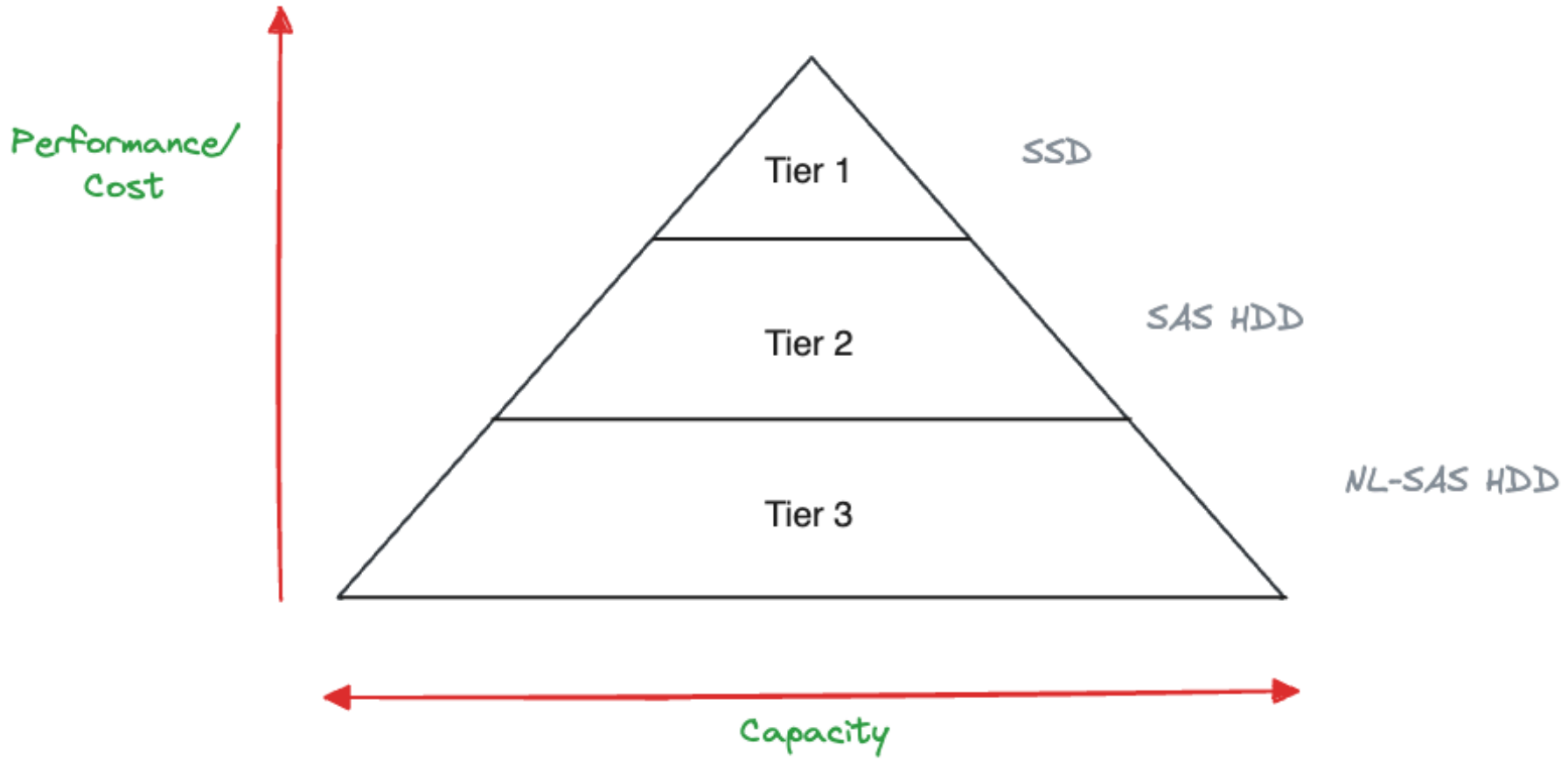
1 Tiered Storage for Kafka - Storage Tiering

Storage tiering is a method used in data storage and management. It involves storing data in different types of storage media to reduce total storage costs.

