Producers and Consumers

Paweł Kordek
DATA ENGINEER

@pawel_kordek https://kordek.github.io

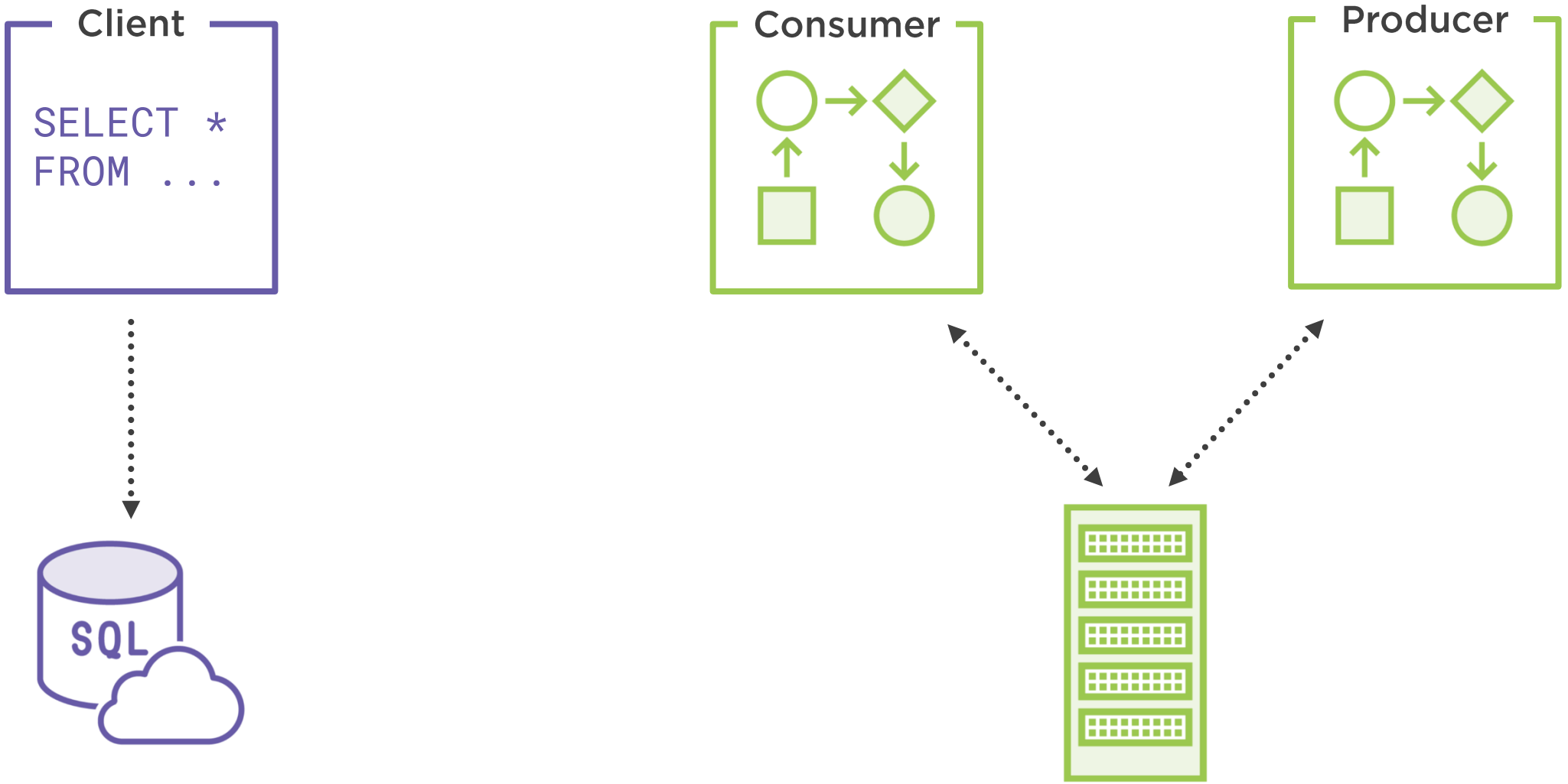
Some questions

How are partitions chosen?

How are they assigned to consumers?

How are messages not lost?

How do consumers commit offsets?


Client
SELECT *
FROM ...
SQL
Consumer
Producer

Booking Service

```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("key.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");
props.setProperty("value.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<String, String> = new KafkaProducer<>(props);

ProducerRecord<String, String> r =
    new ProducerRecord("quote-feedback", "Booking 44321 accepted.");

producer.send(r);
```

Key in Kafka

Optional part of a ProducerRecord

Used for partitioning

Choosing a partition based on its value is producer's job

Booking Service

```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("key.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");
props.setProperty("value.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<String, String> = new KafkaProducer<>(props);

ProducerRecord<String, String> r =
    new ProducerRecord("quote-feedback", 1094, "Booking 44321 accepted.");

producer.send(r);
```

Booking Service

```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("key.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");
props.setProperty("value.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<Integer, String> = new KafkaProducer<>(props);

ProducerRecord<Integer, String> r =
    new ProducerRecord("quote-feedback", 1094, "Booking 44321 accepted.");

producer.send(r);
```

Booking Service

```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("key.serializer", IntegerSerializer.class.getName());
props.setProperty("value.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<Integer, String> = new KafkaProducer<>(props);

ProducerRecord<Integer, String> r =
    new ProducerRecord("quote-feedback", 1094, "Booking 44321 accepted.");

producer.send(r);
```

