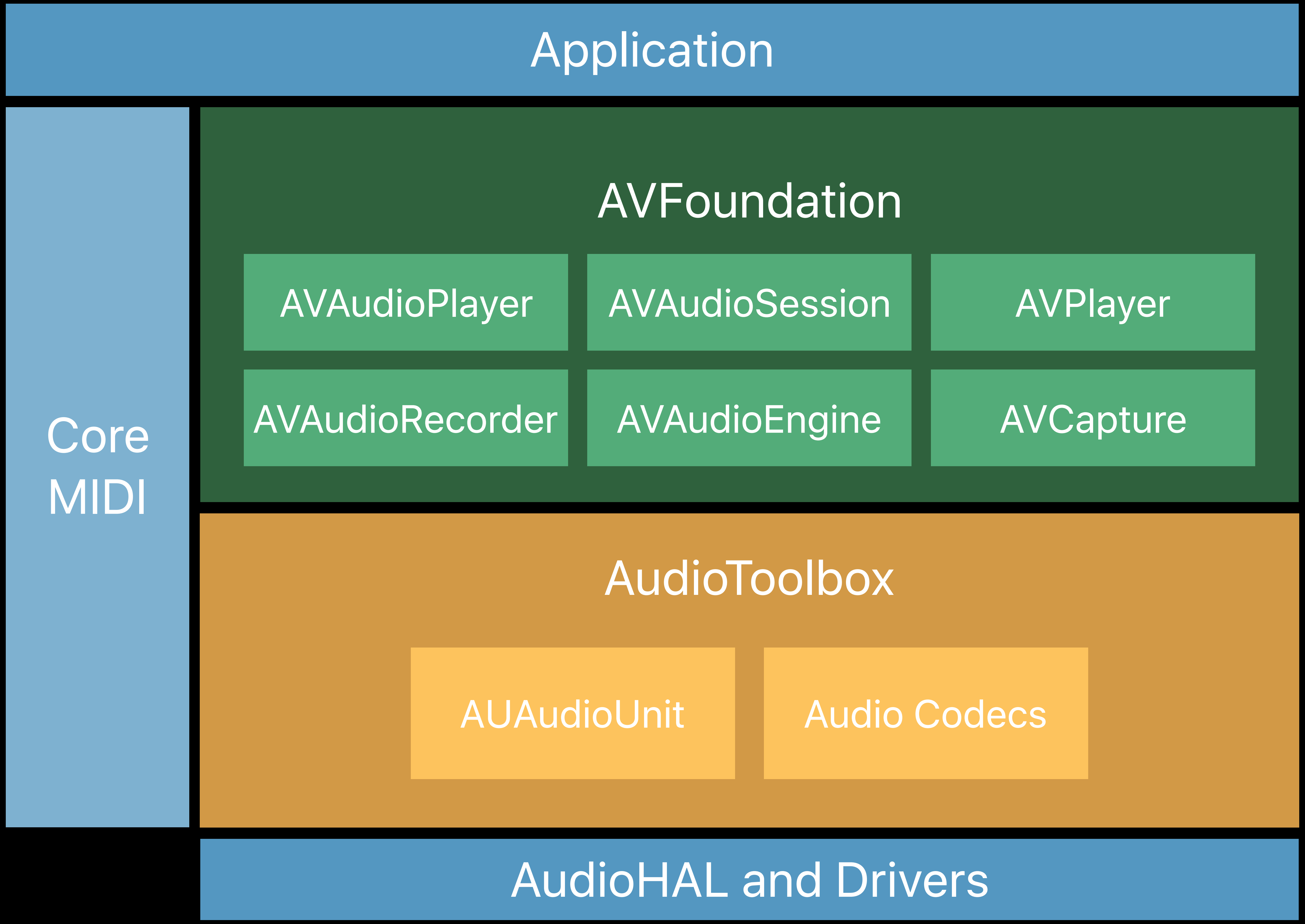
What’s New in Audio

Session 501

Akshatha Nagesh, AudioEngine-eer
Béla Balázs, Audio Artisan
Torrey Holbrook Walker, Audio/MIDI Black Ops

Audio Stack


Application
Core MIDI
AVFoundation
AVAudioPlayer
AVAudioSession
AVPlayer
AVAudioRecorder
AVAudioEngine
AVCapture
AudioToolbox
AUAudioUnit
Audio Codecs
AudioHAL and Drivers

AVAudioEngine

AVAudioSession

watchOS

AUAudioUnit

Other Enhancements

Inter-Device Audio Mode (IDAM)

