CONFLUENT

kafka KRU

2021년 3rd Meetup

Apache Kafka 3.0 Update & Confluent Platform 7.0

About me

황주필
(jhwang@confluent.io)

Senior Solutions Engineer
Confluent Korea

Github: jupilhwang
Twitter: @jupil_hwang

- BEA Systems
- Oracle
  - Middleware / SOA
  - Cloud
- Pivotal
  - Agile, DDD, TDD
  - Cloud-Native Application / MSA / EDA
  - Container Based PaaS (Cloud Foundry, Kubernetes)
- Confluent
  - Event Streaming Platform based on Kafka
  - Data Pipeline / Data Mesh
  - Stream Processing with KStream / ksqlDB

Agenda

- Apache 3.0 Update
- KRaft mode
- Confluent Platform/Cloud 차별 기능
- Q&A

Parallel Consumer

CONFLUENT PRODUCTS SOLUTIONS LEARN DEVELOPERS GET STARTED FREE US English

CLIENTS

Introducing the Confluent Parallel Consumer

ANTONY STUBBS

DECEMBER 15, 2020

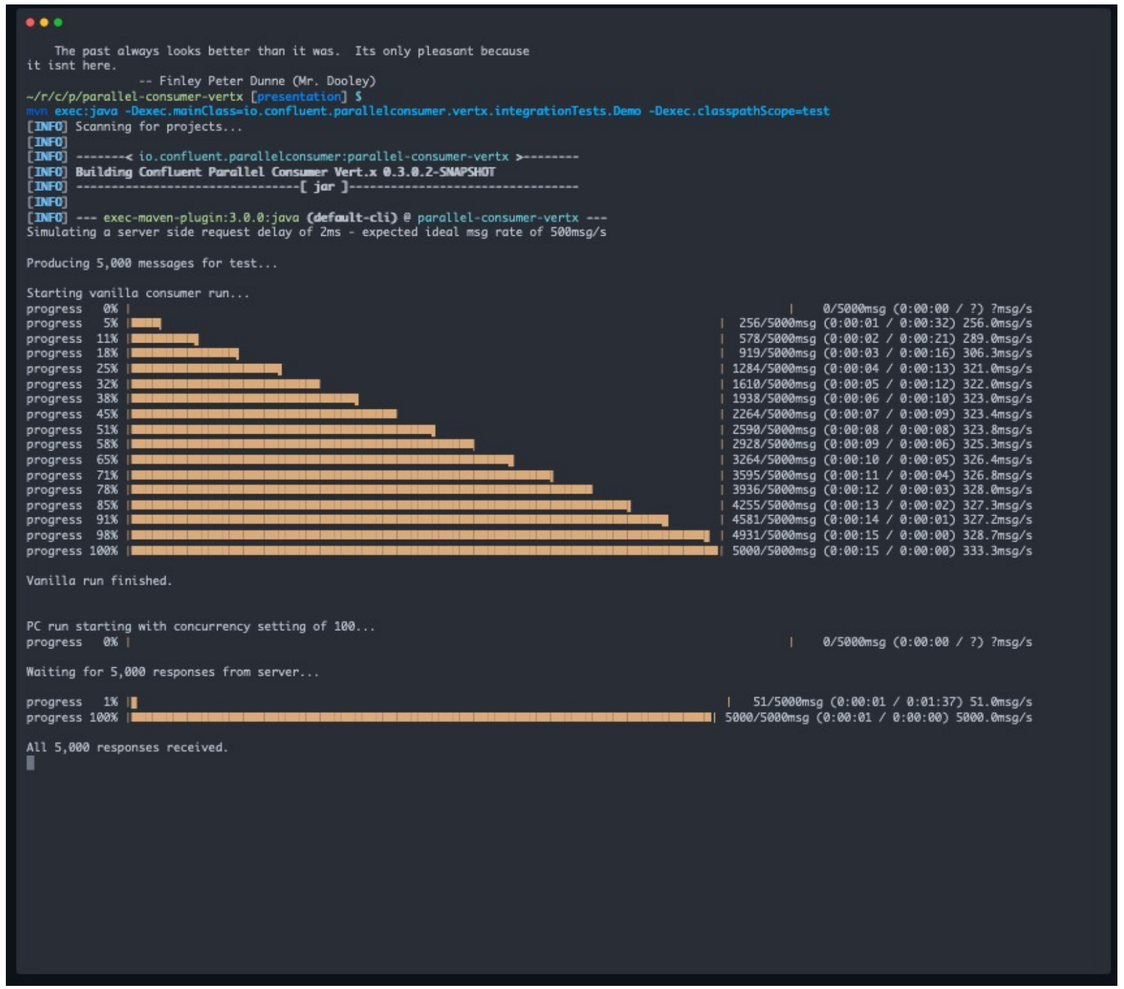
Consuming messages in parallel is what Apache Kafka® is all about, so you may well wonder, why would we want anything else? It turns out that, in practice, there are a number of situations where Kafka's partition-level parallelism gets in the way of optimal design. For example, when partition counts are fixed for a reason beyond your control, you need to call other databases or microservices—which can take a while to respond—or use queue-like semantics, where slow-to-process messages don't hold up faster ones further back in the queue. These are just a few of the reasons why we wrote the Confluent Parallel Consumer, which provides an alternate approach to parallelism that subdivides the unit of work from a partition down to a key or even a message.

In essence, the Parallel Consumer is a JVM-based, Apache 2.0 client library that includes everything you'd expect in regular Kafka consumers: consumer groups, transactions/exactly-once semantics, etc., but also three new features in addition to these.

First, the Parallel Consumer makes it easy to process messages with a higher level of parallelism than the number of partitions for the input data. The Parallel Consumer also lets you define parallelism in terms of key-level ordering guarantees, rather than the coarser-grained, partition-level parallelism that comes with the Kafka consumer groups. It does this using a thread pool, with the library handling all the tricky bookkeeping required in Kafka. By switching from partition-level parallelism to key-level parallelism, you don't have to over-provision topic partitions or change the ones you have just so you can scale your consumer group out.

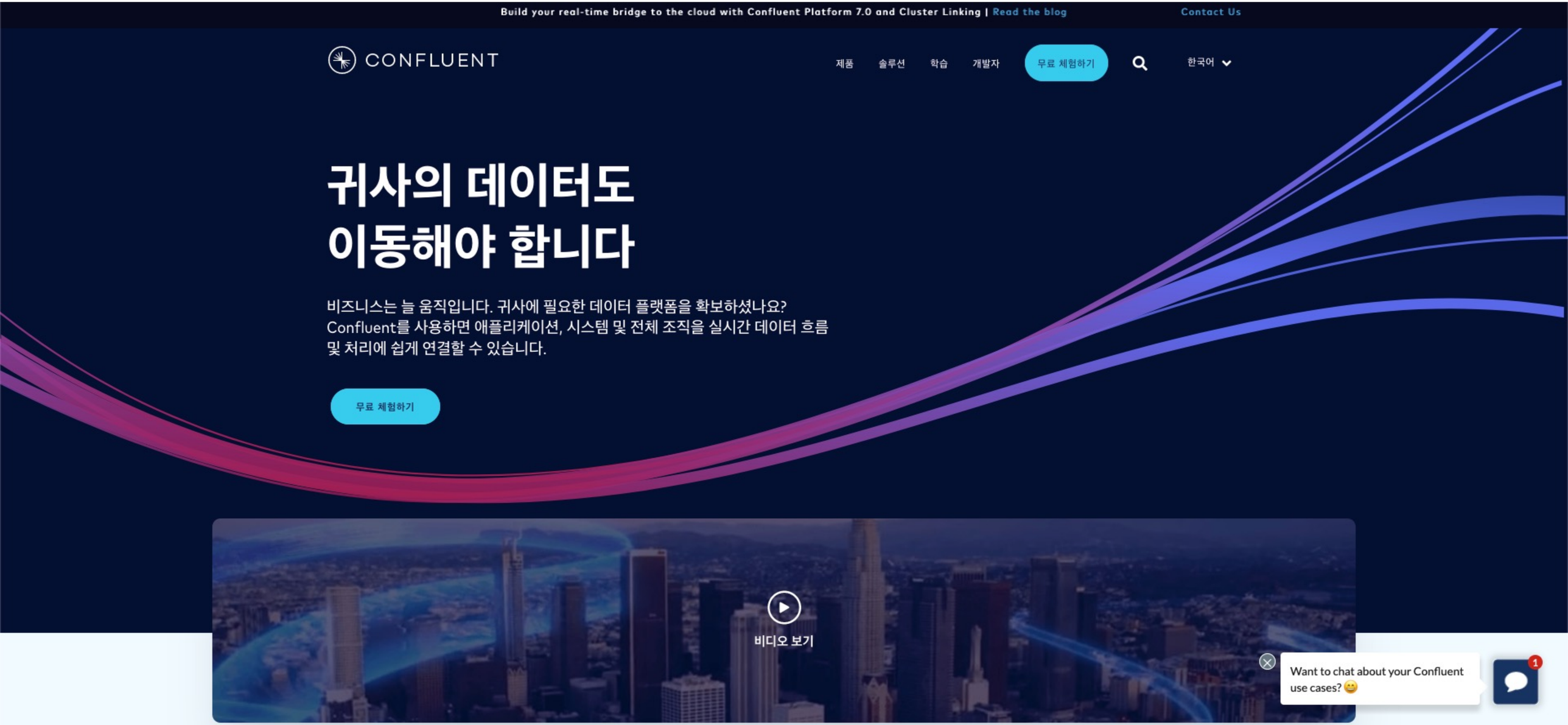
Second, the Parallel Consumer makes it easy for you to call out to other services efficiently without stalling your application. For instance, if you need to look up customer details from a database or while you are

https://github.com/confluentinc/parallel-consumer



Confluent 한글 홈페이지 오픈

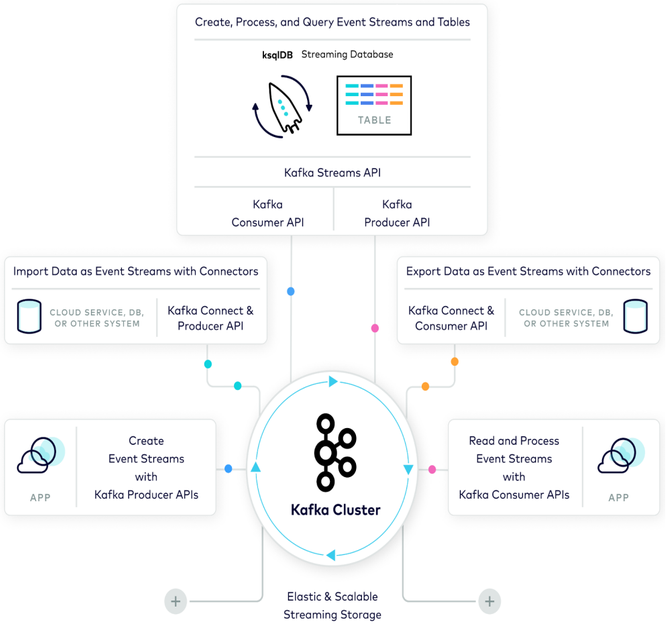
https://confluent.io/ko-kr



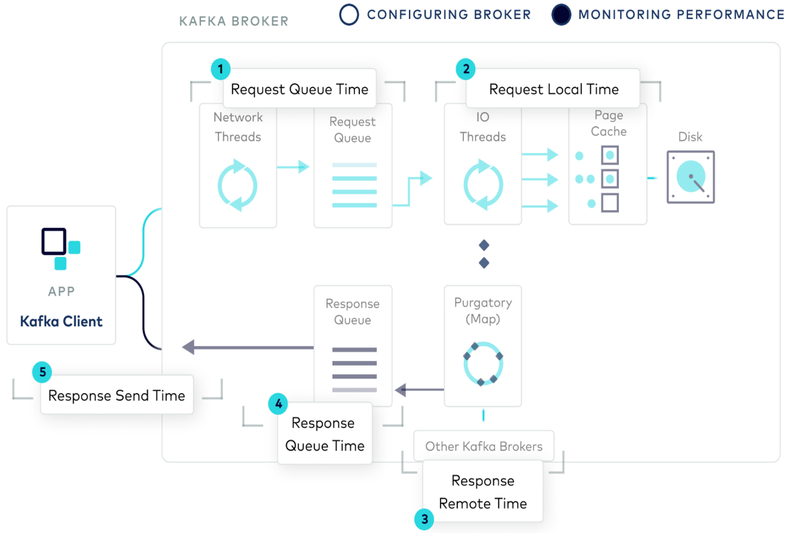
Kafka 개발자 사이트

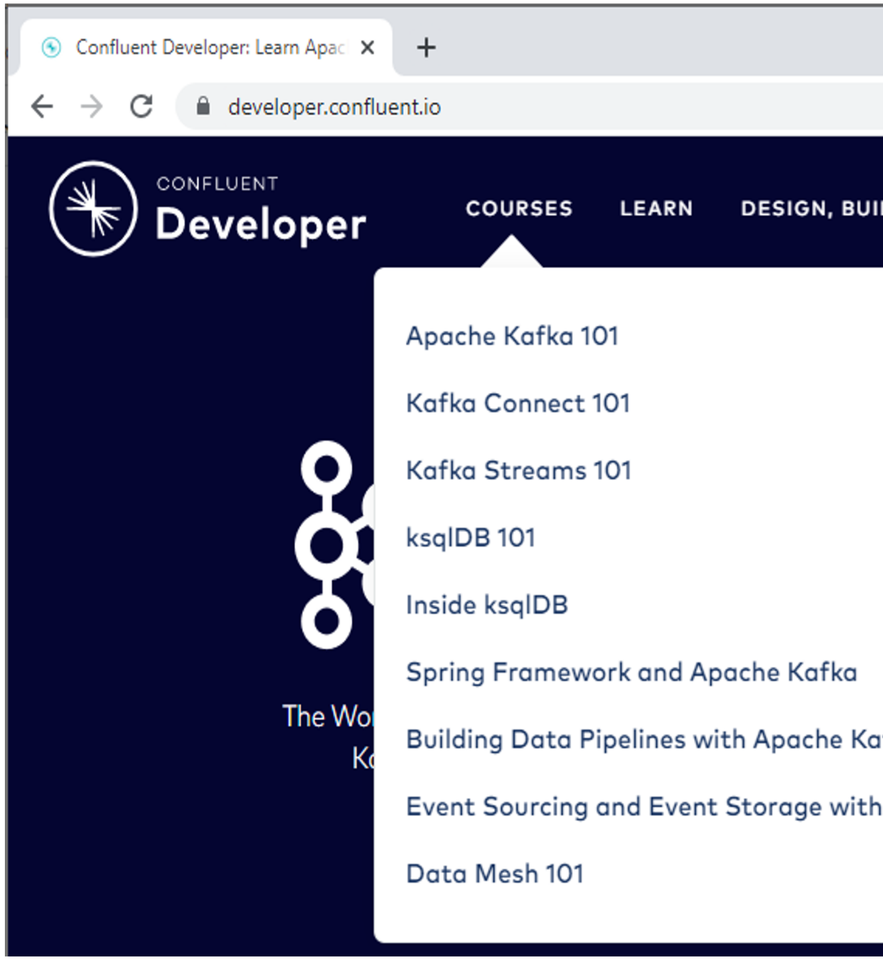
https://developer.confluent.io

Kafka high-level architecture



Kafka Broker Internals





*** 각 코스에서는 Confluent Cloud 를 무료로 사용할 수 있는 Promo Code 를 제공합니다. (예, KAFKA101 etc..)**

Apache 3.0 – What's New

2021.09.21

What's New in Apache Kafka 3.0.0

I'm pleased to announce the release of Apache Kafka 3.0 on behalf of the Apache Kafka® community. Apache Kafka 3.0 is a major release in more ways than one. Apache Kafka 3.0 introduces a variety of new features, breaking API changes, and improvements to KRaft—Apache Kafka's built-in consensus mechanism that will replace Apache ZooKeeper™.

While KRaft is not yet recommended for production (list of known gaps), we have made many improvements to the KRaft metadata and APIs. Exactly-once and partition reassignment support are worth highlighting. We encourage you to check out KRaft's new features and to try it out in a development environment.

Starting with Apache Kafka 3.0, the Producer enables the strongest delivery guarantees by default (`acks=all`, `enable.idempotence=true`). This means that users now get ordering and durability by default.

Also, don't miss the Kafka Connect task restart enhancements, Kafka Streams improvements in timestamp-based synchronization, and MirrorMaker2's more flexible configuration options.

To review the full list of features and enhancements, be sure to read the release notes. You can also watch the release video for a summary of what's new in Apache Kafka 3.0.0.

Universal changes

KIP-750 (Part I): Deprecate support for Java 8 in Kafka

Support for Java 8 is deprecated across all components of the Apache Kafka project in 3.0. This will give users time to adapt before the next major release (4.0), when Java 8 support is planned to be removed.

KIP-751 (Part I): Deprecate support for Scala 2.12 in Kafka

Support for Scala 2.12 is also deprecated everywhere in Apache Kafka 3.0. As with Java 8, we're giving users time to adapt because support for Scala 2.12 is planned to be removed in the next major release (4.0).

Kafka Broker, Producer, Consumer and AdminClient

KIP-630: Kafka Raft Snapshot

A major feature that we are introducing with 3.0 is the ability for KRaft Controllers and KRaft Brokers to generate, replicate, and load snapshots for the metadata topic partition named `__cluster_metadata`. This topic is used by the Kafka Cluster to store and replicate metadata information about the cluster like Broker configuration, topic partition assignment, leadership, etc. As this state grows, Kafka Raft Snapshot provides an efficient way to store, load, and replicate this information.

KIP-746: Revise KRaft Metadata Records

Experience and continuous development since the first version of the Kafka Raft Controller have surfaced the need to revise a few of the metadata record types that are used when Kafka is configured to run without ZooKeeper (ZK).

KIP-730: Producer ID generation in KRaft mode

With 3.0 and KIP-730 the Kafka Controller is now completely taking over the responsibility of generating a Kafka Producer ID. The Controller is doing so both in ZK and KRaft modes. This takes us closer to the bridge release, which will allow users to transition from Kafka deployments that use ZK to new deployments that use KRaft.

KIP-679: Producer will enable the strongest delivery guarantee by default

Starting with 3.0, the Kafka Producer turns on by default idempotency and the acknowledgement of delivery by all of the replicas. This makes record delivery guarantees stronger by default.

KIP-735: Increase default consumer session timeout

The default value of the Kafka Consumer's configuration property `session.timeout.ms` is increased from 10 seconds to 45 seconds. This will allow the Consumer to adapt better by default to transient network failures and avoid consecutive rebalances when a Consumer appears to leave the group only temporarily.

KIP-709: Extend OffsetFetch requests to accept multiple group ids

Requesting the current offsets of a Kafka Consumer group has been possible for quite some time. But fetching the offsets of multiple consumer groups requires an individual request for each group. In 3.0 and with KIP-709, the fetch and AdminClient APIs are extended to support reading the offsets of multiple consumer groups at the same time within a single request/response.

KIP-699: Update FindCoordinator to resolve multiple Coordinators at a time

Supporting operations that can be applied to multiple consumer groups at the same time in an efficient way heavily depends on the ability of the clients to discover the coordinators of these groups efficiently. This becomes possible with KIP-699, which adds support for discovering the coordinators for multiple groups with one request. Kafka clients have been updated to use this optimization when talking to new Kafka Brokers that support this request.

KIP-724: Drop support for message formats v0 and v1

Four years since its introduction in June 2017 with Kafka 0.11.0, message format v2 has been the default message format. Thus, with enough water (or streams if you may) having flowed under the bridge, the major release of 3.0 gives us a good opportunity to deprecate the older message formats—namely v0 and v1. These formats are rarely in use today. With 3.0, users will get a warning if they configure their Brokers to use the message formats v0 or v1. This option will be removed in Kafka 4.0 (see KIP-724 for details and implications from the deprecation of v0 and v1 message formats).

KIP-707: The future of KafkaFuture

When the `KafkaFuture` type was introduced to facilitate the implementation of the Kafka AdminClient, pre-Java 8 versions were still in widespread use and Java 7 was officially supported by Kafka. Fast forward to a few years later and now Kafka runs on Java versions that support the `CompletionStage` and `CompletableFuture` class types. With KIP-707, `KafkaFuture` adds a method to return a `CompletionStage` object and in that way enhances the usability of `KafkaFuture` in a backwards compatible way.

KIP-466: Add support for List<T> serialization and deserialization

KIP-466 adds new classes and methods for the serialization and deserialization of generic lists—a feature useful to Kafka clients and Kafka Streams alike.

KIP-734: Improve AdminClient.listOffsets to return timestamp and offset for the record with the largest timestamp

The users' capabilities to list offsets of Kafka topic/partitions have been extended. With KIP-734, users can now ask the AdminClient to return the offset and timestamp of the record with the highest timestamp in a topic/partition. (This is not to be confused with what the AdminClient returns already as the latest offset—which is the offset of the next record to be written in the topic/partition.) This extension to the existing ListOffsets API allows users to probe the liveliness of a partition by asking which is the offset of the most recent record written and what its timestamp is.

Kafka Connect

KIP-745: Connect API to restart connector and tasks

In Kafka Connect a connector is represented during runtime as a group of a `Connector` class instance and one or more `Task` class instances, and most operations on connectors available through the Connect REST API can be applied to the group as a whole. A notable exception since the beginning has been the restart endpoints for the `Connector` and `Task` instances. To restart the connector as a whole, users had to make individual calls to restart the `Connector` instance and the `Task` instances. In 3.0, KIP-745 gives the ability to the users to restart either all or only the failed of a connector's `Connector` and `Task` instances with a single call. This feature is an add-on capability and the previous behavior of the `restart` REST API remains unchanged.

KIP-738: Removal of Connect's internal converter properties

Following their deprecation in the previous major release (Apache Kafka 2.0), `internal.key.converter` and `internal.value.converter` are removed as configuration properties and prefixes in the Connect worker's configuration. Moving forward, internal Connect topics will exclusively use the `JsonConverter` to store records without embedded schemas. Any existing Connect clusters that used different converters will have to port their internal topics to the new format (see KIP-738 for details on the upgrade path).

언어 지원 중단

- KIP-750: Drop support for Java 8 in Kafka 4.0 (deprecate in 3.0)
- KIP-751: Drop support for Scala 2.12 in Kafka 4.0 (deprecate in 3.0)

- **Scala 2.12** 와 **Java 8** 지원 중단됨
- 다음 Apache Kafka 4.0 버전에서 Scala 2.12 와 Java 8 지원은 완전히 삭제될 예정

KIP-679: Producer will enable the strongest delivery guarantee by default

- **enable.idempotence = true** 와 **acks=all** 이 기본 설정이 됨.
- 메시지의 순서 보장, 중복 방지 그리고 안전성 증가

Producer Config Default	Old Setting	New Setting
enable.idempotence	false	true
acks	1	all

KIP-745: Connect API to restart connector and tasks

What

Gives users the option to restart a connector's tasks when restarting the connector. The user can choose to restart all the tasks or just the failed tasks when calling the restart API for a connector

Why

Previously, when a connector or one of its tasks failed, users would call the connector restart API expecting that both the connector and its tasks would restart. However, calling the connector restart API only restarted the connector instance leading to failed tasks not being restarted.

For Whom

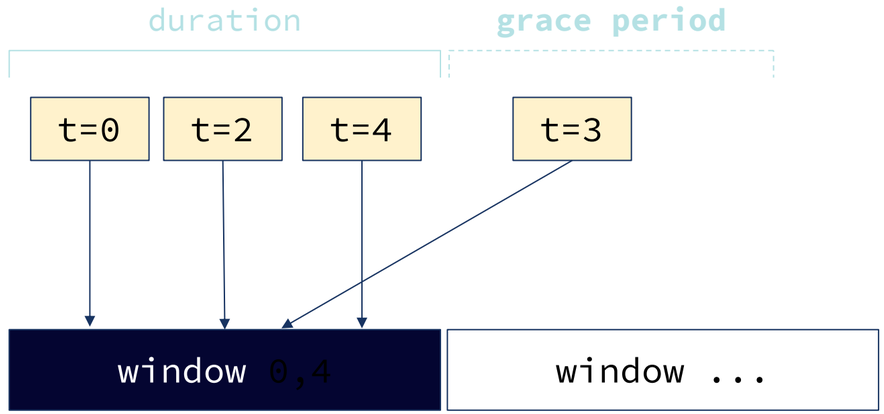
Developers/operators

Example Usage:

```
POST /connectors/my-connector/restart?includeTasks=true&onlyFailed=true
202 ACCEPTED
{
    "name": "my-connector",
    "connector": {
        "state": "RUNNING",
        "worker_id": "fakehost1:8083"
    },
    "tasks":
    [
        {
            "id": 0,
            "state": "RUNNING",
            "worker_id": "fakehost2:8083"
        },
        {
            "id": 1,
            "state": "RESTARTING",
            "worker_id": "fakehost3:8083"
        },
        {
            "id": 2,
            "state": "RESTARTING",
            "worker_id": "fakehost1:8083"
        }
    ]
}
```

KIP-633: Remove default 24-hour grace period used in windows

- The grace period is the length of time a window accepts late-arriving data.
- In some cases, like when using suppression, the 24-hour default is too long. This can lead to confusing behavior.
- In Kafka Streams, the grace period is a fundamental concept – not a hidden or advanced feature, so it makes sense to choose this duration explicitly.



