HLD: Facebook Messenger

→ Step by Step approach for any HLD Problem

→ Deep Dive

(i) Polling v/s long Polling v/s WS

(ii) Idempotency

(iii) Sharding for Messenger

(iv) How Hotkeys are considered

Functional Req

⇒
1. Send and receive message.
2. Only 1-1 chats
3. Only Text Messages.
4. Store the Messages on Server.
5. Get Recent Conv
6. Get History of a conversation.

Add^n Features

→ Media uploads

→ Read Receipts.

→ Group Chats.

Non f^n Req

C → Yes Avail → No

* Consistency v/s Availability

Consistent

But v.v.v.v.v.v Highly Available

* Consistency v/s Low latency

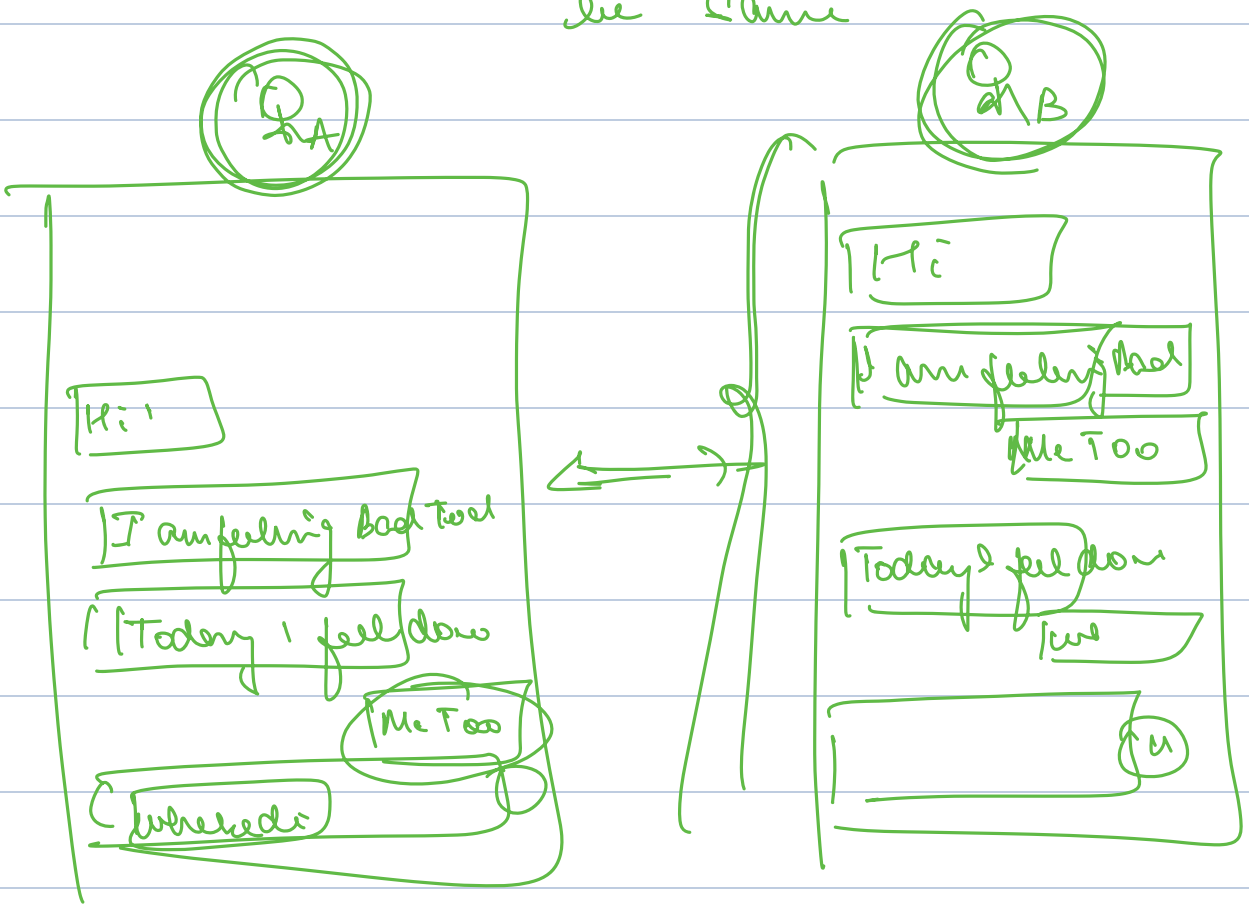
Consistent

But v.v.v.v.v.v Low latency

Slack, Teams, ⇒ Available

Messenger ⇒ Consistent

Consistency ⇒ order of Messages should be same



(3) Back of Envelope Calculations

QPS → Read
QPS → Write

Storage

1 QPS (write)

New Message

Assume 1B users

avg # of messages / person / day = 50

⇒ Total # of Messages = 50B

⇒ 50B writes / day

$= \frac{50 \times 10^9}{10^5} \rightarrow 50 \times 10^4$

avg load = 500,000 ⇐

peak load = 5x avg

= 2.5M messages / sec

Similar # of reads as well

→ R and W ratio ≈ 1:1

⇒ Both read and write heavy system

Sharding

50B/ day

⇒ 5 years → $50 \times 10^9 \times 5 \times 400$
(50 years, 400 days)

⇒ $100\,000 \times 10^9$ messages / 5 year

⇒ 100 Trillion messages / 5 years.

4B/ char → UNICODE

⇒ Emojis

Message

- id (8B)
- Content (500B)
- Created At (8B)
- user ID (8B)
- Conversation ID (8B)

