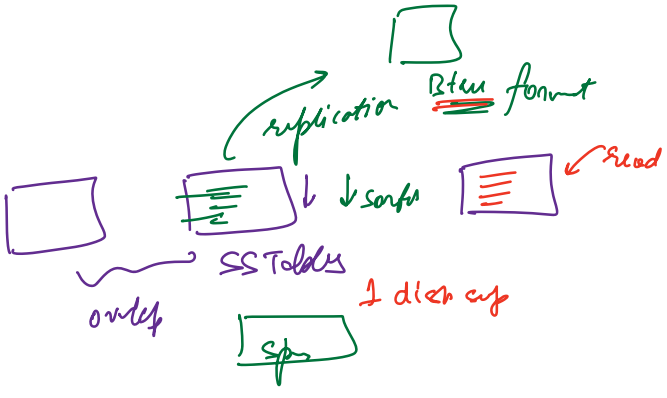
Agenda

- Problem 1
- Zookeeper
- Problem 2
- Kafka

starting at 9:05

LSM tree

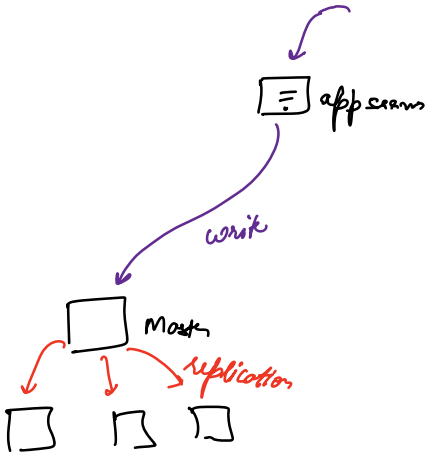


Problem -1 Consistent State Tracking

Master - Slave architecture

↳ all writes go to → master

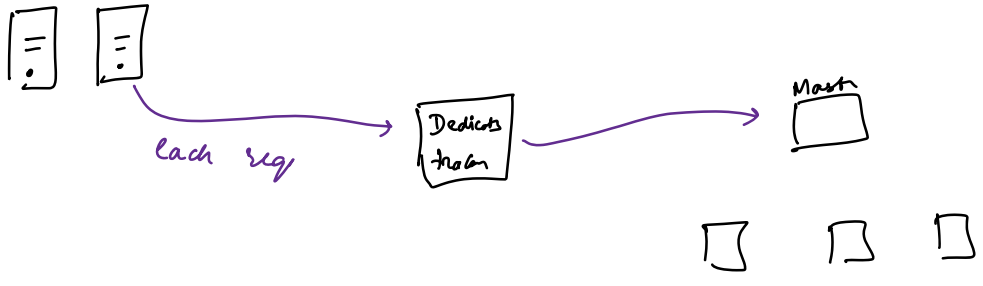
↳ app server must be aware of who the master is (all app servers agree on the same master)



Problem → keep track of the master

↳ when master dies → re-elect the master

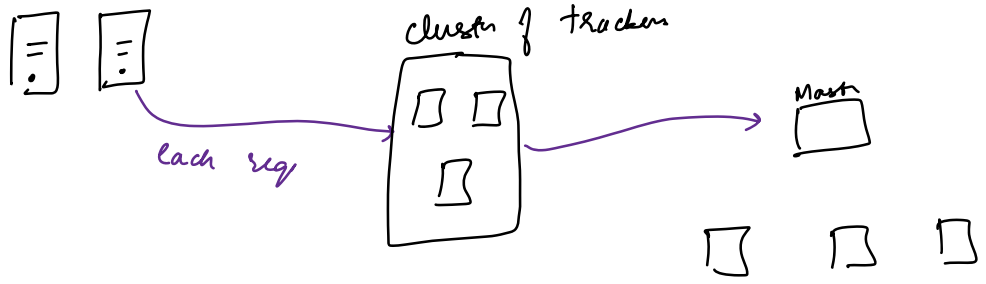
Approach -1 → dedicated machine → keep track of master



① Extra hop for every request

② Dedicated Machine → Single point of failure

To solve (2)



→ if master changes, how do they keep track of it

→ how to ensure that all machines agree on same master

Zookeeper → stores configuration in distributed settings

tracks (config) data in a strongly consistent manner.

