DEVIEW 2020

안드로이드 미디어 프레임워크를 활용한 동영상 플레이어 만들기

박지환 NAVER ETECH

NAVER Entertainment Emerging TECHnology

DEVIEW 2020

NAVER ETECH. 는 미디어의 생산 <-> 클라우드 <-> 소비기술을 연구 개발합니다.
단단한 기반기술력을 바탕으로, 미디어의 Immersive Experience, AI., XR. 기술을
End-to-End로 제공하는 글로벌 미디어 플랫폼을 지향합니다.

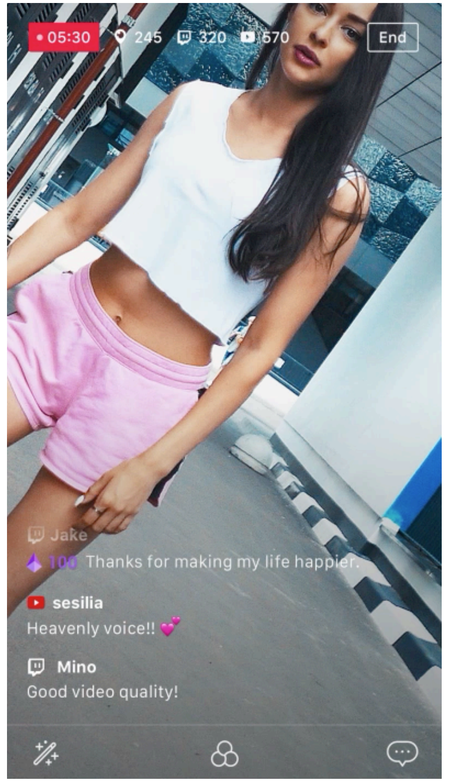
Speaker

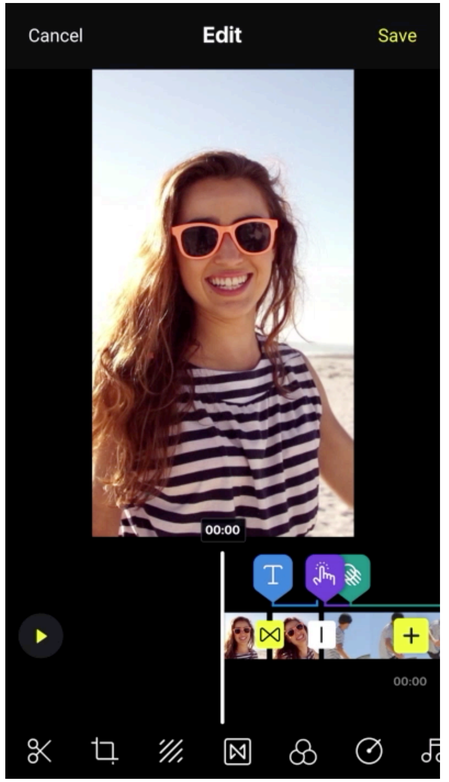
DEVIEW 2020

박지환

NAVER ETECH

PRISM MOBILE
LIVE/VOD SDK





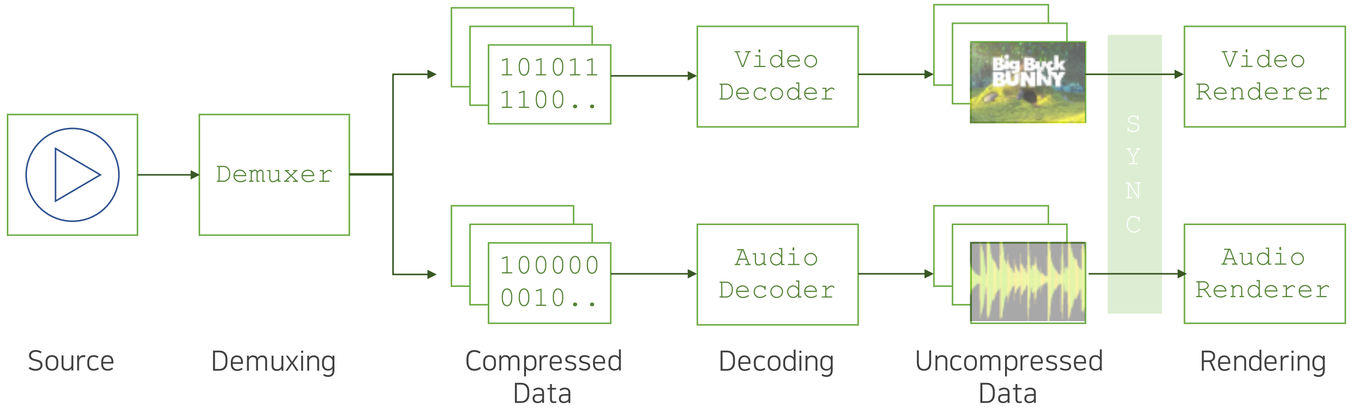
CONTENTS

Intro

왜 플레이어인가?

동영상 편집기의 가장 **기본적인 기능**

Pipeline Architecture



이 발표에서는 **플레이어의 동작 원리를 이해**하기 위해, 밑바닥부터 직접 만들어 봅니다.

Today

다루는 것들

동영상 플레이어의 동작 원리

각 구성 요소의 동작 방식

다루지 않는 것들

API 사용법

코드 설명 (line by line)

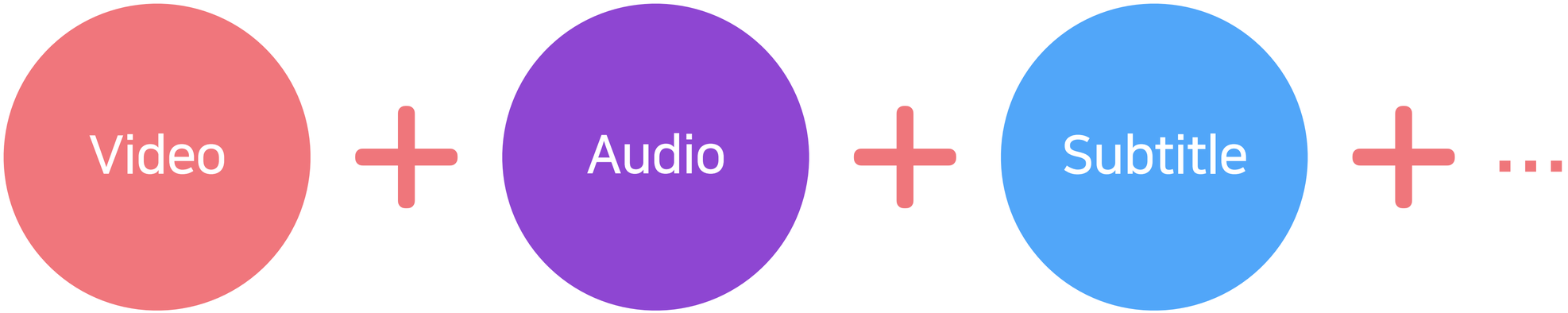
안드로이드 앱 개발

1.동영상이란?

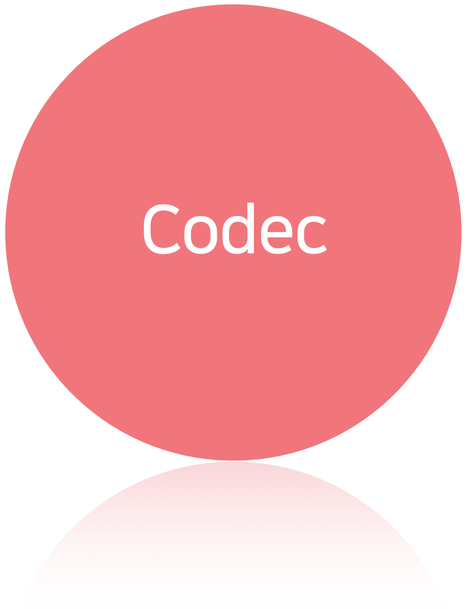
1.1 동영상이란?

출처: https://www.mediacollege.com/video/frame-rate/img/frame-rates.jpg

1.1 동영상이란?



1.1 동영상이란?



1.2 Codec이란?

DEVIEW 2020

Coder + Decoder

"데이터를 스트림이나 신호에 대해, **인코딩 또는 디코딩**,
혹은 둘다를 할 수 있는 하드웨어 또는 소프트웨어 [위키백과]"

"자료를 **압축하고 풀거나, 소리, 동영상 등의 자료를**
다른 형식으로 변환하는 장치 및 소프트웨어 [위키백과]"

Q. 왜 압축을 해야 하는가?

A. 비용 절약

- 압축되지 않은 FHD 해상도 이미지:
  장당 약 5.9 MB → 초당 약 177.9 MB → 분당 약 10.4 GB

- 압축된 FHD 해상도 영상 (8000 Kbps):
  초당 약 1MB

1.2 Codec이란?

DEVIEW 2020

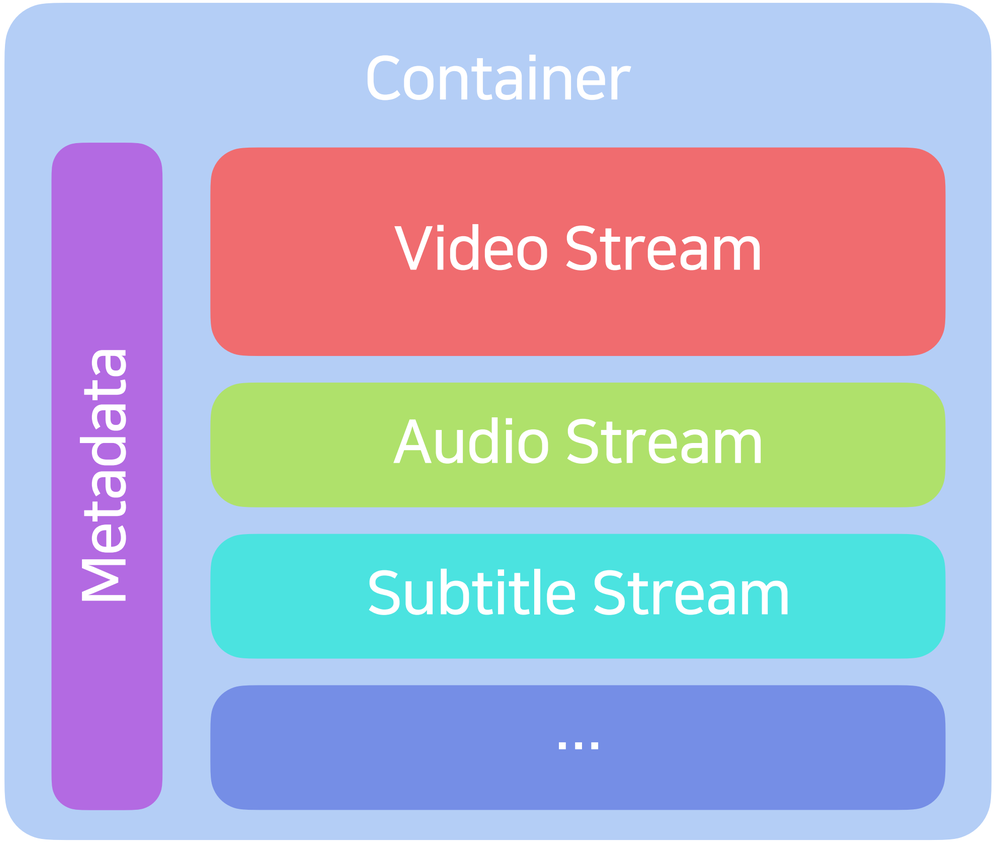
대표적인 비디오 Codec

H.264 H.265 VP9 ...

대표적인 오디오 Codec

MP3 AAC AC3 ...

1.3 Container란?



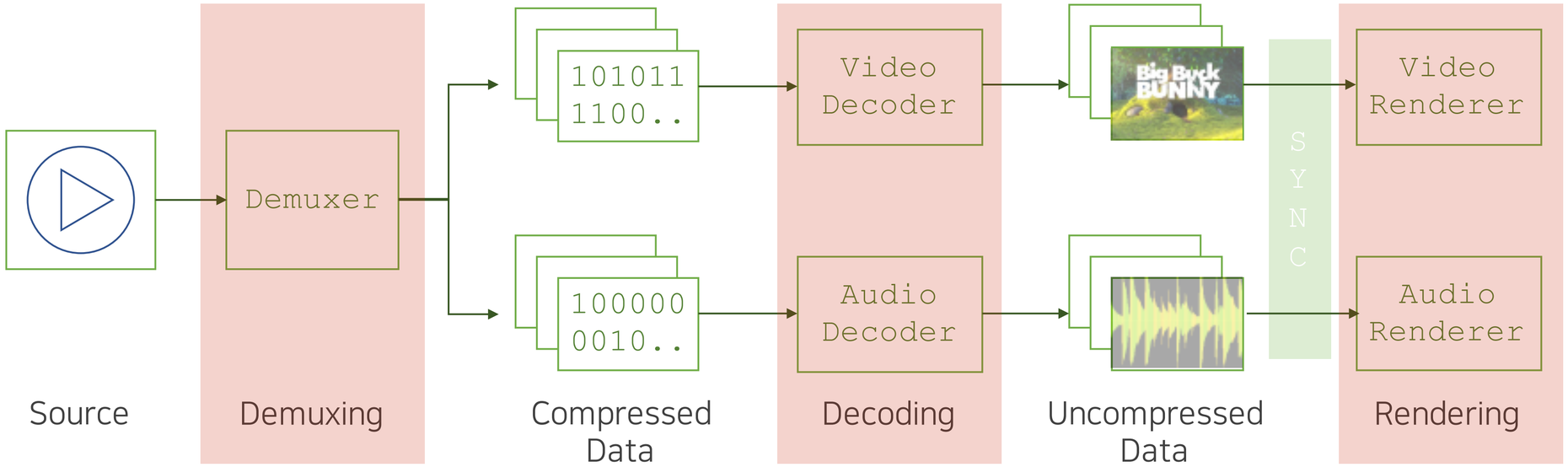
1.3 Container란?

