

Confluent Developer Skills for Building Apache Kafka® Applications

Version 7.0.0-v1.0.5



CONFLUENT

Table of Contents

Introduction	
Class Logistics and Overview	1
Fundamentals Review	2
	9
Core Overview	12
01: Introductory Concepts	14
01a: How Can You Connect to a Cluster?	16
01b: How Do You Control How Kafka Retains Messages?	22
01c: How Can You Leverage Replication?	30
Lab: Introduction	38
Lab: Using Kafka's Command-Line Tools	39
02: Starting with Producers	40
02a: What are the Basic Concepts of Kafka Producers?	42
02b: How Do You Write the Code for a Basic Kafka Producer?	49
Lab: Basic Kafka Producer	61
03: Preparing Producers for Practical Uses	62
03a: How Can Producers Leverage Message Batching?	64
03b: How Do Producers Know Brokers Received Messages?	76

03c: How Can a Producer React to Failed Delivery?	88
04: Starting with Consumers	95
04a: How Do You Request Data to Fetch from Kafka?	97
04b: What are the Basic Concepts of Kafka Consumers?	104
04c: How Do You Write the Code for a Basic Kafka Consumer?	111
Lab: Basic Kafka Consumer	123
05: Groups, Consumers, and Partitions in Practice	124
05a: How Do Groups Distribute Workload Across Partitions?	126
05b: How Does Kafka Manage Groups?	137
05c: How Do Consumer Offsets Work with Groups?	147
Additional Components of Kafka/CP Deployment Overview	155
06: Starting with Schemas	157
06a: Why Should You Care About Schemas?	159
06b: How Do You Write Schemas in Avro or Protobuf?	165
06c: How Do You Design Schemas that can Evolve?	176
07: Integrating with the Schema Registry	190
07a: How Do You Make Producers and Consumers Use the Schema Registry?	192
Lab: Schema Registry, Avro Producer and Consumer	207
08: Introduction to Streaming and Kafka Streams	208
08a: What Can You Do with Streaming Applications?	

08b: What is Kafka Streams?	218
08c: A Taste of the Kafka Streams DSL	229
08d: How Do You Put Together a Kafka Streams App?	238
Lab: Kafka Streams	249
09: Introduction to ksqlDB	250
09a: What Does a Kafka Streams App Look Like in ksqlDB?	252
09b: What are the Basic Ideas You Should Know about ksqlDB?	262
Lab: ksqlDB Exploration	268
09c: How Do Windows Work?	269
09d: How Do You Join Data from Different Topics, Streams, and Tables?	276
10: Starting with Kafka Connect	287
10a: What Can You Do with Kafka Connect?	289
10b: How Do You Configure Workers and Connectors?	302
10c: Deep Dive into a Connector & Finding Connectors	311
11: Applying Kafka Connect	319
Lab: Kafka Connect - Database to Kafka	321
11a: Full Solutions Involving Other Systems	322
More Advanced Kafka Development Matters	330
12: Challenges with Offsets	332
12a: How Does Compaction Affect Consumer Offsets?	

12b: What if You Want or Need to Adjust Consumer Offsets Manually?	342
Lab: Kafka Consumer - offsetsForTimes	352
13: Partitioning Considerations	353
13a: How Should You Scale Partitions and Consumers?	355
Lab: Increasing Topic Partition Count	362
13b: How Can You Create a Custom Partitioner?	363
14: Message Considerations	370
14a: How Do You Guarantee How Messages are Delivered?	373
14b: How Should You Deal with Kafka's Message Size Limit?	380
14c: How Do You Send Messages in Transactions?	385
15: Robust Development	400
15a: What Should You Think About When Testing Kafka Applications?	402
15b: How Can You Leverage Error Handling Best in Kafka Connect?	407
Conclusion	413
Appendix: Additional Problems to Solve	420
Problem A: Comparing Producers and Consumers	422
Problem B: Partitioning with Keys	423
Problem C: Groups, Consumers, and Partitions	425
Problem D: Partitioning without Keys	427
Appendix: Additional Content	

Appendix A: A Taste of Kafka Security for Developers	430
• • • • •	
Appendix B: Confluent Cloud vs. Self-Managed Kafka	435
• • • • •	
Appendix C: Developing with the REST Proxy	440
• • • • •	
Appendix D: Comparing the Java and .NET Consumer API	452
• • • • •	
Appendix E: Detailed Transactions Demo	454
• • • • •	
Appendix: Confluent Technical Fundamentals of Apache Kafka® Content	469
• • • • •	
1: Getting Started	471
• • • • •	
2: How are Messages Organized?	480
• • • • •	
3: How Do I Scale and Do More Things With My Data?	485
• • • • •	
4: What's Going On Inside Kafka?	494
• • • • •	
5: Recapping and Going Further	503
• • • • •	

Introduction



CONFLUENT Global Education

Class Logistics and Overview

Copyright & Trademarks

Copyright © Confluent, Inc. 2014-2022. [Privacy Policy](#) | [Terms & Conditions](#).

Apache, Apache Kafka, Kafka, and the Kafka logo are trademarks of the
[Apache Software Foundation](#)

All other trademarks, product names, and company names or logos cited herein are the property of their respective owners.

Prerequisite

This course requires a working knowledge of the Apache Kafka architecture.

New to Kafka? Need a refresher?

Sign up for free ***Confluent Fundamentals for Apache Kafka*** course at
<https://confluent.io/training>

Agenda



This course consists of these major parts:

- 1. Core Kafka Development**
 - a. Bridging from Fundamentals
 - b. Producers
 - c. Consumers
- 2. Other Components of a Kafka Deployment**
 - a. Schema management
 - b. A taste of stream processing
 - c. Integrating with other systems
- 3. Additional Challenges in Core Kafka Components**
 - a. Advanced matters
 - b. Design decisions

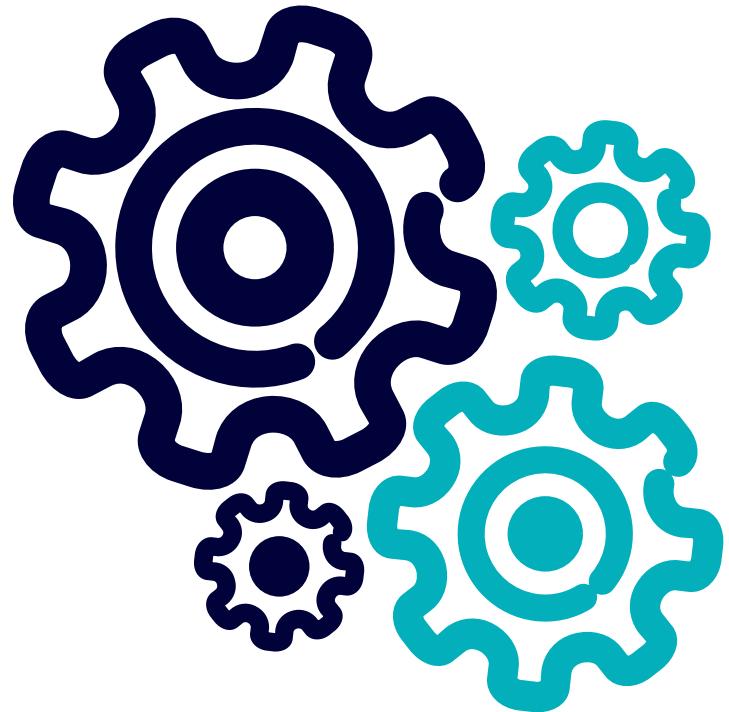
Course Objectives

Upon completion of this course, you should be able to:

- Write Producers and Consumers to send data to and read data from Apache Kafka
- Create schemas, describe schema evolution, and integrate with Confluent Schema Registry
- Integrate Kafka with external systems using Kafka Connect
- Write streaming applications with Kafka Streams & ksqlDB
- Describe common issues faced by Kafka developers and some ways to troubleshoot them
- Make design decisions about acks, keys, partitions, batching, replication, and retention policies

Throughout the course, Hands-On Exercises will reinforce the topics being discussed.

Class Logistics



- Timing
 - Start and end times
 - Can I come in early/stay late?
 - Breaks
 - Lunch
- Physical Class Concerns
 - Restrooms
 - Wi-Fi and other information
 - Emergency procedures
 - Don't leave belongings unattended



No recording, please!

How to get the courseware?



1. Register at **training.confluent.io**
2. Verify your email
3. Log in to **training.confluent.io** and enter your **license activation key**
4. Go to the **Classes** dashboard and select your class

Introductions



- About you:
 - What is your name, your company, and your role?
 - Where are you located (city, timezone)?
 - What is your experience with Kafka?
 - Which other Confluent courses have you attended, if any?
 - Optional talking points:
 - What are some other distributed systems you like to work with?
 - What technology most excited you early in your life?
 - Anything else you want to share?
- About your instructor

Fundamentals Review

Discussion

Question Set 1 [6 mins]

Determine if each statement is true or false and why:

1. All messages in a topic are on the same broker.
2. All messages in a partition are on the same broker.
3. All messages that have the same key will be on the same broker.
4. The more partitions a topic has, the better.

Question Set 2 [3 mins]

Determine the best answer to each question.

1. What are the roles of a producer and a consumer?
2. How is it decided which messages consumers read?
3. Who initiates the reading of messages: consumers or the Kafka cluster?

Discussion, Cont'd.

Question 3 [1 min]

Suppose there is a message in our Kafka cluster about my breakfast purchase of \$12.73. Consumer c_0 has consumed it to process the charge. Could consumer c_7 consume this same message this afternoon?

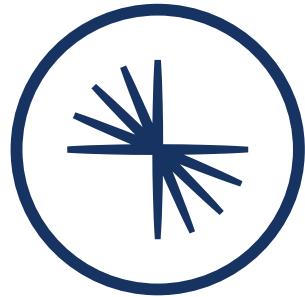
Question 4 [2 mins]

Kafka has a transactions API. When we know that all messages that are part of a transaction successfully made it to the cluster, we want to tag those messages as "good." When we know that not all messages in a transaction made it, we want to tag those messages that did make it as "bad." Kafka uses markers in the logs, that are effectively new messages written after existing messages to do this. Why not just put something in the metadata? Why not delete "bad" messages?

Instructor-Led Review

Some time is allocated here for an instructor-led review/Q&A on prerequisite concepts from Fundamentals.

Core Overview



CONFIDENT
Global Education

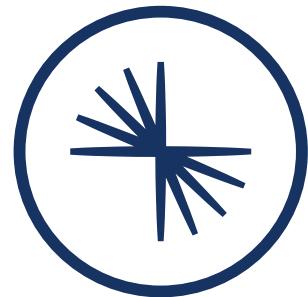
Agenda



This is a branch of our developer content on core developer concepts. It is broken down into the following modules:

1. Introductory Concepts
2. Starting with Producers
3. Preparing Producers for Practical Uses
4. Starting with Consumers
5. Groups, Consumers, and Partitions in Practice

01: Introductory Concepts



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. How Can You Connect to a Cluster?
- b. How Do You Control How Kafka Retains Messages?
- c. How Can You Leverage Replication?

Where this fits in:

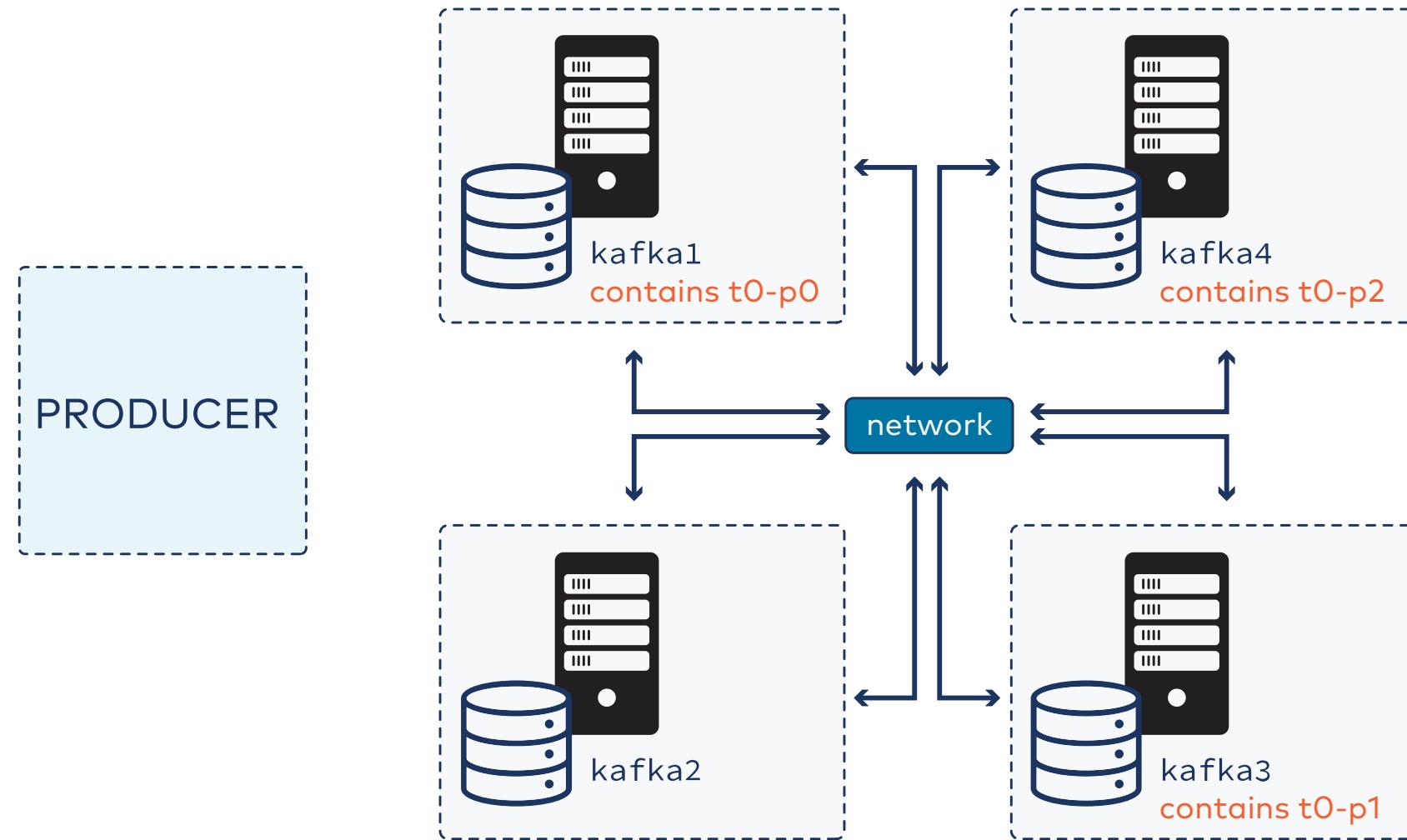
- Hard Prerequisite: Fundamentals Course
- Recommended Follow-Up: Starting with Producers

01a: How Can You Connect to a Cluster?

Description

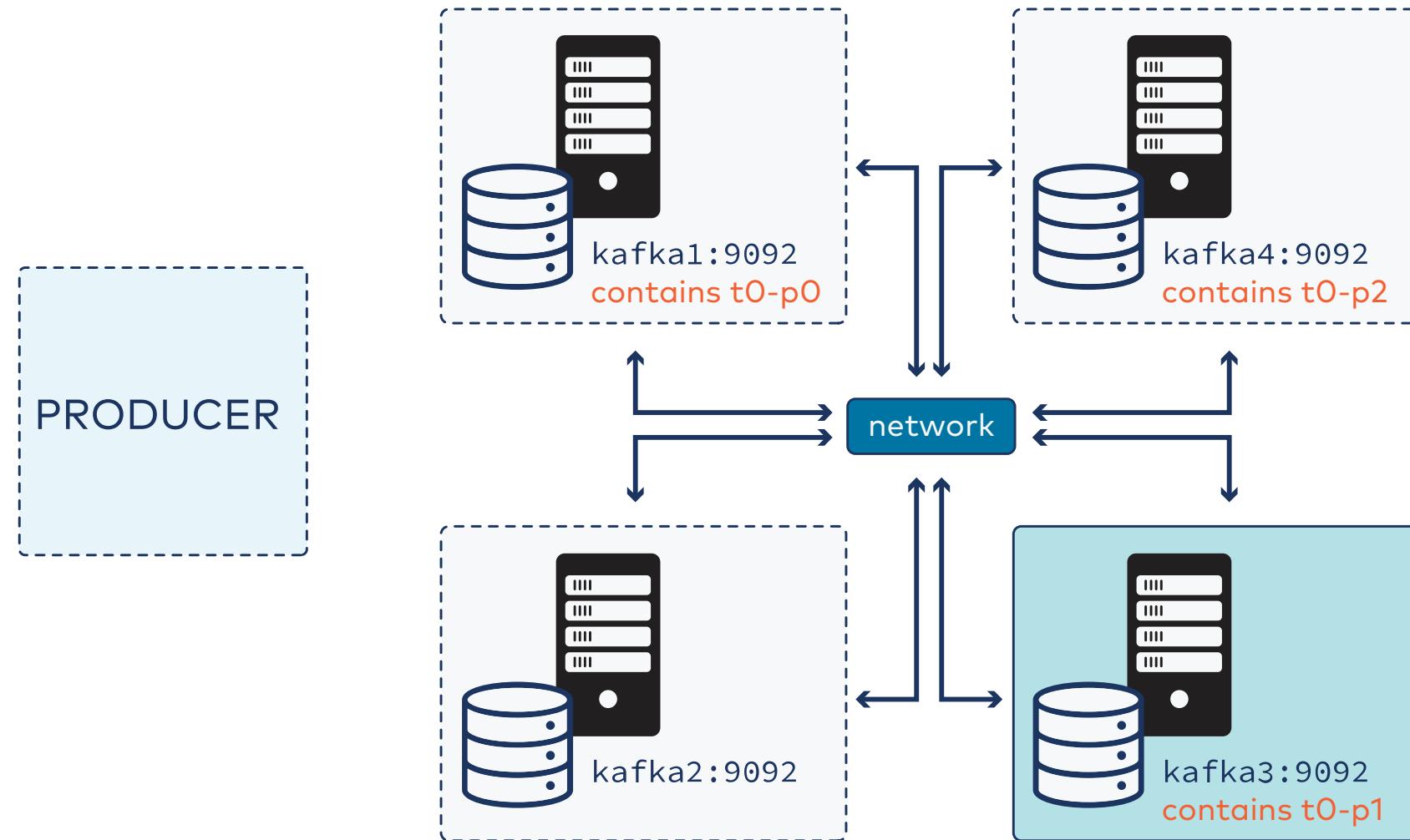
How brokers are interconnected and how this allows us to leverage bootstrap servers.
Best practice for bootstrap servers and examples of how to configure for various clients
and in a CLI example as well.

View of a Cluster and Producer



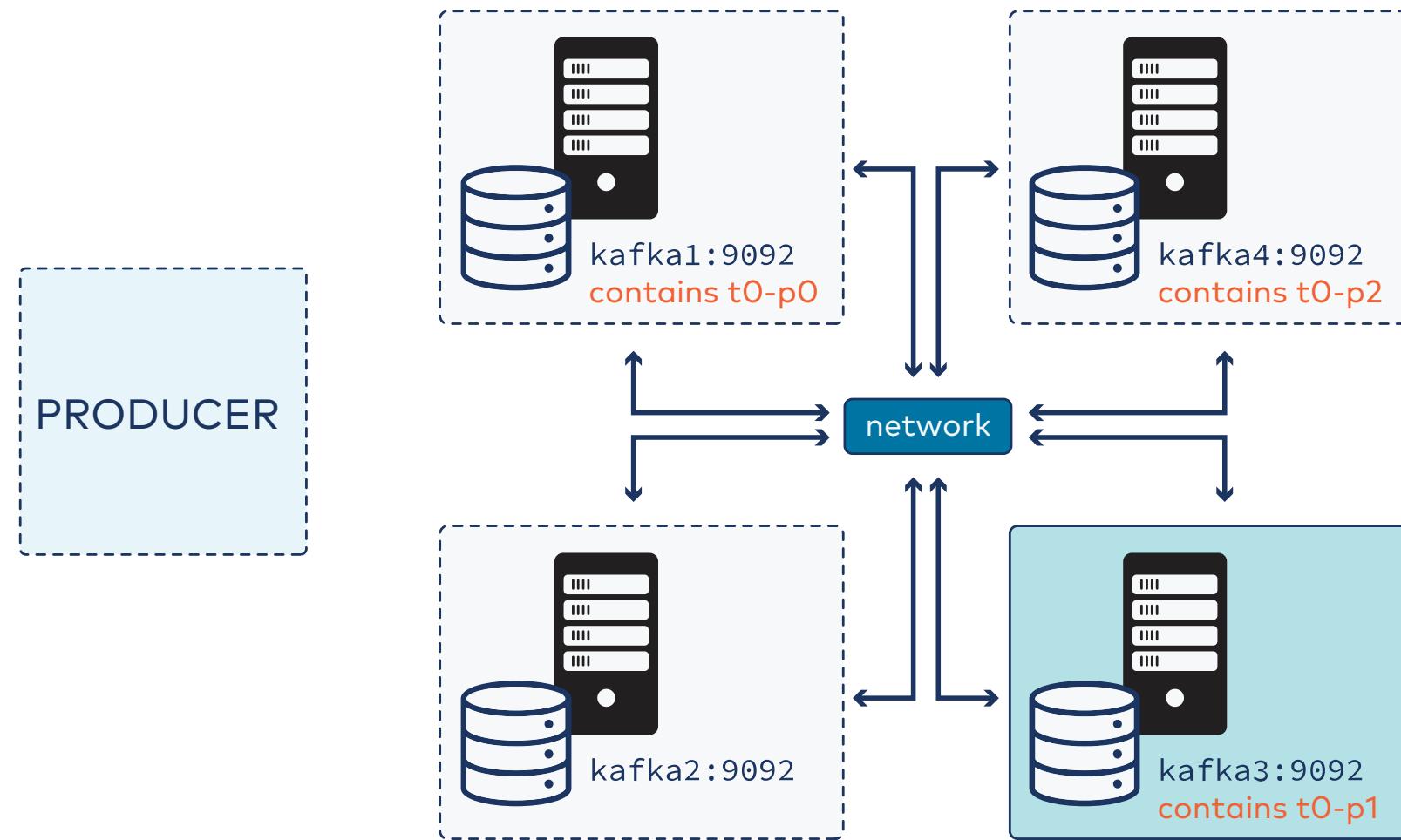
- Producer wants to send new message with key 10
- Producer chooses partition (how?)

Identifying Brokers



- Brokers identified by `host:port` pairs
- Which broker do we write our new message to?

Bootstrap Server



- Clients, like producers, need to specify a **bootstrap server**
- This is the **initial** connection to the cluster and all you need to specify in your code, configuration

Bootstrap Servers in Practice (1)

Specify bootstrap server in CLI commands, e.g.,

```
kafka-topics  
  --bootstrap-server kafka:9092  
  --create  
  --partitions 1  
  --replication-factor 1  
  --topic testing
```

Specify bootstrap server in client code, e.g.,

```
props.put("bootstrap.servers", "kafka:9092");
```

Specify bootstrap server in a properties file, e.g. for a Connect worker, e.g.,

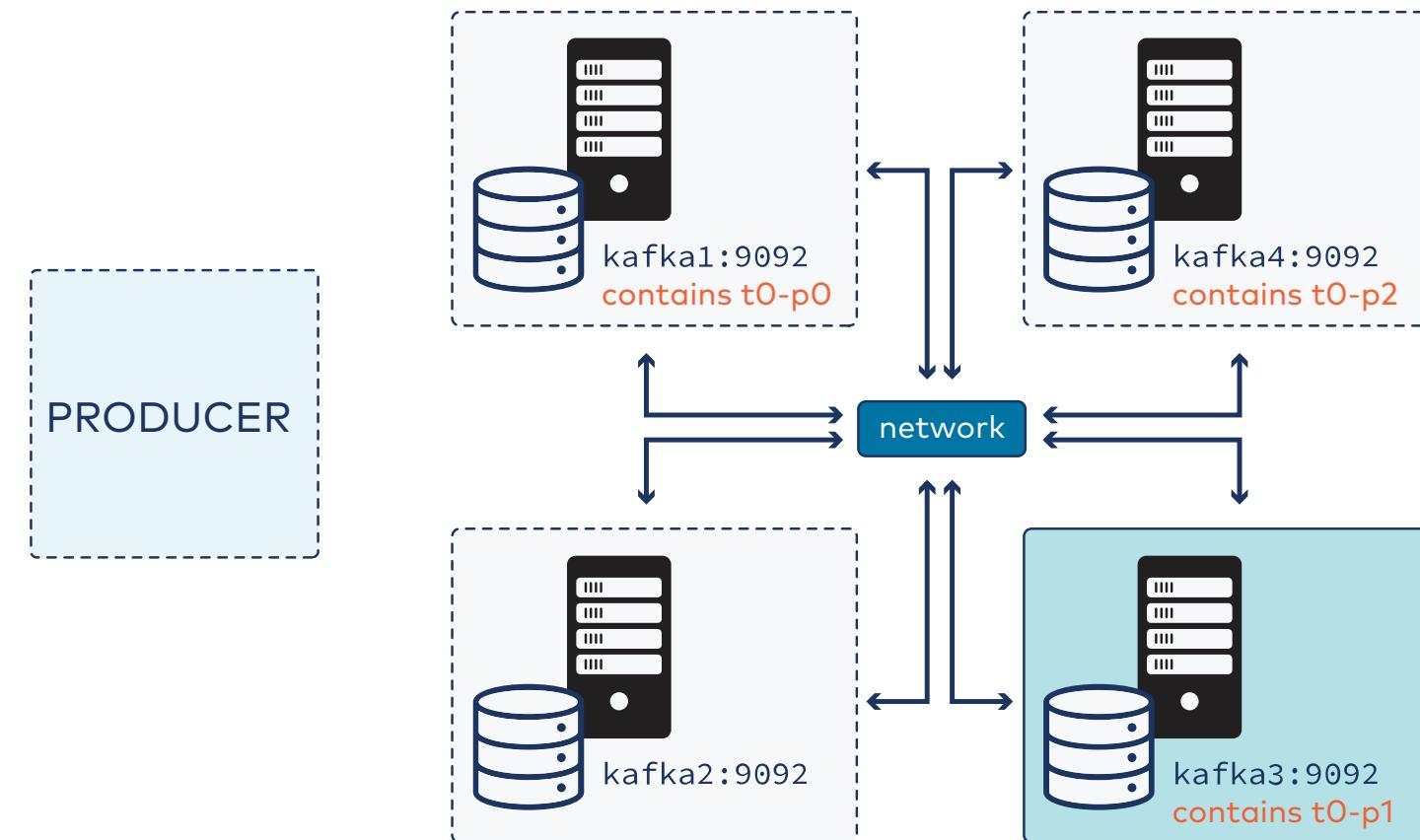
```
bootstrap.servers = kafka:9092
```

Bootstrap Servers in Practice (2)

In practice, specify a comma-separated list of bootstrap servers:

```
props.put("bootstrap.servers", "kafka1:9092, kafka2:9092, kafka3:9092");
```

Why?



01b: How Do You Control How Kafka Retains Messages?

Description

Review of the basics of the two retention policies with concrete examples. Active vs. inactive segments and how they affect retention.

Retention Policies

- `delete`
- `compact`
- `delete, compact`

Deletion Retention Policy

Delete segments when they get **too old**:

- **Too old** means newest message is older than `retention.ms`
 - Default: 7 days

Delete oldest segments when **partition gets too large**:

- `retention.bytes`
 - Default: -1 (unlimited)

Scenarios

Each picture shows the **age in days** of each message.

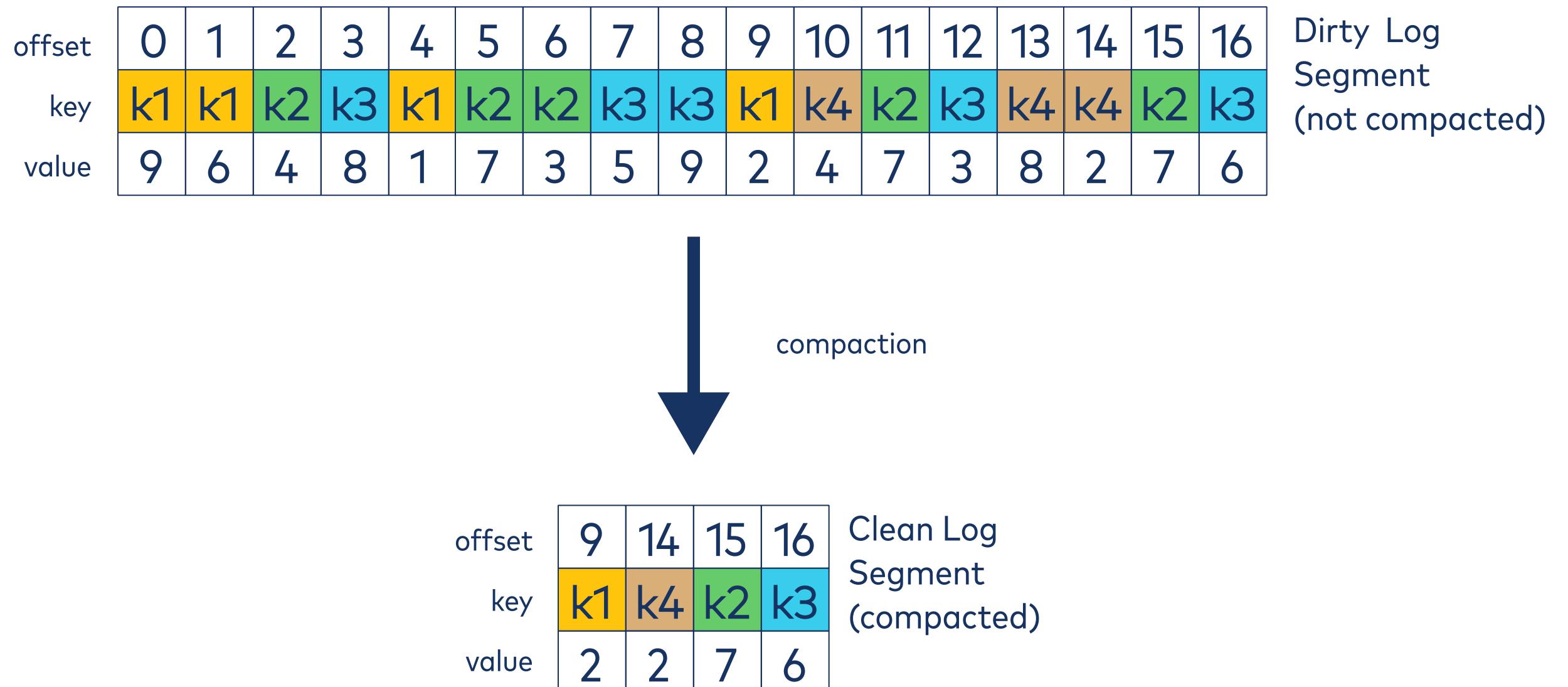
What will be left after deletion runs in each case?

Example 1	Example 2																				
<table><tr><td>12</td><td>10</td><td>9</td><td>8</td></tr><tr><td>4</td><td>3</td><td></td><td></td></tr></table>	12	10	9	8	4	3			<table><tr><td>22</td><td>16</td><td>13</td><td>12</td></tr><tr><td>10</td><td>9</td><td>5</td><td></td></tr><tr><td>5</td><td>4</td><td>3</td><td>3</td></tr></table>	22	16	13	12	10	9	5		5	4	3	3
12	10	9	8																		
4	3																				
22	16	13	12																		
10	9	5																			
5	4	3	3																		
segment 1 segment 2	segment 1 segment 2 segment 3																				

Compaction

- Keep only the latest value per key
- When is this useful?

Compaction Example



Active and Inactive Segments

- Messages are written to the **active** segment of logs
- Active segment **rolls** to become inactive
- Consumers read from all segments



Retention policies do not alter the active segment

Clean and Dirty Segments

One more distinction about log segments that applies to compaction:

- Log segments that have been compacted are called **clean** segments
- Log segments that have not been compacted are called **dirty** segments

Activity: Compacting an Uncompacted Log



Consider the log shown. Sketch out what the log will look like after compaction.

offset	0	1	2	3	4	5	6	7	8	9	10
key	y	j	y	x	k	z	x	q	x	w	a
inactive segment 1			inactive segment 2			inactive segment 3			active segment		



Note that colors denote *segments* here, not keys as in a prior example.

01c: How Can You Leverage Replication?

Description

Review of leaders vs. followers. Replication factor. How messages get from leaders to followers and config. ISRs. Leader failover / leader election.

Review: Basics of Replication

- Ensure high availability of data with backup copies

- Replicas:

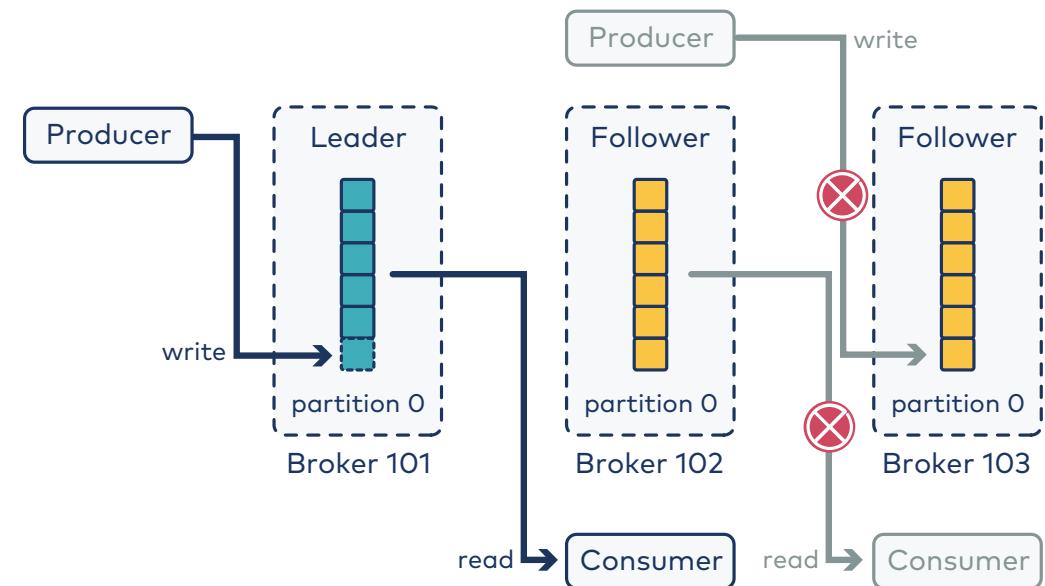
Leader

Clients write to and read from, always one leader

Follower

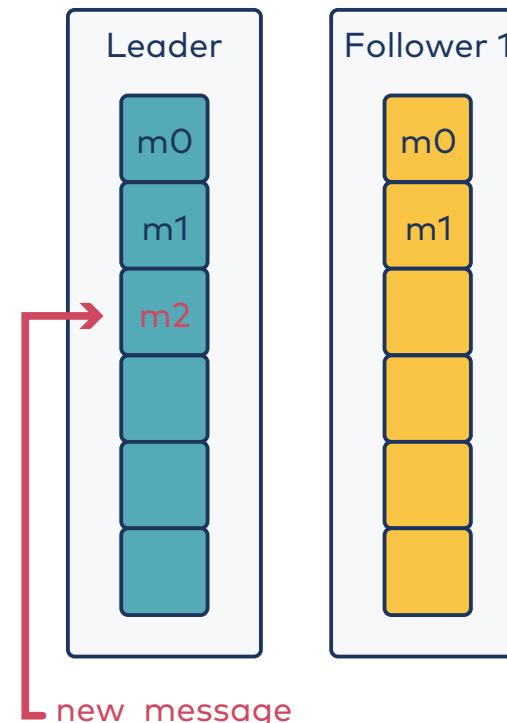
Backup copies, keep up with leader, generally multiple followers

- Topic setting `replication.factor`

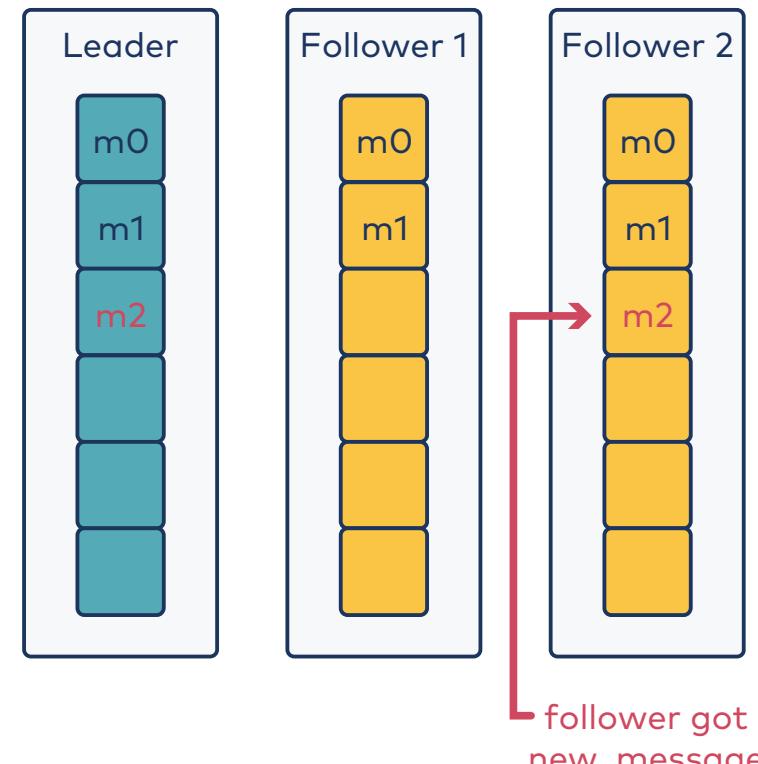


"Follow the Leader"

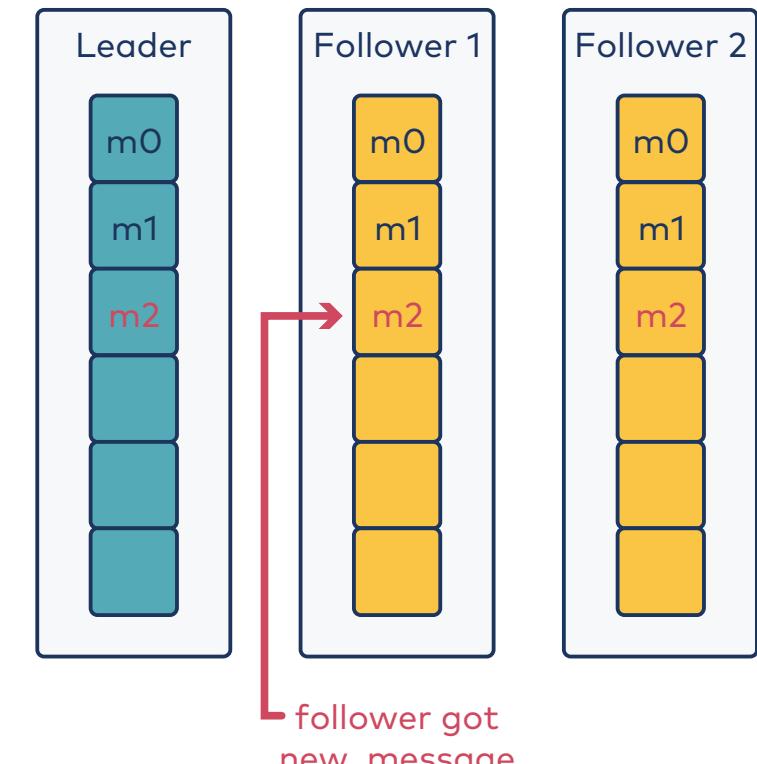
Step 1



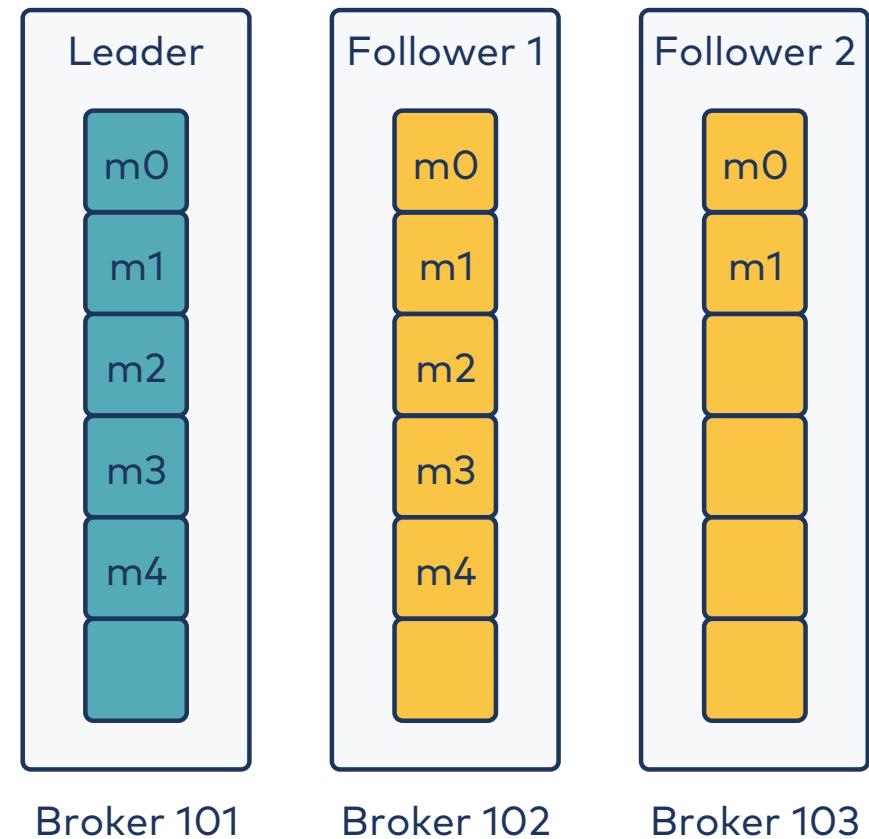
Step 2



Step 3



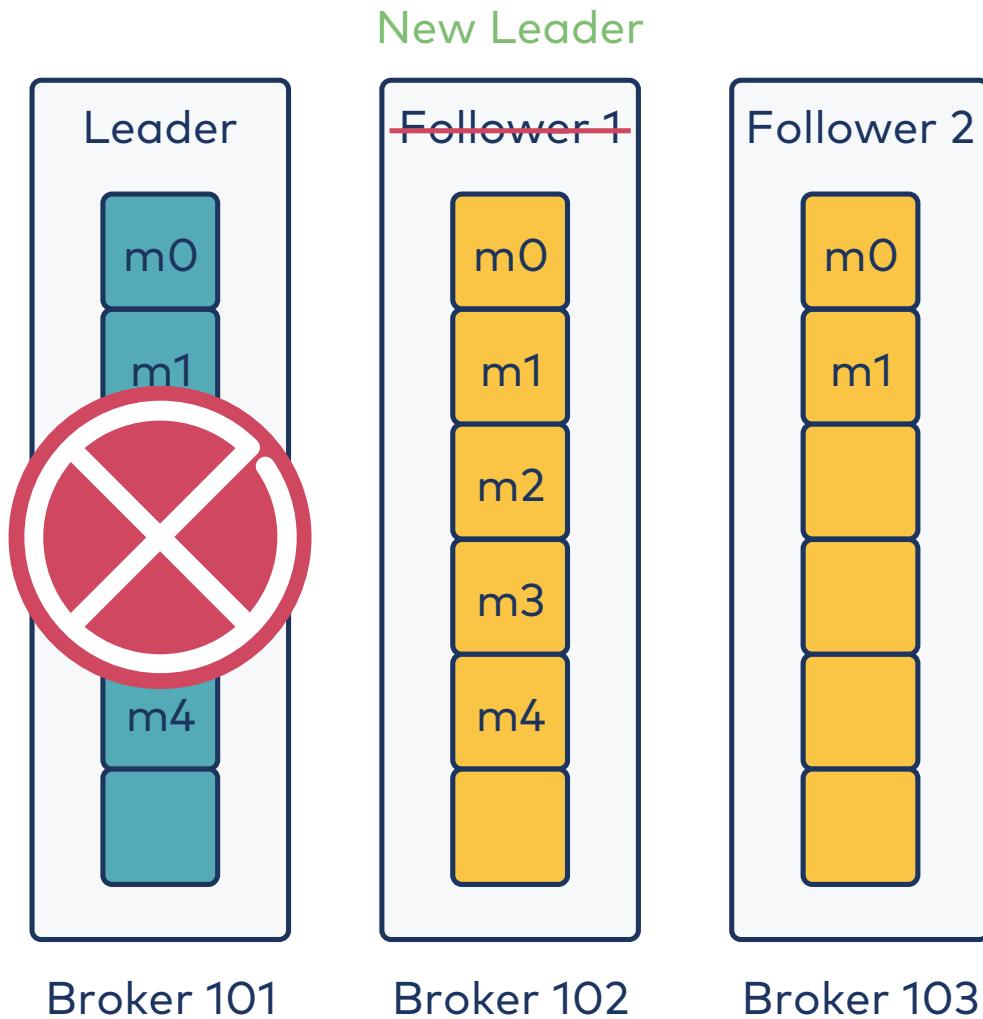
But...



Observe:

- Follower 1 (on Broker 102) is an **in-sync replica (ISR)**
- Follower 2 (on Broker 103) is **not**

Leader Failover



Question: Would either choice of follower have been equally good to replace the leader that had died?

How Does Kafka Choose Leaders?

- Leader election happens automatically
- Kafka will generally choose an in-sync follower to become leader
- Leader election does not happen in parallel
- Background processes manage balance of leadership

Activity: Exploring Replica Placement & Replication Behavior



Say we have 5 brokers - b_0, b_1, \dots, b_4 .

Say we have a replication factor of 4.

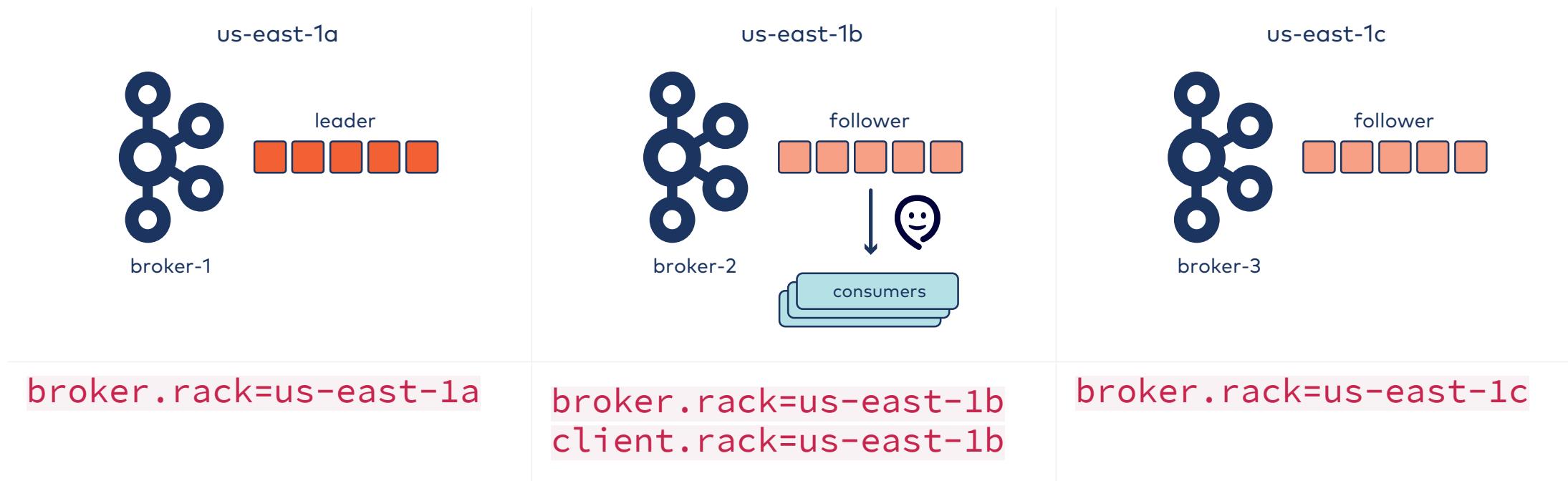
1. How many followers would we have?
2. Say leader is on broker b_4 .
 - a. Where could the followers be?
 - b. Where could a follower **not** be?
3. Say we have 3 successfully written messages that have been properly replicated. It's time to write the fourth message.
 - a. Where does it go?
 - b. What happens next?
4. Say broker b_4 fails. What happens? Why?



A Step Beyond

Follower Fetching

In this lesson, we told you all clients must interact with the leader. But, it is possible to configure it so consumers fetch from followers in the same AZ to reduce costs...



All brokers: `replica.selector.class=org.apache.kafka.common.replica.RackAwareReplicaSelector`

QUESTION: What is the tradeoff?

Lab: Introduction

Please work on **Lab 1a: Introduction**

Refer to the Exercise Guide



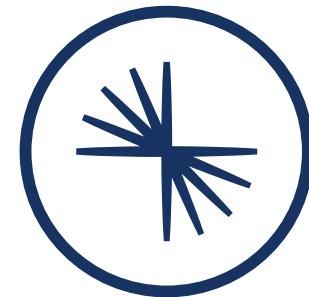
Lab: Using Kafka's Command-Line Tools

Please work on **Lab 1b: Using Kafka's Command-Line Tools**

Refer to the Exercise Guide



02: Starting with Producers



CONFLUENT
Global Education

Module Overview



This module contains two lessons:

- a. What are the Basic Concepts of Kafka Producers?
- b. How Do You Write the Code for a Basic Kafka Producer?

Where this fits in:

- Hard Prerequisite: Fundamentals Course
- Recommended Prerequisite Module: Introductory Concepts
- Recommended Follow-Up Module: Preparing Producers for Practical Uses

O2a: What are the Basic Concepts of Kafka Producers?

Description

Conceptually, basics of a producer. What is needed to specify a record. Objects one needs to instantiate to set up any producer. Basics of partitioning and serialization.

Specifying a Producer Record

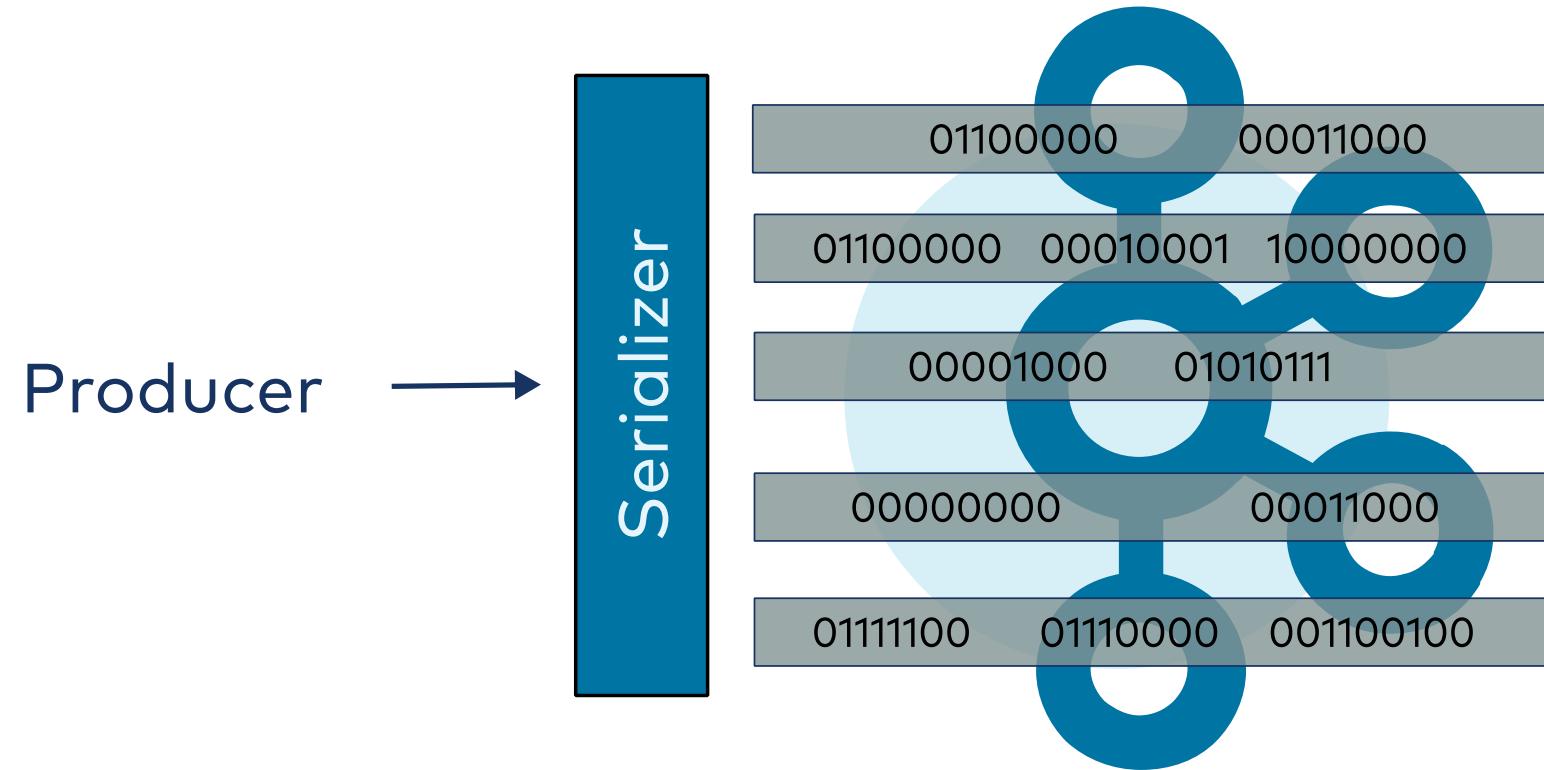
- All records
 - Topic
 - Value
- Most records
 - Key
- Some records
 - Headers - custom metadata
 - Timestamp - override default
 - Partition - force a specific partition

Core Objects for Setting Up a Producer

To specify a producer, you must instantiate each of the following:

Name	Example in One Client	Description
Configuration	<code>Properties</code>	A map of configuration settings to their values, e.g., bootstrap servers, serializers, client IDs, performance tuning settings
Producer Object	<code>KafkaProducer</code>	An abstraction of the producer itself
Record	<code>ProducerRecord</code>	An abstraction of an individual record, as per the previous slide

Serialization

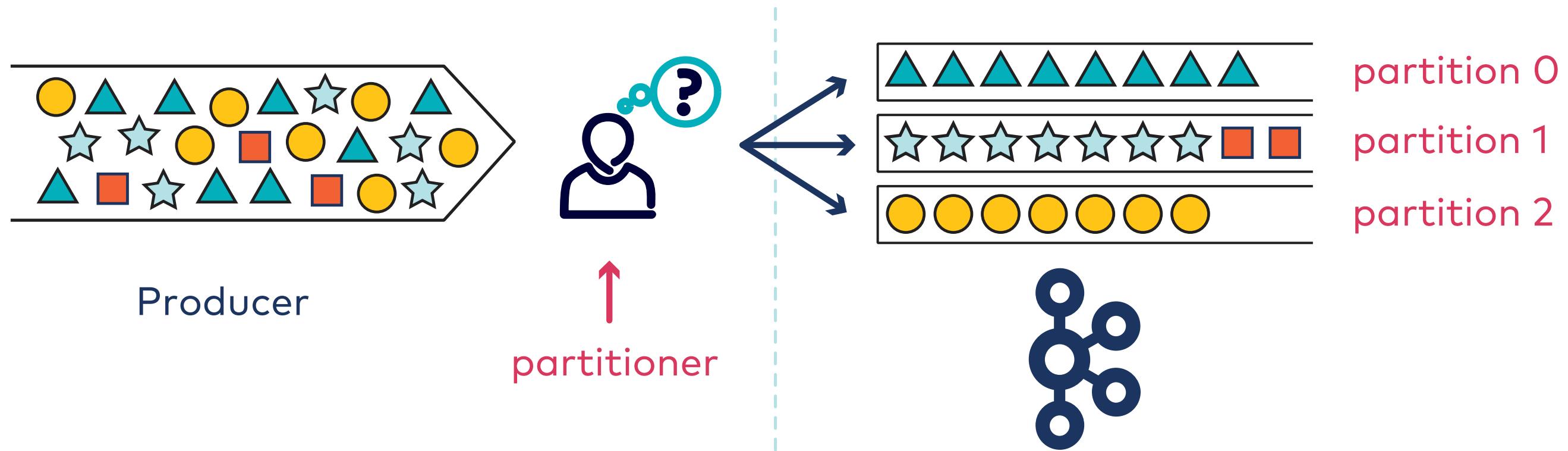


- Kafka stores byte arrays
- Need byte arrays to send data across the network
- ... need to specify **serializer**
 - Many built-in serializers

Partitioning: Default

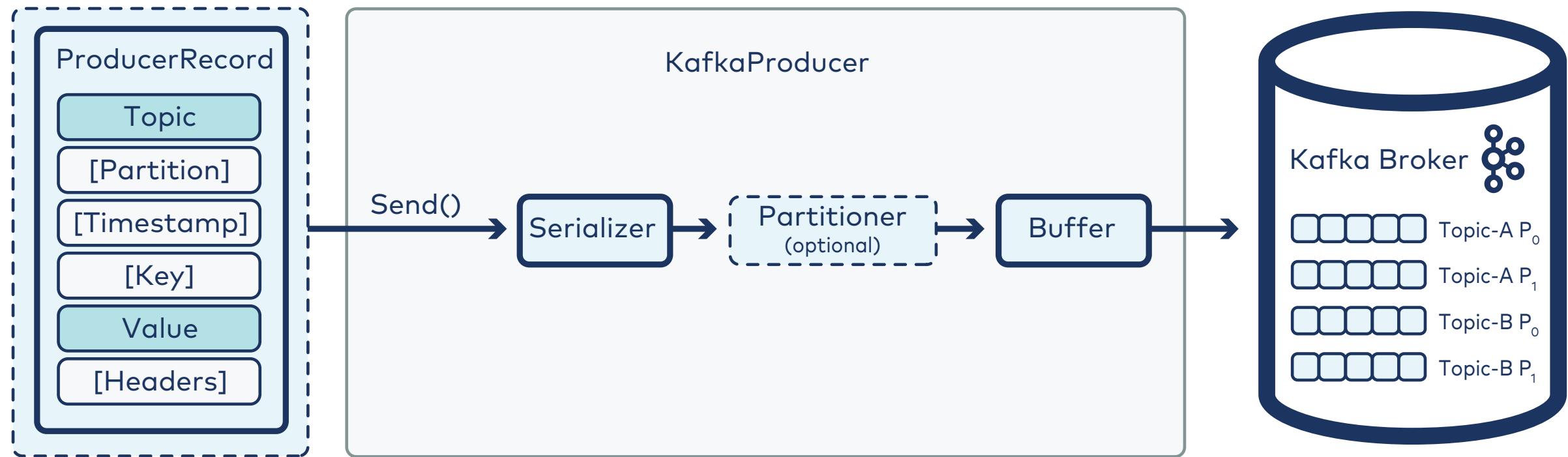
Partitions are indexed from 0 to `numberOfPartitions - 1`. When we have keyed messages...

```
partitionIndex = hash(key) % numberOfPartitions
```



A Bigger Picture

There's more to come...



