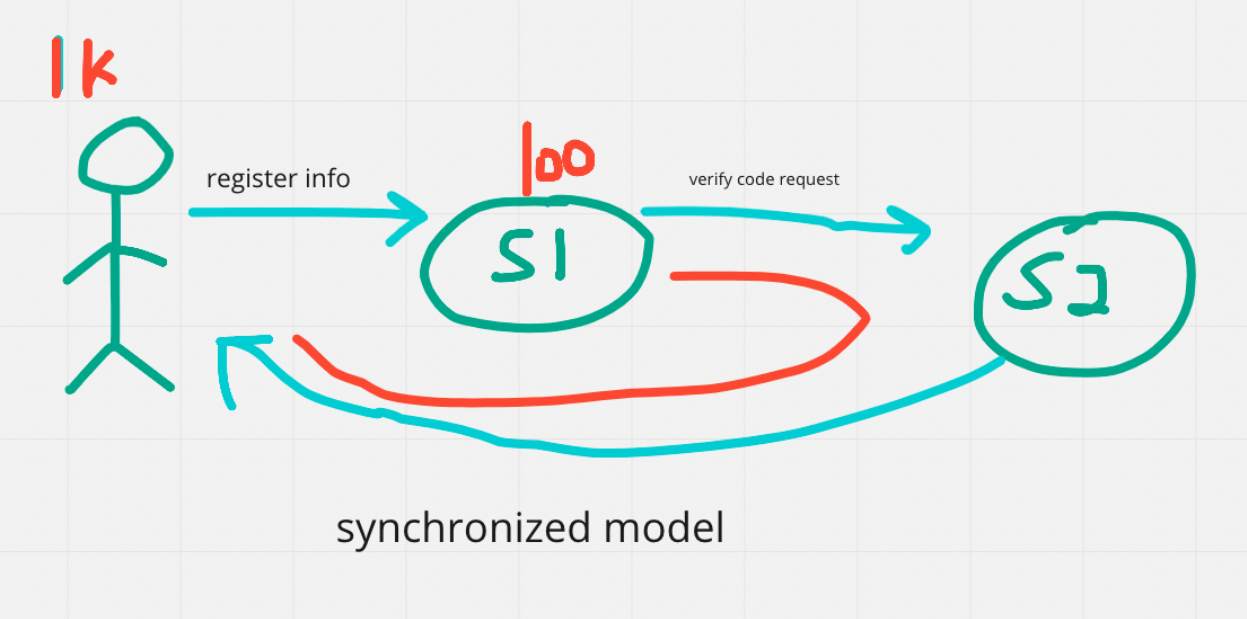
Spring - Mar 7

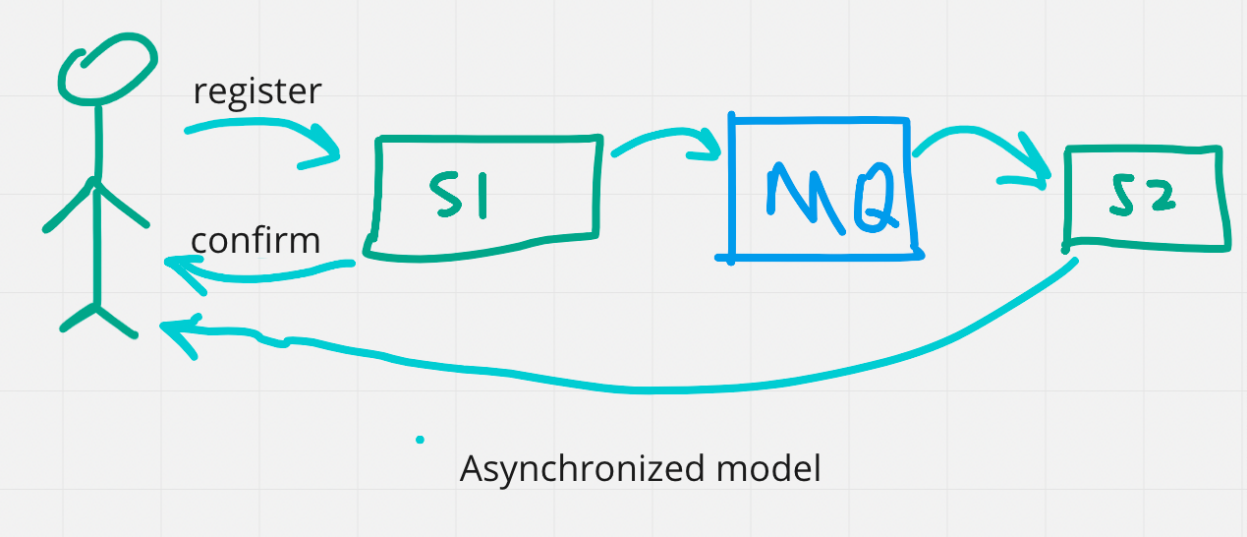
Mar 8 9am summary

- review message Q
- Kafka architecture
-

37, Kafka

why message queue (message Q)





sever needs to register user's info and store in db

- for register, we need to verify user

user start register → S1 request verification→ S2 send email to user for verification → User click email link to verify

