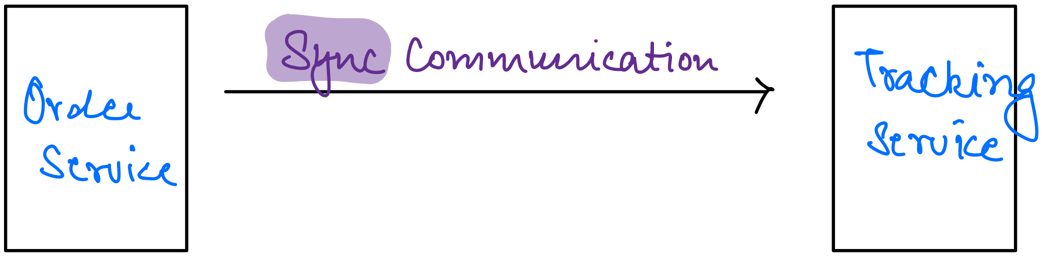
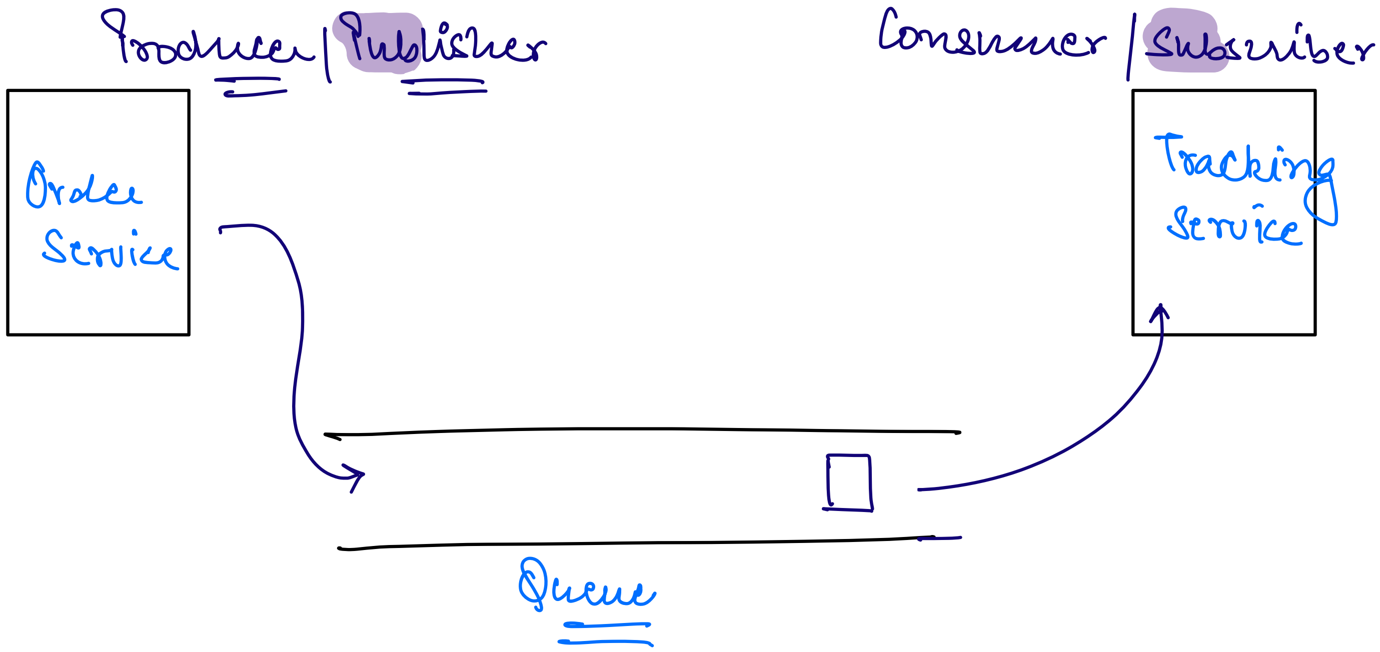
Messaging Queues.



→ Asynchronous.

