

Design an Instagram feed

Q : What are the feature that we need to design?

A : Create post (image/video/reel)

Follow/Unfollow

See his/her own feed.

Q: How many users can we follow ?

A : Unlimited.

Q: Daily Active Users?

A : 20 Million

Back of the Envelope calculations:
DAU : 20 million

Assumptions :

QPS :
Read --> Feed read. (1 Req)
Write --> POST (1 Req)
User opens/refresh his/her own feed --> 10 times a day.
User post --> 1 picture a day. (1 MB)

write : Read --> 1 : 10

Storage unit : (Assuming for 5 years)

QPS --> Read QPS , Write QPS ?

Storage Unit ?

Functional Requirements :

1. User can create a Post (Img/video/Reel)
2. User can follow another user.
3. User can see his/her own feed.

Non-functional :

1. Availability is more important then consistency.
2. Scalable (It could handle Billions of users)

Social media interaction :

Facebook, Instagram, LinkedIn

Graph DB --> Neo4J (NO- SQL)

Facebook : Friend --> Bi-directional connection

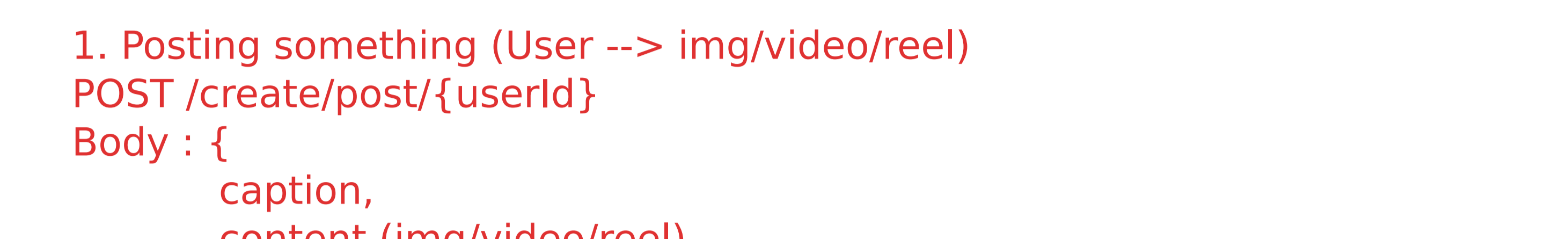
U1 <---> U2 (Un-directed graphs) u1 is friend of u2 then u2 is also friend of u1

Instagram : Follow --> Uni-directional connection

U1 --> U2 (Directed graph) u1 is friend of u2 but u2 is not friend o

LinkedIn : Hybrid Approach (Connection / Follow)

API End Points :



1. Posting something (User --> img/video/reel)

POST : /create/post/{userId}

Body : {
caption,
content (img/video/reel),
timestamp
}

Response : 200 ok.

2. Following a user (Directed Graph)

Idempotent APIs --> Dedupe (double request) i.e.. same double request

Even if double request comes, our api does not do anything wrong.

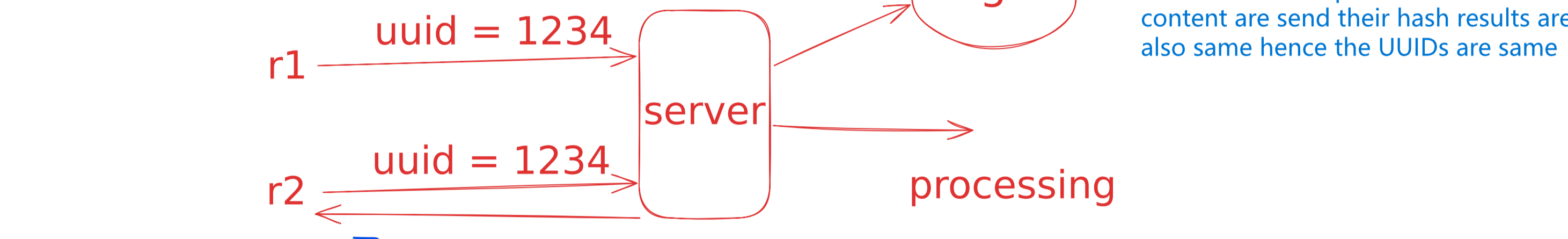


POST :
/create/post/{userId} {}
/create/post/{userId} {}
Request signature is identical

