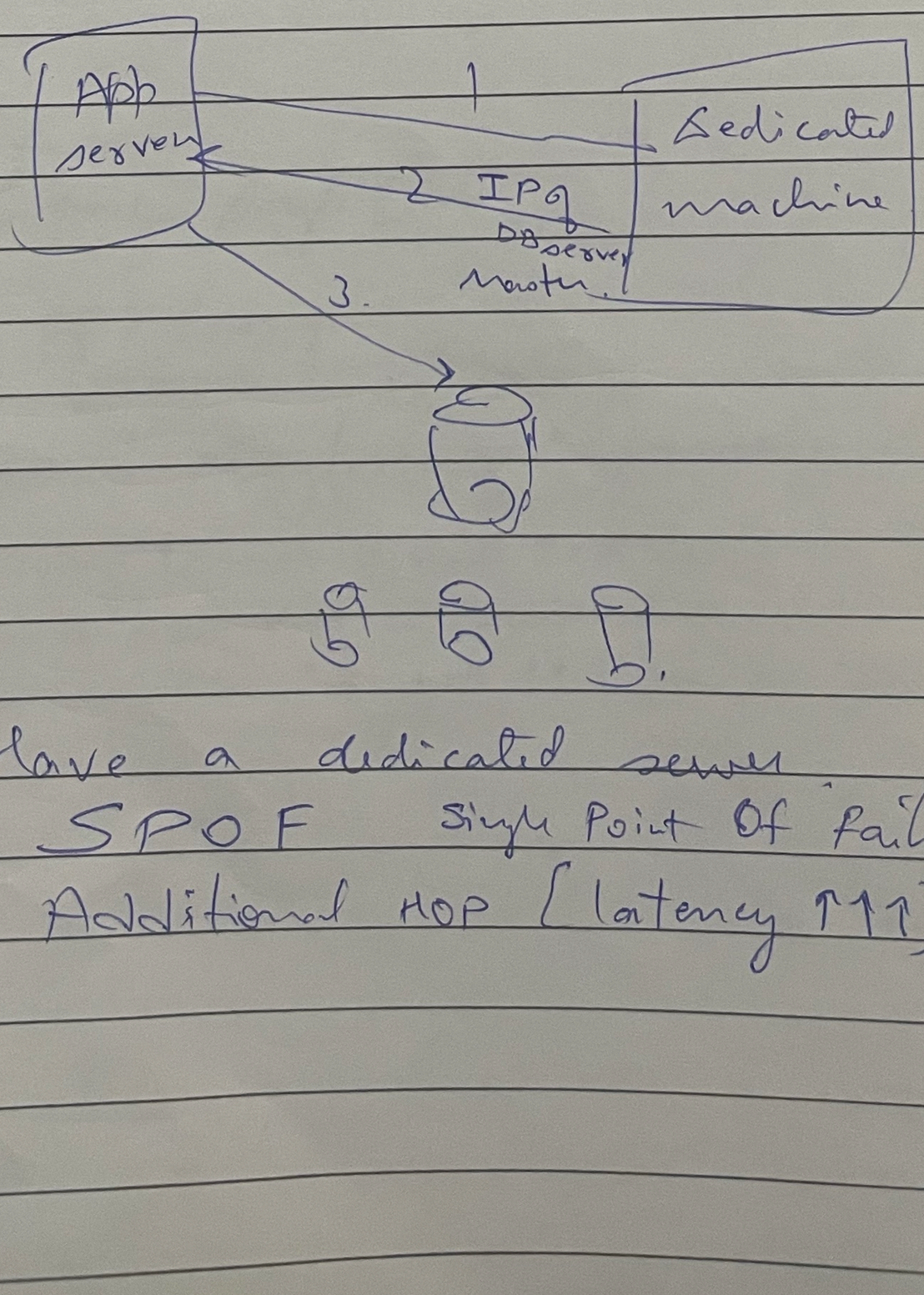
SPOF → Sol^n: Replication (Introduce Replicas)

Zookeeper & Kafka

↳ Config Management System.

State Manager.

Approach - 1:



→ Have a dedicated server.

Cons 1) SPOF Single Point Of Failure

2) Additional HOP [latency ↑↑↑]

Mostly an large level system to maintain data, usually a proper segregation & folder structure is done, kind of tree like structure.

Solution: Zookeeper [ZK]

→ ZK is exactly File system. (Same like Linux)

root → services

/ → Payment Service = Service A
- → Appserver
- → Payments DB
- → Global Cache

/ B → Order Service

/ C →

/A/payments DB
- → ShardingConfig (file)
- → Shard 1
- → Shard 2.
  - → Master IP
  - → master-port.
  - → Slave-A-IP
  - → Slave-A-port
  - → Slave-B-IP
  - → Slave-B-port.

files.

Q Why cant we maintain all the above config in just a ~~xxxx~~ (XML), as it is also structured? This being a multiple server system, we need data to be consistent. So if the data is in diff files, we can obtain LOCK on the files to keep it consistent. We can't have race conditions.