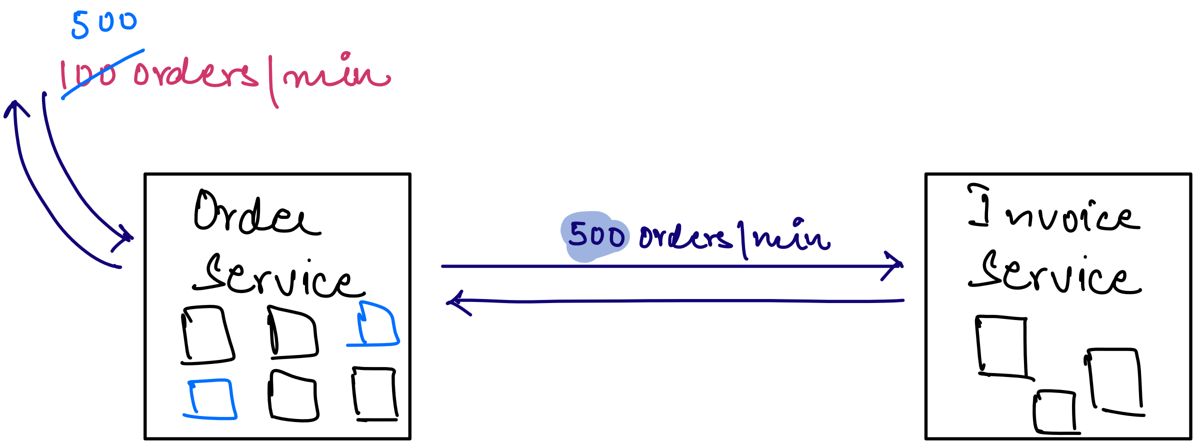
→ Messages | Events are placed in the Queue by the Producers without waiting for the Subscribers to process them.

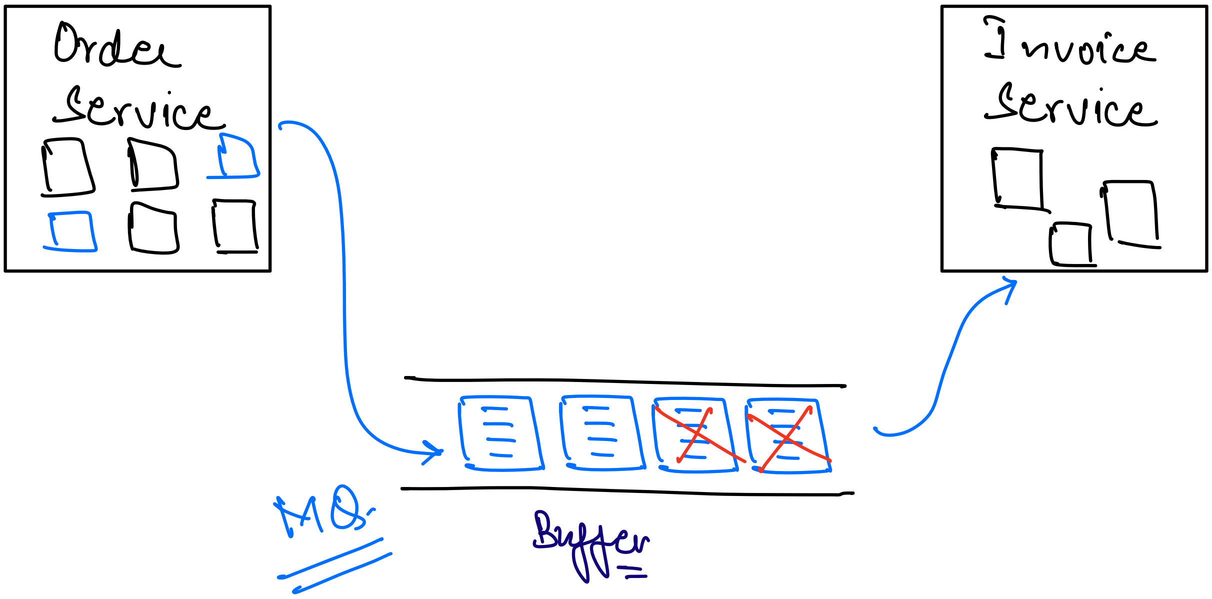


Advantages of MQ.