DEVIEW 2020

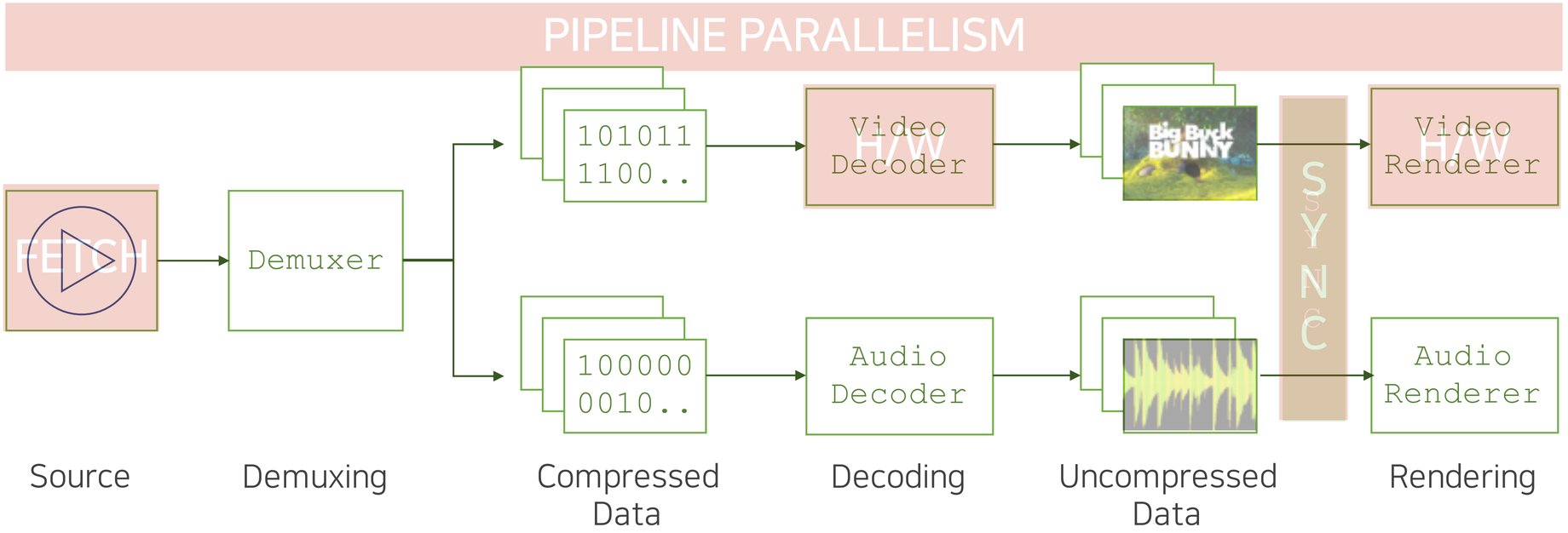
대표적인 Container

AVI　MP4　MKV　…

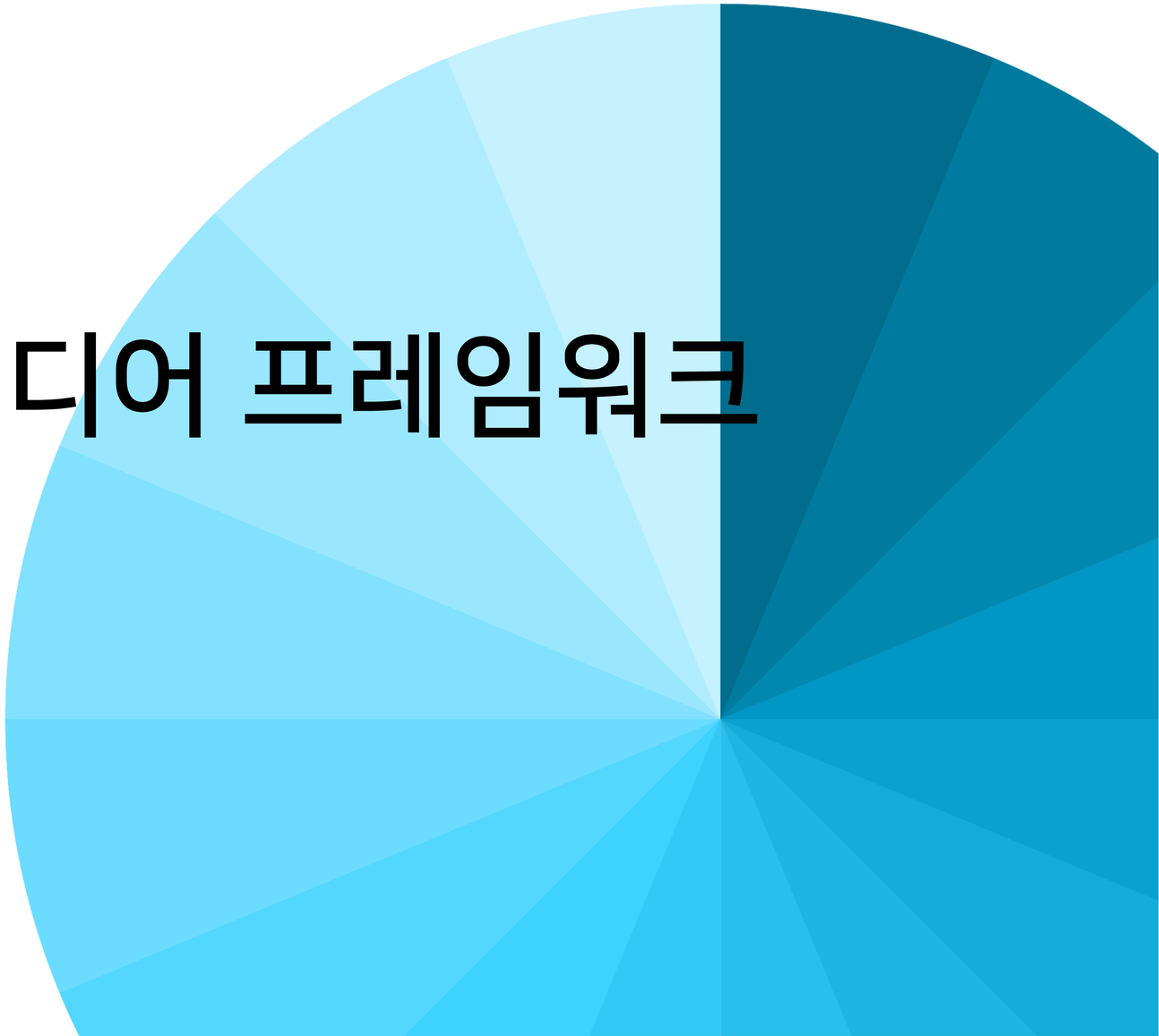
1.4 동영상 플레이어란?


Demuxer
101011
1100..
100000
0010..
Video
Decoder
Audio
Decoder
SYNC
Video
Renderer
Audio
Renderer
Source
Demuxing
Compressed
Data
Decoding
Uncompressed
Data
Rendering

1.4 동영상 플레이어란?


PIPELINE PARALLELISM
FETCH
Demuxer
101011
1100..
100000
0010..
Video
Decoder
H/W
Audio
Decoder
Big Buck
BUNNY
SYNC
Video
Renderer
H/W
Audio
Renderer
Source
Demuxing
Compressed
Data
Decoding
Uncompressed
Data
Rendering

2.안드로이드 미디어 프레임워크

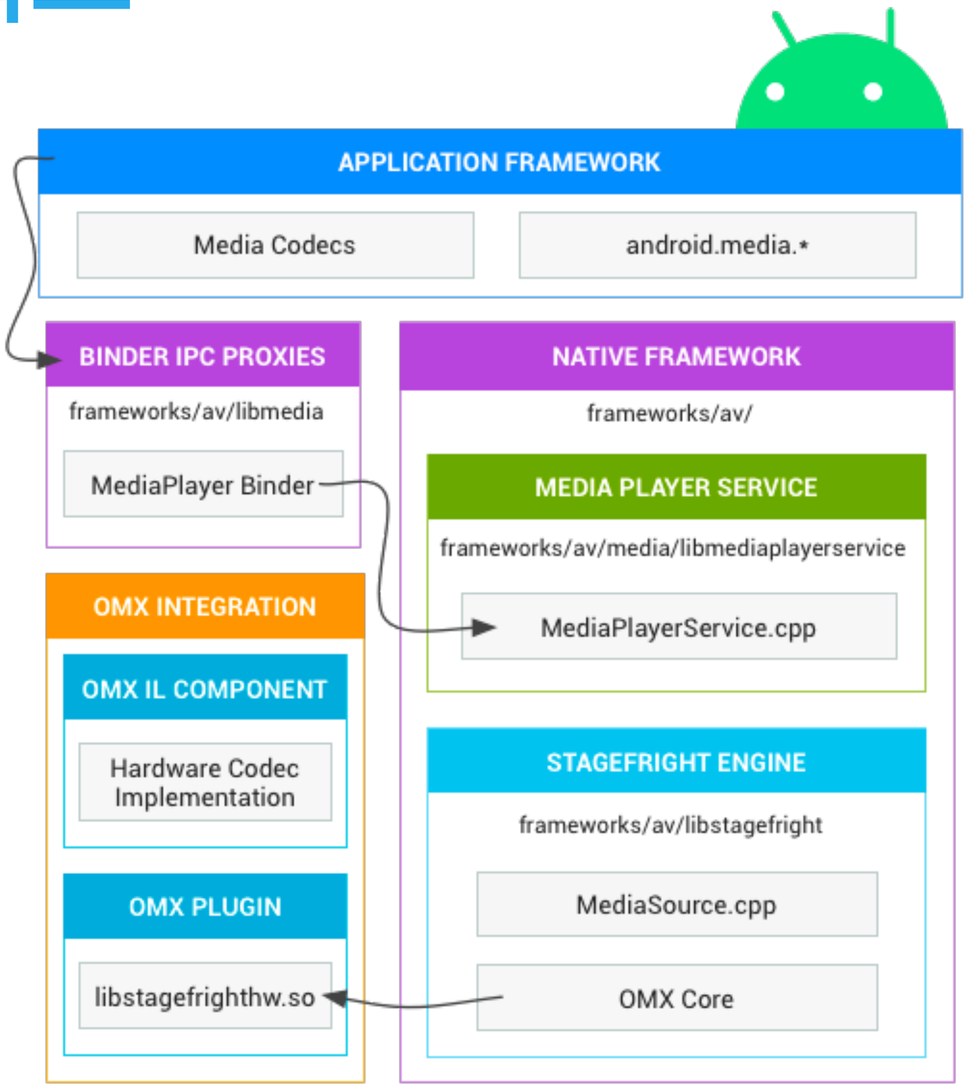


2 안드로이드 미디어 프레임워크

DEVIEW 2020

멀티미디어에 대한 다양한 API를 제공

MediaPlayer
MediaRecorder
MediaExtractor
MediaMuxer
MediaCodec
etc.



