Edit Timelines & Efficient Streaming of Media

Mangala Prabhu and Eric Reinecke

NETFLIX

Agenda

- Part I: Trailers at Netflix
- Part II: Edit Intelligence In Pipelines, OpenTimeLineIO

NETFLIX

Trailers at Netflix

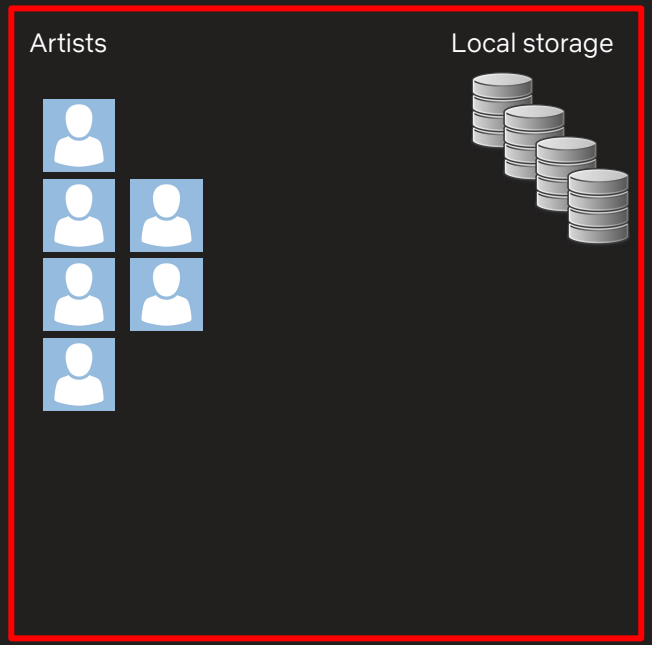
Mangala Prabhu, Compute and Storage Infrastructure @ Netflix

NETFLIX

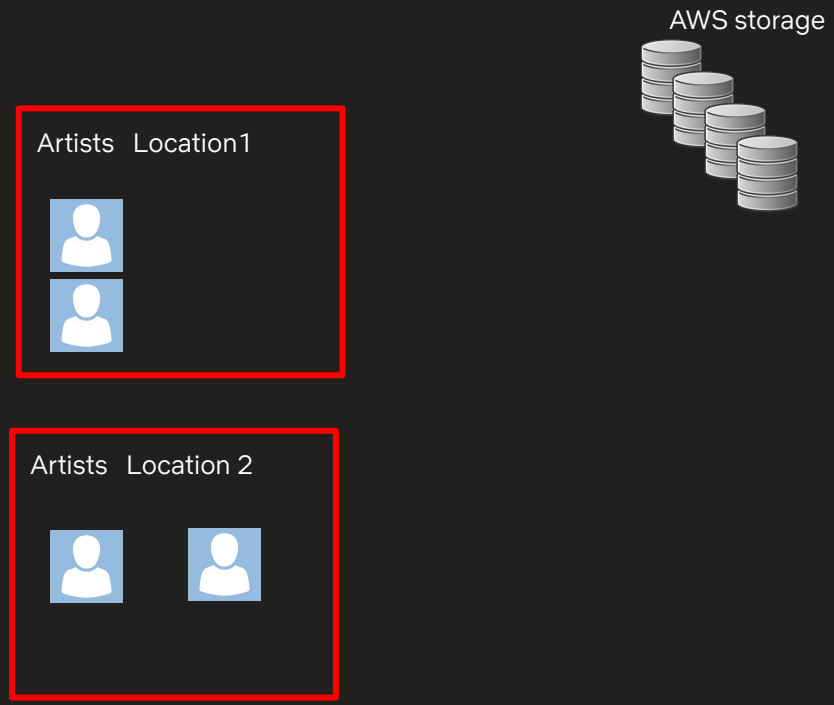
What do we in CSI?

- Managing cloud infrastructure for media processing
- Cloud compute efficiency
- Secure cloud storage of media
- Media transport layer

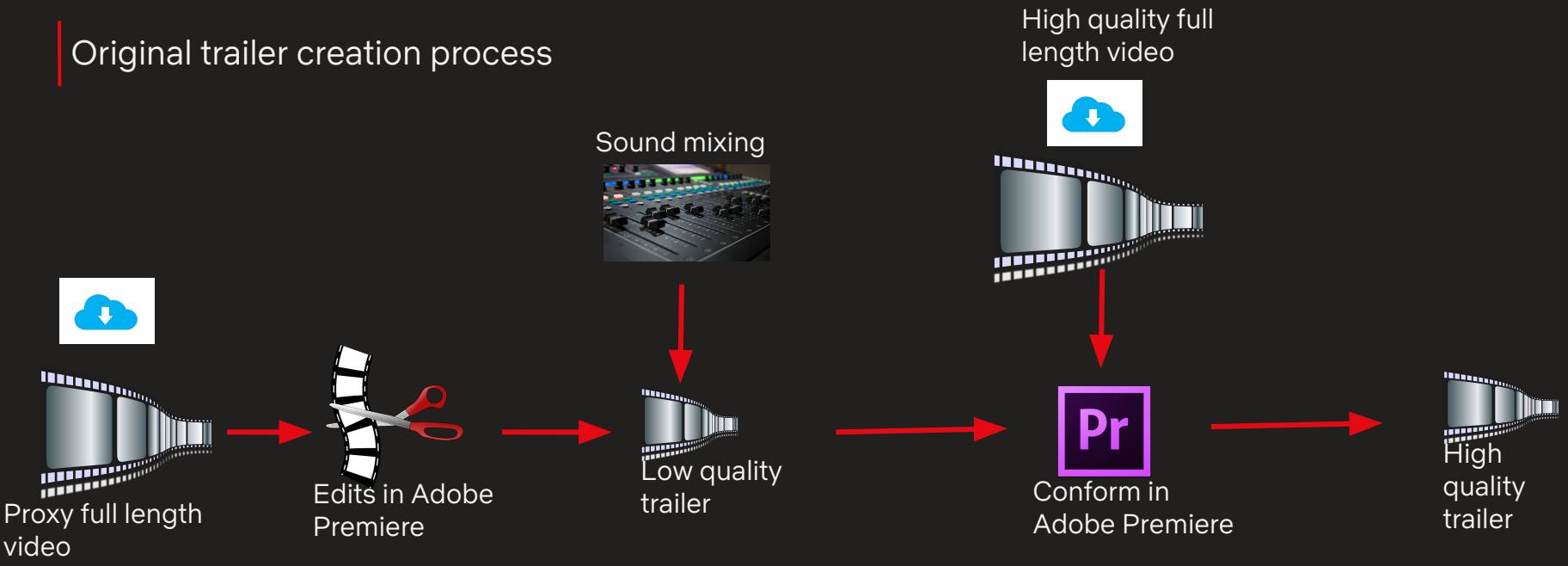
Traditional studios - storage on premises



