Design YouTube/Netflix/Hulu

Shawn K.

YouTube

(Netflix, Hulu)

Design

1. **Intro**
2. **Requirements**
   - Considerations
   - FR, NFR
3. **High Level Design**
   - Capacity Estimation
   - Components
   - Video Uploading Workflow
   - Video Streaming Workflow
4. **Detailed Design**
   - Video Transcoding
   - Optimizations
   - Error Handling
5. Wrap Up
   - Scalability: API, DB
   - Live Streaming
   - Video Takedowns

1. Intro - YouTube

- Content creators upload videos
- Viewers click play.

↓

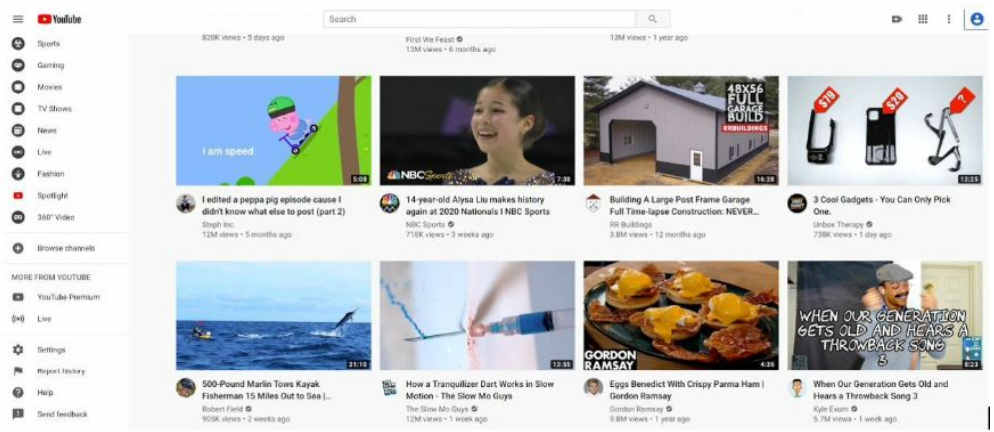
Simple?

↓

There are lots of complex technologies underneath the simplicity.

2020 Statistics:

- MAU (Monthly Active Users): 2 billions
- # of videos watched: 5 billions / day
- US adults: 73%
- Video creators: 50 millios
- Ad revenue: $15.1 billions (2019, up 36% from 2018)
- Mobile traffic: 37%
- Languages: 80 ea.



2. Requirements - Consideration

General Features

- Watch a video
- Comment
- Share
- Like a video
- Save a video to playlists
- Subscribe to a channel
- etc.

Clarification for 45-60 min

Understand Problem & Establish Scope

- What features are important? → Ability to upload and watch a video
- What clients do we need to support? → Mobile apps, web browsers, smart TV
- How many daily active users do we have? → 5 million
- What is the average daily time spent on the product? → 30 minutes
- Do we need to support international users? → Yes
- What are the supported video resolutions?
  - The system accepts most of video resolutions & formats
- Is encryption required? → Yes
- Any file size requirement for videos? → max: 1GB
- Can we leverage some of the existing cloud infrastructures provided by GCP, AWS, Azure?
  - Great question! Building everything from scratch is unrealistic for most companies, it is recommended to leverage some of the existing cloud services.

2. Requirements - FR

General Features

- Watch a video
- Comment
- Share
- Like a video
- Save a video to playlists
- Subscribe to a channel
- etc.

Clarification for 45-60 min

Functional Requirements

- Upload a video and watch a video
- Change video quality
- Clients: mobile apps, web browsers, TV

2. Requirements - NFR

Scalability

- DAU (Daily Active Users): 5 million
- Average Daily Time spent on product: 30 minutes
- International users

Performance

- **Smooth video streaming:** Users can stream videos in real-time without any lag or latency issue

High Availability

Reliability

- The system will be highly reliable

Security

- Encryption

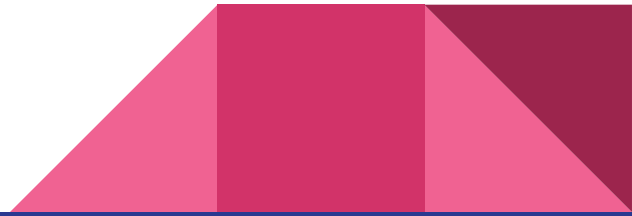
Cost

- Low infrastructure cost

3. High Level Design - Capacity Estimation (1/2)

[Assumption]

- **Network**
  - **DAU** (Daily Active Users): 5 million / day
  - **# of videos watched / day**: 5 videos per day
  - **# of videos uploaded / day**: 10% of users upload 1 video per day = 5 million (DAU) * 10% = 500K
  - **Average video size:** 300MB
- **Storage**
  - Per Day: 5 million * 10% * 300MB = 500,000 * 300 = 150,000,000 MB = 150 TB
  - **Per Month**: 150TB * 30 = 4,500 TB = 4.5 PB
  - **5 Years**: 4.5 PB * 12 months * 5 years = 270 EB
- **CDN Cost**
  - CloudFront: 100% of traffic (US), average cost per GB: $0.02
  - 5 million * 5 videos * 0.3GB * $0.02 = $150K per day



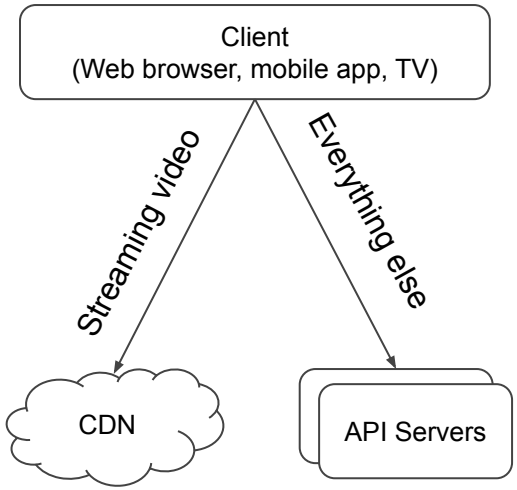
3. High Level Design - Capacity Estimation (2/2)

AWS CDN CloudFront: On-demand pricing for Data Transfer to the internet (per GB)

Per Month	United States & Canada	Europe & Israel	South Africa & Middle East	South America	Japan	Australia	Singapore, South Korea, Taiwan, Hong Kong, & Philippines	India
First 10TB	$0.085	$0.085	$0.110	$0.110	$0.114	$0.114	$0.140	$0.170
Next 40TB	$0.080	$0.080	$0.105	$0.105	$0.089	$0.098	$0.135	$0.130
Next 100TB	$0.060	$0.060	$0.090	$0.090	$0.086	$0.094	$0.120	$0.110
Next 350TB	$0.040	$0.040	$0.080	$0.080	$0.084	$0.092	$0.100	$0.100
Next 524TB	$0.030	$0.030	$0.060	$0.060	$0.080	$0.090	$0.080	$0.100
Next 4PB	$0.025	$0.025	$0.050	$0.050	$0.070	$0.085	$0.070	$0.100
Over 5PB	$0.020	$0.020	$0.040	$0.040	$0.060	$0.080	$0.060	$0.100

