HLS In Depth

Kinds of Playback

Local File	file:///.../example.MOV		
Progressive Download	https://example.com/example.MOV		
HTTP Live Streaming	master playlist	video 6Mbit playlist	 segments
		video 4Mbit playlist	 segments
		video 2Mbit playlist	 segments
		audio stereo playlist	 segments
		audio surround playlist	 segments

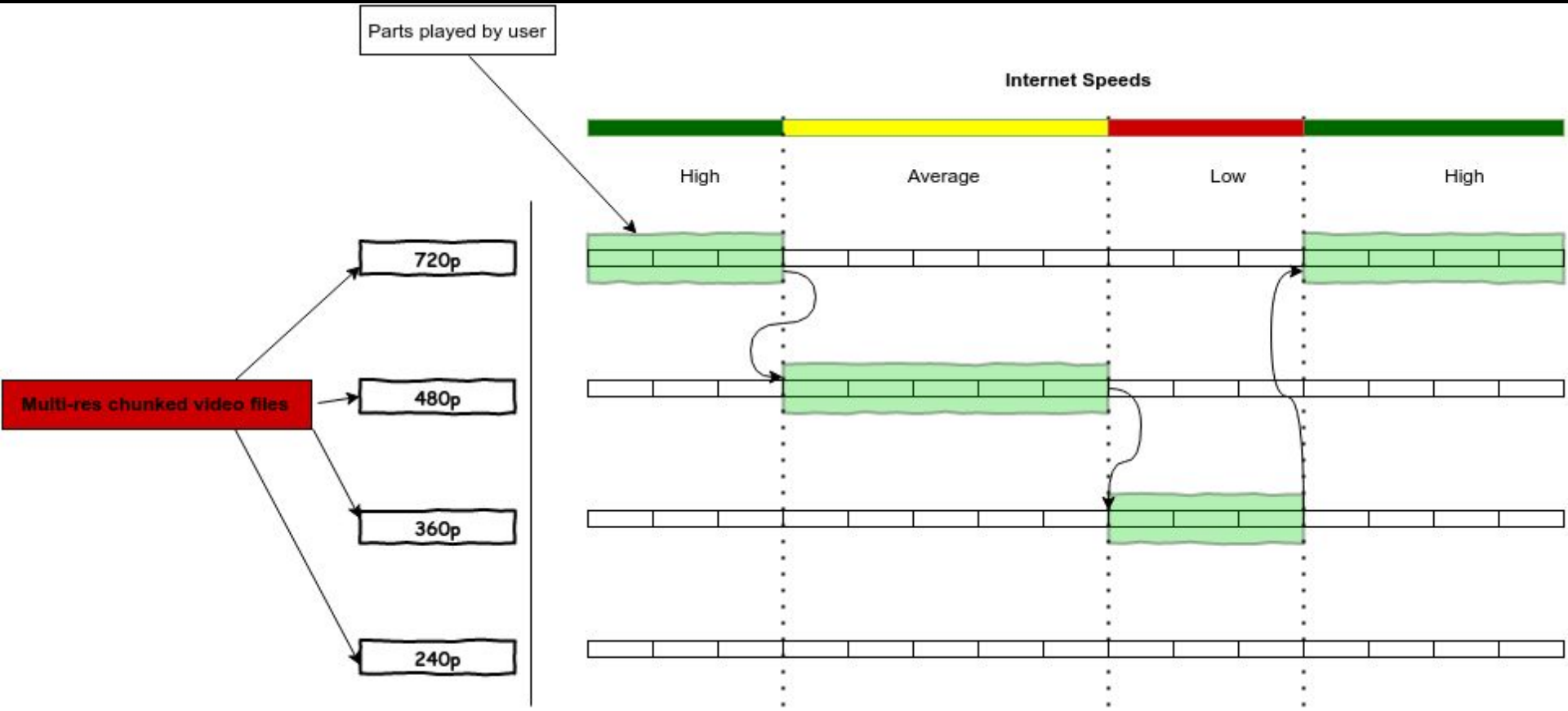
What is HLS?

HTTP Live Streaming (HLS) sends audio and video over HTTP from an ordinary web server for playback on iOS-based devices—including iPhone, iPad, iPod touch, and Apple TV—and on desktop computers (macOS). Using the same protocol that powers the web, HLS deploys content using ordinary web servers and content delivery networks. HLS is designed for reliability and dynamically adapts to network conditions by optimizing playback for the available speed of wired and wireless connections.

What does HLS Support?

- Live broadcasts and prerecorded content (video on demand, or VOD)
- Multiple alternate streams at different bit rates
- Intelligent switching of streams in response to network bandwidth changes
- Media encryption and user authentication

