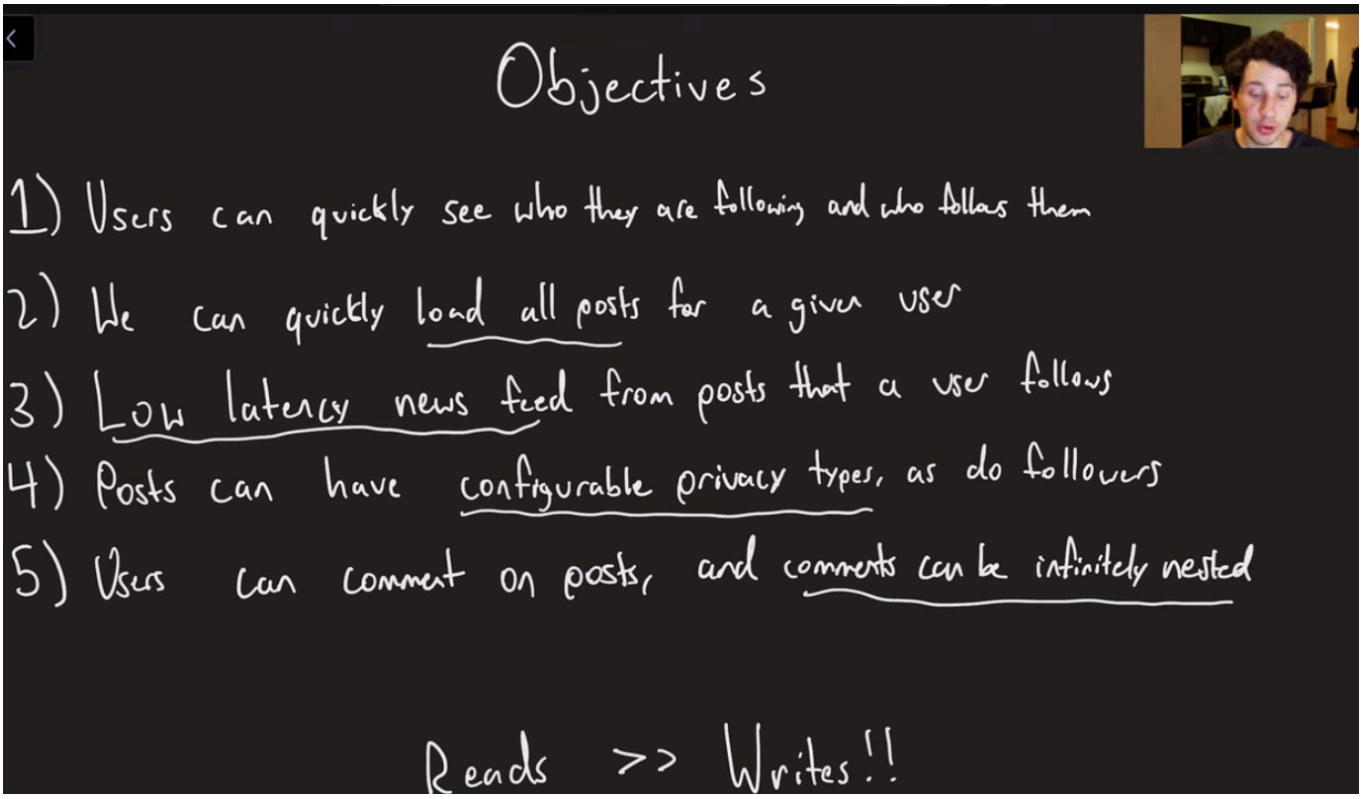
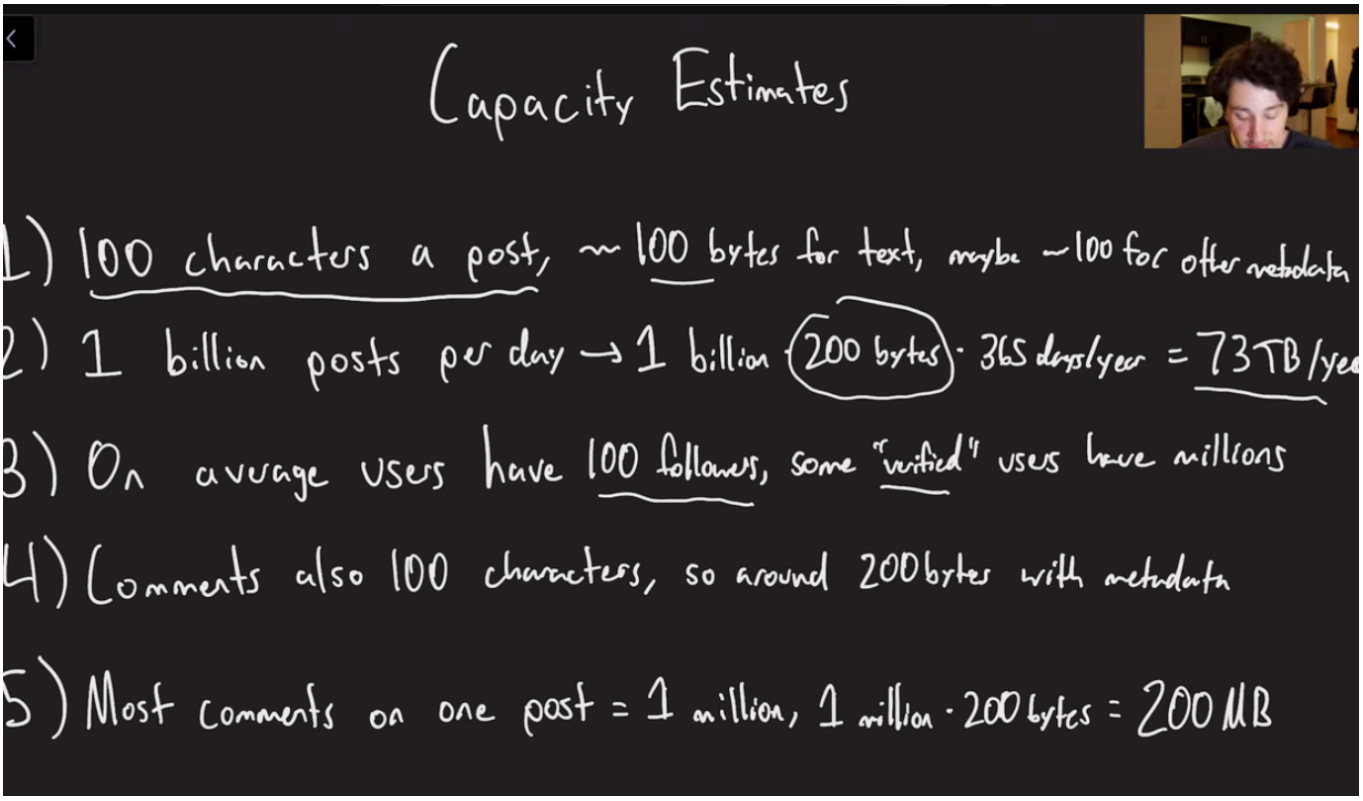
