

# Apache Kafka® Administration By Confluent

Version 7.8.1-v1.0.0



# CONFLUENT

# Table of Contents

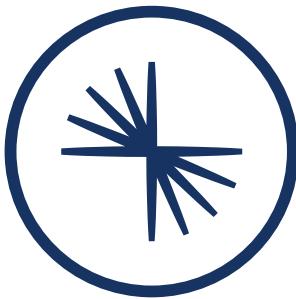
<b>Introduction</b>	1
Class Logistics and Overview	2
Fundamentals Review	9
<b>1: Bridging From Fundamentals</b>	11
1a: How Can You Leverage Replication?	15
Labs: Bridging From Fundamentals	23
<b>2: Replicating Data: A Deeper Dive</b>	24
2a: How Does Kafka Determine Which Messages Can be Consumed?	26
2b: How Does Kafka Place Replicas and How Can You Control Replication Further?	35
2c: How Does Kafka React When a Leader Dies?	43
2d: How Does Kafka Track Follower Responsiveness?	46
<b>3: Producing Messages Reliably</b>	50
3a: How Do Producers Know Brokers Received Messages?	52
3b: How Can Kafka recognize Duplicates caused by Retries?	58
3c: How Does Kafka Handle the Notion of Producers Sending Messages in Transactions?	61
<b>4: Storing the Records Persistently</b>	68
4a: How Does Kafka Organize Files to Store Partition Data?	70
Lab: Investigating the Distributed Log	77
4b: How Can You Decide How Kafka Keeps Messages?	78

4c: How to Scale Storage Beyond Kafka Servers?	90
<b>5: Configuring a Kafka Cluster</b>	<b>95</b>
5a: How Do You Configure Brokers?	97
5b: What if You Want to Adjust Settings Dynamically or Apply at the Topic Level?	104
Labs: Configuring a Kafka Cluster	115
<b>6: Managing a Kafka Cluster</b>	<b>116</b>
6a: What Should You Consider When Installing and Upgrading Kafka?	118
6b: What is a Controller vs a Broker?	124
6c: What are the Basics of Monitoring Kafka?	130
6d: How Can You Move Partitions To New Brokers Easily?	139
6e: What Should You Consider When Shrinking a Cluster?	146
Lab: Kafka Administrative Tools	149
<b>7: Balancing Load with Consumer Groups and Partitions</b>	<b>150</b>
7a: What are the Basics of Scaling Consumption?	152
7b: How Do Groups Distribute Work Across Partitions?	157
7c: How Does Kafka Manage Groups?	163
7d: How Do Partitions and Consumers Scale?	169
7e: How Does Kafka Maintain Consumer Offsets?	176
Lab: Modifying Partitions and Viewing Offsets	182
<b>8: Optimizing Kafka's Performance</b>	<b>183</b>
8a: How Does Kafka Handle the Idea of Sending Many Messages at Once?	186
Lab: Exploring Producer Performance	

<b>8b: How Do Produce and Fetch Requests Get Processed on a Broker?</b>	195
<b>8c: How Can You Measure and Control How Requests Make It Through a Broker?</b>	201
<b>8d: What Else Can Affect Broker Performance?</b>	212
<b>8e: How Do You Control It So One Client Does Not Dominate the Broker Resources?</b>	218
<b>8f: What Should You Consider in Assessing Client Performance?</b>	227
<b>8g: How Can You Test How Clients Perform?</b>	232
<b>Lab: Performance Tuning</b>	236
<b>9: Securing a Kafka Cluster</b>	237
<b>9a: What are the Basic Ideas You Should Know about Kafka Security?</b>	239
<b>9b: What Options Do You Have For Securing a Kafka/Confluent Deployment?</b>	246
<b>9c: How Can You Easily Control Who Can Access What?</b>	251
<b>9d: What Should You Know Securing a Deployment Beyond Kafka Itself?</b>	265
<b>Lab: Securing the Kafka Cluster</b>	270
<b>10: Understanding Kafka Connect</b>	271
<b>10a: What Can You Do with Kafka Connect?</b>	273
<b>10b: How Do You Configure Workers and Connectors?</b>	283
<b>10c: Deep Dive into a Connector &amp; Finding Connectors</b>	292
<b>10d: What Else Can One Do With Connect?</b>	300
<b>Lab: Running Kafka Connect</b>	303
<b>11: Deploying Kafka in Production</b>	304
<b>11a: What Does Confluent Advise for Deploying Brokers in Production?</b>	307
<b>11b: What Does Confluent Advise for Deploying Kafka Connect in Production?</b>	

11c: What Does Confluent Advise for Deploying Schema Registry in Production?	323
11d: What Does Confluent Advise for Deploying the REST Proxy in Production?	329
11e: What Does Confluent Advise for Deploying Kafka Streams and ksqlDB in Production?	333
11f: What Does Confluent Advise for Deploying Control Center in Production?	337
<b>Conclusion</b>	<b>340</b>
<b>Appendix: Additional Content</b>	<b>347</b>
Appendix A: Detailed Transactions Demo	349
Appendix B: How Can You Monitor Replication?	364
Appendix C: Multi-Region Clusters	369
Appendix D: SSL and SASL Details	383

# Introduction



# CONFLUENT Global Education

# Class Logistics and Overview

## Copyright & Trademarks

Copyright © Confluent, Inc. 2014-2025. [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 modules:

- Bridging From Fundamentals
- Producing Messages Reliably
- Replicating Data: A Deeper Dive
- Providing Durability in Other Ways
- Configuring a Kafka Cluster
- Managing a Kafka Cluster
- Balancing Load with Consumer Groups and Partitions
- Optimizing Kafka's Performance
- Securing a Kafka Cluster
- Understanding Kafka Connect
- Deploying Kafka in Production

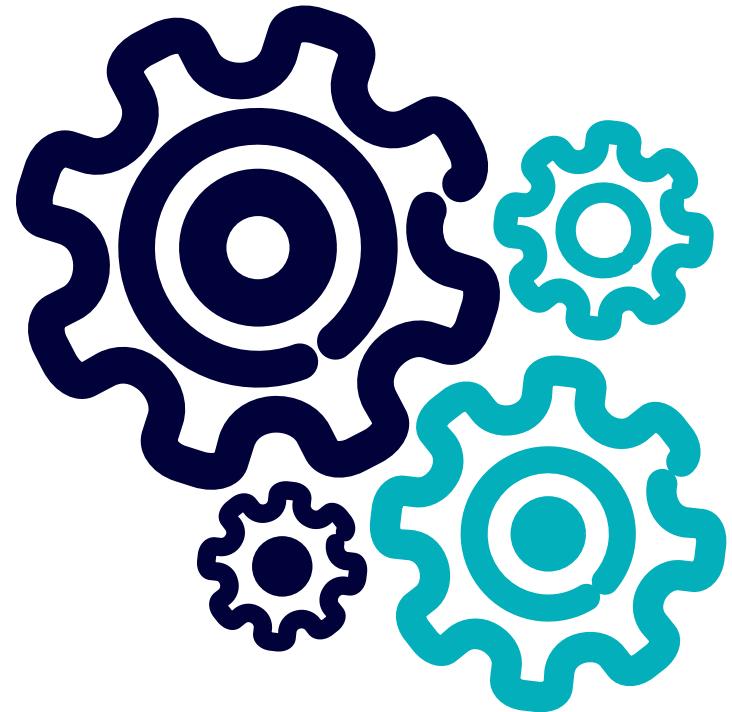
## Course Objectives

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

- Describe how Kafka brokers, producers, and consumers work
- Describe how replication works within the cluster
- Understand hardware and runtime configuration options
- Monitor and administer your Kafka cluster
- Integrate Kafka with external systems using Kafka Connect
- Design a Kafka cluster for high availability & fault tolerance

Throughout the course, Hands-On Exercises and Activities 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?
- (For Administration classes) What is your experience with Linux?  
With container technology?
- (For Developer classes) Which Computer Languages are you fluent with?

## About your instructor

# Fundamentals Review

## Discussion

### Question Set 1 [4 minutes]

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. In a topic, all messages that have the same key will be on the same broker.
4. The more partitions a topic has, the higher the performance that can be achieved.

### Question Set 2 [4 minutes]

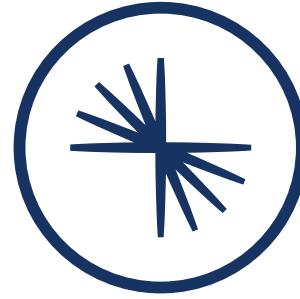
Determine the best answer to each question.

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?
4. How many times can a single message be consumed?

## Instructor-Led Review

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

# 1: Bridging From Fundamentals



CONFLUENT  
**Global Education**

# Module Overview



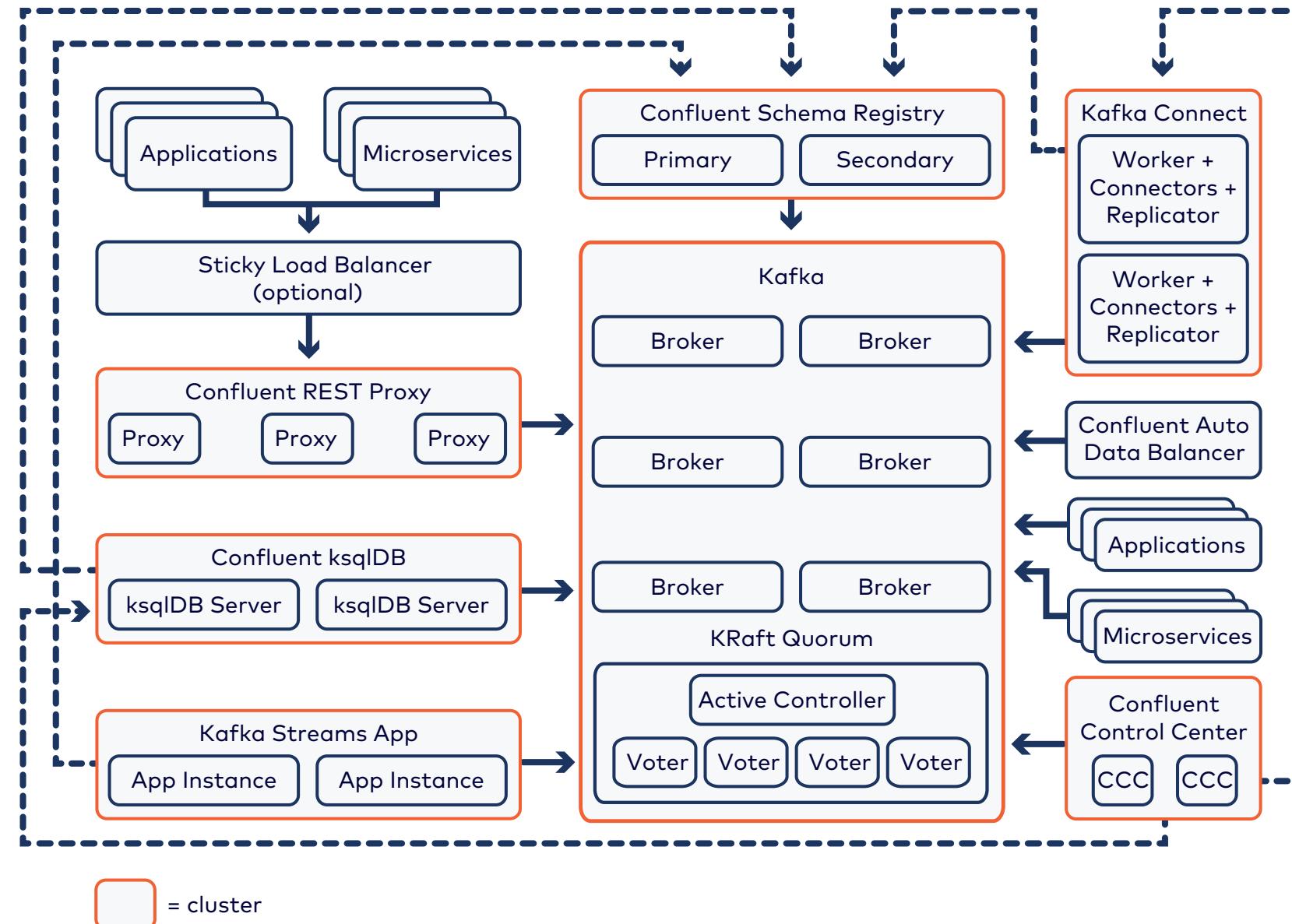
This module contains 1 lesson:

- How Can You Leverage Replication?

Where this fits in:

- Recommended Prerequisite: Fundamentals course

# Confluent's Deployment Architecture for Kafka



# 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 | Confluent for K8s | Ansible Playbooks

### MANAGEMENT & MONITORING

Control Center | Health+

### DEVELOPMENT & CONNECTIVITY

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

### APACHE KAFKA®

Core | Connect API | Streams API

Commercial Features

Community Features

Open Source Features

## 1a: 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.

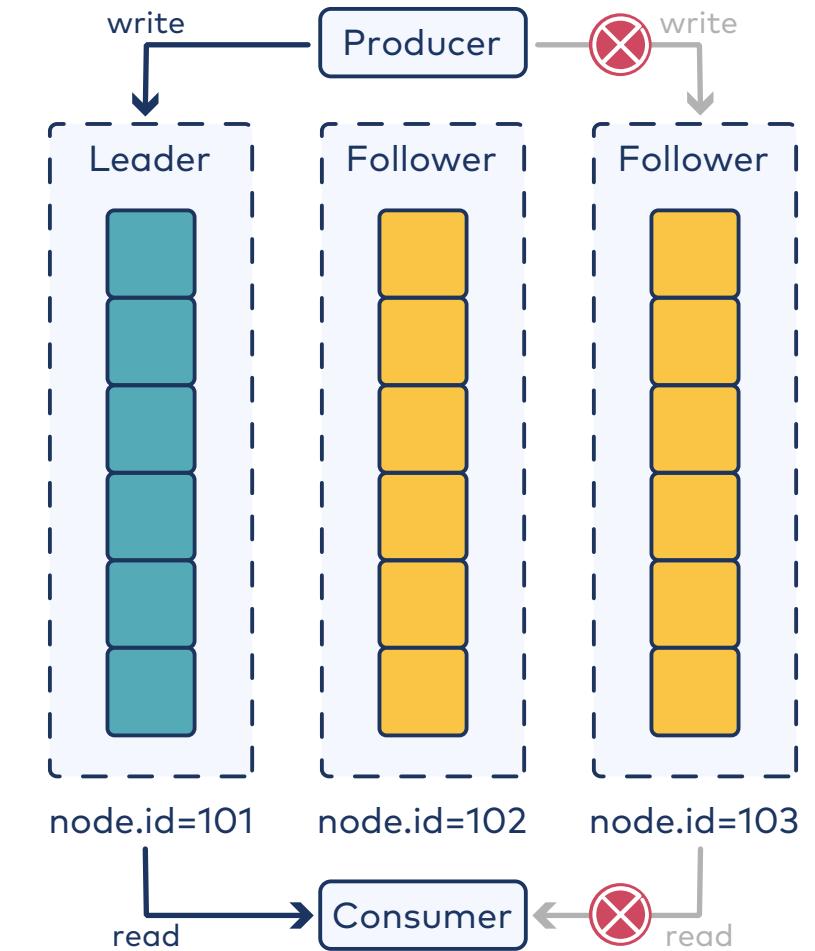
## Review: Basics of Replication

Ensure high availability of data with multiple **replicas** of partitions

**Leader** Always one per partition.  
Clients connect to it to write and read

**Follower** Generally multiple per partition.  
They keep extra copies from the leader

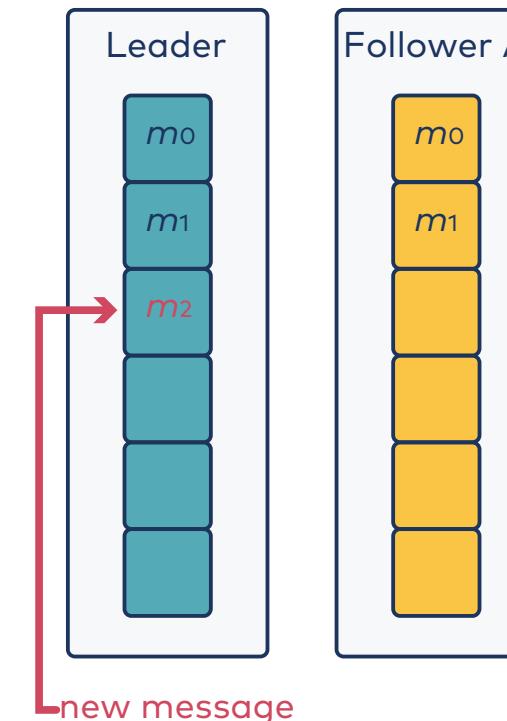
Topic setting `replication.factor`



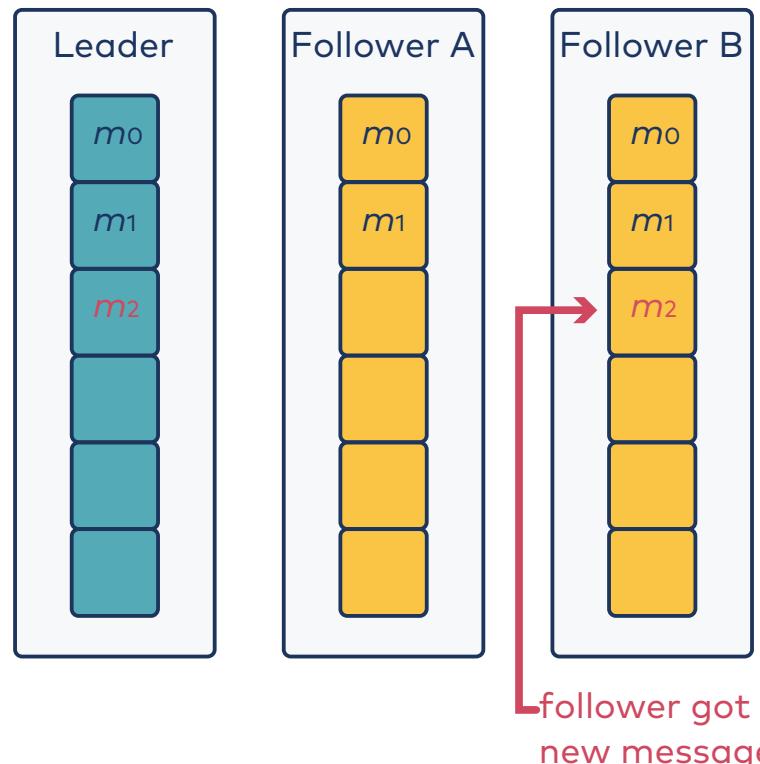
(**one** partition of a topic with  
`replication.factor=3`)

# "Follow the Leader"

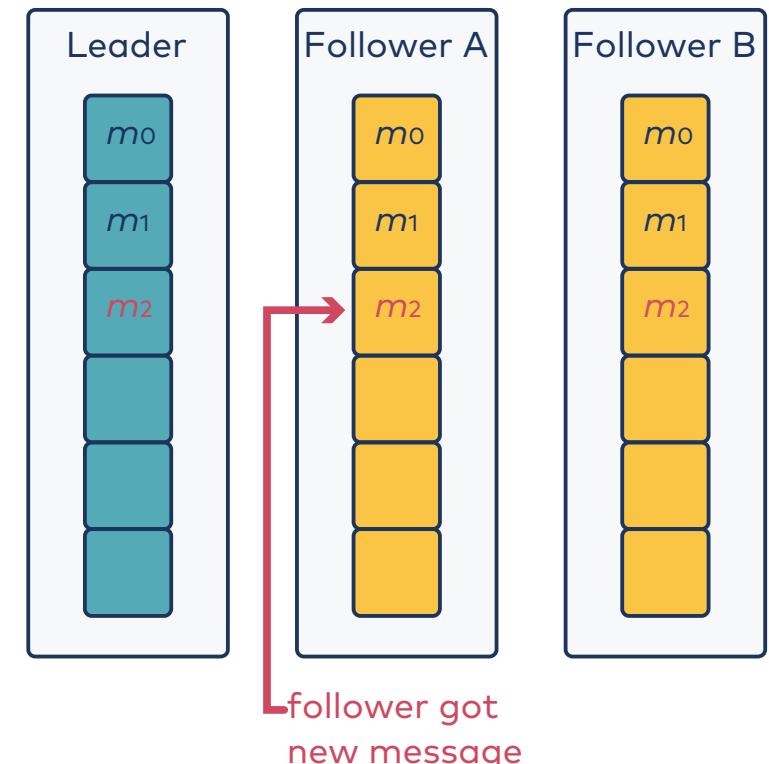
## Step 1



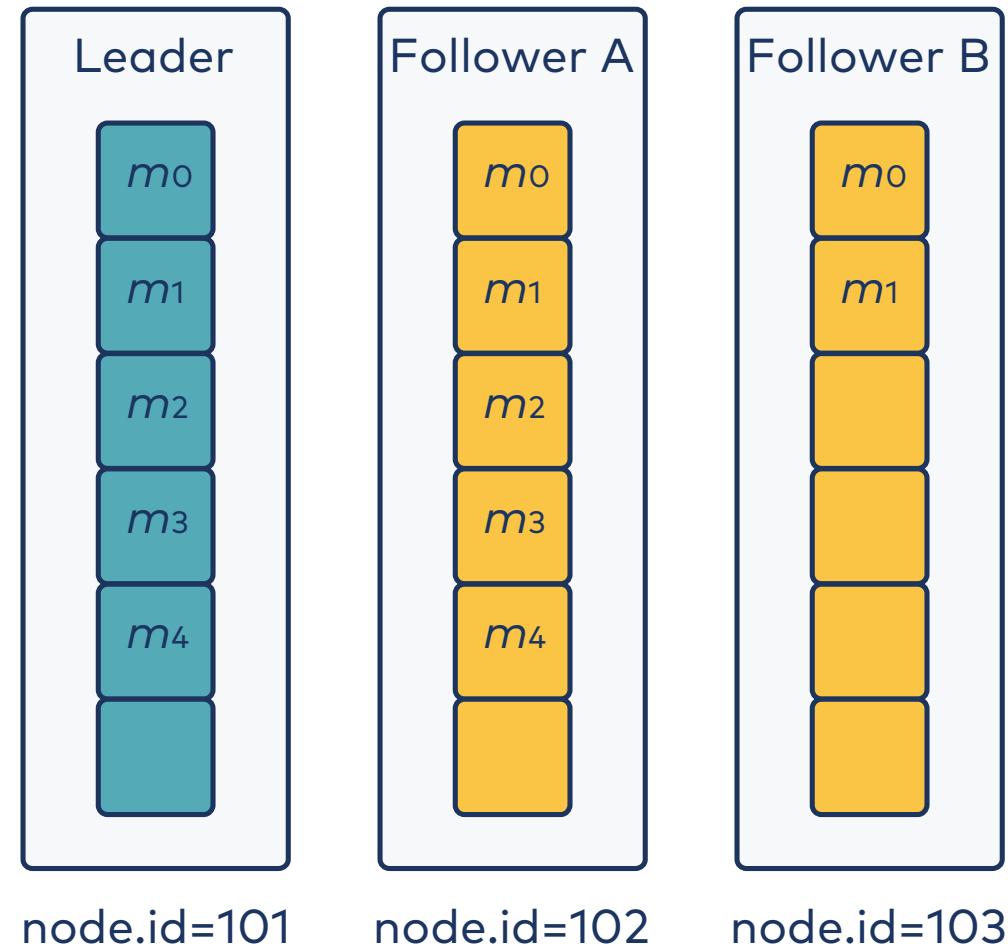
## Step 2



## Step 3

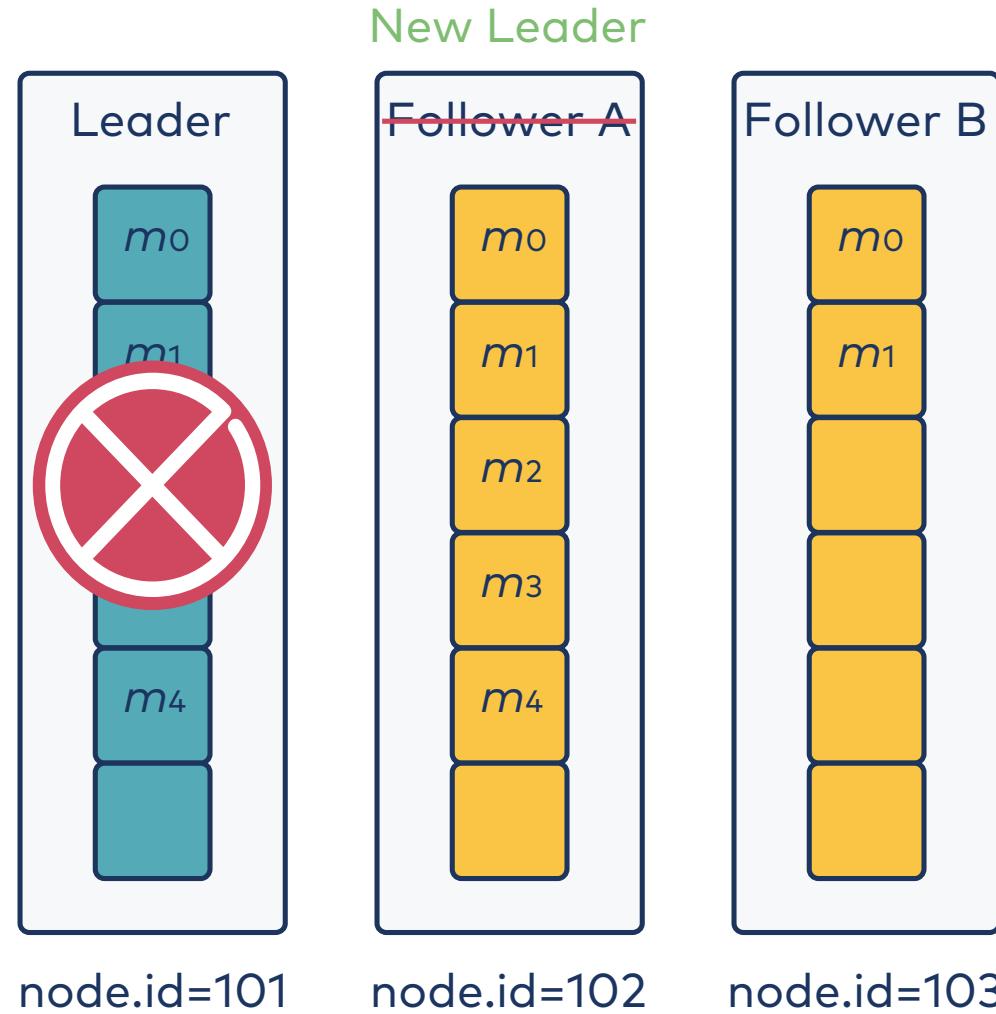


But Observe:



- Follower A (on node 102) is an **in-sync replica (ISR)**
- Follower B (on node 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

# Configuring Replication Factor

Increase the replication factor for better durability guarantees

- When creating a topic:

```
$ kafka-topics \
  --bootstrap-server kafka-1:9092 \
  --create
  --topic my_topic \
  --replication-factor 3 \
  --partitions 2
```

- Cluster-wide default value in `server.properties` (on all Kafka nodes)
  - `default.replication.factor` (Default: 1)

## Activity: Exploring Replica Placement & Replication Behavior



Say we have 5 brokers -  $b_1, b_2, \dots, b_5$ .

Say we have a replication factor of 4.

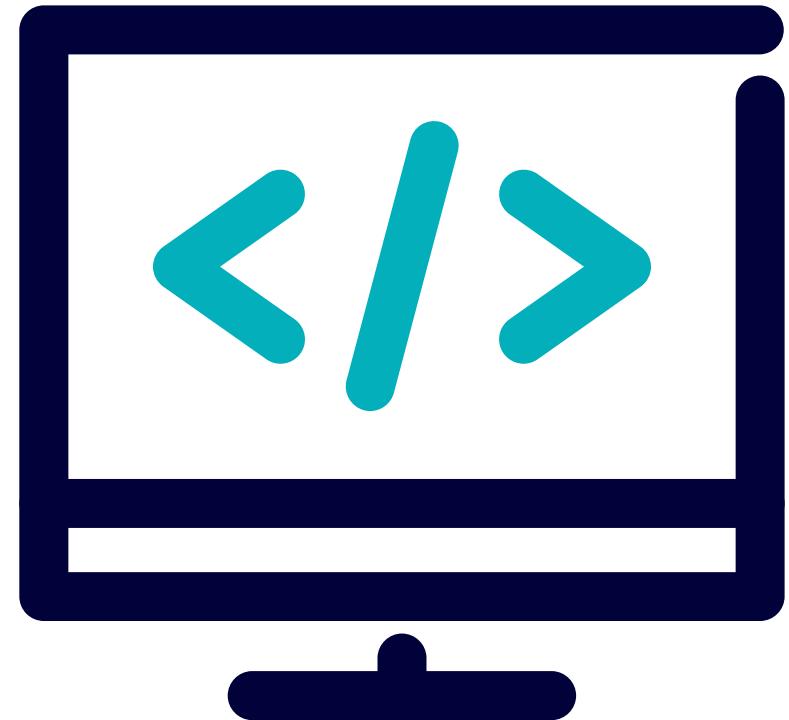
1. How many followers would we have?
2. Say leader is on broker  $b_5$ . Where could the followers be?
3. Say we have 3 successfully written messages that have been properly replicated. It's time to write the fourth message.
  - a. Where does it go?
  - b. What happens next?
4. Say broker  $b_5$  fails. What happens? Why?

# Labs: Bridging From Fundamentals

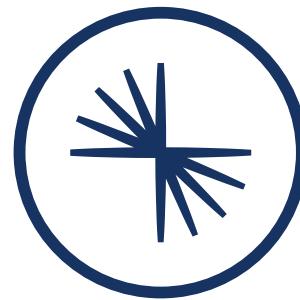
Please work on

- Lab 1a: Introduction
- Lab 1b: Using Kafka's Command Line Tools
- Lab 1c: Producing Records with a Null Key

Refer to the Exercise Guide



## 2: Replicating Data: A Deeper Dive



CONFLUENT  
**Global Education**

# Module Overview



This module contains 4 lessons:

- How Does Kafka Determine Which Messages Can be Consumed?
- How Does Kafka Place Replicas and How Can You Control Replication Further?
- How Does Kafka React When a Leader Dies?
- How Does Kafka Track Follower Responsiveness?

Where this fits in:

- Hard Prerequisite: Fundamentals course

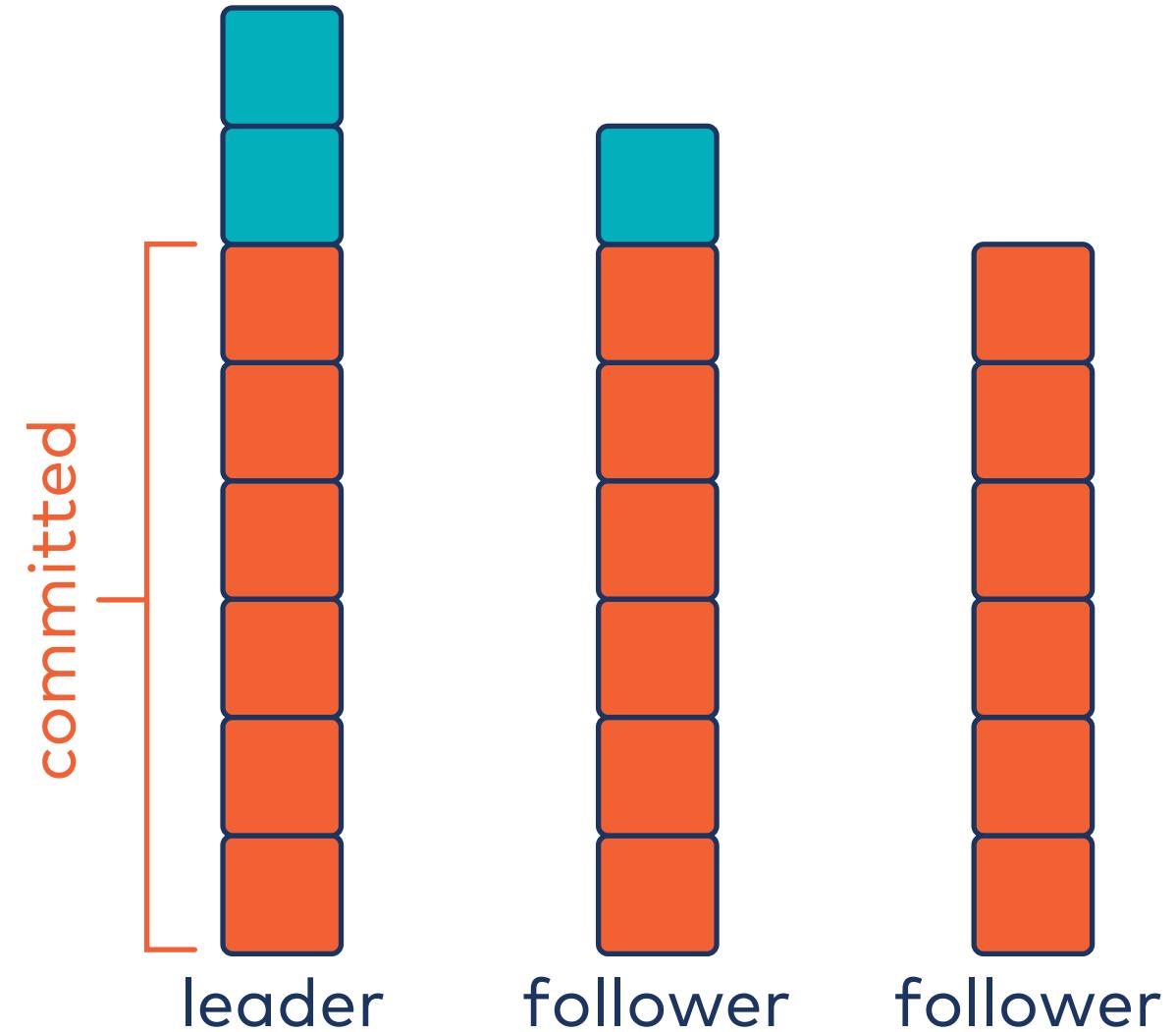
## 2a: How Does Kafka Determine Which Messages Can be Consumed?

### Description

High Water Mark. Committing messages.

## Which Messages can be Consumed?

- A message is called "committed" when it is replicated by all the replicas in the ISR list
- The leader keeps track of when a message is committed
- Consumers **can only** read committed messages
- The **High Water Mark** (HWM) points to the first non-committed offset
  - Consumers can consume up to (but not including) the HWM
  - The HWM is checkpointed to disk

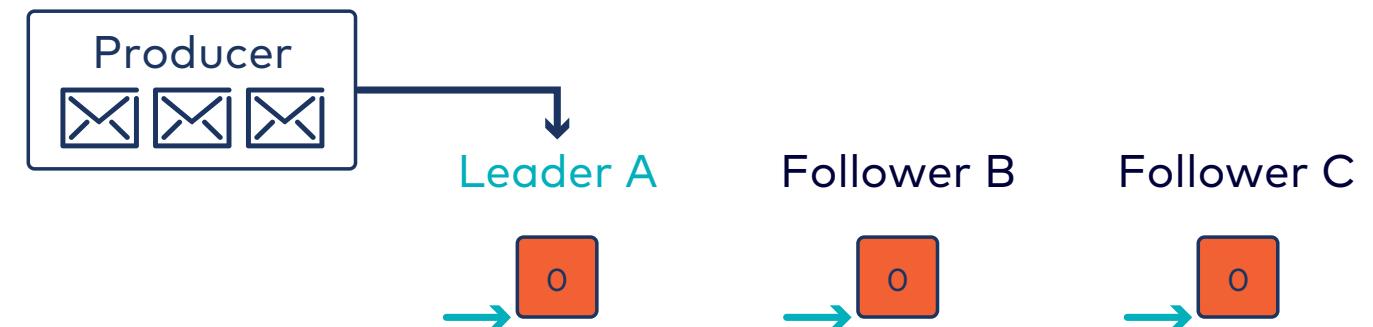


## Refining ISR Definition

- We said before that a follower that has all the messages the leader also has is considered an **in-sync replica**
- More accurately, a replica is an **in-sync replica** if it has all the messages the leader has *up to but not including* the high water mark. So ...
  - a leader may have messages that are not committed
  - a follower that does not have some or all not-committed messages may still be an in-sync replica

# Committing Messages with Replicas (1)

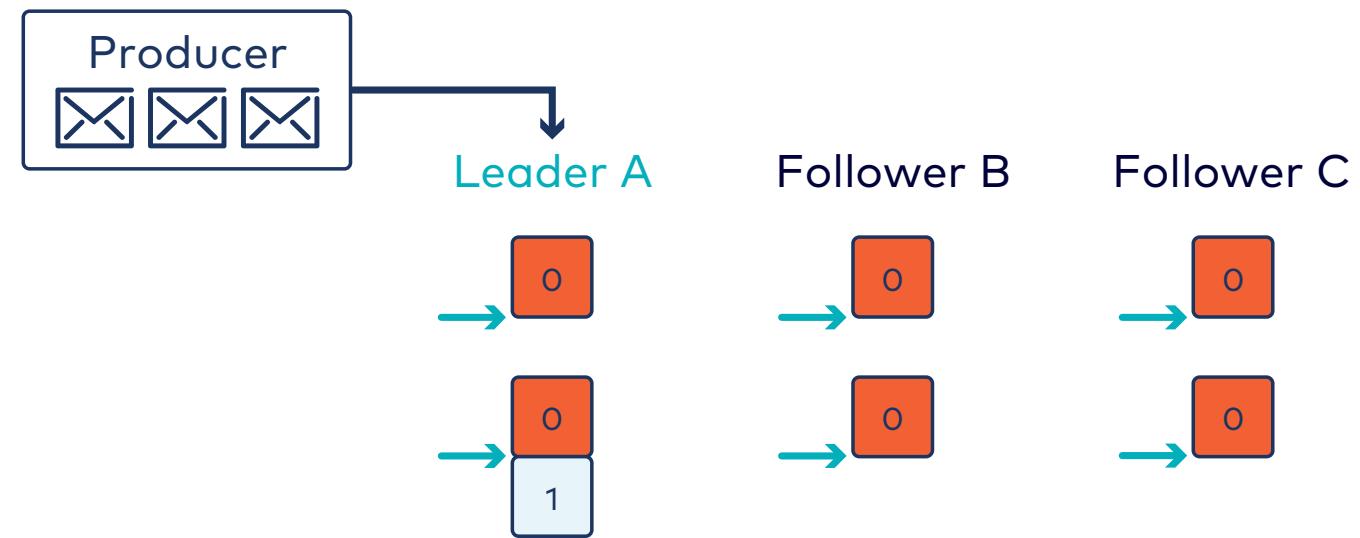
1 Initial state



## Committing Messages with Replicas (2)

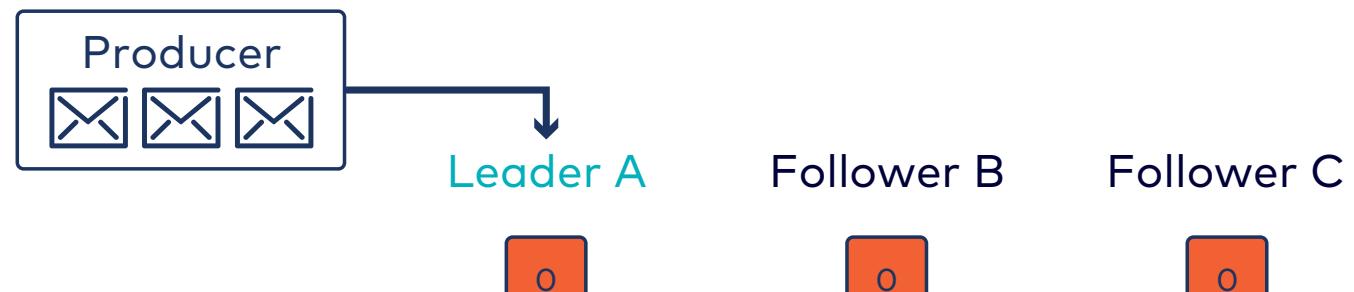
1 Initial state

2 A appends new message at offset 1

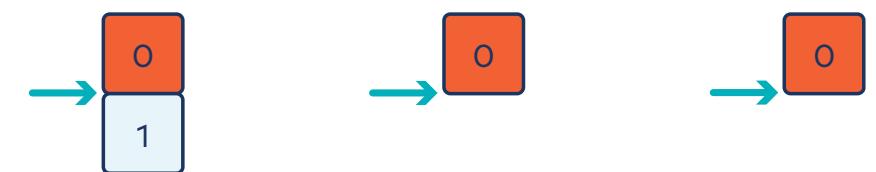


## Committing Messages with Replicas (3)

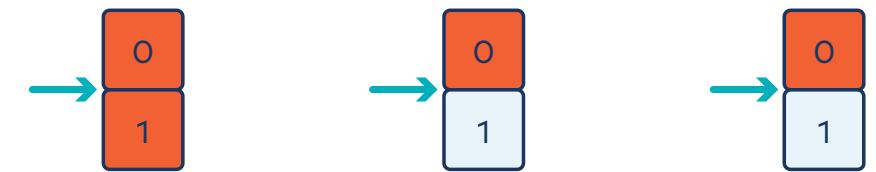
1 Initial state



2 A appends new message at offset 1



3 B and C fetch and append message at offset 1



## Committing Messages with Replicas (4)

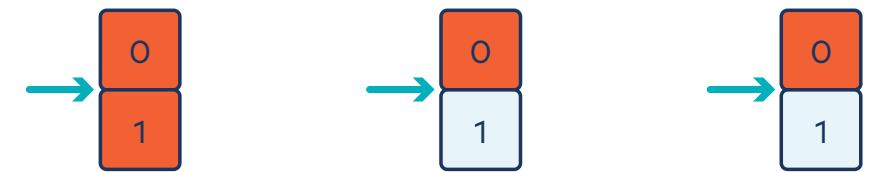
1 Initial state



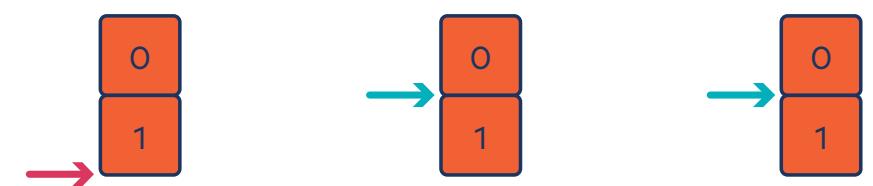
2 A appends new message at offset 1



3 B and C fetch and append message at offset 1



4 B and C fetch null at offset 2; A advances high water mark

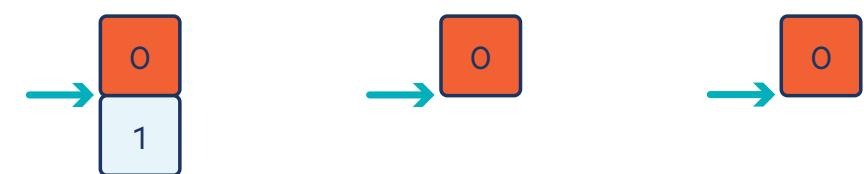


## Committing Messages with Replicas (5)

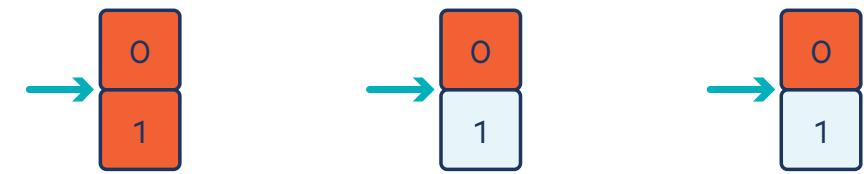
1 Initial state



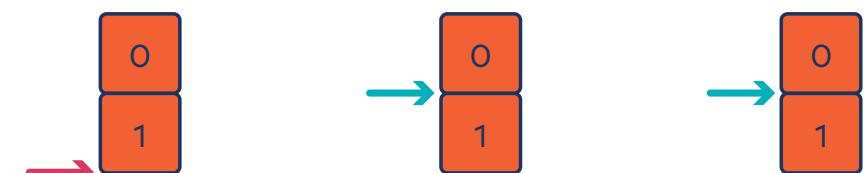
2 A appends new message at offset 1



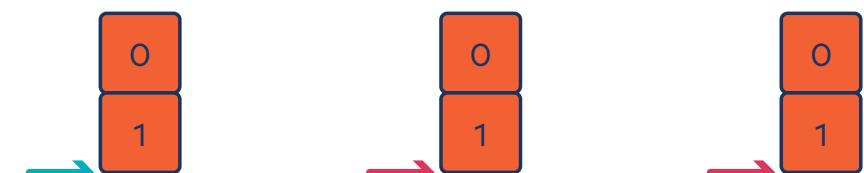
3 B and C fetch and append message at offset 1



4 B and C fetch null at offset 2; A advances high water mark



5 B and C fetch null at offset 2 again and receive new high water mark



## Activity: Assessing Consumption of a Recent Message



Discuss:

**Scenario:**

- Partition  $p$ 's leader got a message at time 7.
- Consumer  $c$ , configured correctly and assigned to partition  $p$  polls and gets an empty object back at time 8.

How is this possible? Use appropriate Kafka terminology.

## 2b: How Does Kafka Place Replicas and How Can You Control Replication Further?

### Description

Replica placement. Preferred replicas. Under-replicated and offline partitions.

## Replica Placement

### Kafka...

- places replicas on brokers
- tries to balance the placement of replicas across brokers

## Balancing Leadership and Preferred Replicas

Kafka attempts to balance brokers in terms of how many leaders are on one broker

For each partition...

- a broker is designated as the default leader - the *preferred replica*
- failover can select other replica as new leader
- Kafka monitors how many leaders are on their preferred replicas
  - When a threshold is exceeded, Kafka "fails back" leaderships to preferred replicas

## Viewing Partition Placement Across Cluster (1)

The same data tracked from the CLI with:

```
$ kafka-topics \
  --bootstrap-server kafka-1:9092 \
  --describe \
  --topic i-love-kafka
```

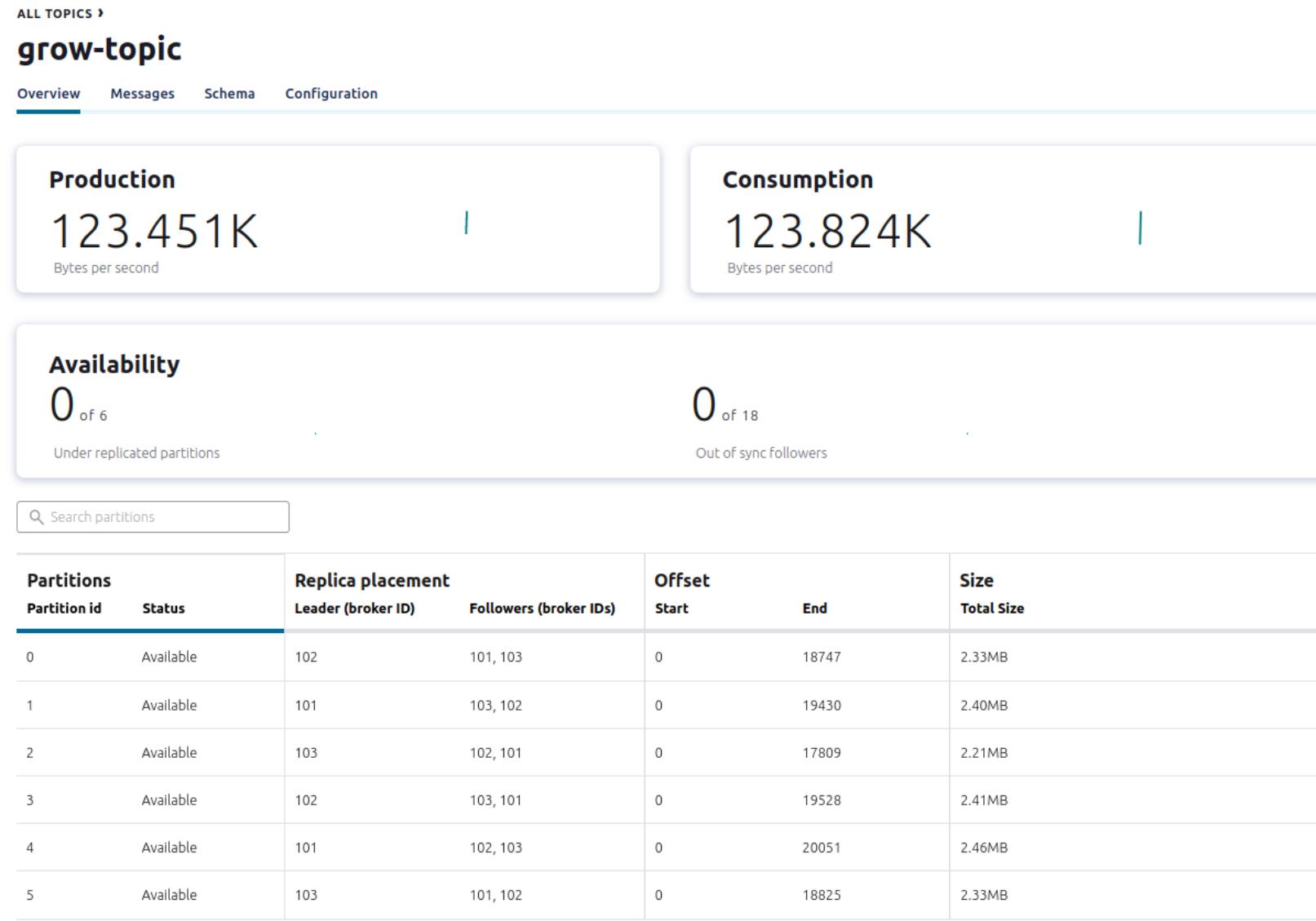
Topic:i-love-kafka PartitionCount:3	ReplicationFactor:3 Configs:
Topic: i-love-kafka Partition: 0	Leader: 101 Replicas: <b>101</b> ,102,103 Isr: 101,102,103
Topic: i-love-kafka Partition: 1	Leader: 103 Replicas: <b>103</b> ,101,102 Isr: 103,101,102
Topic: i-love-kafka Partition: 2	Leader: 102 Replicas: <b>102</b> ,103,101 Isr: 102,103,101



Preferred replicas highlighted in bold

## Viewing Partition Placement Across Cluster (2)

Confluent Control Center provides per-topic replica view:



## Two More Important Definitions

### **Under-Replicated Partition**

partition where the number of in-sync replicas < replication factor

### **Offline partition**

partition for which no leader exists



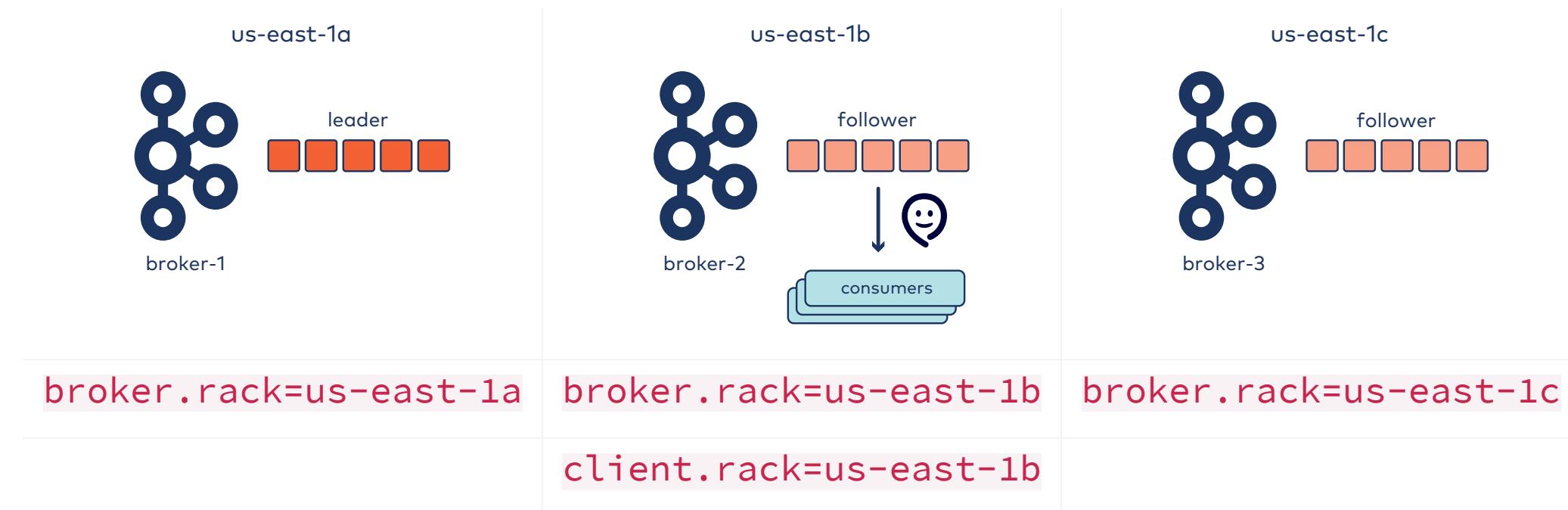
You can monitor both of these. More on that in the appendix.