Data that is frequently accessed is kept in the primary storage which is usually faster and more expensive, while data that is less frequently accessed is moved to secondary storage which is typically slower but cheaper.

This helps in managing large volumes of data without the need for a proportional increase in hardware resources.

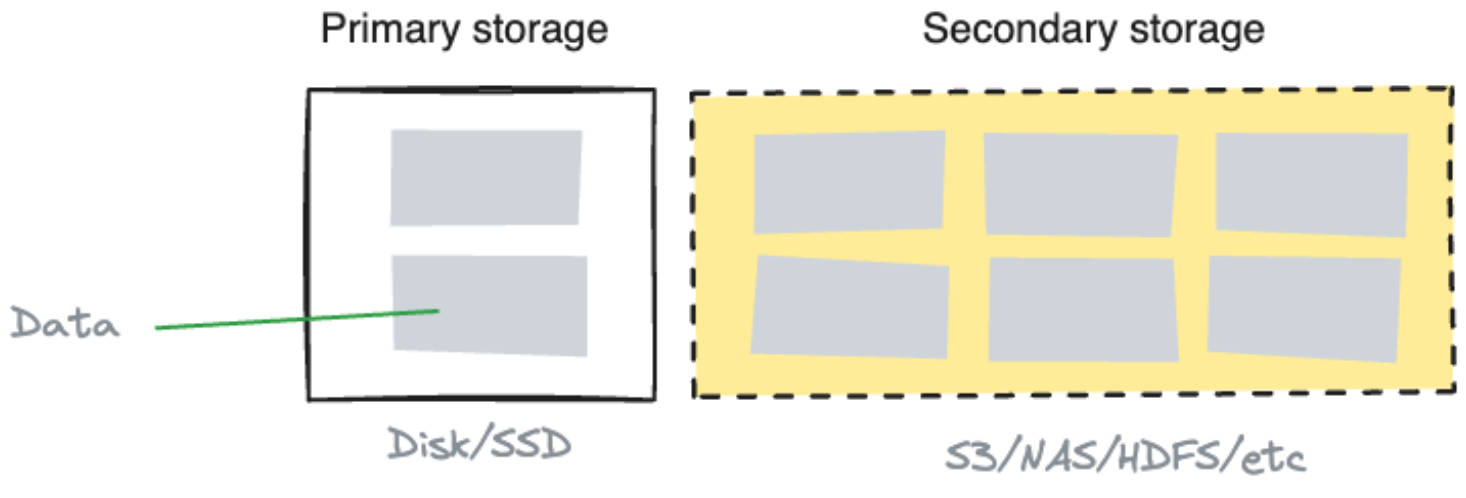


As the above illustration shows, the cost of storage increases as more performance is needed.

In the following pages, we will discuss how a streaming data platform like Apache Kafka implements tiered storage. Although the concepts are the same, they also apply to other Kafka API-compatible streaming data platforms, such as Redpanda.

2 Tiered Storage for Kafka - Primary vs. Secondary Storage

When considering a streaming data platform such as Apache Kafka or any Kafka API-compatible platform, like Redpanda, it's important to note their two-tiered storage architecture.



- **Primary Storage**: This is the high-performance, low-latency storage typically used for the most recent and frequently accessed data. It usually consists of fast SSDs. For example, a broker may keep the last hour's data in the primary storage for efficient access.

- **Secondary Storage**: This storage tier is used for older data that is accessed less frequently. It is typically more cost-effective and has higher latency compared to primary storage. Examples include cloud-based object storage systems like Amazon S3, Google Cloud Storage, or on-premises solutions like HDFS.

As data ages and becomes less frequently accessed, Kafka automatically moves it from the primary storage tier to the secondary storage tier. This process is transparent to users and applications accessing the data.

When we refer to Kafka's tiered storage, we're talking about the secondary storage tier where Kafka stores its older data, typically an object storage like S3.

Let's delve into how it operates internally.

