Agenda

- Problem Statement
- Minimal Viable Product (MVP)
- Scale Estimation
- Design Goals
- ~~API~~
- System Design

Problem Statement → design an OTT (over-the-top) platform

- video streaming (consumption)
  - existing / stored video → fav show (game of thrones)
  - Streaming live event
- uploading videos
  - live events
  - Stored videos
- → Live messaging during Matches ?

- Recommender Systems
- Payment Processing
- Analytics
- Advertisement

} ✗ not our focus.

Requirements

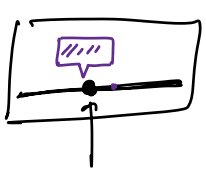
Functional

- upload video
- stream live
- watch video
- watch live stream

} play / pause / video resolution / rewind / skip to timestamp

User can manually set resolution

auto decide video resolution.



Non-functional

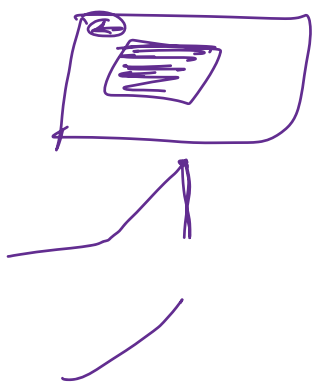
- video should not pause (buffer) while streaming
- Availability + Eventual Consistency.
- prevent parts of video from getting dropped.

→ ability to scale quickly (1 M new users every minute)

Scale

estimate # users → DAU → # API calls / day
→ QPS → Peak QPS
↳ Queries / sec

→ 25+ Million users streaming concurrently (50 M)



→ 1M API requests / sec

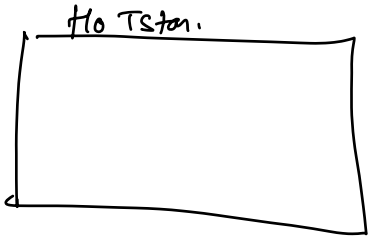
→ 10 Tbps of bandwidth
↳ 75% of India's total bandwidth.

→ 10B+ clickstream messages per day

→ 100+ hours of live video transcoding every day.

System Design

Watching movies/shows



each user will have a diff bandwidth.

↳ user can manually specify resolution → 1440p

↳ automatically detect the correct resolution.

ABS (Adaptive bitrate streaming

↳ measure download speeds as user is streaming video.

↳ detect user device → decide the max resolution.

Instant smooth streaming.

John Wick 4

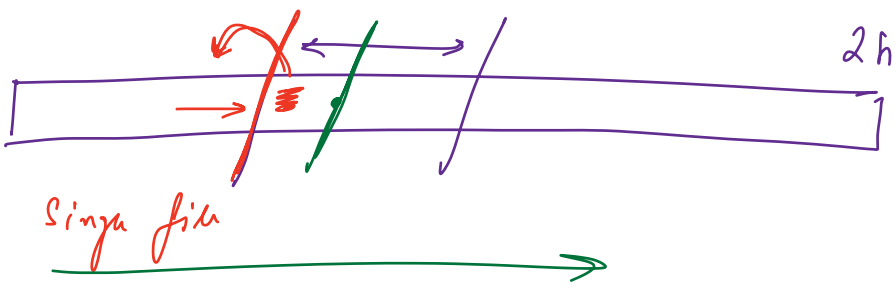
↳ 4 GB

FHD (1080p)

split file into chunks of size →

10ms
10ms
10s
2min
10min
2 hour

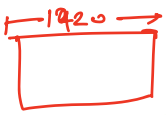
GOT



Codecs (mp4, m4v, flv, _avi, amv)

Resolutions

144p
240p
360p
480p
720p (HD)
1080p (FHD)
2K
1440p (QHD)
4K
8K



RAW

1920 * 1080 pixels ← 1080p

≈ 2×10^6 pixels

↓

3 bytes

6×10^6 bytes = 6Mb per frame

fps → frames per second.