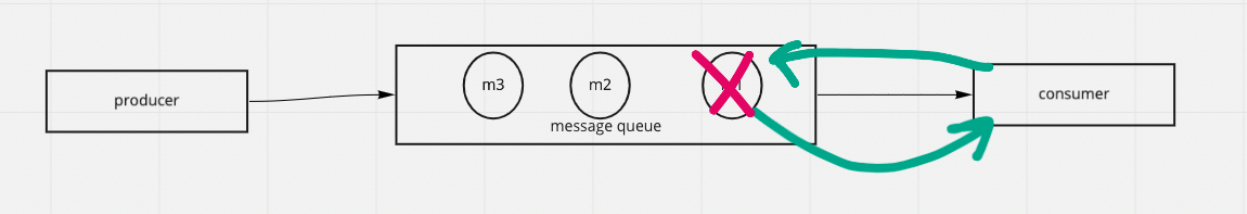
Problem: everyone want to access s1(limited threads)

use asynchronized model, s1 can send the confirm message to user immediately, and not waiting the whole process to finish

- loosely coupled
- cache
- flexible
- asynchronized communication

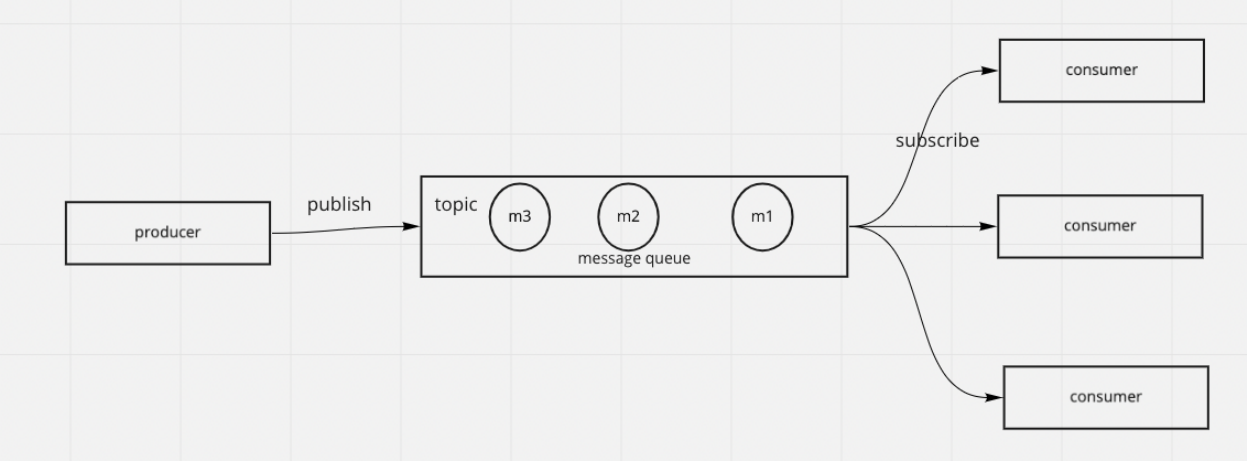
Message queue model

1. point to point (one to one)



