DESIGN A CHAT SYSTEM

Clarifying questions -

1. What kind of chat app shall we design ? 1 on 1 or group based ?
   Both
2. Mobile app or group?
   Both
3. Scale of the app?
   50 million DAU (Daily active users)
4. For group chat, what is the group member limit ?
   maximum of 100 people
5. What features are important for this chat app ? Can it support attachment ?
   1 on 1 chat, group chat, online indicator. Only supports text messages
6. Is there a message size limit ?
   text length should be less than 1,00,000 characters long
7. How long should we store chat-history ?
   Forever

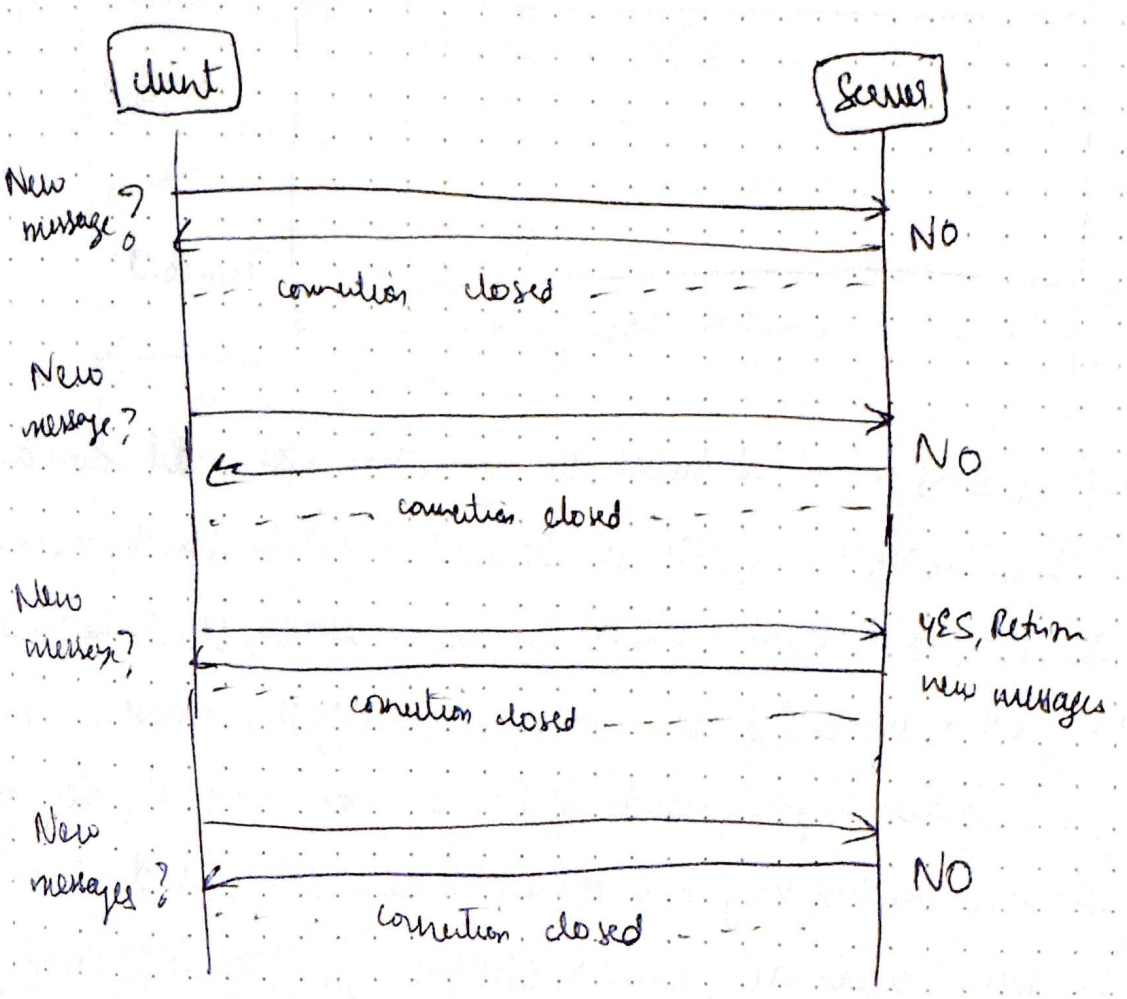
→ Receive messages in real time
→ low latency

