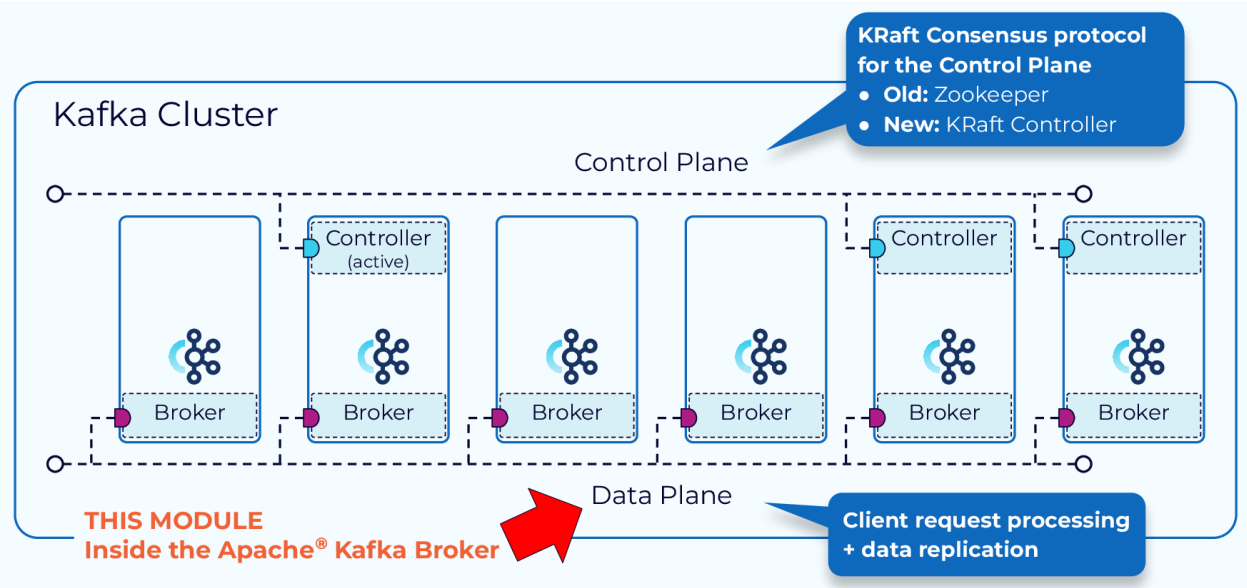
Inside the Apache Kafka Broker

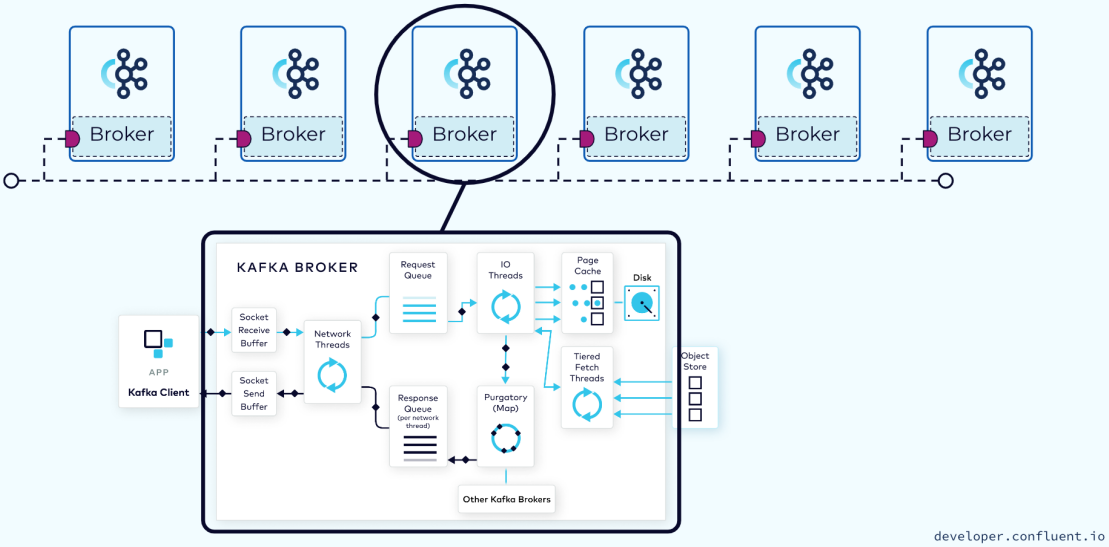
Kafka Manages Data and Metadata Separately



- The functions within a Kafka cluster are broken up into a data plane and a control plane.
- The control plane handles management of all the metadata in the cluster.
- The data plane deals with the actual data that we are writing to and reading from Kafka.

