# iPhone Application Programming
# Lecture 13: Multimedia and WatchOS

*Jan-Peter Krämer*
*Media Computing Group*
*RWTH Aachen University*

*Winter Semester 2015/2016*

http://hci.rwth-aachen.de/iphone

RWTH AACHEN UNIVERSITY

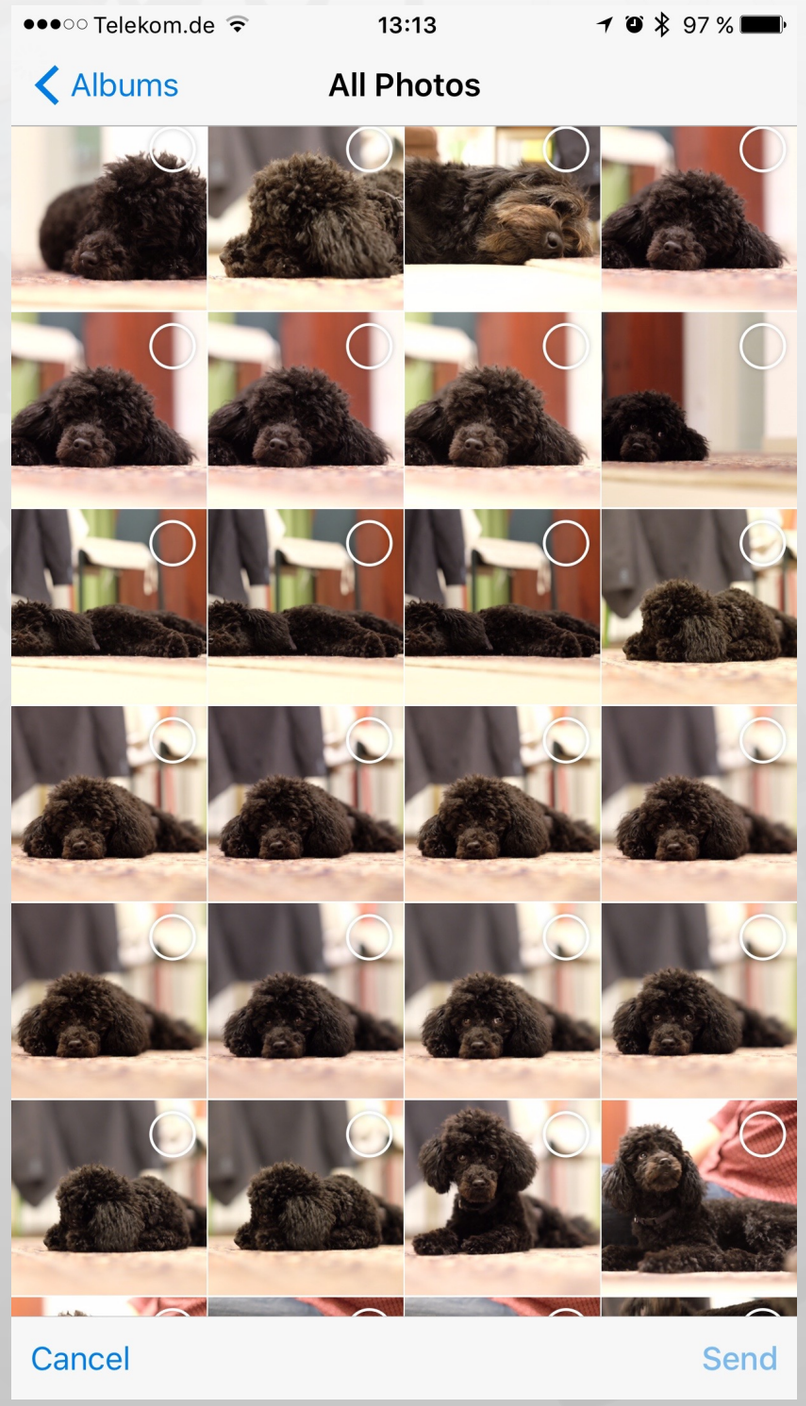
# Overview

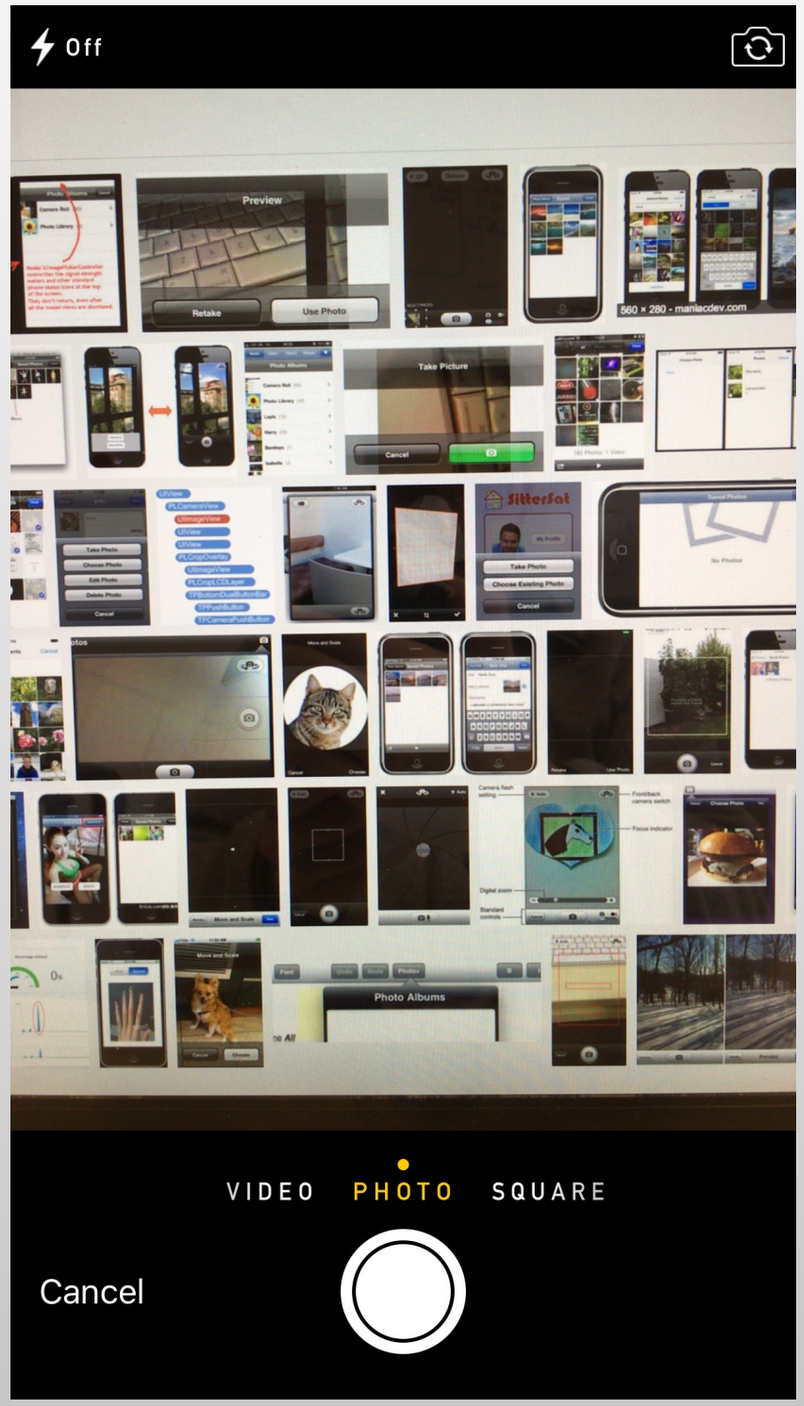
# Overview

| | Photos | Audio | Video |
|---|---|---|---|
| User Libraries | UIImagePicker Controller | iPod Library Access | UIImagePicker Controller |
| Create | UIImagePicker Controller | AVAudioRecorder | UIImagePicker Controller |
| View | UIImage | AVAudioPlayer SysSound | AVPlayer |
| Edit | Core Image | Core Audio OpenAL | Core Image |

# Images and Movies

# Select or Create



# UIImagePickerController

## 1. Check if source type is available

```
UIImagePickerController.isSourceTypeAvailable(.Camera)
```

## 2. Check which media types are available

```
UIImagePickerController.availableMediaTypesForSourceType(.Camera)
-> [“public.image", "public.movie"]
```

