will start at 9:10 PM

Messaging Queues

⇒ Persistent Queues

⇒ Kafka

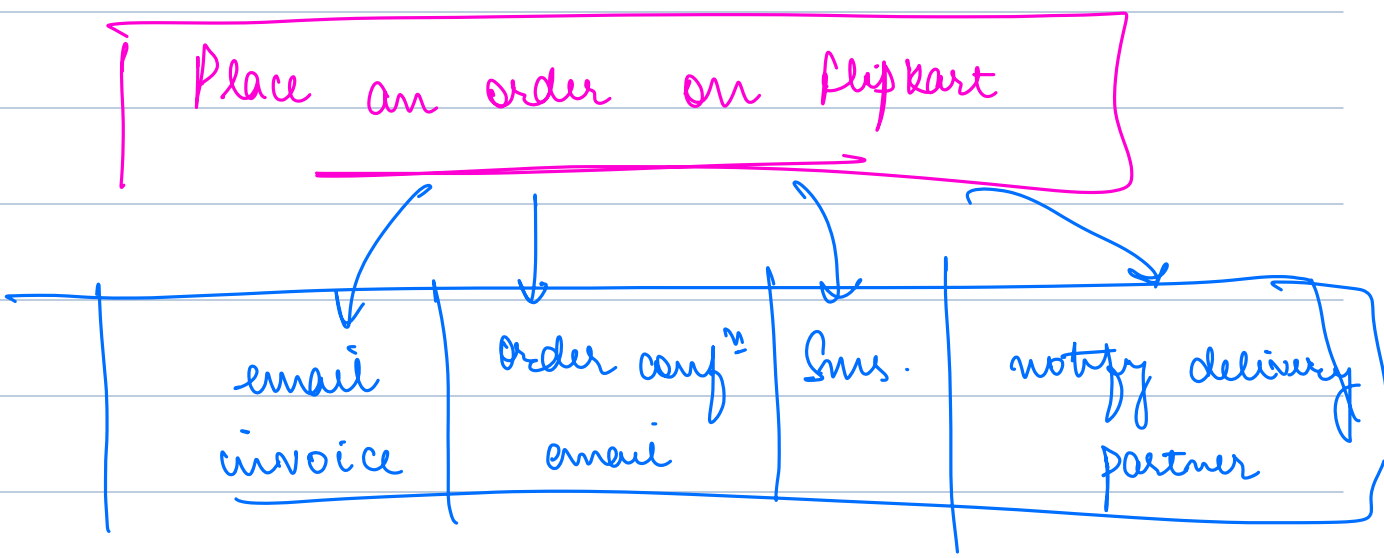
↳ Internals

↳ Terminology

⇒ Prog

ElasticSearch

Why Persistent Queues



class Flipkart {

⇒ void placeOrder() {

emailService.sendEmail(invoice, jerin)

NOT MUST — smsService.sendSMS( )

dbService.createOrder( )

}

}

- Twilio

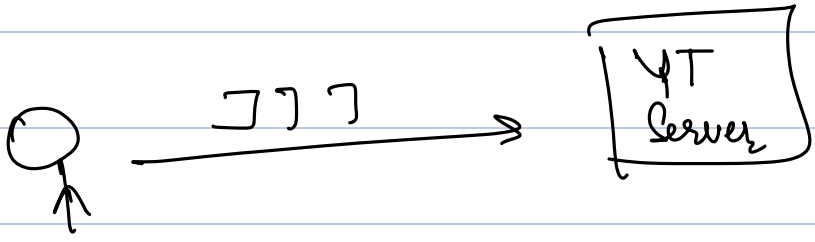
- +91-9996202771

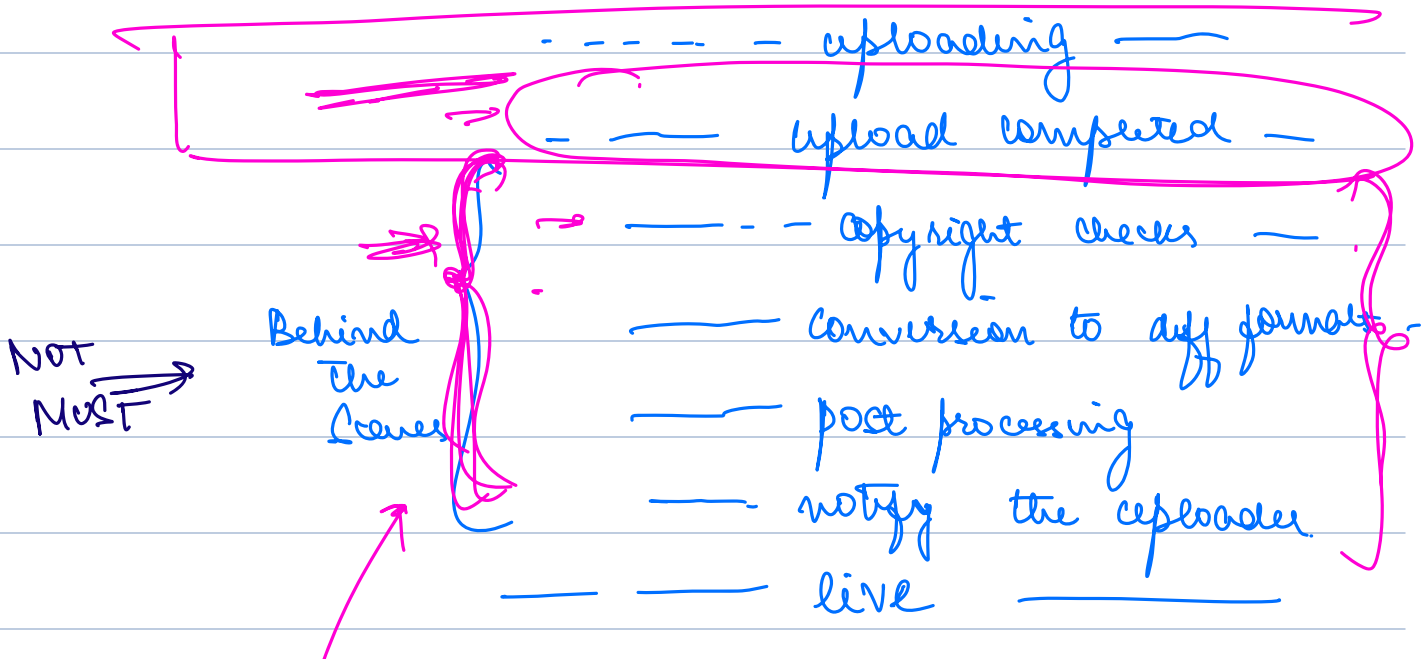
Should an order be considered placed only after Email, SMS, bell are notified?

→ NO

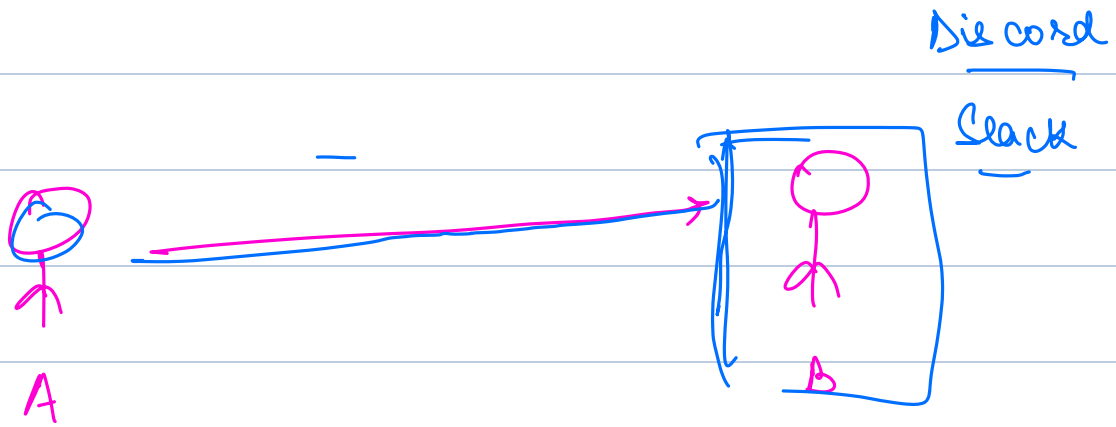
→ email to be sent, SMS to be sent, etc are not mandatory to have completed by considering an order placed.

