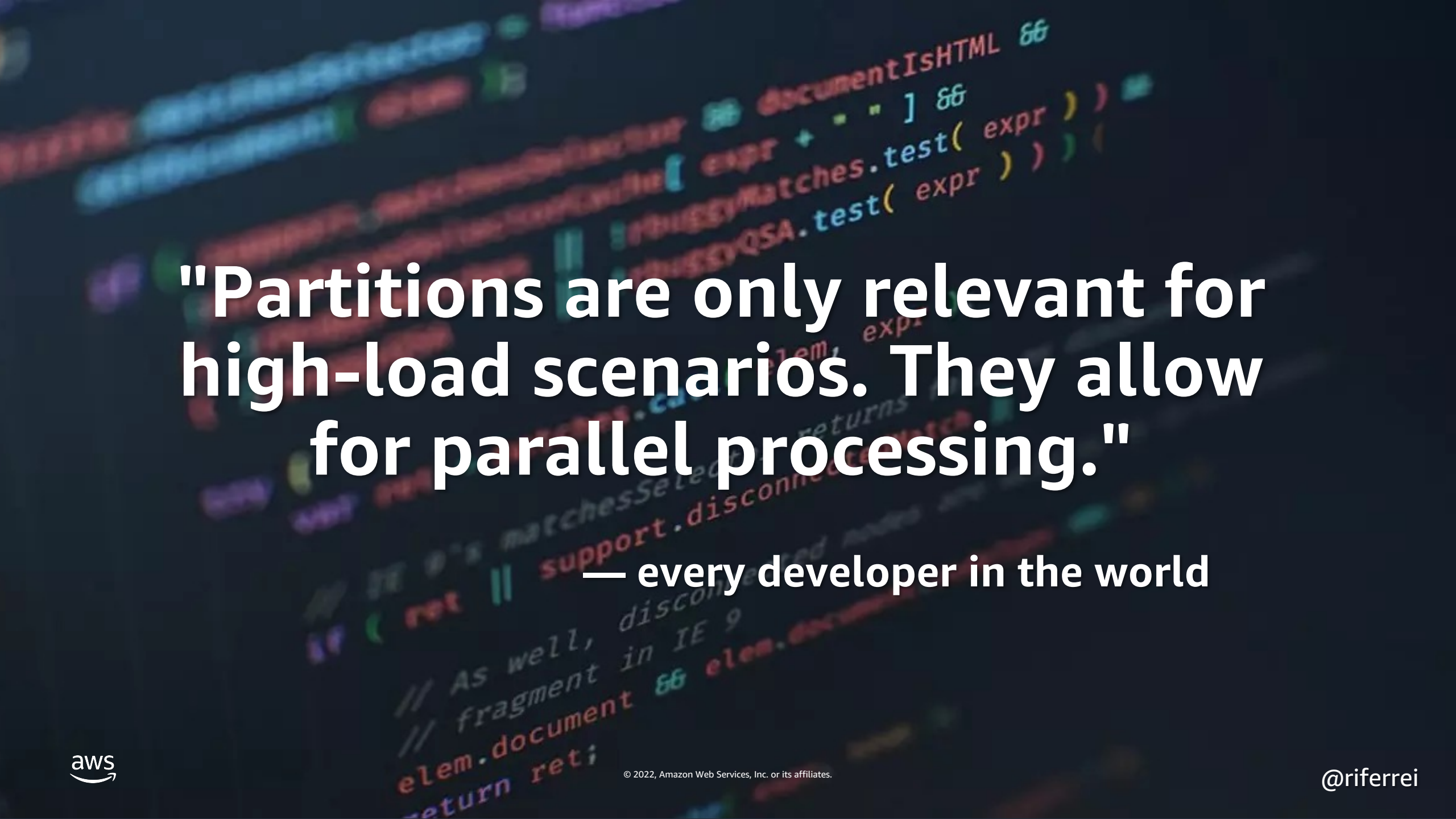
In the land of the sizing,

the one-partition Kafka topic is king

Ricardo Ferreira

Senior Developer Advocate
Amazon Web Services

@riferrei



"Partitions are only relevant for high-load scenarios. They allow for parallel processing."

— every developer in the world

@riferrei

@riferrei

In reality, Kafka partitions are:

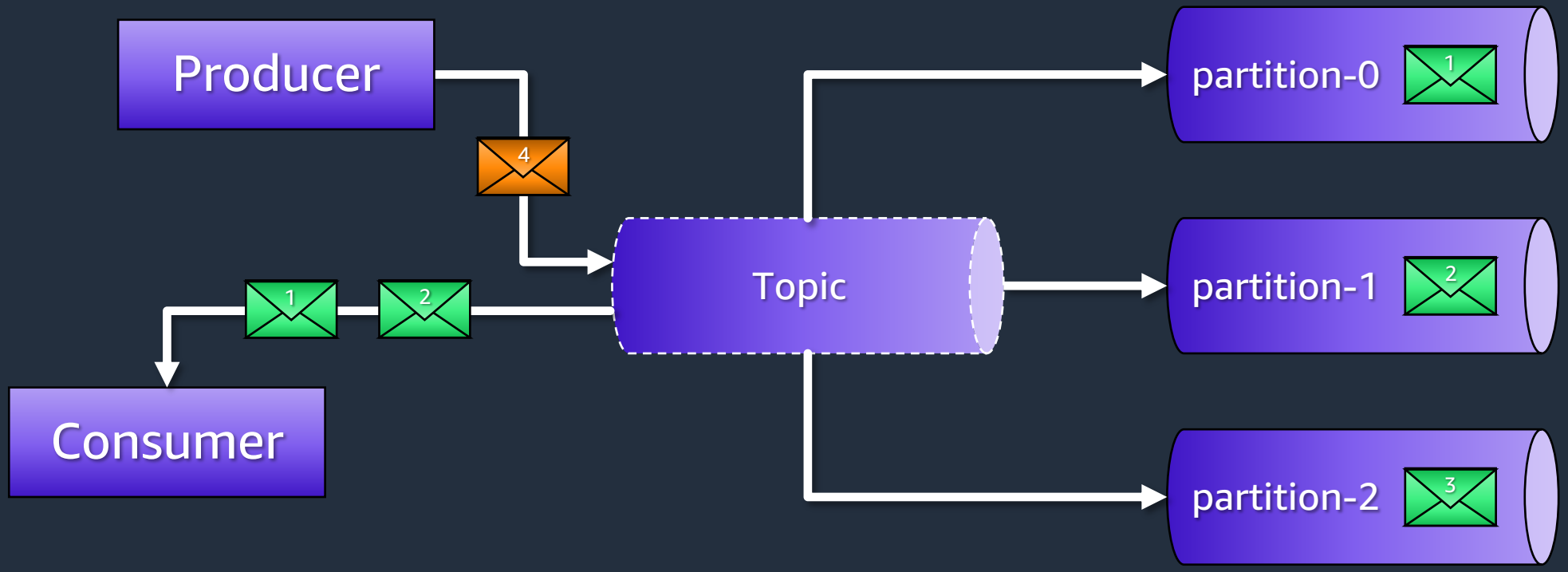
1. Unit-Of-Storage

2. Unit-Of-Parallelism

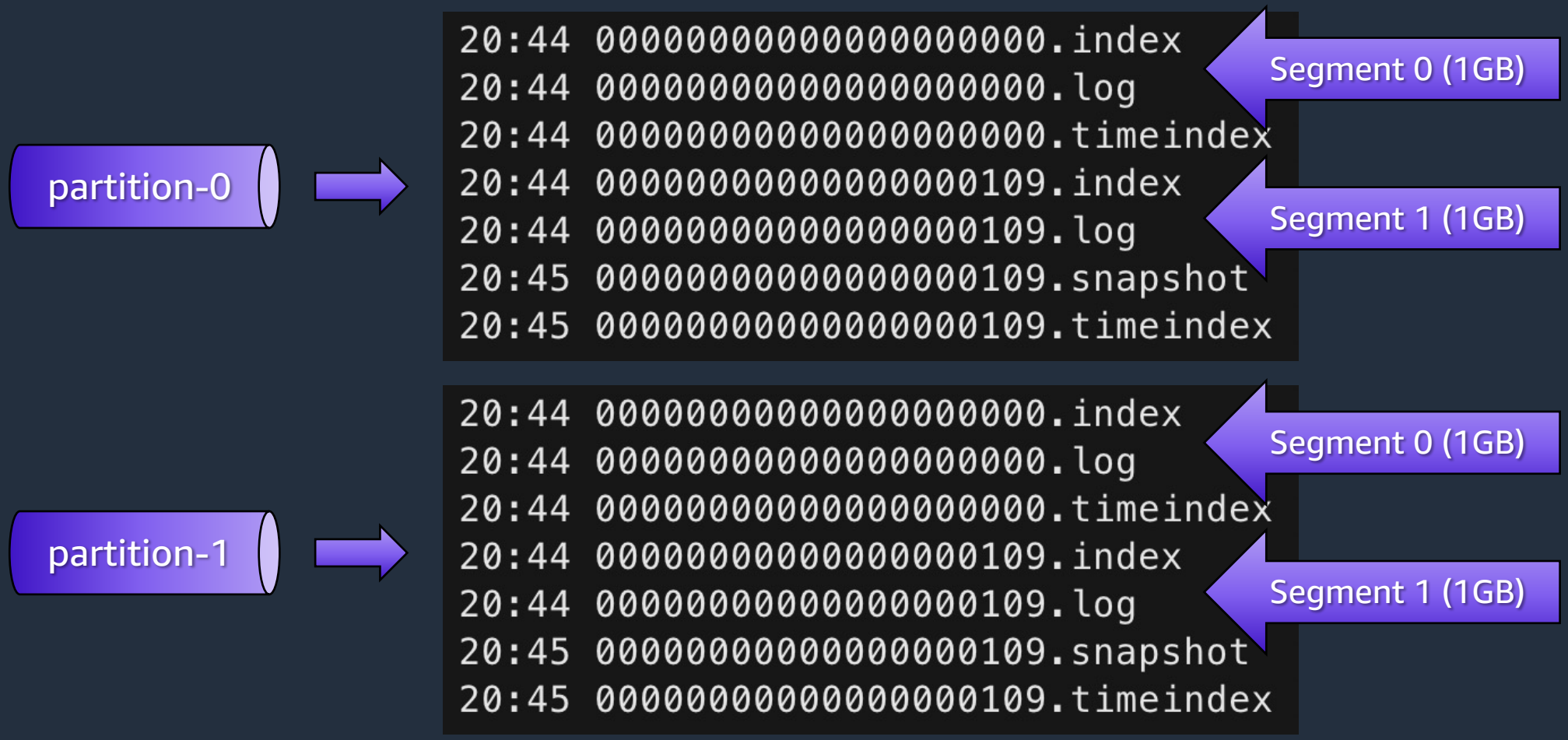
3. Unit-Of-Durability

Partitions as the unit-of-storage

Topics are just high-level abstractions

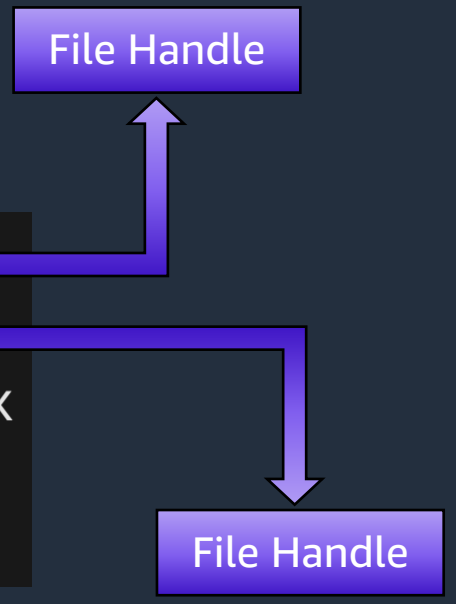


Partitions and segments



Segments and file handles

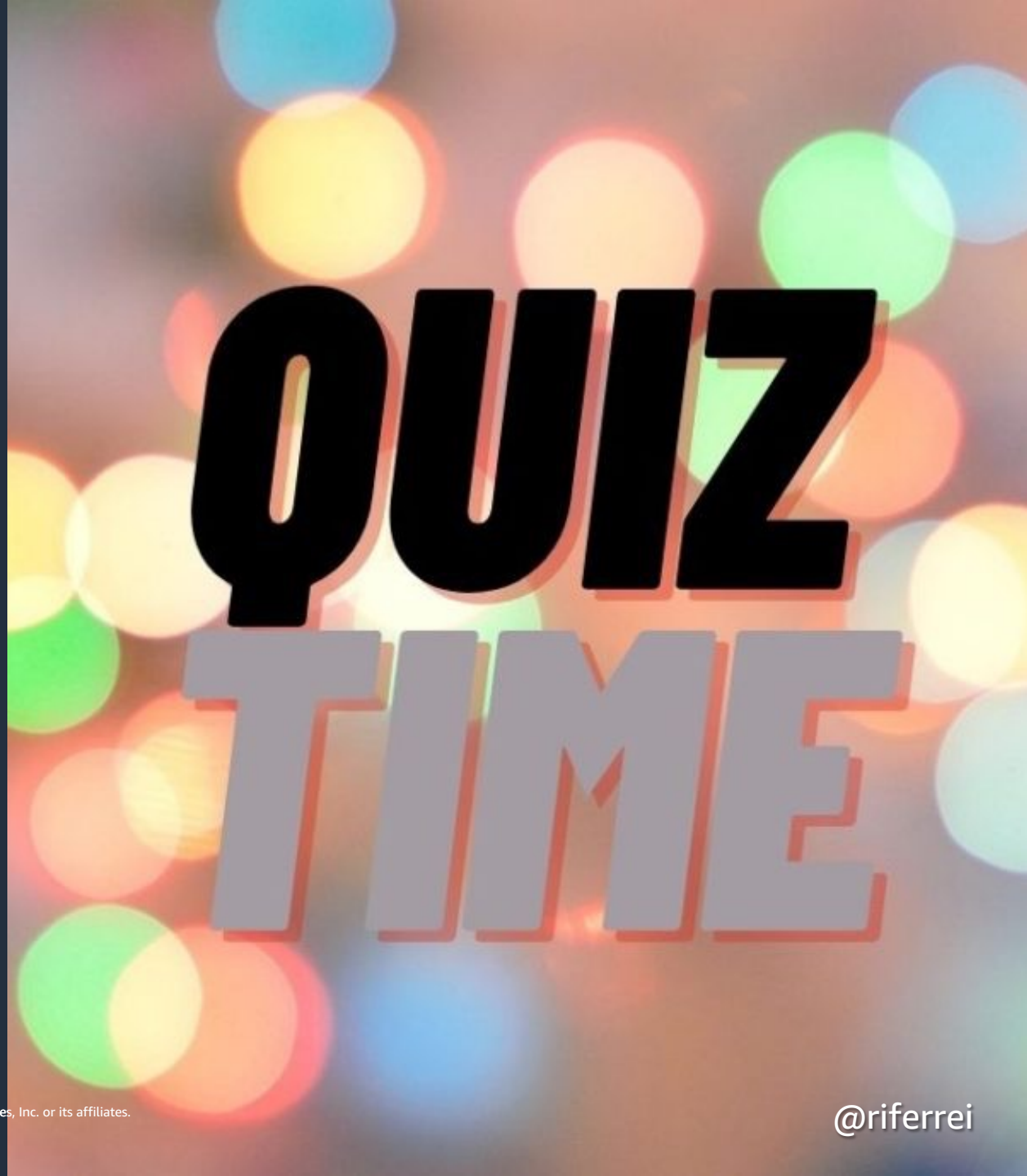
```
 10M Sep  9 16:59 00000000000000000000.index
250M Sep  9 17:16 00000000000000000000.log
 10M Sep  9 16:59 00000000000000000000.timeindex
  0B Sep  9 16:59 leader-epoch-checkpoint
 43B Sep  9 16:59 partition.metadata
```



