

Working with HEIF and HEVC

Session 511

Erik Turnquist, CoreMedia Engineer
Brad Ford, Camera Software

What is HEVC?

Why?



Up to 40%

Better compression than H.264

Up to 2x

Better compression

Today

Access

Playback

Capture

Export

Access

Playback

Capture

Export

HEVC Assets From Photos

PhotoKit will deliver HEVC assets for playback

```
// PHImageManager
manager.requestPlayerItem(forVideo: asset, options: nil) { (playerItem, dictionary) in
    // use AVPlayerItem
}

manager.requestLivePhoto(for: asset, targetSize: size, contentMode: .default, options: nil) {
    (livePhoto, dictionary) in
        // use PHLivePhoto
}
```

HEVC Assets From Photos

PhotoKit will deliver HEVC assets

```
// PHImageManager
manager.requestExportSession(forVideo: asset, options: nil, exportPreset: preset) {
    (session, dictionary) in
        // use AVAssetExportSession
    }
manager.requestAVAsset(forVideo: asset, options: nil) { (asset, audioMix, dictionary) in
    // use AVAsset
}
```

HEVC Assets From Photos

Access HEVC movie file data

```
// PHAssetResourceManager
resourceManager.requestData(for: assetResource, options: nil, dataReceivedHandler: { (data) in
    // use Data
}, { (error) in
    // handle Error
})
```

Access

Playback

Capture

Export

Native Playback Support



Supported in modern media frameworks

Streaming, play-while-download, and local files are supported

MPEG-4, QuickTime file format container support

No API opt-in required

Decode

macOS	
iOS	

HEVC Decode Support

Minimum configurations

	iOS	macOS
8-bit Hardware Decode	A9 chip	6th Generation Intel Core processor
10-bit Hardware Decode		7th Generation Intel Core processor
8-bit Software Decode	All iOS Devices	All Macs
10-bit Software Decode		


```
let player = AVPlayer(url: URL(fileURLWithPath: "MyAwesomeMovie.mov"))
player.play()
```

Decode Capability

Useful for non-realtime operations

Can be limited by hardware support

```
assetTrack.isDecodable
```

Playback Capability

Not all content can be played back in realtime

Differing capabilities on device

```
assetTrack.isPlayable
```

Hardware Decode Availability

Longest battery life

Best decode performance

```
let hardwareDecodeSupported = VTIsHardwareDecodeSupported(kCMVideoCodecType_HEVC)
```

Which Codec For Playback?

H.264

HEVC

Most Compatible



Smaller file size



Access

Playback

Capture

Export

Capture

Capture HEVC movies with AVFoundation

MPEG-4, QuickTime file format container support

HEVC Capture Support

Minimum configurations

iOS

8-Bit Hardware Encode

A10 Fusion chip

Capturing Movies with HEVC

Capturing Movies with HEVC



AVCaptureSession

Capturing Movies with HEVC

AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

Capturing Movies with HEVC

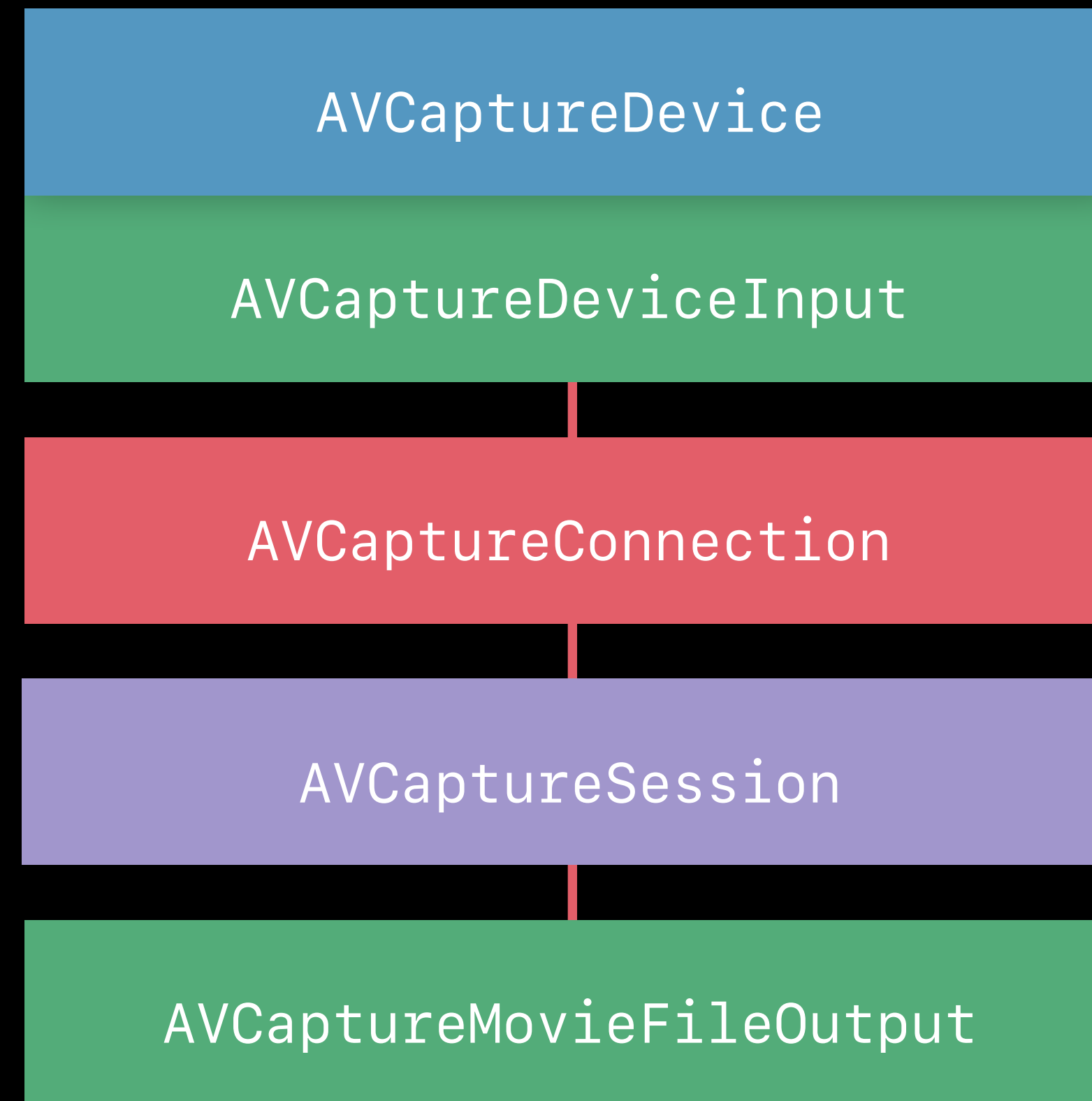
AVCaptureDevice

AVCaptureDeviceInput

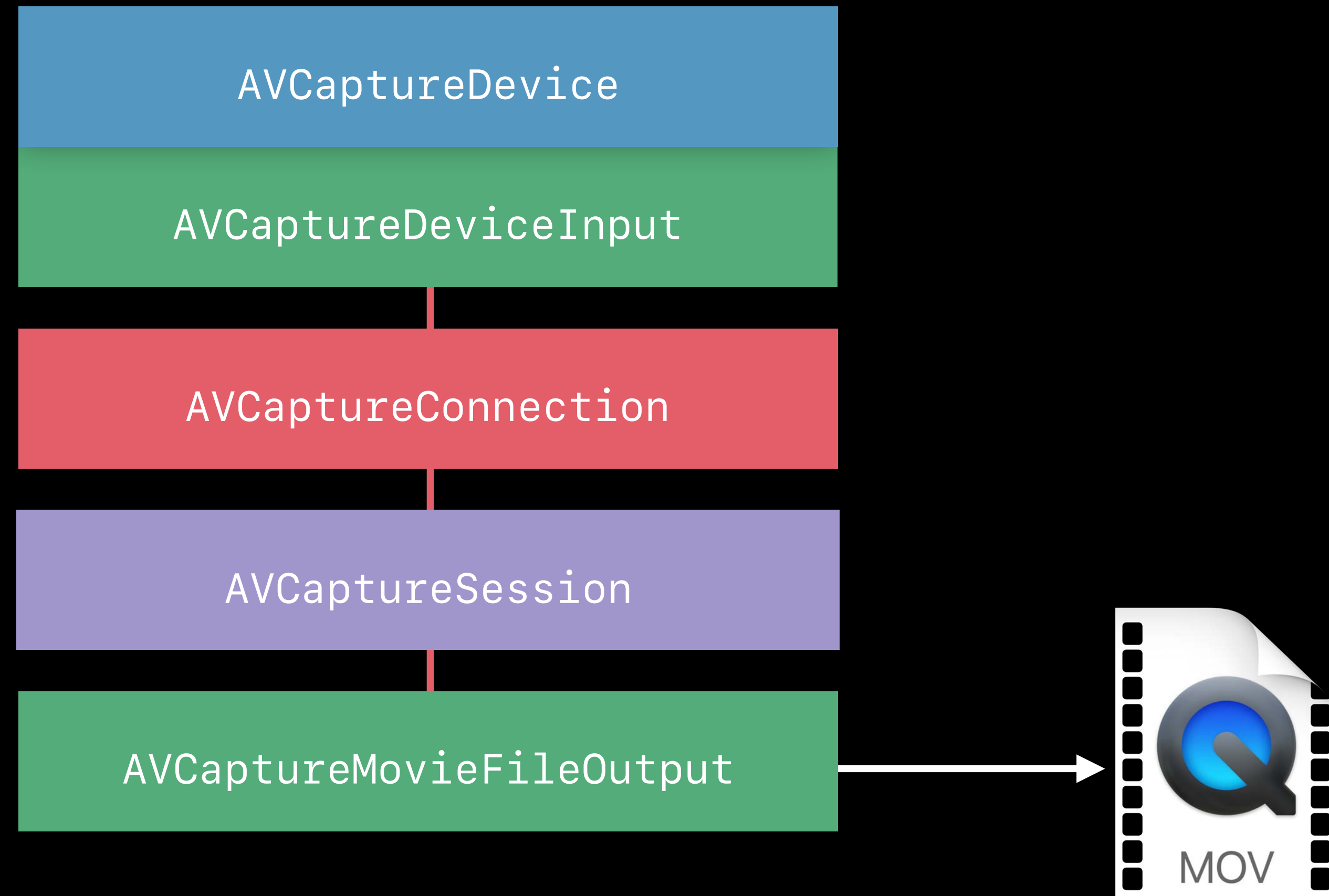
AVCaptureSession

AVCaptureMovieFileOutput

Capturing Movies with HEVC



Capturing Movies with HEVC



```
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```

```
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160
```

```
let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)
```

```
let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)
```

```
session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```



```
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160
```

```
let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)
```

```
let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)
```

```
session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```

```
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160
```

```
let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)
```

```
let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)
```

```
session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```

```
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```

```
let connection = movieFileOutput.connection(with: .video)
if movieFileOutput.availableVideoCodecTypes.contains(.hevc) {
    outputSetings = [AVVideoCodecKey: AVVideoCodecType.hevc]
} else {
    outputSetings = [AVVideoCodecKey: AVVideoCodecType.h264]
}

movieFileOutput.setOutputSettings(outputSetings, for: connection!)
```

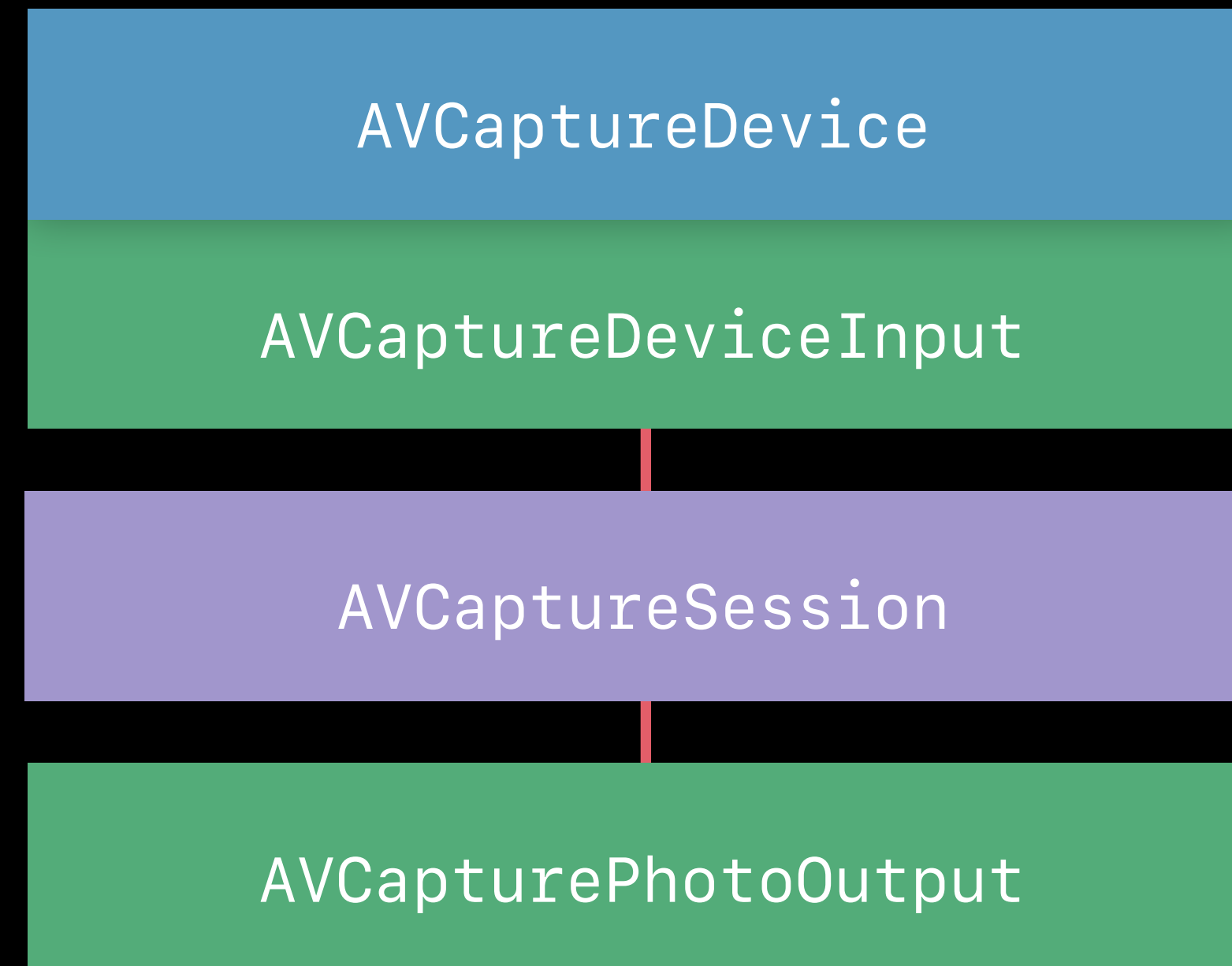
```
let connection = movieFileOutput.connection(with: .video)
if movieFileOutput.availableVideoCodecTypes.contains(.hevc) {
    outputSetings = [AVVideoCodecKey: AVVideoCodecType.hevc]
} else {
    outputSetings = [AVVideoCodecKey: AVVideoCodecType.h264]
}

movieFileOutput.setOutputSettings(outputSetings, for: connection!)
```

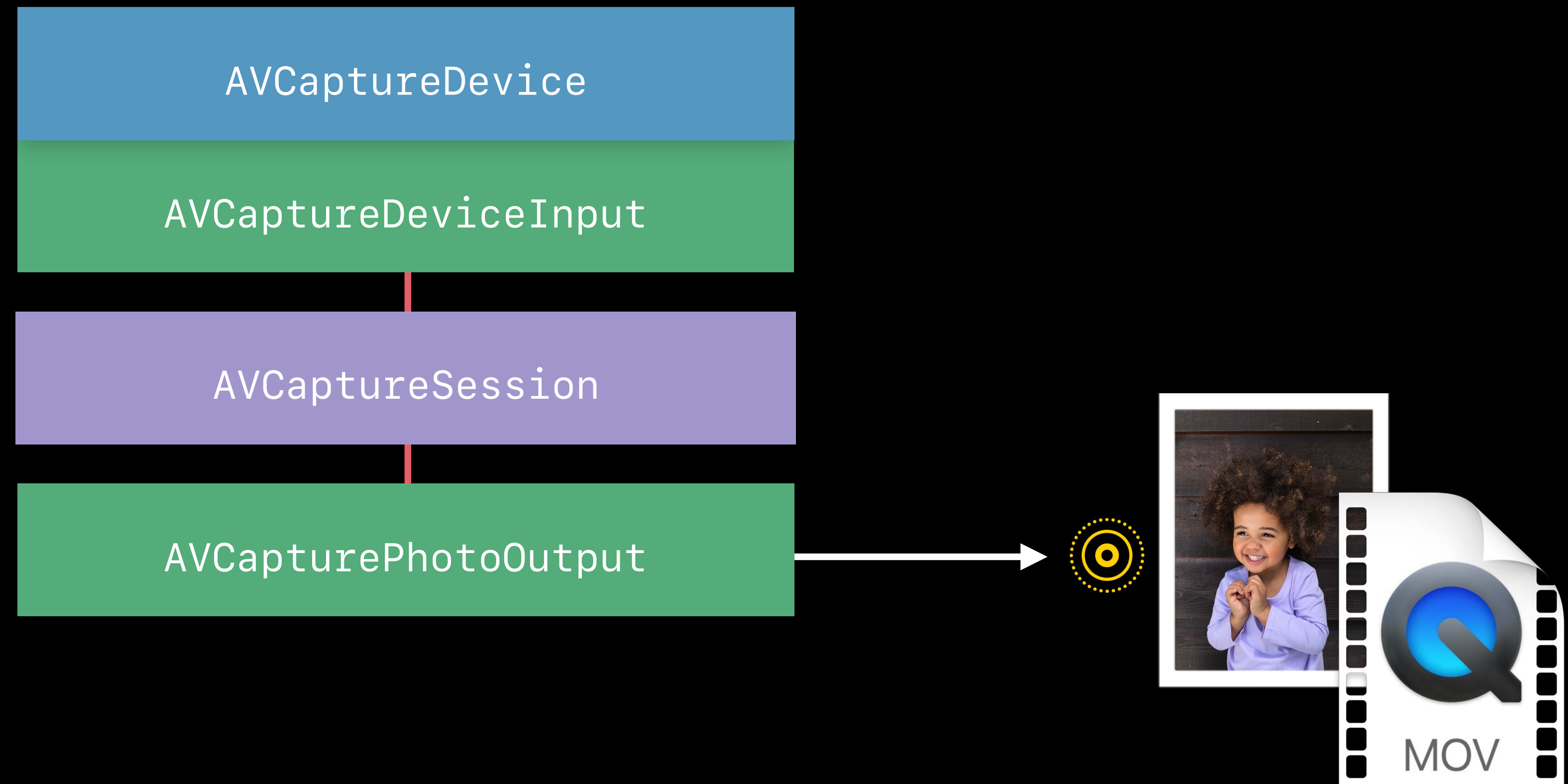
```
let connection = movieFileOutput.connection(with: .video)
if movieFileOutput.availableVideoCodecTypes.contains(.hevc) {
    outputSetings = [AVVideoCodecKey: AVVideoCodecType.hevc]
} else {
    outputSetings = [AVVideoCodecKey: AVVideoCodecType.h264]
}

movieFileOutput.setOutputSettings(outputSetings, for: connection!)
```

Capturing Live Photo Movies with HEVC



Capturing Live Photo Movies with HEVC



Live Photo Enhancements

Video stabilization

Music playback

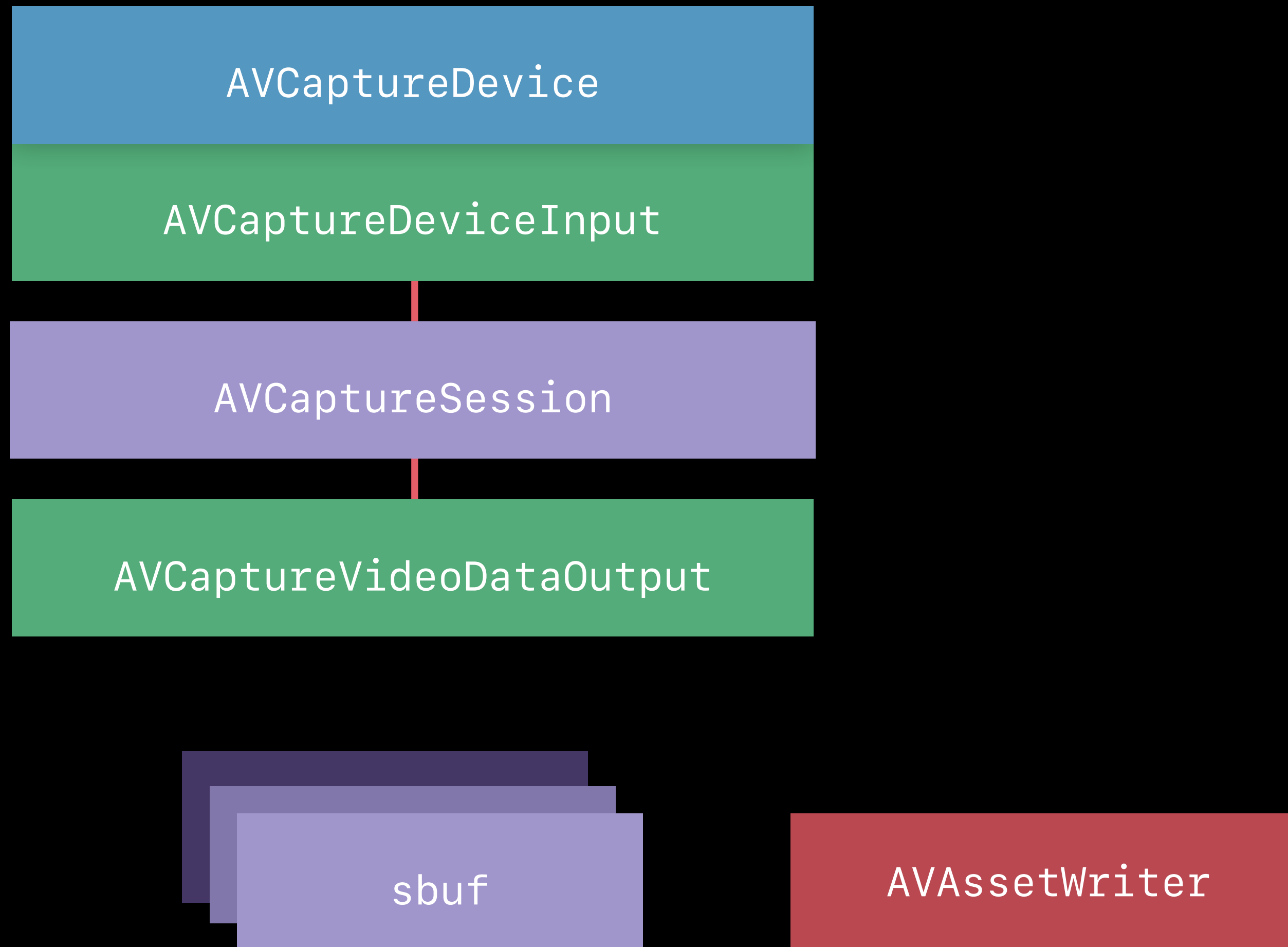
30 FPS

```
let photoSettings = AVCapturePhotoSettings()
photoSettings.livePhotoMovieFileURL = URL(fileURLWithPath: myFilePath)
if photoOutput.availableLivePhotoVideoCodecTypes.contains(.hevc) {
    photoSettings.livePhotoVideoCodecType = .hevc
}
photoOutput.capturePhoto(with: photoSettings, delegate: self)
```

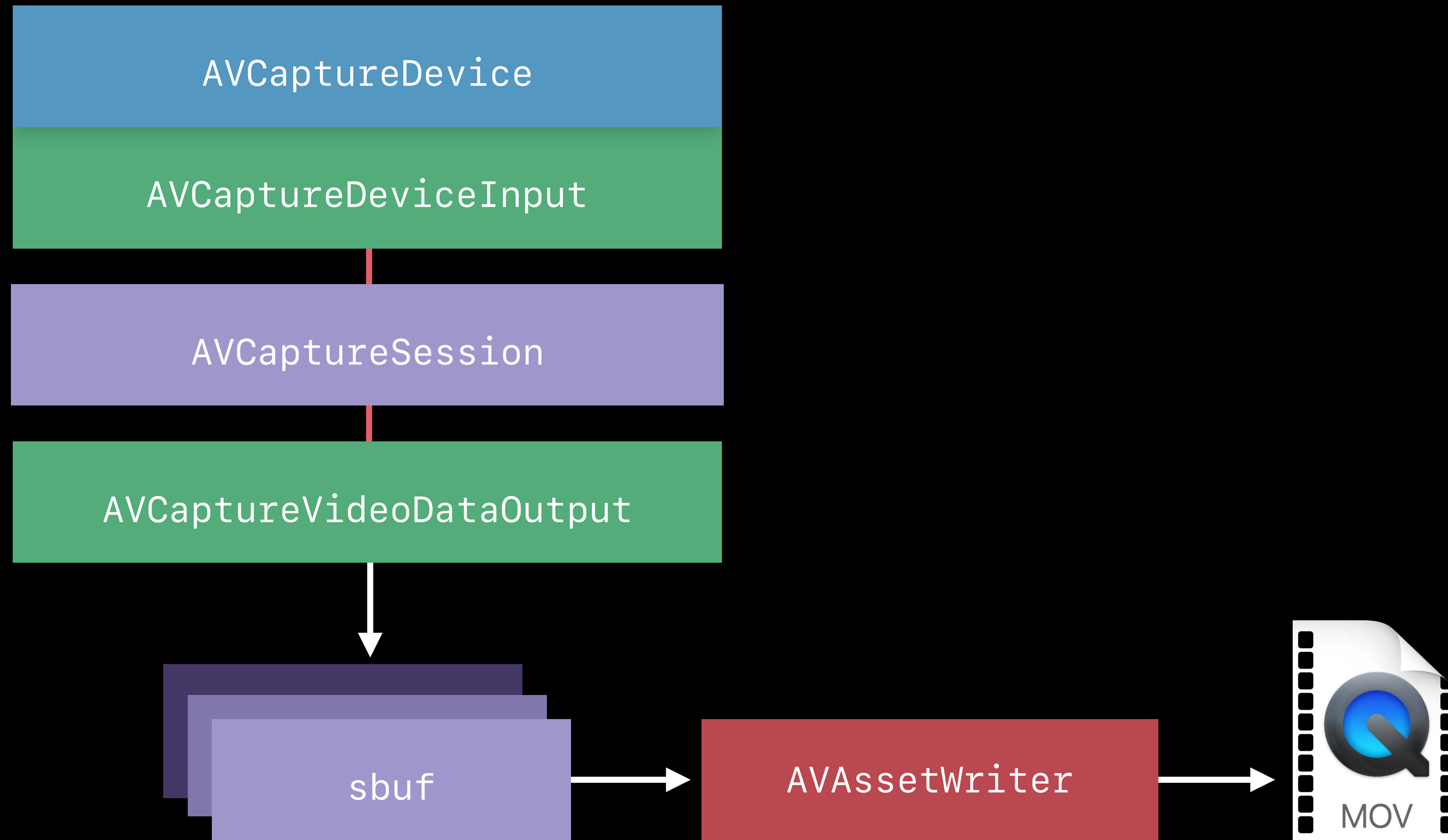
```
let photoSettings = AVCapturePhotoSettings()
photoSettings.livePhotoMovieFileURL = URL(fileURLWithPath: myFilePath)
if photoOutput.availableLivePhotoVideoCodecTypes.contains(.hevc) {
    photoSettings.livePhotoVideoCodecType = .hevc
}
photoOutput.capturePhoto(with: photoSettings, delegate: self)
```

