Agenda

- Problem Statement ✓
- Minimal Viable Product ✓
- Scale Estimation ✓
- Design Goals
- API
- System Design

Design / sharding / caching / database

↳ database schema

Problem Statement

IRCTC → Online ticket booking service for Indian Railways → transport backbone of India & most populous country.

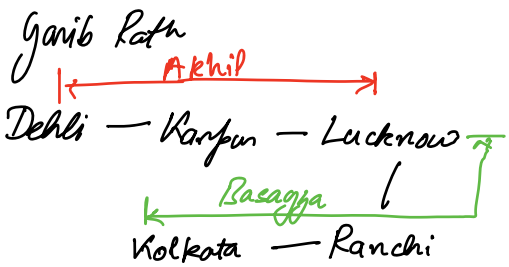
Tatkal → opens 1 day before

Emergency

at 10am — huge traffic spike

Minimal Viable Product (v0)

- login & registration
- search for trains
- train details — planned - schedule
  - stations & day + time
- seat availability (class)
- booking — passenger details
- payment
- seat allotment
- cancellation (v1)
- station wise seat breakdown (v1)
- seat preferences (v1)
- RAC (v2) → PNR
- notifications (v2)
- Integration with other services (accommodation booking) (v3)



Scale Estimation

end goal → estimate # queries / sec
↳ estimate data size

India Population → 1.5B

users → 1% of Indians are registered on IRCTC

$\frac{1.5B}{100} = \frac{1.5 \times 10^{3}}{100}\ M = 15M$ users.

5% = 75 M users

Actual # users ≈ 40M

MAU = all user base → 40M

DAU = 10% of entire user base ≈ 4M users

not everyone might be booking (20-80)

Estimate # bookings per day = 1 → 5 M per day
20% of 4M

Nikhil → # trains = 10,000 → 10^4

assume 100% occupancy

approx # seats per train → # coaches per train (15) * # seats per coach (72)

= 15 * 72 = 1080 → 10^3

9 segments / coach
≡
≡ 11 } 8 seats per segment

approx # seats booked per day $= 10^4 * 10^3$

$= 10^7 \approx$ 10 Million.

tickets < # seats — each ticket on avg books 2 seats

∵ 1 ticket can demand multiple seats

$\frac{10M}{2} \approx 2M$

bookings (seats) per day = 1 M — 10M

bookings per second? = $\frac{\bullet}{86400}$ = $\frac{10^6}{10^5} = 10$ $\frac{10^7}{10^5} = 100$

peak

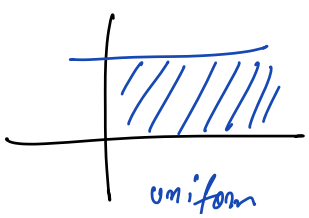
10^5

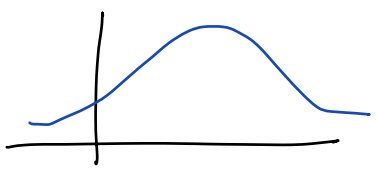
normal tickets

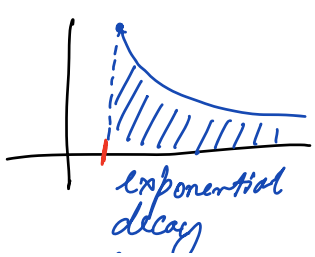
tatkal tickets

9am 10am 11am 9pm

AC Non-AC







(law of "U long nurbs) (Zipf's law)

many independent distributions
the tray avg out to gaussian.

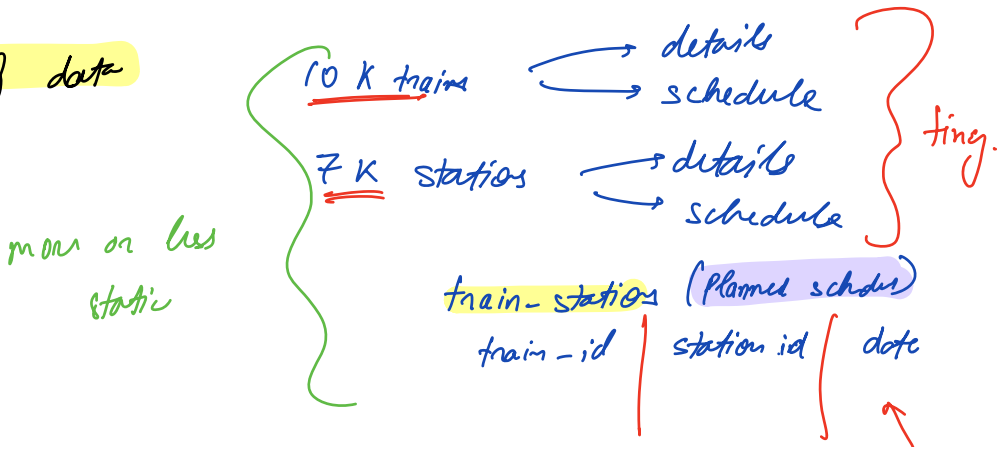
Actual scale

- # users ≈ 40M
- # stations ≈ 7K
- # trains ≈ 13K (passenger) + 7K (freight)
- # tickets/day ≈ 1.1M (avg for 2021–2022)
  - spikes during holiday season