## 3. Setup an ImagePickerController for your needs

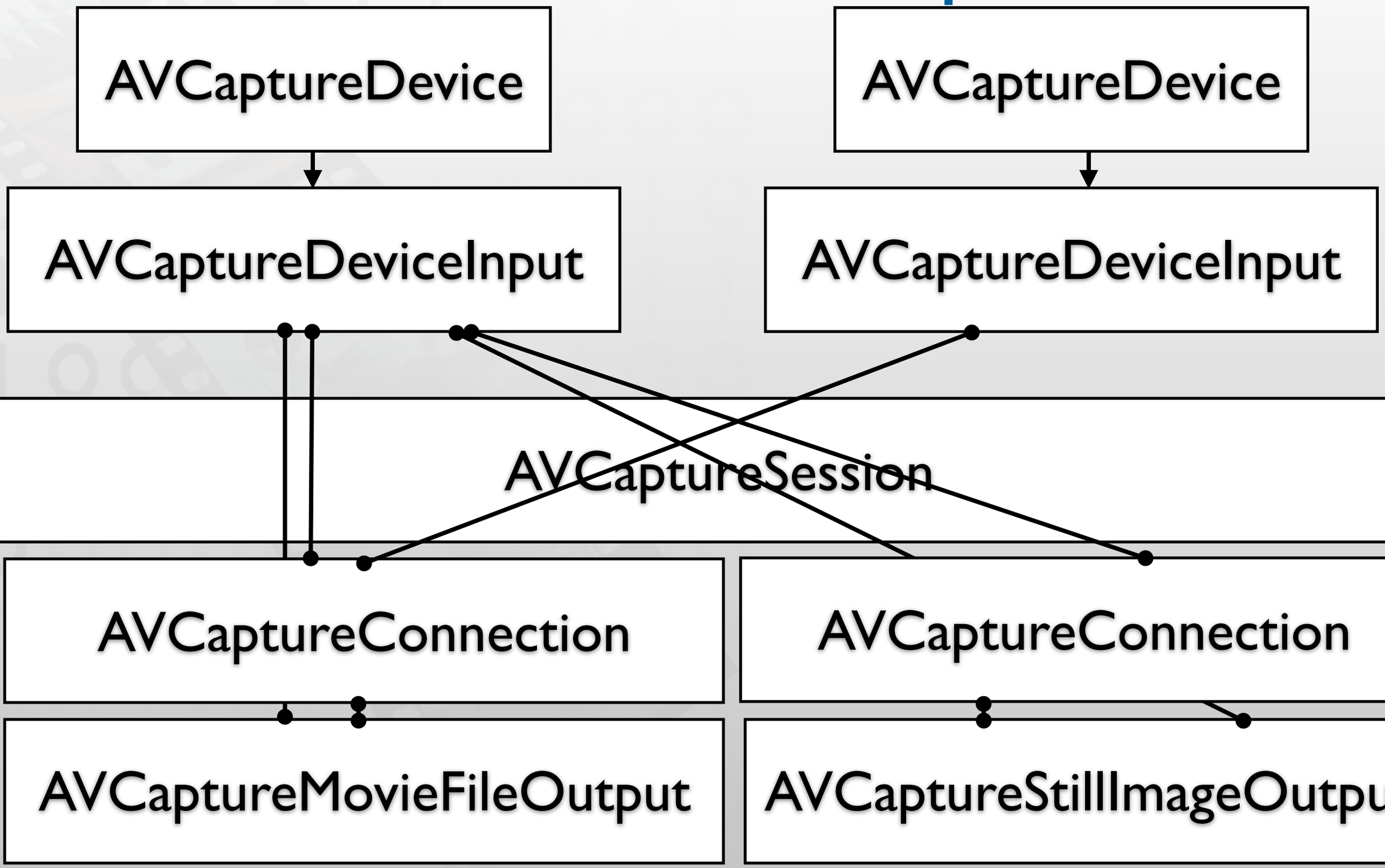
```
let imagePickerController = UIImagePickerController()
imagePickerController.sourceType = sourceType
imagePickerController.mediaTypes =
    [UIImagePickerControllerOriginalImage]
```

# UIImagePickerController

## 4. Get Results

```
func imagePickerController(picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String : AnyObject])
{
    if let image = info[UIImagePickerControllerOriginalImage] as? UIImage {
        self.imageView.image = image
    }
}
```

# AVFoundation Capture



# Video Playback

```
let playerViewController = AVPlayerViewController()
    playerViewController.player = AVPlayer(URL: NSURL(string: "")!)

    self.presentViewController(playerViewController, animated: true) {
        playerViewController.player!.play()
    }
```