Activity: First Impressions of Producers



Say you have just one minute to explain the basics of producers to a new teammate. What would you say?

02b: How Do You Write the Code for a Basic Kafka Producer?

Description

Turning a producer from the level of Producers Basic Concepts into code using the Java Client API. Quick overview of supported clients.

Basic Producer Properties

Name	Description
bootstrap.servers	Comma separated list of broker host/port pairs used to establish the initial connection to the cluster. Example: <code>kafka-1:9092, kafka-2:9092, kafka-3:9092</code>
key.serializer	Class used to serialize the key. Example: <code>StringSerializer.class</code>
value.serializer	Class used to serialize the value. Example: <code>KafkaAvroSerializer.class</code>
client.id	String to identify this producer uniquely; used in monitoring and logs. Example: <code>producer1</code>

Configuring a Producer in Java Code (1)

Instantiate and populate a **Properties** object:

```
1 Properties props;  
2  
3 props = new Properties();  
4 props.put("bootstrap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");  
5 props.put("key.serializer",  
6           "org.apache.kafka.common.serialization.StringSerializer.class");  
7 props.put("value.serializer",  
8           "org.apache.kafka.common.serialization.KafkaAvroSerializer.class");  
9 props.put("client.id", "my_first_producer");
```

Configuring a Producer in Java Code (2)

Question: We know this code works to set bootstrap servers:

```
4 props.put("bootstrap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

What about this code:

```
4 props.put("bootsrtap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

Will it compile?

Virtual Classroom Poll:



it compiles



it does not compile

Configuring a Producer in Java Code (3)

Question: We know this code works to set bootstrap servers:

```
4 props.put("bootstrap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

What about this code:

```
4 props.put("bootsrtap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

Will it work?

Virtual Classroom Poll:



it works



it does not work

Configuring a Producer in Java Code (4)

Wanted:

```
4 props.put("bootstrap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

Typo - compiles but doesn't run:

```
4 props.put("bootsrtap.servers", "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

Fix - helper classes:

```
4 props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-1:9092, kafka-2:9092, kafka-3:9092");
```

Configuring a Producer in Java Code (5)

We can also use a properties file to specify configuration:

```
1 InputStream propsFile;  
2  
3 propsFile = new FileInputStream("src/main/resources/producer.properties");  
4 props.load(propsFile);
```

Creating a Producer Object

We must create an instance of `KafkaProducer`, e.g.

```
KafkaProducer<String, MyObject> producer;
```

And we must pass the configuration `Properties` object to it during initialization:

```
producer = new KafkaProducer<>(props);
```

Creating and Sending a Record

We must create an instance of `ProducerRecord` and instantiate it with a topic, key, and value. Here's an example:

```
ProducerRecord<String, String> record;  
  
record = new ProducerRecord<String, String>("my_topic", "my_key", "my_value");
```

Then we tell the producer send it:

```
producer.send(record);
```

We may not need to name the `ProducerRecord`, so you might, succinctly, do this:

```
producer.send(new ProducerRecord<String, String>("my_topic", "my_key", "my_value"));
```

Cleaning Up

- `producer.close();`

Blocks until all previously sent requests complete

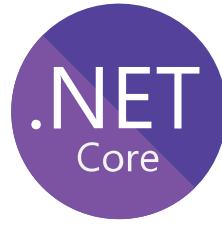
- `producer.close(Duration.ofMillis(...));`

Waits until complete or given timeout expires

Other `Duration` units are allowed

More Clients...

Here are the Kafka clients supported by Confluent:

JVM	librdkafka (C library)
   	   



For other languages, consider the REST Proxy.

Hands-On Exercise Environment

- **Visual Studio Code**—development environment
- Exercises available in:
 - Java
 - Python
 - C#
- Front end webserver is written with a community NodeJS client

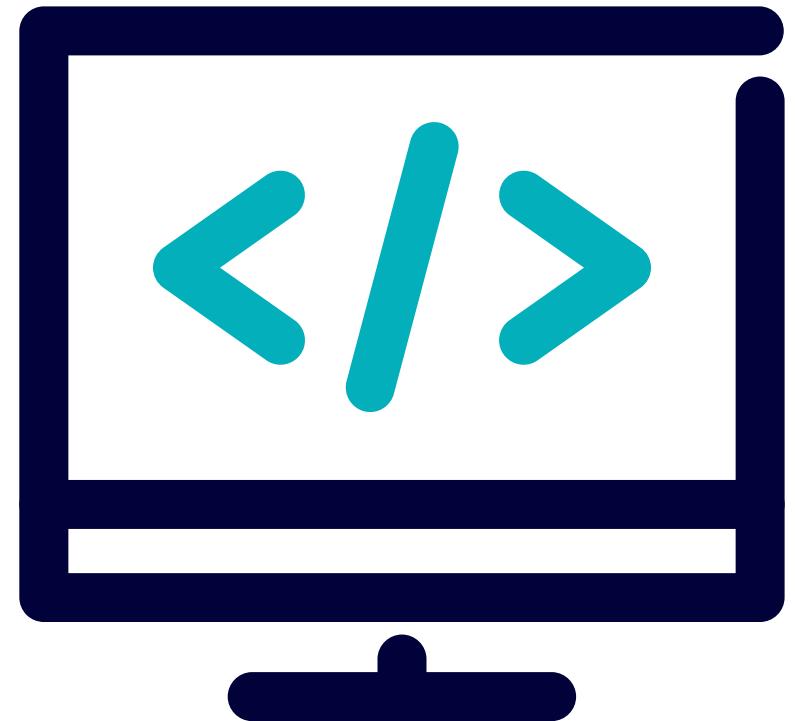


You are encouraged to try the exercises in multiple languages!

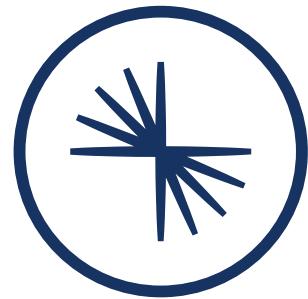
Lab: Basic Kafka Producer

Please work on **Lab 2a: Basic Kafka Producer**

Refer to the Exercise Guide



03: Preparing Producers for Practical Uses



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. How Can Producers Leverage Message Batching?
- b. How Do Producers Know Brokers Received Messages?
- c. How Can a Producer React to Failed Delivery?

Where this fits in:

- Hard Prerequisite: Starting with Producers
- Recommended Follow-Up: Starting with Consumers

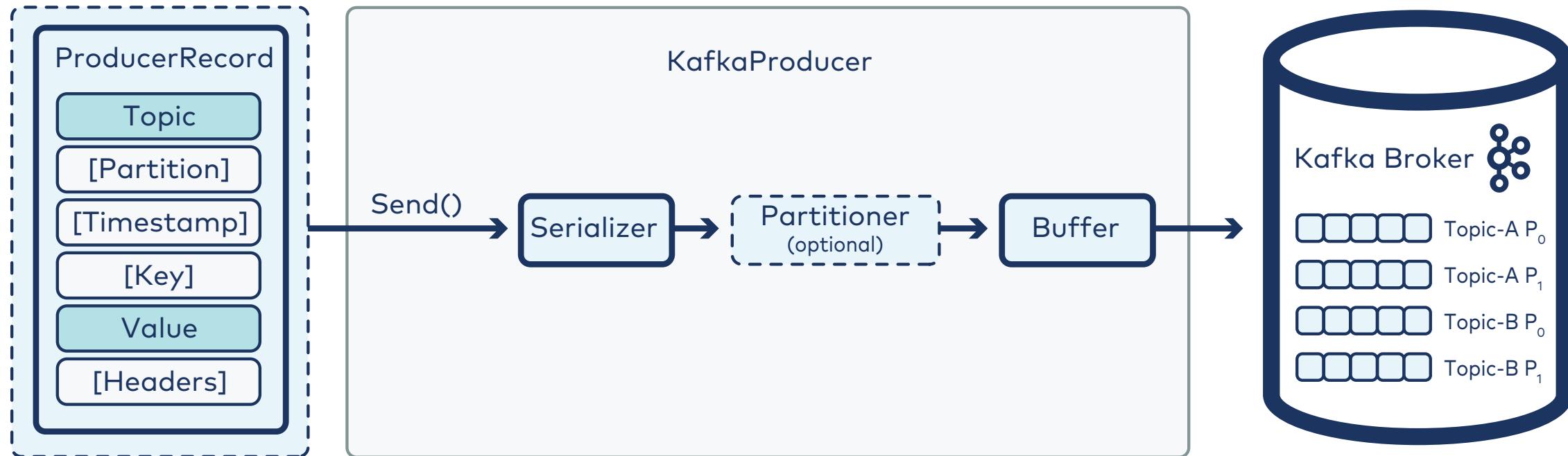
03a: How Can Producers Leverage Message Batching?

Description

The motivation for batching on producers, configuration of batching and buffers, and understanding the shared buffer and implications. Setting compression on batches.

Producers So Far...

Let's go back to this picture...



Let's now see what's going on with the buffers...

Diversion

Say it takes you 10 minutes to get to your nearest grocery store.

Say you need three things:

- ice cream
- cereal
- cheese

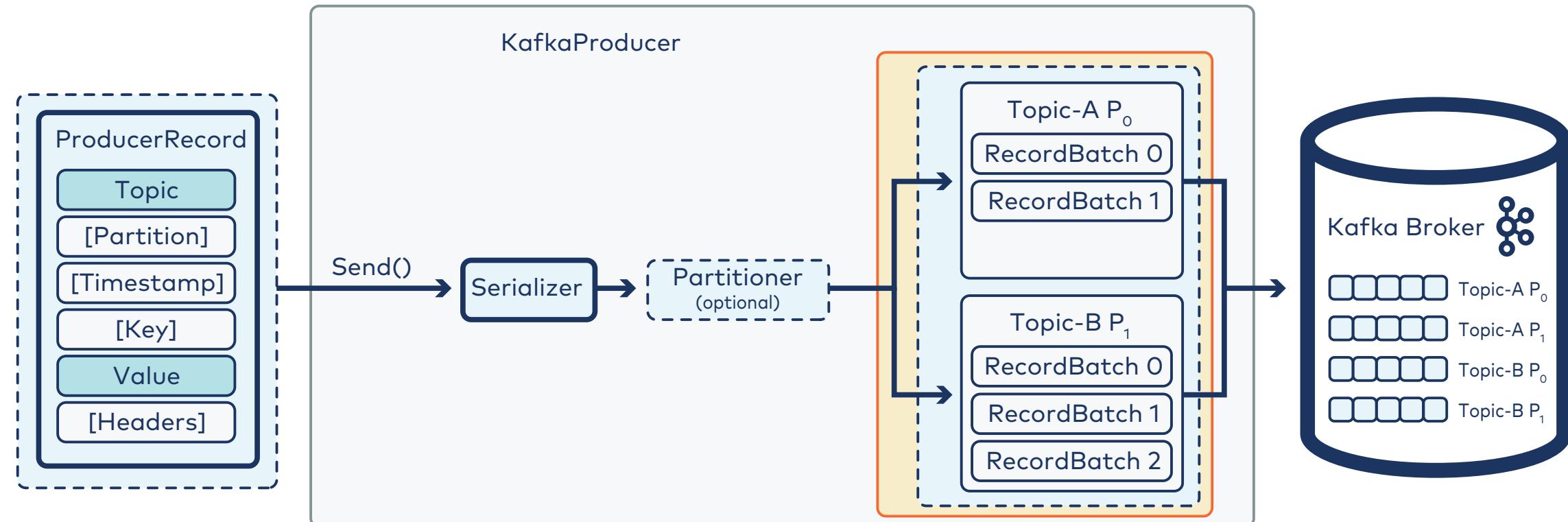
You could...

1. Go to the store, buy the ice cream, go home, and put the ice cream away.
2. Go to the store again, buy the cereal, go home, and put the cereal away.
3. Go to the store again, buy the cheese, go home, and put the cheese away.

Thoughts?



Refining Our Producer Design Picture



Messages are stored in the buffer **per topic-partition.**

Back to that Grocery Store Diversion

- Before:
 - Grocery store 10 minutes away
 - Needed ice cream, cereal, cheese
 - We agreed we could batch, i.e., get all three at once
- Clarifying constraint:
 - Walking to store
 - Carrying everything
- Now: Additional things to buy:
 - 5 cases of soda
 - Bulk pack of 10 rolls of paper towels
 - Bulk pack of 8 boxes of tissues

Now what? Same batching?

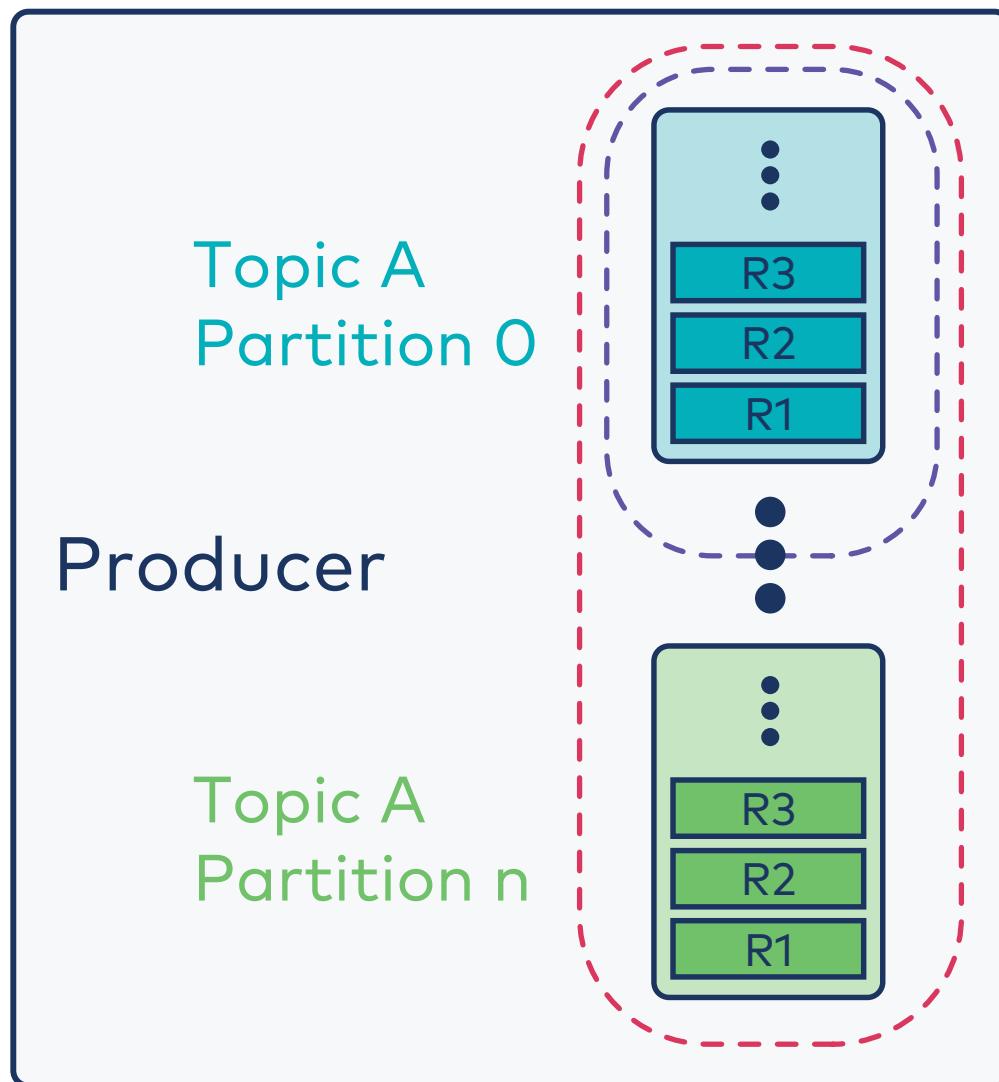
What Do You Care About?

- Throughput?
 - Want to send many messages at once
- Latency?
 - Want to have messages consumed as quickly as possible after they are produced
- We can specify
 - How big batches can get
 - How long batches can accumulate



Balance! Probably we care about both matters

Visualizing the Buffer

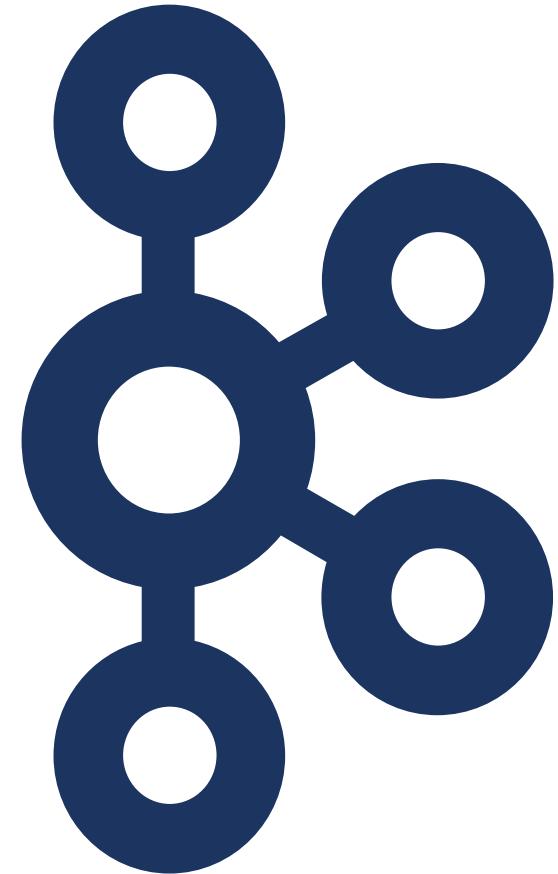


`linger.ms`

`batch.size`

flush to Kafka

`buffer.memory`



Not So Fast...

- `send()` returns when a message has been added to the buffer
- The shared buffer is of size `buffer.memory`
- What if, when we send, there is not enough room in the buffer? Are we doomed?

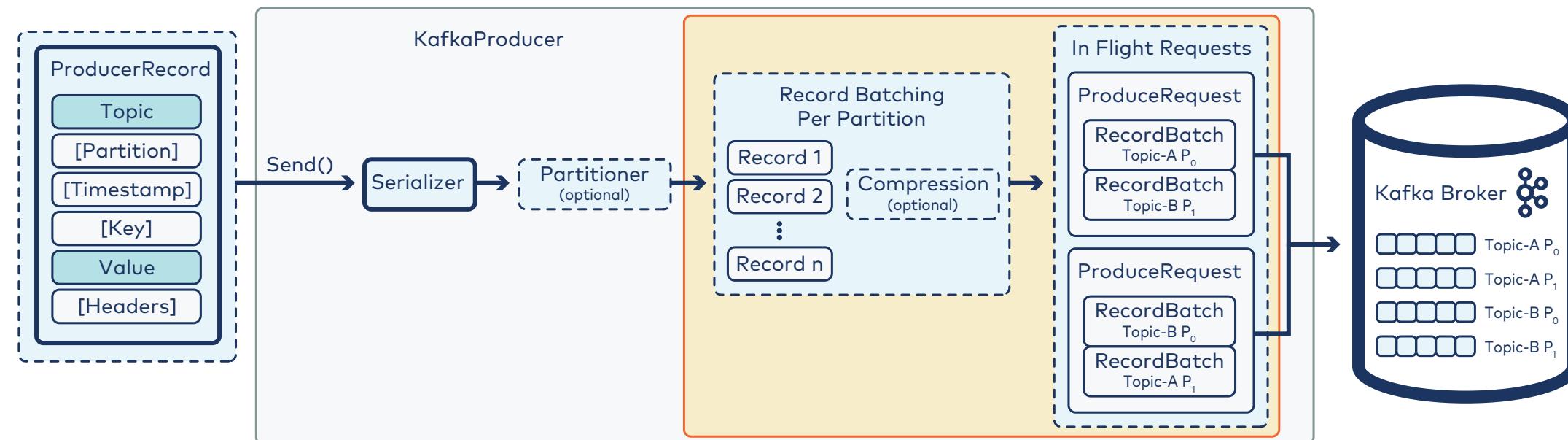
We're Okay After All! (Maybe)

- `send()` returns when a message has been added to the buffer
- The shared buffer is of size `buffer.memory`
- What if, when we send, there is not enough room in the buffer? Are we doomed?
 - **NO!**
 - Maybe a batch accumulating for some partition is big enough to meet `batch.size` or `linger.ms` is flushed and space is freed
- Config setting `max.block.ms` puts a limit on how long a producer will wait for there to be space in the buffer before a `send()` fails.

Compression 101

- It is possible to turn on compression on producers
- Records are sent as compressed batches

New picture:



Summarizing new Configurations

Name	Description	Default
batch.size	Minimum number of bytes needed to accumulate in a batch for it to be considered complete and ready to send.	16384
linger.ms	Maximum time a batch will wait to accumulate before sending.	0
buffer.memory	Maximum size of the producer's buffer, shared across all partitions.	32 MB
max.block.ms	How long producer will wait for a full buffer to have free space before failing a <code>send()</code>	1 min
compression.type	How data should be compressed. Values are <code>none</code> , <code>snappy</code> , <code>gzip</code> , <code>lz4</code> , <code>zstd</code> . Compression is performed on batches of records.	none

Activity: Applying Batching Configurations



Consider the following settings:

- `batch.size`
- `linger.ms`
- `buffer.memory`
- `max.block.ms`

Which setting(s) would be important to adjust to accommodate each of the following scenarios?

- a. You want high throughput, but don't care about latency
- b. Latency is a major goal but we don't want to forget about throughput entirely
- c. You are finding the overall buffer gets full a lot in (a)

03b: How Do Producers Know Brokers Received Messages?

Description

The various levels of producer acknowledgements and implications of each. Using callbacks in producer code to react to acks.

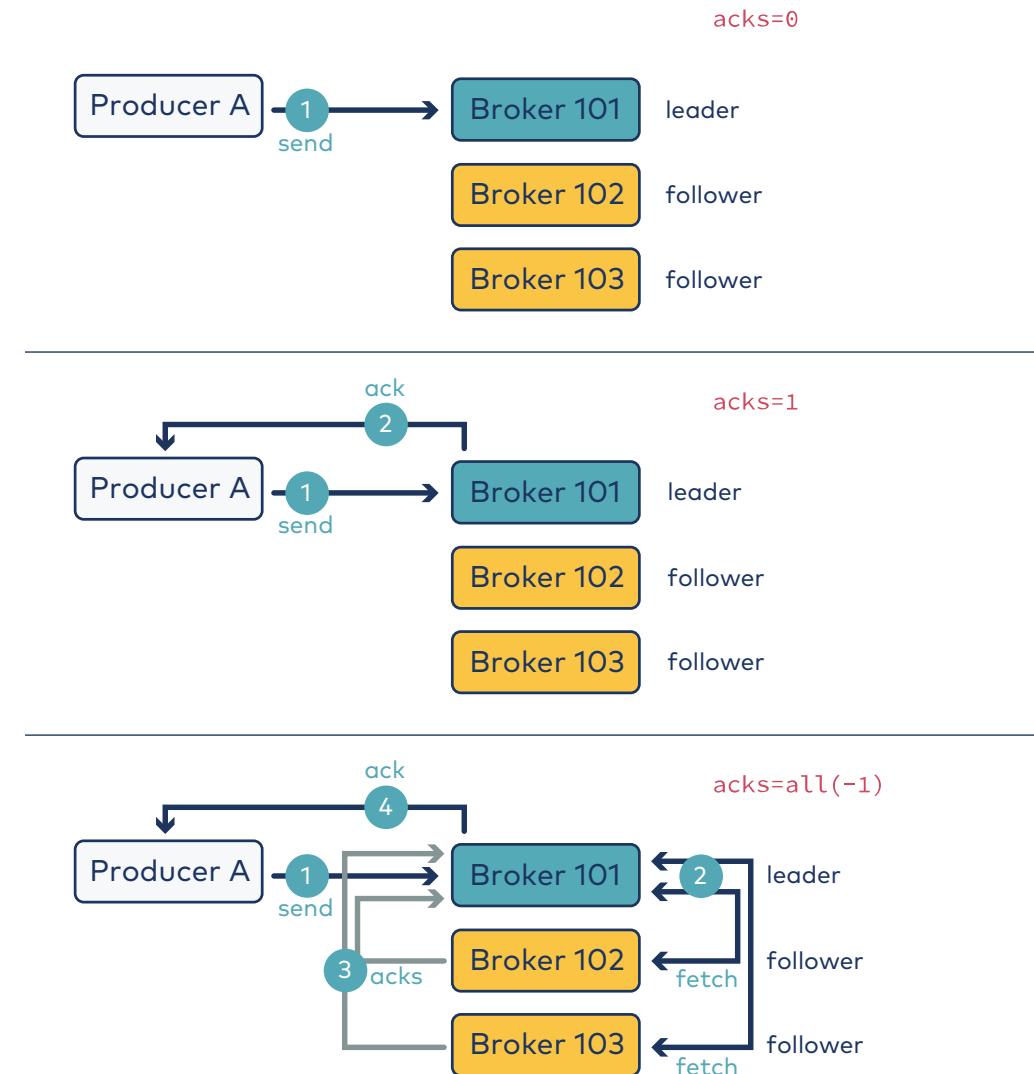
The Notion of an Acknowledgment

A producer might want to know if a request was successfully delivered to the Kafka cluster.

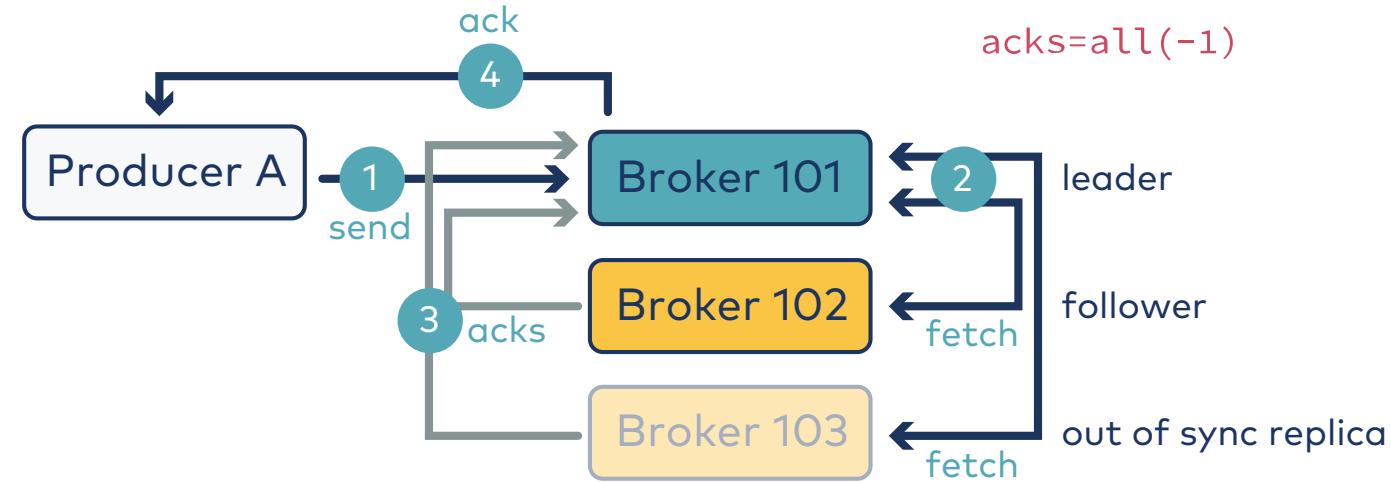
We add the following configuration setting to our list of producer properties:

Name	Description
acks	Used to determine when a write request is successful. Can be 0, 1, or all (-1). If <code>acks</code> is not zero, then the producer will retry failed requests. Default: <code>all</code>

acks - Three Cases, Ideal Performance



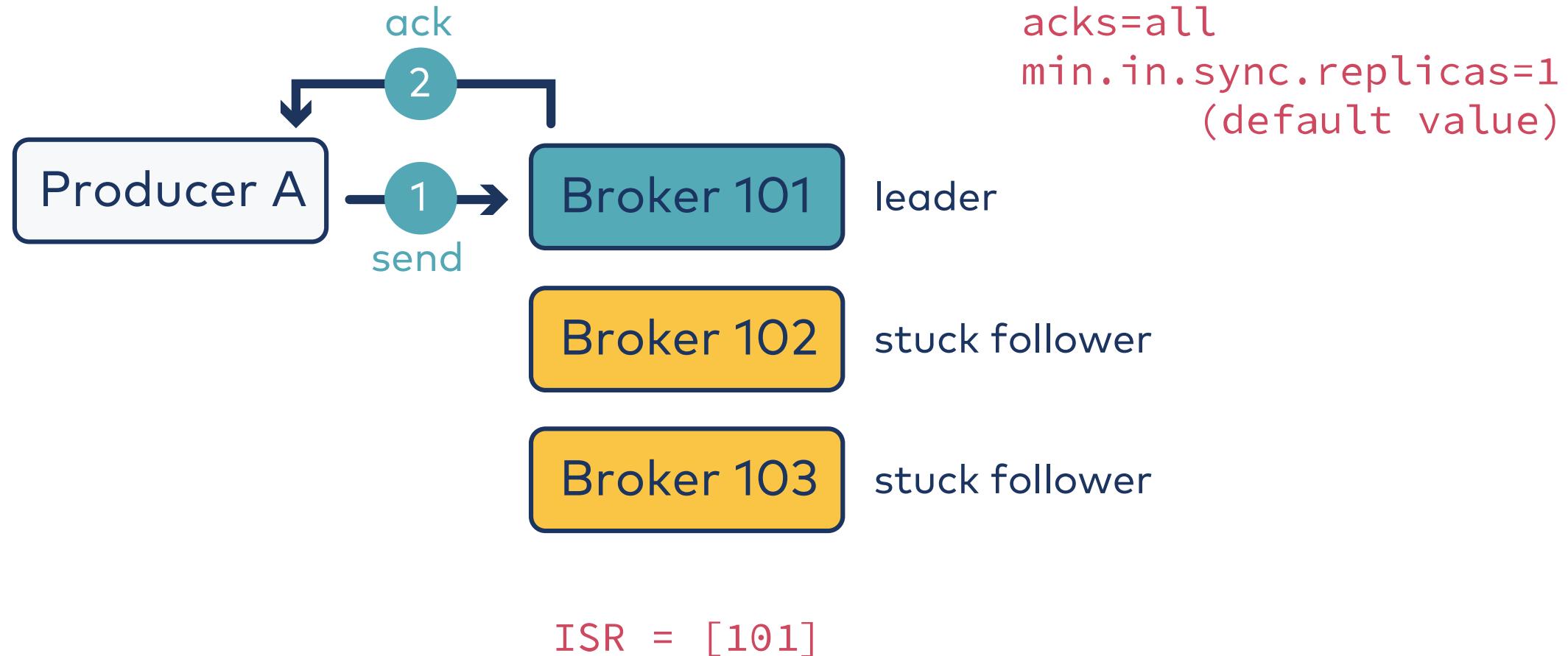
...But not all followers are in-sync...



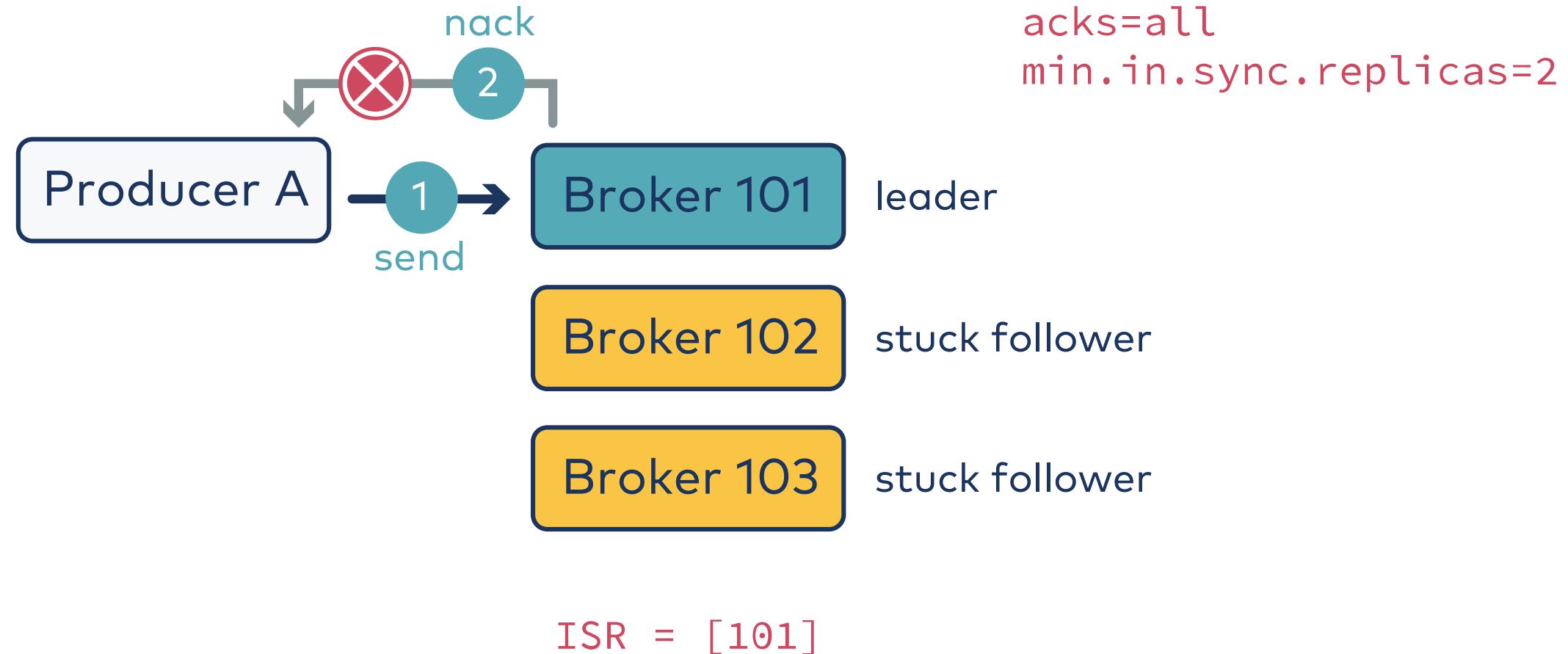
So... when the leader gets a new message and **acks=all**...

1. The leader notes which followers are **in sync** with the leader at the time it receives the message
2. Followers fetch from the leader and send acks to the leader
3. When the leader receives acks from all of the followers from (1), it sends an ack to the producer

What if We Don't Have Any In-Sync Followers?



Guaranteeing Meaningful `acks=all`



What Happens When Producers Send? [A Review]

- When we call `send()` on a producer...
 - Message is serialized
 - Message runs through partitioner (in most cases)
 - Message is written to the buffer
- `send()` call returns once the message has been written to the buffer
 - Producer does not wait to find out if Kafka brokers have received the message
 - More code, likely more `send()` calls, runs (i.e., producer does not block further execution) as a batch may get more messages after the `send()` has returned
 - Maybe we want to know if a `send()` was successful...

Callbacks

- We can provide a `Callback` as the second argument to a `send()` call
- The `Callback` is an interface
- The key method is `onCompletion`:

```
onCompletion(RecordMetadata metadata, java.lang.Exception exception) {...}
```

- Typically, `onCompletion` is implemented with a lambda expression
- Parameters of `onCompletion`:
 - Send in uninitialized objects
 - `exception` is `null` when we have success
 - Test for this first

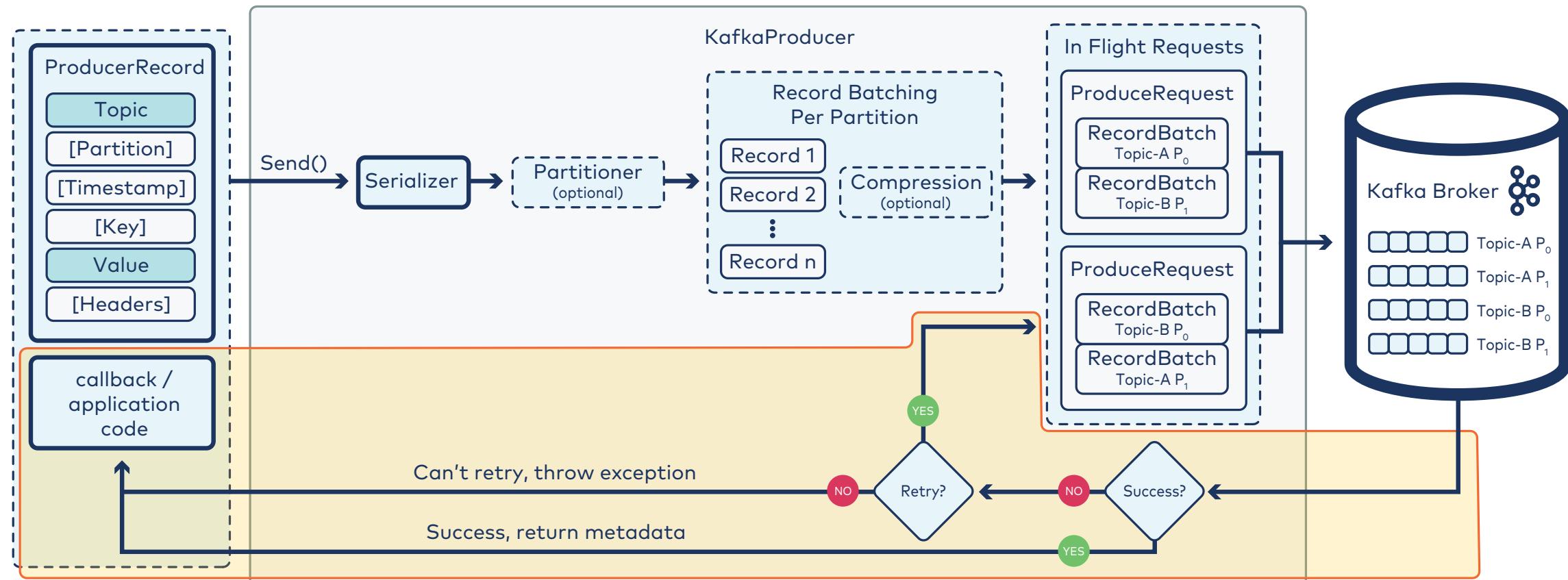


When `exception` is not `null` in the callback, metadata will contain the special -1 value for all fields except for `topicPartition`, which will be valid.

send() with Callback Example

```
1 producer.send(record, (recordMetadata, e) -> {
2     if (e != null)
3         e.printStackTrace();
4     else
5         System.out.println("Message String = " + record.value() +
6                             ", Offset = " + recordMetadata.offset());
7 }
8});
```

Refining Our Producer Design Picture



Activity: Tracing Message Writes with Acks

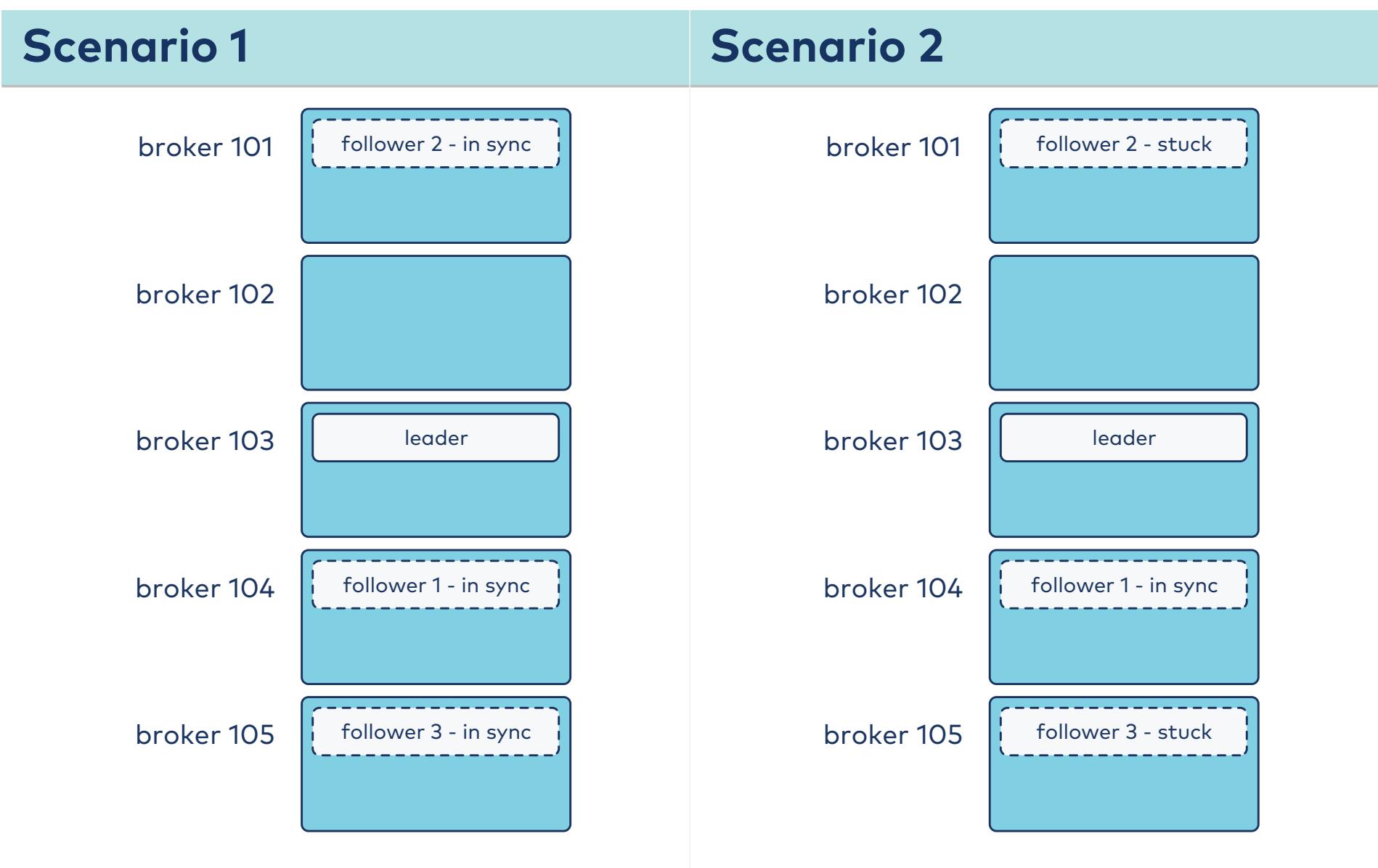


Discuss each scenario with a partner or a small group before we discuss as a class.

Suppose a producer produces a new message to the partition shown. Describe a possible sequence of what could happen from the moment the cluster receives the message until the producer receives an acknowledgment

1. when `acks = 1`
2. when `acks = all`

Activity, continued



03c: How Can a Producer React to Failed Delivery?

Description

Retrying failed messages, different components of time from send until delivery and configuration, and best practices.

Big Picture

- We can configure producers with `acks = all` or `acks = 1` to make the Kafka brokers respond to the producer know if messages make it successfully.
- What if a message fails?
 - Give up?
 - Try again?
 - How long to keep trying?

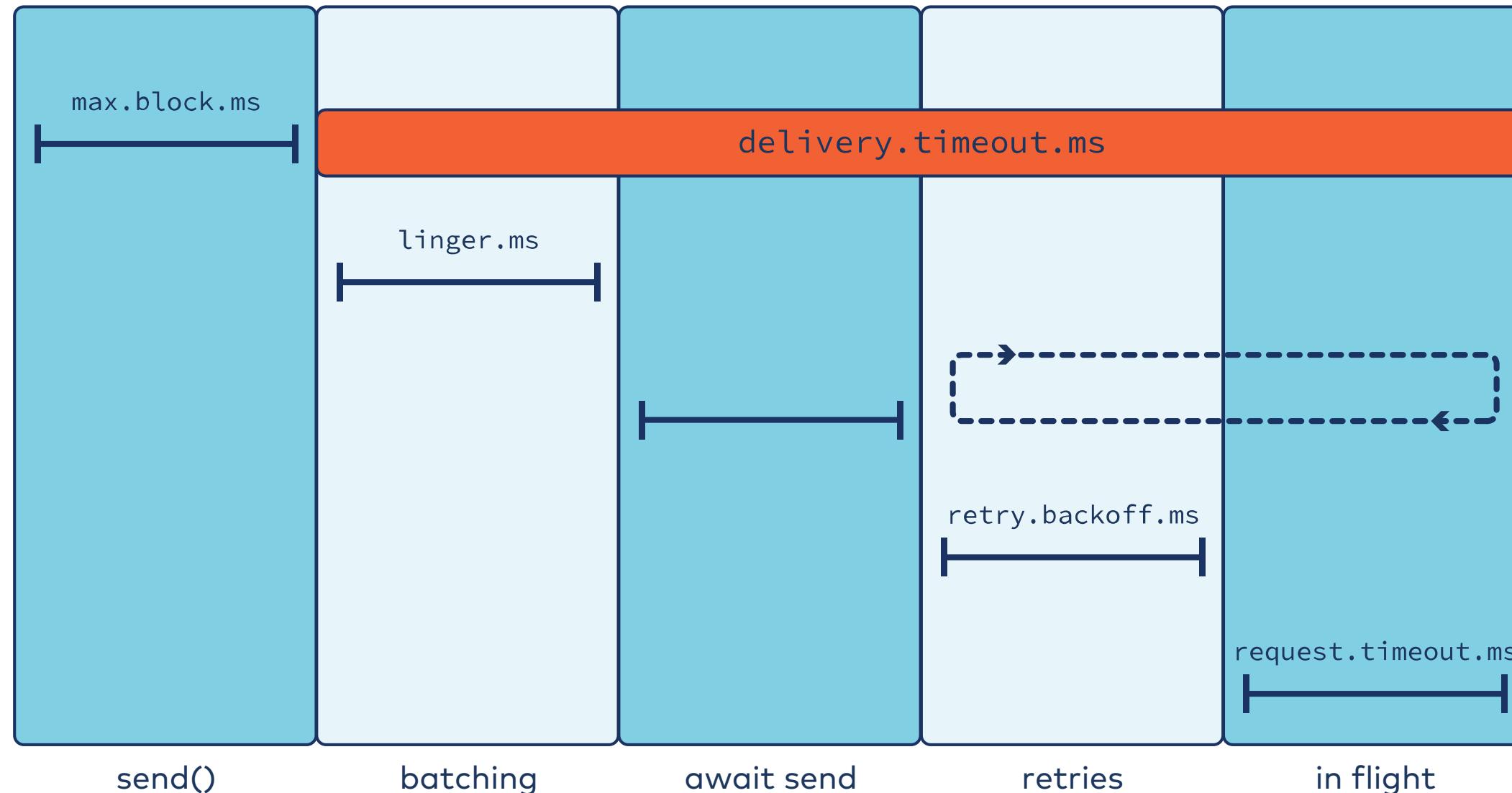
Retries and Timeouts

- We can configure a number of retries a producer has
 - Setting `retries` with default `MAX_INT`
- We can also limit how much time a producer spends waiting overall at various stages after sending
- **Best practice:** Limit delivery with timeouts, not number of retries

Some Time Limits

Name	Description	Default
<code>max.block.ms</code>	An upper bound on the time to wait for a buffer that doesn't have enough room to accept a new message to gain more space.	1 min.
<code>linger.ms</code>	Time a batch will wait to accumulate before sending.	0
<code>request.timeout.ms</code>	An upper bound on the time a producer will wait to hear acknowledgments back from the cluster.	30 sec.
<code>retry.backoff.ms</code>	How much time is added after a failed request before retrying it.	100
<code>delivery.timeout.ms</code>	An upper bound on the time to report success or failure after a call to <code>send()</code> returns. Use this to control producer retries.	2 mins.

Visualizing Those Times



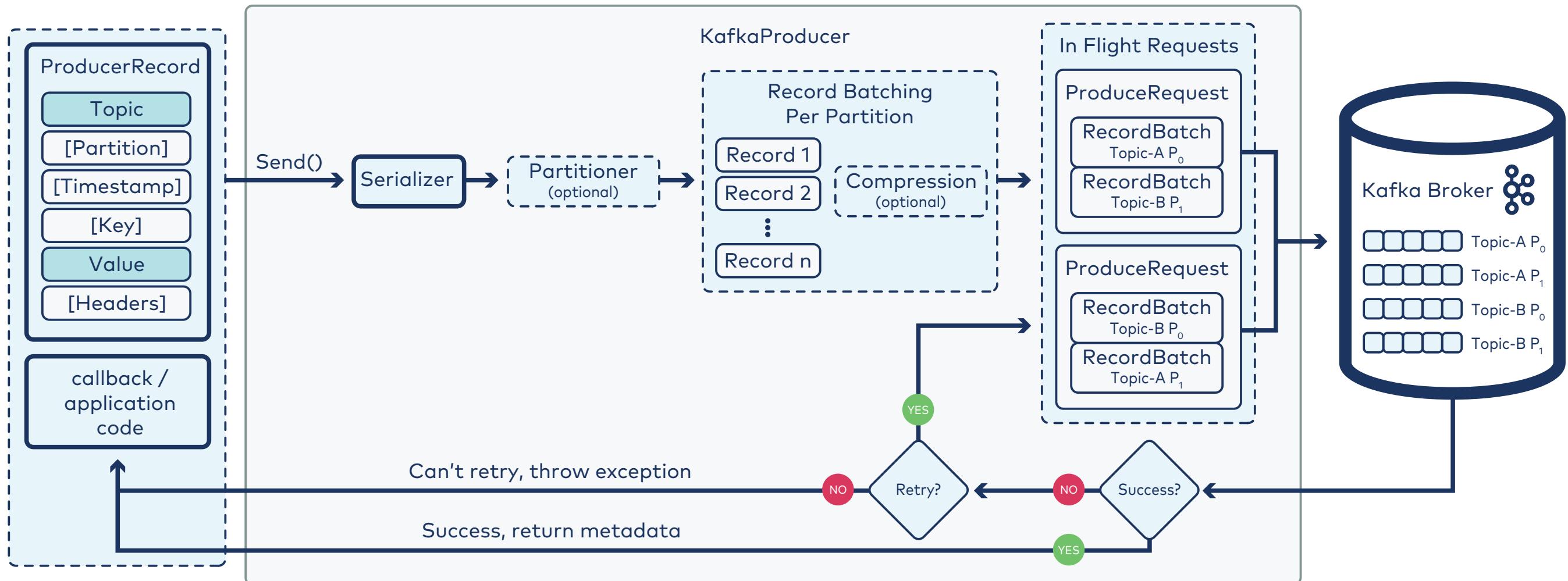
Activity: Interpreting and Applying Time Limits



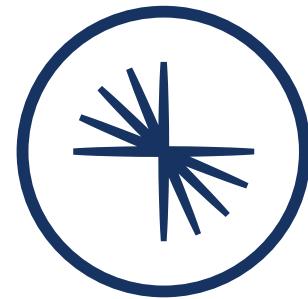
Assume you're working with all settings starting at their defaults. Work with a classmate or small group to interpret the configurations on the last few slides and answer these questions:

1. Someone reports to you that while `batch.size` is set rather high, only one message is ever being sent at a time. Batching never occurs. What setting can we change to fix this?
2. You have messages that are extremely time sensitive. No matter what happens, if they don't make it to the broker within 30 seconds of `send()` returning, there's no point. How can you enforce this?
3. Suppose you fixed the last problem correctly and have also implemented a callback, but in fact, some messages don't fail until 90 seconds after the producer tries to `send()`. Where could this extra time be coming from?

Summarizing Producer Design



04: Starting with Consumers



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. How Do You Request Data to Fetch from Kafka?
- b. What are the Basic Concepts of Kafka Consumers?
- c. How Do You Write the Code for a Basic Kafka Consumer?

Where this fits in:

- Hard Prerequisite: Starting with Producers
- Recommended Prerequisite: Preparing Producers for Practical Uses
- Recommended Follow-Up: Groups, Consumers, and Partitions in Practice

04a: How Do You Request Data to Fetch from Kafka?

Description

The consumer parallel of producer batching. Configuring consumer properties affecting fetch requests. Configuring broker properties that affect fetch requests from followers of leaders.