/A/payment DB/.
↳ aws_key.
↳ Sharding Config
↳ shard1
↳ shard2.

It can have all types of Config not only Shard config.

ZK calls all its files as Node

ZK

Persistent Node (OR) Persistent File.

Ephemeral Node (OR) Temp files

Simple files that rarely changes (once its made)
→ AWS_key
→ Shard Config

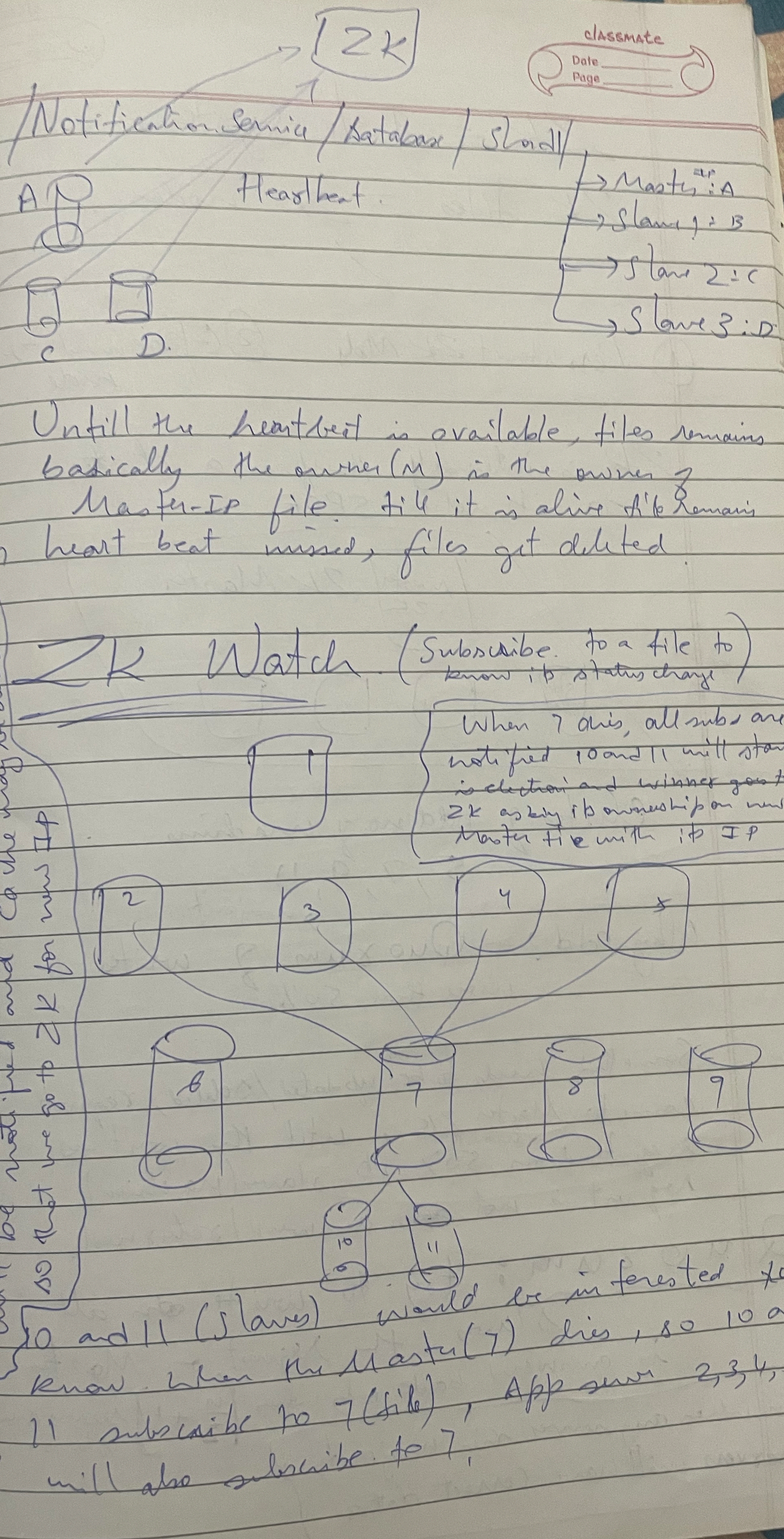
Nodes which has to live only till its underlying machine is ACTIVE

→ Master IP

10.0.0.31 /M/ /Notification Service/Database/Shard3/

→ Master IP 10.0.0.31

If M dies its file in the above folder should also be deleted as its no more connected



Untill the heartbeat is available, files remains basically the owner(M) is the owner of Master-IP file. till it is alive it remains heart beat missed, files get deleted.