3. High Level Design - Components

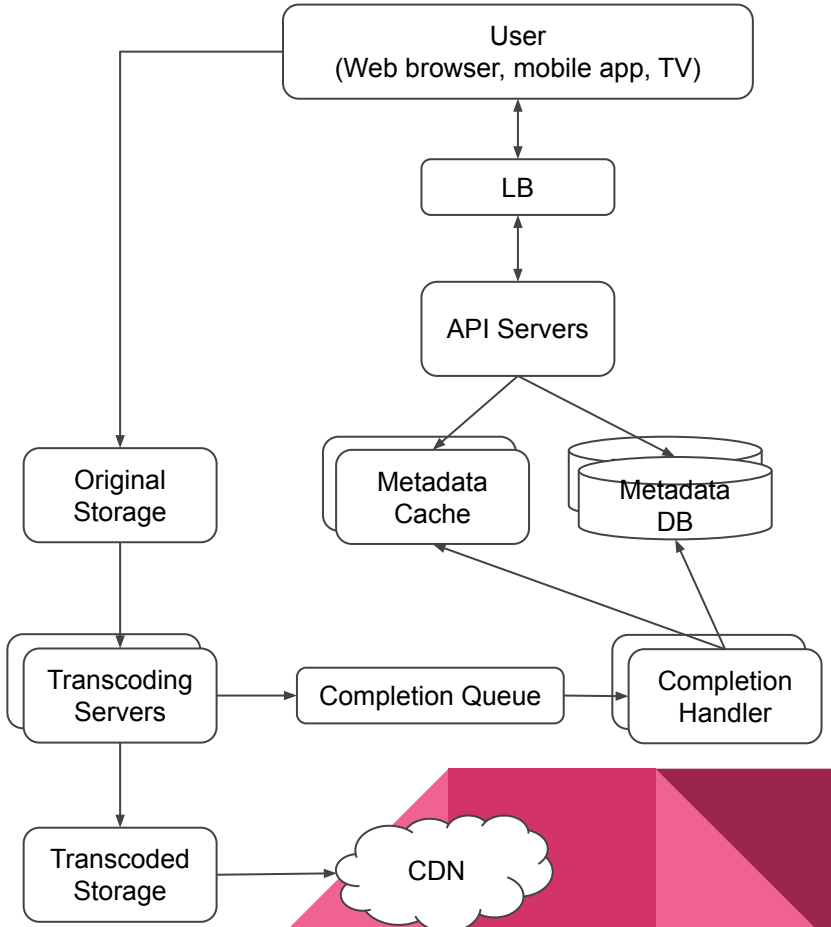
- **Client**
  - Watch YouTube on computer web browser, mobile phone, smart TV
- **CDN**
  - Videos are stored in CDN.
  - When pressing play → a video is streamed from the CDN.
- **API Servers**
  - Everything else except video streaming goes through API servers.
  - Feed recommendation
  - Generating video upload URL
  - Updating metadata database & cache
  - User signup



3. High Level Design

(Video Uploading Workflow)

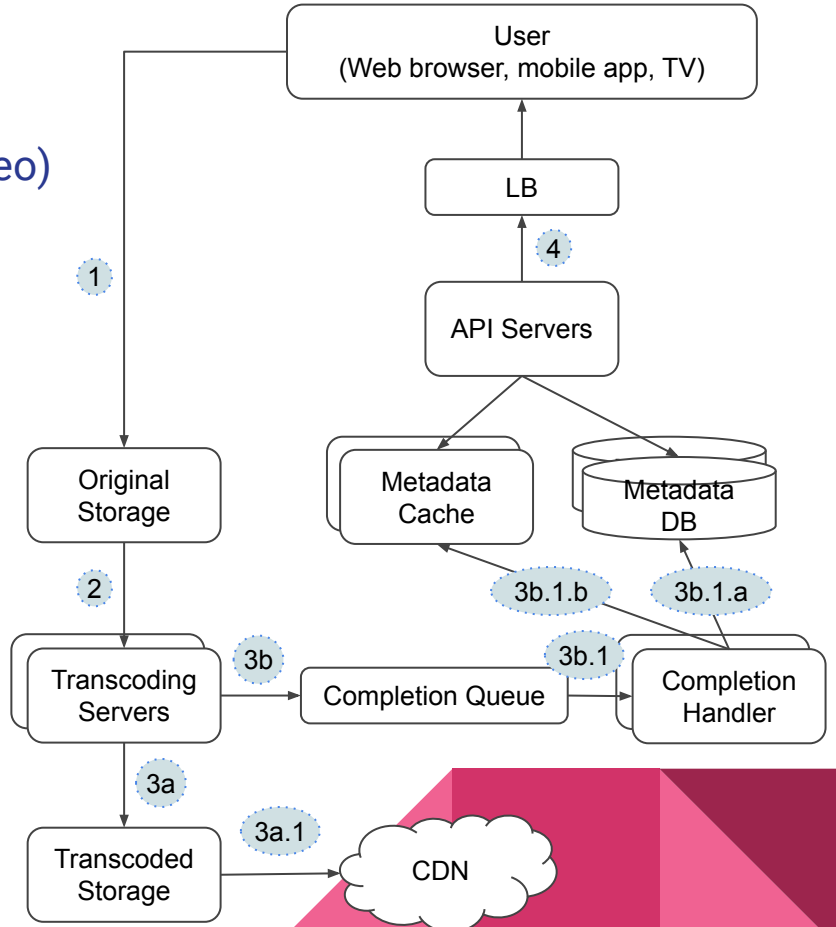
- **User, LB**
- **API Servers**: handles all reqs w/o video streaming
- **Metadata DB**
  - Stores video metadata.
  - Sharding/Replication for high performance/availability
- **Metadata Cache**: for better performance
  - Video metadata
  - User objects
- **Original Storage**: store original videos (BLOB)
- **Transcoding Servers**
  - Video encoding: convert video format
  - Provides best video streams for different devices and bandwidth
- **Transcoding Storage**: store transcoded video files (MPEG, HLS)
- **CDN**
  - Videos are cached in CDN.
  - When clicking 'play' button → a video is streamed from CDN
- **Completion Queue**
  - Message queue: stores info (transcoding completion event)
- **Completion Handler**
  - List of workers: pull event data from completion queue
  - Update metadata cache and database



3. High Level Design

(Video Uploading Workflow - upload actual video)