500B/ message

⇒ Over 5 years ⇒ $10 \times 10^{12} \times 500$ B

5 PB

5000 TB

replicas

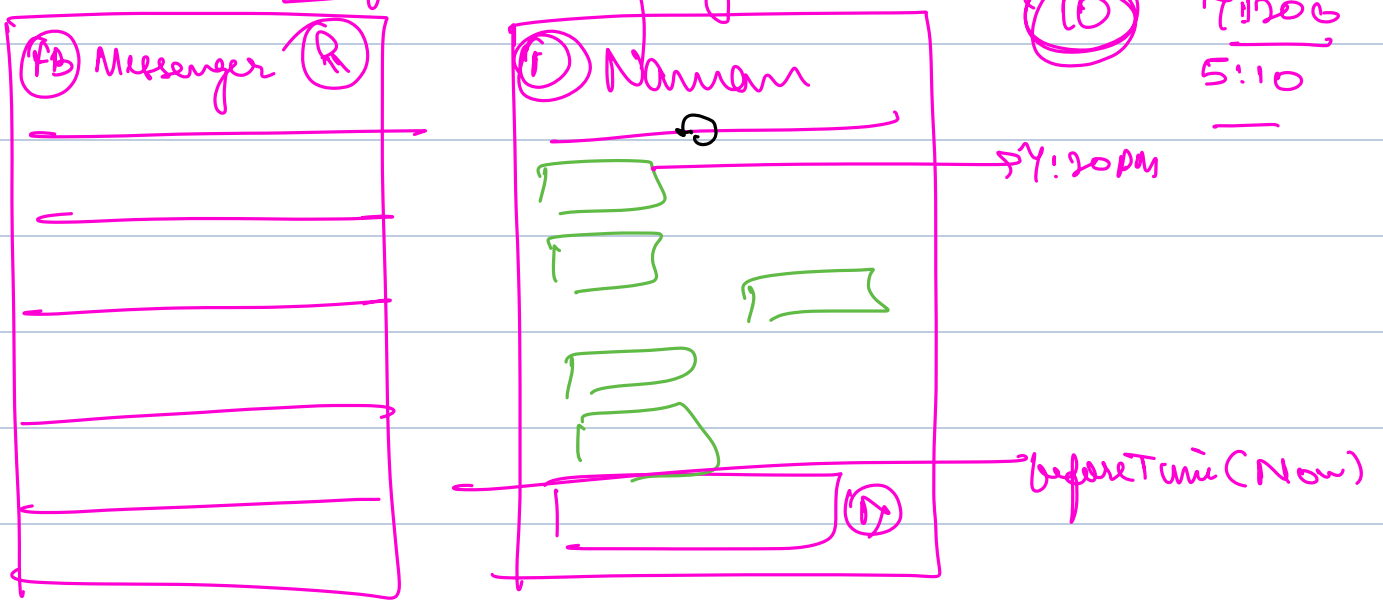
Sharding

Can't survive w/o sharding.

---

④ API Design

- getRecentConv(user-id)
- sendMessage(convID, content, userID)
- getConvHistory(convID, count, beforeTime)