Apache Kafka Broker

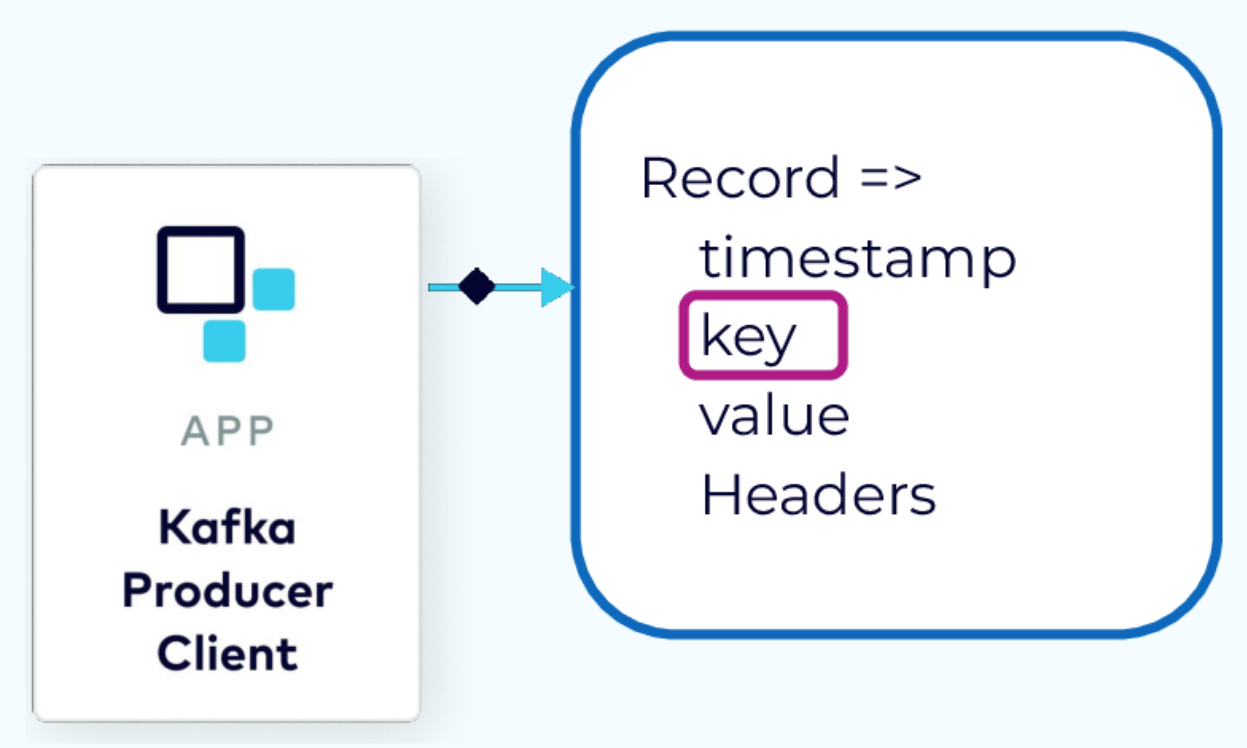
- An computer, instance, or container running the Kafka process
- Manage partitions
- Handle write and read requests
- Manage replication of partitions
- Intentionally very simple



- Client requests fall into two categories: produce requests and fetch requests.
- A produce request is requesting that a batch of data be written to a specified topic.
- A fetch request is requesting data from Kafka topics.

