DEVNEXUS™

The Right Number of Partitions for a Kafka Topic

Ricardo Ferreira

Senior Developer Advocate
Amazon Web Services

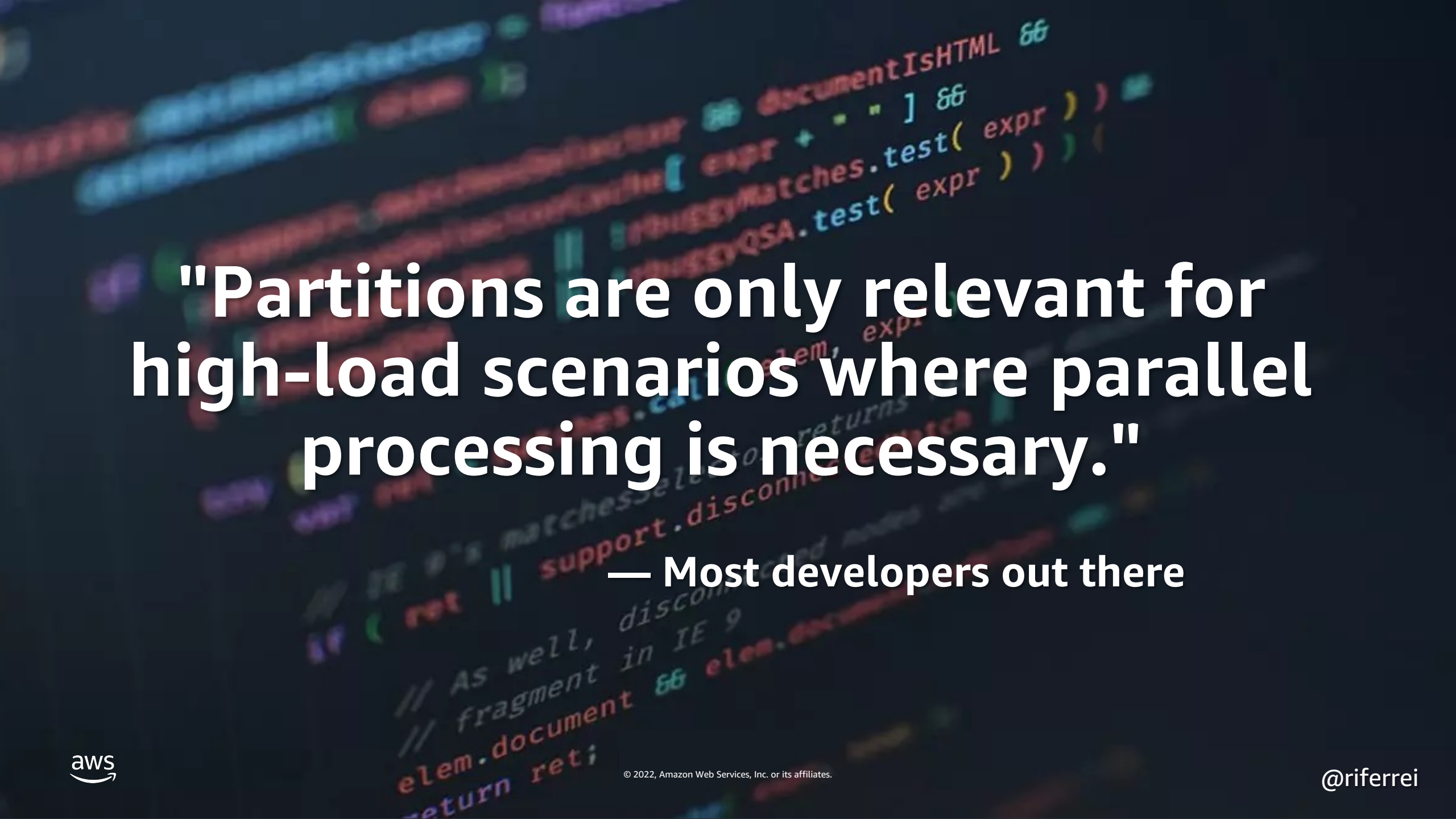
@riferrei

The impact of partitions

"My stream processor code in Flink worked well with 4 partitions, but with higher partitions the result is wrong."

"My K8S cluster is acting crazy since I increased the partitions of a Kafka topic. Consumer pods are dying."

"I tried to enable tiered-storage in my cluster, but it didn't work with topics that are compacted."



"Partitions are only relevant for high-load scenarios where parallel processing is necessary."

— Most developers out there

@riferrei

In reality, partitions in Kafka are:

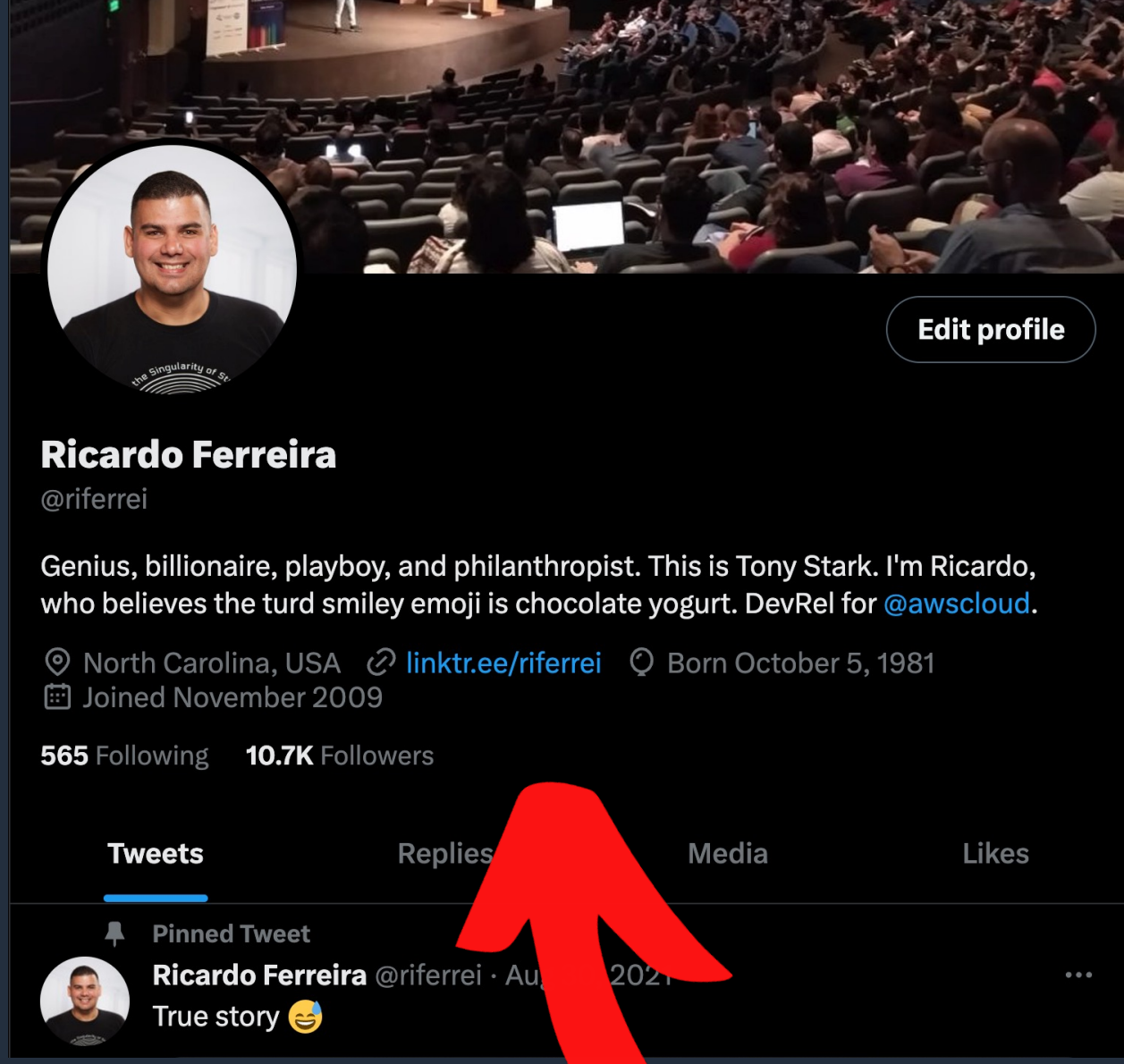
1. Unit of Parallelism

2. Unit of Storage

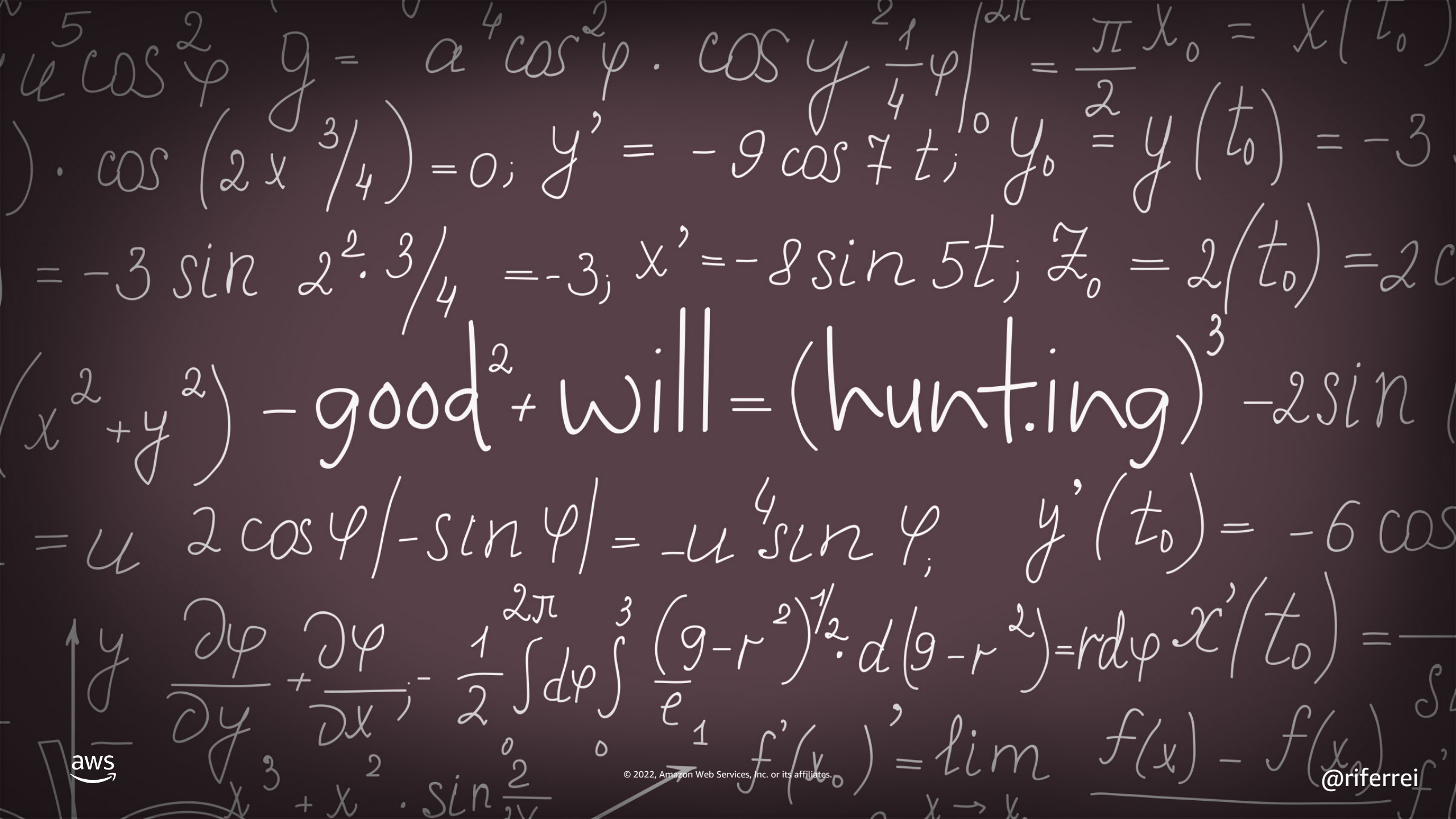
3. Unit of Durability

👋 Hi, I'm Ricardo Ferreira

- Developer Advocate at AWS.
- Fanatic Marvel fan. My favorite characters are Daredevil, Venon, Deadpool, and the Punisher.
- Yes, they are all anti-heroes.