root folder

/

/ config /

/ config / aws-key → files are called 'nodes' ZK Nodes in Zookeeper

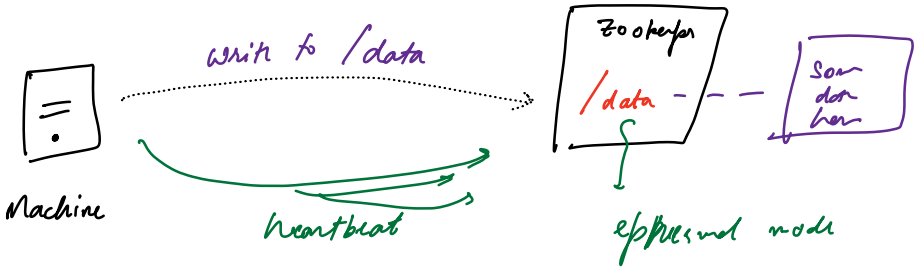
/ config / dp-ip

/ master

Types of ZK Nodes

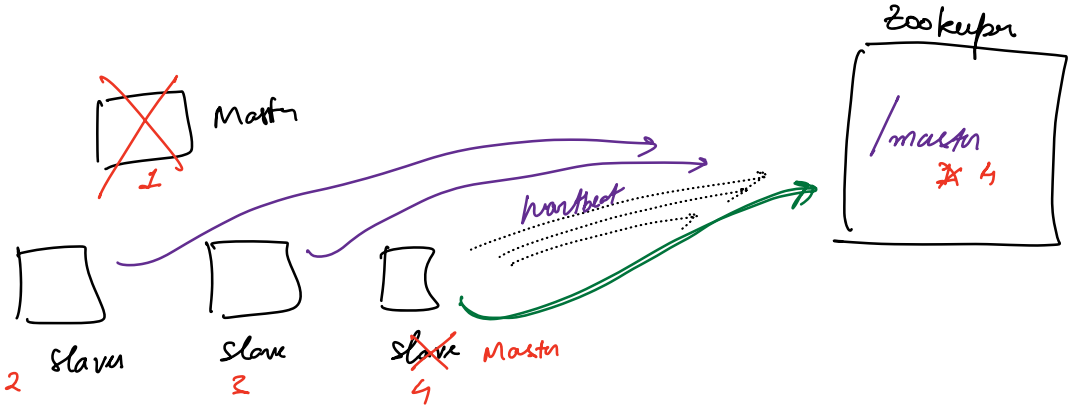
- Persistent → write data to it → persists until explicitly deleted
- Ephemeral → data written → Owner

  this data is only persists as long as the owner is alive/available



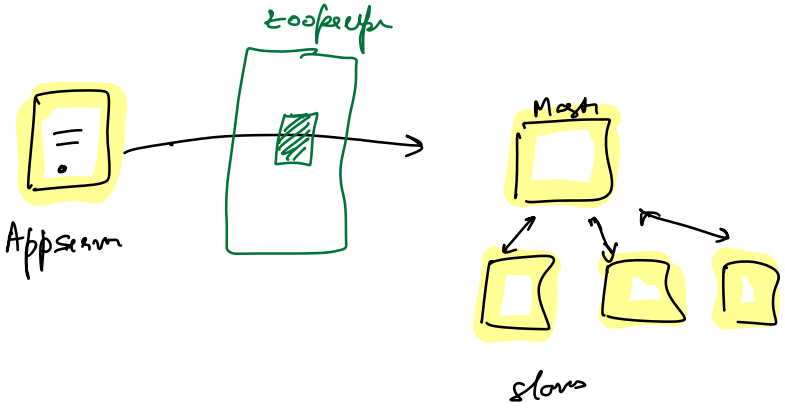
In a master-slave → whoever is the master
↳ can't to /master inside Zookeeper → store their ip in the ephemeral node