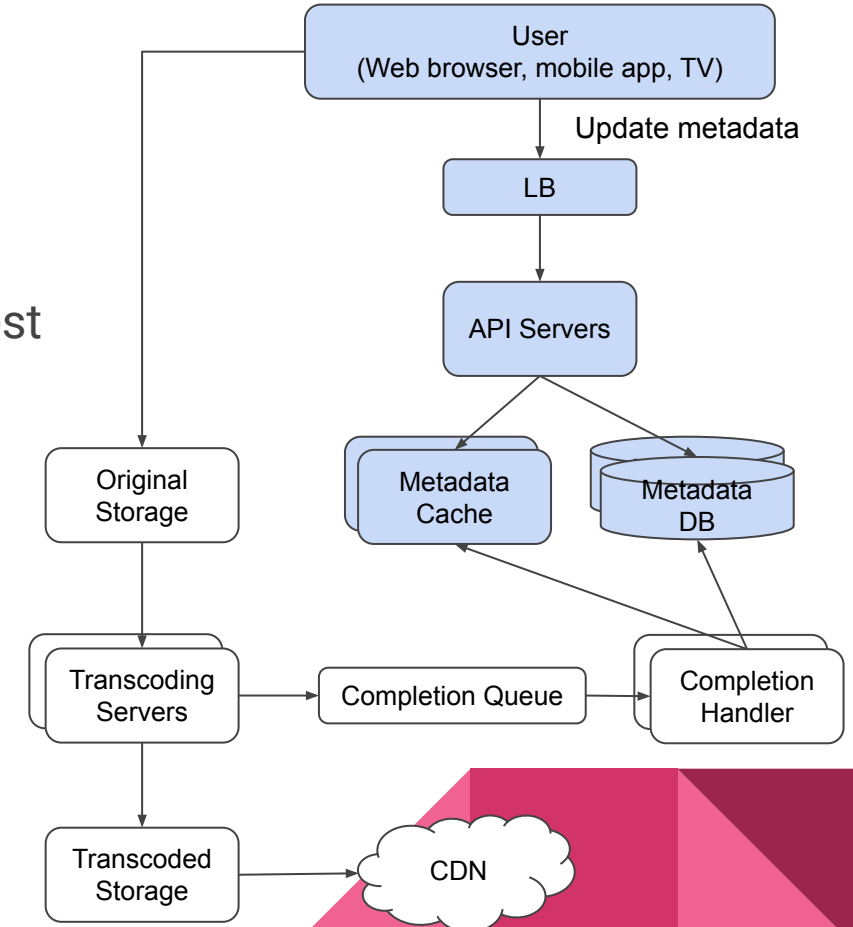
1	Videos: uploaded → original storage
2	Fetch videos from original storage & start transcoding
3	Transcoding complete → parallel processing followings
	3.a transcoded video → transcoded storage → CDN
	3.b completion event → queue
	3.b.1 Handler (workers): pull event data from queue - Update metadata (a. Database, b. cache)
4	API server → (video uploaded, ready for streaming) → client



3. High Level Design

(Video Uploading Workflow: update metadata)

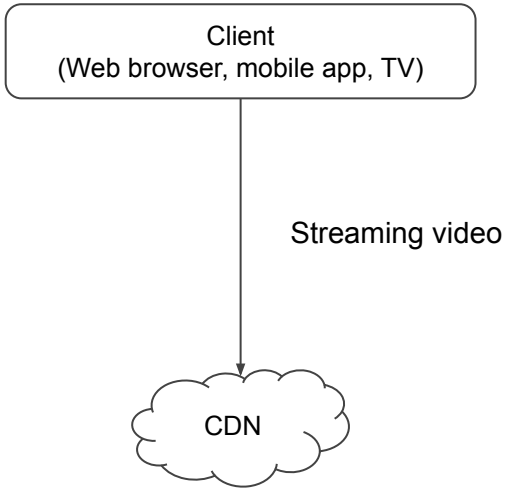
- While a file is being uploaded to the original storage, the client sends a request to update the video metadata
- Request Body (Metadata)
  - File Name
  - Size
  - Format
  - Etc.
- API Servers
  - Update the metadata cache & DB



3. High Level Design

(Video Streaming Workflow)

- Purpose
  - Whenever you watch a video on YouTube, it starts streaming immediately
  - Don’t wait until whole video is downloaded on device.
- Videos are streamed from CDN directly.
  - The edge server closest to you deliver video.
  - Very little latency
- Different streaming protocols support different video encodings and playback players.
  - MPEG-DASH
  - Apple HLS (HTTP Live Streaming)
  - Microsoft Smooth Streaming
  - Adobe HDS (HTTP Dynamic Streaming)



4. Detailed Design

- High Level Design
  - Video Uploading
  - Video Streaming

- Design Deep Dive
  - Refine both workflows w/
    - Optimizations
    - Error Handling Mechanisms

4. Detailed Design - Video Transcoding

Reasons

- **Space** Optimization: HD raw video (60 frames/sec: a few hundreds GB)
- Device/Browser **Compatibility**: they only support certain types of video format
- **High-quality** videos: deliver high/low resolution video per network bandwidth
- **Smooth UX** (for playing continuously): switch quality automatically/manually

Encoding Format

- **Container**
  - Basket for containing video file, audio, and metadata
  - Format by file extension (.AVI, .MOV, .MP4)
- **Codecs**
  - Compression/Decompression Algorithms: H.264, VP9, HEVC
  - To **reduce video size** while preserving video quality

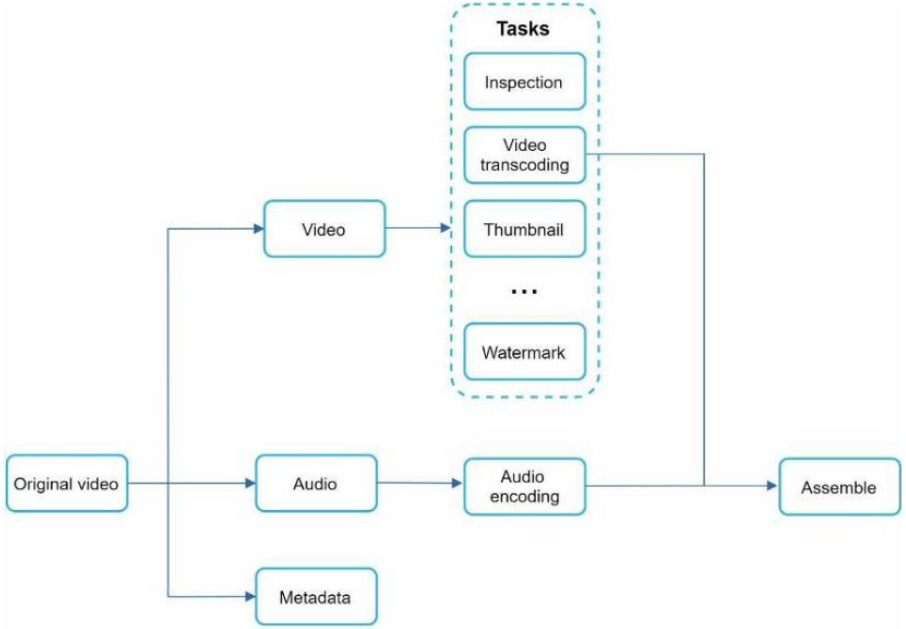
4. Detailed Design - DAG Model (Directed Acyclic Graph)

