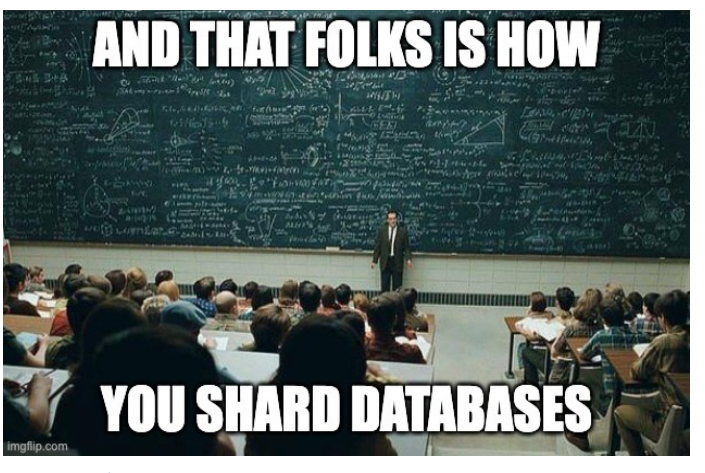
HLD-6: SQL vs NO-SQL & Sharding

Saturday, 15 January 2022 8:21 PM

Welcome to the class :-)
We will start at 9:05 PM

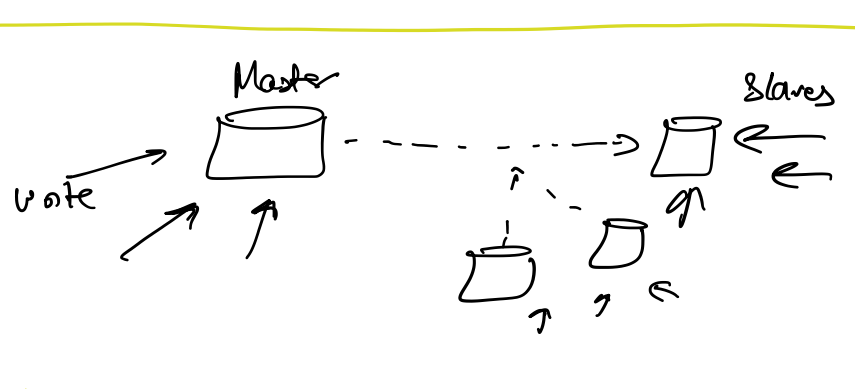

AND THAT FOLKS IS HOW
YOU SHARD DATABASES
imgflip.com

Last Class Learnings:-

1. Replication ✓
2. Master Slave Architecture ✓
3. Degree of Consistency ✓
4. Gossip Protocol ✓

Agenda:-

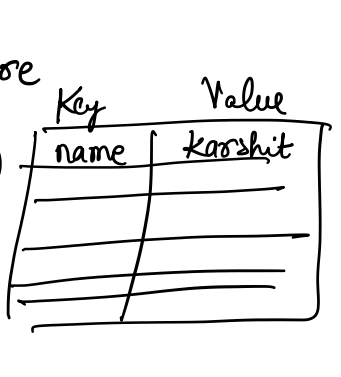
1. Introduction to Redis (101) {Part of last lecture on Replication}
2. Recap on ACID txns
3. Data Sharding
4. Develop intuition on how to shard data
5. Google BIG TABLE Case study


Master
Slaves
write

Redis :-

1. Cache System
2. Key → Value Store
3. SET("name", "karshit")

GET


Key
Value
name
karshit

(-1)
↳ indefinite

(TTL) → Cache

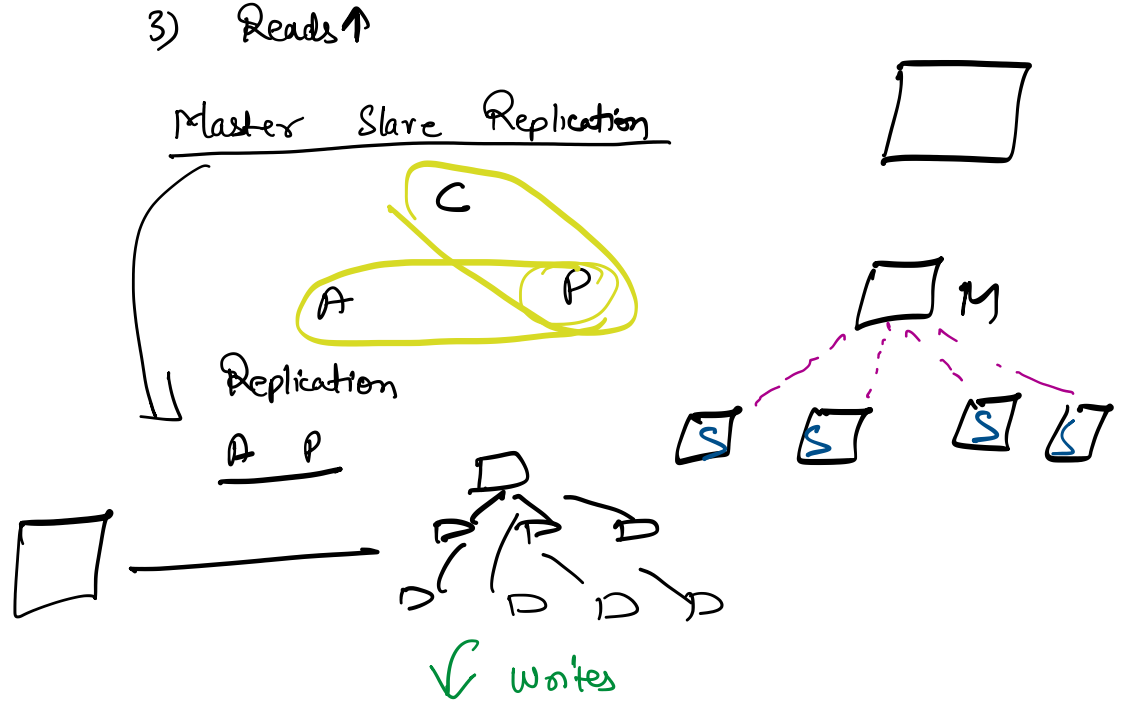
set (key, value) {-1} → indefinite
delete (key)
set (key, value, ts) → ttl

