HLD of Live Streaming Services

HLD

→ Naman Bhalla

(Will be starting at 5:10PM)

→ 2.30Mins → 7:30

Break will be around

6:30PM

Scaler.com/events

Naman Bhalla

Youtube Live | Hotstar

How do OTT Platforms Scale for such a large userbase

SQL

HLD Masterclass

(High level Design) System Design

⇒ No code

⇒ Not a frontend architecture

(Backend Heavy)

(Backend Flow)

Agenda

→ What is HLD

→ How to approach HLD

→ Live Streaming Platform

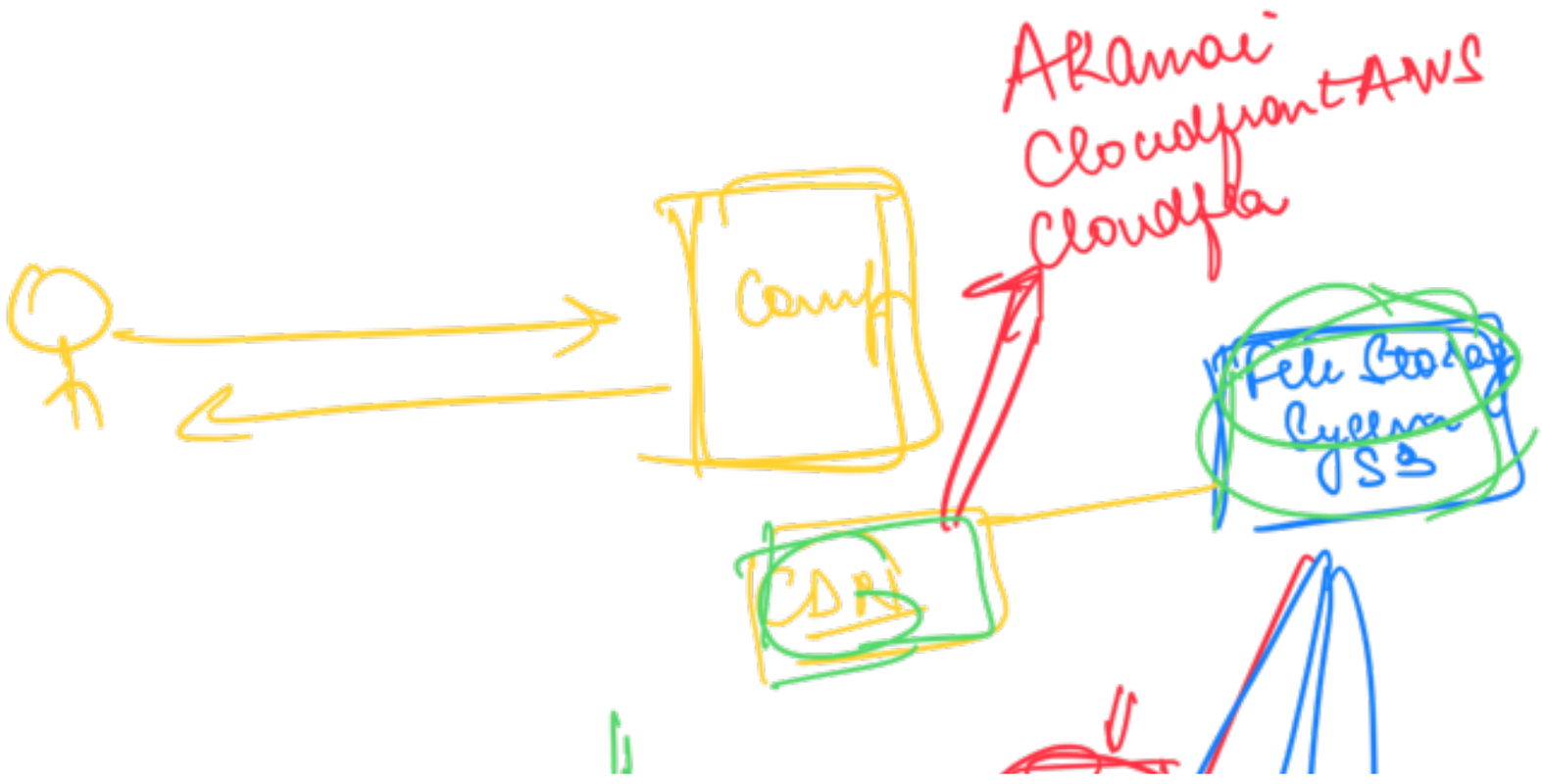
Bird's Eye View

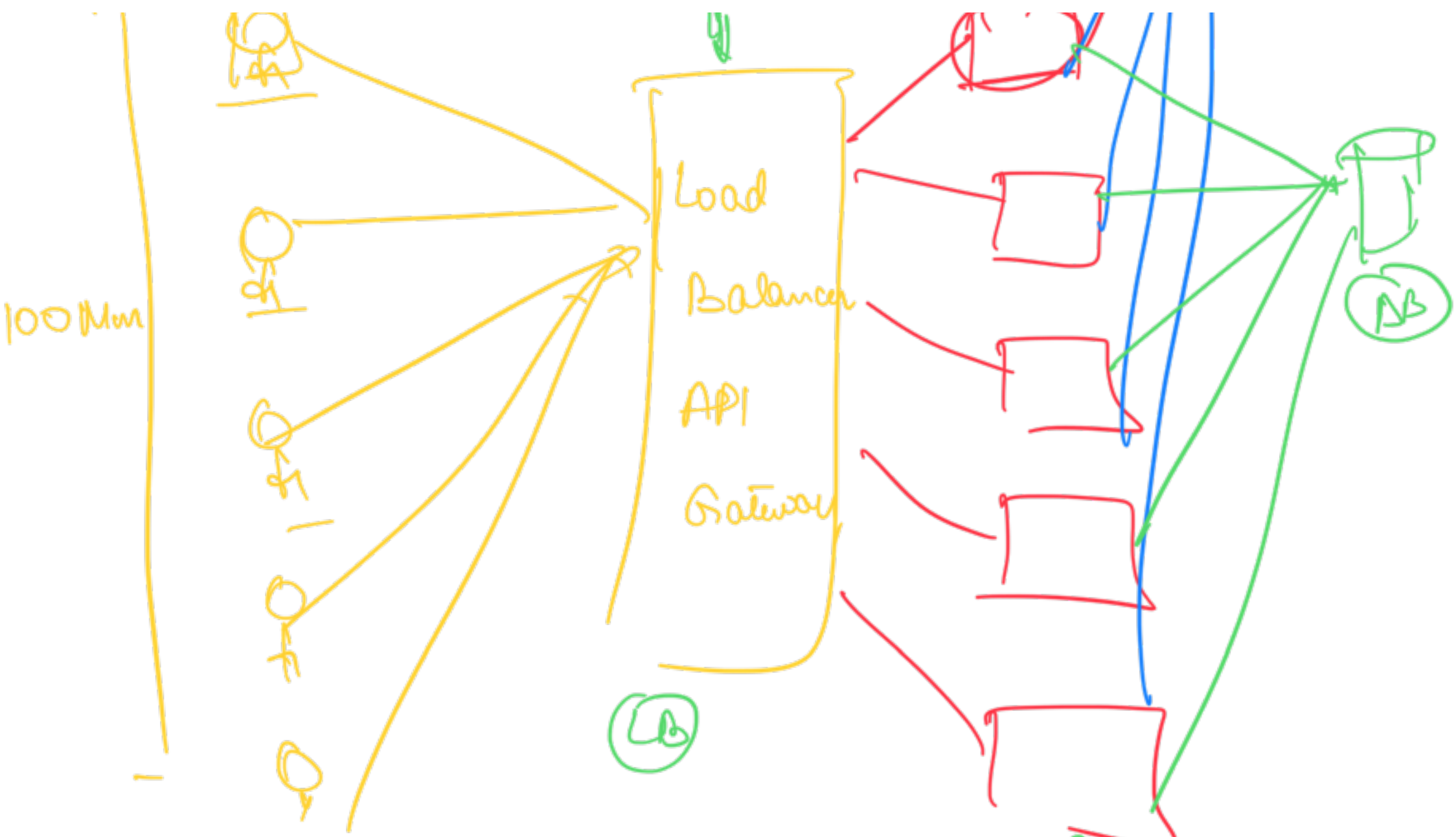
What is HLD → Bird's Overview

HLD: High level Design

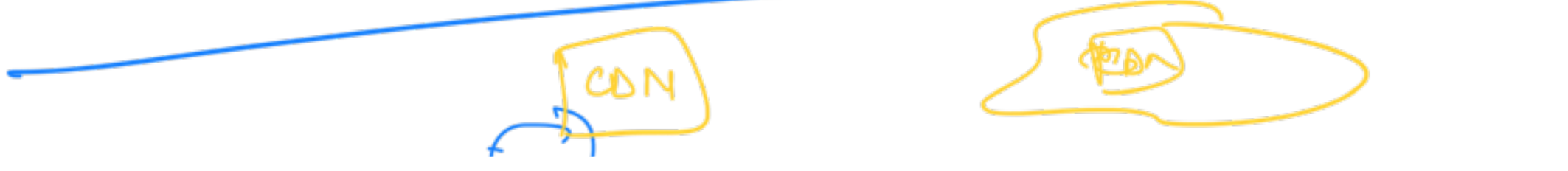
System Design

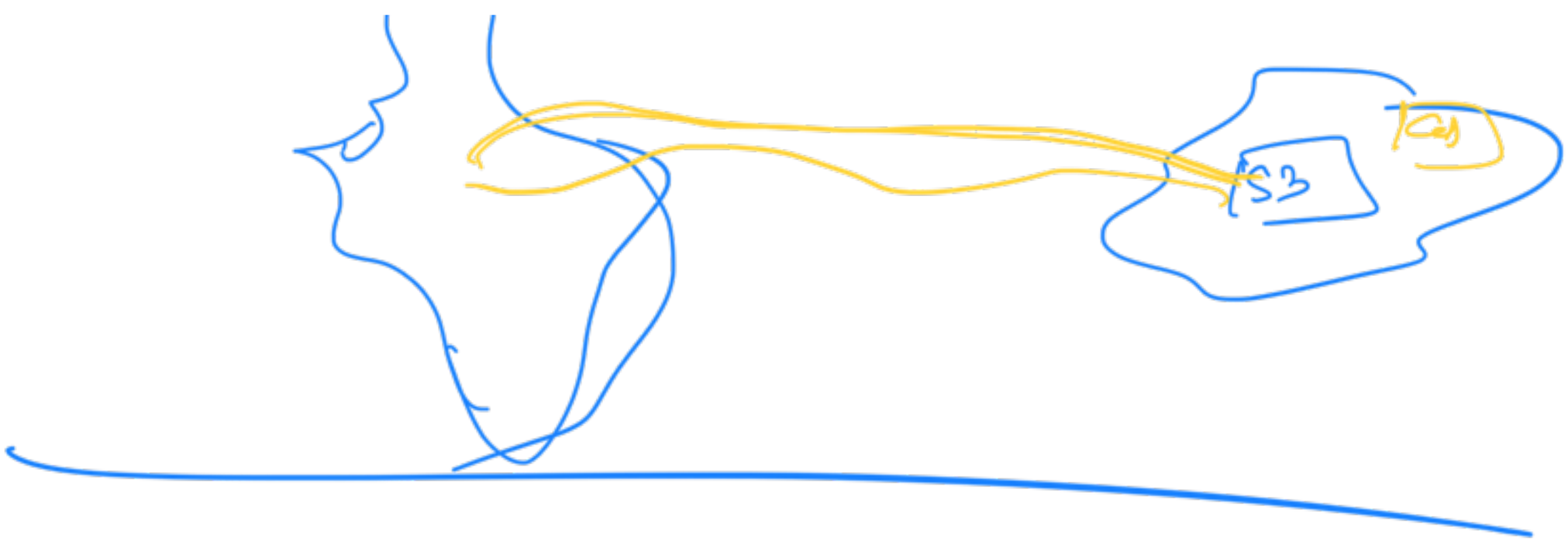
Bird's Eye View | Overview of a software system.





CDN: Content Delivery Networks





Infrastructure layers

HLD / System Design: Deals with the design and interaction of different infrastructural layers that work together to serve you

the app^{n} at a high

efficiency

< 10 Sec

How to approach HLD

① Understand the Problem Statement

a) Functional Requirements
(diff features of the system)

b) Non Functional Requirements
(Quality / Expectations)

Netflix

1. Auth
2. Streaming
3. Recommend

Whatsapp

1. Call
2. Multimedia Mess
3. Groups

CAP Theorem

→ Latency (delay)