@riferrei



@riferrei

Good "good enough" method:

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

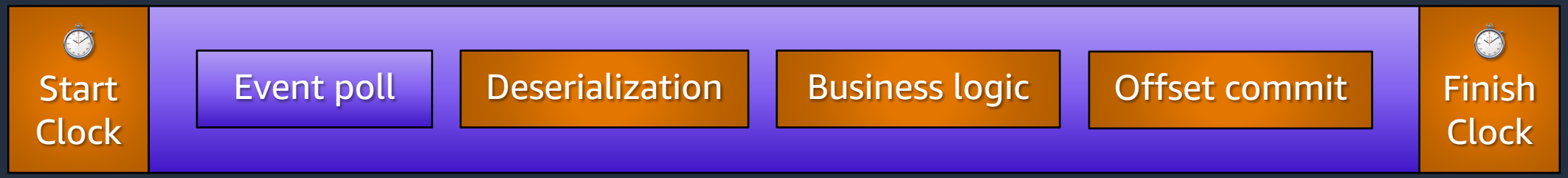
T = Target throughput in events/second

W = Write throughput in events/second

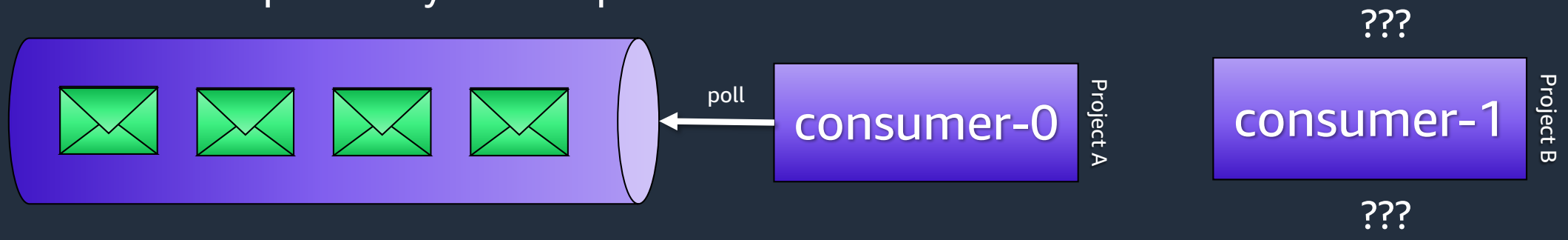
R = Read throughput in events/second

For the read throughput, keep in mind:

1. Complete processing time of each event



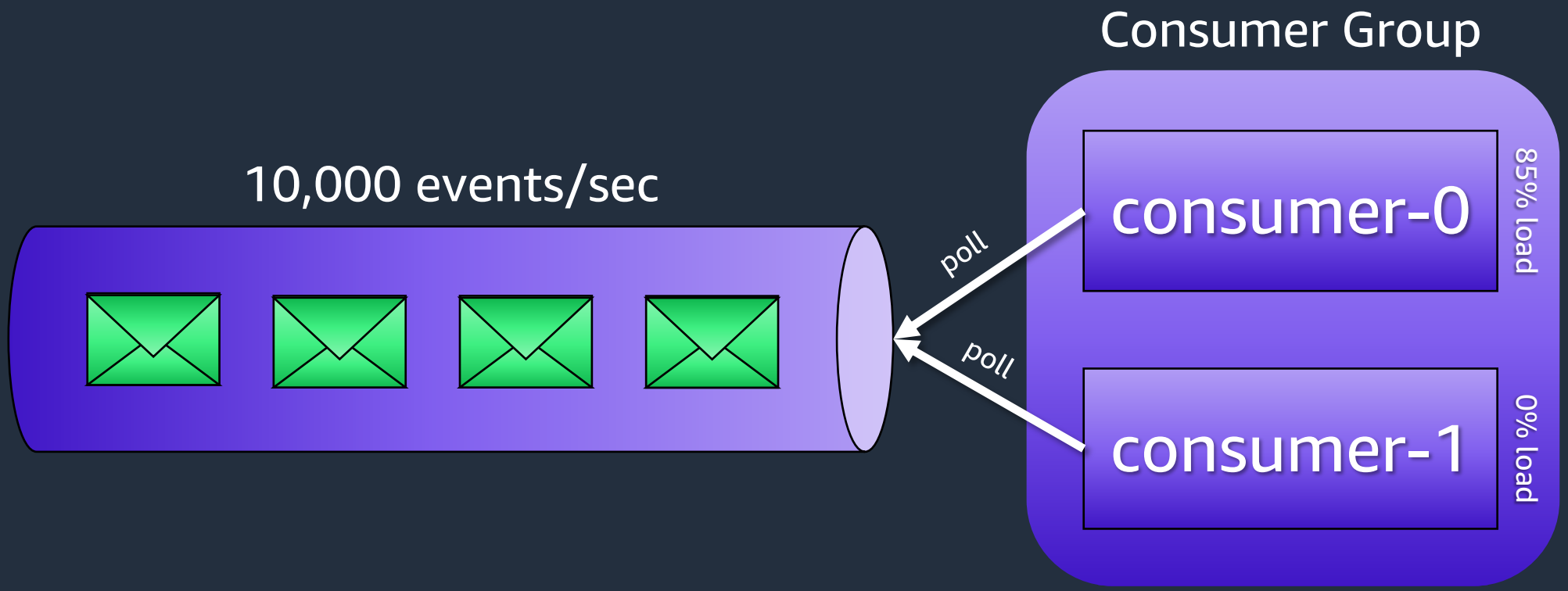
2. Check the possibility of multiple consumers



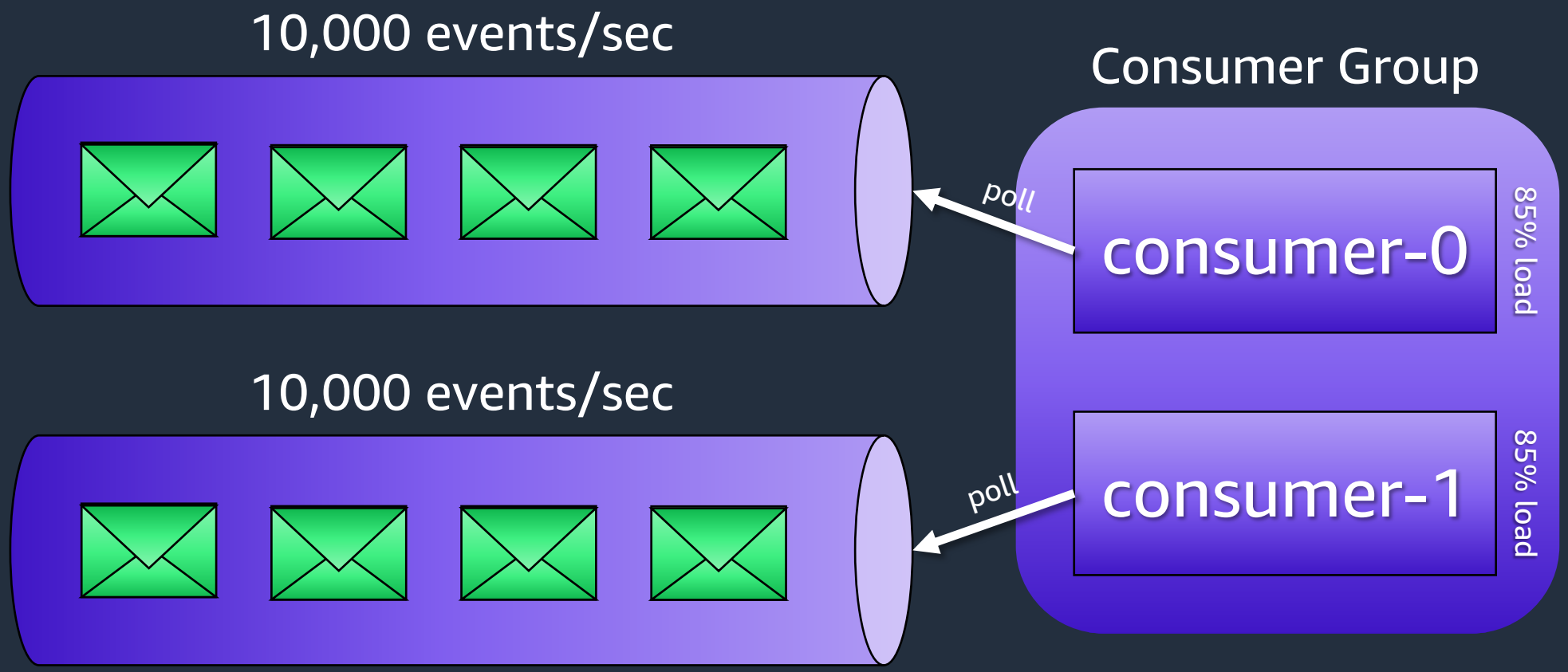
@riferrei

Partitions as unit of parallelism

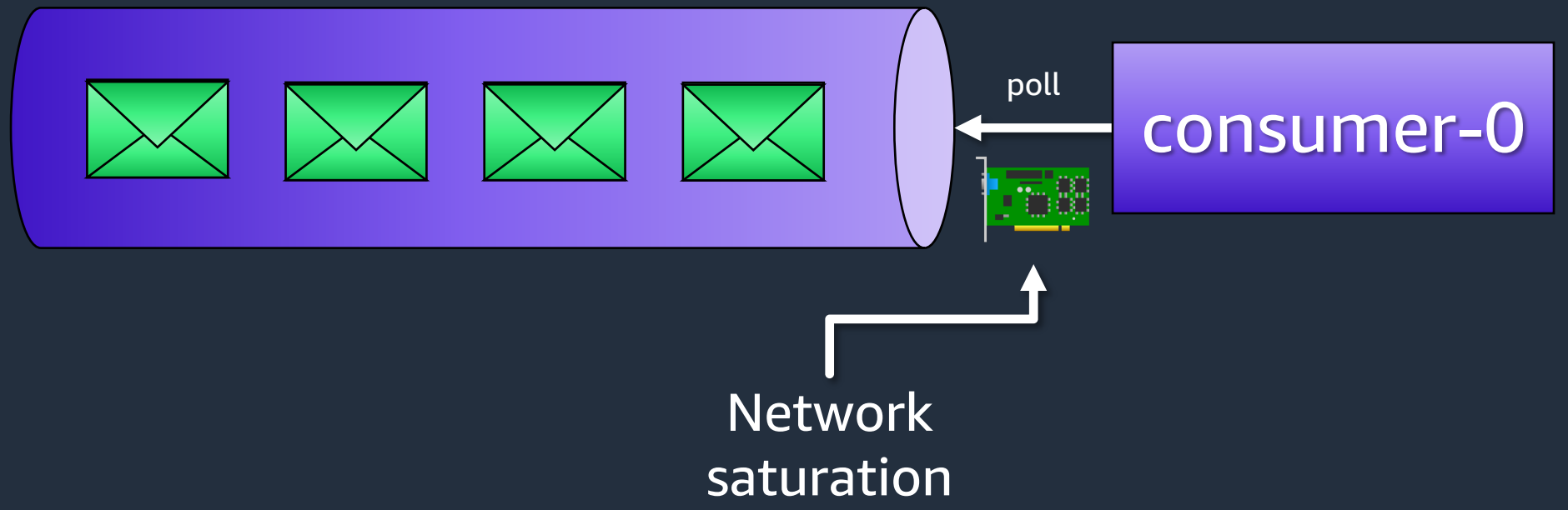
Scaling things up with multiple consumers