Quiz time:

- One broker with **1024** file handles.
- One topic with only **one** partition.
- Producer writes data **every minute**.
- Each write holds **250MB** of data.

What is going to happen? 🤔

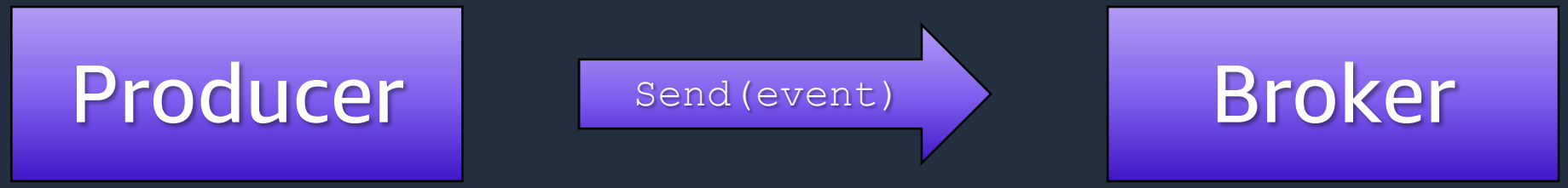


@riferrei

Result:

In **48 hours**, this broker will run out of file handles and **crash**. Because each **log segment** grow up to **1GB**.

Partitions from the developer perspective



KIP-794: How partitioning works

1. Partition set

If the event has an partition set, then it goes to that partition.

2. Partitioner.class

Otherwise it is based on the logic from the configured partitioner.

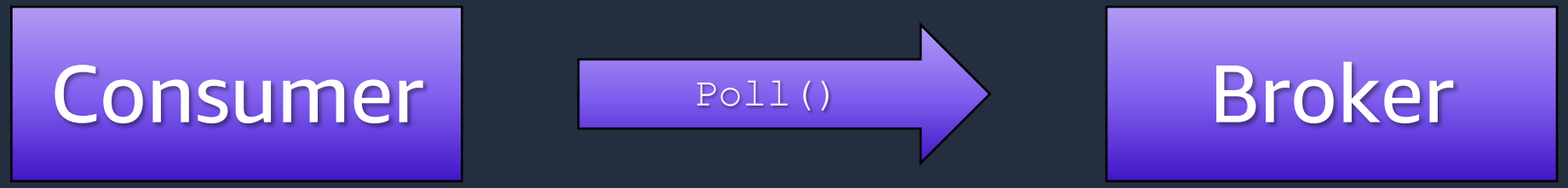
3. Using keys

Otherwise it will use the assigned key to figure the partition.

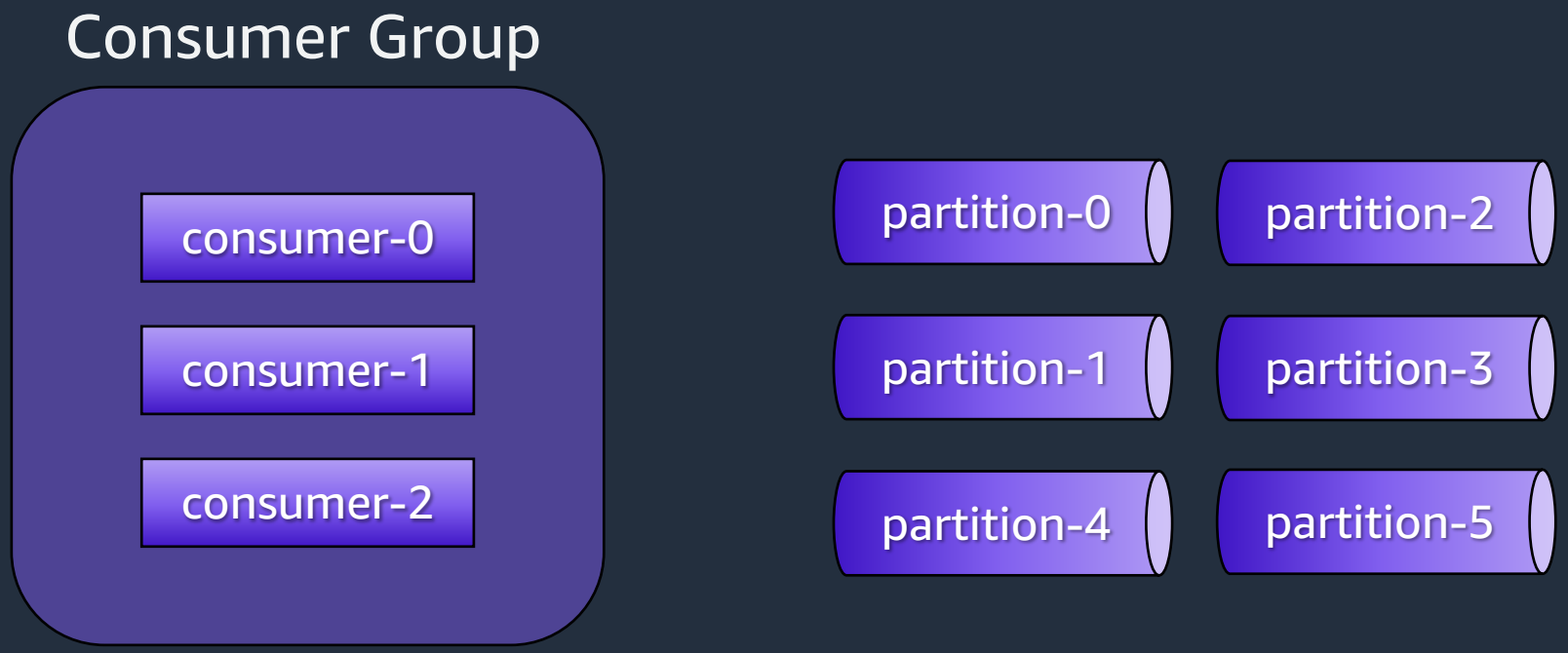
4. Broker load

Last fallback is based on factors like broker load, amount of data produced to each partition, etc.

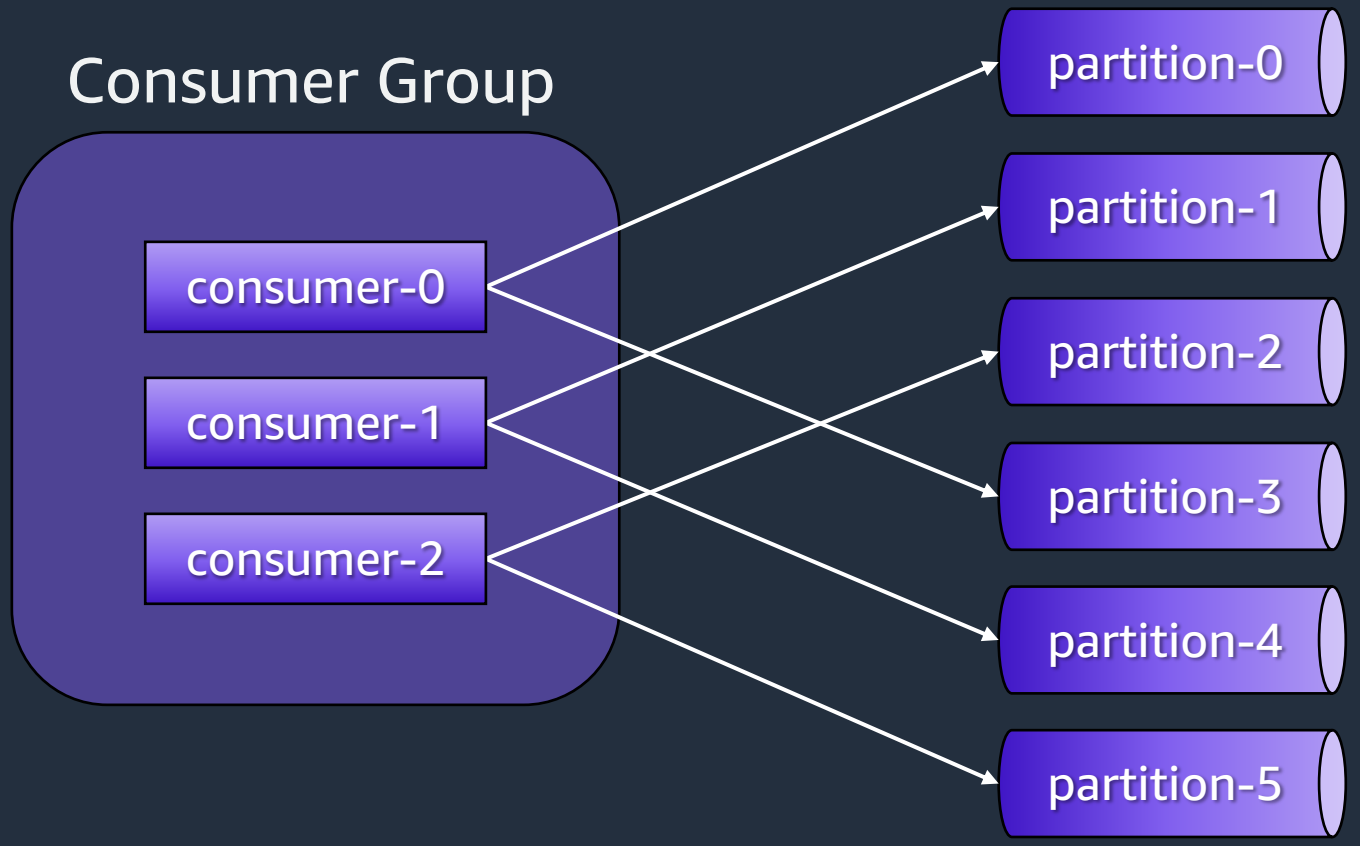
Partitions from the developer perspective



How assignment works



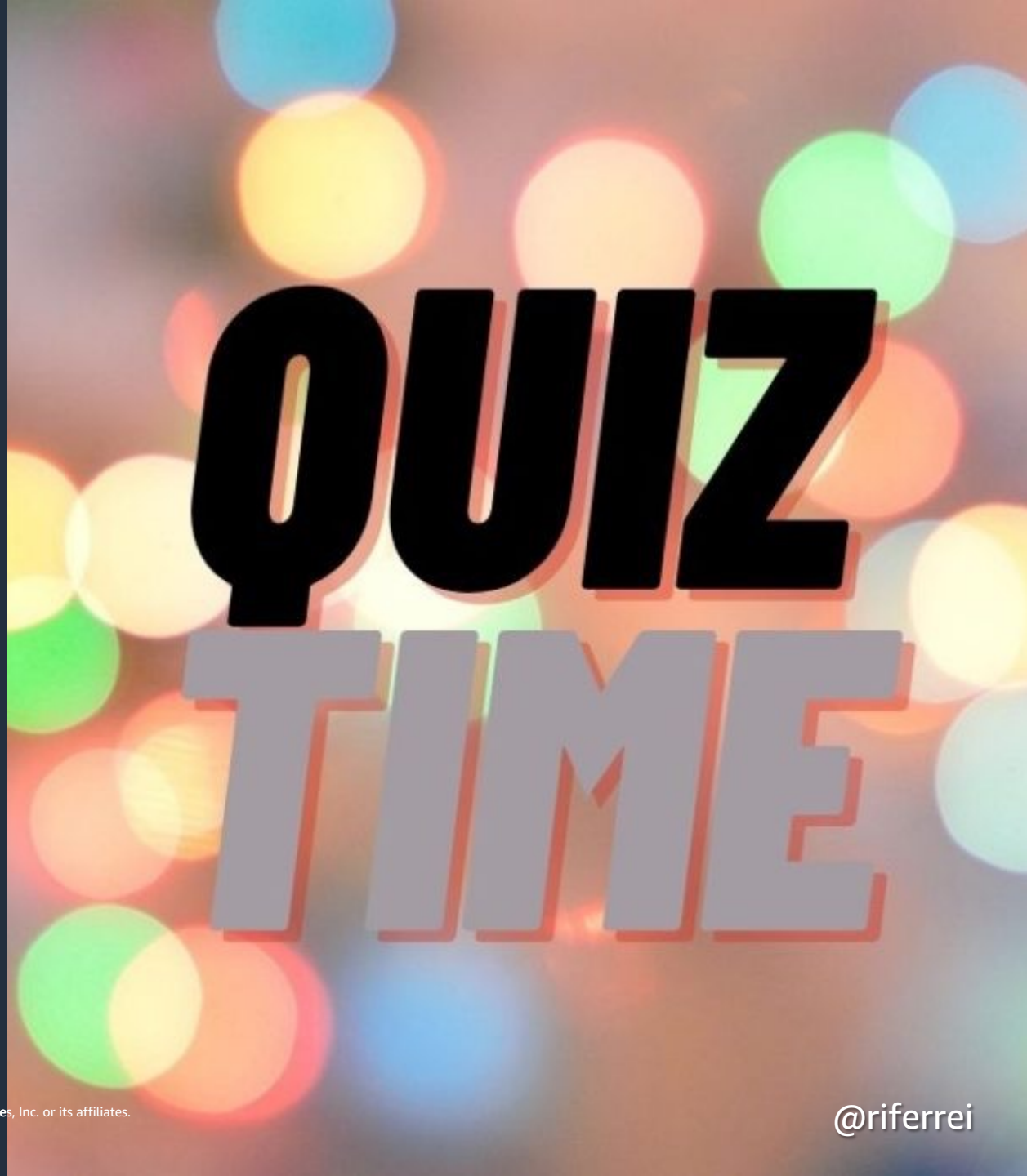
How assignment works: RangeAssignor



Quiz time:

- Single topic with **6** partitions.
- Consumer group with **3** consumers.
- **RangeAssignor** is being used.
- One of the consumers **dies**.