API --> double calls --> treat as one Idempotent APIs

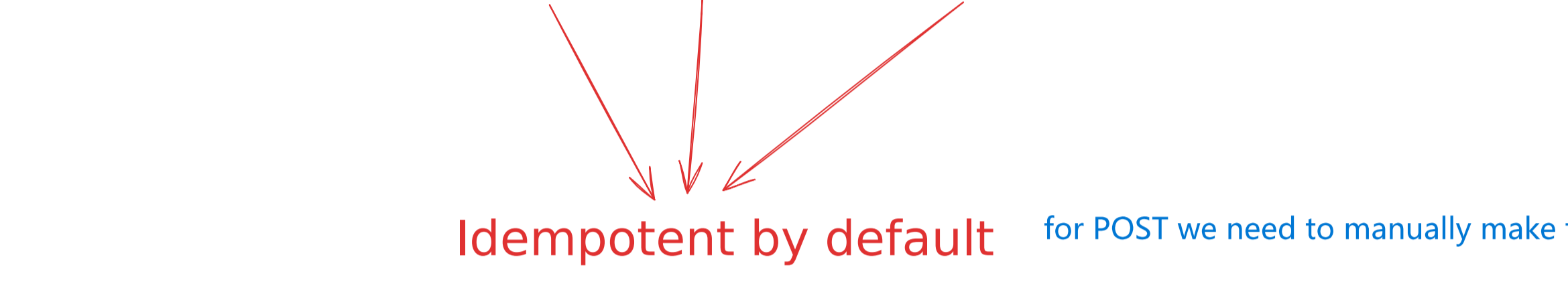


Req : headers --> UUID, IdempotentID send UUID in headers

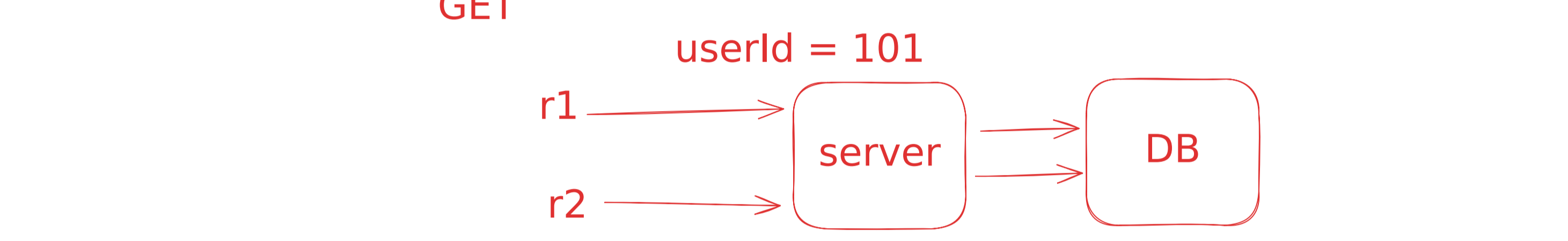


GET, PUT, POST, DELETE

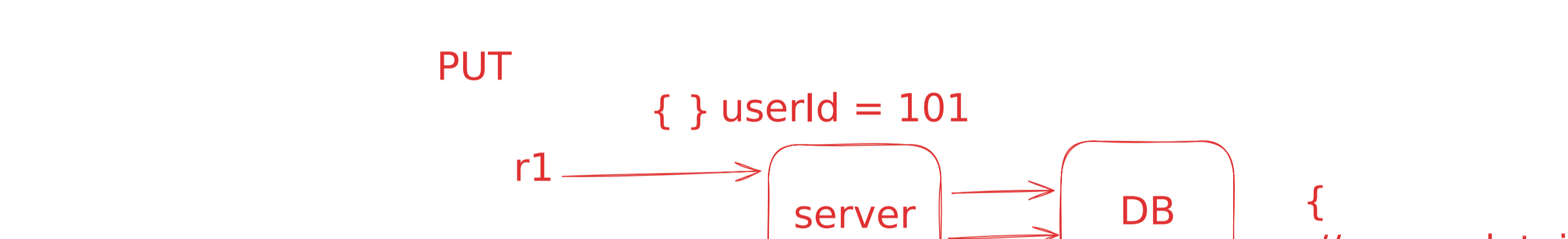
Idempotent by default for POST we need to manually make the request idempotent



PUT



POST (Not idempotent)



This is about 1. posting something

all this just to say if u1 follows u2 and if u1 again clicks the follow button for u2 then server should respond as already followed

2. Follow a user. (Idempotent by default)

u1 --> u2 (follow) --> Graph DB --> Neo4J

PUT : /user/{userId}

{

}

Reponse : 200 ok

3. View our feed.

GET : fetch/feed/{userId}

Response:

{

2

}

High level overview :

1. user posting something (img/video/reel)



1. used Simple key-value store to store our metadata

2. used Amazon S3 to store content.

