(Video 37. Google Calendar System Design)

(Functional Requirements)

① Schedule ~~not~~ meeting

② Check availability of users

③ request RSVP → { Request Response for Users }

④ Meeting reminders → { Get notification 30 mins before meeting }

(Non-Functional Requirements)

① Available

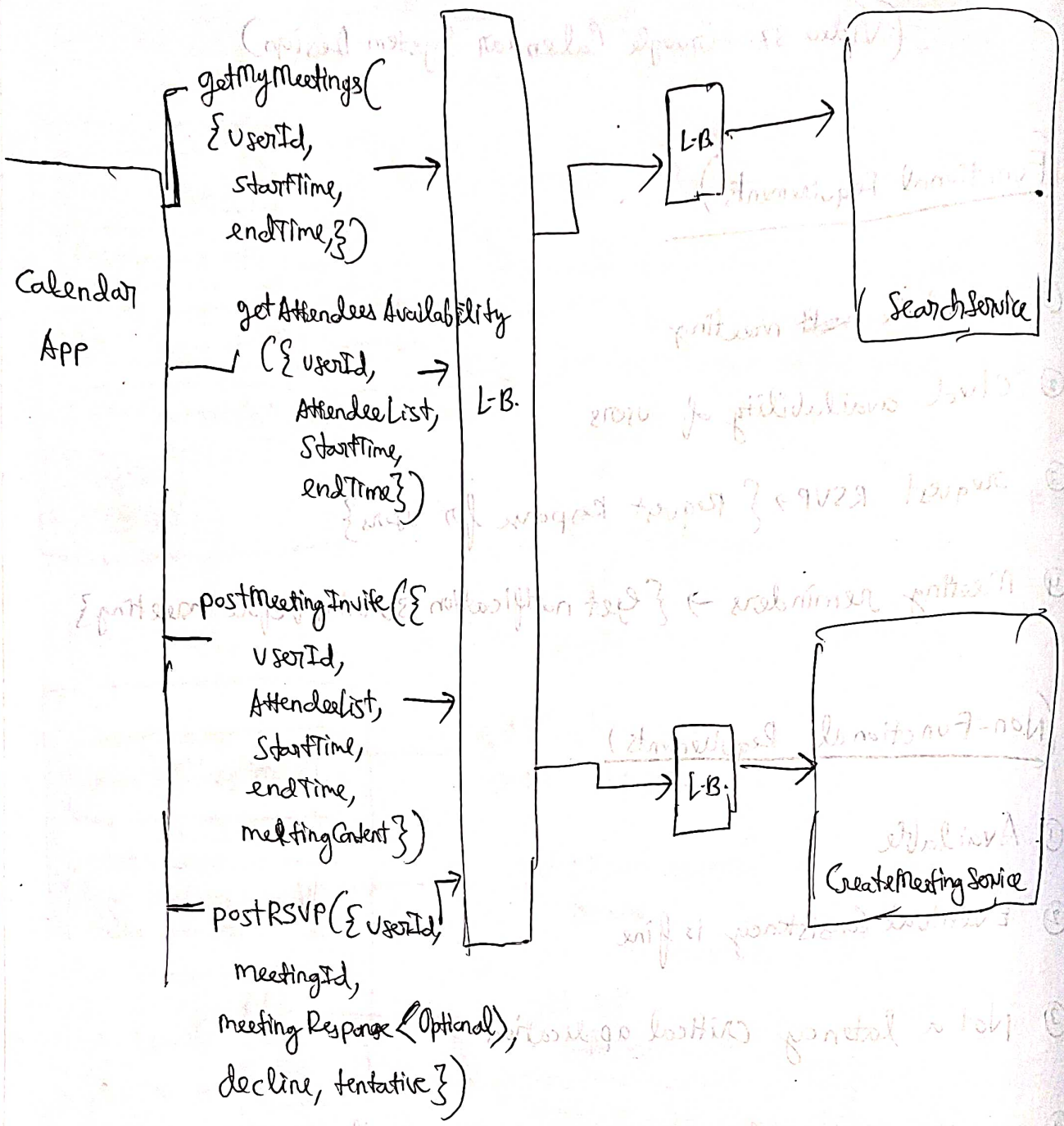
② Eventual Consistency is fine

③ Not a latency critical application

④ Meetings should be persisted unless manually deleted → Durable

(Capacity Estimations)

* Lets say number of users = 100M
* Out of all users $1/10^{th}$ of them post invites = $1/10 \times 100 = 10M$
* Each user makes 5 posts per day = $5 \times 10M = 50M$ invites per day.
* Approx Storage for 1 invite = { meeting content, userId, invitees, time } → 1KB
* Total Storage for a year = $1KB \times 50M \times 365 = \sim 20$ TB/year.



