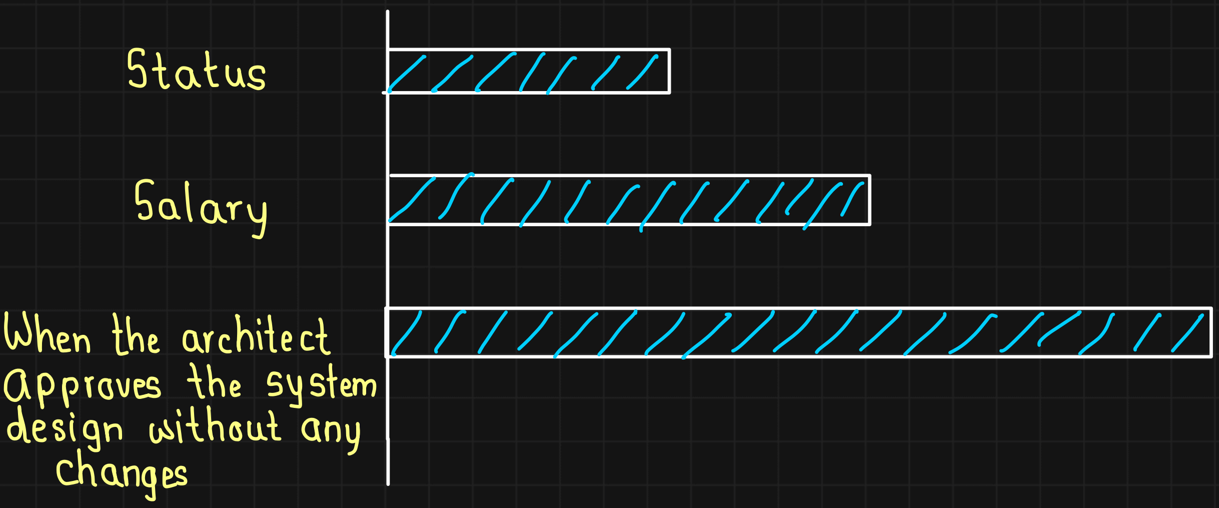
Approaching Any System Design Discussion

Chapter 0:
Let's Dive In

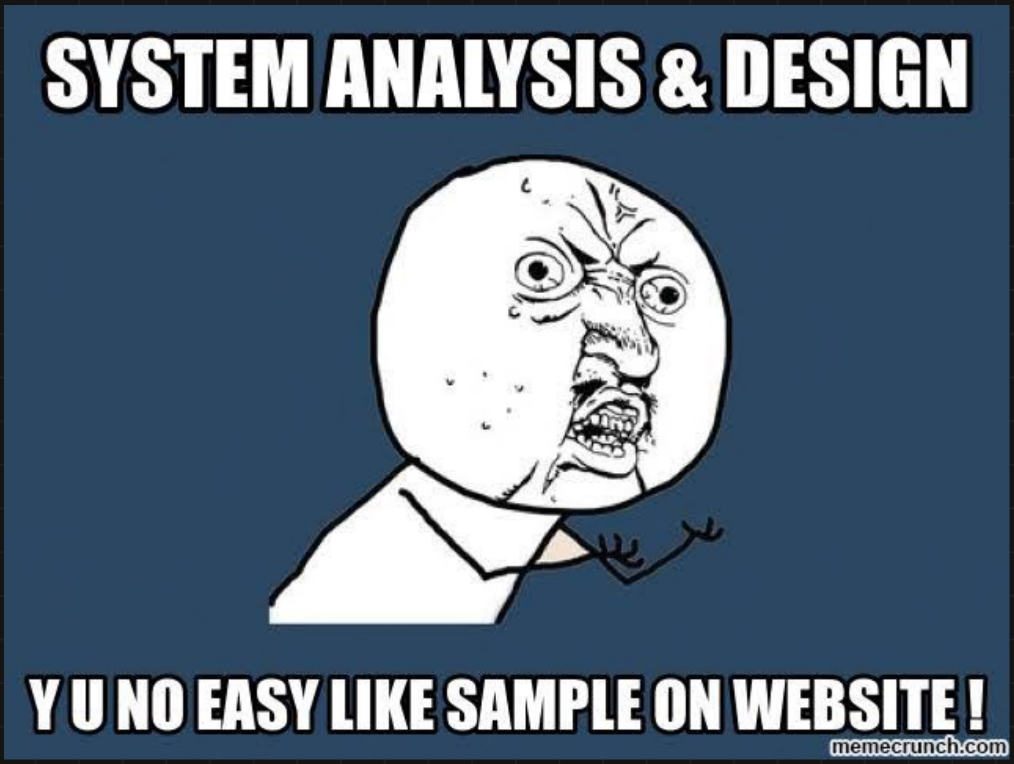
Approaching any System Design Discussion

Let me ask you, one simple question.