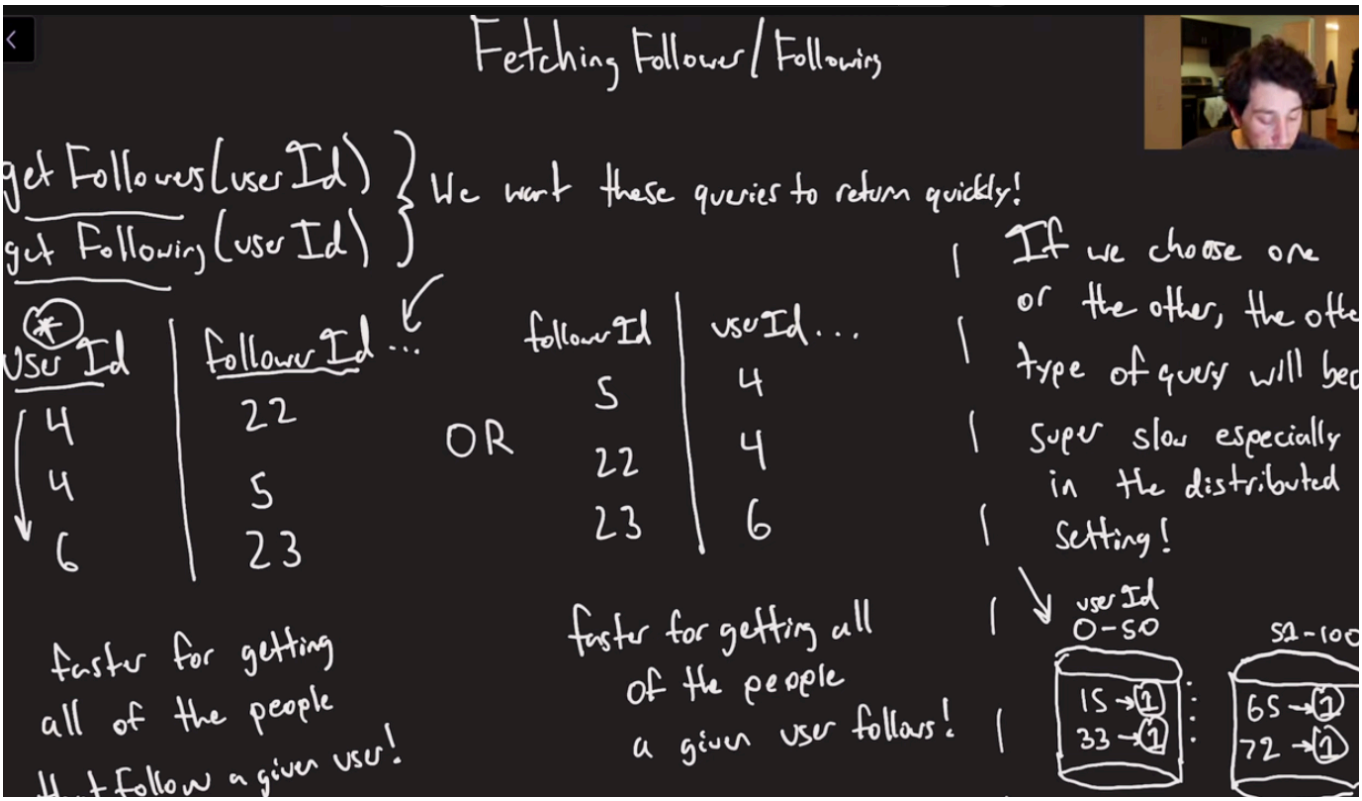
Instagram + Twitter + Facebook + Reddit