3.동영상 플레이어 만들기

3 동영상 플레이어 만들기

DEVIEW 2020

https://github.com/jeehwan/MediaPlayerWithAndroidMediaFramework

3 동영상 플레이어 만들기

DEVIEW 2020

용어 정의

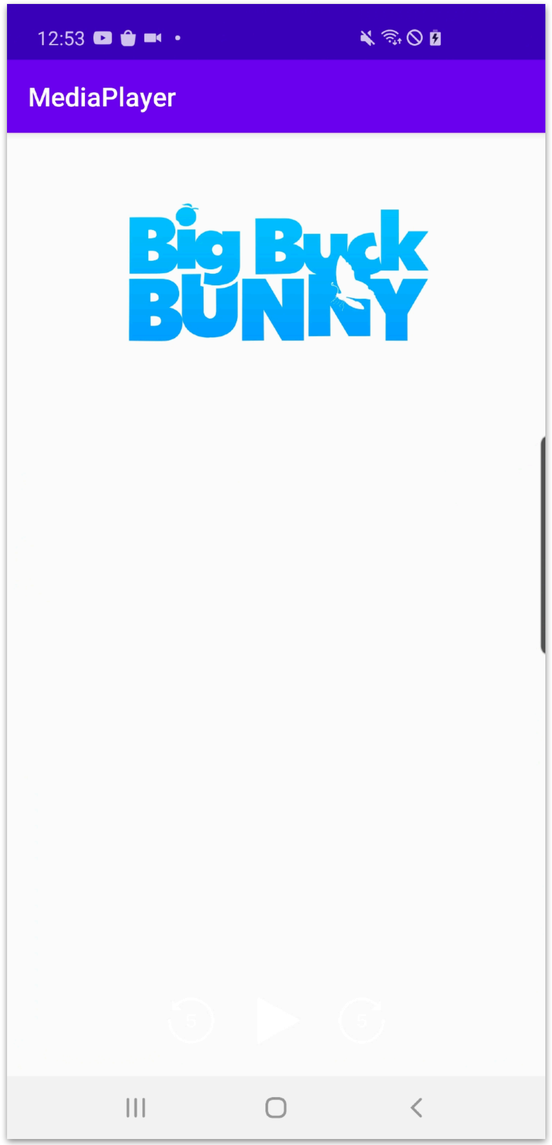
Sample: 디코딩하기 전의 영상 및 사운드 데이터

Frame: 디코딩한 후의 영상 및 사운드 데이터

3.1 화면에 출력하기

DEVIEW 2020

```
Video
Demuxing &
Decoding &
Rendering
```



3.1 화면에 출력하기

```
private val TIMEOUT_US = 10_000L
private val MEDIA_FILE = "somewhere_in_your_sdcard"

private fun createVideoThread(surface: Surface) = thread {
    val extractor = MediaExtractor().apply {
      setDataSource(MEDIA_FILE)
    }

    val trackIndex = extractor.firstVideoTrack
        ?: error("This media file doesn't contain any video tracks")

    extractor.selectTrack(trackIndex)

    val format = extractor.getTrackFormat(trackIndex)
    val mime = format.getString(MediaFormat.KEY_MIME)
        ?: error("Video track must have the mime type")

    val decoder = MediaCodec.createDecoderByType(mime).apply {
        configure(format, surface, null, 0)
        start()
    }

    doExtract(extractor, decoder)

    decoder.stop()
    decoder.release()
    extractor.release()
}
```

3.1 화면에 출력하기

```
private val TIMEOUT_US = 10_000L
private val MEDIA_FILE = "somewhere_in_your_sdcard"

private fun createVideoThread(surface: Surface) = thread {
    val extractor = MediaExtractor().apply {
      setDataSource(MEDIA_FILE)
    }

    val trackIndex = extractor.firstVideoTrack
        ?: error("This media file doesn't contain any video tracks")

    extractor.selectTrack(trackIndex)

    val format = extractor.getTrackFormat(trackIndex)
    val mime = format.getString(MediaFormat.KEY_MIME)
        ?: error("Video track must have the mime type")

    val decoder = MediaCodec.createDecoderByType(mime).apply {
        configure(format, surface, null, 0)
        start()
    }

    doExtract(extractor, decoder)

    decoder.stop()
    decoder.release()
    extractor.release()
}
```

새로운 thread 생성

3.1 화면에 출력하기

```
private val TIMEOUT_US = 10_000L
private val MEDIA_FILE = "somewhere_in_your_sdcard"

private fun createVideoThread(surface: Surface) = thread {
    val extractor = MediaExtractor().apply {
      setDataSource(MEDIA_FILE)
    }

    val trackIndex = extractor.firstVideoTrack
        ?: error("This media file doesn't contain any video tracks")

    extractor.selectTrack(trackIndex)

    val format = extractor.getTrackFormat(trackIndex)
    val mime = format.getString(MediaFormat.KEY_MIME)
        ?: error("Video track must have the mime type")

    val decoder = MediaCodec.createDecoderByType(mime).apply {
        configure(format, surface, null, 0)
        start()
    }

    doExtract(extractor, decoder)

    decoder.stop()
    decoder.release()
    extractor.release()
}
```

MediaExtractor, MediaCodec
생성 및 준비

3.1 화면에 출력하기

```
private val TIMEOUT_US = 10_000L
private val MEDIA_FILE = "somewhere_in_your_sdcard"

private fun createVideoThread(surface: Surface) = thread {
    val extractor = MediaExtractor().apply {
      setDataSource(MEDIA_FILE)
    }

    val trackIndex = extractor.firstVideoTrack
        ?: error("This media file doesn't contain any video tracks")

    extractor.selectTrack(trackIndex)

    val format = extractor.getTrackFormat(trackIndex)
    val mime = format.getString(MediaFormat.KEY_MIME)
        ?: error("Video track must have the mime type")

    val decoder = MediaCodec.createDecoderByType(mime).apply {
        configure(format, surface, null, 0)
        start()
    }

    doExtract(extractor, decoder)

    decoder.stop()
    decoder.release()
    extractor.release()
}
```

Demuxing & Decoding

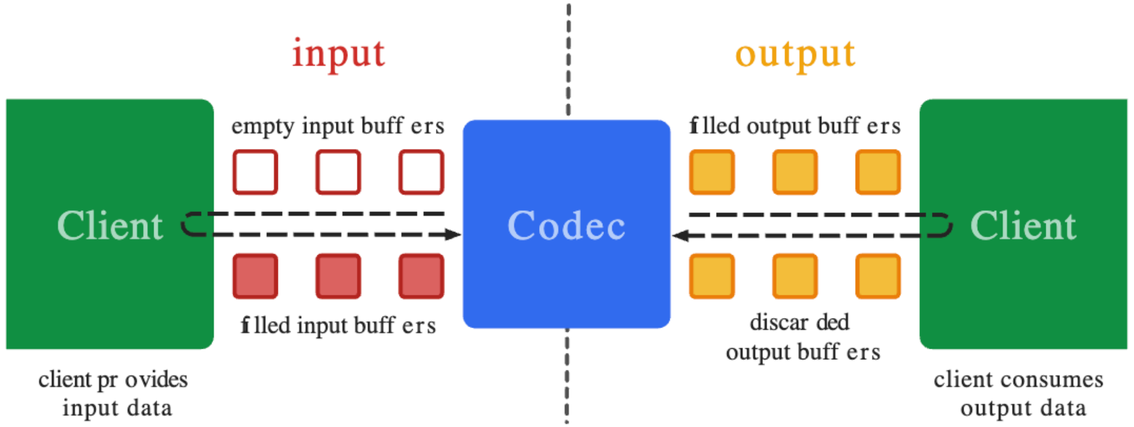
3.1 화면에 출력하기

```
private val TIMEOUT_US = 10_000L
private val MEDIA_FILE = "somewhere_in_your_sdcard"

private fun createVideoThread(surface: Surface) = thread {
    val extractor = MediaExtractor().apply {
      setDataSource(MEDIA_FILE)
    }

    val trackIndex = extractor.firstVideoTrack
        ?: error("This media file doesn't contain any video tracks")

    extractor.selectTrack(trackIndex)

    val format = extractor.getTrackFormat(trackIndex)
    val mime = format.getString(MediaFormat.KEY_MIME)
        ?: error("Video track must have the mime type")

    val decoder = MediaCodec.createDecoderByType(mime).apply {
        configure(format, surface, null, 0)
        start()
    }

    doExtract(extractor, decoder)

    decoder.stop()
    decoder.release()
    extractor.release()
}
```

MediaExtractor, MediaCodec 제거

```
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                          "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

3.1 화면에 출력하기

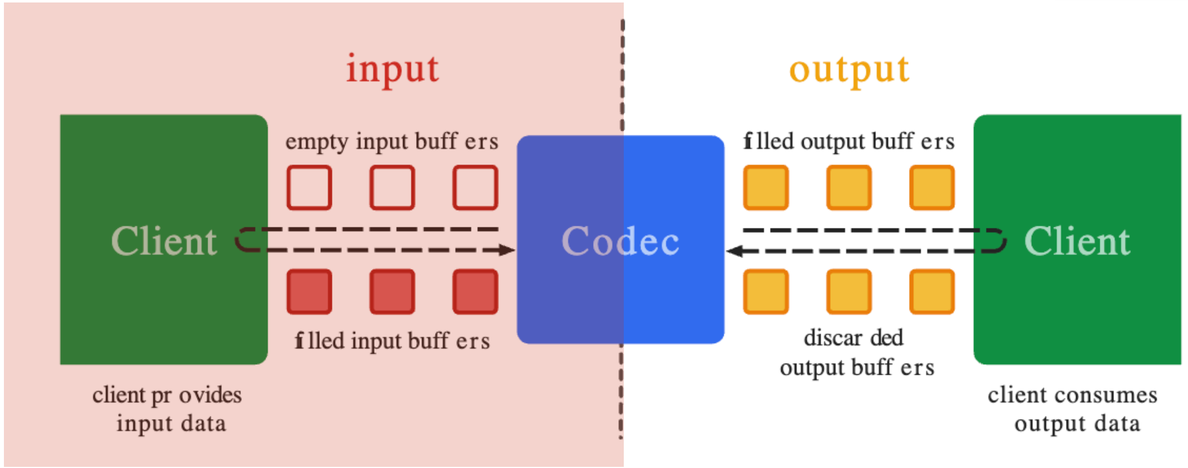


출처: https://developer.android.com/reference/android/media/MediaCodec

```
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                        "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

3.1 화면에 출력하기

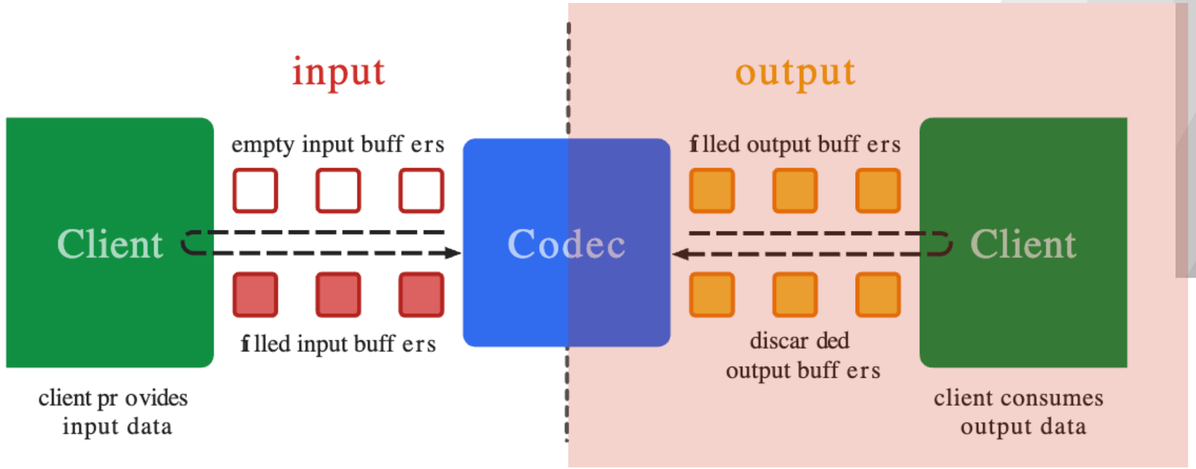
Decoder 입력파트



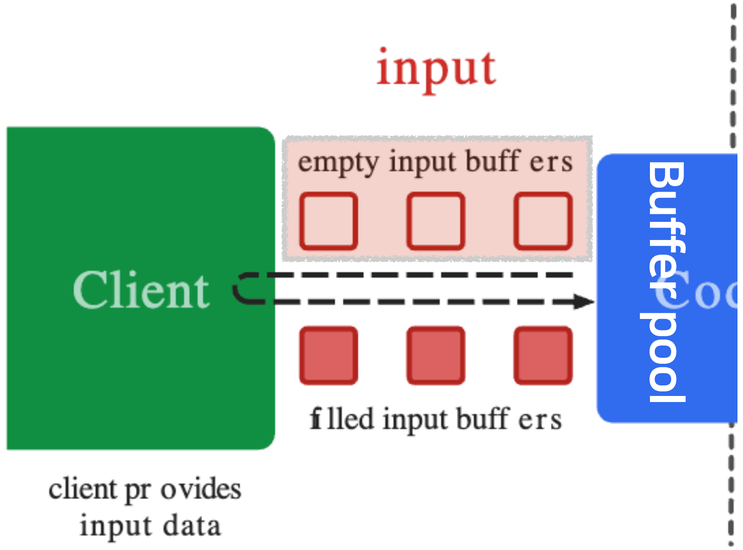
```
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                          "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

3.1 화면에 출력하기

Decoder 출력파트

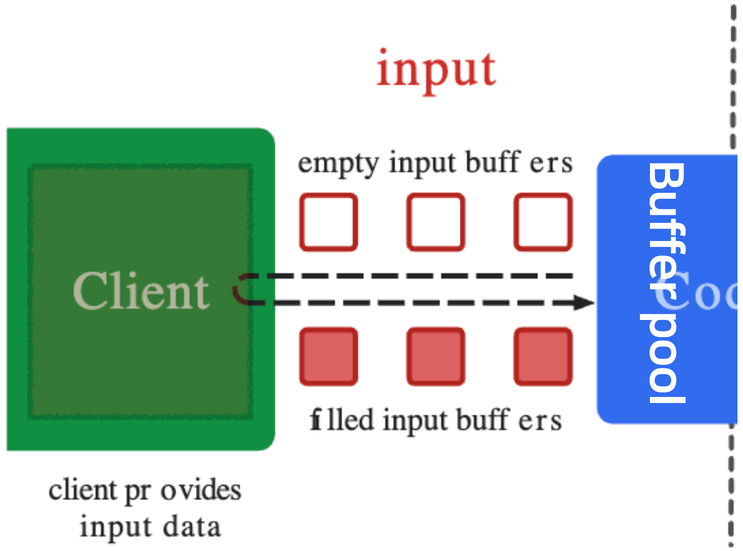


3.1 화면에 출력하기



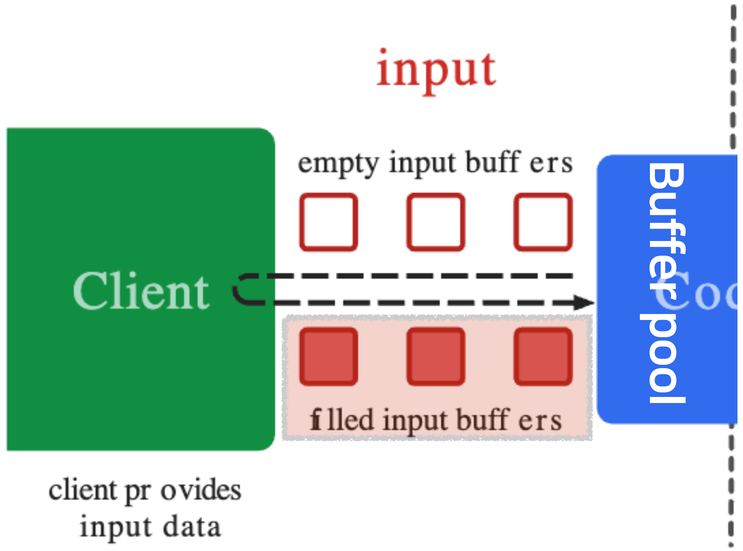