Fetching Data: Quick Overview

- Data has been produced to brokers
- Who fetches it?
 - Consumers
 - Followers

Back to Producers

Recall:

- A producer `send()` call writes a message to a local buffer
- Buffer accumulates batches of messages per partition
- A batch is sent from the producer to the cluster when it
 - Reaches a size threshold
 - Reaches a time threshold

Fetch requests generally work with batches of messages too

How to Control Fetch Requests

size of data in a fetch

time to wait for a fetch

Consumer Fetch Request Settings

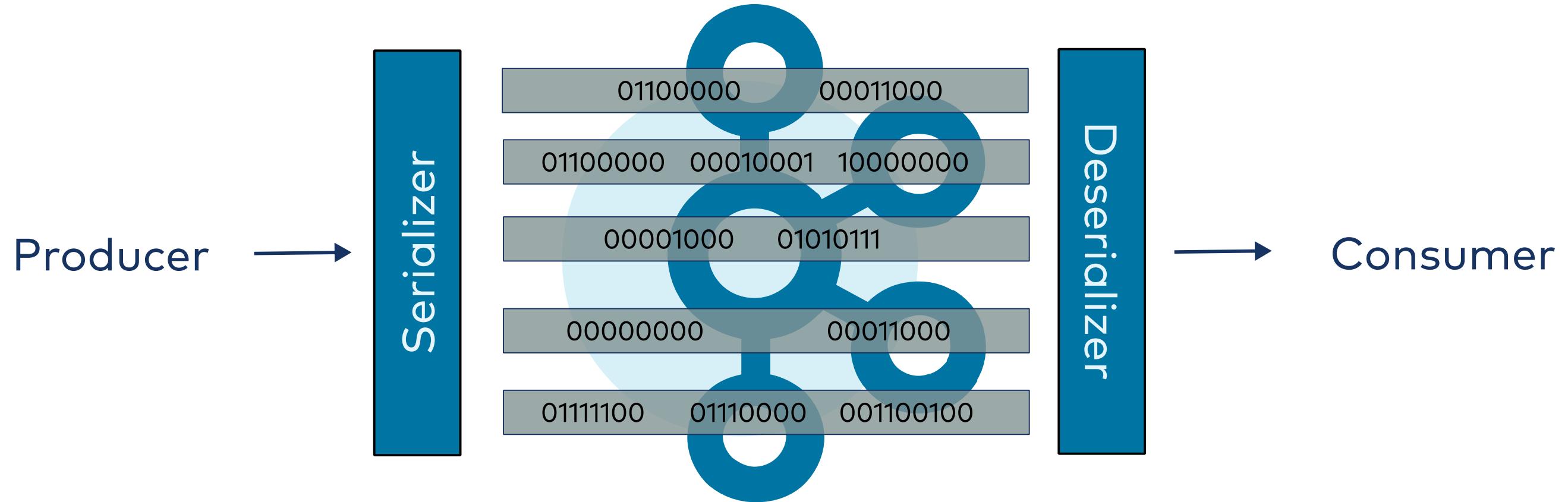
minimum size of data in a fetch	maximum amount of data in a fetch	maximum time to wait for a fetch
<code>fetch.min.bytes</code>	<ul style="list-style-type: none">• <code>max.partition.fetch.bytes</code> - size per partition• <code>max.poll.records</code> - # records across all partitions	<code>fetch.max.wait.ms</code>

04b: What are the Basic Concepts of Kafka Consumers?

Description

Conceptual view of a basic consumer.

One Config Note: Deserialization



How Do Consumers Know What Messages to Read?

You must take action

- subscribe to topic(s)

Handled automatically by Kafka/Consumer API

- choose partition to read from
- maintain consumer offset in partition



More on both of these in the recommended next module

04c: How Do You Write the Code for a Basic Kafka Consumer?

Description

Coding a basic consumer using the Java client API.

Configuration (1)

Important properties:

Name	Description
<code>bootstrap.servers</code>	List of broker host/port pairs used to establish the initial connection to the cluster
<code>key.deserializer</code>	Class used to deserialize the key. Must implement the <code>Deserializer</code> interface
<code>value.deserializer</code>	Class used to deserialize the value. Must implement the <code>Deserializer</code> interface
<code>client.id</code>	String to identify this consumer uniquely; used in monitoring and logs

Configuration (2)

We specify properties in code just like with producers.

We need a `Properties` object:

```
1 Properties props;
```

Here we show using a helper class:

```
6 props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-1:9092, kafka-2:9092, kafka-3:9092");
7 // other properties
```

Creating a Consumer

We need a `KafkaConsumer` object:

```
2 KafkaConsumer<String, String> consumer;
```

Initialization is just like with a producer:

```
10 consumer = new KafkaConsumer<>(props);
```

Subscribing to Topics

This is an additional step:

```
14 consumer.subscribe(Arrays.asList("my_topic", "my_second_topic));
```

Note that calling `subscribe` again **replaces** an existing topic list.

Consumer Record and Polling

We need an object to hold what we get back:

```
3 ConsumerRecord record;
```

Then we call `poll()`:

```
18     record = consumer.poll();
```

Consumer Records & Processing

But, remember, records are batched and we must process the batches so...

```
4 ConsumerRecords records;
```

We must process the batch of records we receive:

```
18     records = consumer.poll();
19
20     for(record : records)
21     {
22         System.out.printf("offset: %d, key: %s, value, %s\n",
23                           record.offset(), record.key(), record.value());
24     }
```

Consumers Run Indefinitely

```
16  while(true)
17  {
18      records = consumer.poll();
19
20      for(record : records)
21      {
22          System.out.printf("offset: %d, key: %s, value, %s\n",
23                            record.offset(), record.key(); record.value());
24      }
25 }
```

Timeout on Polling

Let's tweak the poll:

```
16  while(true)
17  {
18      records = consumer.poll(Duration.ofMillis(100));
19
20      for(record : records)
21      {
22          System.out.printf("offset: %d, key: %s, value, %s\n",
23                            record.offset(), record.key(); record.value());
24      }
25  }
```

Cleaning Up

Like with producers, we must `close` to clean up open resources:

```
29 consumer.close();
```

We could supply a timeout, as with the producer:

```
consumer.close(Duration.ofMillis(100));
```

Coding Safely and Putting It All Together

To do this safely, we should employ exception handling. Here's the full consumer:

```
1 Properties                  props;
2 KafkaConsumer<String, String> consumer;
3 ConsumerRecord               record;
4 ConsumerRecords              records;
5
6 props = new Properties();
7 props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "broker1:9092");
8 // other properties
9
10 consumer = new KafkaConsumer<>(props);
11
12 try
13 {
14     consumer.subscribe(Arrays.asList("my_topic", "my_second_topic"));
```

Coding Safely and Putting It All Together, continued

```
16  while(true)
17  {
18      records = consumer.poll(Duration.ofMillis(100));
19
20      for(record : records)
21      {
22          System.out.printf("offset: %d, key: %s, value, %s\n",
23                            record.offset(), record.key(); record.value());
24      }
25  }
26 }
27 finally
28 {
29     consumer.close();
30 }
```

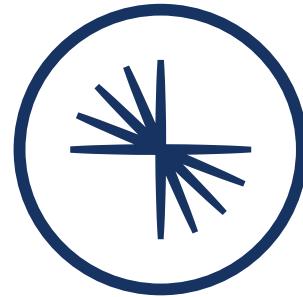
Lab: Basic Kafka Consumer

Please work on **Lab 4a: Basic Kafka Consumer**

Refer to the Exercise Guide



05: Groups, Consumers, and Partitions in Practice



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. How Do Groups Distribute Workload Across Partitions?
- b. How Does Kafka Manage Groups?
- c. How Do Consumer Offsets Work with Groups?

Where this fits in:

- Hard Prerequisite: Starting with Consumers
- Recommended Follow-Up: Either of these branches of developer content:
 - Other Components of a Kafka Deployment
 - Additional Challenges in Core Kafka Components

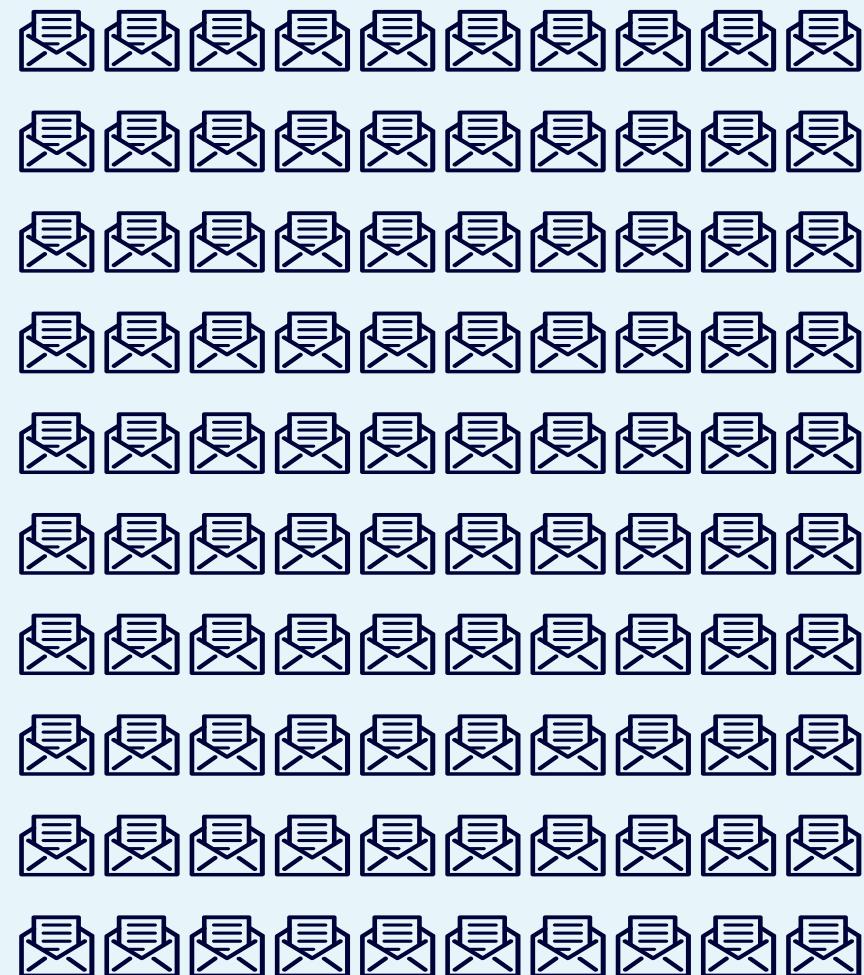
05a: How Do Groups Distribute Workload Across Partitions?

Description

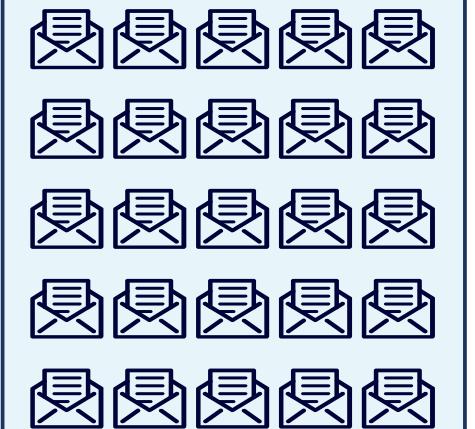
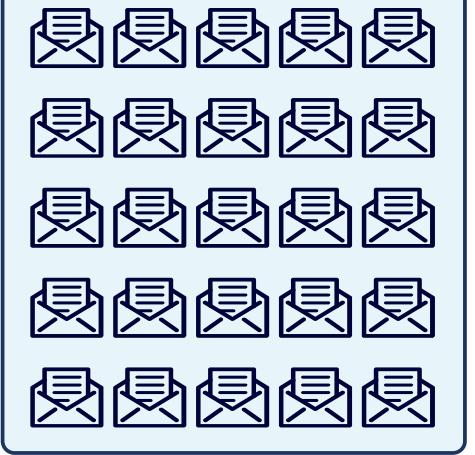
Consumers in groups, what distinguishes consumers in same group and why we'd use more than one group. Valid assignments of consumers to partitions, range and round robin partition assignment strategies and when to use each.

Teamwork in Real Life

Working Alone



Working as a Team



Grouping Consumers

- Best practice is for consumers to operate in **consumer groups**
- Consumers in a group...
 - ...all do the same thing
 - ...using different data
- Why?
 - Share workload
 - Consume in parallel
 - Scale up and down

Groups Aren't Just for Consumers!

- Groups allow for parallel processing and scaling up and down
- Kafka has built automatic group management
 - next lesson!
- Groups are leveraged in many places in Kafka and Confluent Platform:
 - consumers
 - Kafka Connect workers
 - Kafka Streams applications
 - ksqlDB servers

How do we Configure Consumer Groups?

Set consumer property `group.id` like any other property, e.g.,

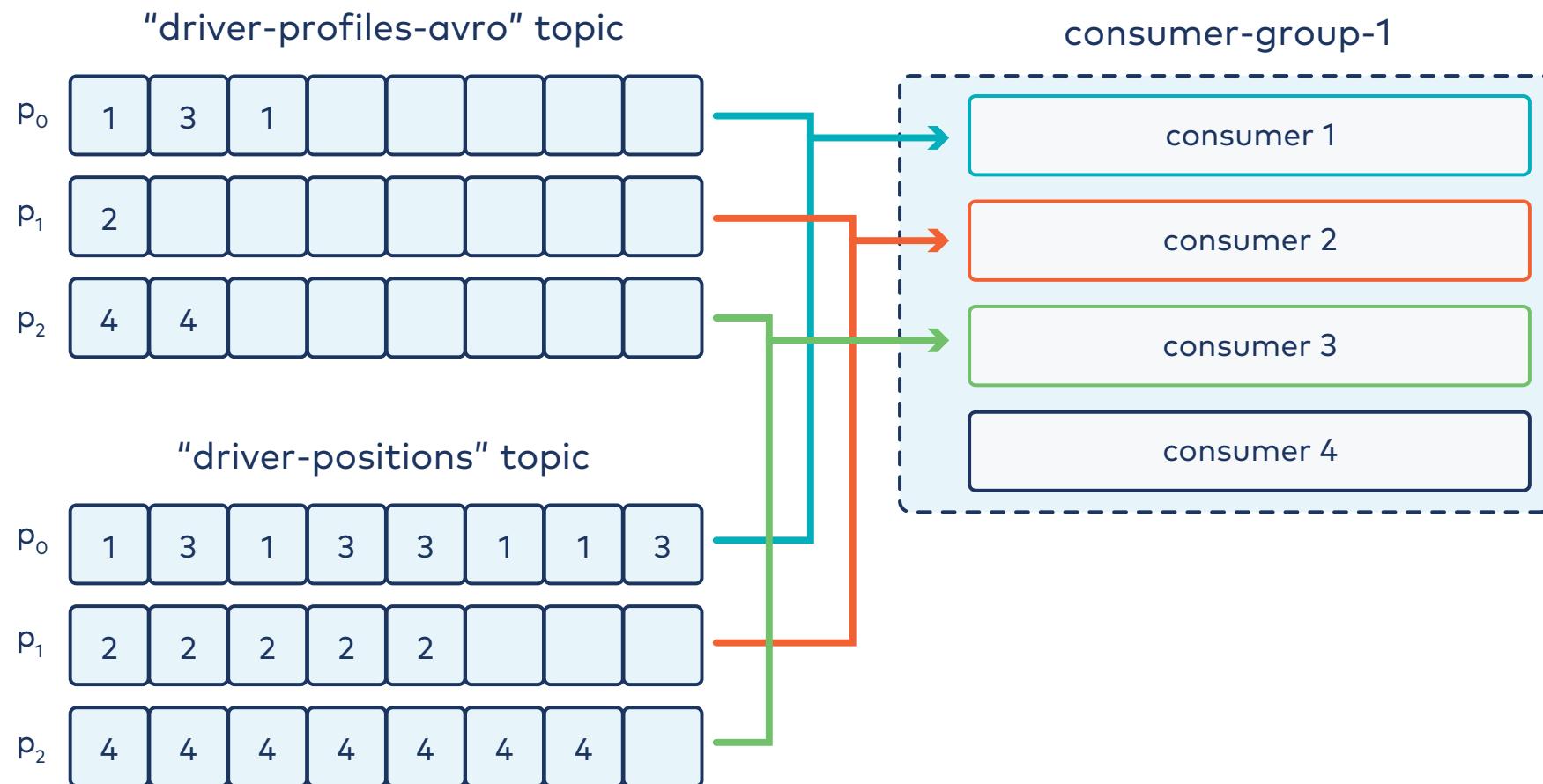
```
props.put(ConsumerConfig.GROUP_ID_CONFIG, "order_processor");  
// other properties
```

Consumers and Partitions

- Earlier, we said consumers in a group do the same thing on different data
- Given a subscribed topic, consumers in a group work together to consume all partitions of a topic
- Consumers get assigned partitions to consume
 - Two major types of `partition.assignment.strategy...`

Partition Assignment: Range

Why? Relate data across two or more co-partitioned topics.



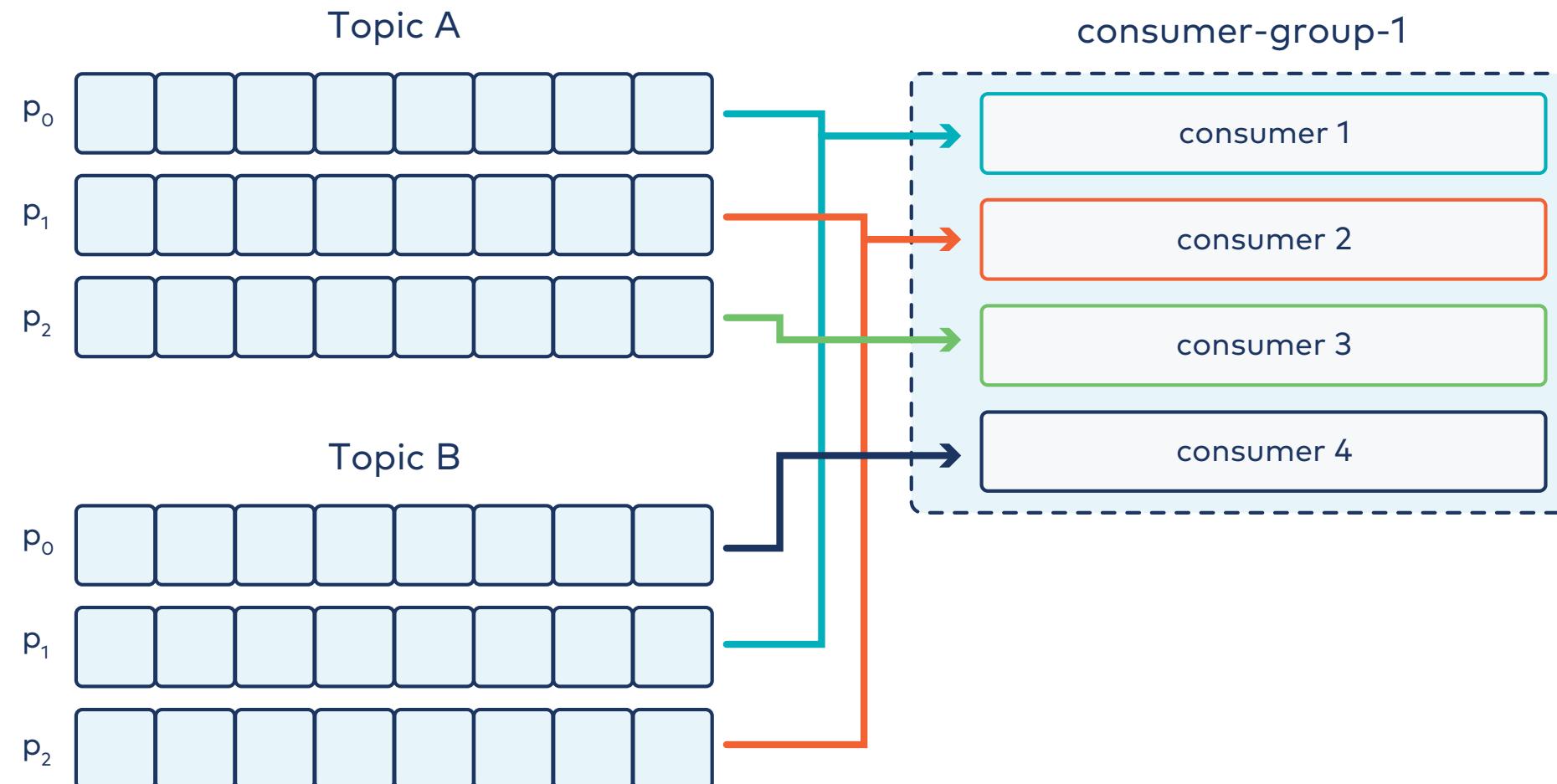
All topics must be **co-partitioned**, i.e., have

- 1. same number of partitions
- 2. same partitioner
- 3. same set of keys

This is the default strategy.

Partition Assignment: Round Robin

Why? Balance load so each consumer has roughly the same number of partitions.



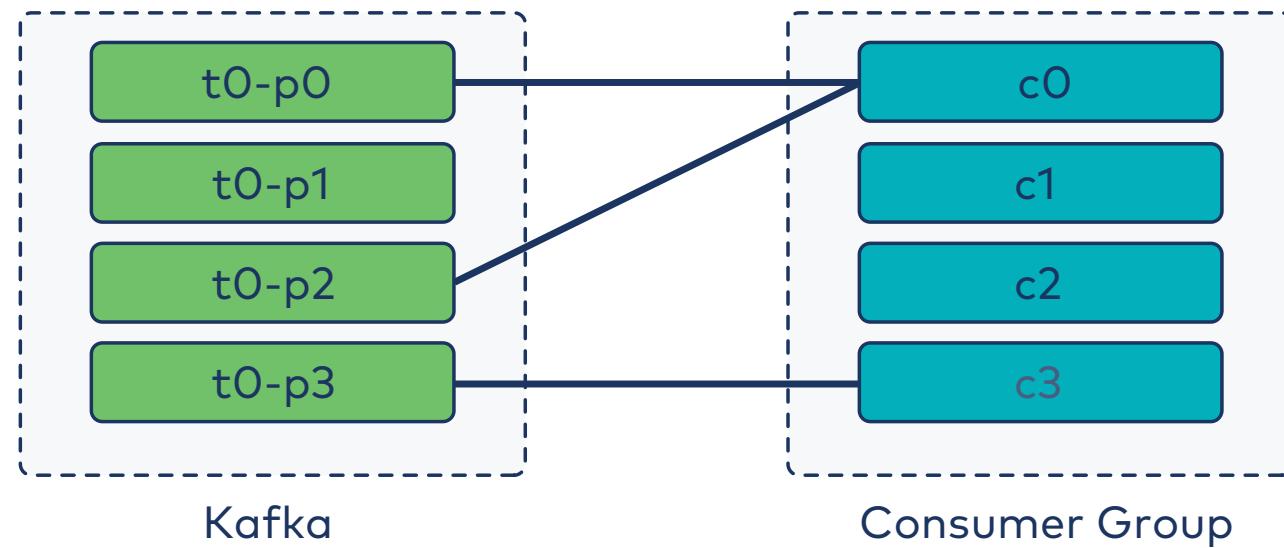
05b: How Does Kafka Manage Groups?

Description

Heartbeats, what triggers rebalancing, pros and cons of rebalancing. Relating Kafka's group management protocol across various components, e.g., consumers and workers. Reasons to grow and shrink group size.

What's Wrong With This Picture?

Here, the consumers in the group are all subscribed to topic `t0`.



Rebalancing Partition/Consumer Assignments

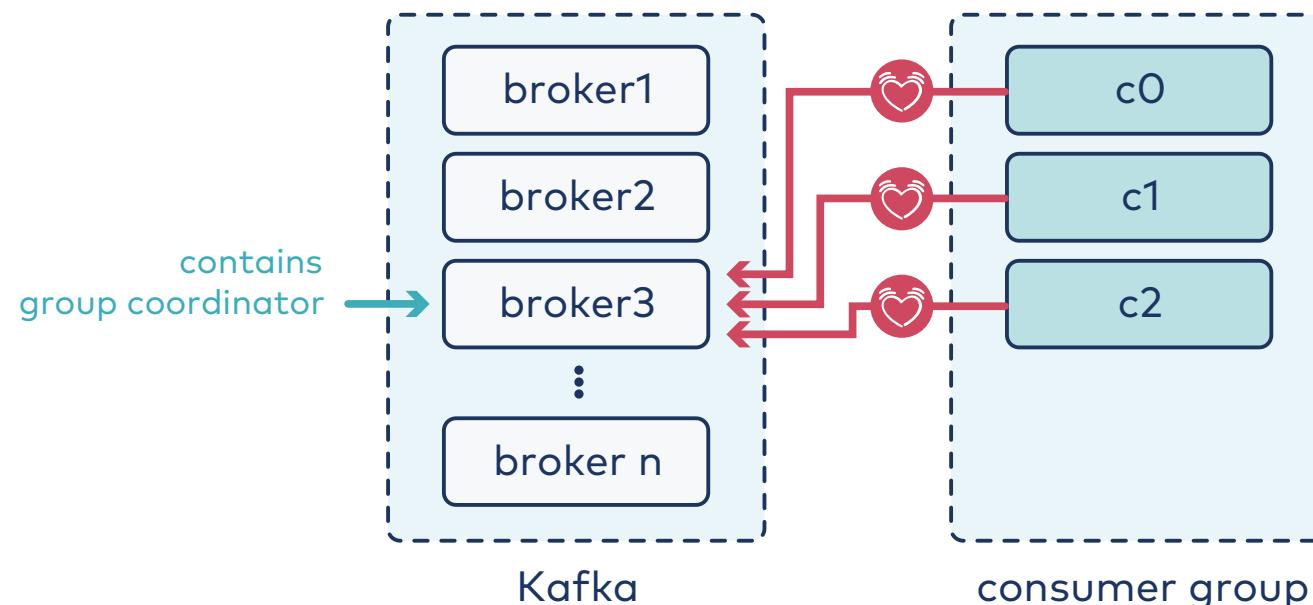
In situations like on the last slide, the assignment isn't optimal. Kafka's group management protocol detects non-ideal assignments and **rebalances**.

Triggers of rebalance:

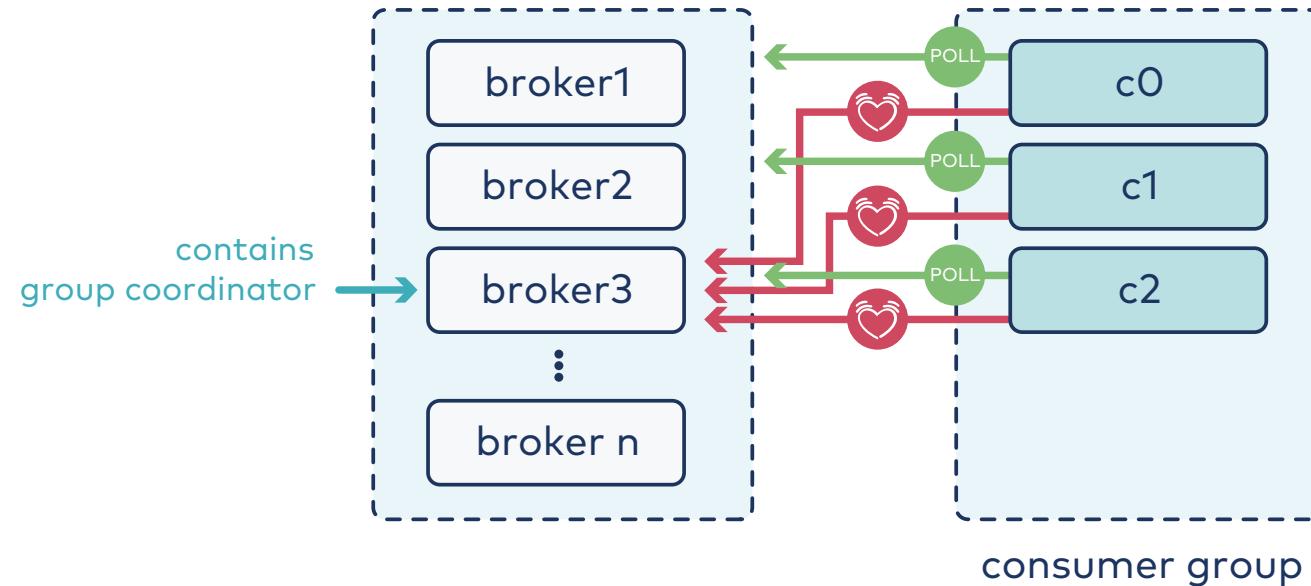
- **Number of consumers in play changes**
 - Consumer was added
 - Consumer was removed
 - Intentionally
 - Consumer died
- **Number of partitions in play changes**
 - Number of partitions for topic increased
 - Topic subscription changed

How Does Kafka Detect Dead Consumers?

- Consumers send a heartbeat to Kafka every 3 s (or `heartbeat.interval.ms`)
- If 45 s (or `session.timeout.ms`) passes without a consumer heartbeating, it is deemed dead



But wait...



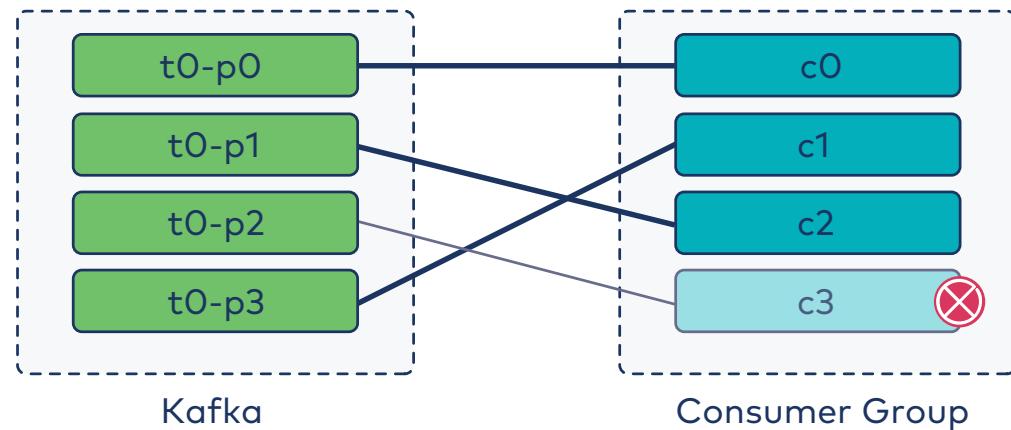
- Consumers must poll
- If 5 mins (or `max.poll.interval.ms`) passes without poll, consumer is also deemed dead

Implications of Rebalance

- Consumption pauses
- A consumer may read a partition some other consumer had been reading
 - Offsets need to be communicated—see next lesson!
- Stateful application may need to rebuild state
- Kafka takes the opportunity to optimize assignments

Do We Always Want to Reassign **Everything**?

Suppose you have this scenario:



So...

- Prerequisite: Using round robin assignment, stateful consumer
- Problem: One partition isn't being read
 - But others are okay
 - Rebalance takes time
- Solution: Use **sticky partitioner**
 - Preserves assignments that don't need to change
 - Selection of new assignment is faster

Remember: Not Just Consumers

Group management applies not just to consumers.

Kafka leverages similar logic for groups in other places that you may learn about in other modules.

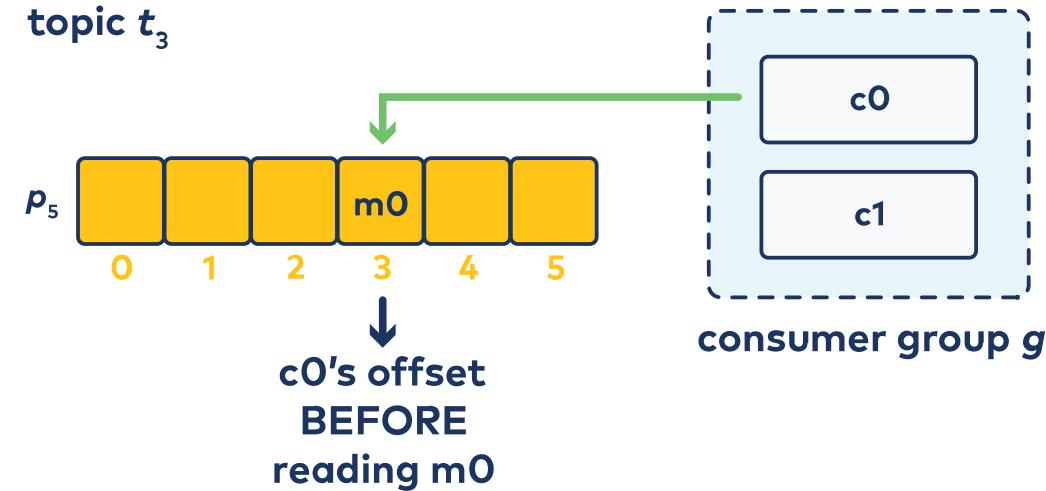
05c: How Do Consumer Offsets Work with Groups?

Description

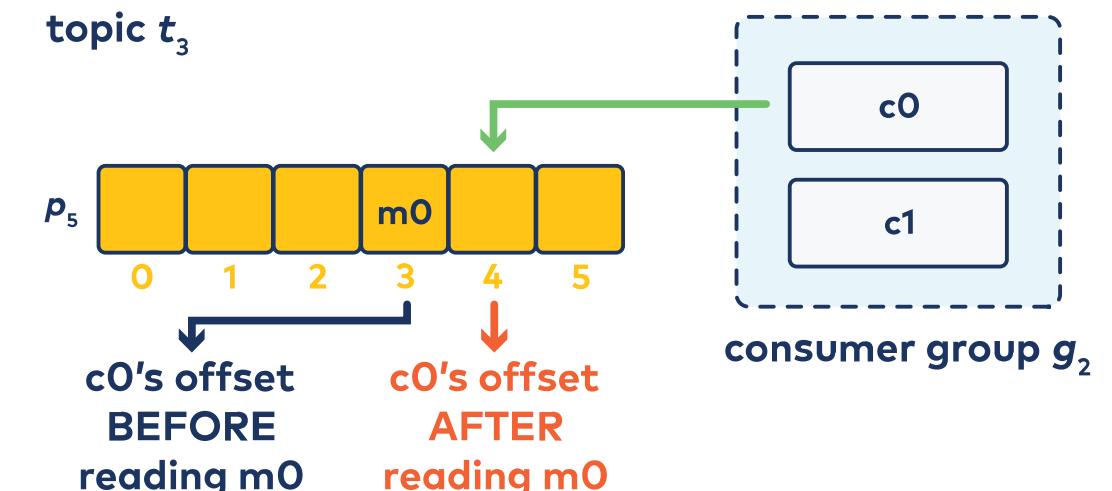
Consumer offsets locally vs. committed to Kafka. Automatic committing. How consumers newly assigned to a partition can recover committed offsets.

Review: What is a Consumer Offset?

Before read



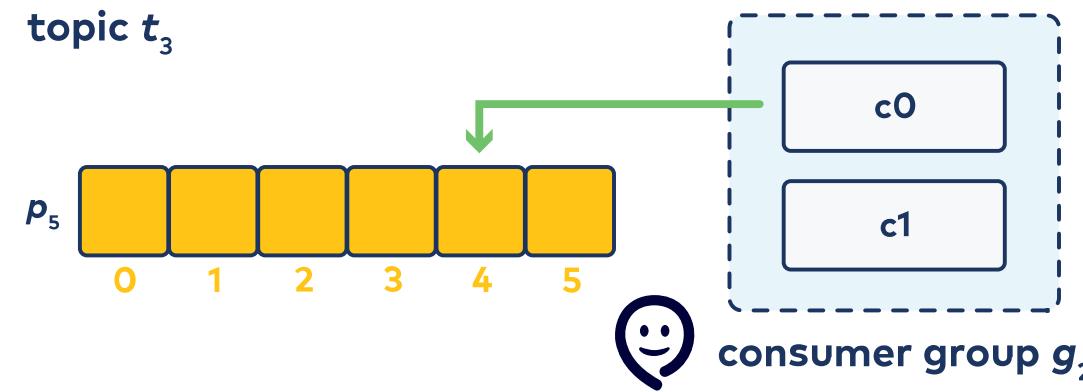
After read



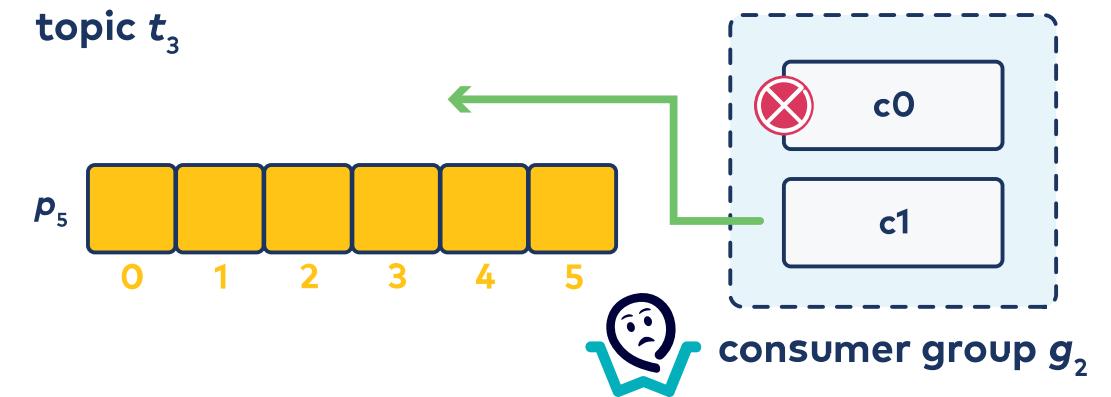
- Per consumer
- Per partition
- Where to read **next**

What about Rebalancing?

Before rebalance



After rebalance



Q: Where does c_1 start reading?

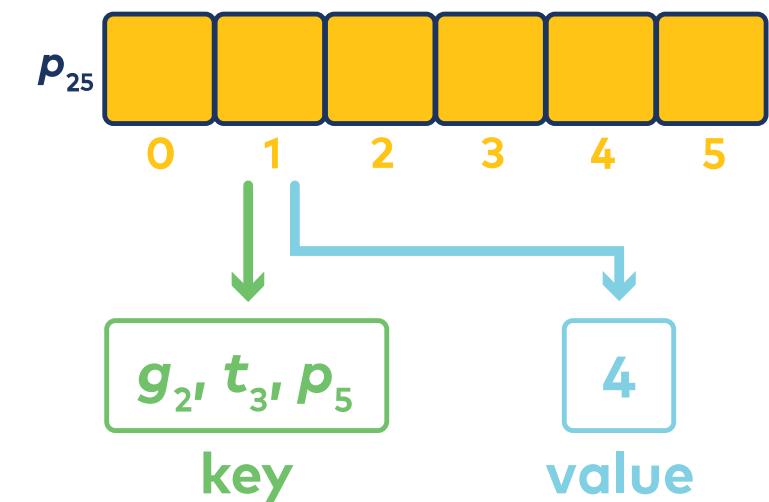
Committing Offsets

- Internal topic `__consumer_offsets` tracks consumer offsets. Each entry:

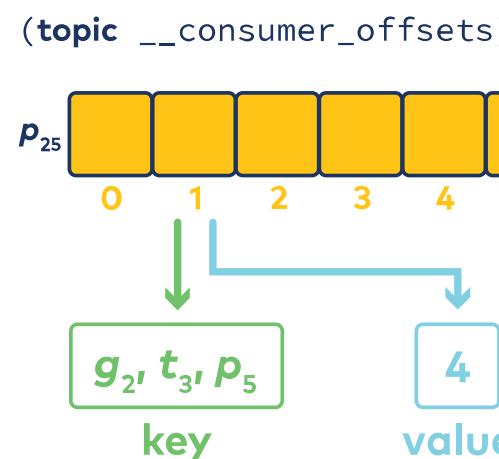
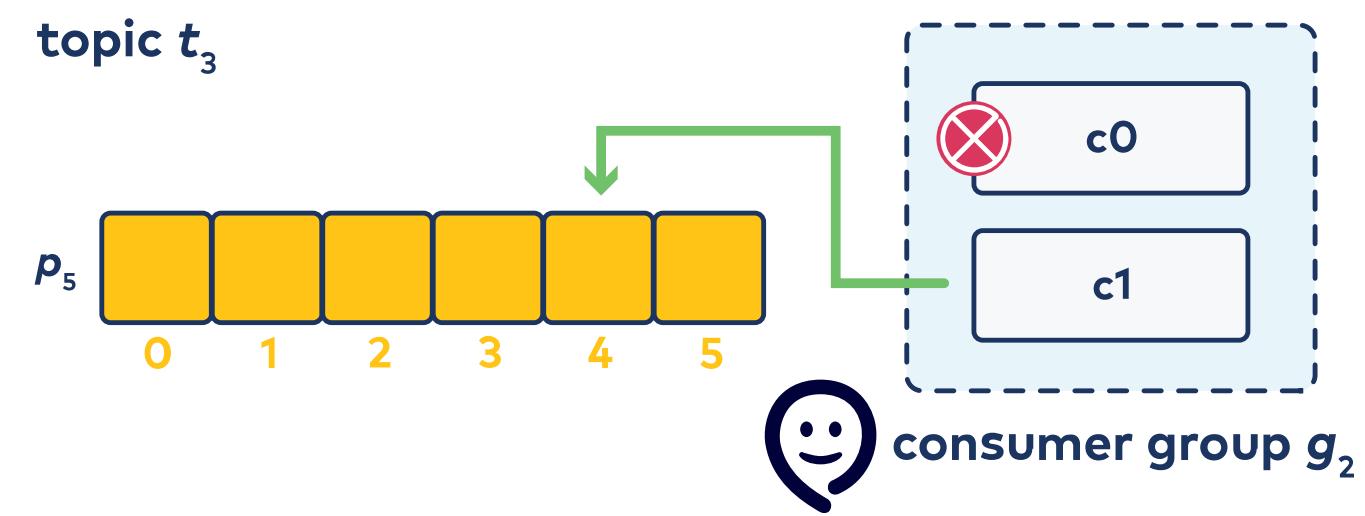
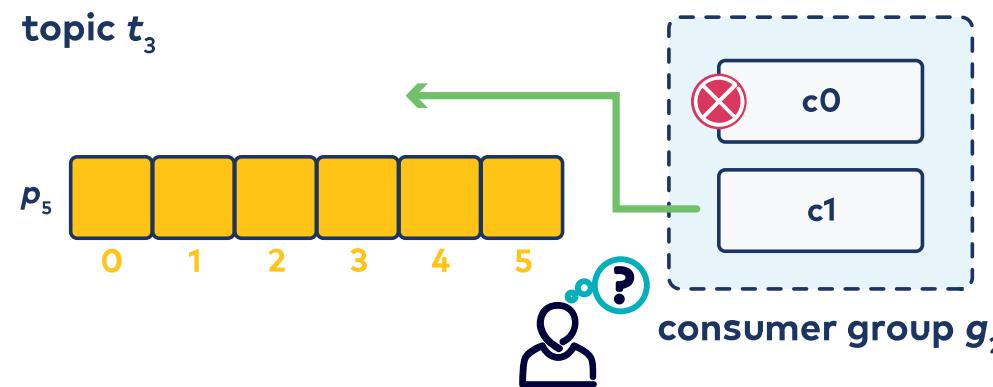
Key	Value
<ul style="list-style-type: none">grouptopicpartition	<ul style="list-style-type: none">offset

- Consumers back up their offsets to this topic
 - called **committing offsets**
 - every 5 s (or `auto.commit.interval.ms`)
- A consumer newly assigned to a partition checks here to know where to start consuming

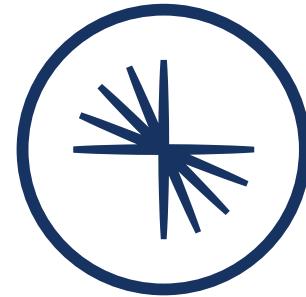
(topic `__consumer_offsets`)



So...



Additional Components of Kafka/CP Deployment Overview



CONFLUENT
Global Education

Agenda

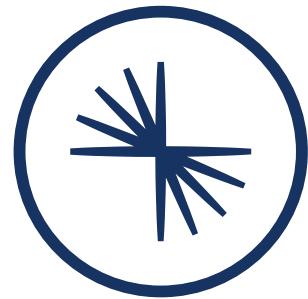


This is a branch of our developer content on additional components common in Kafka/Confluent Platform deployment. It is broken down into the following modules:

6. Starting with Schemas
7. Integrating with the Schema Registry
8. Introduction to Streaming and Kafka Streams
9. Introduction to ksqlDB
10. Starting with Kafka Connect
11. Applying Kafka Connect

This branch assumes proficiency in concepts from the Core Kafka Development branch.

06: Starting with Schemas



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. Why Should You Care About Schemas?
- b. How Do You Write Schemas in Avro or Protobuf?
- c. How Do You Design Schemas that can Evolve?

Where this fits in:

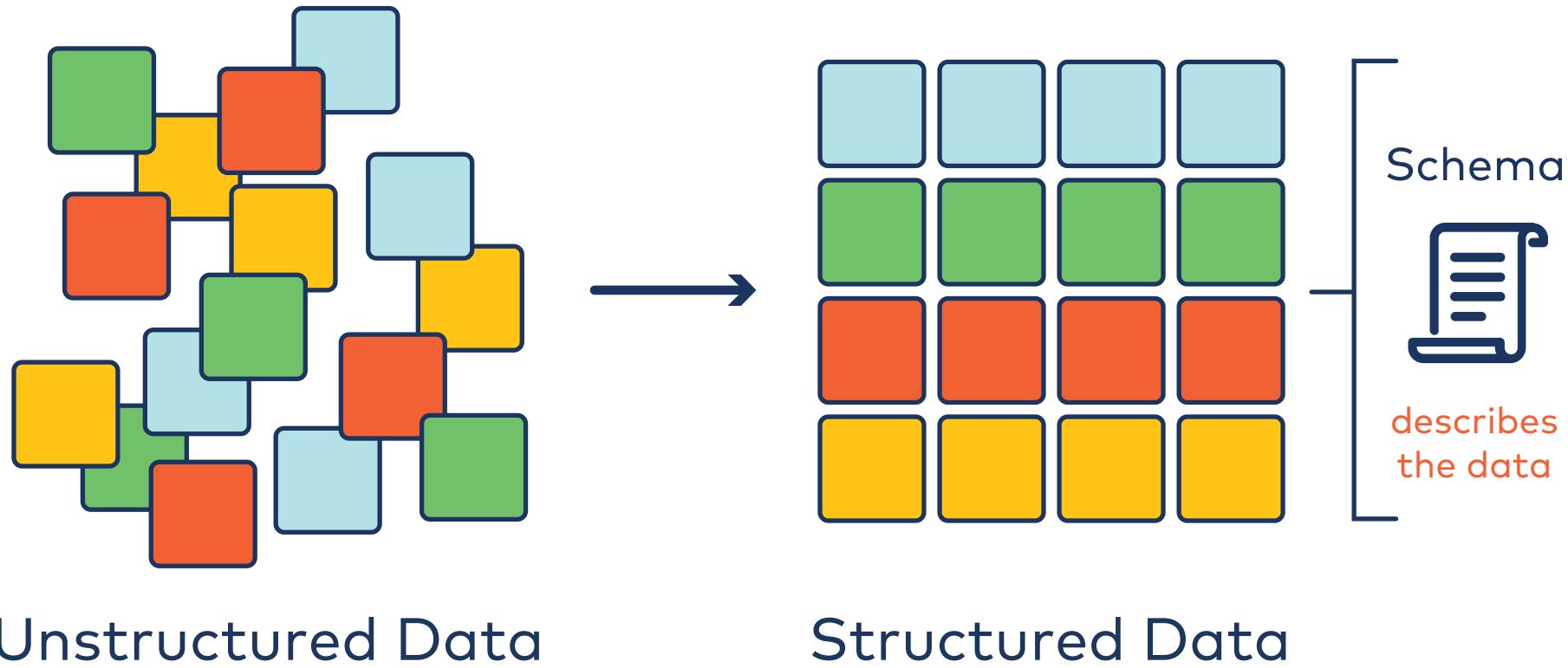
- Hard Prerequisite: Starting with Consumers
- Recommended Follow-Up: Integrating with the Schema Registry

06a: Why Should You Care About Schemas?

Description

What schemas are and why we use them.

The Need For Schemas



Schemas Can Change

Imagine `BusinessCustomer` schema:

1980s	1990s	now
<ul style="list-style-type: none">• company name• contact person• physical address• phone• fax number	<ul style="list-style-type: none">• company name• contact person• physical address• phone• fax number• email address	<ul style="list-style-type: none">• company name• contact person• physical address• phone• fax number• email address



Schema Evolution will be an important topic we visit in two lessons.

Specifying Schemas

In a subsequent lesson, we'll formalize how we should specify a schema and how we can do it in Confluent Platform.

For now, here's a working conceptual version of a schema for a simple `WritingUtensil`:

- type, a string
- tip size in MM, a number
- color, a string
- brand, a string

Design with Schemas in Mind from the Start

- Think about schemas early
- Plan for schema evolution
- Confluent Schema Registry can help
 - Can leverage with producers and consumers
 - ...and much more

Let's get into the technical details!

06b: How Do You Write Schemas in Avro or Protobuf?

Description

Defining why one would use Avro or Protobuf. Specifying basic schemas in both. More involved schemas in Avro via activity.

Schema Specification and Serialization Mechanisms (1)

- Can leverage a mechanism to help with
 - Specifying schemas
 - Serialization
- Both Avro and Protobuf support this
- Both are supported by Confluent Schema Registry
- Example schemas on next slide...

Schema Specification and Serialization Mechanisms (2)

Example schema in both:

Avro

```
{  
  "namespace": "clients.avro",  
  "type": "record",  
  "name": "PositionValue",  
  "fields": [  
    {"name": "latitude",  
     "type": "double"},  
    {"name": "longitude",  
     "type": "double"}  
  ]  
}
```

Protobuf

```
syntax = "proto3";  
  
option java_package = "clients.proto";  
message PositionValue {  
  double latitude = 1;  
  double longitude = 2;  
}
```

Honing in on Avro

- The way we'll use Avro:
 - Specify schema using JSON
 - Use it to generate an equivalent Java class to include in our code
 - Called **specific** mode
- Details:
 - File extension `.avsc`
 - Provide a `namespace` with a schema that becomes Java `package` to import
 - Specify a schema, commonly done with a data type of `record`



Sample Schema In Avro

```
{  
  "namespace": "clients.avro",  
  "type": "record",  
  "name": "WritingUtensil",  
  "fields": [  
    {"name": "type", "type": "string"},  
    {"name": "tipSizeMM", "type": "double"},  
    {"name": "color", "type": "string"},  
    {"name": "brand", "type": "string"}  
  ]  
}
```

Adding a Few Things

```
{  
  "namespace": "clients.avro",  
  "type": "record",  
  "name": "WritingUtensil",  
  "fields": [  
    {"name": "type", "type": "string", "default": "pen"},  
    {"name": "tipSizeMM", "type": "double", "doc": "size of tip in mm"},  
    {"name": "color", "type": {  
      "type": "enum",  
      "name": "color",  
      "symbols": "blue", "black", "red", "green"},  
    "name": "brand", "type": "string"}  
  ]  
}
```

- Observe:
1. `default` value for a field
 2. `doc` string
 3. `enum` data type

What Else Can You Do?

- Nest records within records
- Use `array` data type
- Use `map` data type
 - Keys must be strings
- Allow fields to take on a `null` value
- Use any of these simple data types: `boolean`, `int`, `long`, `float`, `double`, `string`



Documentation link and more examples in your handbook.

Choosing Between Avro and Protobuf

- Both are supported by Confluent Schema Registry
 - Avro support has been around longer
- Both encode data efficiently
 - But differently - see comparison link in guide
- What is your organization otherwise using?



A Step Beyond

Schemas Written in JSON Format

Avro Schema in JSON Format

```
{  
  "namespace": "clients.avro",  
  "type": "record",  
  "name": "PositionValue",  
  "fields": [  
    {"name": "latitude",  
     "type": "double"},  
    {"name": "longitude",  
     "type": "double"}  
  ]  
}
```

JSON Schema

```
{  
  "type": "object",  
  "title": "driverposition",  
  "properties": {  
    "latitude": { "type": "number" },  
    "longitude": { "type": "number" }  
  }  
}
```



Messages that use JSON encoding store field names in each individual record...

06c: How Do You Design Schemas that can Evolve?

Description

Defining schema evolution. Comparing the enforcement modes - backward, forward, full, none, transitive modes - with concrete examples via exercises.

Introduction

Schemas can change!

