DEVIEW
2019

고품질 라이브 서비스의 유지 비결
(라이브 미디어 스트리밍 서버 개발기)

강인철
동영상플랫폼
NAVER

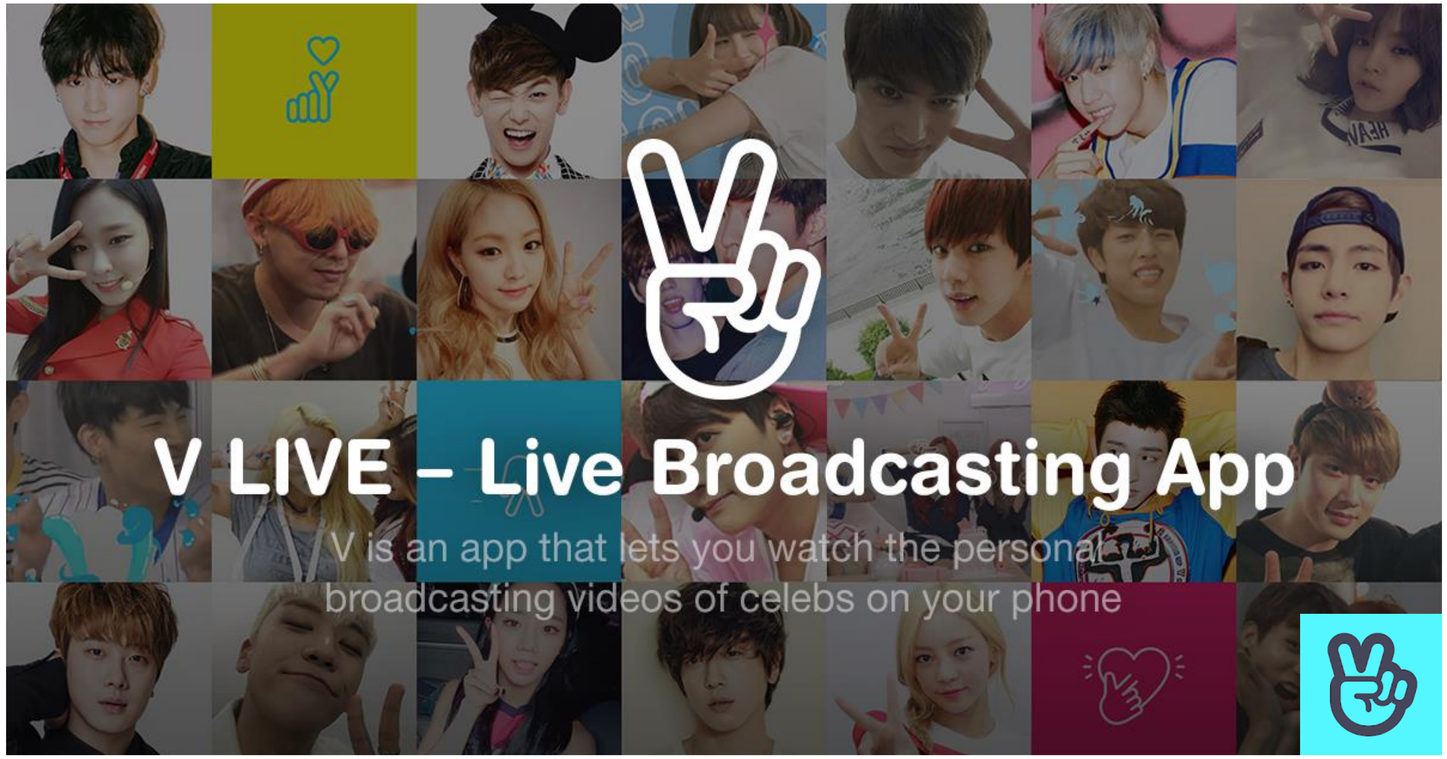
CONTENTS

DEVIEW
2019

DEVIEW
2019

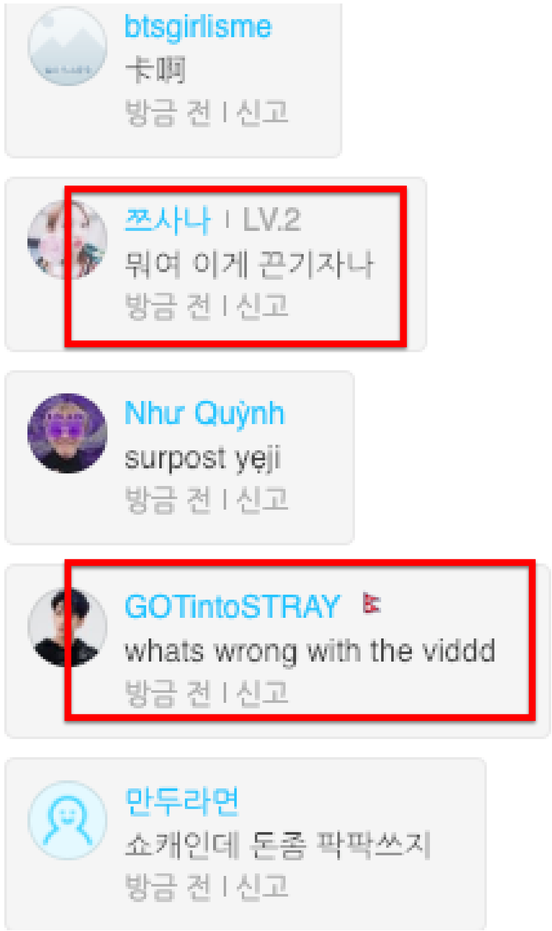
1. 네이버의 고민과 결정

1. 네이버의 첫 라이브 서비스



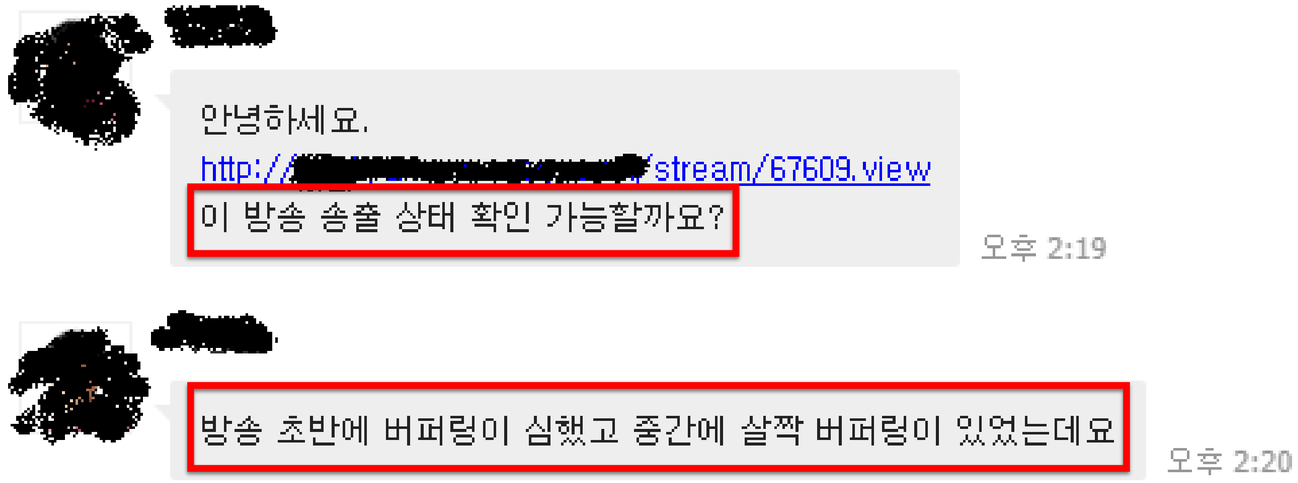
1. 대표적인 유저 보이스 : 재생 끊김. 버퍼링.

DEVIEW 2019



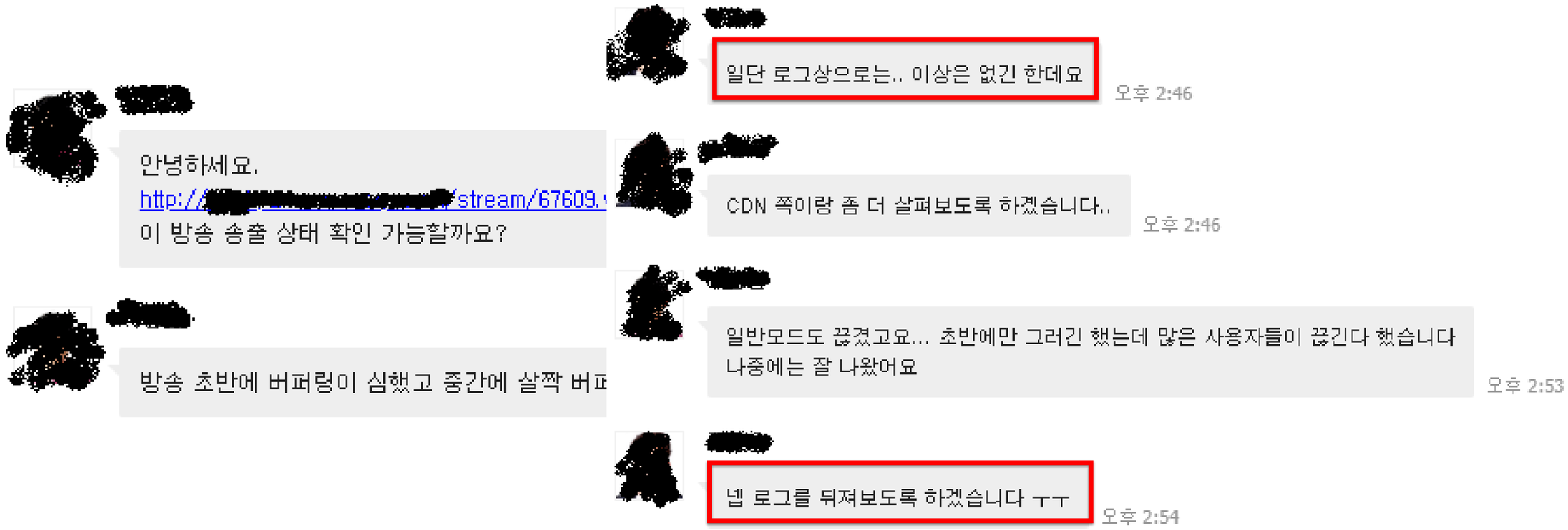
1. 유저 보이스는 강력하다.

DEVIEW
2019



1. 어디가 문제인지…

DEVIEW 2019



1. 이슈가 있다는 걸 알았어도

Sent: 2016-10-12 (수) 19:00:21
Subject: vlive OTG 연동시 레코딩 파일의 음성 싱크 이슈

안녕하세요, [redacted]입니다.

OTG 카메라 연동시 [redacted]레코딩 파일의 음성이 앞으로 shift 되는 사항을 체크하고 있는데요,
[redacted]측과 기술 확인 시 인코더 설정 정보도 필요하다고 하여 상세 설정 정보도 받기 위해 문의드려요.
Encoder 종류가 필요하다고 하는데 인코딩 상세 옵션 정보 포함하여 전달주시면 참고하겠습니다.

1. 해결을 장담하기가 어려웠습니다.

DEVIEW 2019

Sent: 2016-10-12 (수) 19:00:21
Subject: vlive OTG 연동시 레코딩 파

안녕하세요, [redacted]입니다.

OTG 카메라 연동시 [redacted]레코딩 파일의
[redacted]측과 기술 확인 시 인코더 설정 정
Encoder 종류가 필요하다고 하는데 인코딩

Sent: 2016-11-02 (수) 19:13:47
Subject: RE: vlive OTG 연동시 레코딩 파일의 음성 싱크 이슈

안녕하세요, [redacted]입니다.

계속 테스트로 장비만 대여했었으나 크게 업데이트 사항이 없어 메일은 오랜만에 드리게 되었습니다. ^^;
재현 및 debug 정보들로 테스트들로 통해 최종[redacted] 측으로부터 내용을 확인했습니다.

레코딩 부분의 이슈는 bug로 확인하였으며 개선할 예정이나 개발 일정은 확정되지 않았다고 합니다.
그래서 해당 bug 수정이 언제 개발되어 몇 버전에 추가될지 명확한 사항은 공유드리지 못해 아쉽네요.

1. 해결을 장담하기가 어려웠습니다.

DEVIEW 2019

Sent: 2016-10-12 (수) 19:00:21
Subject: vlive OTG 연동시 레코딩 파

안녕하세요, [redacted]입니다.

OTG 카메라 연동시 [redacted]레코딩 파일의

Sent: 2016-11-02 (수) 19:13:47
Subject: RE: vlive OTG 연동시 레코딩 파일의 음성 싱크 이슈

안녕하세요, [redacted]입니다.

재현 및 debug 정보들로 테스트들로 통해 최종[redacted]측으로부터 내용을 확인했습니다.

레코딩 부분의 이슈는 bug로 확인하였으며 개서할 예정이나 개발 일정은 확정되지 않았다고 합니다.

그래서 해당 bug 수정이 언제 개발되어 몇 버전에 추가될지 명확한 사항은 공유드리지 못해 아쉽네요.

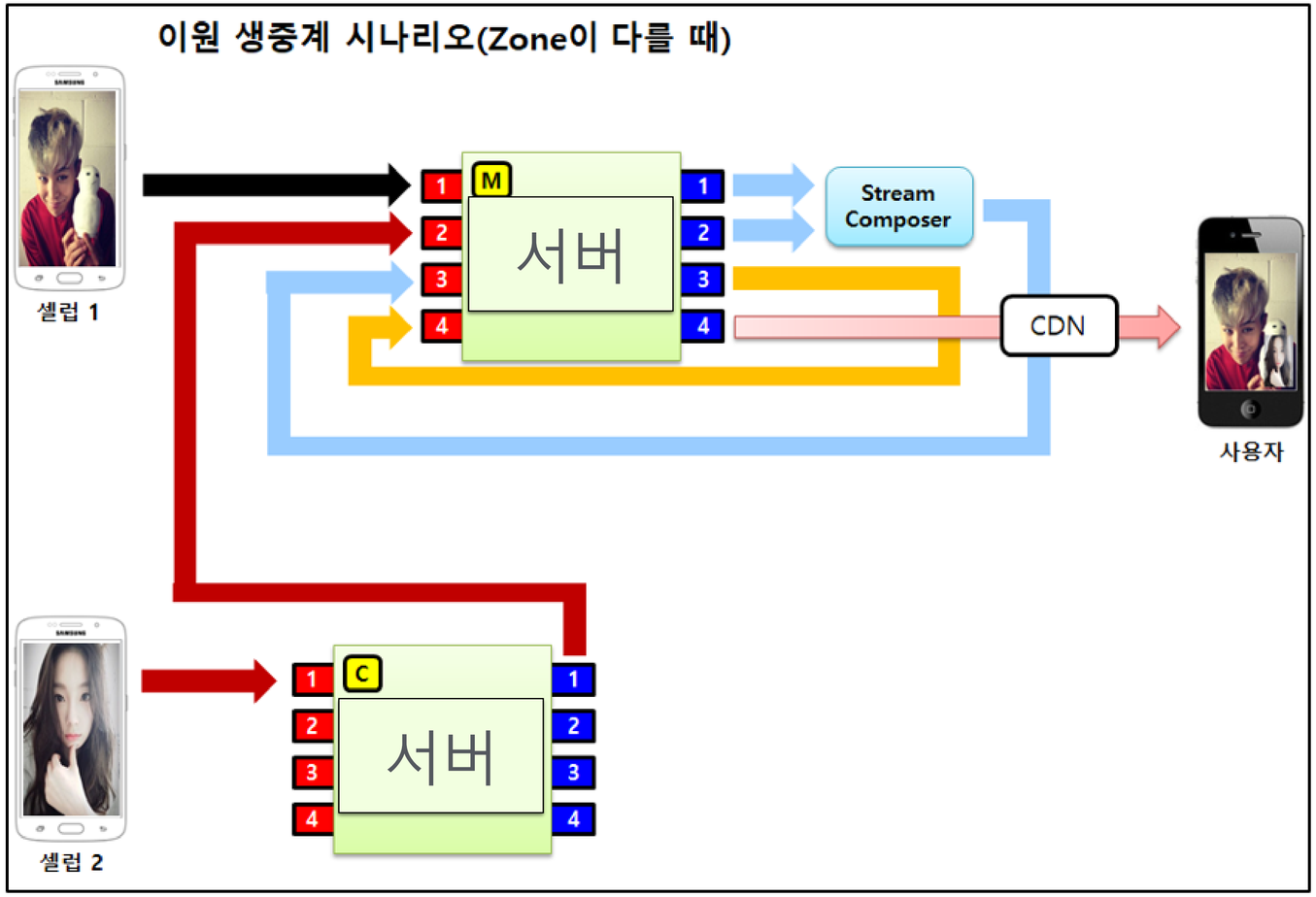
문제점1. 이슈 구간 파악 & 해결이 어려움

1. 리더님의 한마디

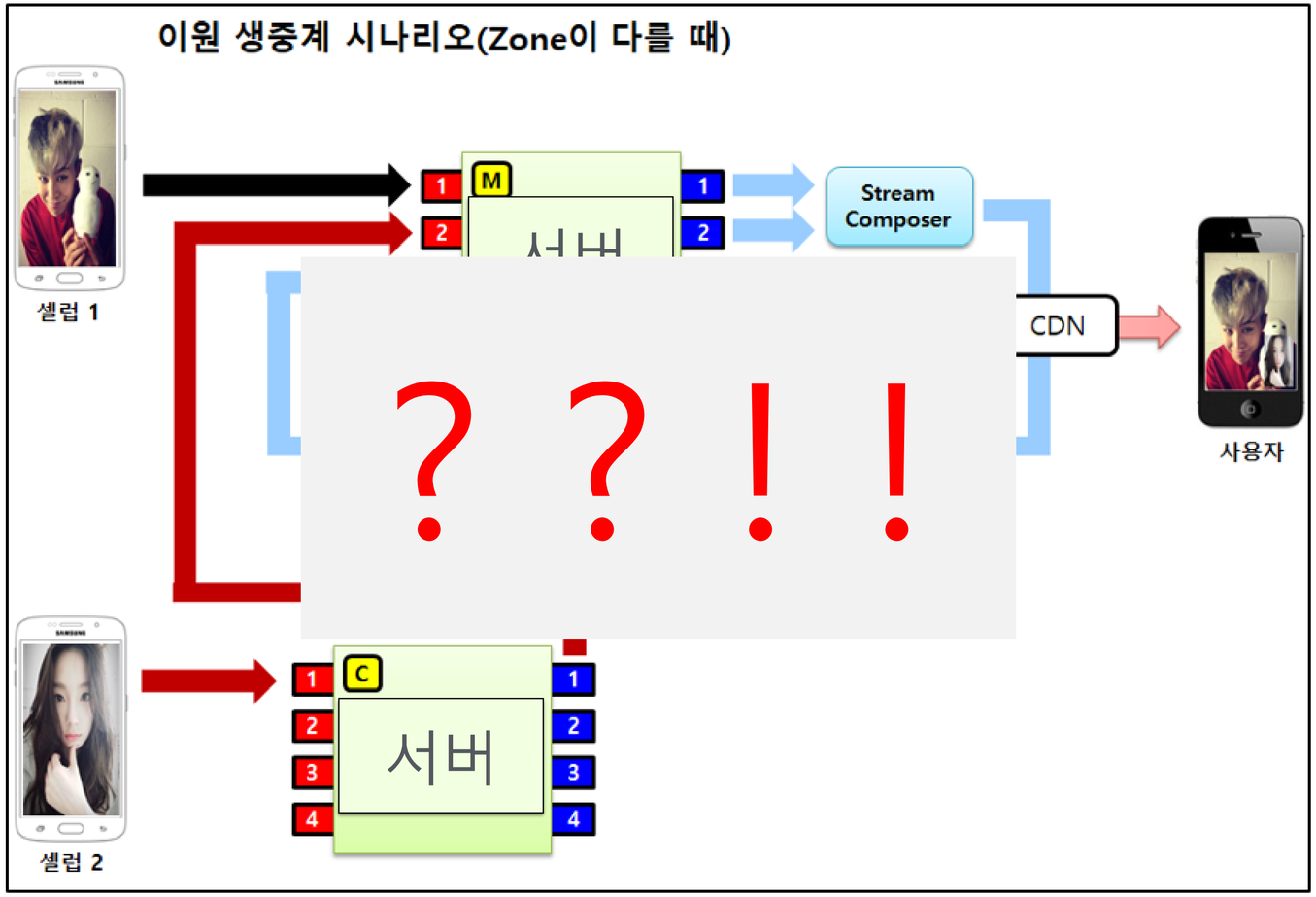
온라인으로 합방하면

재미있지 않을까?

1. 넵! 이원생중계 설계안입니다.

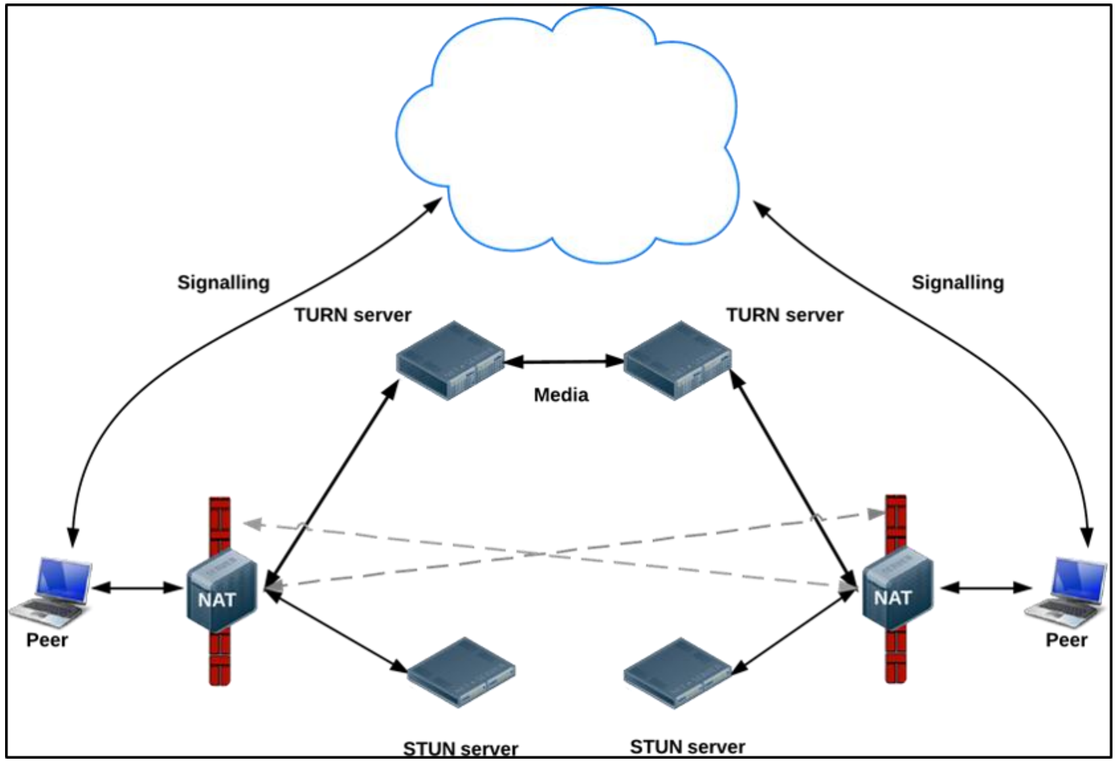


1. 복잡한 구조


이원 생중계 시나리오(Zone이 다를 때)
셀럽 1
M
1
2
1
2
Stream Composer
CDN
사용자
? ? ! !
셀럽 2
C
1
2
3
4
서버
1
2
3
4

1. 인프라 구조도 다시 설계 해야해요.

DEVIEW
2019



1. 인프라도 추가 확보해야 해요

문제점2. 특화된 기능을 추가하기 어렵다

1. 어려움이 있던 이유?

우리만의
미디어 서버가
없다

1. 그래서 네이버의 결정은

기술 내재화로

정면돌파!

DEVIEW
2019

2. 라이브 미디어 스트리밍 서버 소개

2. 2017년 3월, 프로젝트 킥오프

DEVIEW
2019

목표 설정의 고민

어떤 기본기를 갖춰야 할까?