Transcoding a video

- Computationally expensive
- Time-consuming
- Different content creators: different video processing requirements
  - Watermarks on top of their videos
  - Thumbnail images themselves
  - Someone upload HD videos, whereas others do not

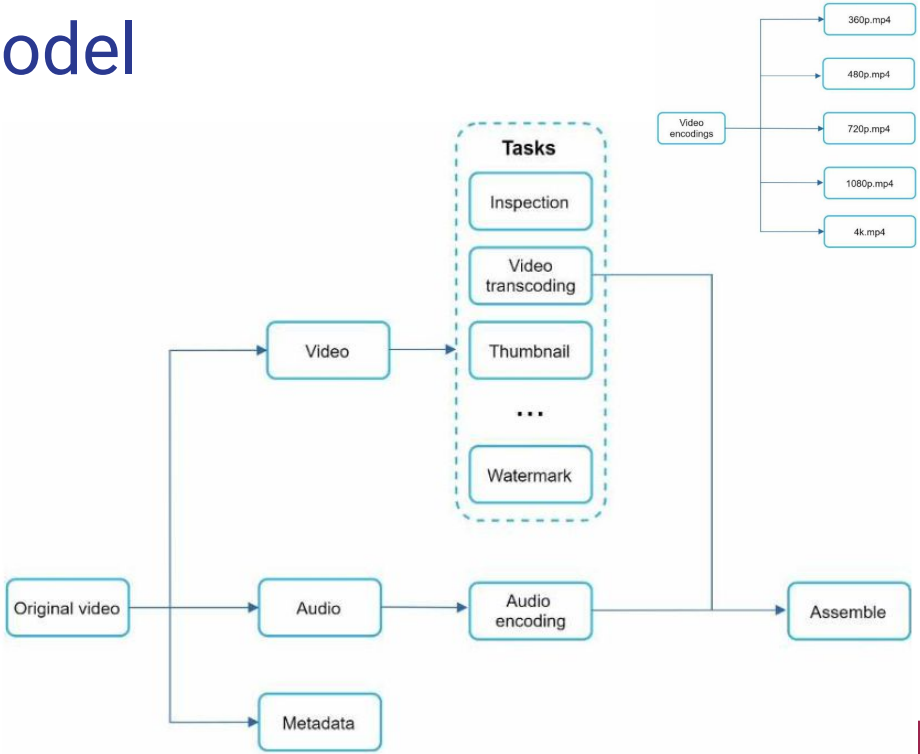
Directed acyclic graph (DAG) programming model

- Purpose
  - Support different video processing pipelines
  - Maintain high parallelism
- Add some level of abstraction
- Let client SWEs define what tasks to execute in stages
  - DAG can be executed sequentially or parallely

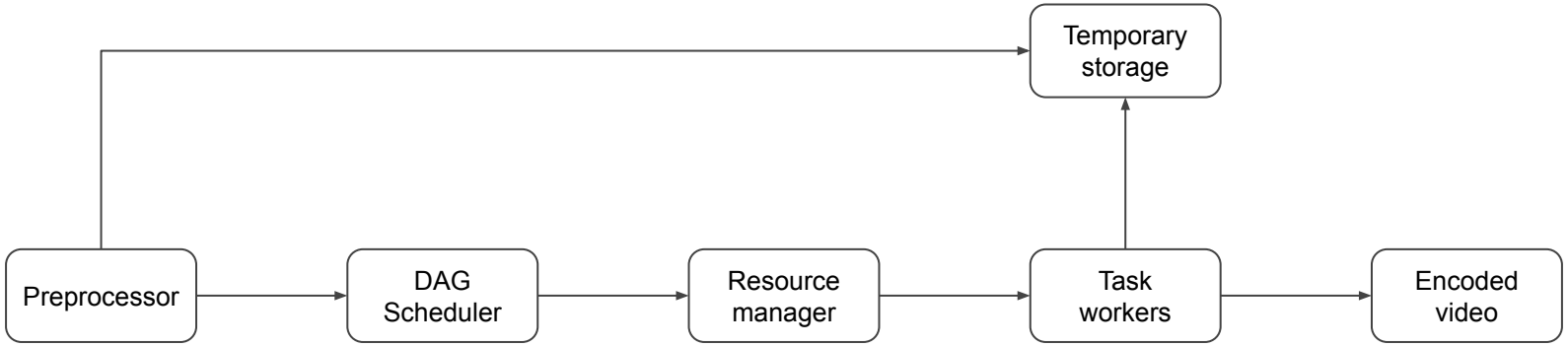


4. Detailed Design - DAG Model

- **Inspection**
  - Make sure the videos have good quality and aren't malformed
- **Video encodings**
  - Converted to support different resolutions, codec, bitrates, etc.
- **Thumbnail**
  - Uploaded by user
  - Automatically generated by system
- **Watermark**
  - Image overlay on top of video (contains identifying info about your video)

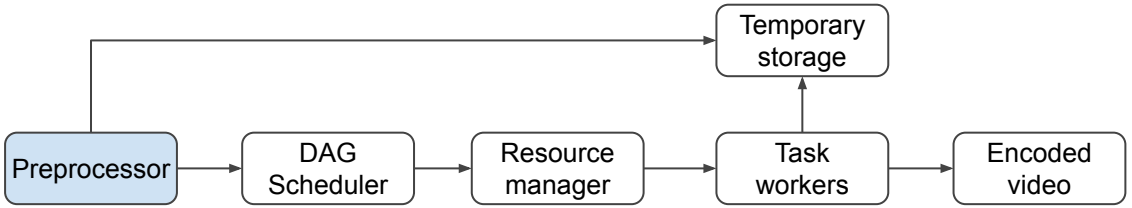


4. Detailed Design - Video Transcoding Architecture



4. Detailed Design

(Video Transcoding Architecture)



- **Video Splitting**
  - Groups of Pictures(GOP) Alignment: group/chunk of frames in a specific order
  - Chunk: independently playable (a few secs)
  - Not supported browser/old-devices: preprocessor split videos by GOP alignment
- **DAG Generation**
  - Based on configuration files
- **Cache Data**
  - Cache for segmented videos
  - For better reliability: store GOPs and metadata in temporary storage.
    - If video encoding fails → use persisted data for retry operations