```
when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
    in 0..Int.MAX_VALUE -> {
        val inputBuffer = decoder.getInputBuffer(inputIndex)!!
```

3.1 화면에 출력하기



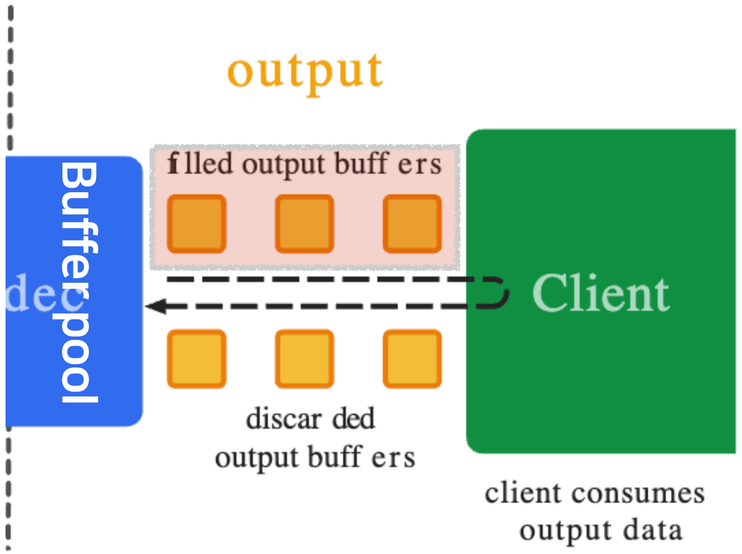

```
val chunkSize = extractor.readSampleData(inputBuffer, 0)
```

3.1 화면에 출력하기



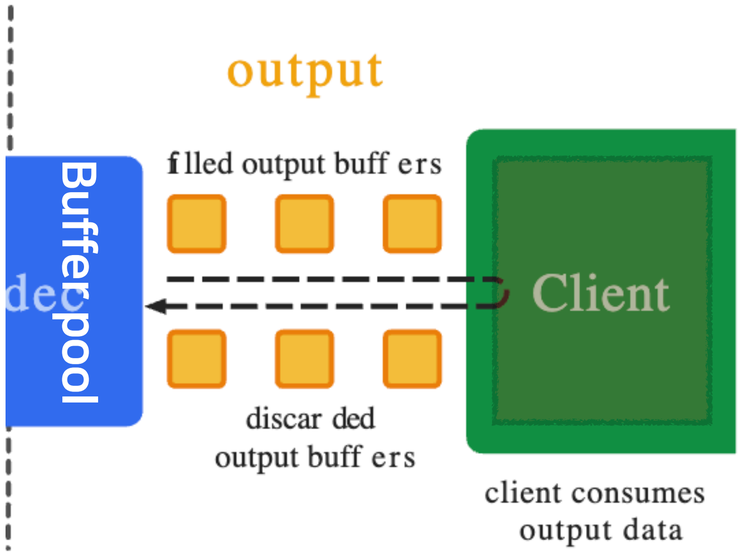

```
if (chunkSize < 0) {
    decoder.queueInputBuffer(inputIndex, 0, 0, -1,
        MediaCodec.BUFFER_FLAG_END_OF_STREAM)
    inEos = true
} else {
    val sampleTimeUs = extractor.sampleTime
    decoder.queueInputBuffer(inputIndex, 0, chunkSize,
        sampleTimeUs, 0)
    extractor.advance()
}
```

3.1 화면에 출력하기




```
when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
    in 0..Int.MAX_VALUE -> {
```

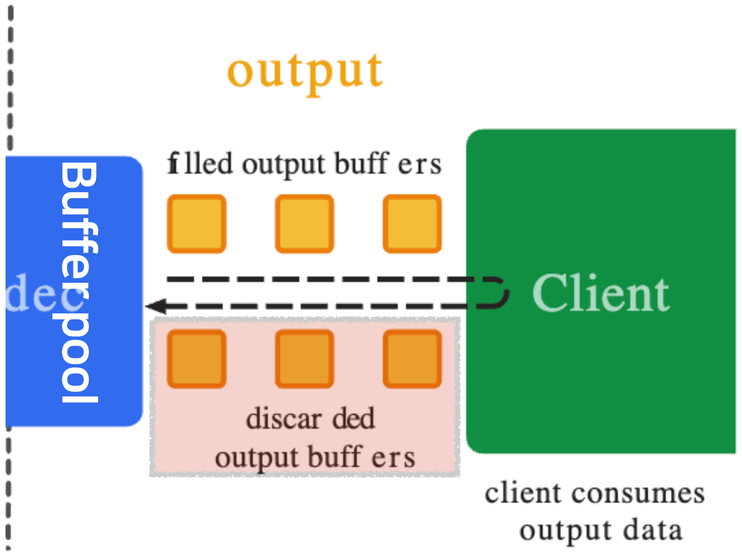
3.1 화면에 출력하기




```
in 0..Int.MAX_VALUE -> {
    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
        decoder.releaseOutputBuffer(outputIndex, false)
        outEos = true
    } else {
        decoder.releaseOutputBuffer(outputIndex, true)
    }
}
```

consumes output data

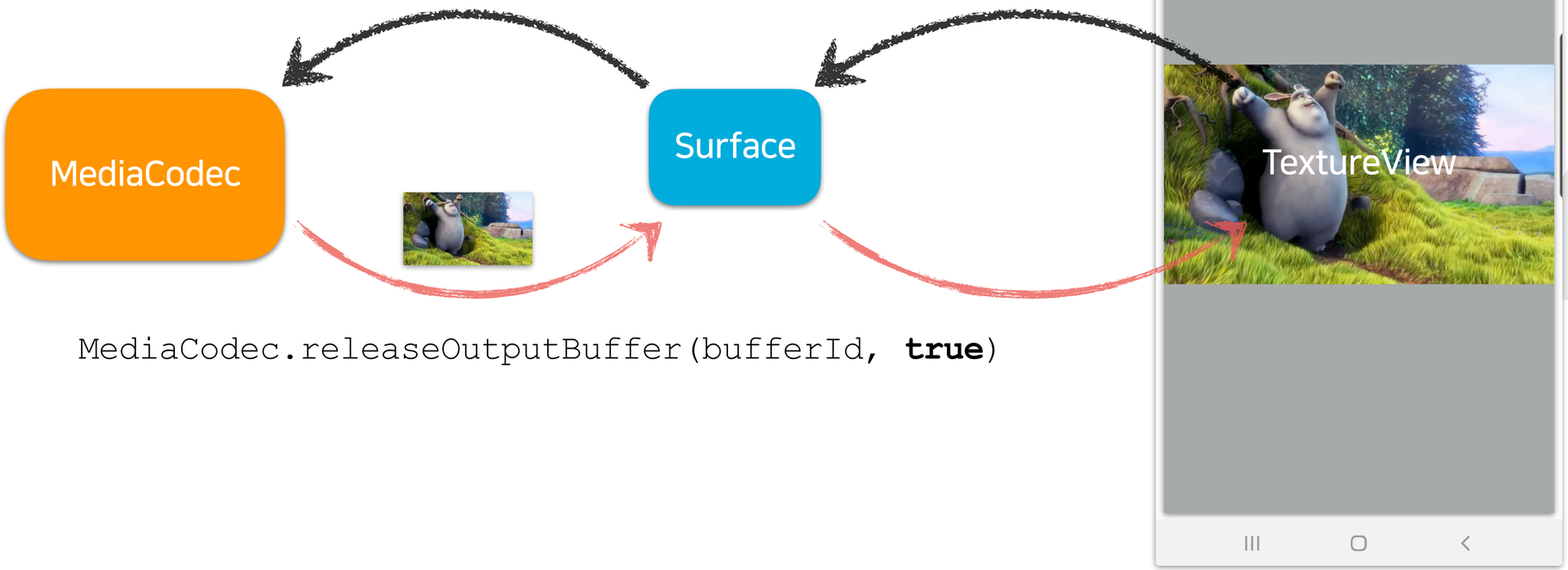
3.1 화면에 출력하기




```
if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
    decoder.releaseOutputBuffer(outputIndex, false)
    outEos = true
} else {
    decoder.releaseOutputBuffer(outputIndex, true)
}
```

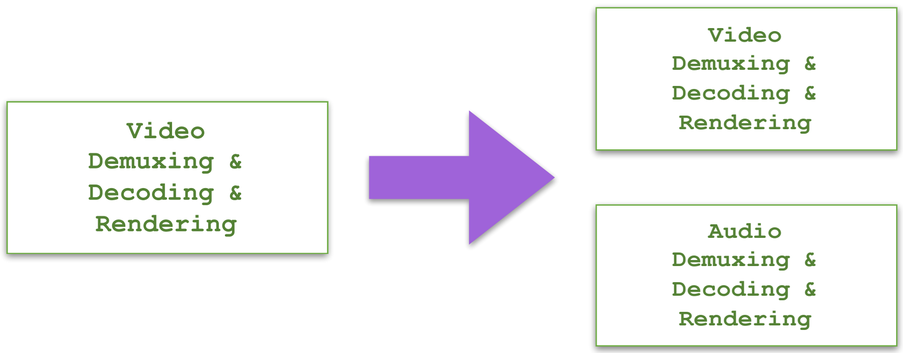
3.1 화면에 출력하기

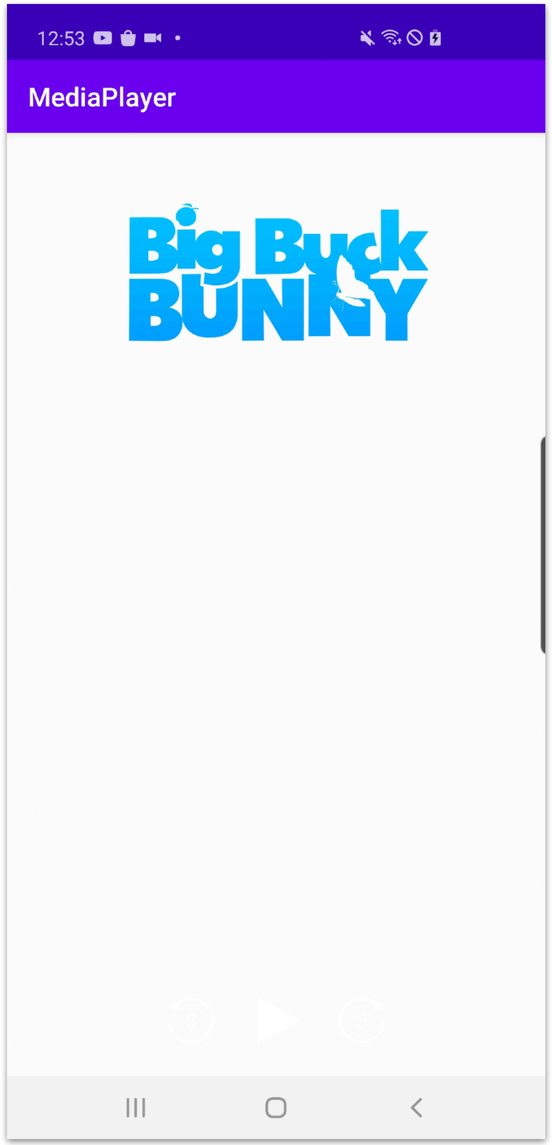
```
MediaCodec.configure(..., surface, ...)
```




