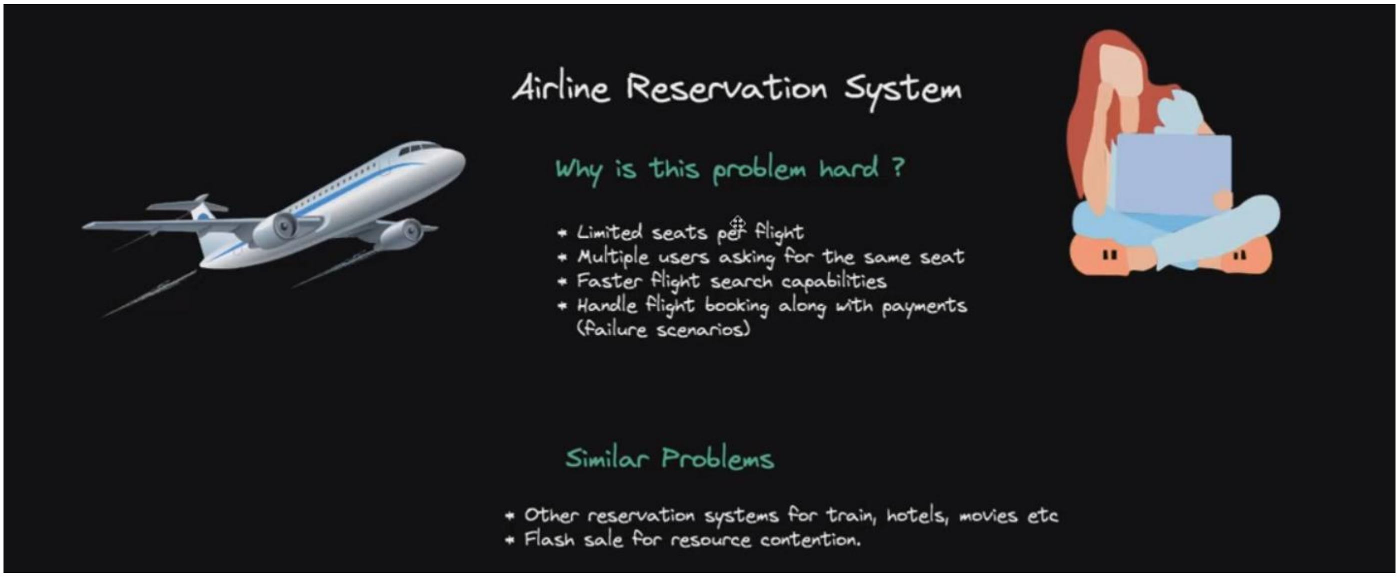


Airline Reservation System

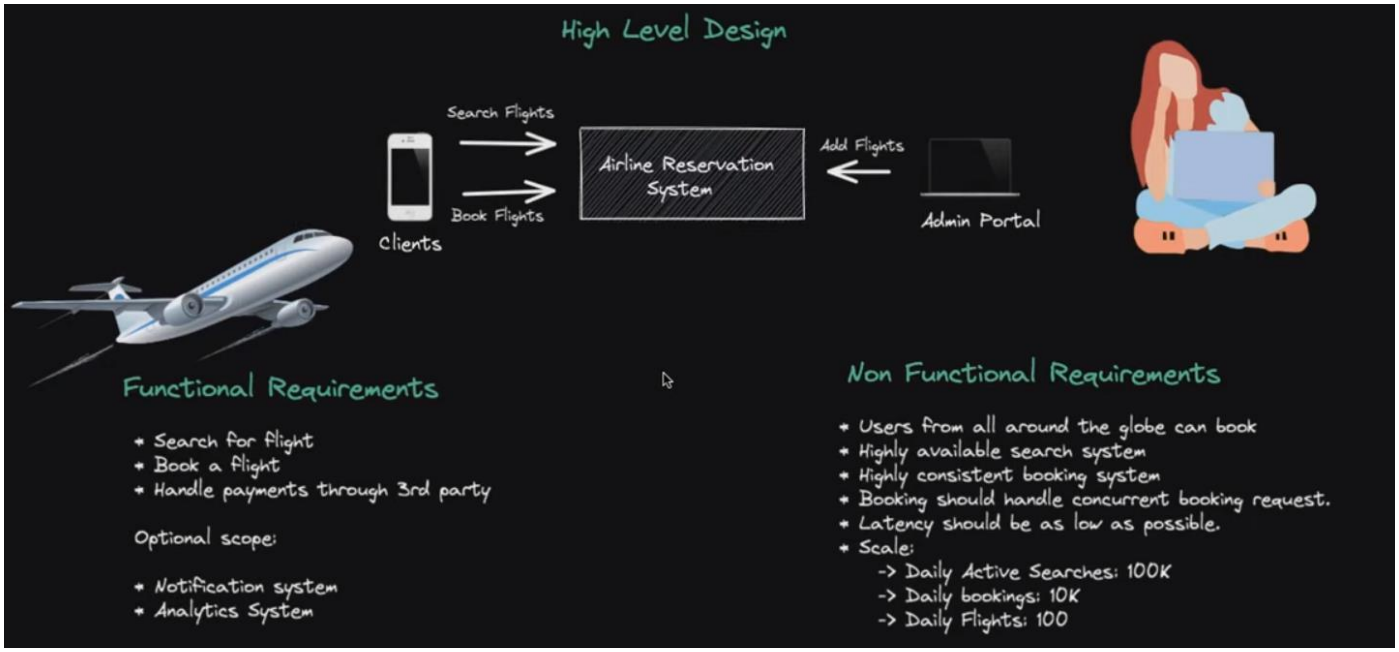
Why is this problem hard ?