tickets/min (peak) = 26,458 tickets/min
On Nov 2020

QPS ⇒ $\frac{26K}{60}$ = 430 bookings/sec

Amount of data



10k * 20 ≈ 200K entries / tiny

changes

Booking

id	user_id	train_id	src	dest	class	created_at	seat_id
8	8	8	4	4	1	8	4
PK	FK	FK					

Booking_passengers

booking_id	passenger_name	age	meal	berth_pref	Doc Type	Doc_id
8	50	1	1	1	1	30

≈ 100 bytes
= 100MB per day

Size of row = 50 bytes

booking / day = 1M

data per day = 50M bytes = 50MB

20 years = 50M * 20 * 365 ≈ 365 B bytes

≈ 365 GB

= 500 GB (total data)

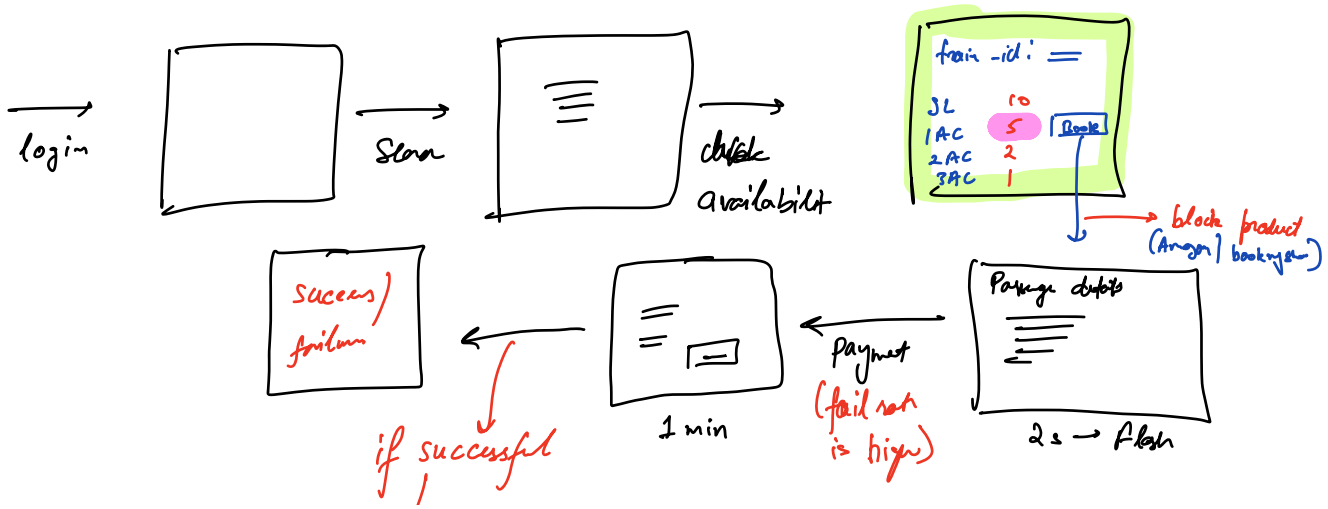
Do we need sharding

∵ of data size? No !!

∵ of load → we'll see

Design Goals (Non-functional requirements)

- ✓ login & registration → separate microservice
- ✓ search for trains → availability over consistency
- ✓ train details — planned - schedule
  - stations & day + time → read / write → once a month
- ✓ seat availability (class) → read / write
- ✓ booking — passenger details → consistency / availability
- ✓ payment — independent service → no reads / writes (user input)
- seat allotment → consistency / availability. read / write
- cancellation (v-1) → availability / consistency read / write
- station wise seat breakdown (v1) → both! → check → block
- seat preferences (v1)



Lack of seat

There is a significant time gap b/w seeing the availability & seat allotment.

Design

System - 1

1. List of trains
2. List of stations
3. search trains based on source, dest, date
4. train details
   - number
   - planned schedule

- read heavy
- mostly static
- prefer availability
- little data
- relational

SQL + Redis Cache
↳ Invalidation (write-through)

trains

id	name	number	details	---
1	garib	100	—	

stations

id	name	num-platforms	category
1	Delhi		
2	Lucknow		
3	Kol		

planned-schedule

train-id	index	station-id	day-of-week	arrival-time	departure-time
1	0	1			
1	1	2			

1 2 3

Seat availability → read & write heavy
whenever a booking happens
for availability changes.

→ availability over consistency.