~~Capacity Estimates~~

Before jumping to the capacity estimates lets learn about

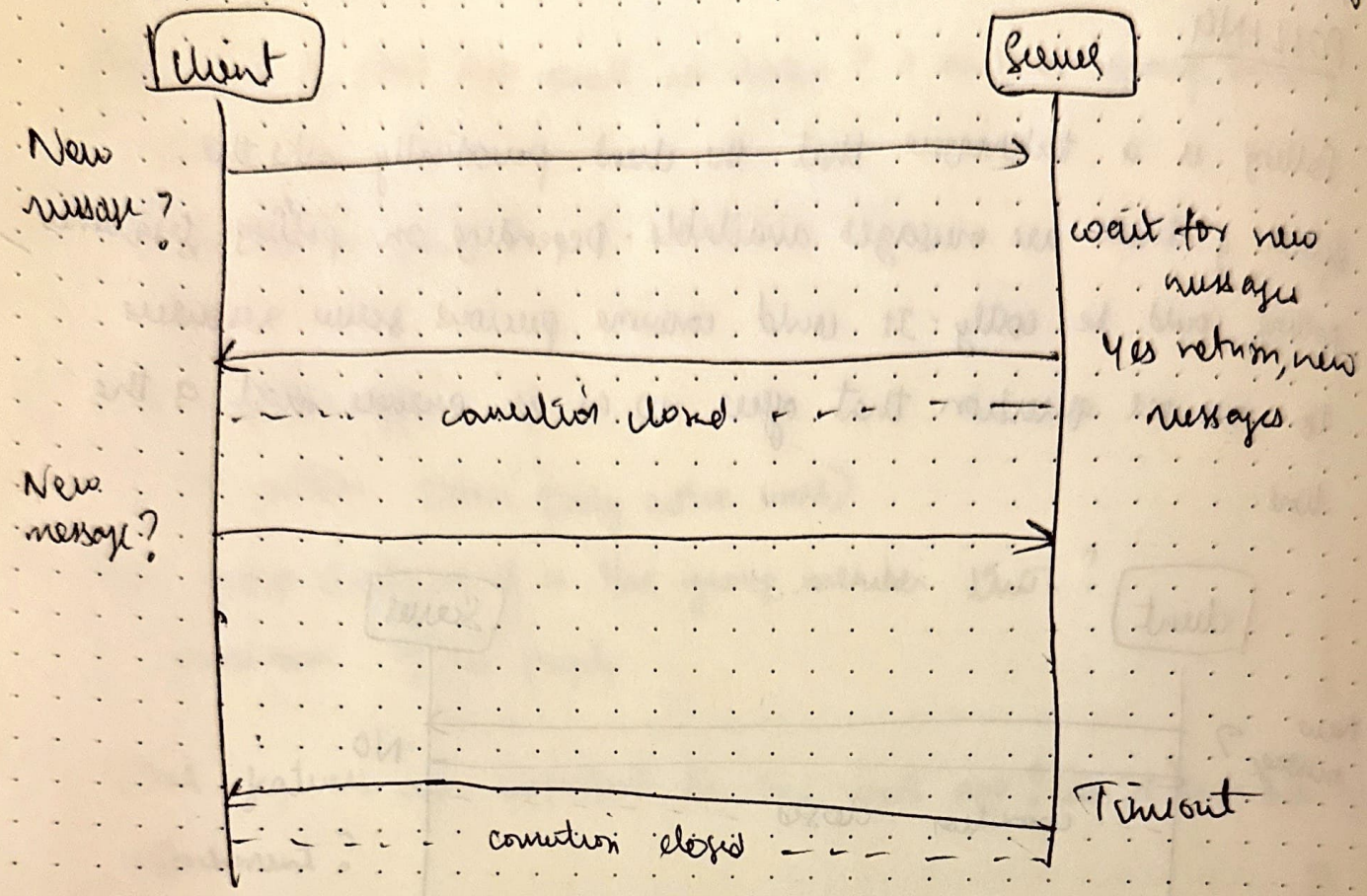
1. POLLING

Polling is a technique that the client periodically asks the server if there are messages available. Depending on polling frequency, polling could be costly. It could consume precious server resources to answer question that offers no as an answer most of the time