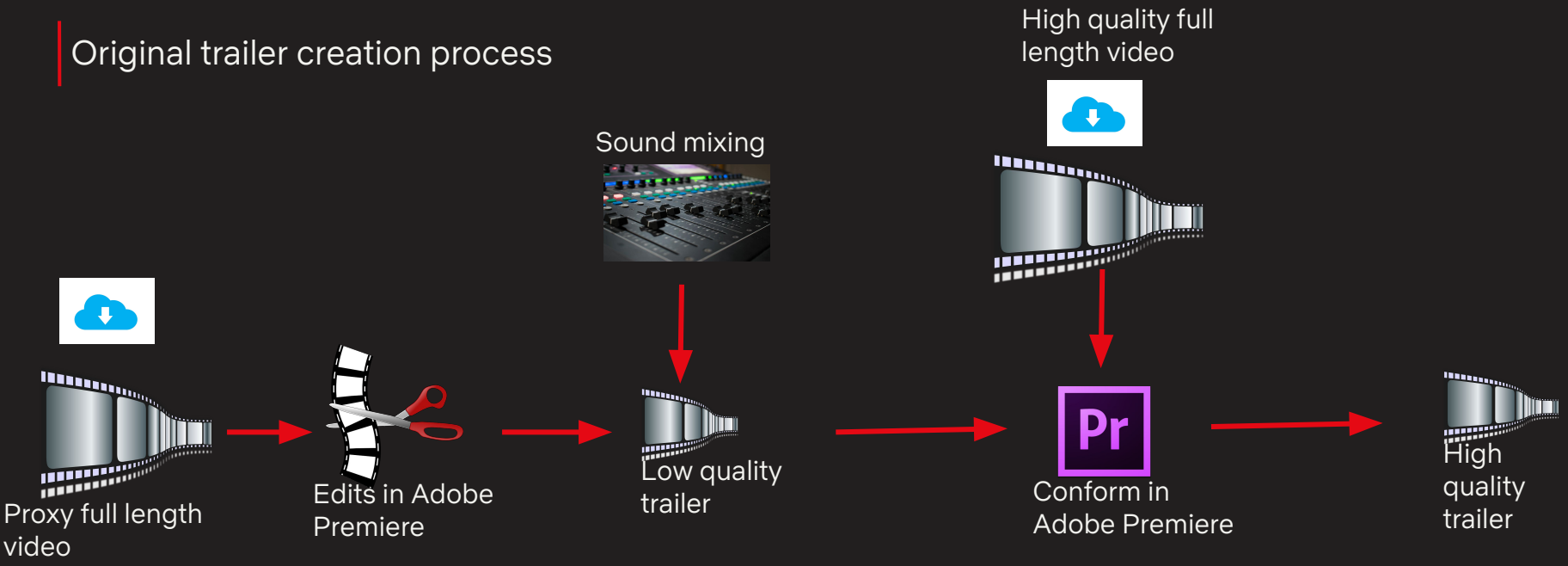
Netflix studios - storage in the cloud



Original trailer creation process

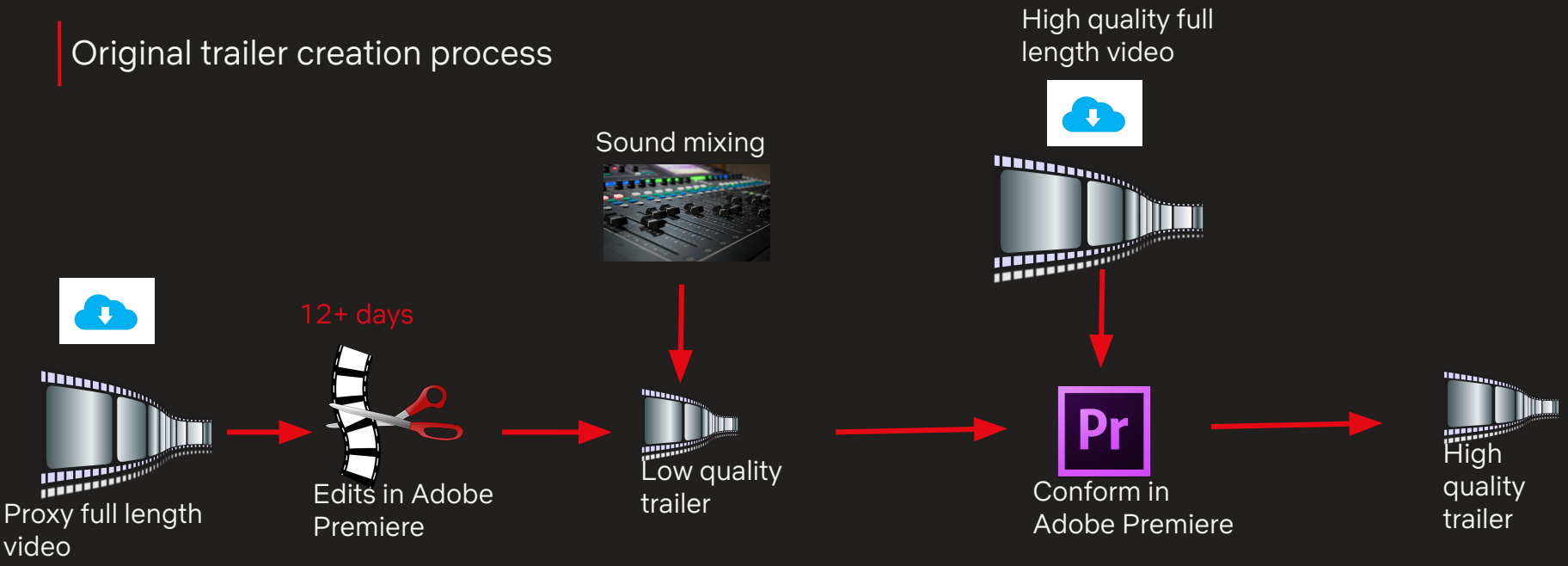


Original trailer creation process



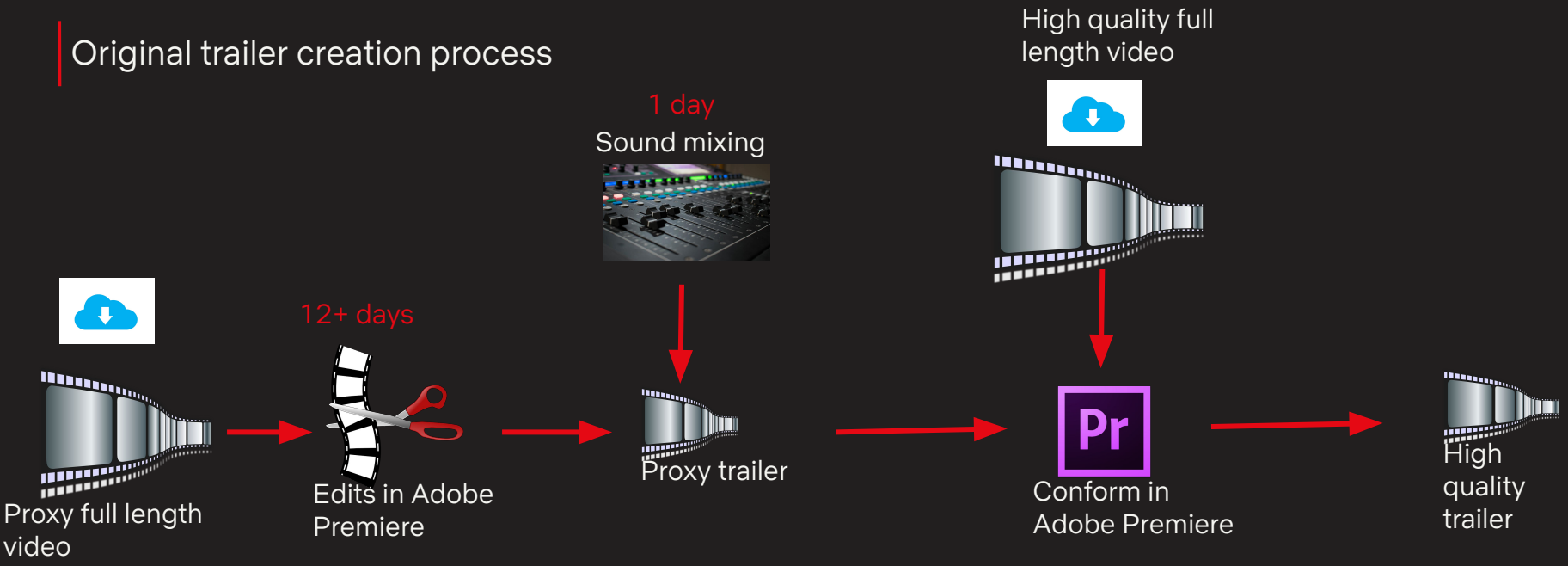
- 300 - 500 MB
- 1-13 episodes / Film
- 2 mins - 5 mins download time

Original trailer creation process



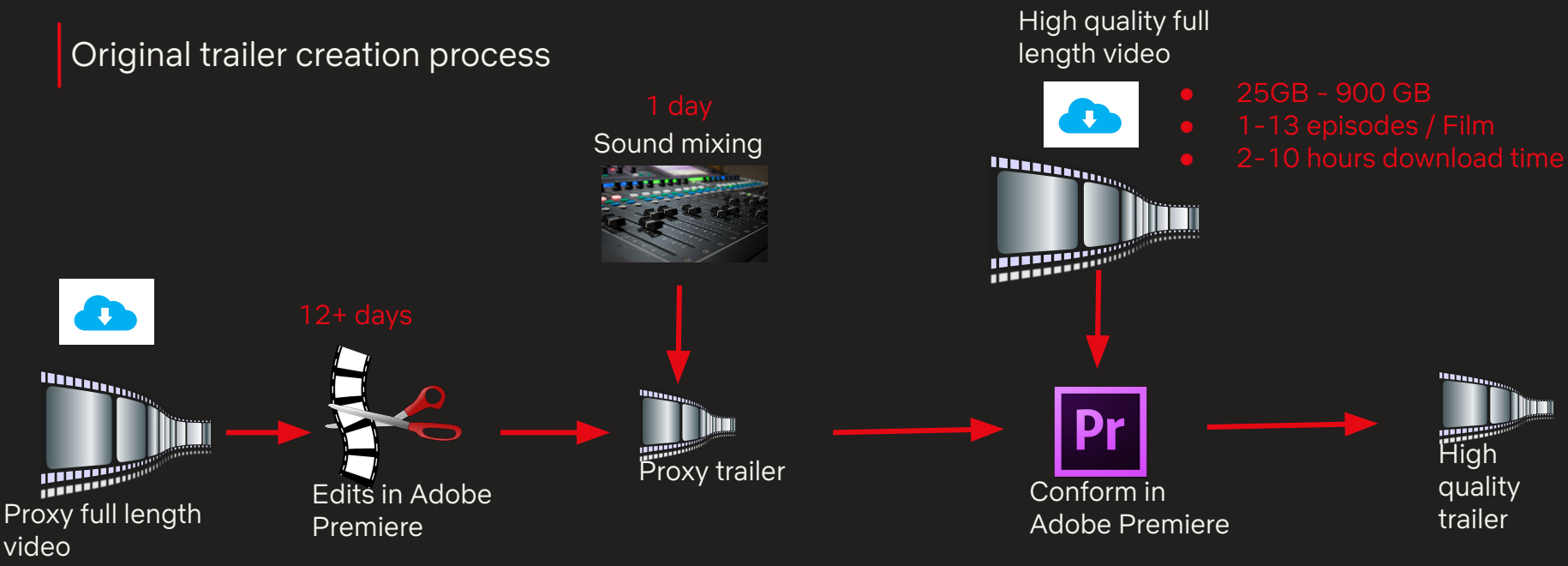
- 300 - 500 MB
- 1-13 episodes / Film
- 2 mins - 5 mins download time