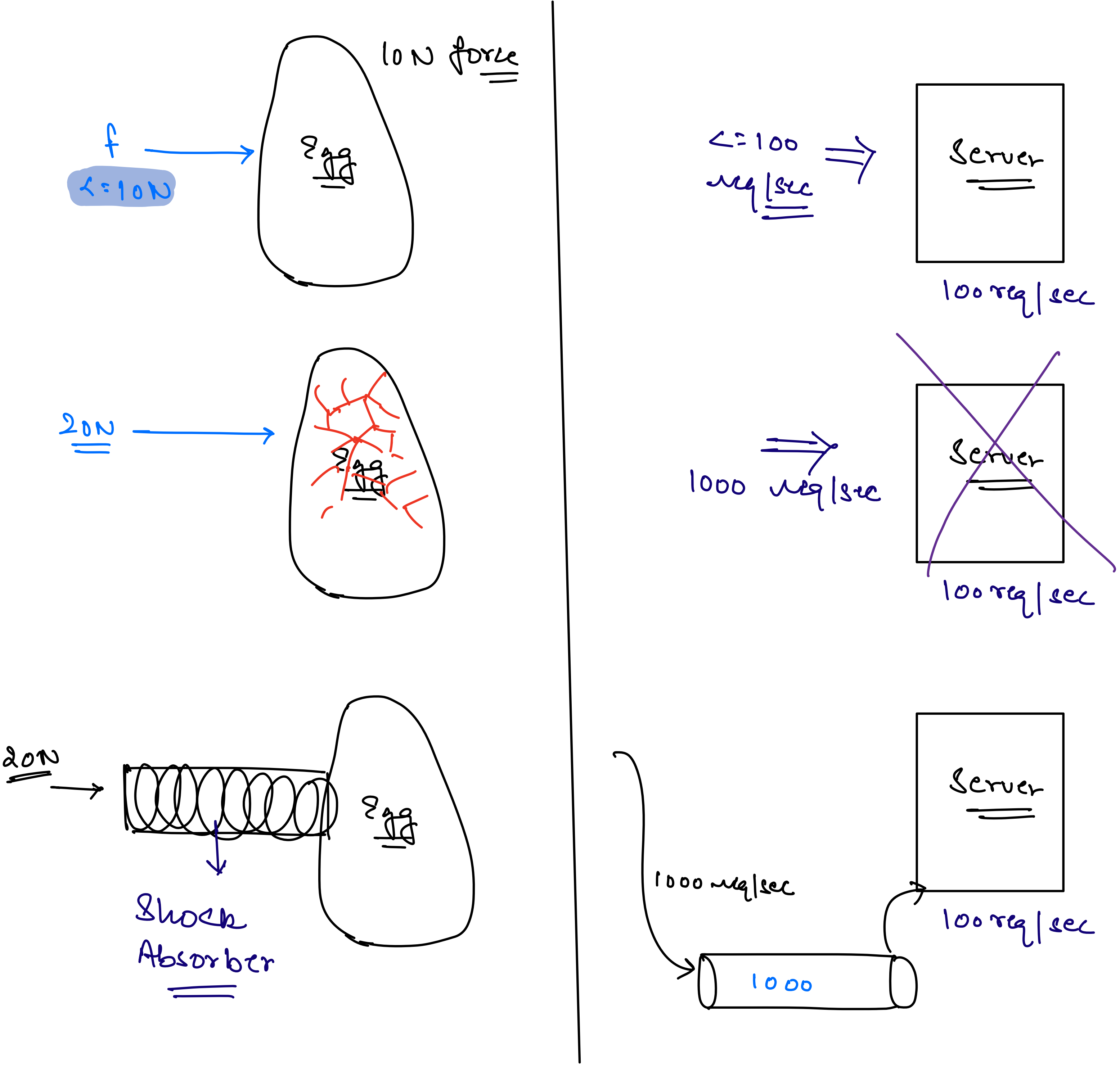
1) Decoupling.

→ Scalability : Producers & Consumers can be scaled independently.





→ Fault Tolerance



⇒ Buffering & Persistence

Use Cases.

1) Communication b/w Microservices.

2) Async processing | Task scheduling.

Cons of MQ.

→ Can't perform sync processing.

→ There can be a high latency.

→ Needs additional infrastructure.

→ High N/w Overhead.

Examples of MQ

→ Kafka.

→ Rabbit MQ

→ Amazon SQS

↳ Managed Infra.

1. Open Source: built at Linkedin, now part of Apache
2. High Throughput: a cluster can handle up to 10 million msg/s (trillion/day)
3. Ultra Low Latency: claims < 2ms latency from message insertion to consumption
4. Horizontal Scalability: automatically manages partitions across multiple servers (brokers)
1000s of brokers (servers), 100,000s of partitions, petabytes of data
5. Fault Tolerant: automatically replicates messages across multiple servers
6. High Availability: can deploy clusters across availability zones
7. Persistent: saves the messages on disk to ensure they're not lost. Multiple consumer groups can consume the same message.
8. Message Ordering: provides strict* message ordering (*within each partition only)