get('name') : karshit

AWS → {ElastiCache}   Hotcopy

1) 1 machine is not sufficient
2) Write ↓
3) Reads ↑

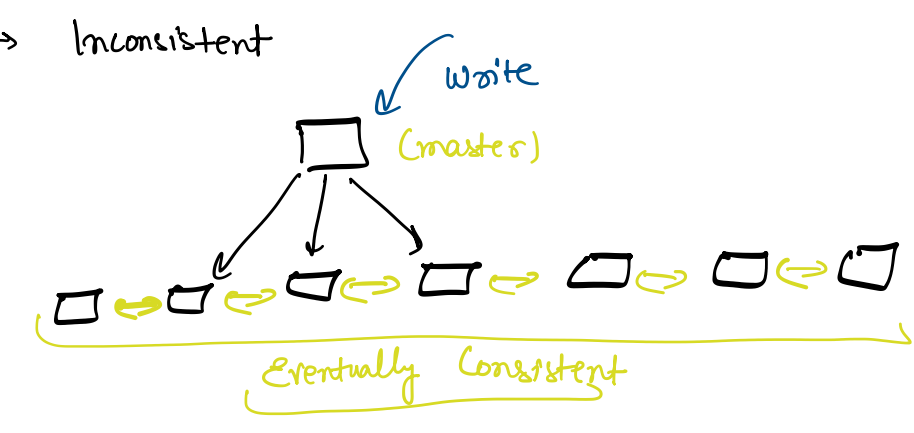
Master Slave Replication


C
A
P
M
S
S
S
S
Replication
A P
Writes
1
2
3
4
Read

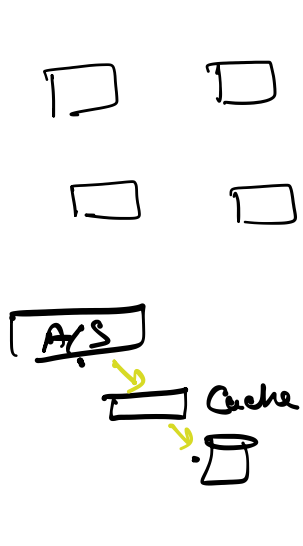
In presence of replication :-

Redis:-

→ Available for read & writes
→ Inconsistent


write
(master)
Eventually Consistent

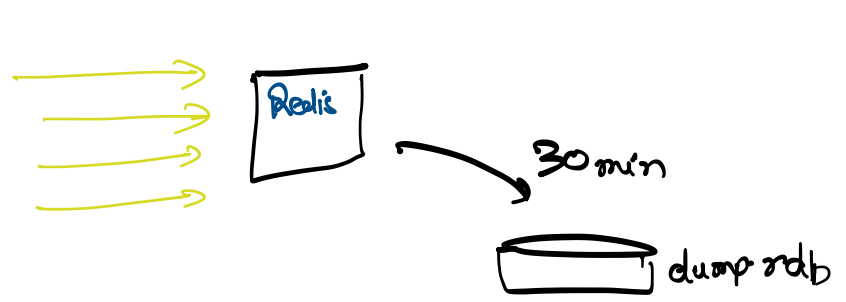
→ Gossip Protocol



→ Redis & Memcache

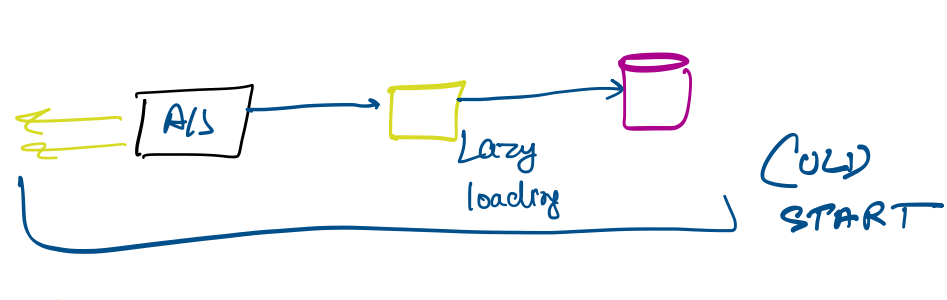
→ persistent → not persist

RDB: Redis Databackup File
AOF: Ahead of File


Redis
30min
dump.rdb

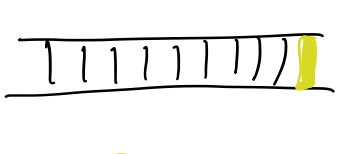
Pros:-
- → write performance
- → less load
- → Don't have cold start

Cons
- → Data loss


AFS
Lazy loading
COLD START

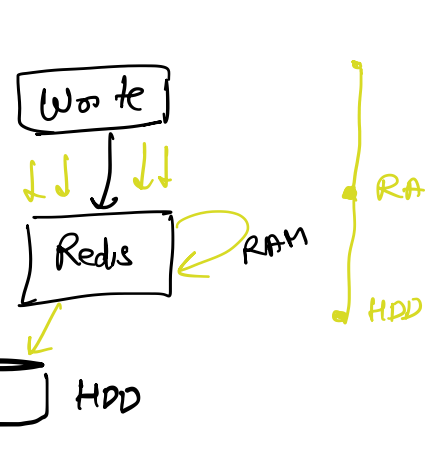
AOF:-

set(key, value)