AVAudioEngine

Recap

Powerful, feature-rich, Objective-C / Swift API set

Simplifies realtime audio, easier to use

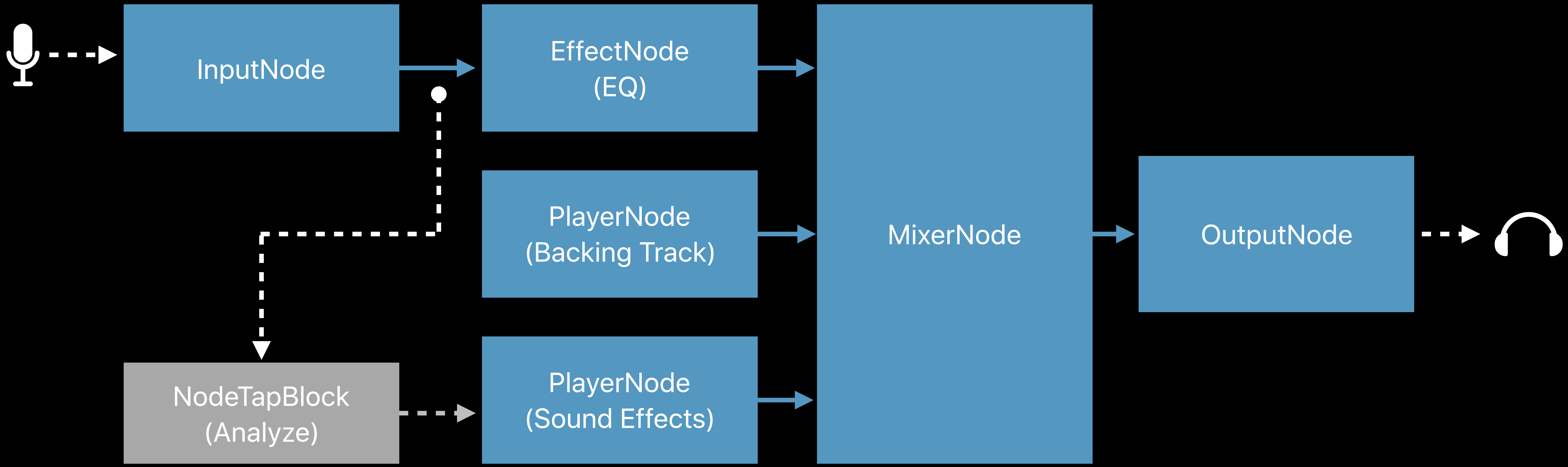
Supports

- Playback, recording, processing, mixing
- 3D spatialization

AVAudioEngine in Practice	WWDC14
What’s New in Core Audio	WWDC15

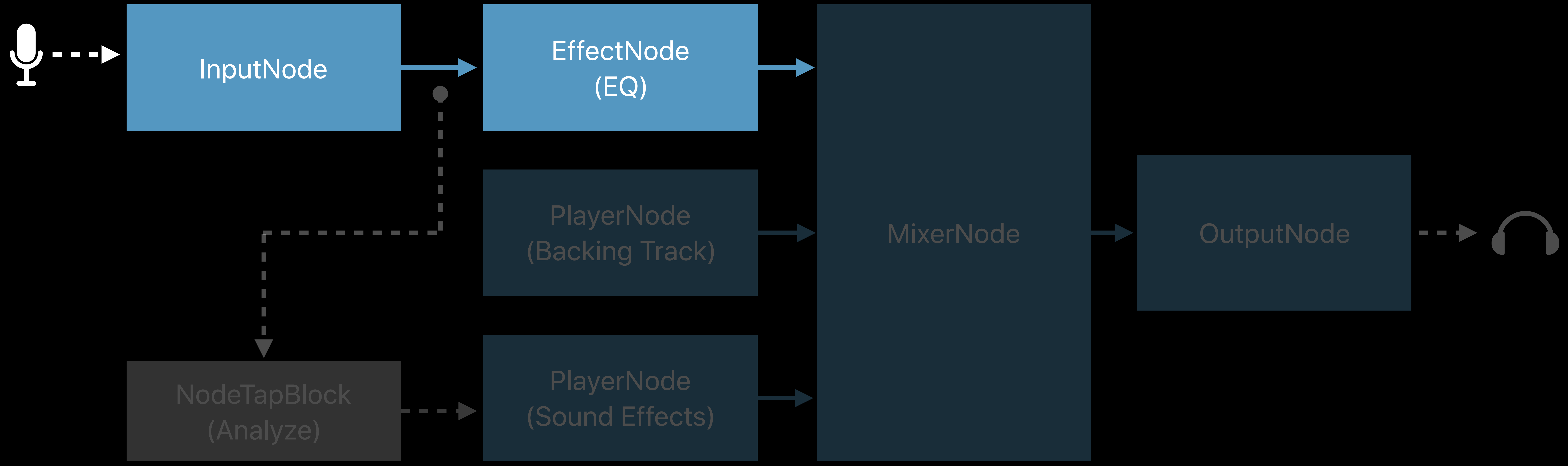
Sample Engine Setup

Karaoke



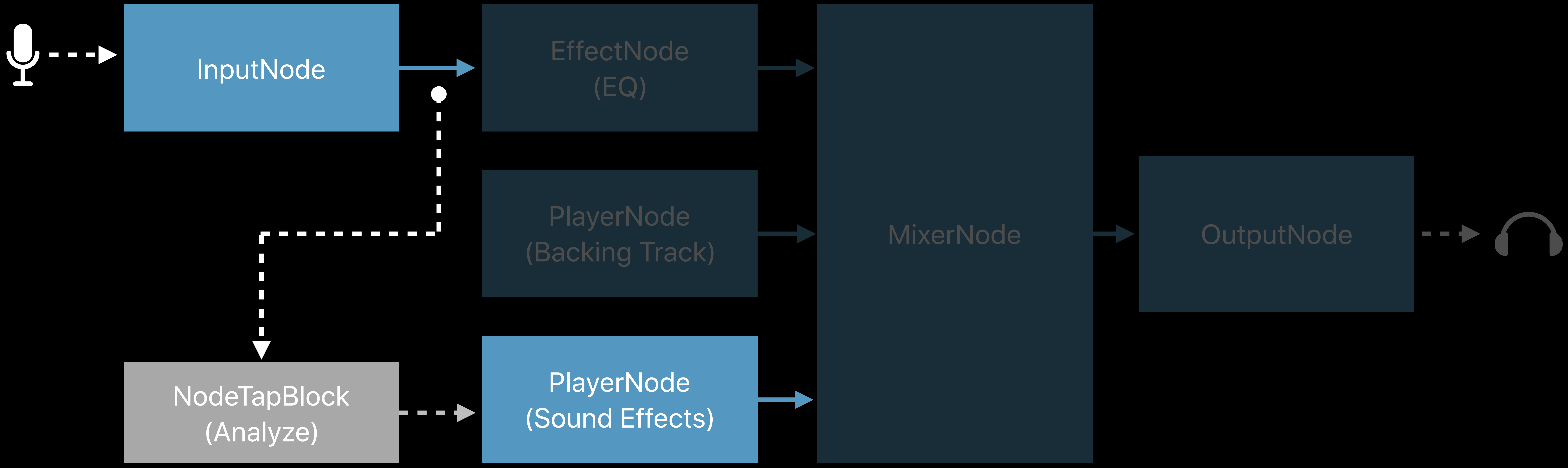
Sample Engine Setup

Karaoke



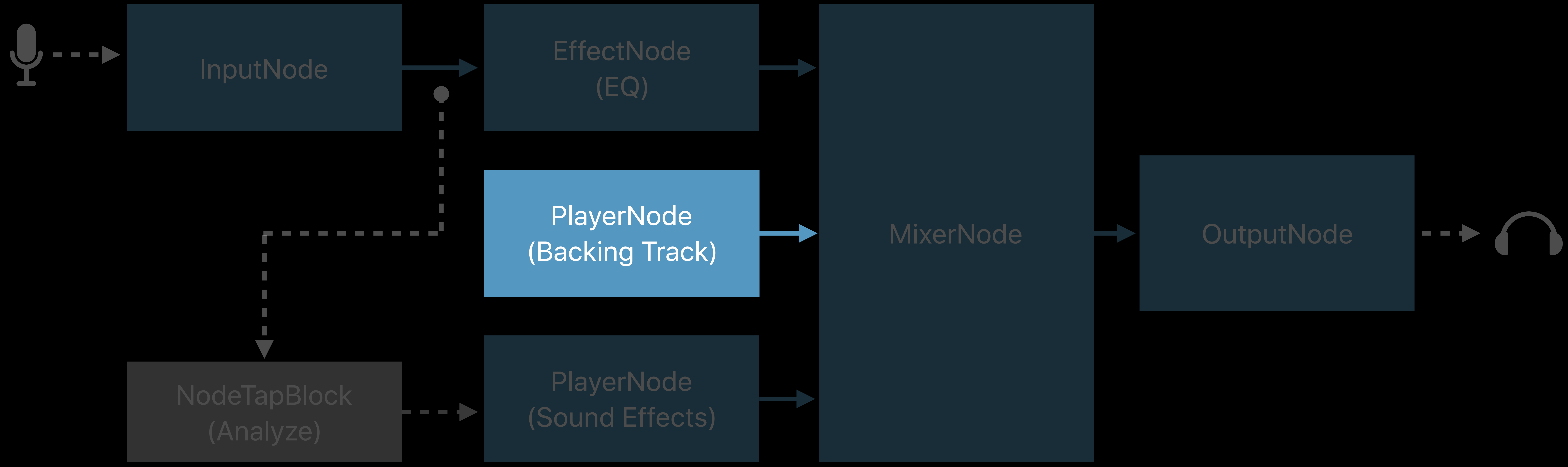
Sample Engine Setup

Karaoke



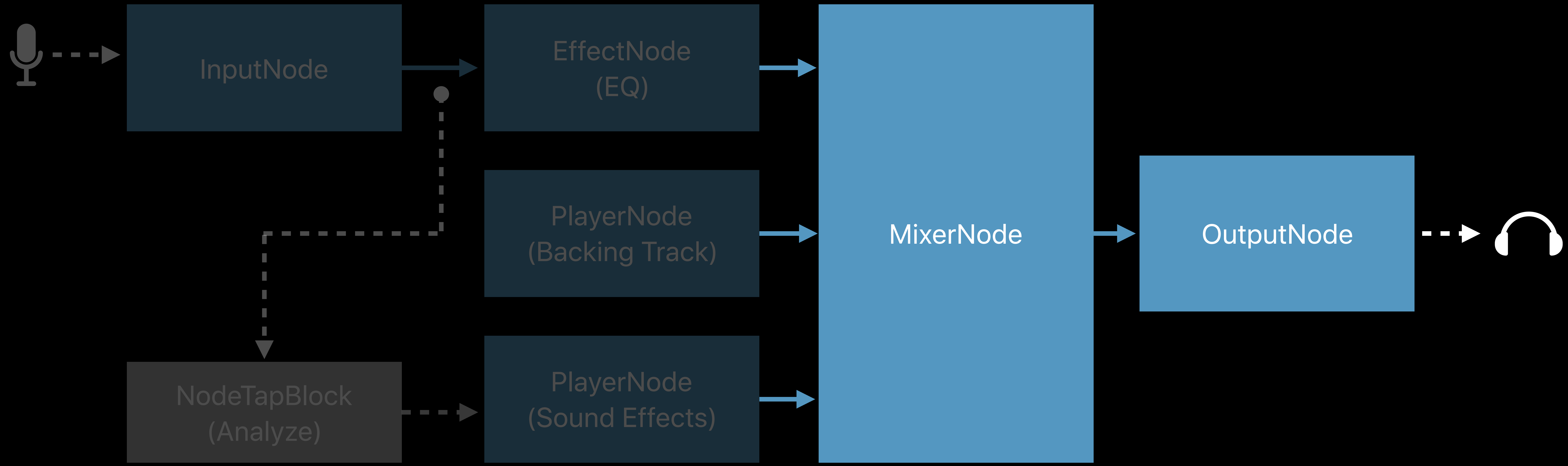
Sample Engine Setup

Karaoke



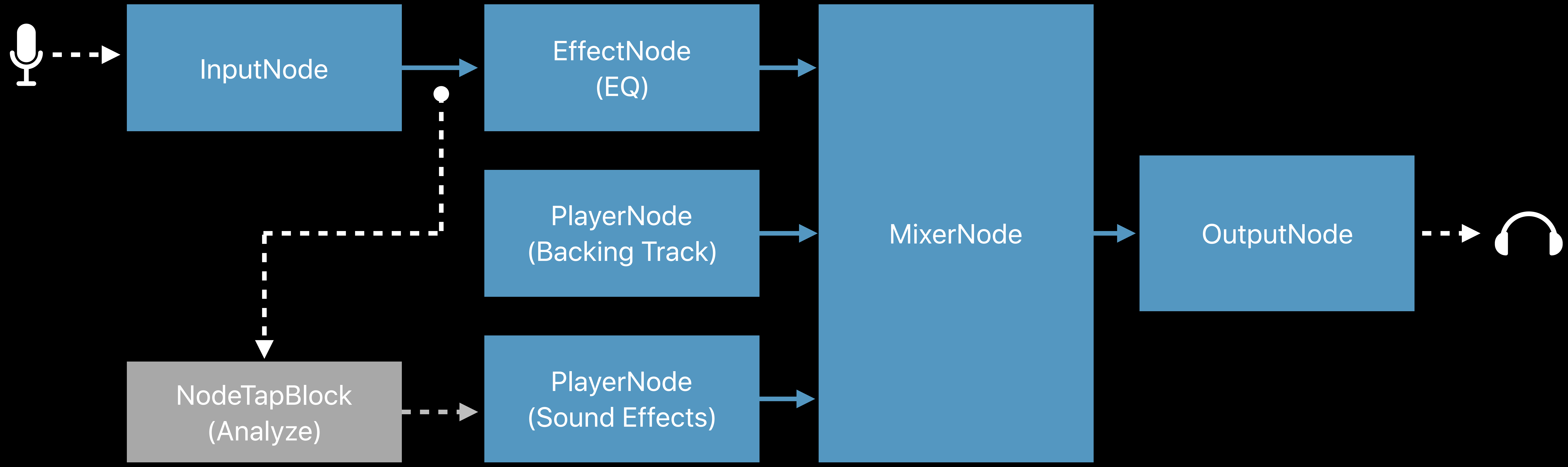
Sample Engine Setup

Karaoke



Sample Engine Setup

Karaoke



What’s New

NEW

AVAudioEngine

- Manual rendering
- Auto shutdown

AVAudioPlayerNode

- Completion callbacks

What's New

NEW

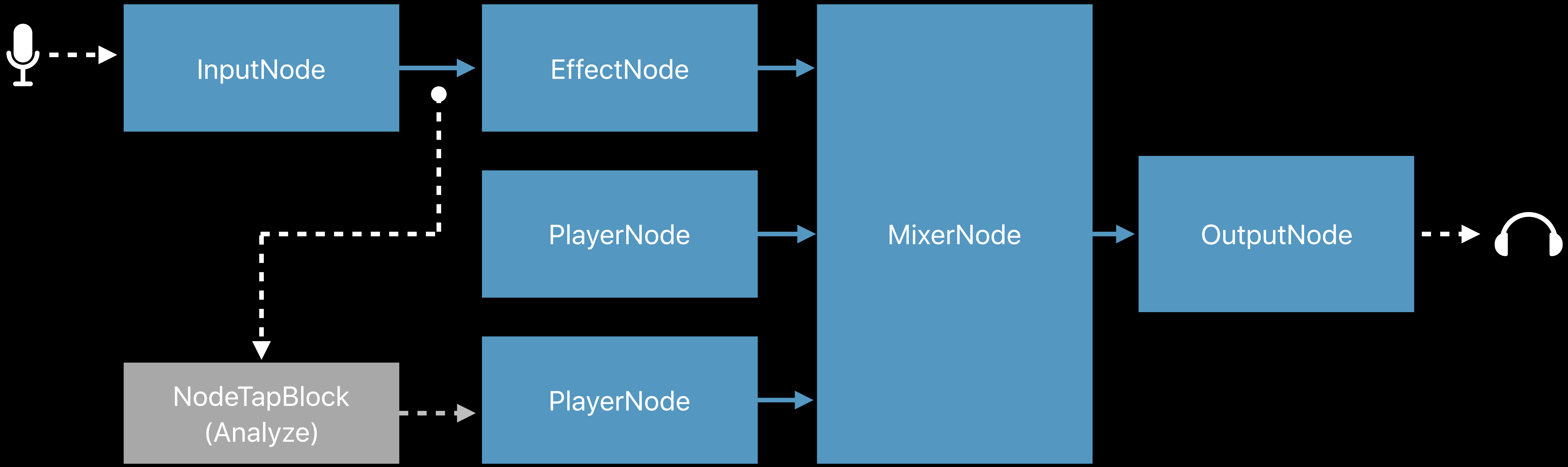
AVAudioEngine

- Manual rendering
- Auto shutdown

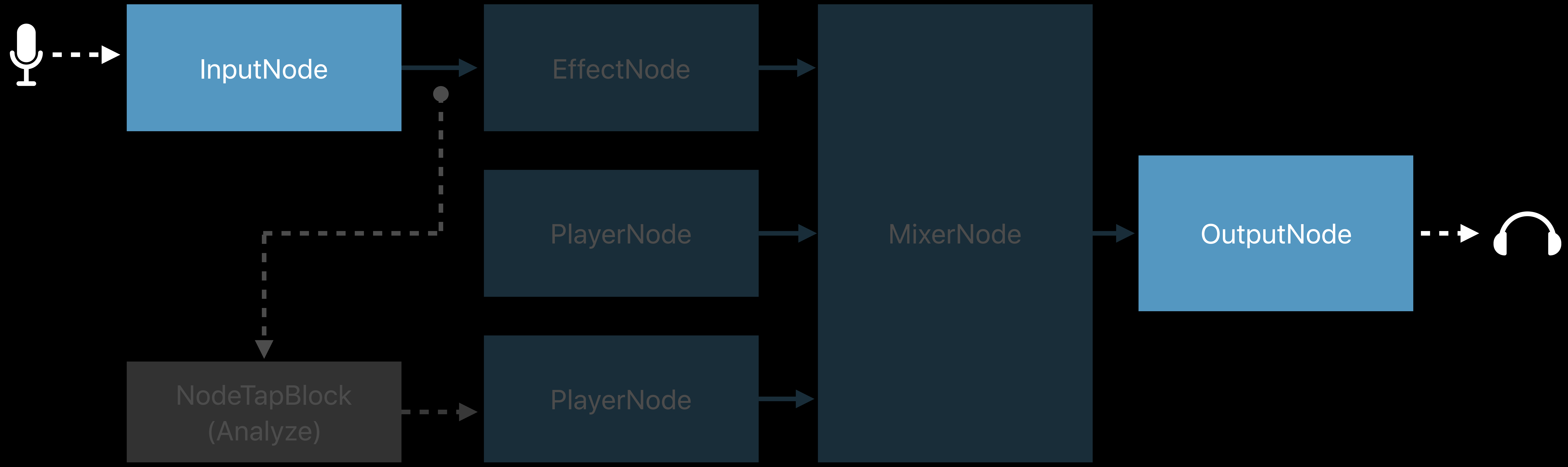
AVAudioPlayerNode

- Completion callbacks

Sample Engine Setup



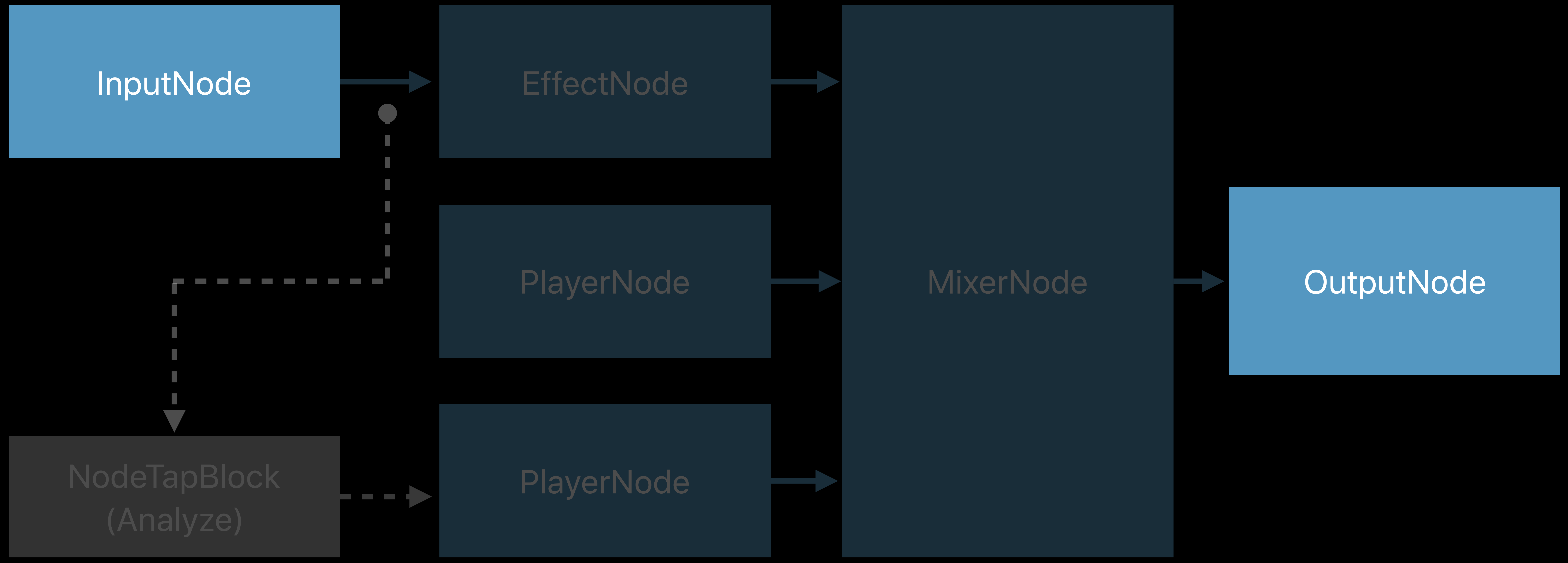