Master Election

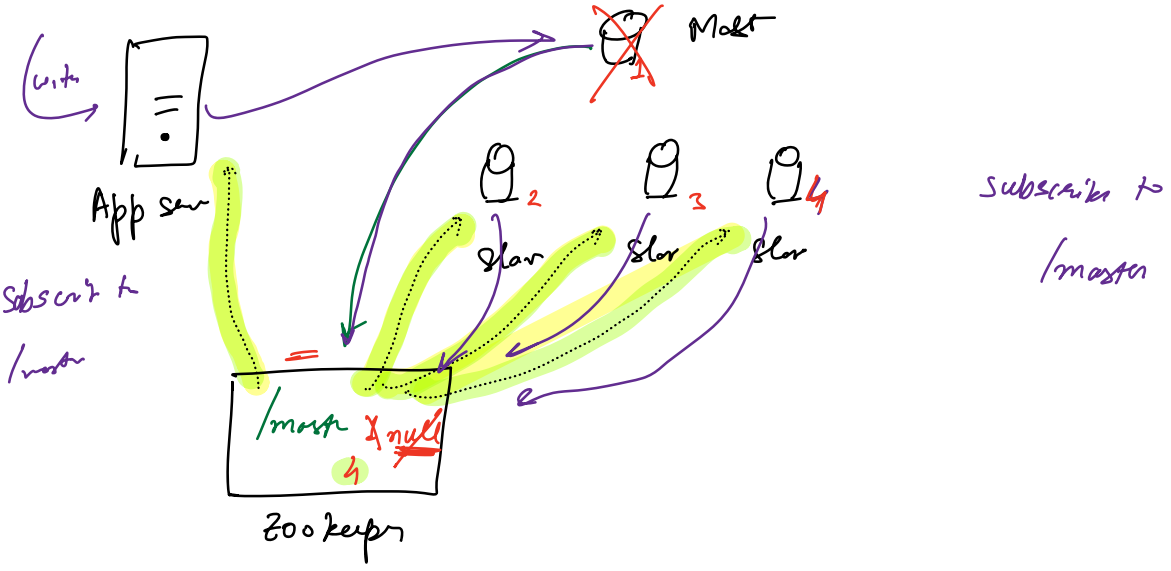


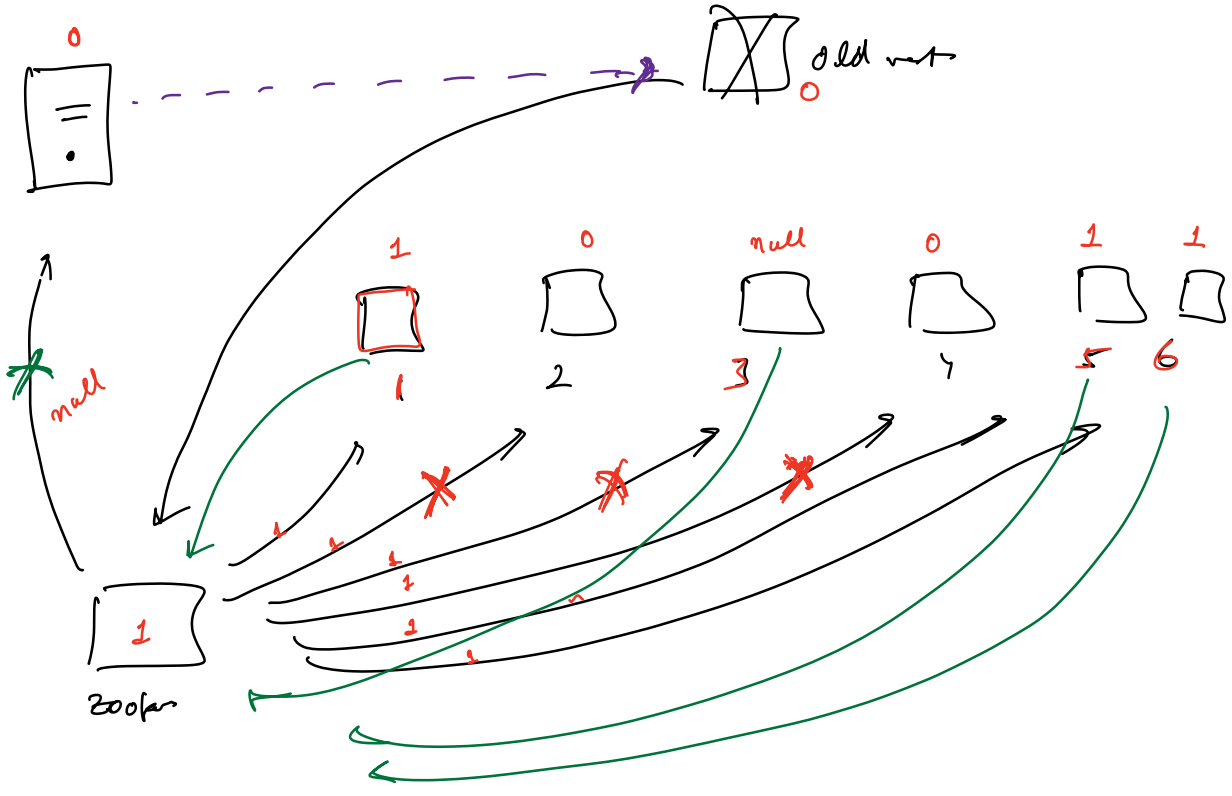
For re-election every slave will try to create its own ip in zookeeper's /master node.

Inside zookeeper → lock is acquired & only 1 create is allowed to pass through



Set a watch / subscribe to any node





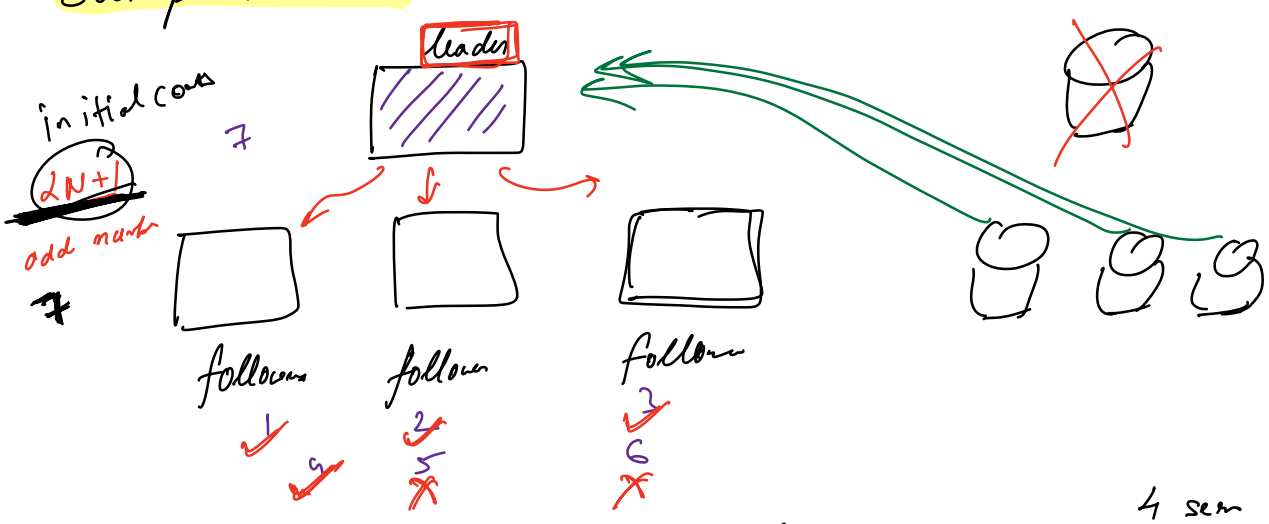
Zookeeper must notify all subscribes → ~~1 way~~ channel.

2 way connection.

the moment any app slave detects that it is no longer connected to Zookeeper or if it gets a notification from Zookeeper saying that master is null → stop serving write requests.

Extra hop ⟶ (solved) Subscriptions to ZK nodes
↳ 2 way connection

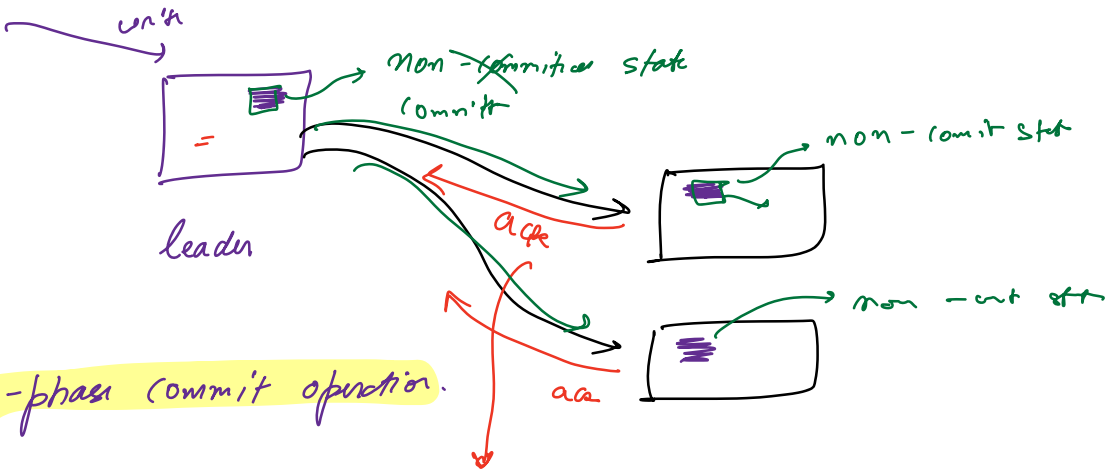
Zookeeper Architecture



When a write req comes to ZK's leader,
→ it writes to self
→ it also writes to a majority of slaves.