What gives backend developers feelings of power?

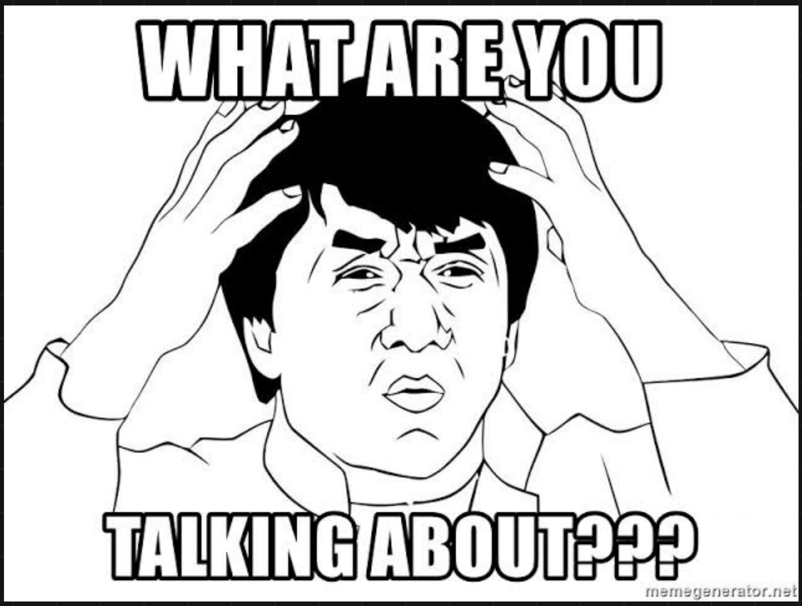


Most of us in system design discussions:



Buzzwords in Software Engineering in recent years

1) NoSQL !
2) Big Data / Map Reduce !
3) ACID !
4) Web Scaling !
5) DB Sharding !
6) CAP Theorem !
7) Eventual Consistency !
8) Real time !
9) Cloud Services !