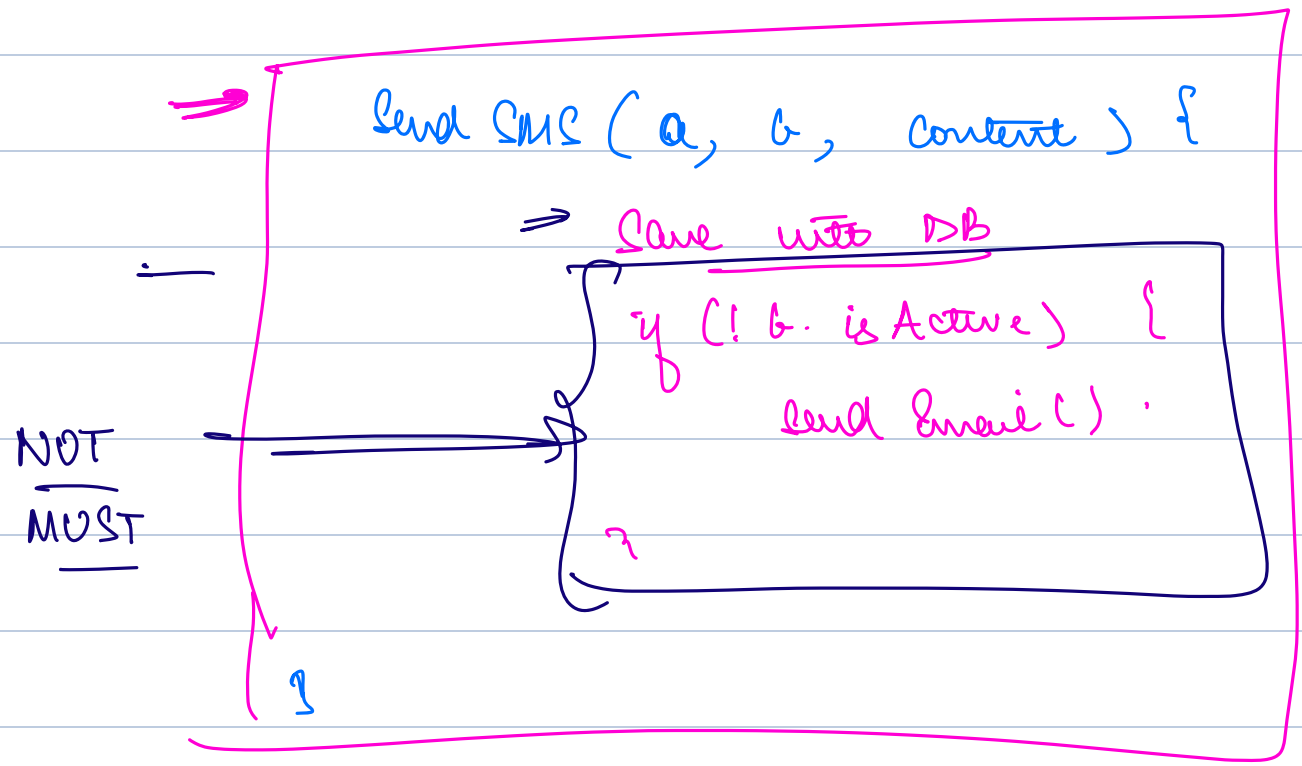
Upload video to YT





all these dont need to be completed before considering a video to be uploaded.





Hotstar view counter

→ watchVideo() {

⇒ db.incrementCount(1)

}

⇒ If some tasks can be done later but → still send a response to client only after all are done, my client may think that API is slow.

→ Doing tasks (async) ⇒ not in sync
↳ in parallel
↳ later.

⇒ Persistent Queue

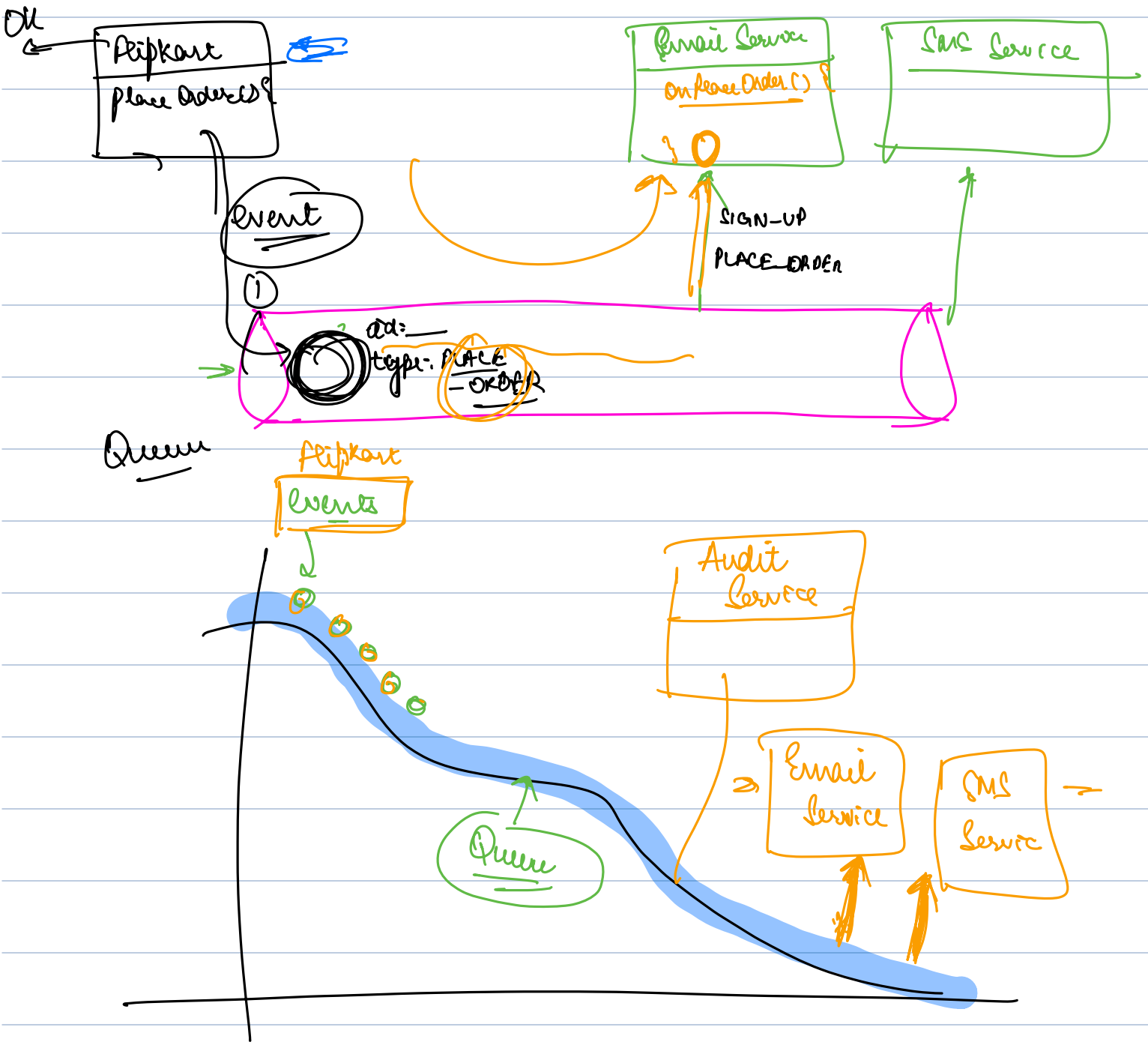
↳ keep the events lifelong (retention period)
= 60 days
configurable.