Recall example from introduction on changes to customer record:

- add email address
- remove fax number



Schema Compatibility Modes

- There are various modes defining whether one version of a schema is compatible with another
- A system like Confluent Schema Registry can enforce compatibility
- We'll focus on the modes relating two adjacent versions of a schema defined in Schema Registry

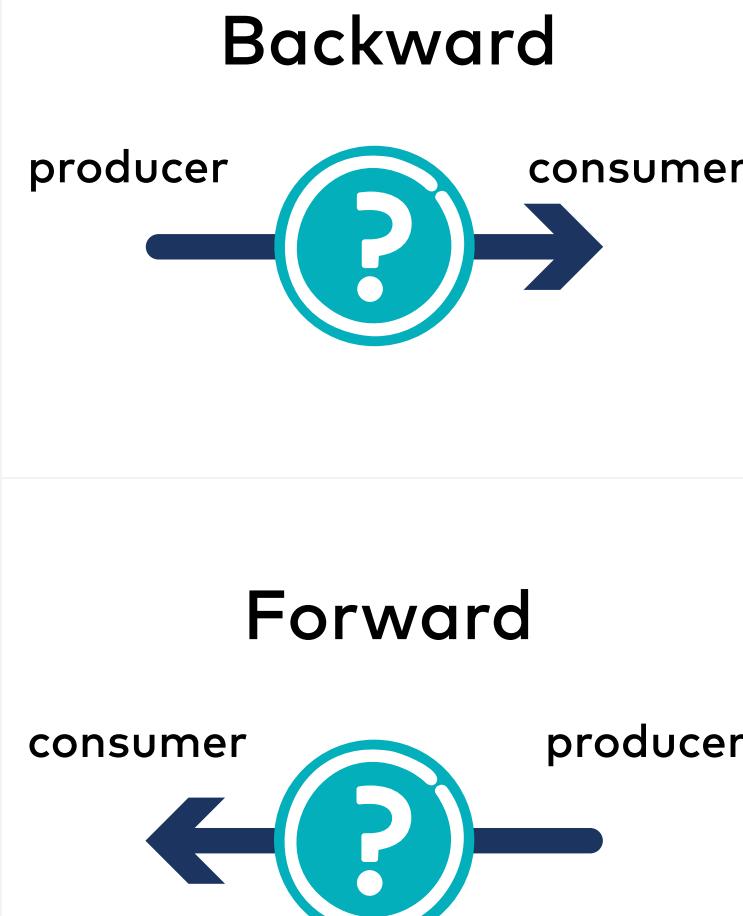
Basic Schema Compatibility Modes

Mode	Explanation
BACKWARD	Consumers expecting data using new form of schema can process data using previous version of schema
FORWARD	Consumers expecting data using previous form of schema can process data using next version of schema
FULL	Both BACKWARD and FORWARD
NONE	No compatibility checking

Example

Schema V1 fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  }]
```



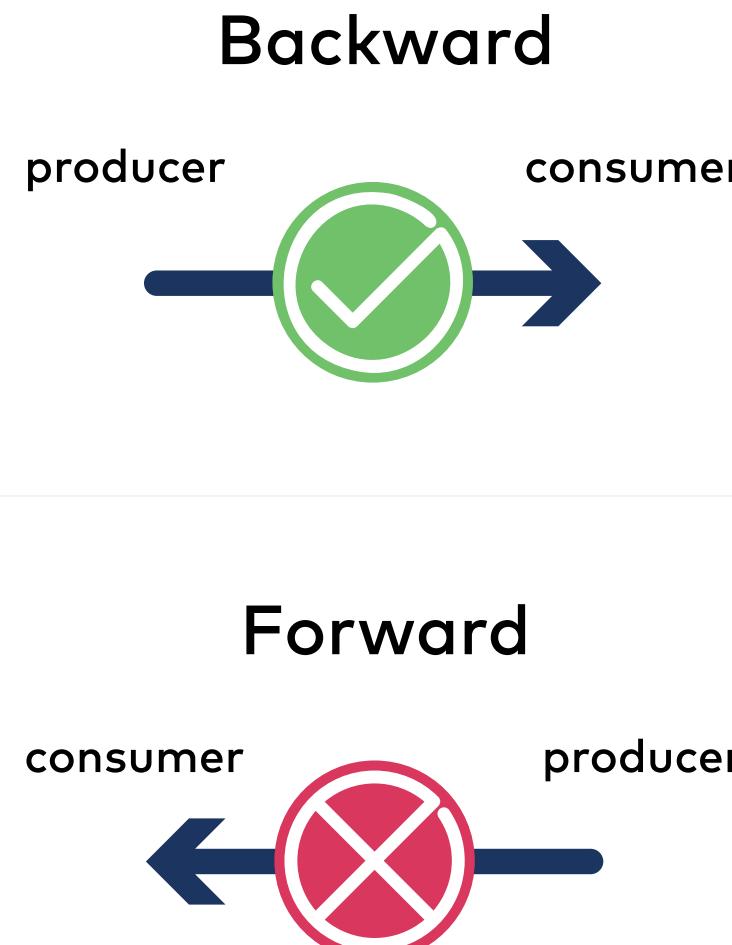
Schema V2 fields

```
"fields":  
[  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  },  
  {  
    "name": "hobby",  
    "type": "string",  
    "default": ""  
  }]
```

Example Solved

Schema V1 fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  }  
]
```



Schema V2 fields

```
"fields":  
[  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  },  
  {  
    "name": "hobby",  
    "type": "string",  
    "default": ""  
  }  
]
```

Activity: Assessing Schema Compatibility

Now evaluate in both directions - Pair 1:

Schema V1 Fields		Schema V2 Fields
<pre>"fields": [{ "name": "firstname", "type": "string" }, { "name": "lastname", "type": "string" }, { "name": "age", "type": "int", "default": -1 }]</pre>	<p style="text-align: center;">Backward</p> <p>producer  consumer</p> <p style="text-align: center;">Forward</p> <p>consumer  producer</p>	<pre>"fields": [{ "name": "firstname", "type": "string" }, { "name": "lastname", "type": "string" }, { "name": "hobby", "type": "string" }]</pre>

Activity, continued

Now evaluate in both directions - Pair 2:

Schema V1 Fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  }  
]
```

Backward



Forward



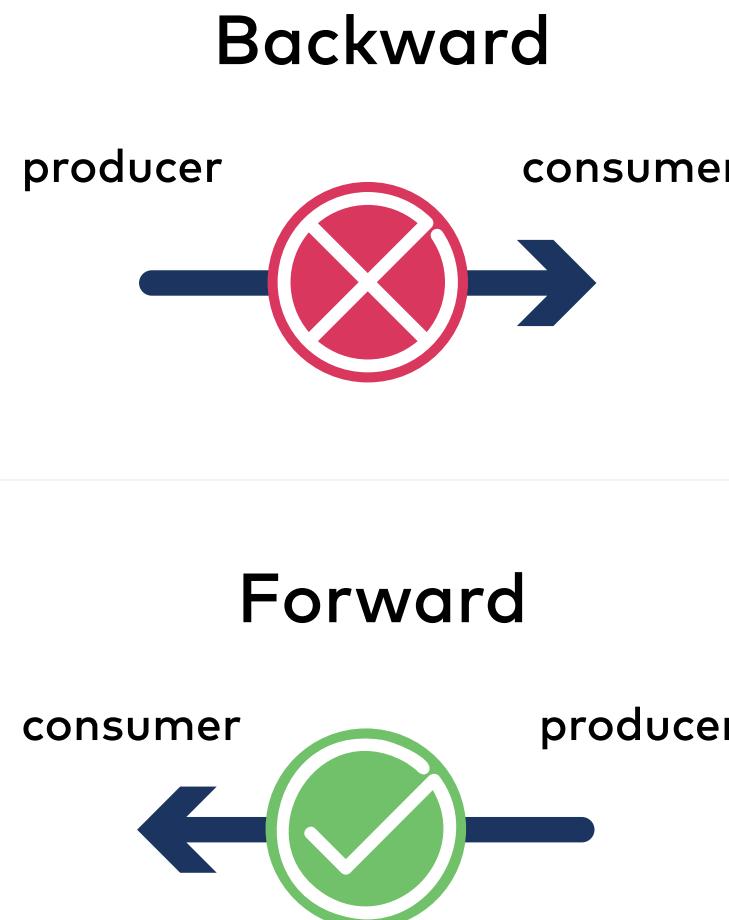
Schema V2 Fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "hobby",  
    "type": "string",  
    "default": ""  
  }  
]
```

Activity Solution, Part 1

Schema V1 Fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  }]
```



Schema V2 Fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "hobby",  
    "type": "string"  
  }]
```

Activity Solution, Part 2

Schema V1 Fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": -1  
  }]
```



Schema V2 Fields

```
"fields":  
[  
  {  
    "name": "firstname",  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "hobby",  
    "type": "string",  
    "default": ""  
  }]
```

Beyond One Version Change

Suppose for some schema,

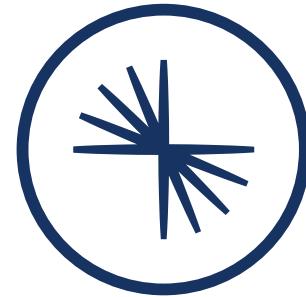
- Version 3 is backward compatible with Version 2
- Version 2 is backward compatible with Version 1

This does **not** mean Version 3 is backward compatible with Version 1

Want to enforce backward compatibility with **all** prior versions → set `BACKWARD_TRANSITIVE` mode.

Similarly, Schema Registry supports `FORWARD_TRANSITIVE` and `FULL_TRANSITIVE`.

07: Integrating with the Schema Registry



CONFLUENT
Global Education

Module Overview



This module contains one lesson:

- a. How Do You Make Producers and Consumers Use the Schema Registry?

Where this fits in:

- Hard Prerequisite: Starting with Schemas
- Recommended Follow-Up: continuing with other modules in this branch

07a: How Do You Make Producers and Consumers Use the Schema Registry?

Description

What Confluent Schema Registry is and benefits of using it. Life cycle of a message using SR. Specifying schema evolution restrictions using SR.

Scenario

- Say...
 - We have a schema for a bank transaction
 - It contains many details and uses 1 KB of data
 - It is sent with every record for a transaction
 - It is stored with every record for a transaction
 - A bank produces roughly a million transactions per hour

Question: How much disk space does this use? How much bandwidth?

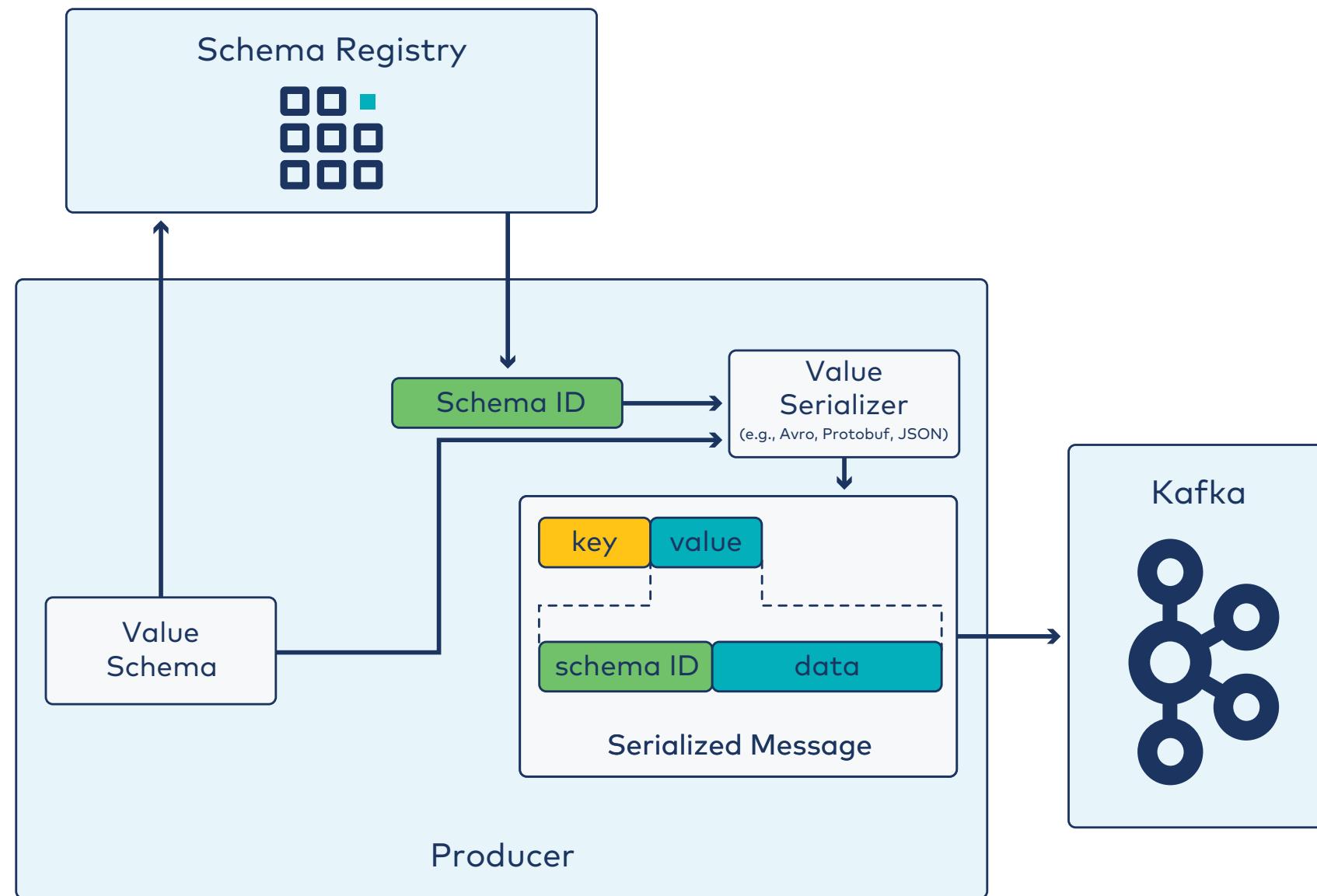
Schema ID

- Maintained by Schema Registry
- Uniquely identifies
 - Schema
 - Version
- Sent with records instead of whole schema → efficient!
- Schema Registry tracks schemas and IDs in internal `_schemas` topic in Kafka

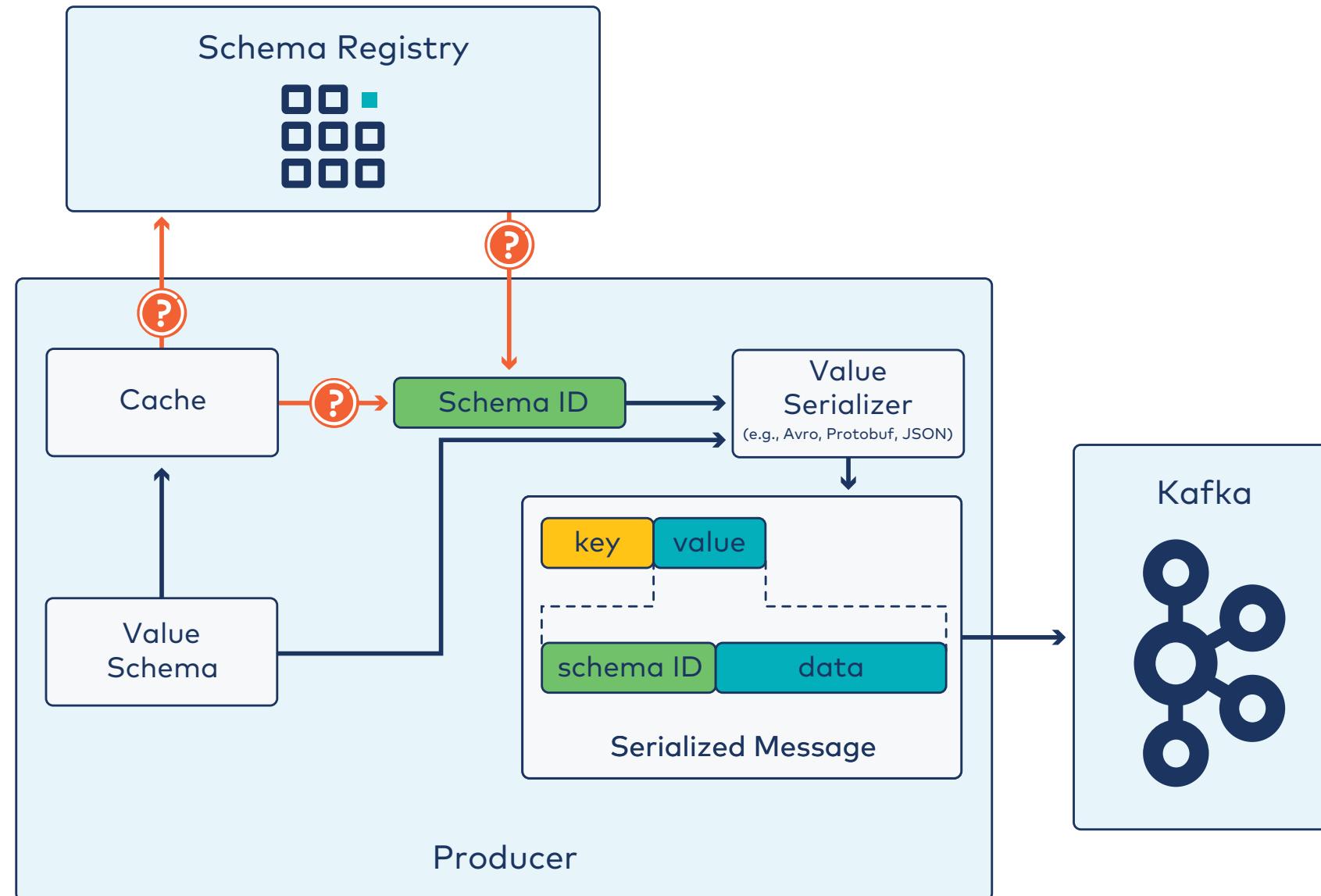
What Else Can Schema Registry Do For You?

- Check records to make sure they conform to schemas
- Enforce schema evolution

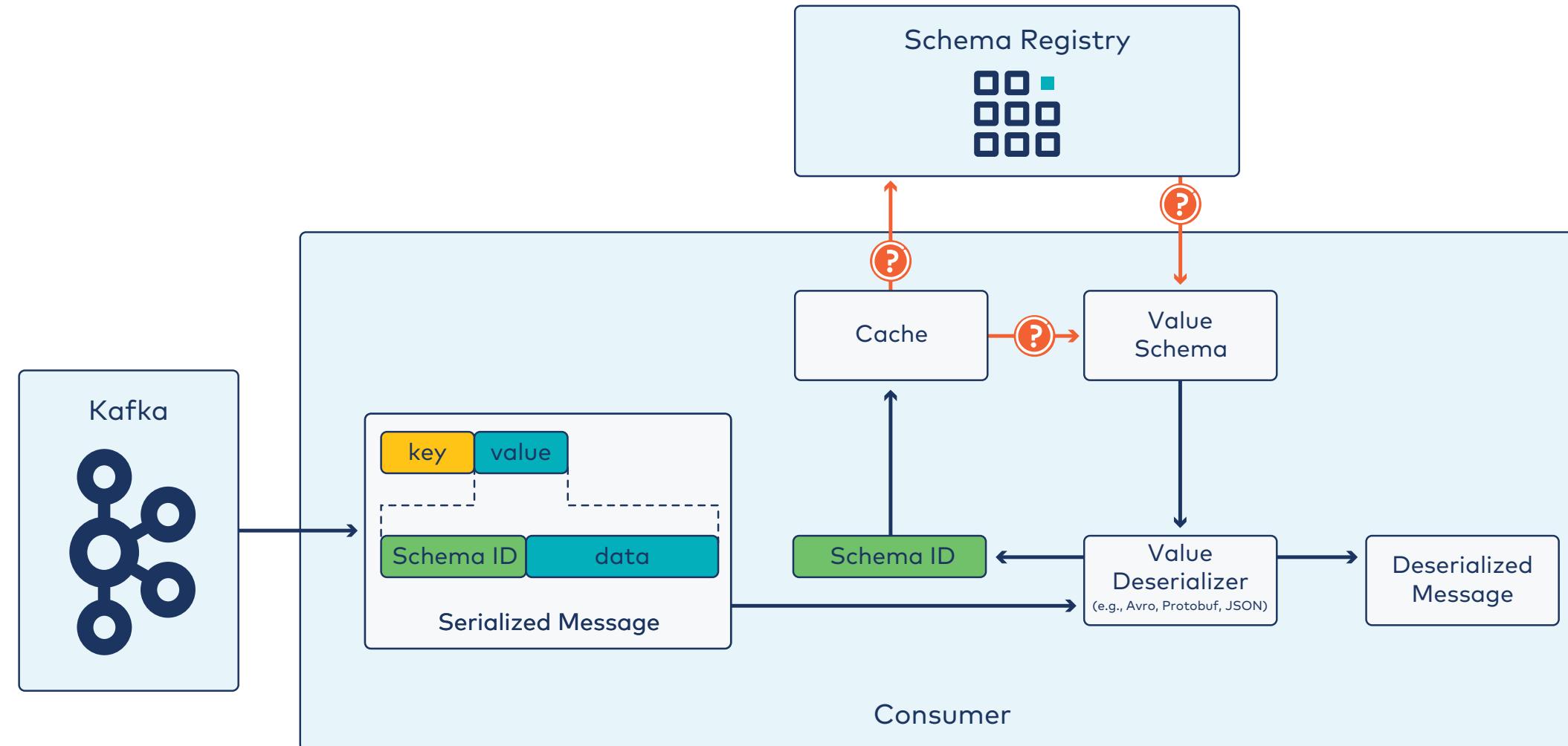
Schema Registry: Producers



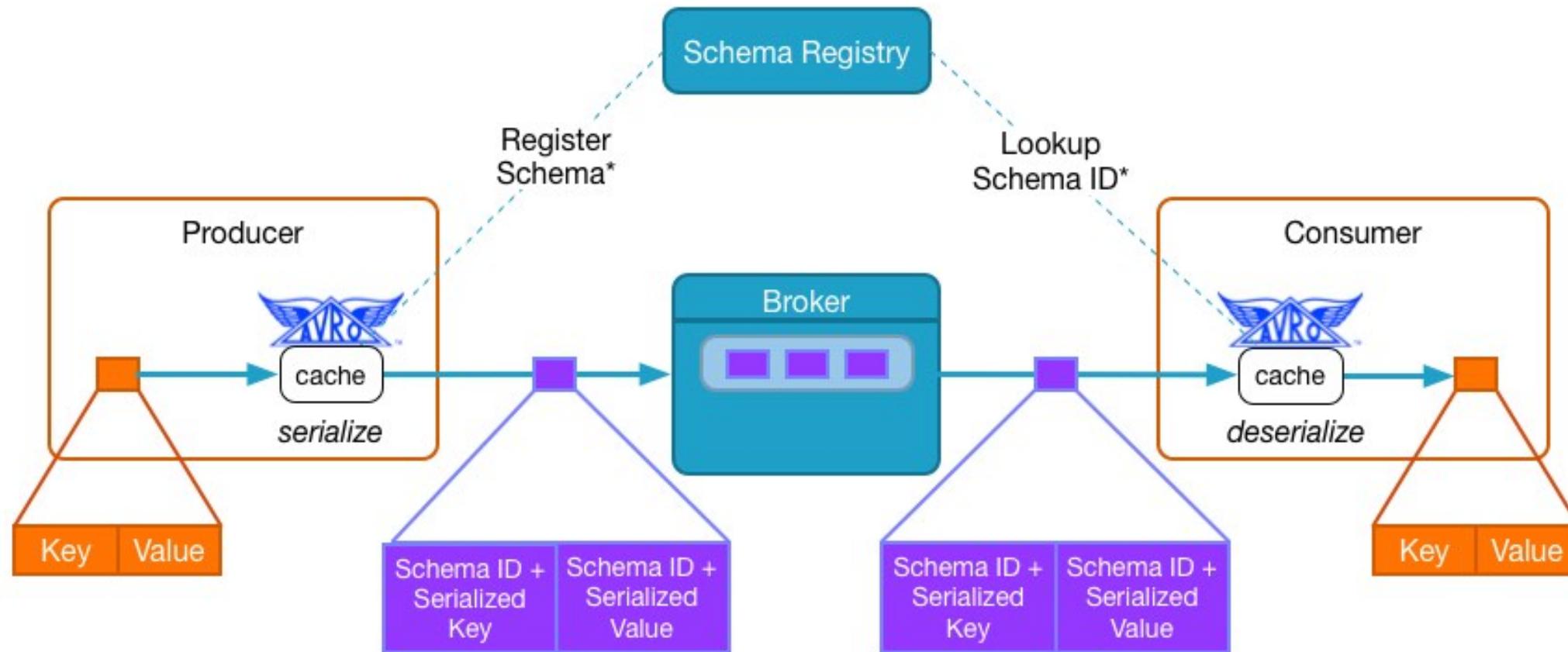
There's a Cache Too!



Schema Registry: The Consumer End



Schema Registry: The Whole Picture



Schema-Enabled Java Producer Example

Writing a producer that supports Schema Registry and uses Avro isn't much different from before. Two new steps...

First, we must use the Avro serializer for the key or value, e.g.,

```
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, KafkaAvroSerializer.class);
```

Then, we must also configure the producer to know where our schema registry is:

```
props.put(AbstractKafkaAvroSerDeConfig.SCHEMA_REGISTRY_URL_CONFIG,  
        "http://schema-registry1:8081");
```

We otherwise use a Java class generated by Avro. You'll experience how to generate this in lab.



Everything else is the same!

Schema Enabled Java Consumer Example

Three (really two) new things here...

We use the Avro deserializer instead of serializer:

```
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,  
         KafkaAvroDeserializer.class);
```

We still need to identify where SR is:

```
props.put(AbstractKafkaAvroSerDeConfig.SCHEMA_REGISTRY_URL_CONFIG,  
          "http://schema-registry1:8081");
```

We need to tell our consumer to look for schema IDs:

```
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "true");
```

Setting Schema Evolution: Modes Reference

For reference, Schema Registry supports the following schema evolution modes:

Mode	Explanation
BACKWARD	Consumers expecting data using new form of schema can process data using previous version of schema
BACKWARD_TRANSITIVE	Consumers expecting data using new form of schema can process data using any previous version of schema
FORWARD	Consumers expecting data using previous form of schema can process data using next version of schema
FORWARD_TRANSITIVE	Consumers expecting data using previous form of schema can process data using any later version of schema

Setting Schema Evolution: Modes Reference, continued

Mode	Explanation
FULL	Both BACKWARD and FORWARD
FULL_TRANSITIVE	Both BACKWARD_TRANSITIVE and FORWARD_TRANSITIVE
NONE	No compatibility checking

Setting Schema Evolution: Command

We can use a command like the following to set schema evolution:

```
# This example disables compatibility checking using "NONE"  
$ curl -X PUT -i -H "Content-Type: application/vnd.schemaregistry.v1+json" \  
--data '{"compatibility": "NONE"}' \  
http://schemaregistry1:8081/config/my_topic-value
```

HTTP/1.1 200 OK

Content-Type: application/vnd.schemaregistry.v1+json
{"compatibility": "NONE"}

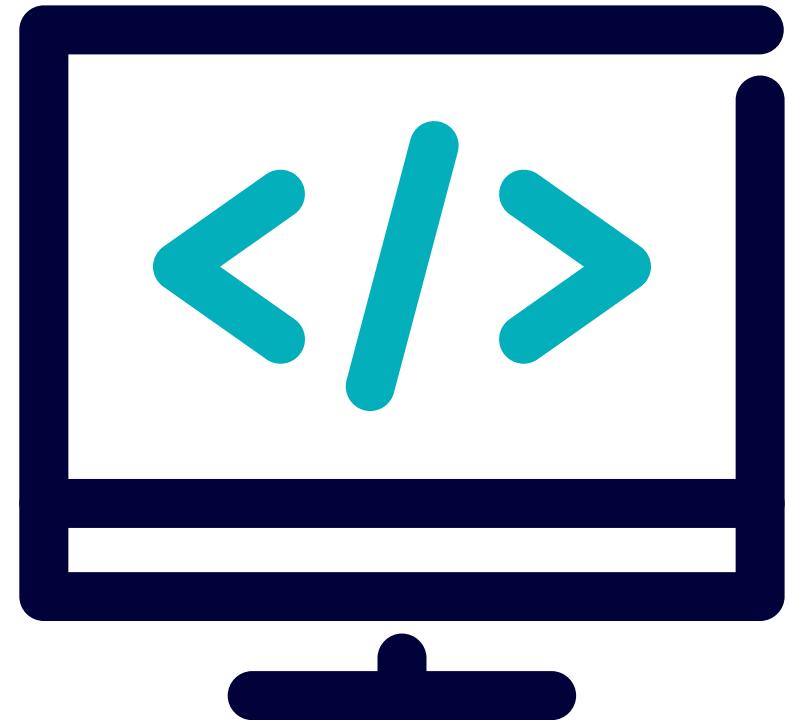
Support for Schema Registry

- Part of Confluent Platform
- Supported by clients
 - Java clients
 - `librdkafka` clients except Go
- Supported by Confluent REST Proxy
- Supported by Kafka Streams
 - ...and, in turn, ksqlDB
- Supported by Kafka Connect

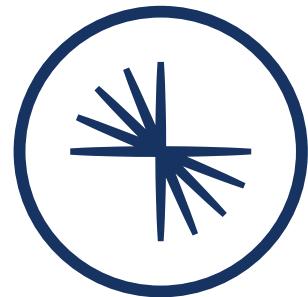
Lab: Schema Registry, Avro Producer and Consumer

Please work on **Lab 7a: Schema Registry, Avro Producer and Consumer**

Refer to the Exercise Guide



08: Introduction to Streaming and Kafka Streams



CONFLUENT
Global Education

Module Overview



This module contains four lessons:

- a. What Can You Do with Streaming Applications?
- b. What is Kafka Streams?
- c. A Taste of the Kafka Streams DSL
- d. How Do You Put Together a Kafka Streams App?

Where this fits in:

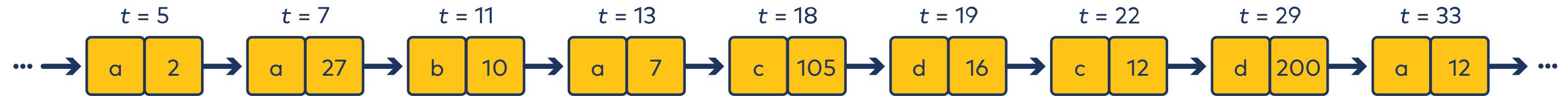
- Hard Prerequisite: Starting with Consumers
- Recommended Prerequisite: Groups and Consumers in Practice
- Recommended Follow-Up: Introduction to ksqlDB

08a: What Can You Do with Streaming Applications?

Description

Defining what streams are with a concrete conceptual example and relating streams to topics. Comparing streams and tables and which to use when. How time drives streams instead of offsets and the basic idea of windowing.

A Sample Stream



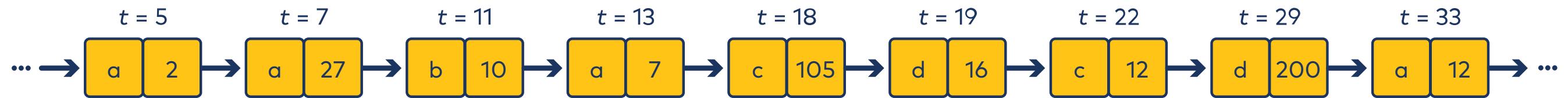
Note:

- **Events** are key/value pairs
- Ours will be sourced from Kafka topics
- **Time** is a big deal

Operating on the Stream

We can do **stateless operations** on a stream, like filtering.

What would happen if we filtered to keep those records whose value exceeded 50?

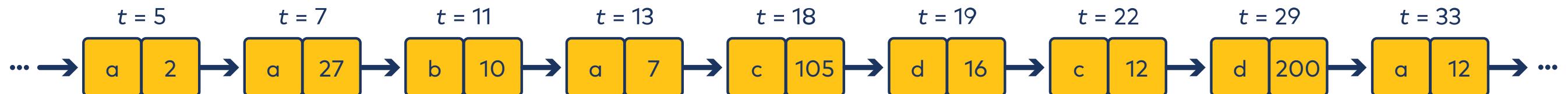


Operating on the Stream

We can do **stateless operations** on a stream, like filtering.

What would happen if we filtered to keep those records whose value exceeded 50?

Input stream:



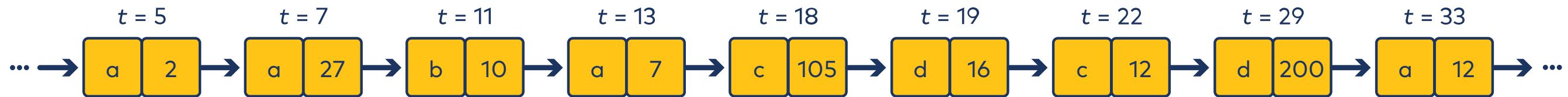
Output stream:



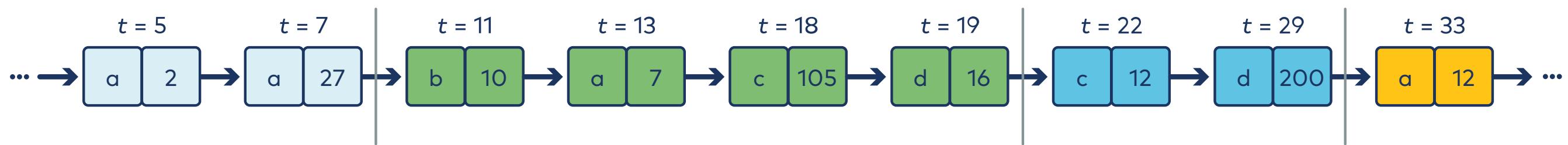
Windowing the Stream

We can split up time into windows.

Here's our input stream:



Here's the same stream, divided into windows of size 10:

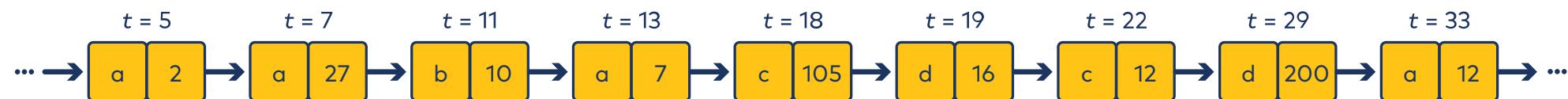


Stream-Table Duality

We can view a stream as a table.

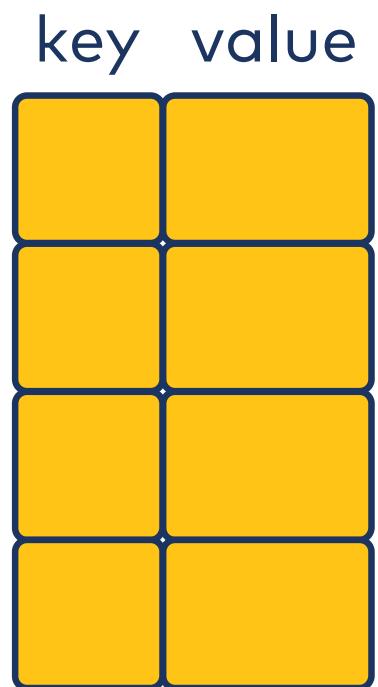
Stream

Records are events in time



Table

Records are updates to same-key table entries



Streams vs. Tables

	Stream	Table
Use for...	Working with events in time/history	Storing status/state
Example 1	Driver positions	Driver profiles
Example 2	Order status changes	Orders, Customers
Example 3	Ledger of sales	Sales totals

Activity: Choosing Between Streams and Tables



Quick Check

For each concept described, would it make more sense to represent it as a stream or as a table?

- a. number of purchases per customer
- b. heartbeat readings per patient coming in every 30 seconds
- c. number of high heart rate events per patient observed
- d. current addresses of known restaurants

Virtual Classroom Poll:



use a stream



use a table

08b: What is Kafka Streams?

Description

The consume-process-produce model. How Kafka Streams fits into the Kafka ecosystem, what a streaming topology is, and how data gets back to Kafka.
Immutability of streams. Stateless vs. stateful processing conceptually.

The Consume-Process-Produce Model

Idea:

1. **Consume** data from a topic
2. Do something with that data, i.e. **Process**
3. Generate a new record or records based on the processing and **Produce** them to a topic

Kafka Streams

- Separate application
- Tied to a Kafka cluster
- Groups
 - Objects `KStream` and `KTable`
 - Sourced from a Kafka topic
 - Acts as a Consumer
 - Data goes out to a different Kafka topic
 - Acts as a Producer

Unbounded, Immutable Streams

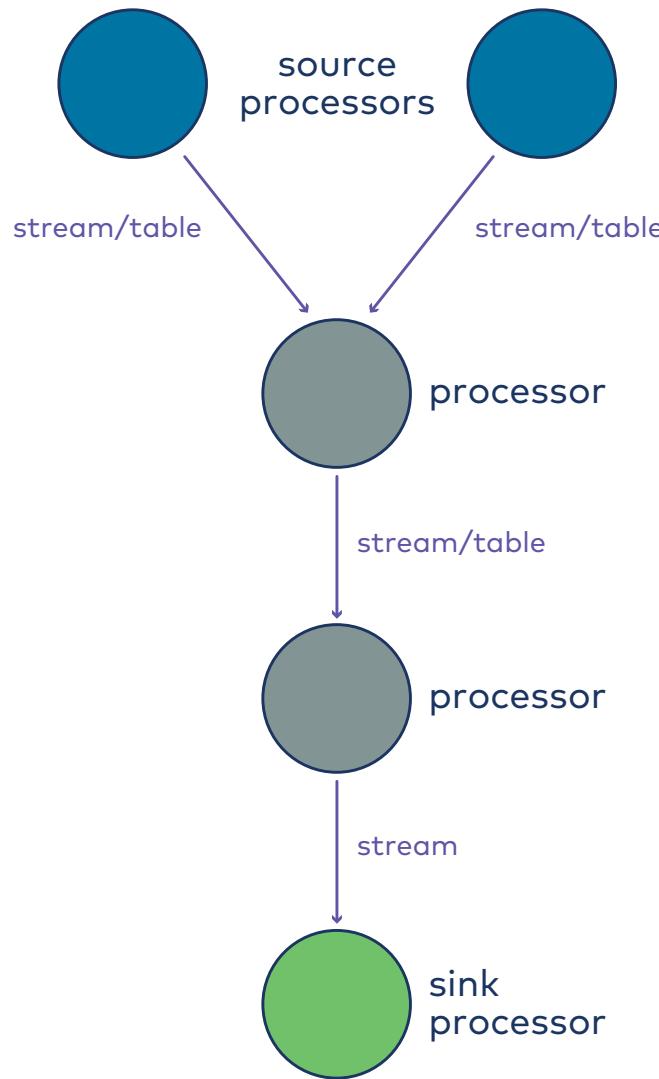
- Streams are unbounded
 - Sourced from a topic
 - As the topic receives new records, so does the stream
- Streams are immutable
 - Never change a stream
 - Instead, output a **new** stream from operations

Processors

- A **stream processor** operates on stream or table
- One operation
- Examples
 - Filter a stream
 - Get a table where all cost values have tax added
- Special processors:
 - **Source** processor:
 - Read from a Kafka topic
 - Generate a `KStream` or `KTable`
 - **Sink** processor:
 - Take in a `KStream`
 - Produce each event to a Kafka topic

Processor Topologies

Processors are put together to form a **processor topology**



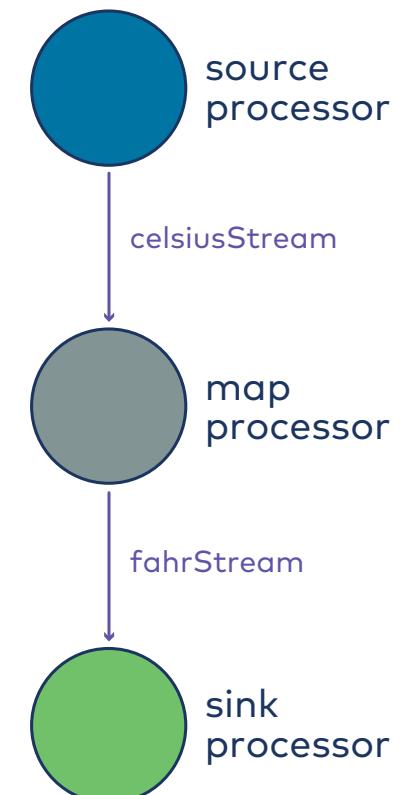
Example Stateless Application Topology

Kafka cluster has topic:

- `temp_readings`
 - keys: postal codes
 - values: temperatures in degrees C

Topology:

- Source processor creates `KStream celsiusStream` from `temp_readings`
- Processor maps each temperature in degrees C to degrees F
 - Creates `KStream fahrStream`
- Sink processor writes `fahrStream` to Kafka topic `temp_readings_fahr`



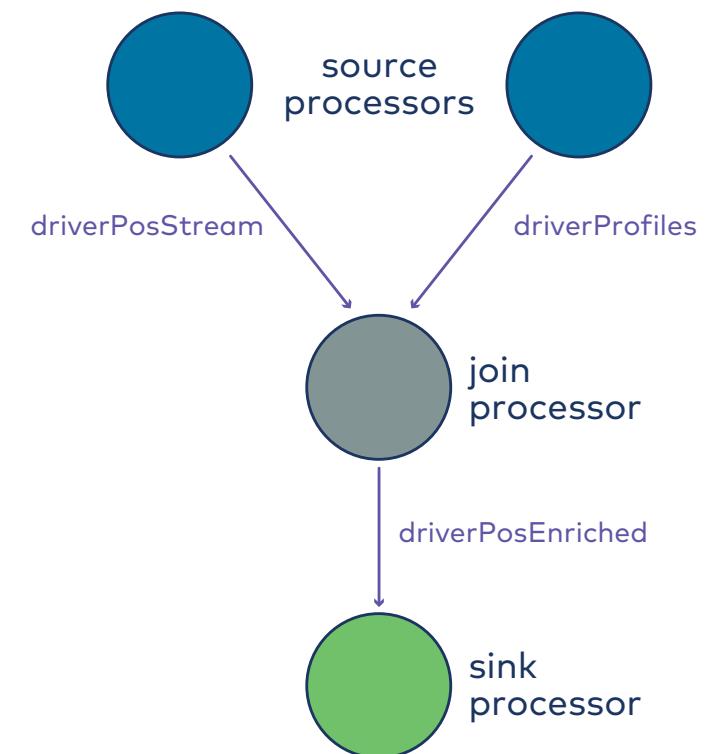
Another Example Application Topology

Kafka cluster has topics:

- `driver_profiles`
- `driver_positions`

Topology:

- Source processors create...
 - KStream `driverPosStream` from `driver_positions`
 - KTable `driverProfiles` from `driver_profiles`
- Processor joins `driverPosStream` with `driver_profiles`
→ Creates stream `driverPosEnriched`
- Sink processor writes `driverPosEnriched` to Kafka topic `driver_positions_enriched`



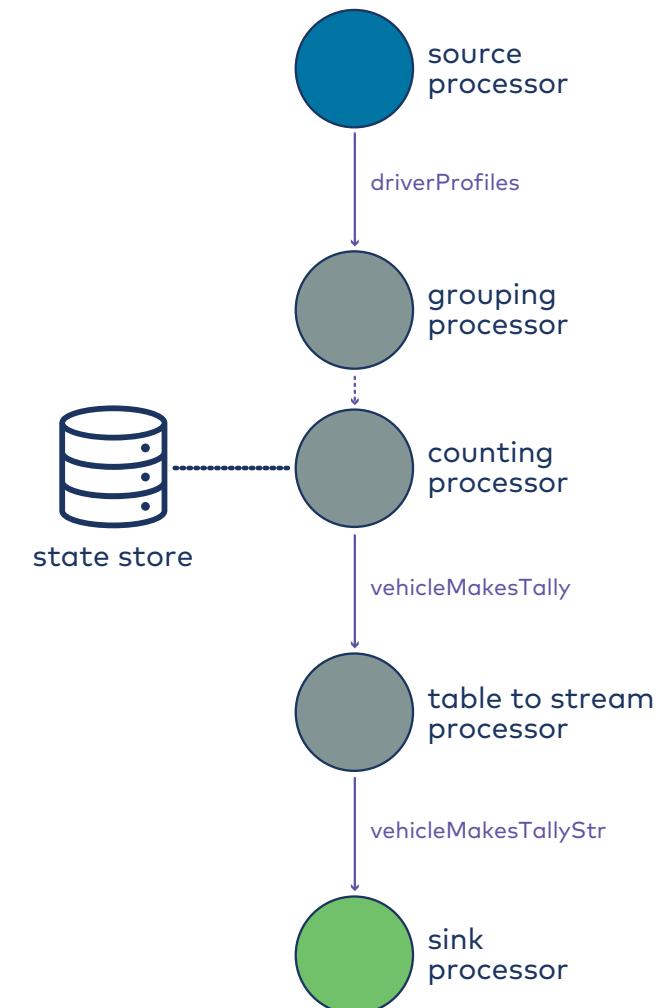
Example Stateful Application Topology

Kafka cluster has topic:

- `driver_profiles`

Topology:

- Source processor creates `KTable driverProfiles` from `driver_profiles`
- Processors group `driverProfiles` by make of vehicle and count number in each group
 - Creates `KTable vehicleMakesTally`
- Processor generates equivalent `KStream vehicleMakesTallyStr` from `vehicle_makes_tally`
- Sink processor writes `vehicleMakesTallyStr` to Kafka topic `vehicle_makes_tally_topic`



Activity: Reviewing Kafka Streams Concepts



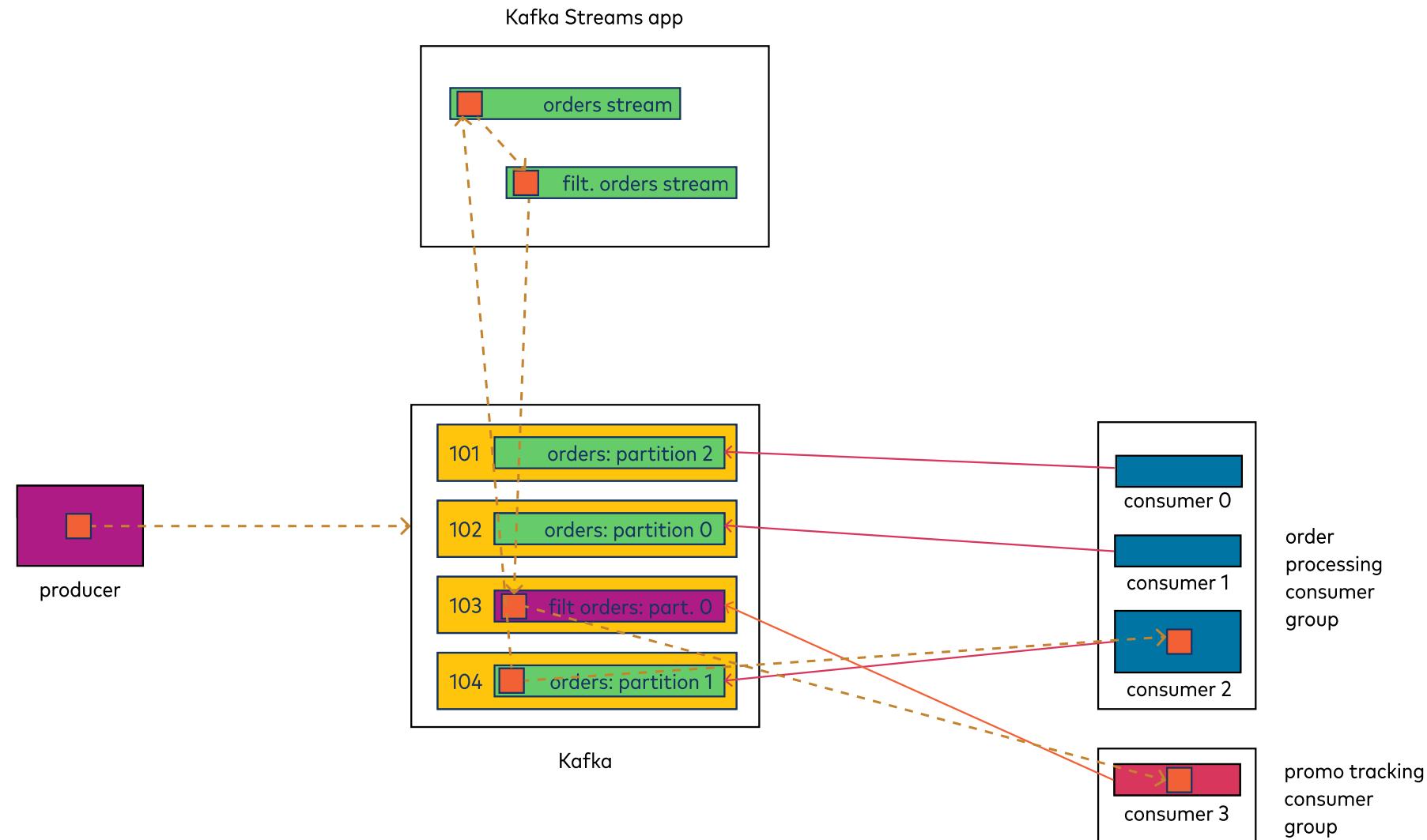
Quick Knowledge Check: Myth or Fact?

1. When we say processor in this context, we refer to all the logic of a Kafka Streams application
2. Kafka Streams applications act as consumers
3. Kafka Streams applications run on Kafka brokers
4. The input data to Kafka Streams can directly come from anywhere
5. Kafka Streams applications act as producers
6. Streams can be changed
7. A processor can take more than one stream as its input



A Step Beyond

More on Data Flow with Kafka Streams Apps



08c: A Taste of the Kafka Streams DSL

Description

Defining the DSL. Getting streams from Kafka topics in code. Examples of some stateless operations with code: `filter`, `map(Values)`, `flatMap(Values)`. Stateful operation examples: `groupBy` and `count`. Writing a stream back to Kafka.

Interacting with Kafka Streams: the DSL

Two ways to interact with Kafka Streams:

- Domain Specific Language (DSL) ← our focus
- Processor API (PAPI)

Getting Streams and Tables from Kafka

This is how we implement a source processor.

Use the `StreamsBuilder` class and operations `stream()` and `table()`.

Objects:

```
StreamsBuilder builder;  
KStream<Integer, String> driverPosStream;  
KTable<Integer, String> driverProfiles;
```

Creating entities:

```
builder = new StreamsBuilder();  
driverPosStream = builder.stream("driver_positions");  
driverProfiles = builder.table("driver_profiles");
```

Filtering a Stream (Stateless Operation)

filter

Creates a new `KStream` containing only records from the input `KStream` which meet some specified criteria

```
largePurchases = purchases.filter((key,value) -> value.amount > 50.0);
```

Mapping (Stateless Operation)

mapValues

Creates a new `KStream` by transforming the value of each element in the input stream into a different element in the output stream. Use when only transforming the value.

```
fahrStream = celsiusStream.mapValues(value -> value*9.0/5.0 + 32);
```

map

Creates a new `KStream` by transforming each element in the input stream into a different element in the output stream. Use this if you must change the key.

```
fahrStreamByCity = celsiusStream.map((key,value)
                                      ->
                                      new KeyValue<>(cityFromZIP(key),
                                                       value*9.0/5.0 + 32));
```

Mapping, Part 2 (Stateless Operation)

flatMapValues

Creates a new `KStream` by transforming the value of each element in the input stream into zero or more elements in the output stream. Only changes value.

`factoredStream`

```
= numbersStream.flatMapValues(value -> getPrimeFactors(value));
```

flatMap

Creates a new `KStream` by transforming each element in the input stream into zero or more elements in the output stream. Use this if you must change the key.

Activity: Transforming a Bank Account Stream



Suppose you have a stream where:

- keys of events are bank account numbers
- values of events are delimited strings containing several transactions for the account whose number is the key

You want a corresponding stream where each event:

- has a key that is a bank account number
- has a value that is a *single* transaction

Your quest:

1. Among all the functions we looked at in the DSL, which can solve this? Which can't? Which is best?
2. What would you do if you want the stream to contain only those transactions involving over \$100?

Grouping and Counting (Stateful Example)

count

Counts the number of instances of each key in the stream; results in a new, ever-updating KTable

```
stream.groupByKey()  
    .count()
```

Writing A Stream to Kafka

To implement a sink processor, call the `to` method on a `KStream`.

Example:

```
fahrStream.to("temp_readings_fahr");
```



A `KTable` needs to be converted to a `KStream` first

08d: How Do You Put Together a Kafka Streams App?

Description

The five parts of a Kafka Streams application overall. Code examples of three parts we haven't seen, then a full example of a stateless application and a stateful one.

Kafka Streams Application Anatomy

Five parts:

1. Imports
2. Configuration
3. Topology
4. Create and start Kafka Streams application
5. Graceful shutdown

Running Example

In a prior lesson, we solved this problem...

Suppose you have a stream where:

- keys of events are bank account numbers
- values of events are delimited strings containing several transactions for the account whose number is the key

You want a corresponding stream where each event:

- has a key that is a bank account number
- has a value that is a *single* transaction

We will be working in the same context for all of our examples in this lesson.

SerDes

One quick note before we dive in...

- With producers, we need to specify serializers
- With consumers, we need to specify deserializers
- Streams apps both consume **and** produce...
- ... need both
- ... hence **SerDes = Serializers and Deserializers**

Example: Defining Configuration

```
1  public static Properties getConfig()
2  {
3      Properties config = new Properties();
4
5      // Give the application a name, which must be unique in the cluster
6      config.put(StreamsConfig.APPLICATION_ID_CONFIG,
7                  "simple-streams-example");
8
9      // Connect to cluster
10     config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
11                  "broker-1:9092, broker-2:9092");
12
13     // Specify default (de)serializers for record keys and for record values.
14     config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
15                  Serdes.Integer().getClass());
16     config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
17                  Serdes.String().getClass());
18
19     return config;
20 }
```

Example: Defining Topology (Stateless)

```
1  public static Topology getTopology()
2  {
3      StreamsBuilder builder;
4      KStream<Integer, String> delimTransStream;
5      KStream<Integer, String> indTransStream;
6      Topology result;
7
8      builder = new StreamsBuilder();
9
10     delimTransStream = builder.stream("delim_transactions_topic");
11
12     indTransStream = delimTransStream.flatMapValues(value -> split(value, "|"));
13
14     indTransStream.to("individual_transactions_topic");
15
16     result = builder.build();
17
18     return result;
19 }
```

Stateless Example: Main Program

Here we bring together the prior two results and do Steps 4 and 5 of the anatomy:

```
1  public static void main(String[] args) throws Exception
2  {
3      // Create a streams application based on config & topology defined already
4      KafkaStreams streams = new KafkaStreams(getTopology(), getConfig());
5
6      // Run the Streams application via `start()`
7      streams.start();
8
9      // Stop the application gracefully
10     Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
11 }
```

Example: Defining Topology (Stateful)

Now let's look at example of stateful processing. We use the same configuration, but a new topology.

```
1  public static Topology getTopology()
2  {
3      StreamsBuilder builder;
4      KStream<Integer, String> delimTransStream;
5      KStream<Integer, String> indTransStream;
6      KTable<Integer, Integer> transByAcctTally;
7      Topology result;
8
9      builder = new StreamsBuilder();
10
11     // Source processor: get stream from Kafka topic
12     delimTransStream = builder.stream("delim_transactions_topic");
13
14     // Internal processor: break up the stream
15     indTransStream = delimTransStream.flatMapValues(value -> split(value, "|"));
```

Example: Defining Topology (Stateful), continued

```
16     // Group transactions by account number and count
17     transByAcctTally = indTransStream.groupByKey()
18         .count();
19
20     // Sink processor: convert table to stream, then write to new Kafka topic
21     transByAcctTally.toStream()
22         .to("acct_activity_tally_topic",
23             Produced.with(Serdes.Integer(), Serdes.Integer()));
24
25     // Generate and return topology
26     result = builder.build();
27     return result;
28 }
```

Stateful Example: Main Program

Our main program is identical to the prior example; only the topology is different.

```
1  public static void main(String[] args) throws Exception
2  {
3      // Create a streams application based on config & topology defined already
4      KafkaStreams streams = new KafkaStreams(getTopology(), getConfig());
5
6      // Run the Streams application via `start()`
7      streams.start();
8
9      // Stop the application gracefully
10     Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
11 }
```

Going Further

- This is just a taste of Kafka Streams
- Consider more training:
 - Instructor-led course Stream Processing using Apache Kafka® Streams and Confluent ksqlDB
- Consider reading:
 - Documentation on our website
 - *Kafka Streams in Action* book (William Bejeck)

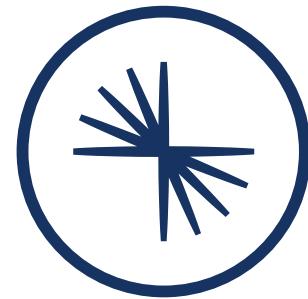
Lab: Kafka Streams

Please work on **Lab 8a: Kafka Streams**

Refer to the Exercise Guide



09: Introduction to ksqlDB



CONFLUENT
Global Education

Module Overview



This module contains four lessons:

- a. What Does a Kafka Streams App Look Like in ksqlDB?
- b. What are the Basic Ideas You Should Know about ksqlDB?
- c. How Do Windows Work?
- d. How Do You Join Data from Different Topics, Streams, and Tables?

Where this fits in:

- Hard Prerequisite: Introduction to Streaming and Kafka Streams
- Recommended Follow-Up: Kafka Connect

09a: What Does a Kafka Streams App Look Like in ksqlDB?

Description

Creating a ksqlDB app analogous to an example Kafka streams application.

Recall...

Last module, we arrived at a Kafka Streams application to read in grouped transactions from a topic, split them, tally by account number, and write out to Kafka. More practically, maybe we read in individual transactions, so here's a slightly simplified version of that application:

```
1 public class KStreamEx
2 {
3     public static Properties getConfig()
4     {
5         Properties config = new Properties();
6
7         // Give the application a name, which must be unique in the cluster
8         config.put(StreamsConfig.APPLICATION_ID_CONFIG,
9             "simple-streams-example");
10
11        // Connect to cluster
12        config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
13            "broker-1:9092, broker-2:9092");
```

More of That Code...

```
14     // Specify default (de)serializers for record keys and for record values.  
15     config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,  
16         Serdes.Integer().getClass());  
17     config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,  
18         Serdes.String().getClass());  
19  
20     return config;  
21 }  
22  
23 public static Topology getTopology()  
24 {  
25     StreamsBuilder builder;  
26     KStream<Integer, String> indTransStream;  
27     KTable<Integer, Integer> transByAcctTally;  
28     Topology result;  
29  
30     builder = new StreamsBuilder();  
31  
32     // Source processor: get stream from Kafka topic  
33     indTransStream = builder.stream("transactions_topic");
```

Still More of That Code...

```
33     // Group transactions by account number and count
34     transByAcctTally = indTransStream.groupByKey()
35         .count();
36
37     // Sink processor: convert table to stream, then write to new Kafka topic
38     transByAcctTally.toStream()
39         .to("acct_activity_tally_topic",
40             Produced.with(Serdes.Integer(), Serdes.Integer()));
41
42     // Generate and return topology
43     result = builder.build();
44     return result;
45 }
46
47 public static void main(String[] args) throws Exception
48 {
49     // Create a streams application based on config & topology defined already
50     KafkaStreams streams = new KafkaStreams(getTopology(), getConfig());
```

The Rest of That Code...