What is going to happen? 🤔

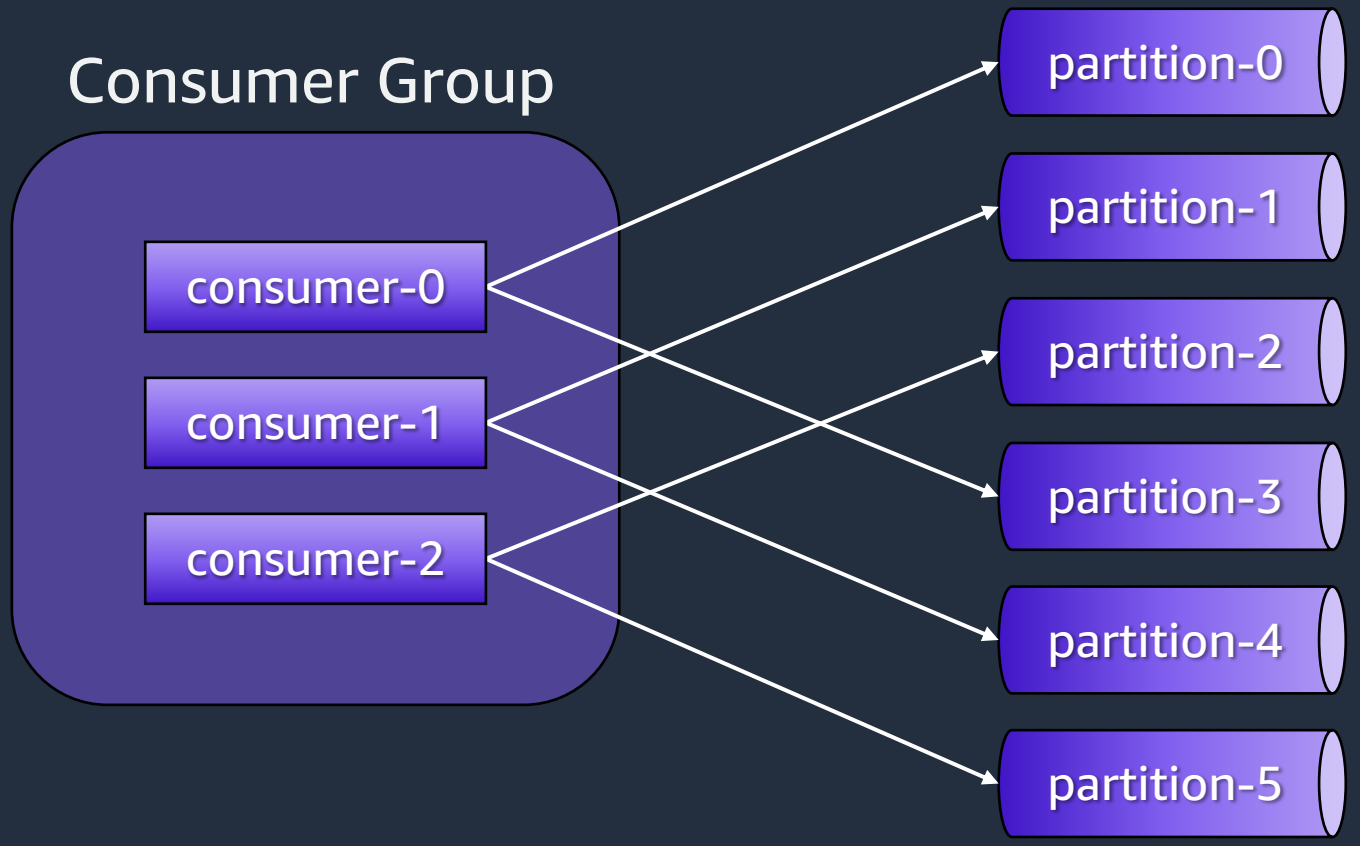


Result:

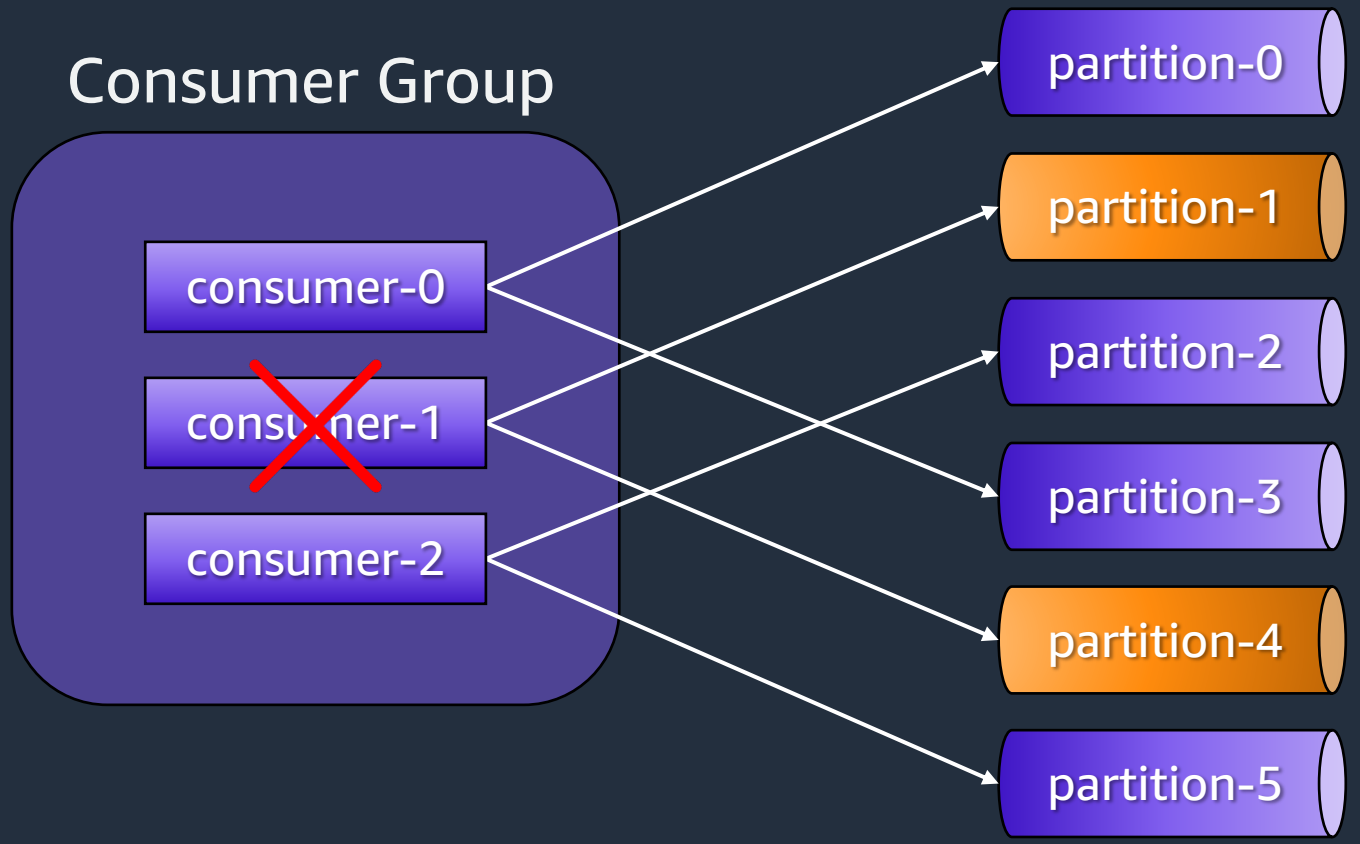
The remaining **2** consumers will **stop** processing events until **all partitions** are **reassigned**.

This problem is known as **stop-the-world pauses**. Reason why you must used the **CooperativeStickyAssignor** as much as possible for new Kafka clusters.

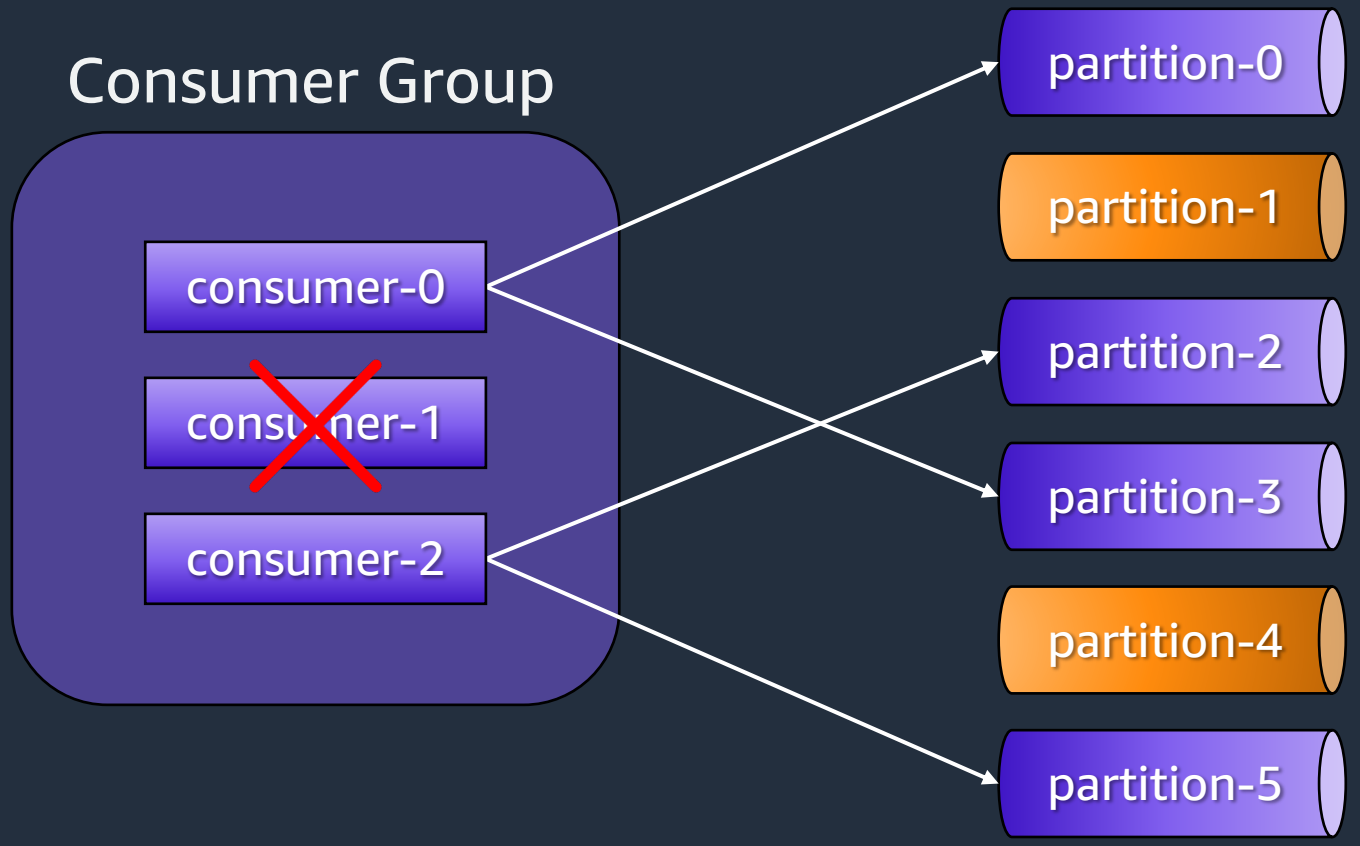
How assignment works: CooperativeStickyAssignor