placeOrder(____) {

- emailService ____
- smsService ____
- orderService ____

}

Instead of calling methods directly puts an event on the queue

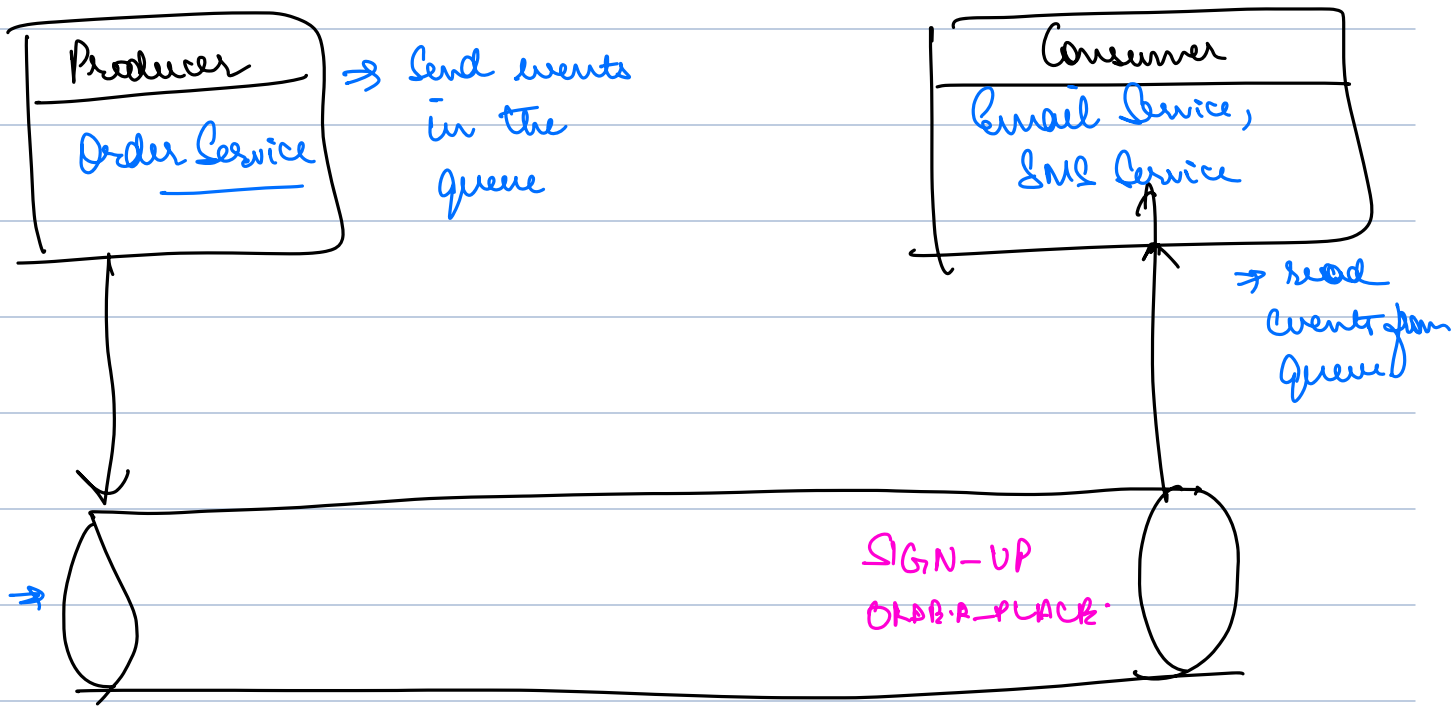


ES, SS, etc :

Waiting for events of particular type on queue.

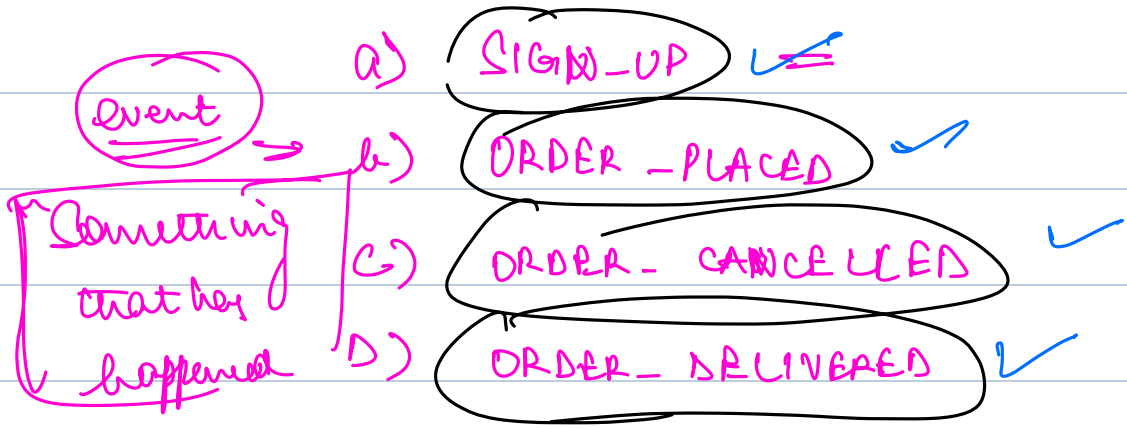
PQ works on the model of Publisher - Subscriber

Producer - Consumer



Topics

⇒ the same queue can be used to transport multiple types of event ⇒ Topic



⇒ Diff consumers can subscribe to diff topics

```
placeOrder() {
    queue.publish Event
        ("ORDER_PLACED", {101, ___, ___});
}
```

KAFKA

How Kafka Works

Terminologies

1. **Producer**: Any service that may want to put an event into the queue (event generator)

2. **Consumer**: Any service that may want to do something on a particular type of event.

3. **Broker**: Machine that is running kafka

4. **Partition**: # of broken events of a particular topic will be recorded into.
   - ↳ decided at topic level
   - ↳ configurable

