Media

#WWDC16

What's New in HTTP Live Streaming

Session 504

Roger Pantos Media Systems Engineer
Jordan Schneider Media Systems Engineer

HTTP Live Streaming in 20 Seconds

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
```

HTTP Live Streaming in 20 Seconds

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
```

HTTP Live Streaming in 20 Seconds

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
```

HTTP Live Streaming in 20 Seconds

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
```

HTTP Live Streaming in 20 Seconds

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
```

HTTP Live Streaming in 20 Seconds

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
#EXTINF 10.001
segment3.ts
```

NEW

MPEG-4 Fragment Support in HLS

MPEG-4 Fragment Support in HLS

NEW

Extension of the familiar MPEG-4 file format (myMovie.mp4)

- An MPEG-4 file has a sample table followed by sample data

MPEG-4 Fragment Support in HLS

NEW

Extension of the familiar MPEG-4 file format (myMovie.mp4)

- An MPEG-4 file has a sample table followed by sample data

fMP4 “Fragments” divide myMovie.mp4 into separately decodable chunks

- Each with its own sample table and sample data

MPEG-4 Fragment Support in HLS

NEW

Extension of the familiar MPEG-4 file format (myMovie.mp4)

- An MPEG-4 file has a sample table followed by sample data

fMP4 "Fragments" divide myMovie.mp4 into separately decodable chunks

- Each with its own sample table and sample data

Adding fragmented MP4 as a supported Segment format to HLS spec

- Beta version available to Apple Developer Program members

MPEG-4 Fragment Support in HLS

NEW

Extension of the familiar MPEG-4 file format (myMovie.mp4)

- An MPEG-4 file has a sample table followed by sample data

fMP4 “Fragments” divide myMovie.mp4 into separately decodable chunks

- Each with its own sample table and sample data

Adding fragmented MP4 as a supported Segment format to HLS spec

- Beta version available to Apple Developer Program members

fMP4 Segments support the same set of features as TS

MPEG-4 Fragment Support in HLS

NEW

Extension of the familiar MPEG-4 file format (myMovie.mp4)

- An MPEG-4 file has a sample table followed by sample data

fMP4 "Fragments" divide myMovie.mp4 into separately decodable chunks

- Each with its own sample table and sample data

Adding fragmented MP4 as a supported Segment format to HLS spec

- Beta version available to Apple Developer Program members

fMP4 Segments support the same set of features as TS

Works on iOS, macOS, and tvOS

Benefits of fMP4 Segments

Allows a single media library to be delivered to multiple ecosystems

- HLS, MPEG-DASH, others
- Increases CDN cache efficiency

Benefits of fMP4 Segments

Allows a single media library to be delivered to multiple ecosystems

- HLS, MPEG-DASH, others
- Increases CDN cache efficiency

Common authoring and validation tools across ecosystems

Benefits of fMP4 Segments

Allows a single media library to be delivered to multiple ecosystems

- HLS, MPEG-DASH, others
- Increases CDN cache efficiency

Common authoring and validation tools across ecosystems

Higher network efficiency at low bit rates

How Does HLS Change?

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
```

How Does HLS Change?

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="moov.mp4"
#EXTINF 10.001
segment1.mp4
#EXTINF 10.001
segment2.mp4
```

Encrypting fMP4 Segments

Whole-segment encryption is same as TS

Encrypting fMP4 Segments

Whole-segment encryption is same as TS

Sample encryption uses part of ISO/IEC 23001:7 2016

- MPEG standard—“Common Encryption”
- 'cbcs' mode

Getting to Interoperability

Achieving a single media library

MPEG is working to define a "Common Media Application Format" (CMAF)

- Originally proposed by Apple and Microsoft
- Has attracted broad support at MPEG

Getting to Interoperability

Achieving a single media library

MPEG is working to define a "Common Media Application Format" (CMAF)

- Originally proposed by Apple and Microsoft
- Has attracted broad support at MPEG

Constrains the MPEG-4 Fragment definition (ISO 14496 Part 12)

- Requires unmixed audio and video delivery
- Requires that every video segment start with a key frame
- Requires precise segment alignment across bit rate variants
- And more

NEW

In-Playlist Timed Metadata

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	
Usually authored as text	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	
Usually authored as text	
Easily added to playlist or JSON	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	
Usually authored as text	
Easily added to playlist or JSON	
Static	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	
Usually authored as text	
Easily added to playlist or JSON	
Static	
Available immediately	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	e.g. ad marker
Usually authored as text	
Easily added to playlist or JSON	
Static	
Available immediately	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	e.g. ad marker
Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	
Static	
Available immediately	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	e.g. ad marker
Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Requires specialized tools
Static	
Available immediately	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	e.g. ad marker
Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Requires specialized tools
Static	Dynamic
Available immediately	

A Comparison of HLS Metadata

NEW

Static metadata	ID3 timed metadata
e.g. content title	e.g. ad marker
Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Requires specialized tools
Static	Dynamic
Available immediately	Delivered as played

A Comparison of HLS Metadata

Static metadata	ID3 timed metadata
e.g. content title	e.g. ad marker
Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Requires specialized tools
Static	Dynamic
Available immediately	Delivered as played

A Comparison of HLS Metadata

Static metadata	In-playlist timed metadata	ID3 timed metadata
e.g. content title		e.g. ad marker
Usually authored as text		Binary format (ID3)
Easily added to playlist or JSON		Requires specialized tools
Static		Dynamic
Available immediately		Delivered as played

A Comparison of HLS Metadata

Static metadata	In-playlist timed metadata	ID3 timed metadata
e.g. content title	e.g. ad marker	e.g. ad marker
Usually authored as text		Binary format (ID3)
Easily added to playlist or JSON		Requires specialized tools
Static		Dynamic
Available immediately		Delivered as played

A Comparison of HLS Metadata

Static metadata	In-playlist timed metadata	ID3 timed metadata
e.g. content title	e.g. ad marker	e.g. ad marker
Usually authored as text	Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON		Requires specialized tools
Static		Dynamic
Available immediately		Delivered as played

A Comparison of HLS Metadata

Static metadata	In-playlist timed metadata	ID3 timed metadata
e.g. content title	e.g. ad marker	e.g. ad marker
Usually authored as text	Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Easily added to playlist	Requires specialized tools
Static		Dynamic
Available immediately		Delivered as played

A Comparison of HLS Metadata

Static metadata	In-playlist timed metadata	ID3 timed metadata
e.g. content title	e.g. ad marker	e.g. ad marker
Usually authored as text	Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Easily added to playlist	Requires specialized tools
Static	Dynamic	Dynamic
Available immediately		Delivered as played

A Comparison of HLS Metadata

Static metadata	In-playlist timed metadata	ID3 timed metadata
e.g. content title	e.g. ad marker	e.g. ad marker
Usually authored as text	Usually authored as text	Binary format (ID3)
Easily added to playlist or JSON	Easily added to playlist	Requires specialized tools
Static	Dynamic	Dynamic
Available immediately	Available immediately	Delivered as played

In-Playlist Timed Metadata

NEW

In-Playlist Timed Metadata

NEW

Metadata is expressed as a date-based range inside a playlist

In-Playlist Timed Metadata

NEW

Metadata is expressed as a date-based range inside a playlist

Each range carries a content-defined set of attribute/value pairs

In-Playlist Timed Metadata

NEW

Metadata is expressed as a date-based range inside a playlist

Each range carries a content-defined set of attribute/value pairs

Ranges can be added and removed from live streams

In-Playlist Timed Metadata

#EXT-X-DATERANGE

```
#EXTM3U
#EXT-X-PROGRAM-DATE-TIME:2016-06-13T11:15:15Z
#EXT-X-DATERANGE:ID="ad3",START-DATE="2016-06-13T11:15:00Z",DURATION=20,X-AD-ID="1234",
X-AD-URL="http://ads.example.com/beacon3"

#EXTINF 10,
ad3.1.ts
#EXTINF 10,
ad3.2.ts
```

In-Playlist Timed Metadata

#EXT-X-DATERANGE

```
#EXTM3U
#EXT-X-PROGRAM-DATE-TIME:2016-06-13T11:15:15Z
#EXT-X-DATERANGE:ID="ad3",START-DATE="2016-06-13T11:15:00Z",DURATION=20,X-AD-ID="1234",
X-AD-URL="http://ads.example.com/beacon3"

#EXTINF 10,
ad3.1.ts
#EXTINF 10,
ad3.2.ts
```

In-Playlist Timed Metadata

#EXT-X-DATERANGE

```
#EXTM3U
#EXT-X-PROGRAM-DATE-TIME:2016-06-13T11:15:15Z
#EXT-X-DATERANGE:ID="ad3",START-DATE="2016-06-13T11:15:00Z",DURATION=20,X-AD-ID="1234",
X-AD-URL="http://ads.example.com/beacon3"

#EXTINF 10,
ad3.1.ts
#EXTINF 10,
ad3.2.ts
```

In-Playlist Timed Metadata

Content authoring

In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists

In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists

Can be authored with media, or added in post-production

In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists

Can be authored with media, or added in post-production

Spec includes bindings for SCTE-35 tags

In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists

Can be authored with media, or added in post-production

Spec includes bindings for SCTE-35 tags

mediastreamvalidator support

In-Playlist Timed Metadata

Playback

In-Playlist Timed Metadata

Playback

AVFoundation API for obtaining DATE-RANGE info

In-Playlist Timed Metadata

Playback

AVFoundation API for obtaining DATE-RANGE info

All timed metadata available as soon as playlist is loaded

In-Playlist Timed Metadata

Playback

AVFoundation API for obtaining DATE-RANGE info

All timed metadata available as soon as playlist is loaded

Notification when list changes

In-Playlist Timed Metadata

AVPlayerItemMetadataCollector

```swift
let asset = AVURLAsset(url: url)

let playerItem = AVPlayerItem(asset: asset)

let collector = AVPlayerItemMetadataCollector()

collector.set(delegate: self, queue: mainQueue)

playerItem.addMediaDataCollector(collector)
```

NEW

Offline HLS Playback

Jordan Schneider Media Systems Engineer

What is Offline HLS?