The Produce Request

Partition Assignment



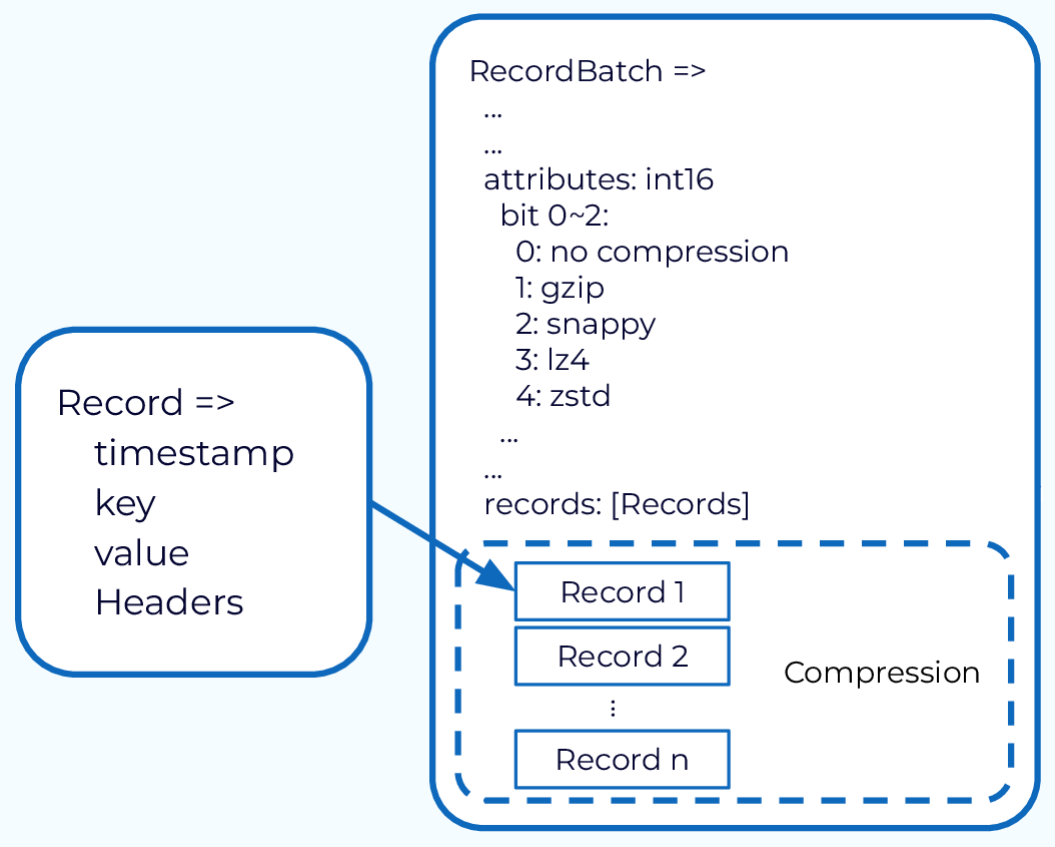
- When a producer is ready to send an event record, it will use a configurable partitioner to determine the topic partition to assign to the record.
- If the record has a key, then the default partitioner will use a hash of the key to determine the correct partition.

```
key=cts

hash(key) % no of partitions = 2
```

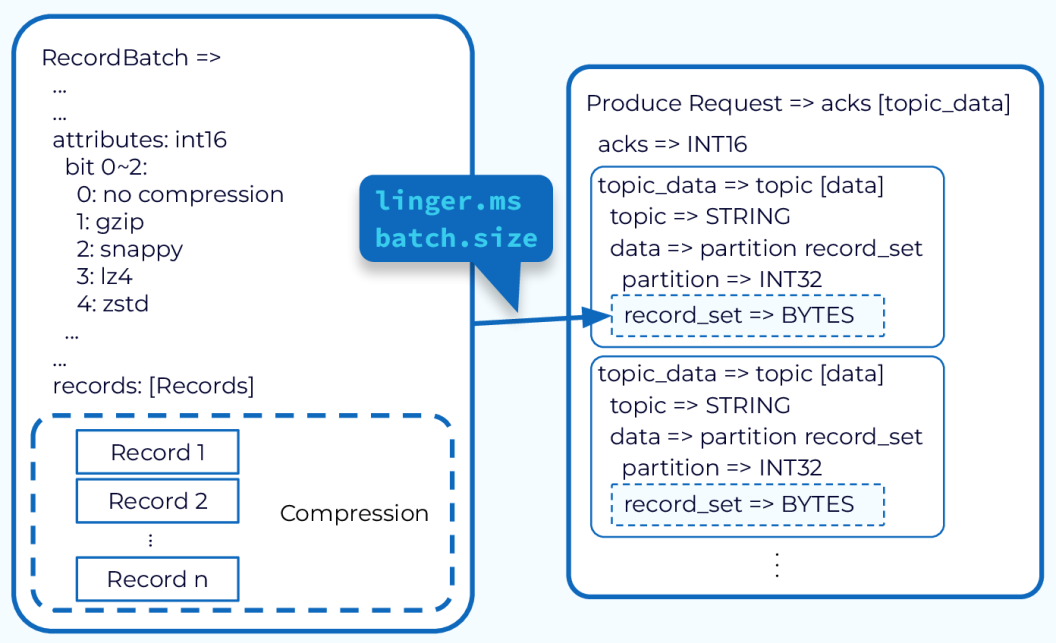
- After that, any records with the same key will always be assigned to the same partition.
- If the record has no key then a partition strategy is used to balance the data in the partitions.