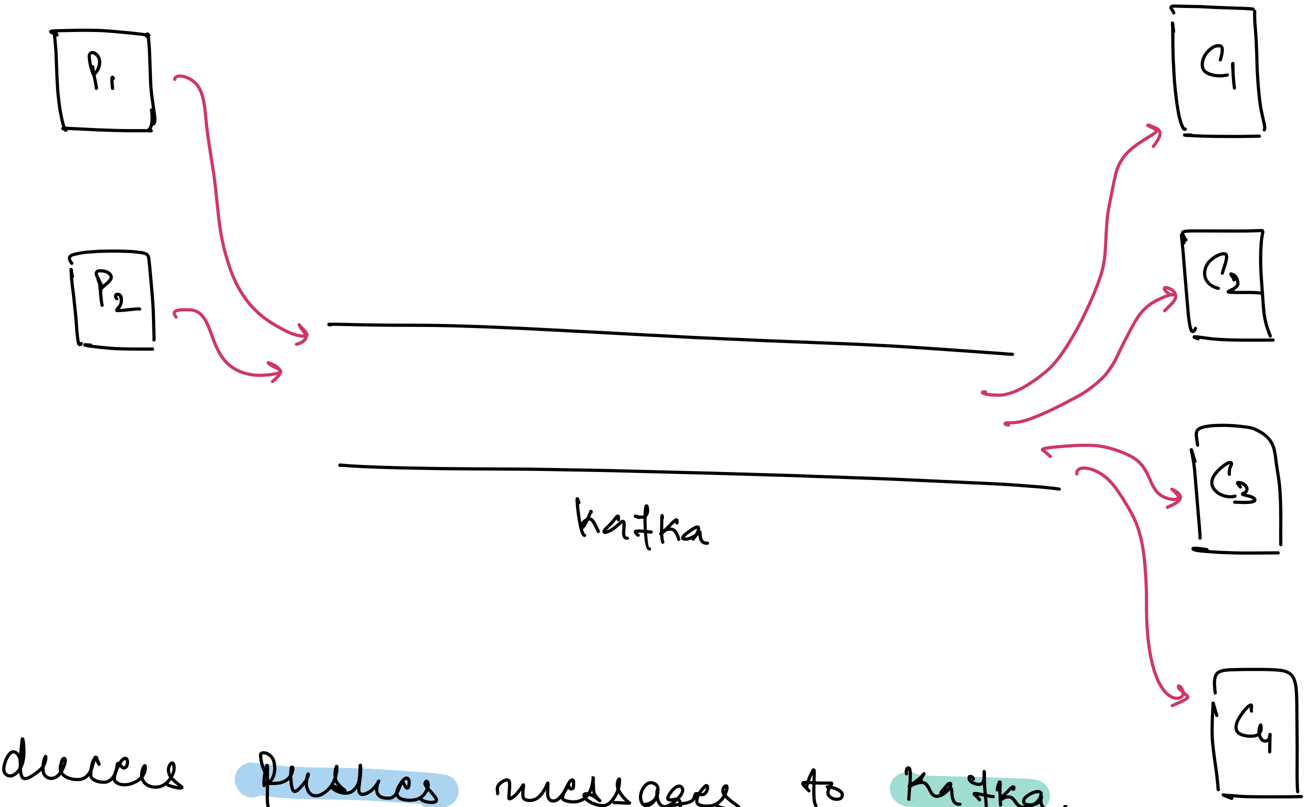
⇒ msg|event = {
"key" : ______
"value" : ______
______
______
______
}

⇒ The size of the msg|event should be limited to few KB's

Producers & Consumers.

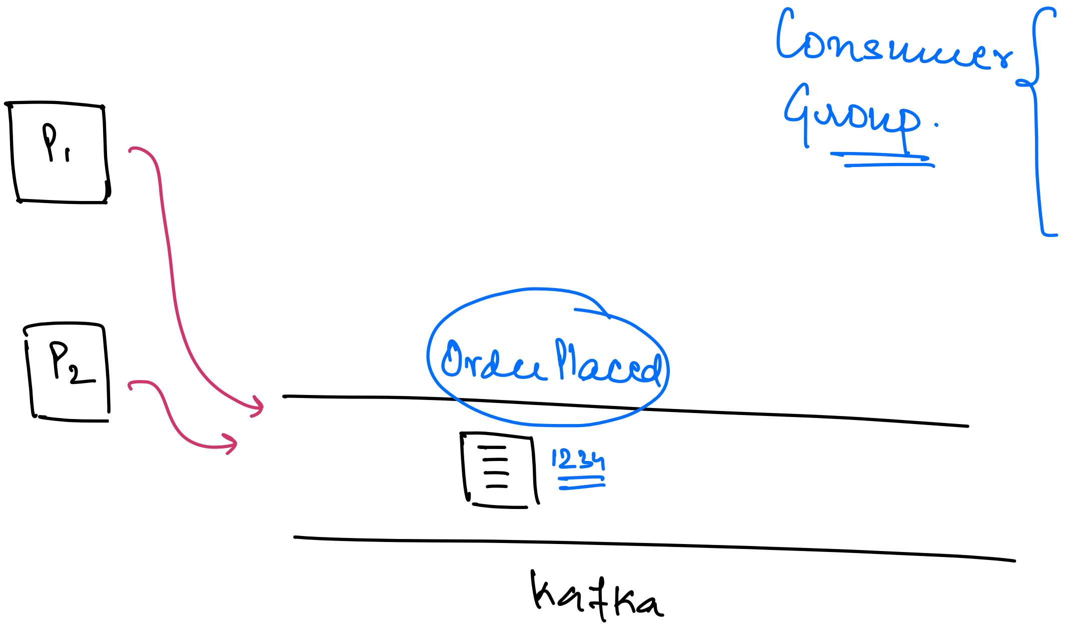
Writes to kafka.