Booking Service

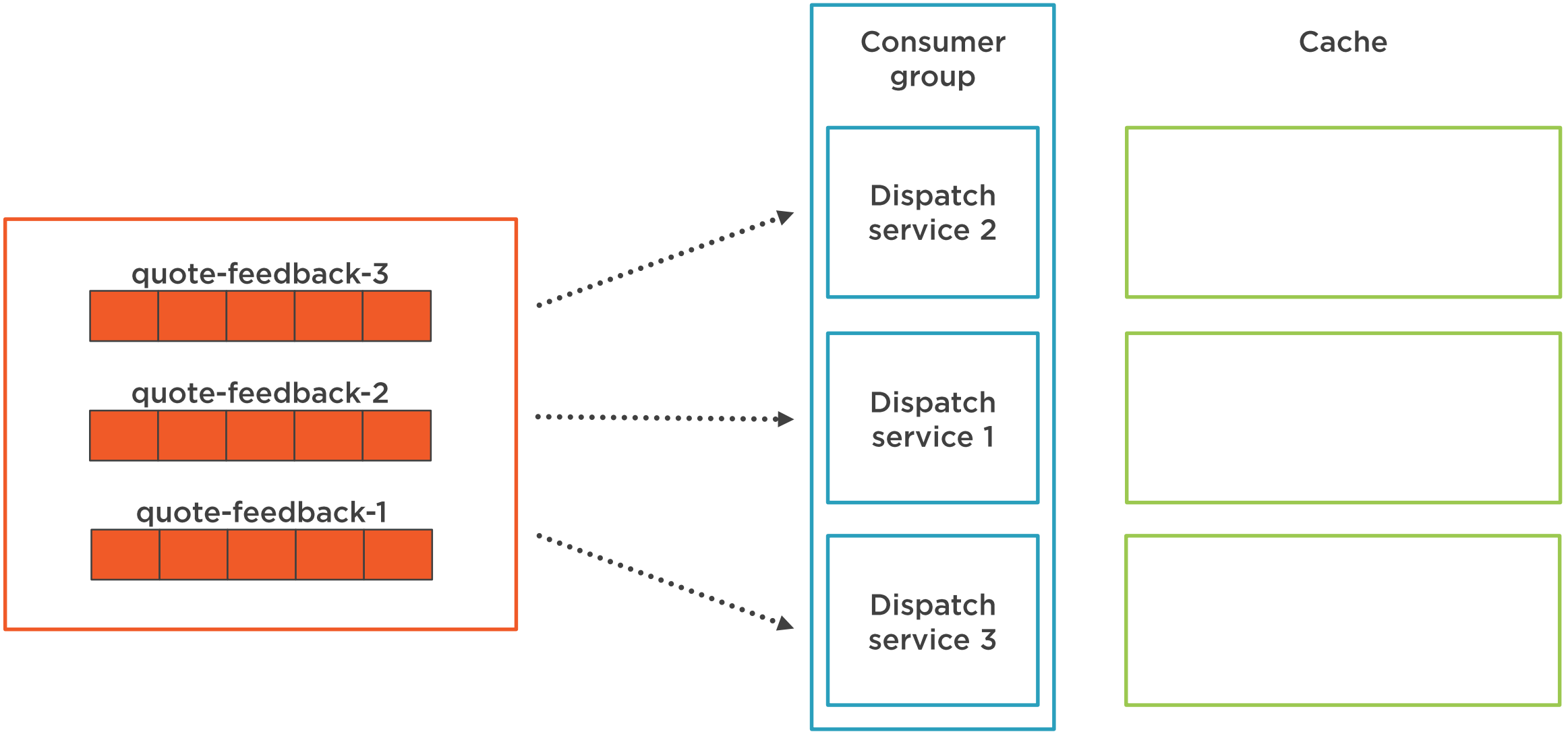
```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("key.serializer", IntegerSerializer.class.getName());
props.setProperty("value.serializer", StringSerializer.class.getName());

KafkaProducer<Integer, String> = new KafkaProducer<>(props);

ProducerRecord<Integer, String> r =
    new ProducerRecord("quote-feedback", 1094, "Booking 44321 accepted.");

producer.send(r);
```