Comm ID

User1	User2

ID = 64 Byte ⇒ 128 Byte UUID

100 trillion every yr

10000 0000 0000 0000 0000

$(2^{64}) \approx (10^9) * (10^9)$

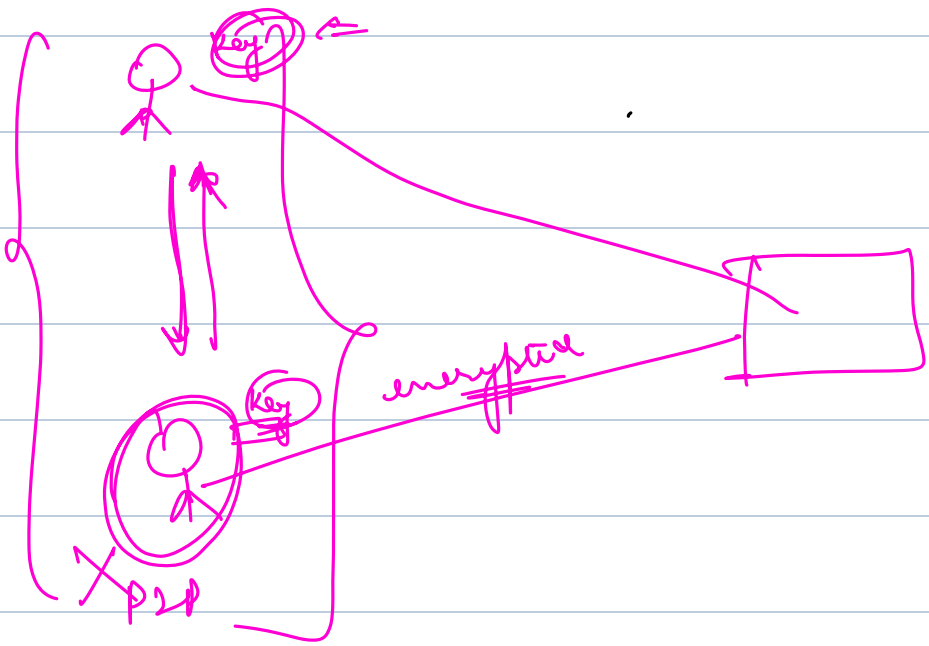
$\rightarrow 10^{18}$

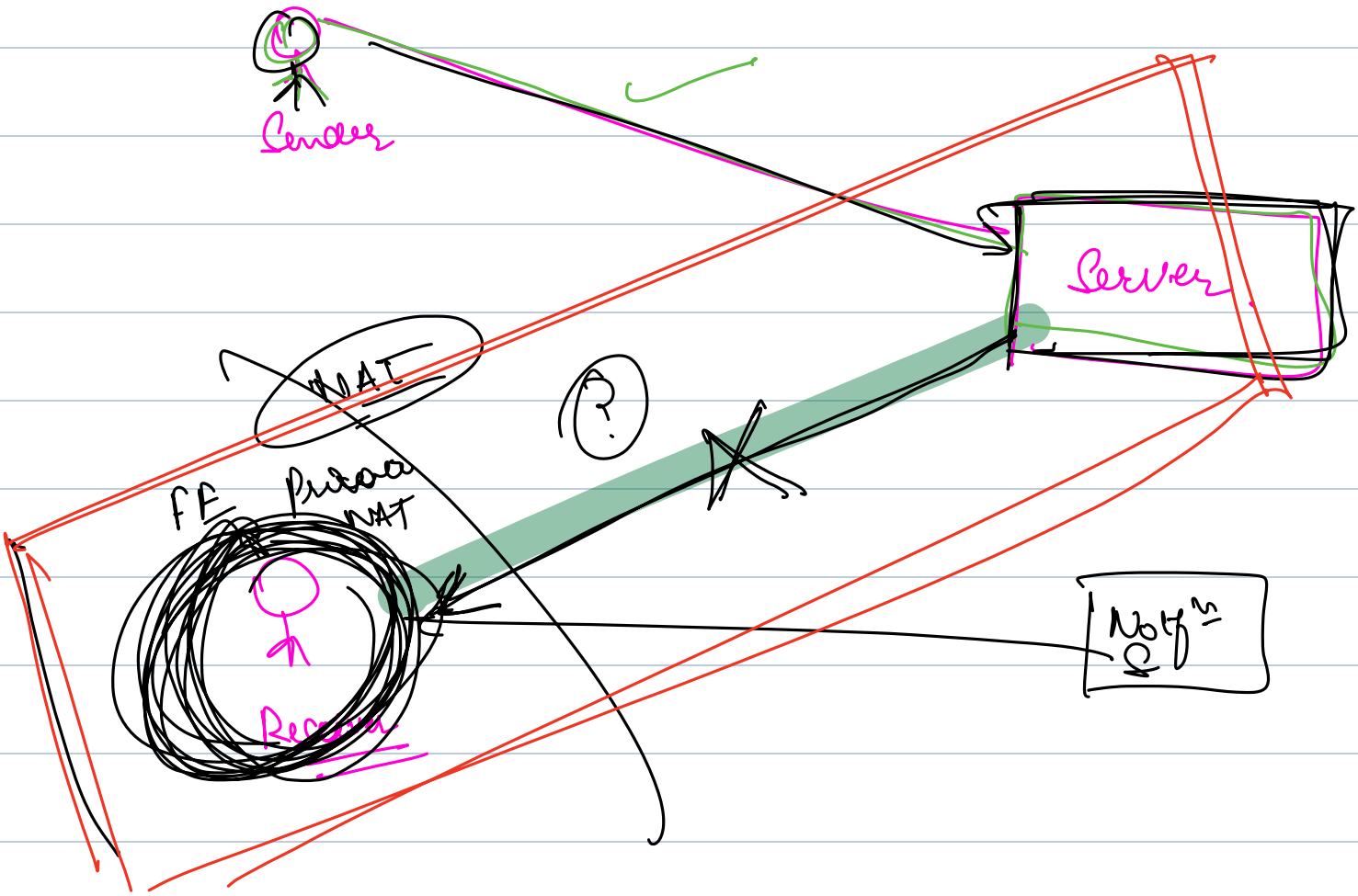
50000

50000 $\times 10^9$ year earlier

2 Bn mo

(5) DESIGN


Key
encrypted
Key
P2P

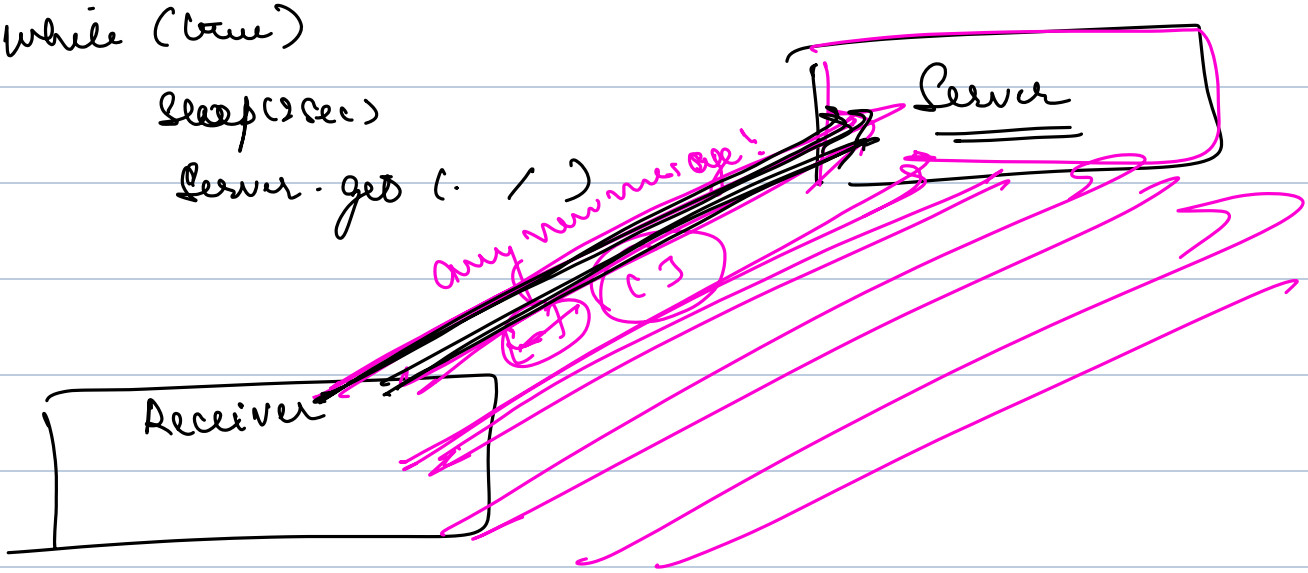

Sender
Server
NAT
?
FF
Private
NAT
Receiver
Notify

Server - Receiver Comm

- Polling
- Long Polling
- Websocket

Polling

```
while (true)
    sleep(1sec)
    server.get( . / )
```



receiver initiates a conn with server

every "x" sec:

if server has new message:

it sends true

else:

[ ]

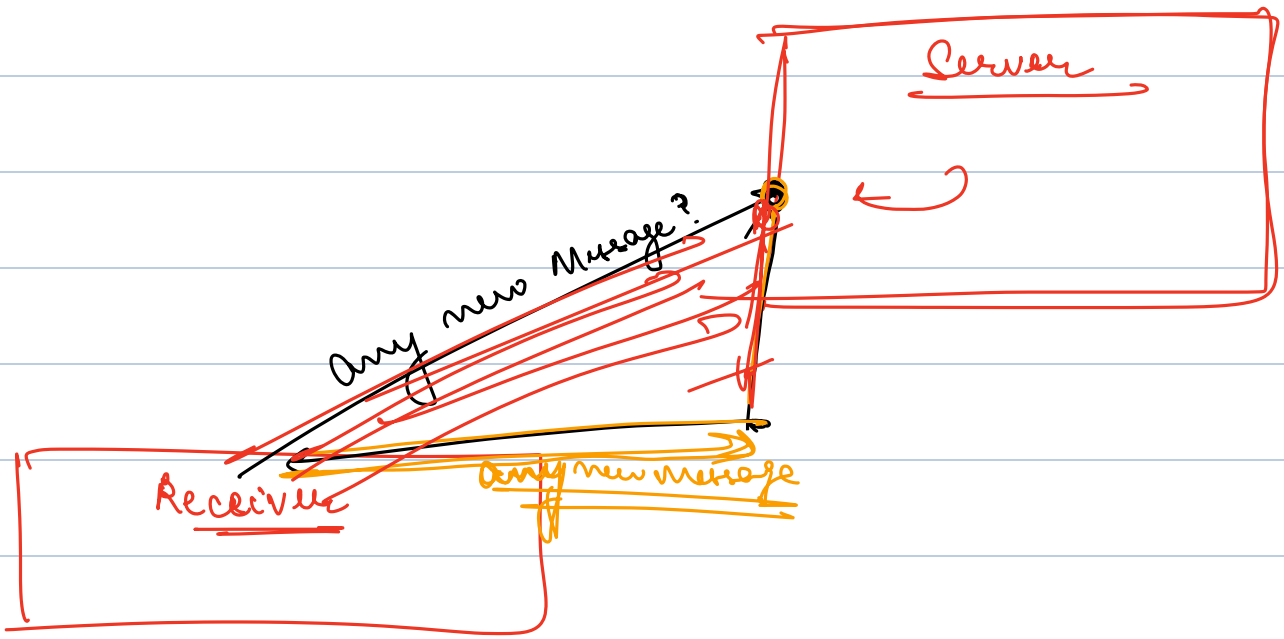
CON

- Inefficient
- Consume energy, battery
- Lot of n/w calls

PRO

- Simple

Long Polling



1) Only request count = (# of messages)