Record Batching

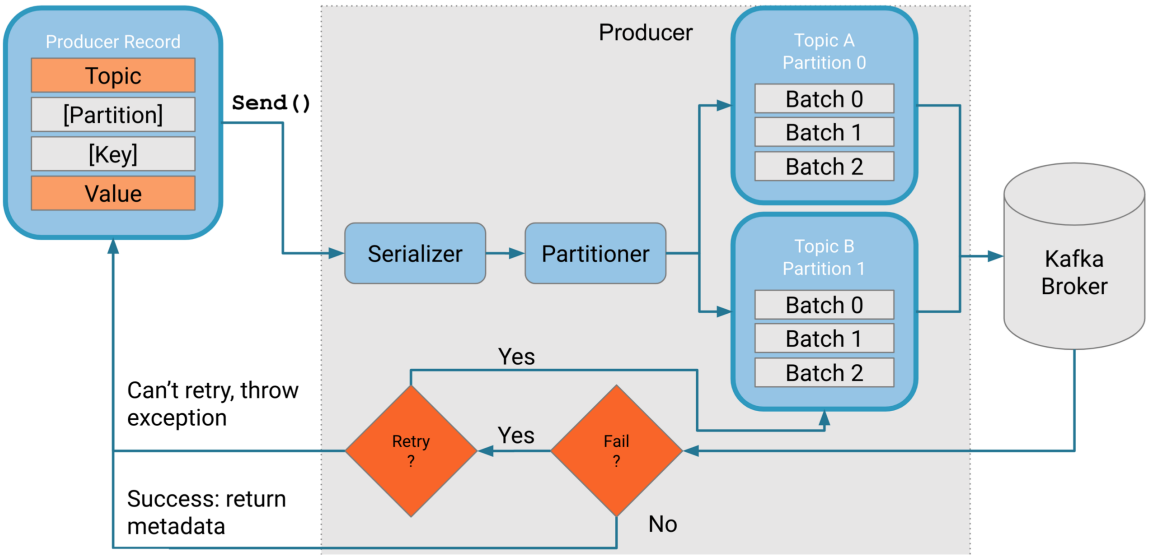


- Sending records one at a time would be inefficient due to the overhead of repeated network requests.

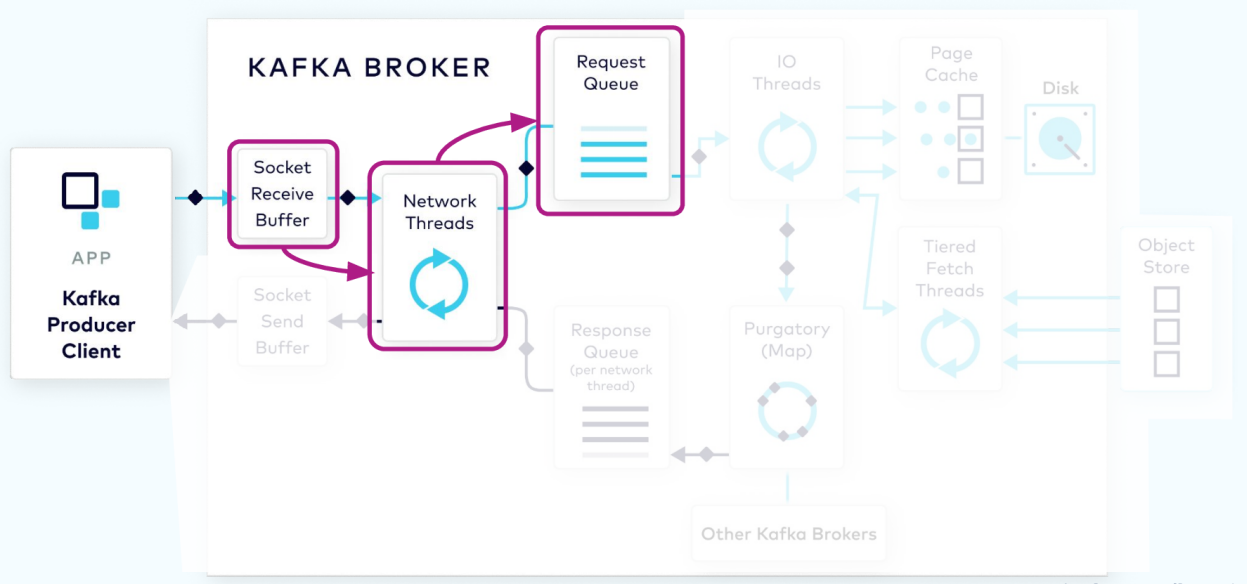
- So, the producer will accumulate the records assigned to a given partition into batches.
- Batching also provides for much more effective compression, when compression is used.



- The producer also has control as to when the record batch should be drained and sent to the broker.
- This is controlled by two properties. One is by **time**. The other is by **size**.
- So once enough time or enough data has been accumulated in those record batches, those record batches will be drained, and will form a produce request.
- And this produce request will then be sent to the broker that is the leader of the included partitions.

