→ Purpose of HLD

→ Steps to approach HLD problem

→ Problem Statement

→ Minimal Viable Product [MVP]

→ API Design

→ Scale Estimation

→ Design goals

→ System Design

HLD → High level Design

- challenges that come with scale
- designing a system
  - a billion dollar
  - millions req/sec
  - Petabytes of data
  - 10000+ servers

Planet Scale Design

SDE 2 → above Staff Engineer / Architect → 3 or PA

3 to 5

[8-10]

How do you approach HLD interviews?

→ vague problem statement → expectations ⇒ think, talk & ask

→ expected to design a system that can handle load of a huge scale

→ Problem Statement

Shorts | Reels | Tiktoks

HLD of Short content platforms

- Storage
  - Replication
  - fault Tolerance
  - Consistency
- Handling Scale
  - Sharding
  - Load Balancing
  - Caching
- Video
  - encode
  - Streaming
  - processing

→ MVP → Minimal Viable Product

↳ minimal possible features to build a usable product

now

MVP

* upload video
* play video
* login / signup
* compression
* limit video length
* thumbnail
* title

future

Good features

* like / share / comment
* save video
* quality selection
* delete
* search & explore

distant future

Very good features

* content suggestion
* moderation
* edit video
* ad support
* video speed

* API Design:

  how will the client / user will interact with the system

* login (username, password)
* signup (username, password)
* uploadVideo (user-id, video-file, title)
* playVideo (video-id, start-time, resolution)
  - start-time → DS
  - resolution → Flag
* view_thumbnail (user-id)
* view_thumbnail (video-id)

→ Scale Estimation :

: estimate the maxm scale for your system

Scaler ⇒ HLD ⇒ Google | Amazon | Netflix | Fr.ckr | Whatsapp etc.

→ What % of Internet is video Streaming :

: How much % is Netflix of global internet bandwidth ⇒ 15 %

: 40 % of internet is video streaming

→ Back of the envelope estimates :

↓

Quick & dirty

i) Amount of data in total
ii) Number of req. / sec
iii) Number of servers needed

no. of users on our platform

⇒ No. of users ≈ 4 Billion

World population ⇒ 8B

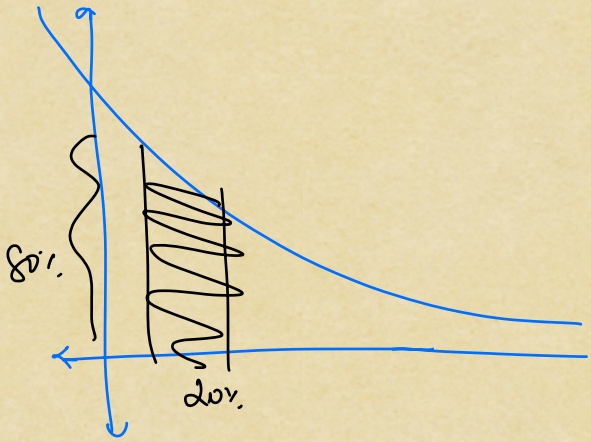
→ Pareto Principle

80-20

DAU → Daily Active Users

⇒ 20% of 4B

⇒ 800M

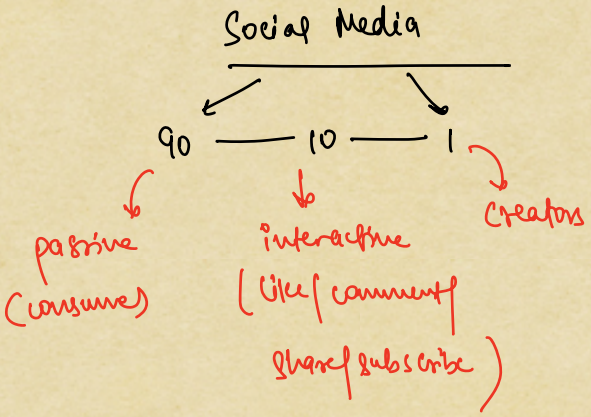


Content creators

1% of DAU

⇒ 1% of 800 M

⇒ 8M



⇒ Amount of data generated

8M users upload video daily

per day ⇒ 3-5 videos.