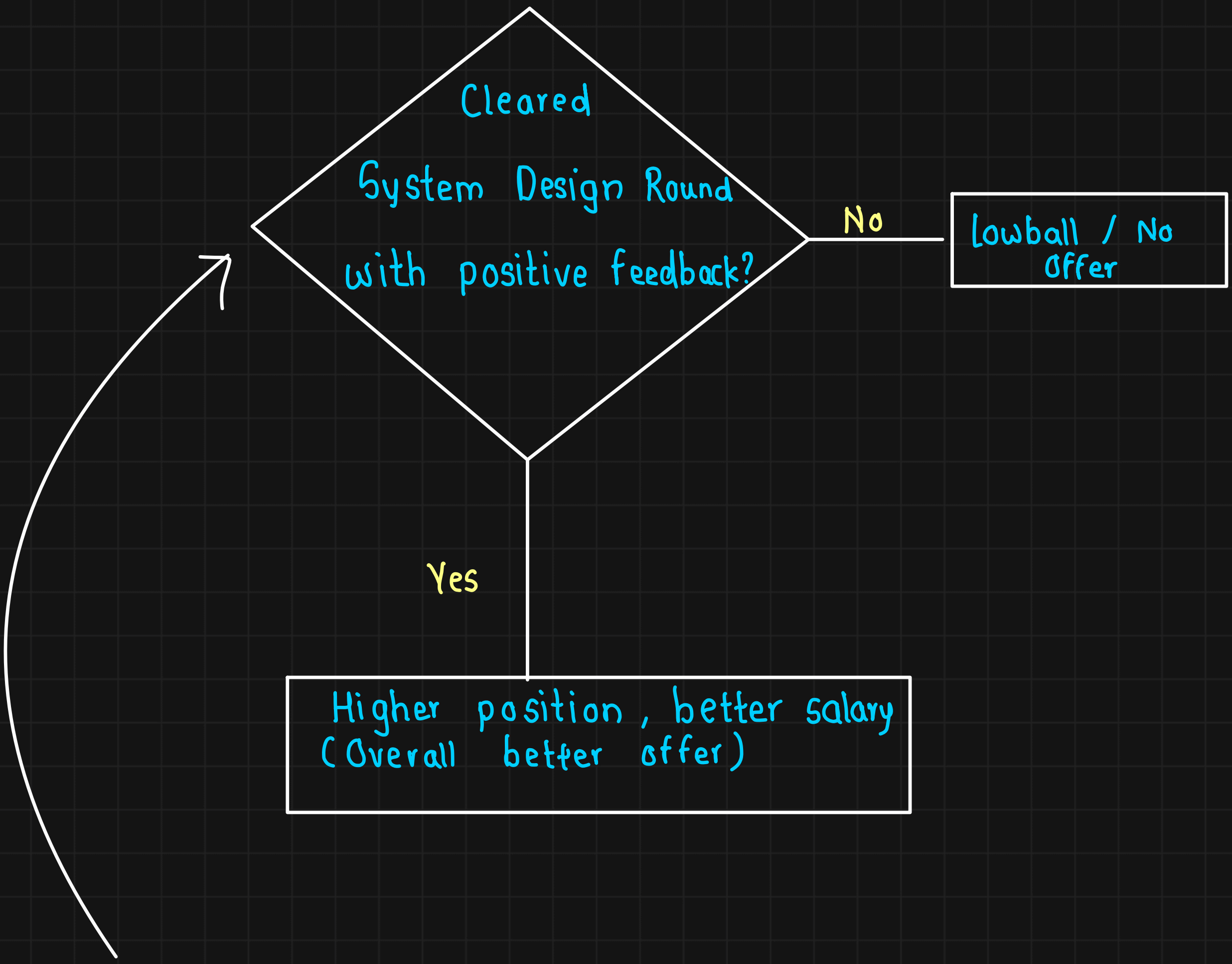


If you are as confused as Jackie Chan in the above image, check out my last post on System Design Basics.

↳ Link is in the post.

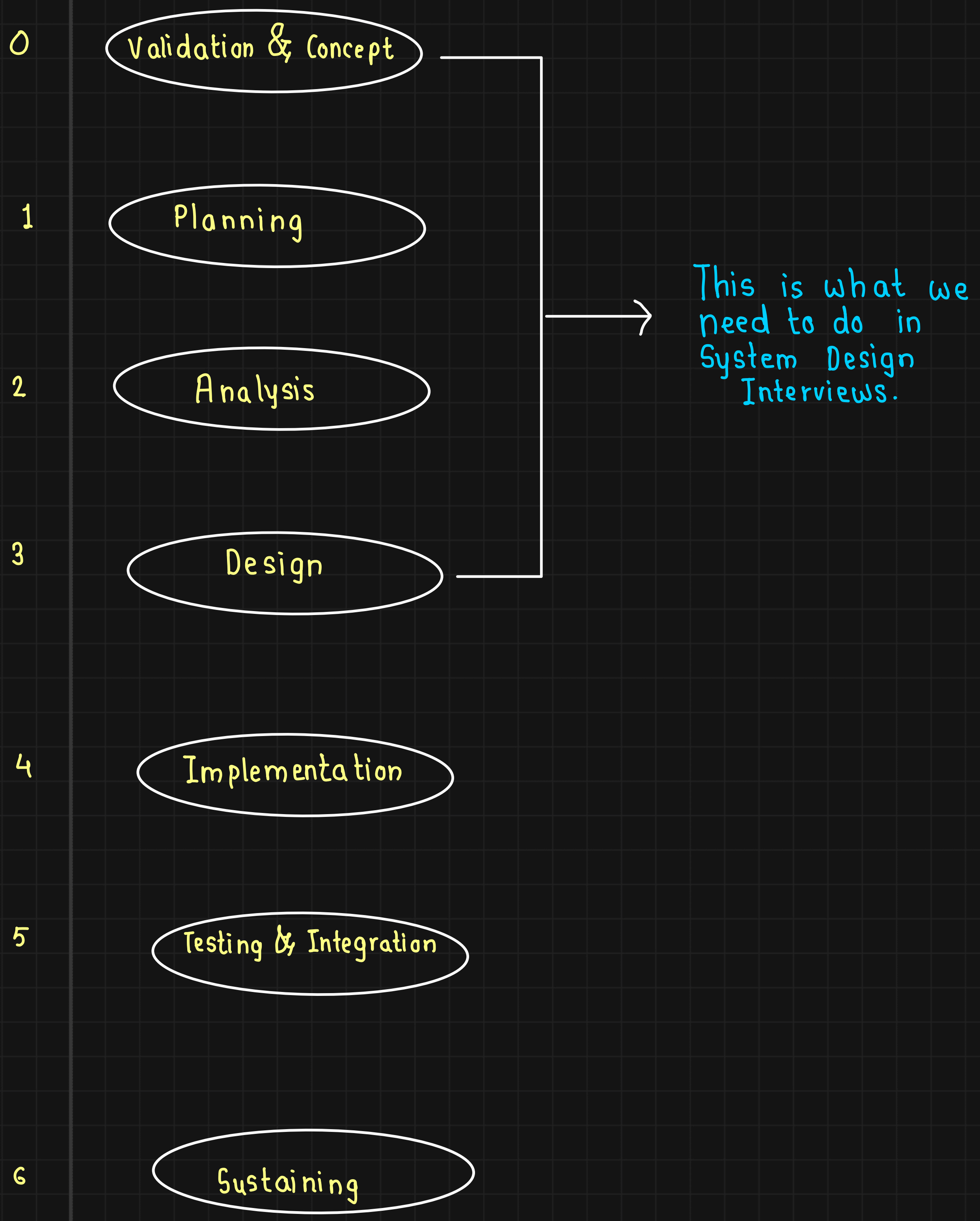
Problems that I faced in design discussions