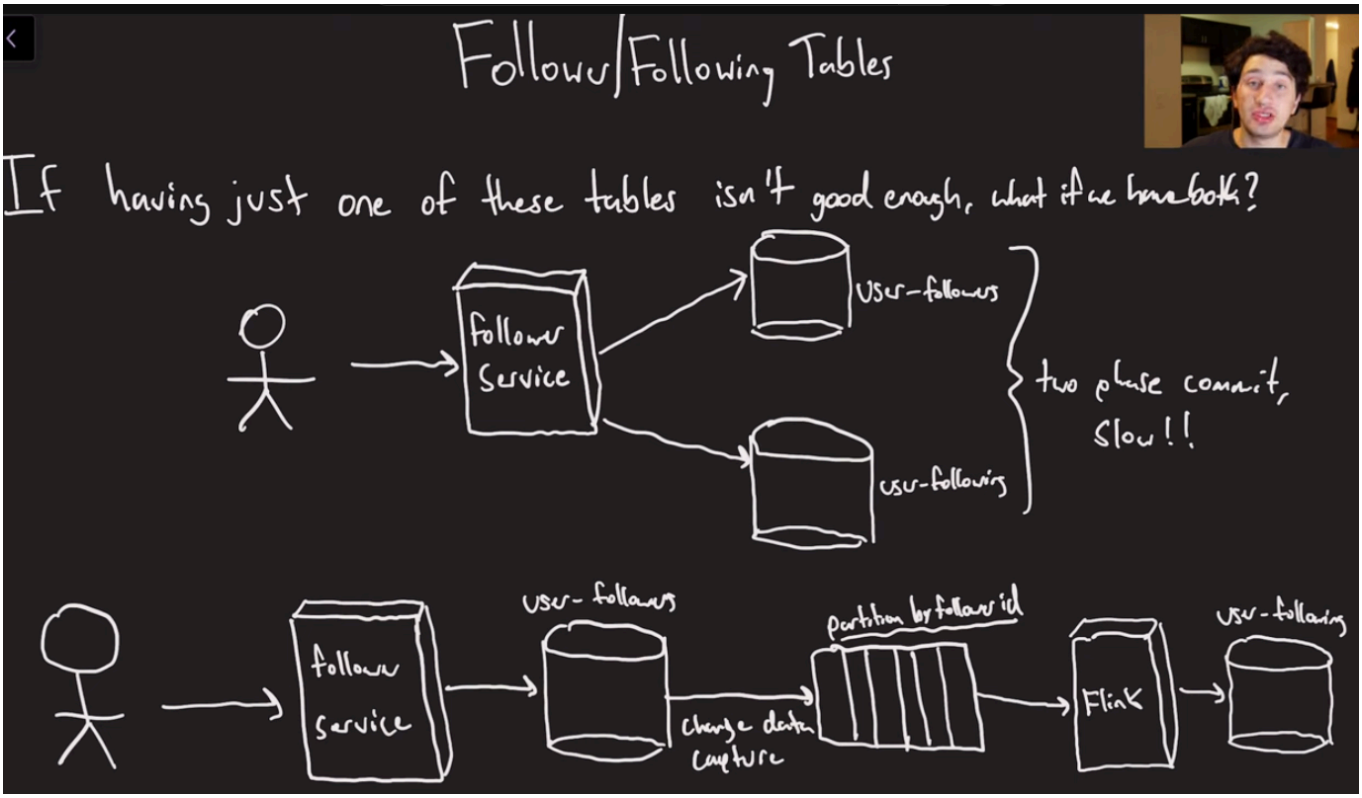


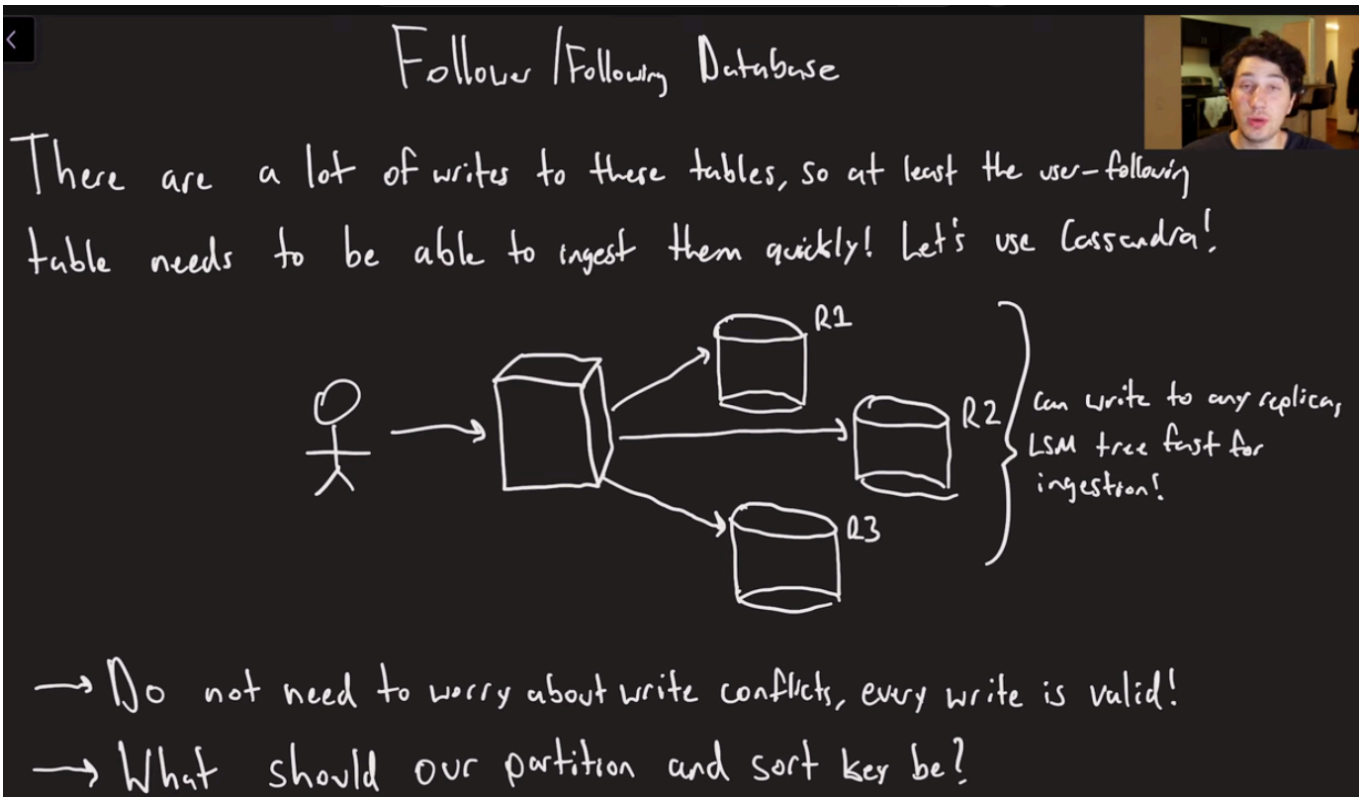


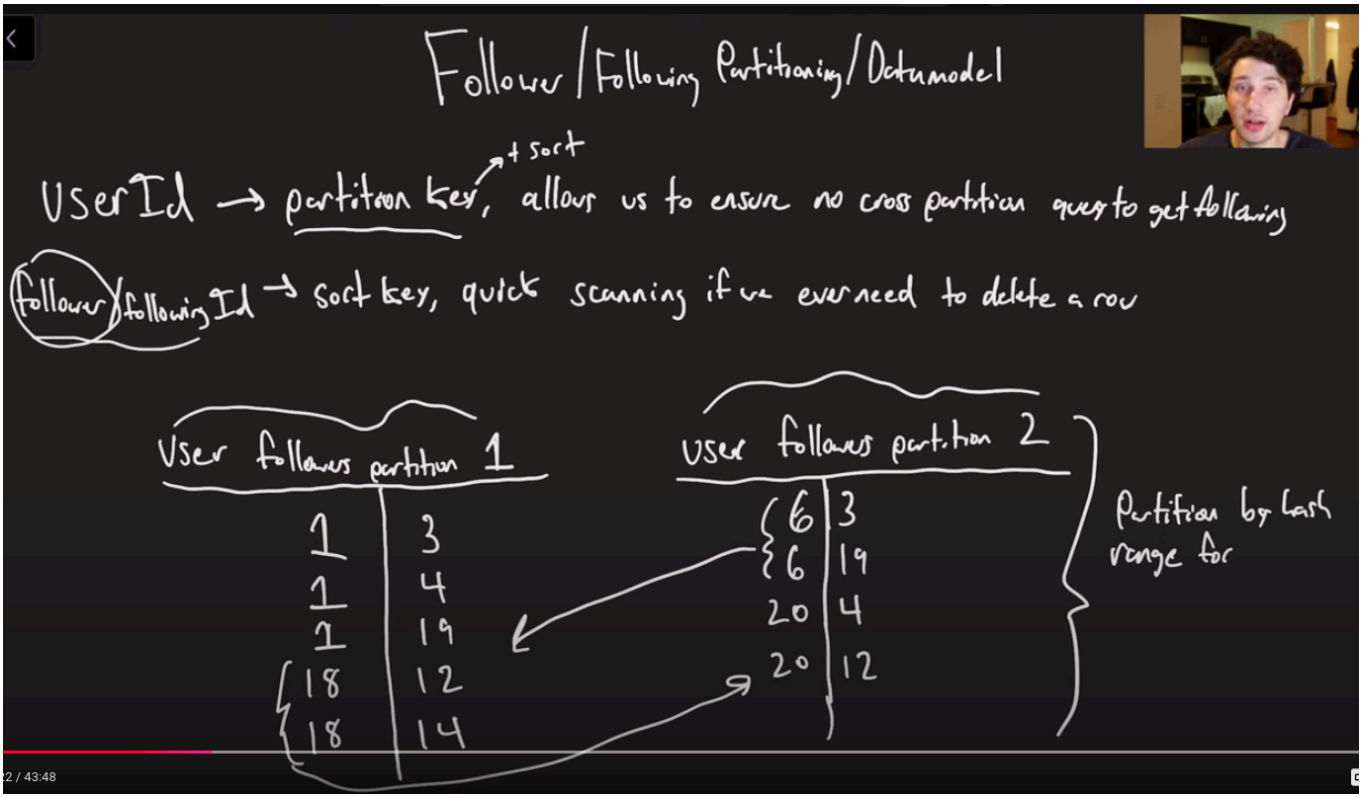


If we index on userId getting all followers will be slow and if we index on followerId getting all the people the user follows would be slow
this is becuz in the distributed setting we would have to do a linear scan of all the partitions and then aggregate it which would be quite slow.

so we will use cdc/ change data capture

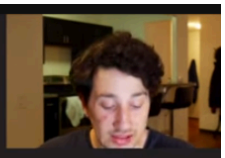