Sample Engine Setup



Sample Engine Setup

Manual rendering

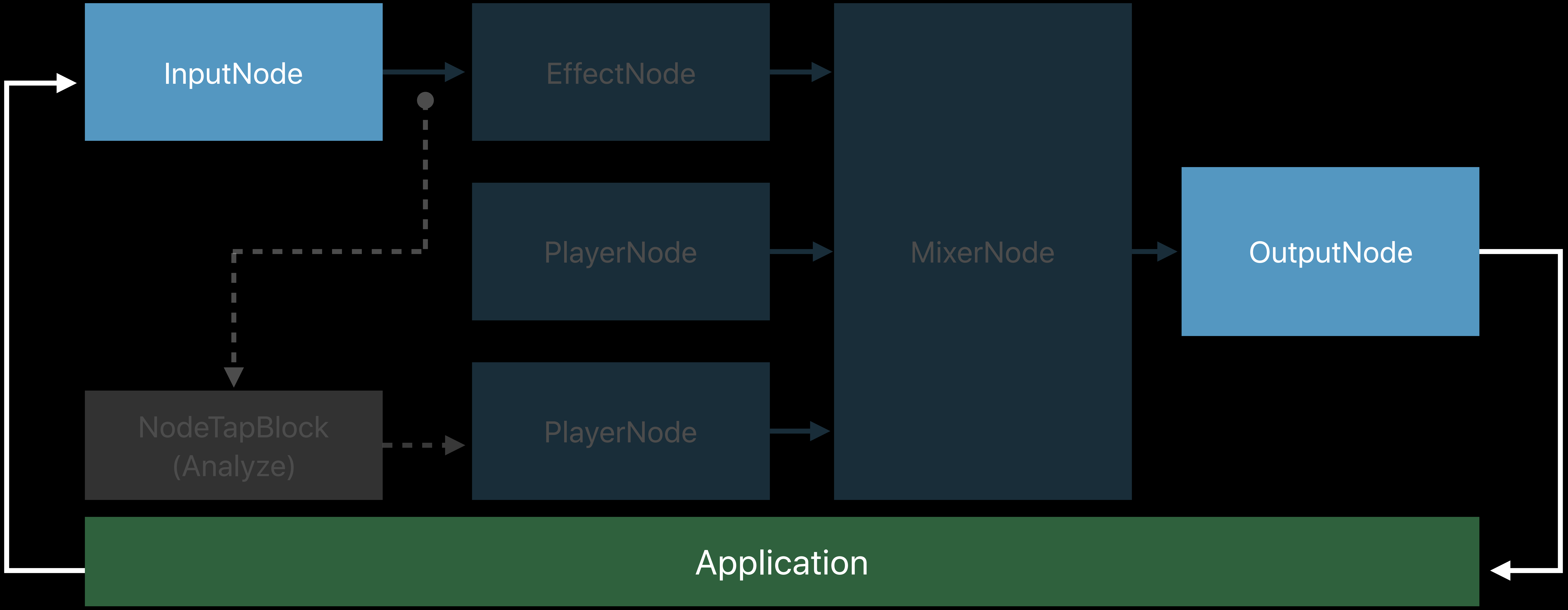
NEW



Sample Engine Setup

Manual rendering

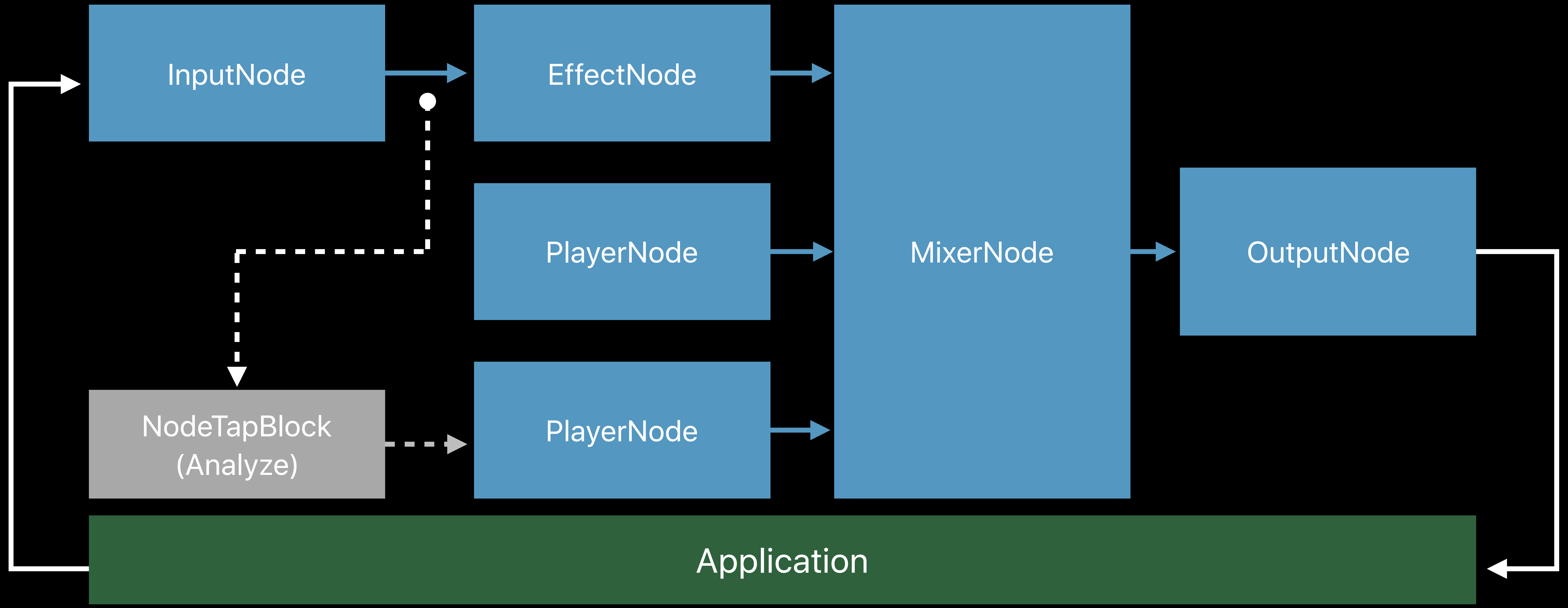
NEW



Sample Engine Setup

Manual rendering

NEW



Manual Rendering

NEW

Engine is not connected to any audio device

Renders in response to requests from the client

Modes

- Offline
- Realtime

Offline Manual Rendering

NEW

Engine and nodes operate under no deadlines or realtime constraints

A node may choose to:

- Use a more expensive signal processing algorithm
- Block on render thread for more data if needed
    - For example, player node may wait until its worker thread reads the data from disk

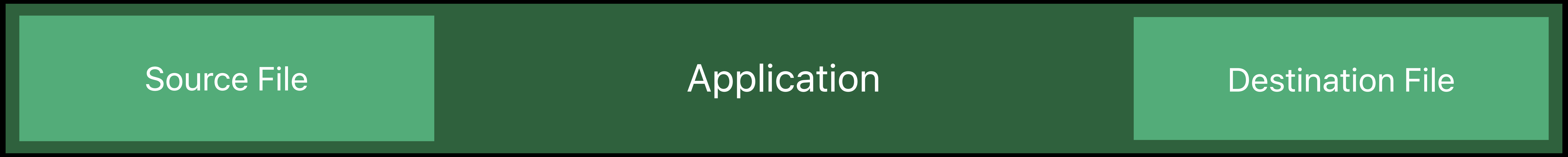
Offline Manual Rendering
Example

NEW

Offline Manual Rendering

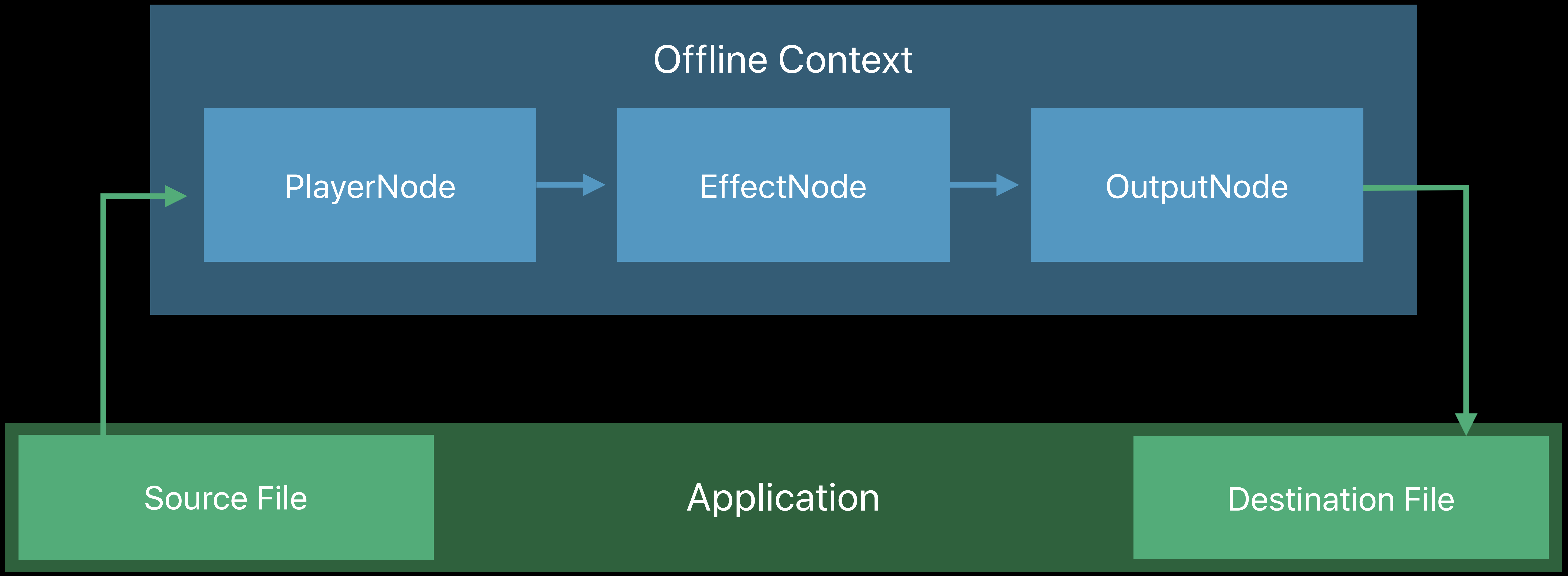
Example

NEW



Offline Manual Rendering
Example

NEW



Offline Manual Rendering

Applications

NEW

Post-processing of audio files, for example, apply reverb, effects etc.