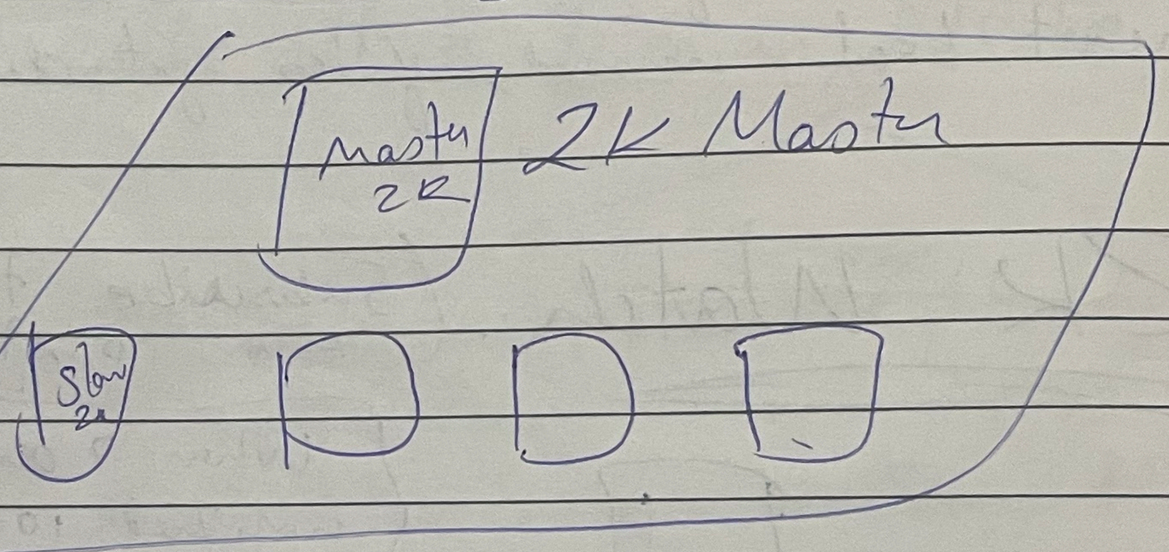
10 and 11 (slaves) would be interested to know when the Master(7) dies, so 10 and 11 subscribe to 7(file), App server 2,3,4,5 will also subscribe to 7.

ZK = File System

ZK files = ZK nodes.

① Persistent ZK Node ② Ephemeral ZK node.

ZK is internally itself is a cluster so its not a SPOF.



Cluster of odd no. of machines

5, 7, 9, 11 → majority.

Why odd? Quorum of writes is more than 50%

Something needs to be updated/deleted, request come to Master ZK, util the writ is registed on more than 50% of slaves (machines) request is not ~~ans~~ answered/returned.

Quorum: try to write on all slaves, the request on Master is answerd back only when $\left[\frac{n}{2}+1\right]$ servers are slaves.

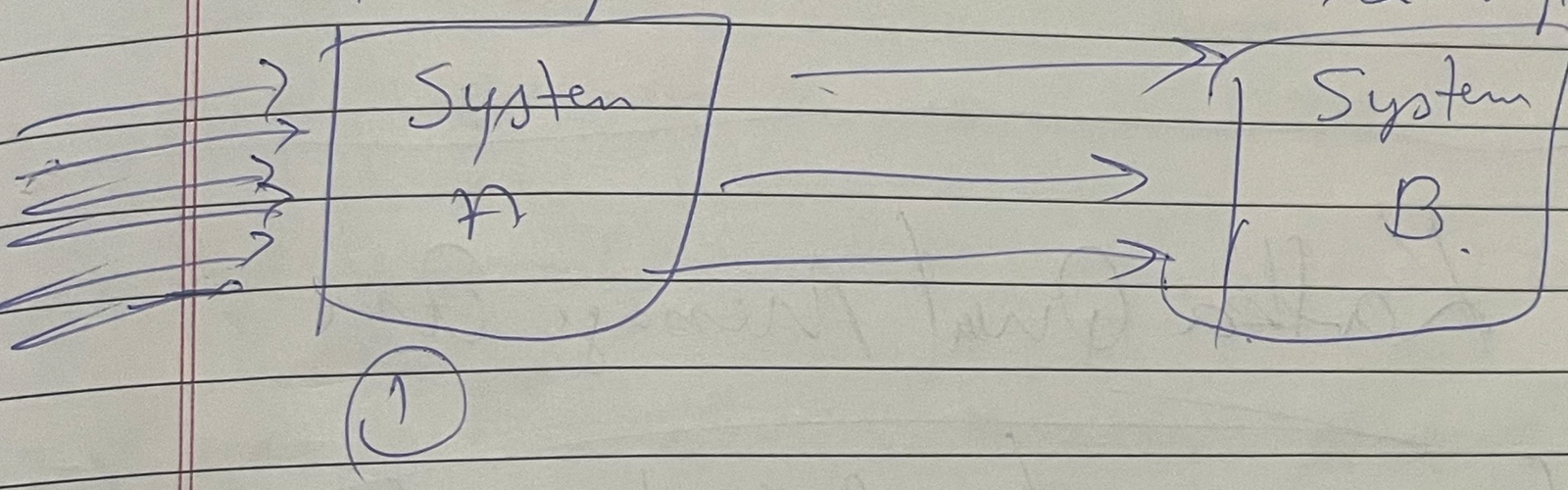
Even when any server dies majority of servers will have correct data

Gossip Protocal.