System-2

DB → SQL

Redis cache

Invalidation → write through → inc booking latency not needed

→ TTL / write around

10:48 → 11:00 pm

Seat Allotment / Ticket Booking

(train_id, user_id, class, source, dest, date,
#seats, berth-preference for each seat, meal for each seat)

① Ensure seat is available → seat-availability-service
② mark seat as booked.

X cannot be used ∵ cache and immediate consistency

↳ both can be done in a single query/transaction.

db ensure ACID

Begin Transaction
→ get list of available seats that match criteria
→ mark some as booked
→ return success/failure
↳ seat nos
End transaction

400 bookings/sec at peak → a single server can Not handle this

Dealing with load
↳ cache → reads but not writes
↳ replicate → reads but not writes
↳ Shard → Only way to handle high write throughput.

Sharding-key → date of journey — load is unequally distributed.

→ train-id — every booking depends on train & every train is independent.

booking for any date of journey that has passed → moved to a diff db at end of day.
CRON Job.

↳ Total live data → amount of active booking
(today → 2 months in future)

1M booking / day
60M bookings (active)
50 bytes / booking

= 3GB of data (tiny!!!)

S. Sudarshan → Head of CSE at IIT-B
↳ god of databases

1ms → 500ms for a query to run in db

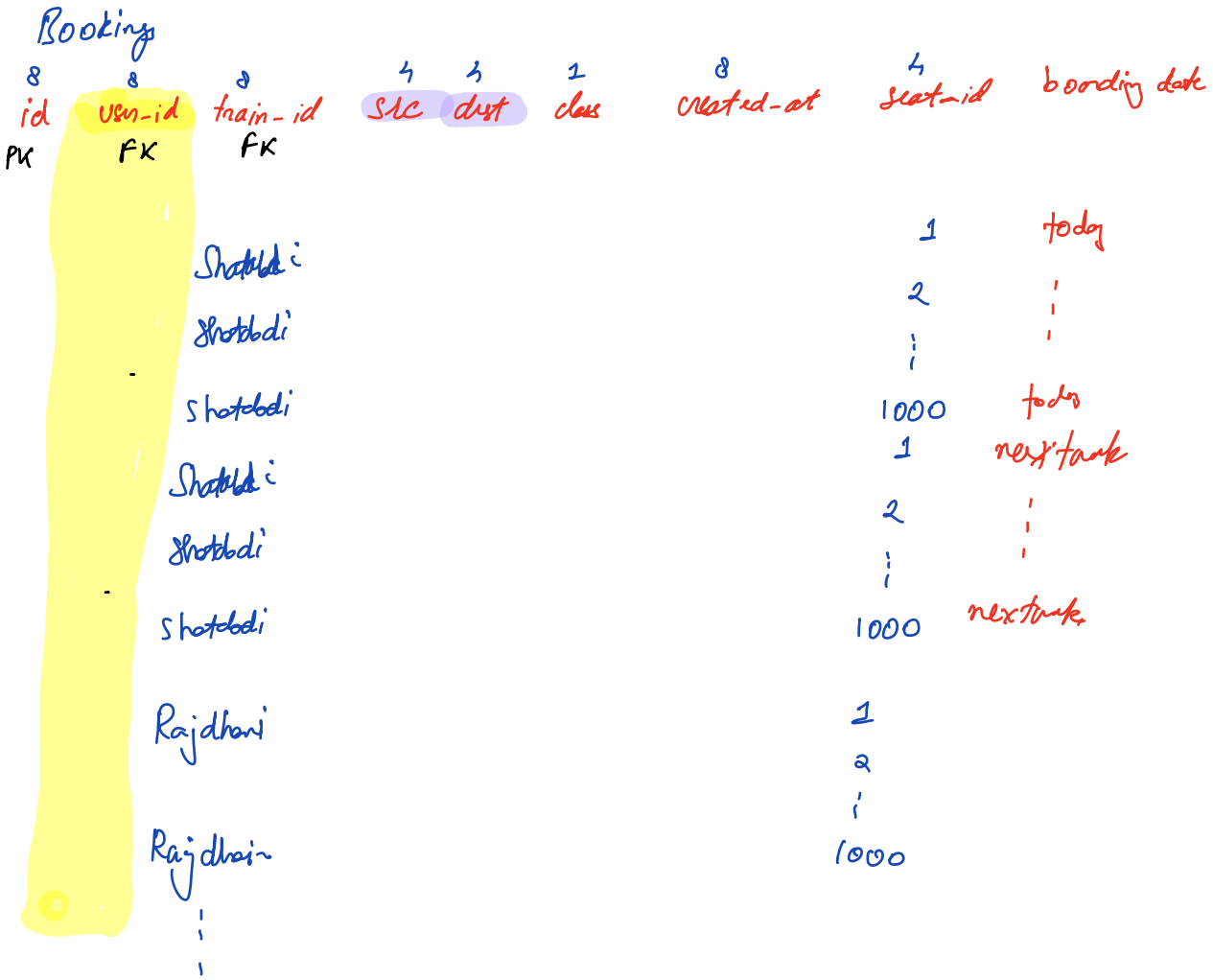
1000 writes / sec → 2 writes / sec

Database Schema

↳ berth preference
↳ booking same seat over non-overlapping stations.
↳ Check which seats are available

bookings
store booked seats
(Σ − booked seats) → available seats

pre-create seat-train combinations.