Adaptive Video Streaming

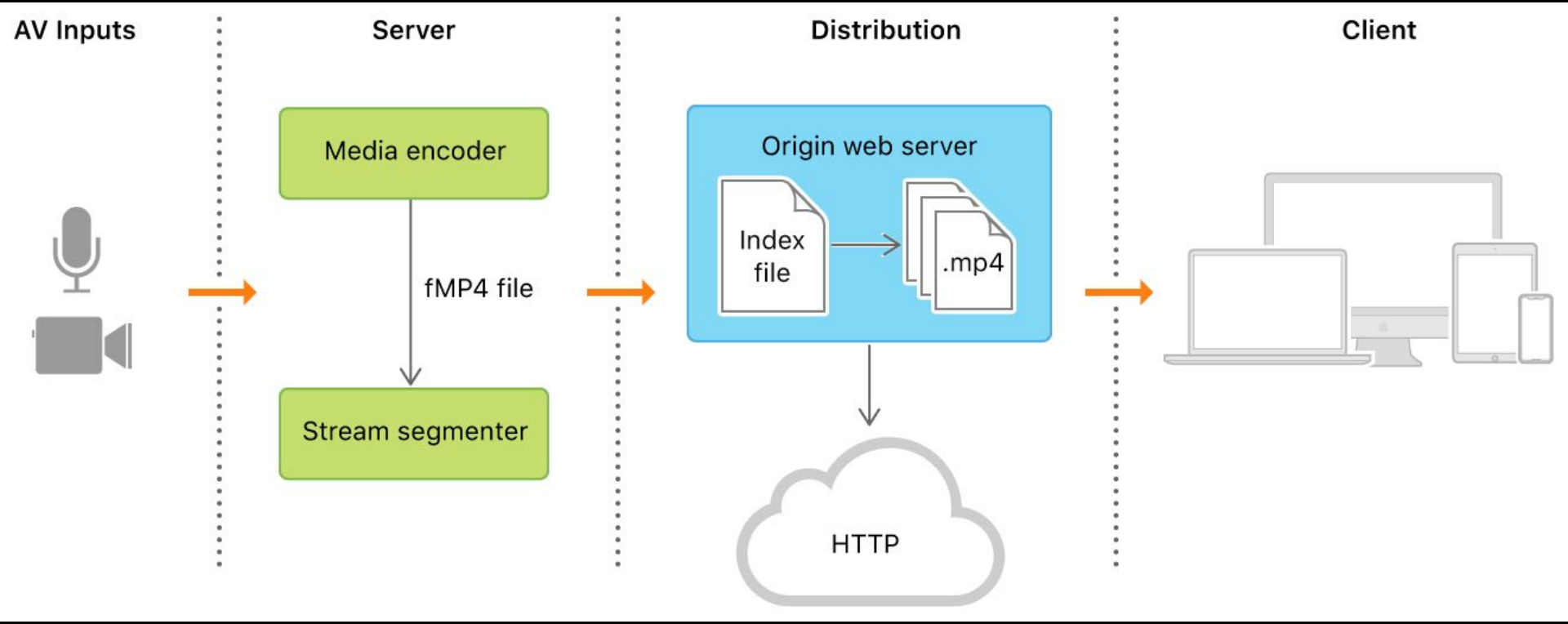


HTTP Live Streaming Architecture

HTTP Live Streaming consists of three parts:

- Server Component
- Distribution Component
- Client Software.

How it all works?



Server(Encode + Transcode + Segment)

Media Encoder In HLS - takes audio-video input from source, encodes it as HEVC or H.264 video and AAC or AC-3 audio, and outputs a fragmented MPEG-4 file or an MPEG-2 transport stream.

Stream Segmenter - breaks the stream into a series of short media files, which are placed on a web server. The segmenter also creates and maintains an index file(.m3u8) containing a list of the media files.

Distribution(Origin web server + CDN)

Origin Web Server - system that delivers the media files and index files to the client over HTTP.

CDN(Edge Servers) - refers to a geographically distributed group of servers which work together to provide fast content delivery(cache content at the network edge) e.g. Akamai, Amazon Cloudfront, Azure CDN etc.

Client(Player)

Player begins by fetching the index file, using a URL that identifies the stream(.m3u8 url). The index file, in turn, specifies the location of the available media files, decryption keys, and any alternate streams available. For the selected stream, the player downloads each available media file in sequence. Each file contains a consecutive segment of the stream. Once it has a sufficient amount of data downloaded, the player begins presenting the reassembled stream to the user.

This process continues until the player encounters the EXT-X-ENDLIST tag in the index file. If no EXT-X-ENDLIST tag is present, the index file is part of an ongoing broadcast.

Deploying a Basic HTTP Live Stream

Requirements for deploying an HTTP Live Stream:

- Either an HTML page for browsers or a client app to act as a receiver.
- A web server or CDN to act as a host.
- A way to encode your source material or live streams as fragmented MPEG-4 media files containing HEVC or H.264 video and AAC or AC-3 audio.