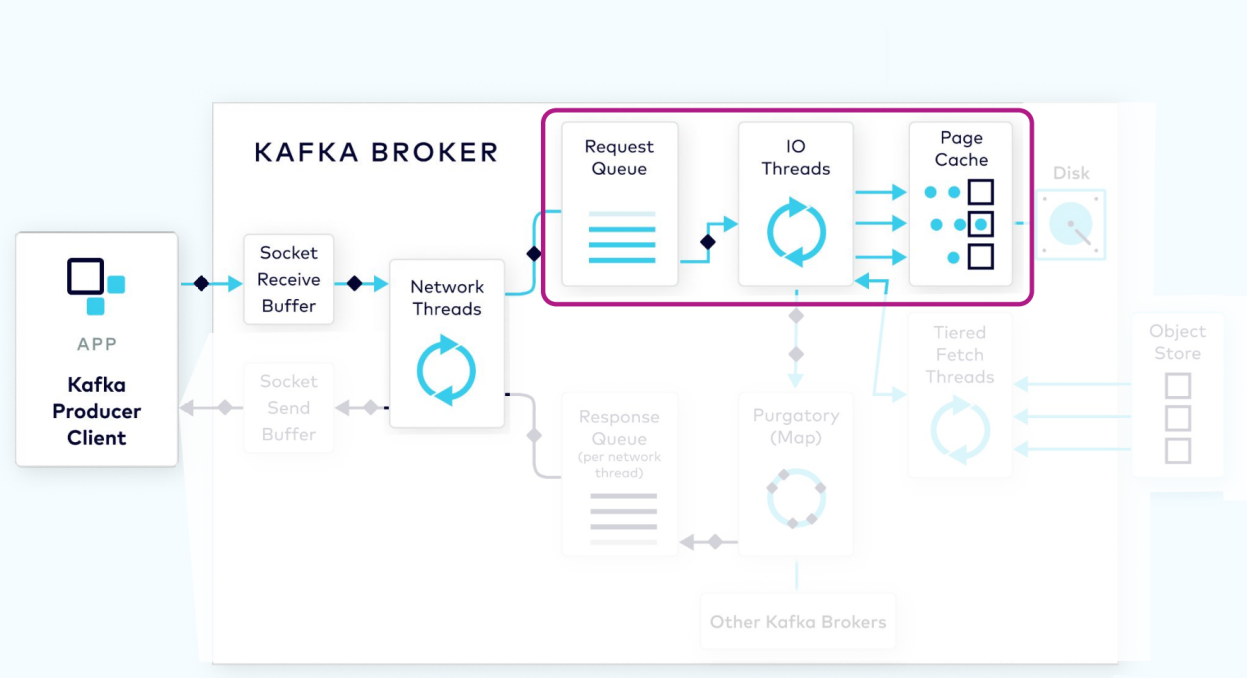


Network Thread Adds Request to Queue



- The request first lands in the broker's socket receive buffer where it will be picked up by a network thread from the pool.
- That network thread will handle that particular client request through the rest of its lifecycle.
- The network thread will read the data from the socket buffer, form it into a produce request object, and add it to the request queue.

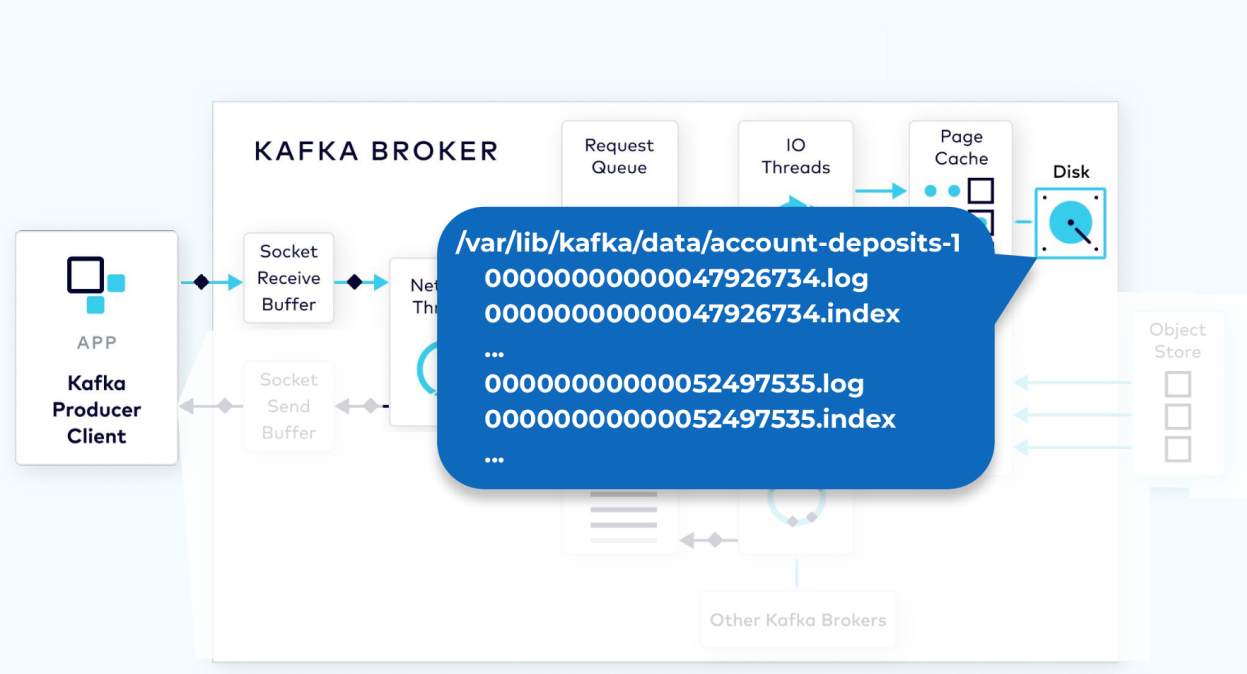
I/O Thread Verifies and Stores the Batch