2. 내재화 목표 : 3가지 키워드

DEVIEW
2019

성능

실시간 처리

2. 내재화 목표 : 3가지 키워드

성능

실시간 처리

품질

좋은 화질을 빠르게

2. 내재화 목표 : 3가지 키워드

성능

실시간 처리

품질

좋은 화질을 빠르게

안정성

24/7 서비스

2. 내재화 목표 : 3가지 키워드

DEVIEW
2019

Live

실시간으로

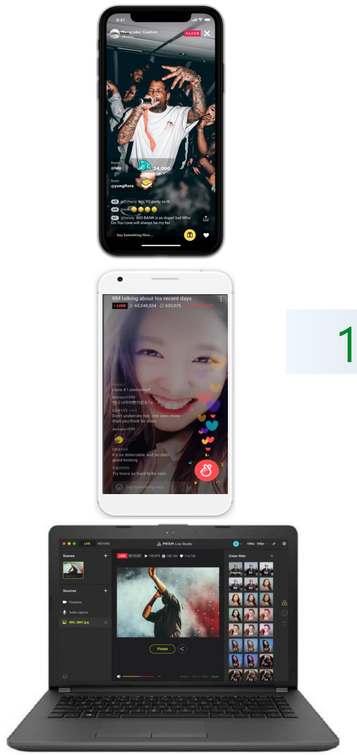
Media Streaming

좋은 화질을 보다 빠르게

Server

안정적으로

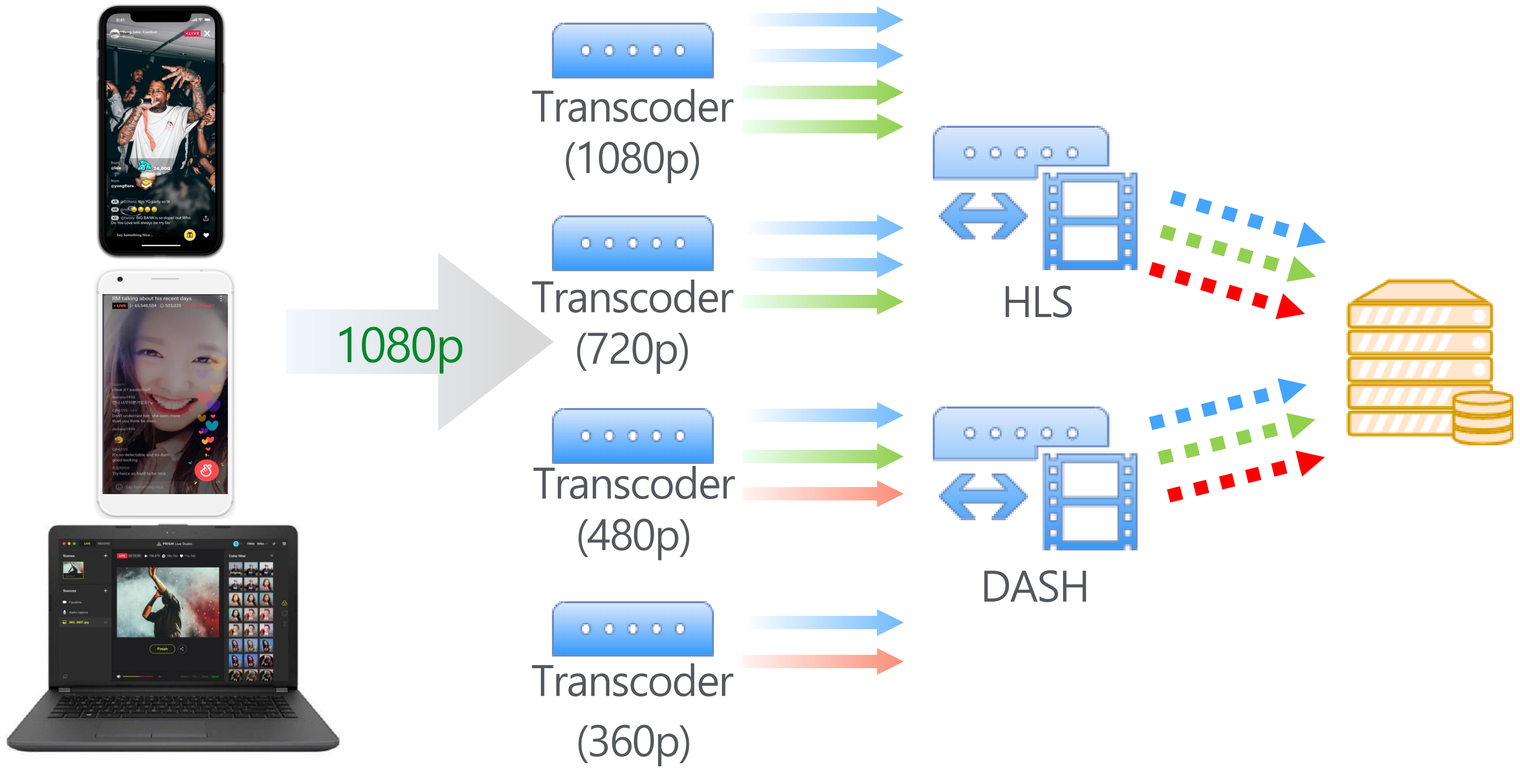
2. 송출자의 원본 영상을 수신



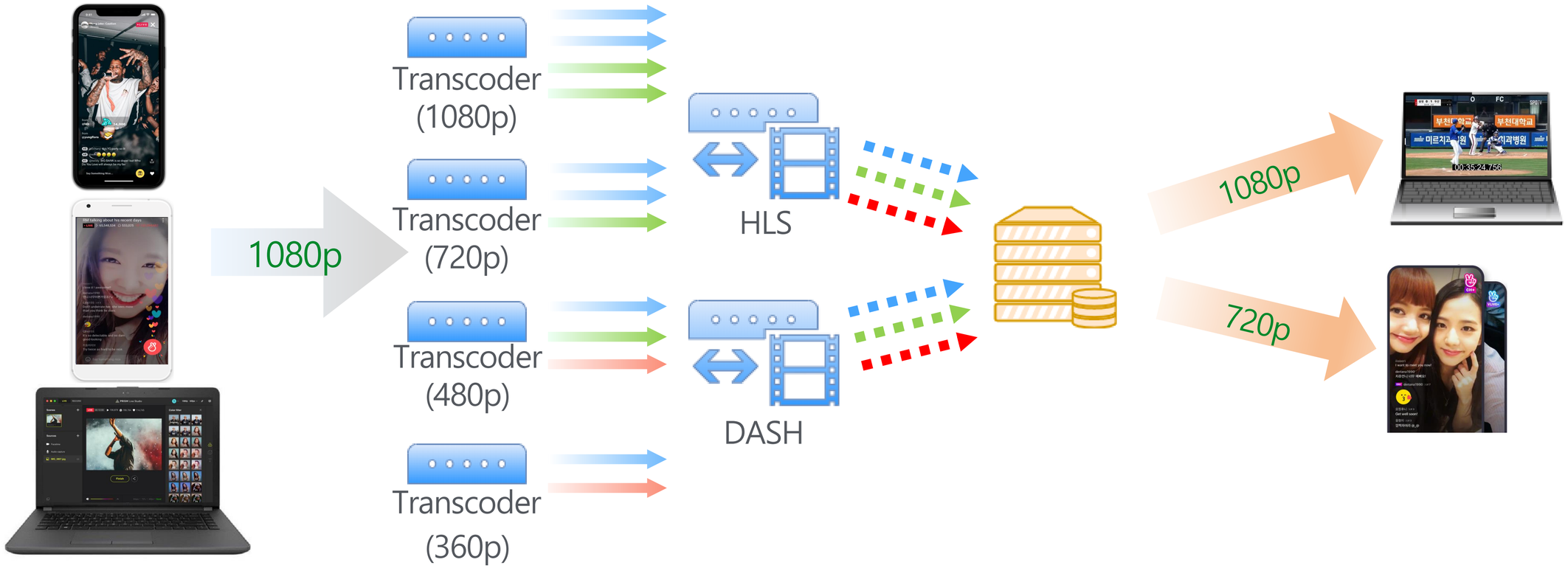
1080p

라이브 미디어
스트리밍 서버

2. 다양한 코덱/해상도/프로토콜 처리



2. 시청자에게 전달하는 중간 서버



2. 현재까지 확보된 기술 Spec

DEVIEW 2019

Codec / Format

- H.264 / HEVC
- AAC / MP3
- MPEG2-TS, MP4
- MKV, FLV
- WebVTT, SRT, TTML2

Processing

- 트랜스코딩 / 레코딩
- 영상/음성/자막 합성
- 동시 송출
- Custom 이벤트 삽입
- GPU Processing

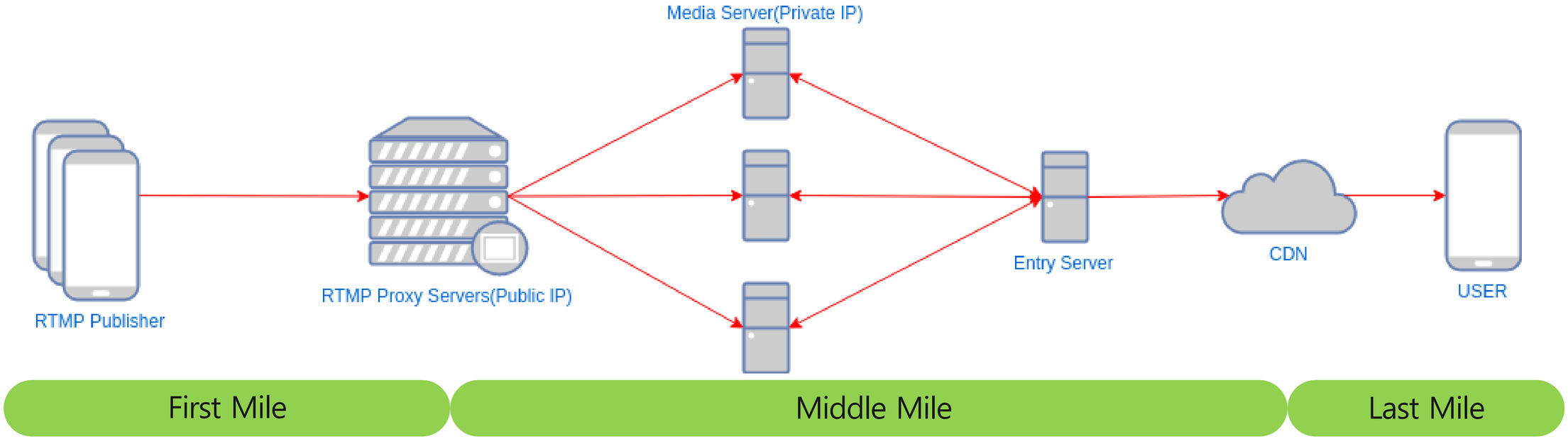
Protocol

- RTMP
- HLS
- MPEG-DASH
- SRT
- WebRTC

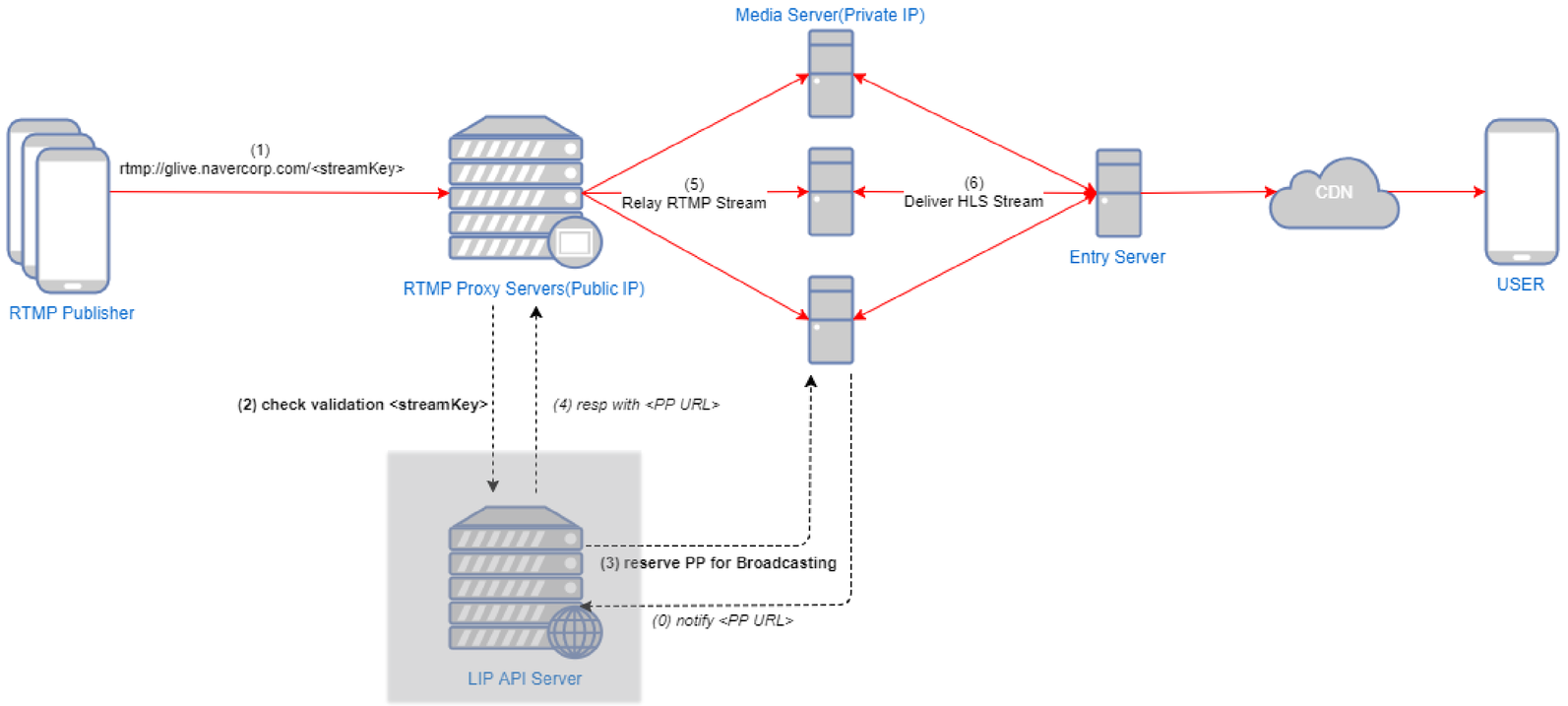
2. 전 구간 기술 확보

DEVIEW 2019

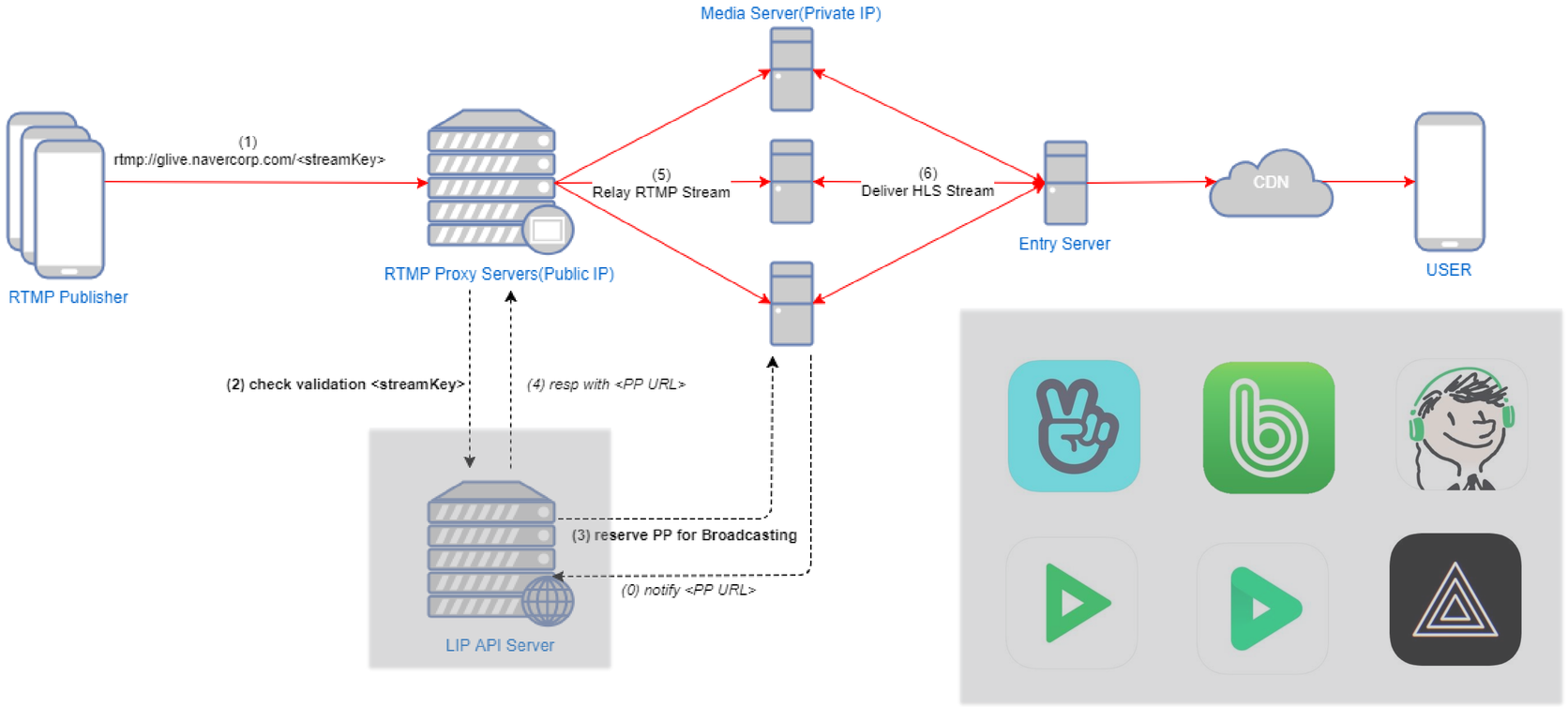
✓ 송출부터 미디어서버, 그리고 재생까지



2. Live Cloud Platform 구축



2. 라이브 스트리밍 적용 서비스 확대



2. 글로벌 플랫폼 구축

DEVIEW
2019

2. 데이터로 보는 서비스 적용 현황

DEVIEW 2019

약 17만건
방송 수

약 7만 7천 시간
실시간 인코딩 시간

2018.04.27
첫 방송

DEVIEW
2019

3. 네이버의 기술과 노하우

DEVIEW
2019

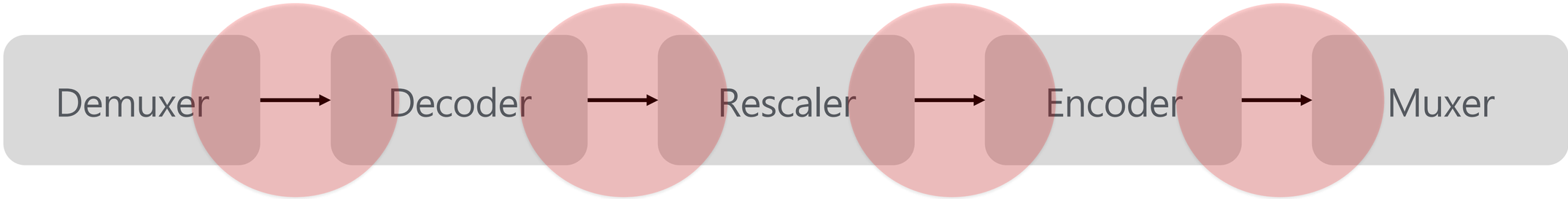
3-1. 실시간 처리를 위한 최적화 기법

3.1 성능 최적화를 해야하는 이유

- 라이브(실시간) 스트리밍 서버
- 동영상을 **사용자용 미디어로 실시간 변환 처리**해야 하는 서버
- 늦게 처리될수록 사용자의 **버퍼링은 지속**

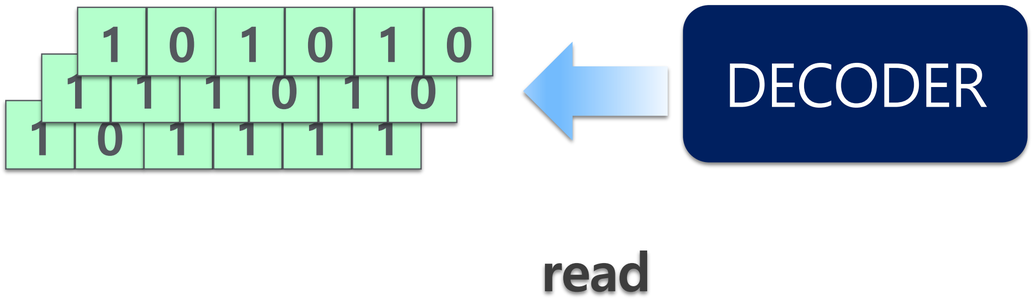
3.1 최초 설계에서의 성능 이슈

- 추상화 단계를 높여 각 구간의 모듈화를 목표
- Data Processing간 빈번한 Memory 할당과 복사 발생



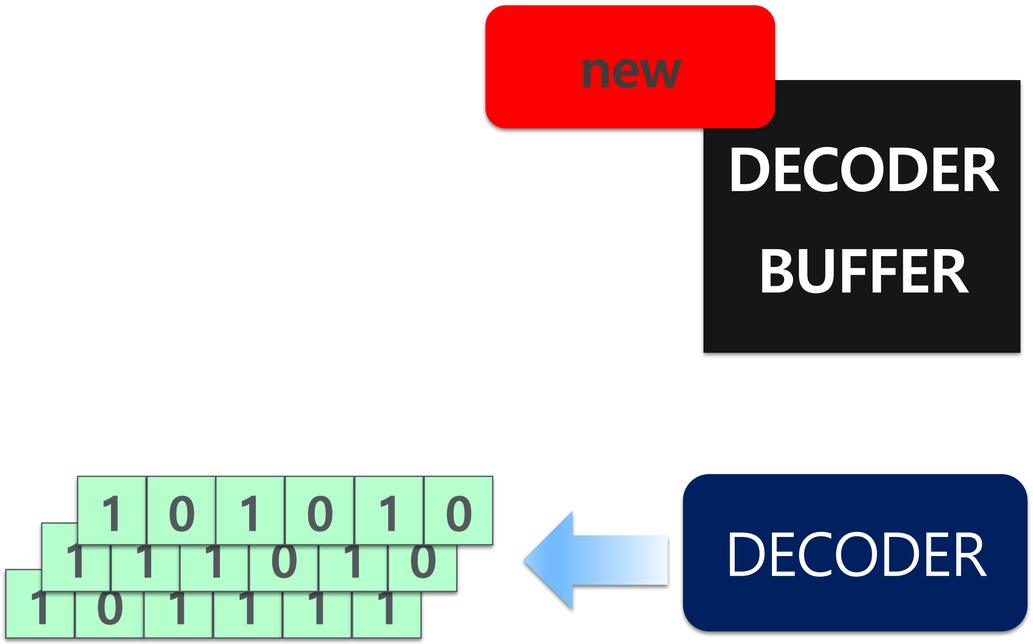
3.1 예) 영상 압축 해제와 압축 과정(1/7)

DEVIEW
2019



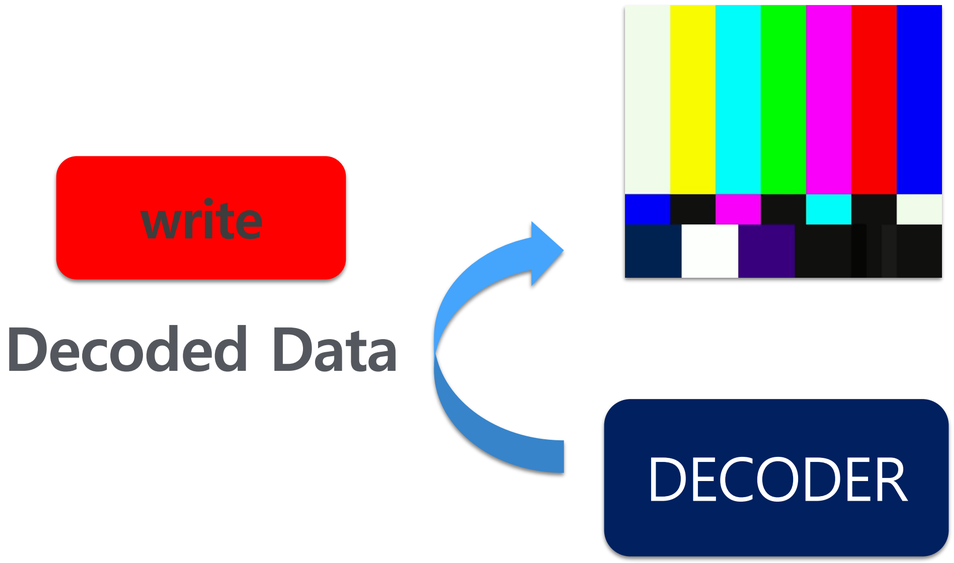
3.1 예) 영상 압축 해제와 압축 과정(2/7)

DEVIEW
2019

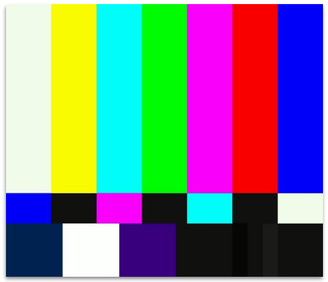


3.1 예) 영상 압축 해제와 압축 과정(3/7)

DEVIEW 2019



3.1 예) 영상 압축 해제와 압축 과정(4/7)



DECODER

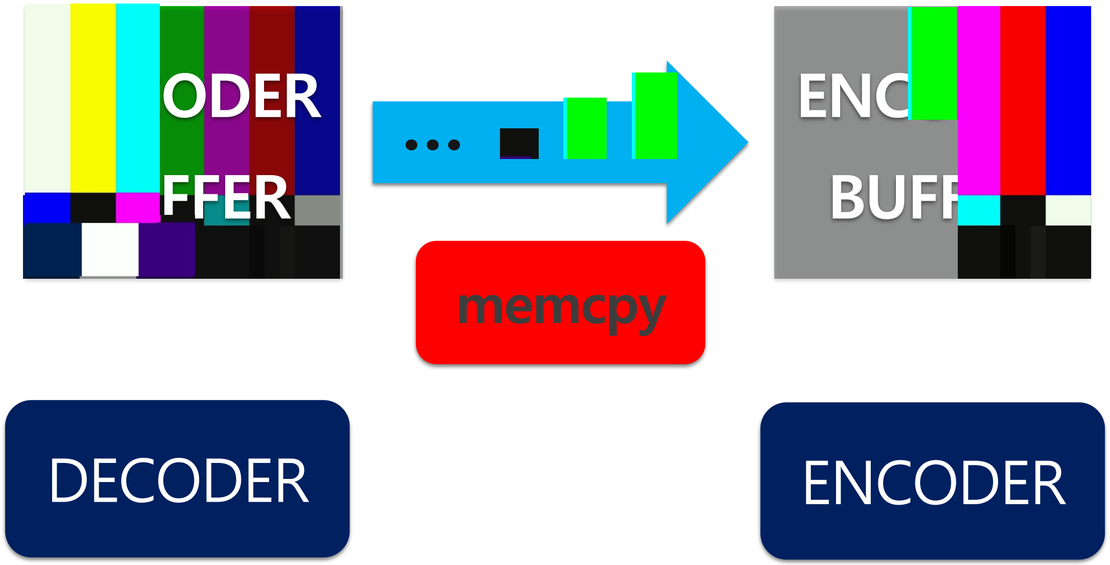
new

ENCODER
BUFFER

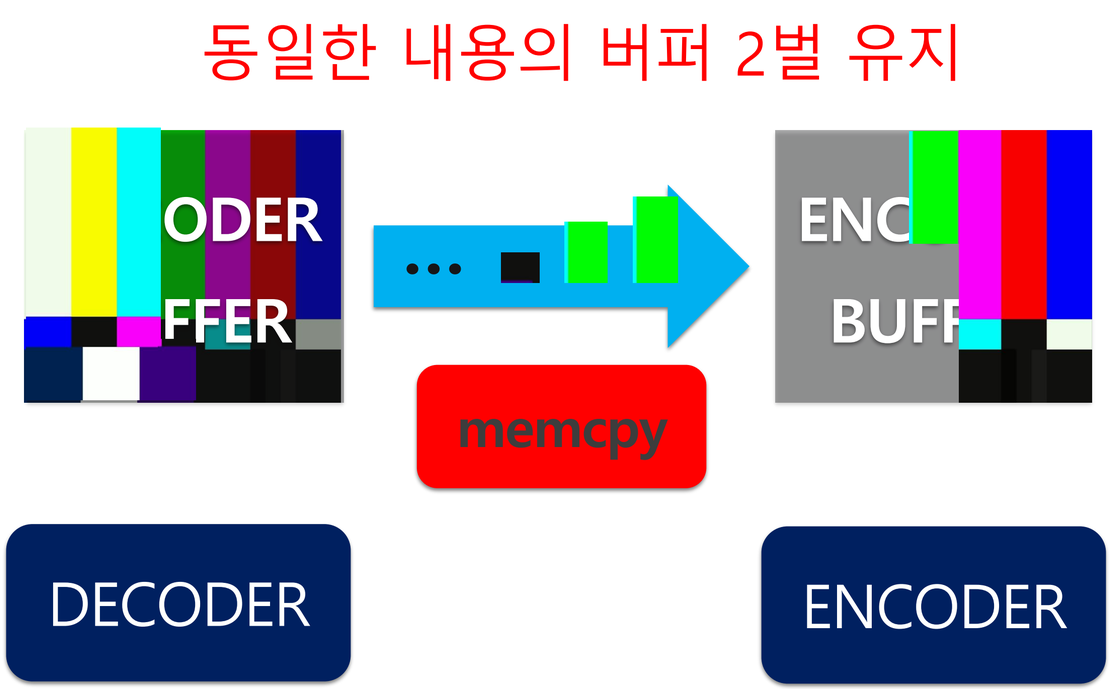
ENCODER

3.1 예) 영상 압축 해제와 압축 과정(5/7)

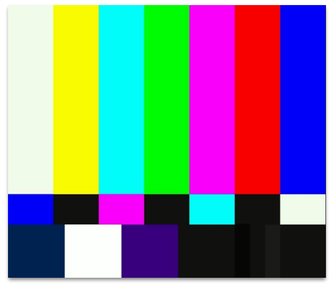
DEVIEW 2019



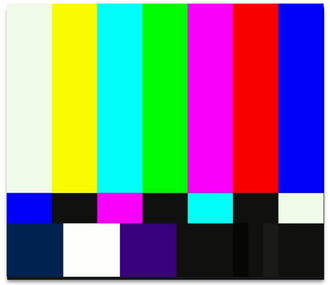
3.1 예) 영상 압축 해제와 압축 과정(5/7)



3.1 예) 영상 압축 해제와 압축 과정(6/7)



DECODER



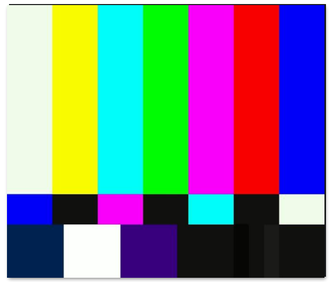
ENCODER

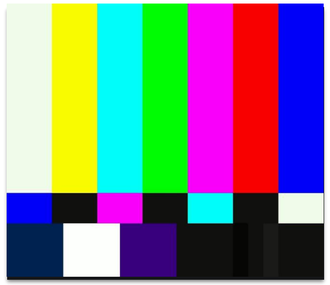
read

Decoded Data

3.1 예) 영상 압축 해제와 압축 과정(7/7)

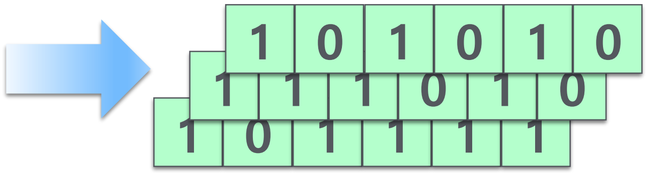
DEVIEW
2019





DECODER

ENCODER



write

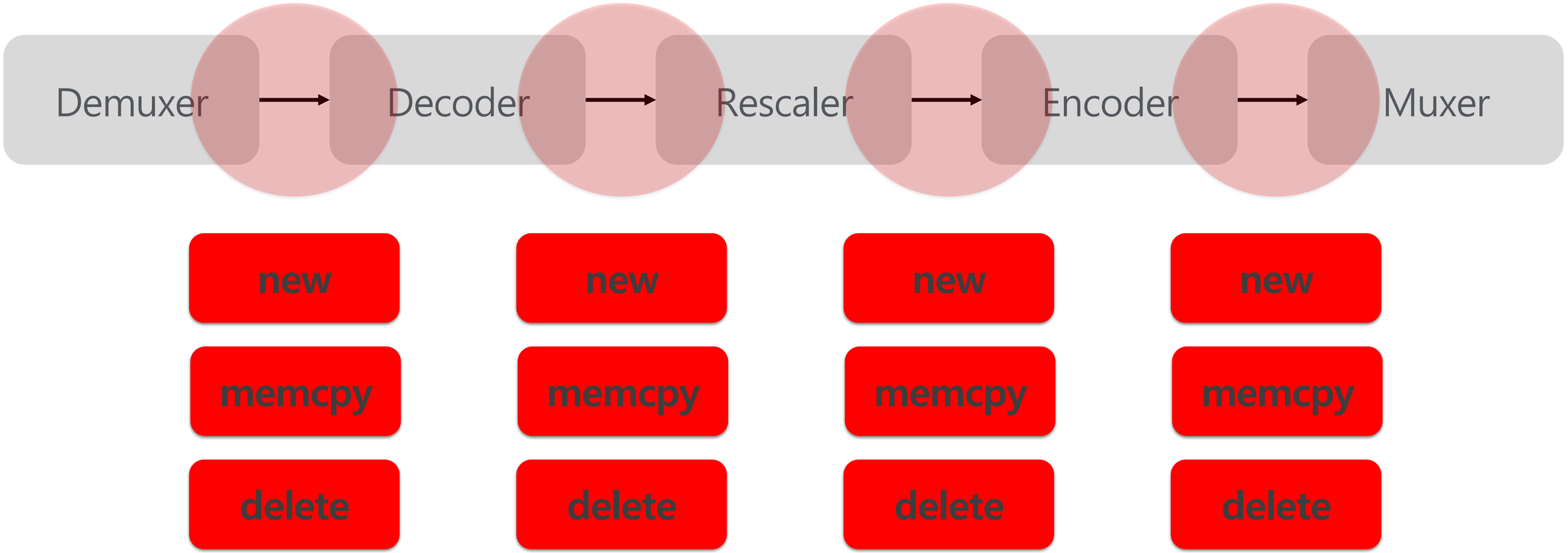
3.1 문제점 복기

- new / delete, read / write 등의 system call cost가 중복 발생
  => system call은 고비용!

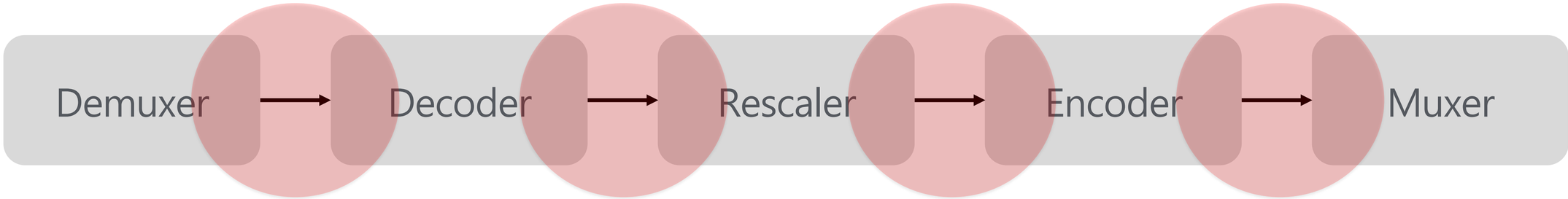
- 동일한 Memory를 2벌씩 유지하는 구간 발생. 불필요한 Memory Usage 증가
  => 장비를 비효율적으로 사용.

- 요즘 같이 H/W 스펙 좋은 시대에는 큰 문제가 없지 않나요??

3.1 구간별 메모리 할당 및 복사 발생



3.1 예) 720p 1분짜리 영상을 480p로 변환



총 **7200회** 할당/해제/복사

총 메모리 사용량 3.6 GB

실제로는 한 번에 훨씬 더 많은 해상도를 동시에 처리하고
더 많은 시간의 분량을 처리합니다.

3.1 Memory를 컴포넌트끼리 Share 하면?

DEVIEW 2019

장점

데이터 단위 당

Read/Write operation

각 1회의 생략 가능

단점

Copy 기반 접근 대비 높은 구현 복잡도

- 생산 모듈과 소비 모듈 간
  동기화 이슈
- 메모리 크기가 가변적인 경우
  반복적인 공유 메모리 재할당의
  Overhead 발생

3.1 Zero-Copy & Buffer-Pool 활용!

DEVIEW 2019

장점

데이터 단위 당

Read/Write operation

각 1회의 생략 가능

&

메모리 **재상용성** 확보

극복

메모리 소유권이 명확한

Buffer-pool로 해결

Video data는 메모리 특성상

고정된 메모리 구조를 가짐

단점

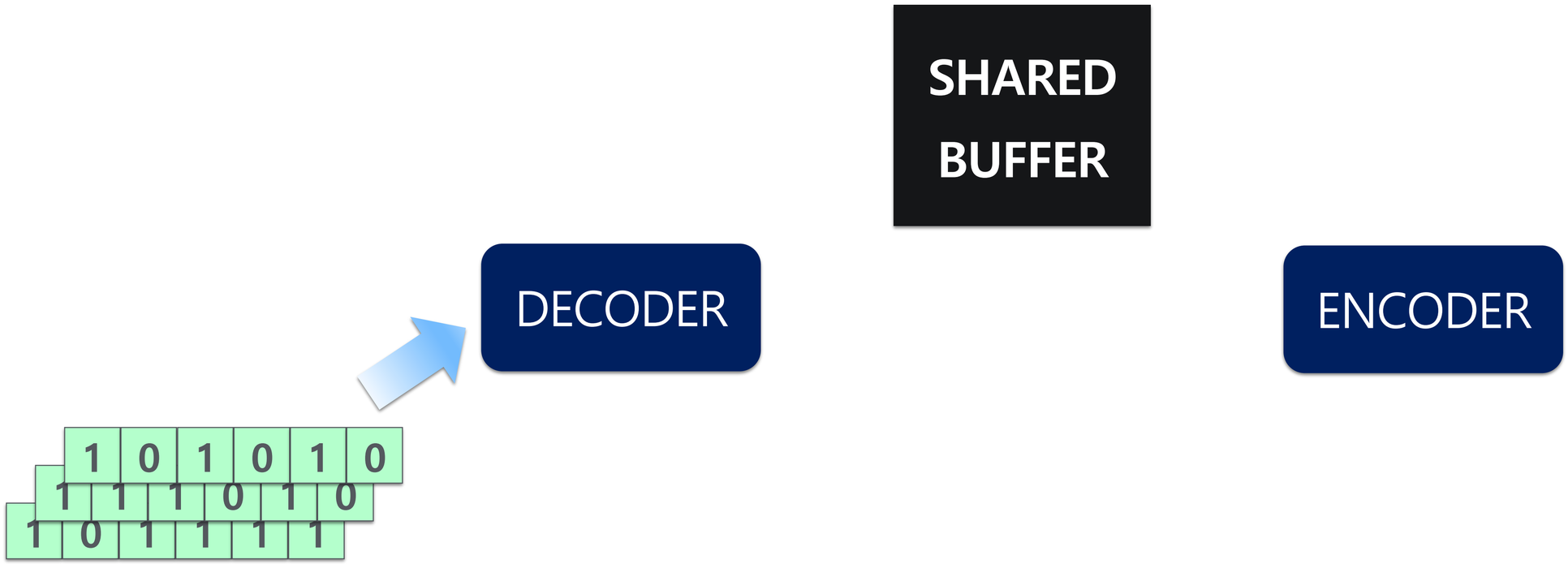
Overhead 발생

3.1 해결1 : Zero-Copy

- 동일한 메모리를 사용하는 모듈끼리 Memory를 Share하는 전략
  - 두 모듈에서 필요로 하는 **메모리 버퍼를 1벌로 줄임**
  - 메모리 new / delete, read / write **operation 1회 감소**

- 미디어 처리과정에서 zero-copy의 중요성
  - **압축되지 않은** 동영상 데이터의 **높은 메모리 사용량**
  - 예) 4K@60fps = 약 710 Mbytes/s