LONG POLLING

Because polling could be inefficient, the next progression is long polling



In long polling, a client holds the connection open until there are actually new messages available or a timeout threshold has been reached. Once the client receives the new messages, it immediately sends another request to the server, restarting the process.

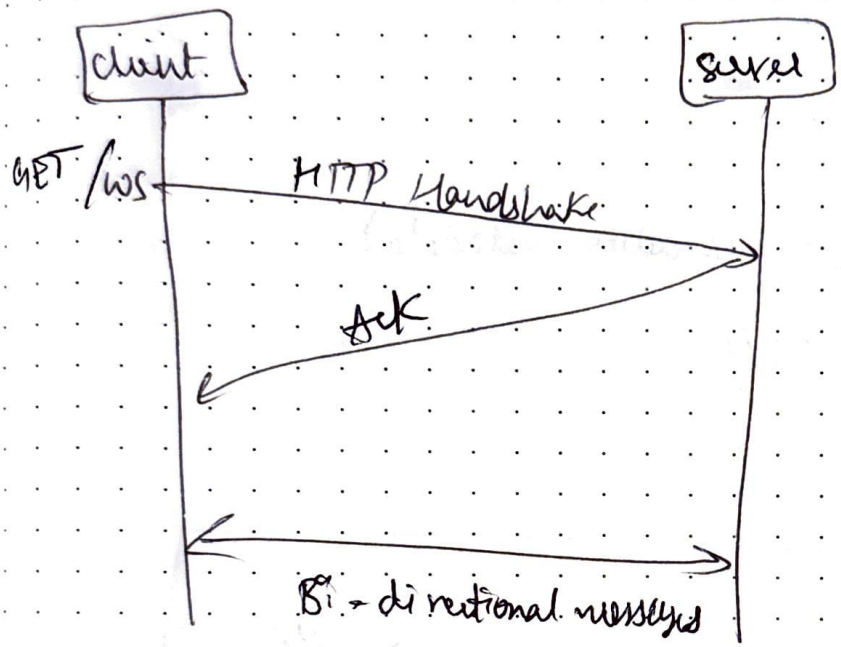
Long polling has a few drawbacks:

→ Sender and receiver may not connect to the same chat server. HTTP based servers are usually stateless. If you use round robin for load balancing, the ~~next~~ server that receives the message might not have a long-polling connection with the client who receives the message.

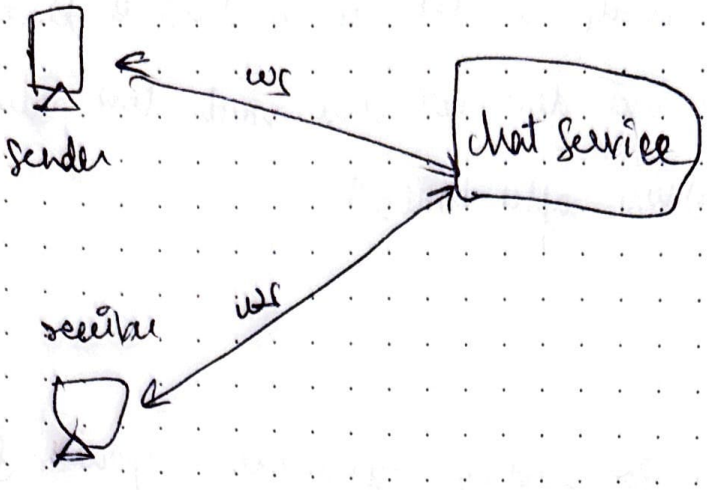
→ A server has no good ways to tell if a client is disconnected.
→ It is inefficient if a user does not chat much, long polling still makes periodic connections after timeouts.