Mixing of audio files

Offline audio processing using CPU intensive (higher quality) algorithms

Tuning, debugging or testing the engine setup

Demo

AVAudioEngine - Offline Manual Rendering

Realtime Manual Rendering

NEW

The engine and nodes:

- Operate under realtime constraints
- Do not make any blocking calls like blocking on a mutex, calling libdispatch etc., on the render thread
  - A node may drop the data if it is not ready to be rendered in time

Realtime Manual Rendering
Applications

NEW

Processing audio in an AUAudioUnit’s internalRenderBlock

Processing audio data in a movie/video during streaming/playback

Realtime Manual Rendering

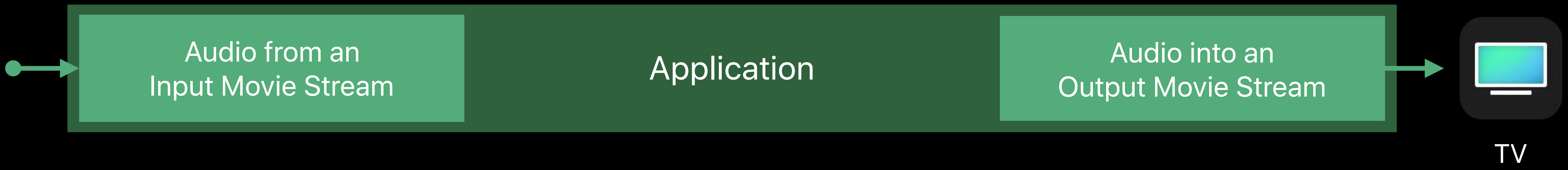
Example

NEW

Realtime Manual Rendering

Example

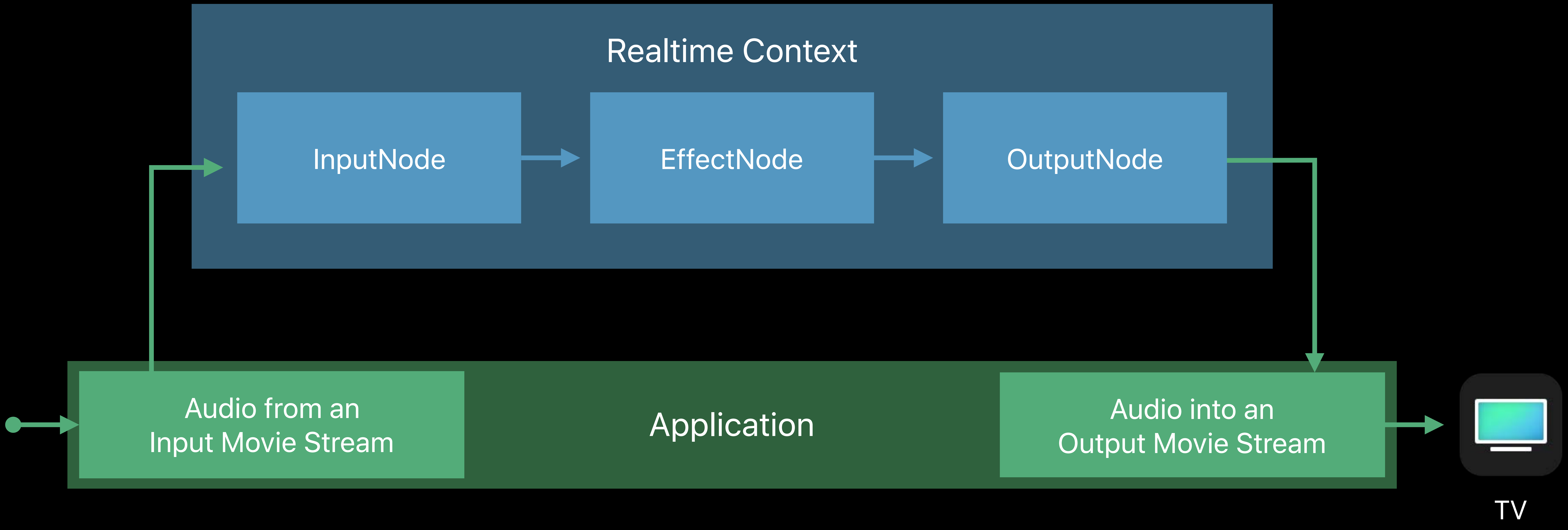
NEW


Audio from an Input Movie Stream
Application
Audio into an Output Movie Stream
TV

Realtime Manual Rendering

Example

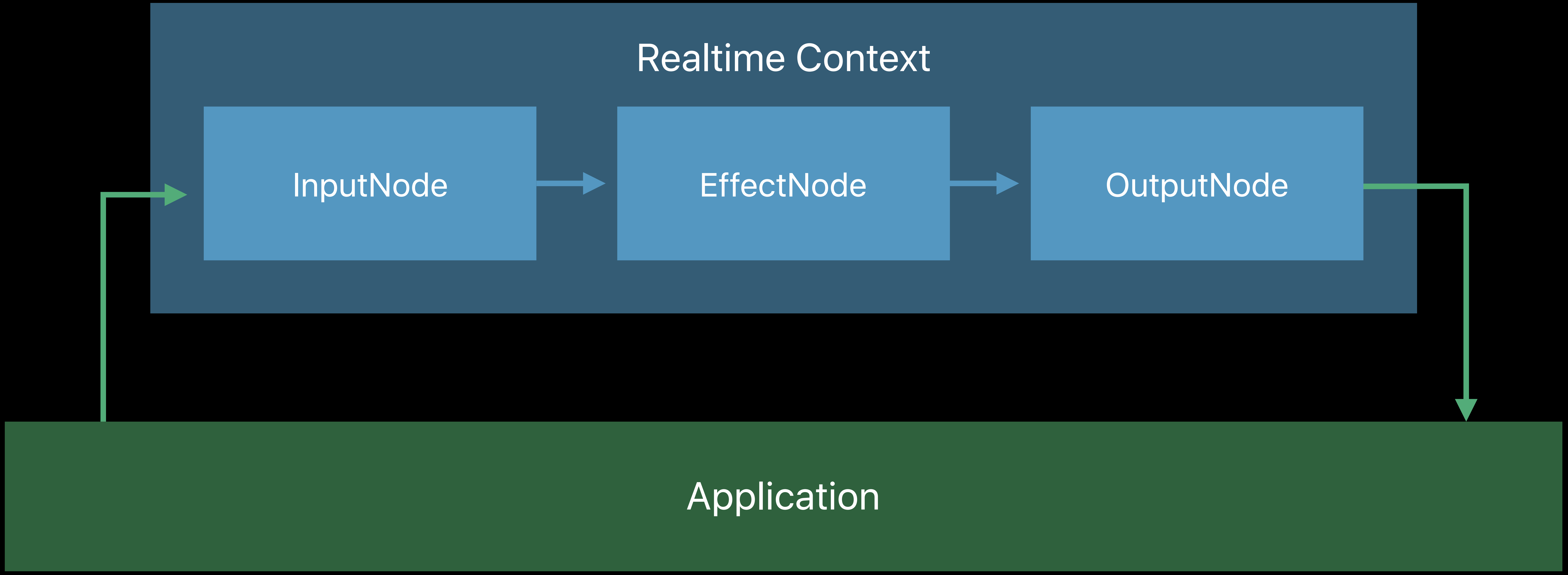
NEW


Realtime Context
InputNode
EffectNode
OutputNode
Audio from an Input Movie Stream
Application
Audio into an Output Movie Stream
TV

Realtime Manual Rendering

Code example

NEW



NEW

```swift
//Realtime Manual Rendering, code example

do {
   let engine = AVAudioEngine() // by default engine will render to/from the audio device
   // make connections, e.g. inputNode -> effectNode -> outputNode

   // switch to manual rendering mode
   engine.stop()
   try engine.enableManualRenderingMode(.realtime, format: outputPCMFormat,
               maximumFrameCount: frameCount) // e.g. 1024 @ 48 kHz = 21.33 ms

   let renderBlock = engine.manualRenderingBlock // cache the render block
```

NEW

```
//Realtime Manual Rendering, code example

do {
    let engine = AVAudioEngine() // by default engine will render to/from the audio device
    // make connections, e.g. inputNode -> effectNode -> outputNode

    // switch to manual rendering mode
    engine.stop()
    try engine.enableManualRenderingMode(.realtime, format: outputPCMFormat,
                 maximumFrameCount: frameCount) // e.g. 1024 @ 48 kHz = 21.33 ms

    let renderBlock = engine.manualRenderingBlock // cache the render block
```

NEW

```swift
//Realtime Manual Rendering, code example

do {
   let engine = AVAudioEngine() // by default engine will render to/from the audio device
   // make connections, e.g. inputNode -> effectNode -> outputNode

   // switch to manual rendering mode
   engine.stop()
   try engine.enableManualRenderingMode(.realtime, format: outputPCMFormat,
               maximumFrameCount: frameCount) // e.g. 1024 @ 48 kHz = 21.33 ms

   let renderBlock = engine.manualRenderingBlock // cache the render block
```

NEW

```swift
//Realtime Manual Rendering, code example

do {
    let engine = AVAudioEngine() // by default engine will render to/from the audio device
    // make connections, e.g. inputNode -> effectNode -> outputNode

    // switch to manual rendering mode
    engine.stop()
    try engine.enableManualRenderingMode(.realtime, format: outputPCMFormat,
                maximumFrameCount: frameCount) // e.g. 1024 @ 48 kHz = 21.33 ms

    let renderBlock = engine.manualRenderingBlock // cache the render block
```

NEW

```
    // set the block to provide input data to engine
    engine.inputNode.setManualRenderingInputPCMFormat(inputPCMFormat) {
       (inputFrameCount) -> UnsafePointer<AudioBufferList>? in
           guard haveData else { return nil }

           // fill and return the input audio buffer list
           return inputBufferList
    })
    // create output buffer, cache the buffer list
    let buffer = AVAudioPCMBuffer(pcmFormat: outputPCMFormat,
                                  frameCapacity: engine.manualRenderingMaximumFrameCount)!
    buffer.frameLength = buffer.frameCapacity
    let outputBufferList = buffer.mutableAudioBufferList
    try engine.start()
} catch { // handle errors }
```

NEW

```
    // set the block to provide input data to engine
    engine.inputNode.setManualRenderingInputPCMFormat(inputPCMFormat) {
        (inputFrameCount) -> UnsafePointer<AudioBufferList>? in
            guard haveData else { return nil }

            // fill and return the input audio buffer list
            return inputBufferList
    })
    // create output buffer, cache the buffer list
    let buffer = AVAudioPCMBuffer(pcmFormat: outputPCMFormat,
                                  frameCapacity: engine.manualRenderingMaximumFrameCount)!
    buffer.frameLength = buffer.frameCapacity
    let outputBufferList = buffer.mutableAudioBufferList
    try engine.start()
} catch { // handle errors }
```

NEW

```
    // set the block to provide input data to engine
    engine.inputNode.setManualRenderingInputPCMFormat(inputPCMFormat) {
        (inputFrameCount) -> UnsafePointer<AudioBufferList>? in
            guard haveData else { return nil }

            // fill and return the input audio buffer list
            return inputBufferList
    })
    // create output buffer, cache the buffer list
    let buffer = AVAudioPCMBuffer(pcmFormat: outputPCMFormat,
                                  frameCapacity: engine.manualRenderingMaximumFrameCount)!
    buffer.frameLength = buffer.frameCapacity
    let outputBufferList = buffer.mutableAudioBufferList
    try engine.start()
} catch { // handle errors }
```

NEW

```
    // set the block to provide input data to engine
    engine.inputNode.setManualRenderingInputPCMFormat(inputPCMFormat) {
        (inputFrameCount) -> UnsafePointer<AudioBufferList>? in
            guard haveData else { return nil }

            // fill and return the input audio buffer list
            return inputBufferList
    })
    // create output buffer, cache the buffer list
    let buffer = AVAudioPCMBuffer(pcmFormat: outputPCMFormat,
                                  frameCapacity: engine.manualRenderingMaximumFrameCount)!
    buffer.frameLength = buffer.frameCapacity
    let outputBufferList = buffer.mutableAudioBufferList
    try engine.start()
} catch { // handle errors }
```

NEW

```
    // set the block to provide input data to engine
    engine.inputNode.setManualRenderingInputPCMFormat(inputPCMFormat) {
        (inputFrameCount) -> UnsafePointer<AudioBufferList>? in
            guard haveData else { return nil }

            // fill and return the input audio buffer list
            return inputBufferList
    })
    // create output buffer, cache the buffer list
    let buffer = AVAudioPCMBuffer(pcmFormat: outputPCMFormat,
                                  frameCapacity: engine.manualRenderingMaximumFrameCount)!
    buffer.frameLength = buffer.frameCapacity
    let outputBufferList = buffer.mutableAudioBufferList
    try engine.start()
} catch { // handle errors }
```

NEW

```
// to render from realtime context
OSStatus outputError = noErr;
const auto status = renderBlock(framesToRender, outputBufferList, &outputError);
switch (status) {
   case AVAudioEngineManualRenderingStatusSuccess:
      handleProcessedOutput(outputBufferList); // data rendered successfully
      break;

   case AVAudioEngineManualRenderingStatusInsufficientDataFromInputNode:
      handleProcessedOutput(outputBufferList); // input node did not provide data,
                                               // but other sources may have rendered
      break;
   ..
   default:
      break;
}
```

NEW

```
// to render from realtime context
OSStatus outputError = noErr;
const auto status = renderBlock(framesToRender, outputBufferList, &outputError);
switch (status) {
   case AVAudioEngineManualRenderingStatusSuccess:
      handleProcessedOutput(outputBufferList); // data rendered successfully
      break;

   case AVAudioEngineManualRenderingStatusInsufficientDataFromInputNode:
      handleProcessedOutput(outputBufferList); // input node did not provide data,
                                               // but other sources may have rendered
      break;
   ..
   default:
      break;
}
```

NEW

```
// to render from realtime context
OSStatus outputError = noErr;
const auto status = renderBlock(framesToRender, outputBufferList, &outputError);
switch (status) {
    case AVAudioEngineManualRenderingStatusSuccess:
        handleProcessedOutput(outputBufferList); // data rendered successfully
        break;

    case AVAudioEngineManualRenderingStatusInsufficientDataFromInputNode:
        handleProcessedOutput(outputBufferList); // input node did not provide data,
                                                 // but other sources may have rendered
        break;
    ..
    default:
        break;
}
```

```
// to render from realtime context
OSStatus outputError = noErr;
const auto status = renderBlock(framesToRender, outputBufferList, &outputError);
switch (status) {
   case AVAudioEngineManualRenderingStatusSuccess:
      handleProcessedOutput(outputBufferList); // data rendered successfully
      break;

   case AVAudioEngineManualRenderingStatusInsufficientDataFromInputNode:
      handleProcessedOutput(outputBufferList); // input node did not provide data,
                                               // but other sources may have rendered
      break;
   ..
   default:
      break;
}
```

NEW

Manual Rendering

Render calls

NEW

Offline

- Can use either ObjC/Swift render method or the block based render call

Realtime

- Must use the block based render call

What's New

NEW

AVAudioEngine

- Manual rendering
- Auto shutdown

AVAudioPlayerNode

- Completion callbacks

Auto Shutdown

NEW

Hardware is stopped if running idle for a certain duration, started dynamically when needed

Safety net for conserving power

Enforced behavior on watchOS, optional on other platforms

`isAutoShutdownEnabled`

What's New

NEW

AVAudioEngine

- Manual rendering
- Auto shutdown

AVAudioPlayerNode

- Completion callbacks

Completion Callbacks

NEW

Existing buffer/file completion handlers called when the data has been consumed

New completion handler and callback types

```
AVAudioPlayerNodeCompletionCallbackType
    .dataConsumed
    .dataRendered
    .dataPlayedBack
```

Completion Callbacks

NEW

`.dataConsumed`

- Data has been consumed, same as the existing completion handlers
- The buffer can be recycled, more data can be scheduled

`.dataRendered`

- Data has been output by the player
- Useful in manual rendering mode
- Does not account for any downstream signal processing latency

Completion Callbacks

NEW

`.dataPlayedBack`

- Buffer/file has finished playing
- Applicable only when the engine is rendering to an audio device
- Accounts for both (small) downstream signal processing latency and (possibly significant) audio playback device latency

Completion Callbacks

NEW

```
.dataPlayedBack
```

- Buffer/file has finished playing
- Applicable only when the engine is rendering to an audio device
- Accounts for both (small) downstream signal processing latency and (possibly significant) audio playback device latency

Completion Callbacks

NEW

`.dataPlayedBack`

- Buffer/file has finished playing
- Applicable only when the engine is rendering to an audio device
- Accounts for both (small) downstream signal processing latency and (possibly significant) audio playback device latency

```
player.scheduleFile(file, at: nil, completionCallbackType: .dataPlayedBack) {
    (callbackType) in
    // file has finished playing from listener's perspective
    // notify to stop the engine and update UI
})
```

AVAudioEngine
Summary

NEW

AVAudioEngine

- Manual rendering
- Auto shutdown

AVAudioPlayerNode

- Completion callbacks

Deprecation coming soon (2018)

- AUGraph ✕

AVAudioSession

AirPlay 2 Support

NEW

AirPlay 2 - new technology on iOS, tvOS and macOS

- Multi-room audio with AirPlay 2 capable devices

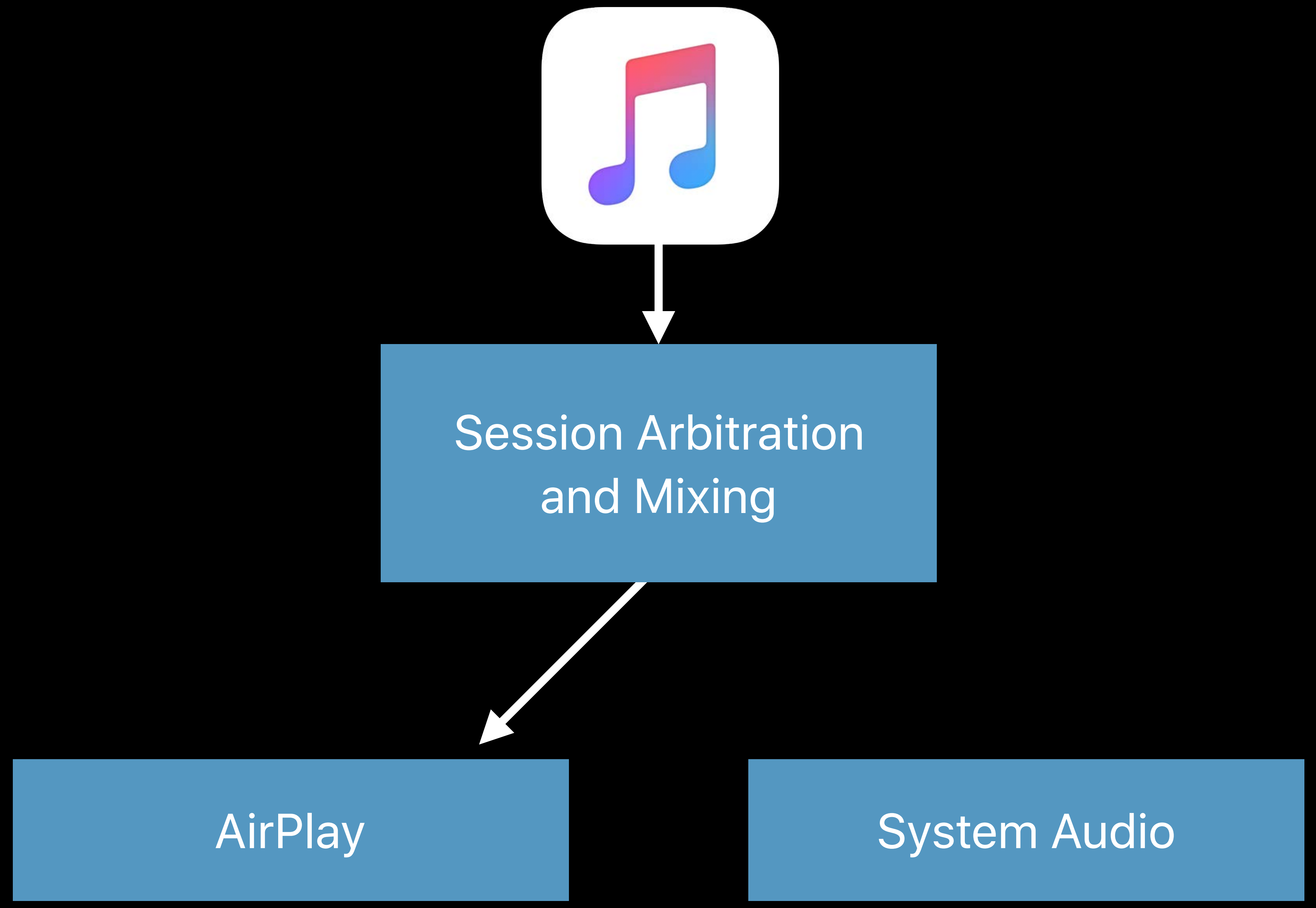
Long-form audio applications

- Content - music, podcasts etc.
- Separate, shared audio route to AirPlay 2 devices
- New AVAudioSession API for an application to identify itself as long-form