we will use hash based partitioning not range based partitioning

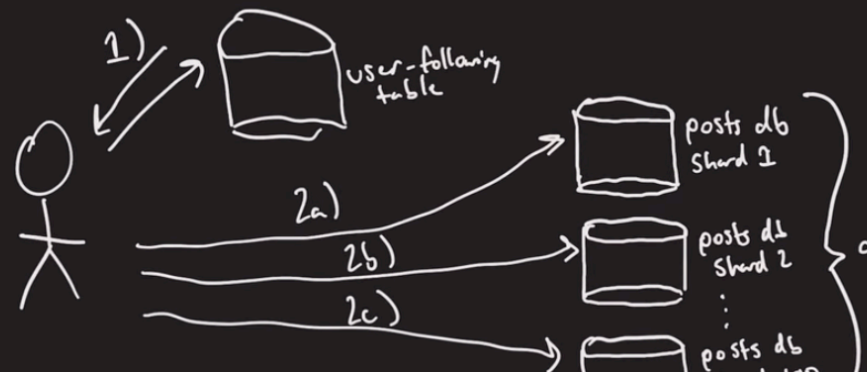
News Feed (Naive)



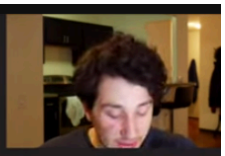
To build a news feed for user x

→ We need following(x)

→ We need posts(following(x))



News Feed (Optimal)



How can we avoid reading from multiple different places??

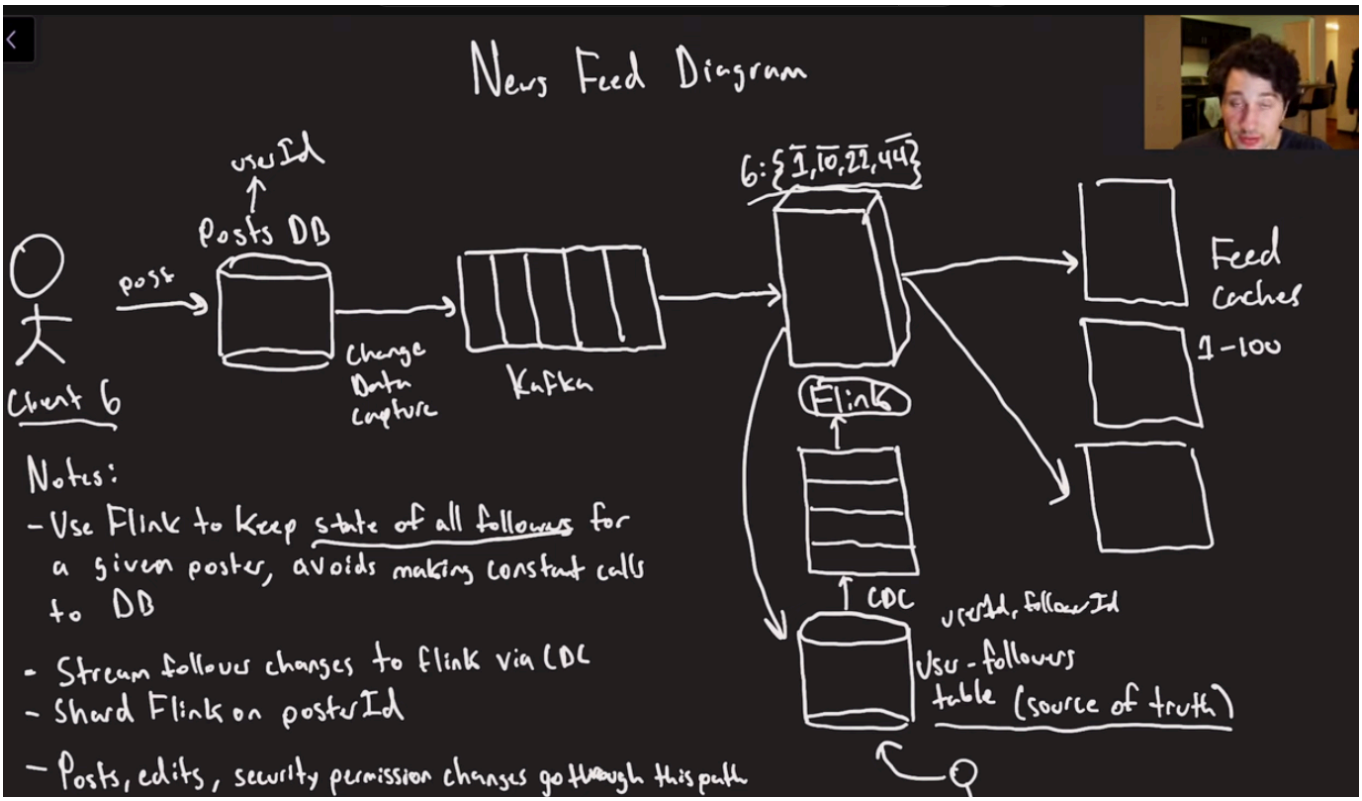
→ Would somehow need to index tweets by what news feed they belong to

→ But a tweet can be in ~100 different news feeds!