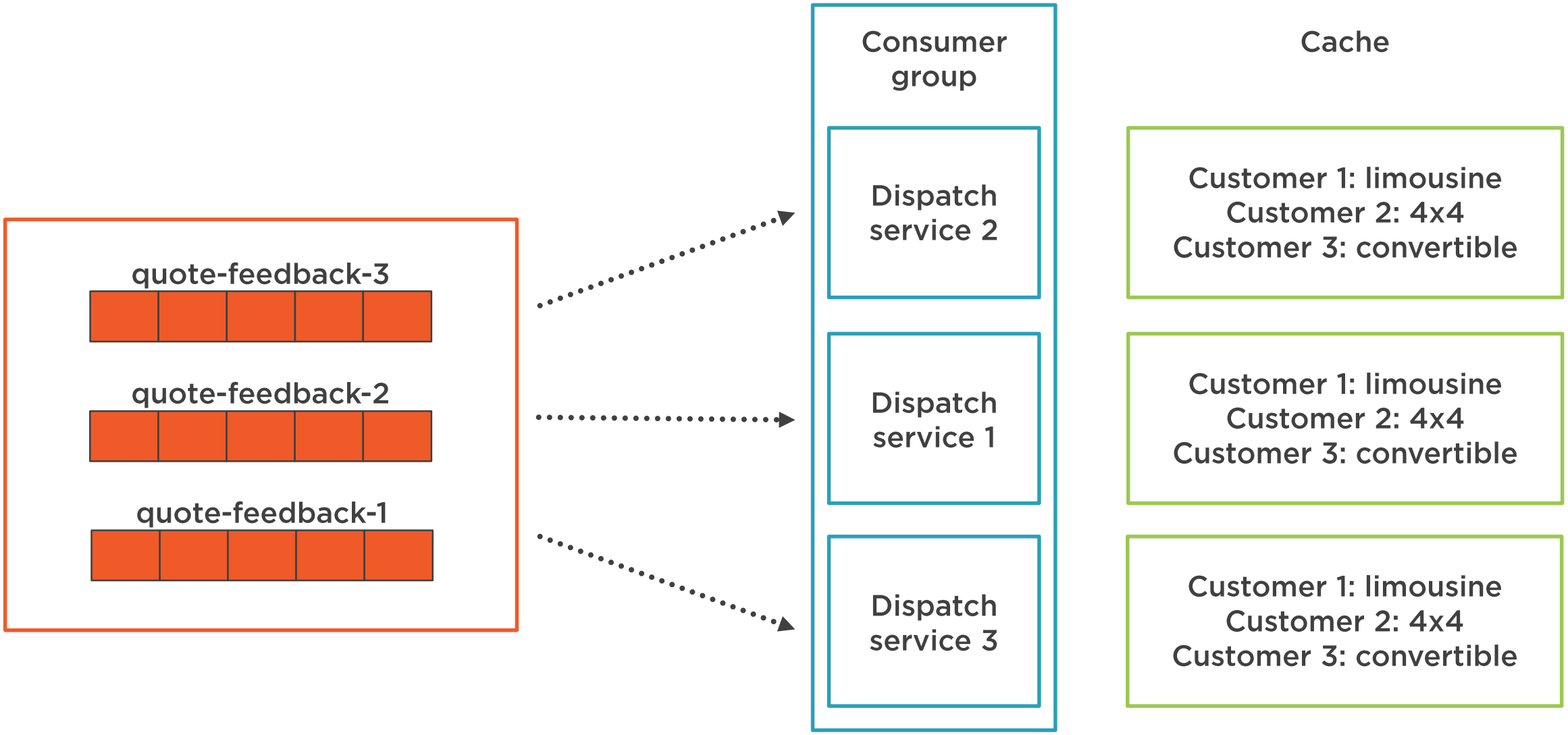
Demo

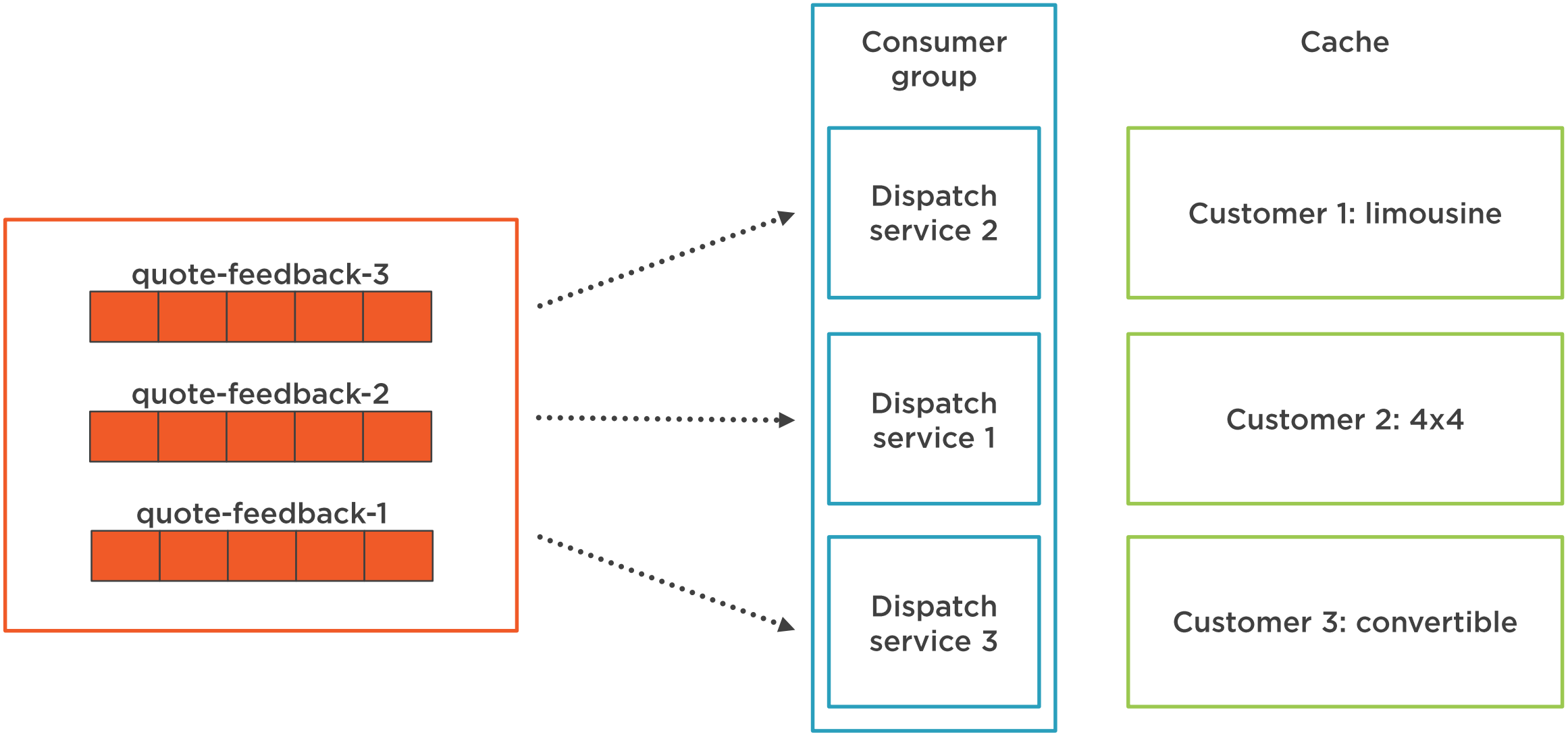
Verify how producer assigns partitions

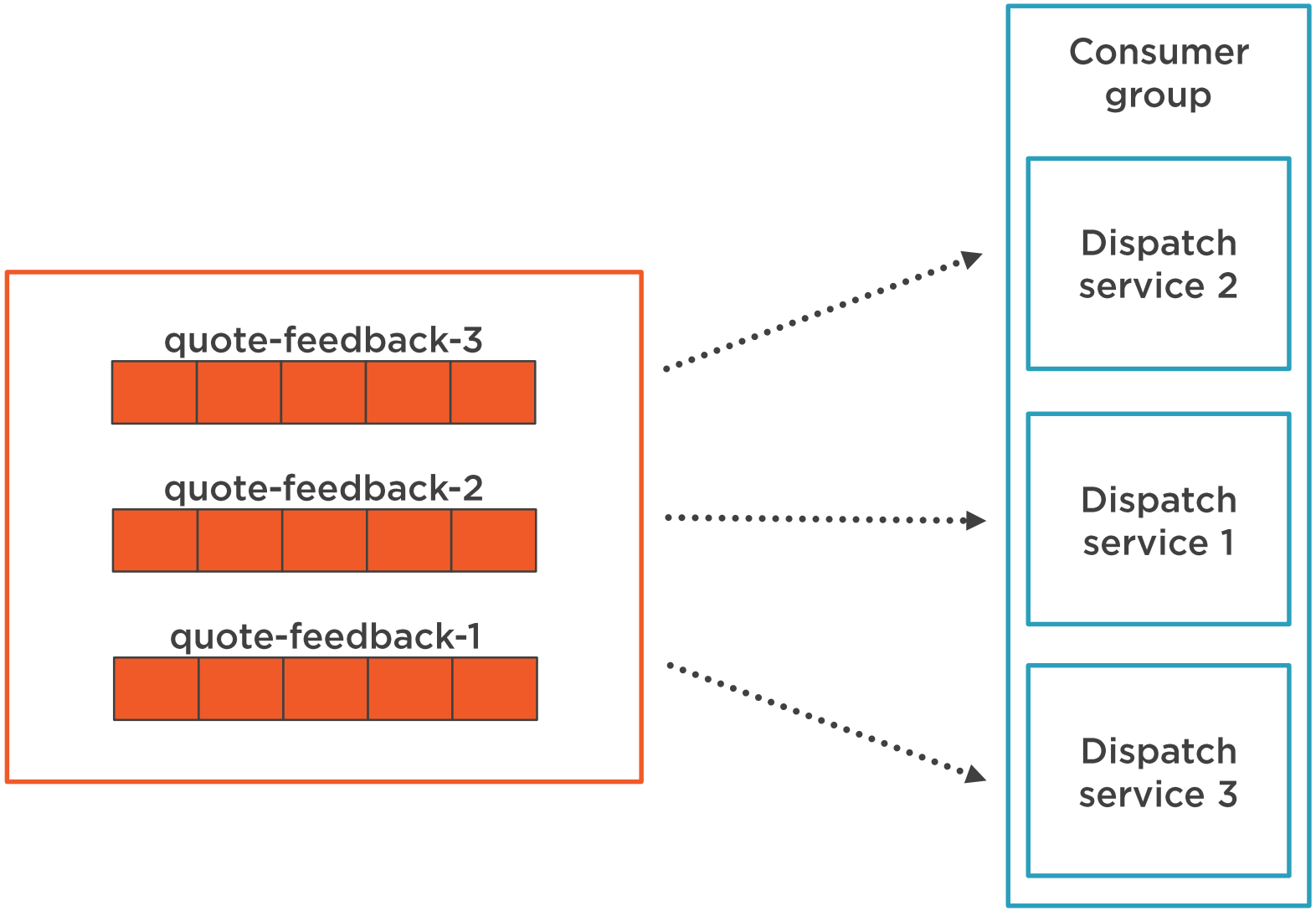
Choosing the Right Key

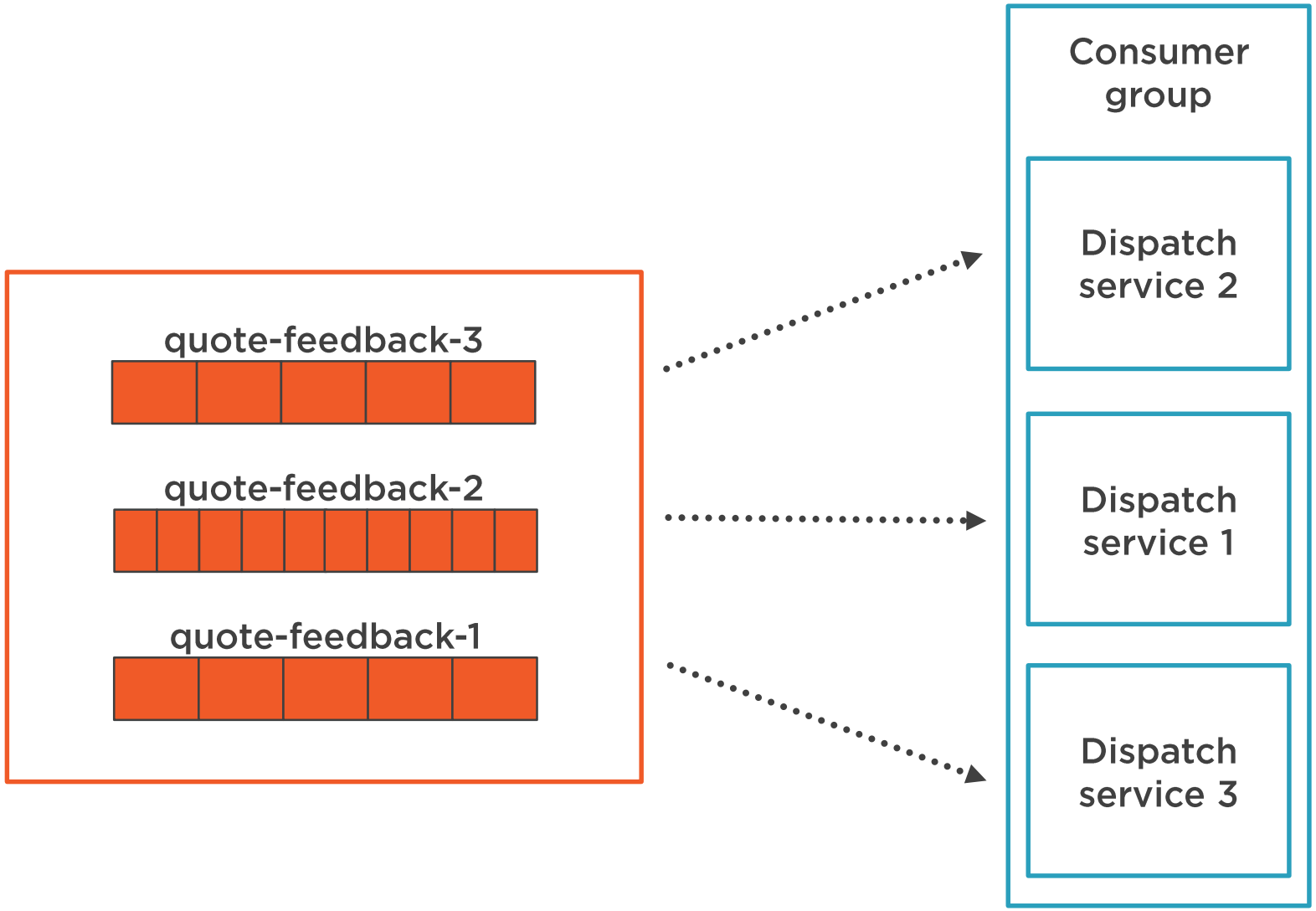