Cons:-
- Slow writes
- High load

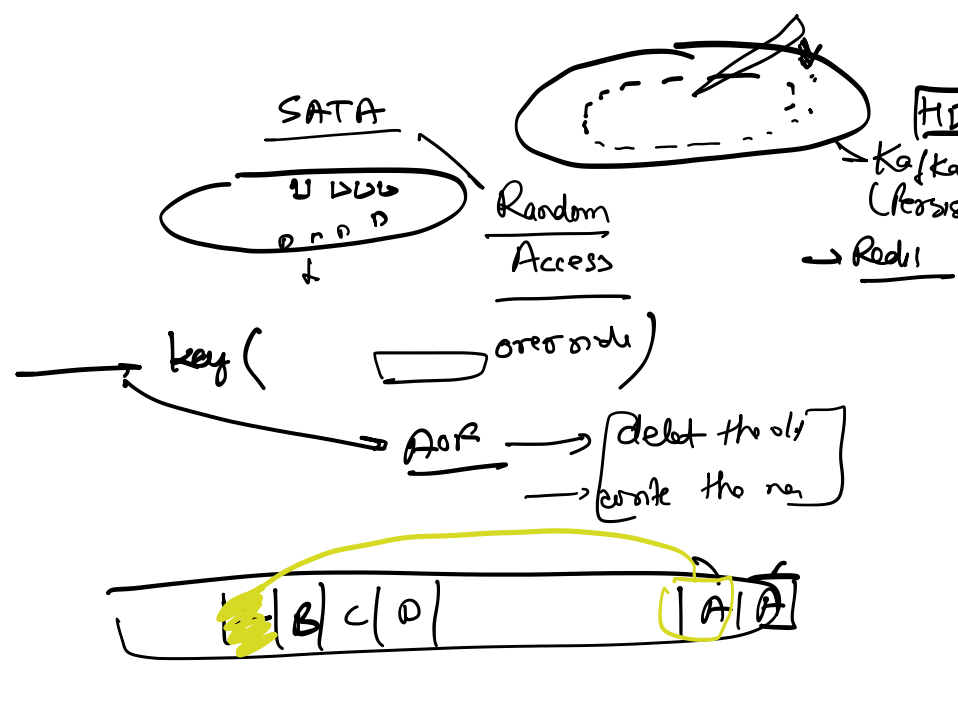
Pros
- data persistence


Write
Redis
RAM
RAM
HDD
HDD

userId: popular restaurants: {csv}

Karshit → 999

"user: 999: popular restaurants" → {1, 2, 3, 5, 5}


SATA
Random Access
HDD
Kafka
(Persistent Que)
Redis
key
AOF
delete the old
write the new
B
C
D
A
A

Redis (key/value) → RAM
↳ AOF
↳ Used for persistence

Redis
- → K/V
- → Replication
- → Master-Slave (AP) {CAP}
- → Persistence {RDB / AOF}

SQL v/s NoSQL

SQL:-
1. MySQL
2. Postgre SQL

[RDBMS]

No-SQL
- HBase
- MongoDB
- Cassandra

SQL Benefits

- → Indexing
- → Joins
- → ACID in txns
- → Maintainability / Maturity / Stability

1. 1 year
2. 35 Million users
3. < 10GB

→ not → text files → images → videos

Application data → QUORA

1GB	100MB	1TB

{Google / FB}

MySQL / PostgreSQL (SQL Database)

→ Transactions

A C I D – Atomicity
↳ All or nothing

Data Estimations

From Scratch
- → Pace of development
- → Iterative
- → Framework support
- ↳ SQL Databases

Already established
- → User Base
- → DAU / MAU
- → growth in next 5years
- >1TB
- → NoSQL / SQL

Banking A: Atomicity

Karshit Rs 100 → 10Rs → Vishal Rs 1000

1. (100-10) 90 → 90
2. 1000 + 10

Commit 1010

CytusDb
↳ Relation
MemSQL
NewSQL

C: Consistency

100 ← A 100
90 → 90 B

I: Isolation

Karshit 100 — 10Rs, 10Rs

Vishal 1000 — 1010

Work lock

Divya 2000 = 2010

100 – 10
↳ 90

90 – 10
80

100 – 10 = 90
100 – 10 = 90

80 — 90

D: Durability

Tx : 9:00PM
9:01PM → Restarted
9:05PM ← Persistent

{Once a tx is completed/committed, it stays forever}

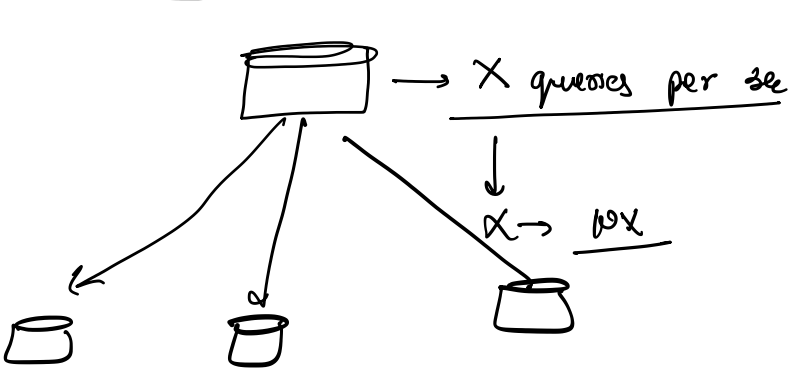
NoSQL
↳ ACID may not be guaranteed

My advice :- {Stick to SQL
Sticks to Basics}

#
- → less traffic
- → less data

→ SQL database make more sense

or both


X queries per sec
X → 10X

- → Indexing ✓
- → Caching ✓
- → Replication

Q: When do we switch to NoSQL?
→ when data does not fit in one machine

1. If my data is geo-located
   - ↳ Tinder
   - ↳ Uber
2. Schemaless
   - TShirt → Color, Fabric, Brand
   - Watch → Strap, Material
   - Shoes → formal / Sports

Atob Str

{JSON}

No Relation → Denormalized

MySQL

Books
↳ Shelf → Library

Users
↳ User Books

Normalization

userId	book	Library	Shelf

Denormalized

Facebook
↳ Harvard

1. Build Facebook for one college (MySQL)
   ↳ All the data can fit in one machine

Users: id | phone | email | ...