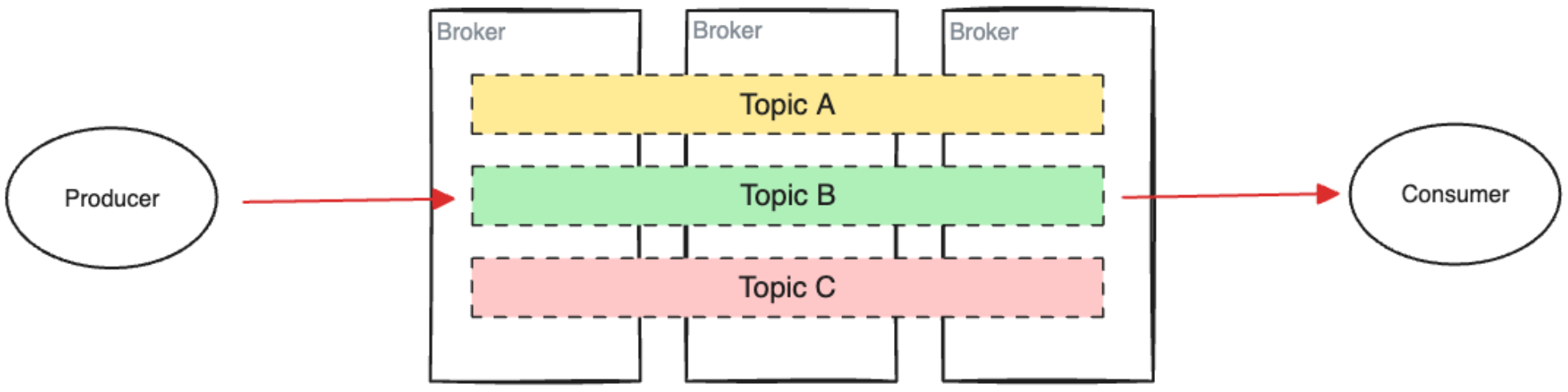
3 Tiered Storage for Kafka - Storage Hierarchy of Kafka

Kafka's storage engine uses a hierarchy of storage objects. Understanding this hierarchy can help us better comprehend how tiered storage works.

Topics

A topic is a logical grouping of messages. It’s analogous to a table from the relational database world, which keeps related records together.

A topic is at the top of Kafka's information hierarchy. Topic implements the publish-subscribe functionally, allowing multiple client applications to both produce data to and consume data from it simultaneously.



Partitions

Topics are not continuous; rather, they are composed of partitions.

A topic partition stores a subset of data belonging to a topic. A topic can have more than one partition and these partitions are scattered across different brokers of a cluster to provide load balancing and fault tolerance.

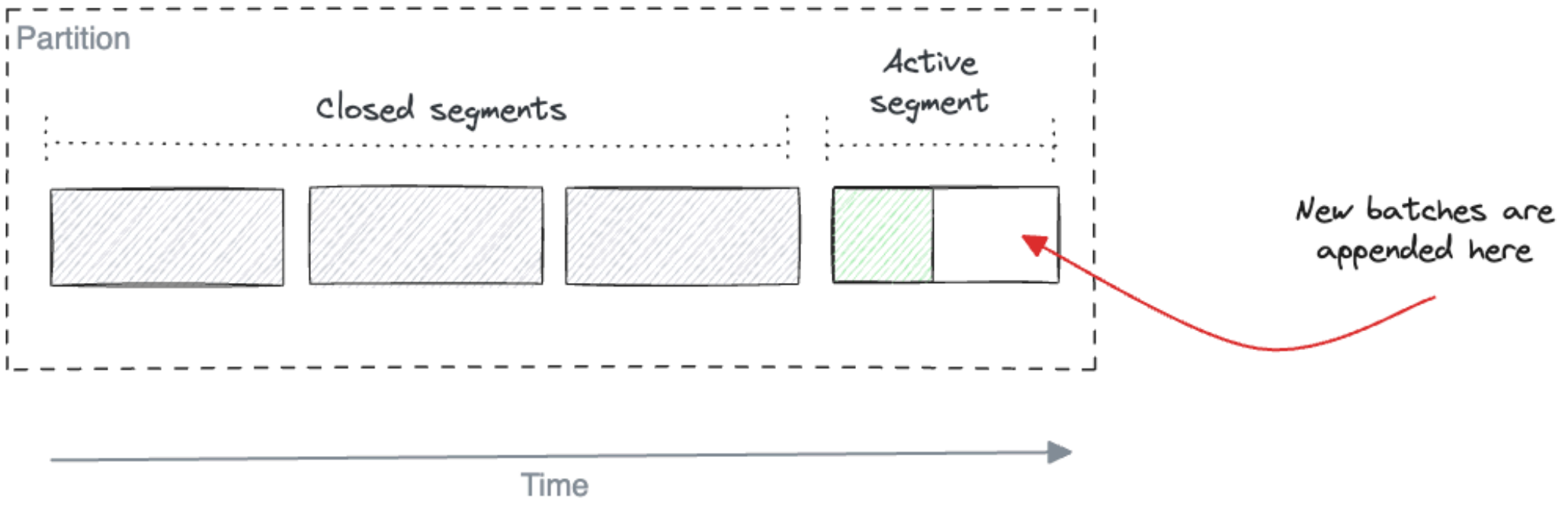
4 Tiered Storage for Kafka - Storage Hierarchy of Kafka