Scaling things up with multiple partitions



#1 Culprit: network saturation

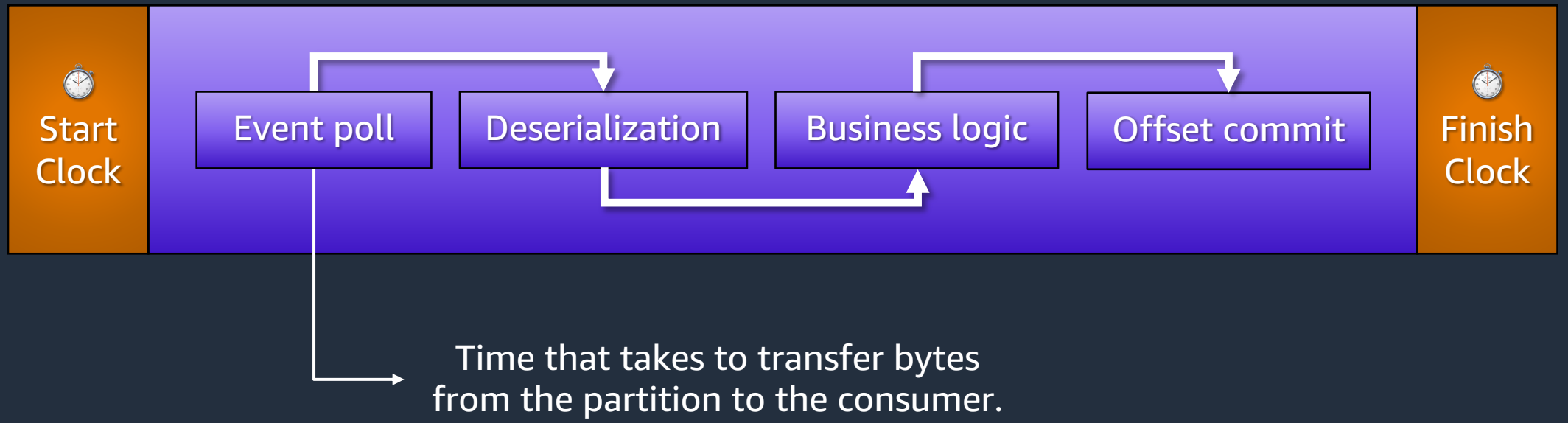


#2 Culprit: event processing

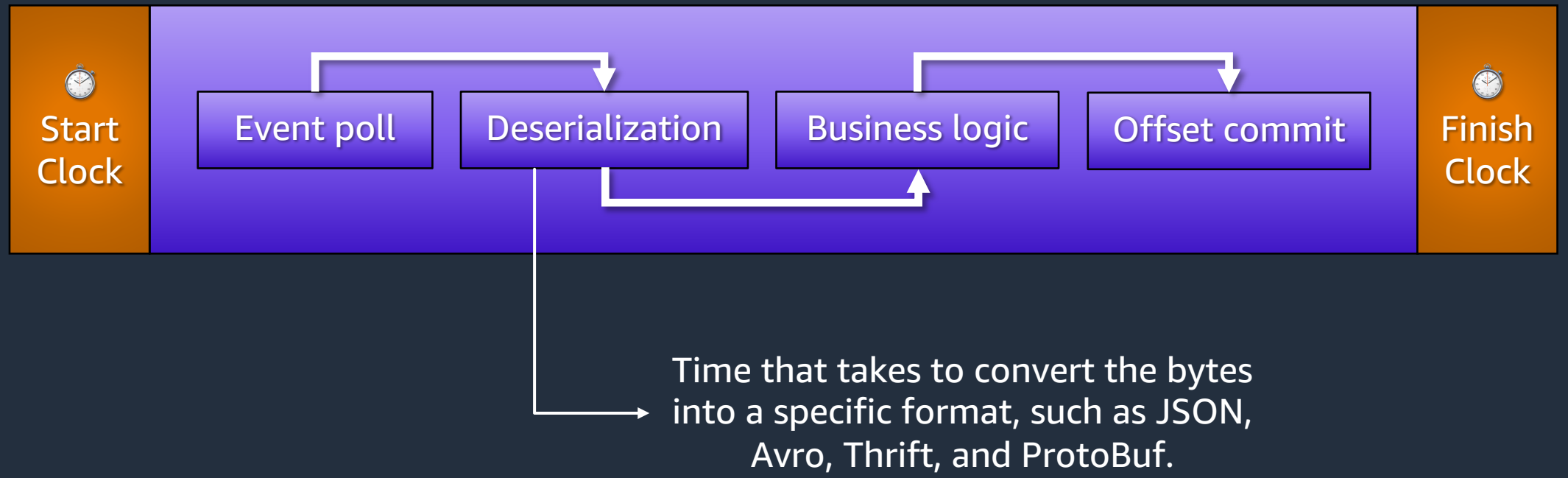
```
while(true) {
    ConsumerRecords records = consumer.poll(Duration.ofMillis(10000));
    processRecords(records);
}
```