read → from multiple nodes.
if we establish a quorum.
collection
consensus

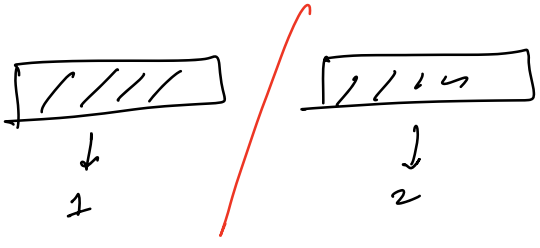
4 servers
N+1
machines out of (2N+1)
total have been written to.
more than 50%



2-phase commit operation.

guarantee that if in future you tell me to commit data → I will be able to do so.

2N nodes



Split Brain

2N+1 nodes

2N+1 machine initially → read/write will succeed only if it is successful at atleast (N+1) machines

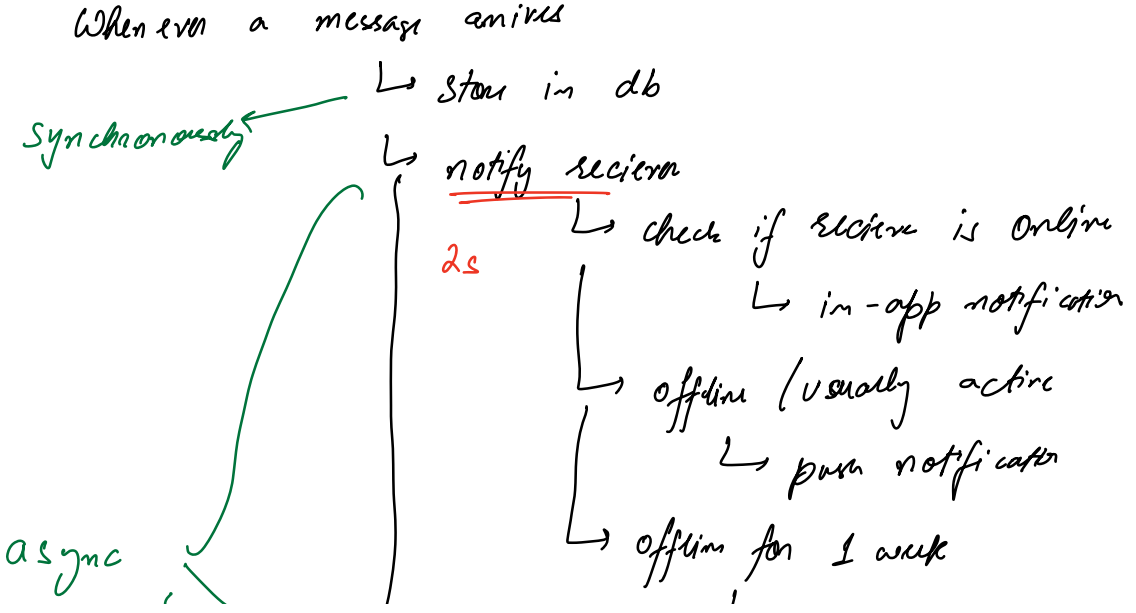
7 4

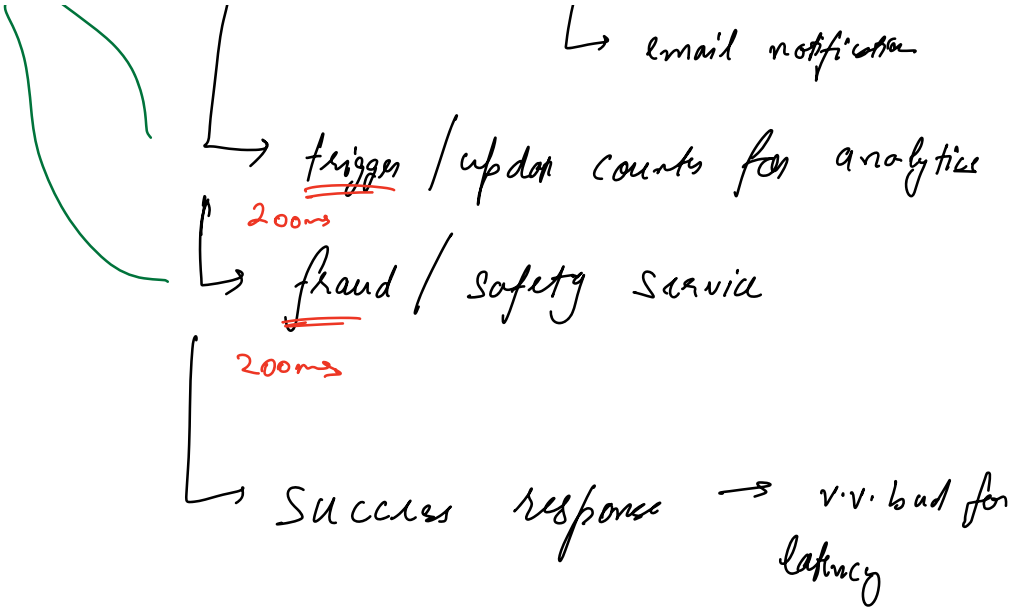
CAP / AP CP

ZK's leader is down
↳ leader re-election is happening.
↳ any write to ZK are forbidden.
↳ db writes/reads → still happening.
↳ any appserver/db can still read from ZK