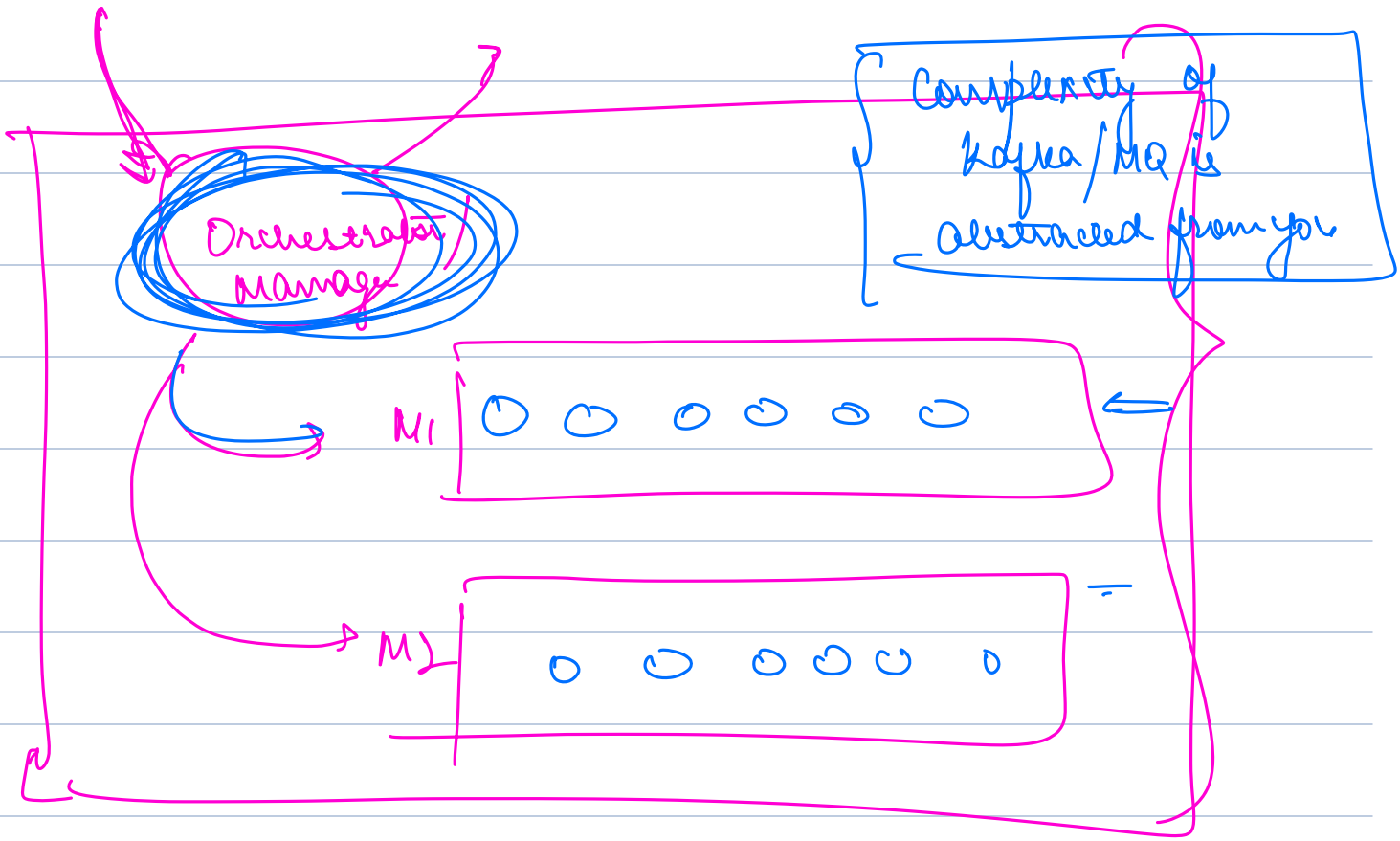
eg

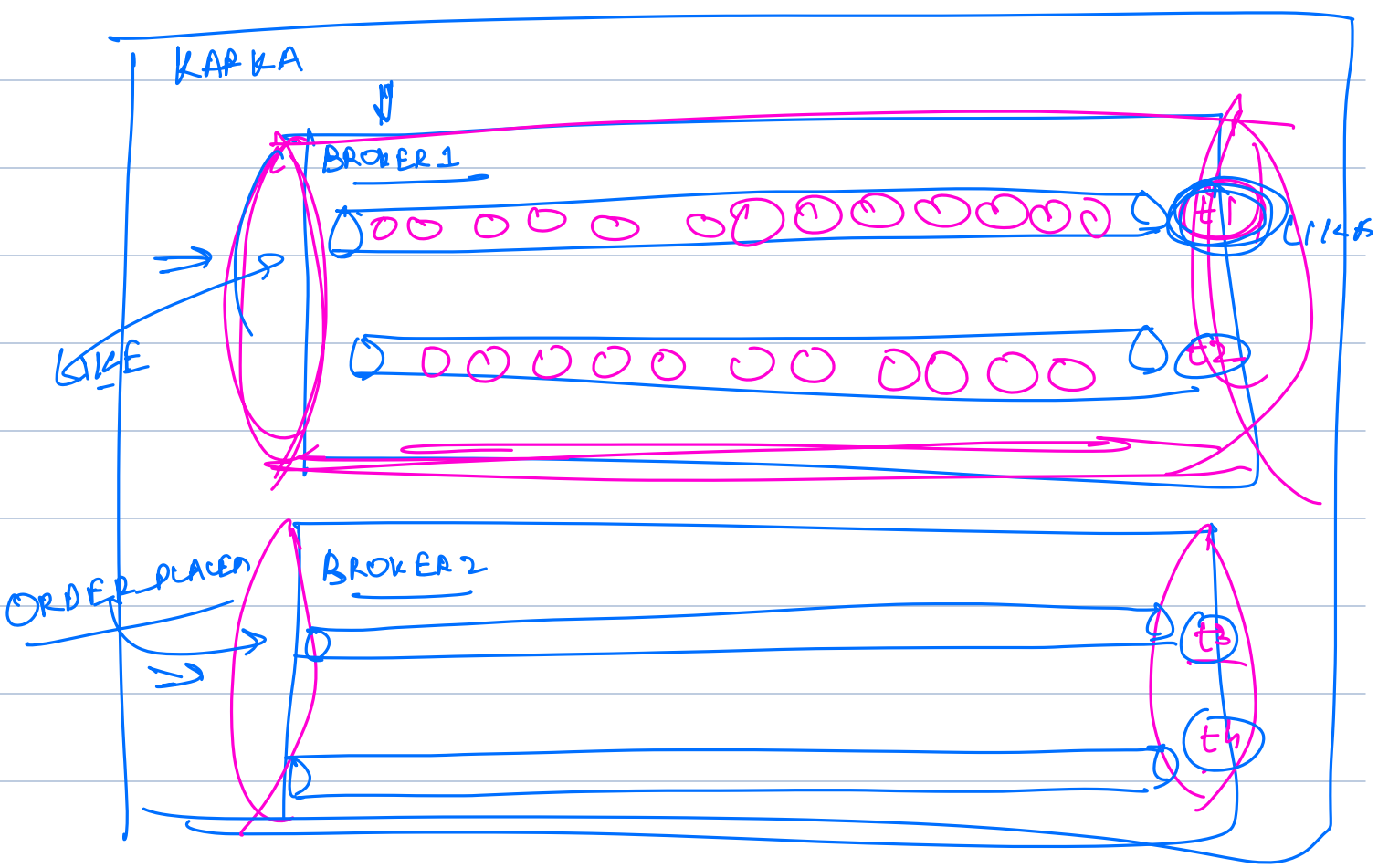
Anyone clicks a like button

- email to the poster
- notfn to poster
- updates recommendation system

⇒ 100M likes / day * 100 ⇒ 10B events

Message Queue (Kafka) may need to dist across multiple machine


Orchestrator / Manager
Complexity of Kafka / MQ is abstracted from you
M1
M2


KAFKA
BROKER 1
BROKER 2
ORDER_PLACED
t1
t2
t3
t4

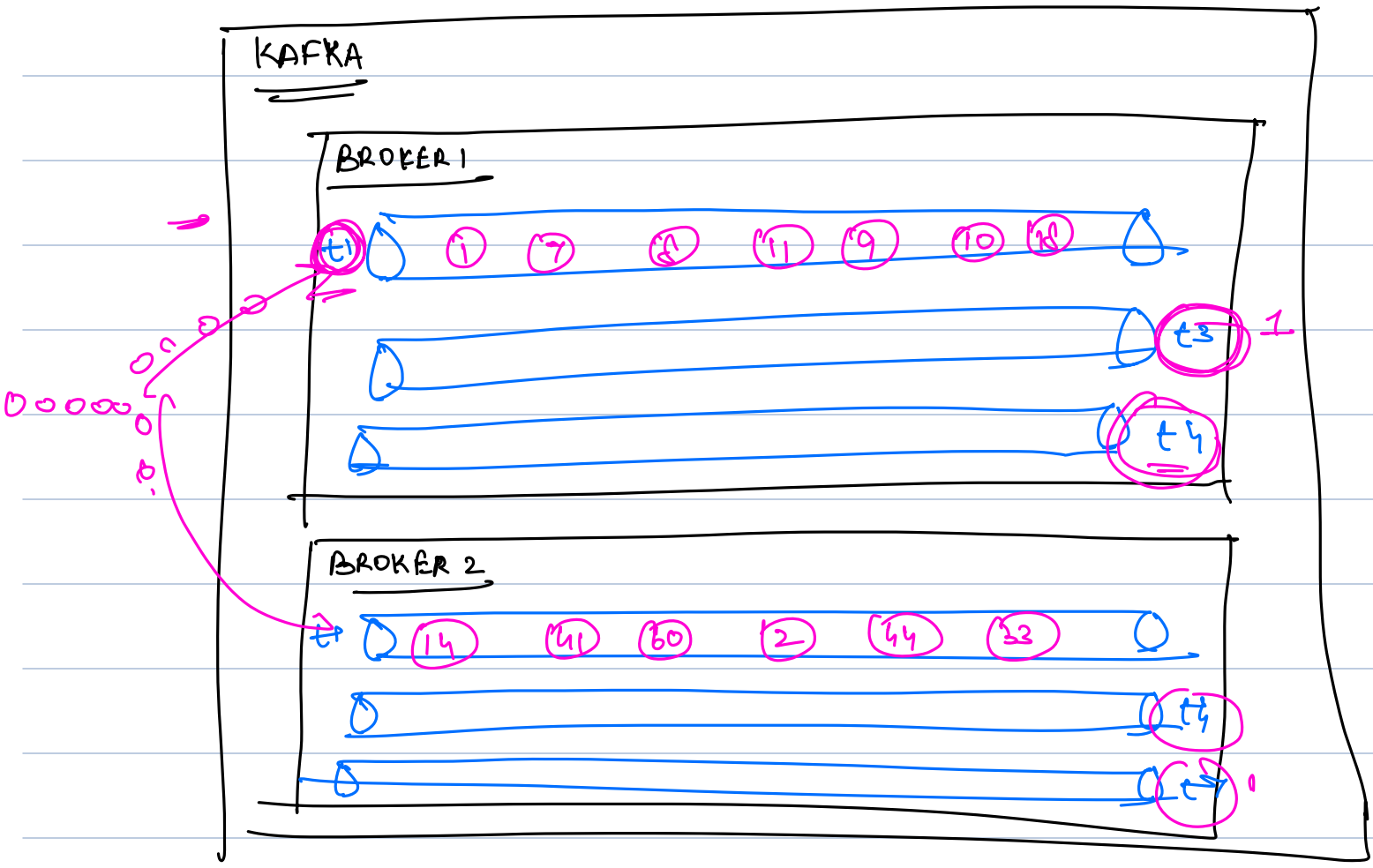