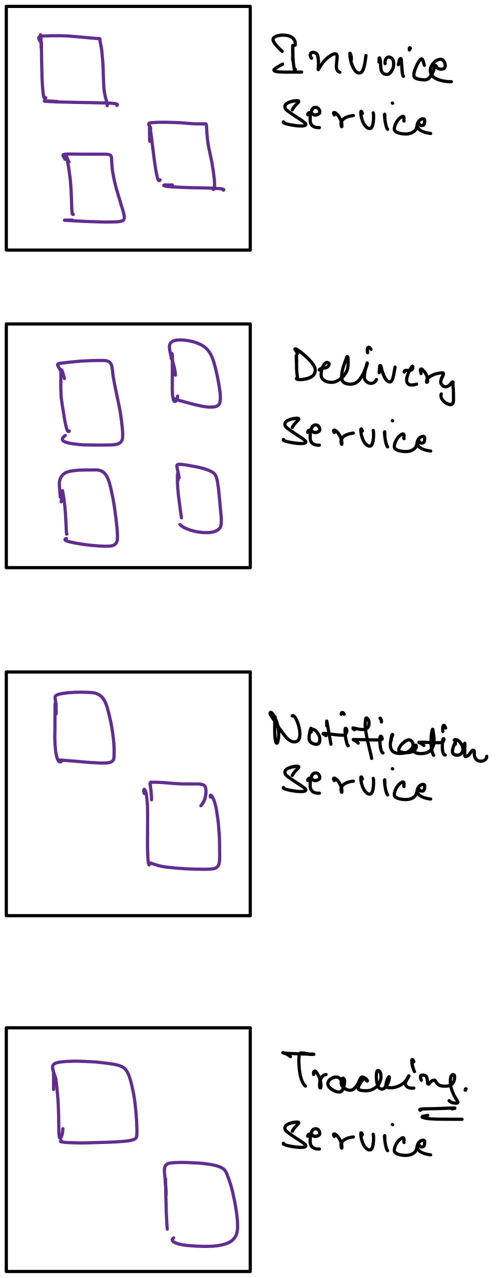
reads from kafka.



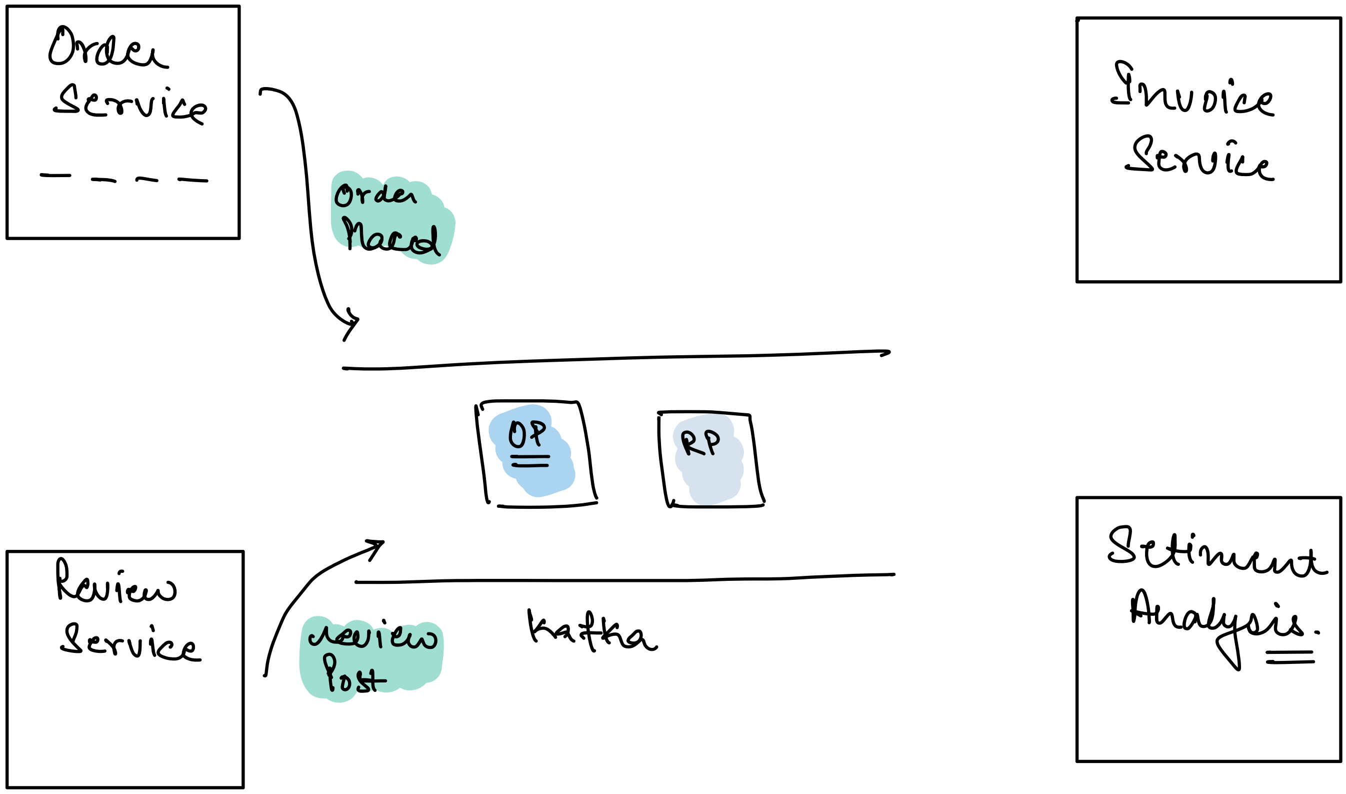
Producers pushes messages to kafka.