Demo: HLS In Action

1. Using Apple’s Web Server
2. Using Local Web Server
3. Example Playlists

M3U8 Playlist

#EXTM3U

#EXT-X-PLAYLIST-TYPE

#EXT-X-TARGETDURATION

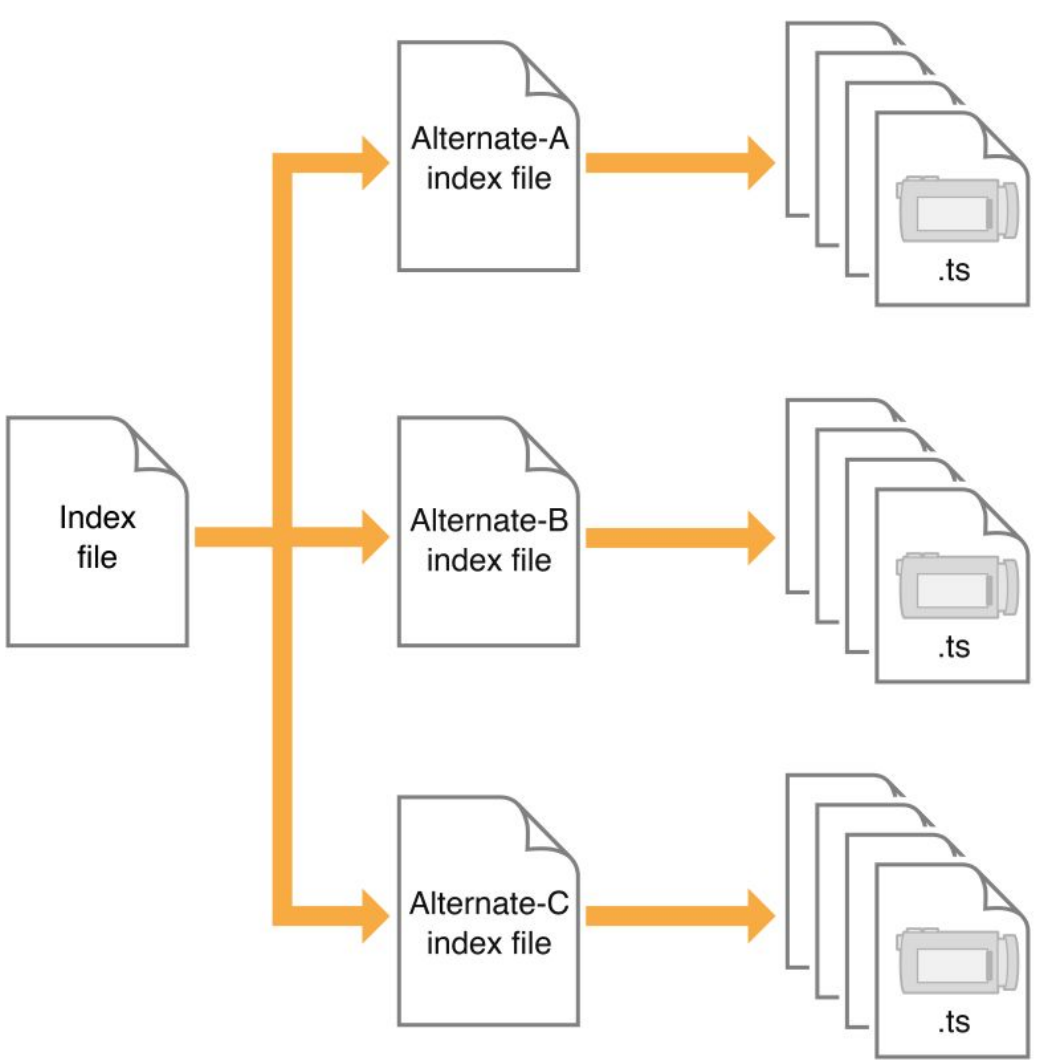
#EXT-X-VERSION

#EXT-X-MEDIA-SEQUENCE

#EXTINF

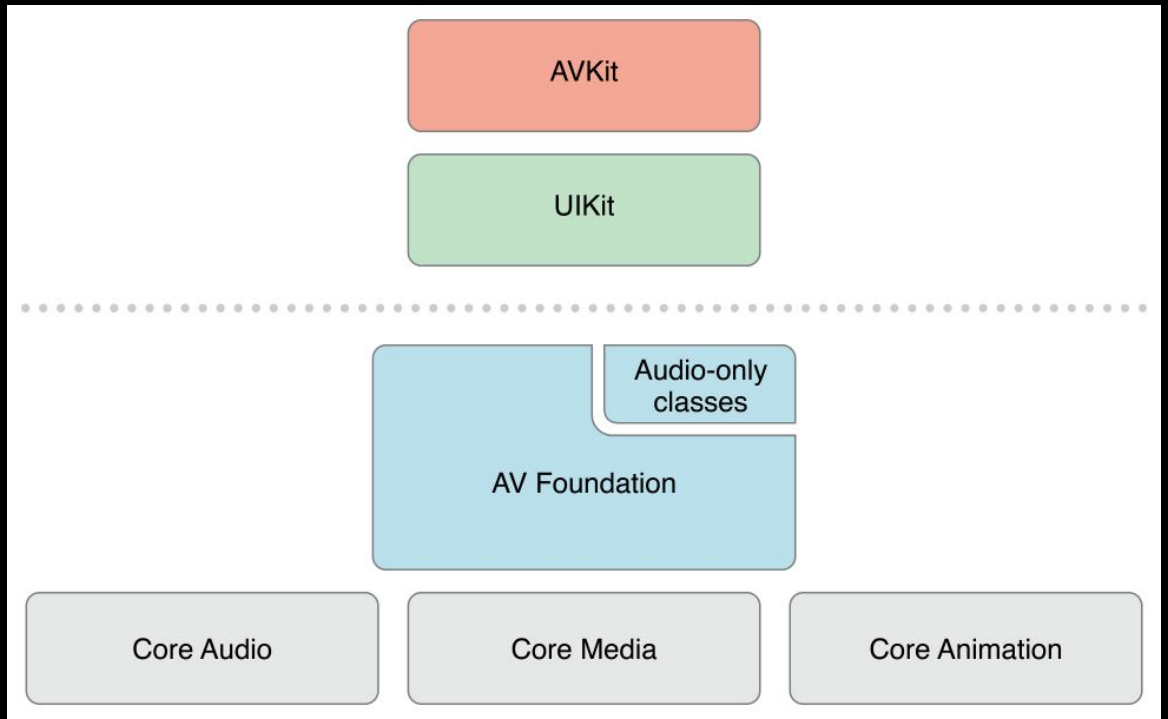
#EXT-X-STREAM-INF

#EXT-X-ENDLIST



Introduction to AVKit

The AVKit framework provides a high-level interface for playing video content.



Demo: Play HLS using AVPlayerViewController

Let’s use AVPlayerViewController to play HLS with system system-supplied playback controls.

File Playback

Network Playback

Video Processing

Metadata

Audio Mixing

Alternate Audio

AVFoundation

Photo Capture

Video Capture

Export

Subtitles

Editing

Video Effects

Introduction to AVFoundation

AVFoundation is the full featured framework for working with time-based audiovisual media on iOS, macOS, watchOS and tvOS. Using AVFoundation, you can easily play, create, and edit QuickTime movies and MPEG-4 files, play HLS streams, and build powerful media functionality into your apps.

AVFoundation: Main classes for HLS

AVPlayerLayer - This special CALayer subclass can display the playback of a given AVPlayer object i.e. manages a player's visual output.

AVAsset - These are static representations of a media asset. An asset object defines the collective properties of the tracks that comprise it such as duration, creationDate, preferredRate, preferredVolume, playable, metadata etc.

AVPlayerItem - The dynamic counterpart to an AVAsset. This object represents the current state of an asset(playable video) played by the player.

AVPlayer - Controller object used to manage the playback and timing of a media asset.