↳ ~~for Master~~

↳ For Master election.

Async Tasks.



② Rate of Production ≠ Rate of Consumption.

③ If we want to do aggregations in b/w Queues can be done. (Digests)

outlook digest.

ASYNC Tasks → Persistent Queues.

Ex: Kafka. Publish Subscribe

RabbitMQ. Pub-Sub.

Message Queues.

Redis
- DB
- Cache
- Message Queue.

Kafka

Principles

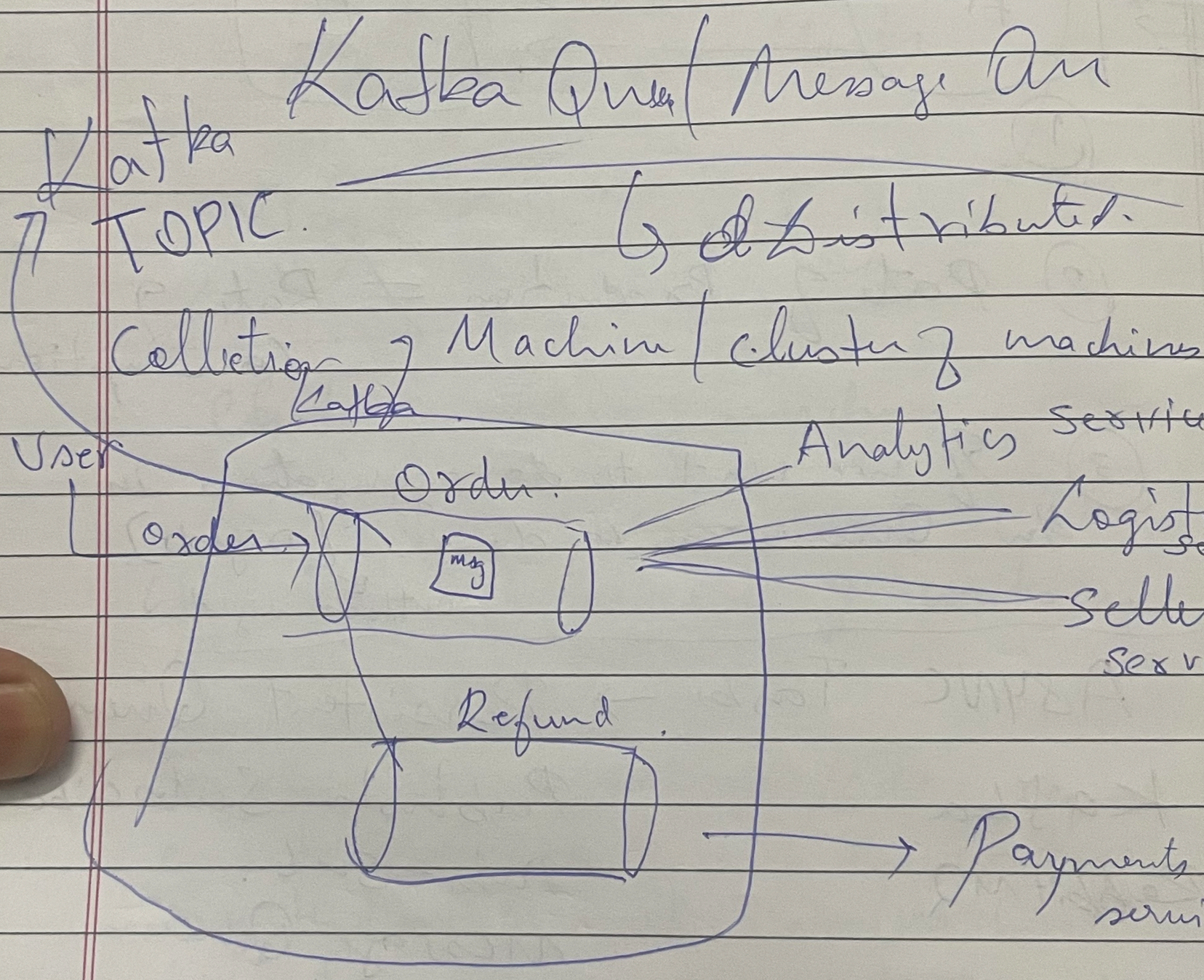
1) Distributed ≅ Horizontally scaled easily
   Ex: 10 Million Msg/sec.

2) Persisted Queue.