```
let photoSettings = AVCapturePhotoSettings()
photoSettings.livePhotoMovieFileURL = URL(fileURLWithPath: myFilePath)
if photoOutput.availableLivePhotoVideoCodecTypes.contains(.hevc) {
    photoSettings.livePhotoVideoCodecType = .hevc
}
photoOutput.capturePhoto(with: photoSettings, delegate: self)
```

Capturing HEVC Movies with AVAssetWriter



Capturing HEVC Movies with AVAssetWriter



Capturing HEVC Movies with AVAssetWriter

Configure *AVAssetWriterInput* with output settings

Video data output can recommend settings

```
// iOS 7  
vdo.recommendedVideoSettingsForAssetWriter(writingTo: .mov)  
  
// iOS 11  
vdo.recommendedVideoSettings(forVideoCodecType: .hevc, assetWriterOutputFileType: .mov)
```

Capturing HEVC Movies with AVAssetWriter

Configure AVAssetWriterInput with output settings

Video data output can recommend settings

```
// iOS 7
vdo.recommendedVideoSettingsForAssetWriter(writingTo: .mov)

// iOS 11
vdo.recommendedVideoSettings(forVideoCodecType: .hevc, assetWriterOutputFileType: .mov)
```


Access

Playback

Capture

Export



Export

Transcode to HEVC with AVFoundation and VideoToolbox

MPEG-4, QuickTime file format container support

API opt-in required

Encode

macOS	
iOS	

HEVC Encode Support

Minimum configurations

iOS

macOS

8-bit Hardware Encode	A10 Fusion chip	6th Generation Intel Core processor
10-bit Software Encode		All Macs

Transcode with AVAssetExportSession

Transcode with AVAssetExportSession



Export Session

No change in behavior for existing presets

Convert from H.264 to HEVC with new presets

Produce smaller AVAssets with same quality

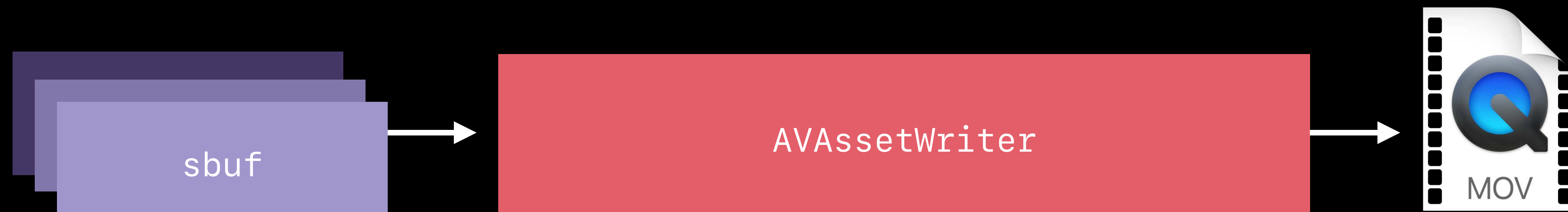
```
AVAssetExportPresetHEVC1920x1080
```

```
AVAssetExportPresetHEVC3840x2160
```

```
AVAssetExportPresetHEVCHighestQuality
```

Compression with AVAssetWriter

Compression with AVAssetWriter



Asset Writer

Specify HEVC with output settings for `AVAssetWriterInput`

```
settings = [AVVideoCodecKey: AVVideoCodecType.hevc]
```

Convenient output settings with `AVOutputSettingsAssistant`

```
AVOutputSettingsPreset.hevc1920x1080
```

```
AVOutputSettingsPreset.hevc3840x2160
```

Valid Output Settings

Query encoder for supported properties in output settings

```
let error = VTCopySupportedPropertyDictionaryForEncoder(  
    3840, 2160,  
    kCMVideoCodecType_HEVC,  
    encoderSpecification,  
    &encoderID, &properties)  
  
if error == kVTCouldNotFindVideoEncoderErr {  
    // no HEVC encoder  
}
```

Encoder ID is a unique identifier for an encoder

Properties and encoder ID can be used in output settings

Valid Output Settings

Query encoder for supported properties in output settings

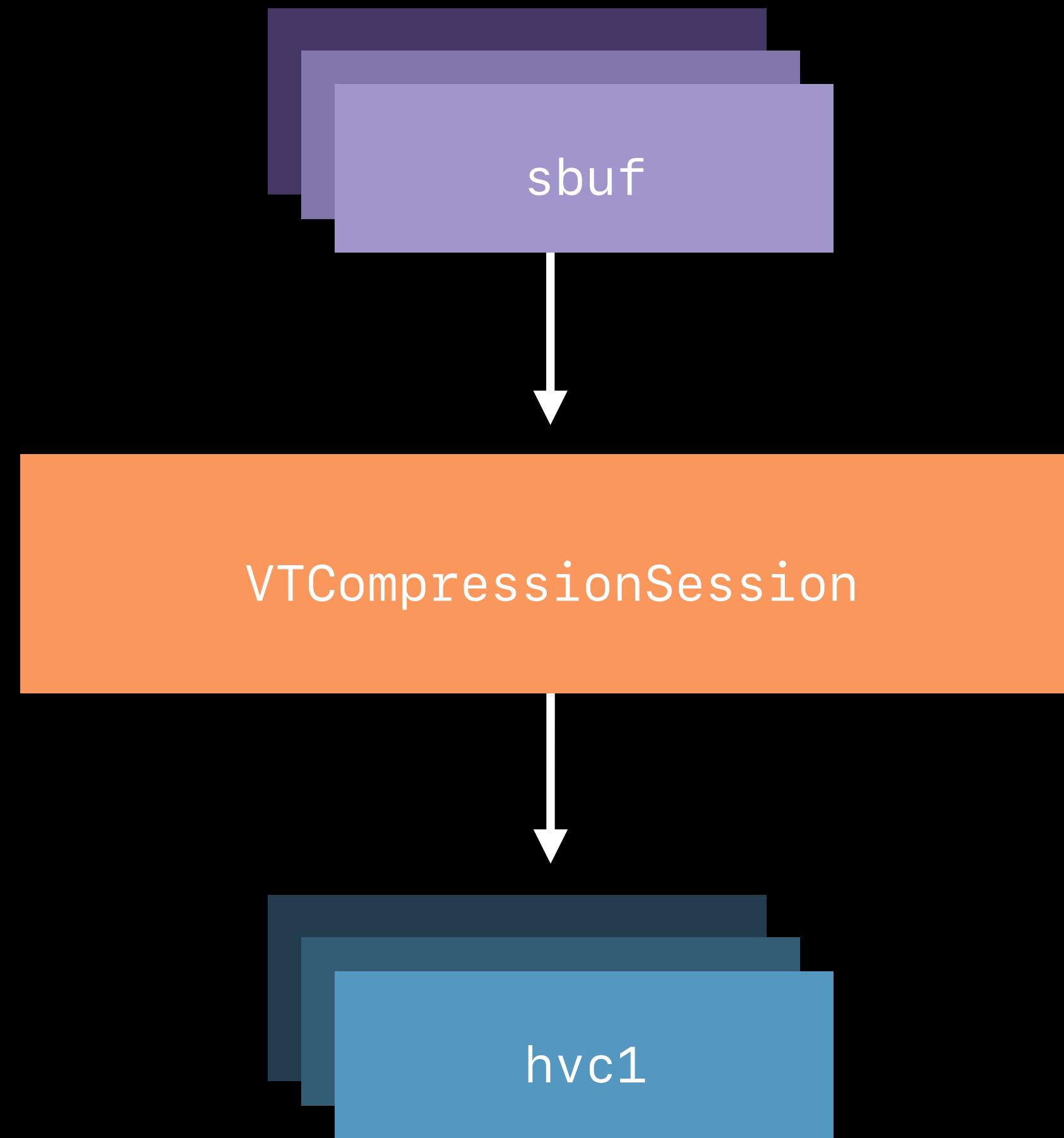
```
let error = VTCopySupportedPropertyDictionaryForEncoder(  
    3840, 2160,  
    kCMVideoCodecType_HEVC,  
    encoderSpecification,  
    &encoderID, &properties)  
  
if error == kVTCouldNotFindVideoEncoderErr {  
    // no HEVC encoder  
}
```

Encoder ID is a unique identifier for an encoder

Properties and encoder ID can be used in output settings

Compress Samples with VTCompressionSession

Compress Samples with VTCompressionSession



Compression Session

Create session with HEVC encoder

```
let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_H264,
    encoderSpecification,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session);

if error == kVTCouldNotFindVideoEncoderErr {
    // no H.264 encoder
}
```

Compression Session

Create session with HEVC encoder

```
let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session);

if error == kVTCouldNotFindVideoEncoderErr {
    // no HEVC encoder
}
```



```
// Use hardware when available on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

// Using hardware, or software
```

```
// Use hardware when available on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

// Using hardware, or software
```

```
// Use hardware when available on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

// Using hardware, or software
```

```
// Realtime encode, fail if no hardware exists on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_RequireHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

if error == kVTCouldNotFindVideoEncoderErr {
    // no hardware HEVC encoder
}
```

```
// Realtime encode, fail if no hardware exists on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_RequireHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

if error == kVTCouldNotFindVideoEncoderErr {
    // no hardware HEVC encoder
}
```

```
// Realtime encode, fail if no hardware exists on macOS

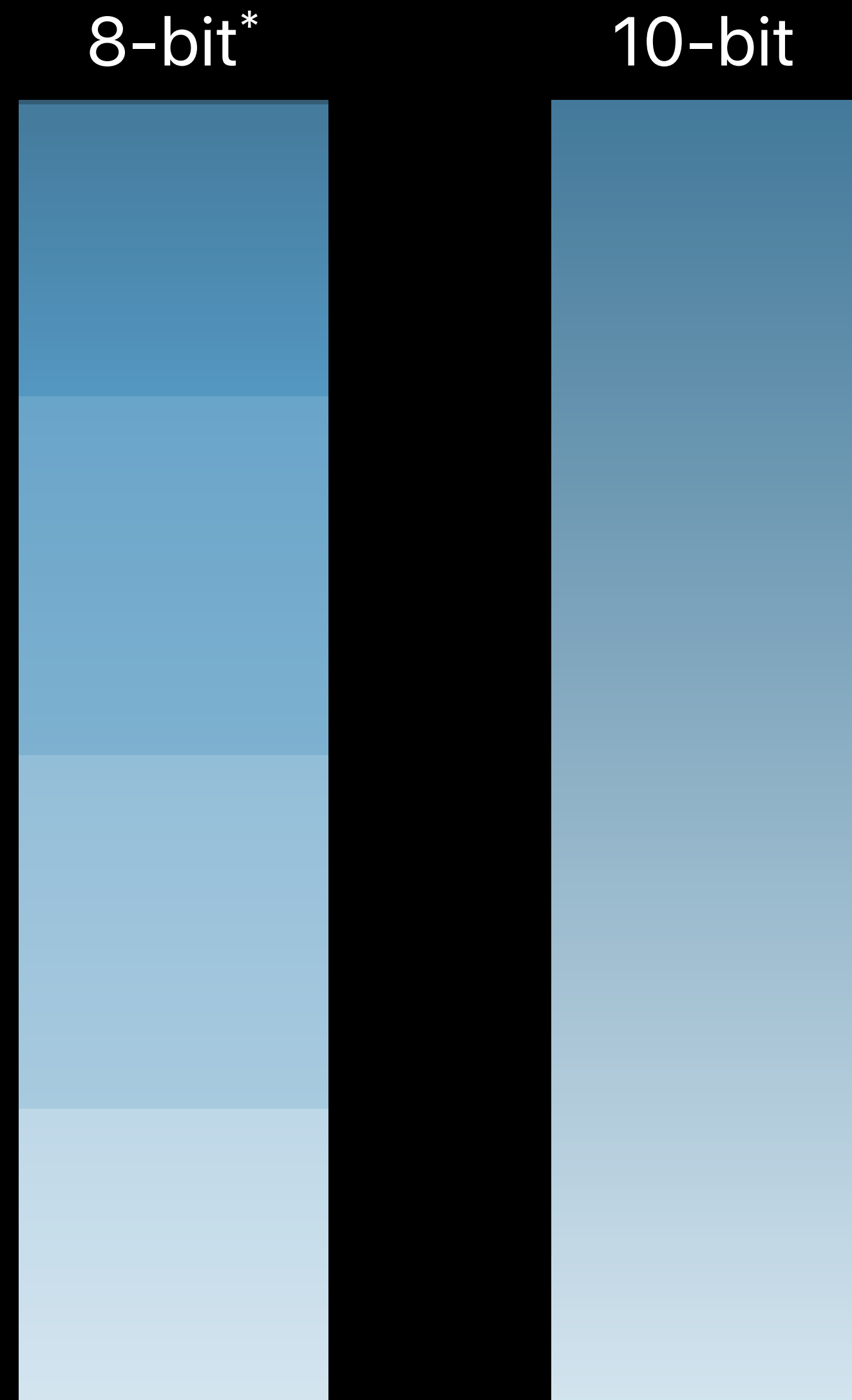
let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_RequireHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

if error == kVTCouldNotFindVideoEncoderErr {
    // no hardware HEVC encoder
}
```

Bit Depth

Bit depth



*effect amplified for illustration

HEVC 10-bit Encode Settings

Set profile via `kVTCompressionPropertyKey_ProfileLevel`

```
// Check VTSessionCopySupportedPropertyDictionary() for support  
kVTProfileLevel_HEVC_Main10_AutoLevel
```

CoreVideo pixel buffer format

```
kCVPixelFormatType_420YpCbCr10BiPlanarVideoRange // 10-bit 4:2:0
```

HEVC 10-bit Encode Settings

Set profile via `kVTCompressionPropertyKey_ProfileLevel`