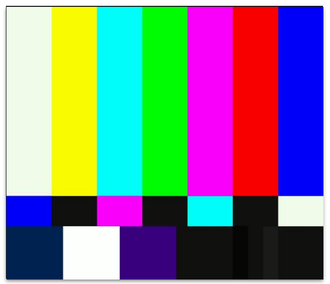
3.1 해결1 : Zero-Copy



3.1 해결1 : Zero-Copy

Write

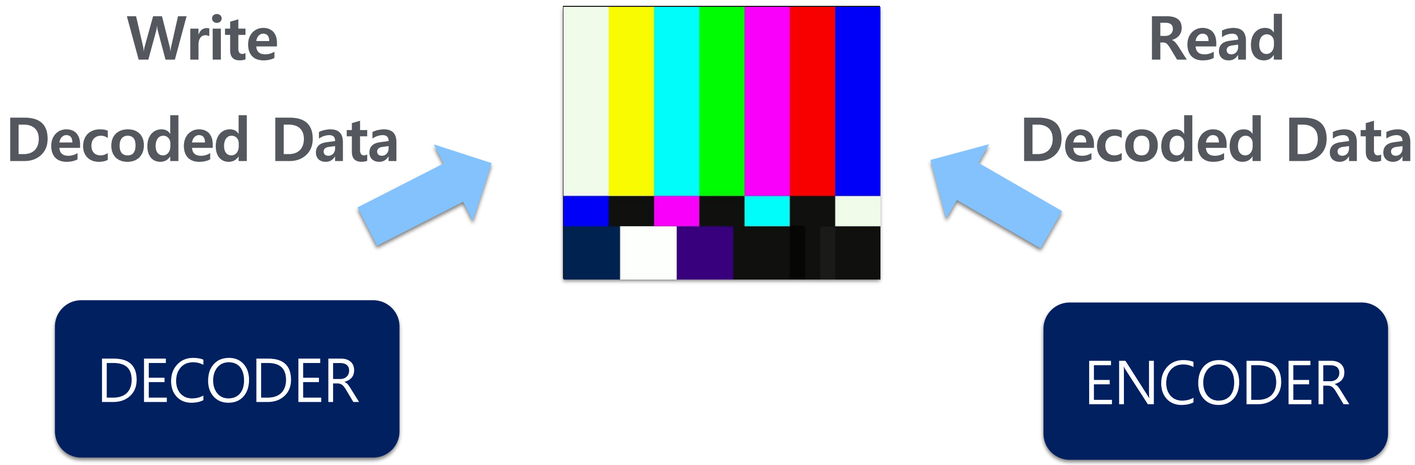
Decoded Data



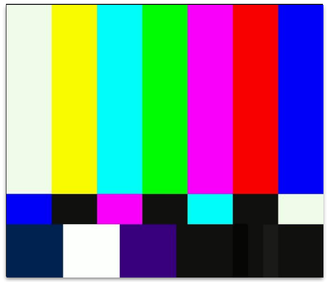
DECODER

ENCODER

3.1 해결1 : Zero-Copy

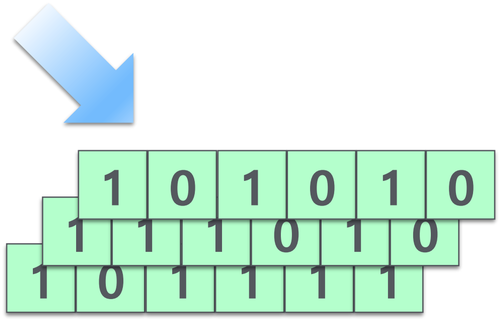


3.1 해결1 : Zero-Copy



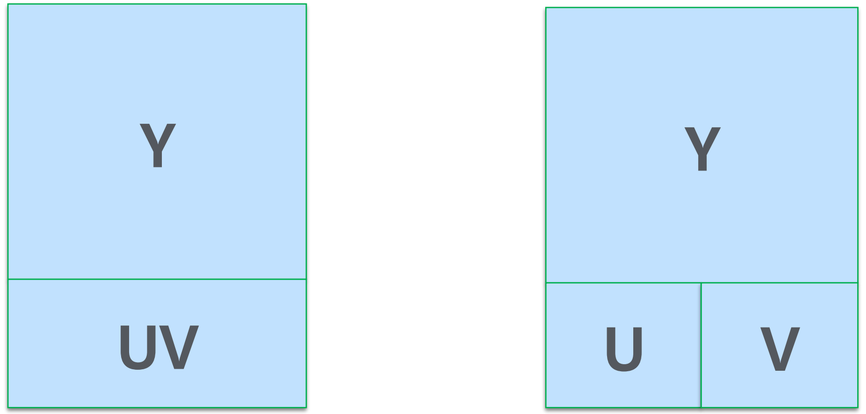
DECODER

ENCODER

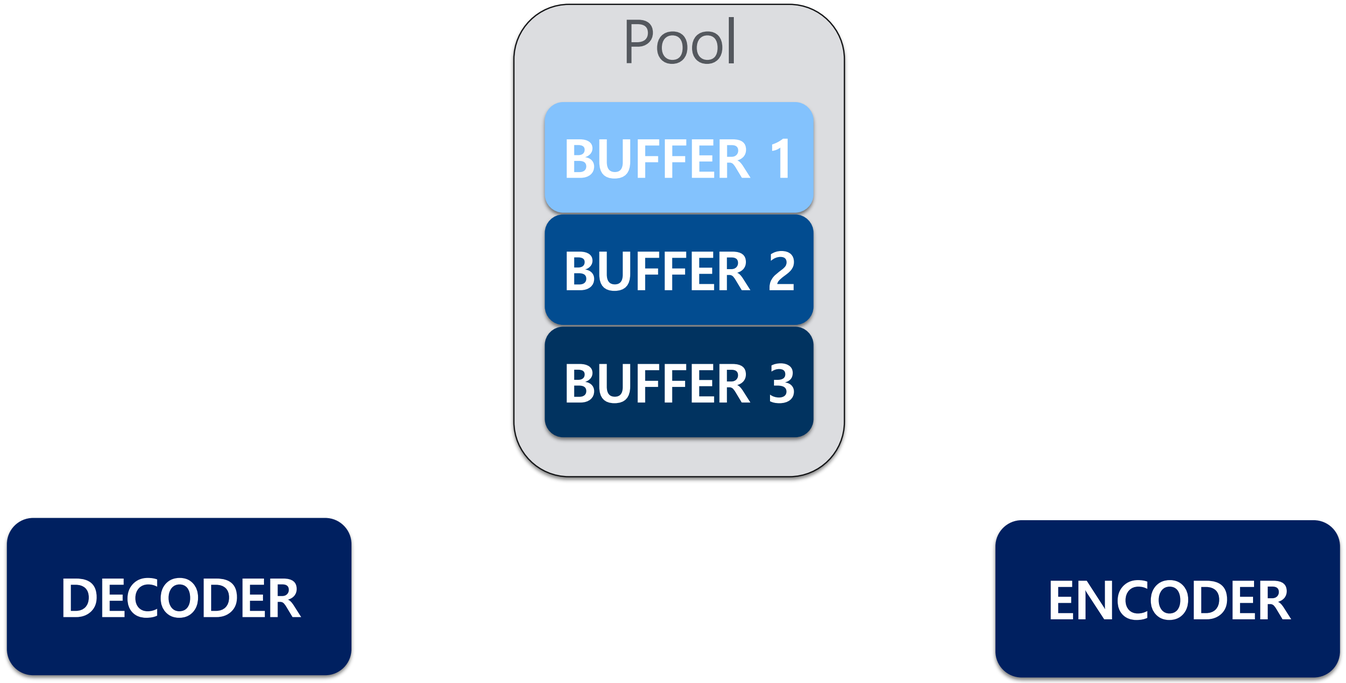


3.1 해결2 : Buffer-Pool

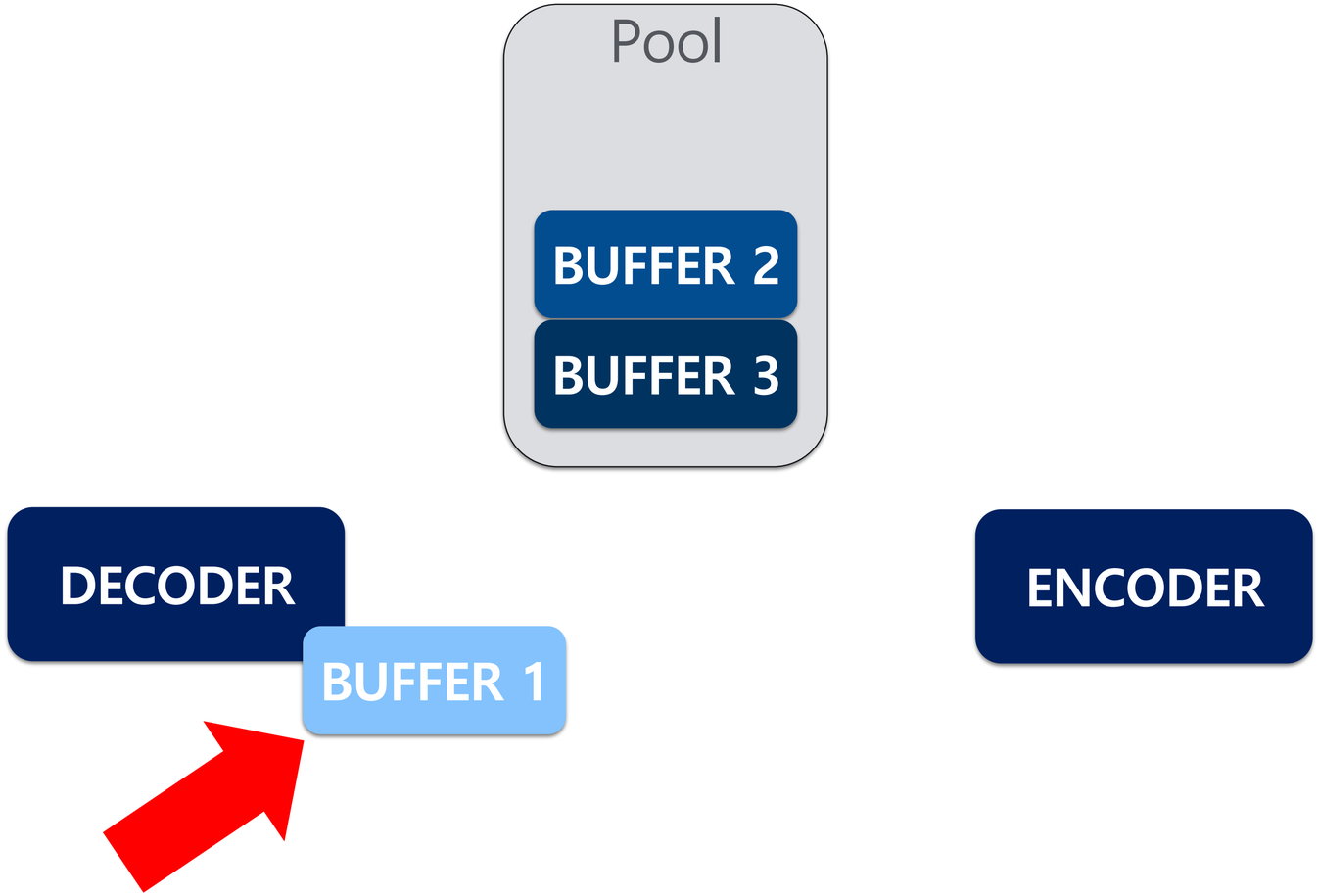
- 사용할 메모리를 일정량 미리 할당 후 순환하여 재사용하는 기법
  - **고정된 메모리 layout/size**를 가지는 미디어 데이터 구조에 적합
  - 반복적인 메모리 할당/해제 회피



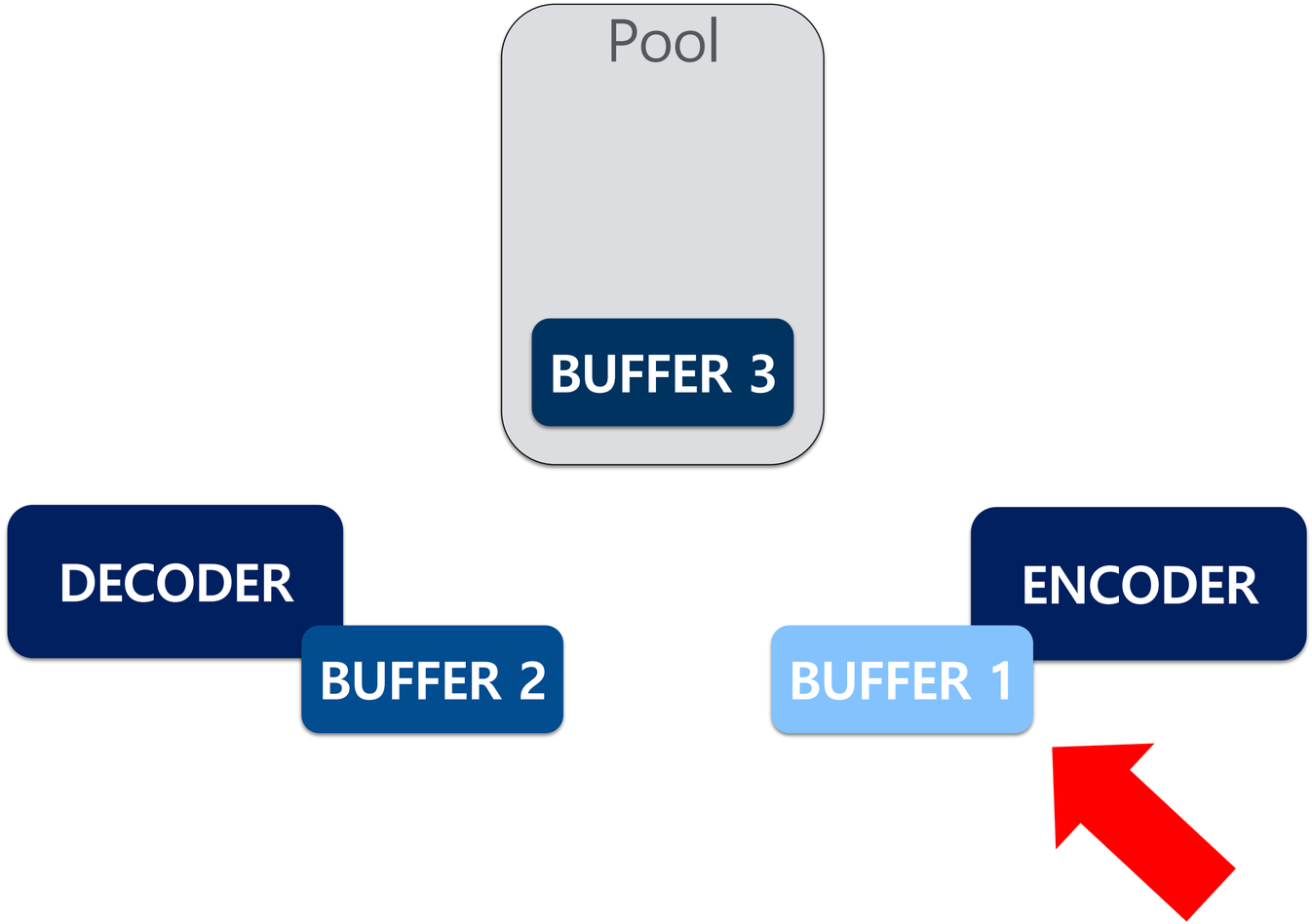
3.1 해결2 : Buffer-Pool



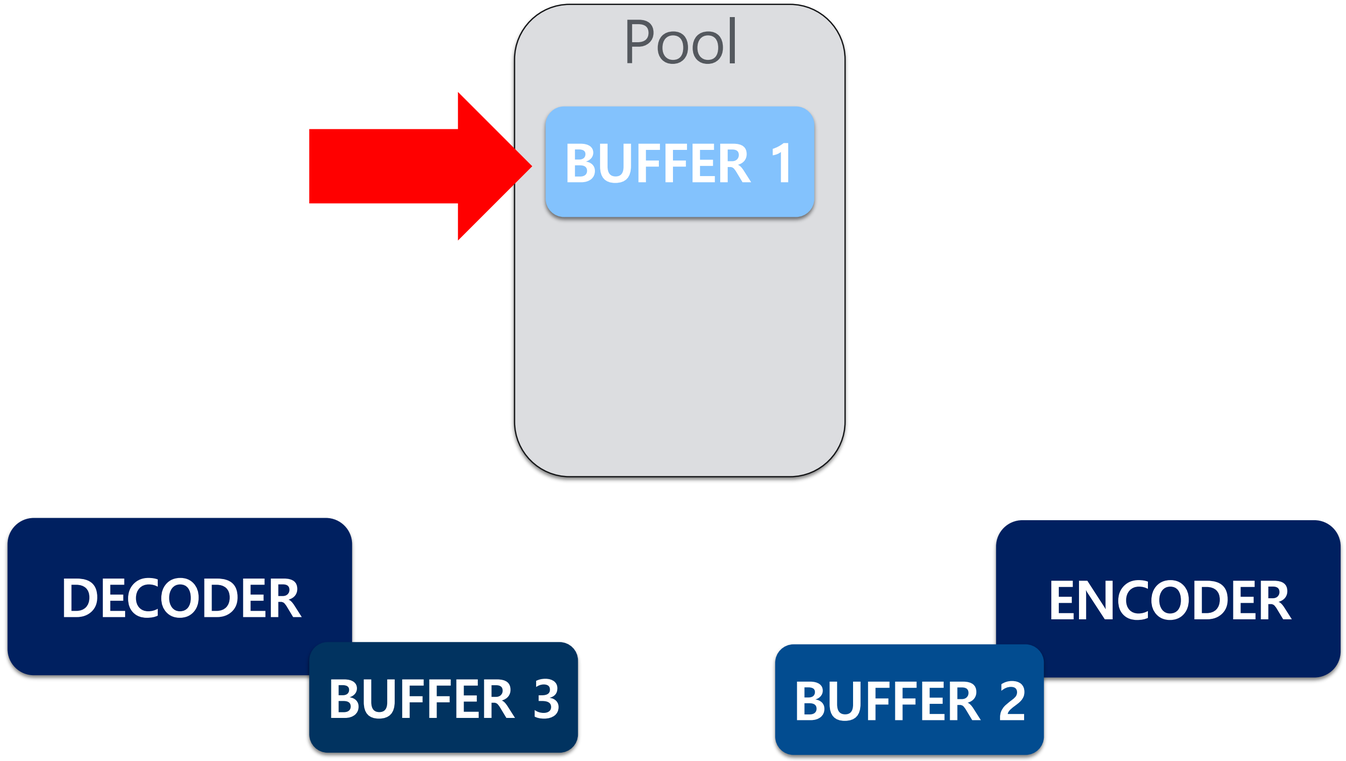
3.1 해결2 : Buffer-Pool



3.1 해결2 : Buffer-Pool



3.1 해결2 : Buffer-Pool



3.1 Zero-Copy & Buffer-Pool 개선 효과

- 반복적인 new / delete 와 read / write 감소
  → 기존 대비 CPU usage 약 10% 감소

- 모듈별 video memory 공유 (불필요한 메모리 할당 회피)
  → Memory usage 약 30% 감소

3.1 S/W를 넘어 H/W Customizing으로

- 신규 세대 장비 도입으로 성능 향상 기대했으나
- 이전 세대 장비보다 성능이 떨어지는 현상 발견
- Why??

3.1 Debugging with Intel Korea

DEVIEW
2019

3.1 AVX?? Advanced Vector Extensions

- SIMD 명령어 세트로, SSE 시리즈의 후속작
  - vector 연산을 가속화하는 기능. 부동소수점 연산 처리 능력이 향상됨.

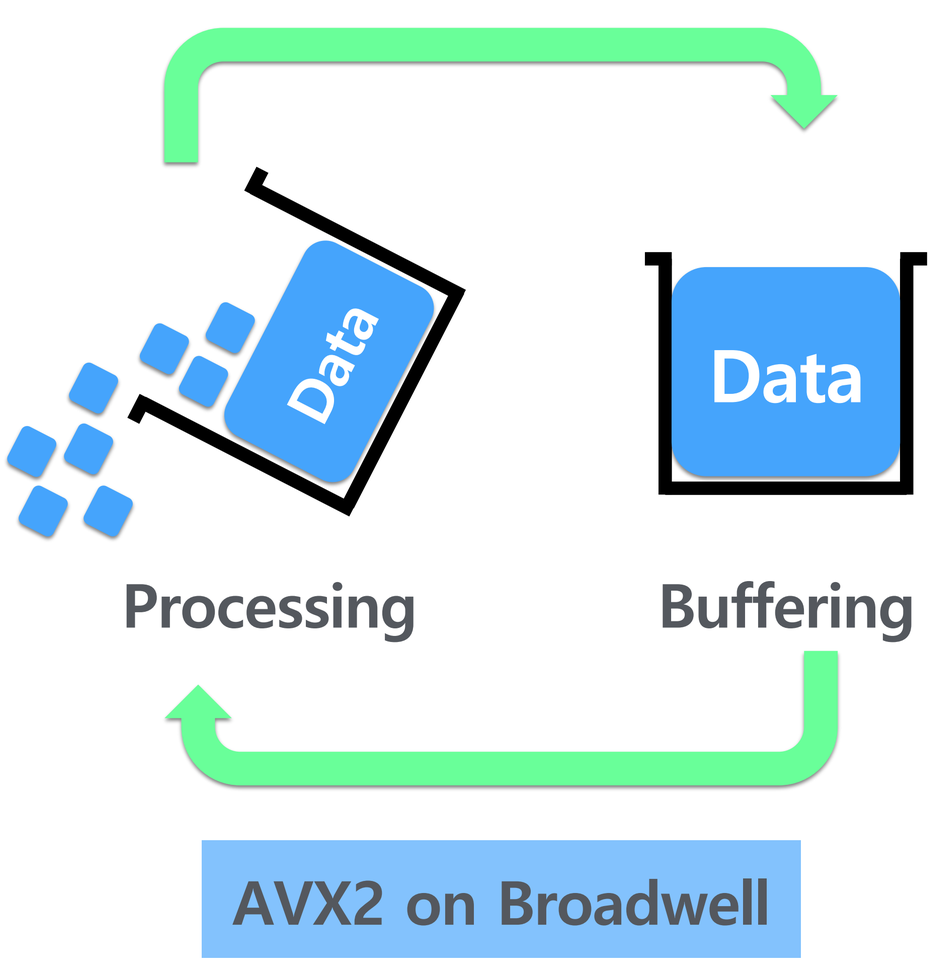
- 세대를 거듭할 수록, Register 크기 확대
  - SSE : 128-bit SIMD
  - AVX2 : 256-bit SIMD
  - AVX512 : 512-bit SIMD

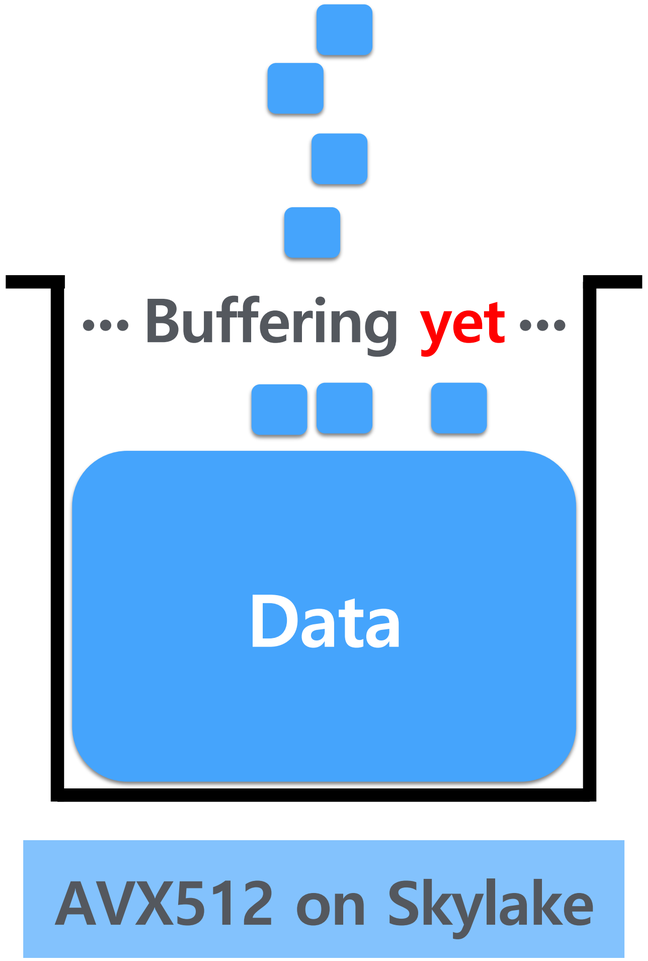
3.1 문제를 다시 짚어보면

- 신규 세대 장비
  - AVX512, AVX2 지원
- 이전 세대 장비
  - AVX2만 지원

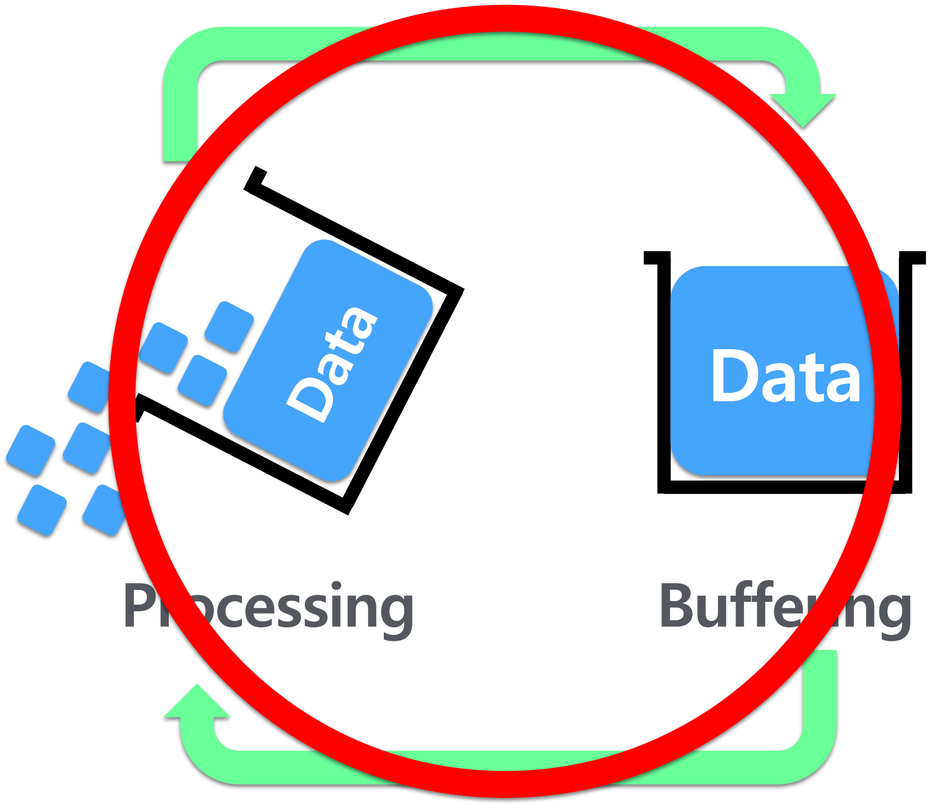
- 레지스터 크기가 크면, 성능이 더 좋아지는게 아닌가요?

3.1 CPU 동작의 2가지 모드

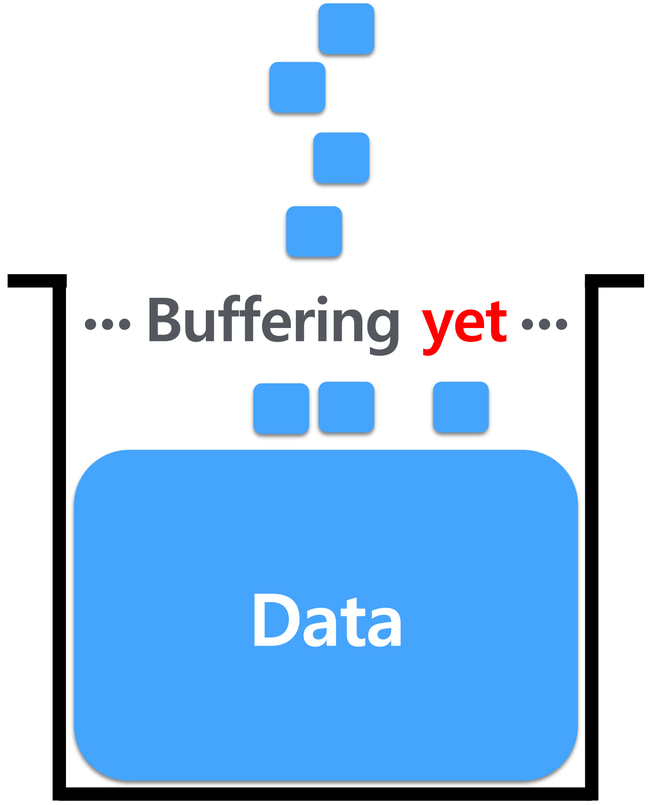




3.1 실시간 처리에 적합한 모델은



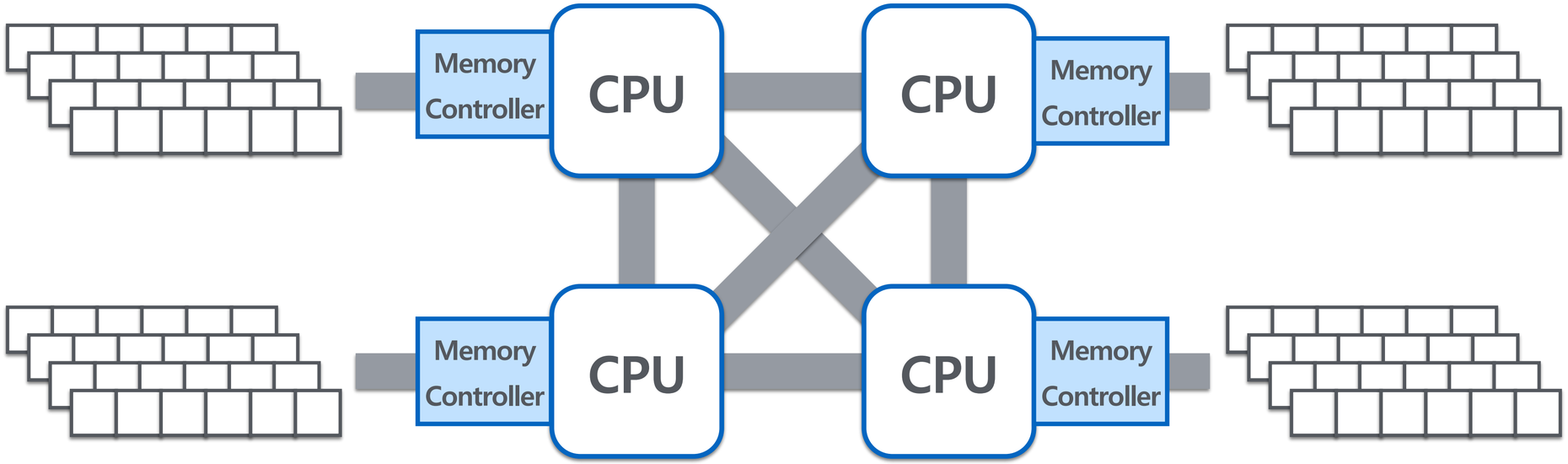
AVX2 on Broadwell



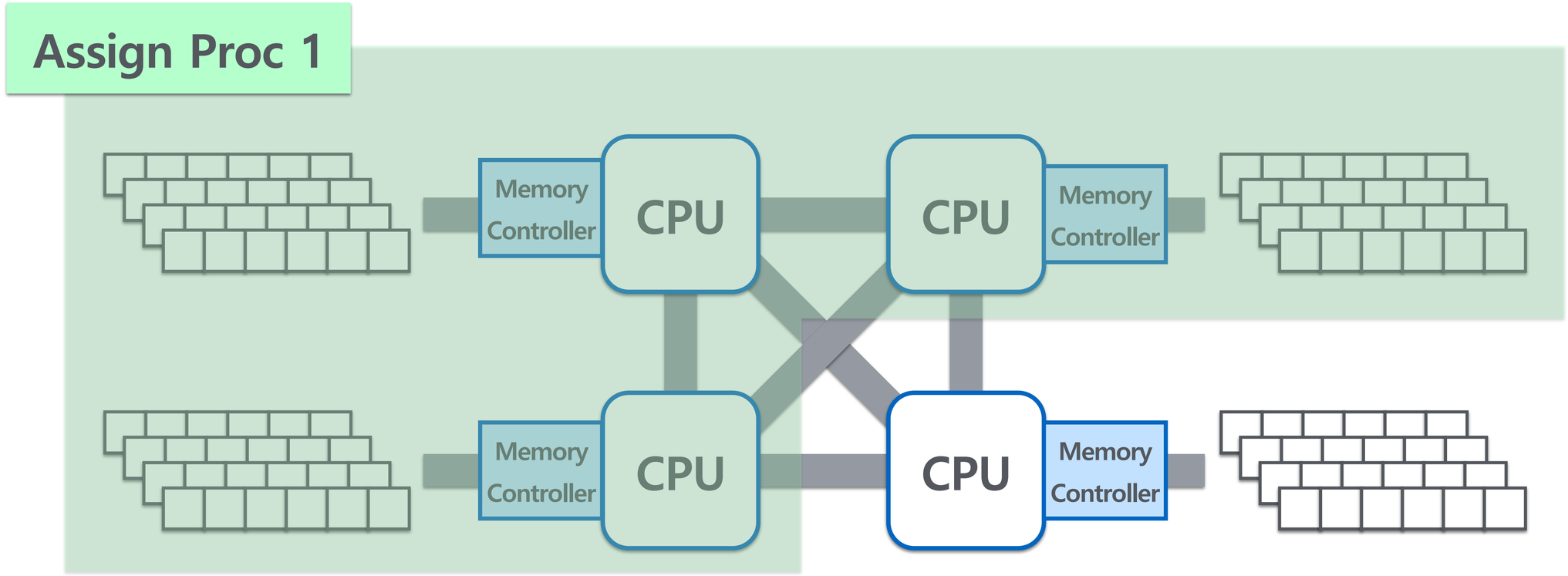
AVX512 on Skylake

3.1 NUMA?

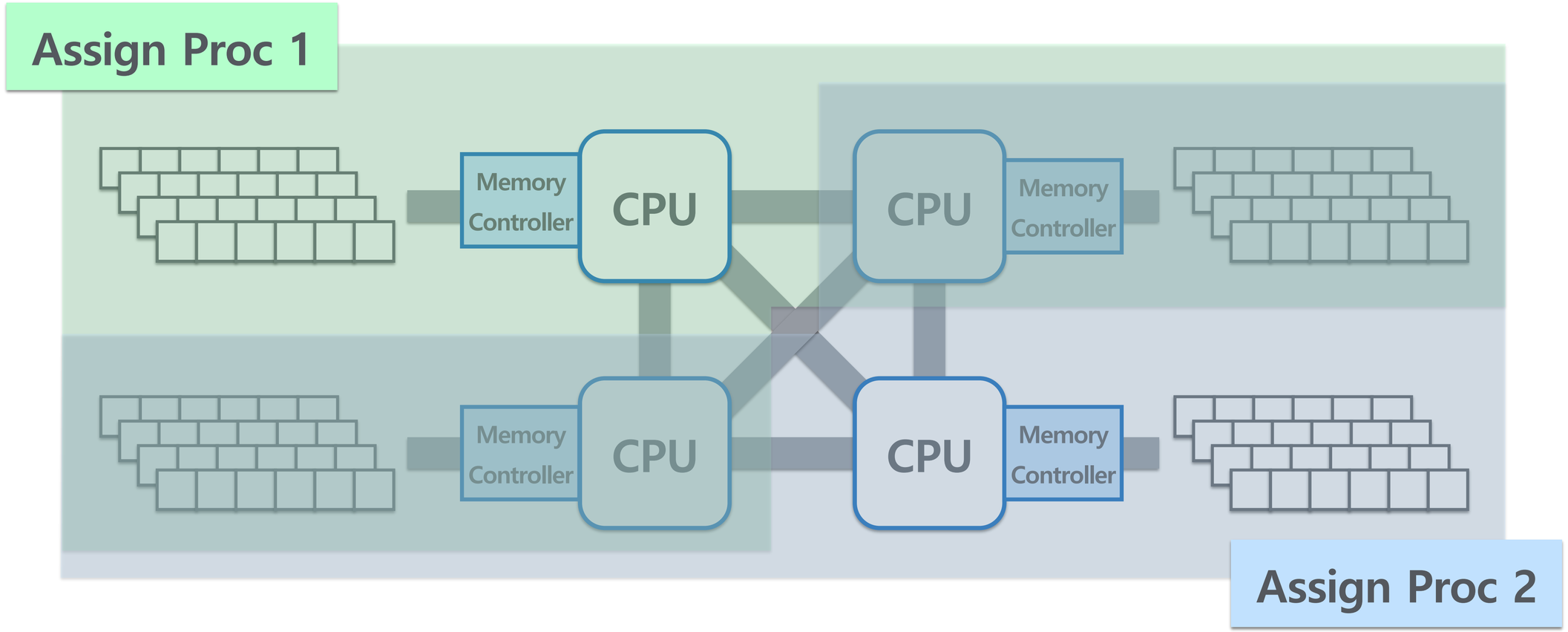
- Non-Uniform Memory Access (NUMA)



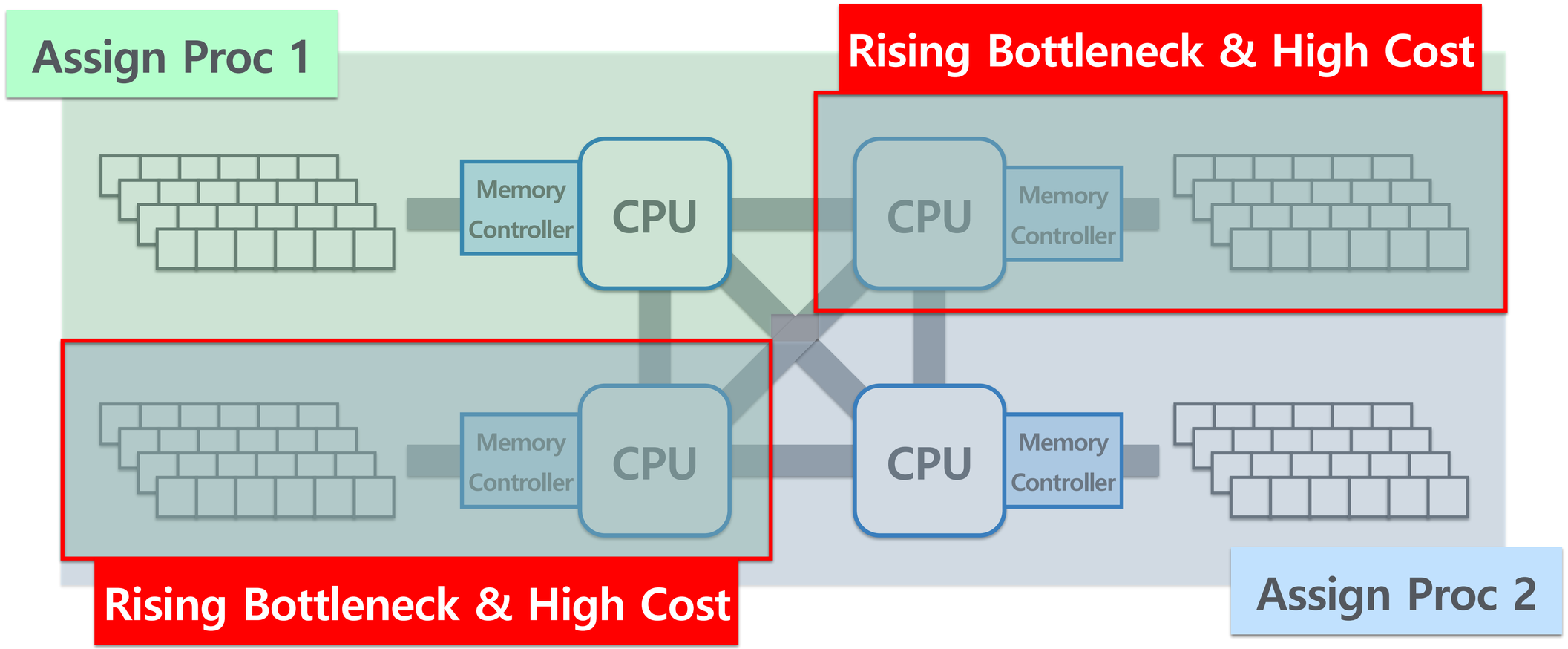
3.1 NUMA?