Introducing AirPlay 2 Session 509 Thursday 4:10PM

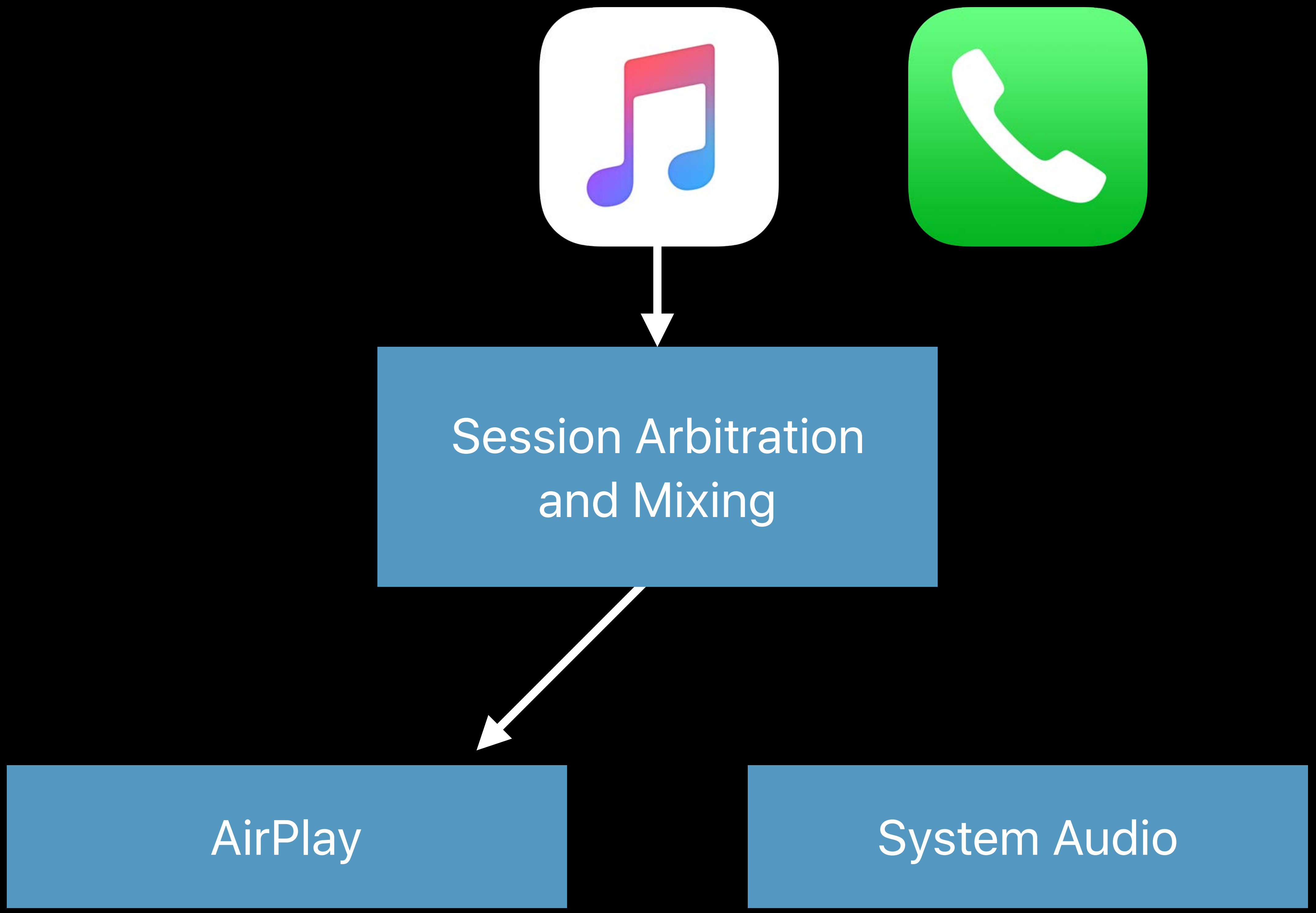
Audio Routing (iOS) - Current Behavior

Music and phone call



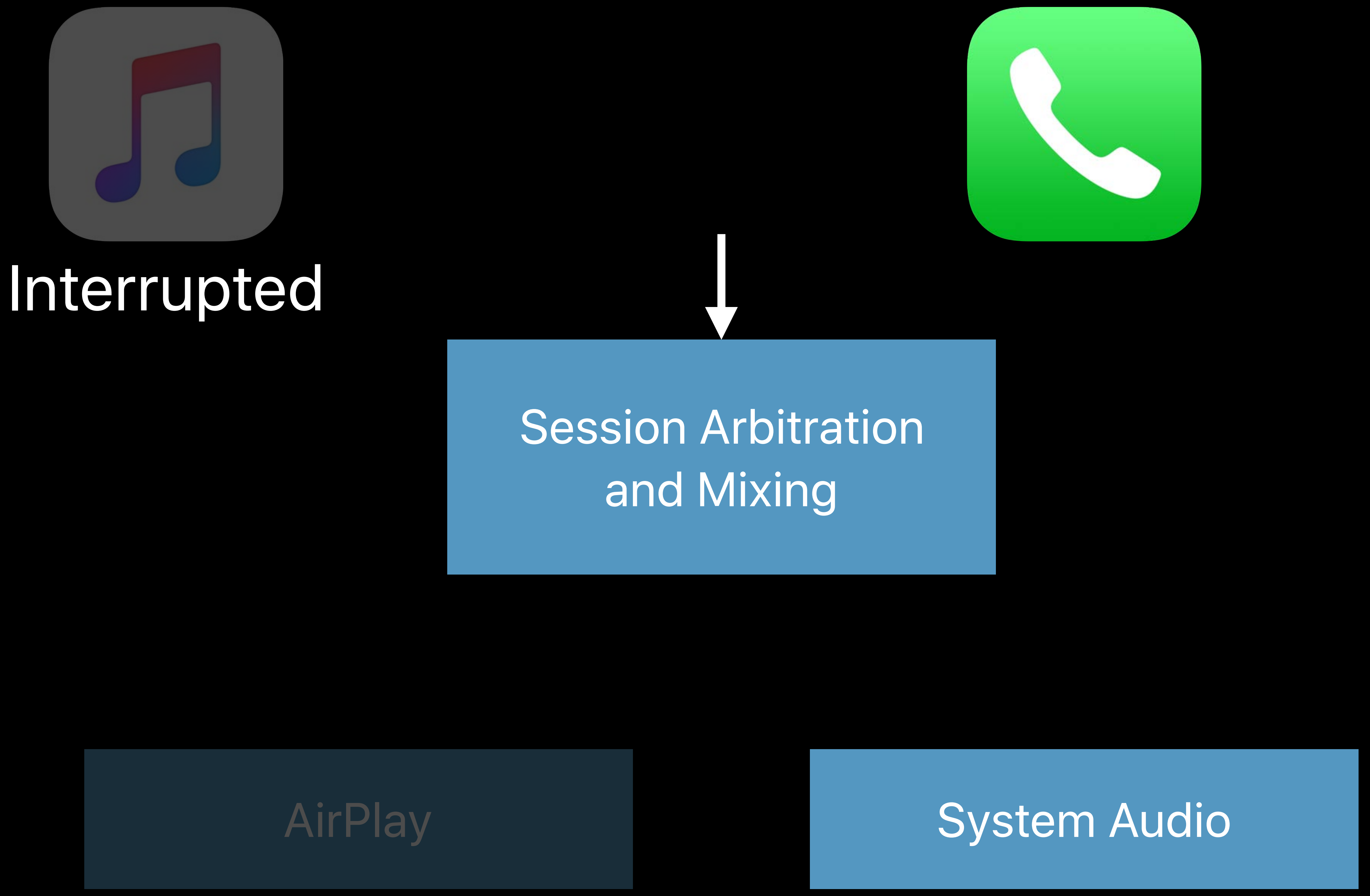
Audio Routing (iOS) - Current Behavior

Music and phone call



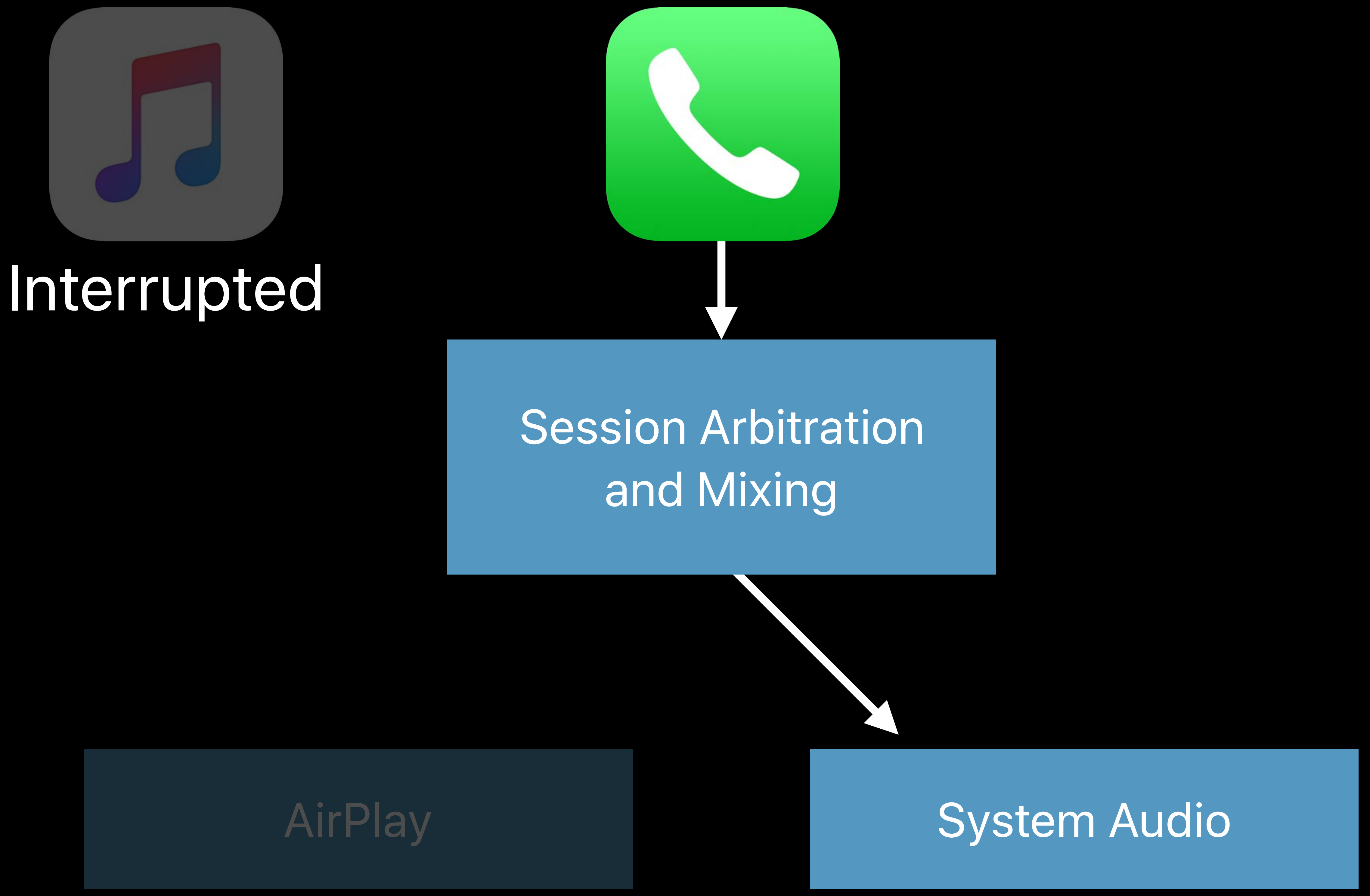
Audio Routing (iOS) - Current Behavior

Music and phone call



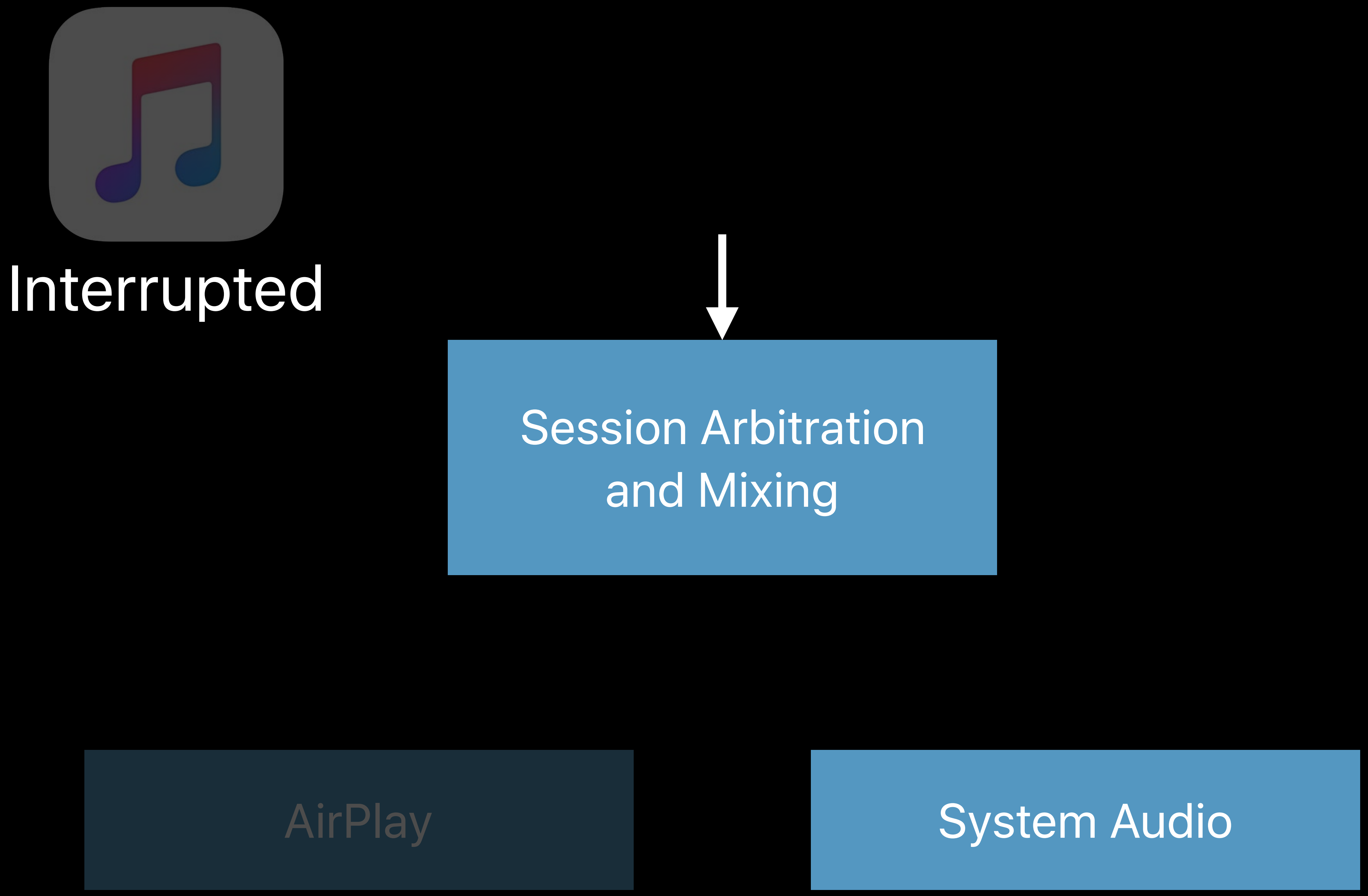
Audio Routing (iOS) - Current Behavior

Music and phone call



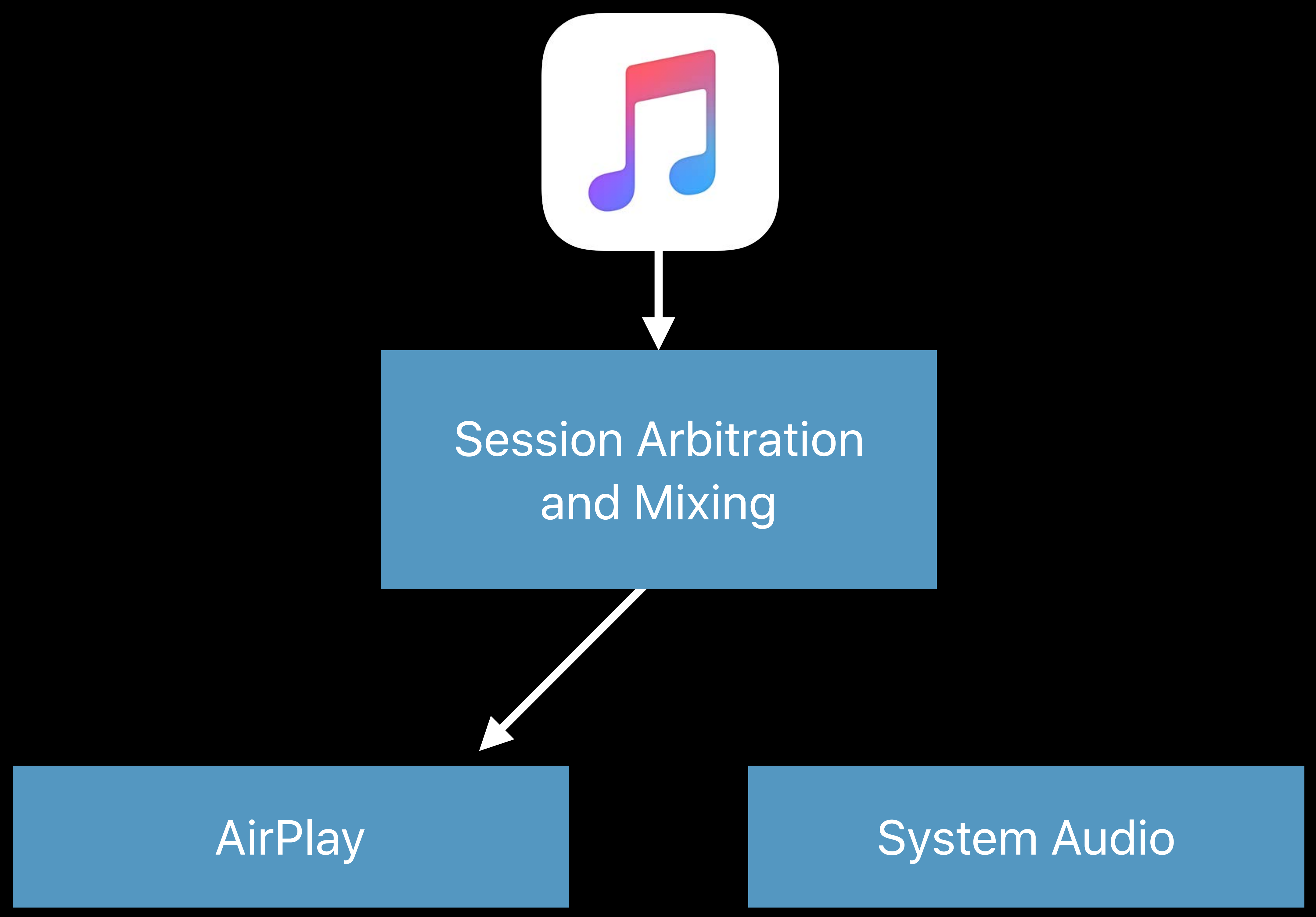
Audio Routing (iOS) - Current Behavior

Music and phone call



Audio Routing (iOS) - Current Behavior

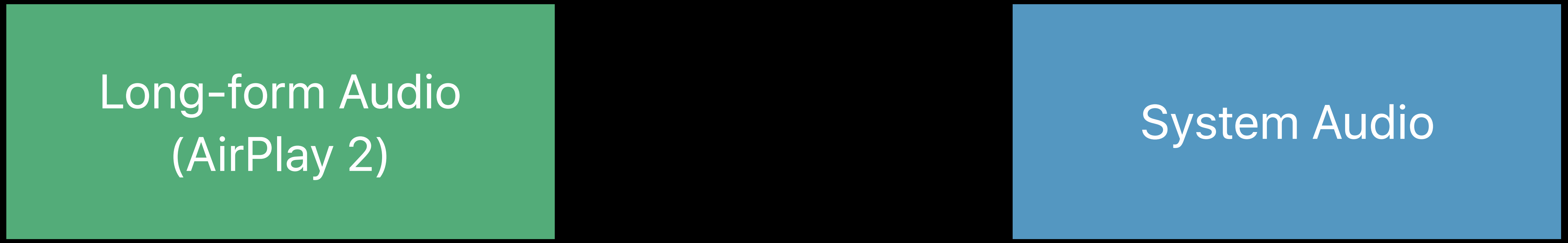
Music and phone call



Long-form Audio Routing (iOS)

Music and phone call coexistence

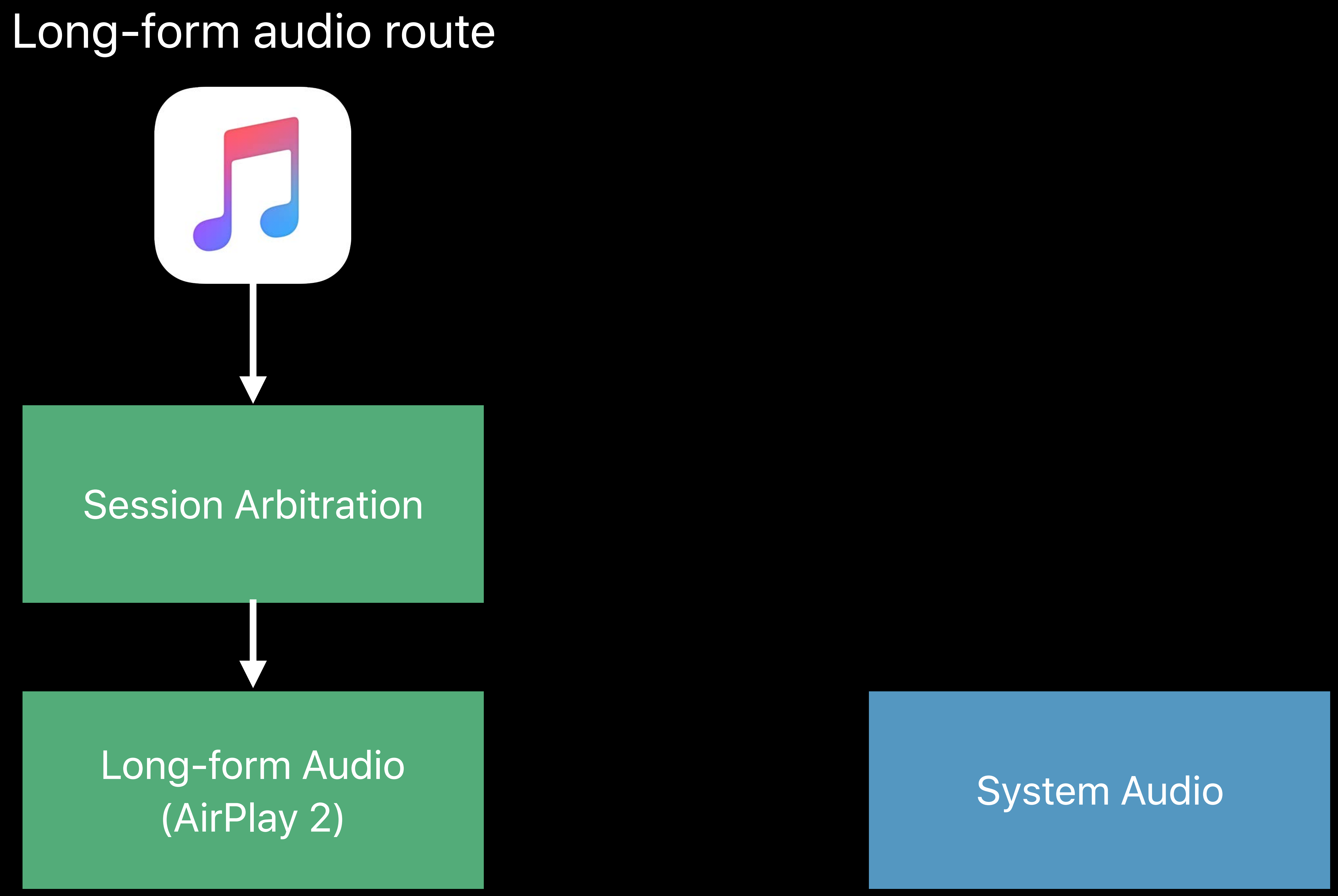
NEW



Long-form Audio Routing (iOS)

Music and phone call coexistence

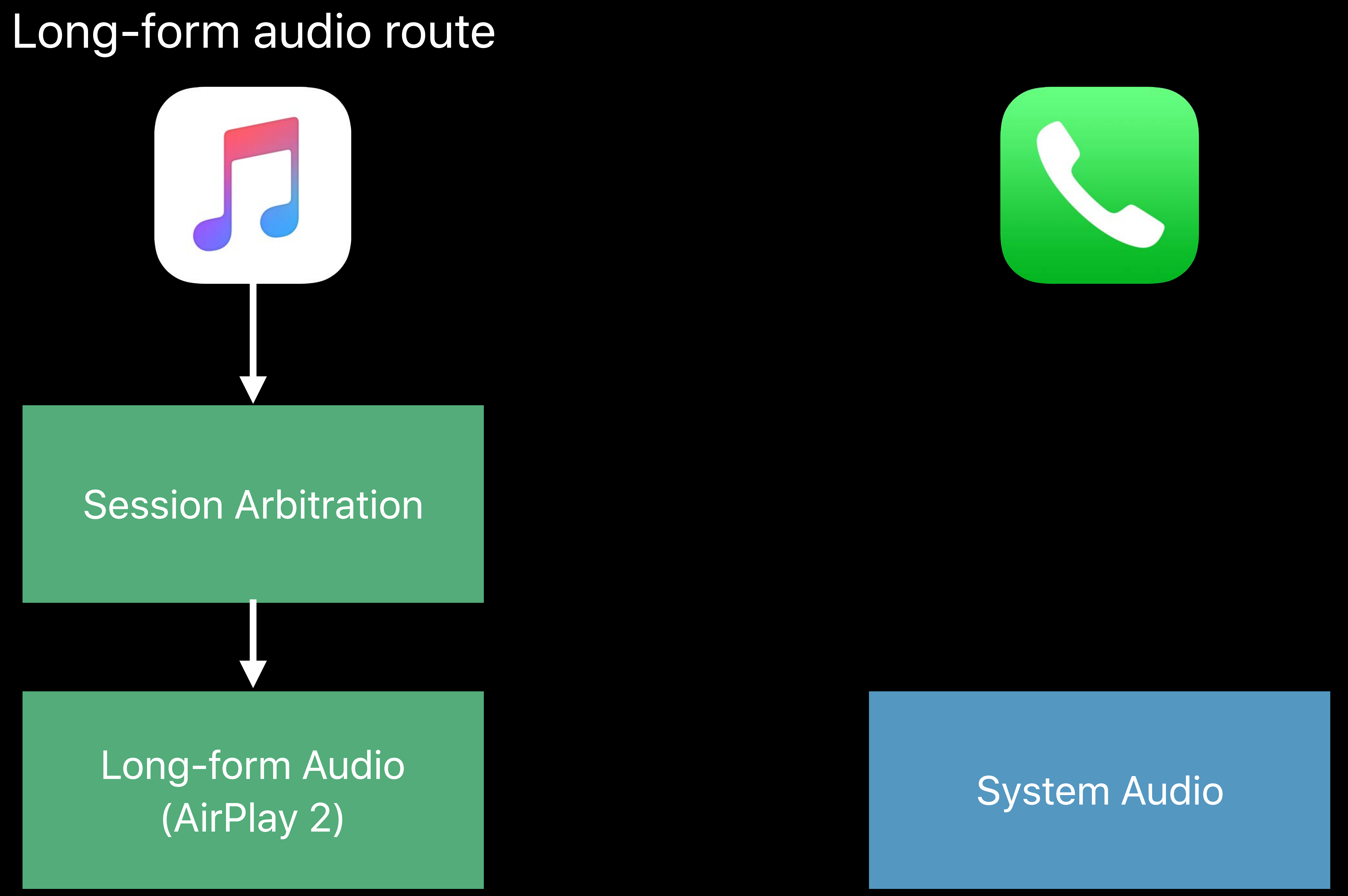
NEW



Long-form Audio Routing (iOS)

Music and phone call coexistence

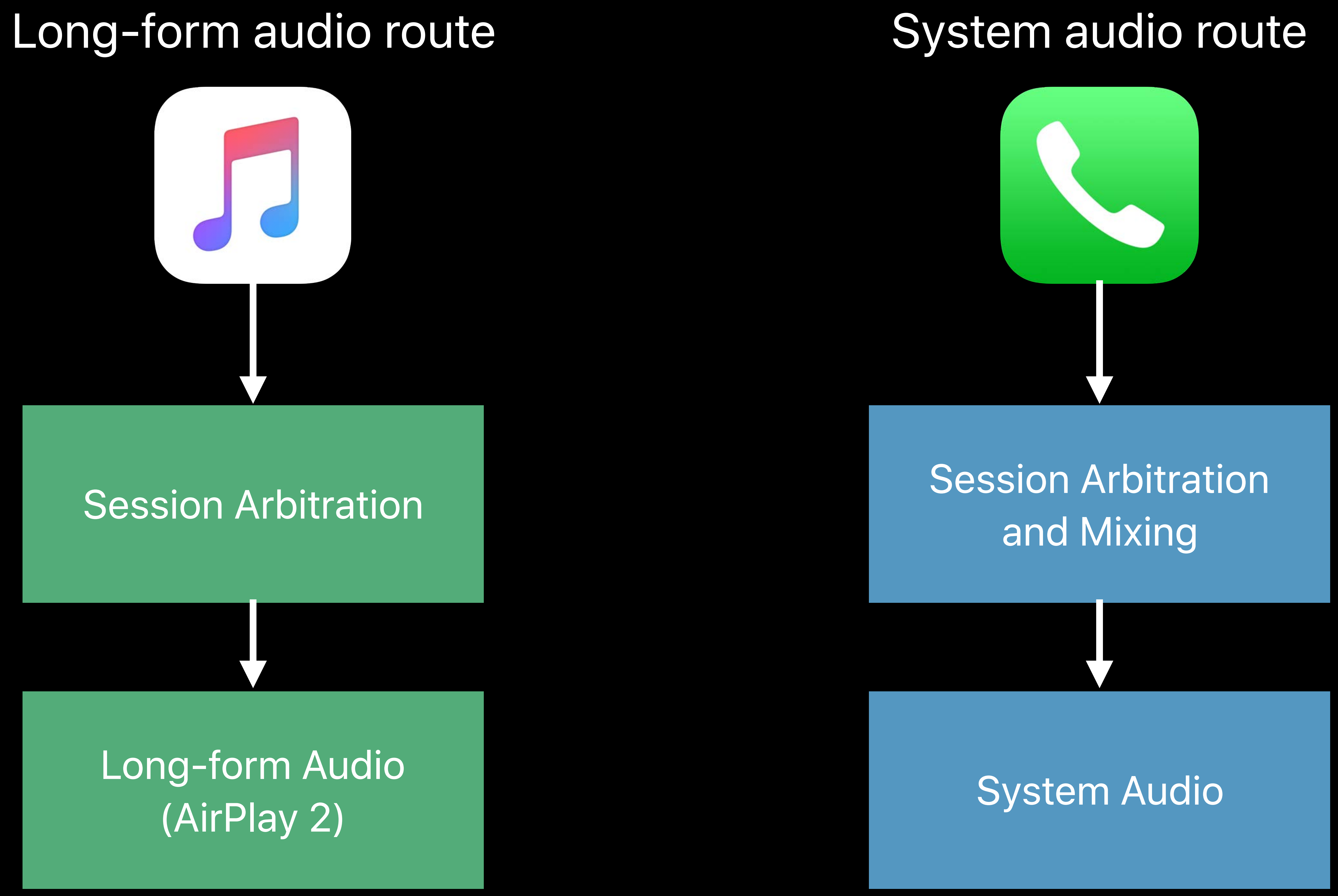
NEW



Long-form Audio Routing (iOS)