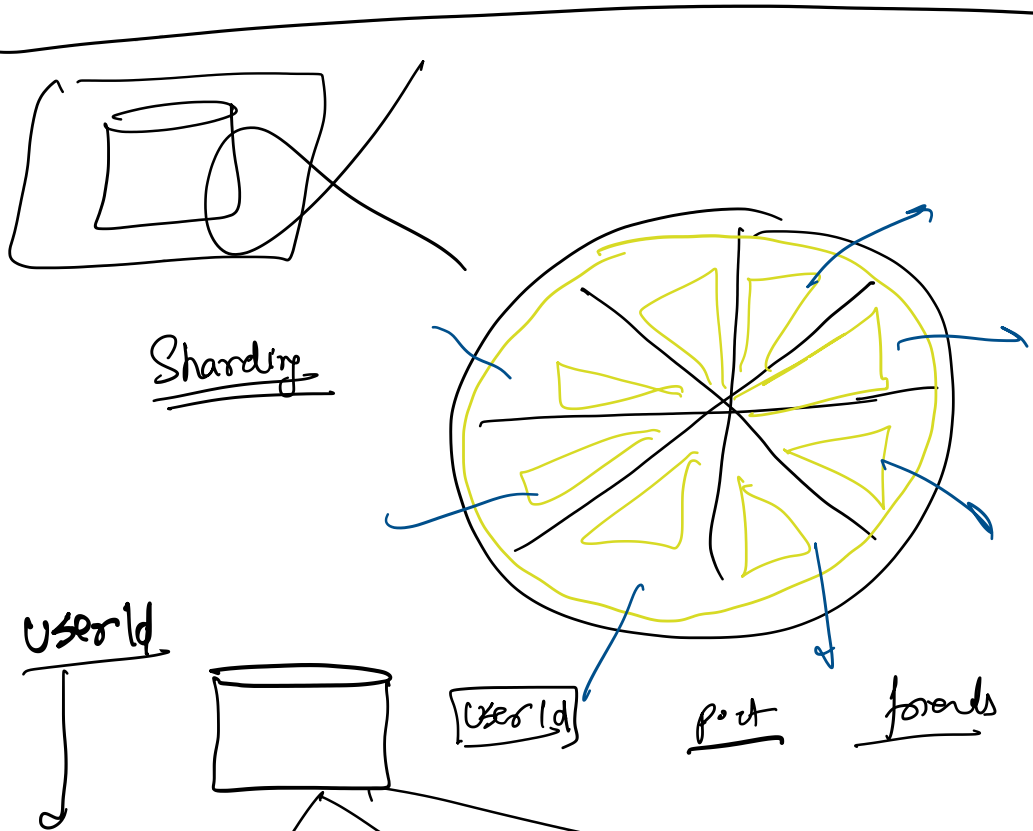
Post: id | userId | content | ... | ts

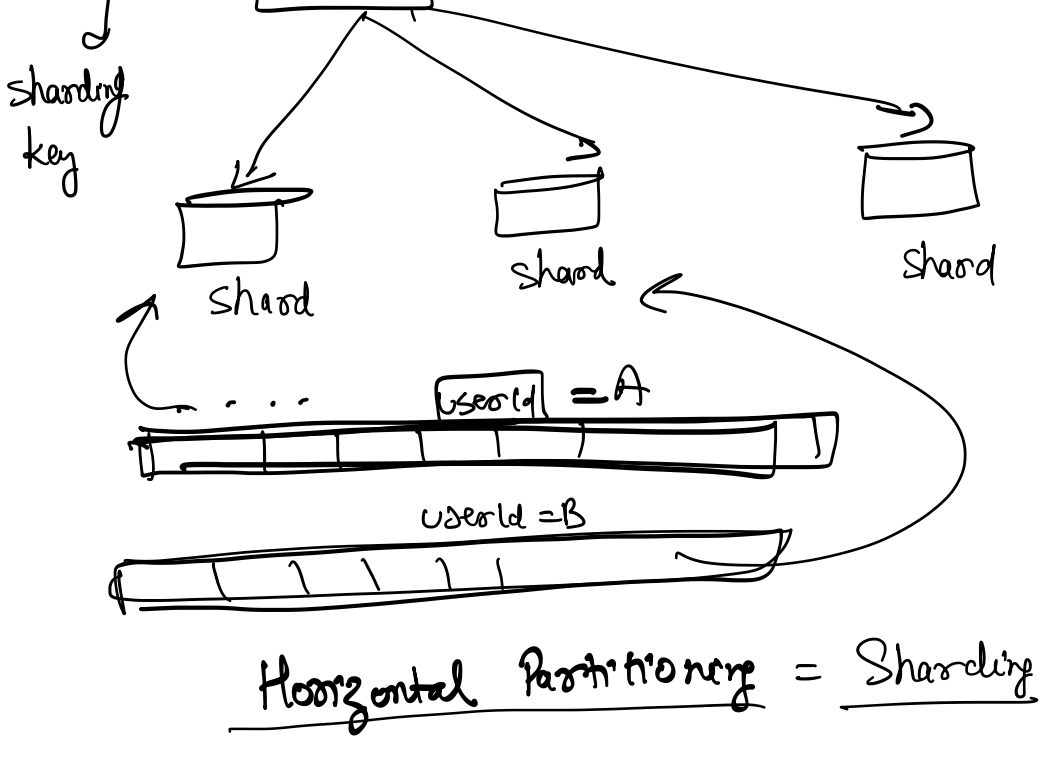
Friends: userId | friendId

1. Visit someone's profile : getPosts (userId)
2. Get UserFeed : getUserFeed (userId)
3. getTimeline : getTimeline (userId)