Kafka Queue / Message Qu

Kafka
TOPIC.
↳ Distributed

Collection → Machine / cluster of machine



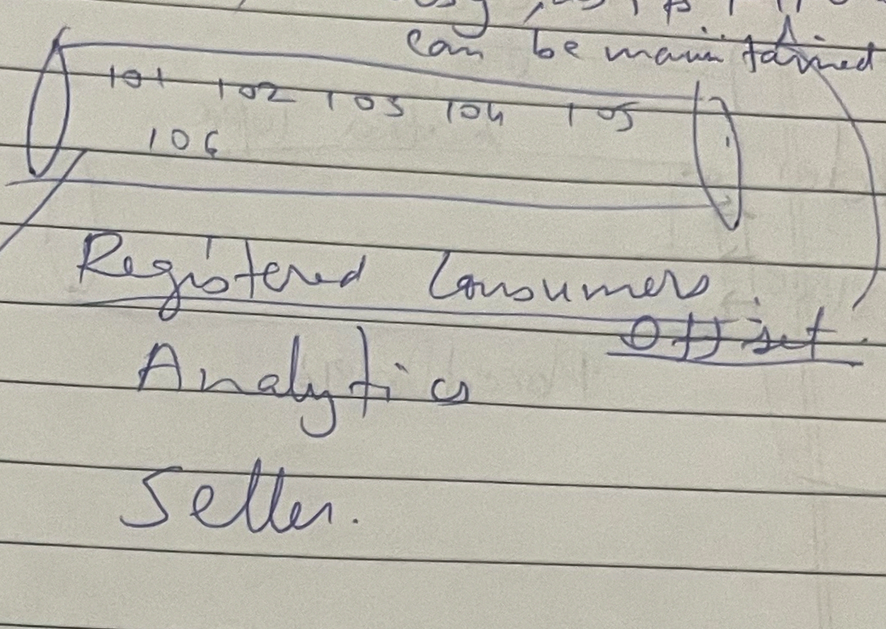
Msg should be read by all 3 services and only Once.

Web hooks.

Working:

FIFO

msg-id.



→ Msg doesn't get deleted after reading

→ Msg ~~will be set with~~ TTL

→ Msg will be deleted only when TTL expires

Kafka internally used Zookeeper to save configs ~~such~~ as OFFSET.

Msg could/should be kept lite, have only orderID or any required info only, any more info required, DB call should be made to get more info.

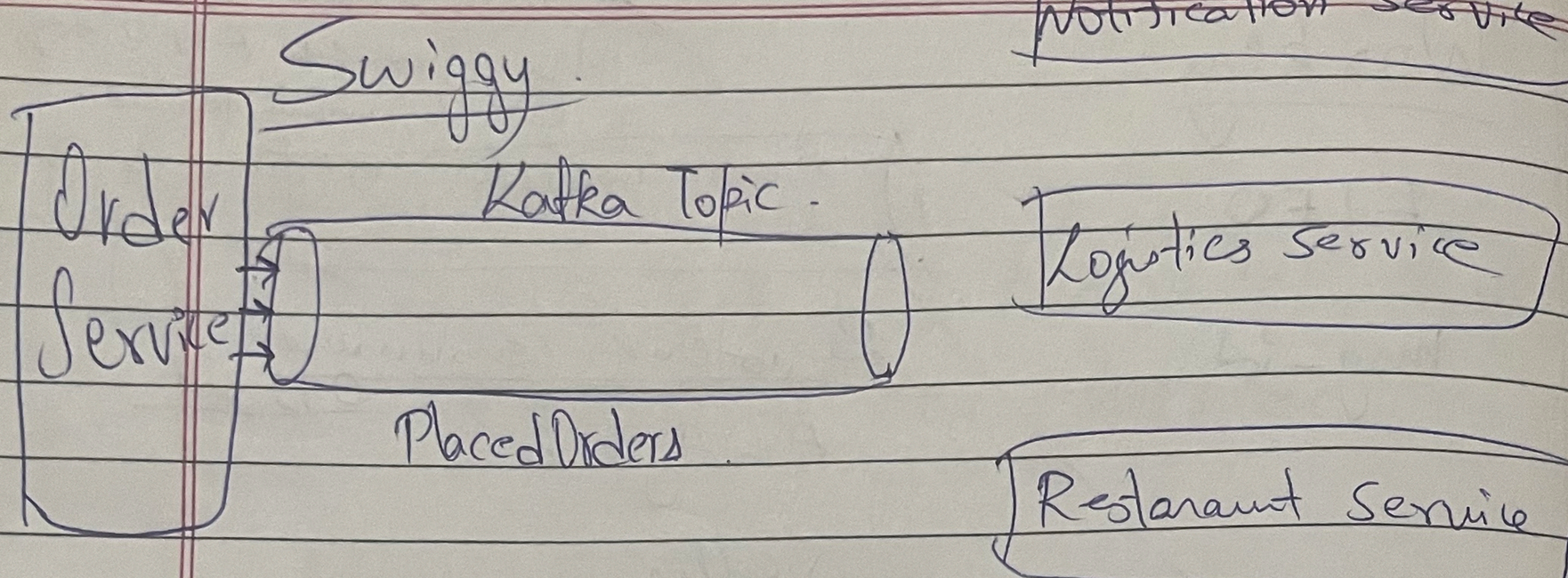
Problems

→ As Queue is FIFO (ordered) all msg will have to be in one server (one topic in one server)

→ Can't be read parallely as it's FIFO outputted.

18/12/2025

Kafka Contd...



→ If there are many Topics in a server
⇒ If we encounter space issue, separate Topics into separate servers.