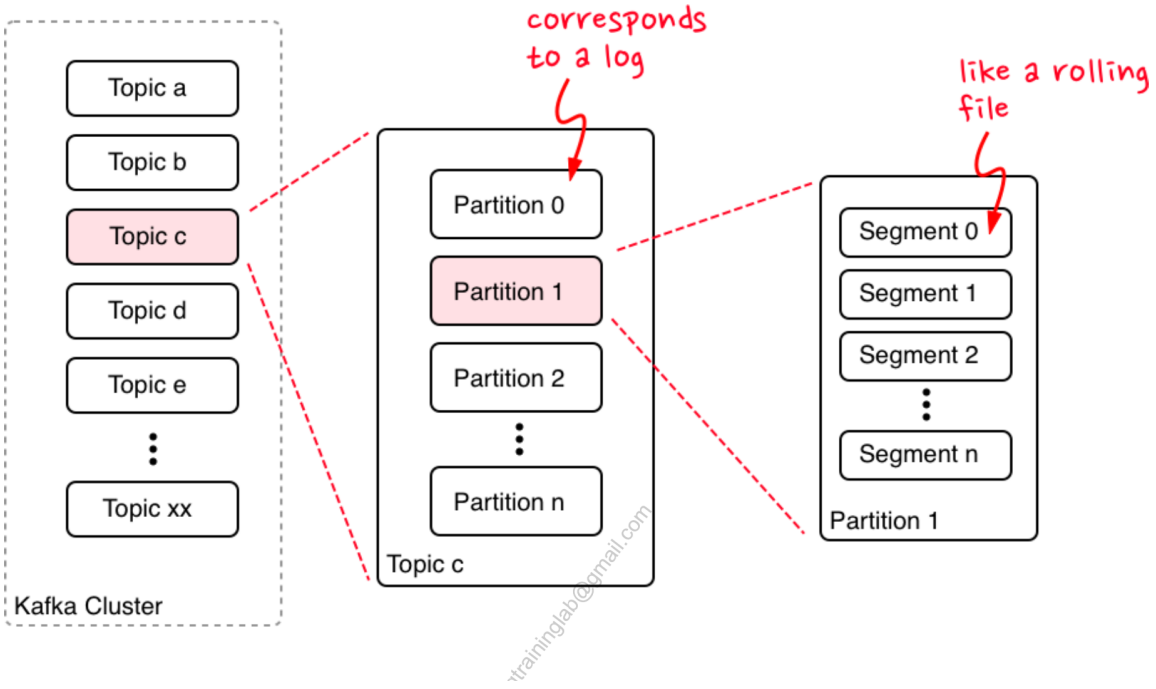


- Next, a thread from the I/O thread pool will pick up the request from the queue.
- The I/O thread will perform some validations, including a CRC check of the data in the request.

- It will then append the data to the physical data structure of the partition, which is called a commit log.