Videos/day ⇒ 5 x 8M = 40M

max length video ⇒ (< 1 min) = 100 MB

4K

→ data being generated per day :

40M x 100 MB

⇒ $40 \times 10^6 \times 100 \times 10^6$ bytes

⇒ 4×10^5 bytes

⇒ 4 petabytes

⇒ 4,000,000 GB / day

→ 5yrs = 5 x 365 x 4 PB

⇒ 7300 PB ≈ 7.3 EB

↓

(1 machine)

partitions [distribute] across multiple machines ⇒ Sharding

⇒ Number of requests / second :-

Read heavy ✓

Write heavy

* most of the requests will be read requests
* DAU ⇒ 500M
* each user watches 50 videos / day

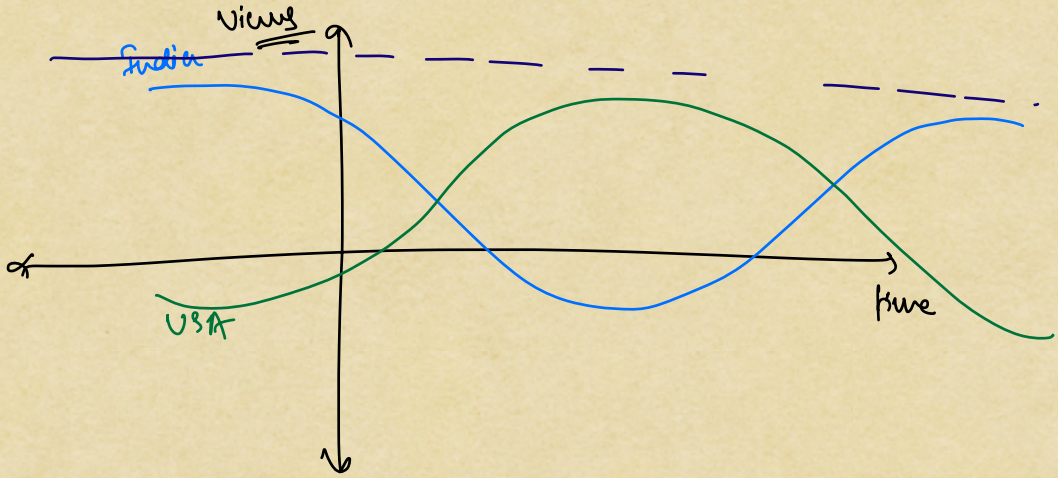
⇒ total views per day ⇒ 50 X 800M

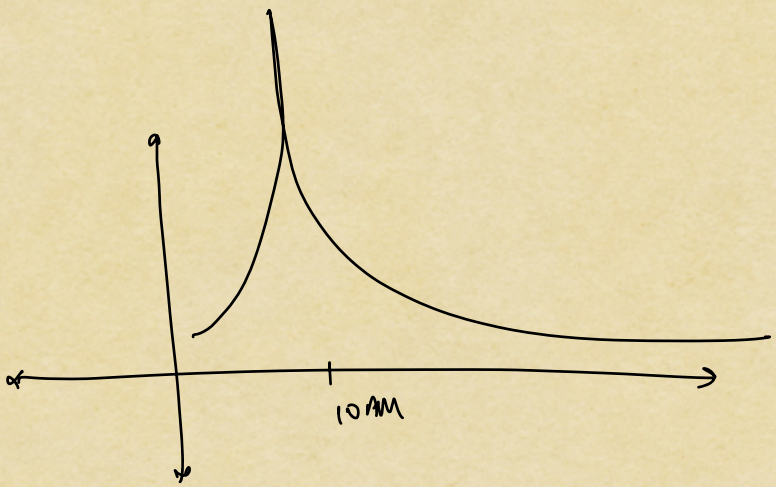
⇒ 40B views/day

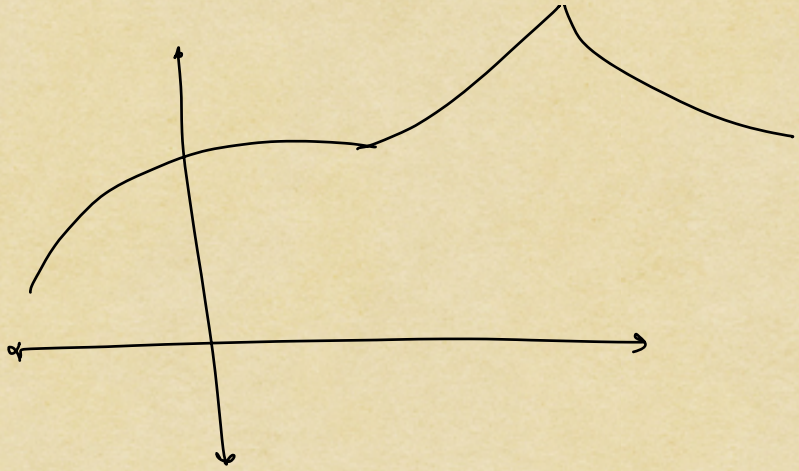
total views per sec

$$\Rightarrow \frac{40 \times 10^9}{60 \times 60 \times 24} \Rightarrow \frac{40 \times 10^9}{86400} \approx \frac{40 \times 10^9}{10^5}$$