10:30 → 10:40

Problem - 2 Async Tasks / Low Latency





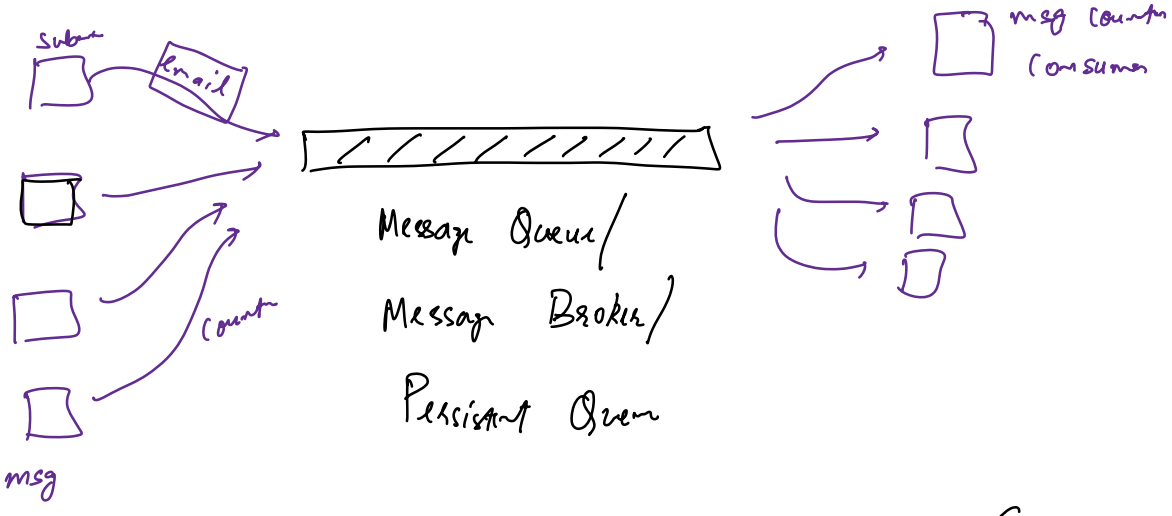
(1) we want to keep latency low
↳ return success as soon as sync tasks are done

(2) async tasks are still important.
↳ even though we've prematurely returned success
↳ these tasks are actually performed without failure

Persistent Queues
↳ any data stored is durable → stored on disk / replicated

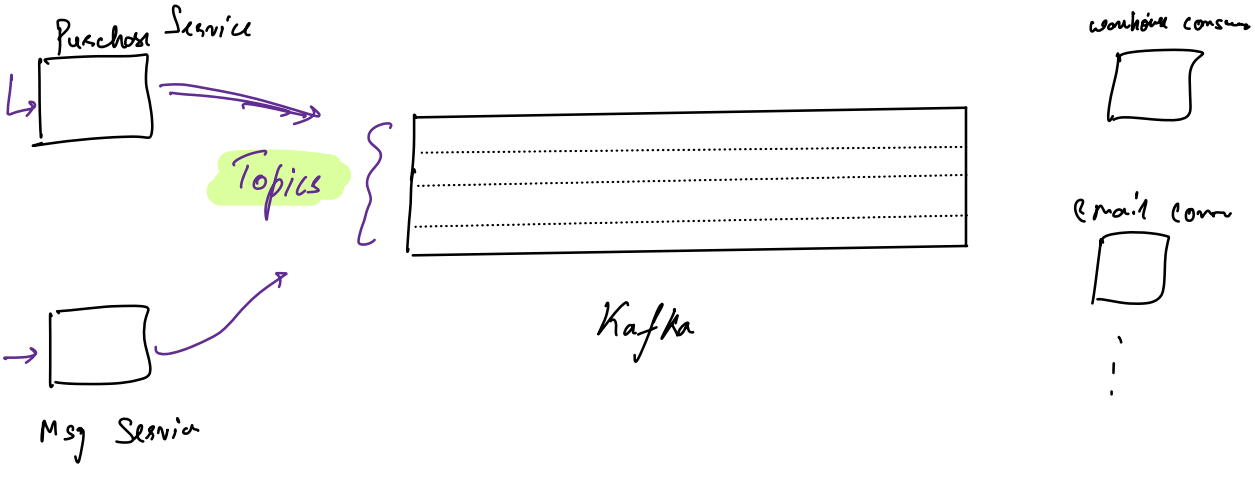
↳ shock absorber

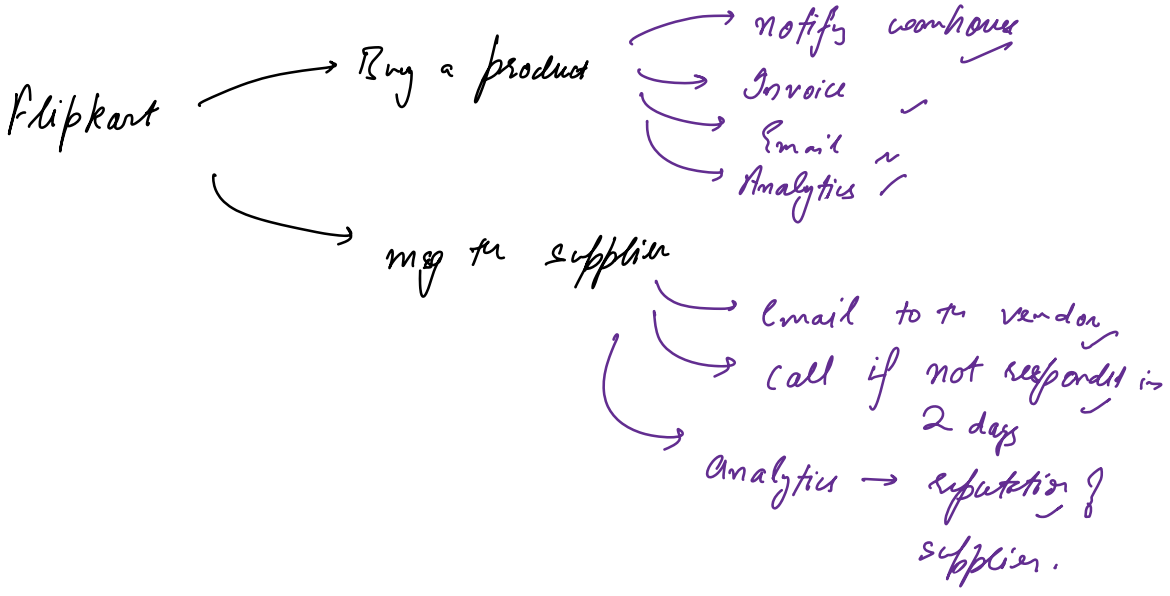
Publisher / subscriber (Pub - Sub)



Producers

Kafka → internally uses Zookeeper.