NEW

What is Offline HLS?

NEW

HLS without network connectivity

What is Offline HLS?

NEW

HLS without network connectivity

Uses your existing media library

What is Offline HLS?

NEW

HLS without network connectivity

Uses your existing media library

Offline FairPlay Streaming

What is Offline HLS?

NEW

HLS without network connectivity

Uses your existing media library

Offline FairPlay Streaming

Downloads in the background

What is Offline HLS?

NEW

HLS without network connectivity

Uses your existing media library

Offline FairPlay Streaming

Downloads in the background

Plays while download is in progress

Should You Use Offline HLS?

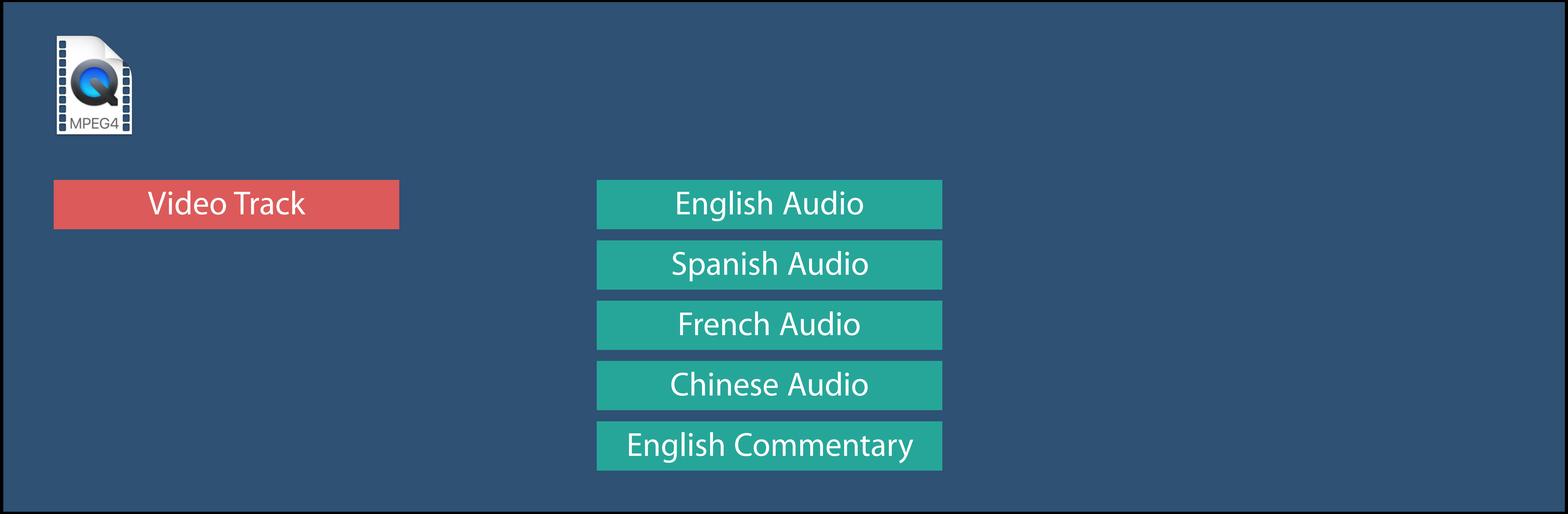
Should You Use Offline HLS?