PRO

1) Relatively Simple (Still just HTTP Req-Resp)

CONS

1) Keeping a lot of conn alive will increase RAM usage

(need to have servers with high RAM)

↳ to hold the pending Req

⇒ increase cost.

2) Realtime Chat Exp is still missing

3) Connection can timeout

↳ create a new conn.

WEB SOCKETS

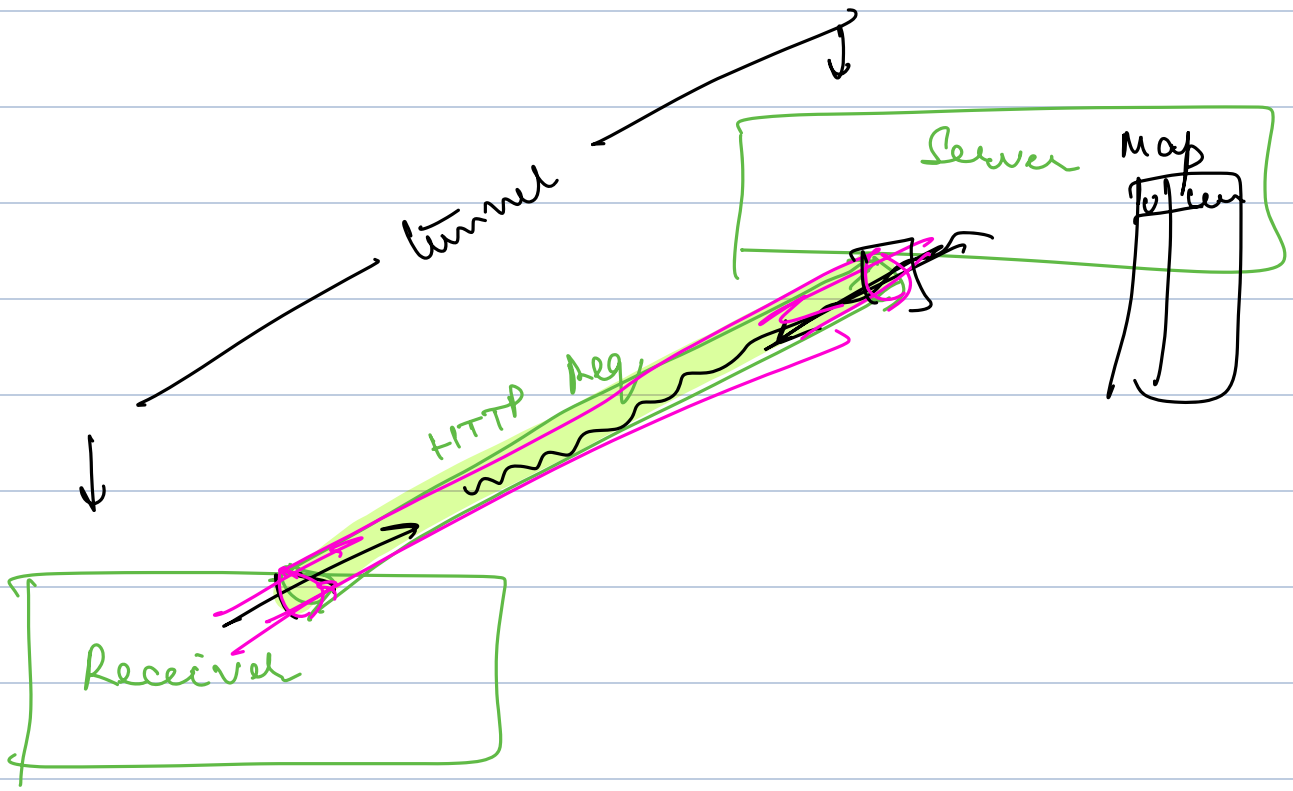
⇒ persistent

⇒ bi-directional

conn b/w clients and servers

⇒ initiates as an HTTP Req

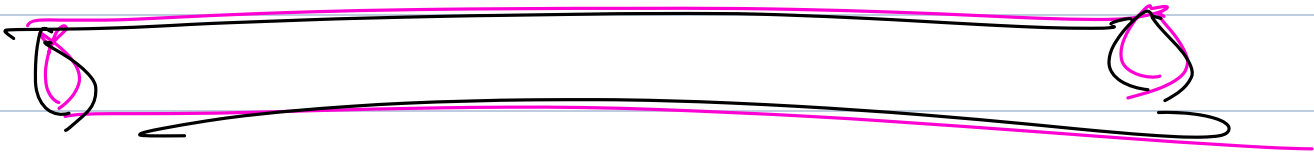
↳ and upgrades itself into a bidirectional conn.



ws. send ( )

map[uid]. send ( )

messenger.fb.com



http.