How to split across multi machine

(1) tag each topic to a specific broker

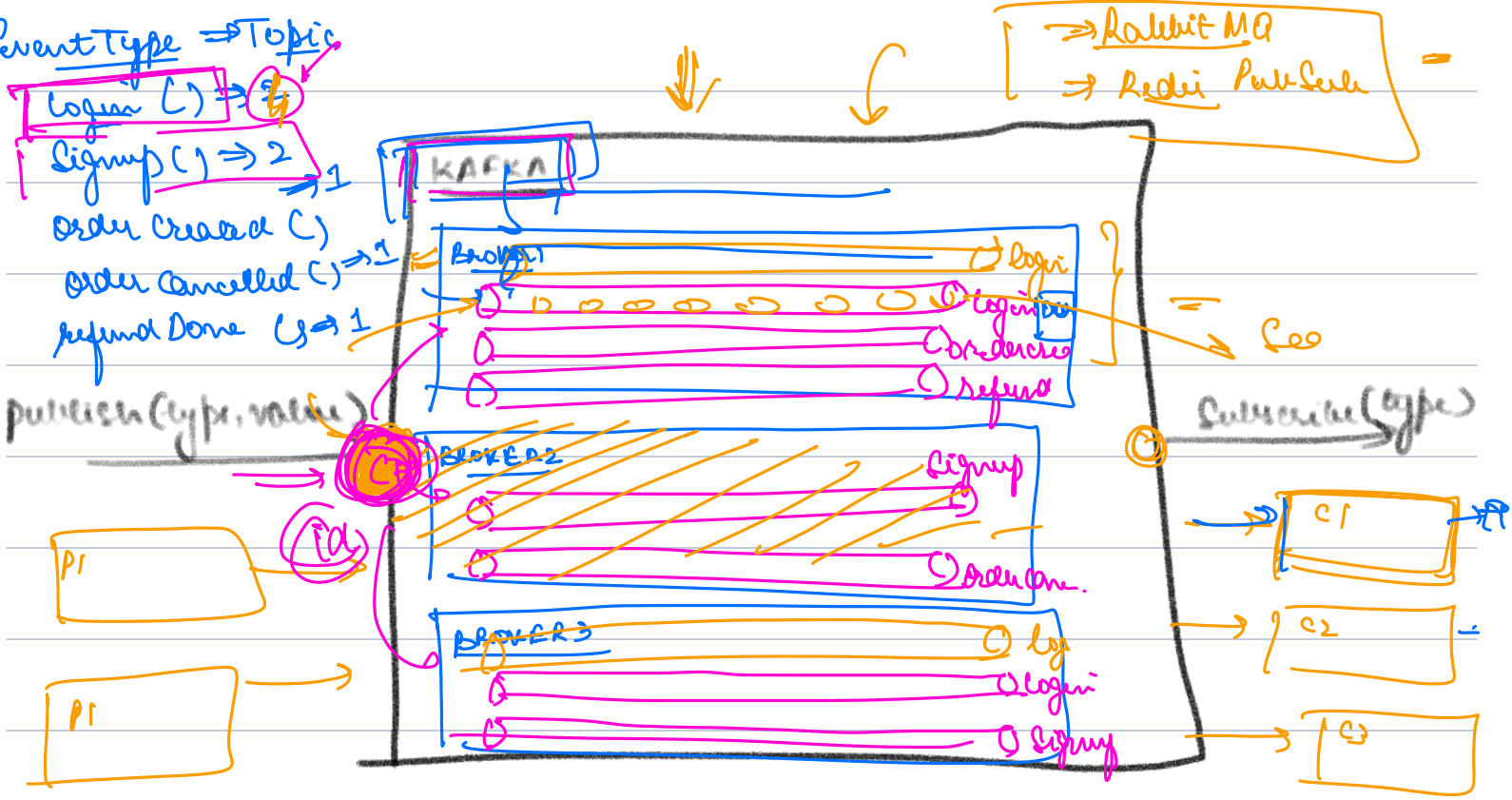
CONS

a.) uneven load dist

b.) a single topic (type of event) may exceed size of the broker (machine)



(1) I may decide to keep events of every topic divided amongst (x) machine

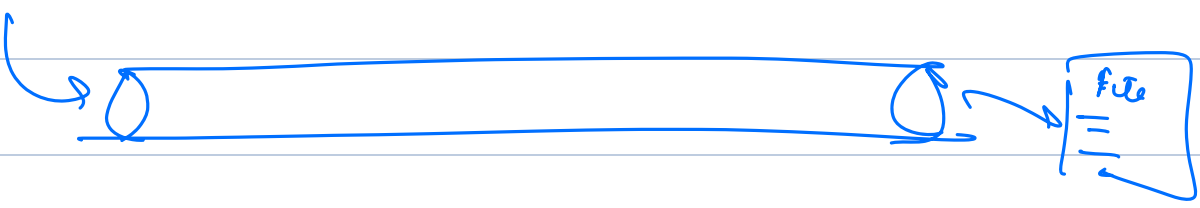


Kafka : automatically internally is figuring out what topics to put in what brokers.