→ Consistency V/S Availability

2 Back of Envelope Calculations

⇒ Real SWE Job ✗

⇒ Based on the requirements, quantify the scale

⇒ guesstimates

a) Size of data — Total — Worst Case

b) N/W Bandwidth ⇒ Avg

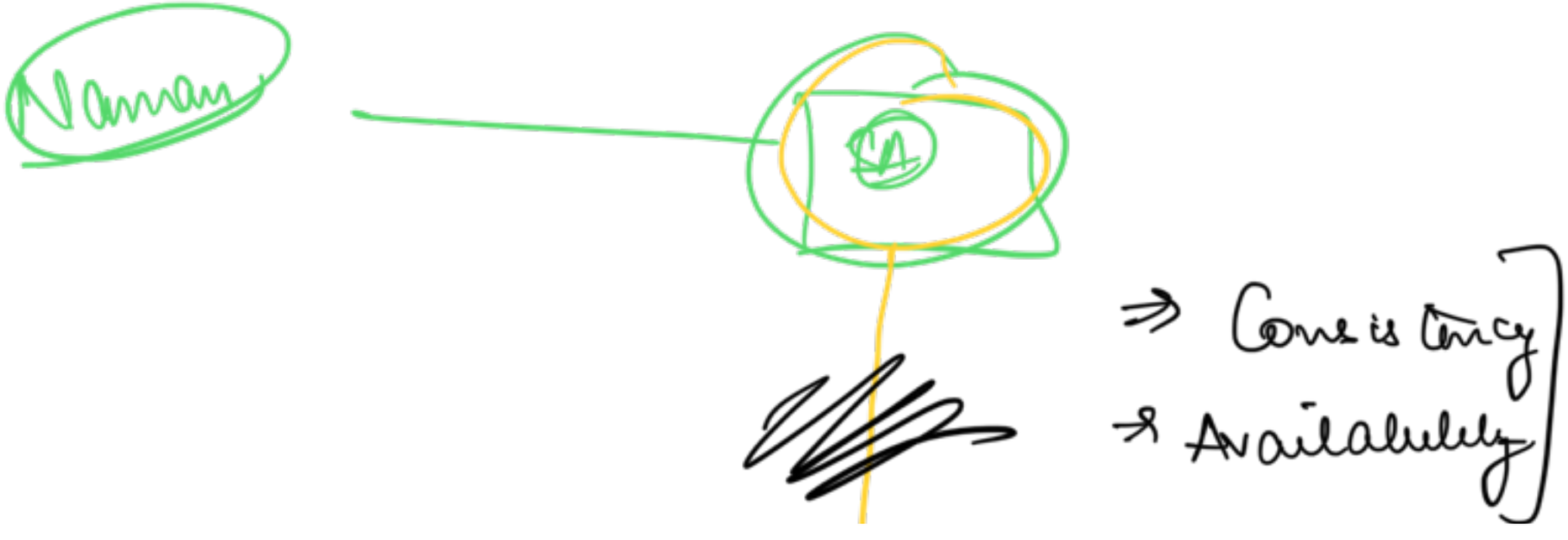
c) Concurrent Users

d) Usage of a feature

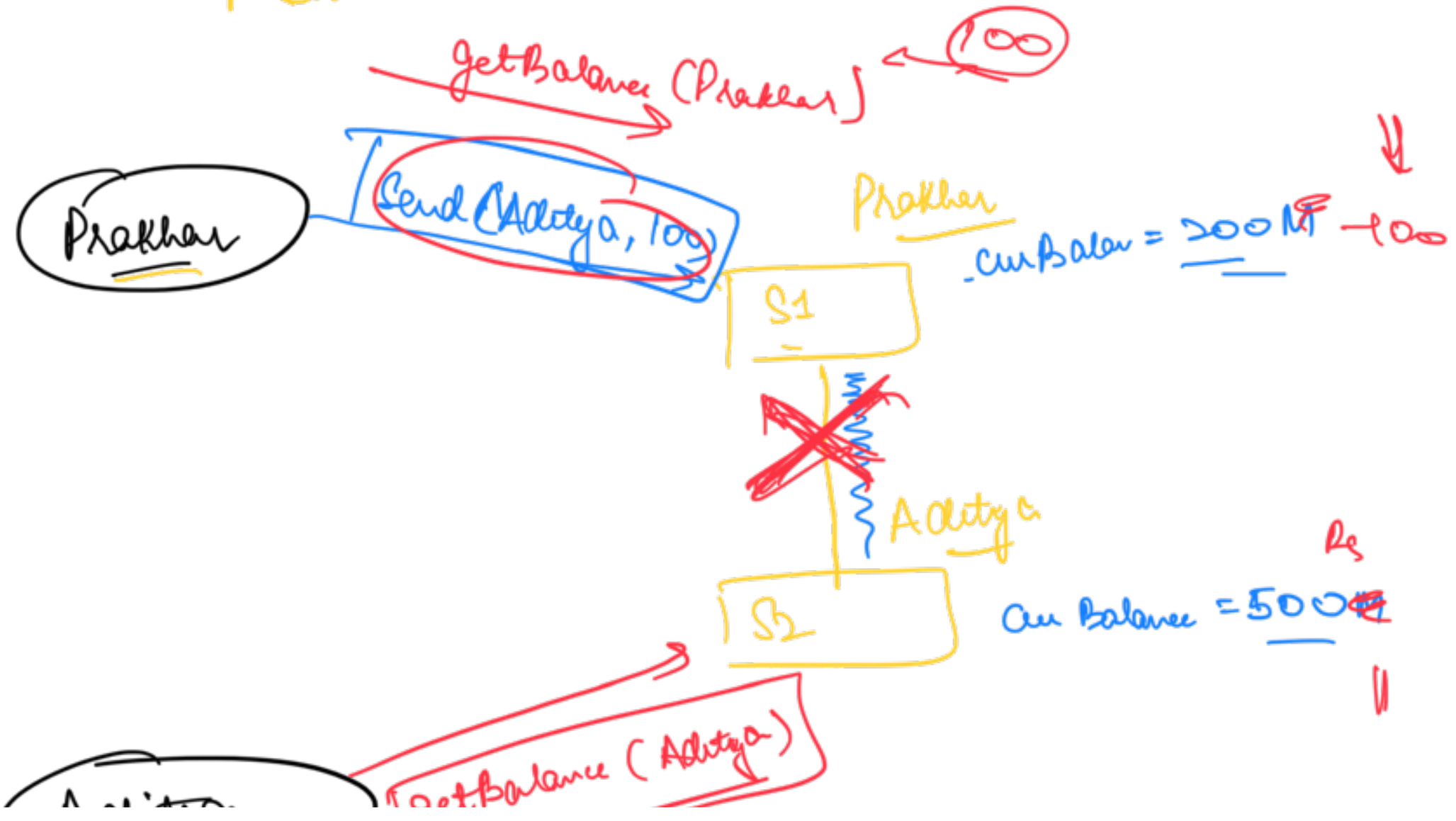
100 GB data

If you want to build an application that supports (P)artition Tolerance you can either have (C)onsistency or (A)vailability. Not Both.

20 GB data



P sends 100 rs to Aditya



PT ✓

Consistent ✗

Available ✓

CAP Theorem

AP ⇒ Availability ⇒ Hotstar, WhatsApp

OR

CP ⇒ Consistency ⇒ Bank

WhatsApp

Max^m # of users at same time

2BN people who use

1BN people will have day

5yr. → 10x open at same time

⇒ 100 Mn Concurrent user

⇒ 500 Mn

---

2 Bn people use daily

5y. of people

100 Mn

⇒ 2Bn Msgs

20 messages

1% of people ⇒ 500 msgs

---

③ APIs to Support

/send (receiver_id, message)

/watch (video-id)

(4) Schema ⇐

what are going to be diff entities that I will store and attributes for those entities

(Data Models)

Users

id	name	phoneNo	pic

(5) Design

a.) Start with a basic design

b.) Iterate (finding a bottleneck and resolving it)

Live Streaming Platform (YT Live, Hotstar, LS)

Functional Req

→ Streaming at different qualities (360p, 480p, 1080p)

→ Upload video

→ Replay

Non functional Req

① Streaming

→ Very low latency

→ High Availability

② Upload Video

→ High Availability

→ Very low latency

Eventual Consistency

Quantification

# of users watching concurrently = 20 Million

Upload bandwidth at one time = 20 TBPS

20 million * 1MB

20 Million

20×10^{12} Bytes

20×10^{3} GB

20 TB

1 Machine can Have 1 GBPS

1 Machine → 1 GBPS

1 GBPS → 1 Mb

20×10^3 GB → 20×10^3 Machine

# of Videos → 1 Lakh

every video → 4Q → 4lakh

Avg size of 1 vid → 30 min

Storage → Avg size / min = 60MB

$400 \times 10^3 \times 30 \times 60$ MB

$400 \times 30 \times 60$ GB

40K3 x 6 TB

720 TB

≈ 1PB

AWS S3

API & Schema Design & HLD (1 hr)