Think about your consumption patterns

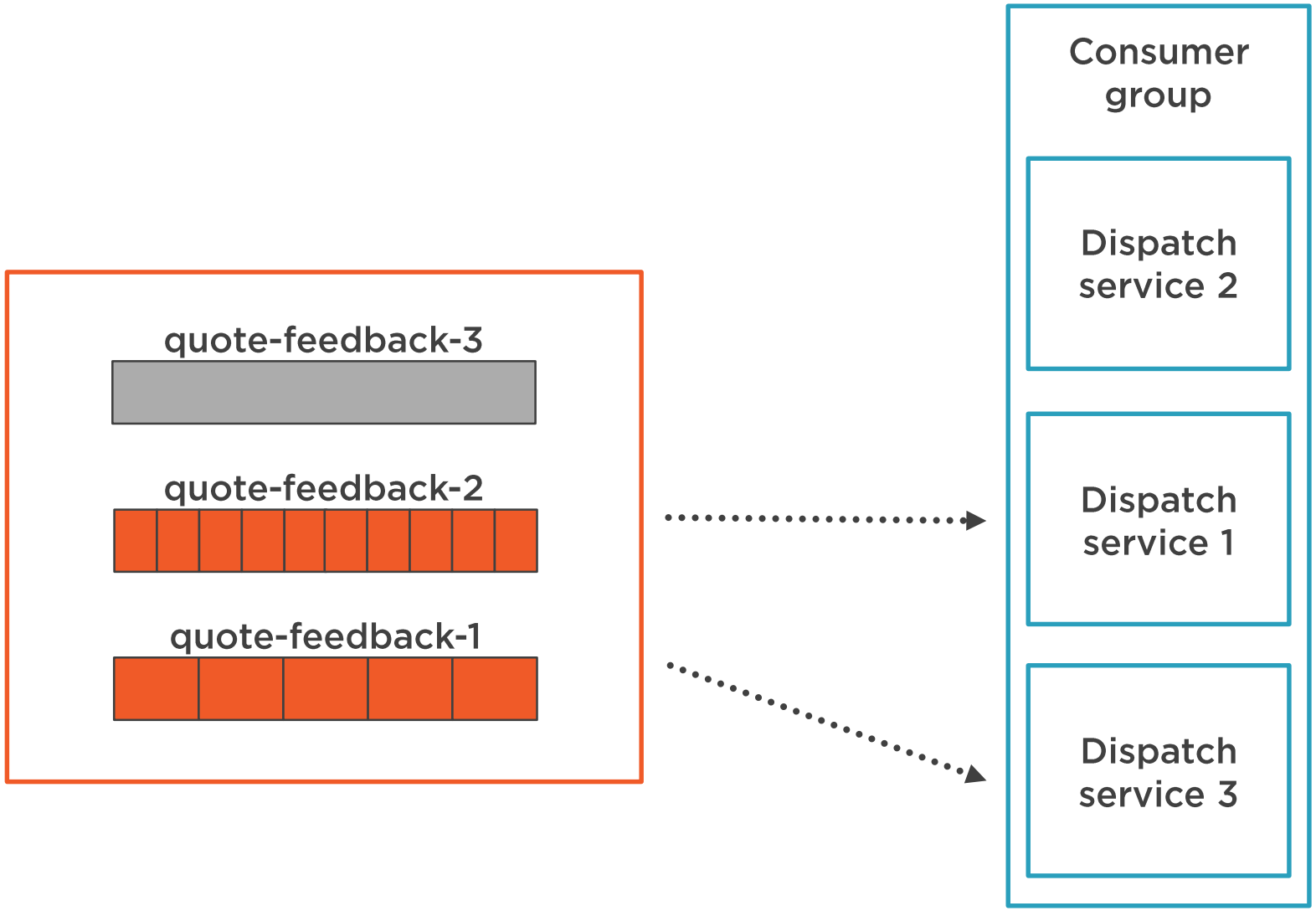

quote-feedback-3
quote-feedback-2
quote-feedback-1
Consumer group
Dispatch service 2
Dispatch service 1
Dispatch service 3
Cache