24 fps (most old movies)

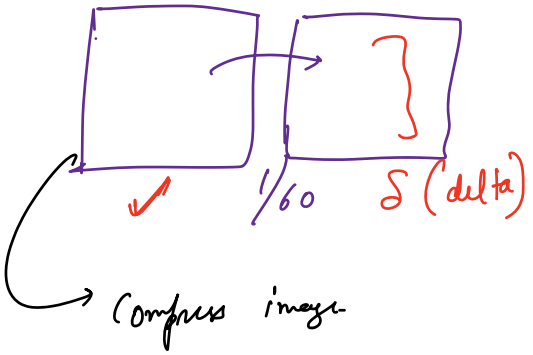
30 fps

60 fps → modern movies (Avatar)

120 fps

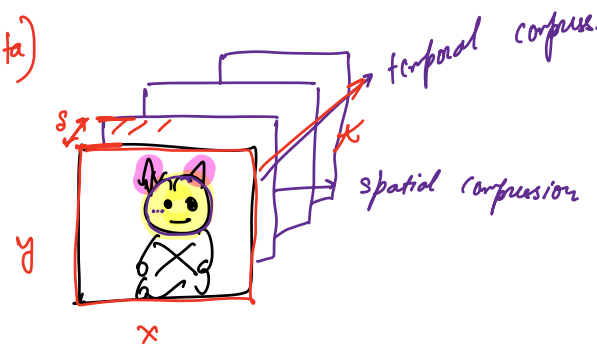
every second = 6Mb * 60

= 360Mb /s



2 hour movie = 2 * 360 * 3600

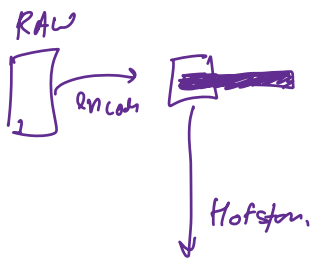
1 TB

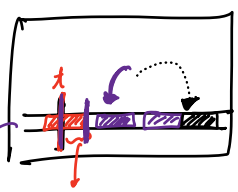


JW4.mp4 → original file (2 hour)

- JW4_1.mp4 (2 mins)
- JW4_2.mp4 (2 mins)
- ...

happens for all resolutions.





client → bandwidth

→ Chunks stored in backend. (meta data)

how much downloaded

time is remaining. ⇒ how much time to download next chunk

fetch all meta data of the video.

metadata

video → content
name
director
cast
genre

1 Kb

chunk

8	8	4	4	4	100	100
content-id	chunk-id	resolution	start-time	end-time	thumbnail-url	video url
JWH	0	720		→ diff.	↓	↓
	0	1080			links to CDN	links to CDN
	1	720				
	1	1080				
	2	720				
	2	1080				

60 chunks.

250 bytes.

any video → 2 10 Kb

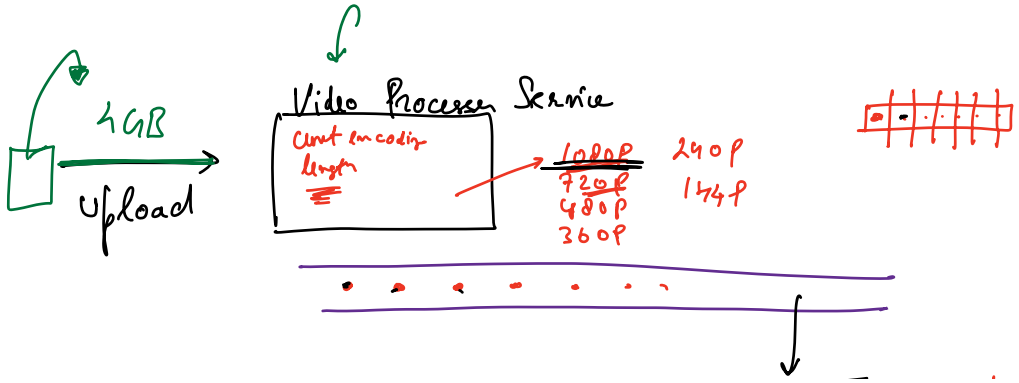
total metadata.

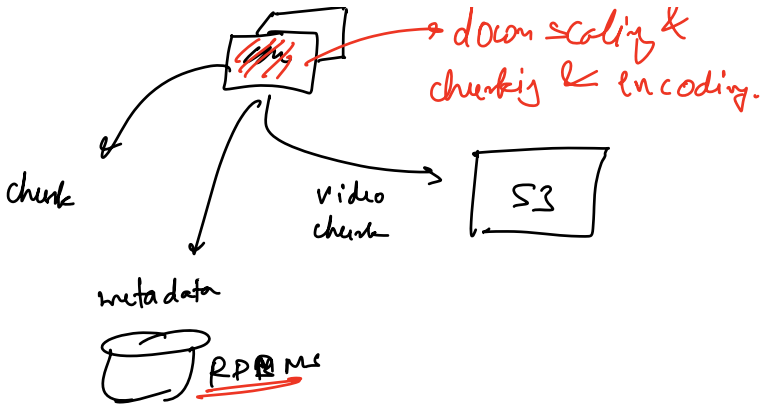
60 * 250 Bytes.