KIP-500 (Preview)

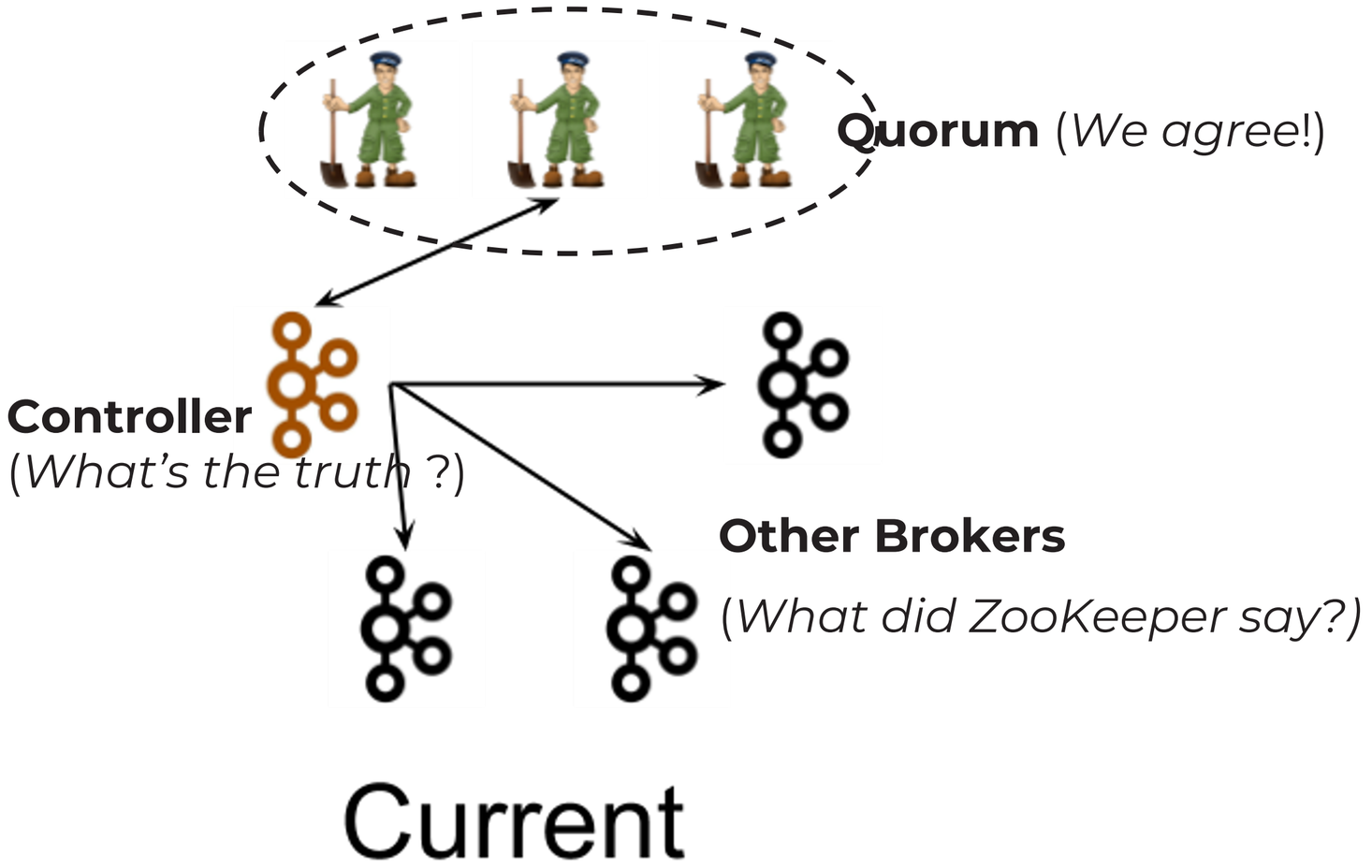
- 지금 바로 이 기능을 사용해 보고 ZK 없이 클러스터를 가동할 수 있다:
- https://github.com/apache/kafka/blob/3.0/config/kraft/README.md

* 일정은 변경될 수 있음

Kafka 와 ZooKeeper: 현재 구조

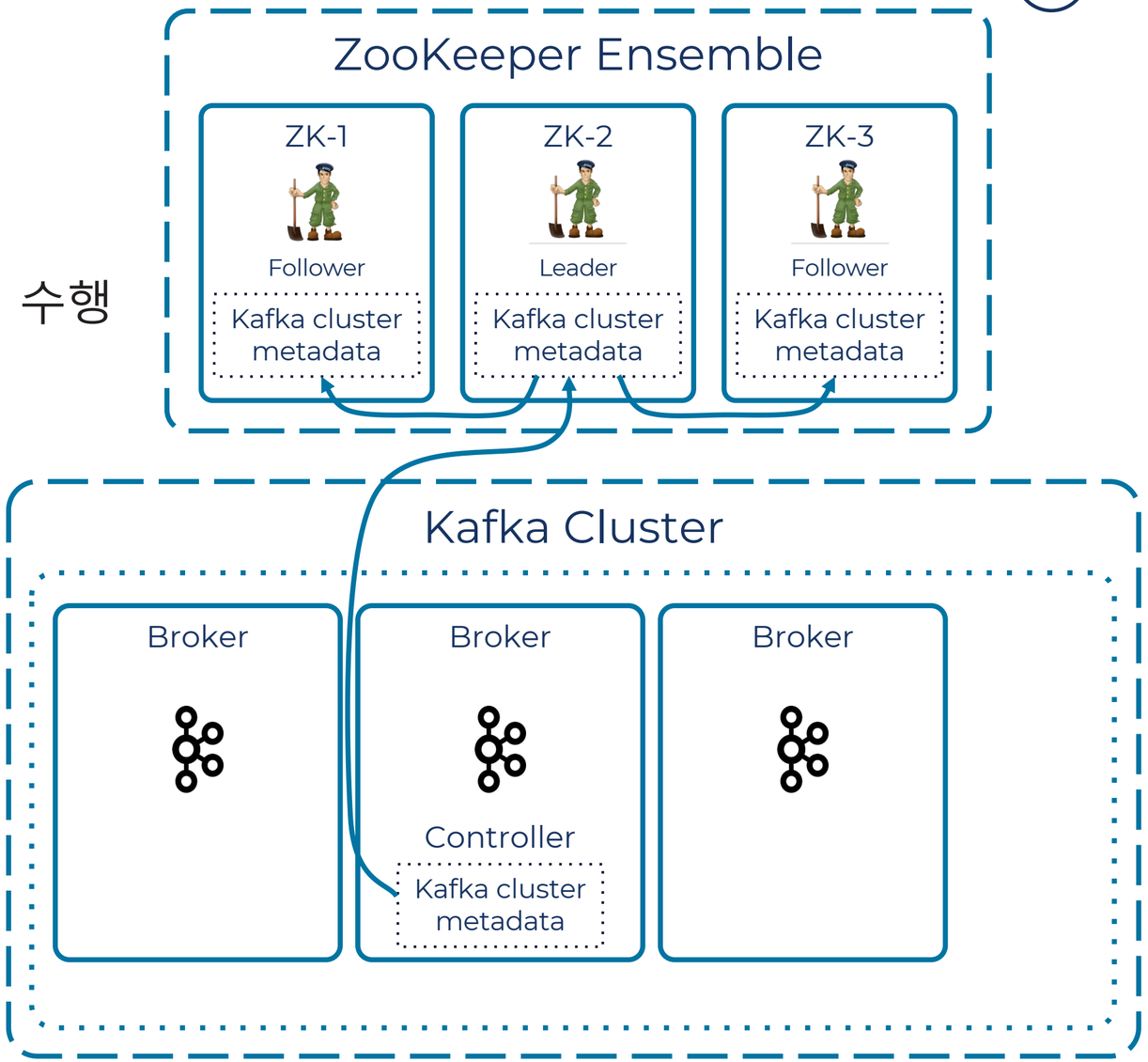
ZooKeeper : 메타데이터 와 클러스터 상태의 신뢰할 수 있는 저장소

Controller : Zookeeper 와 메타데이터 동기화
(모든 브로커,토픽,파티션,리더 등의 현재 상태)



Kafka Cluster Controller

- 파티션 및 복제본의 상태 관리
- 토픽 생성 시 브로커에 파티션 할당과 같은 관리 작업 수행
- 브로커의 동작 상태 모니터링
- 브로커 실패 시 새로운 파티션 리더 선출
- 새로운 리더를 브로커에게 전달



Kafka 와 ZooKeeper 구조

문제점? ZK와 동작에는 아무런 문제가 없습니다. 하지만, 좀 더 좋은 방법이 있습니다.

비효율적인 면:

- 운영 복잡성 증대
- 제한적인 확장성
- 추가적인 Java 프로세스
- ZooKeeper 의 데이터는 Controller 에 동기화 됨 (이중 캐싱, 동기화 문제 발생 가능성)

ZooKeeper

ZooKeeper solves many important **distributed systems challenges**, including quorum and metadata management.

However, ZooKeeper has become **unnecessarily complex to manage** as it requires tuning, configuring, monitoring, and securing performance across ***two*** log implementations, ***two*** network layers, and ***two*** security implementations, each with distinct tools and monitoring hooks.

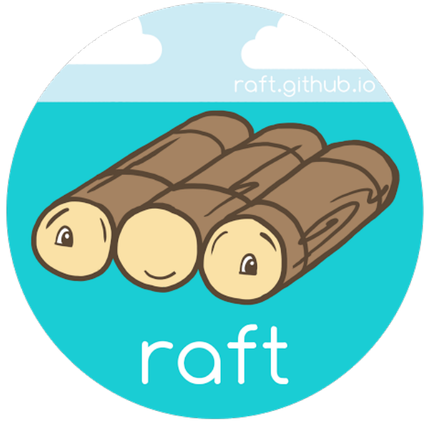
KIP-500 이란?

KIP-500: Move beyond ZK as single source of truth

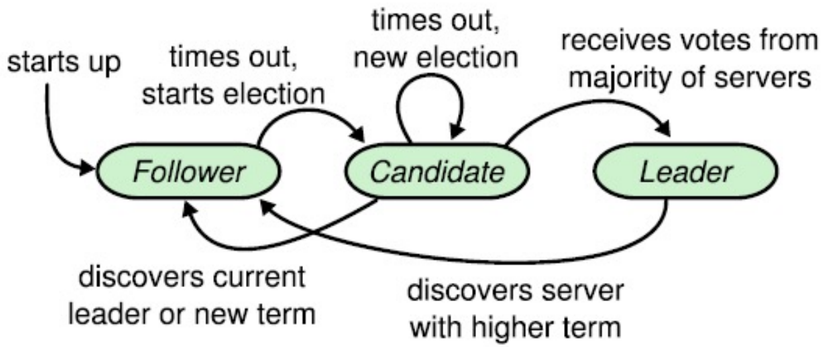
Kafka 자체에 저장된 메타데이터로 “Kafka on Kafka” 구현

내부 쿼럼(Quorum) 서비스는 _confluent-metada 토픽에 메타데이터를 가지는 **KRaft (Kafka Raft)** 모드 사용

Raft : raft.github.io

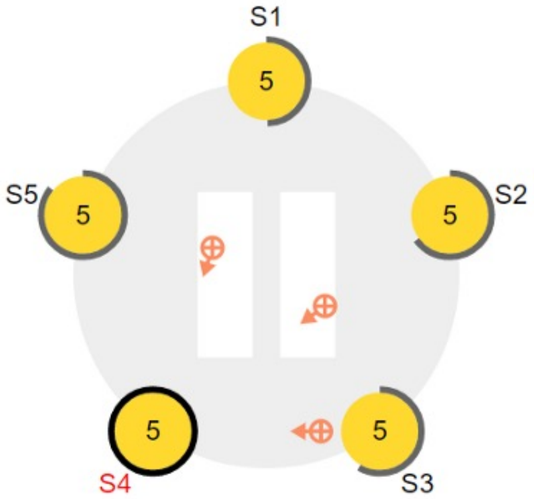


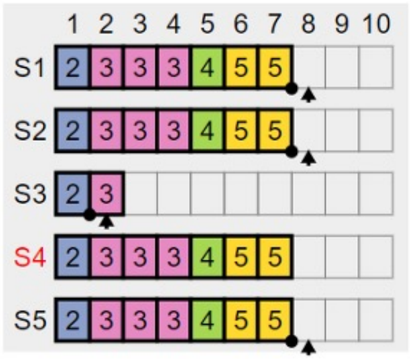
- 분산 컴퓨팅 시스템들에서 보편적으로 사용하는 합의(Consensus) 알고리즘
  - etcd, Scylla, mongoDB, CochroachDB, kudu 등 에서 다양하게 사용



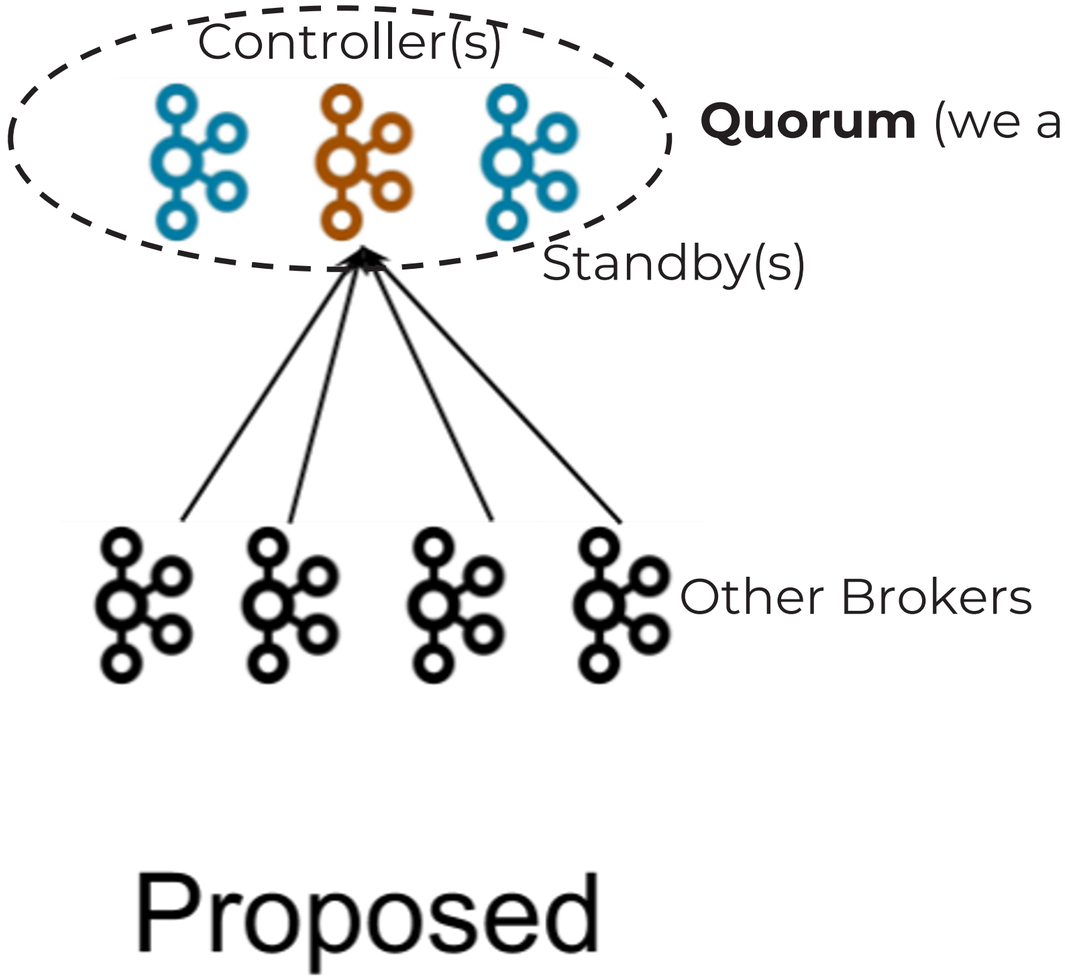
- 비교
  - Raft
  - Paxos
  - PoW (Proof of Work)
  - PoS (Proof of Stake)
  - Practical Byzantine Fault Tolerance (PBFT)
  - 등…

Raft Visualization





Kafka Without ZooKeeper



Proposed

KRaft nodes: 설정에 따라 Broker, Controller 또는 둘 다 될 수 있다. Controller nodes는 Active Controller 리더 선출

Controller 는 완전 자율로 동작

Nodes 는 메타데이터 토픽과 동기화하고 **빠른 복구** 를 위해 snapshots 저장

Controller 의 변경/시작 등에서 단지 로그를 읽고 상태를 맞춤

Nodes 는 **hot standby** 상태로 동작

장점

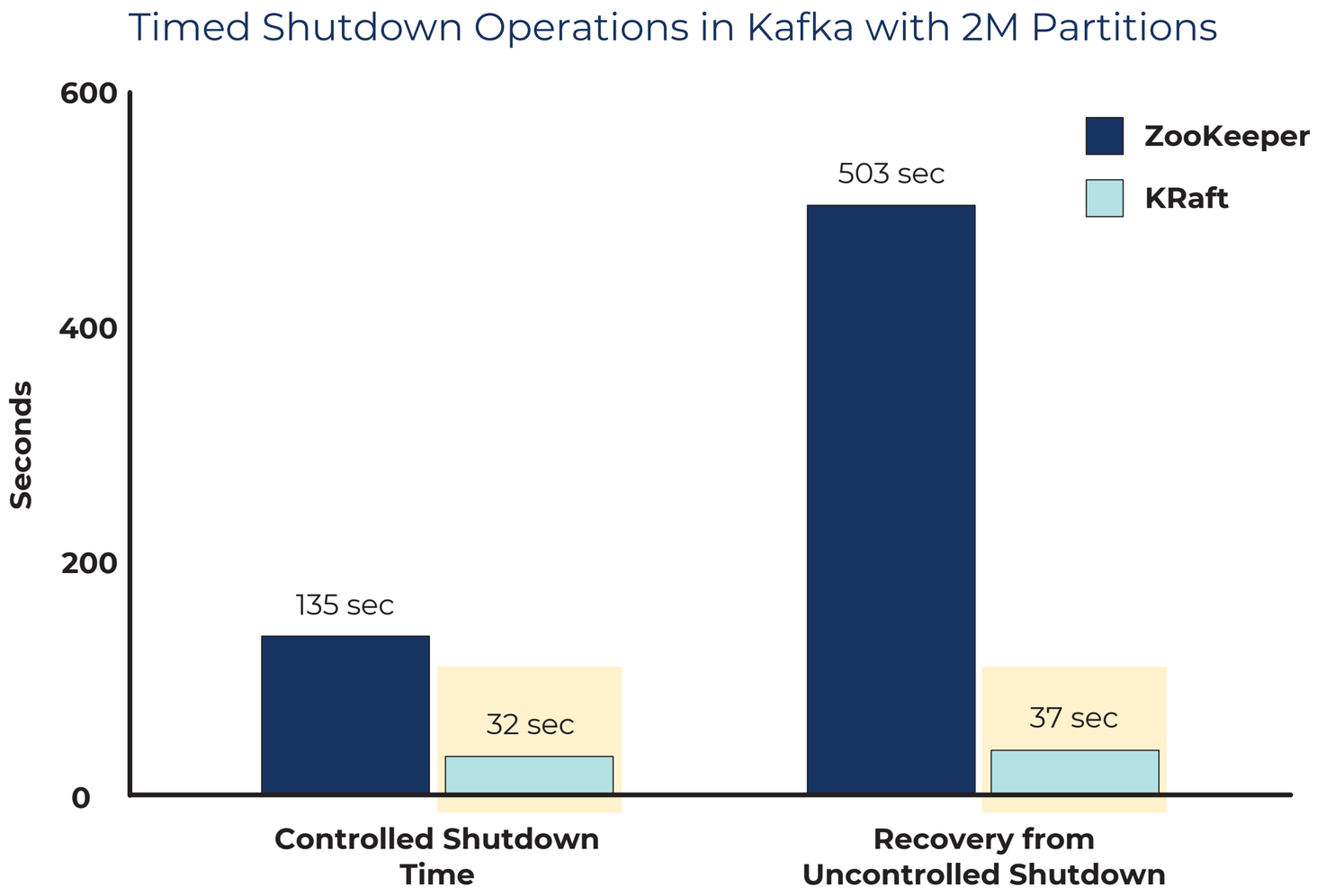
- Kafka의 확장성 향상
- 수백만 개의 토픽/파티션 지원 가능
- 더 빠른 시작 / 운영 경험
- 보안 향상 (단일 보안 모델/구성)
- 컨트롤러 로직 단순화
  - 컨트롤러가 상태를 유지하고 상태변경사항을 브로커에 전파해야 했다
  - KRaft를 사용하면 상태 전파는 항상 메타데이터 로그에서 이루어 진다.

Scale Kafka to millions of partitions by removing its dependency on ZooKeeper with KRaft

KRaft (preview) greatly simplifies Kafka's architecture by consolidating responsibility for metadata management into Kafka itself, rather than dividing it between two different systems (ZooKeeper and Kafka)

- ***Simplified deployment:*** Improves stability and makes it easier to administer, monitor, and support Kafka clusters
- ***Enhanced scalability****:* Enables clusters to scale to millions of partitions and achieve up to a 10x improvement in recovery time

Note - this feature is in preview and not yet supported for production workloads



AK 3.0 server.properties

server.properties

- node.id=0 # broker.id
- process.roles=broker,controller
- controller.quorum.voters=<controller-id>@<controller-ip>:<controller-port>, <controller-id>@<controller-ip>:<controller-port>

process.roles	Description
broker, controller	server acts as both a broker and a controller in KRaft mode
controller	server acts as controller in KRaft mode
broker	server acts as broker in KRaft mode
not set	server is assumed to be part of a Kafka cluster in ZooKeeper-based mode

controller quorum은 **`controller.quorum.voters`** 로 정의하며, 클러스터내의 모든 서버는 이 설정을 가져야 함

AK 3.0 – kafka-storage

$ kafka-storage random-uuid

$ kafka-storage format –t <cluster-uuid> -c <kafka-server-properites>

```
❯ kafka-storage random-uuid
f3QsyR3eRumgB1wo7Ko-xg
 ~/works/kafka/kafka_2.13-3.0.0/bin
❯ kafka-storage format -t f3QsyR3eRumgB1wo7Ko-xg -c ~/works/kafka/kafka_2.13-3.0.0/config/kraft/controller.properties
Formatting /tmp/raft-controller-logs
 ~/works/kafka/kafka_2.13-3.0.0/bin
❯ cat /tmp/raft-controller-logs/meta.properties
#
#Mon Nov 29 17:12:54 KST 2021
node.id=1
version=1
cluster.id=f3QsyR3eRumgB1wo7Ko-xg
```

$ kafka-metadata-shell --snapshot /tmp/raft-controller-logs/__cluster-metadata-0/000000000.log

$ kafka-server-start config/kraft/server.properties

AK 2.8 에서 @metadata 가 __cluster-metadata 로 변경됨

core/src/main/scala/kafka/server/KafkaRaftServer.scala

```
@@ -128,7 +128,7 @@ class KafkaRaftServer(
 }

 object KafkaRaftServer {
-  val MetadataTopic = "@metadata"
+  val MetadataTopic = "__cluster_metadata"
   val MetadataPartition = new TopicPartition(MetadataTopic, 0)
   val MetadataTopicId = Uuid.METADATA_TOPIC_ID
```

Kafka Raft (KRaft) Mode

KRaft 모드는 Raft 합의 프로토콜을 기반으로 함

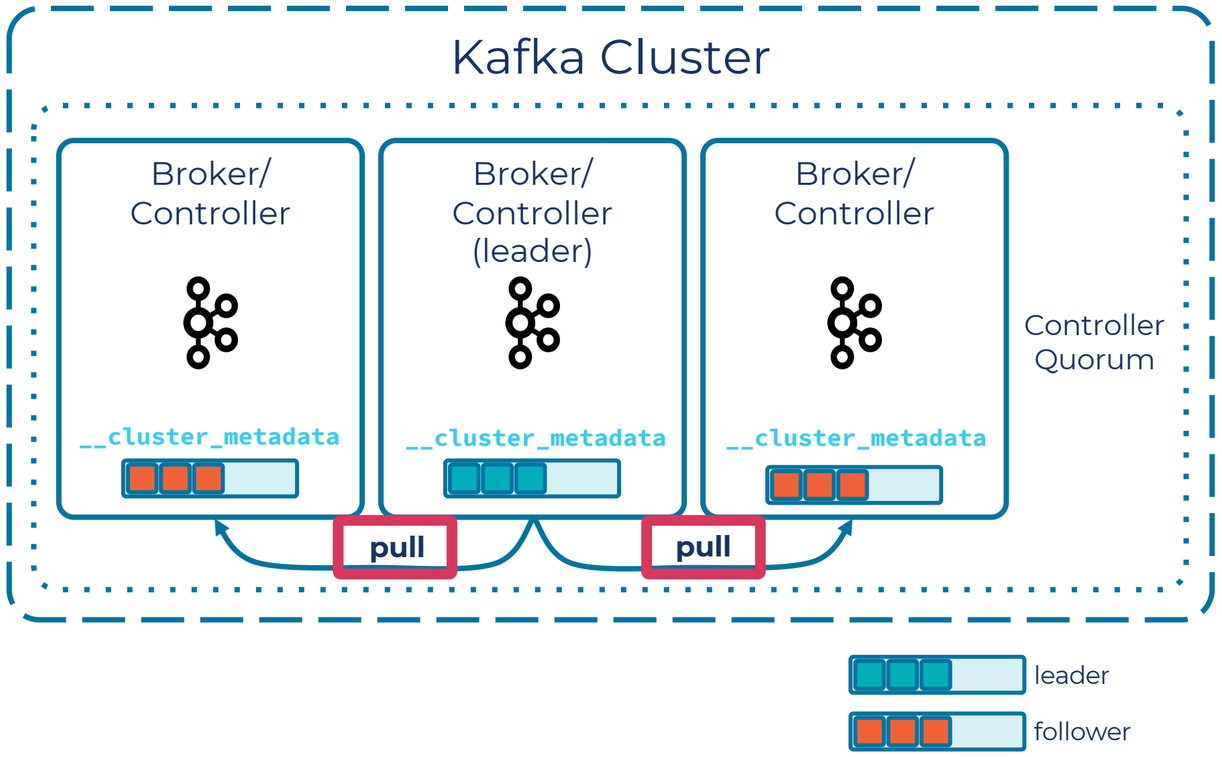
- KIP-500 및 기타 KIP의 일부로 도입됨

KRaft는 이벤트 기반이라는 점에서 Raft와 다름

- 단일 파티션 토픽를 사용하여 컨트롤러 노드 및 브로커 간에 변경 사항(상태 동기화)을 전달

메타데이터 업데이트는 리더가 푸시하는 것이 아니라 팔로워가 가져감(Fetch).