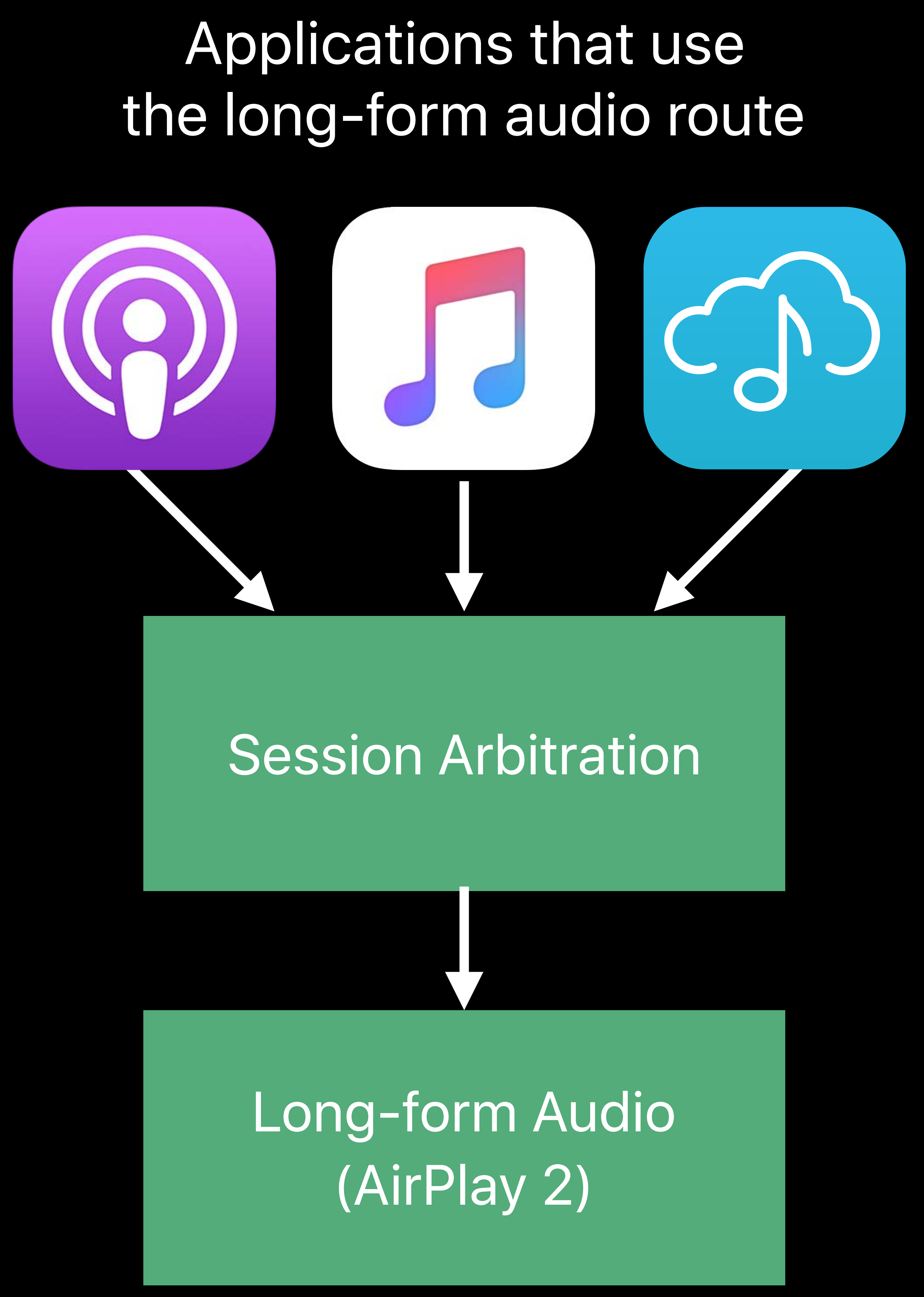
Music and phone call coexistence

NEW



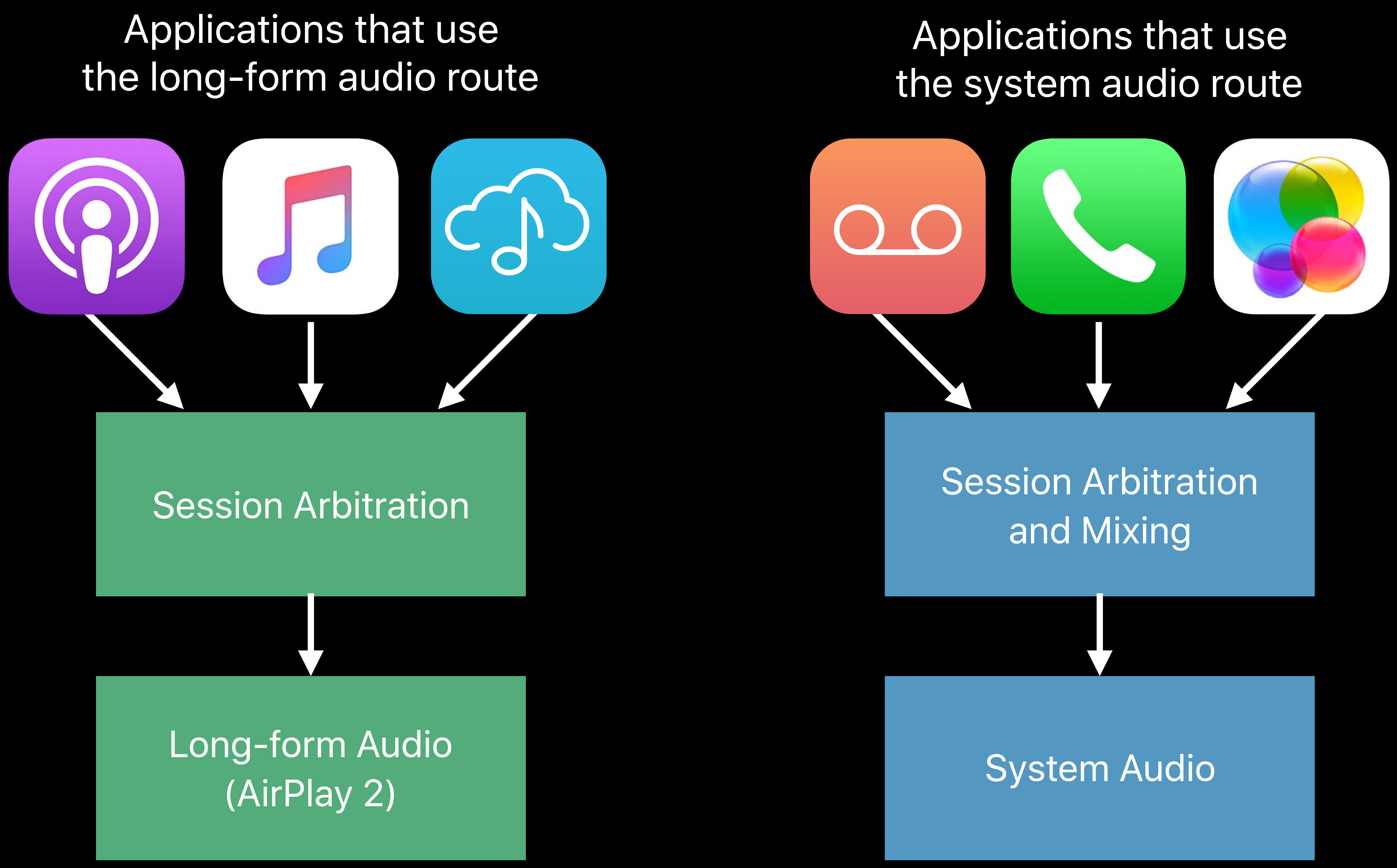
Long-form Audio Routing (iOS and tvOS)

NEW



Long-form Audio Routing (iOS and tvOS)

NEW


Applications that use the long-form audio route
Session Arbitration
Long-form Audio (AirPlay 2)
Applications that use the system audio route
Session Arbitration and Mixing
System Audio

```
//Long-form Audio Routing (iOS and tvOS), code example
```

NEW

```swift
let mySession = AVAudioSession.sharedInstance()
do {
   try mySession.setCategory(AVAudioSessionCategoryPlayback,
            mode: AVAudioSessionModeDefault,
            routeSharingPolicy: .longForm)
} catch {
   // handle errors
}
```

```
//Long-form Audio Routing (iOS and tvOS), code example
```

NEW

```swift
let mySession = AVAudioSession.sharedInstance()
do {
   try mySession.setCategory(AVAudioSessionCategoryPlayback,
               mode: AVAudioSessionModeDefault,
               routeSharingPolicy: .longForm)
} catch {
   // handle errors
}
```

```
//Long-form Audio Routing (iOS and tvOS), code example
```

NEW

```swift
let mySession = AVAudioSession.sharedInstance()
do {
    try mySession.setCategory(AVAudioSessionCategoryPlayback,
                mode: AVAudioSessionModeDefault,
                routeSharingPolicy: .longForm)
} catch {
    // handle errors
}
```

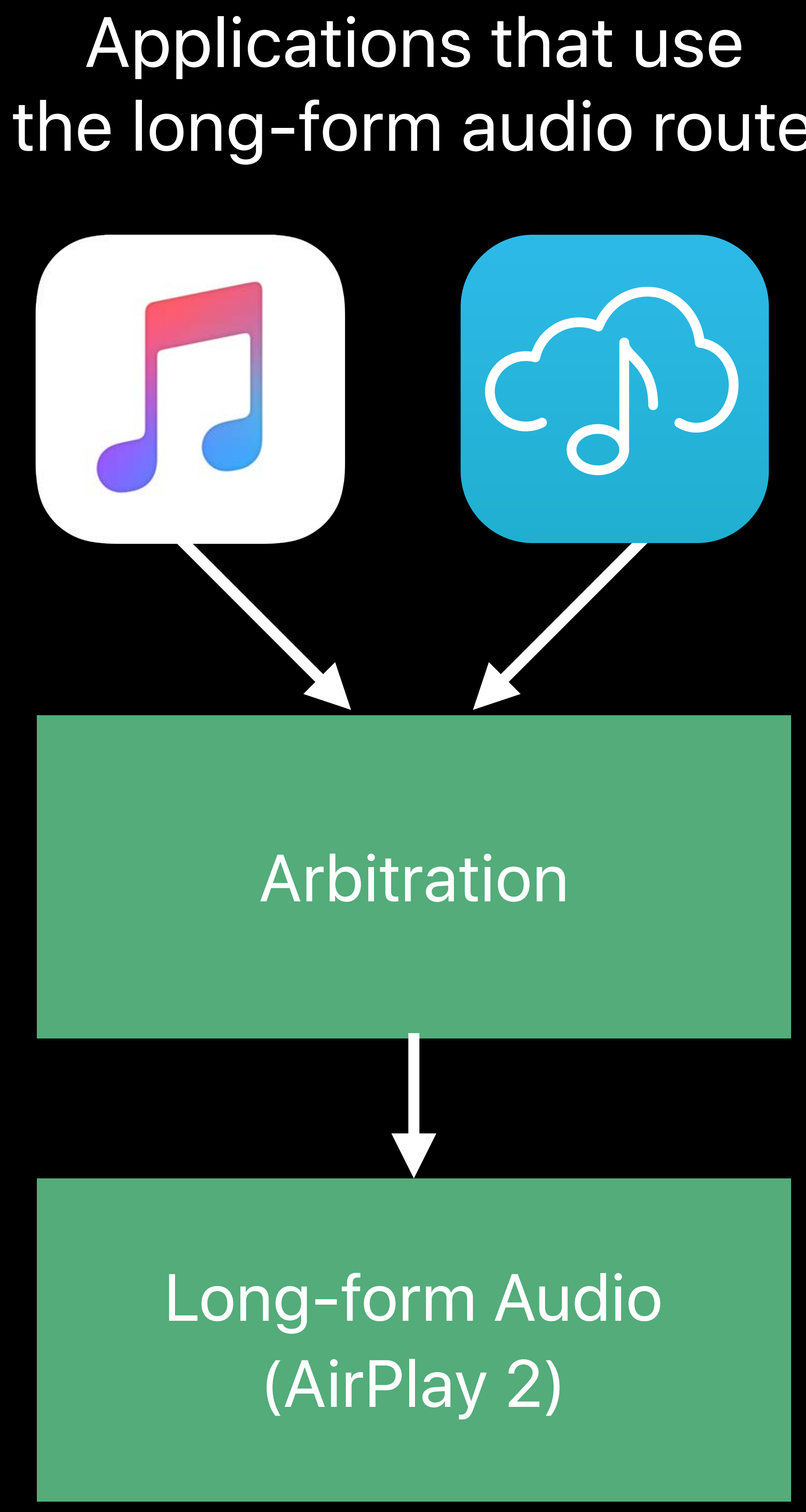
```
//Long-form Audio Routing (iOS and tvOS), code example
```

NEW

```swift
let mySession = AVAudioSession.sharedInstance()
do {
   try mySession.setCategory(AVAudioSessionCategoryPlayback,
           mode: AVAudioSessionModeDefault,
           routeSharingPolicy: .longForm)
} catch {
   // handle errors
}
```

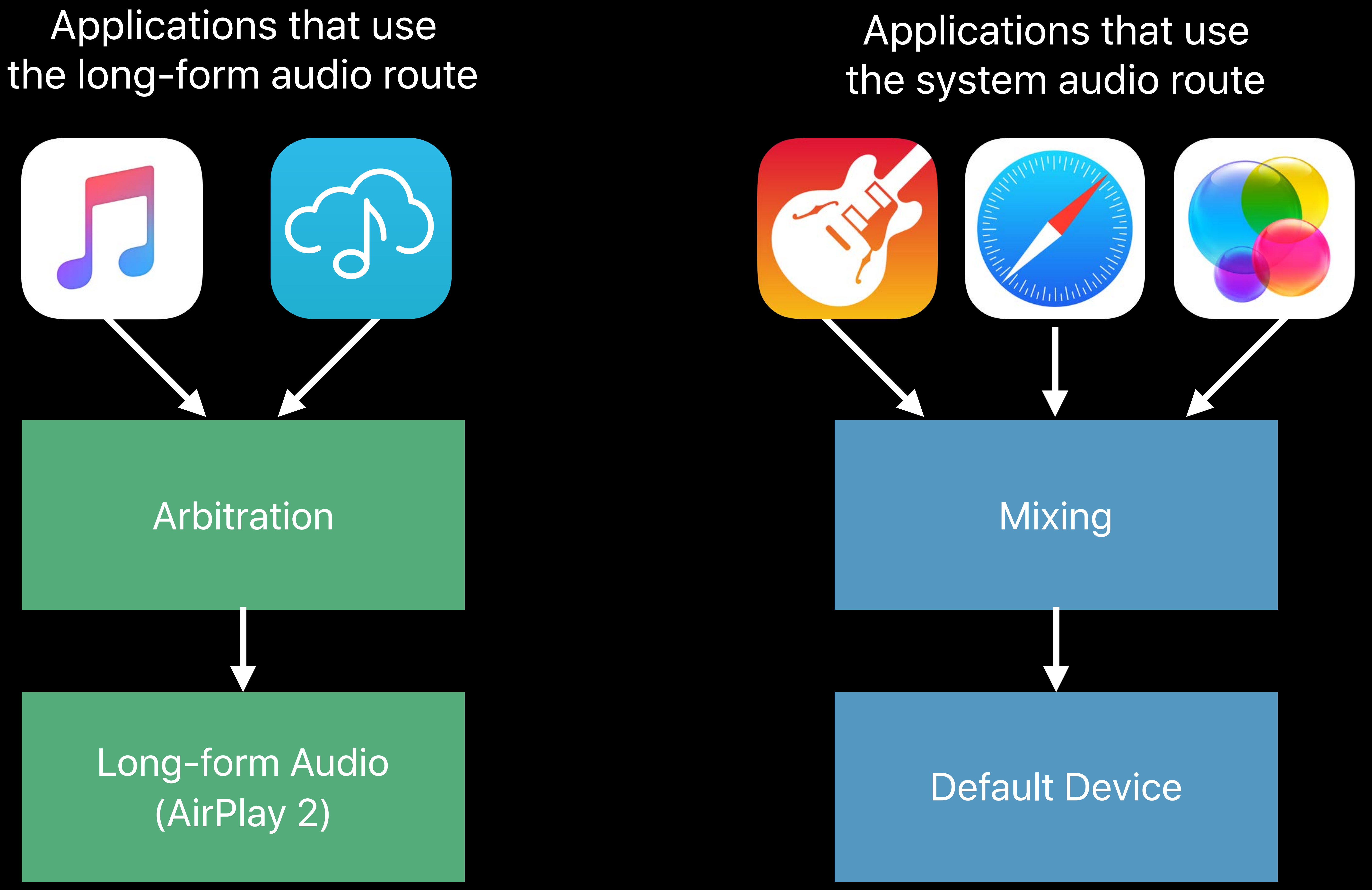
Long-form Audio Routing (macOS)

NEW



Long-form Audio Routing (macOS)

NEW


Applications that use the long-form audio route
Applications that use the system audio route
Arbitration
Long-form Audio (AirPlay 2)
Mixing
Default Device

```
//Long-form Audio Routing (macOS), code example
```

NEW

```swift
let mySession = AVAudioSession.sharedInstance()
do {
   try mySession.setRouteSharingPolicy(.longForm)
} catch {
   // handle errors
}
```

//Long-form Audio Routing (macOS), code example

NEW

```
let mySession = AVAudioSession.sharedInstance()
do {
   try mySession.setRouteSharingPolicy(.longForm)
} catch {
   // handle errors
}
```

NEW

```
//Long-form Audio Routing (macOS), code example


let mySession = AVAudioSession.sharedInstance()
do {
    try mySession.setRouteSharingPolicy(.longForm)
} catch {
    // handle errors
}
```

Enhancements in watchOS

watchOS 4.0

Playback and recording

NEW

Playback

- AVAudioPlayer (watchOS 3.1 SDK)

Recording

- AVAudioInputNode (AVAudioEngine)
- AVAudioRecorder
- AVAudioSession recording permissions

Formats supported

- AAC-LC, AAC-ELD, HE-AAC, HE-AACv2, MP3 (decoding only), Opus

watchOS 4.0

Recording policies

NEW

Recording can start only in foreground

Recording allowed to continue in the background (red microphone icon displayed)

Recording in background is CPU limited

- https://developer.apple.com/reference/healthkit/hkworkoutsession

AVAudioEngine

AVAudioSession

watchOS

AUAudioUnit

Other Enhancements

Inter-Device Audio Mode (IDAM)

AUAudioUnit

AU View Configuration

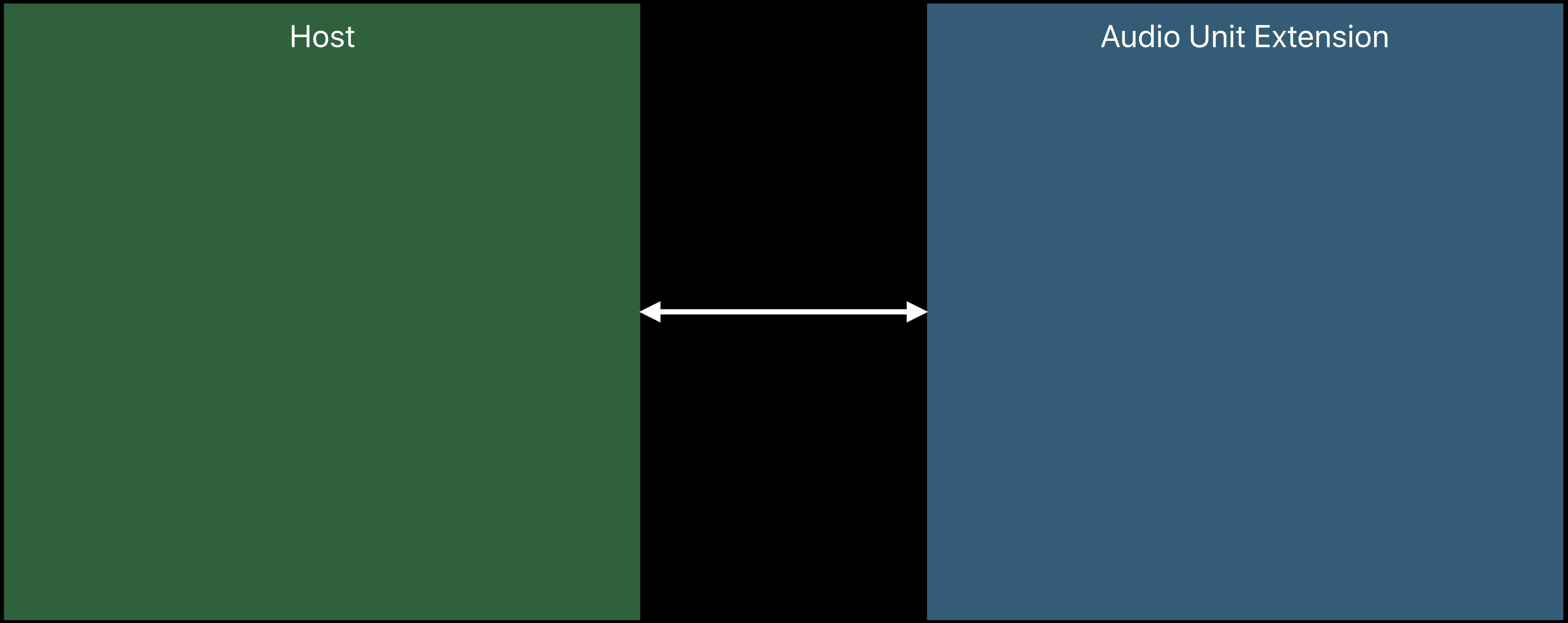
Host applications decide how to display UI for AUs

Current limitations

- No standard view sizes defined
- AUs are supposed to adapt to any view size chosen by host