WEB SOCKET

Most common solution for sending asynchronous updates from servers to client.

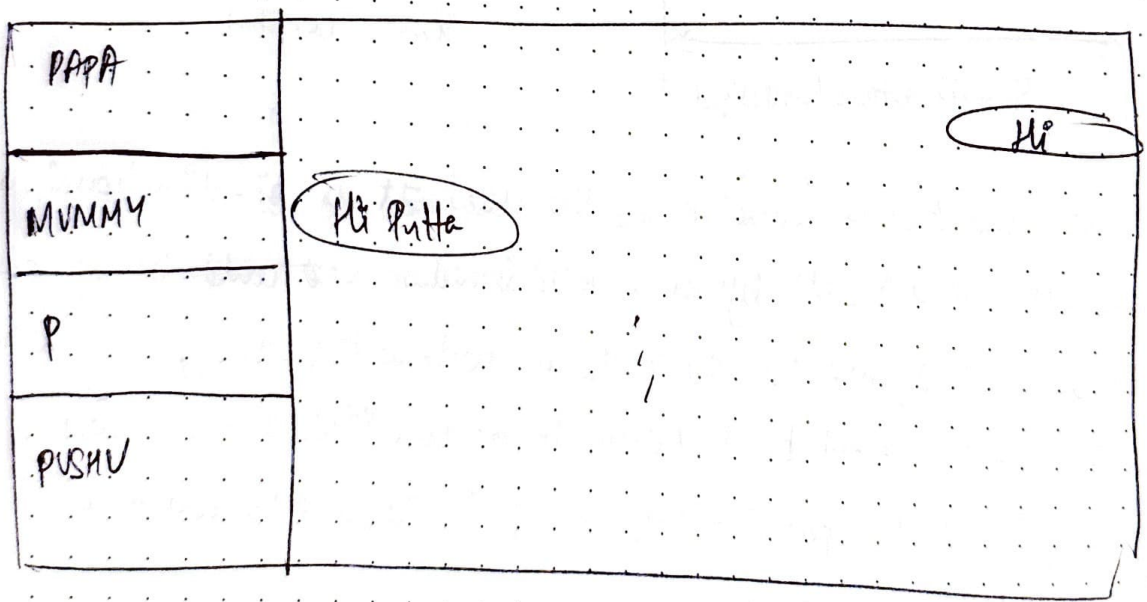


Web-socket connection is initiated by the client. It is bi-directional and persistent. It starts its life as a HTTP connection and could be upgraded via some well-defined handshake to a web socket. Through persistent connection, a server could send updates to a client. WebSockets work even if firewall is present. They use port 80 or 443 which is by HTTP / HTTPS connections.



Capacity estimates

1. Say 1 billion users a day
2. Each user sends 100 messages a day
3. Each message is around 100 bytes (including metadata)
4. 1 billion x 100 x 100 bytes = 10 TB/day
   ~ 4 PB per year



Chat members database

→ We should be able to fetch all chats for a given user

→ Left pane of the diagram

→ Want all chatIds for a given user to be on the same chart,

userId	chatId
3	12
3	14
10	12
10	85

Let's just use a MySQL

Leaderless replica

Index + Partition Key

Users database

schema → userId, email, passwordHash

Partition on UserId

Lot of rows but, reads/writes less than messages table

so maybe MySQL

Messages table

We want to make sure that getting all the messages for a given chat is fast.

Schema: chatId (PK), timestamp (sort key), message, metadata

→ Messages for chat already on same partition and pre-sorted

→ Timestamps ensure consistent ordering on all devices (Order by TS on frontend)

↳ All messages of a chat stay together

Messages database

What should we optimize for?