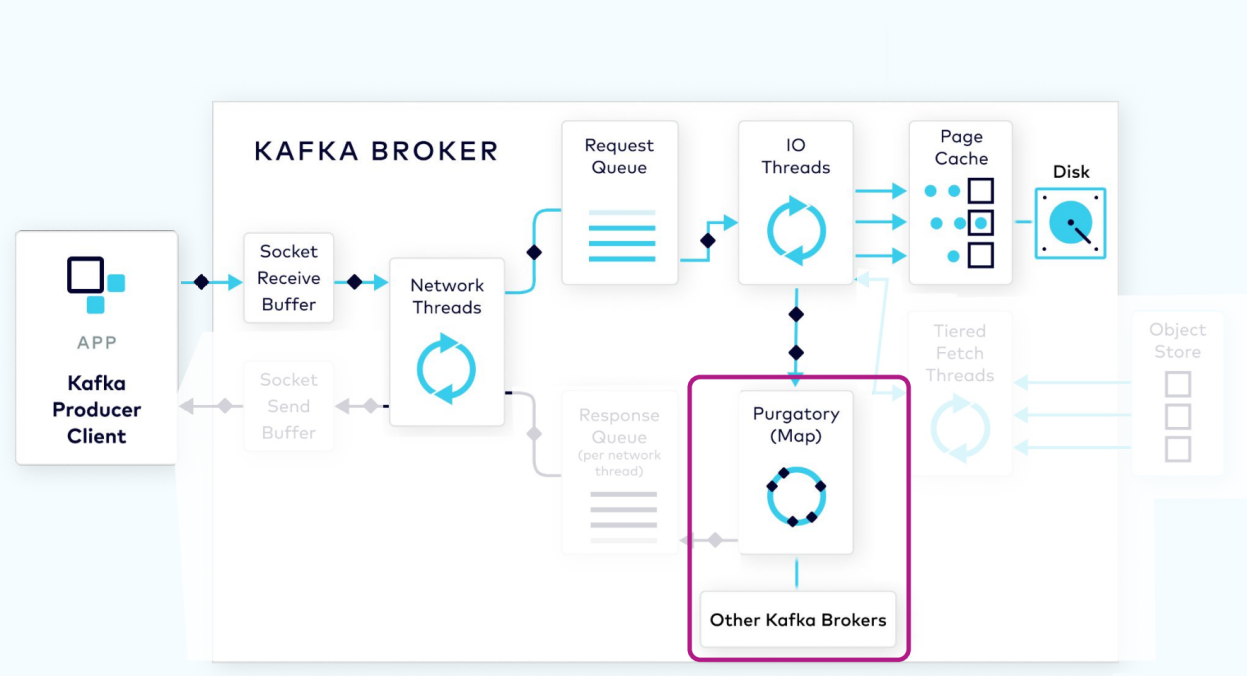
Kafka Physical Storage





- On disk, the commit log is organized as a collection of segments.
- Each segment is made up of several files.
- One of these, a .log file, contains the event data.
- A second, a .index file, contains an index structure, which maps from a record offset to the position of that record in the .log file.

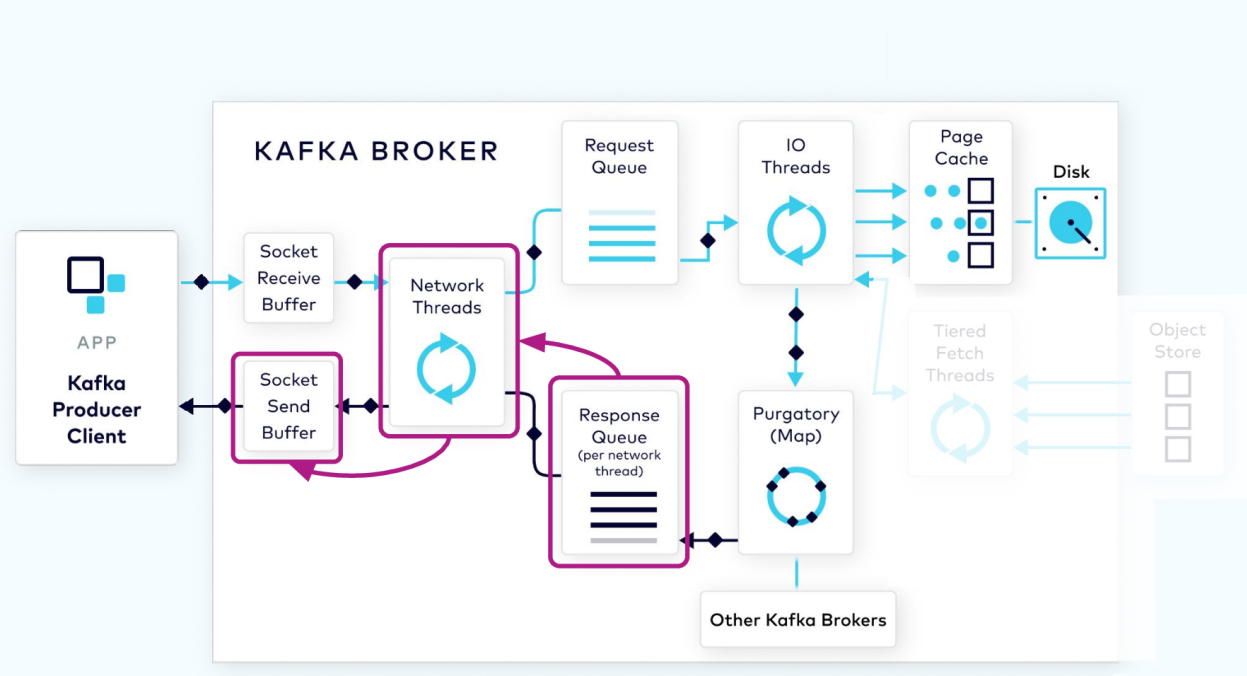
Purgatory Holds Requests Until Replicated



- Since the log data is not flushed from the page cache to disk synchronously, Kafka relies on replication to multiple broker nodes, in order to provide durability.
- By default, the broker will not acknowledge the produce request until it has been replicated to other brokers.