```
52     // Run the Streams application via `start()`
53     streams.start();
54
55     // Stop the application gracefully
56     Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
57 }
58 }
```

Let's do the same thing with ksqlDB!

Get a Stream from a Kafka Topic

Here's how we get our stream from our source Kafka topic:

```
CREATE STREAM ind_trans_stream (acct_id INT KEY, amount DOUBLE, details VARCHAR)
    WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'transactions_topic');
```

Notes:

- In ksqlDB, a `STREAM` is the same thing as a Kafka Streams `KStream`
- We'll use the `CREATE STREAM ... WITH` construct to do what we did with `StreamsBuilder` and `builder.stream()`

Write a SQL Query to Group & Count

In our streaming topology in Kafka Streams, we grouped by key and counted. Here's the equivalent in ksqlDB:

```
SELECT acct_id, count(*)  
FROM ind_trans_stream  
GROUP BY acct_id;
```

Indeed, this looks just like SQL. But... we also stored our results in a table. Let's do that too:

```
CREATE TABLE trans_by_acct_tally AS  
SELECT acct_id, count(*)  
FROM ind_trans_stream  
GROUP BY acct_id;
```

Here again you see a `CREATE` statement, this time creating as the result of a query.

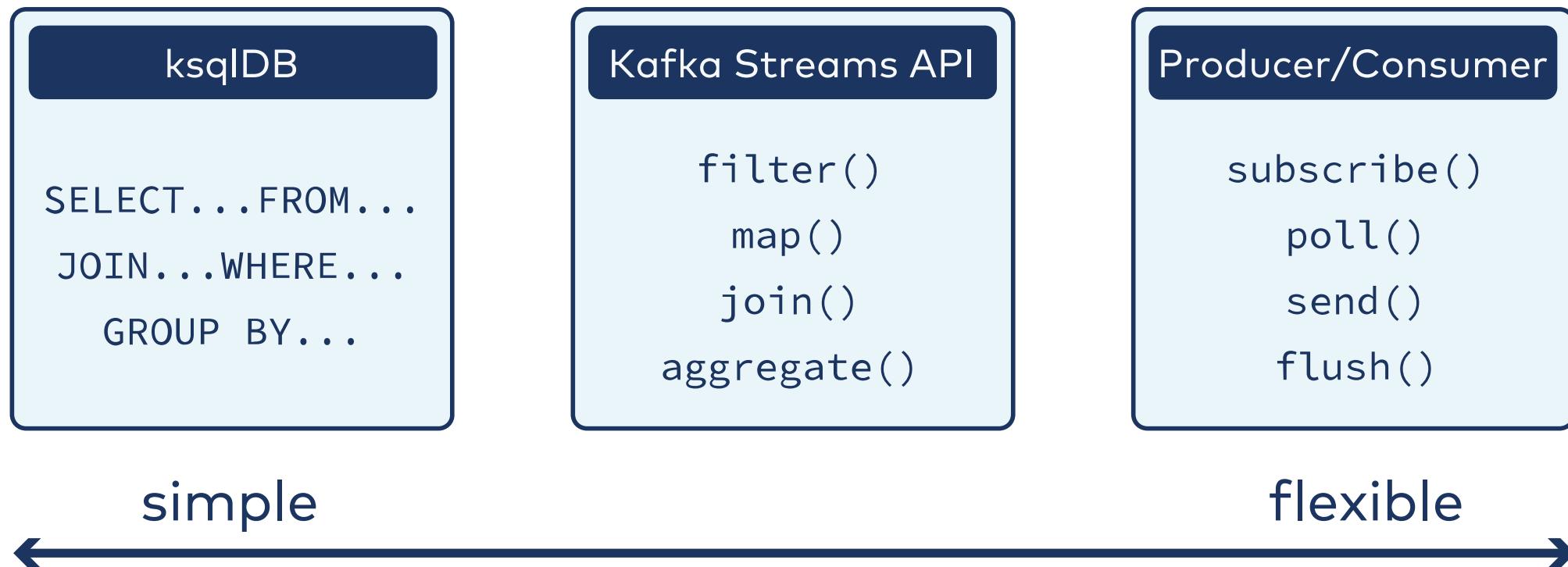
09b: What are the Basic Ideas You Should Know about ksqlDB?

Description

What ksqlDB is, how it fits in, persistent vs. non-persistent queries, push vs. pull queries, and summarizing basic syntax.

Introducing ksqlDB

So, we've seen we can do some things we can do with Kafka Streams simpler with ksqlDB.



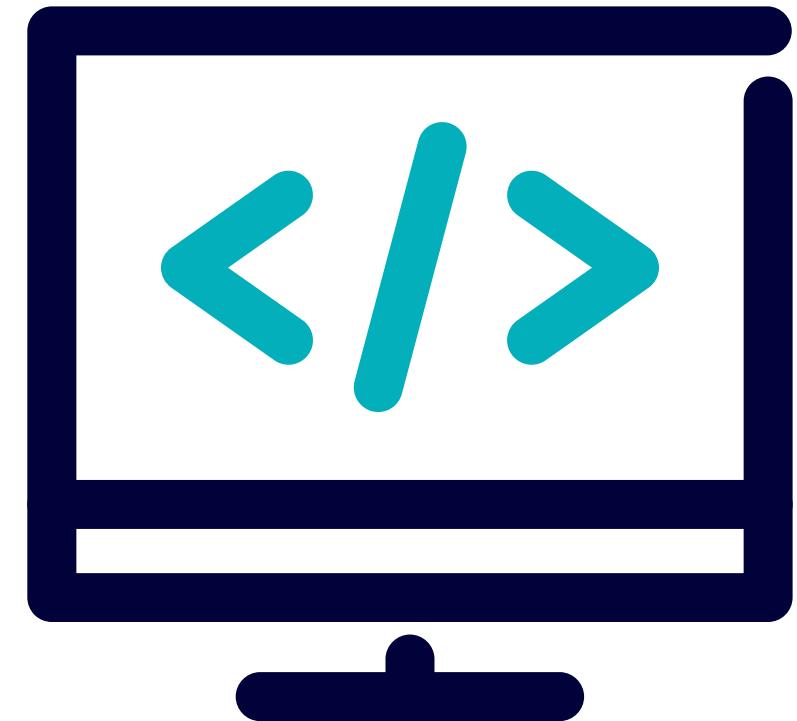
ksqldb SQL

- Designed to be accessible to you if you've worked with most flavors of SQL
- Standard `SELECT`, `FROM` clauses
- Standard aggregation with `GROUP BY`
- Standard filtering with `WHERE` and `HAVING`
- Many popular scalar functions
- More functions specific to ksqldb, e.g., `EXTRACTJSONFIELD(...)`
- Working with `STREAM` and `TABLE` objects, not just tables
- Saving `STREAM` and `TABLE` objects...
- Windowing capabilities...
- Joining capabilities...

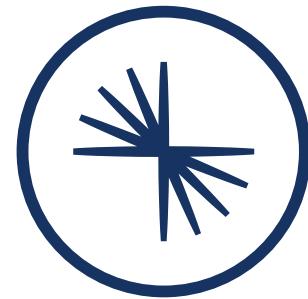
Lab: ksqlDB Exploration

Please work on **Lab 9a: ksqlDB Exploration**

Refer to the Exercise Guide



10: Starting with Kafka Connect



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. What Can You Do with Kafka Connect?
- b. How Do You Configure Workers and Connectors?
- c. Deep Dive into a Connector & Finding Connectors

Where this fits in:

- Hard Prerequisite: Groups and Consumers in Practice
- Recommended Prerequisite: Introduction to ksqlDB
- Recommended Follow-Up: Applying Kafka Connect

10a: What Can You Do with Kafka Connect?

Description

Motivating what Connect can do and why to use it over self-made solutions. Motivating how it can “factor out” common behavior yet leverage Connectors. Connectors vs. tasks vs. workers. Relating Connect to other components of Kafka and how it works at a high level, e.g., scalability, converters, offsets.

Wanted: Copy Database Table into Kafka

Given: Customer information in a table in a MySQL database.

Goal: Get that data into a Kafka topic `customers`.

Question: Could you do this using what you've learned in this course so far and anything else you know about Java?

Wanted: Copy CSV File Info Into Kafka

Given: Weather station information stored in a flat file, where each row describes one station.

Goal: Get that data into a Kafka topic `stations`.

Question: Could you do this using what you've learned in this course so far and anything else you know about Java?

Challenges with Writing Your Own Producer to Copy Data

So, you can write your own producers to copy data from another system to Kafka.

You can write your own consumers to copy data from Kafka to another system.



BUT...

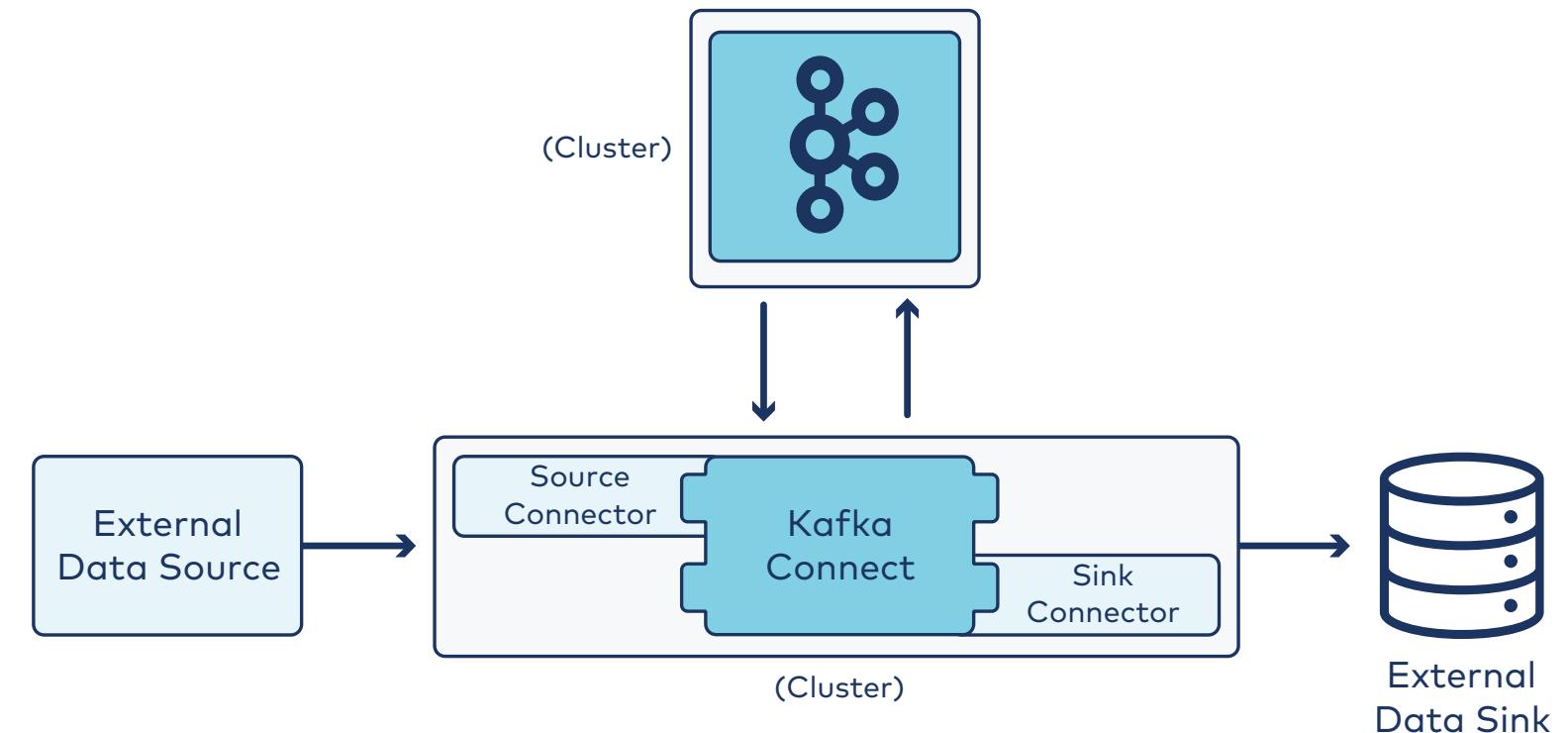
- It takes time
- You might miss an edge case
- You might miss a bug
- You'd write an application for each external system.

Kafka Connect to the Rescue!

Kafka Connect does the work for us!

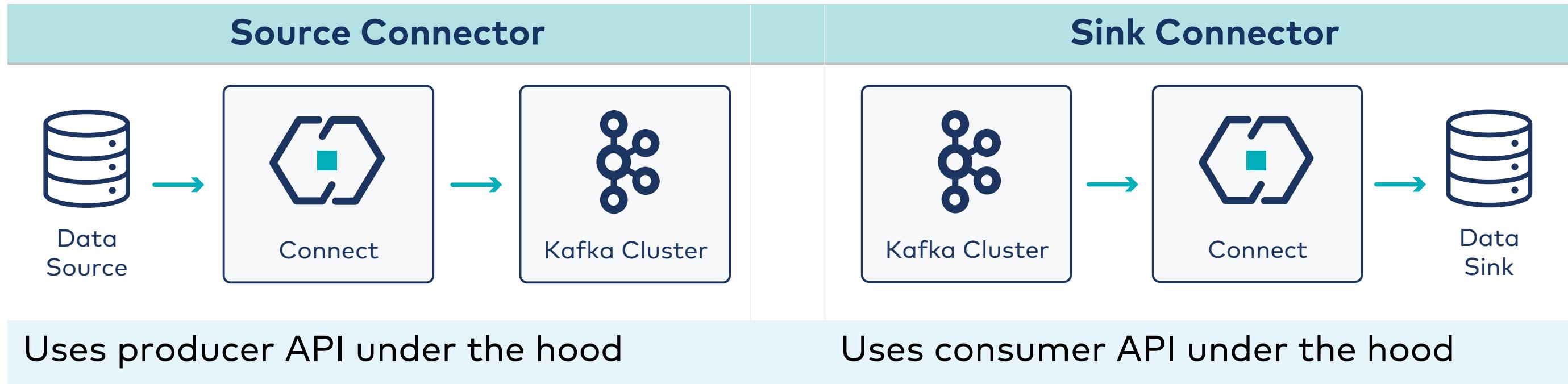
All copying behavior is in Kafka Connect.

Plugins called **Connectors** contain the logic specific to particular external systems.



Sources and Sinks

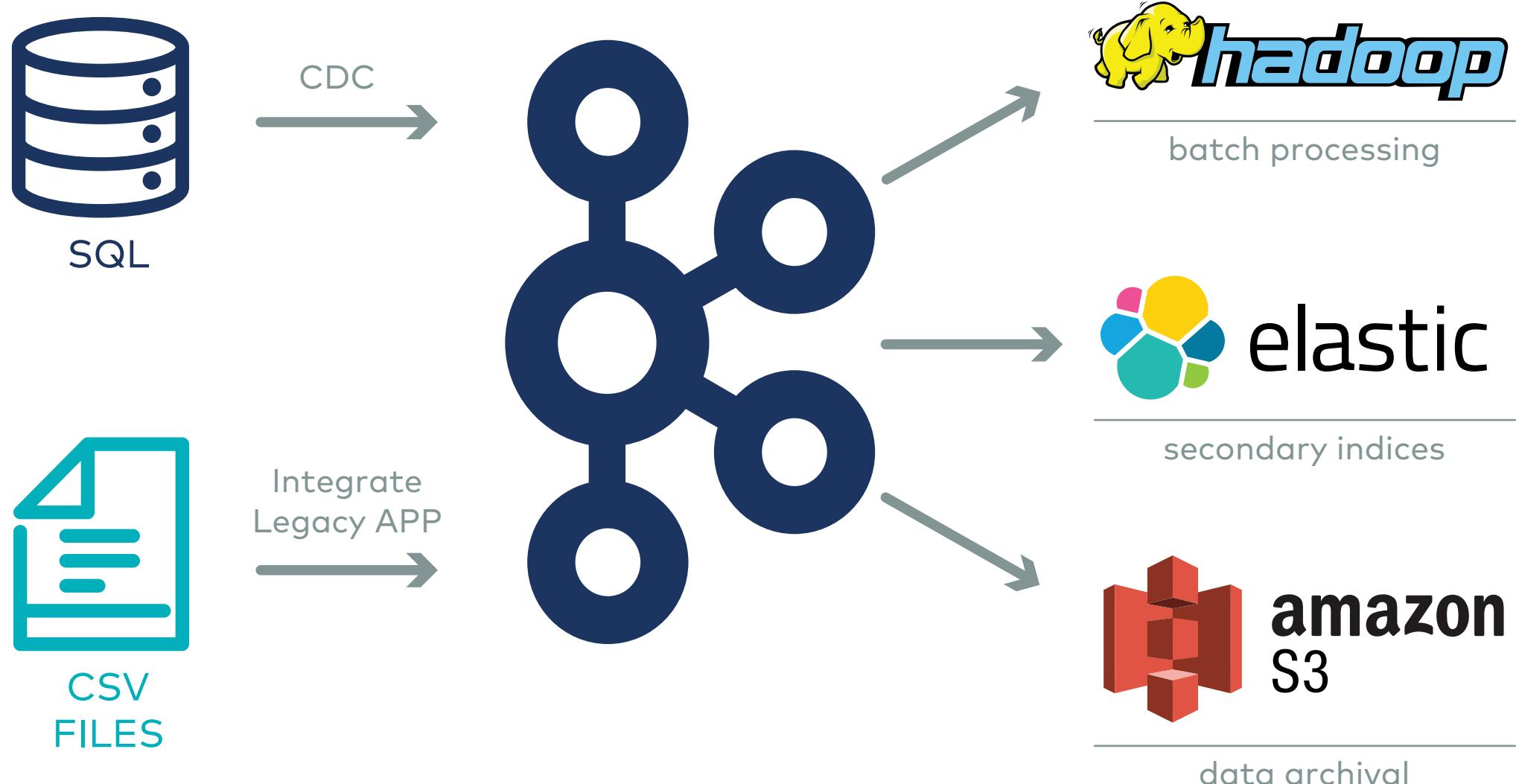
Two kinds of connectors...



Powered by Kafka, and Behaving Like Kafka

- Same group management protocol
 - Failure detection
 - Scaling up and down
- Producer and consumer under the hood
- Sink connectors maintain consumer offsets → **sink offsets**
- Source connectors track **source offsets**
- Data must be serialized and deserialized → **converters**

Use Cases



10b: How Do You Configure Workers and Connectors?

Description

Configuration of workers in distributed mode and configuration of connectors in general. Quick overview of standalone mode differences.

Configuring Connectors

Name	Description	Default
name	Connector's unique name	
connector.class	Name of the Java bytecodes file for the connector	
tasks.max	Maximum number of tasks to create - if possible	1
key.converter	Converter to (de)serialize keys	(worker setting)
value.converter	Converter to (de)serialize values	(worker setting)
topics	For sink connectors only , comma-separated list of topics to consume from	

10c: Deep Dive into a Connector & Finding Connectors

Description

Details of the JDBC Source Connector, configuration details, working through why one would do certain configs with examples. Finding Connectors on Confluent Hub.

JDBC Source Connector

- Java Database Connectivity (JDBC) API is common amongst databases.
- JDBC Source Connector is a great way to get database tables into Kafka topics.
- JDBC Source periodically polls a relational database for new or recently modified rows.
 - Creates a record for each row, and Produces that record as a Kafka message.
- Each table gets its own Kafka topic.
- New and deleted tables are handled automatically.

Configuration

Property	Description
<code>connection.url</code>	The JDBC connection URL for the database
<code>mode</code>	The mode for detecting table changes. Options are <code>bulk</code> , <code>incrementing</code> , <code>timestamp</code> , <code>timestamp+incrementing</code>
<code>query</code>	The custom query to run, if specified
<code>poll.interval.ms</code>	The frequency in milliseconds to poll for new data in each table (Default: 5000)
<code>topic.prefix</code>	Prefix to prepend to table names to generate the Kafka Topic name
<code>table.blacklist</code>	A list of tables to ignore and not import.
<code>table.whitelist</code>	A list of tables to import.

JDBC Source Connector Config Example

```
1 {
2   "name": "Driver-Connector",
3   "config": {
4     "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
5     "connection.url": "jdbc:postgresql://postgres:5432/postgres",
6     "connection.user": "postgres",
7     "table.whitelist": "driver",
8     "topic.prefix": "",
9     "mode": "timestamp+incrementing",
10    "incrementing.column.name": "id",
11    "timestamp.column.name": "timestamp",
12    "table.types": "TABLE",
13    "numeric.mapping": "best_fit",
14  }
15 }
```

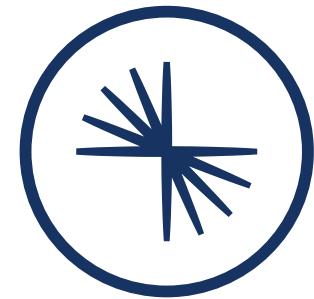
Other Connectors

Search Confluent Hub at confluent.io/hub for connectors!

The screenshot shows the Confluent Hub homepage with a search bar and a list of results. On the left, there are filters for Plugin type (Sink, Source, Transform, Converter), Enterprise support (Confluent supported, Partner supported, None), Verification (Confluent built, Confluent Tested, Verified gold, Verified standard, None), and License (Commercial, Free). The results section shows two connectors: "Kafka Connect GCP Pub-Sub" (Source Connector) and "Kafka Connect S3" (Sink Connector). Both connectors are marked as available on Confluent Cloud.

Connector	Type	Description	Status
Kafka Connect GCP Pub-Sub	SOURCE CONNECTOR	A Kafka Connect plugin for GCP Pub-Sub	Available fully-managed on Confluent Cloud
Kafka Connect S3	SINK CONNECTOR	The S3 connector, currently available as a sink, allows you to export data from Kafka topics to S3 objects in either Avro or JSON formats	Available fully-managed on Confluent Cloud

11: Applying Kafka Connect



CONFLUENT
Global Education

Module Overview



This module contains a hands-on lab for Connect and one lesson:

- a. Full Solutions Involving Other Systems

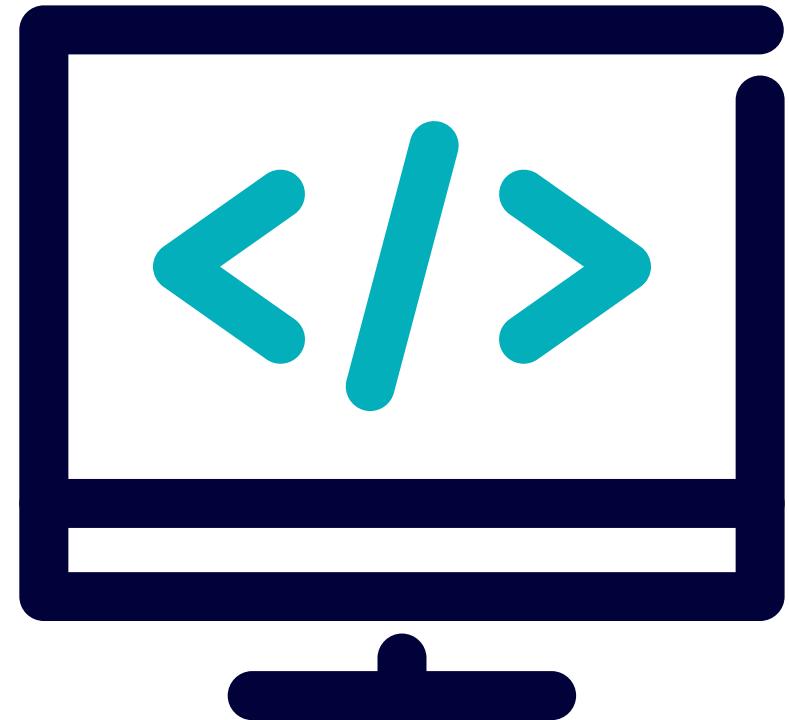
Where this fits in:

- Hard Prerequisite: Starting with Kafka Connect

Lab: Kafka Connect - Database to Kafka

Please work on **Lab 11a: Kafka Connect - Database to Kafka**

Refer to the Exercise Guide



11a: Full Solutions Involving Other Systems

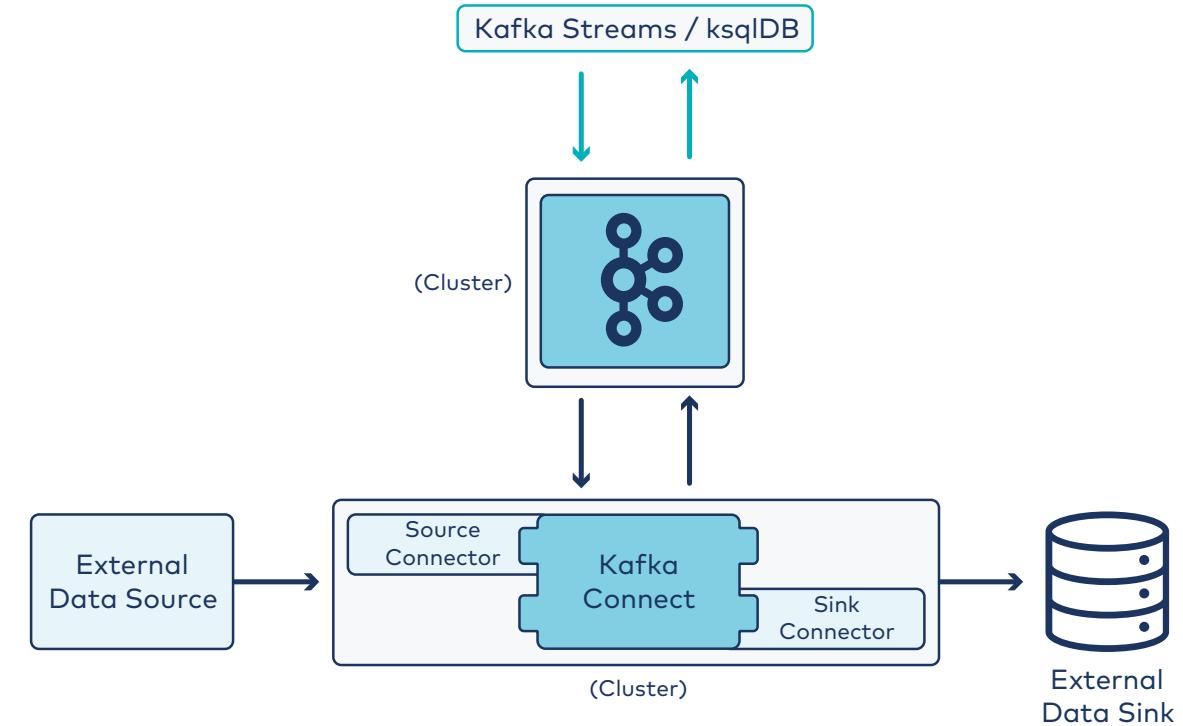
Description

Case studies of using a Connector to read in data from a source, transform the data, and write it to a sink. SMTs vs. Kafka Streams vs. ksqlDB as options for transforming data.

Generalizing & Extending

Here we have

1. Kafka Connect source connectors to read data from external systems, database tables in this case.
2. Kafka topics on the brokers to receive our input.
3. Streaming applications to transform our data.
4. Another Kafka topic on the brokers to receive our transformed data.



We could also send this temperature data to another system that displays temperatures on a dashboard.

This is an example of **ETL**.

A Related Problem

Say we

- Have access to a hospital's database.
- Want to extract information on patients who've been diagnosed with cancer.
(and compare with those who have not been)
- Want to load this information to CSV files that will be given to medical researchers studying correlations between diagnoses and patient traits.

Does this fit what we just did?

One Last Example

- You are maintaining information about car insurance rates for customers.
- Regulations based on where the cars are principally garaged drive a base cost that's part of an insurance rate.
- Information on vehicles and their garage locations is in one external system.
- Insurance rate information is in a different external system.
 - A base rate is one field of a table in such a system.

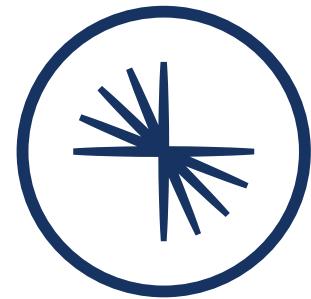
Questions:

- When a vehicle's location changes, what has to happen?
- What tools would you use to achieve the necessary updates?

Choose Your Tool

Tool	Good when...	Challenges
Kafka Streams	<ul style="list-style-type: none">Want a lot of controlWant custom logic	<ul style="list-style-type: none">Need to know Java/ScalaOverkill for very simple transformations
ksqlDB	<ul style="list-style-type: none">Logic fits SQL-like syntaxWant to develop quickly	<ul style="list-style-type: none">Not everything fits the syntaxNeed to set up ksqlDB server
Kafka Connect SMTs	<ul style="list-style-type: none">Simple transform, e.g.,<ul style="list-style-type: none">Remove data for security or performanceMake your data conform to the schema of the output systemTransform exists	<ul style="list-style-type: none">Need to have an SMTNot for coding business logicLess control

More Advanced Kafka Development Matters



CONFLUENT
Global Education

Agenda

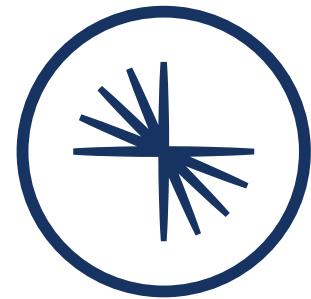


This is a branch of our developer content on more advanced matters in Kafka development. It is broken down into the following modules:

12. Challenges with Offsets
13. Partitioning Considerations
14. Message Considerations
15. Robust Development

This branch assumes proficiency in the Core Kafka Development branch. The last lesson of the last module assumes having completed the Kafka Connect module of the Additional Components of Kafka/CP Deployment Branch.

12: Challenges with Offsets



CONFLUENT
Global Education

Module Overview



This module contains two lessons:

- a. How Does Compaction Affect Consumer Offsets?
- b. What if You Want or Need to Adjust Consumer Offsets Manually?

Where this fits in:

- Hard Prerequisite: Groups, Consumers, and Partitions in Practice
- Recommended Follow-Up: Other modules in this branch, other courses

12a: How Does Compaction Affect Consumer Offsets?

Description

How consumers deal with missing offsets, what happens when offsets don't make sense, and getting into the details of how compaction works: how it affects offsets and is triggered. Deleting keys.

Getting Started: Log Refresher

Recall: messages are written to logs...

- Per partition
- Messages have offsets
- Divided into segments
 - Active, inactive
 - Clean, dirty
- Log retention policies:
 - **delete**
 - **compact**

(refer back to lesson 1b)

12b: What if You Want or Need to Adjust Consumer Offsets Manually?

Description

Reprocessing of messages, finding consumer offsets both now and in the past, programmatically changing offsets, and automatic vs. manual committing strategies.

How Does a Consumer Know Its Offset?

	Consumer is consuming from a partition it was already consuming	Consumer gets assigned a partition that another consumer in group was consuming	Consumer is starting up and is first in group to consume from a partition
Local	Consumer uses offset for this partition from memory	Consumer doesn't have any offset in memory for this partition	Consumer doesn't have any offset in memory for this partition
Kafka	Not needed	Offset comes <code>__consumer_offsets</code> topic for this group/partition pair	There won't be an entry in <code>__consumer_offsets</code> . Offset determined by <code>auto.offset.reset</code> - one of <code>earliest, latest, none</code>

What if a Consumer's Offset Makes No Sense?

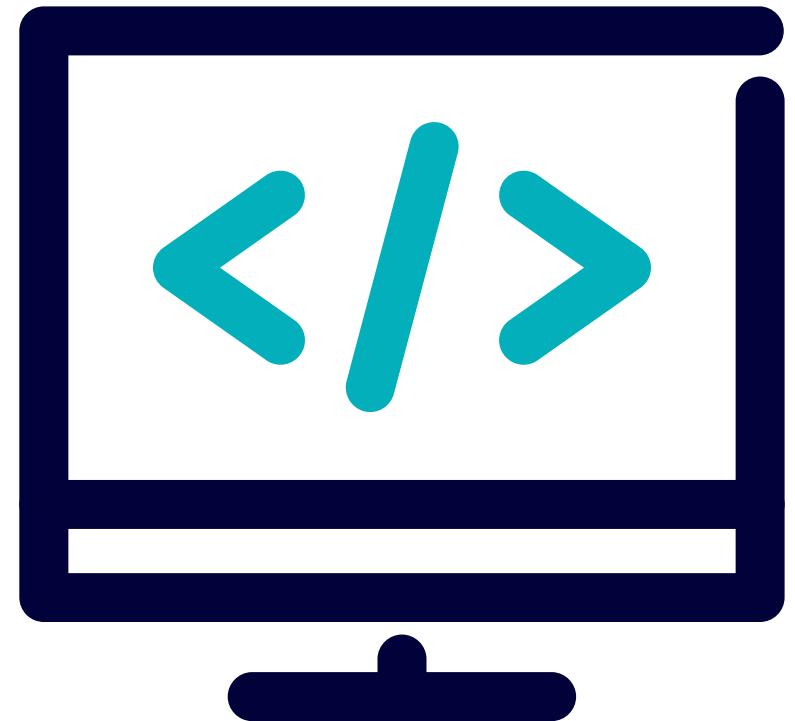
Here again, `auto.offset.reset` determines how the consumer proceeds.

2 New Scenarios	Values for <code>auto.offset.reset</code>	
Consumer offset < smallest offset	<code>earliest</code>	Reset offset to earliest available
Consumer offset > last offset + 1	<code>latest</code>	Reset offset to latest available
	<code>none</code>	Throw exception

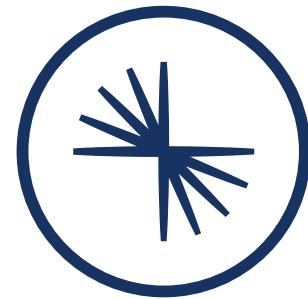
Lab: Kafka Consumer - offsetsForTimes

Please work on **Lab 12a: Kafka Consumer - offsetsForTimes**

Refer to the Exercise Guide



13: Partitioning Considerations



CONFLUENT
Global Education

Module Overview



This module contains two lessons:

- a. How Should You Scale Partitions and Consumers?
- b. How Can You Create a Custom Partitioner?

Where this fits in:

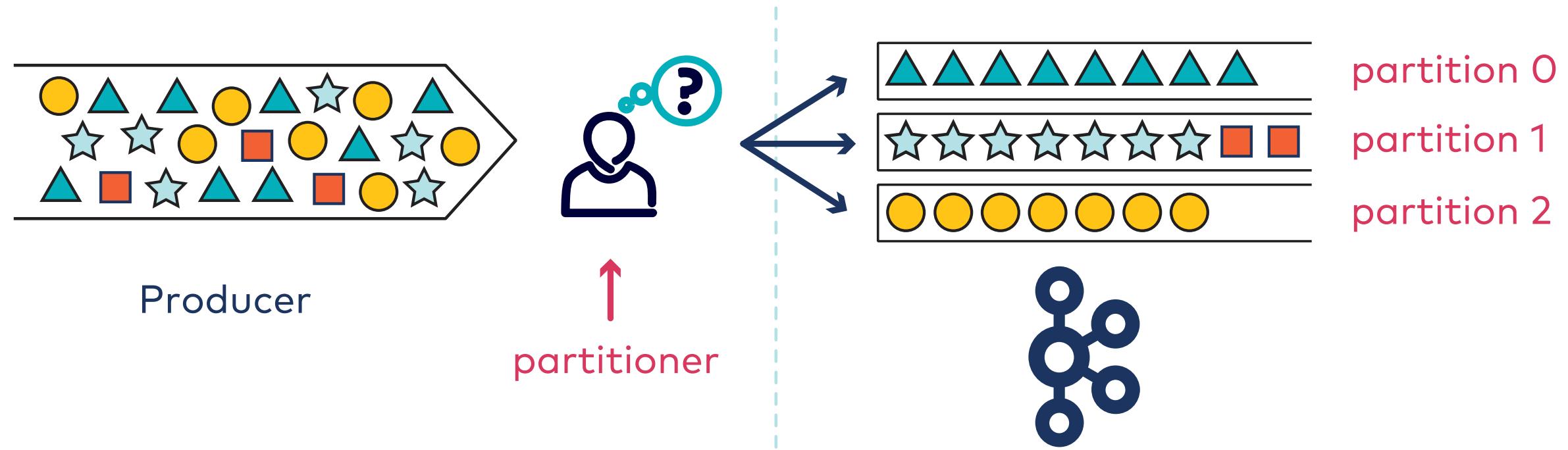
- Hard Prerequisite: Groups, Consumers, and Partitions in Practice
- Recommended Follow-Up: Other modules in this branch, other courses

13a: How Should You Scale Partitions and Consumers?

Description

Discussions about choosing a number of partitions, a number of consumers, and the effects of changing the number of partitions.

Discussion: Deciding Number of Partitions



- How many partitions should you create for your topic?

A Guideline for Choosing Number of Partitions

- Suggested number of partitions: $\max(t/p, t/c)$
 - t : target throughput
 - p : producer throughput per partition
 - c : consumer throughput per partition

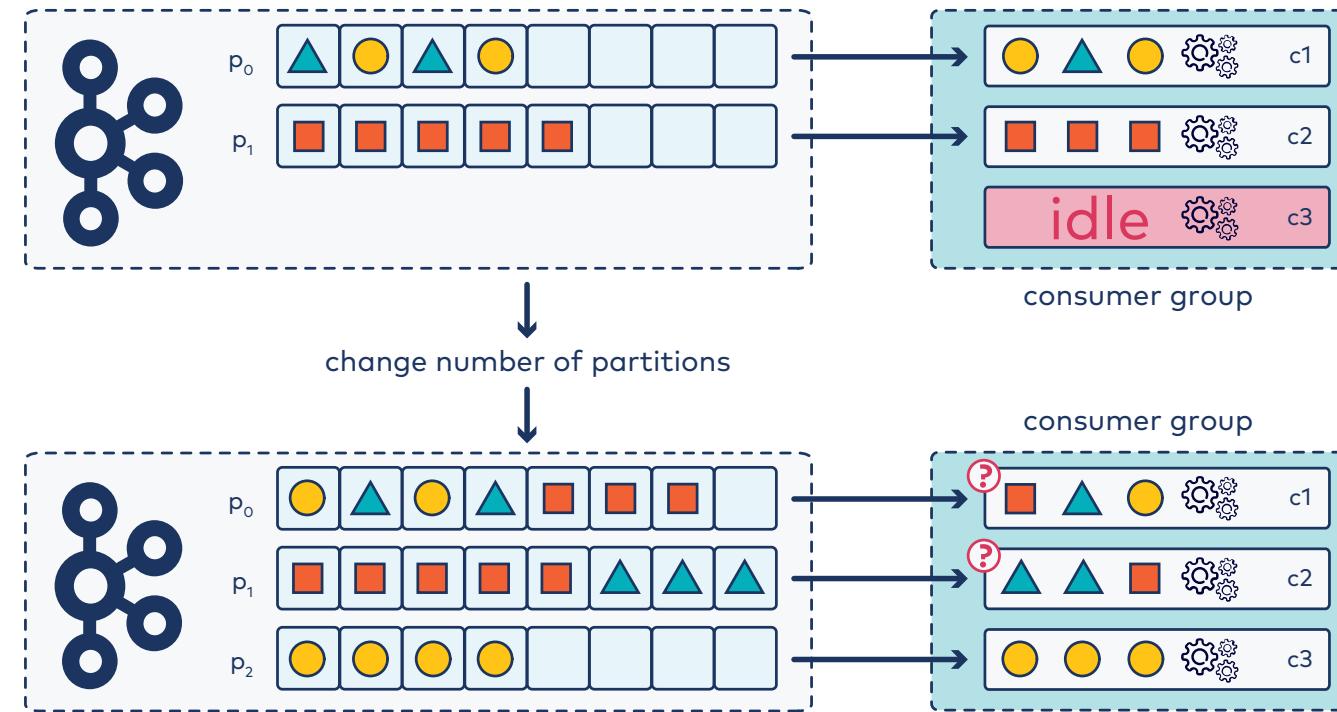
Discussion: Deciding the Number of Consumers

- How does a topic's partition count affect consumer group scalability?
- Why should all topics have a **highly divisible** number of partitions?
- With the default partition assignment strategy (range), what would happen if a consumer group of 10 consumers subscribed to 10 topics, each topic with 1 partition?

Discussion: What If I Need More Partitions?

What are the consequences and options around increasing a topic's number of partitions?

Increasing Partitions: Accept Your Fate



Lab: Increasing Topic Partition Count

Please work on **Lab 13a: Increasing Topic Partition Count**

Refer to the Exercise Guide



13b: How Can You Create a Custom Partitioner?

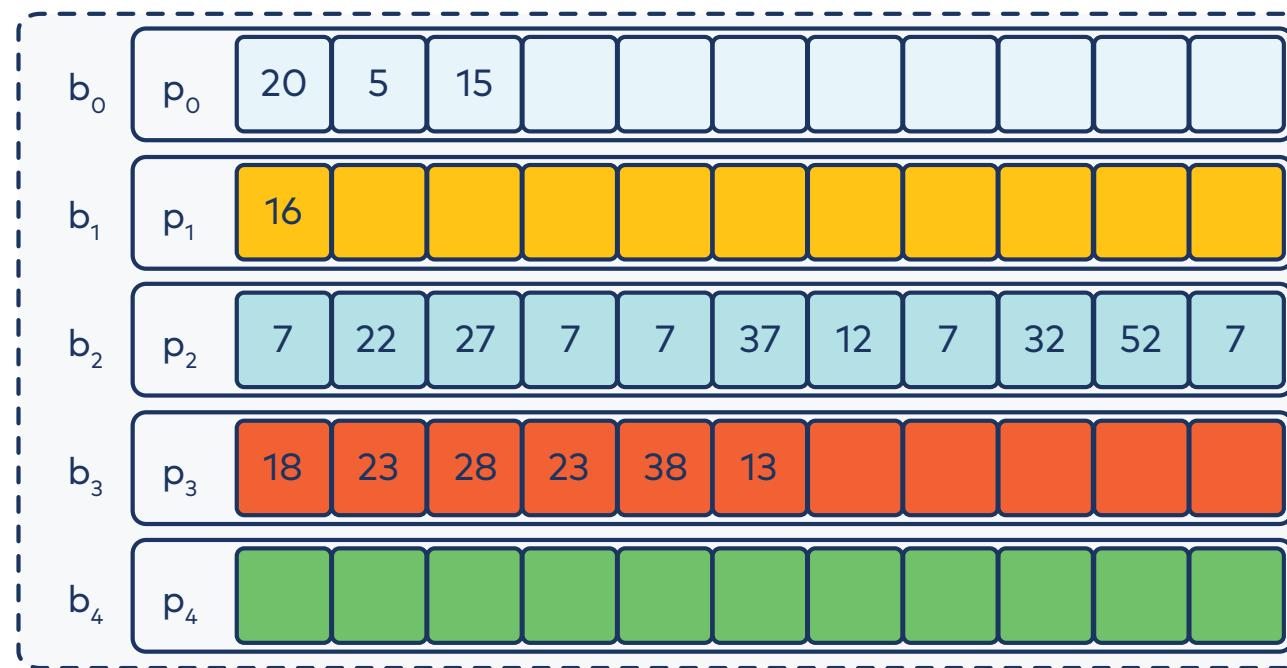
Description

The motivation for and how to write a custom partitioner.

Partitioning with Keys

Default: `hash(key) % numPartitions`

But what if there is skew?



Custom Producer Partitioner

- Implement `Partitioner` interface and customize `partition()`

```
1 public interface Partitioner extends Configurable, ...
2 {
3     void configure(java.util.Map<java.lang.String,?> configs);
4     void close();
5     void onNewBatch(String topic, Cluster cluster, int prePartition);
6
7     int partition(java.lang.String topic,
8                  java.lang.Object key,
9                  byte[] keyBytes,
10                 java.lang.Object value,
11                 byte[] valueBytes,
12                 Cluster cluster);
13 }
```

- Set producer property `partitioner.class`

Custom Partitioner: Example (1)

- In this example, we want to store all messages with a particular key in one partition and distribute all other messages across the remaining partitions. The following code sets the stage:

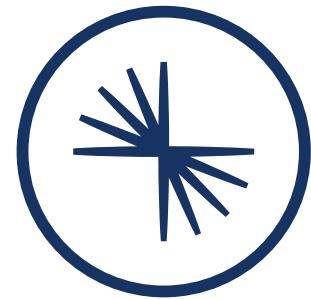
```
1 public class MyPartitioner implements Partitioner
2 {
3     public void configure(Map<String, ?> configs) {}
4     public void close() {}
5     public void onNewBatch() {}
6
7     public int partition(String topic, Object key, byte[] keyBytes,
8                         Object value, byte[] valueBytes, Cluster cluster)
9     {
10         int numPartitions = cluster.partitionsForTopic(topic).size();
11
12         if ((keyBytes == null) || (!(key instanceof String)))
13             throw new InvalidRecordException("Record did not have a string Key");
```

Custom Partitioner: Example (2)

- And the remaining code contains the decision logic:

```
13     // This key will always go to Partition 0
14     if (((String) key).equals("OurBigKey"))
15         return 0;
16
17     // Other records will go to remaining partitions using a hashing function
18     return (Math.abs(Utils.murmur2(keyBytes)) % (numPartitions - 1)) + 1;
19 }
20 }
```

14: Message Considerations



CONFLUENT
Global Education

Module Overview



This module contains three lessons:

- a. How Do You Guarantee How Messages are Delivered?
- b. How Should You Deal with Kafka's Message Size Limit?
- c. How Do You Send Messages in Transactions?

Where this fits in:

- Hard Prerequisite: Preparing Producers for Practical Uses
- Recommended Prerequisite: Groups, Consumers, and Partitions in Practice
- Recommended Follow-Up: Other modules in this branch, other courses

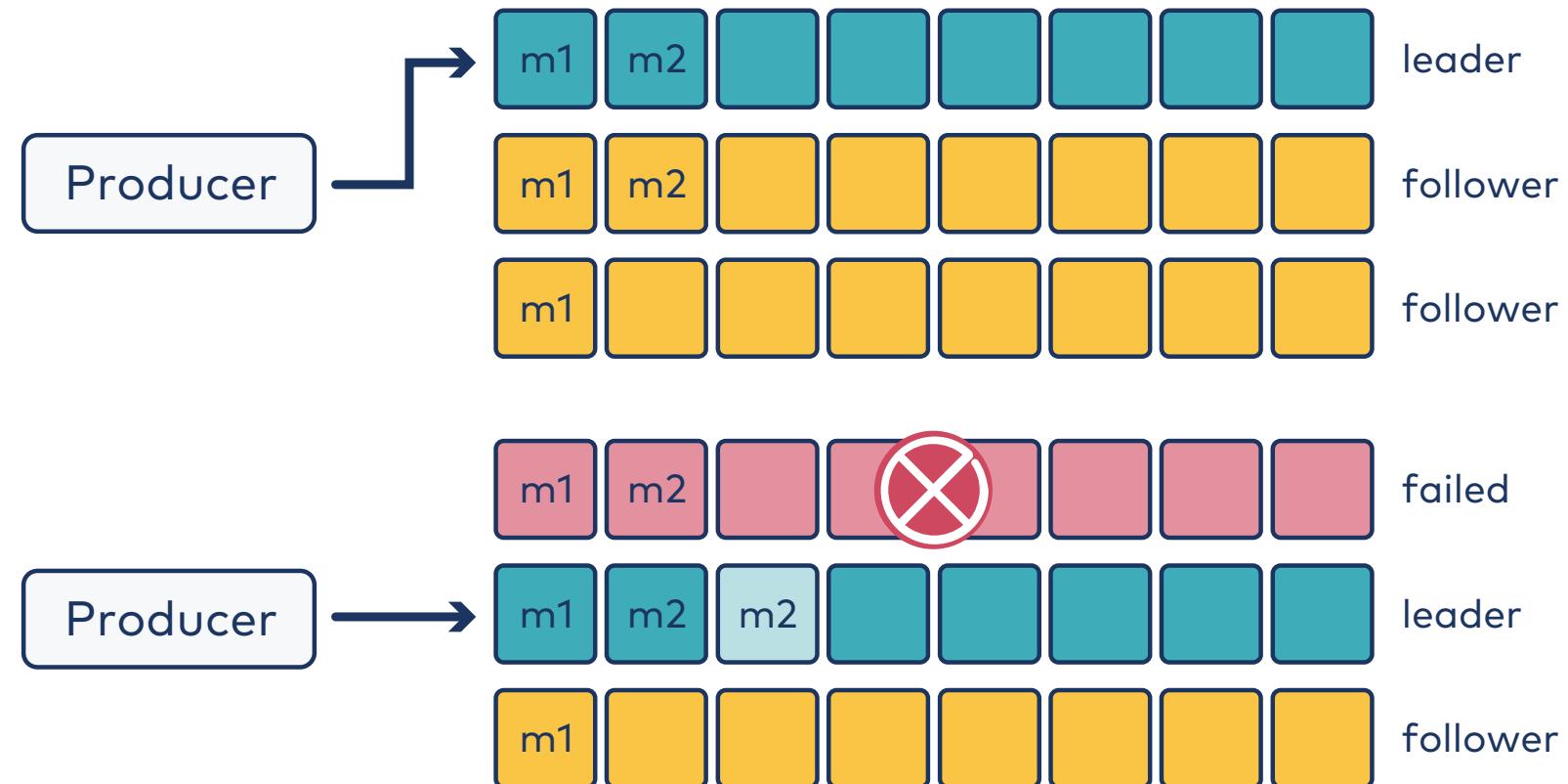
14a: How Do You Guarantee How Messages are Delivered?

Description

How to deal with guaranteeing ordered delivery of messages and non-duplicated writes with idempotence, along with how it works.

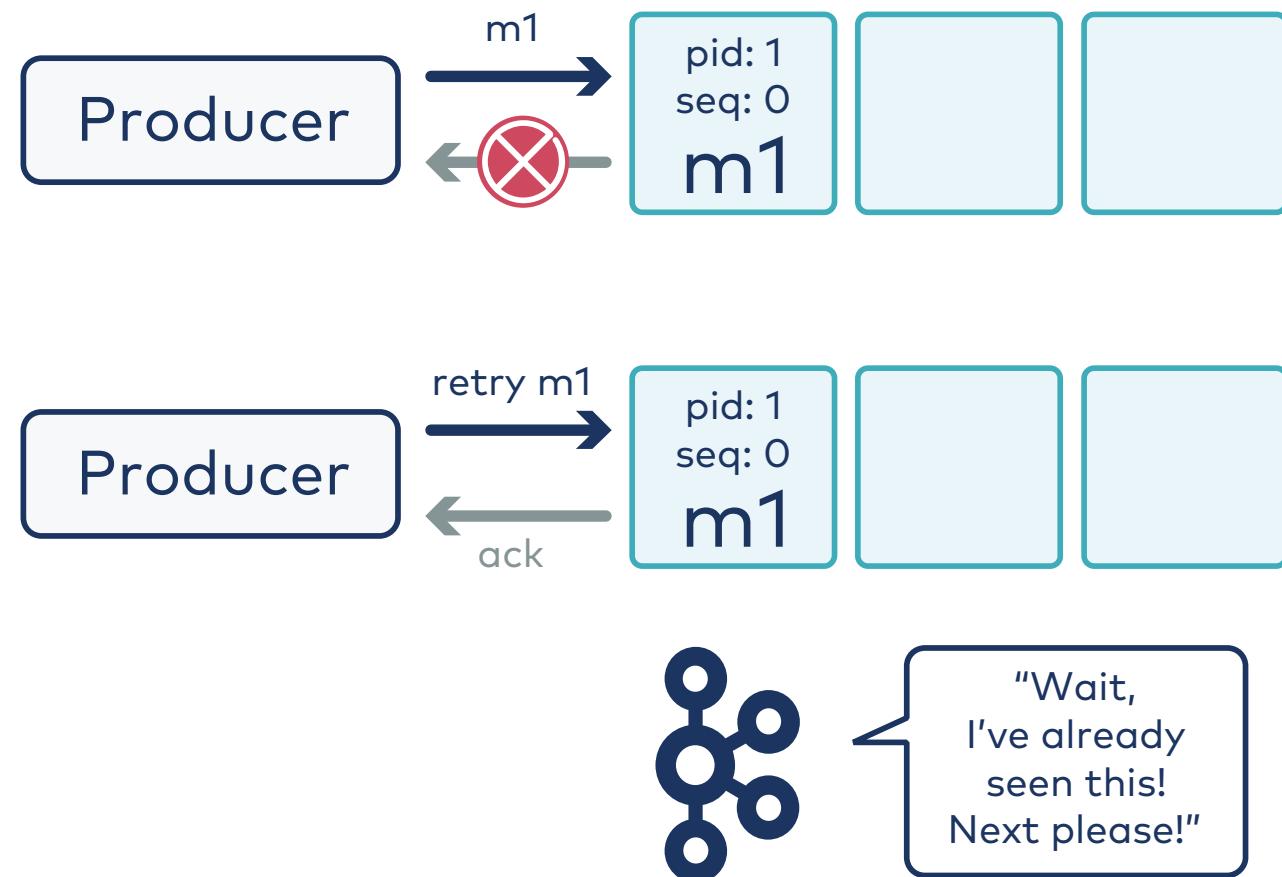
Problem: Producing Duplicates to the Log

- `acks = all`
- `retries > 0`



Solution: Idempotent Producers

- `enable.idempotence = true`
- `acks = all`

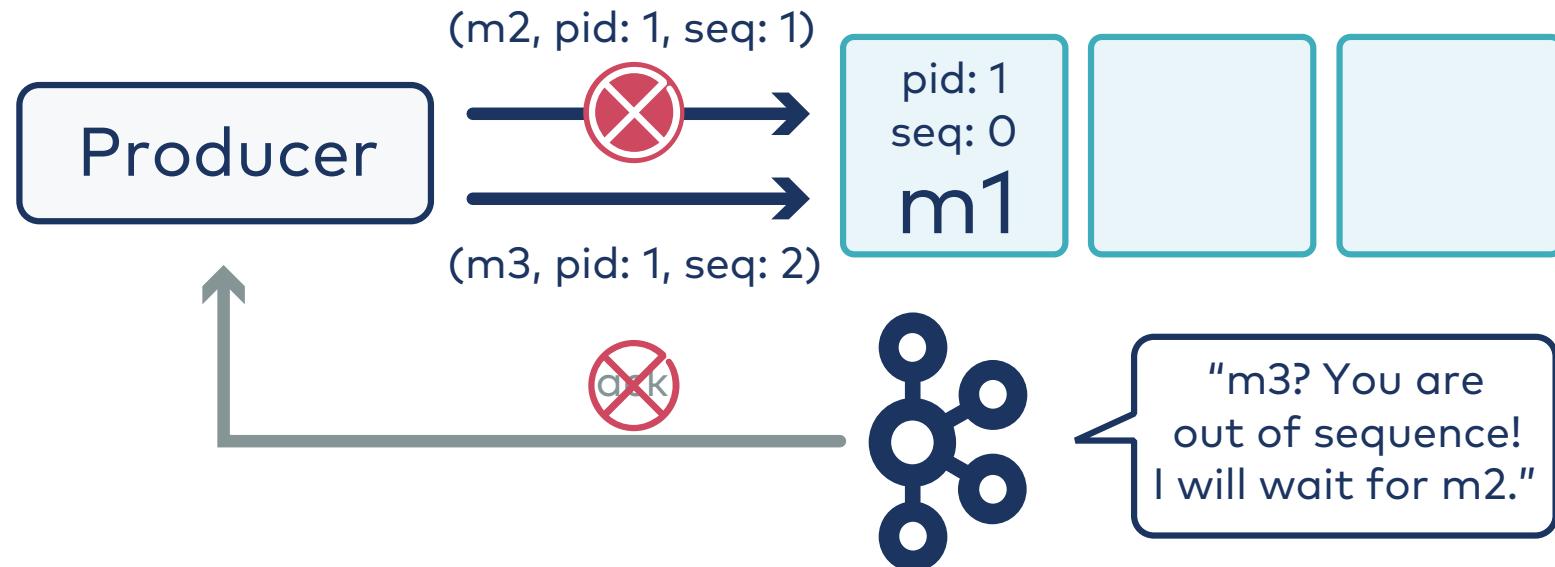


Problem: Producing Messages Out of Order

- `max.in.flight.requests.per.connection > 1`
- `retries > 0`



Solution: Idempotent Producers



14b: How Should You Deal with Kafka's Message Size Limit?

Description

Understanding message size limits, best practices, and strategies for dealing with natural demands for large messages.

Kafka's Message Size Requirements

- Broker default for `message.max.bytes` is 1 MB
- Producer default for `max.request.size` is also 1 MB

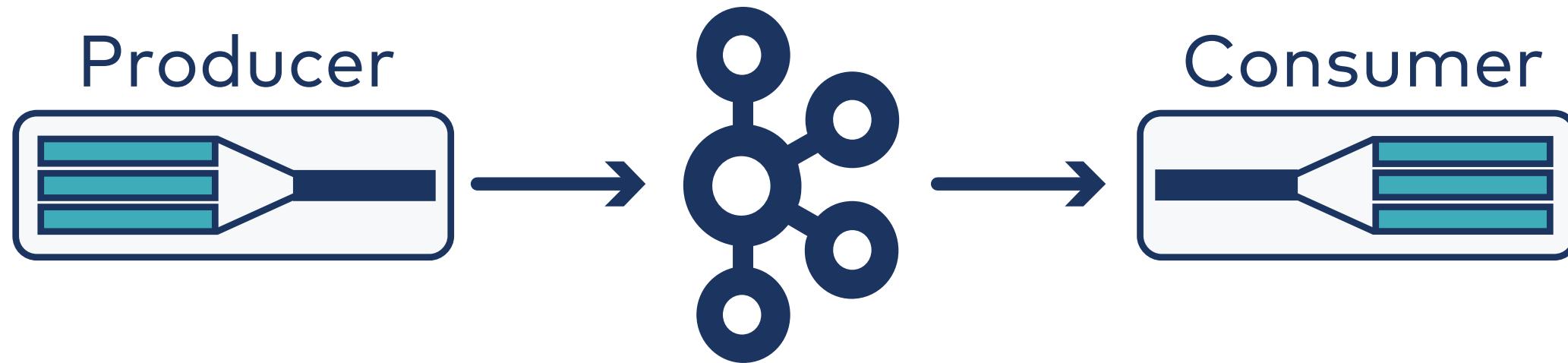


Confluent Cloud has larger defaults.

- Increasing message size limit can lead to:
 - poor garbage collection performance
 - less memory available for other important broker business
 - more resources needed to handle requests

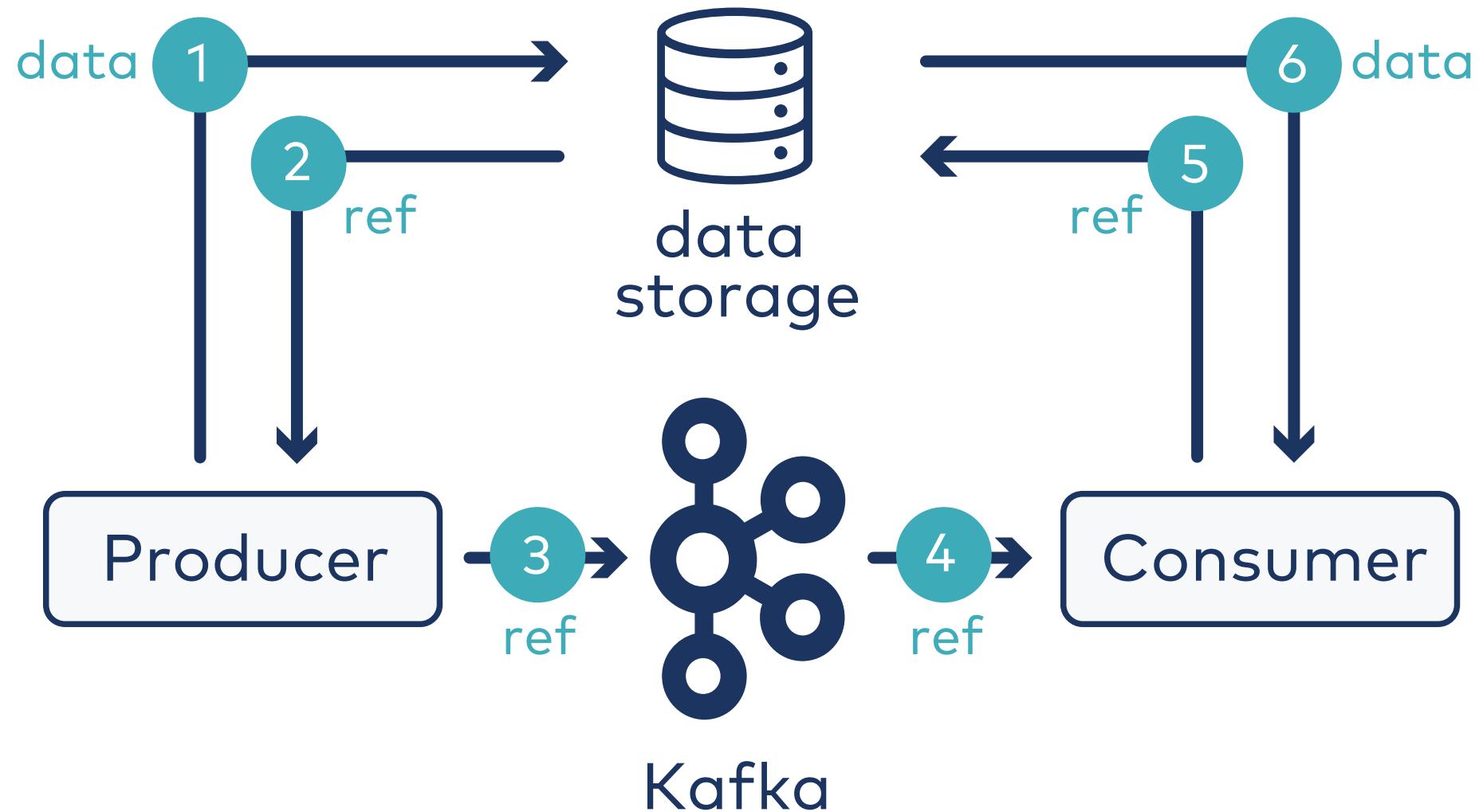
Question: So how can we deal with this?

Solution 1: Compression

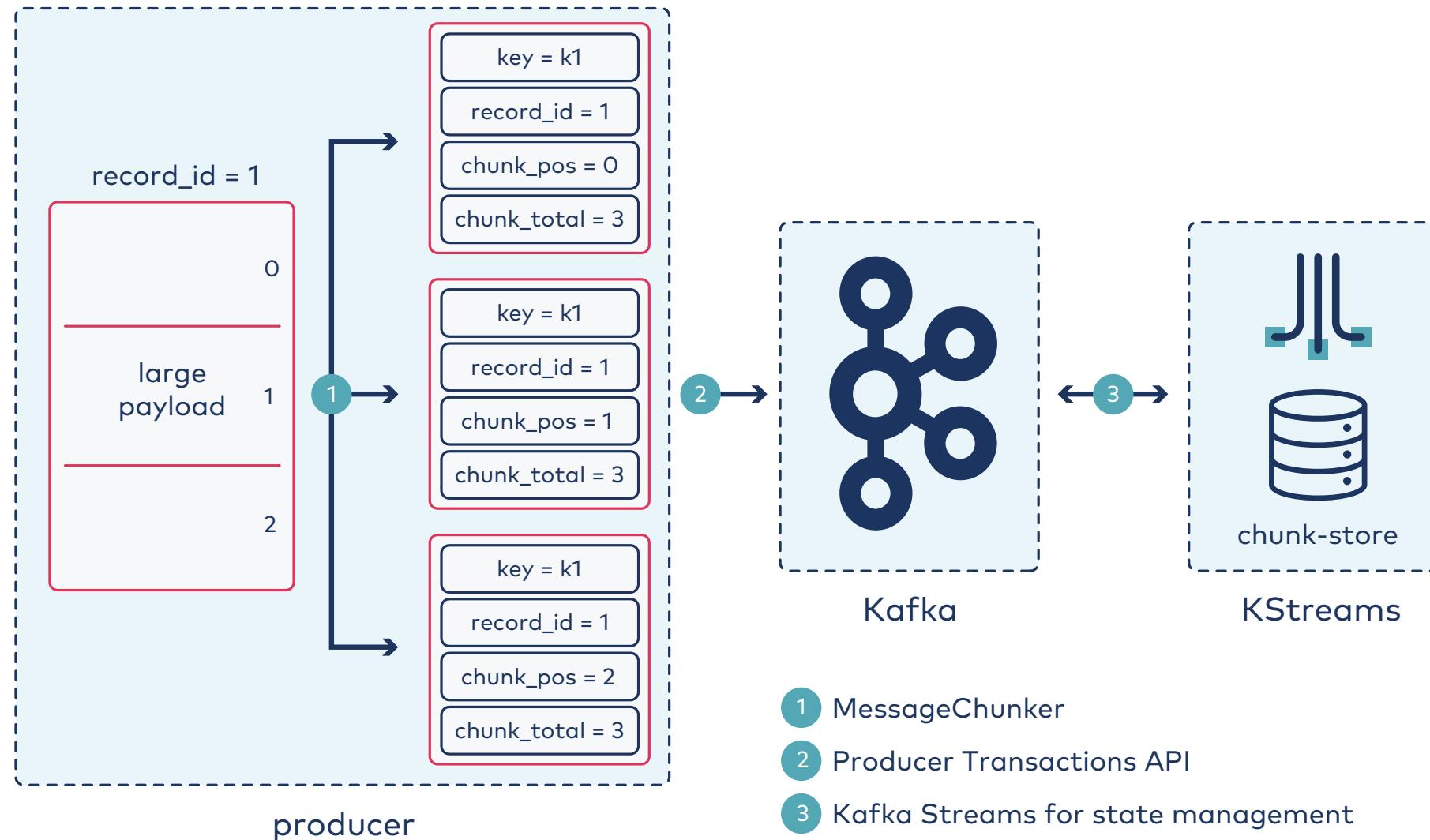


1. Producer batches and then compresses the record batch
2. Compressed record batch stored in Kafka
3. Consumer decompresses

Solution 2: Pass Reference to External Store



Solution 3: Record Chunking



14c: How Do You Send Messages in Transactions?

Description

What a transaction is, committed vs. aborted transactions, how Kafka marks them, and what a single-partition log looks like with transactions involved. Consumer side of transactions. Code for a transactional producer. How the above would change if the producer first consumed or if it produced to more than one partition.

Overview

The idea: Group messages together as a **transaction**. Process them only if all can be processed.

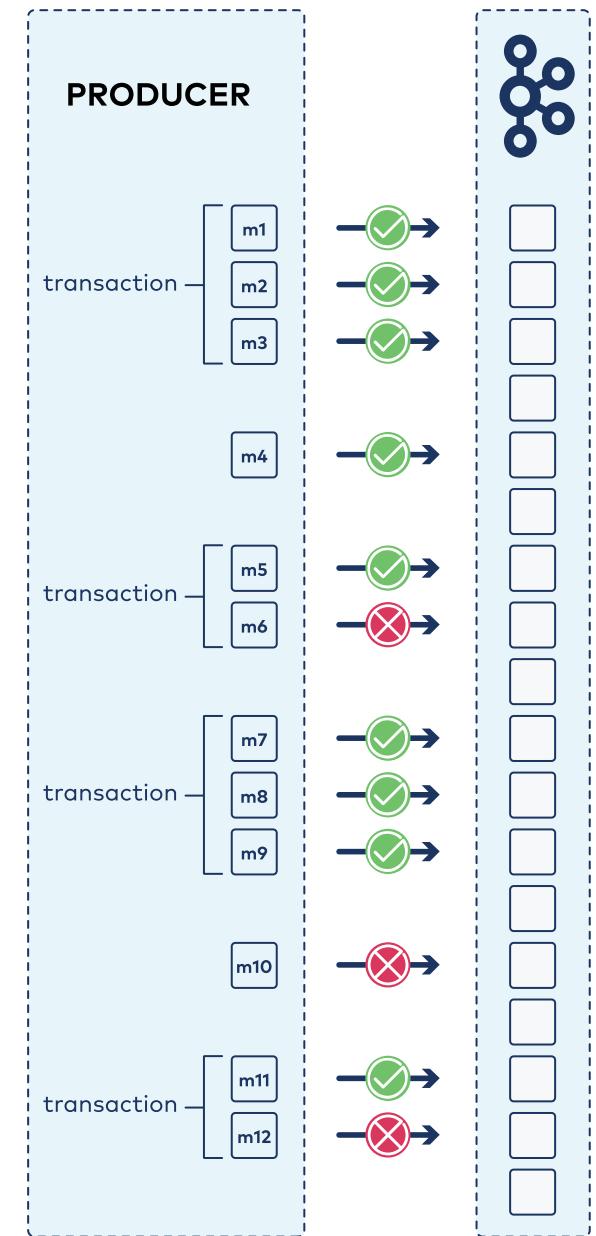
Kafka has a Transactions API. Support for this is part of core Kafka.

Interactive Example Overview

A producer is going to send some messages in transactions and some other messages

We see here which ultimately succeed and fail to make it to Kafka

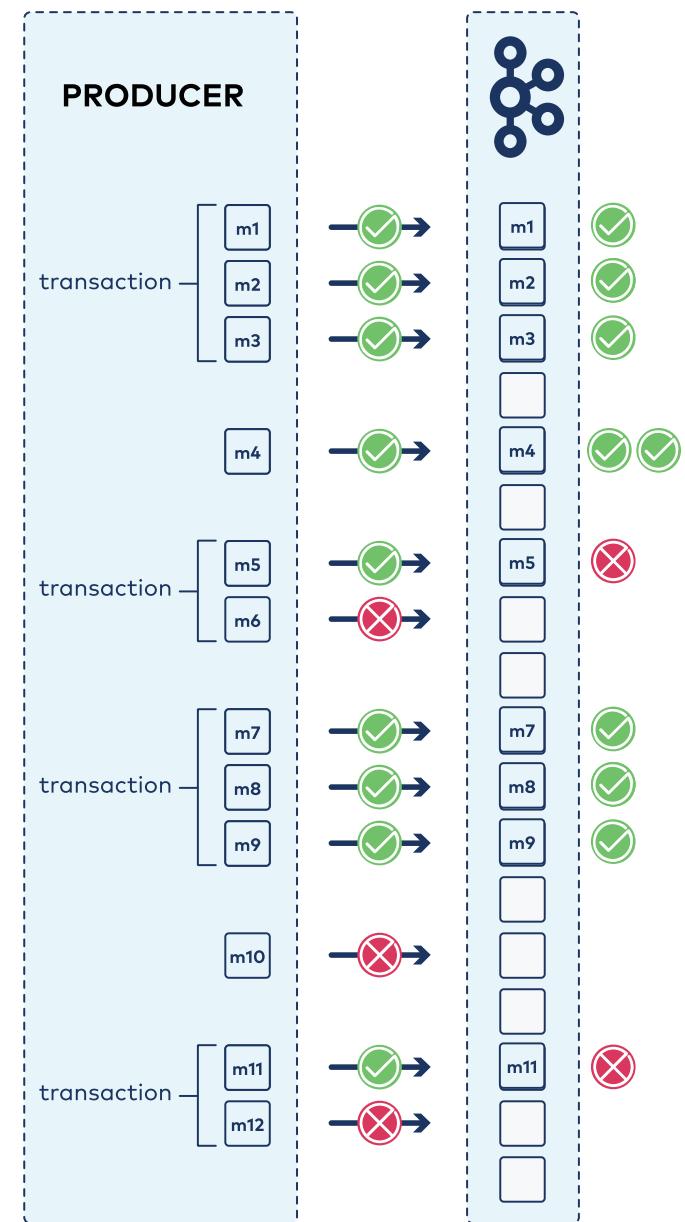
We'll assess sequentially which ones consumers should process



Putting it all Together and Going Further

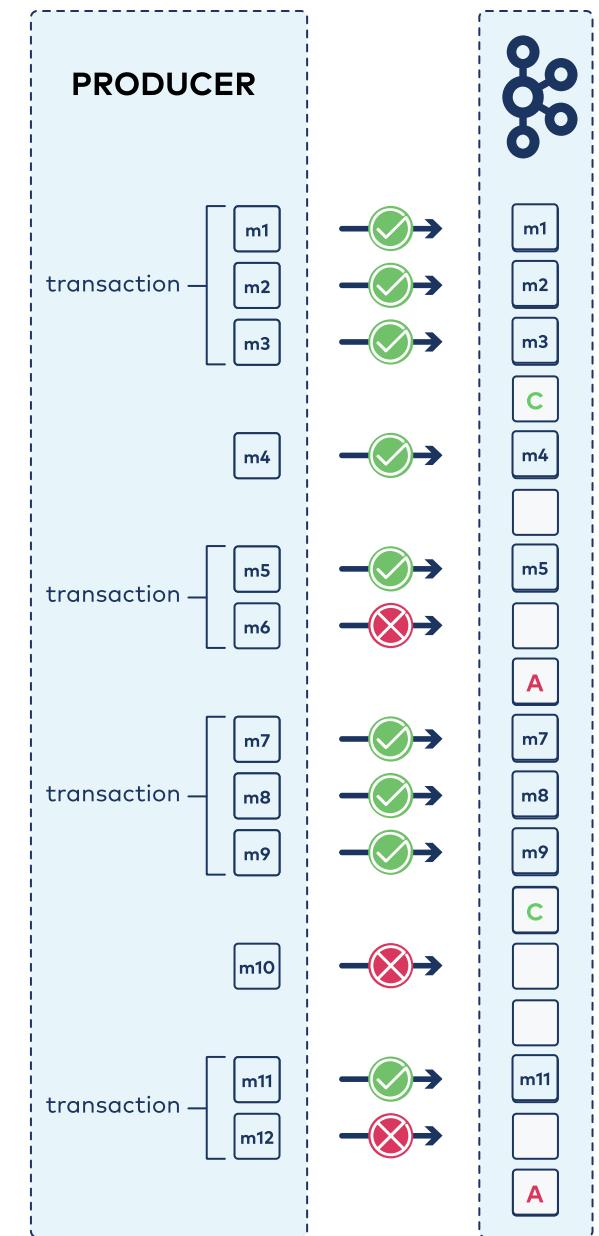
Here's everything from the last three slides and more:

Question: So how can Kafka logs denote committed transactions?

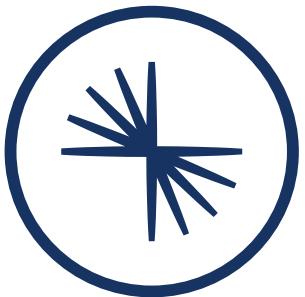


Commit and Abort Markers

- Remember, messages are **immutable**.
- `commitTransaction()` causes a **C** marker to be written to the log
- `abortTransaction()` causes an **A** marker to be written to the log
- Consumers use these markers



15: Robust Development



CONFLUENT
Global Education

Module Overview



This module contains two lessons:

- a. What Should You Think About When Testing Kafka Applications?
- b. How Can You Leverage Error Handling Best in Kafka Connect?

Where this fits in:

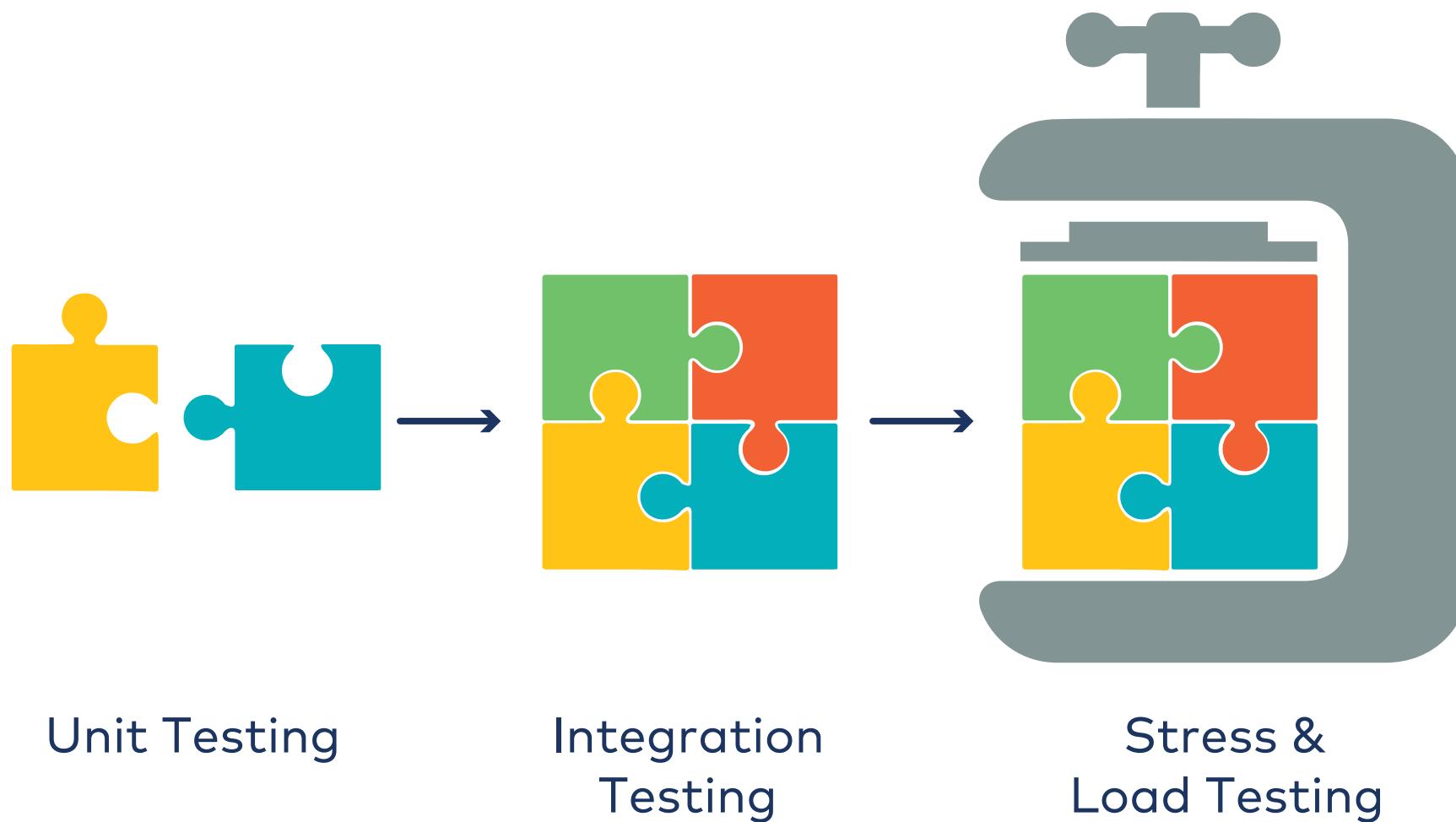
- Hard Prerequisite: Starting with Consumers, Kafka Connect
- Recommended Prerequisite: Groups, Consumers, and Partitions in Practice
- Recommended Follow-Up: Other modules in this branch, other courses

15a: What Should You Think About When Testing Kafka Applications?

Description

Testing considerations as they apply to Kafka development.

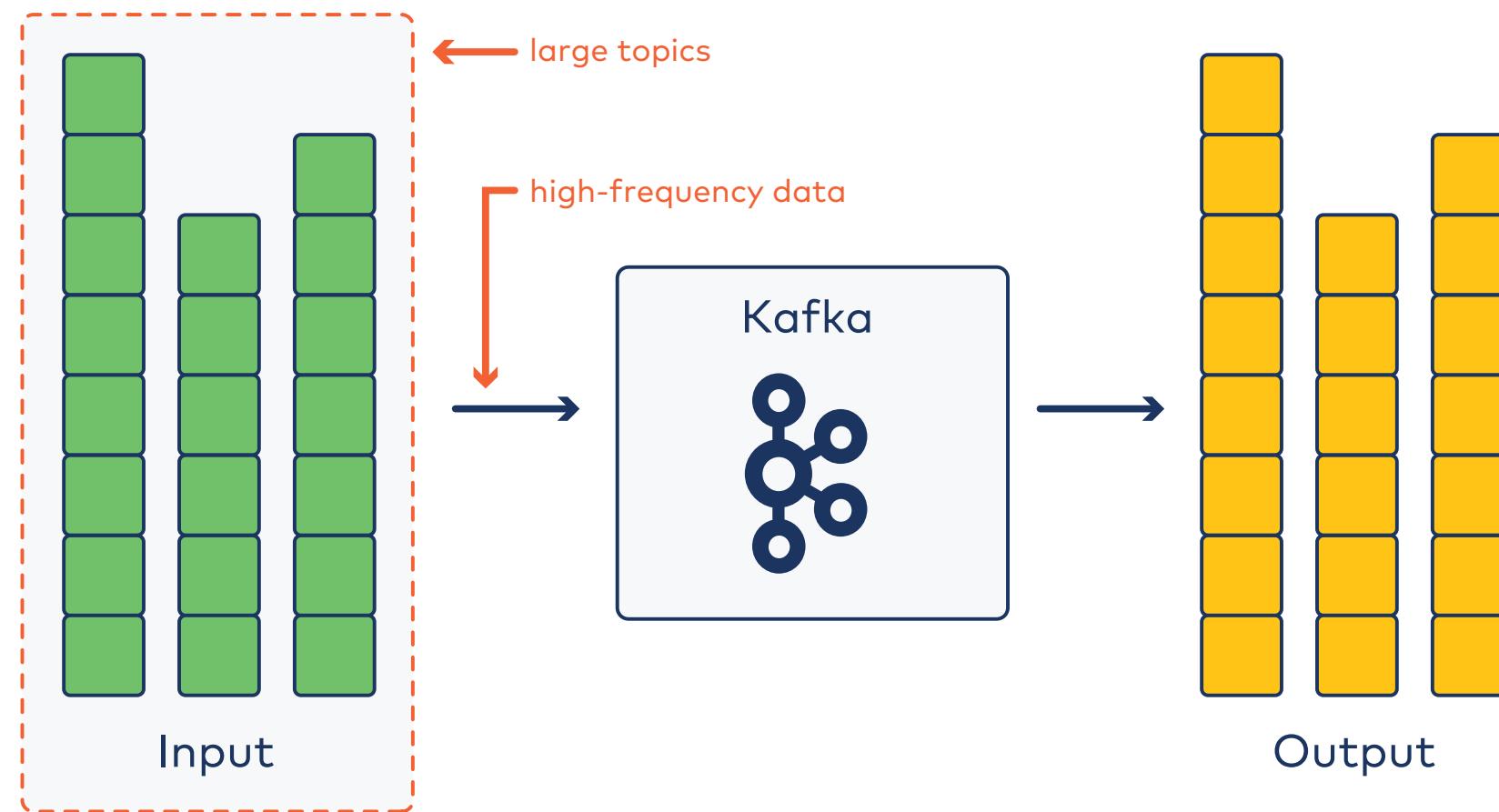
Kinds of Testing



Considerations

- Test individual producers and consumers before testing how they integrate with Kafka.
- Mock or stub dependencies for unit testing.
- Consider the "Anatomy of a Kafka Streams Application" lesson
 - Having `getTopology()` and `getProperties()` as individual methods facilitates better testing.

Stress & Load Testing Considerations



15b: How Can You Leverage Error Handling Best in Kafka Connect?

Description

Kafka Connect error handling framework options. Dead letter queue.

Handling Errors in Connect

Copying to/from external systems can fail...

- source system unavailable
- destination system unavailable
- messages that cannot be processed
- transient failure

What to do?

- Give up completely?
- Allow some errors?
- Retry on failure?
- Log problem messages to handle separately?

Fail Fast Scenario

Why?

- **Poisoned messages**
i.e., cannot be processed
- Source/target system unavailable

How?

Configuration settings:

```
# disable retries on failure (default 0)
errors.retry.timeout=0

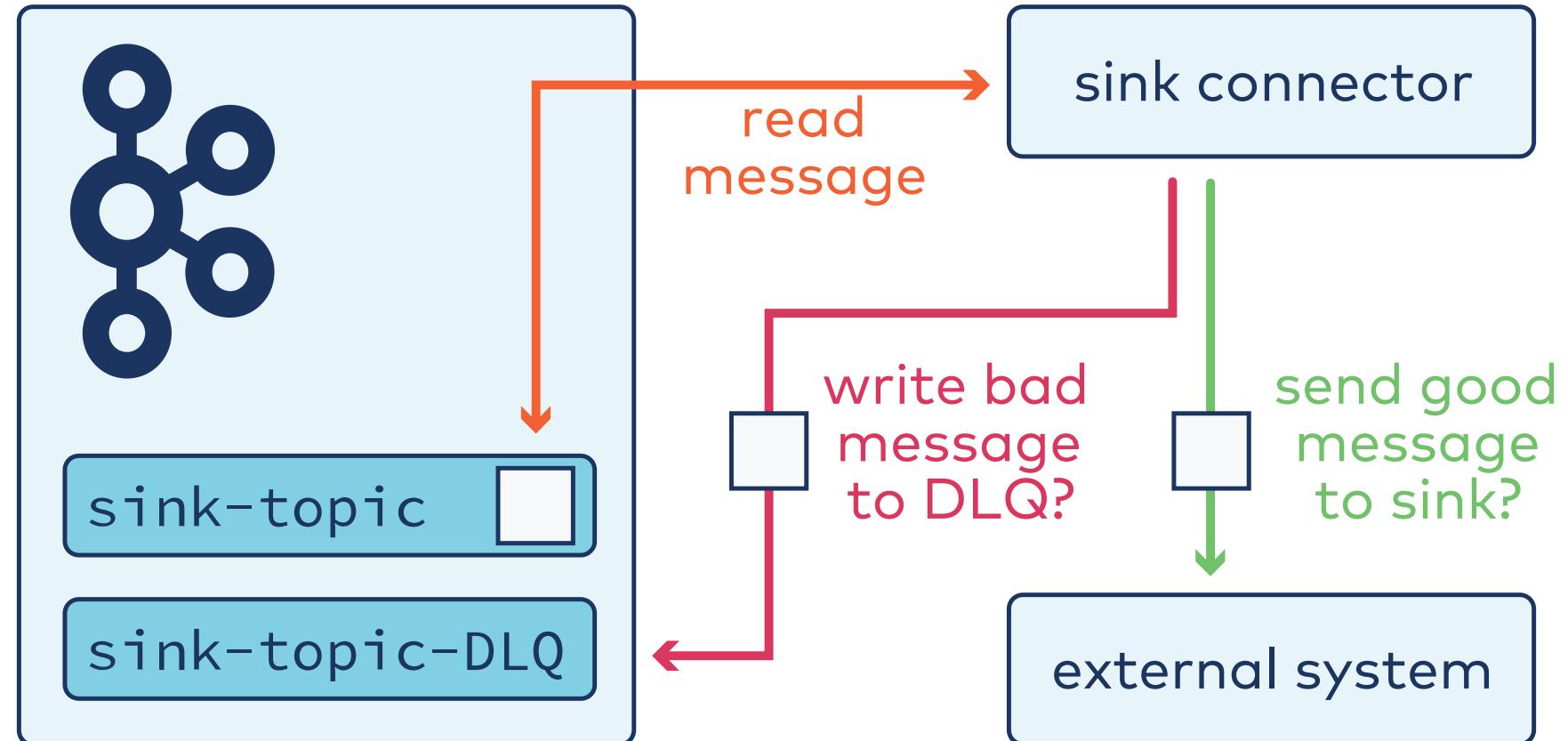
# do not log the error and their contexts
errors.log.enable=false

# do not record errors in a
# dead letter queue topic
errors.deadletterqueue.topic.name=""

# Fail on first error
errors.tolerance=none
```

Dead Letter Queue

- **Problem:** Writing message to external system fails
- **Solution:**
 - Rather than giving up, produce this message to a special Kafka topic
→ Called the **dead letter queue (DLQ)**
 - Can inspect those messages separately and decide how to handle



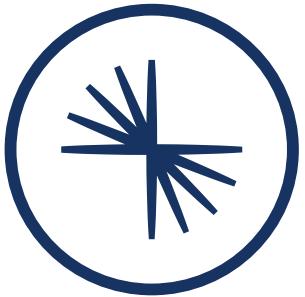
The DLQ is for sink connectors only.

Error Management Options

To configure error management, configure the **connector** settings:

Name	Default	Source Connectors?	Sink Connectors?
errors.retry.timeout	0	yes	yes
errors.retry.delay.max.ms	1 min	yes	yes
errors.tolerance	-	yes	yes
errors.deadletterqueue.topic.name	""	no	yes
errors.log.enable	false	yes	yes
errors.log.include.messages	false	yes	yes

Conclusion



CONFLUENT
Global Education

Course Contents



Now that you have completed this course, you should have the skills to:

- Write Producers and Consumers to send data to and read data from Apache Kafka
- Create schemas, describe schema evolution, and integrate with Confluent Schema Registry
- Integrate Kafka with external systems using Connect
- Write streaming apps with Kafka Streams & ksqlDB
- Describe common issues faced by Kafka developers and ways to troubleshoot
- Make design decisions about acks, keys, partitions, batching, replication, and retention policies

Other Confluent Training Courses

- Confluent Stream Processing Using Apache Kafka® Streams & ksqlDB
 - recommended to take this next!
- Apache Kafka® Administration by Confluent
- Confluent Advanced Skills for Optimizing Apache Kafka®
- Managing Data in Motion with Confluent Cloud



For more details, see <https://confluent.io/training>

Confluent Certified Developer for Apache Kafka

Duration: 90 minutes

Qualifications: Solid understanding of Apache Kafka and Confluent products, and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours a day!

Cost: \$150

Register online: www.confluent.io/certification



Confluent Certified Administrator for Apache Kafka

Duration: 90 minutes

Qualifications: Solid work foundation in Confluent products and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours per day!

Cost: \$150

Register online: www.confluent.io/certification



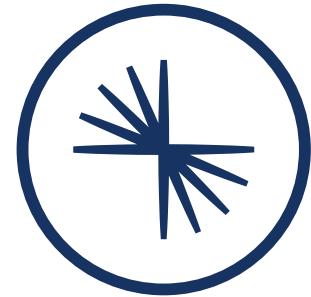
We Appreciate Your Feedback!



Please complete the course survey now.

Thank You!

Appendix: Additional Problems to Solve



CONFLUENT
Global Education

Overview

This section contains a few additional problems to be solved that will reinforce the concepts in this course.

These problems were originally written as warm-up problems for instructor-led training for this course. Your instructor may or may not choose to incorporate some or all of these problems in class; you may find them to provide additional enrichment in any case. Some other problems originally created as warm-up problems have been adapted into activities in the content of this version of this course.

Problem A: Comparing Producers and Consumers

We looked at Java code for writing custom producers and consumers yesterday...

- a. To create producers, we instantiated objects of three different classes. What were they? How are they related?
- b. What are the analogous classes for consumers?
- c. What additional step must we do for consumers in setting them up? Why not for producers?
- d. What is the main operation a producer does? A consumer?

Problem B: Partitioning with Keys

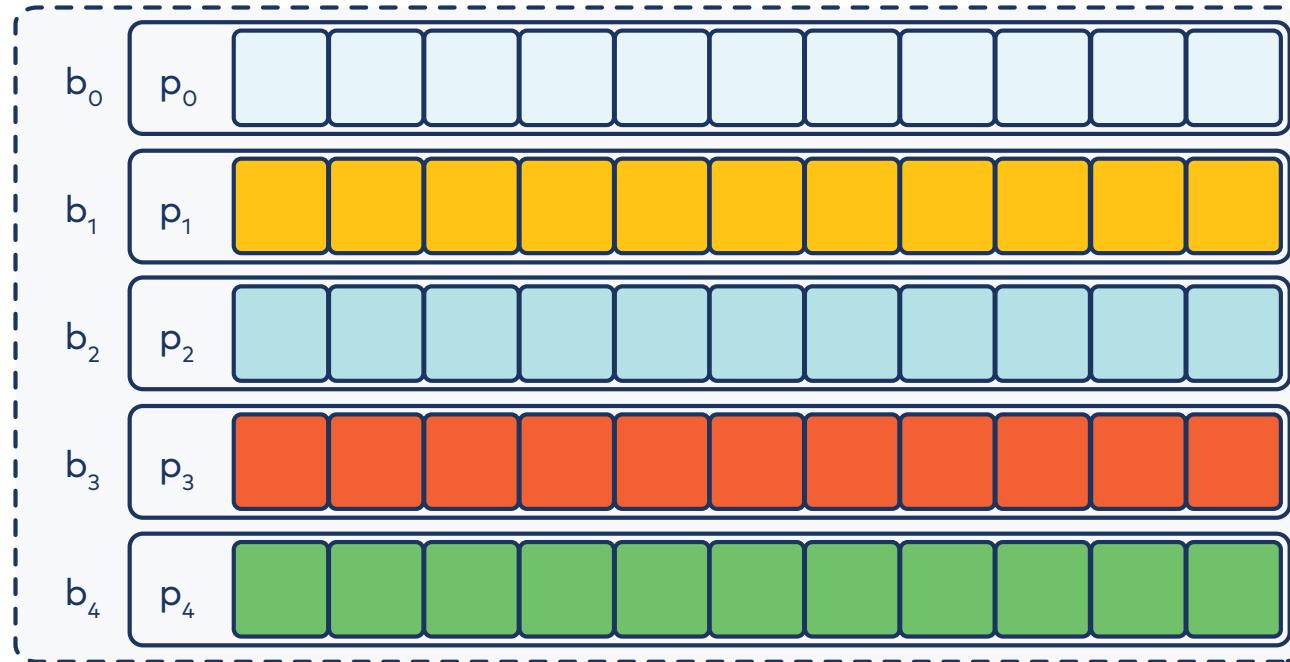
Suppose we have 5 brokers and 5 partitions. The five partitions are p_0, p_1, p_2, p_3 , and p_4 ; and they are stored on brokers b_0, b_1, b_2, b_3 , and b_4 , respectively. Suppose we are using default Kafka settings and $\text{hash}(n) = n$. Then...

Suppose we send these messages:

- m_0 with key 1
- m_1 with key 7
- m_2 with key 12
- m_3 with key 18
- m_4 with key 27
- m_5 with key 10

Which partitions contain which messages after the Kafka cluster has received them all?

Here's an illustration of the situation:

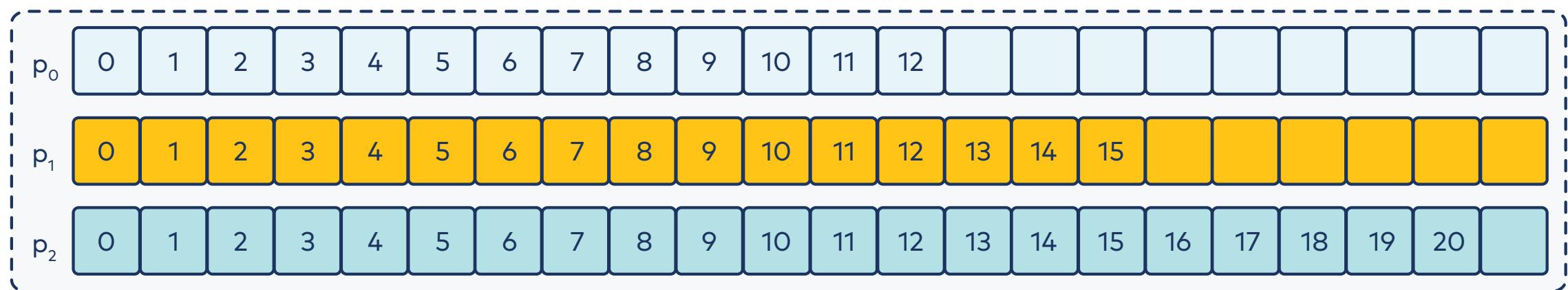


Problem C: Groups, Consumers, and Partitions

Suppose we have partitions p_0, p_1, p_2 each with first offset 0 and with most recent offsets of 12, 15, and 20, respectively (where every offset between contains data).

Suppose we have consumer group g_0 with consumers c_0, c_1 , and c_2 and consumer group g_1 with consumers c_3 , and c_4 . (Some details may be missing - you interpret!)

The setup might look like this:

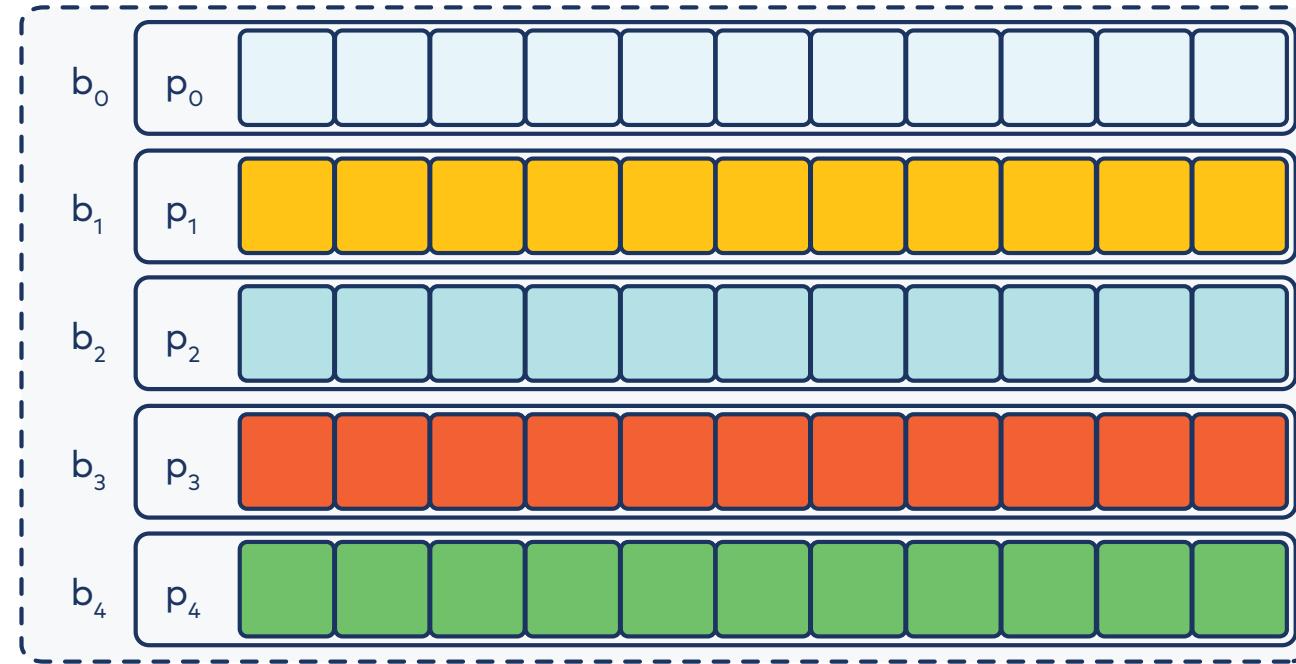


Your quest:

- a. Give a valid assignment of consumers to partitions that could result.
- b. How many different consumer offsets are stored in this scenario? Explain/list them.
- c. For each consumer offset in your list, give a valid value it could have
- d. Suppose c_1 , goes down. What happens? Concretely illustrate the scenario now.
- e. For any one consumer, tell the offsets of the messages read in the case that it gets 2 messages in the next batch. Repeat for some other consumer in the case that this other consumer gets 3 messages in the next batch.
- f. Give two examples of things that could happen that would trigger a consumer/partition assignment rebalance. One must not involve anything with consumers changing (before the rebalance).

Problem D: Partitioning without Keys

We will again work with this scenario of five brokers and five partitions:



Suppose instead of the keyed messages in another problem, we have messages with null keys.

Skim [KIP 480](#) for more on how Kafka handles partitioning by default when messages do not have keys.

Suppose we have messages that get assigned to partitions in the buffer and these were the batches before being flushed:

batch 0: a, b, c

batch 1: d, e

batch 2: f, g, h, i

batch 3: j

batch 4: k, l, m, n

batch 5: o, p

batch 6: q, r

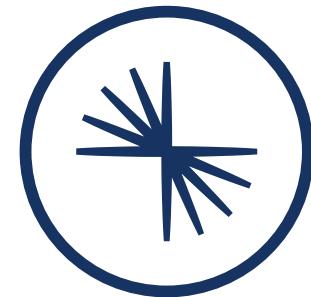
batch 7: s, t, u

batch 8: v, w, x, y, z

Give an illustration of which messages would land on which partitions...

- ...if you are in a breakout room with an even number: assume we had relatively decent luck, or
- ...if you are in a breakout room with an odd number: assume we had not such great luck

Appendix: Additional Content



CONFLUENT
Global Education

Overview

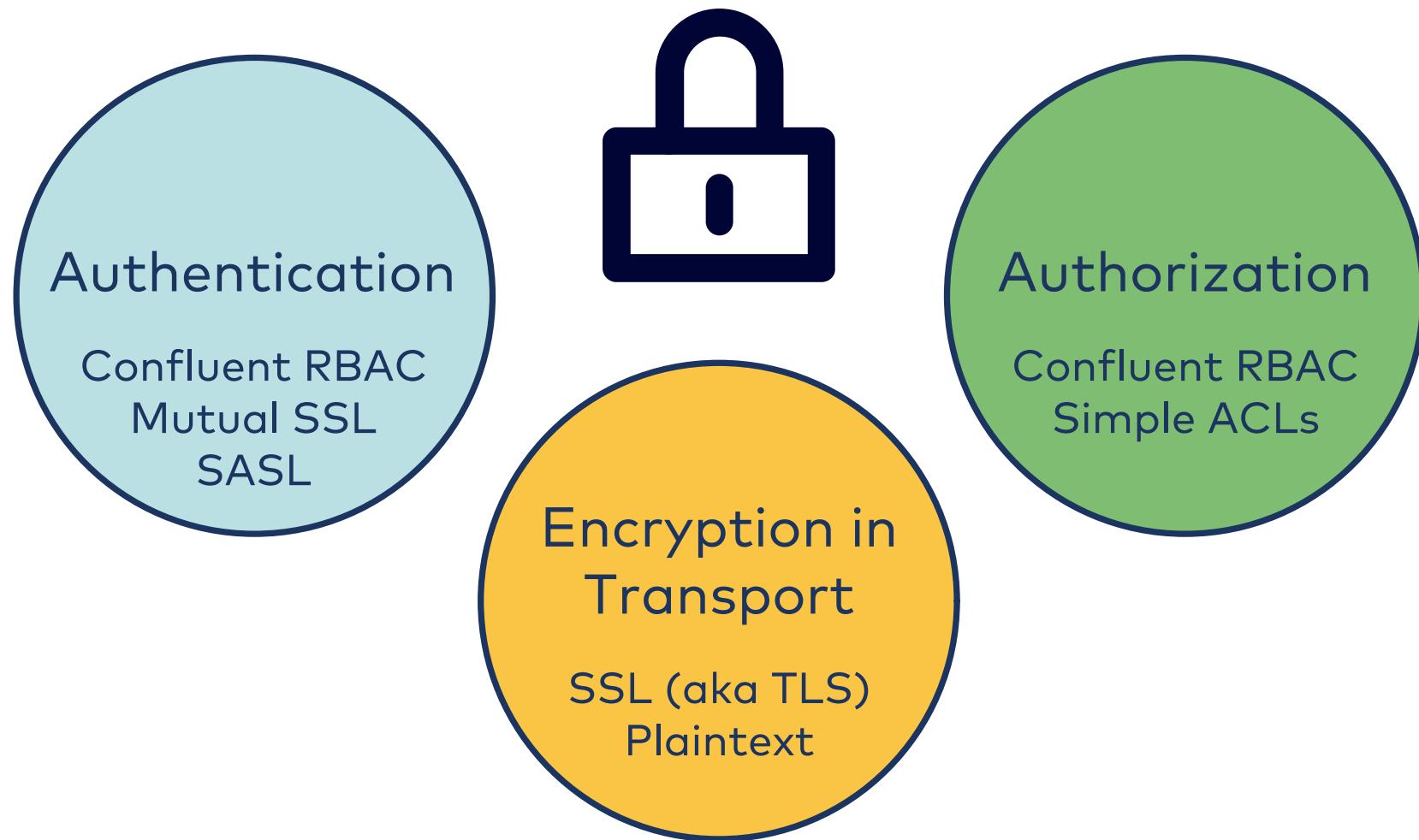
This appendix contains a few additional lessons. These lessons are for additional information for you, but are not designed the same as the rest; namely, they do not have activities or labs to reinforce the content like the rest.

Appendix A: A Taste of Kafka Security for Developers

Description

What Kafka security supports vs. does not for self-managed vs. Confluent Cloud. High level overview of security options in Kafka/CP. Simple, basic config examples.

Security Overview (1)



Security Overview (2)

Security is mostly out of your hands as a developer, but be aware:

- For all deployments of Kafka, authorization and authentication are supported
- So is encryption. But... what is supported varies:
 - For self-managed deployments of Kafka, only encryption of data in transport
 - For Confluent Cloud deployments, encryption at rest also (and standard)
- Authorization rights can be set to allow clients to
 - Describe topics
 - Read from topics
 - Write to topics



If your application is failing but you suspect your logic is correct, make sure you have the appropriate permissions.

Appendix B: Confluent Cloud vs. Self-Managed Kafka

Description

Defining what Confluent Cloud is and comparing CCloud deployments to self-managed deployments. Key considerations for developing clients for CCloud vs. for self-managed deployments.

What is Confluent Cloud?

- Can deploy CP as self-managed software but...
- Confluent Cloud = **fully-managed** deployment of CP
 - Many administrative tasks done for you
- Confluent Cloud available on
 - AWS
 - Google Cloud Platform
 - Microsoft Azure



Confluent Cloud removes some administrative tasks from your organization's responsibilities.

Quick Look at the UI

CLI

```
$ ccloud kafka topic list  
demo-topic  
other-topic  
my-topic
```

```
$ ccloud kafka topic create product-topic \  
--replication-factor 3 \  
--partitions 6
```

GUI

The screenshot shows the Confluent Cloud Kafka UI. On the left, the 'Cluster settings' page for 'RUSS-RANDOM-LHCPSDBU' cluster is displayed. It includes sections for Cluster details (Cluster name: russ-random-lhcpsdbu, Cluster ID: lkv-wkvw, Bootstrap server: pkc-4mym6.us-east-1.aws.confluent.cloud:90), Kafka API keys, Consumers, and Tools & client configuration. Below this, 'Usage limits' show Ingress up to 100 Mbps, Egress up to 100 Mbps, Storage up to 5 TB, Partitions up to 2048, and Uptime SLA None. On the right, the 'Topics' page lists existing topics: PAGEVIEWS_FEMALE (6 partitions), PAGEVIEWS_FEMALE_LIKE_09 (6 partitions), PAGEVIEWS_REGIONS (6 partitions), USERS_ORIGINAL (6 partitions), pageviews (6 partitions), pksdlc-43d60-processing-log (1 partition), and pksdlc-43d60PAGEVIEWS_FE... (6 partitions). A 'Topic Summary' sidebar on the far right provides detailed settings for the selected topic.

What's Different?

Topics, partitions, replication, producing, consuming, etc. all behave the same whether you're developing for Confluent Cloud or developing for a self-managed deployment of Confluent Platform.

The biggest differences are administrative:

- With self-managed Kafka, there is no security setup out of the box. With CCloud, there is.
- With self-managed Kafka security enabled, there is no support for encryption of data at rest, only data in transit. With CCloud, data is encrypted at rest.

How Can I Learn More?

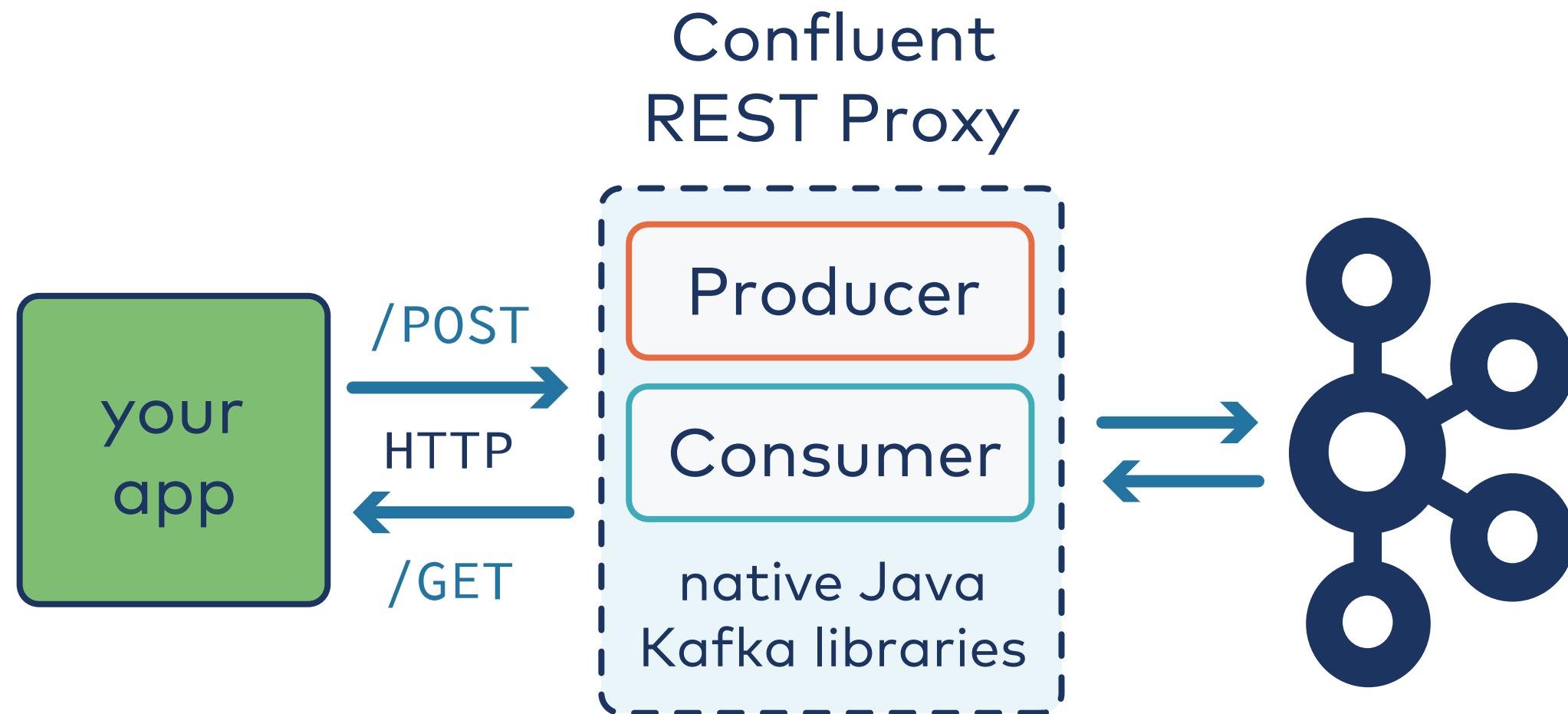
- Managing Data in Motion with Confluent Cloud instructor-led training class - one-day
- Free self-paced course "Introduction to Confluent Cloud" available via Content Raven
- sign up at <http://training.confluent.io>
- Introductory exercise using Confluent Cloud in Fundamentals course: [Getting Started in Confluent Cloud](#)

Appendix C: Developing with the REST Proxy

Description

What is the REST Proxy, why one would use it, how a REST producer might look, how a REST consumer might look, examples of other places in the DEV modules where REST fits in.

Confluent REST Proxy (1)



Confluent REST Proxy (2)

- The Confluent REST Proxy allows you to use HTTP to perform actions on the Kafka cluster
- The REST calls are translated into Java Kafka client calls
- This allows virtually any language to access Kafka
- Uses POST to send data to Kafka:
 - JSON, binary, Avro, Protobuf and JSON Schema
- Uses GET to retrieve data from Kafka

Creating a Producer with REST Proxy

```
1 import requests
2 import json
3
4 url = "http://restproxy:8082/topics/my_topic"
5 headers = {"Content-Type" : "application/vnd.kafka.json.v2+json"}
6 # Create one or more messages
7 payload = {"records":
8   [
9     {"key": "firstkey",
10      "value": "firstvalue"
11   }]
12 # Send the message
13 r = requests.post(url, data=json.dumps(payload), headers=headers)
14 if r.status_code != 200:
15   print "Status Code: " + str(r.status_code)
16   print r.text
```

Creating a Consumer with REST Proxy (1)

- Main Logic