Advantages of Offline HLS

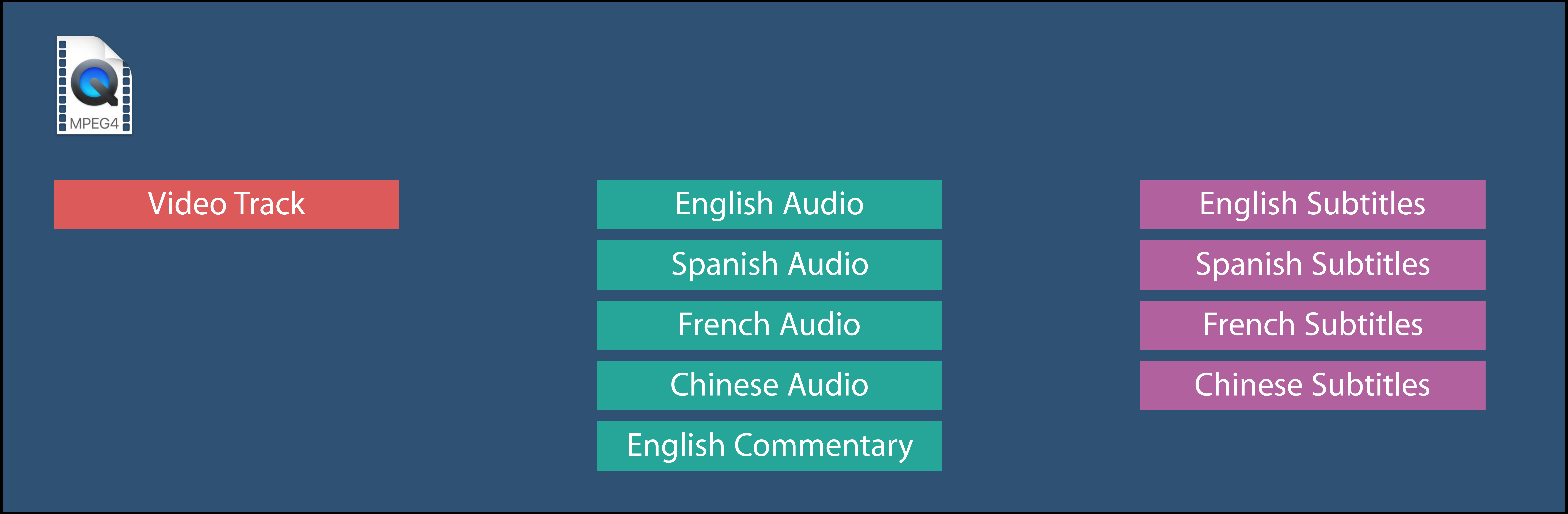
Advantages of Offline HLS

MPEG4
Video Track

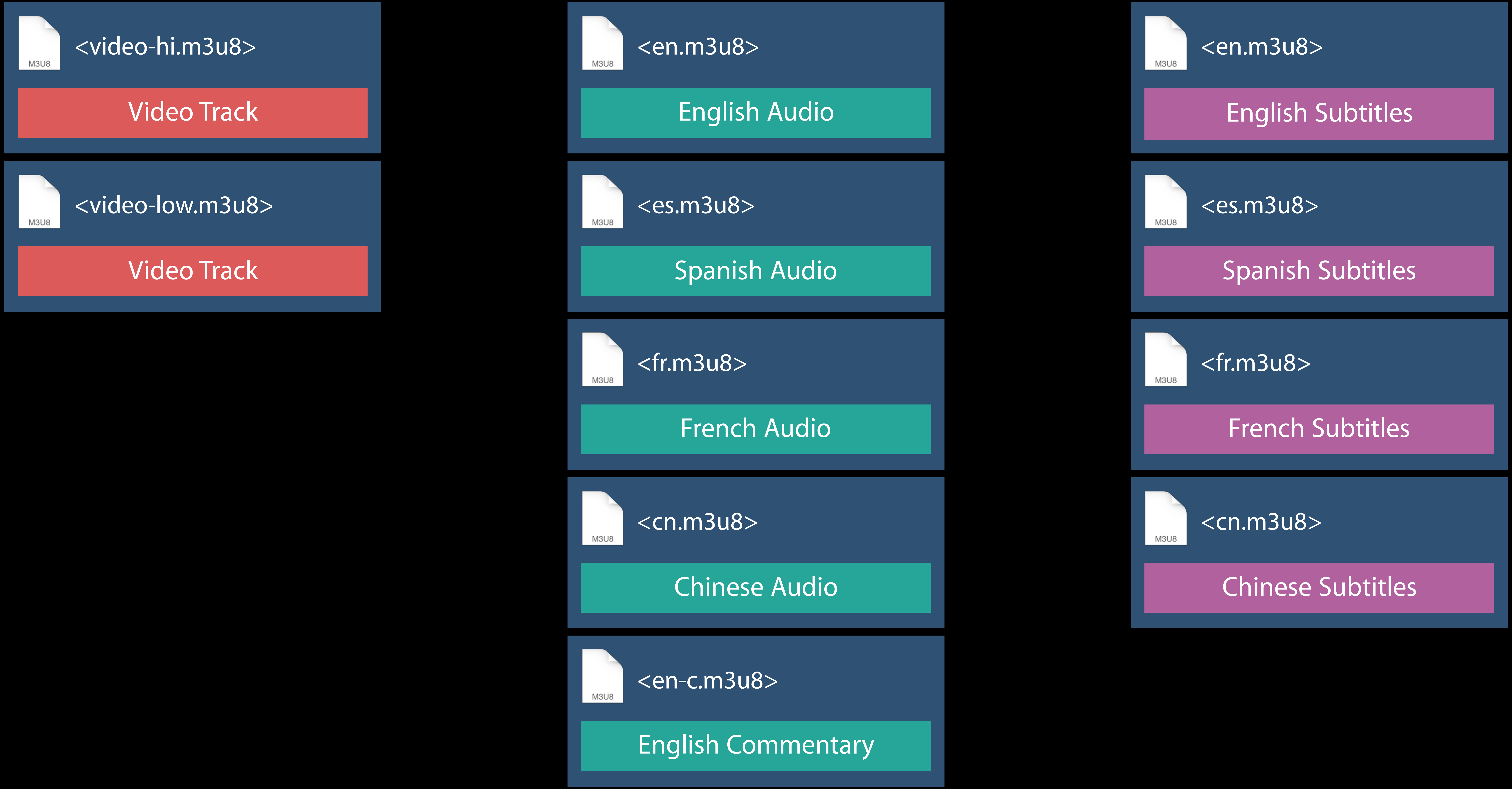
Advantages of Offline HLS



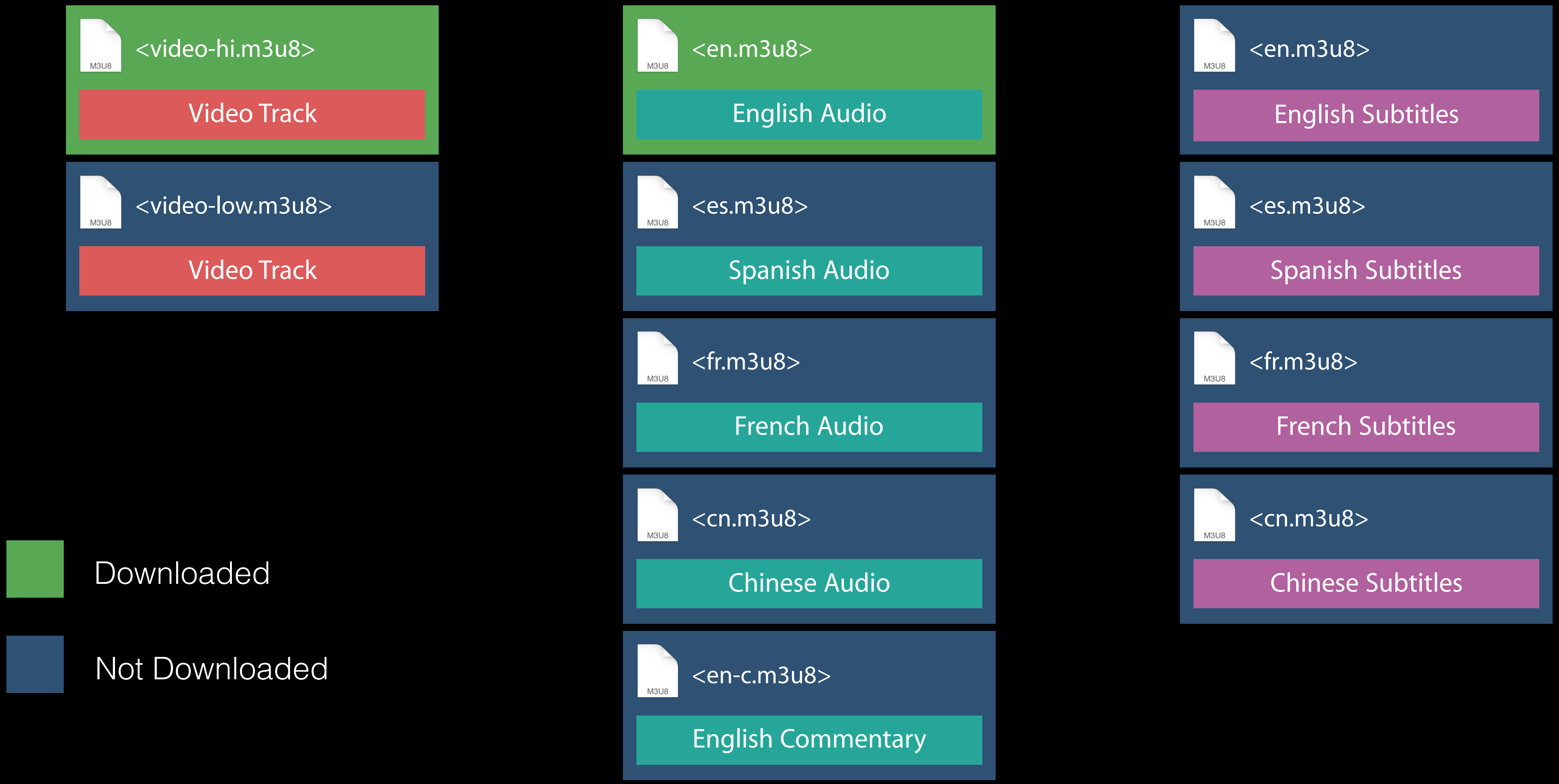
Advantages of Offline HLS



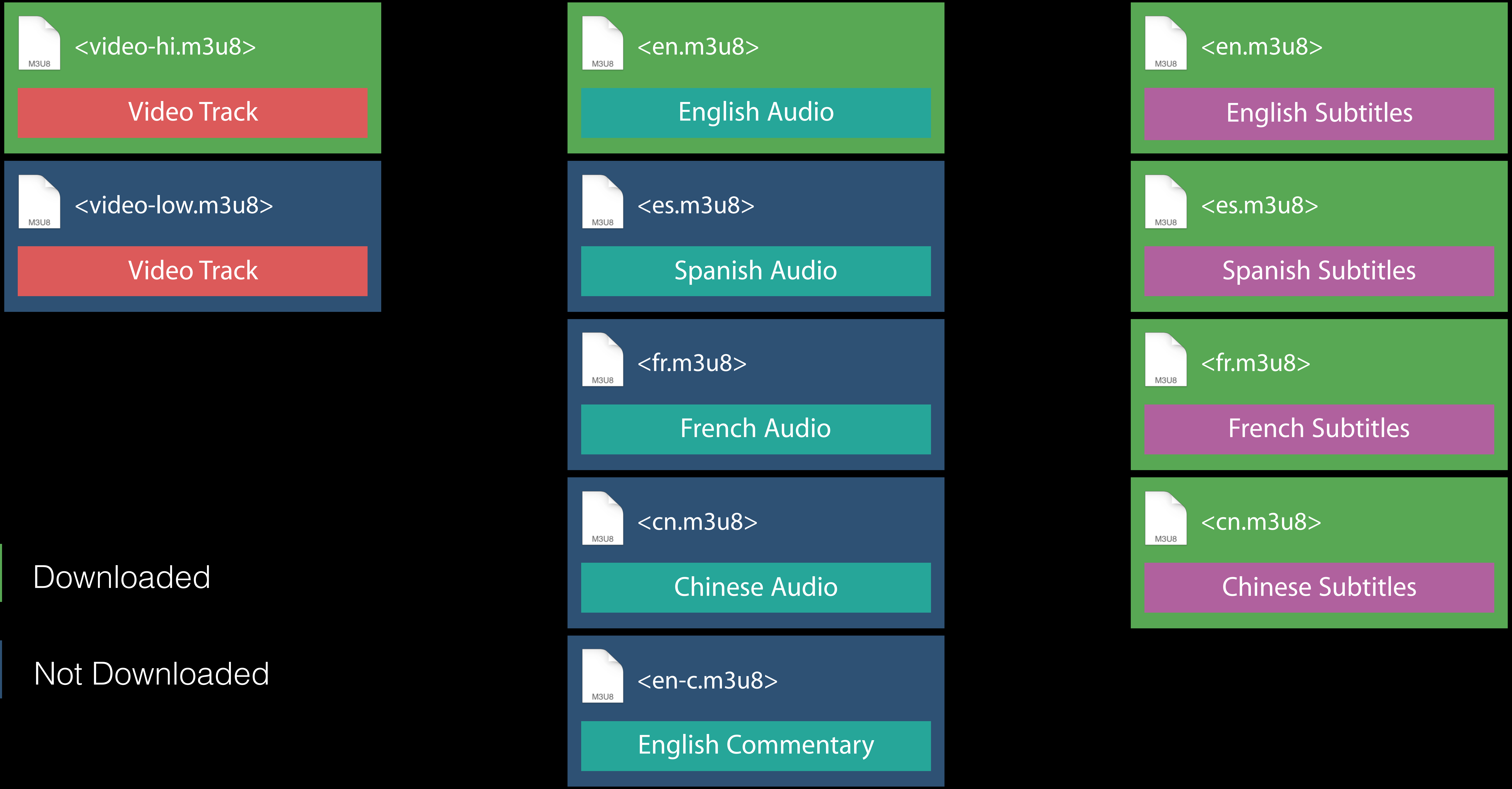
Advantages of Offline HLS



Advantages of Offline HLS



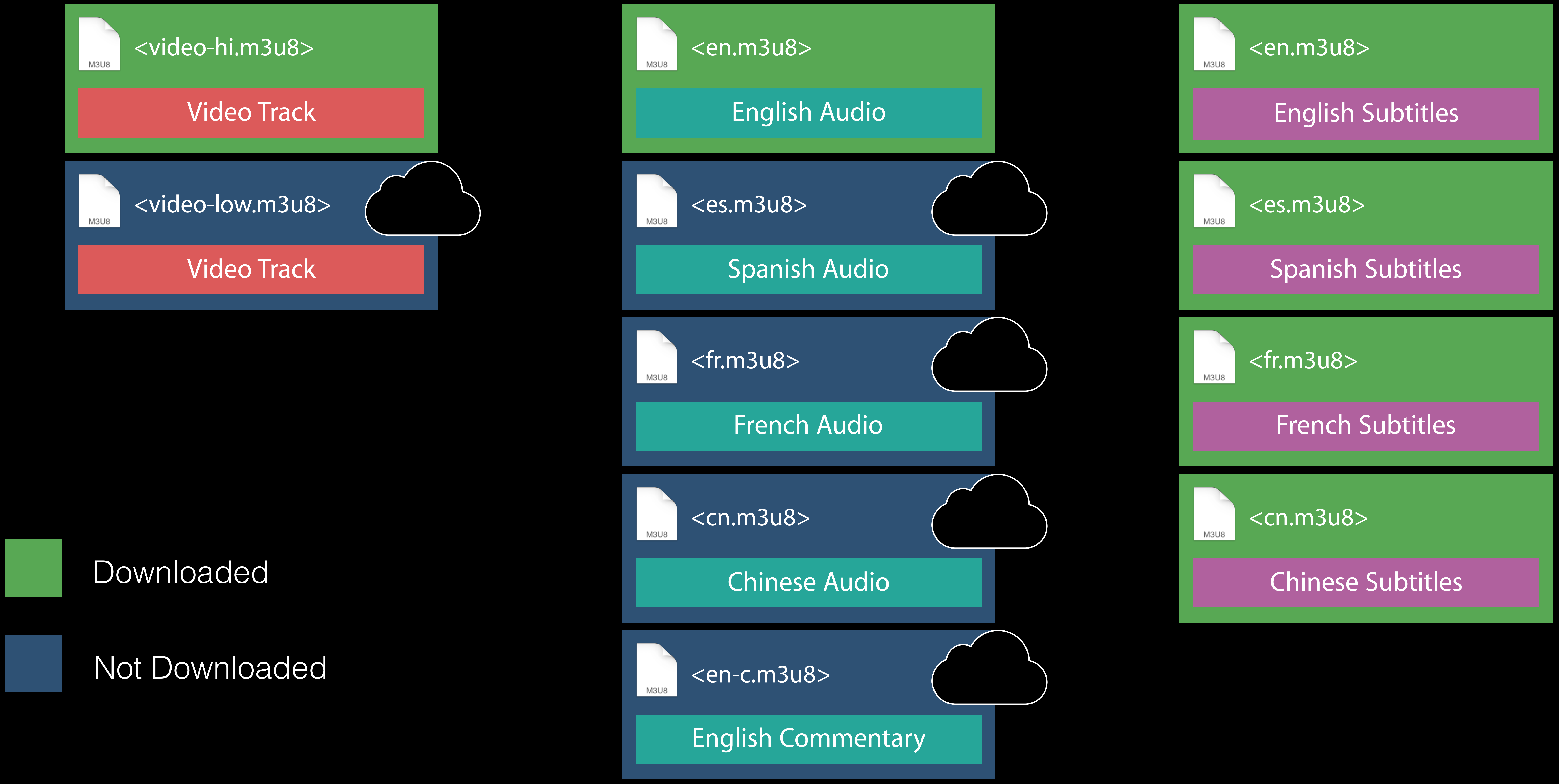
Advantages of Offline HLS



🟩 Downloaded

🟦 Not Downloaded

Advantages of Offline HLS



AVAssetDownloadTask

AVAssetDownloadTask

Inherits features of URLSession

AVAssetDownloadTask

Inherits features of URLSession

- Background downloading

AVAssetDownloadTask

Inherits features of URLSession

- Background downloading

Media selection

AVAssetDownloadTask

Inherits features of URLSession

- Background downloading

Media selection

Quality selection

AVAssetDownloadTask

```
public class AVAssetDownloadTask: URLSessionTask {
   ...
}
public class AVAssetDownloadURLSession: URLSession {
   func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                              assetArtworkData artworkData: Data?,
                              options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}
public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
```

AVAssetDownloadTask

```
public class AVAssetDownloadTask: URLSessionTask {
   ...
}
public class AVAssetDownloadURLSession: URLSession {
   func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                              assetArtworkData artworkData: Data?,
                              options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}
public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
```

AVAssetDownloadTask

```swift
public class AVAssetDownloadTask: URLSessionTask {
   ...
}
public class AVAssetDownloadURLSession: URLSession {
   func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                              assetArtworkData artworkData: Data?,
                              options: [String : AnyObject]? = [:])
       -> AVAssetDownloadTask?
}
public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
```

AVAssetDownloadTask

```
public class AVAssetDownloadTask: URLSessionTask {
   ...
}
public class AVAssetDownloadURLSession: URLSession {
   func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                              assetArtworkData artworkData: Data?,
                              options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}
public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
```

AVAssetDownloadTask

```
public class AVAssetDownloadTask: URLSessionTask {
   ...
}
public class AVAssetDownloadURLSession: URLSession {
   func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                              assetArtworkData artworkData: Data?,
                              options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}
public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
```

AVAssetDownloadTask

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset

```swift
// Setup and Start AVAssetDownloadTask
func setupAssetDownload() {
   let hlsAsset = AVURLAsset(url: assetURL)
   let backgroundConfiguration = URLSessionConfiguration.background(
      withIdentifier: "assetDownloadConfigurationIdentifier")
   let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
      assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

   // Download a Movie at 2 mbps
   let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
      assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
   assetDownloadTask.resume()
}
```

```swift
// Setup and Start AVAssetDownloadTask
func setupAssetDownload() {
   let hlsAsset = AVURLAsset(url: assetURL)
   let backgroundConfiguration = URLSessionConfiguration.background(
      withIdentifier: "assetDownloadConfigurationIdentifier")
   let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
      assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

   // Download a Movie at 2 mbps
   let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
      assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
   assetDownloadTask.resume()
}
```

```swift
// Setup and Start AVAssetDownloadTask
func setupAssetDownload() {
   let hlsAsset = AVURLAsset(url: assetURL)
   let backgroundConfiguration = URLSessionConfiguration.background(
      withIdentifier: "assetDownloadConfigurationIdentifier")
   let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
      assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

   // Download a Movie at 2 mbps
   let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
      assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