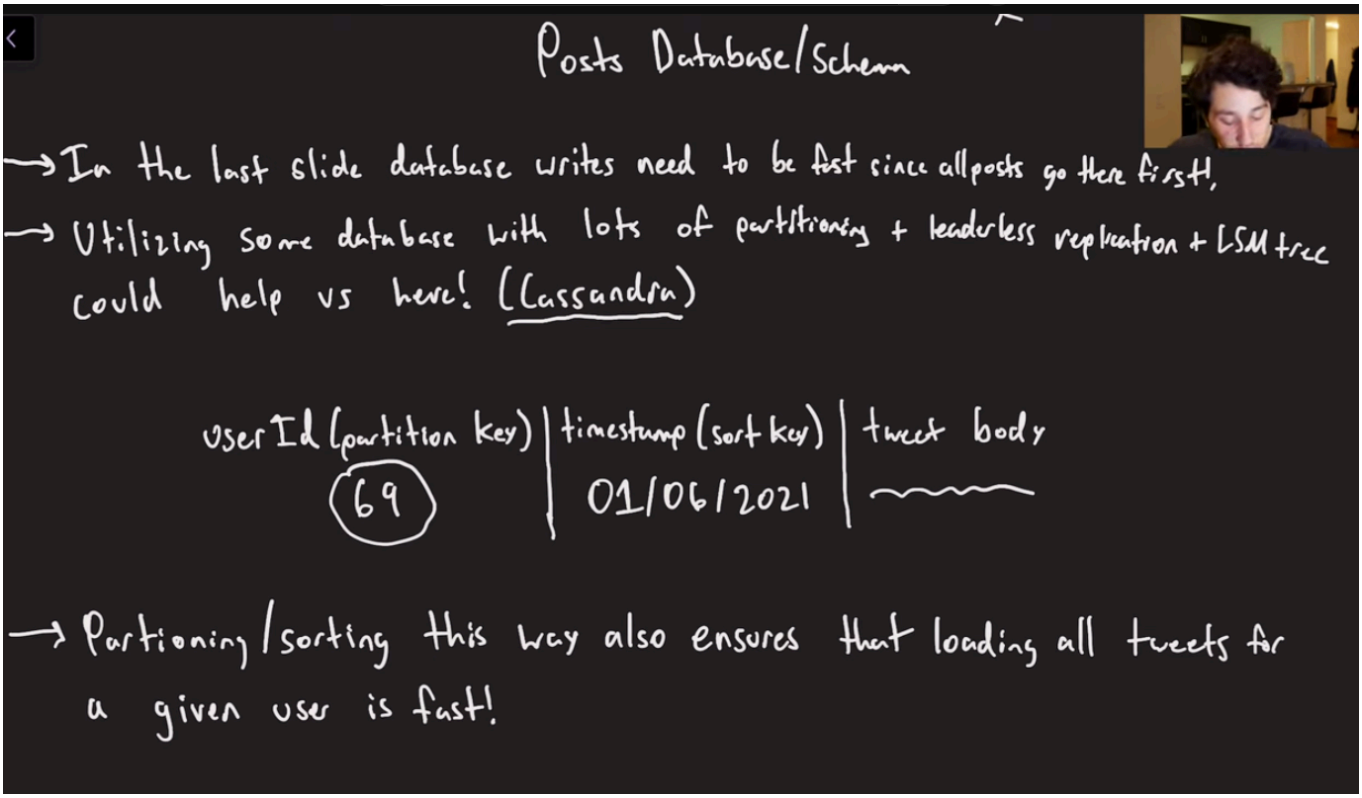
→ 1 billion tweets/day · 200 bytes/tweet · 100 copies = 20 TB of tweets per day

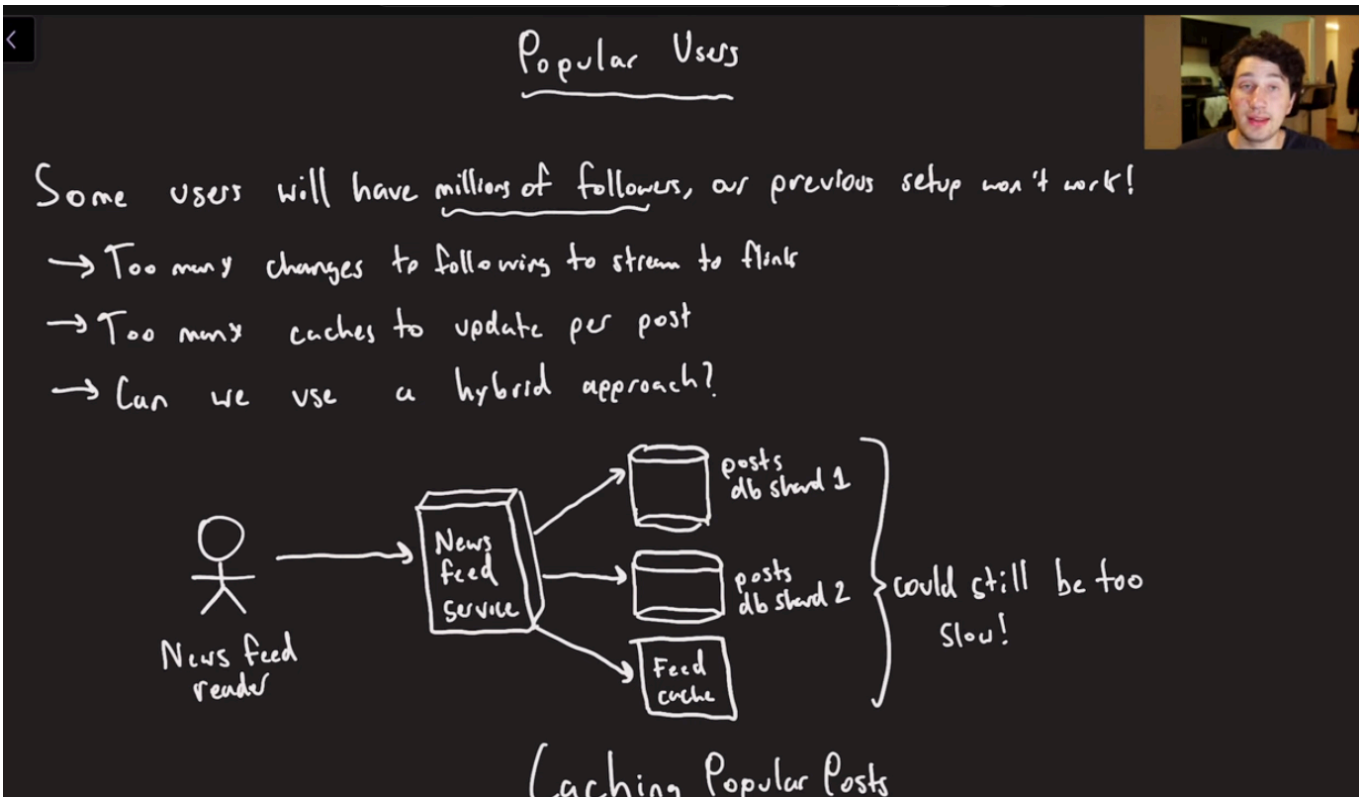
→ 20 TB / 256 gb beefy hosts as caches = 80 in memory caches

→ This is doable, let's sketch it out!