```
1 import requests
2 import json
3 import sys
4
5 FORMAT = "application/vnd.kafka.v2+json"
6 POST_HEADERS = { "Content-Type": FORMAT }
7 GET_HEADERS = { "Accept": FORMAT }
8
9 base_uri = create_consumer_instance("group1", "my_consumer")
10 subscribe_to_topic(base_uri, "hello_world_topic")
11 consume_messages(base_uri)
12 delete_consumer(base_uri)
```



This is just a Confluent REST Proxy example. This is not the Python consumer client.

Creating a Consumer with REST Proxy (2)

- Creating the Consumer Instance

```
13 def create_consumer_instance(group_name, instance_name):  
14     url = f'http://rest-proxy:8082/consumers/{group_name}'  
15     payload = {  
16         "name": instance_name,  
17         "format": "json"  
18     }  
19     r = requests.post(url, data=json.dumps(payload), headers=POST_HEADERS)  
20  
21     if r.status_code != 200:  
22         print ("Status Code: " + str(r.status_code))  
23         print (r.text)  
24         sys.exit("Error thrown while creating consumer")  
25  
26     return r.json()["base_uri"]
```

Creating a Consumer with REST Proxy (3)

- Subscribing to a topic

```
27 def subscribe_to_topic(base_uri, topic_name):  
28     payload = {  
29         "topics": [topic_name]  
30     }  
31  
32     r = requests.post(base_uri + "/subscription",  
33                         data=json.dumps(payload),  
34                         headers=POST_HEADERS)  
35  
36     if r.status_code != 204:  
37         print("Status Code: " + str(r.status_code))  
38         print(r.text)  
39         delete_consumer(base_uri)  
40         sys.exit("Error thrown while subscribing the consumer to the topic")
```

Creating a Consumer with REST Proxy (4)

- Consuming and processing messages

```
41 def consume_messages(base_uri):  
42     r = requests.get(base_uri + "/records", headers=GET_HEADERS, timeout=20)  
43  
44     if r.status_code != 200:  
45         print ("Status Code: " + str(r.status_code))  
46         print (r.text)  
47         sys.exit("Error thrown while getting message")  
48  
49     for message in r.json():  
50         if message["key"] is not None:  
51             print ("Message Key:" + message["key"])  
52             print ("Message Value:" + message["value"])
```

Creating a Consumer with REST Proxy (5)

- Deleting the consumer