- Limited seats per flight
- Multiple users asking for the same seat
- Faster flight search capabilities
- Handle flight booking along with payments (failure scenarios)

Similar Problems

- Other reservation systems for train, hotels, movies etc
- Flash sale for resource contention.

Suggestions - for flight information table, we have primary key as flight id but most relational databases create unique index on primary key across partitions so we shall better than flight id and departure date as composite primary key. Though I like the idea of departure date as partition key. Here, we are assuming that if same plane does multiple round trips between two destination, each will have a separate flight id on given day. Also, instead of booking open, we can call that column as booking status - instead of just open and close, there could be more status like flight cancelled.



High Level Design

Functional Requirements

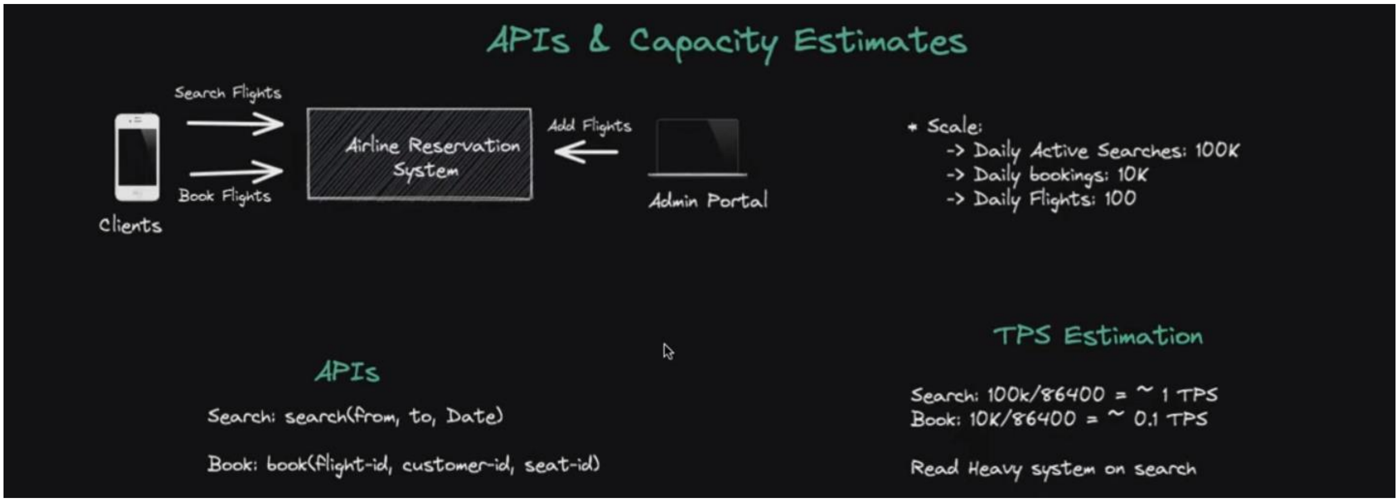
- Search for flight
- Book a flight
- Handle payments through 3rd party

Optional scope:

- Notification system
- Analytics System

Non Functional Requirements

- Users from all around the globe can book
- Highly available search system
- Highly consistent booking system
- Booking should handle concurrent booking request.
- Latency should be as low as possible.
- Scale:
  - -> Daily Active Searches: 100K
  - -> Daily bookings: 10K
  - -> Daily Flights: 100