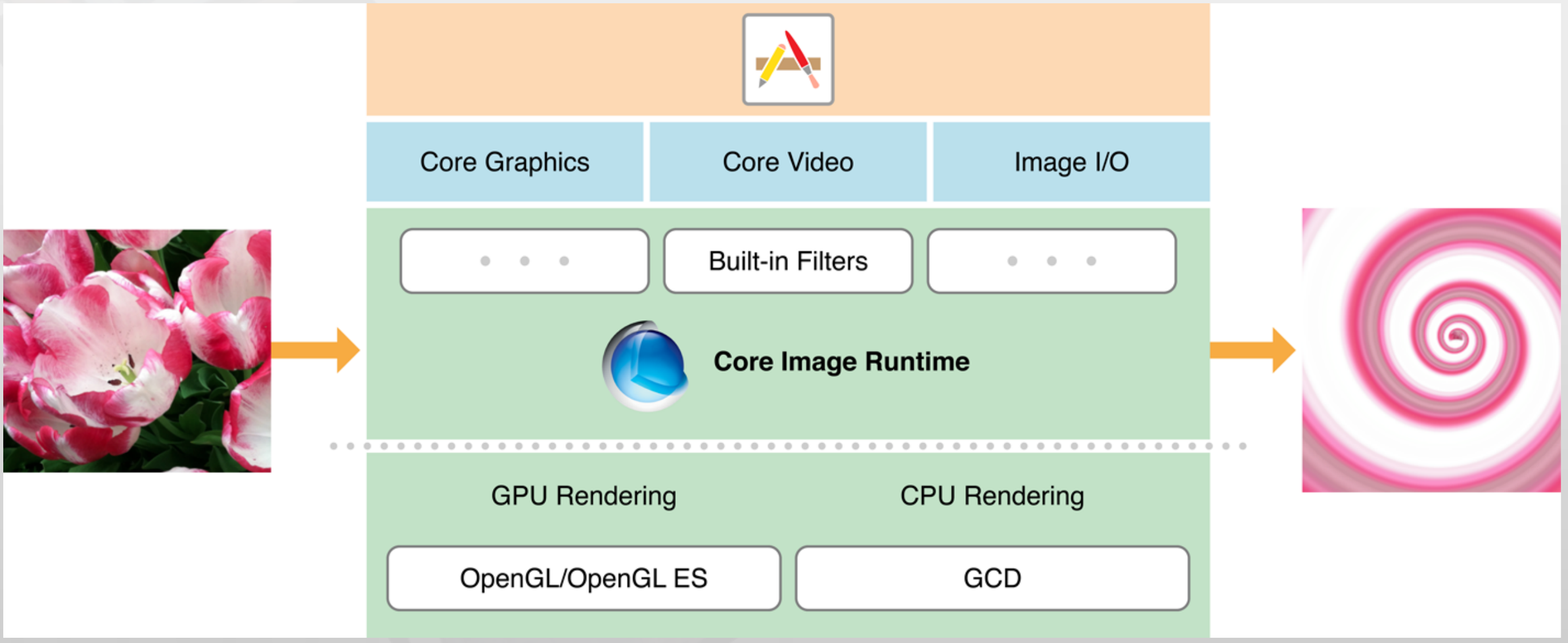
For PIP:

1. Link to iOS 9
2. Add Audio Background Mode
3. Configure AudioSession for Background Playback

For Airplay:

```
playerViewController.player!.allowsExternalPlayback = true
```

# Core Image


Core Graphics
Core Video
Image I/O
Built-in Filters
Core Image Runtime
GPU Rendering
CPU Rendering
OpenGL/OpenGL ES
GCD

# Core Image

## 1. Create a CIContext

```
let context = CIContext()
```

## 2. Load an Image

```
let ciimage = CIImage(image: image)
```

## 3. Setup Filters

```
let filter = CIFilter(name: “CIExposureAdjust")
filter.setValue(0.3, forKey: kCIInputEVKey)
```

# Core Image

## 4. Setup filter chain

```
filter.setValue(ciimage, forKey: kCIInputImageKey)
filter2.setValue(filter.outputImage, forKey: kCIInputImageKey)

let result = filter2.valueForKey(kCIOutputImageKey) as! CIImage
```

## 5. Transform to UIImage

```
let cgimage = context.createCGImage(result, fromRect: result.extent)
let image = UIImage(CGImage: cgimage)
```