APIs

/upload

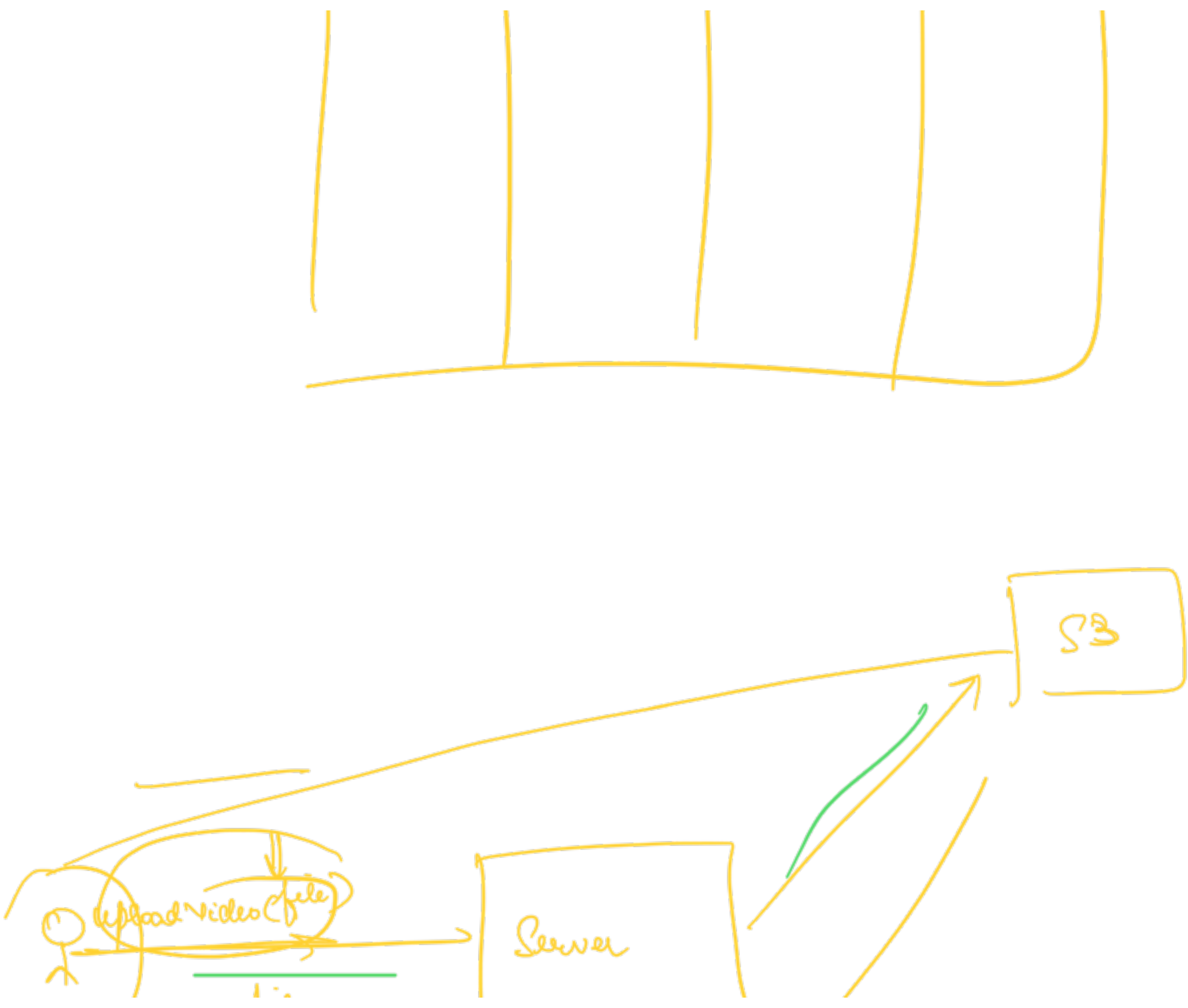
/watch

Schema

data about comments

Video Metadata

id	title	description	url
			s3_url

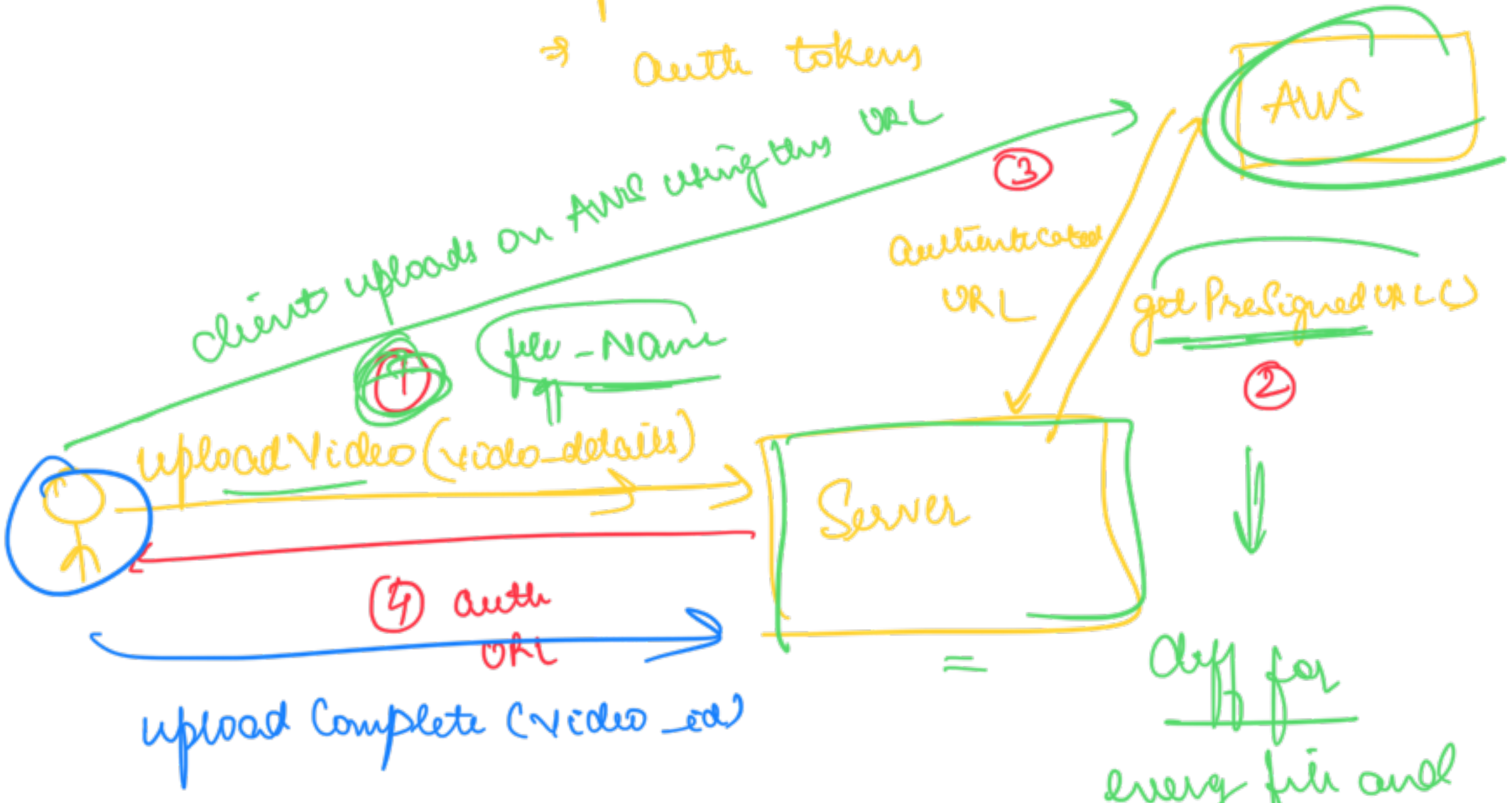

S3
Upload Video file
Server

file

Pre Signed URL :⇒ feature of AWS

⇒ provides authentication

⇒ auth tokens



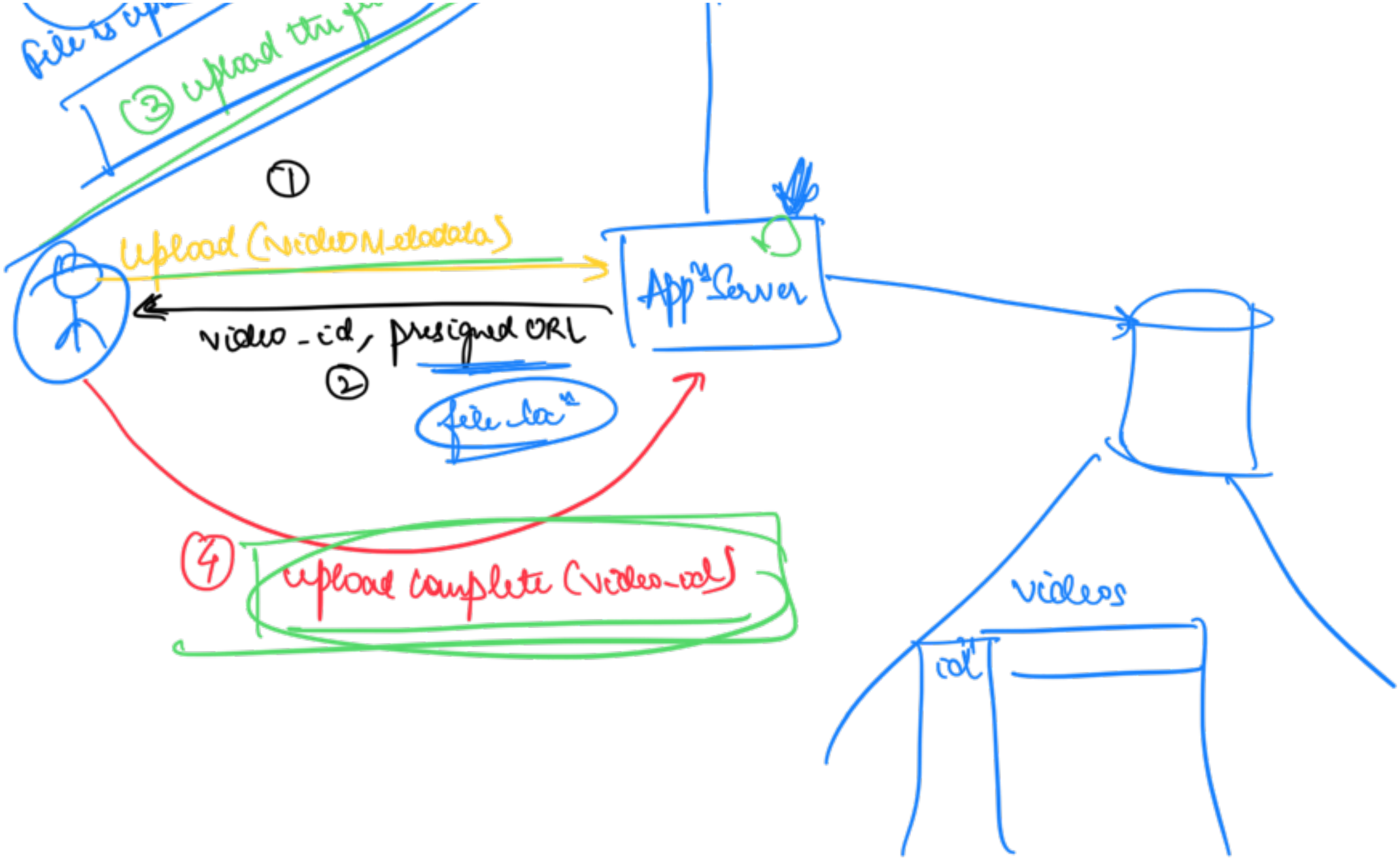
It is valid only for Console

Flow of Video upload

1. Send request to Server with video metadata.
   - ⇒ a:) VideoMD gets stored in DB
