Movie>stream(movieTopic)
    .map((key, movie) -> new KeyValue<>(String.valueOf(movie.getId()), movie));

movieStream.to(rekeyedMovieTopic);

KTable<String, Movie> movies = builder.table(rekeyedMovieTopic);

KStream<String, Rating> ratings = builder.<String, Rating>stream(ratingTopic)
     .map((key, rating) -> new KeyValue<>(String.valueOf(rating.getId()), rating));

KStream<String, RatedMovie> ratedMovie = ratings.join(movies, joiner);
     .to(ratedMoviesTopic, Produced.with(Serdes.String(),
                                         ratedMovieAvroSerde(allProps)));
```

Join a Stream and a Table (ValueJoiner)

```
public class MovieRatingJoiner implements ValueJoiner<Rating, Movie, RatedMovie> {

  public RatedMovie apply(Rating rating, Movie movie) {
    return RatedMovie.newBuilder()
        .setId(movie.getId())
        .setTitle(movie.getTitle())
        .setReleaseYear(movie.getReleaseYear())
        .setRating(rating.getRating())
        .build();
  }
}
```

Count Events by Time Window

```
builder.<String, Rating>stream(ratingTopic)
    .map((key, rating) -> new KeyValue<>(rating.getTitle(), rating))
    .groupByKey()
    .windowedBy(TimeWindows.of(Duration.ofMinutes(10)))
    .count()
    .toStream()
    .map((Windowed<String> key, Long count) -> new KeyValue<>(key.key(),
                                                              count.toString()))
    .to(ratingCountTopic, Produced.with(Serdes.String(), Serdes.String()));
```

Exactly Once Semantics

In Kafka Streams app configuration:

```
processing.guarantee = "exactly_once"
