Uber

1. Case studies → 4 steps
2. Redis
3. Anshuman's Talk DB to use
4. Industry - Anshuman + Transactions

1. Features
2. Estimation of Scale
3. Design Trade Offs
4. Deepdive

---

1. Features

1. Request / book cabs
2. Drivers — get rides
3. Matching → Algo
4. ⋮

---

2. Estimation of scale

0.1 M drivers / big city

0.1 M × 100 = 10 M drivers / globe

20 M drivers

7B x 2% ⇒ 140M user base Uber

700M x 2 = 140M

(C) Design Trade Offs

(1) low latency – matching problems

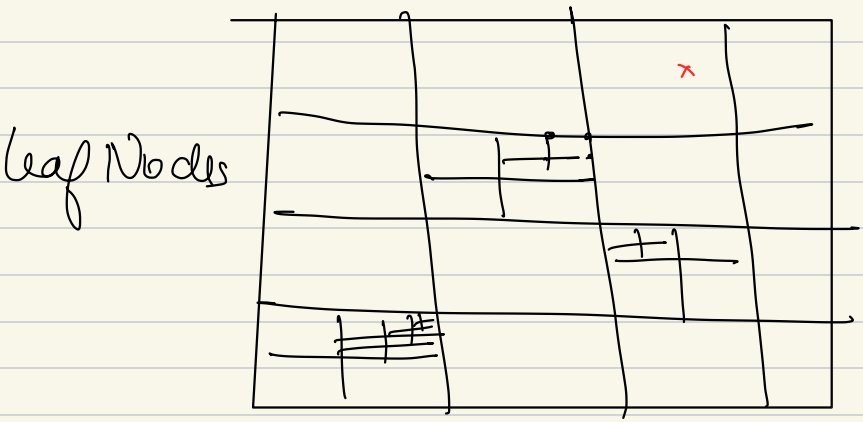
(2)