https

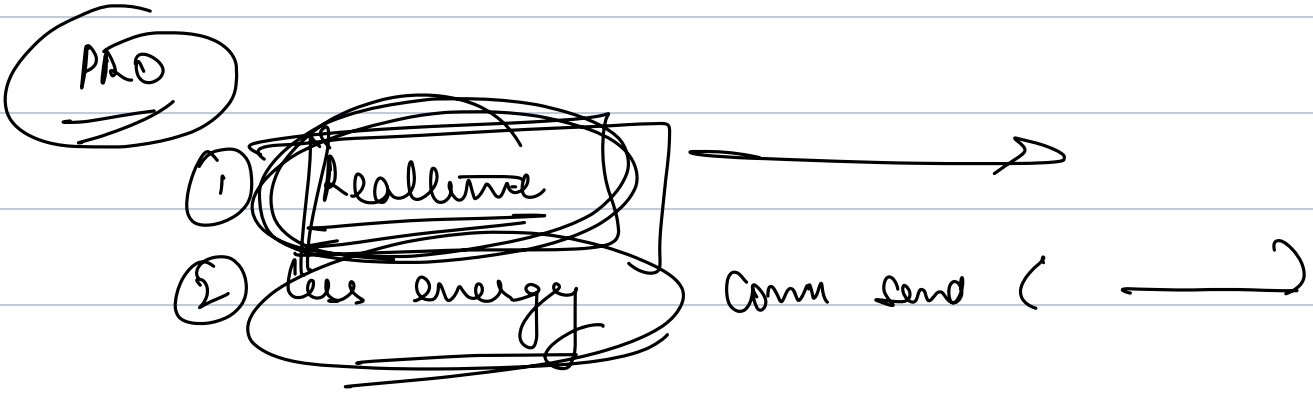
ws

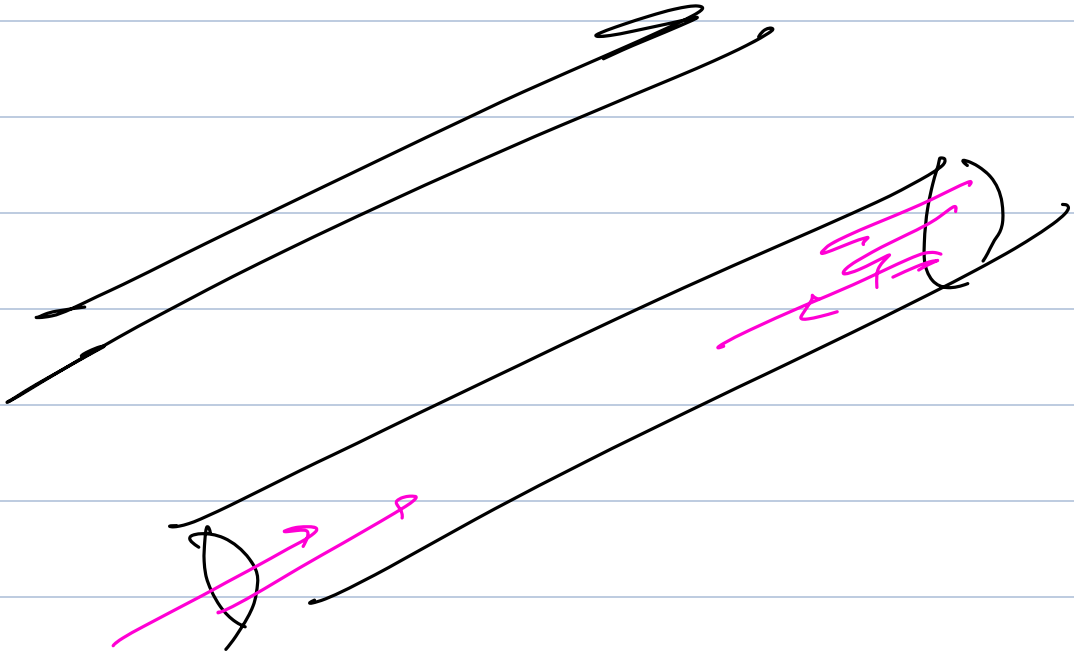
wss://

CON

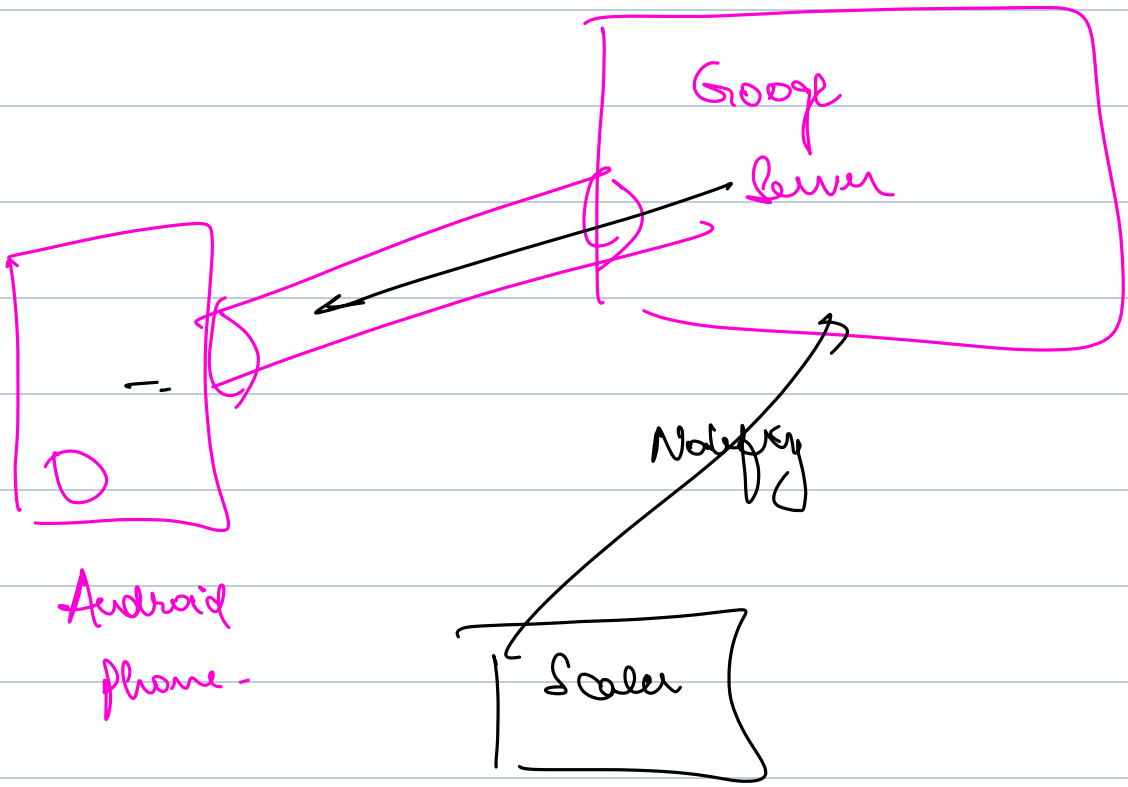
1. Use RAM
2. Handle complexity of reconnection etc.