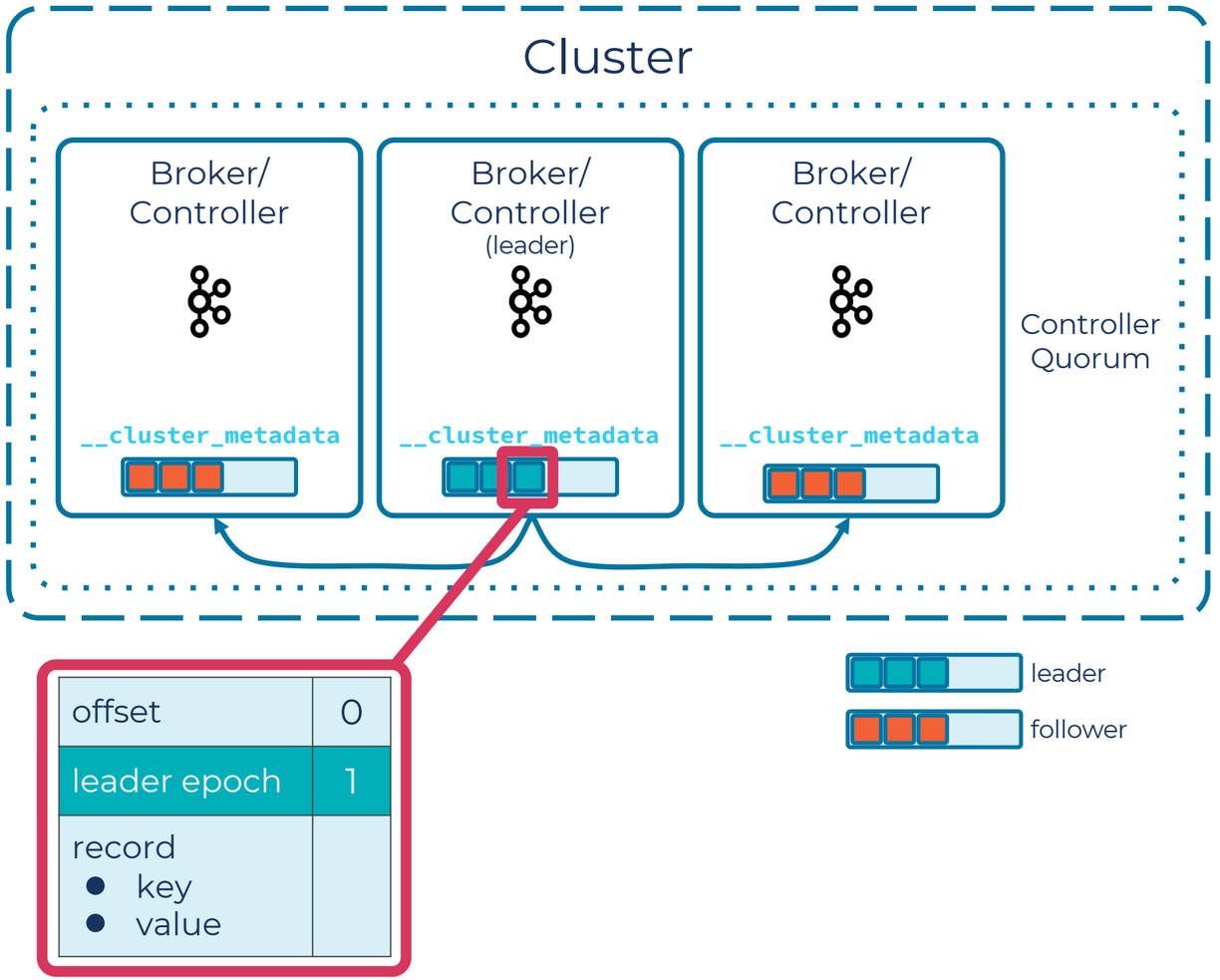
서버 시작 시 또는 Leader 가 중지되거나 Fail시 Controller Leader 선정 함



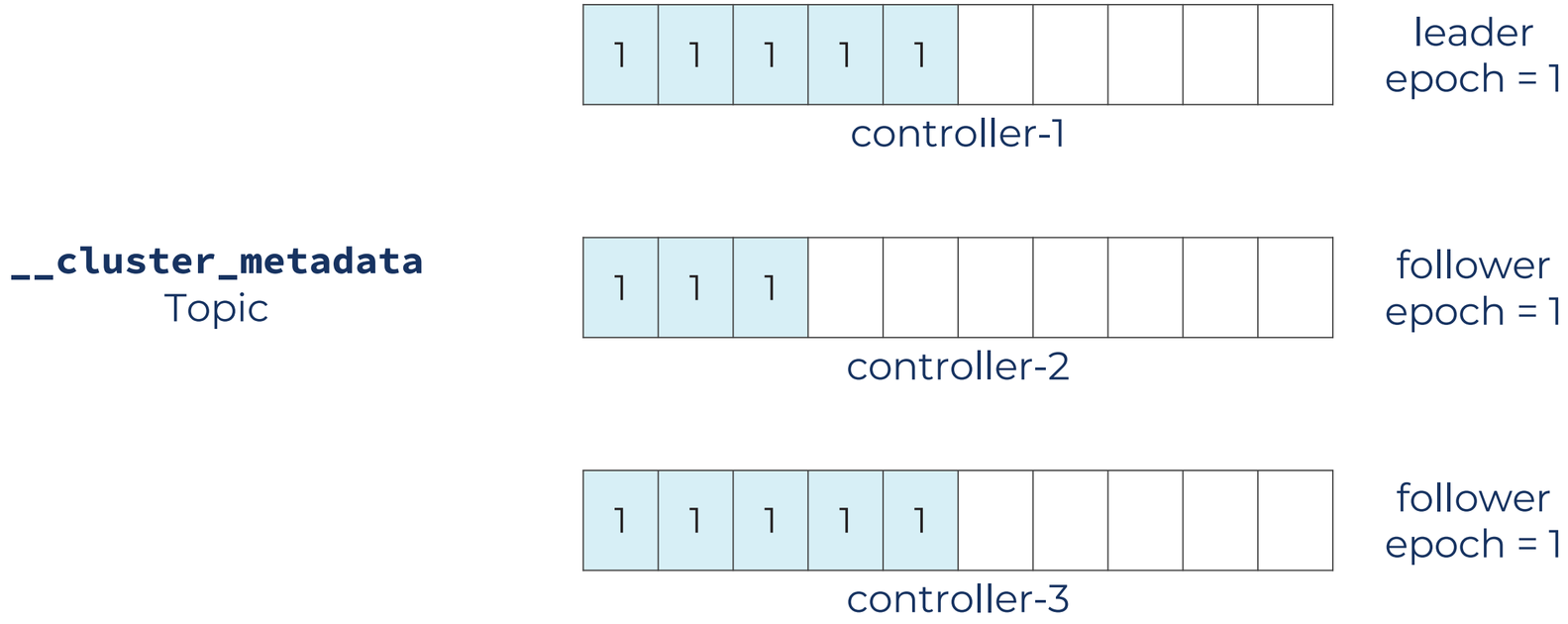
Controller Leader Election

Leader를 사용할 수 없는 경우 KRaft 프로토콜은 Leader가 없다고 인지하고 리더 선출을 시작

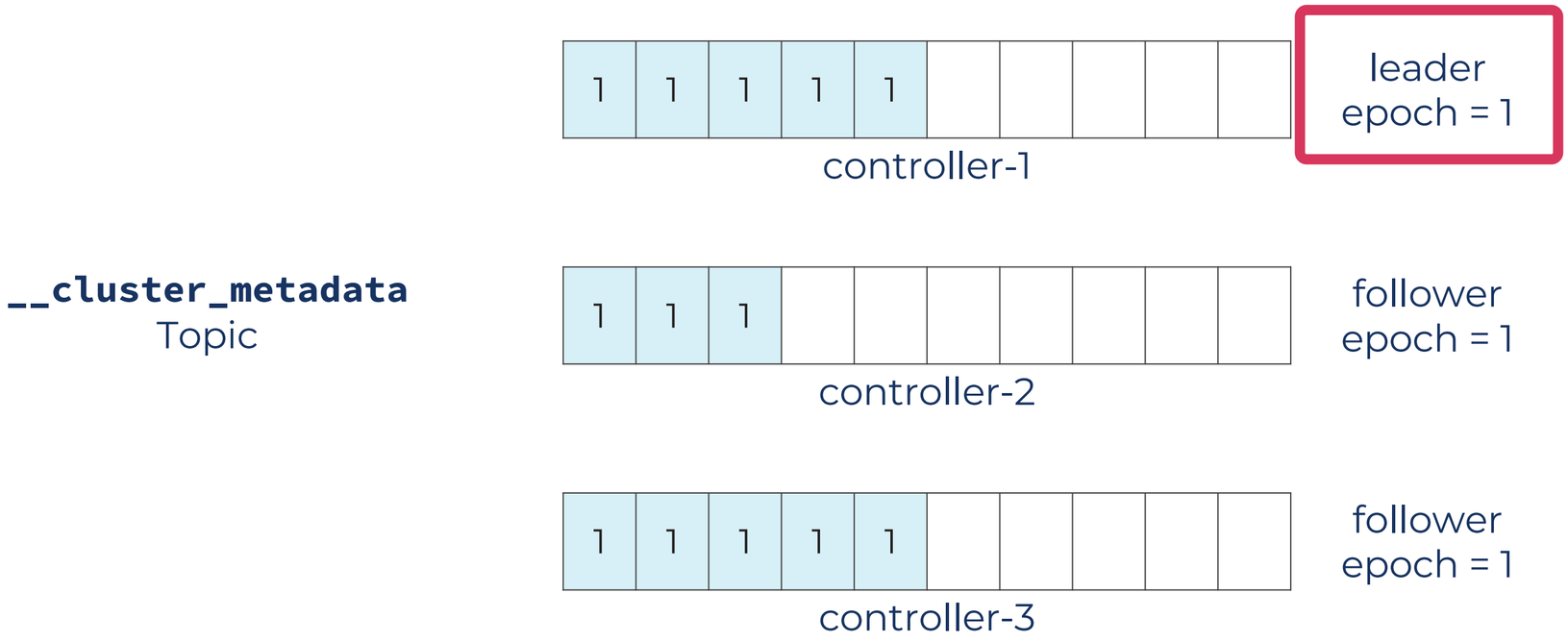
- KRaft increments the leader epoch whenever leader election starts
- Each event written to the **__cluster_metadata** topic including the current leader epoch
- Other controller nodes subscribe to these changes.



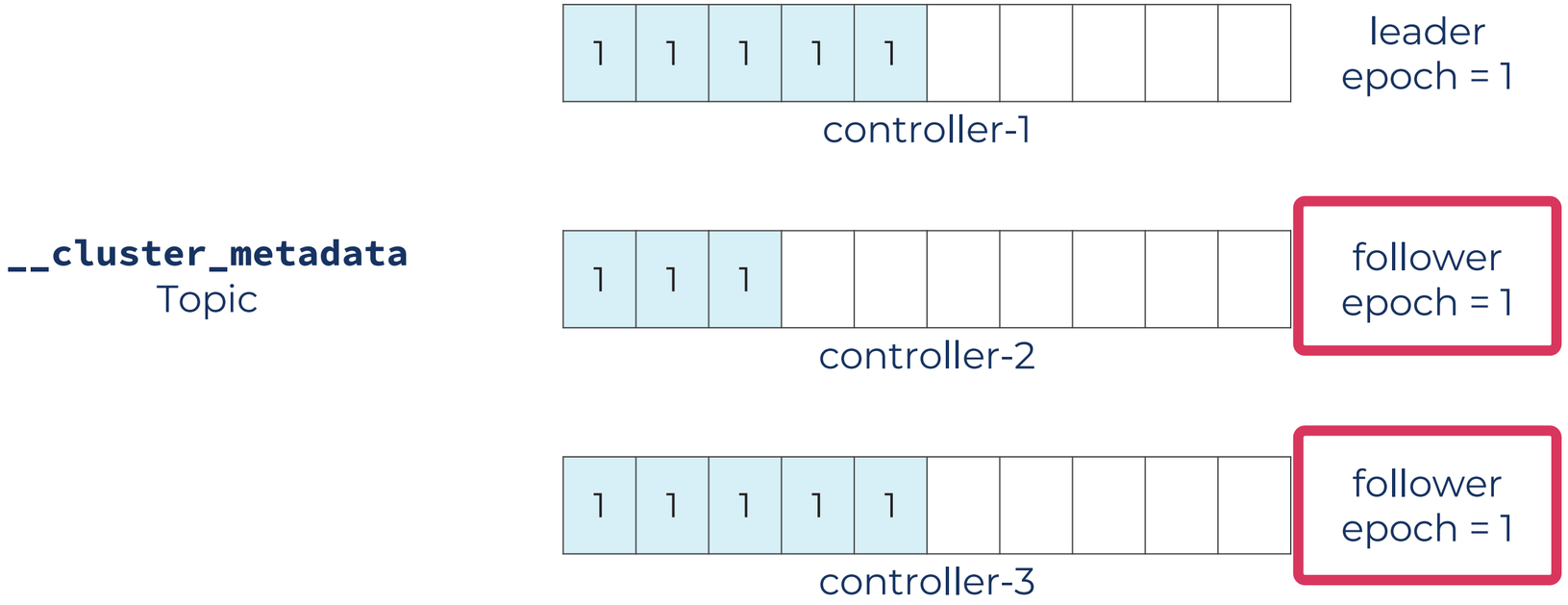
Controller Leader Election



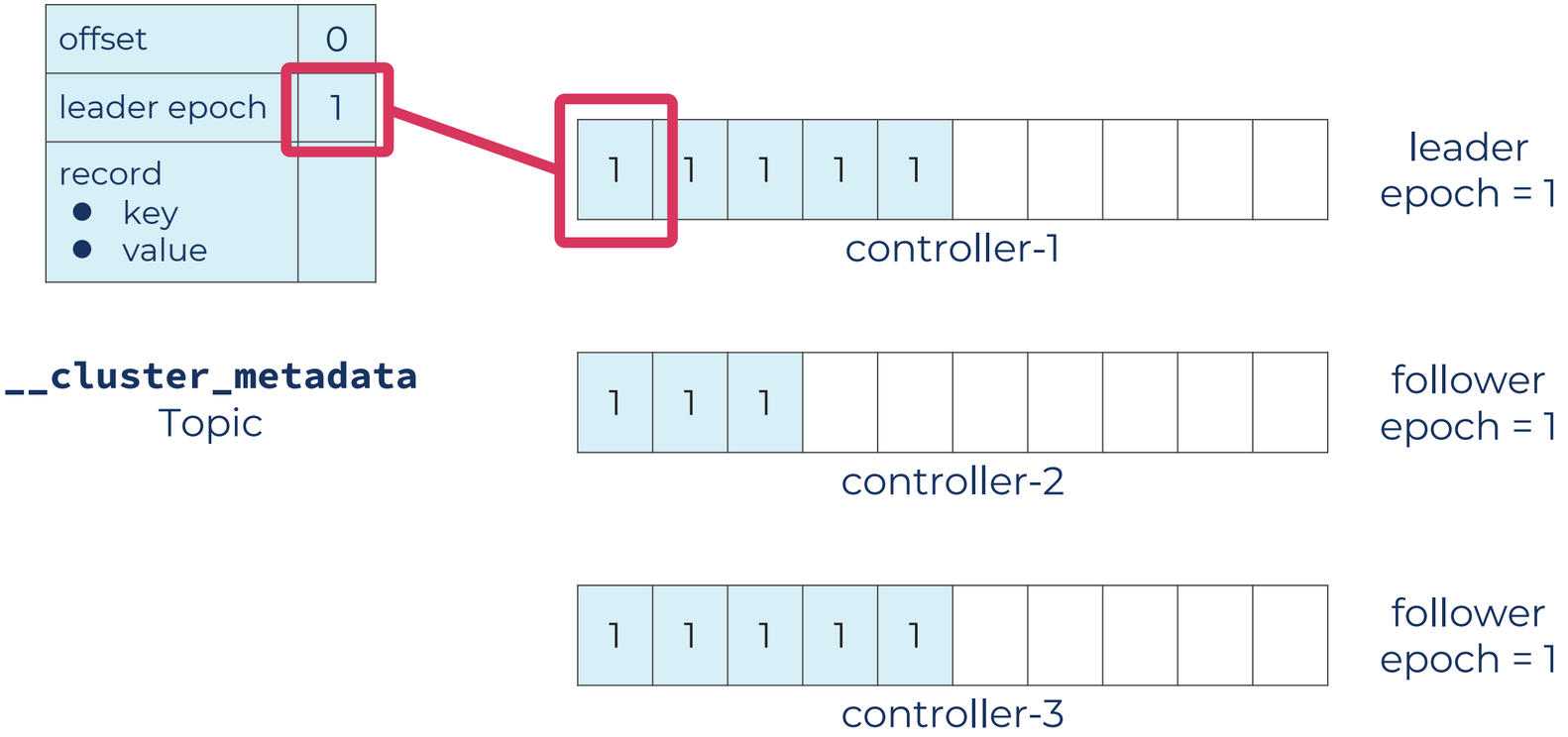
Controller Leader Election



Controller Leader Election



Controller Leader Election

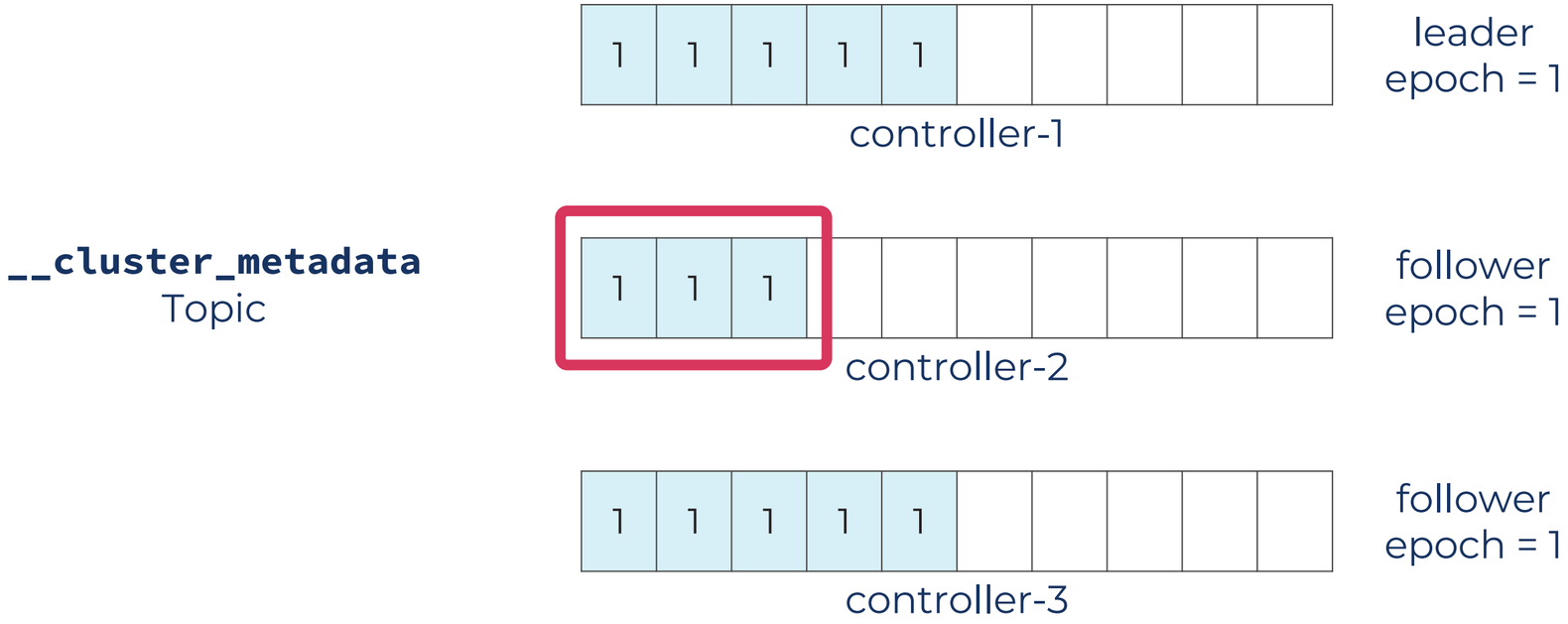


Controller Leader Election

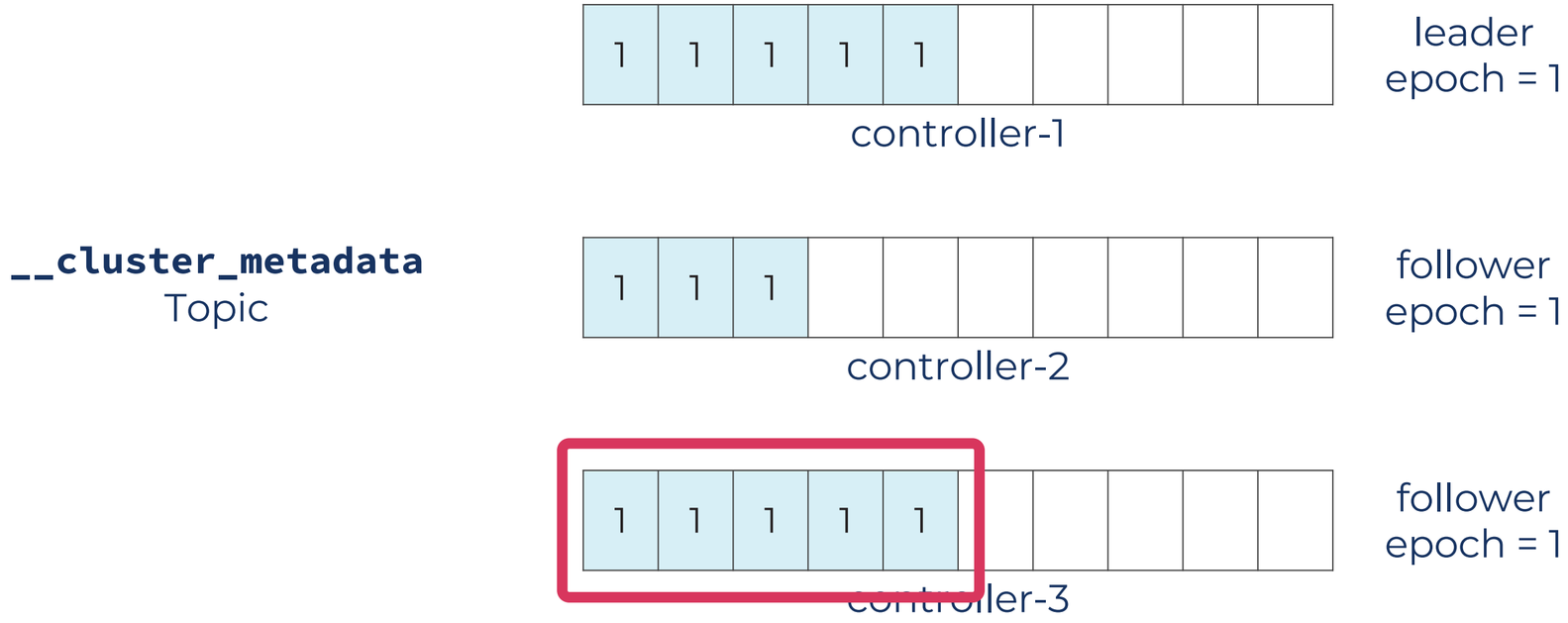
offset	0
leader epoch	1
record • key • value	

