KAFKA Workshop

Jul 2025
Egypt - Karim Tawfik

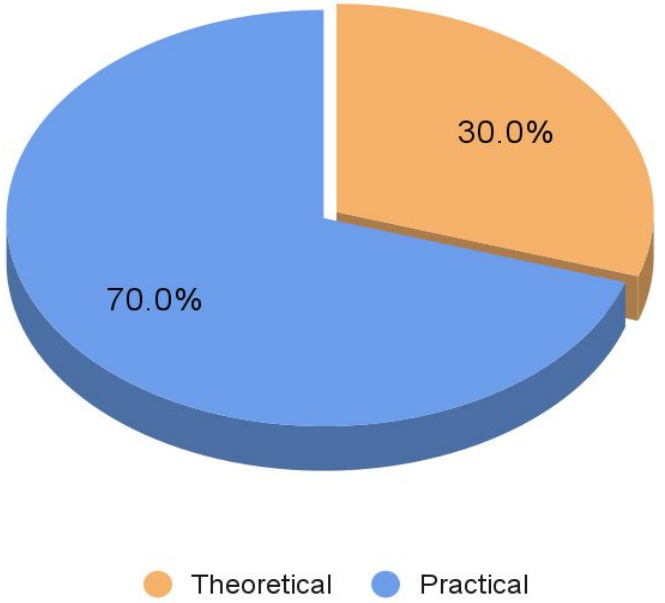
$(Whoami)

- Co-founder & CTO at sympl.
- ~14 years of experience in Software industry.
- Has been into the data domain ~9 years ago, and trapped by data since then.
- I like playing Tennis 🎾.

By End of the workshop, What will you be able to Build with Kafka

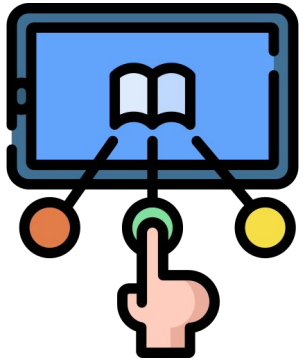
1. Real time Streaming Apps.
2. CDC pipelines from OLTP to OLAP.
3. Schema management / Evolution.
4. Exposing internal kafka cluster via API for external services.
5. Monitoring and alerting on the entire ecosystem.

Workshop content Structure



Engaging Education Approach

Collaborative



Interactive

Interactive

Feedback

Without Further Due, Let’s Start [Day-1]

Day-1 Agenda

---

1. Introduction
2. What is Apache Kafka?
3. Kafka Use Cases
4. Why Apache Kafka?
5. Messaging Models (in Kafka)
6. Kafka Fundamentals
7. Advanced Topics in Kafka
8. Wrap-up / Q&A
9. Hands-on Lab

What is Apache Kafka?

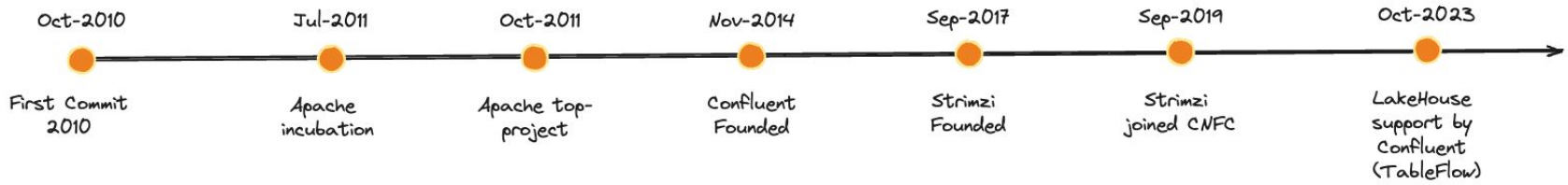
New Beginning Definition

Kafka is an event streaming platform used to collect, process, store, and integrate data at scale. It has numerous use cases including distributed logging, stream processing, data integration, and pub/sub messaging.

Kafka is a distributed and replicated, commit log.

Introduction

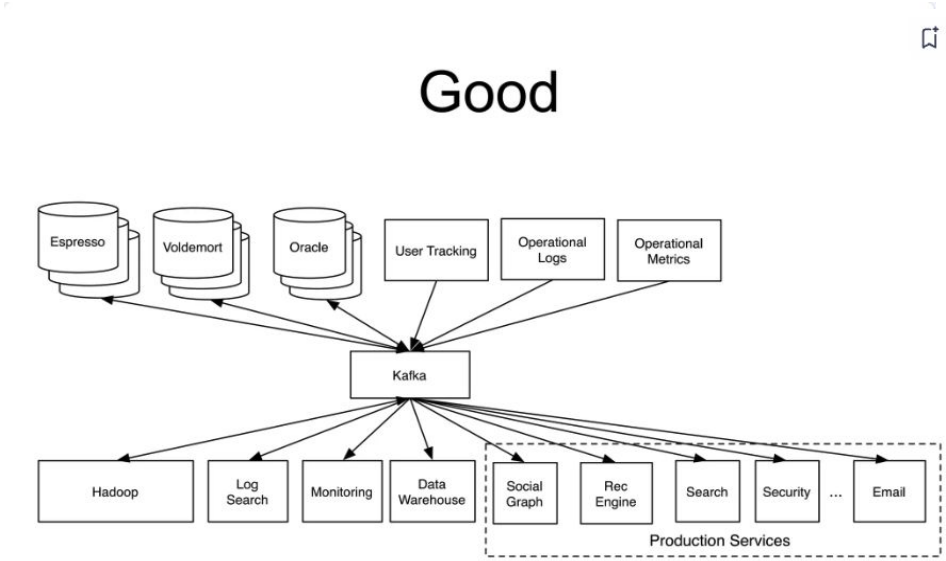
- Why is it named Kafka?
- Developed at Linkedin By Jay Kreps, Neha Narkhede, and Jun Rao.
- Why is it Developed?



Introduction -Why is it developed?

New Types of Systems
- Real Time Systems
- Batch/offline Systems

New Types of Data
- Events
- App Metrics
- App Logs

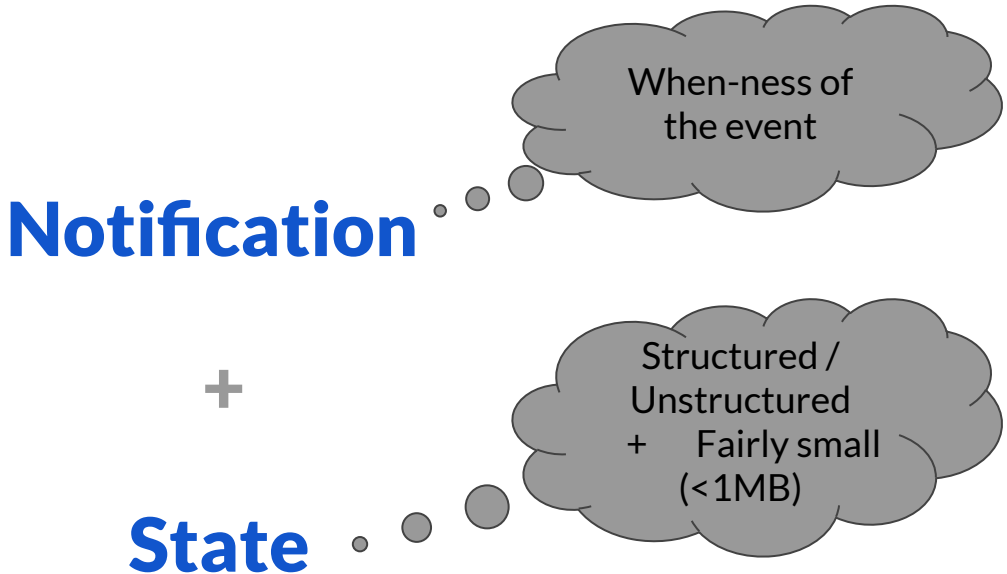


By Jay Kreps

What is an Event???

- **A thing that has happened (Action, Incident, Change,...)**
  a. **IoT**
  b. **Microservice result**
  c. **Business Process**
  d. **User action (on mobile/web)**

What is an Event???

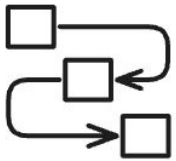


Kafka and Events – Key/Value Pairs

- Kafka Model events in Key/Value Pair
- Internally, Keys and Values are just sequence of bytes.
- Externally, Keys and Values must have a structure (Json, Bson, Avro, PBuf, and … PlainText 🤷‍♂️).
- Keys are usually serialized as Strings or Integers.
- Keys are not like how keys used in DB.
  - mainly used for: parallelization, data locality, and compaction.
  - Not for faster access.

Kafka Use Cases

Log Aggregation



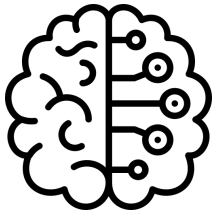
Event Sourcing and CQRS

Microservice Communication

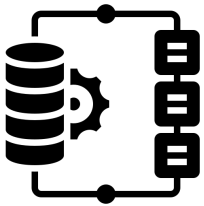
Fraud Detection

CDC

Real time Analytics

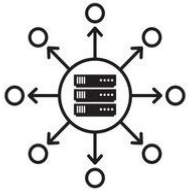


Machine Learning



Data Integrations

Kafka Use Cases (In Teleco)

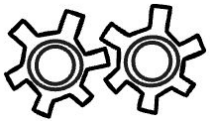


Middleware In/Ex

CDR Logging

Customer 360

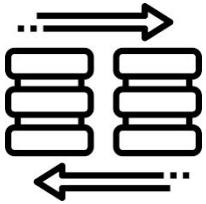
Edge Device real time monitoring



OSS/BSS integrations

Growth Verticals

M&A



System Migrations

Billing Cycles

SMS/RCS

APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

10 OUT OF 10

MANUFACTURING

7 OUT OF 10

BANKS

10 OUT OF 10

INSURANCE

8 OUT OF 10

TELECOM

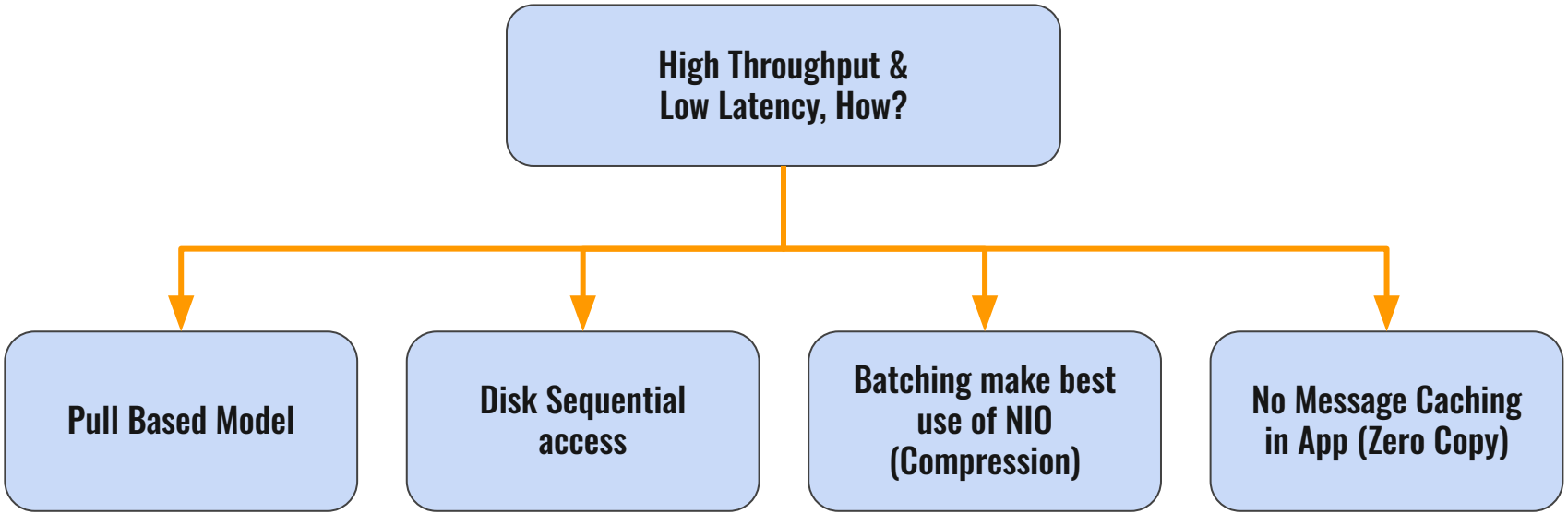
Kafka In the Wild



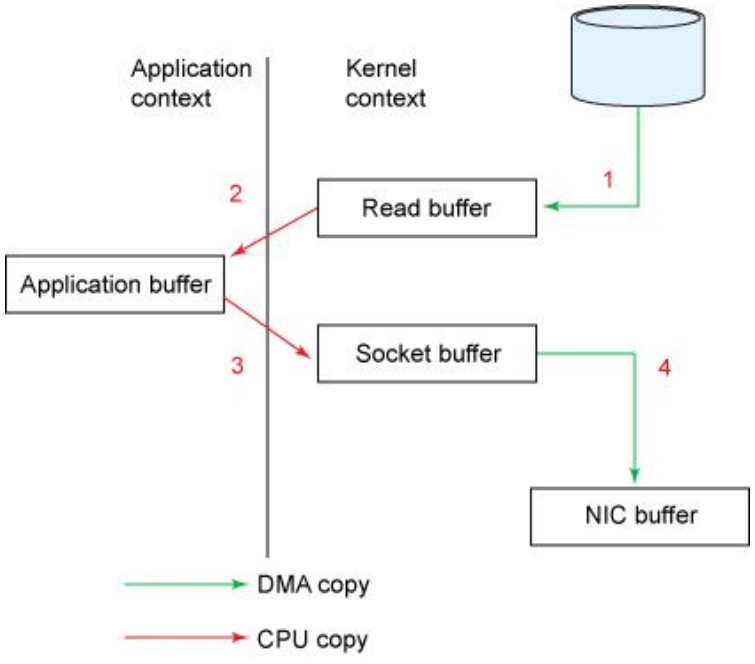

Why Kafka?

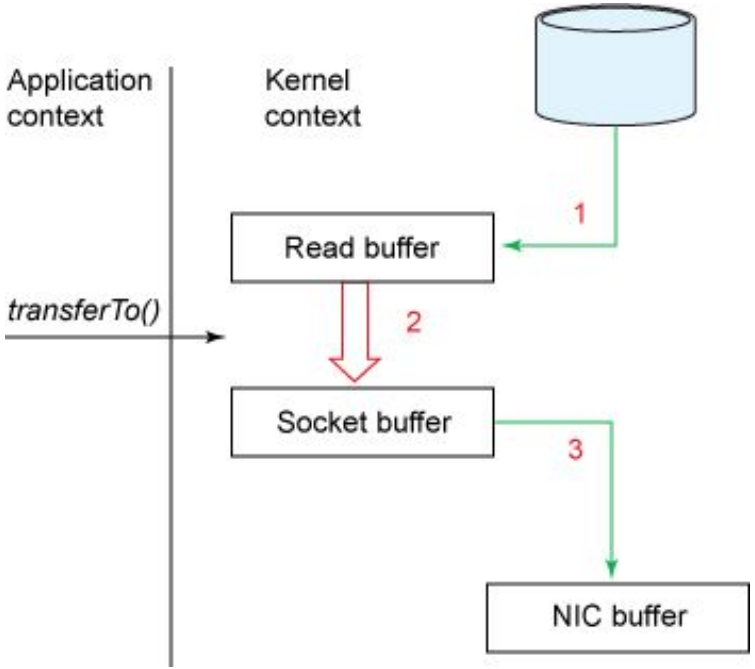
Feature	Desc
Scalability	Data Streams are partitioned and spread across brokers for seamless scalability
Durability	Data persisted into disk, and provide inter-cluster replication to avoid data loss
Reliability	Data is replicated,
Performance	- Binary communication over TCP - Zero copy feature - Batching mechanism that utilize the usage of compression algorithms
Growing Ecosystem	It is not just another MQ, it has a wide range of offerings and components satisfying huge amount of use cases (Connect, Streams, KSQL, SchemaReg,...)

Why Kafka Cont'd (Performance)?



Zero Copy In a nutshell

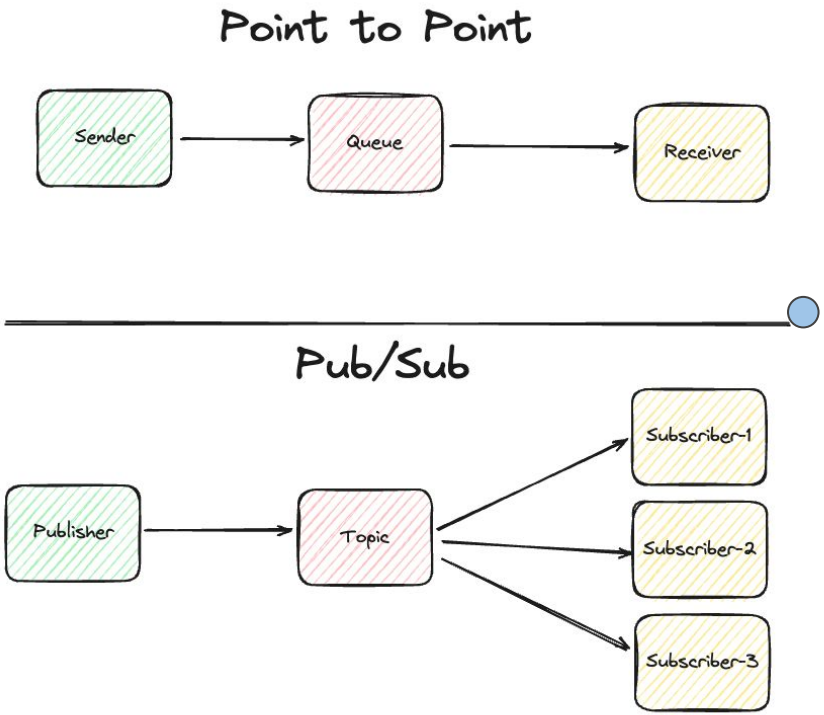




Kafka Vs RabbitMQ

Feature	Kafka	RabbitMQ
Performance	Kafka has real-time transmission of up to millions of messages per second.	RabbitMQ has low latency. It sends thousands of messages per second.
Core Architecture	Distributed log-based system	Traditional message broker (AMQP)
Scalability	Horizontally scalable, designed for large data volumes	Supports clustering for scalability but is generally vertically scalable
Message Ordering	Strong ordering within partitions	Offers ordered delivery but limited by single-threaded consumption in queues
Message Retention	Retains messages by default, configurable retention policies	Messages typically removed once acknowledged
Operational Model	Pull-based, consumers request data	Push-based, broker pushes messages to consumers.

Messaging Models



Kafka is supporting both models, how?

Apache Kafka Fundamentals

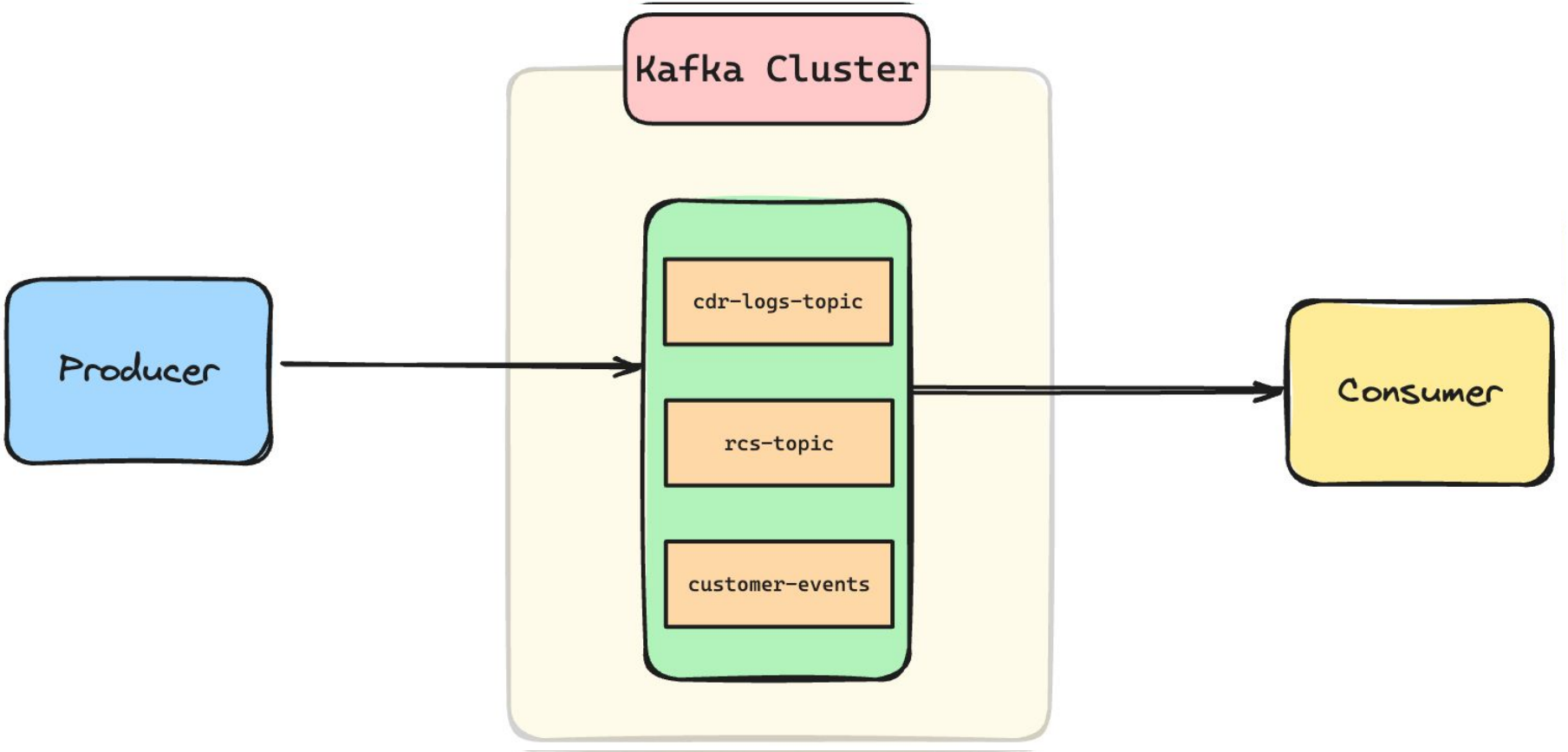
---

1. Kafka’s Architecture (Simple)
2. Kafka’s Ecosystem
3. Core Concepts
4. Log (Kafka 💙Logs)
5. Topics / Partitions / Replication
6. Producers
7. Consumers
8. Brokers / Zookeepers

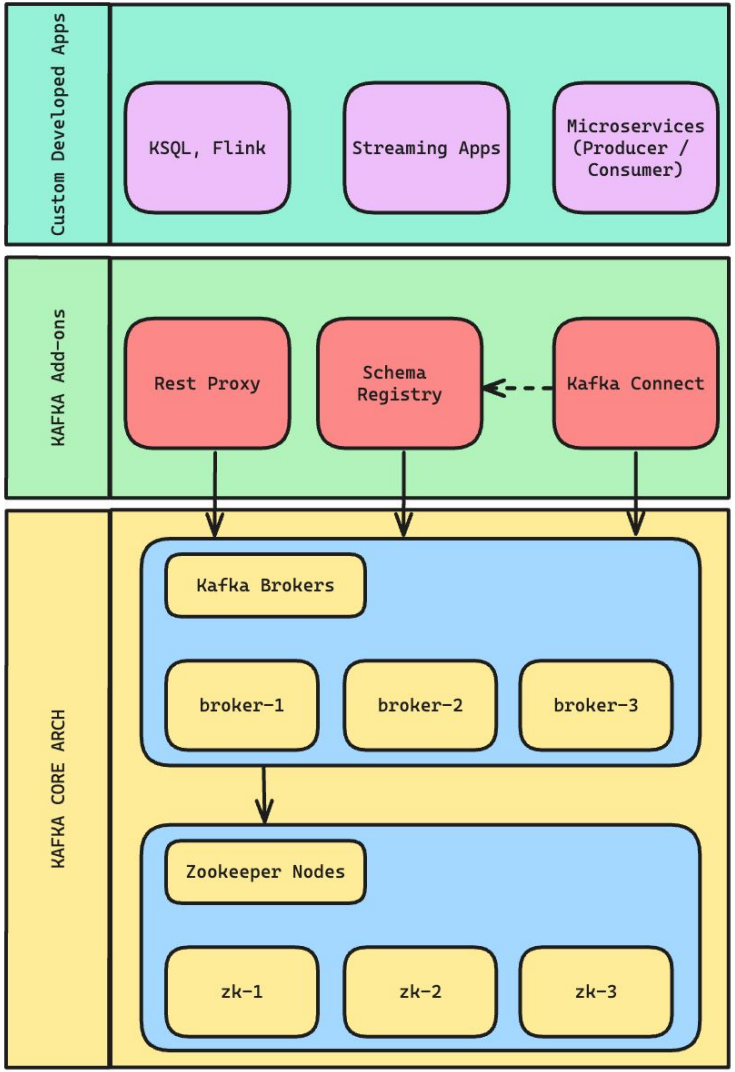
———

M@r921#5

Kafka Simple Architecture



Fundamentals - EcoSystem

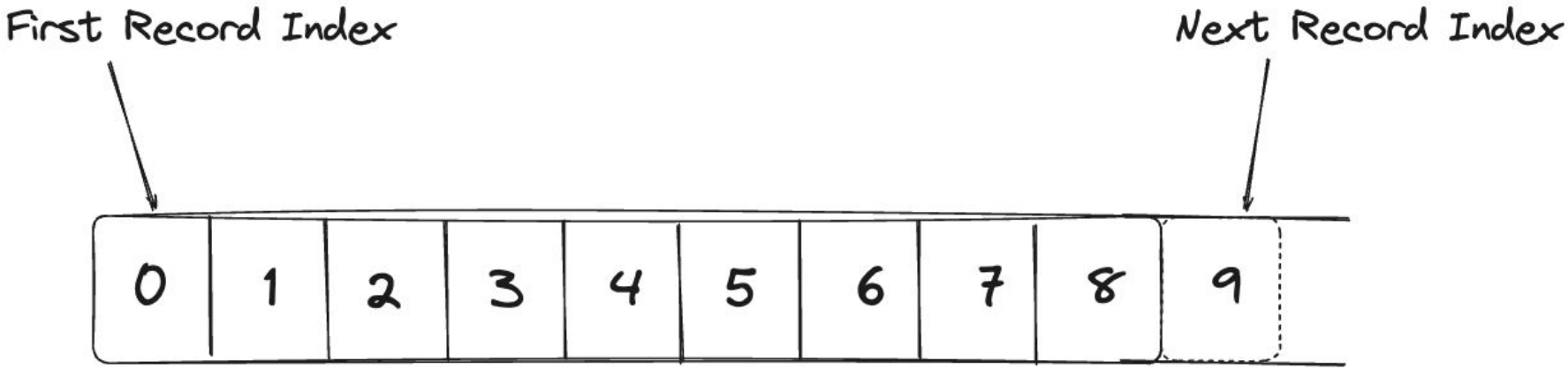


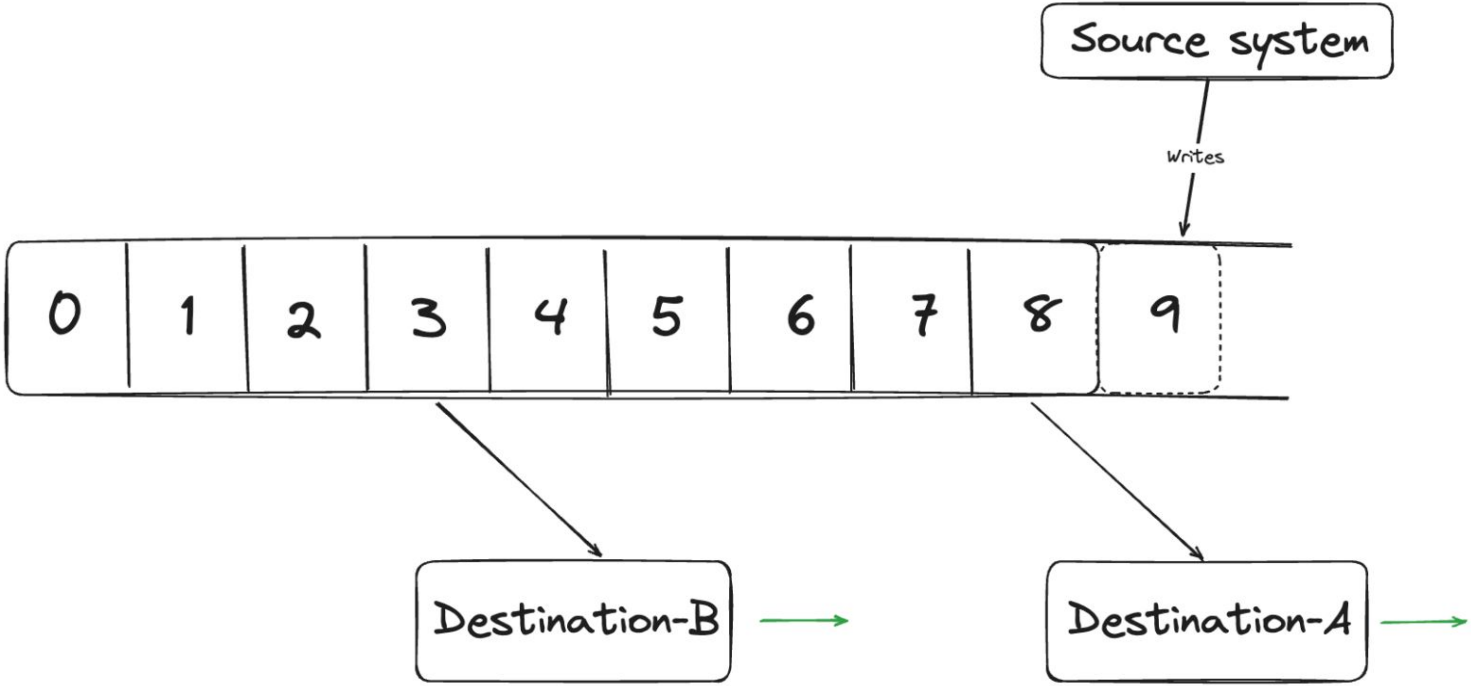
Fundamentals - Core Concepts

Broadly, Kafka accepts streams of **events** written by data **producers**. Kafka stores stores records chronologically in **partitions** across **brokers** (servers); multiples brokers comprise a **cluster**. Each record contains information about an event ands consists of a key-value pair; timestamp and header are optional additionalion. information. Kafka groups records into topics; data **consumers** get their data byo subscribing to the **topics** they want.

Fundamentals - What is a Log?

- Append Only
- Totally Ordered Sequence of Records
- Used to record what happened and When.

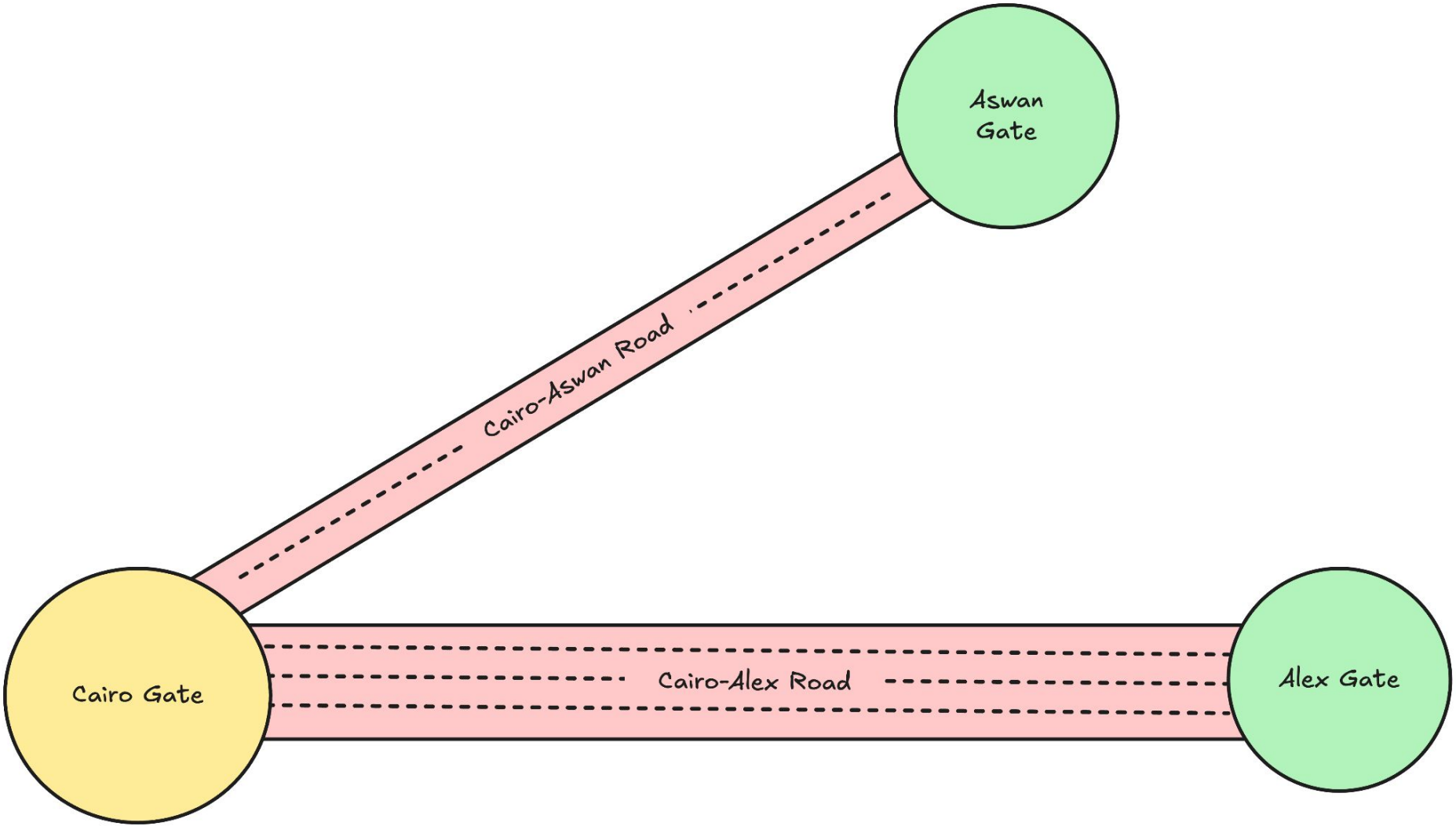


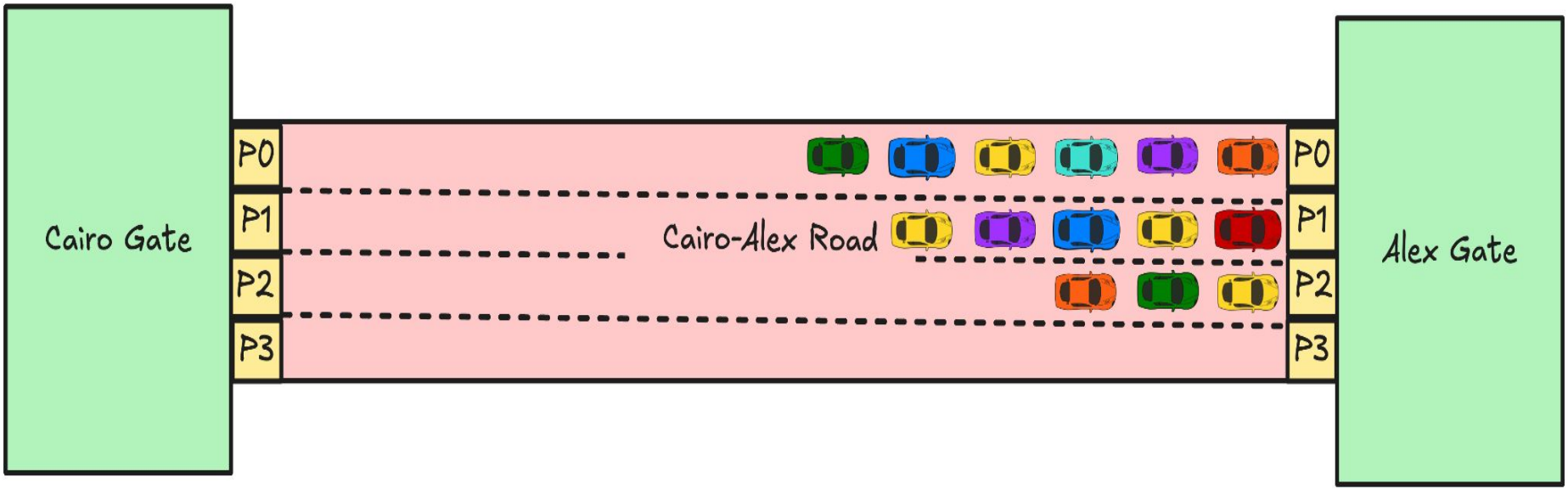

Source system
Writes
0
1
2
3
4
5
6
7
8
9
Destination-B
Destination-A

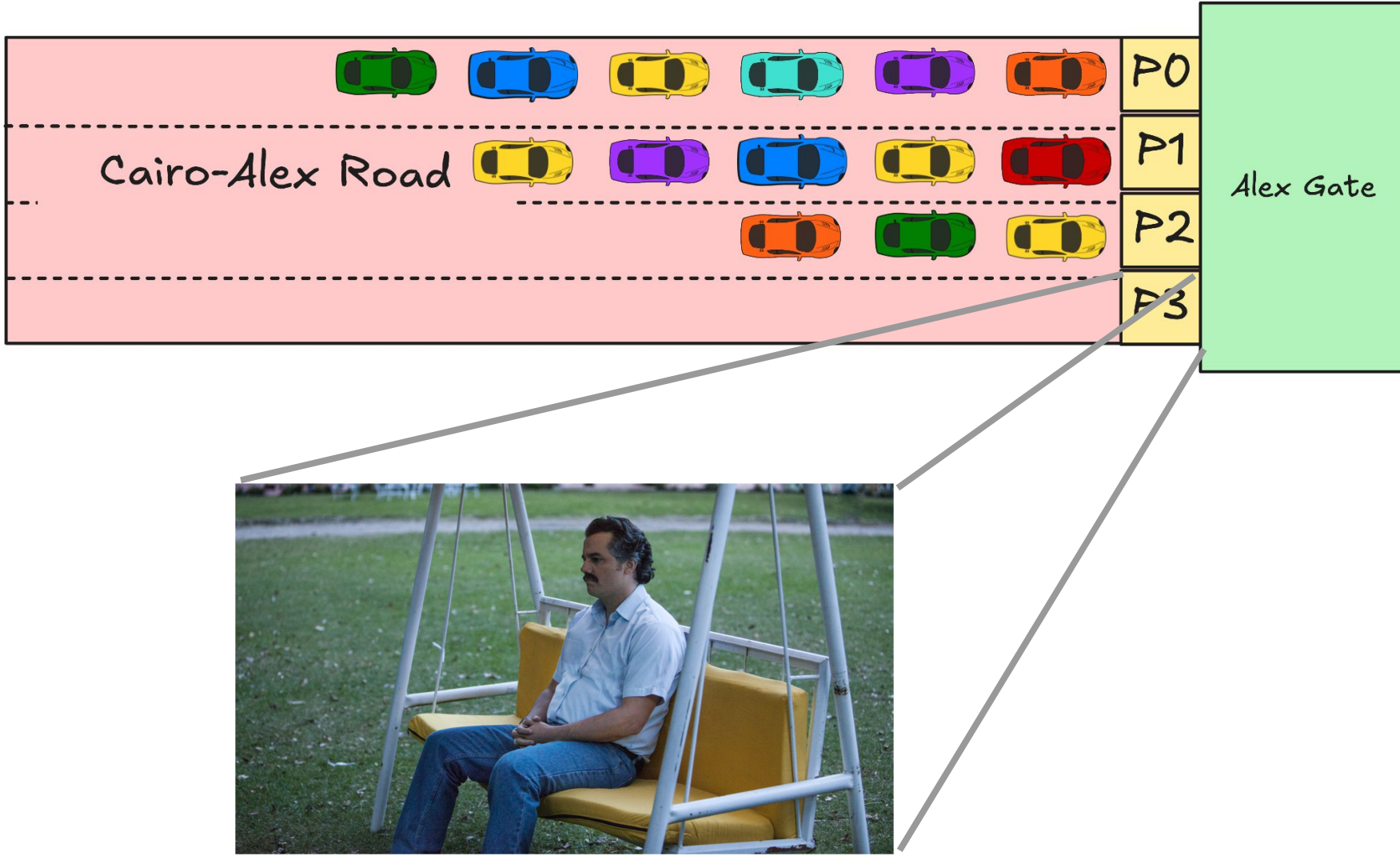
Fundamentals Cont'd - Topics

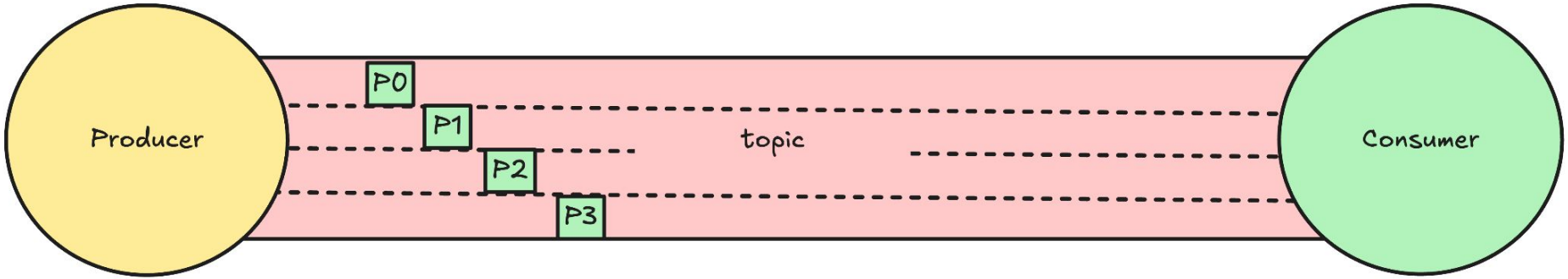
---

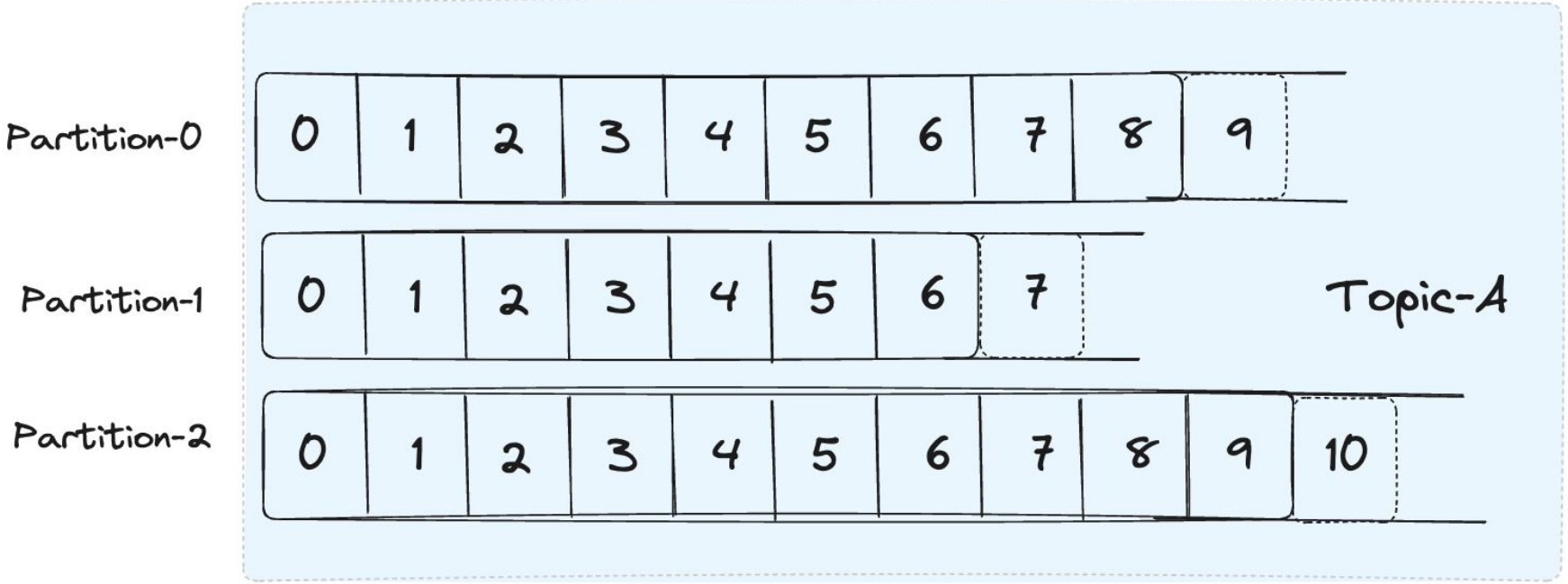
1. A collection of Messages categorized under a specific category or feed name (organize messages).
2. Topics are a build up of 1 or more **partitions**.
3. These partitioned are **Replicated** and **Ordered** Log.
4. Data Appended at the end of the topic partition
5. Topics' Partitions are stored **across different Nodes** (Kafka Brokers).