Consumers pulls messages from kafka.





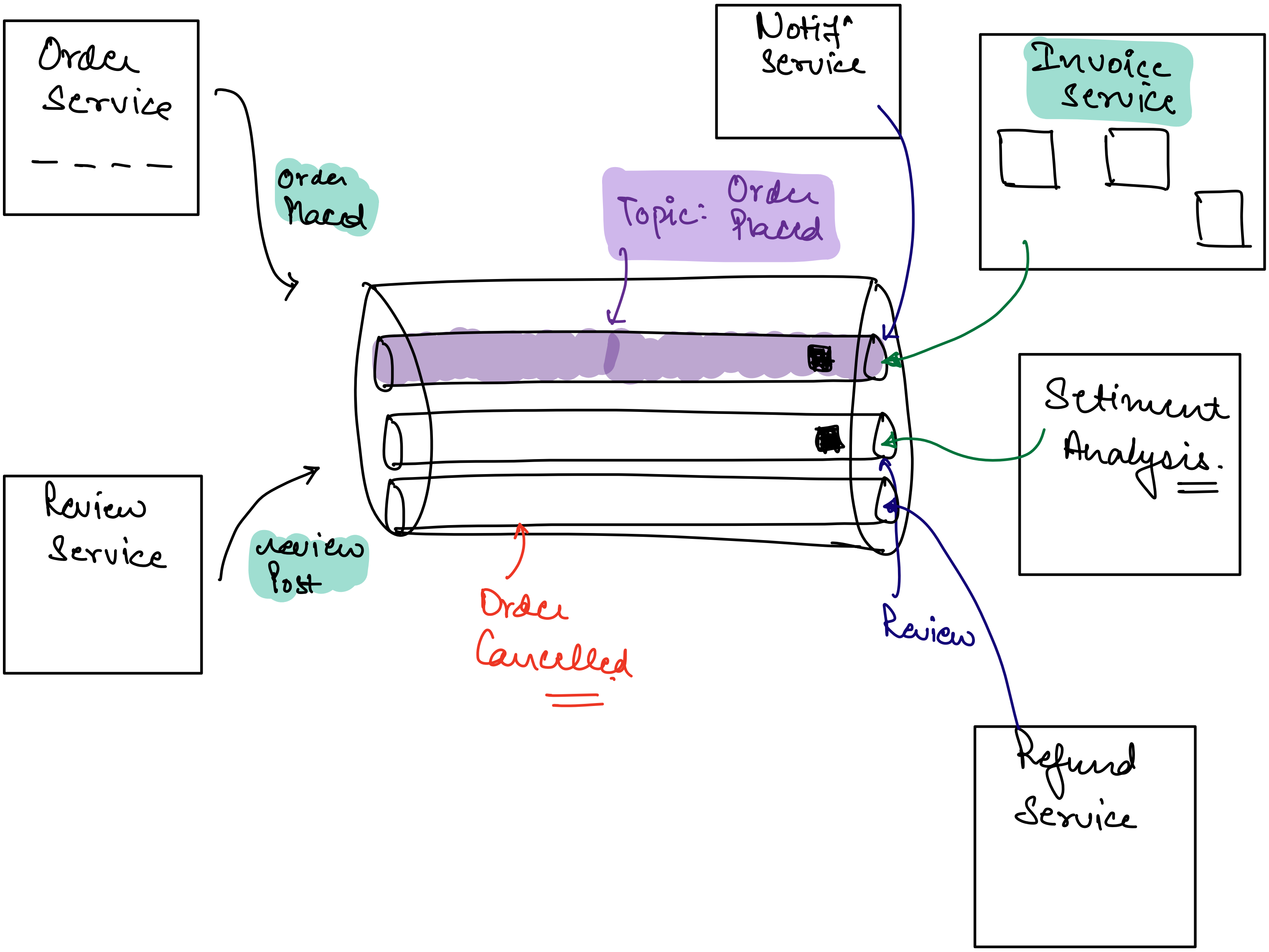
⇒ An event can be consumed by multipe Consumer Groups, but within the group it should be consumed only by one Consumer.