   - ⇒ b:) Server creates a pre signed URL and sends to client.
2. Client uploads directly to AWS.

I have saved dual bandwidth usage.

Multipart upload



When a video is uploaded,

① I might want to transcode it

compressing

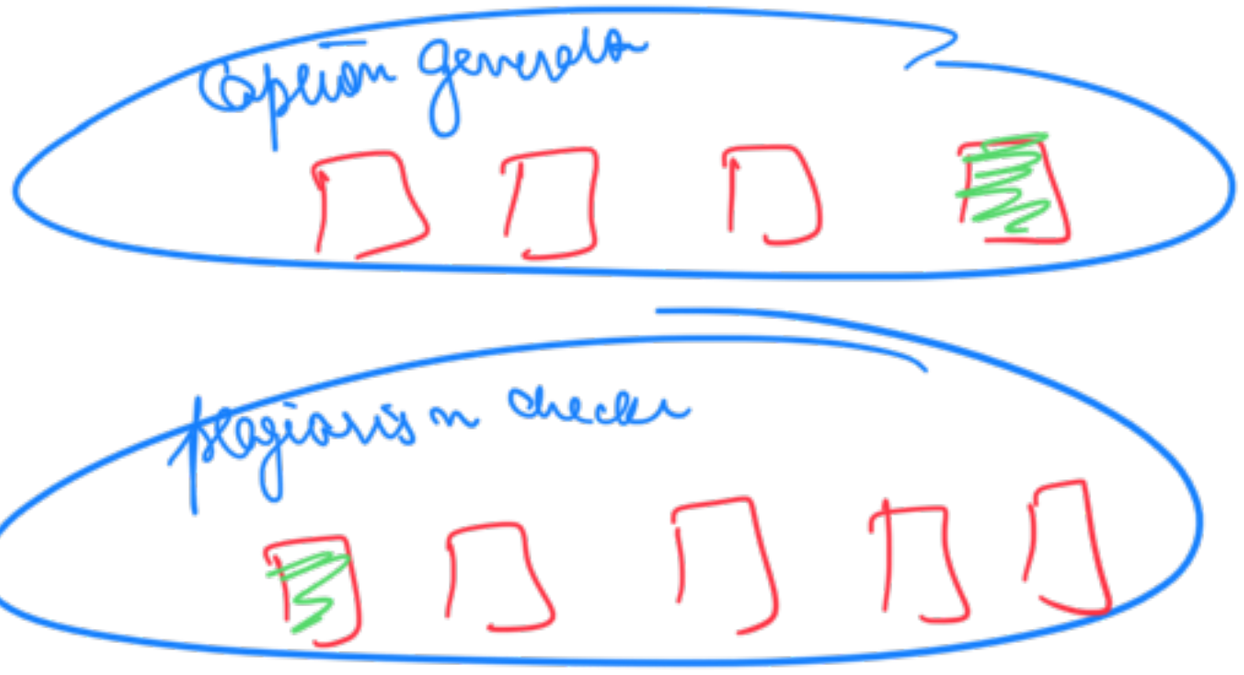
changing format

converting to diff resolns

Event Driven Systems (SNS by AWS, Kafka, Redis, Rabbit MQs)

Messaging Queue

Any event will be delivered to ONLY 1 server per group



What does each worker do?