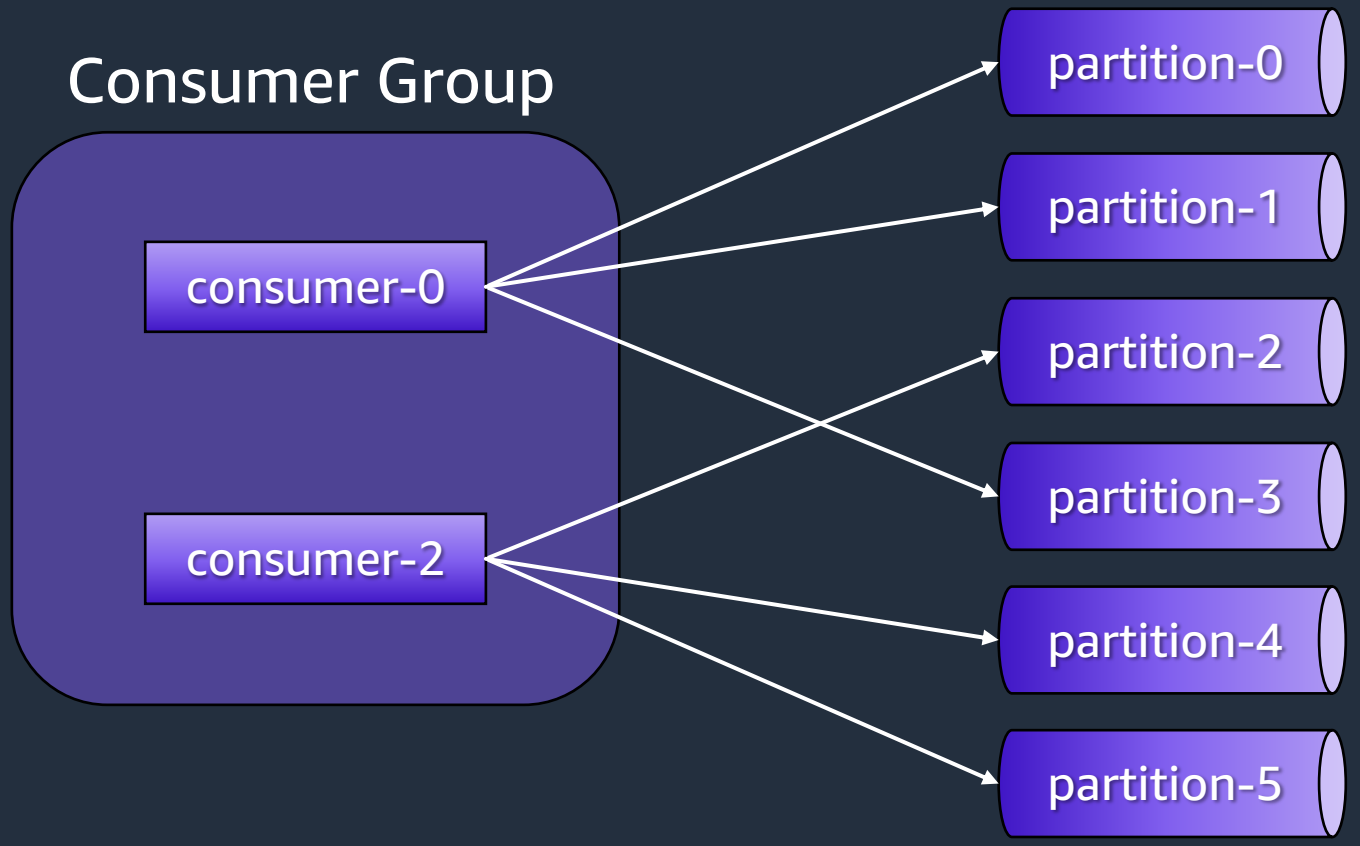
How assignment works: CooperativeStickyAssignor



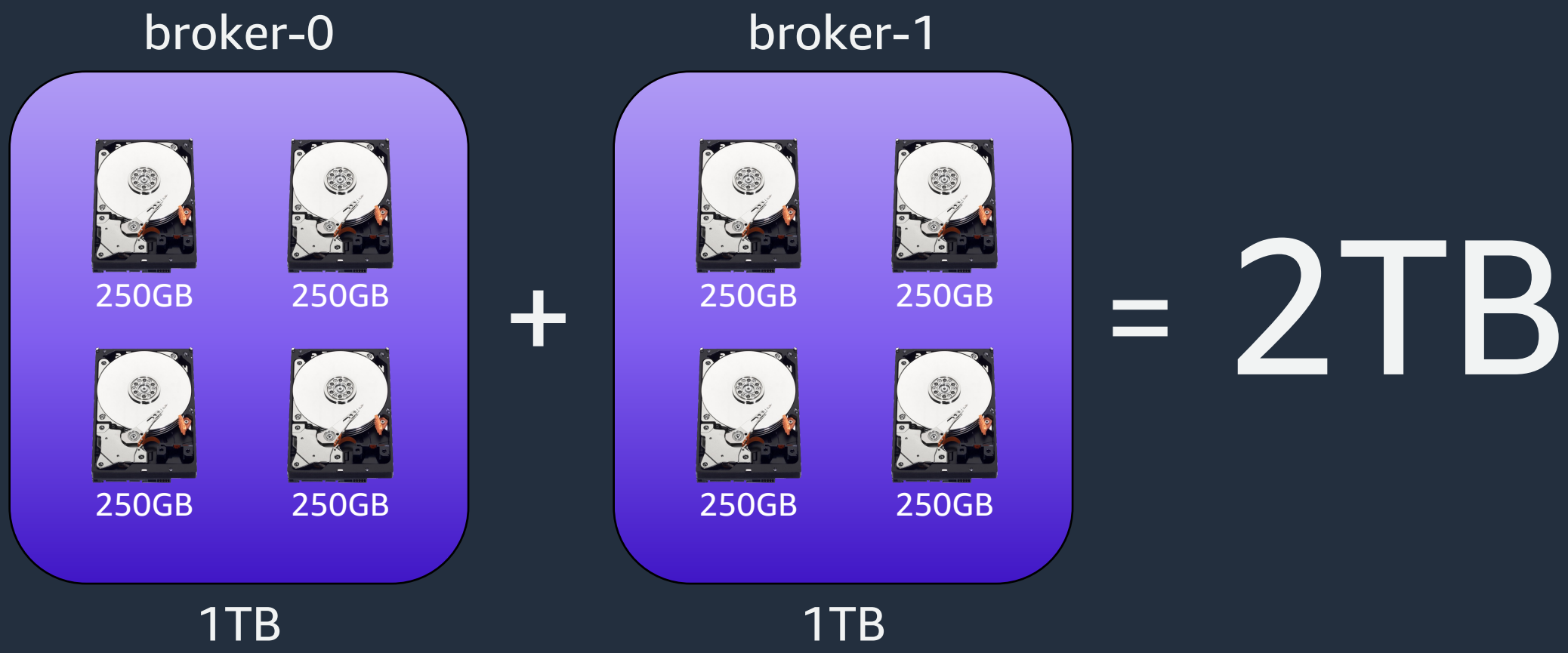
How assignment works: CooperativeStickyAssignor



How assignment works: CooperativeStickyAssignor



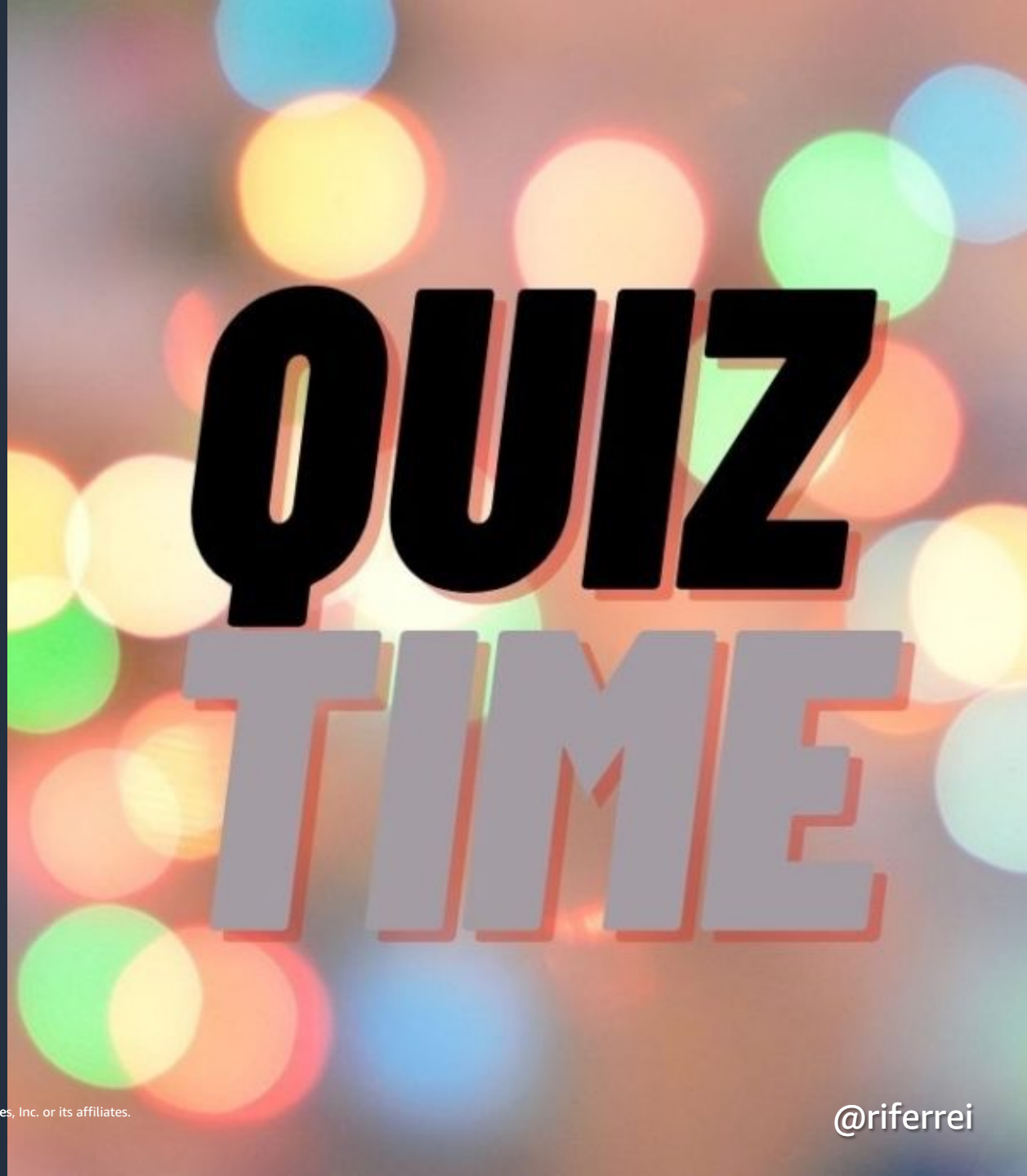
Partitions from the storage perspective



Quiz time:

- Single cluster with **2** brokers.
- One topic with **4** partitions.

How partitions are distributed? 🤔



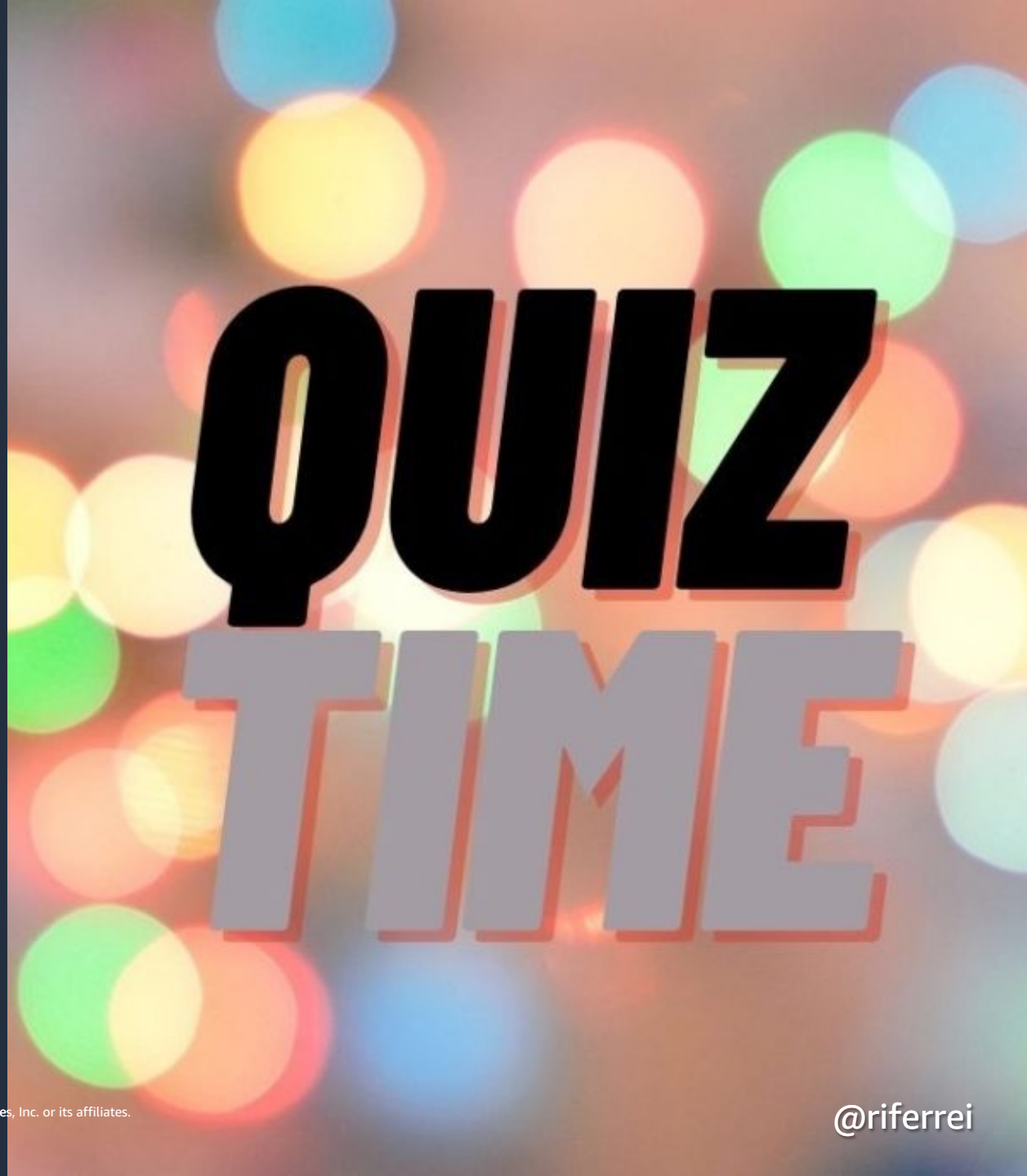
@riferrei

Partitions are distributed atomically

Quiz time:

- Single cluster with **2** brokers.
- One topic with **4** partitions.
- You add more **2** brokers.