Figure 14-12

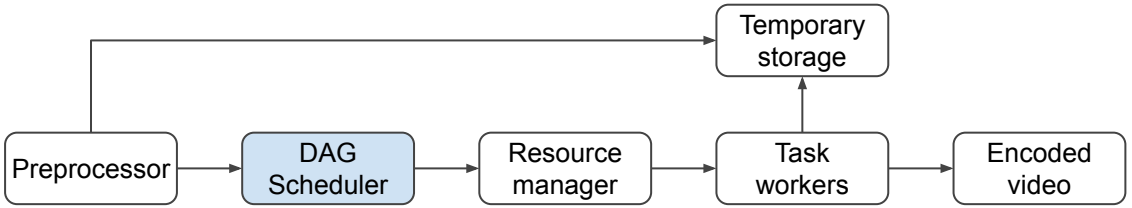
This DAG representation is generated from the two configuration files below (Figure 14-13):

```
task {
    name 'download-input'
    type 'Download'
    input {
        url config.url
    }
    output { it->
        context.inputVideo = it.file
    }
    next 'transcode'
}
```

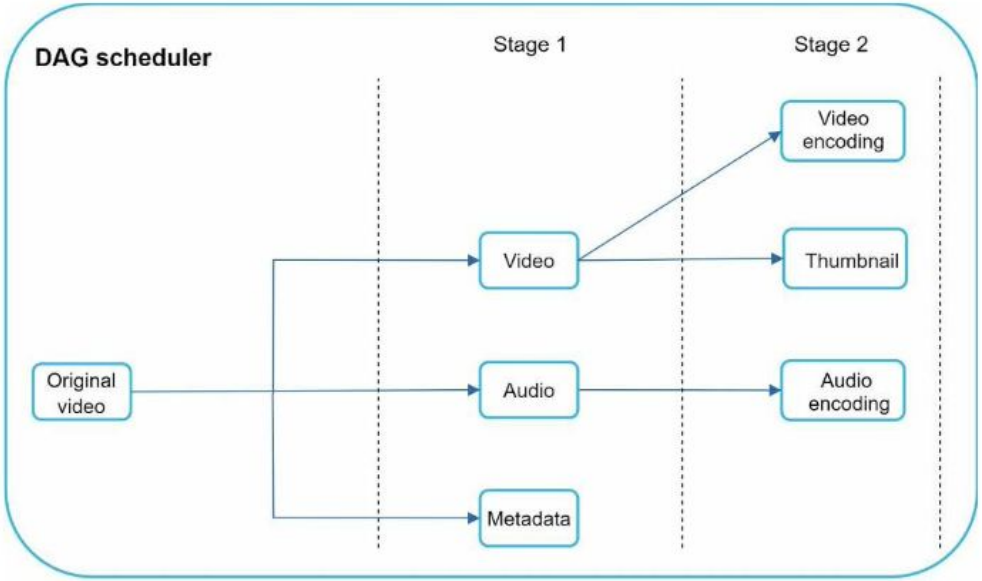
```
task {
    name 'transcode'
    type 'Transcode'
    input {
        input context.inputVideo
        config config.transConfig
    }
    output { it->
        context.file = it.outputVideo
    }
}
```

4. Detailed Design

(Video Transcoding Architecture)

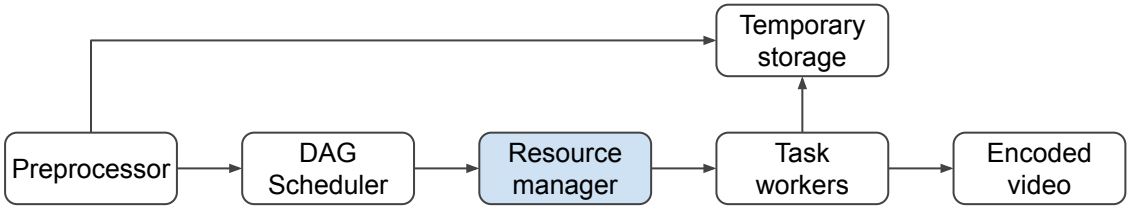


- Split a DAG into tasks' stages
  - Original video → video, audio, metadata
  - Video file → 2 tasks (video encoding, thumbnail)
  - Audio file → audio encoding
- Put them into task queue in resource manager

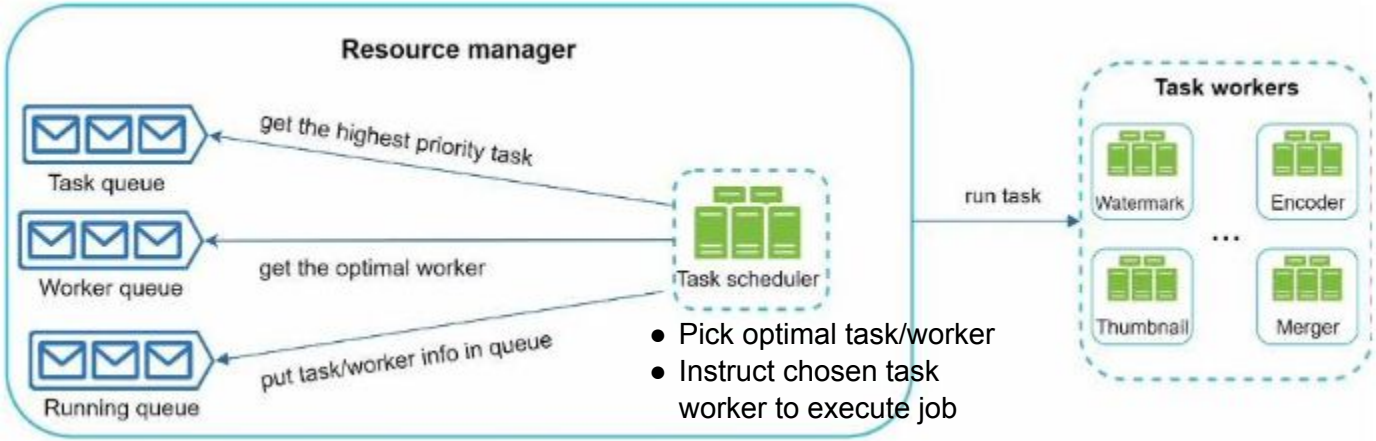


4. Detailed Design

(Video Transcoding Architecture)



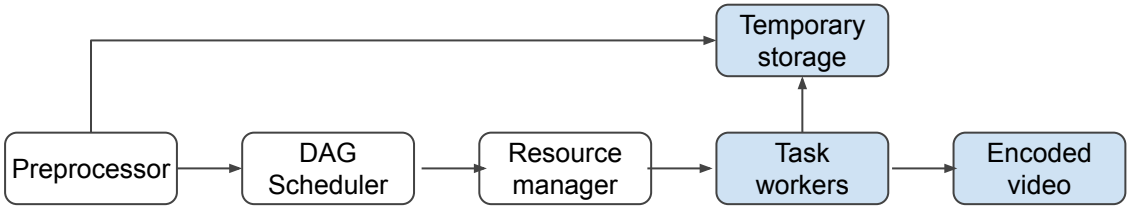
Managing resource allocation efficiency



- Pick optimal task/worker
- Instruct chosen task worker to execute job

4. Detailed Design

(Video Transcoding Architecture)