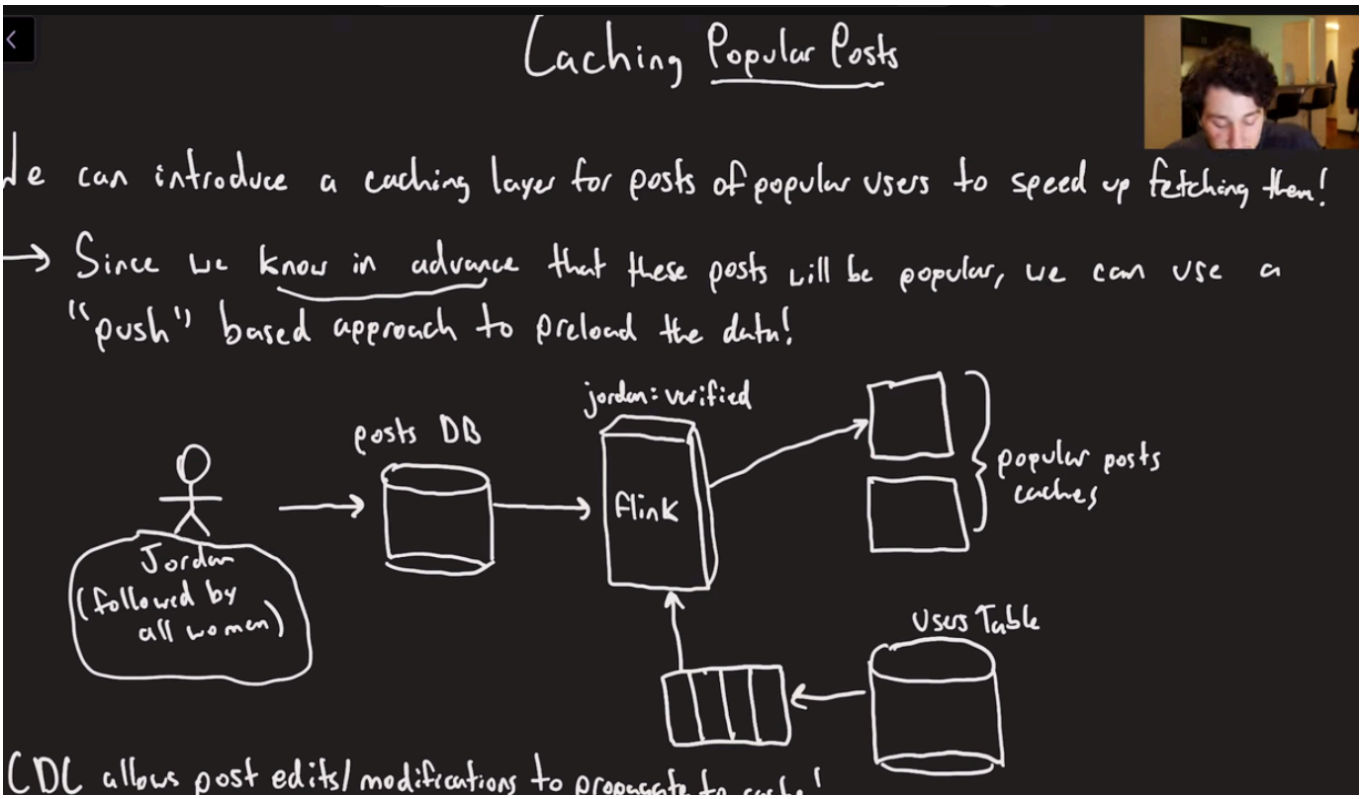


flink will be ingesting via cdc from the user followers table all the users to send the news feed to, and then flink will send the news to all the feed caches.



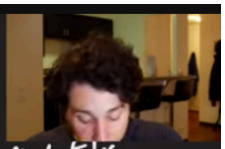


for our verifiifed users we will read from our posts db and for our non verified users from our feed cache
but this may be too slow so we can introduce a cache for the popular users



from the users table we get through cdc whether the user is verified or not and if it is verifiifed we will populate the popular posts cache.

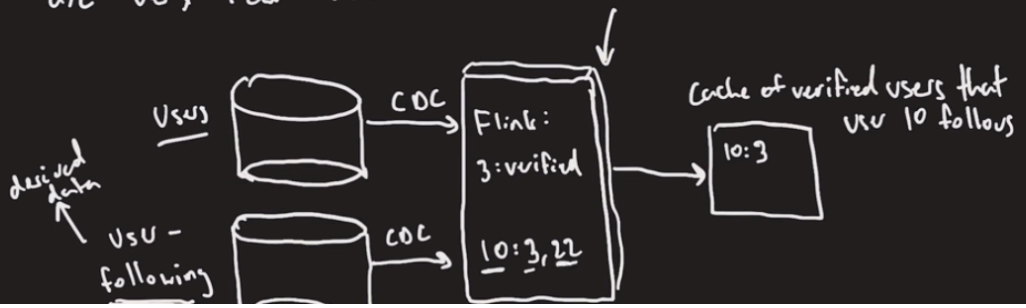
Fetching popular users that a given user follows



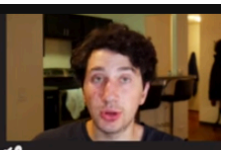
To perform our hybrid news feed solution we also need to be able to quickly figure out who a user follows that is verified so that we can fetch their posts from the popular cache.

→ We can again perform this using derived data!

→ Have to stream changes in verified users to each flink consumer, which is ok because there are very few verified users



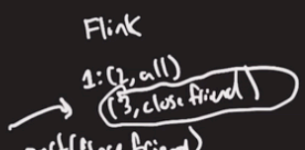
Security Levels on Posts



Let's say a user can specify whether a post is for "all" followers or "close friend" followers. Put this information in the followers table.