Original trailer creation process



- 300 - 500 MB
- 1-13 episodes / Film
- 2 mins - 5 mins download time

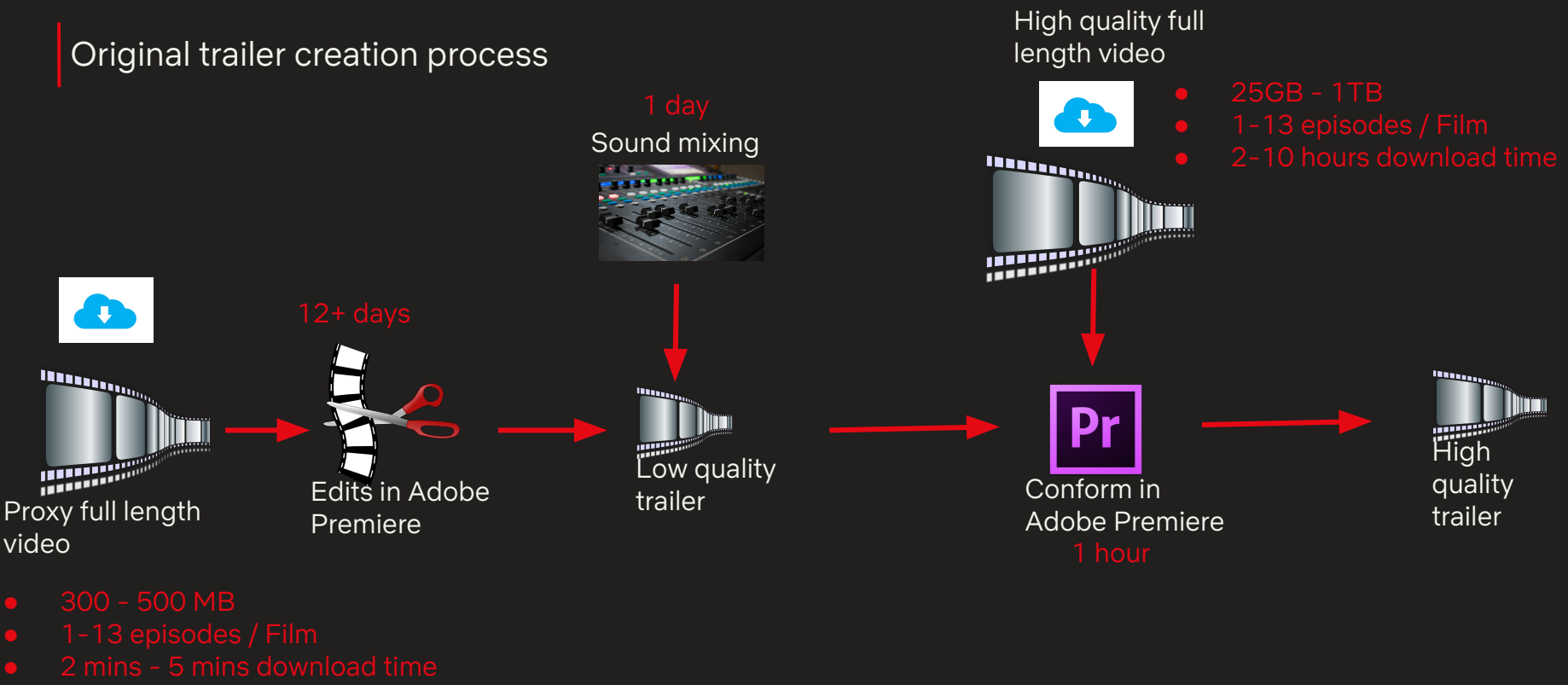
Original trailer creation process



- 25GB - 900 GB
- 1-13 episodes / Film
- 2-10 hours download time

- 300 - 500 MB
- 1-13 episodes / Film
- 2 mins - 5 mins download time

Original trailer creation process



Editor's pain point

High quality full length video

- 25GB - 1TB
- 1-13 episodes / Film
- 2-10 hours download time

1 day

Sound mixing

12+ days

Proxy full length video

Edits in Adobe Premiere

Low quality trailer

Conform in Adobe Premiere

1 hour

High quality trailer

- 300 - 500 MB
- 1-13 episodes / Film
- 2 mins - 5 mins download time

Editor's pain point

High quality full length video

- 25GB - 1TB
- 1-13 episodes / Film
- 2-10 hours download time

1 day

Sound mixing

12+ days

Proxy full length video

Edits in Adobe Premiere

Low quality trailer

Conform in Adobe Premiere

1 hour

High quality trailer

- 300 - 500 MB
- 1-13 episodes / Film
- 2 mins - 5 mins download time

How do we do it?

- Which parts of the movie do we really want?
- How to make this trailer length high quality video appear as a full length video?

How do we get the artist’s creative decisions?

- The artist decides on what goes into the trailer
- Adobe Premiere can export this decision into human readable format - an EDL file

EDL (Edit decision list)

- EDLs have the timecodes from the proxy source that made it in the trailer and where it is placed in the trailer.

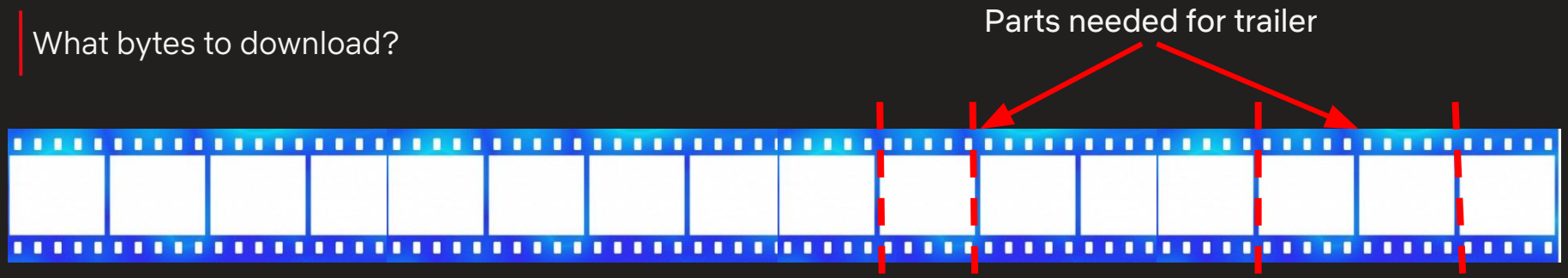
Input Time codes **Output Time codes**

```
003  AX       V     C        00:14:41:01 00:14:49:04 01:00:03:19 01:00:11:22
* FROM CLIP NAME: Norm Macdonald Has a Show_S01E03_Judge Judy_1445392.mp4

004  AX       V     C        00:19:33:01 00:19:41:03 01:00:11:22 01:00:19:24
* FROM CLIP NAME: Norm Macdonald Has a Show_S01E01_Drew Barrymore_1445390.mp4

005  AX       V     C        00:15:36:27 00:15:40:17 01:00:19:24 01:00:23:14
* FROM CLIP NAME: Norm Macdonald Has a Show_S01E03_Judge Judy_1445392.mp4

006  AX       V     C        00:26:25:21 00:26:26:13 01:00:23:14 01:00:24:06
* FROM CLIP NAME: Norm Macdonald Has a Show_S01E01_Drew Barrymore_1445390.mp4

007  AX       V     C        00:27:24:15 00:27:26:16 01:00:24:06 01:00:26:07
* FROM CLIP NAME: Norm Macdonald Has a Show_S01E01_Drew Barrymore_1445390.mp4

008  AX       V     C        00:21:55:26 00:21:57:21 01:00:26:07 01:00:28:02
* FROM CLIP NAME: Norm Macdonald Has a Show_S01E03_Judge Judy_1445392.mp4

009  AX       V     C        00:59:56:12 00:59:57:12 01:00:28:02 01:00:29:02
* FROM CLIP NAME: Black Video
```

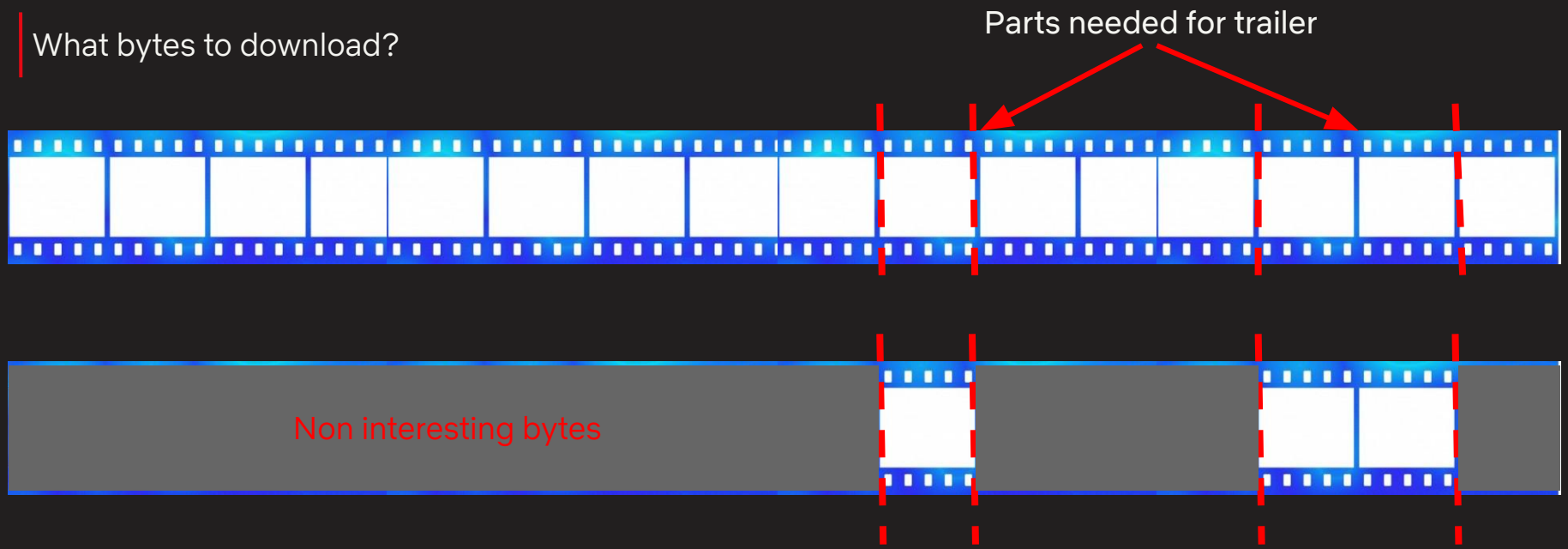
What bytes to download?