⇒ Different consumer groups can choose to consumes specific category of messages.

⇒ Categorize messages into Topics.

⇒ A msg can belong to exactly one Topic.



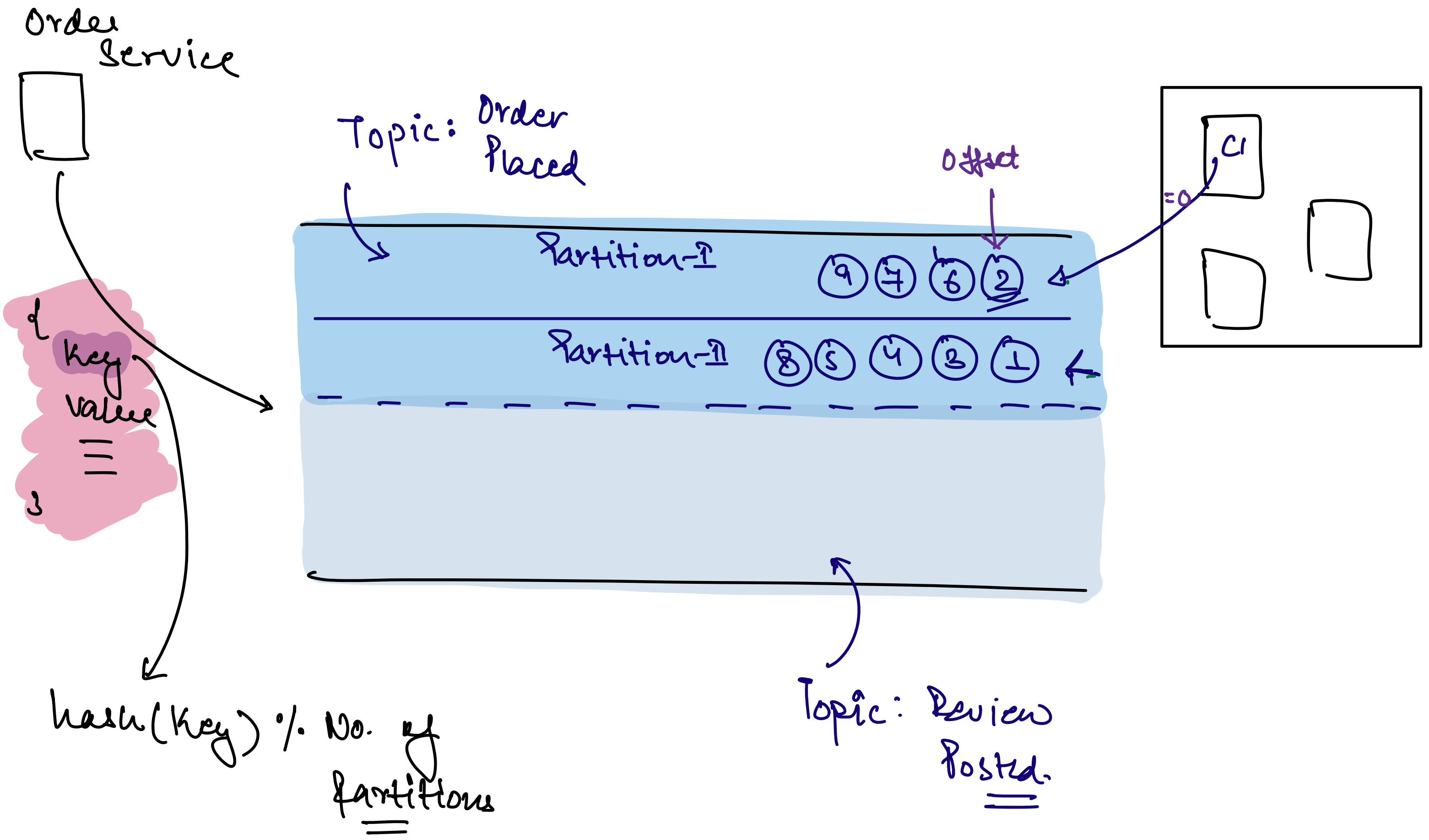
Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events

PARTITIONS

→ A single server is NOT enough to store all the messages in a topic.