Aswan
Gate
Cairo-Aswan Road
Cairo Gate
Cairo-Alex Road
Alex Gate


Cairo Gate
P0
P1
P2
P3
Cairo-Alex Road
P0
P1
P2
P3
Alex Gate


Cairo-Alex Road
P0
P1
P2
P3
Alex Gate


Producer
P0
P1
P2
P3
topic
Consumer


Partition-0
0 1 2 3 4 5 6 7 8 9
Partition-1
0 1 2 3 4 5 6 7
Topic-A
Partition-2
0 1 2 3 4 5 6 7 8 9 10

Fundamentals Cont'd - What are Partitions?

---

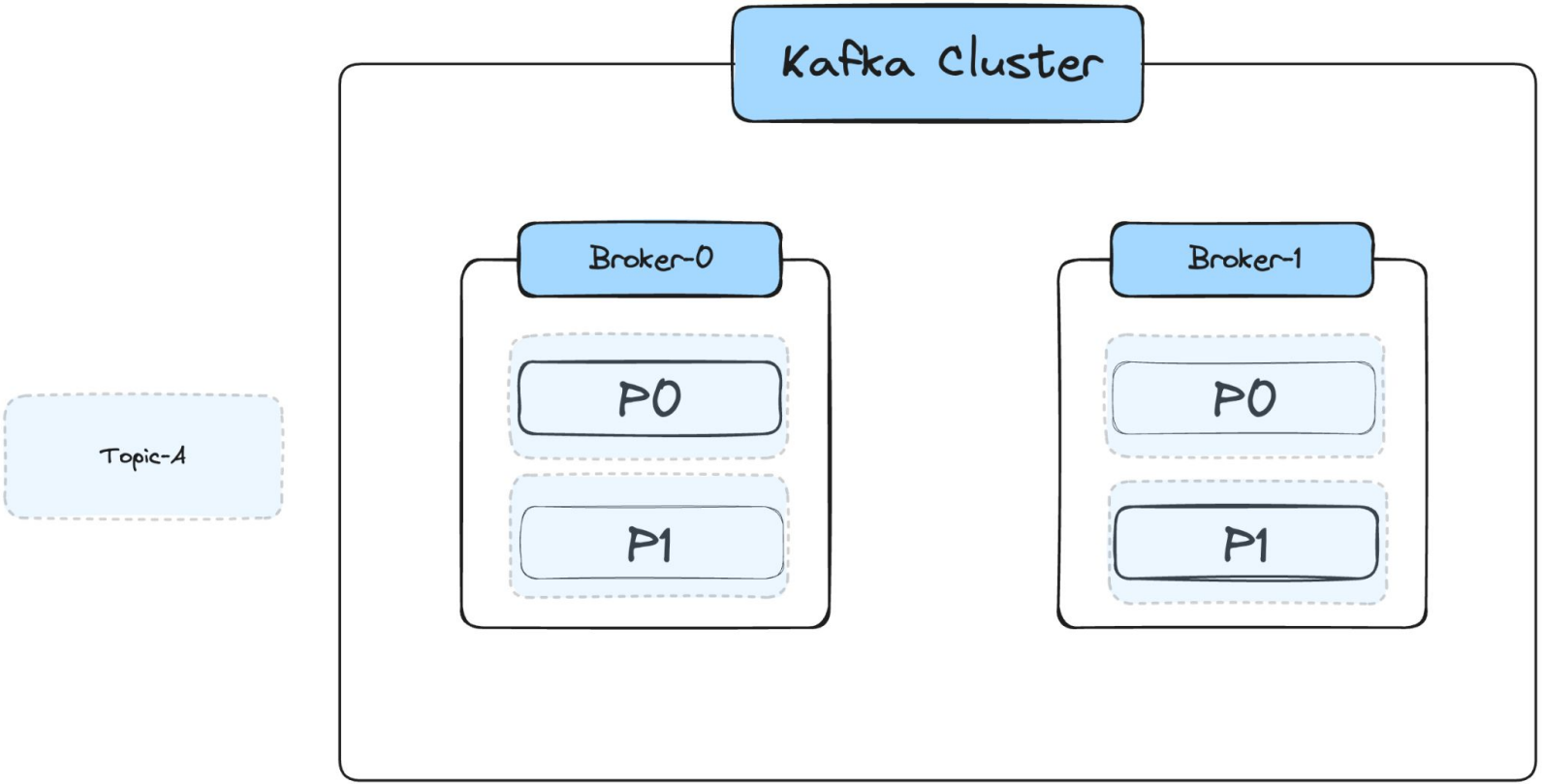
1. **Ordered, Append only**, and **immutable** sequence of messages.
2. Each Message is assigned a unique ID (**offset**).
3. **# of partitions** define the level of **parallelization**.
4. Partitions can be **replicated** across multiple server for high availability.
5. Partitions has 1 **leader** and other replicas act as **followers** for failover.

Fundamentals Cont’d - Why Partitions?

---

1. Scaling
2. Load Balancing
3. Semantic Partitioning
    a. By Key. (key Hash % # of partitions)
    b. Custom Partitioning (e.g. range partitioning).
    c. Round Robin

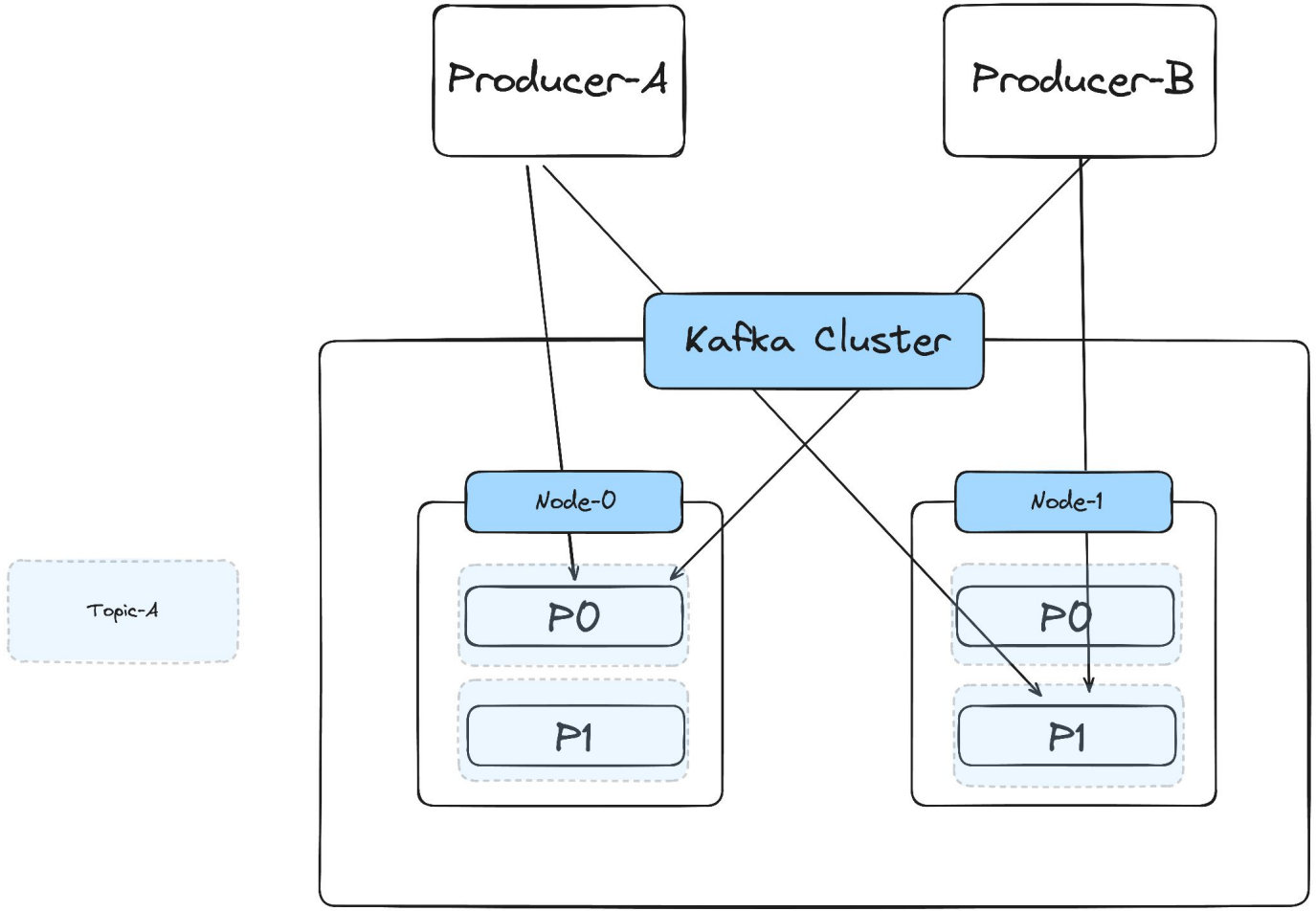
How Partitions Look like on Brokers?



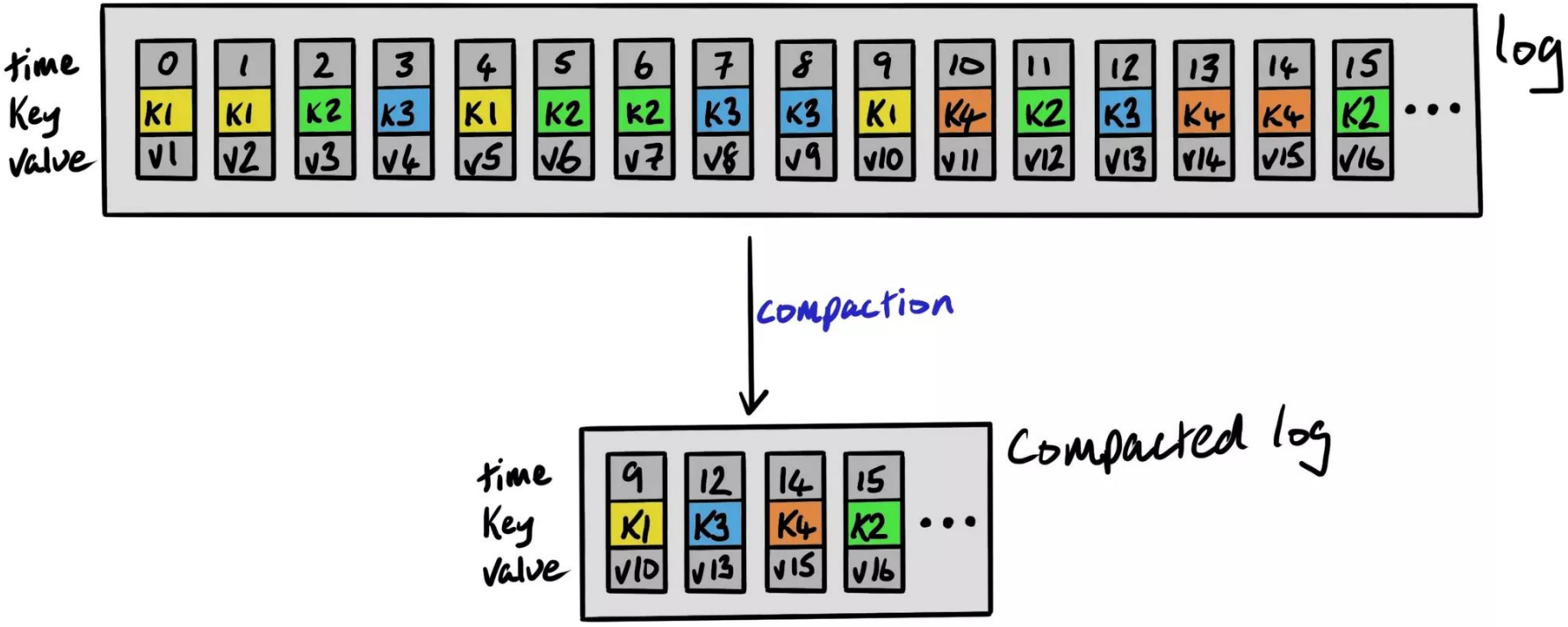
Fundamentals Cont'd (Core Concepts - Producers)

---

1. Send data to topic partition's **leader**
2. Messages can be load balanced (Key, RR)
3. Messages are batched (`**batch.size, linger.ms**`)
4. Sending is async
    a. Retrial can be managed via acknowledgment which can be configured by **acks=(0,1, all)**
5. Compaction
6. Transactions Management


Producer-A
Producer-B
Kafka Cluster
Node-0
Node-1
P0
P1
P0
P1
Topic-A

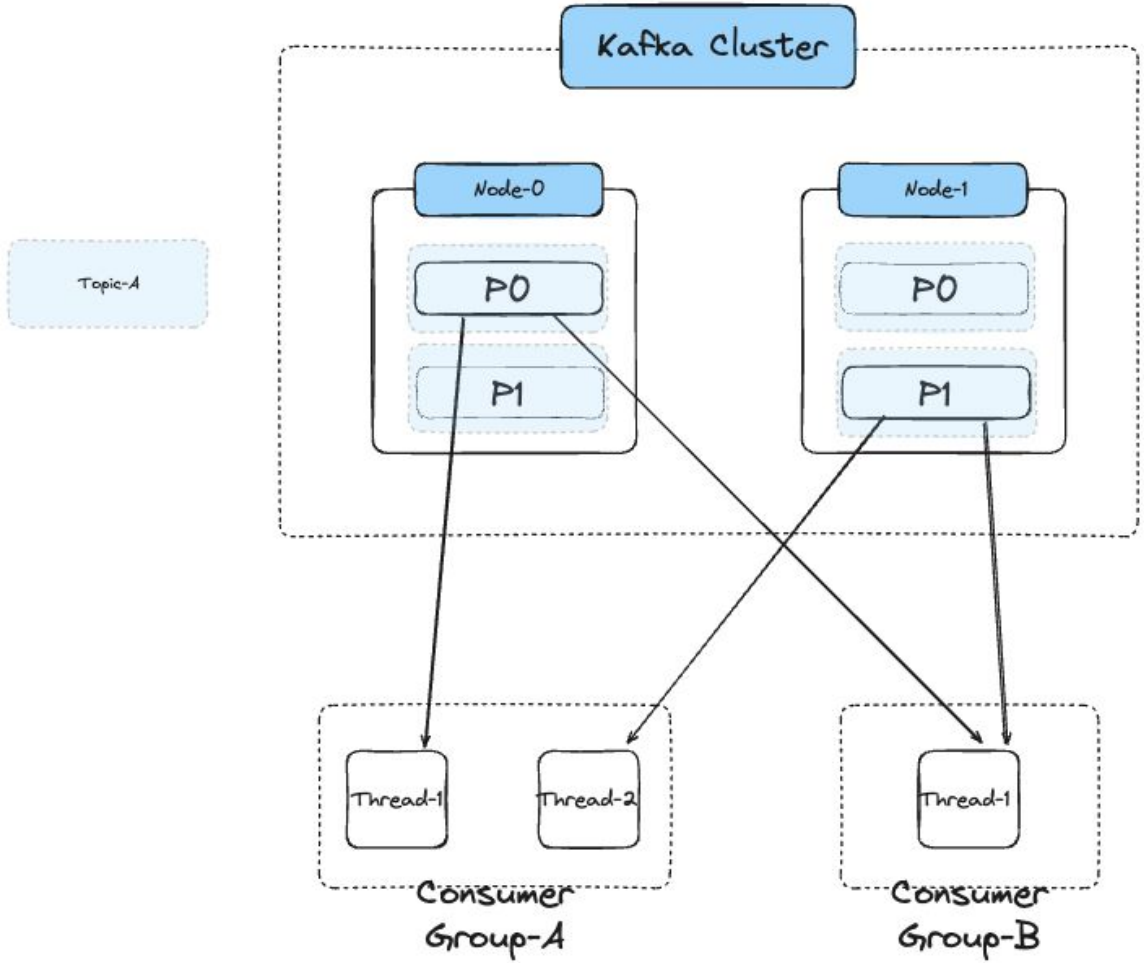
What is Log Compaction?



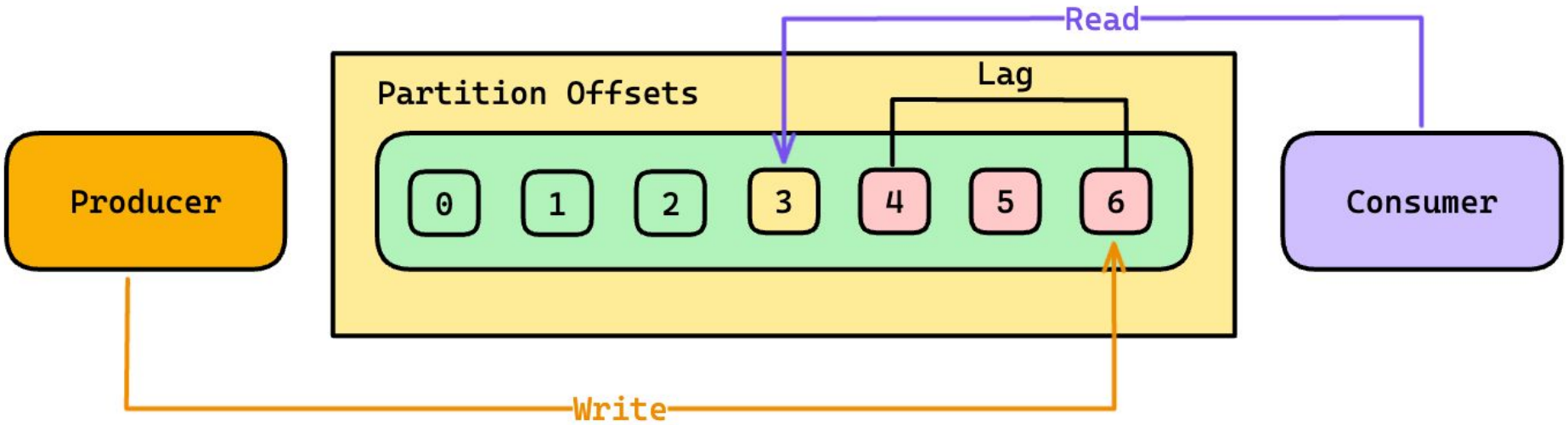
Fundamentals Cont'd - Consumers

---

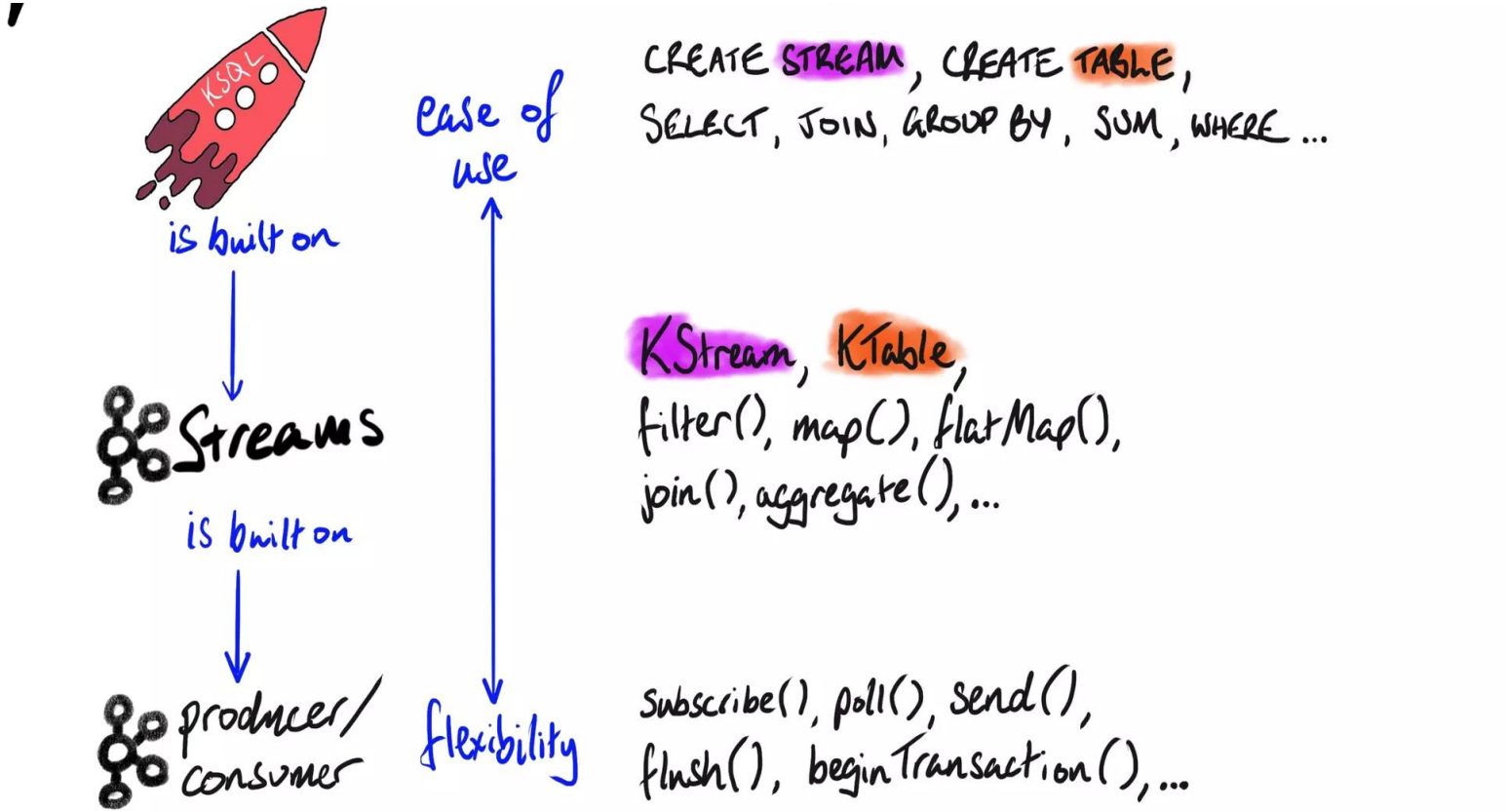
1. Consumers pull messages from Topics,
   a. **QQ.** why Pull, not Push?
2. Consumers assigned to a group, why?
3. Multiple consumer groups can read from the same topic, each on his own pace.
4. Consumer tracks its position (offset) under a topic called `__consumer_offsets`
5. When consumer is behind the latest offset, we call it "Lag"
6. Long Polling mechanism


Kafka Cluster
Node-0
Node-1
Topic-A
P0
P1
P0
P1
Thread-1
Thread-2
Thread-1
Consumer Group-A
Consumer Group-B

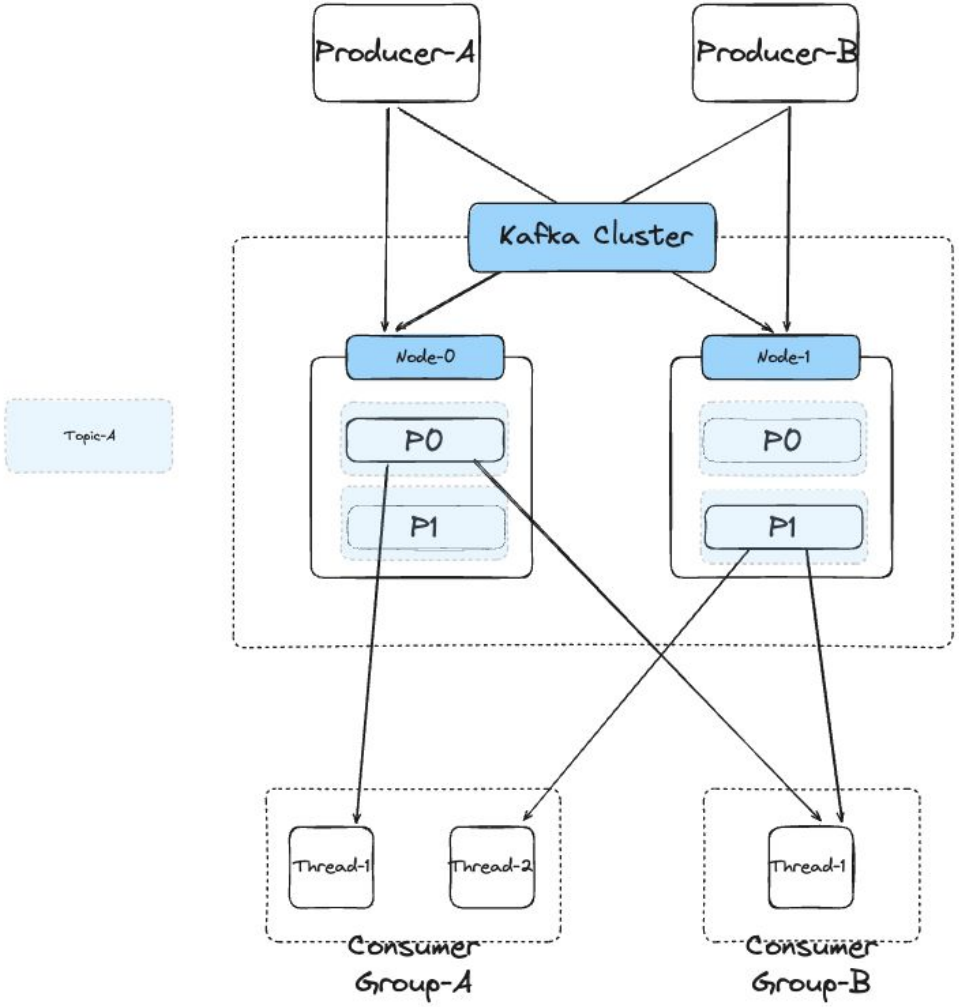
How Lag Looks Like?



What Kind of consumers in Kafka?



Fundamentals Cont’d - All Together



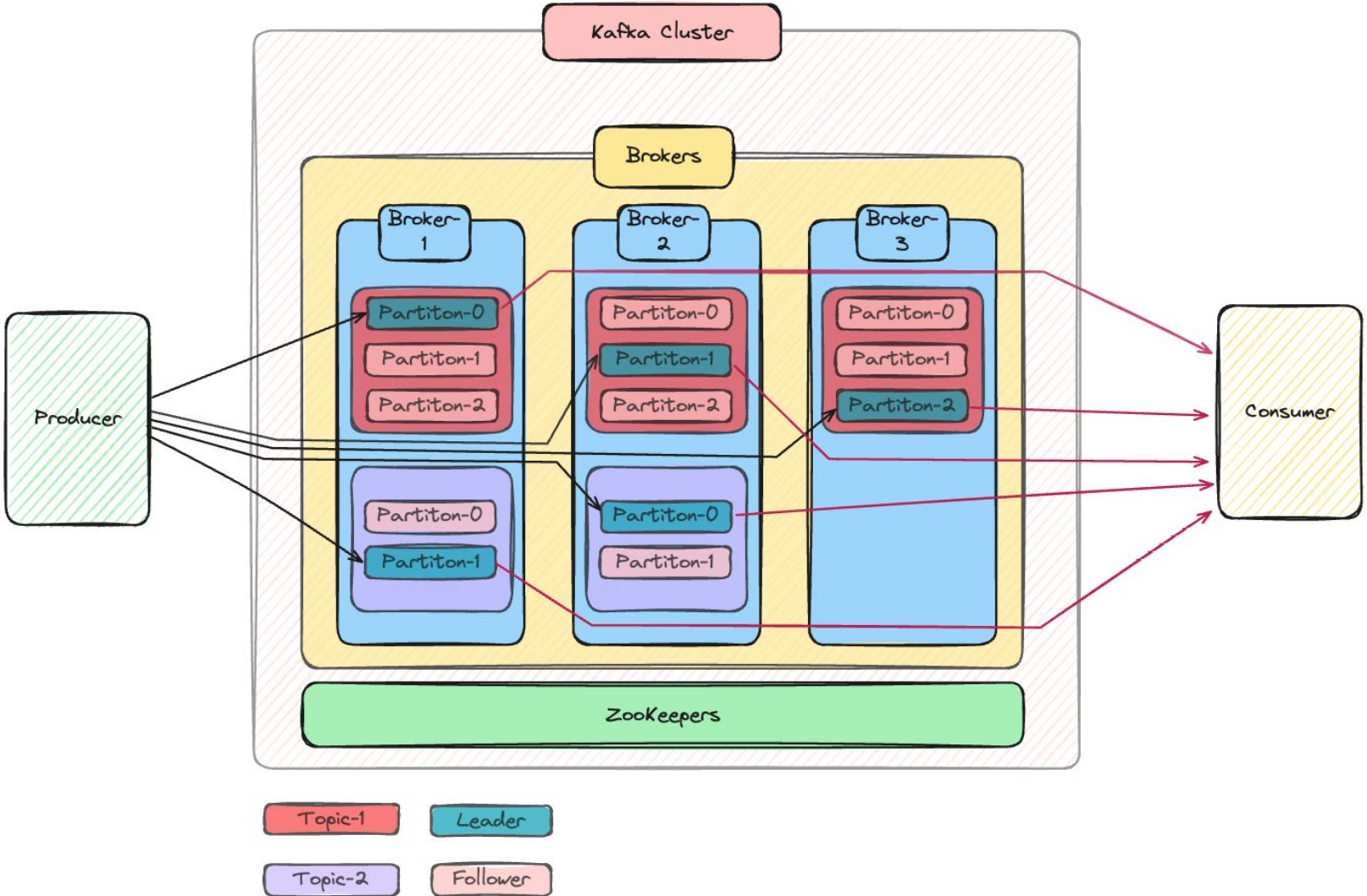
What we can Answer from the Diagram?

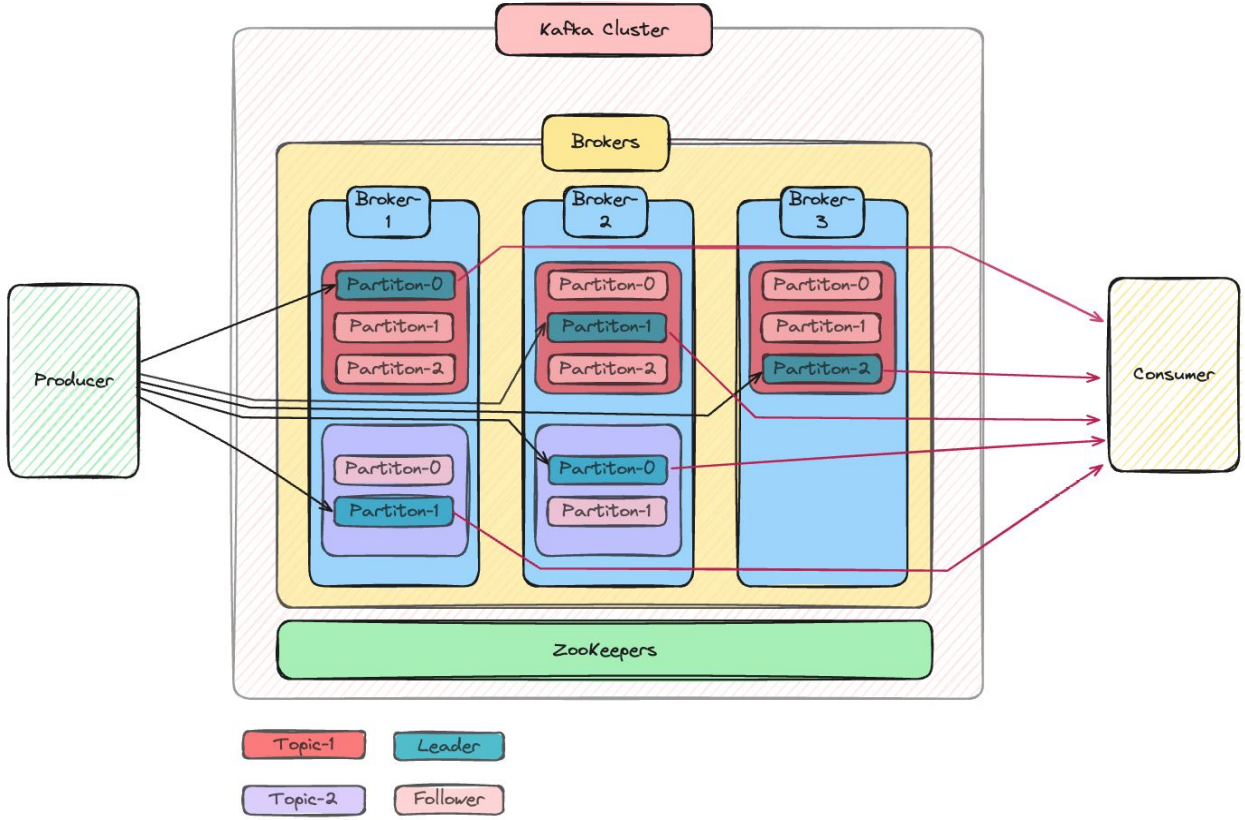
- How Many Producers?
- How Many Consumers Groups?
- How Many consumer thread under each group?
- How Many Topics?
- What is the Replication Factor?
- How Many Nodes in the cluster?

But, What we can't Answer???

- Topic Retention Period
- In Sync Replicas
- Partition Assignment Strategy

All Set Together


Kafka Cluster
Brokers
Broker-1
Broker-2
Broker-3
Partiton-0
Partiton-1
Partiton-2
Partiton-0
Partiton-1
Partiton-0
Partiton-1
Partiton-0
Partiton-1
Partiton-0
Partiton-1
Partiton-2
Producer
Consumer
ZooKeepers
Topic-1
Leader
Topic-2
Follower



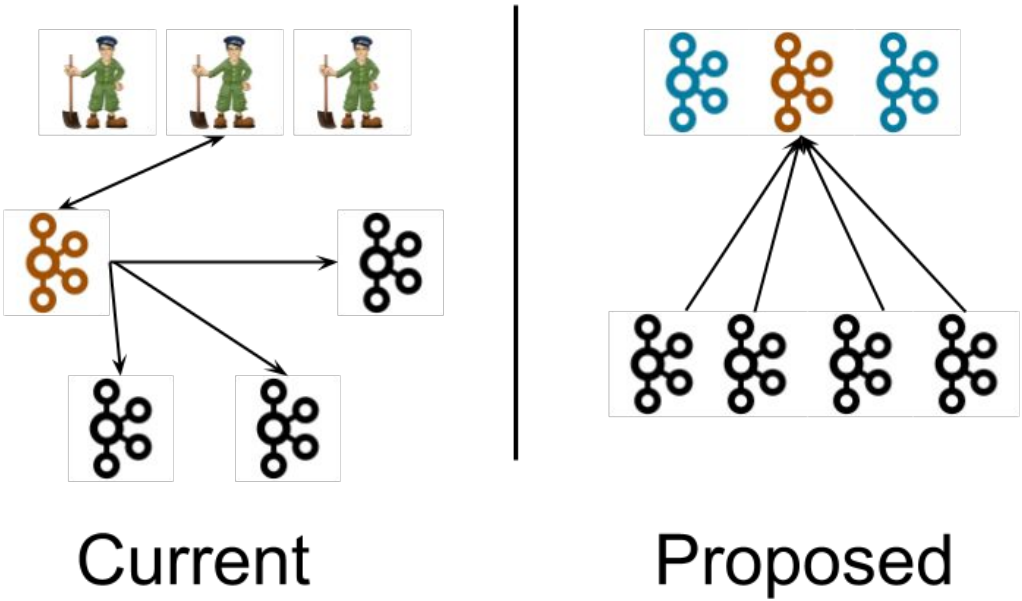
What can we answer from this Diagram?

- How Many Producers?
- How Many Consumers Groups?
- How Many consumer thread under each group?
- How Many Topics?
- What is the Replication Factor?
- What is the Max Rep Factor?
- How Many Nodes in the cluster?

Still, What we can't Answer is:

- Topic Retention Period
- In Sync Replicas
- Partition Assignment Strategy

Fundamentals Cont'd - Zookeepers



Advanced Topics

---

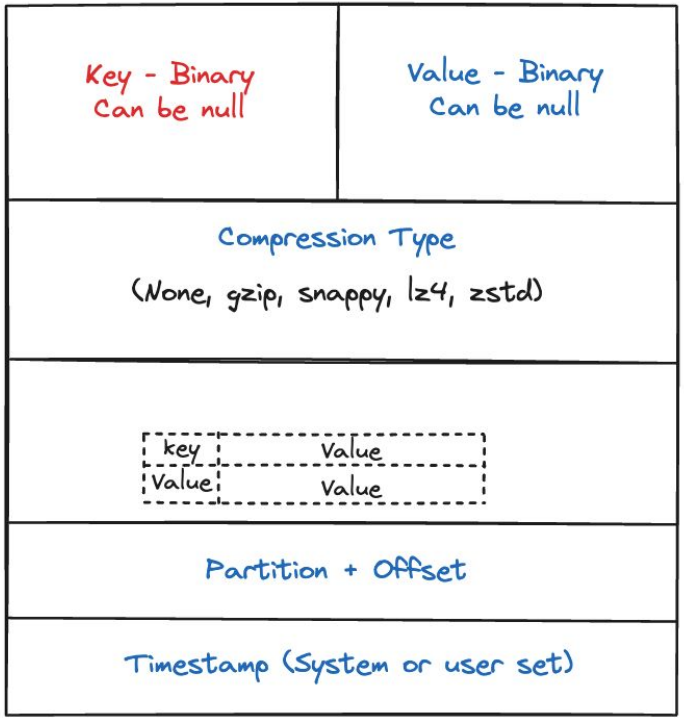
1. Messages
2. How Producer Works under the Hoods.
3. How Consumer Works under the Hoods.
4. Broker Important Configs.
5. Topic Important Configs.