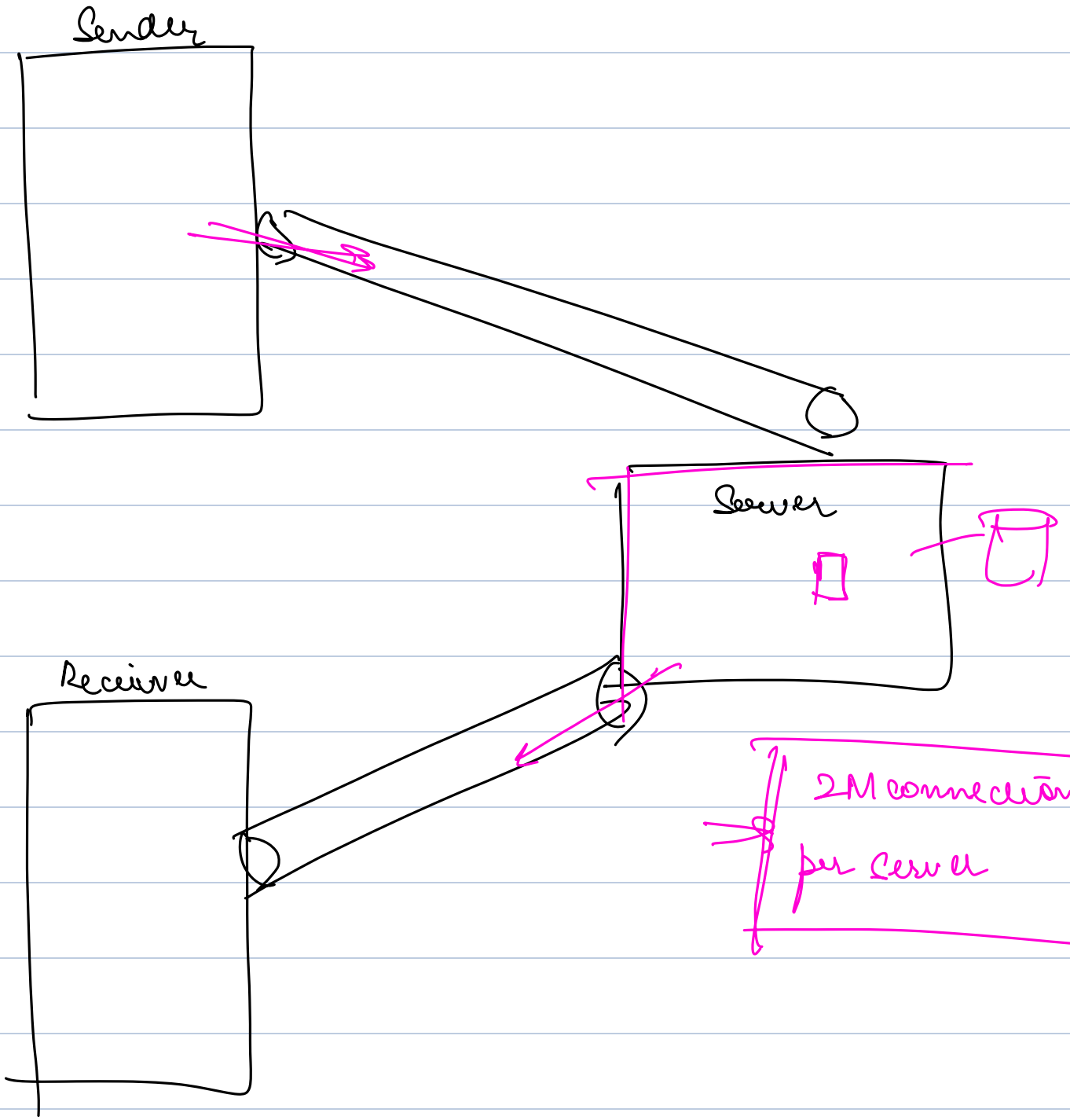
PRO

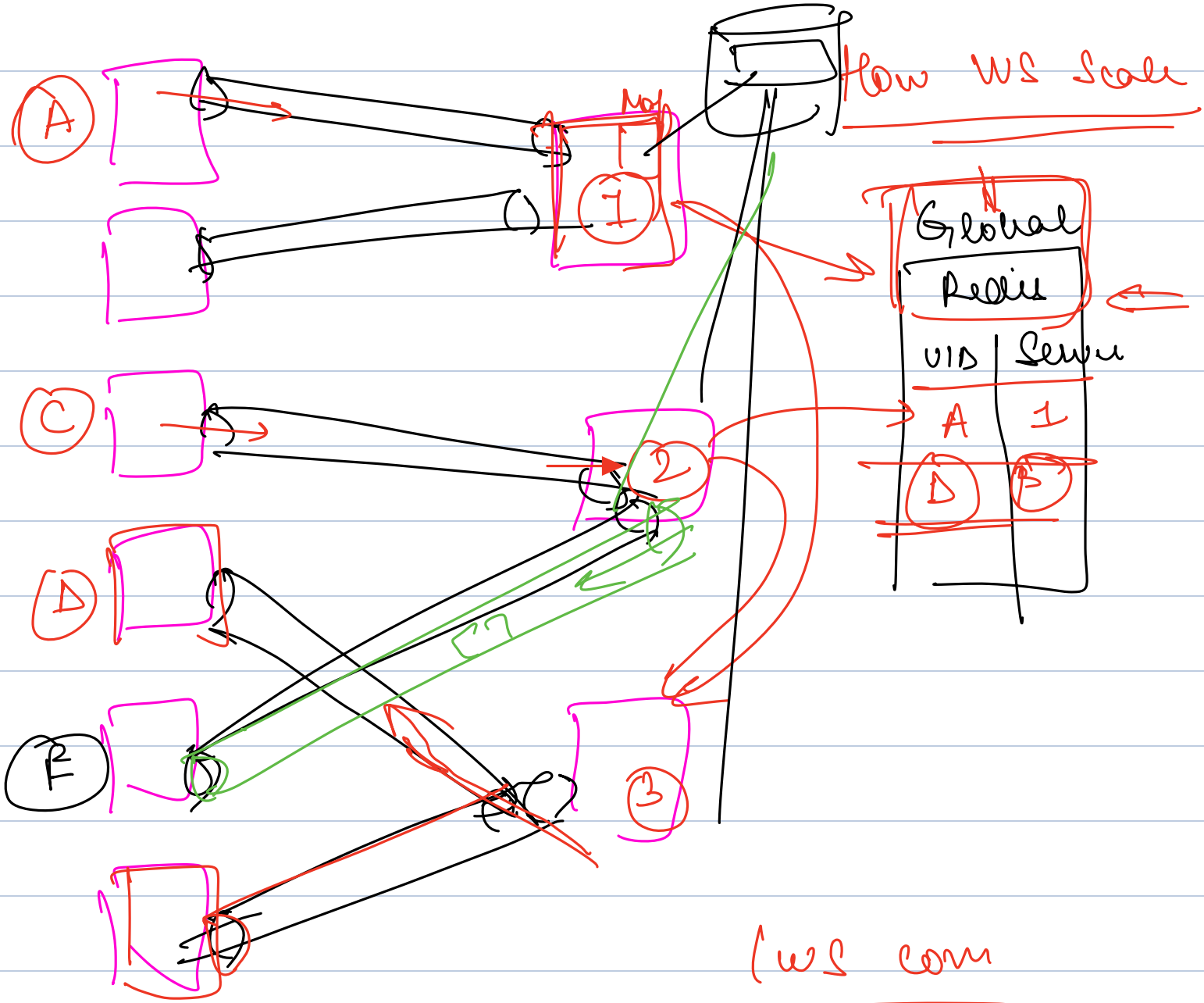




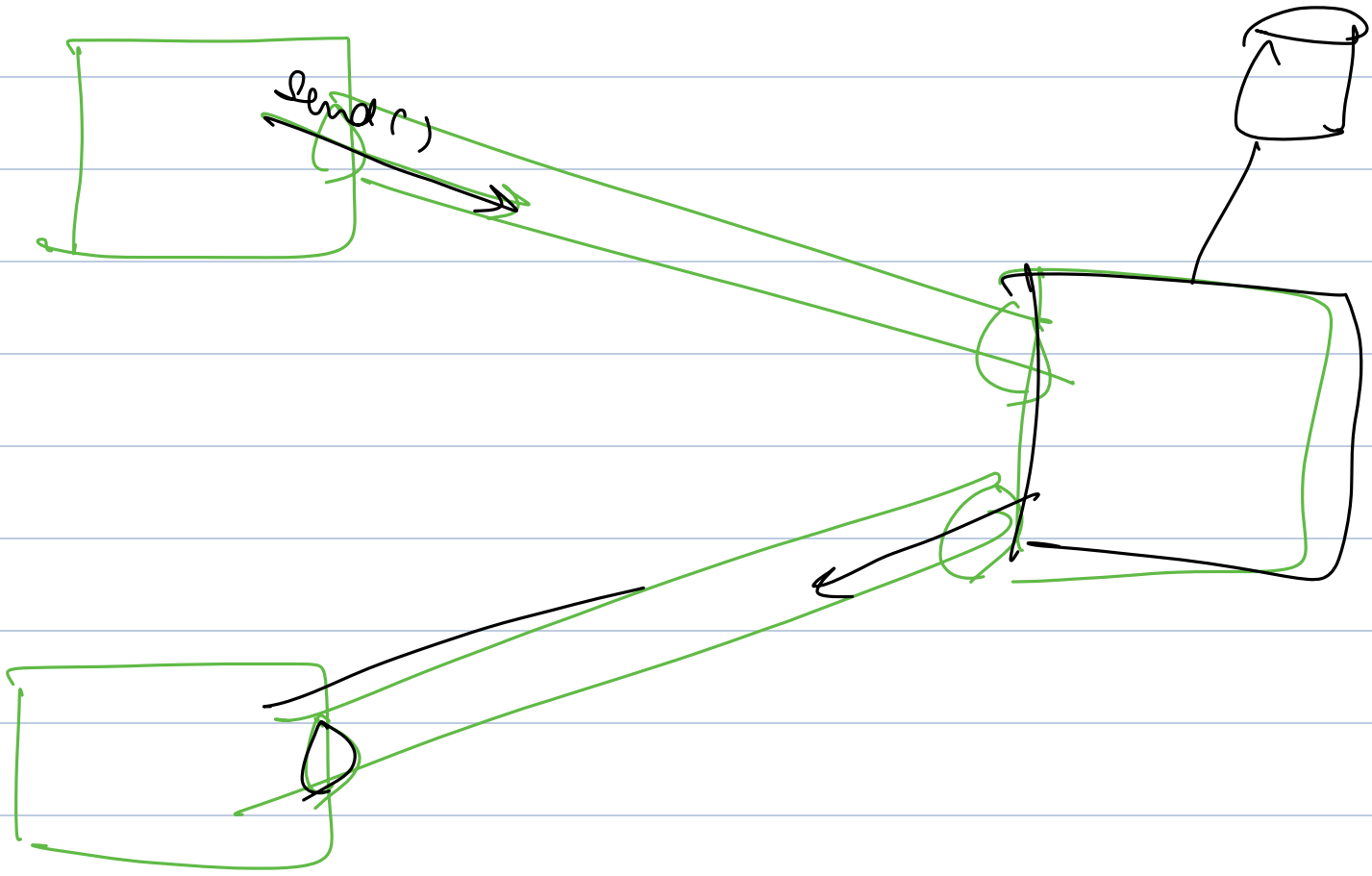
Google Firebase Cloud Messaging



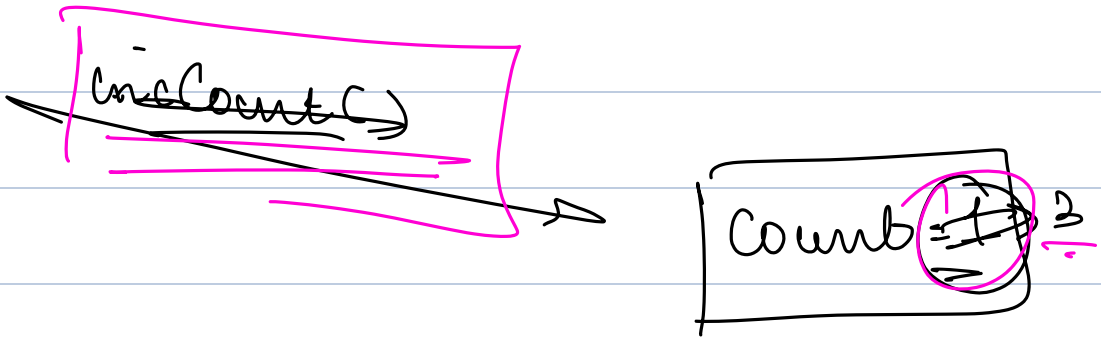