```
// Check VTSessionCopySupportedPropertyDictionary() for support  
kVTProfileLevel_HEVC_Main10_AutoLevel
```

CoreVideo pixel buffer format

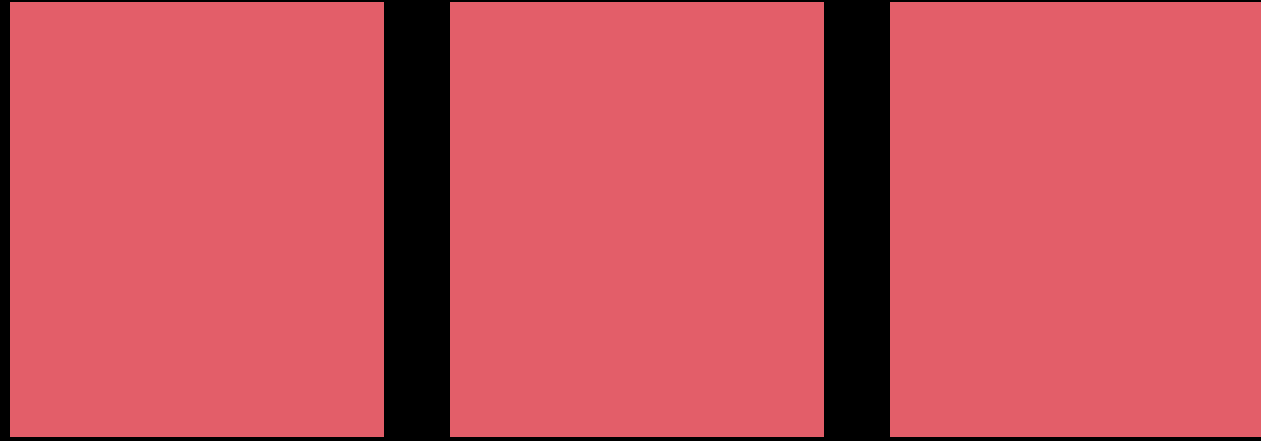
```
kCVPixelFormatType_420YpCbCr10BiPlanarVideoRange // 10-bit 4:2:0
```

Hierarchical Frame Encoding

Video Encoding 101

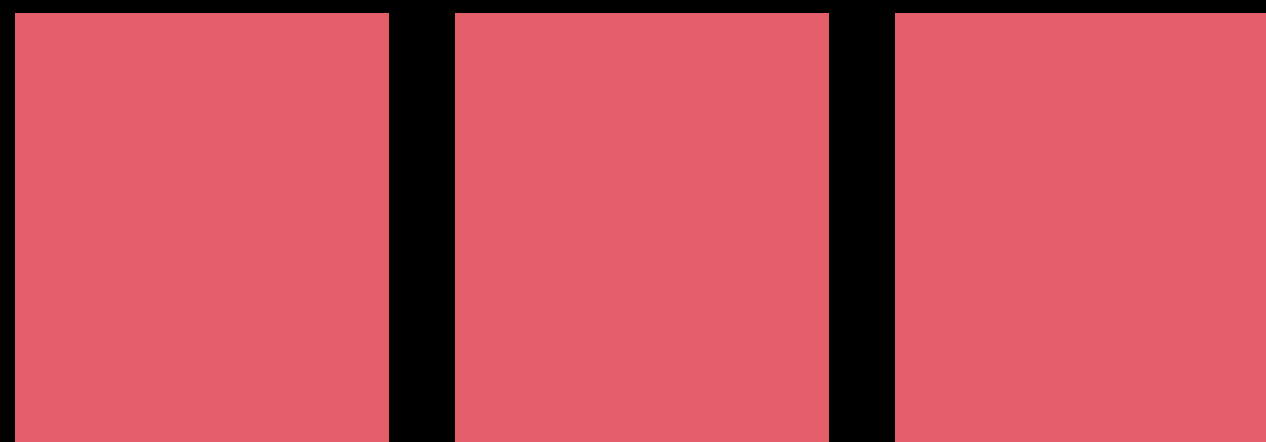
Video Encoding 101

I Frame

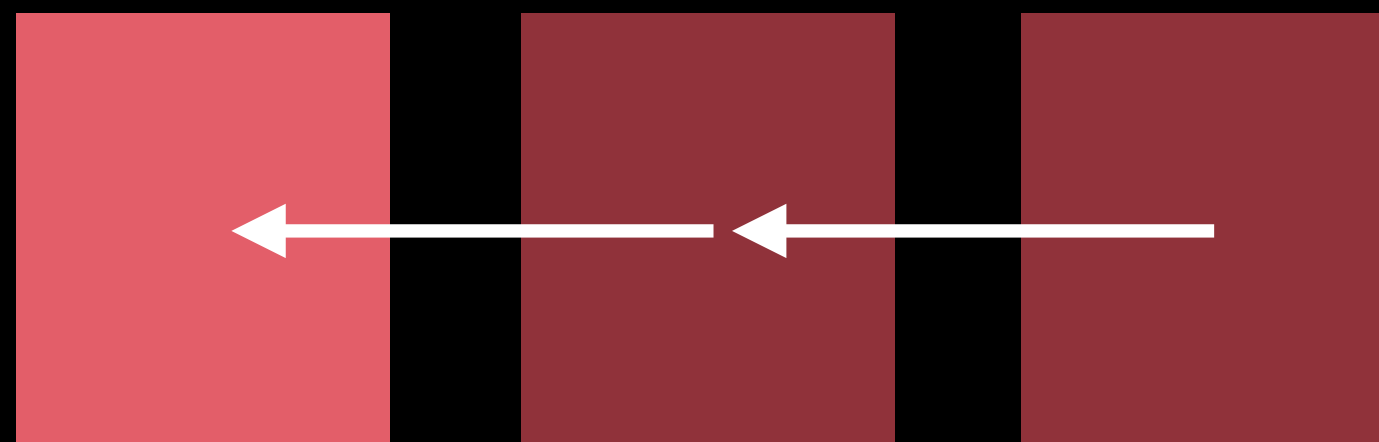


Video Encoding 101

I Frame

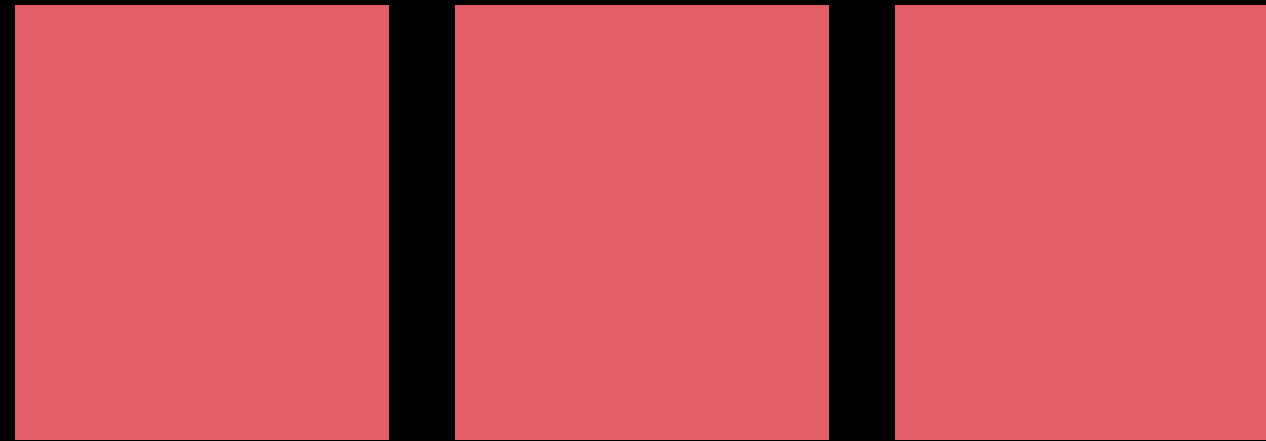


P Frame

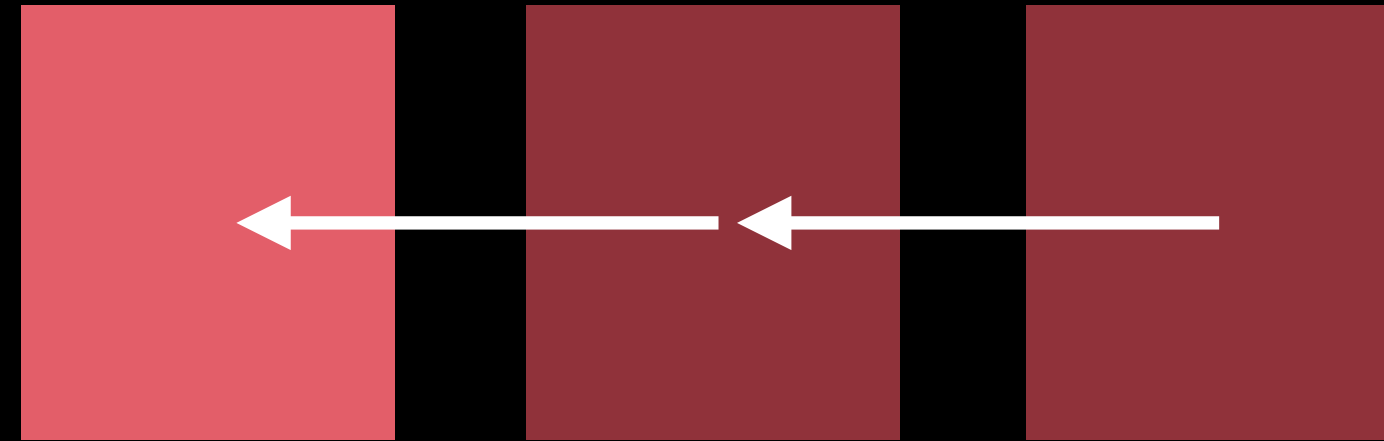


Video Encoding 101

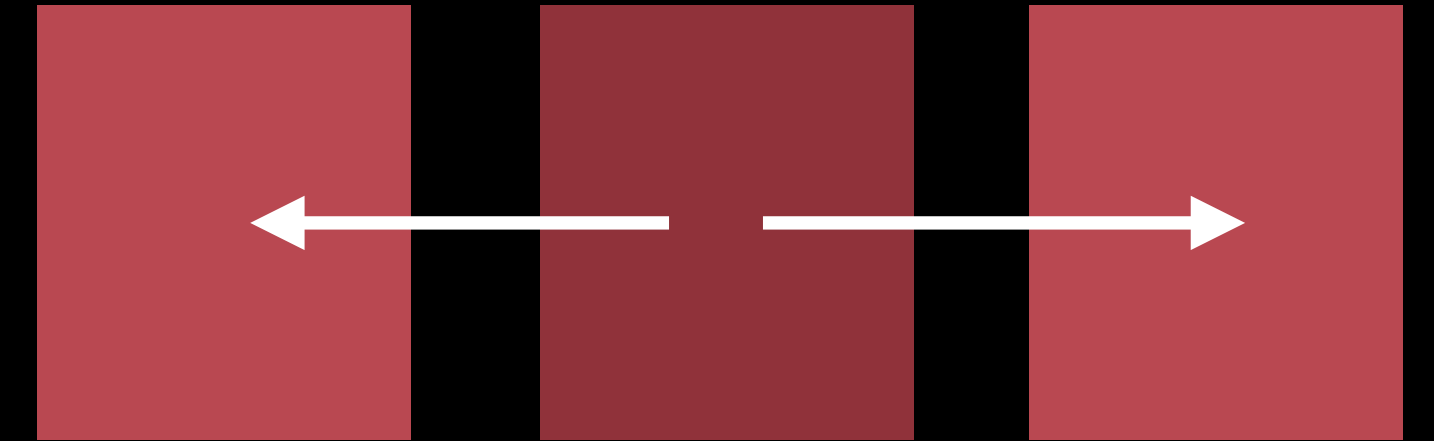
I Frame



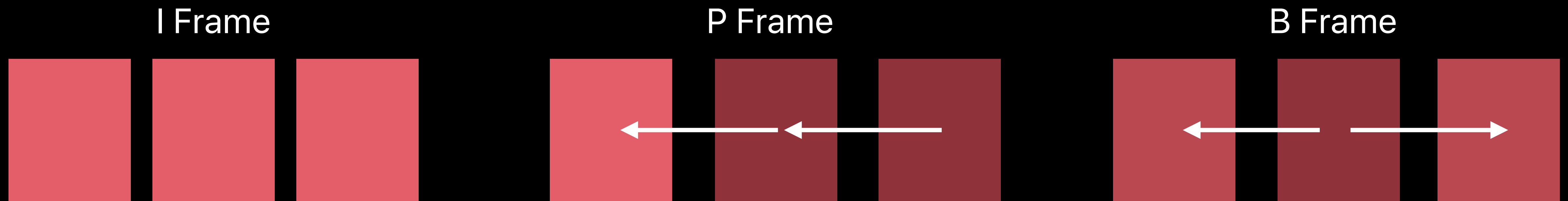
P Frame



B Frame

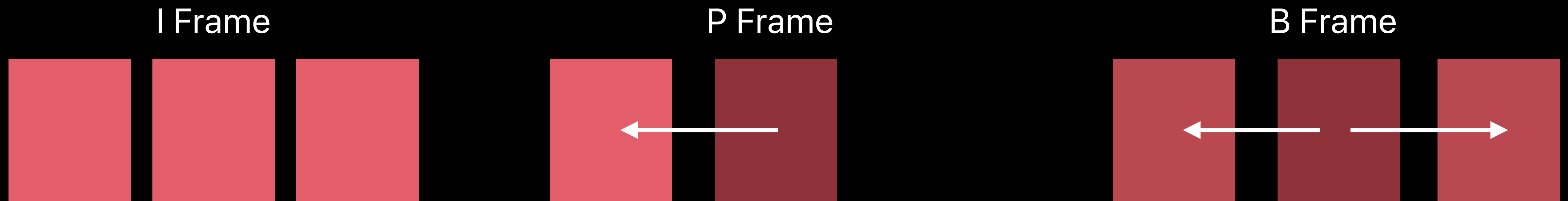


Video Encoding 101



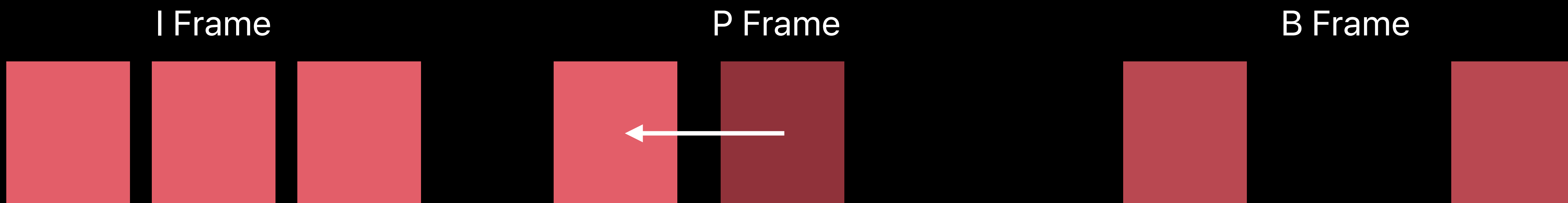
Can drop a frame when another doesn't depend on it

Video Encoding 101



Can drop a frame when another doesn't depend on it

Video Encoding 101



Can drop a frame when another doesn't depend on it

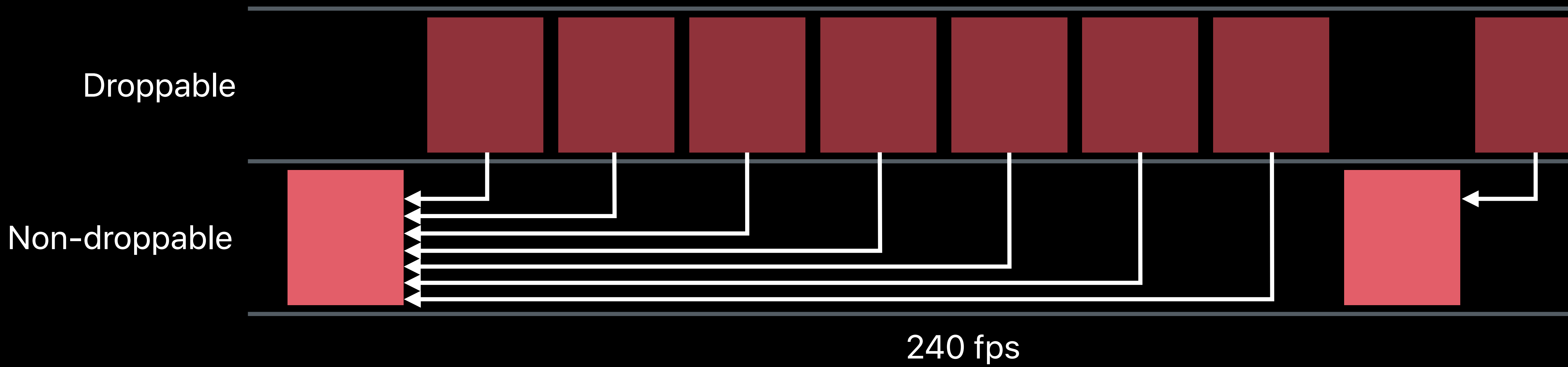
Compatible High Frame Rate Content

Droppable

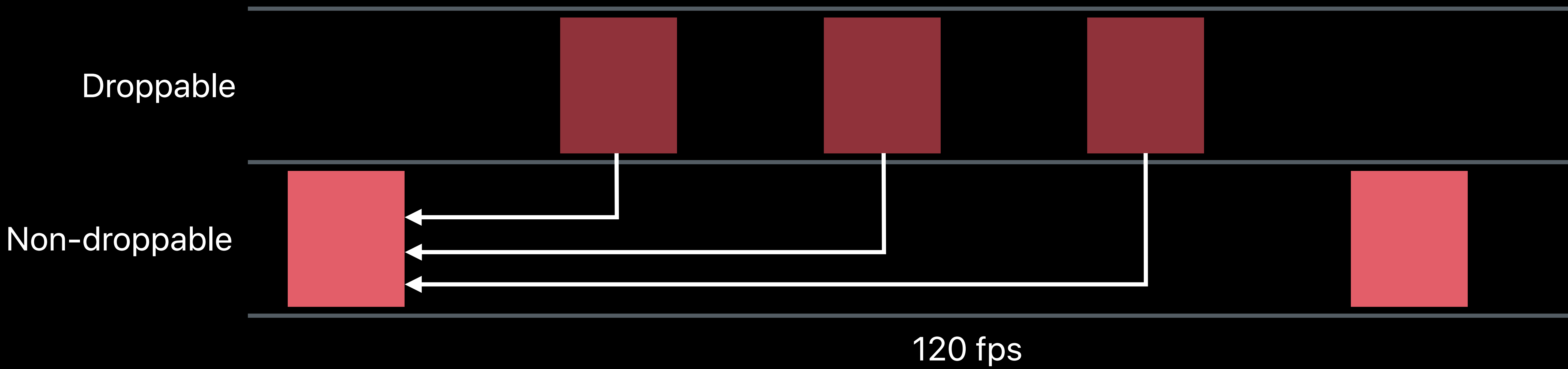


Non-droppable

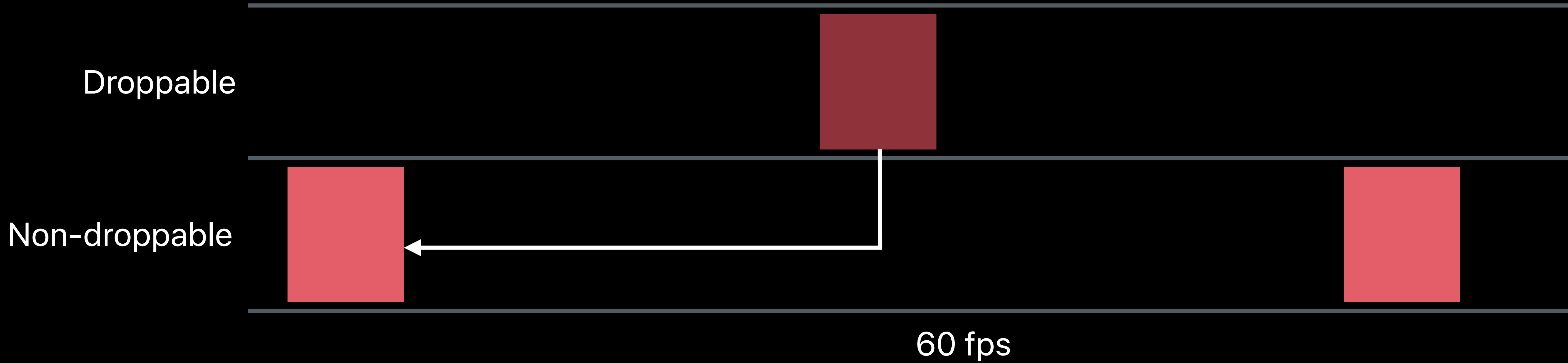
Compatible High Frame Rate Content



Compatible High Frame Rate Content



Compatible High Frame Rate Content

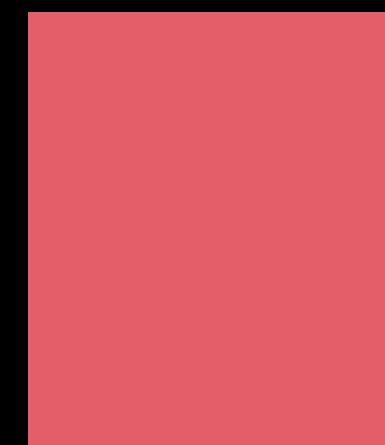


Compatible High Frame Rate Content

Droppable

Non-droppable

30 fps



Temporal Levels

Level 3

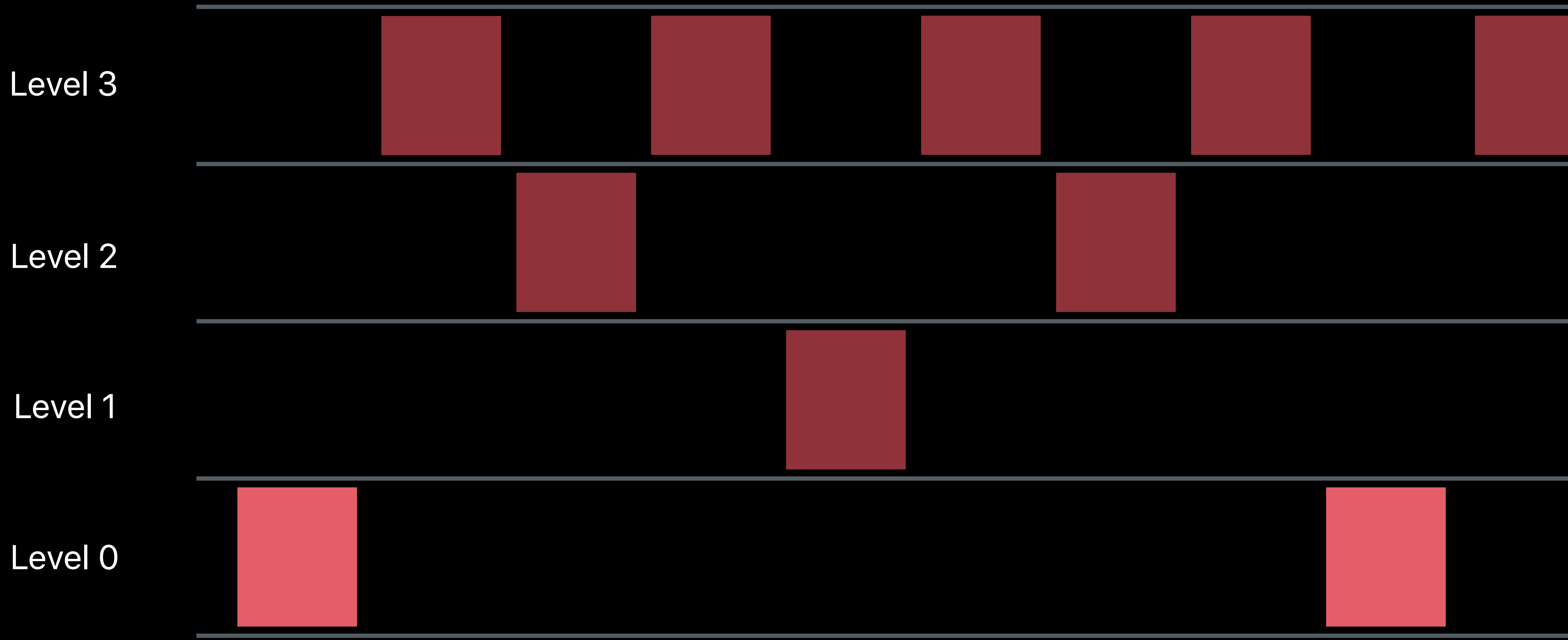
Level 2

Level 1

Level 0

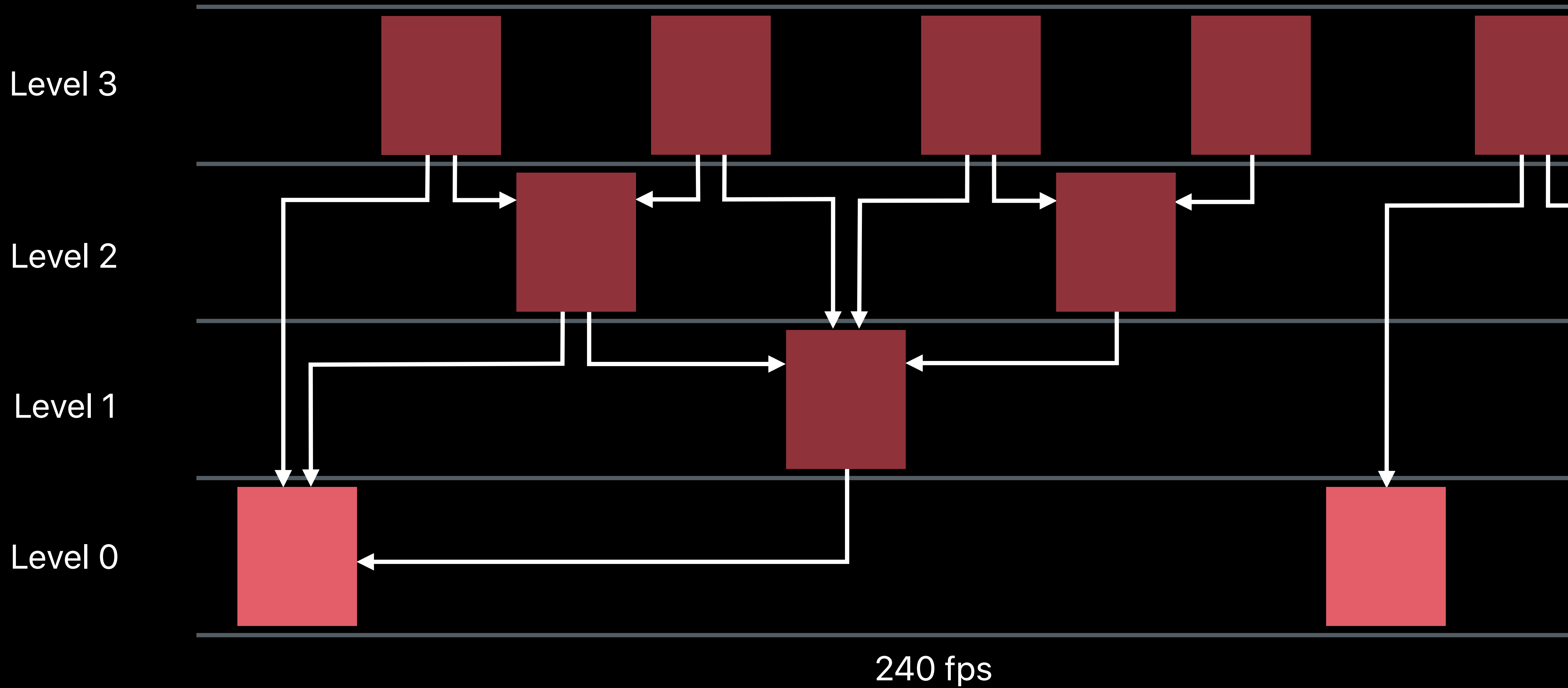


Temporal Levels



240 fps

Temporal Levels



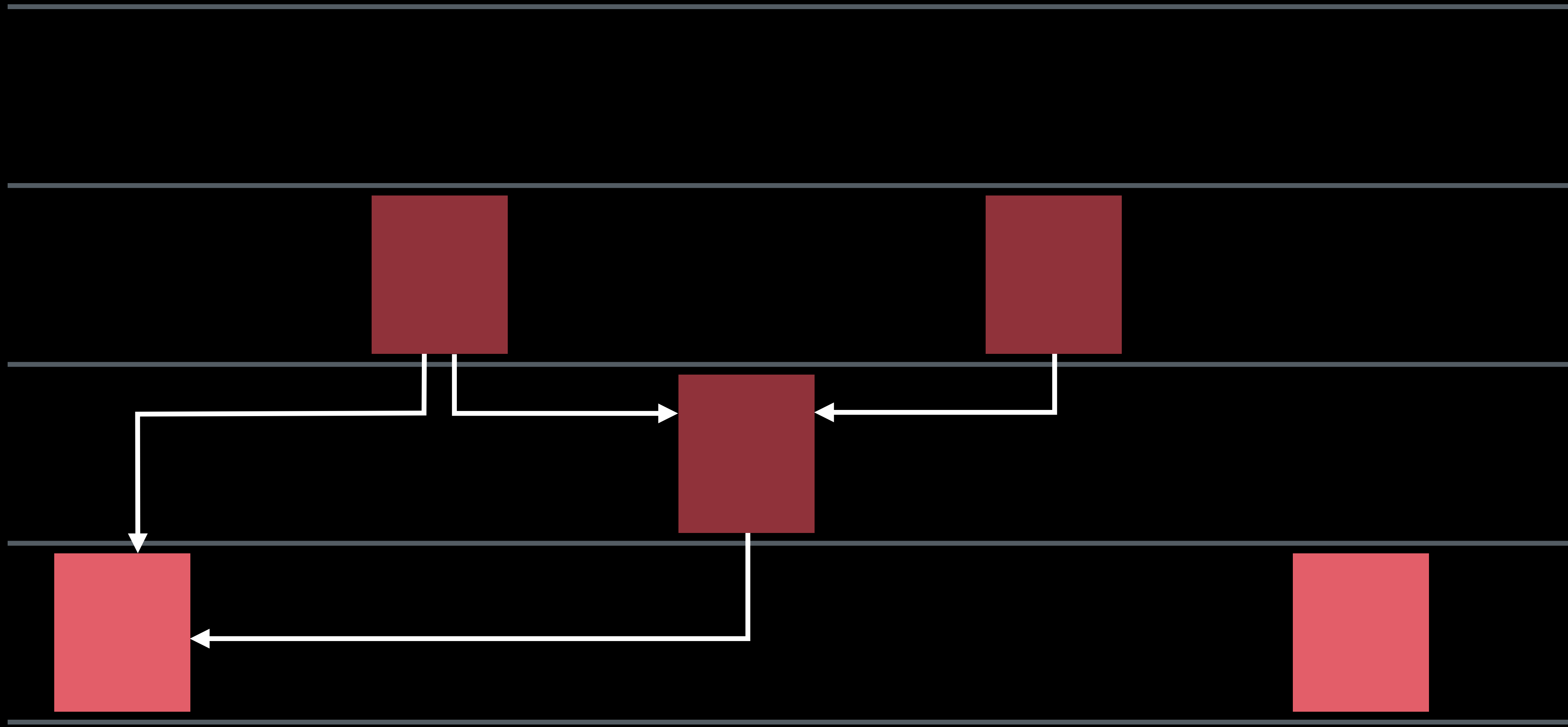
Temporal Levels

Level 3

Level 2

Level 1

Level 0



120 fps

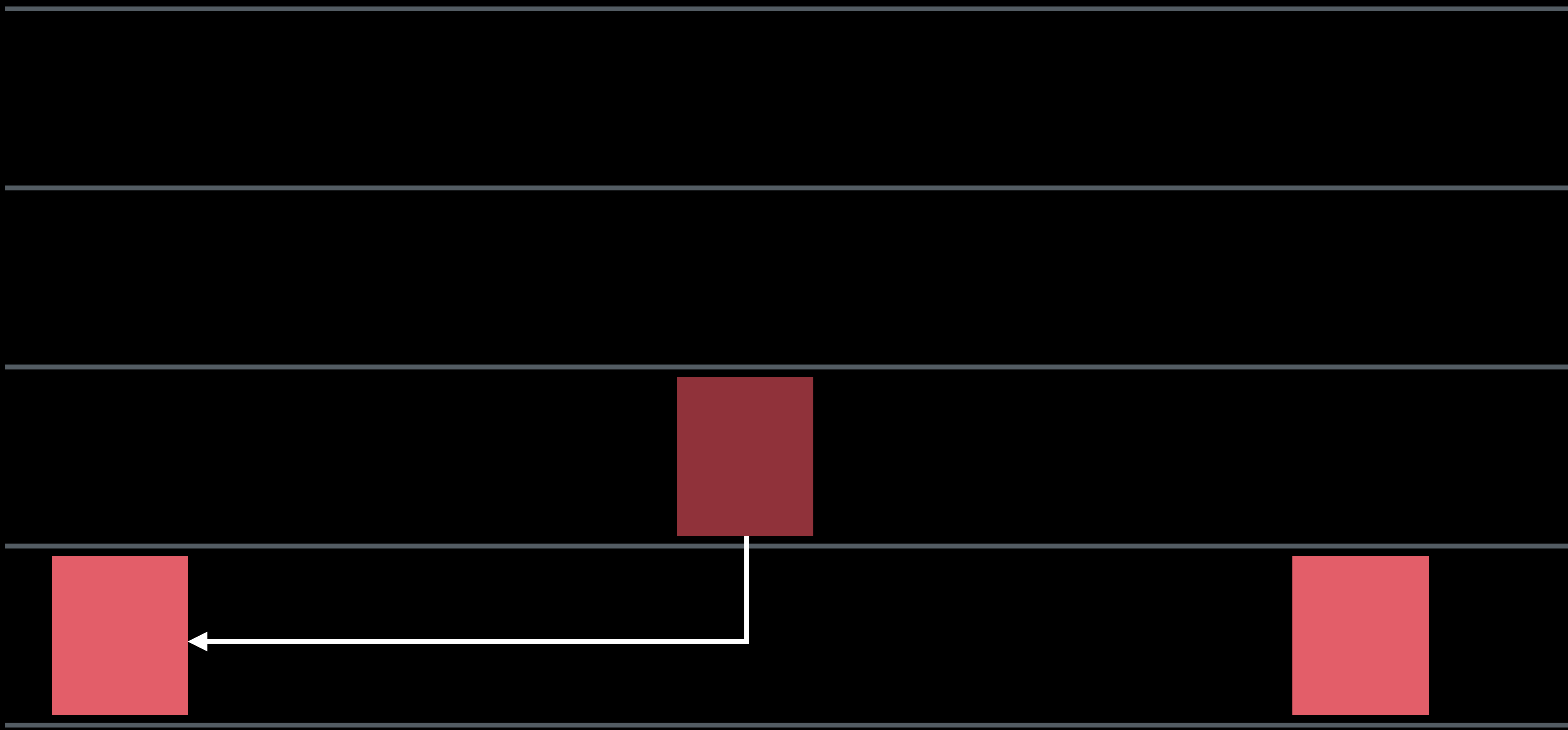
Temporal Levels

Level 3

Level 2

Level 1

Level 0



60 fps

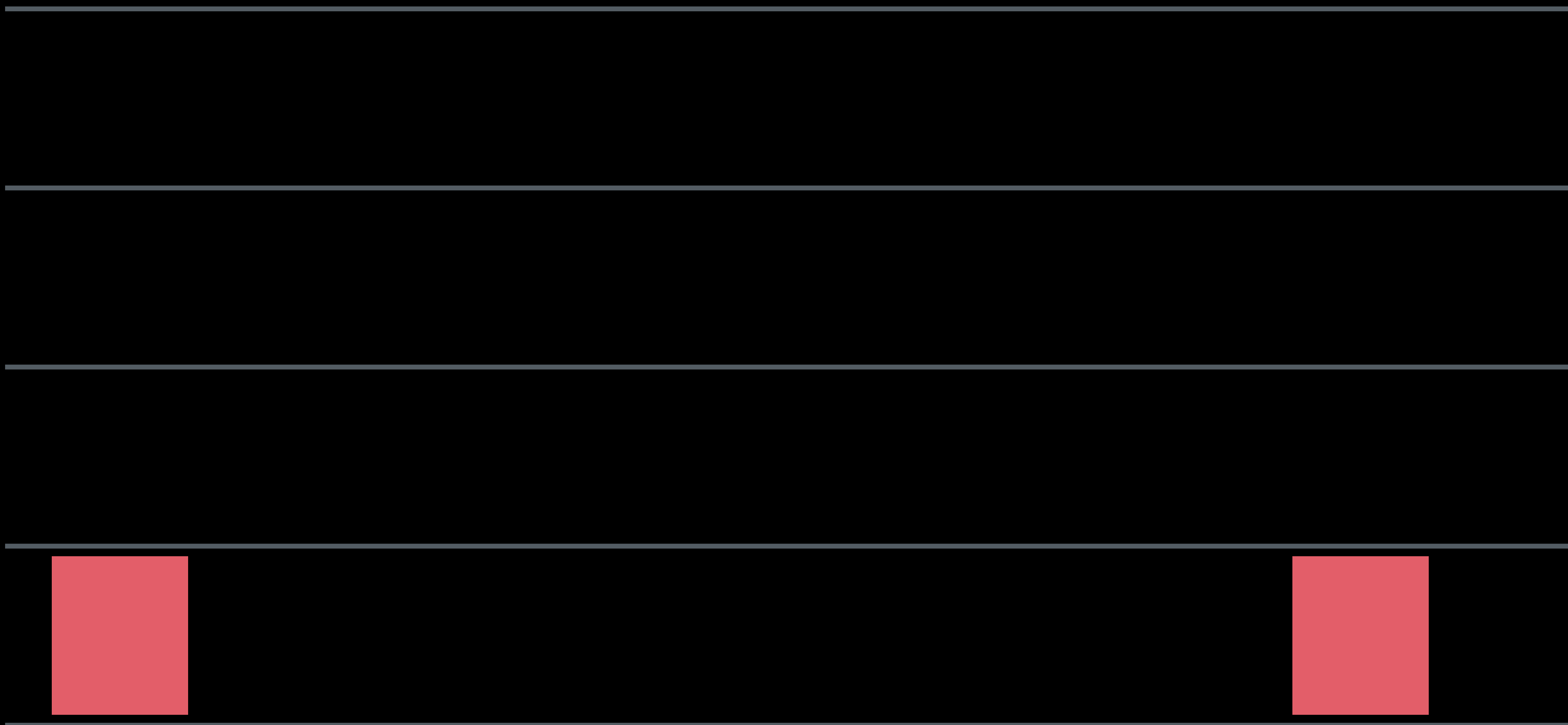
Temporal Levels

Level 3

Level 2

Level 1

Level 0



30 fps

HEVC Hierarchical Encoding

Improved temporal scalability

Improved motion compensation

File annotations (MPEG-4 Part 15 - 8.4)

HEVC Hierarchical Encoding

Create compatible high frame rate content

Set base layer and capture frame rate