→ There may be a situation where 2 entire machine may not be sufficient to store all msgs
1 Queue(Topic) also ?

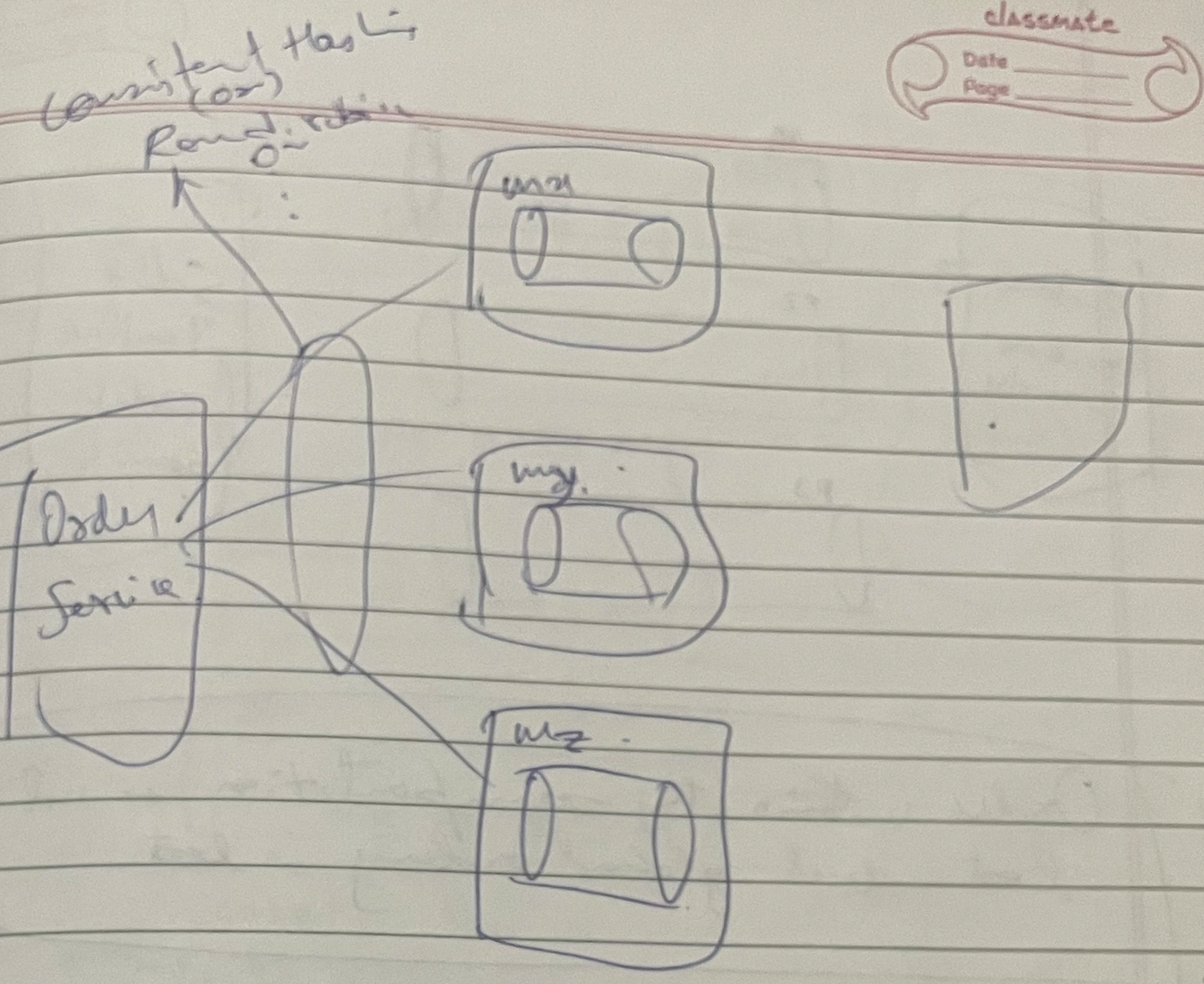
in notes

The Usual way to solve above problem → SHARDING

But we can't use sharding directly before understanding it...

↳ Queue is FIFO [order should be maintained]

Problem: When we divide a Queue across multiple machines, it no longer remains One Queue
It becomes multiple pieces of One Queue
→ if we put up msg in any fashion (CH or RR) it will be difficult to maintain order (FIFO) to read the msg.