Advanced Topics- Messages

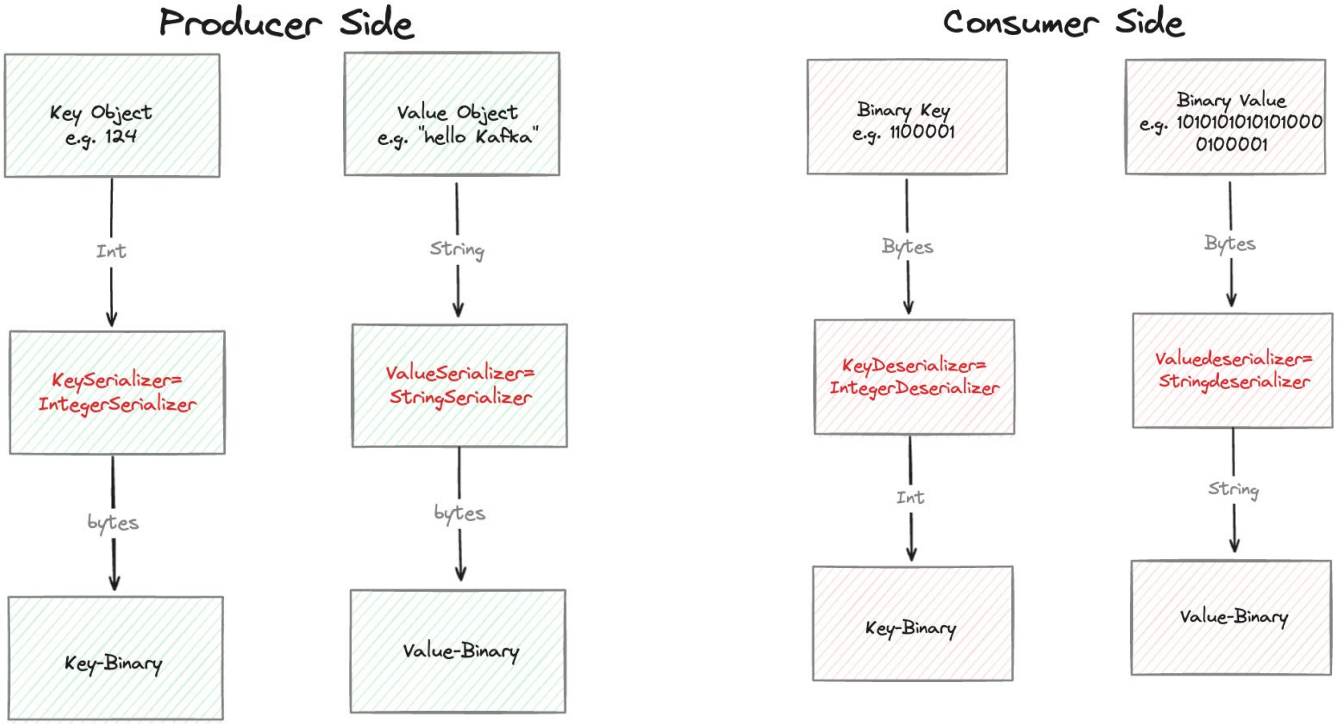
- Structure
- Serde

Advanced Topics - Message Structure

- **Key:** It is optional in the Kafka message and it can be null. A key may be a string, number, or any object and then the key is serialized into binary format.
- **Value:** Represents the content of the message and can also be null. The value format is arbitrary and is then also serialized into binary format.
- **Compression Type:** Kafka messages may be compressed. The compression type can be specified as part of the message. Options are `none`, `gzip`, `lz4`, `snappy`, and `zstd`
- **Headers:** There can be a list of optional Kafka message headers in the form of key-value pairs. It is common to add headers to specify metadata about the message, especially for tracing.
- **Partition + Offset:** Once a message is sent into a Kafka topic, it receives a partition number and an offset id. The combination of topic+partition+offset uniquely identifies the message
- **Timestamp:** A timestamp is added either by the user or the system in the message.



Advanced Topics - Message Serde

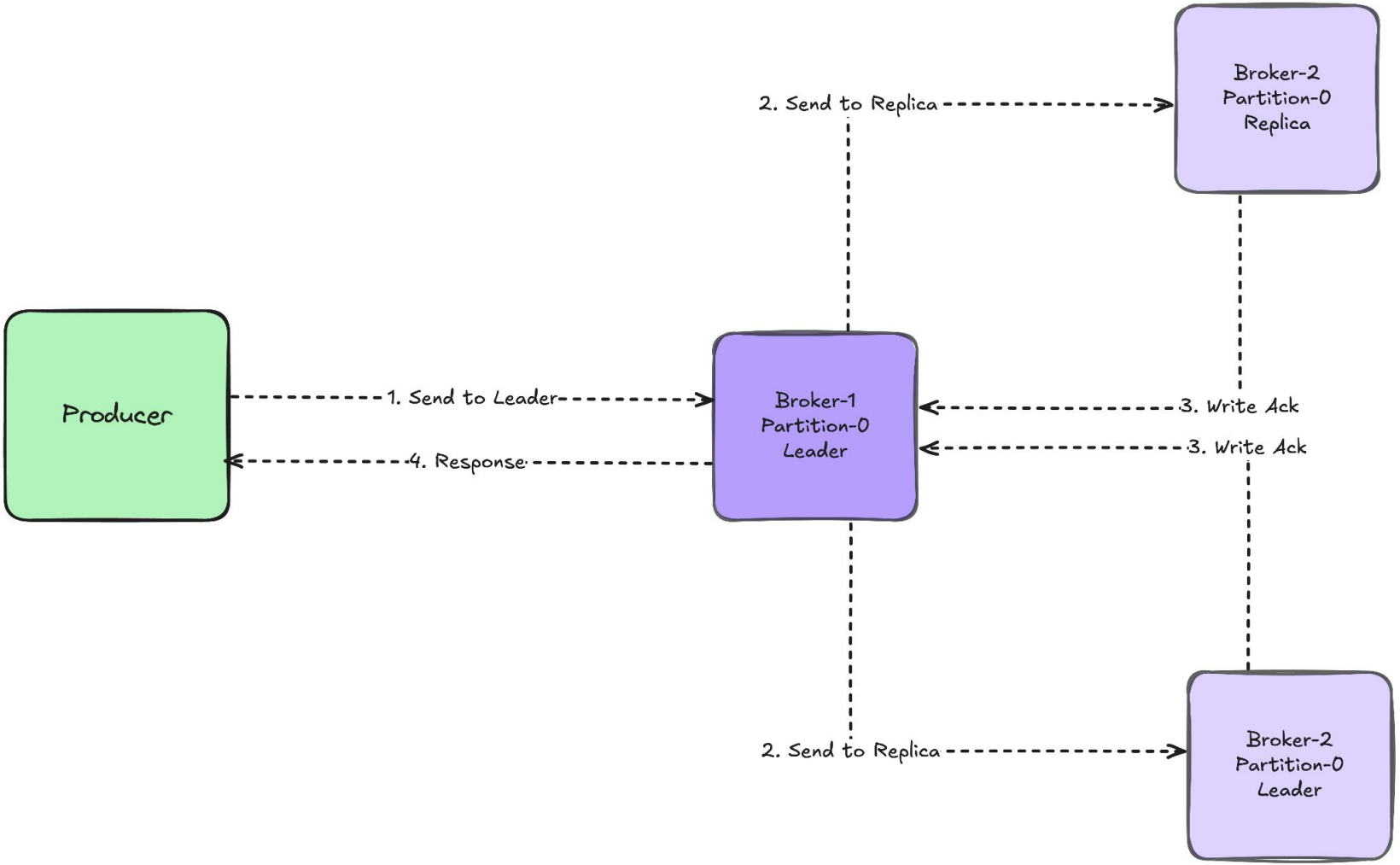


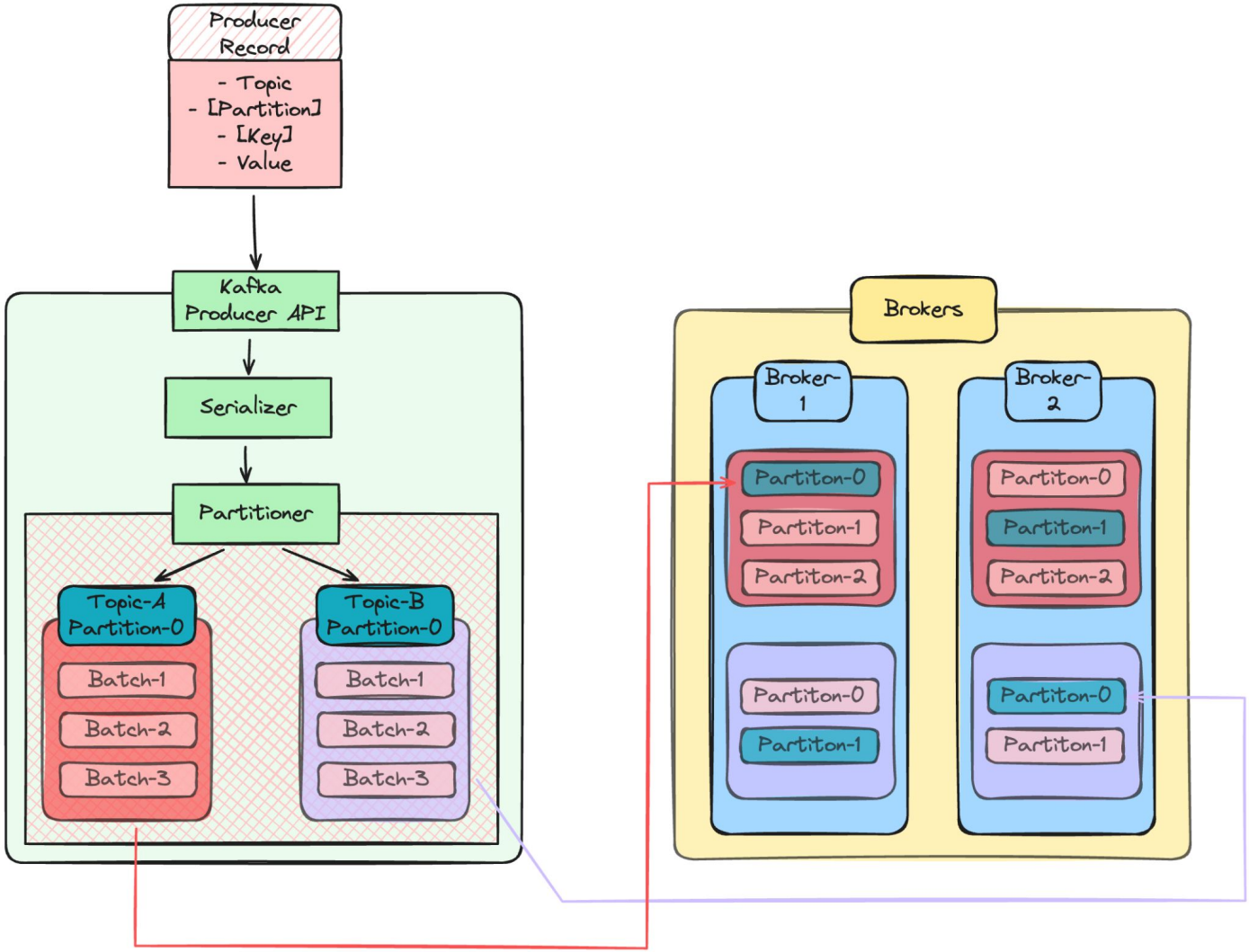
Advanced Topics- Producers

- How Producer Finds Partition Leader?
- Acks and retrials
- Idempotency
- Batching & Compression
- Partitioner (RR, Sticky Partitioner)
- Important Producer Configs

How Producer Find Partitions Leaders?

https://app.eraser.io/workspace/jydtaItgRgnAI81UFMiw


Broker-2
Partition-0
Replica
2. Send to Replica
Producer
1. Send to Leader
4. Response
Broker-1
Partition-0
Leader
3. Write Ack
3. Write Ack
2. Send to Replica
Broker-2
Partition-0
Leader


Producer Record
- Topic
- [Partition]
- [Key]
- Value
Kafka Producer API
Serializer
Partitioner
Topic-A Partition-0
Batch-1
Batch-2
Batch-3
Topic-B Partition-0
Batch-1
Batch-2
Batch-3
Brokers
Broker-1
Partiton-0
Partiton-1
Partiton-2
Partiton-0
Partiton-1
Broker-2
Partiton-0
Partiton-1
Partiton-2
Partiton-0
Partiton-1

Advanced Topics - Producer (Acks - Durability)

- Producer uses acknowledgments to controls messages durability
- For messages to be completed, Partition leader has to acknowledge the message delivery and being committed.
- This acknowledgement is being controlled by a producer config ‘acks’
- ‘Acks’ accepts these values (0, 1, all)
  - acks=0: Producer will not wait for any acknowledgment, Retries config will not take any effect, the offset given back to each record is set to -1. (No Durability Guarantee)
  - acks=1: This will mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers, In case the Leader has failed after acknowledging but before replicating to other replicas, then the record will be Lost (Less Durability Guarantee)
  - acks=all (-1): Leader will acknowledge only when data is committed in the local log and replicated in all in-sync-replicas. (Max Durability Guarantee)
- System with N-Replicas and M min-ISR can tolerate upto N-M broker outages.

Advanced Topics - Producer (Retries)

- There are 2 Categories of Errors:
  - Retriable: `NotEnoughReplicasException` in this case, producer can retry again, and maybe the replica broker will come back online and second attempt will succeed.
  - Non Retriable: `INVALID_CONFIG` exception.
- Producer retries are controlled via `retries` config
- Retry setting determines how many times the producer will attempt to send the message before marking as failed.
- The producer config value is `retries`, it is an int value accepts up to MAX_INT.
- Retries is not the only config used to control retries, but it is bounded by a time config which is `delivery.timeout.ms`.
  - For example if the `retries=2147483647` and `delivery.timeout.ms=120000` the producer won;t retry all this number of trials, however it is bounded by the `delivery.timeout.ms` (2 minutes), which after it will mark the message failed.
- There is another important metric to consider when handling retries `retry.backoff.ms=100`, which is the time the producer will wait between retries.

Retriable Vs Non-Retriable

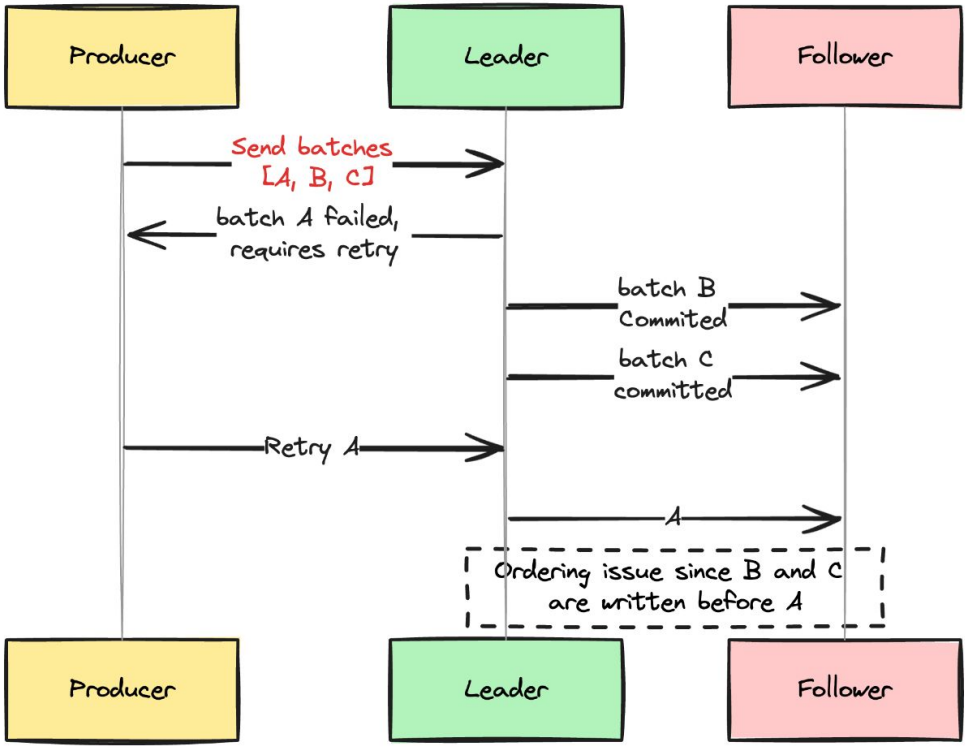
ERROR	CODE	RETRIABLE	DESCRIPTION
UNKNOWN_SERVER_ERROR	-1	False	The server experienced an unexpected error when processing the request.
NONE	0	False	
OFFSET_OUT_OF_RANGE	1	False	The requested offset is not within the range of offsets maintained by the server.
CORRUPT_MESSAGE	2	True	This message has failed its CRC checksum, exceeds the valid size, has a null key for a compacted topic, or is otherwise corrupt.
UNKNOWN_TOPIC_OR_PARTITION	3	True	This server does not host this topic-partition.
INVALID_FETCH_SIZE	4	False	The requested fetch size is invalid.
LEADER_NOT_AVAILABLE	5	True	There is no leader for this topic-partition as we are in the middle of a leadership election.
NOT_LEADER_OR_FOLLOWER	6	True	For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this error indicates that the broker is not a replica of the topic partition.
REQUEST_TIMED_OUT	7	True	The request timed out.
BROKER_NOT_AVAILABLE	8	False	The broker is not available.
REPLICA_NOT_AVAILABLE	9	True	The replica is not available for the requested topic-partition. Produce/Fetch requests and other requests intended only for the leader or follower return NOT_LEADER_OR_FOLLOWER if

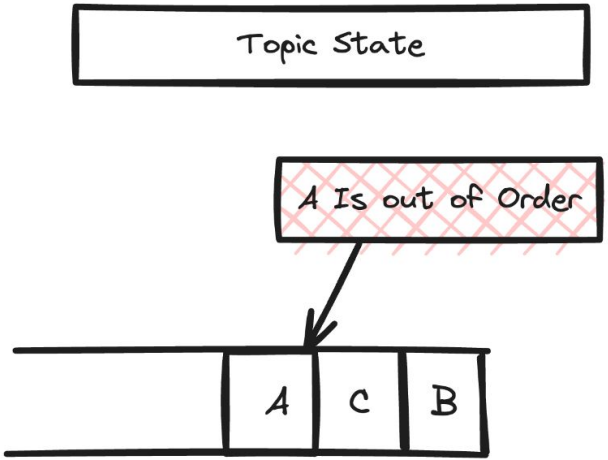
Advanced Topics - Producer (Retries - Ordering)

- There is an important config that we need to considered when using retries while need to keep ordering guaranteed
- `**max.in.flight.requests.per.connection=5**` is a producer config value that controls the number of concurrent requests can be sent from the producer.
- Having this **value > 1**, means that there will be **N** requests sent to the broker in a batch.
- The broker process the messages one by one, if (for example) the first message failed and others were successful, then the broker will report that the first msg has failed, hence retrial from the producer.
- If the producer retried the message and it got committed successfully the data will be out of order.

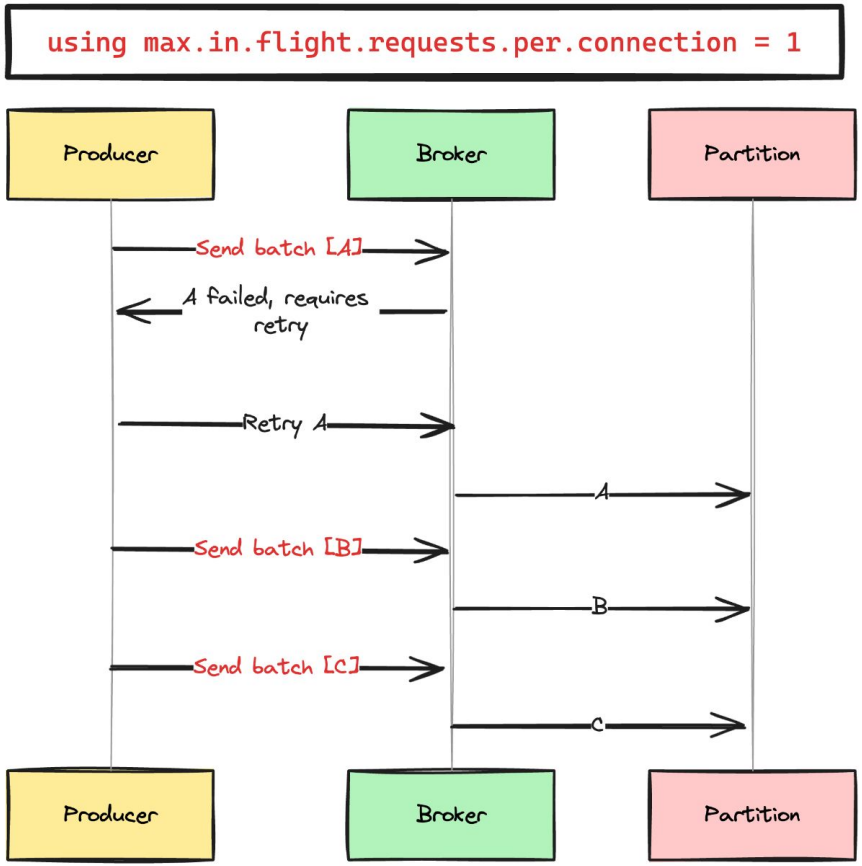
Advanced Topics - Producer (Retries - Ordering)

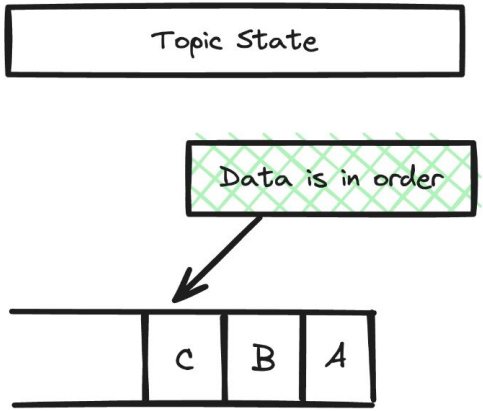
using max.in.flight.requests.per.connection > 1





Advanced Topics - Producer (Retries - Ordering)



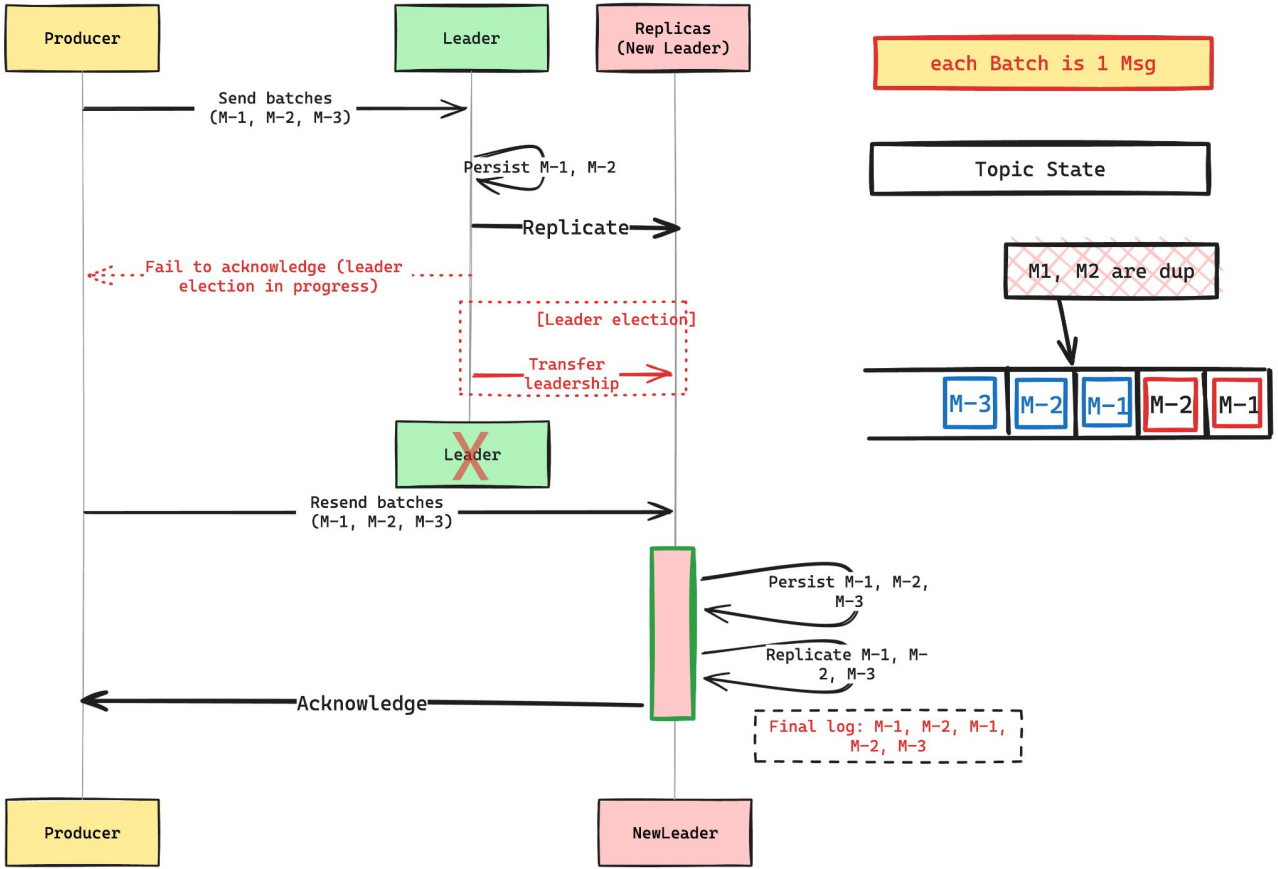


Advanced Topics - Producer (Idempotency)

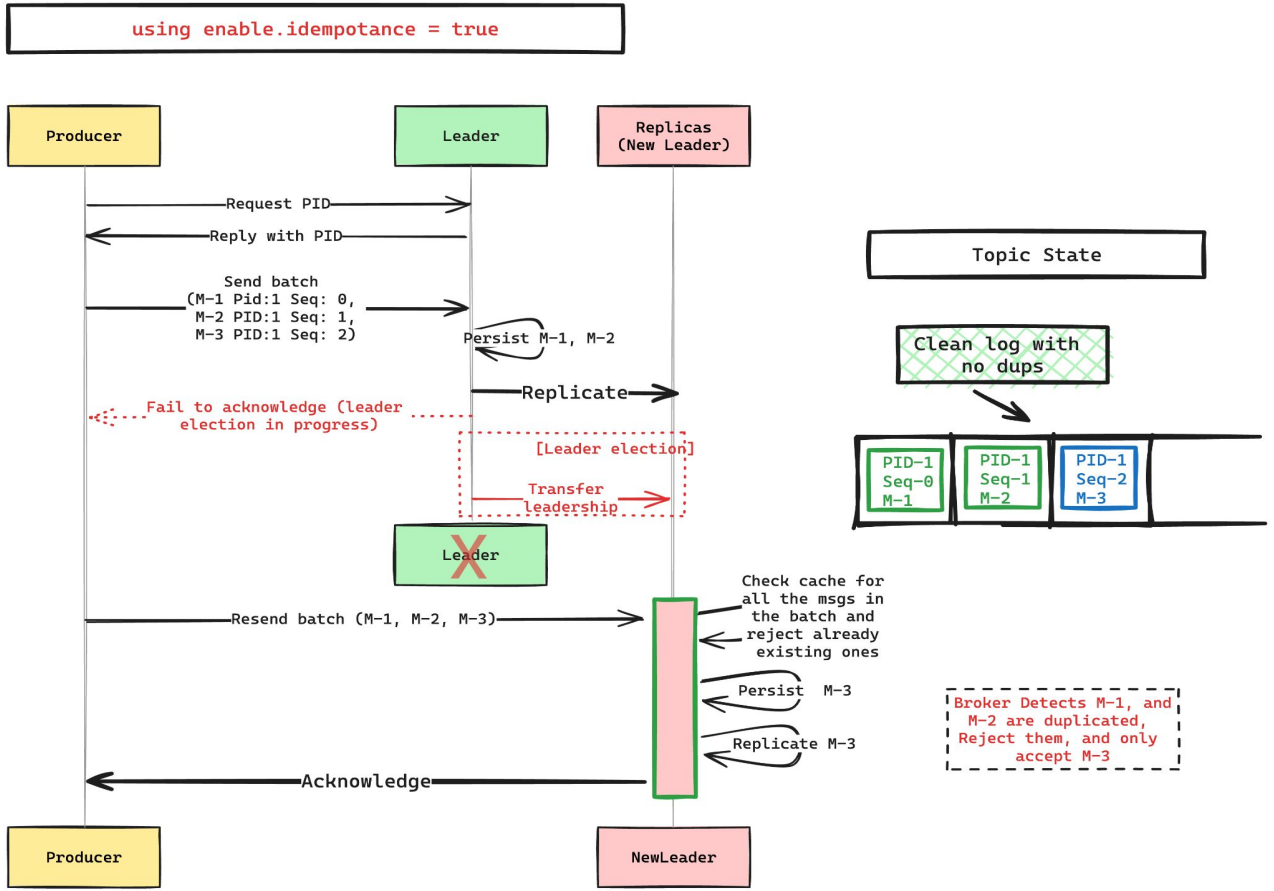
---

1. Producer idempotence ensures that duplicates are not introduced due to unexpected retries.
2. How can retries introduce duplicate???

Advanced Topics - Producer (Retries - Duplicate)



Advanced Topics - Producer (Retries - Dup Solution)



Advanced Topics - Producer (Idempotency)

- How Idempotent producer works under the hood?
  a. When enable.idempotence is set to true, each producer gets assigned a Producer Id (PID).
  b. **Coordinator Broker** is the one generates the PID (Typically the broker handling Metadata request).
  c. **The PID is included every time** a producer sends messages to a broker.
  d. Each message gets a monotonically increasing sequence number (different from the offset - used only for protocol purposes).
  e. **A separate sequence** is maintained for **each topic partition** that a producer sends messages to.
  f. On the broker side, on a per partition basis, it keeps track of the largest PID-Sequence Number combination that is successfully written.
  g. When a lower sequence number is received, it is discarded.

Advanced Topics - Producer (Idempotency)

- PID get changes in multiple scenarios:
  a. Producer restart
  b. Brokers rebalance (in very rare cases).
  c. Transaction timeout (transaction.timeout.ms)

Producer Side

PID	Sequence Num	Message	Status
123	0	msg-1	Ack
123	1	msg-2	Ack
123	2	msg-3	Failed / Retry

Broker Side

Partition	PID	Sequence Num	Message
orders-0	123	0	msg-1
orders-0	123	1	msg-2
orders-0	123	2	msg-3

Advanced Topics - Producer (Delivery Semantics)

- At Most Once:
  - acks=0
- At Least Once:
  - acks=1
- Exactly Once:
  - acks=all
  - idempotence=true
  - max.inflight<=5
  - Retries > 0

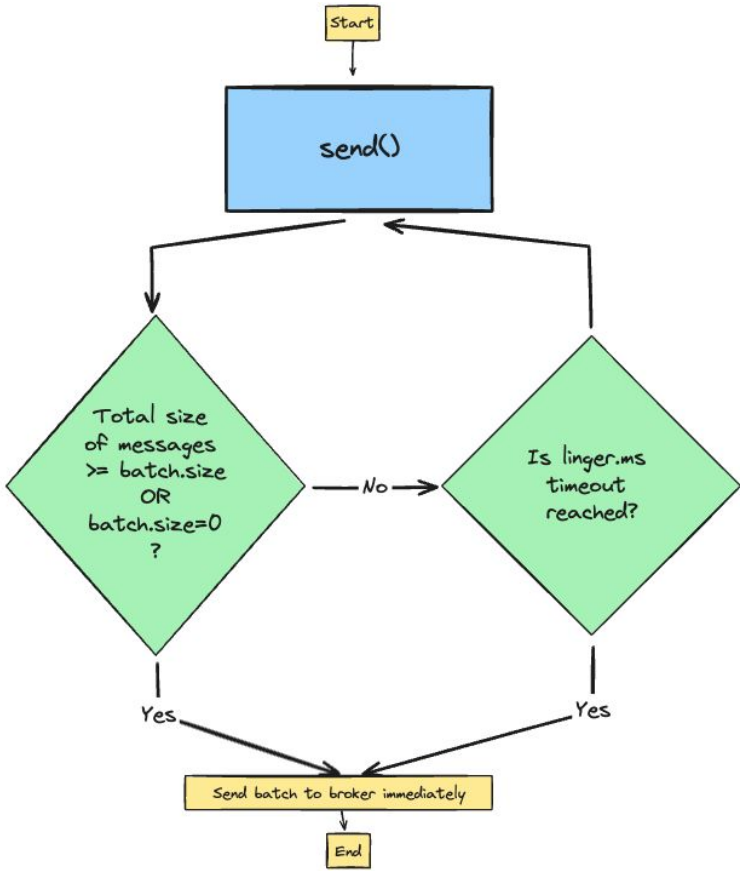
Advanced Topics - Producer (Why Max.in.flight<=5)

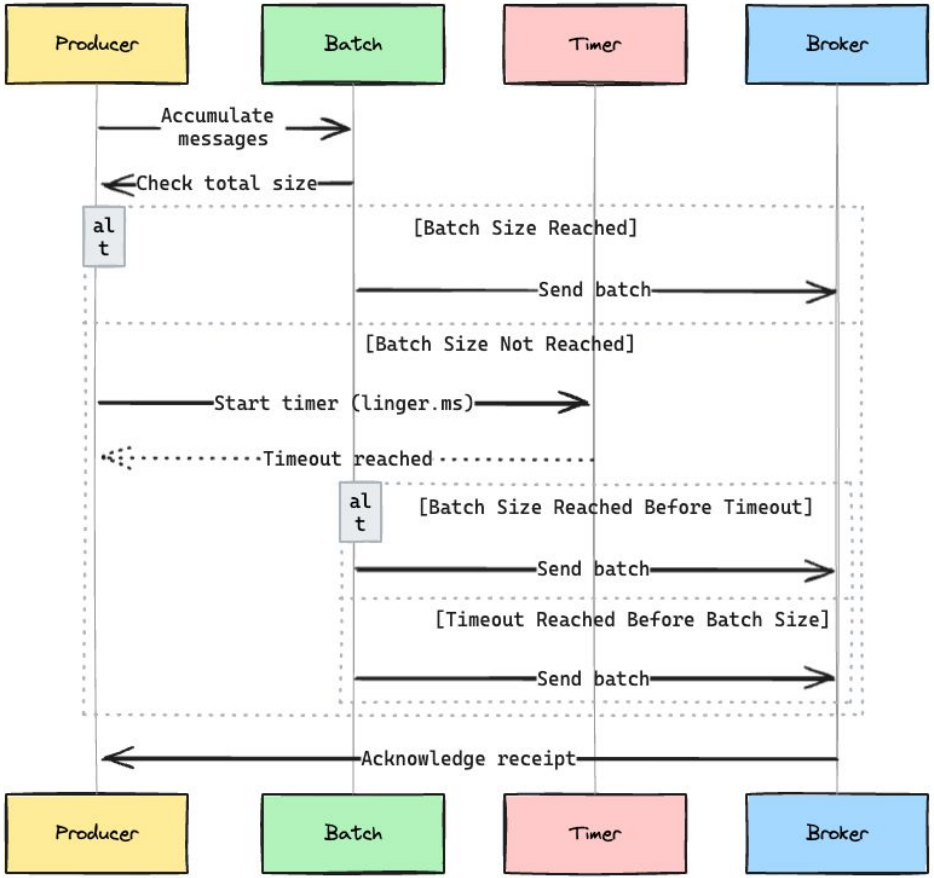
- Balance between throughput and reliability.
- Leader keeps updating the five last sequence IDs every time a new message is produced. However if more than five messages are inflight, it is possible that messages are duplicated because their sequence ID ran out of cache.