```
// Check VTSessionCopySupportedPropertyDictionary() for support  
kVTCompressionPropertyKey_BaseLayerFrameRate // temporal level 0 frame rate  
kVTCompressionPropertyKey_ExpectedFrameRate // frame rate of content
```

Base layer must be decoded

Decode or drop other levels

HEIF — it's what's for dinner

Brad Ford, Camera Software

What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files

What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files

What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files

What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files

What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files

What is HEIF?

H

E

I

F

High

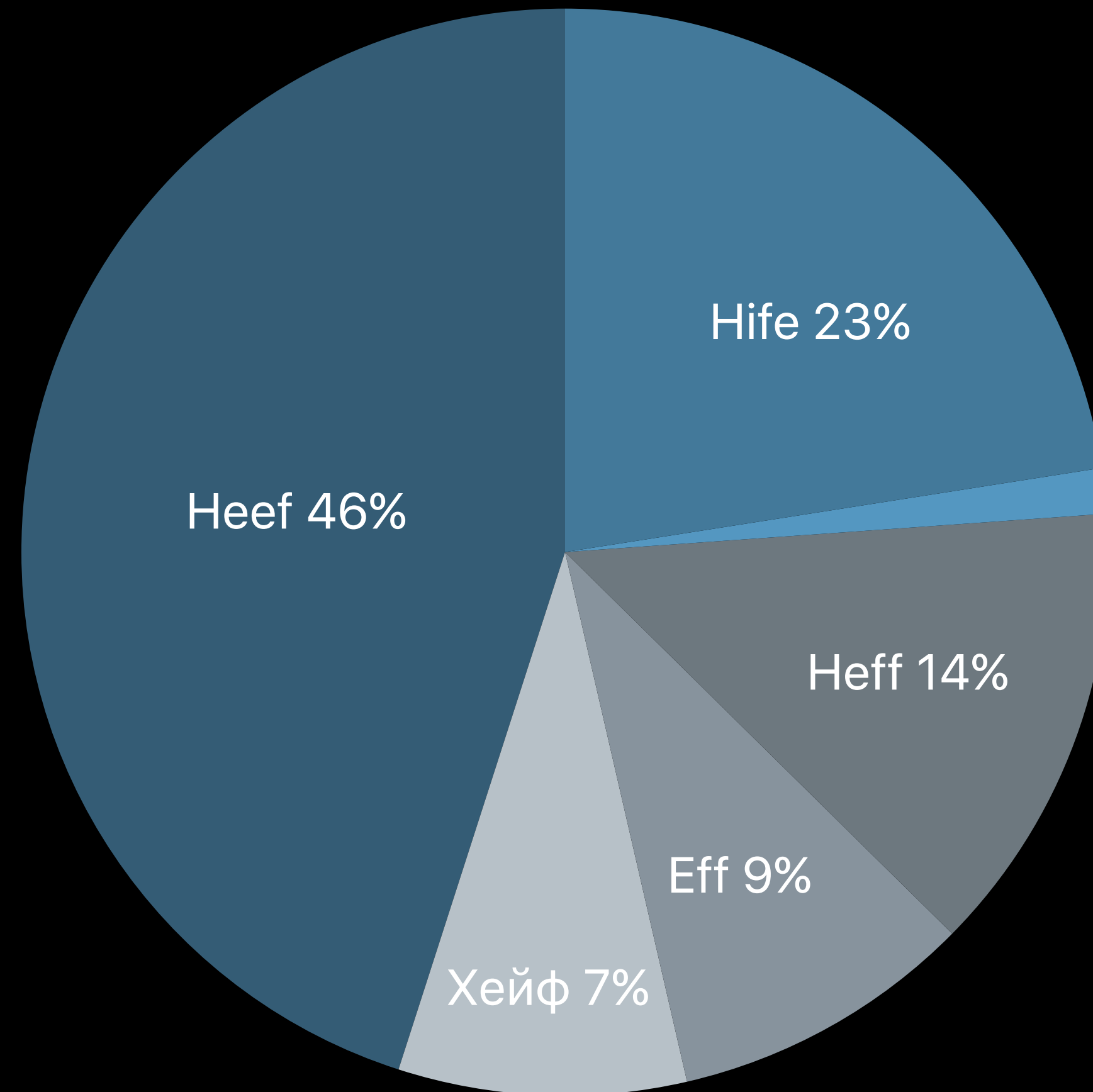
Efficiency

Image

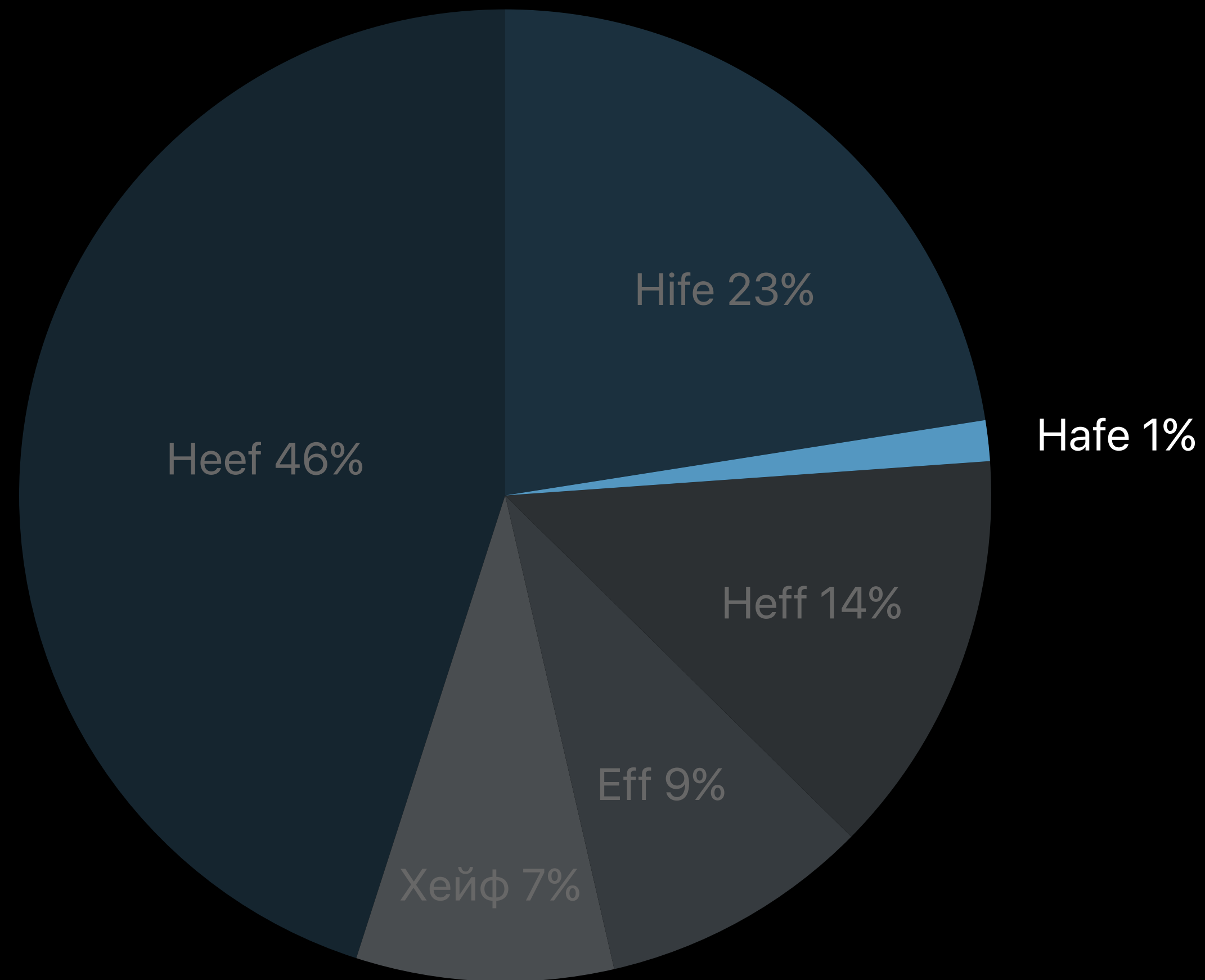
File

(Format)

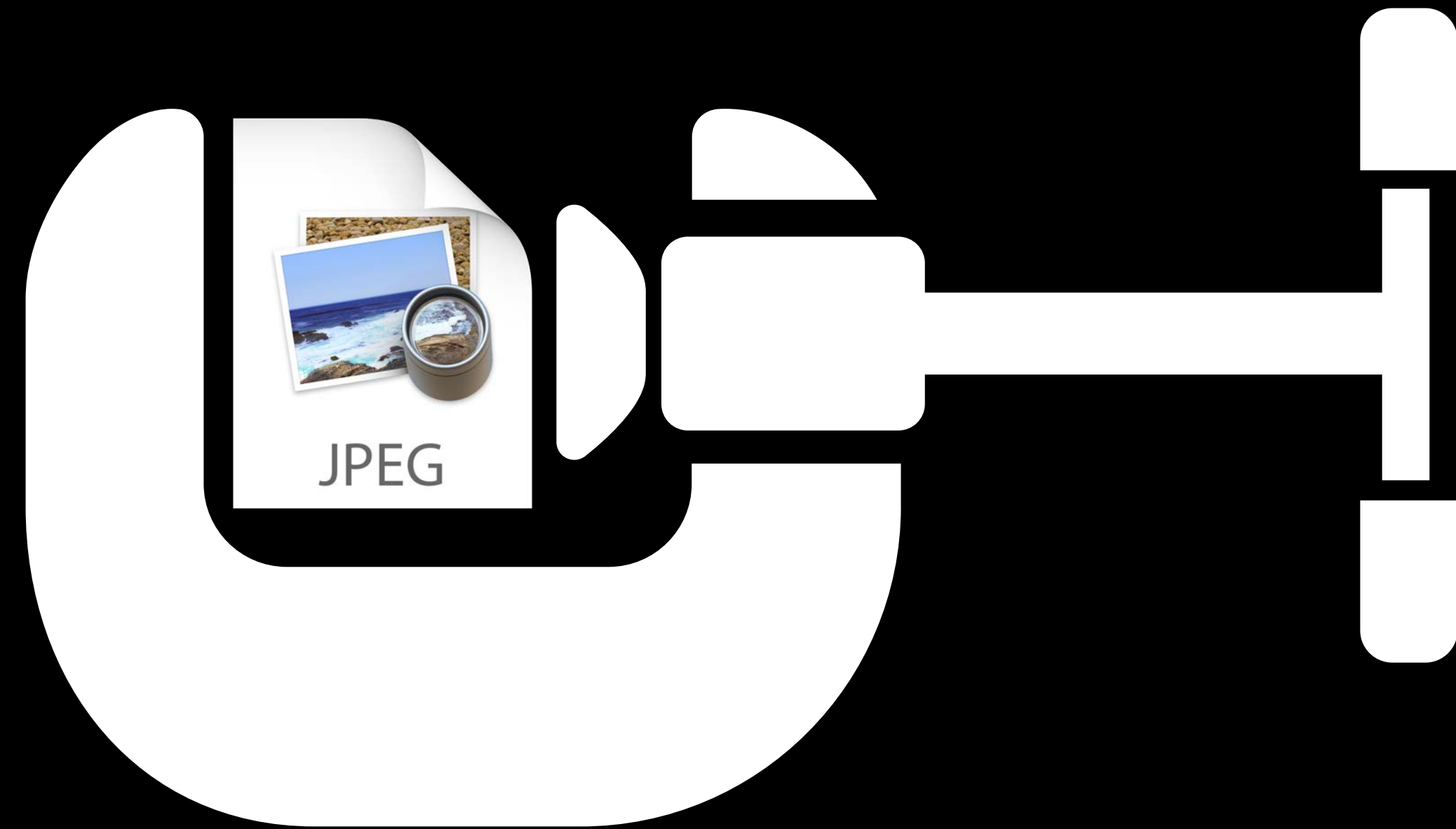
HEIF Pronunciation



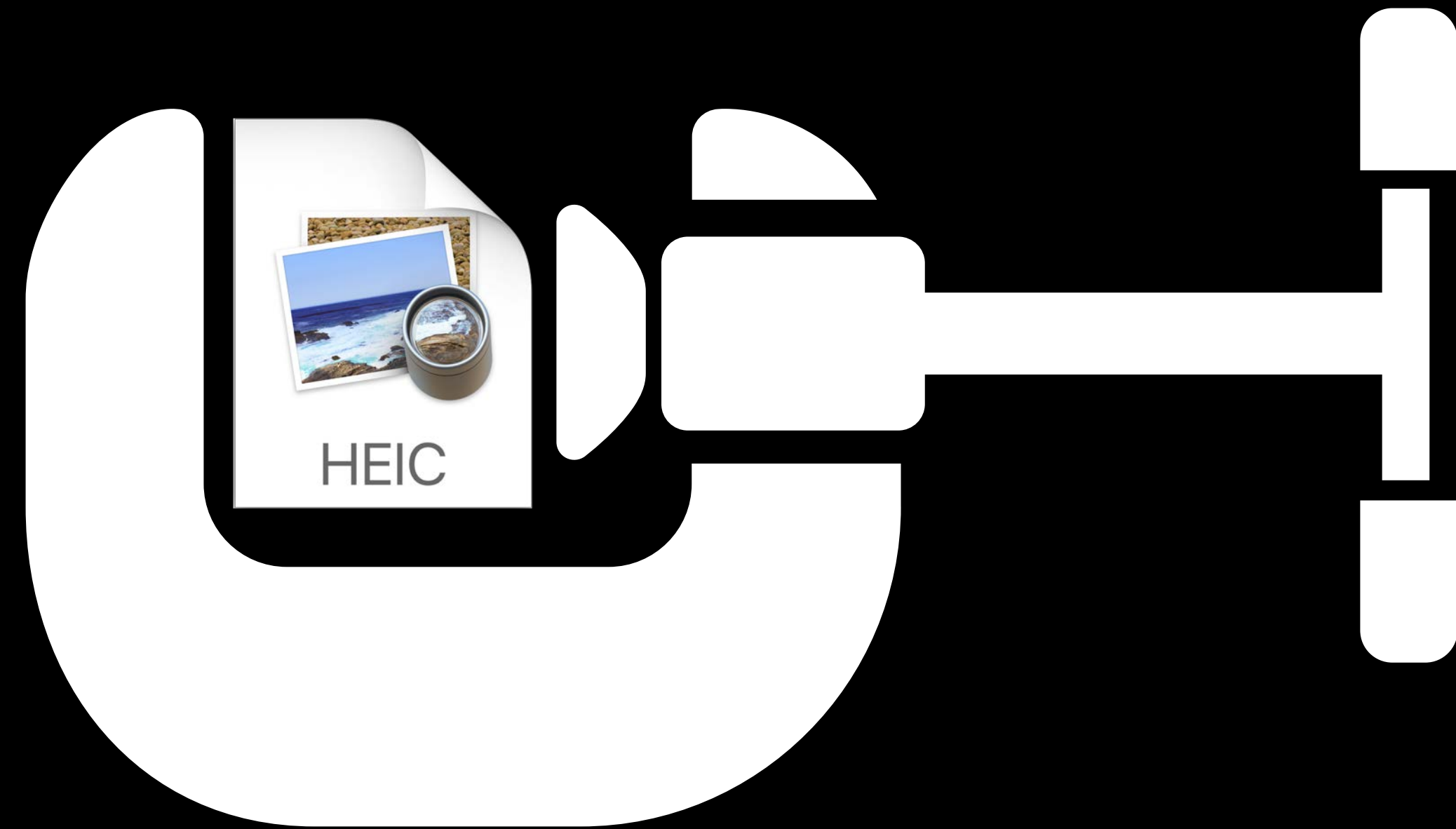
HEIF Pronunciation



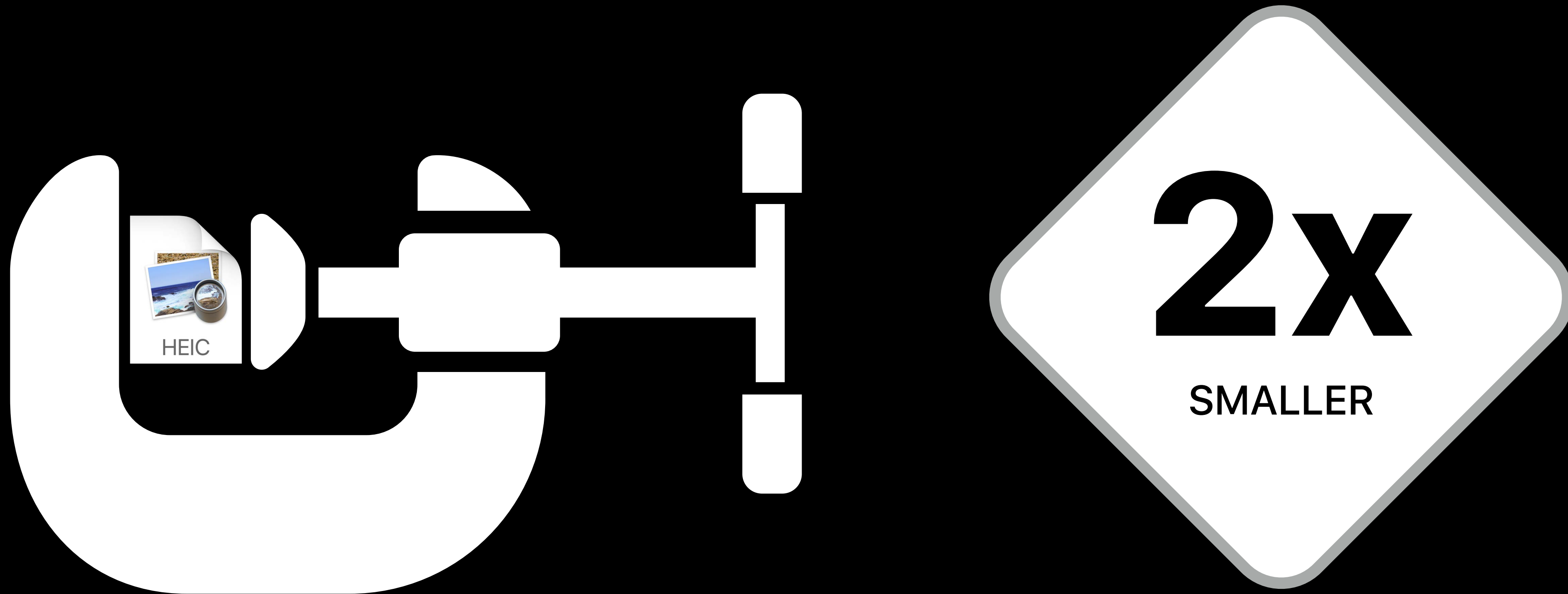
Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?



Why HEIF?

EXIF

xmp

Why HEIF?



Demo

Extreme zooming with HEIF

What the .HEIC?

Payload	Extension	UTI
HEVC Image	.HEIC	"public.heic"
H.264 Image	.AVCI	"public.avci"
Anything Else	.HEIF	"public.heif"

What the .HEIC?

Payload	Extension	UTI
HEVC Image	.HEIC	"public.heic"
H.264 Image	.AVCI	"public.avci"
Anything Else	.HEIF	"public.heif"

What the .HEIC?

Payload	Extension	UTI
HEVC Image	.HEIC	"public.heic"
H.264 Image	.AVCI	"public.avci"
Anything Else	.HEIF	"public.heif"

What the .HEIC?

Payload	Extension	UTI
HEVC Image	.HEIC	"public.heic"
H.264 Image	.AVCI	"public.avci"
Anything Else	.HEIF	"public.heif"

Supported HEIF Flavors (Writing)



Payload

Extension

UTI

HEVC Image

.HEIC

"public.heic"

Low-Level Access to HEIF

UIKit

CoreImage

CoreGraphics

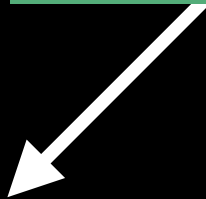
ImageIO

UIKit

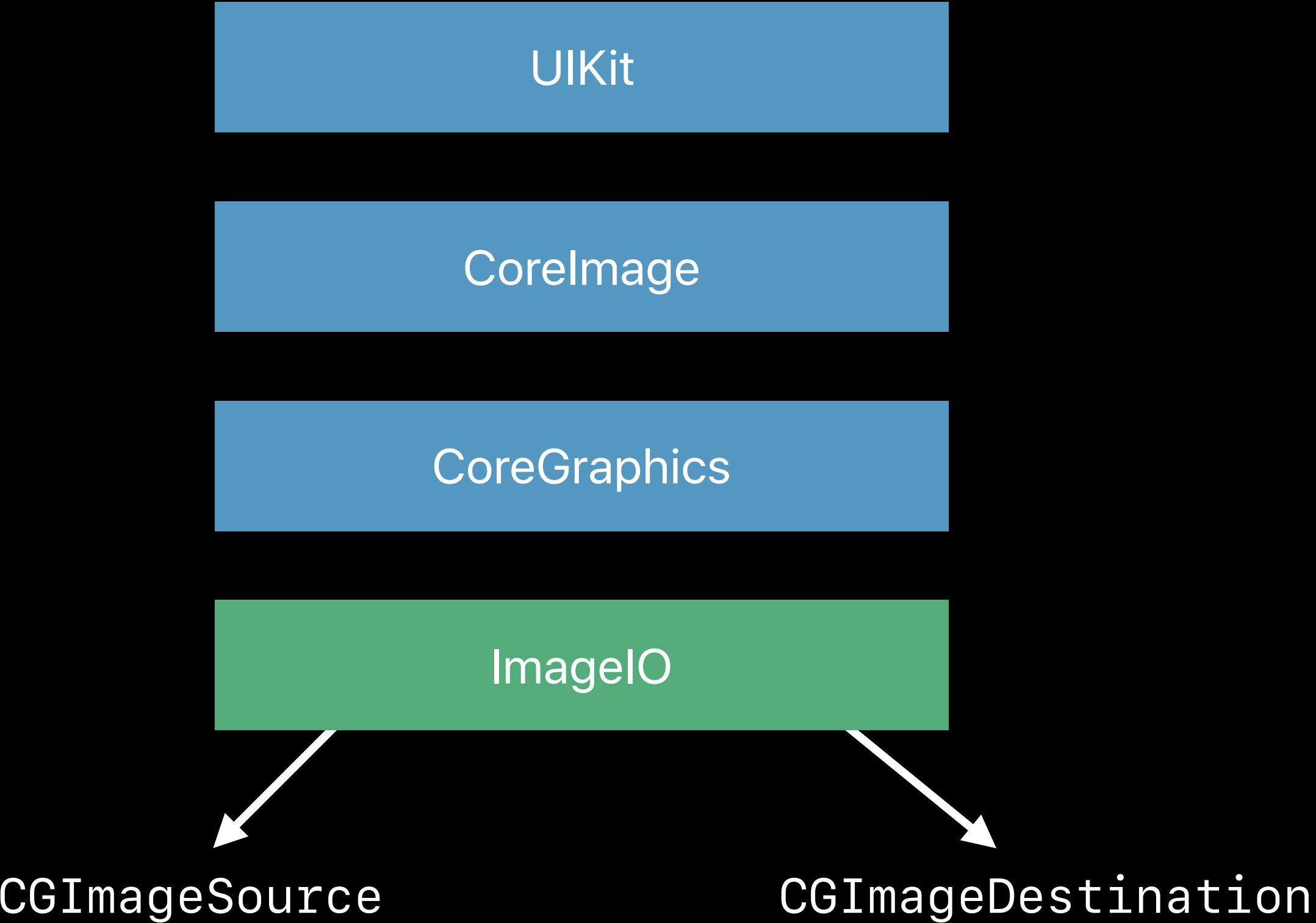
CoreImage

CoreGraphics

ImageIO



CGImageSource



```
// Read a jpeg image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```

```
// Read a jpeg image from file
```

```
let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")
```

```
let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)
```

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]
```

```
let image = CGImageSourceCreateImageAtIndex(source, 0, nil)
```

```
let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,  
              kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]
```

```
let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```

```
// Read a jpeg image from file
```

```
let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")
```

```
let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)
```

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]
```

```
let image = CGImageSourceCreateImageAtIndex(source, 0, nil)
```

```
let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,  
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]
```