quote-feedback-3
quote-feedback-2
quote-feedback-1
Consumer group
Dispatch service 2
Dispatch service 1
Dispatch service 3
Cache
Customer 1: limousine
Customer 2: 4x4
Customer 3: convertible
Customer 1: limousine
Customer 2: 4x4
Customer 3: convertible
Customer 1: limousine
Customer 2: 4x4
Customer 3: convertible


quote-feedback-3
quote-feedback-2
quote-feedback-1
Consumer group
Dispatch service 2
Dispatch service 1
Dispatch service 3
Cache
Customer 1: limousine
Customer 2: 4x4
Customer 3: convertible


quote-feedback-3
quote-feedback-2
quote-feedback-1
Consumer group
Dispatch service 2
Dispatch service 1
Dispatch service 3


quote-feedback-3
quote-feedback-2
quote-feedback-1
Consumer group
Dispatch service 2
Dispatch service 1
Dispatch service 3


quote-feedback-3
quote-feedback-2
quote-feedback-1
Consumer group
Dispatch service 2
Dispatch service 1
Dispatch service 3

Order of messages can be guaranteed only within a single partition

Custom partitioning is possible, but should be reserved for special cases

Automatic Retries

Maximum number limited by the 'retries' property

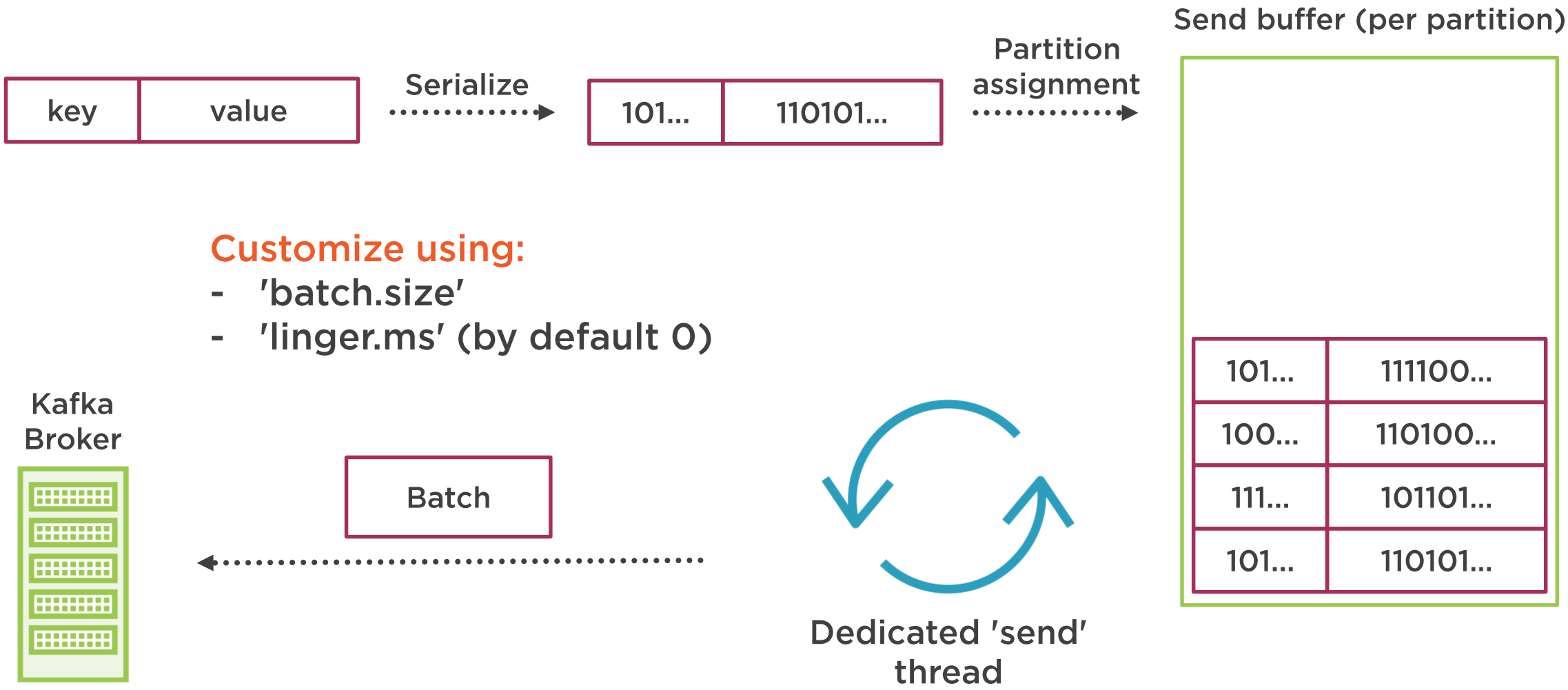
Timeout specified by 'delivery.timeout.ms'

Batching and Compression

Booking Service

```
ProducerRecord<Integer, String> r =
    new ProducerRecord("quote-feedback", 1094, "Booking 44321 accepted.");

producer.send(r);

producer.flush();
```

Producer Flow



Latency vs. Throughput

Quote feedback	Location data
Latency is important	High-volume
Volume is low	Increased latency admissible
We can keep 'linger.ms' at 0	'linger.ms' larger than 0 - beneficial

Compression

Enabled by setting 'compression.type'