$$\Rightarrow 4 \times 10^5 \text{ views/sec}$$





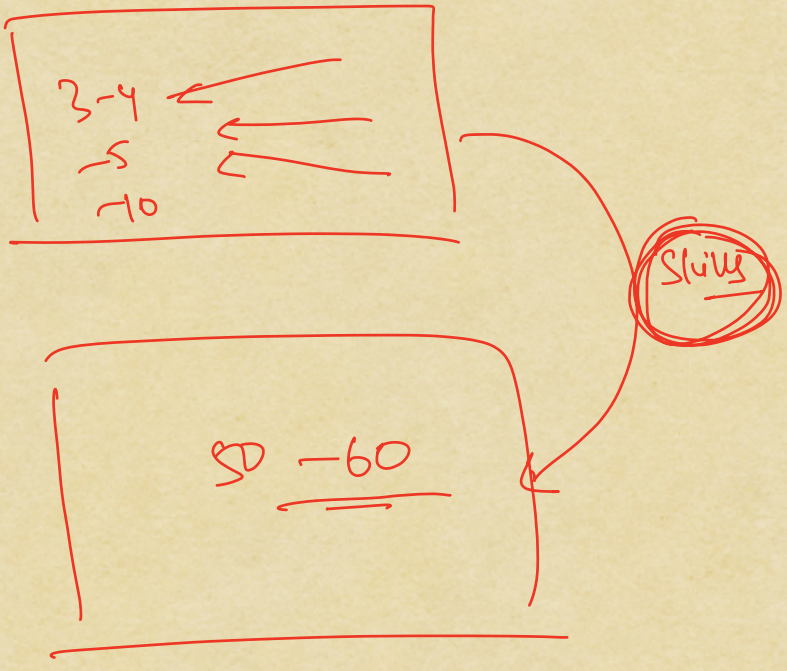


Spikes / Peaks

Normal / Gaussian distribution

Pareto distribution / Decaying distribution

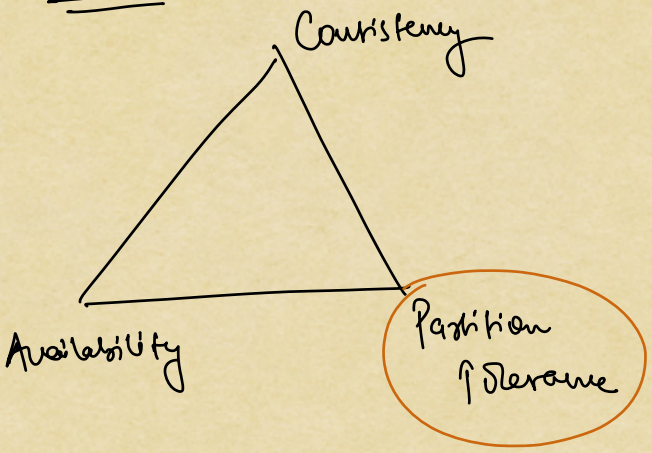
Uniform distribution



⇒ Design goals

→ CAP

⇒ CAP



You can't have all 3, can only have any 2

for video streaming

⇒ Consistency | Availability

↓ running smoothly

↓ 50 videos | 51 videos