```

Producer config:

```
enable.idempotence = true
max.inflight.requests.per.connection = 1
retries = Integer.MAX_VALUE
transactional.id
```

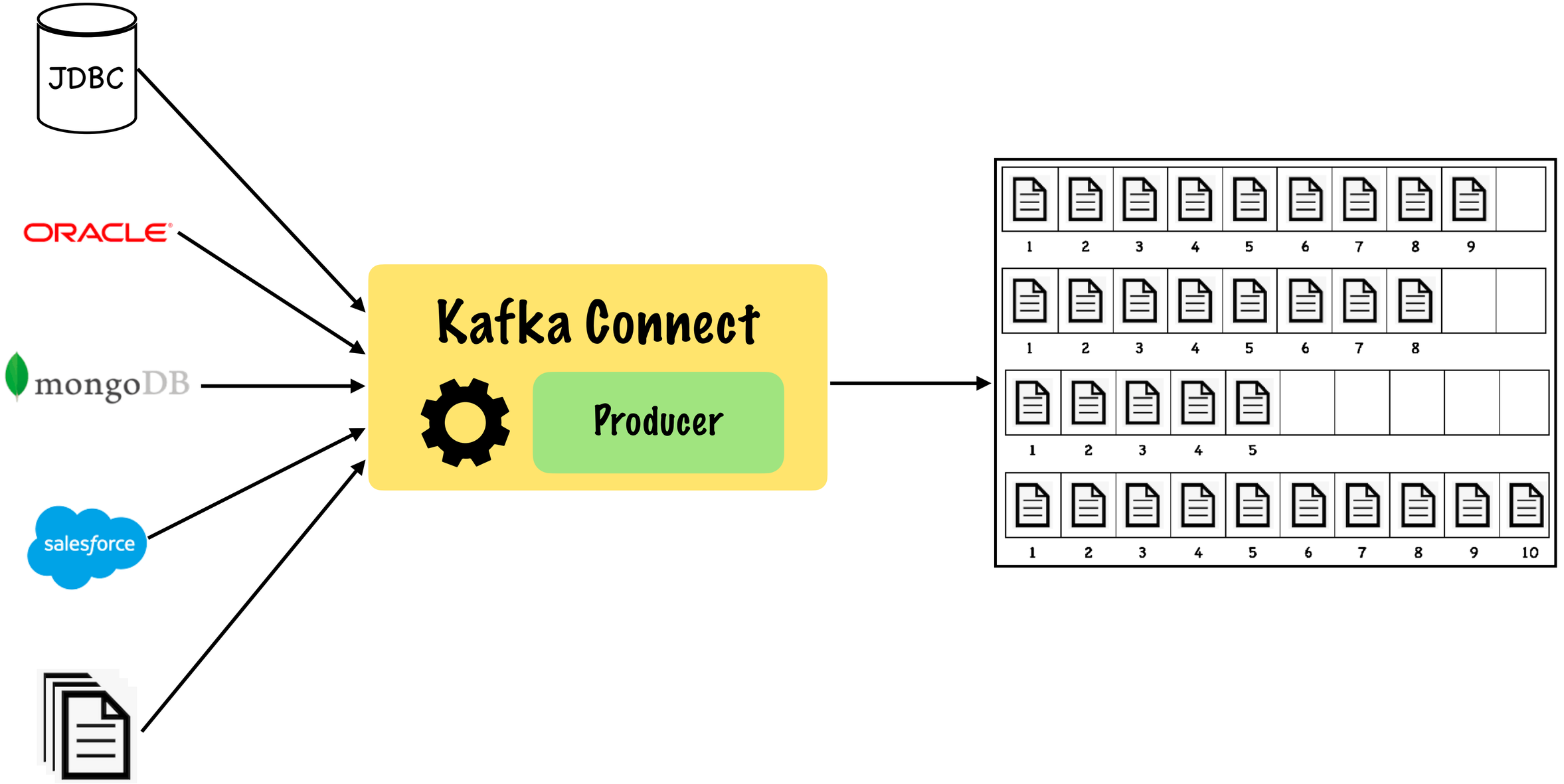
Consumer config:

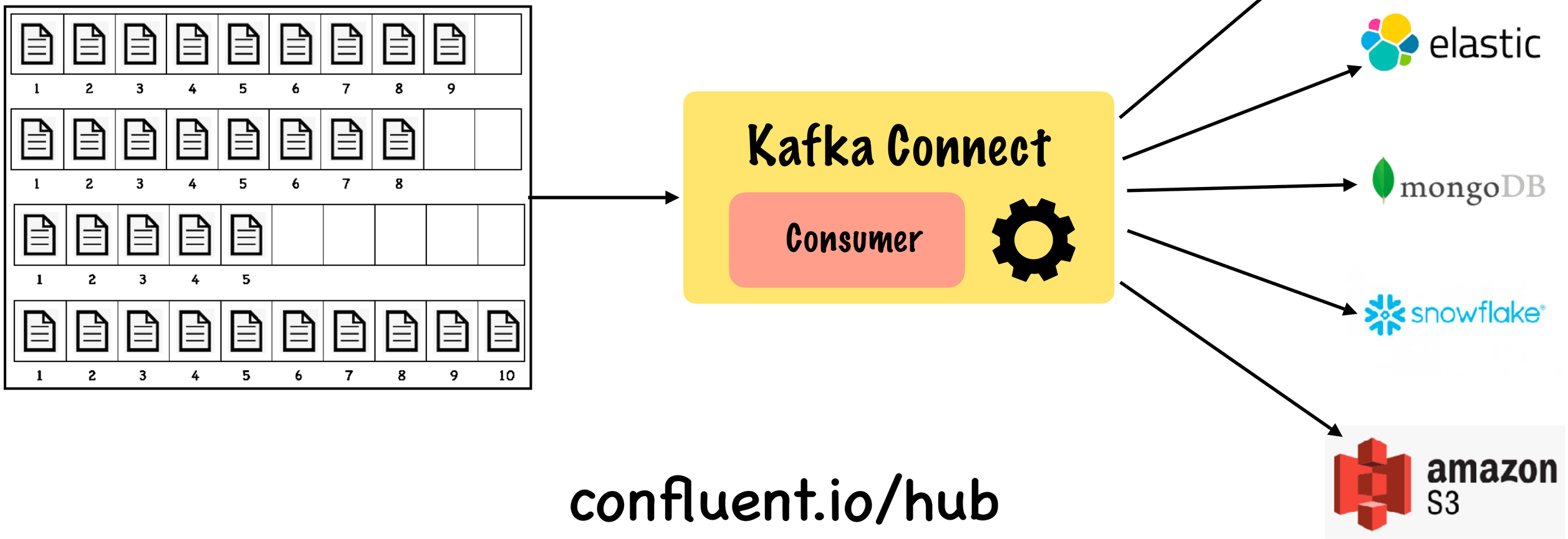
```
isolation.level = "read_committed"
```

https://www.confluent.io/blog/enabling-exactly-once-kafka-streams

Exercises Part 2

More Kafka Tools and Resources


JDBC
ORACLE
mongoDB
salesforce
Kafka Connect
Producer
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8
1 2 3 4 5
1 2 3 4 5 6 7 8 9 10



confluent.io/hub

ksqlDB

```
ksql> CREATE STREAM books (
    title VARCHAR, author VARCHAR, year_published
INT
 ) WITH (
    KAFKA_TOPIC='booklist', VALUE_FORMAT='AVRO'
 );
```

```