Demo: Custom Player with AVFoundation

1. Custom Player View with AVPlayerLayer
2. HLSViewController with controls for play/pause and state management.

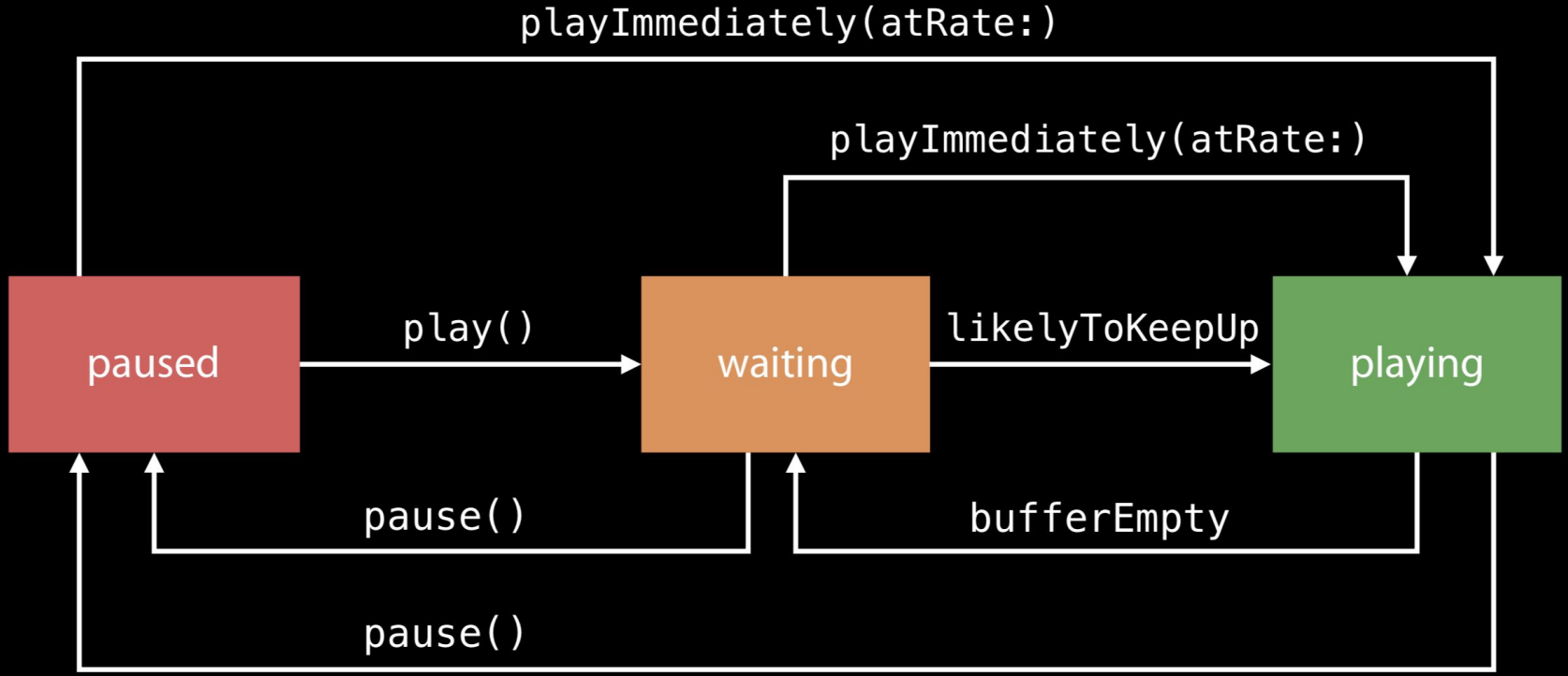
Order Matters

Ask for the final goal first

```
let asset = AVURLAsset(url: url)
let playerItem = AVPlayerItem(asset: asset)
let player = AVPlayer()
let playerLayer = AVPlayerLayer(player: player)
player.replaceCurrentItemWithPlayerItem(playerItem)
```

1. set up audio+video playback

Playing Paused and Waiting States



AVPlayer.rate

	waiting
`AVPlayer.rate` the app's requested playback rate	`1.0`
`AVPlayerItem.timebase.rate` the rate at which playback is actually occurring	`0.0`
`AVPlayer.timeControlStatus` Paused, WaitingToPlayAtSpecifiedRate, Playing	`WaitingToPlayAtSpecifiedRate`
`AVPlayer.reasonForWaitingToPlay`	`WaitingToMinimizeStallsReason`

rate vs timeControlStatus

AVPlayer.rate	AVPlayer.timeControlStatus
0.0	paused
1.0	waiting
1.0	playing

Seek playback to time

seek(to:toleranceBefore:toleranceAfter:) - Sets the current playback time within a specified time bound.

The time seeked to will be within the range [time-beforeTolerance, time+afterTolerance], and may differ from the specified time for efficiency. You can request sample accurate seeking by passing a time value of kCMTimeZero for both toleranceBefore and toleranceAfter. Sample accurate seeking may incur additional decoding delay which can impact seeking performance.

Passing kCMTimePositiveInfinity for both toleranceBefore and toleranceAfter is the same as messaging **seek(to:)** directly.

Video Quality

preferredPeakBitRate - The desired limit, in bits per second, of network bandwidth consumption for this item i.e. indicate that the player should attempt to limit item playback to that bit rate.

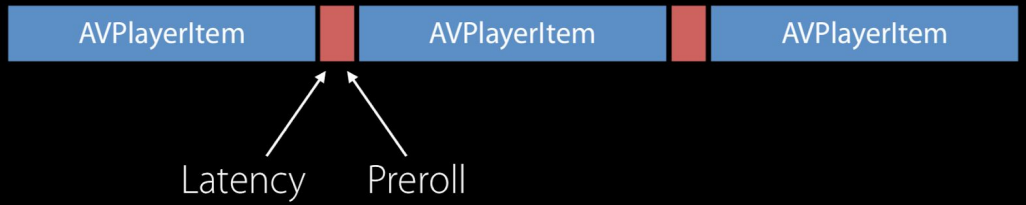
preferredMaximumResolution - The desired maximum resolution of a video that is to be downloaded. Defaults to CGSizeZero, which indicates there is no limit on the video resolution.

Demo: Custom Player with AVFoundation

1. Video Scrubbing i.e. seekToTime
2. Forward/Backward 10 seconds
3. Playback Rate, Volume and Brightness
4. Video Quality(Bitrate/Resolution)

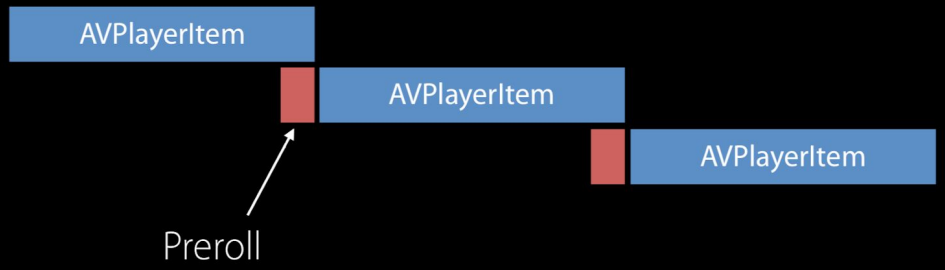
How Do You Loop an AVPlayerItem?

When end reached, rewind?



How Do You Loop an AVPlayerItem?