`__cluster_metadata`
Topic

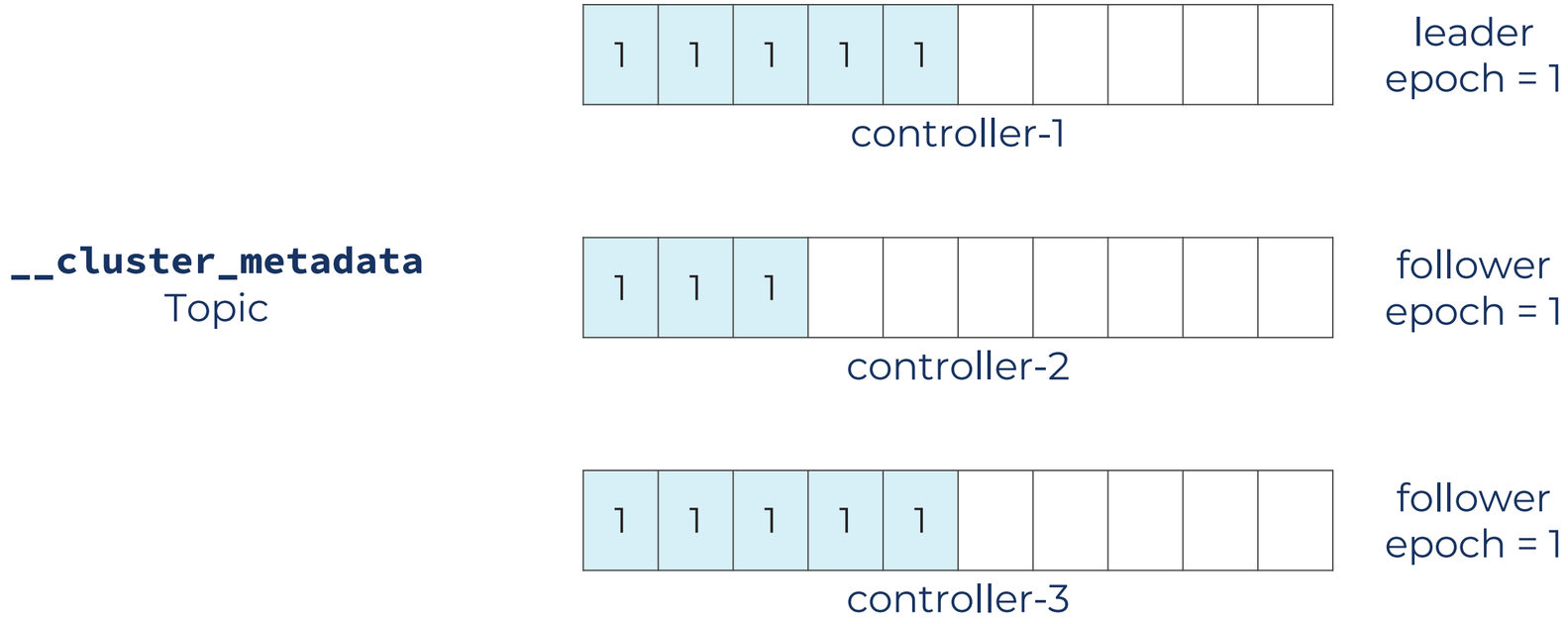
Controller Leader Election



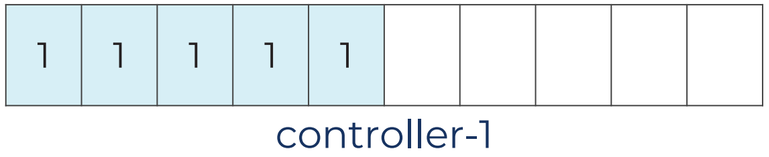
Controller Leader Election



Controller Leader Election



Controller Leader Election



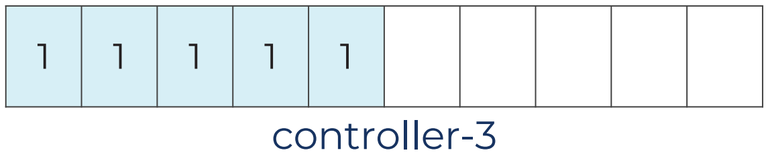
leader
epoch = 1

controller-1

__cluster_metadata
Topic

follower
epoch = 1

controller-2



follower
epoch = 1

controller-3

Election Rules:

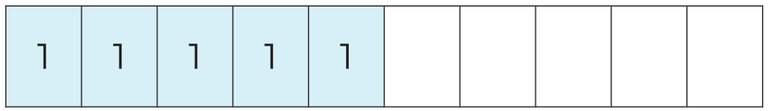
1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

Controller Leader Election

controller-1

`__cluster_metadata`
Topic

controller-2

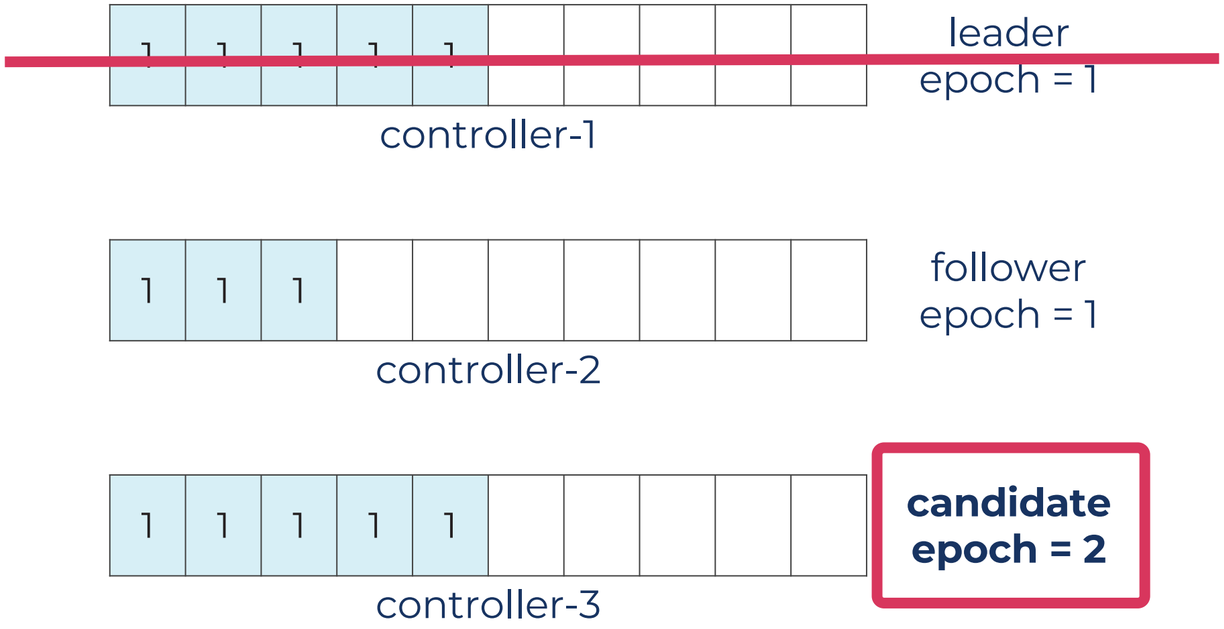


controller-3

Election Rules:

1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

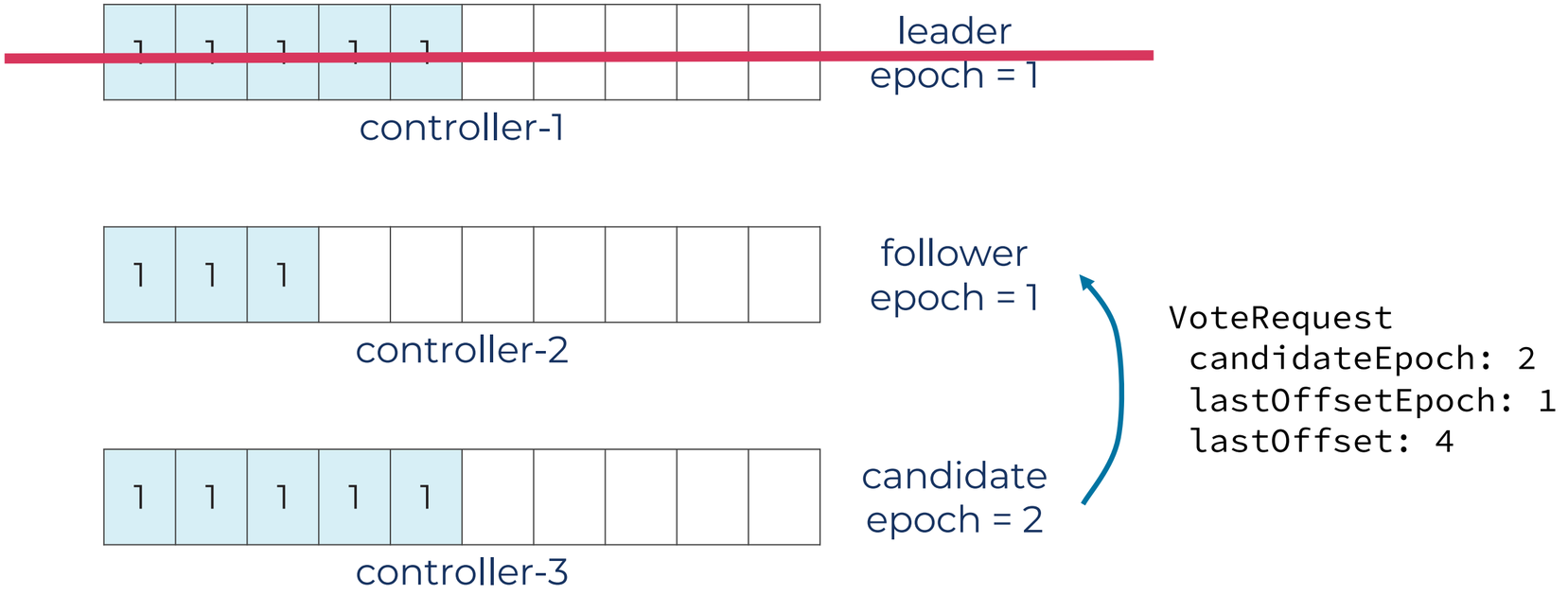
Controller Leader Election



Election Rules:

1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

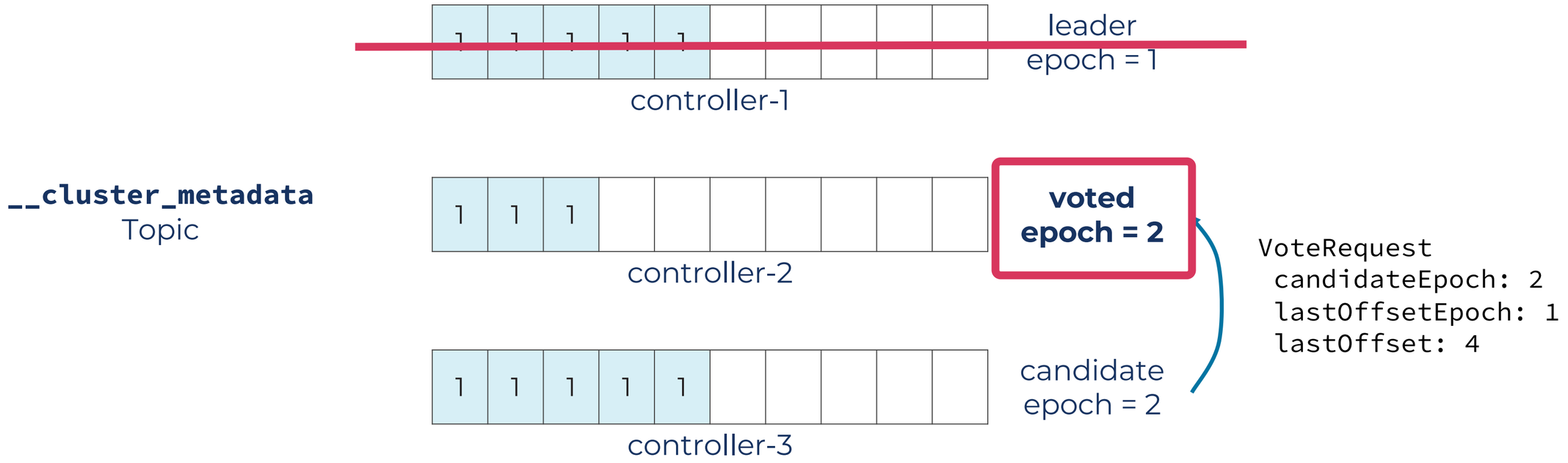
Controller Leader Election



Election Rules:

1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

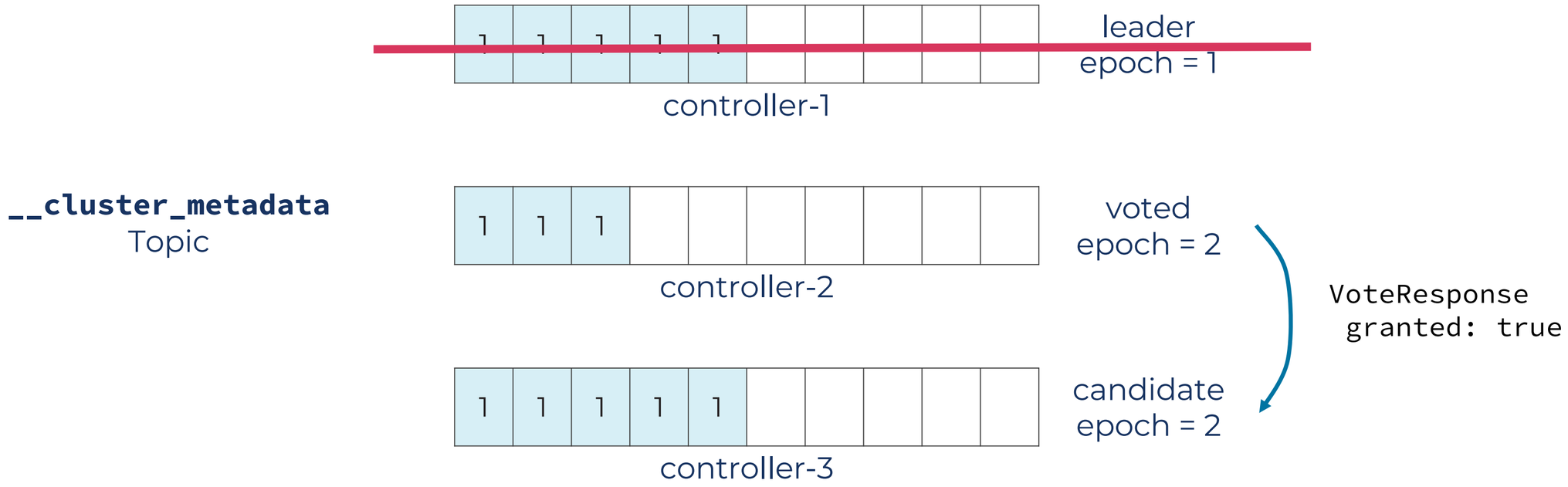
Controller Leader Election



Election Rules:

1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

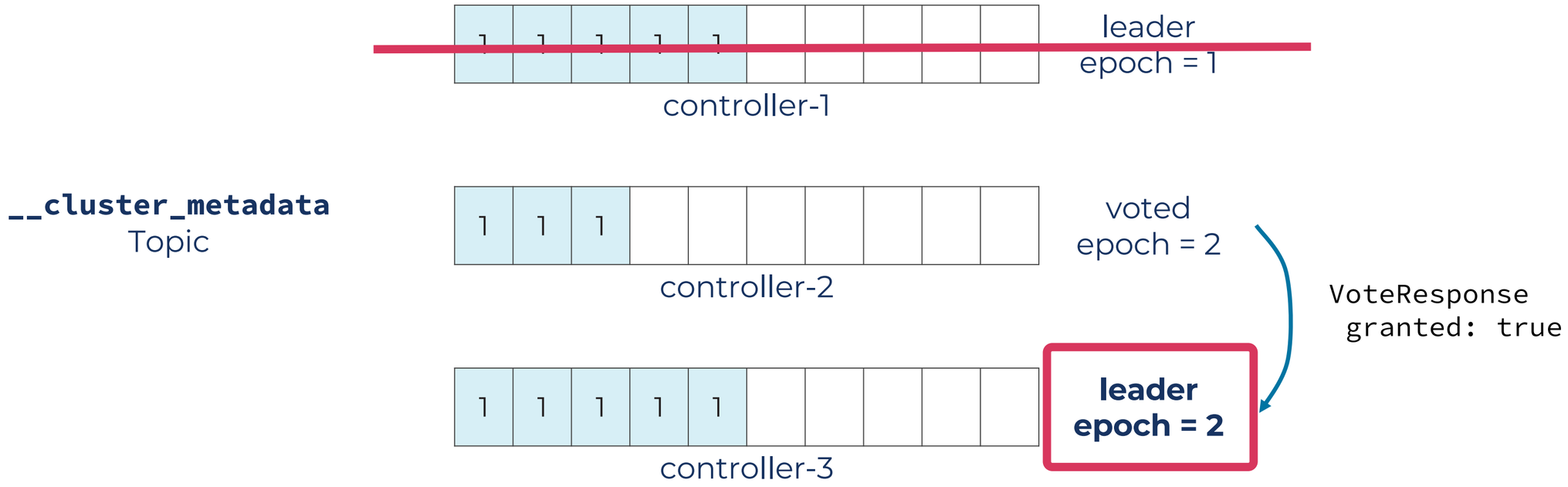
Controller Leader Election



Election Rules:

1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

Controller Leader Election



Election Rules:

1. Epoch당 투표는 한 번만
2. 더 긴 로그에 투표
3. 단순한 다수결 승리

Controller Leader Election

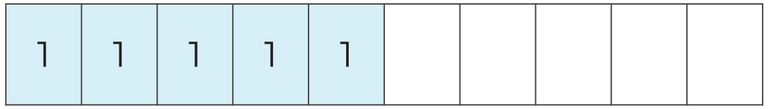
leader
epoch = 1

controller-1

__cluster_metadata
Topic

voted
epoch = 2

controller-2



leader
epoch = 2

controller-3

Controller Leader Election

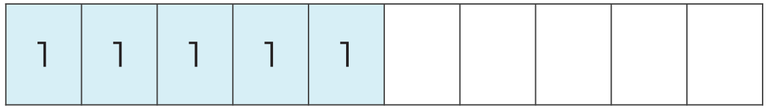
leader
epoch = 1

controller-1

__cluster_metadata
Topic

voted
epoch = 2

controller-2

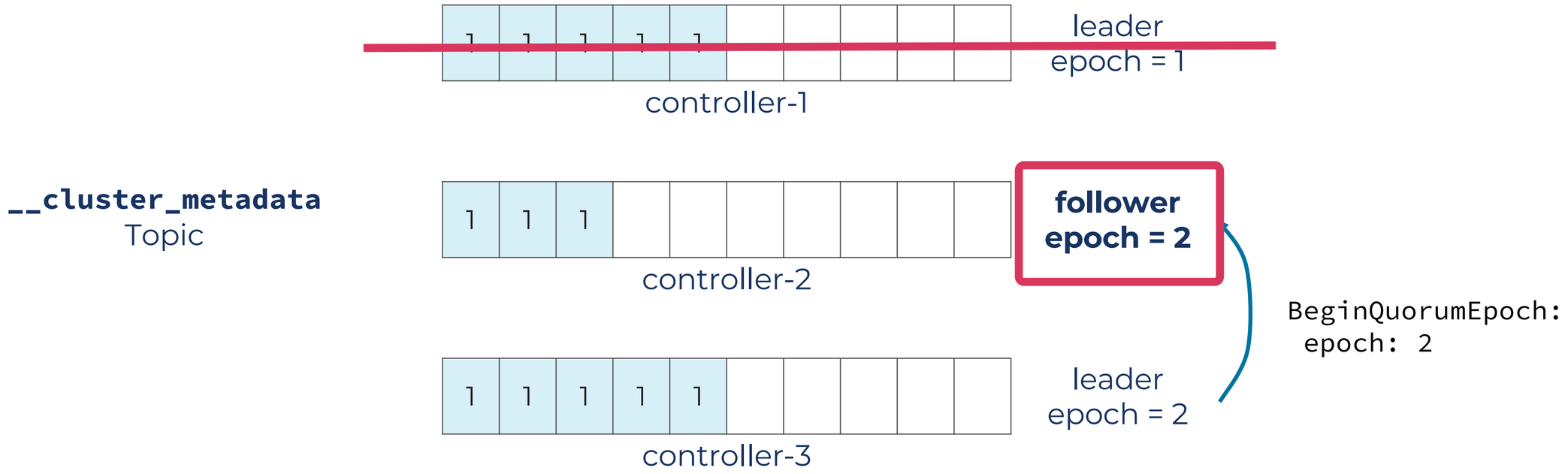


leader
epoch = 2

controller-3

```
BeginQuorumEpoch:
 epoch: 2
```

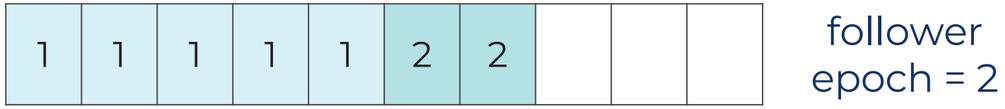
Controller Leader Election



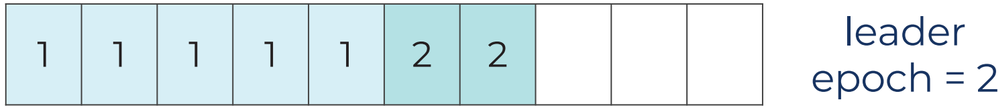
Controller Leader Election

controller-1

__cluster_metadata
Topic



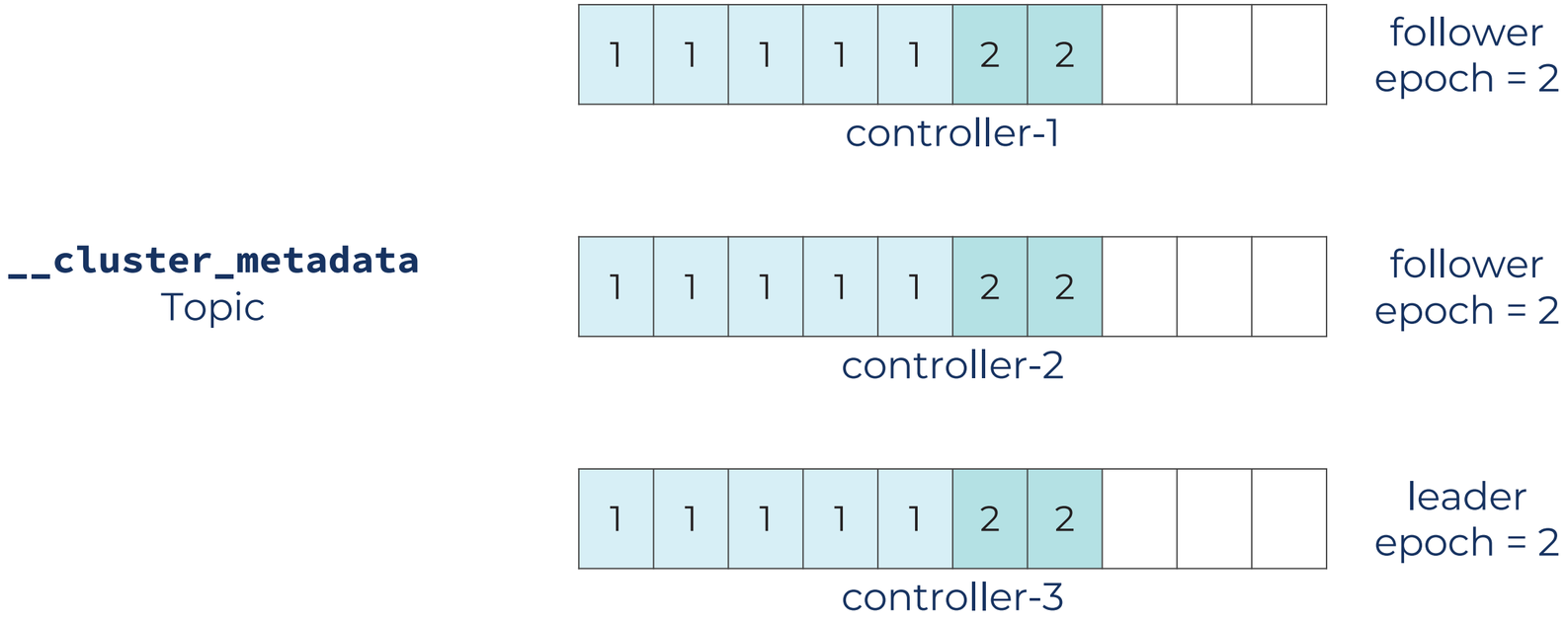
controller-2



controller-3

2 new messages are accepted by the new leader as epoch 2 and replicated

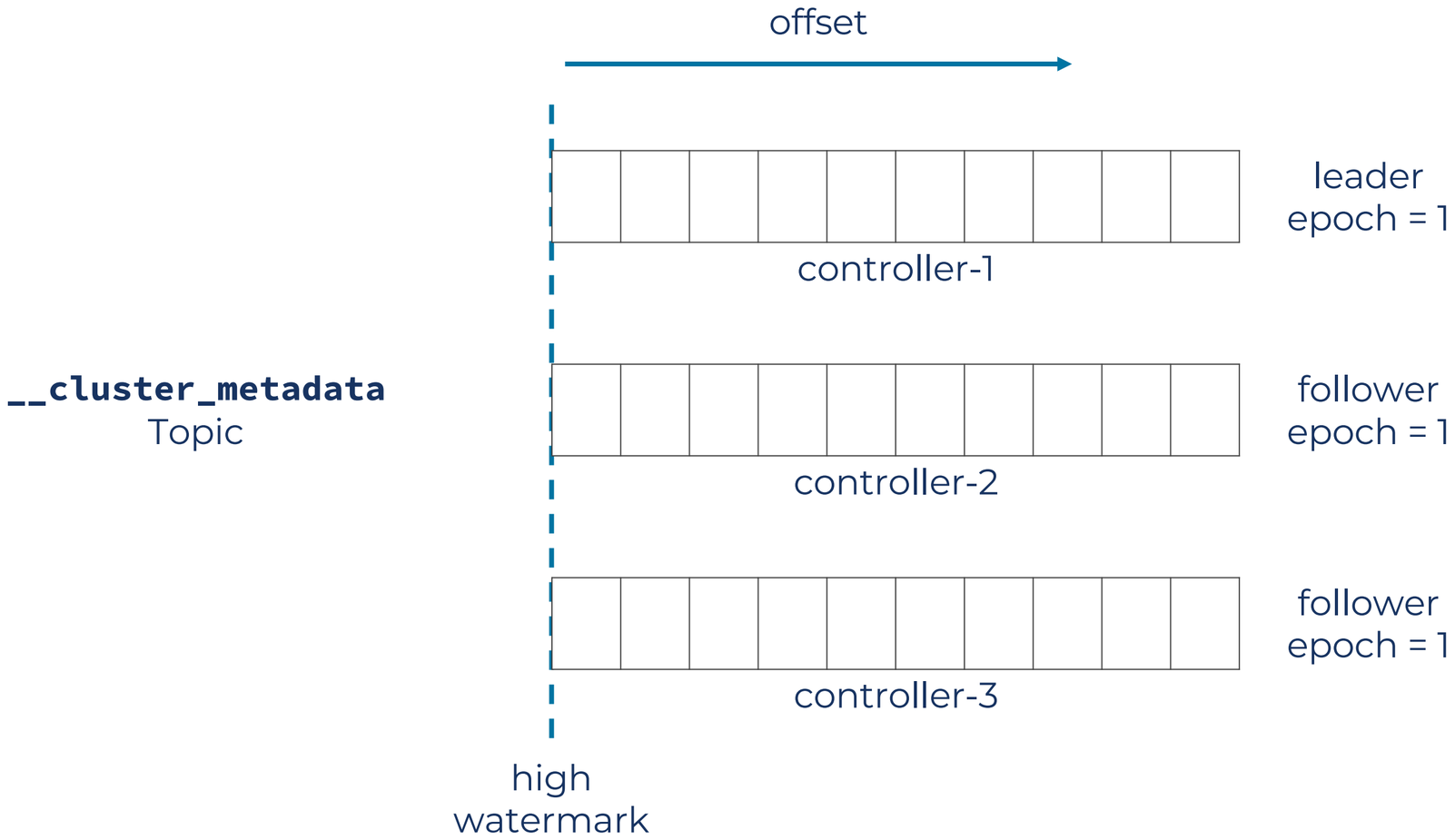
Controller Leader Election



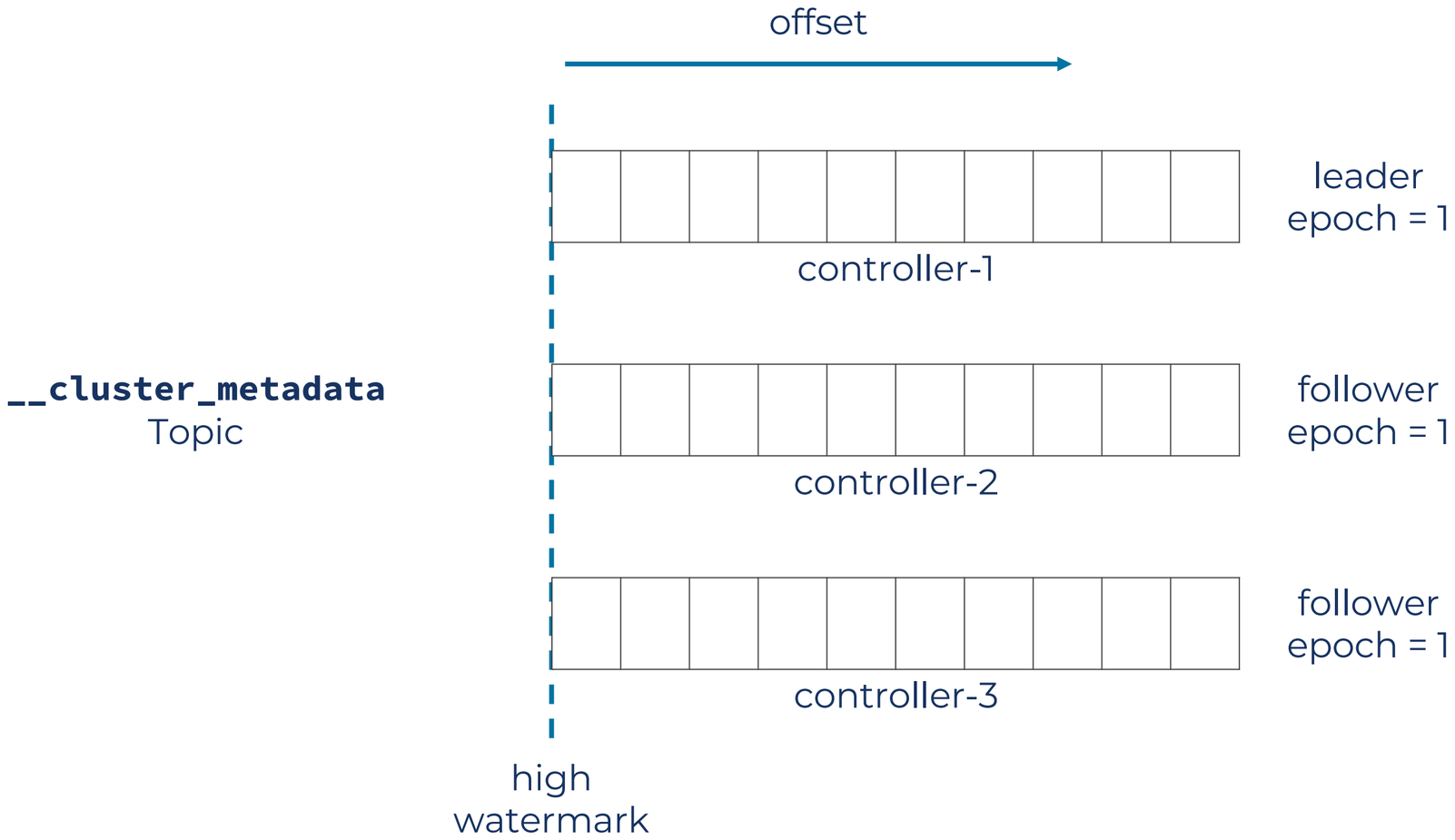
CONFLUENT

Control Plane: Replication Protocol (KRaft)