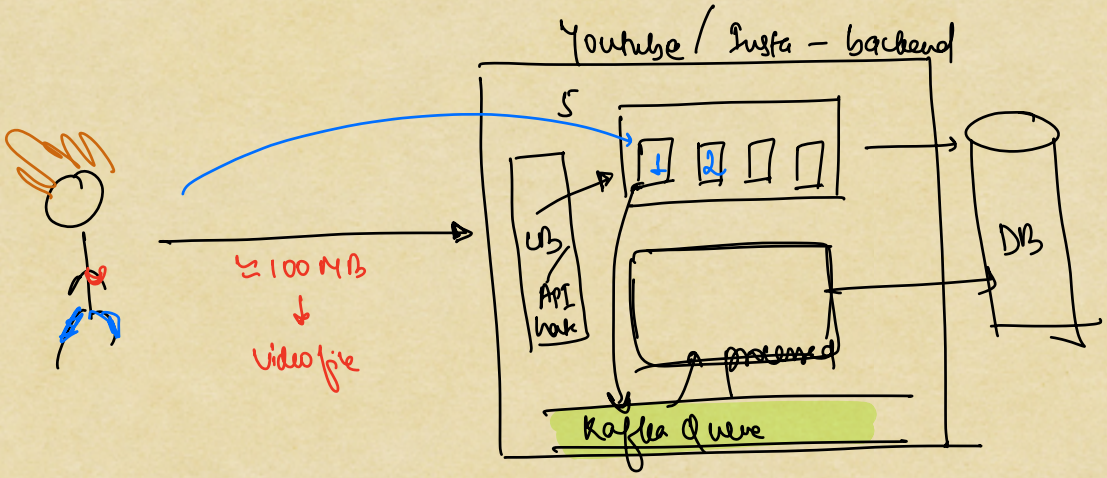
500 comments | 400 comments

banking ⇒ Partition

Availability | Consistency

⇒ System Design:

1) Content Creator will upload video



Client-side (App) will upload the video in Chunks to backend
1 MB / 10 MB

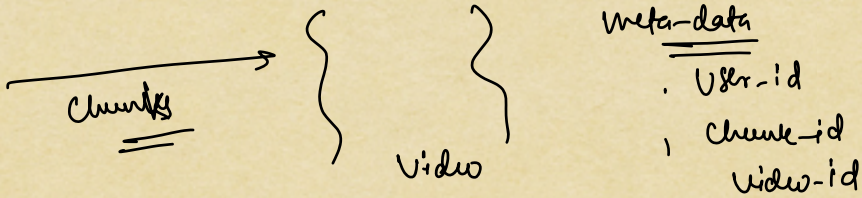
10 MB Chunks → 100 MB

Chunks => 10

& entire video (all the chunks) should be uploaded to the same server.