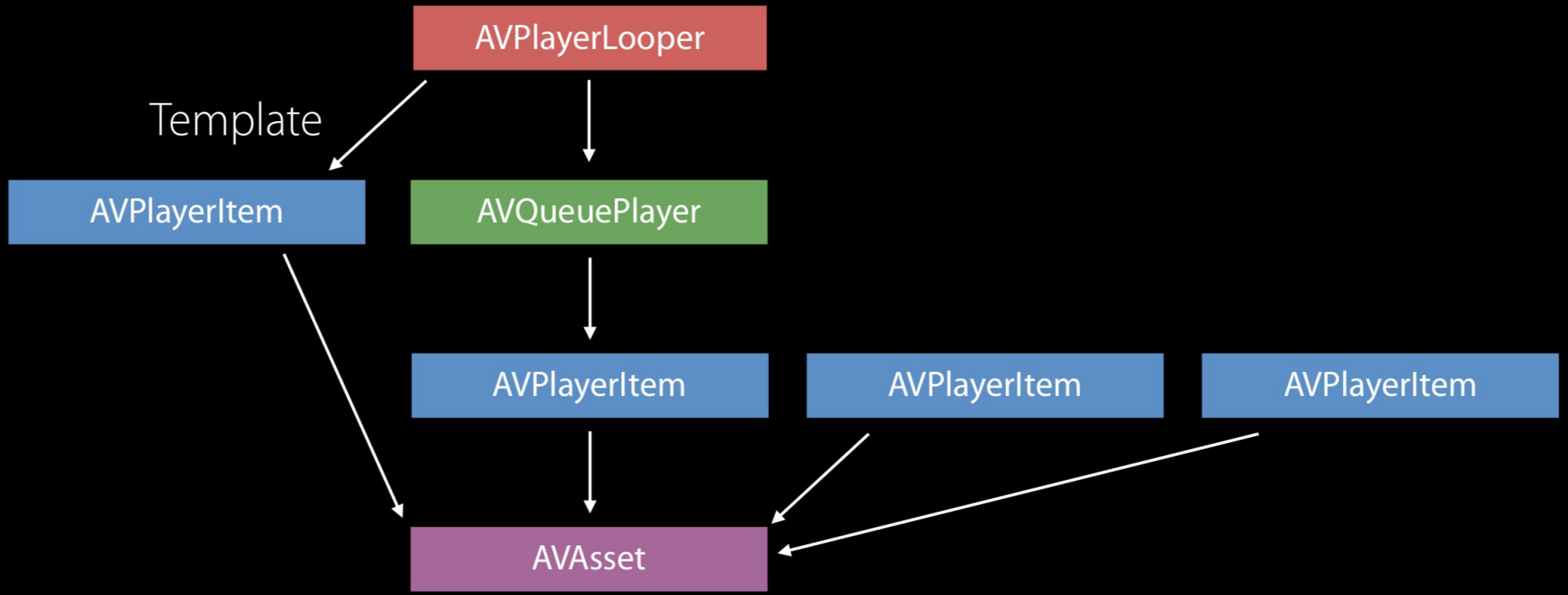
When end reached, rewind?



Playing Items in Queue

AVQueuePlayer - A player used to play a number of items in sequence i.e. create and manage a queue of player items.

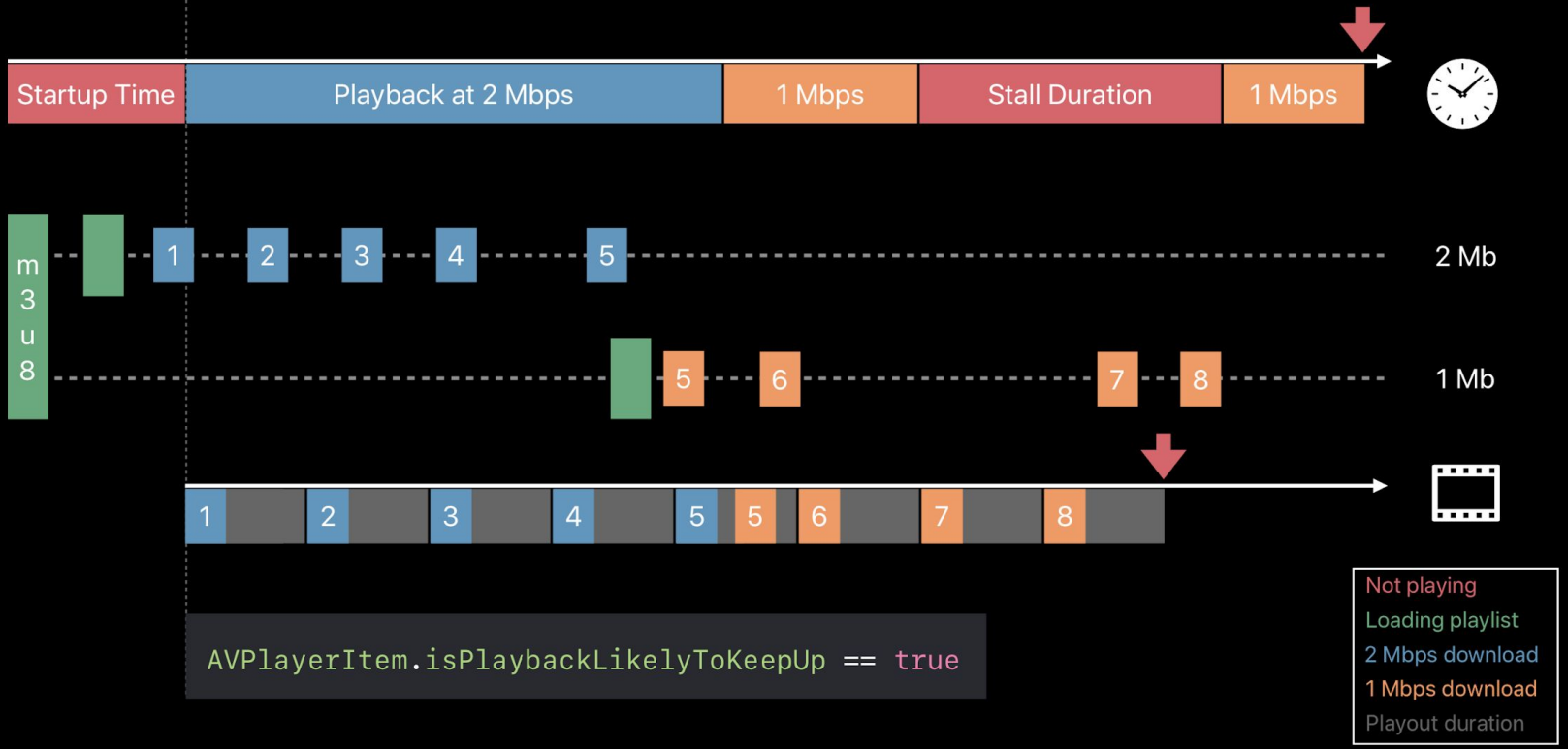
AVPlayerLooper(Playing single Item in Loop)



Demo: Playing Streams in Loop

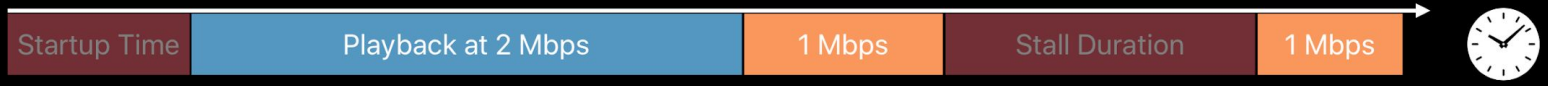
1. Loop videos by using AVQueuePlayer.
2. Looping Logic i.e. state management

Anatomy of an HLS Playback Session



Quantifying the User Experience

Key Performance Indicators (KPIs) for HLS



How much time did the user spend waiting for playback to start?