Q. SQL Query for userFeed :-

```
select * from posts where userId in
( select friendId from friends where userId = X );
```




userId
sharding key
userId
post
friends
shard
shard
shard
userId = A
userId = B

Horizontal Partitioning = Sharding

{sharding key}

object

user key : post [ ]
friends [ ]
profile [ ]
value

Vishal: { }

100GB

Anshum: { }

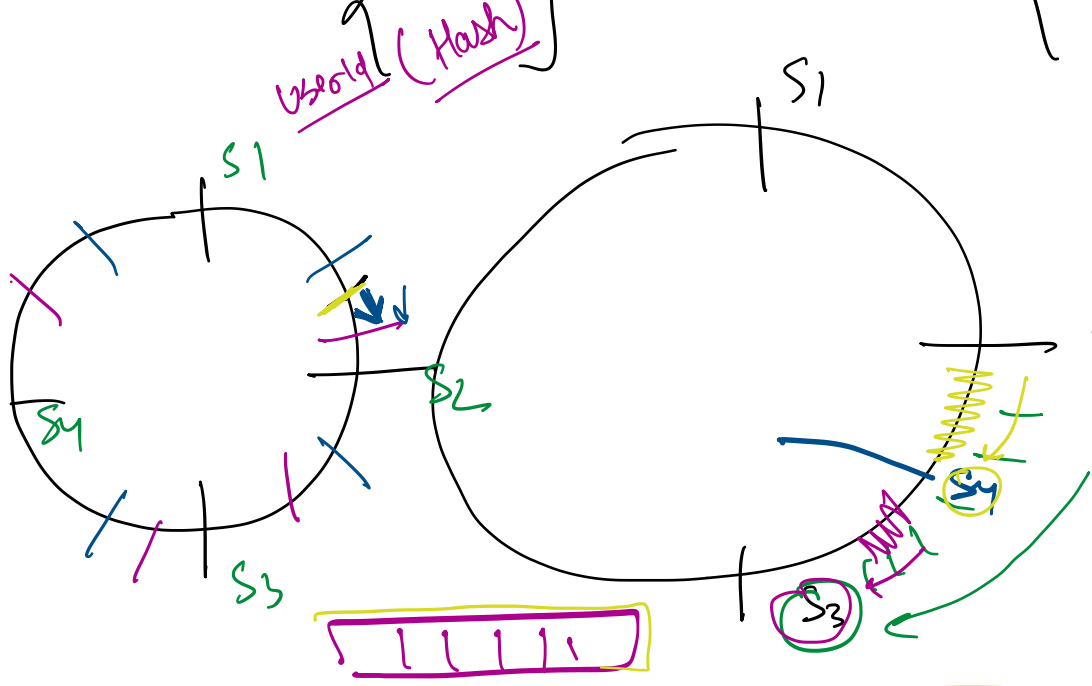
1TB

Aditi: { }

Bose: { }

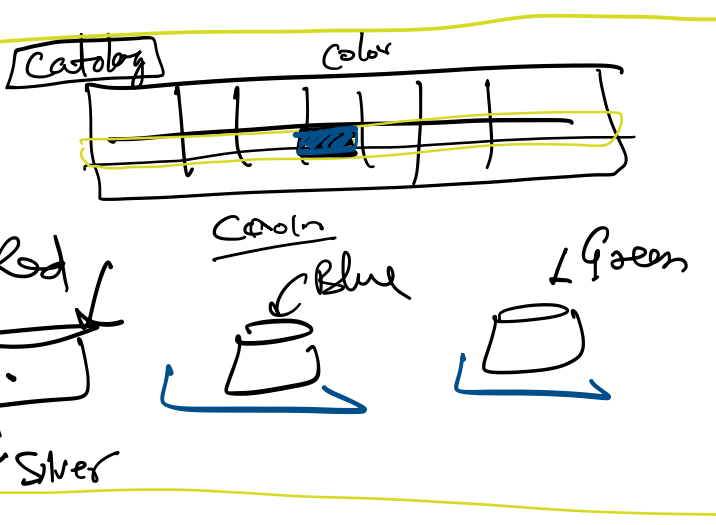
~100GB

userId (Hash)


S1
S4
S2
S3
S1
S2
S4
S3

Things to keep in mind while selecting a Shard key:-

1. Most frequent use cases (access patterns)
2. Every data is present on some machine

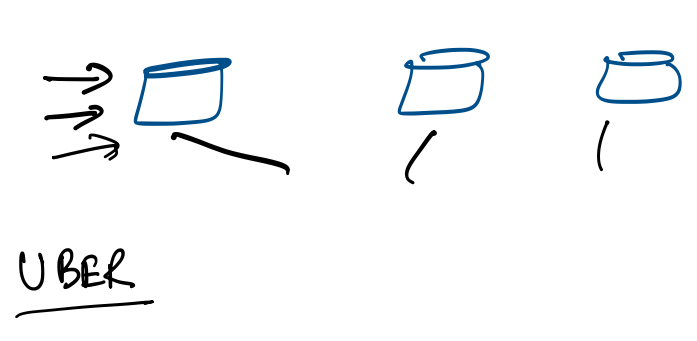

Catalog
Color
Red
Silver
Cream
Blue
Green

"color is not null"