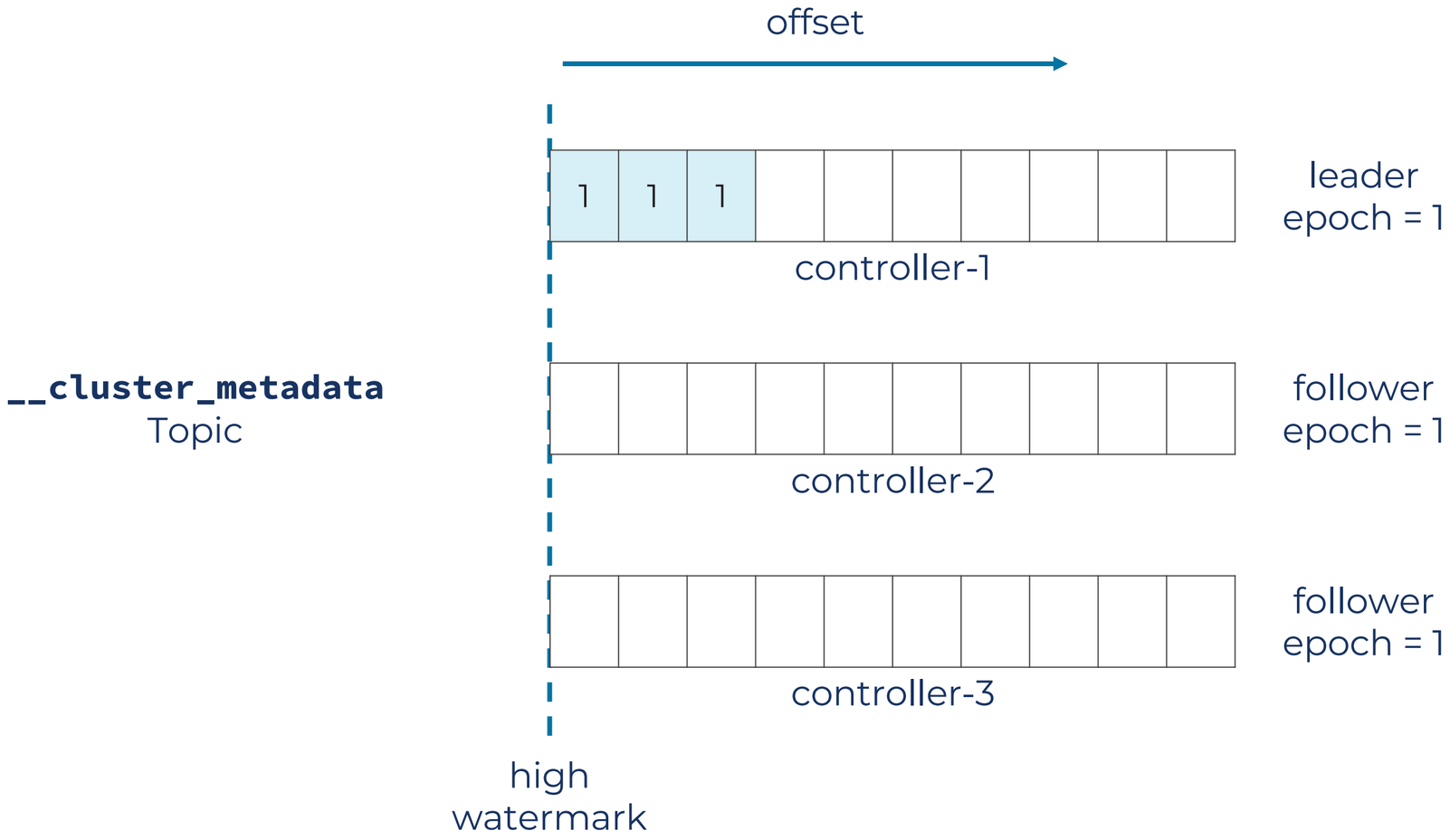
Metadata Topic Replication



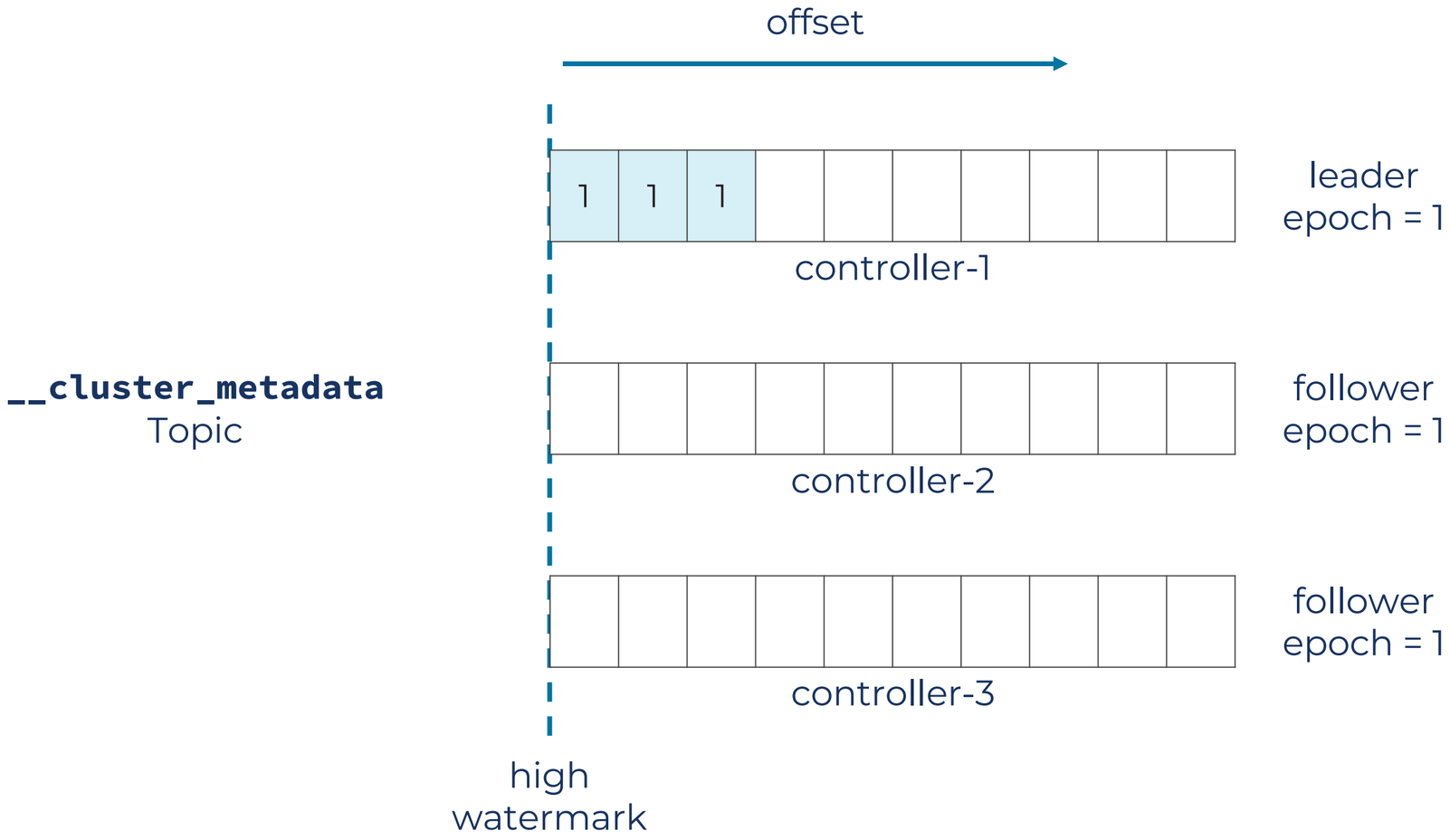
Metadata Topic Replication



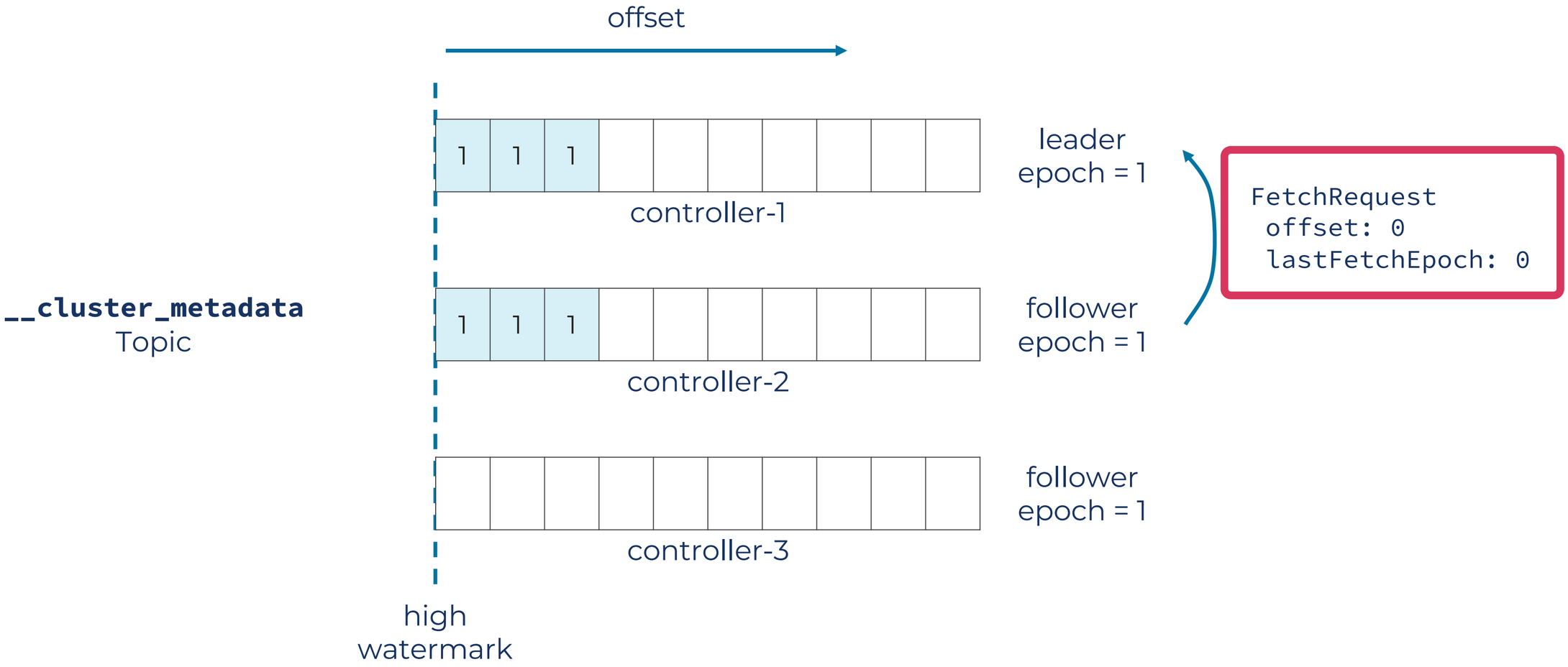
Metadata Replication



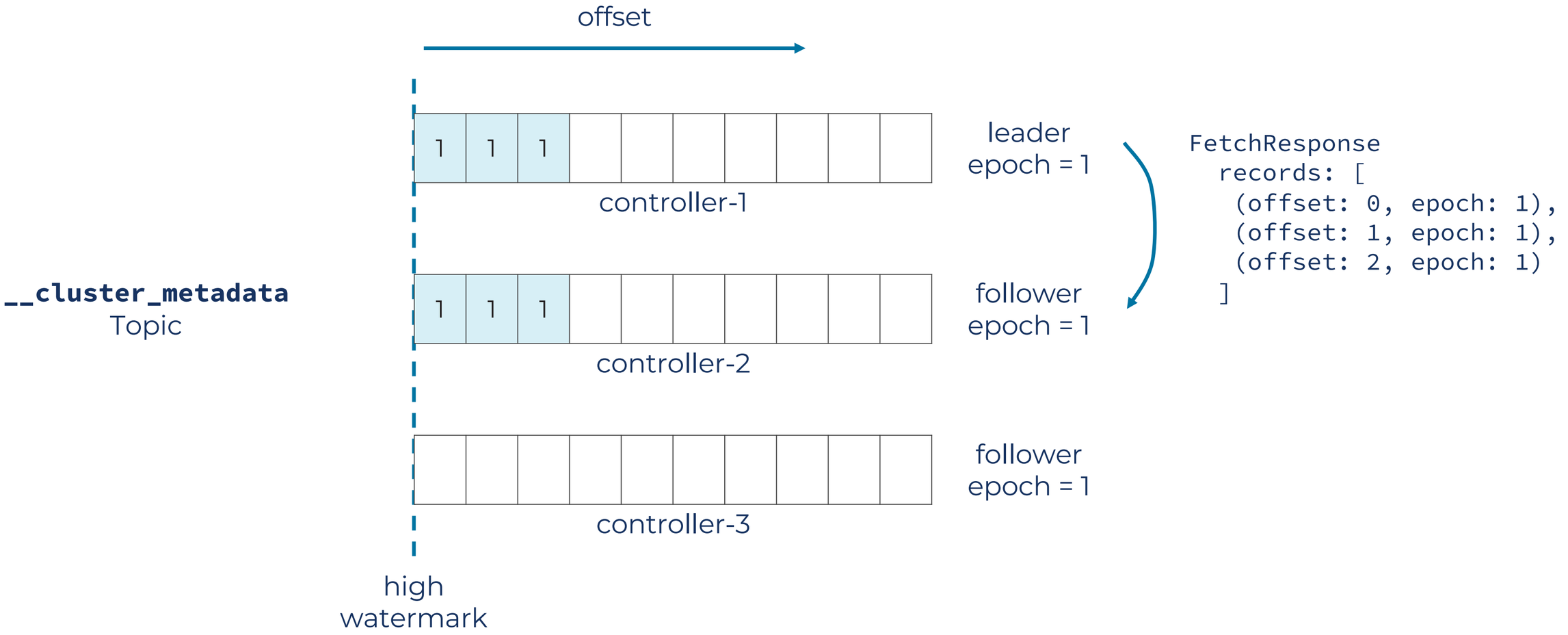
Metadata Replication



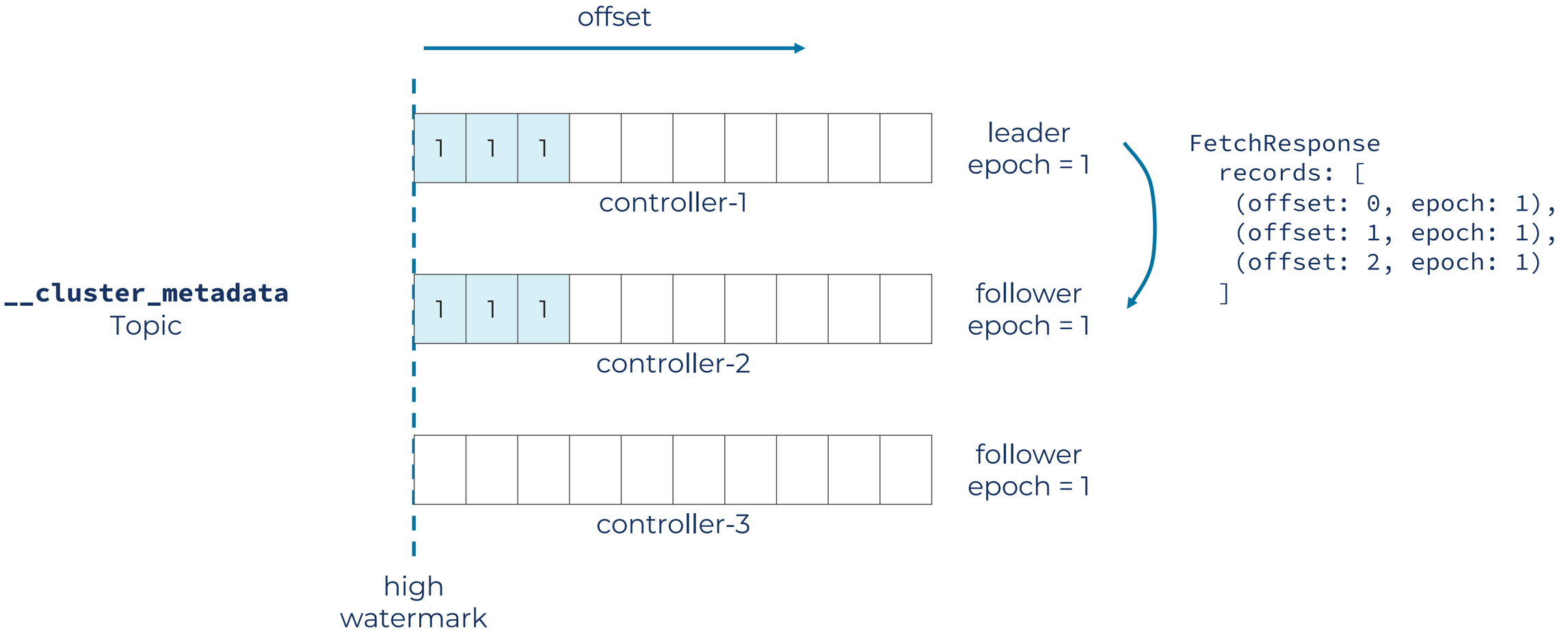
Metadata Replication



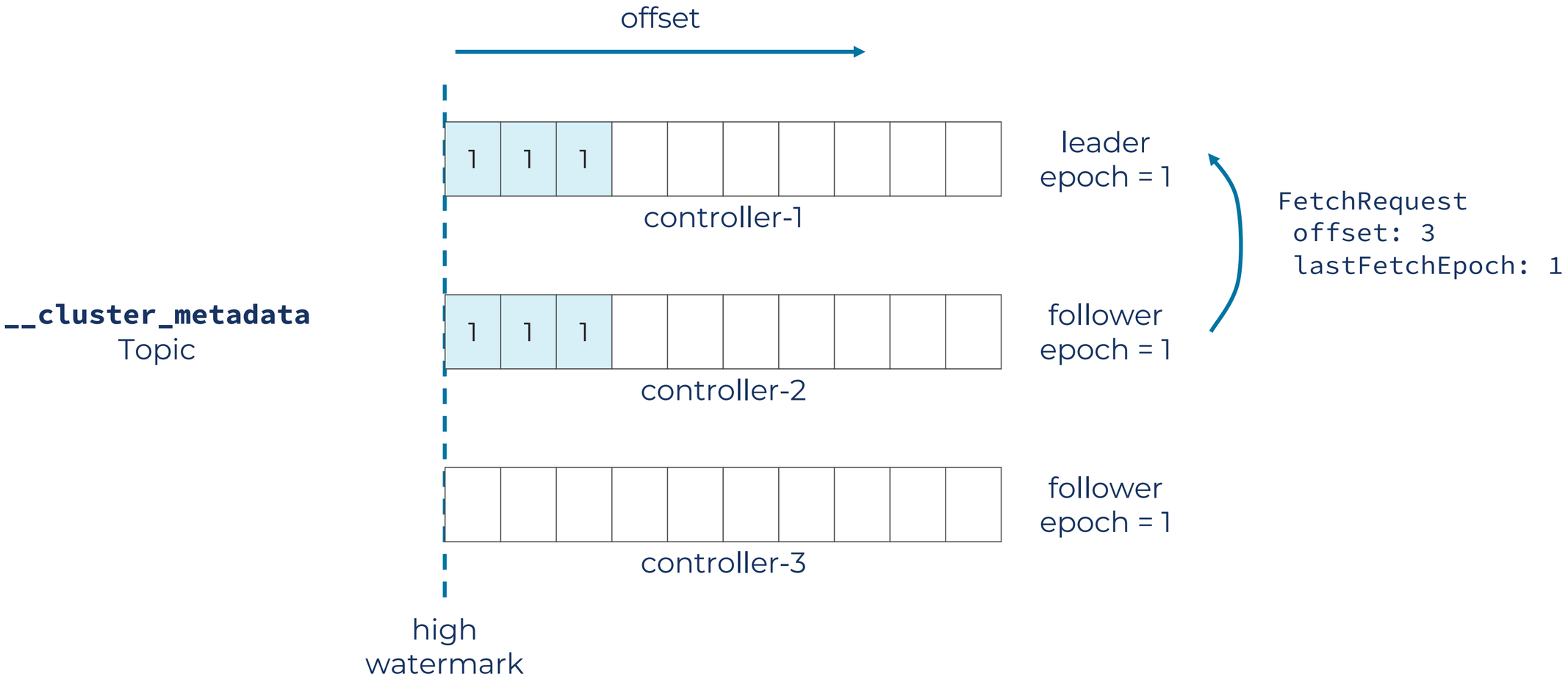
Metadata Replication



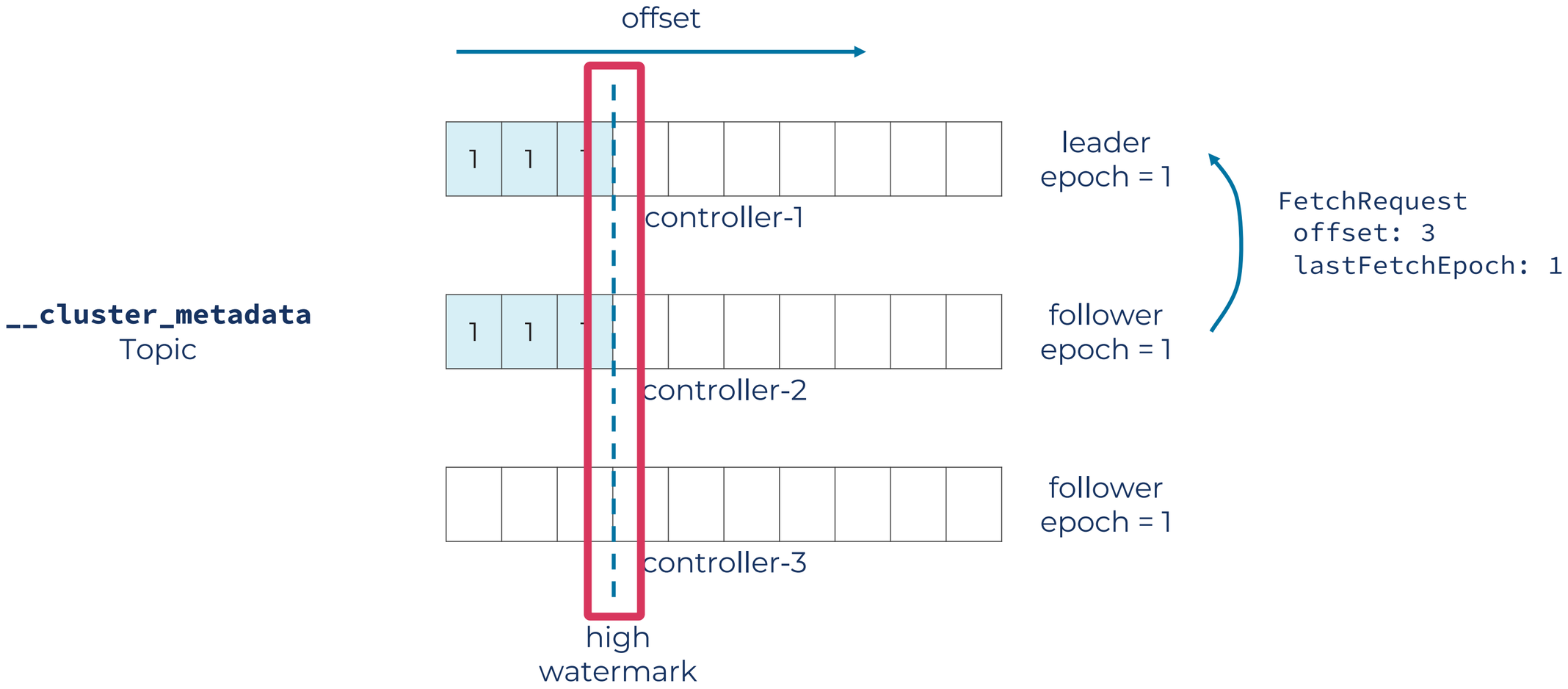
Metadata Replication



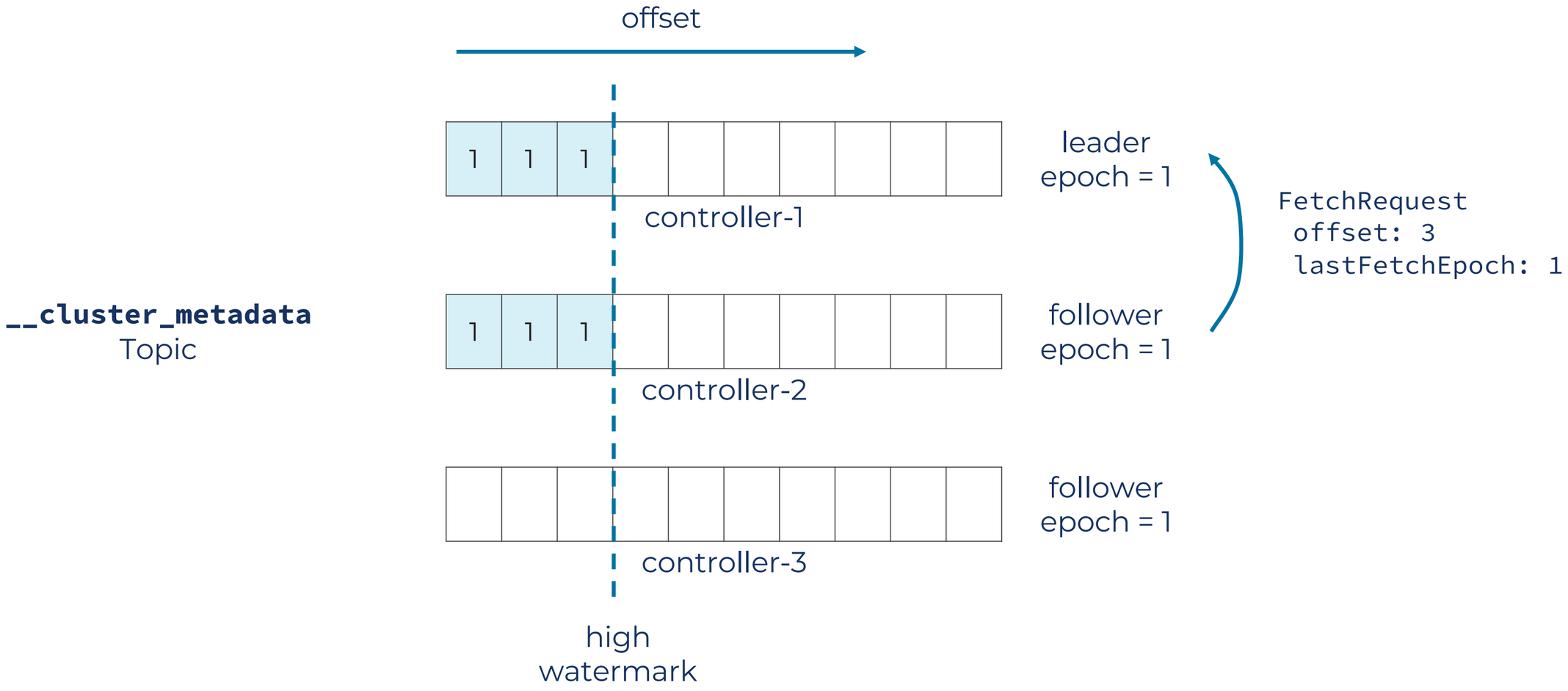
Metadata Replication



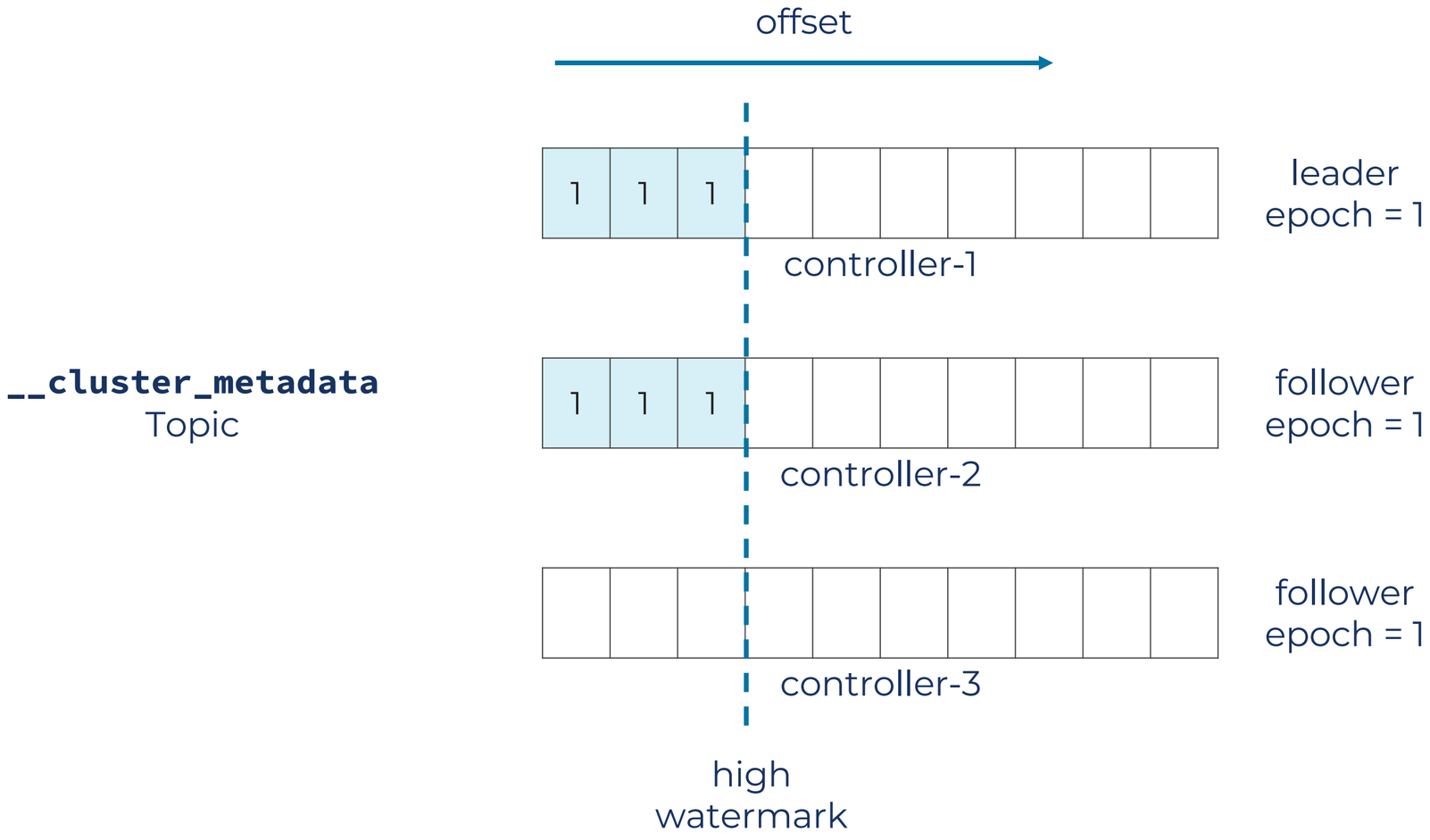
Metadata Replication



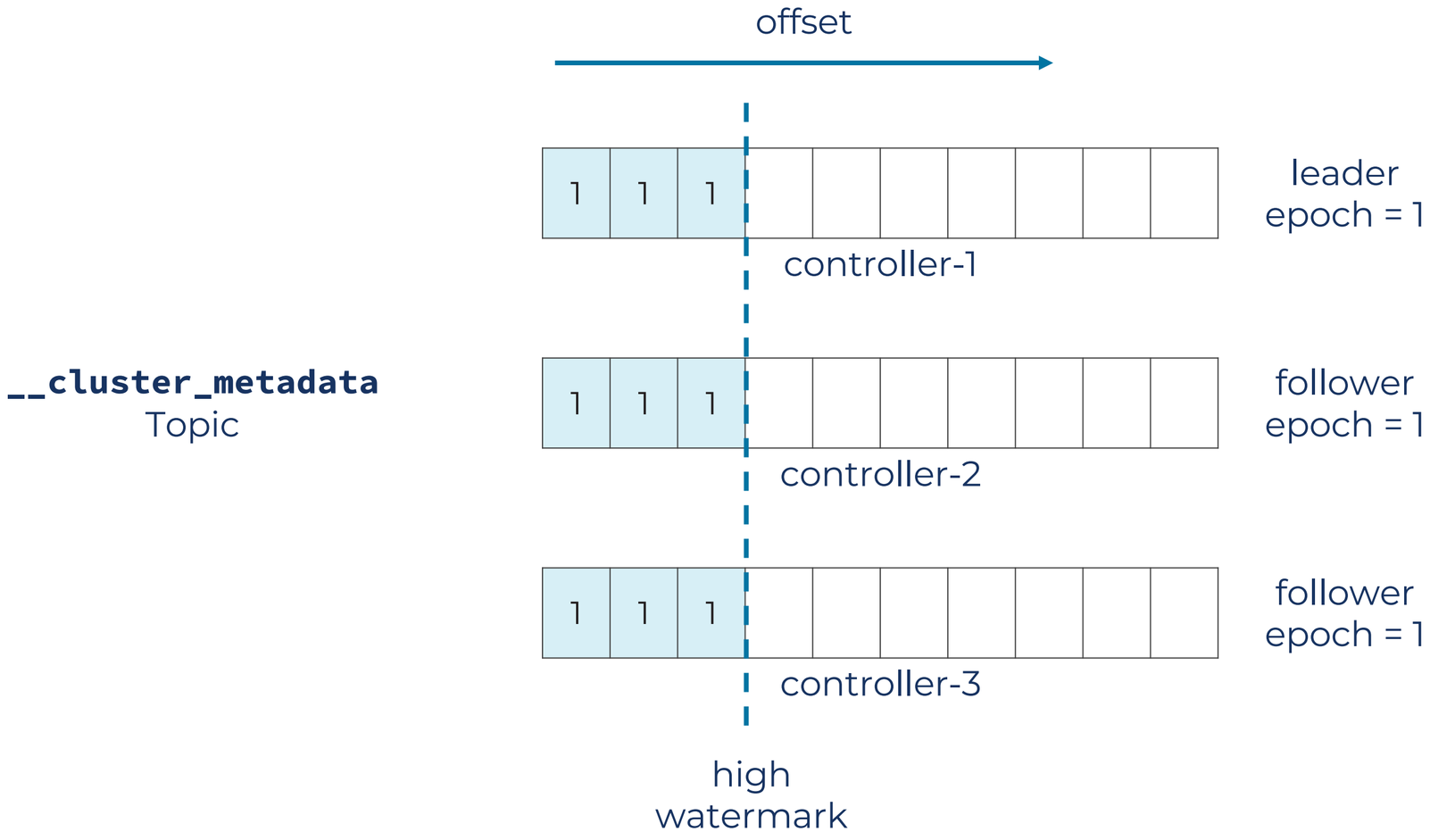
Metadata Replication



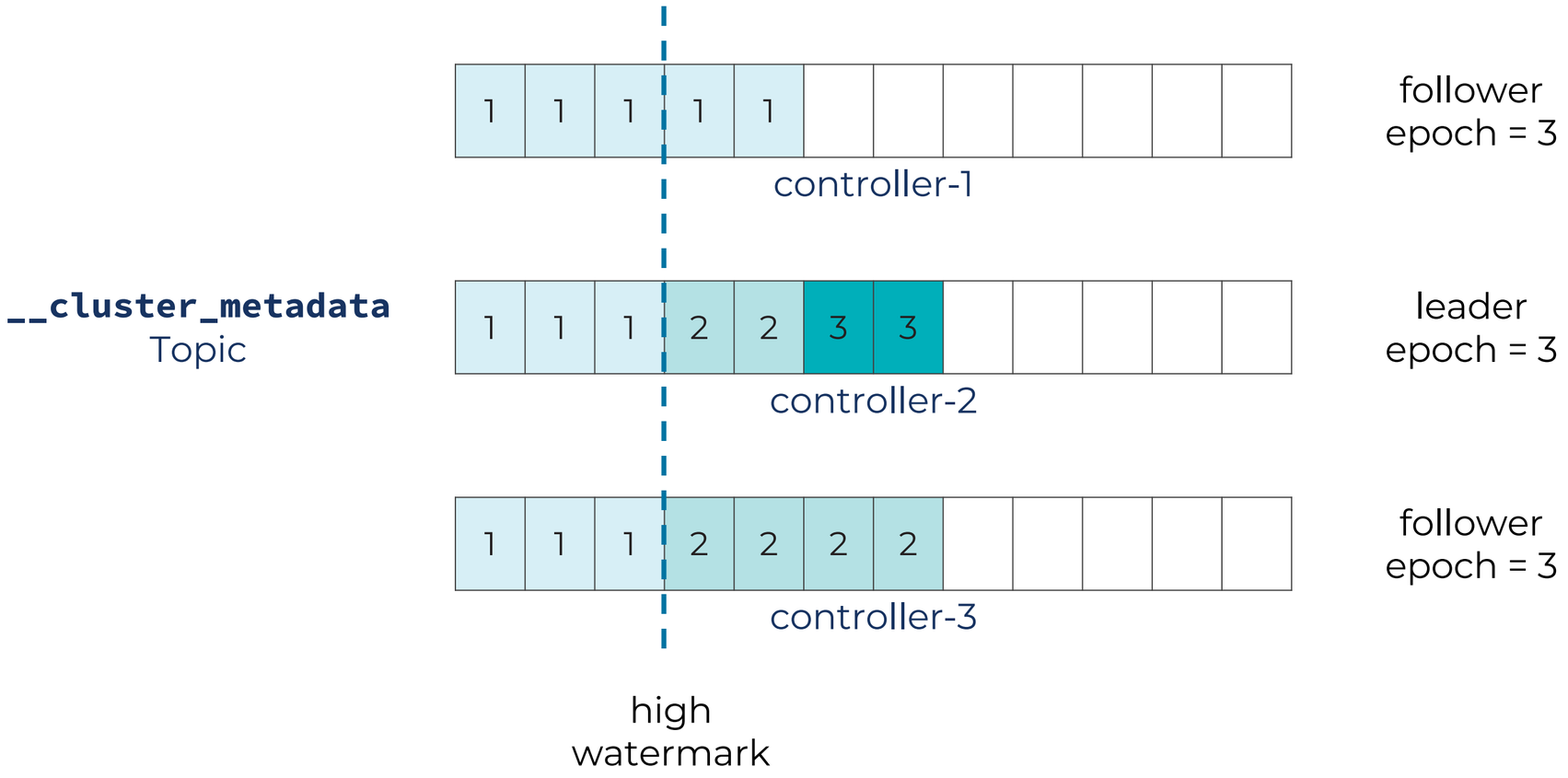
Metadata Replication



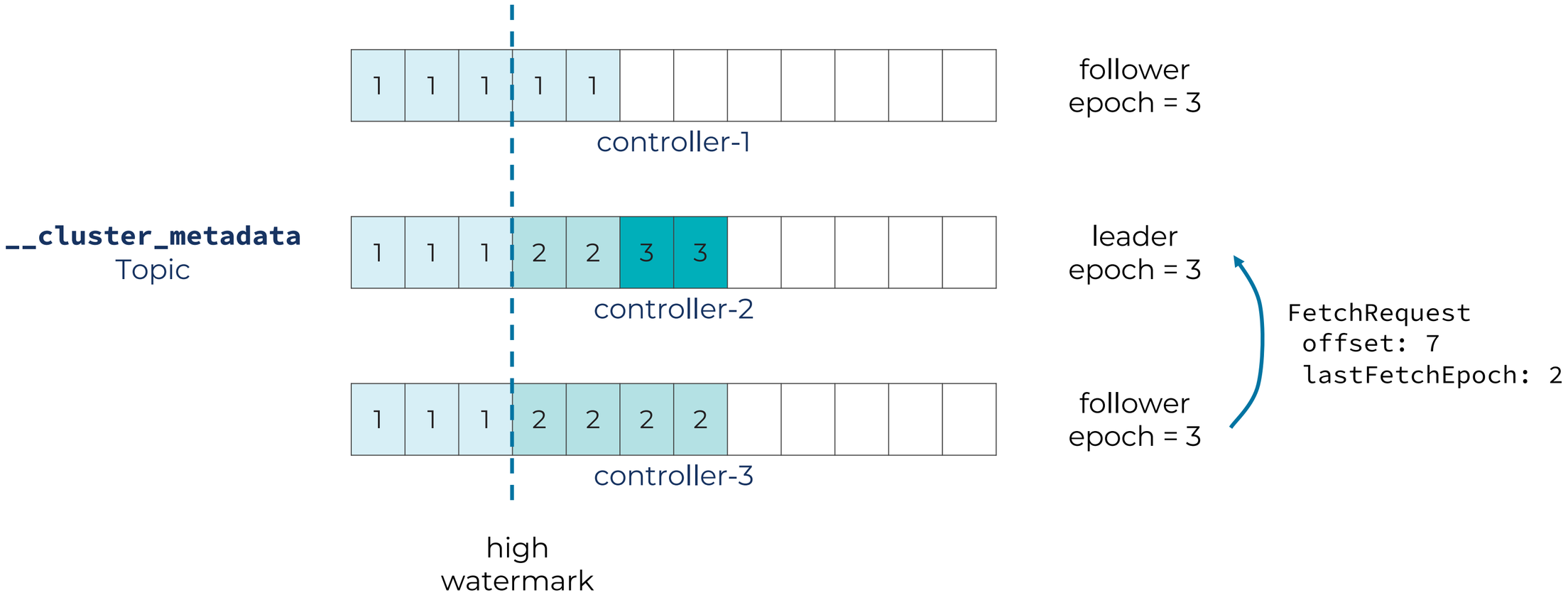
Metadata Replication



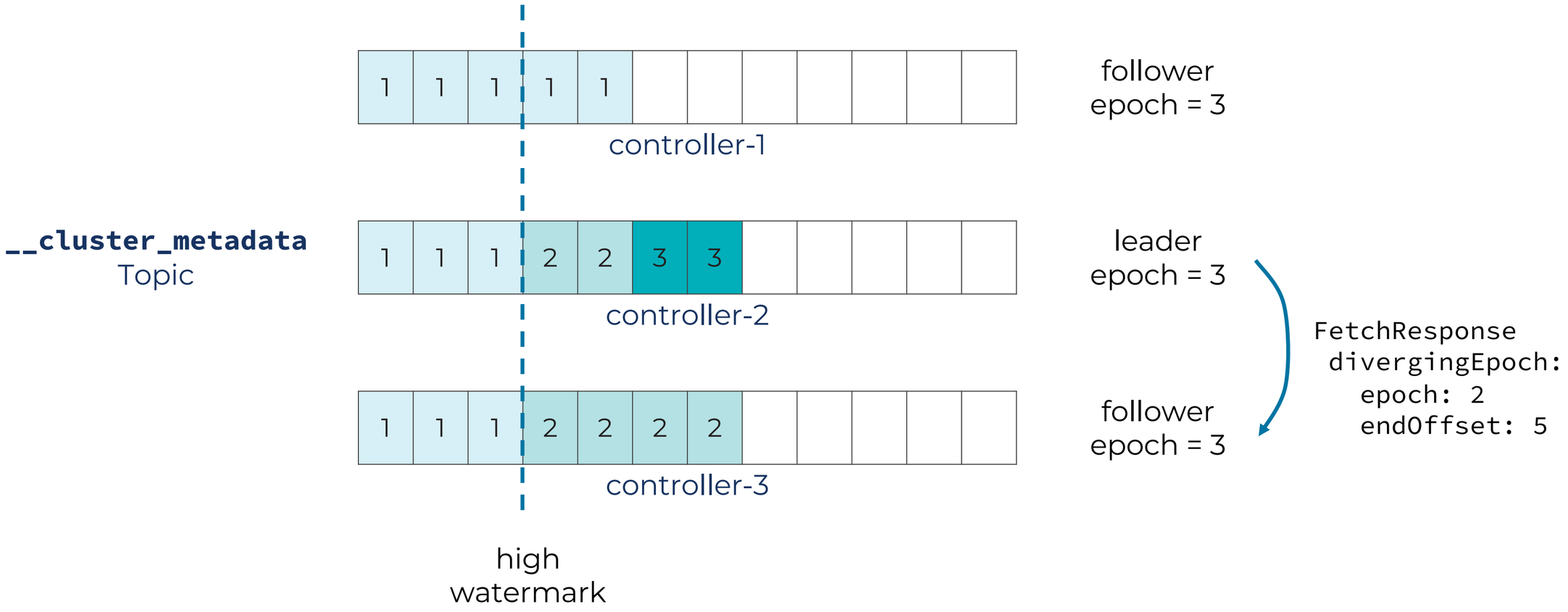
Metadata Replica Reconciliation



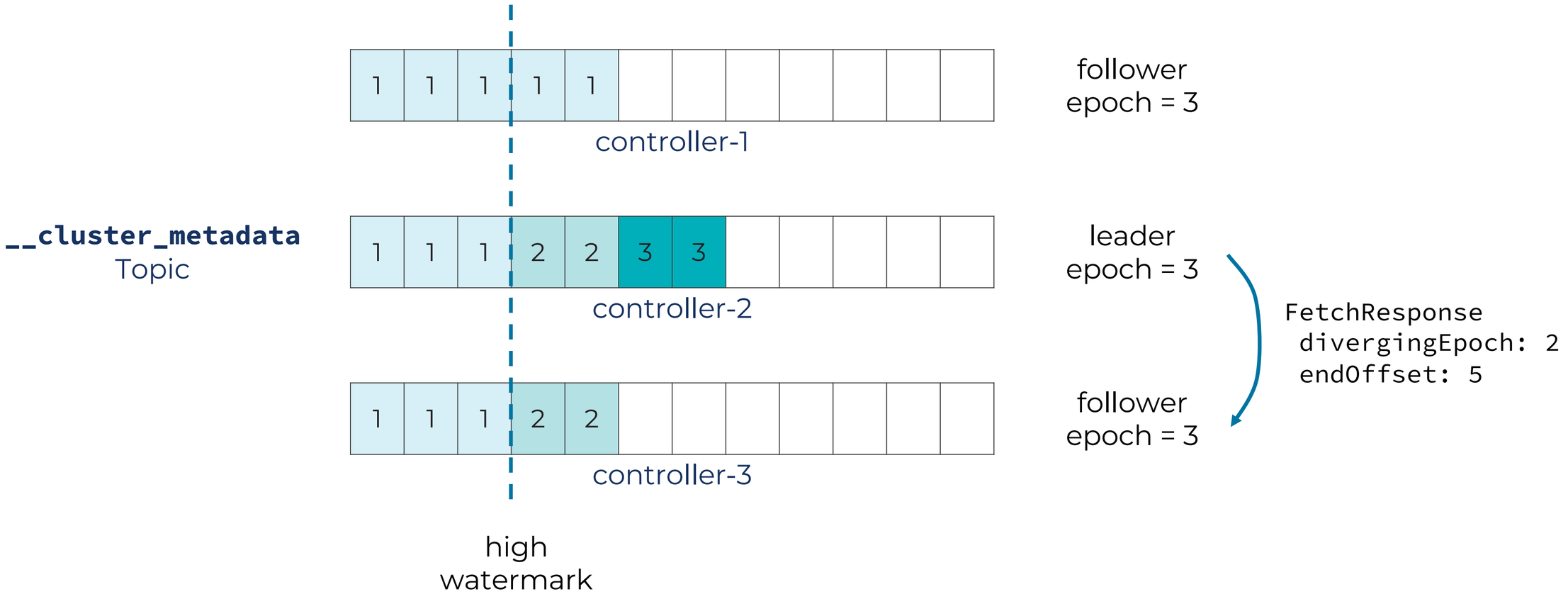
Metadata Replica Reconciliation



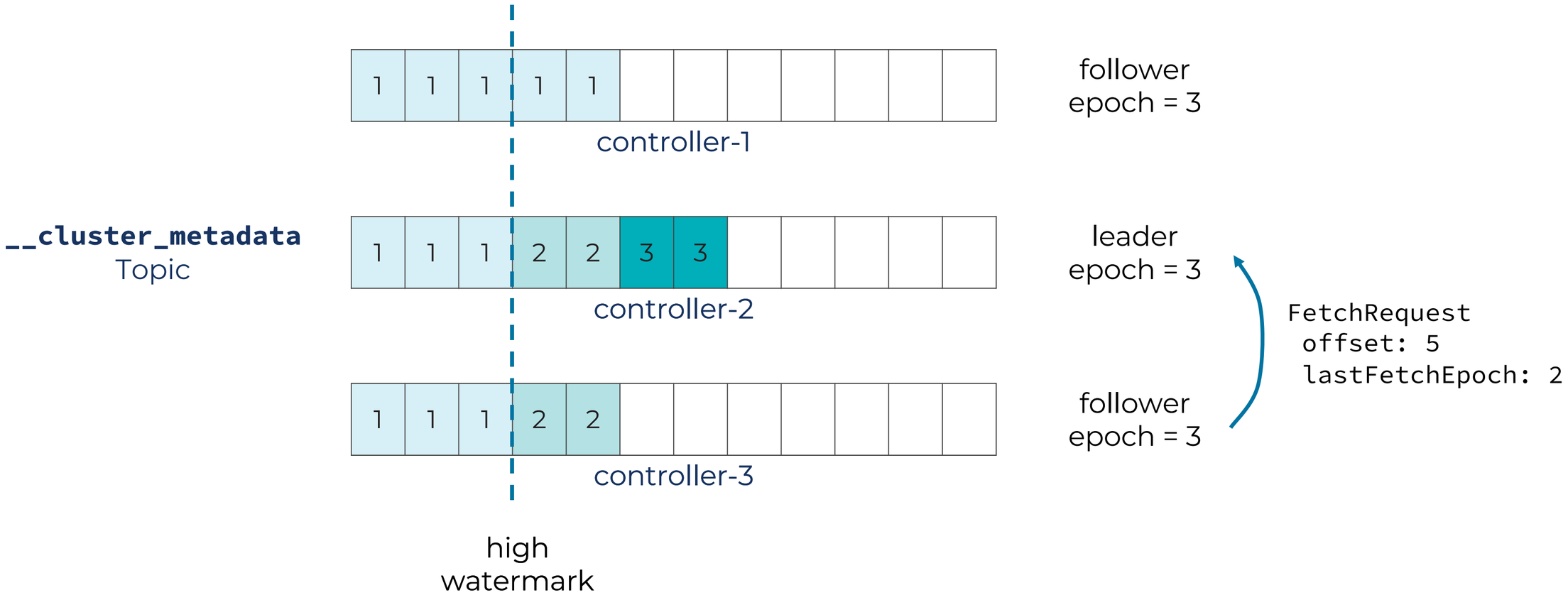
Metadata Replica Reconciliation



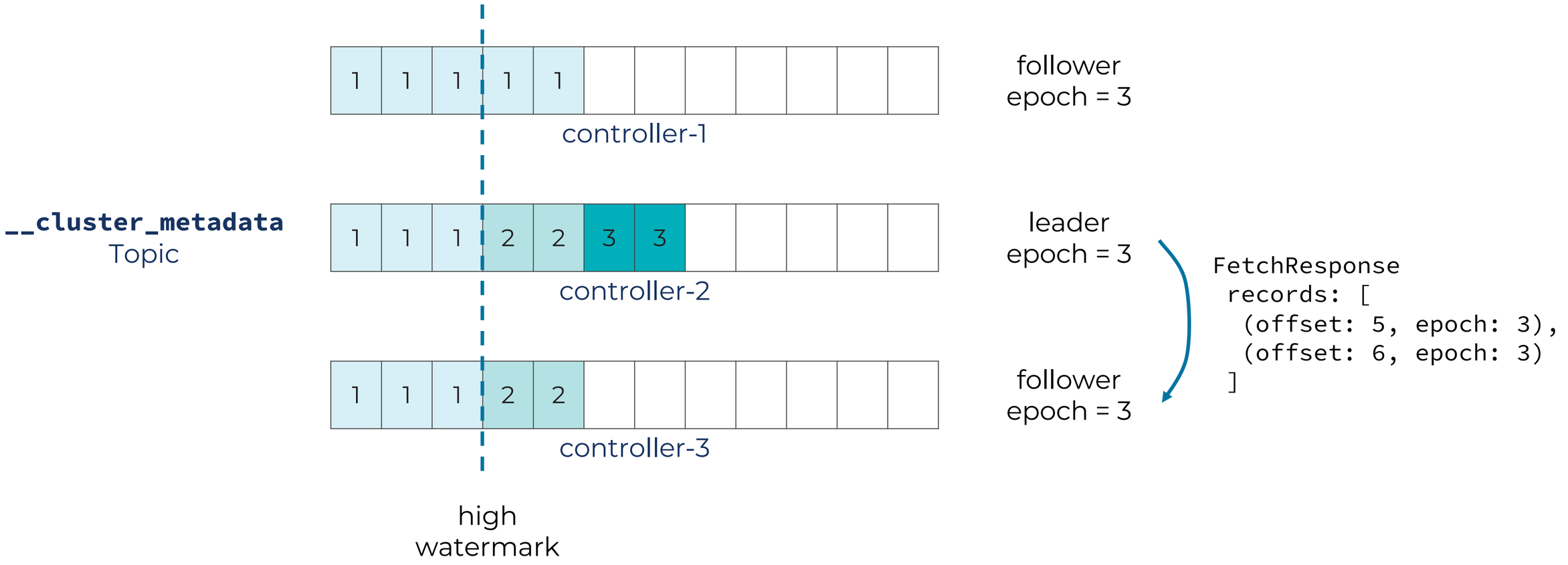
Metadata Replica Reconciliation



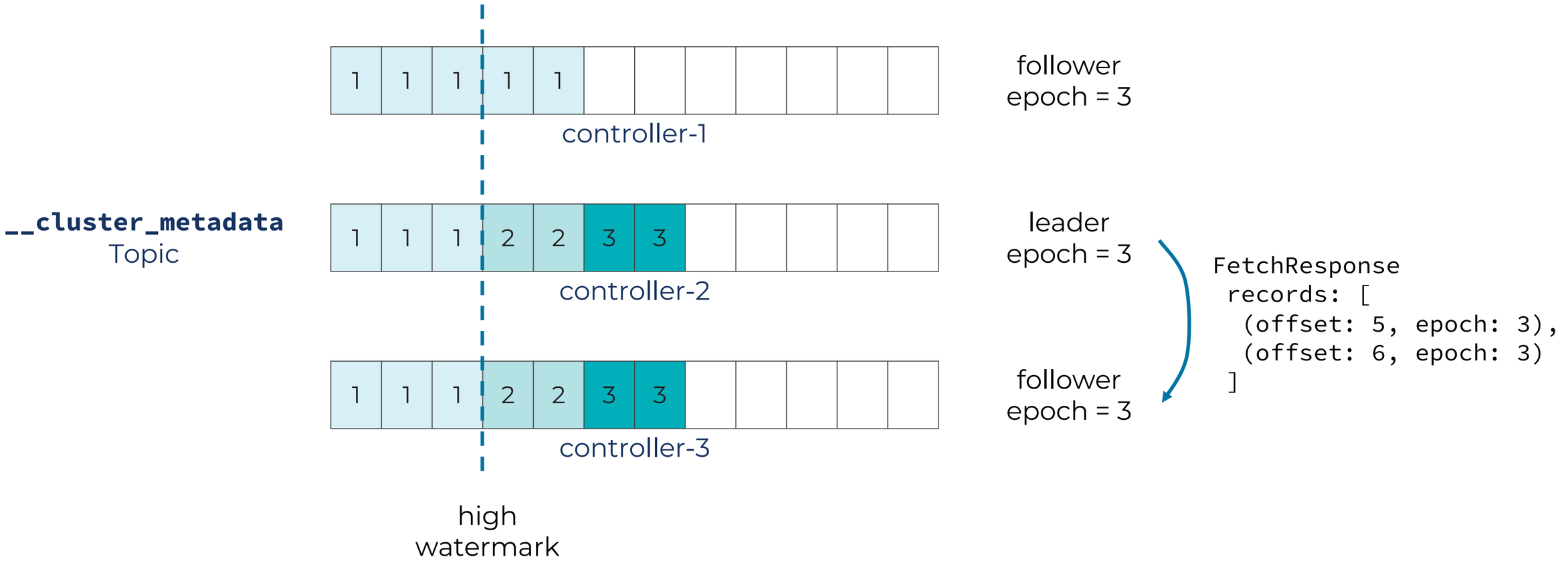
Metadata Replica Reconciliation



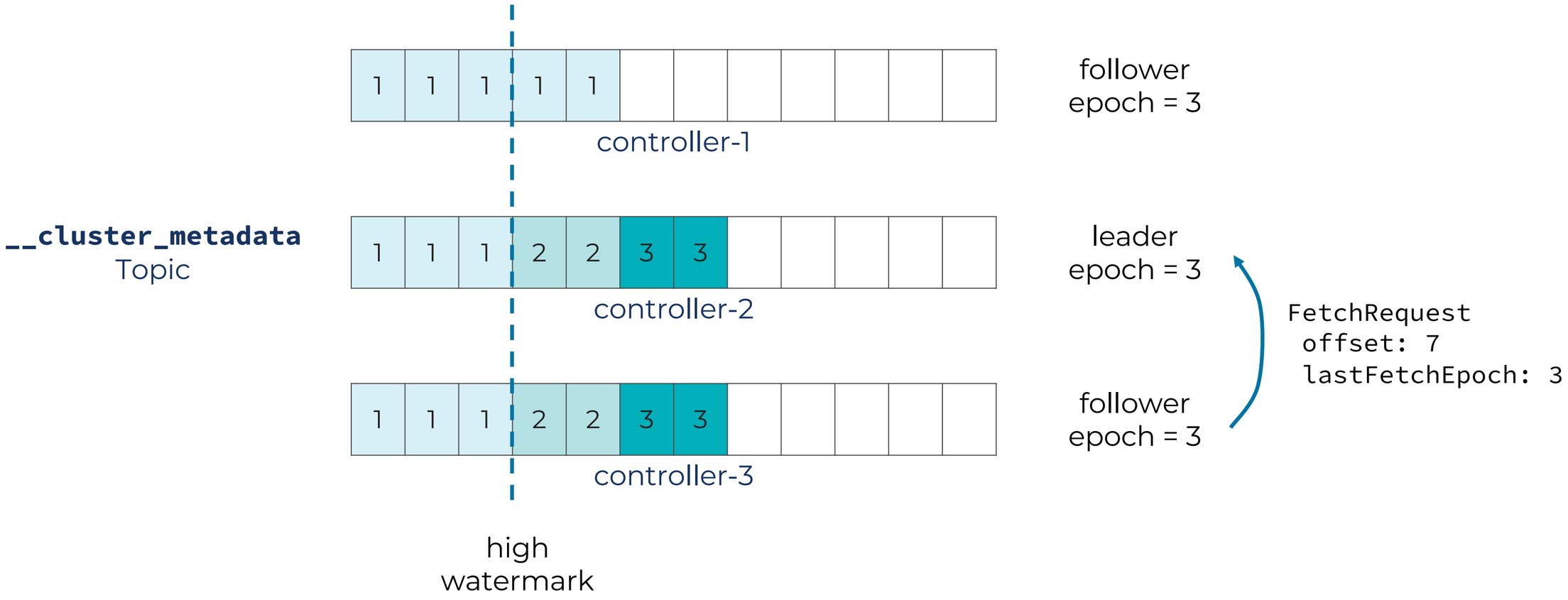
Metadata Replica Reconciliation



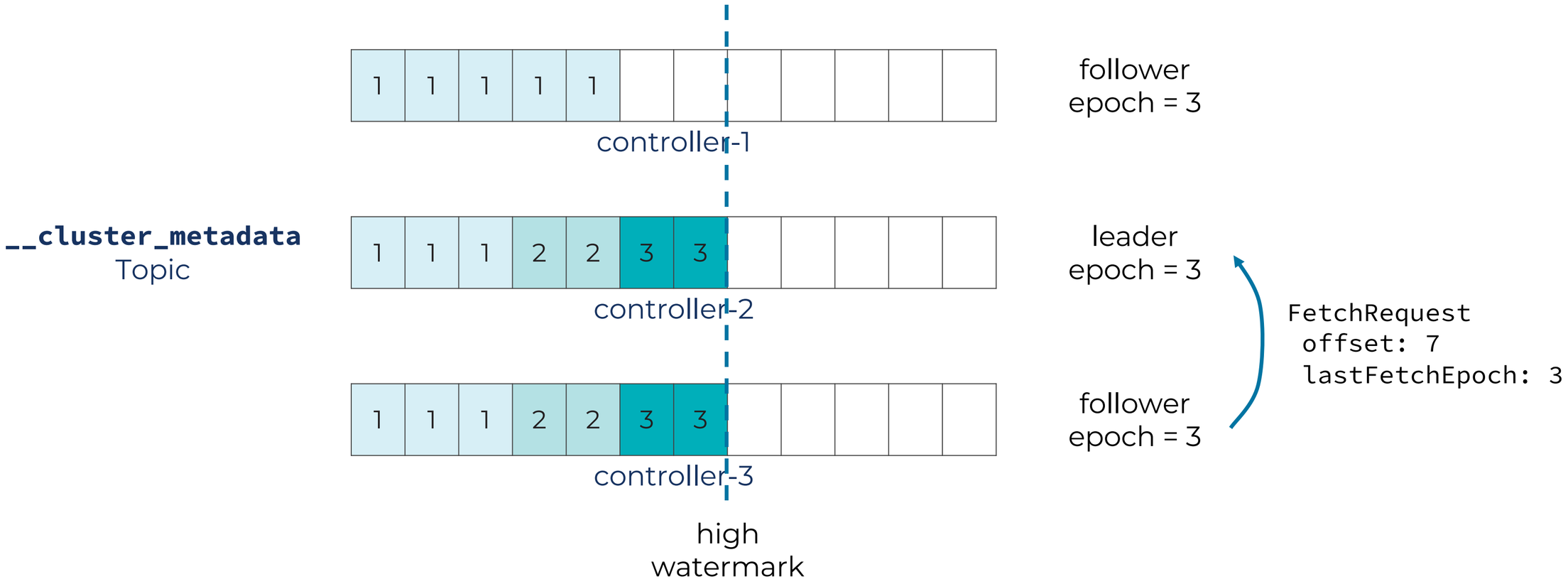
Metadata Replica Reconciliation



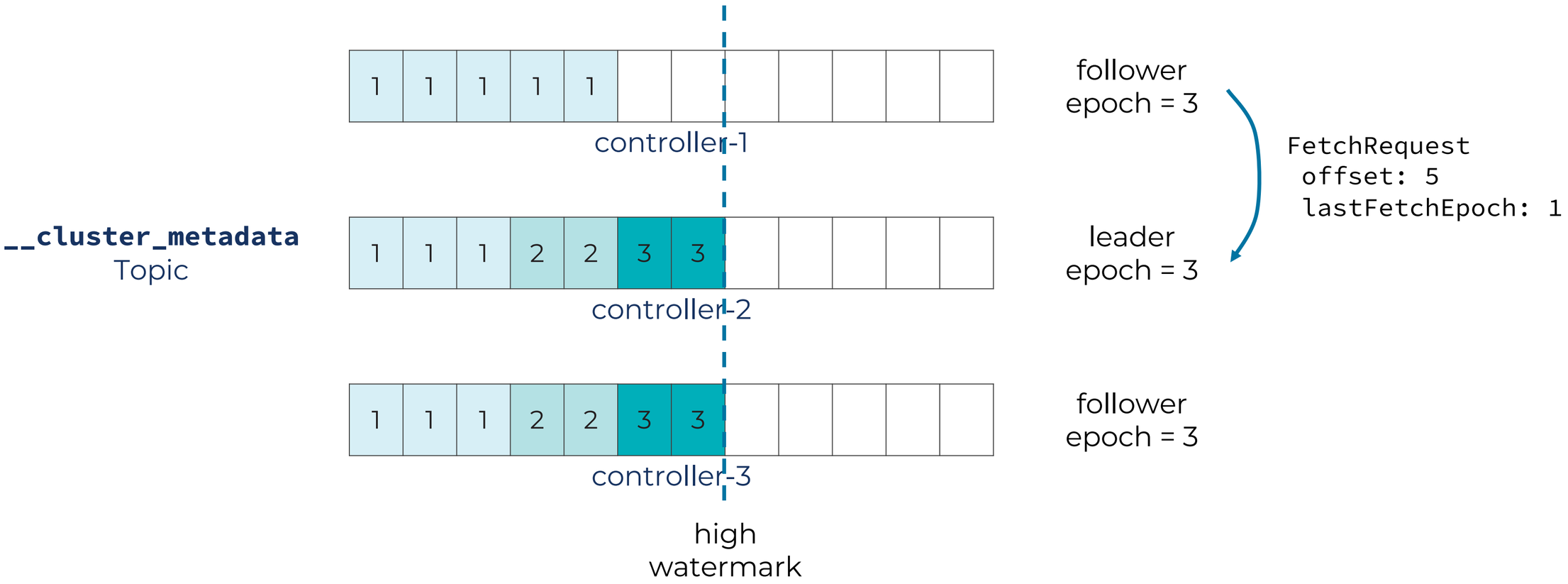
Metadata Replica Reconciliation



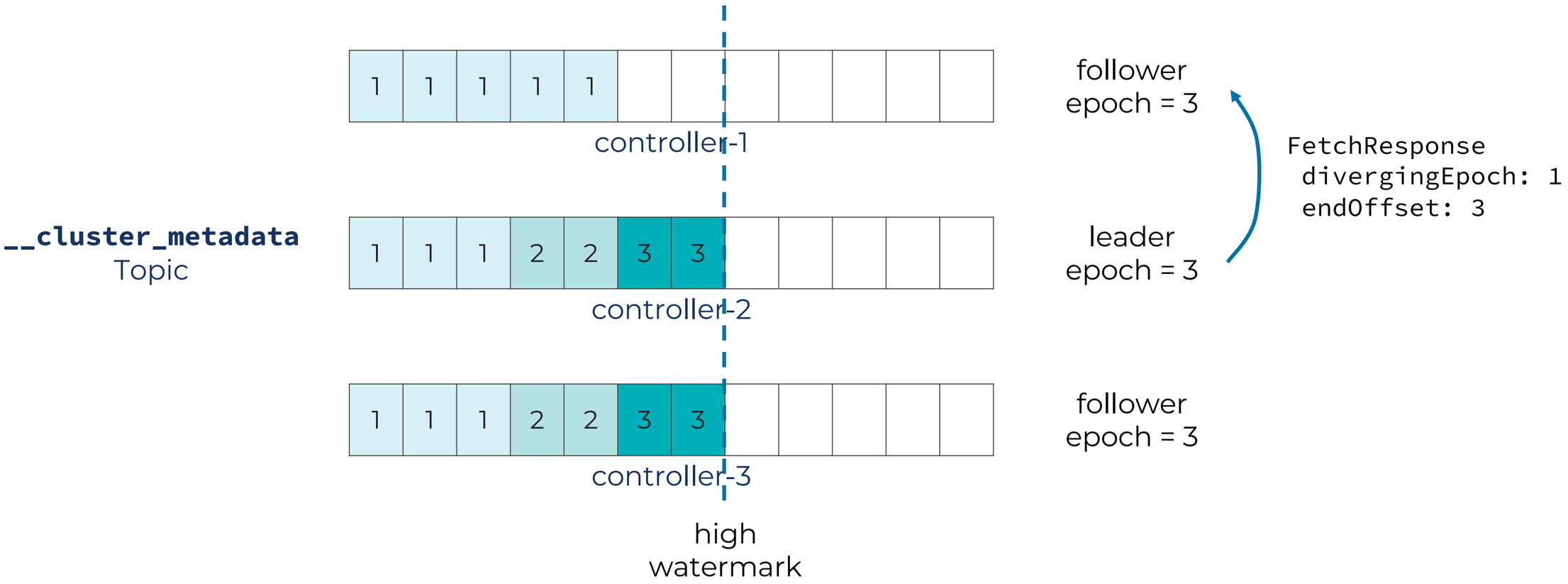
Metadata Replica Reconciliation



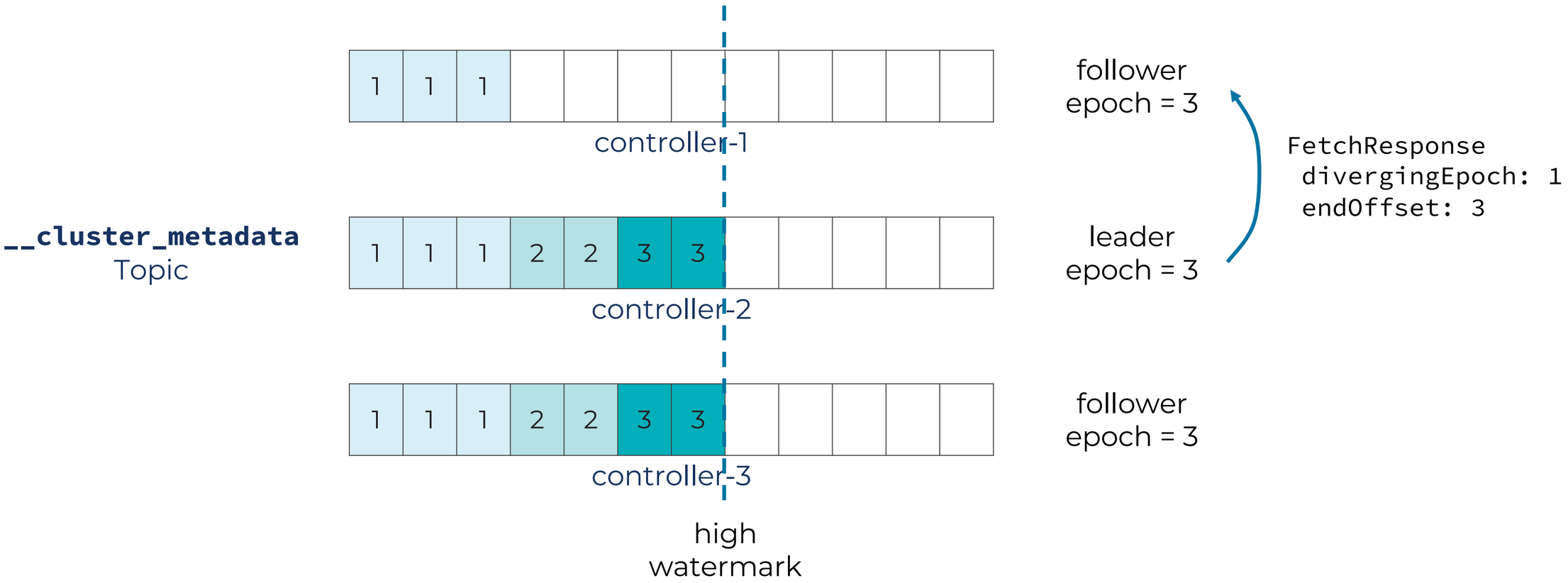
Metadata Replica Reconciliation



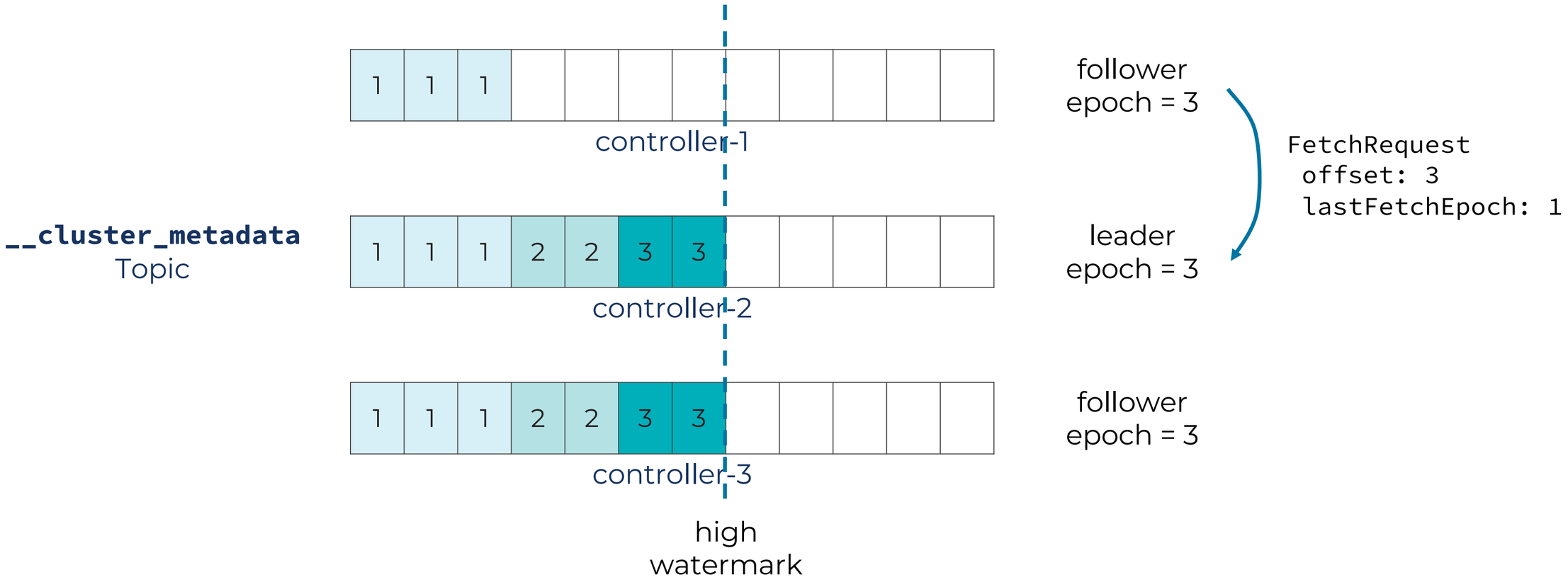
Metadata Replica Reconciliation



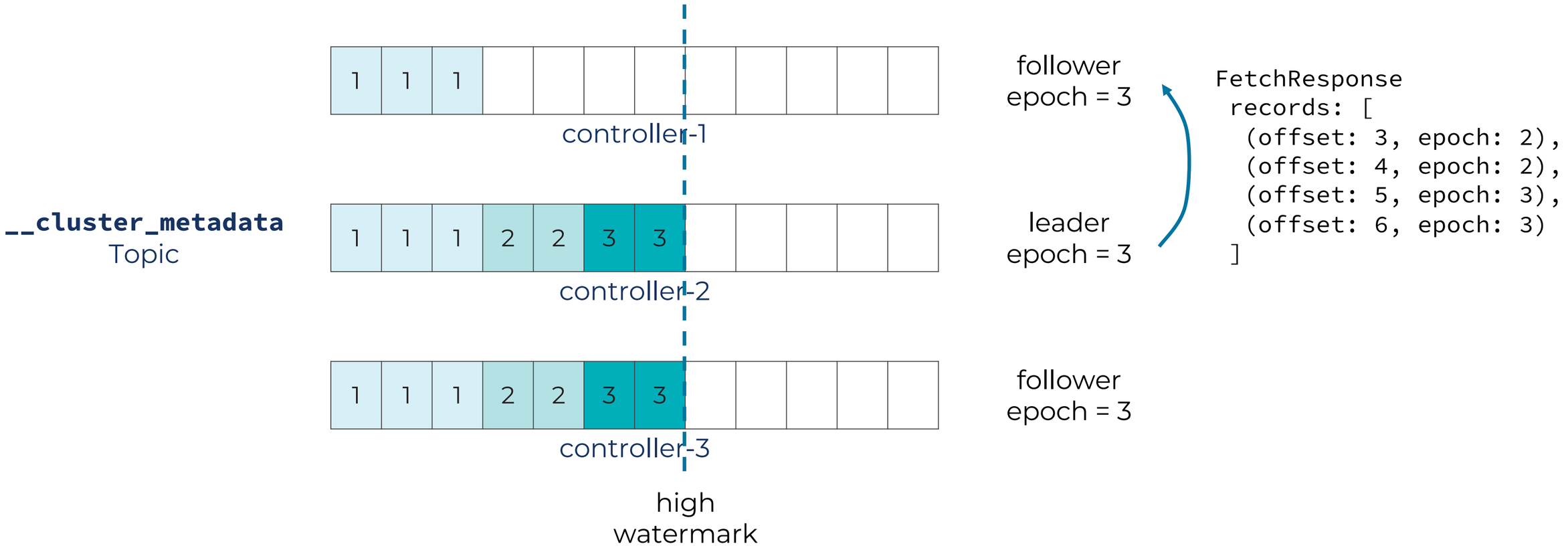
Metadata Replica Reconciliation



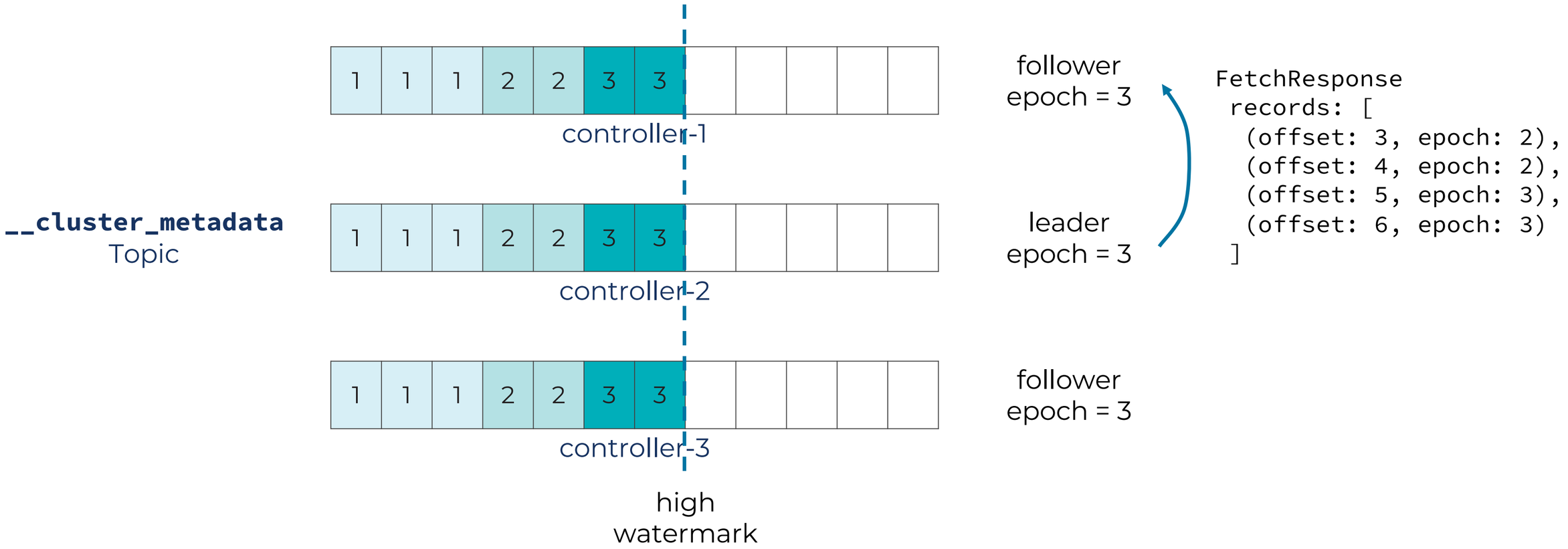
Metadata Replica Reconciliation



Metadata Replica Reconciliation



Metadata Replica Reconciliation



Kraft Mode 방식

KRraft:

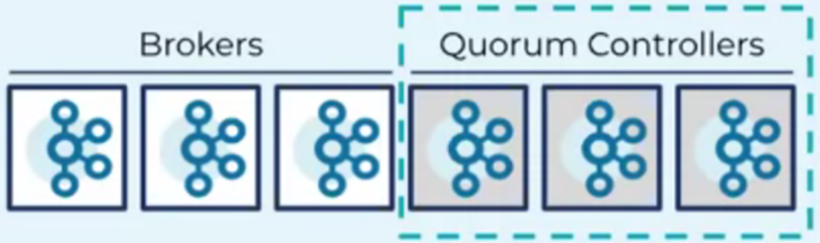
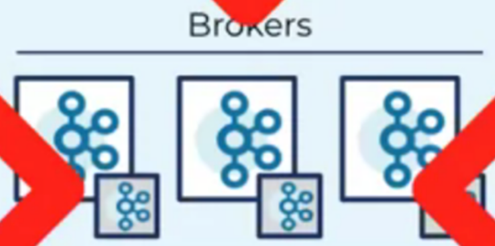
Kafka Raft Metadata Mode. KRaft 는 RAFT 위에 구축된 Kafka의 합의(consensus) 구현체

KRaft 는 두 가지 모드에서 동작:

1. **Internal mode** - Kafka Broker 에서 메시지 처리와 컨센서스를 모두 담당
2. **External mode** – 특정 브로커에게 합의 또는 메시지 처리하도록 할당, 이러한 기능 분리는 더 성능이 좋고 안정적이며, 현재 Zookeeper / Kafka 가 작동하는 방식과 유사함.

   이는 권장사항이며, 기존 노드 수와 동일함

Internal vs External Modes

	KRaft External Mode	KRaft Internal Mode[1]
Overview	Quorum Controllers run on separate nodes, replacing current ZK nodes 	Quorum Controllers run within the Broker / Confluent Server 
Impact on Node Counts	**No node count impact**; 1-to-1 replacement of ZK with Controllers	ZK nodes are removed with no replacement nodes from Controllers
Confluent recommendation	**Default Confluent recommendation for CP customers**	**Not recommended by Confluent - proceed at own risk**

KRaft External mode 를 권고하는 세 가지 이유

Metadata Management

리소스가 메타데이터 관리에 할당되도록(예: ZooKeeper가 수행했던 작업) 브로커와 독립적으로 확장 가능

Scalability

사용자가 더 많은 수의 파티션을 지원하고 사용량과 워크로드가 증가하더라도 잘 작동하도록 함

Reliability / resilience

Quorum Controller는 여전히 이전과 동일한 양의 "작업"을 수행해야 하므로 처리량의 변화로 인해 부하가 걸렸을 때 시스템의 안정성을 보호해야 함

다운타임, 데이터 손실 및 비즈니스 중단의 위험을 방지하고, 안정적이고 탄력적이며 확장 가능한지 확인 해야 한다.

KRaft References

- Blog: Removing ZK from Apache Kafka
- Blog: How to prepare for ZK removal from Apache Kafka
- Blog: Apache Kafka 2.8
- Youtube: The truth about ZK Removal and KIP-500
- Those KIPS:
  - KIP-500 itself
  - KIP-595: specifies the Raft protocol, which is used for the @metadata topic
  - KIP-631: specifies the event-driven controller model, including the new broker lifecycle and the metadata record schemas
  - KIP-590: specifies a new protocol to allow forwarding client requests from brokers to the controller