# A Step Beyond

## Follower Fetching

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



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

**Question:** What is the tradeoff?

## 2c: How Does Kafka React When a Leader Dies?

### Description

Active Controller. Leader election. Unclean leader election.

## The Active Controller

- Kafka needs to maintain internal metadata (such as ISRs). This is stored in the system **metadata** topic
- The leader of the metadata topic is called the Controller Leader, **Active Controller** or just Controller

The Active Controller also:

- Monitors that brokers are alive
- Facilitates leader election
- Persists partition state to metadata topic
- Pushes leadership/ISR changes to involved brokers

## Unclean Leader

- Kafka defaults to choosing a follower from the ISR
- What if the ISR is empty (no in-sync follower), but you want the partition to have a leader?
  - Topic configuration property: `unclean.leader.election.enable`
  - Determines whether a new leader can be elected even if it is not in-sync, if there is no other choice
  - Can result in data loss if enabled (Default: `false`)

## 2d: How Does Kafka Track Follower Responsiveness?

### Description

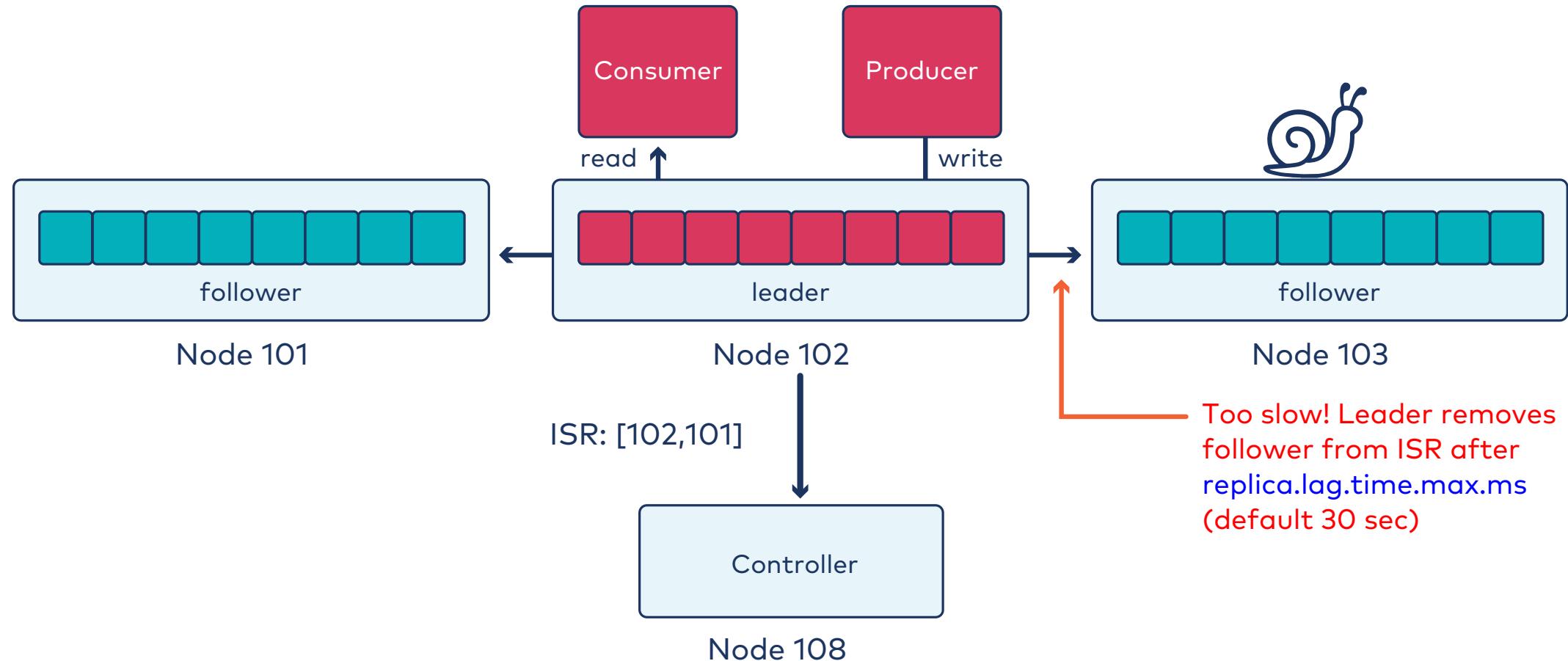
Replica Health. Slow Replicas. ISR Recovery

## Replica Health

Kafka includes mechanisms to assess the health of the nodes:

- Per-node check: broker sends heartbeats to Active Controller
  - If this check fails: Active Controller removes the broker from all ISRs and initiates elections for partitions that had this broker as leader
- Per-partition check: follower replicates in a short time
  - If this check fails: Leader asks Active Controller to remove the broker (only) from this partition's ISR

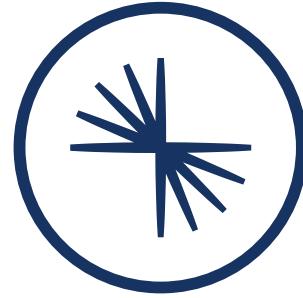
# Detecting Slow Replicas



## ISR Recovery

- Leader adds a replica back when both conditions are true:
  - Replica is up (sends heartbeats)
  - Replica is up to date (requests to fetch the HWM)

## 3: Producing Messages Reliably



CONFLUENT  
**Global Education**

# Module Overview



This module contains 3 lessons:

- How Do Producers Know Brokers Received Messages?
- How Can Kafka recognize Duplicates caused by Retries?
- How are Transactional Messages Tagged and Handled?

Where this fits in:

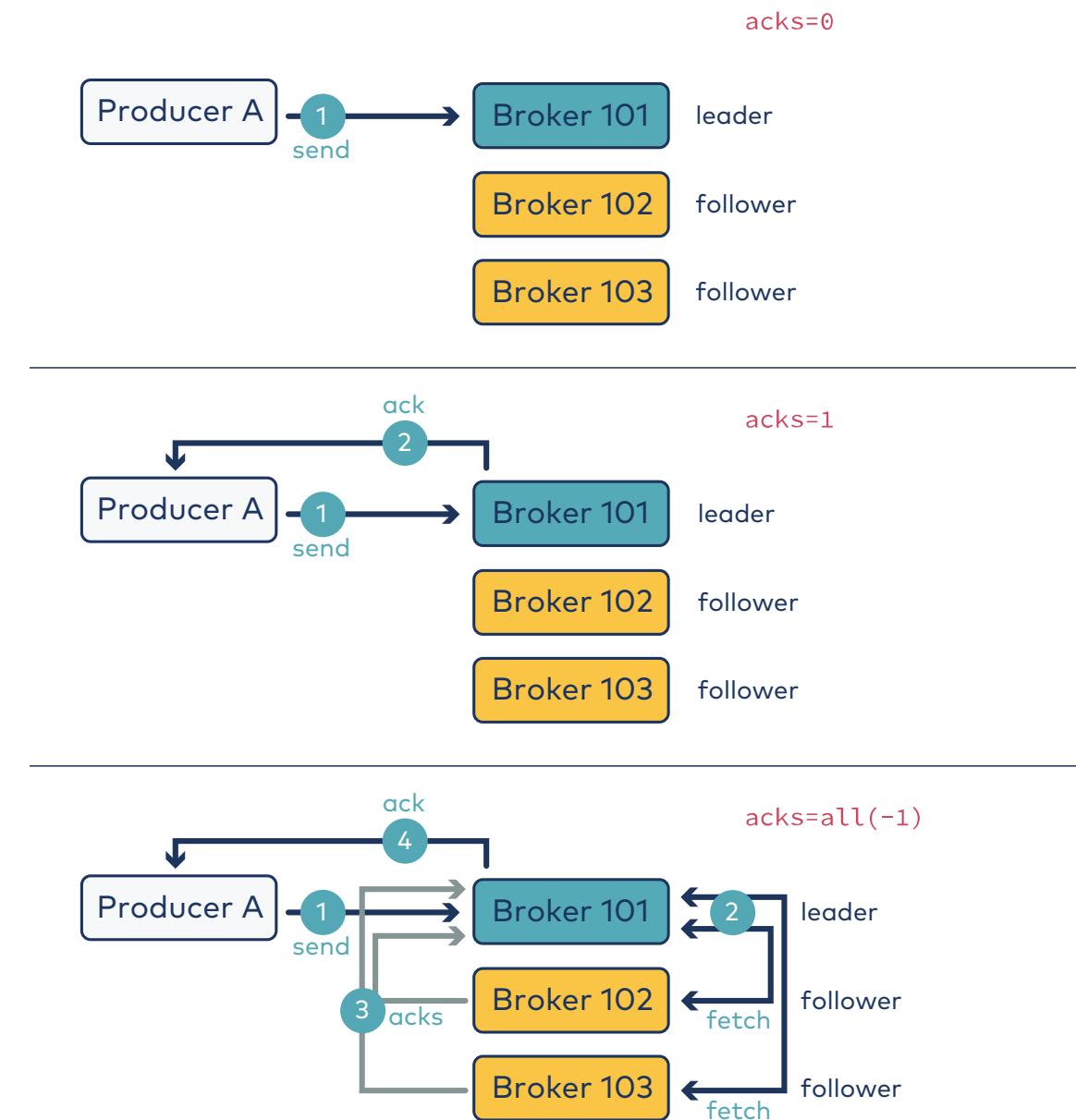
- Hard Prerequisite: Fundamentals course

## 3a: How Do Producers Know Brokers Received Messages?

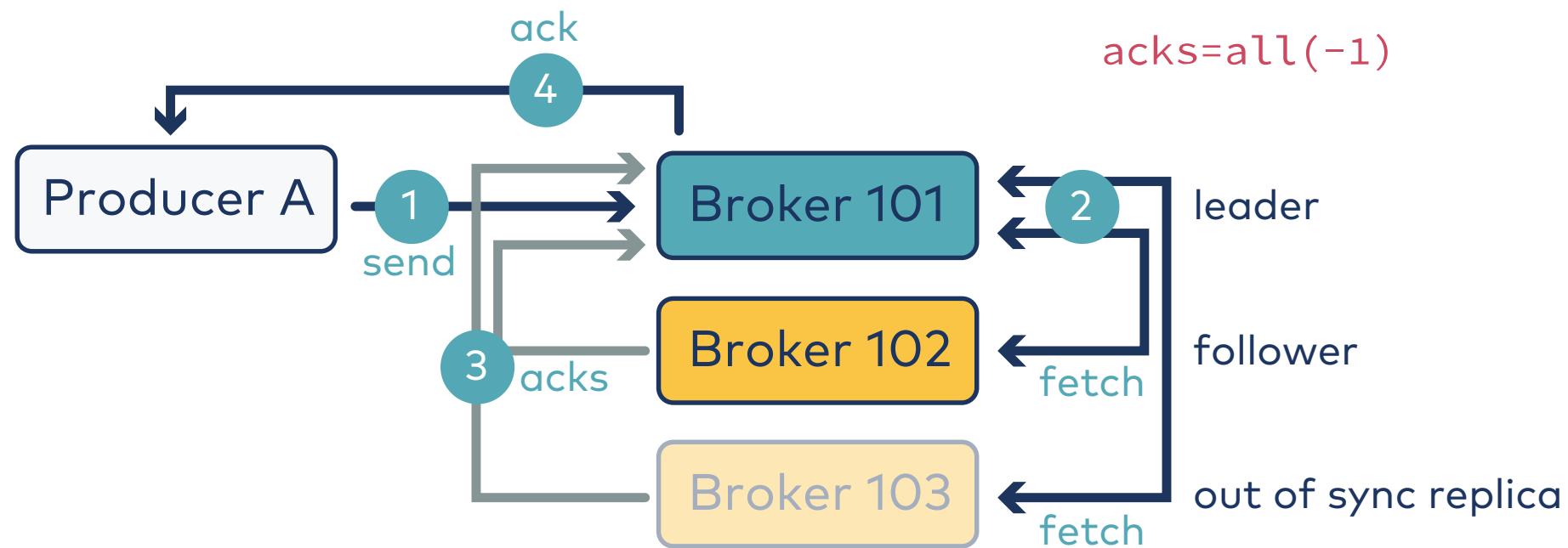
### Description

Producer acknowledgements.

## acks - Three Cases, Ideal Performance



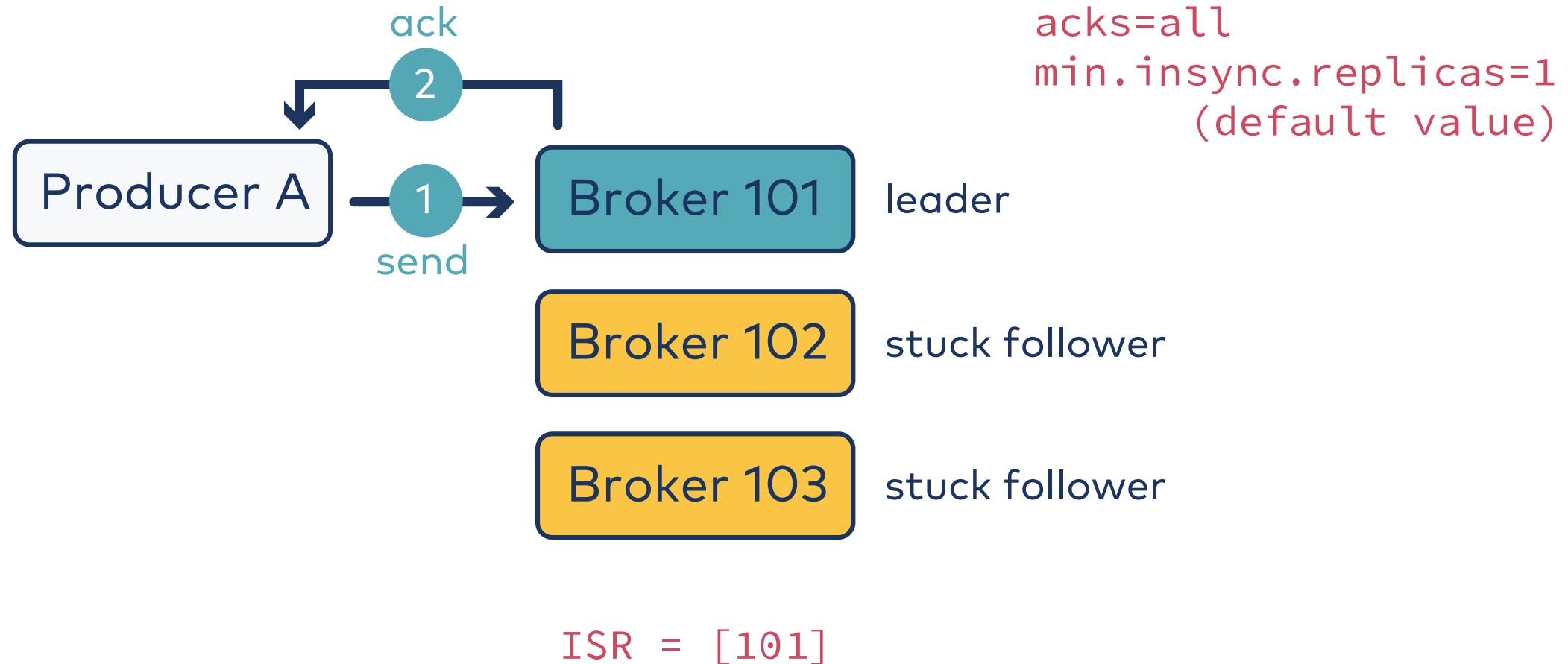
...But not all followers are in-sync...



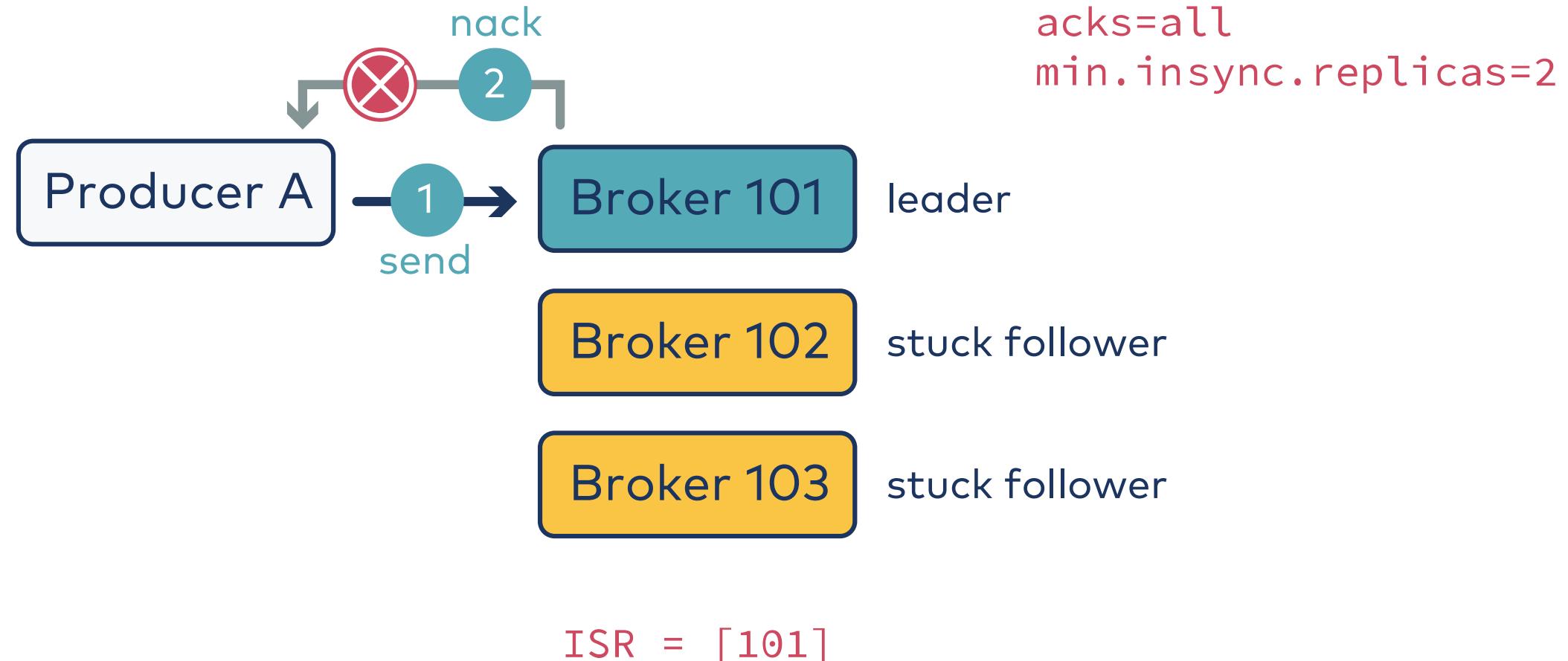
So... when the leader gets a new message and `acks=all`...

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

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



## Guaranteeing Meaningful `acks=all`



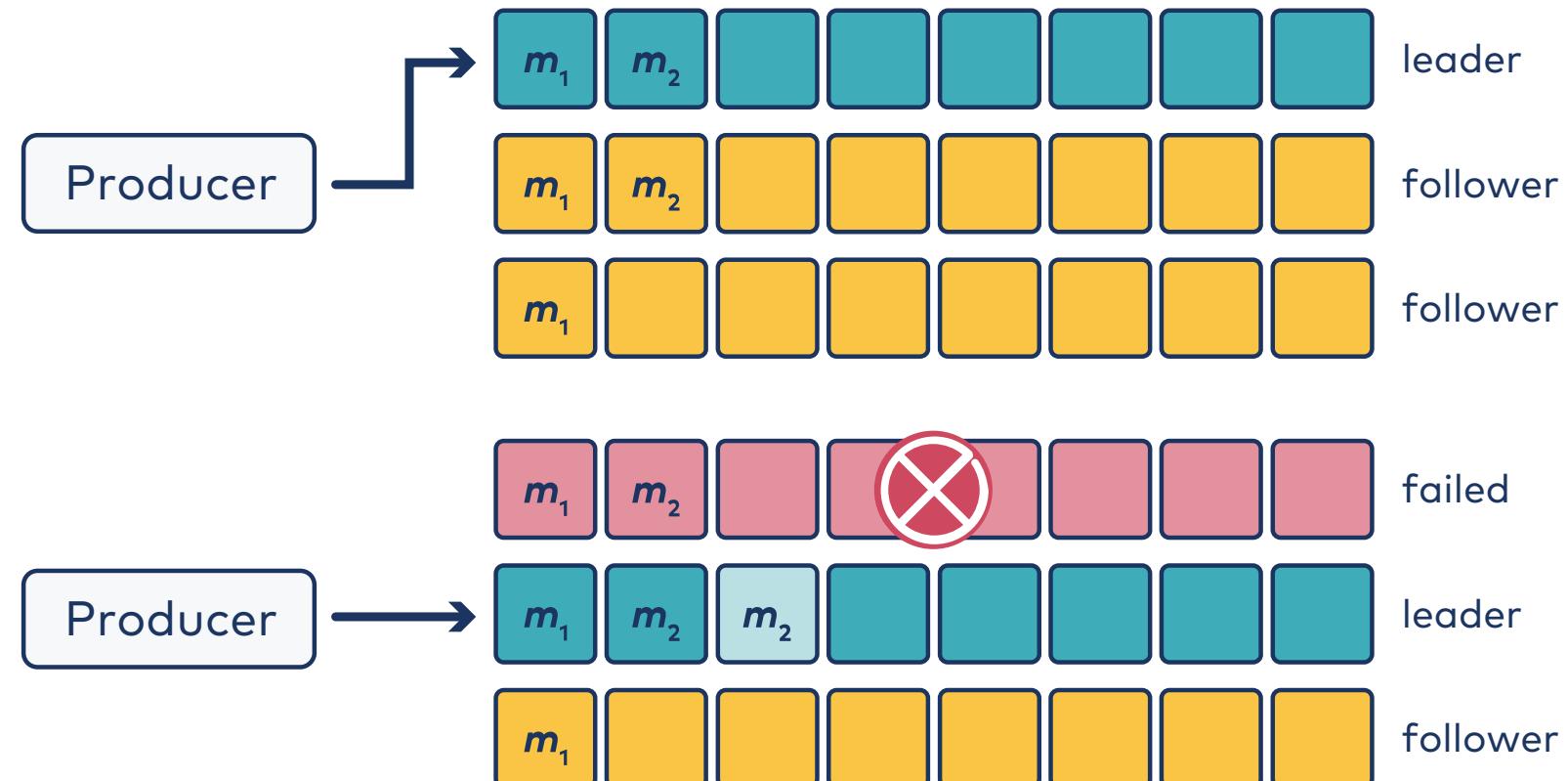
## 3b: How Can Kafka recognize Duplicates caused by Retries?

### Description

Idempotent producers: why, metadata, benefits.

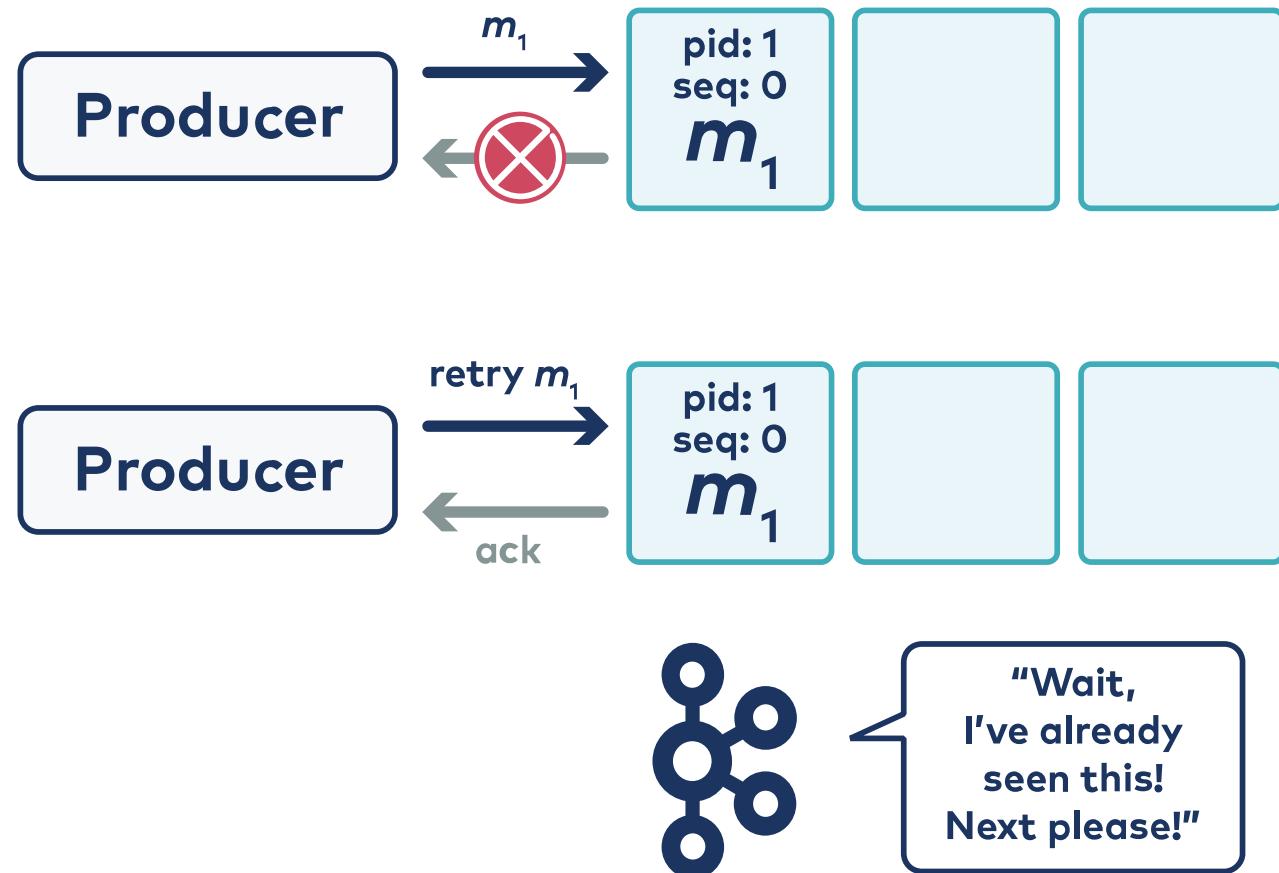
# Problem: Producing Duplicates to the Log

- `acks = all`
- `retries > 0` (default)



## Solution: Idempotent Producers

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



## 3c: How Does Kafka Handle the Notion of Producers Sending Messages in Transactions?

### Description

Transactions. How to enable on producers. Metadata. Effects. How brokers handle messages in transactions. How consumers handle transactional messages.

## Why Transactional Messages?

- Input message → Multiple related Output messages
  - Either all output messages are produced or none at all
  - E.g. money transfer order changes 2 bank account balances
- Kafka has a **Transactional API**

## Motivation for Exactly Once Semantics (EOS)

- Write real-time, mission-critical streaming applications that require guarantees that data is processed “exactly once”
- Exactly Once Semantics (**EOS**) depends on 2 mechanisms:
  - Idempotent producers: Prevents duplicate messages from being produced within a producer session (no restarts)
  - Consume-Process-Produce Design Pattern: developers can implement this pattern in code (needs Transactional API), preventing duplicates even between producer sessions (restarts)
- Sample use cases:
  - tracking ad views for billing
  - processing financial transactions
  - tracking inventory in the supply chain

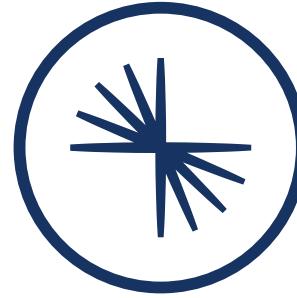
## Overview of Exactly Once Semantics

- Fully supported on all versions of the Java clients and librdkafka-based clients (v.1.4.0 and later)
  - Producer and consumer
  - Kafka Streams API
  - Confluent REST Proxy
  - Kafka Connect
- **Transaction Coordinator:**
  - Broker thread that manages a special **transaction log**

More...

See the Appendix in your Student Handbook for a detailed example of a transaction.

## 4: Storing the Records Persistently



CONFLUENT  
**Global Education**

# Module Overview



This module contains 3 lessons:

- How Does Kafka Organize Files to Store Partition Data?
- How Can You Decide How Kafka Keeps Messages?
- How to Scale Storage Beyond Kafka Servers?

Where this fits in:

- Hard Prerequisite: Fundamentals course
- Recommended Prerequisite: Replicating Data: A Deeper Dive

## 4a: How Does Kafka Organize Files to Store Partition Data?

### Description

Logs vs. segments. Details of files created per partition and per segment. Rolling of segments.

## Log File Subdirectories

- A Kafka partition is stored as a sequence of **log segments**
- Kafka log segment files are sometimes called **data files**.
- Each broker has one or more data directories specified in the `server.properties` file, e.g.,

```
log.dirs = /var/lib/kafka/data-a, /var/lib/kafka/data-b, /var/lib/kafka/data-c
```

- Each topic-partition has a separate subdirectory
  - e.g., `/var/lib/kafka/data-a/my_topic-0` for partition 0 of topic `my_topic`
- Brokers detect log directory failures and notify Controller

## Example of Log Files

Example of one broker's subdirectory for topic `my_topic` with partition `0`

```
$ ls /var/lib/kafka/data-b/my_topic-0
000000000000283423.index
000000000000283423.timeindex
000000000000283423.log
...
0000000000008296402.index
0000000000008296402.timeindex
0000000000008296402.log
leader-epoch-checkpoint
```

- Each `.log` filename is equal to the offset of the first message it contains, e.g.,
  - `000000000000283423.log`

## File Types Per Topic Partition

- Per log segment

`.log`

Log segment file holds the messages and metadata

`.index`

Index file that maps message offsets to their byte position in the log file

`.timeindex`

Time-based index file that maps message timestamps to their offset number

- Additional per log segment for *certain producers*:

`.snapshot`

If using idempotent producers, checkpoints PID and seq #

`.txnidex`

If using transactional producers, indexes aborted transactions

- Per partition

`leader-epoch-checkpoint`

Helper file when re-syncing with leader after a crash

## Log Segment Properties

- Messages are written to the **active** segment
- The active segment **rolls** to a new segment file if any are exceeded:
  - `log.segment.bytes` (Default: 1GB)
  - `log.roll.ms` (Default: 168 hours = 1 week)
  - `log.index.size.max.bytes` (Default: 10MB)
- A former active segment is called an **inactive** segment after it has rolled over

## Checkpoint Files

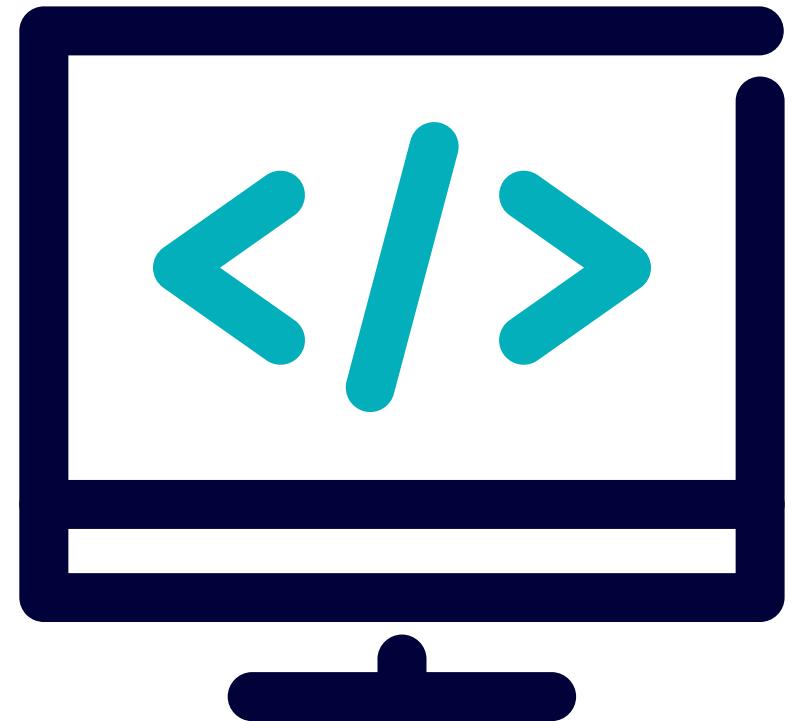
In addition, each broker has two checkpoint files:

- **replication-offset-checkpoint**
  - Contains the **high water mark** (the first offset after the committed messages)
  - On startup, followers use this to truncate any uncommitted messages
- **recovery-point-offset-checkpoint**
  - Contains the offset up to which data has been flushed to disk
  - During recovery, broker checks whether messages past this point have been lost

# Lab: Investigating the Distributed Log

Please work on **Lab 4a: Investigating the Distributed Log**

Refer to the Exercise Guide



## 4b: How Can You Decide How Kafka Keeps Messages?

### Description

Deletion. Compaction. Examples. Details of implementation of compaction. Monitoring and logging of compaction.

# Managing Log File Growth

- `cleanup.policy` at topic level
  - `delete` (default)
  - `compact`
  - `both`: `delete,compact`



Regardless of the policy, the active segment is kept intact as it is still open in read+append mode.

## The **Delete** Retention Policy

How log segments roll:

- `log.roll.ms`
- `log.segment.bytes`
- `log.retention.ms`

When segments are checked:

- `log.retention.check.interval.ms`
  - Default: 5 min

- Segments whose newest message is older than `log.retention.ms` will be deleted

What the log cleaner checks:

- `log.retention.ms`
- `log.retention.bytes`  
(disabled by default)

## Deleting All Messages in a Topic

1. Turn off all producers and consumers
2. Temporarily configure `retention.ms` to 0

```
$ kafka-configs \
  --bootstrap-server broker_host:9092 \
  --alter \
  --topic my_topic \
  --add-config retention.ms=0
```

3. Wait for cleanup thread to run  
(every 5 minutes by default)
4. Restore default `retention.ms` configuration

```
$ kafka-configs \
  --bootstrap-server broker_host:9092 \
  --alter \
  --topic my_topic \
  --delete-config retention.ms
```



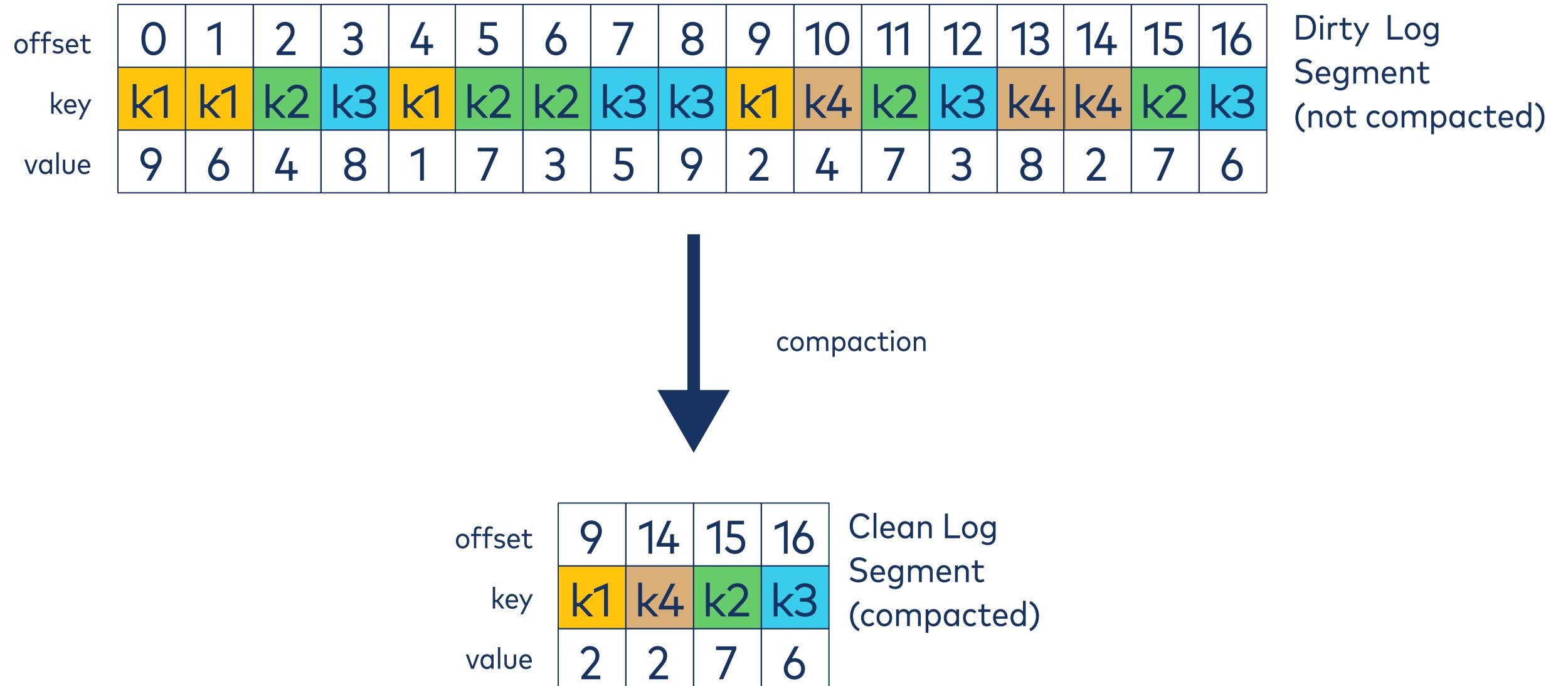
Do not just delete log files!

## Compact Policy Use Case

Keep only the **most recent** value for a given key

- Examples:
  - Database change capture
  - Real-time table look-ups during stream processing
  - Maintaining a topic of temperatures per postal code
  - Tracking the progress of e-commerce orders

# Log Compaction: What is it?



## Log Compaction: Important Configuration Values

- `log.cleaner.min.cleanable.ratio` (default 0.5)  
Triggers log clean if the ratio of `dirty/total` is larger than this value
- `log.cleaner.min.compaction.lag.ms` (default 0)  
The minimum time a message will remain uncompacted in the log
- `log.cleaner.max.compaction.lag.ms` (default infinite)  
The maximum time a message will remain ineligible for compaction
- `log.cleaner.io.max.bytes.per.second` (default infinite)  
throttle log cleaning

## Log Messages During Cleaning

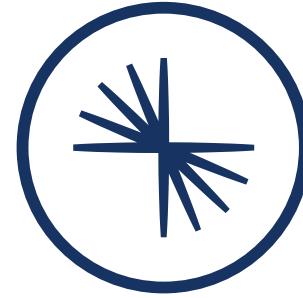
When the cleaning process is taking place, you will see log messages like this:

```
Beginning cleaning of log my_topic,0
```

```
Building offset map for my_topic,0 ...
```

```
Log cleaner thread 0 cleaned log my_topic,0 (dirty section=[100111,200011])
```

# 5: Configuring a Kafka Cluster



CONFLUENT  
**Global Education**

# Module Overview



This module contains 2 lessons:

- How Do You Configure Kafka Nodes?
- What if You Want to Adjust Settings Dynamically or Topic-wise?

Where this fits in:

- Hard Prerequisite: Fundamentals course
- Recommended Prerequisites: Other Ways Kafka Provides Durability

## 5a: How Do You Configure Brokers?

### Description

Static node settings. Where to set. Examples of classes of node settings.

## Where to Set Node Settings?

- Properties file: `/etc/kafka/server.properties`
  - On each node
- File is only read when node (re-)starts
- Includes configuration for:
  - Node itself
  - Cluster Defaults
  - Topics Defaults



It is possible to override these node settings. More on that in the next lesson.

## Some Node Properties

Some properties are clearly "this Node" things, e.g.,

- `node.id`
- `log.dirs` - where data directories for logs are
- `num.network.threads` - how many network threads
- `num.io.threads` - how many worker/IO threads

## Another "This Node" Config: Listeners

First off, recall, we use `host:port` pairs to describe connections, e.g., `broker1:9092`

Two node settings guide what ports are used and their security mechanism:

- `listeners` - host and port information from the local (server) endpoint
- `advertised.listeners` - host and port information from the remote (client) endpoint

You'd specify a listener something like `listeners = [protocol]://[host]:[port]`

So you might have settings like these (different examples):

- `listeners = PLAINTEXT://broker1:9092`
- `listeners = PLAINTEXT://broker-1.intranet:9092, SSL://broker1:9093`
- `advertised.listeners = SSL://broker1:9093`

## Cluster Defaults

Some properties you set in `server.properties` are meant as cluster defaults for all nodes.

Examples:

- `auto.create.topics.enable`



But `server.properties` is for **this one** node. If the cluster defaults are not consistent across the nodes in the cluster, you may experience unpredictable behavior.

## Some Properties Related to Topics

Some node properties apply to topics:

- `default.replication.factor`
- `num.partitions`
- log rolling threshold: `log.segment.bytes` and `log.roll.ms`



Some of these can be overridden at the topic level.

## What if I Don't Configure a Property?

Kafka defaults apply!

See documentation.

But, there's more. See next lesson...

## 5b: What if You Want to Adjust Settings Dynamically or Apply at the Topic Level?

### Description

Dynamic vs. static node settings. Cluster-wide vs. per-broker settings. Topic overrides. Order of precedence.

## There Aren't Only Static Node Settings...

Other options:

- Dynamic broker settings
- Dynamic cluster-wide settings
- Topic overrides

## Order of Precedence

1. Topic settings
2. Dynamic per-broker config
3. Dynamic cluster-wide default config
4. Static server config in `server.properties`
5. Kafka default

## Viewing Dynamic Broker Configurations

Display dynamic broker configurations for broker with ID 103:

```
$ kafka-configs \  
  --bootstrap-server kafka-1:9092 \  
  --broker 103 \  
  --describe
```



To display all config settings, not just dynamic changes, specify --all.

# Changing Broker Configurations Dynamically

- Change a cluster-wide default configuration

```
$ kafka-configs \
  --bootstrap-server kafka-1:9092 \
  --broker-defaults \
  --alter \
  --add-config log.cleaner.threads=2
```

- Change a broker configuration

```
$ kafka-configs \
  --bootstrap-server kafka-1:9092 \
  --broker 101 \
  --alter \
  --add-config log.cleaner.threads=2
```



To alter multiple config settings, use `--add-config-file new.properties`

## Deleting a Broker Config

### Delete a broker configuration

```
$ kafka-configs \
  --bootstrap-server kafka-1:9092 \
  --broker 107 \
  --alter \
  --delete-config min.insync.replicas
```

**Question:** What will the value of `min.insync.replicas` be now?



## Topic Overrides

Topic level configurations to override broker defaults



The names are often different, but similar.

Examples:

Meaning	Topic Override	Broker Config
Threshold log segment size for rolling active segment	<code>segment.bytes</code>	<code>log.segment.bytes</code>
Maximum size of a message	<code>max.message.bytes</code>	<code>message.max.bytes</code>

## Setting Topic Configurations from the CLI (1)

Set a topic configuration at time of topic creation

```
$ kafka-topics \  
  --bootstrap-server kafka-1:9092 \  
  --create \  
  --topic my_topic \  
  --partitions 1 \  
  --replication-factor 3 \  
  --config segment.bytes=1000000
```

## Setting Topic Configurations from the CLI (2)

- Change a topic configuration for an existing topic

```
$ kafka-configs \
  --bootstrap-server broker_host:9092 \
  --alter \
  --topic my_topic \
  --add-config segment.bytes=1000000
```

- Delete a topic configuration

```
$ kafka-configs \
  --bootstrap-server broker_host:9092 \
  --alter \
  --topic my_topic \
  --delete-config segment.bytes
```

## Viewing Topic Settings from the CLI

- Show the topic configuration settings

```
$ kafka-configs \  
  --bootstrap-server broker_host:9092 \  
  --describe \  
  --topic my_topic
```

Configs for topic 'my\_topic' are segment.bytes=1000000

- Show the partition, leader, replica, ISR information

```
$ kafka-topics \  
  --bootstrap-server broker_host:9092 \  
  --describe \  
  --topic my_topic
```

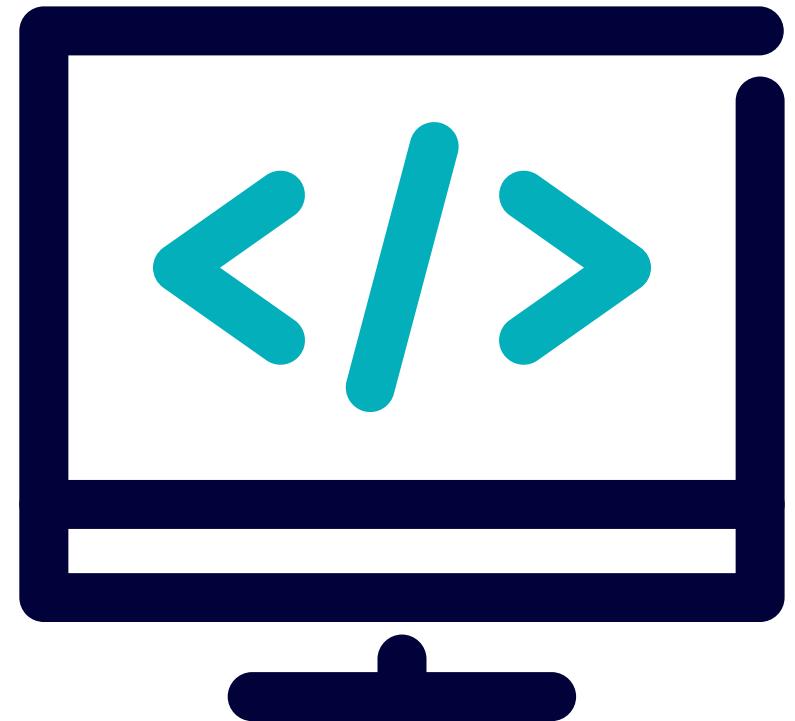
Topic:my\_topic PartitionCount:1 ReplicationFactor:3 Configs:segment.bytes=1000000  
Topic: my\_topic Partition: 0 Leader: 101 Replicas: 101,102,103 Isr: 101,102,103

# Labs: Configuring a Kafka Cluster

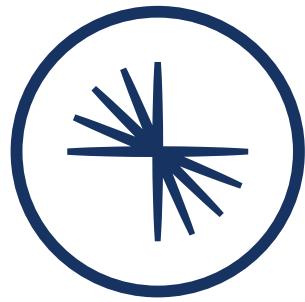
Please work on:

- Lab 5a: Exploring Configuration
- Lab 5b: Increasing Replication Factor
- (as per Instructor's Guidance): Lab 6a -  
**Prerequisite** section only

Refer to the Exercise Guide



# 6: Managing a Kafka Cluster



CONFLUENT  
**Global Education**

# Module Overview



This module contains 5 lessons:

- What Should You Consider When Installing and Upgrading Kafka?
- What is a Controller vs a Broker?
- What are the Basics of Monitoring Kafka?
- How Can You Move Partitions To New Brokers Easily?
- What Should You Consider When Shrinking a Cluster?