   assetDownloadTask.resume()
}
```

```swift
// Setup and Start AVAssetDownloadTask
func setupAssetDownload() {
   let hlsAsset = AVURLAsset(url: assetURL)
   let backgroundConfiguration = URLSessionConfiguration.background(
      withIdentifier: "assetDownloadConfigurationIdentifier")
   let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
      assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

   // Download a Movie at 2 mbps
   let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
      assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
   assetDownloadTask.resume()
}
```

```swift
// Setup and Start AVAssetDownloadTask
func setupAssetDownload() {
   let hlsAsset = AVURLAsset(url: assetURL)
   let backgroundConfiguration = URLSessionConfiguration.background(
      withIdentifier: "assetDownloadConfigurationIdentifier")
   let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
      assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

   // Download a Movie at 2 mbps
   let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
      assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
   assetDownloadTask.resume()
}
```

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset

Monitoring the Download

AVAssetDownloadDelegate

```
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
      timeRangeExpectedToLoad: CMTimeRange)

   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL)
}
```

Monitoring the Download

AVAssetDownloadDelegate

```
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
      timeRangeExpectedToLoad: CMTimeRange)

   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL)
}
```

Monitoring the Download

AVAssetDownloadDelegate

```
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
      timeRangeExpectedToLoad: CMTimeRange)

   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL)
}
```

Monitoring the Download

AVAssetDownloadDelegate

```swift
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
      timeRangeExpectedToLoad: CMTimeRange)

   optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL)
}
```

```swift
// In-progress Delegate Methods
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
      timeRangeExpectedToLoad: CMTimeRange) {
      // Convert loadedTimeRanges to CMTimeRanges
      var percentComplete = 0.0
      for value in loadedTimeRanges {
         let loadedTimeRange: CMTimeRange = value.timeRangeValue
         percentComplete += CMTimeGetSeconds(loadedTimeRange.duration) /
            CMTimeGetSeconds(timeRangeExpectedToLoad.duration)
      }
      percentComplete *= 100
      print("percent complete: \(percentComplete)")
   }
}
```

```swift
// Restore Tasks on App Launch
class MyAppDelegate: UIResponder, UIApplicationDelegate {
   func application(_ application: UIApplication,
      didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]? = [:]) -> Bool {
      let configuration = URLSessionConfiguration.background(withIdentifier:
         "assetDownloadConfigurationIdentifier")
      let session = URLSession(configuration: configuration)
      session.getAllTasks { tasks in
         for task in tasks {
            if let assetDownloadTask = task as? AVAssetDownloadTask {
               // restore progress indicators, state, etc...
            }
         }
      }
   }
}
```

```swift
// Restore Tasks on App Launch
class MyAppDelegate: UIResponder, UIApplicationDelegate {
   func application(_ application: UIApplication,
      didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]? = [:]) -> Bool {
         let configuration = URLSessionConfiguration.background(withIdentifier:
            "assetDownloadConfigurationIdentifier")
         let session = URLSession(configuration: configuration)
         session.getAllTasks { tasks in
            for task in tasks {
               if let assetDownloadTask = task as? AVAssetDownloadTask {
                  // restore progress indicators, state, etc...
               }
            }
         }
   }
}
```

```swift
// Restore Tasks on App Launch
class MyAppDelegate: UIResponder, UIApplicationDelegate {
   func application(_ application: UIApplication,
      didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]? = [:]) -> Bool {
      let configuration = URLSessionConfiguration.background(withIdentifier:
         "assetDownloadConfigurationIdentifier")
      let session = URLSession(configuration: configuration)
      session.getAllTasks { tasks in
         for task in tasks {
            if let assetDownloadTask = task as? AVAssetDownloadTask {
               // restore progress indicators, state, etc...
            }
         }
      }
   }
}
```