① explode db to pre-emptively create seat-train combinations for next 60 days

② usr_id is initially blank

③ if seat gets booked → usr_id gets filled

update bookings

set usr_id = req.usr_id
src = :
dest = :

→ mark seat as booked from src to dest

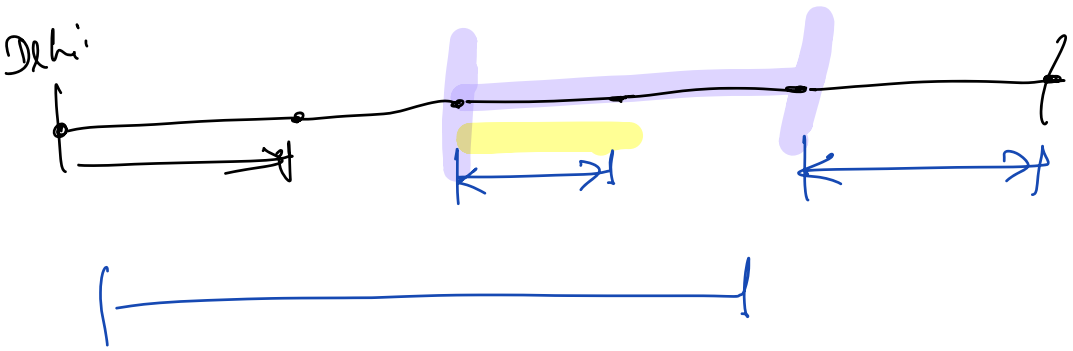
Seat is blocked for entire journey! (bad)

where train_id = req.train_id
date = :
class =

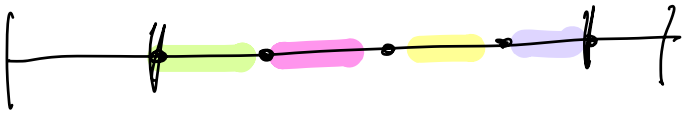
usr_id is NULL → seat is currently unoccupied.

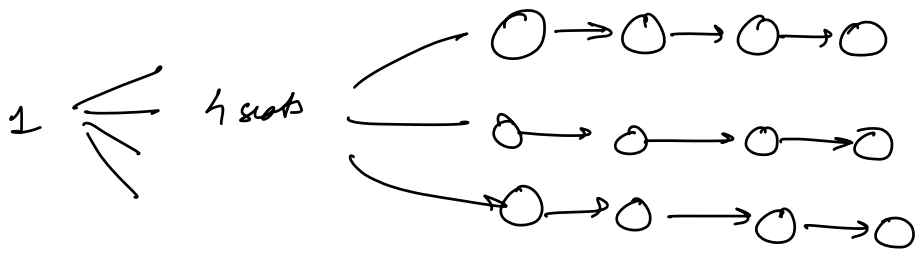
order by select(1 case berth_type = req.bt 0 else) desc

limit req.seat_count



each row in bookings-table only represents journey from 1 station to the immediate next station.





~~dest~~ → src+1 station

id | user-id | train-id | date of journey | src | seat-id | berth-type | class

10^4 10^2 60

Pre-populate for every train, for every seat, for every date, for every station!!!