Where this fits in:

- Hard Prerequisite: Fundamentals course
- Recommended Prerequisite: Other Ways Kafka Provides Durability

## 6a: What Should You Consider When Installing and Upgrading Kafka?

### Description

Considerations for installation and upgrading.

## Methods of Installation

- You can deploy Confluent Platform from

...

- A Tar archive
- DEB or RPM package
- Docker container

- Deploy in a distributed environment using one of...

- Confluent for Kubernetes (formerly called Confluent Operator)
- Ansible playbooks



Schema Registry needs to have Kafka installed first

## Local vs. Distributed Installation

Local	Distributed
<ul style="list-style-type: none"><li>• When installed on a single machine.</li><li>• Installed via zip, tar, or Docker images</li></ul>	<ul style="list-style-type: none"><li>• When services are installed across several machines</li><li>• Installed via Kubernetes or Ansible</li></ul>

## Upgrading a Cluster (1)

There are many things to consider when upgrading the Confluent Platform:

- The order of upgrade for Kafka nodes (controllers and brokers) and the rest of the Confluent Platform is very important.
- Always read the upgrade documentation on our website to get the full scope of what the upgrades entail.

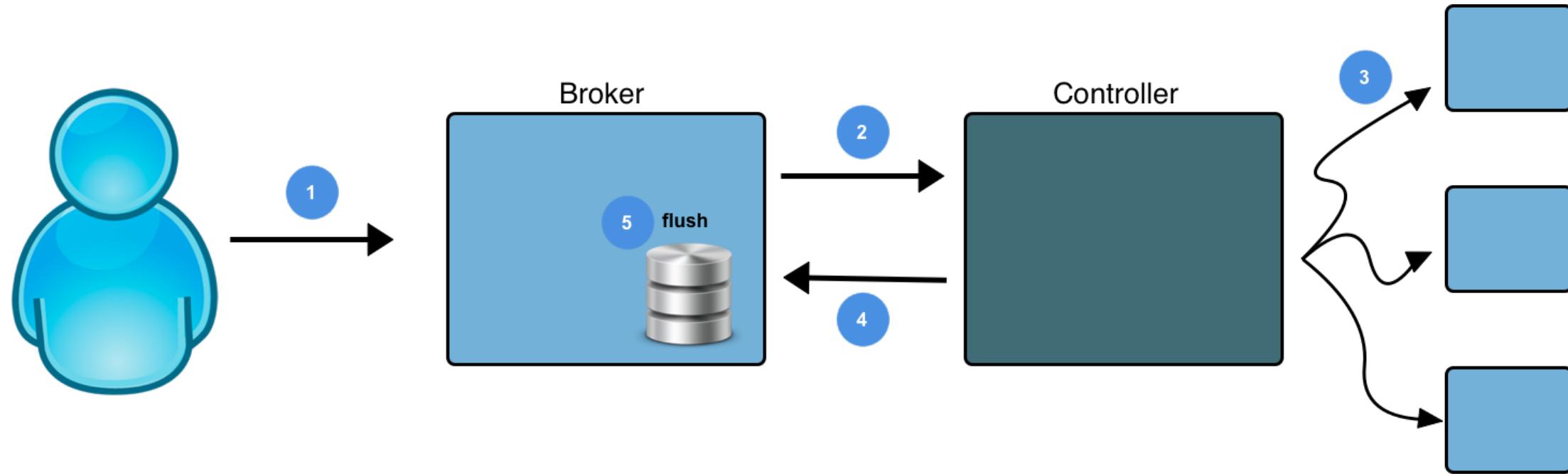
## Upgrading a Cluster (2)

- Be aware of the broker protocol version and the log message format version between the various releases.
- Apply a license key for the Confluent Platform.
  - Know how to apply the key.
- Do rolling restart—the entire cluster stays up as you take down each broker one by one to upgrade.



See link in your guide!

## Controlled Shutdown



1. Administrator sends a `SIGTERM` to the Java process running Kafka, e.g., `kafka-server-stop`
2. The node sends request to the Controller.
3. The Controller facilitates leader elections
4. The Controller responds to the node that it can now leave.
5. The node flushes file system caches to disk.
6. The node shuts down.

## 6b: What is a Controller vs a Broker?

### Description

KRaft. Kafka Roles. Controller Election. Metadata Topic.

# Cluster Metadata and the Controller

## Cluster Metadata

- Stored in `--cluster_metadata` topic
- Active Controller is its Leader
- Contains:
  - Topic and Dynamic Configuration
  - Partition Leader, ISR
  - Quotas
  - ACLs
  - ...

## Active Controller

- Monitors Nodes' Health
- Controller Chooses Leaders
- High-Availability Role

Can you spot the problem?

## KRaft

- Implementation of a Distributed Consensus Algorithm within Kafka
- Voting system with quorum (3 or 5 nodes)
- Quorum voters elect the leader of `__cluster_metadata`

## Roles for Kafka Nodes

A Kafka cluster comprises a number of voter nodes plus some broker nodes.

**Quorum voter** `process.roles=controller` (for `__cluster_metadata` topic)

**Broker** `process.roles=broker` (for all the other topics)

All nodes must know how to locate the quorum voters:

`controller.quorum.voters=nodeid1@host1:port1,nodeid2@host2:port2,nodeid3@host3:port3`

## 6c: What are the Basics of Monitoring Kafka?

### Description

Basics of metrics and monitoring tools. Kafka logs vs. system logs.

# Monitoring Your Kafka Deployments

- You can use Confluent Control Center to
  - Optimize performance
  - Verify broker and topic configurations
  - Identify potential problems before they happen
  - Troubleshoot issues
- What else to monitor
  - Kafka logs
  - Kafka metrics ([JMX](#))
  - System logs
  - System resource utilization

## Logs vs. Logs

- Kafka topic data
  - `log.dirs` property in `server.properties`
- Application logging with Apache `log4j`
  - `LOG_DIR`: configure the `log4j` files directory by exporting the environment variable  
(Default: `/var/log/kafka`)

## Important log4j Files

- By default, the broker log4j log files are written to `/var/log/kafka`
  - `server.log`: broker configuration properties and transactions
  - `controller.log`: all broker failures and actions taken because of them
  - `state-change.log`: every decision broker has received from the controller
  - `log-cleaner.log`: compaction activities
  - `kafka-authorizer.log`: requests being authorized
  - `kafka-request.log`: fetch requests from clients
- Manage logging via `/etc/kafka/log4j.properties`

## Tools for Collecting Metrics

- **Confluent Control Center**
- Other metrics tools:
  - JConsole
  - Graphite
  - Grafana
  - CloudWatch
  - DataDog
- Starting with AK 3.7 (KIP-714), Kafka Metrics API can help centralizing clients' metrics

## Kinds of JMX Metrics

There are two kinds of JMX metrics:

- **gauge** - a measure of something *right now*
  - e.g., number of offline partitions
- **meter** - a measure of event occurrence over a time sample
  - e.g., count, weight, throughput

## Configuring the Cluster for Monitoring

- Enable JMX metrics by setting `JMX_PORT` environment variable

```
$ export JMX_PORT=9990
```

- Configure `client.id` on producers and consumers
  - Monitor by application
  - Used in logs and JMX metrics

## Troubleshooting Issues

- Check metrics
- Parse the `log4j` logs
- Avoid unnecessary restarts

## 6d: How Can You Move Partitions To New Brokers Easily?

### Description

Basics of Self-Balancing Cluster mechanism.

## Self-Balancing Cluster Overview

- Applies when:
  - You've added new (empty) brokers and want to move partitions to them
  - You want to remove brokers from the cluster and want to move partitions from them beforehand
  - Disk usage among brokers is not balanced (optional)
- What it does:
  - Calculates an optimal placement of partitions among brokers
  - Automatically moves partitions
- SBC is part of Confluent Platform Enterprise (paid feature)

## Basic configuration of Self-Balancing

Settings in `server.properties` in all brokers and controllers:

- `confluent.balancer.enable` (default `false`)
- `confluent.balancer.heal.uneven.load.trigger`
  - `EMPTY_BROKER` (default): rebalance after adding a broker
  - `ANY_UNEVEN_LOAD`: rebalance **also** on uneven workload

These two settings are **dynamic** and can be changed using `kafka-configs` or in Confluent Control Center



After enabling Self-Balancing, the system needs to gather metrics for around **30 minutes**: no balancing will happen during that initial period.

# Self-Balancing Configuration in Control Center

You can find this page in Cluster settings → Self-balancing:

The screenshot shows the Confluent Control Center interface. The top navigation bar includes the Confluent logo, a trial message ("Enterprise trial ends in 29 days"), and a three-dot menu icon. The left sidebar lists cluster management sections: Cluster overview, Brokers, Topics, Connect, ksqlDB, Consumers, Replicators, and cluster settings (which is selected and highlighted in blue). Below the sidebar, the main content area has a title "Cluster settings". A navigation bar at the top of the content area includes tabs: General, Cluster defaults, Self-balancing (which is active and underlined), and Tiered storage. The "Self-balancing" section contains a "Status" field with two buttons: "On" (which is highlighted in blue) and "Off". To the right of the status is a checkbox labeled "Show raw configs". Below the status is a "Throttle (MB/s)" input field set to "10", with a small info icon next to it. Underneath the throttle setting is a section titled "Automatically improve cluster balance" with two radio button options: "Anytime" and "Only when brokers are added" (which is selected). The "Only when brokers are added" option is highlighted with a blue circle. Below this is a "Static settings" section with four entries: broker failure detection (1 hour), disk space available for partitions (85%), topics excluded by prefixes (--), and topics excluded by names (--). At the bottom of the section are "Save" and "Cancel" buttons.

## How to trigger Self-Balancing

Self-balancing starts a task when:

- A broker is added to the cluster (automatic)
- A broker is being prepared for removal from cluster (manually from CLI or CCC)
- A broker is considered permanently failed (automatic after threshold)
- Workload on brokers is uneven (automatic, only if `ANY_UNEVEN_LOAD`)
- Manually (only from CLI)

## 6e: What Should You Consider When Shrinking a Cluster?

### Description

Tips for shrinking a cluster. Replacing a server.

## Shrinking the Cluster

- Why reduce the number of brokers in the cluster?
  - Maintenance on a broker
  - Reduce cost during periods of low cluster utilization
- Decommissioning a broker
  - If using Self-Balancing Cluster, issue the broker removal from CLI or from Control Center
  - If using `kafka-reassign-partitions`, reassign **all** replicas that is currently hosting (leaders and followers) to other brokers, then perform a controlled shutdown

## Replacing a Failed Node

- On new hardware/container, deploy a new Kafka Server with the same value for `node.id`
- The new Server will automatically bootstrap data on start-up
- If possible, start up the replacement Server at an off-peak time



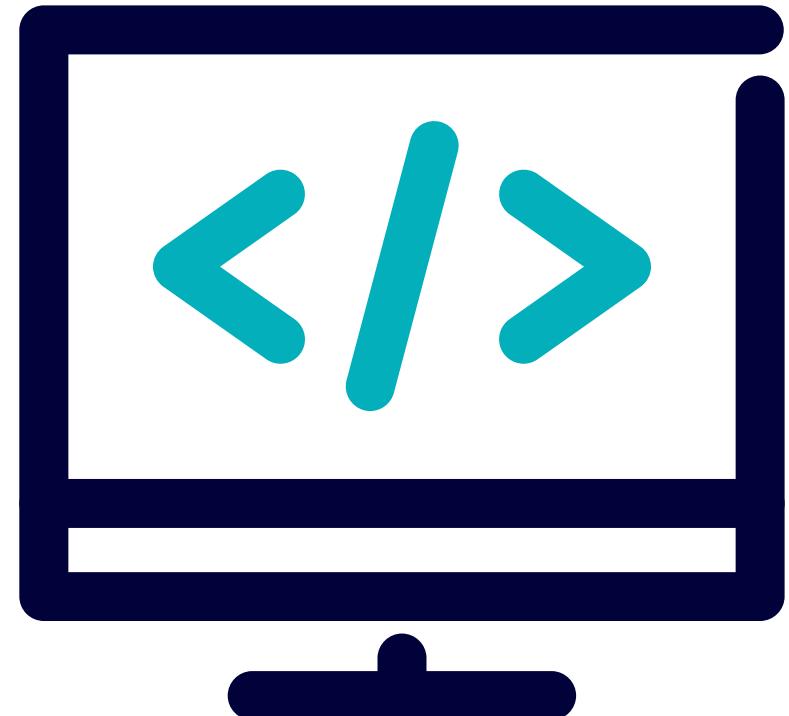
Server will copy the data as fast as it can during recovery. This can have a significant impact on the network.

# Lab: Kafka Administrative Tools

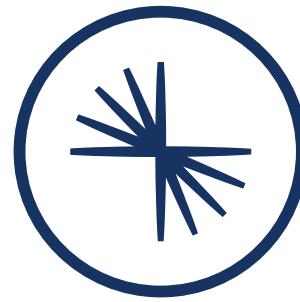
Please work on:

- Lab 6a: Rebalancing the Cluster
- Lab 6b: Simulate a Completely Failed Broker
- Lab 6c: Deleting Topics in the Cluster

Refer to the Exercise Guide



# 7: Balancing Load with Consumer Groups and Partitions



CONFLUENT  
**Global Education**

# Module Overview



This module contains 5 lessons:

- What are the Basics of Scaling Consumption?
- How Do Groups Distribute Work Across Partitions?
- How Does Kafka Manage Groups?
- How Do Partitions and Consumers Scale?
- How Does Kafka Maintain Consumer Offsets?

Where this fits in:

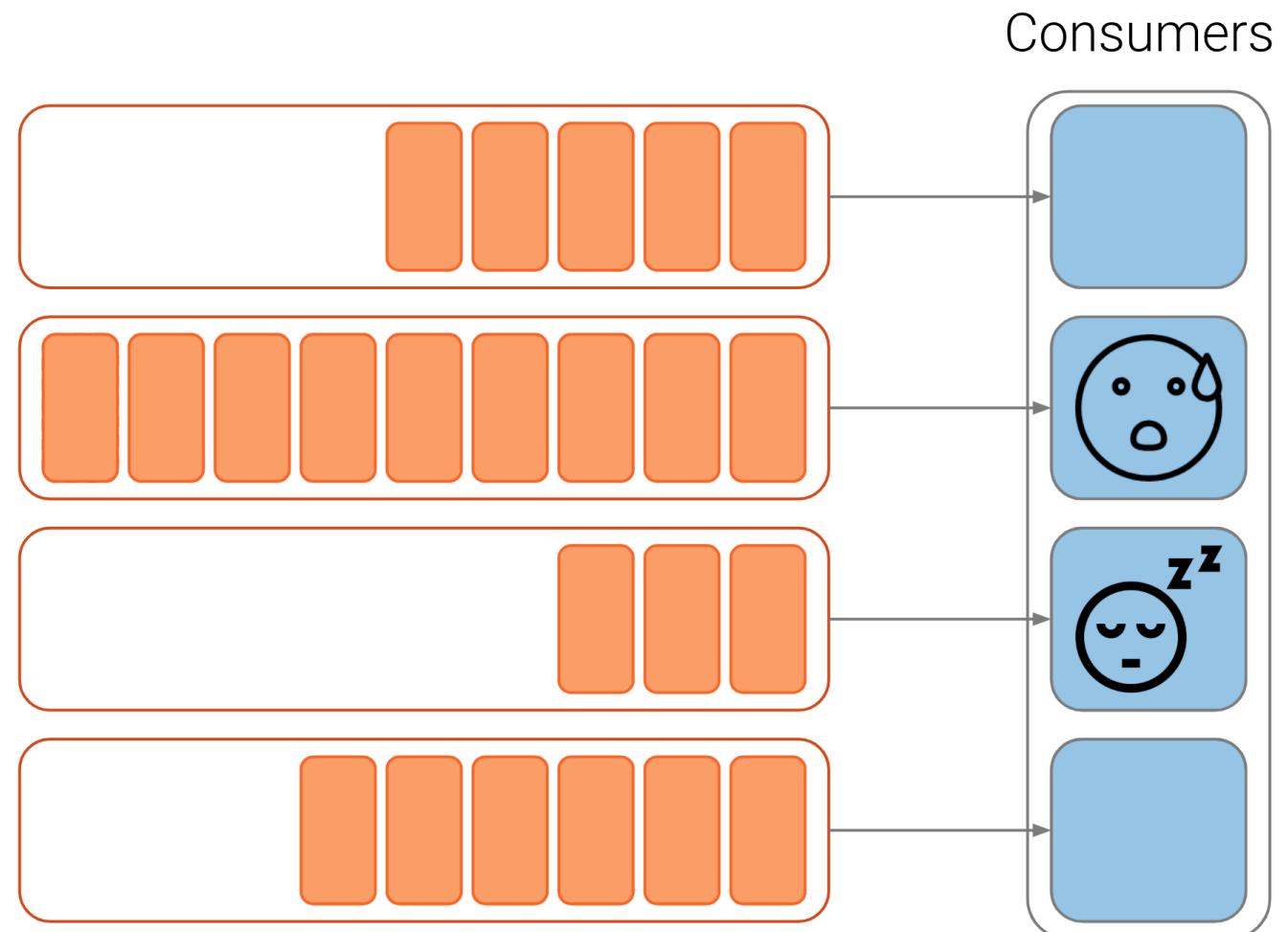
- Hard Prerequisite: Fundamentals Course
- Recommended Prerequisite: Storing the Records Persistently

## 7a: What are the Basics of Scaling Consumption?

### Description

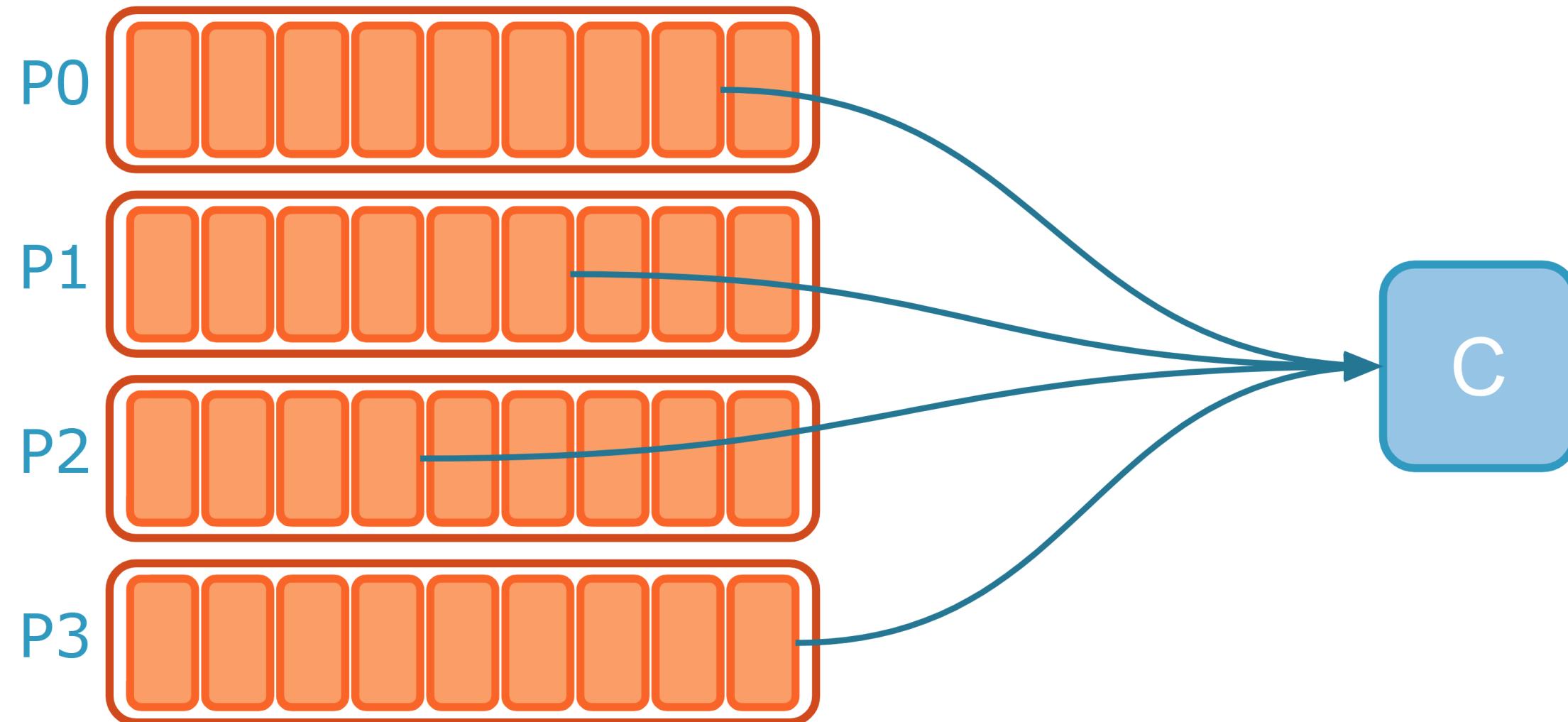
Consuming with one consumer vs. multiple consumers in a group vs. multiple groups.

# Cardinality



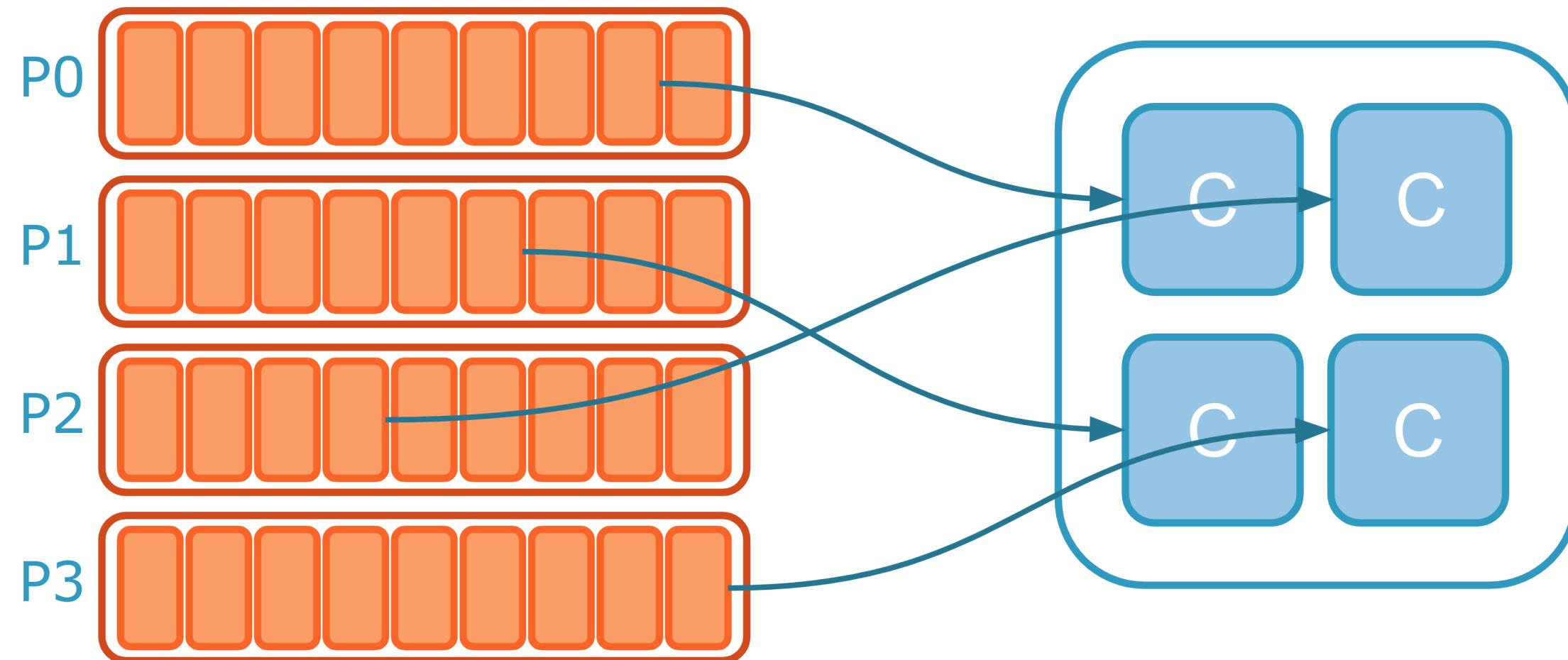
## Consuming from Kafka - Single Consumer

### Topic driver-positions

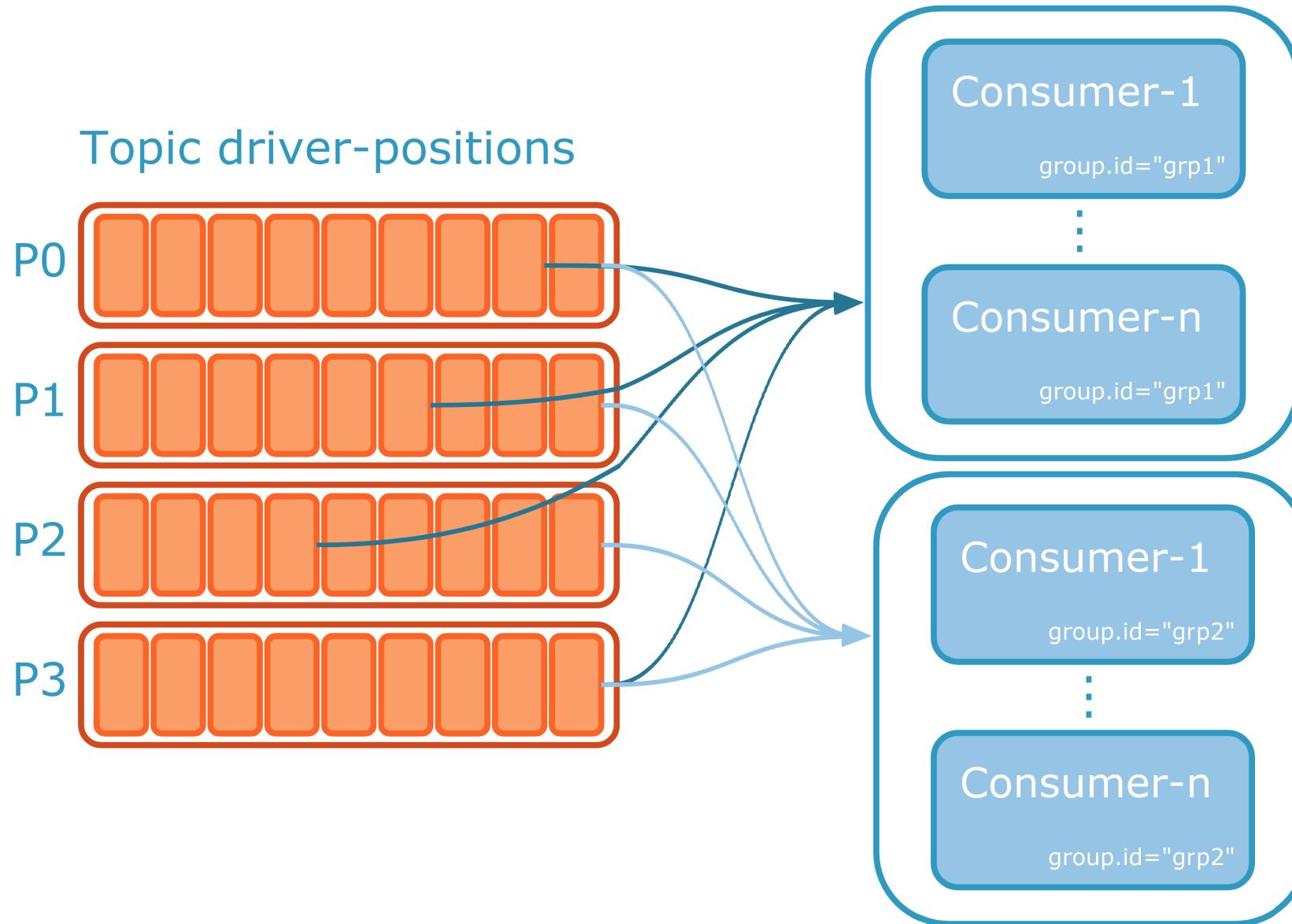


## Consuming as a Group

### Topic driver-positions



# Multiple Consumer Groups



## 7b: How Do Groups Distribute Work Across Partitions?

### Description

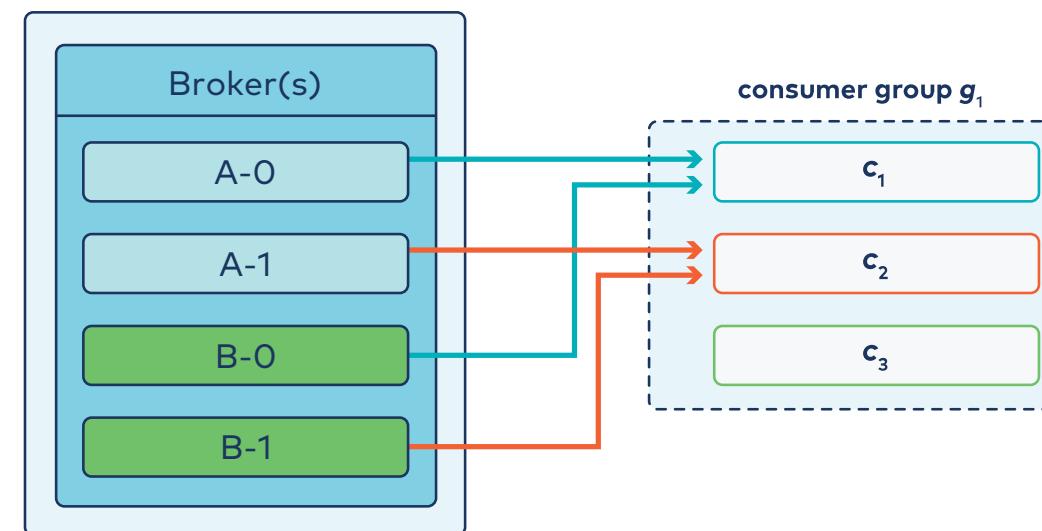
Assignment of partitions to consumers. Strategies: range, round-robin, sticky, cooperative sticky.

## Partition Assignment within a Consumer Group

- Partitions are 'assigned' to consumers
- A single partition is consumed by only one consumer in any given consumer group
  - Messages with same key will go to same consumer (unless you change number of partitions)
  - `partition.assignment.strategy` in the consumer configuration

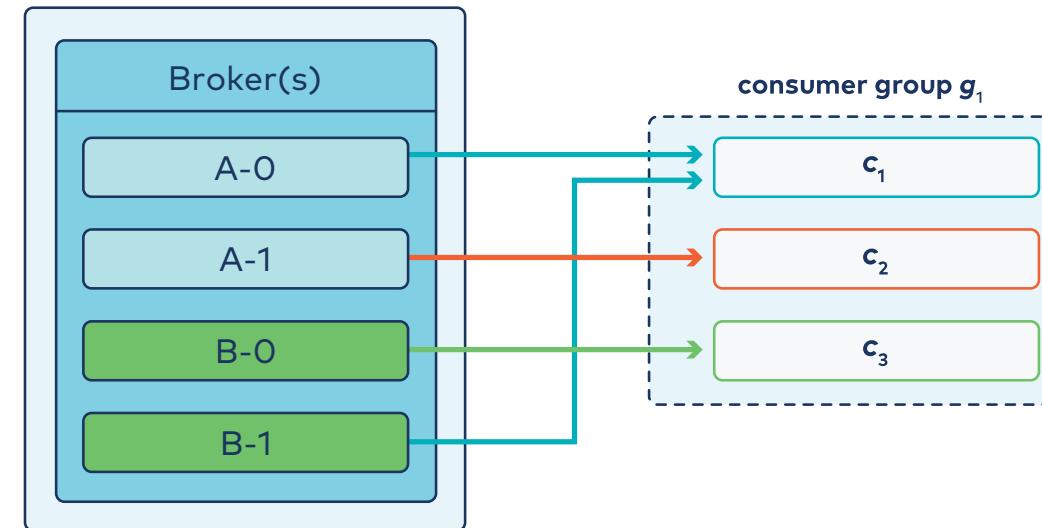
## Partition Assignment Strategy: Range

- Range is the default `partition.assignment.strategy`
- Useful for co-partitioning across topics, e.g.:
  - Package ID across `delivery_status` and `package_location`
  - User ID across `search_results` and `search_clicks`



## Partition Assignment Strategy: RoundRobin

- Partitions assigned one at a time in rotating fashion



## Partition Assignment Strategy: Sticky and CooperativeSticky

- Sticky
  - Is RoundRobin with assignment preservation across rebalances
- CooperativeSticky
  - Is Sticky without its "stop-the-world" rebalancing of all partitions

## 7c: How Does Kafka Manage Groups?

### Description

Group management. Rebalances. Heartbeats and failure detection.

# Consumer Rebalancing

## Rebalance triggers:

- Consumer leaves consumer group
- New consumer joins consumer group
- Consumer changes its topic subscription
- Consumer group notices change to topic metadata (e.g., increase # partitions)

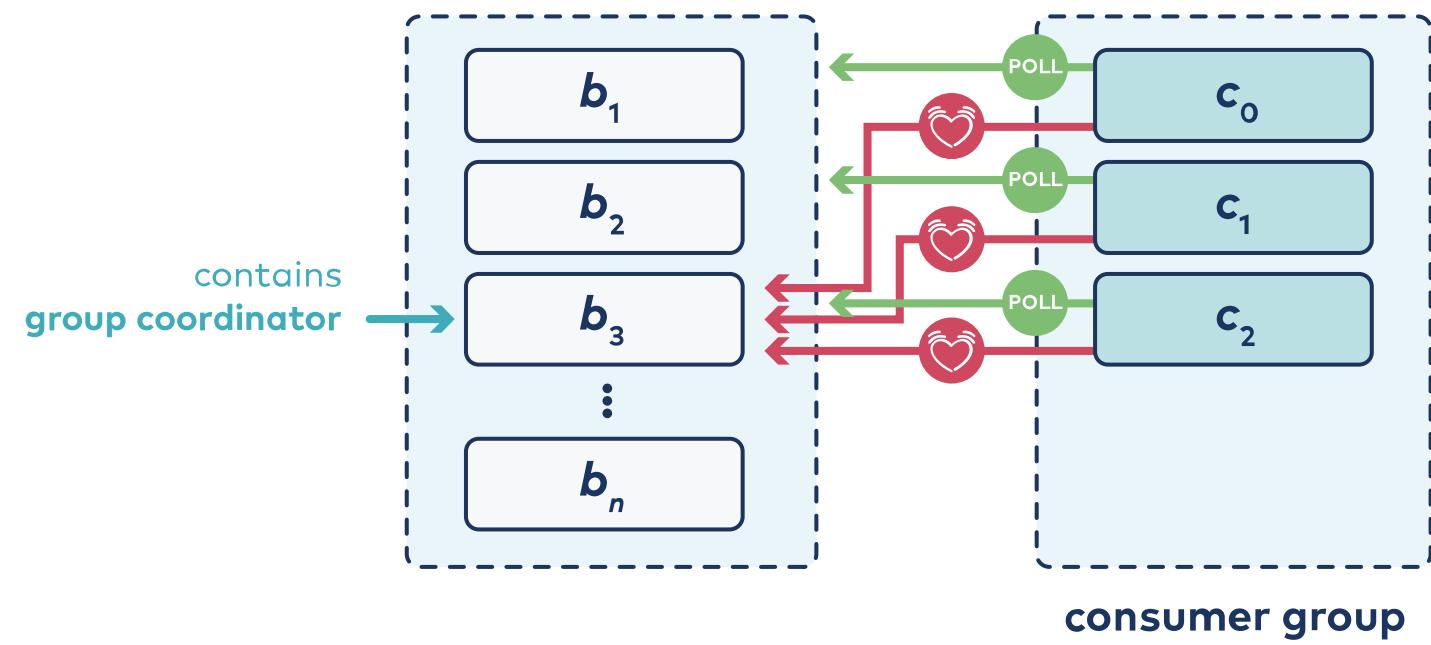
## Rebalance Process:

1. Group coordinator uses flag in heartbeat to signal rebalance to consumers
2. Consumers pause, commit offsets
3. Consumers rejoin into new "generation" of consumer group
4. Partitions are reassigned
5. Consumers resume from new partitions



Consumption pauses during rebalance. Avoid unnecessary rebalances.

# Consumer Failure Detection



- Consumers send heartbeats in background thread, separate from `poll()`
  - `heartbeat.interval.ms` (Default: 3 s)
- `session.timeout.ms` (Default: 45 s)
  - If no heartbeat is received in this time, consumer is dropped from group
- `poll()` must still be called periodically
  - `max.poll.interval.ms` (Default: 5 minutes)

## Avoiding Excessive Rebalances

- Tune `session.timeout.ms`:
  - Pro: gives more time for consumer to rejoin
  - Con: takes longer to detect hard failures
- Tune `max.poll.interval.ms`
  - Give consumers enough time to process data from `poll()`

### Static group membership:

- Assign each consumer in group unique `group.instance.id`
- Consumers do not send `LeaveGroupRequest`
- Rejoin doesn't trigger rebalance for known `group.instance.id`
- Rebalances are still triggered on heartbeat and poll timeouts



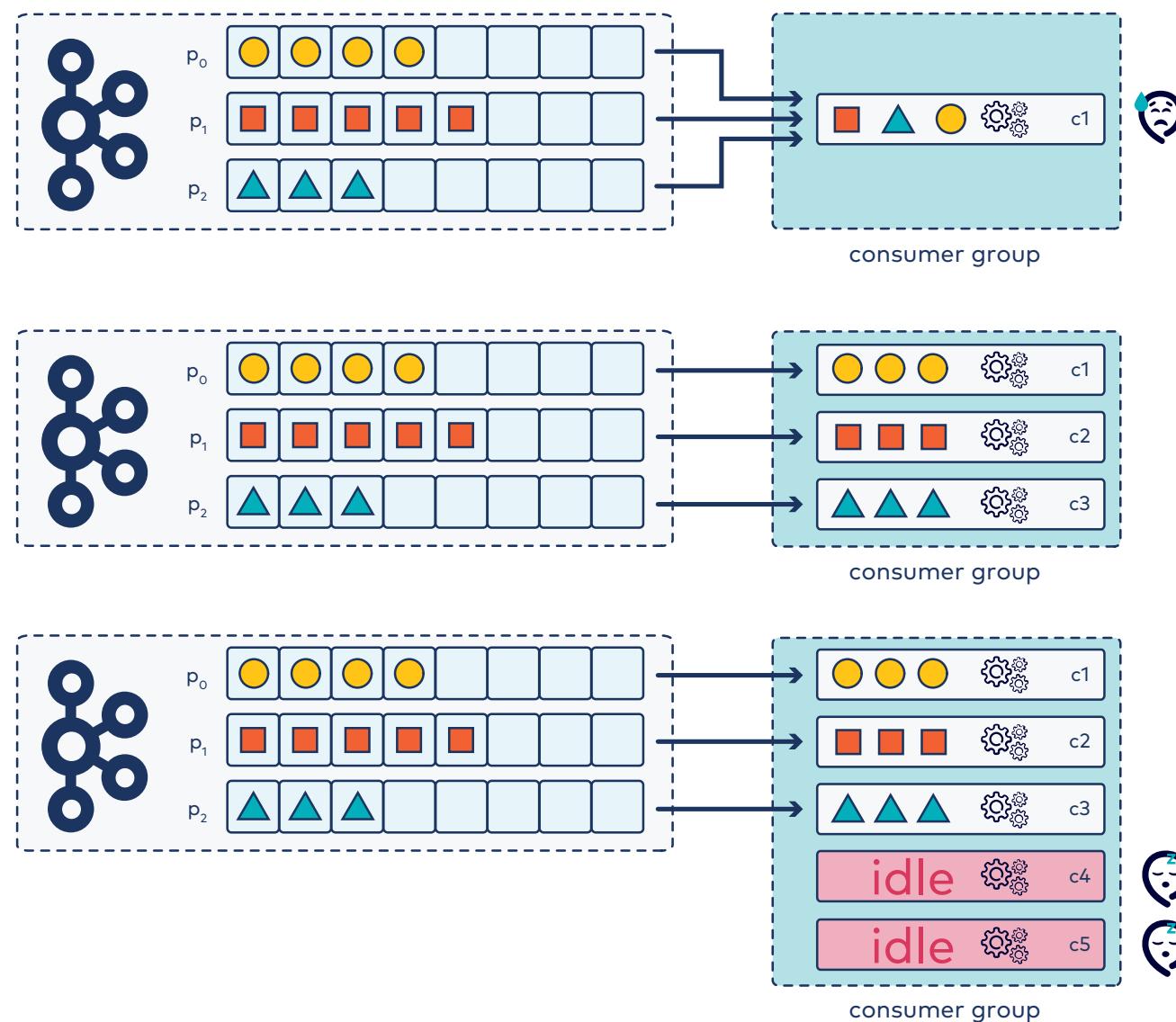
These configurations are set in consumer code.

## 7d: How Do Partitions and Consumers Scale?

### Description

Scalability of consumer groups. Adding partitions. Benefits and challenges of adding partitions.

# Consumer Group Scalability



## Adding Partitions

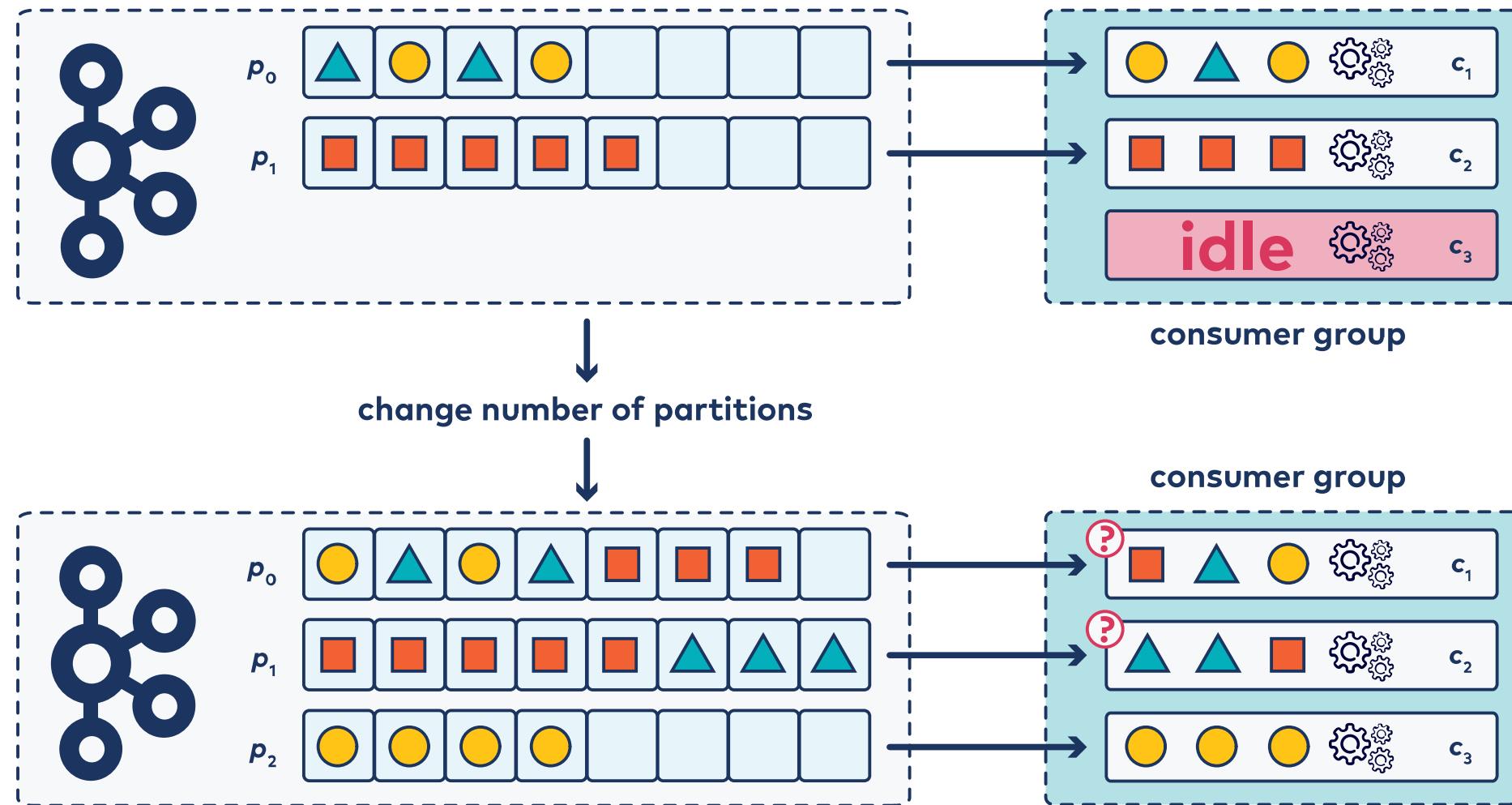
Use the `kafka-topics` command, e.g.:

```
$ kafka-topics \
  --bootstrap-server broker_host:9092 \
  --alter \
  --topic my_topic \
  --partitions 30
```

### Notes:

- Doesn't move data from existing partitions
- Messages with the same key will no longer be on the same partition
  - Workaround: Consume from old topic and produce to a new topic with the correct number of partitions

# Consumer Groups: Caution When Changing Partitions



## Number of Partitions

- Minimum number of partitions:  $\max(t/p, t/c)$ 
  - $t$ : target throughput
  - $p$ : Producer throughput per Partition
  - $c$ : Consumer throughput per Partition
- Other considerations:
  - A number with many divisors
  - Same number of partitions as other related topics

## Improving Throughput With More Partitions

- More partitions → higher throughput
- Rule of thumb for maximums:
  - Up to 4,000 partitions per broker
  - Up to 200,000 partitions per cluster

## Downside to More Partitions

- More open file handles
- Longer leader elections → more downtime after broker failure
- Higher latency due to replication
- More client memory (buffering per partition)



When producing keyed messages: avoid unbalanced key utilization. This leads to "hot partitions."

## 7e: How Does Kafka Maintain Consumer Offsets?

### Description

Consumer offsets topic: uses, special properties. Viewing offsets.