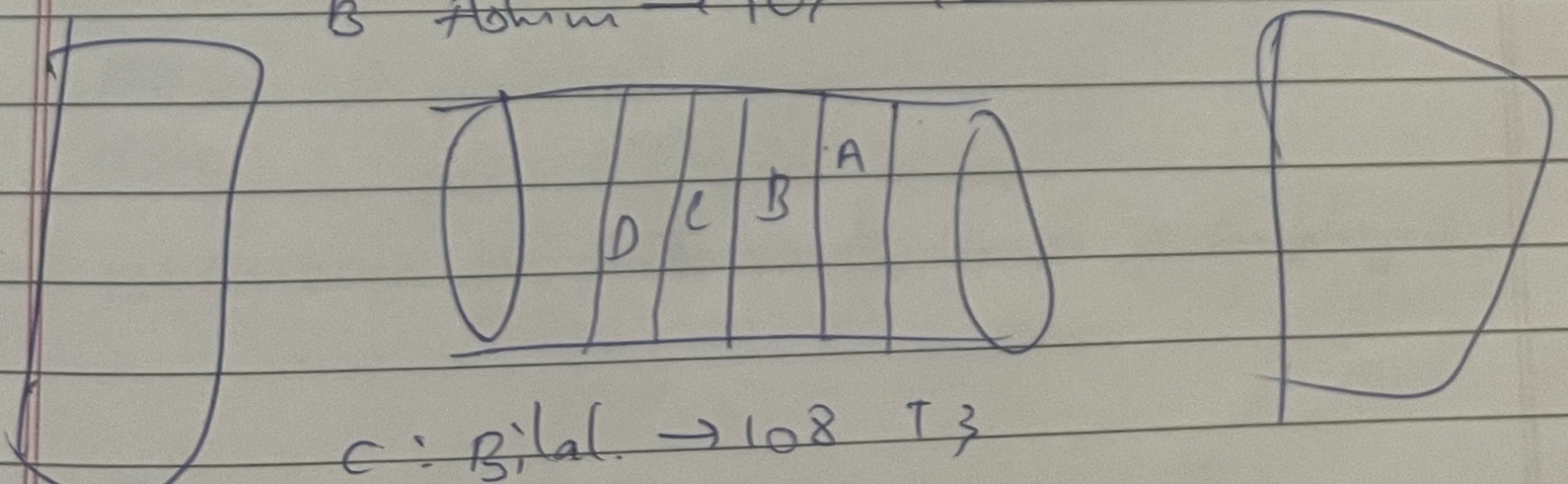
→ One possible sol: get Timestamp from all partitions (m_1, m_1, m_2) and pick the latest (very very unoptimized)

in notes

The Kafka Topic when partitioned Kafka no longer gives FIFO guarantees across all partitions