Sender
Server
Receiver
2M connection
per server

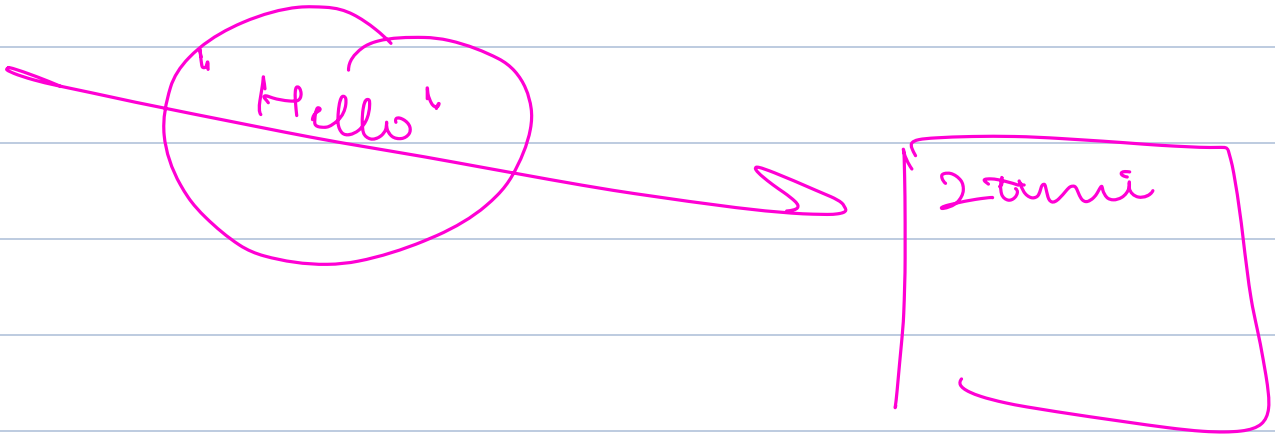

How WS Scale
A
C
D
F
1
2
3
Global
Redis
UID
Server
A
1
D
2
WS conn
1 MB
2*10^4 * 10^6 → 2*10^C

messenger.fr.com