Advanced Topics - Producer (Batching & Compression)

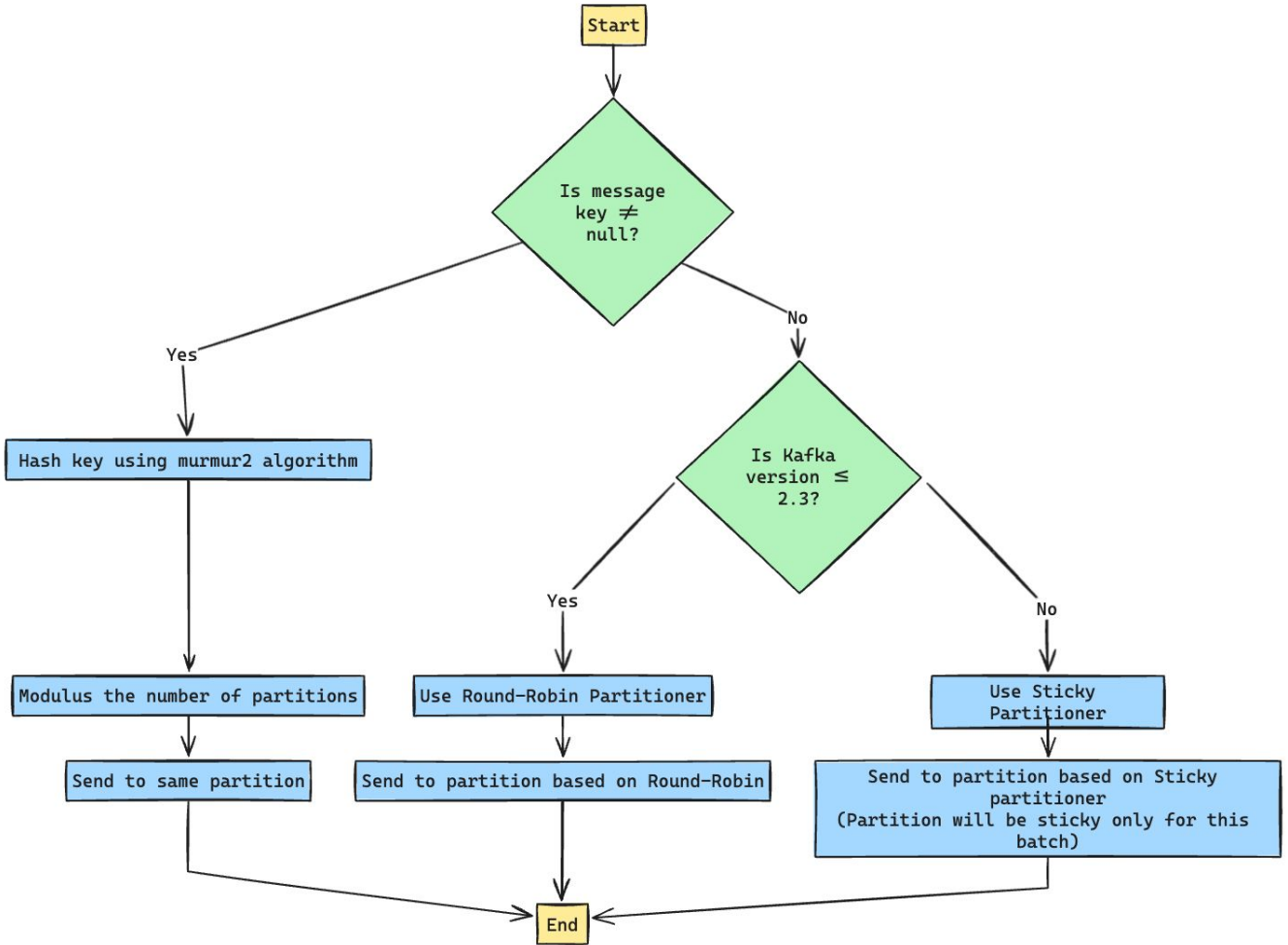
- Batching in Kafka Producer refers to the process of grouping multiple messages together into a single batch before sending them to a Kafka broker.
- This process improves throughput and reduces the overhead associated with network communication, as fewer requests are sent over the network.
- Batching is allocated per partition.
- There are 2 main configs controlling Batching:
  - `batch.size`: 16384 (16KB)
  - `linger.ms`: 0
  - Performance tip:
    - In case you have a high traffic, it is recommended to increase the `linger.ms=50` and `batch.size=32K` this can help in increasing compression ratio and throughput.
- And one config manages compression:
  - `compression.type`: producer [zstd, lz4, snappy, gzip]

Advanced Topics - Producer (Batching)

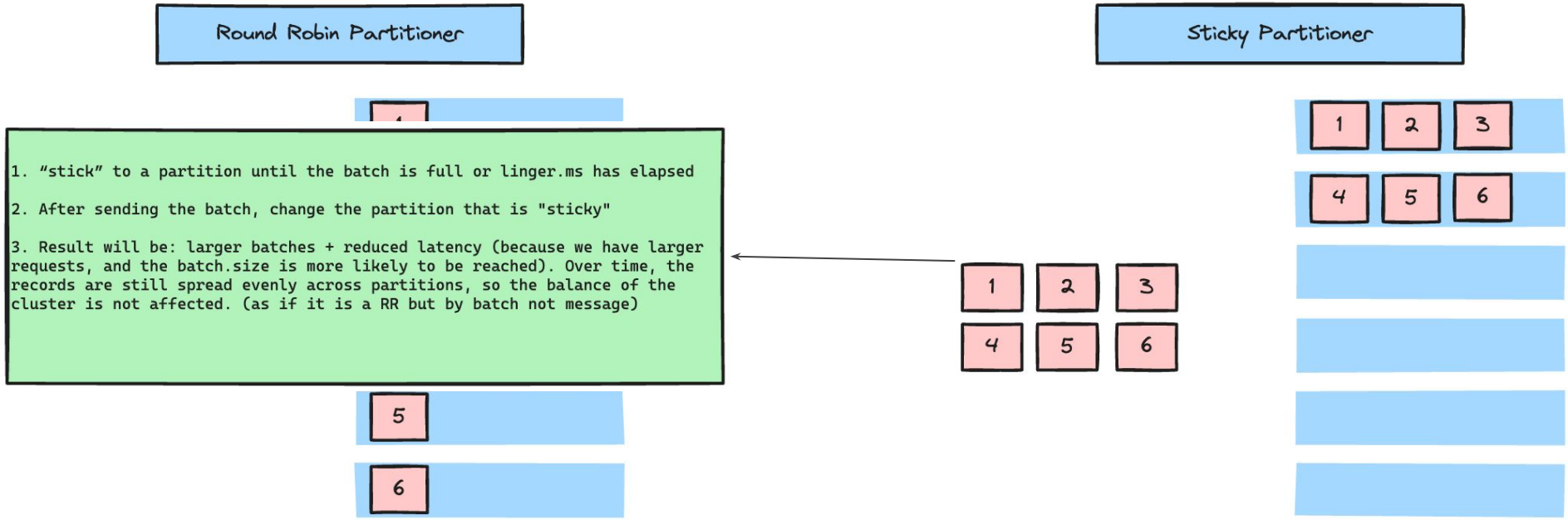




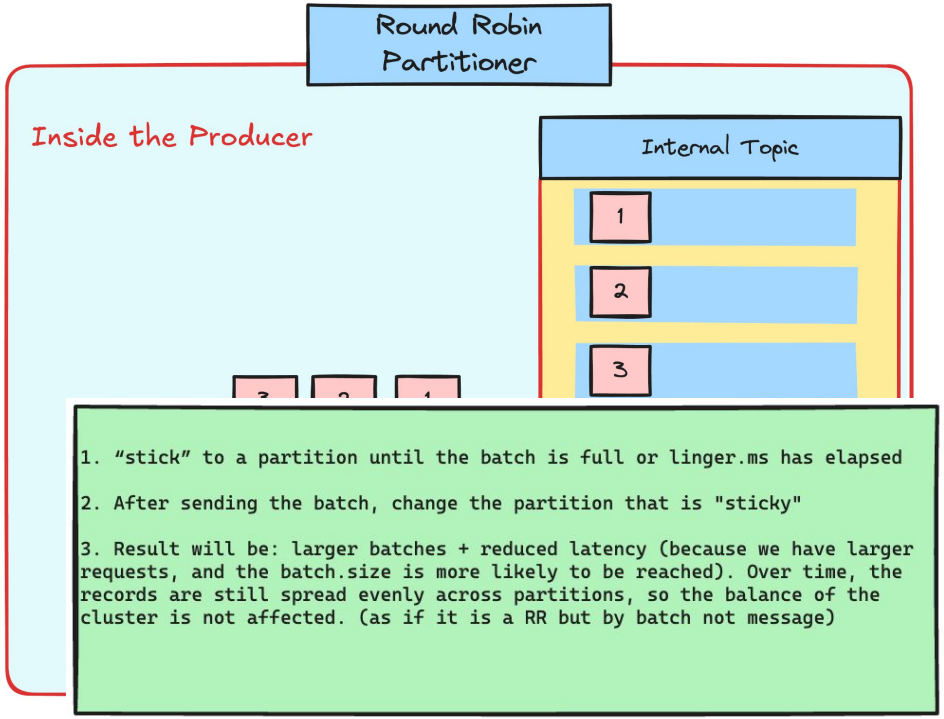
Advanced Topics - Producer (Partitioner)

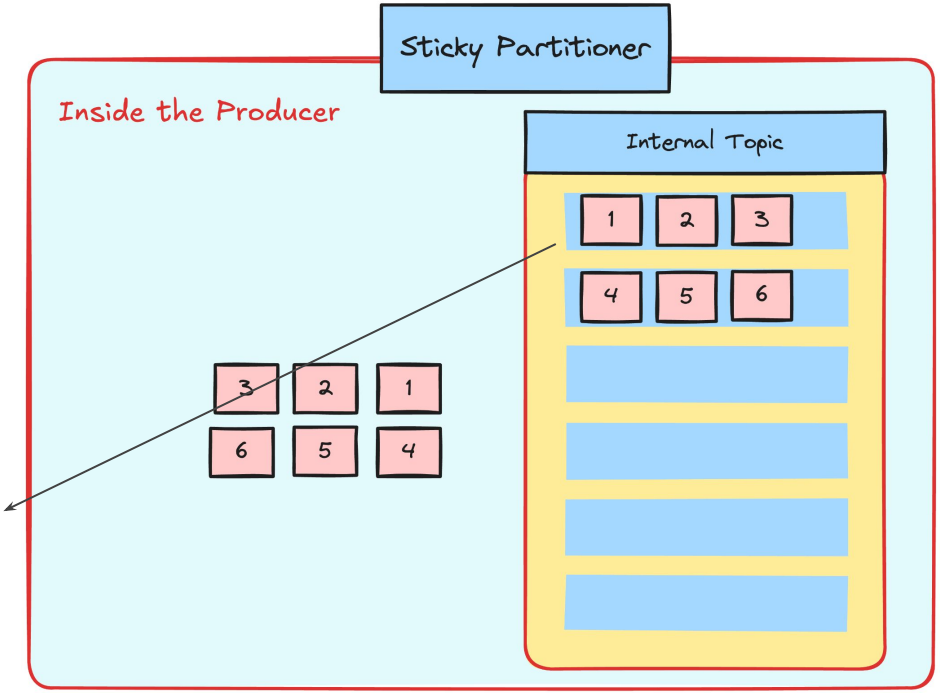


Advanced Topics - Producer (Partitioner)



Advanced Topics - Producer (Partitioner)

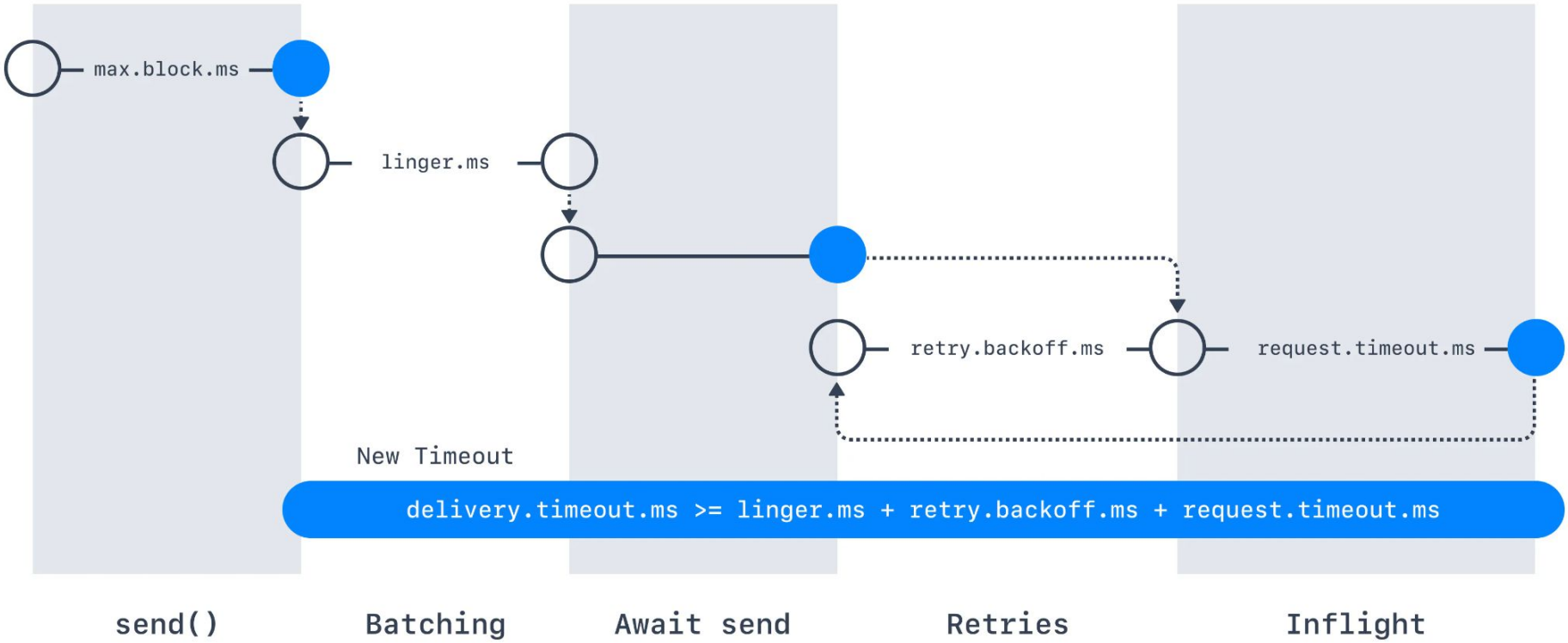




Advanced Topics - Producer (Recommended Configs)

- The recommended configs for exactly one delivery semantics + while preserving the messages order is as following:

```
acks=all (default)
retries=MAX_INT (default)
delivery.timeout.ms=120000 (default)
retry.backoff.ms=100 (default)
max.in.flight.requests.per.connection=5 (default)
enable.idempotence=true (default)
batch.size=16384 (default)
linger.ms=0 (default)
```


max.block.ms
linger.ms
retry.backoff.ms
request.timeout.ms
New Timeout
delivery.timeout.ms >= linger.ms + retry.backoff.ms + request.timeout.ms
send()
Batching
Await send
Retries
Inflight

Advanced Topics - Producer (Other Configs)

```
bootstrap.servers ⇒ must be set
key.serializer
value.serializer ⇒ recommended to be set
max.request.size
partitioner.class ⇒ In case of custom partitioner is needed
request.timeout.ms
buffer.memory
max.block.ms
```

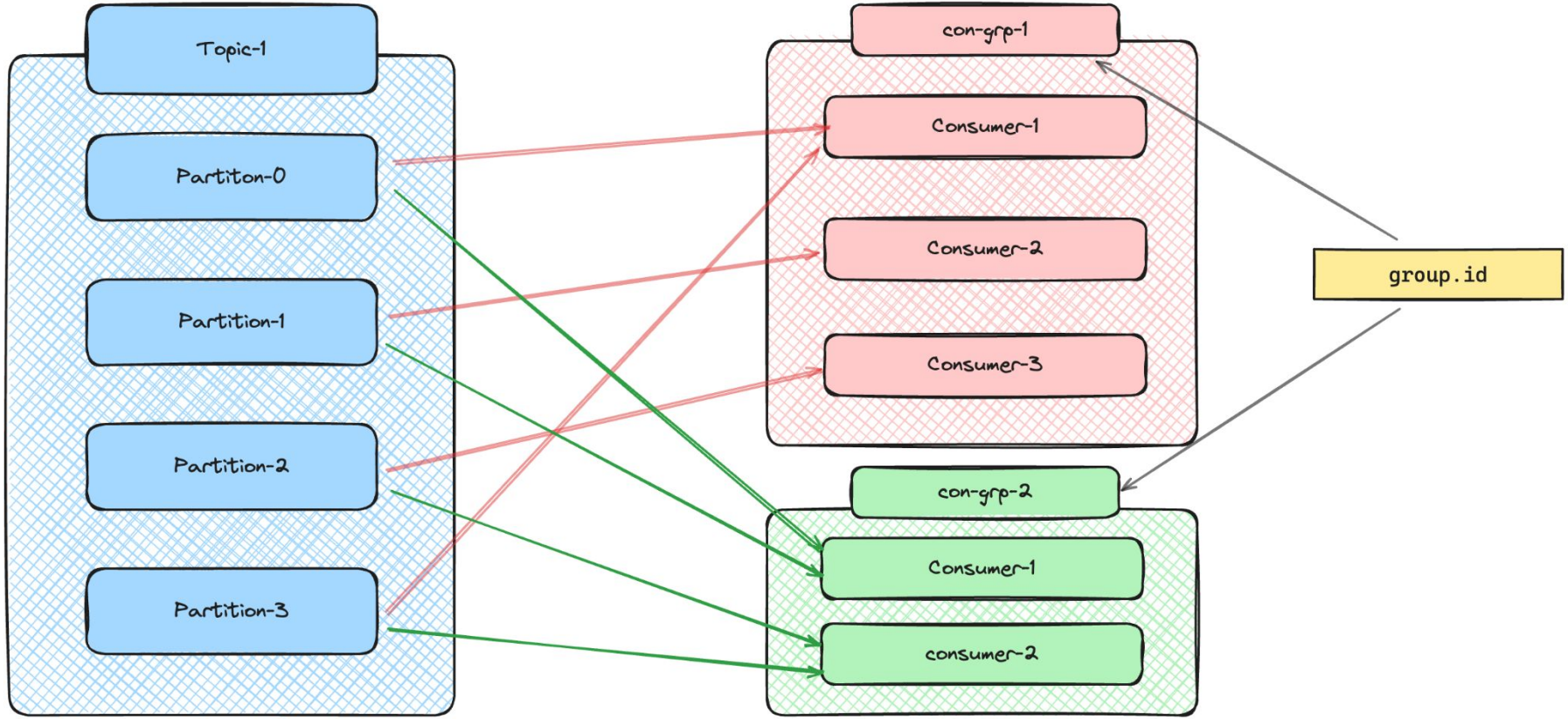
Advanced Topics- Consumers

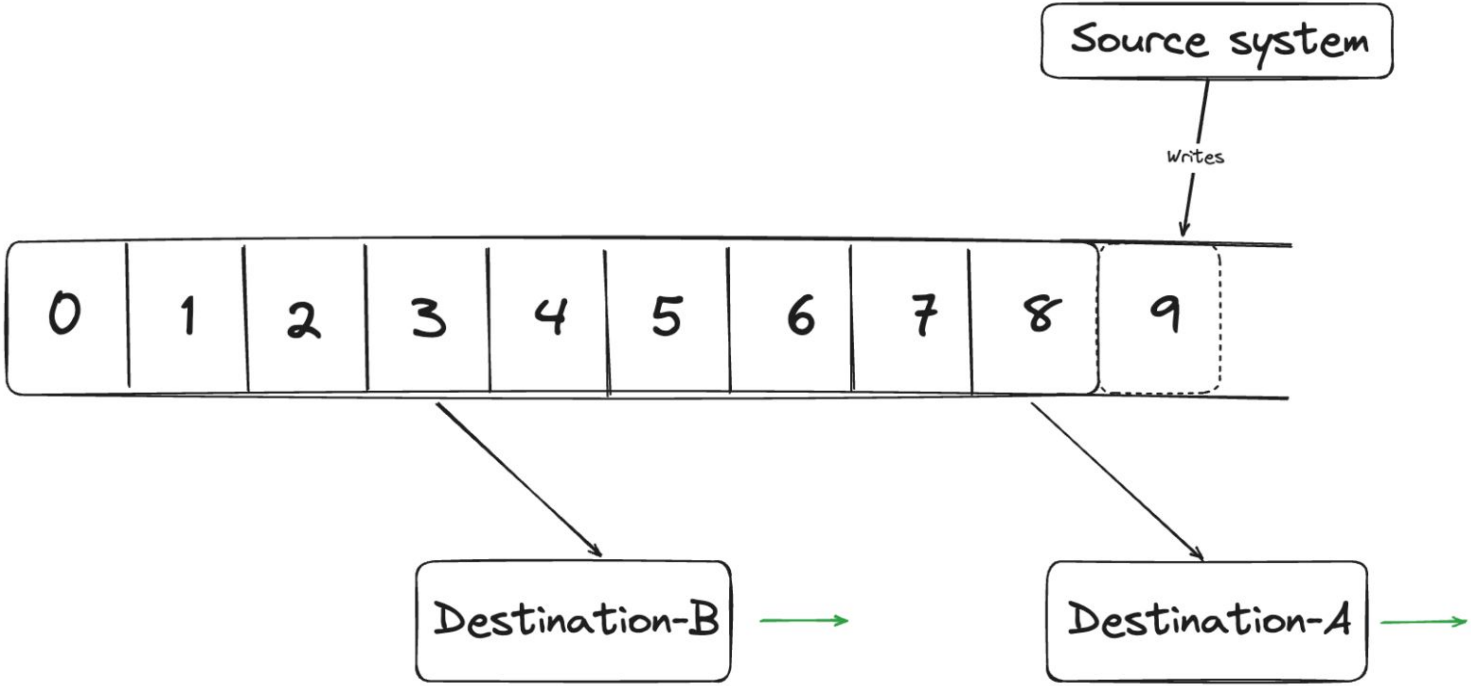
- Recap
- Consumer Groups Deep Dive
- Message delivery semantics
- Partition Assignment Strategies
- Static membership
- Important Consumer Configs

Advanced Topics - Consumers (Recap)

---

1. Consumers continuously pull data from brokers (pull model).
2. Consumers are grouped into consumers groups, which enable parallelizing the work across multiple thread under the same group.
3. Topic usually consists of many partitions, number of partitions define the max parallelization each consumer group can go up to.
4. Consumers use “offset” to track there location for each topic / partition.
    a. It is used in case of rebalance, Crashes, scalability (up/down).
    b. Latest committed offset help consumers to continue/start/restart from where.
    c. Offsets per consumer as stored in `__consumer_offsets` topic.
5. Consumers always read from the lower offset to the higher offset.
6. Consumers use Message Deserializers to read messages from Kafka.
    a. Message sent to Kafka topic that doesn’t respect the agreed-upon format are called “**Poison Pill**”.


Topic-1
Partiton-0
Partition-1
Partition-2
Partition-3
con-grp-1
Consumer-1
Consumer-2
Consumer-3
group.id
con-grp-2
Consumer-1
consumer-2


Source system
Writes
0
1
2
3
4
5
6
7
8
9
Destination-B
Destination-A

Advanced Topics - Consumers (Delivery Semantics)

---

1. **At most once:**
   a. Offsets are committed as soon as the message is received.
   b. If processing went wrong, message will be lost.
2. **At least once: (the preferred option)**
   a. Offsets are committed after the message is processed.
   b. If processing went wrong, the message be will be read again (double reading).
   c. Duplicate processing (partially processing) the message, it is recommended to make sure data processing is idempotent.
3. **Exactly once:**
   a. This can only be achieved in case Kafka Topics using Transactions API.
   b. Kafka Streams simplifies this using `processing.guarantee=exactly_once_v2`
   c. When integrating with external Kafka topics (unmanaged clusters), you must use ‘Idempotent consumer’.

Advanced Topics - Consumers (Consumer Group)

1. **QQ:** Can you recall what is a consumer group?
2. To define a consumer group we just need to set the `group.id` in the consumer config.
3. Once that is set, every new consumer instance of that consumer will be added to the same group.
4. When this group subscribes to topics, their partitions will be evenly distributed between the group instances.
5. As described before, the unit of parallelization is the partition.
6. For a given consumer group, Consumers can process more than 1 partition, but a partition can only be processed by one consumer.
7. Max numbers of consumers in any given group is capped by the total number of partitions in all the topics this consumer group subscribe to.
    a. If our group is subscribed to two topics and each one has two partitions then we can effectively use up to four consumers in the group.
    b. We could add a fifth consumer, but it will be idle.

Advanced Topics - Consumers (Consumer Group)

Group Coordinator

(broker side)

Group Leader

(consumer side)

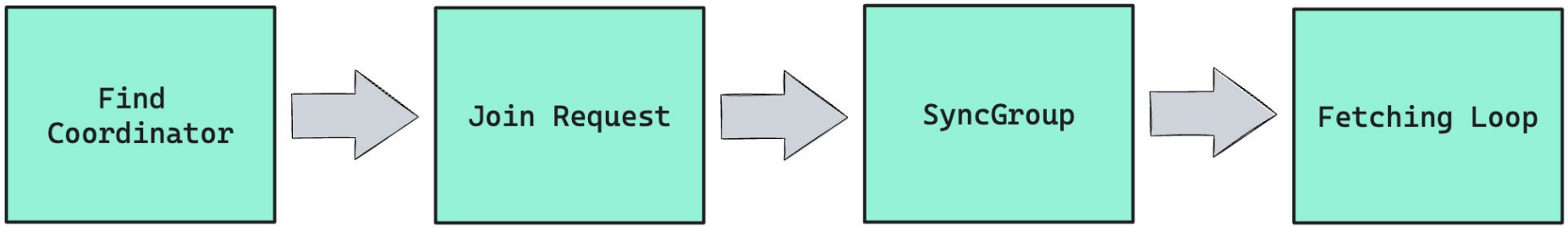
Group Protocol

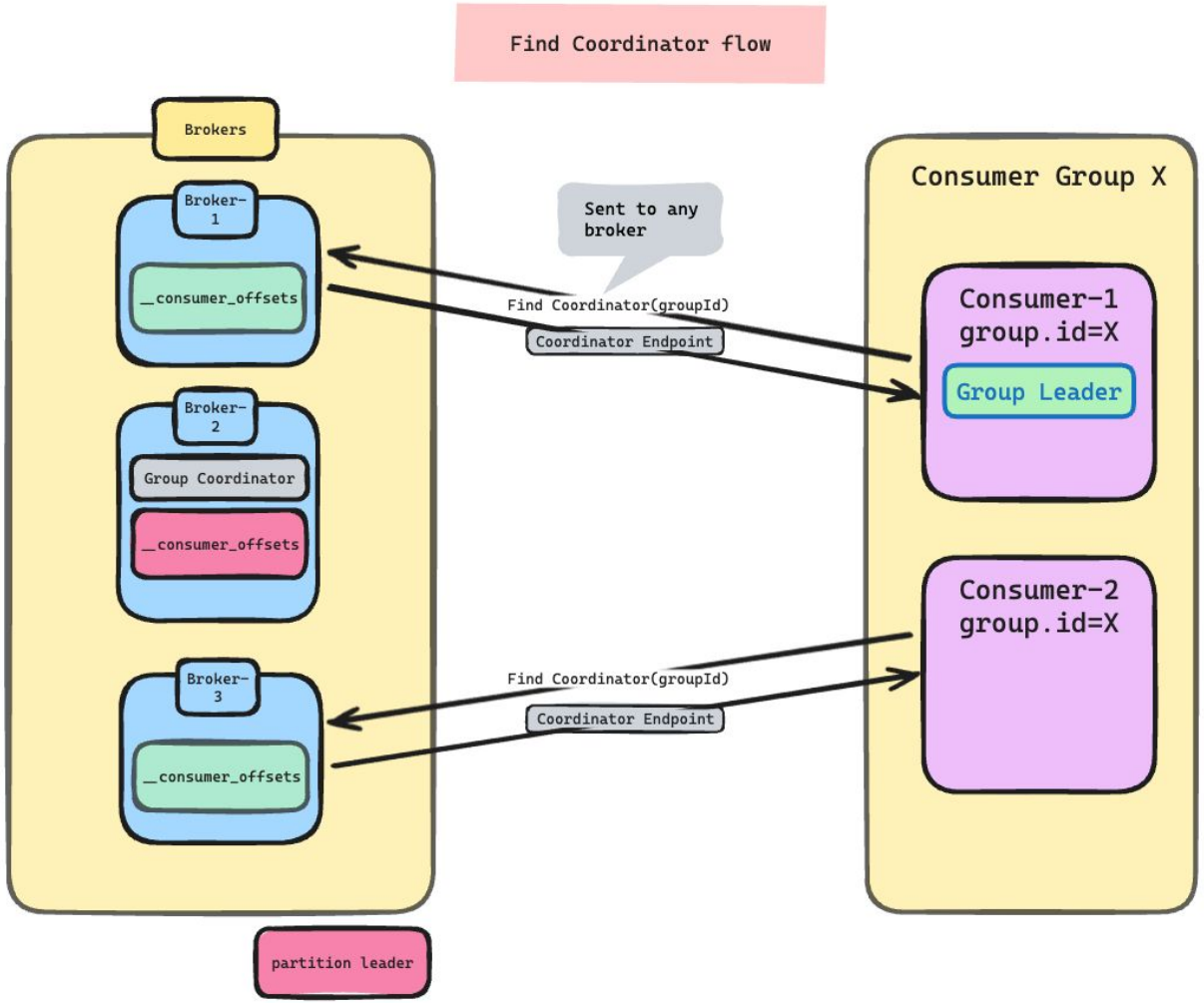
(consumer side)

Advanced Topics - Consumers (Consumer Group)

1. **GroupCoordinator:**
   a. When the consumer group started, One of the brokers is designated as the **GroupCoordinator**.
   b. It is responsible for managing the members of the group and their partition assignments.
   c. How is the **GroupCoordinator** selected?
      i. **GroupCoordinator** is the Leader of the internal offset topic `__consumer_offsets` where this group.id is being persisted at.
      ii. The group's ID is hashed to one of the partitions in the __consumer_offsets topic.
      iii. The leader of the partition corresponding to this hash becomes the coordinator for the consumer group.
      iv. In this way, consumer group management is divided "roughly equally" across brokers in the cluster, which enable better scalability
2. **GroupLeader:**
   a. This is the first consumer in the group to send JoinRequest.
   b. **GroupLeader** will maintain a full list of partition assignments.
   c. Consumers other than the leader will only see their partition assignments.
   d. Once **GroupLeader** assigned partitions, it sends it to the **GroupCoordinator** to distribute it to all consumers.
   e. **GroupLeader** don't communicate directly with other consumers.
   f. This process happens every time a rebalance occurs.
3. **Group Protocol:**
   a. Is the approach used during rebalances by the **GroupLeader** to distribute partitions on other consumers.
   b. It is totally handled by the client side, and the **GroupLeader** send the encoded assignment to the **GroupCoordinator**, and then **GroupCoordinator** handed it to the other group members.

Advanced Topics - Consumers (Consumer Group Startup)

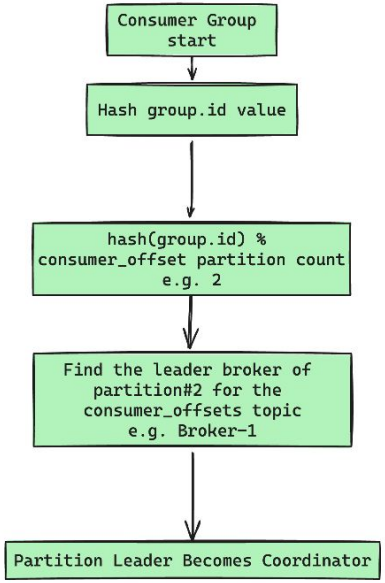


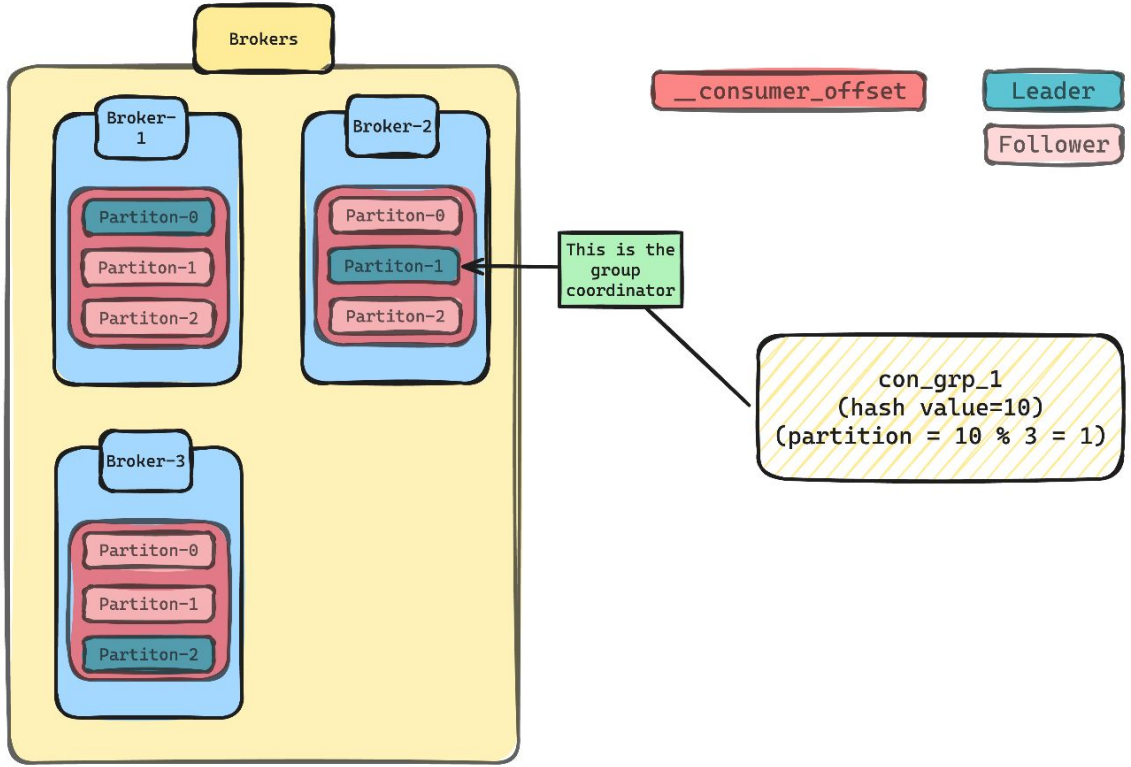


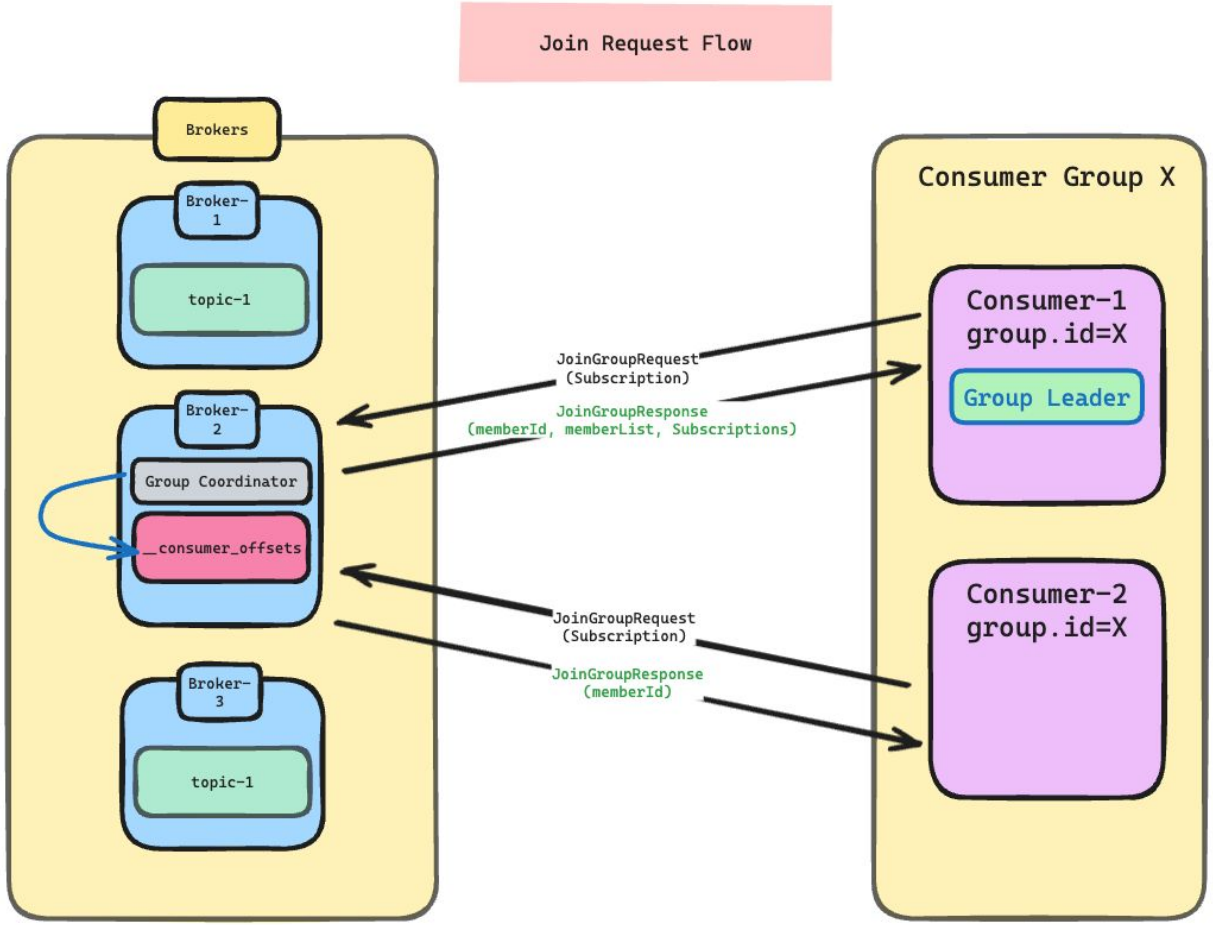
Finding the Group Coordinator:

- The consumer determines which broker is the GroupCoordinator for its consumer group by hashing the group ID.
- The hashed group ID maps to a partition in the `__consumer_offsets` topic.
- The leader of that partition is the GroupCoordinator for the consumer group.
- The consumer sends a `FindCoordinator` request to any broker.
- The broker responds with the address of the GroupCoordinator.

Advanced Topics - Consumers (Finding Coordinator)







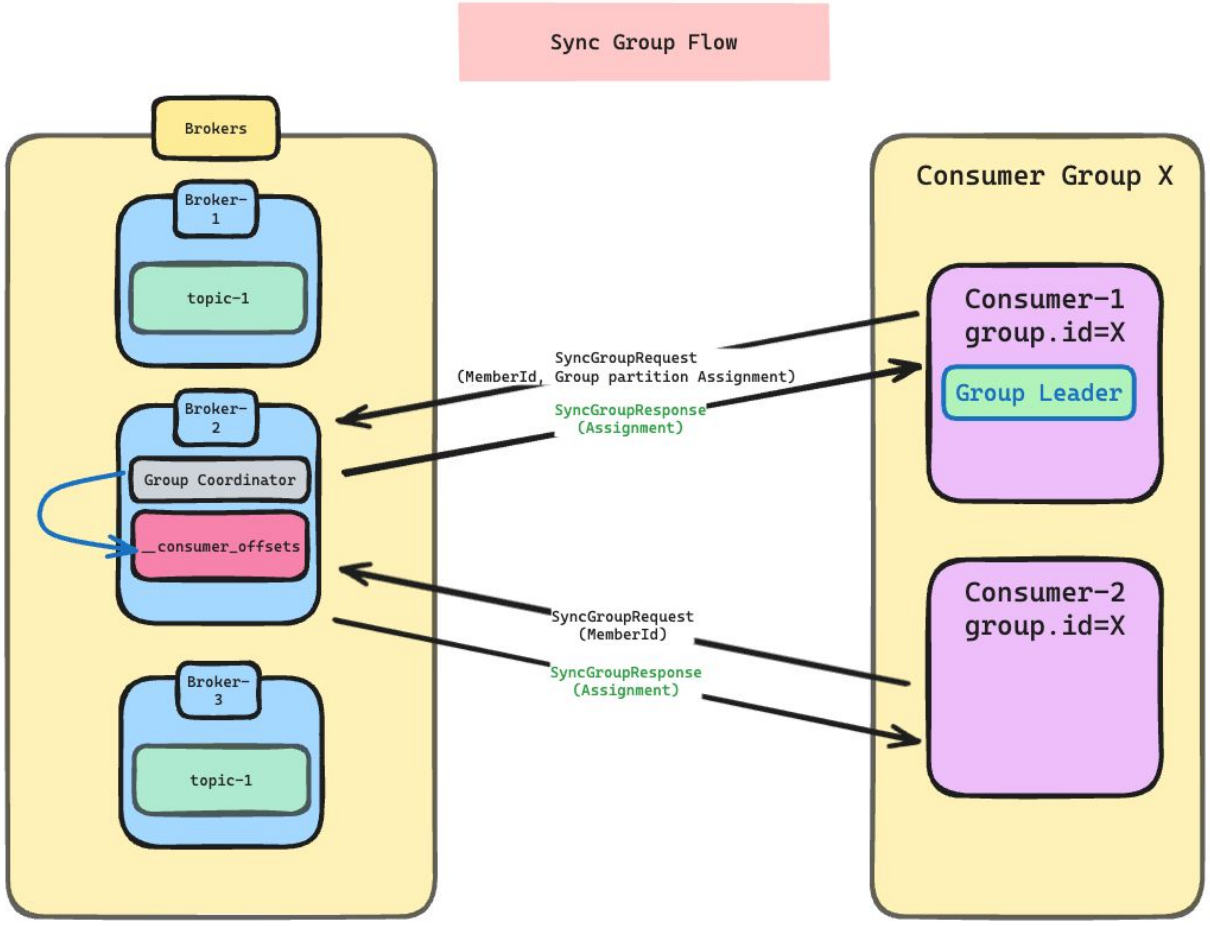
Join Request Flow:

Sending Join Group Request:

- Once the consumer knows the **`GroupCoordinator`**, it sends a `JoinGroupRequests` to the coordinator.
- This request includes the consumer's metadata (subscription topics).

Group Coordination:

- The **`GroupCoordinator`** receives `JoinGroupRequests` from all consumers in the group.
- If it is the first consumer to join, it becomes the group leader.
- The **`GroupCoordinator`** waits for a configurable amount of time (default 10 seconds) to gather join requests from other consumers.
- Send `JoinGroupResponse` to the **`GroupLeader`**, it sends its **`memberId`**, **all memberList**, and **subscriptions**, **GenerationId**
- Send to other consumers their **`memberId`** and **GenerationId**



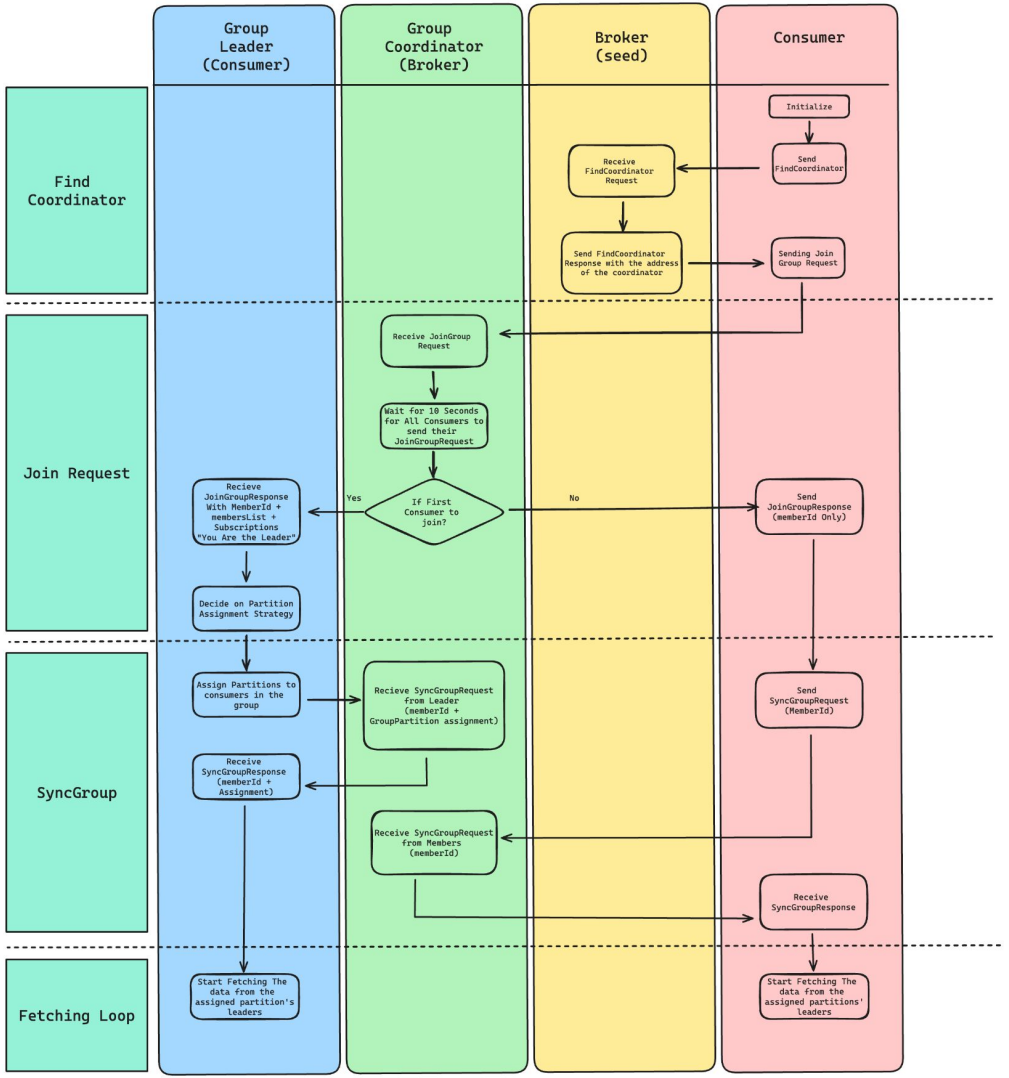
SyncGroup Request Flow:

Sending Sync Group Request:

- **`GroupLeader`** do the partitions Assignment on the memberList (based on the `` `partition.assignment.strategy` `` consumer config)
- **`GroupLeader`** send it to back the **`GroupCoordinator`** in the `SyncGroupRequest`.
- Each consumer sends a `SyncGroupRequest` to the **`GroupCoordinator`** to receive its assignment.
- **`GroupCoordinator`** reply with the assignment of each consumer in `SyncGroupResponse` for each consumer.