~~10^2~~ ↳ # station for train on avg 20

size of table = $10^4 * 10^3 * 60 * 20 = 1.2 \times 10^{10}$ entries

each entry is 100 bytes ⇒ total size = 1.2×10^{12} bytes

= 1.2 TB

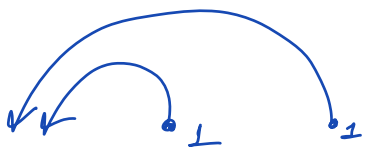
1 train → T1

5 stations → S1, S2, S3, S4

T1 → S1 — S2 — S3 — S4

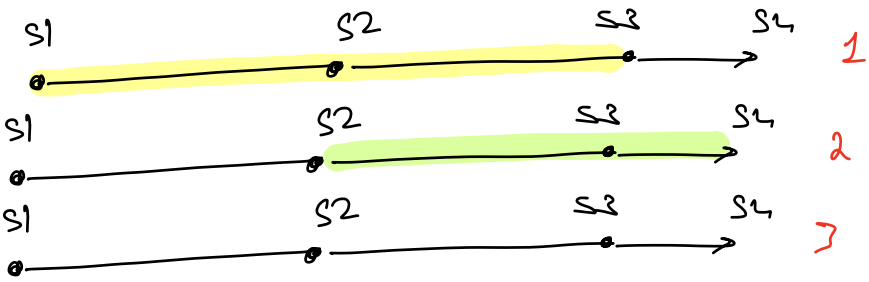
each train has 3 seats → 1, 2, 3

two days → 10, 11th

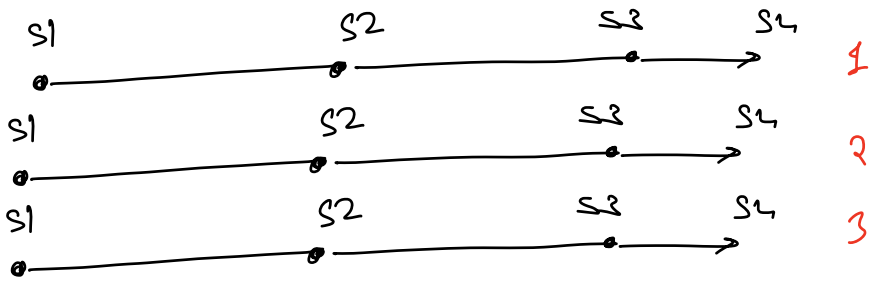


id	user_id	train_id	date of journey	src	seat_id	berth_type	class
1	v	T1	10	S1	1		
2		T1	10	S1	2		
3		T1	10	S1	3		
4	v	T1	10	S2	1		
5	n	T1	10	S2	2		
6		T1	10	S2	3		
7		T1	10	S3	1		
8	n	T1	10	S3	2		
9		T1	10	S3	3		
10		T1	11	S1	1		
11		T1	11	S1	2		
12		T1	11	S1	3		
13		T1	11	S2	1		
14		T1	11	S2	2		
15		T1	11	S2	3		
16		T1	11	S3	1		
17		T1	11	S3	2		
18		T1	11	S3	3		

10th Apr



11th Apr



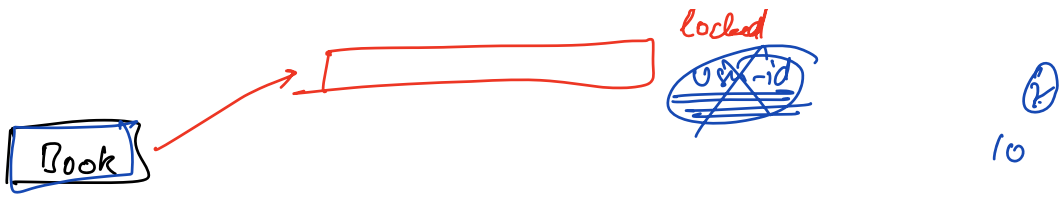
Varaja → 10th 1 seat S1 to S2

{S1 ↓ S2 , S2 ↓ S3}

Neeraj → 10th 1 seat S2 to S4

S2 → S3
S3 → S4

→ Berth prefer
→ gender
→ VIP
→ Quotas (women, elderly, veteran, vip, staff)



72 props for bogie → 7 tons

bogie → 1000 tons.

---

① List requirements

② identify non functional

③ solutions

Read heavy → caching / replication

Write heavy → Sharding

Read & write heavy

↳ break down into components

↳ R+W heavy → Sharding.

Master - Slave

1. write → leader
   read → any slave
   — No consistency

2. write → leader + 1 slave
   read → any slave
   — eventual consistency, high availability

3. write → leader + all slave
   read → any slave
   — immediate cons, low availability
   reads — fast, writes — slow

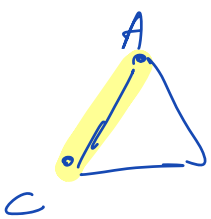
4. write → majority slaves
   read → majority slaves
   — Quorum, immediate cons, somewhat availability
   reads & writ → slow