Segments

Each topic partition is further divided into multiple segments.

A segment is an append-only ordered log file that holds a subset of messages belonging to a partition. Segments receive writes from producers as well as serve reads to consumers. While topics and partitions are abstract concepts, a segment is the smallest unit of storage within a topic partition that is physically present on the disk.

For each partition, only one active segment always receives data. Once enough messages accumulate in the active segment, it is closed or "**rolls over**" to the next active segment. This segment size threshold is configurable.



5 Tiered Storage for Kafka - How it Works?

As Kafka ingests more data, the active segment within each partition fills up faster. This leads to more closed segments occupying the primary storage. This can increase the demand for storage and may require provisioning more hardware to accommodate it.

What if we could archive older, closed log segments into secondary storage? That's the idea behind Kafka's Tiered Storage feature.

Kafka tiered storage

Tiered storage allows Kafka to offload older, less frequently accessed data from primary storage (usually faster and more expensive) to secondary storage (typically slower but cheaper), freeing up space on faster, more expensive tiers for new segments.

This allows Kafka to handle large volumes of data without requiring a proportional increase in hardware resources.

Storage tiering/retention policies

Kafka uses configurable tiering policies to determine when data should be moved from primary to secondary storage. These policies can be based on factors such as segment age, size, or the amount of time data has been inactive.

For example, a policy might specify that segments older than a week should be moved to secondary storage. The relevant configurations in Redpanda are as follows.

```
retention.local.target.bytes=1073741824
retention.local.target.ms=168
```

Here, **retention.local.target.bytes** sets the maximum size of a single log segment. In this case, it's set to 1 GB. **retention.local.target.ms** sets the duration after which local segments will be deleted. It's set to 168 hours, which is equivalent to one week.