How often was playback interrupted?

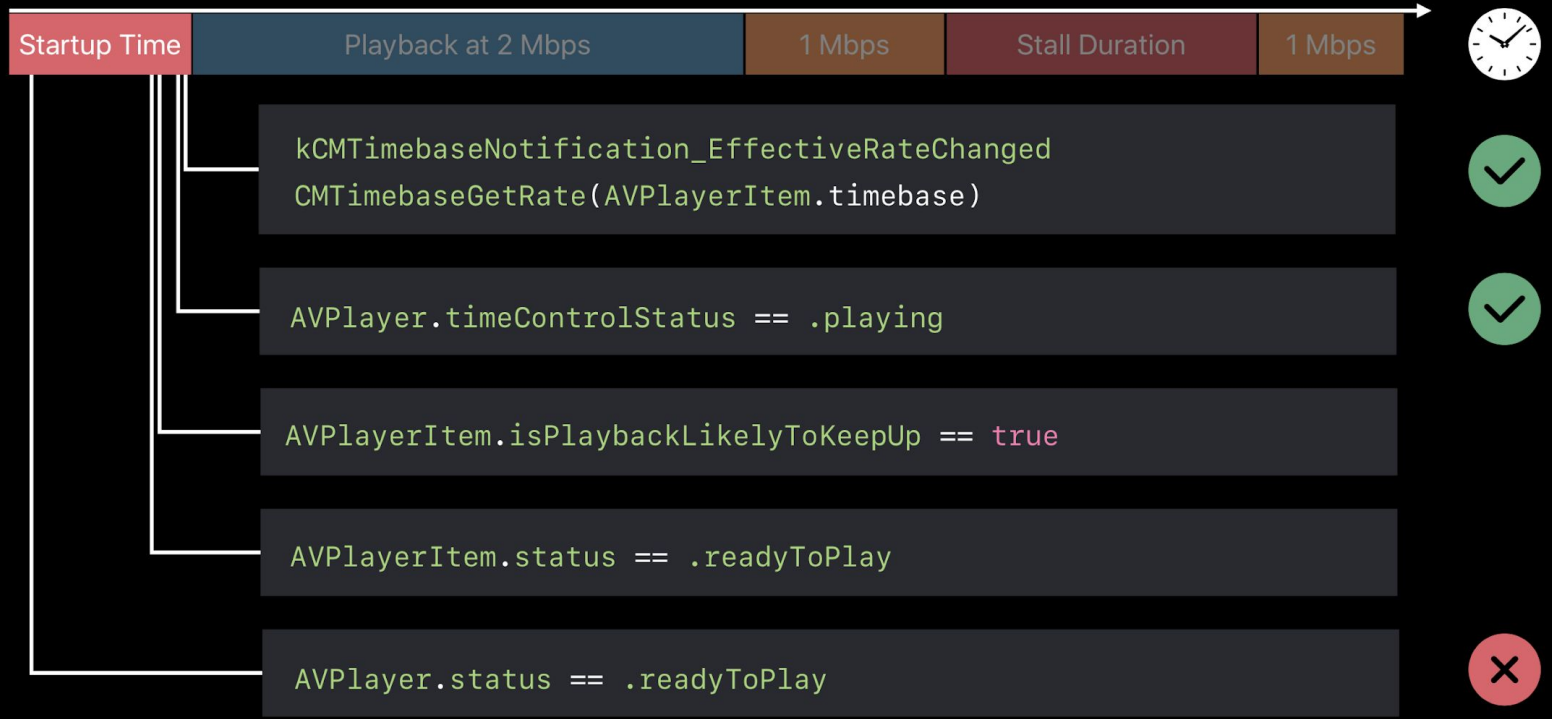
How long was playback interrupted?

What was the overall media quality?

How often did the stream end due to an error?

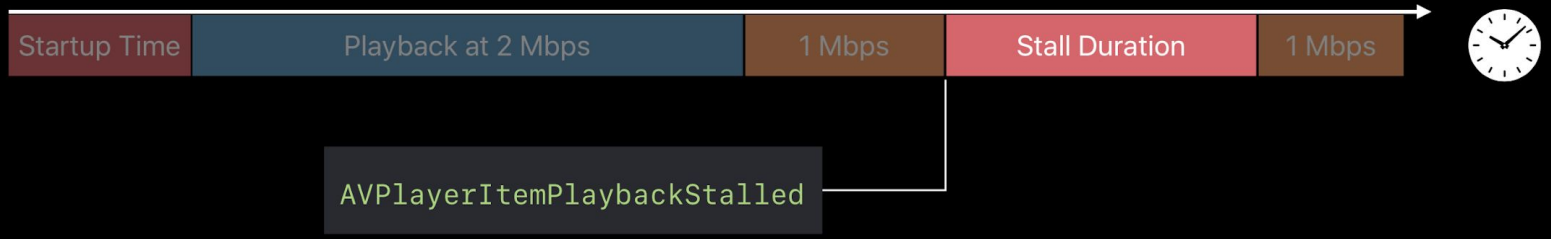
Startup Time

How much time did users spend waiting for playback to start?



Stall Rate

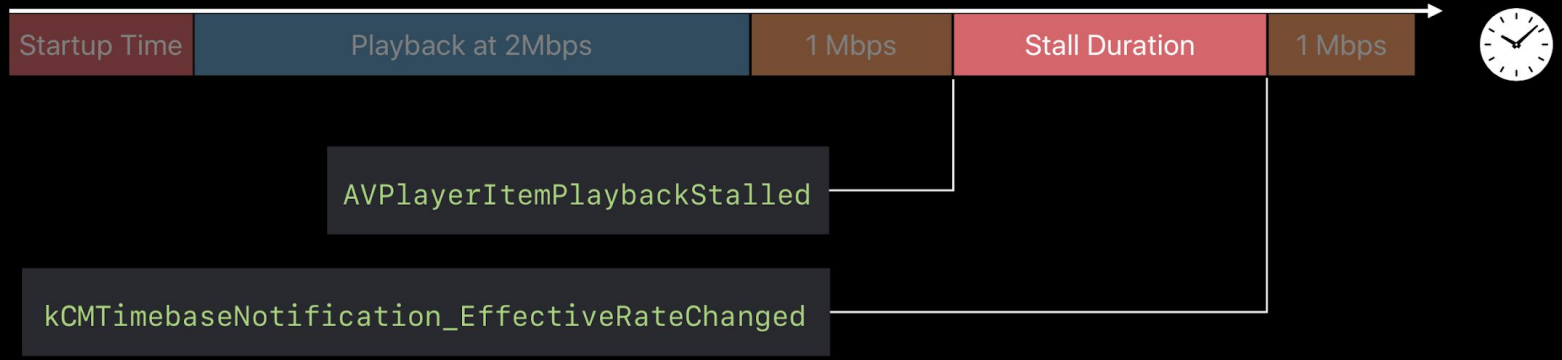
How often was playback interrupted?