- EDL parser
  - Gives expected time ranges
- Movie metadata in DB
  - fps - map time interval to frames
  - Index file - map frame to a byte range

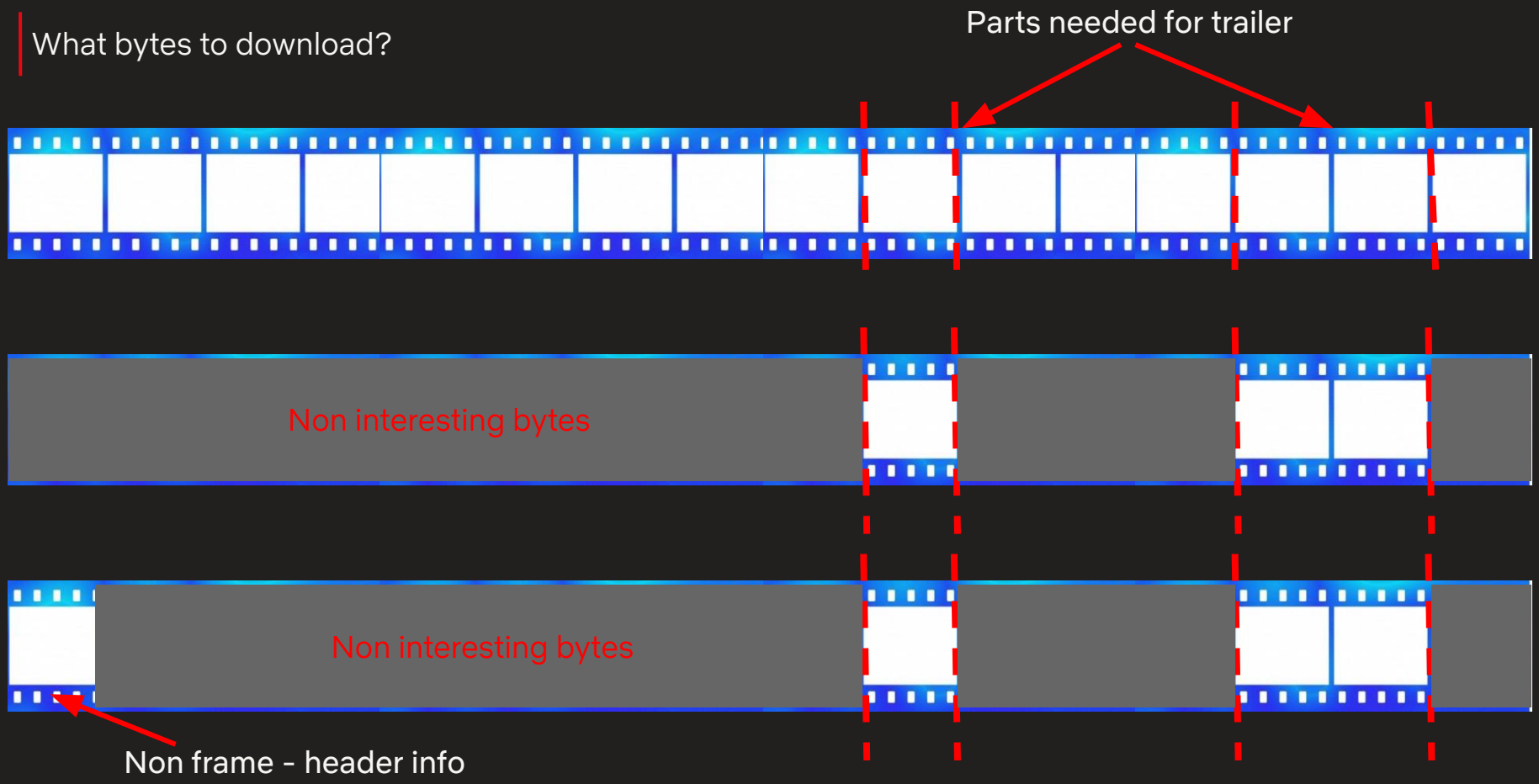
What bytes to download?



What bytes to download?



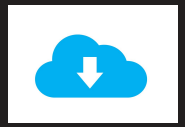
What bytes to download?



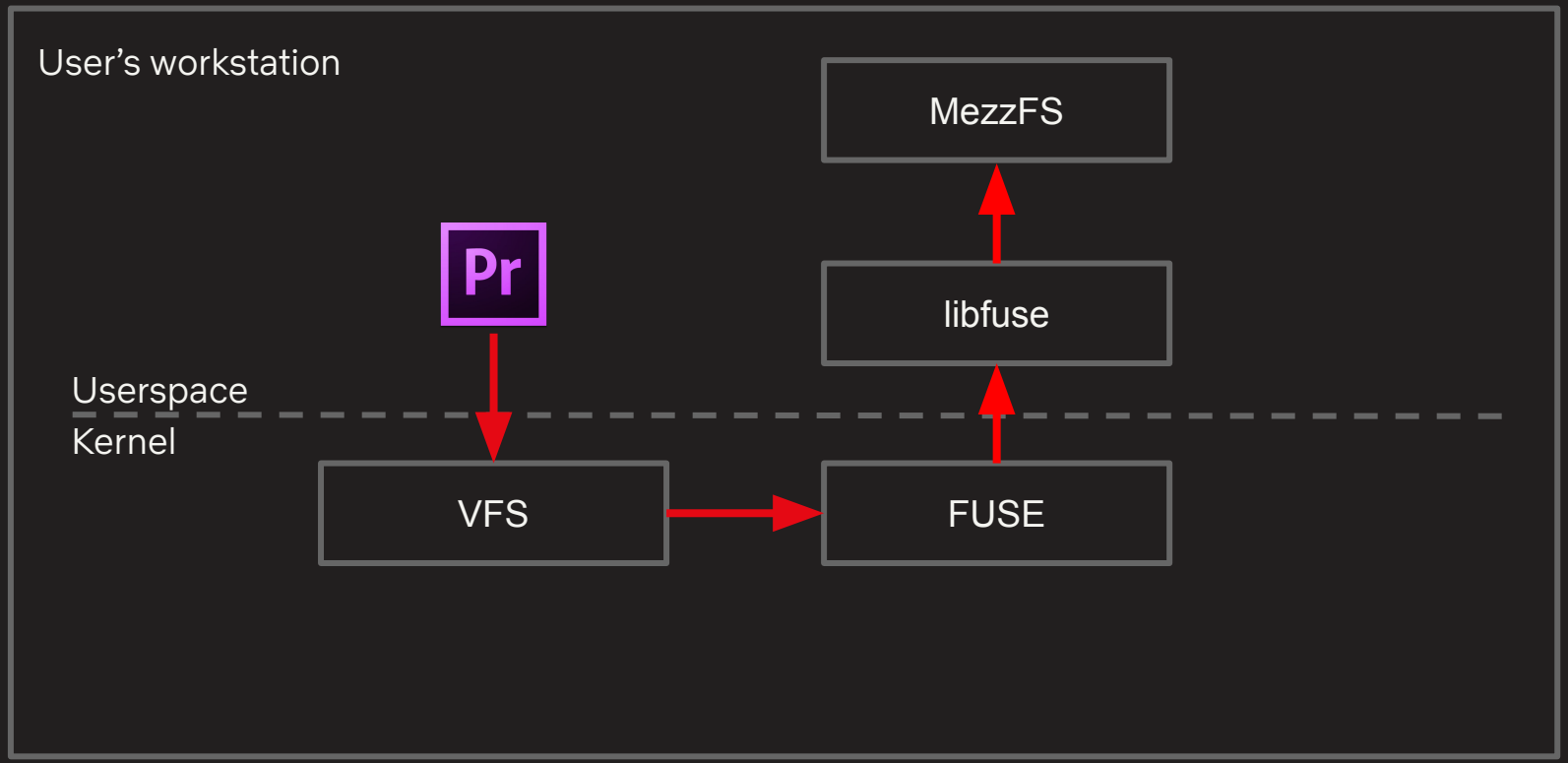
Netflix tool - MezzFS (FUSE wrapper)

- Mounts cloud objects as local files
- Streams bytes from cloud storage to the user's workstation
- Option to cache streamed bytes
- Streaming a cloud object from a byte offset
- Lets user set the context of "interesting bytes" versus "non-interesting bytes"

How to fake bytes?

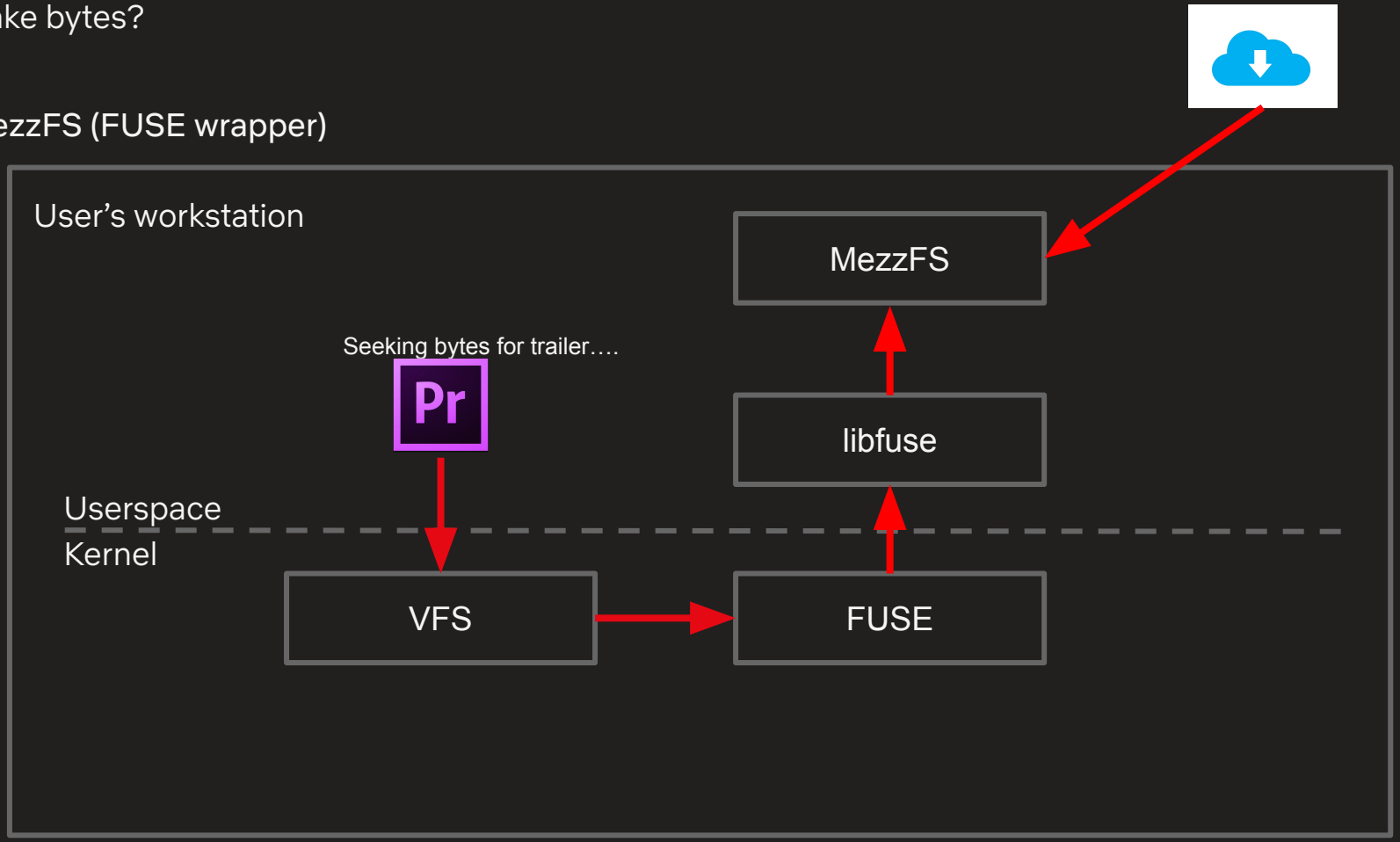


- MezzFS (FUSE wrapper)