Task Workers

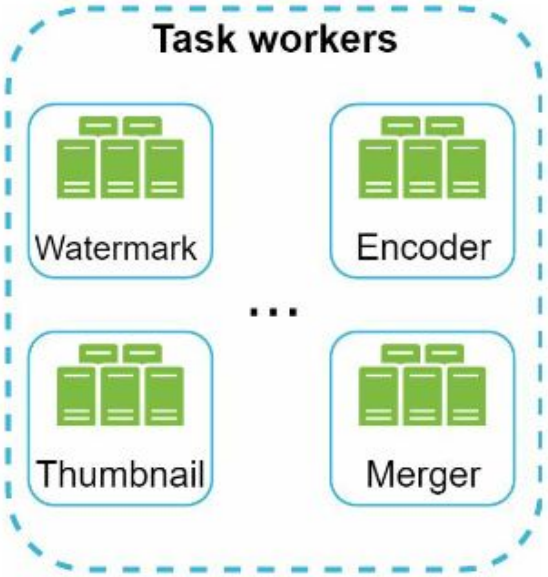
- Run tasks (defined in DAG)
- Different task workers may run different tasks

Temporary Storage

- System **Choice**: depends on factors (type, size, access freq., life span)
- **Metadata**: Frequency: Highly accessed by workers; Size: small → In memory caching: good idea
- **Video/Audio Data**: Blob storage; Data is freed up once corresponding video processing is complete

Encoded Video

- Final output of encoding pipeline: funny_720p.mp4



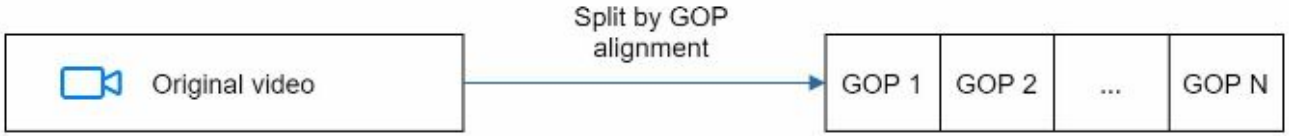
4. Detailed Design - Optimization

Speed Optimization
- **Parallelize video uploading**
- Place upload centers close to users
- Parallelism everywhere

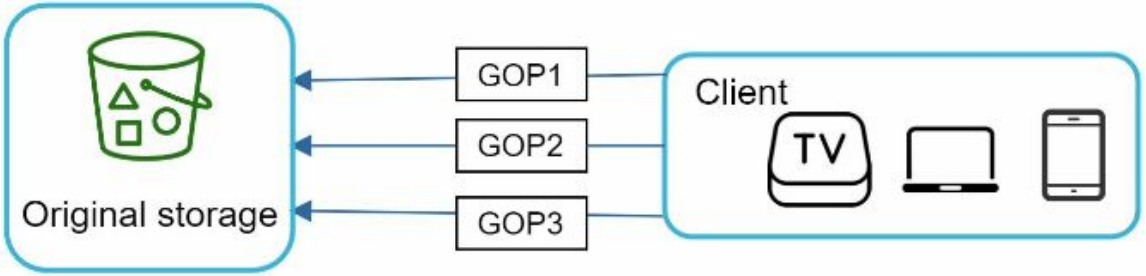
Safety Optimization
- Pre-signed upload URL
- Protect your videos

Cost-Saving Optimization

Upload a video (whole unit) → Inefficient → Split a video → smaller chunks



Previous upload failure → fast resumable uploads (Client to implement to improve upload speed)



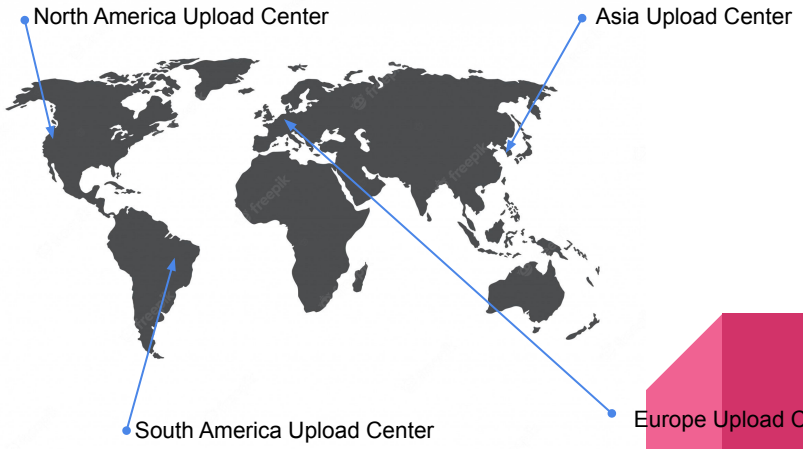
4. Detailed Design - Optimization

Setting up multiple upload centers across the globe.

Speed Optimization
- Parallelize video uploading
- **Place upload centers close to users**
- Parallelism everywhere

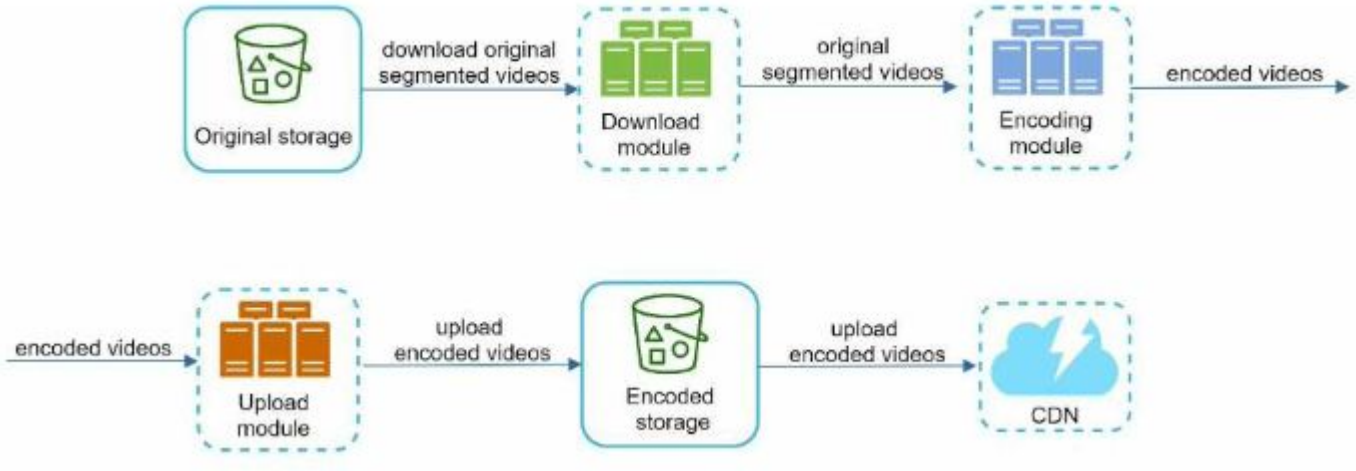
Safety Optimization
- Pre-signed upload URL
- Protect your videos