## Consumer Offset Management

- Consumption is tracked per topic-partition
- Each consumer maintains an offset in its memory for each partition it is reading
- As a consumer processes messages, it periodically commits the offset of the next message to be consumed
  - Offsets are committed to an internal Kafka topic `__consumer_offsets`
  - Offsets can be committed automatically or manually by the consumer

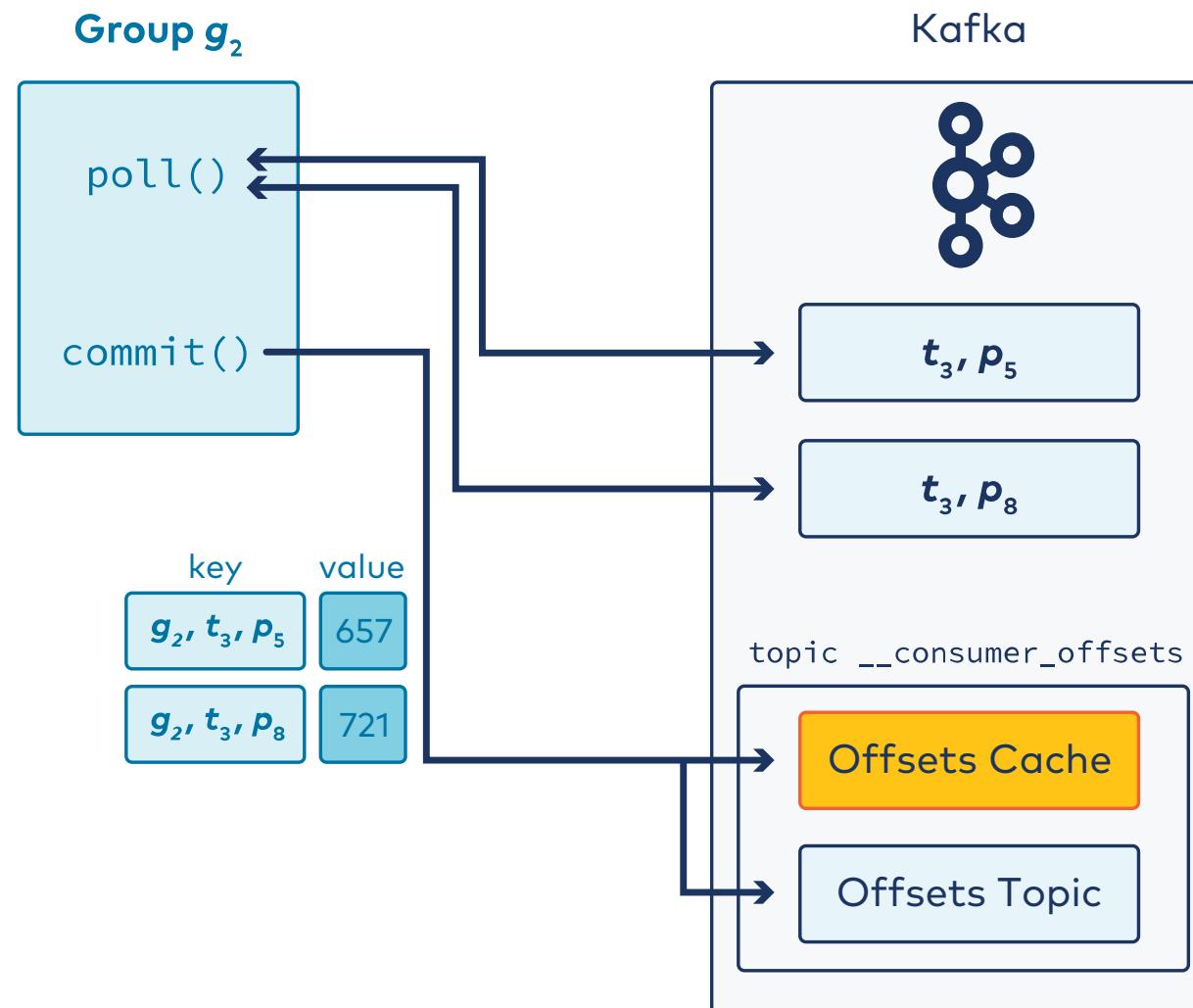
## Important Configuration Settings for Offsets

- `--consumer_offsets` auto-created upon first consumption
- Scalability: `offsets.topic.num.partitions` (Default: 50)
- Resiliency: `offsets.topic.replication.factor` (Default: 3)

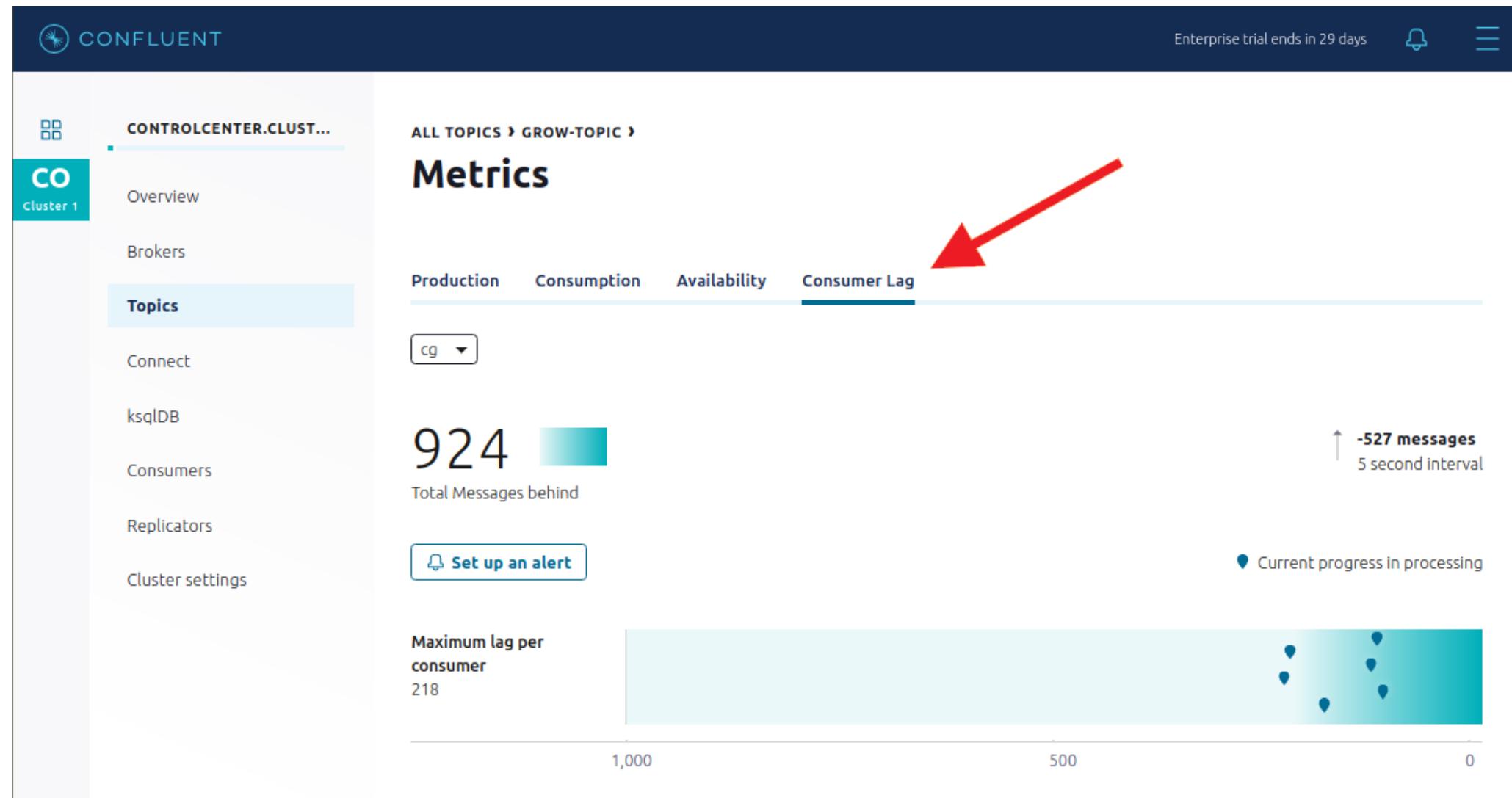


If there aren't enough brokers, auto-creation of `--consumer_offsets` fails. Consumers should only begin consuming after all brokers are running.

# Consumers and Offsets (Kafka Topic Storage)



# Checking Consumer Offsets (1)



## Checking Consumer Offsets (2)

Look for the current offset and lag:

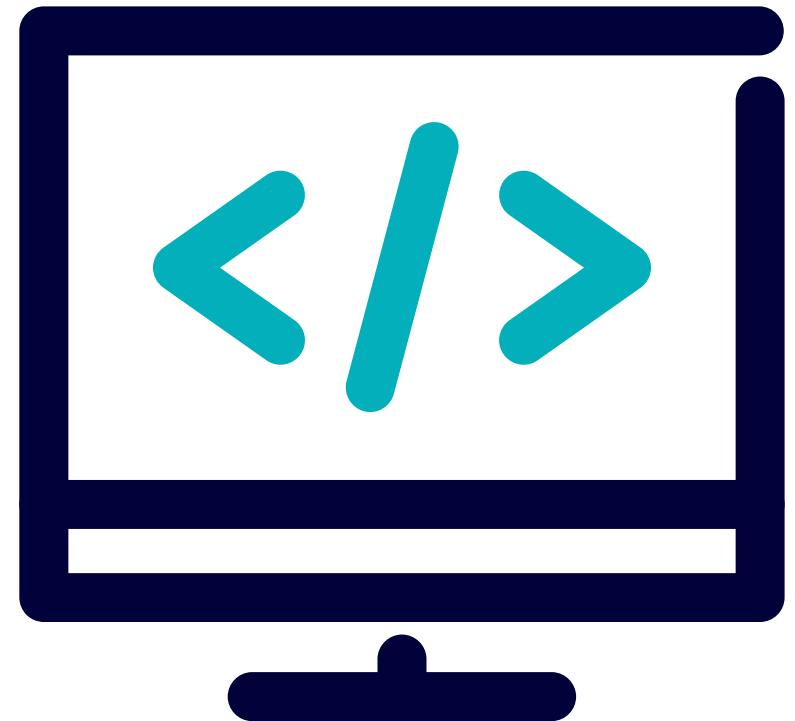
```
$ kafka-consumer-groups --group my-group \
    --describe \
    --bootstrap-server=broker101:9092,broker102:9092,broker103:9092
```

TOPIC,	PARTITION,	CURRENT OFFSET,	LOG END OFFSET,	LAG,	CONSUMER-ID
my_topic,	0,	400,	500,	100,	consumer-1_/127.0.0.1
my_topic,	1,	500,	500,	0,	consumer-1_/127.0.0.1

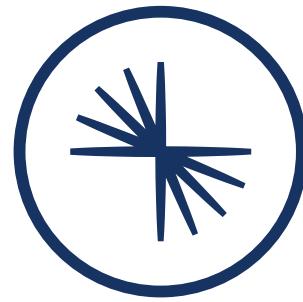
# Lab: Modifying Partitions and Viewing Offsets

Please work on **Lab 7a: Modifying Partitions and Viewing Offsets**

Refer to the Exercise Guide



## 8: Optimizing Kafka's Performance



CONFLUENT  
**Global Education**

# Module Overview



This module contains 7 lessons:

- How Does Kafka Handle the Idea of Sending Many Messages at Once?
- How Do Produce and Fetch Requests Get Processed on a Broker?
- How Can You Measure and Control How Requests Make It Through a Broker?
- What Else Can Affect Broker Performance?
- How Do You Control It So One Client Does Not Dominate the Broker Resources?
- What Should You Consider in Assessing Client Performance?
- How Can You Test How Clients Perform?

# The Meaning of Performance

- **Throughput**
  - amount of data moving through Kafka per second
- **Latency**
  - The delay from the time data is written to the time it is read
- **Recovery Time**
  - The time to return to a “good” state after some failure

## 8a: How Does Kafka Handle the Idea of Sending Many Messages at Once?

### Description

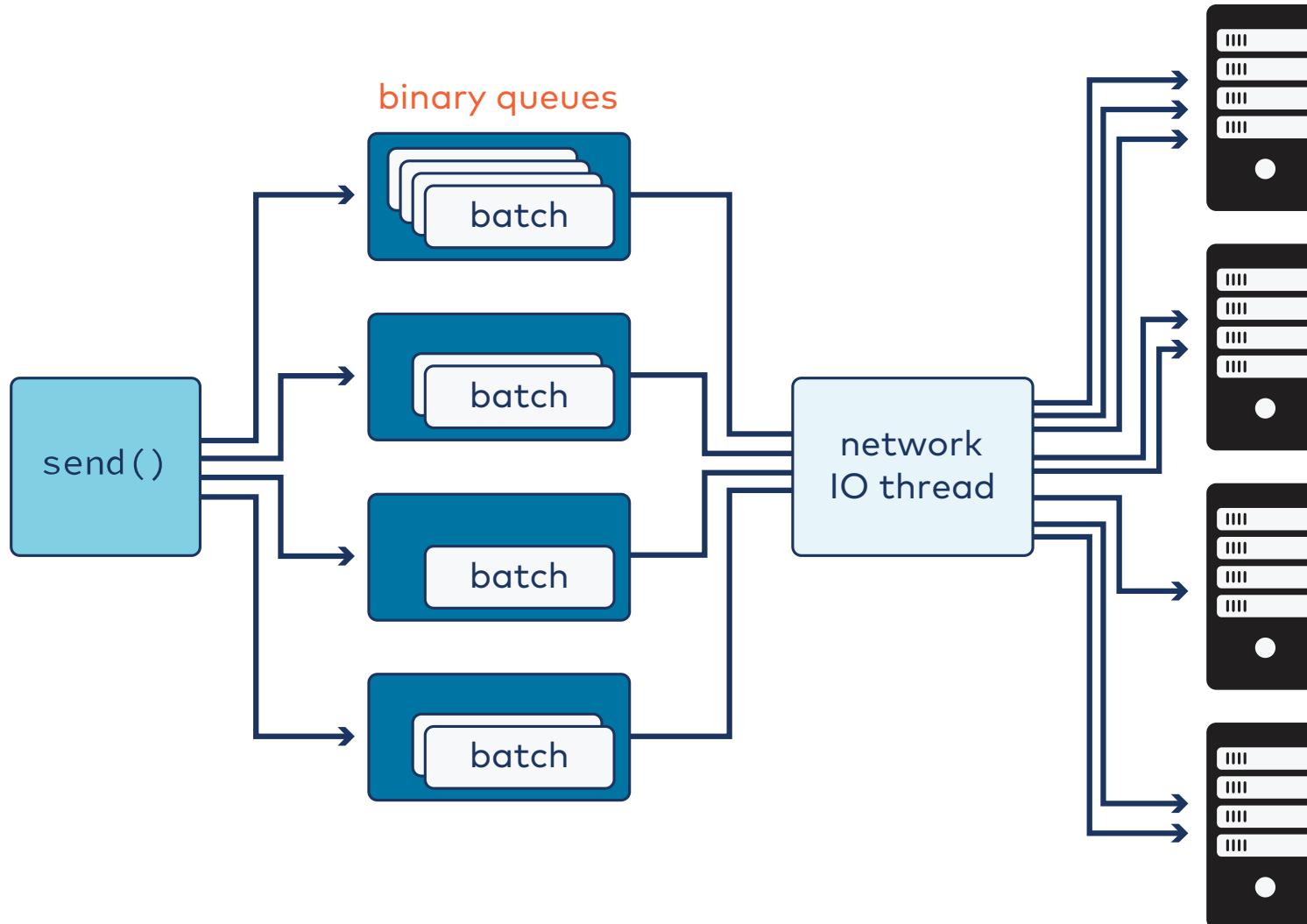
Batching. Pipelining. Tuning batching. Compression.

# Batching for Higher Throughput

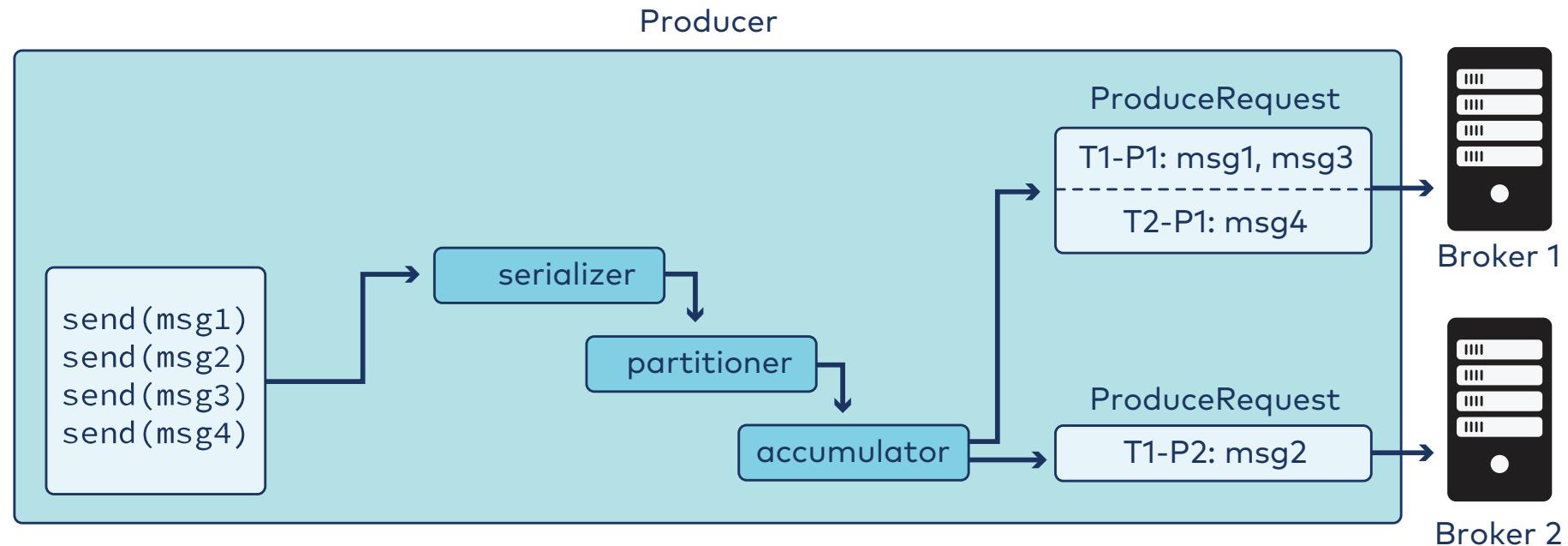


# Producer Architecture

- **Pipelining:** multiple in-flight send requests per broker
  - `max.in.flight.requests.per.connection` (default: 5)



## Batching Messages (1)

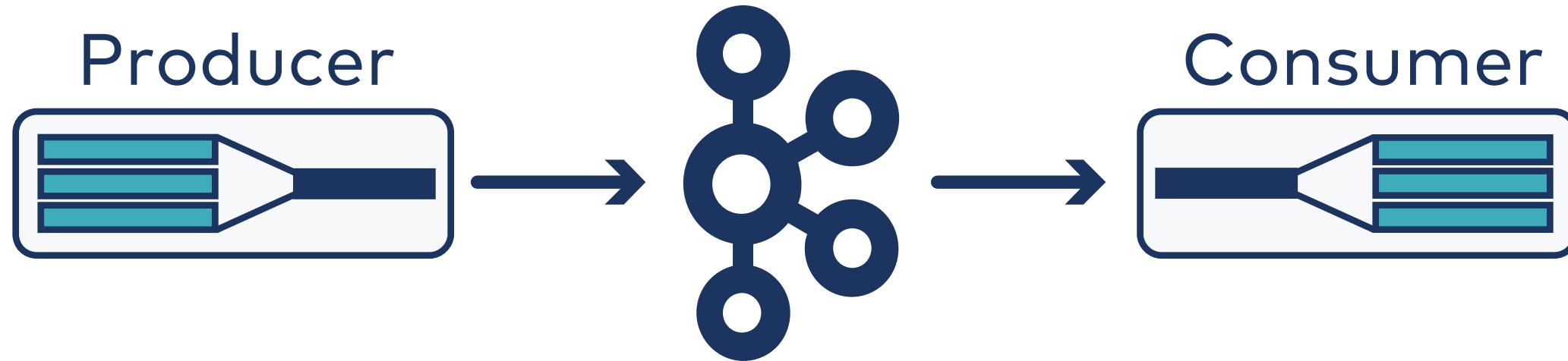


1. First, batch messages by partition
2. Then, collect batches into ProduceRequests to brokers

## Batching Messages (2)

- `batch.size` (Default: 16 KB):
  - The maximum size of a batch before sending
- `linger.ms` (Default: 0, i.e., send immediately):
  - Time to wait for messages to batch together

## End-To-End Batch Compression



1. Producer batches messages and compresses the batch
2. Compressed batch stored in Kafka
3. Consumer decompresses and un-batches messages

# Tuning Producer Throughput and Latency

- `batch.size`, `linger.ms`
  - High throughput: large `batch.size` and `linger.ms`, or flush manually
  - Low latency: small `batch.size` and `linger.ms`
- `buffer.memory`
  - Default: 32 MB
  - The producer's buffer for messages to be sent to the cluster
  - Increase if producers are sending faster than brokers are acknowledging, to prevent blocking
- `compression.type`
  - `gzip`, `snappy`, `lz4`, `zstd`
  - Configurable per producer, topic, or broker

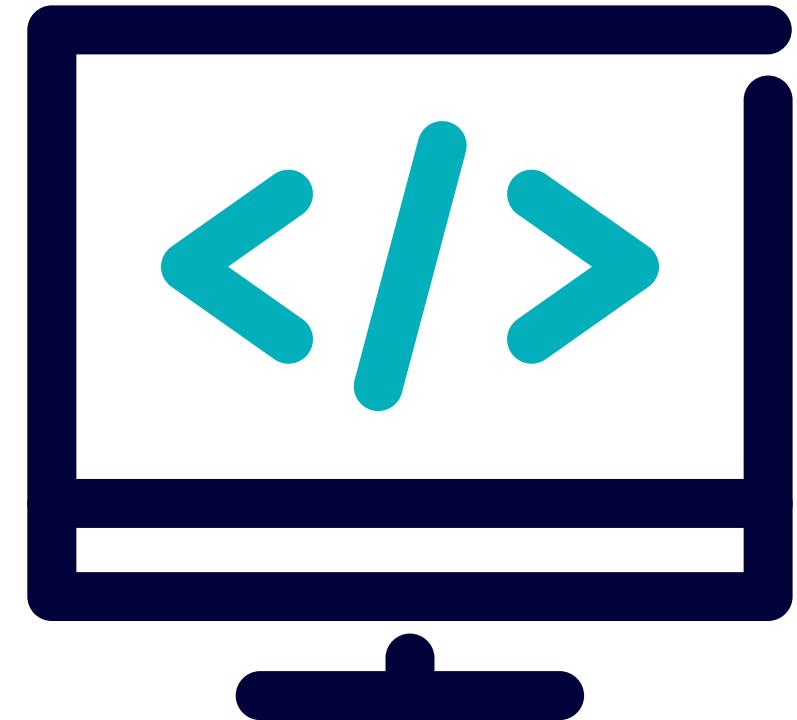
## Tuning Consumer Throughput and Latency

- High throughput:
  - Large `fetch.min.bytes` (Default: 1)
  - Reasonable `fetch.max.wait.ms` (Default: 500)
- Low latency:
  - `fetch.min.bytes=1`

# Lab: Exploring Producer Performance

Please work on **Lab 8a: Exploring Producer Performance**

Refer to the Exercise Guide



## 8b: How Do Produce and Fetch Requests Get Processed on a Broker?

### Description

Thread pools. Queues. Purgatory. The path of a produce request from broker receipt to acknowledgement. The path of a fetch request from broker receipt to response with data.

## Review

We know:

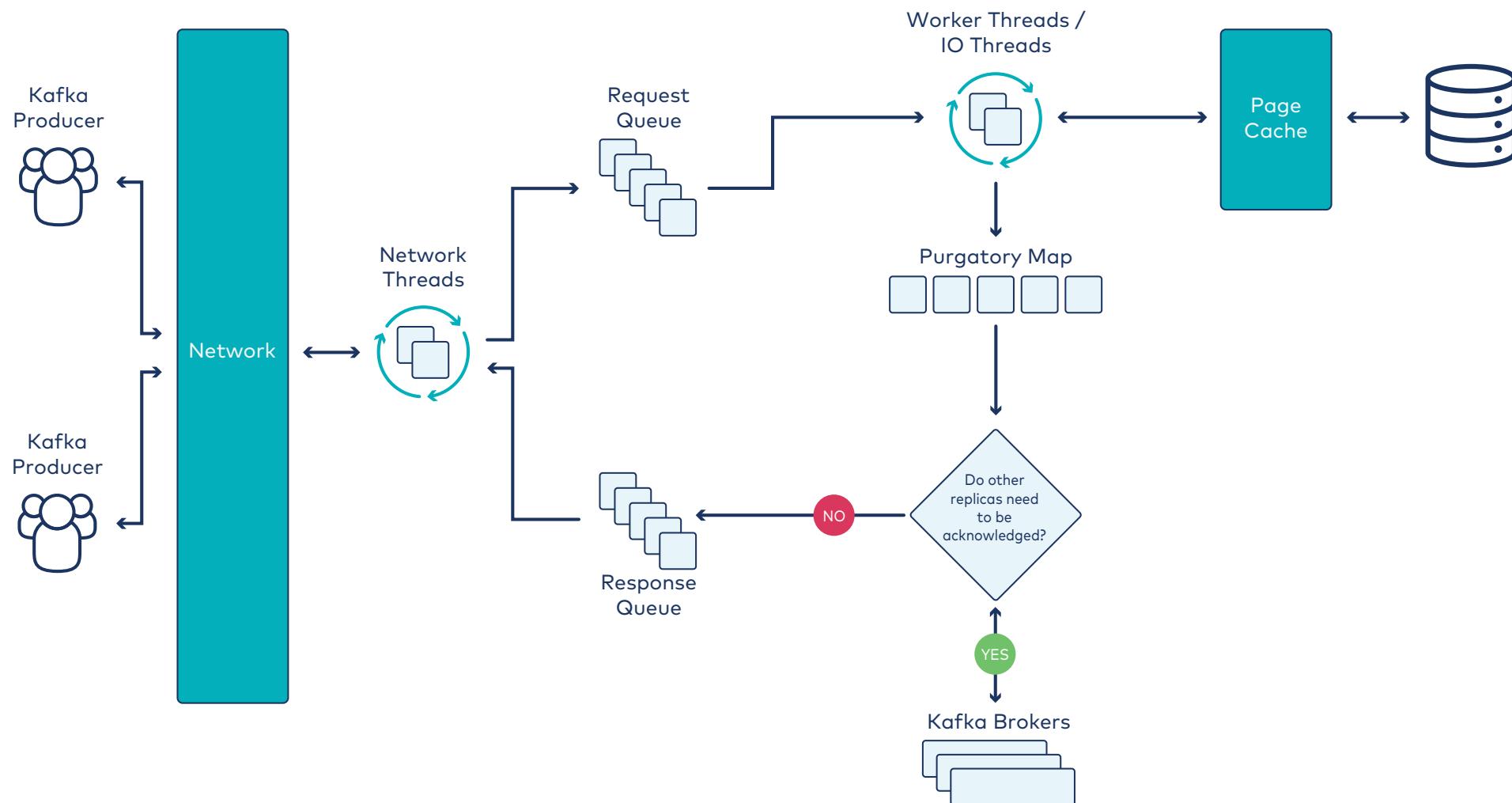
- Producers prepare messages to send.
- Producer settings like `linger.ms` and `batch.size` control how messages get grouped in batches to send.
- Batches of messages may be compressed, according to producer setting `compression.type`.
- We can configure producer property `acks` to have producers request to hear back from Kafka when messages are successfully written.

When a producer sends a batch, we say it is creating a **produce request**. So...

- What does a broker do when it receives a produce request?
- How exactly does a broker satisfy a producer's `acks` request?

Let's find out...

# Anatomy of a Produce Request

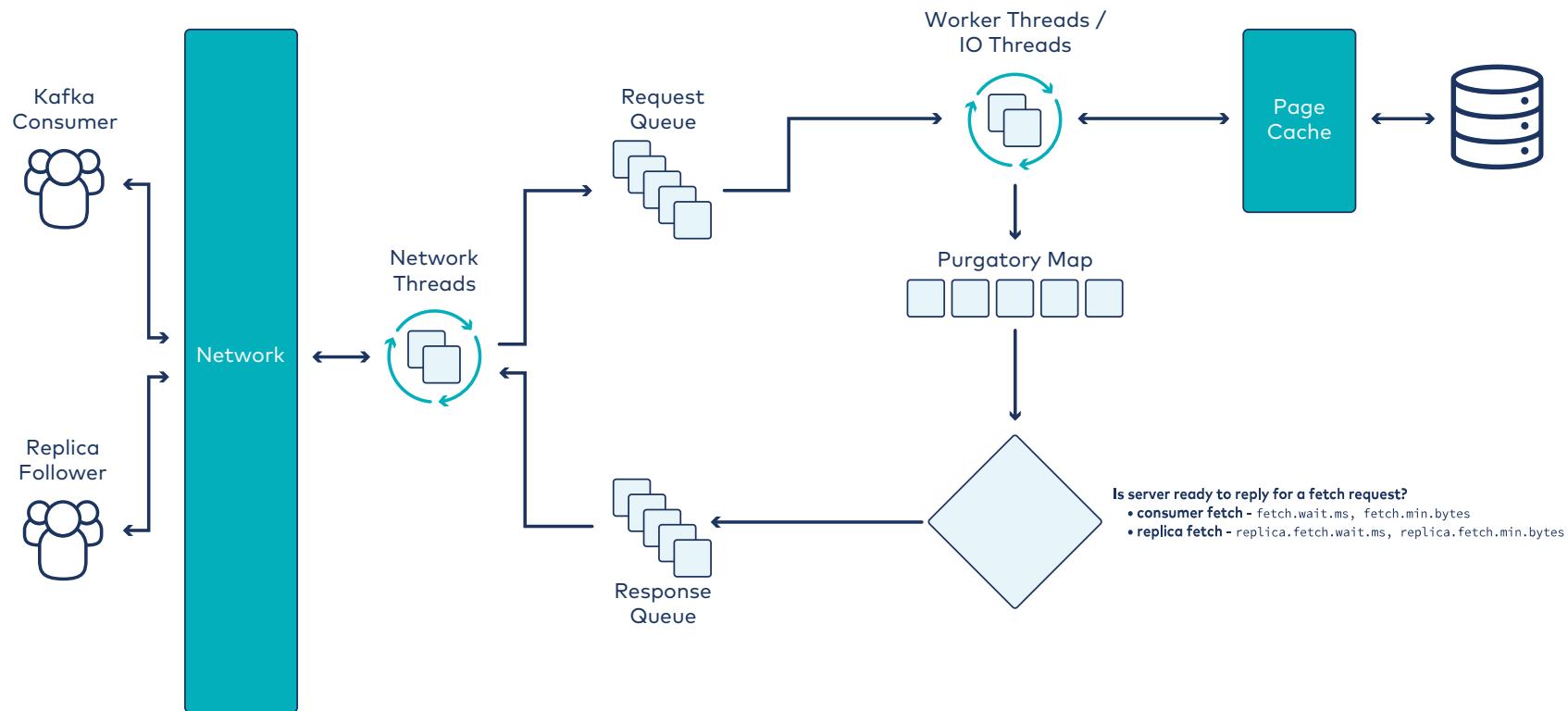


## Handling Replication and `acks = all`: Purgatory

### Purgatory

Structure in memory for holding produce requests that are not yet complete

# Anatomy of a Fetch Request



The queues and thread pools here are exactly the same as we saw before.



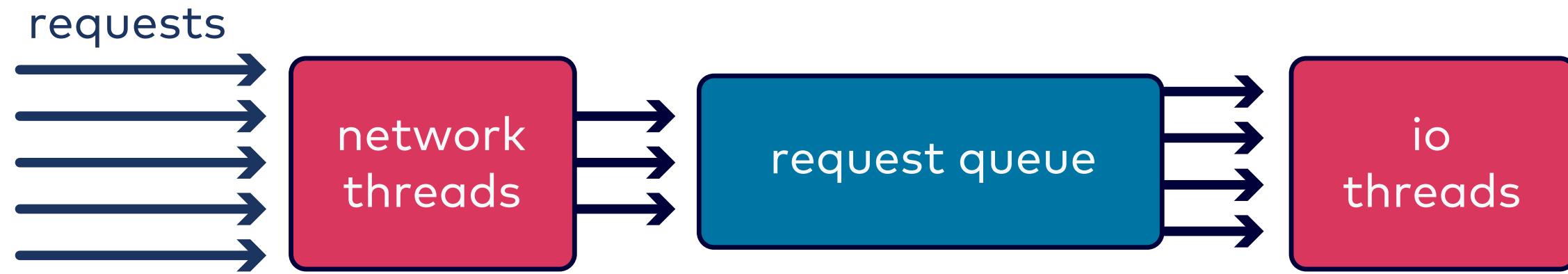
The produce purgatory and fetch purgatory are separate.

## 8c: How Can You Measure and Control How Requests Make It Through a Broker?

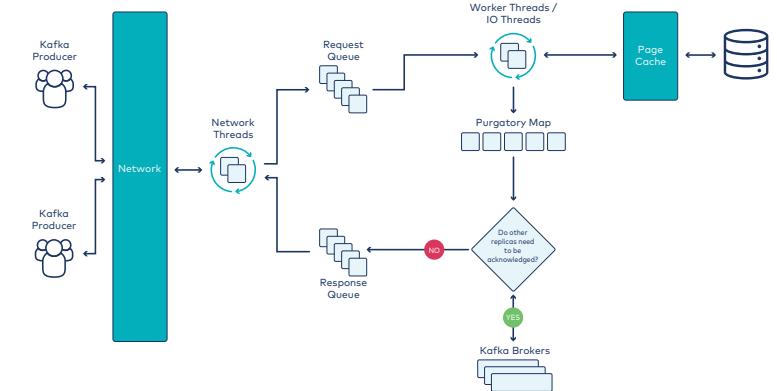
### Description

Metrics and tuning settings for thread pools and queues in the broker's request anatomy. Measuring request latency overall and at stages.

# Performance Tuning the Thread Pools



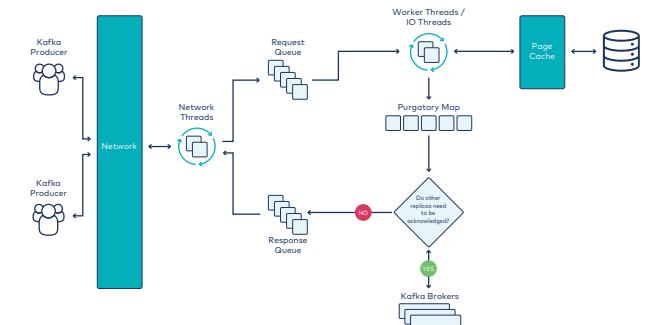
- Each thread pool is configurable
  - `num.network.threads` (default: 3, increase for TLS)
  - `num.io.threads` (default: 8)



## Performance Tuning the Request Queue



- The size of the request queue is `queued.max.requests` (default: 500)
- If the request queue is filled, the network threads stop reading in new requests
- So, consider the number of clients and brokers

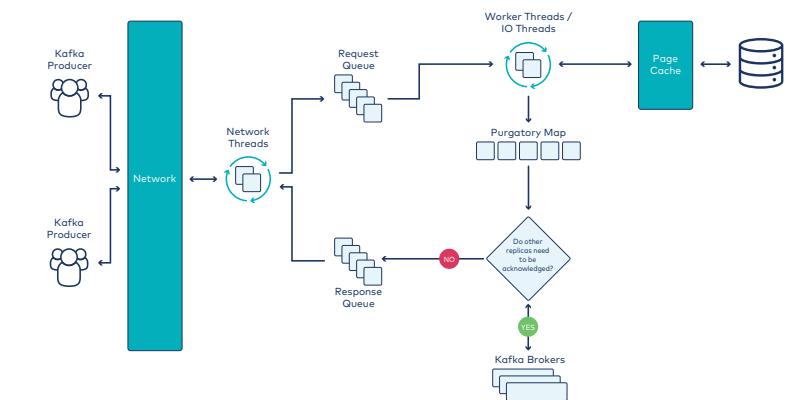


# Monitoring The Request Queue

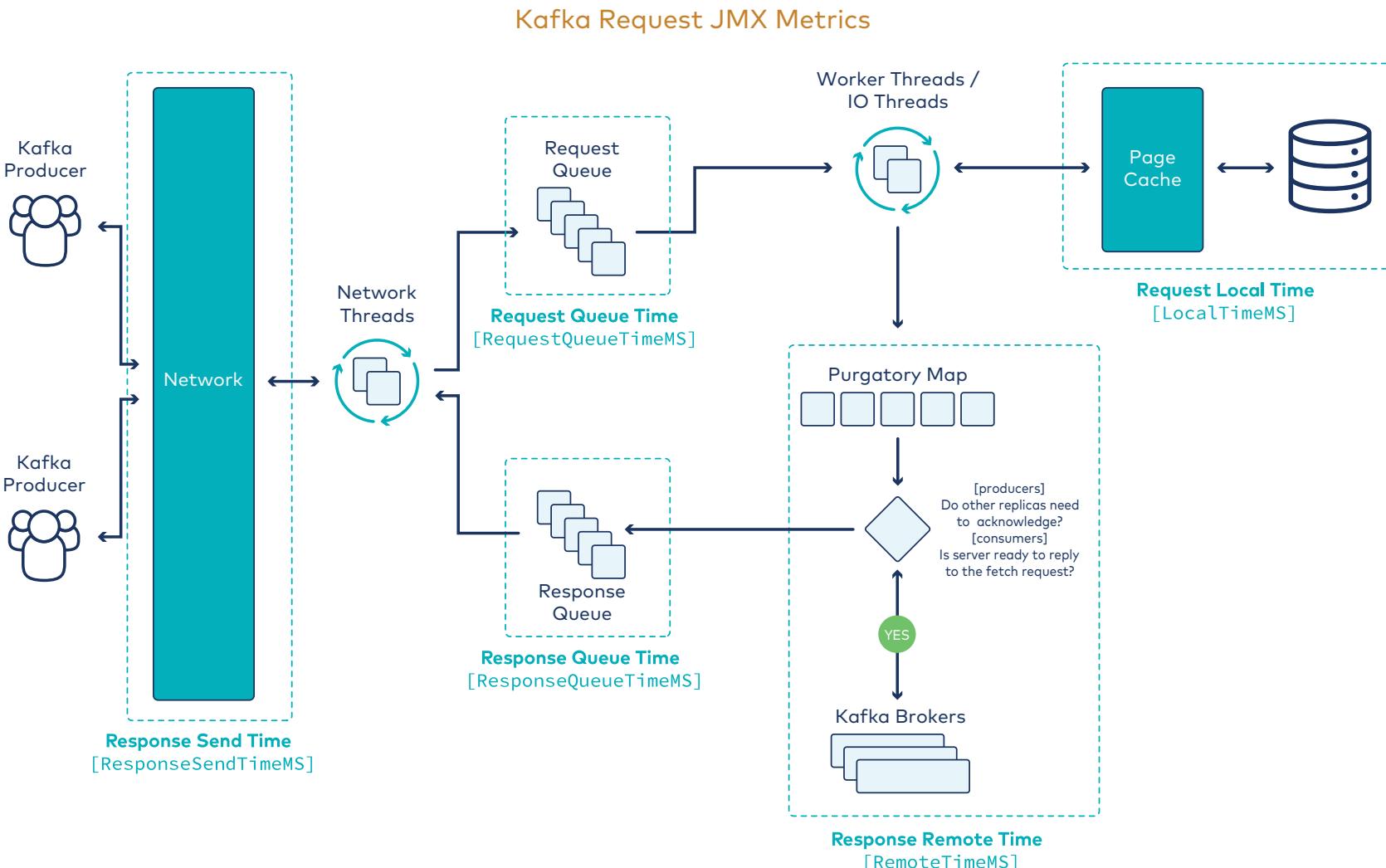
- JMX metrics:

```
kafka.network:type=RequestChannel, name=RequestQueueSize  
kafka.network:type=RequestMetrics, name=RequestQueueTimeMs
```

- Congested request queue can't process incoming or outgoing requests



# Monitoring Requests on the Broker



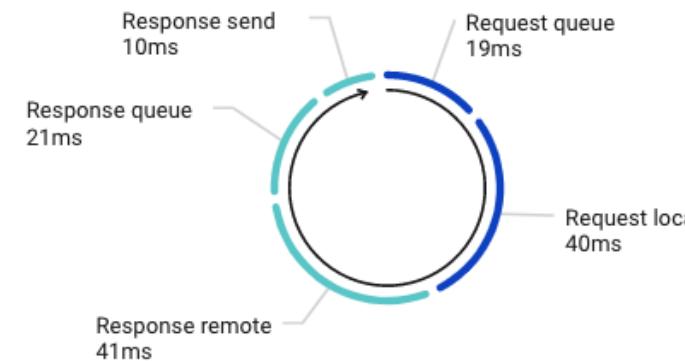
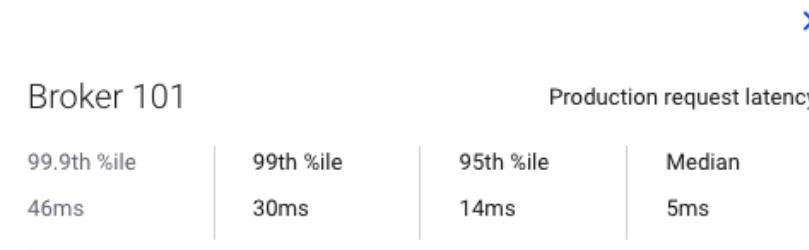
## Request Lifecycle and Latencies

Break down `TotalTimeMs` further to see the entire request lifecycle:

Metric	Description
<code>RequestQueueTimeMs</code>	Time the request waits in the request queue
<code>ResponseSendTimeMs</code>	Time to send the response
<code>ResponseQueueTimeMs</code>	Time the request waits in the response queue
<code>LocalTimeMs</code>	Time the request is processed at the leader
<code>RemoteTimeMs</code>	Time the request waits for the follower

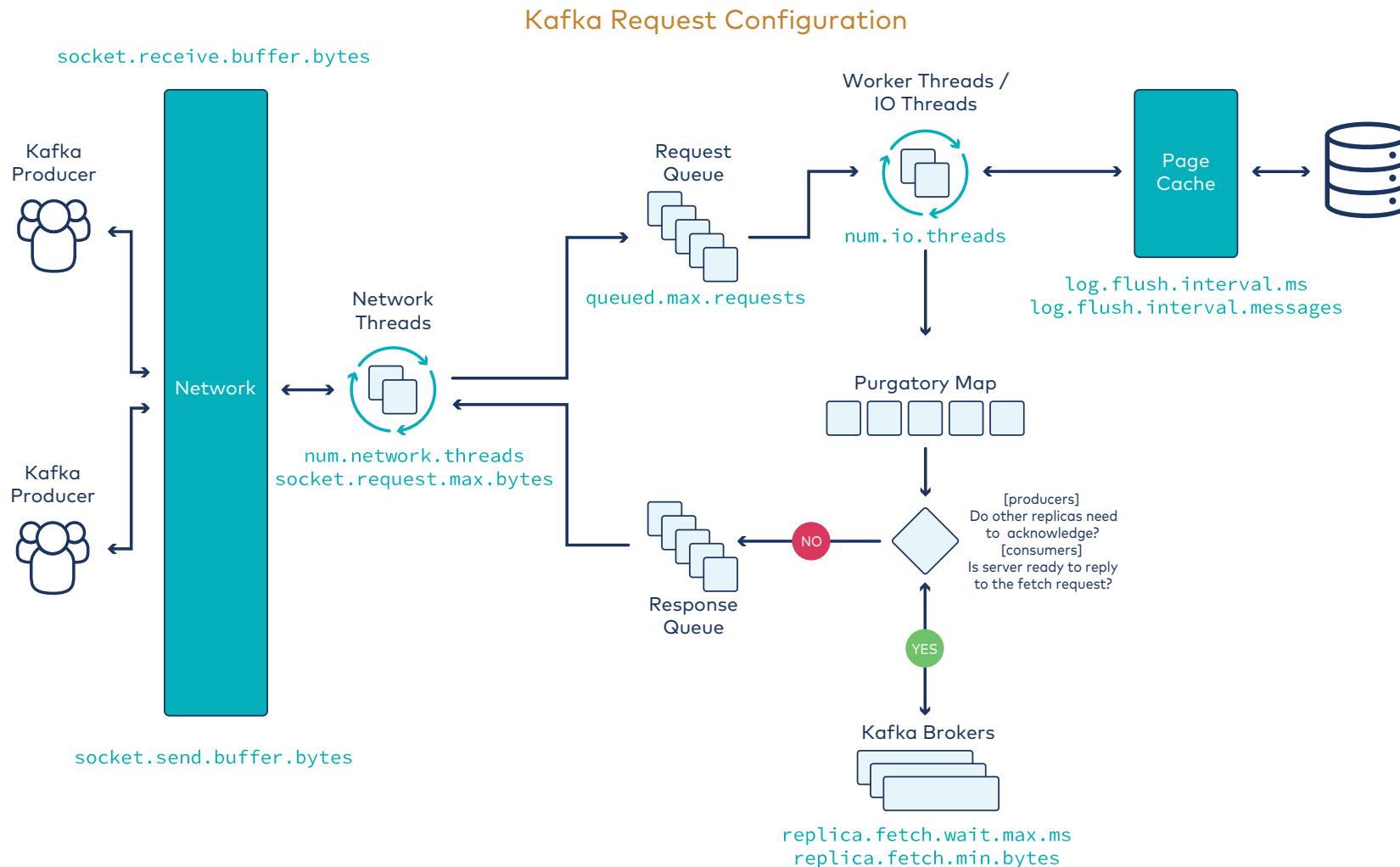
# Request Lifecycle and Latencies

## Confluent Control Center:



Each metric is a percentile, and percentile metrics don't add associatively.

# Configuring Requests on the Broker



## 8d: What Else Can Affect Broker Performance?

### Description

Other monitoring on the broker. Message size. Garbage collection considerations.  
Troubleshooting exercise.

# Monitoring Leaders and Partitions

- Monitoring with JMX Metrics

- LeaderCount (gauge)

```
kafka.server:type=ReplicaManager,name=LeaderCount
```

- PartitionCount (gauge)

```
kafka.server:type=ReplicaManager,name=PartitionCount
```



Leadership and partitions should be spread evenly across brokers

## Message Size Limit



Avoid changing maximum message size. Kafka is not optimized for very large messages.

Brokers or Topics	Replication on Brokers	Consumers
<ul style="list-style-type: none"><li>• <code>message.max.bytes</code> (B)</li><li>• <code>max.message.bytes</code> (T)<ul style="list-style-type: none"><li>◦ maximum size of message that the broker can receive from a producer (Default: 1 MB)</li></ul></li></ul>	<ul style="list-style-type: none"><li>• <code>replica.fetch.max.bytes</code><ul style="list-style-type: none"><li>◦ maximum amount of data per-partition that brokers send for replication (Default: 1 MB)</li></ul></li></ul>	<ul style="list-style-type: none"><li>• <code>max.partition.fetch.bytes</code><ul style="list-style-type: none"><li>◦ maximum amount of data per-partition the broker will return (Default: 1 MB)</li></ul></li></ul>

B → Broker Setting | T → Topic Override

## 8e: How Do You Control It So One Client Does Not Dominate the Broker Resources?

### Description

Client quotas: motivation, configuration, monitoring.

## Ensure High Performance with Quotas

- High volume clients can result in:
  - Monopolizing broker resources
  - Network saturation
  - Denial of Service (DoS) to other producers and consumers
  - DoS brokers themselves
- Use **quotas** to throttle clients or groups of clients from overloading a broker
  - Quotas are per-broker, not cluster-wide

## How Quotas Work

- Quotas can be applied to:
  - Client-id: logical group of clients, identified by the same `client.id`
  - User: authenticated user principal
  - User and client-id pair: group of clients belonging to a user
- If quota exceeded, broker will:
  1. Compute a delay time for the client
  2. Instruct client to not send more requests during delay
  3. Mute client channel so its requests are not processed during delay

## Configure Quotas (1)

- Quotas can be defined by network bandwidth or request rate:
  - Network bandwidth: `producer_byte_rate`, `consumer_byte_rate`
  - Request rate: `request_percentage` (percentage of time a client can utilize the request handler I/O threads and network threads)
- Network bandwidth quota defaults, for example, to 1 KBps

```
$ kafka-configs \
  --bootstrap-server broker_host:9092 \
  --alter \
  --add-config 'producer_byte_rate=1024,consumer_byte_rate=1024' \
  --client-defaults
```

## Configure Quotas (2)

- Request rate quota override for a specific client-id, user, or user and client-id pair

```
$ kafka-configs --bootstrap-server broker_host:9092 \
  --alter \
  --add-config 'request_percentage=50' \
  --client clientA \
  --user user1
```

- To describe the quota for a specific user and client-id pair

```
$ kafka-configs --bootstrap-server broker_host:9092 \
  --describe \
  --client clientA \
  --user user1
```

## 8f: What Should You Consider in Assessing Client Performance?

### Description

IO ratio and IO wait ratio. Implications. Other JMX metrics for clients.

## Client Performance Metrics

- Client-level JMX metrics:

```
kafka.producer:type=producer-metrics,client-id=my_producer
```

```
kafka.consumer:type=consumer-metrics,client-id=my_consumer
```

- Producer-only metrics:

```
batch-size-avg
```

```
compression-rate-avg
```

- Per-topic metrics:

```
kafka.producer:type=producer-topic-metrics,client-id=my_producer,topic=my_topic
```

```
record-send-rate
```

```
byte-rate
```

```
record-error-rate
```

## 8g: How Can You Test How Clients Perform?

### Description

CLI client performance testing tools and their use.

## Why Test Kafka Performance At All?

- Benchmarks establish baseline for performance
  - Broker and client benchmarks → capacity planning
  - Analyze effect of cluster/client changes against baseline

## How to Test Performance

Determine **p**: producer throughput per partition

- Run a single producer on a single server
- `kafka-producer-perf-test`

Determine **c**: consumer throughput per partition

- Run a single consumer on a single server
- `kafka-consumer-perf-test`

## Measuring Throughput

- All topics bytes/messages in (meter)

```
kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec  
kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec  
kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesInPerSec
```

- All topics bytes out (meter)

```
kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec  
kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesOutPerSec
```

- Confluent Control Center provides per-broker and per-topic throughput metrics

# Lab: Performance Tuning

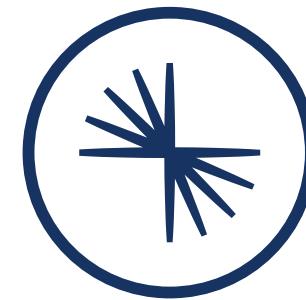
Please work on:

- Lab 8b: Performance Monitoring in Brokers
- Lab 8c: Tune Consumer Performance

Refer to the Exercise Guide



# 9: Securing a Kafka Cluster



CONFLUENT  
Global Education

# Module Overview



This module contains 4 lessons:

- What are the Basic Ideas You Should Know about Kafka Security?
- What Options Do You Have For Securing a Kafka/Confluent Deployment?
- How Can You Easily Control Who Can Access What?
- What Should You Know Securing a Deployment Beyond Kafka Itself?