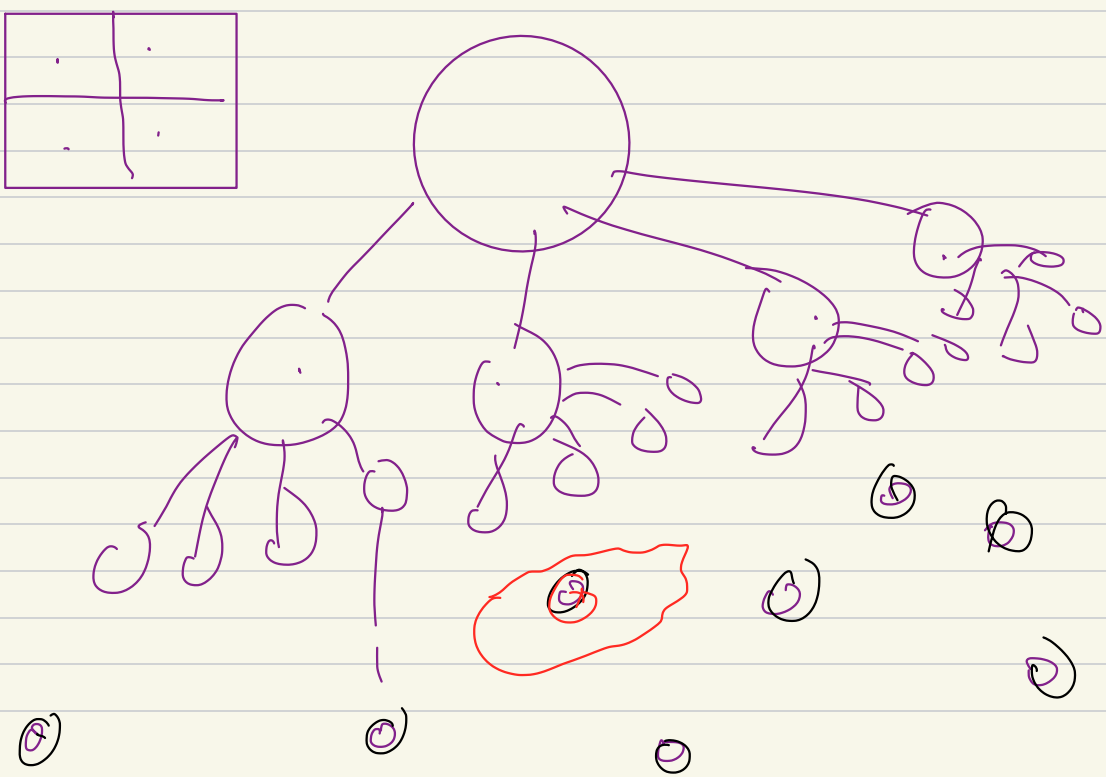
(D) Design Deepdive

Quad Trees

① Precompute step — build a quad Tree

Leaf Nodes



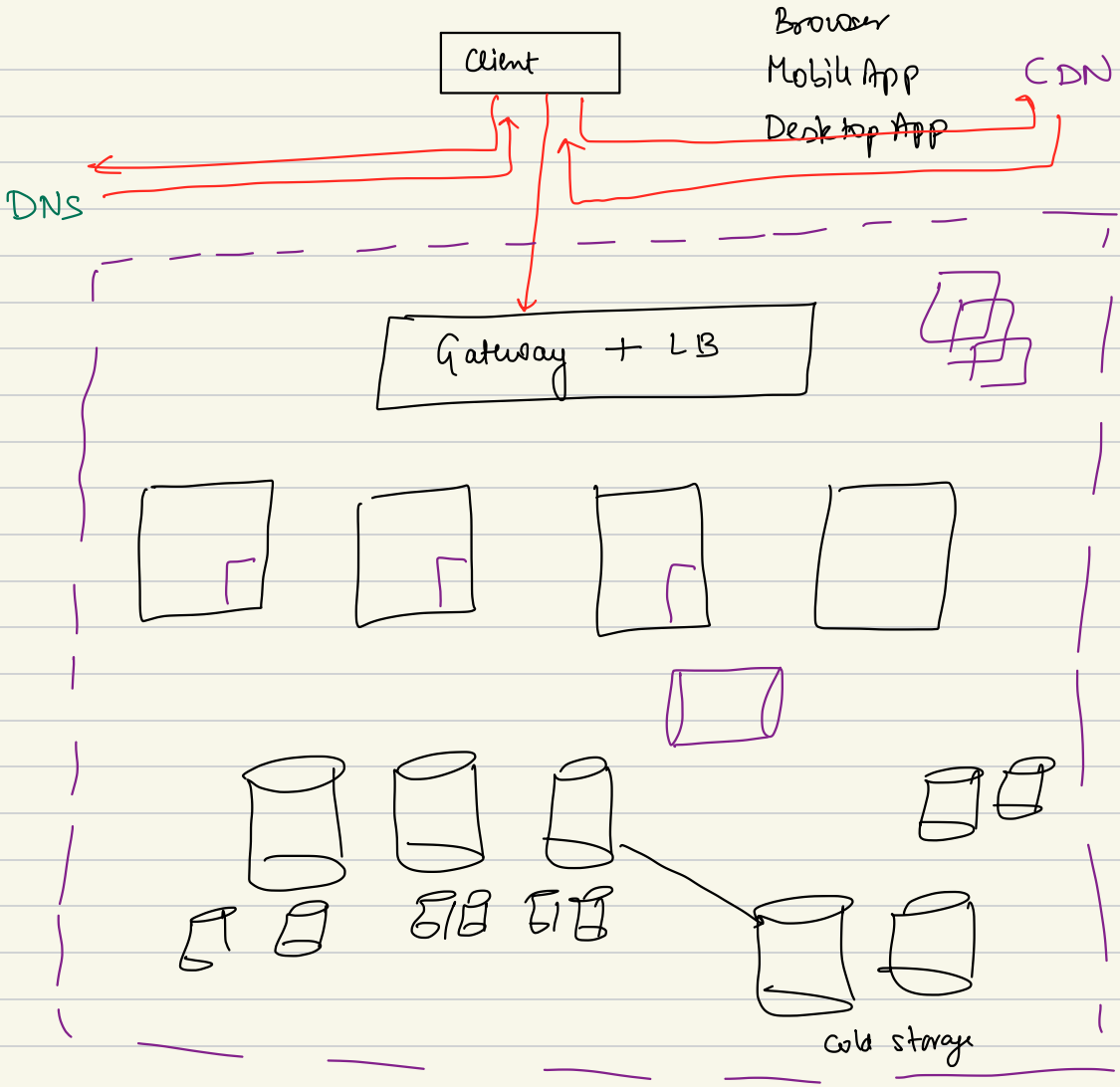


(2) query → Find Nearby Places

[piggyback on Quad Tree and answer]