# Core Image for Video

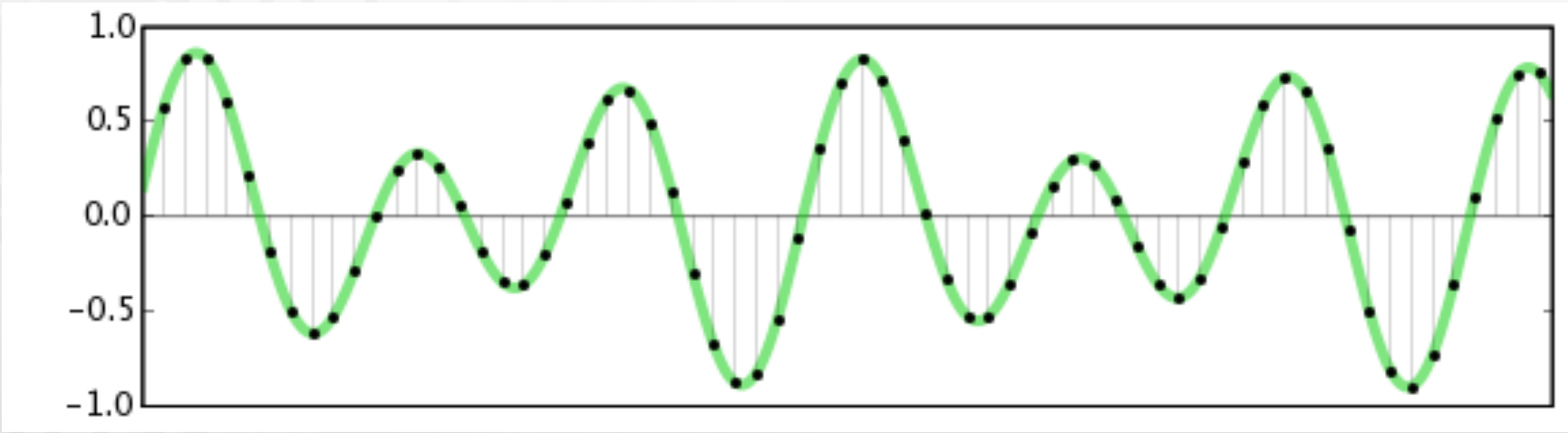
- Use AVPlayer for decoding
- Output to CVPixelBuffer
- Convert to CIImage
- Render CIImage in an OpenGL ES-backed CIContext

Demo

https://www.objc.io/issues/23-video/core-image-video/

# Audio Basics

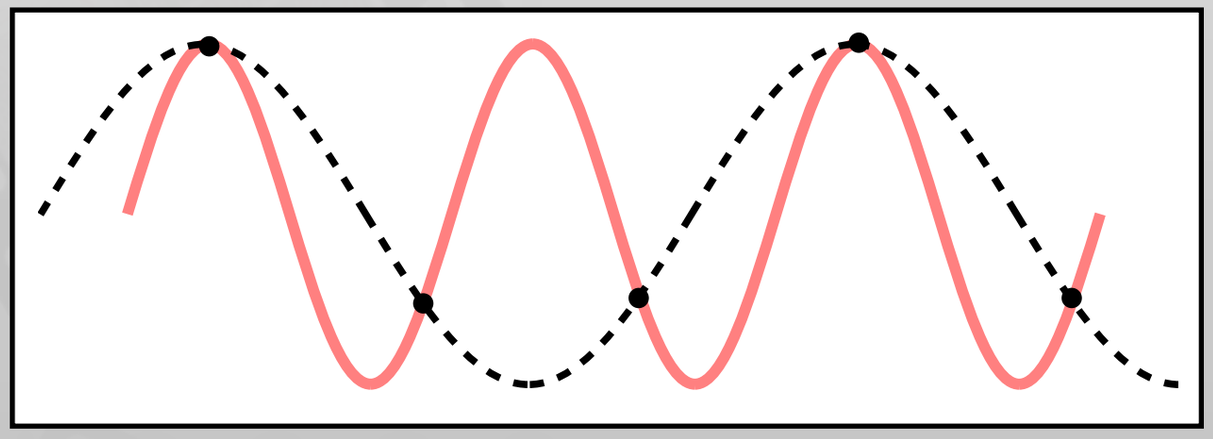
# Digital Audio




X Axis Resolution: Sample Rate => min. $2f_{max}$

Y Axis Resolution: Bit Depth

typical: 44.1kHz, 16bit



# Audio Codecs

- Hardware codecs
  - Excellent performance (no CPU load)
  - No simultaneous playback
- Software Codecs
  - Decoding (playback) & encoding (recording)
  - Simultaneous playback

# Audio Codecs

| Audio decoder/playback format | HW-assisted decoding | SW-based decoding |
|---|---|---|
| AAC (MPEG-4 Advanced Audio Coding) | Yes | Yes |
| ALAC (Apple Lossless) | Yes | Yes |
| HE-AAC (MPEG-4 High Efficiency AAC) | Yes | |
| iLBC (internet Low Bitrate Codec, a format for speech) | | Yes |
| IMA4 (IMA/ADPCM) | | Yes |
| Linear PCM (uncompressed, linear pulse-code modulation) | | Yes |
| MP3 (MPEG-1 audio layer 3) | Yes | Yes |
| µ-law and a-law | | Yes |