Where this fits in:

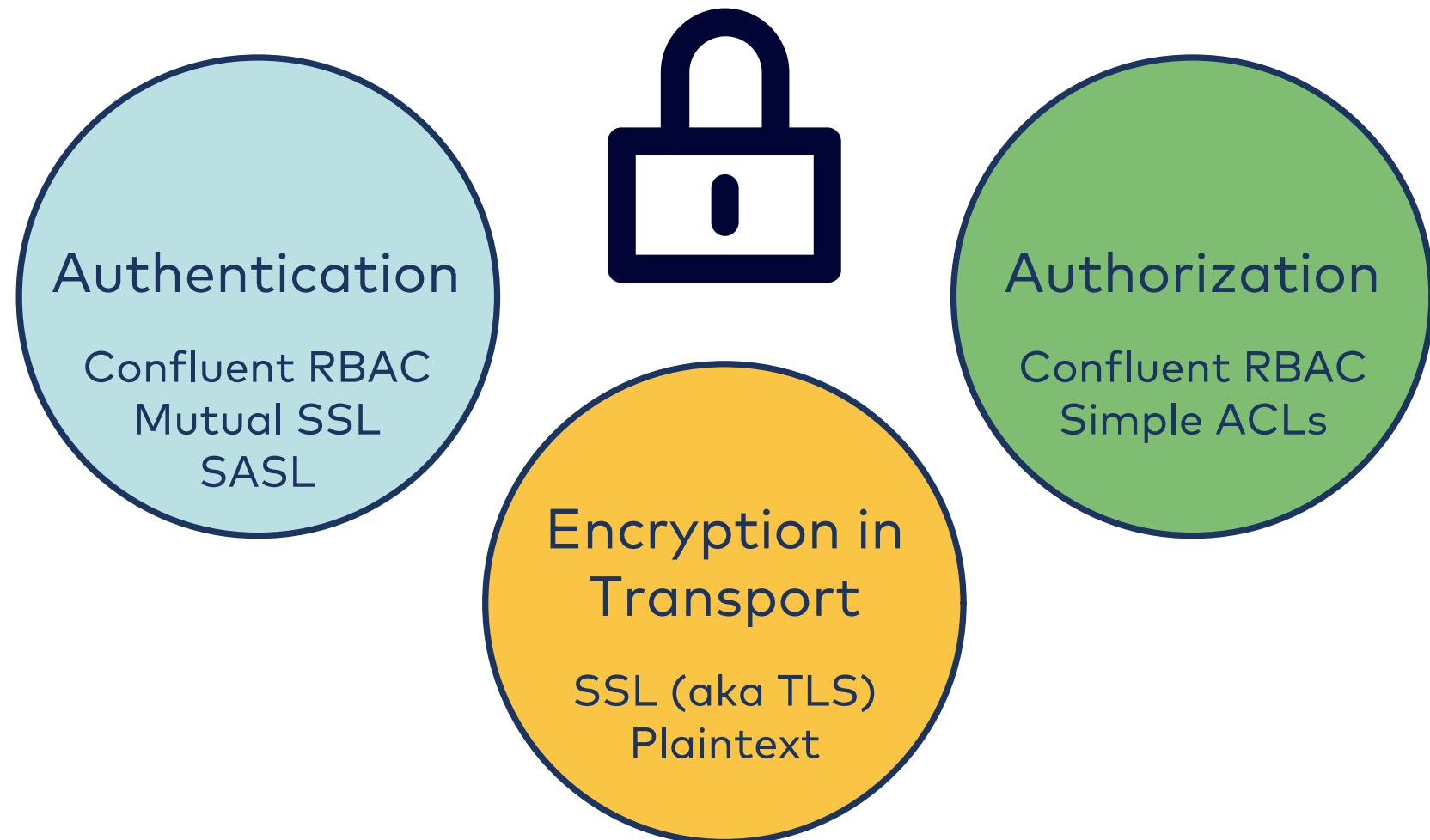
- Hard Prerequisite: Fundamentals Course
- Recommended Prerequisite: Managing a Kafka Cluster

## 9a: What are the Basic Ideas You Should Know about Kafka Security?

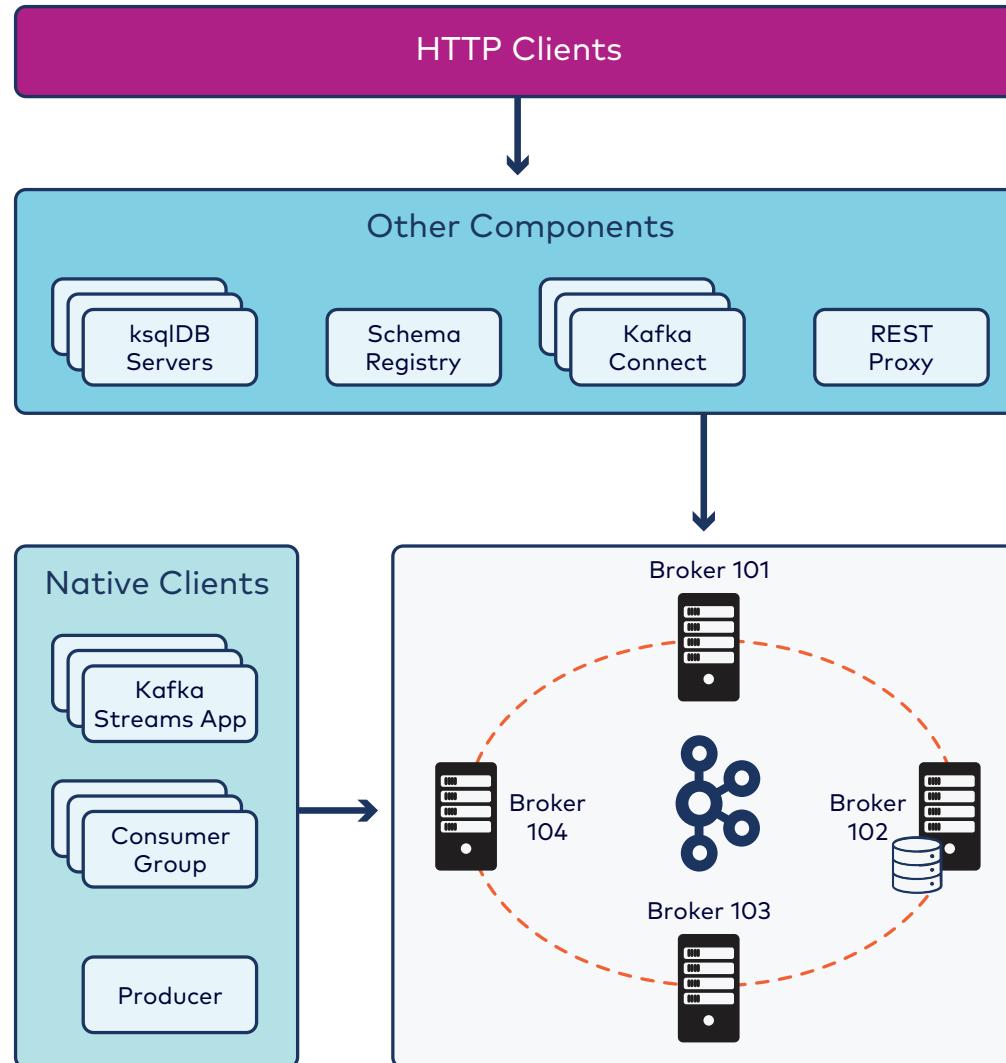
### Description

Overview of security in Kafka. Authentication vs. authorization. Encryption. Points of vulnerability.

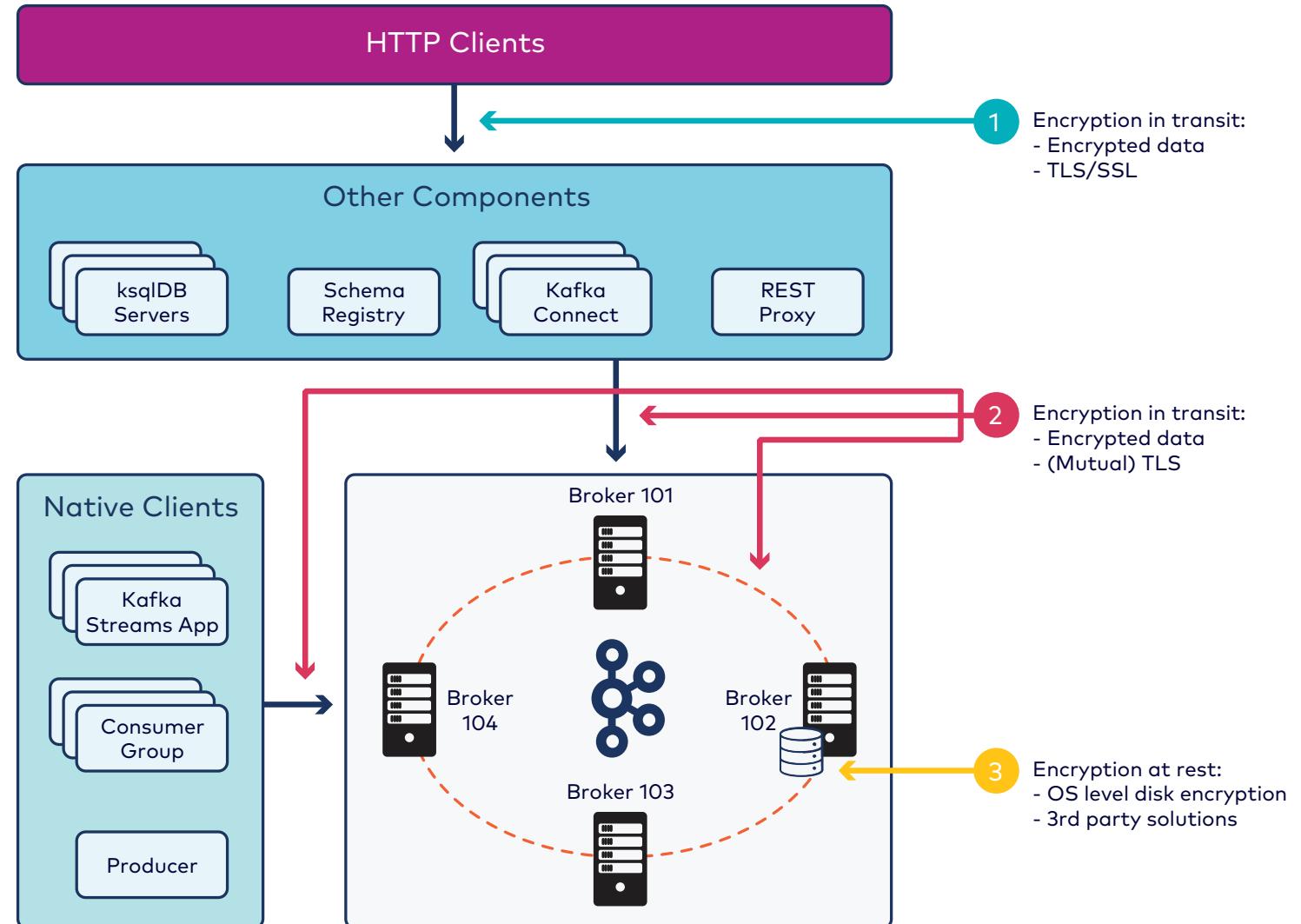
# Security Overview



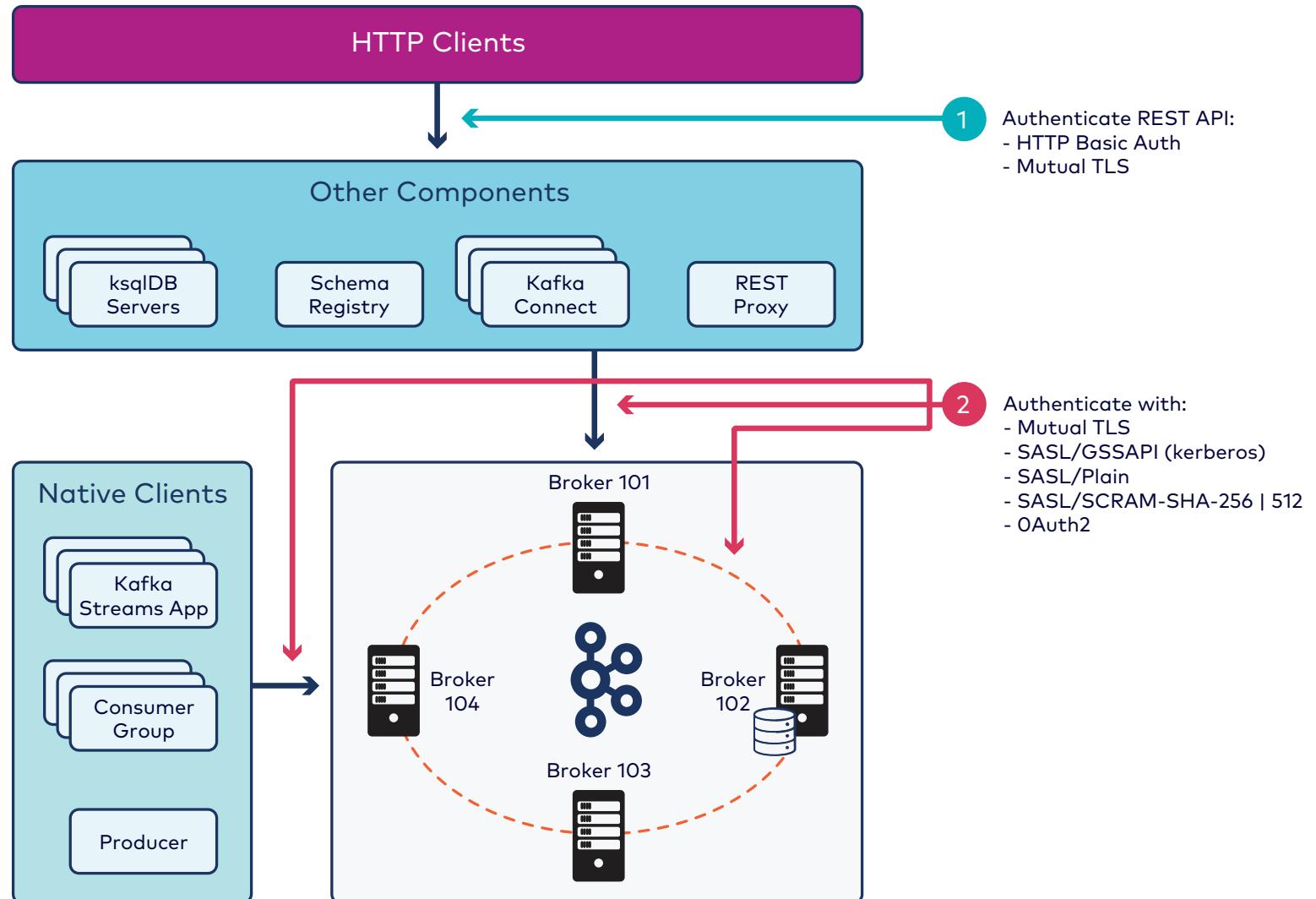
# Security - Architecture



# Security - Encryption at Rest & in Transit



# Authentication



## Broker Ports for Security

- Plain text (no wire encryption, no authentication)

```
listeners=PLAINTEXT://kafka-1:9092
```

- SSL (wire encryption, authentication)

```
listeners=SSL://kafka-1:9093
```

- SASL (authentication)

```
listeners=SASL_PLAINTEXT://kafka-1:9094
```

- SSL + SASL (SSL for wire encryption, SASL for authentication)

```
listeners=SASL_SSL://kafka-1:9095
```

- Clients choose **only one** port to use



Brokers may need to set up `advertised.listeners` in addition to `listeners` when the hostname/IP/port resolved by clients is different from those resolved by brokers themselves (e.g. NAT, aliasing, ...)

## An Advanced Listeners Config Example

We can configure different listeners for different sources of traffic

- Useful to designate one interface for clients and one interface for replication traffic:

```
listeners = CLIENTS://kafka-1a:9092,REPLICATION://kafka-1b:9093
```

- Listeners can have any name as long as `listener.security.protocol.map` is defined to map each name to a security protocol:

```
listener.security.protocol.map = CLIENTS:SASL_SSL, REPLICATION:SASL_PLAINTEXT
```

## 9b: What Options Do You Have For Securing a Kafka/Confluent Deployment?

### Description

Survey of security options.

## SSL/TLS or SASL Manual Configuration

- Free
- Based on configuration: no need to develop code
- But you have to do all the work
- You will experience this in lab.



See the appendix for examples and details regarding SSL/TLS and SASL.

## Example - Simple TLS Configuration

- `server.properties` on brokers:

```
listeners = SSL://<host>:<port>
inter.broker.listener.name = SSL
ssl.keystore.location = /var/private/ssl/kafka.server.keystore.jks
ssl.keystore.password = password-to-keystore-file
ssl.key.password = password-to-private-key
```

- Client `*.properties` Files:

```
ssl.truststore.location = /var/private/ssl/kafka.server.truststore.jks
ssl.truststore.password = password-to-truststore-file
security.protocol = SSL
```

## Confluent RBAC

- **Role Based Access Control**
- Paid Confluent feature
- Includes both authentication and authorization...
- ...by defined roles

## Example - Security on Confluent Cloud

- Security enabled **out-of-the-box**
- User → Cloud communication secured by TLS
- Data encrypted in motion & at rest
- CCloud is hosted in multiple AWS, GCP, and Azure regions

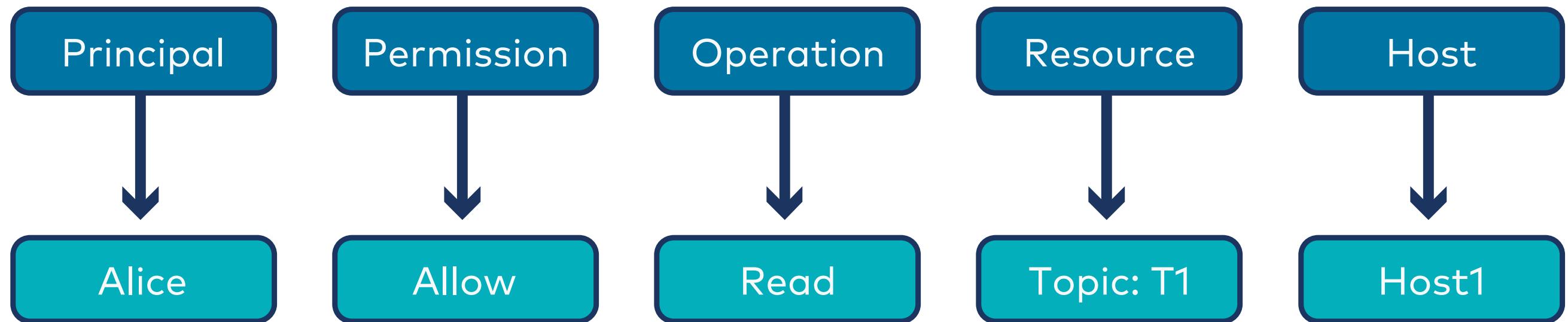
## 9c: How Can You Easily Control Who Can Access What?

### Description

Components of Kafka ACL entries. How to add and remove ACLs. Wildcards.

## Access Control Lists (ACLs)

ACL example: Alice is allowed to read data from topic `T1` from host `Host1`



## Principal

- Type + name
- Supported types: **User**
  - `User:Alice`
- Extensible, so users can add their own types (e.g., group)

## Permissions

- Allow and Deny
  - Deny takes precedence
  - Deny makes it easy to specify "everything but"
- By default, anyone without an explicit Allow ACL is denied

# Operations and Resources

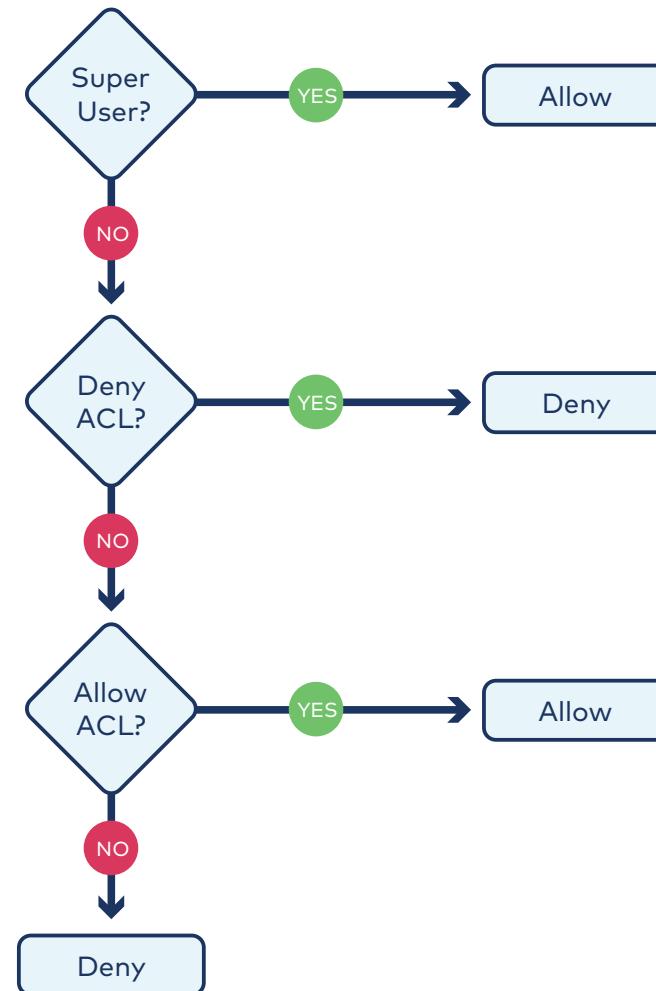
- Operations:
  - `Read, Write, Create, Describe, ClusterAction, All`
- Resources:
  - `Topic, Cluster, and ConsumerGroup`

Operations	Resources
<code>Read, Write, Describe</code>	<code>Topic</code>
<code>Read and Write imply Describe</code>	
<code>Read</code>	<code>ConsumerGroup</code>
<code>Create, ClusterAction</code> intra-cluster operations (leader election, replication, etc.)	<code>Cluster</code>

## Hosts

- Allows firewall-type security, even in a non-secure environment
  - Without needing system/network administrators to get involved

# Permission Check Sequence



## Configuring ACLs - Producers

- `kafka-acls` can be used to add authorization
- Producer:
  - Grant `Write` on the topic, `Create` on the Cluster (for topic auto-creation)
  - Or use `--producer` option in the CLI

```
$ kafka-acls \
  --bootstrap-server kafka-1:9092 \
  --add \
  --allow-principal User:Bob \
  --producer \
  --topic my_topic
```

## Configuring ACLs - Consumers

- Consumer:
  - Grant `Read` on the topic, `Read` on the ConsumerGroup
  - Or use the `--consumer` option in the CLI

```
$ kafka-acls \
  --bootstrap-server kafka-1:9092 \
  --add \
  --allow-principal User:Bob \
  --consumer \
  --topic my_topic \
  --group group1
```

## Removing Authorization

- `kafka-acls` can be used to remove or change authorization
  - May use additional options, e.g., `deny-principal`, `remove`, etc
- If needed, also useful to revoke authorization after connections are established
  - SSL and SASL authentication happens only once during the connection initialization process
  - Since no re-authentication occurs after connections are established:
    - Use `kafka-acls` to remove all permissions for a principal
    - All requests on that connection will be rejected
    - Reset connections as needed

## Wildcard Support (1)

- Allow user **Jane** to produce to any topic whose name starts with "Test-"

```
$ kafka-acls \
--bootstrap-server kafka-1:9092 \
--add \
--allow-principal User:Jane \
--producer --topic Test- \
--resource-pattern-type prefixed
```

- Allow all users **except** **BadBob** and all hosts **except** **198.51.100.3** to read from **Test-topic**:

```
$ kafka-acls \
--bootstrap-server kafka-1:9092 --add \
--allow-principal User:'*' \
--allow-host '*' \
--deny-principal User:BadBob \
--deny-host 198.51.100.3 \
--operation Read --topic Test-topic
```

## Wildcard Support (2)

- List all ACLs for the topic `Test-topic`:

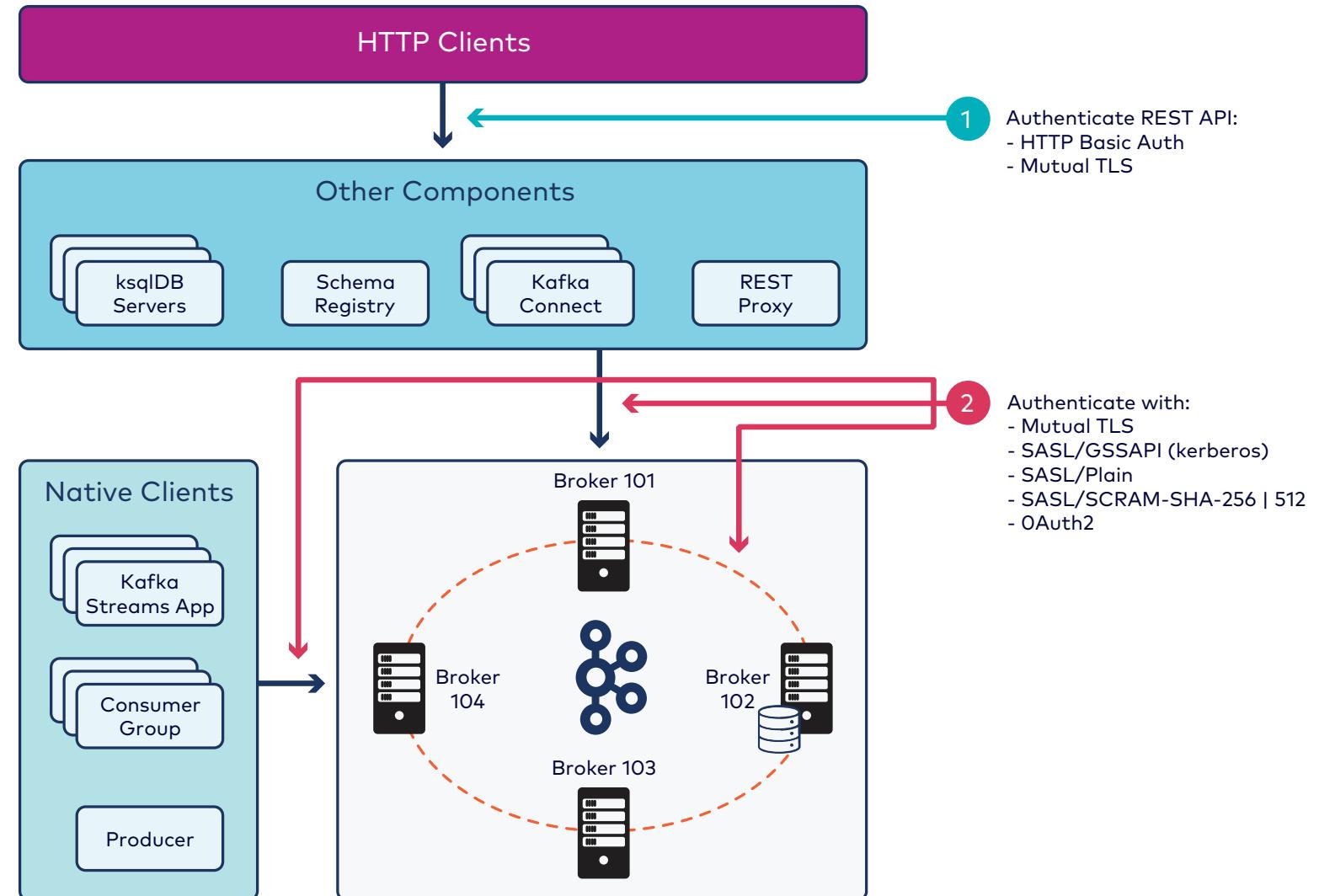
```
$ kafka-acls \
--bootstrap-server kafka-1:9092,kafka-2:9092 \
--list --topic Test-topic \
--resource-pattern-type match
```

## 9d: What Should You Know Securing a Deployment Beyond Kafka Itself?

### Description

Securing the whole environment.

# Securing Schema Registry and REST Proxy



## Securing the Schema Registry

- Secure communication between REST client and Schema Registry (HTTPS):
  - HTTP Basic Authentication
  - SSL (transport)
- Secure transport and authentication between the Schema Registry and the Kafka cluster:
  - SSL (transport)
  - SASL (authentication)
  - Mutual SSL (transport + authentication)
- Confluent Enterprise **security plugin**:
  - Restricts schema evolution to administrative users
  - Client application users get read-only access

## Securing the REST Proxy

- Secure communication between REST clients and the REST Proxy (HTTPS)
  - HTTP Basic Authentication
  - SSL (transport)
- Secure communication between the REST Proxy and Apache Kafka
  - SSL (transport)
  - SASL (authentication)
  - Mutual SSL (transport + authentication)
- Confluent Enterprise **security plugin**:
  - Propagates client principal authentication to Kafka brokers
  - More granular than single authentication for all clients

# Migrating Non-Secure to Secure Kafka Cluster

## 1. Configure brokers with multiple ports

```
listeners=PLAINTEXT://host.name:port,SSL://host.name:port
```

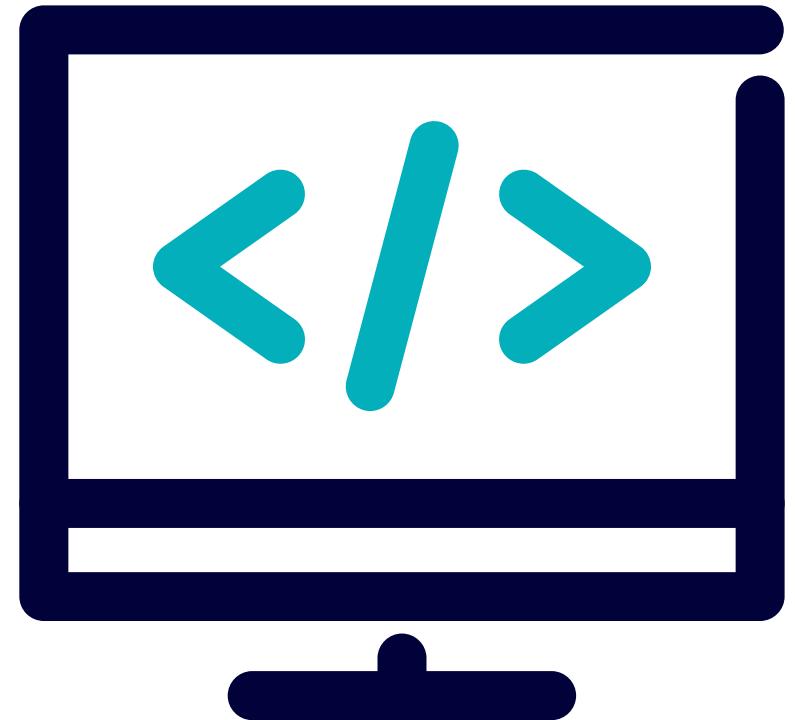
## 2. Gradually migrate clients to the secure port

## 3. When done, turn off **PLAINTEXT** listener on all Brokers

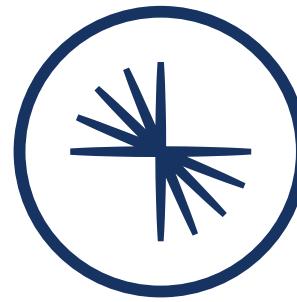
# Lab: Securing the Kafka Cluster

Please work on **Lab 9a: Securing the Kafka Cluster**

Refer to the Exercise Guide



# 10: Understanding Kafka Connect



CONFLUENT  
**Global Education**

# Module Overview



This module contains 4 lessons:

- What Can You Do with Kafka Connect?
- How Do You Configure Workers and Connectors?
- Deep Dive into a Connector & Finding Connectors
- What Else Can One Do With Connect?

Where this fits in:

- Hard Prerequisite: Fundamentals Course
- Recommended Prerequisite: Consumer Groups and Load Balancing

## 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: Data From Another System in Kafka; Kafka Data To Another System

Suppose you have

- Some data in some other system and you want to get it into Kafka
- Some data in Kafka and want to export it to another system

Your development team could program custom producers or consumers with hooks into the other system to make this happen...

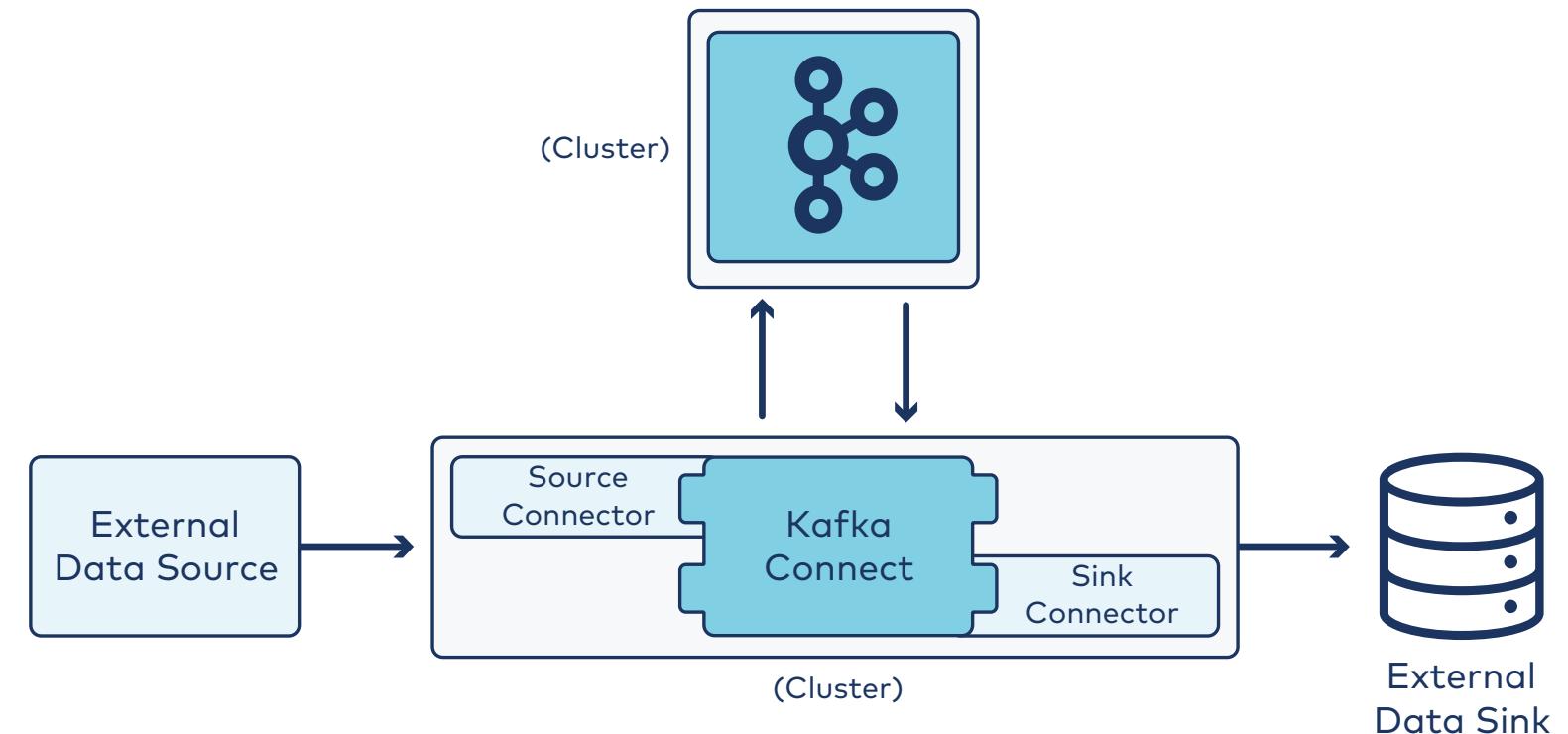
But... there's a better way...

# Kafka Connect to the Rescue!

Kafka Connect does the work for us!

All copying behavior is in Kafka Connect.

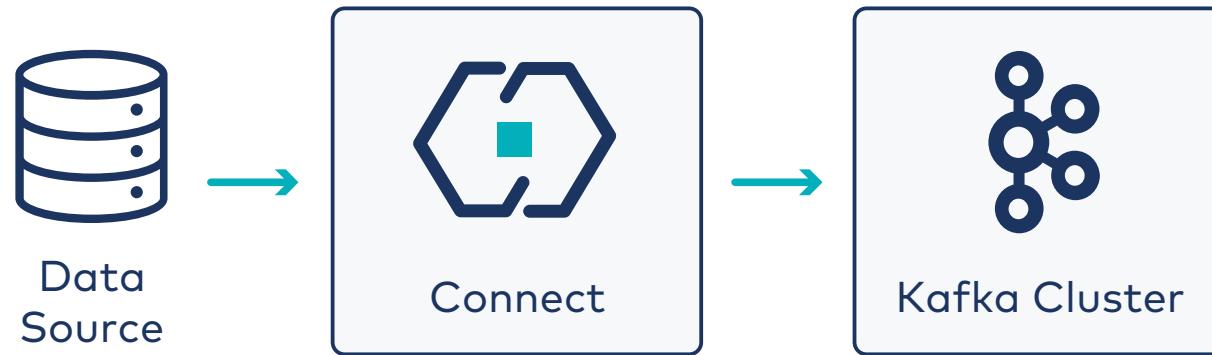
Plugins called **Connectors** contain the logic specific to particular external systems.



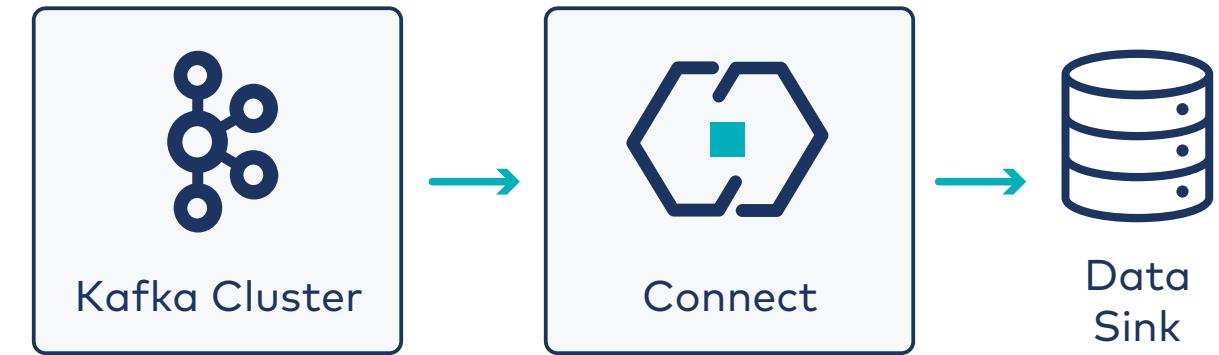
# Sources and Sinks

Two kinds of connectors...

## Source Connector



## Sink Connector



Uses producer API under the hood

Uses consumer API under the hood

# Players in the Kafka Connect World

## Kafka Connect

logic for copying regardless of the external system

## Connector

logic for copying specific to/from a given external system

## Task

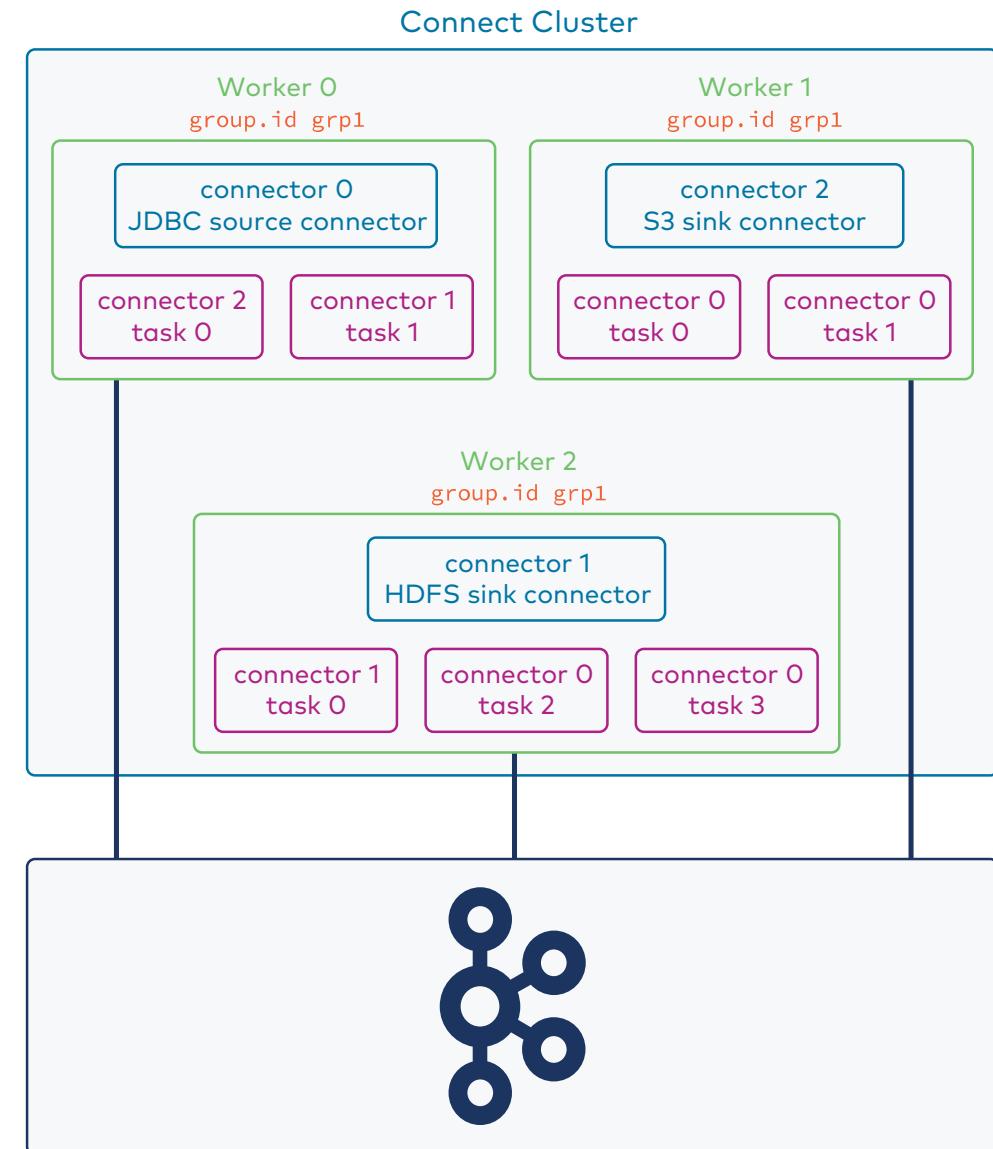
unit of parallelism into which connector copying logic is broken up

## Worker

process that runs connectors and/or tasks

- A connector has
  - one or more tasks
- A worker runs
  - zero or more connectors
  - zero or more tasks

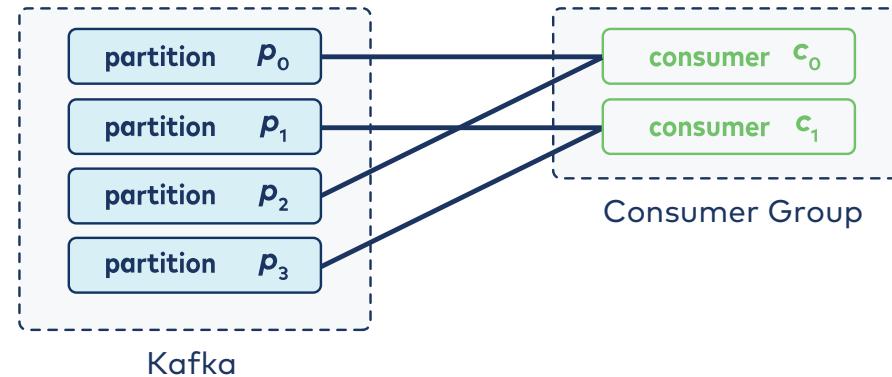
# Example of a Connect Cluster



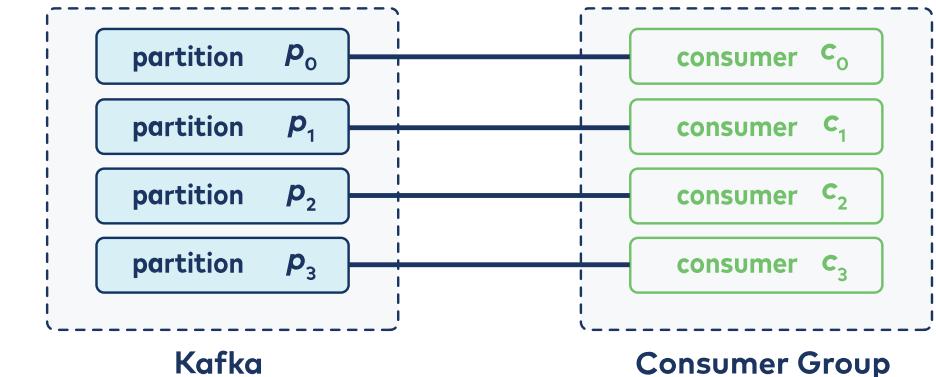
# Groups Again!

## Consumer Group

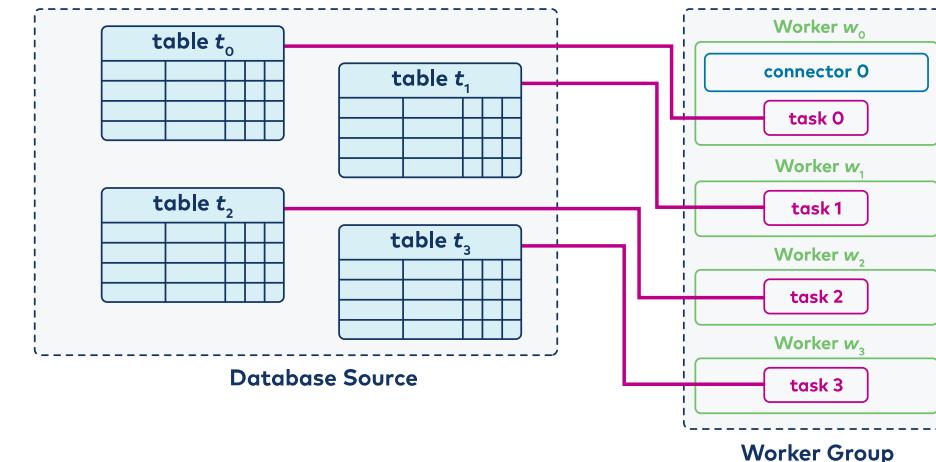
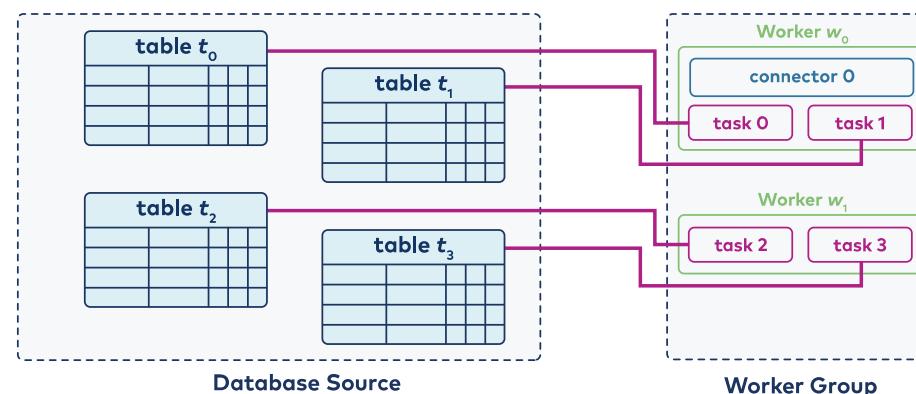
### Before Scaling



### After Scaling



## Worker Group

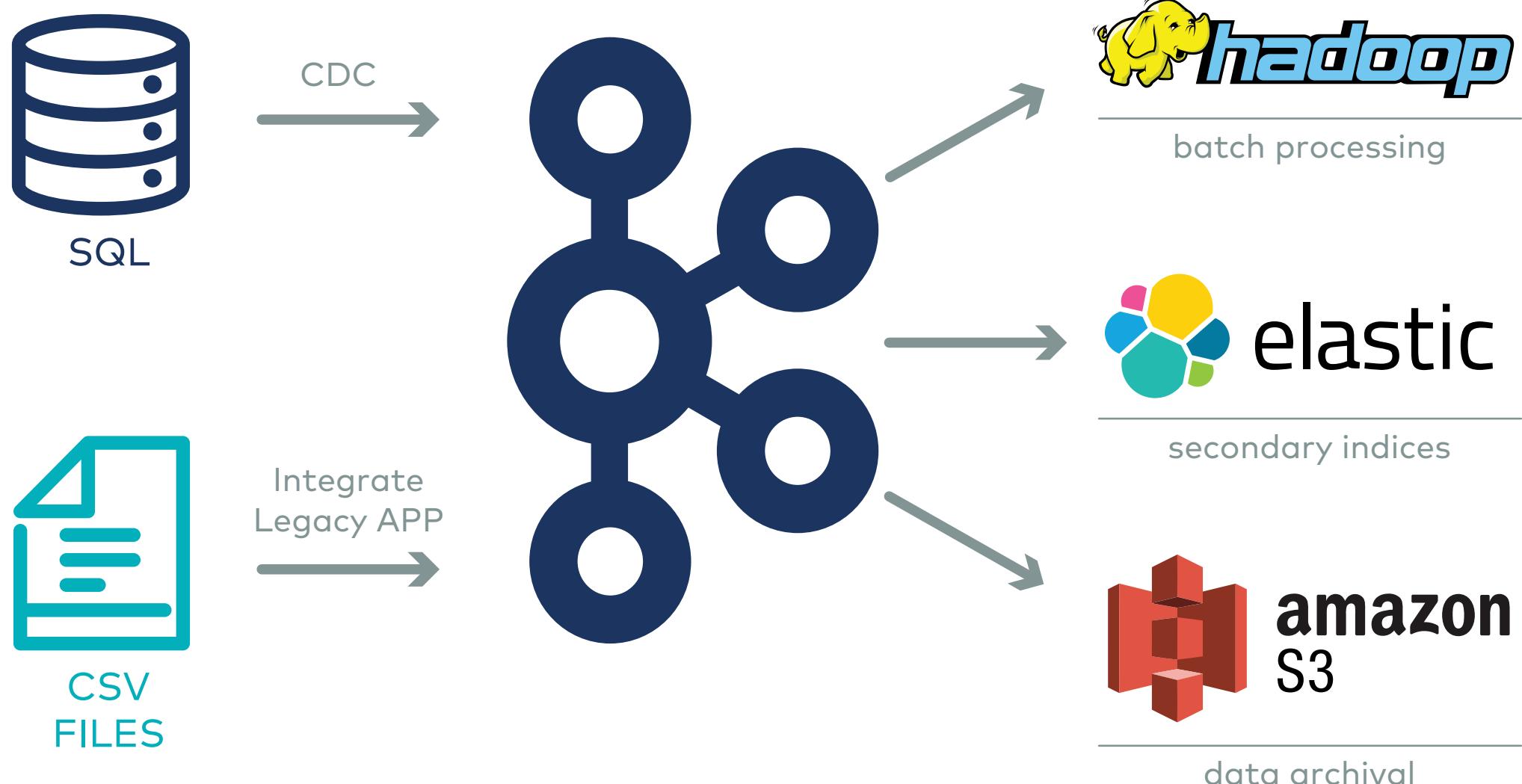


Like with consumers, we can add workers to groups and get automatic balancing.