Event processing flow

Event processing flow



Event processing flow



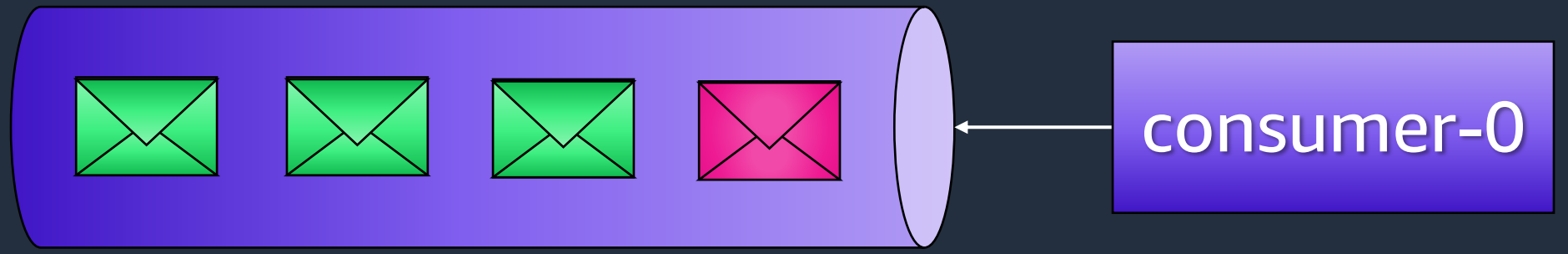
Event processing flow

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

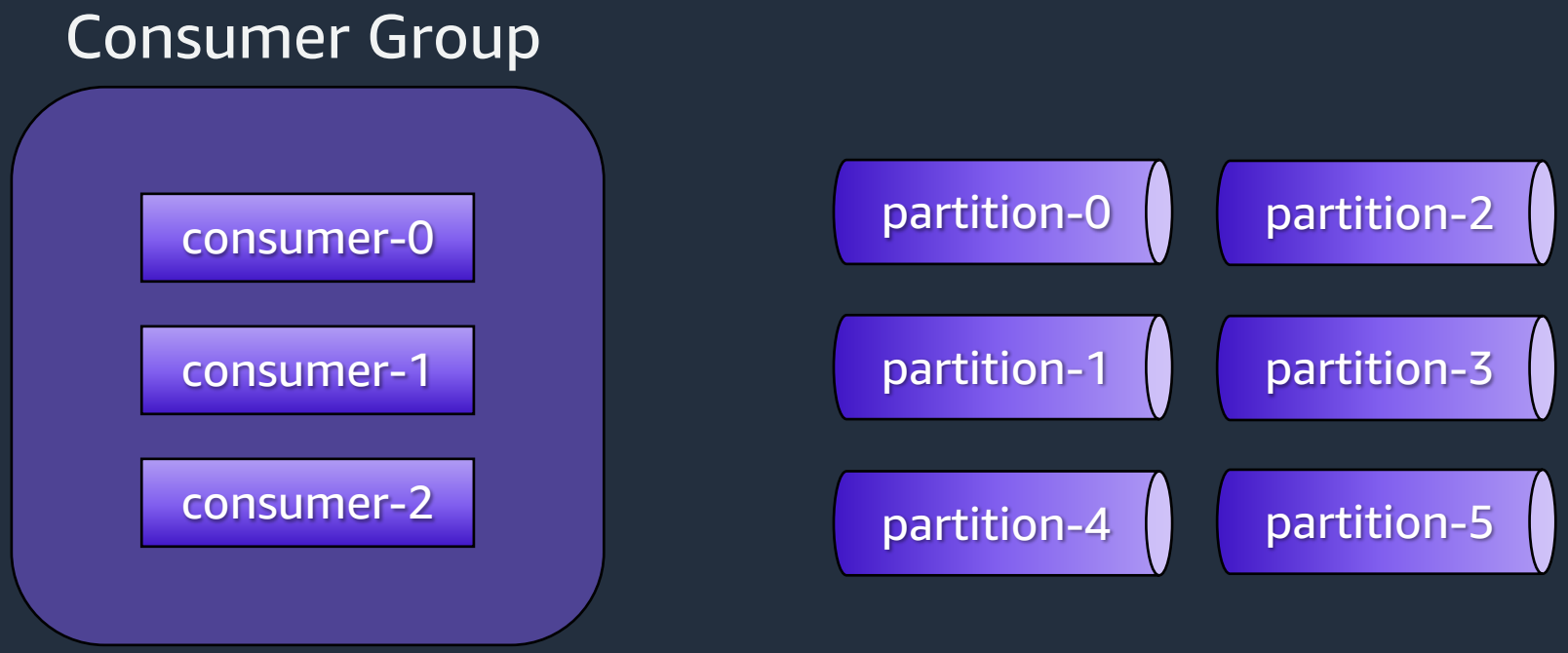
Event processing flow

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

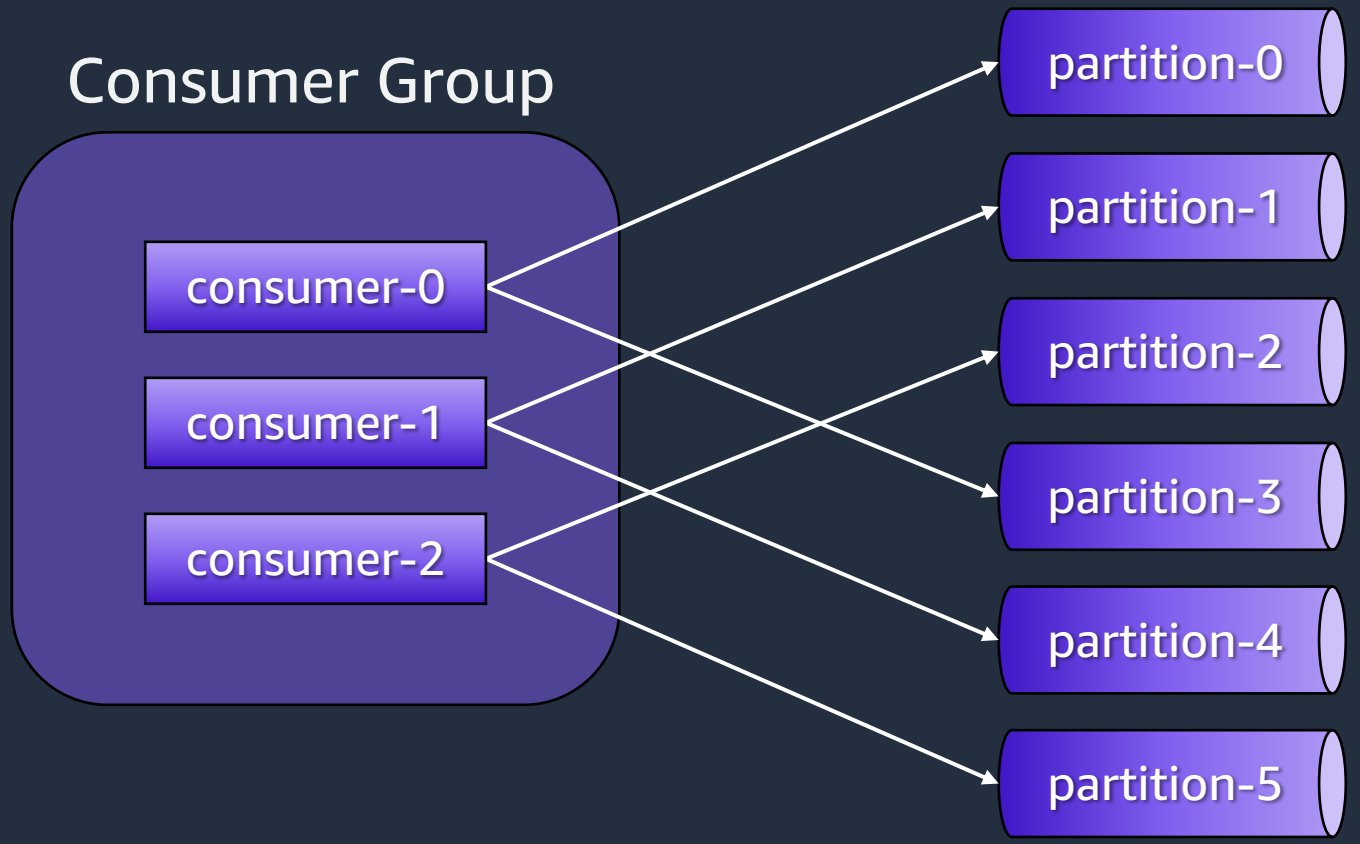
#3 Culprit: records with poison pills