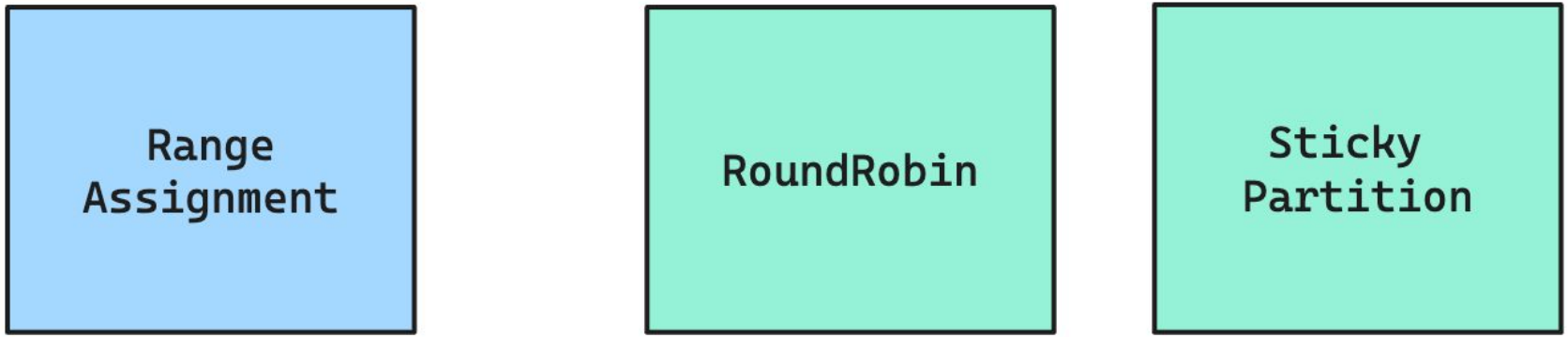
Receiving Partition Assignment:

- Each consumer receives the partition assignment from the `SyncGroup` response.
- The consumer updates its internal state with the assigned partitions.



Subscription:
are the info shared from
all consumers about
the topic they want to subs to

Advanced Topics - Consumers (Partition Assignment Strategies)

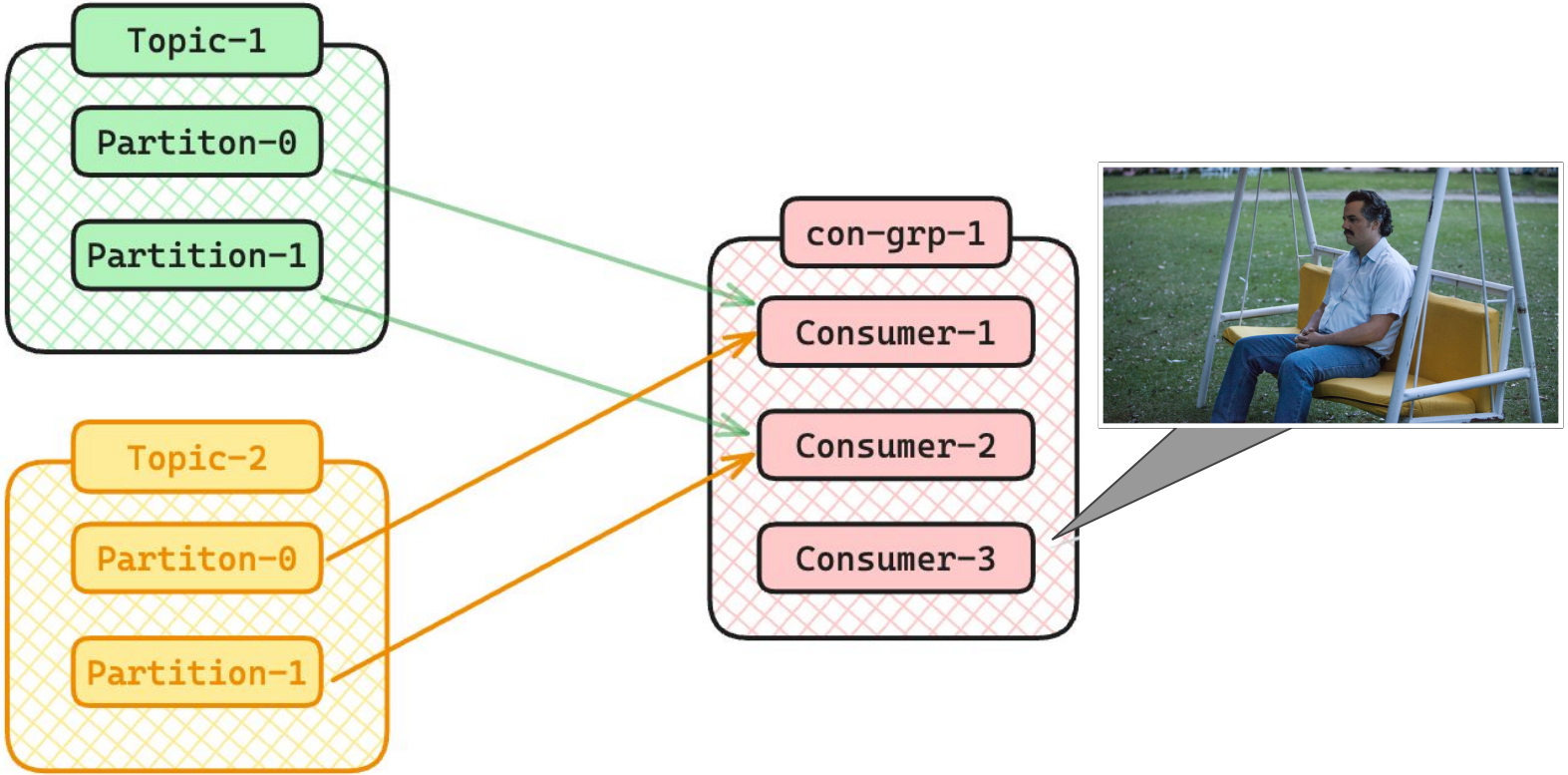


Advanced Topics - Consumers (Range Assignment)

Range Assignment Strategy

a. It goes through each topic in the subscription and assigns each partition to a consumer, starting with the first consumer.
b. The first partition of each topic is assigned to the first consumer, the second partition to the second consumer, and so on.
c. If the number of partitions in a topic is less than the number of consumers, some consumers will remain idle for that topic.
d. It seems to be a not good strategy, however there is a special case this strategy works well for, which is, Joining events from more than one topic, and events has to be read by the same consumer, using same key, then they will be falling in the same partition number in both topics, hence same consumer.

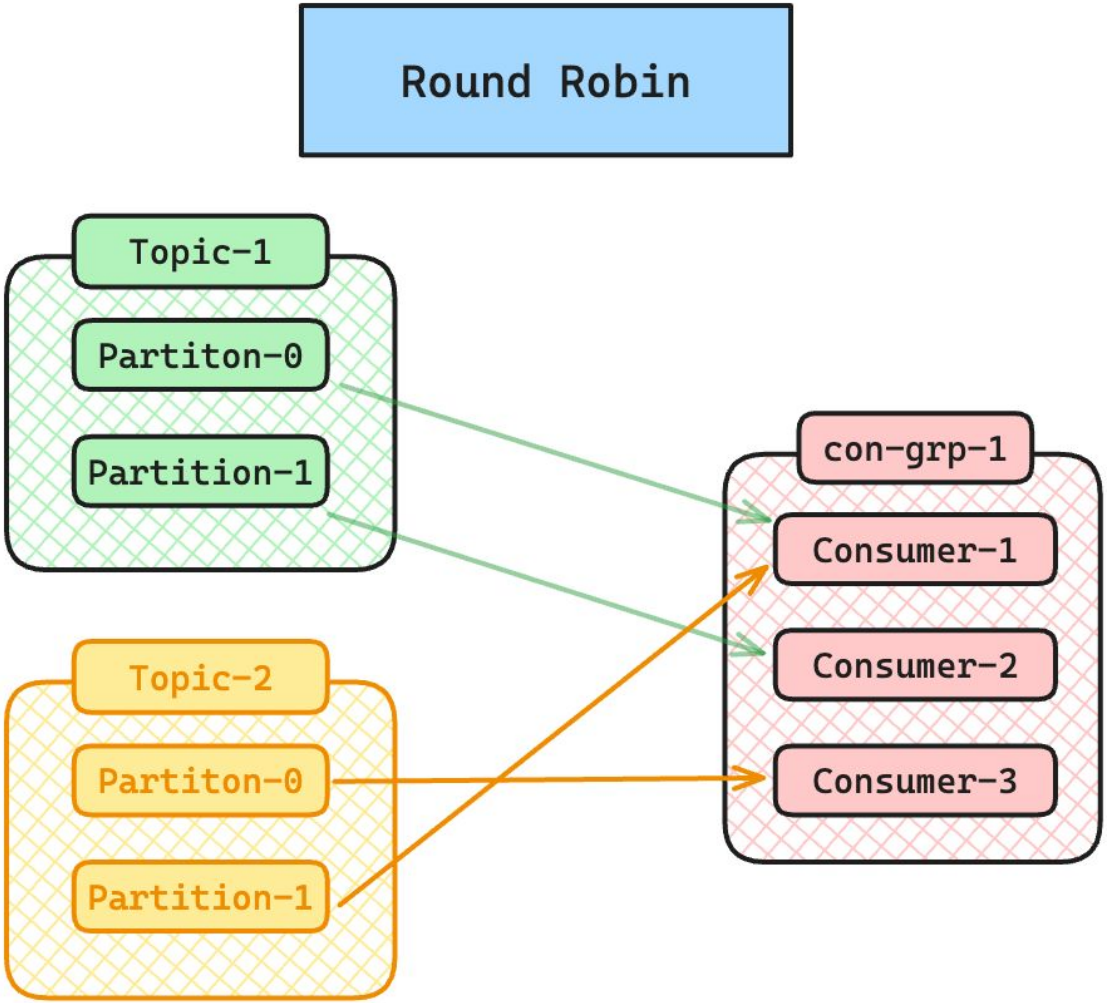
Range Assignment



Advanced Topics - Consumers (Round Robin Strategy)

Round Robin Strategy:

a. Partitions of the subscriptions are distributed evenly across the available consumers.
b. Fewer idle consumer instances, and higher parallelization.
c. Idleness could happen in case number of consumers > subscripted partitions.


Round Robin
Topic-1
Partiton-0
Partition-1
Topic-2
Partiton-0
Partition-1
con-grp-1
Consumer-1
Consumer-2
Consumer-3

Advanced Topics - Consumers (Sticky Partition Strategy)

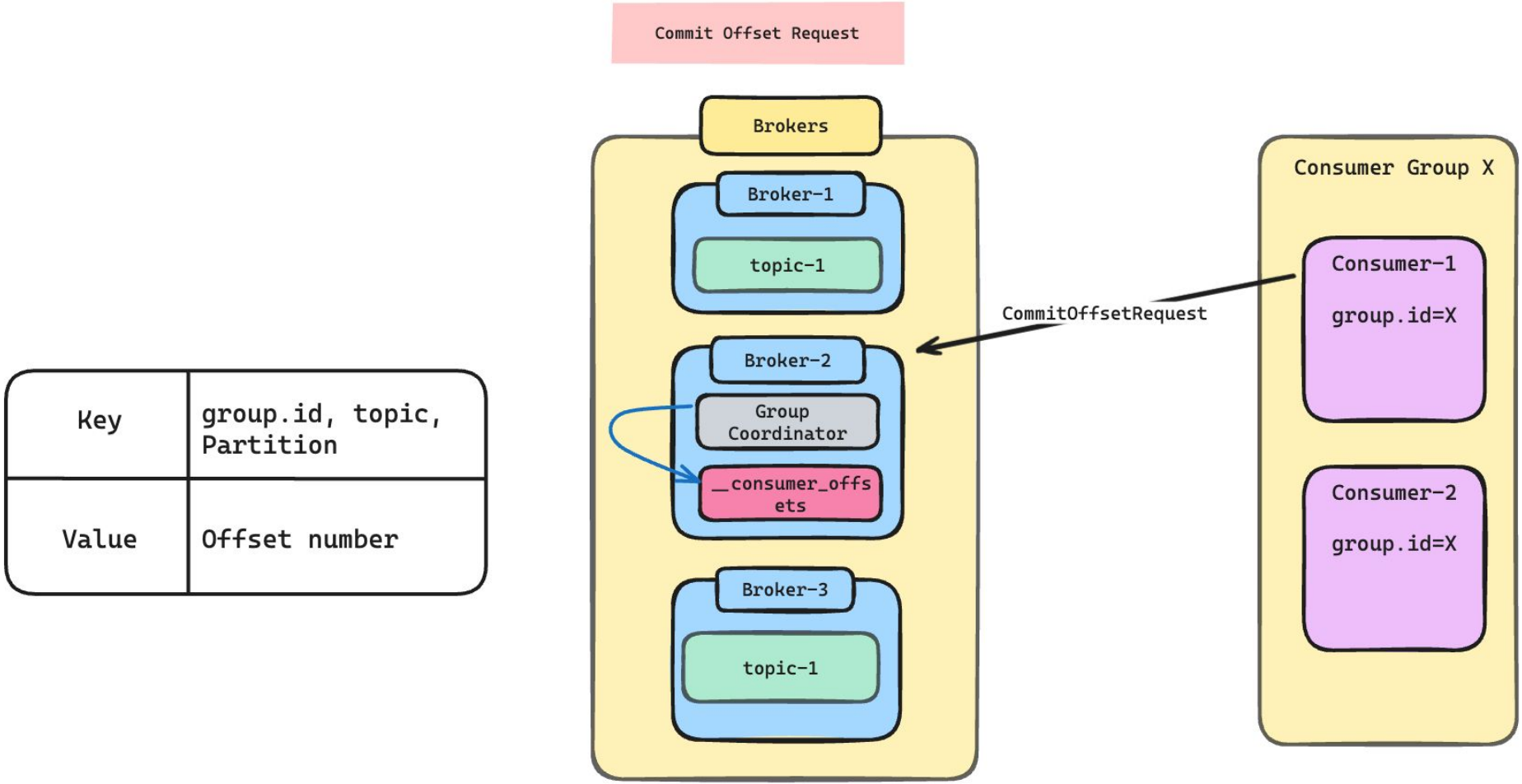
Sticky Partition Strategy

a. A variant of Round Robin, use the same concept.
b. But, make a best effort to stick to the previous assignment during rebalance.
c. This strategy provides Faster and more efficient rebalance.

Advanced Topics - Consumers (Tracking Partition offsets)

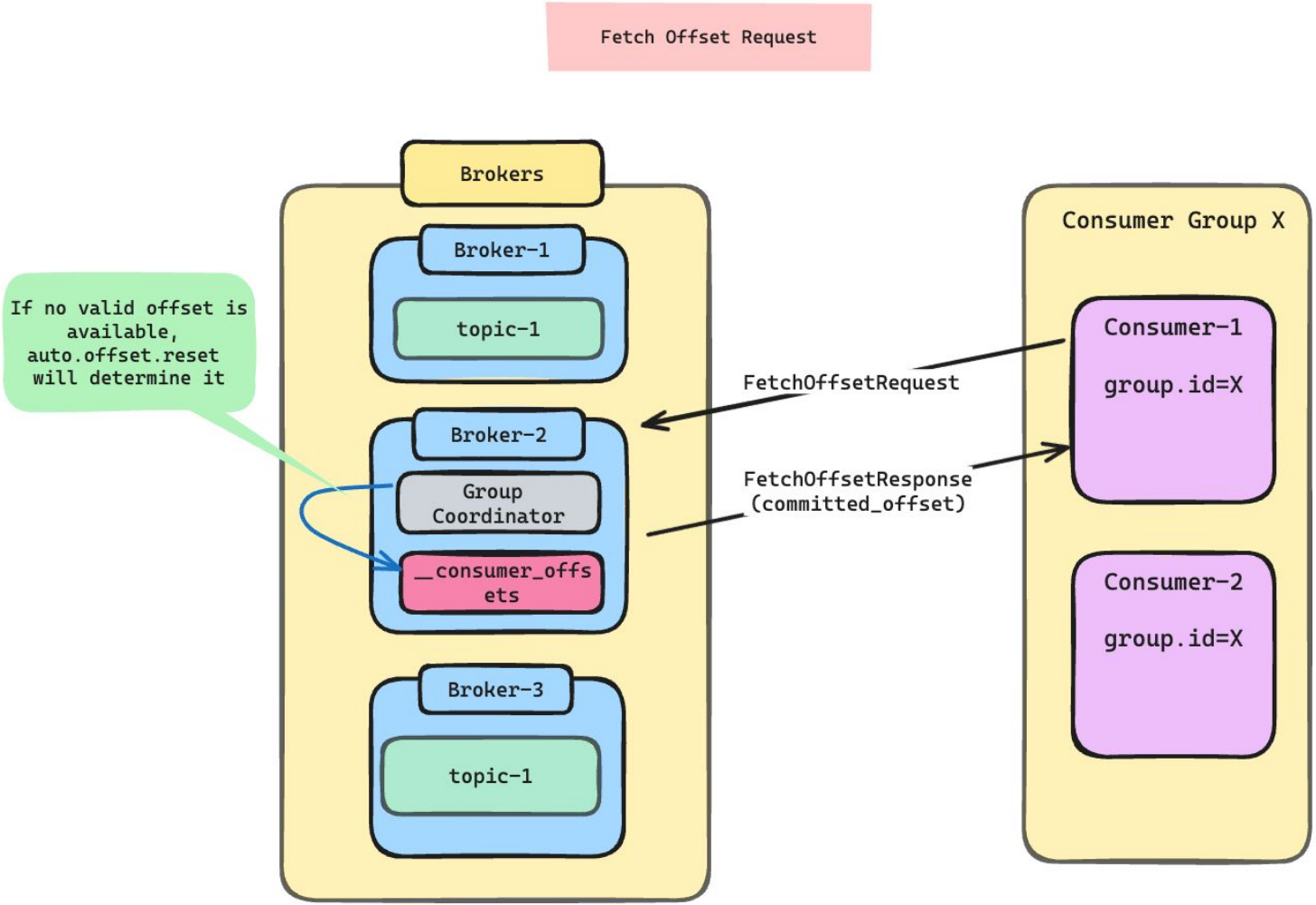
---

a. Given partition is always assigned to a single consumer.
b. Event in that partition are read by the consumer in offset order (lower to higher offset).
c. So, Consumer has to track the last offset is has consumed.
d. To do so, consumer will issue a “**ConsumerOffsetRequest**” to the group coordinator.
e. Coordinator will then persist this piece of info in `**__consumer_offsets**` internal topic.


Commit Offset Request
Brokers
Broker-1
topic-1
Broker-2
Group Coordinator
__consumer_offsets
Broker-3
topic-1
CommitOffsetRequest
Consumer Group X
Consumer-1
group.id=X
Consumer-2
group.id=X
Key
group.id, topic, Partition
Value
Offset number

Advanced Topics - Consumers (Determining Starting Offset)

a. When consumer group instance is restarted, it will send a "**OffsetFetchRequest**" to the group coordinator to get the last committed offset for its assigned partition.
b. Once it has the offset it will start consumption normally from that offset onwards.
c. In case the consumer instance is starting for the very first time and there is no saved offset position, then GroupCoordinator will fallback to the "**auto.offset.reset=latest/earliest**" configuration.


Fetch Offset Request
Brokers
Broker-1
topic-1
Broker-2
Group Coordinator
__consumer_offsets
Broker-3
topic-1
If no valid offset is available, auto.offset.reset will determine it
FetchOffsetRequest
FetchOffsetResponse (committed_offset)
Consumer Group X
Consumer-1
group.id=X
Consumer-2
group.id=X

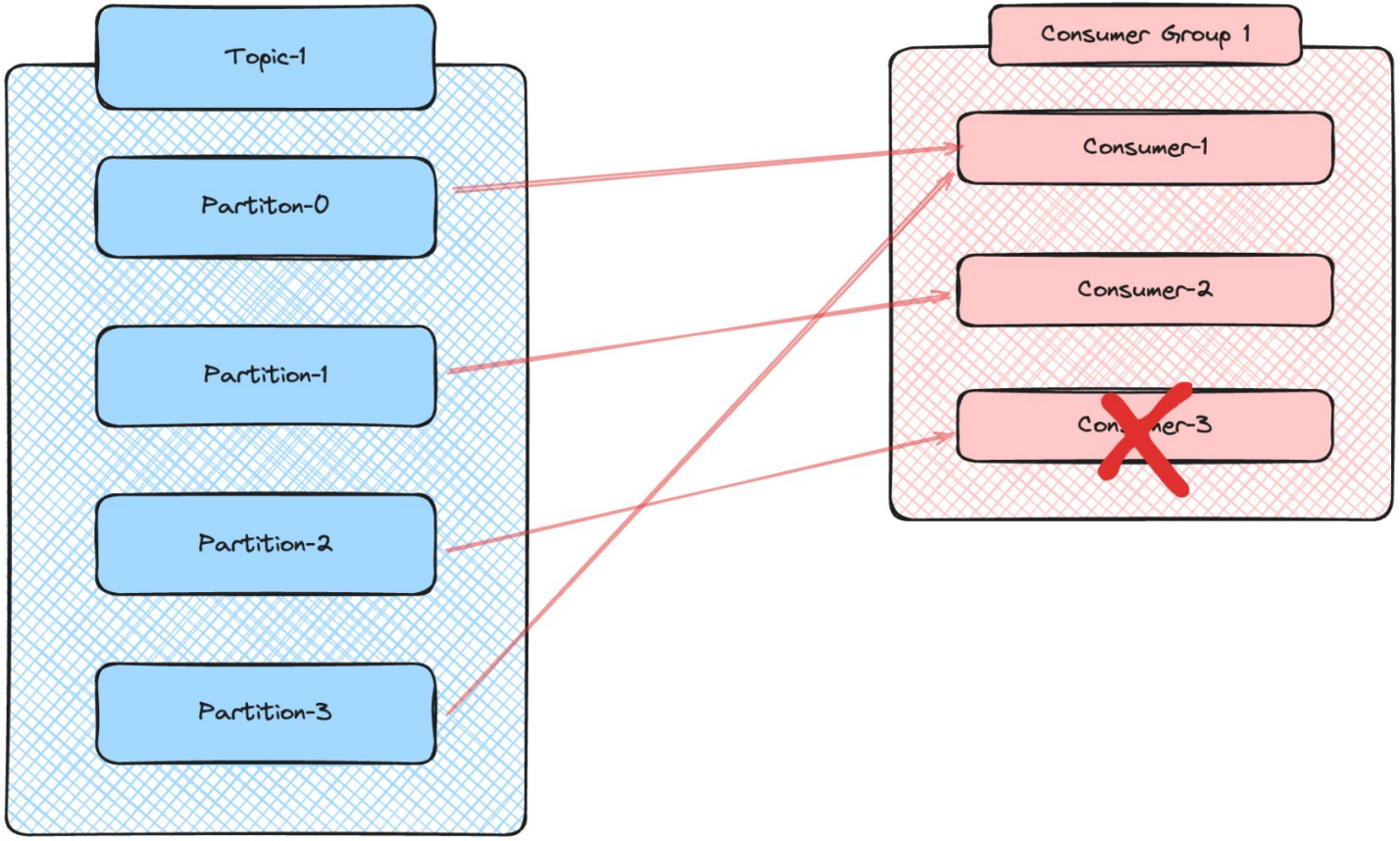
Advanced Topics - Consumers (Group Coordinator Failure)

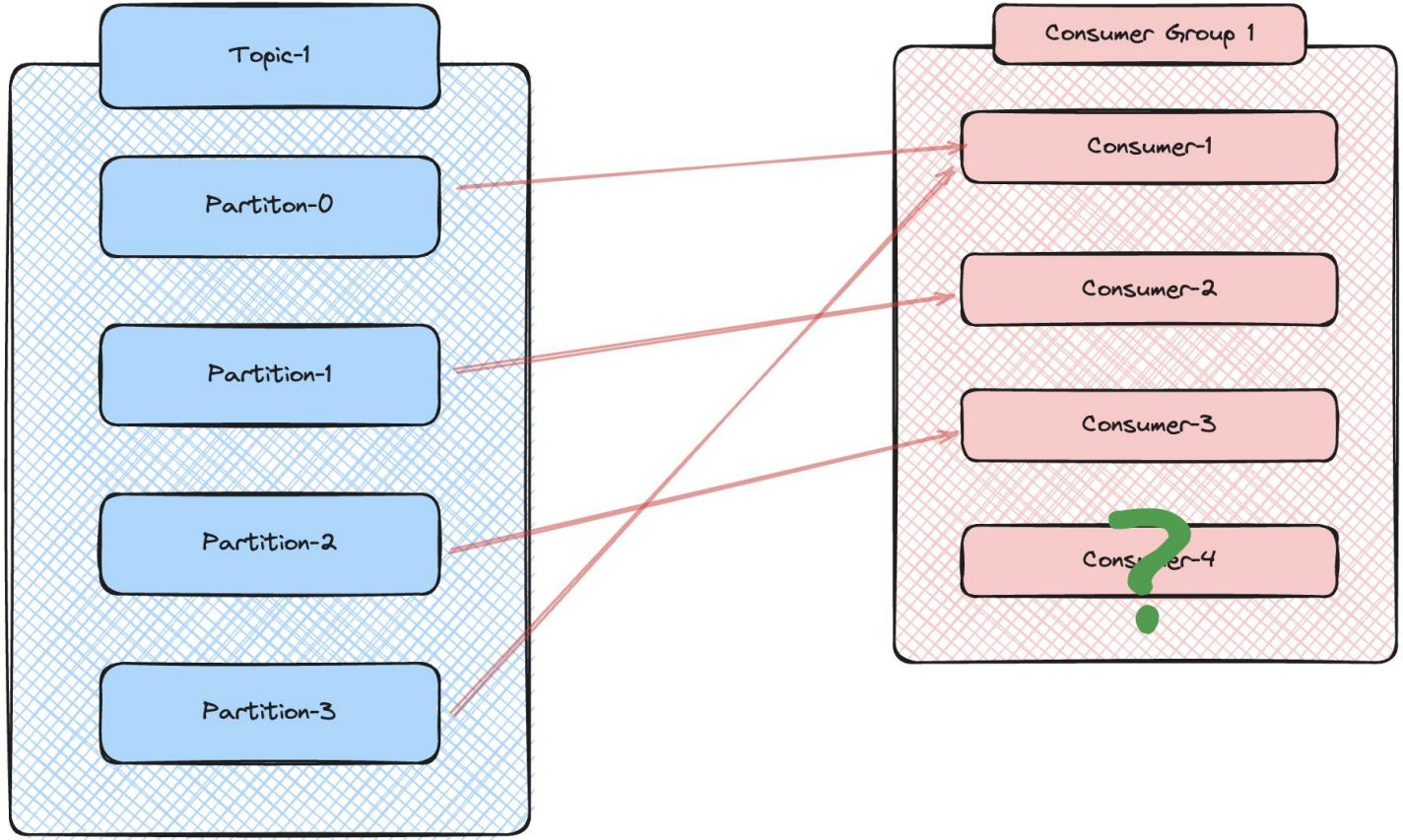
- Brokers go down for many many reasons (failure, upgrades, …)
- Recalling that the broker is the partition leader of the “__consumer_offsets” that corresponds to a consumer group(s).
- When a broker which is acting as a GroupCoordinator fails, a new Leader is elected to handle the “__consumer_offsets” partitions.
- Consumers will be notified of the new coordinator when they try to make a call to the old one, and things will continue normally.

```
Log Sample of Coordinator Failure:
INFO [Worker clientId=connect-1, groupId=demo] Group coordinator broker:29092 (id: 2147483546 rack: null) is unavailable or invalid due to cause: coordinator unavailable. isDisconnected: false. Rediscovery will be attempted. (org.apache.kafka.connect.runtime.distributed.WorkerCoordinator)
```

Advanced Topics - Consumers (Group Rebalance)

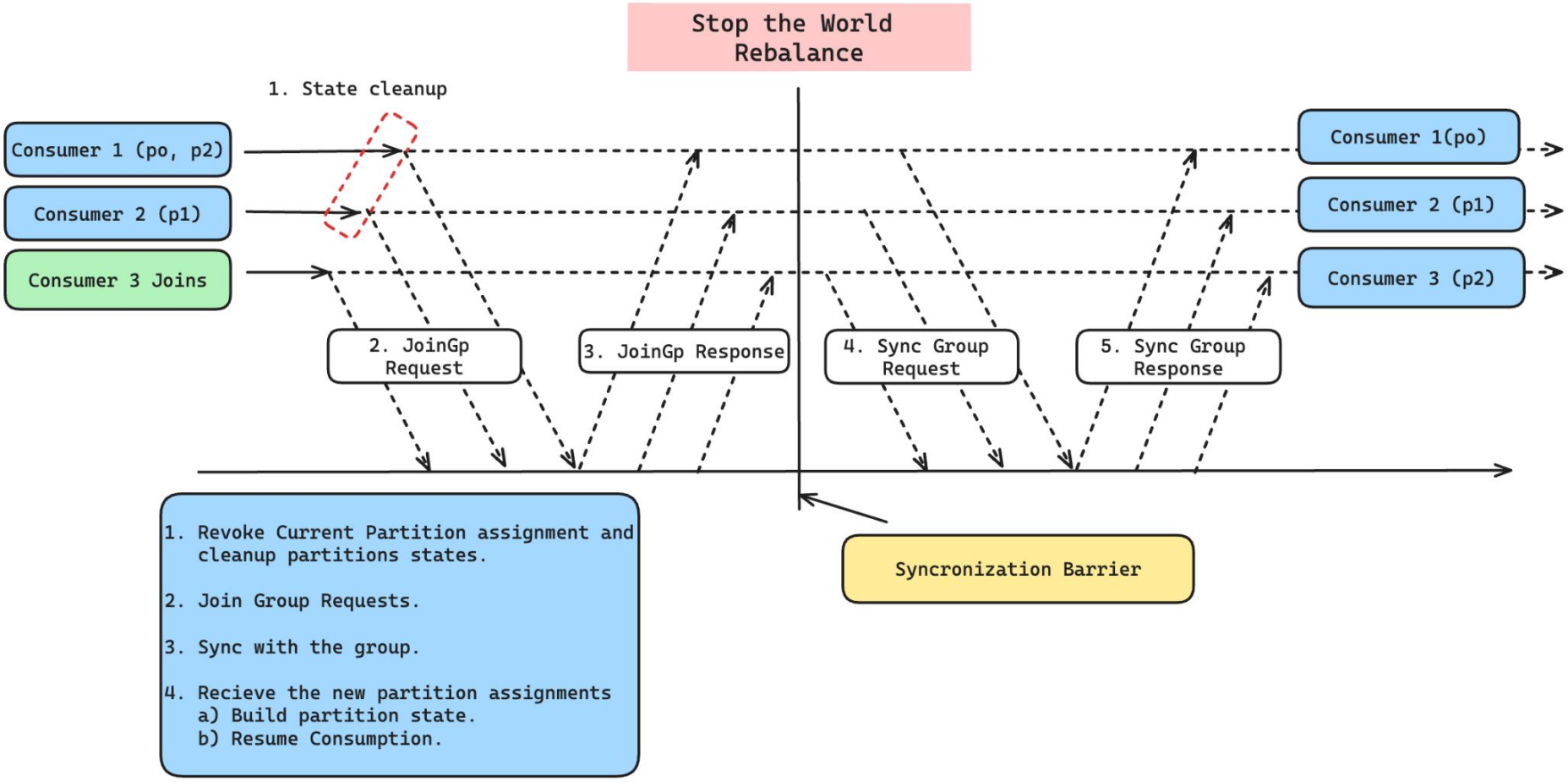
- What is a Consumer Rebalance
- What Could cause a consumer group to rebalance?
  - A consumer leaves the group. (e.g. heartbeat timeout)
  - A new consumer instance joins the group.
  - Group startup, this could happen in case there is a lag (>10 sec) between any of the consumers join request.
  - Partitions are add to the topic in the subscription.
  - Topic added that match a subscription pattern (e.g. **consumer.subscribe(Pattern.compile("topic_.");**")
- When rebalance occurs, GroupCoordinator starts to reply to all consumers in most of the requests with "**REBALANCE_IN_PROGRESS**".
- Stop-The-World Rebalance:
  - All consumers stop fetching data from brokers until the partitions resassigned, and all consumers follow the consumer group startup flow (JoinGroupRequest/Response, SyncRequest/Response, FetchData).
  -


Topic-1
Partiton-0
Partition-1
Partition-2
Partition-3
Consumer Group 1
Consumer-1
Consumer-2
Consumer-3


Topic-1
Partiton-0
Partition-1
Partition-2
Partition-3
Consumer Group 1
Consumer-1
Consumer-2
Consumer-3
?

Advanced Topics - Consumers (Group Rebalance)

- When rebalance occurs, GroupCoordinator starts to reply to all consumers in most of the requests with “**REBALANCE_IN_PROGRESS**”.
- Stop-The-World Rebalance:
  - All consumers stop fetching data from brokers until the partitions reassigned, and all consumers follow the consumer group startup flow (JoinGroupRequest/Response, SyncRequest/Response, FetchData).


Stop the World
Rebalance
1. State cleanup
Consumer 1 (po, p2)
Consumer 2 (p1)
Consumer 3 Joins
2. JoinGp Request
3. JoinGp Response
4. Sync Group Request
5. Sync Group Response
Consumer 1(po)
Consumer 2 (p1)
Consumer 3 (p2)
1. Revoke Current Partition assignment and cleanup partitions states.
2. Join Group Requests.
3. Sync with the group.
4. Recieve the new partition assignments
a) Build partition state.
b) Resume Consumption.
Syncronization Barrier

Advanced Topics - Consumers (Stop-The-World Problems)

— — —

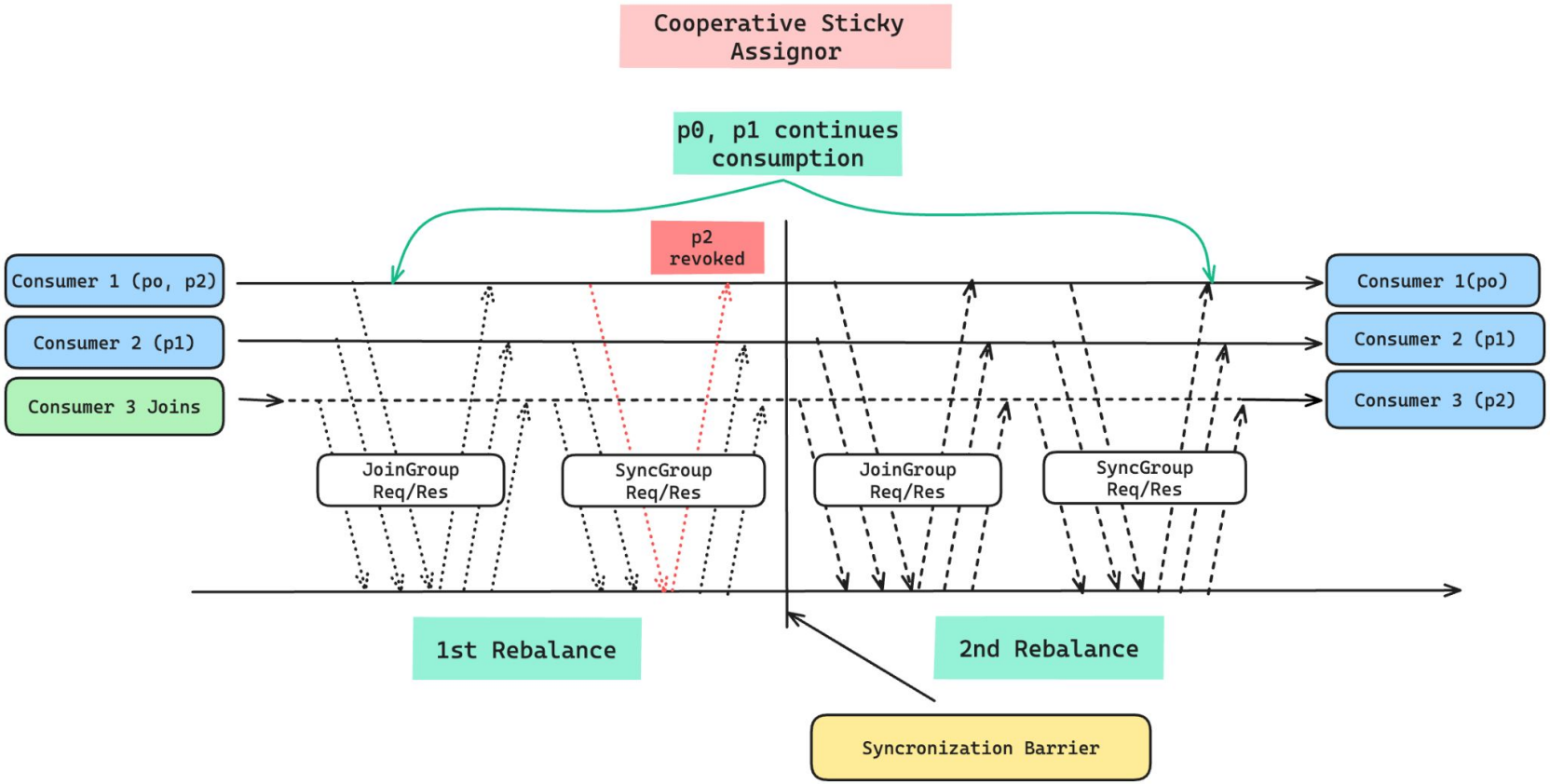
Stop-The-World introduced 2 main problems:

1. **State Rebuild**
   a. In case state depends on the events from the partitions, consumer may need to read all of the events in the partition to rebuild the state.
   b. Even if the consumer is assigned the same partitions it was assigned before, it has to re-read all the event again, WHY???, because it has revoked the partitions and cleared the state.
   c. **Solution:**
      i. **StickyAssignor:** The state cleanup and is moved to a later step, after the assignments are complete.
      ii. That way if a consumer is reassigned the partitions or few of them, it will not require to rebuild the entire state.
2. **Pause the entire processing**
   a. When rebalance starts, all processing must be paused until the rebalance process is completed and the all the partitions re-assigned, as if it is the ConsumerGroup startup process.
   b. **Solution:**
      i. **CooperativeStickyAssignor:**

Advanced Topics - Consumers (Stop-The-World Problems)

CooperativeStickyAssignor works in 2 steps:

1. **1st Rebalance Cycle (JoinGroup / SyncGroup)** is done just to determine which partitions will require partitions to be revokek, this will be done without pausing the consumption.
2. **A 2nd Rebalance Cycle (JoinGroup / SyncGroup),** by which the revoked partitions will be assigned.