Optimizes network and broker disk usage

Although disabled by default, usually recommended

Demo

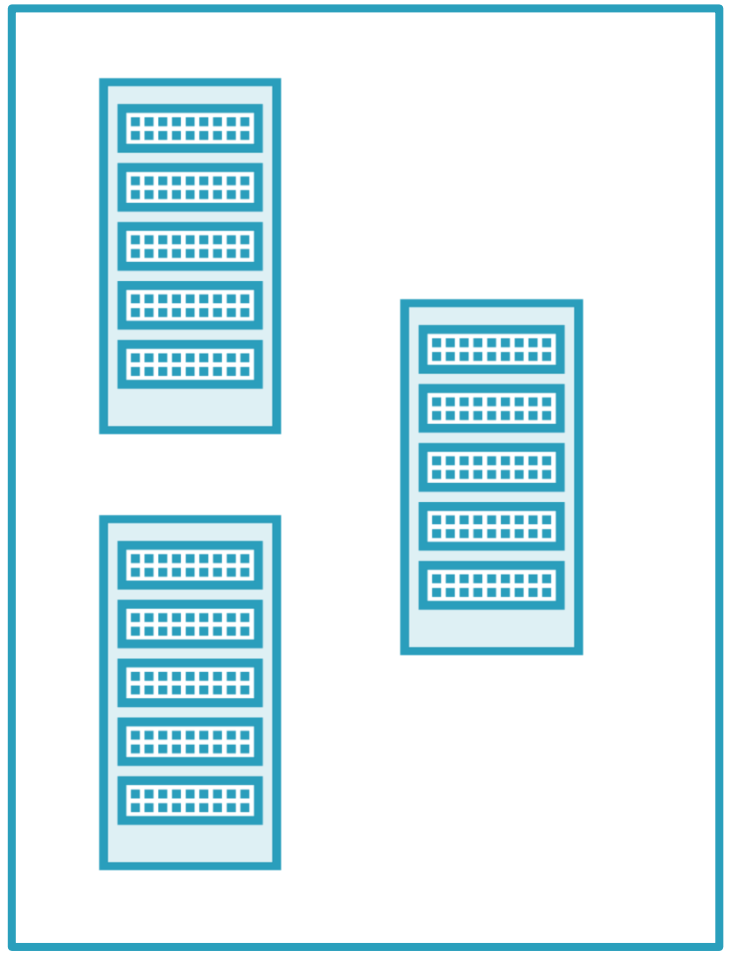
Retries

Batching

Compression

Consumers

C1



Only consumer – gets all partitions

C1

Group coordinator

Group members

- C1

Only consumer – gets all partitions

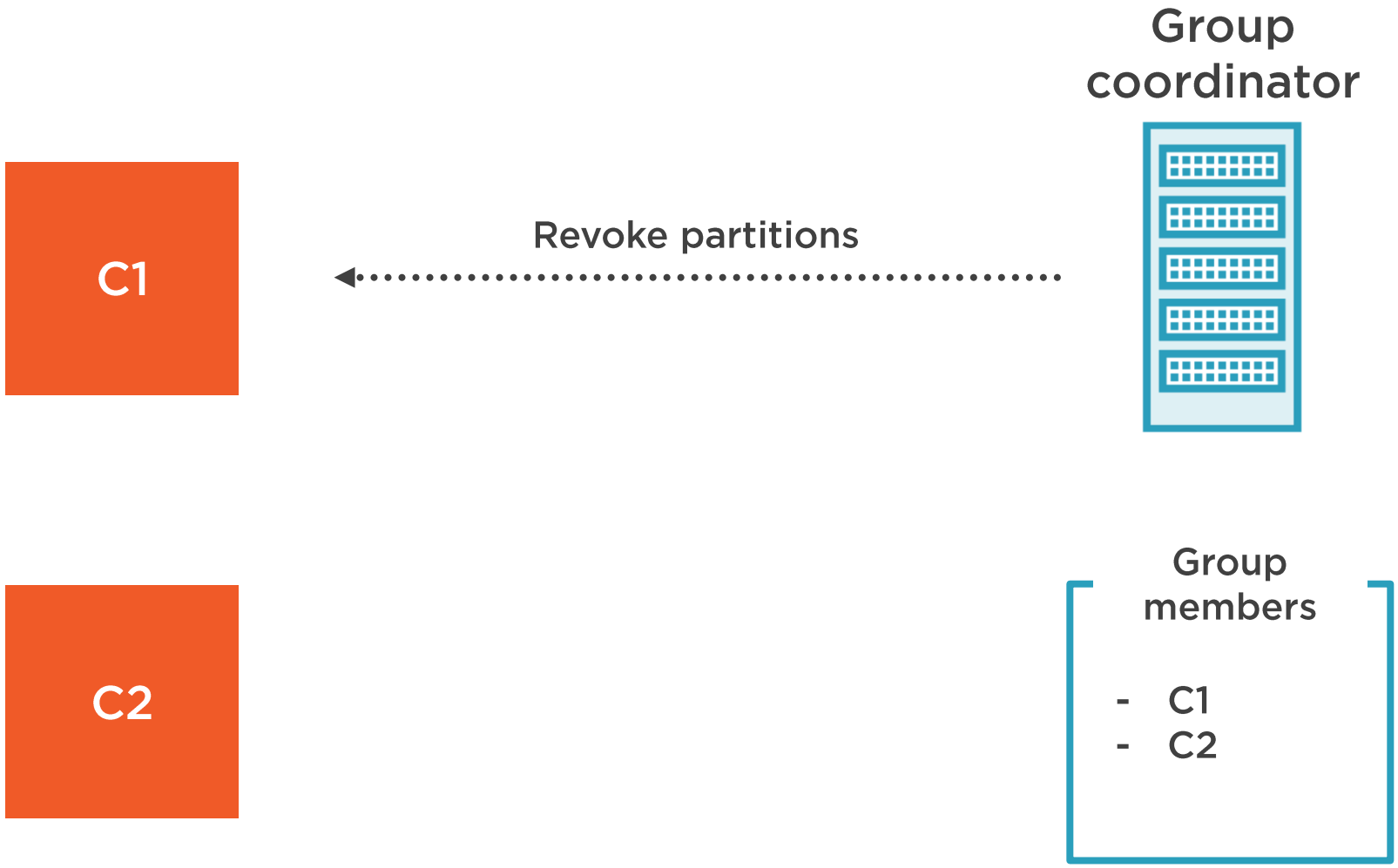
C1

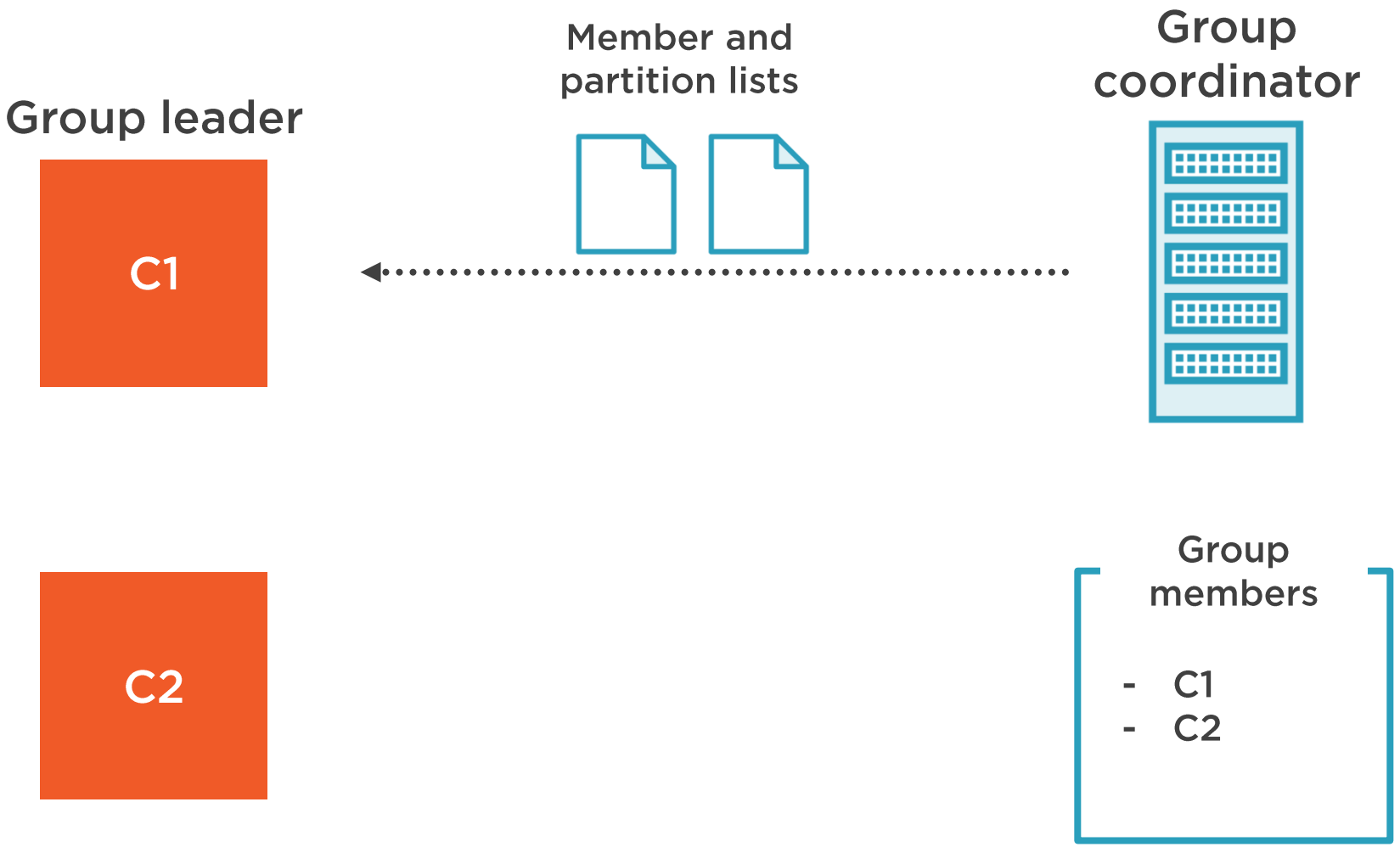
C2

Group coordinator

Group members

- C1
- C2


Group coordinator
C1
Revoke partitions
C2
Group members
- C1
- C2


Group leader
C1
C2
Member and
partition lists
Group
coordinator
Group
members
- C1
- C2