↓

CONSISTENT HASHING (algo)



| title
| datetime

Video + MetaData

↓ ↓

large file (100 MB) → S3

small data but structured & easily searchable → SQL

if size of each data point > 1 MB

↓

always use

Blob store | Large file Storage System | Object Store

eg : S3 | HDFS | Amazon Blob Storage | Google Cloud Storage

Video → Sequence of still images

↓

matrix of pixels → color data

R & B → [0, 255]

↓ ↓

[0, 255] [0, 255]

1 pixel

size → 3 bytes

. ← dot on screen

Resolution ⇒ pixel matrix size

⇒ width × height

1080p → 1920 × 1080

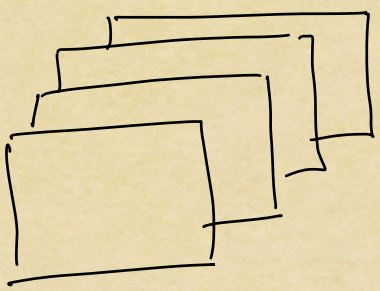
4K ⇒ 3840 × 2160

≈ 8M pixels

≈ 8M × 3 bytes

⇒ 24MB

images → frames



fps — frames / sec

movie ⇒ 24

video ⇒ 30

Half modern ⇒ 60 movies

games ⇒ 90 | 120 | 160

one 4k at 60 fps 1 min

approx size ⇒ 24 × 60 × 60

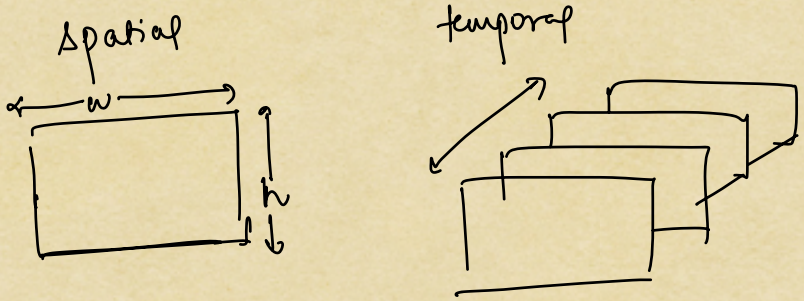
= 86400 MB

≈ 86GB

↓ compress

compressor engine ⇒ compress the video files

dimension ⇒



codecs h264 | h265 | mkv | mpeg | m4v | conv./avi | 3gp---

⇒ view the video ⇒ diff clients will have diff bandwidth

1 Gbps | 100 MBps | 100 kBps

⇒ export service | streaming service

→ Adaptive Bitrate Streaming

→ Adaptive Bitrate Streaming

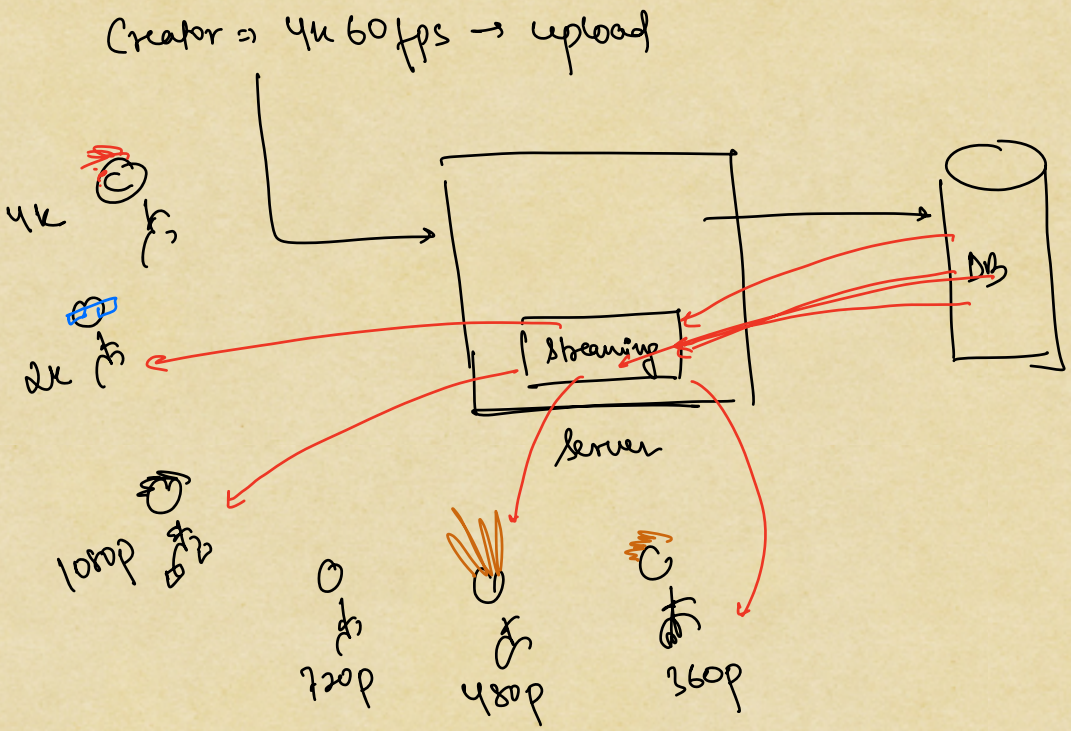
[quality]