- pull model (trigger by consumer)
  - consumer pulls the message from MQ
- push model (trigger by message queue is push model)
  - MQ pushes the message to consumer
  - When the consumer receives the data, it will send acknowledge message back to MQ. Then MQ deletes the m1.
- each message only can be consume once

2, publish and subscribe model (one to many)

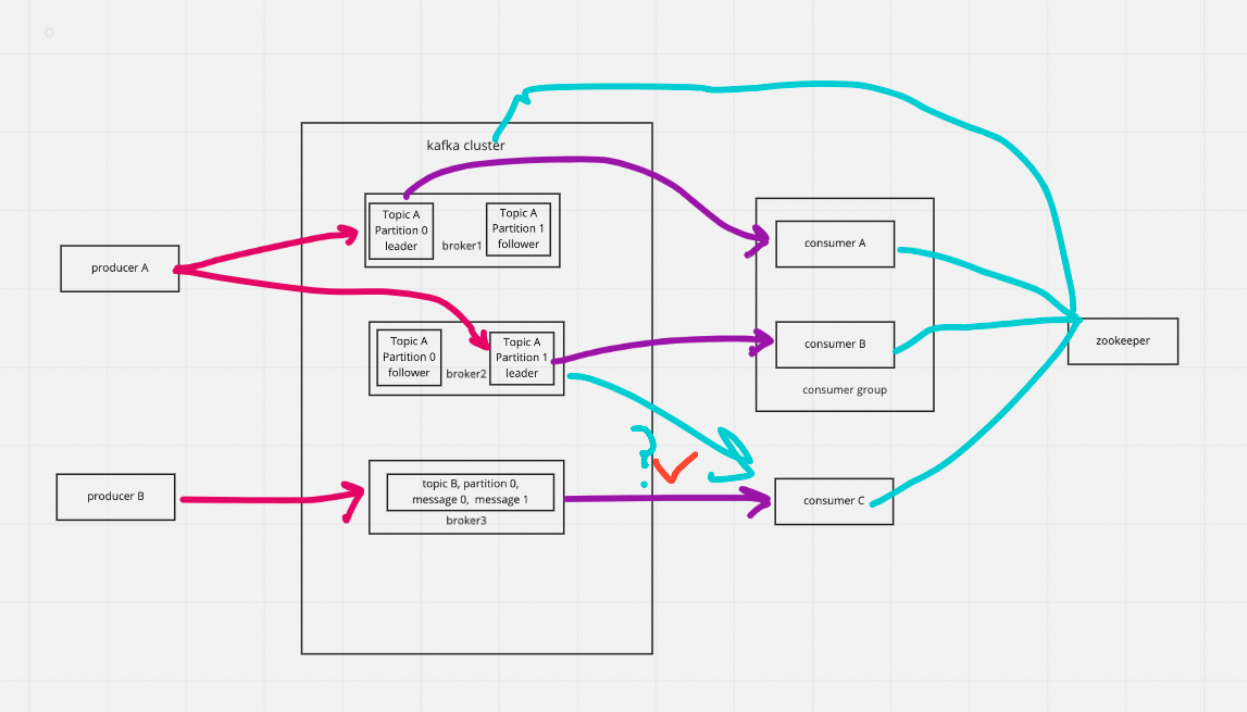


- producer publishes multiple messages to MQ, we called topic
  - consumer will register/subscribe to the topic, and retrieve the message from MQ

- message can be consumed multiple times

Kafka architecture(popular)

- another: rabbitMQ (message tool)



kafka use publish/subscribe model → different topic

- every cluster represent a sever

producer produce a message will send to leader partition

the follower will “sync up”

topic B just has one partition

zookeeper: coordinator will coordinate all the stuff

- manage kafka cluster

        - When leader node is down, zookeeper will handle it
        - the follow node for backup
    - manage consumers