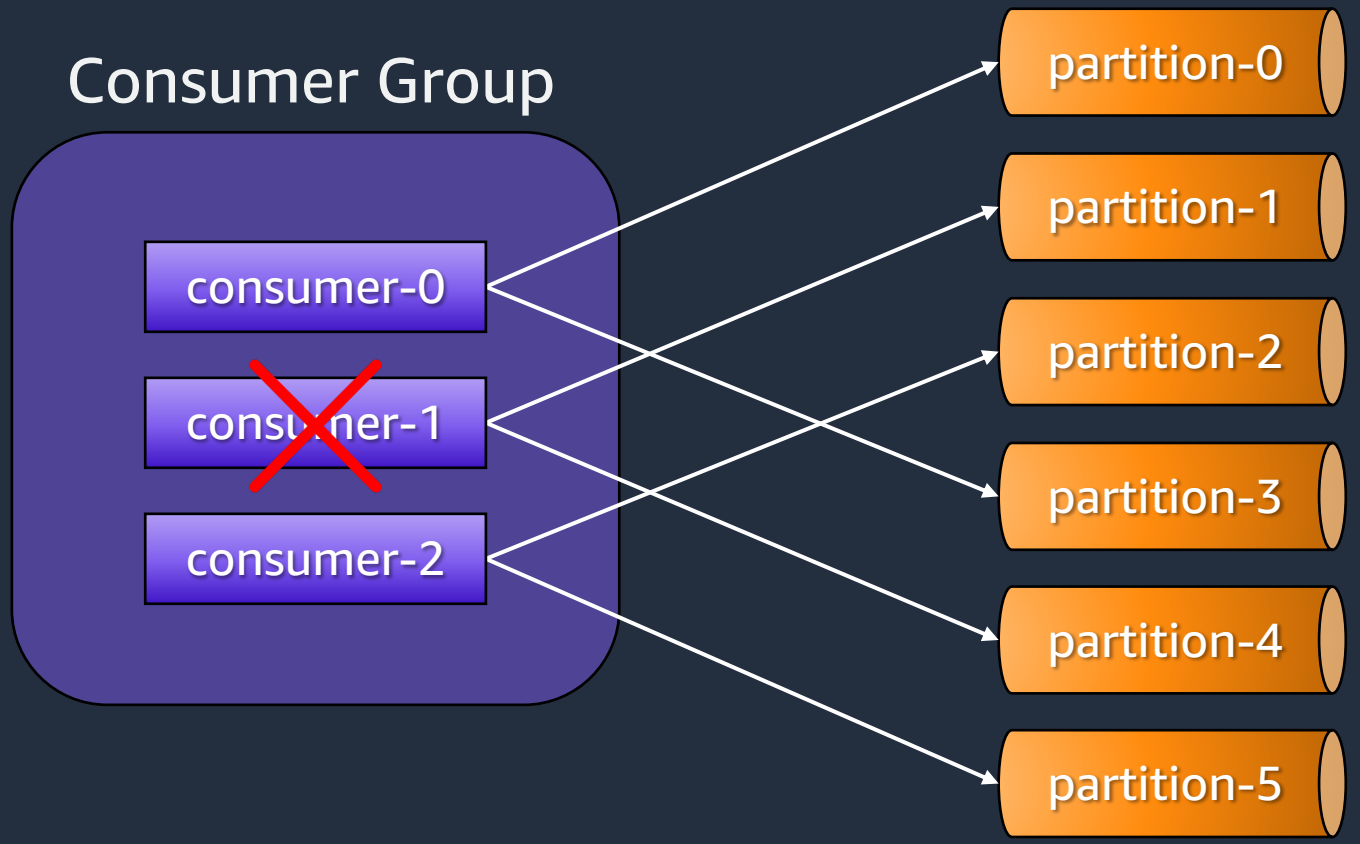
#4 Culprit: partition assignment



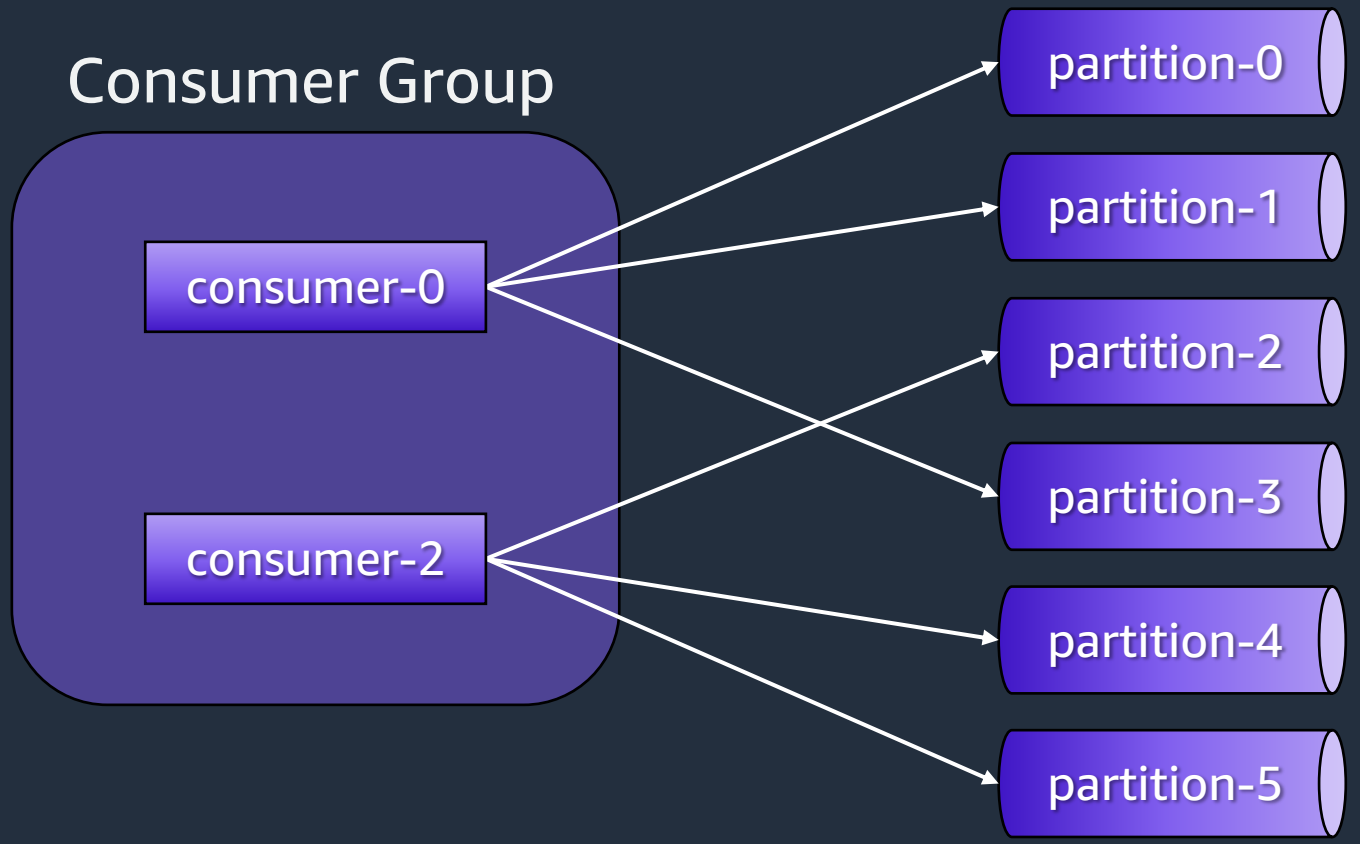
How assignment works: RangeAssignor



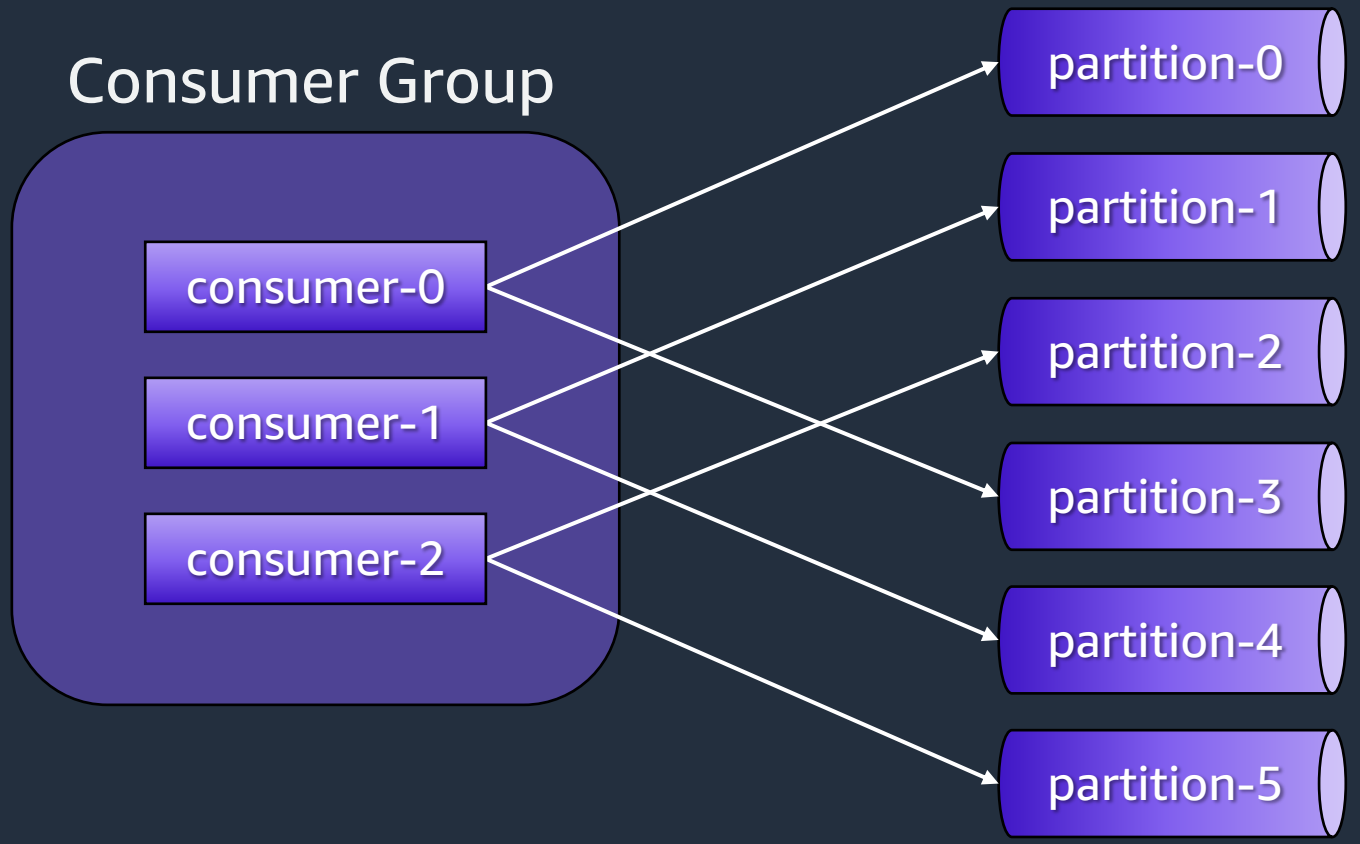
How assignment works: RangeAssignor



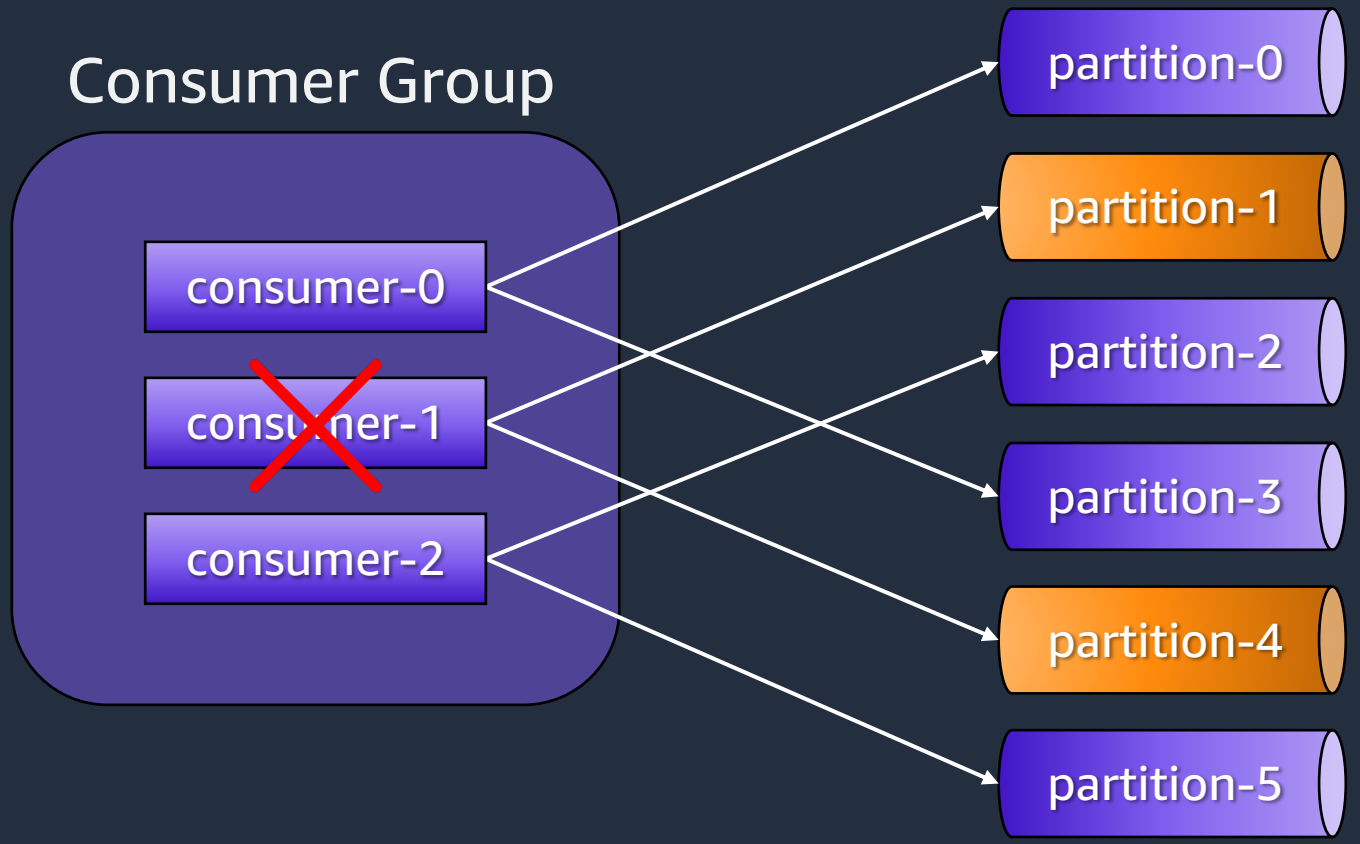
How assignment works: RangeAssignor



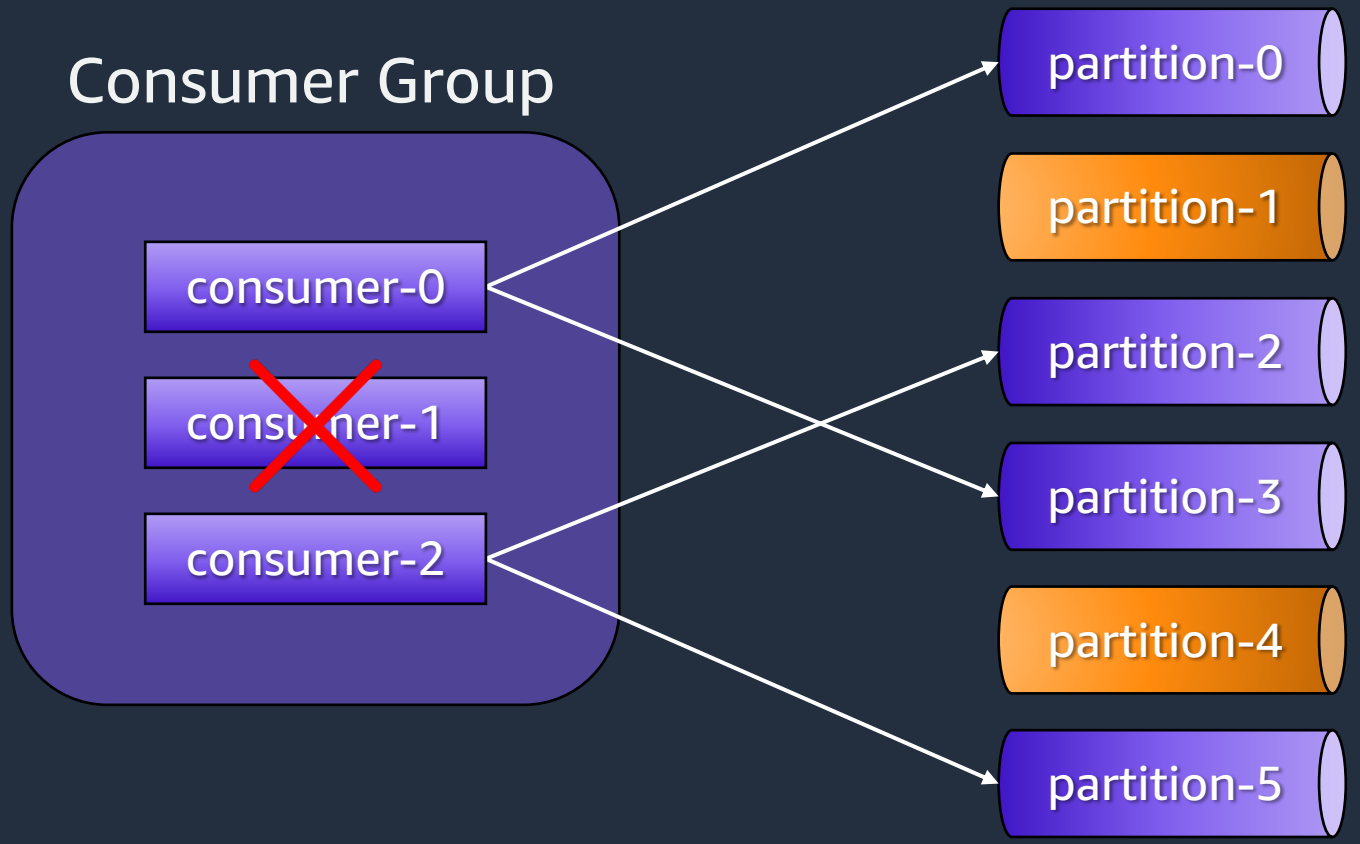
How assignment works: CooperativeStickyAssignor