Reads → B+ trees, index, column oriented storage? (Hbase)

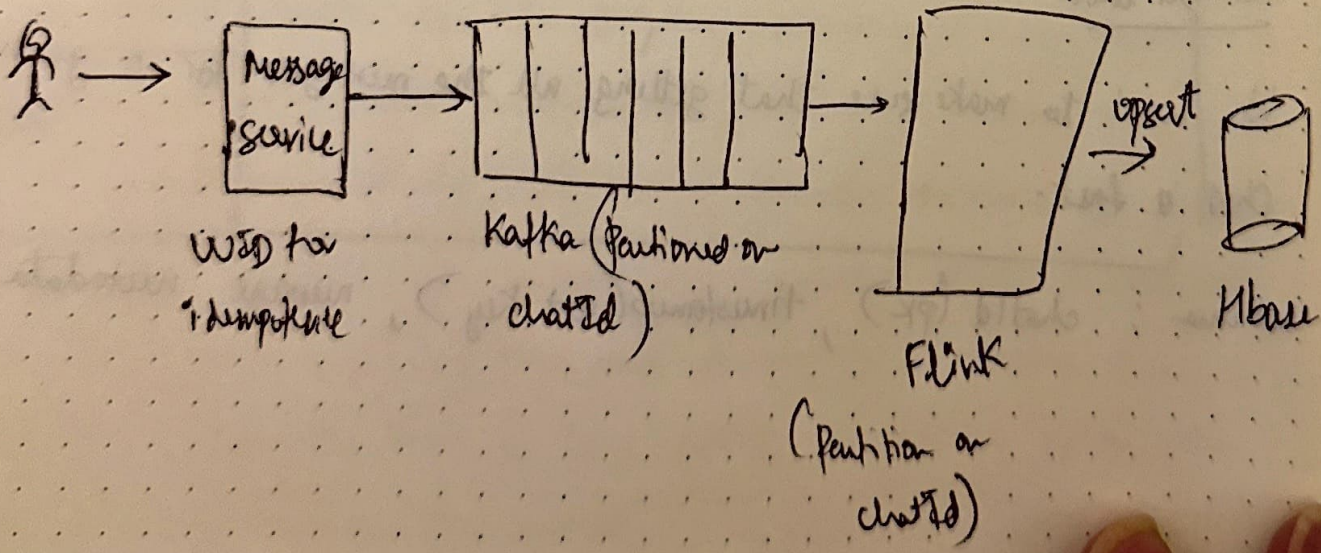
chatId, time, message, sender etc. ✓	metadata X

Writes → LSM tree index, leaderless replication

↳ Cassandra?

Message Delivery

It would be great if to be able to read messages from Hbase, while being able to quickly ingest them in our system!



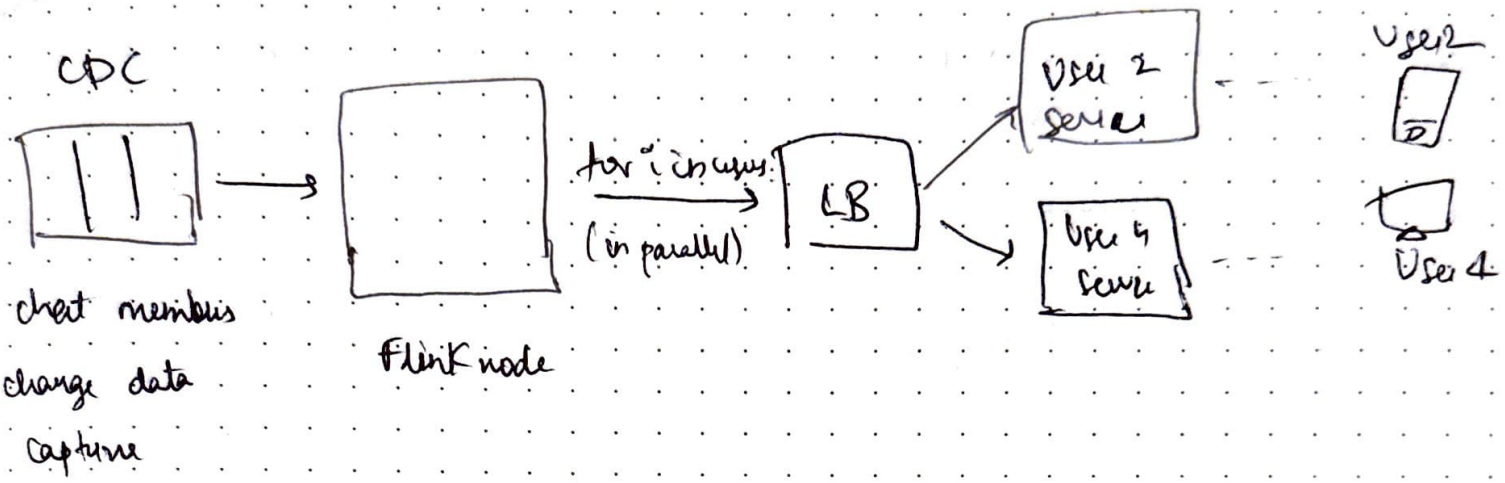
Kafka + Flink → ensures every message is processed once.