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset

```swift
// Store Location of Downloaded Asset
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   // called whenever anything is deposited at location
   func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL) {
      // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
      let locationToSave = location.relativePath!
      // Stash Away This Location
      ...
   }
}
```

```swift
// Store Location of Downloaded Asset
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    // called whenever anything is deposited at location
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
        didFinishDownloadingTo location: URL) {
        // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
        let locationToSave = location.relativePath!
        // Stash Away This Location
        ...
    }
}
```

```swift
// Store Location of Downloaded Asset
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   // called whenever anything is deposited at location
   func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL) {
      // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
      let locationToSave = location.relativePath!
      // Stash Away This Location
      ...
   }
}
```

```swift
// Store Location of Downloaded Asset
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   // called whenever anything is deposited at location
   func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL) {
      // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
      let locationToSave = location.relativePath!
      // Stash Away This Location
      ...
   }
}
```

```swift
// Store Location of Downloaded Asset
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   // called whenever anything is deposited at location
   func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
      didFinishDownloadingTo location: URL) {
      // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
      let locationToSave = location.relativePath!
      // Stash Away This Location
      ...
   }
}
```

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset

```swift
 // Download Additional Media Selections
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, task: URLSessionTask,
      didCompleteWithError error: NSError?) {
      guard error == nil else { return }
      let assetURLSession = session as! AVAssetDownloadURLSession
      let assetDownloadTask = task as! AVAssetDownloadTask
      let audioGroup: AVMediaSelectionGroup = ...
      let spanishOption: AVMediaSelectionOption = ...

      guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
         AVMutableMediaSelection else { return }
      additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
      let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
         assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
         options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
      newAssetDownloadTask.resume()
   }
}
```

```swift
 // Download Additional Media Selections
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, task: URLSessionTask,
      didCompleteWithError error: NSError?) {
      guard error == nil else { return }
      let assetURLSession = session as! AVAssetDownloadURLSession
      let assetDownloadTask = task as! AVAssetDownloadTask
      let audioGroup: AVMediaSelectionGroup = ...
      let spanishOption: AVMediaSelectionOption = ...

      guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
         AVMutableMediaSelection else { return }
      additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
      let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
         assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
         options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
      newAssetDownloadTask.resume()
   }
}
```

```swift
 // Download Additional Media Selections
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, task: URLSessionTask,
      didCompleteWithError error: NSError?) {
      guard error == nil else { return }
      let assetURLSession = session as! AVAssetDownloadURLSession
      let assetDownloadTask = task as! AVAssetDownloadTask
      let audioGroup: AVMediaSelectionGroup = ...
      let spanishOption: AVMediaSelectionOption = ...

      guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
         AVMutableMediaSelection else { return }
      additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
      let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
         assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
         options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
      newAssetDownloadTask.resume()
   }
}
```

```swift
 // Download Additional Media Selections
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, task: URLSessionTask,
      didCompleteWithError error: NSError?) {
      guard error == nil else { return }
      let assetURLSession = session as! AVAssetDownloadURLSession
      let assetDownloadTask = task as! AVAssetDownloadTask
      let audioGroup: AVMediaSelectionGroup = ...
      let spanishOption: AVMediaSelectionOption = ...

      guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
         AVMutableMediaSelection else { return }
      additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
      let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
         assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
         options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
      newAssetDownloadTask.resume()
   }
}
```

```swift
 // Download Additional Media Selections
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, task: URLSessionTask,
      didCompleteWithError error: NSError?) {
      guard error == nil else { return }
      let assetURLSession = session as! AVAssetDownloadURLSession
      let assetDownloadTask = task as! AVAssetDownloadTask
      let audioGroup: AVMediaSelectionGroup = ...
      let spanishOption: AVMediaSelectionOption = ...

      guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
         AVMutableMediaSelection else { return }
      additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
      let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
         assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
         options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
      newAssetDownloadTask.resume()
   }
}
```

```swift
 // Download Additional Media Selections
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
   func urlSession(_ session: URLSession, task: URLSessionTask,
      didCompleteWithError error: NSError?) {
      guard error == nil else { return }
      let assetURLSession = session as! AVAssetDownloadURLSession
      let assetDownloadTask = task as! AVAssetDownloadTask
      let audioGroup: AVMediaSelectionGroup = ...
      let spanishOption: AVMediaSelectionOption = ...

      guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
         AVMutableMediaSelection else { return }
      additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
      let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
         assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
         options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
      newAssetDownloadTask.resume()
   }
}
```

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset

```swift
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
   assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)
```

```swift
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
   assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)
```

```swift
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
   assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)
```

One Week Later…

```swift
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
   assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)
```

```swift
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
   assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 3) When Your Original AVURLAsset Instantiated with a Network URL is No Longer Available
let fileURL = URL(fileURLWithPath: self.savedAssetDownloadLocation)
let asset = AVURLAsset(url: fileURL)
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 4) Augmenting a Download with Additional Media Selection
let task = session.makeAssetDownloadTask(asset: playerItem.asset as! AVURLAsset,
   assetTitle: "My Movie", assetArtworkData: nil,
   options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])
```

```swift
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
   assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 3) When Your Original AVURLAsset Instantiated with a Network URL is No Longer Available
let fileURL = URL(fileURLWithPath: self.savedAssetDownloadLocation)
let asset = AVURLAsset(url: fileURL)
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 4) Augmenting a Download with Additional Media Selection
let task = session.makeAssetDownloadTask(asset: playerItem.asset as! AVURLAsset,
   assetTitle: "My Movie", assetArtworkData: nil,
   options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])
```

Query for Cached Media Selections

AVAssetCache

```
public class AVURLAsset {
   public var assetCache: AVAssetCache? { get }
}

public class AVAssetCache {
   public var isPlayableOffline: Bool { get }

   public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup)
      -> [AVMediaSelectionOption]
}
```

Query for Cached Media Selections

AVAssetCache

```swift
public class AVURLAsset {
    public var assetCache: AVAssetCache? { get }
}


public class AVAssetCache {
    public var isPlayableOffline: Bool { get }

    public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup)
        -> [AVMediaSelectionOption]
}
```

Query for Cached Media Selections

AVAssetCache

```swift
public class AVURLAsset {
   public var assetCache: AVAssetCache? { get }
}

public class AVAssetCache {
   public var isPlayableOffline: Bool { get }

   public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup)
      -> [AVMediaSelectionOption]
}
```

Query for Cached Media Selections

AVAssetCache

```
public class AVURLAsset {
   public var assetCache: AVAssetCache? { get }
}

public class AVAssetCache {
   public var isPlayableOffline: Bool { get }

   public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup)
      -> [AVMediaSelectionOption]
}
```

AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset

Securing Your Offline Content

Offline FPS content protection

Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS

Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS

AVFoundation handles packaging keys for offline storage

Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS

AVFoundation handles packaging keys for offline storage

App is expected to store its own keys to disk

Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS

AVFoundation handles packaging keys for offline storage

App is expected to store its own keys to disk

Support for offline keys is opt-in in the key server

Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS

AVFoundation handles packaging keys for offline storage

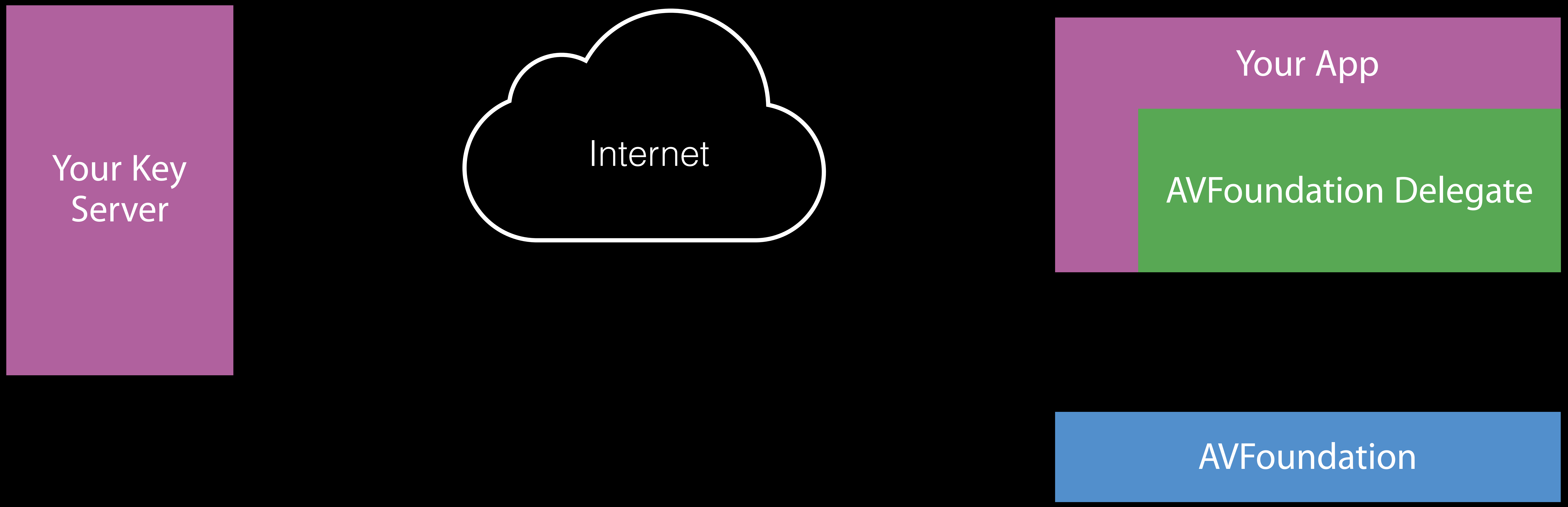
App is expected to store its own keys to disk