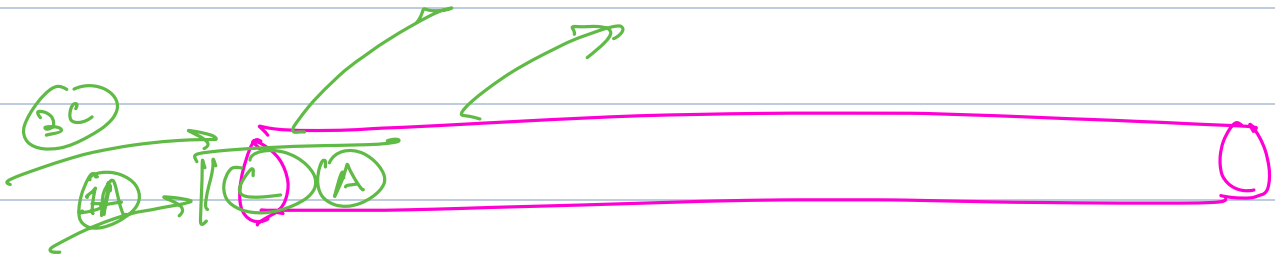
CH

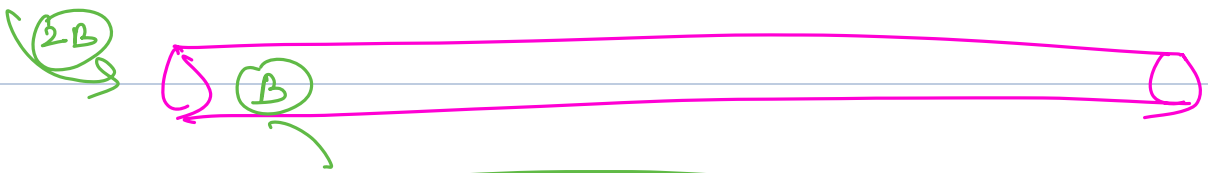
Hard Kafka Things

1. Order of Messages (FIFO)



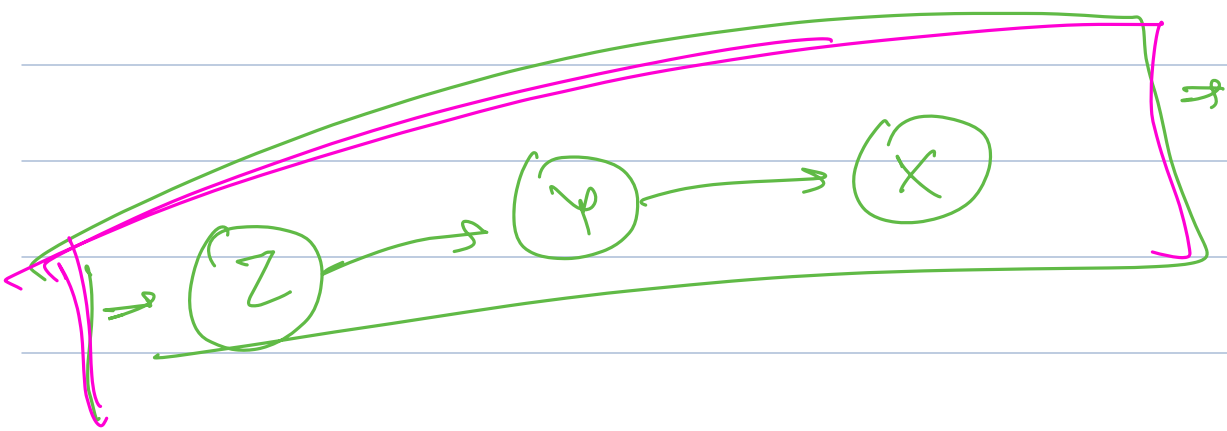
In 1 partition / topic very easy to do





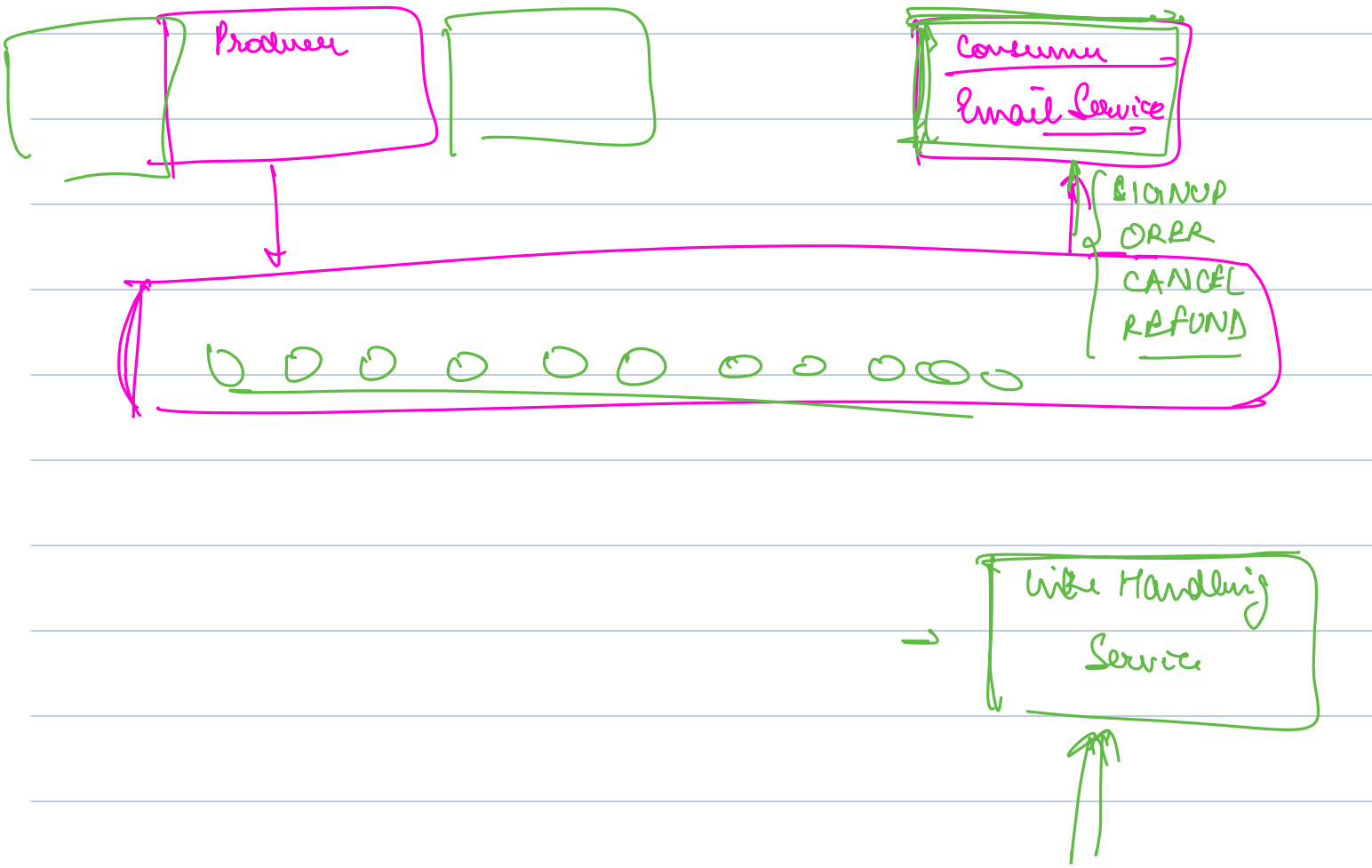
within a partition, messages will be ordered.