How to select Sharding Keys

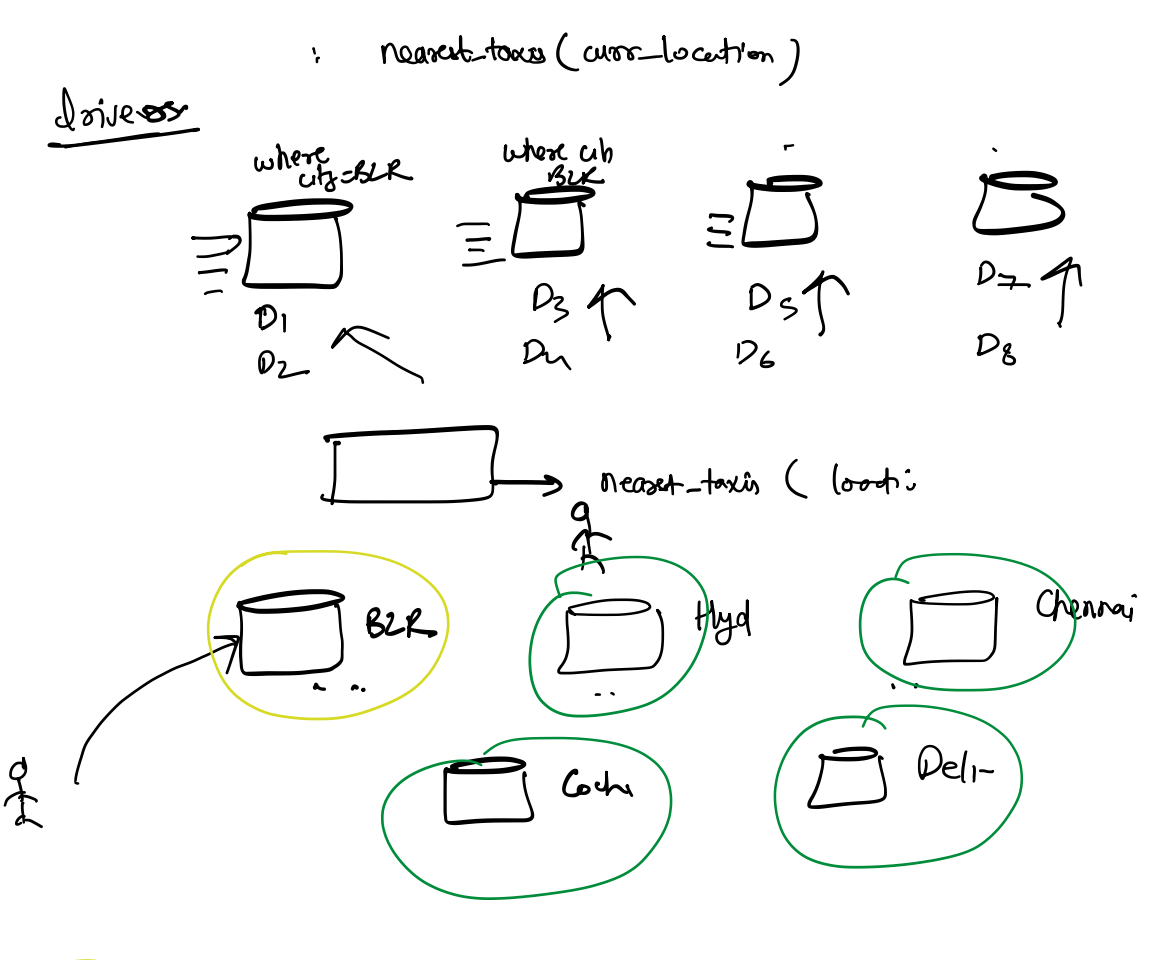
1. Banking System
   - a. Account Details
   - b. Transactions
   - c. Cards (Debit / Credit)

1. Data cannot fit in one machine
2. Users can move across cities
3. Use Cases:-
   - a) account_balance (accountId) / userId
   - b) debit (accountId, amount)
   - c) credit (" , ")
   - d) Statement (accountId, period)

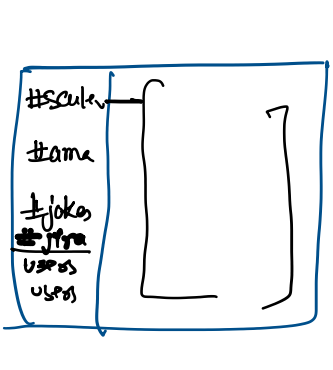

Aditi
acc
Karshit
Anshuma
Divya
Vishal

2. UBER
   - a) Users
   - b) Drivers (Partners)

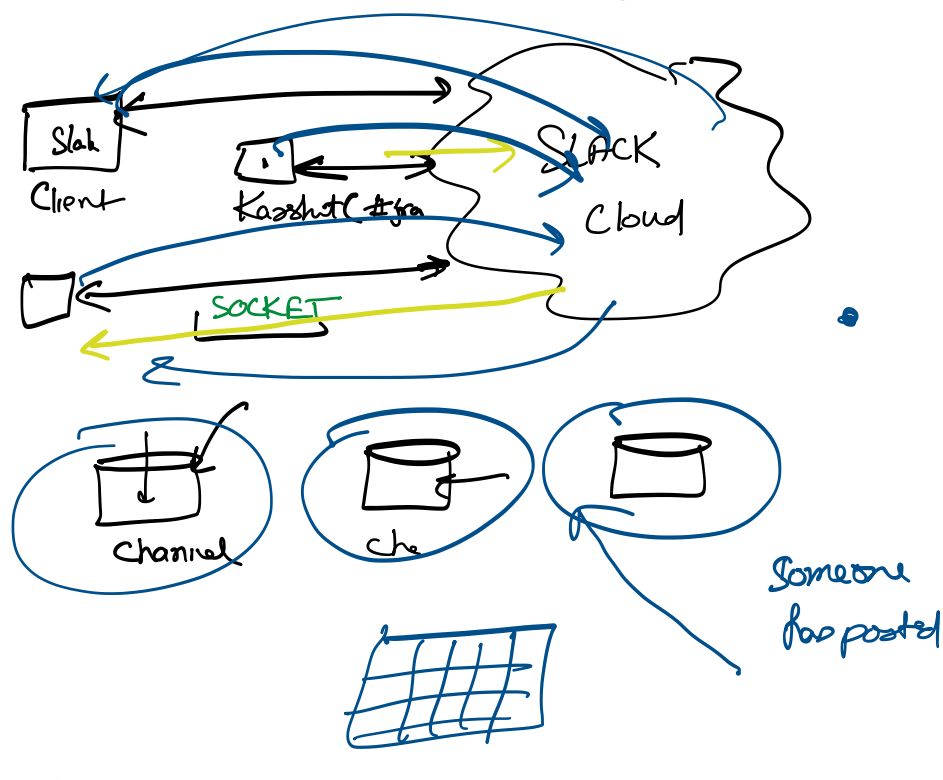
Usecase:
- Given a current location, give me nearest taxi
  - : nearest_taxis (curr_location)