How partitions are distributed? 🤔



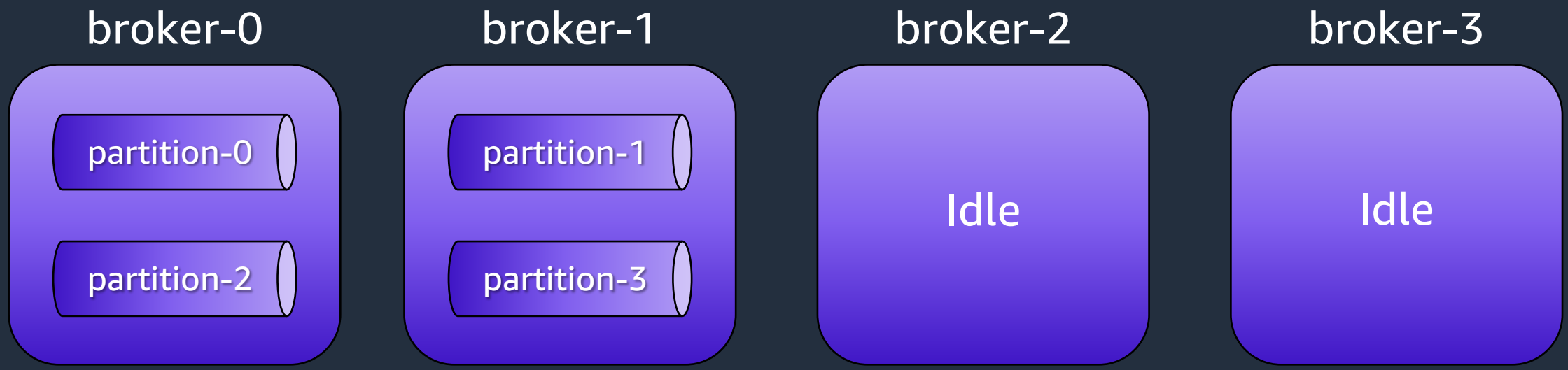
@riferrei

Result:

There is **no distribution**. The partitions will be sitting in the **first two brokers**.

Partitions are **not automatically re-assigned** as you add new brokers. You have to do this manually.

Partitions from the storage perspective



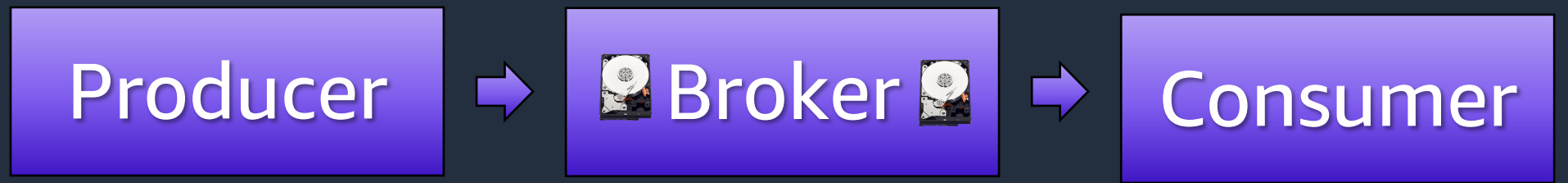
How to reassign the partitions?

1. **Bounce the entire cluster:** leads to unavailability but once its back the partitions are reassigned.
2. **kafka-reassign-partitions:** CLI tool you where you can reassign the partitions with the cluster online.

Demo time!

Partitions as the unit-of-parallelism

Understanding read throughput issues



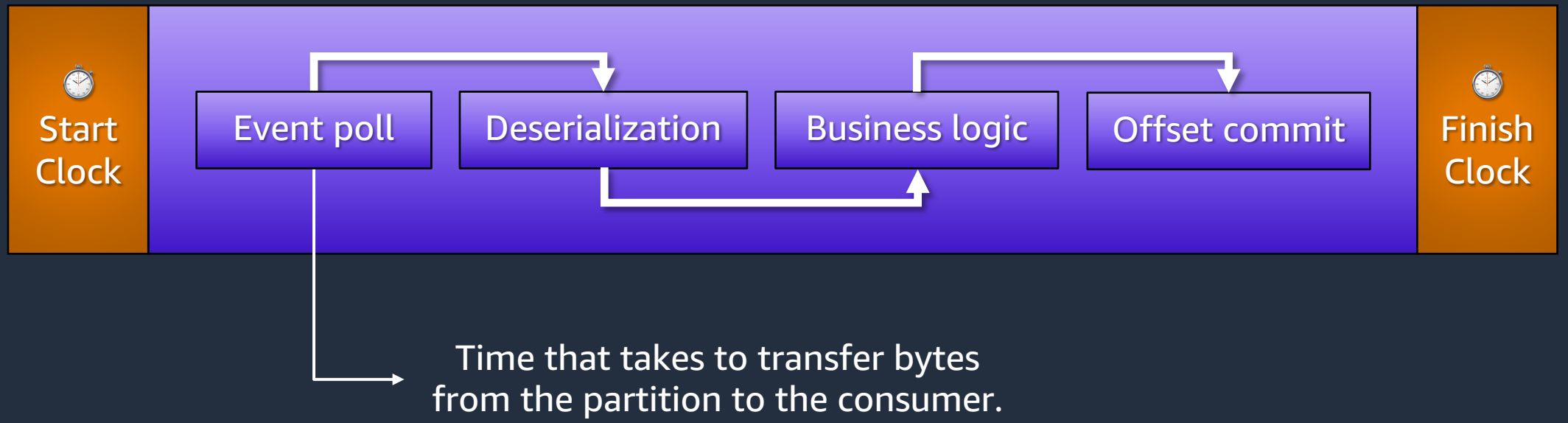
Throughput
in events/sec

Disk I/O and
Network I/O

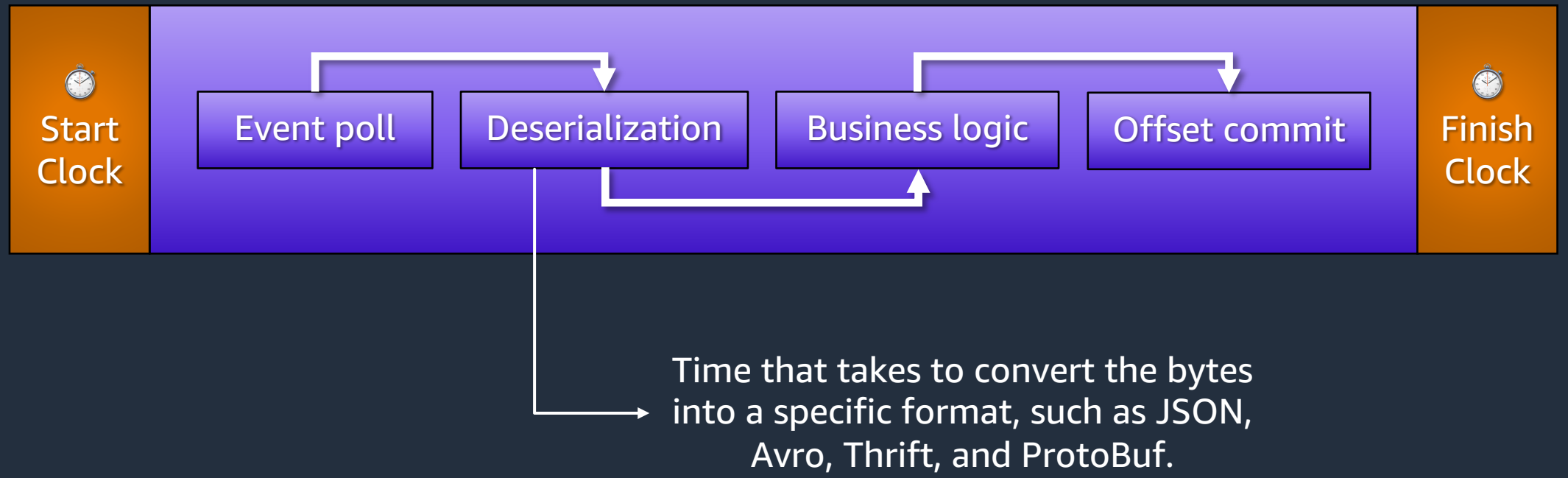
Throughput
in events/sec

Event processing flow

Event processing flow



Event processing flow



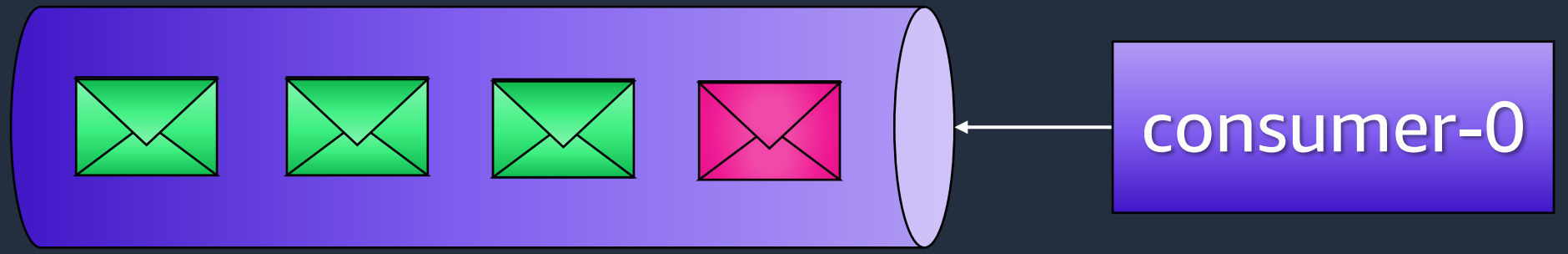
Event processing flow

Processing that takes place before writing the data off into a specific target, or simply dispose the data.

Event processing flow

Time that takes to send the offsets from the processed records back to the broker holding the partition.

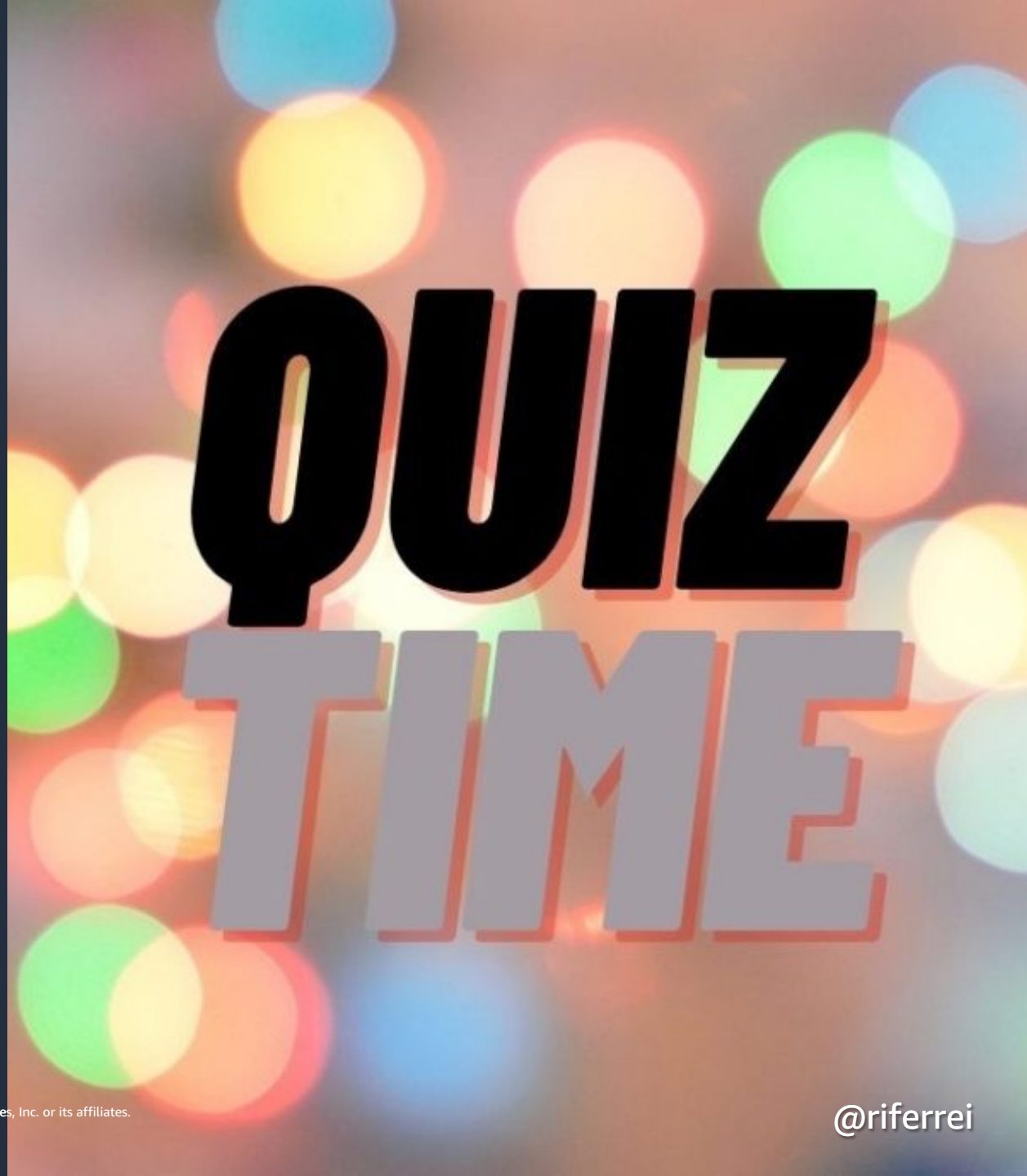
Handling poison pills



Quiz time:

- One topic with **1** partition.
- **100** messages waiting processing.
- **5** last messages are poison pills.
- Consumer with **default** config.