across partition ⇒ no guarantee

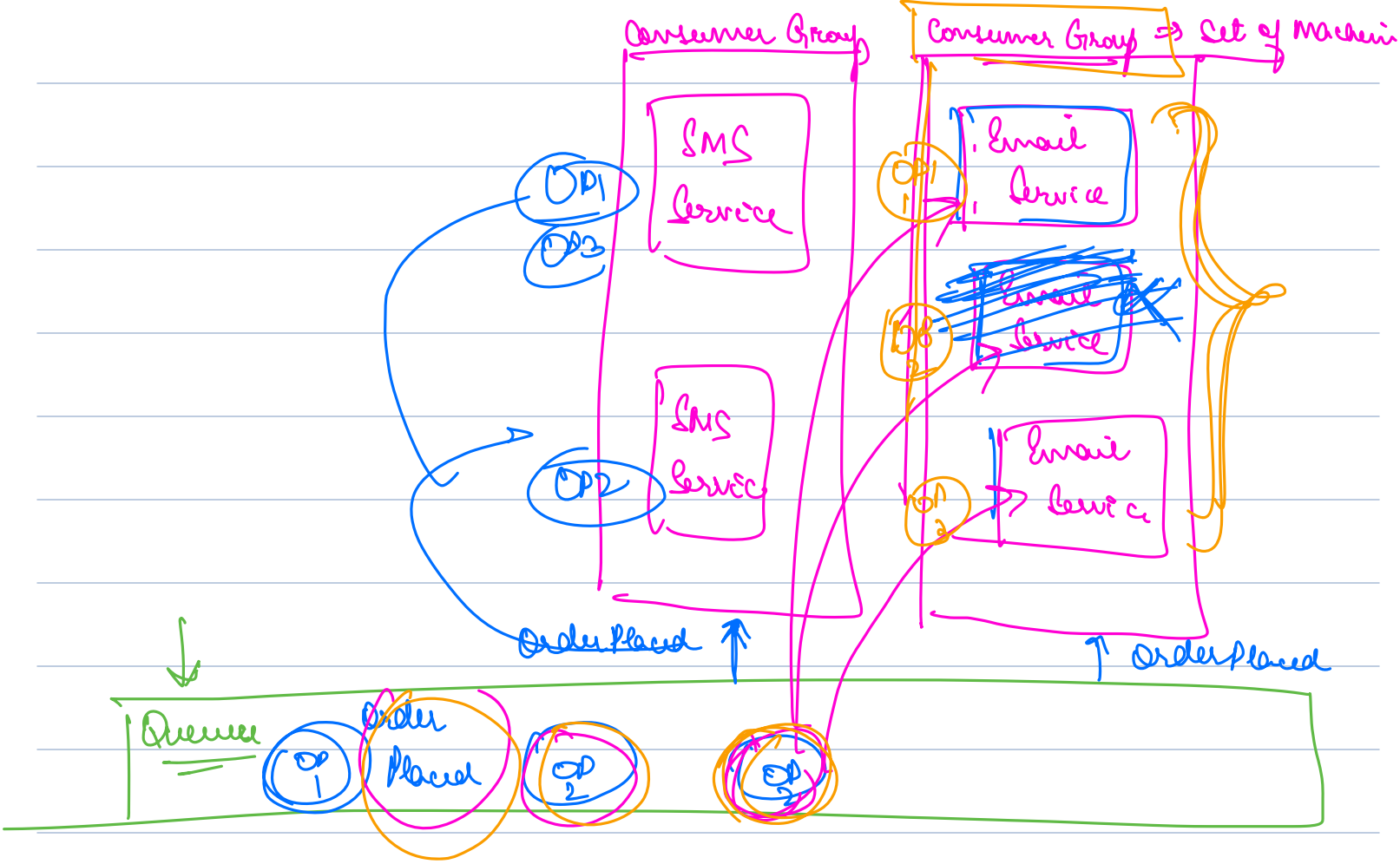


⇒ will have to borrow even if they are in the same partition

⇒ Kafka allows you to let your own Sharding Key

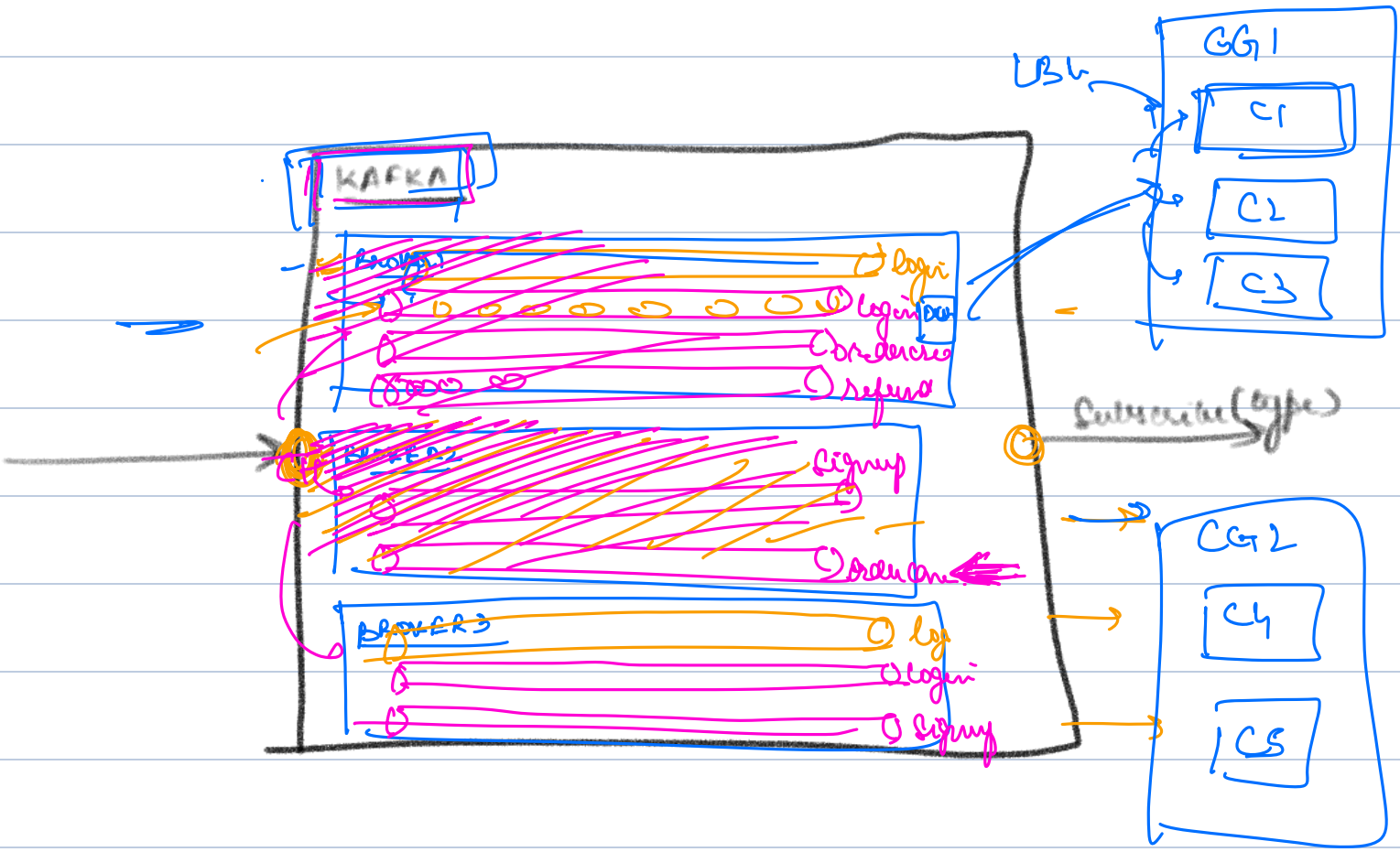

Producer
Consumer
Email Service
SIGNUP
ORDER
CANCEL
REFUND

Like Handling
Service
LIKE
DB



⇒ Only 1 consumer of each CG get an Event

⇒ uniformly distribute amongst all cons in a CG



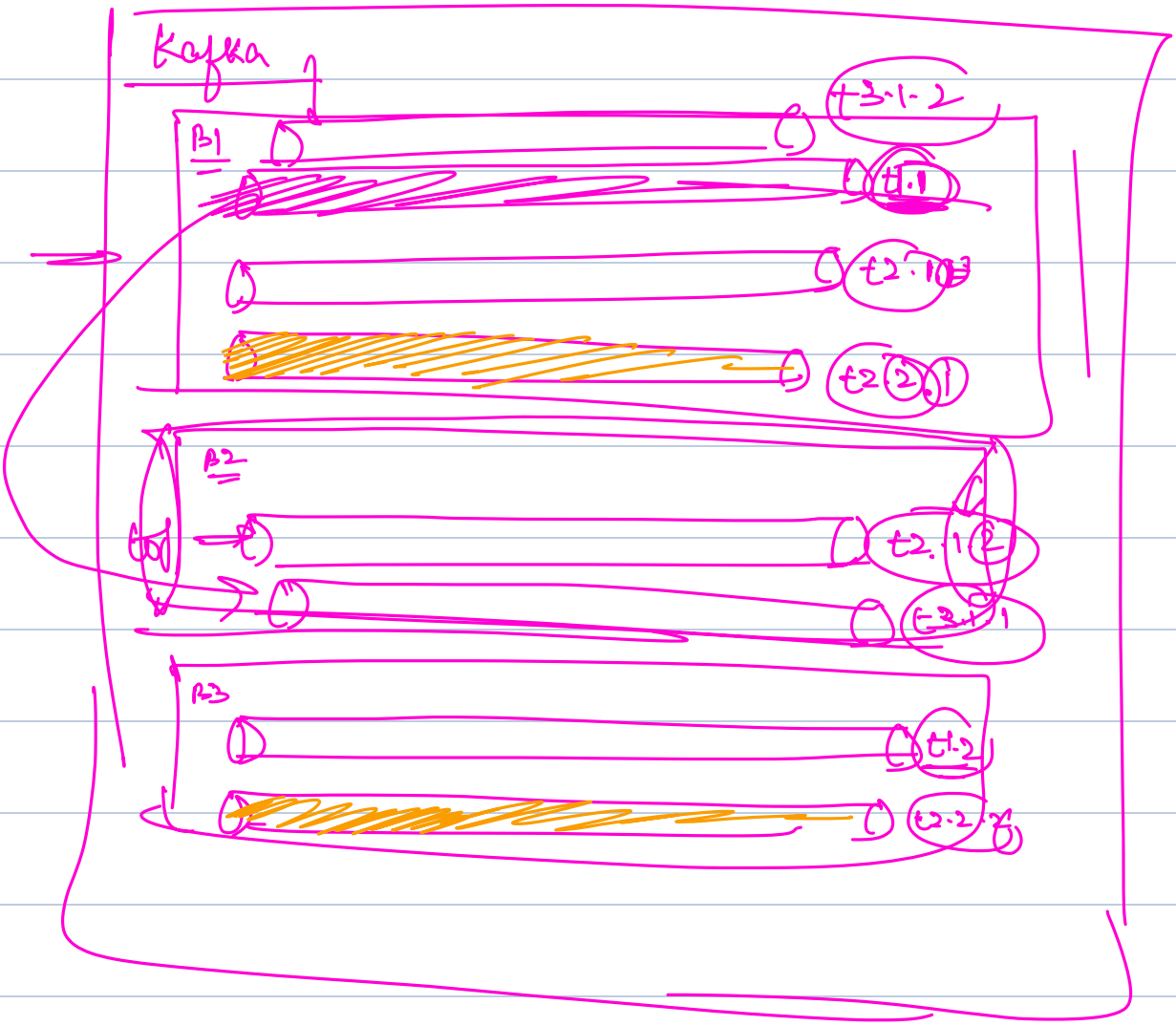