(1) Downloads the raw video from AWS

(2) Compresses, converts to resolution

(2.1) Break the video into Multiple chunks

(3) Uploads to AWS (as chunk)

(4) Stores the details in DB

(along with Chunk details)

(every chunk of 10s)

DB

videos

id	name	description	url

DB

videos

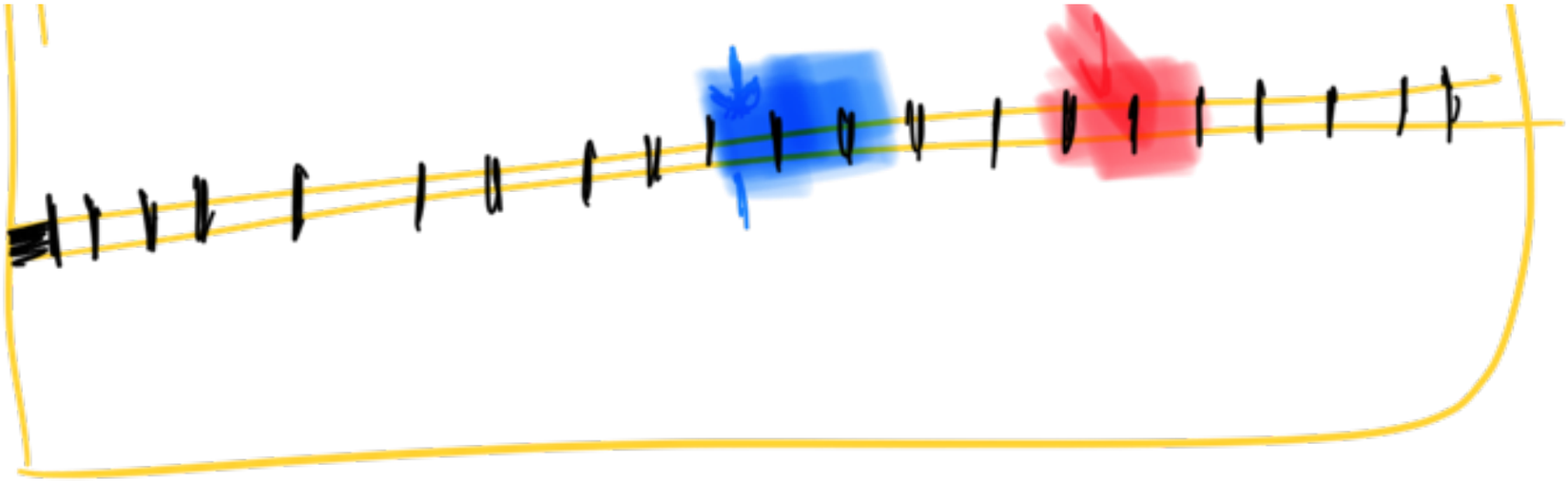
id	name	desc
1	Krab get	—

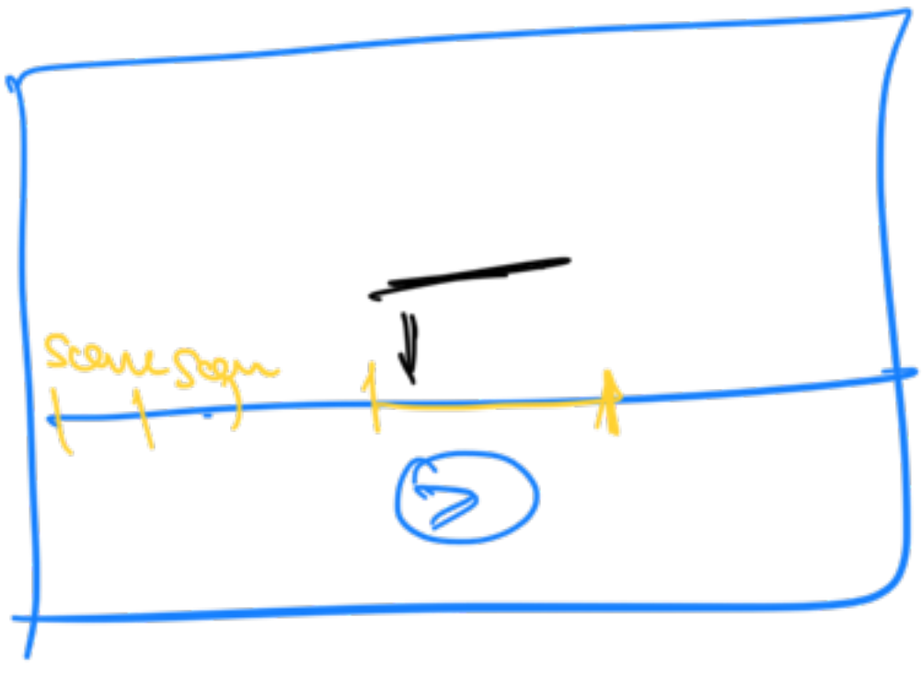
video - files

video_id	resolution	aws_url
1	RAW	—
1	360p	—

getVideoURL (video_id)