Multimaster

X → Replicas for
R → # Reads
W → # writes

R+W > X → immediate
<= X → eventual

any spectrum

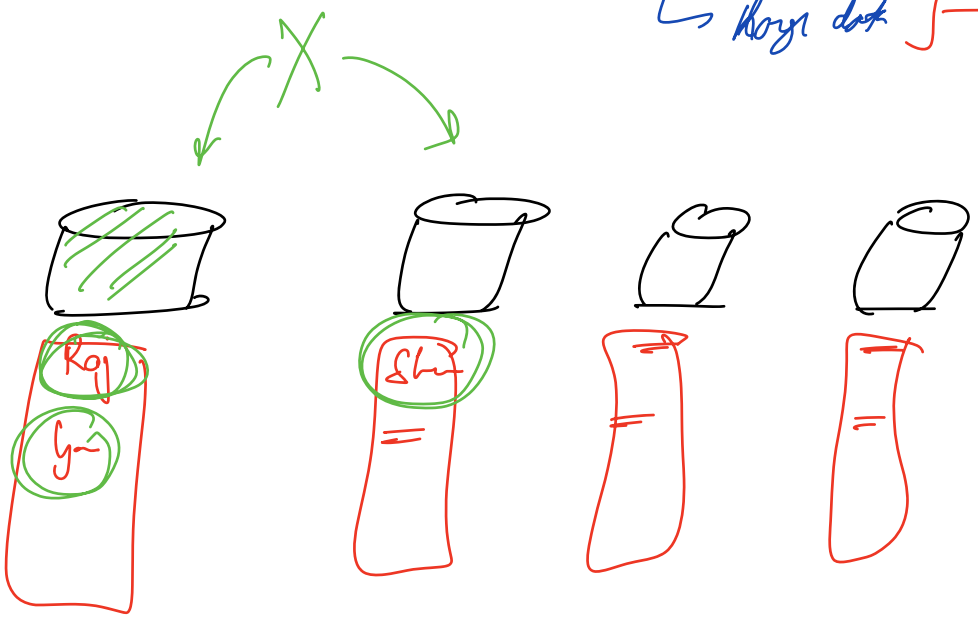


R+W heavy ✓

immediate consistency ✓

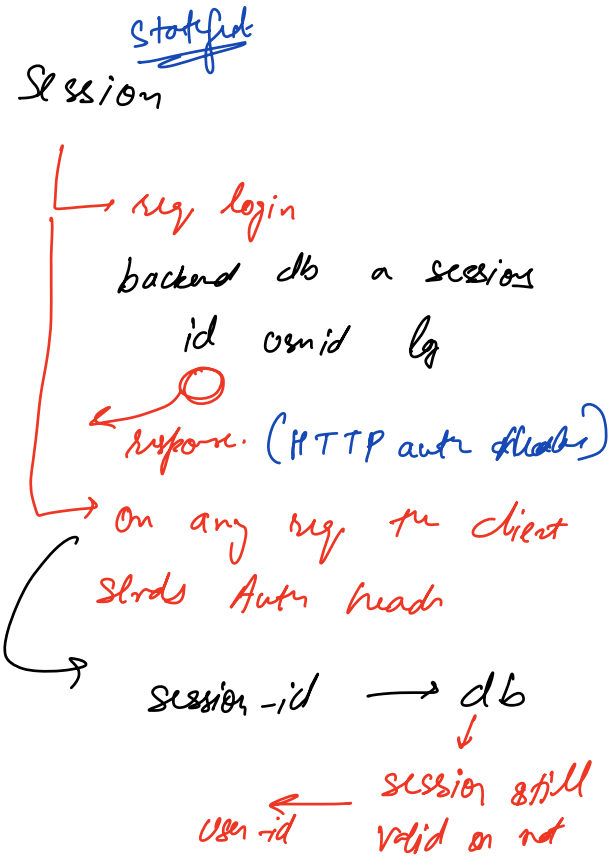
Sharding

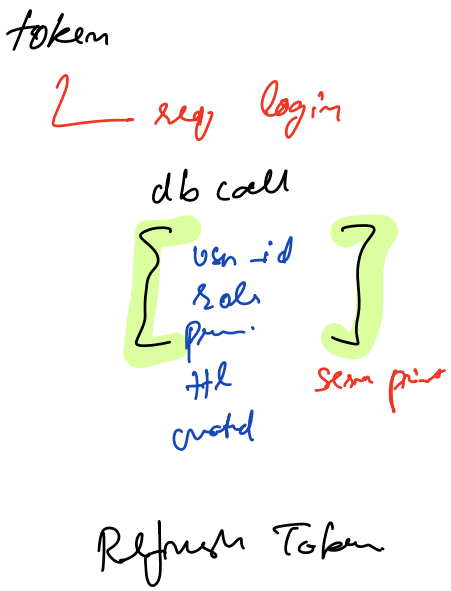
- high load → low
- Huge data → small

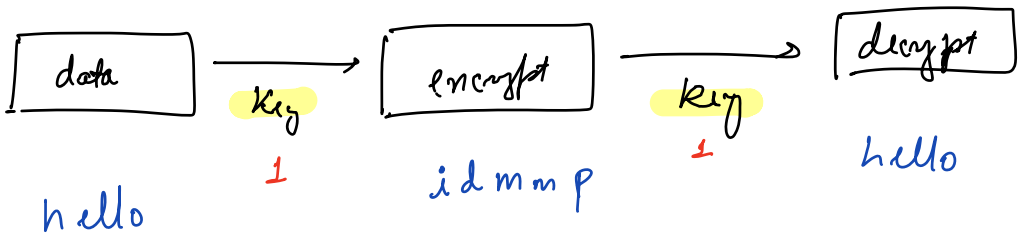


Single server → read qps > 1000 per sec
write qps > 100 per sec
} NoSQL

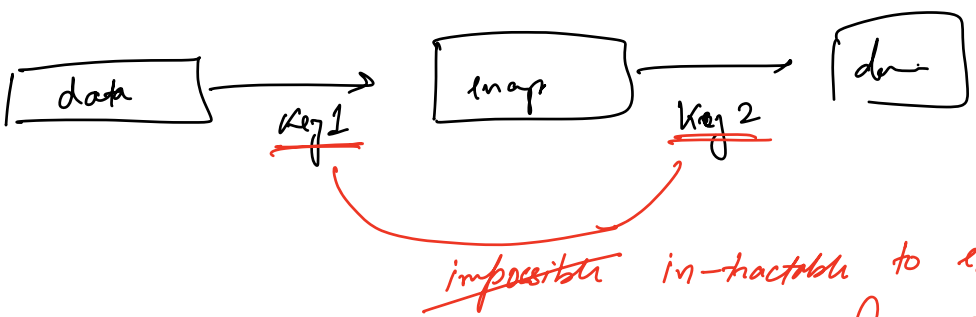
- Symmetric-key cryptography
- Asymmetric-key crypto


Stateful
Session
req login
backend db a session
id usrid lg
response (HTTP auth
On any req to client
sends Auth header
session_id
db
session still
valid or not
user_id