Cost-Saving Optimization



4. Detailed Design - Optimization

As-Is: Dependency: original storage → CDN: parallelism difficult



Speed Optimization
- Parallelize video uploading
- Place upload centers close to users
- **Parallelism everywhere**

Safety Optimization
- Pre-signed upload URL
- Protect your videos

Cost-Saving Optimization

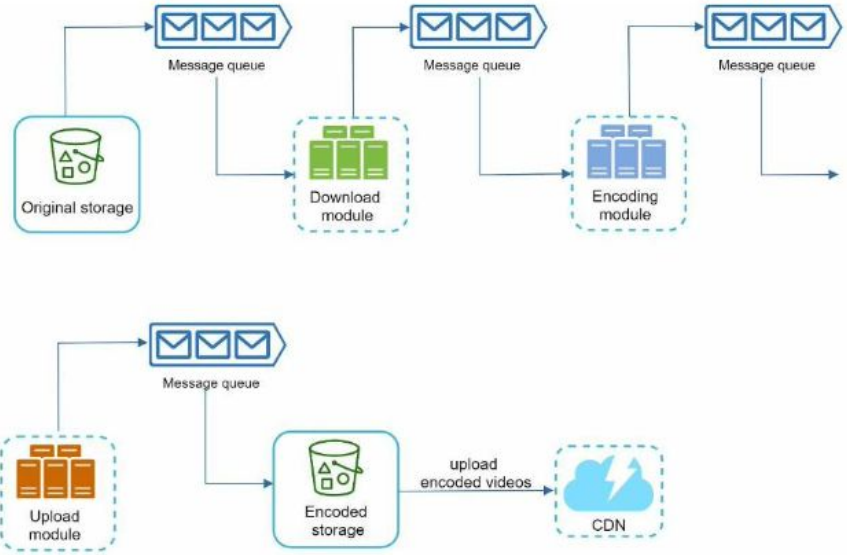
4. Detailed Design - Optimization

- Encoding module: wait before message queue is introduced
- After msg queue is introduced
  - Encoding module: no wait for output of Download module
  - Many events in the queue → Encoding module execute jobs in parallel

Speed Optimization
- Parallelize video uploading
- Place upload centers close to users
- **Parallelism everywhere**

Safety Optimization
- Pre-signed upload URL
- Protect your videos

Cost-Saving Optimization



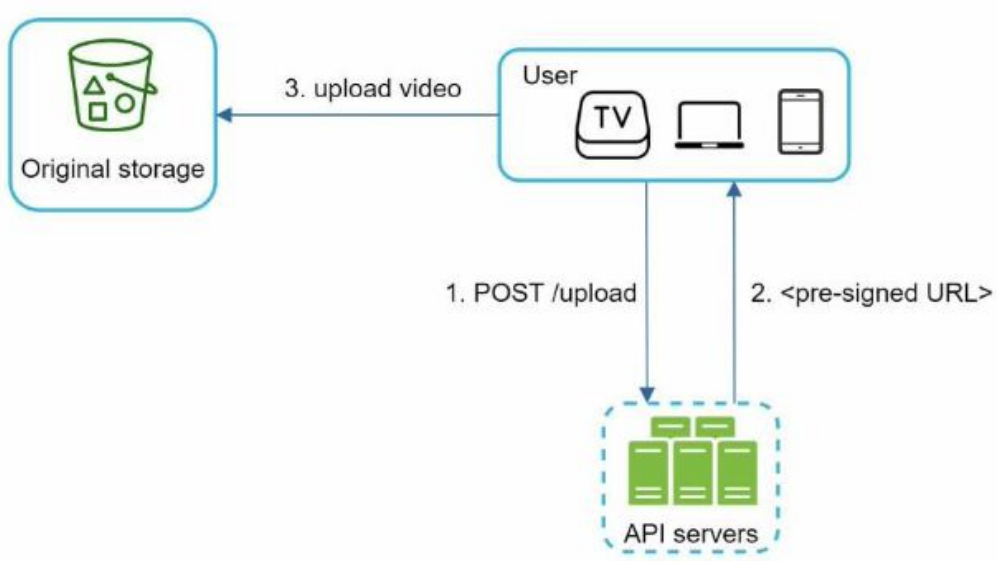
4. Detailed Design - Optimization

Ensure only authorized users upload videos → right location.

Speed Optimization
- Parallelize video uploading
- Place upload centers close to users
- Parallelism everywhere

Safety Optimization
- **Pre-signed upload URL**
- Protect your videos

Cost-Saving Optimization



- Amazon S3
- Azure blob storage (via Shared Access Signature)

4. Detailed Design - Optimization

Speed Optimization
- Parallelize video uploading
- Place upload centers close to users
- Parallelism everywhere

Safety Optimization
- Pre-signed upload URL
- **Protect your videos**

Cost-Saving Optimization

Content Makers

- Reluctant to upload to post videos online
- Reason: fear original videos → stolen

To **Protect** Copyrighted Videos:

- Digital rights management (**DRM**) system: Apple FairPlay, Google Widevine, Microsoft PlayReady
- **AES** encryption
  - Encrypt a video, configure an authZ policy
  - Decrypt upon playback; Authorized users can only watch an encrypted video
- Visual **watermarking**:
  - Image on top of video (w/ identifying info for your video)
  - E.g., Company logo or name

4. Detailed Design - Optimization

Speed Optimization
- Parallelize video uploading
- Place upload centers close to users
- Parallelism everywhere

Safety Optimization
- Pre-signed upload URL
- Protect your videos

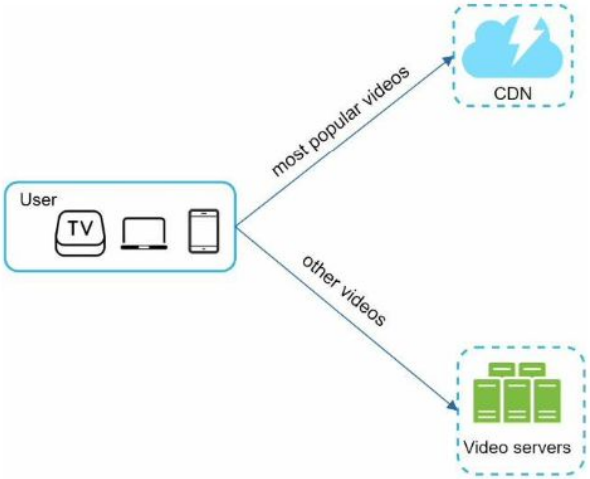
Cost-Saving Optimization

CDN

- Pros: Crucial component for fast video deliver on global scale
- Cons: Expensive (size large?) → How to reduce cost?