# Audio Sessions

- Define the audio behavior of your app
- Express your app's audio intention
  - Should the audio be muted with the Ring/Silent switch?
  - Should the iPod playback continue when you app starts?
- Help to manage interruptions
- Route change notifications

# Default Audio Session

- Playback is enabled and recording is disabled.
- The silence switch mutes your audio
- Your audio is silenced when the screen is locked
- Already playing audio (e.g., iPod) is silenced
- However, do not ship your app with the default audio session.

# Audio Session Categories

| Category | Silenced by the Ring/Silent switch and by screen locking | Allows audio from other applications | Allows audio input (recording) and output (playback) |
|---|---|---|---|
| AVAudioSessionCategoryAmbient | Yes | Yes | Output Only |
| AVAudioSessionCategorySoloAmbient | Yes | No | Output Only |
| AVAudioSessionCategoryPlayback | No | No (default) but override switch | Output Only |
| AVAudioSessionCategoryRecord | No | No | Input Only |
| AVAudioSessionCategoryPlayAndRecord | No | No (default) but override switch | Input and output |
| AVAudioSessionCategoryAudioProcessing | — | No | No input or output |

# Audio Sessions

```
do {
    let session = AVAudioSession.sharedInstance()
    try session.setCategory(AVAudioSessionCategoryPlayback)
    try session.setActive(true)
} catch let e as NSError {
    print(e.localizedDescription)
}
```

# Handling Audio Interruptions

- Interruption starts
  - Check wether resumption of audio process is supported
  - Save state and context
  - Update user interface
- Interruption ends
  - Restore state and context
  - Reactivate audio session
  - Update user interface

# Handling Interruptions

```
// Register for the interruption notifications

    NSNotificationCenter.defaultCenter().addObserver(self, selector:
"handleInterruption:",
name: AVAudioSessionInterruptionNotification,
object: AVAudioSession.sharedInstance())
```

# Handling Interruptions

```
func handleInterruption(notification: NSNotification) {
    if let interruptionTypeInt = notification.userInfo?[AVAudioSessionInterruptionTypeKey]
as? UInt,
        interruptionType = AVAudioSessionInterruptionType(rawValue: interruptionTypeInt)
    {
        switch interruptionType {
        case .Began:
            print("Interruption Began")
        case .Ended:
            print("Interruption Ended")
        }
    }
}
```

# Reacting to Route Changes

- A route consists of ports
- Input ports (for example):
  - Wired microphone
- Output ports (for example):
  - Built-in speaker
  - Bluetooth A2DP output
- If the user plugs/unplugs a wired headset, the route changes and the app should react accordingly.

# Reacting to Route Changes

```
Old outputs: (
"<AVAudioSessionPortDescription: 0x17800edb0, type = Speaker;
name = Speaker; UID = Speaker; selectedDataSource = (null)>"
)
New outputs : (
"<AVAudioSessionPortDescription: 0x17800ee50, type = Headphones;
name = Headphones; UID = Wired Headphones; selectedDataSource =
(null)>"
```

# Audio Routes

- Route can be overridden
  - Output should remain on speaker
  - Input should be the device microphone (not the headset)
  - Input should be one specific microphone (on multi-microphone devices)

# AVAudioPlayer

- Plays any supported file format
- Play sounds of arbitrary length
- Files or memory buffers
- Loop
- Simultaneous playback
- Volume level control
- Seek to a point in an audio file
- Audio power data

# Audio Playback

```
player = try! AVAudioPlayer(contentsOfURL: url)
player.prepareToPlay()

self.player.play()
```

# System Sounds

- File requirements
  - No longer than 30s
  - Linear PCM or IMA4
  - .caf, .aif, or .wav file

- Capabilities
  - Current volume, no programmatic control
  - Playback starts immediately
  - Only one sound at a time

# System Sound

```
//Setup
let url = NSBundle.mainBundle().URLForResource("tap", withExtension: "aif")!
AudioServicesCreateSystemSoundID(url, &systemSound)

//Play
AudioServicesPlaySystemSound(systemSound)

//Play and/or vibrate
AudioServicesPlayAlertSound(systemSound)

//Vibrate only
AudioServicesPlaySystemSound(kSystemSoundID_Vibrate)
```