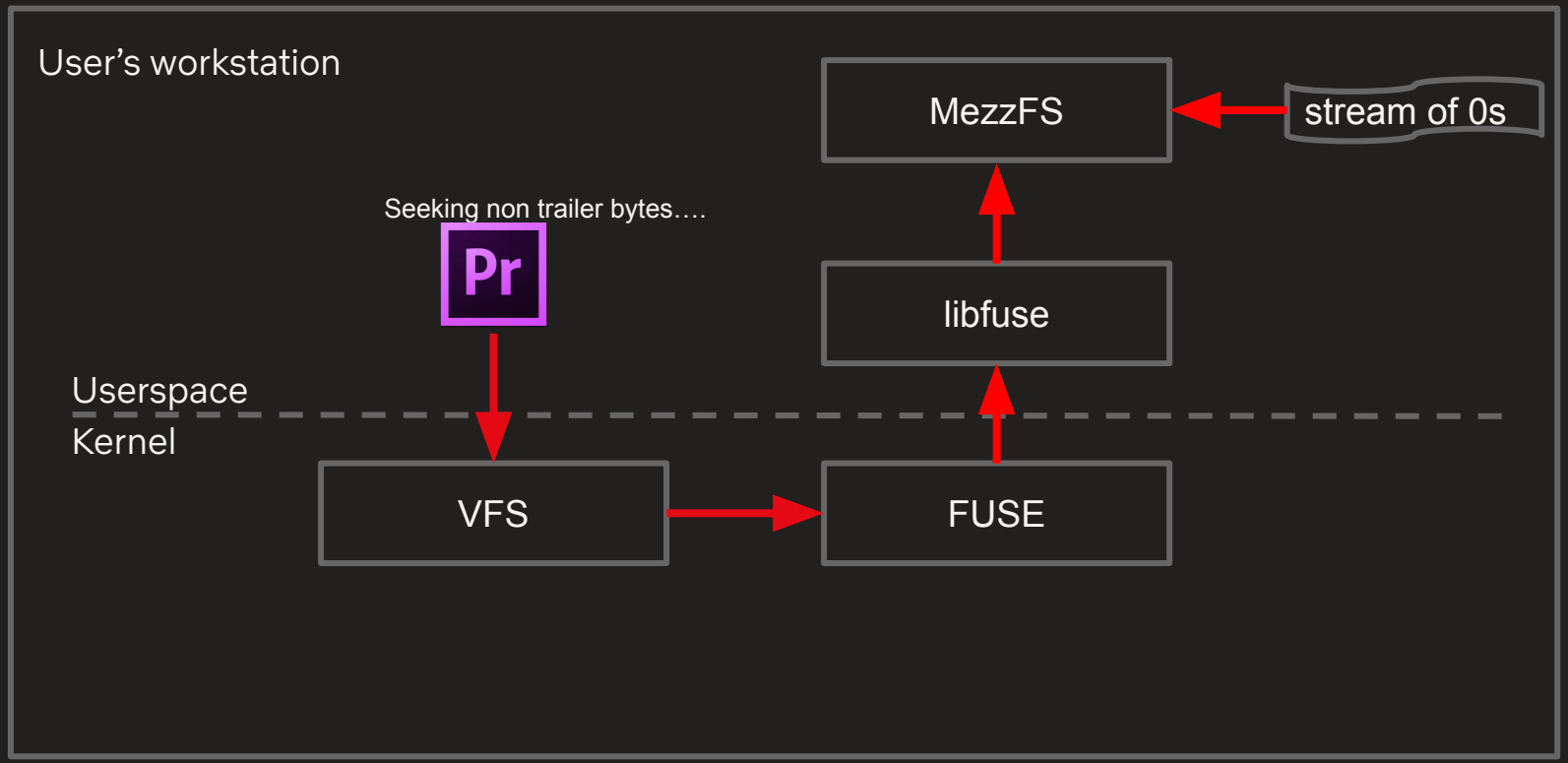
How to fake bytes?

- MezzFS (FUSE wrapper)

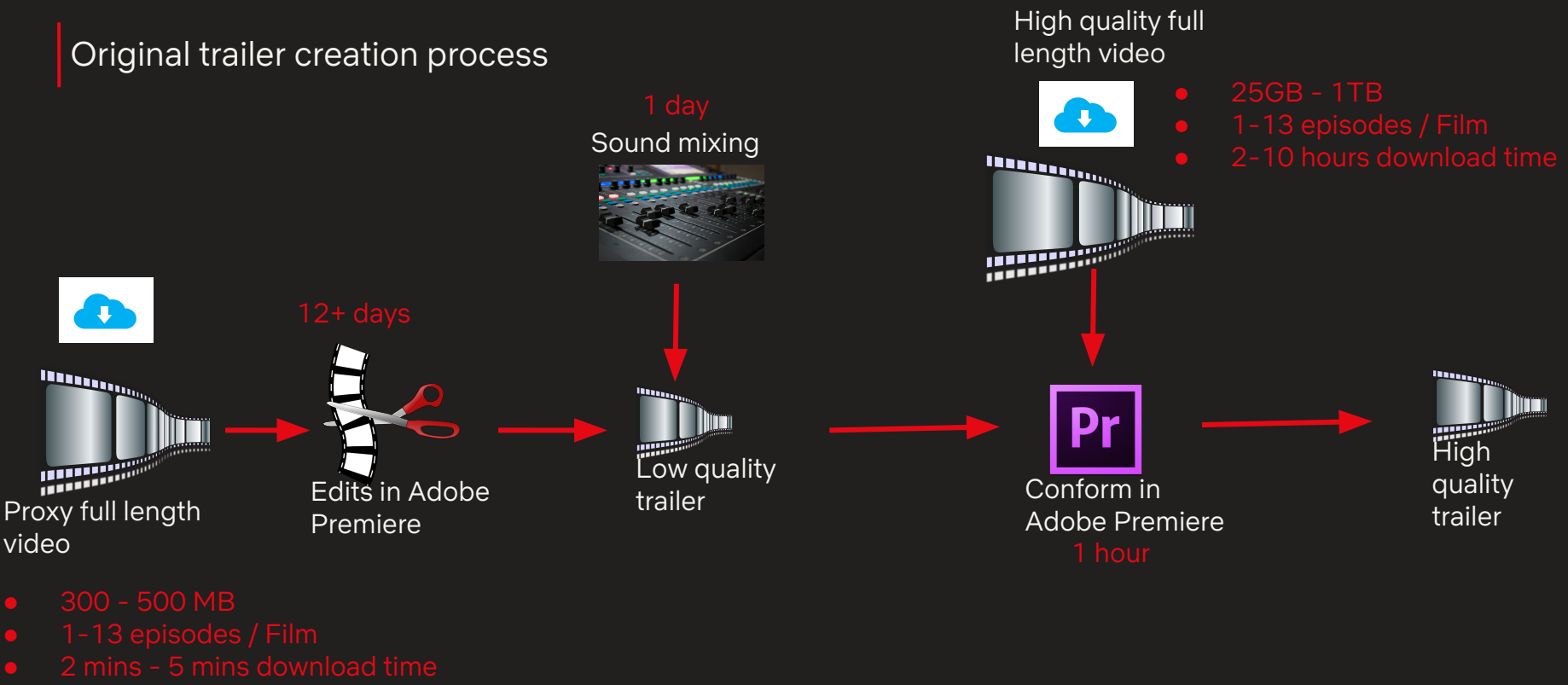


How to fake bytes?