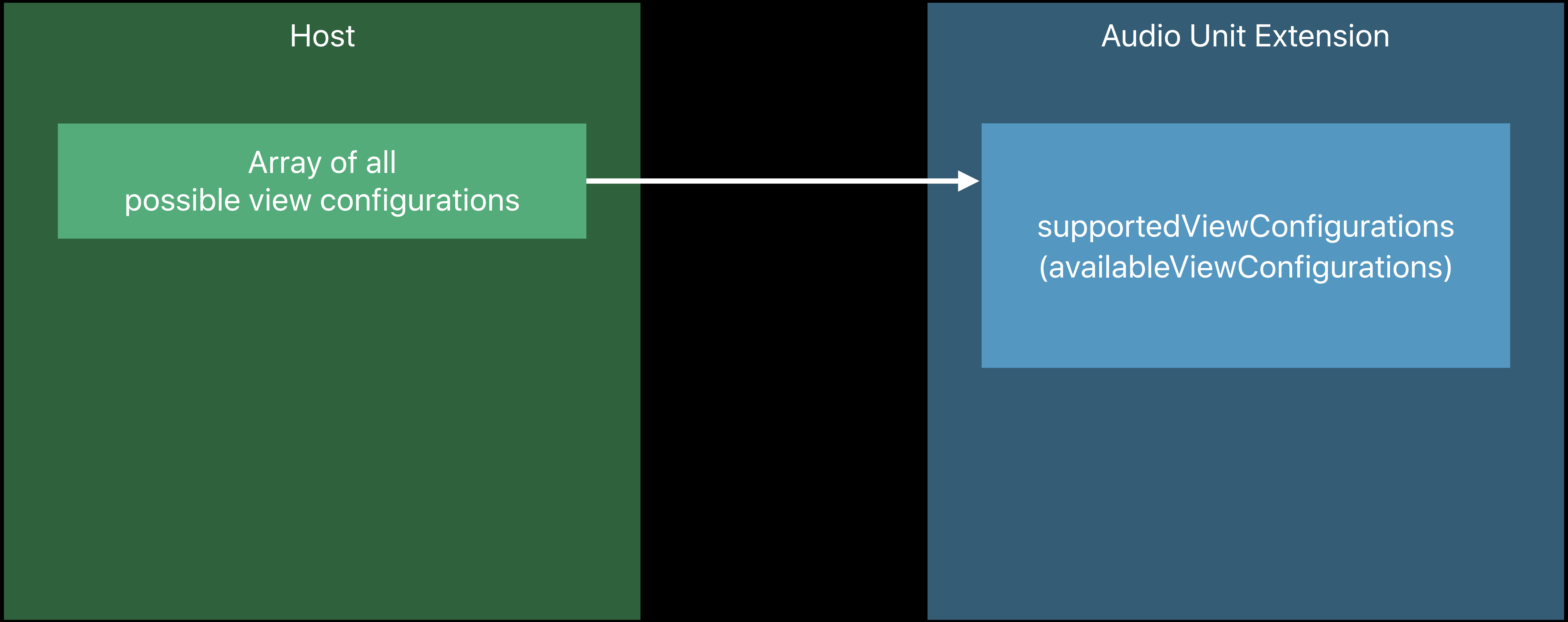
AU Preferred View Configuration

NEW



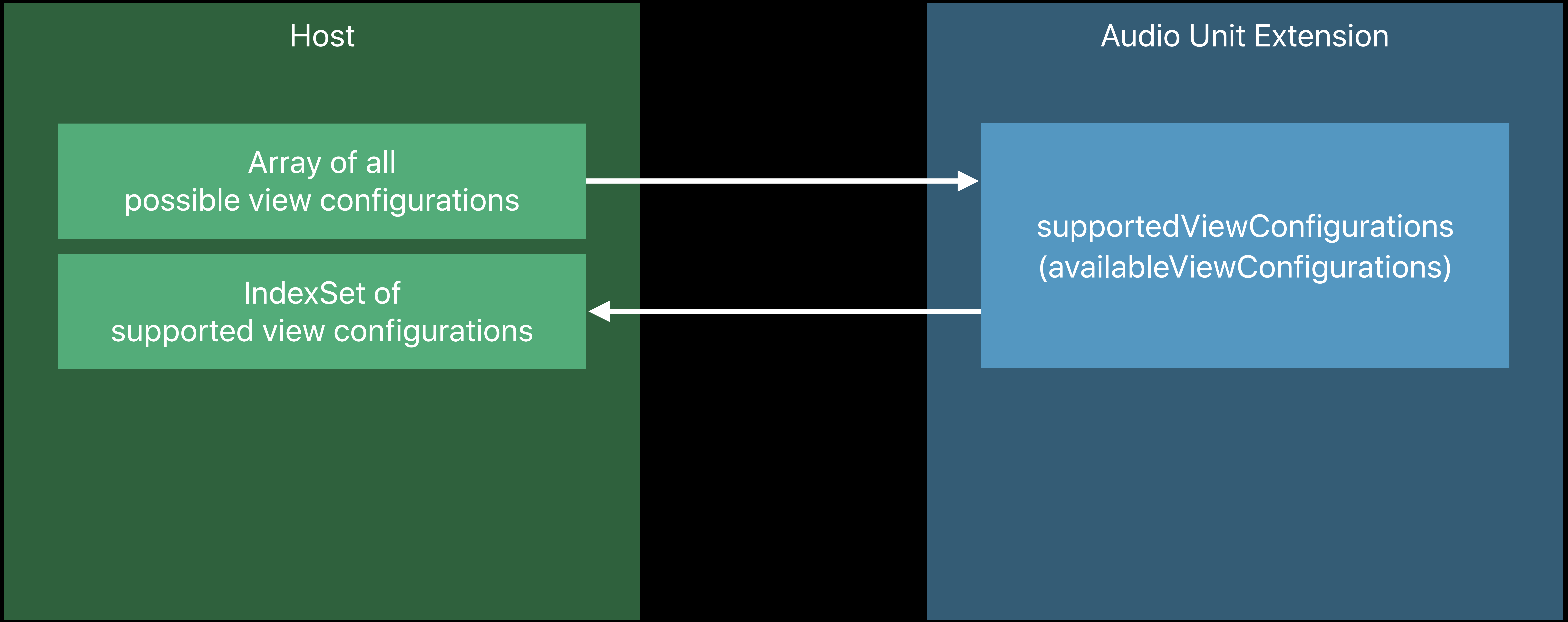
AU Preferred View Configuration

NEW


Host
Array of all possible view configurations
Audio Unit Extension
supportedViewConfigurations
(availableViewConfigurations)

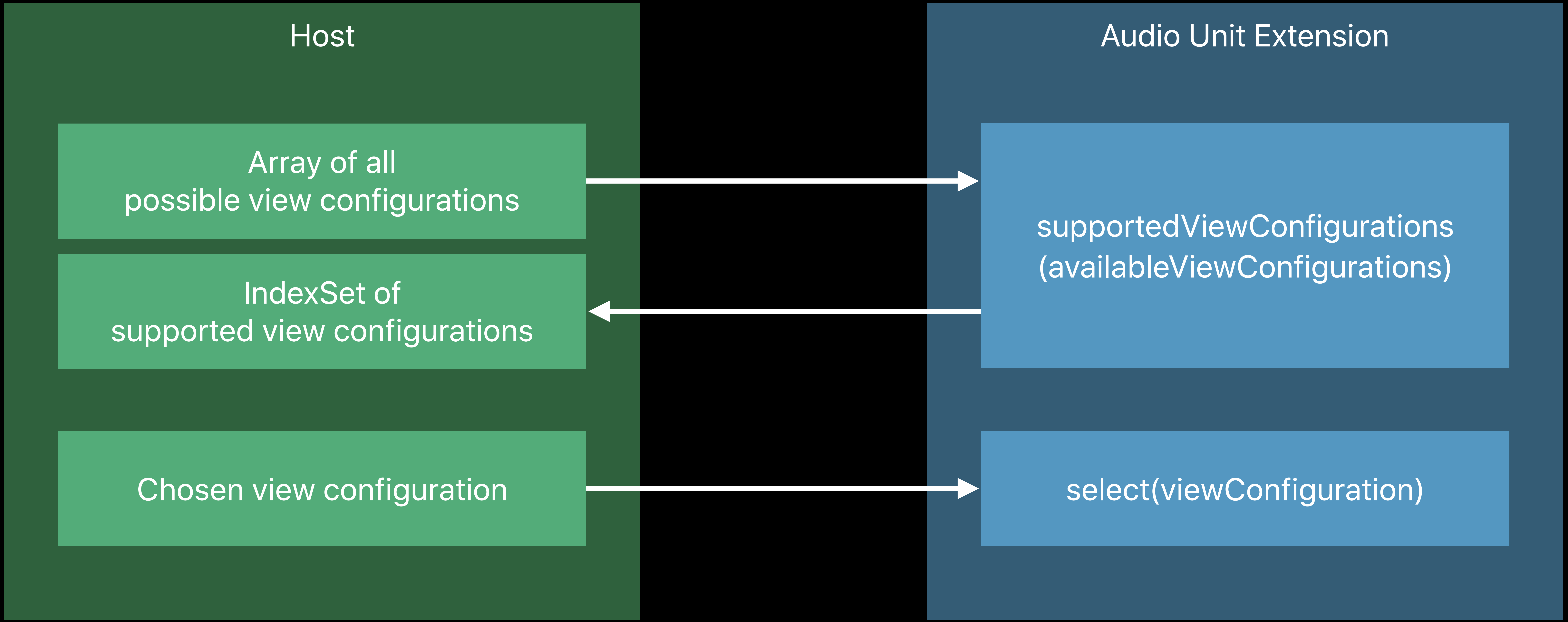
AU Preferred View Configuration

NEW


Host
Array of all possible view configurations
IndexSet of supported view configurations
Audio Unit Extension
supportedViewConfigurations (availableViewConfigurations)

AU Preferred View Configuration

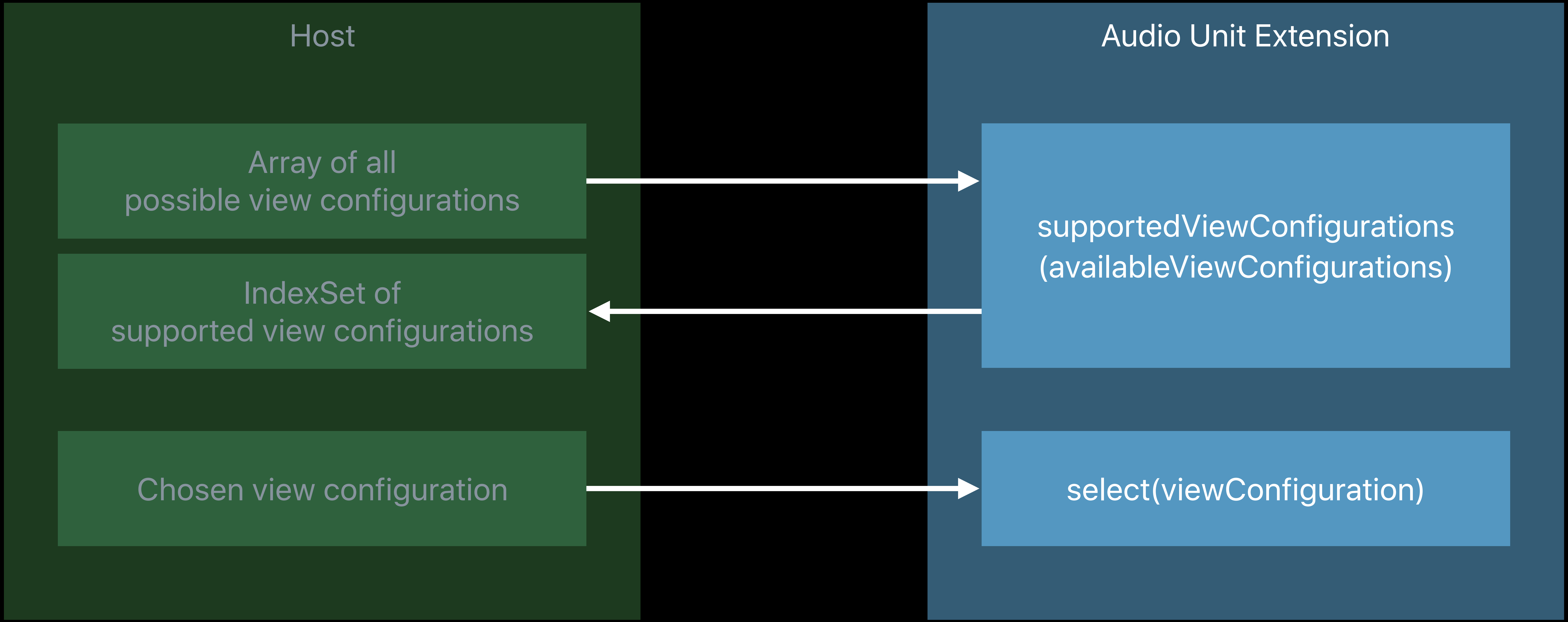
NEW


Host
Array of all possible view configurations
IndexSet of supported view configurations
Chosen view configuration
Audio Unit Extension
supportedViewConfigurations (availableViewConfigurations)
select(viewConfiguration)

AU Preferred View Configuration

Code example - AU extension

NEW


Host
Audio Unit Extension
Array of all possible view configurations
IndexSet of supported view configurations
Chosen view configuration
supportedViewConfigurations (availableViewConfigurations)
select(viewConfiguration)

```
//AU Preferred View Configuration
Code example - AU extension
```

NEW

```
override public func supportedViewConfigurations(_ availableViewConfigurations:
    [AUAudioUnitViewConfiguration]) -> IndexSet {
    var result = NSMutableIndexSet()
    for (index, config) in availableViewConfigurations.enumerated() {
        // check if the config (width, height, hostHasController) is supported
        // a config of 0x0 (default full size) must always be supported
        if isConfigurationSupported(config) {
            result.add(index)
        }
    }
    return result as IndexSet
}
```

```
//AU Preferred View Configuration
Code example - AU extension
```

NEW

```swift
override public func supportedViewConfigurations(_ availableViewConfigurations:
   [AUAudioUnitViewConfiguration]) -> IndexSet {
   var result = NSMutableIndexSet()
   for (index, config) in availableViewConfigurations.enumerated() {
      // check if the config (width, height, hostHasController) is supported
      // a config of 0x0 (default full size) must always be supported
      if isConfigurationSupported(config) {
         result.add(index)
      }
   }
   return result as IndexSet
}
```

NEW

```
//AU Preferred View Configuration
Code example - AU extension

override public func supportedViewConfigurations(_ availableViewConfigurations:
   [AUAudioUnitViewConfiguration]) -> IndexSet {
   var result = NSMutableIndexSet()
   for (index, config) in availableViewConfigurations.enumerated() {
      // check if the config (width, height, hostHasController) is supported
      // a config of 0x0 (default full size) must always be supported
      if isConfigurationSupported(config) {
         result.add(index)
      }
   }
   return result as IndexSet
}
```

```
//AU Preferred View Configuration
Code example - AU extension
```

NEW

```swift
override public func supportedViewConfigurations(_ availableViewConfigurations:
   [AUAudioUnitViewConfiguration]) -> IndexSet {
   var result = NSMutableIndexSet()
   for (index, config) in availableViewConfigurations.enumerated() {
      // check if the config (width, height, hostHasController) is supported
      // a config of 0x0 (default full size) must always be supported
      if isConfigurationSupported(config) {
         result.add(index)
      }
   }
   return result as IndexSet
}
```

```
//AU Preferred View Configuration
Code example - AU extension

override public func supportedViewConfigurations(_ availableViewConfigurations:
   [AUAudioUnitViewConfiguration]) -> IndexSet {
   var result = NSMutableIndexSet()
   for (index, config) in availableViewConfigurations.enumerated() {
      // check if the config (width, height, hostHasController) is supported
      // a config of 0x0 (default full size) must always be supported
      if isConfigurationSupported(config) {
         result.add(index)
      }
   }
   return result as IndexSet
}
```

NEW

```
//AU Preferred View Configuration
Code example - AU extension

override public func select(_ viewConfiguration: AUAudioUnitViewConfiguration) {
   // configuration selected by host, used by view controller to re-arrange its view
   self.currentViewConfiguration = viewConfiguration
   self.viewController?.selectViewConfig(self.currentViewConfiguration)
}
```

NEW

```
//AU Preferred View Configuration
Code example - AU extension
```

NEW

```swift
override public func select(_ viewConfiguration: AUAudioUnitViewConfiguration) {
   // configuration selected by host, used by view controller to re-arrange its view
   self.currentViewConfiguration = viewConfiguration
   self.viewController?.selectViewConfig(self.currentViewConfiguration)
}
```

```
//AU Preferred View Configuration
Code example - AU extension
```

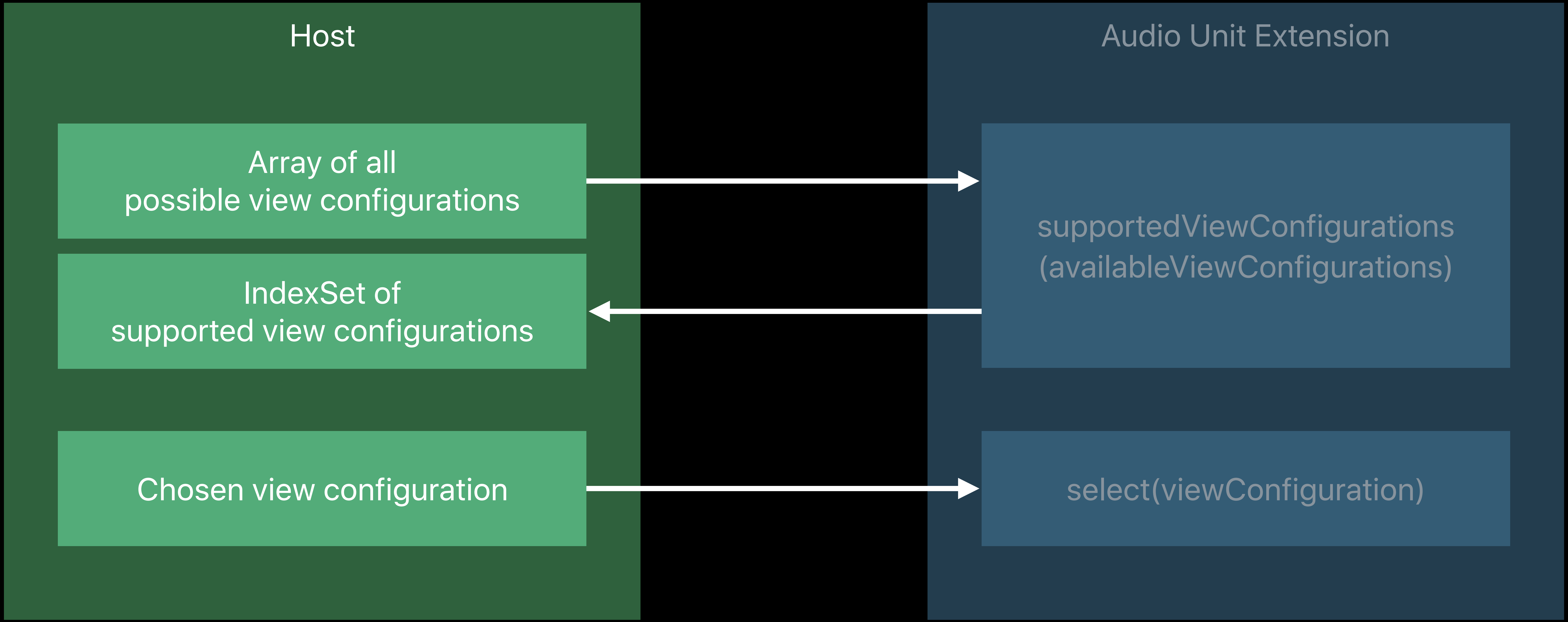
NEW

```
override public func select(_ viewConfiguration: AUAudioUnitViewConfiguration) {
    // configuration selected by host, used by view controller to re-arrange its view
    self.currentViewConfiguration = viewConfiguration
    self.viewController?.selectViewConfig(self.currentViewConfiguration)
}
```