Message Routing

To minimize load on client devices we want them to maintain as few active connections as possible.

→ Each user can just maintain a connection with one chat server

→ chats at most have to be sent to 10 different places

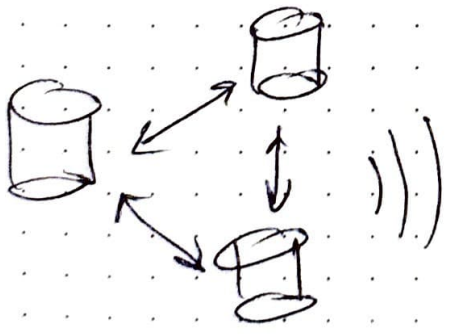


Connection to Chat Server

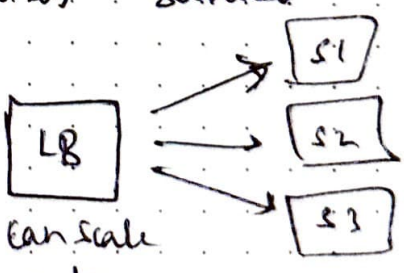
We need to reliably know the function to connect a given userId to a server for:

→ sending messages
→ receiving messages

} allows us to use bi-directional real-time conversation!



f(userId) = serverId

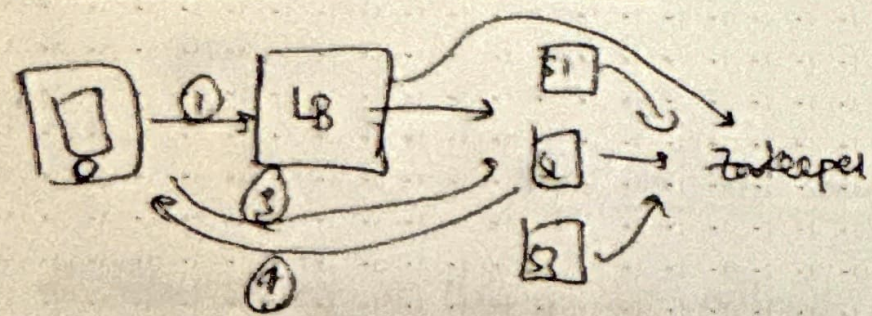


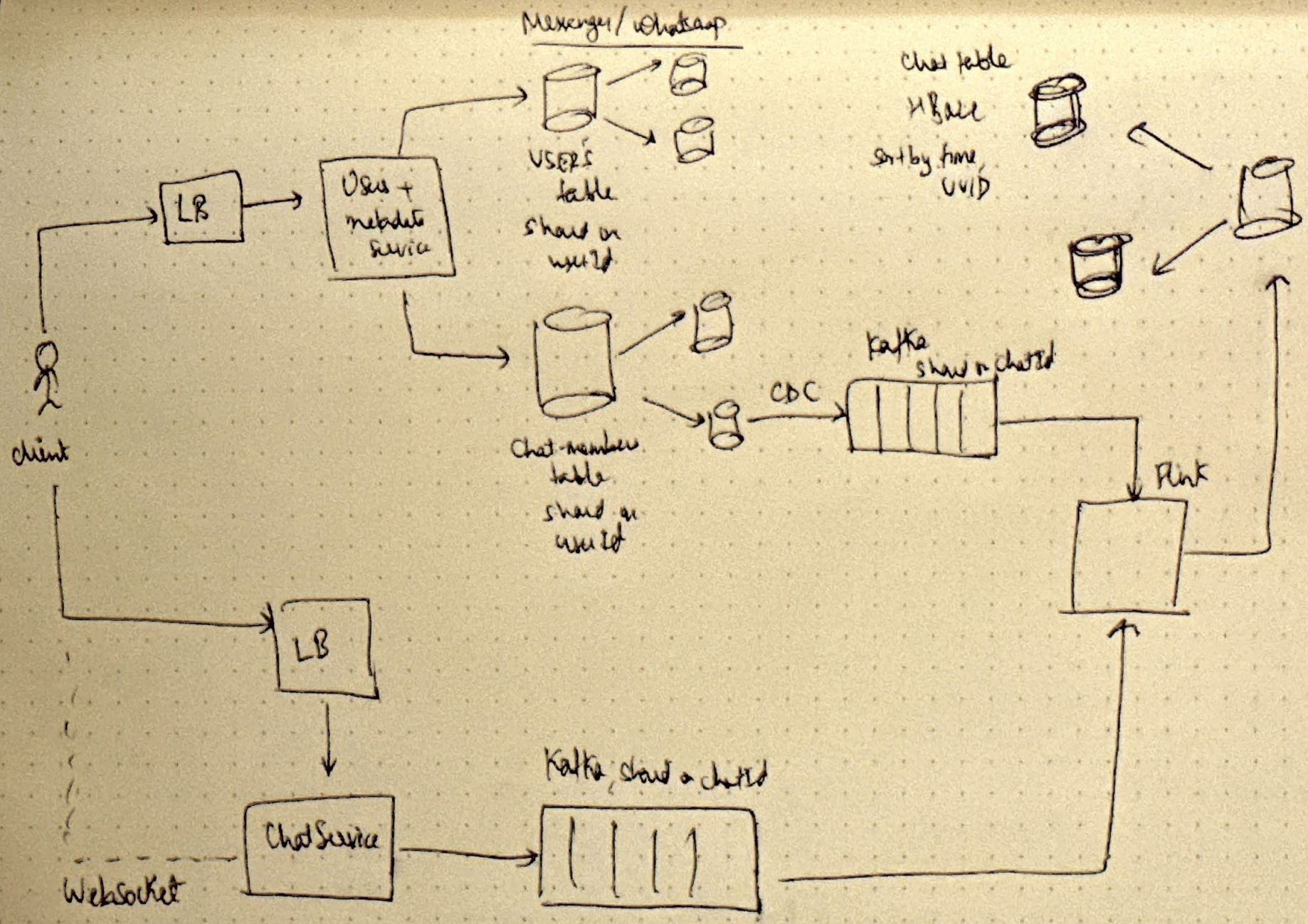
Apache Zookeeper

Distributed coordination service that manages
- Kafka brokers
- Leader election for partitions
- Metadata storage (which broker owns which partition)
- Fault tolerance

Why does Kafka need zookeeper? → Kafka is a distributed system with multiple brokers. Zookeeper ensure brokers communicate properly and recover from failures.

Connections to Chat Server




Messenger / Whatsapp
LB
User + metadata service
USER'S table shard on userId
Chat table HBase sort by time, UUID
client
Chat-member table shard on userId
CDC
Kafka shard on chatId
Flink
LB
ChatService
WebSocket
Kafka, shard on chatId