A: Ashin → 107 T1

B Ashim → 107 T2

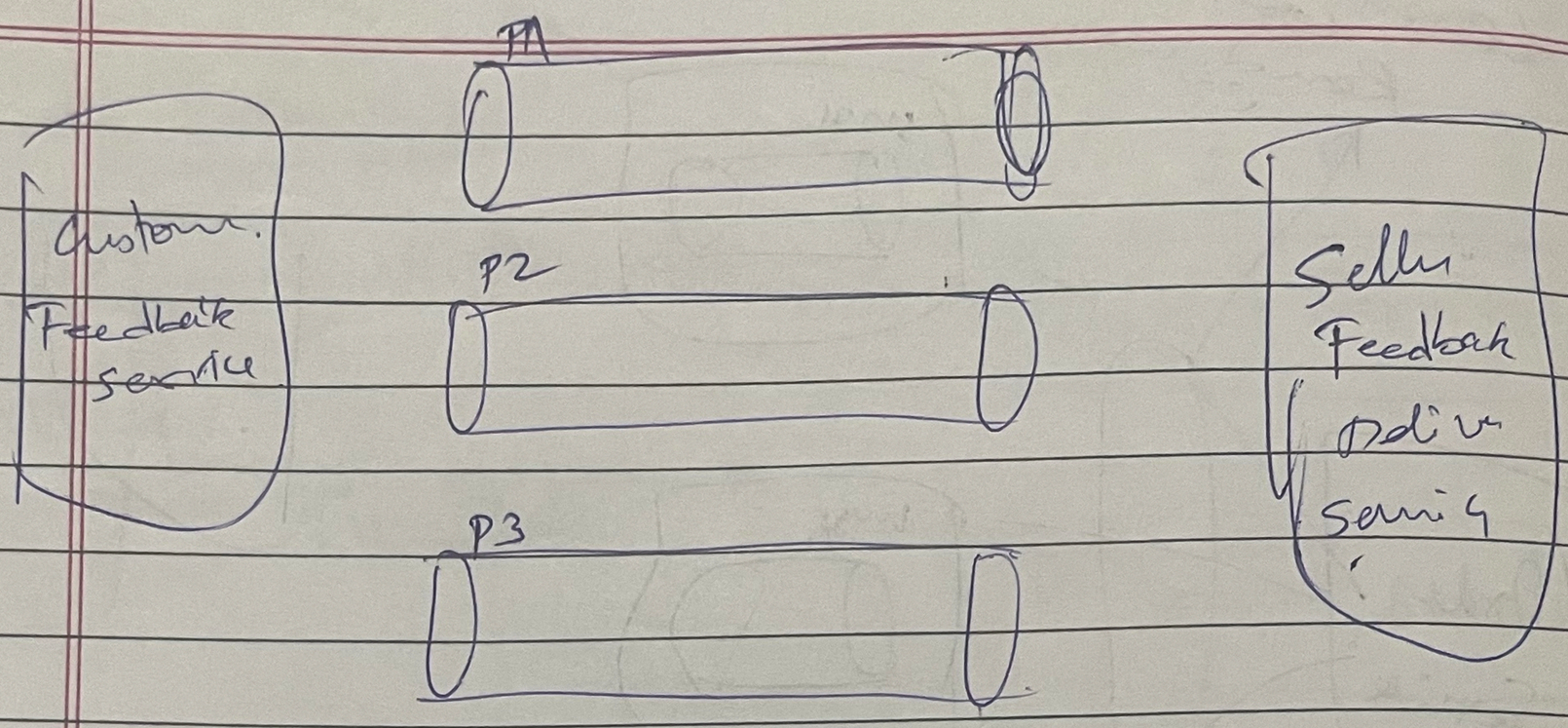


C: Bilal → 108 T3

D Bilal → 107 T4.

→ the order is imp. with partitioning & Recieve as well. Because T2:

T_2

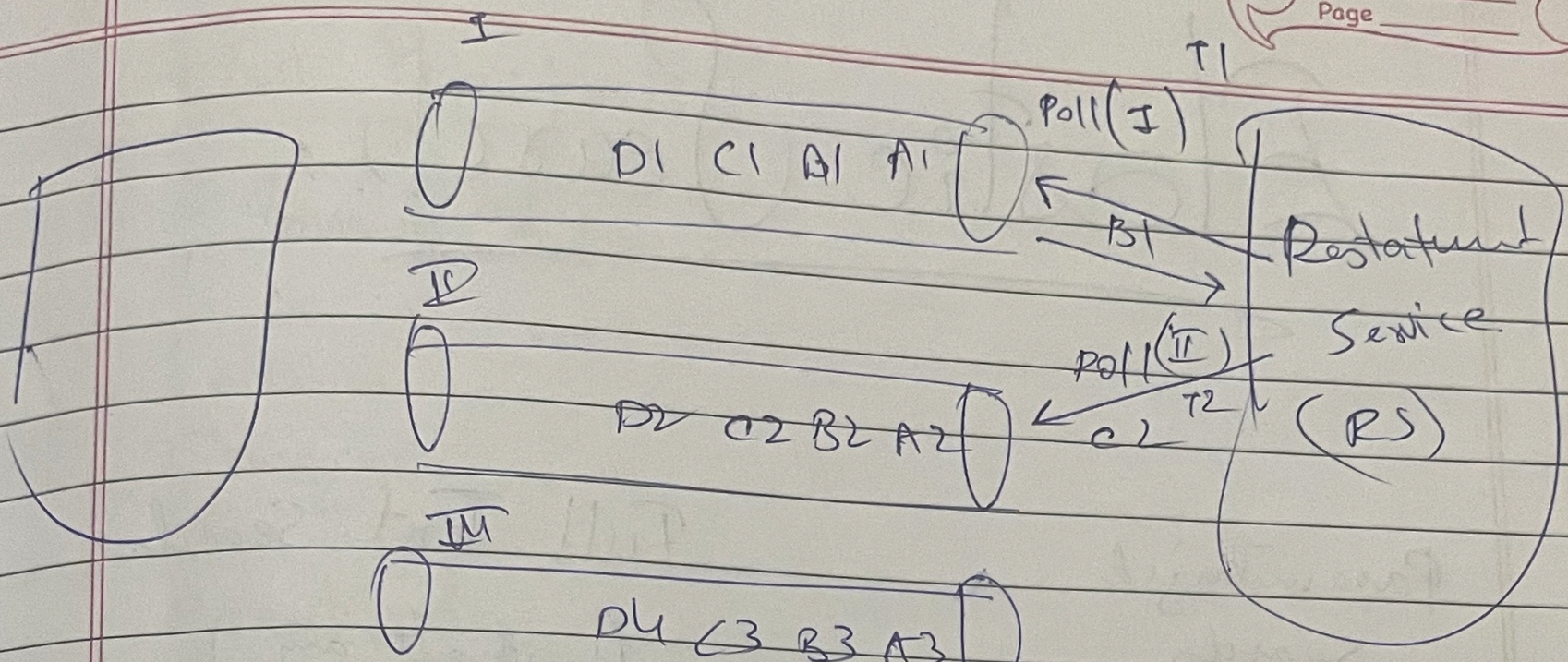


* Order within the same partition is not lost but global ordering is lost

% Seller wants to read in sequence (or) it is imp to seller is read in order maintain CH on seller ID

→ basically depends on the consumer servig

→ % we have multiple consumers, we have to decide on one CH key and every other consumer will have to leave with it (Trade-off)



Offset. T1

RS I → A1 B1

II → B2 C2 T2

III → A3.

IV → C4.

% RS is reading one by one from partitions we can use this idea to increase performance and employ different RS subservers to read from diff partitions separately

RS1 read from partition I

RS2 " " " II

RS3 " " " III

RS4 " " " IV

RSS will be waste even it employed