We have exposed 4 APIs

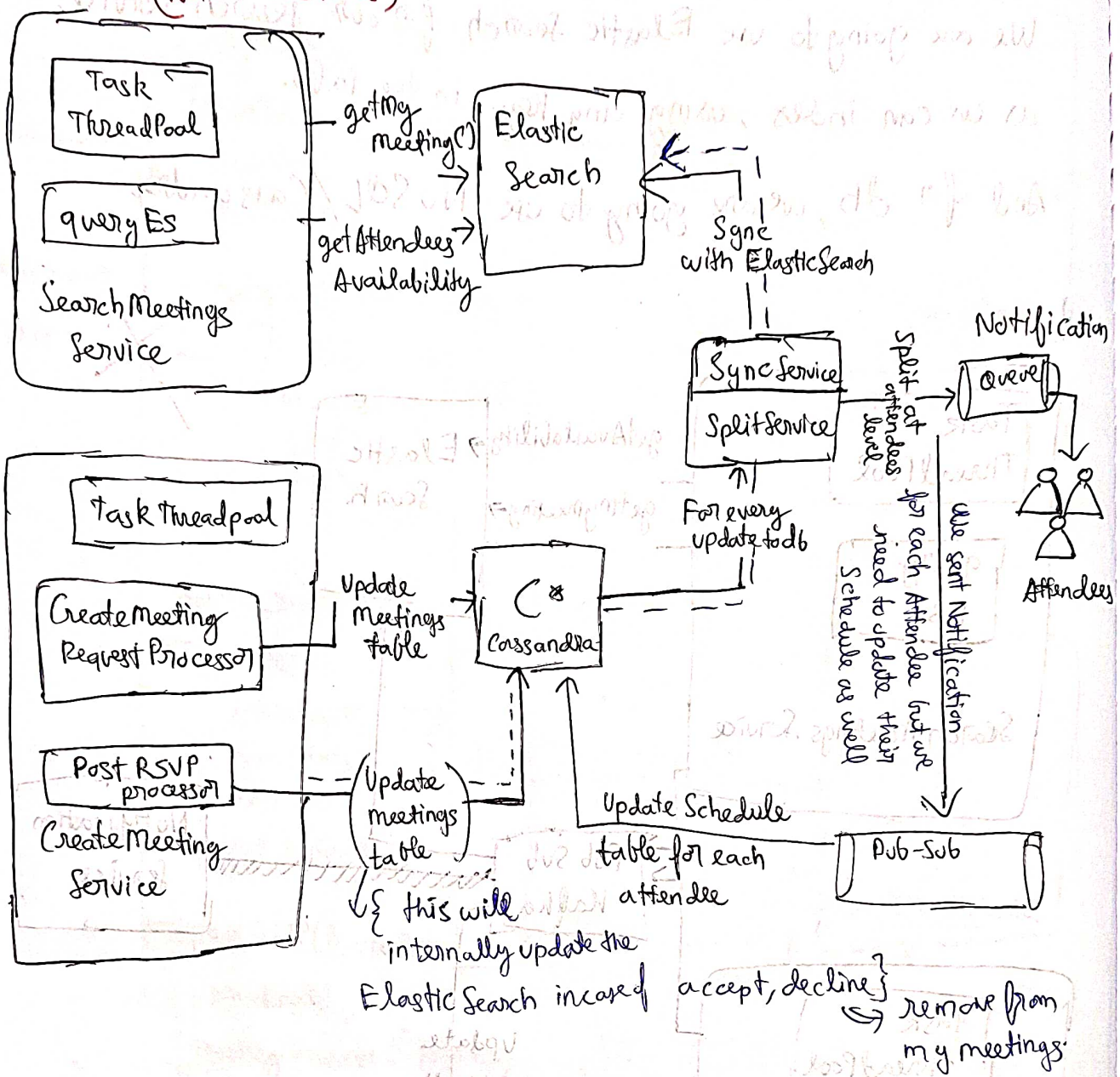
1) getMyMeetingsAPI → For any given userId it will fetch the meeting details.

2) getAttendeesAvailability → Pass the attendee list and given a start and endtime get Availability

3) postMeetingInvite → Send Invite to attendee List

4) postRSVP → take userId and meetingId and send response

(Architecture)



(High Level Design)

How our schema will look like will be decided by the query patterns.

Query Patterns :-

① Get meetings for a user in a given timestamp

② Get availability for a list of user in given timestamp.

③ Check status of meeting → who accepted / declined.

{Meetings Table}

partition Key - meetingId (uuid)

clustering Key - userId

Attendees - List {userId : status}

StartTime

EndTime

meeting Content

{Schedule Table}

partition key - scheduleId (uuid)

Clustering key - userid + meetingId

meetingId

StartTime

endTime

(Why Elastic Search)

① Make Search Faster as Cassandra don't have primary key as all the attributes, and as per the query pattern we saw, we will require to query and index on each attribute of table.

This calander app could be created without Elastic search as well if we did not need the search capability

(Points to Consider)

* Incase of duplicate meetings we leave the decision to attendee to accept/decline

* Is PostMeeting Invite() completely Sync?

Since this API takes significant time to update-DB and then update Elastic Search + posting Notifications,

The sync API can handle till updating it to Cassandra and send Ack and rest can be done in background.

* Why NoSql is ~~choosen~~ chosen here over Relational Database?

  * ACID is not a requirement
  * there are relationships in database but not that strong that require table joins
  * Priority of availability over strong consistency

* Handling Recurring Invite? For each meeting-invite, per userId entry is created, So it will be handled.

* API getAttendeesAvailability() can be called in background before PostmeetingInvite() API → will help user to check availability of users before sending invite

* Reminder Service → Can be implemented at client side, to poll meetings for every 30 mins to store data