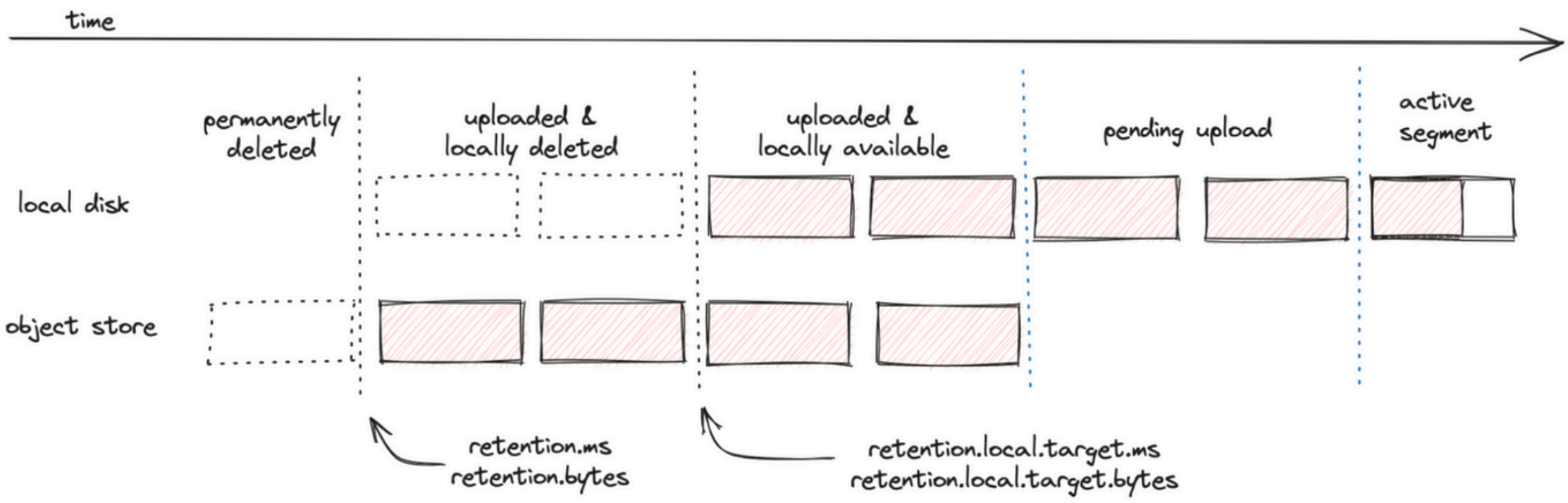
6 Tiered Storage for Kafka - How it Works?

Log segment offloading

Once a segment meets the criteria defined by the tiering policy, Kafka triggers the offloading process. This involves copying the segment from the primary storage (e.g., local disks or SSDs) to the secondary storage (e.g., cloud storage like Amazon S3 or Google Cloud Storage).

During offloading, the segment remains available on the primary storage until the copy operation is complete to ensure data availability and consistency.

The following illustrates how Redpanda offloads its log segments to object store based on the time.



Metadata update and management

When segments exist in multiple locations, Kafka needs to keep track of them.
Kafka maintains metadata about the location of each segment. This metadata includes information about which segments are stored in primary storage and which have been offloaded to secondary storage.

When a segment is offloaded, Kafka updates its metadata to reflect the new location.