```
MediaCodec.releaseOutputBuffer(bufferId, true)
```

3.2 소리 재생하기


Video
Demuxing &
Decoding &
Rendering
Video
Demuxing &
Decoding &
Rendering
Audio
Demuxing &
Decoding &
Rendering

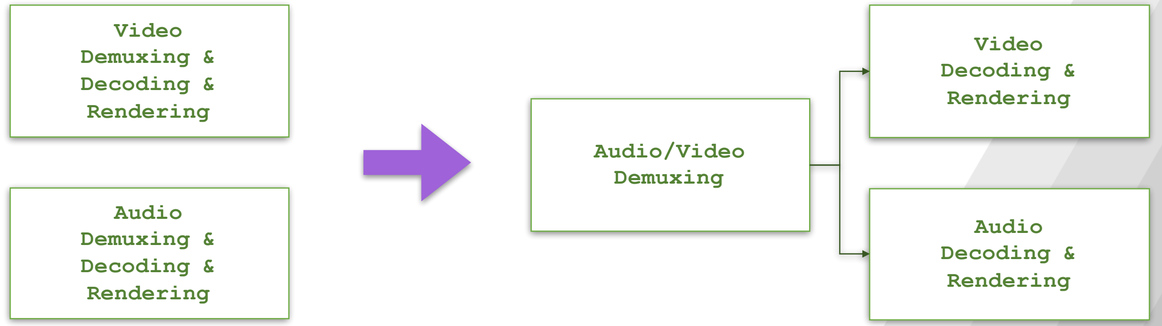

12:53
MediaPlayer
Big Buck
BUNNY

3.2 소리 재생하기

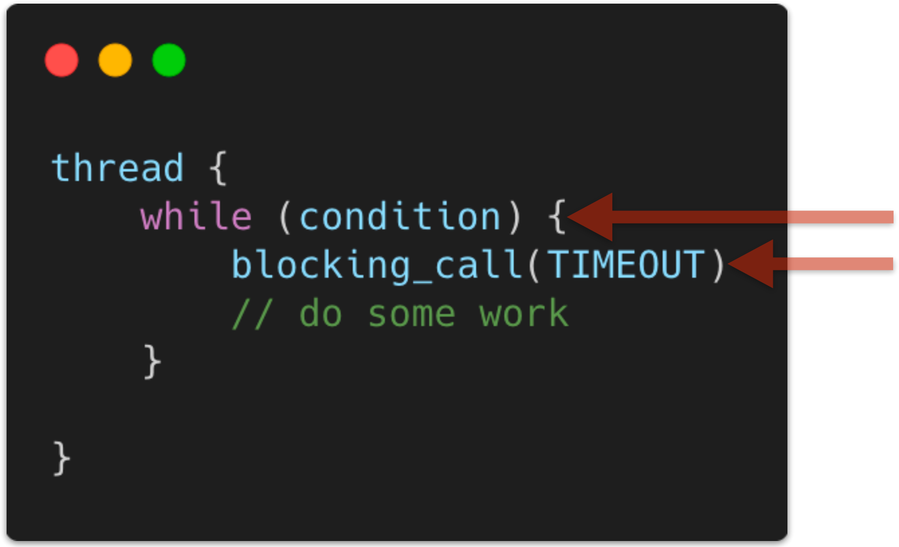
```
when (val outputIndex = audioDecoder.dequeueOutputBuffer(audioBufferInfo, 0)) {
    in 0..Int.MAX_VALUE -> {
        // 생략...

        val outputBuffer = audioDecoder.getOutputBuffer(outputIndex)!!
        outputBuffer.position(audioBufferInfo.offset)
        outputBuffer.limit(audioBufferInfo.offset + audioBufferInfo.size)

        audioTrack.write(outputBuffer, audioBufferInfo.size,
            AudioTrack.WRITE_BLOCKING)

        audioDecoder.releaseOutputBuffer(outputIndex, false)

        // 생략...
    }
}
```

```
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                          "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

3.2 소리 재생하기

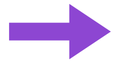


3.2 소리 재생하기



3.2 소리 재생하기

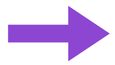
```
thread {
    while (condition) {
        blocking_call(TIMEOUT)
        // do some work
    }

}
```



```
thread {
    while (condition) {
        val ready = nonblocking_call()
        if (ready) {
            // do some work
        } else {
            sleep(TIMEOUT)
        }
    }
}
```

3.2 소리 재생하기

```
thread {
    while (condition) {
        val ready = nonblocking_call()
        if (ready) {
            // do some work
        } else {
            sleep(TIMEOUT)
        }
    }
}
```



```
class Scheduler {
    fun schedule(block: () -> Unit, delay: Long)
}


scheduler.schedule(task@{
    val ready = nonblocking_call()
    if (ready) {
        // do some work

        scheduler.schedule(task, 0)
    } else {
        scheduler.schedule(task, TIMEOUT)
    }
}, 0)
```

*설명을 위한것으로, 문법적으로 정확하지 않음

3.2 소리 재생하기

```
val task = object : Runnable {
    @Override
    fun run() {
        val ret = nonblocking_call()
        if (ret) {
            // do some work

            scheduler.schedule(this, 0)
        } else {
            scheduler.schedule(this, TIMEOUT)
        }
    }
}

scheduler.schedule(task, 0)
```

```
fun postTaskRepeatedly(delay: Long) {
    scheduler.schedule({
        val ret = nonblocking_call()
        if (ret) {
            // do some work

            postTaskRepeatedly(0)
        } else {
            postTaskRepeatedly(TIMEOUT)
        }
    }, delay)
}

postTaskRepeatedly(0)
```

```kotlin
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                          "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

```kotlin
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                        "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

DEVIEW 2020

```kotlin
private fun postExtractVideo(delayMillis: Long) {
    demuxHandler.postDelayed({
        if (!videoInEos) {
            when (val inputIndex = videoDecoder.dequeueInputBuffer(0)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = videoDecoder.getInputBuffer(inputIndex)!!
                    val chunkSize = videoExtractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        videoDecoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        videoInEos = true
                    } else {
                        val sampleTimeUs = videoExtractor.sampleTime
                        videoDecoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        videoExtractor.advance()
                    }

                    postExtractVideo(0)
                }
                else -> postExtractVideo(TIMEOUT_MS)
            }
        }
    }, delayMillis)
}
```

```kotlin
private fun doExtract(extractor: MediaExtractor, decoder: MediaCodec) {
    val info = MediaCodec.BufferInfo()

    var inEos = false
    var outEos = false

    while (!outEos) {
        if (!inEos) {
            when (val inputIndex = decoder.dequeueInputBuffer(TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    val inputBuffer = decoder.getInputBuffer(inputIndex)!!
                    val chunkSize = extractor.readSampleData(inputBuffer, 0)
                    if (chunkSize < 0) {
                        decoder.queueInputBuffer(inputIndex, 0, 0, -1,
                            MediaCodec.BUFFER_FLAG_END_OF_STREAM)
                        inEos = true
                    } else {
                        val sampleTimeUs = extractor.sampleTime
                        decoder.queueInputBuffer(inputIndex, 0, chunkSize,
                            sampleTimeUs, 0)
                        extractor.advance()
                    }
                }
                else -> Unit
            }
        }

        if (!outEos) {
            when (val outputIndex = decoder.dequeueOutputBuffer(info, TIMEOUT_US)) {
                in 0..Int.MAX_VALUE -> {
                    if ((info.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        decoder.releaseOutputBuffer(outputIndex, false)
                        outEos = true
                    } else {
                        decoder.releaseOutputBuffer(outputIndex, true)
                    }
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                          "decoder.dequeueOutputBuffer: $outputIndex")
            }
        }
    }
}
```

```kotlin
private fun postDecodeVideo(delayMillis: Long) {
    videoDecodeHandler.postDelayed({
        if (!videoOutEos) {
            when (val outputIndex = videoDecoder.dequeueOutputBuffer(videoBufferInfo, 0)) {
                in 0..Int.MAX_VALUE -> {
                    if ((videoBufferInfo.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        videoDecoder.releaseOutputBuffer(outputIndex, false)
                        videoOutEos = true
                    } else {
                        videoDecoder.releaseOutputBuffer(outputIndex, true)
                    }

                    postDecodeVideo(0)
                    return@postDelayed
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                        "decoder.dequeueOutputBuffer: $outputIndex")
            }

            postDecodeVideo(TIMEOUT_MS)
        }
    }, delayMillis)
}
```

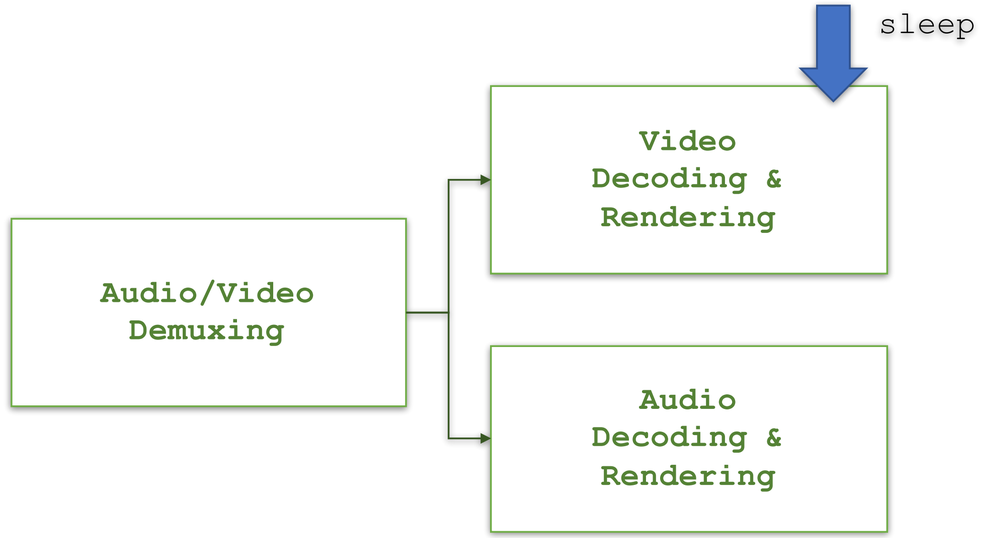
3.3 비디오 속도 맞추기

DEVIEW 2020

문제점. 비디오 재생 속도가 너무 빠름

원인. 비디오 Rendering 속도가 너무 빠름

해결. 비디오 출력 부분에 sleep 추가



3.3 비디오 속도 맞추기

DEVIEW 2020

```
private fun postDecodeVideo(delayMillis: Long) {
    videoDecodeHandler.postDelayed({
        if (!videoOutEos) {
            when (val outputIndex = videoDecoder.dequeueOutputBuffer(videoBufferInfo, 0)) {
                in 0..Int.MAX_VALUE -> {
                    if ((videoBufferInfo.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        videoDecoder.releaseOutputBuffer(outputIndex, false)
                        videoOutEos = true
                    } else {
                        videoDecoder.releaseOutputBuffer(outputIndex, true)
                    }

                    postDecodeVideo(0)
                    return@postDelayed
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                        "decoder.dequeueOutputBuffer: $outputIndex")
            }

            postDecodeVideo(TIMEOUT_MS)
        }
    }, delayMillis)
}
```

3.3 비디오 속도 맞추기

DEVIEW 2020

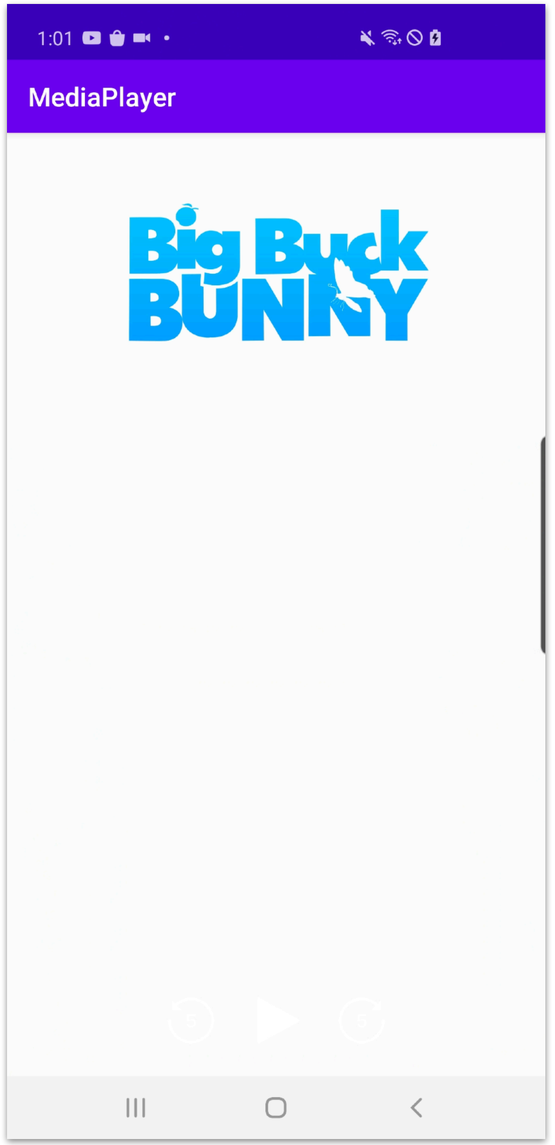
```
private fun postDecodeVideo(delayMillis: Long) {
    videoDecodeHandler.postDelayed({
        if (!videoOutEos) {
            when (val outputIndex = videoDecoder.dequeueOutputBuffer(videoBufferInfo, 0)) {
                in 0..Int.MAX_VALUE -> {
                    if ((videoBufferInfo.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
                        videoDecoder.releaseOutputBuffer(outputIndex, false)
                        videoOutEos = true
                    } else {
                        videoDecoder.releaseOutputBuffer(outputIndex, true)
                    }

                    postDecodeVideo(0)
                    return@postDelayed
                }
                MediaCodec.INFO_TRY_AGAIN_LATER -> Unit
                MediaCodec.INFO_OUTPUT_FORMAT_CHANGED -> Unit
                MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED -> Unit
                else -> error("unexpected result from " +
                        "decoder.dequeueOutputBuffer: $outputIndex")
            }

            postDecodeVideo(TIMEOUT_MS)
        }
    }, delayMillis)
}
```

```
} else {
    val curTimeUs = SystemClock.uptimeMillis() * 1000L
    if (startTimeUs < 0) {
        startTimeUs = curTimeUs
    }
    val curPtsUs = curTimeUs - startTimeUs
    val sleepTimeMs = info.presentationTimeUs - curPtsUs
    if (sleepTimeMs > 0) {
        TimeUnit.MILLISECONDS.sleep(sleepTimeMs)
    } else {
        // TODO
    }

    videoDecoder.releaseOutputBuffer(outputIndex, true)
}
```