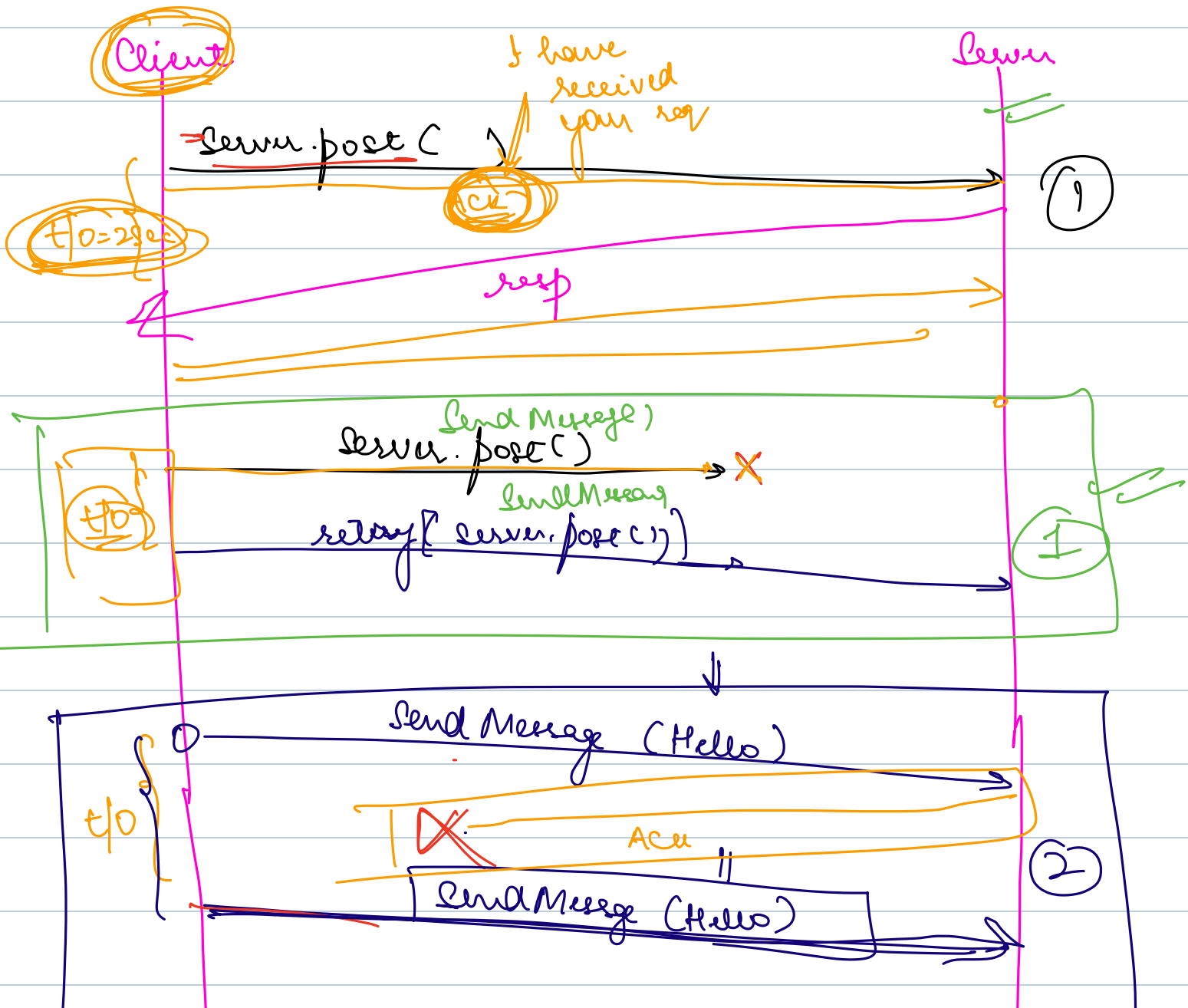


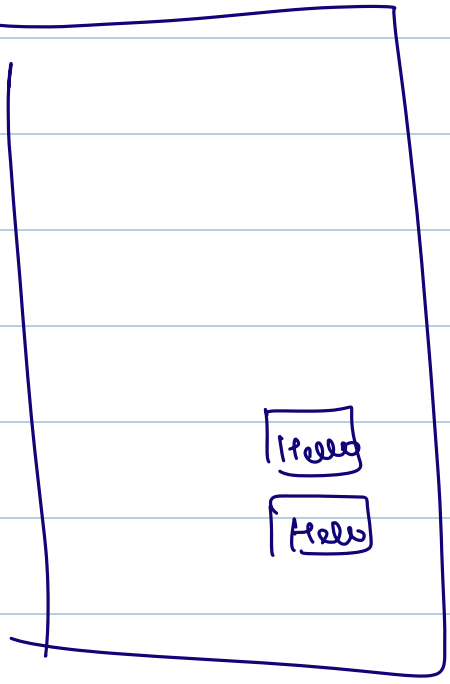
2 IDEMPOTENCY





How internet Works





⇒ we sent once, got delivered twice

⇒ we need a way to recognize a request is duplicate