Assign Proc 1
Memory Controller
CPU
CPU
Memory Controller
Memory Controller
CPU
CPU
Memory Controller

3.1 NUMA?


Assign Proc 1
Memory Controller
CPU
CPU
Memory Controller
Memory Controller
CPU
CPU
Memory Controller
Assign Proc 2

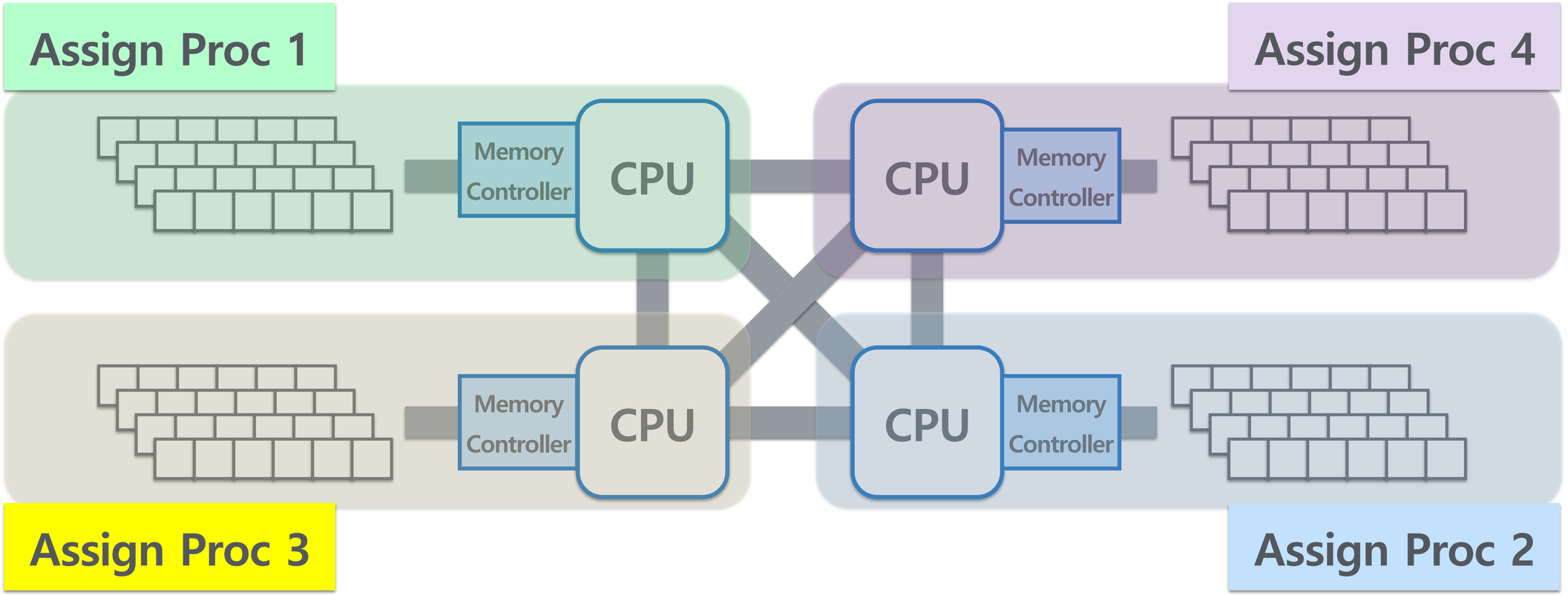
3.1 NUMA?


Assign Proc 1
Rising Bottleneck & High Cost
Memory Controller
CPU
CPU
Memory Controller
Memory Controller
CPU
CPU
Memory Controller
Rising Bottleneck & High Cost
Assign Proc 2

3.1 NUMA : 최적화

DEVIEW 2019

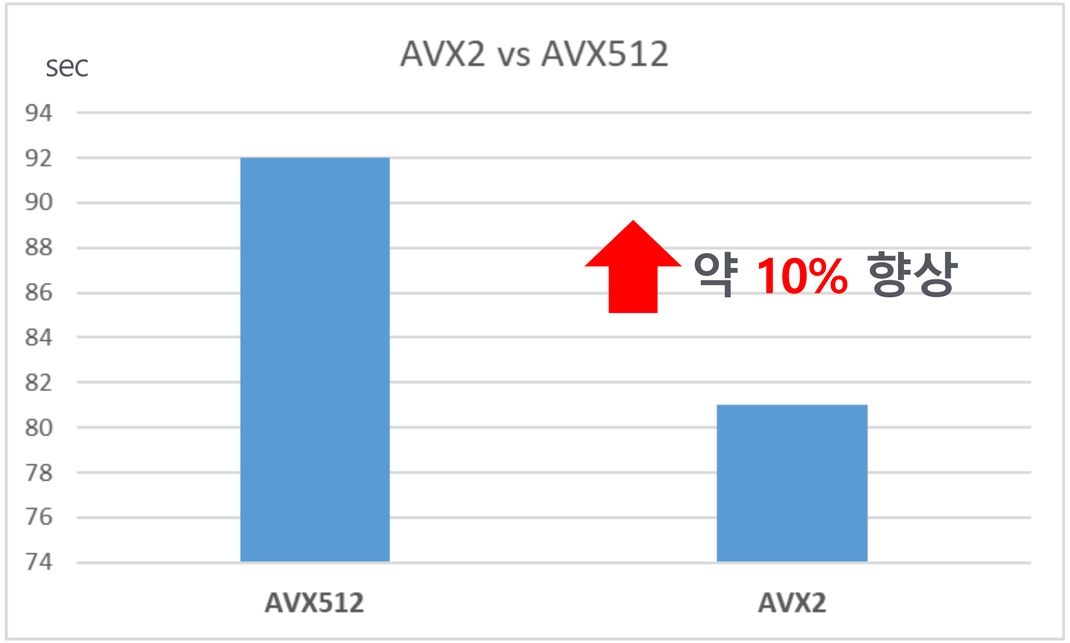
✓ Process 실행 시 CPU 지정 할당

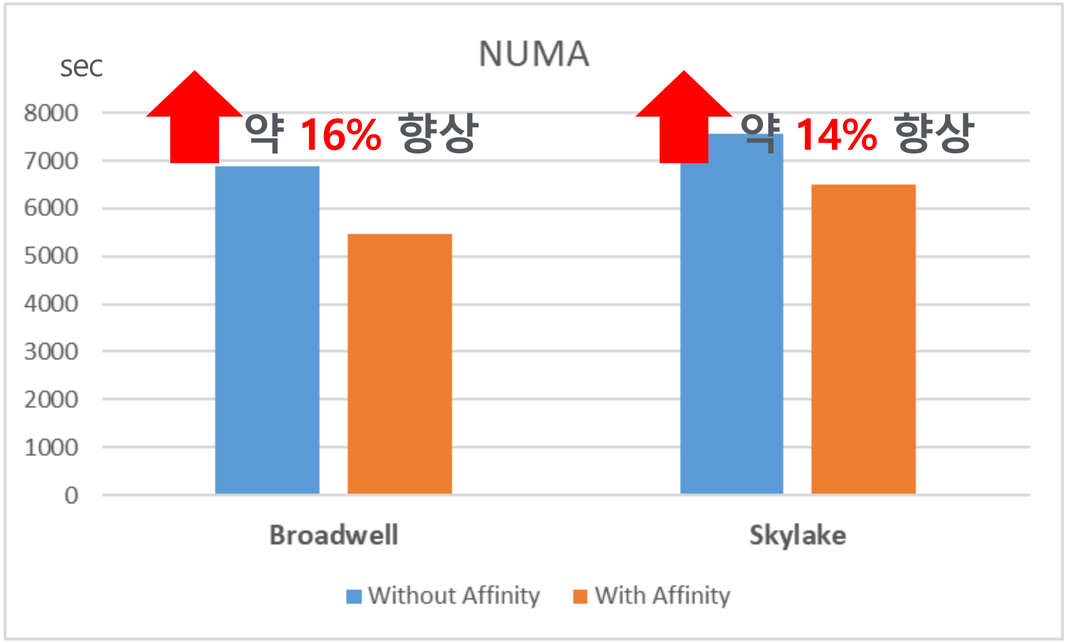


3.1 H/W Customizing 결과

DEVIEW 2019

✓ 향상된 성능 (Lower is Better)





DEVIEW
2019

3-2. 초저지연 & 초고화질 서비스를 달성한 기술

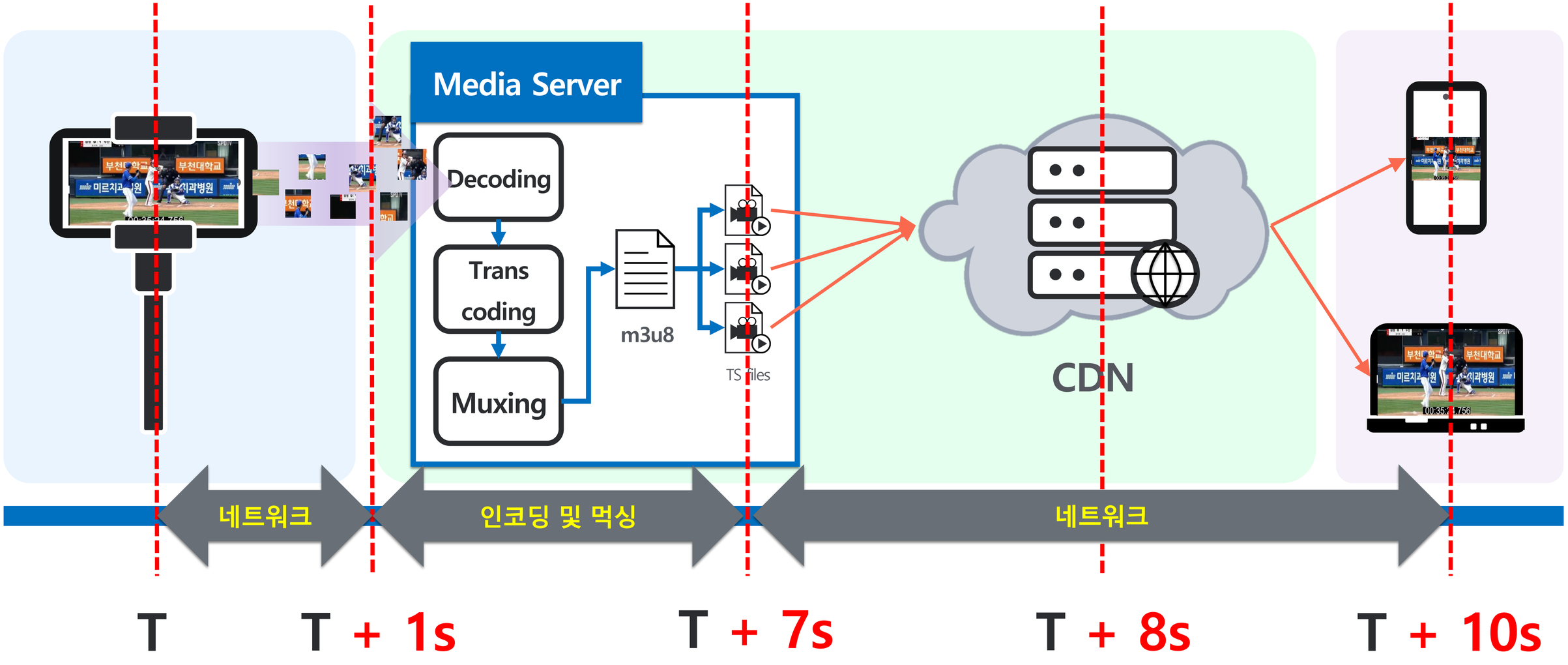
3.2 고품질의 중요 요소 2가지

- 미디어 스트리밍 서버 == 동영상을 처리하고 전송하는 서버
- 전송은 **빠를수록** 좋다.
- 동영상은 **고화질일수록** 좋다.

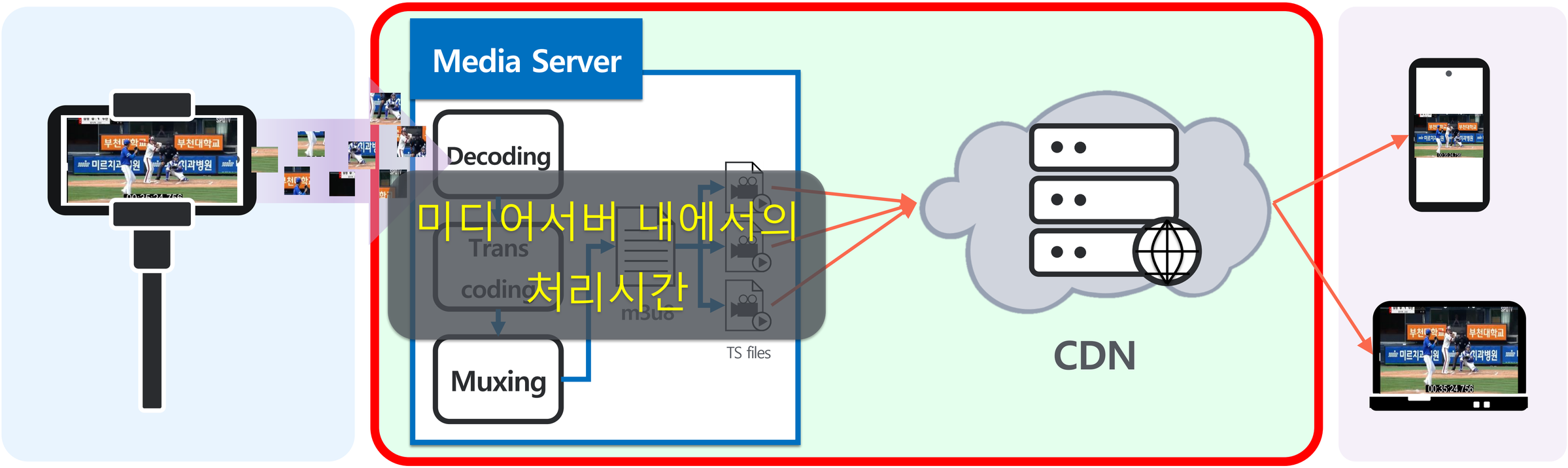
3.2 Latency가 크면 클수록

- 네이버로 손흥민 경기 시청 중 옆집에서 먼저 함성을 질러 스포를 당할 수 있습니다.
- 우리 BTS 오빠들과 댓글 소통이 실시간으로 이뤄지기 어렵습니다.

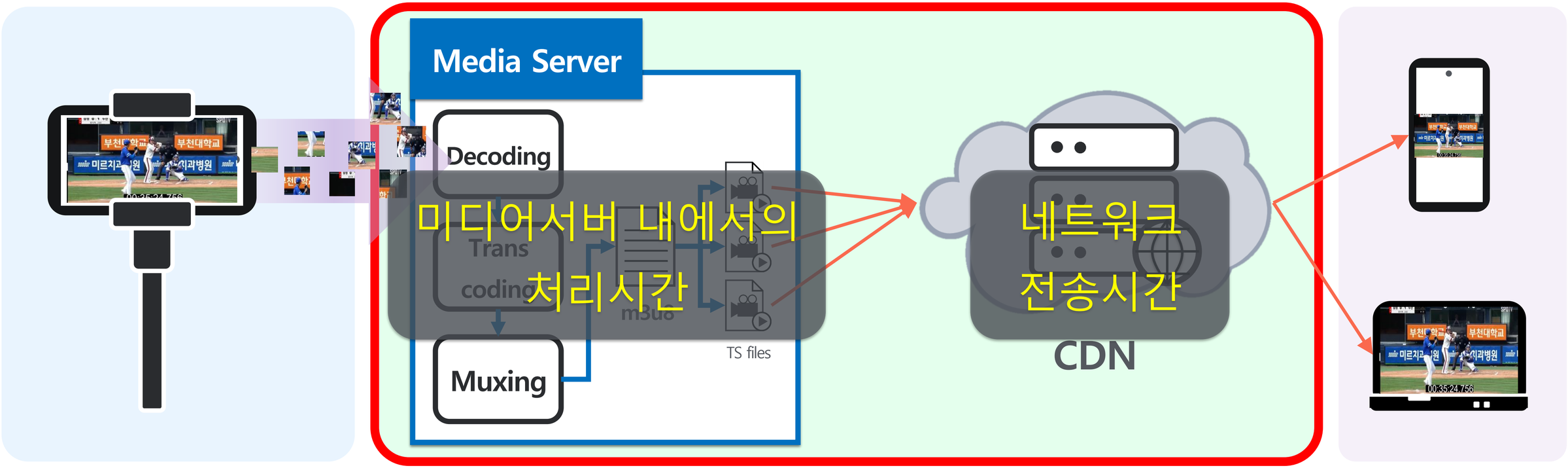
3.2 기존의 Latency 발생 시간



3.2 Latency를 줄일 수 있는 구간 - 1



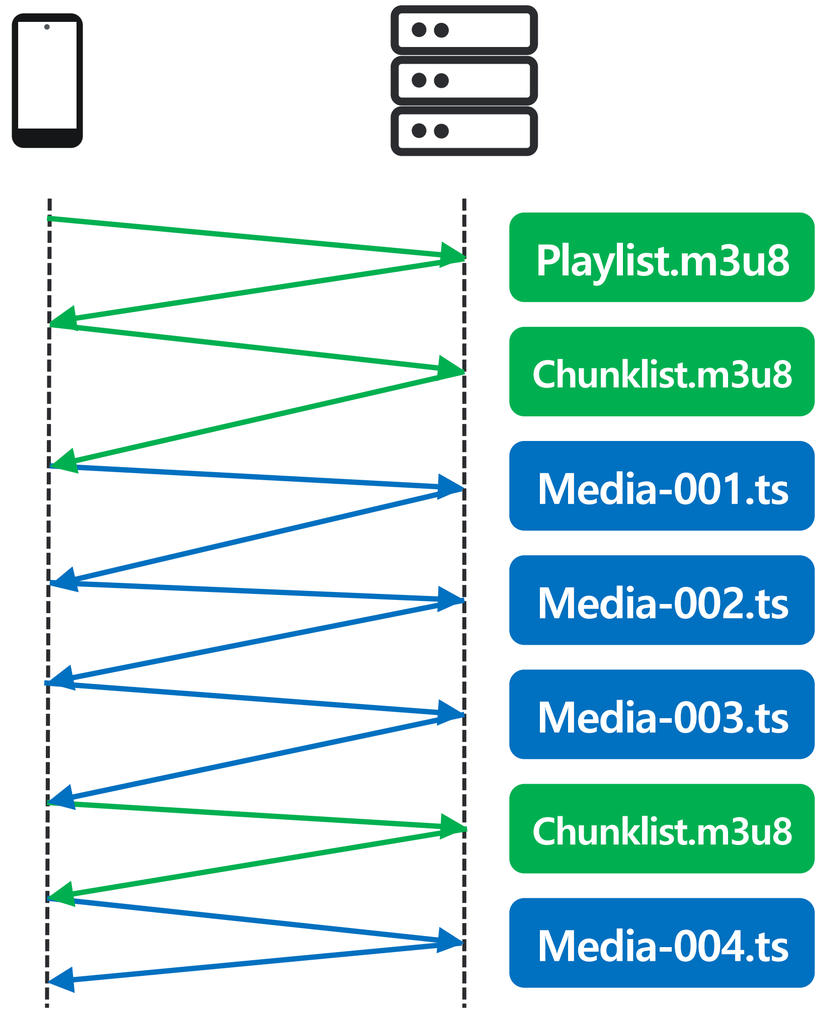
3.2 Latency를 줄일 수 있는 구간 - 2



3.2 기존 프로토콜의 동작 방식(1/3)

DEVIEW 2019

HLS

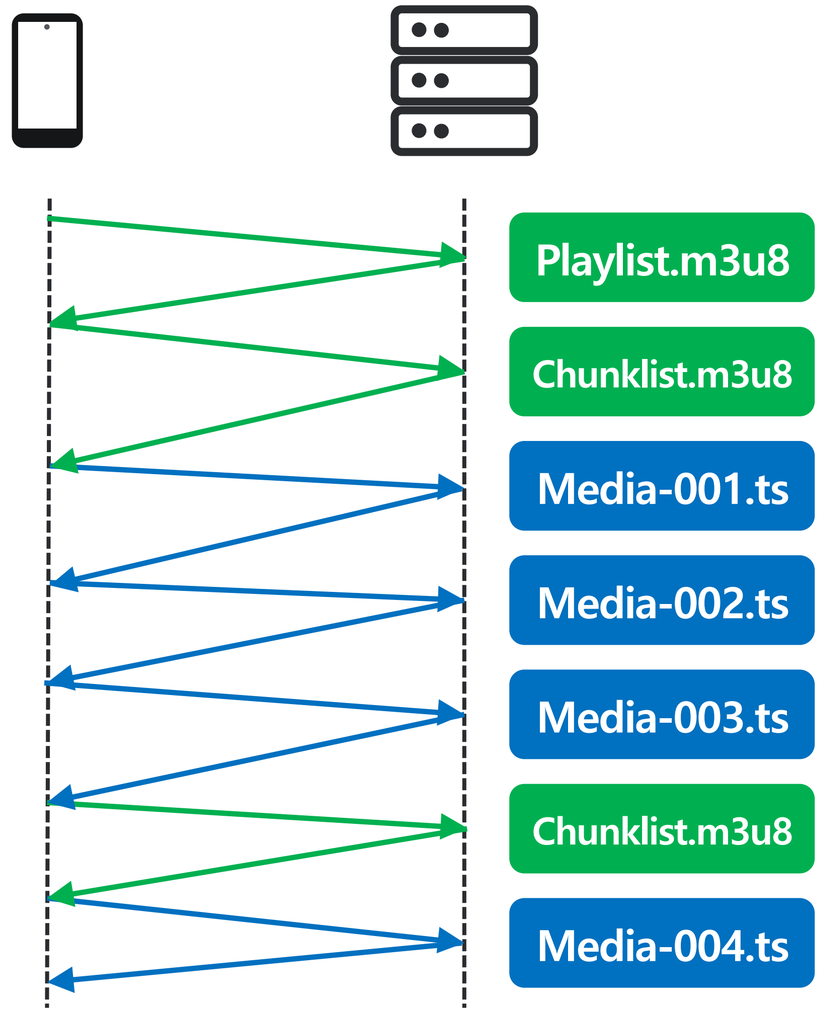


동영상 정보를 구성하는 파일 을 받고

3.2 기존 프로토콜의 동작 방식(2/3)

DEVIEW 2019

HLS



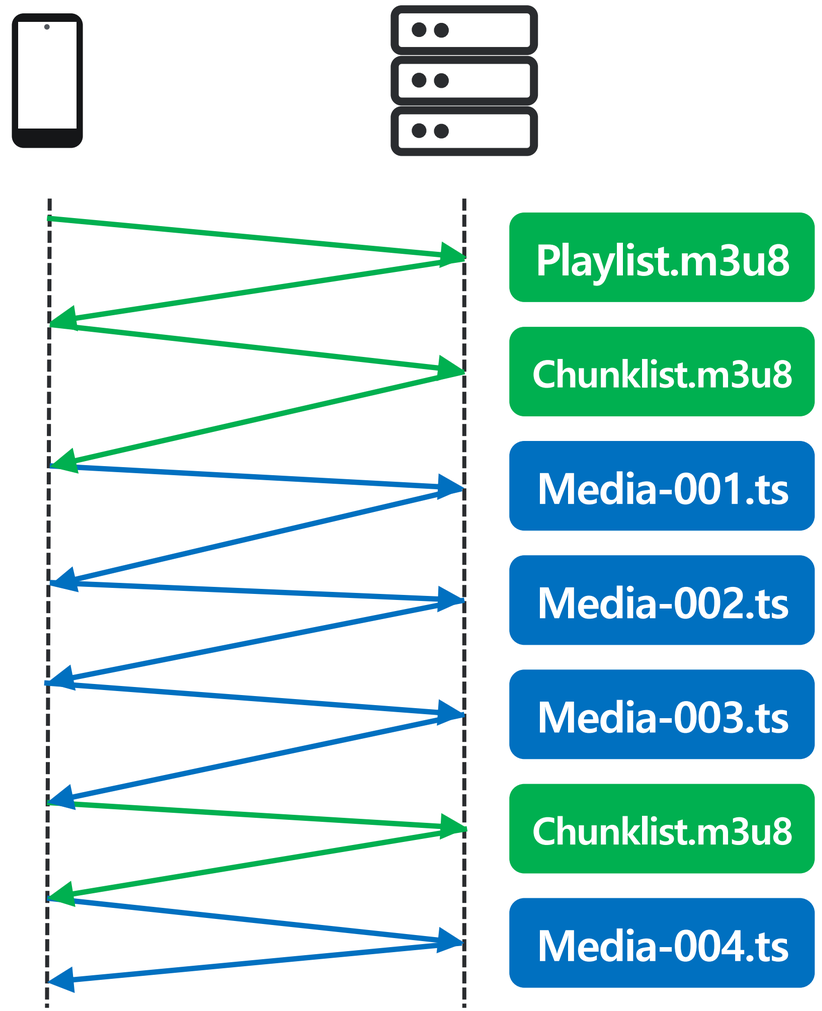
동영상 정보를 구성하는 파일 을 받고

실제 동영상이 담긴 미디어 파일 을 받아서

3.2 기존 프로토콜의 동작 방식(3/3)

DEVIEW 2019

HLS

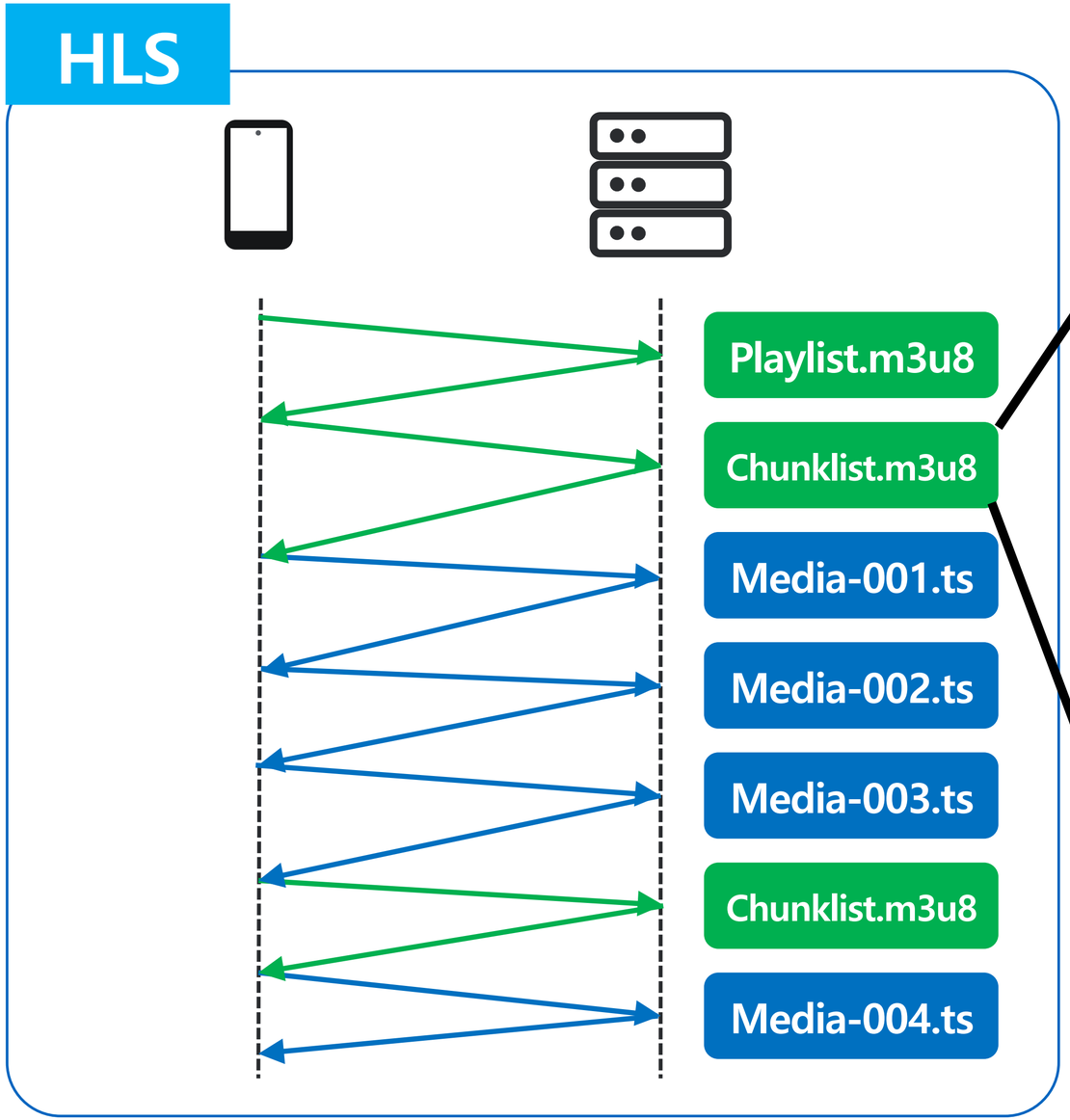


동영상 정보를 구성하는 파일 을 받고

실제 동영상이 담긴 미디어 파일 을 받아서

플레이어가 재생 시작

3.2 정보 파일에 담겨져 있는 내용은

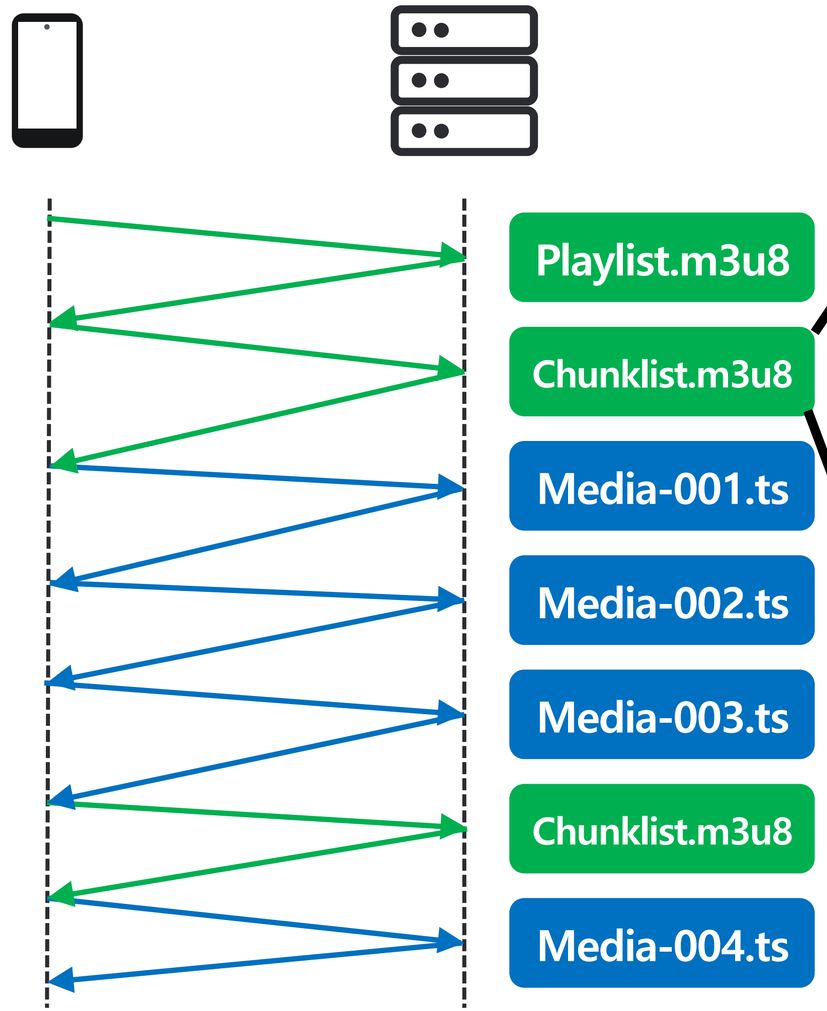



```
...
#EXT-X-PROGRAM-DATE-TIME:2019-06-04T01:12:10.241Z

#EXTINF:2.000000,
Media-001.ts
#EXTINF:2.000000,
Media-002.ts
#EXTINF:2.000000,
Media-003.ts
#EXTINF:2.000000,
Media-004.ts
#EXTINF:2.000000,
...
```