2. User following other users.



1. Many to Many relationships.

2. Can be easily model by graph.

3. Thats why we are using graph DB.

3. User viewing his/her own feed

user ---> post (to whom you follow)

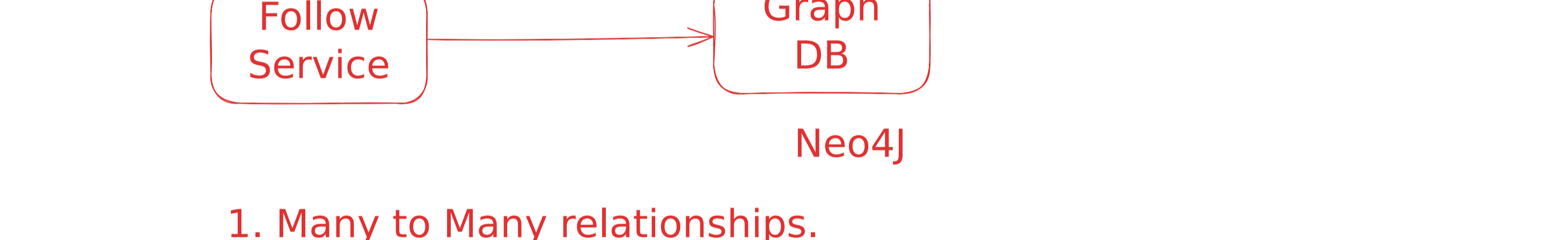
--> Reverse chronological order.

Flow :

1. Fetch the user's following list.

2. Fetch all the new post from this list. (metadata) (not actual content)

3. Finally, sort the list in reverse chronological order



cache then pushed into feed DB service to do not directly communicate with feed DB

write behind cache algo - cache communicating with server

not best approach because the design is to slow

contains all the contents(img,videos,reels) from all the world not just my followers

Instagram --> Read (far more then write)

Data fetching should be fast.

Write --> can be slower but,

Read --> should be smarter.

2 types of mode :

Pull Based pulling responses from server on refresh

Push Based.

Pull based Model : (Read slow)

Cons : slow

Pros : Better for users that are inactive. because reading can be slow for such users

Push based Model : (Read is fast)

user --> post --> pre compute in my follower's feed

cache

Pre calculation / pre compute model.

Pros : read is faster.

cons : Bad for inactive user. because the inactive user is follower list do not use the application regularly but we are updating his cache which is memory taking

2. Hot-key problem / Celebrity problem

Virat Kohli --> Instagram account (250 M followers)

Let's use Push based approach : means posting in 250 million followers cache which push based model is very slow

1. User posting something

Flow :

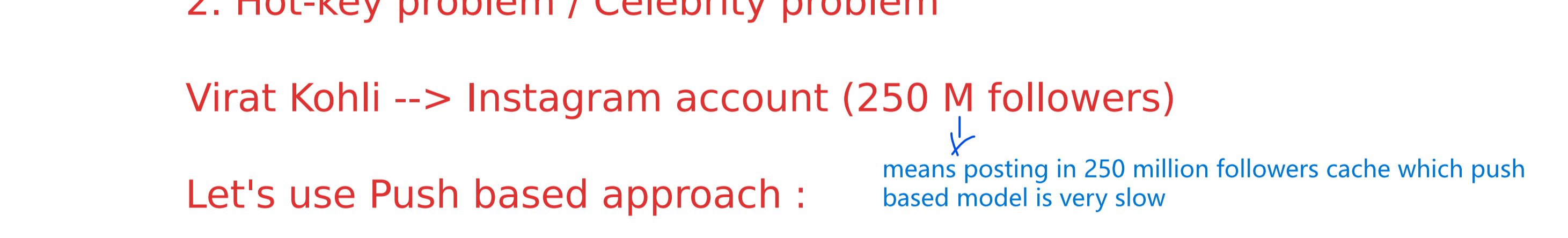
1. We will fetch the followers list of the current user.

2. then we will add the postId, to the feed cache. all the followers feed cache

3. We will store a list of pair of <userId, postId>. every follower do not have its own cache



2. When user see his/her own feed.



Everything is pre-computed here.

Problems ?

1 . Both approach has some disadvantages.

Hybrid approach solution

Every user has a boolean variable --> isPreComputed.

Virat Kohli --> 250 M approach has reading fast but has problems when it comes to celebrity problem

follower <= 1 Million (less followers --> push based approach)

People with more followers --> pull based approach

People with less followers --> push based approach

2. We can limit the number both pull based and push based (hybrid)

of post appear in feed cache i.e. when virat kohli updates picture it reaches to 200 followers

200 people data. (90 % people) when picture is uploaded for virat kohli by his followers then comes to first 200 followers then 200 followers as so

fetch next 200 --> Pagination mostly people see less than 200 posts

the above approach is still slow

i have many followers (500K followers)



Reason to use Kafka ?

1. Cache/Db throughput is less.

2. To make our Api call Async2