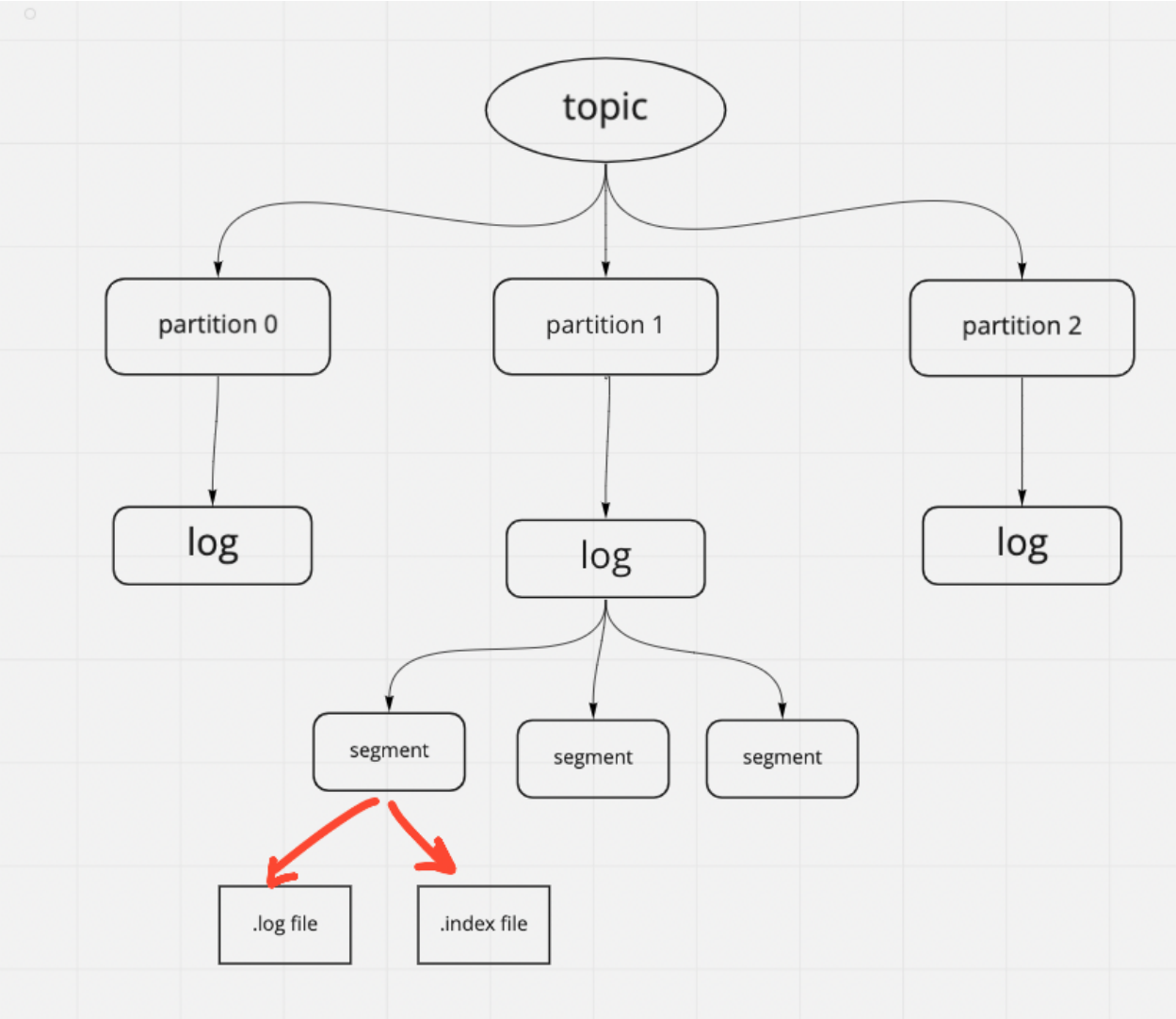
producer sends messages to leader node(partition) directly

- Producer
    - produce some messages
- consumer
    - betry information from
- consumer group
    - each consumer in the group can not consume same partition in the same time
- broker(physical sever)
    - kafka sever
    - each broker can carry different topic, and have different topics
- topic
    - theoretical idea
- partition
    -
- replica
    - backup
- leader
    - interacted leader and consumer
- follower
    - “thing up” data following the leader node