3.2 언제 방송이 시작됐는지

HLS

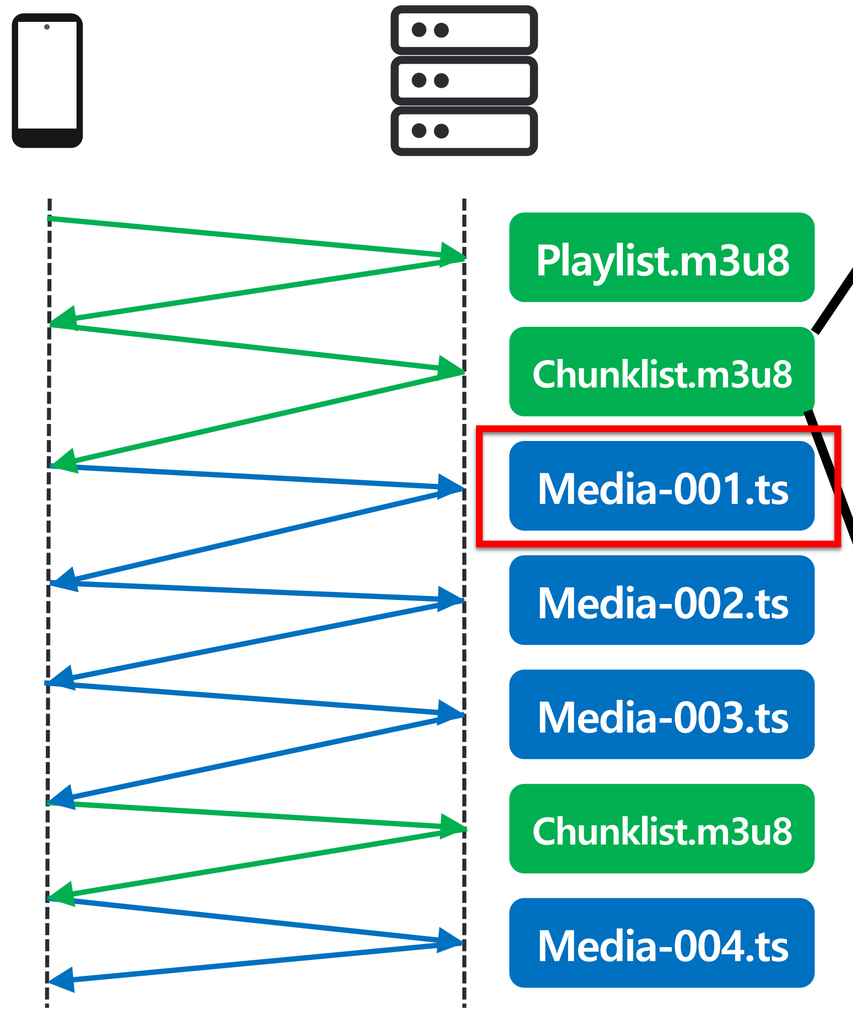



```
...
#EXT-X-PROGRAM-DATE-TIME:2019-06-04T01:12:10.241Z

#EXTINF:2.000000,
Media-001.ts
#EXTINF:2.000000,
Media-002.ts
#EXTINF:2.000000,
Media-003.ts
#EXTINF:2.000000,
Media-004.ts
#EXTINF:2.000000,
...
```

3.2 동영상 파일 하나의 길이는 얼마인지

HLS

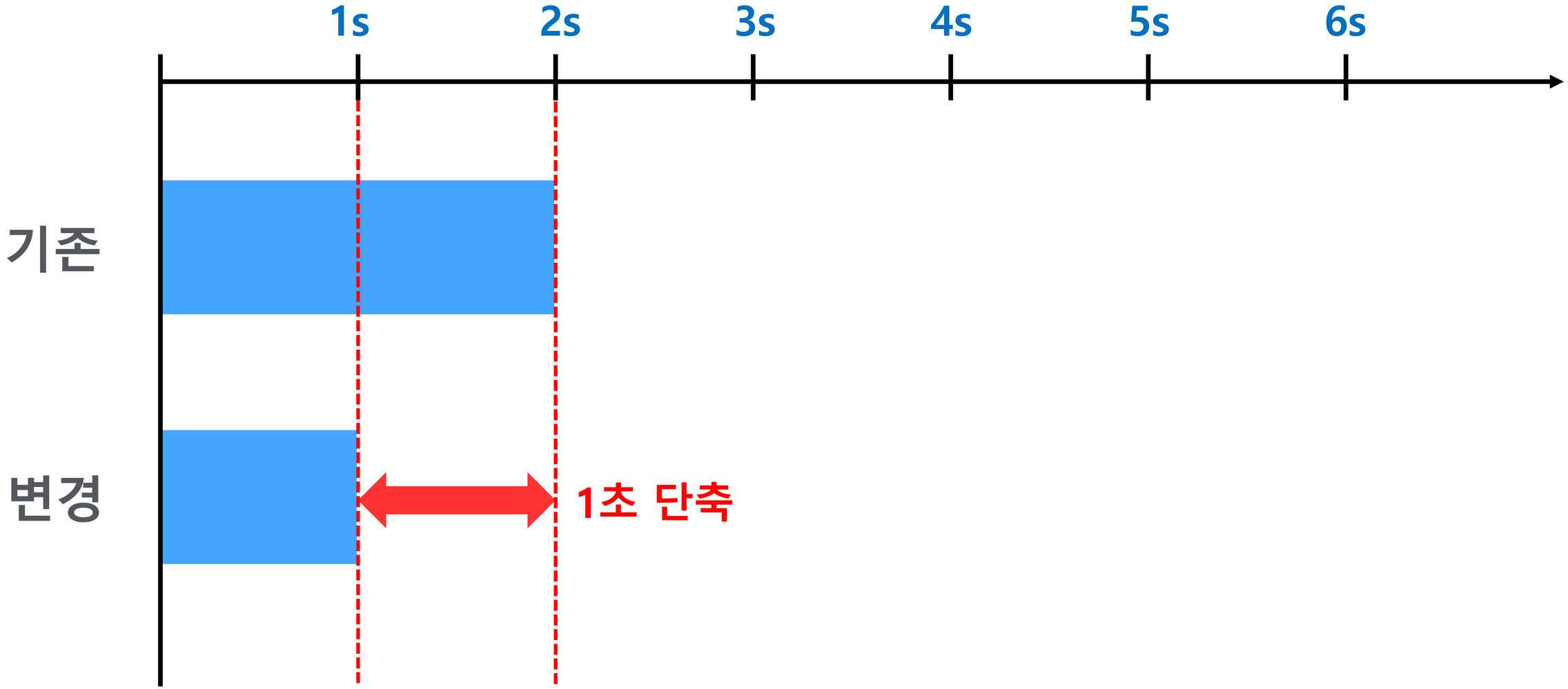



```
...
#EXT-X-PROGRAM-DATE-TIME:2019-06-04T01:12:10.241Z

#EXTINF:2.000000,
Media-001.ts
#EXTINF:2.000000,
Media-002.ts
#EXTINF:2.000000,
Media-003.ts
#EXTINF:2.000000,
Media-004.ts
#EXTINF:2.000000,
...
```

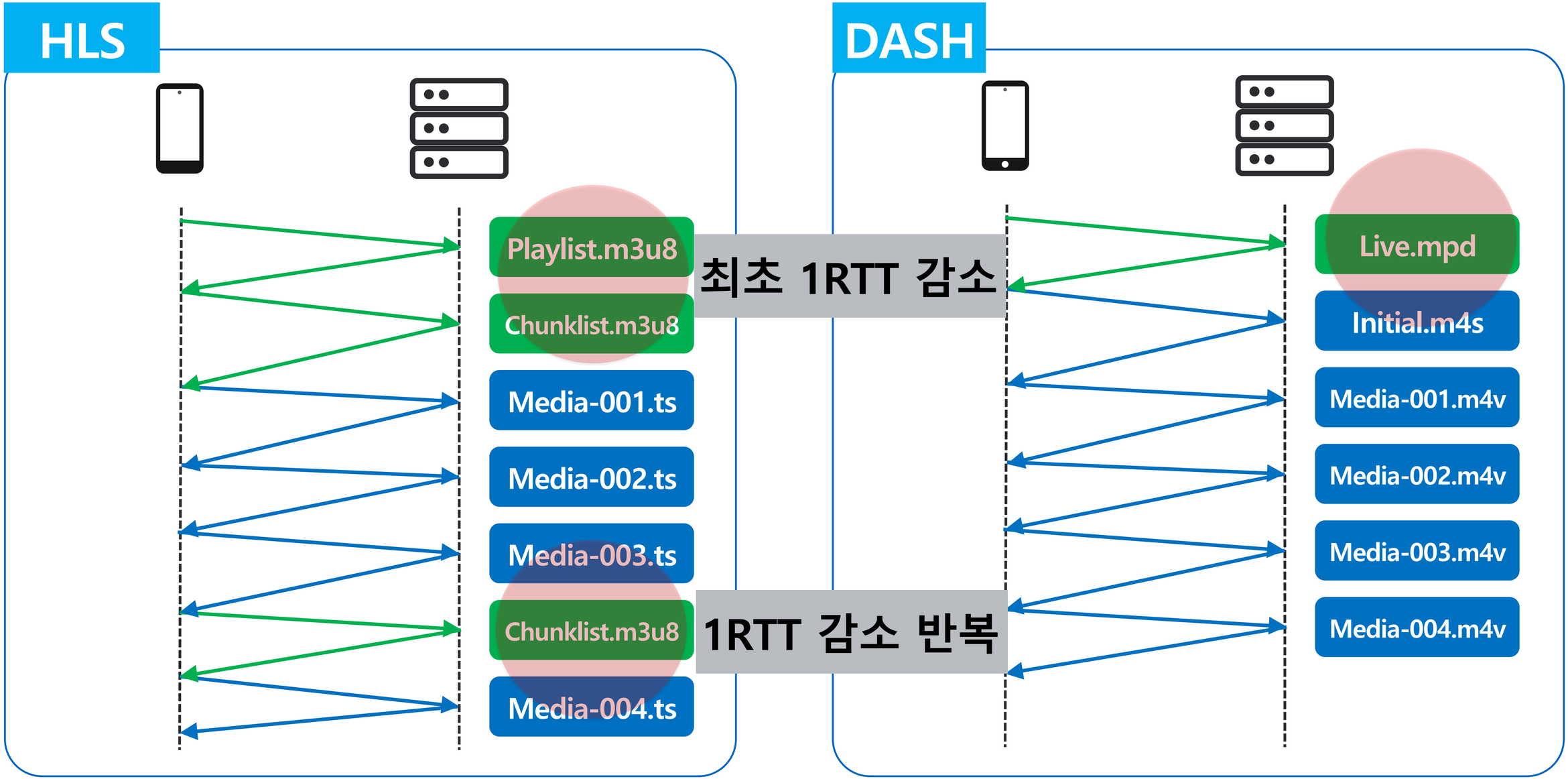
3.2 시도 #1 : 동영상 길이를 짧게 해보자

DEVIEW
2019



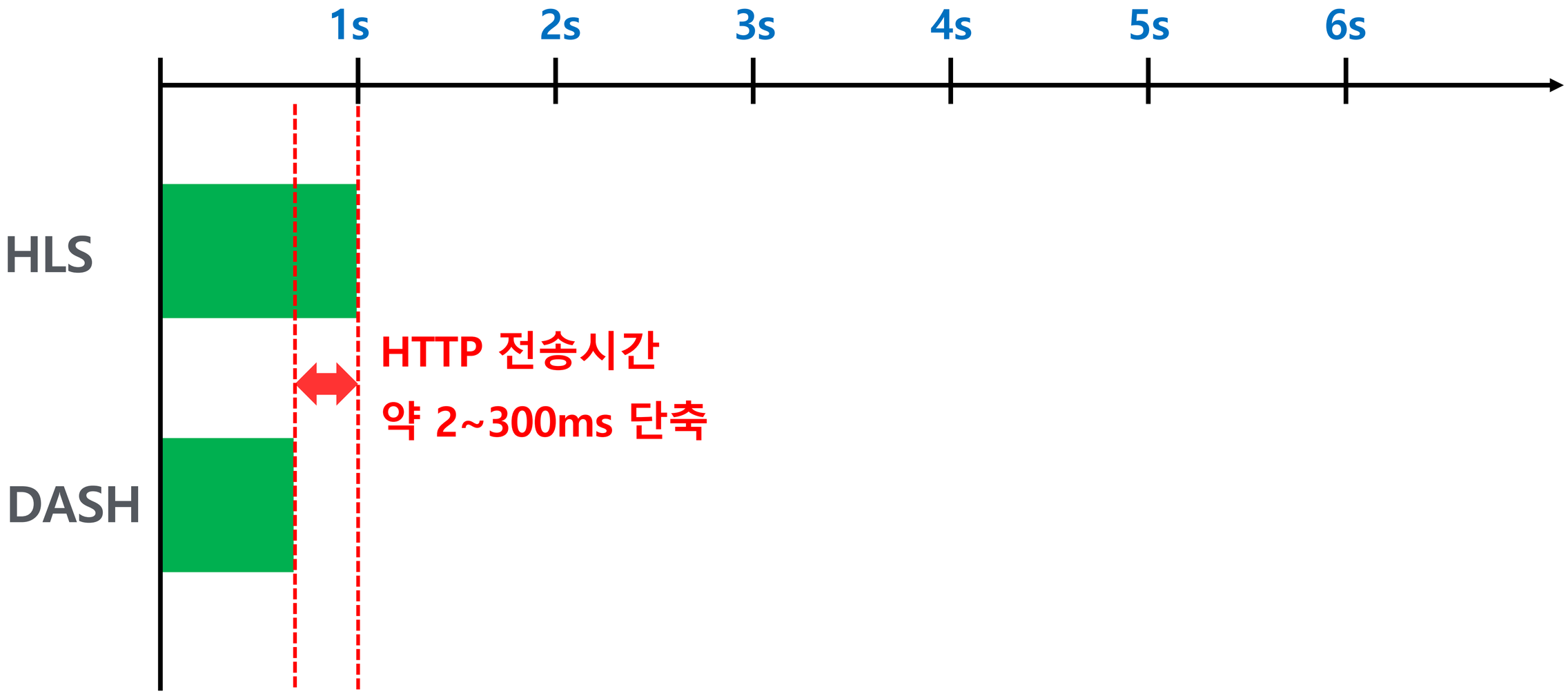
3.2 MPEG-DASH 프로토콜 도입 검토

DEVIEW 2019



3.2 시도 #2 : 신규 프로토콜로 RTT를 줄이자

DEVIEW 2019



3.2 DASH 프로토콜의 이점(1/4)

- 미디어 서버의 시간이 단서로 주어지고

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ...
    type="dynamic" profiles="urn:mpeg:dash:profile:isoff-live:2011“
    …
    availabilityStartTime="2019-10-07T06:40:51.932Z" publishTime="2019-10-07T06:41:02.932Z">
    <UTCTiming schemeIdUri="urn:mpeg:dash:utc:direct:2014" value="2019-10-07T06:41:02.932Z"/>
    <Period start="PT0.0S" id="1">
        …
        <SegmentTemplate timescale="44100" media="$RepresentationID$/$Number%09d$.m4a" … />
….
```

3.2 DASH 프로토콜의 이점(2/4)

DEVIEW 2019

- 서버와 클라이언트의 시간차를 역산 가능

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ...
    type="dynamic" profiles="urn:mpeg:dash:profile:isoff-live:2011“
    ...
    availabilityStartTime="2019-10-07T06:40:51.932Z" publishTime="2019-10-07T06:41:02.932Z">
    <UTCTiming schemeIdUri="urn:mpeg:dash:utc:direct:2014" value="2019-10-07T06:41:02.932Z"/>
    <Period start="PT0.0S" id="1">
        ...
        <SegmentTemplate timescale="44100" media="$RepresentationID$/$Number%09d$.m4a" ... />
....
```