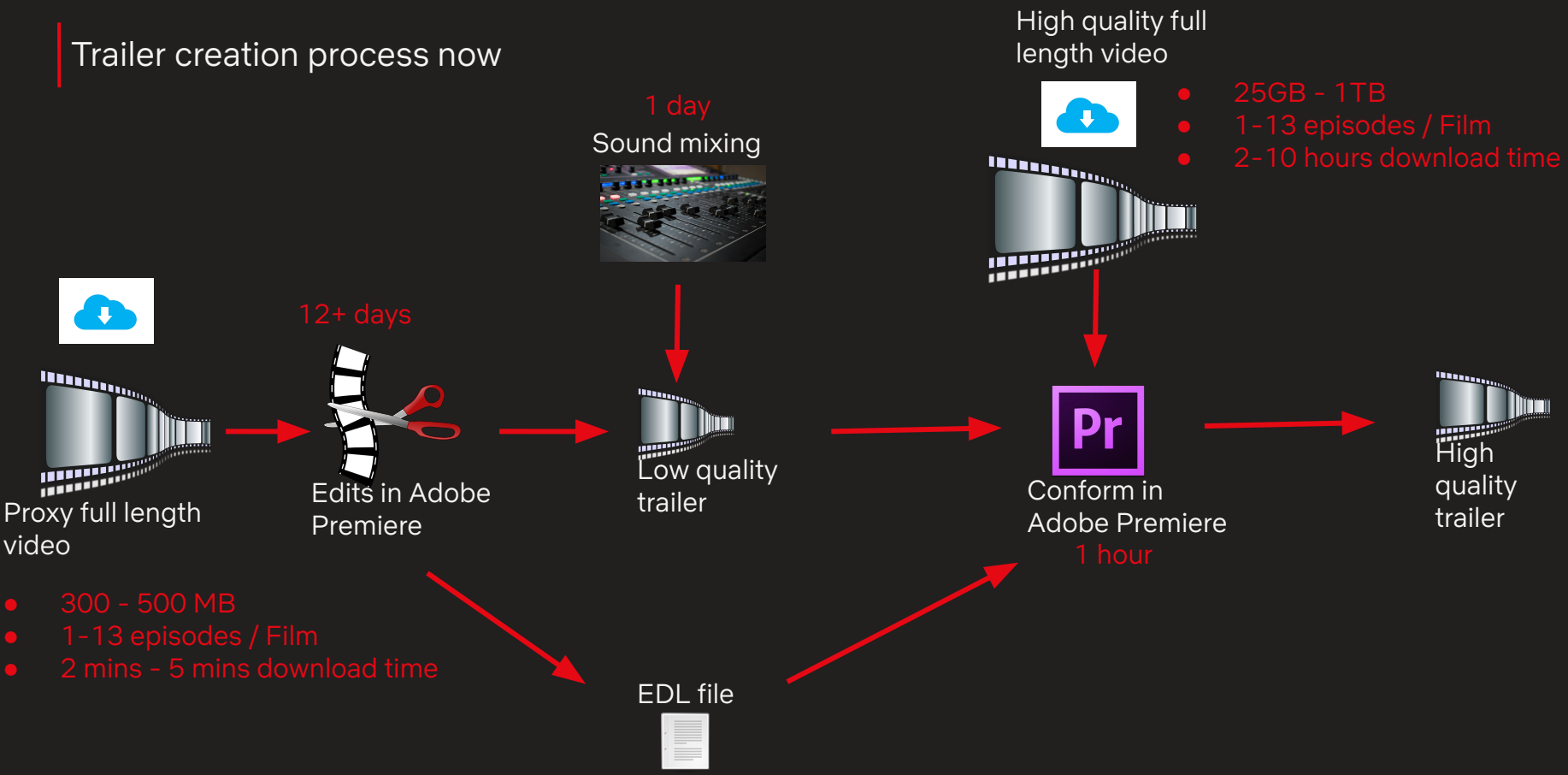
- MezzFS (FUSE wrapper)



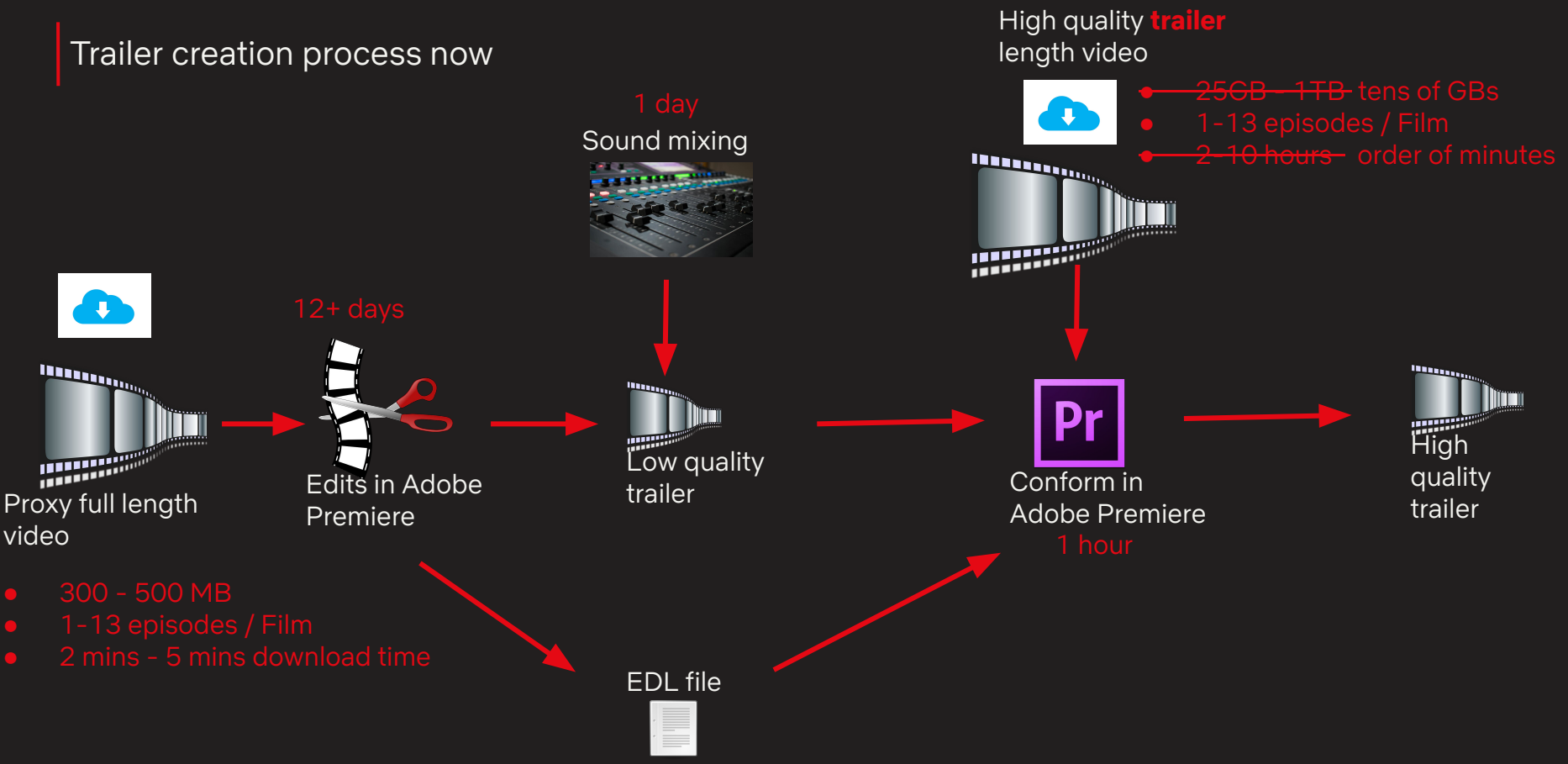
Original trailer creation process



Trailer creation process now



Trailer creation process now



Original Model

Norm Macdonald (1 min Trailer)

- 6 episodes 268 GB / 2 hr 40 mins

Bordertown Recap(3 mins Recap)

- 11 episodes 985 GB / 9 hrs

Current Model

Norm Macdonald(1 min Trailer)

- 6 episodes 11.26 GB / 4 mins

Bordertown Recap(3 mins Recap)

- 11 episodes 70 GB / 18 mins

Benefits of the approach

- Reduced download times
- No heavy disk space requirements
- Security advantage
- Lets creative folks to focus more on their creative work

NETFLIX

Edit Intelligence In Pipelines

Eric Reinecke, Encoding Team @ Netflix

NETFLIX

What do we do?

- Video and Audio encoding at scale
- VMAF Perceptual Video Quality Assessment
- Spearhead development of new codecs
- Media asset analysis and title metadata management
- Workflow tools for the asset creation pipeline

NETFLIX

The Three Rewrites

NETFLIX

Part II Agenda

- Timeline aware pipelines
- Some of the ways the edit has moved through pipelines
- How does OpenTimelineIO enable timeline-aware pipelines

NETFLIX

What would my movie look like if I shipped it right now?

Pitch → **Play**

?

NETFLIX

Where should I focus my energy?

How many frames do I need to animate for this shot?

What does the shot I'm reviewing look like in the context of all the other shots?

How long is the movie running right now?

What credits are shown at what time that I need to have translated?

How many visual effects shots are we up to?

NETFLIX

What are the “Interesting Bytes”?

15 Hours data transfer

22 minutes

NETFLIX

All I have to do is get an EDL?

Option 1: CMX EDL

Editor

CMX 3600 Keyboard

NETFLIX

```
AUTO_ASSEMBLE_DIRECTIVE: ''WAIT''
/* Stop auto-assembly when the following edit is encountered. */
| ''SKIP”
/* Do not perform the following edit. */
| ''BELL''
/ *Sound an audible indicator before performing the following edit.*/
;
```

```
TITLE:   dissolve_test_2
FCM: NON-DROP FRAME
001  TST V     C        01:00:04:05 01:00:04:10 01:00:00:00 01:00:00:05
* FROM CLIP NAME:  clip_A
002  TST V     C        01:00:04:10 01:00:04:10 01:00:00:05 01:00:00:05
002  TST V     D    010 01:00:08:04 01:00:08:14 01:00:00:05 01:00:00:15
* BLEND, DISSOLVE
* FROM CLIP NAME:  clip_A
* TO CLIP NAME:  clip_B
003  TST V     C        01:00:08:14 01:00:08:19 01:00:00:15 01:00:00:20
* FROM CLIP NAME:  clip_B
```

Graham Fisher
@GrahamFisher

#timelinetuesday of the #Trollhunters show finale episodes 3:12 and 3:13. Pure creative bliss to edit. #Netflix #DreamWorks