1 million videos = 1M * 10 Kb = 10 GB

RDBMS

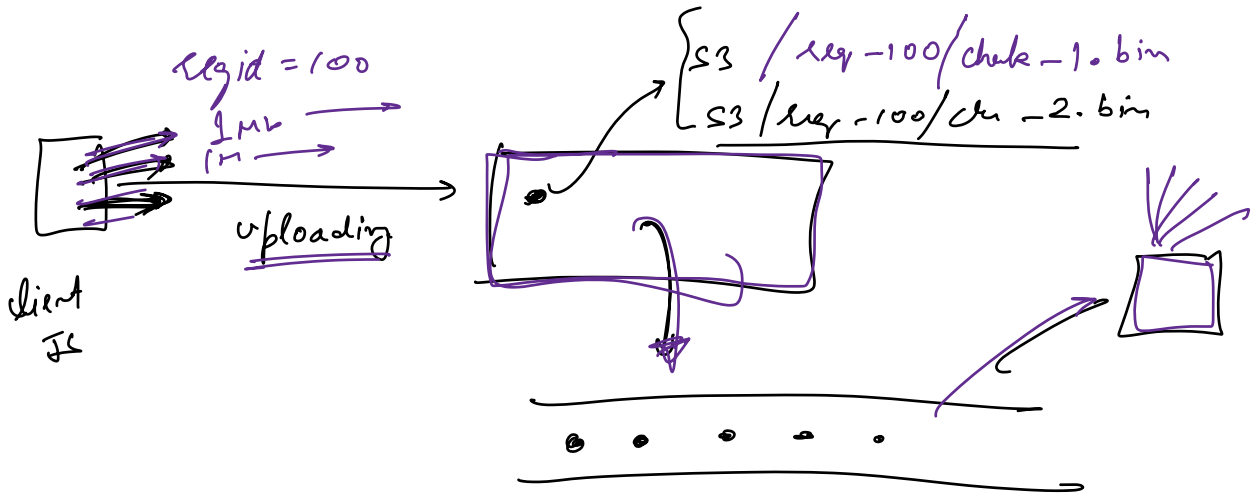
heavy cache on top.





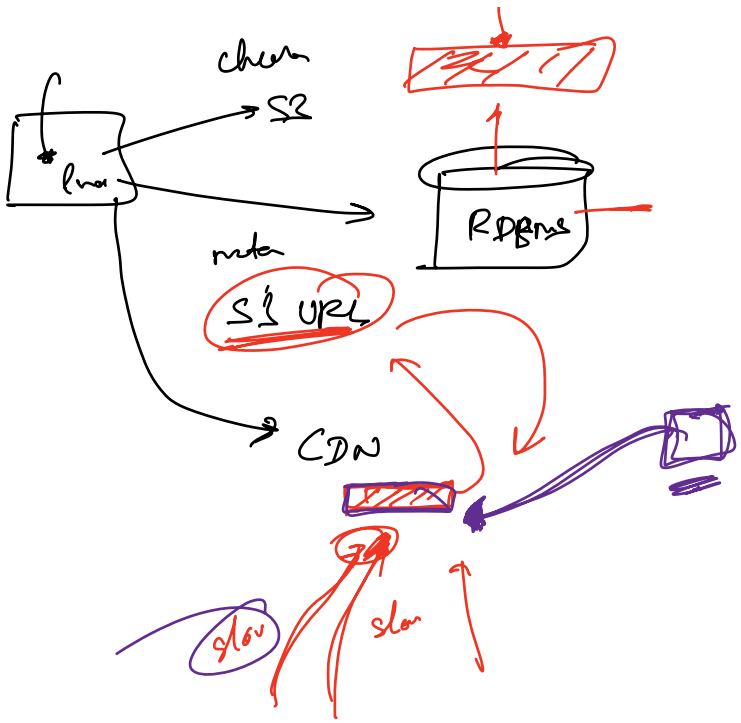
256 MB chunks in Hbase

Name Node



---

1. we don't want our server to be overloaded.
2. static content → easy to cache
3. CDN → any cast