→ Internet bandwidth

→ Screen resolution

→ Screen responsiveness (refresh-rate)

→ device configuration



* On-the-fly conversion / scaling down ⇒ never do

pro → store + only video

cons → extremely slow.

* pre-process the data → Should do

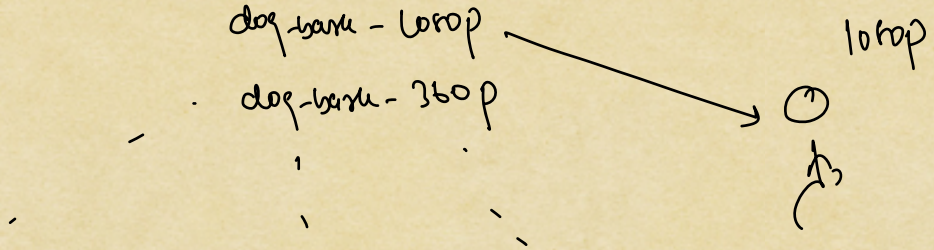
→ once the video is upload,
it will be processed and will generate
multiple copies covering all resolutions

→ dog-bark_4k

dog-bark_2k

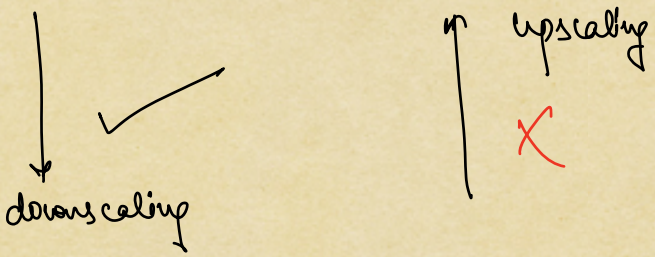
dog-bark-1080p

dog-bark-360p



pro ⇒ super efficient & fast

cons ⇒ multiple copies of same video, wastage of storage



Client Cache :