KPI	Unit
Stall rate (normalized to duration watched)	stalls / hour watched

Stall Duration

How long was playback interrupted?



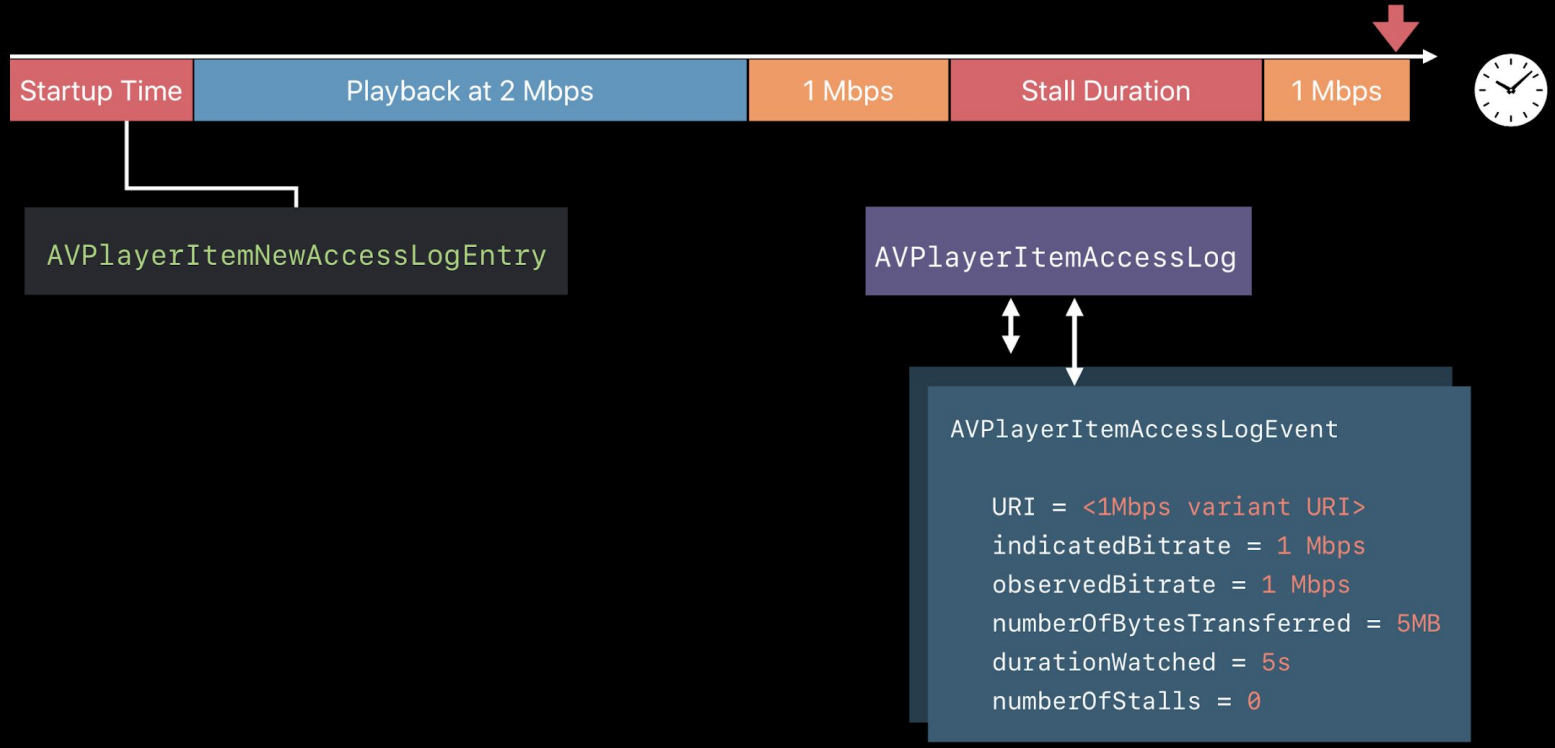
KPI	Unit
Stall duration to duration watched ratio	total stall duration / total duration watched

```
//Computing total duration watched from AVPlayerItem's access log

var totalDurationWatched = 0.0
if let accessLog = playerItem.accessLog() {
    for event in accessLog.events {
        if event.durationWatched > 0 {
                totalDurationWatched += event.durationWatched
        }
    }
}
```

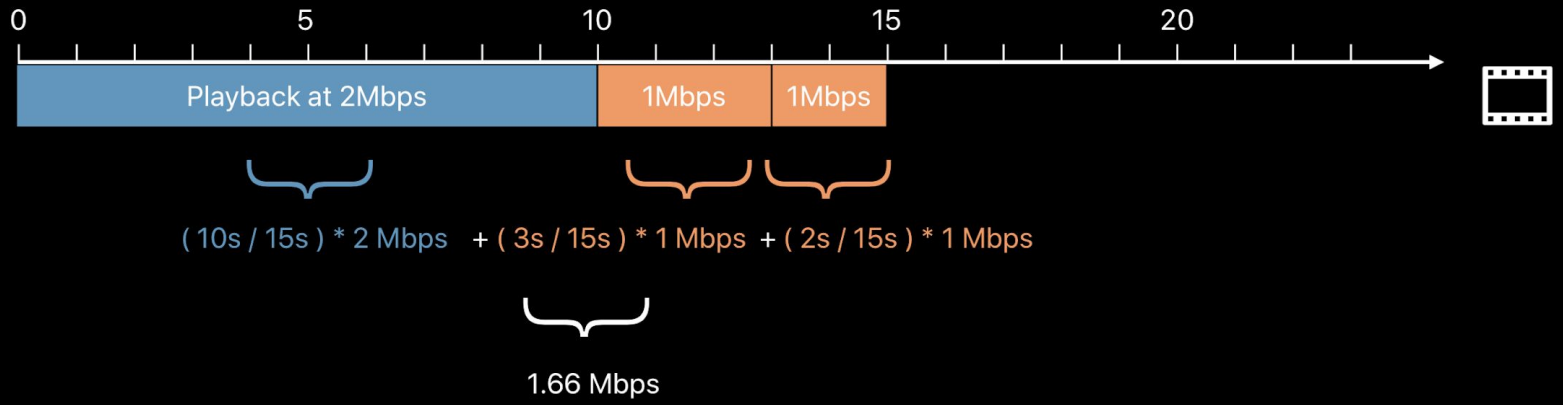
AVPlayerItemAccessLog

Provides a history of the playback session



Time-Weighted Indicated Bitrate

What was the overall media quality?