# AVAudioRecorder

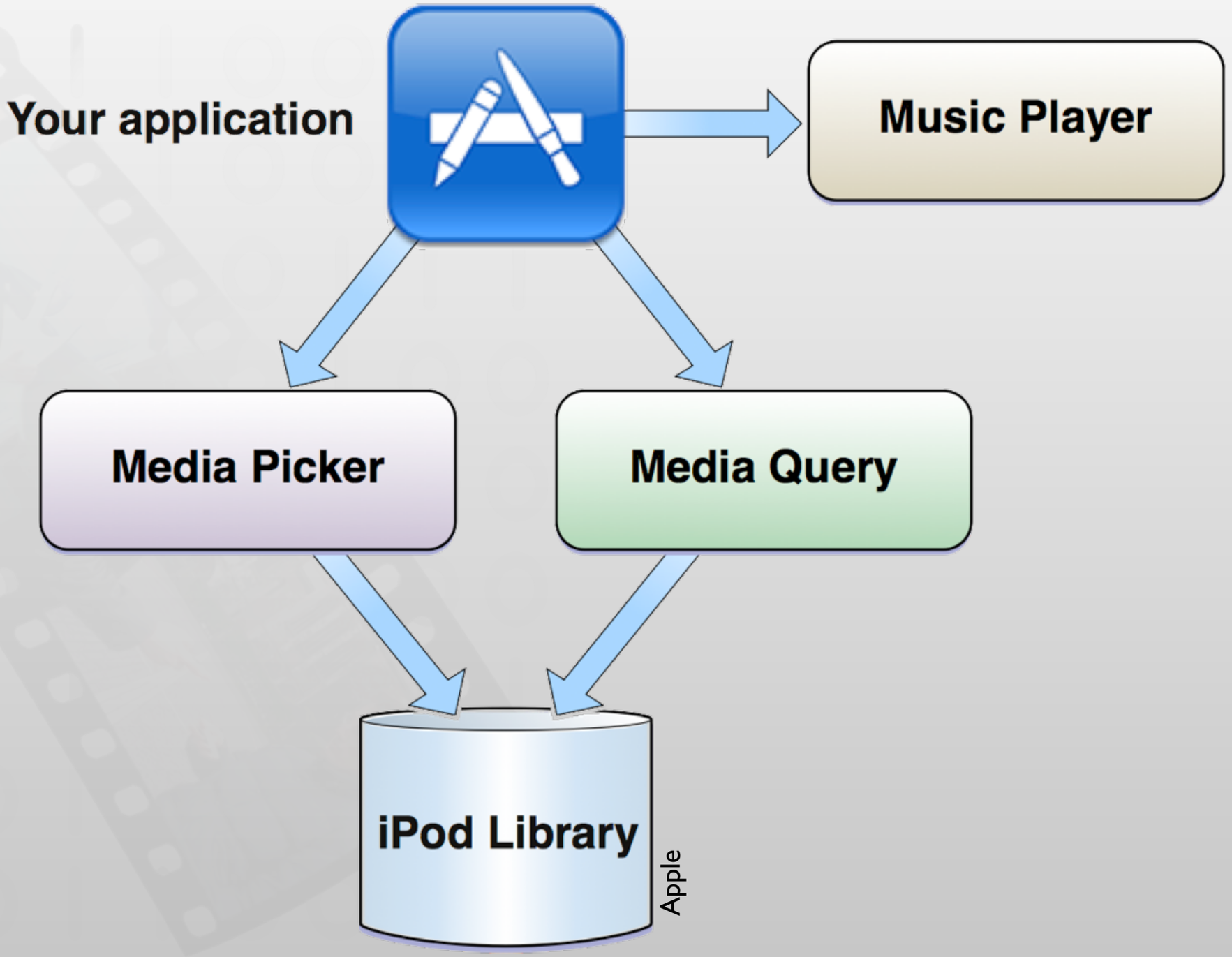
```
let session = AVAudioSession.sharedInstance()
let path = NSTemporaryDirectory() + "sound.caf"
self.recordURL = NSURL(fileURLWithPath: path)

try! session.setCategory(AVAudioSessionCategoryPlayAndRecord)
try! session.setActive(true)
```