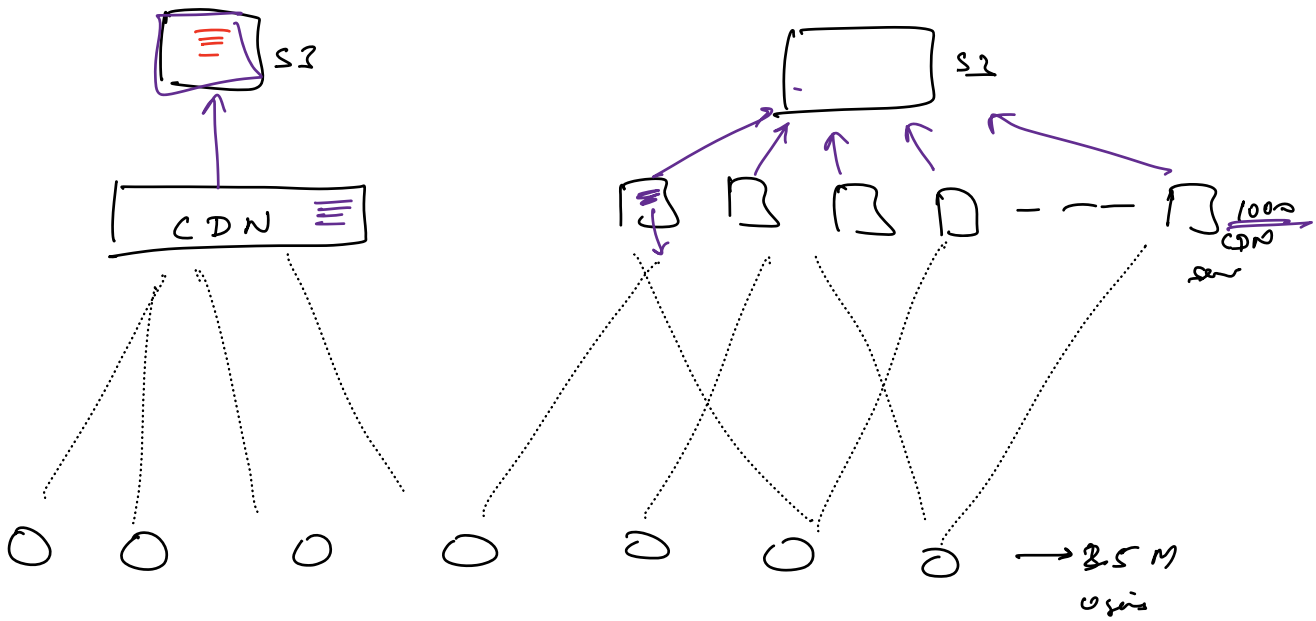
Videos which are pre-stored.