1) Open - ended problem
2) No standard / ideal / perfect answer
3) Unstructured nature of interviews.
4) Lack of experience in building distributed systems
5) Not enough preparation.


Cleared System Design Round with positive feedback?
No
Lowball / No Offer
Yes
Higher position, better salary (Overall better offer)

Shows your ability to handle complex distributed systems.

What do we follow in SDLC (Software Development Lifecycle)?



Steps that you can follow while approaching any SD.

1. Requirements/ Goals Analysis

→ Ask questions } Will help you in defining scope.
→ Get your doubts cleared.
→ Keep in mind (No answer is perfect)
→ Spend enough time in requirements analysis (Do NOT rush)
→ Real life systems does NOT consists of 1 or 2 parts.
(You have limited time (40 minutes around),
clarify / ask interviewer what parts of system you should focus on)

Going ahead, for each step,
I'll try to give different design considerations
for developing movie ticket booking system like
BookMyShow.

book my show

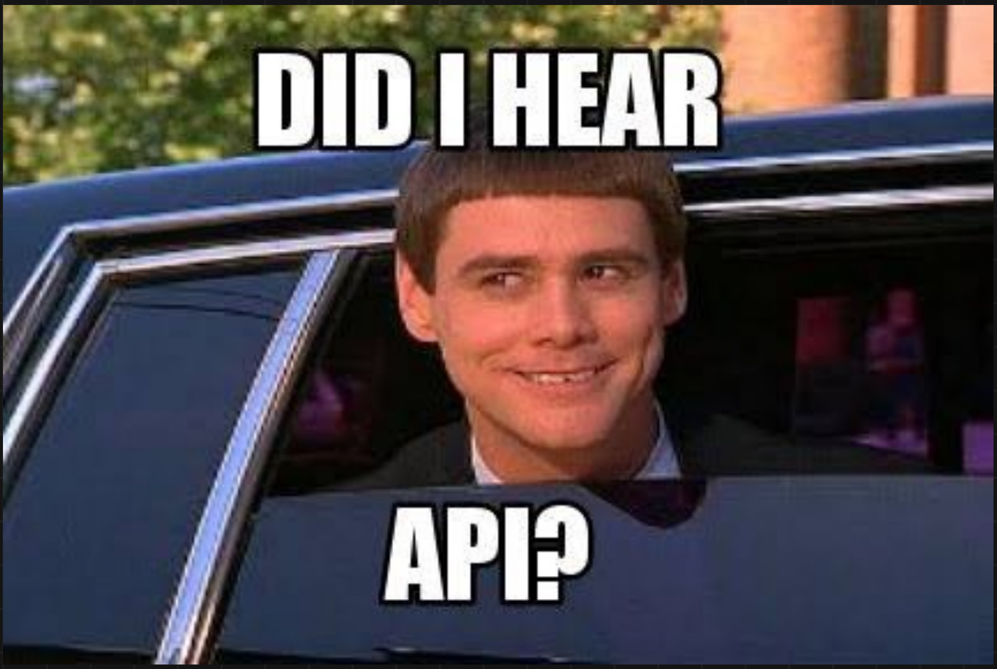
Here are some points for designing BookMyShow that should be
discussed before moving on to next steps:

Functional Requirements:
1) List down cities
2) After selecting city, we need to list down movies.
3) On selecting movie, system should show cinema halls & shows.
4) User should be able to select cinema hall, show and seats.
↑
Should we add any limit in seats booking?
5) Distinguish betⁿ booked / on hold / available seats.
How much time should we allow before payments to release the
booked seats?
Are we focusing on backend only or are we developing frontend to
Do we need to display trending movies as per location?