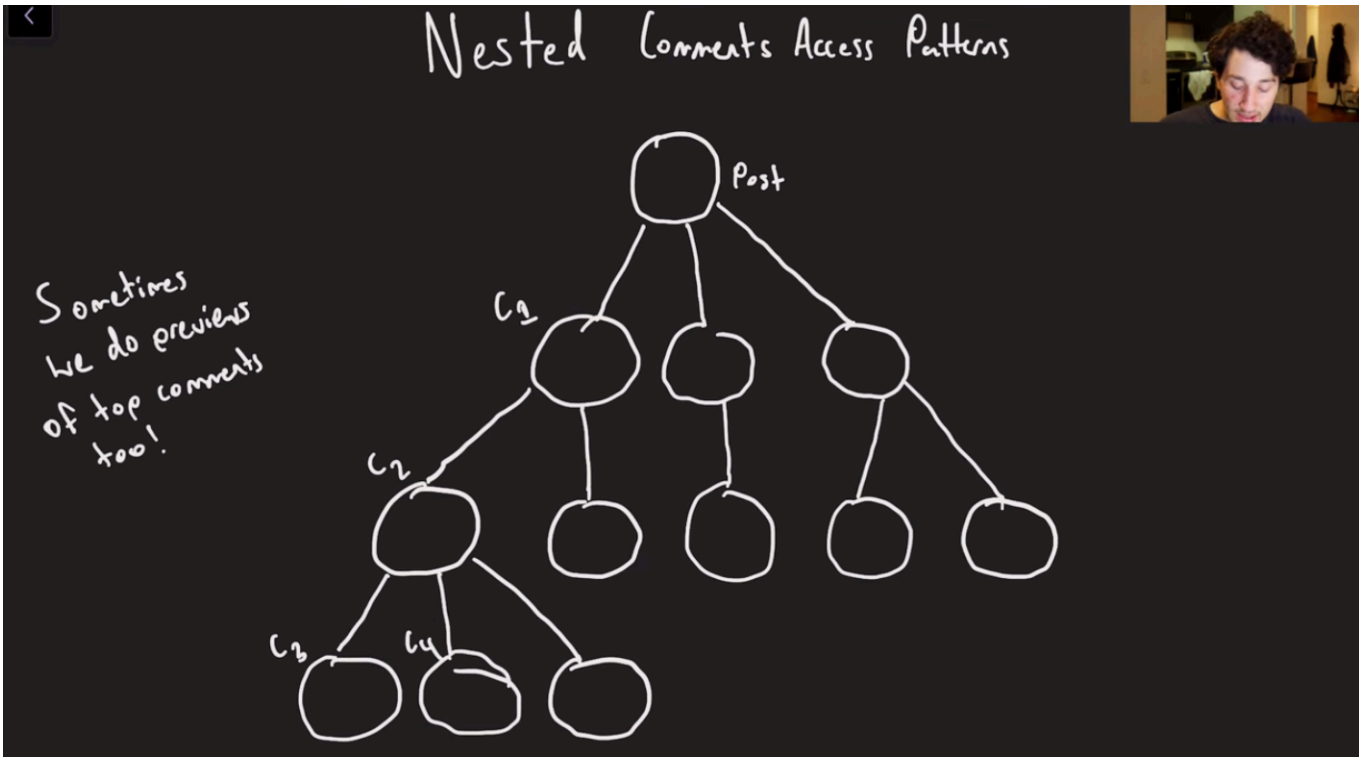
User-followers table:

user	follower	security level
1	2	all
1	3	close friend

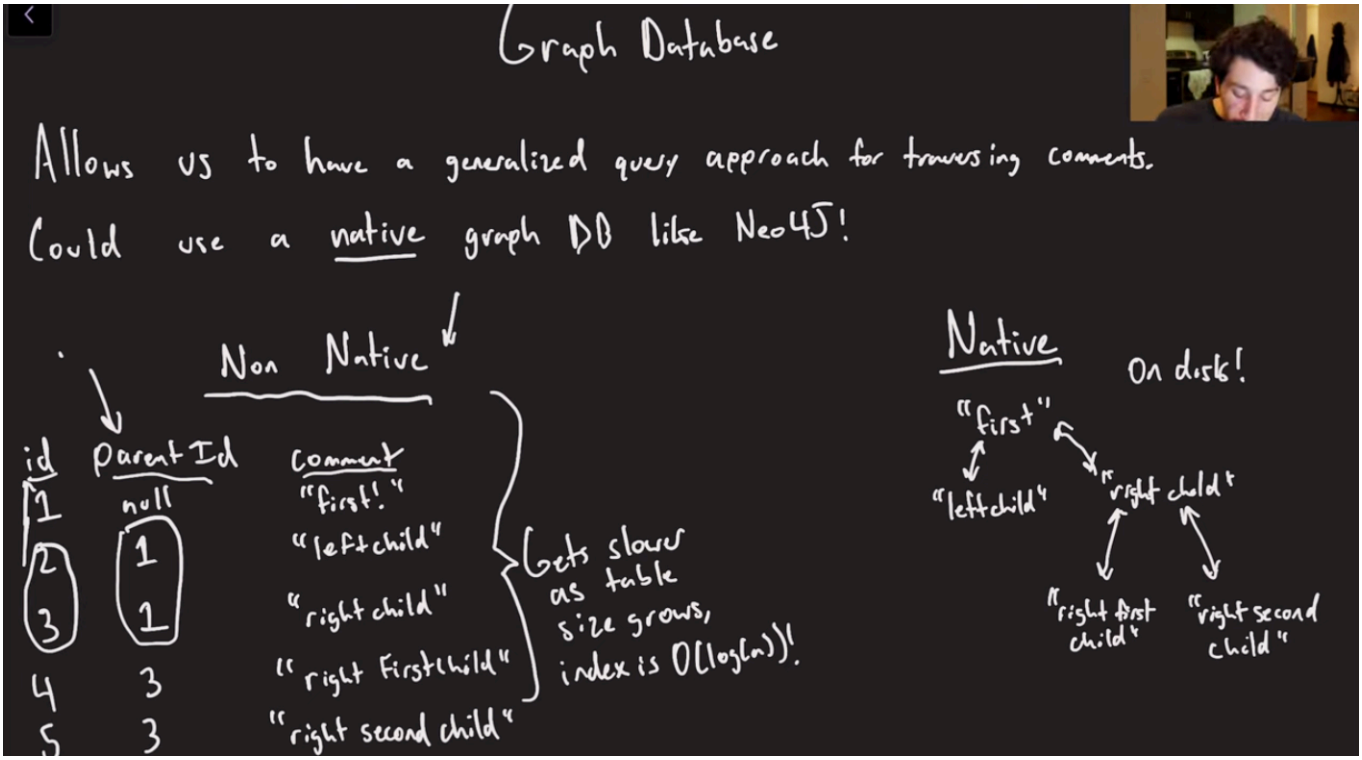


Changes to post security / follower security level will flow through our posting pipeline and eventually flow to the news feed caches