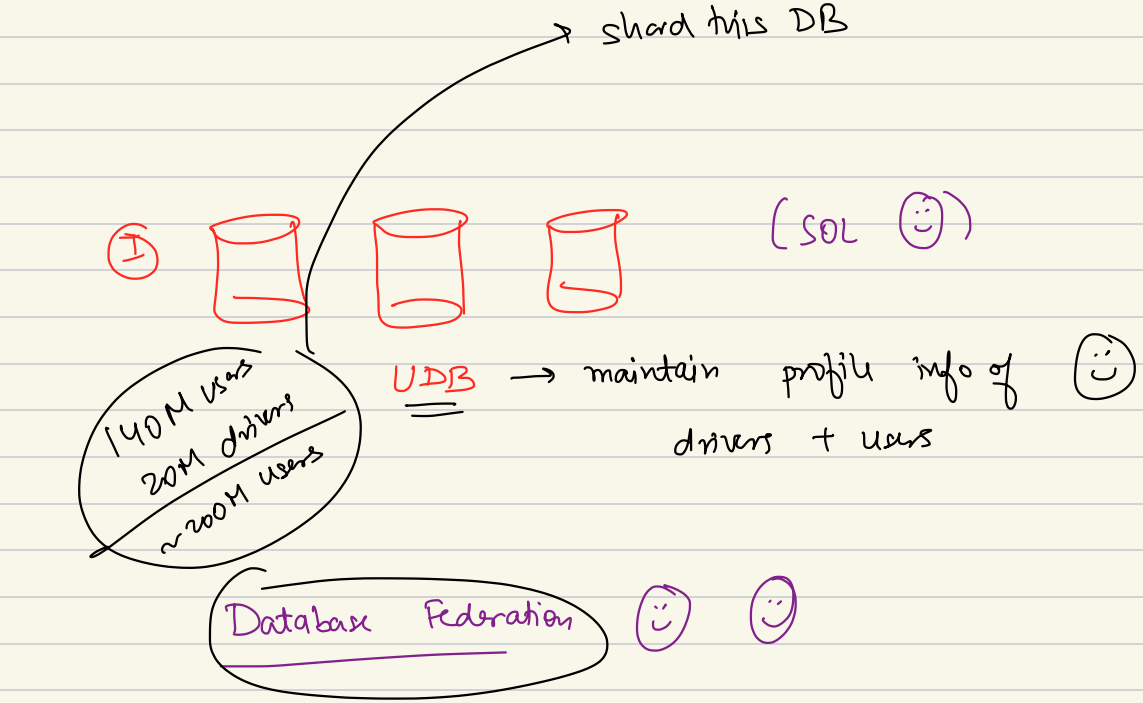
(3) places — added

— removed


Client
Browser
Mobile App
Desktop App
CDN
DNS
Gateway + LB
Cold storage

Uber

Federation of DB

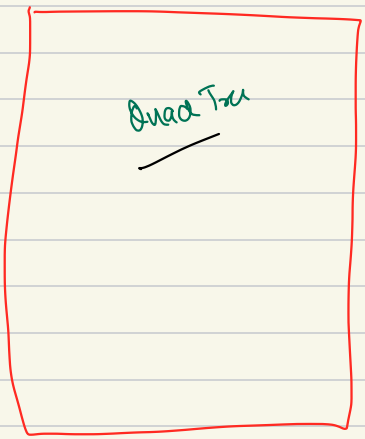


II Quad Tree → 100M Places of Interest → 6.5GB of storage

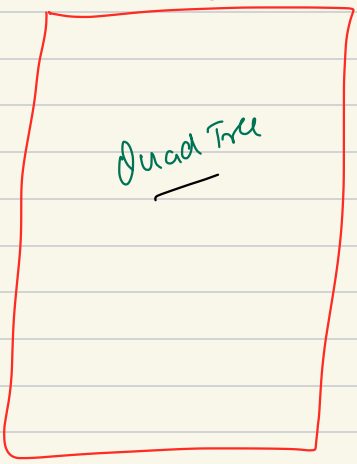
fitting in a single machine

normal commodity hardware → 4 TB storage
64GB of RAM

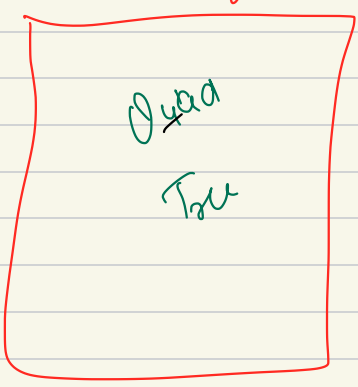
Uber Region – Delhi NCR



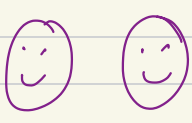