## 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 <b>sink connectors only</b> , comma-separated list of topics to consume from	

## Configuring a Worker

Name	Description	Default
bootstrap.servers	List of host:port pairs to connect	
group.id	Identifier of what group this worker is a member of	
heartbeat.interval.ms	How frequently heartbeats are sent	3 sec.
session.timeout.ms	Time after which a worker that does not heartbeat is deemed dead	10 sec.
key.converter	Converter to (de)serialize keys	
value.converter	Converter to (de)serialize values	
topic.creation.enable	Whether source connectors are permitted to create topics	true

## 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.

## Query Mode (1)

Incremental query mode	Description
Bulk	Load all rows in the table. Does not detect new or updated rows.

## Query Mode (2)

Incremental query mode	Description
Incrementing column	Check a single column where newer rows have a larger, auto-incremented ID. Does not capture updates to existing rows.
Timestamp column	Checks a single 'last modified' column to capture new rows and updates to existing rows. If task crashes before all rows with the same timestamp have been processed, some updates may be lost.
Timestamp and incrementing column	Detects new rows and updates to existing rows with fault tolerance. Uses timestamp column, but reprocesses current timestamp upon task failure. Incrementing column then prevents duplicate processing.

## Query Mode: Custom Query

We can also define a **custom query** to use in conjunction with the previous options for custom filtering.

## 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](https://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.

Confluent Hub

Discover Kafka® connectors  
and more

What plugin are you looking for?

Filters

Plugin type

Sink

Source

Transform

Converter

Enterprise support

Confluent supported

Partner supported

None

Verification

Confluent built

Confluent Tested

Verified gold

Verified standard

None

License

Commercial

Free

Results (158)

+ Submit a plugin

**Kafka Connect GCP Pub-Sub**

A Kafka Connect plugin for GCP Pub-Sub

Available fully-managed on Confluent Cloud

Enterprise support: Confluent supported

Installation: Confluent Hub CLI, Download

Verification: Confluent built

Author: Confluent, Inc.

License: Commercial

Version: 1.0.2

**Kafka Connect S3**

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

Enterprise support: Confluent supported

Installation: Confluent Hub CLI, Download

Verification: Confluent built

Author: Confluent, Inc.

License: Free

Version: 5.5.1

## 10d: What Else Can One Do With Connect?

### 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.

## Development Power

Kafka Connect can be leveraged along with other tools to solve various development problems.

Common classes of problems include:

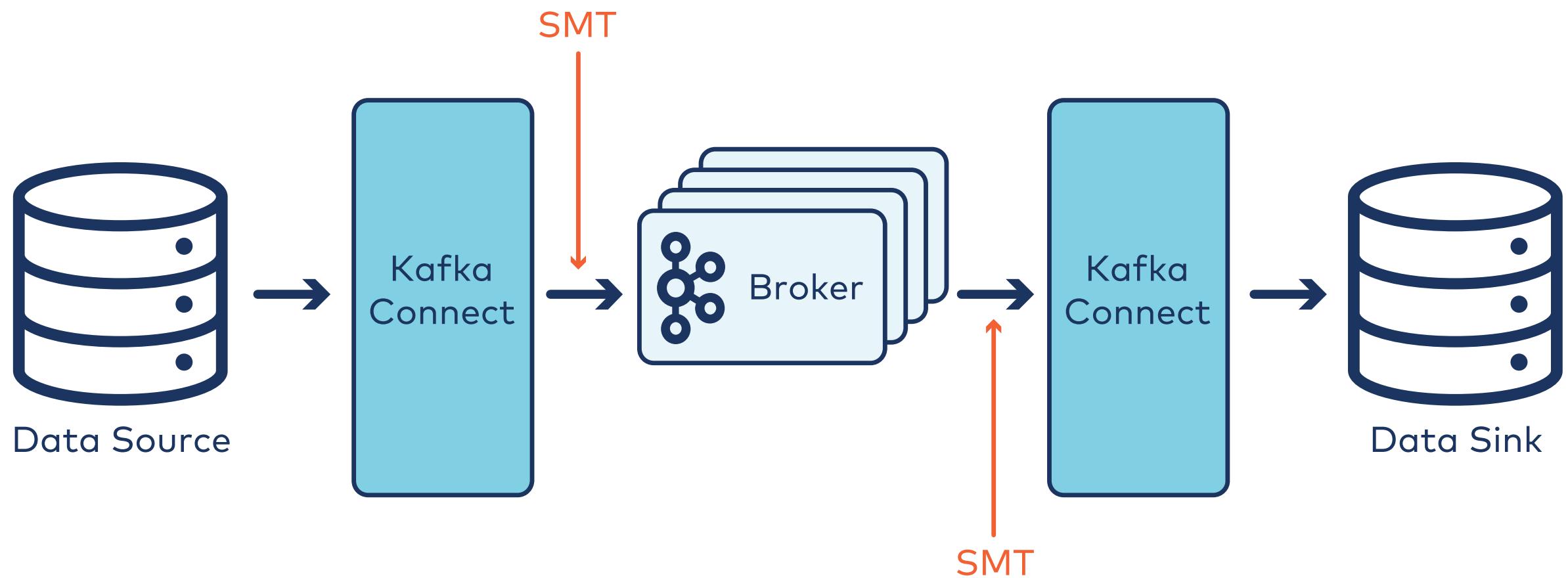
- **ETL** = Extract, Transform, Load
- **CDC** = Change Data Capture

In both cases, developers can leverage Kafka Connect to get data into Kafka.

The "transform" may be achieved either via:

- Streaming applications
  - Kafka Streams
  - ksqlDB
- Single Message Transforms...

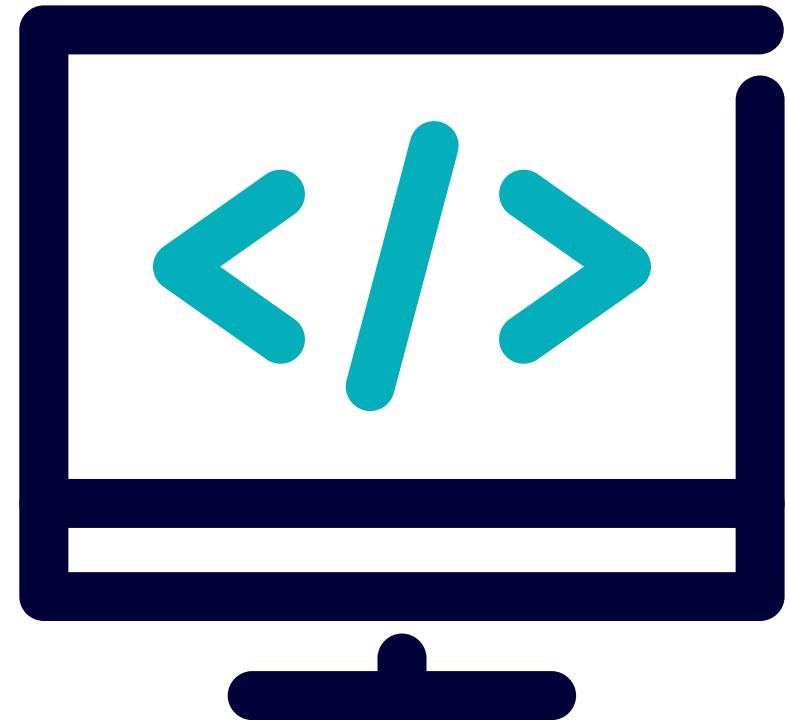
# Single Message Transforms



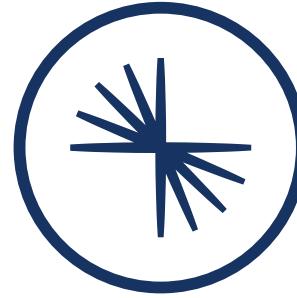
# Lab: Running Kafka Connect

Please work on **Lab 10a: Running Kafka Connect**

Refer to the Exercise Guide



# 11: Deploying Kafka in Production



CONFLUENT  
**Global Education**

# Module Overview



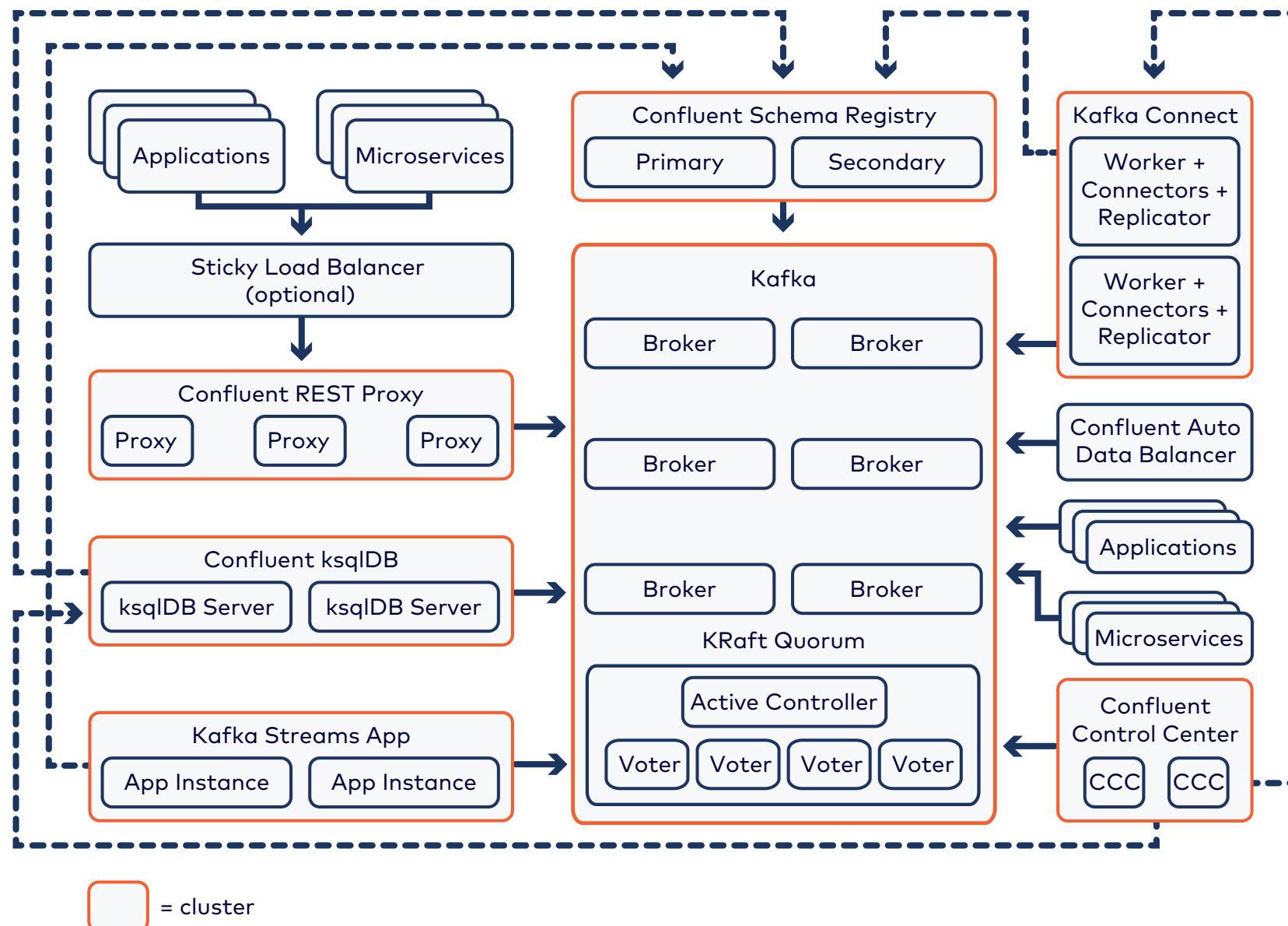
This module contains 6 lessons. Each lesson enables you to answer what Confluent advises for deploying each of the following in production:

- Kafka
- Kafka Connect
- Confluent Schema Registry
- Confluent REST Proxy
- Kafka Streams and ksqlDB
- Confluent Control Center

Where this fits in:

- Hard Prerequisite: All modules of this Administration course

# Confluent's Reference Architecture for Kafka

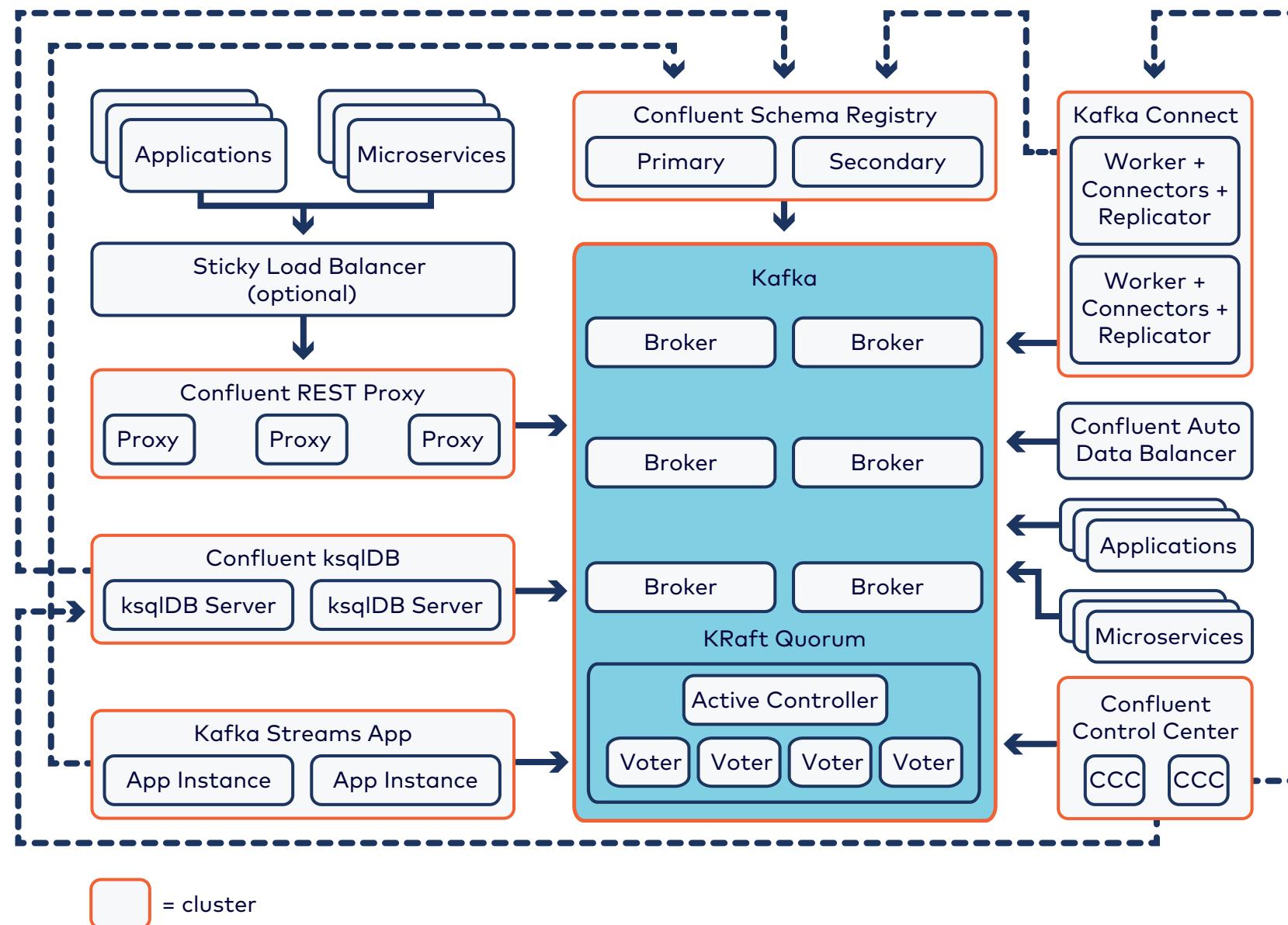


# 11a: What Does Confluent Advise for Deploying Brokers in Production?

## Description

Best practices and capacity planning for brokers in production.

# Reference Architecture: Kafka Nodes



## Broker Design

- Run on dedicated servers
  - 5 controllers
    - In multi-DC / multi-AZ deployment, distribute controllers
  - $n$  brokers → replication factor up to  $n$
  - Can use virtual IP + load balancer as `bootstrap.servers`
    - Pro: don't have to change client config code when cluster changes
    - Con: More infrastructure to worry about
- Discussion Questions:
    1. What replication factor is acceptable for mission-critical data?
    2. How many brokers do you need to accommodate this?
    3. With that many brokers, how many can fail without permanent data loss? What would happen to write access and read access?

## Capacity Planning: Brokers

- Disk space and I/O
- Network bandwidth
- RAM (for page cache)
- CPU

## Broker Disk

- 12 x 1TB filesystems mounted to data directories for topic data + separate disks for OS
  - A single partition can only live on **one** volume
  - Partitions are balanced across `log.dirs` in round-robin
  - RAID-10 optional
- Use XFS or EXT4 filesystems
  - Mount with `noatime`

## Network Bandwidth

- Gigabit Ethernet sufficient for smaller applications
- 10Gb Ethernet needed for large installations
- Enable compression on producers
- Optional: Isolate Inter-Broker traffic to separate network

## Broker Memory

- JVM heap ~ 6 GB
- OS ~ 1 GB
- Page cache:
  - Lots and lots!
  - What might your consumer lag be?

## Tuning the Java Heap (1)

- Java heap memory is allocated for storing Java objects
- By default `kafka-server-start` configures heap size to 1 GB
  - `Xmx`: maximum Java heap size
  - `Xms`: start Java heap size

```
$ export KAFKA_HEAP_OPTS="-Xms6g -Xmx6g"
```

- Recommended JVM performance tuning options:

```
-Xms6g -Xmx6g -XX:MetaspaceSize=96m -XX:+UseG1GC -XX:MaxGCPauseMillis=20  
-XX:InitiatingHeapOccupancyPercent=35 -XX:G1HeapRegionSize=16M  
-XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=80
```

## Tuning the Java Heap (2)

- Suggested formula for determining the broker's heap size:

```
(message.max.bytes * num Partitions per Broker) + log.cleaner.dedupe.buffer.size + 500MB
```

- Property defaults
  - `message.max.bytes` default is 1MB
  - `log.cleaner.dedupe.buffer.size` default is 128MB
- Consider tuning the Java heap size

Java Heap Size	Deployment Type
1 GB (default)	Testing and small production deployments
6 GB	Typical production deployments
12 GB+	Deployments with very large messages or many partitions per broker

## Tuning Virtual Memory Settings

- Minimize memory swapping:
  - `vm.swappiness=1` (Default: 60)
- Decrease the frequency of blocking flushes (synchronous)
  - `vm.dirty_ratio=80` (Default: 20)
- Increase the frequency of non-blocking background flushes (asynchronous)
  - `vm.dirty_background_ratio=5` (Default: 10)



Set these parameters in `/etc/sysctl.conf` and load with `sysctl -p`

## Open File Descriptors and Client Connections

- Brokers can have a **lot** of open files
  - `$ ulimit -n 100000`
- Brokers can have a **lot** of client connections:
  - `max.connections.per.ip` (Default: 2 billion)

## Broker CPU

- Dual 12-core sockets
- Relevant broker properties:
  - `num.io.threads` (Default: 8)
  - `num.network.threads` (Default: 3)
  - `num.recovery.threads.per.data.dir`  
(Default: 1)
  - `background.threads` (Default: 10)
  - `num.replica.fetchers` (Default: 1)
  - `log.cleaner.threads` (Default: 1)
- Discussion:
  - Given 12 data disks and dual 12-core CPU sockets, how would you modify the default broker threading properties?

## Capacity Planning: Number of Brokers

- Storage

Number of brokers =

(messages per day \* message size \* Retention days \* Replication) / (disk space per Broker)

- Network bandwidth

Number of brokers =

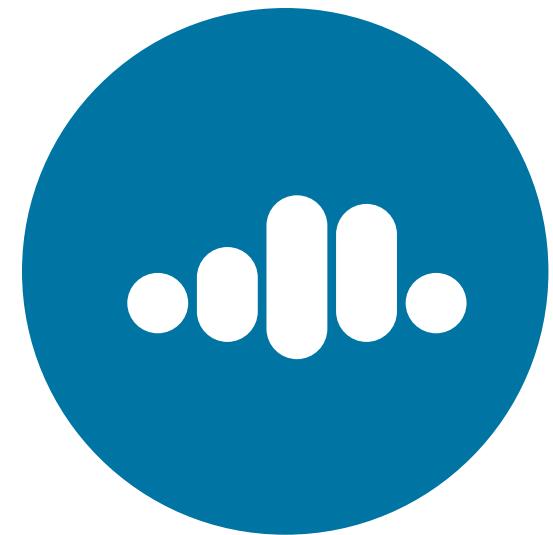
(messages per sec \* message size \* Number of Consumers) / (Network bandwidth per Broker)



Recommended limits: 4,000 partitions per broker and 200,000 partitions per cluster.

## Deploying Kafka in the Cloud

- Self-managed cloud deployment:
  - Memory optimized compute instances
  - Multiple availability zones (`broker.rack`)
  - Private subnet for inter-broker traffic
  - Lockdown firewall rules, Kafka security
  - For AWS: "EBS optimized" instances
- Or consider:



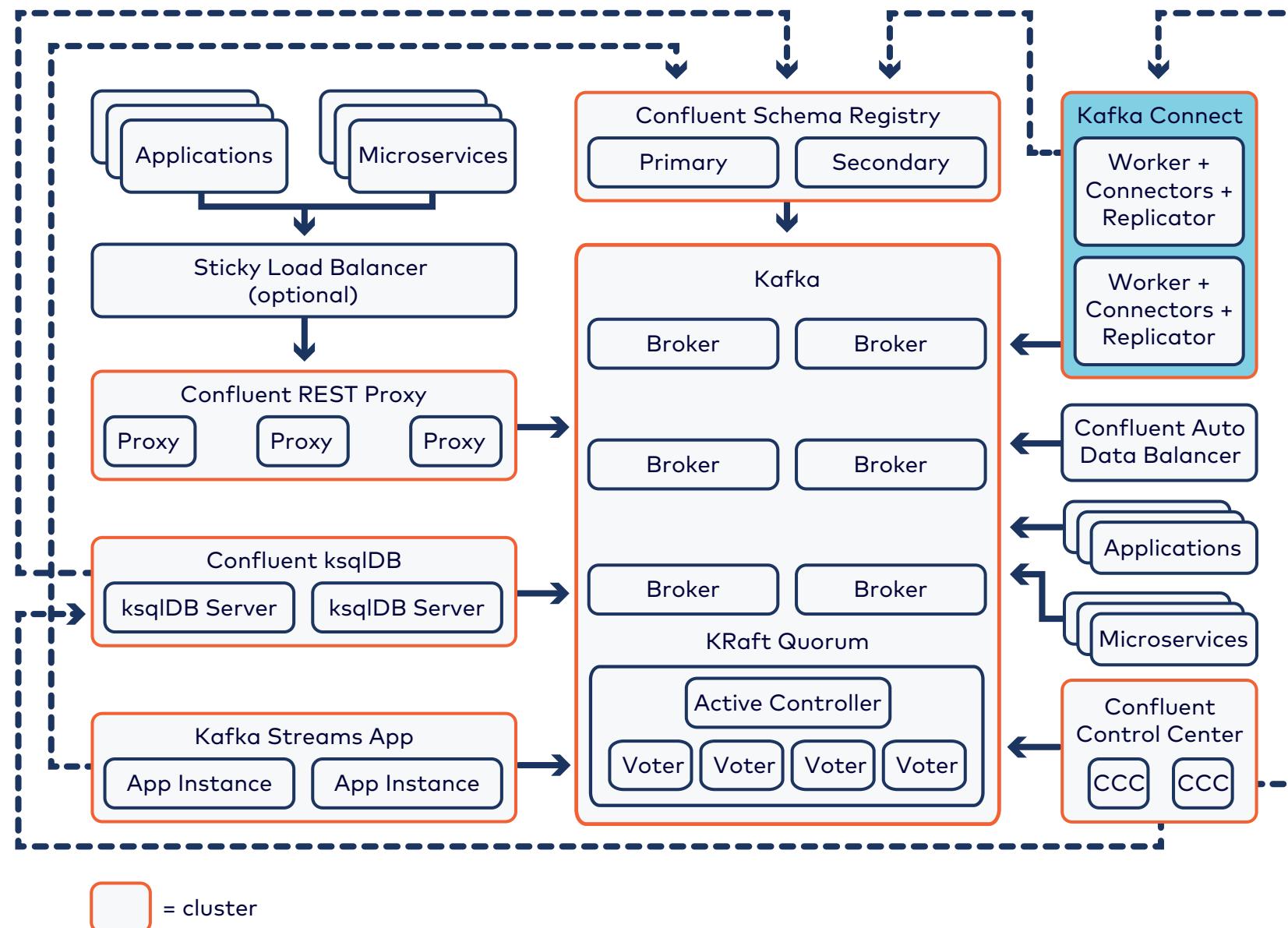
Virtual cores are weaker than physical cores

# 11b: What Does Confluent Advise for Deploying Kafka Connect in Production?

## Description

Best practices and capacity planning for Kafka Connect in production.

# Reference Architecture: Kafka Connect



## Connect Workers for High Availability

- Deploy machines with same `group.id` to form cluster
- Deploy at least 2 machines behind load balancer
- Add machines with same `group.id` to add capacity

## Tune `tasks.max` for Scalability

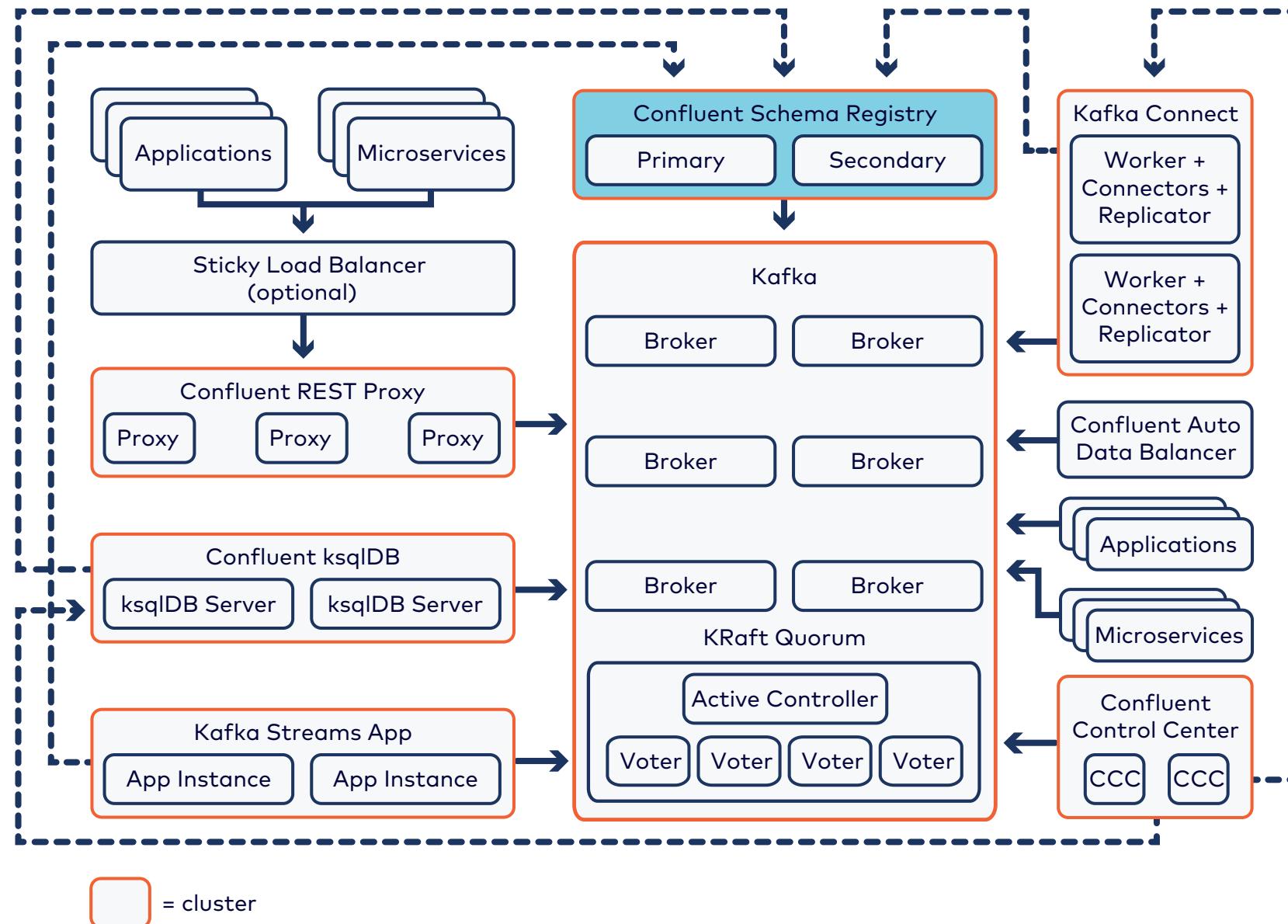
- `tasks.max` is a config sent to Connect via REST API
- Set to minimum of:
  - Number topic-partitions (for sink connectors)
  - Desired throughput / Throughput per task
  - Machines \* Number of cores per machine
- The more cores in the Connect cluster, the better

## 11c: What Does Confluent Advise for Deploying Schema Registry in Production?

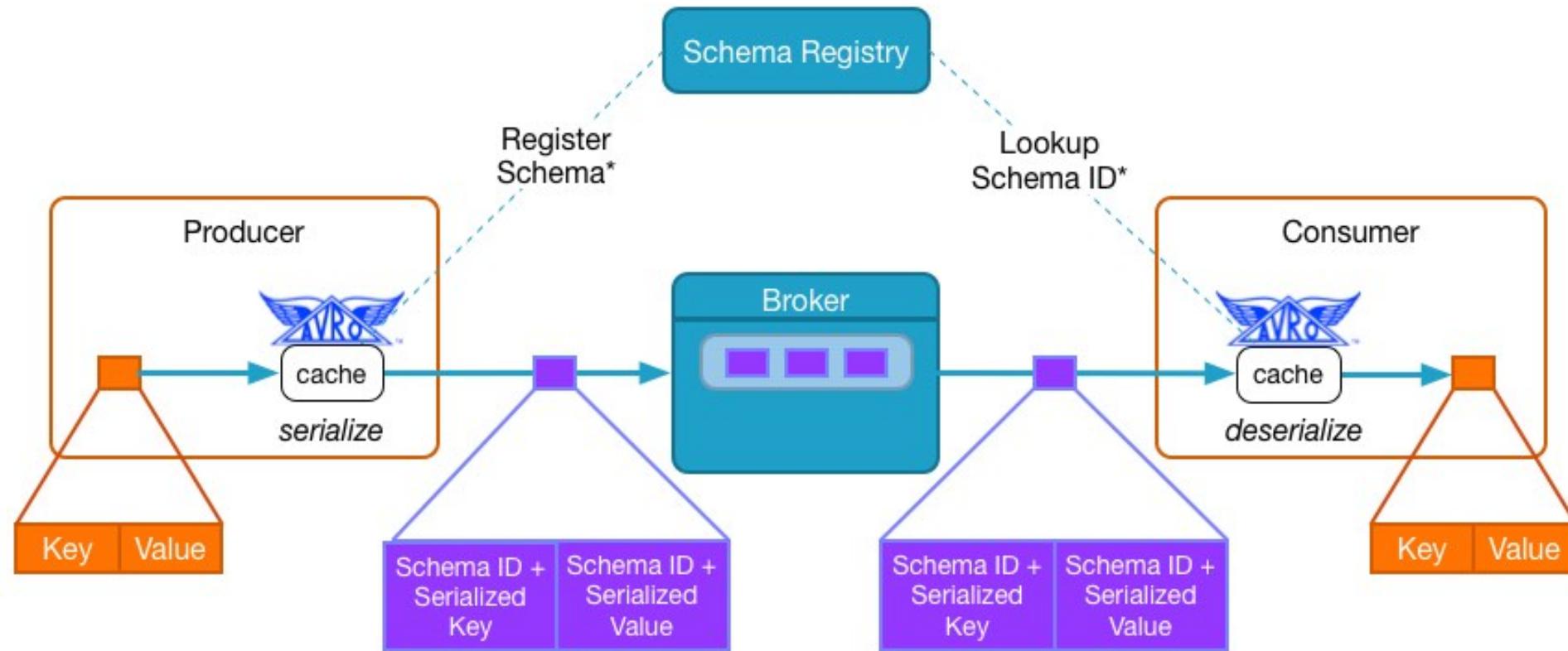
### Description

Best practices and capacity planning for Schema Registry in production.

# Reference Architecture: Confluent Schema Registry



# Schema Registry Overview



## Capacity Planning: Schema Registry

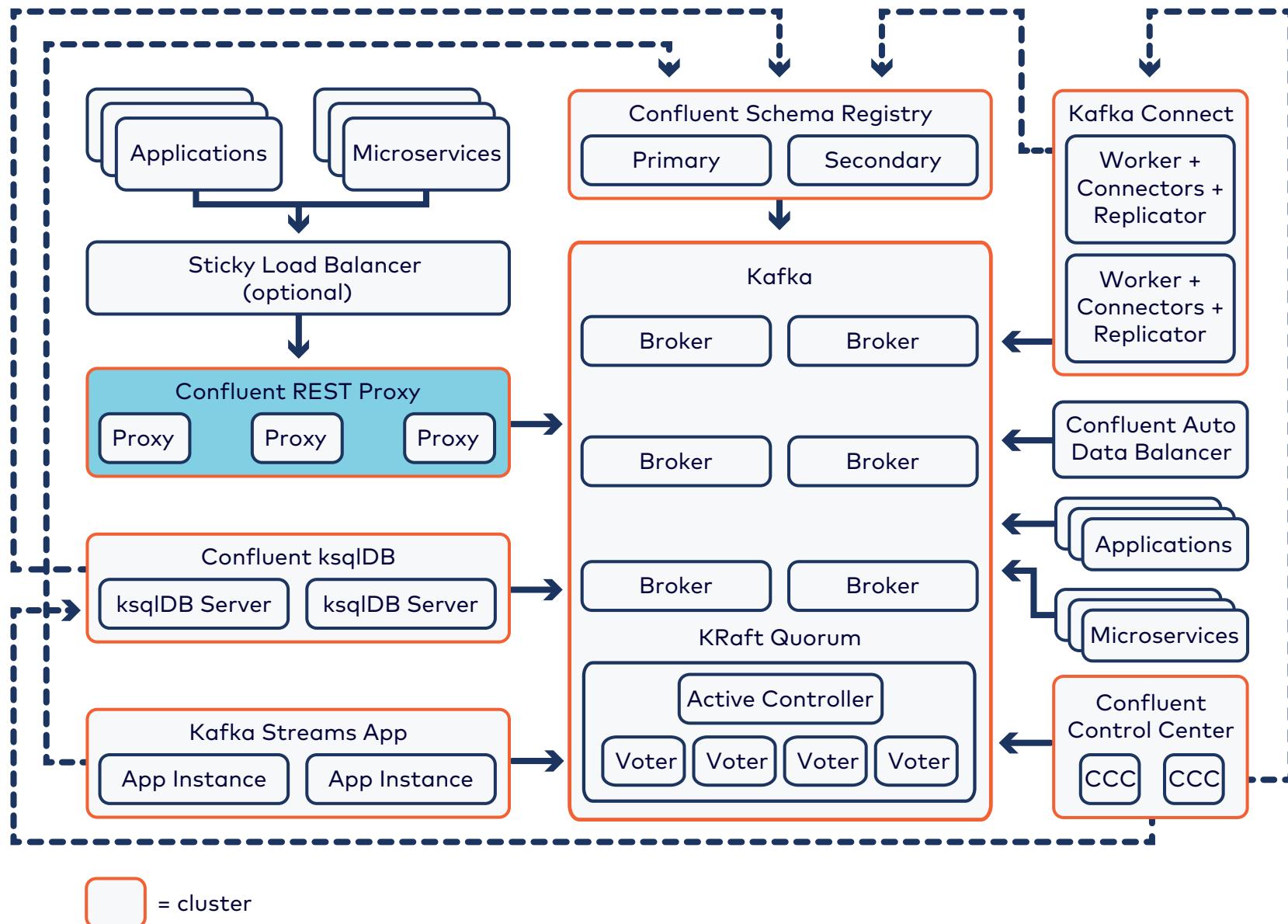
- Minimal system requirements
- Deploy 2+ servers behind load balancer
- Single-primary architecture
  - One primary node at a time
  - Primary node responds to write requests
  - Secondary nodes forward write requests to primary node
  - All nodes respond to read requests

## 11d: What Does Confluent Advise for Deploying the REST Proxy in Production?

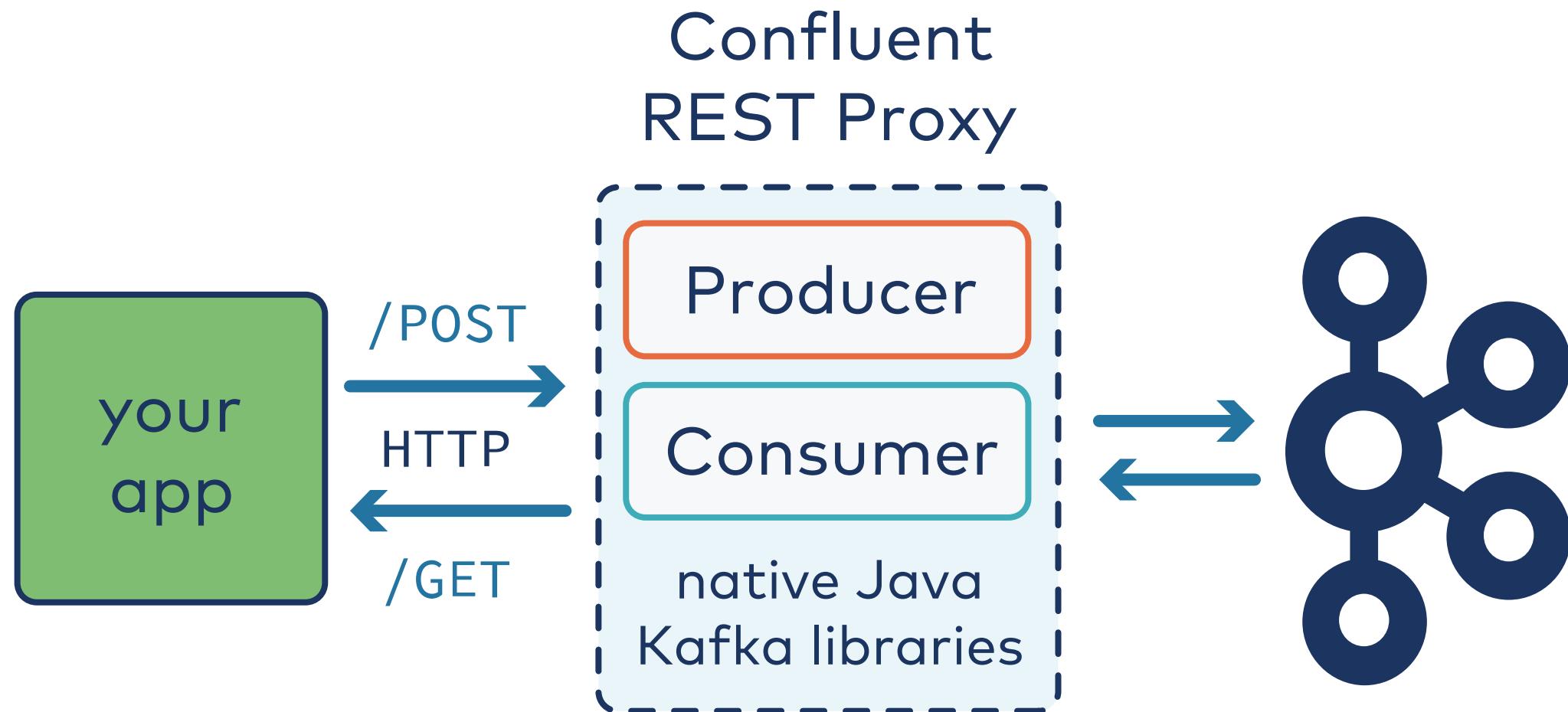
### Description

Best practices and capacity planning for the REST Proxy in production.

# Reference Architecture: Confluent REST Proxy



# What is the Confluent REST Proxy?

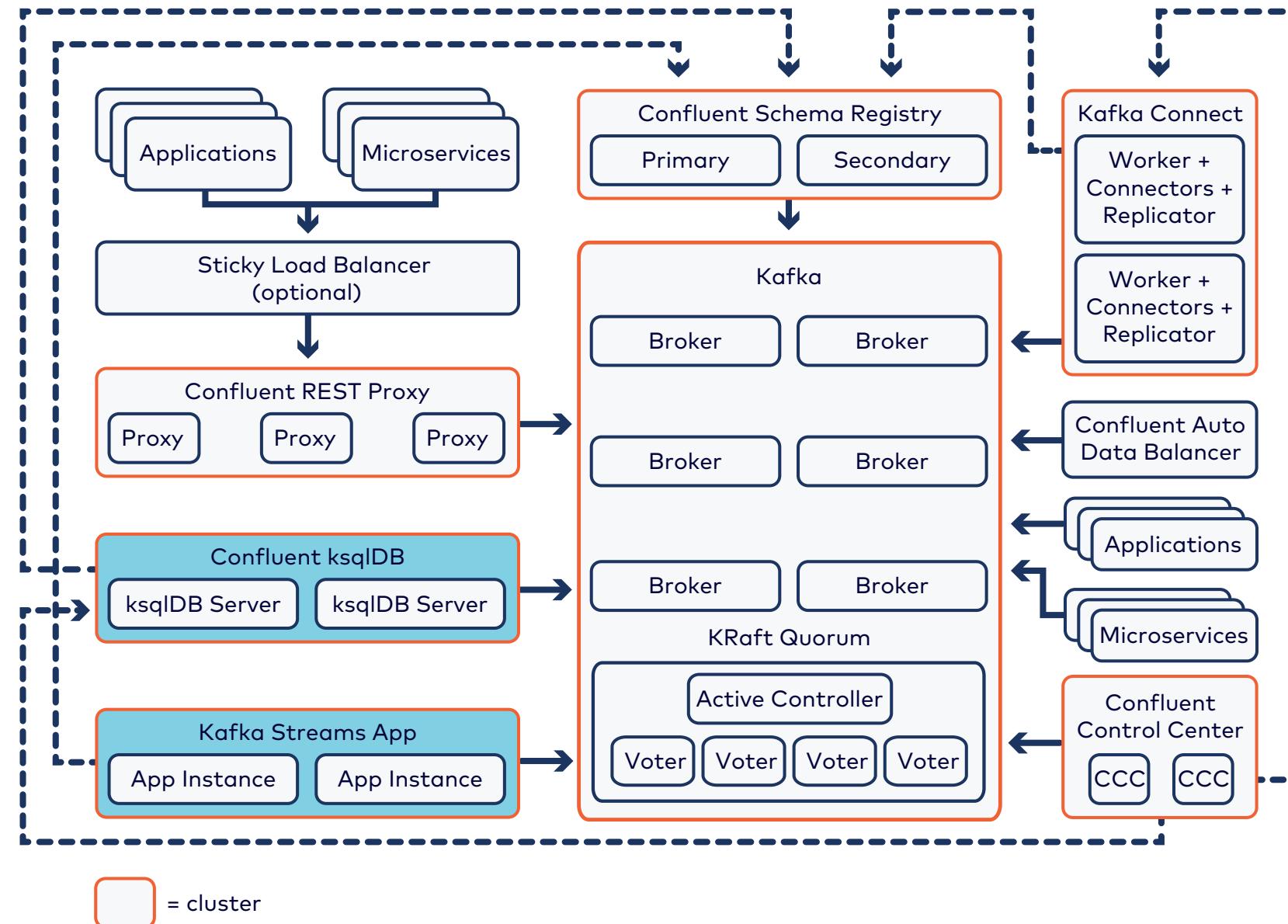


# 11e: What Does Confluent Advise for Deploying Kafka Streams and ksqlDB in Production?

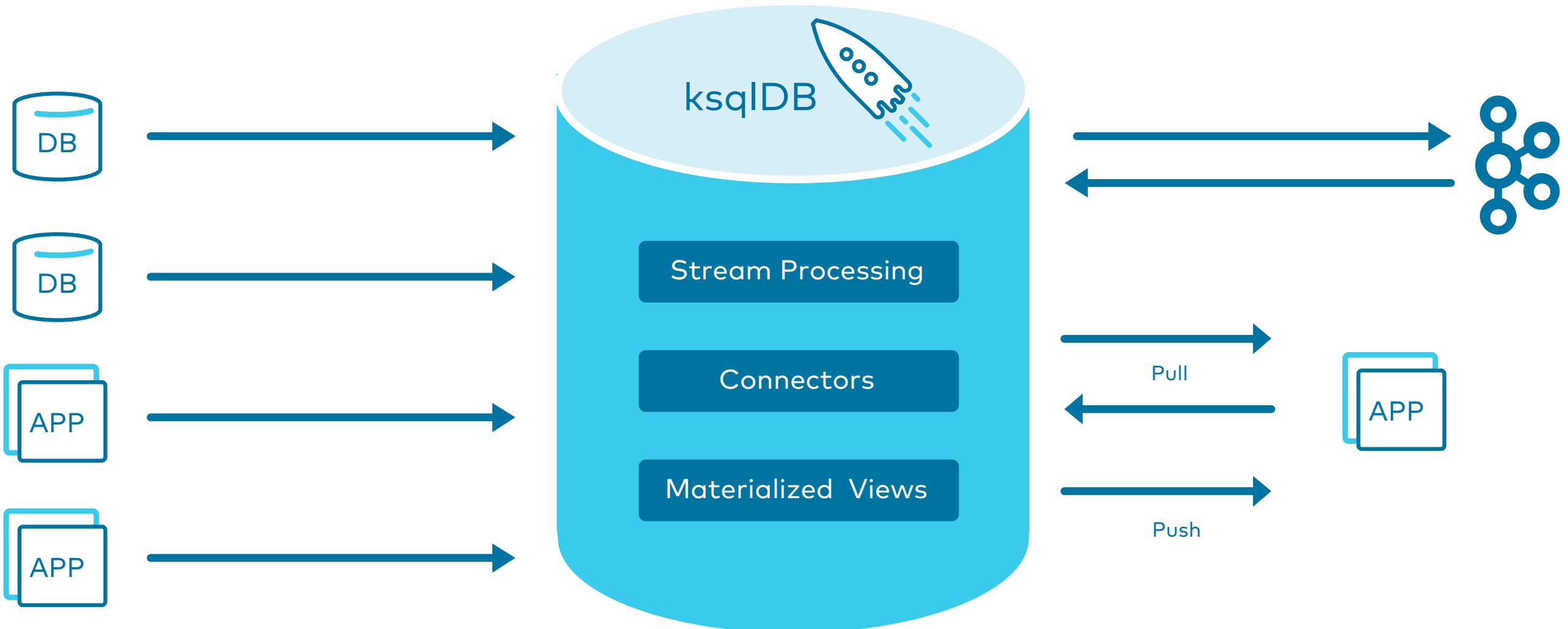
## Description

Best practices and capacity planning for Kafka Streams and ksqlDB in production.

# Reference Architecture: Kafka Streams and ksqlDB



# What is ksqlDB?



# Kafka Streams and ksqlDB Applications

Deploy multiple instances,  
multiple clusters

- Load balancing via Kafka
- Failover via Kafka
- One cluster per team or app

Configure `num.standby.replicas`

- Pro: Faster task failover
- Con: More load on Kafka cluster



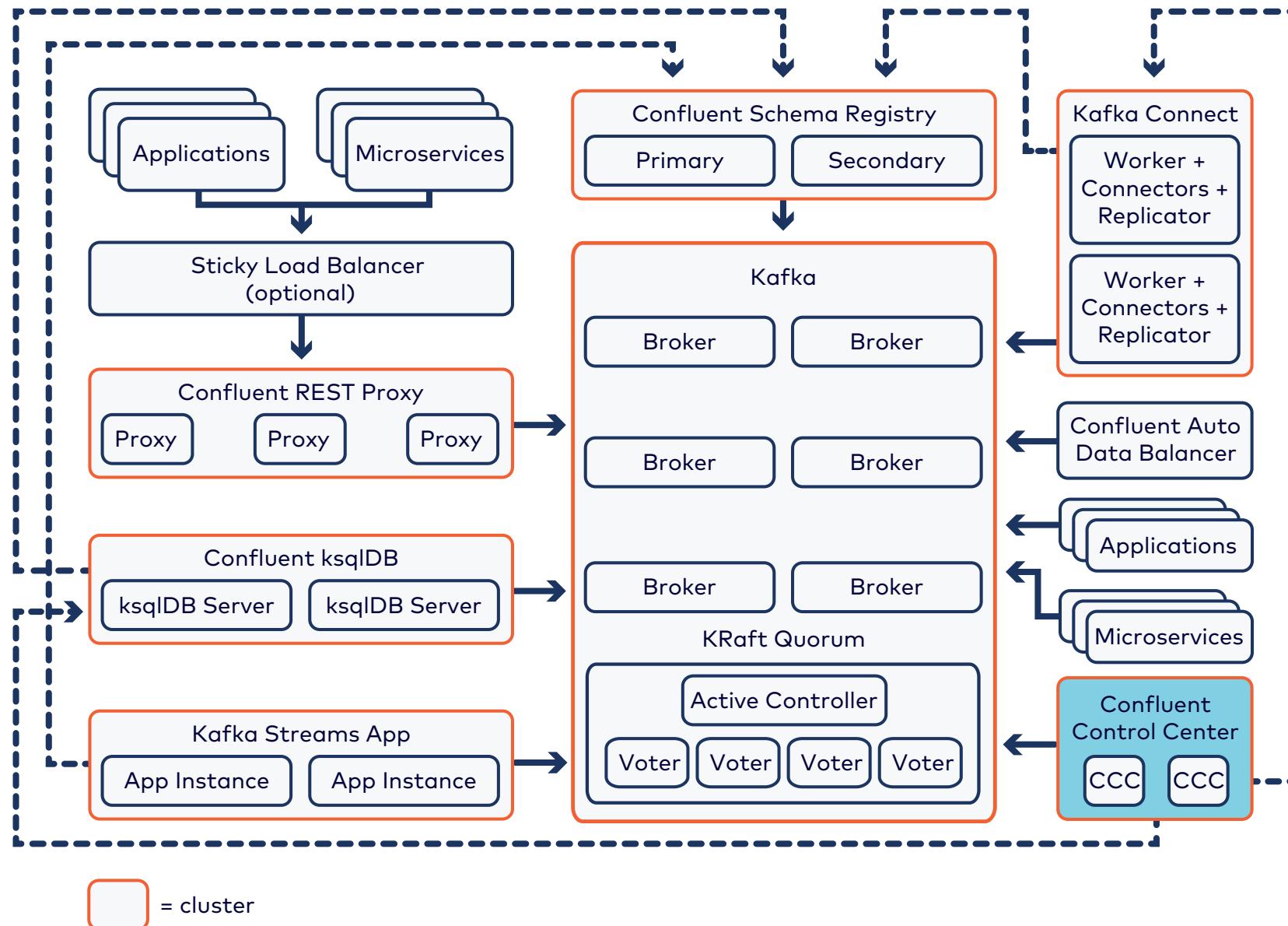
Consider using Kafka quotas to apply a throttling mechanism to ensure high performance for all clients

# 11f: What Does Confluent Advise for Deploying Control Center in Production?

## Description

Best practices and capacity planning for Confluent Control Center in production.