3.2 DASH 프로토콜의 이점(3/4)

- Template 형태로 Media 파일 요청 가능

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ...
    type="dynamic" profiles="urn:mpeg:dash:profile:isoff-live:2011“
    ...
    availabilityStartTime="2019-10-07T06:40:51.932Z" publishTime="2019-10-07T06:41:02.932Z">
    <UTCTiming schemeIdUri="urn:mpeg:dash:utc:direct:2014" value="2019-10-07T06:41:02.932Z"/>
    <Period start="PT0.0S" id="1">
        ...
        <SegmentTemplate timescale="44100" media="$RepresentationID$/$Number%09d$.m4a" ... />
....
```

3.2 DASH 프로토콜의 이점(4/4)

- 시간차 값을 통해 최신 미디어 파일 Number 계산 가능
- 즉, 서버-클라이언트 협약으로 늘 Latency를 최소로 유지 가능

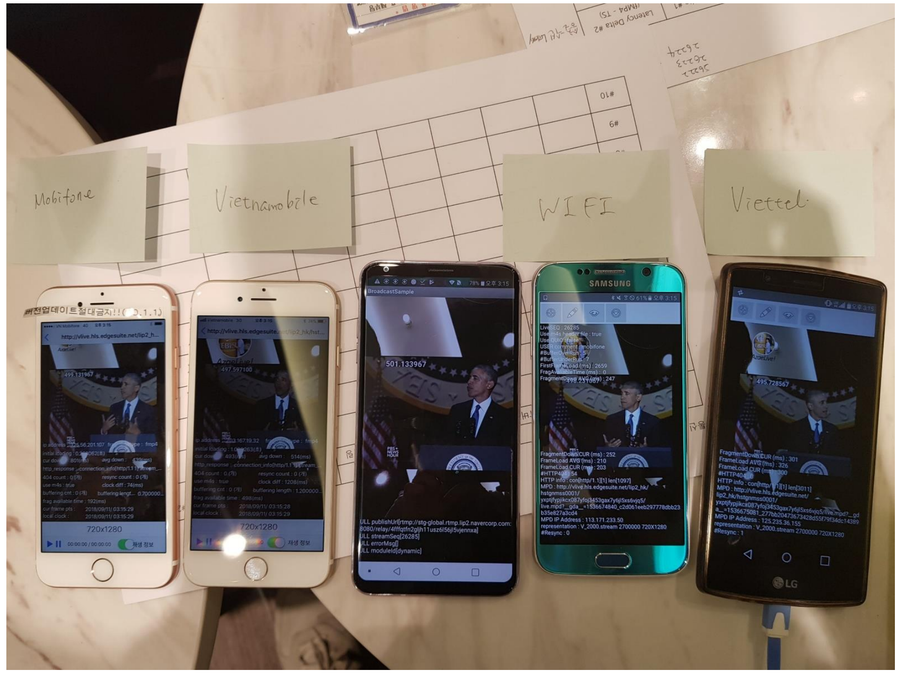
```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ...
    type="dynamic" profiles="urn:mpeg:dash:profile:isoff-live:2011“
    ...
    availabilityStartTime="2019-10-07T06:40:51.932Z" publishTime="2019-10-07T06:41:02.932Z">
    <UTCTiming schemeIdUri="urn:mpeg:dash:utc:direct:2014" value="2019-10-07T06:41:02.932Z"/>
    <Period start="PT0.0S" id="1">
        ...
        <SegmentTemplate timescale="44100" media="$RepresentationID$/$Number%09d$.m4a" ... />
....
```

3.2 그럼에도 불구하고, 어려운 점은

- 무턱대고 동영상 파일의 길이를 줄이면
  - Latency는 줄일 수 있으나,
  - 재생이 불안할 수 있다. == 네트워크 상황에 따라 잦은 버퍼링 발생

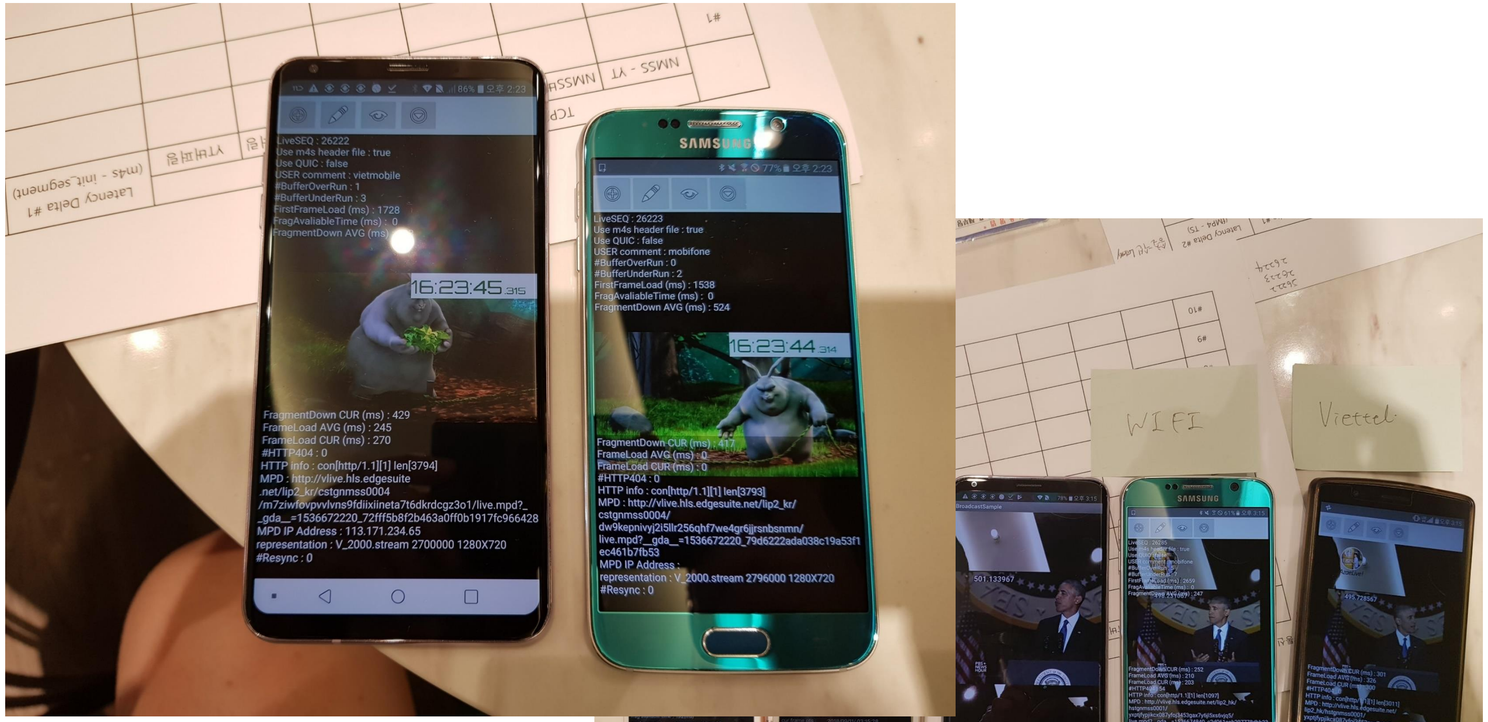
- 그럼 어떻게 하나요??
  - 거의 유일한 방법은 **반복적인 테스트와 튜닝**
  - 네트워크 상황 판단과 최신 동영상 파일을 유추하는 **로직 개선**

3.2 시도 #3 : 테스트하고 튜닝하고

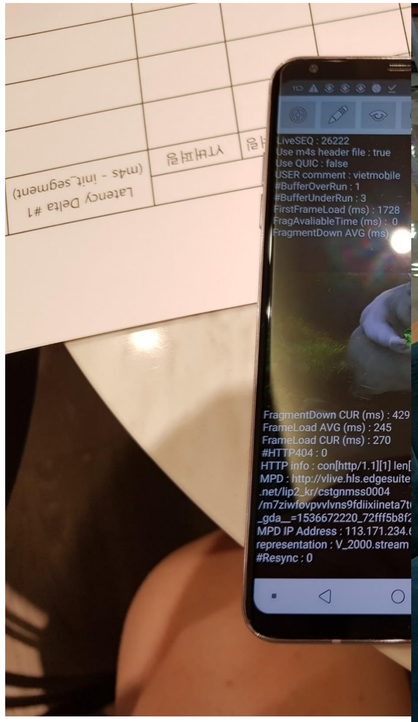


3.2 시도 #3 : 또 테스트하고 튜닝하고

DEVIEW
2019



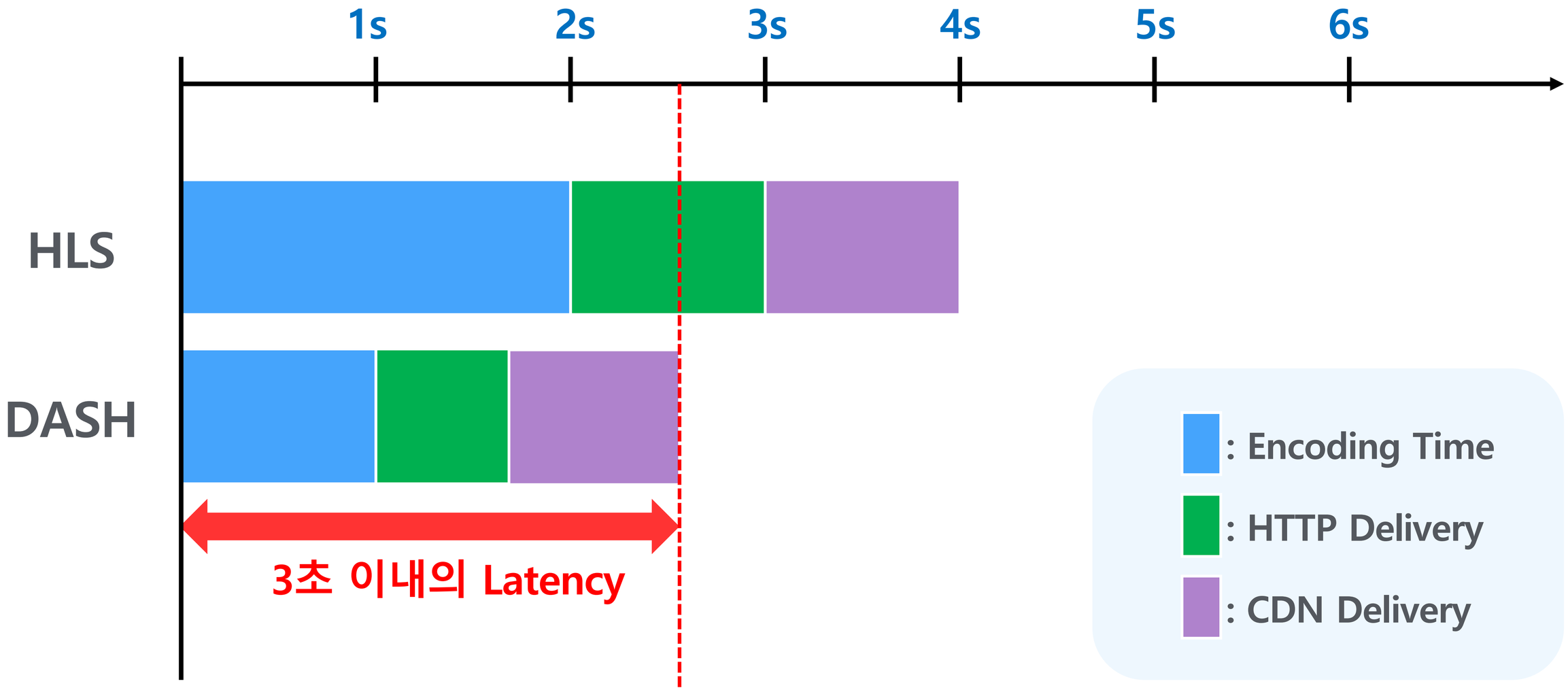
3.2 좌절 : 또 테스트 하….


LiveSEQ : 26222
Use m4s header file : true
Use QUIC : false
USER comment : vietmobile
#BufferOverRun : 1
#BufferUnderRun : 3
FirstFrameLoad (ms) : 1728
FragAvaliableTime (ms) : 0
FragmentDown AVG (ms)
FragmentDown CUR (ms) : 429
FrameLoad AVG (ms) : 245
FrameLoad CUR (ms) : 270
#HTTP404 : 0
HTTP info : con[http/1.1][1] len[
MPD : http://vlive.hls.edgesuite
.net/lip2_kr/cstgnmss0004
/m7ziwfovpvvlvns9fdiixiineta7t
_gda__=1536672220_72fff5b8f2
MPD IP Address : 113.171.234.
representation : V_2000.stream
#Resync : 0


NEW ERA

3.2 그 결과, 현재의 Latency는

DEVIEW 2019



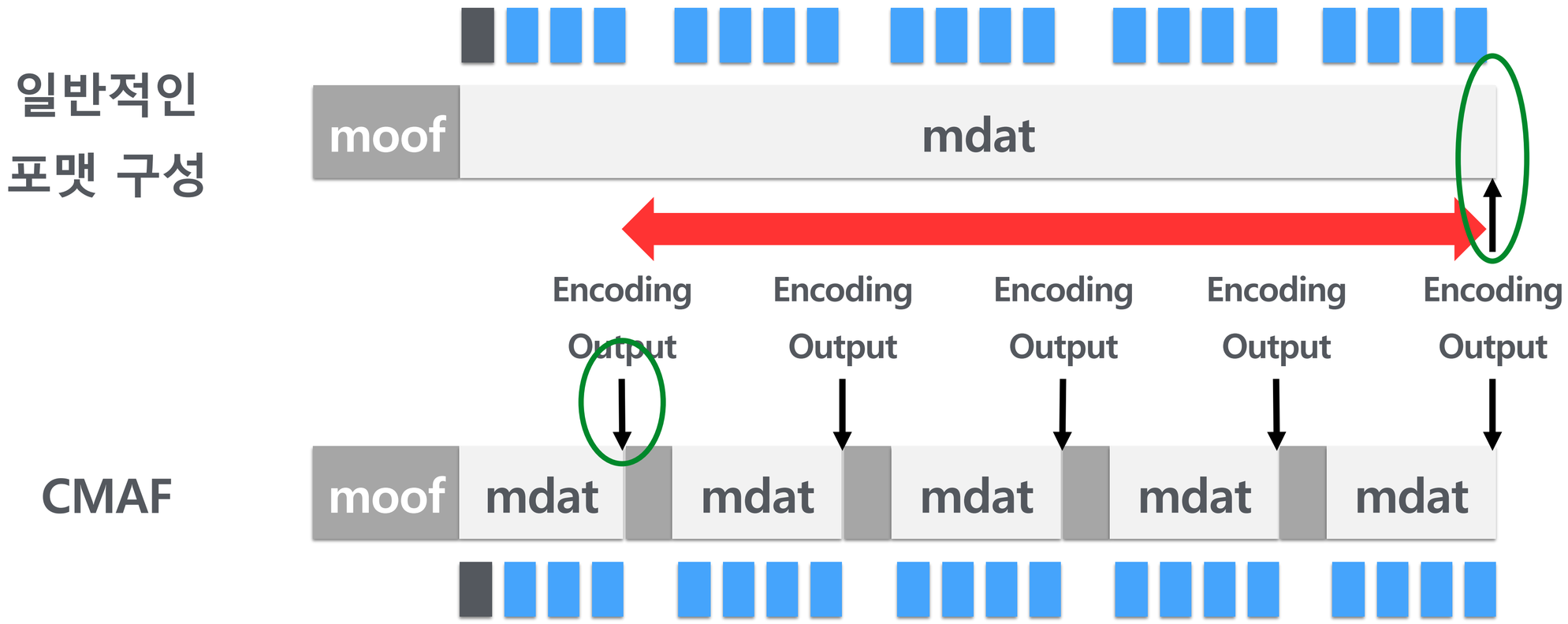
3.2 다음 목표 : Sub-Second Streaming

DEVIEW 2019

- 1초 미만(Sub-Second)의 Latency로 사용자에게 전달

- 시도 #4 : Encoding 결과물을 더 잘게 쪼개보자!

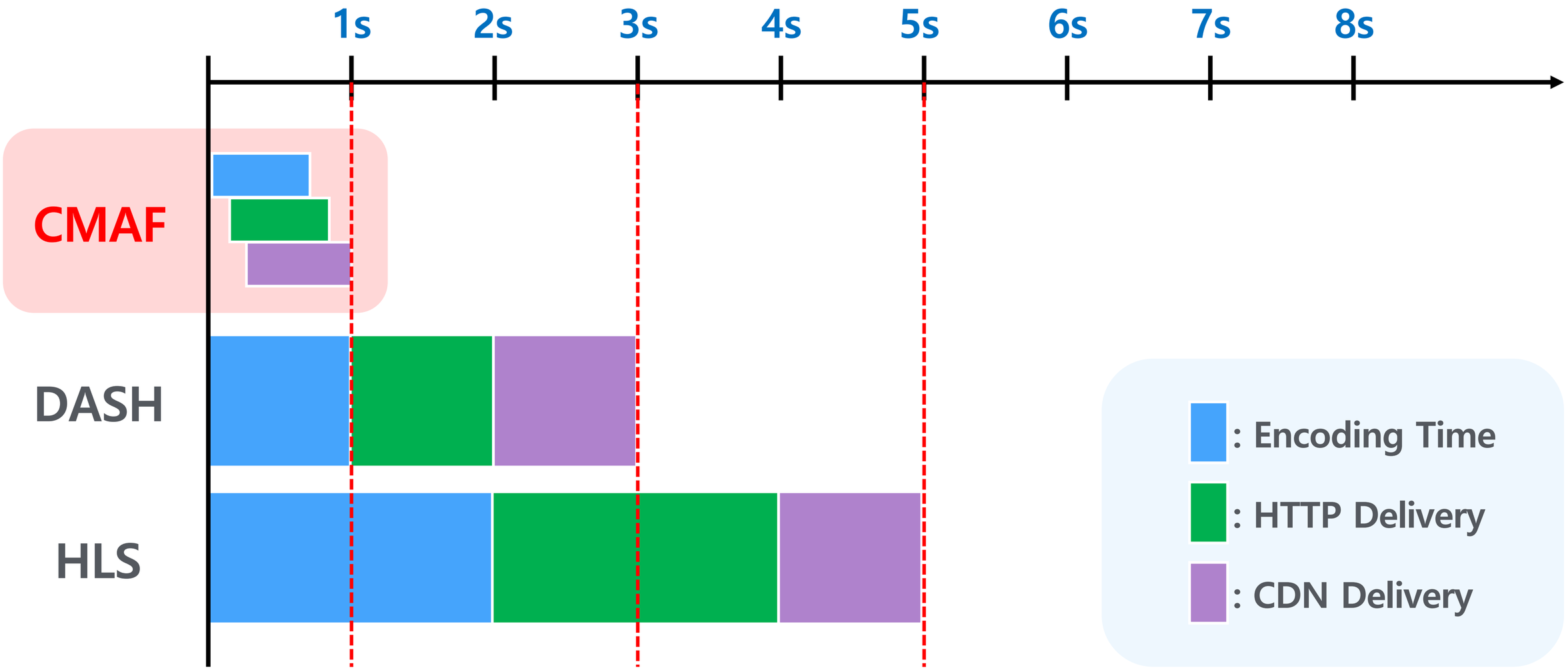
3.2 시도 #4 : CMAF 포맷



3.2 즉, 전송 시작 시간을 당길 수 있다.

DEVIEW 2019

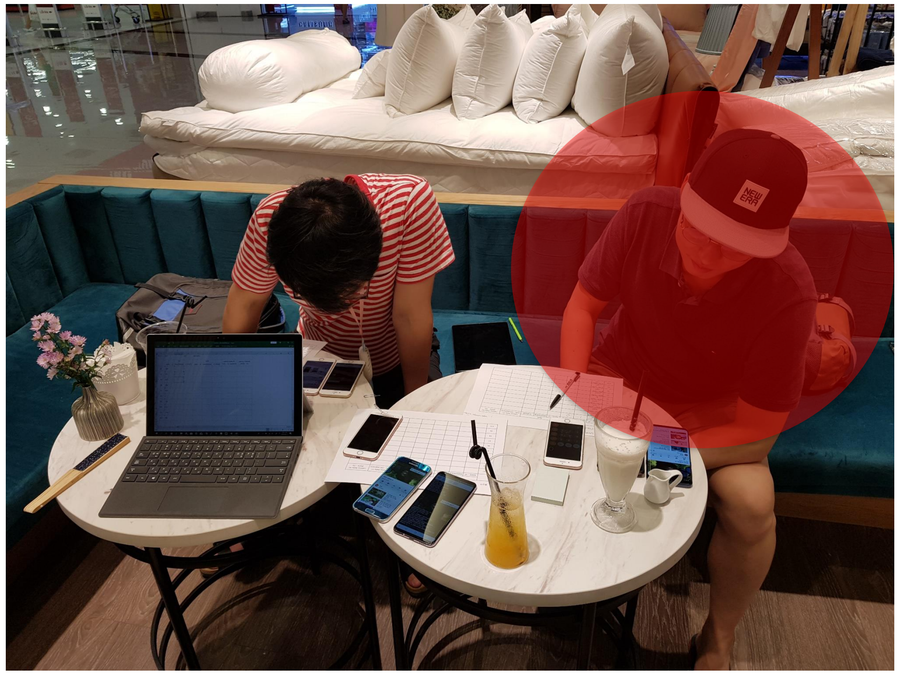
3.2 진정한 Ultra Low Latency



3.2 더 자세한 노하우 공유는…

DEVIEW 2019

- 바로 오늘, 5번째 세션 4번째 트랙
- 동영상 스트리밍 서비스, 빠르게 더 빠르게 (Ultra Low Latency, Ultra Fast Playing)
- 다양한 테스트와 개선으로 이뤄낸 노하우들을 공유합니다.



3.2 송출 기술과 재생 디바이스의 발전

DEVIEW
2019

- HD 720p
- Full HD 1080p
- Ultra HD 4K
- Full UHD 8K

- 고화질 라이브 스트리밍에 대한 준비가 필요

3.2 고화질 스트리밍을 하려면

DEVIEW
2019

- 방법 1 : 높은 해상도로 인코딩
- 방법 2 : 비트레이트를 충분히 할당

3.2 고화질 스트리밍 제약 #1

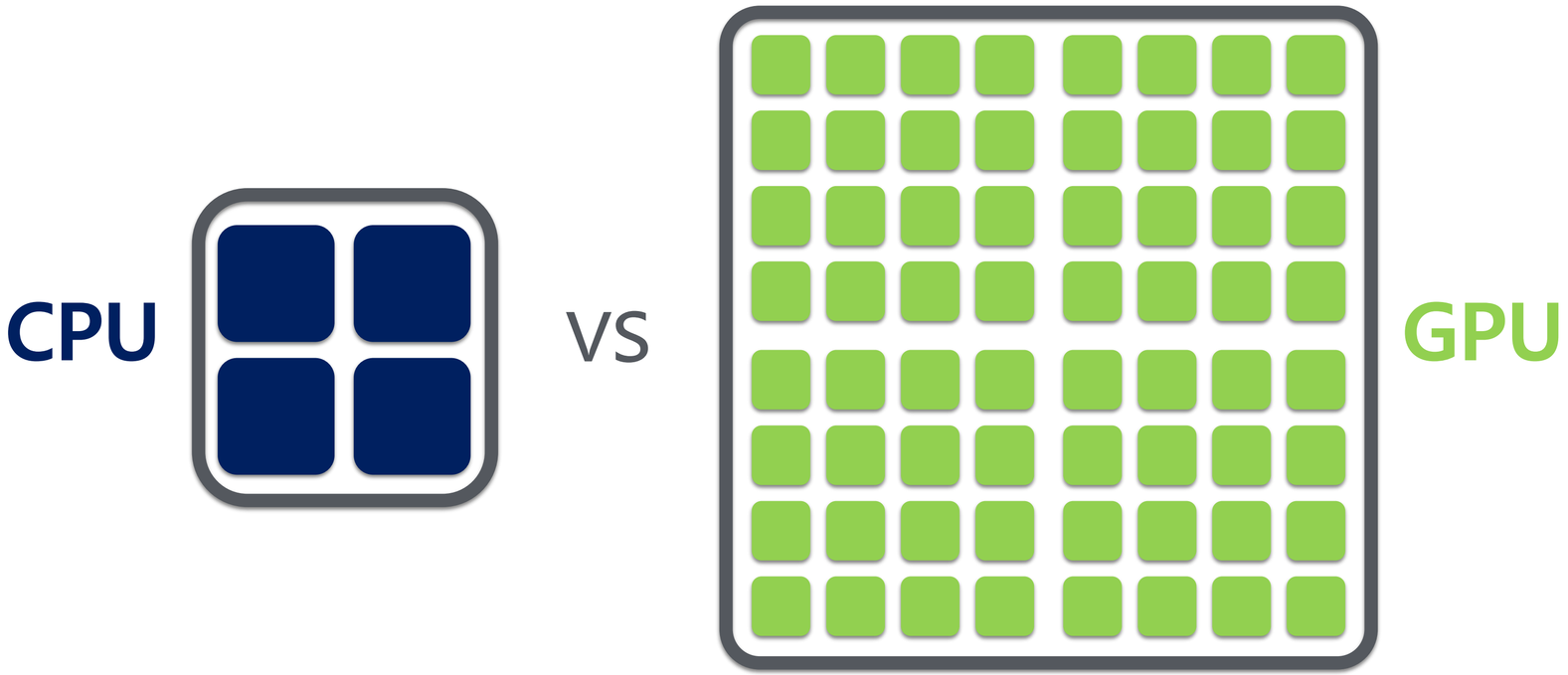
- 방법 1 : 높은 해상도로 인코딩
  - CPU 기반의 S/W 처리로는 **실시간 처리 불가** (고해상도 X, 신규코덱 X)
- 방법 2 : 비트레이트를 충분히 할당

3.2 고화질 스트리밍 제약 #2

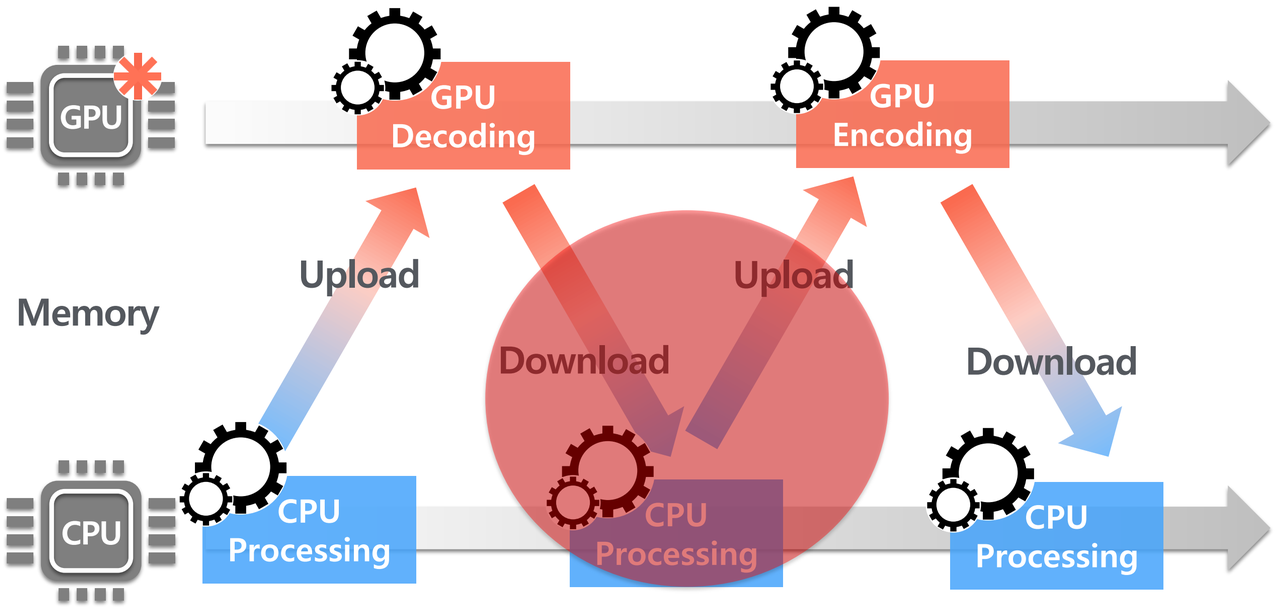
- 방법 1 : 높은 해상도로 인코딩
  - CPU 기반의 S/W 처리로는 **실시간 처리 불가** (고해상도 X, 신규코덱 X)
- 방법 2 : 비트레이트를 충분히 할당
  - 무턱대고 높이면 사용자들의 **네트워크 비용이 증가**

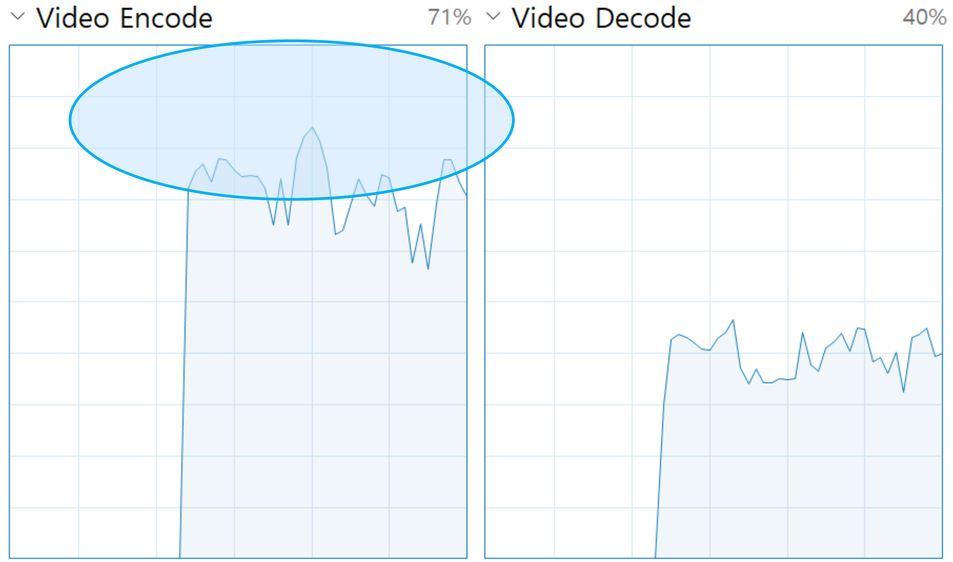
3.2 GPU Processing 도입

- 효율적인 병렬 처리로 고해상도 / HEVC 코덱 처리가 가능



3.2 GPU Processing의 병목 구간





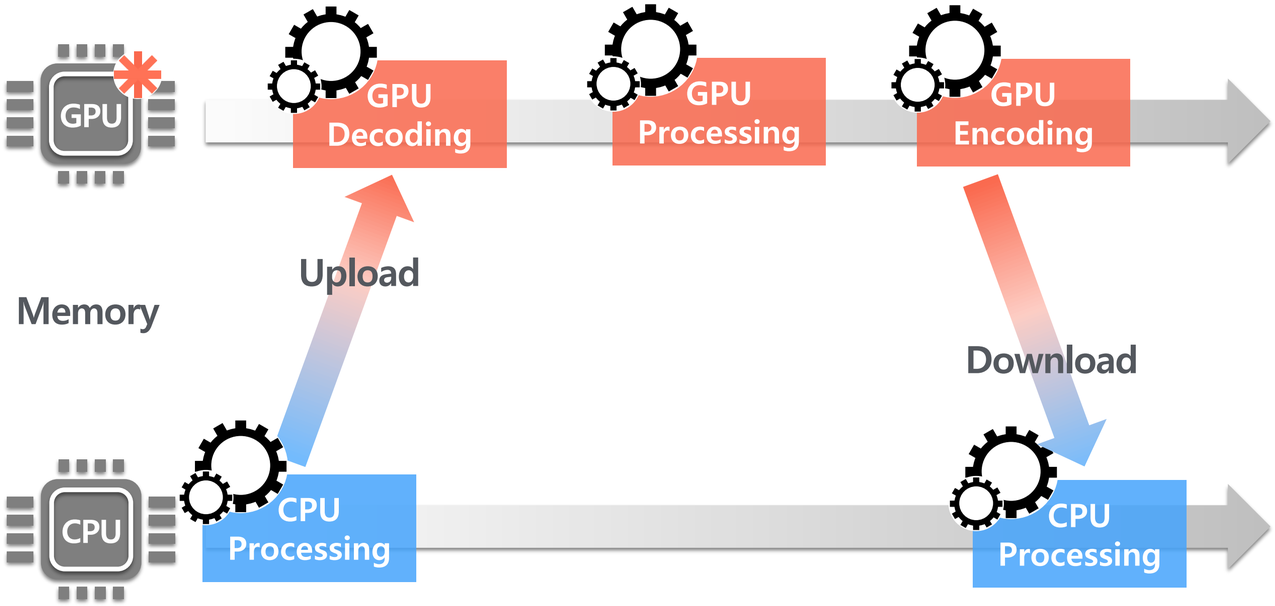
3.2 GPU Processing 최적화

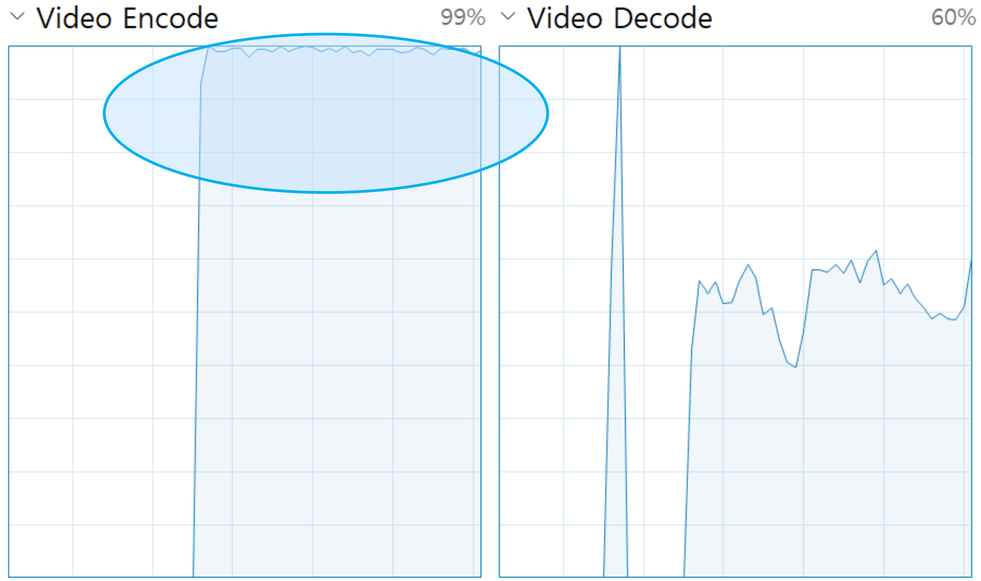
DEVIEW 2019

- GPU Processing 최적화를 위한 기반 기술
  - CUDA / OpenGL을 최대한 활용하자!
  - 기존 CPU 기반의 영상처리 기능을 GPU 위로
    => 영상 Resizing, color space conversion
  - zero-copy pipeline 구현으로 성능 향상

3.2 GPU Processing 최적화

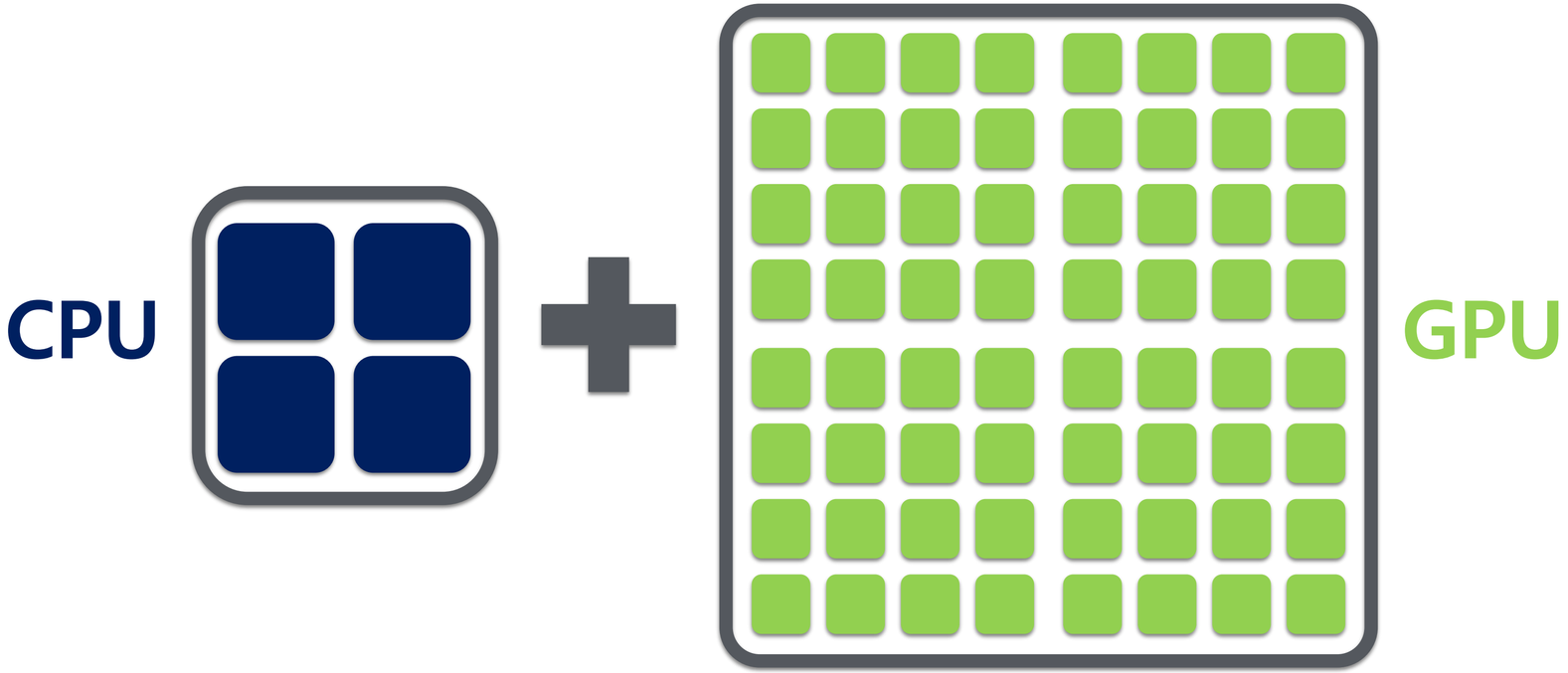
- GPU 리소스를 최대한으로 끌어올려서





3.2 CPU와 GPU의 접목

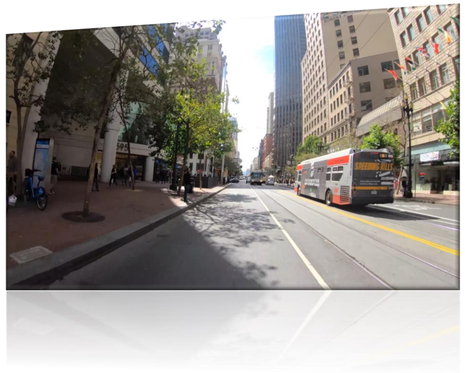
- 고해상도 처리는 GPU로, 저해상도 처리는 CPU로 병렬 처리



3.2 GPU 적용 : 고해상도 인코딩 기능 확보

DEVIEW 2019

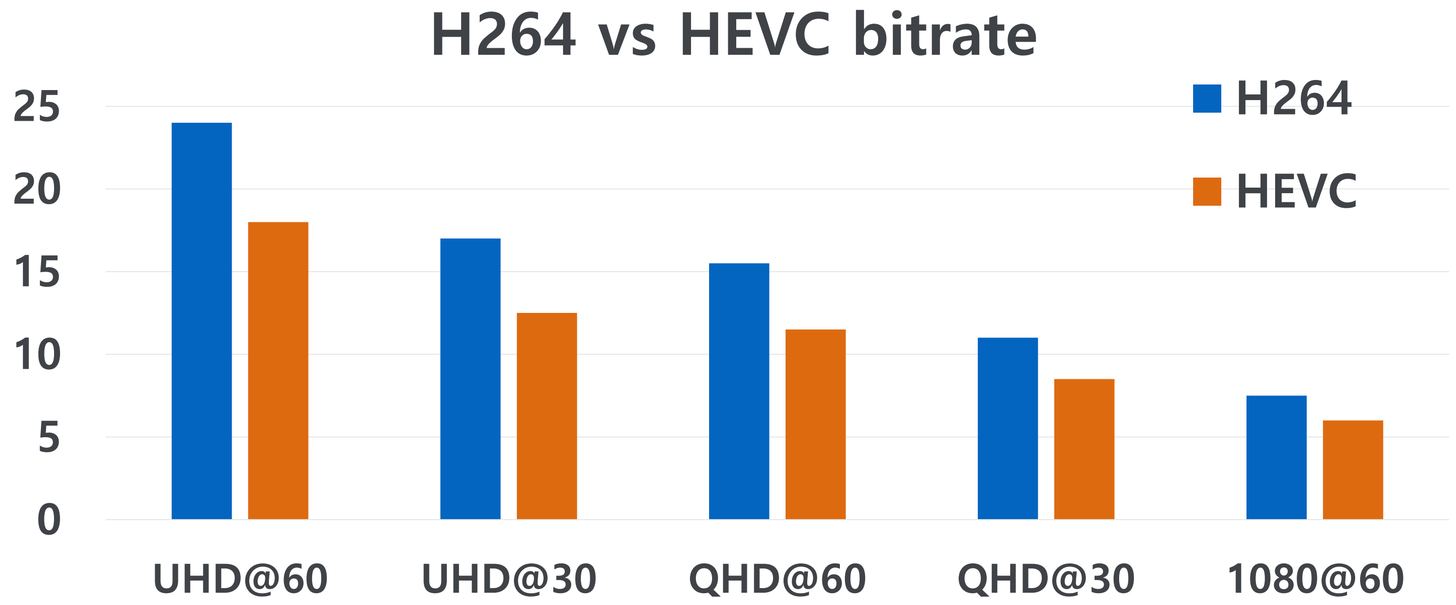
CPU 기반



GPU 기반

3.2 GPU 적용 : 압축률 향상

- HEVC 코덱은 H.264 코덱 대비 **최대 30% 의 압축률 향상**을 보임



DEVIEW
2019

3-3. DevOps를 위한 노력

3.3 안정성을 갖춰야 하는 이유

DEVIEW
2019

- 미디어 스트리밍 서버

3.3 안정성을 갖춰야 하는 이유

DEVIEW 2019

- 미디어 스트리밍 **서버** == 라이브 방송이 중단되면 안된다

무사고 운영

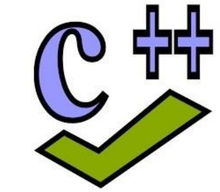
DevOps의 필요성

3.3 무사고 운영을 하려면

DEVIEW
2019

- 사고를 **예방**
  - ✓ 개발 단계에서의 꼼꼼한 체크로 미연에 방지
  - ✓ 예상 가능한 예외 상황들을 최대한 사전에 체크

3.3 사고 예방을 위해 활용 중인 보조 도구



Cppcheck RAHIM SOFTWARES

sonarqube

googletest
Google C++ Testing Framework

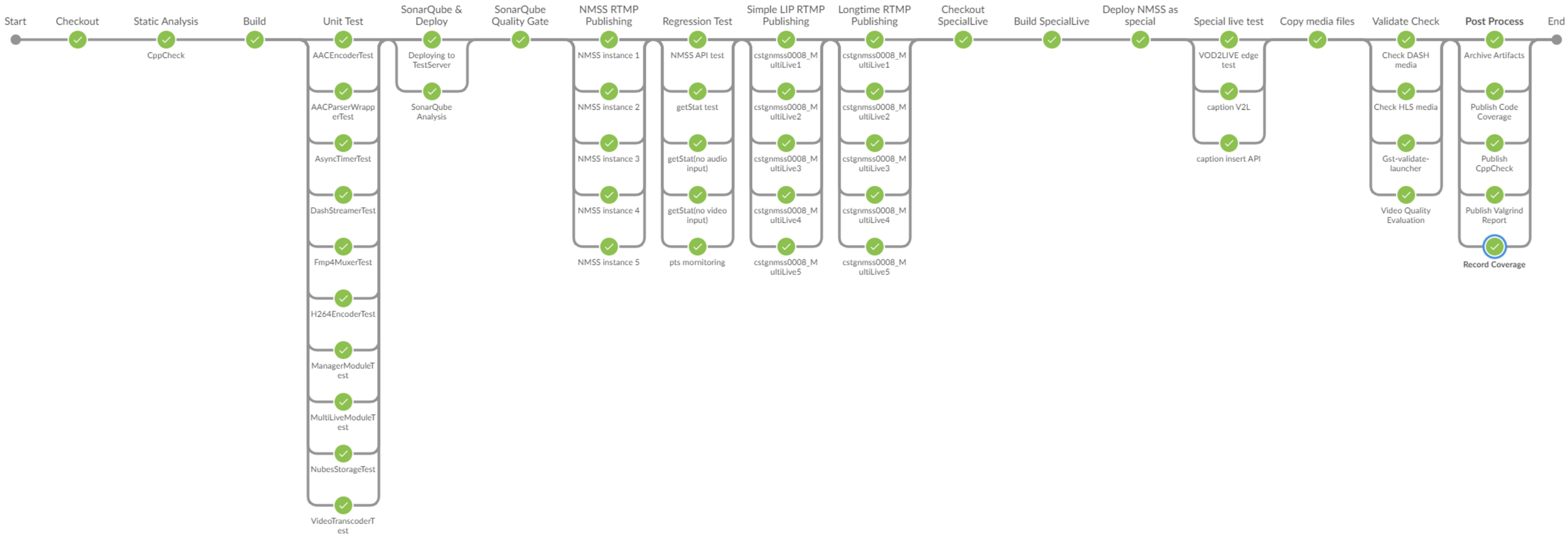
Valgrind

Jenkins

3.3 한 줄의 코드를 배포하기까지

DEVIEW 2019

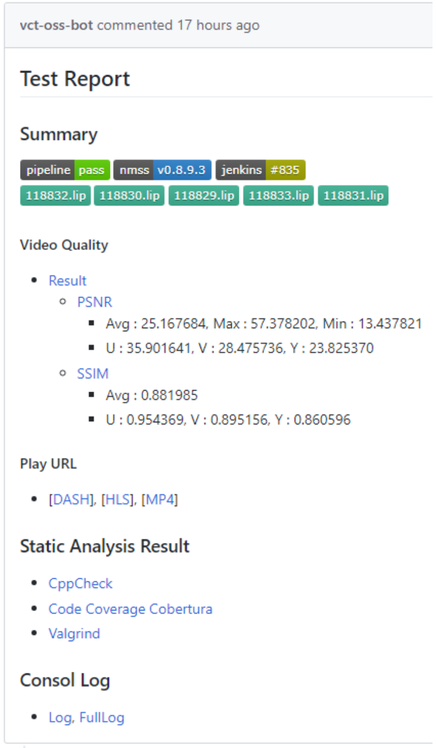
- ✓ CI/CD 자동화 구축 : 개발자 리소스 절감
- ✓ 빌드, 정적 분석, 배포, 테스트, 리포트 과정 모두 자동화하여 안정성 수시 체크



3.3 한 줄의 코드를 배포하기까지

DEVIEW 2019

- ✓ Daily : 빌드 테스트 + 장시간(부하) 테스트 등 Full Test 수행
- ✓ PR : 중요한 테스트 위주로 수행 후 결과를 PR에 리포트



vct-oss-bot commented 17 hours ago

Test Report

Summary

pipeline pass nmss v0.8.9.3 jenkins #835

118832.lip 118830.lip 118829.lip 118833.lip 118831.lip

Video Quality

- Result
  - PSNR
    - Avg : 25.167684, Max : 57.378202, Min : 13.437821
    - U : 35.901641, V : 28.475736, Y : 23.825370
  - SSIM
    - Avg : 0.881985
    - U : 0.954369, V : 0.895156, Y : 0.860596

Play URL

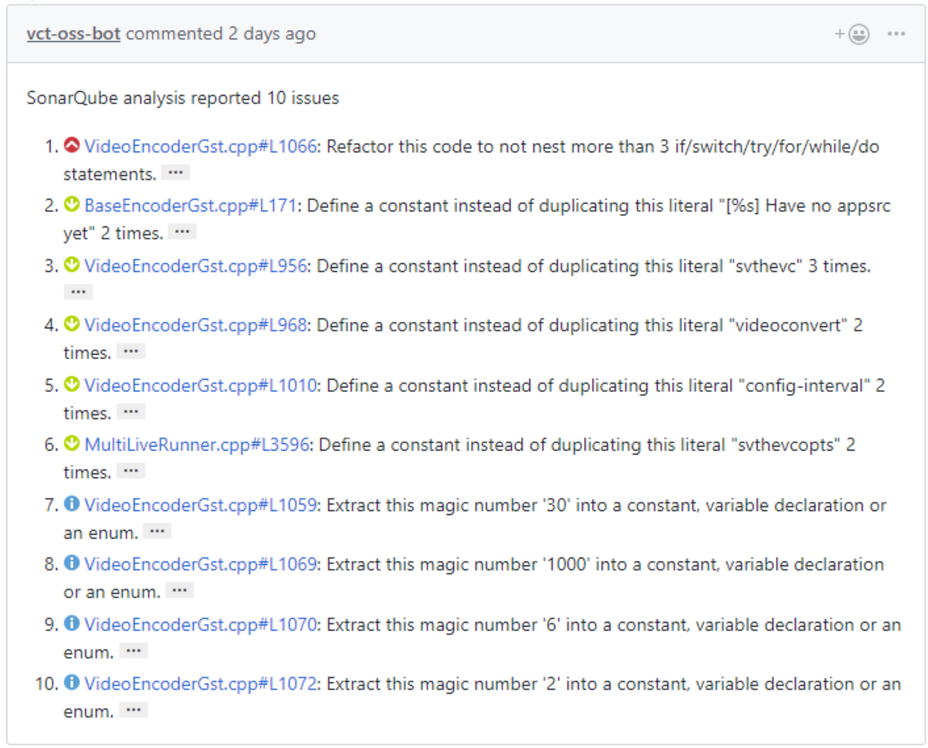
- [DASH], [HLS], [MP4]

Static Analysis Result

- CppCheck
- Code Coverage Cobertura
- Valgrind

Consol Log

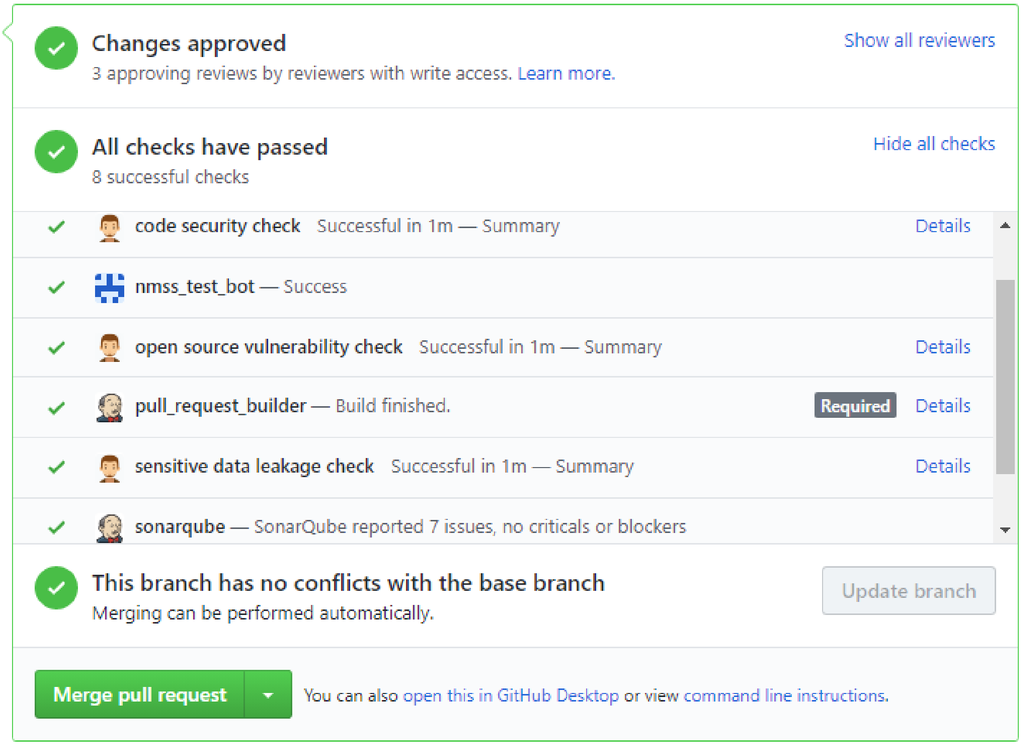
- Log, FullLog



vct-oss-bot commented 2 days ago

SonarQube analysis reported 10 issues

1. VideoEncoderGst.cpp#L1066: Refactor this code to not nest more than 3 if/switch/try/for/while/do statements.
2. BaseEncoderGst.cpp#L171: Define a constant instead of duplicating this literal "[%s] Have no appsrc yet" 2 times.
3. VideoEncoderGst.cpp#L956: Define a constant instead of duplicating this literal "svthevc" 3 times.
4. VideoEncoderGst.cpp#L968: Define a constant instead of duplicating this literal "videoconvert" 2 times.
5. VideoEncoderGst.cpp#L1010: Define a constant instead of duplicating this literal "config-interval" 2 times.
6. MultiLiveRunner.cpp#L3596: Define a constant instead of duplicating this literal "svthevcopts" 2 times.
7. VideoEncoderGst.cpp#L1059: Extract this magic number '30' into a constant, variable declaration or an enum.
8. VideoEncoderGst.cpp#L1069: Extract this magic number '1000' into a constant, variable declaration or an enum.
9. VideoEncoderGst.cpp#L1070: Extract this magic number '6' into a constant, variable declaration or an enum.
10. VideoEncoderGst.cpp#L1072: Extract this magic number '2' into a constant, variable declaration or an enum.



Changes approved — Show all reviewers

3 approving reviews by reviewers with write access. Learn more.

All checks have passed — Hide all checks

8 successful checks

- code security check — Successful in 1m — Summary — Details
- nmss_test_bot — Success
- open source vulnerability check — Successful in 1m — Summary — Details
- pull_request_builder — Build finished. — Required — Details
- sensitive data leakage check — Successful in 1m — Summary — Details
- sonarqube — SonarQube reported 7 issues, no criticals or blockers

This branch has no conflicts with the base branch — Update branch

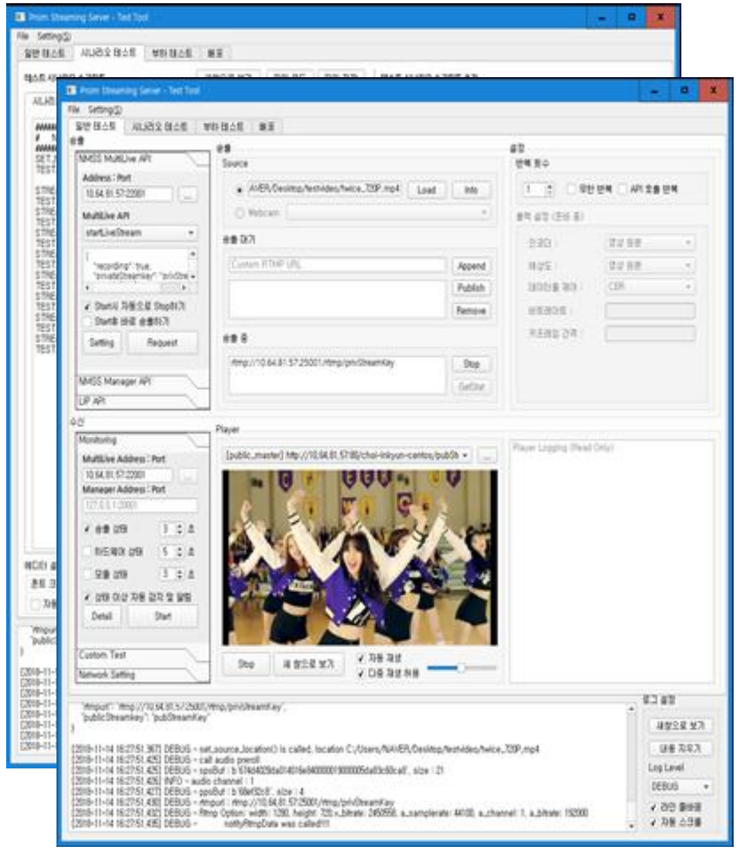
Merging can be performed automatically.

Merge pull request — You can also open this in GitHub Desktop or view command line instructions.

3.3 시나리오 테스트 자동화

DEVIEW 2019

- ✓ 자체 테스트 도구(GUI/CLI) 및 시나리오 테스트용 스크립트 파서 개발
- ✓ API 호출부터 RTMP 송출, 재생, 네트워크 제어 등 다양한 테스트 결과 확인



```
##############################################
#  Simple RTMP publishing using lip api #
##############################################
SET_VIDEO_FILE, $filepath
TEST_SLEEP, 1000
LIP_GET_PP, dev, 2000l
TEST_SLEEP, 1000
RTMP_PUBLISH
TEST_SLEEP, 60000
LIP_PLAYINFO
TEST_SLEEP, 1000
LIP_STATUS
TEST_SLEEP, 1000
LIP_STATUS
TEST_SLEEP, 1000
LIP_STATUS
TEST_SLEEP, 1000
LIP_STATUS
TEST_SLEEP, 1000
RTMP_PUBLISH_STOP
LIP_RELEASE_PP
```

```
##############################################################
#    Special Live (VOD2LIVE) in/out time exception 1         #
# both intime and outtime are bigger than file duration(10s). #
##############################################################
SET_VIDEO_FILE, $filepath
SET_NMSS_URL, $nmss_address
STREAM_START, "{'vodmanifest' : {'intime' : 20000, 'outtime' : 20000}}"

TEST_SLEEP, 60000

STREAM_STOP
```

```
#########################################################
#    Special Live (VOD2LIVE) index exception test 1  #
# start index bigger than vodmanifest max.           #
#########################################################
SET_VIDEO_FILE, $filepath
SET_NMSS_URL, $nmss_address
STREAM_START, "{'vodtoliveinfo' : {'startindex' : 8, 'endindex' : 2}}"

TEST_SLEEP, 26000

STREAM_STOP
```

3.3 크래시 유발 코드 사례. 문제는?