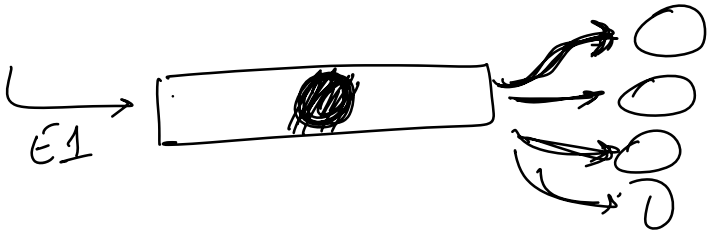




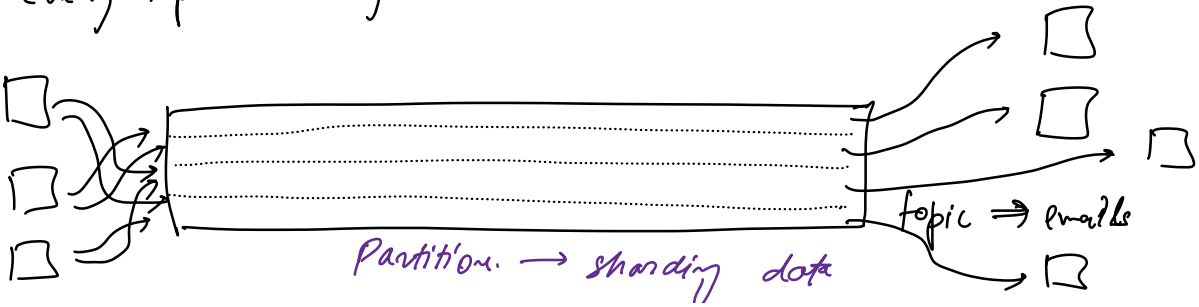
Producer can produce at different rates

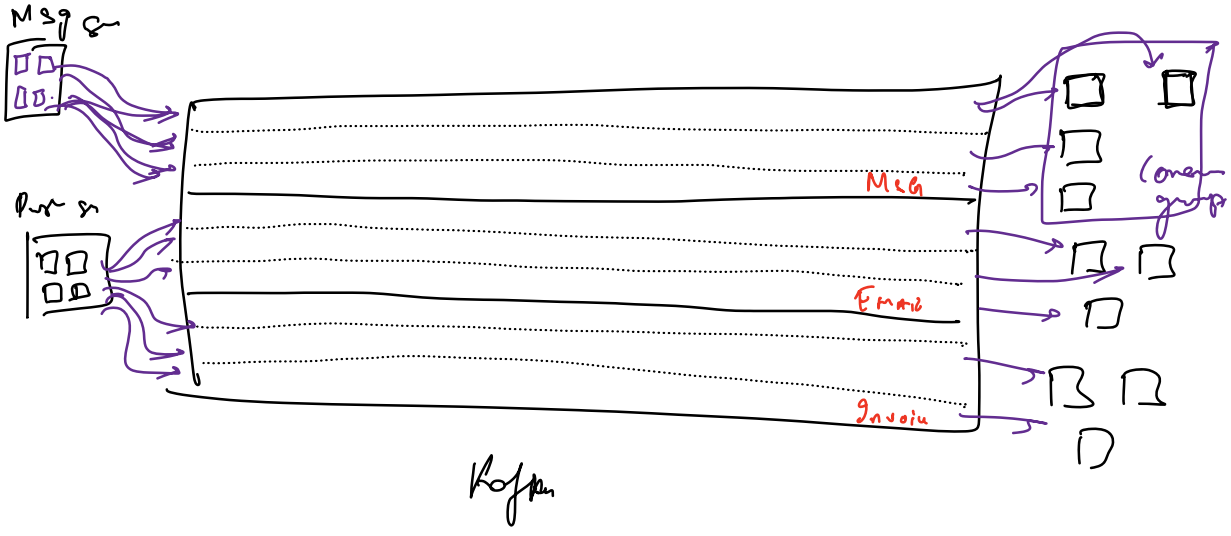
Consumer can consume at different rates.

any data produced is persisted for 1 week (retention period)