- Github: KRaft (aka KIP-500) mode Preview Release

Confluent Platform / Cloud

Confluent Platform 7.0 based on Apache Kafka 3.0

Confluent Platform 7.0 provides key features to reinforce our core product pillars

Everywhere

Cluster Linking

Create a real-time bridge to cloud in a simple, secure, reliable, and cost-effective manner

Cloud-Native

Confluent for Kubernetes 2.2

Accelerate time-to-value with cloud-native management of connectors, schemas, and cluster links and quickly scale to meet demand with single-command elastic scaling

KRaft (Preview)

Scale clusters to millions of partitions and simplify your data architecture by removing dependencies on ZooKeeper for metadata management

Complete

Reduced Infra Mode for C3

Reduce infrastructure costs by offloading monitoring from self-managed clusters with Confluent Control Center (C3) to a scalable, cloud-based solution with Health+

ksqlDB Updates

Boost developer productivity with support for foreign key table joins and support for new data types

Confluent Platform 7.0 provides key features to reinforce our core product pillars

Everywhere

Cluster Linking

Create a real-time bridge to cloud in a simple, secure, reliable, and cost-effective manner

Cloud-Native

Confluent for Kubernetes 2.2

Accelerate time-to-value with cloud-native management of connectors, schemas, and cluster links and quickly scale to meet demand with single-command elastic scaling

KRaft (Preview)

Scale clusters to millions of partitions and simplify your data architecture by removing dependencies on ZooKeeper for metadata management

Complete

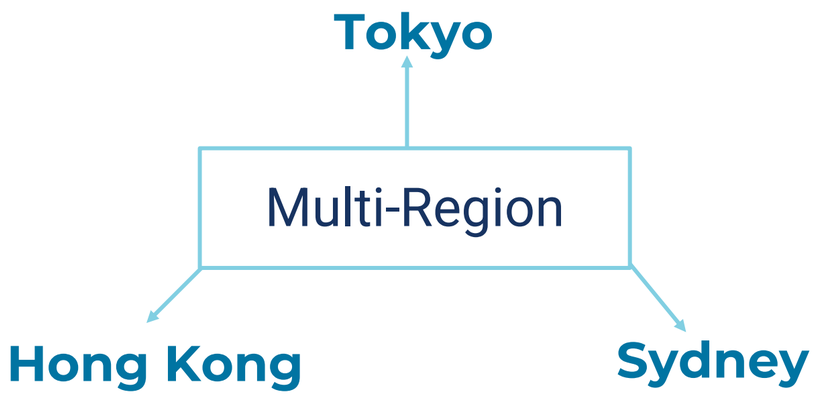
Reduced Infra Mode for C3

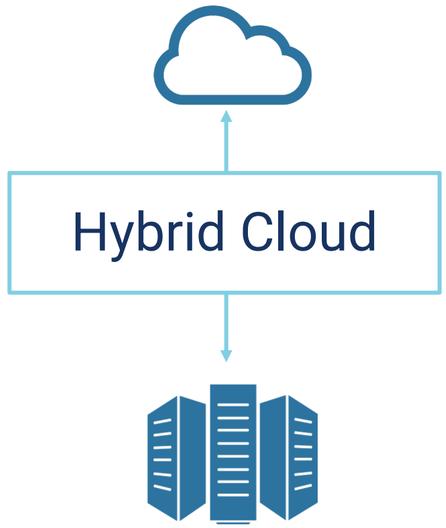
Reduce infrastructure costs by offloading monitoring from self-managed clusters with Confluent Control Center (C3) to a scalable, cloud-based solution with Health+

ksqlDB Updates

Boost developer productivity with support for foreign key table joins and support for new data types

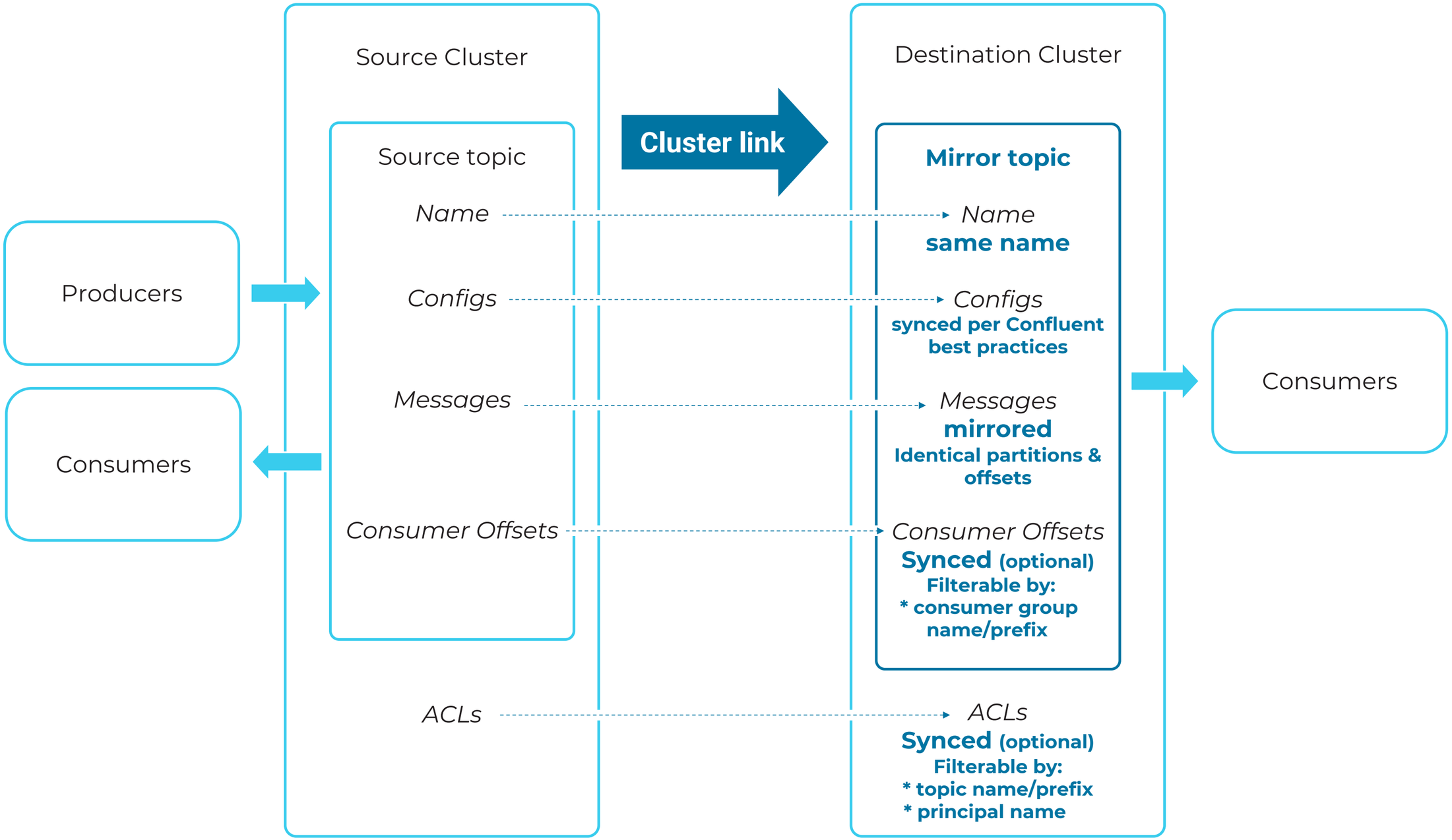
Geo-replication architectures





Geo-replication use cases

- **Global**
- **Bridge to Cloud**
- **Cluster Migration**
- **Edge Aggregation**
- **Data Sharing between LOBs or Organizations**
- **Disaster Recovery (“DR”)**


Source Cluster
Destination Cluster
Source topic
Cluster link
Mirror topic
Producers
Consumers
Name
Name
same name
Configs
Configs
synced per Confluent best practices
Messages
Messages
mirrored
Identical partitions & offsets
Consumer Offsets
Consumer Offsets
Synced (optional)
Filterable by:
* consumer group name/prefix
ACLs
ACLs
Synced (optional)
Filterable by:
* topic name/prefix
* principal name
Consumers

Cloud-native geo-replication is *API-driven*

Replicator & MM2 are configuration-based. Adding topics / changing settings requires a restart.

Cluster Linking has an intuitive **CLI**

```
> kafka-cluster-links  --create    --link nyc-to-iad    \
    --cluster-id lkc-12345          --config-file link.config  \
    --bootstrap-server <dest>:9092 --command-config iad-auth.config
```

and a first-class **REST API**

```
> POST <REST-API>:443
/kafka/v3/clusters/<cluster-id>/links/nyc-to-iad/mirrors/trades
{
    "source_topic_name": "trades",
    "replication_factor": 3
}
```

Confluent Platform 7.0 provides key features to reinforce our core product pillars

Everywhere

Cluster Linking

Create a real-time bridge to cloud in a simple, secure, reliable, and cost-effective manner

Cloud-Native

Confluent for Kubernetes 2.2

Accelerate time-to-value with cloud-native management of connectors, schemas, and cluster links and quickly scale to meet demand with single-command elastic scaling