APIs & Capacity Estimates

* Scale:
  -> Daily Active Searches: 100K
  -> Daily bookings: 10K
  -> Daily Flights: 100

APIs

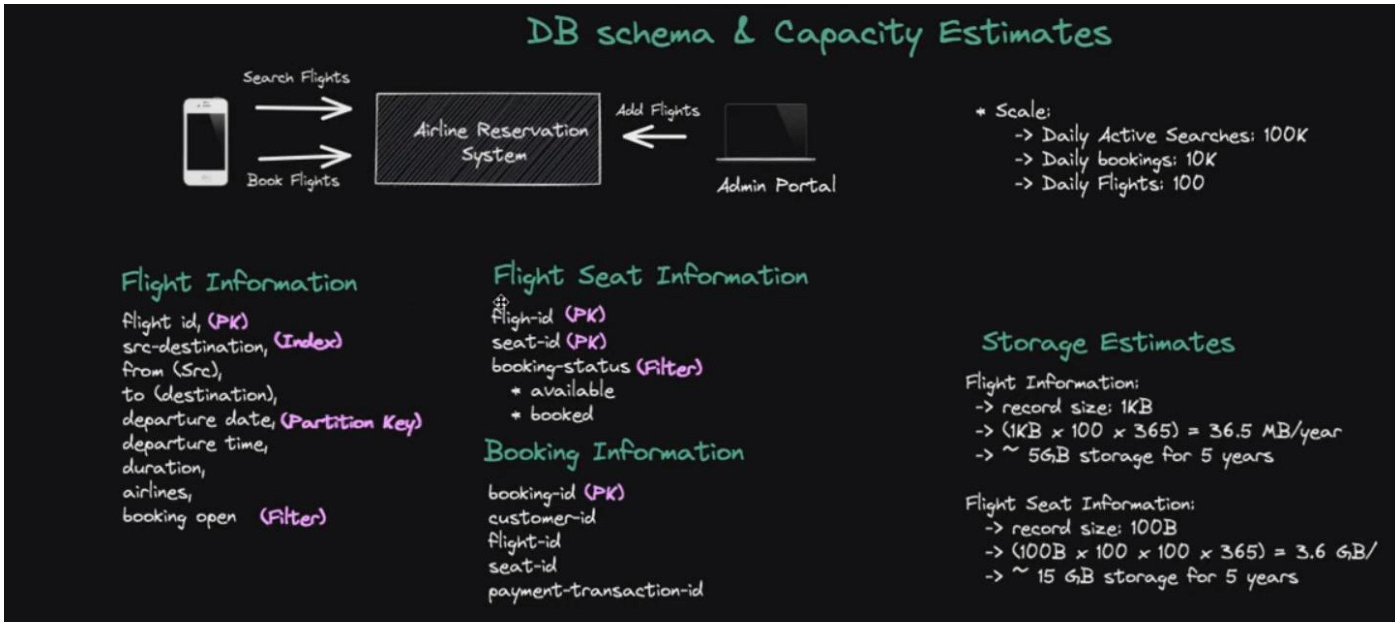
Search: search(from, to, Date)

Book: book(flight-id, customer-id, seat-id)

TPS Estimation

Search: 100k/86400 = ~ 1 TPS
Book: 10K/86400 = ~ 0.1 TPS

Read Heavy system on search



DB schema & Capacity Estimates

* Scale:
  -> Daily Active Searches: 100K
  -> Daily bookings: 10K
  -> Daily Flights: 100

Flight Information

flight id, (PK)
src-destination, (Index)
from (Src),
to (destination),
departure date, (Partition Key)
departure time,
duration,
airlines,
booking open (Filter)