There can be many more questions.
All these questions will help in determining how our end design
look like.
(Note: We'll discuss more about Non-functional Requirements
in the upcoming chapter)

2. API Design

→ Discuss what APIs are expected from the system
→ Establishes exact contract from the system.
(Also helps in validating the requirements specified in first step.)



Some API definitions for our BookMyShow-like service :

searchMovies (keyword, city / lat_lang , radius = x km , keywordSpellCheck, start_datetime , end_datetime, results_per_page, order_by)

reserveSeats (user_session_id, movie_id, cinema_hall_id, show_id, seats_to_reserve [ ], mobile_number, email_id)

→ Will return the status of the reservation.

→ Will return JSON with list of movies & shows.

3. Approximate Scale Estimation

→ Always better to estimate the scale of the system
→ How it'll help you?
- → Scaling
- → Partitioning
- → Load balancing
- → Caching

→ Traffic Estimation:
Assume number of pageviews (x billions) / month
tickets sold (x millions) / month

→ Storage Estimation:
Assume each booking (Seat IDs [ ], Show ID, Movie ID, Timestamp, User ID) ⇒ 100 bytes of storage
Movies & Cinema data will take another 100 bytes.
Single day storage estimate:
1000 cities * 10 cinemas * 1000 seats * 3 shows * (100 +100) bytes
= 6 GB / day

→ Network Bandwidth Estimation:
- → Traffic Management
- → Load Balancing

4. Data Model / DB Design

→ Clarifies how data will flow in System.
→ Helps in data management, sharding, partitioning.
→ Identify the data entities
→ Identify relationships between them
→ Advanced aspects can include.
1) Storage
2) Transportation
3) Compression
4) Encryption / Decryption

Some entities for BookMyShow-like service:

User : UserID, Name, Password, Email, Phone, City

Movie: MovieID, Title, Summary, Release Date, Language, Genre

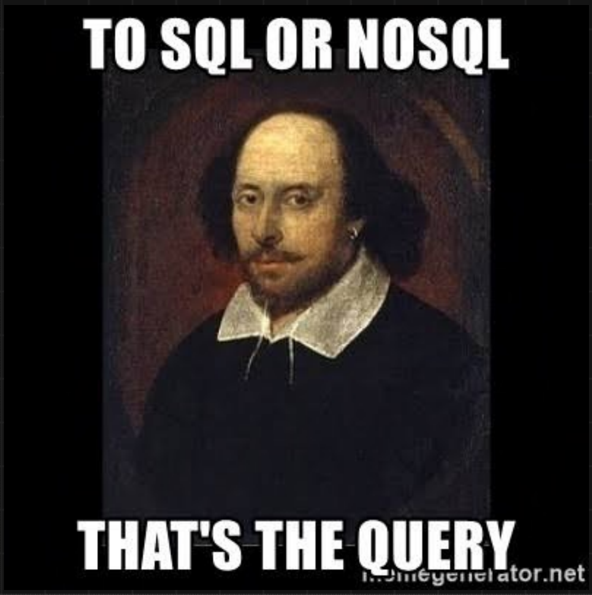
Booking : Booking ID, Seats, Timestamp, Status, User ID, Show ID

Show : Show ID, Date, Start Time, End-Time, Movie ID, Theatre

Which database should we use?

Would NoSQL like MongoDB best fits our requirements, or we should use MySQL-like Solution. (Do we need block storage for storing pictures / trailer videos).