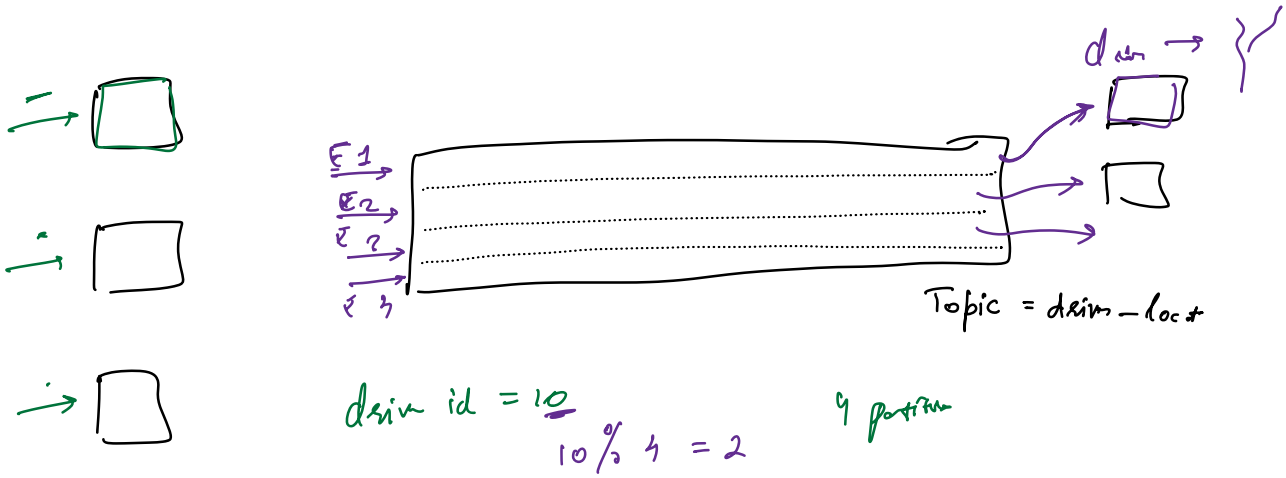
Every topic is further subdivided into partitions





partition in a topic ≤ # consumer for the topic

Uber



driver id = 10

10 % 4 = 2

4 partition

event ⟶ partition ⇒ Round-robin fashion.
insert

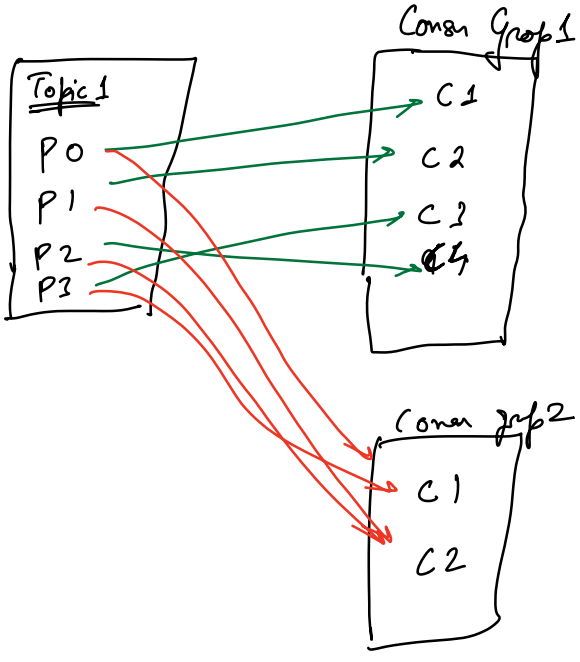
allows you to supply a key

(hash(key) % # partition)

If topic is huge

→ 1 Consumer → forever to consume all even

Consumer Groups



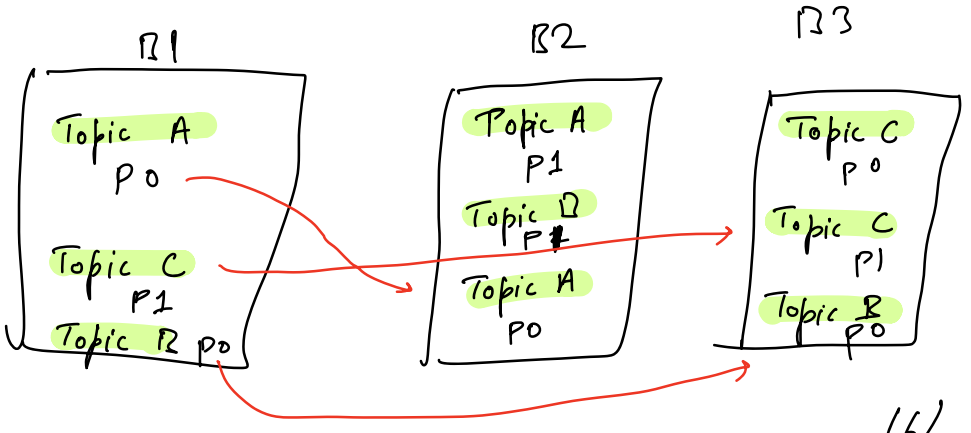
Consumer → Topic, group.

What happens if 1 machine (broker) dies → prevent data loss?

replication.

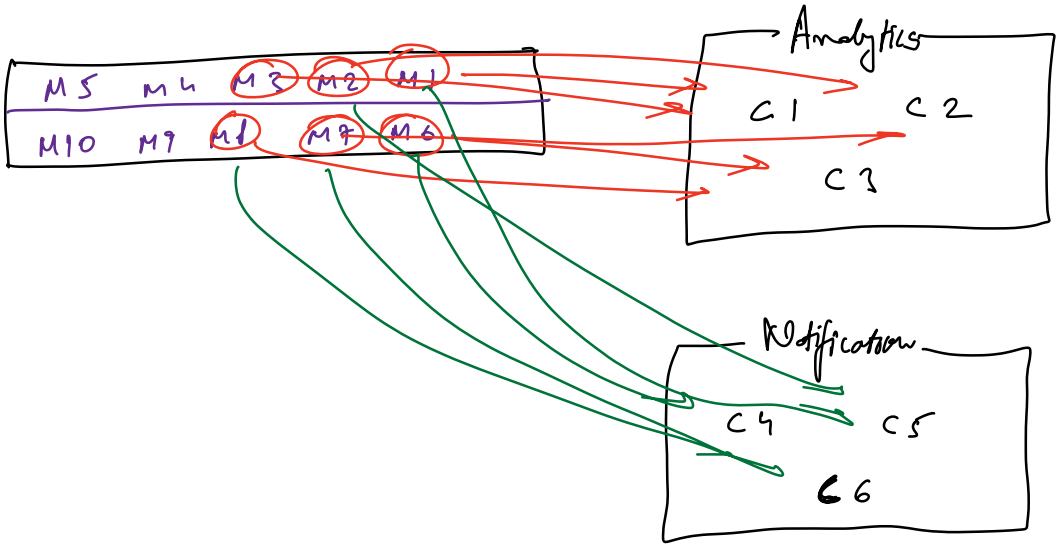
4 topics → 2 partitions each → application factor is 2