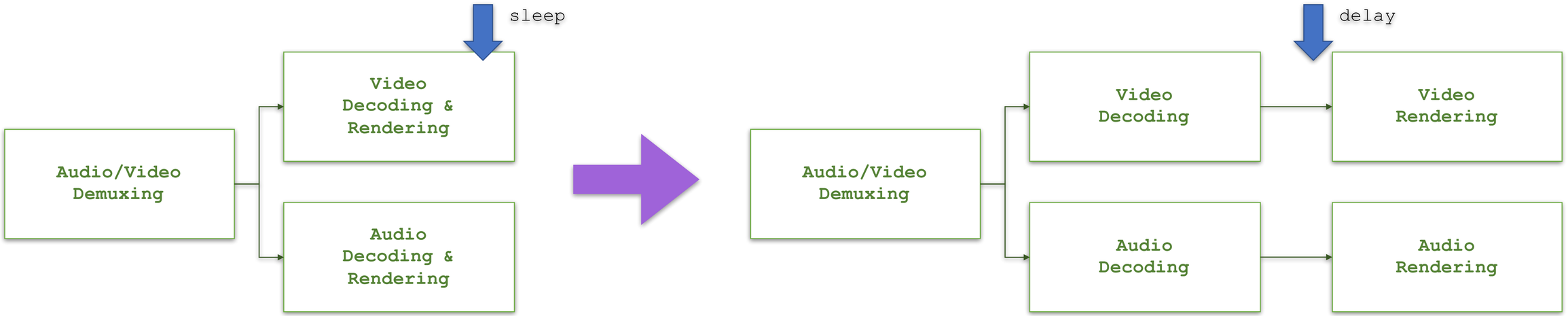
3.3 비디오 속도 맞추기

왜 오디오는 괜찮을까?

- Blocking mode
- Non-blocking mode일 경우 sleep 필요



3.3 비디오 속도 맞추기


sleep
Audio/Video Demuxing
Video Decoding & Rendering
Audio Decoding & Rendering
delay
Audio/Video Demuxing
Video Decoding
Video Rendering
Audio Decoding
Audio Rendering

3.3 비디오 속도 맞추기

DEVIEW 2020

그럴 듯 해 보이나 문제 발생 여지 많음

4.Advanced

4.1 A/V sync

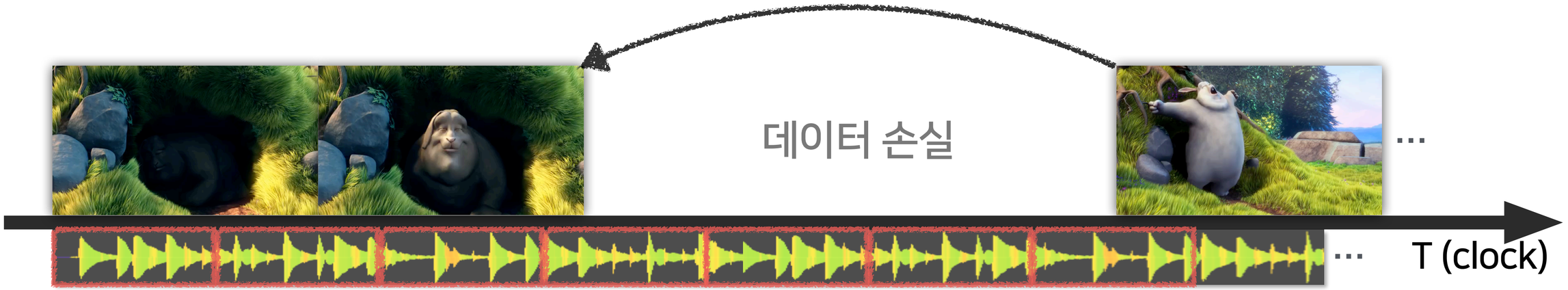
DEVIEW 2020

완벽하지 않은 세상

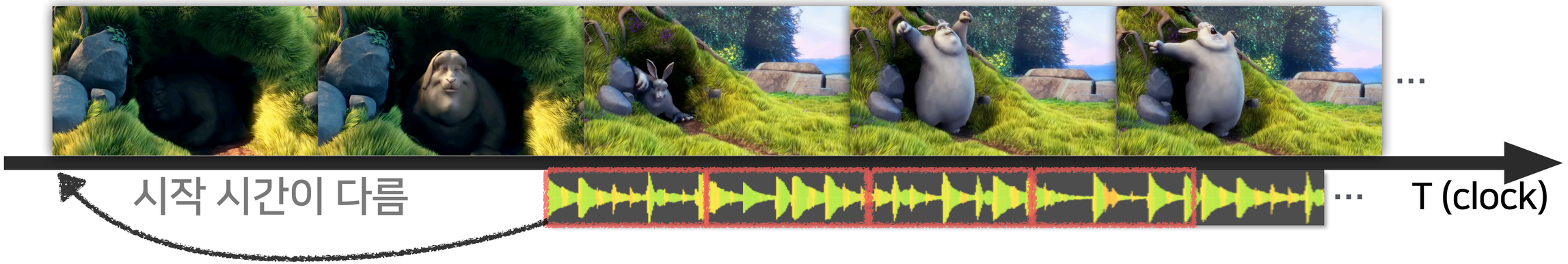
- 네트워크 버퍼링
- 패킷 손실
- 불완전한 파일

4.1 A/V sync

4.1 A/V sync



4.1 A/V sync



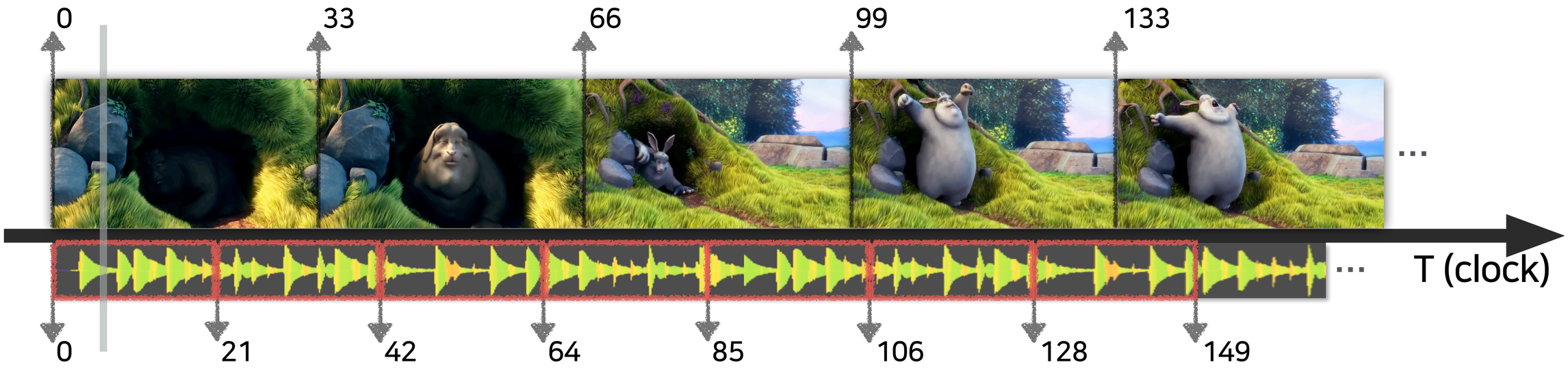
4.1 A/V sync

DEVIEW 2020

PTS (Presentation Time Stamp)

- Frame이 렌더링 되어야 하는 시간

4.1 A/V sync



4.1 A/V sync

DEVIEW 2020

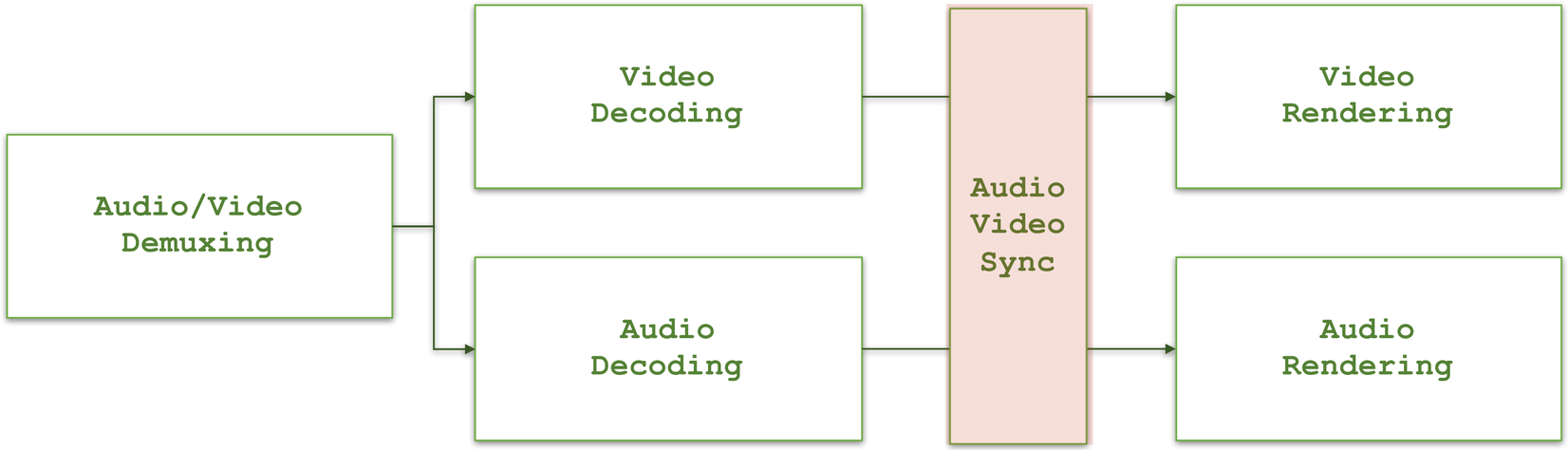
AVSync 맞추는 3가지 방법:

- 오디오 시간을 기준
- 비디오 시간을 기준
- 시스템 시간을 기준

우리는 **시스템 시간**을 기준으로

4.1 A/V sync

DEVIEW 2020



4.1 A/V sync

DEVIEW 2020

```
private val syncThread = HandlerThread("SyncThread").apply { start() }
private val syncHandler = Handler(syncThread.looper)

private val audioFrameQueue: Queue<AudioFrame> = ConcurrentLinkedQueue<AudioFrame>()
private val videoFrameQueue: Queue<VideoFrame> = ConcurrentLinkedQueue<VideoFrame>()
private var startTimeMs = -1L


private data class AudioFrame(val data: ByteBuffer, val bufferId: Int, val ptsUs: Long)
private data class VideoFrame(val bufferId: Int, val ptsUs: Long)


private fun queueAudio(data: ByteBuffer, bufferId: Int, ptsUs: Long) {
    audioFrameQueue.add(AudioFrame(data, bufferId, ptsUs))
    postSyncAudioVideo(0)
}

private fun queueVideo(bufferId: Int, ptsUs: Long) {
    videoFrameQueue.add(VideoFrame(bufferId, ptsUs))
    postSyncAudioVideo(0)
}
```

4.1 A/V sync

```
in 0..Int.MAX_VALUE -> {
    if ((videoBufferInfo.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
        videoDecoder.releaseOutputBuffer(outputIndex, false)
        videoOutEos = true
    } else {
        queueVideo(outputIndex, videoBufferInfo.presentationTimeUs)
    }
}
```

```
in 0..Int.MAX_VALUE -> {
    if ((audioBufferInfo.flags and MediaCodec.BUFFER_FLAG_END_OF_STREAM) != 0) {
        audioDecoder.releaseOutputBuffer(outputIndex, false)
        audioOutEos = true
    } else {
        val outputBuffer = audioDecoder.getOutputBuffer(outputIndex)!!
        outputBuffer.position(audioBufferInfo.offset)
        outputBuffer.limit(audioBufferInfo.offset + audioBufferInfo.size)

        queueAudio(outputBuffer, outputIndex,
            audioBufferInfo.presentationTimeUs)
    }
}
```