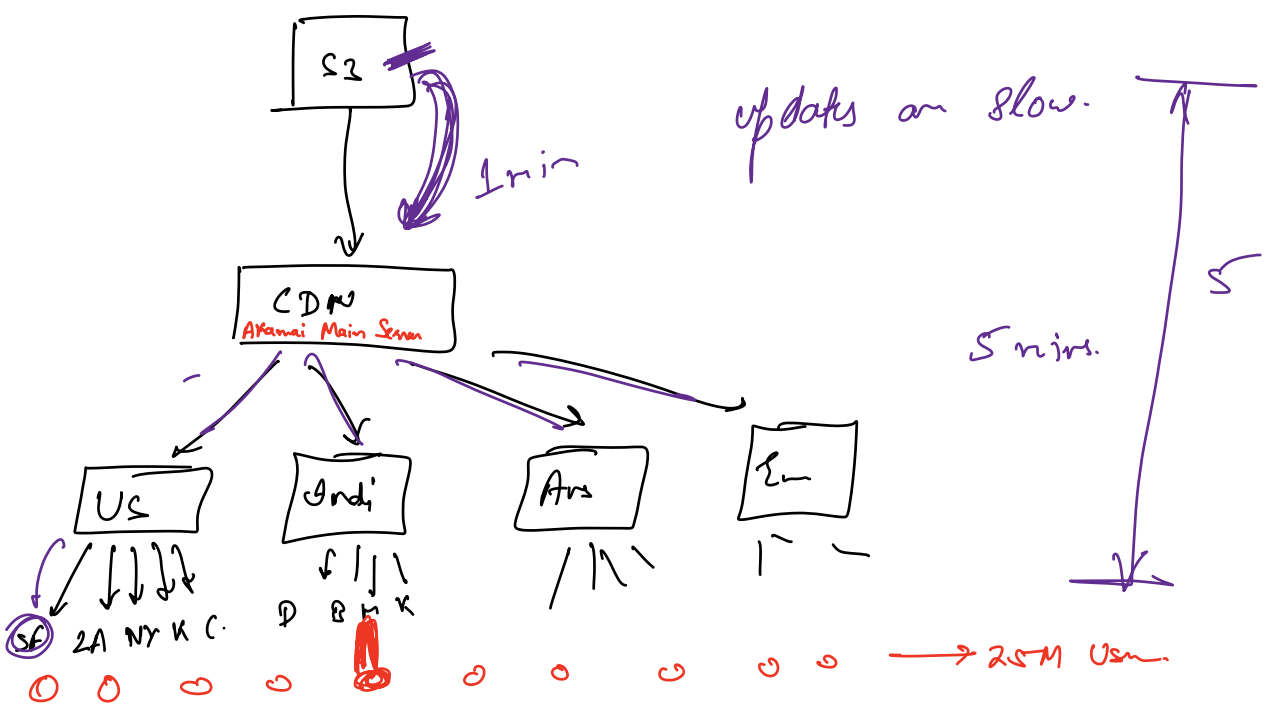
10:39 → 10:50

~ 50K students

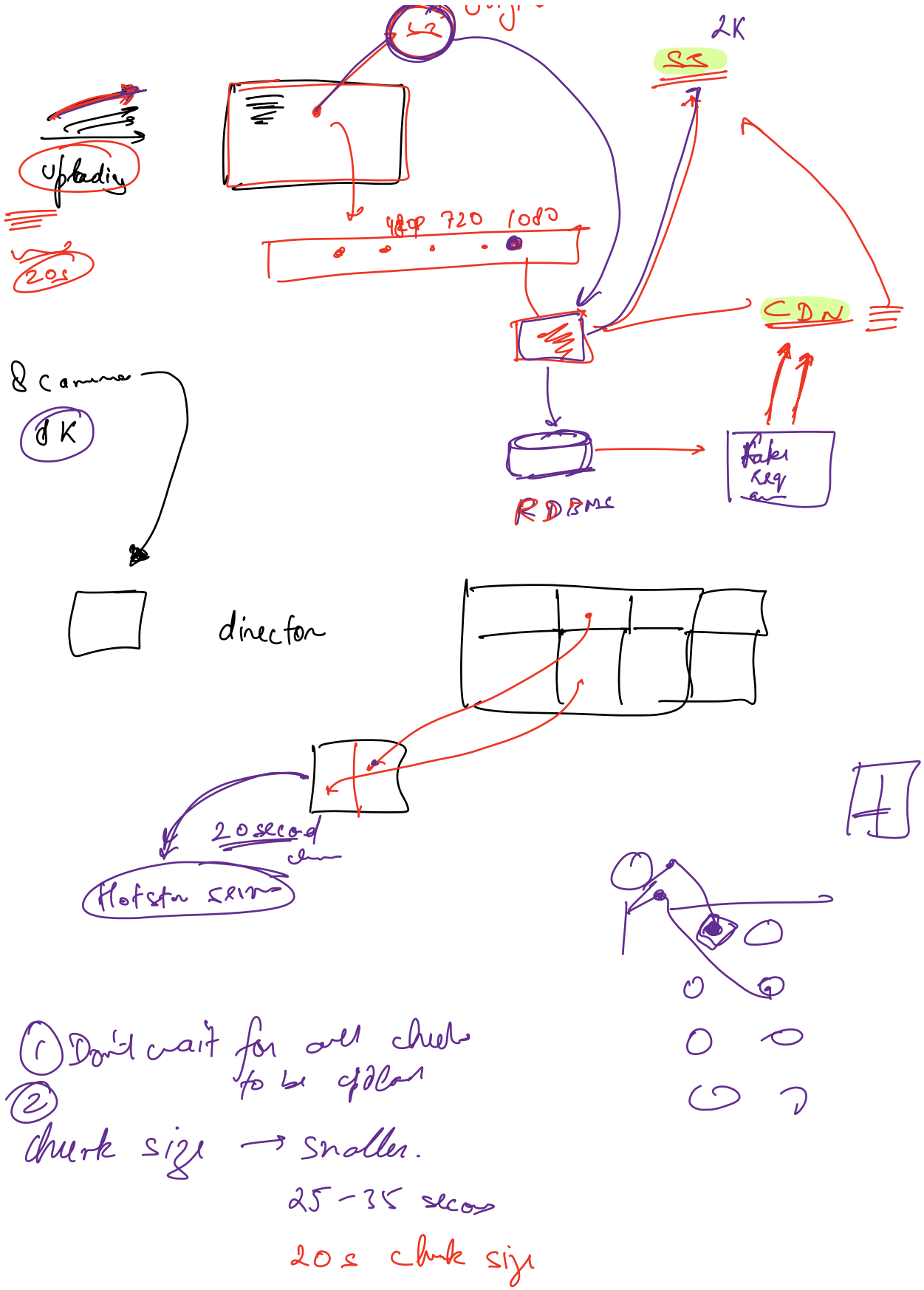
Stream each day.

Live streaming → 25M people might be consuming same video stream.