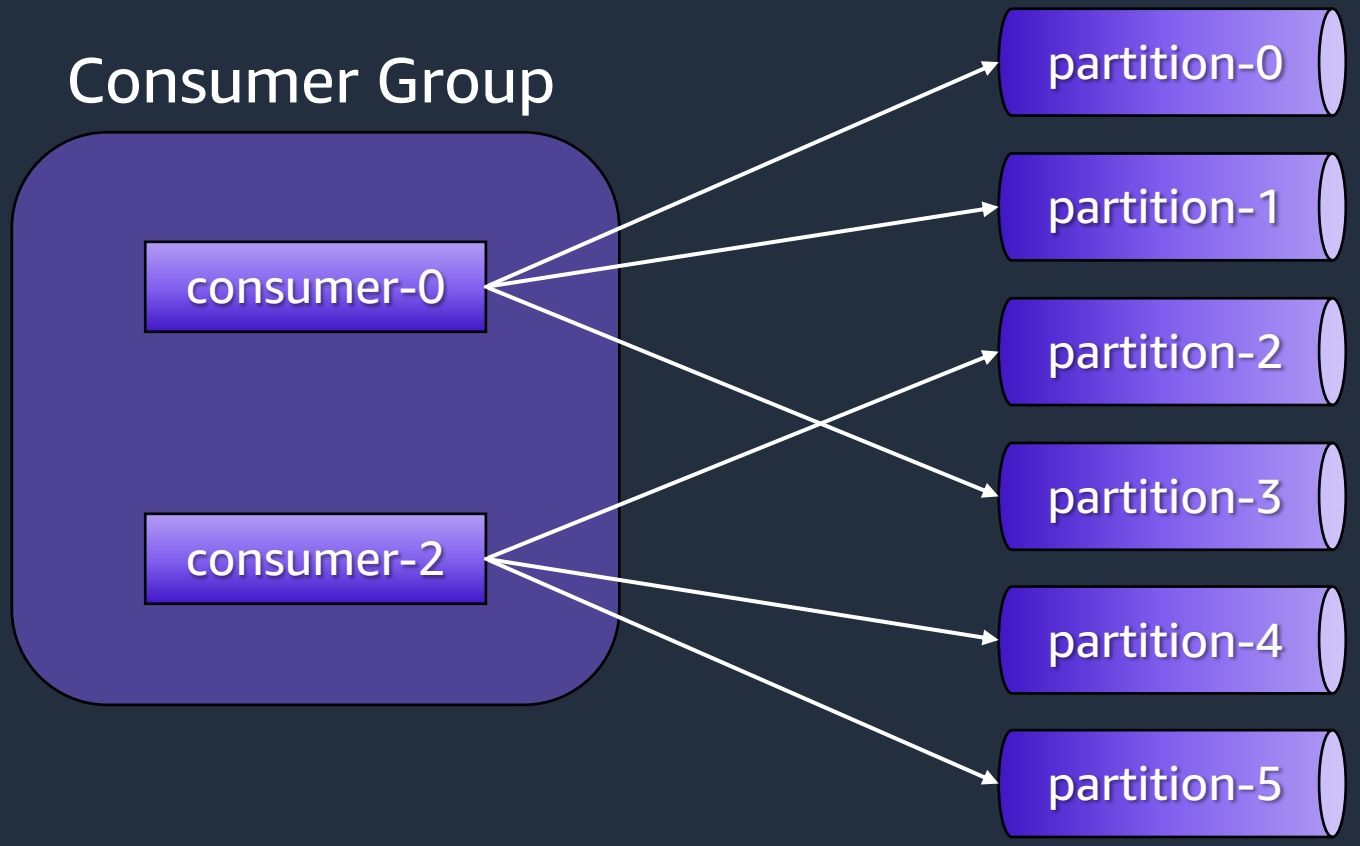
How assignment works: CooperativeStickyAssignor



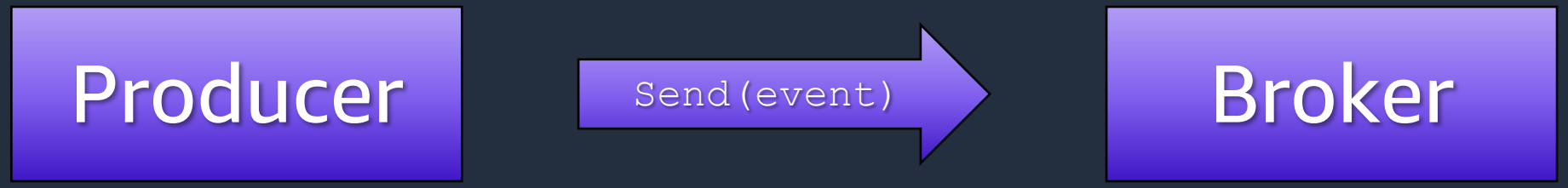
How assignment works: CooperativeStickyAssignor



How assignment works: CooperativeStickyAssignor



#5 Culprit: data partitioning



"When no keys are specified, data is spread over partitions using round-robin. Otherwise, records with the same key are written in the same partition."

— Most developers out there

KIP-794: Strictly Uniform Sticky Partitioner

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

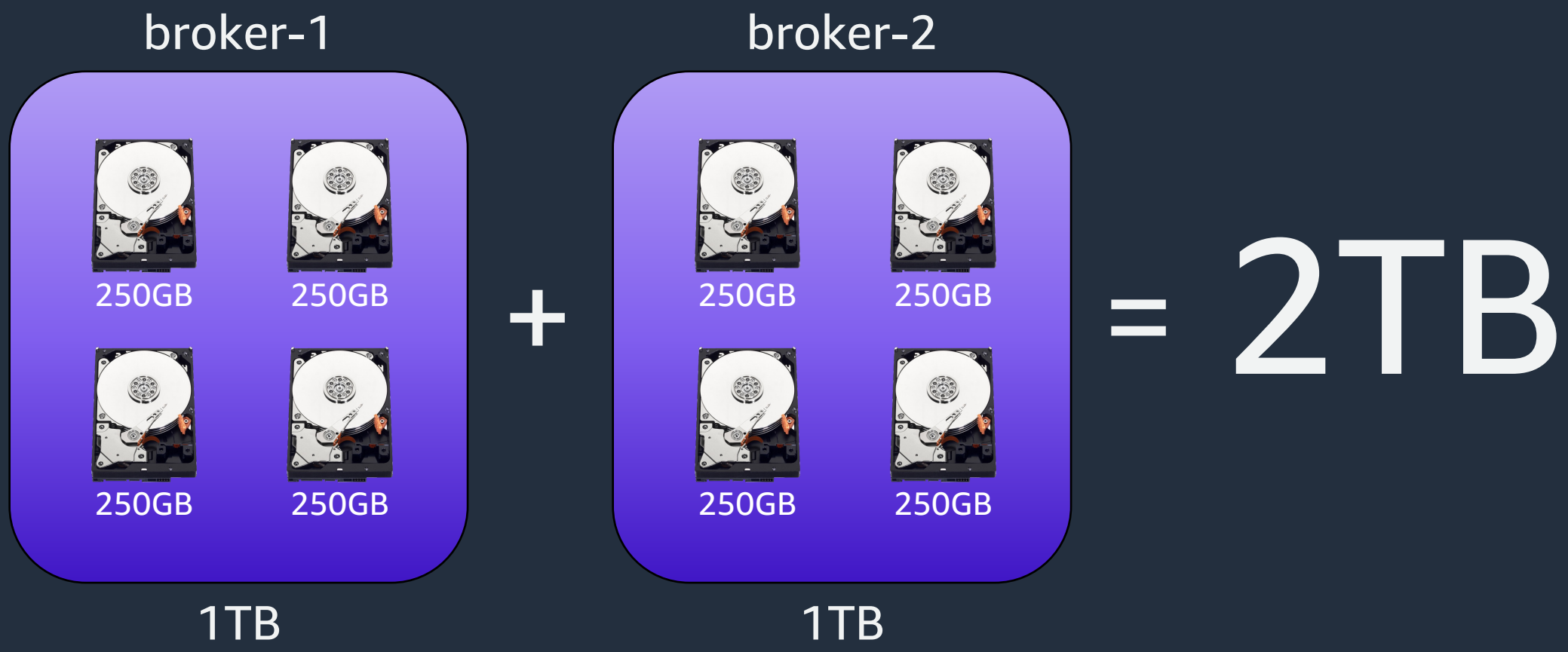
Otherwise it will use the assigned key to figure the partition.

4. Broker load

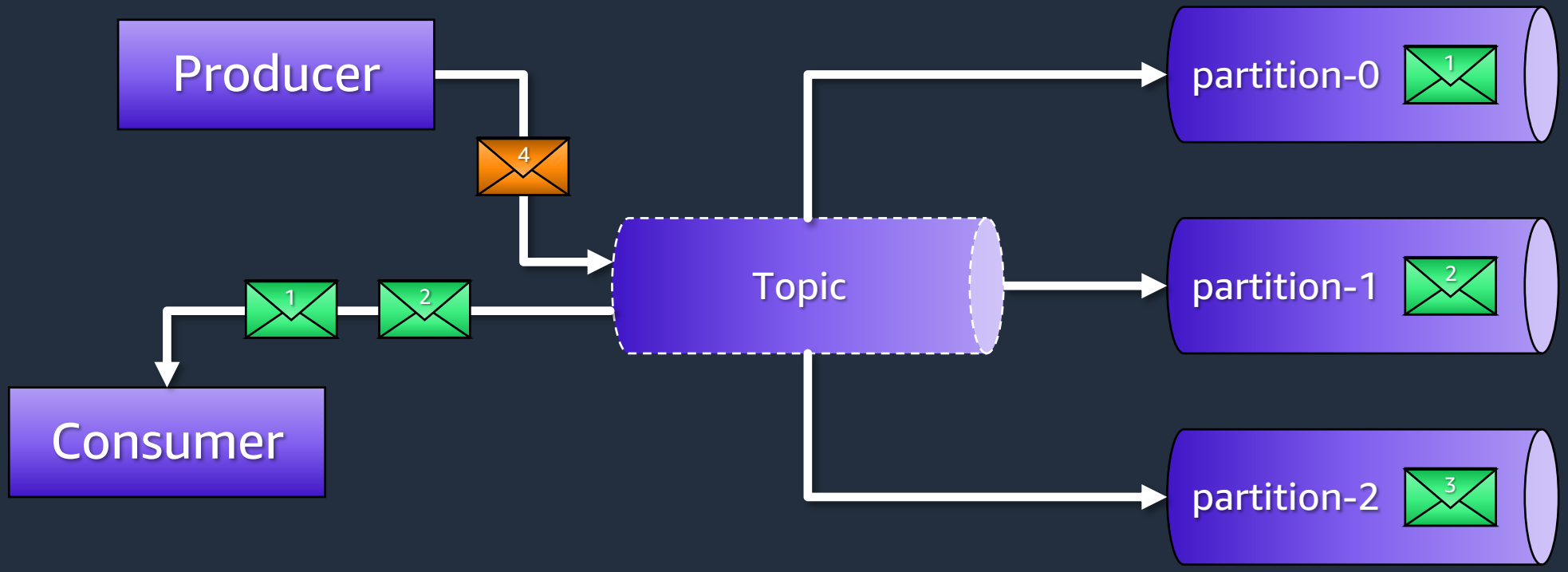
Last fallback is based on factors like broker load, amount of data produced to each partition, etc.