4.1 A/V sync

```kotlin
private val syncThread = HandlerThread("SyncThread").apply { start() }
private val syncHandler = Handler(syncThread.looper)

private val audioFrameQueue: Queue<AudioFrame> = ConcurrentLinkedQueue<AudioFrame>()
private val videoFrameQueue: Queue<VideoFrame> = ConcurrentLinkedQueue<VideoFrame>()
private var startTimeMs = -1L


private data class AudioFrame(val data: ByteBuffer, val bufferId: Int, val ptsUs: Long)
private data class VideoFrame(val bufferId: Int, val ptsUs: Long)


private fun queueAudio(data: ByteBuffer, bufferId: Int, ptsUs: Long) {
    audioFrameQueue.add(AudioFrame(data, bufferId, ptsUs))
    postSyncAudioVideo(0)
}

private fun queueVideo(bufferId: Int, ptsUs: Long) {
    videoFrameQueue.add(VideoFrame(bufferId, ptsUs))
    postSyncAudioVideo(0)
}
```

```kotlin
private fun postSyncAudioVideo(delayMillis: Long) {
    syncHandler.postDelayed({
        val curTimeMs = SystemClock.uptimeMillis()

        val audioFrame: AudioFrame? = audioFrameQueue.peek()
        val videoFrame: VideoFrame? = videoFrameQueue.peek()

        if (audioFrame == null && videoFrame == null) {
            return@postDelayed
        }

        if (startTimeMs < 0) {
            if (audioFrame == null || videoFrame == null) {
                return@postDelayed
            }

            val startPtsUs = min(audioFrame.ptsUs, videoFrame.ptsUs)
            startTimeMs = curTimeMs - startPtsUs / 1000L
        }

        if (audioFrame != null) {
            val renderTimeMs = startTimeMs + audioFrame.ptsUs / 1000L
            postRenderAudioAtTime(audioFrame, renderTimeMs)
            audioFrameQueue.remove()
        }

        if (videoFrame != null) {
            val renderTimeMs = startTimeMs + videoFrame.ptsUs / 1000L
            if (renderTimeMs >= curTimeMs) {
                postRenderVideoAtTime(videoFrame, renderTimeMs)
            } else {
                // TODO: 단순히 프레임을 skip하는 것 외에 어떤 방법이 있는지 살펴보세요.
                videoDecoder.releaseOutputBuffer(videoFrame.bufferId, false)
            }

            videoFrameQueue.remove()
        }

        if (!audioFrameQueue.isEmpty() || !videoFrameQueue.isEmpty()) {
            postSyncAudioVideo(0)
        } else {
            postSyncAudioVideo(10)
        }
    }, delayMillis)
}
```

4.1 A/V sync

```
private fun postSyncAudioVideo(delayMillis: Long) {
    syncHandler.postDelayed({
        val curTimeMs = SystemClock.uptimeMillis()

        val audioFrame: AudioFrame? = audioFrameQueue.peek()
        val videoFrame: VideoFrame? = videoFrameQueue.peek()

        if (audioFrame == null && videoFrame == null) {
            return@postDelayed
        }

        if (startTimeMs < 0) {
            if (audioFrame == null || videoFrame == null) {
                return@postDelayed
            }

            val startPtsUs = min(audioFrame.ptsUs, videoFrame.ptsUs)
            startTimeMs = curTimeMs - startPtsUs / 1000L
        }

        if (audioFrame != null) {
            val renderTimeMs = startTimeMs + audioFrame.ptsUs / 1000L
            postRenderAudioAtTime(audioFrame, renderTimeMs)
            audioFrameQueue.remove()
        }

        if (videoFrame != null) {
            val renderTimeMs = startTimeMs + videoFrame.ptsUs / 1000L
            if (renderTimeMs >= curTimeMs) {
                postRenderVideoAtTime(videoFrame, renderTimeMs)
            } else {
                // TODO: 단순히 프레임을 skip하는 것 외에 어떤 방법이 있는지 살펴보세요.
                videoDecoder.releaseOutputBuffer(videoFrame.bufferId, false)
            }

            videoFrameQueue.remove()
        }

        if (!audioFrameQueue.isEmpty() || !videoFrameQueue.isEmpty()) {
            postSyncAudioVideo(0)
        } else {
            postSyncAudioVideo(10)
        }
    }, delayMillis)
}
```

4.1 A/V sync

```
private fun postSyncAudioVideo(delayMillis: Long) {
    syncHandler.postDelayed({
        val curTimeMs = SystemClock.uptimeMillis()

        val audioFrame: AudioFrame? = audioFrameQueue.peek()
        val videoFrame: VideoFrame? = videoFrameQueue.peek()

        if (audioFrame == null && videoFrame == null) {
            return@postDelayed
        }

        if (startTimeMs < 0) {
            if (audioFrame == null || videoFrame == null) {
                return@postDelayed
            }

            val startPtsUs = min(audioFrame.ptsUs, videoFrame.ptsUs)
            startTimeMs = curTimeMs - startPtsUs / 1000L
        }

        if (audioFrame != null) {
            val renderTimeMs = startTimeMs + audioFrame.ptsUs / 1000L
            postRenderAudioAtTime(audioFrame, renderTimeMs)
            audioFrameQueue.remove()
        }

        if (videoFrame != null) {
            val renderTimeMs = startTimeMs + videoFrame.ptsUs / 1000L
            if (renderTimeMs >= curTimeMs) {
                postRenderVideoAtTime(videoFrame, renderTimeMs)
            } else {
                // TODO: 단순히 프레임을 skip하는 것 외에 어떤 방법이 있는지 살펴보세요.
                videoDecoder.releaseOutputBuffer(videoFrame.bufferId, false)
            }

            videoFrameQueue.remove()
        }

        if (!audioFrameQueue.isEmpty() || !videoFrameQueue.isEmpty()) {
            postSyncAudioVideo(0)
        } else {
            postSyncAudioVideo(10)
        }
    }, delayMillis)
}
```

시스템 시간을 획득

4.1 A/V sync

```
private fun postSyncAudioVideo(delayMillis: Long) {
    syncHandler.postDelayed({
        val curTimeMs = SystemClock.uptimeMillis()

        val audioFrame: AudioFrame? = audioFrameQueue.peek()
        val videoFrame: VideoFrame? = videoFrameQueue.peek()

        if (audioFrame == null && videoFrame == null) {
            return@postDelayed
        }

        if (startTimeMs < 0) {
            if (audioFrame == null || videoFrame == null) {
                return@postDelayed
            }

            val startPtsUs = min(audioFrame.ptsUs, videoFrame.ptsUs)
            startTimeMs = curTimeMs - startPtsUs / 1000L
        }

        if (audioFrame != null) {
            val renderTimeMs = startTimeMs + audioFrame.ptsUs / 1000L
            postRenderAudioAtTime(audioFrame, renderTimeMs)
            audioFrameQueue.remove()
        }

        if (videoFrame != null) {
            val renderTimeMs = startTimeMs + videoFrame.ptsUs / 1000L
            if (renderTimeMs >= curTimeMs) {
                postRenderVideoAtTime(videoFrame, renderTimeMs)
            } else {
                // TODO: 단순히 프레임을 skip하는 것 외에 어떤 방법이 있는지 살펴보세요.
                videoDecoder.releaseOutputBuffer(videoFrame.bufferId, false)
            }

            videoFrameQueue.remove()
        }

        if (!audioFrameQueue.isEmpty() || !videoFrameQueue.isEmpty()) {
            postSyncAudioVideo(0)
        } else {
            postSyncAudioVideo(10)
        }
    }, delayMillis)
}
```

최초 한번 시작 시간을 기록

4.1 A/V sync

```
private fun postSyncAudioVideo(delayMillis: Long) {
    syncHandler.postDelayed({
        val curTimeMs = SystemClock.uptimeMillis()

        val audioFrame: AudioFrame? = audioFrameQueue.peek()
        val videoFrame: VideoFrame? = videoFrameQueue.peek()

        if (audioFrame == null && videoFrame == null) {
            return@postDelayed
        }

        if (startTimeMs < 0) {
            if (audioFrame == null || videoFrame == null) {
                return@postDelayed
            }

            val startPtsUs = min(audioFrame.ptsUs, videoFrame.ptsUs)
            startTimeMs = curTimeMs - startPtsUs / 1000L
        }

        if (audioFrame != null) {
            val renderTimeMs = startTimeMs + audioFrame.ptsUs / 1000L
            postRenderAudioAtTime(audioFrame, renderTimeMs)
            audioFrameQueue.remove()
        }

        if (videoFrame != null) {
            val renderTimeMs = startTimeMs + videoFrame.ptsUs / 1000L
            if (renderTimeMs >= curTimeMs) {
                postRenderVideoAtTime(videoFrame, renderTimeMs)
            } else {
                // TODO: 단순히 프레임을 skip하는 것 외에 어떤 방법이 있는지 살펴보세요.
                videoDecoder.releaseOutputBuffer(videoFrame.bufferId, false)
            }

            videoFrameQueue.remove()
        }

        if (!audioFrameQueue.isEmpty() || !videoFrameQueue.isEmpty()) {
            postSyncAudioVideo(0)
        } else {
            postSyncAudioVideo(10)
        }
    }, delayMillis)
}
```

오디오 프레임 렌더링 시간 계산

4.1 A/V sync

```
private fun postSyncAudioVideo(delayMillis: Long) {
    syncHandler.postDelayed({
        val curTimeMs = SystemClock.uptimeMillis()

        val audioFrame: AudioFrame? = audioFrameQueue.peek()
        val videoFrame: VideoFrame? = videoFrameQueue.peek()

        if (audioFrame == null && videoFrame == null) {
            return@postDelayed
        }

        if (startTimeMs < 0) {
            if (audioFrame == null || videoFrame == null) {
                return@postDelayed
            }

            val startPtsUs = min(audioFrame.ptsUs, videoFrame.ptsUs)
            startTimeMs = curTimeMs - startPtsUs / 1000L
        }

        if (audioFrame != null) {
            val renderTimeMs = startTimeMs + audioFrame.ptsUs / 1000L
            postRenderAudioAtTime(audioFrame, renderTimeMs)
            audioFrameQueue.remove()
        }

        if (videoFrame != null) {
            val renderTimeMs = startTimeMs + videoFrame.ptsUs / 1000L
            if (renderTimeMs >= curTimeMs) {
                postRenderVideoAtTime(videoFrame, renderTimeMs)
            } else {
                // TODO: 단순히 프레임을 skip하는 것 외에 어떤 방법이 있는지 살펴보세요.
                videoDecoder.releaseOutputBuffer(videoFrame.bufferId, false)
            }

            videoFrameQueue.remove()
        }

        if (!audioFrameQueue.isEmpty() || !videoFrameQueue.isEmpty()) {
            postSyncAudioVideo(0)
        } else {
            postSyncAudioVideo(10)
        }
    }, delayMillis)
}
```

비디오 프레임 렌더링 시간 계산

4.2 Seeking

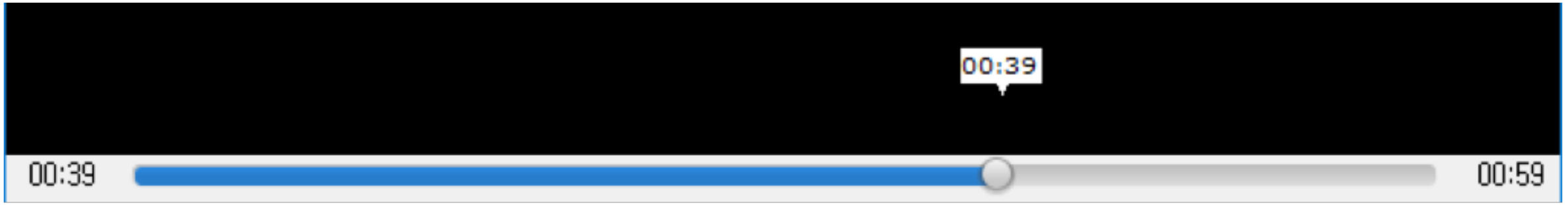
DEVIEW 2020

재생 중 임의의 지점으로 이동

4.2 Seeking

DEVIEW 2020

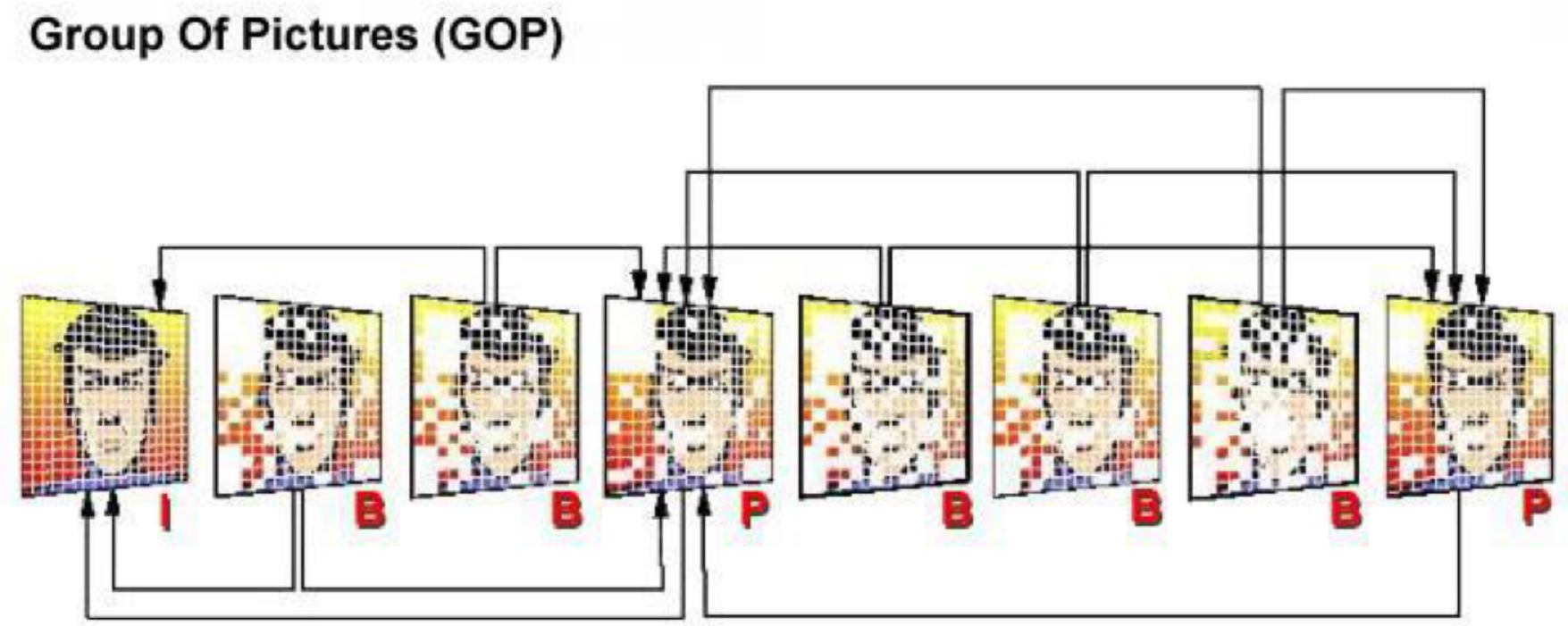
이런적 있나요?