ksql> SELECT * FROM books WHERE title LIKE '%kafka%' EMIT CHANGES;
```

TITLE	AUTHOR	YEAR
Kafka: The Definitive Guide	Shapira, Narkhede, Palino	2017
Kafka Streams in Action	Bill Bejeck	2018
Kafka in Action	Dylan Scott	2021
Apache Kafka Cookbook	Raul Estrada	2017

Confluent REST API

```
curl -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" \
     -H "Accept: application/vnd.kafka.v2+json" \
     --data '{"records":[{"value":{"foo":"bar"}}]}' \
     "http://localhost:8082/topics/jsontest"
```

Command Line Tools

Kafka Shell Scripts
- Start and stop brokers
- Create / manage topics
- Produce / consume events

Confluent CLI - https://docs.confluent.io/confluent-cli/current/overview.html
- Start / stop services in Confluent Platform / Cloud
- Manage access control and secrets

kcat (formerly Kafkacat) - https://github.com/edenhill/kcat
- Produce / consume events in style

kcctl (Casey Cuddle) - https://github.com/kcctl/kcctl
- Command-line client for Kafka Connect

CONFLUENT

Search

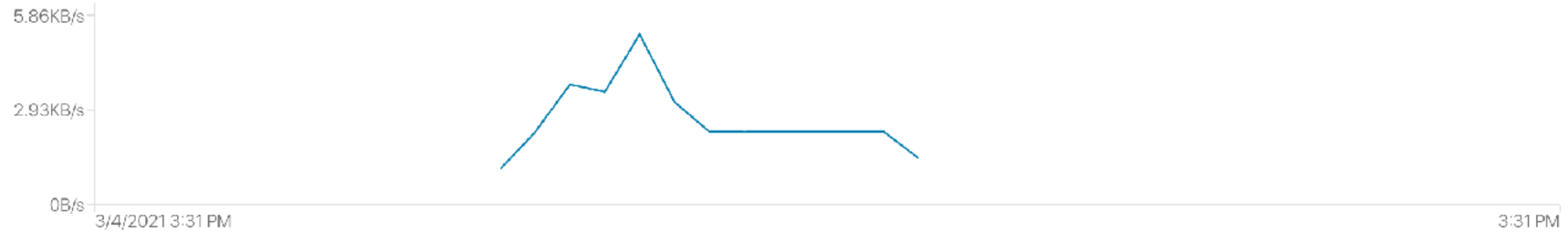
- Cluster overview
- Data flow
- Topics
- Connectors
- Consumers
- ksqlDB
- API access
- Clients
- CLI and Tools
- Cluster settings

Overview

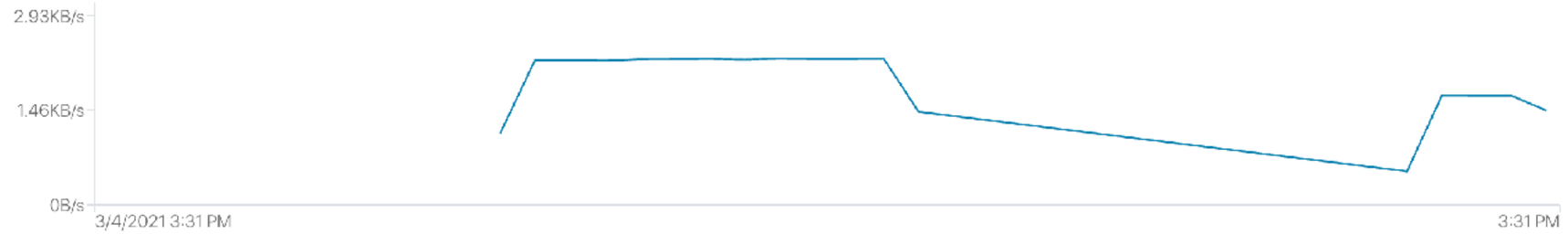
Last 7 days

Throughput

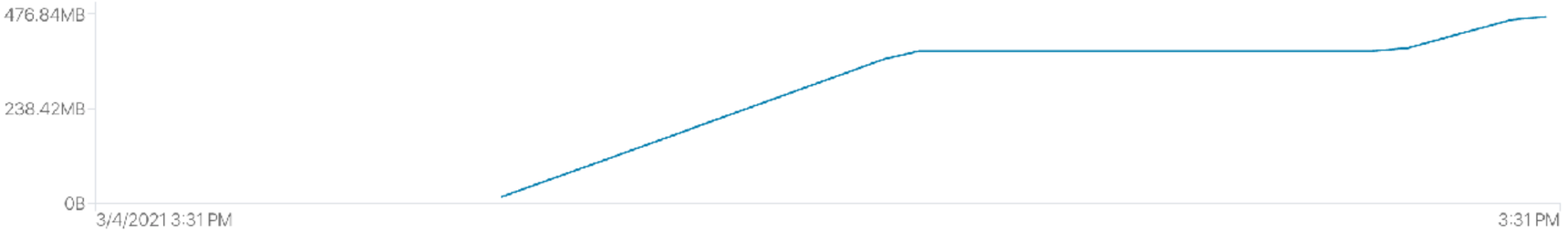
Consumption (bytes/sec)



Production (bytes/sec)



Storage



Cluster overview
Data flow
Topics
Connectors
Consumers
ksqlDB
API access
Clients
CLI and Tools
Cluster settings

C#

CONFLUENT SUPPORTED

C/C++

CONFLUENT SUPPORTED

Clojure

Go

CONFLUENT SUPPORTED

Groovy

Java

CONFLUENT SUPPORTED