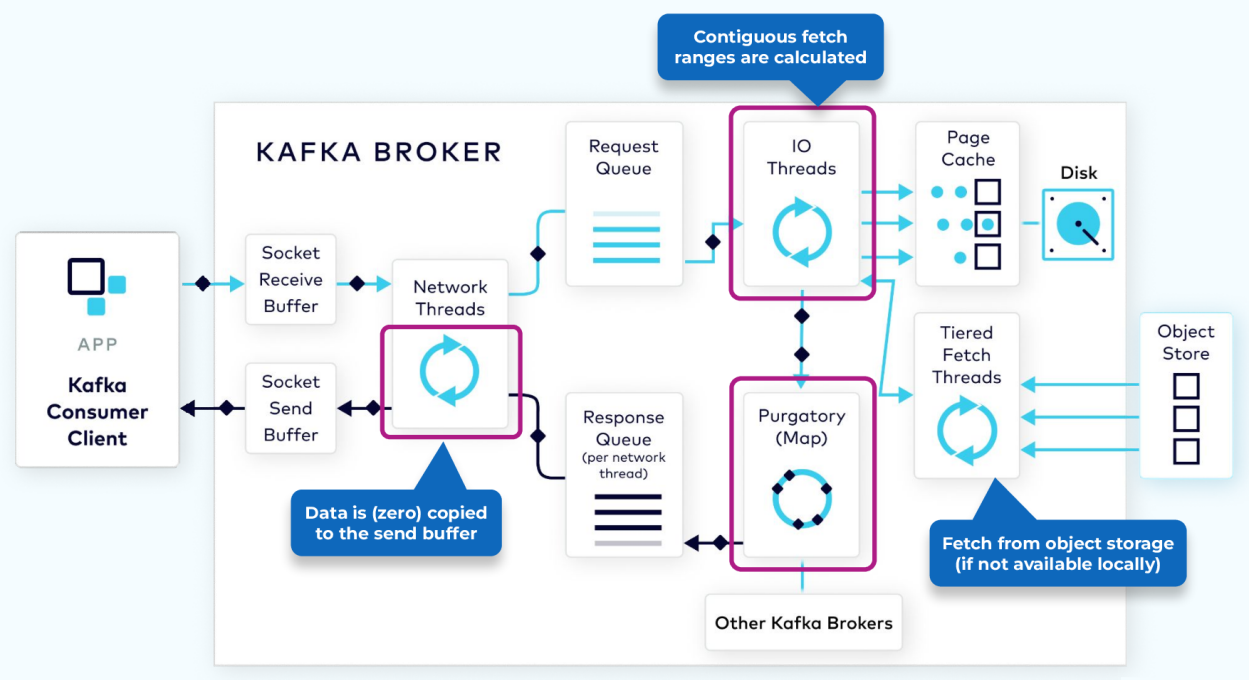
- To avoid tying up the I/O threads while waiting for the replication step to complete, the request object will be stored in a map-like data structure called purgatory (it's where things go to wait).
- Once the request has been fully replicated, the broker will take the request object out of purgatory, generate a response object, and place it on the response queue.

Response Added to Socket



- From the response queue, the network thread will pick up the generated response, and send its data to the socket send buffer.
- The network thread also enforces ordering of requests from an individual client by waiting for all of the bytes for a response from that client to be sent before taking another object from the response queue.

The Fetch Request



- In order to consume records, a consumer client sends a fetch request to the broker, specifying the topic, partition, and offset it wants to consume.
- The fetch request goes to the broker's socket receive buffer where it is picked up by a network thread.
- The network thread puts the request in the request queue, as was done with the produce request.
- The I/O thread will take the offset that is included in the fetch request and compare it with the `.index file` that is part of the partition segment.
- That will tell it exactly the range of bytes that need to be read from the corresponding `.log file` to add to the response object.

- However, it would be inefficient to send a response with every record fetched, or even worse, when there are no records available.
- To be more efficient, consumers can be configured to wait for a minimum number of bytes of data, or to wait for a maximum amount of time before returning a response

to a fetch request.

- While waiting for these criteria to be met, the fetch request is sent to purgatory.
- Once the size or time requirements have been met, the broker will take the fetch request out of purgatory and generate a response to be sent back to the client.

The rest of the process is the same as the produce request.

- Broker (aka. Kafka Server)