# Reference Architecture: Confluent Control Center



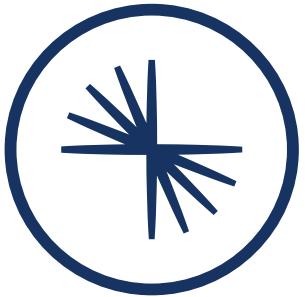
# Highly Available Confluent Control Center

The screenshot shows the CCC interface with the following data:

Brokers	Name	Throughput Bytes in/sec	Bytes out/sec	Latency (fetch) 99.9th %ile	99th %ile	95th %ile	Median	Latency (produ 99.9th %ile
broker.id_101	702kB	702kB	190ms	100ms	50ms	4.0ms	66ms	
broker.id_102	703kB	703kB	320ms	120ms	60ms	4.0ms	94ms	
broker.id_103	697kB	695kB	260ms	120ms	59ms	5.0ms	83ms	

- Deploy a **separate Kafka cluster** dedicated for metrics
- Deploy 2+ machines and load balancer dedicated to CCC
- Configure each UI server with a unique `confluent.controlcenter.id`

# Conclusion



CONFLUENT  
**Global Education**

# Course Contents



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

- Describe how Kafka brokers, producers, and consumers work
- Describe how replication works within the cluster
- Understand hardware and runtime configuration options
- Monitor and administer your Kafka cluster
- Integrate Kafka with external systems using Kafka Connect
- Design a Kafka cluster for high availability & fault tolerance

## Other Confluent Training Courses

- Confluent Developer Skills for Building Apache Kafka®
- Confluent Stream Processing Using Apache Kafka® Streams & ksqlDB
- 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](http://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](http://www.confluent.io/certification)



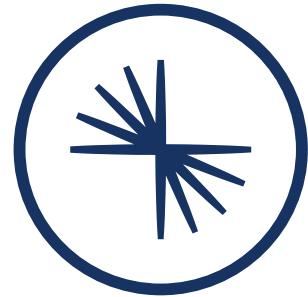
# We Appreciate Your Feedback!



Please complete the course survey now.

# Thank You!

# 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.

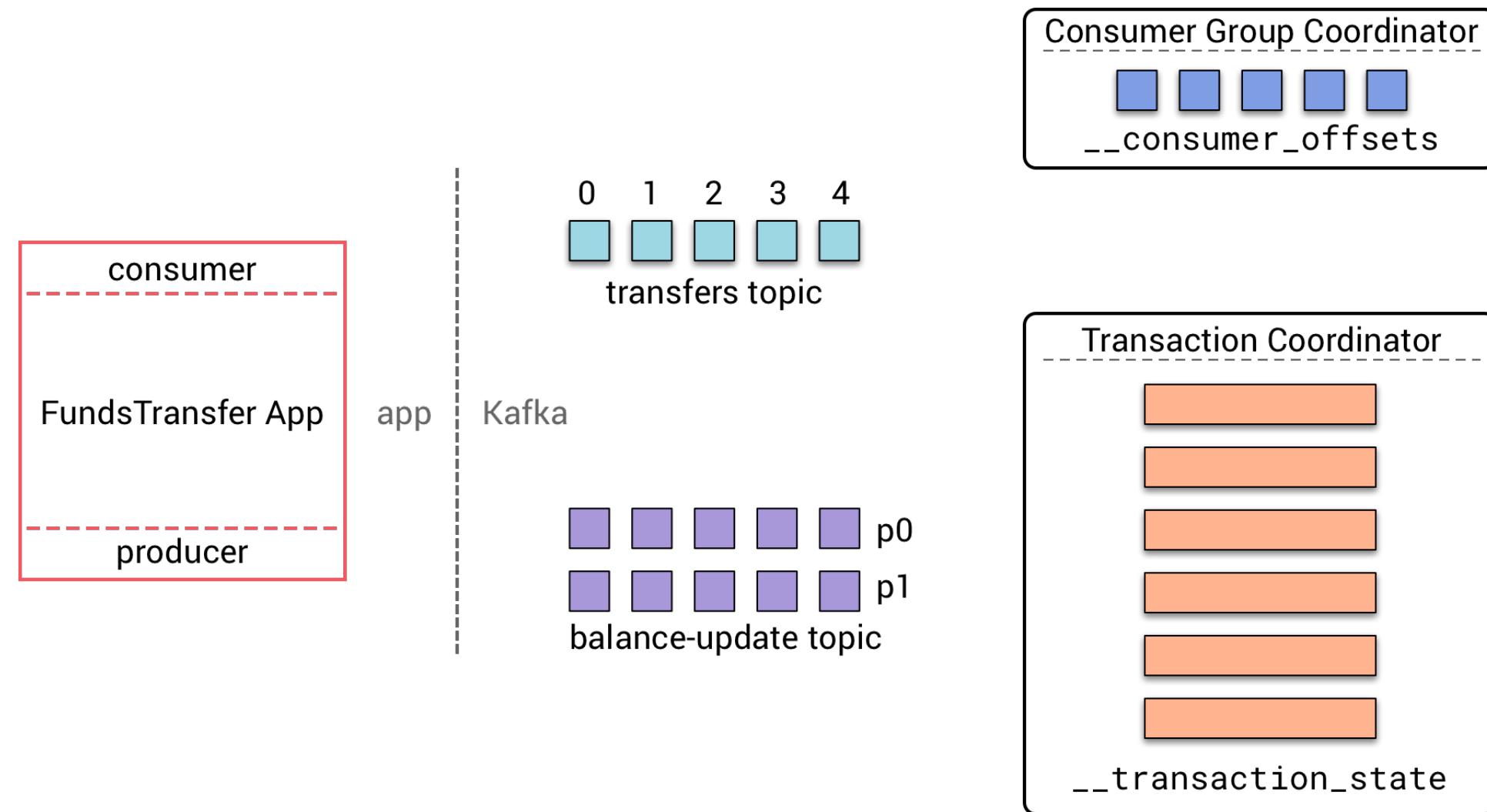
Some lessons that were part of modules of a previous version of this course. Some are lessons taken from another course.

# Appendix A: Detailed Transactions Demo

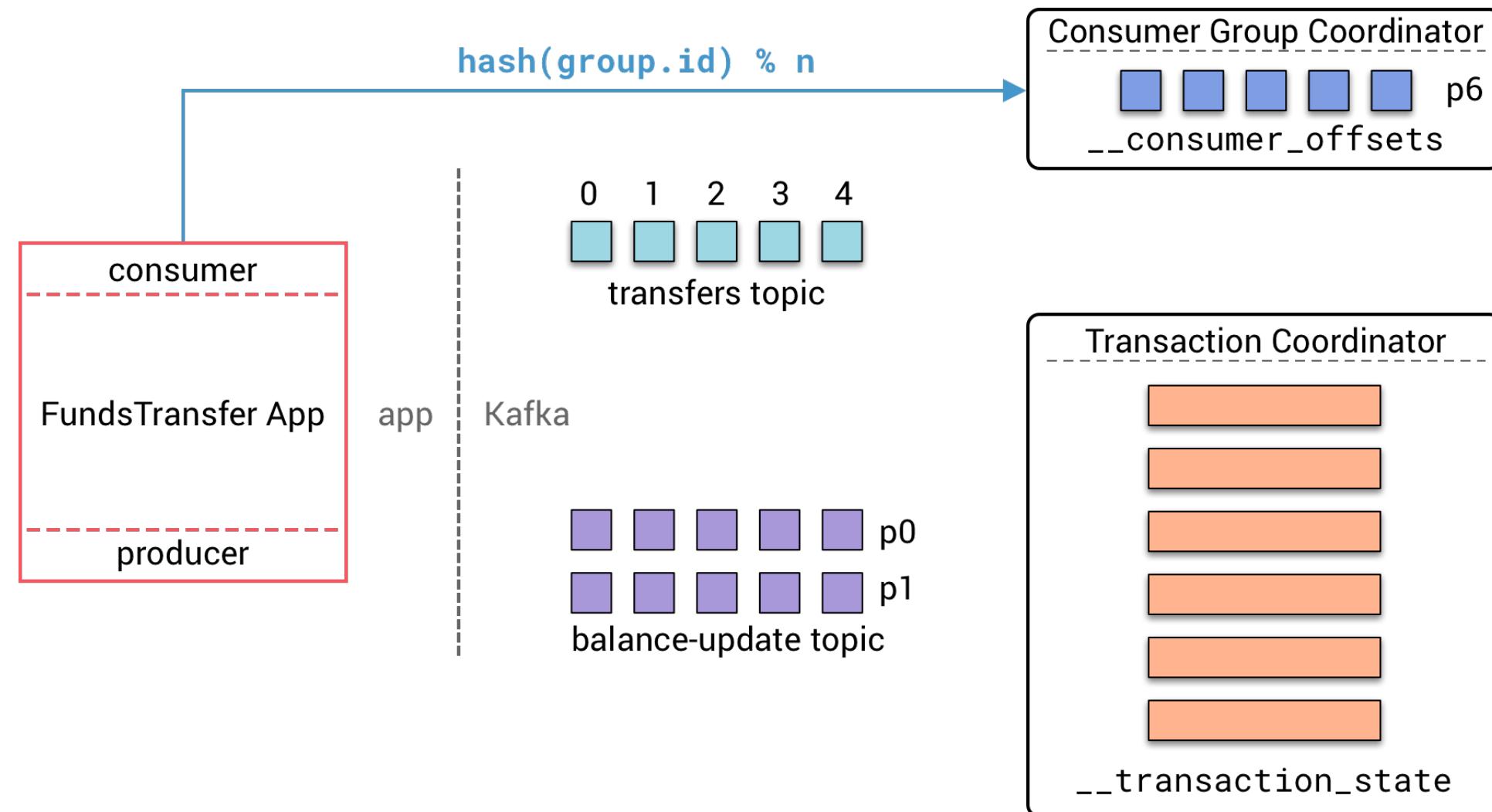
## Description

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

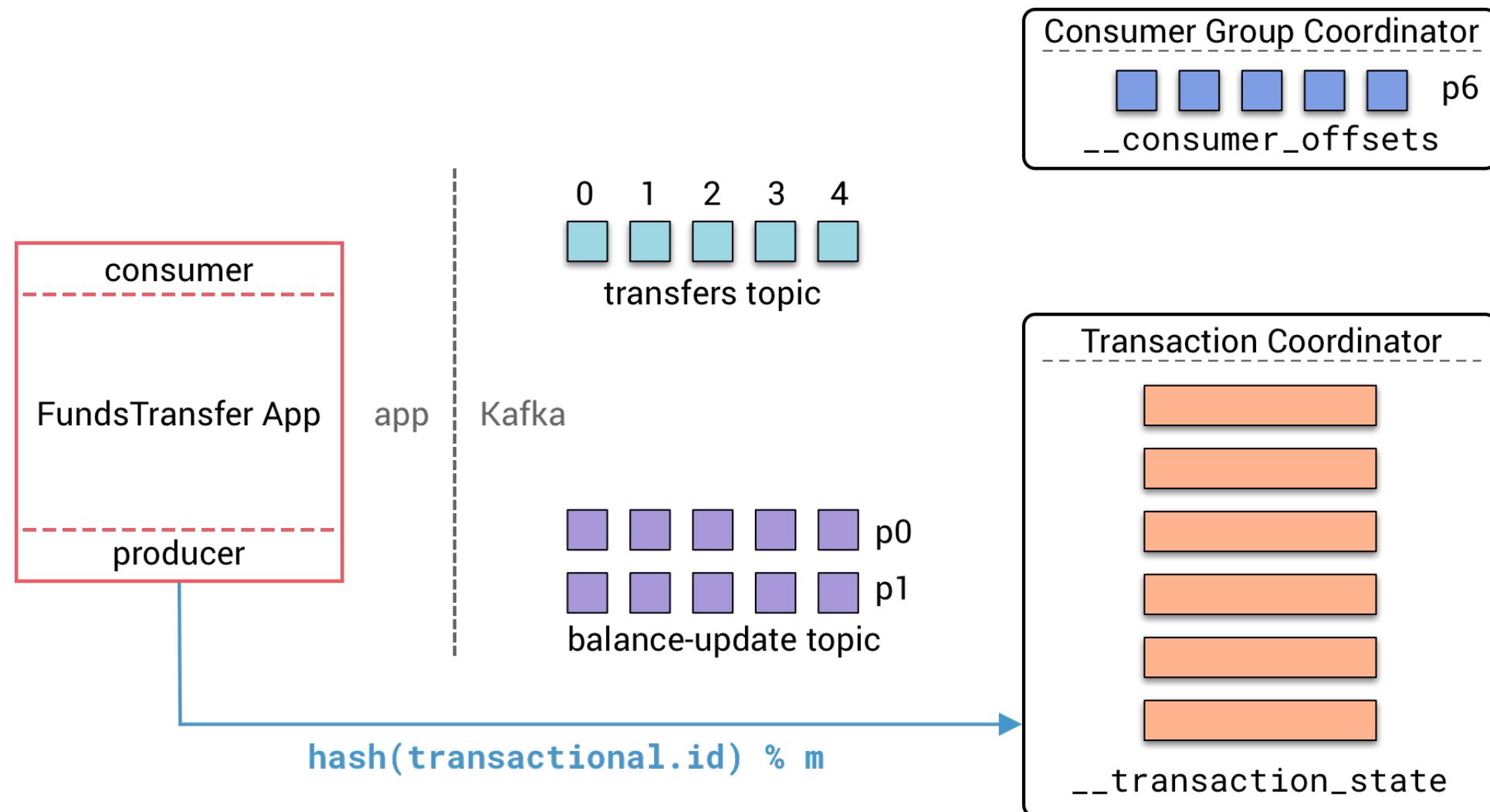
## Transactions (1/14)



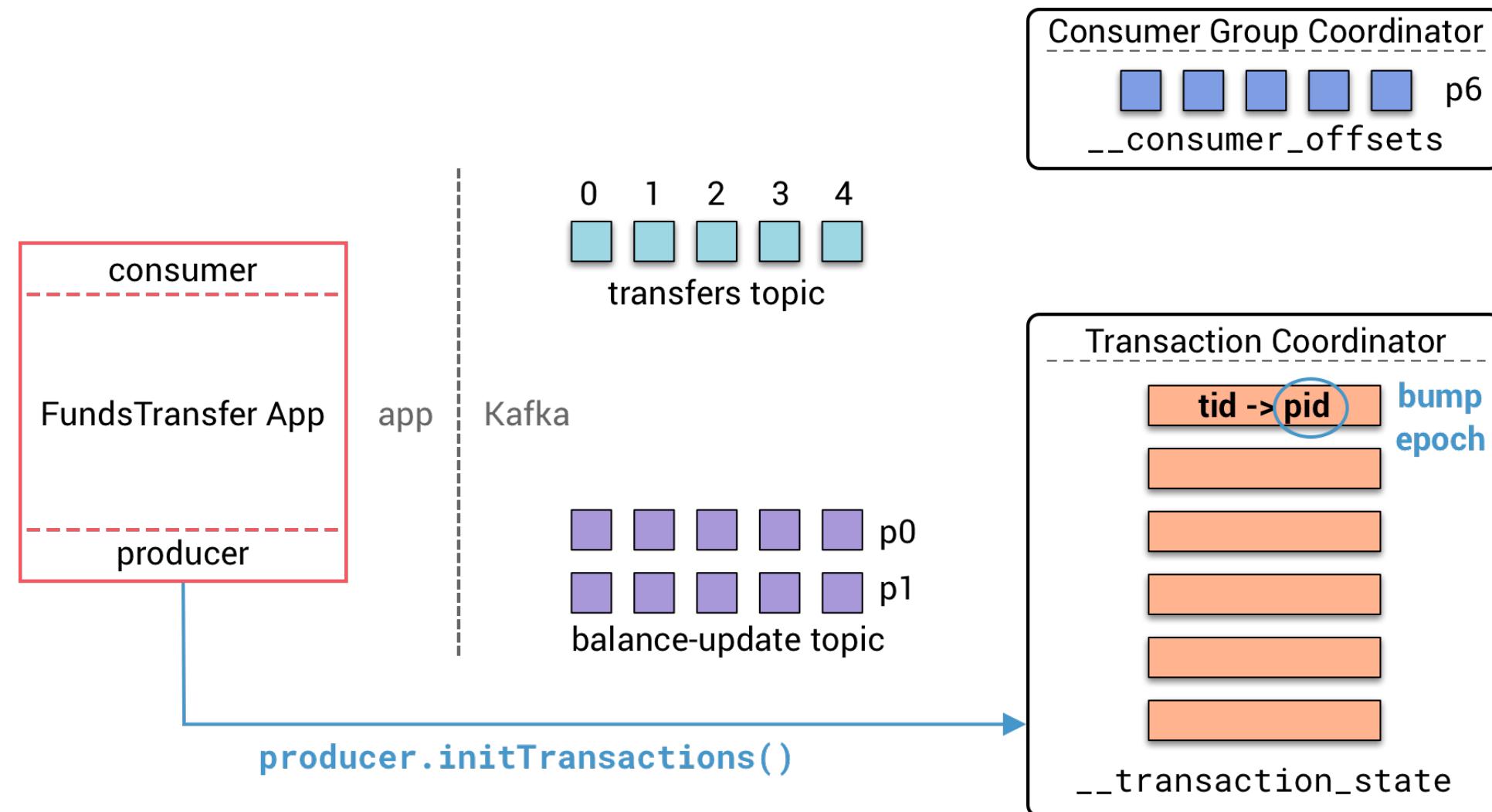
## Transactions - Initialize Consumer Group (2/14)



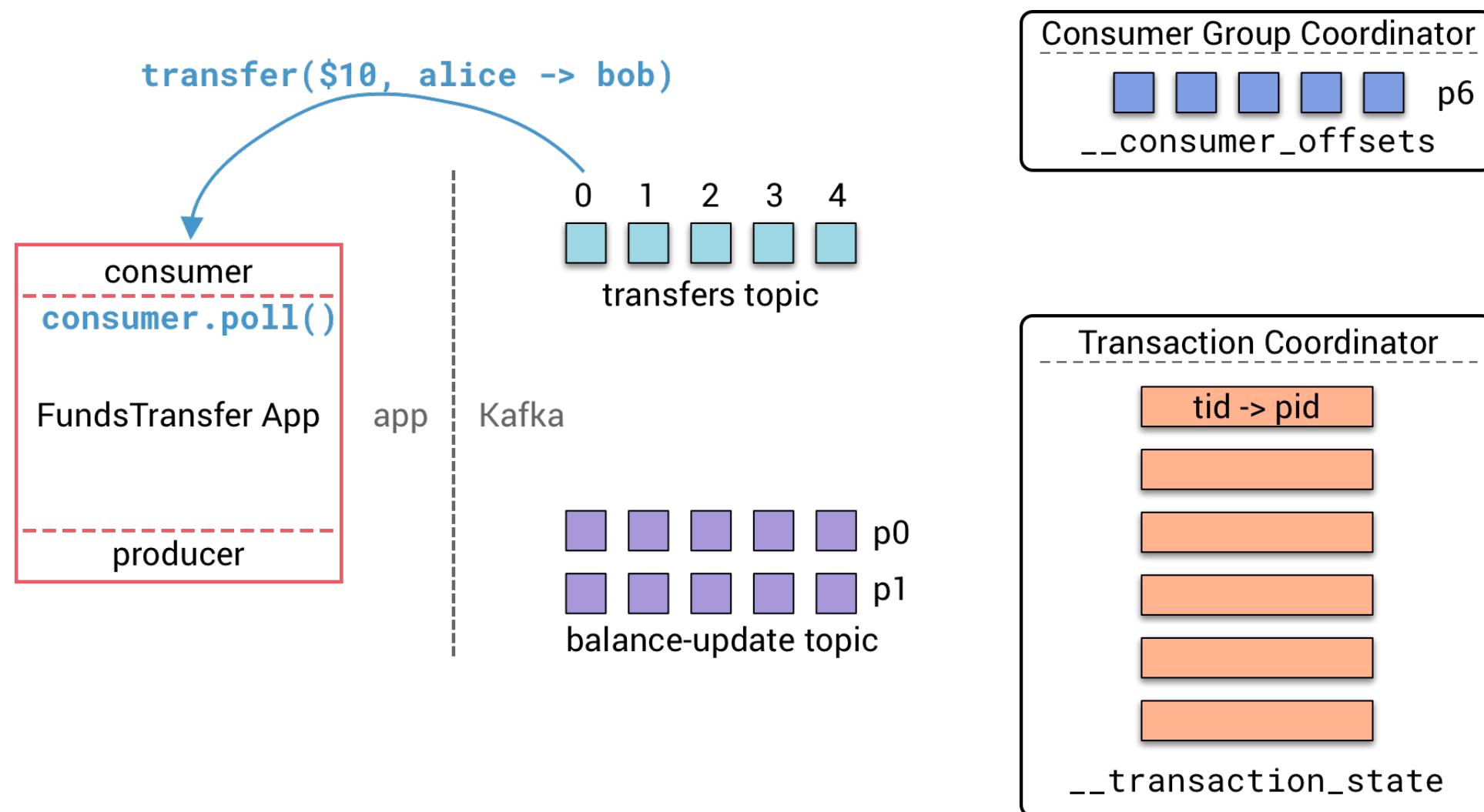
## Transactions - Transaction Coordinator (3/14)



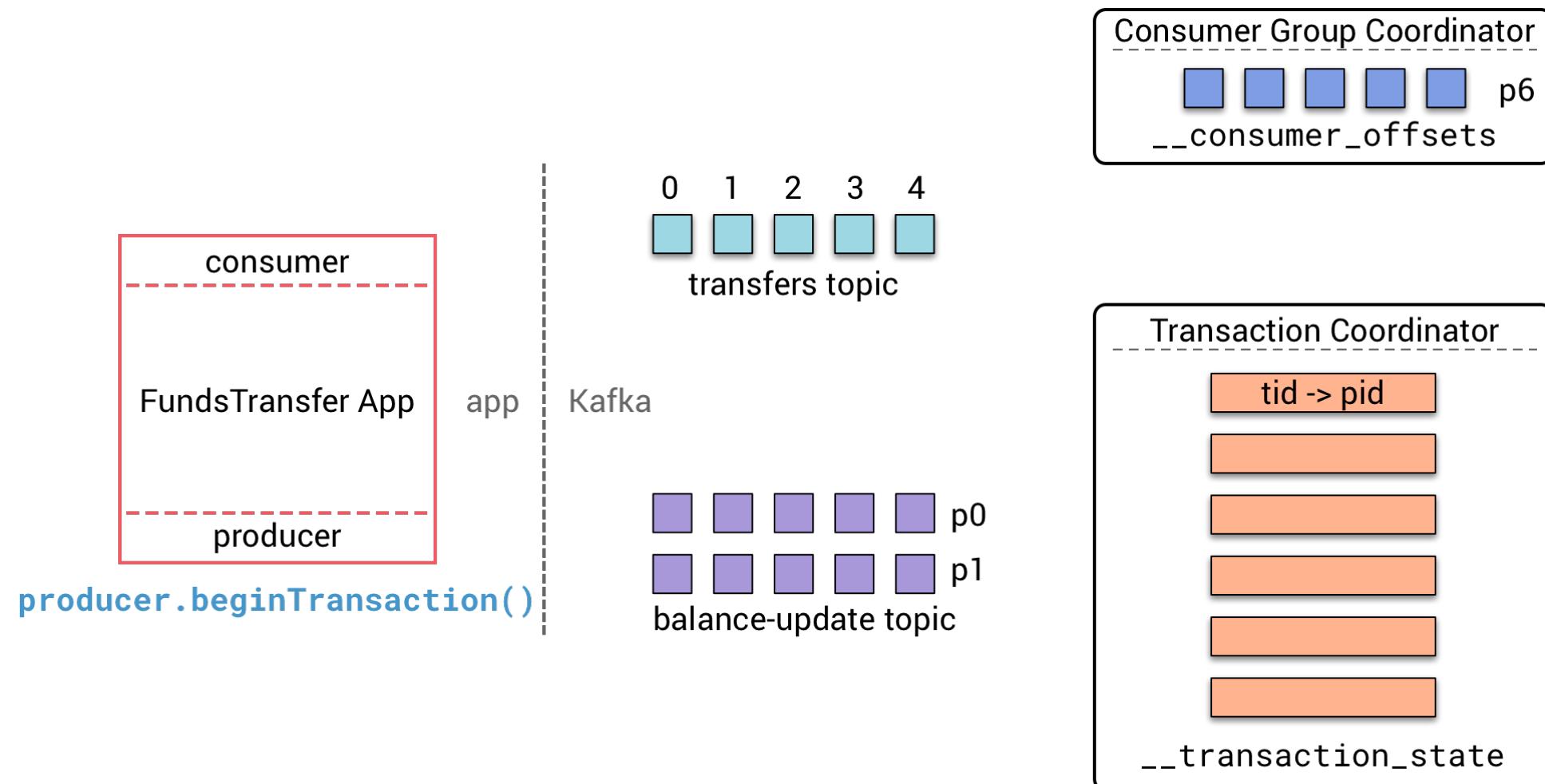
## Transactions - Initialize (4/14)



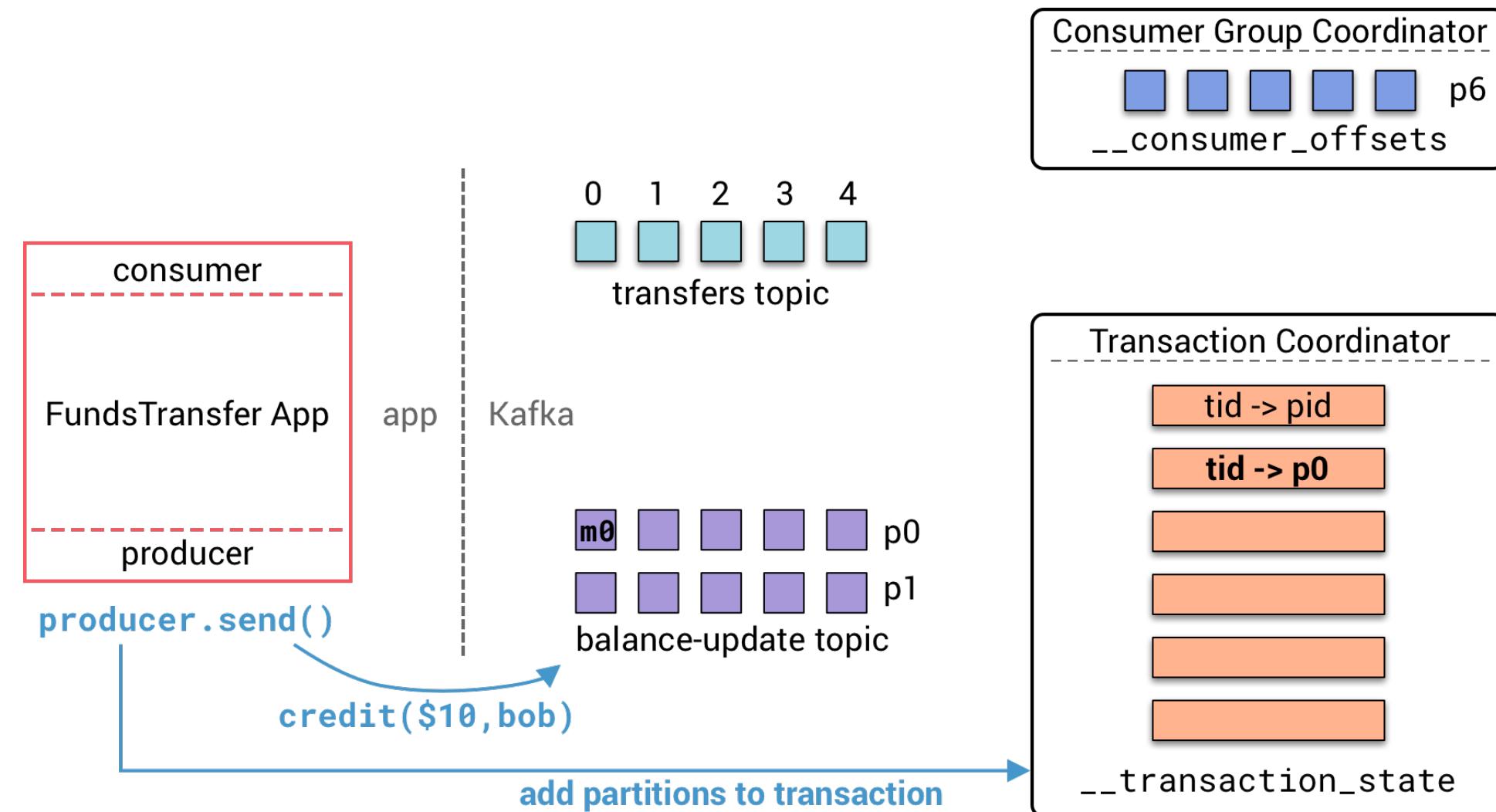
## Transactions - Consume and Process (5/14)



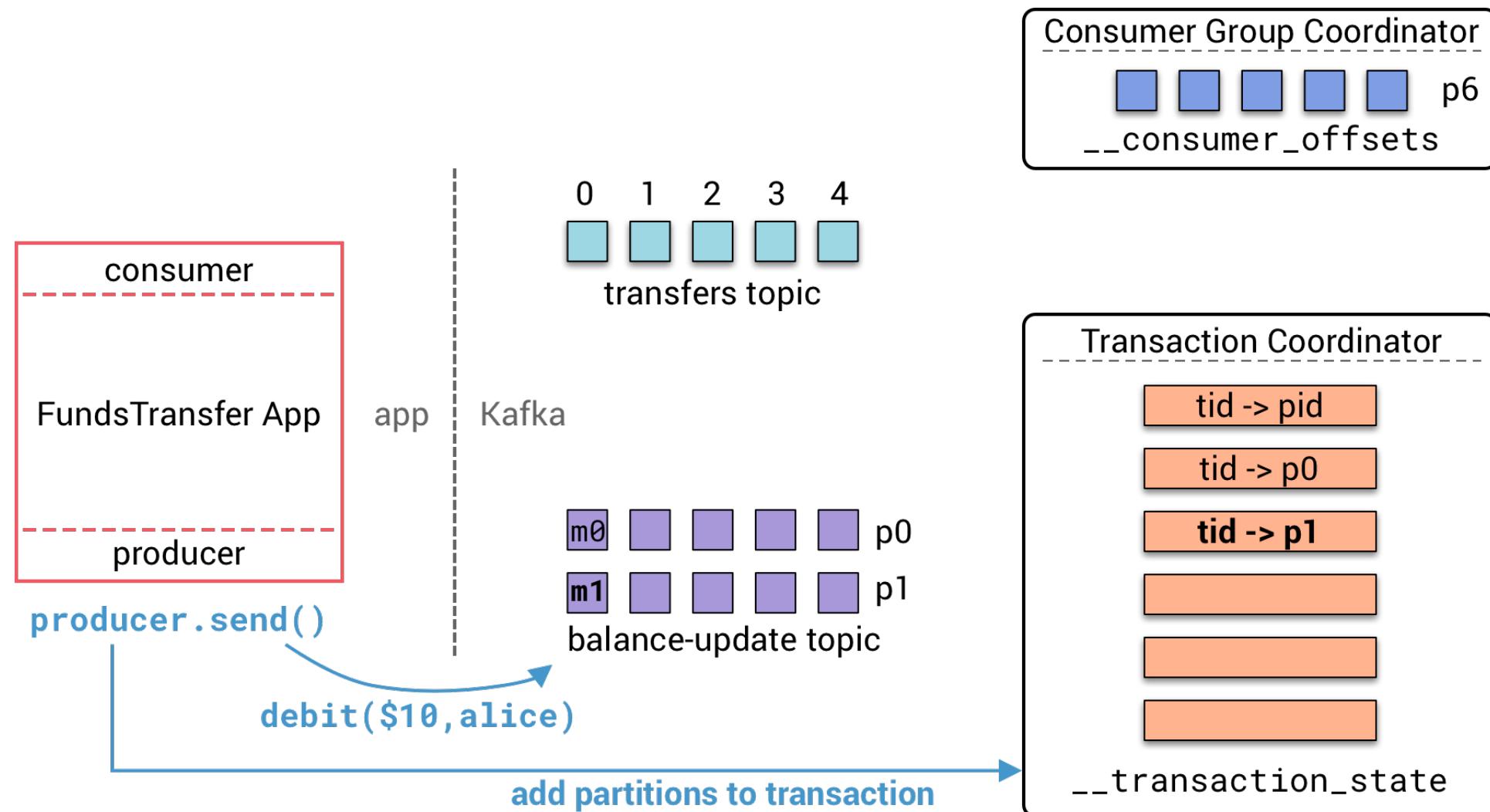
## Transactions - Begin Transaction (6/14)



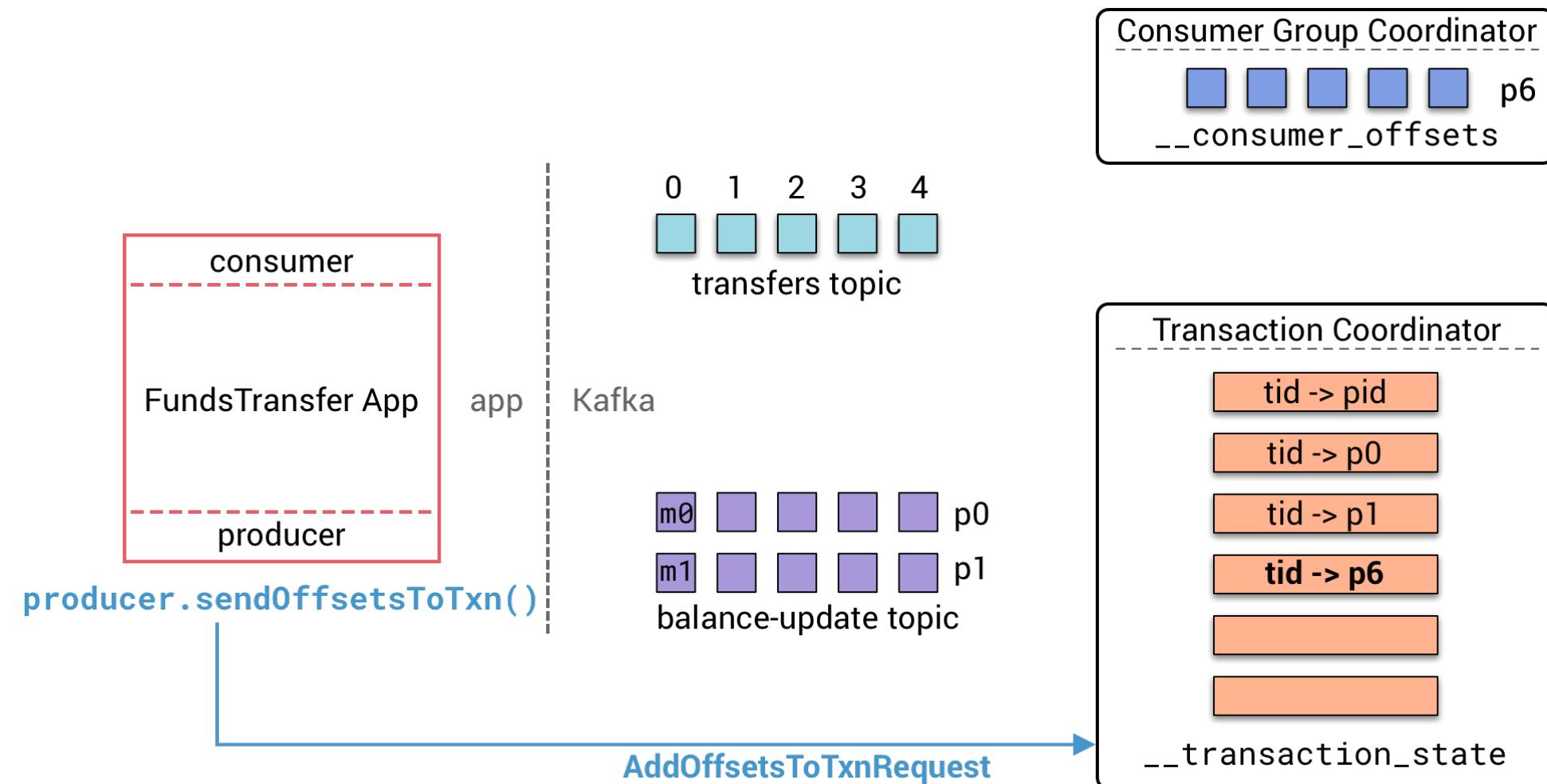
## Transactions - Send (7/14)



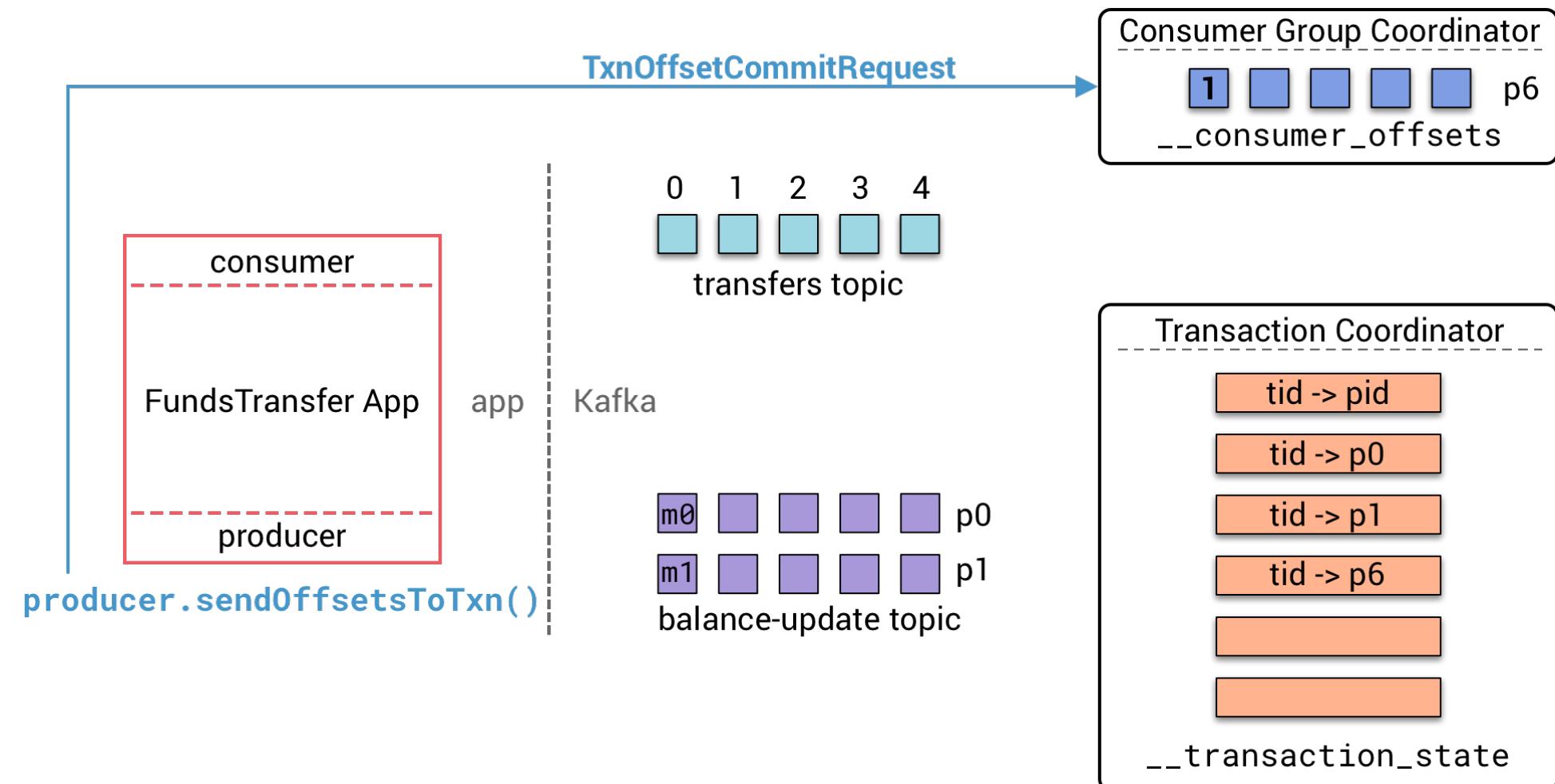
## Transactions - Send (8/14)



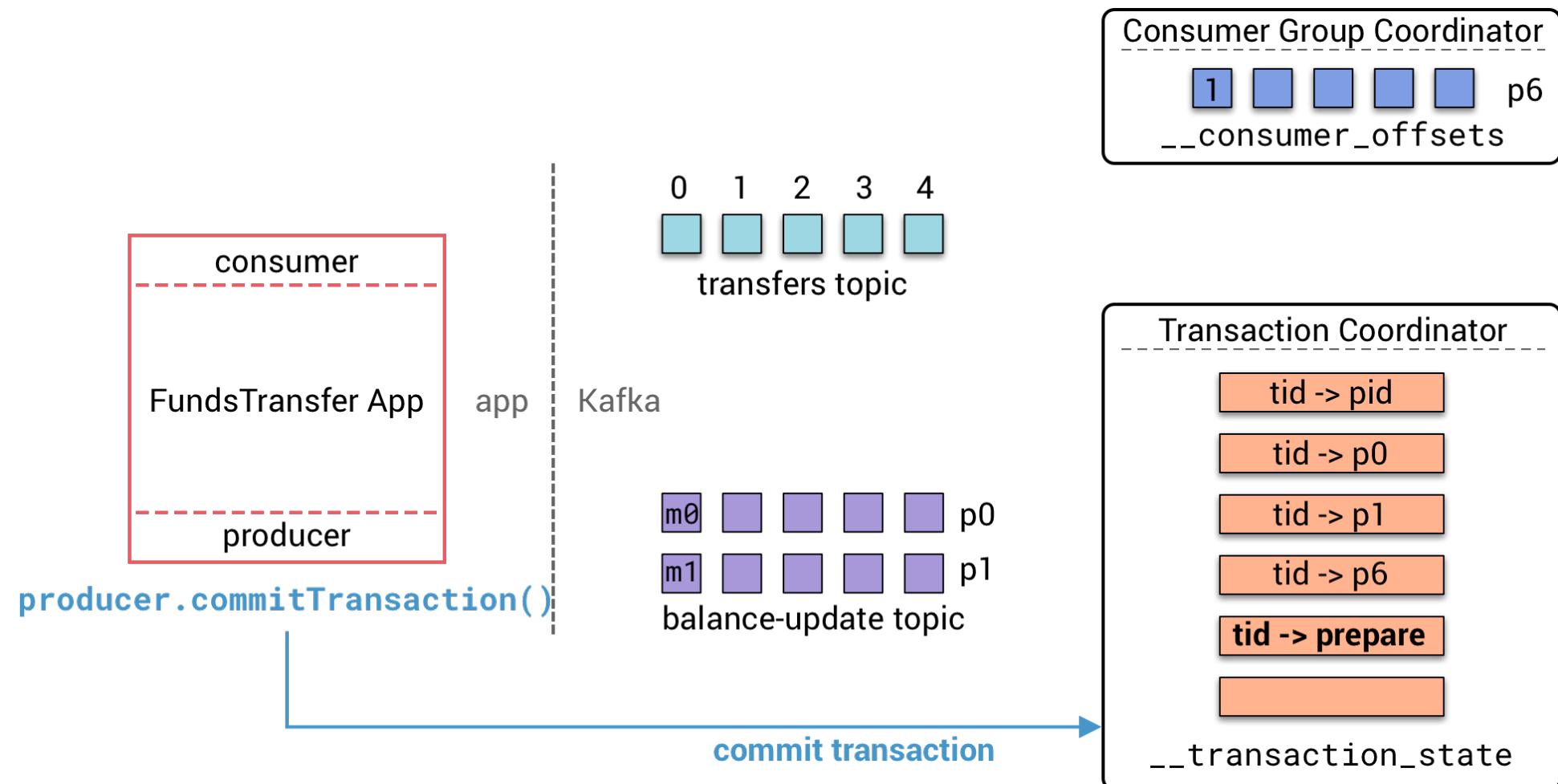
## Transactions - Track Consumer Offset (9/14)



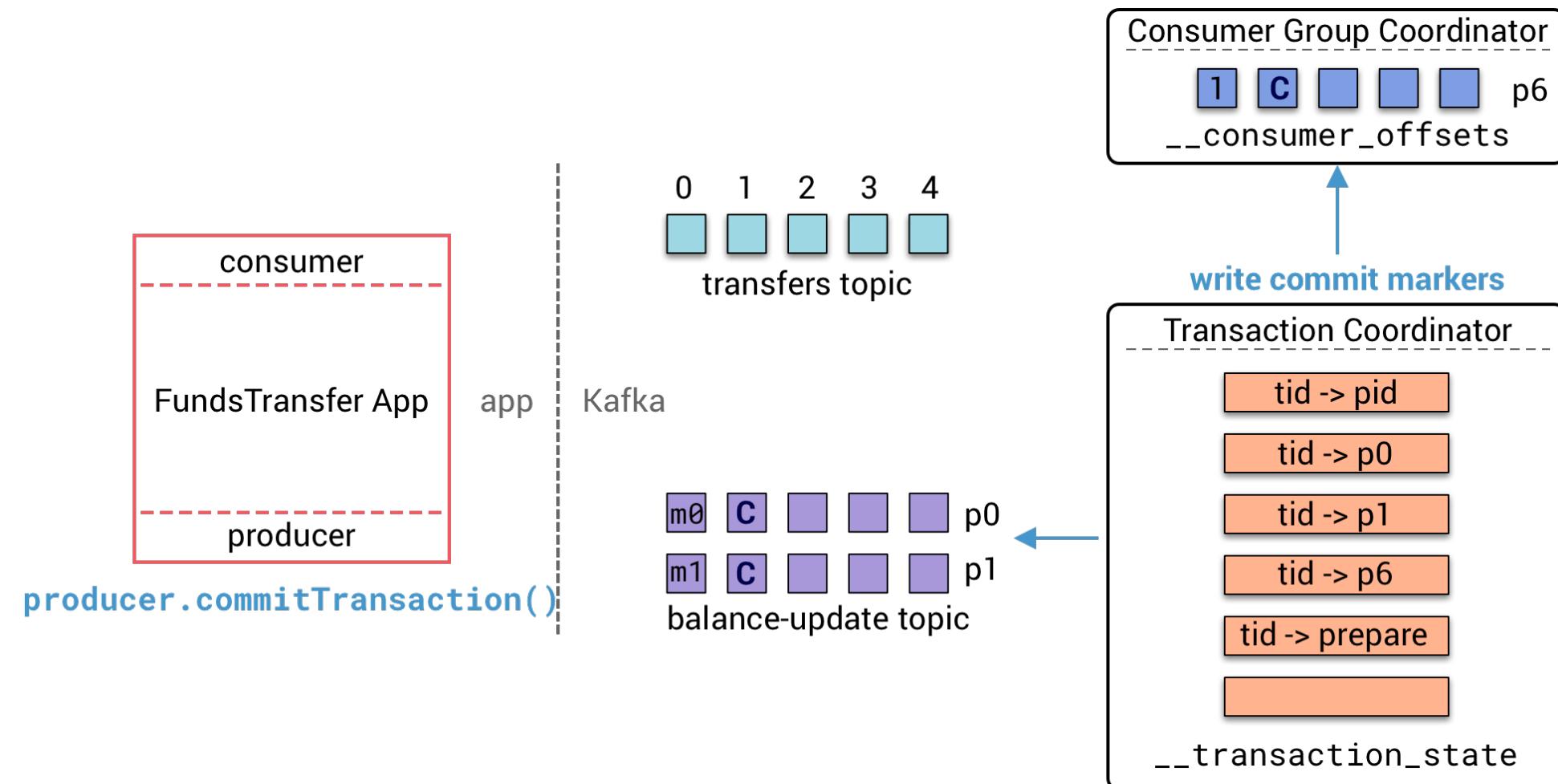
## Transactions - Commit Consumer Offset (10/14)



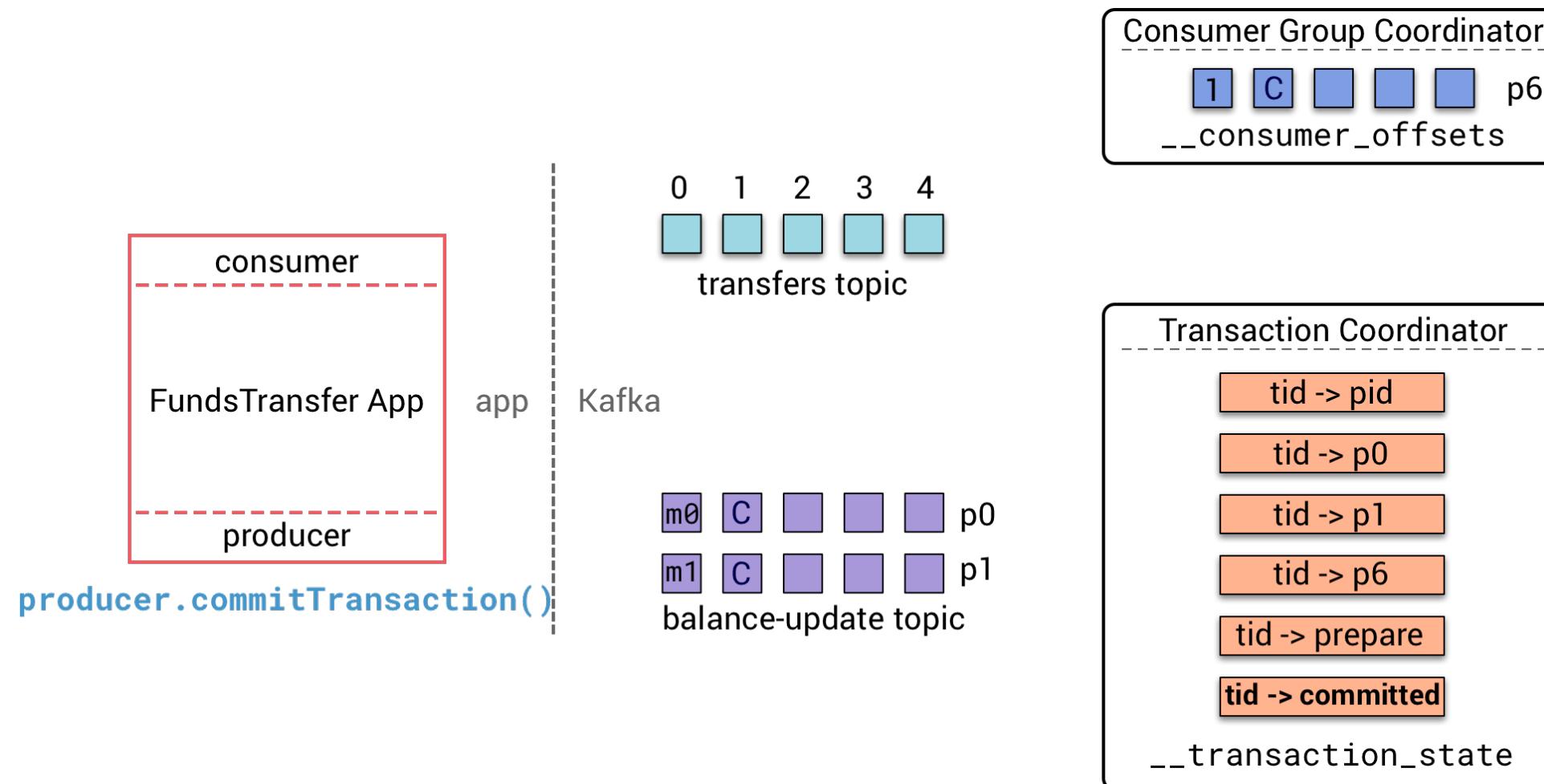
## Transactions - Prepare Commit (11/14)



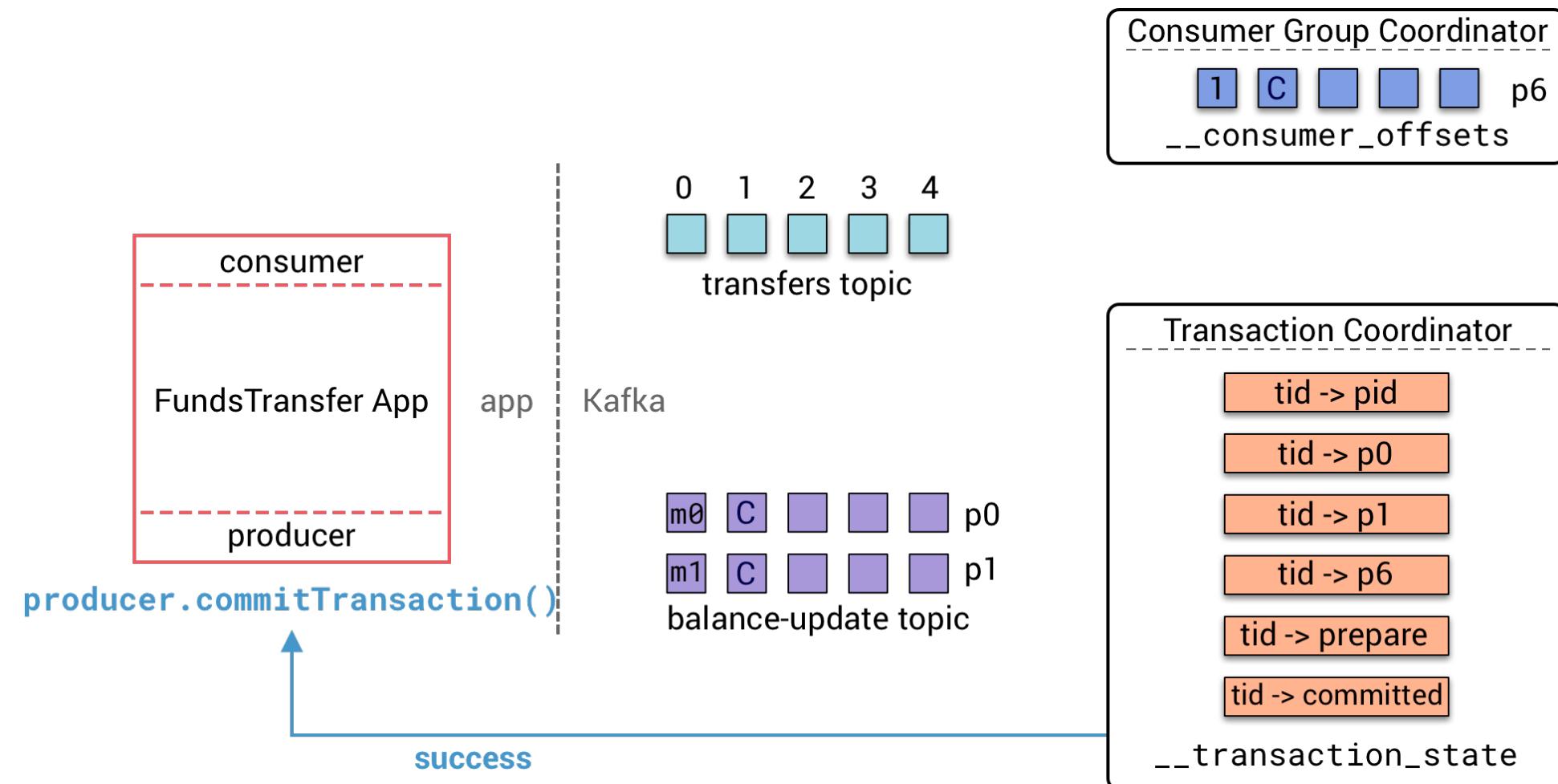
## Transactions - Write Commit Markers (12/14)



## Transactions - Commit (13/14)



## Transactions - Success (14/14)



# Appendix B: How Can You Monitor Replication?

## Description

Monitoring considerations for replication.