See example Copy

```
# Required connection configs for Kafka producer, consumer, and admin
bootstrap.servers=pkc-43n10.us-central1.gcp.confluent.cloud:9092
security.protocol=SASL_SSL
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule   required username='{{ CLUSTER_API_KEY }}'   password='{{
CLUSTER_API_SECRET }}';
sasl.mechanism=PLAIN
# Required for correctness in Apache Kafka clients prior to 2.6
client.dns.lookup=use_all_dns_ips
```

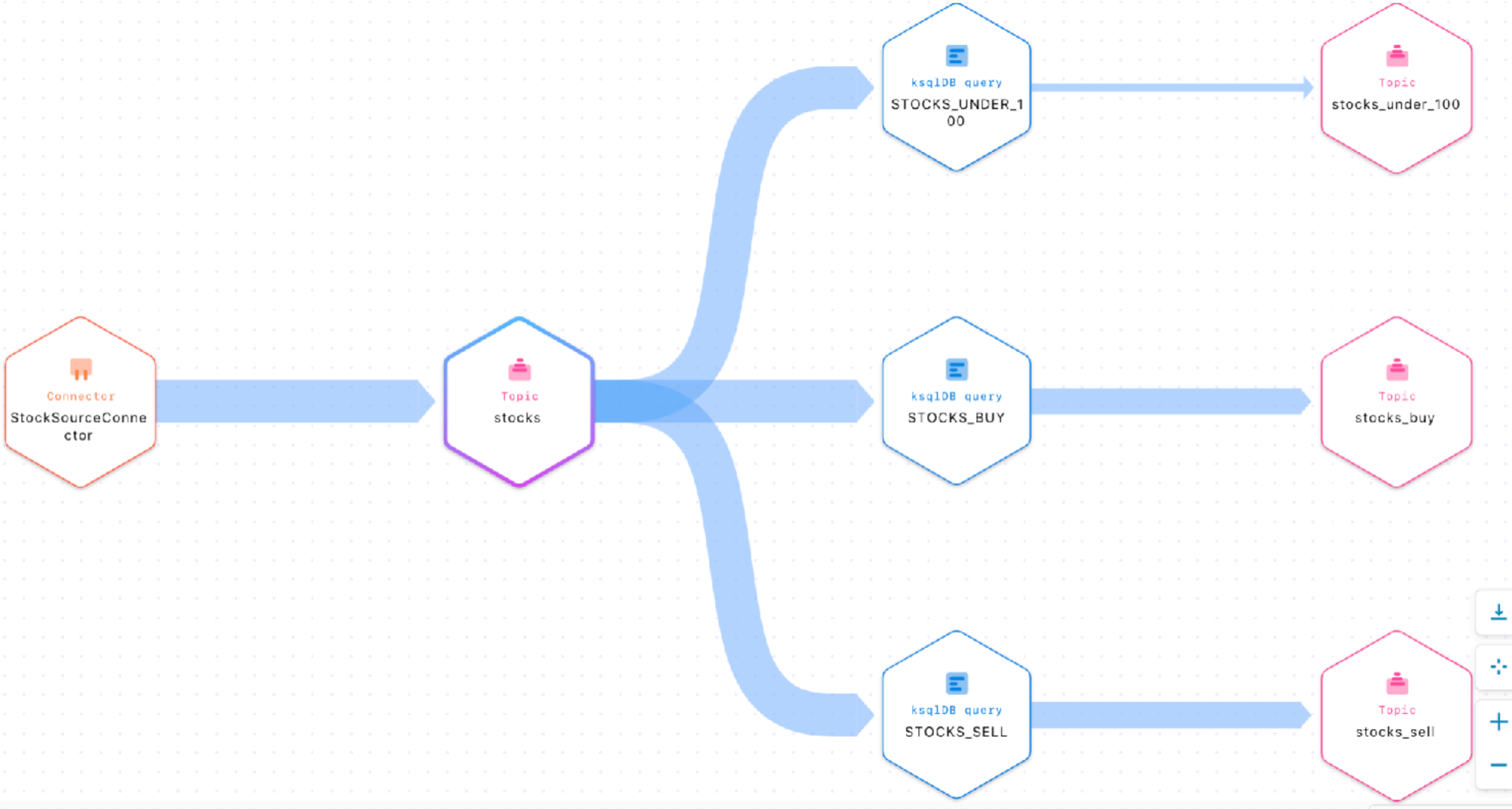
Kotlin

Node.js

Python

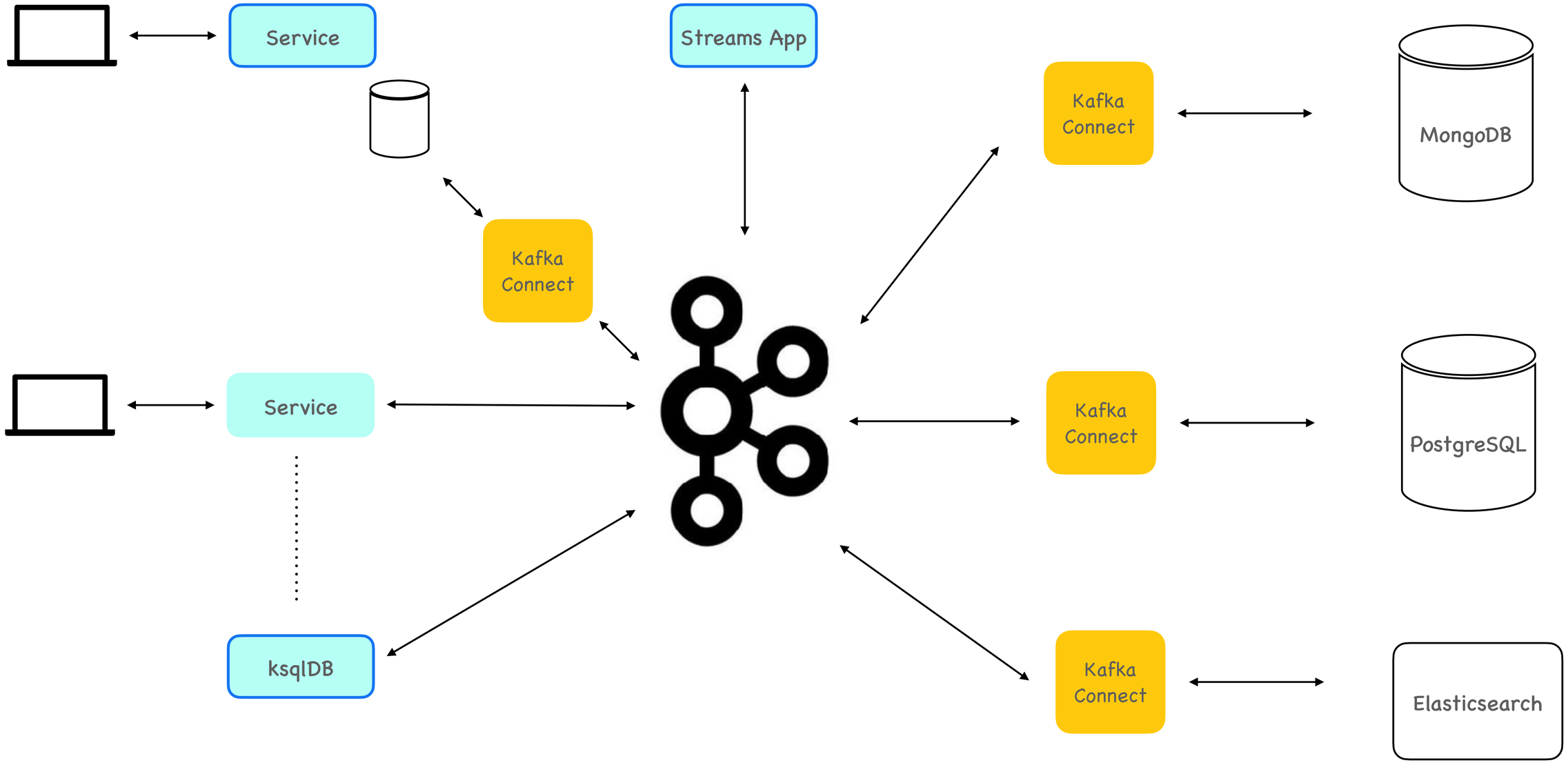
CONFLUENT SUPPORTED

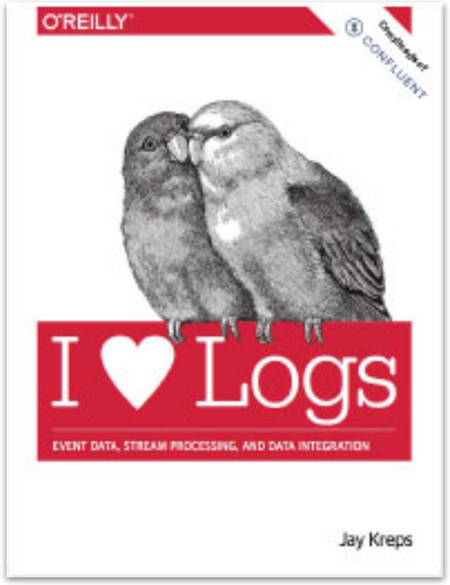
Hide internal topics Use consumer groups*

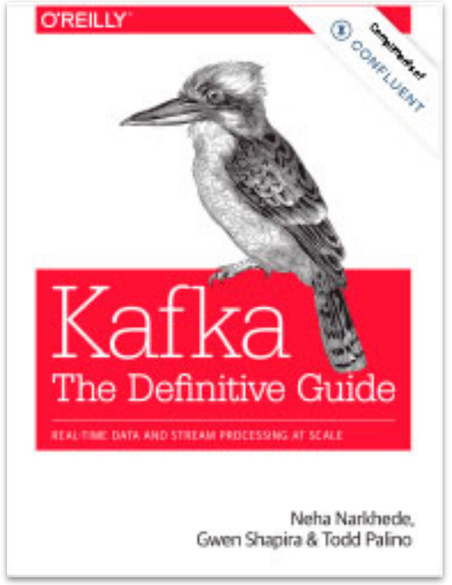


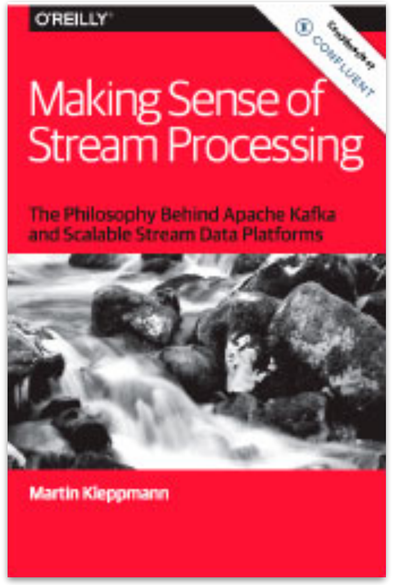
All

Event Streaming Platform