KRaft (Preview)

Scale clusters to millions of partitions and simplify your data architecture by removing dependencies on ZooKeeper for metadata management

Complete

Reduced Infra Mode for C3

Reduce infrastructure costs by offloading monitoring from self-managed clusters with Confluent Control Center (C3) to a scalable, cloud-based solution with Health+

ksqlDB Updates

Boost developer productivity with support for foreign key table joins and support for new data types

Build your own private cloud Kafka service with the latest version of Confluent for Kubernetes

Confluent for Kubernetes provides a complete, declarative API to deploy and operate Confluent Platform, enabling you to build your own private cloud Kafka service

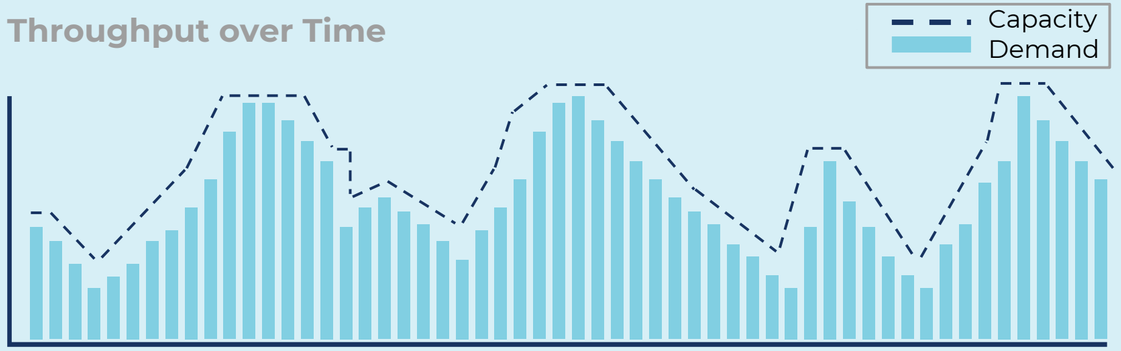
Available for:

- VMware Tanzu
- Red Hat OpenShift
- Kubernetes distributions meeting CNCF standards

Expanded API-driven Operations

- Connectors
- Schemas
- Cluster Links

Enhanced Elasticity with Shrink API



Self-Balancing Clusters Shrink Update

The new and improved cluster shrink backend has been ported from ccloud to CP 7.0

- This is effectively the same shrink backend ccloud uses.

Old Shrink:

- Would shut down the broker before doing reassignments.
- This would cause collisions with restart policies in both Kafka and Kubernetes.

New Shrink:

- Brokers (replicas) remain online but become “excluded” from changes.
- This effectively isolates the broker from the rest of cluster while reassignment and leadership changes are happening.
- CFK will directly use these new internal APIs to shrink clusters which will support multi-broker removals.

Confluent Platform 7.0 provides key features to reinforce our core product pillars

Everywhere

Cluster Linking

Create a real-time bridge to cloud in a simple, secure, reliable, and cost-effective manner

Cloud-Native

Confluent for Kubernetes 2.2

Accelerate time-to-value with cloud-native management of connectors, schemas, and cluster links and quickly scale to meet demand with single-command elastic scaling

KRaft (Preview)

Scale clusters to millions of partitions and simplify your data architecture by removing dependencies on ZooKeeper for metadata management

Complete

Reduced Infra Mode for C3

Reduce infrastructure costs by offloading monitoring from self-managed clusters with Confluent Control Center (C3) to a scalable, cloud-based solution with Health+

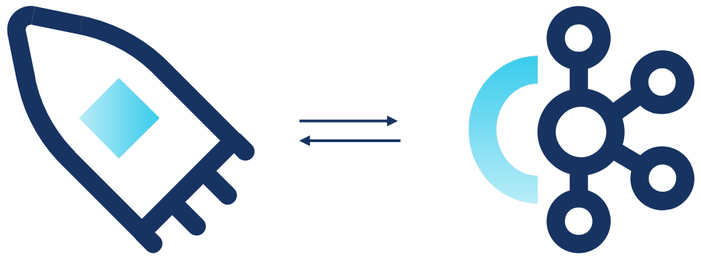
ksqlDB Updates

Boost developer productivity with support for foreign key table joins and support for new data types

Easily build real-time applications with expanded functionality in ksqlDB

ksqlDB is a streaming database purpose-built for developing real-time applications that leverage stream processing, enabling you to **build a complete real-time application with just a few SQL statements**

ksqlDB **Kafka**



Compute *Storage*

Foreign-key table joins

```
t0 JOIN t1 ON t0.key = t1.value
```

ksqlDB now supports 1:N joins

t_0key_0	t_1value_0
t_0key_1	t_1value_1
t_0key_2	t_1value_1

DATE and TIME data types

DAT
E

TIM
E

어디에나 배포할 수 있는 완전한 선택의 자유

Self-Managed Software

Confluent Platform

Apache Kafka®의 엔터프라이즈 배포

모든 플랫폼, 온 프레미스 또는 클라우드에 배포

Fully-Managed Service

Confluent Cloud

클라우드를 위해 재설계된
Cloud-Native Apache Kafka®

주요 퍼블릭 클라우드에서 사용 가능

aws

Google Cloud Platform

Microsoft Azure

Private Cloud

Hybrid Cloud

Public/Multi-Cloud

Any questions?

Stay in touch!

Try Confluent
cnfl.io/download

Confluent Blog
cnfl.io/blog

Community
cnfl.io/meetups

CONFLUENT