250 5:57 PM - Jul 24, 2018

57 people are talking about this

Credit: @GrahamFischer

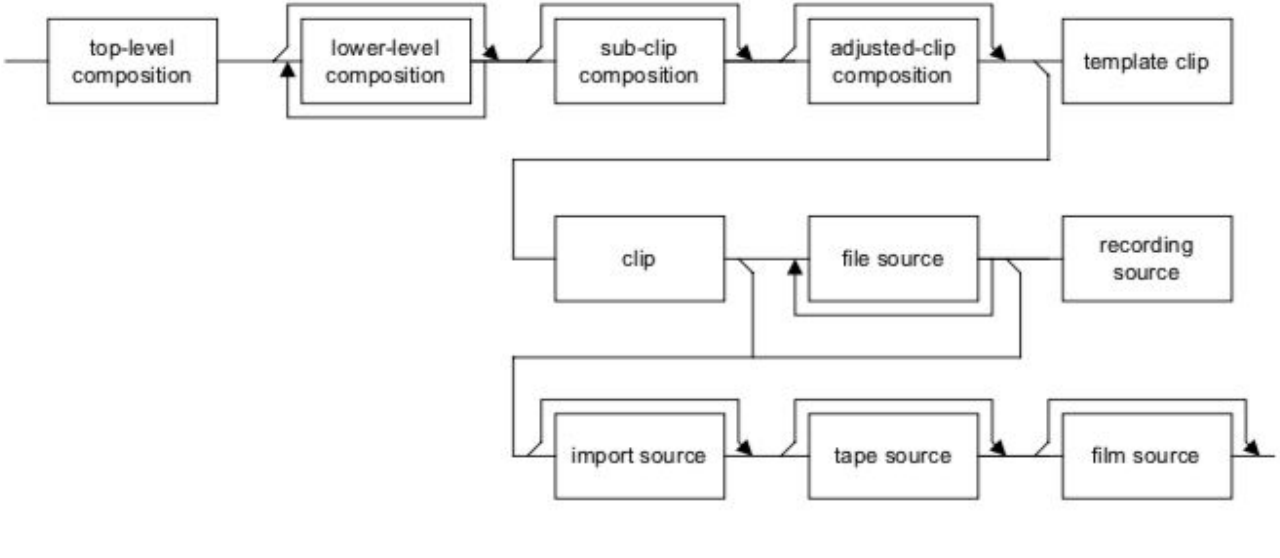
NETFLIX

Credit: @GrahamFischer

NETFLIX

Option 2: Advanced Authoring Format (AAF)

NETFLIX



Referencing between Mobs in the AAF derivation chain

Credit: AAF Association - AAF Edit Protocol

NETFLIX

Option 3: Final Cut Pro XML

Final Cut Pro 7

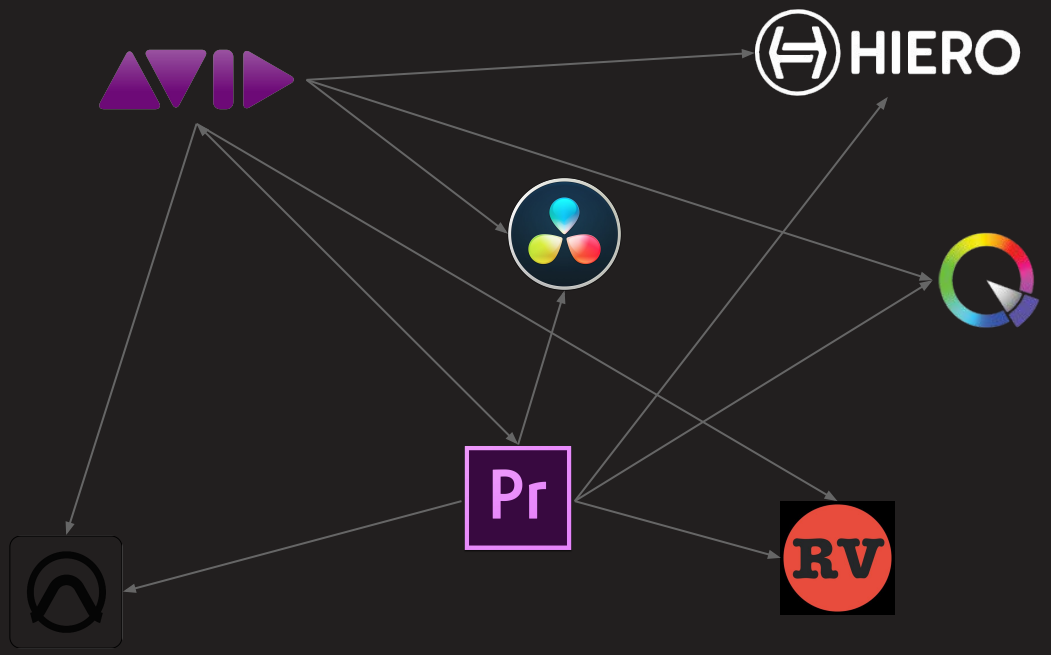
Final Cut Pro X

NETFLIX

```
<xmeml version="4">
  <sequence id="sequence-2">
    <name>dissolve_test_2</name>
    (...)
    <media>
      <video>
        <track>
          <clipitem frameBlend="FALSE">
            <name>clip_A</name>
            <file id="file-1"/>
            <duration>10</duration>
            <start>0</start>
            <end>-1</end>
            <in>86501</in>
            <out>86516</out>
          </clipitem>
          <transitionitem>
            <start>5</start>
            <end>15</end>
            (...)
```

NETFLIX

The EDL Landscape



NETFLIX

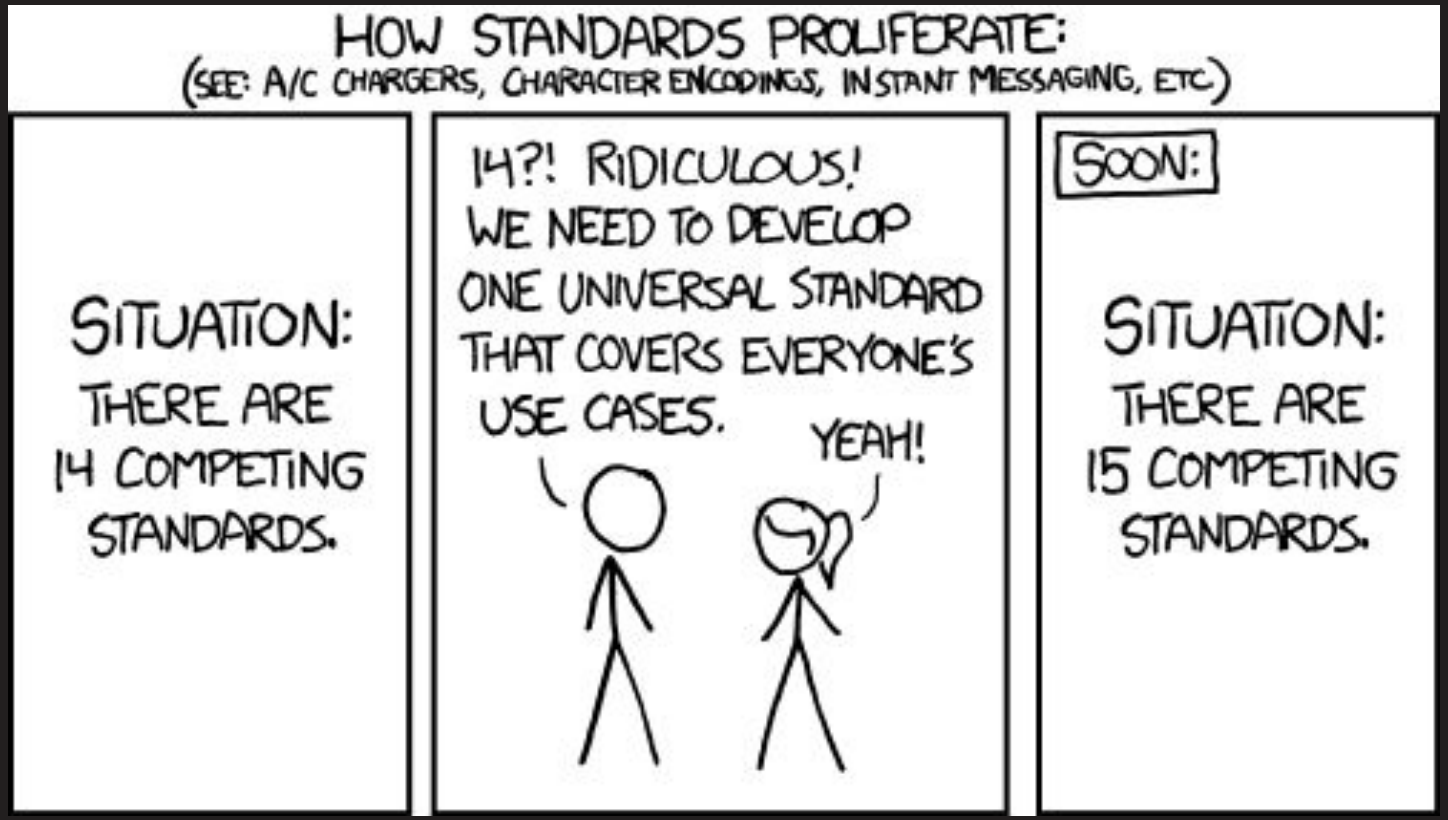
Open Source API and interchange format for editorial timeline information.

NETFLIX

OpenTimelineIO

1. An API defining an editorial data model and functionality for working with it
2. An interchange format to communicate timelines between applications
3. A collection of adapters to import to, and export from that data model

NETFLIX



Source: https://xkcd.com/927/

NETFLIX

OpenTimelineIO - The Timeline Continuum

Simple — **“Just Right”** — **Complex**

EDL — **OTIO** — **AAF**

NETFLIX

OpenTimelineIO

1. An API defining an editorial data model and functionality for working with it
2. An interchange format to communicate timelines between applications
3. A collection of adapters to import to, and export from that data model