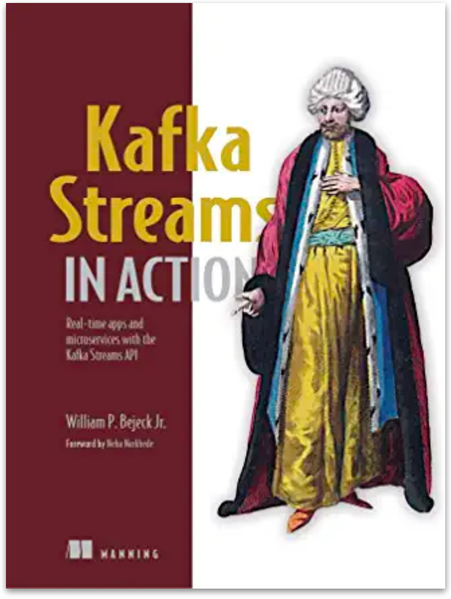


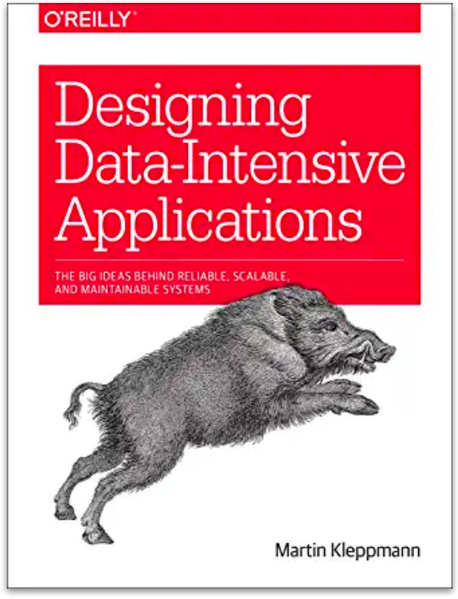

O'REILLY
CONFLUENT
I ♥ Logs
EVENT DATA, STREAM PROCESSING, AND DATA INTEGRATION
Jay Kreps

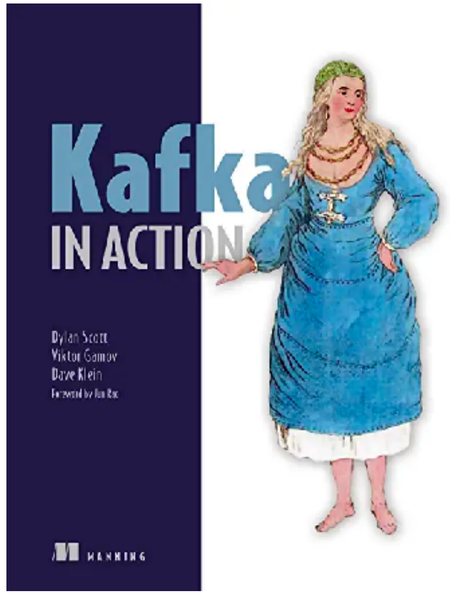

O'REILLY
CONFLUENT
Kafka
The Definitive Guide
REAL-TIME DATA AND STREAM PROCESSING AT SCALE
Neha Narkhede,
Gwen Shapira & Todd Palino

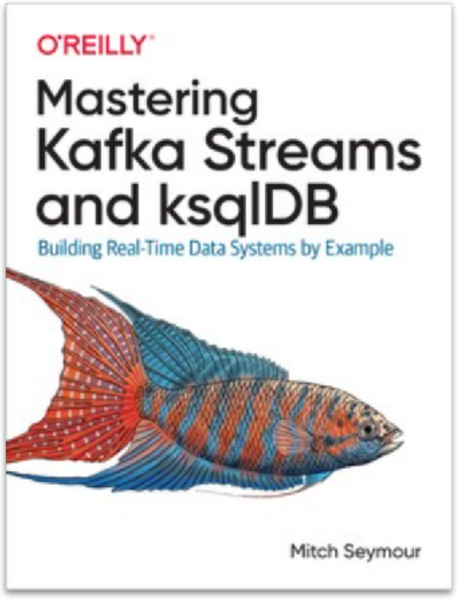

O'REILLY
CONFLUENT
Making Sense of Stream Processing
The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms
Martin Kleppmann


O'REILLY
Designing Event-Driven Systems
Concepts and Patterns for Streaming Services with Apache Kafka
Ben Stopford
Foreword by Sam Newman


Kafka Streams
IN ACTION
Real-time apps and microservices with the Kafka Streams API
William P. Bejeck Jr.