How many messages committed? 🤔



@riferrei

Result:

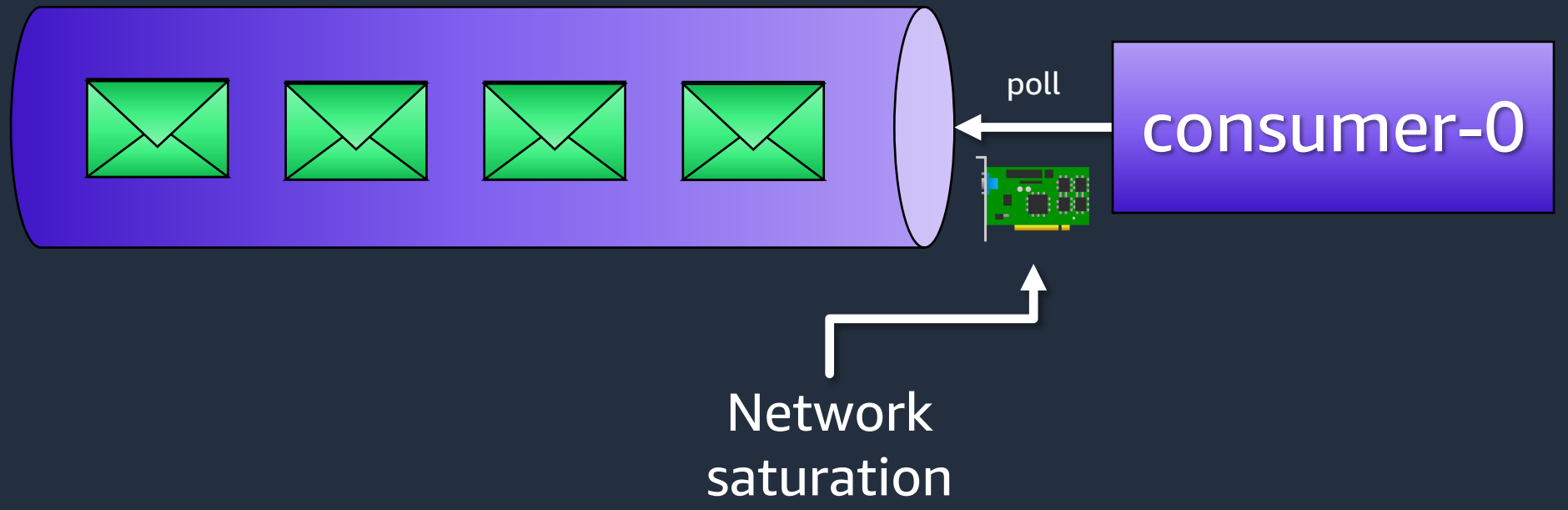
No one knows. By default Kafka auto commit the messages in a interval. If the poison pill **raises an exception**, the the polling will break and stop the committing. If not, then congrats: you got yourself a **integrity issue** problem.

For use cases with integrity semantics, you have to **manually commit** the offsets in a per-message basis.

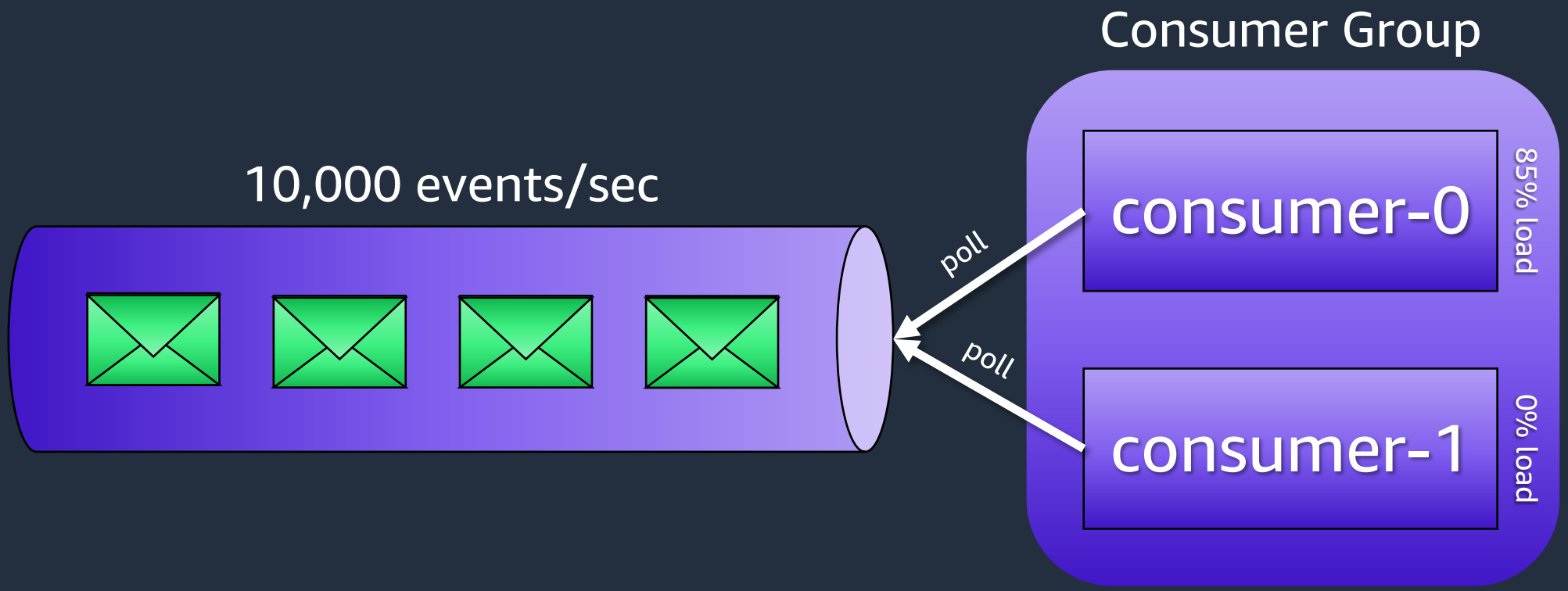
The event poll loop

```
while(true) {
    ConsumerRecords records = consumer.poll(Duration.ofMillis(10000));
    processRetrievedRecords(records);
}
```

Consumer throughput versus saturation



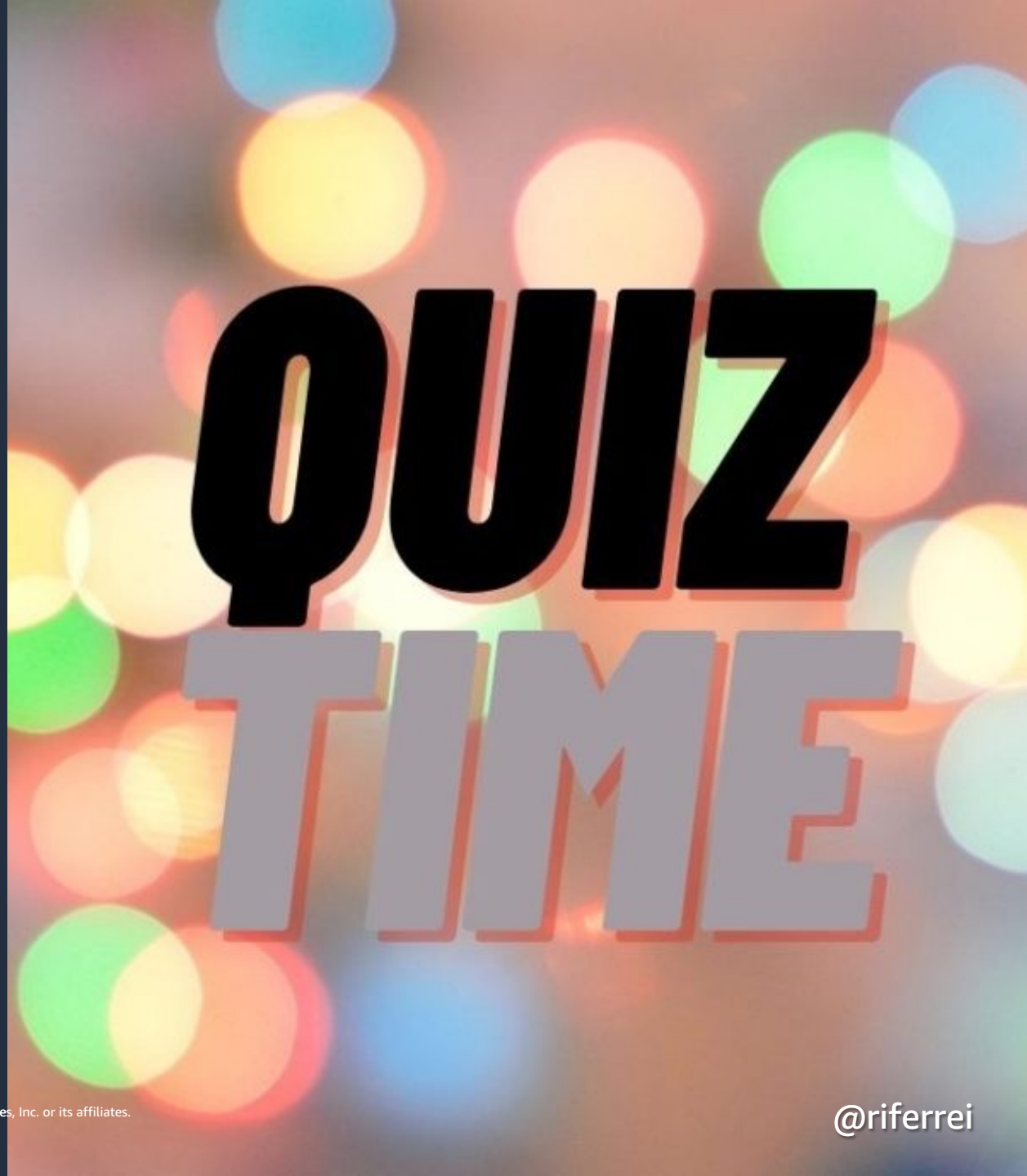
Scaling things up with multiple consumers



Quiz time:

- One topic with **1** partition.
- **2** consumers in different nodes.
- Consumers in **different** groups.