## Monitoring Leader Election Rate

- Leader election rate (JMX metric)

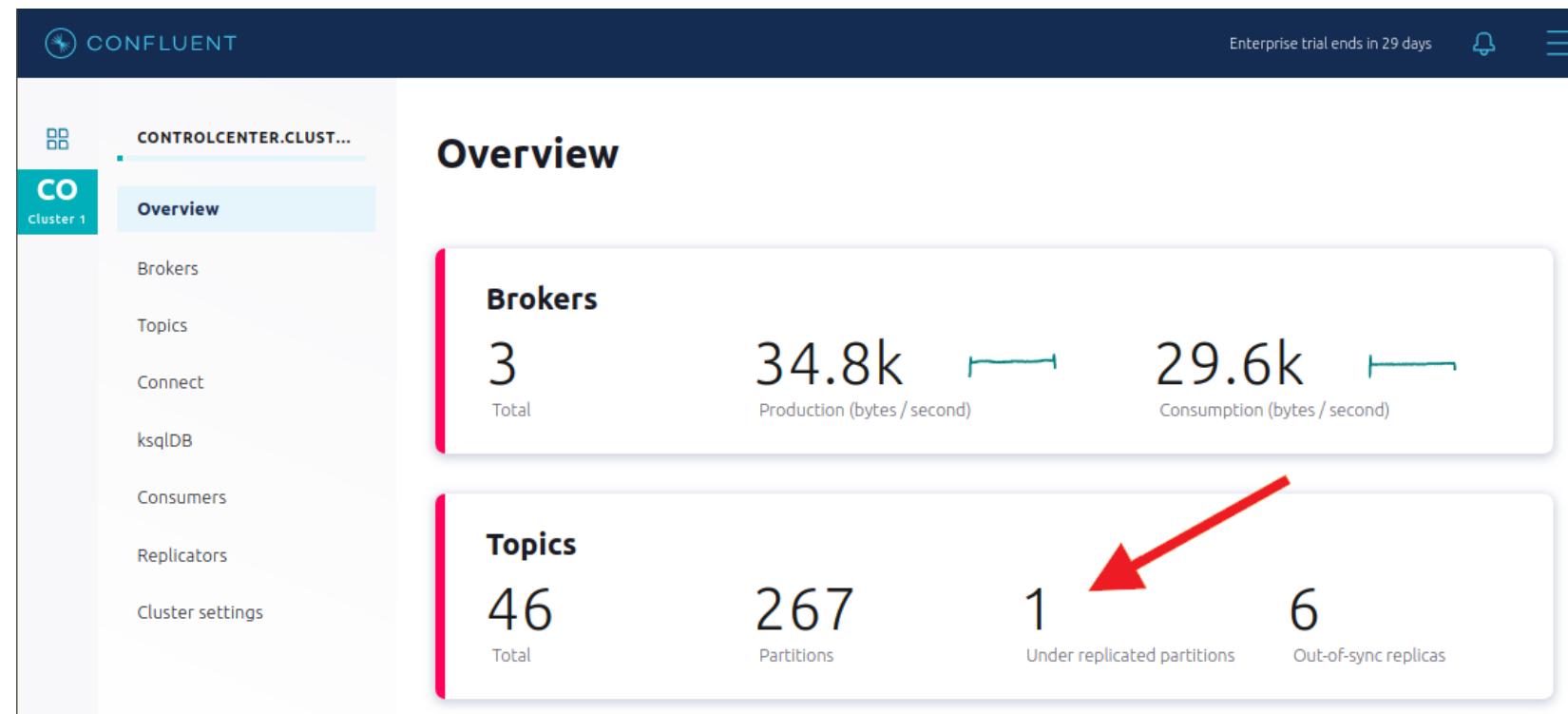
```
kafka.controller:type=ControllerStats, name=LeaderElectionRateAndTimeMs
```

## Monitoring ISR

- Monitor under-replicated partitions with the JMX metric:
  - `kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions`
  - Alert if the value is greater than 0 for a long time
- Track changes of ISR lists (shrinks and adds) with these JMX metrics:
  - `kafka.server:type=ReplicaManager,name=IsrExpandsPerSec`
  - `kafka.server:type=ReplicaManager,name=IsrShrinksPerSec`

## Monitoring for Under Replicated Partitions

- If a broker goes down, the ISR for some partitions will shrink
- Confluent Control Center shows partition health at a glance:



## Monitoring Offline Partitions

- Track offline partitions with JMX metric:
  - `kafka.controller:type=KafkaController,Name=OfflinePartitionsCount`
- Leader failure makes partition unavailable until re-election
  - Producer `send()` will retry according to `retries` configuration
  - Callback raises `NetworkException` if `retries == 0`

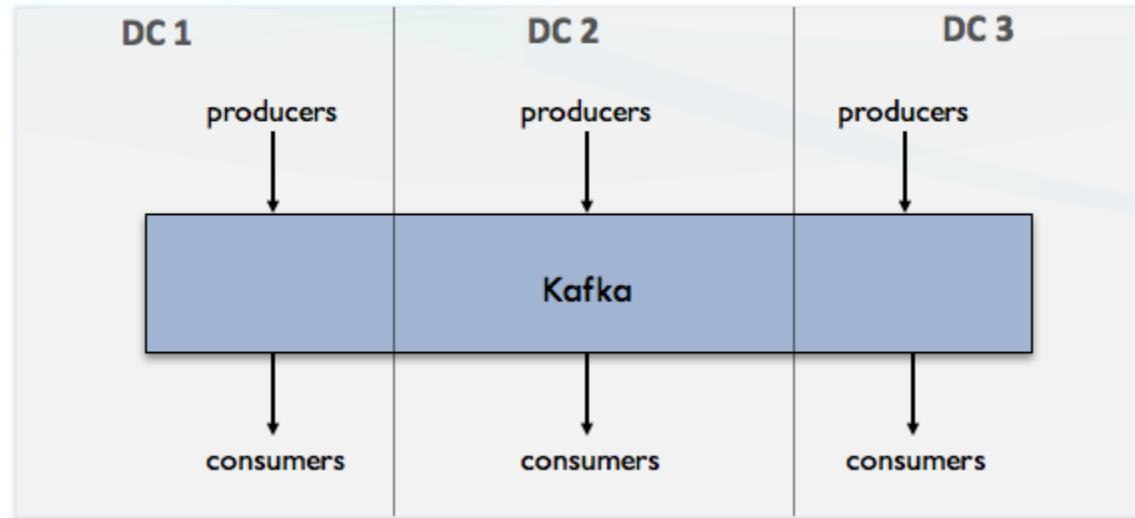
## Appendix C: Multi-Region Clusters

# Multiple Data Centers

- Kafka only:
  - Stretched (a.k.a. multi-AZ)
- Confluent Replicator:
  - Cluster aggregation
  - Active/Passive
  - Active/Active



# Stretched Deployment



- Discussion Questions
  - How should you deploy ZooKeeper and Kafka across 3 availability zones?
  - What are possible tradeoffs between a stretched cluster vs. a single DC cluster?
  - What are some possible failure scenarios and how does Kafka respond?

## **Confluent Replicator or Apache Kafka MirrorMaker**

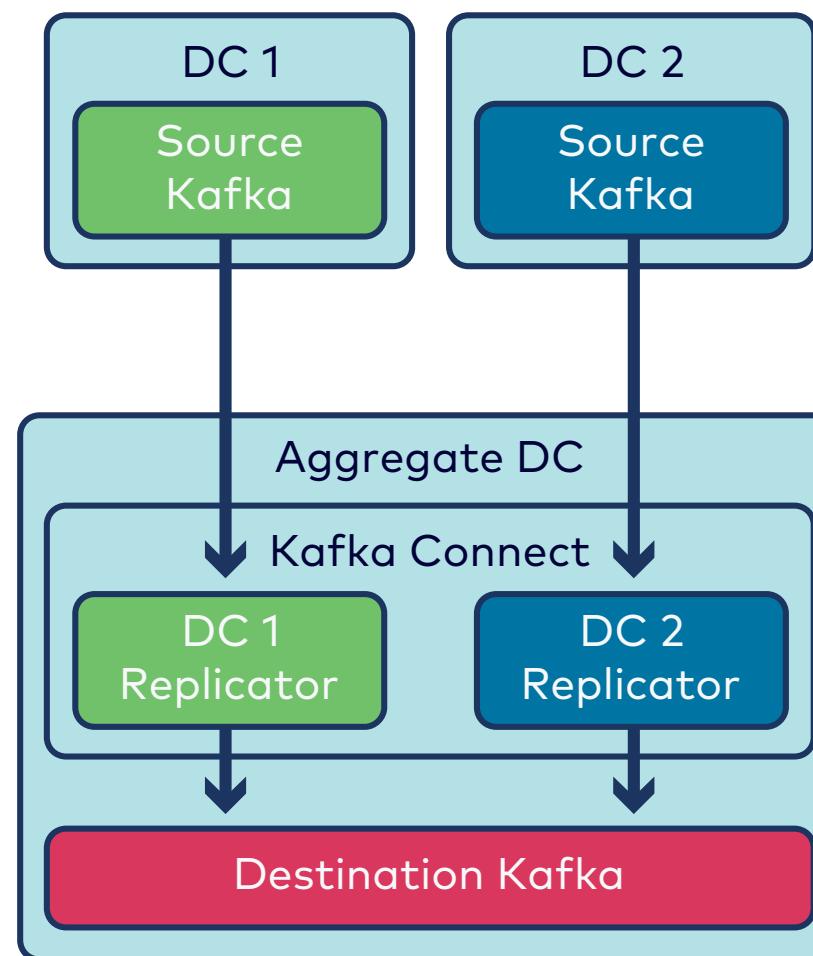
## Deploying Replicator

1. Provision machines in **destination data center**
2. Install with `confluent-hub` if not already using CP
3. Configure `worker.properties` file on each Connect machine
4. Ways to start Replicator:
  - Submit HTTP request with Replicator-specific properties, **or**
  - Use the `replicator` command on each Kafka Connect machine

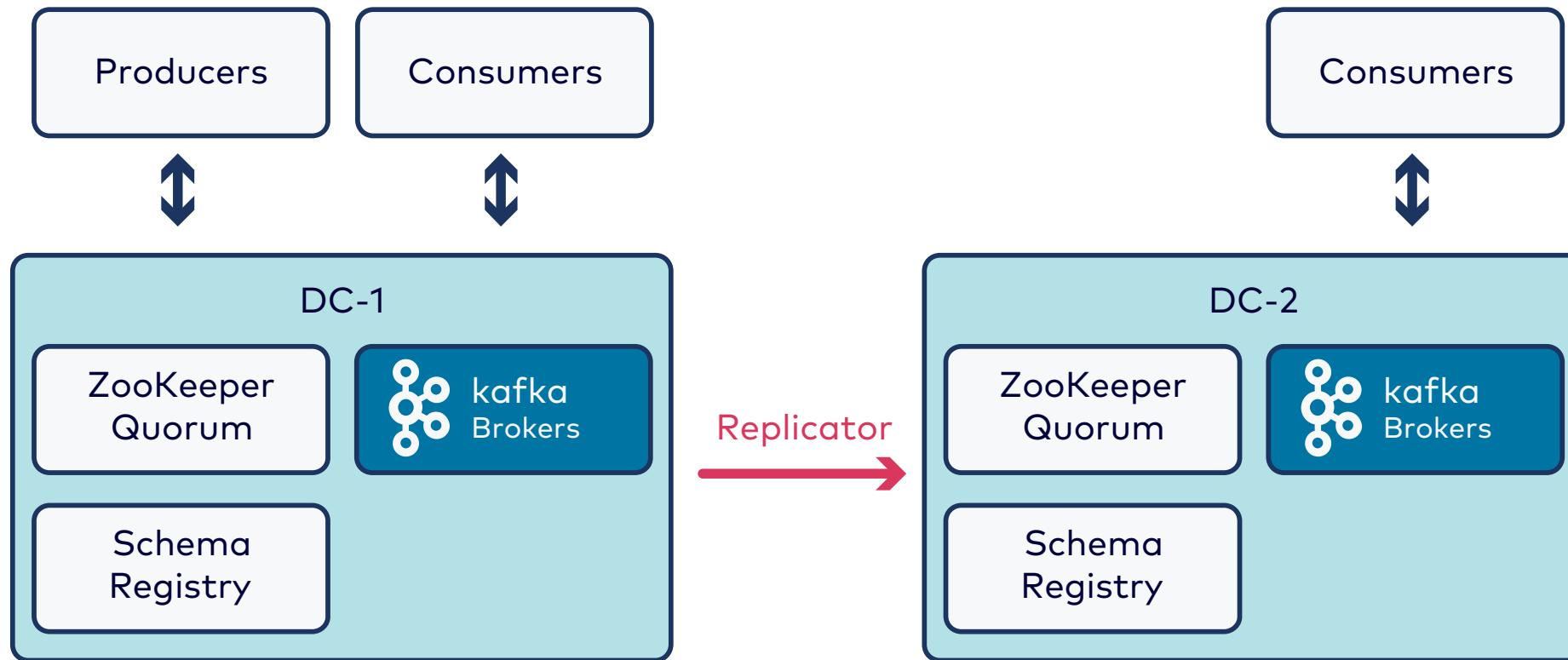


Confluent Replicator requires an enterprise license

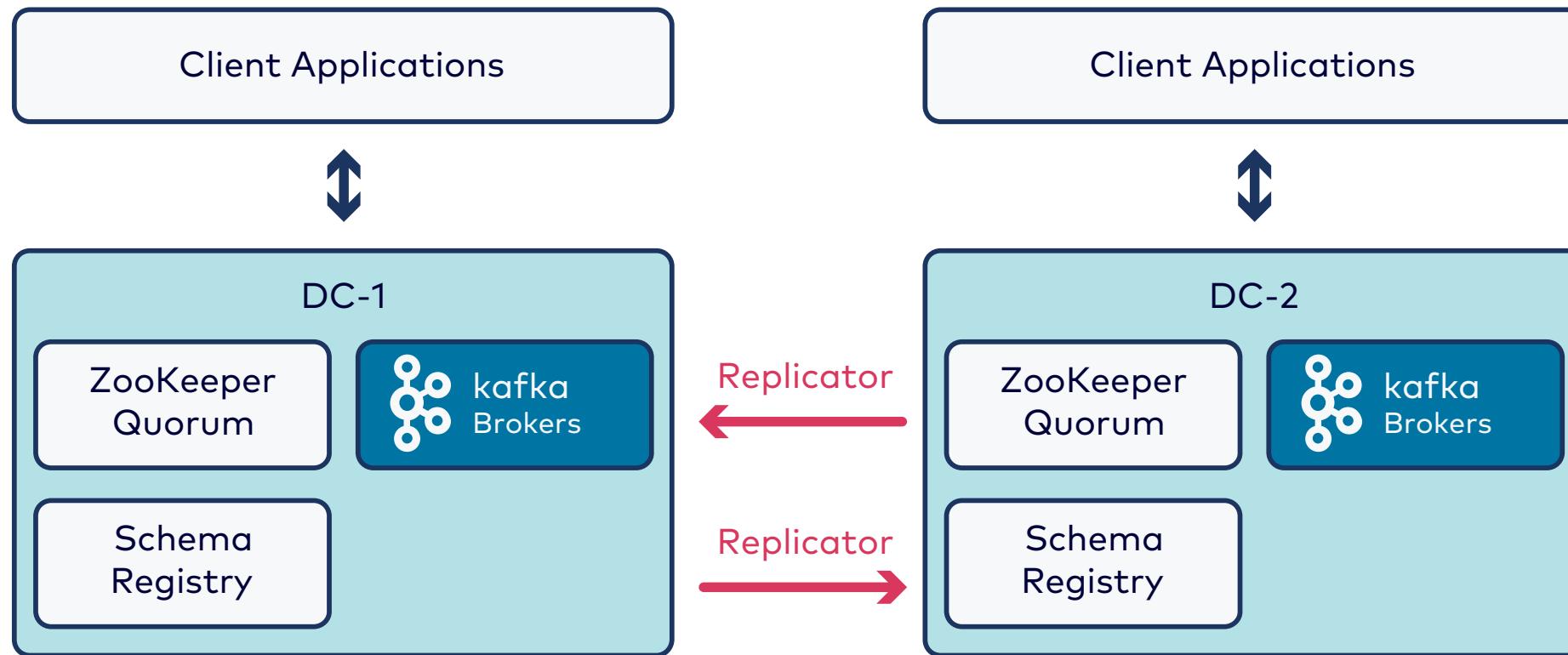
# Cluster Aggregation



# Active-Passive

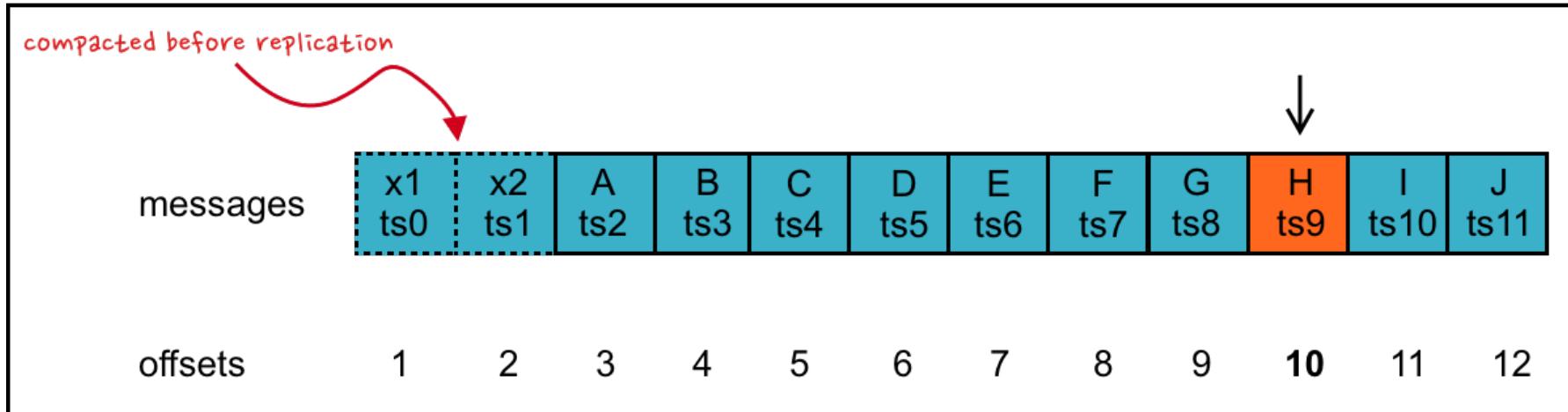


# Active-Active



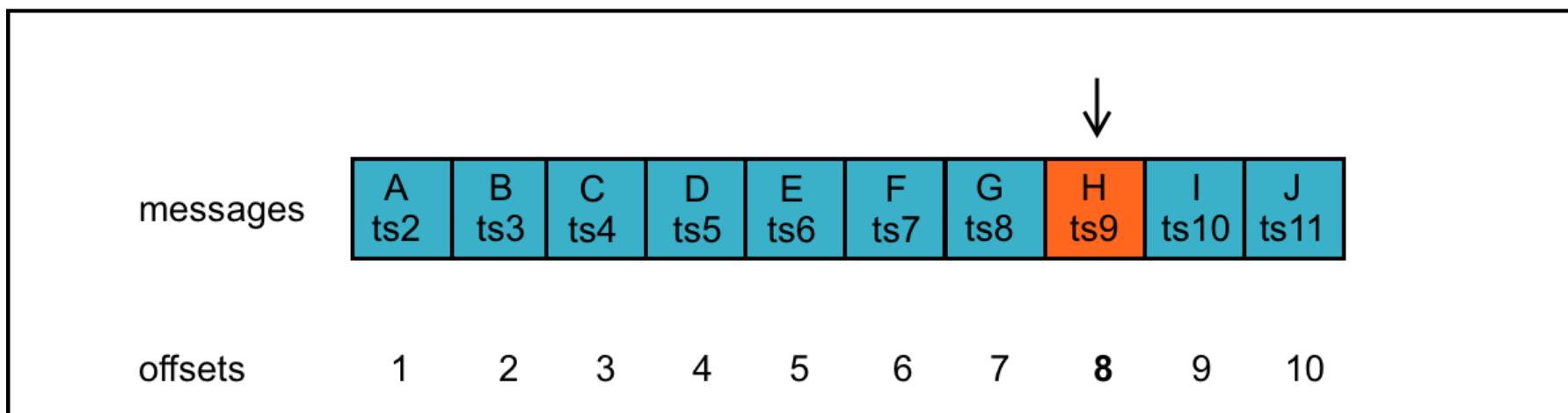
# Client Offset Translation (1)

Data Center 1



Consumer Group	Offset	Timestamp
my-group-1	10	ts9

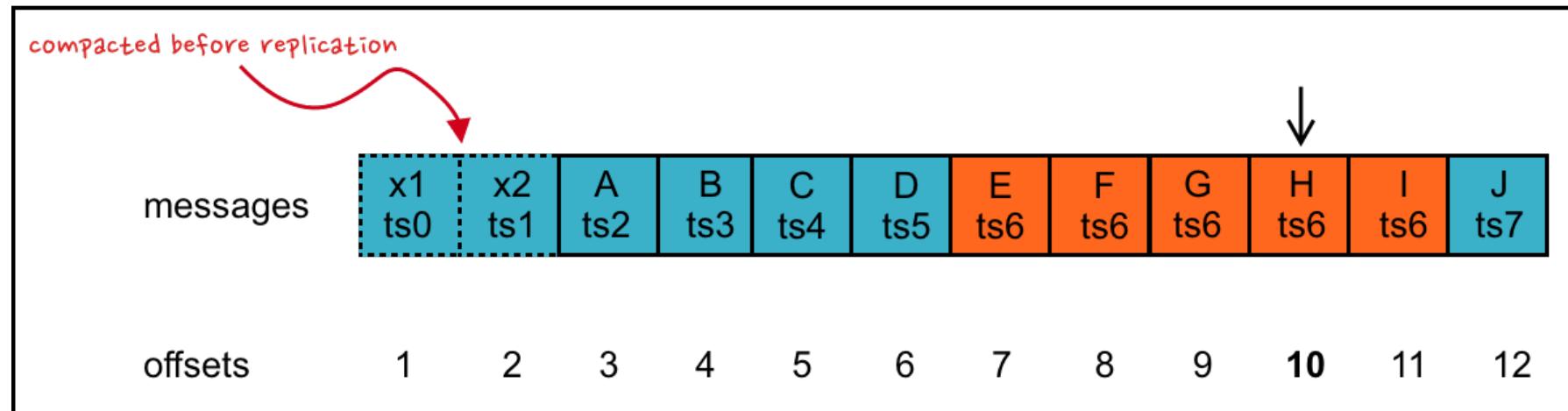
Data Center 2



Consumer Group	Offset
my-group-1	8

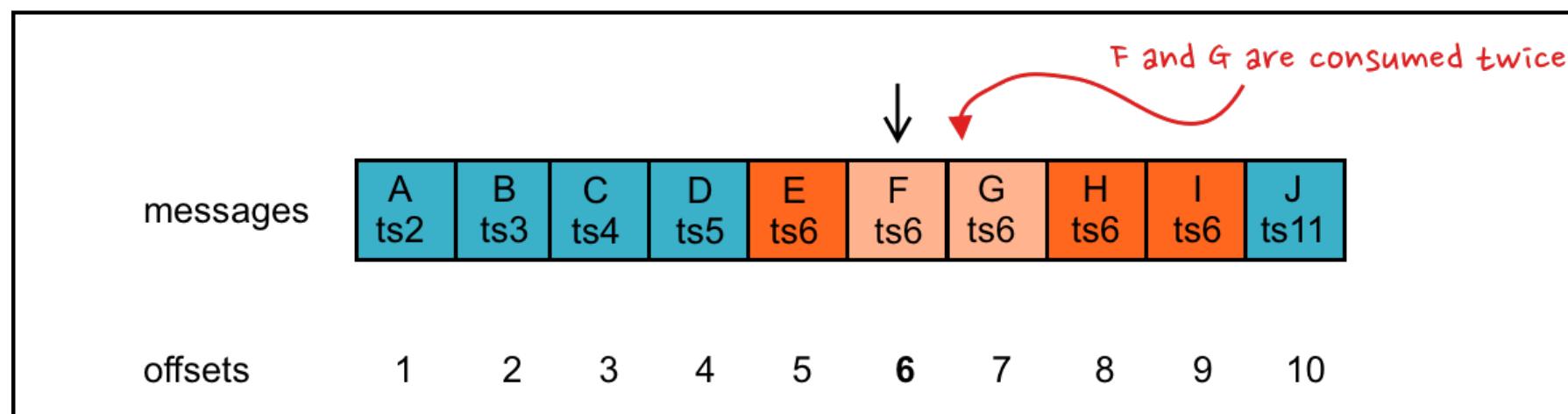
## Client Offset Translation (2)

Data Center 1



Consumer Group	Offset	Timestamp
my-group-1	10	ts6

Data Center 2



Consumer Group	Offset
my-group-1	6

## Manual Disaster Recovery: Consumer Group Tool

The command `kafka-consumer-groups` can set a Consumer Group to seek to an offset derived from a timestamp

```
$ kafka-consumer-groups \
  --bootstrap-server dc2-broker-101:9092 \
  --reset-offsets \
  --topic dc1-topic \
  --group my-group \
  --execute \
  --to-datetime 2017-08-01T17:14:23.933
```

# Manual Disaster Recovery: Java Client API

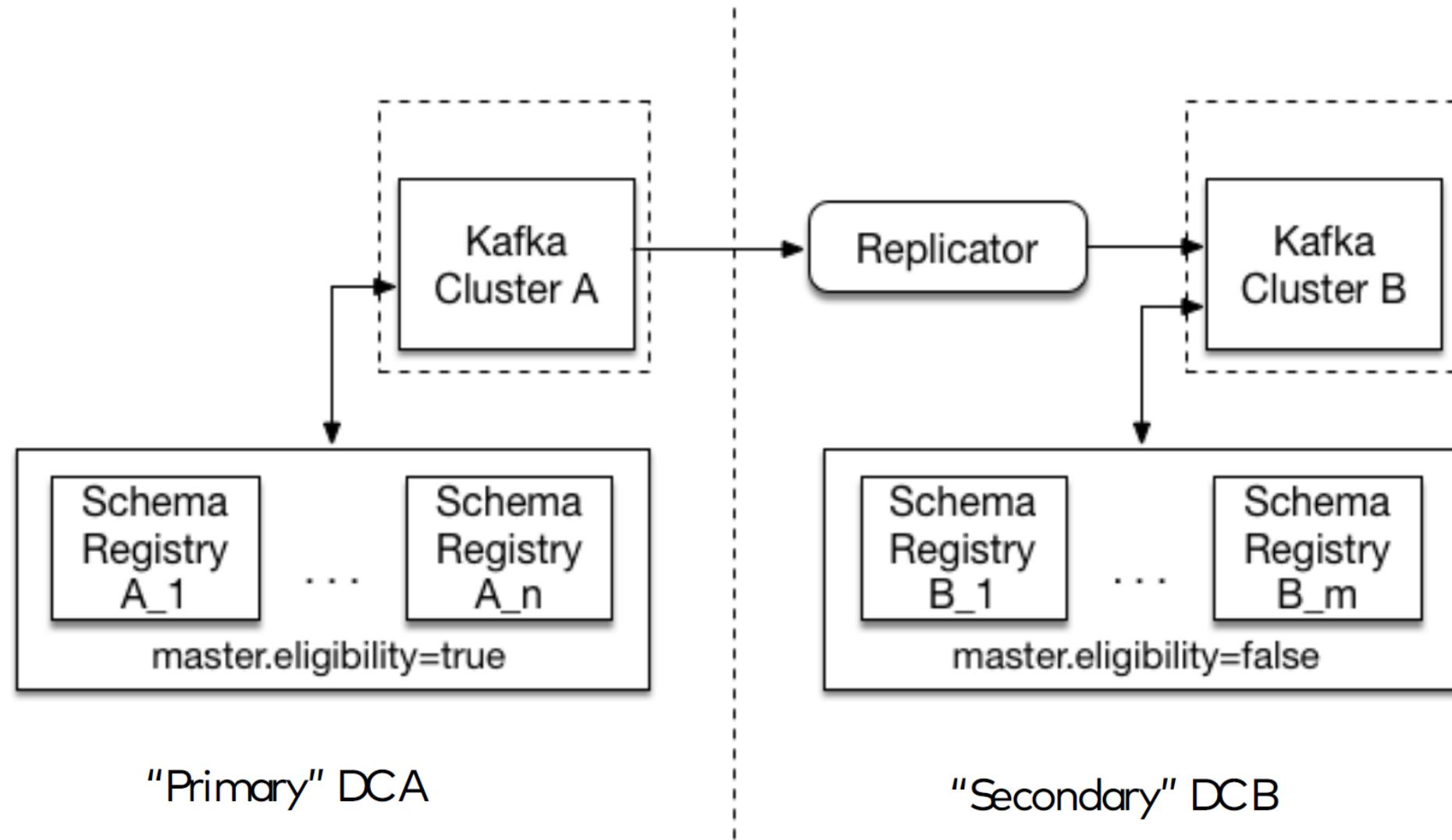
## Producer API:

- Replication process must use same partitioner class to preserve message ordering for keyed messages
- Topic properties must be same in source and destination clusters (automatic with Replicator)

## Consumer API:

- Use `offsetsForTimes()` method to find an offset for a given timestamp
- Use `seek()` to move to the desired offset

# Schema Registry Across Data Centers



## Improving Network Utilization

- If network latency is high, increase the TCP socket buffer size in Kafka
  - `socket.send.buffer.bytes` on the origin cluster's broker (default: 102400 bytes)
  - `receive.buffer.bytes` on Replicator's consumer (default: 65536 bytes)
  - Increase corresponding OS socket buffer size

# Appendix D: SSL and SASL Details

## Overview

This appendix contains two sections (as one appendix):

- SSL
- SASL



This section is a literal copy of old content - no formatting done, just making sure everything renders.

## Why is SSL Useful?

- Organization or legal requirements
- One-way authentication
  - Secure wire transfer using encryption
  - Client knows identity of Broker
  - Use case: Wire encryption during cross-data center mirroring
- Two-way authentication with **Mutual SSL**
  - Broker knows the identity of the client as well
  - Use case: authorization based on SSL principals
- Easy to get started
  - Just requires configuring the client and server
  - No extra servers needed, but can integrate with enterprise

## SSL Data Transfer

- After an initial handshake, data is encrypted with the agreed-upon encryption algorithm
- There is overhead involved with data encryption:
  - Overhead to encrypt/decrypt the data
  - Can no longer use zero-copy data transfer to the Consumer
  - SSL overhead will increase

```
kafka.network:type=RequestMetrics,name=ResponseSendTimeMs
```

## SSL Performance Impact

- Performance was measured on Amazon EC2 r3.xlarge instances

	Throughput(MB/s)	CPU on client	CPU on Broker
<b>Producer (plaintext)</b>	83	12%	30%
<b>Producer (SSL)</b>	69	28%	48%
<b>Consumer (plaintext)</b>	83	8%	2%
<b>Consumer (SSL)</b>	69	27%	24%

## Data at Rest Encryption

- "Data at rest encryption" refers to encrypting data stored on a hard drive that is currently not moving through the network
- Options for encrypting data at rest:
  1. Linux OS encryption utilities that encrypt full disk or disk partitions (e.g. LUKS)
  2. Producers encrypt messages before sending to Kafka, and Consumers decrypt after consuming
- Consider running Brokers on a machine with an encrypted filesystem or encrypted RAID controller

# SSL: Keystores and Truststores

Kafka client truststore.jks



CA certificate

Kafka Broker keystore.jks



Broker private key



Broker certificate  
with CA signature



- Client truststore:
  - CA certificate used to verify signature on Broker cert
  - If signature is valid, Broker is **trusted**
- Broker keystore:
  - Uses private key to create **certificate**
  - Cert must be signed by Certificate Authority (CA)
  - Cert includes public key, which client will use to establish secure connection

## Preparing for SSL

1. Generate certificate (X.509) in Broker **keystore**
2. Create your own Certificate Authority (CA) or use a well known CA
3. Sign Broker certificate with CA
4. Import signed certificate and CA certificate to Broker **keystore**
5. Import CA certificate to client **truststore**



Mutual SSL: generate client **keystore** and corresponding Broker **truststore** in the same way.

## SSL Everywhere! (1)

- Client and Broker `*.properties` Files:

```
ssl.keystore.location = /var/private/ssl/kafka.server.keystore.jks  
ssl.keystore.password = password-to-keystore-file  
ssl.key.password = password-to-private-key  
ssl.truststore.location = /var/private/ssl/kafka.server.truststore.jks  
ssl.truststore.password = password-to-truststore-file
```

- Brokers:

```
listeners = SSL://<host>:<port>  
ssl.client.auth = required  
inter.broker.listener.name = SSL
```

- Client:

```
security.protocol = SSL
```

## SSL Everywhere! (2)

### Discussion Questions:

- How could you secure the clear credentials that are stored in the `*.properties` files?
- What configurations would you change to do one-way SSL (SSL from Broker to client only) and plaintext between Brokers?

## SSL Principal Name

- By default, Principal Name is the distinguished name of the certificate

```
CN=host1.example.com,OU=organizational unit,O=organization,L=location,ST=state,C=country
```

- Can be customized through `principal.builder.class`
  - Has access to X509 Certificate
  - Makes setting the Broker principal and application principal convenient

## Troubleshooting SSL

- To troubleshoot SSL issues
  - Verify all Broker security configurations
  - Verify client security configuration
  - On the Broker, check that the following command returns a certificate:

```
$ openssl s_client -connect <broker>:<port> -tls1
```

- Enable SSL debugging using the `KAFKA_OPTS` environment variable
  - The output can be verbose but it will show the SSL handshake sequence, etc.
  - If you are debugging on the Broker, you must restart the Broker

```
$ export KAFKA_OPTS="-Djavax.net.debug=ssl $KAFKA_OPTS"
```

## SASL for Authentication (1)

- SASL: Simple Authentication and Security Layer
  - Challenge/response protocols
  - Server issues challenge; client sends response
  - Continues until server is satisfied
- All of the SASL authentication methods send data over the wire in **PLAINTEXT** by default, but SASL authentication can (should) be combined with **SSL** transport encryption.

## SASL for Authentication (2)

- SASL supports different mechanisms
  - GSSAPI: Kerberos
  - SCRAM-SHA-256, SCRAM-SHA-512: “salted” and hashed passwords
    - SCRAM (Salted Challenge Response Authentication Mechanism)
    - Can use username/password stored in ZK or delegation token (OAuth 2)
  - PLAIN: cleartext username/password
  - OAUTHBEARER: authentication tokens

## Using SASL SCRAM

- SCRAM should be used with SSL for secure authentication
- ZooKeeper is used for the credential store
  - Create credentials for Brokers and clients
  - Credentials must be created before Brokers are started
  - ZooKeeper should be secure and on a private network

## Configuring SASL SCRAM Credentials

- Create credentials for inter-Broker communication (user "admin")

```
$ kafka-configs --bootstrap-server broker_host:port \
  --alter \
  --add-config \
  'SCRAM-SHA-256=[password=admin-secret],SCRAM-SHA-512=[password=admin-secret]' \
  --user admin
```

- Create credentials for Broker-client communication (e.g. user "alice")

```
$ kafka-configs --bootstrap-server broker_host:port \
  --alter \
  --add-config \
  'SCRAM-SHA-256=[password=alice-secret],SCRAM-SHA-512=[password=alice-secret]' \
  --user alice
```

## Configuring SASL Using a JAAS File (Broker)

- JAAS (Java Authentication and Authorization Service) can be included in `*.properties` files on clients and Brokers using the `sasl.jaas.config` property

Broker (`server.properties`):

```
...
listeners=SASL_SSL://<host>:<port>
inter.broker.listener.name=SASL_SSL
sasl.mechanism.inter.broker.protocol=SCRAM-SHA-256
sasl.enabled.mechanisms=SCRAM-SHA-256

listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config= \
    org.apache.kafka.common.security.scram.ScramLoginModule required \
    username="admin" \
    password="admin-secret";
...
```

# Configuring SASL Using a JAAS File (Client)

## Client configuration file:

```
...
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256

sasl.jaas.config= \
    org.apache.kafka.common.security.scram.ScramLoginModule required \
    username="alice" \
    password="alice-secret";
...
```

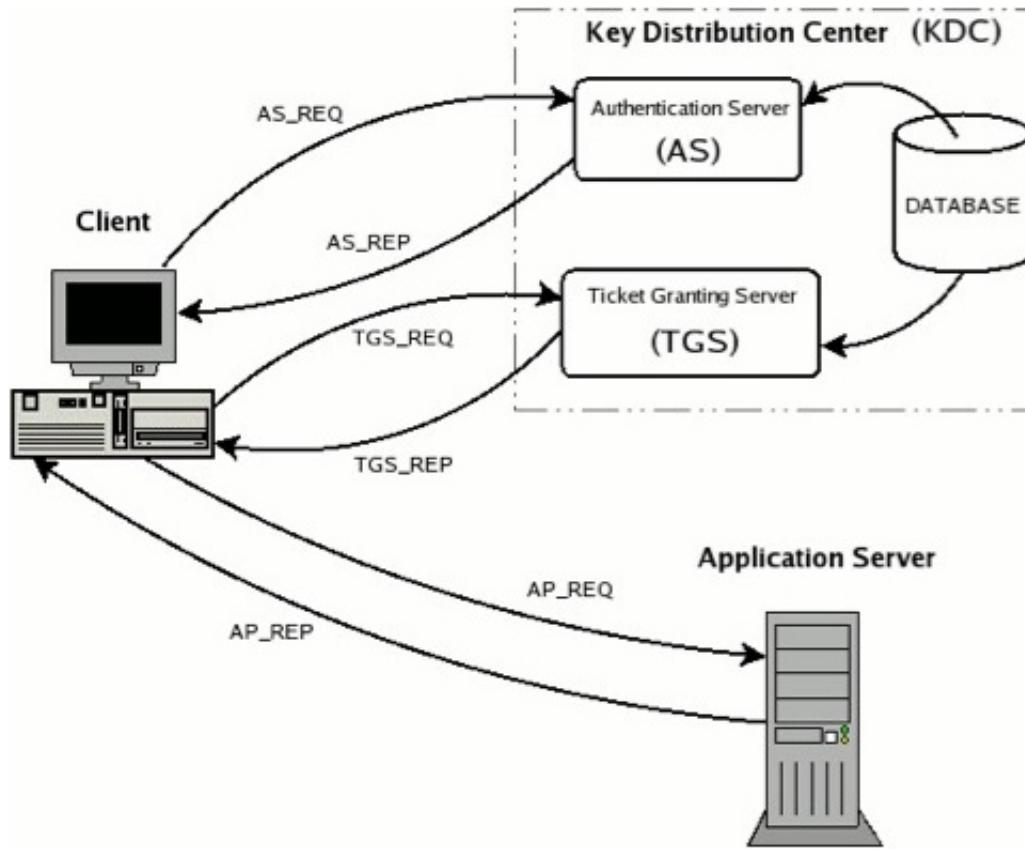
## Discussion questions:

- Why is `SASL_SSL` wire encryption recommended when using SASL SCRAM?
- What are the tradeoffs of an environment where clients authenticate with SASL SCRAM over SSL, but inter-Broker communication is done over plaintext? How would you change these `*.properties` files for this situation?
- Notice again that clear passwords are stored in these files. What would you do to ensure the security of these credentials?

## Why Kerberos?

- Kerberos provides secure single sign-on
  - An organization may provide multiple services, but a user just needs a single Kerberos password to use all services
- More convenient where there are many users
- Requires a Key Distribution Center (KDC)
  - Each service and each user must register a Kerberos principal in the KDC

# How Kerberos Works



1. Services authenticate with the Key Distribution Center on startup
  - a. Client authenticates with the Authentication Server on startup
  - b. Client obtains a service ticket from Ticket Granting Server
2. Client authenticates with the service using the service ticket

## Preparing Kerberos

- Create principals in the KDC for:
  - Each Kafka Broker
  - Application clients
- Create Keytabs for each principal
  - Keytab includes the principal and encrypted Kerberos password
  - Allows authentication without typing a password

## The Kerberos Principal Name

- Kerberos principal
  - Primary[/Instance]@REALM
  - Examples:
    - kafka/kafka1.hostname.com@EXAMPLE.COM
    - kafka-client-1@EXAMPLE.COM
- Primary is extracted as the default principal name
- Can customize the username through sasl.kerberos.principal.to.local.rules

# Configuring Kerberos (Broker)

Broker configuration file:

```
listeners = SASL_PLAINTEXT://<host>:<port>    # or SASL_SSL://<host>:<port>
inter.broker.listener.name=SASL_PLAINTEXT      # or SASL_SSL
sasl.kerberos.service.name=kafka

listener.name.sasl_plaintext.gssapi.sasl.jaas.config= \
  com.sun.security.auth.module.Krb5LoginModule required \
  useKeyTab=true \
  storeKey=true \
  keyTab="/etc/security/keytabs/kafka_server.keytab" \
  principal="kafka/kafka1.hostname.com@EXAMPLE.COM";
```

# Configuring Kerberos (Client)

## Client configuration file:

```
...
security.protocol=SASL_PLAINTEXT      # or SASL_SSL
sasl.kerberos.service.name=kafka

sasl.jaas.config= \
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/etc/security/keytabs/kafka_client.keytab"
    principal="kafka-client-1@EXAMPLE.COM";
...
```

## Advanced Security Configurations

- Configure multiple SASL mechanisms on the Broker
  - Useful for mixing internal (e.g., GSSAPI) and external (e.g., SCRAM) clients

```
sasl.enabled.mechanisms = GSSAPI,SCRAM-SHA-256
```

- Configure different listeners for different sources of traffic
  - Useful to designate one interface for clients and one interface for replication traffic:

```
listeners = CLIENTS://kafka-1a:9092,REPLICATION://kafka-1b:9093
```

- Listeners can have any name as long as `listener.security.protocol.map` is defined to map each name to a security protocol:

```
listener.security.protocol.map=CLIENTS:SASL_SSL,REPLICATION:SASL_PLAINTEXT
```

- Specify listener for communication between brokers and controller with `control.plane.listener.name`

## Troubleshooting SASL

- To troubleshoot SASL issues
  - Verify all Broker security configurations
  - Verify client security configuration
  - Verify JAAS files and passwords
  - Enable SASL debugging for Kerberos using the `KAFKA_OPTS` environment variable
    - If you are debugging on the Broker, you will have to restart the Broker

```
$ export KAFKA_OPTS="-Dsun.security.krb5.debug=true"
```