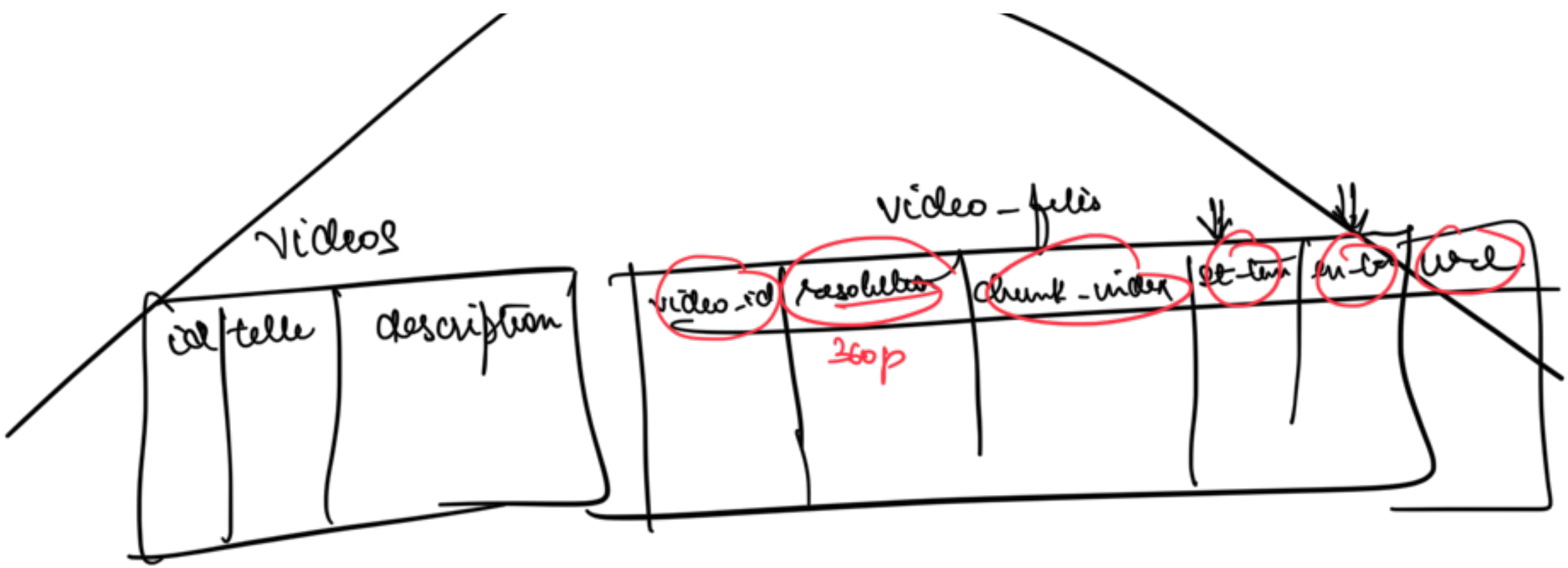
20:00

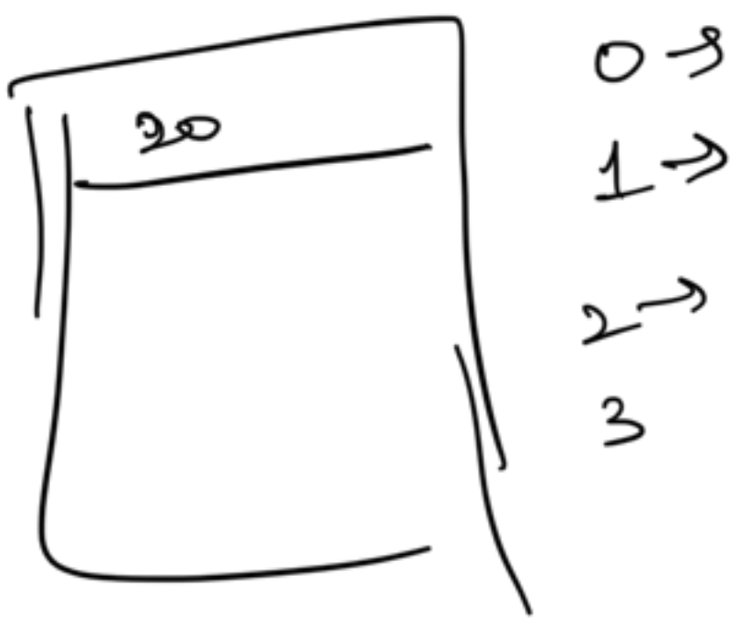


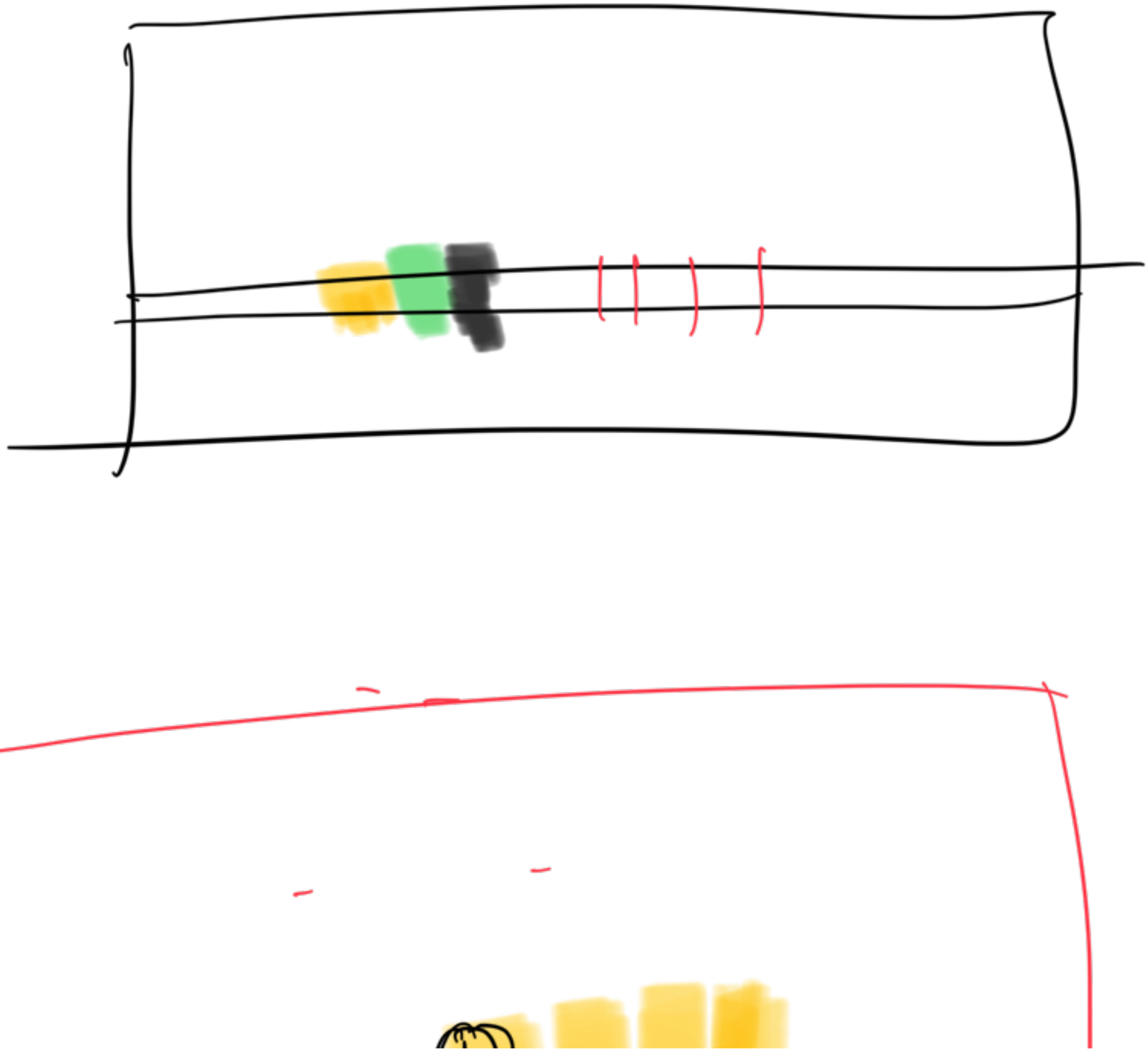

Scene Sign


Manual Process

DB

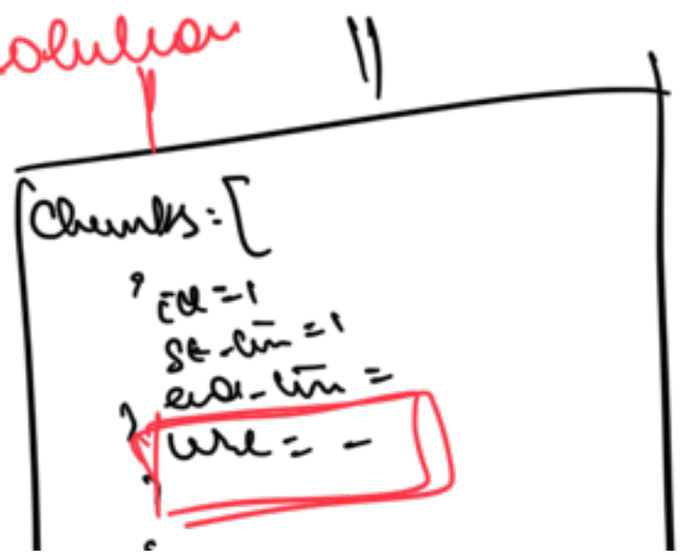

Videos
id
title
description
video_files
video_id
resolution
chunk_video
360p

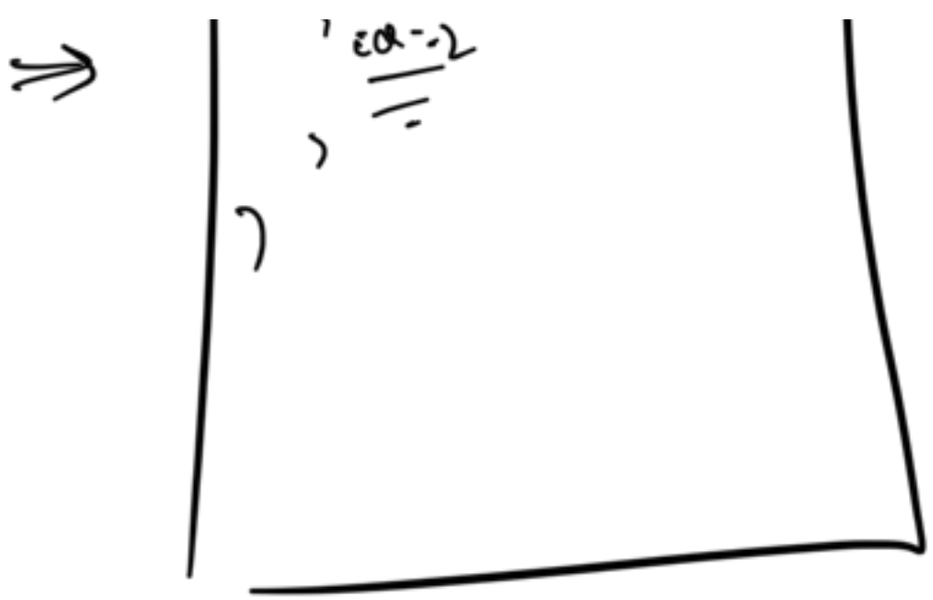

20
0
1
2
3



How Client Watches a Video:

1. Client opens video Player
2. Video Player sends Request:
   get Video Details (resolution, video id)
3. Server sends whole lists of chunks for that video for that resolution