Can throughput be doubled? 🤔



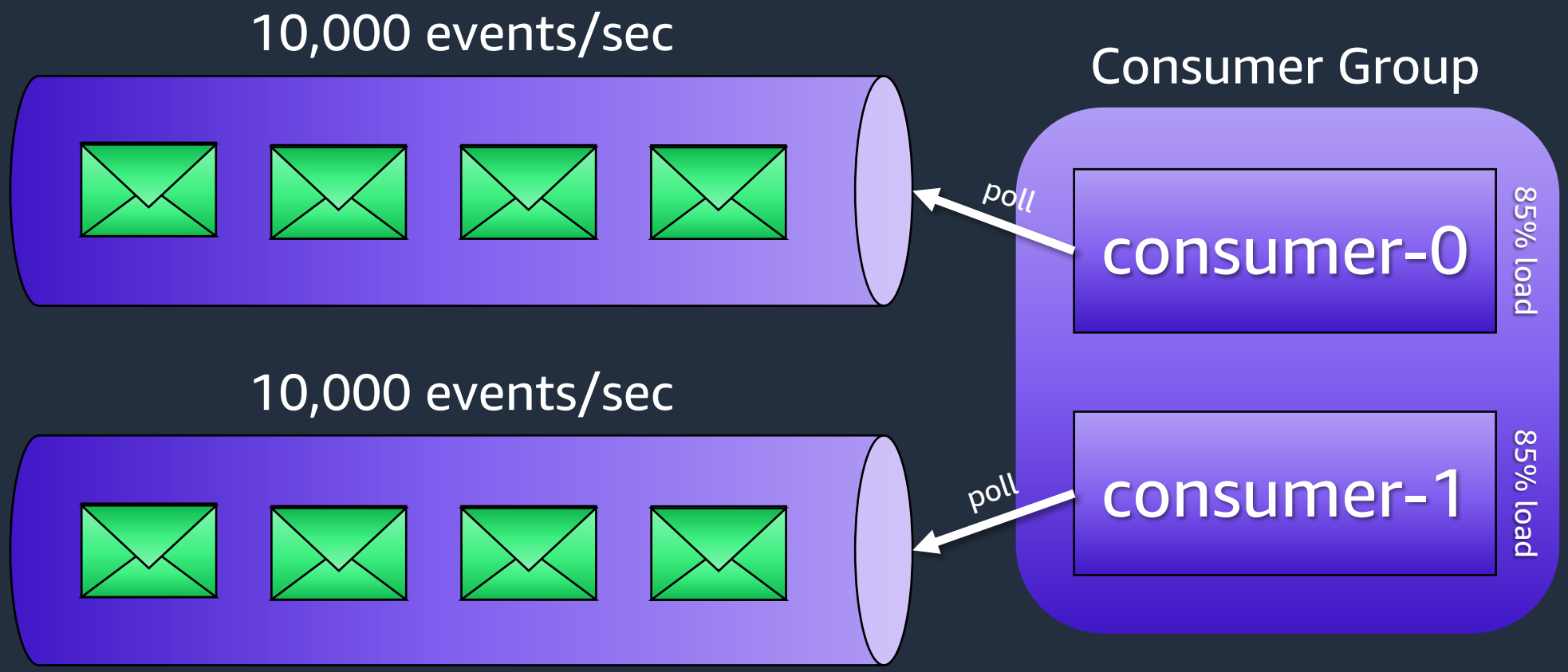
@riferrei

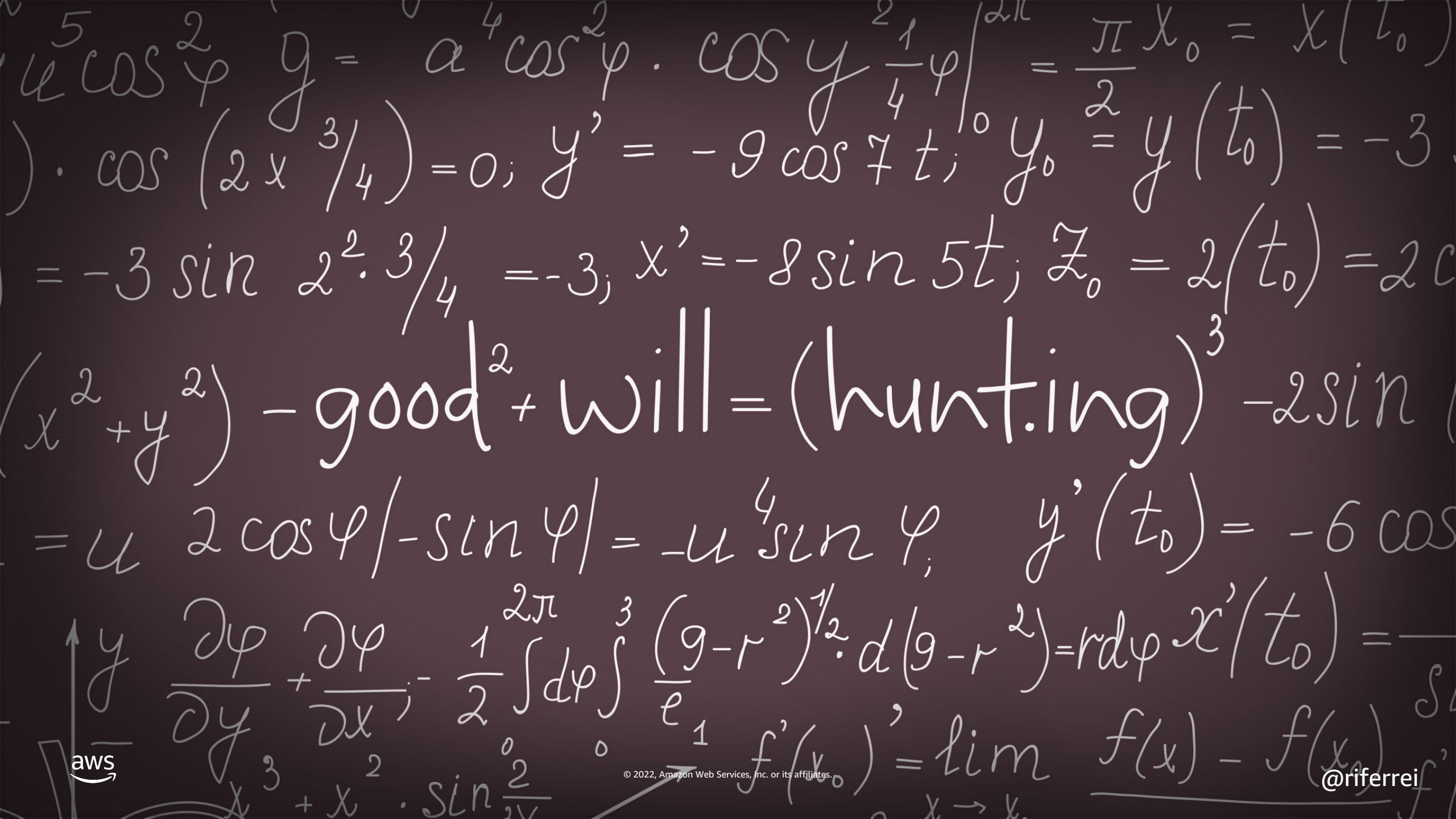
Result:

No. Putting different consumers in different groups will tell Kafka to **broadcast** the same message to the groups, leading to a serious problem of **data consistency**.

If you need true parallelism, then you need to have dedicated partitions to each group.

Scaling things up with multiple partitions





@riferrei

Good "good enough" method:

NUM_PARTITIONS = MAX(T/W, T/R)

T = Target throughput in events/second

W = Write throughput in events/second

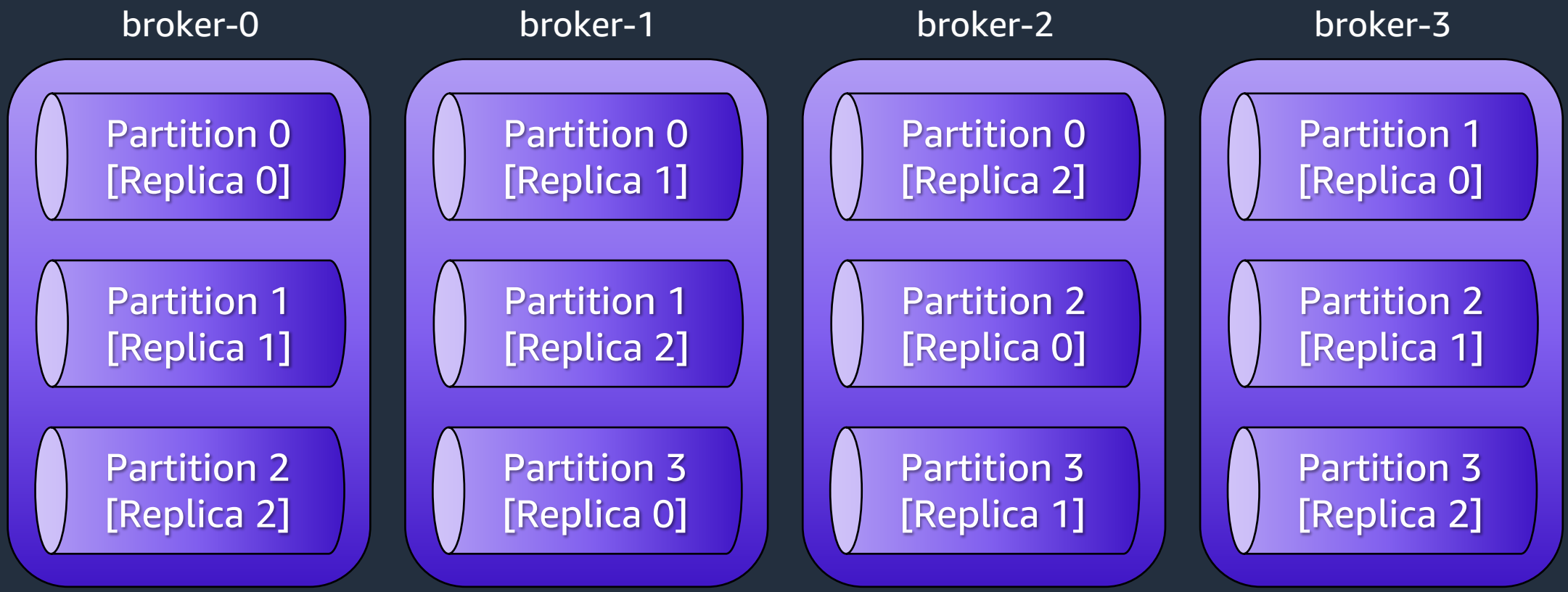
R = Read throughput in events/second

Demo time!

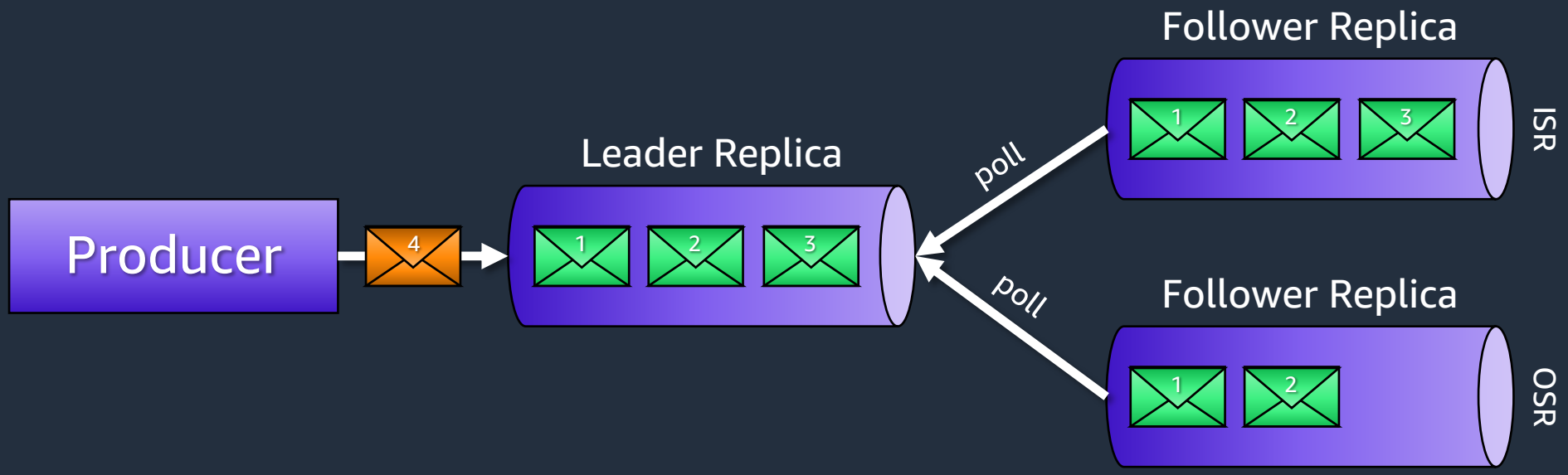
Keep mind extra processing time

Partitions as the unit-of-durability

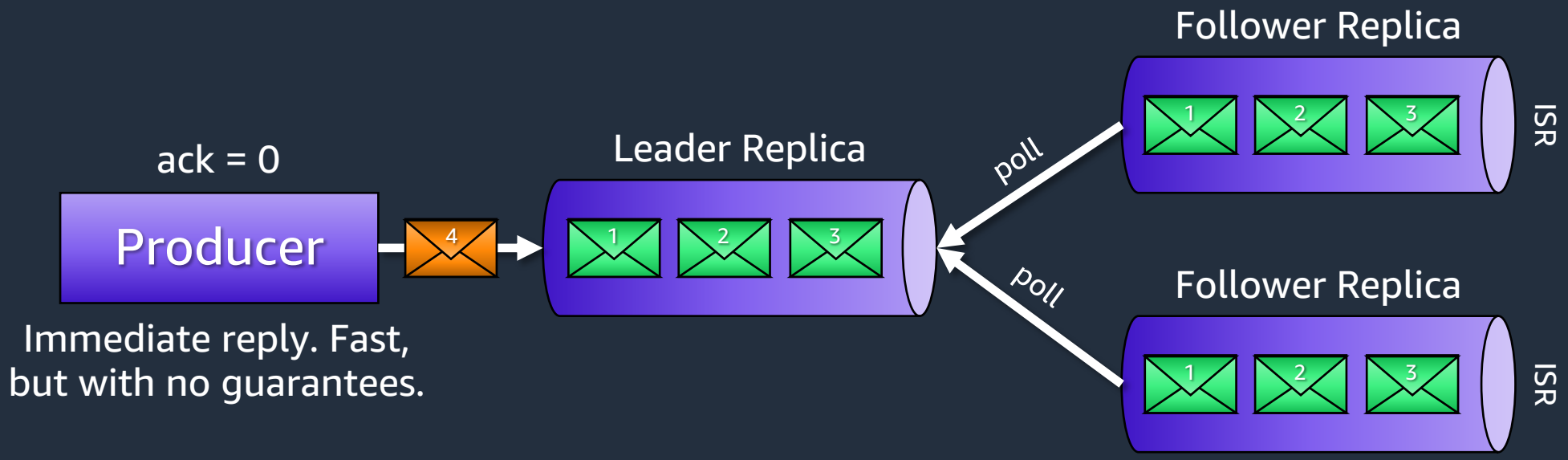
Handling failures with replication



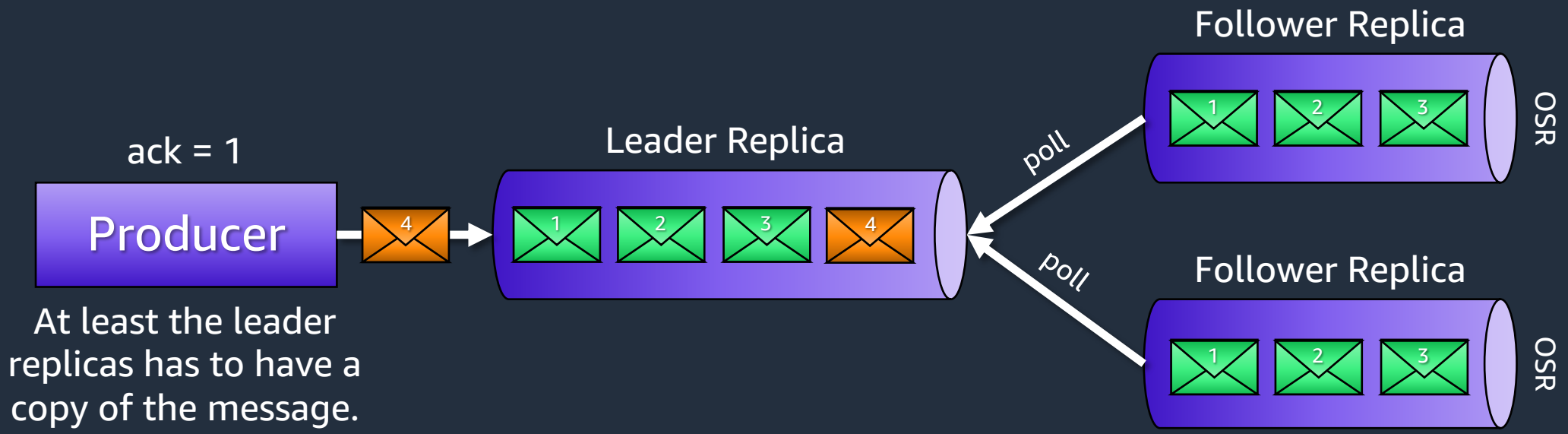
Types of replicas: leaders and followers