KPI	Unit
Time-weighted indicated bitrate	Mbps

```
//Computing time-weighted indicated bitrate from AVPlayerItem's access log

var timeWeightedIBR = 0.0
if let accessLog = playerItem.accessLog(), totalDurationWatched > 0 {
    for event in accessLog.events {
        if event.durationWatched > 0 && event.indicatedBitrate > 0 {
            let eventTimeWeight = event.durationWatched / totalDurationWatched
            timeWeightedIBR += event.indicatedBitrate * eventTimeWeight
        }
    }
}
```

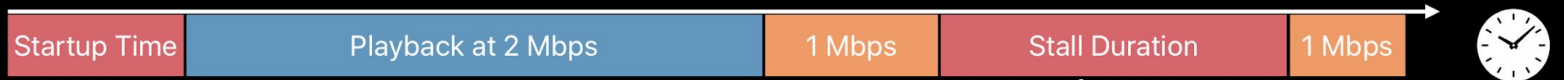
Playback Failure Percentage

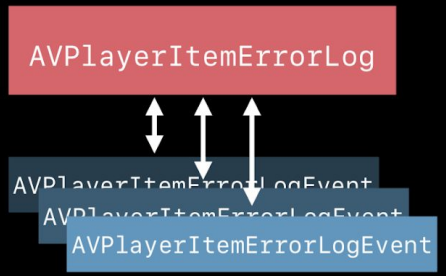
How often did the stream end due to an error?

KPI	Unit
Playback failure percentage	Percent of total sessions

AVPlayerItemErrorLog

Conveys failures with varying degrees of user impact






```
date = 2018-06-16 15:04:41
errorStatusCode = -12889
errorDomain = "CoreMediaErrorDomain"
errorComment = "No response for media file in 9.9767s"
```

`AVPlayerItemNewErrorLogEntry`

Demo: Key Performance Indicators for HLS

User Experience	Metric	KPI
Waiting for playback to start	Startup time	Startup time per session
Playback stalls	Stall count	Stall rate (normalized to duration watched)
Waiting for playback to resume	Stall duration	Stall duration to duration watched ratio
Overall stream quality	Indicated bitrate	Time-weighted indicated bitrate
Playback fails	Error	Playback failure percentage

Speeding Up HTTP Live Streaming

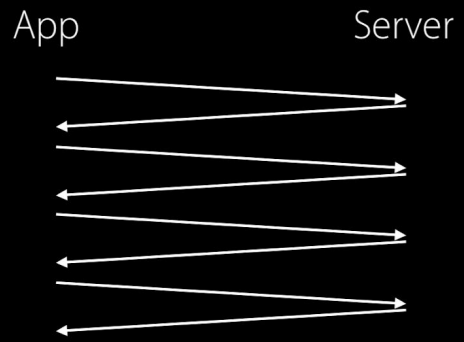
Consider the network round-trips

Retrieve master playlist

Retrieve content keys

Retrieve selected variant playlist

Retrieve segments



Can you do some of these before the user hits "play"?

Speeding Up HTTP Live Streaming

Preloading the Master Playlist

```
var asset = AVURLAsset(url: url)
asset.loadValuesAsynchronously(forKeys: ["duration"], completionHandler: nil)
```

Speeding Up HTTP Live Streaming

Compress Playlists

Compress Master Playlists and Variant Playlists with gzip

- Your server may be able to do this for you

Speeding Up HTTP Live Streaming

Preload segments before playback

```
// on title card
var playerItem = AVPlayerItem(asset: asset)
playerItem.preferredForwardBufferDuration = CMTime(value: 5, timescale: 1)
let player = AVPlayer()
let playerLayer = AVPlayerLayer(player: player) // keep the layer hidden
player.replaceCurrentItemWithPlayerItem(playerItem)
```

```
// as soon as playback begins, reset it to default
playerItem.preferredForwardBufferDuration = kCMTimeZero
```

Demo: Speeding Up HLS

1. Laying out video cards on home screen.
2. Preload segments(pre-cache) before playback(initial items).

Improving Initial Quality

AVPlayerLayer

Size your AVPlayerLayer appropriately and connect it to AVPlayer early

- Before bringing in playerItem

Set `AVPlayerLayer.contentsScale` on retina iOS devices

Improving Initial Quality

Buffering time to start playback (10-second segments)

Dimensions	Video bitrate	Buffering time @ 2 Mbit/sec
400 x 224	110 kbit/sec	0.55 sec
400 x 224	400 kbit/sec	2 sec
640 x 360	600 kbit/sec	3 sec
960 x 540	1800 kbit/sec	9 sec
1280 x 720	4500 kbit/sec	22.5 sec
1920 x 1080	11000 kbit/sec	55 sec

Improving Initial Quality

Use previous playback's statistics

```
if let lastAccessLogEvent = previousPlayerItem.accessLog()?.events.last {
    lastObservedBitrate = lastAccessLogEvent.observedBitrate
}
```

Controlling Initial Variant Selection

Use preferredPeakBitRate to restrict first choice

Dimensions	Video bitrate
1920 x 1080	11000 kbit/sec
1280 x 720	4500 kbit/sec
960 x 540	1800 kbit/sec
640 x 360	600 kbit/sec
400 x 224	400 kbit/sec
400 x 224	110 kbit/sec

```
// before playback
playerItem.preferredPeakBitRate = 2000

// shortly after playback starts
playerItem.preferredPeakBitRate = 0
```

Profile Your Code Too

Look for delays in your code, before AVFoundation is called

Don't wait for likelyToKeepUp notification before setting rate

Make sure you release AVPlayers and AVPlayerItems from old playback sessions

Use Allocations Instrument to check AVPlayer and AVPlayerItem lifespans

Suspend other network activity in your app during network playback

Questions?

Thank You!