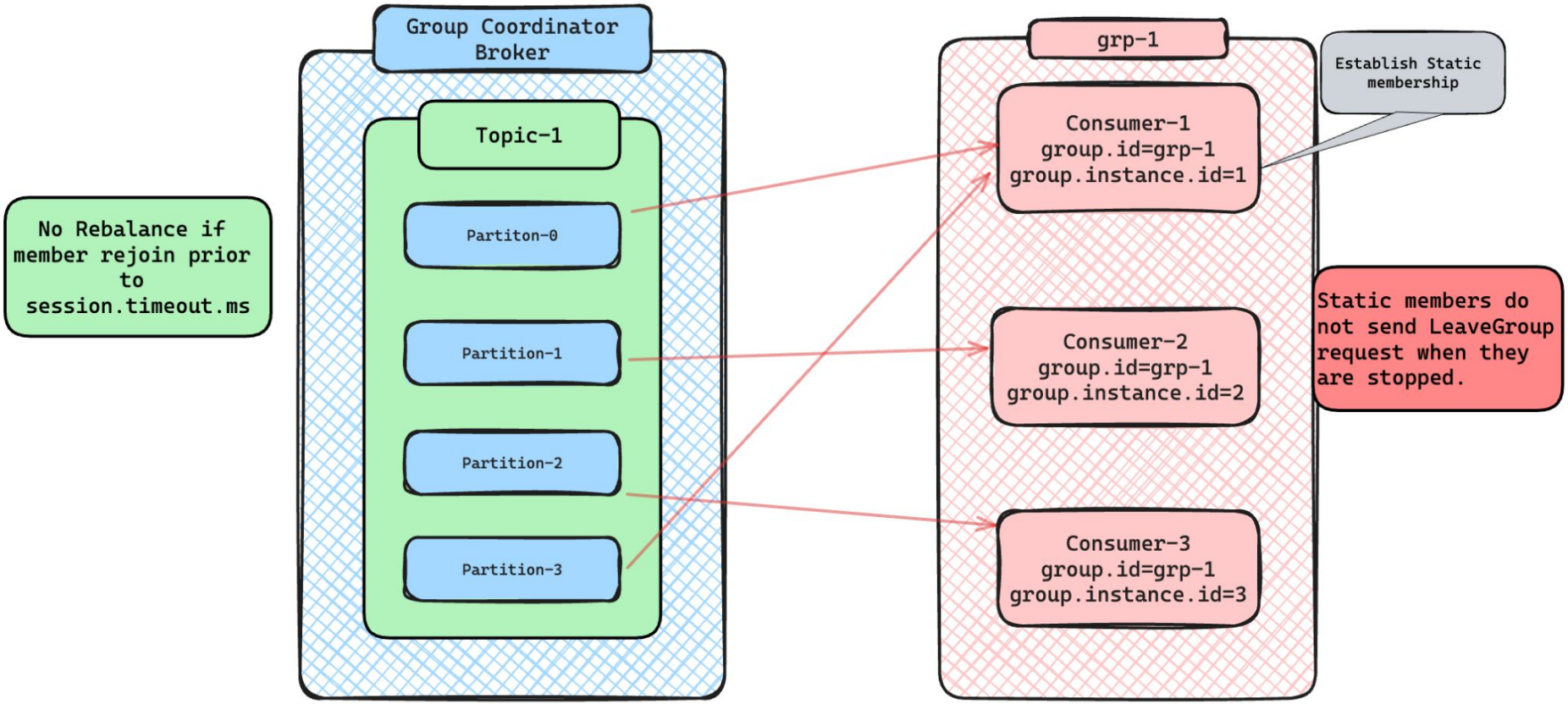

Cooperative Sticky Assignor
p0, p1 continues consumption
p2 revoked
Consumer 1 (po, p2)
Consumer 2 (p1)
Consumer 3 Joins
Consumer 1(po)
Consumer 2 (p1)
Consumer 3 (p2)
JoinGroup Req/Res
SyncGroup Req/Res
JoinGroup Req/Res
SyncGroup Req/Res
1st Rebalance
2nd Rebalance
Syncronization Barrier

Advanced Topics - Consumers (Static Membership)

---

1. So far, partitions are revoked whenever a rebelance happened, regardless the way of handling the effect (StickyAssignor, CooperativeStickyAssignor…)
2. By setting `group.instance.id` in the consumer config, this marks the consumer as a static member.
3. Upon leaving, the consumer has up to `session.timeout.ms` to join back and get back its partitions (without rebalance being triggered), otherwise they will be reassigned to other consumers.

Static Memebership



Advanced Topics - Consumers (Polling / Heartbeat)

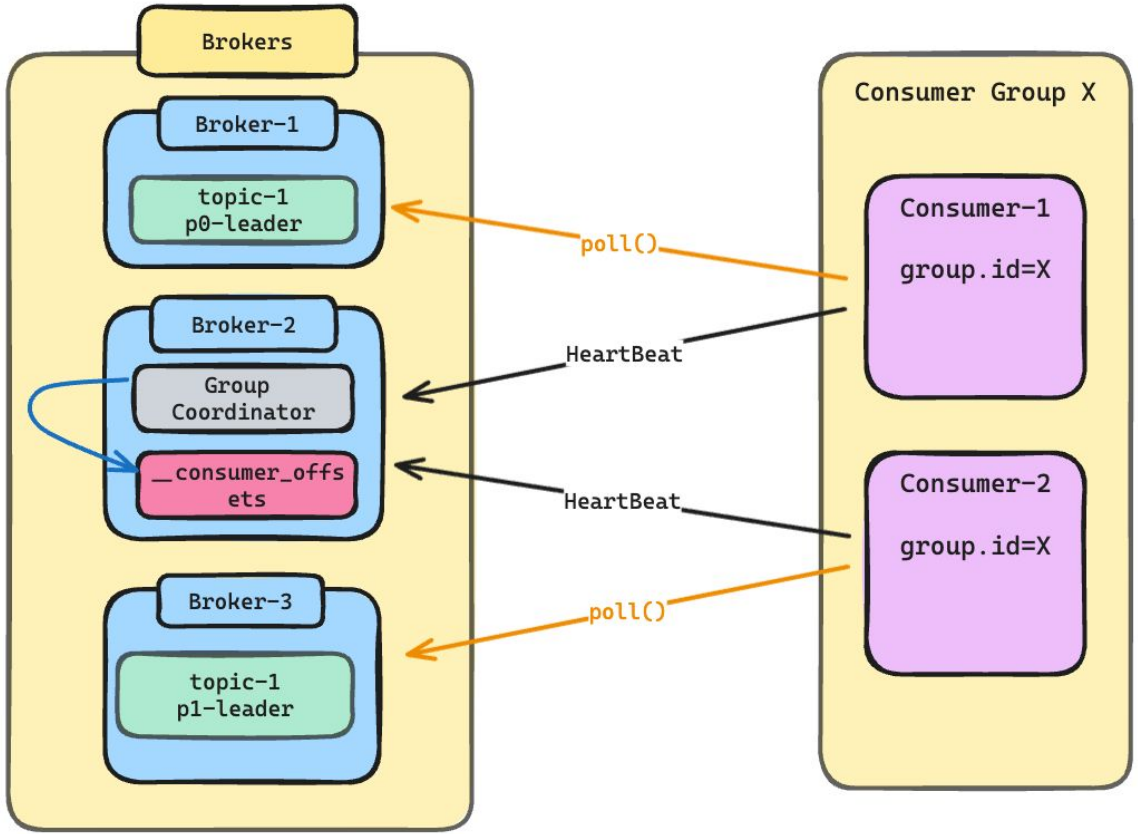
---

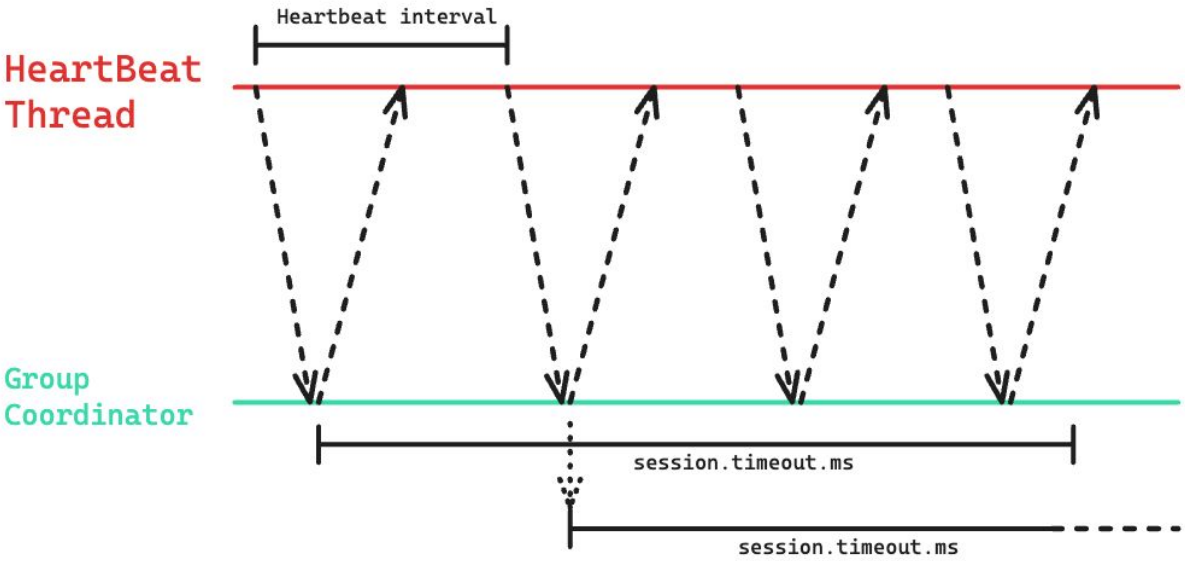
1. Kafka consumer once started and subscribed to a topic, it starts a polling loop.
2. Polling is responsible for:
    a. From where in the log the consumer will consume. (**auto.offset.rest**)
    b. How fast they want to consume
    c. Ability to reply event.
3. HeartBeat is responsible for:
    a. Determine the consumer liveness.
    b. Keeps the consumer membership in the group.
    c. **heartbeat.internal.ms=3000** and **session.timeout.ms=45000** are the 2 main consumer configs control the heartbeat logic.

HeartBeat and Poll

- HeartBeat requests is sent to the group coordinator

- Poll() is sent to the partition leader that the consumer is assigned to.

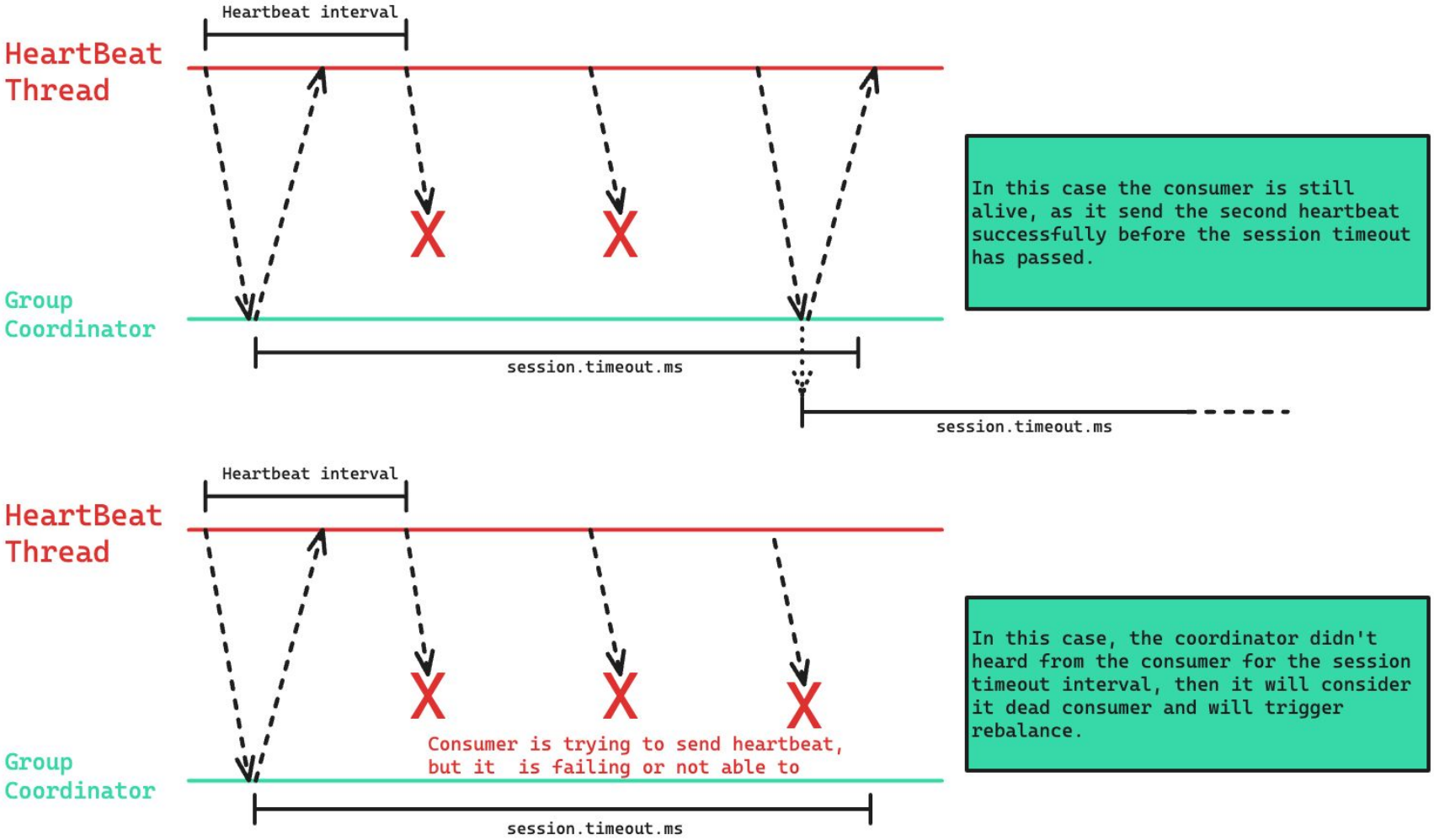


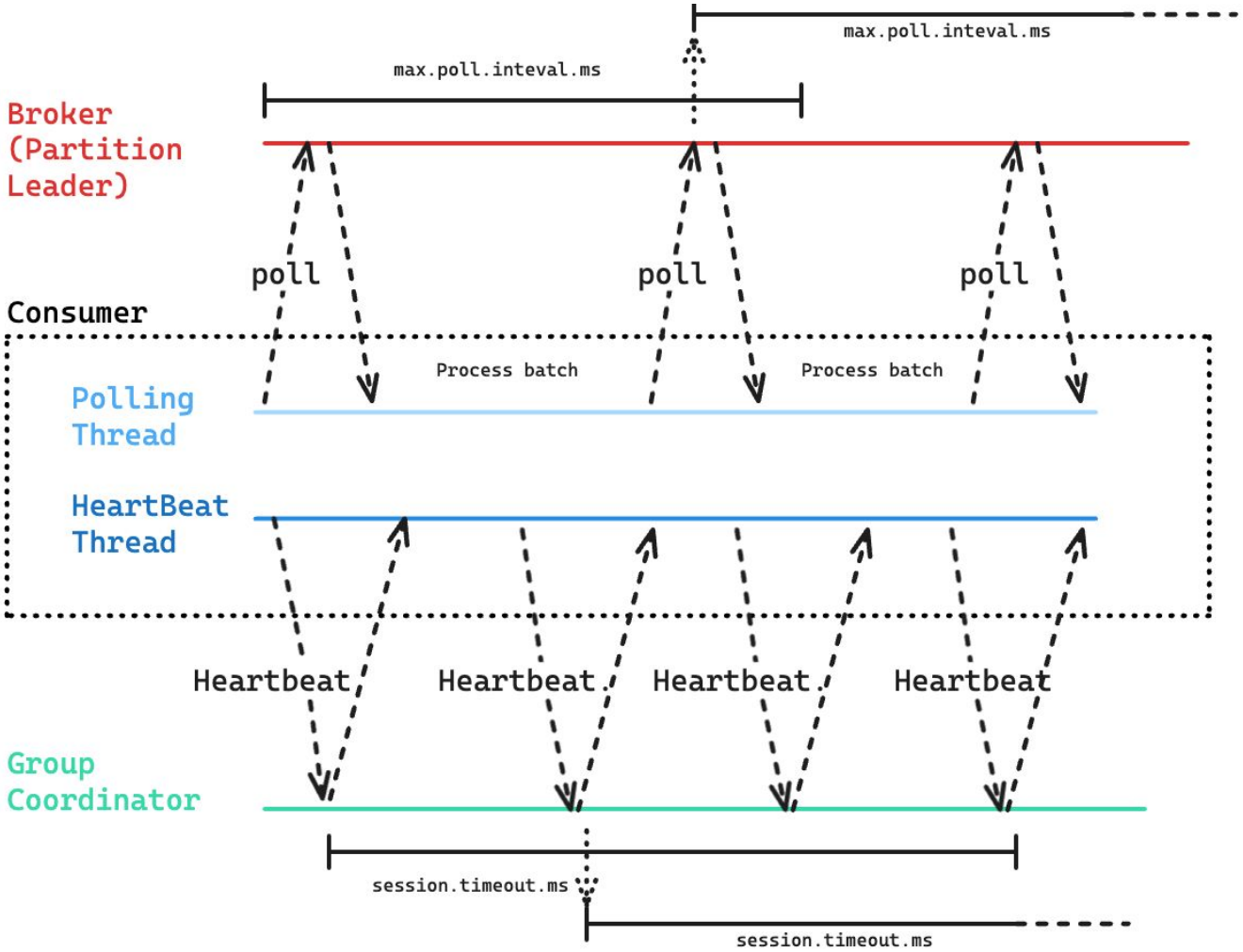


1. heartbeat.interval.ms: is the time between heartbeats signals to the consumer coordinator to indicate the liveness of the consumer.

2. session.timeout.ms: is the max time that the consumer coordinator can tolerate without heartbeat, which after it will consider this consumer is dead.

3. It is recommended to have the session.timeout.ms set as a multiple of the heartbeat interval (the default is 15 times (45/3))


Heartbeat interval
HeartBeat Thread
Group Coordinator
session.timeout.ms
session.timeout.ms
In this case the consumer is still alive, as it send the second heartbeat successfully before the session timeout has passed.
Heartbeat interval
HeartBeat Thread
Consumer is trying to send heartbeat, but it is failing or not able to
Group Coordinator
session.timeout.ms
In this case, the coordinator didn't heard from the consumer for the session timeout interval, then it will consider it dead consumer and will trigger rebalance.


max.poll.inteval.ms
max.poll.inteval.ms
Broker
(Partition
Leader)
poll
poll
poll
Consumer
Process batch
Process batch
Polling
Thread
HeartBeat
Thread
Heartbeat
Heartbeat
Heartbeat
Heartbeat
Group
Coordinator
session.timeout.ms
session.timeout.ms

Advanced Topics - Consumers (Important Timeouts Configs)

Configuration	Description	Default
`session.timeout.ms`	The timeout to detect a consumer has failed, heartbeat is must be received by the group coordinator within this period	45 seconds (>=Kafka 3.0.0) 10 seconds (< Kafka 3.0.0)
`heartbeat.interval.ms`	The interval between heartbeats send by the consumer to the group coordinator, used to ensure consumer is alive.	3 seconds
`max.poll.interval.ms`	The max time between poll invocations, before consumer is considered failed. If passed with no poll, means consumer is stuck.	5 minutes

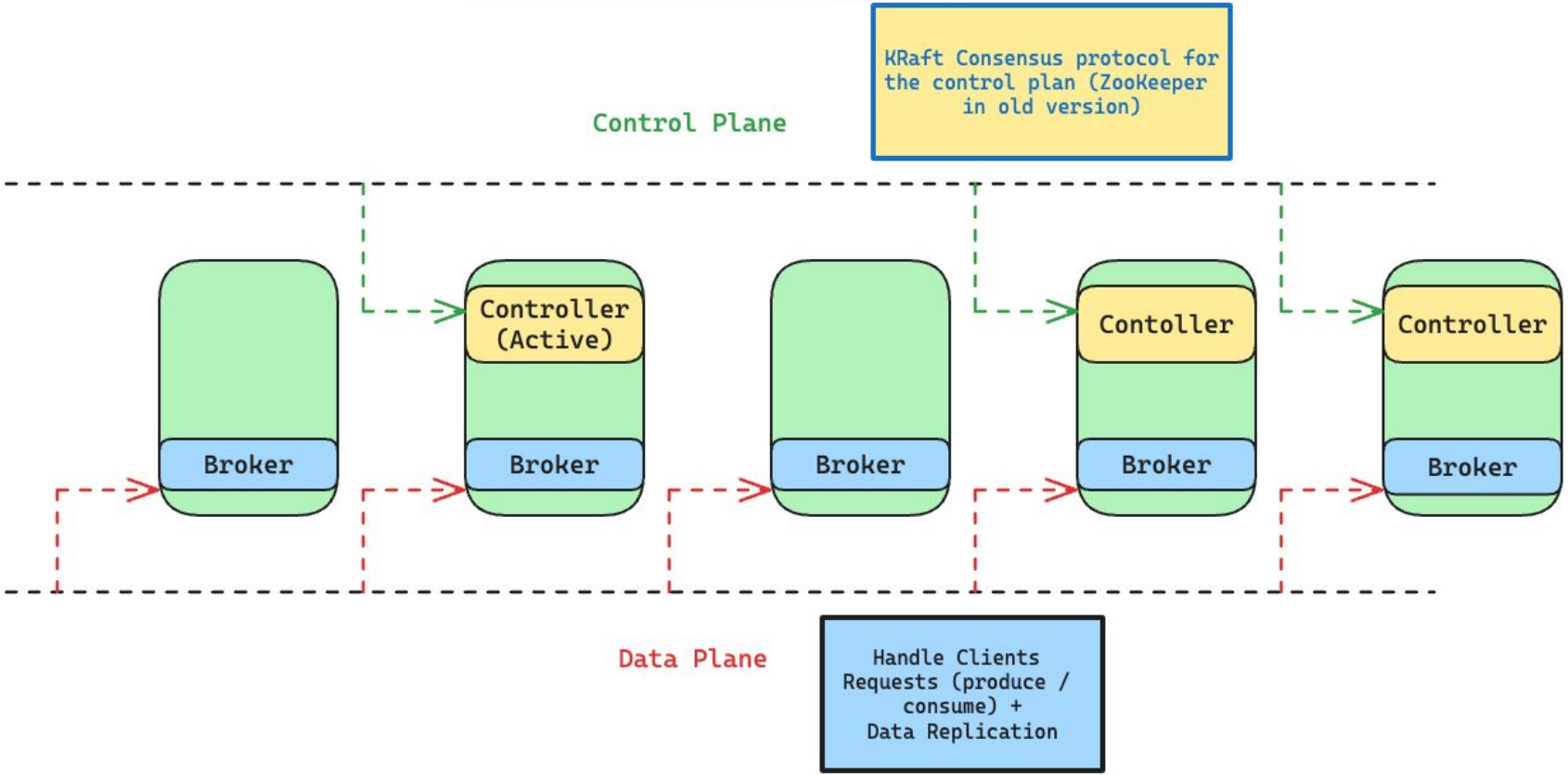
Advanced Topics - Consumers (Other Important Configs)

- `group.id`
- `fetch.max.wait.ms`
- `fetch.min.bytes`
- `fetch.max.bytes`
- `max.partition.fetch.bytes`
- `max.message.bytes (topic or broker)`
- `enable.auto.commit`
- `auto.commit.interval.ms`
- `auto.offset.reset`
- `group.instance.id`
- `Max.poll.records`
- `partition.assignment.strategy`

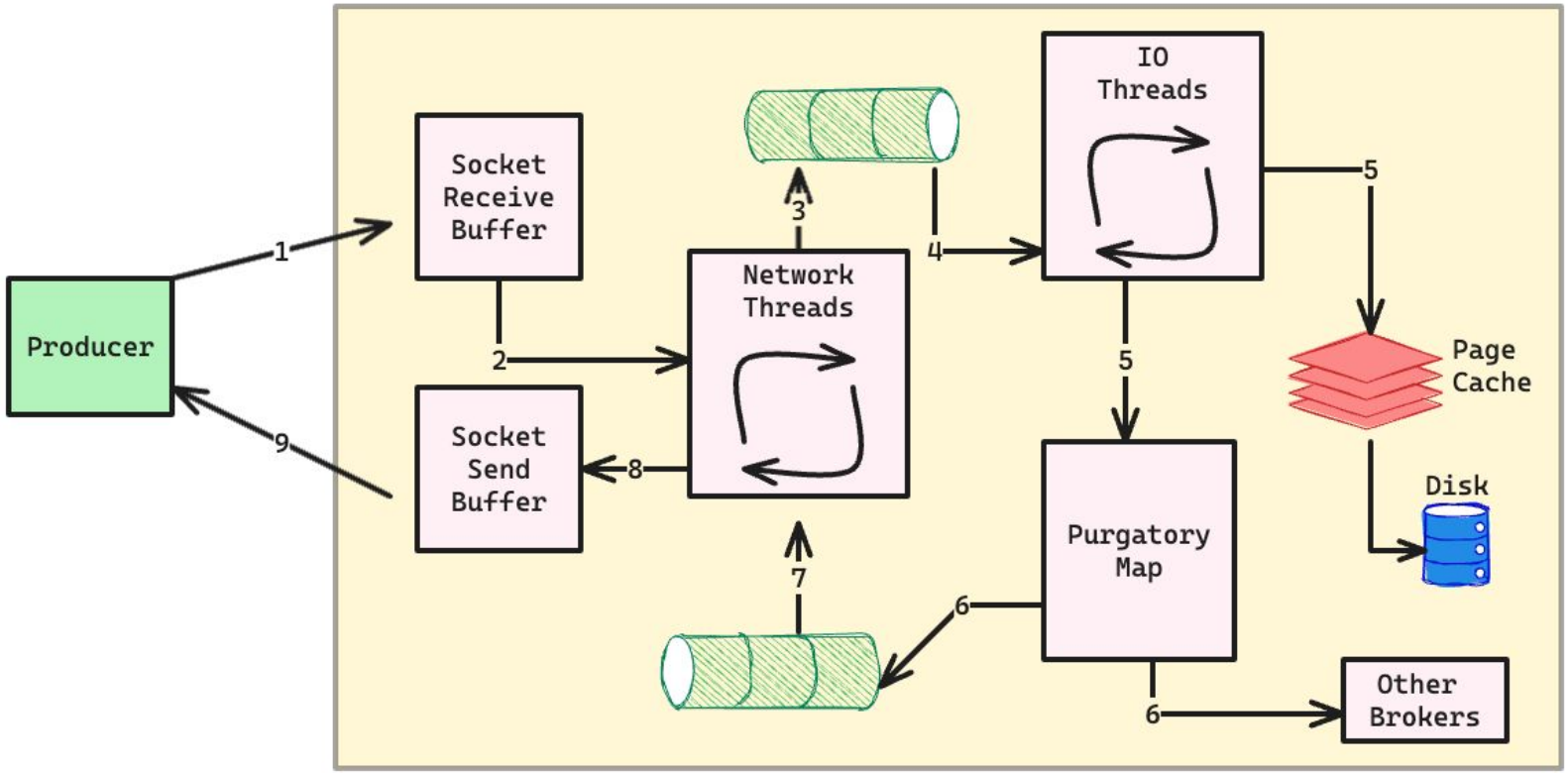
Advanced Topics- Brokers

- In Depth Architecture
- Difference between Partition Leader and Controller Broker
- Important Configs

Broker Architecture



Inside The Broker



Difference between Partition Leader and Controller Broker

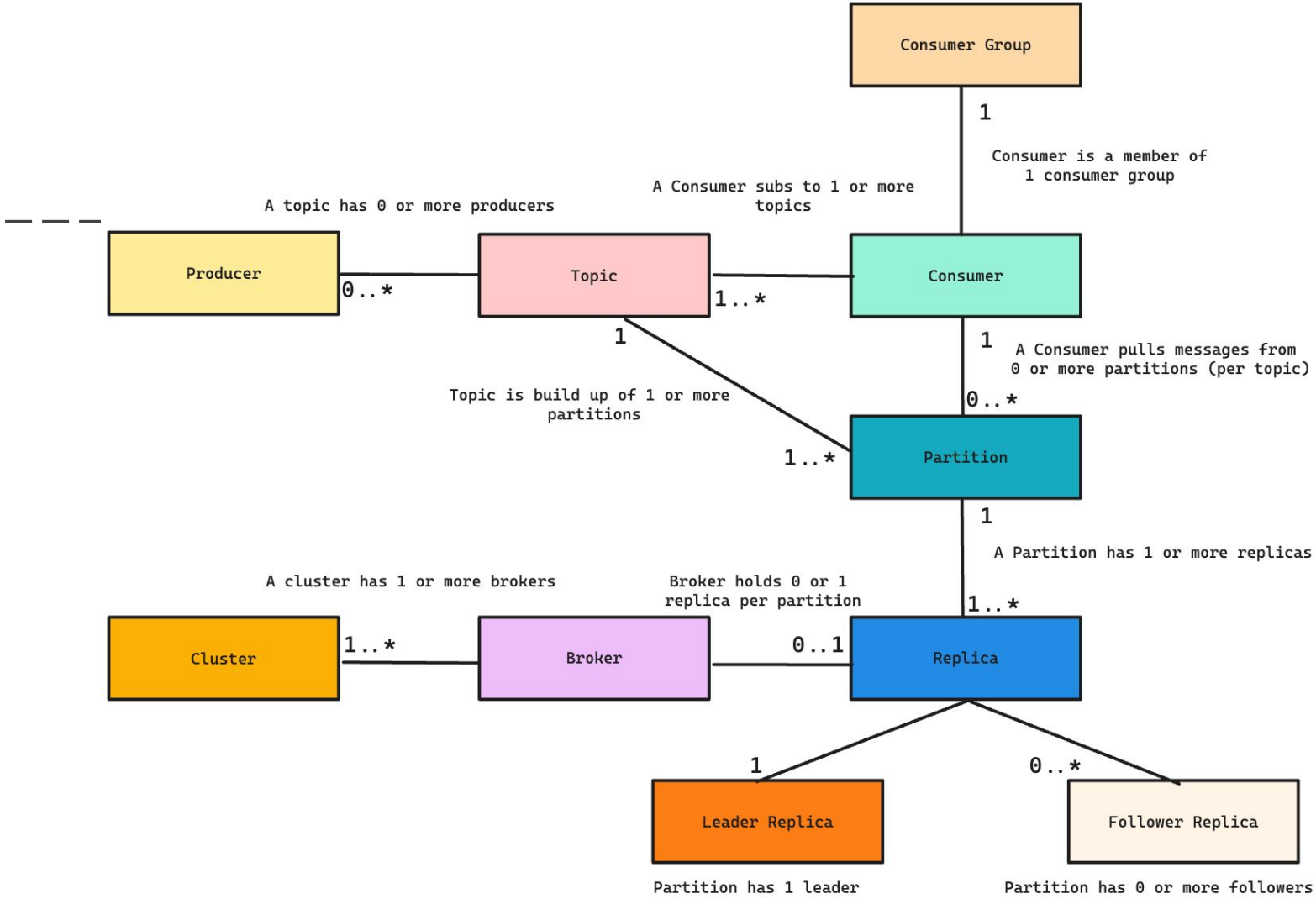
---

1. When a controller is down, a new controller is selected through the Leader Election Process,
2. Broker Leader Election is not the same as Kafka Partition LEader election (Preferred Replica Leader Election).
3. Details:
    1. Regardless the broker is a controller or not, it is a leader for some partitions and follower for others.
    2. when a broker goes down, the replica for which it is leader to will go to another broker via Preferred Replica Leader Election process.
    3. Although we call the process of Zookeeper nominating a follower broker to be a new controller “Leader Election” Process, the controller broker is not a leader.
    4. However the controller is directly responsible for performing PreferredReplicaLeader Election for the new partition leader.
    5. So it is not the same but it is connected.

Broker Important Configs

1. broker.id
2. port
3. log.dirs
4. delete.topic.enable
5. auto.create.topics.enable
6. default.replication.factor
7. num.partitions
8. log.retention.ms
9. log.retention.bytes
10. controller.qourum.voters (kraft)

Wrap-up


Consumer Group
1
Consumer is a member of
1 consumer group
A Consumer subs to 1 or more
topics
A topic has 0 or more producers
Producer
0..*
Topic
1..*
Consumer
1
A Consumer pulls messages from
0 or more partitions (per topic)
1
0..*
Topic is build up of 1 or more
partitions
1..*
Partition
1
A Partition has 1 or more replicas
A cluster has 1 or more brokers
Broker holds 0 or 1
replica per partition
1..*
Cluster
1..*
Broker
0..1
Replica
1
0..*
Leader Replica
Follower Replica
Partition has 1 leader
Partition has 0 or more followers

Q&A

Lab

https://shorturl.at/Zrtqt

Working with Kafka from the Command Line

---

1. Running Kafka on Dev Environment
2. Topic Design and Management
3. Kafka Producer From CLI
4. Kafka Consumer From CLI

Kafka Workshop [Day-2]

Kafka Streams - What is Kafka Streams?

- **Definition:** Kafka Streams is a powerful, lightweight library for building stream processing applications and microservices.
- **Purpose:** It allows processing of data in real-time from **Kafka topics** and can be used to build applications that transform, aggregate, and enrich data.
- **Key Features:**
  - **Scalability:** Easily scales horizontally by running multiple instances.
  - **Fault Tolerance:** Ensures data is processed exactly once (exactly-once semantics).
  - **Flexibility:** Supports both stateless and stateful processing.

Kafka Streams - Table / Stream Duality

Concept Overview:

- **Stream**: A continuous flow of data records (e.g., CDRs being generated in real-time).
- **Table**: A stateful representation of data, capturing the current state (e.g., the latest CDR information for each customer).
- **Duality**: Streams can be materialized into tables, and tables can be turned into streams, allowing for flexible data processing and state management.

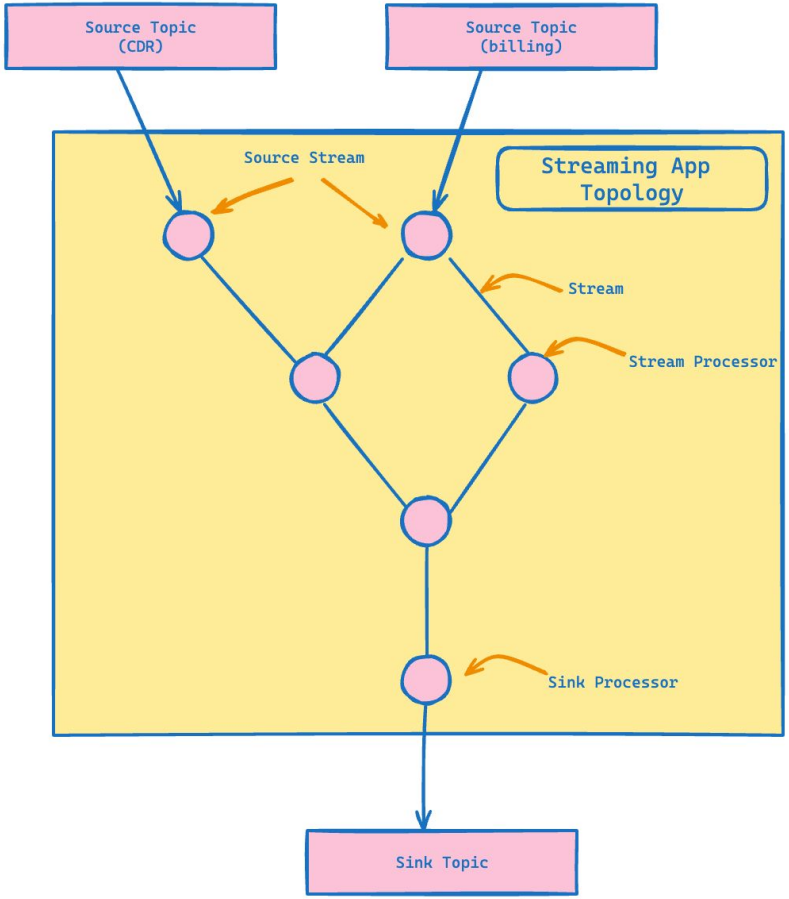
Kafka Streams - API Overview

- **Core Concepts:**
  - **Streams and Tables:** Streams (continuous data flows) vs. Tables (state representation).
  - **Topology:** Defines processing logic, composed of processors and state stores.
  - **KStream, KTable, and GlobalKTable:** Main abstractions for processing streams of records.
- **Stateless Operations:** `filter()`, `map()`, `flatMap()`
- **Stateful Operations:** `aggregate()`, `join()`, `window()`
- **DSL vs. Processor API:**
  - **DSL (Domain Specific Language):** Simplified, high-level API for common stream processing operations (e.g., filtering, mapping, grouping, joining…).
  - **Processor API:** Low-level API for more complex and custom processing, offering fine-grained control over processing logic.

Kafka Streams App Structure

Any Kafka Streaming App consists of the following:

1. Source Topic
2. Processing Topology
   a. Source Processor
   b. Sink Processor
3. Sink / Destination Topic



Kafka Streams - Use Cases and Examples

- **Common Use Cases:**
  - **Real-time Analytics:** E.g., monitoring and alerting.
  - **ETL Processes:** Extract, transform, and load data in real-time.
  - **Data Enrichment:** Enhancing data streams with additional information.
  - **Event Sourcing:** Building event-driven applications.
- **Example:**
  - Real-time fraud detection in financial transactions.
  - Monitoring sensor data in IoT applications.