Support for offline keys is opt-in in the key server

All offline FPS Keys must be declared as EXT-X-SESSION-KEYS

Offline FPS

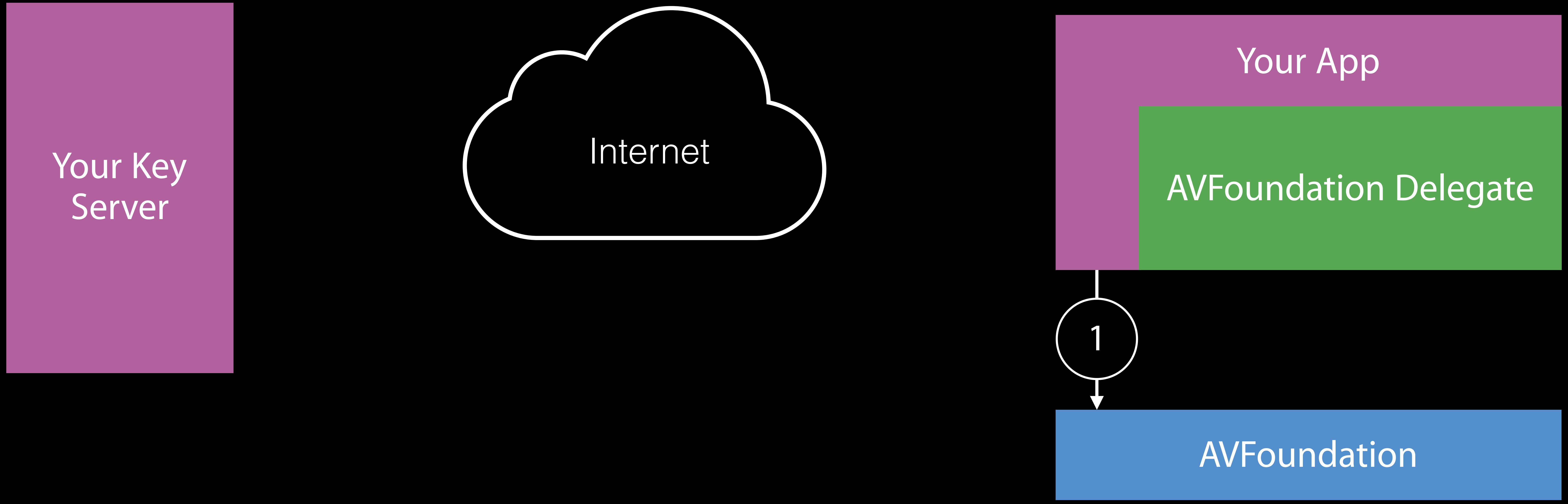
Request flow



Offline FPS

Request flow

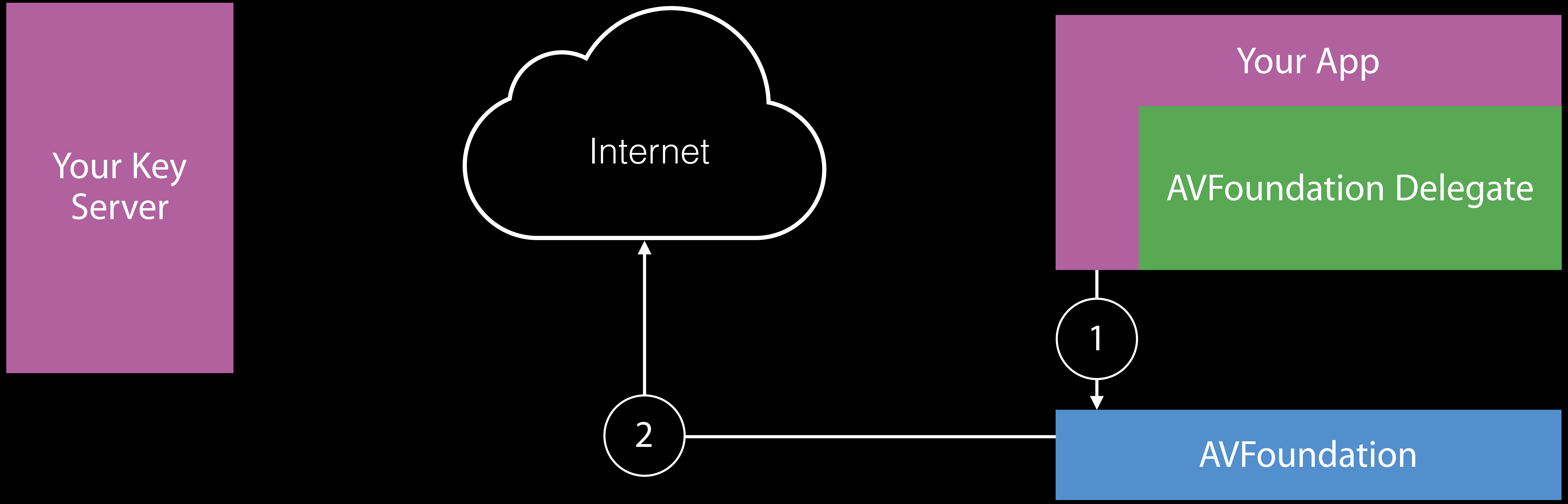
1 Your app asks AVFoundation to download or play your protected HLS asset



Offline FPS

Request flow

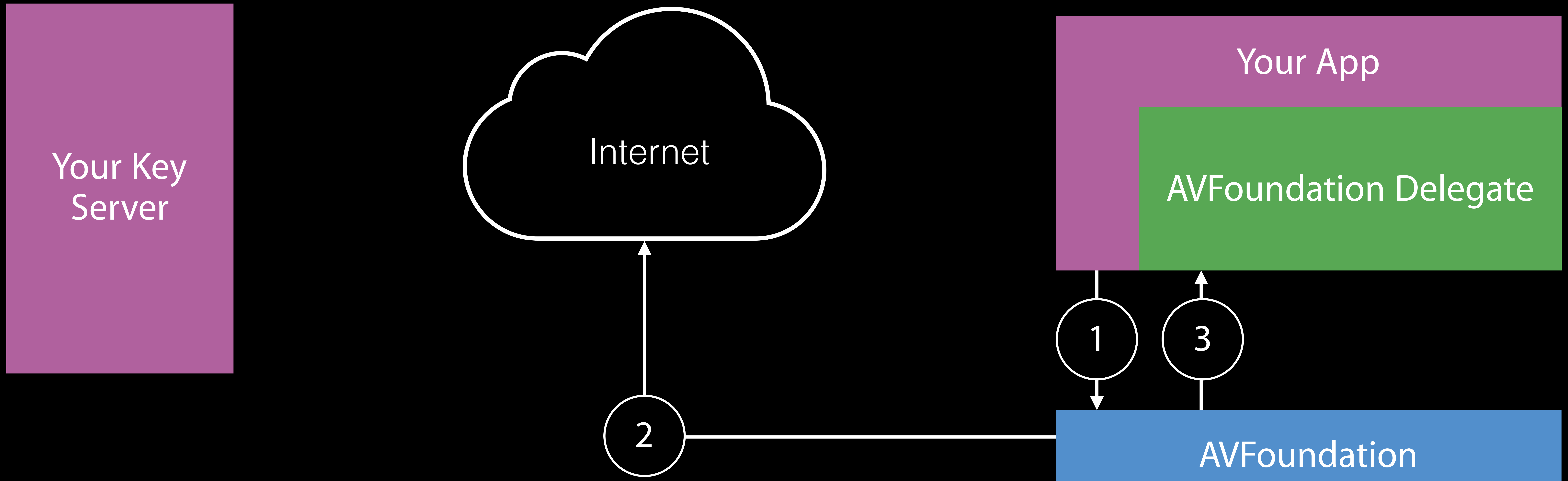
2 AVFoundation will download your m3u8 playlist containing the KEY tag