3 brokers (machines)



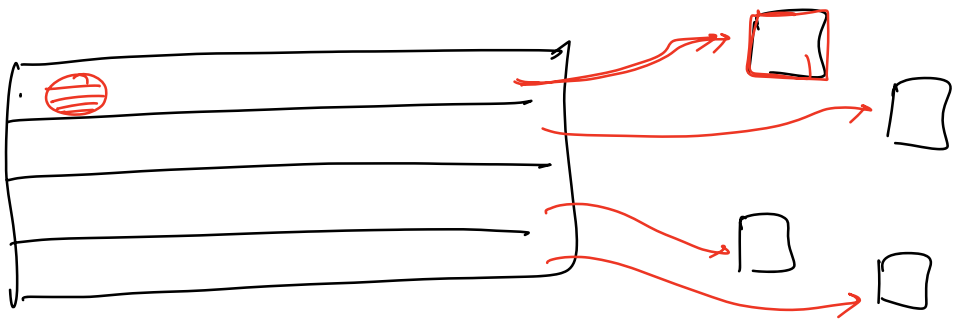
16/3

= 5 1/3



Offset is added to each msg

↳ incremental



N topics, each topic has P partitions, replication factor of X

R brokers

N * P * X → R brokers

N P X / R ideal

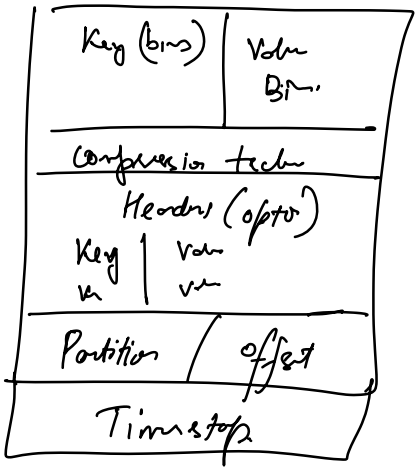
1000 events / sec

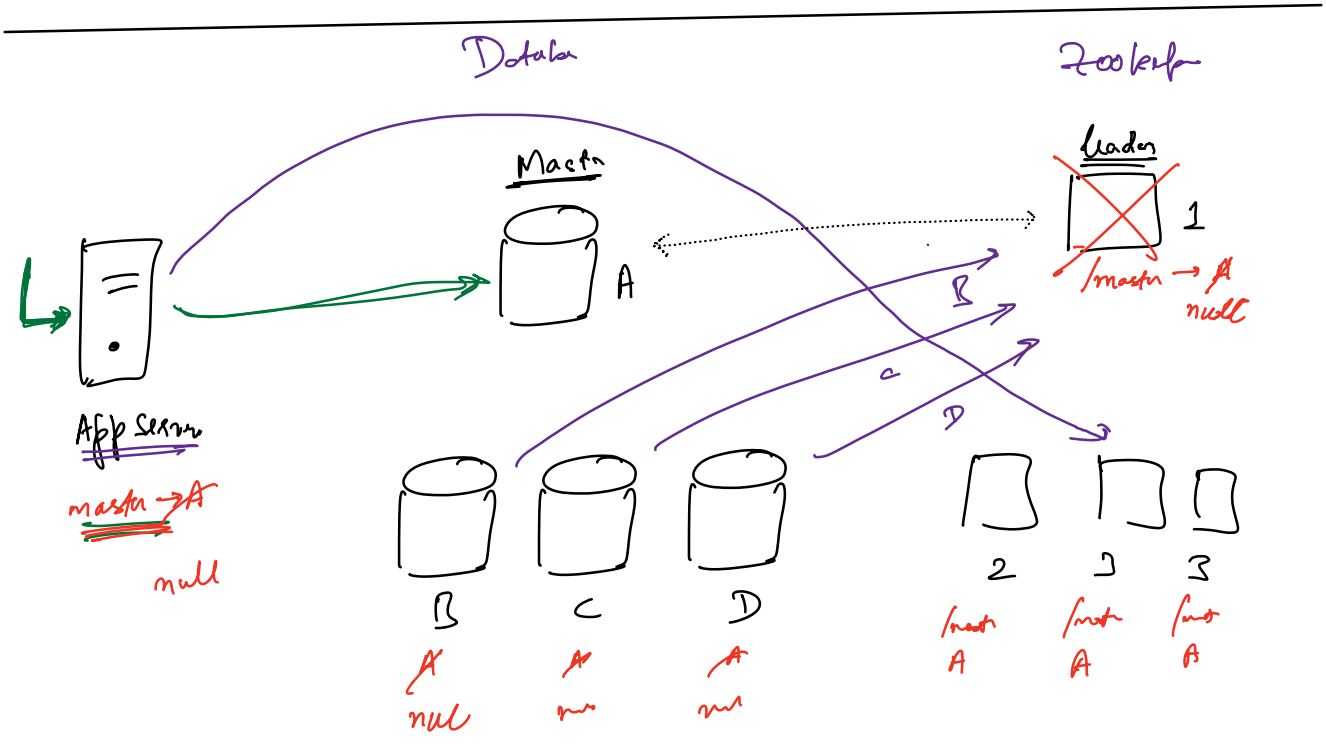
each consumer takes 100 ms to process an event

↳ 10 events / sec

$\frac{1000}{10} = \underline{100}$ consumers

125 consumer





Once a month → master goes down → unavailability for 5 secs

$$\left(1 - \frac{5}{30 * 86400}\right) * 100\ \% \approx 99.999807\ \%$$

SQL databases → not sharded

ACID guarantees