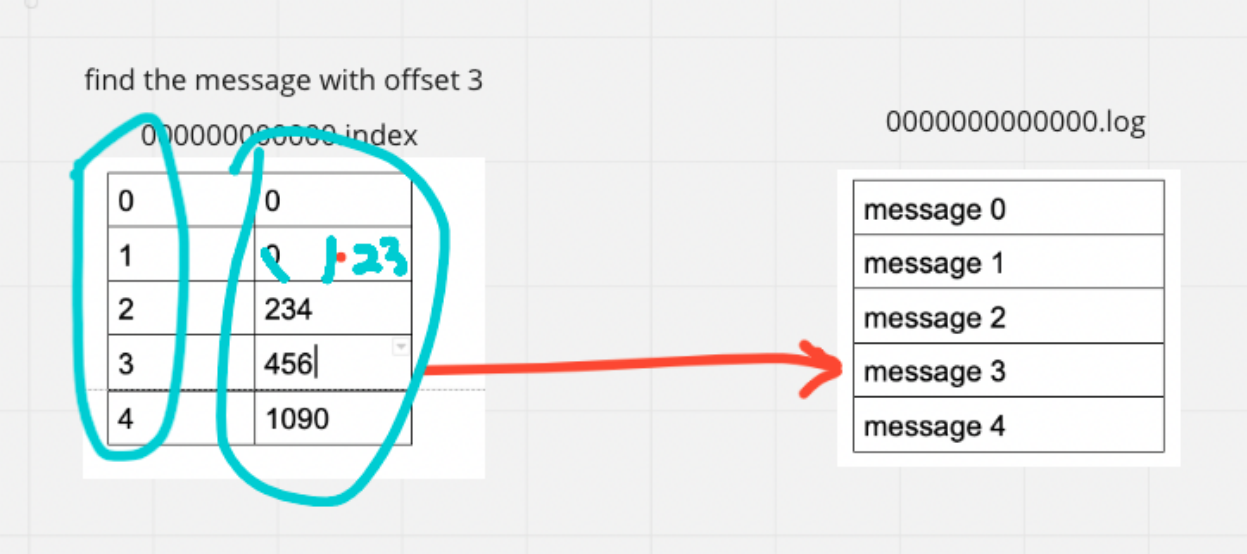
Dive deep into Kafka

file storage mechanism

file the data



we call all messages are “log”