Offline FPS

Request flow

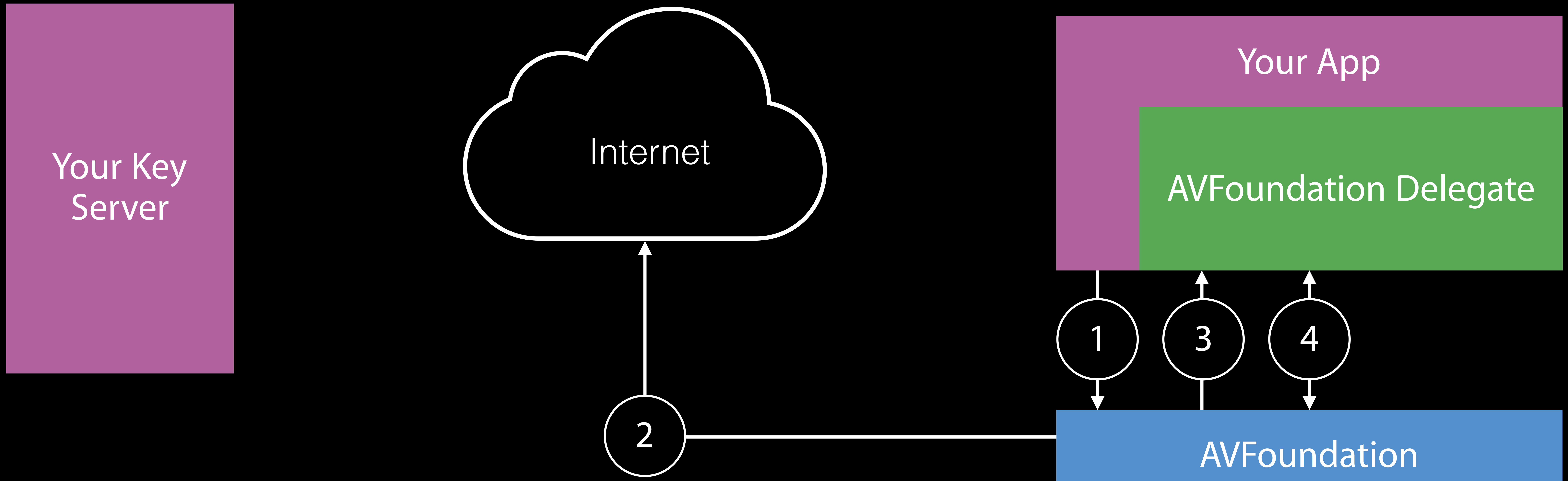
3 AVFoundation will call your app delegate to request the key



Offline FPS

Request flow

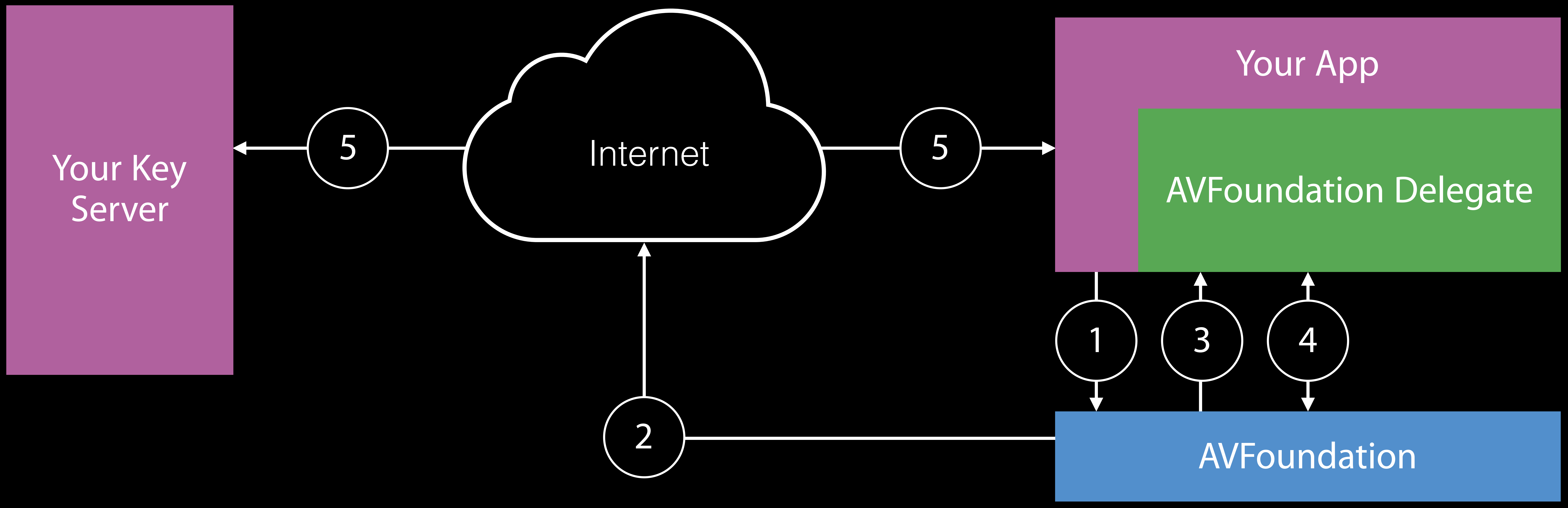
4 Your app delegate calls AVFoundation to create an FPS Server Playback Context request



Offline FPS

Request flow

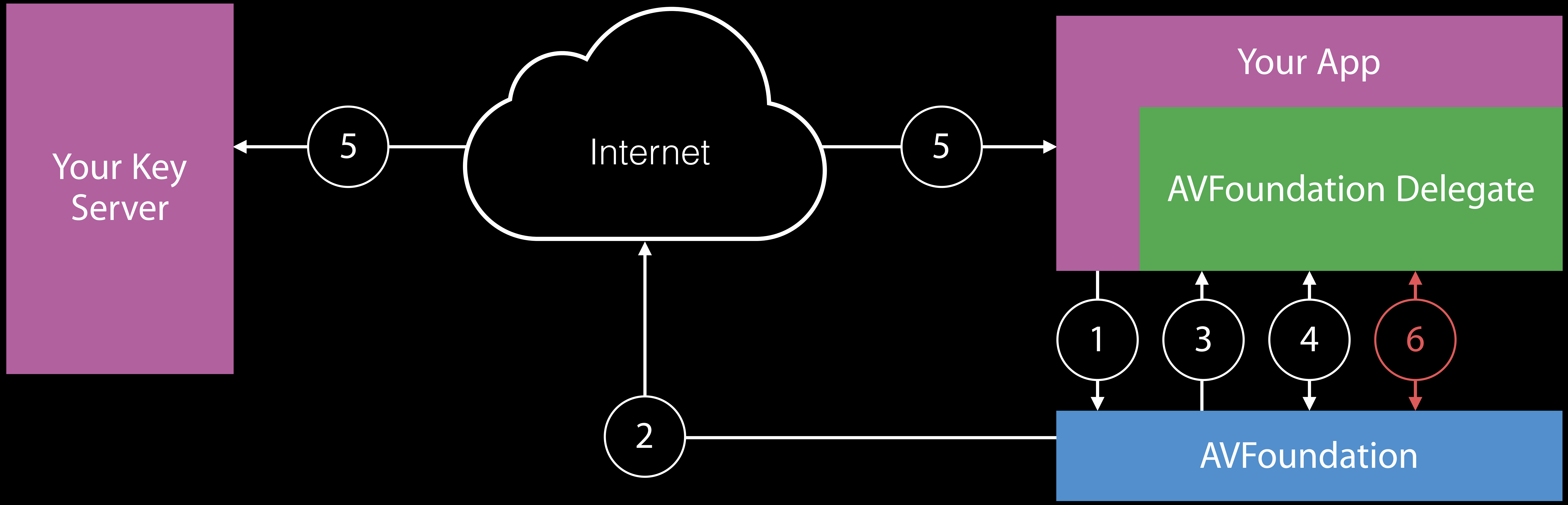
5 Your app delegate sends the FPS SPC to your key server, which returns a FairPlay Content Key Context