Uber Region – Jaipur + Neemrana + Alwar

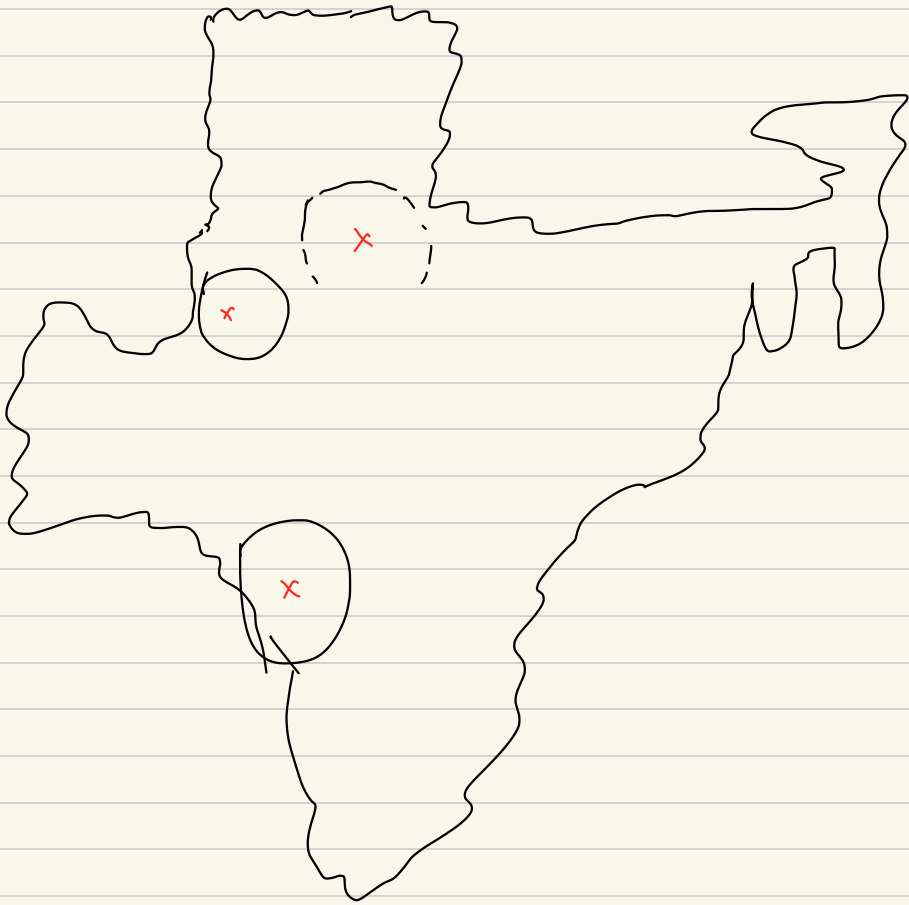


Uber Region – Paris



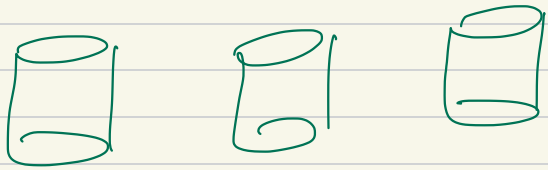
Intra City





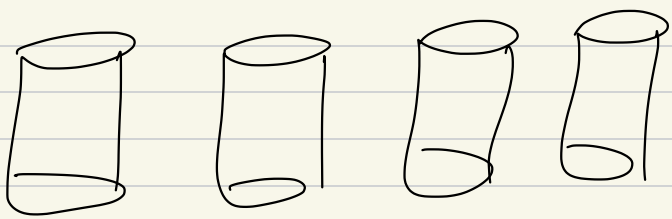
(3) Trips DB

SQL



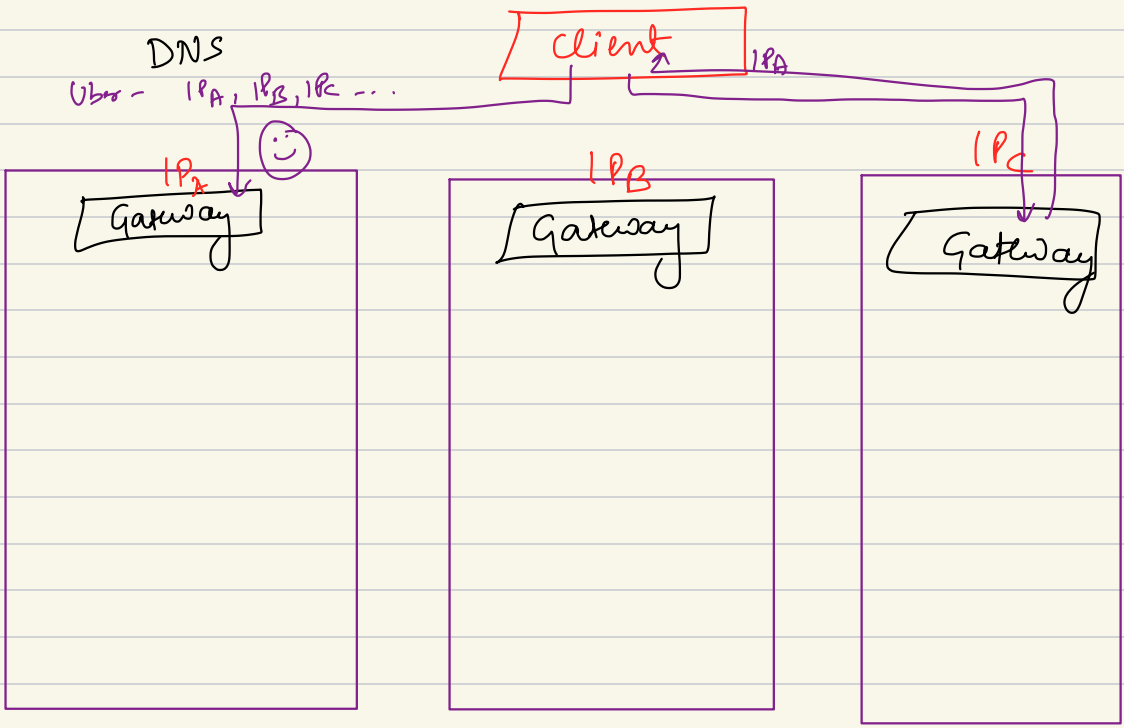
JIO

(4) Detailed location Trackers DB



wide column DB

Uber - as a collection of multiple Regional Uber systems



Zoom into any 1 Uber Region

→ 100M → 6.5 GB

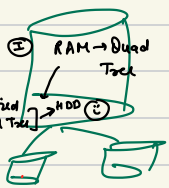
0.1M → few MBs