Partitions as unit of storage

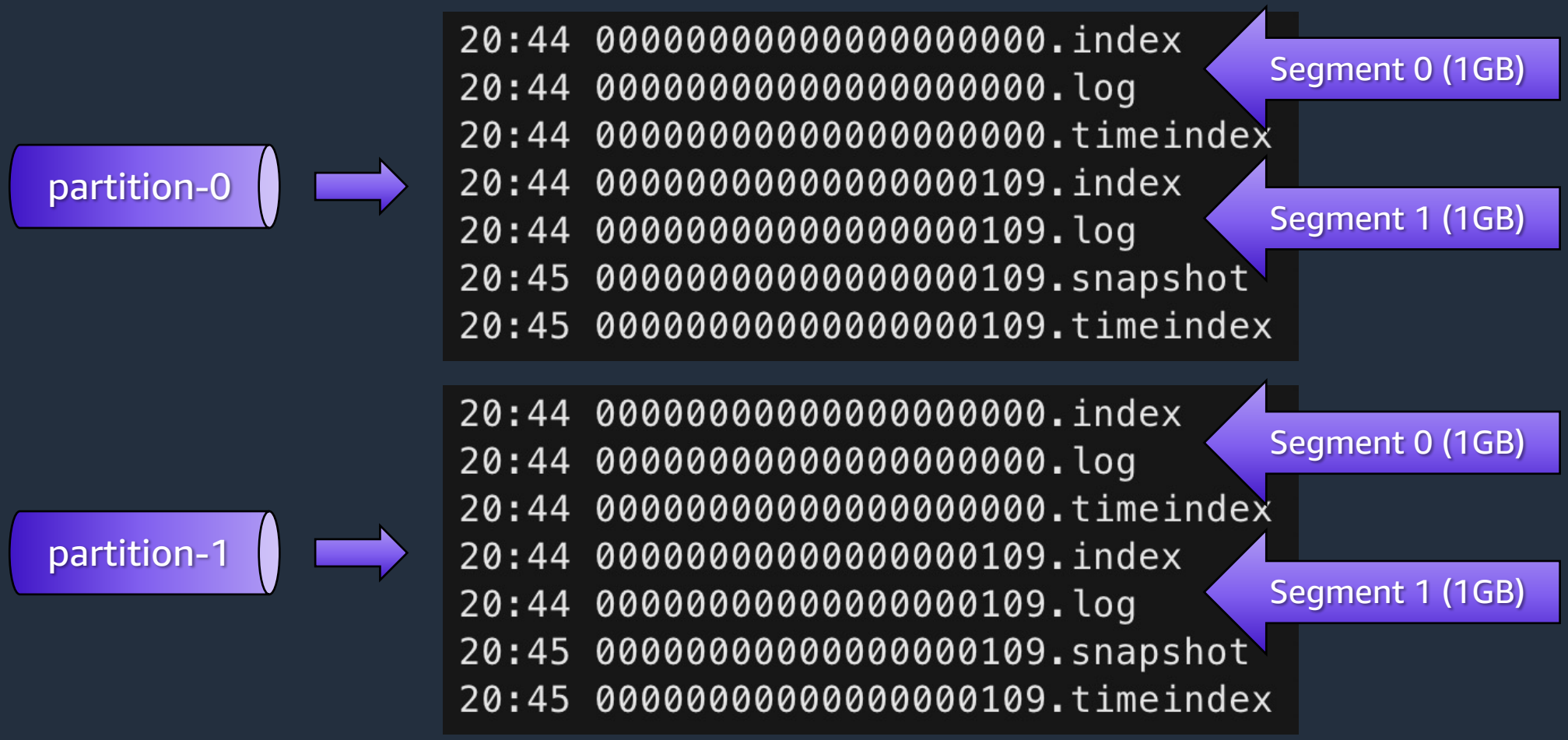
Kafka's shared-nothing storage system



Topics are just high-level abstractions

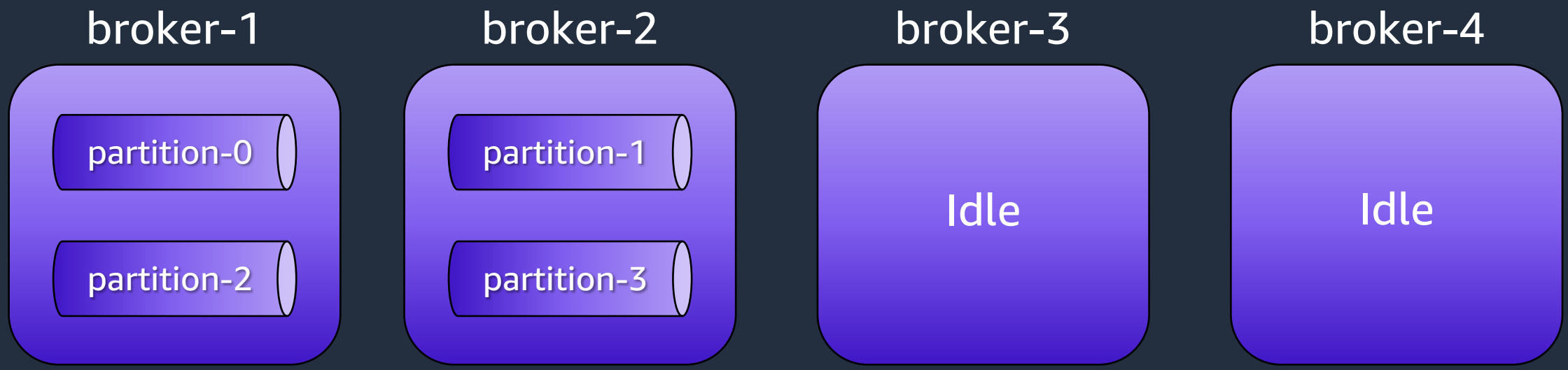


Writes and reads goes through partitions



Even distribution during topic creation

Infrastructure changes lead to unbalanced data

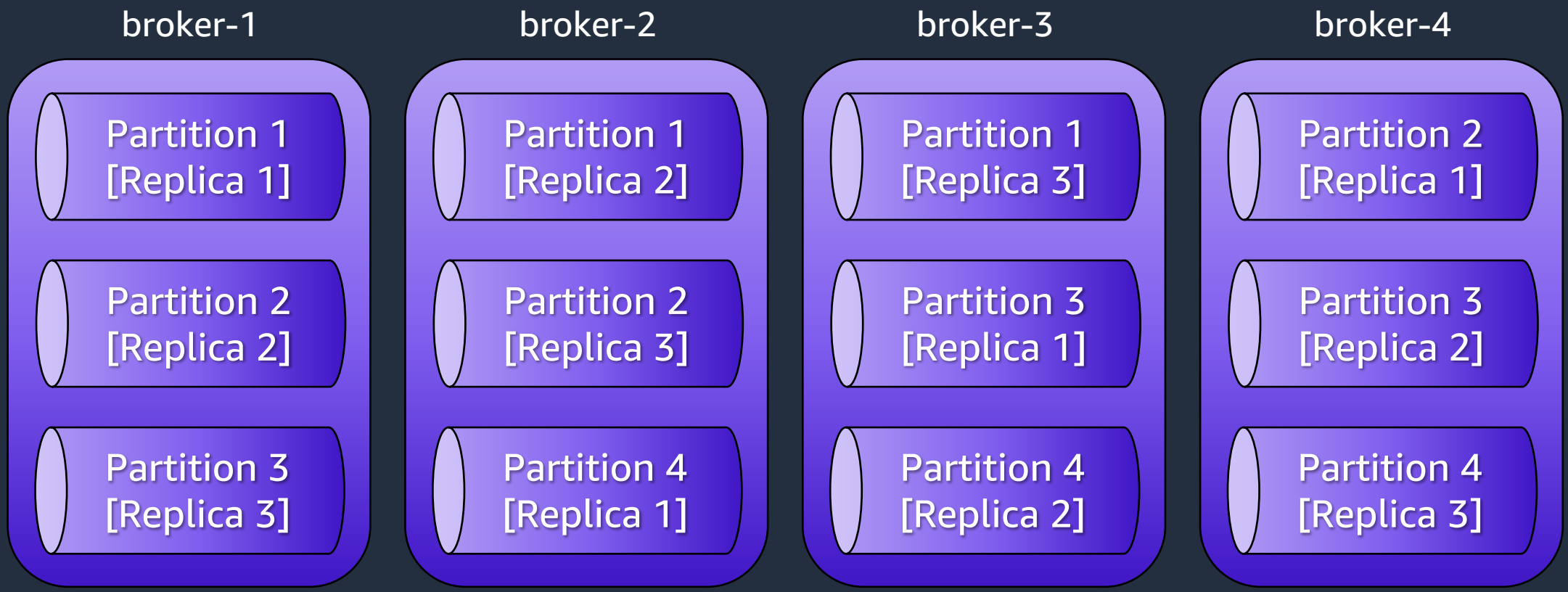


How to reassign the partitions?

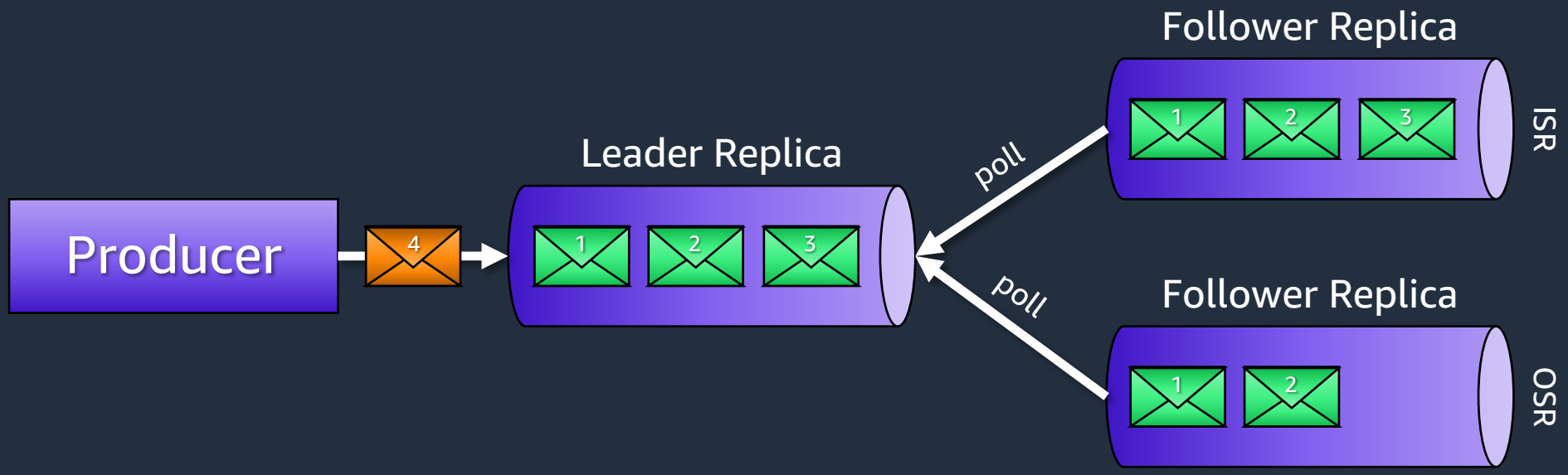
1. **Bounce the entire cluster:** leads to unavailability but once its back the partitions are reassigned.
2. **kafka-reassign-partitions:** CLI tool you where you can reassign the partitions with the cluster online.