```
let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```

```
// Read a jpeg image from file
```

```
let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")
```

```
let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)
```

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]
```

```
let image = CGImageSourceCreateImageAtIndex(source, 0, nil)
```

```
let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,  
              kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]
```

```
let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```



```
// Read a jpeg image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```

```
// Read a jpeg image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```

```
// Read a heic image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.heic")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```



```
// Read a heic image from file
```

```
let inputURL = URL(fileURLWithPath: "/tmp/image.heic")
```

```
let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)
```

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]
```

```
let image = CGImageSourceCreateImageAtIndex(source, 0, nil)
```

```
let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,  
              kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]
```

```
let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
```

Tiling Support in CGImageSource

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

"{TIFF}" = {
    DateTime = "2017:04:01 22:50:24";
    Make = Apple;
    Model = "iPhone 7 Plus";
    Orientation = 1;
    ResolutionUnit = 2;
    Software = "11.0";
    TileLength = 512;
    TileWidth = 512;
    XResolution = 72;
    YResolution = 72;
};
```

Tiling Support in CGImageSource

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]
```

```
"{TIFF}" = {  
    DateTime = "2017:04:01 22:50:24";  
    Make = Apple;  
    Model = "iPhone 7 Plus";  
    Orientation = 1;  
    ResolutionUnit = 2;  
    Software = "11.0";  
    TileLength = 512;  
    TileWidth = 512;  
    XResolution = 72;  
    YResolution = 72;  
};
```

Tiling Support in CGImageSource

```
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

"{TIFF}" = {
    DateTime = "2017:04:01 22:50:24";
    Make = Apple;
    Model = "iPhone 7 Plus";
    Orientation = 1;
    ResolutionUnit = 2;
    Software = "11.0";
    TileLength = 512;
    TileWidth = 512;
    XResolution = 72;
    YResolution = 72;
};
```


Tiling Support in CGImage

```
let subImage = bigImage.cropping(to: rect)
```



Tiling Support in CGImage

```
let subImage = bigImage.cropping(to: rect)
```



```
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
                                                         AVFileType.jpg as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
    }

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
```

```
// Writing a CGImage to a JPEG file
```

```
let url = URL(fileURLWithPath: "/tmp/output.jpg")
```

```
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,  
                                                         AVFileType.jpeg as CFString, 1, nil)
```

```
    else {
```

```
        fatalError("unable to create CGImageDestination")
```

```
    }
```

```
CGImageDestinationAddImage(imageDestination, image, nil)
```

```
CGImageDestinationFinalize(imageDestination)
```

```
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
                                                         AVFileType.jpg as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
    }
```

```
CGImageDestinationAddImage(imageDestination, image, nil)
```

```
CGImageDestinationFinalize(imageDestination)
```

```
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
                                                         AVFileType.jpg as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
    }

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
```

```
// Writing a CGImage to a HEIC file

let url = URL(fileURLWithPath: "/tmp/output.heic")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
                                                         AVFileType.heic as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
    }

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
```



```
// Writing a CGImage to a HEIF file

let url = URL(fileURLWithPath: "/tmp/output.heic")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
                                                         AVFileType.heic as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
    }

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
```

```
// Writing a CGImage to a HEIF file
```

```
let url = URL(fileURLWithPath: "/tmp/output.heic")
```

```
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,  
                                                         AVFileType.heic as CFString, 1, nil)  
  else {  
    fatalError("unable to create CGImageDestination")  
  }
```

```
CGImageDestinationAddImage(imageDestination, image, nil)
```

```
CGImageDestinationFinalize(imageDestination)
```

ImageIO Depth Support

NEW

HEIC

- Auxiliary image (monochrome HEVC) with XMP metadata

JPEG

- MPO image (jpeg encoded) with XMP metadata


High-Level Access to HEIF



Compression Poetry

Brad Ford



A landscape photograph featuring a vast sea of clouds in shades of blue and white, with numerous dark, jagged rock formations rising from the clouds. The sky is a mix of dark and light tones, suggesting a sunset or sunrise. The text is overlaid on the left side of the image.

JPEG is yay big
but
HEIF is brief



Compression Haiku

Brad Ford

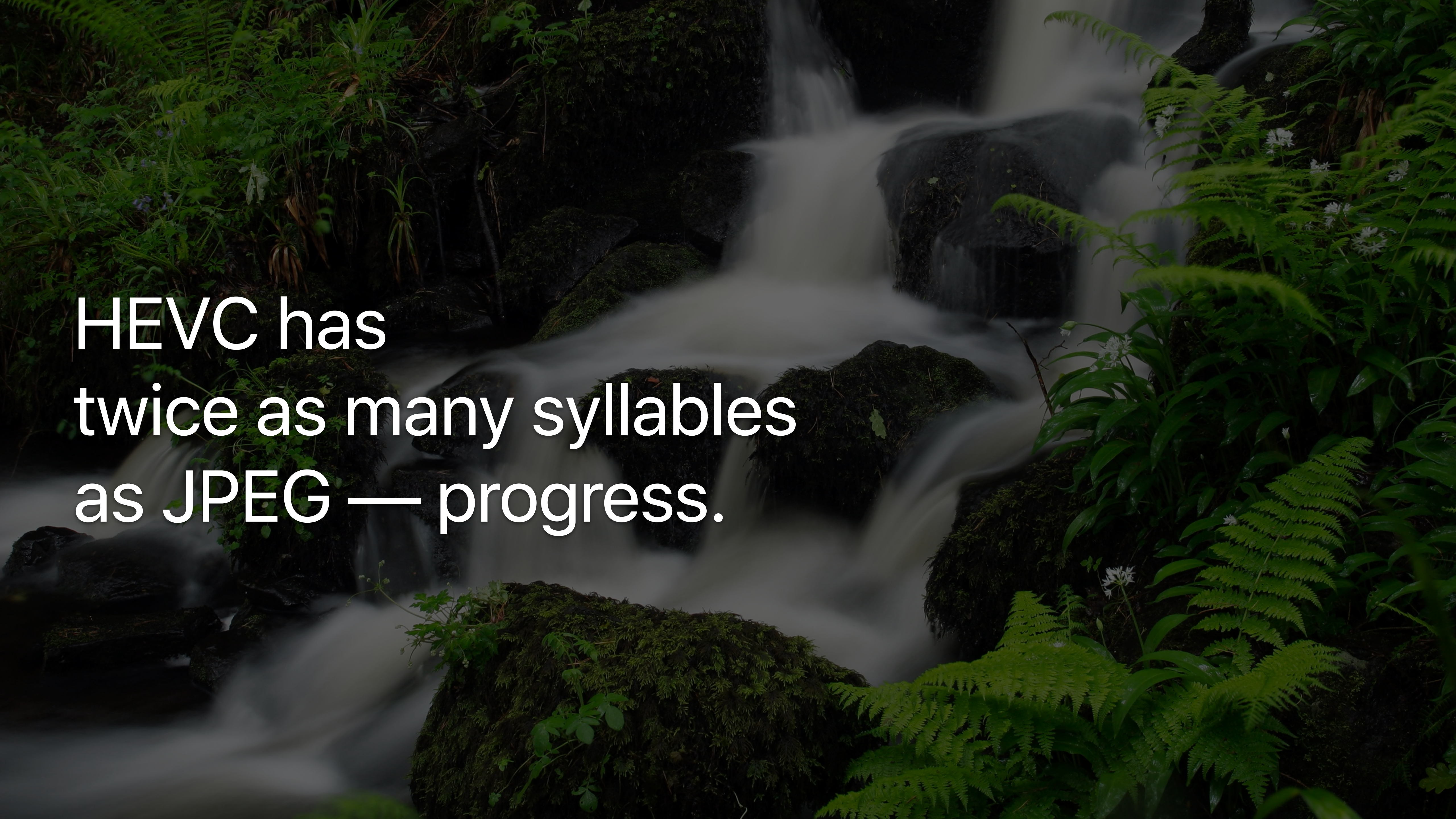


A long-exposure photograph of a waterfall cascading over mossy rocks in a lush forest. The water is blurred, creating a soft, ethereal effect. The surrounding vegetation is dense and vibrant green, with various ferns and small white flowers visible. The overall scene is serene and natural.

HEVC has

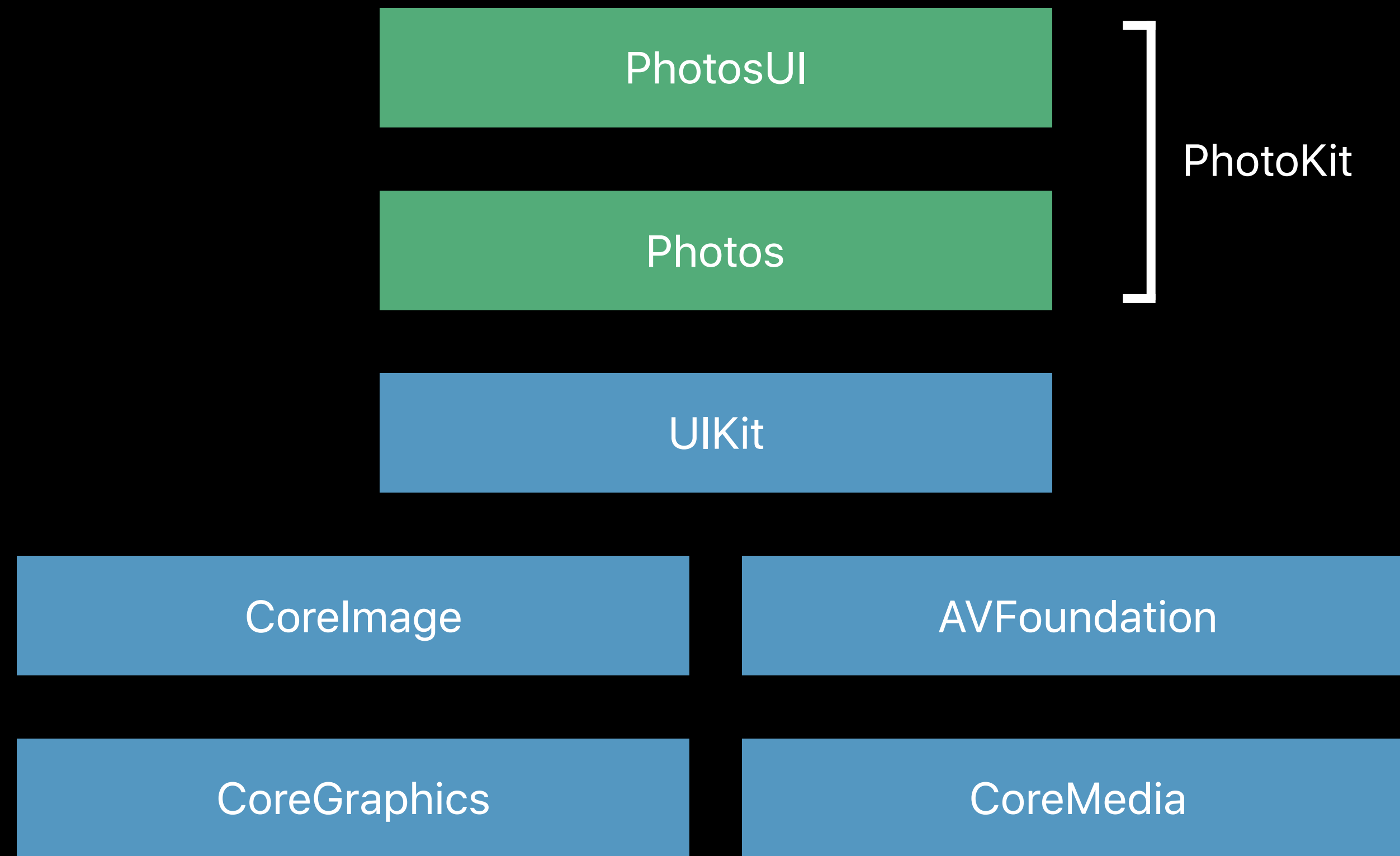


HEVC has
twice as many syllables



HEVC has
twice as many syllables
as JPEG — progress.

HEIF and PhotoKit



HEIF and PhotoKit

Applying adjustments

- Photos
- Videos
- Live Photos

Common workflows

- Display
- Backup
- Share

Apply Adjustments

PHPhotoLibrary performChanges:

Apply Adjustments

PHPhotoLibrary performChanges:

PHAsset



Apply Adjustments

PHPhotoLibrary performChanges:

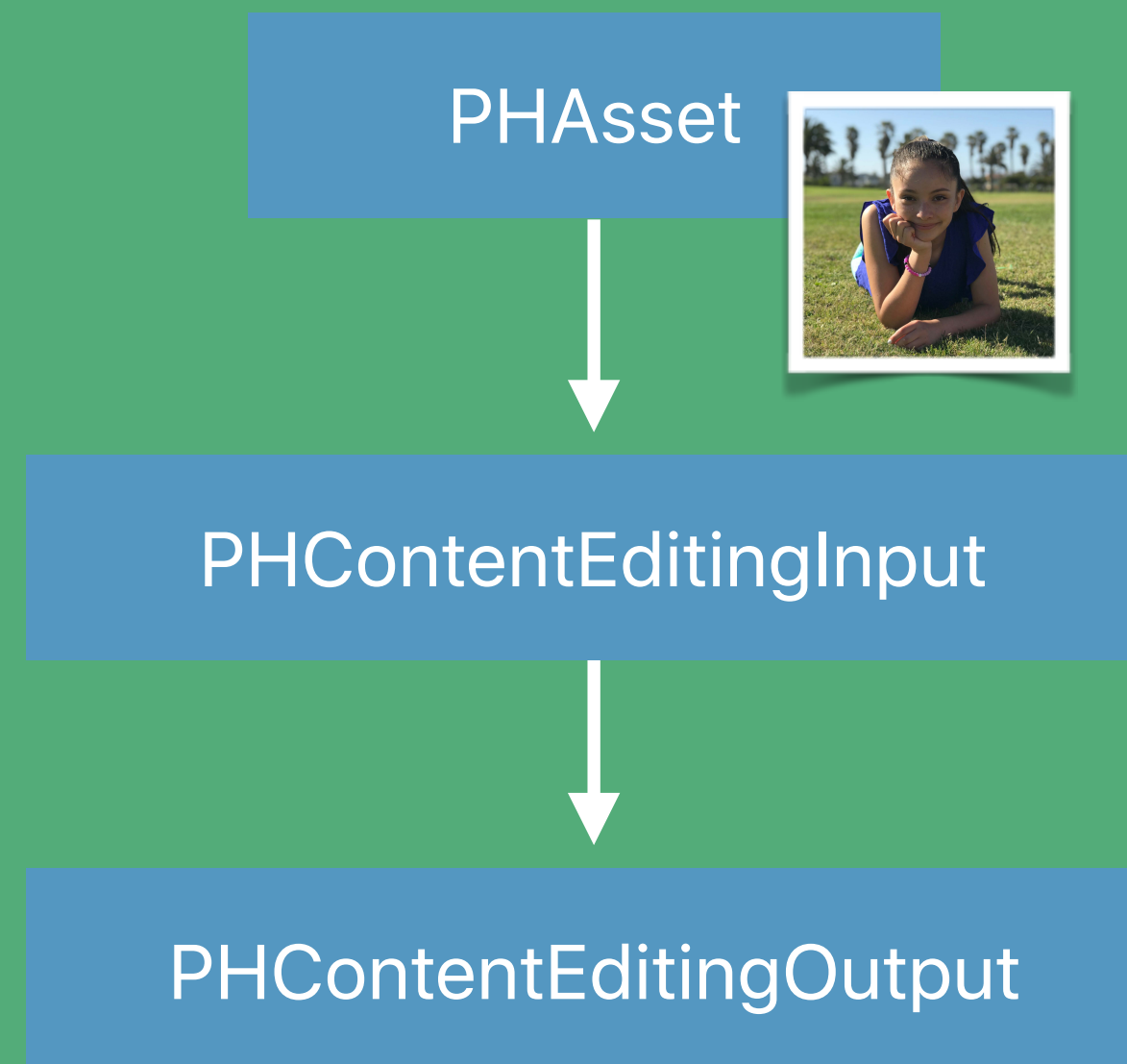
PHAsset



PHContentEditingInput

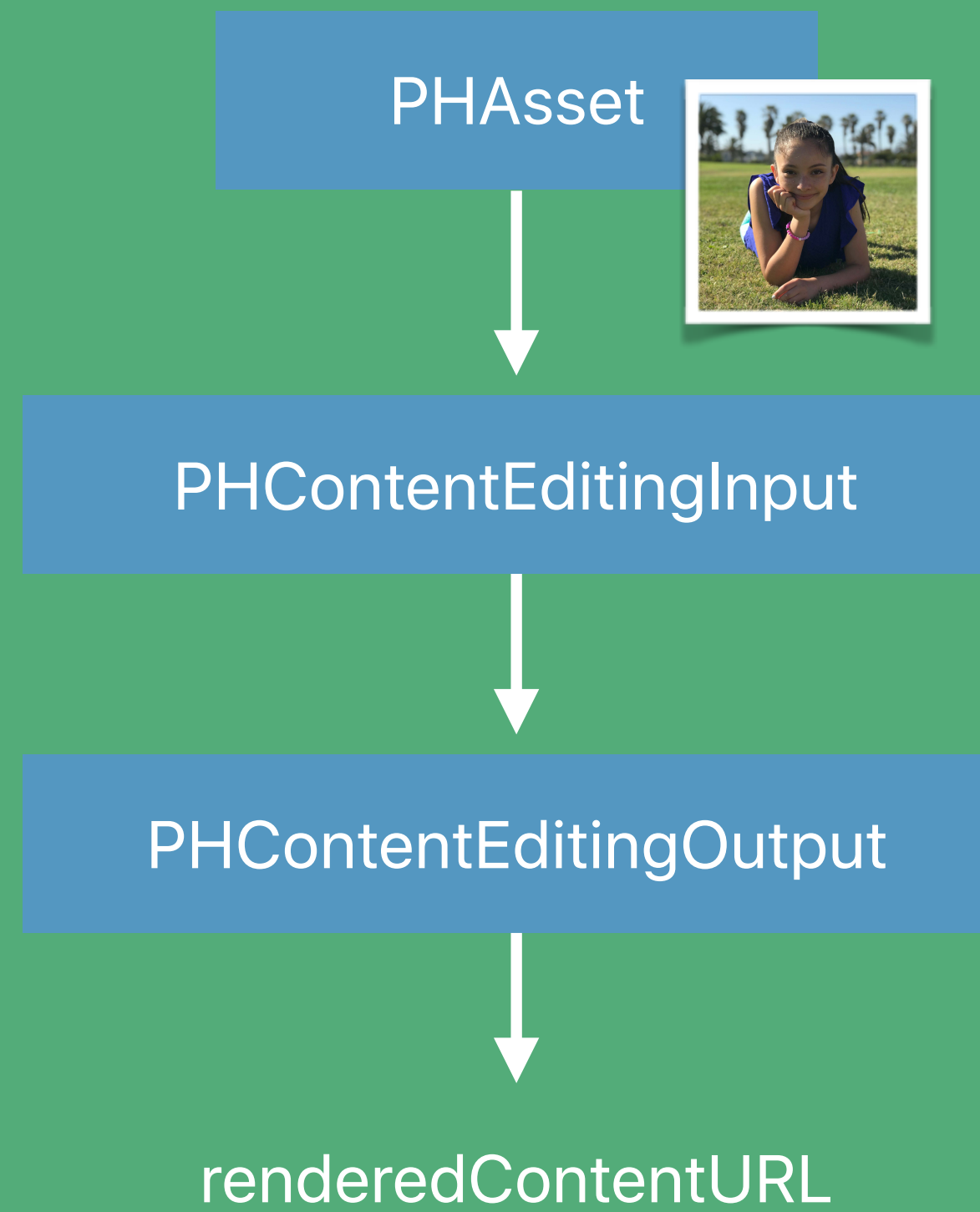
Apply Adjustments

PHPhotoLibrary performChanges:



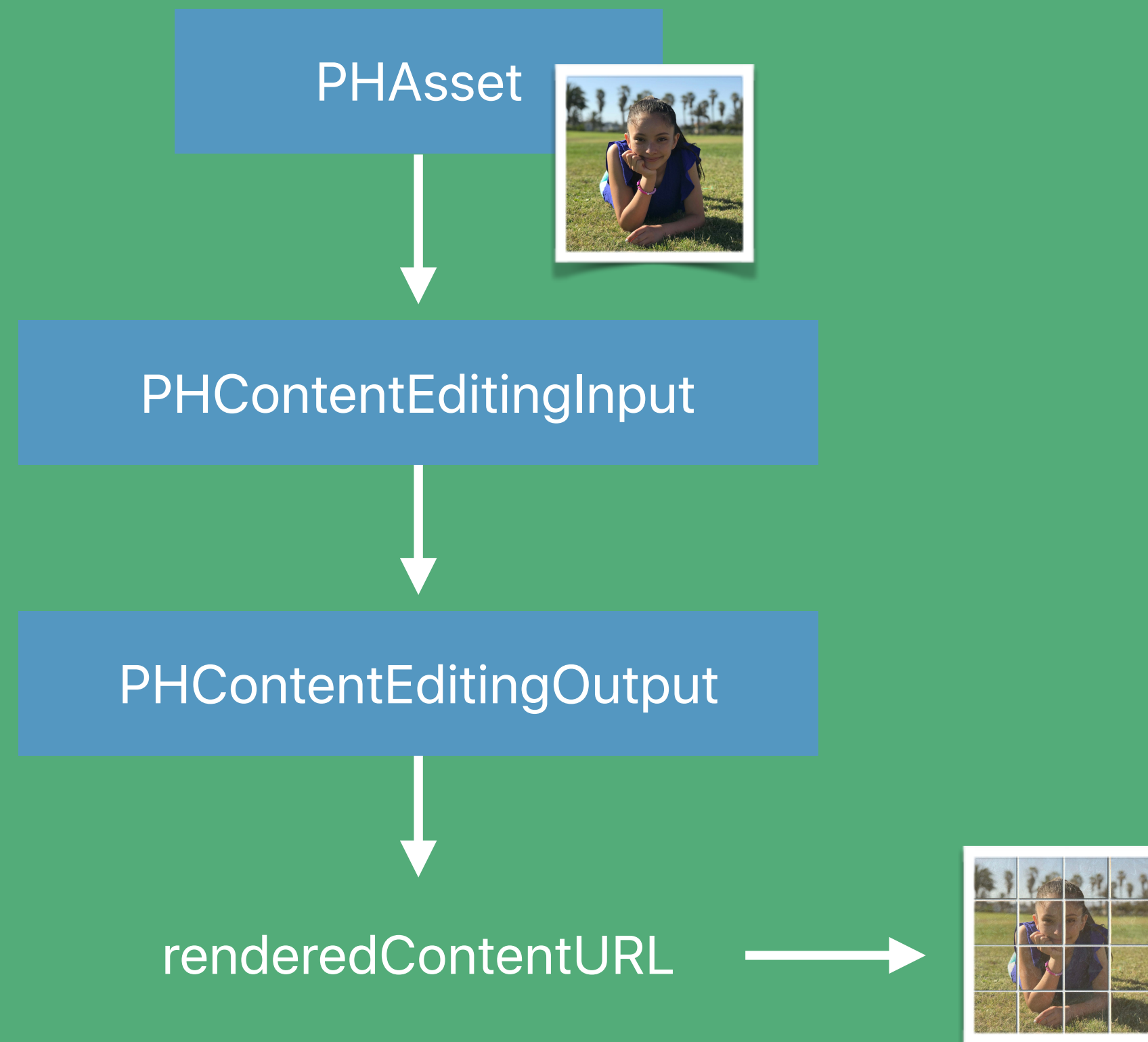
Apply Adjustments

PHPhotoLibrary performChanges:



Apply Adjustments

PHPhotoLibrary performChanges:



Applying Adjustments

PHContentEditingOutput

- Image rendered as JPEG
- Image Exif orientation of 1 (no rotation)

```
// Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output:
PHContentEditingOutput, completion: () -> ()) {

    guard let inputImage = UIImage(contentsOf: input.fullSizeImageURL!)
        else { fatalError("can't load input image") }
    let outputImage = inputImage
        .applyingOrientation(input.fullSizeImageOrientation)
        .applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage,
        to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)") }
    completion()
}
```

```
// Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output:
PHContentEditingOutput, completion: () -> ()) {

    guard let inputImage = UIImage(contentsOf: input.fullSizeImageURL!)
        else { fatalError("can't load input image") }

    let outputImage = inputImage
        .applyingOrientation(input.fullSizeImageOrientation)
        .applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage,
        to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)") }
    completion()
}
```

```
// Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output:
PHContentEditingOutput, completion: () -> ()) {

    guard let inputImage = UIImage(contentsOf: input.fullSizeImageURL!)
        else { fatalError("can't load input image") }

    let outputImage = inputImage
        .applyingOrientation(input.fullSizeImageOrientation)
        .applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage,
        to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)") }
    completion()
}
```



```
// Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output:
PHContentEditingOutput, completion: () -> ()) {

    guard let inputImage = UIImage(contentsOf: input.fullSizeImageURL!)
        else { fatalError("can't load input image") }
    let outputImage = inputImage
        .applyingOrientation(input.fullSizeImageOrientation)
        .applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage,
        to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)") }
    completion()
}
```