d_id → last location; last grid_call_id

Global Cache

driver's loc

Quad Tree



Gateway + LB

2

3

UDB

Trips

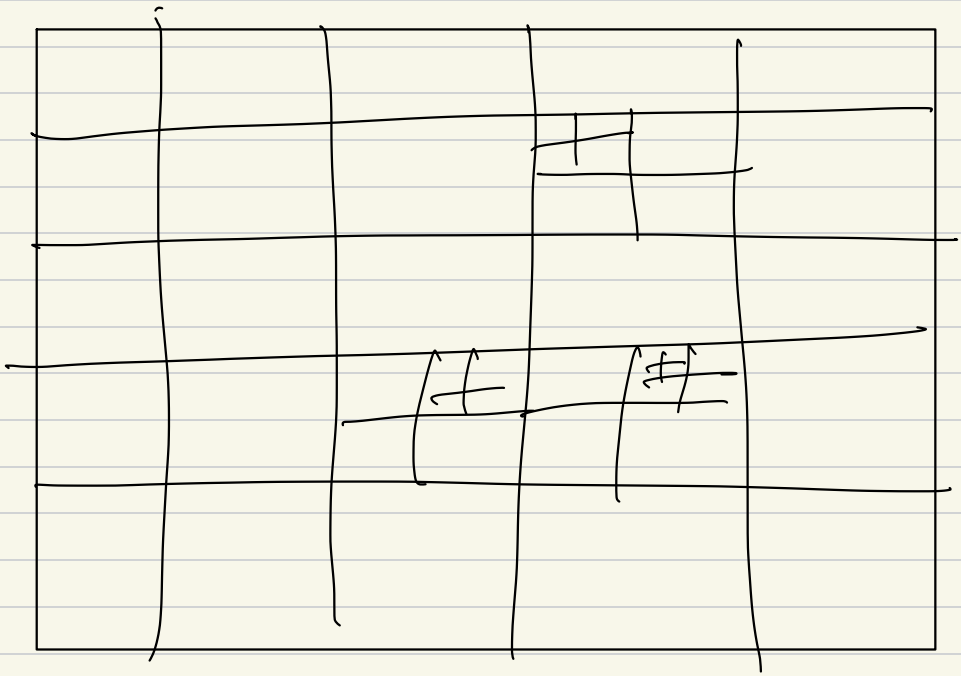
+ entry I

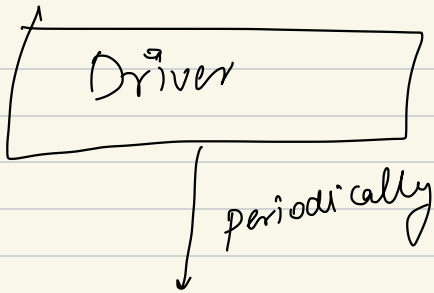
Location DB

10:39 – 10:49 PM

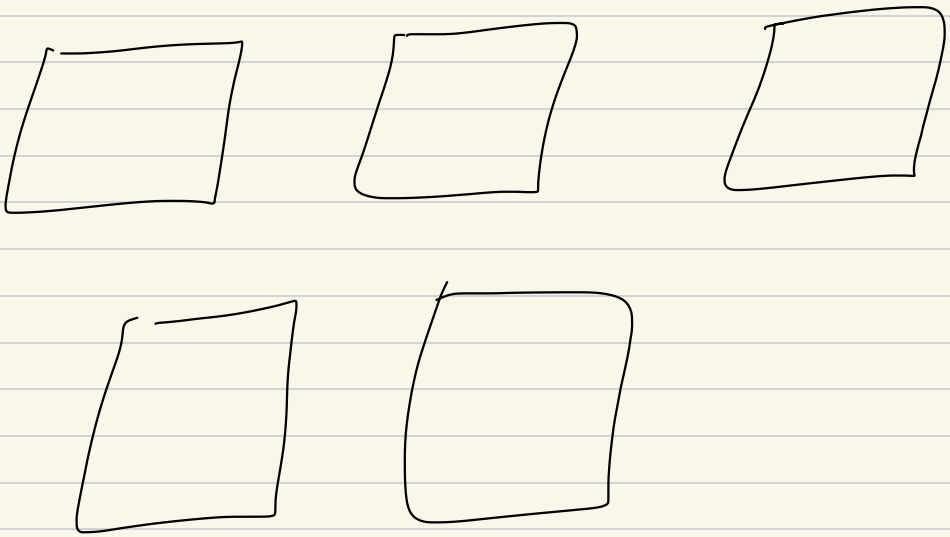
Step 1: Bootstrap your quad Tree

[some reference data to approximate]





Global Cache ≡ Redis Cluster

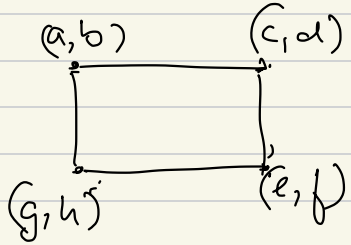


I

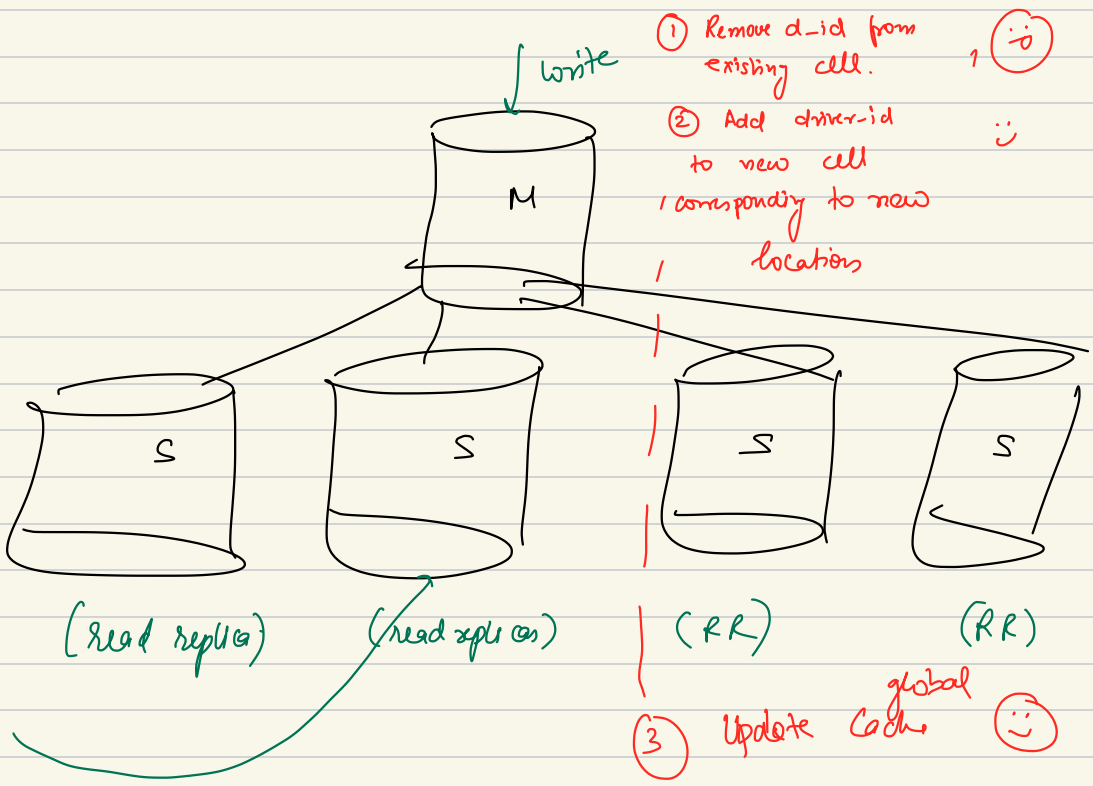
d_id → Last (x,y) location] ≈ Last cell_id