Offline FPS

Request flow

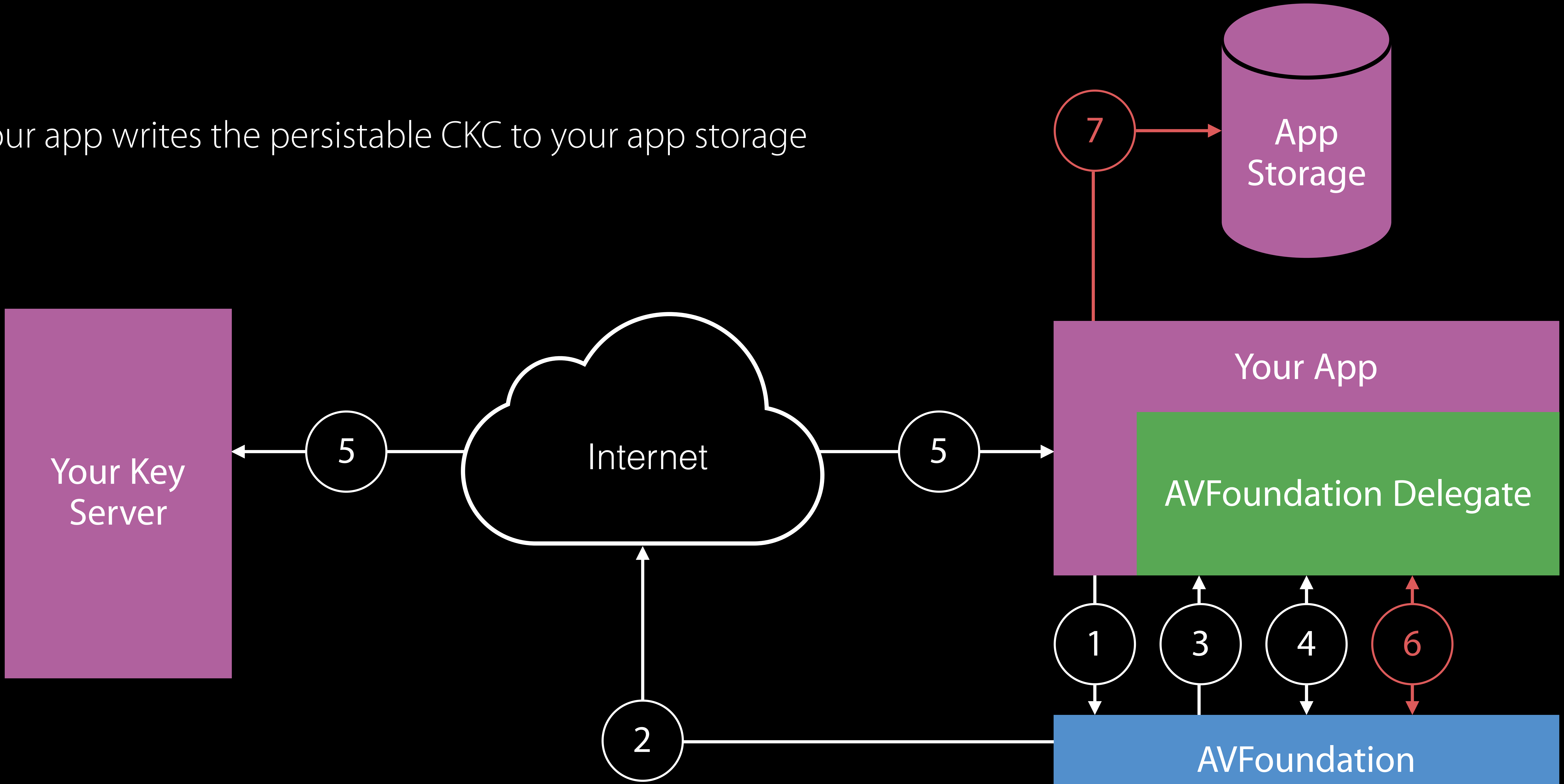
6 Your app delegate sends the CKC to AVFoundation to create a persistable CKC



Offline FPS

Request flow

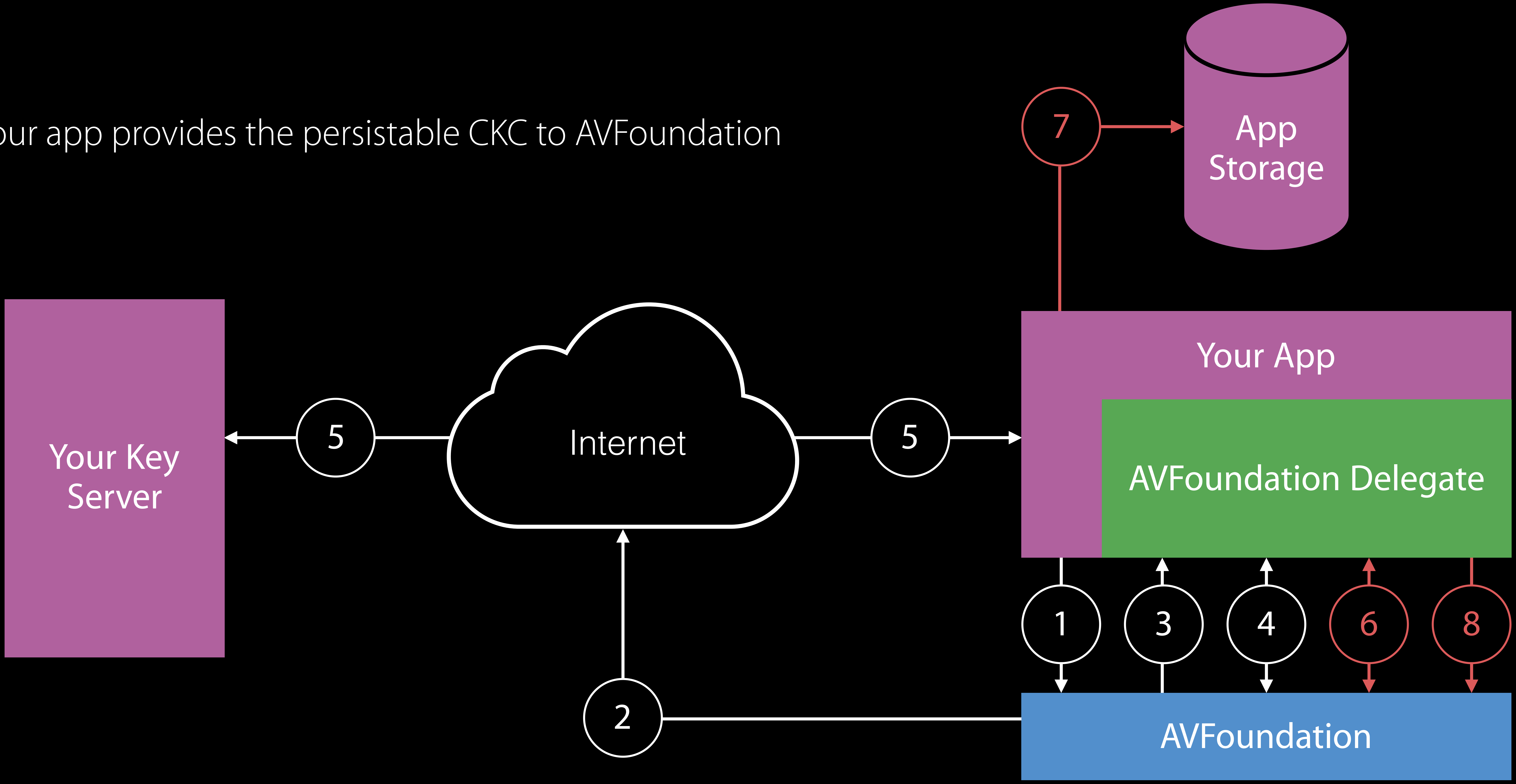
7 Your app writes the persistable CKC to your app storage



Offline FPS

Request flow

8 Your app provides the persistable CKC to AVFoundation



```swift
// FPS Key Fetch
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID, options: nil)
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      loadingRequest.dataRequest!.respond(with: contentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID, options: nil)
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      loadingRequest.dataRequest!.respond(with: contentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID, options: nil)
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      loadingRequest.dataRequest!.respond(with: contentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID, options: nil)
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      loadingRequest.dataRequest!.respond(with: contentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

Offline FPS

Client changes

```
public class AVAssetResourceLoadingRequest {
   public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
      options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}


public let AVStreamingKeyDeliveryPersistentContentKeyType: String
public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
```

Offline FPS

Client changes

```
public class AVAssetResourceLoadingRequest {
   public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
      options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}


public let AVStreamingKeyDeliveryPersistentContentKeyType: String
public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
```

Offline FPS

Client changes

```
public class AVAssetResourceLoadingRequest {
   public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
      options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}

public let AVStreamingKeyDeliveryPersistentContentKeyType: String
public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
```

Offline FPS

Client changes

```
public class AVAssetResourceLoadingRequest {
   public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
      options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}


public let AVStreamingKeyDeliveryPersistentContentKeyType: String
public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
   shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
         contentIdentifier: contentID,
         options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
      // send serverPlaybackContext to server to get contentKeyContext
      ...
      let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
         options: nil, error: nil)
      persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource
   loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
   if loadingRequest.request.url?.scheme == "skd" {
      let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
      loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
      loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
      loadingRequest.finishLoading()
      return true
   }
   return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource
    loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource
    loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
```

```swift
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource
    loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
```

Asset Management

Best practices

Asset Management

Best practices

Clean up unneeded assets on disk

Asset Management

Best practices

Clean up unneeded assets on disk

- Cancelled downloads remain on disk

Asset Management

Best practices

Clean up unneeded assets on disk

- Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Asset Management

Best practices

Clean up unneeded assets on disk

- Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Asset Management

Best practices

Clean up unneeded assets on disk

- Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Be prepared for the system to delete your assets to reclaim disk space

Asset Management

Best practices

Clean up unneeded assets on disk

- Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Be prepared for the system to delete your assets to reclaim disk space

Keep downloaded assets at the system-provided location

Asset Management

Best practices

Clean up unneeded assets on disk

- Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Be prepared for the system to delete your assets to reclaim disk space

Keep downloaded assets at the system-provided location

If server asset changes, host the modified asset at a new URL

Summary

Summary

MPEG-4 Fragment support

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata

Offline HLS playback

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata

Offline HLS playback

- Configurable media downloading

Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata

Offline HLS playback

- Configurable media downloading
- Industrial strength content protection

More Information

https://developer.apple.com/wwdc16/504

Related Sessions

Advances in AVFoundation Playback	Mission	Wednesday 9:00AM
AVKit on tvOS	Presidio	Friday 11:00AM
HTTP Live Streaming Authoring and Validation	Video	Watch on Demand

Labs

HTTP Live Streaming Lab	Graphics, Games, and Media Lab C	Wednesday 4:00PM
AVFoundation / HTTP Live Streaming Lab	Graphics, Games, and Media Lab D	Thursday 9:00AM
AVKit Lab	Graphics, Games, and Media Lab C	Friday 1:00PM

WWDC16