Kafka Streams - Streams StateStore

- **State Stores:** For Aggregation Kafka streams store the state of aggregations, joins, and other stateful operations in these state store for Fault Tolerance and scalability
- **ChangeLog topic:** To ensure fault tolerance and scalability, Kafka Streams backup the state store data to a kafka topic changelog topics , these topics logs every update to the state store, that allows the state to be rebuild in case of failures and rebalancing.
- **Naming pattern:** <app_id>-<store_name>-changelog

Java Copy Caption ···

```
cdr-app-960739767-KSTREAM-AGGREGATE-STATE-STORE-0000000001-changelog
cdr-app1005786326-KSTREAM-AGGREGATE-STATE-STORE-0000000001-changelog
cdr-app1068545878-KSTREAM-AGGREGATE-STATE-STORE-0000000001-changelog
```

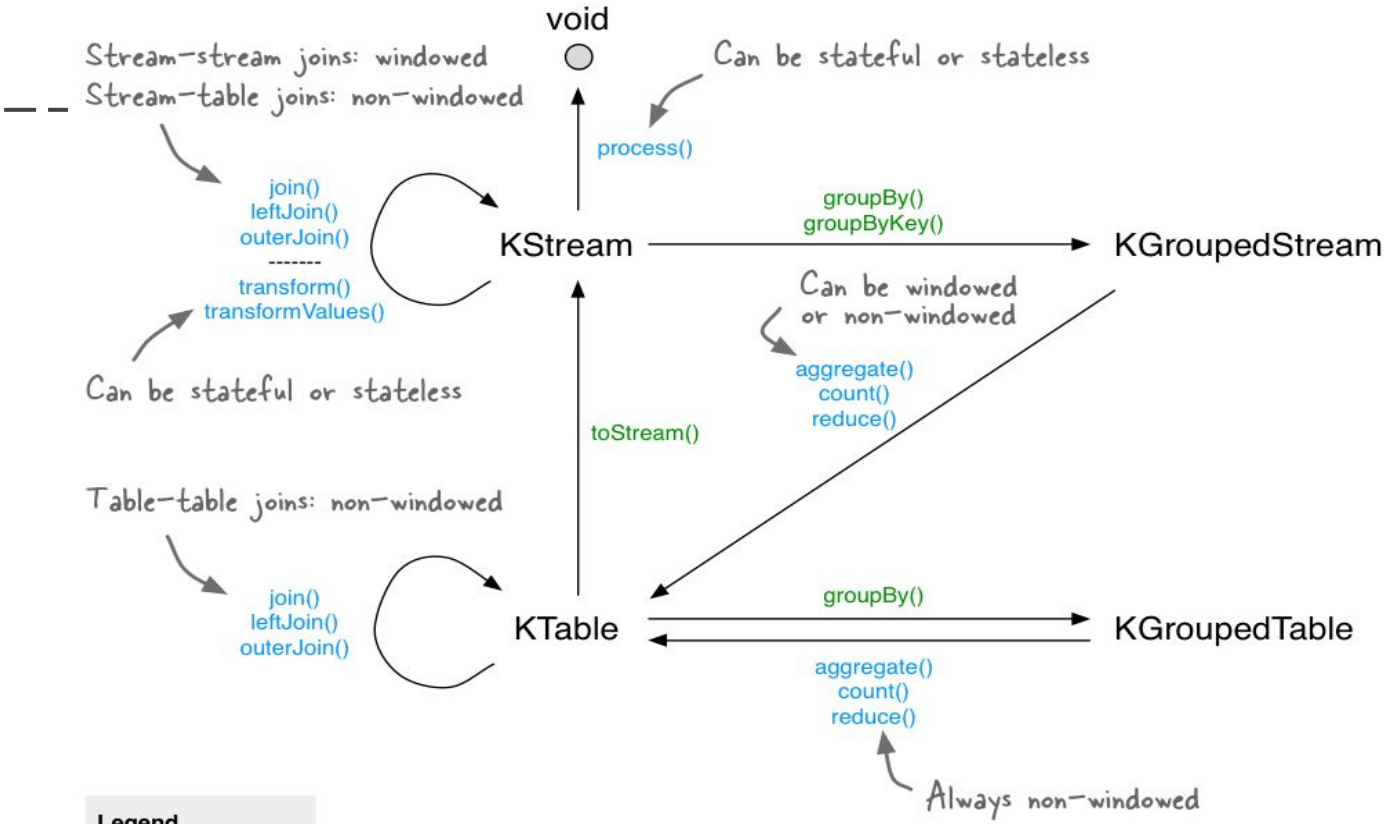
Kafka Streams - Stateless functions

- Filter
- Map
- Branch
- Flatmap
- Foreach
- Merge
- Peek
- Print

Kafka Streams - Stateful functions

- Count
- Reduce
- Joins
- windowing

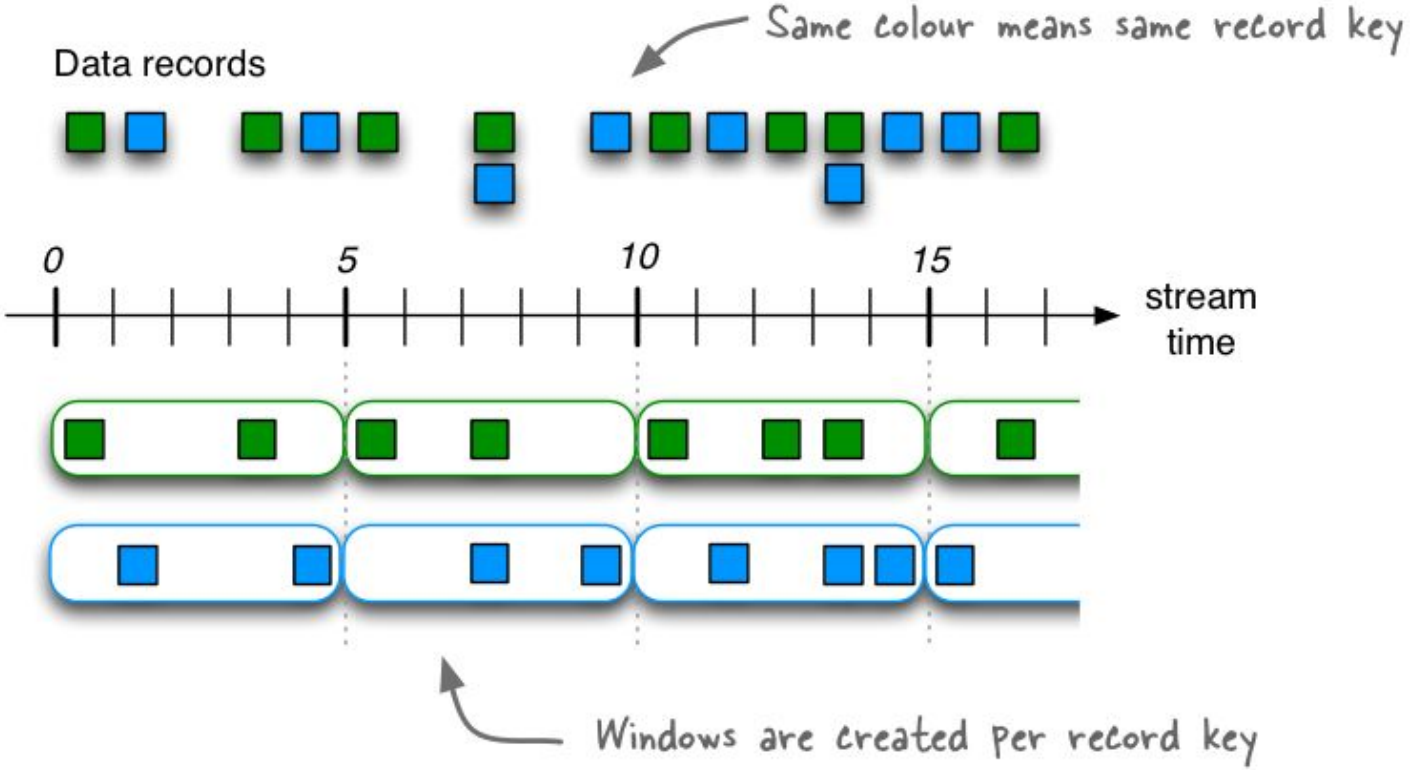
Kafka Streams - Stateful functions

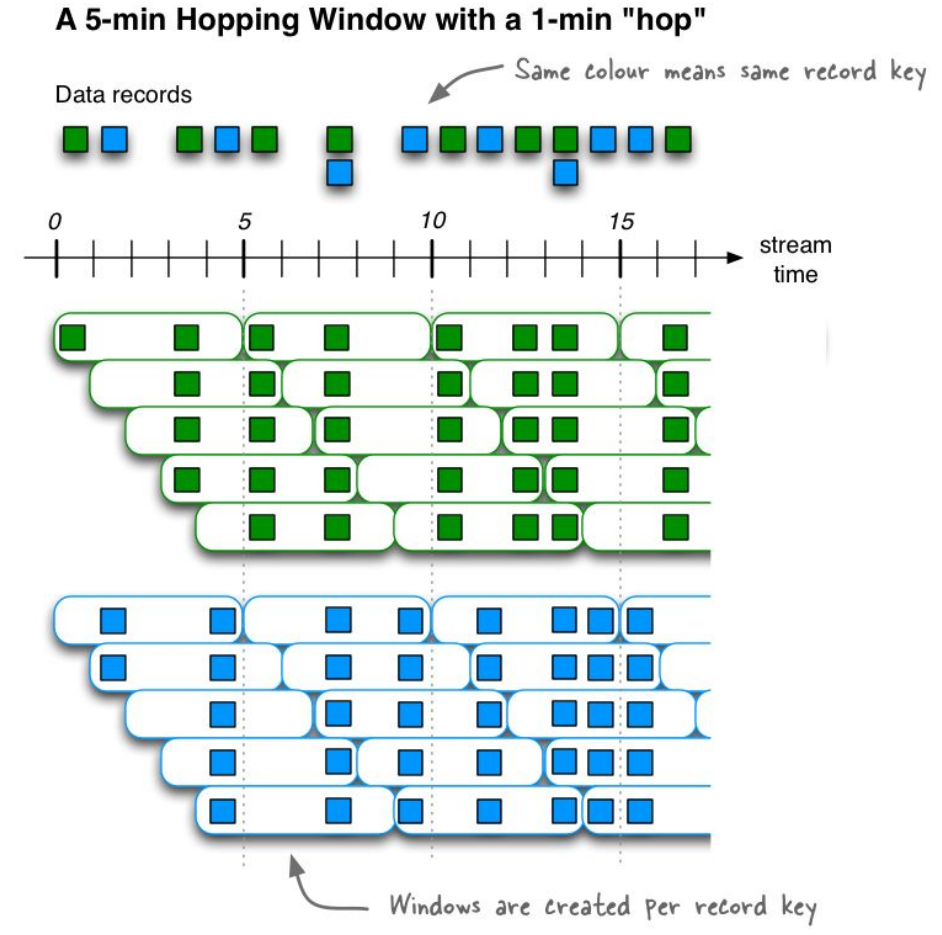


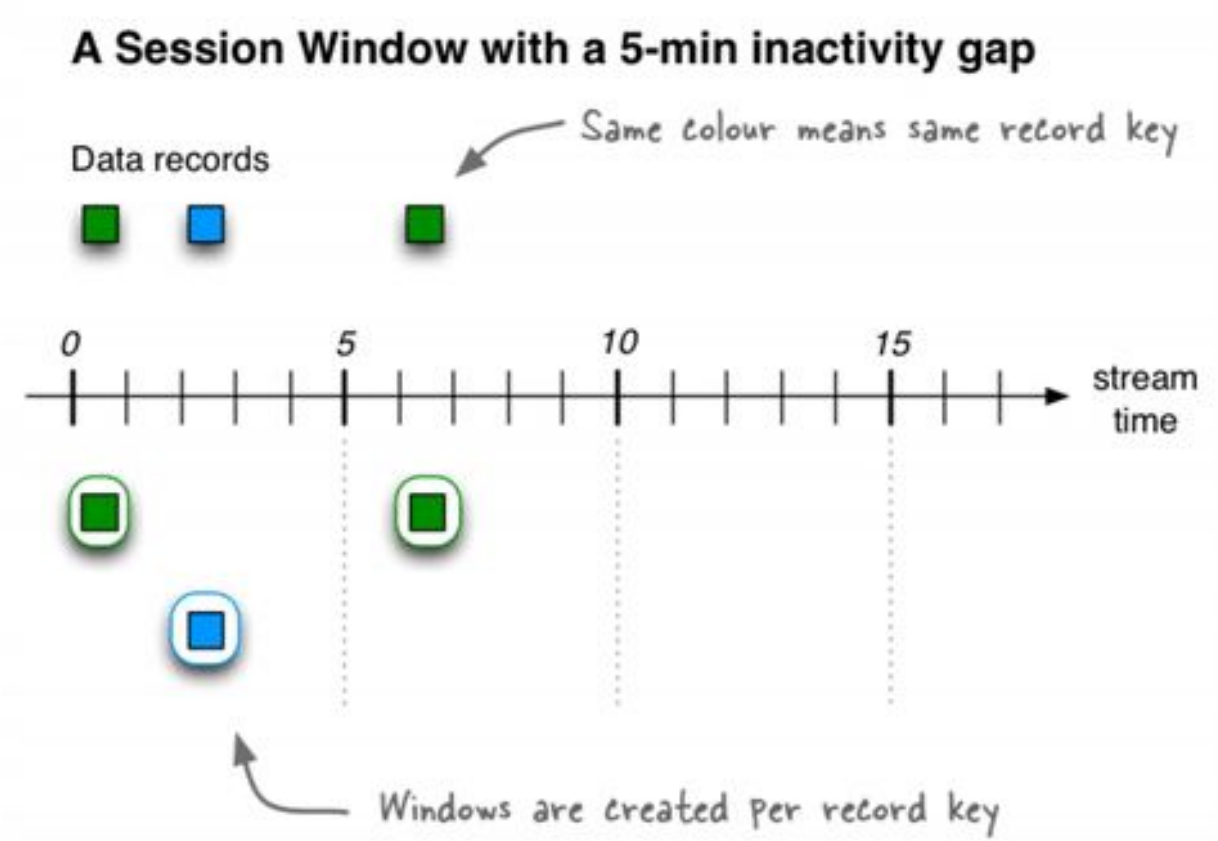
Windowing types

Window type	Behavior	Description
Tumbling Window	Time-based	Fixed-duration, non-overlapping, gap-less windows
Hopping Window	Time-based	Fixed-duration, overlapping windows
Session Window	Session-based	Dynamically-sized, non-overlapping, data-driven windows

A 5-min Tumbling Window




A 5-min Hopping Window with a 1-min "hop"
Same colour means same record key
Data records
0
5
10
15
stream time
Windows are created per record key


A Session Window with a 5-min inactivity gap
Same colour means same record key
Data records
0
5
10
15
stream time
Windows are created per record key

Wrap-up

What we have learned so far?

1.

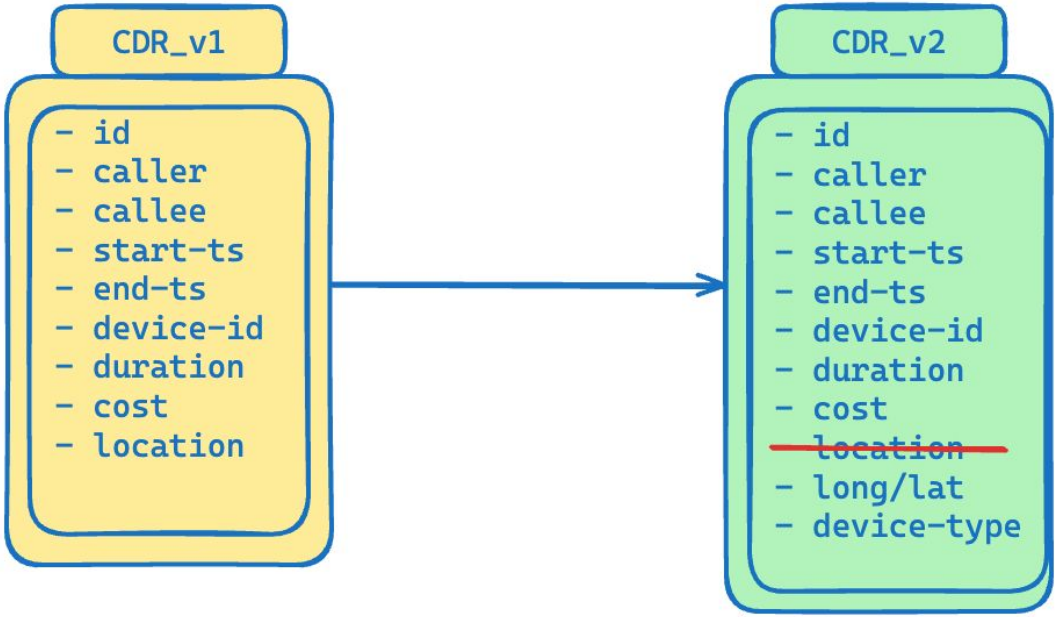
Q&A

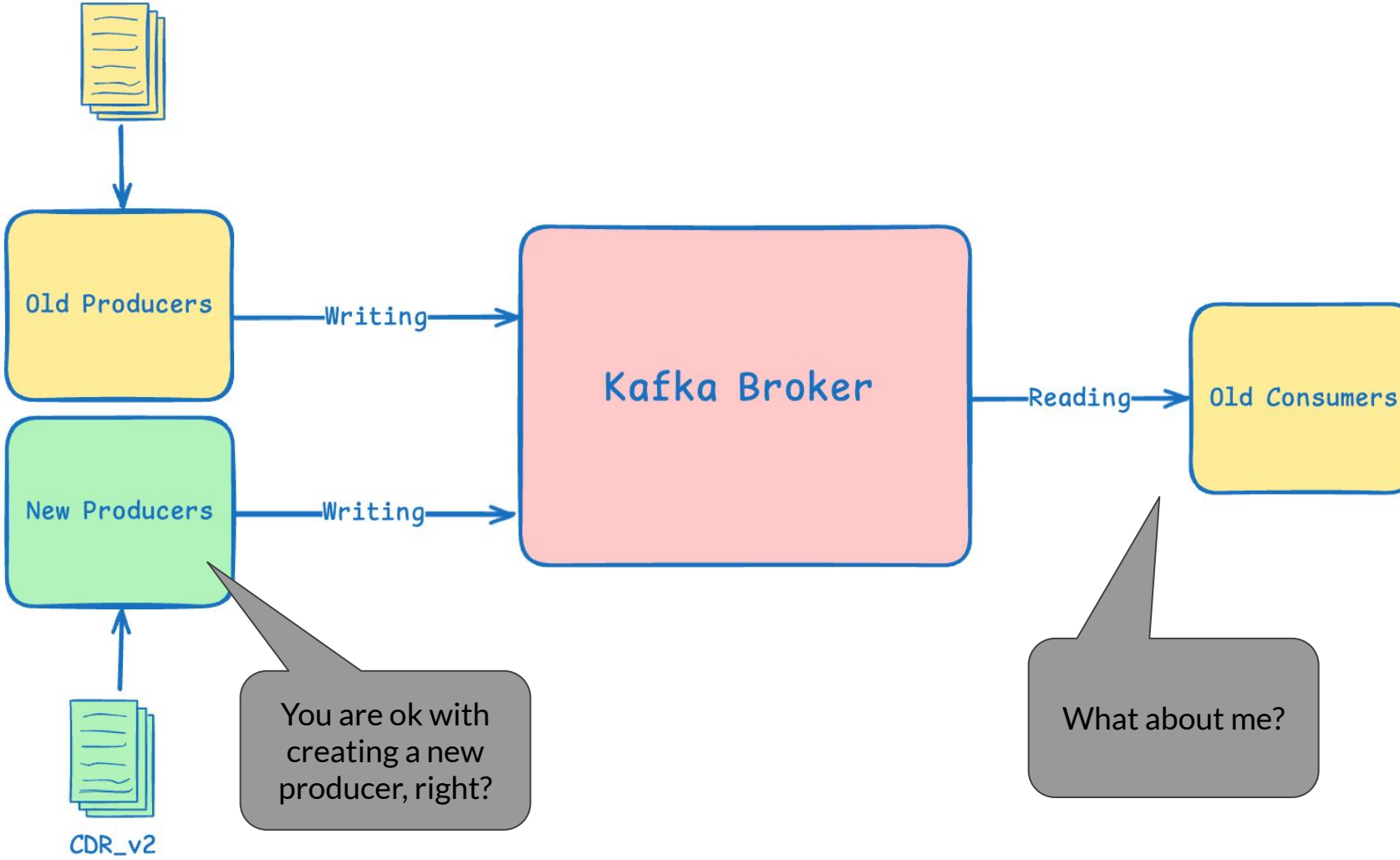
Lab

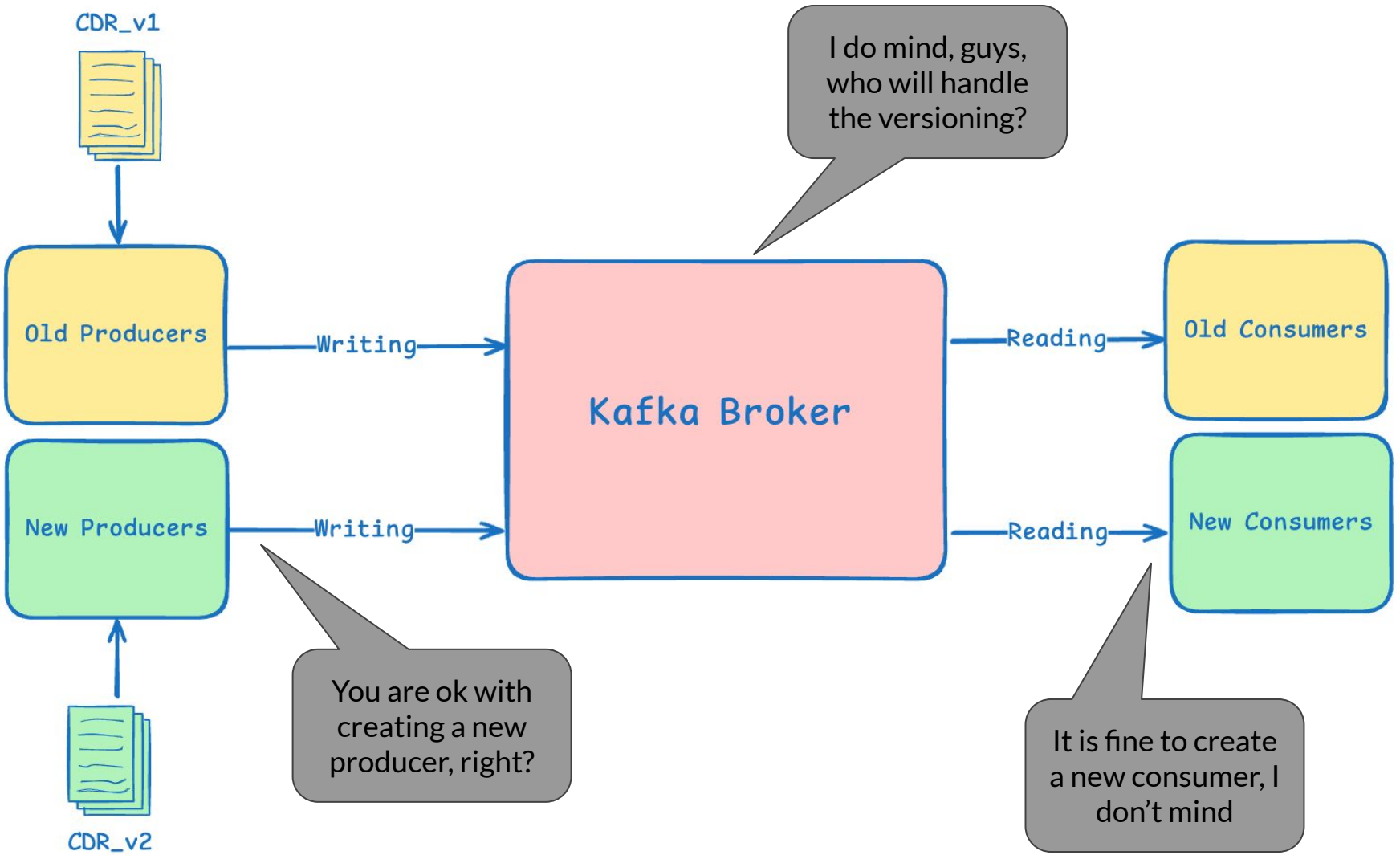
Kafka Workshop [Day-3]

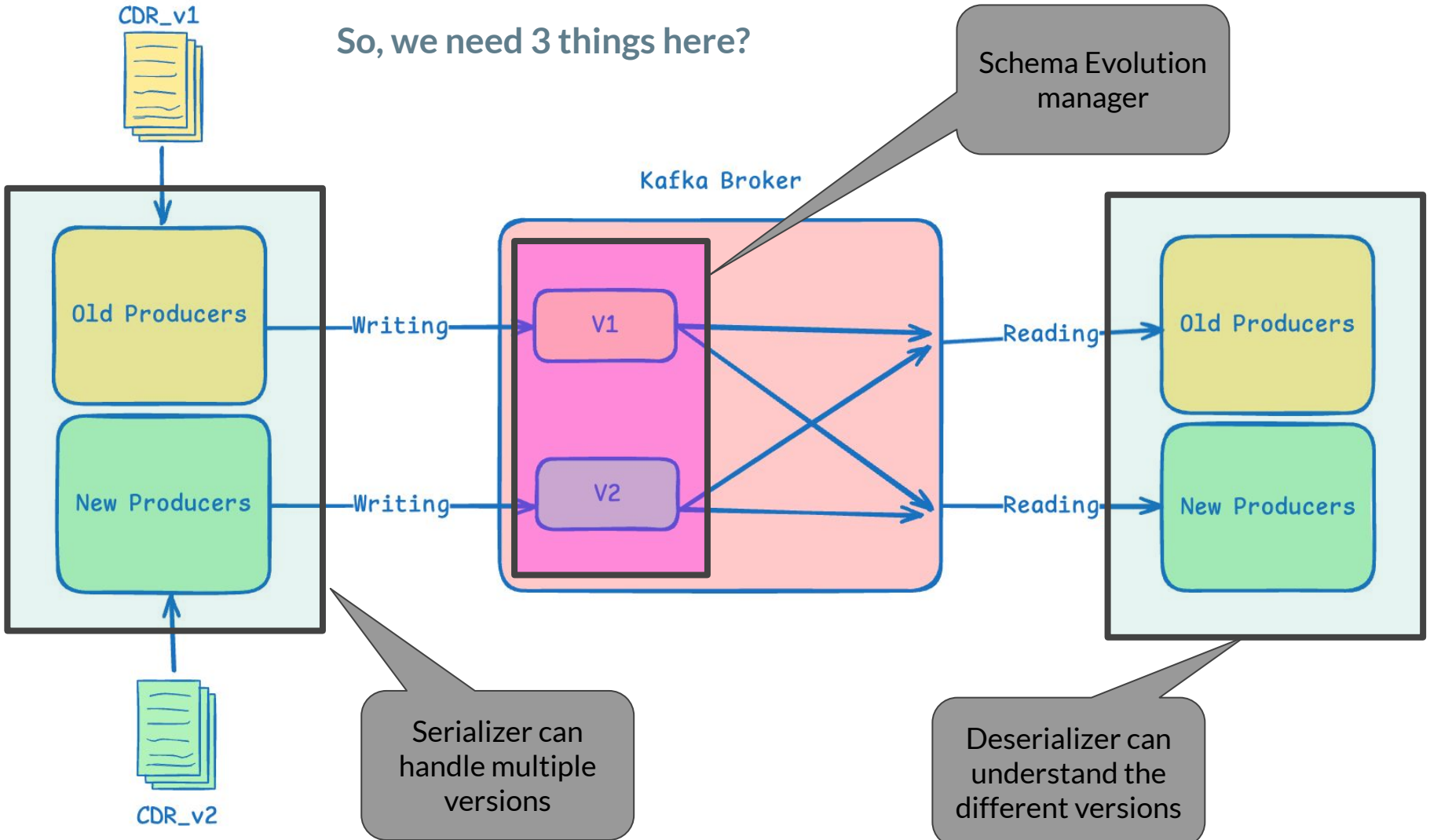
Kafka Schema Registry

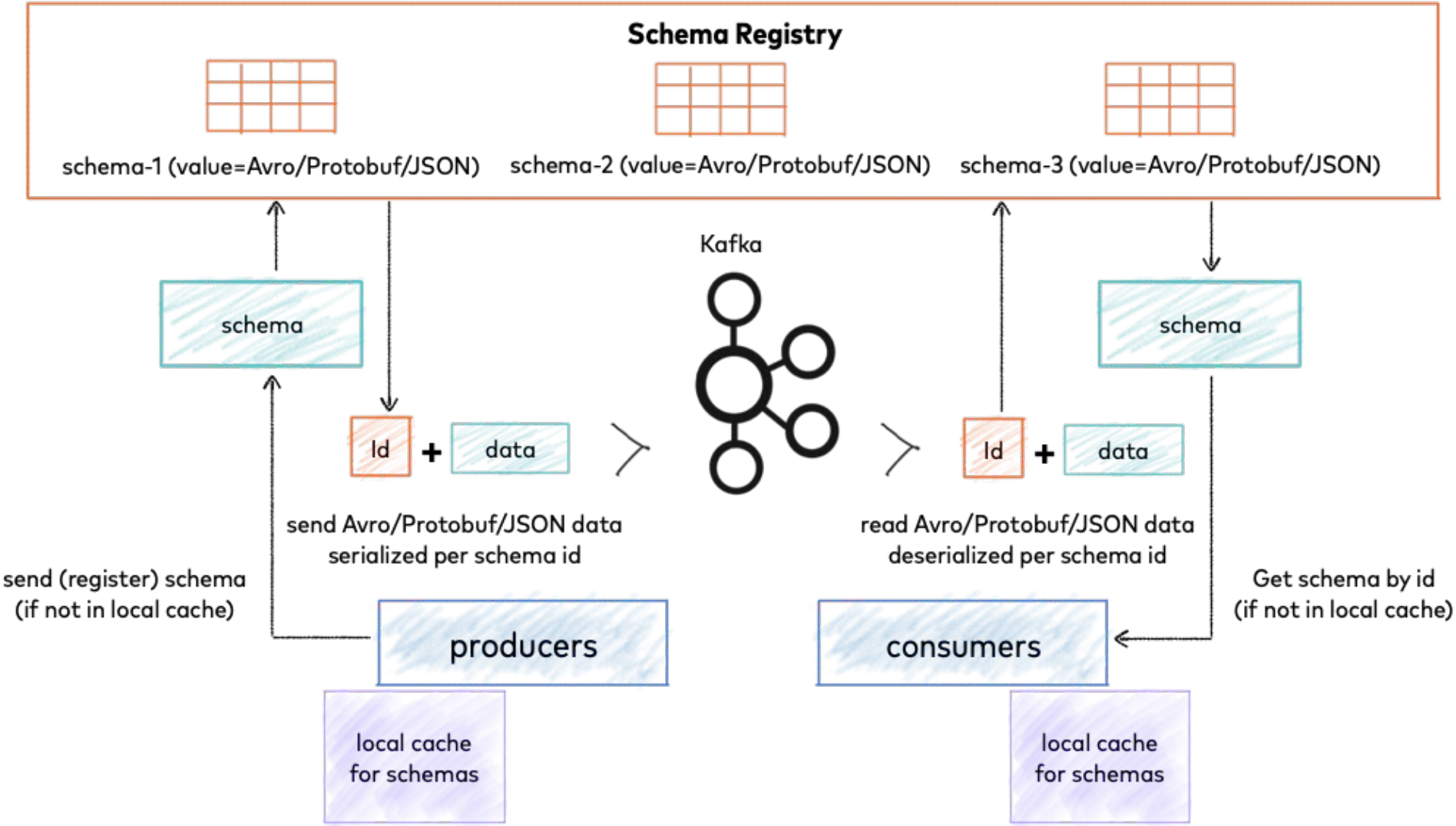
Schema Evolution Problem




Old Producers
Writing
Kafka Broker
Reading
Old Consumers
New Producers
Writing
You are ok with creating a new producer, right?
What about me?
CDR_v2


CDR_v1
I do mind, guys, who will handle the versioning?
Old Producers
Writing
Kafka Broker
Reading
Old Consumers
New Producers
Writing
Reading
New Consumers
You are ok with creating a new producer, right?
It is fine to create a new consumer, I don't mind
CDR_v2


So, we need 3 things here?
CDR_v1
CDR_v2
Old Producers
New Producers
Writing
Writing
Kafka Broker
V1
V2
Reading
Reading
Old Producers
New Producers
Schema Evolution manager
Serializer can handle multiple versions
Deserializer can understand the different versions


Schema Registry
schema-1 (value=Avro/Protobuf/JSON)
schema-2 (value=Avro/Protobuf/JSON)
schema-3 (value=Avro/Protobuf/JSON)
Kafka
schema
schema
Id
+
data
Id
+
data
send Avro/Protobuf/JSON data
serialized per schema id
read Avro/Protobuf/JSON data
deserialized per schema id
send (register) schema
(if not in local cache)
Get schema by id
(if not in local cache)
producers
consumers
local cache
for schemas
local cache
for schemas

Detailed Flow

On Producer Side:

1. Producer serialize the object using avro
2. Send it to the schema registry defined in the producer configs
3. When producing records, the ProducerRecrod is passed to the schema registry.
4. Schema registry store (if-new) the schema and return to the producer the date along with the schema ID.
5. Then the producer send the data + SchemaId to the broker

On Consumer Side:

1. Consumer poll the message from kafka.
2. Since the deserializer is configured to be KafkaAvroDeserializer, it will look for the schema ID in the message.
3. Retrieve the schema from the schema registry by the ID.
4. Deserialize to the corresponding Java Object

Important things to consider?

---

1. Do I need to change all my current Producers?
2. What about downstream consumers?
3. Can both versions live together for awhile?
    a. If yes, do the system I am using will make me capable of doing so???
4. In normal case, with on schema management, what would be the case?
    a. Change Producers (Serializer).
    b. Change consumers (Deserializer).
    c. Handle in-flight requests, or face a downtime.
5. When do I need to ignore the whole hassle of schema migrations?, When:
    a. Data loss is not a big deal
    b. Events can be replayable with the new schema (continuously regenerating events on scheduled interval).
6. QQ: Is there a way to handle multiple versions without schema versioning???

What is Schema Registry?

- **Purpose:** Schema Registry manages and governs schemas for structured data in distributed systems. It acts as a central repository for schema versions, ensuring that all data adheres to a predefined structure and format.
- **Key Features:**
  - **Schema Storage:** Stores and versions schemas independently from the services that use them.
  - **Schema Validation:** Ensures data compatibility and enforces schema evolution rules (e.g., backward, forward, full compatibility).
  - **Compatibility Checks:** Prevents schema changes that could break data communication between services.

Why Use Schema Registry?

- **Data Quality and Consistency:** Guarantees that data conforms to registered schemas, reducing errors and improving quality.
- **Efficient Data Governance:** Centralizes schema management, making it easier to meet regulatory and audit requirements for data formats.
- **Decoupling Services:** Allows services to independently evolve while maintaining compatibility with shared data contracts.

Integration with Kafka

Role in Kafka Ecosystem: Schema Registry is often used with Apache Kafka to ensure that the messages sent through Kafka topics conform to a schema.

How It Works:

- Producers write data to Kafka topics using schemas registered in the Schema Registry.
- Consumers read data from Kafka topics and automatically validate it against the schema obtained from the Schema Registry.

Where is it used in Kafka:

- Producer / Consumer
- Kafka Connect
- KSQL
- Flink Streaming

Benefits of Using Schema Registry with Kafka

- **Reliable Data Streams:** Enhances the reliability of Kafka data streams by ensuring all messages adhere to the schema.
- **Schema Evolution:** Safely manages changes in data schema without disrupting existing applications, using schema versioning.
- **Interoperability:** Supports multiple schema formats like Avro, JSON Schema, and Protobuf, promoting interoperability across different systems and programming languages.

Schema Evolution - Compatibility Modes

Compatibility Type	Schema Resolution
Backward	Consumers using the **new** schema version can read data written with previous schema version.
Forward	Consumers using the **Previous** schema version can read data written with new schema versions.
Full	Both Forward & Backward
None	No Compatibility Checks

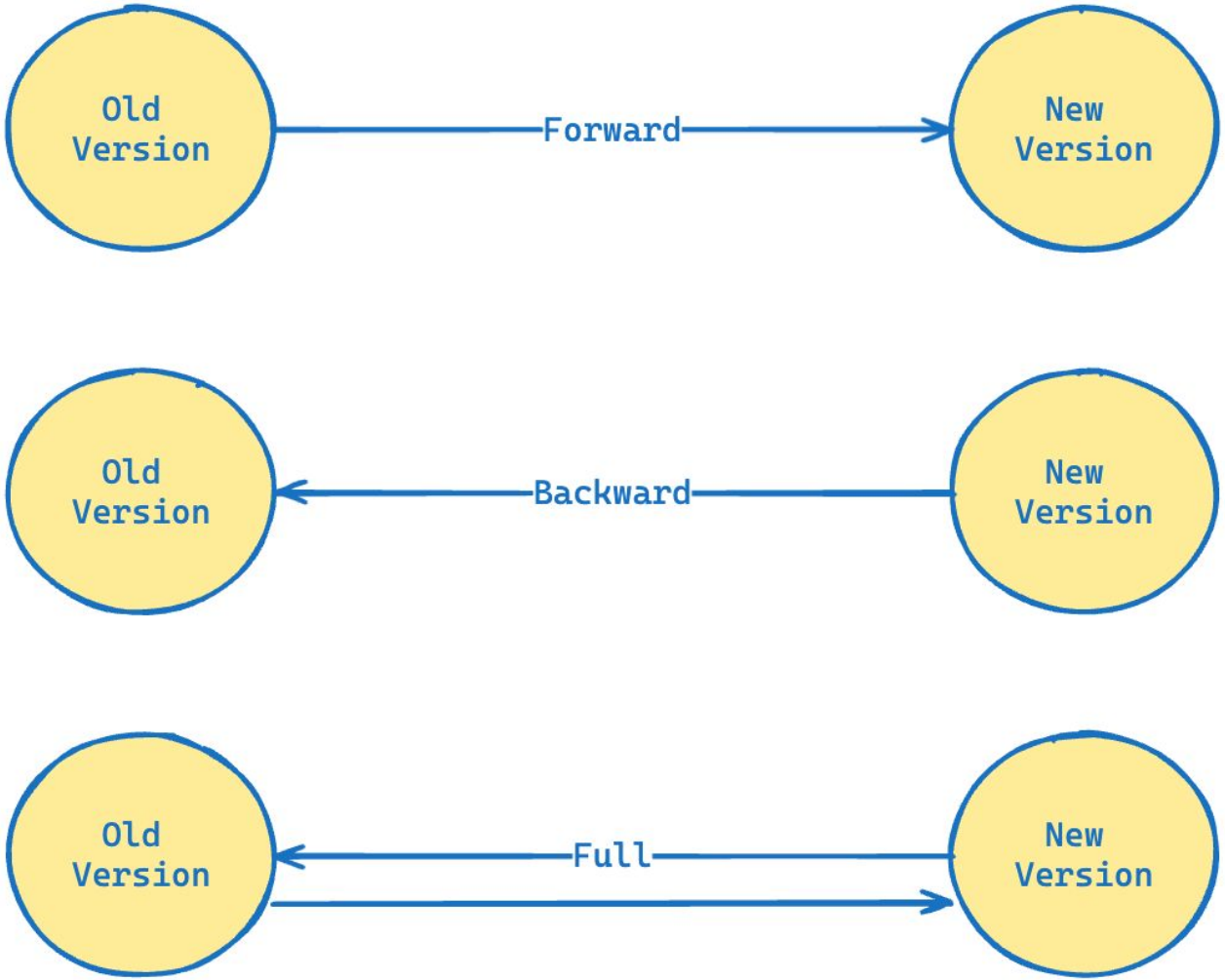
Schema Evolution - Compatibility Modes

Compatibility Type	Schema Resolution
Backward_TRANSTIVE	Consumers using the **new** schema version can read data written with **ALL** previous schema version**s**.
Forward_TRANSTIVE	Consumers using the **Previous** schema version can read data written with ALL later schema version**s**.
Full__TRANSTIVE	Both Forward & Backward

Schema Evolution - Order Of Service Updates

BACKWARD	FORWARD	FULL
- Delete fields - Add fields with default values	- Delete fields with default values - Add Fields	- Delete fields with default values - Add Fields with default values
Update consumer clients first	Update producer clients first	Doesn't matter the order of update

Compatibility Type	Changes allowed	Check against which schemas	Upgrade first
`BACKWARD`	• Delete fields • Add optional fields	Last version	Consumers
`BACKWARD_TRANSITIVE`	• Delete fields • Add optional fields	All previous versions	Consumers
`FORWARD`	• Add fields • Delete optional fields	Last version	Producers
`FORWARD_TRANSITIVE`	• Add fields • Delete optional fields	All previous versions	Producers
`FULL`	• Add optional fields • Delete optional fields	Last version	Any order
`FULL_TRANSITIVE`	• Add optional fields • Delete optional fields	All previous versions	Any order
`NONE`	• All changes are accepted	Compatibility checking disabled	Depends


Old Version
Forward
New Version
Old Version
Backward
New Version
Old Version
Full
New Version

Avro vs Json vs Protobuf

Feature	Avro	JSON	Protobuf
Format Type	Binary (also supports JSON)	Text-based	Binary
Schema	Required (can be embedded)	Schema-less	Required (separate from data)
Schema Evolution	Full support	Not applicable	Supported with restrictions
Usability	Good, with code generation	Excellent, human-readable	Good, with code generation
Performance	High (compact, fast serialization)	Lower (text-based, larger size)	High (very compact and fast)
Interoperability	High with dynamic schema	High, universal support	Moderate (requires .proto files)
Typical Use Cases	Data serialization in Hadoop, Kafka	Web services, simple messaging	High-performance environments like gRPC
Popularity	Widely used in big data ecosystems	Ubiquitous in web and APIs	Common in systems requiring efficient, scalable communication
Compatibility	Excellent forward and backward compatibility	N/A	Forward and backward compatibility with careful schema management
Tooling and Support	Good, part of Apache ecosystem	Excellent, universal support	Good, officially supported by Google, extensive tooling

Avro vs Protobuf

Avro:

- **Dynamic Schemas:** Avro schemas are defined in JSON, which allows for more dynamic and flexible schema definitions. This flexibility is particularly useful in environments where schemas may need to evolve rapidly or are generated dynamically.
- **Complex Types:** Supports records, enums, arrays, maps, unions, and fixed types.
- **Schema Evolution:** supports forward and backward compatibility. This is crucial in distributed systems where producers and consumers of data might not always be updated at the same time.

Protobuf:

- **Static Type System:** Protobuf requires more rigid schema definitions, which are compiled from .proto files into code specific to the programming language being used. This approach ensures very tight control over data structure but reduces flexibility.
- **Data Types:** Supports basic scalar types (integers, floating-point numbers, booleans, strings, bytes) and complex types (enums, maps, oneofs, repeated fields for arrays). it might feel less naturally adaptable to certain complex structures compared to Avro.
- **Less Flexible Schema Evolution:** Protobuf also supports schema evolution but requires careful management of field numbers and rules about how fields can be added or changed (e.g., fields should not be removed; instead, they are deprecated).

Always start with a Schema Registry

Starting from the beginning of a project with Schema Registry is a best practice for several reasons, some of which are already explained in previous sections:

- Prevents data inconsistency and increases data quality.
- Simplifies data governance.
- Reduces development time. Starting with Schema Registry eliminates the need for custom code to manage and validate schemas. This also cuts costs and increases development productivity.
- Facilitates collaboration by providing a standard interface for sharing schemas across different teams and applications. This facilitates collaboration and helps to avoid potential data integration issues.
- Improves system performance. Schema Registry validates and optimizes schemas for efficient data exchange, which can improve system performance and reduce processing time.

Sending Multi-event types to in Kafka

---

1. It's sometimes advantageous to produce distinct but related event types to the same topic, e.g., to guarantee the exact order of different events for the same key, or very low traffic for multiple events.
2. **Example:** User activity Tracking for all actions on the system.
3. **QQ:** What would be the problems from this approach?

Tips for Schema Registry

1. Always use Schema registry in new projects.
2. For production env, it is recommended to suppress auto register schema **auto.register.schemas=false** and instead create the schema manually to the schema registry
3. Make a good use of Schema context.
    a. It is a kind of a sub-registry that group related subjects together.
4. Choose the right compatibility approach. (Backward, forward, Full, None).
5. Avoid removing fields, try to add new fields instead, as much as you can.
6. Monitor the schema metrics (requests, error rates..)
7. Cache Schemas in Producers and Consumers.
    a. Confluent Avro Serde do the job for you ;)
8. Avoid removing/renaming existing fields.
    a. **In case of renaming:** Use Avro aliases instead
9. Assign Default value for fields might be removed in the future.