II

cell_id →



Quad Tree

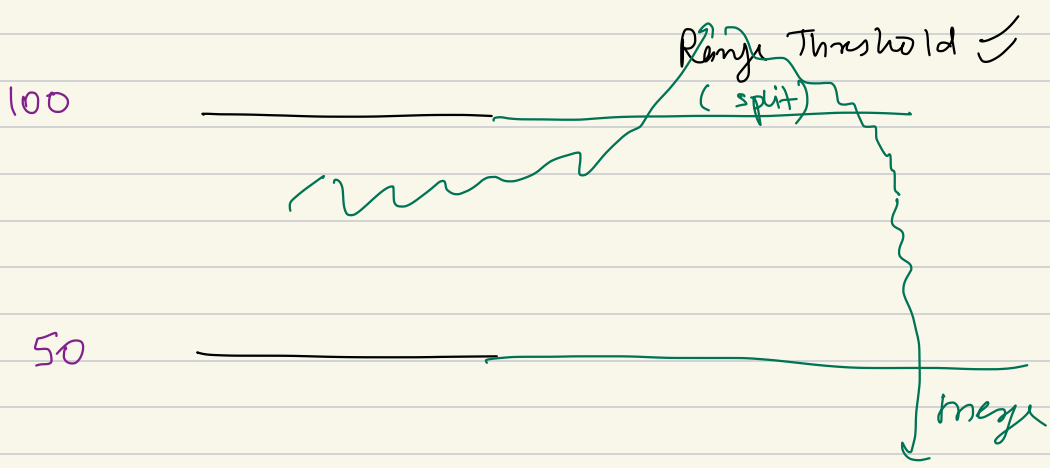


So, Uber system design seems like a clever use case of Quad Trees

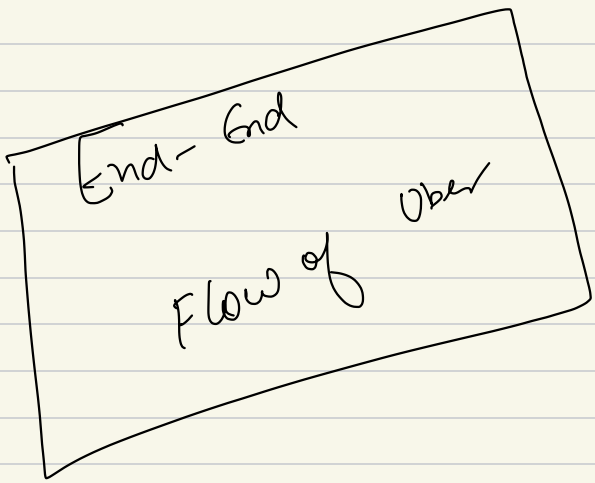
1. We have diff. subsystems for every region.
2. Inside each subsystem, we have an independent Quad Tree.
3. We are cleverly using Redis global cache clusters and/or slave (Read Replicas) of the Quad Tree machine to minimize the write requests to the Quad Tree.