MANNING


O'REILLY
Designing Data-Intensive Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS
Martin Kleppmann


Kafka
IN ACTION
Dylan Scott
Viktor Gamov
Dave Klein
MANNING


O'REILLY
Mastering Kafka Streams and ksqlDB
Building Real-Time Data Systems by Example
Mitch Seymour

CONFLUENT Developer

COURSES LEARN DESIGN, BUILD, RUN COMMUNITY DOCS QUICK START

What Does Kafka Do?

Learn about the fundamentals of Kafka, event streaming, and the surrounding ecosystem. Click on an element to find out more.

Event Streaming

Create, import, share streams of events like payments, orders, and database changes in milliseconds, at scale. Store them as long as you need. Replay and reprocess historical data like a time machine.

Continuous Real-time Processing

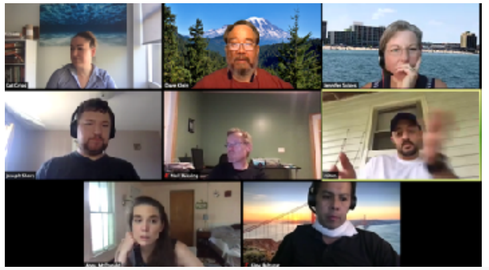
Build lightweight, elastic applications and microservices that respond immediately to events and that scale during live operations. Process, join, and analyze streams and tables of data in real-time, 24x7.

Your Data, Everywhere

Use one platform to set data in motion across your entire enterprise. Connect systems, data centers, and clouds—all

https://developer.confluent.io

https://cnfl.io/book-bundle



https://cnfl.io/meetup-hub

Community

https://cnfl.io/community

twitter.com/daveklein

dklein@confluent.io

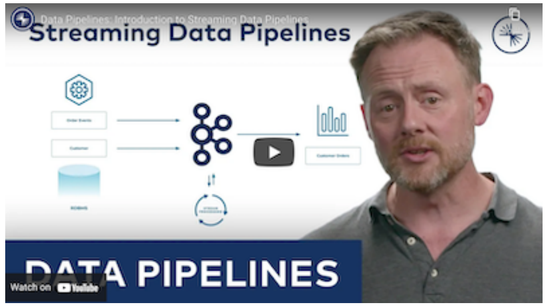
Kafka 101

Spring & Kafka

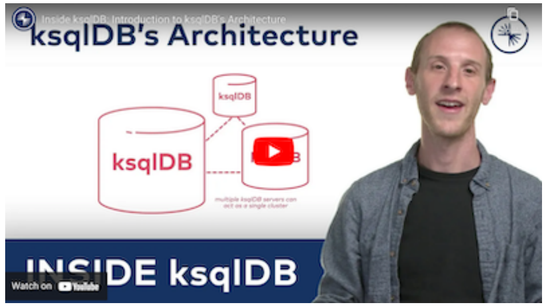
Kafka Connect

Kafka Internals

Kafka Streams 101



Data Pipelines

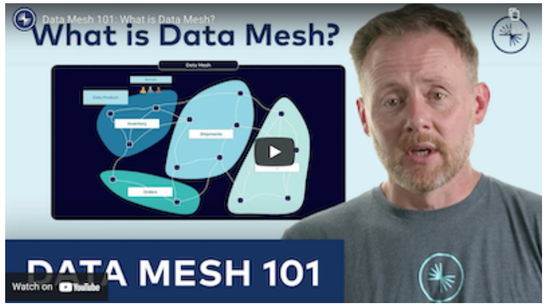


Inside ksqlDB

Kafka Security

ksqlDB 101

Event Sourcing



Data Mesh 101

Schema Registry

‹ Back to Courses

COURSE: APACHE KAFKA® 101

Introduction 5 MIN


Tim Berglund
Sr. Director, Developer Advocacy (Course Presenter)

What Is Apache Kafka®?

Apache Kafka is an event streaming platform used to collect, process, store, and integrate data at scale. It has numerous use cases including distributed logging, stream processing, data integration, and pub/sub messaging.

In order to make complete sense of what Kafka does, we'll delve into what an "event streaming platform" is and how

Related Resources

BLOG
Intro to Apache Kafka: How Kafka Works

TUTORIAL
Apache Kafka Quick Start

CONFERENCE TALK
Kafka Summit: Welcome to Kafka; We're Glad You're Here

E-BOOK
Kafka: The Definitive Guide