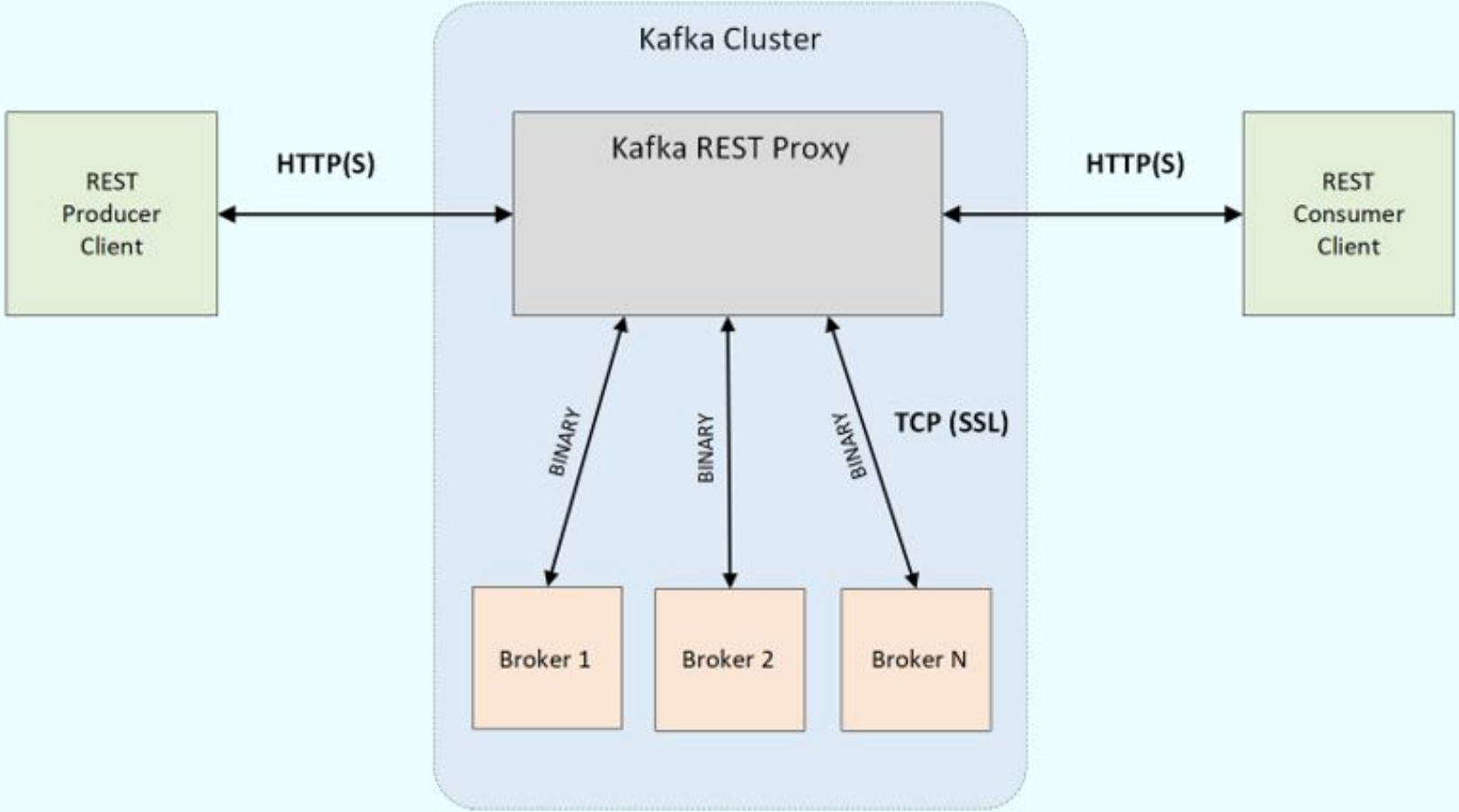
When not to use Schema Registry

- 100% Schema will not change.
- Working on a mission critical system and latency is crucial (ms sensitive).
- Using unsupported data serialization scheme.

Kafka Rest Proxy

What is Kafka Rest Proxy

- Kafka REST Proxy allows you to produce and consume messages over a RESTful HTTP interface as an alternative to the traditional Kafka client APIs. This is part of the Confluent REST Proxy that provides a flexible gateway to a Kafka cluster.
- It is not a mandatory component of Kafka setup.


Kafka Cluster
REST Producer Client
HTTP(S)
Kafka REST Proxy
HTTP(S)
REST Consumer Client
BINARY
BINARY
BINARY
TCP (SSL)
Broker 1
Broker 2
Broker N

High Availability and Scalability

- **Multiple Instances:** Deploy multiple instances of the Kafka REST Proxy to enhance availability and manage larger volumes of client connections.
- **Stateless Design:** The REST Proxy’s stateless architecture supports easy horizontal scaling by adding more instances without the need for complex reconfiguration.

Load Balancing and Stickiness

- **Load Balancing:** Deploy Kafka REST Proxy behind a load balancer to distribute client requests evenly across multiple instances, optimizing resource utilization and response times.
- **Session Stickiness:**
  - **While Kafka REST Proxy is stateless**, ensuring "stickiness" in load balancing can be critical, especially when dealing with consumer group management and offset commits in long polling scenarios.
  - **Why Sticky Sessions?** Stickiness ensures that all requests from a particular client are directed to the same REST Proxy instance during a session. This can be beneficial for maintaining a coherent view of client state, particularly when a client is involved in a specific consumer group operation where offsets and session states are important.
  - **Implementation:** This can be configured in the load balancer (e.g., Nginx, HAProxy) using session cookies or IP-based persistence to ensure that once a consumer connects to a specific proxy instance, it maintains that connection for the duration of its session.

Supported Message Formats

- The REST Proxy can read and write data using :
  a. JSON
  b. Raw bytes encoded with base64
  c. Avro
  d. Protobuf
  e. JSON Schema.
- With Avro, Protobuf, or JSON Schema, schemas are registered and validated against Schema Registry.

Security and Integration:

Security and Integration:

- **Robust Security:** Configurable to support Kafka's SSL/TLS encryption and SASL-based authentication mechanisms.
- **Seamless Integration:** Works well with other components of the Confluent Platform like Schema Registry and Kafka Connect.

Wrap-up

What we have learned so far?

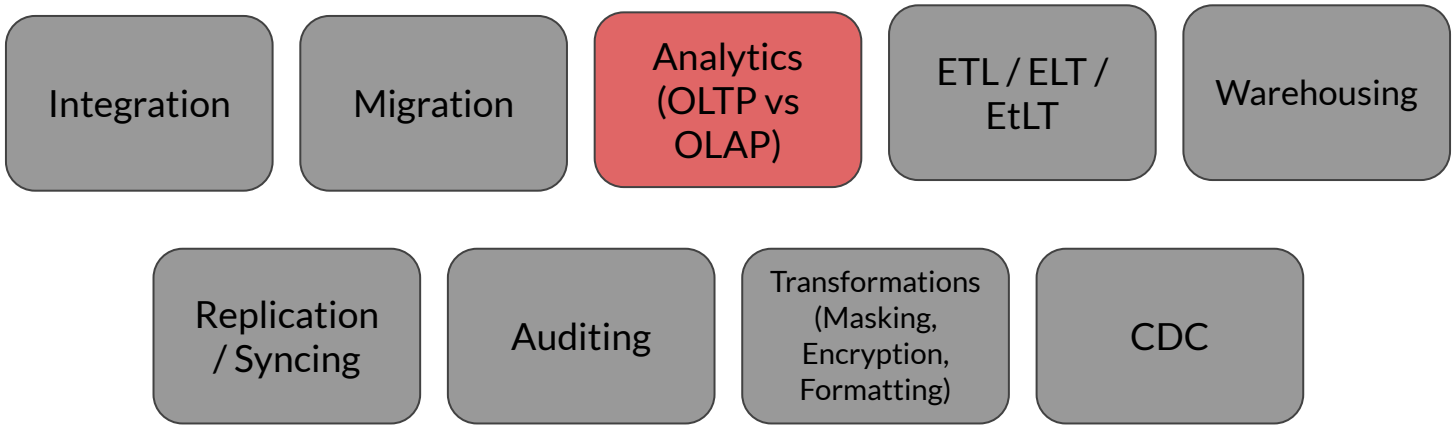
1.

Q&A

Lab

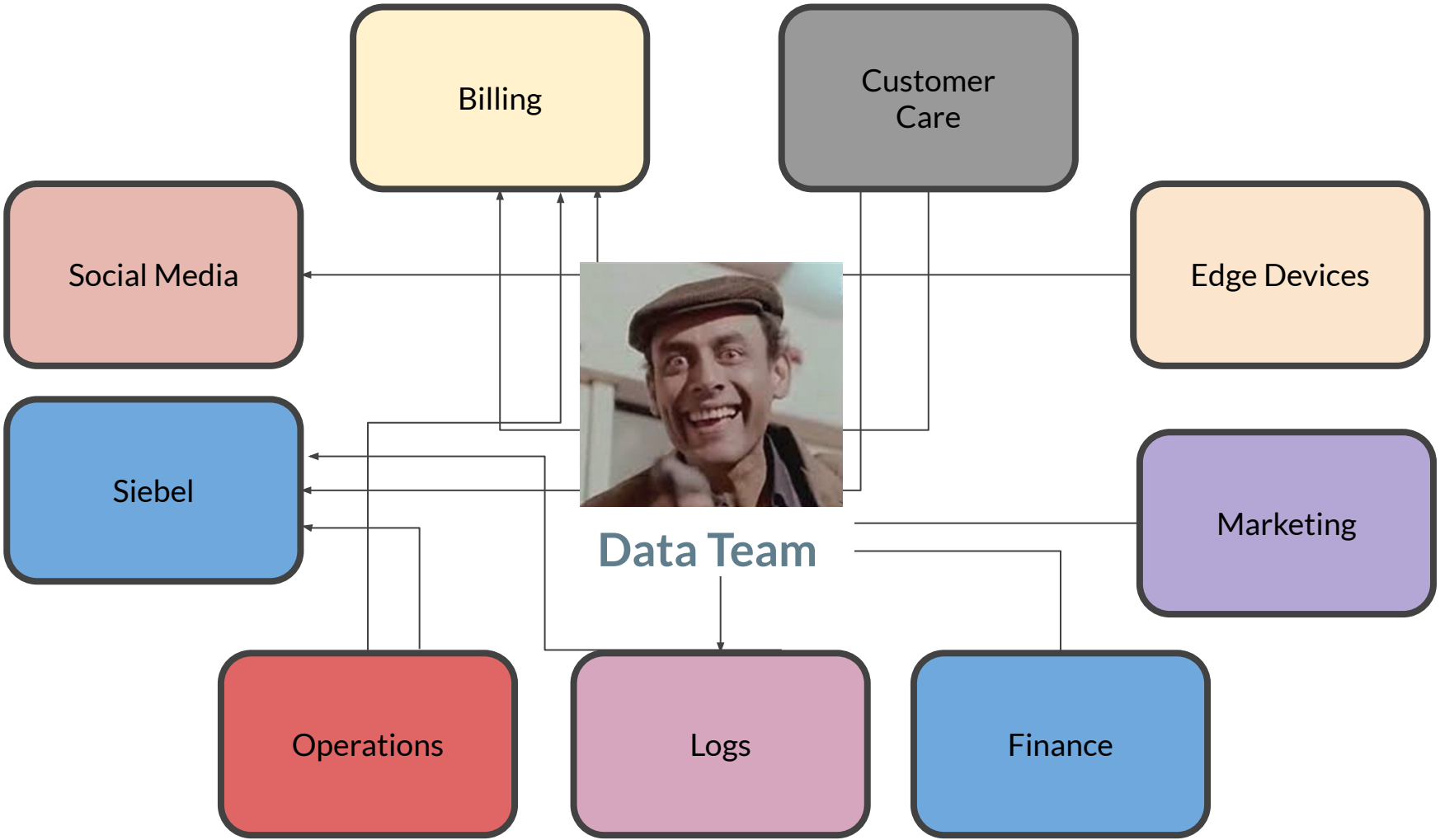
Kafka Workshop [Day-4]

Data Known Challenges

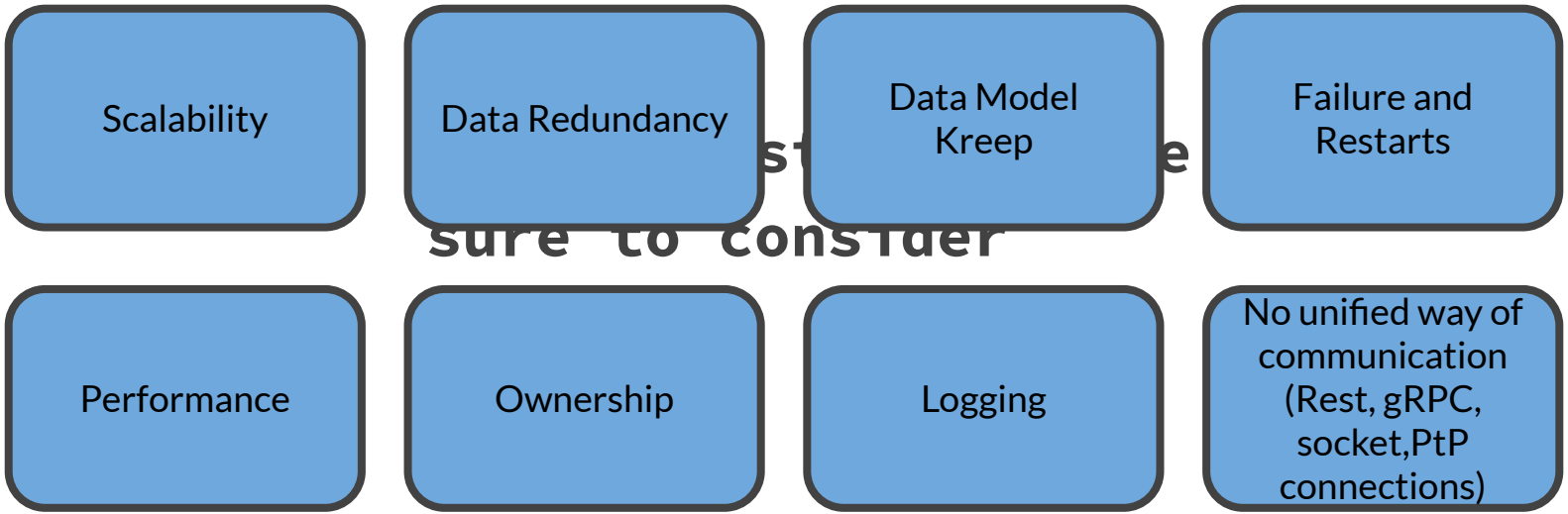



سيبولي الطلعه دي

Multiple Data source Challenge

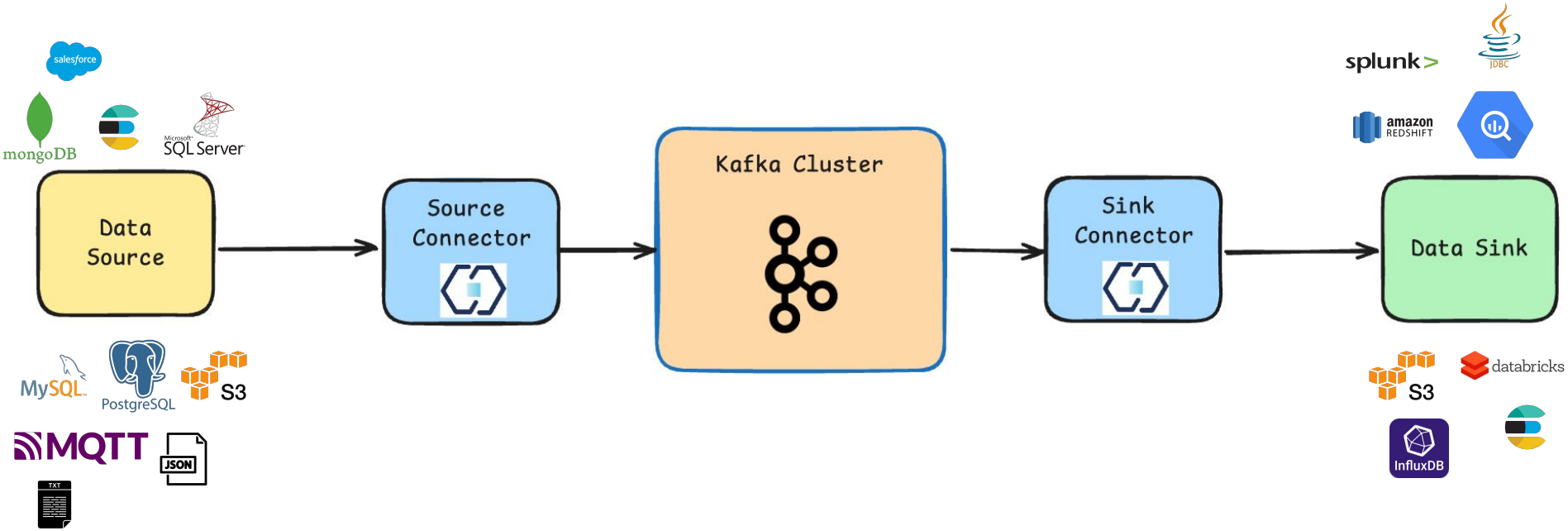

Billing
Customer Care
Social Media
Edge Devices
Siebel
Marketing
Data Team
Operations
Logs
Finance

I can write a script for it

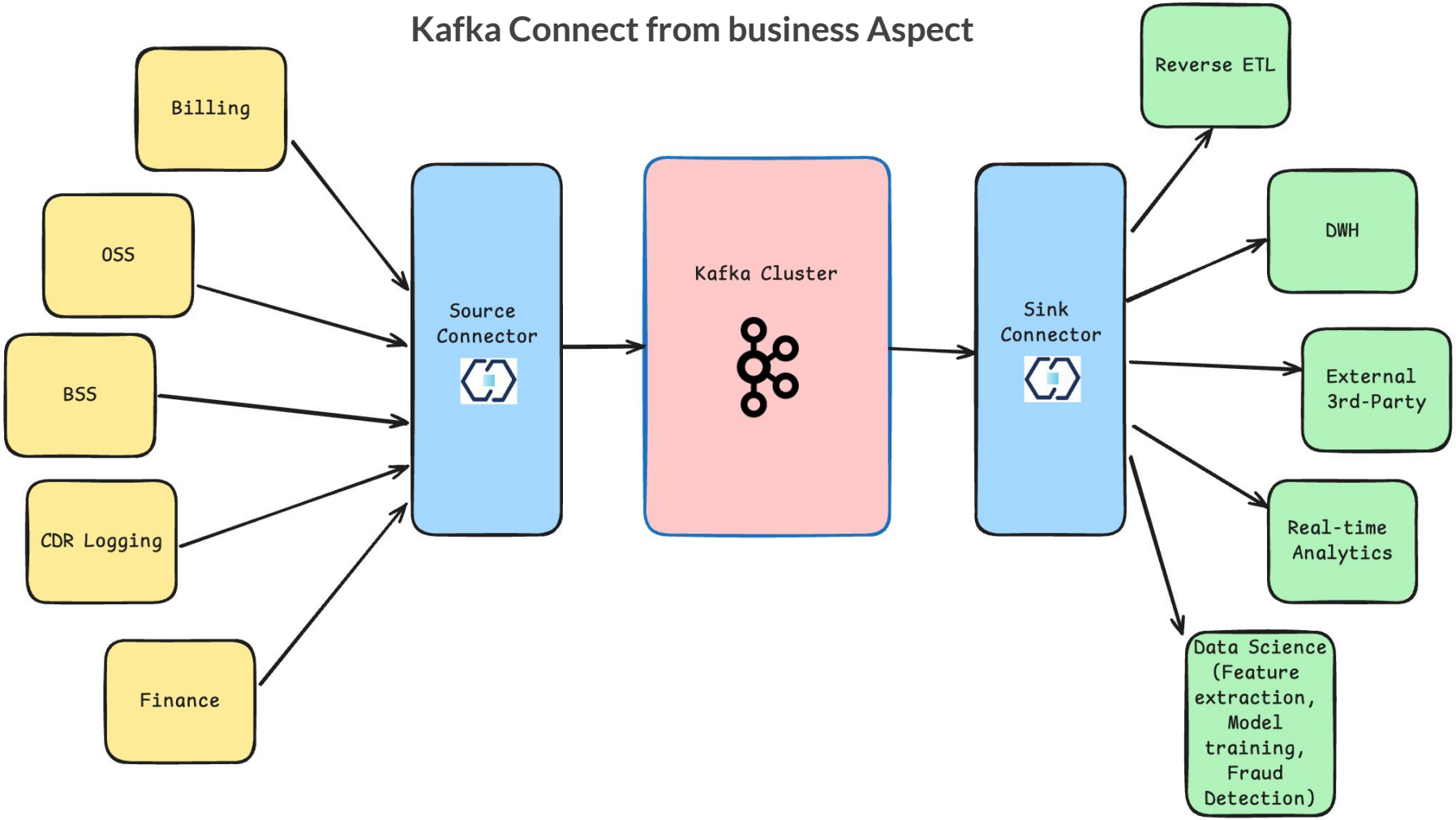



Senior SWE
Data Architect
احنا بنفكر نستخدم Tool
في ال Data XYZ
Migration project
ليه؟؟؟، هو أنا قصرت
معاكم في حاجة؟؟؟

Kafka Connect - High level Architecture



Kafka Connect from business Aspect



Kafka Connect - Definition

- **Kafka Connect** is a component of Apache Kafka that’s used to perform streaming integration between Kafka and other systems such as databases, cloud services, search indexes, file systems, and key-value stores.
- **Kafka Connect** makes it easy to stream data from numerous sources into Kafka, and stream data out of Kafka to numerous targets. There are hundreds of different connectors available for Kafka Connect.
- **Kafka Connects** comes with the pack of all features of Kafka Ecosystem:
  - Scalable
  - Distributed
  - Fault tolerant

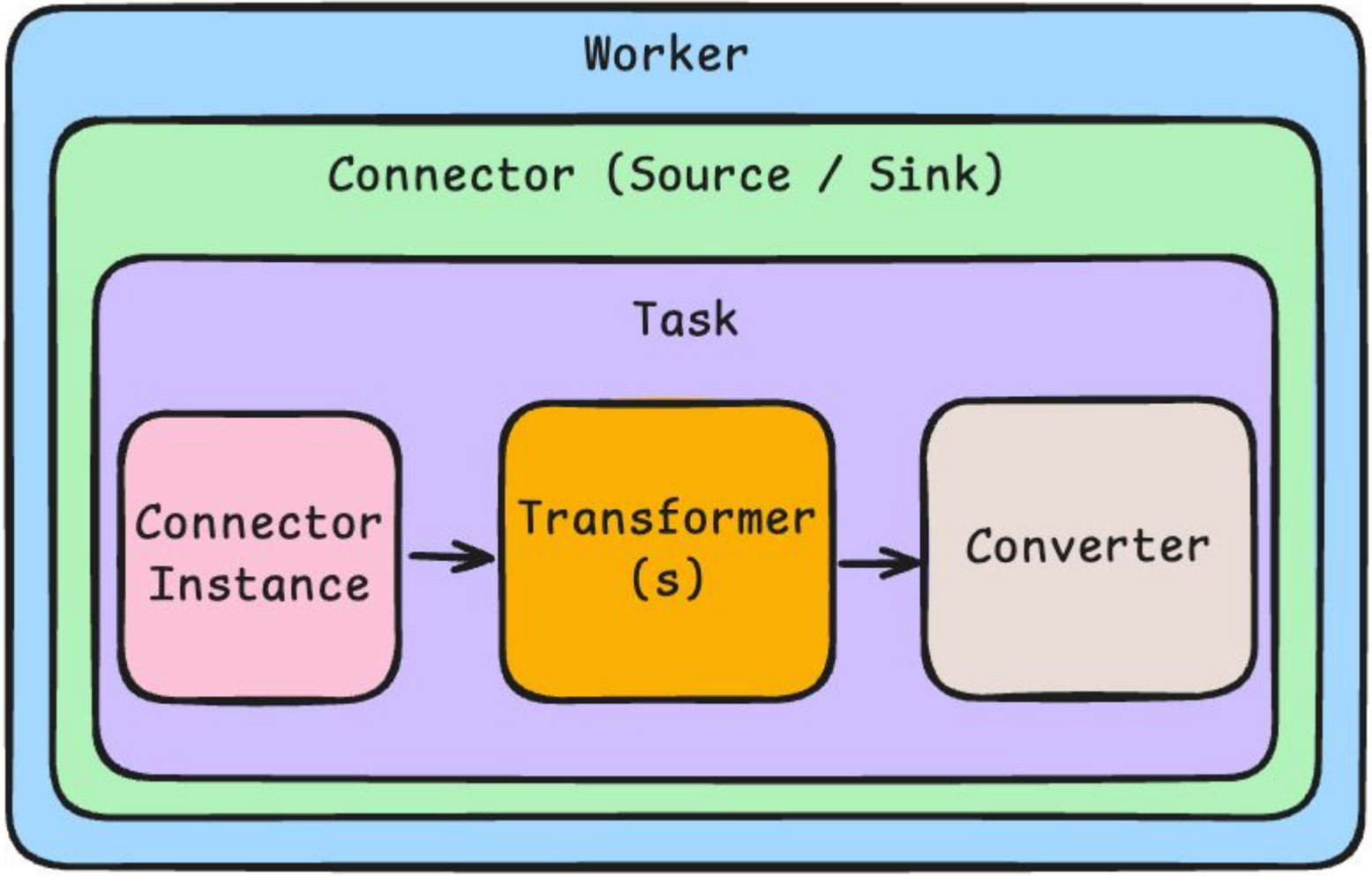
Kafka Connect - Purpose and Motive

Purpose: It simplifies the integration of data sources and sinks with Kafka, enabling continuous data streaming between systems.

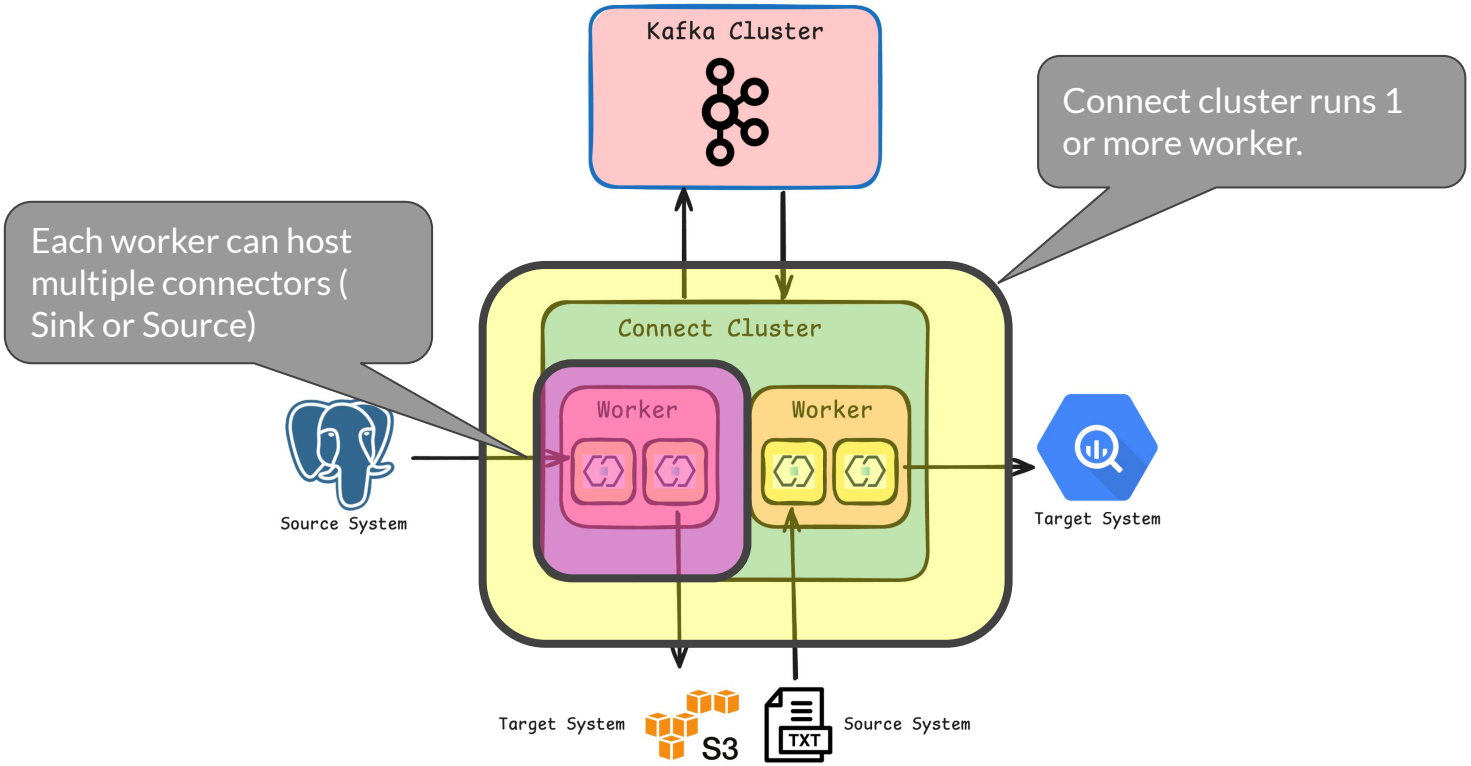
Motive: Designed to handle large-scale, real-time data movement efficiently with minimal effort.

Building Blocks of Kafka Connect

- **Workers:** JVM processes running connectors and tasks. They can operate in standalone or distributed mode.
- **Connectors:** Reusable components that copy data between Kafka and other systems.
  - **Source Connectors:** Pull data from external systems into Kafka.
  - **Sink Connectors:** Push data from Kafka to external systems.
- **Tasks:** Individual units of work that a connector performs. Each connector can be broken down into multiple tasks to parallelize data transfer.
- **Transforms:** Simple data transformations applied to messages as they flow through the connectors.
- **Converters:** handle the serialization and deserialization of data.

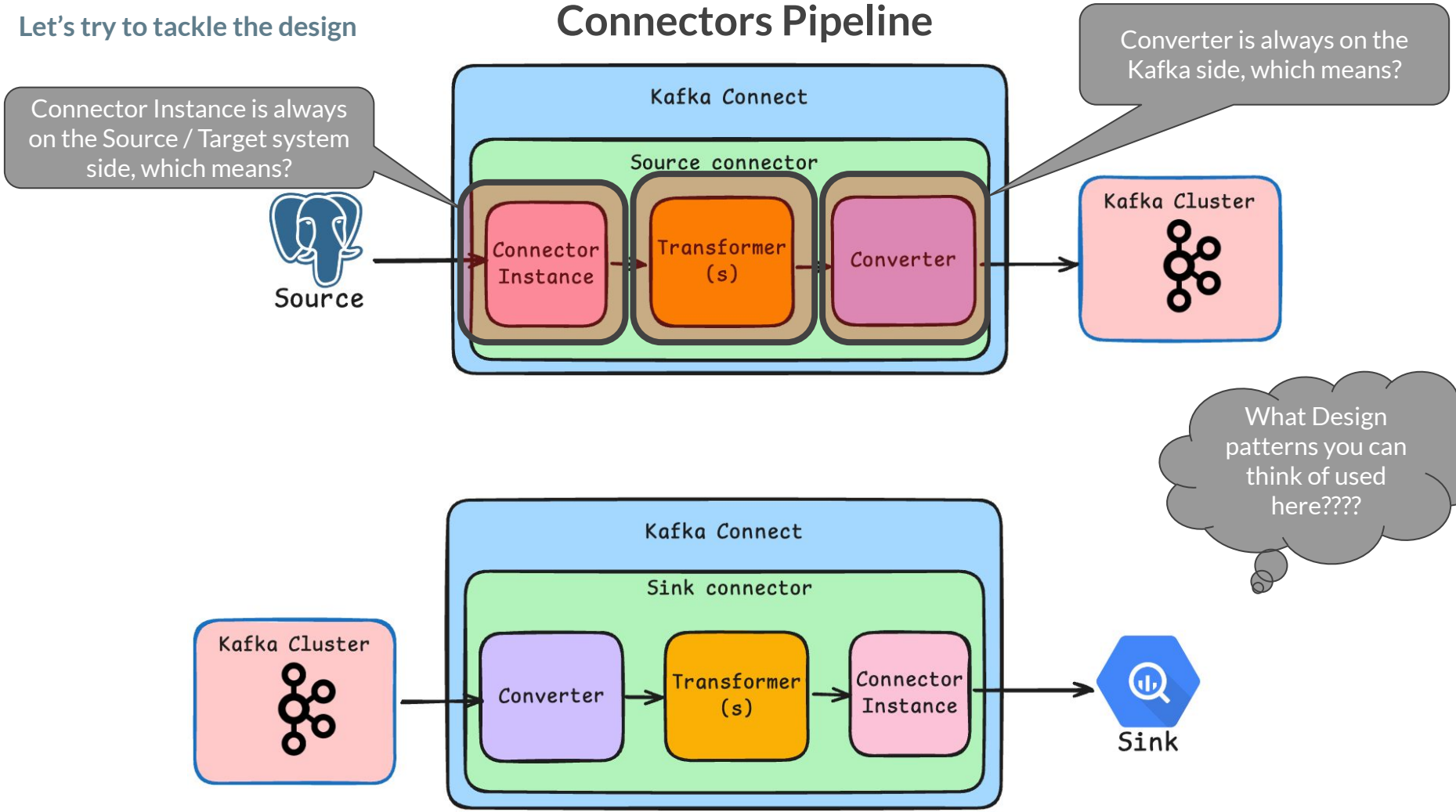

Worker
Connector (Source / Sink)
Task
Connector Instance
Transformer (s)
Converter

Detailed Architecture



Let's try to tackle the design

Connectors Pipeline



https://developer.confluent.io/courses/kafka-connect/intro/

What is Connectors Instance

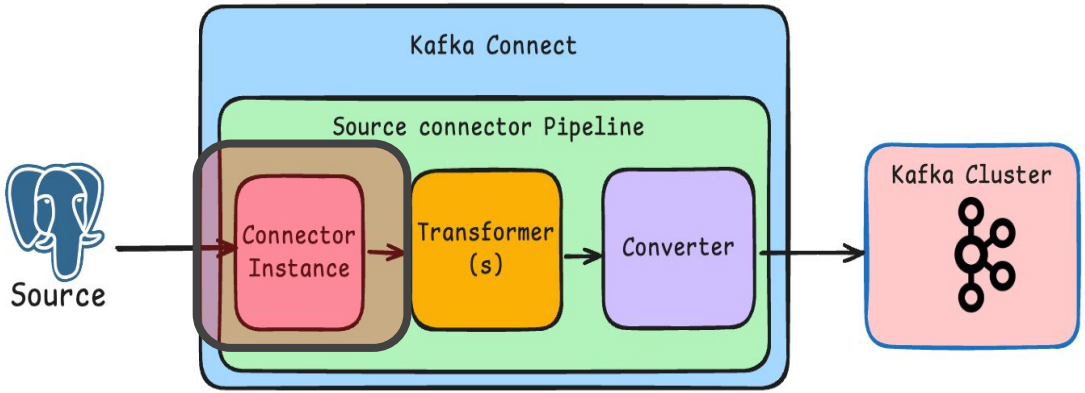
Key Component in any Kafka pipeline (Connect Instance)

All class implemented by Connector Instances are defined in it (Connector Plugin)

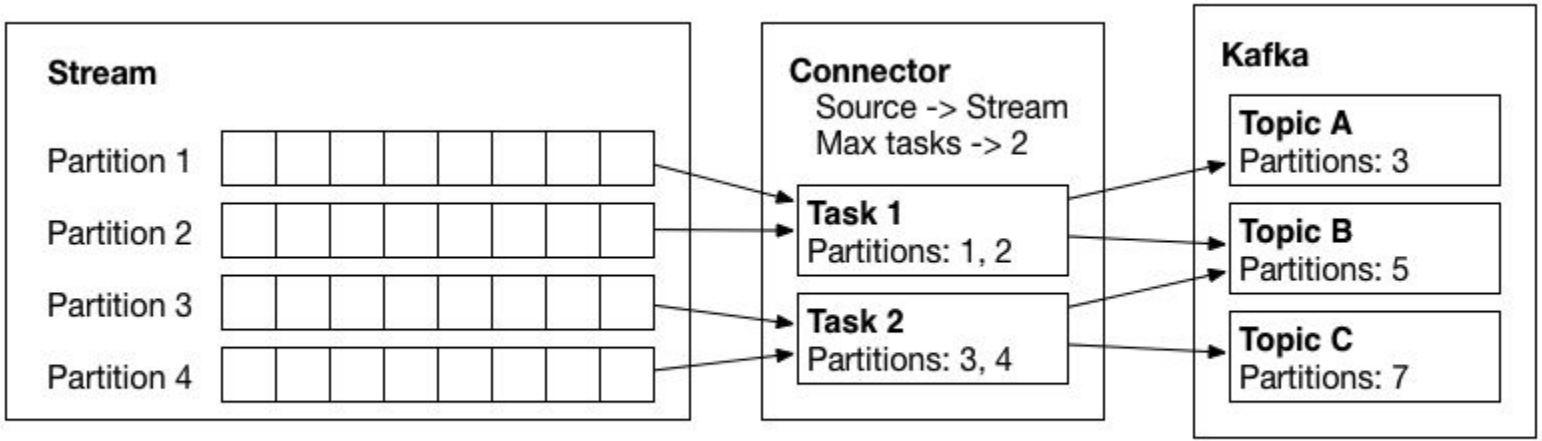
Connector plugins are written by community, vendors, or users.

Example:
1. Debezium
2. ElasticSearch Sink
3. S3 Source/Sink

Connector plugins are reusable components that you can download and install without writing code



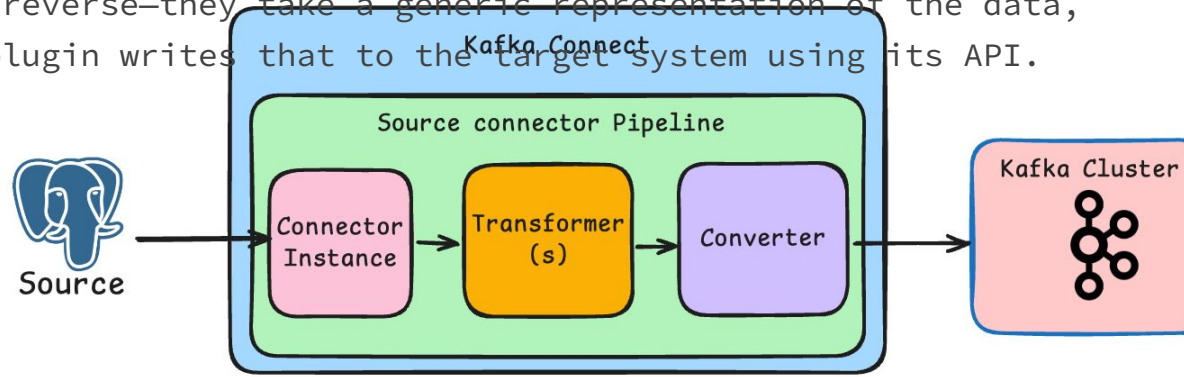
How Tasks can handle topic partitions and distribute the load among themselves.



Example of a source connector which has created two tasks, which copy data from input partitions and write records to Kafka.

What is Connectors Instance

- **Connector plugins** themselves don't read from or write to (consume/produce) Kafka itself. The plugins just provide the interface between Kafka and the external technology. This is a deliberate design.
- **Source connectors** interface with the source API and extract the payload + schema of the data, and pass this internally as a generic representation of the data.
- **Sink connectors** work in reverse—they take a generic representation of the data, and the sink connector plugin writes that to the target system using its API.



What is Connectors Plugin

- **A SOURCE connector plugin:** knows how to talk to a specific SOURCE data system and generate records that Kafka Connect then writes into Kafka.
- **A SINK connector plugin:** knows how to send those records to a specific SINK data system.
- So the connectors know how to work with the records and talk to the external data system, but Kafka Connect workers act as the conductor and take care of the rest.

What is the role of Transformers?

Single Message Transforms (SMTs)

What is the role of Converter?

– – –

-
-

Error Handling in Kafka Connect

Kafka Connect Tips

---

1. Take care of **max.tasks** config, it defines the number of tasks for a given connector, it is set to 1 by default, which may result into CPU under utilization, however not all connectors can work with more than 1 task.
2. Deploy kafka connect on a separate nodes from the kafka related machines.

Kafka Workshop [Day-5]

General Data Engineering concepts

---

1. Challenges.
2. Data Architecture concepts.
3. Tips.

Thinking Like a Data Engineer

Identify business goals & stakeholder needs

1. Identify business goals & stakeholders you will serve
2. Explore existing systems and stakeholder needs
3. Ask stakeholders what actions they will take with the data product

Define system requirements

1. Translate stakeholder needs to functional requirements
2. Define non-functional requirements
3. Document and confirm requirements with stakeholders

Choose tools & technologies

1. Identify tools & tech to meet non-functional requirements
2. Perform cost / benefit analysis and choose between comparable tools & tech
3. Prototype and test your system, align with stakeholder needs

Build, evaluate, iterate & evolve

1. Build & deploy your production data system
2. Monitor, evaluate, and iterate on your system to improve it
3. Evolve your system based on stakeholder needs