Applying Adjustments

PHContentEditingOutput

- Video rendered as H.264

```
// Editing an HEVC video -- save as H.264

func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput,
completionHandler: @escaping () -> ()) {

    guard let avAsset = input.audiovisualAsset
        else { fatalError("can't get AV asset") }
    let composition = AVVideoComposition(asset: avAsset,
        applyingCIFiltersWithHandler: { request in
        let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
        request.finish(with: img, context: nil)
    })
    // Export the video composition to the output URL.
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality)
        else { fatalError("can't set up AV export session") }
    export.outputFileType = AVFileType.mov
    export.outputURL = output.renderedContentURL
    export.videoComposition = composition
    export.exportAsynchronously(completionHandler: completionHandler)
}
```

```
// Editing an HEVC video -- save as H.264
```

```
func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completionHandler: @escaping () -> ()) {
```

```
    guard let avAsset = input.audiovisualAsset
          else { fatalError("can't get AV asset") }
```

```
    let composition = AVVideoComposition(asset: avAsset,
    applyingCIFiltersWithHandler: { request in
```

```
        let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
        request.finish(with: img, context: nil)
```

```
    })
```

```
    // Export the video composition to the output URL.
```

```
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality)
          else { fatalError("can't set up AV export session") }
```

```
    export.outputFileType = AVFileType.mov
```

```
    export.outputURL = output.renderedContentURL
```

```
    export.videoComposition = composition
```

```
    export.exportAsynchronously(completionHandler: completionHandler)
```

```
}
```

```
// Editing an HEVC video -- save as H.264

func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput,
completionHandler: @escaping () -> ()) {

    guard let avAsset = input.audiovisualAsset
        else { fatalError("can't get AV asset") }

    let composition = AVVideoComposition(asset: avAsset,
        applyingCIFiltersWithHandler: { request in
            let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
            request.finish(with: img, context: nil)
        })

    // Export the video composition to the output URL.
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality)
        else { fatalError("can't set up AV export session") }
    export.outputFileType = AVFileType.mov
    export.outputURL = output.renderedContentURL
    export.videoComposition = composition
    export.exportAsynchronously(completionHandler: completionHandler)
}
```

```
// Editing an HEVC video -- save as H.264
```

```
func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completionHandler: @escaping () -> ()) {
```

```
    guard let avAsset = input.audiovisualAsset
```

```
        else { fatalError("can't get AV asset") }
```

```
    let composition = AVVideoComposition(asset: avAsset,
```

```
        applyingCIFiltersWithHandler: { request in
```

```
            let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
```

```
            request.finish(with: img, context: nil)
```

```
        })
```

```
    // Export the video composition to the output URL.
```

```
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality)
```

```
        else { fatalError("can't set up AV export session") }
```

```
    export.outputFileType = AVFileType.mov
```

```
    export.outputURL = output.renderedContentURL
```

```
    export.videoComposition = composition
```

```
    export.exportAsynchronously(completionHandler: completionHandler)
```

```
}
```

Applying Adjustments

PHLivePhotoEditingContext

- Uses CImage frames, automatically converted

```
// Editing a HEIF/HEVC live photo -- format handled automatically

func applyLivePhotoFilter(_ filterName: String, input: PHContentEditingInput, output:
PHContentEditingOutput, completion: @escaping () -> ()) {

    guard let livePhotoContext = PHLivePhotoEditingContext(livePhotoEditingInput: input)
        else { fatalError("can't get live photo") }
    livePhotoContext.frameProcessor = { frame, _ in
        return frame.image.applyingFilter(filterName, withInputParameters: nil)
    }
    livePhotoContext.saveLivePhoto(to: output) { success, error in
        if success { completion() }
        else { print("can't output live photo") }
    }
}
```



```
// Editing a HEIF/HEVC live photo -- format handled automatically

func applyLivePhotoFilter(_ filterName: String, input: PHContentEditingInput, output:
PHContentEditingOutput, completion: @escaping () -> ()) {

    guard let livePhotoContext = PHLivePhotoEditingContext(livePhotoEditingInput: input)
        else { fatalError("can't get live photo") }
    livePhotoContext.frameProcessor = { frame, _ in
        return frame.image.applyingFilter(filterName, withInputParameters: nil)
    }
    livePhotoContext.saveLivePhoto(to: output) { success, error in
        if success { completion() }
        else { print("can't output live photo") }
    }
}
```

Common PhotoKit Workflows

Display

- PHImageManager for UIImage, AVPlayerItem, or PHLivePhoto
- No code changes needed

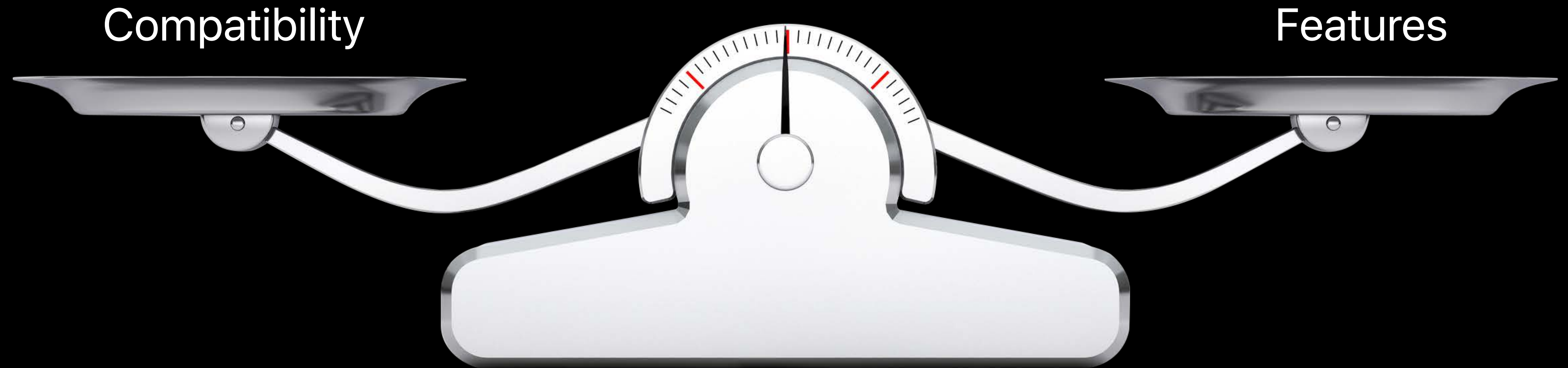
Common PhotoKit Workflows

Backup

- PHAssetResourceManager provides resources in native format

Common PhotoKit Workflows

Sharing



Leaving the Garden

```
// for images, check the UTI and convert if needed:
PHImageManager.default().requestImageData(for: asset, options: options, resultHandler:
    { imageData, dataUTI, orientation, info in
    guard let dataUTI = dataUTI else { return }
    if !UTTypeConformsTo(dataUTI as CFString, kUTTypeJPEG {
        // convert the data to a JPEG representation...
    }
// for videos use export preset to specify the format
PHImageManager.default().requestExportSession(forVideo: asset, options: options,
    exportPreset: AVAssetExportPresetHighestQuality, resultHandler: { exportSession, info in
```

Leaving the Garden

```
// for images, check the UTI and convert if needed:
PHImageManager.default().requestImageData(for: asset, options: options, resultHandler:
    { imageData, dataUTI, orientation, info in
    guard let dataUTI = dataUTI else { return }
    if !UTTypeConformsTo(dataUTI as CFString, kUTTypeJPEG {
        // convert the data to a JPEG representation...
    }
// for videos use export preset to specify the format
PHImageManager.default().requestExportSession(forVideo: asset, options: options,
    exportPreset: AVAssetExportPresetHighestQuality, resultHandler: { exportSession, info in
```

Capturing HEIF



Compression Haiku Two

Brad Ford





HEIF, a container,

A scenic landscape featuring a large, dark mountain peak on the left and a smaller, snow-capped peak on the right. The sky is filled with soft, horizontal bands of orange and yellow light, suggesting a sunset or sunrise. The foreground is a calm body of water that perfectly reflects the mountains and the sky. The overall mood is serene and majestic.

HEIF, a container,
compresses four times better

A scenic landscape featuring a large, dark mountain peak on the left and a smaller, snow-capped peak on the right. The sky is filled with soft, horizontal bands of orange and yellow light, suggesting a sunset or sunrise. The foreground is a calm body of water that perfectly reflects the mountains and the sky. The text is overlaid in the center-left area of the image.

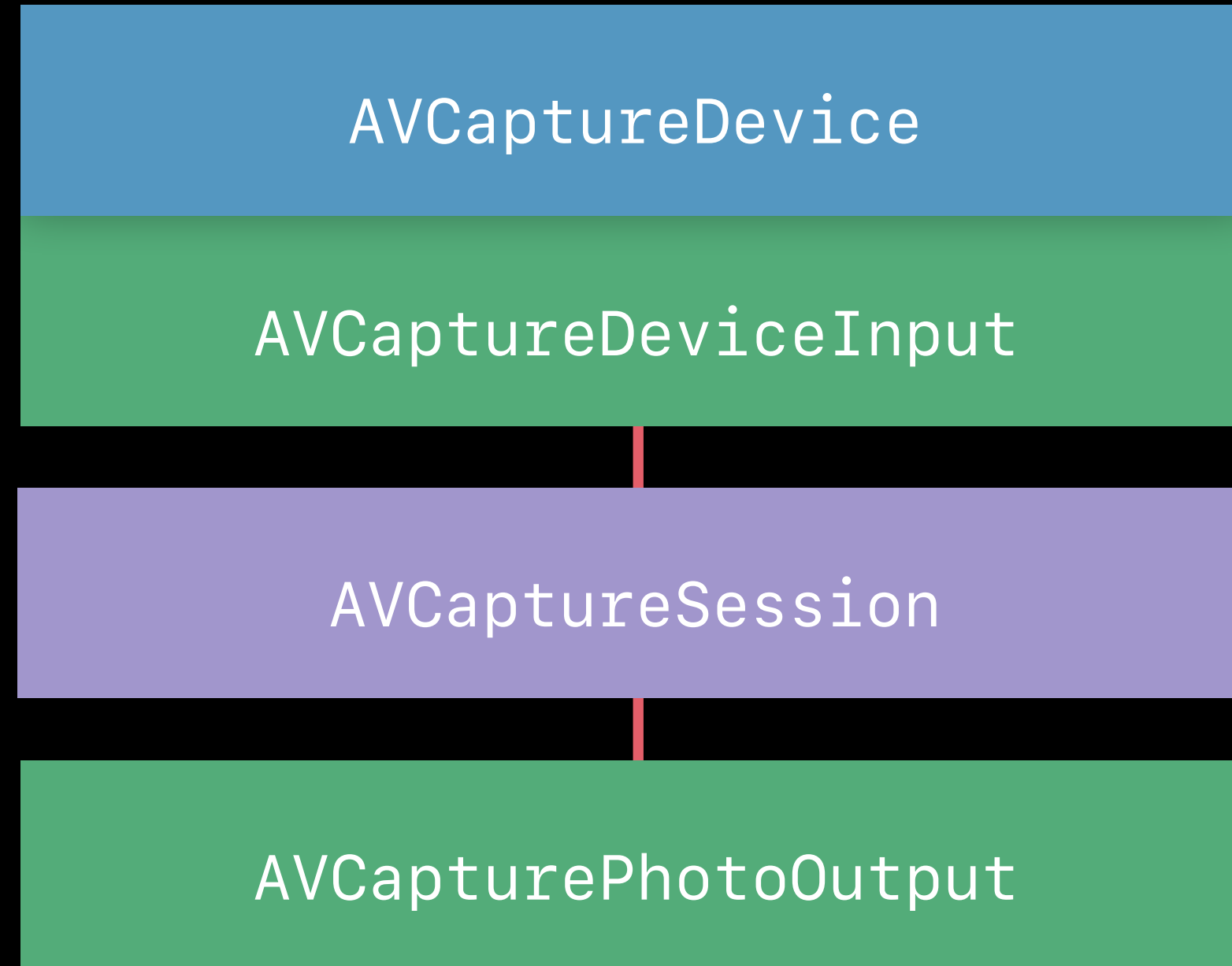
HEIF, a container,
compresses four times better
than HEVC

AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

AVCapturePhotoOutput



AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

AVCapturePhotoOutput

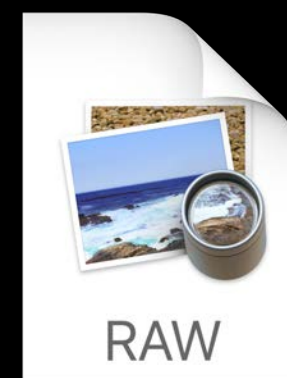


AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

AVCapturePhotoOutput

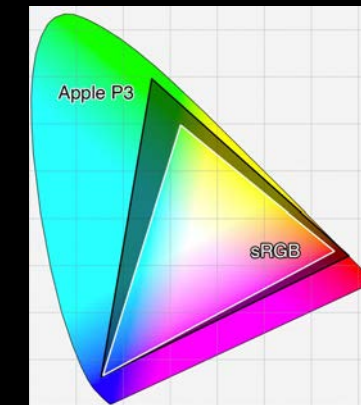
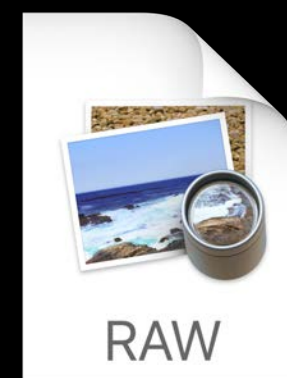


AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

AVCapturePhotoOutput



HEIF Capture Support

iPhone 7 Plus

iPhone 7

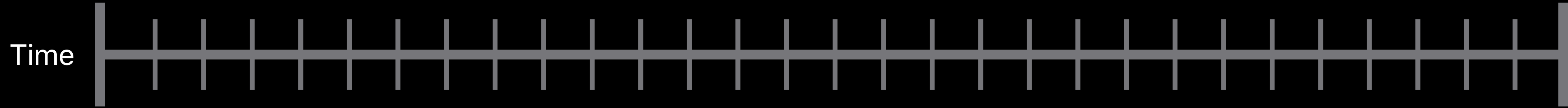
10.5-inch iPad Pro

NEW

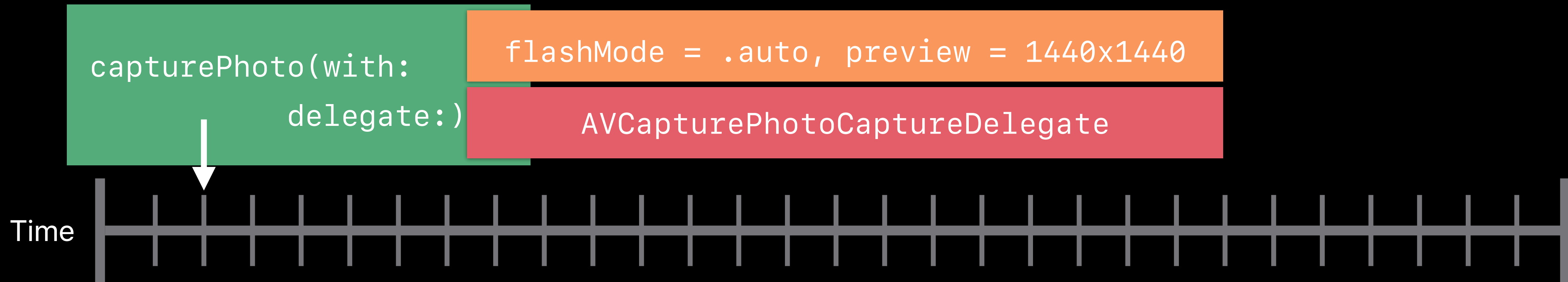
12.9-inch iPad Pro

NEW

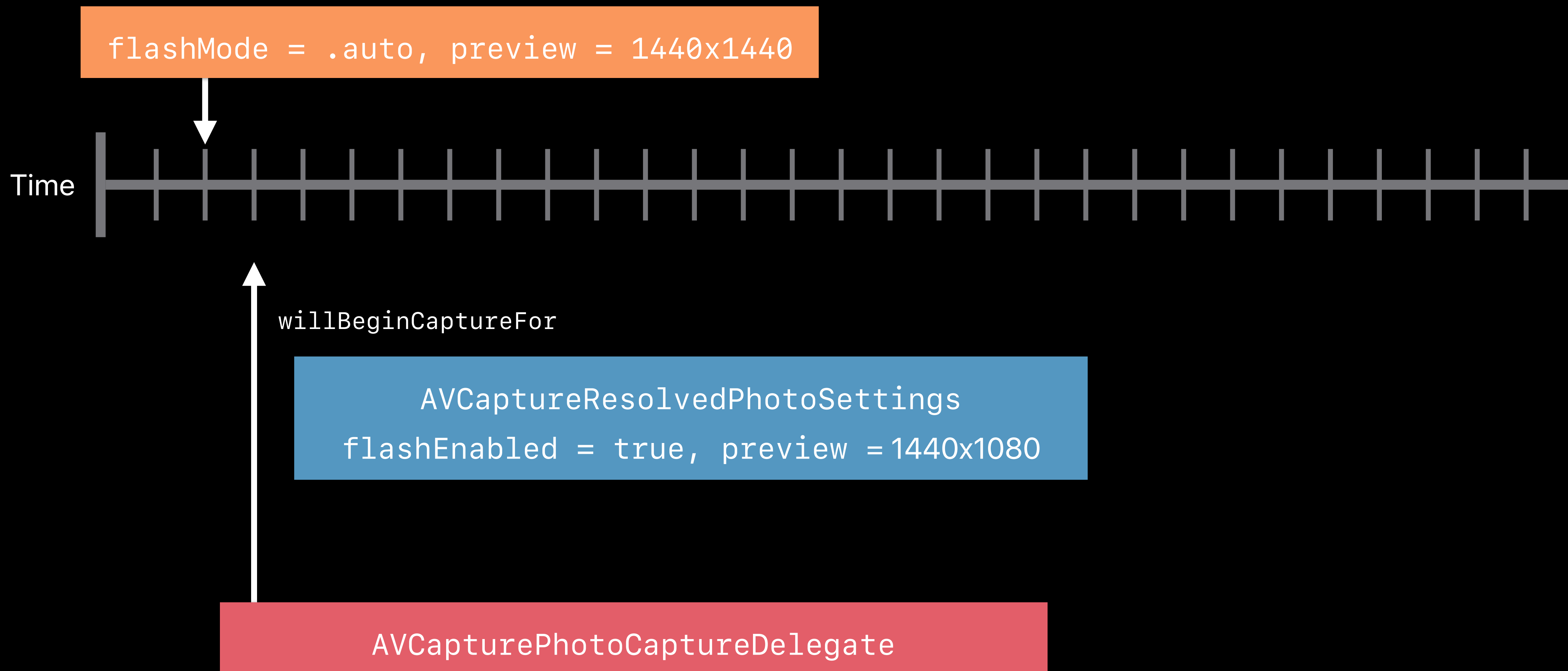
AVCapturePhotoOutput Usage



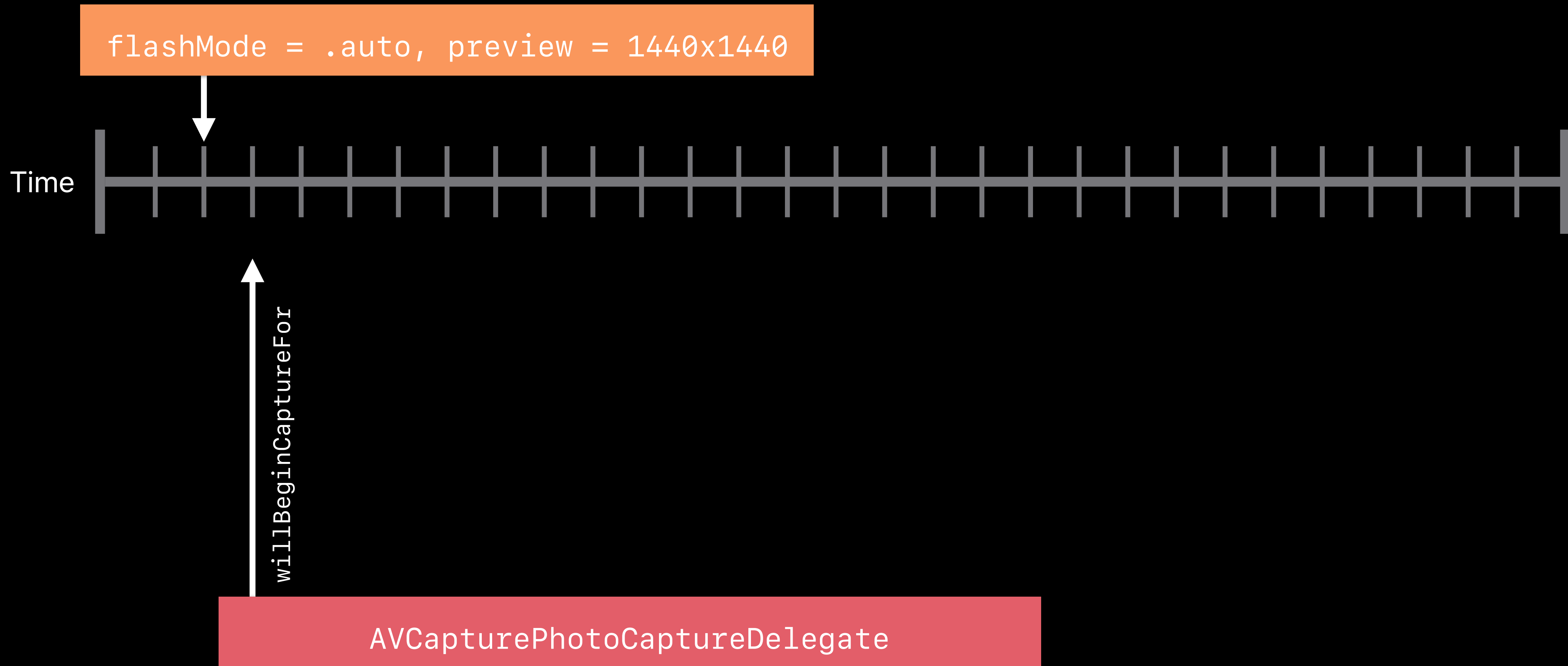
AVCapturePhotoOutput Usage



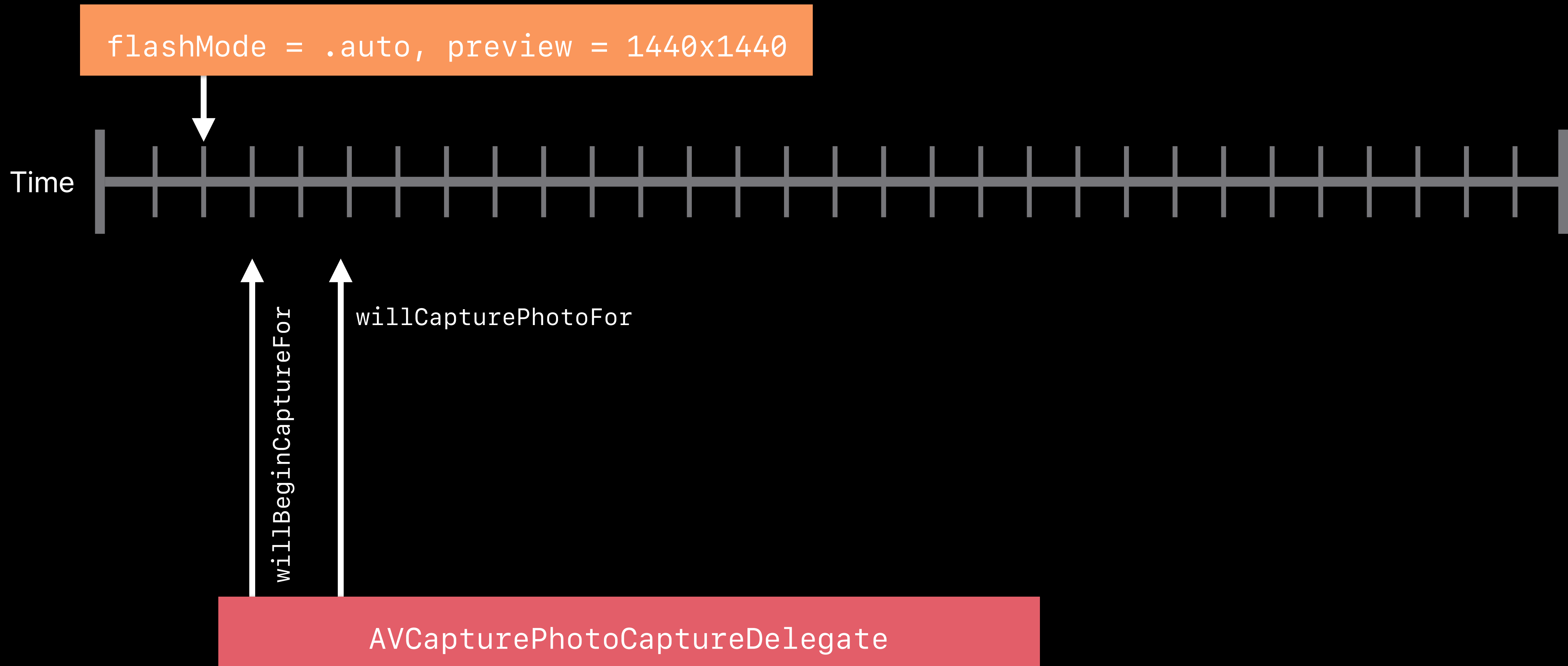
AVCapturePhotoOutput Usage



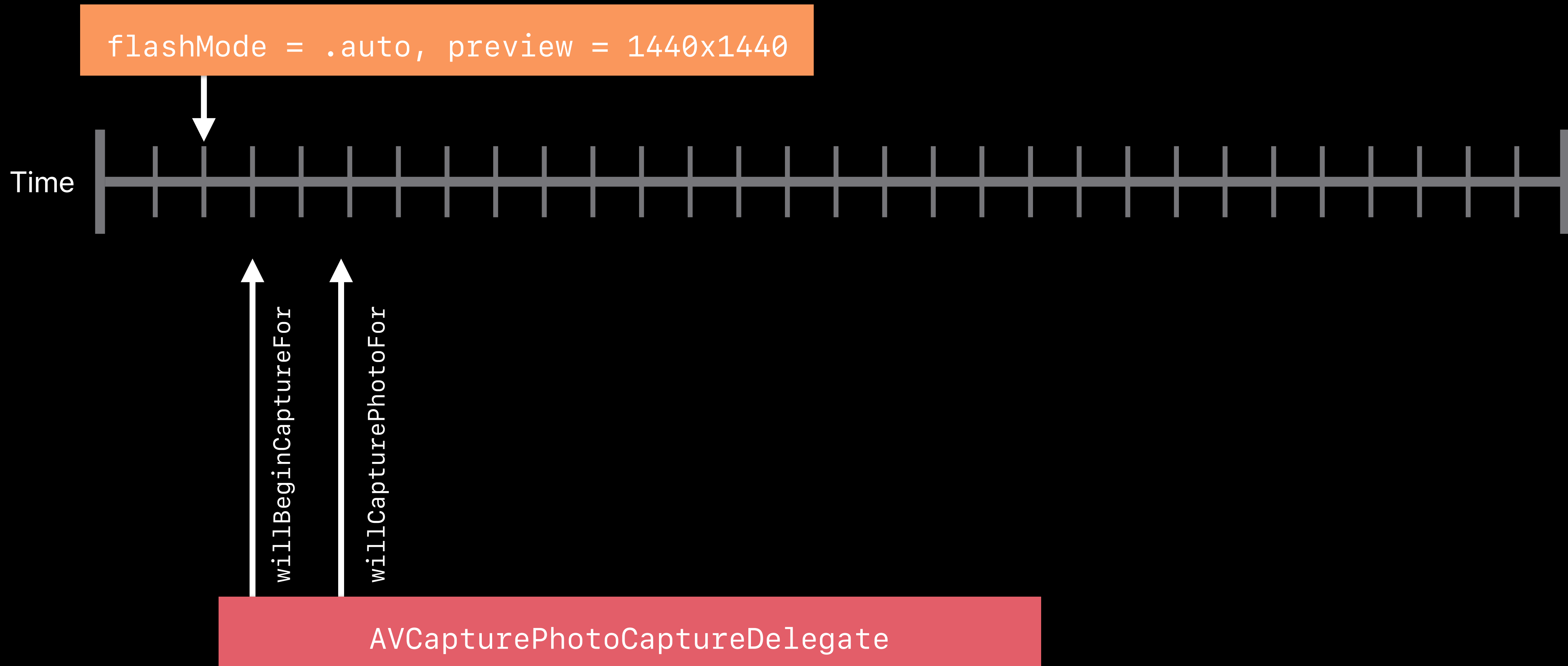
AVCapturePhotoOutput Usage



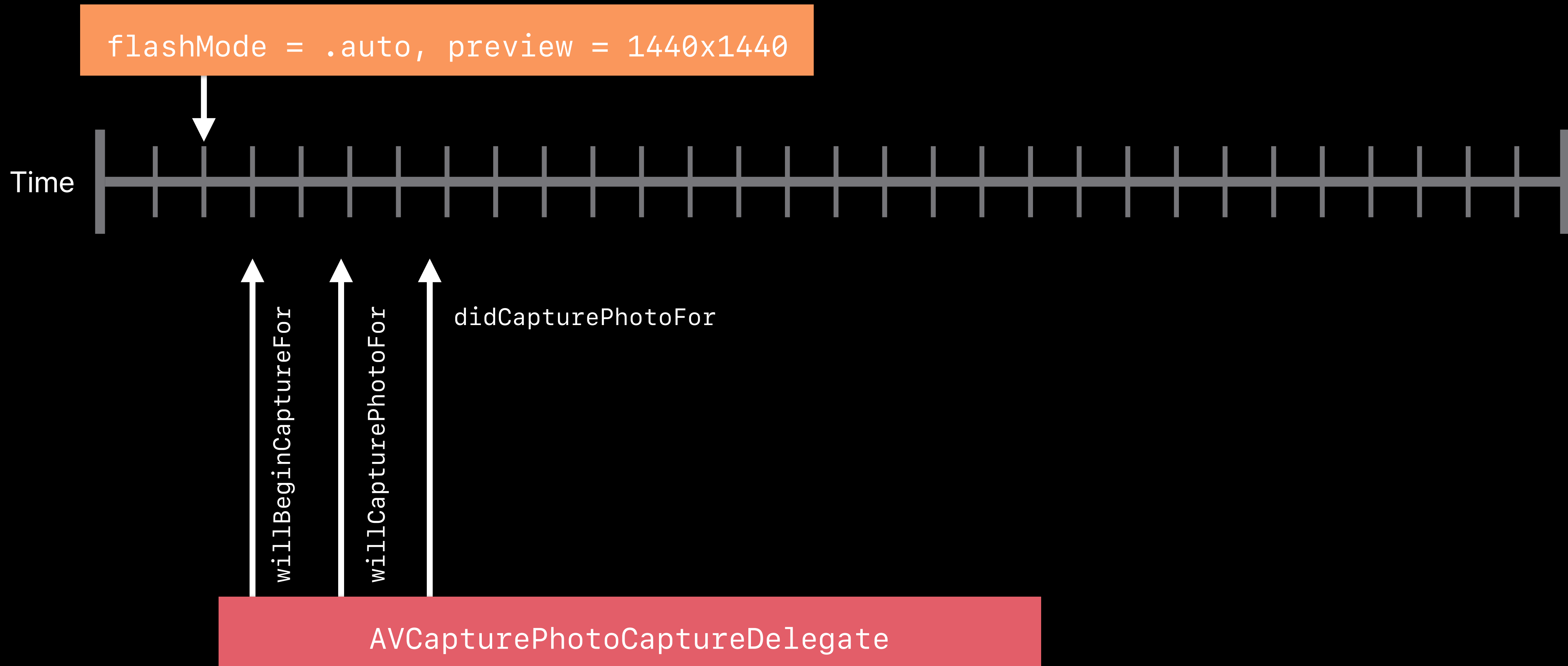
AVCapturePhotoOutput Usage



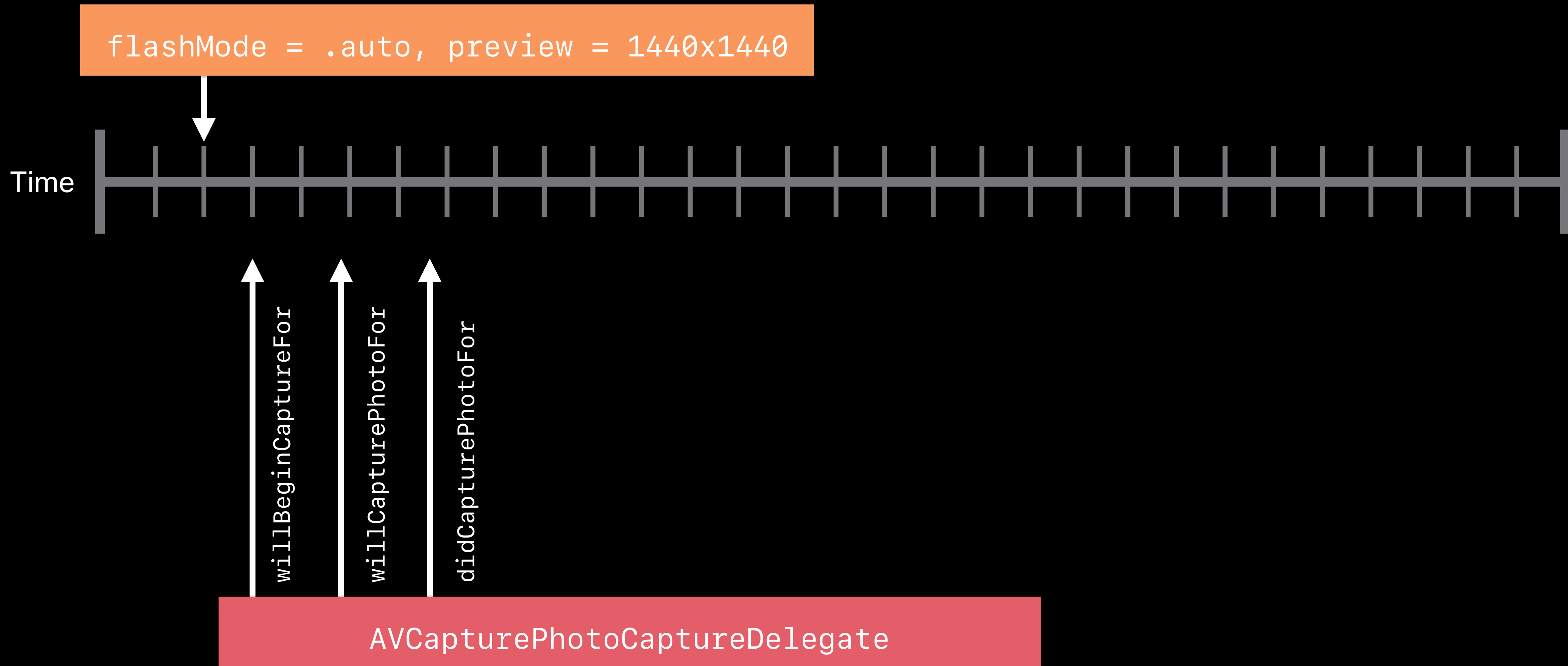
AVCapturePhotoOutput Usage



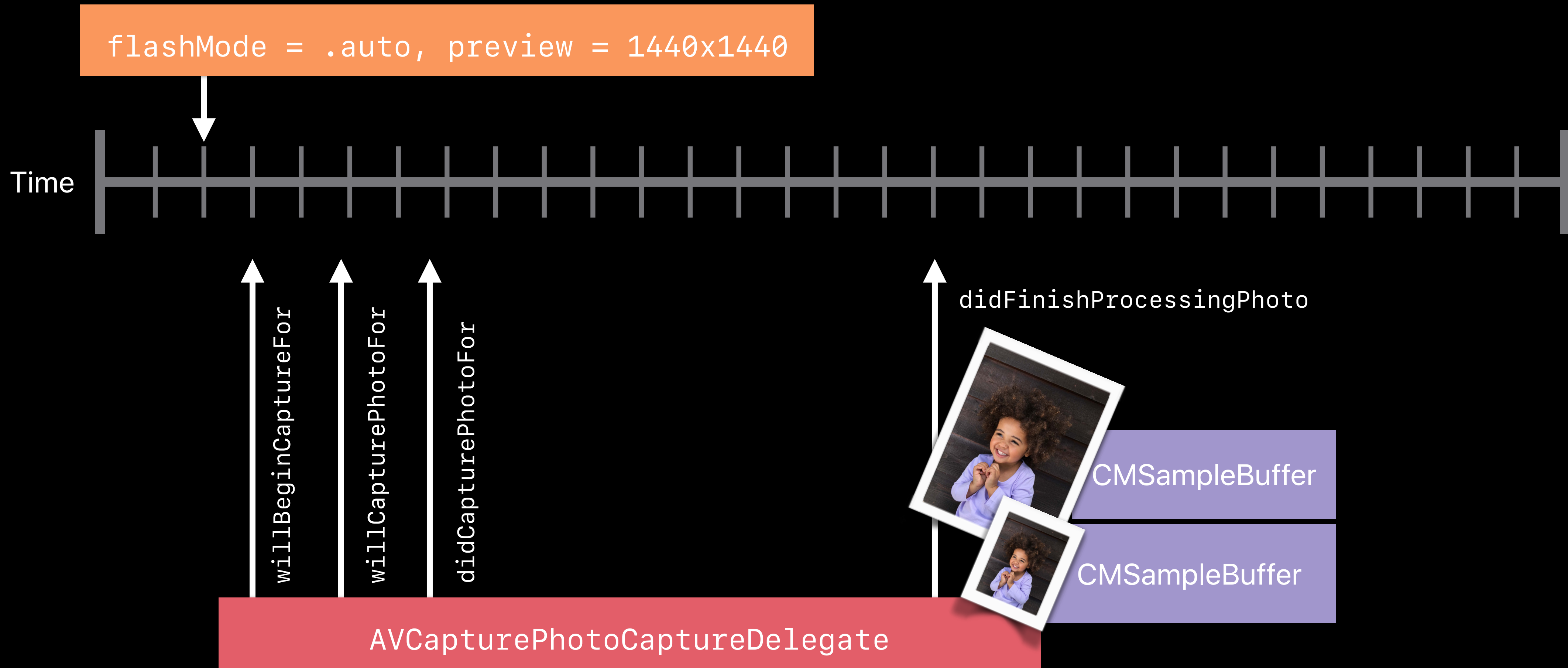
AVCapturePhotoOutput Usage



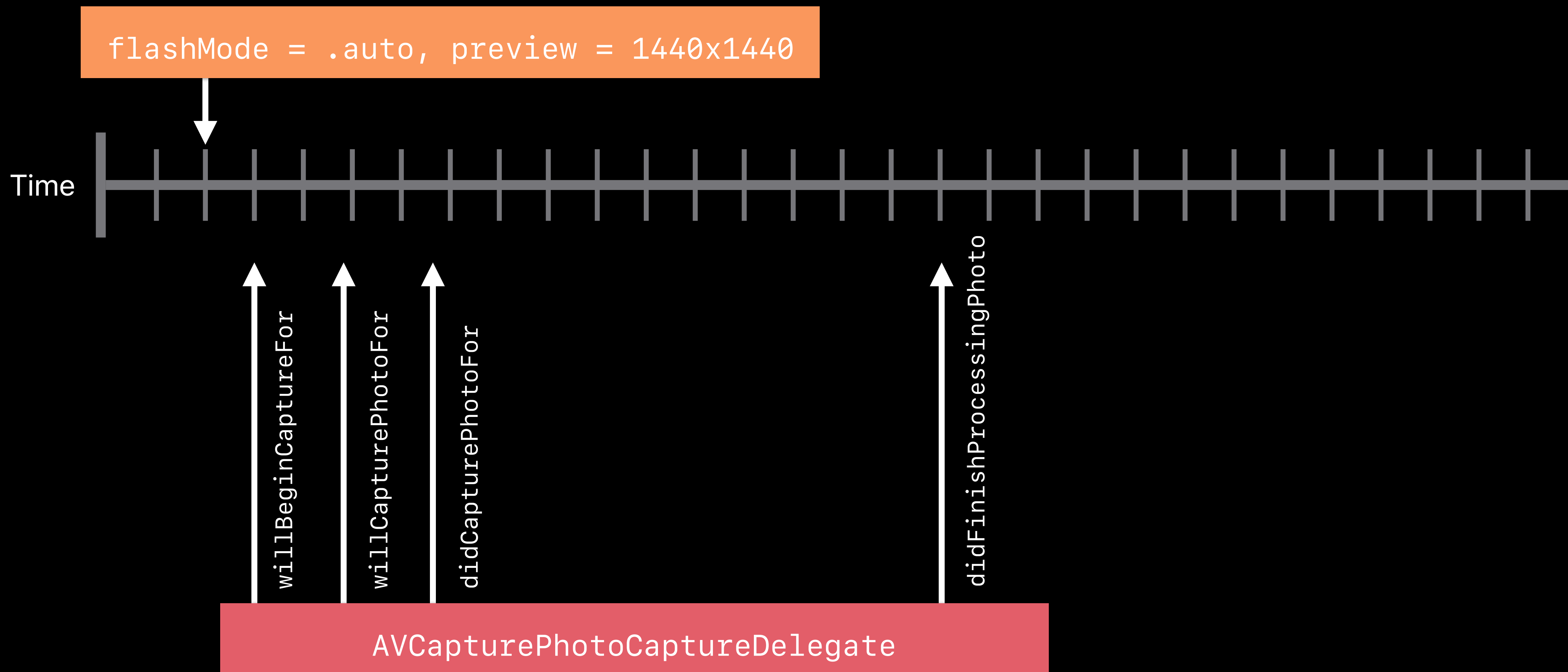
AVCapturePhotoOutput Usage



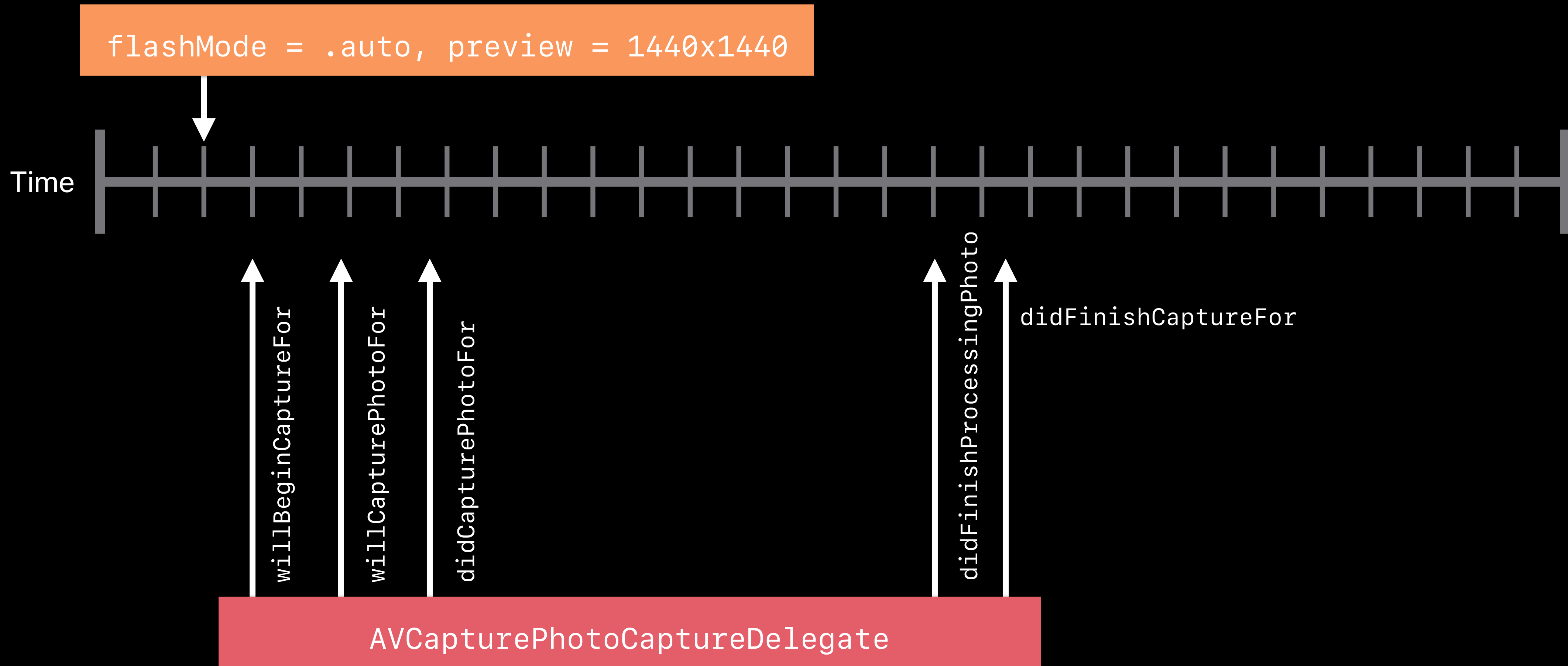
AVCapturePhotoOutput Usage



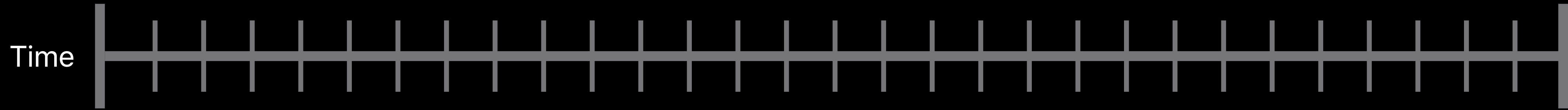
AVCapturePhotoOutput Usage



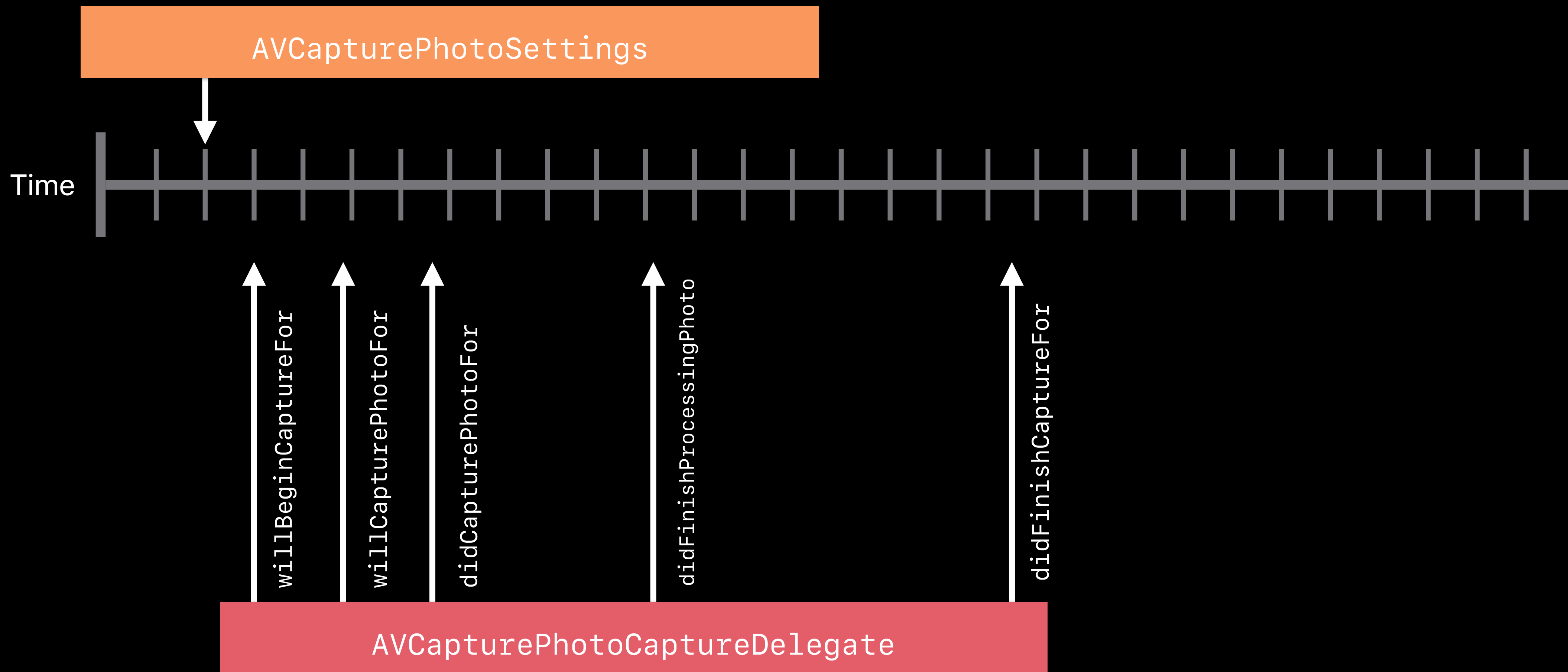
AVCapturePhotoOutput Usage



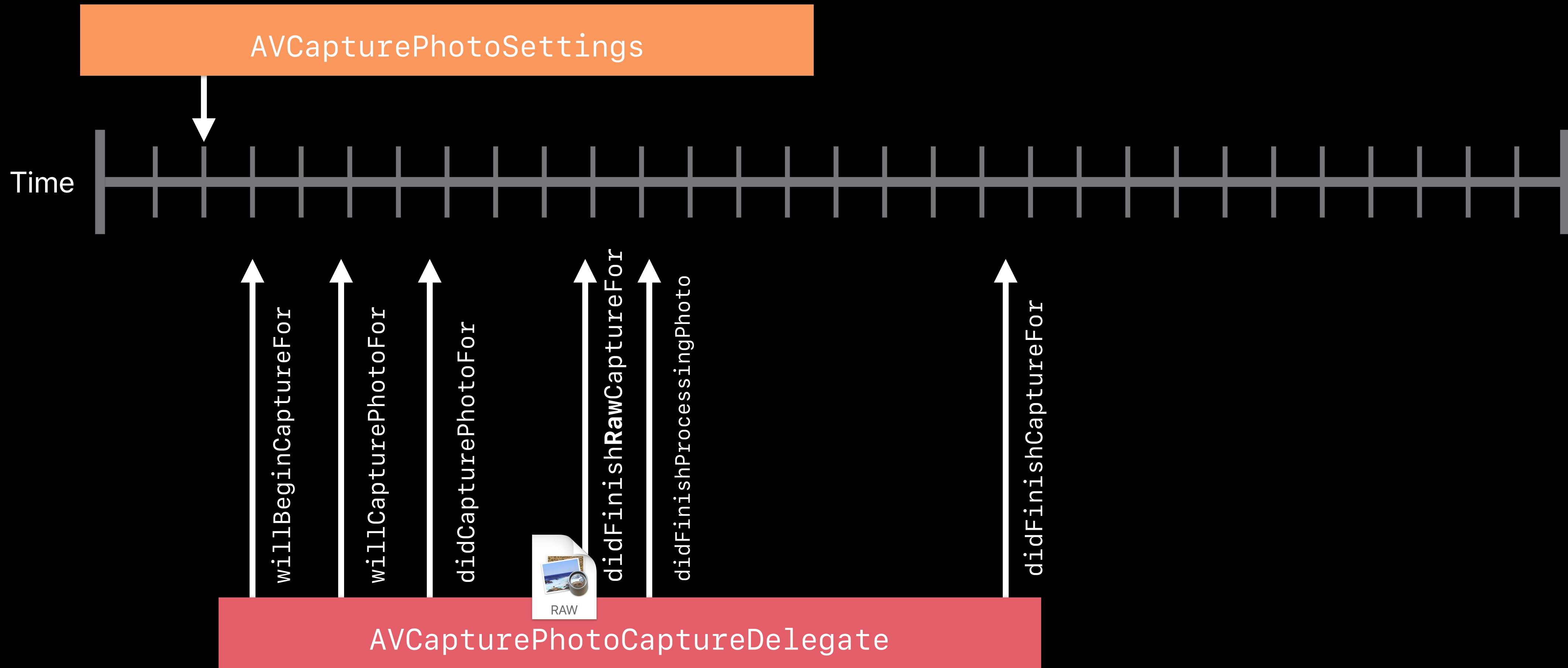
AVCapturePhotoOutput Usage



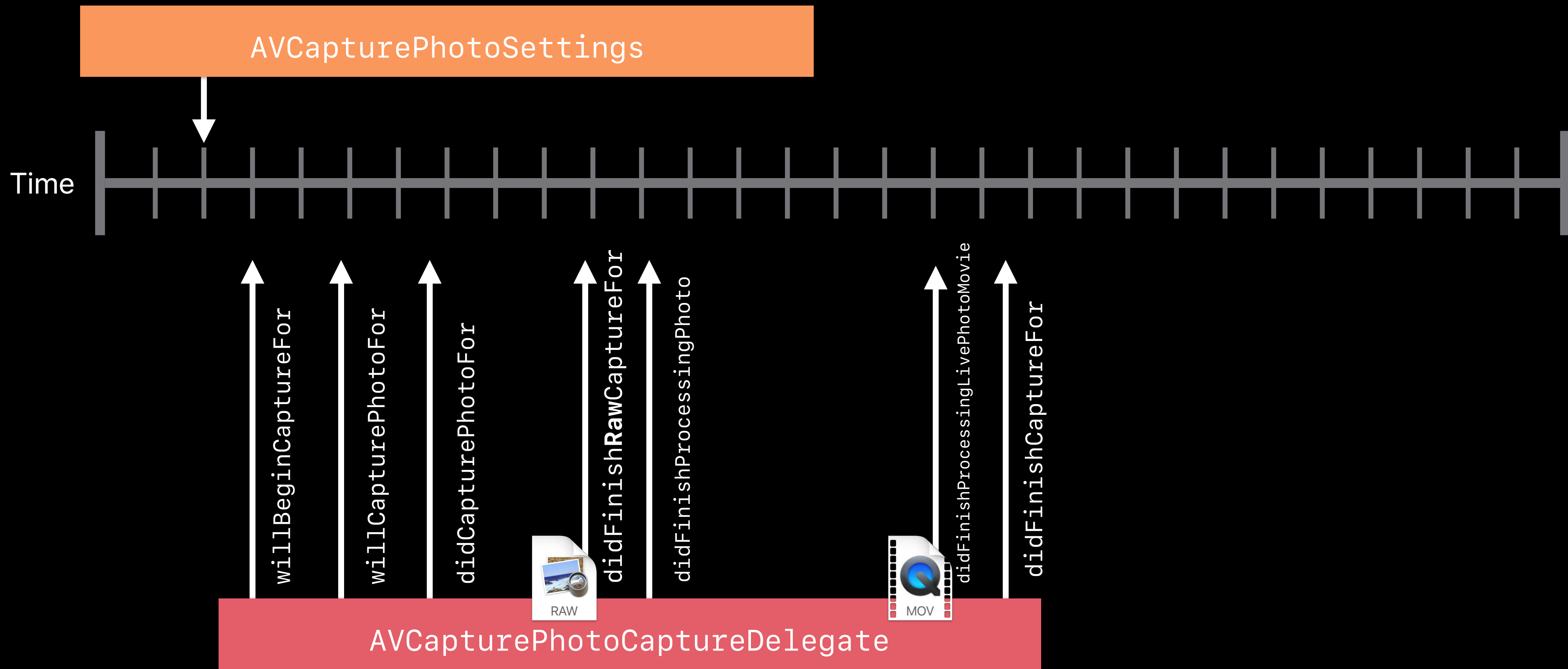
AVCapturePhotoOutput Usage



AVCapturePhotoOutput Usage



AVCapturePhotoOutput Usage



CMSampleBuffer vs. HEIF



CMSampleBuffer

CMSampleBuffer vs. HEIF

Sample buffer contains media data



CMSampleBuffer vs. HEIF

Sample buffer contains media data

HEIF contains a file structure



CMSampleBuffer vs. HEIF

Sample buffer contains media data

HEIF contains a file structure

HEVC video is not the same as HEIF HEVC



CMSampleBuffer



JPEG

NEW



AVCapturePhoto

NEW

Faster CMSampleBuffer replacement



AVCapturePhoto

NEW

Faster CMSampleBuffer replacement

100% immutable



AVCapturePhoto

NEW

Faster CMSampleBuffer replacement

100% immutable

Backed by containerized data



AVCapturePhoto



NEW

```
open class AVCapturePhoto : NSObject {  
  
    open var timestamp: CMTime { get }  
    open var isRawPhoto: Bool { get }  
    open var pixelBuffer: CVPixelBuffer? { get }  
  
    open var previewPixelBuffer: CVPixelBuffer? { get }  
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }  
  
    open var metadata: [String : Any] { get }  
    open var depthData: AVDepthData? { get }  
  
}
```