Mixing



(1) Don't wait for all chunks to be uploaded

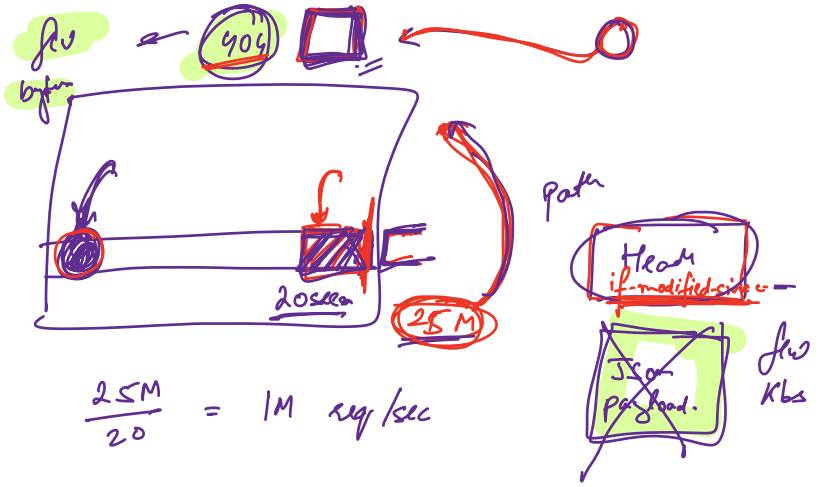
(2) chunk size → smaller.

25-35 secs

20s chunk size

(2)

1M req/s



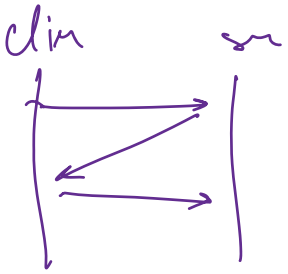
$\frac{25M}{20}$ = 1M req/sec

(1) /fetch_latest_chunk

if_last_modified_since = chk ts

TCP → prevents data loss (reliable)
↳ high latency
encryption

UDP → can drop packets
↳ lower latency



(1) handshake happens with every TCP connection

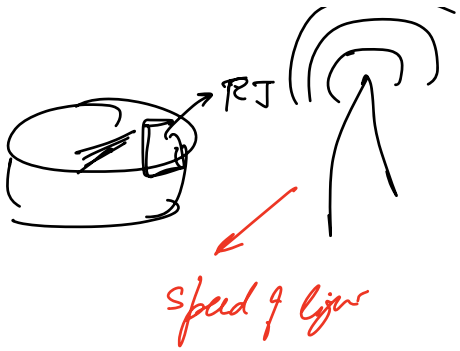
(2) TCP drops connection often sending data

RTMP (Realtime Message Protocol)

Persistent – 2way connection.

lags by ~5 mins.

↳ 1.5 min



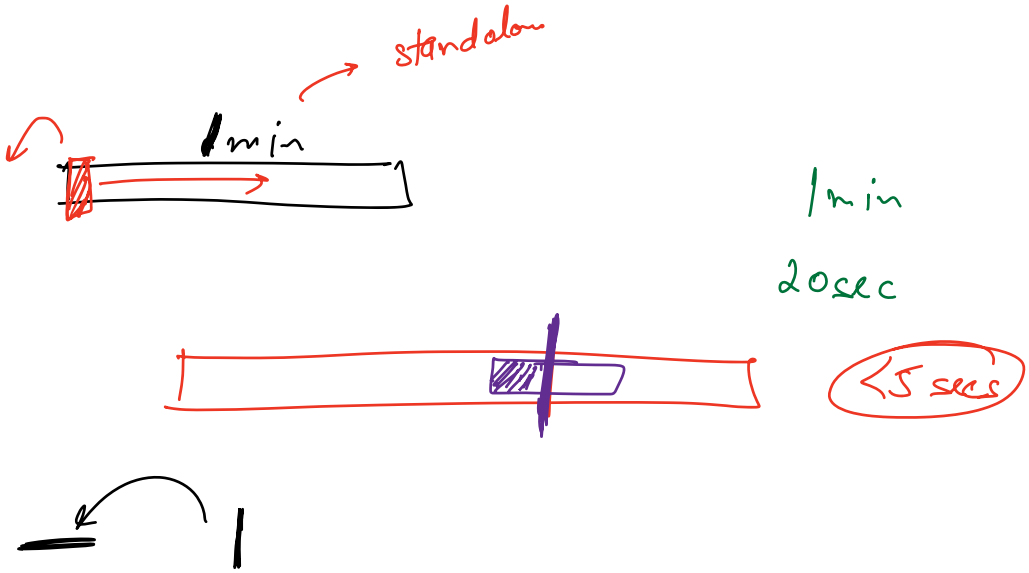
NSE → 20M trades in a day.