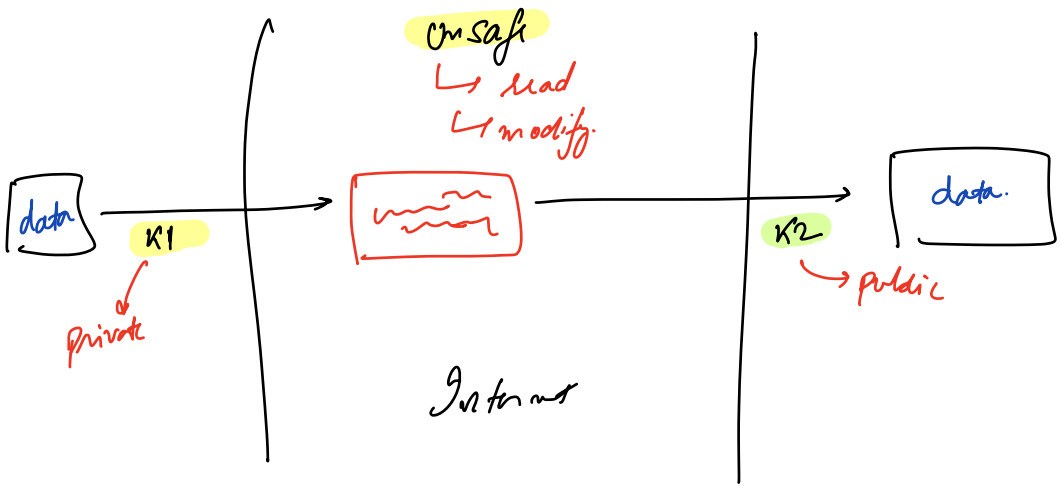

token
req login
db call
usr_id
role
pwd
Hash
created
Secret pwd
Refresh Token



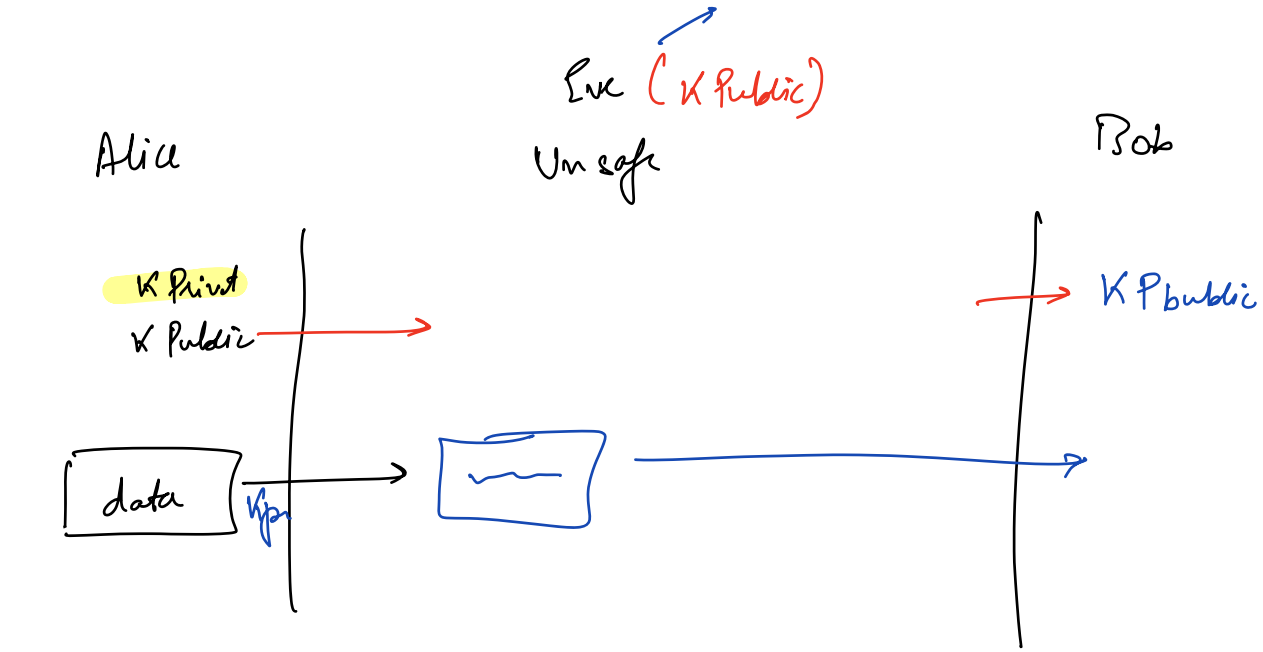
Caesar Cipher with k=1

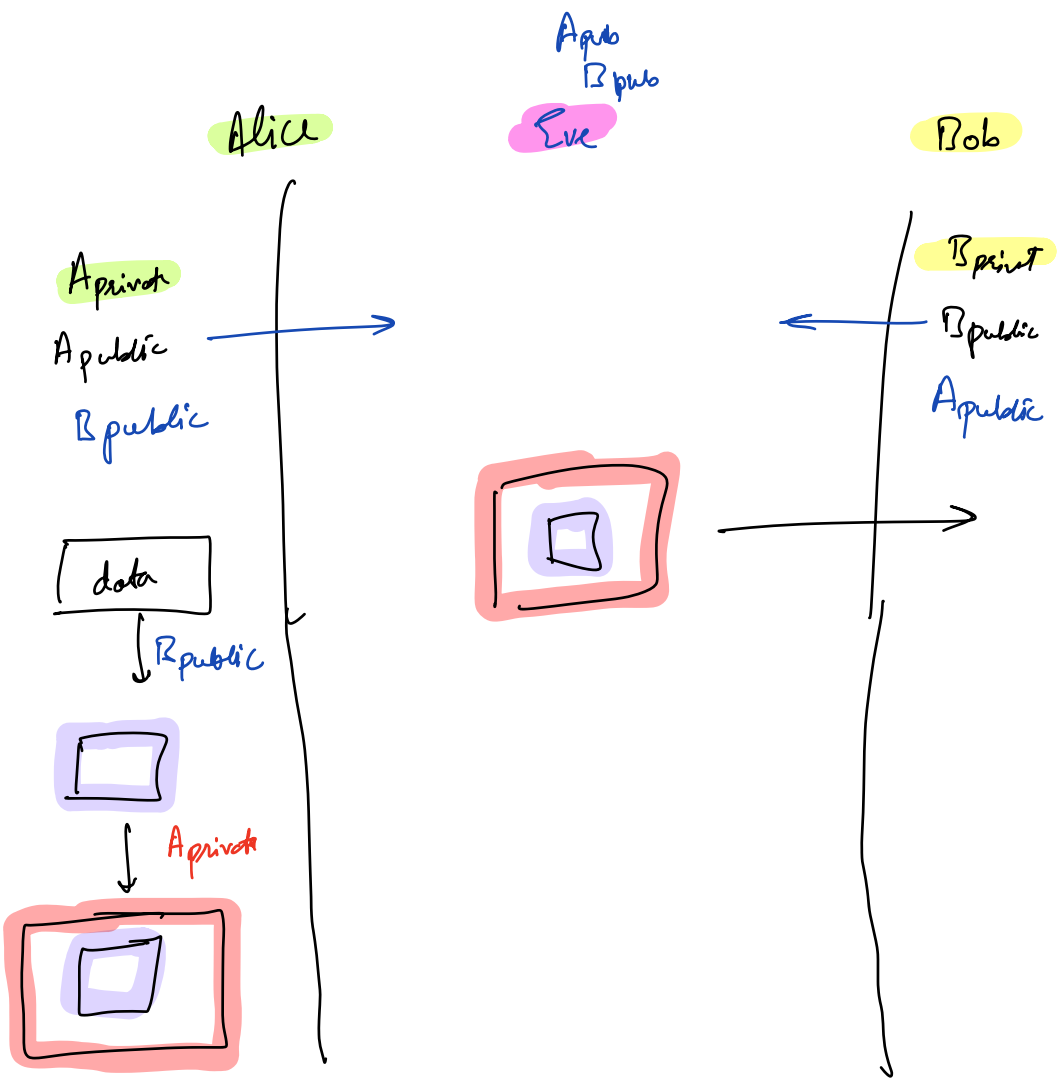


impossible intractable to extract one key from another.



Eavesdropper


Eve (K Public)
Alice
Unsafe
Bob
K Private
K Public
data
K Public


Apub
Bpub
Alice
Eve
Bob
Aprivate
Apublic
Bpublic
data
Bpublic
Bprivat
Bpublic
Apublic
Aprivate

Public key —impossible→ Private key
in-traction