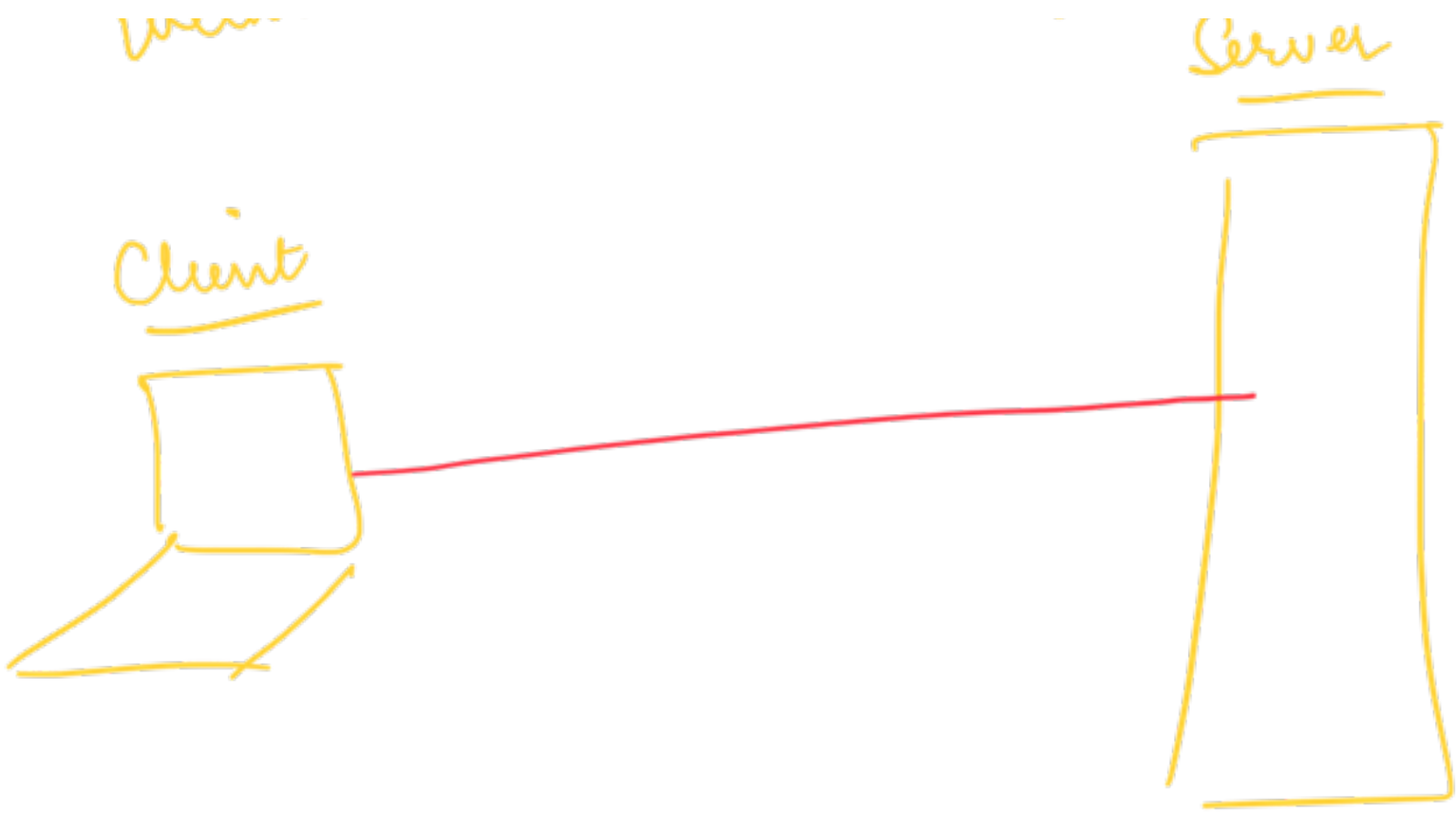


(4) For whatever timeframe we are at in the player sequentially player starts downloading that chunk

---

Web Sockets — into Picture

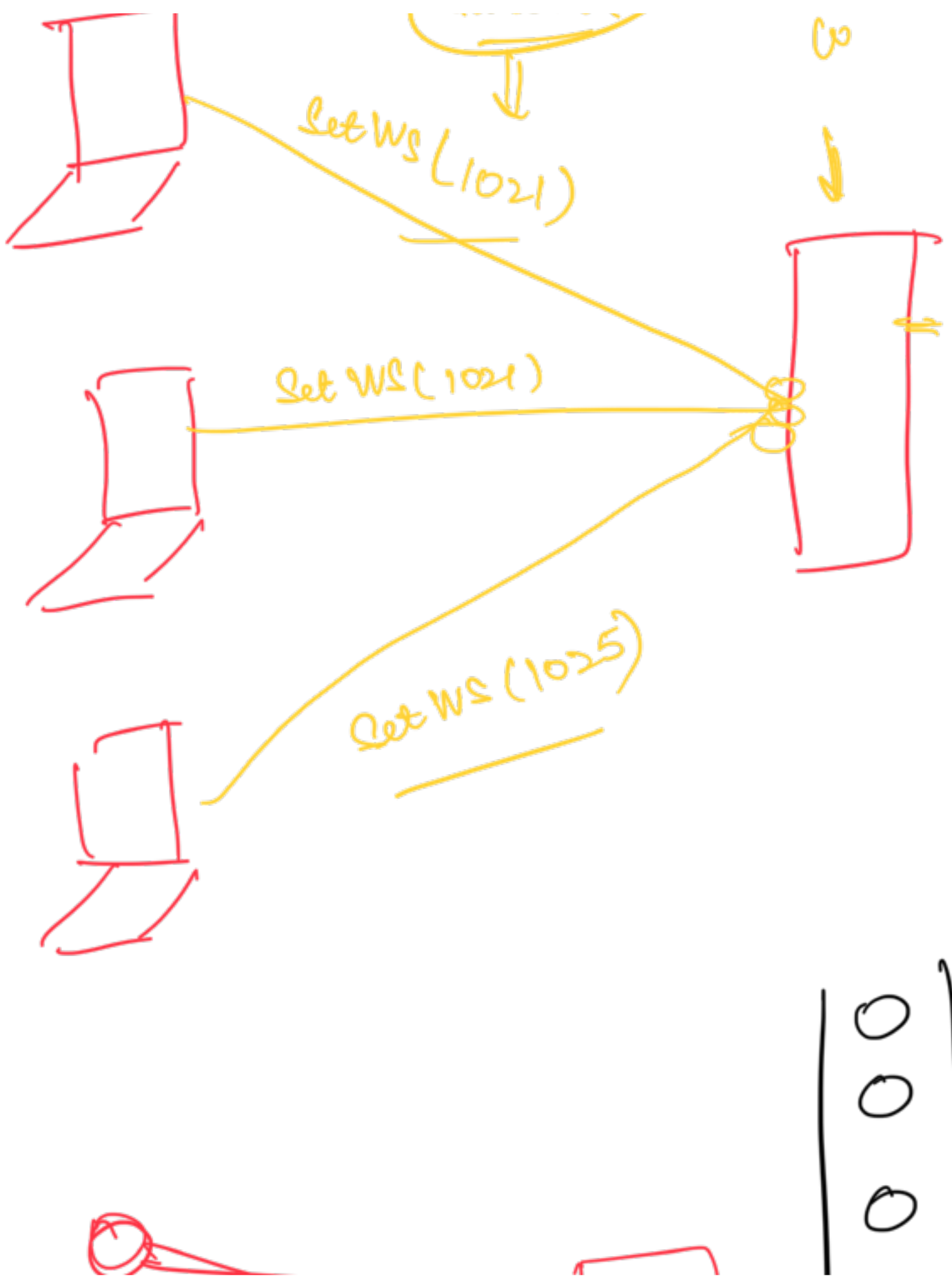
Websockets allow servers to send info to client without client requesting



When a video player loads:

① Client sets a ws:// with the server

② Whenever server has a new chunk for the video
Server sends that via a ws:// conn



Server in its Mem maintains list for every video

Map<video : List<ws>>

Whenever server gets a chunk for a video, Server sends the detail of the chunk to all ws conn for that vide