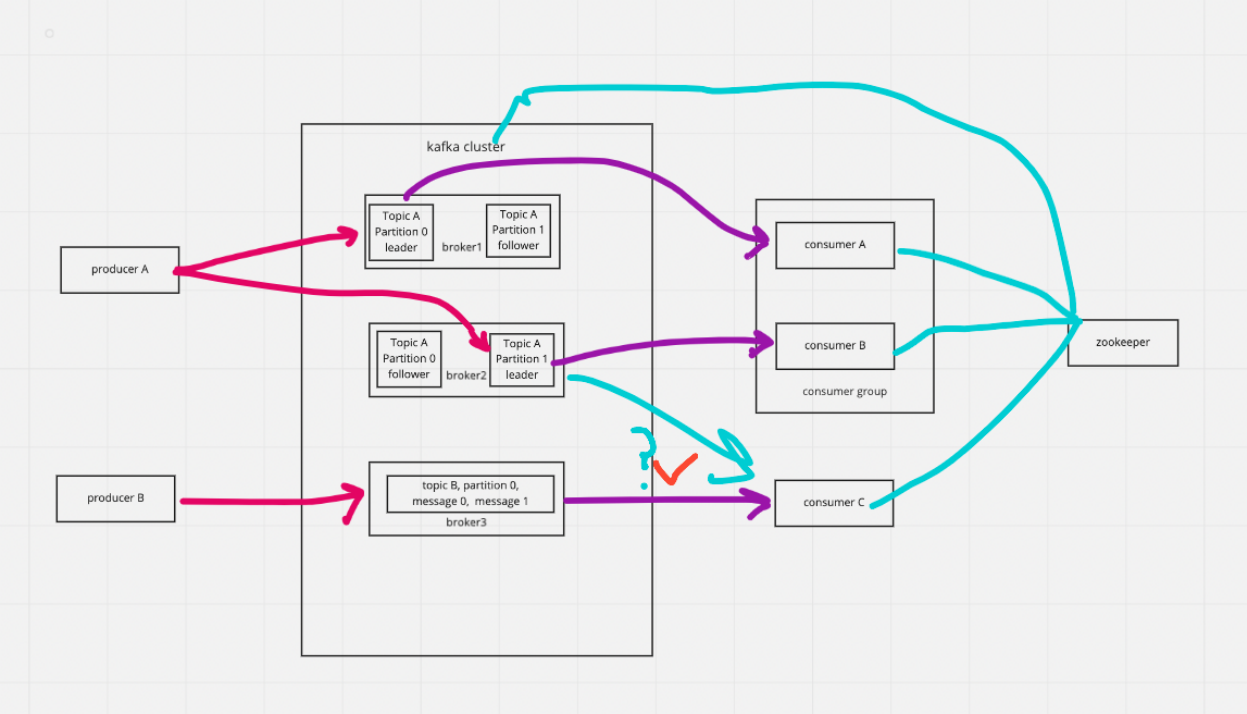
key-value

find the index first and jump the message by address

Producer

- why we need partition
  - divide big data to small partition
- ProderRecord(topic, partition, key)

  → producer put the message into ProduerRecord, and then send the ProduerRecord to the sever.
  - the partition principle
    - if tell producer → partition number
    - else if use key -> hashcode -> %
    - else round robin algorithm
      - P0 → P1 → P0 → P1...



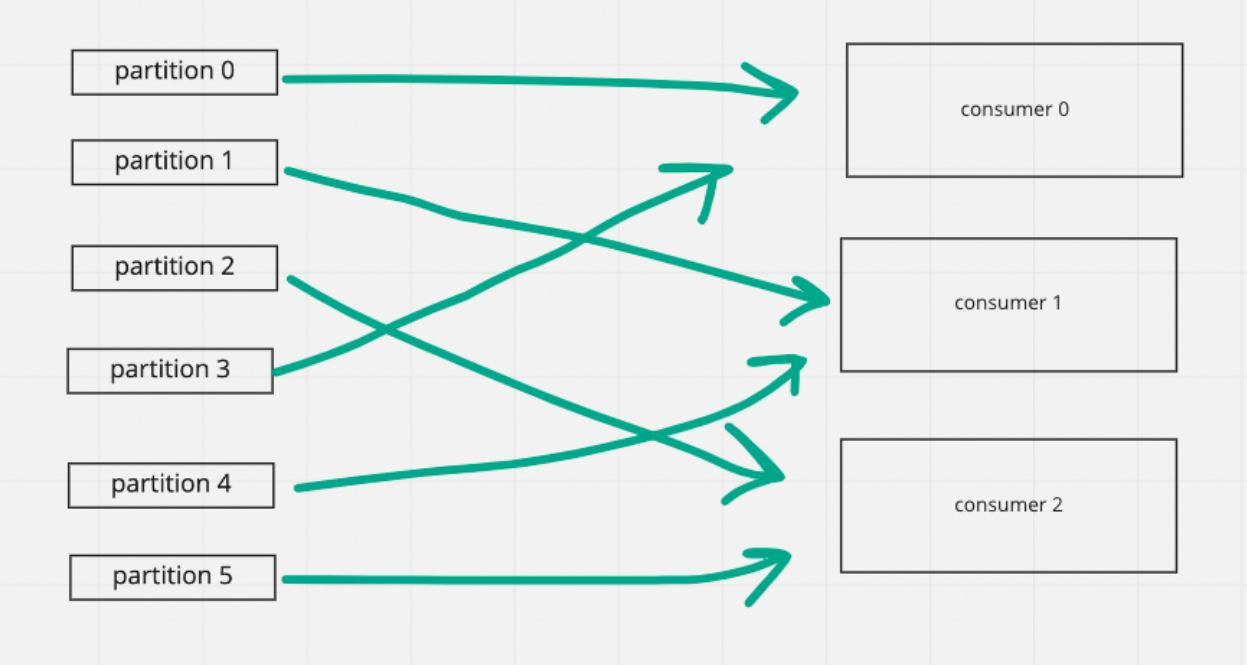
- how to guarantee the reliability
  - ack mechanism (producer)
    - When broker receives message, it will send acknowledge back to producer
    - three levels acknowledgement to handle a long waiting time of producer(wait ack):
      - 0, : at most once (producer does care if broker receives message) → non important data
        - may lose data
      - 1, P just care whether leader received
        - followers may lose data, cuz when leader send ack back, it will shut down. At that time, followers may not finish syncing data
      - -1: at least once → wait all the follows to finish syncing data
        - potentially duplicated data
  - ISR
    - In sync replica set