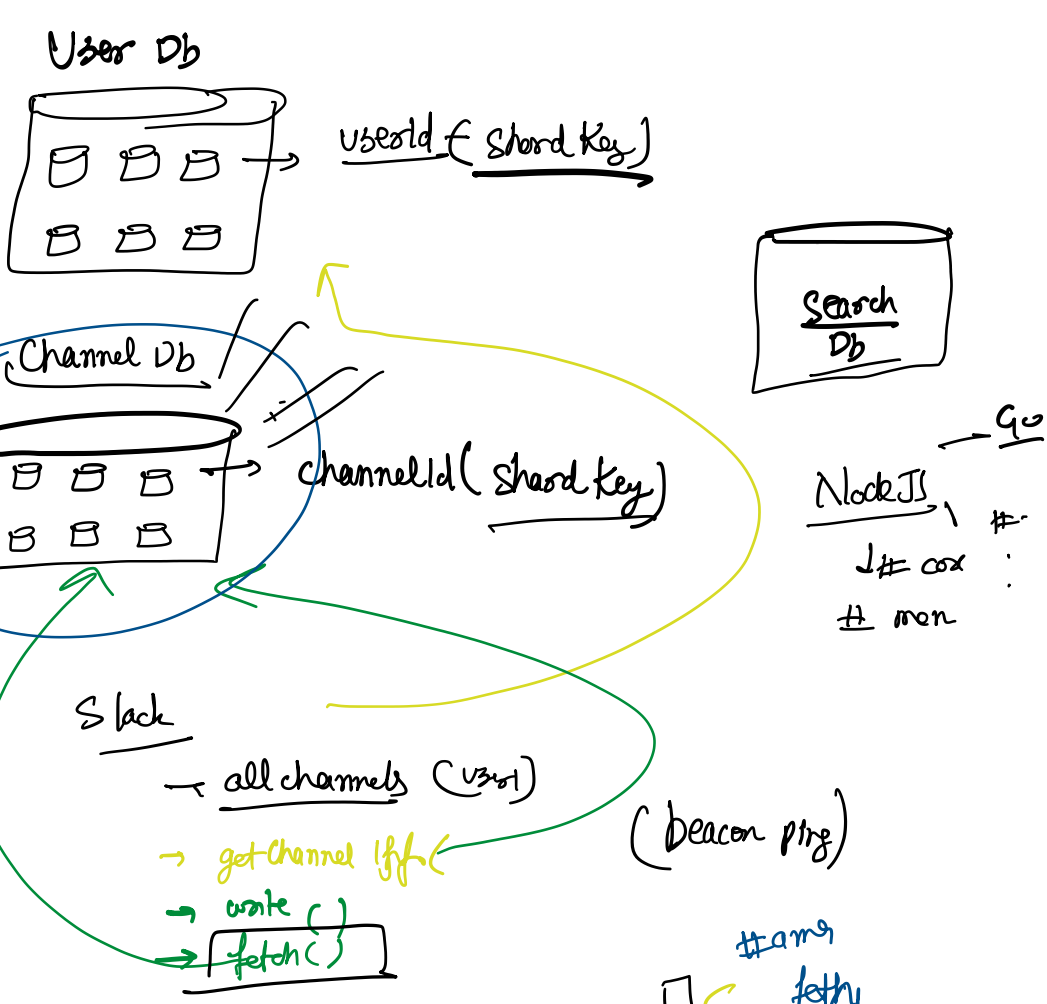

drivers
where city=BLR
where city
D1
D2
D3
D4
D5
D6
D7
D8
nearest_taxis (locati
BLR
Hyd
Chennai
Cochi
Delhi

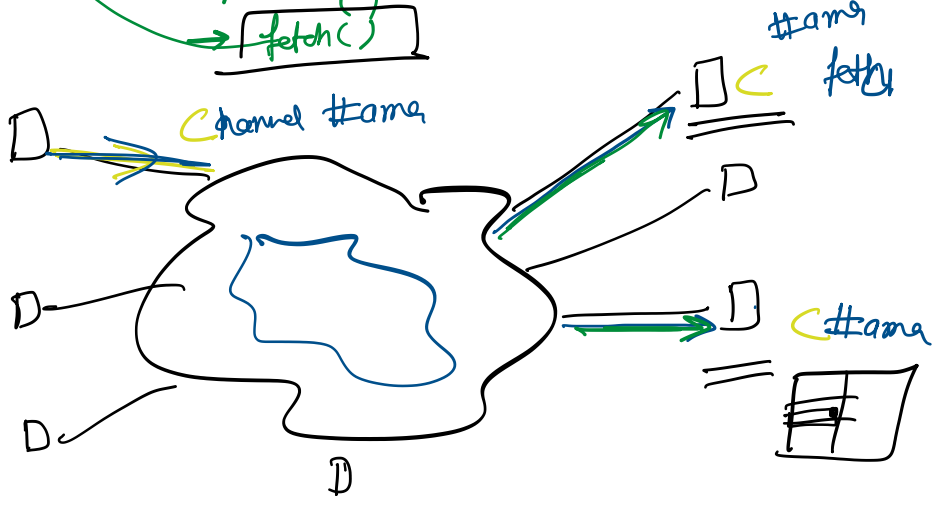
3. SLACK
   - a) Users
   - b) channels
   - c) Messages