Optimization based on content popularity, user access pattern, video size

- **CDN**: Most **popular videos**
  - Some videos: only popular in certain regions; no need to distribute them to other regions
  - Build your own CDN and partner w/ ISPs → improve viewing experience & reduce bandwidth charges
- **Video Servers**: other **less popular videos**
  - May not need to store many encoded video versions



4. Detailed Design - Error Handling Mechanism

Large-scale System Errors: Unavoidable

- Build highly fault-tolerant system
- Handle errors gracefully
- Recover fast

Error Types

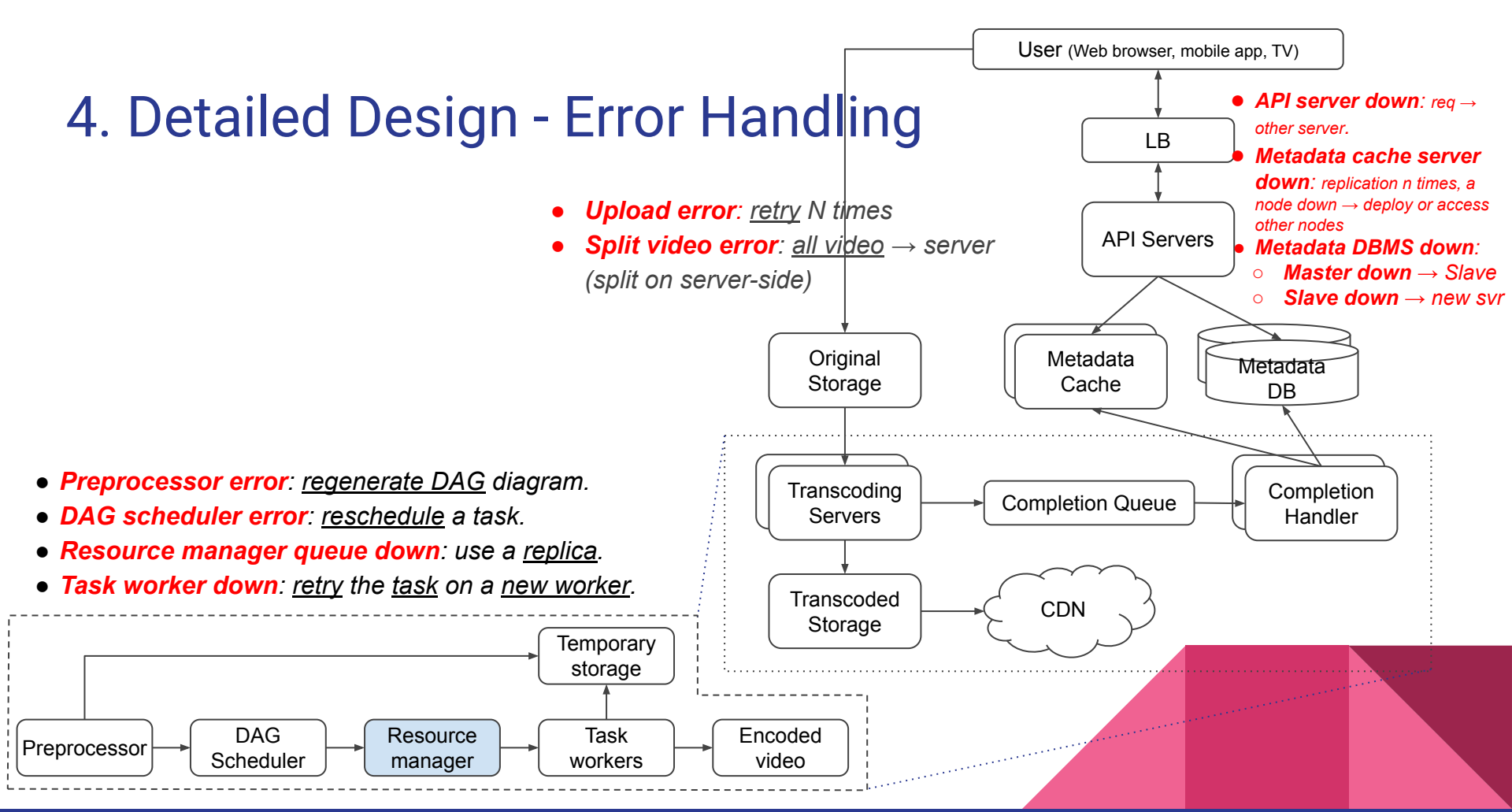
- **Recoverable Error**: video segment fails to transcode → retry n times → fail → error code to client
- **Non-recoverable Error**: malformed video format → stop running tasks → error code to client

4. Detailed Design - Error Handling

- ***Upload error***: *retry N times*
- ***Split video error***: *all video → server (split on server-side)*

- ***API server down***: *req → other server.*
- ***Metadata cache server down***: *replication n times, a node down → deploy or access other nodes*
- ***Metadata DBMS down***:
  - ***Master down*** *→ Slave*
  - ***Slave down*** *→ new svr*

- ***Preprocessor error***: *regenerate DAG diagram.*
- ***DAG scheduler error***: *reschedule a task.*
- ***Resource manager queue down***: *use a replica.*
- ***Task worker down***: *retry the task on a new worker.*



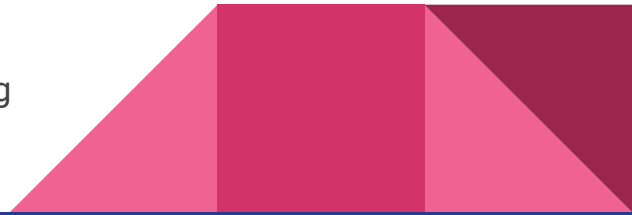
5. Wrap Up

What We Did

- Architecture design for video streaming services (like YouTube)

Additional Points

- **Scale API**: API servers (stateless) → easy to scale horizontally
- **Scale DB**: replication, sharding
- **Live Streaming**:
  - Higher latency → need different streaming protocol
  - Lower requirement for parallelism → small chunks (already processed in real-time)
  - Different sets of error handling: too much time (not acceptable)
- **Video Takedowns**:
  - Copyrights, pornography, other illegal acts: shall be removed
  - Some can be discovered during upload process or via user flagging



References

- Alex Xu, System Design Interview (Insider’s Guide)
- Scaling Patterns for Netflix's Edge