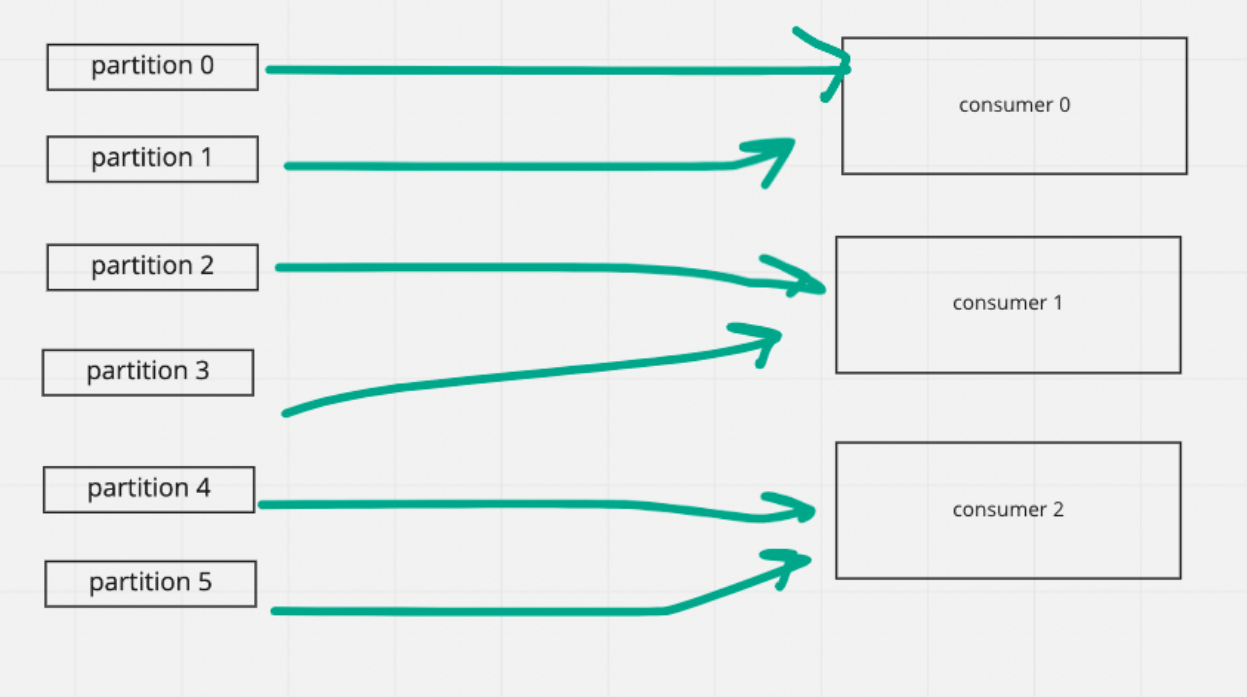
        - all the follow nodes in the ISR will sync up data from leaders
        - after sync, it will send ack back to producer
- exactly once (de-duplicated data)
    - at least once + incompetence
        - enable.idempotence = true;
    - broker do: <PID, Partition, SeqNumber>

      each time producer sends message to broker. Broker does the de-duplication, based on primary key (<PID, Partition, SeqNumber>). check they are three whether exist
        - set each producer a PID
        - set Partition number
        - set sequence Number for each message

Consumer

- partition assignment strategy
    - round robin
    - range





Rebalance

- ConsumerRebalanceListener
  - OnPartitionRevoked (before rebalance, commit the offset)
  - OnParitionAssigned (after rebalance, seek offset)