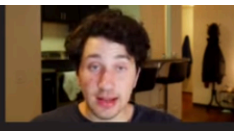
→ Unfortunately: expensive and asynchronous



we can use graph databases for nested comments



Alternatives to Graph DB

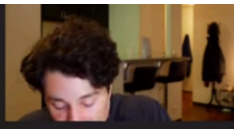


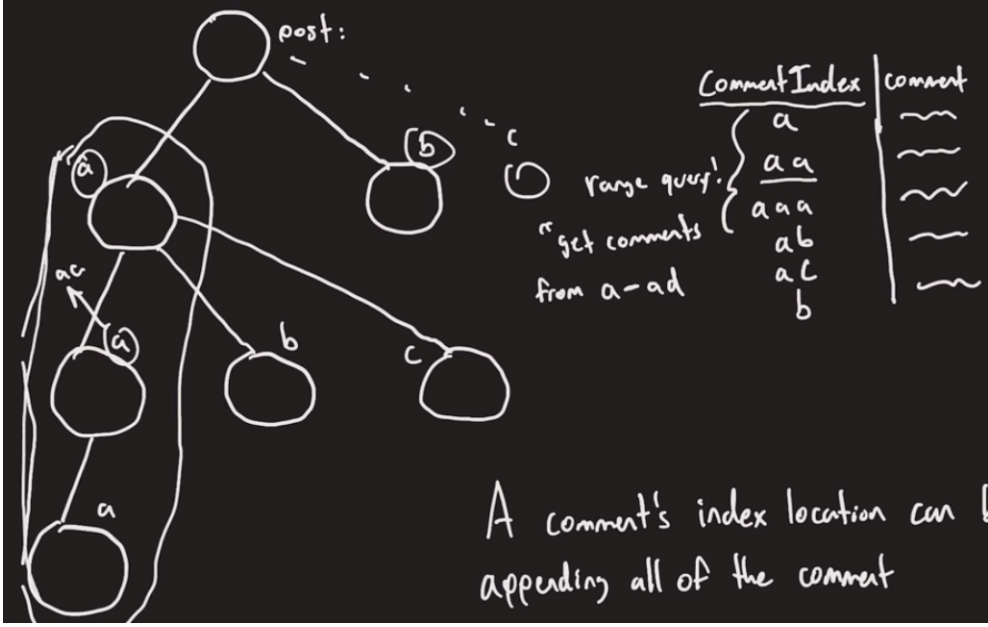
Graph DBs are still going to have somewhat high latency, jumping around on disk is very slow so we should try to avoid it when we can!

Can we do anything smart with traditional DBs to make a better index?

Let's try to build a depth first search index, as breadth first search comment traversal doesn't make much sense, we want to be able to see the parent comment when we look at a child comment!

DFS index (similar to a Geohash!)

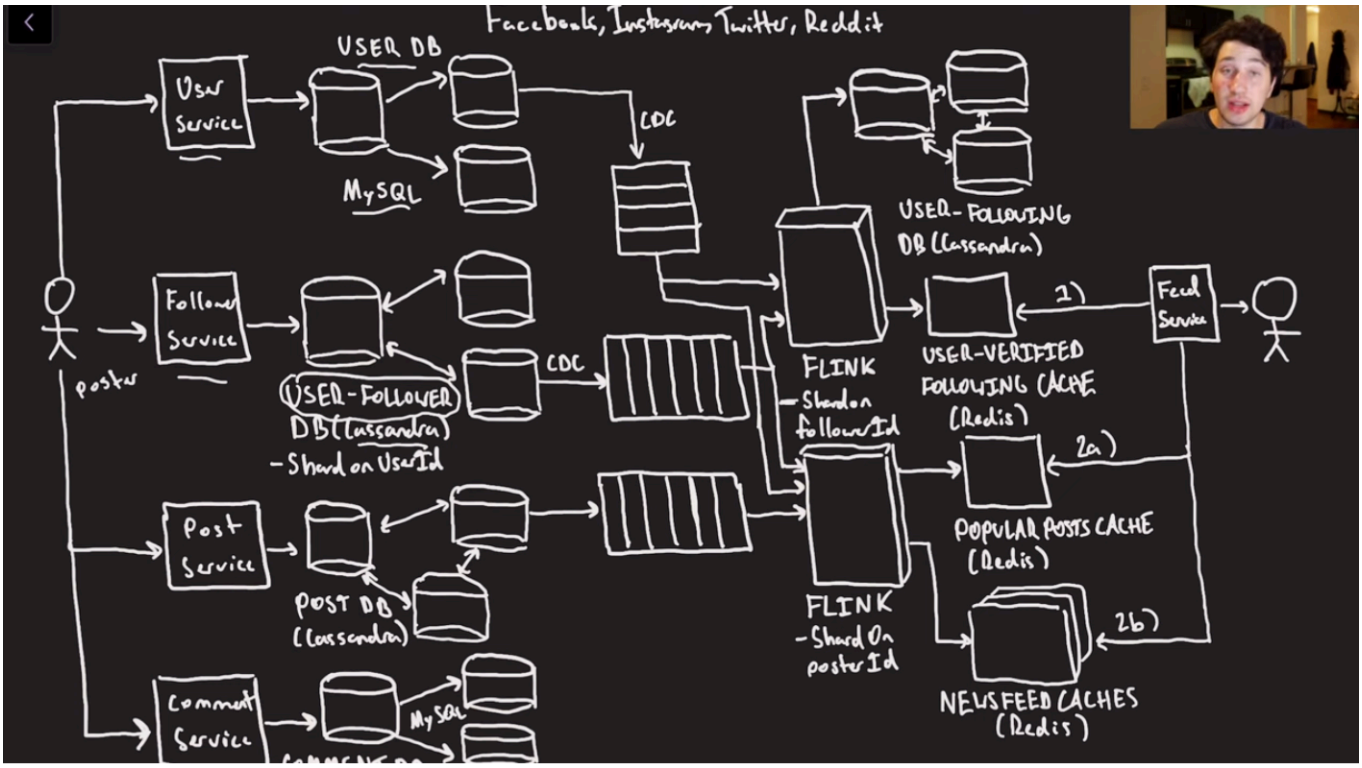




CommentIndex	comment
a	~
aa	~
aaa	~
ab	~
ac	~
b	

range query: "get comments from a-ad"

A comment's index location can be generated by appending all of the comment


Facebook, Instagram, Twitter, Reddit
USER DB
User Service
MySQL
CDC
USER-FOLLOWING DB (Cassandra)
poster
Follow Service
USER-FOLLOWER DB (Cassandra)
- Shard on UserId
CDC
FLINK
- Shard on followerId
USER-VERIFIED FOLLOWING CACHE (Redis)
1)
Feed Service
2a)
POPULAR POSTS CACHE (Redis)
Post Service
POST DB (Cassandra)
FLINK
- Shard On posterId
2b)
NEWSFEED CACHES (Redis)
Comment Service
MySQL