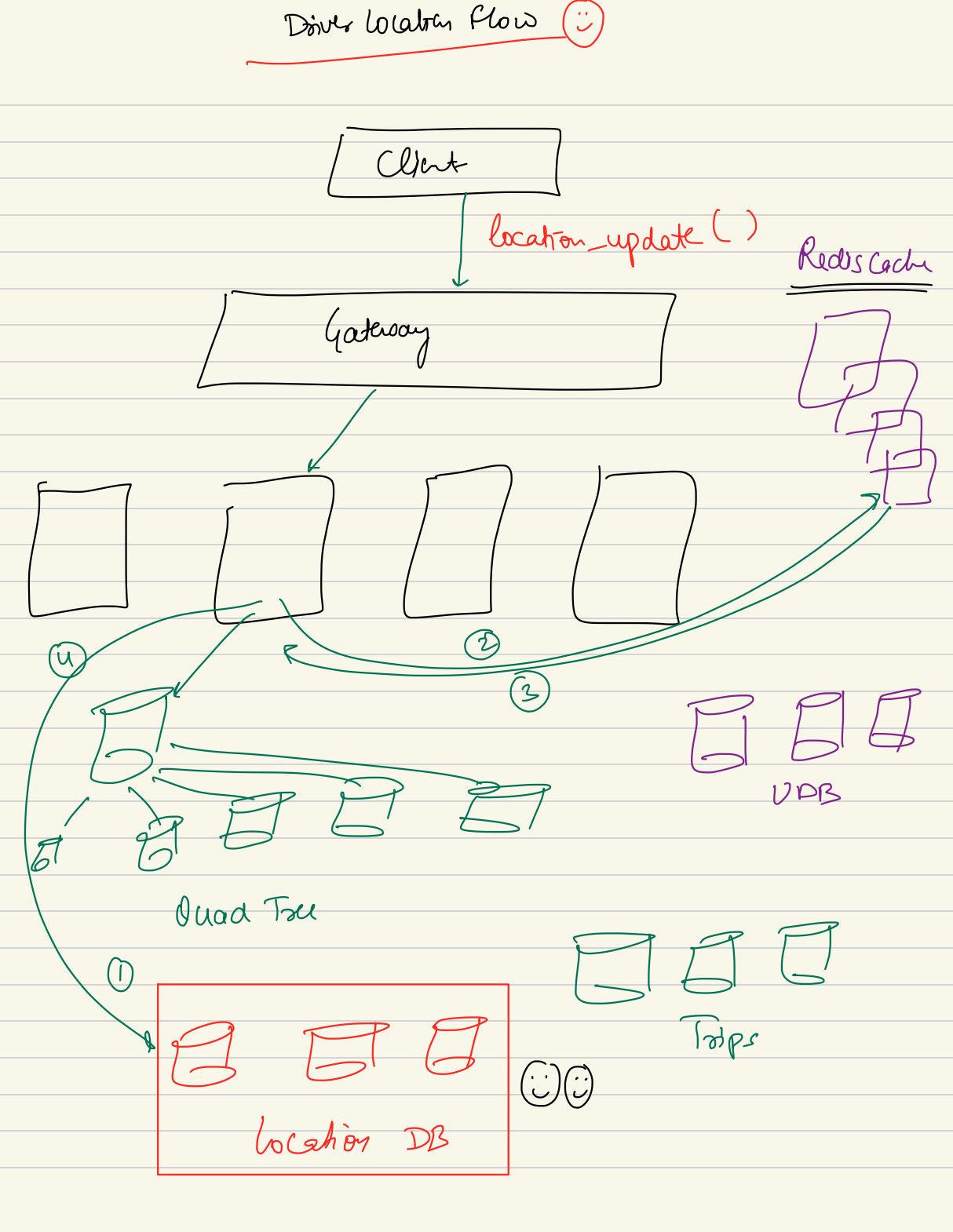
PROBLEM OF CONTINOUS MERGING and SPLITTING