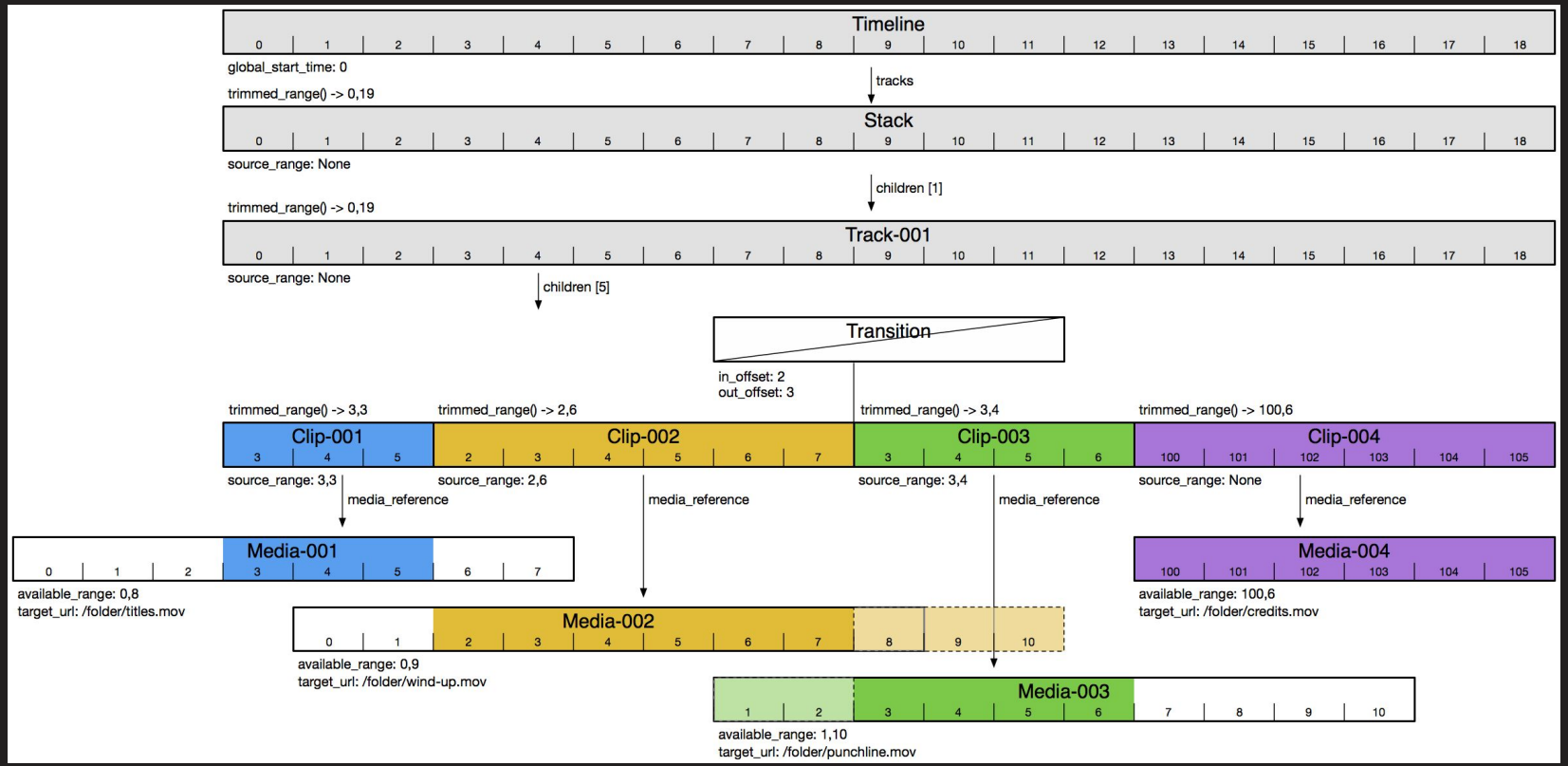
NETFLIX

OpenTimelineIO - Status

- A Pixar-hosted Open Source project driven by real-world use cases
- Contains contributions from lots of studios and industry vendors
- In development since 2016, just released public beta 10
- Currently has a Python API with a C++ API in a preview branch
- Adapters for all the previously described formats
- More adapters can be provided using plugin system

NETFLIX

OpenTimelineIO - Model



NETFLIX

```
{
  "OTIO_SCHEMA": "Clip.1",
  "effects": [],
  "markers": [],
  “Media_reference”: (…)
  "metadata": {
    "cmx_3600": {
      "reel": "TST"
    }
  },
  "name": "clip_A",
  "source_range": {
    "OTIO_SCHEMA": "TimeRange.1",
    "duration": {"OTIO_SCHEMA": "RationalTime.1", "value": 10, "rate": 24.0},
    "start_time": {"OTIO_SCHEMA": "RationalTime.1","value": 86501 "rate": 24.0}
  }
}
```


NETFLIX

```
{
  "OTIO_SCHEMA": "ExternalReference.1",
  "name": bestmovie.mov,
  "target_url": "file:///Volumes/scratch/edl_presentation/bestmovie.mov"
  "available_range": {
    "OTIO_SCHEMA": "TimeRange.1",
    "duration": {
      "OTIO_SCHEMA": "RationalTime.1", "value": 173000,"rate": 24
    },
    "start_time": {
      "OTIO_SCHEMA": "RationalTime.1", "value": 0, "rate": 24
    }
  },
  "metadata": {
    "nflx": {
      "external_id": "33986271-053e-4240-bcc4-72217ce3b647","movie_id": 123456
    }
  }
}
```


OTIO Example: Media Reference

NETFLIX

OpenTimelineIO - Find Items and Ranges Used

```python
import opentimelineio as otio

def clip_ranges_for_edl(edl_path):
    # read the edl
    tl = otio.adapters.read_from_file(edl_path)

    asset_to_range_map = {}
    for video_track in tl.video_tracks():
        for clip in video_track.each_clip():
            # Get the netflix movie id if available, fall back on the clip name
            netflix_metadata = clip.media_reference.metadata.get('nflx', {})
            clip_id = netflix_metadata.get('movie_id', clip.name)

            clip_ranges = asset_to_range_map.setdefault(clip_id, [])
            clip_ranges.append(clip.source_range)

    return asset_to_range_map
```

NETFLIX

OpenTimelineIO - Find Items and Ranges Used

```
import opentimelineio as otio

def combined_ranges(ranges):
    ordered_ranges = sorted(ranges, key=lambda r: r.start_time)
    new_ranges = []
    running_range = None
    for r in ordered_ranges:
        if running_range is None:
            running_range = r
        elif r.overlaps(running_range):
            running_range = running_range.extended_by(r)
        else:
            new_ranges.append(running_range)
            running_range = r

    new_ranges.append(running_range)

    return new_ranges
```

NETFLIX

OpenTimelineIO - Update File References

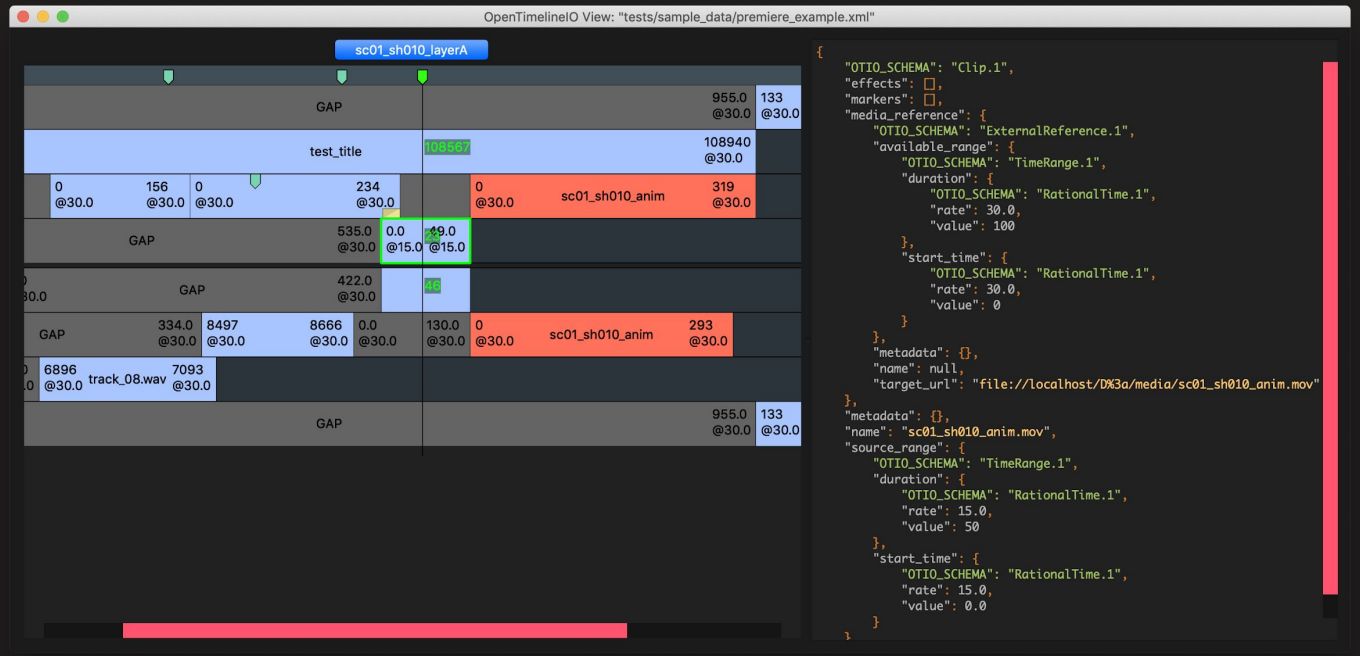
```
#!/usr/bin/env python
import opentimelineio as otio
```

TODO: write me, I should simulate the part of Mangala's workflow that updates file URLs

```
timeline = otio.adapters.read_from_file("/Volumes/scratch/GF5_trailer.xml")
```

NETFLIX

Viewer Application



NETFLIX

OpenTimelineIO - Participation Encouraged!

PixarAnimationStudios / OpenTimelineIO

Unwatch 91 Unstar 3

Code | Issues 76 | Pull requests 18 | Projects 4 | Wiki | Insights

Open Source API and interchange format for editorial timeline information. http://opentimeline.io

editorial timeline interchange-format animation vfx cut python film

OpenTimelineIO Shared publicly

30 of 32 topics (21 unread)

Members · About

Welcome to the Open Timeline IO discussion forum.

OpenTimelineIO is an Open Source API and interchange format for editorial timeline information.

http://opentimeline.io/
https://github.com/PixarAnimationStudios/OpenTimelineIO

OpenTimelineIO Beta 10 Release (1)
By Joshua Minor - 1 post - 0 views

Apr 23

Handling Audio in OTIO

NETFLIX

http://opentimeline.io

```
pip install opentimelineio
```

NETFLIX

Acknowledgements

- Josh Minor @Pixar
- Stephan Steinbach @Pixar

NETFLIX

Questions?

NETFLIX