Group leader

C1

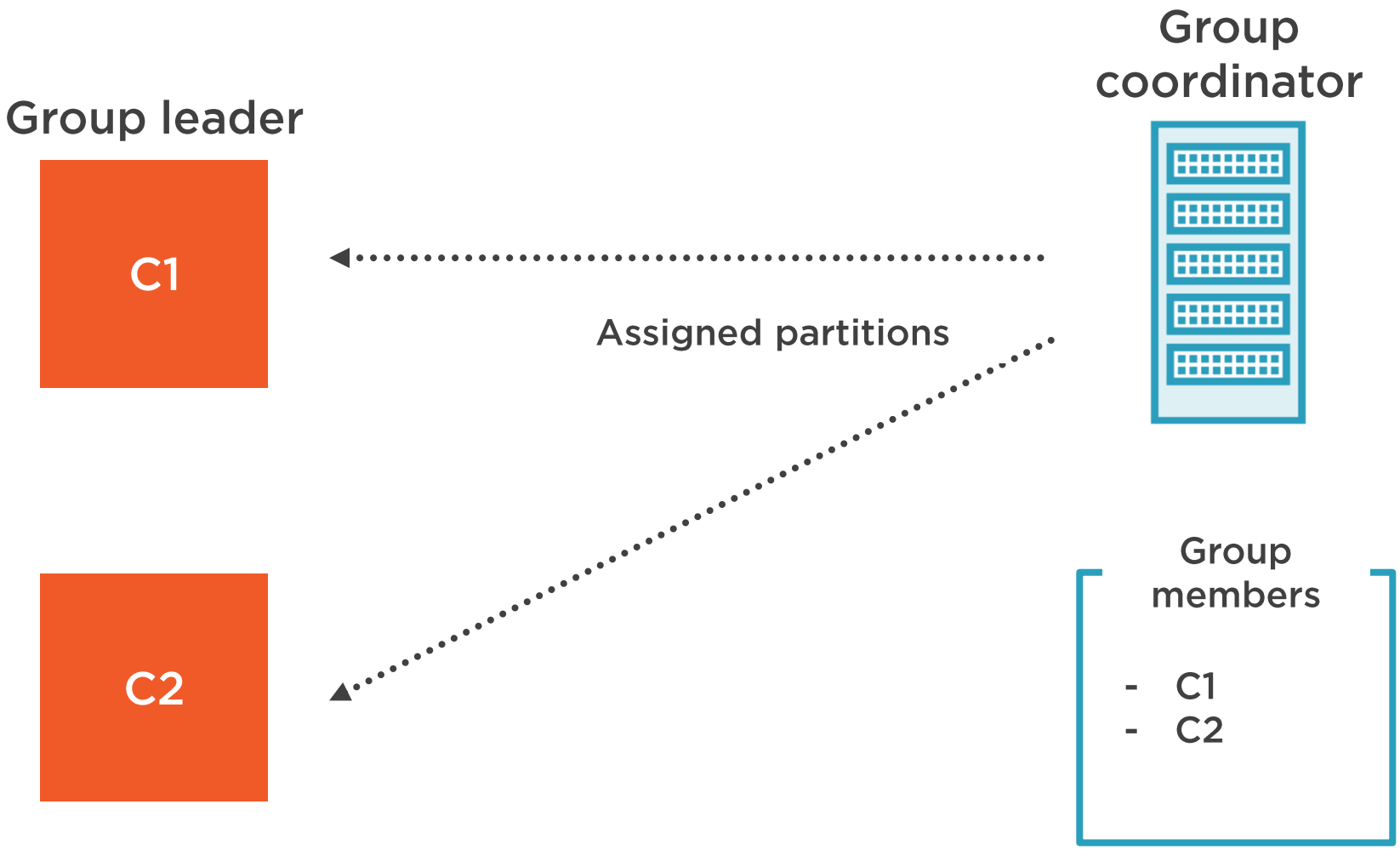
C2

Partition assignments

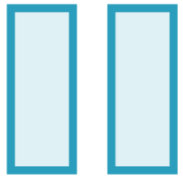
Group coordinator

Group members

- C1
- C2


Group leader
C1
C2
Group coordinator
Assigned partitions
Group members
- C1
- C2

Rebalance



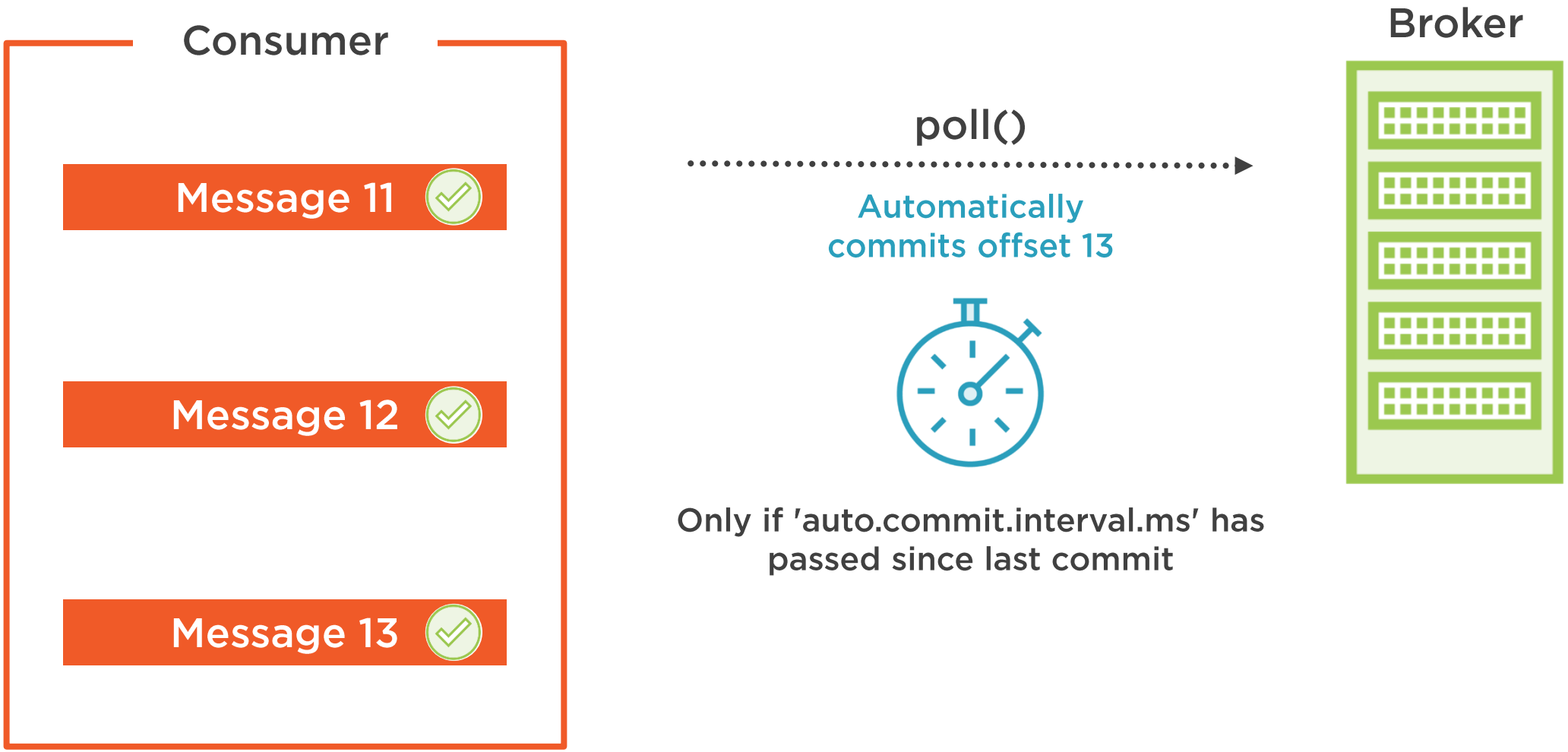
There is always a gap in processing

Some partitions will change ownership

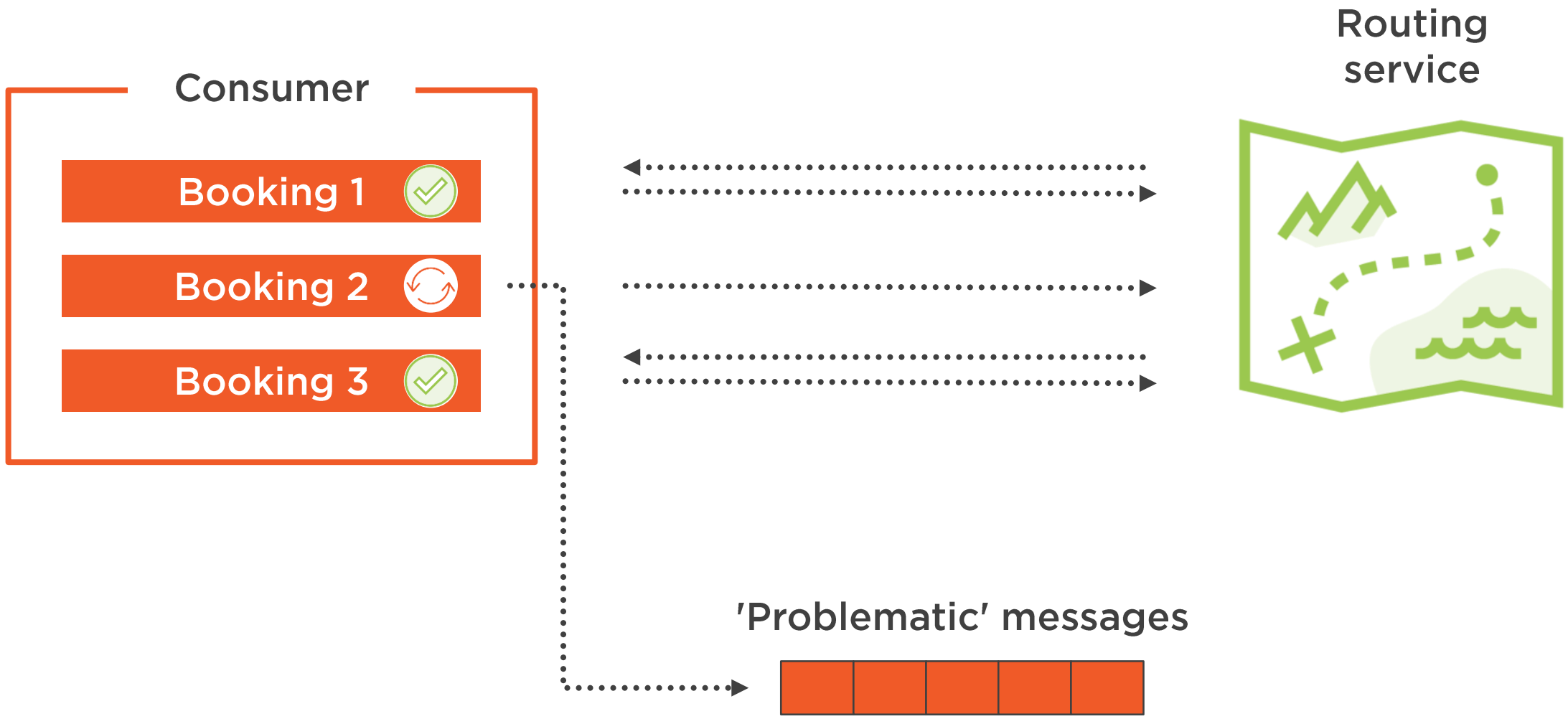
Static assignment is possible

Offsets

Auto Commit


Consumer
Message 11
Message 12
Message 13
poll()
Automatically
commits offset 13
Only if 'auto.commit.interval.ms' has
passed since last commit
Broker

Auto Commit

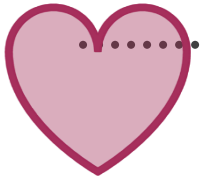

Routing service
Consumer
Booking 1
Booking 2
Booking 3
'Problematic' messages

Consumer Failures

heartbeat.interval.ms

Group coordinator

consumer.close()



'I am leaving!'

session.timeout.ms

Heartbeats run in the background

They do not protect from consumer being stuck and not polling

Therefore the 'max.poll.interval.ms' property

Demo

Offsets management

Consumer failure

Duplicate Messages

At-least-once guarantee

Exactly-once is tricky

Summary

Kafka is a complex technology

Core concepts necessary for reasoning about a system has been laid out

You always need to consult the docs