Kafka also supports replication

→ every topic
↳ # partition
↳ # replicas (≤ # brokers)

2 partition can be in same

Topics	#P	#R
t1	2	1
t2	2	2
t3	1	2

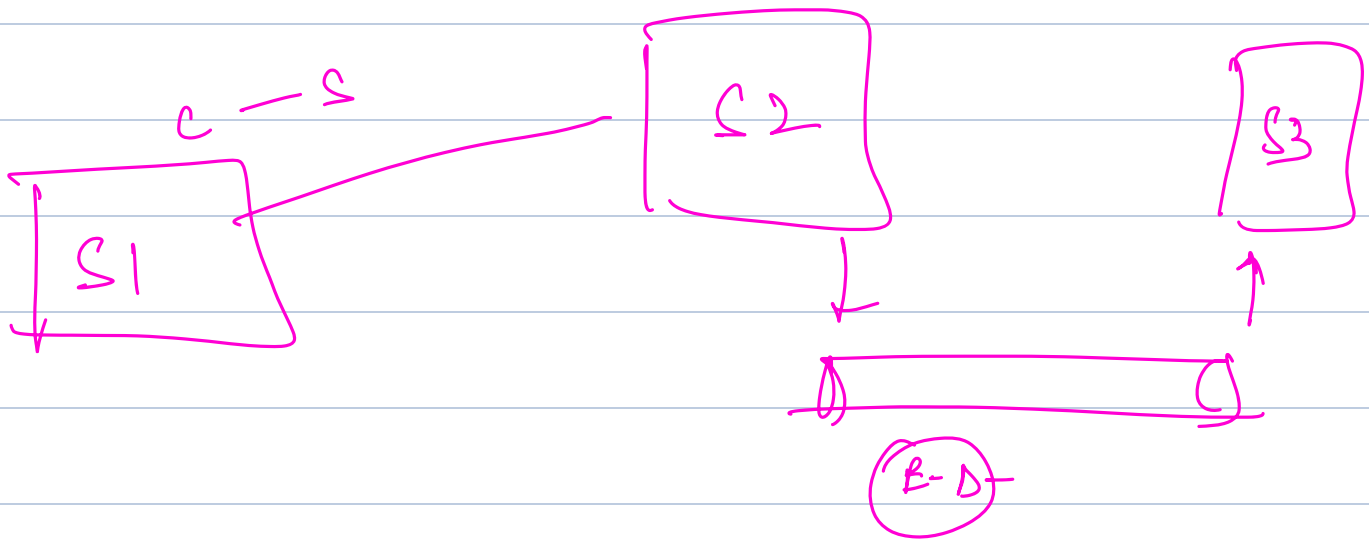
2 replicas must be in diff



partitions of a topic > # brokers

retention period → 180 days | 7 days

RoR / Airflow Sidekiq



Broker 1:1 Queue ⟷ TopicPartition

```
Class Producer {
    try {
=>      kafka.send Msg
    }
}
```