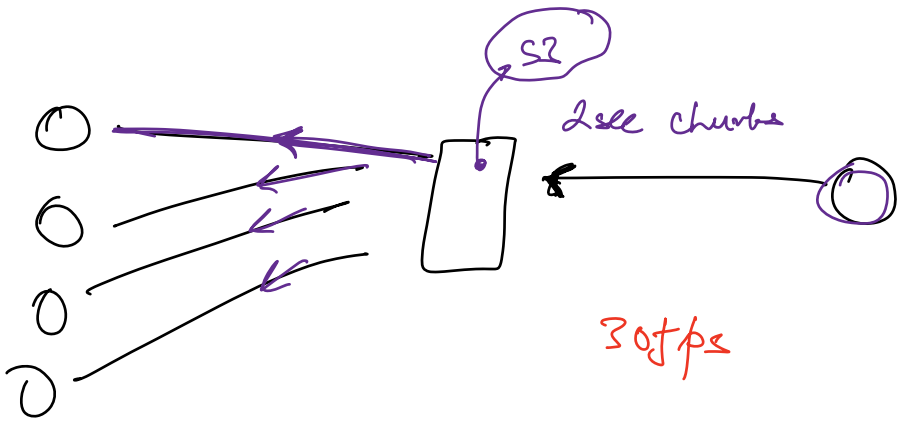
6 hours

↳ 5000 rq/sec

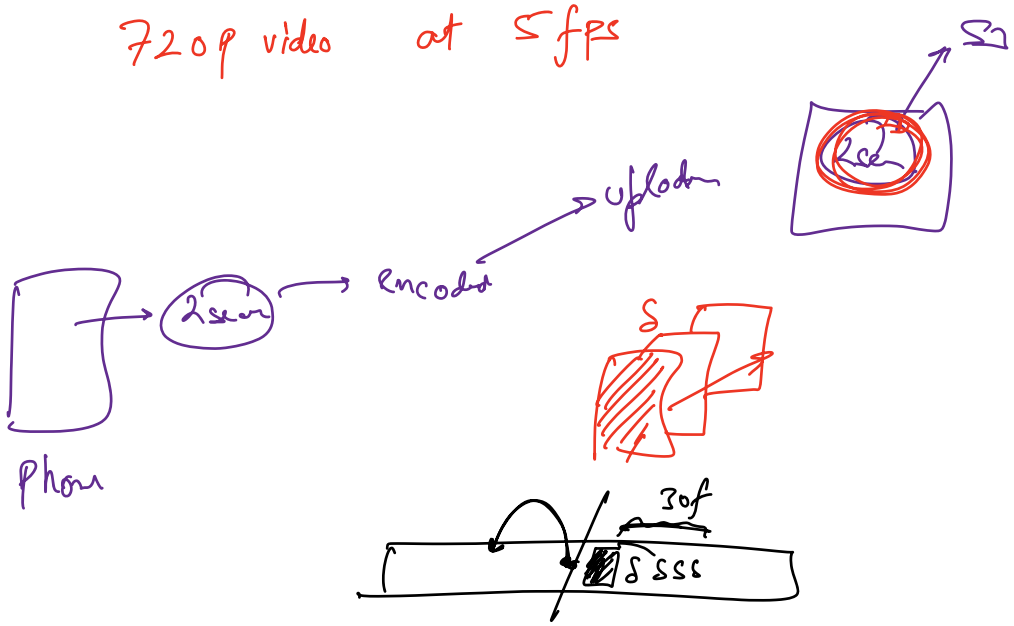
Netflix / Hotstar / OTT 1 Million

Youtube

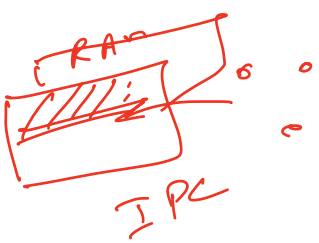


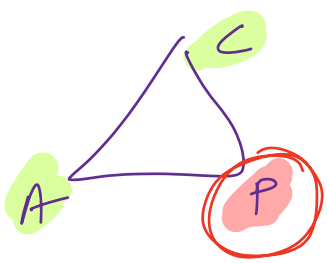


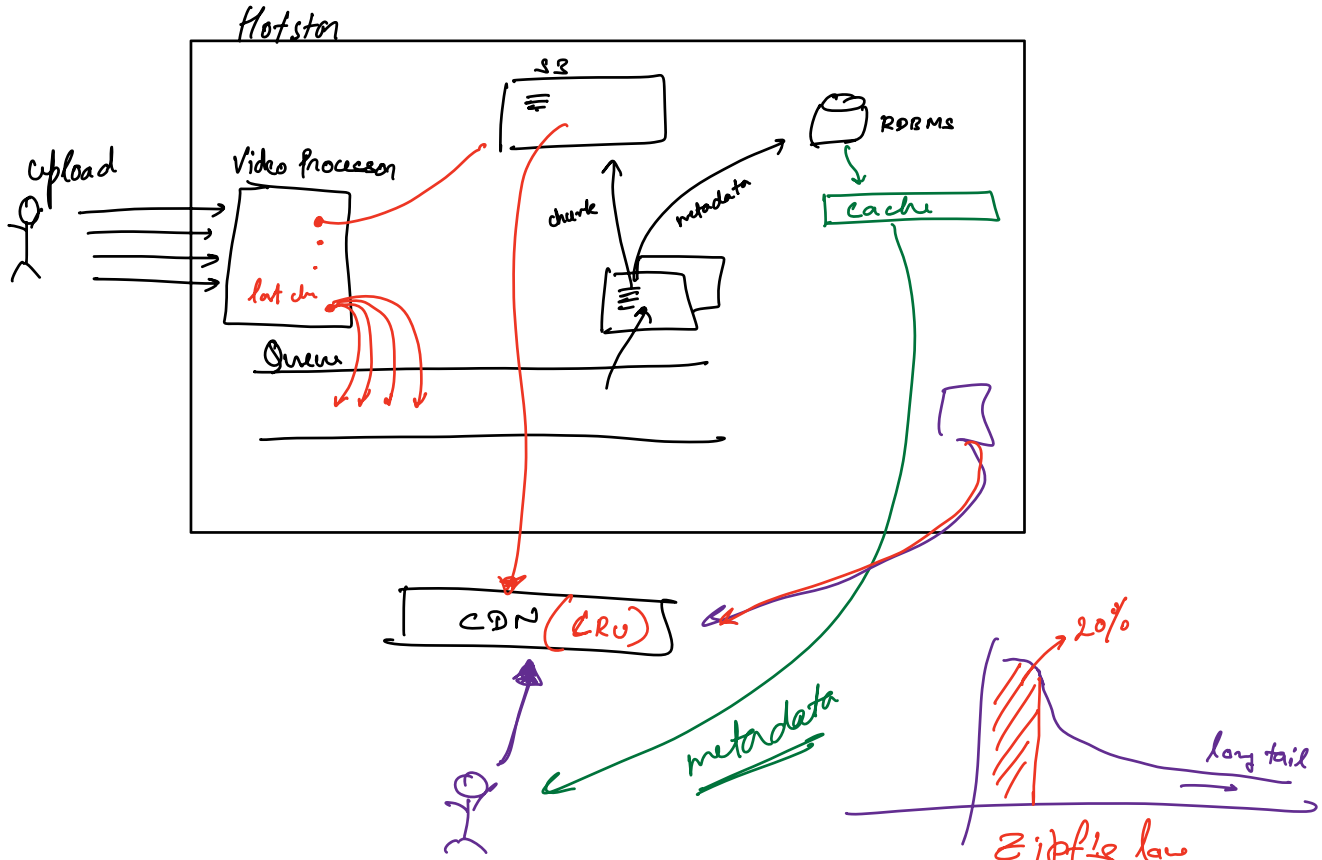
720p video at 5fps



Stock Exchange
v.v.v. low latency.







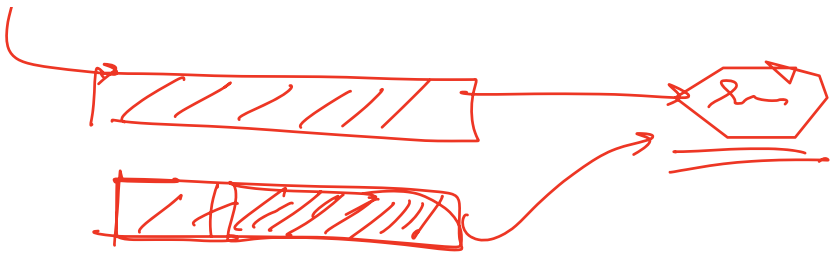
Pdf → 2 MB → open. download

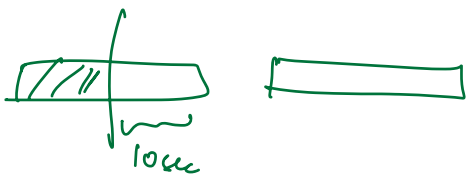
Stream → 2 GB → 10sec → view

10sec

streaming.

1 byte → 1s




10sec