Each WS is a TCP

Normally 1 Server

Can handle

2 2sec

2×10^6 WS connects

5sec

AWS

(SQL) DB

1 top

7 top

360p

Server

123, 36p, 321, 00:20, 00:40, ure

WS

WS

WS

WS

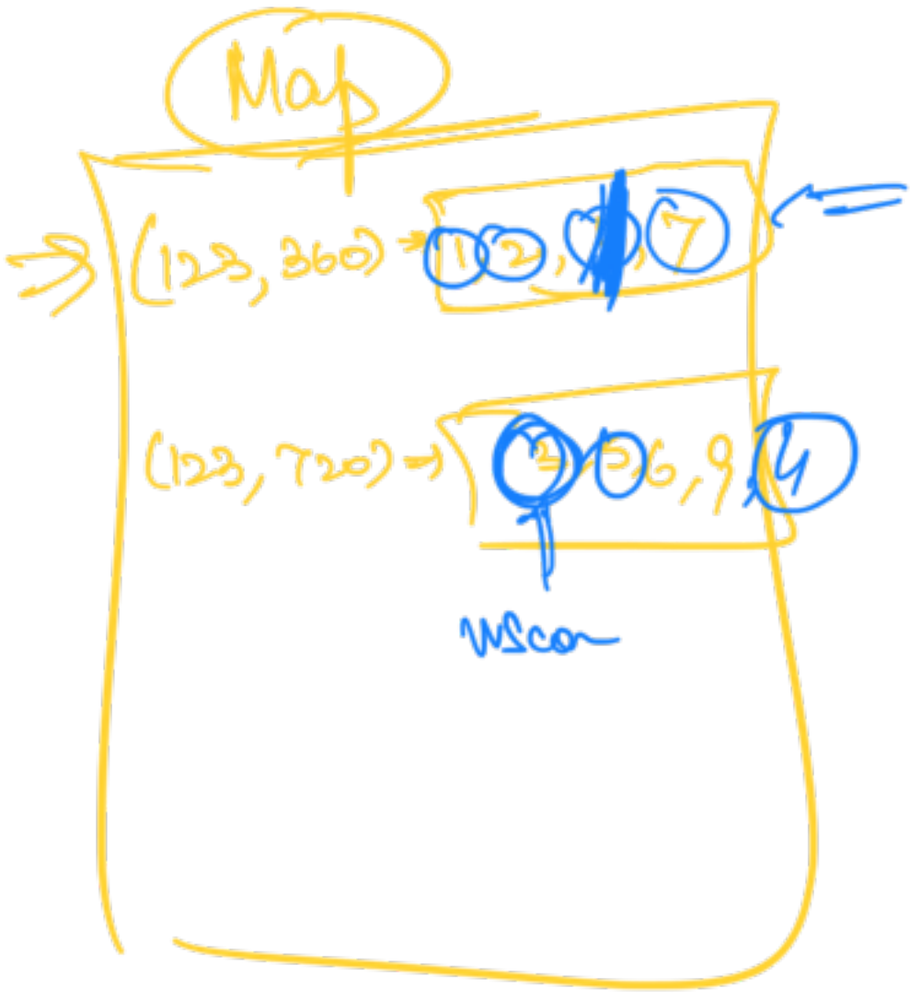
WS

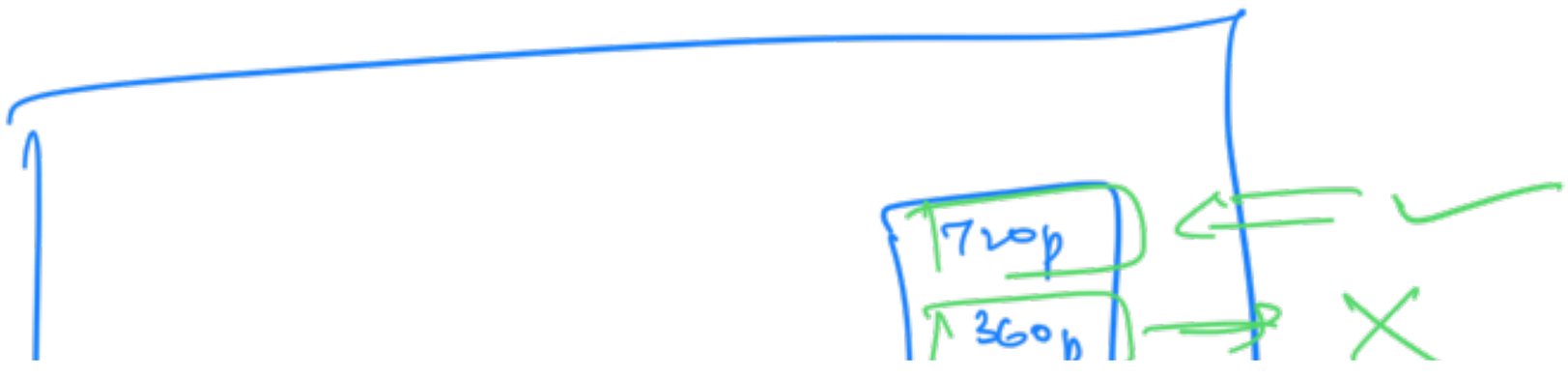
(123, 360p)

① All servers read every event

Map

(video_id, seq_id) → [WS connections]

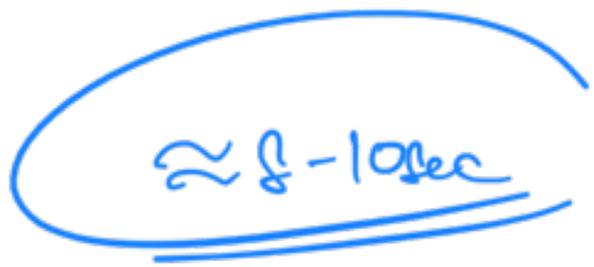

Map
(123, 360)
(123, 720)
WSco

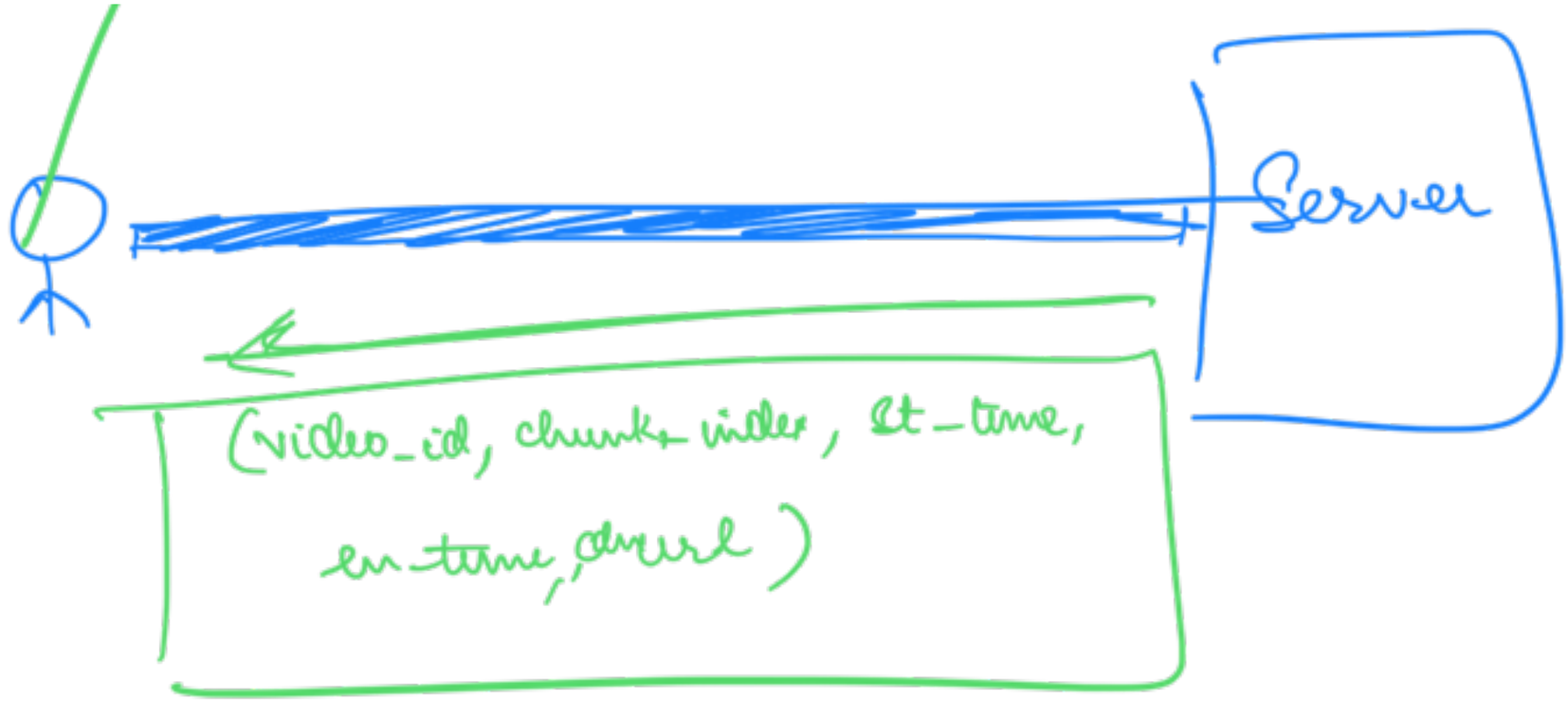

720p
360p


144p


5sec


AWS


≈ 5-10sec


Server
(video_id, chunk_index, st_time,
en_time, durl)


1080p
720p
360p
144p