```
void foo() {
  std::string arg {“Hello"};
  std::thread{[arg]() {
    auto v = arg;
    std::cout << v << std::endl;
  }}.detach();
}
```

Lambda의 parameter capture로
string 복사 의도

3.3 std::string CoW(Copy on Write)

- Write operation이 발생할 때 메모리를 할당하고 복사하겠다는 얘기

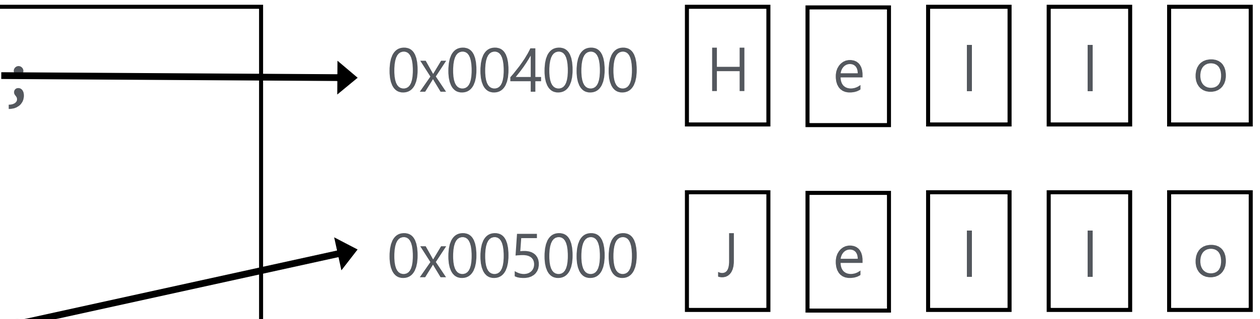
```
std::string arg {“Hello"};
std::thread{[arg]() {
    auto v = arg;
    v[0] = ‘J’;
}}.detach();
```

3.3 std::string CoW(Copy on Write)

DEVIEW 2019

- Write operation이 발생할 때 메모리를 할당하고 복사하겠다는 얘기

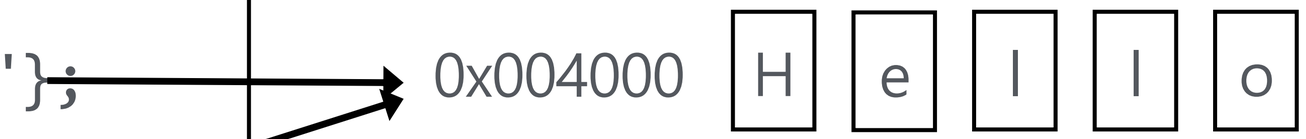
```
std::string arg {“Hello"};
std::thread{[arg]() {
    auto v = arg;
    v[0] = ‘J’;
}}.detach();
```



3.3 Write Operation이 없다면??

- Double Free의 가능성이 존재