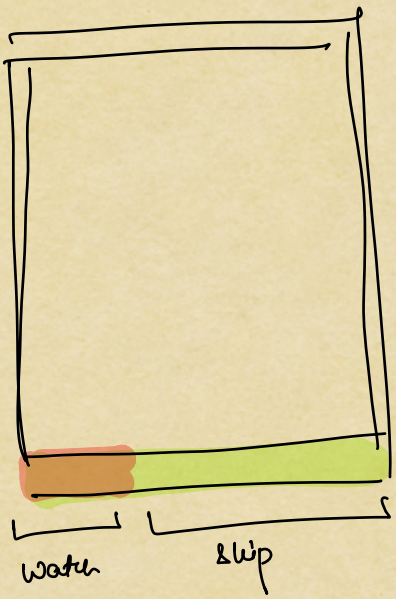
↓ download

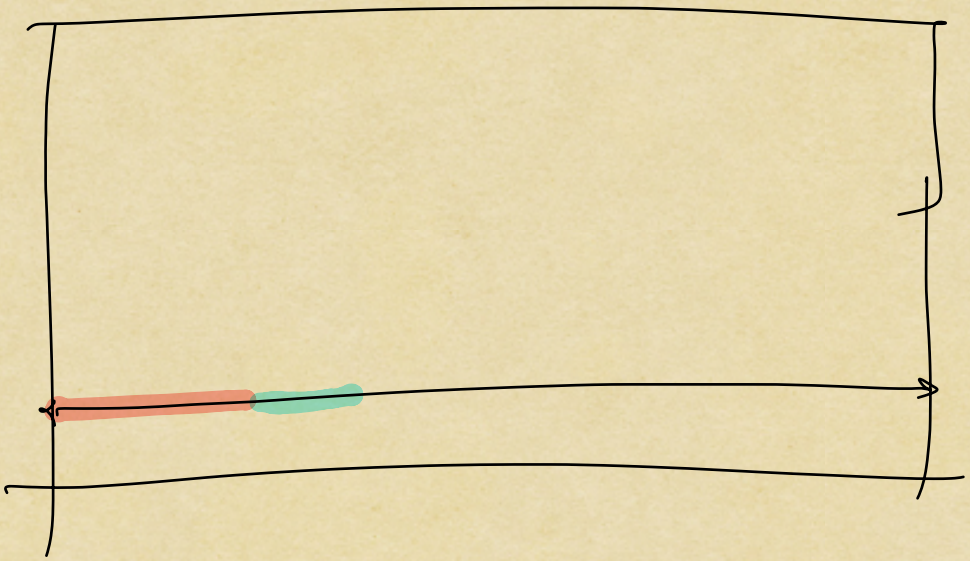
100MB

↓ download

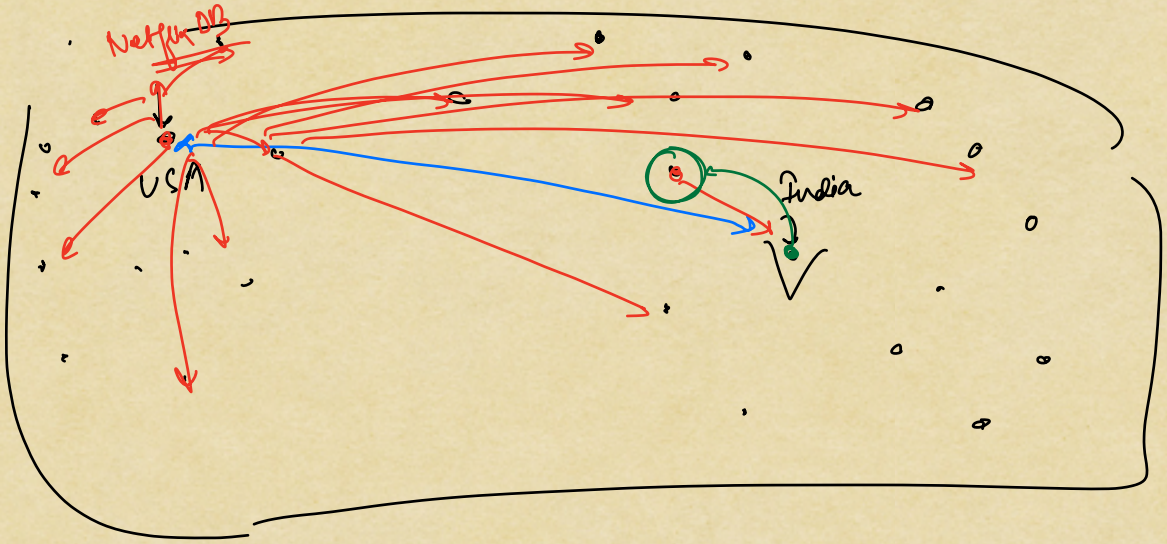
10% of 10 videos

10MB → 10 videos





⇒ CDN → Content Delivery Network



CDN

- Static | mostly Static
  images | videos | js | CSS | pdf
- act like a cache
- located at the edge
  - close to the client

India - US ⇒ $\frac{x}{y}$ ⇒ t

India - India ⇒ $\frac{x/10}{y}$ ⇒ $t/10$ faster by 10 times

CDN → Akamai
Cloudfront
Cloudflare
} 70% Internet