AU Preferred View Configuration

Code example - host application

NEW


Host
Array of all possible view configurations
IndexSet of supported view configurations
Chosen view configuration
Audio Unit Extension
supportedViewConfigurations (availableViewConfigurations)
select(viewConfiguration)

NEW

```
//AU Preferred View Configuration
Code example - host application

var smallConfigActive: Bool = false // true if the small view is the currently active one
@IBAction func toggleViewModes(_ sender: AnyObject?)    {
   guard audioUnit = self.engine.audioUnit else { return }
   let largeConfig = AUAudioUnitViewConfiguration(width: 600, height: 400,
                                hostHasController: false)
   let smallConfig = AUAudioUnitViewConfiguration(width: 300, height: 200,
                                hostHasController: true)

   let supportedIndices = audioUnit.supportedViewConfigurations([smallConfig, largeConfig])
   if supportedIndices.count == 2  {
         audioUnit.select(self.smallConfigActive ? largeConfig : smallConfig)
         self.smallConfigActive = !self.smallConfigActive
   }
}
```

NEW

```
//AU Preferred View Configuration
Code example - host application

var smallConfigActive: Bool = false // true if the small view is the currently active one
@IBAction func toggleViewModes(_ sender: AnyObject?)    {
   guard audioUnit = self.engine.audioUnit else { return }
   let largeConfig = AUAudioUnitViewConfiguration(width: 600, height: 400,
                             hostHasController: false)
   let smallConfig = AUAudioUnitViewConfiguration(width: 300, height: 200,
                             hostHasController: true)

   let supportedIndices = audioUnit.supportedViewConfigurations([smallConfig, largeConfig])
   if supportedIndices.count == 2  {
         audioUnit.select(self.smallConfigActive ? largeConfig : smallConfig)
         self.smallConfigActive = !self.smallConfigActive
   }
}
```

NEW

```
//AU Preferred View Configuration
Code example - host application

var smallConfigActive: Bool = false // true if the small view is the currently active one
@IBAction func toggleViewModes(_ sender: AnyObject?)    {
   guard audioUnit = self.engine.audioUnit else { return }
   let largeConfig = AUAudioUnitViewConfiguration(width: 600, height: 400,
                            hostHasController: false)
   let smallConfig = AUAudioUnitViewConfiguration(width: 300, height: 200,
                            hostHasController: true)

   let supportedIndices = audioUnit.supportedViewConfigurations([smallConfig, largeConfig])
   if supportedIndices.count == 2  {
         audioUnit.select(self.smallConfigActive ? largeConfig : smallConfig)
         self.smallConfigActive = !self.smallConfigActive
   }
}
```

NEW

```
//AU Preferred View Configuration
Code example - host application

var smallConfigActive: Bool = false // true if the small view is the currently active one
@IBAction func toggleViewModes(_ sender: AnyObject?)    {
   guard audioUnit = self.engine.audioUnit else { return }
   let largeConfig = AUAudioUnitViewConfiguration(width: 600, height: 400,
                                                  hostHasController: false)
   let smallConfig = AUAudioUnitViewConfiguration(width: 300, height: 200,
                                                  hostHasController: true)

   let supportedIndices = audioUnit.supportedViewConfigurations([smallConfig, largeConfig])
   if supportedIndices.count == 2  {
         audioUnit.select(self.smallConfigActive ? largeConfig : smallConfig)
         self.smallConfigActive = !self.smallConfigActive
   }
}
```

AU MIDI Output

NEW

AU can emit MIDI output synchronized with its audio output

Host sets a block on the AU to be called every render cycle

Host can record/edit both MIDI performance and audio output from the AU

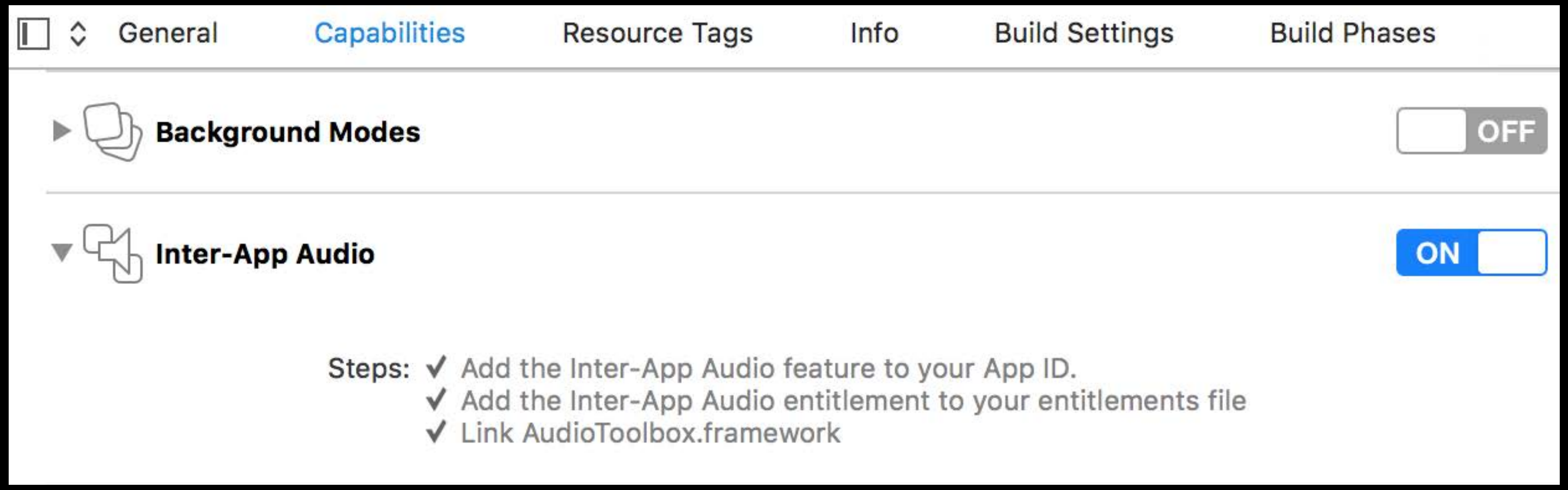
```
*MIDIOutputNames
MIDIOutputEventBlock
```

Other Updates

NEW

Entitlement

- AU extension host applications linked against iOS 11 SDK and later will need 'inter-app-audio' entitlement



Background Modes OFF

Inter-App Audio ON

Steps: ✓ Add the Inter-App Audio feature to your App ID.
✓ Add the Inter-App Audio entitlement to your entitlements file
✓ Link AudioToolbox.framework

AU short name

```
*audioUnitShortName
```

Demo

AUAudioUnit

Béla Balázs, Audio Artisan

Other Enhancements

Audio Formats

NEW

FLAC (Free Lossless Audio Codec)

- Codec, file, and streaming support
- Content distribution, streaming applications

Opus

- Codec support
- File I/O using .caf container
- VOIP applications

Spatial Audio Formats
B-Format

NEW

Audio stream is regular PCM

File container .caf

B-format: W,X,Y,Z

- 1st order ambisonics

```
kAudioChannelLayoutTag_Ambisonic_B_Format
```

Spatial Audio Formats

Higher Order Ambisonics

NEW

N order ambisonics (N is 1..254)

`kAudioChannelLayoutTag_HOA_ACN_SN3D` - SN3D normalized streams

`kAudioChannelLayoutTag_HOA_ACN_N3D` - N3D normalized streams

ACN (Ambisonic Channel Number) Channels

`kAudioChannelLabel_HOA_ACN_0..65024`

AudioFormat support for converting

- Between B-format, ACN_SN3D, ACN_N3D
- From ambisonics to arbitrary speaker layout

Spatial Mixer

Head-Related Transfer Function (HRTF)

NEW

AUSpatialMixer - `kSpatializationAlgorithm_HRTFHQ`

AVAudioEnvironmentNode - `AVAudio3DMixingRenderingAlgorithmHRTFHQ`

Features

- Better frequency response
- Better localization of sources in a 3D space

AVAudioEngine

AVAudioSession

watchOS

AUAudioUnit

Other Enhancements

Inter-Device Audio Mode (IDAM)

Inter-Device Audio Mode (IDAM)

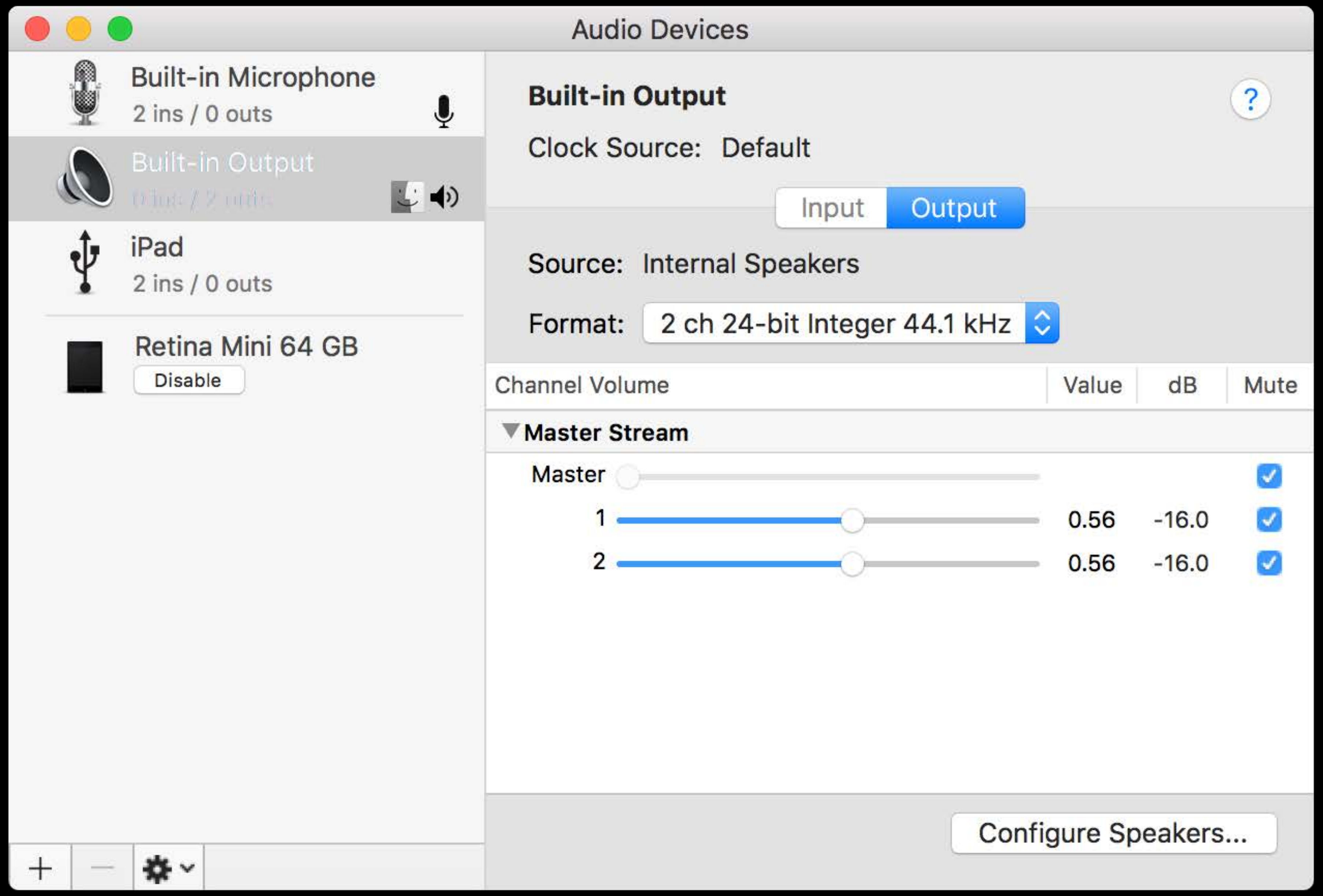
Torrey Holbrook Walker, Audio/MIDI Black Ops

Inter-Device Audio Mode

Record audio digitally via Lightning-to-USB cable

USB 2.0 audio class-compliant implementation

Available since El Capitan and iOS 9



Audio Devices

Built-in Microphone
2 ins / 0 outs

Built-in Output
0 ins / 2 outs

iPad
2 ins / 0 outs

Retina Mini 64 GB
Disable

Built-in Output
Clock Source: Default

Input | Output

Source: Internal Speakers
Format: 2 ch 24-bit Integer 44.1 kHz

Channel Volume	Value	dB	Mute
▼ **Master Stream**			
Master			✓
1	0.56	-16.0	✓
2	0.56	-16.0	✓

Configure Speakers...

IDAM

Inter-Device Audio Mode

IDAM

Inter-Device Audio

IDAM

Inter-Device Audio + MIDI

Inter-Device Audio + MIDI

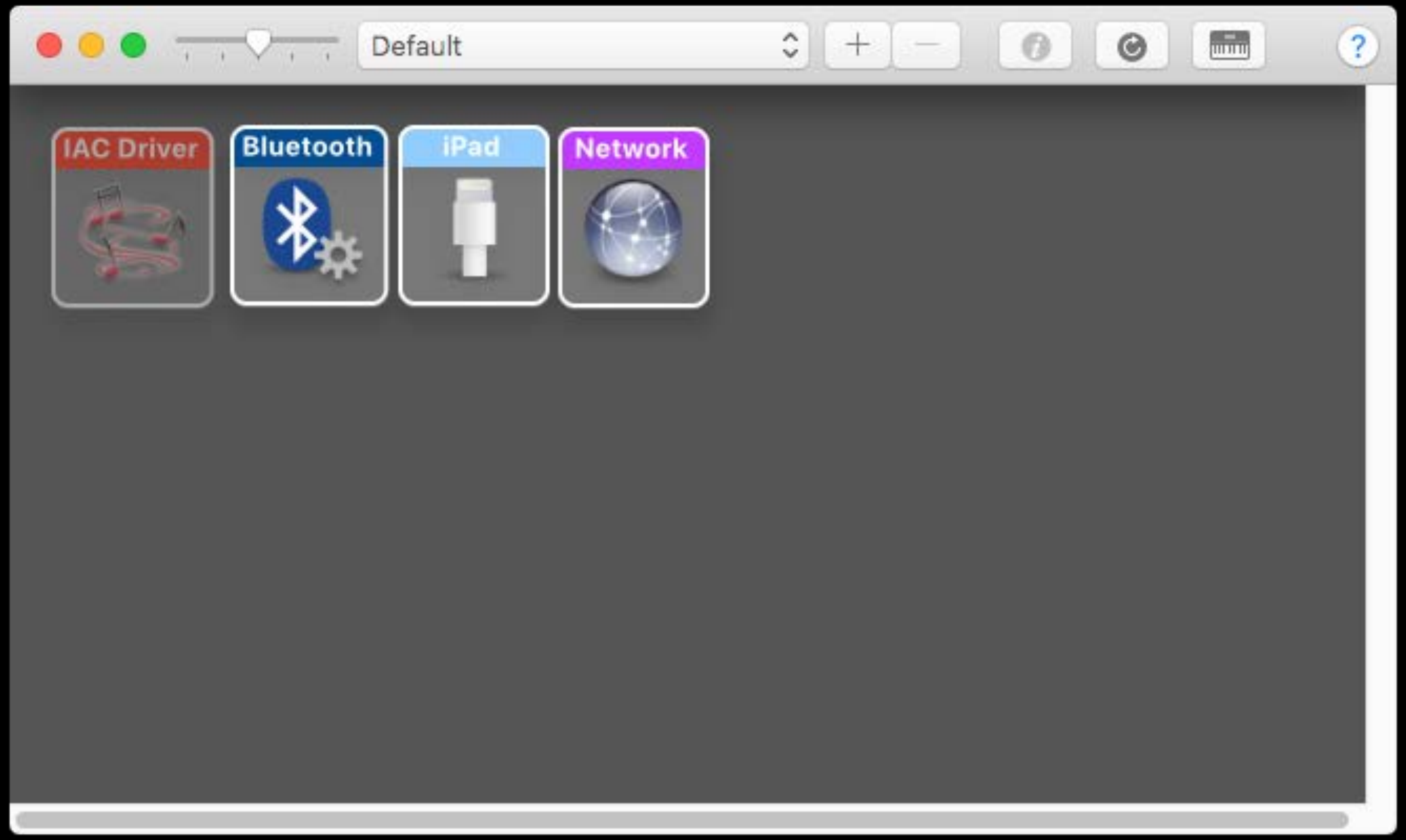
NEW

Send and receive MIDI via Lightning-to-USB cable

Class-compliant USB MIDI implementation

Requires iOS 11 and macOS El Capitan or later

Auto-enabled in IDAM configuration



Inter-Device Audio + MIDI

NEW

Device can charge and sync in IDAM configuration

Photo import and tethering are temporarily disabled

Audio device aggregation is ok

Use your iOS devices as a MIDI controllers, destinations, or both

Demo

MIDI using IDAM Configuration

Summary

AVAudioEngine - manual rendering

AVAudioSession - Airplay 2 support

watchOS - recording

AUAudioUnit - preferred view size, MIDI output

Other enhancements - audio formats (FLAC, Opus, HOA)

Inter-Device Audio and MIDI

More Information

https://developer.apple.com/wwdc17/501

Related Sessions

Introducing MusicKit	Grand Ballroom B	Tuesday 3:10PM
What's New in watchOS	Hall 2	Wednesday 9:00AM
Introducing AirPlay 2	Executive Ballroom	Thursday 4:10PM

Labs

Audio Lab	Technology Lab F	Tue 4:10PM–6:00PM
Airplay Lab	Technology Lab A	Wed 11:00AM-1:00PM
Audio Lab	Technology Lab G	Thu 1:00PM–3:00PM
Airplay Lab	Technology Lab A	Fri 9:00AM-11:00AM

WWDC17