NEW

```
open class AVCapturePhoto : NSObject {
```

```
    open var timestamp: CMTime { get }
```

```
    open var isRawPhoto: Bool { get }
```

```
    open var pixelBuffer: CVPixelBuffer? { get }
```

```
    open var previewPixelBuffer: CVPixelBuffer? { get }
```

```
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }
```

```
    open var metadata: [String : Any] { get }
```

```
    open var depthData: AVDepthData? { get }
```

```
}
```

NEW

```
open class AVCapturePhoto : NSObject {  
  
    open var timestamp: CMTime { get }  
    open var isRawPhoto: Bool { get }  
    open var pixelBuffer: CVPixelBuffer? { get }  
  
    open var previewPixelBuffer: CVPixelBuffer? { get }  
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }  
  
    open var metadata: [String : Any] { get }  
    open var depthData: AVDepthData? { get }  
  
}
```



NEW

```
open class AVCapturePhoto : NSObject {  
  
    open var timestamp: CMTime { get }  
    open var isRawPhoto: Bool { get }  
    open var pixelBuffer: CVPixelBuffer? { get }  
  
    open var previewPixelBuffer: CVPixelBuffer? { get }  
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }  
  
    open var metadata: [String : Any] { get }  
    open var depthData: AVDepthData? { get }  
  
}
```



NEW

```
open class AVCapturePhoto : NSObject {  
  
    open var resolvedSettings: AVCaptureResolvedPhotoSettings { get }  
    open var photoCount: Int { get }  
  
    open var bracketSettings: AVCaptureBracketedStillImageSettings? { get }  
    open var sequenceCount: Int { get }  
    open var lensStabilizationStatus: AVCaptureDevice.LensStabilizationStatus { get }  
  
}
```



NEW

```
open class AVCapturePhoto : NSObject {  
  
    open var resolvedSettings: AVCaptureResolvedPhotoSettings { get }  
    open var photoCount: Int { get }  
  
    open var bracketSettings: AVCaptureBracketedStillImageSettings? { get }  
    open var sequenceCount: Int { get }  
    open var lensStabilizationStatus: AVCaptureDevice.LensStabilizationStatus { get }  
  
}
```




NEW

```
open class AVCapturePhoto : NSObject {  
  
    open var resolvedSettings: AVCaptureResolvedPhotoSettings { get }  
    open var photoCount: Int { get }  
  
    open var bracketSettings: AVCaptureBracketedStillImageSettings? { get }  
    open var sequenceCount: Int { get }  
    open var lensStabilizationStatus: AVCaptureDevice.LensStabilizationStatus { get }  
  
}
```

```
open class AVCapturePhoto : NSObject {  
  
    open func fileDataRepresentation() -> Data?  
  
    open func cgImageRepresentation() -> Unmanaged<CGImage>?  
    open func previewCGImageRepresentation() -> Unmanaged<CGImage>?  
  
}
```



NEW

```
open class AVCapturePhoto : NSObject {
```



NEW

```
    open func fileDataRepresentation() -> Data?
```

```
    open func cgImageRepresentation() -> Unmanaged<CGImage>?
```

```
    open func previewCGImageRepresentation() -> Unmanaged<CGImage>?
```

```
}
```



NEW

```
open class AVCapturePhoto : NSObject {  
  
    open func fileDataRepresentation() -> Data?  
  
    open func cgImageRepresentation() -> Unmanaged<CGImage>?  
    open func previewCGImageRepresentation() -> Unmanaged<CGImage>?  
  
}
```

```
func photoOutput(_ output: AVCapturePhotoOutput,  
                 didFinishProcessingPhoto photo: AVCapturePhoto,  
                 error: Error?)
```



```
func photoOutput(_ output: AVCapturePhotoOutput,  
    didFinishProcessingPhoto photoSampleBuffer: CMSampleBuffer?,  
    previewPhoto: CMSampleBuffer?,  
    resolvedSettings: AVCaptureResolvedPhotoSettings,  
    bracketSettings: AVCaptureBracketedStillImageSettings?,  
    error: Error?)
```



```
func photoOutput(_ output: AVCapturePhotoOutput,  
    didFinishProcessingRawPhoto rawSampleBuffer: CMSampleBuffer?,  
    previewPhoto: CMSampleBuffer?,  
    resolvedSettings: AVCaptureResolvedPhotoSettings,  
    bracketSettings: AVCaptureBracketedStillImageSettings?,  
    error: Error?)
```



iOS 10 AVCapturePhotoOutput Supported Formats

	Image Format
Compressed Formats	jpeg
Uncompressed Formats	420f/420v
	BGRA
RAW Formats	grb4/rgg4/bgg4/gbr4

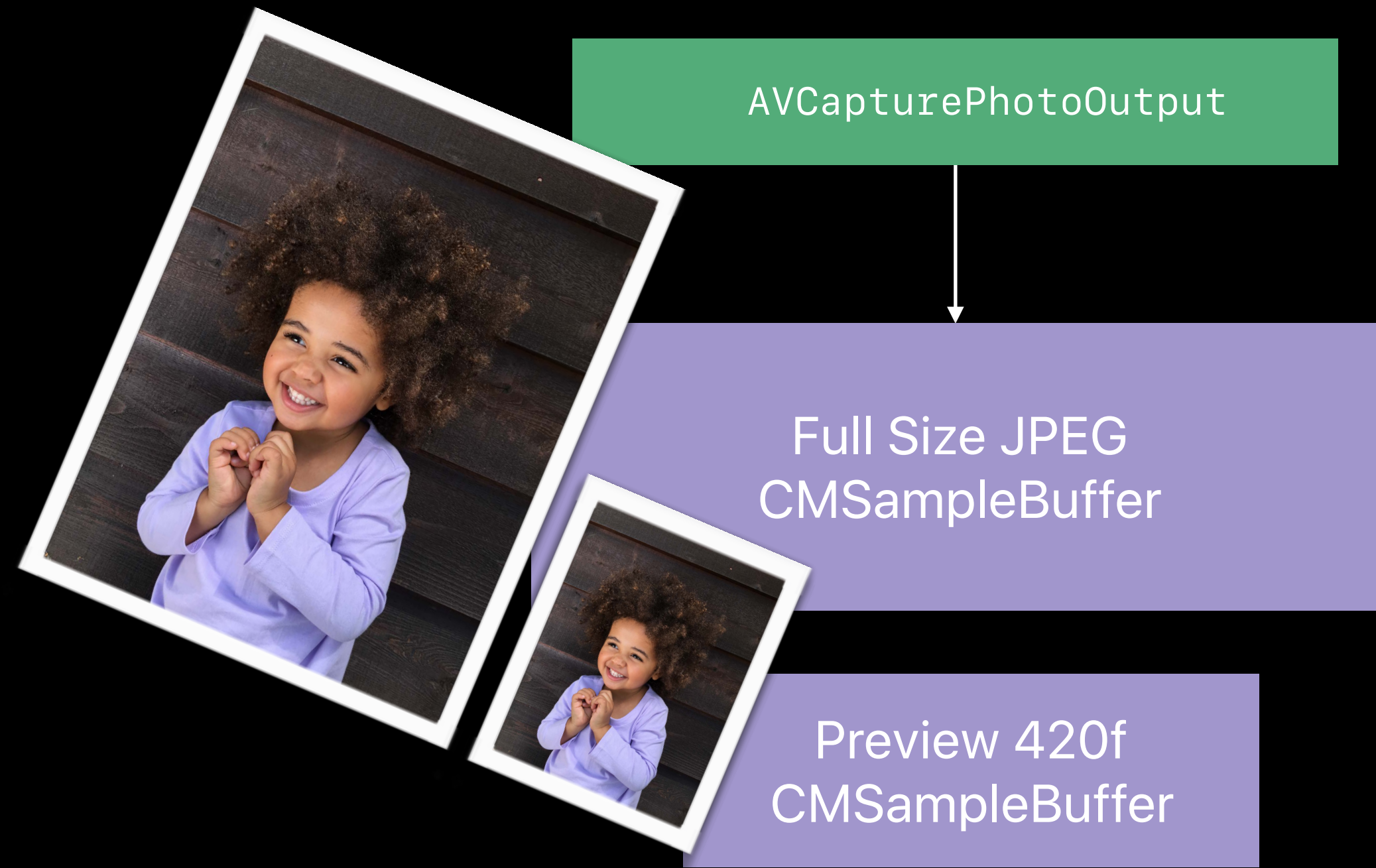
iOS 11 AVCapturePhotoOutput Supported Formats

	Image Format	File Container Format
Compressed Formats	hvc1	HEIC
	jpeg	JFIF
Uncompressed Formats	420f/420v	TIFF
	BGRA	
RAW Formats	grb4/rgg4/bgg4/gbr4	DNG

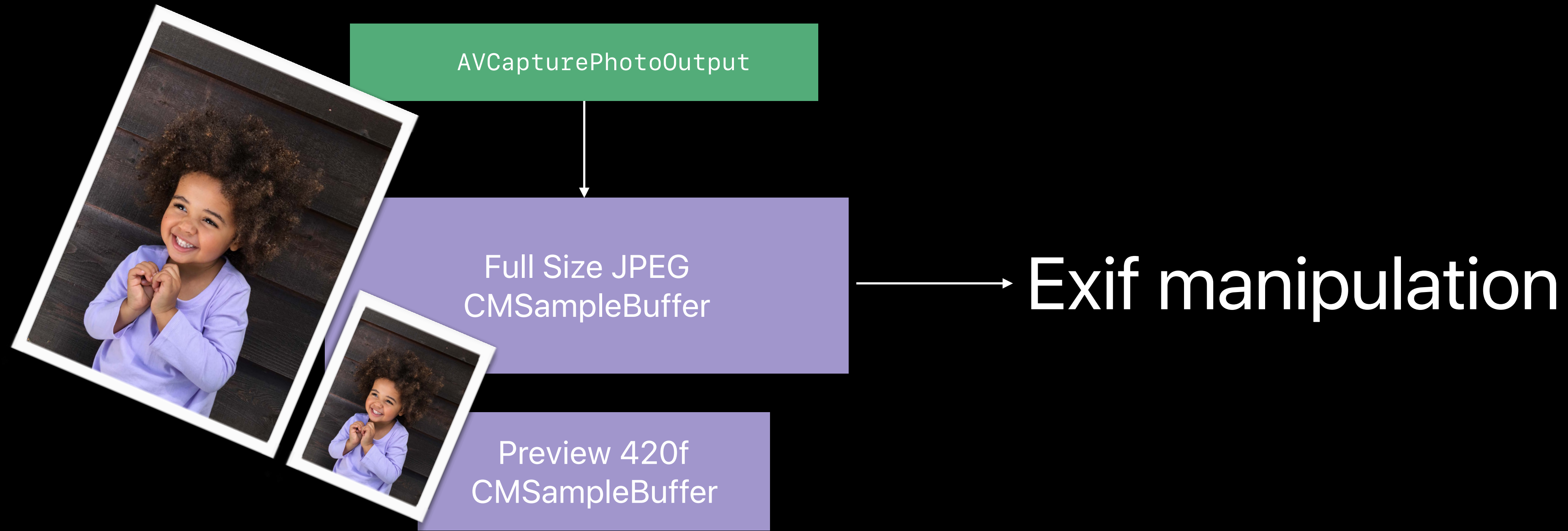
iOS 11 AVCapturePhotoOutput Supported Formats

	Image Format	File Container Format
Compressed Formats	hvc1	HEIC
	jpeg	JFIF
Uncompressed Formats	420f/420v	TIFF
	BGRA	
RAW Formats	grb4/rgg4/bgg4/gbr4	DNG

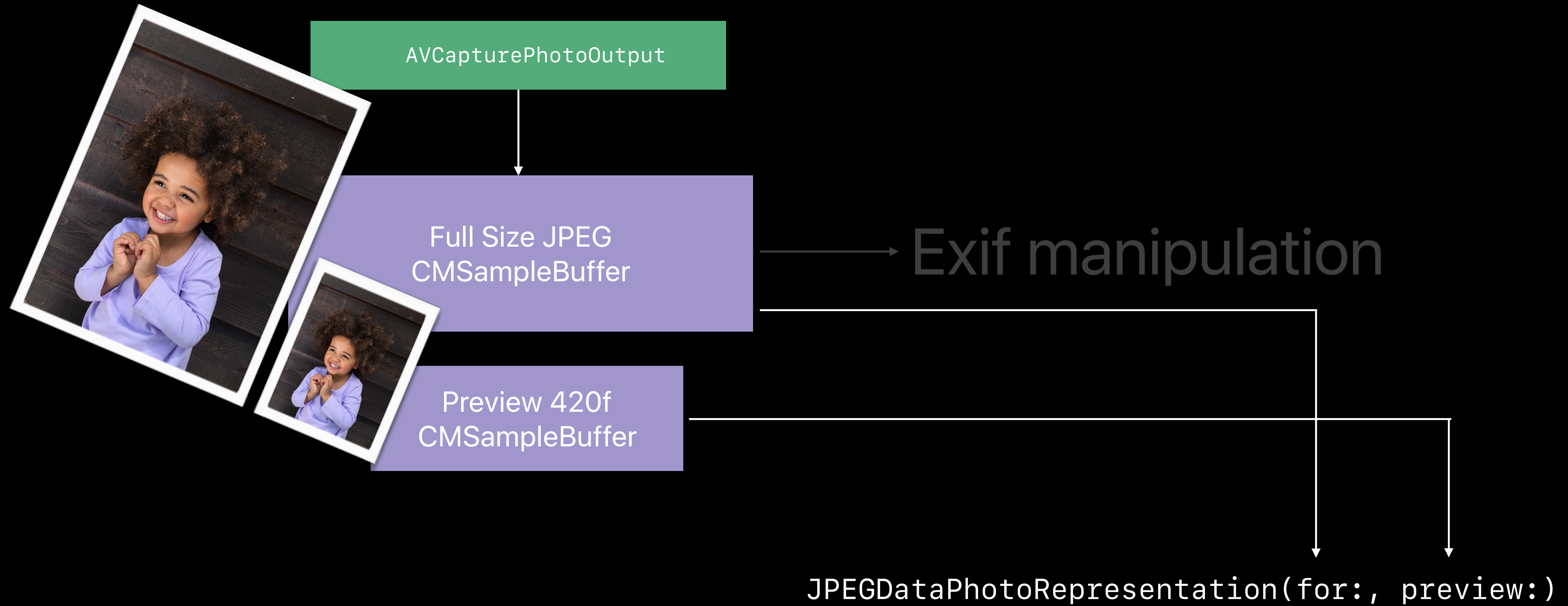
Old Way



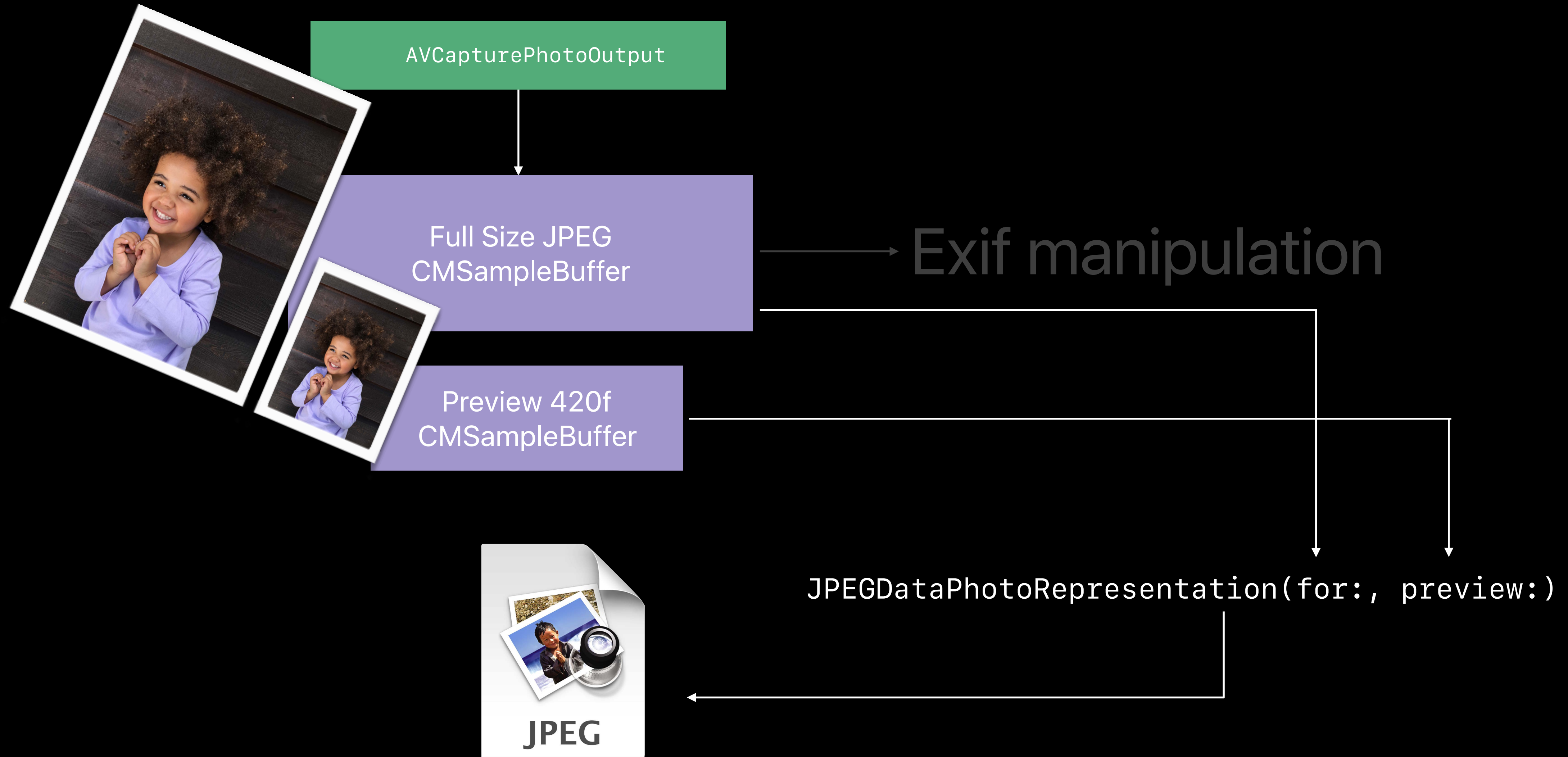
Old Way



Old Way



Old Way



New Way

New Way

AVCapturePhotoSettings

```
.format = [AVVideoCodecType: .hevc]  
.metadata = [...]  
.previewPhotoFormat = [1440x1440]  
.embeddedThumbnailPhotoFormat = [320x320]
```

New Way

AVCapturePhotoOutput

AVCapturePhotoSettings

```
.format = [AVVideoCodecType: .hevc]  
.metadata = [...]  
.previewPhotoFormat = [1440x1440]  
.embeddedThumbnailPhotoFormat = [320x320]
```


New Way

AVCapturePhotoOutput

AVCapturePhotoSettings

```
.format = [AVVideoCodecType: .hevc]  
.metadata = [...]  
.previewPhotoFormat = [1440x1440]  
.embeddedThumbnailPhotoFormat = [320x320]
```

AVCapturePhoto
(HEIC containerized)



New Way

AVCapturePhotoOutput

AVCapturePhotoSettings

```
.format = [AVVideoCodecType: .hevc]  
.metadata = [...]  
.previewPhotoFormat = [1440x1440]  
.embeddedThumbnailPhotoFormat = [320x320]
```



AVCapturePhoto
(HEIC containerized)

photo.fileDataRepresentation()

New Way

AVCapturePhotoOutput

AVCapturePhotoSettings

```
.format = [AVVideoCodecType: .hevc]  
.metadata = [...]  
.previewPhotoFormat = [1440x1440]  
.embeddedThumbnailPhotoFormat = [320x320]
```



AVCapturePhoto
(HEIC containerized)

`photo.fileDataRepresentation()`



Photos During Video Capture

HEVC/H.264 hardware resource contention

- Video is prioritized
- Photos are larger

Consider using JPEG for photos

HEVC and HEIF Bursts

HEVC encode takes longer than JPEG

Recommendation for burst captures is to use JPEG

A scenic landscape at sunrise or sunset. The sun is low on the horizon, casting a warm glow over a range of mountains. The sky is filled with soft, wispy clouds. In the foreground, a calm lake reflects the sky and the mountains. The overall mood is peaceful and serene.

A Compression Essay on WWDC

Brad Ford



World Wide Developer Conference

A scenic landscape at dawn or dusk. The sky is a mix of deep blue and soft orange, with wispy clouds. The sun is low on the horizon, creating a bright glow and a lens flare effect. The foreground is a calm body of water, likely a lake, which perfectly reflects the sky and the surrounding landscape. The background features a range of mountains and a dense forest of trees, all silhouetted against the light sky. The overall mood is peaceful and serene.

A scenic landscape at dawn or dusk. The sky is a mix of deep blue and soft orange, with wispy clouds. The sun is low on the horizon, creating a bright glow and a lens flare effect. The foreground is a calm body of water, likely a lake, which perfectly reflects the sky and the surrounding landscape. The background features silhouettes of trees and mountains. The overall mood is serene and peaceful.

World Wide Developer Conference

WWDC

A scenic landscape at dawn or dusk. The sky is a mix of deep blue and soft orange, with wispy clouds. The sun is low on the horizon, creating a lens flare effect. In the foreground, a calm lake reflects the sky and the surrounding landscape. The middle ground features a dense line of dark trees, and the background shows silhouettes of mountains.

World Wide Developer Conference

WWDC

Dub-Dub

A scenic landscape at dawn or dusk. The sky is a mix of deep blue and soft orange, with wispy clouds. In the foreground, a calm lake reflects the sky and the surrounding landscape. The middle ground is filled with dark silhouettes of trees and a line of mountains in the distance. The overall mood is serene and quiet.

World Wide Developer Conference

WWDC

Dub-Dub

Wuh-Duck

Summary

HEVC movies are up to 40% smaller than H.264

HEVC playback is supported everywhere on iOS 11 and macOS High Sierra

Opt in to create HEVC content using capture and export APIs

HEIC files are twice as small as JPEGs

HEIF decode is supported everywhere on iOS 11 and macOS High Sierra

Capture HEIC files using the new `AVCapturePhoto` interface

More Information

<https://developer.apple.com/wwdc17/511>

Related Sessions

High Efficiency Image File Format

WWDC 2017 [Video](#)

Introducing HEIF and HEVC

WWDC 2017

Depth and Capture

WWDC 2017

Image Editing with Depth

WWDC 2017

Labs

HEIF and HEVC Lab

Technology Lab F

Fri 12:00PM-2:00PM

Photos Depth & Capture Lab

Technology Lab F

Fri 2:00PM-4:00PM