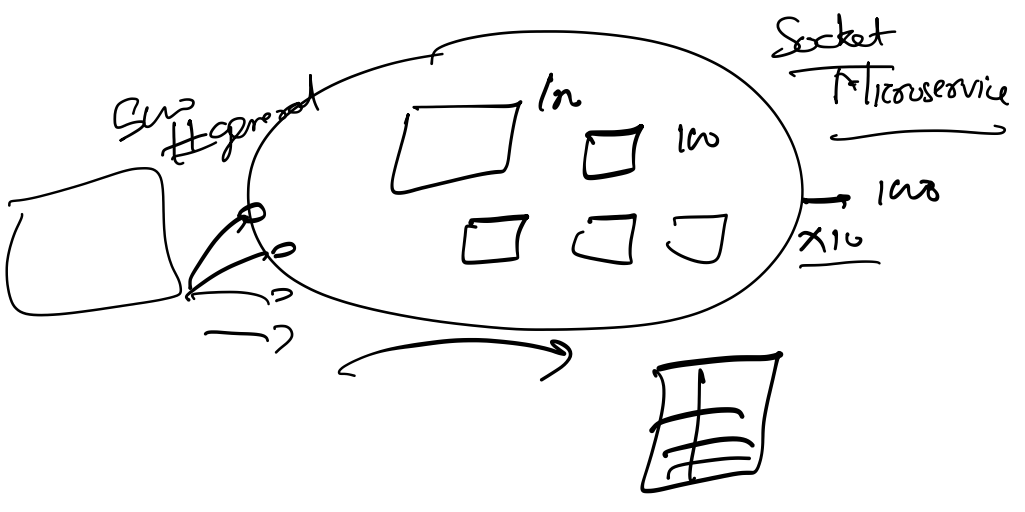

#scaler
#ama
#jokes
#fire
users
users

Use cases:-
- a) for a userId: give all channels (userId)
- b) for a channel: give all messages (channelId)
- c) Post message (channelId, userId)

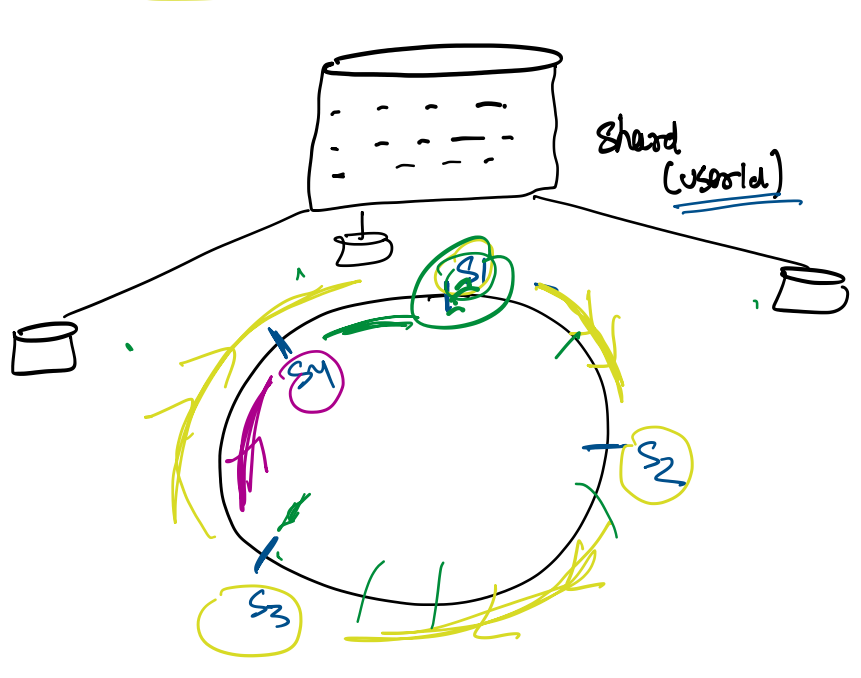

Slack Client
Karshit (#jo
SLACK Cloud
SOCKET
channel
ch
Someone has posted


User Db
userId (Shard Key)
Channel Db
channelId (Shard Key)
Search Db
NodeJS
Go
# cox
# mon
Slack
→ all channels (user)
→ getChannel (ffc
→ write ()
→ fetch ()
(beacon ping)


Channel #ama
#ama
fetch
#ama


Socket Microservice
Sw #general
1m
100
100
X10

1. Data overflows from a machine
- → Shard data across machines
- → Intelligently select a shard key


Shard (userId)
S1
S4
S2
S3

namespaces — key : # millions

user: id : - - - -

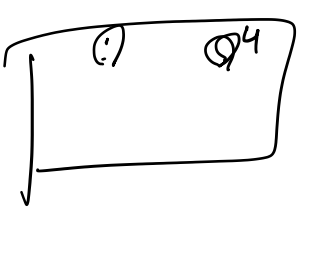
get("user: 1234: *")

get(user)
get(user_city)

User: 1234: city
User: 1234: likes
User: 1234: activity

UBER

del("user: 1234: *")

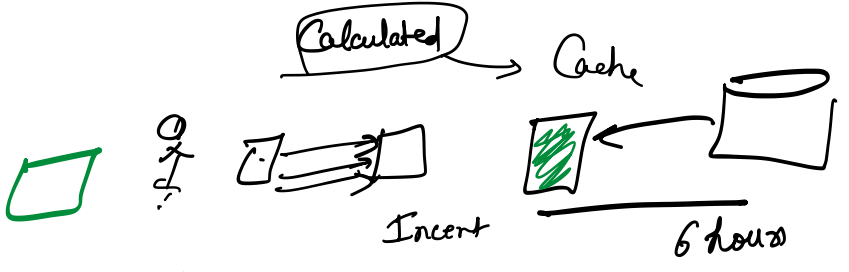


Driver Service

80% of usecases

Driver → Money have → Inc D

Calculated → Cache



Incent

6 hours

Millions → AS → DB

driver: driverId: incent: ddMMYY

Tradeoff

AWS
GCP
Azure

ELK