Ex: FB Messages.

→ As the topic is too large to store in a single Server, We can partition the topic across multiple servers.



⇒ # of partitions ≡ # of Consumers

↳ No consumer will be sitting idle.

⇒ # of Partitions < # of Consumers

↳ few consumers will be sitting idle.

⇒ # of Partitions > # of Consumers

⇒ We must ensure the # partitions >= # Consumers

Note: 1 partition is assigned to exactly 1 consumer, but 1 Consumer can get multiple partitions.

⇒ If we have more consumers than partitions, some consumers will remain idle.

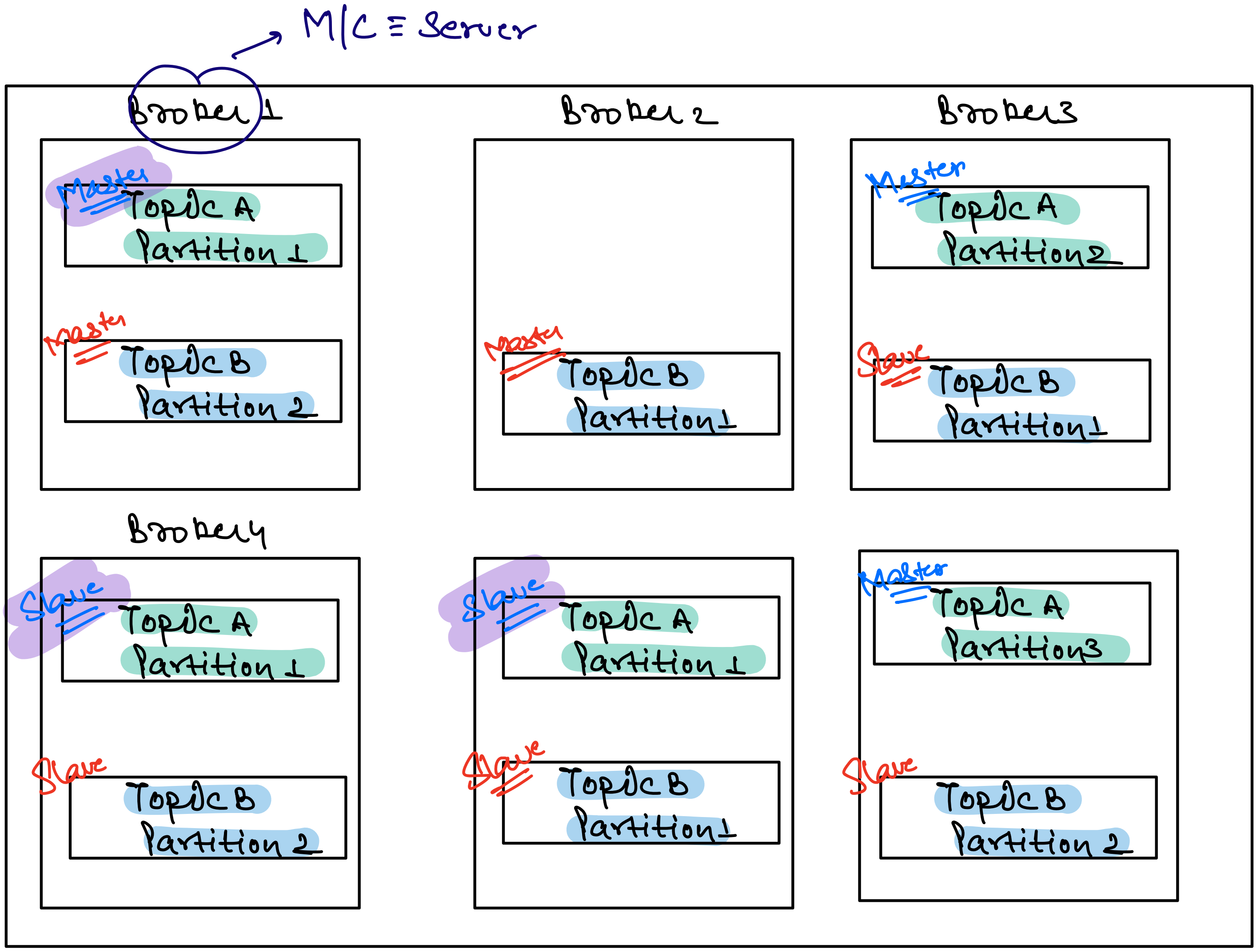
⇒ Since we might need more consumers in future, so we take good enough large # of partitions.

⇒ Kafka ensures the strict ordering with the partition and NOT across partitions.

Scalability

→ SHARDING

→ REPILCATION



ZOOKEEPER.