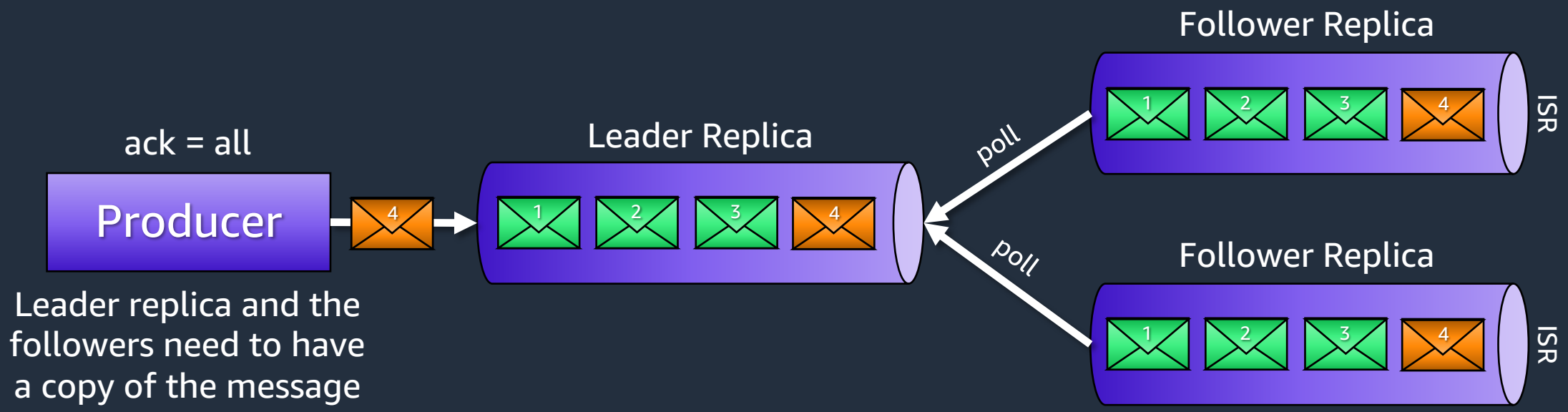
Data consistency with replicas



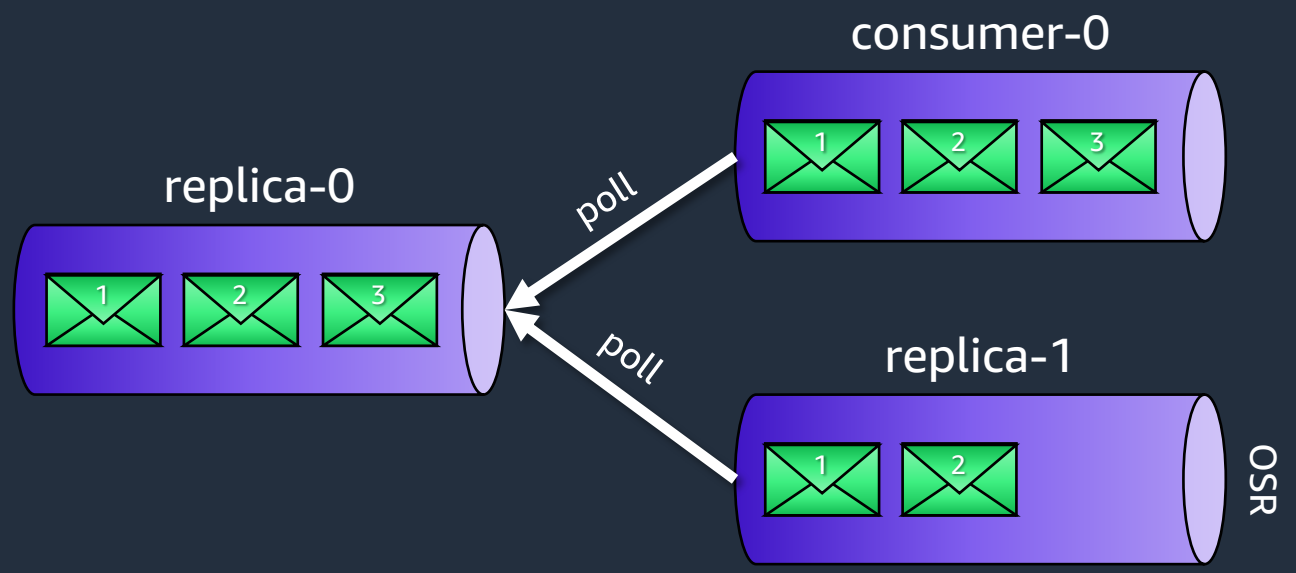
Data consistency with replicas



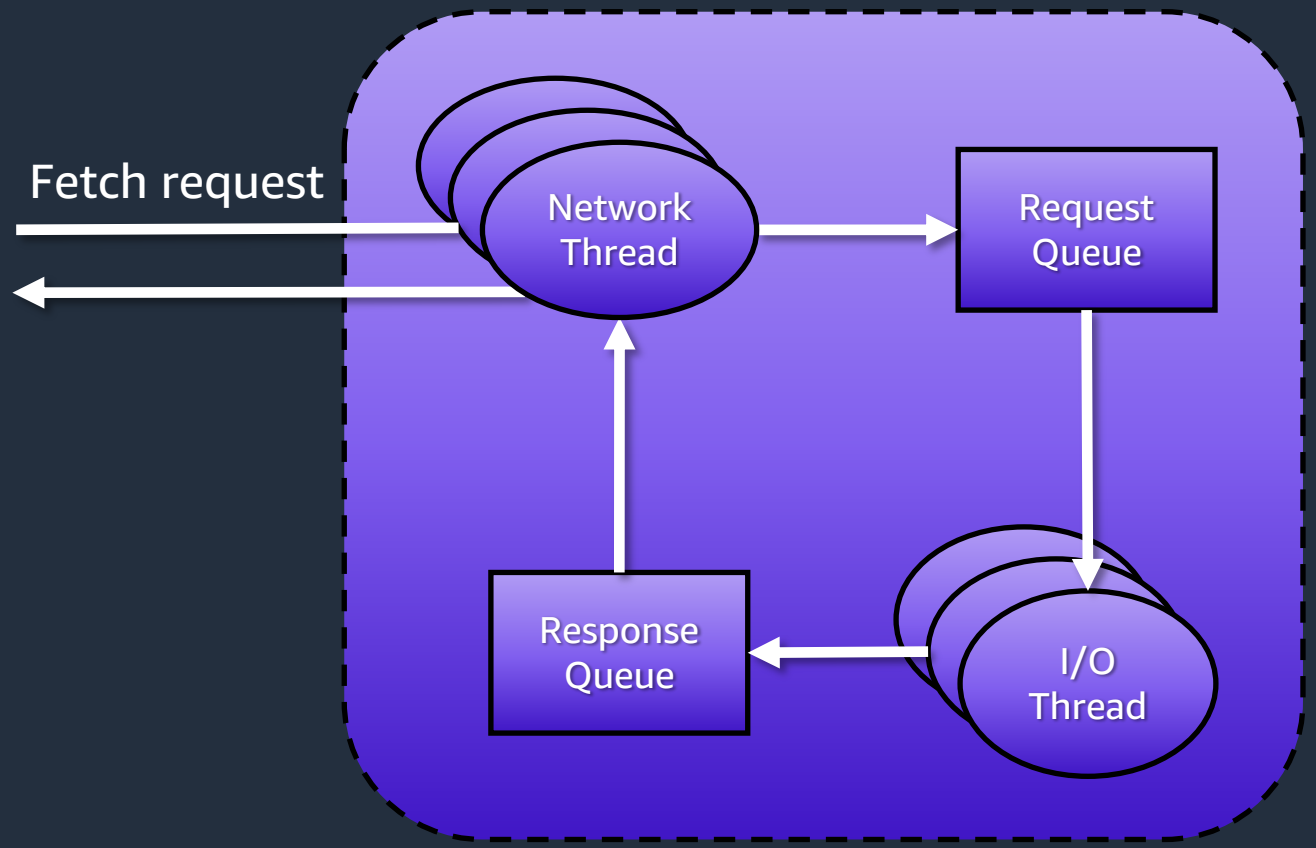
Data consistency with replicas



Why aren't all replicas ISRs?



Replication processing model



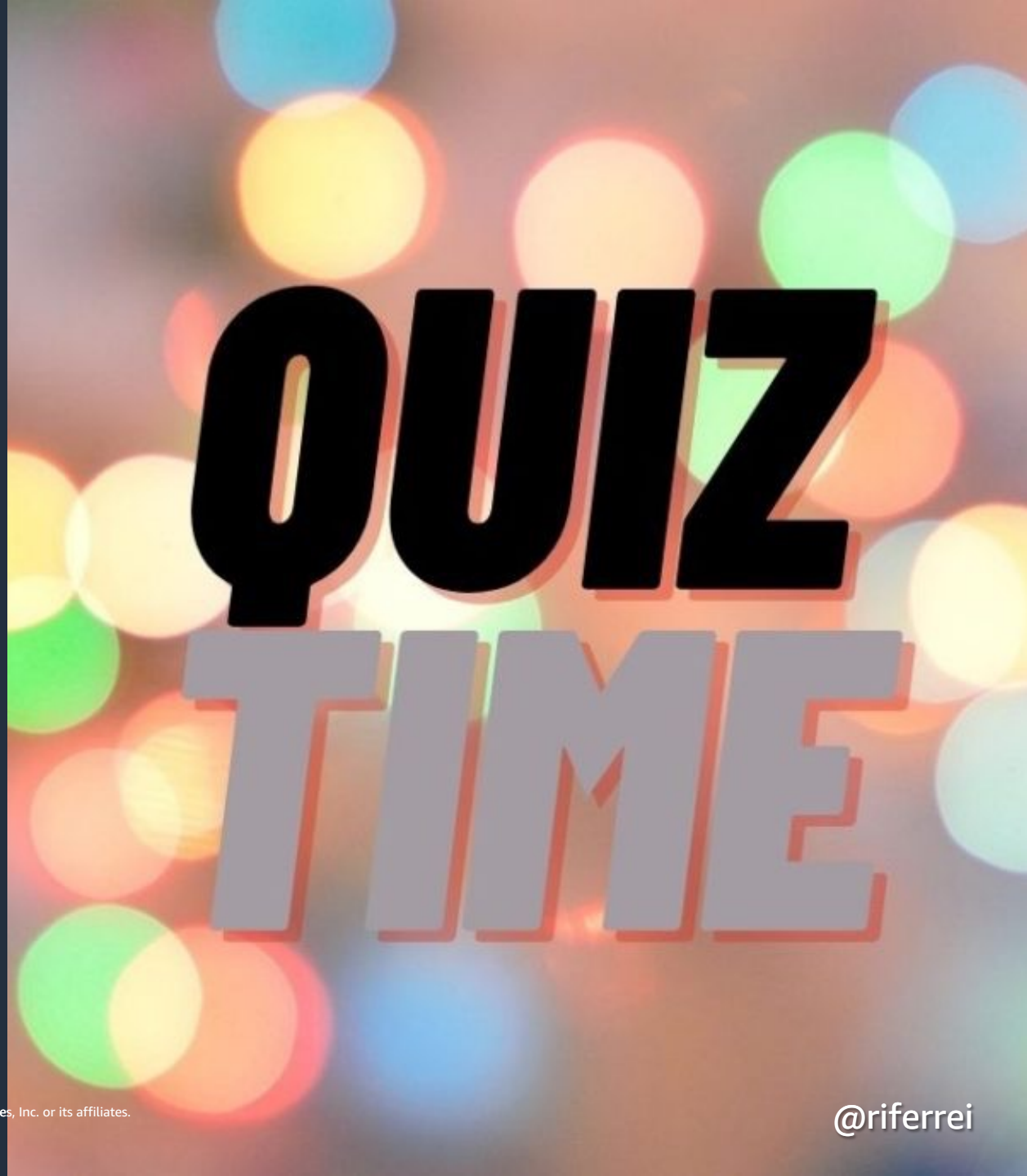
Network thread pool size: **5**

I/O thread pool size: **8**

Quiz time:

- Broker with **default** configuration.
- You notify **lag** with replication.
- Network pool increased to **10**.
- I/O thread pool increased to **16**.

Will this solve the problem? 🤔



@riferrei

Result:

No. Usually the replication saturation occurs in a network interface level, before any need of more CPU power.

In fact, if you increase these thread pools without having enough CPU idle, there is a huge change of you making the problem even worst with lots of context switching.

This session is also available as a blog post:

In the land of the sizing, the one-partition Kafka topic is king

Understand what are partitions in Apache Kafka, and the impact they have on your workloads.

Ricardo Ferreira @ AWS
Published Aug 22, 2022

#apache-kafka #stream-processing #data-streams #big-data

Every technology has that key concept that people struggle to understand. With databases, the struggle usually happens when you have to decide which join clause to use for fetching data from multiple tables. Which one is faster? What about consistency? How will concurrency look like if I pick this one versus the other? It is often a hard choice. Containers are another great example. Implementing persistence with containers is troublesome because each workload has its own set of requirements, and there is no silver bullet. For example, you may have a workload that requires each container to store 20% of the dataset locally, whereas the other 80% should go straight to a shared filesystem that is mounted on every container instance. But, you should not reuse this design for a microservices workload, for example.

https://www.buildon.aws/posts/in-the-land-of-the-sizing-the-one-partition-kafka-topic-is-king/01-what-are-partitions/

@riferrei

Thank you!

Ricardo Ferreira
@riferrei