```
53 def delete_consumer(base_uri):  
54     r = requests.delete(base_uri, headers=POST_HEADERS)  
55  
56     if r.status_code != 204:  
57         print ("Status Code: " + str(r.status_code))  
58         print (r.text)
```

Configuring Connectors with the REST API

- Add, modify delete connectors
- Distributed Mode:
 - Config **only** via REST API
 - Config stored in Kafka topic
 - REST call to **any** worker
- Standalone Mode:
 - Config also via REST API
 - Changes **not persisted!**
- Control Center uses REST API

Using the REST API with Connect

Some important REST endpoints

Method	Path	Description
GET	/connectors	Get a list of active connectors
POST	/connectors	Create a new Connector
GET	/connectors/(string: name)/config	Get configuration information for a Connector
PUT	/connectors/(string: name)/config	Create a new Connector, or update the configuration of an existing Connector

REST API and CCC

Note that many features of Confluent Control Center use the REST API under the hood...

- ksqlDB interactive mode uses the REST API
 - Can access REST API directly
 - Or indirectly via CCC
- Rather than configuring Kafka Connect connectors via the REST API as just shown, you can use CCC
 - ...but this uses the REST API indirectly

Appendix D: Comparing the Java and .NET Consumer API

Description

A comparison of a basic consumer written in Java with a basic consumer written in C#/.NET.

Comparing a Java Consumer with a C#/.NET Consumer

Here we will look at a partial solution to an old basic Consumer lab in two languages. Remember that the C# client is based on the librdkafka library, so this is, on a different level, a comparison of a JVM-based client and a librdkafka-based client.



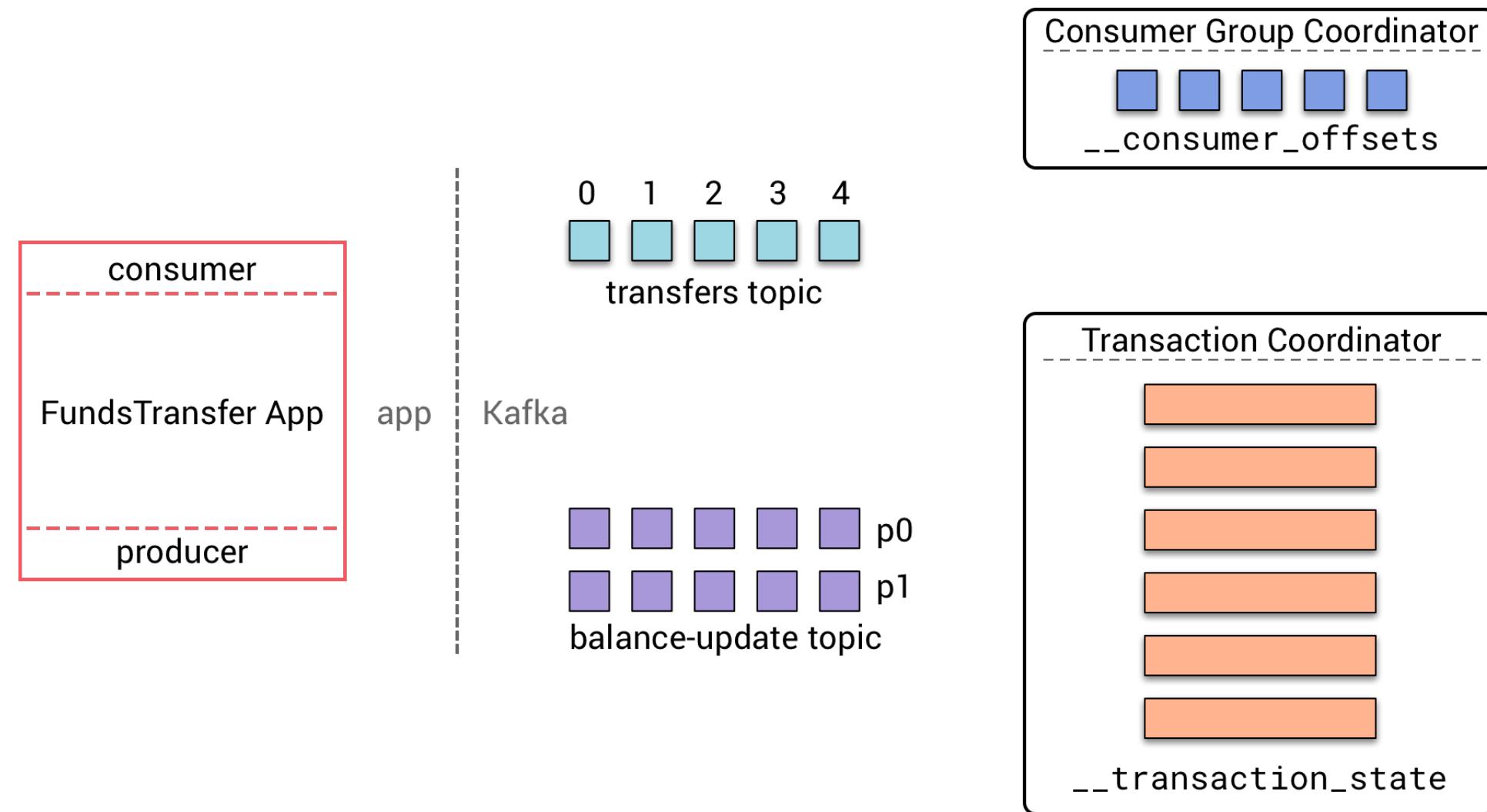
See the guide for this comparison. It does not really work on slides.

Appendix E: Detailed Transactions Demo

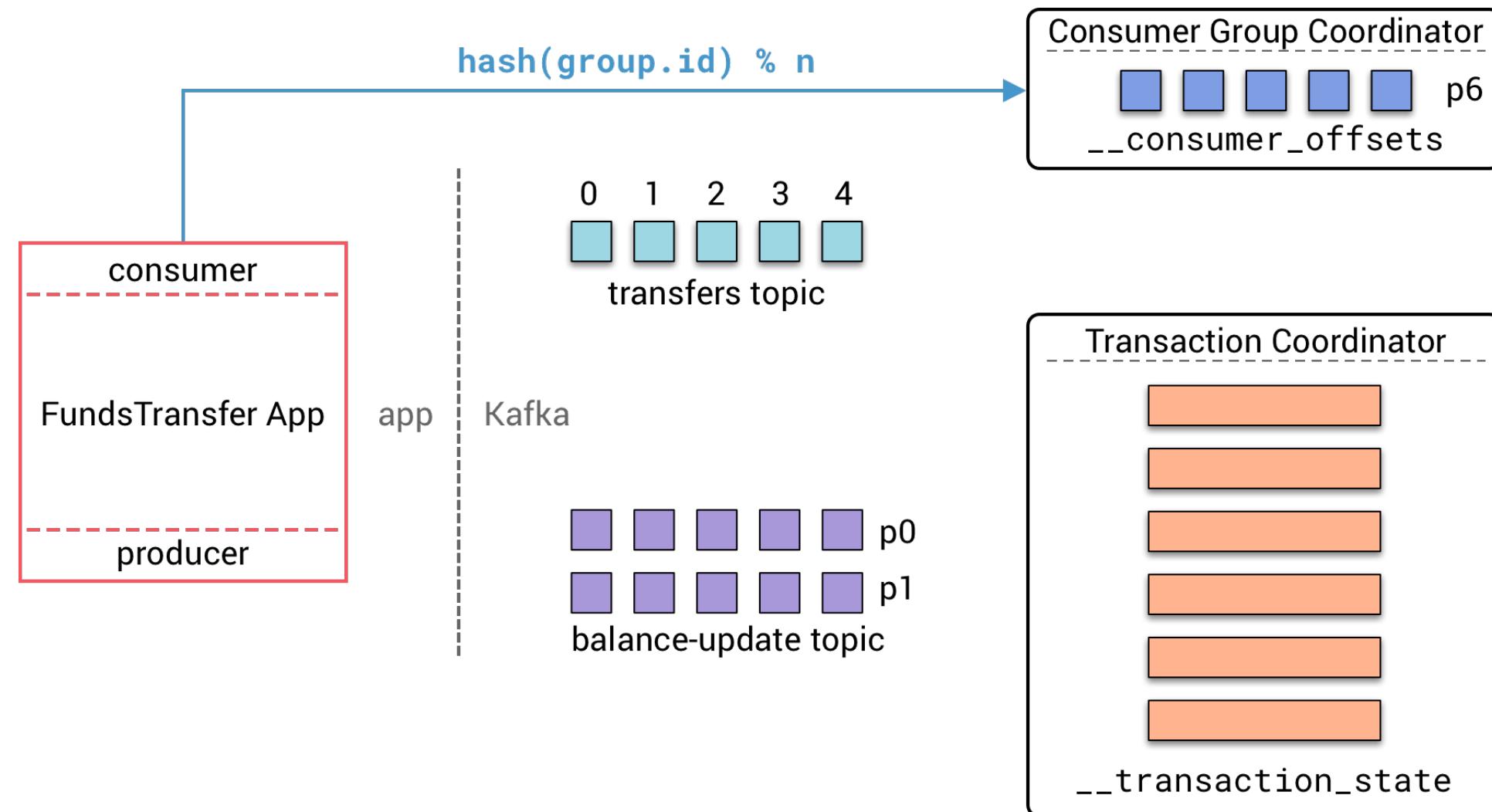
Description

This section presents a more detailed demo of a consume-process-produce application that uses transactions.

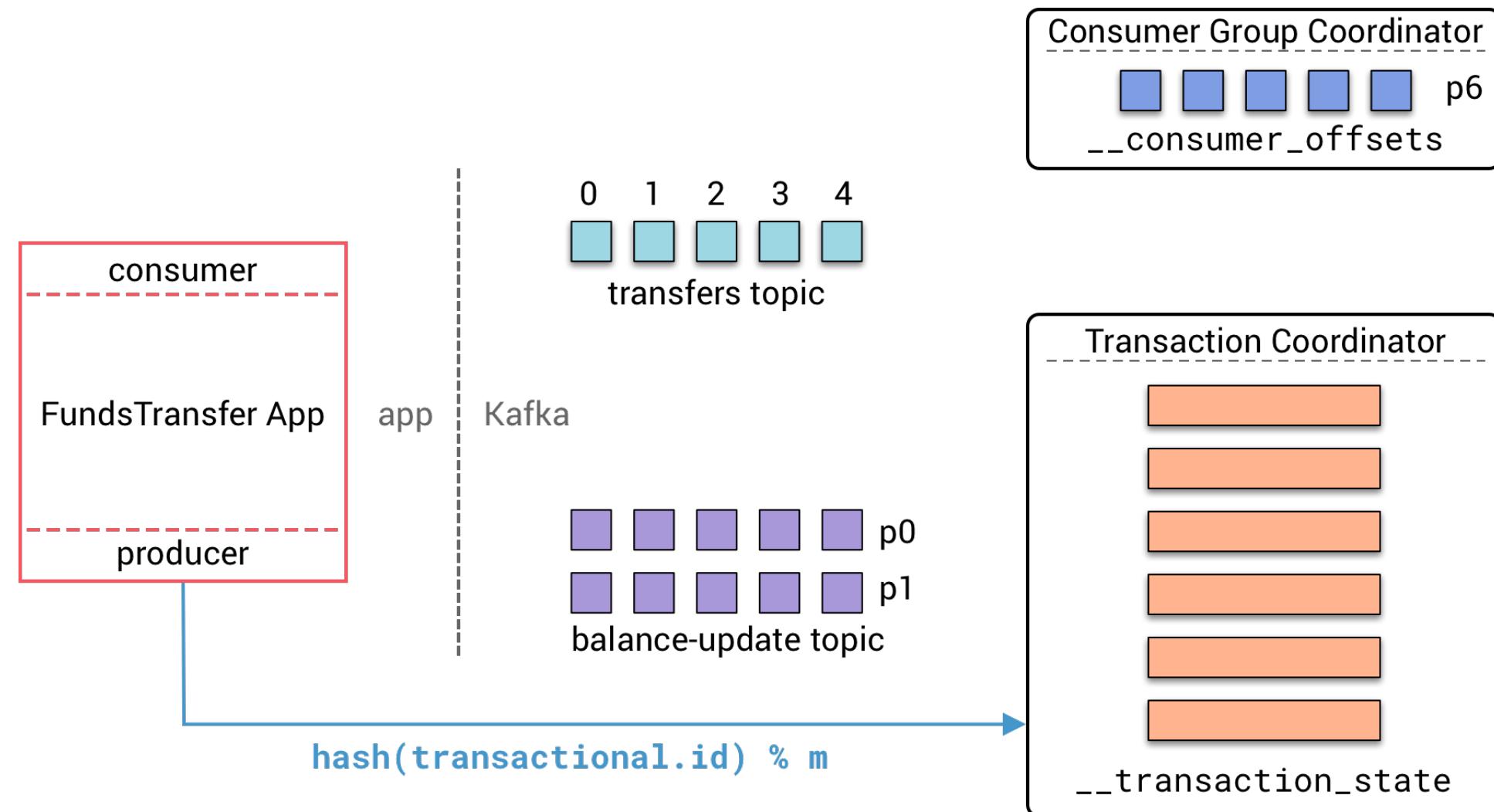
Transactions (1/14)



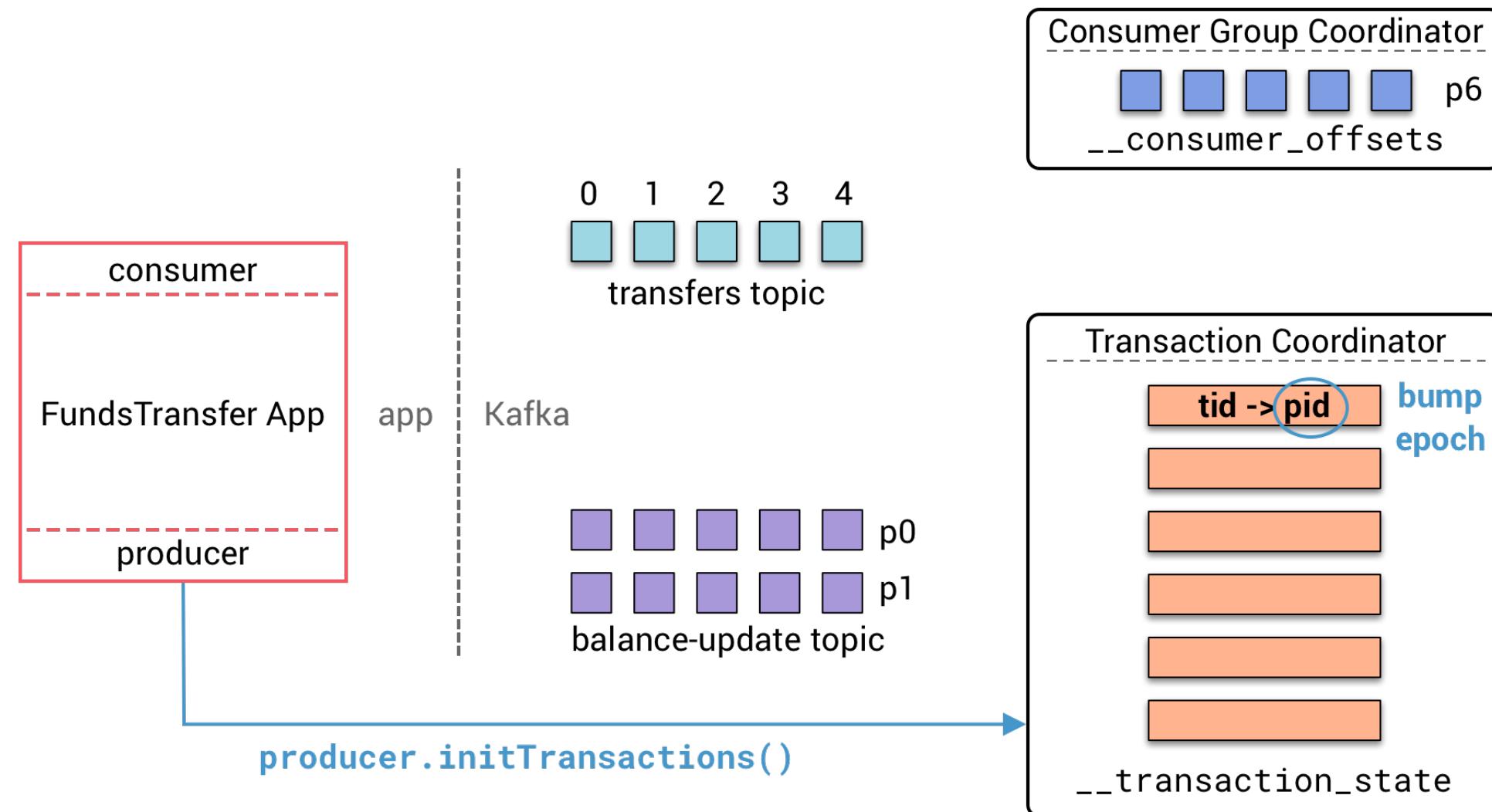
Transactions - Initialize Consumer Group (2/14)



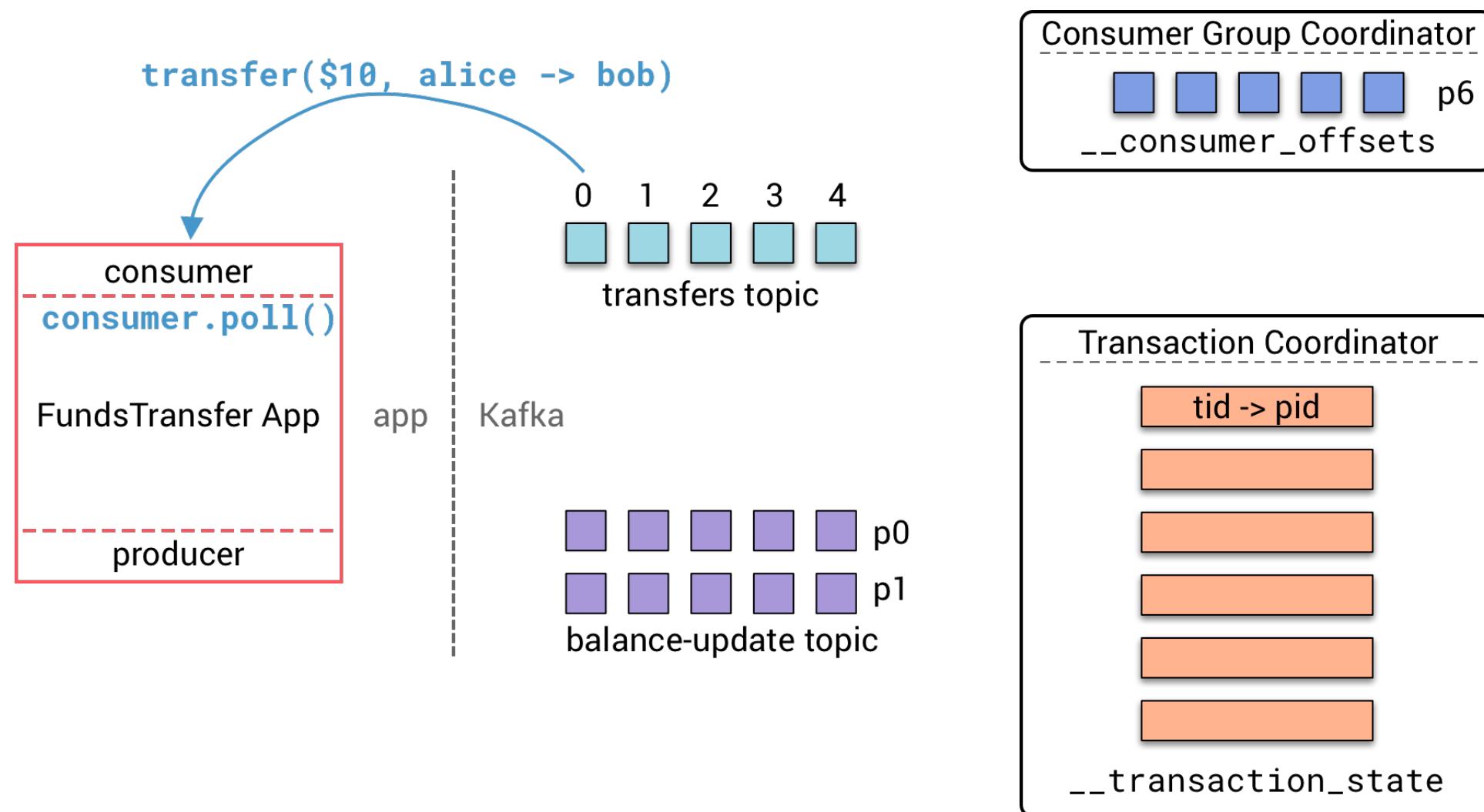
Transactions - Transaction Coordinator (3/14)



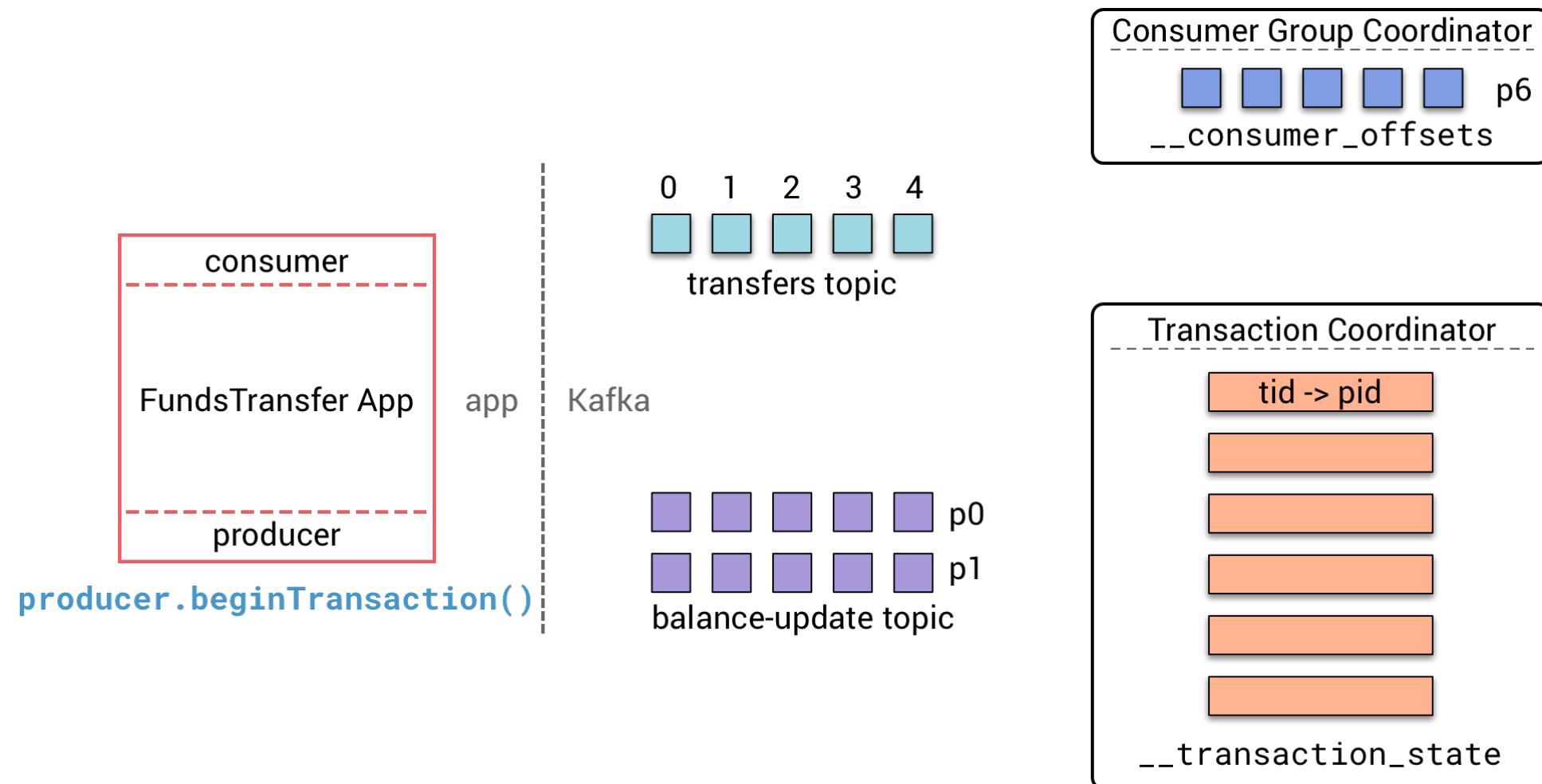
Transactions - Initialize (4/14)



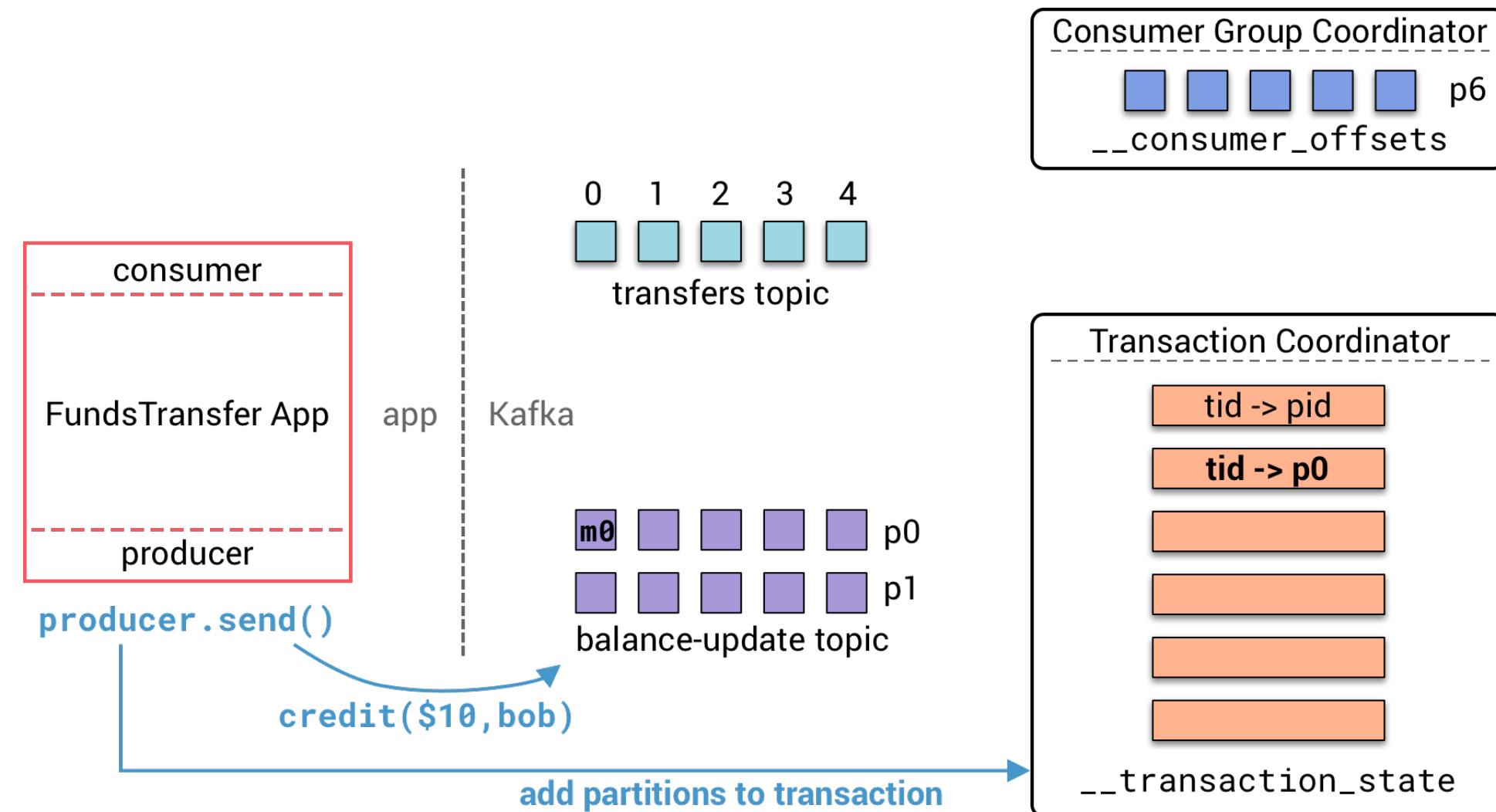
Transactions - Consume and Process (5/14)



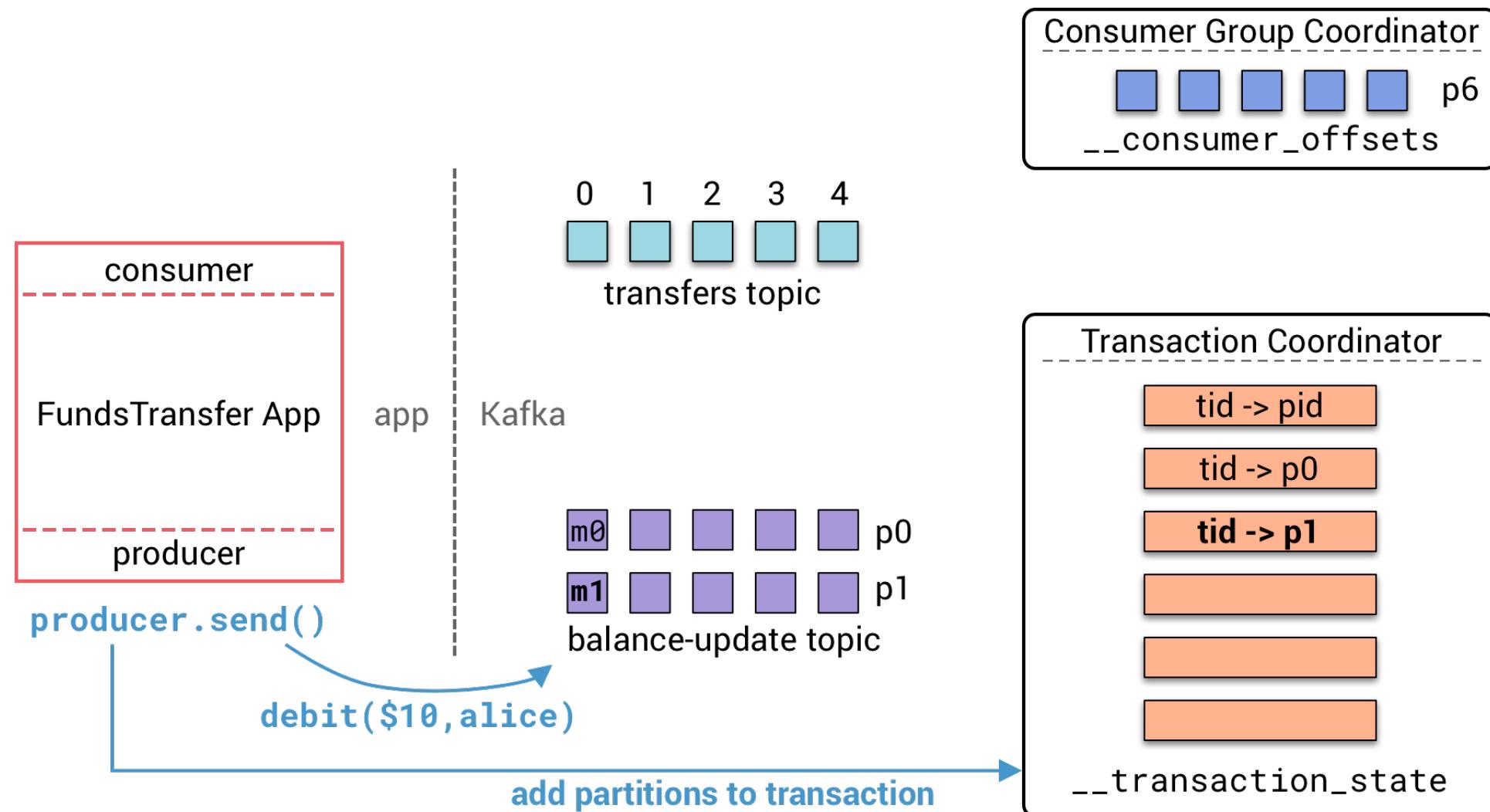
Transactions - Begin Transaction (6/14)



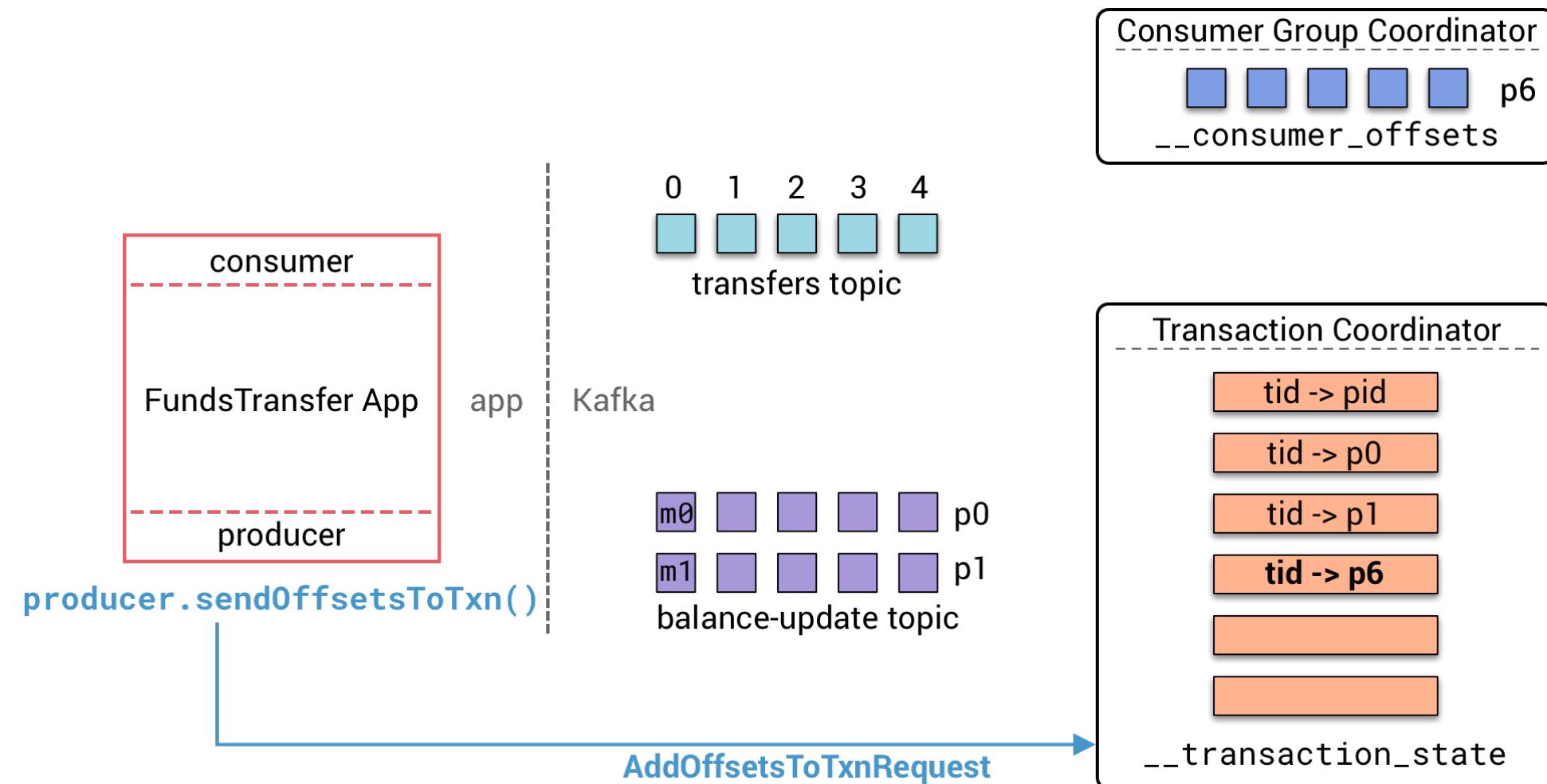
Transactions - Send (7/14)



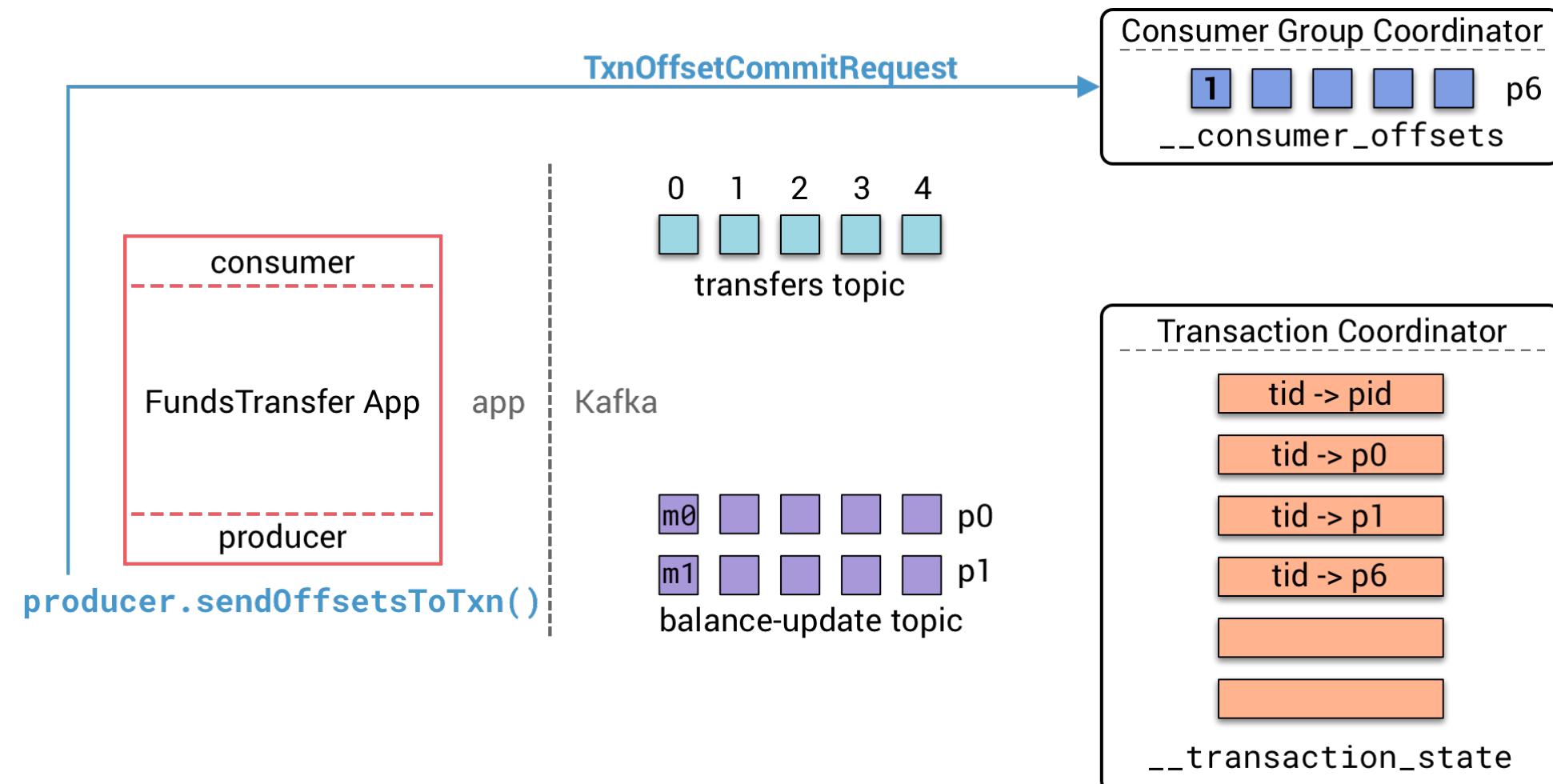
Transactions - Send (8/14)



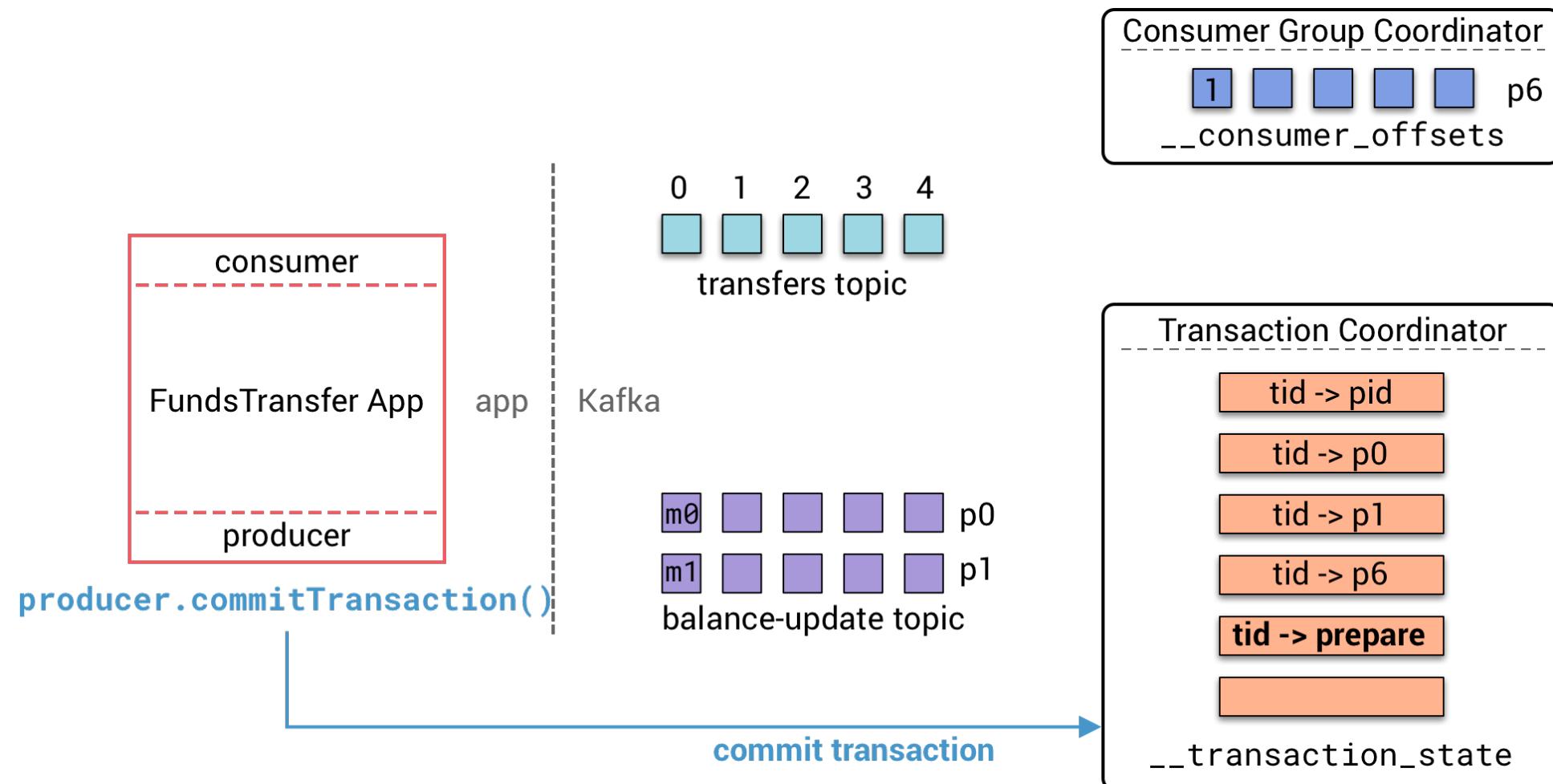
Transactions - Track Consumer Offset (9/14)



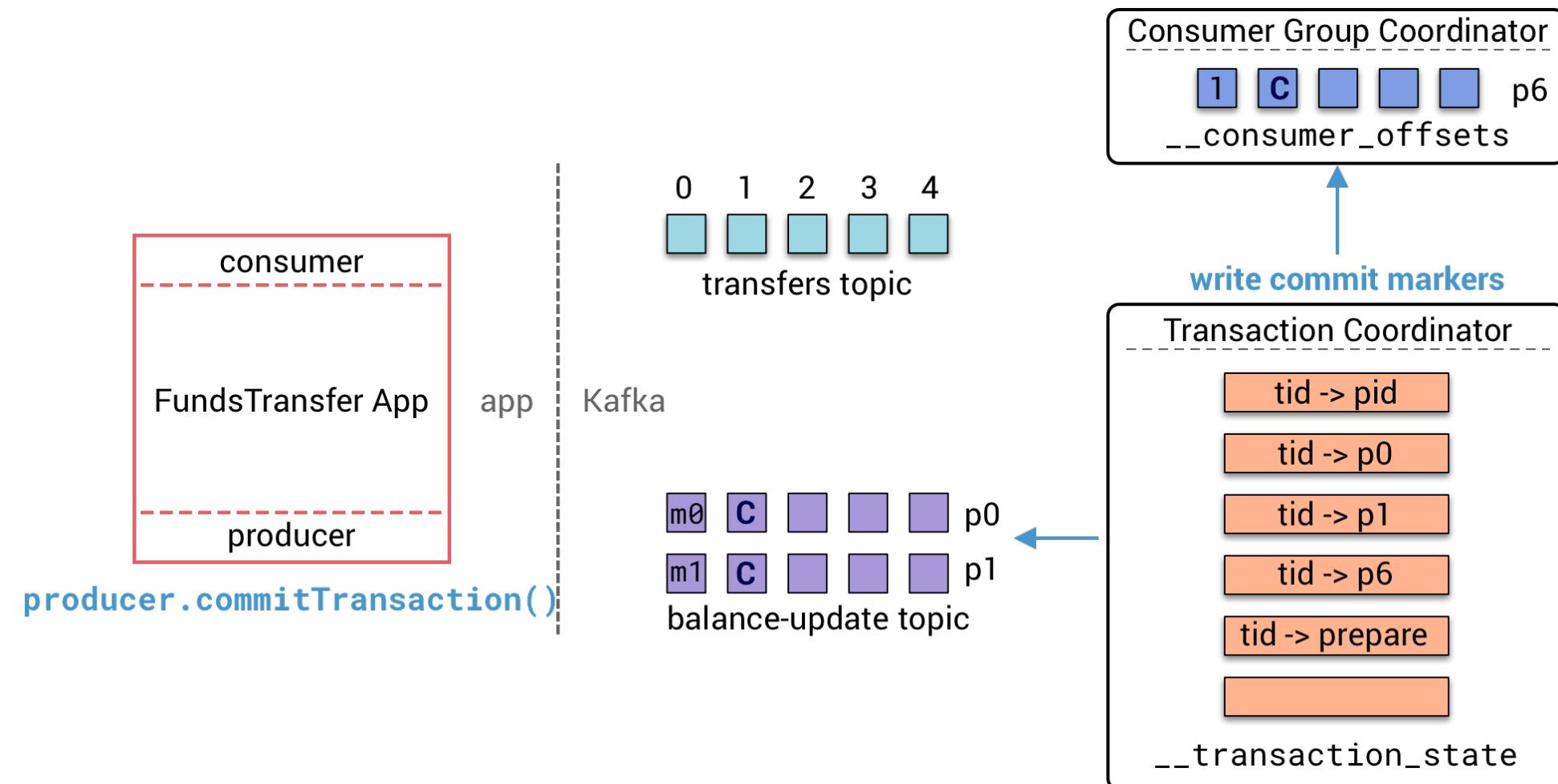
Transactions - Commit Consumer Offset (10/14)



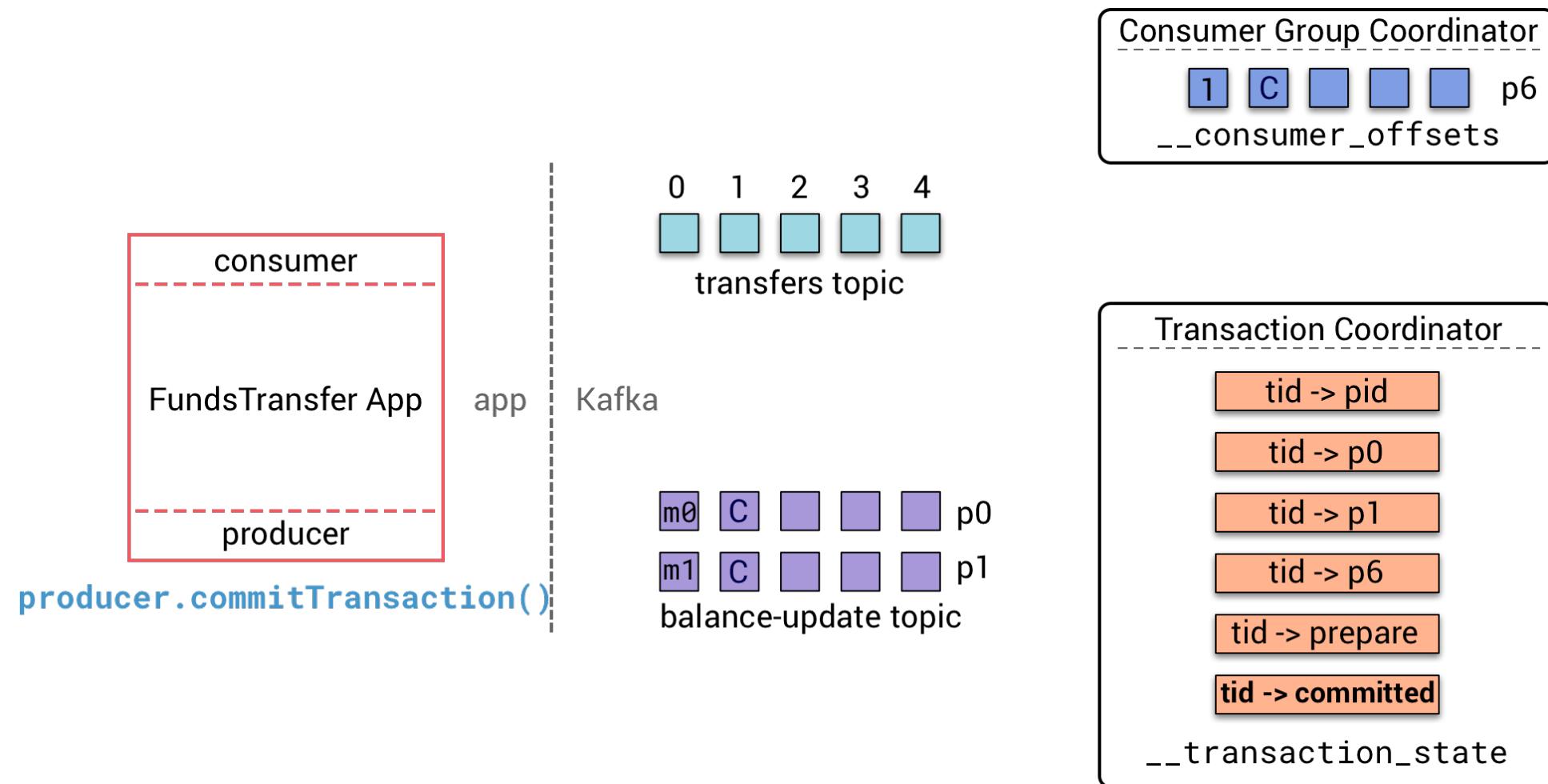
Transactions - Prepare Commit (11/14)



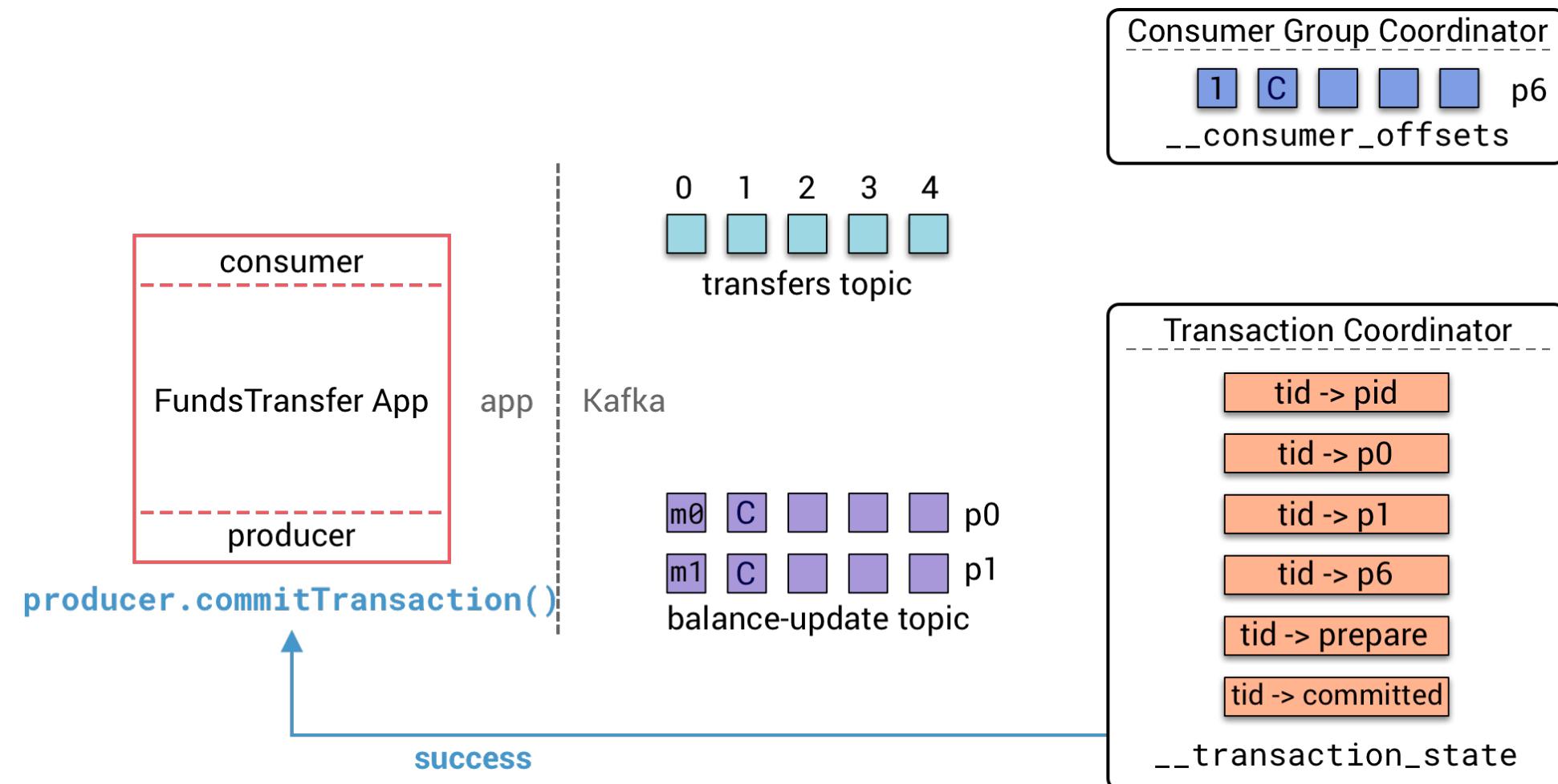
Transactions - Write Commit Markers (12/14)



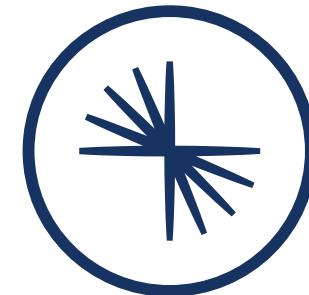
Transactions - Commit (13/14)



Transactions - Success (14/14)



Appendix: Confluent Technical Fundamentals of Apache Kafka® Content



CONFLUENT
Global Education

Module Overview

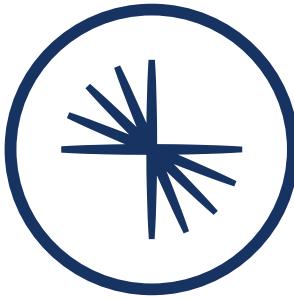


This section contains 5 lessons - the content lessons from the Fundamentals prerequisite:

Lessons of Presentation:

1. Getting Started
2. How are Messages Organized?
3. How Do I Scale and Do More Things With My Data?
4. What's Going On Inside Kafka?
5. Recapping and Going Further

1: Getting Started



CONFLUENT Global Education

Why Kafka?

In a nutshell...

Kafka is good for

- data in motion
- real-time processing

Kafka is not meant for

- batch processing
- archiving data

One Example Use Case: Ordering Food

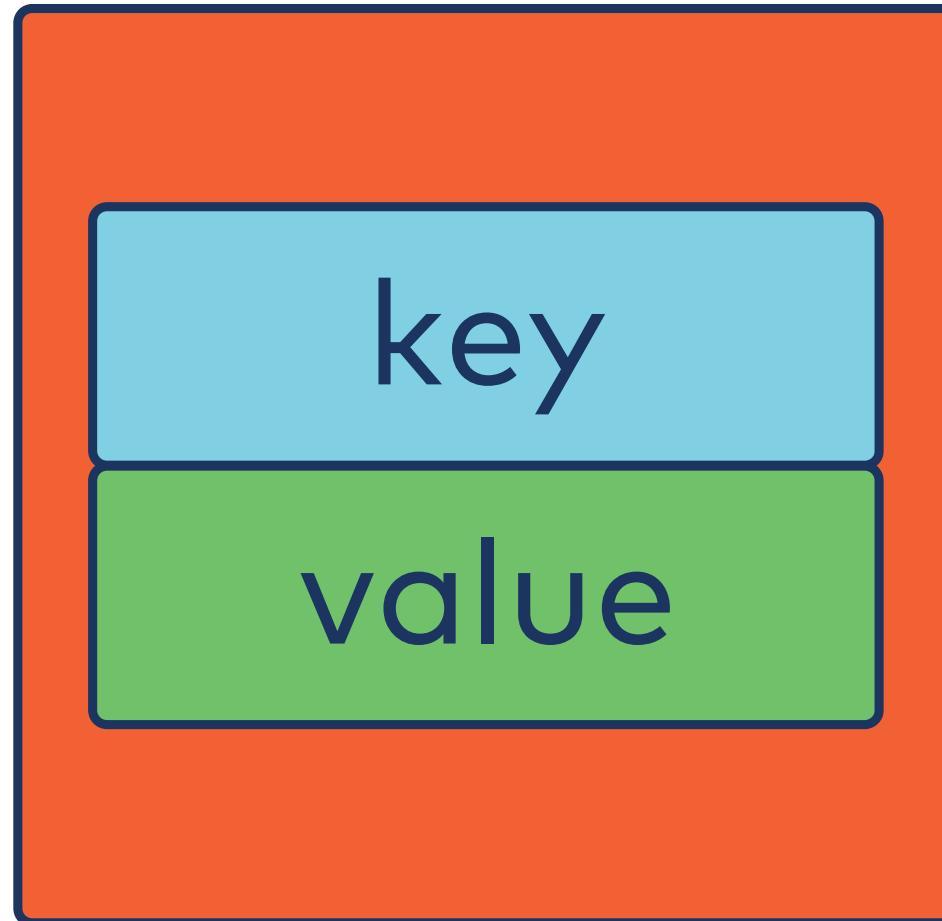
Suppose we are building a system for a restaurant chain:

- customers order food via an app - mobile or kiosk
- staff receive orders to fulfill in real-time
- management tracks inventory based on orders

We will build up some of the fundamental details of Kafka and use this example.

Messages

The atomic unit of Kafka is a **message** or **record** or **event**



Topics

Messages are organized in logical groups called **topics**.

Example topics:

- **orders**
- menu items
- customers
- restaurants

Three Basic Components

Let's start simple - with an **orders** topic in place in Kafka, a producer, and a consumer:



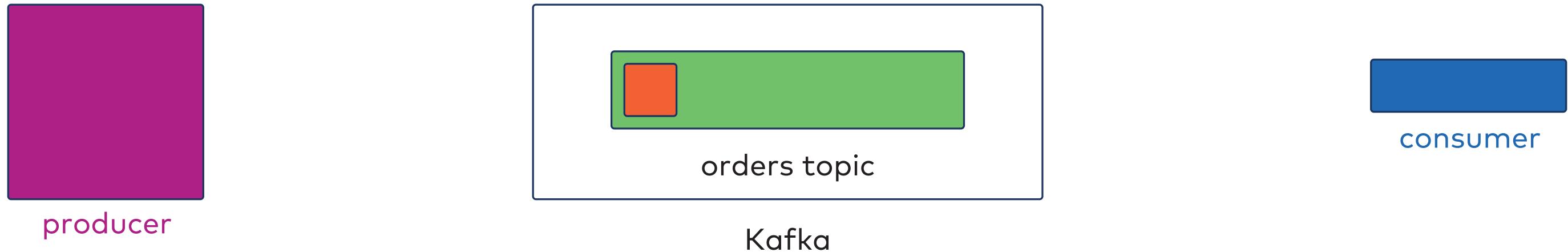
Life Cycle of a Message: Producing

A **producer** prepares messages and publishes them to Kafka.



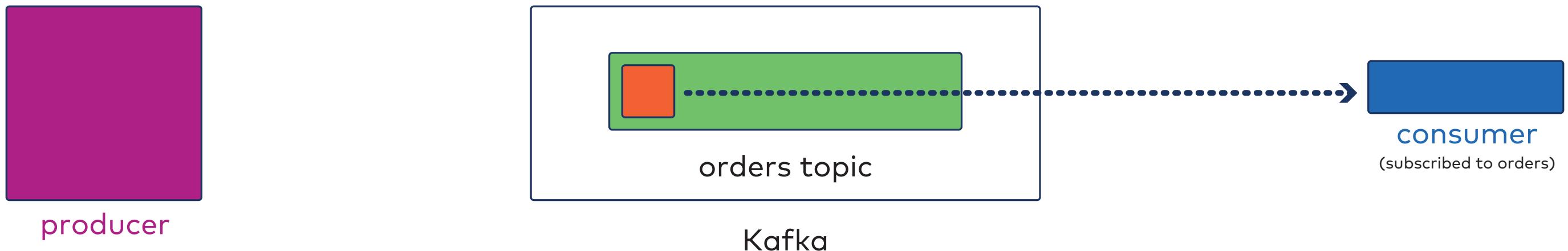
Life Cycle of a Message: Kafka

Produced messages live in Kafka, organized by topic.

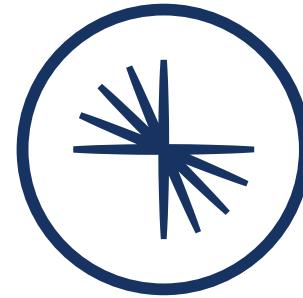


Life Cycle of a Message: Consumption

Consumers subscribe to topics in Kafka and poll for new messages.



2: How are Messages Organized?

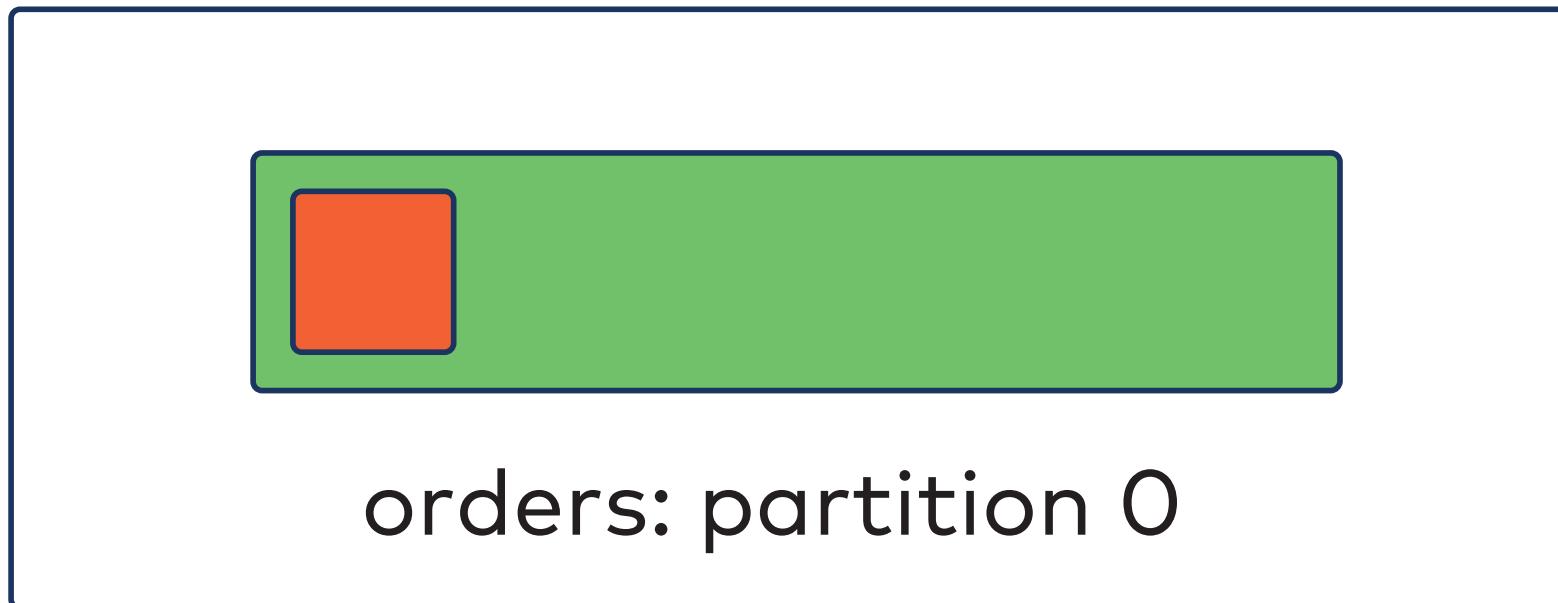


CONFLUENT
Global Education

Topics and Partitions

Topics are broken down into **partitions**

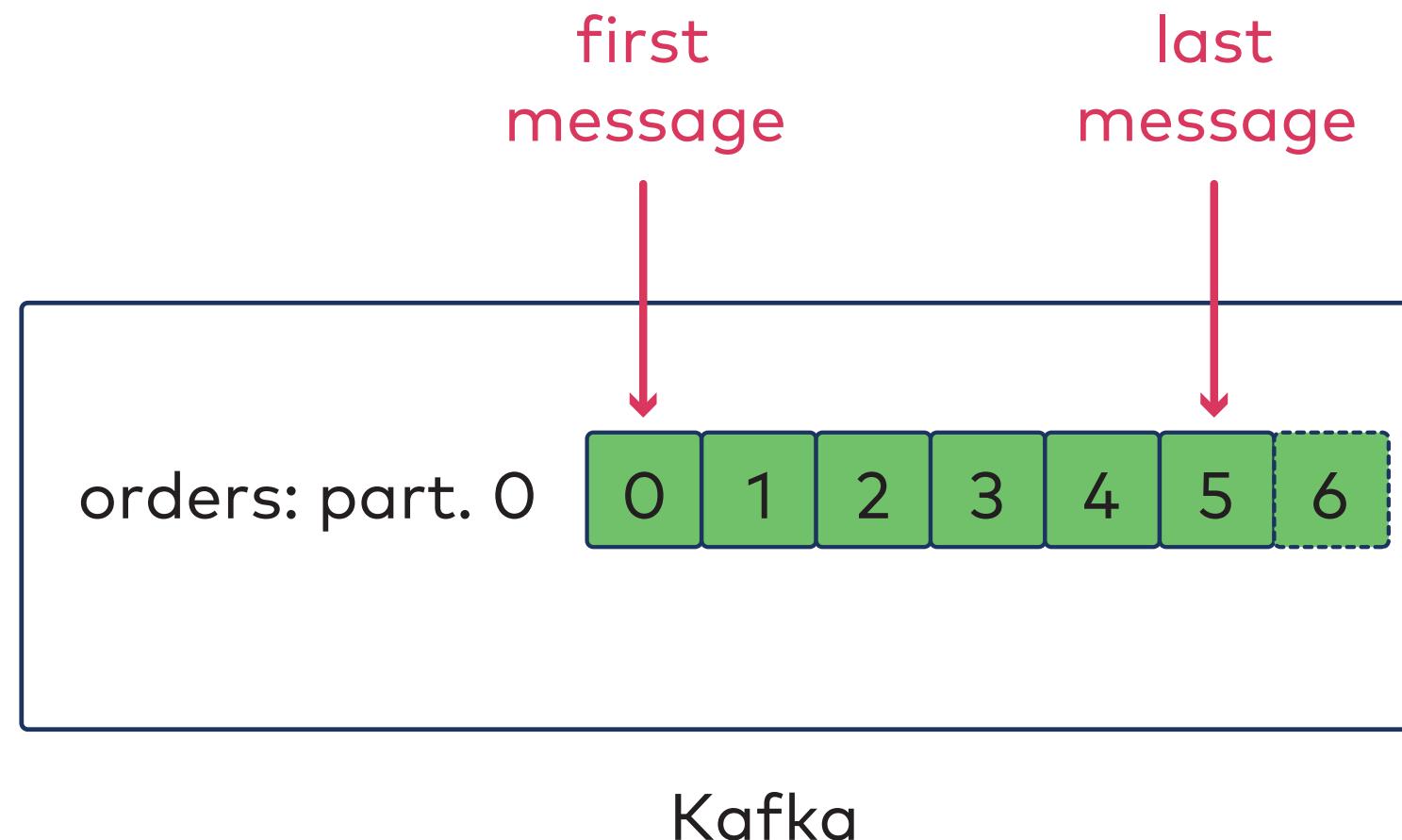
Simplest case: Topic with one partition



Kafka

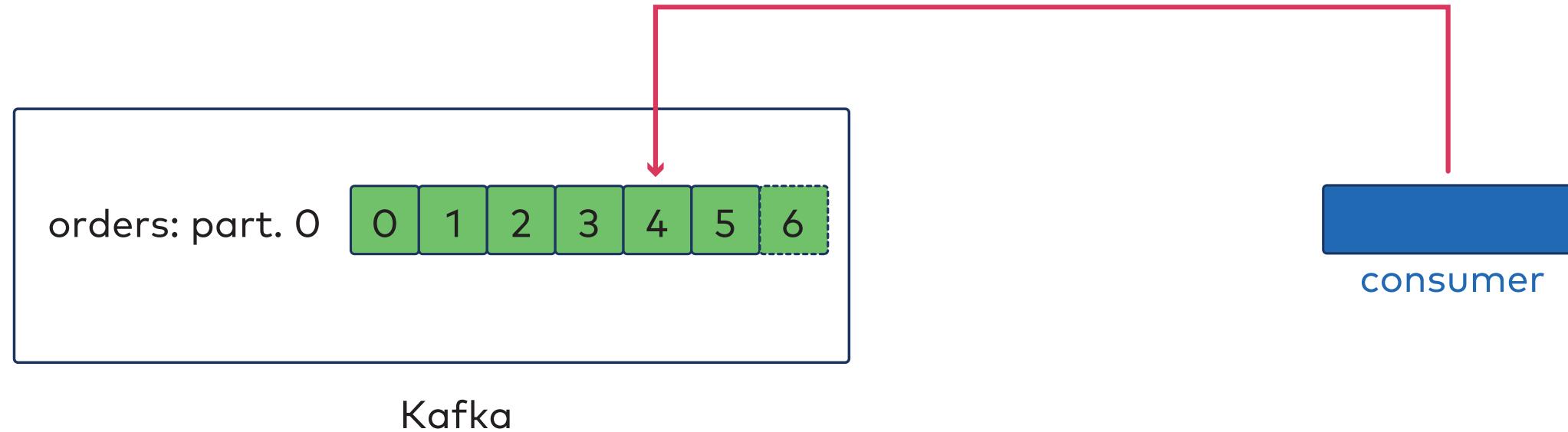
Offsets—in Kafka

- Each message in a partition has an **offset**
- Starting from 0



Offsets—Consumer Offsets

- Consumers track where they will read next via a **consumer offset**



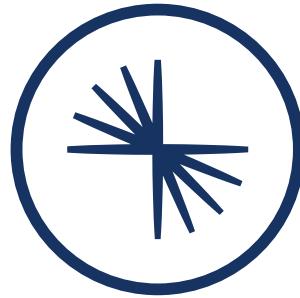
In this picture, the consumer has last read the message at offset 3.

Check Your Knowledge!

Try a [quick quiz on Lessons 1 and 2](#).



3: How Do I Scale and Do More Things With My Data?



CONFLUENT
Global Education

Scaling Up...

So far, we have seen...

- one partition
- one consumer

In practice...

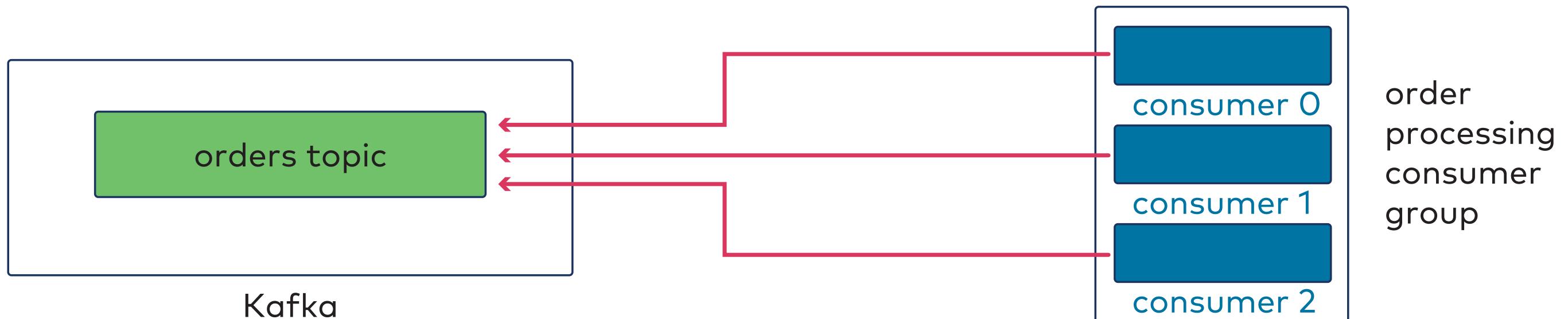
- multiple consumers in a consumer group
- multiple consumer groups
- multiple partitions in a topic

Consumer Groups

Consumers exist in **consumer groups**

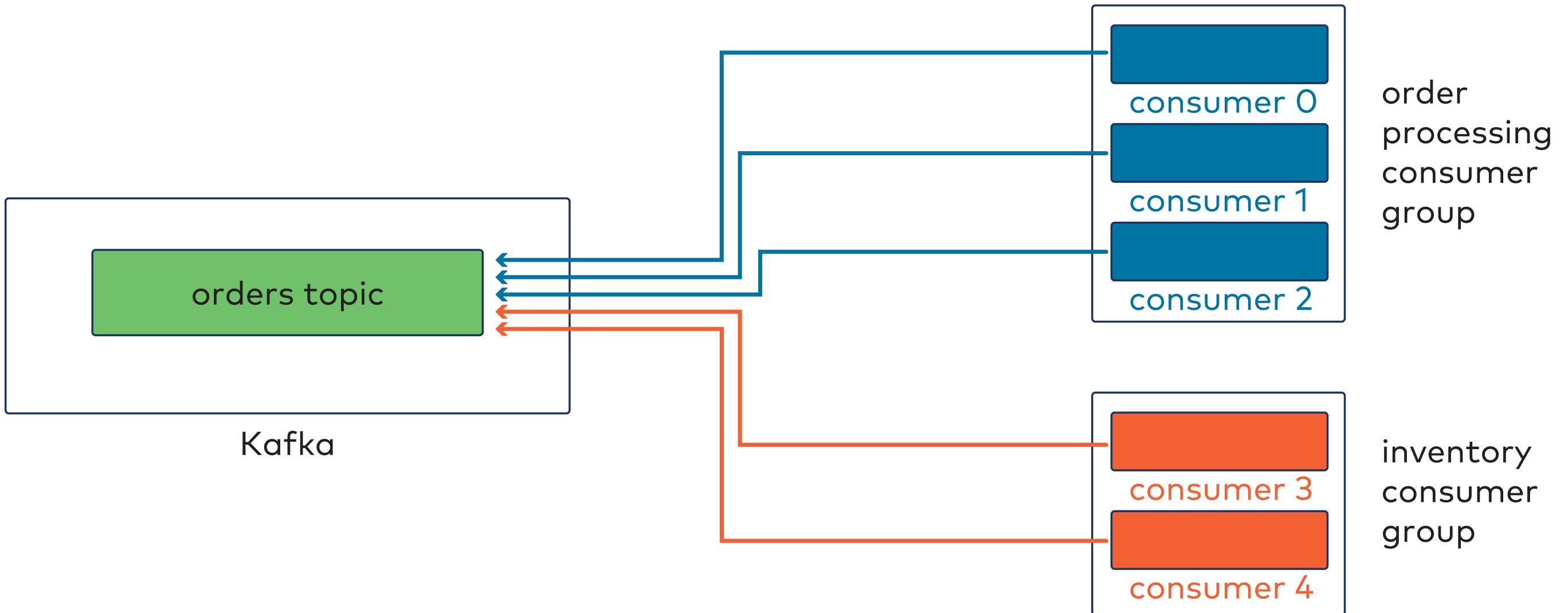
Consumers in a group:

- **same application**
- **different data**



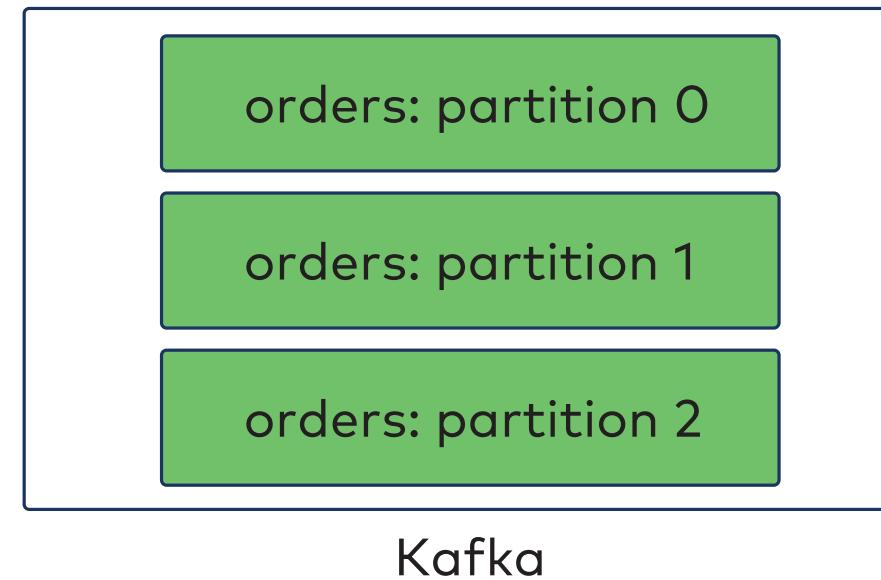
Multiple Consumption

Could have multiple groups using the same data...



Multiple Partitions

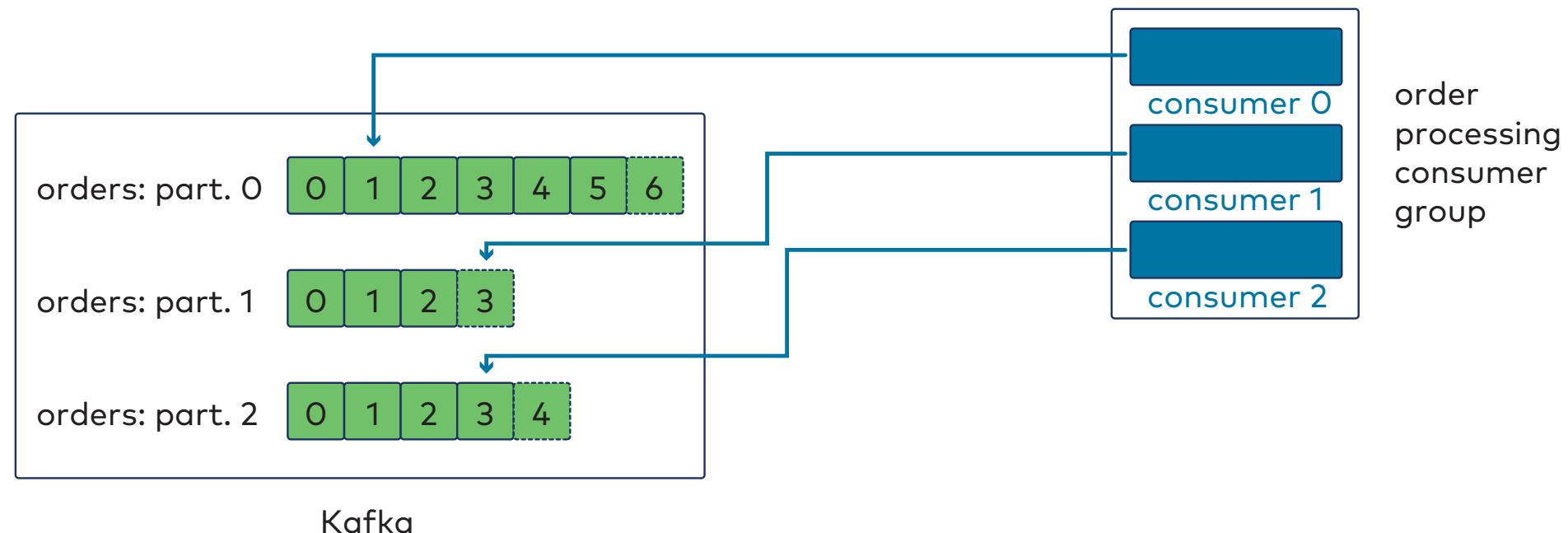
In practice, we want topics to have multiple partitions.



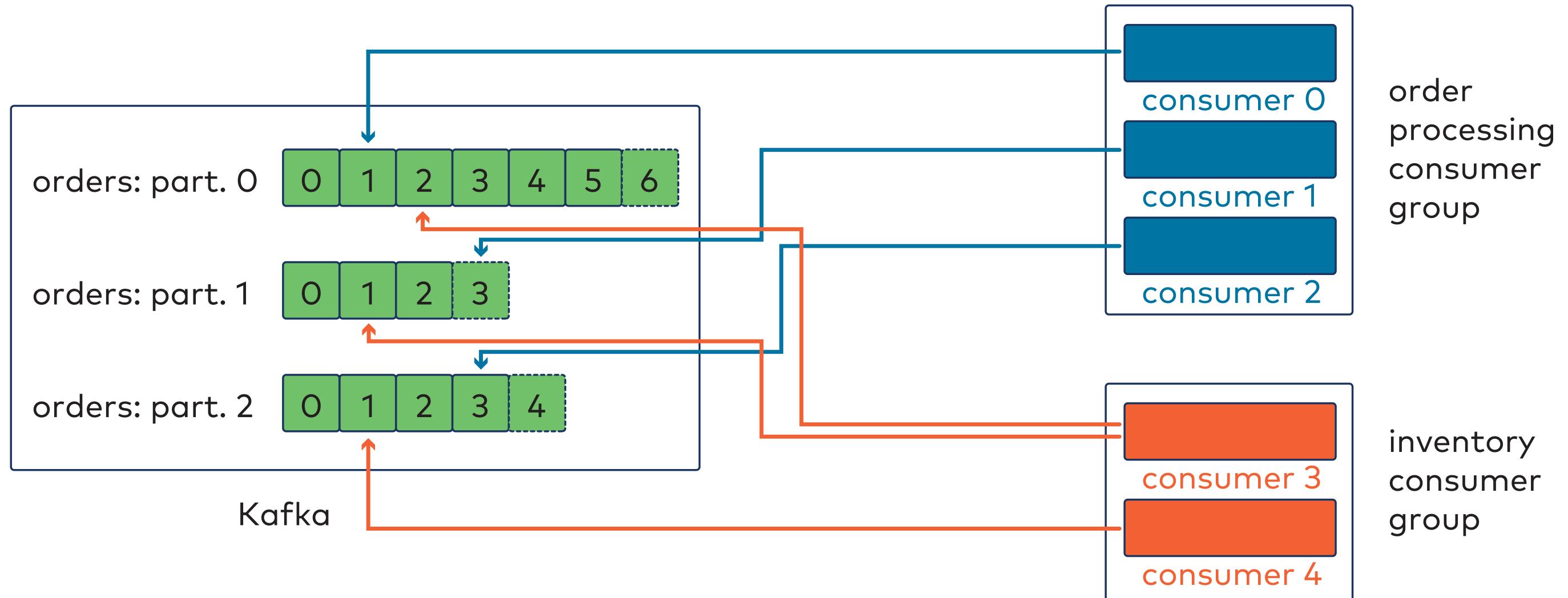
Consuming from Multiple Partitions

Now our consumers can consume in parallel:

- Consumers subscribed to a topic are assigned partitions
- Group covers all partitions
- Each consumer has an offset for each partition



Expanding the Last Picture



How Do Messages Get Partitioned?

- **Producers** decide which messages go to which partition
- Partitions are indexed from `0` to `numberOfPartitions - 1`
- Default partitioner: `partitionIndex = hash(key) % numberOfPartitions`

Scaling is Easy!

Say you want to

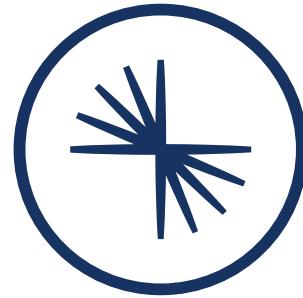
- Increase the number of consumers in a group during a busy season
- Decrease the number of consumers in a group when things are slow
- Increase the number of partitions for a topic

When you do, Kafka *automatically* redistributes the assignments of consumers to partitions!



More on how all of this works in both our Developer and Administrator training!

4: What's Going On Inside Kafka?



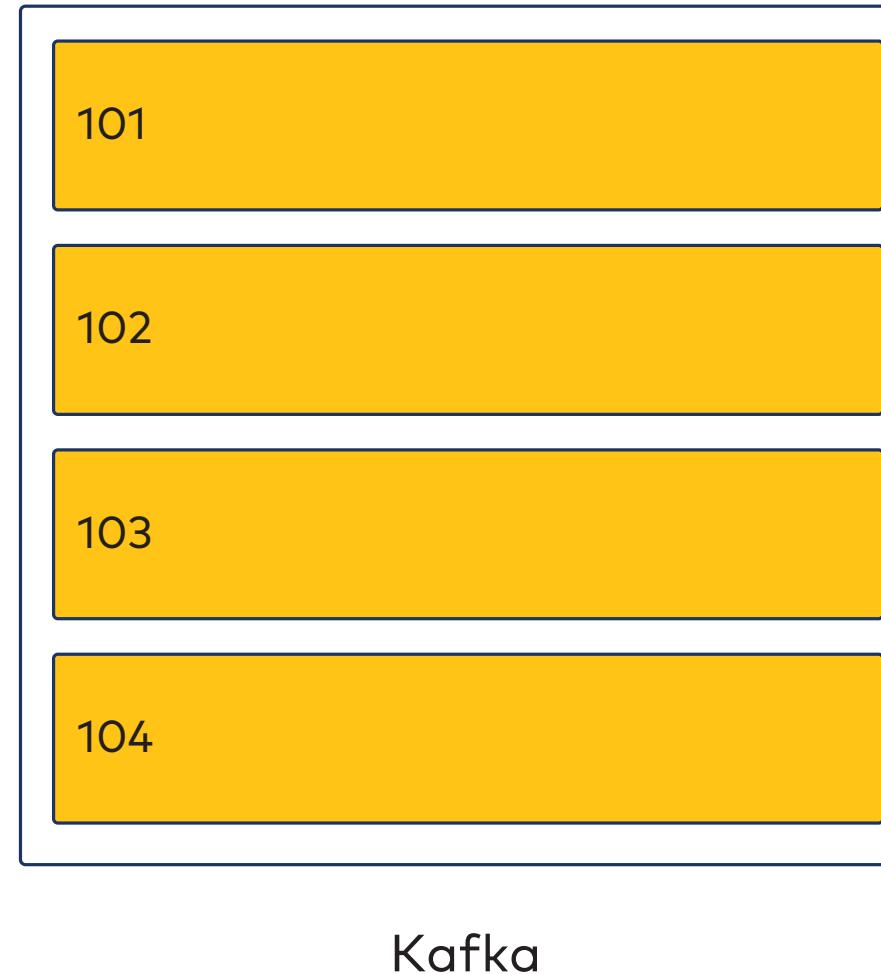
CONFLUENT
Global Education

Going Deeper...

Now let's learn about some more details about a Kafka cluster, especially *physical* things...

Brokers

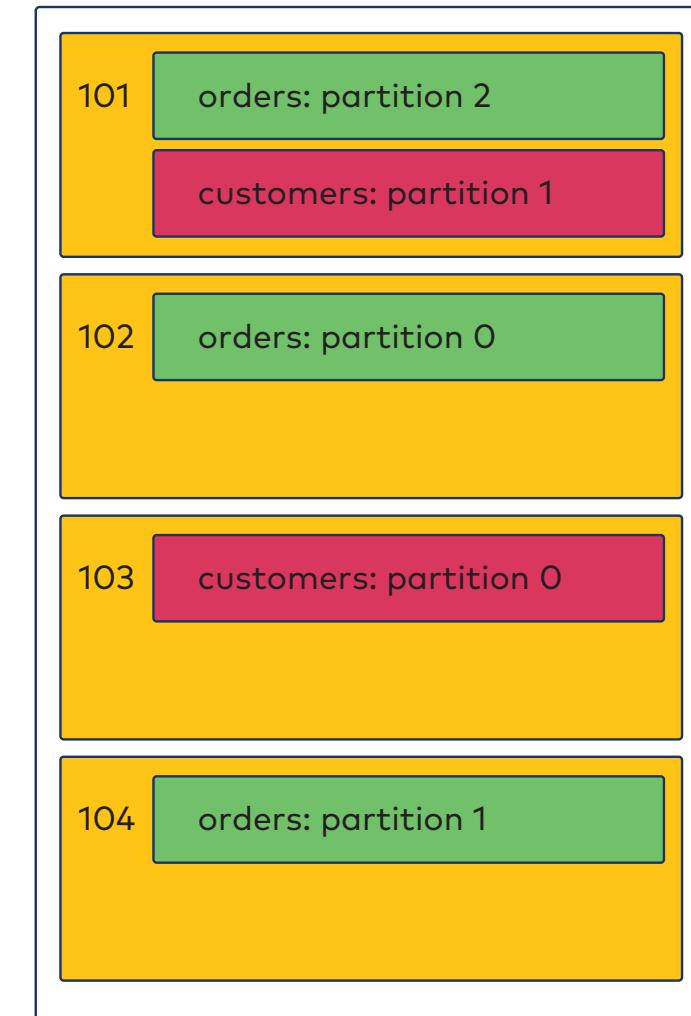
A Kafka cluster consists of multiple **brokers**



Partitions & Brokers (2)

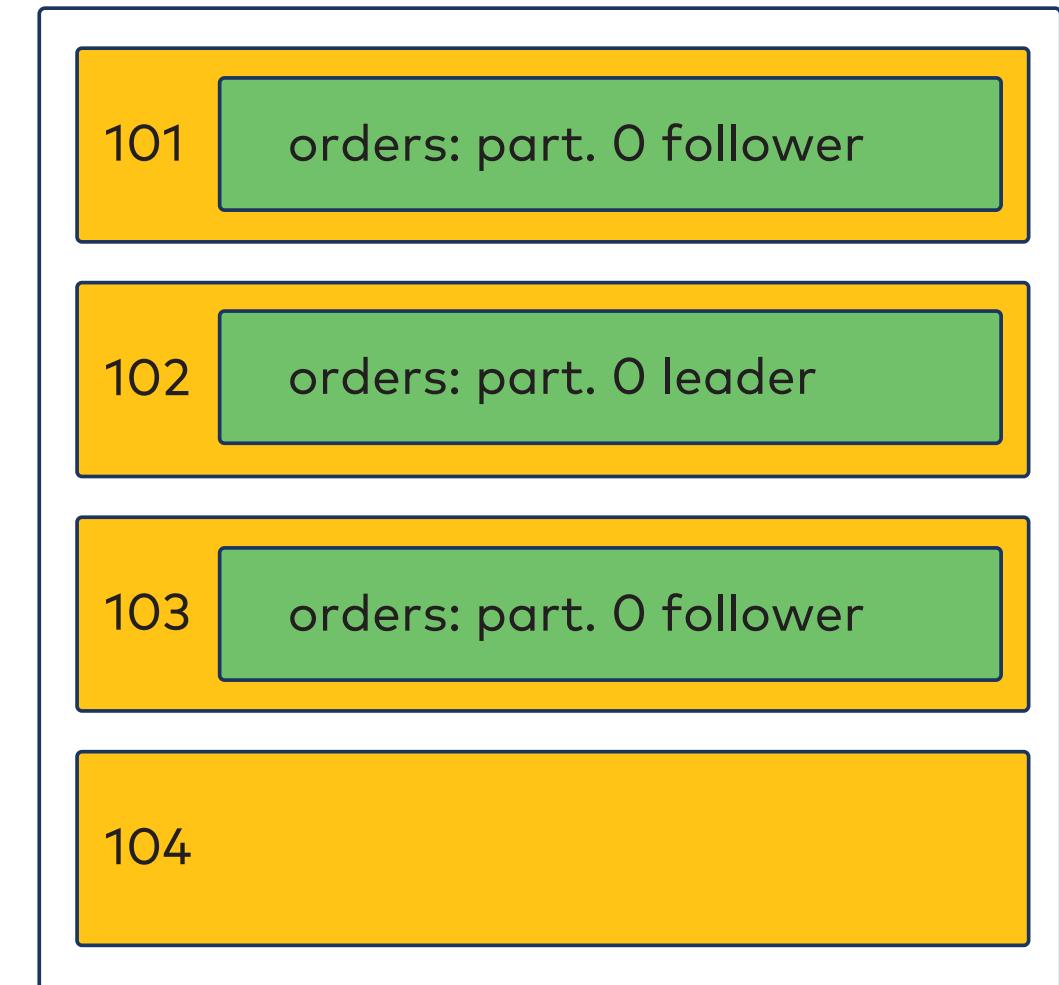
The number of partitions is a topic setting.

Kafka decides how partitions get distributed across brokers.



What if a Broker Goes Down?

- Want *high availability* of data in partitions
- Achieved via **replication**
- Writes are reads go to **leader** replica
- **Follower** replicas keep backup copies of the leader
- If leader dies, a follower becomes the leader



Kafka

Serialization and Deserialization

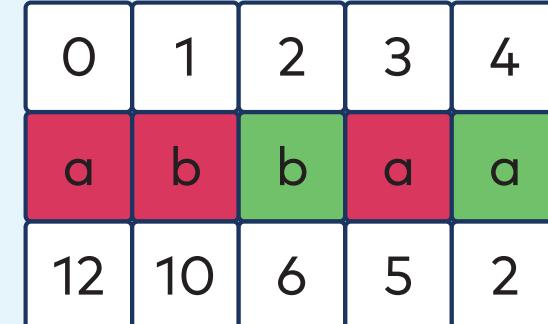
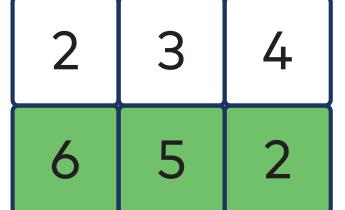
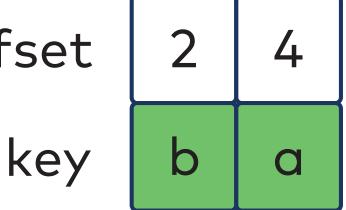
- Kafka stores messages as byte arrays
- Producers must **serialize** messages
- Consumers must **deserialize** messages

Immutable Messages

- Messages are **immutable**
- Once written, we cannot change anything about them

...But We Don't Keep Messages Forever...

Control which messages stay in Kafka via a **retention policy**:

Policy	Deletion	Compaction
Idea	Remove messages older than a certain age (default 7 days)	Keep only the latest value for each key
Before	offset  key age in days	offset  key age in days
After	offset  age in days	offset  key



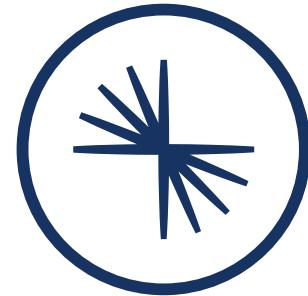
Partitions are divided into segments, which affect both retention policies.

Check Your Knowledge!

Try a [quick quiz on Lessons 3 and 4.](#)

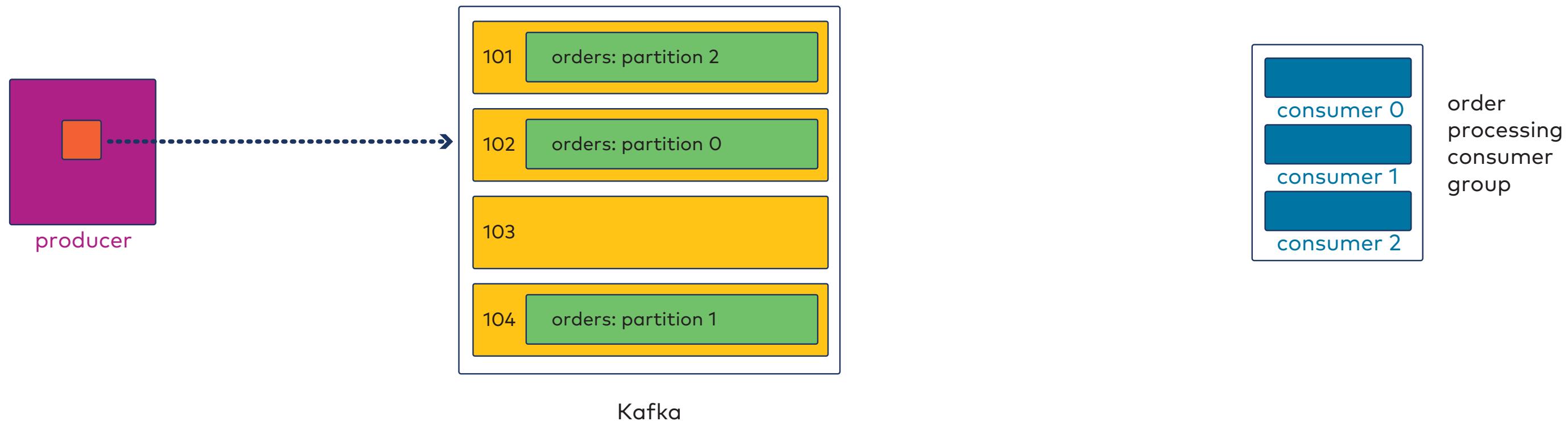


5: Recapping and Going Further



CONFLUENT
Global Education

Life Cycle of a Message: Producing



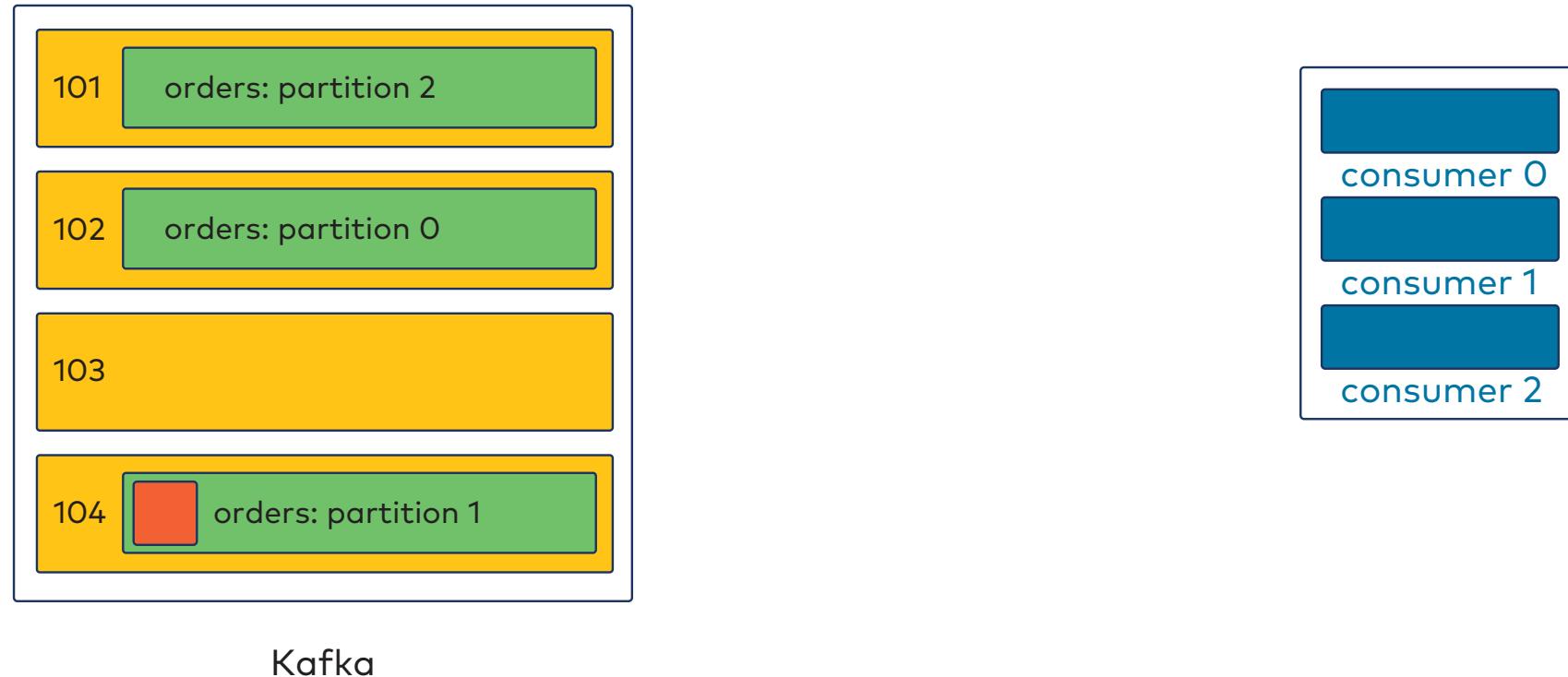
- Producers serialize and partition messages
- Producers send messages
 - ...in batches - can be configured for throughput and latency desires

Life Cycle of a Message: Kafka

Produced messages live in Kafka, organized by topic.



producer



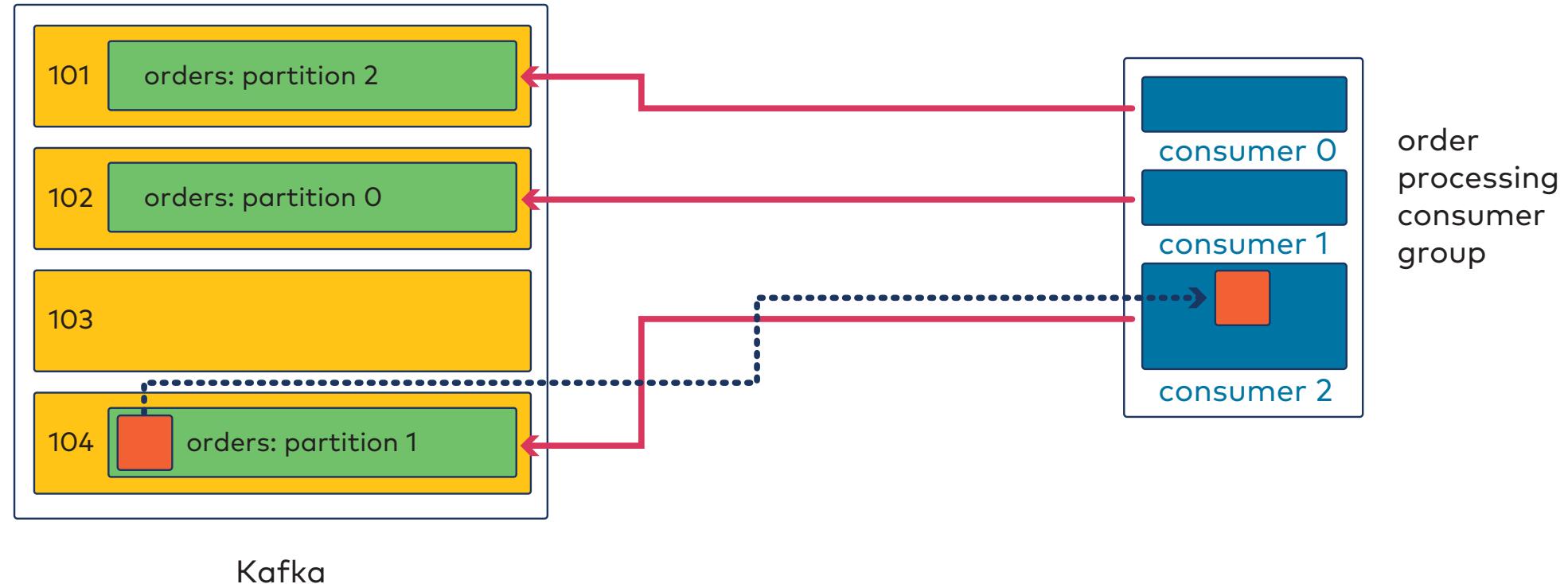
- Kafka consists of brokers
- Brokers contain partitions, which contain messages
- Brokers handle retention and replication

Life Cycle of a Message: Consumption

Consumers subscribe to topics in Kafka and poll for new messages.



producer



- Consumers operate in groups
- Consumers subscribe to topics, are assigned partitions of those topics
- Consumers poll for messages in partitions at consumer offsets
 - ...and fetch in batches - can be configured for throughput and latency desires

A Step Beyond Fundamentals: Other Components

We've addressed some aspects of Core Kafka in this course. Some other topics you may want to learn about include:

- **Kafka Connect** - a tool that helps you copy data to Kafka from other systems and vice-versa
- **Kafka Streams** - a layer on top of the Producer and Consumer APIs that allows for stream processing
- **Confluent ksqlDB** - a tool for stream processing using a more-accessible SQL-like syntax, among other things
- **Confluent Schema Registry** - a tool for managing schemas, guiding schema evolution, and enforcing data integrity

You can learn more about these topics in our Confluent Developer Skills for Building Apache Kafka® and Apache Kafka® Administration by Confluent courses.

What Does Confluent Platform Add to Kafka?

CONFLUENT PLATFORM

SECURITY & RESILIENCY

RBAC | Audit Logs | Schema Validation | Multi-Region Clusters | Replicator | Cluster Linking

PERFORMANCE & SCALABILITY

Tiered Storage | Self-Balancing Clusters | K8s Operator

MANAGEMENT & MONITORING

Control Center | Proactive Support

DEVELOPMENT & CONNECTIVITY

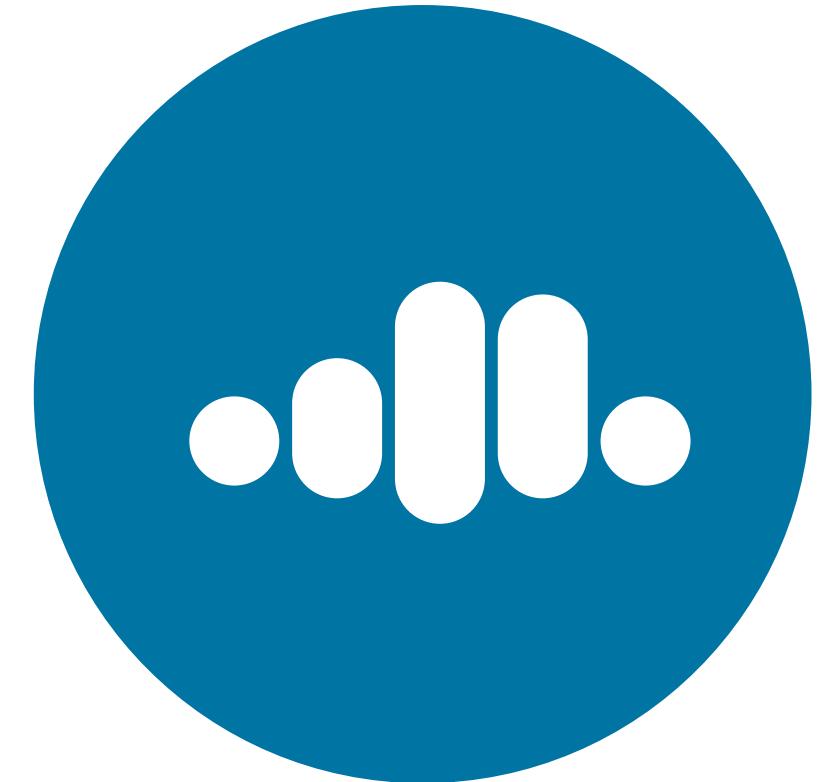
Connectors | Non-Java Clients | REST Proxy | Schema Registry | ksqlDB

APACHE KAFKA®

Core | Connect API | Streams API

Confluent Cloud

- Can deploy CP as self-managed software but...
- Confluent Cloud = **fully-managed** deployment of CP
 - Many administrative tasks done for you
- Confluent Cloud available on
 - AWS
 - Google Cloud Platform
 - Microsoft Azure



Your Next Steps

1. Complete interactive lab on seeing console producers and consumers in action.
 - [Short Confluent Cloud version](#)
 - Gitpod version: [More involved version using Gitpod](#)
2. Work though other Critical Thinking Challenge Exercises.
 - [On the web](#)
 - Solutions on the web too!
3. Enroll in and complete one of these courses, as suits your role:
 - Apache Kafka® Administration by Confluent
 - Confluent Developer Skills for Building Apache Kafka®

Labs:



Critical
Thinking:



Thank You

Thank you for attending the course!