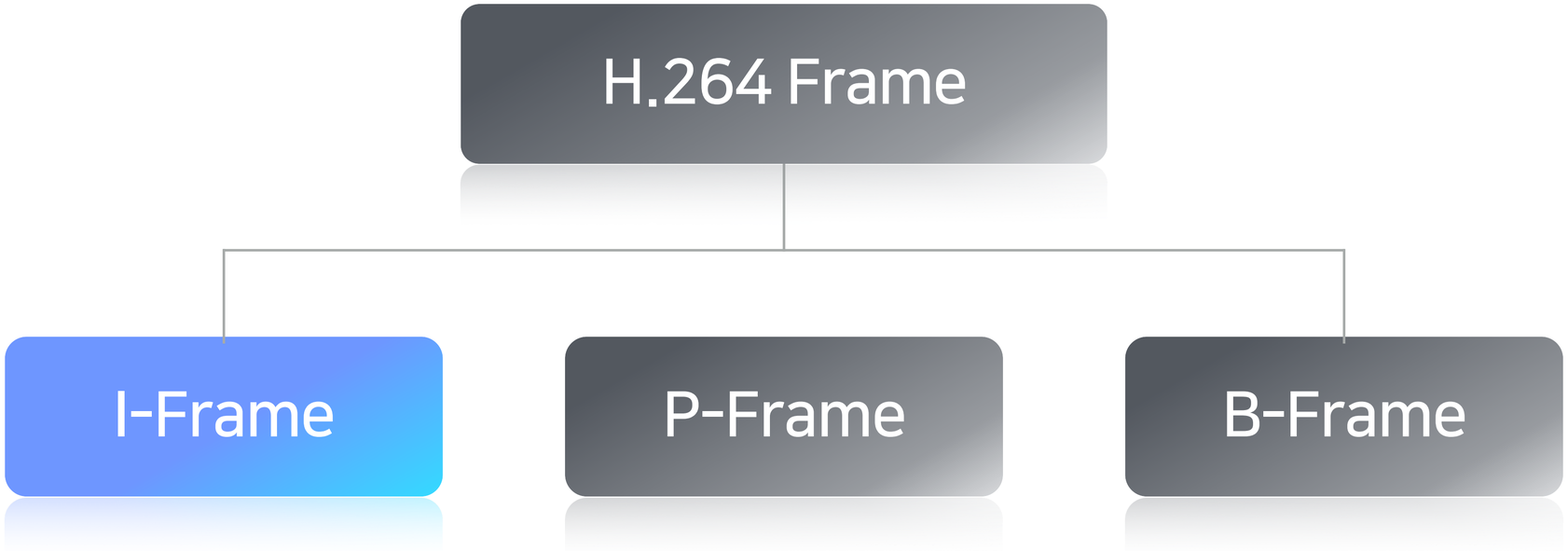
00:39 위치를 클릭했는데, 실제 재생은 00:35 부터 시작

4.2 Seeking

DEVIEW 2020

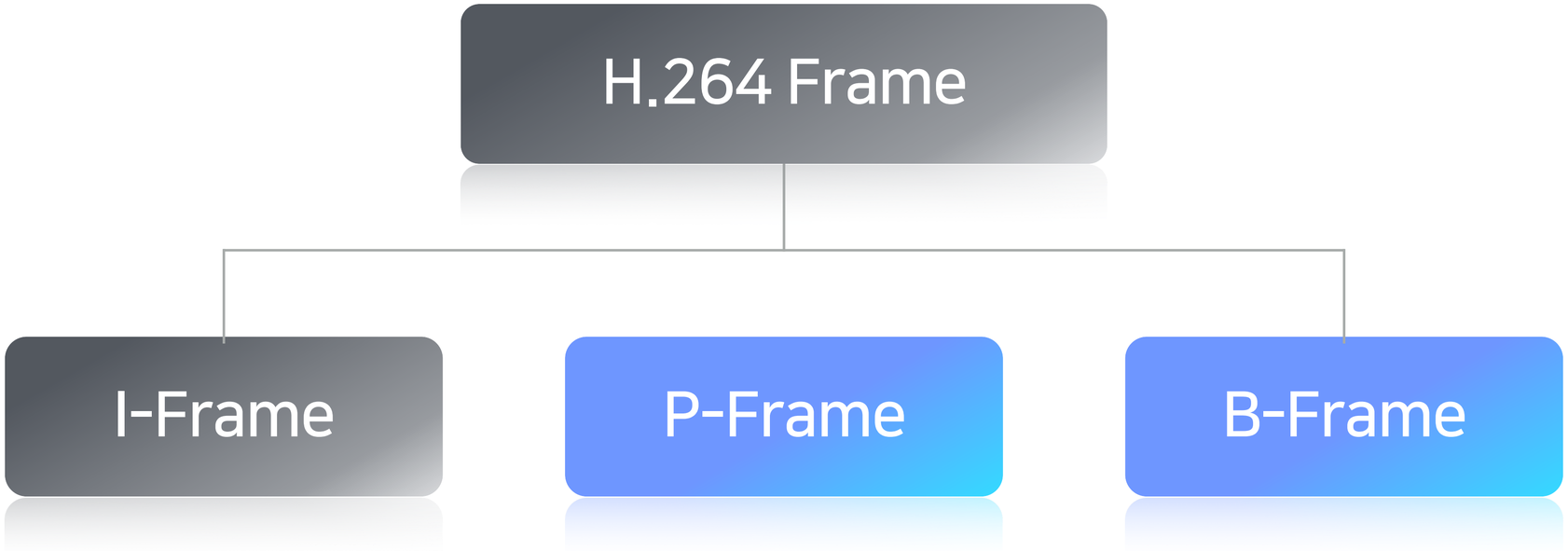


4.2 Seeking



단독으로 디코딩 가능

4.2 Seeking



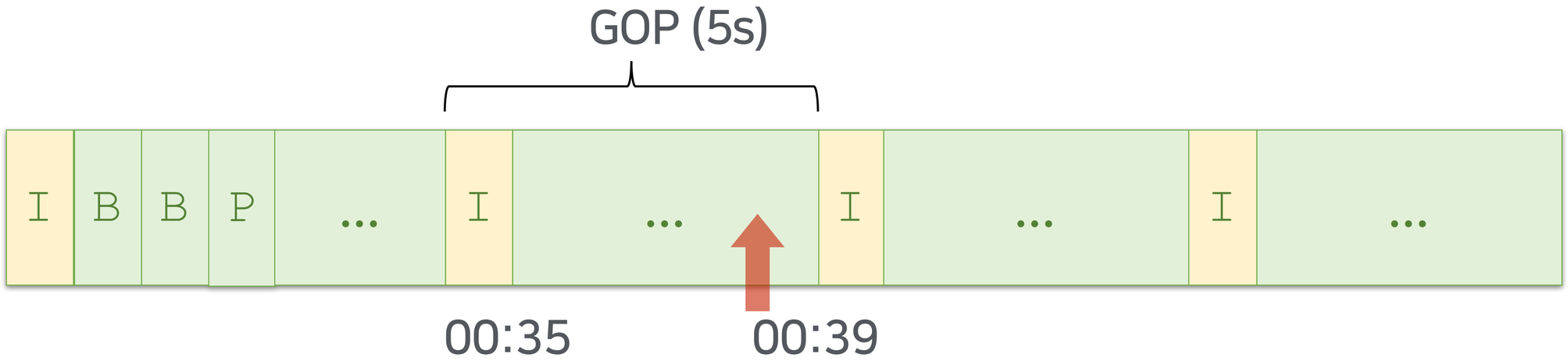
단독으로 디코딩 불가능

4.2 Seeking

DEVIEW 2020

I-Frame을 기준으로 탐색

I-Frame 간격은 **인코딩시 설정** (1초, 5초, 10초, …)



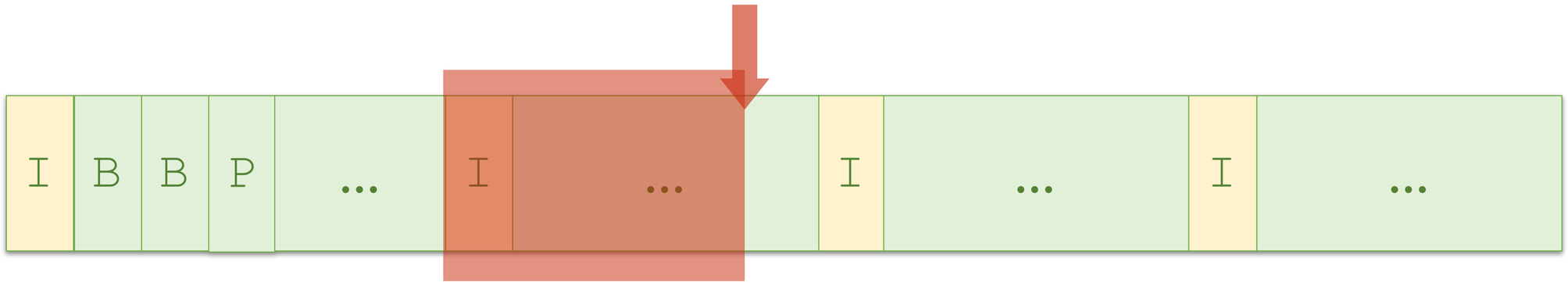
4.2 Seeking

DEVIEW 2020

탐색 위치를 기준으로

- SEEK_PREVIOUS_SYNC
- SEEK_NEXT_SYNC
- SEEK_ACCURATE

SEEK_ACCURATE의 경우 **심각한 성능 문제** 발생 가능

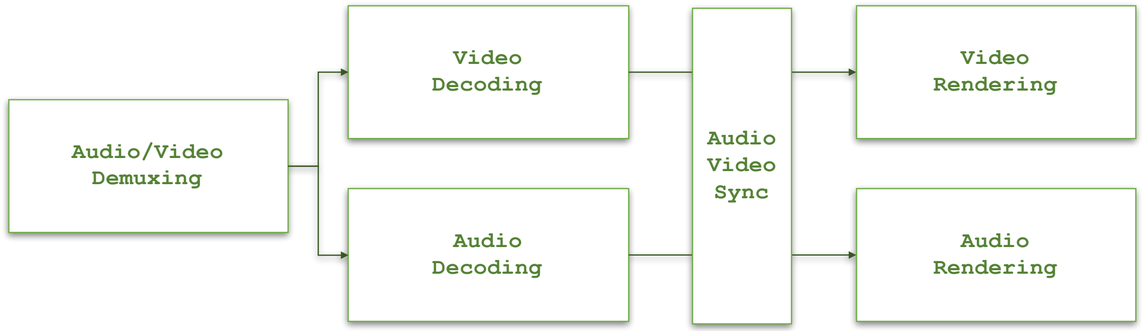


4.2 Seeking

DEVIEW 2020

해야 하는 일

- 모든 쓰레드의 동작을 정지
- Pipeline flush
- Decoder flush
- Demuxer 위치 이동
- 모든 쓰레드 태스크 시작



Outro

Outro

DEVIEW 2020

동영상의 기본 개념을 살펴보고,

동영상 플레이어의 동작 원리를 알아보았습니다.

Outro

DEVIEW 2020

참고 자료:

- https://github.com/leandromoreira/digital_video_introduction
- https://exoplayer.dev
- https://www.vcodex.com/h264-resources
- https://github.com/FFmpeg/FFmpeg/blob/master/fftools/ffplay.c
- https://developer.android.com/reference/android/media/MediaExtractor
- https://developer.android.com/reference/android/media/MediaCodec

Q & A

Thank You