Flight Seat Information

fligh-id (PK)
seat-id (PK)
booking-status (Filter)
* available
* booked

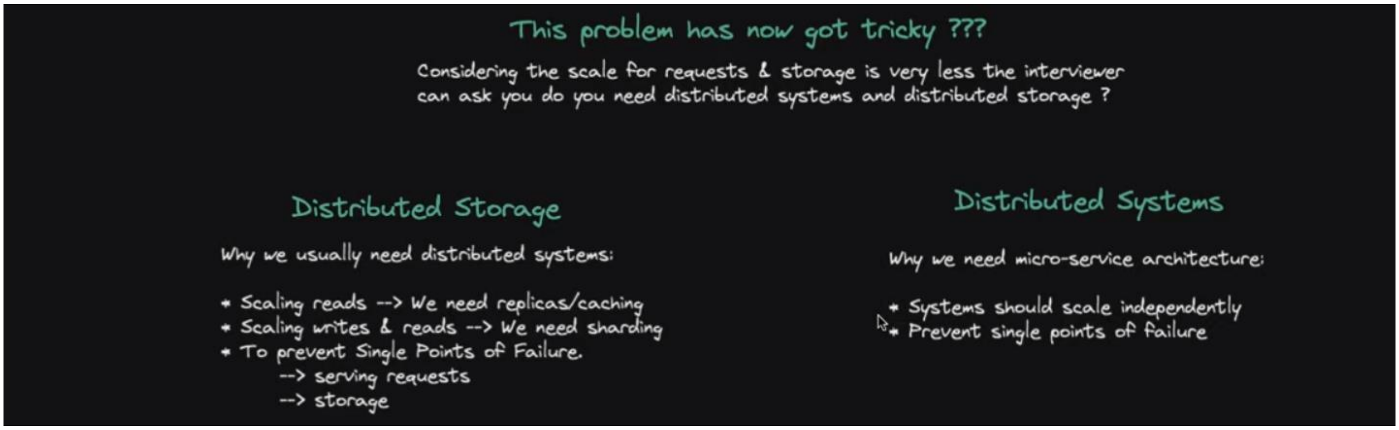
Booking Information

booking-id (PK)
customer-id
flight-id
seat-id
payment-transaction-id

Storage Estimates

Flight Information:
-> record size: 1KB
-> (1KB x 100 x 365) = 36.5 MB/year
-> ~ 5GB storage for 5 years

Flight Seat Information:
-> record size: 100B
-> (100B x 100 x 100 x 365) = 3.6 GB/
-> ~ 15 GB storage for 5 years



This problem has now got tricky ???

Considering the scale for requests & storage is very less the interviewer
can ask you do you need distributed systems and distributed storage ?

Distributed Storage

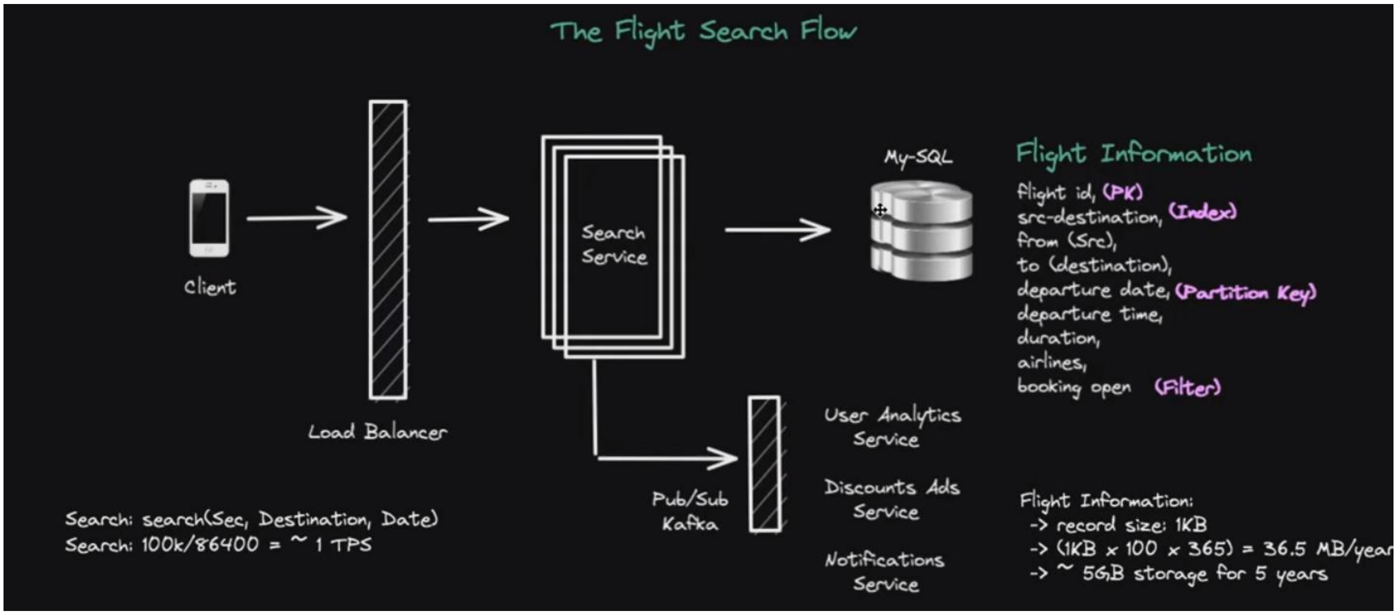
Why we usually need distributed systems:

* Scaling reads --> We need replicas/caching
* Scaling writes & reads --> We need sharding
* To prevent Single Points of Failure.
  --> serving requests
  --> storage

Distributed Systems

Why we need micro-service architecture:

* Systems should scale independently
* Prevent single points of failure

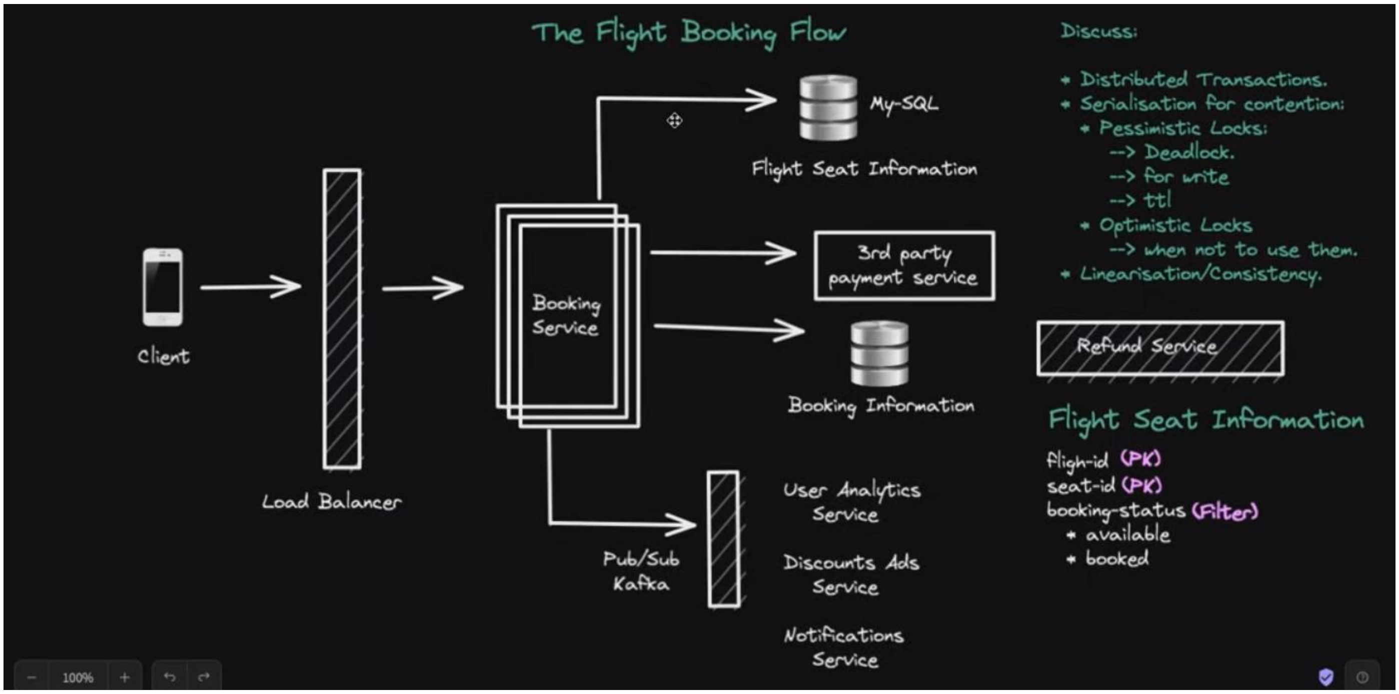

The Flight Search Flow

Flight Information

flight id, (PK)
src-destination, (Index)
from (Src),
to (destination),
departure date, (Partition Key)
departure time,
duration,
airlines,
booking open (Filter)

Search: search(Sec, Destination, Date)
Search: 100k/86400 = ~ 1 TPS

Flight Information:
-> record size: 1KB
-> (1KB x 100 x 365) = 36.5 MB/year
-> ~ 5GB storage for 5 years


The Flight Booking Flow

Discuss:

- Distributed Transactions.
- Serialisation for contention:
  - Pessimistic Locks:
    - --> Deadlock.
    - --> for write
    - --> ttl
  - Optimistic Locks
    - --> when not to use them.
- Linearisation/Consistency.

Flight Seat Information

fligh-id (PK)
seat-id (PK)
booking-status (Filter)
- available
- booked