```
void foo() {
  std::string arg {“Hello"};
  std::thread{[arg]() {
    auto v = arg;
    std::cout << v << std::endl;
    // arg, v dtor
  }}.detach();
  // arg dtor
}
```



메모리 해제는 누가??

3.3 C++ Spec 기반 검토 후 조치

- 비표준 컴파일러에 대한 회피 권고
  - C++11에서는 CoW 구현을 금지하고 있음
  - **GCC 4.x 이하**에서는 thread-safe 하지 않음
  - https://gcc.gnu.org/bugzilla/show_bug.cgi?id=21334

✓ Modern C++의 표준 Spec이 제대로 반영된 컴파일러로 버전 교체

3.3 무사고 운영을 하려면

- 사고를 예방
  - ✓ 개발 단계에서의 꼼꼼한 체크로 미연에 방지
  - ✓ 예상 가능한 예외 상황들을 최대한 사전에 체크하자.

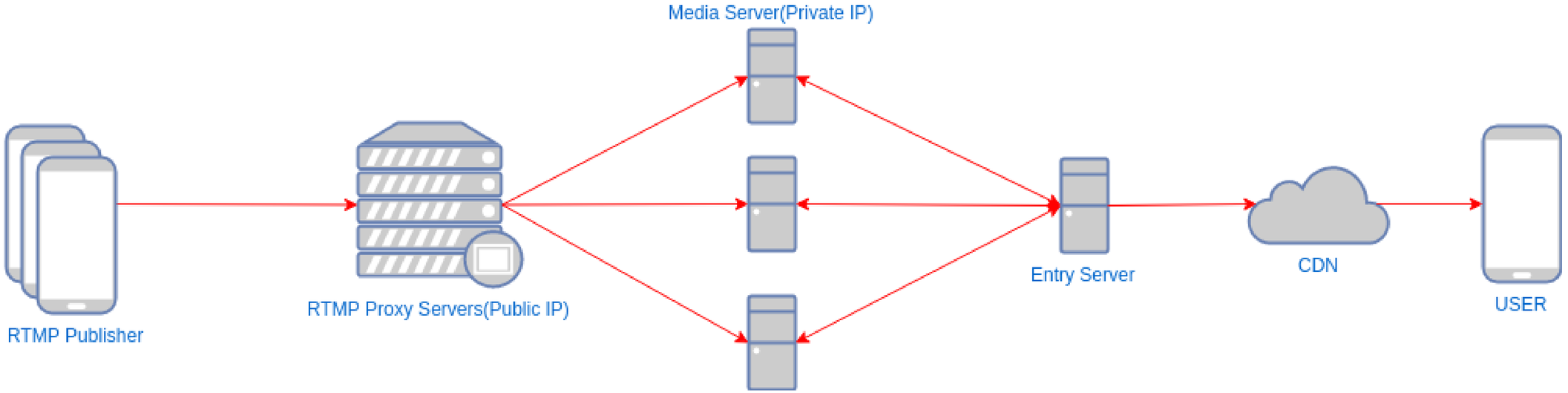
  - std::string CoW 예시처럼, **모든 예외를 막는 것은 현실적으로 불가능!**
    - C++ Spec은 방대하다…

3.3 중요한 것은 사고 발생시의 대처

- 사고가 났을 때 빠른 시간 내에 **수습할 수 있도록**
  - ➢ 사고 파악에 도움이 되는 지표와 로그들을 수집하고
  - ➢ 상시 모니터링 할 수 있게 준비
  - ➢ 이슈 발생 지점을 빠르게 파악 할 수 있다면 어느정도 가능

3.3 사고 대비 : 전구간 로그 수집

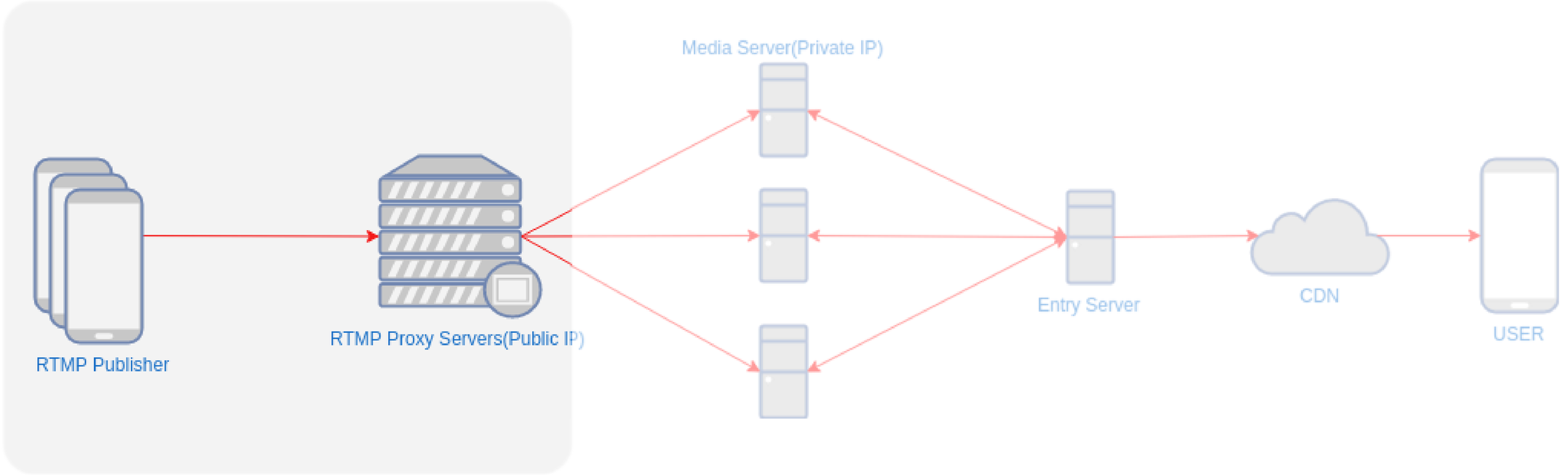
DEVIEW 2019



First Mile | Middle Mile | Last Mile

3.3 송출 데이터 수집

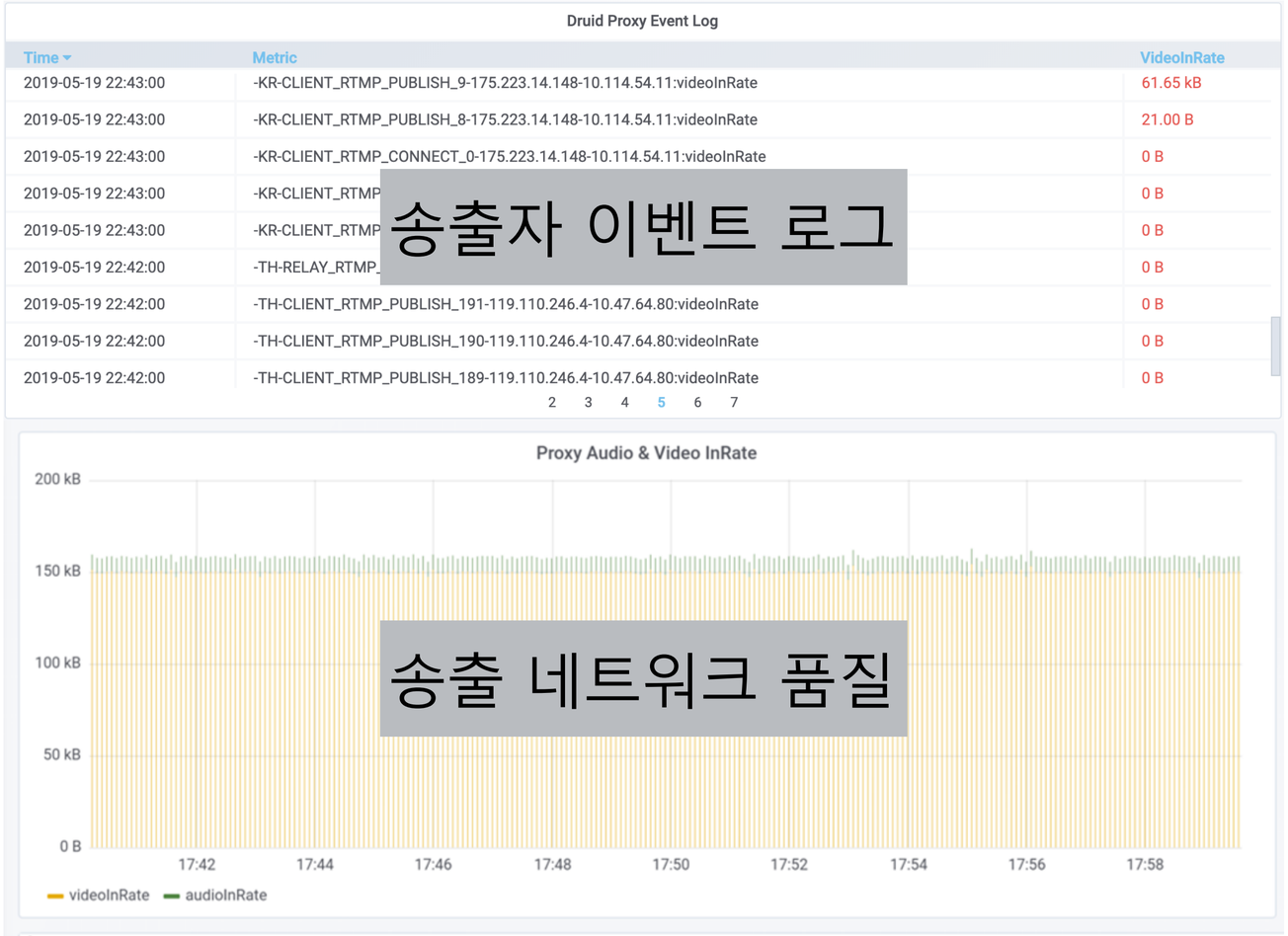
DEVIEW 2019



First Mile | Middle Mile | Last Mile

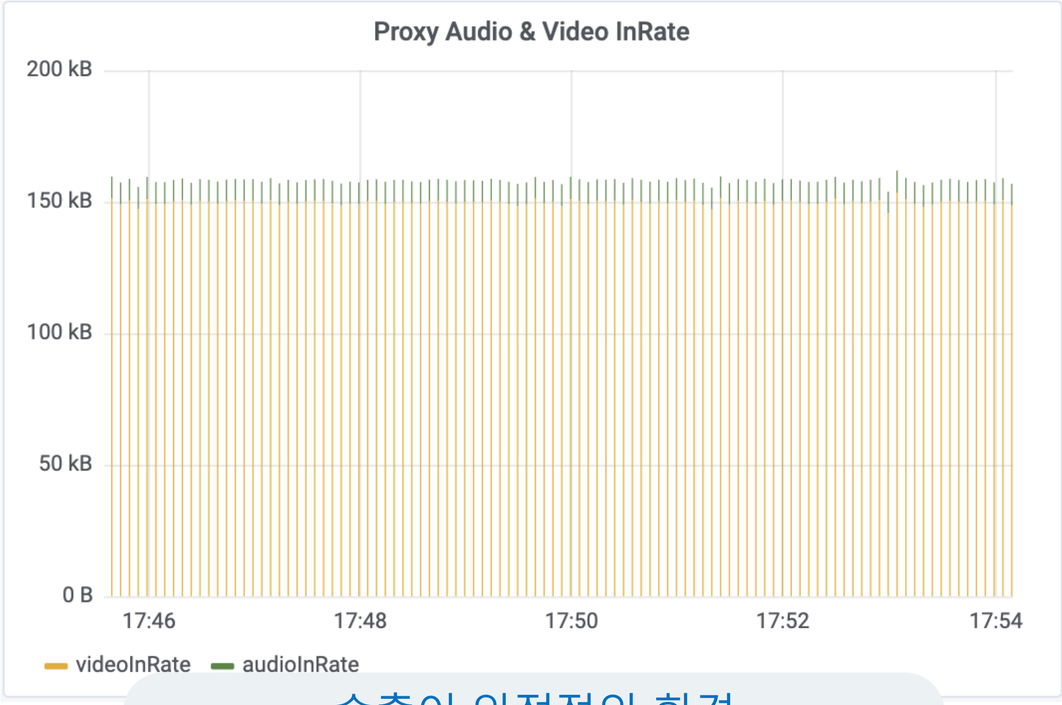
3.3 송출 데이터 수집

DEVIEW 2019



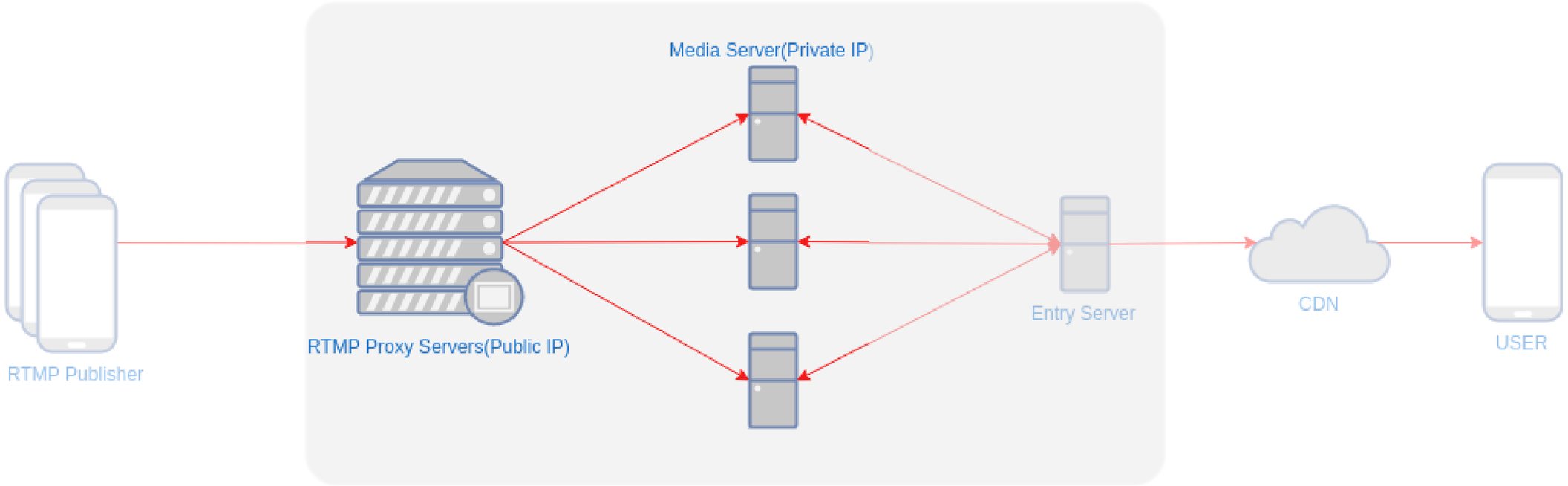
3.3 송출 환경 상태 모니터링

DEVIEW 2019



송출이 안정적인 환경

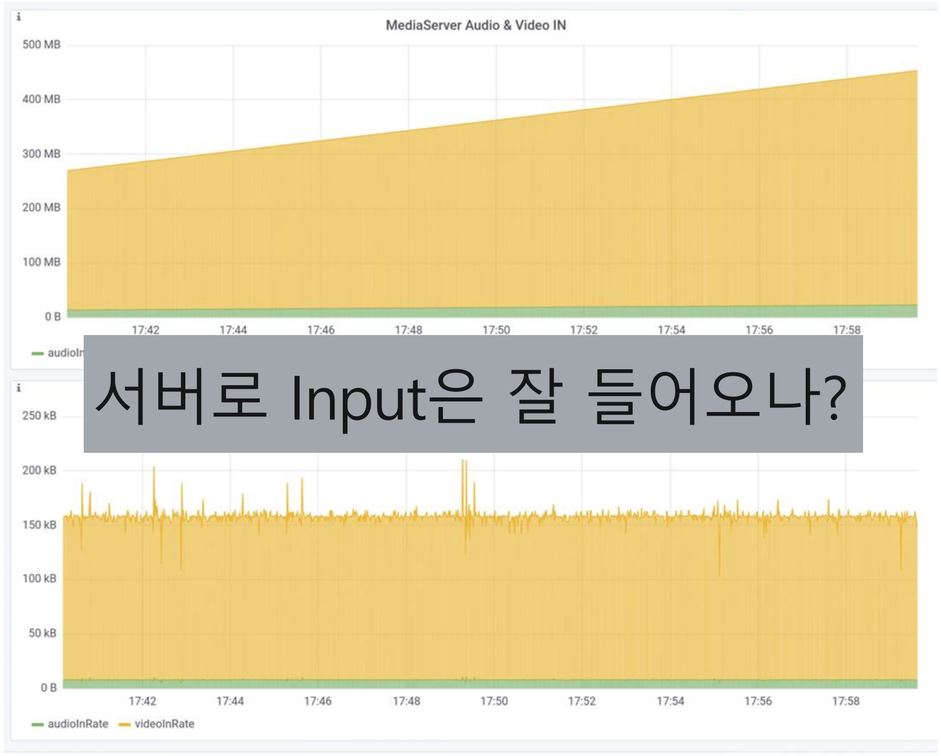
3.3 Middle Mile Layer



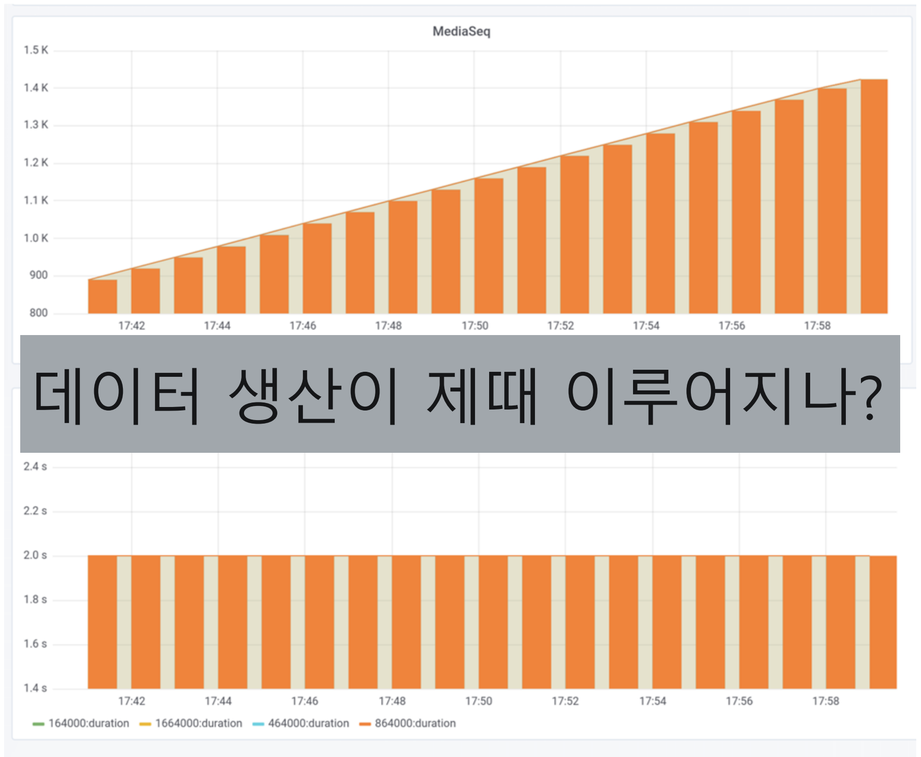
First Mile | Middle Mile | Last Mile

3.3 미디어 데이터 처리 관련 지표

DEVIEW 2019



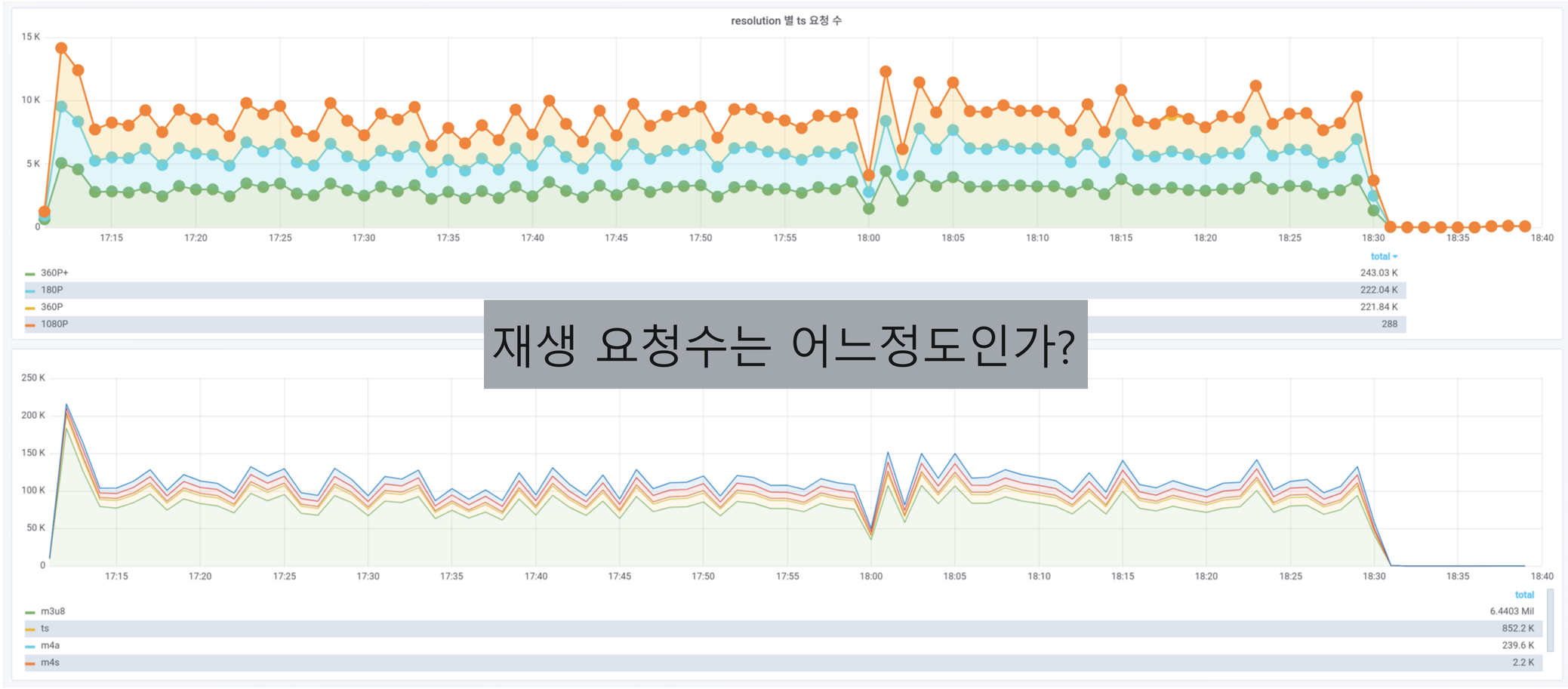
서버로 Input은 잘 들어오나?



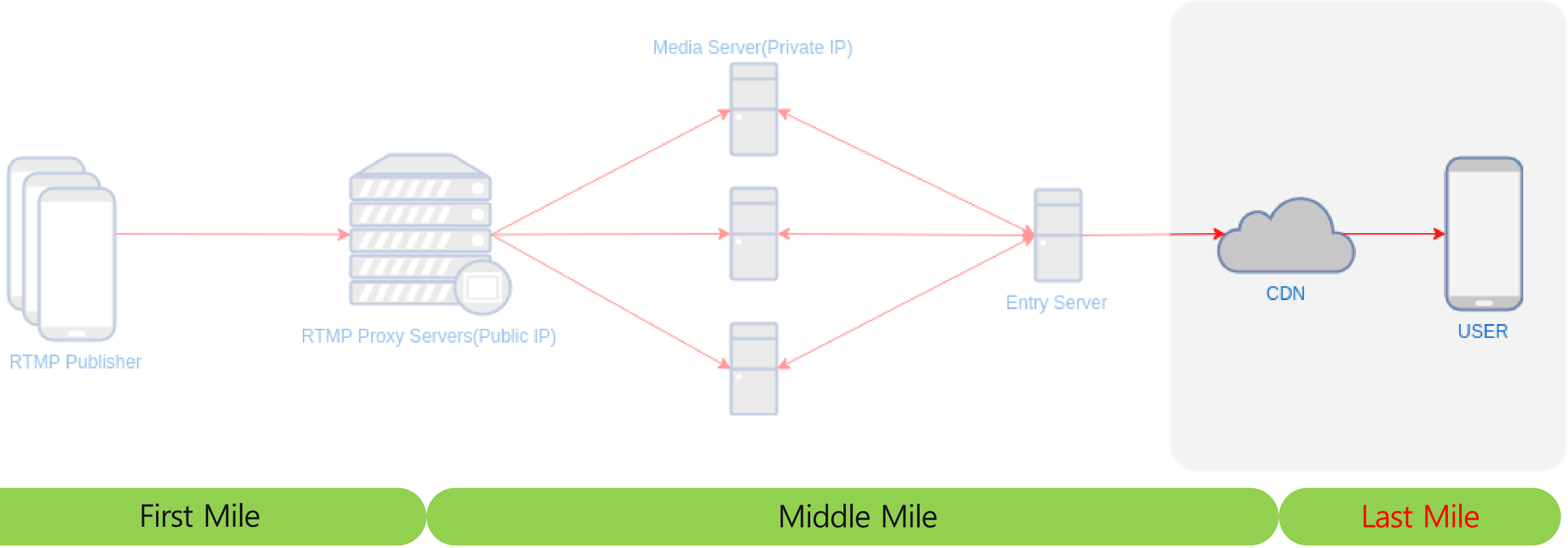
데이터 생산이 제때 이루어지나?

3.3 데이터 전송 관련 지표

DEVIEW 2019

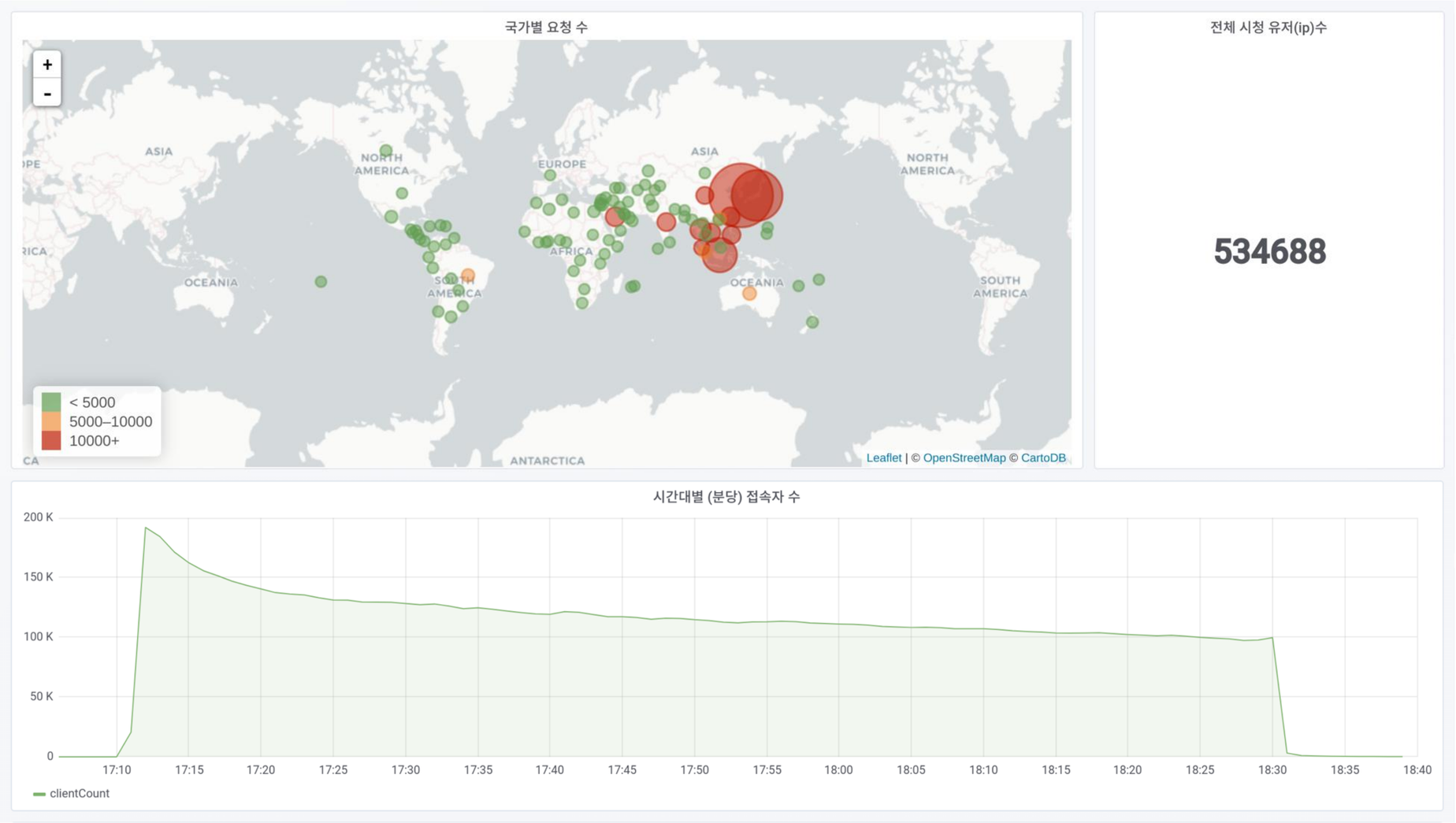


3.3 재생구간 지표


Media Server(Private IP)
RTMP Publisher
RTMP Proxy Servers(Public IP)
Entry Server
CDN
USER
First Mile
Middle Mile
Last Mile

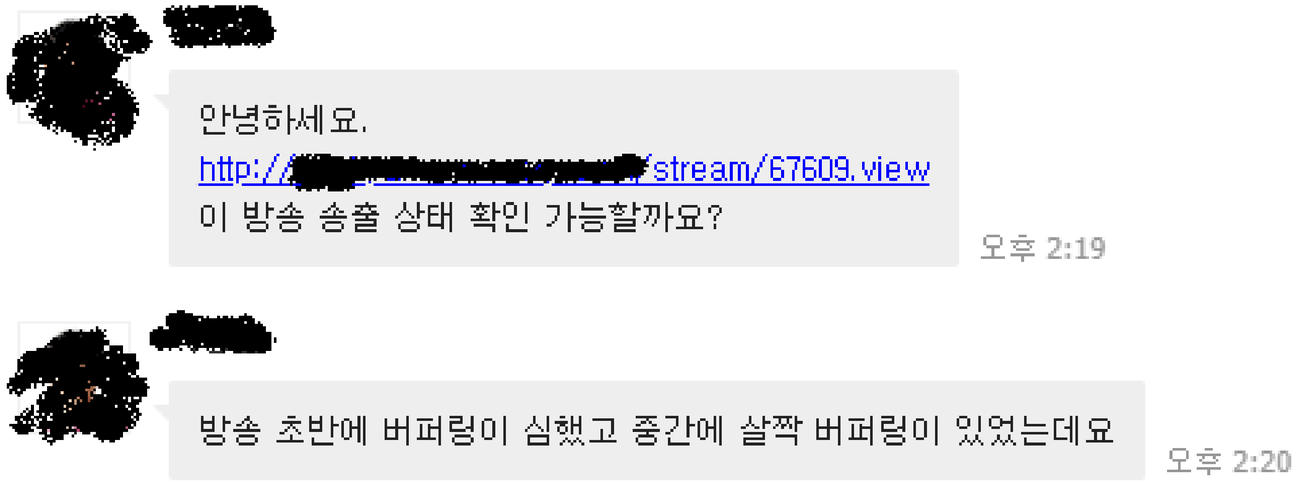
3.3 재생구간 지표

DEVIEW 2019



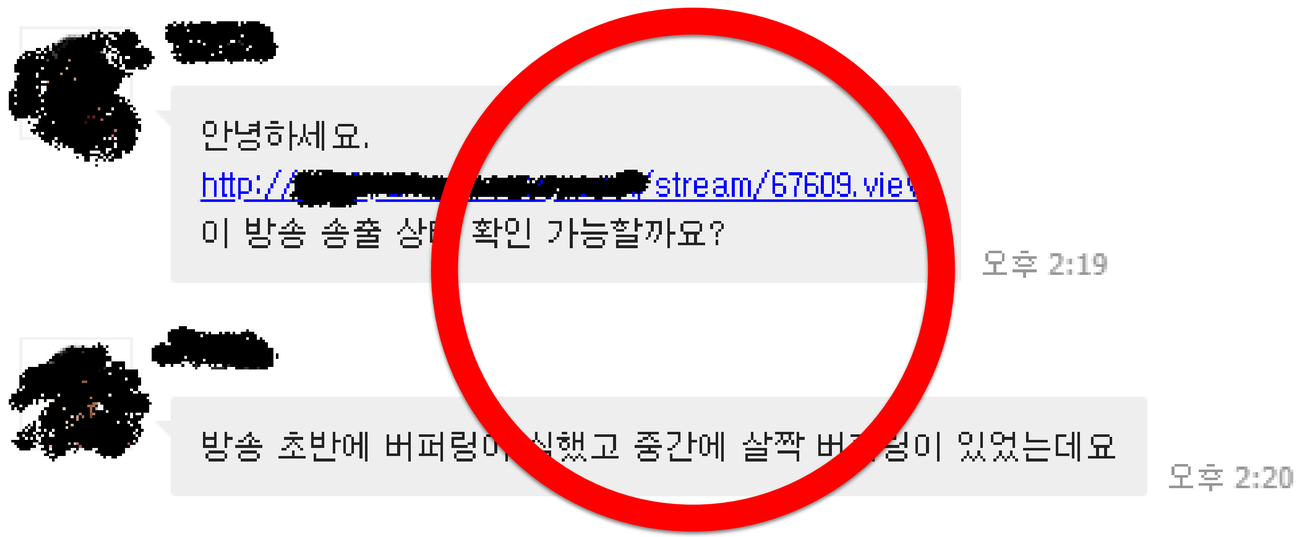
3.3 이제 이런 문의가 오면

DEVIEW
2019

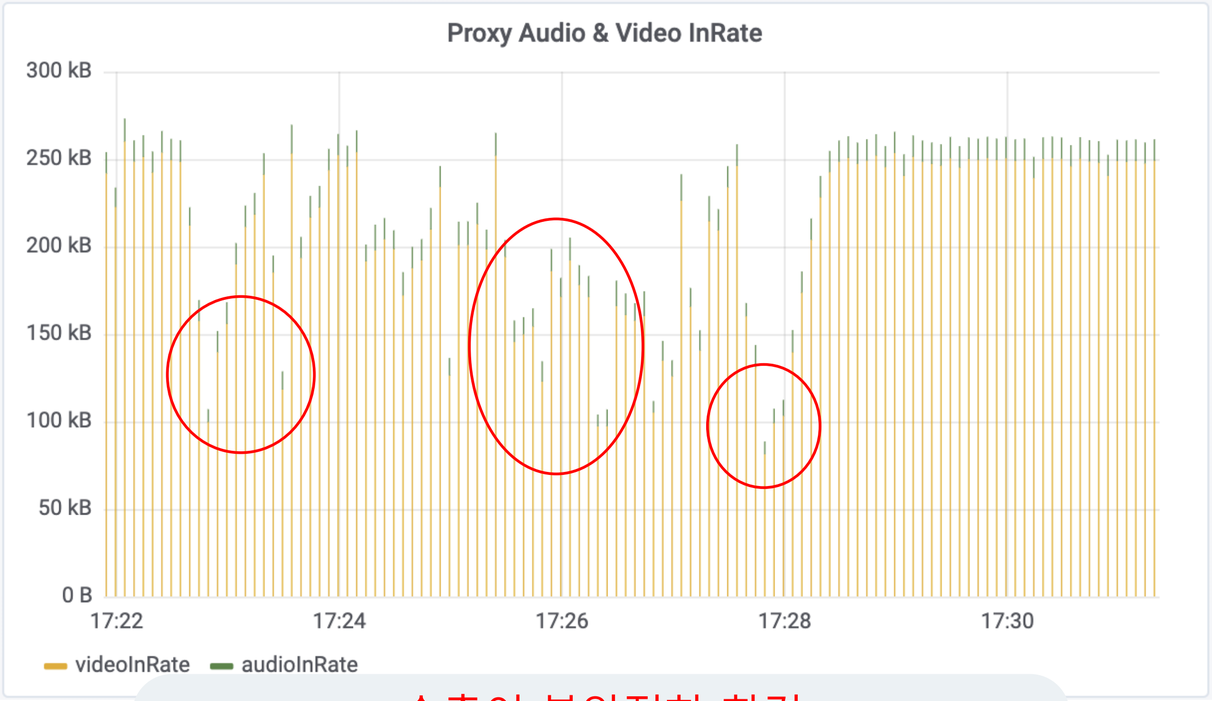


3.3 네. 가능합니다.

DEVIEW
2019



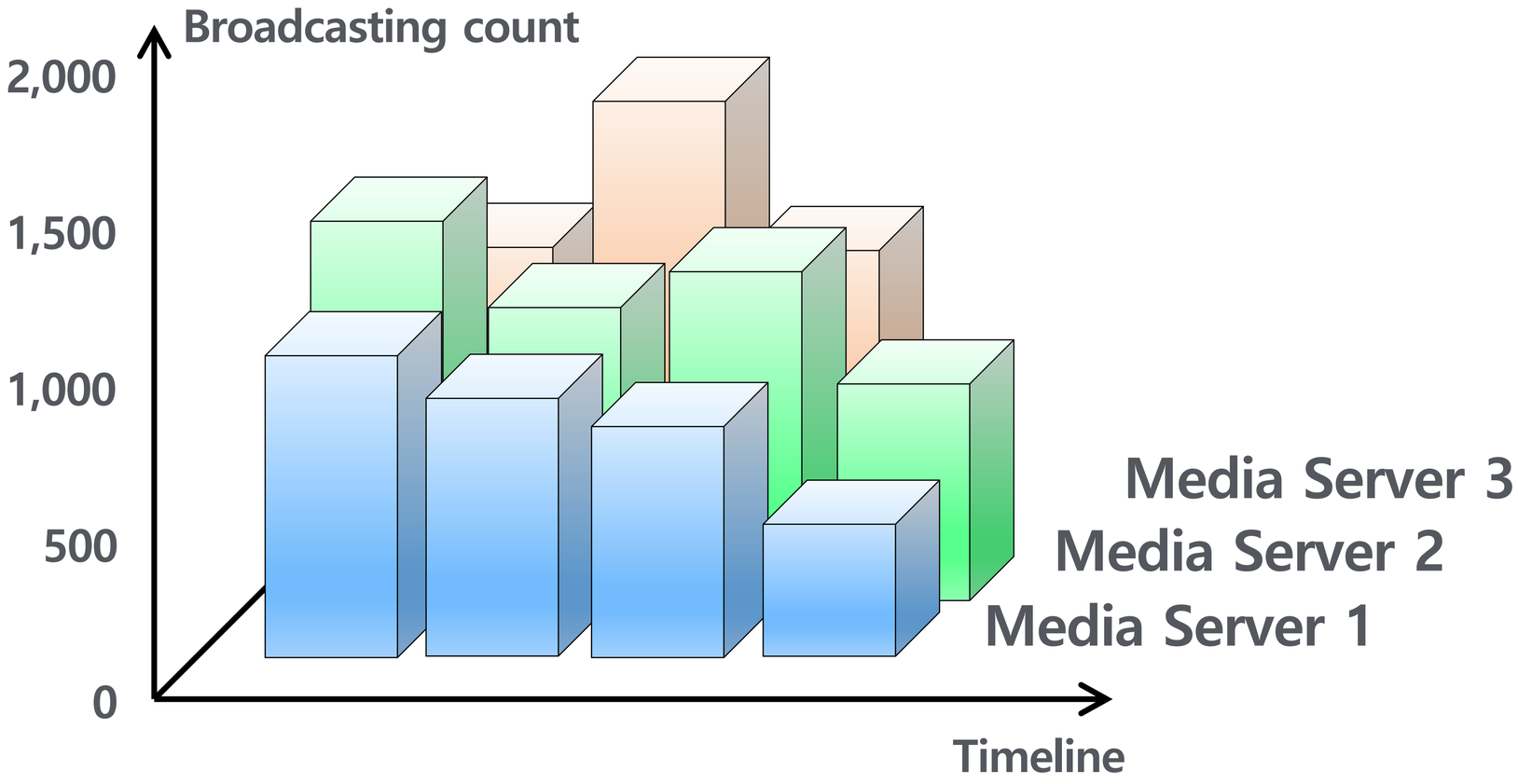
3.3 송출 환경의 이상 감지 사례



송출이 불안정한 환경

3.3 지표 수집 이후, 또 하나의 고민

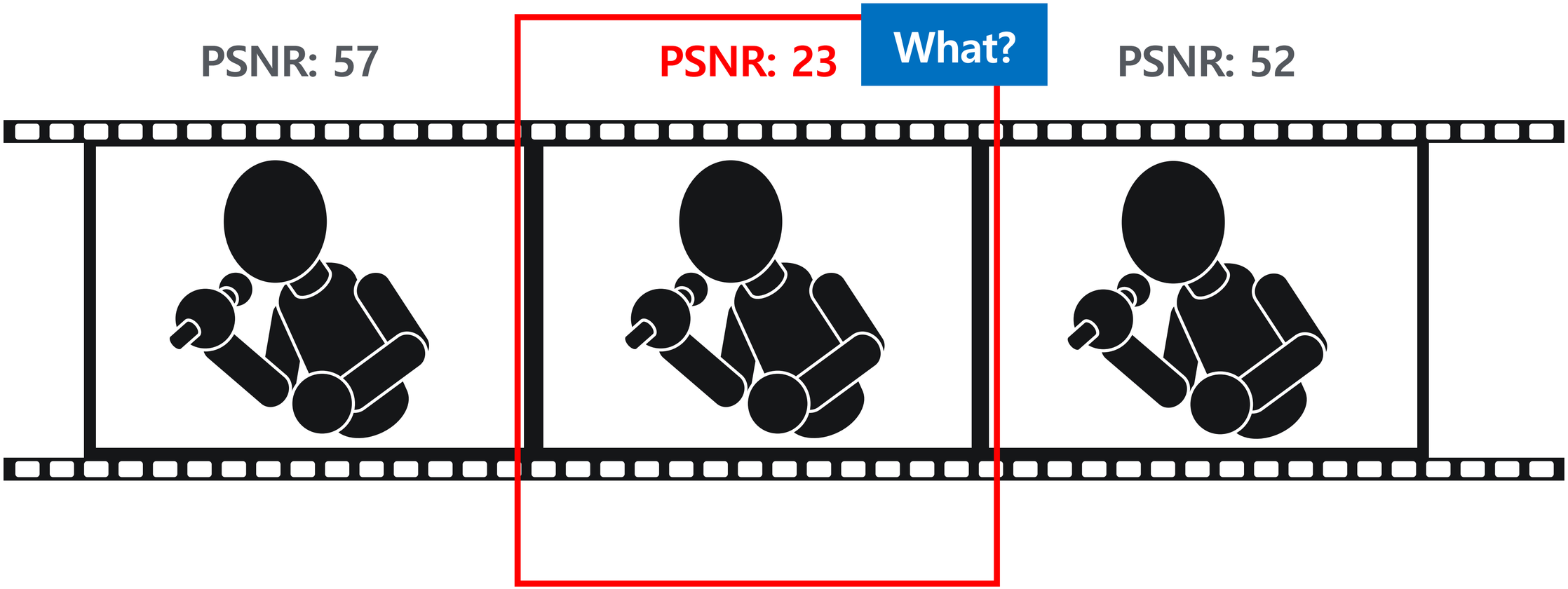
- 하루에도 수천 건의 라이브와 수만 시간의 데이터가 생성



3.3 지표 수집 이후, 또 하나의 고민

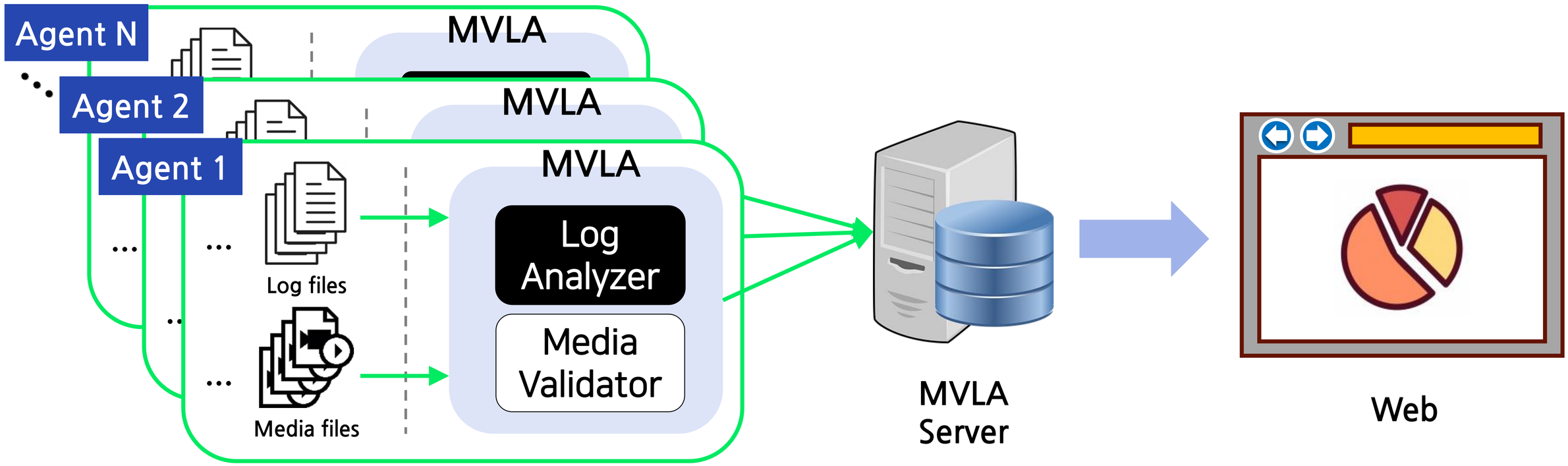
DEVIEW 2019

- 동영상의 품질은 사람이 육안으로 감별해야하는 어려움



3.3 자동화를 시도해보자!

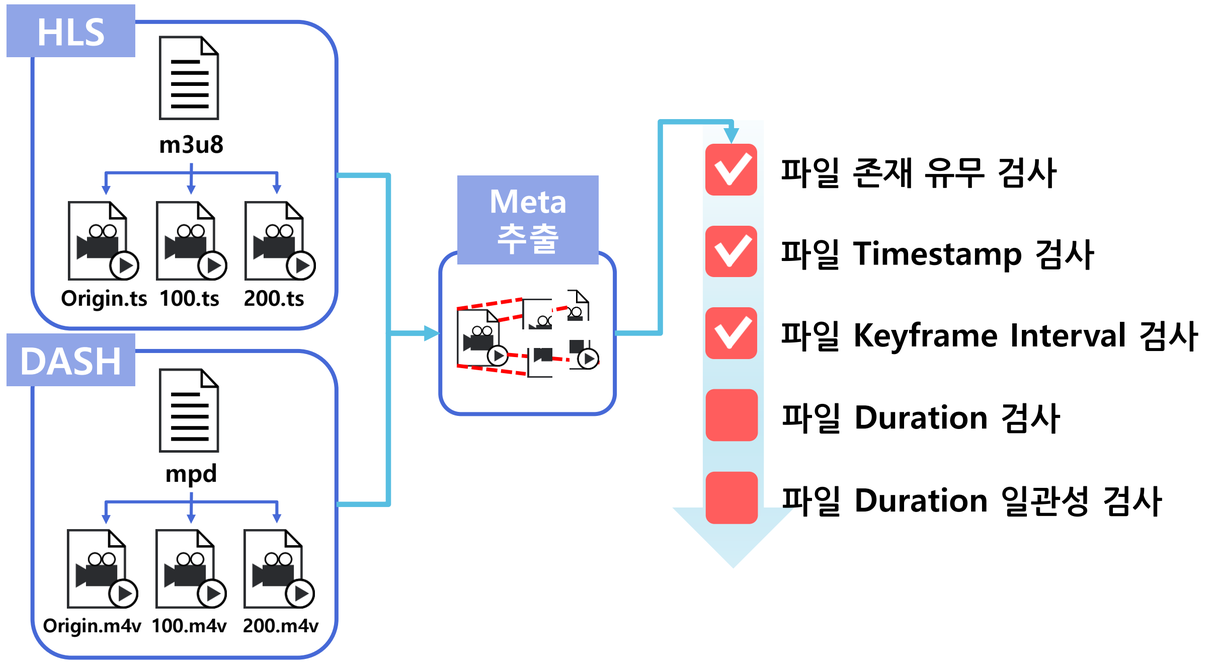
- 영상 데이터들에 대한 프레임 단위의 분석 + 로그 실시간 분석
- 품질 평가를 계량화하려는 시도



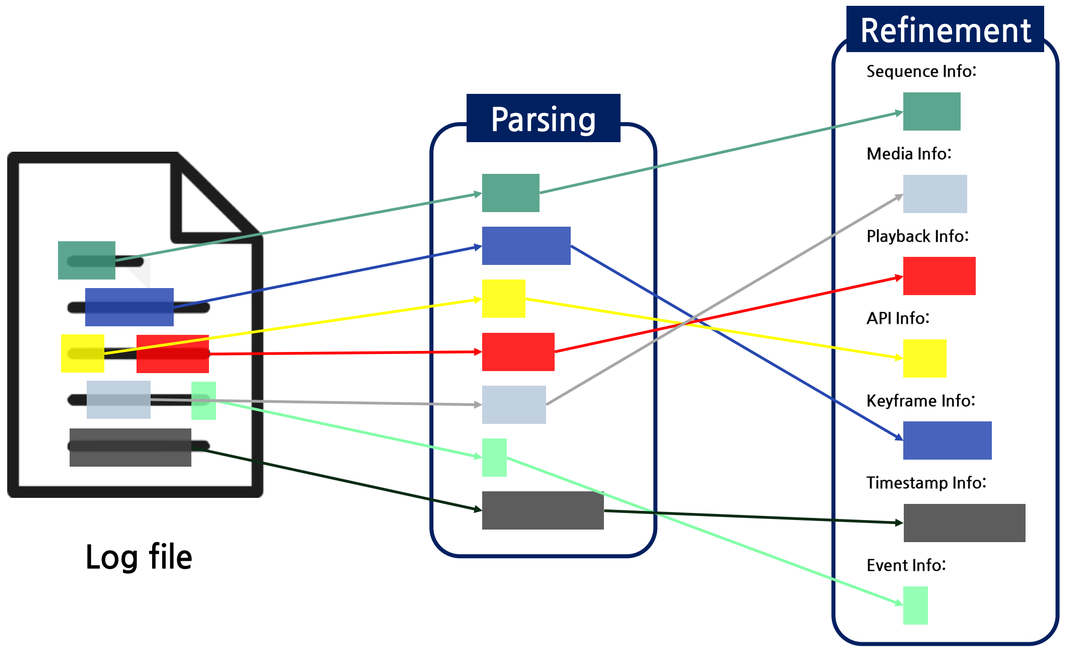
3.3 라이브 방송을 라이브로 분석

DEVIEW 2019

미디어 파일 검증

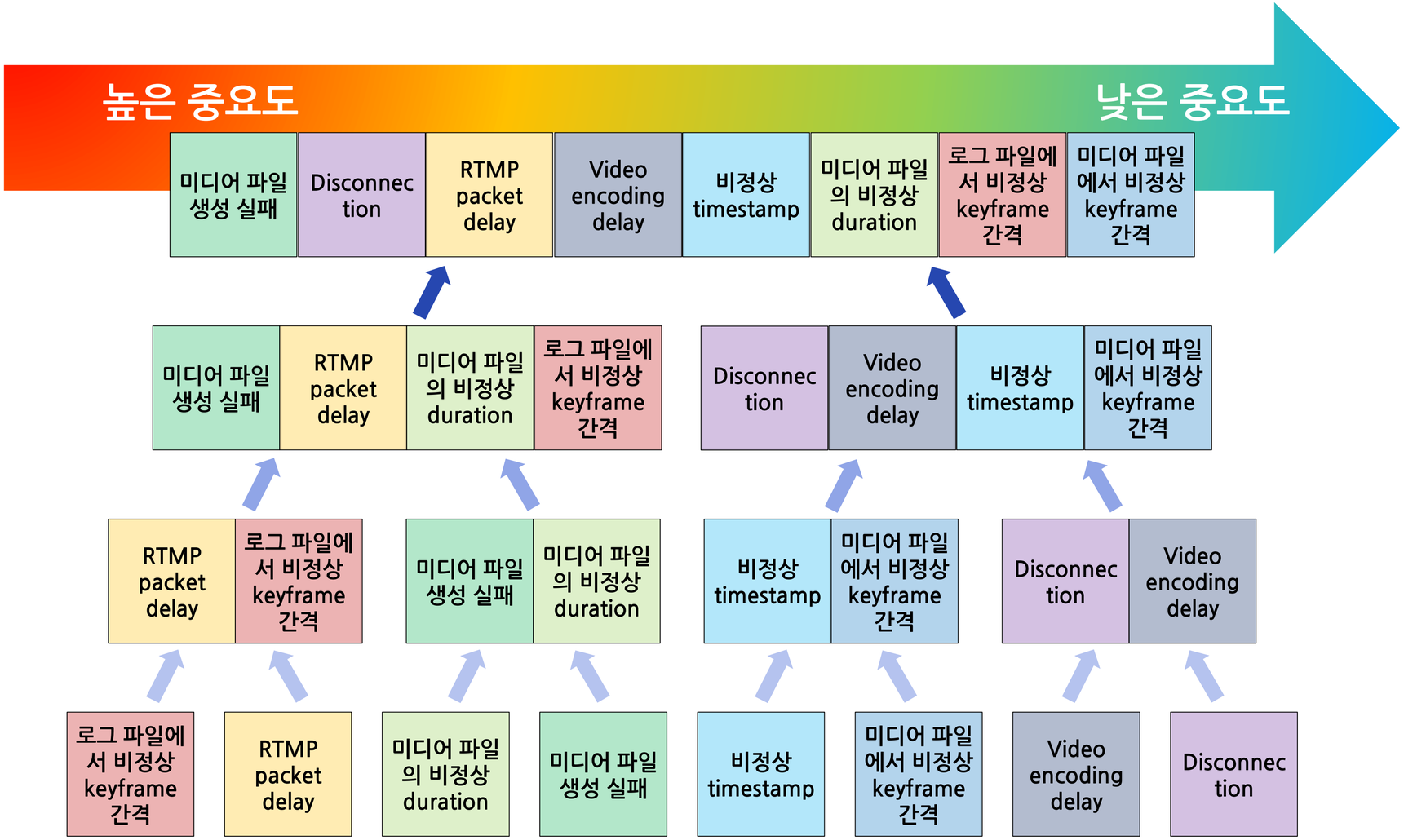


실시간 로그 분석



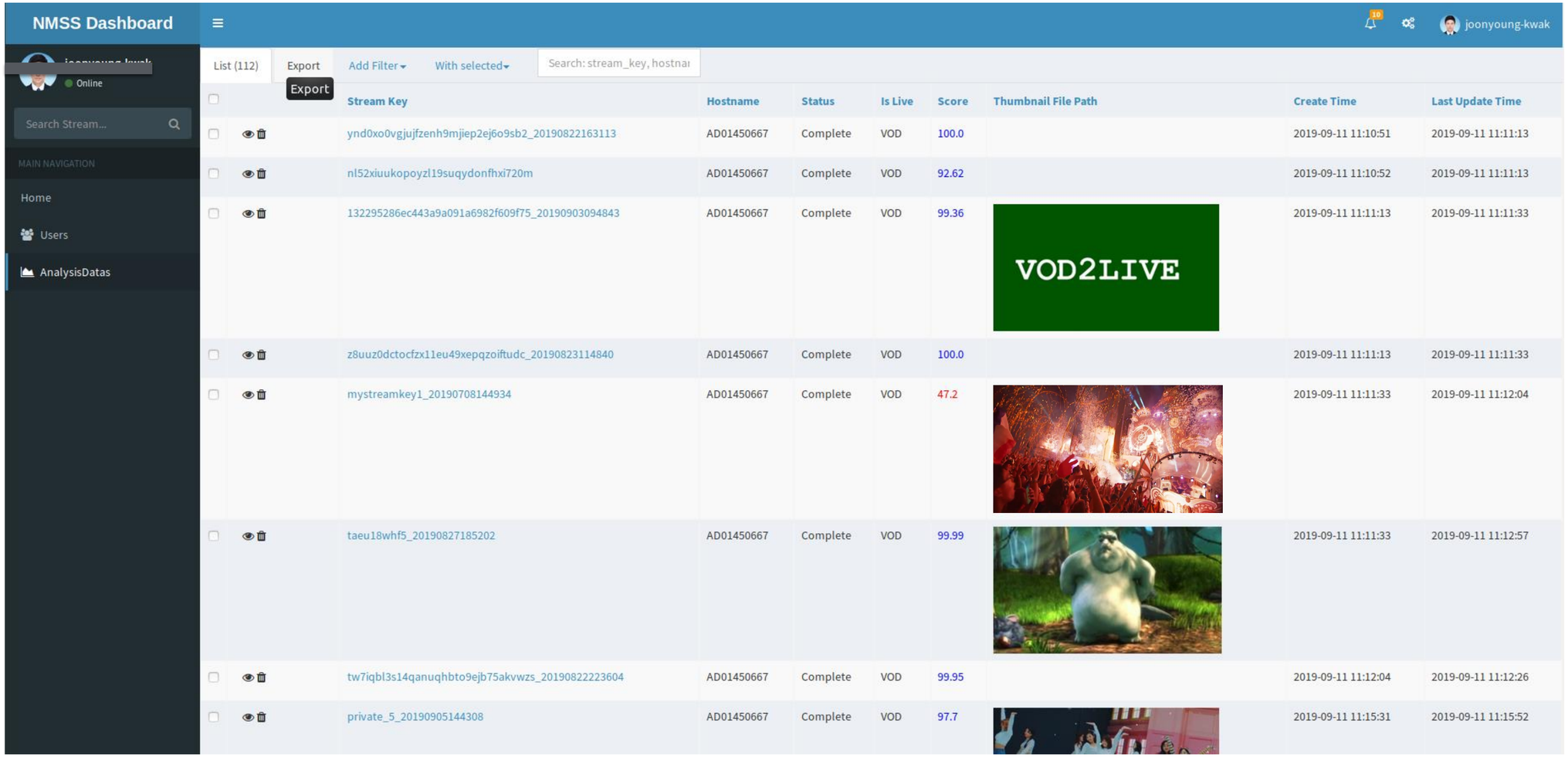
3.3 계량화를 위한 Scoring 가중치 분류

DEVIEW 2019



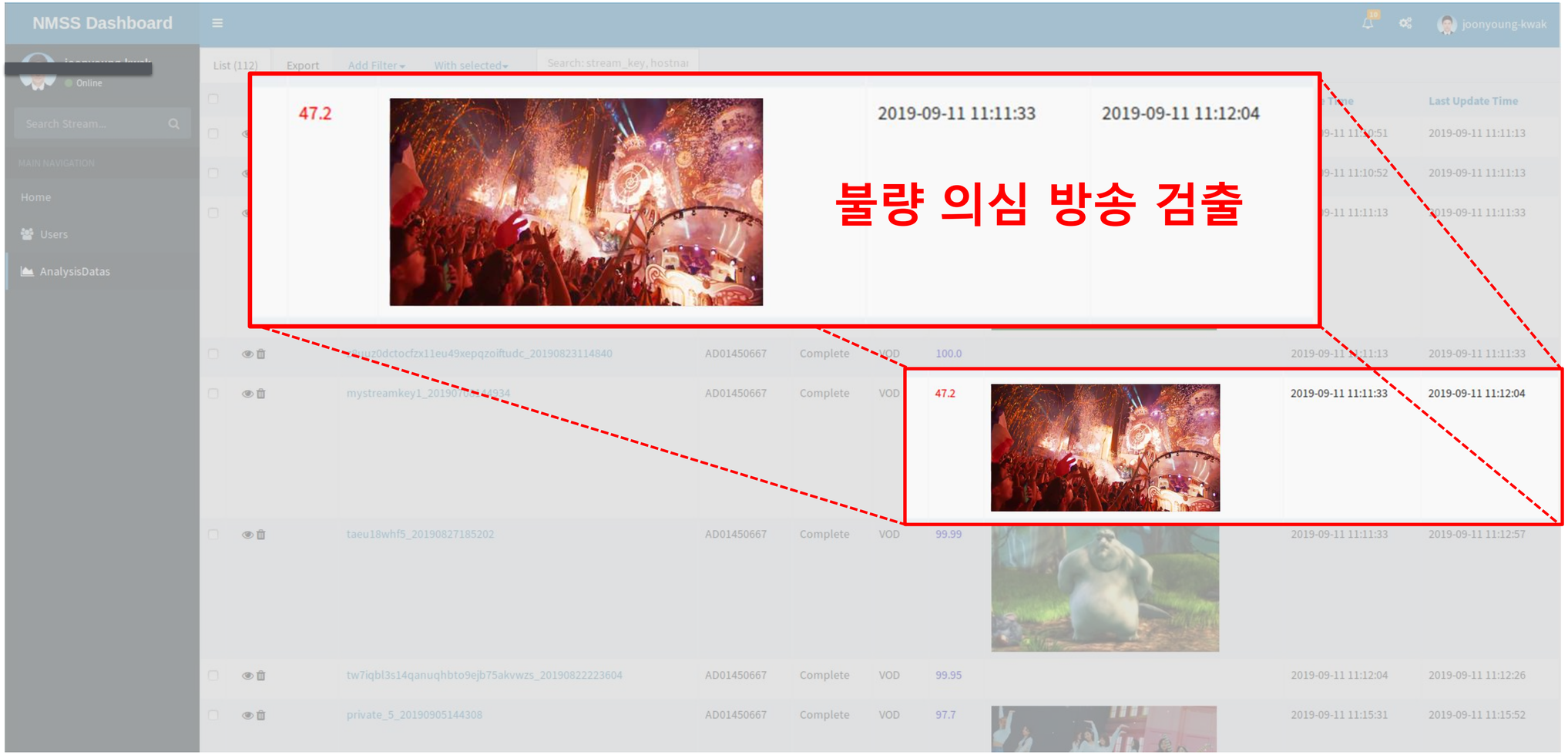
3.3 실시간 품질 모니터링 시스템

DEVIEW 2019



3.3 실시간 품질 모니터링 시스템

DEVIEW 2019



3.3 품질 모니터링 시스템 개발 후

- 인적 리소스의 효율성을 높임
  - **반복 투입**되던 리소스를 **자동화**로 바꿈

- 문제의 지점을 보다 **빠르게 파악**할 수 있음
  - 영상의 문제부터 이상 로그의 지점까지

DEVIEW
2019

4. 네이버의 Next

4. 2020년 키워드

DEVIEW
2019

5G

초저지연/초고해상도

Zero-Latency / 8K

Experience

Immersive

Video / Audio

A.I

실시간 자막 번역

인코딩 최적화

DEVIEW
2019

Appendix

AVX

- https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

- https://www.intel.la/content/www/xl/es/architecture-and-technology/avx-512-animation.html

NUMA

- https://software.intel.com/en-us/articles/optimizing-applications-for-numa
- https://github.com/numactl/numactl
- https://access.redhat.com/documentation/ko-kr/red_hat_enterprise_linux/6/html/performance_tuning_guide/main-cpu#s-cpu-tuning

DEVIEW
2019

Thank You