1. Point Threshold ✗

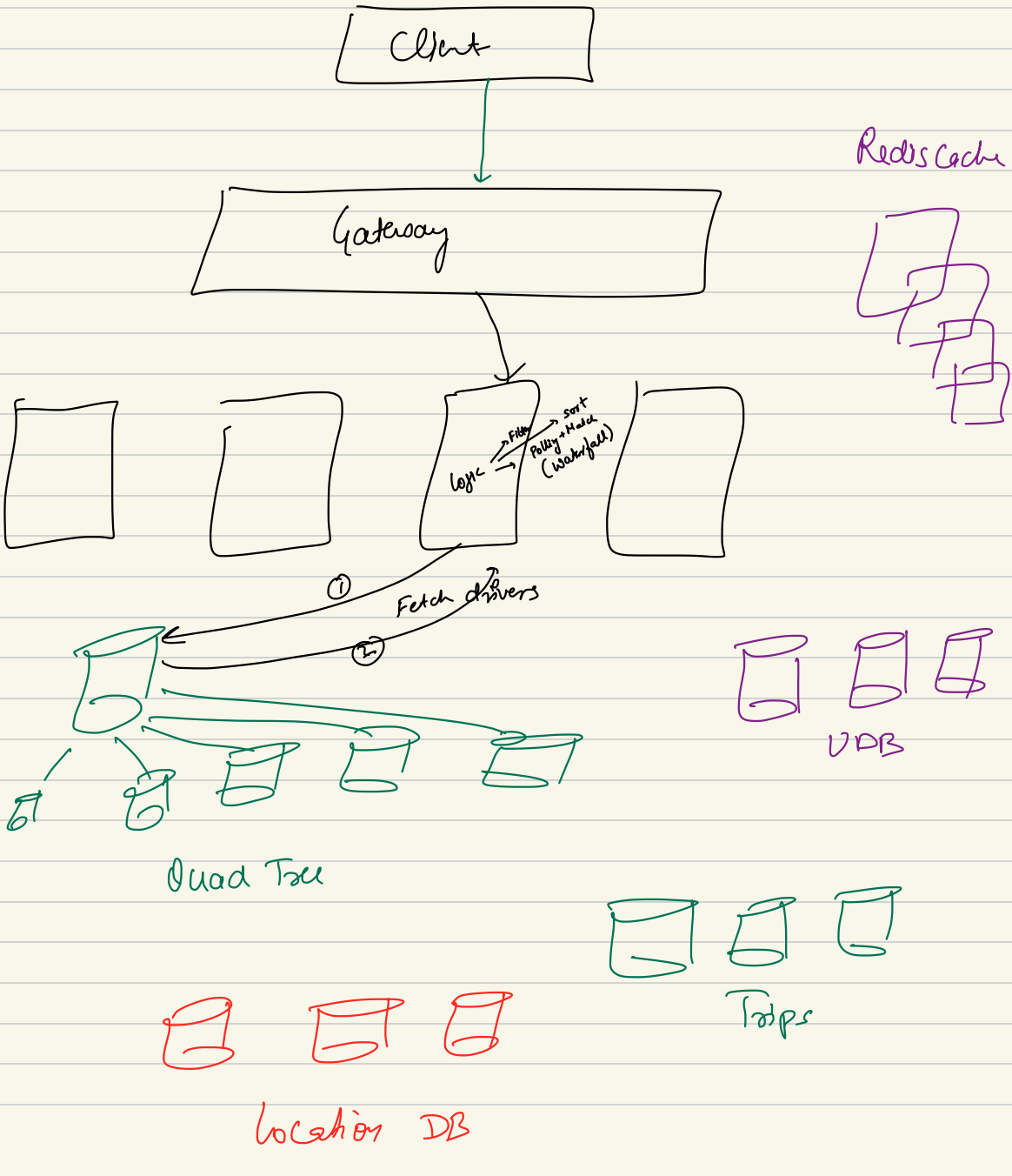


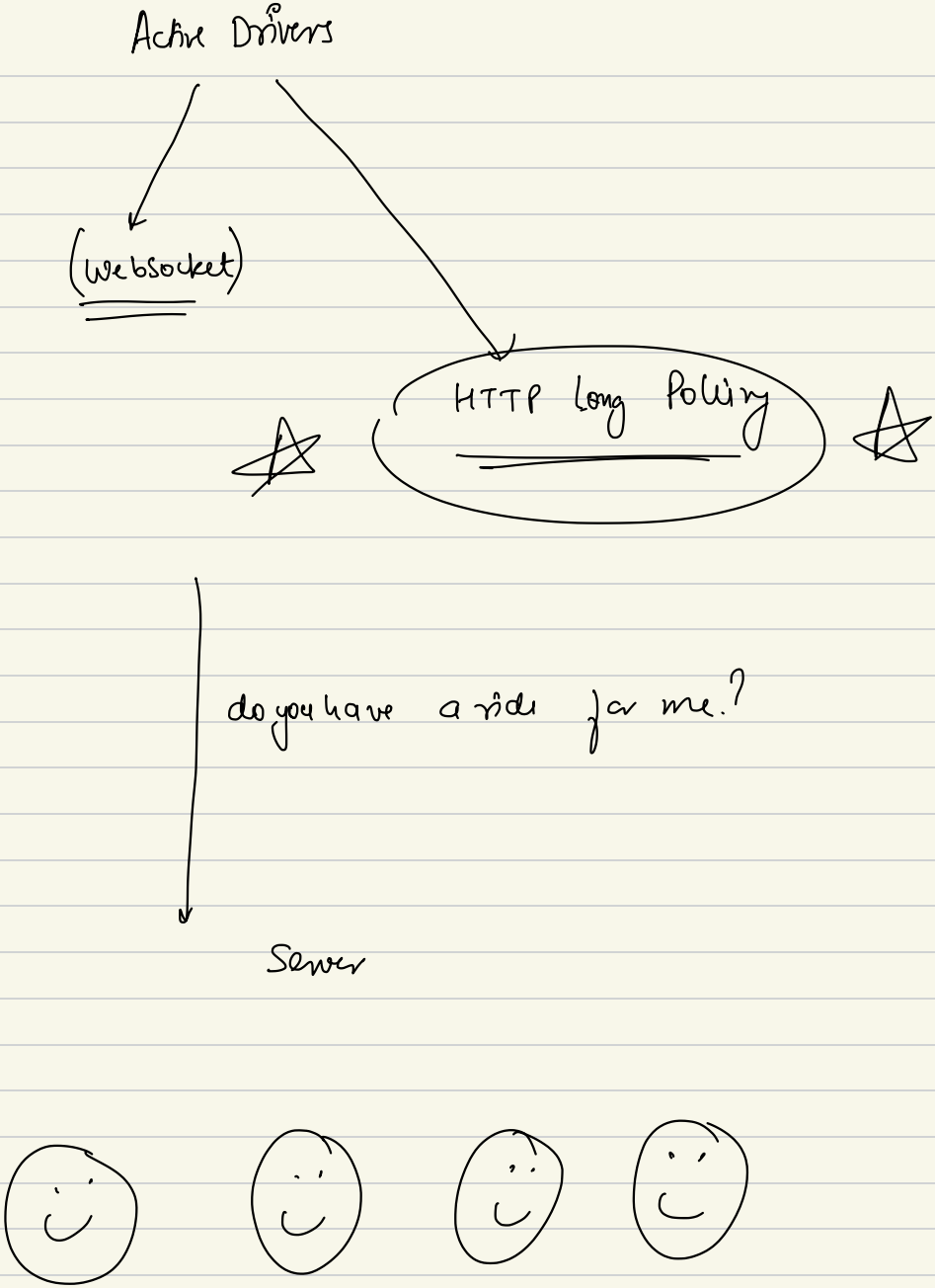
(2) Uber decides that grid cells are only going to change once ever 1 hour / 2 hour




Driver location flow
Client
location_update()
Gateway
Redis Cache
1
2
3
4
VDB
Quad Tree
Trips
location DB

Ride Booking Flow




Active Drivers
(Websocket)
HTTP Long Polling
do you have a ride for me?
Server