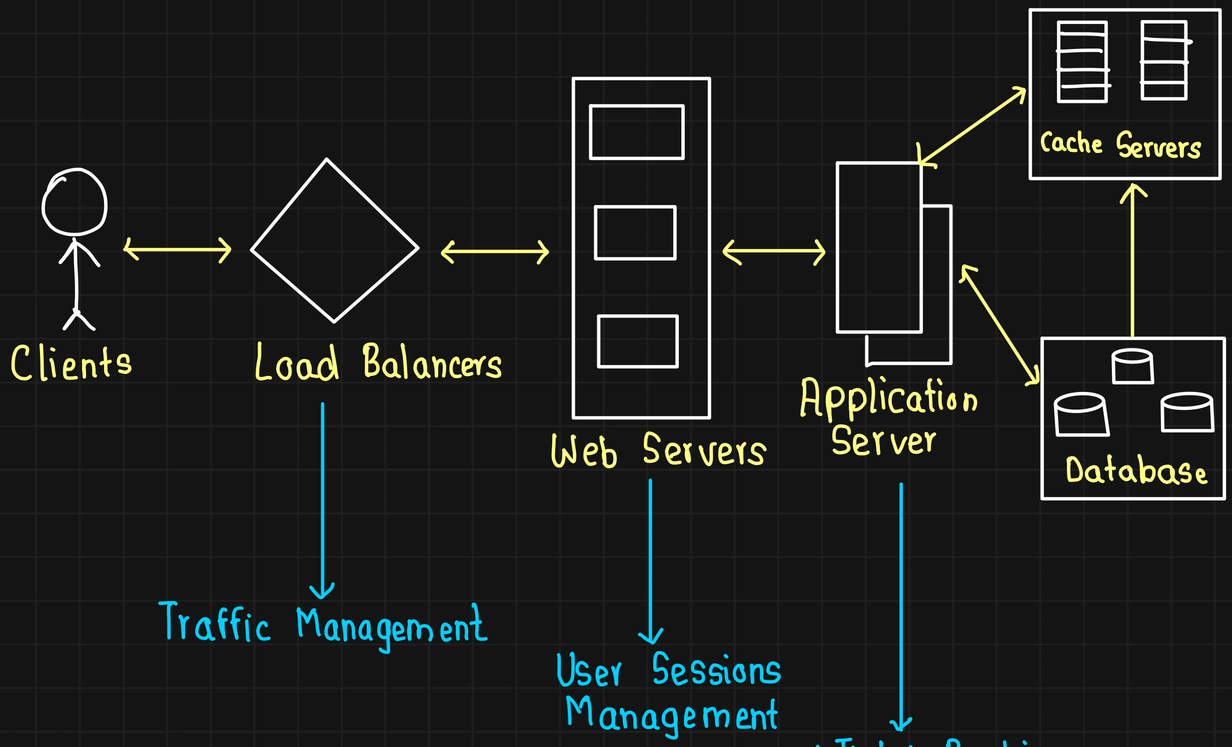
→ Discuss pros & cons along with your requirements in deciding your database.



5. High - level Design (HLD)

→ Try to draw block diagram of core components of the system.
→ Try to identify each and every components that are needed to solve the use cases defined in first step.

At a high level,



→ Ticket Bookings
→ Data storage in DB
→ Caching the data in cache servers to process reservations.

6. Detailed Component Design	→ Ask interviewer for which components you should dig deeper.

→ Dig deeper into 2-3 components only.

→ Provide different approaches.
(Discuss pros & cons)
(Consider all tradeoffs & system constraints & choose one)

For our use case,
you can discuss Reservation Workflow
→ Activity Diagram
→ Data flow diagram

→ We'll be storing tons of data,
how should we handle DB distribution?
1) Raise a question
2) Come up with a solution.
3) Discuss the issues.

→ How much and at which layer we should use caching?

→ What components needs better load balancing?

→ How are we gonna track all the active reservations that haven't completed the payment yet?

→ How are we gonna keep track of and serve the waiting customers?

→ How would we handle trending / blockbuster movies bookings?

7. Find out bottlenecks & mitigate them

→ Try to discuss as many tradeoffs, bottlenecks as possible.

→ Discuss different approaches to minimize / remove them.

Things that you can discuss:

1) Single point of failures & their mitigation

2) Data replicated or NOT? if we lost any server?

3) Do we have enough instances of microservices running such that few failures should not cause total breakdown.

4) Performance monitoring

5) System health checks

6) How to handle concurrency? (Multiple users trying to book same seat.)

We can use SQL transactions
(with isolation level Serializable)

↖ highest isolation level amongst
→ Read uncommitted
→ Read committed
→ Repeatable read
→ Serializable

guarantees safety from:
→ Dirty reads
→ Non-repeatable reads
→ Phantom reads.

7) Fault tolerance

what if our active reservation system fails?

↳ How are we gonna retrieve all active reservations?

↳ Fetch users from Booking table with Status Reserved (not Booked).

OR

we can have master-slave configuration to make it fault tolerant.