7 Tiered Storage for Kafka - Reading the Data Back

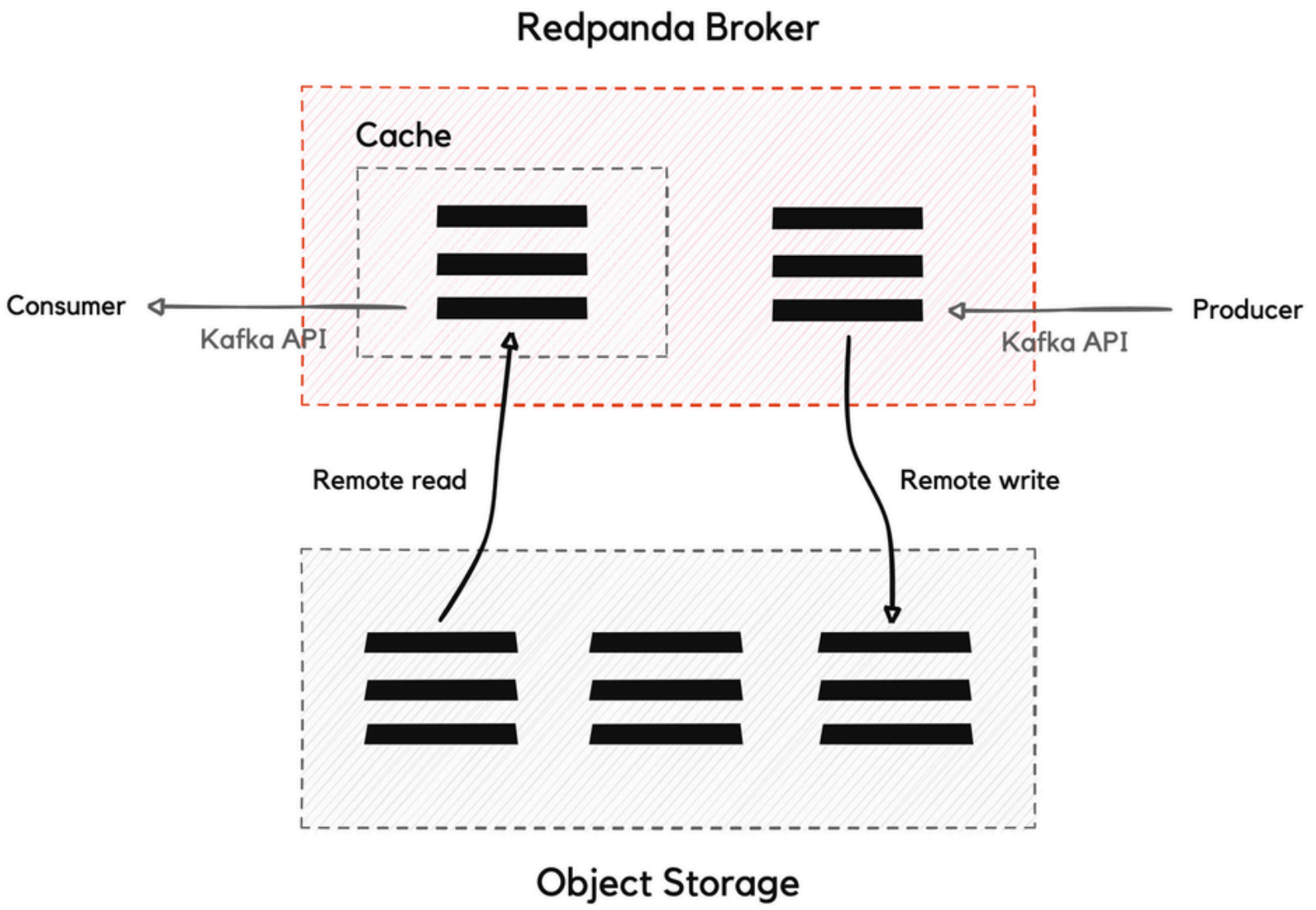
Kafka's consumer API ensures that data access is transparent to clients.
When a consumer requests data, Kafka checks its metadata to determine whether the requested segment is on primary or secondary storage.

If the data is in secondary storage, Kafka fetches the segment from there and serves it to the consumer without any additional action required by the client. For instance, if a consumer wants to read a seven-year-old message, it simply needs to send a fetch request with the appropriate offset.

Read caching

Depending on the implementation and configuration, the broker might temporarily store the fetched segment locally to serve subsequent requests more efficiently. To improve performance for future reads, the broker may cache the recently accessed segments or portions of segments fetched from secondary storage.

This reduces latency for subsequent read requests for the same data.



8 Tiered Storage for Kafka - Benefits and Use Cases

Kafka can be configured to eventually delete segments from secondary storage based on retention policies, ensuring that storage costs are kept under control by only retaining data as long as necessary.

These retention policies can be aligned with compliance and business requirements, specifying the duration for which data must be kept.

To wrap up, let's discuss potential use cases and benefits of using tiered storage with Kafka.

Benefits of using tiered storage

- **Scalability**: Tiered storage allows Kafka clusters to scale more efficiently by leveraging cheaper, scalable storage solutions for historical data.

- **Cost Savings**: By using a mix of storage types, organizations can reduce their overall storage costs without compromising on data availability.

- **Improved Performance**: Keeping frequently accessed data in fast primary storage ensures high performance for current workloads.

- **Data Retention**: It becomes feasible to retain larger amounts of data for longer periods, which can be important for compliance, analytics, and other use cases.

Use cases for tiered storage

- **Log Retention**: For applications that generate high volumes of log data, tiered storage enables long-term retention without incurring high costs.

- **Analytics and Reporting**: Historical data can be stored cost-effectively and accessed when needed for analytics and reporting.

- **Regulatory Compliance**: Organizations can meet regulatory requirements for data retention without the prohibitive costs associated with storing all data on high-performance storage.