Partitions as unit of durability

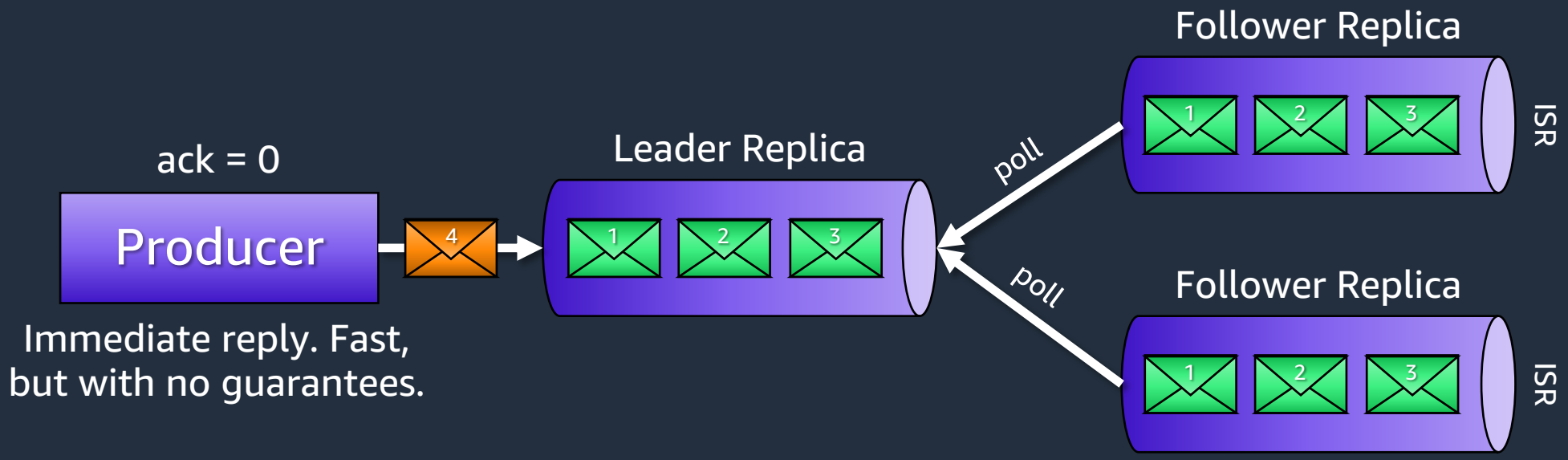
A topic with 4 partitions and 3 replicas



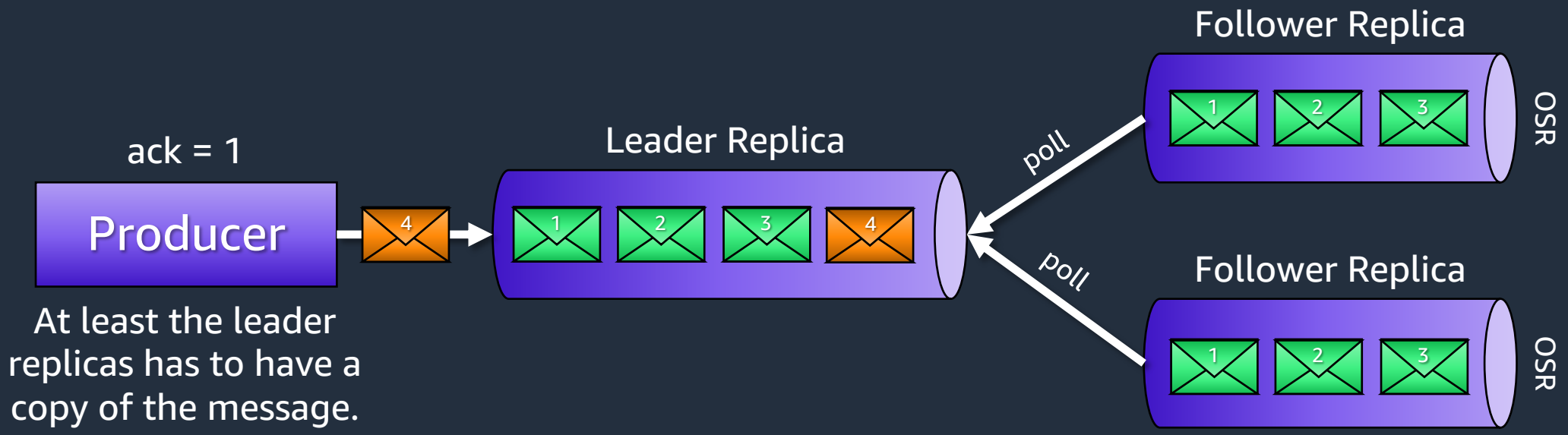
Types of replicas: leaders and followers



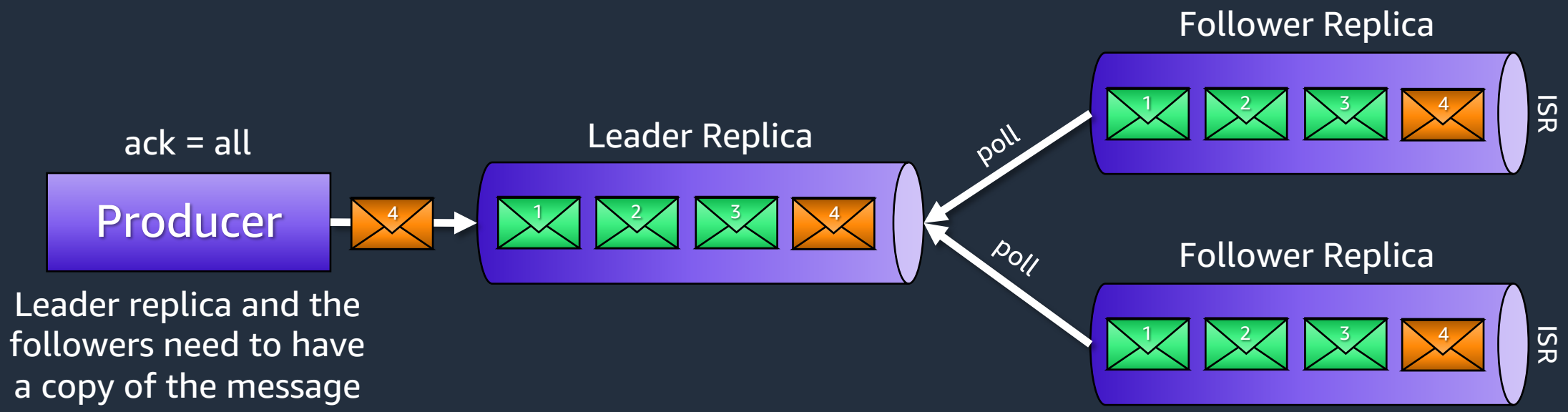
Data consistency with replicas



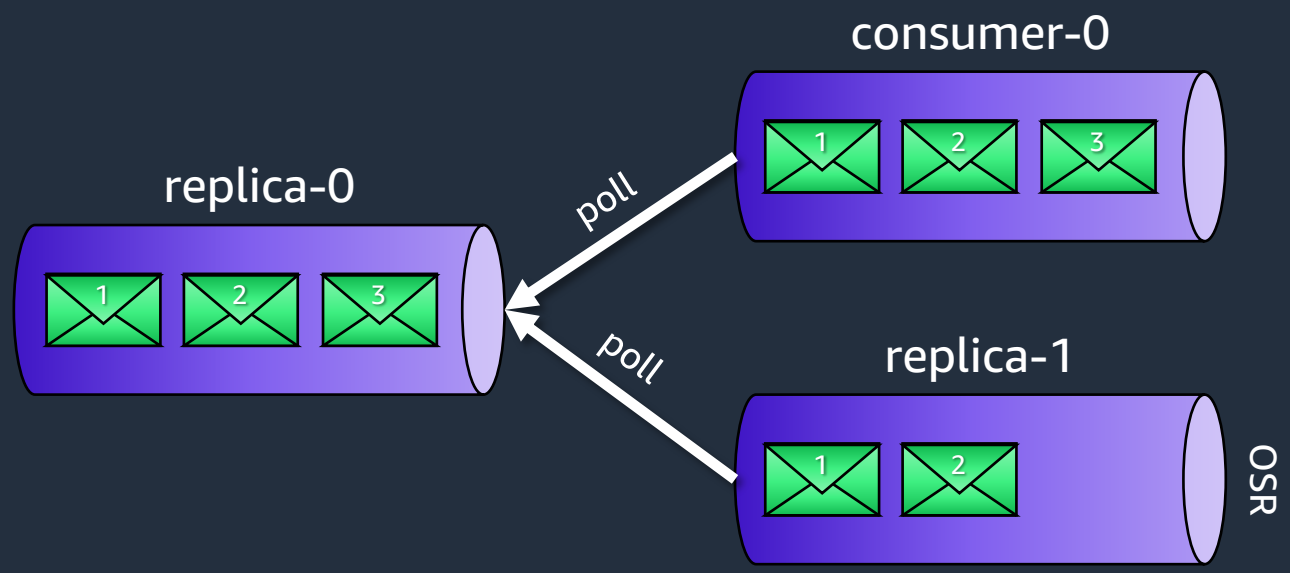
Data consistency with replicas



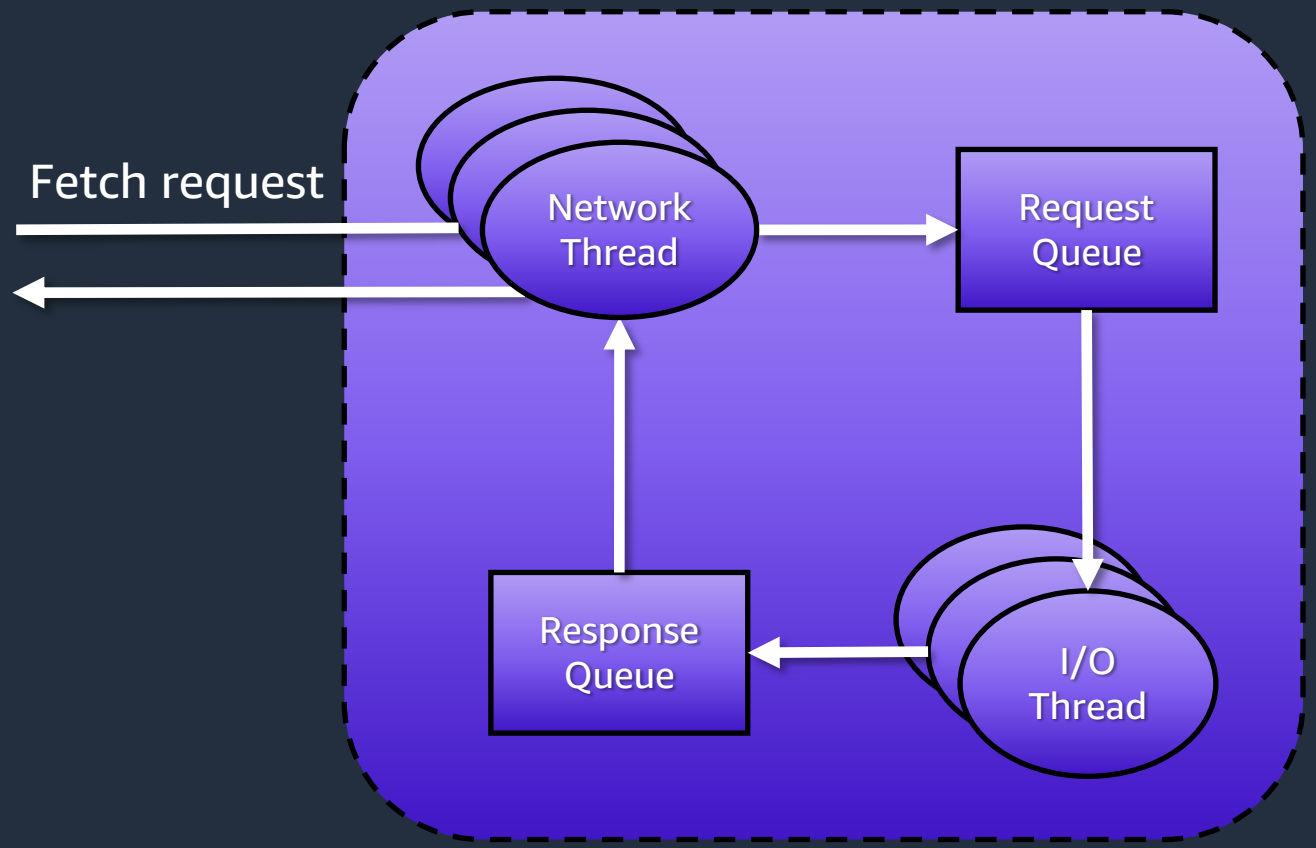
Data consistency with replicas



Why aren't all replicas ISRs?



Replication processing model



Network thread pool size: **5**

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

I wrote everything I showed here in this 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