# AVAudioRecorder

```
session.requestRecordPermission { (success) -> Void in
    if (!success) { return }
    if let existingURL = self.recordURL {
        let settings: [String:AnyObject] = [
            AVSampleRateKey: 44100.0,
            AVFormatIDKey: Int(kAudioFormatAppleLossless),
            AVNumberOfChannelsKey: 1,
            AVEncoderAudioQualityKey: AVAudioQuality.Max.rawValue
        ]

        if let recorder = try? AVAudioRecorder(URL: existingURL, settings: settings) {
            self.recorder = recorder

            recorder.delegate = self
            recorder.prepareToRecord()
            recorder.record()
        }
    }
}
```

# iPod Library Access

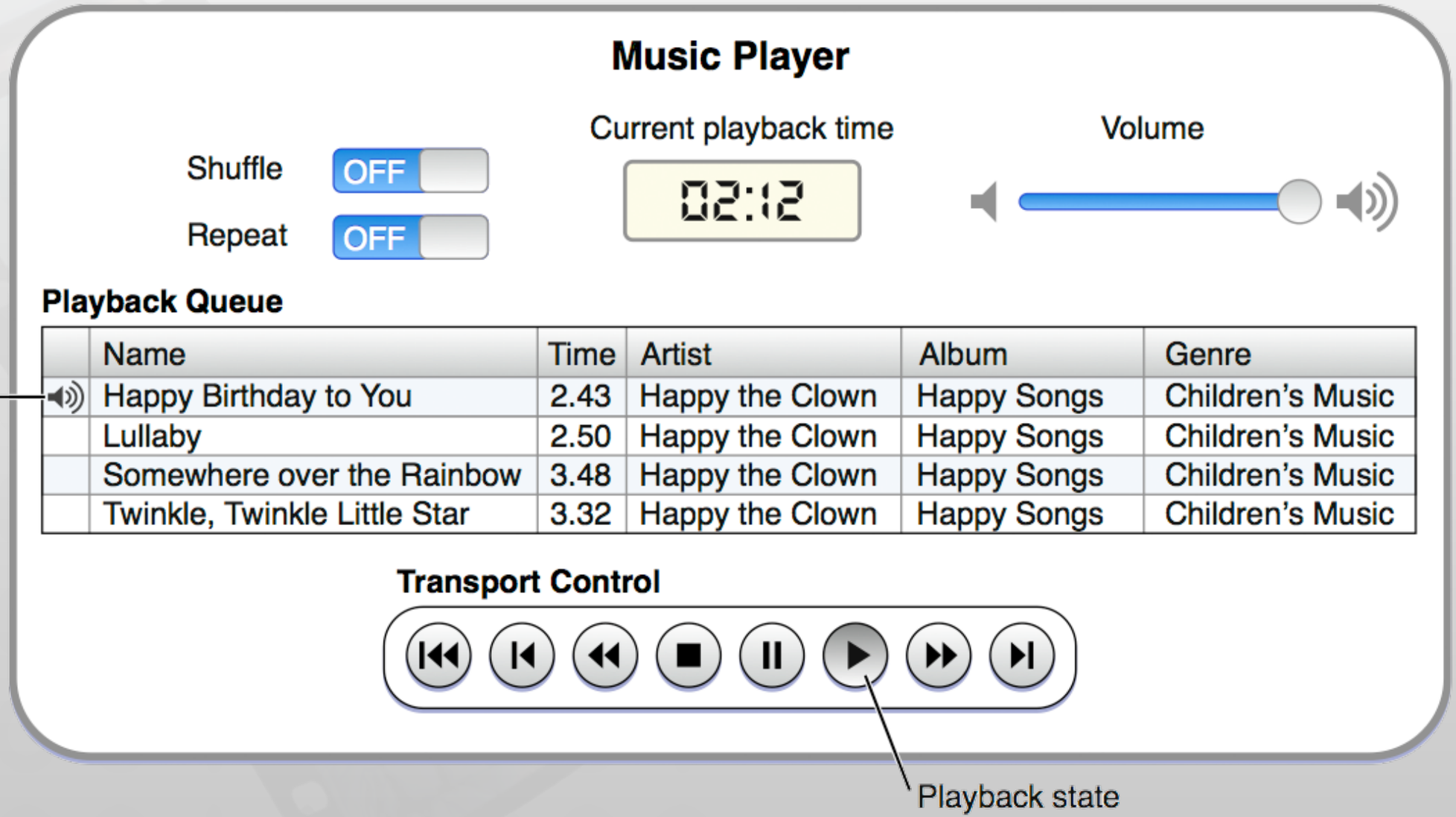
# iPod Library Access



# Music Player Terminology

- Music player is an object that plays media items
- A playback queue is a list of media items to play
- A media item is a song, audio podcast or an audio book
- The set of media items is called the iPod library

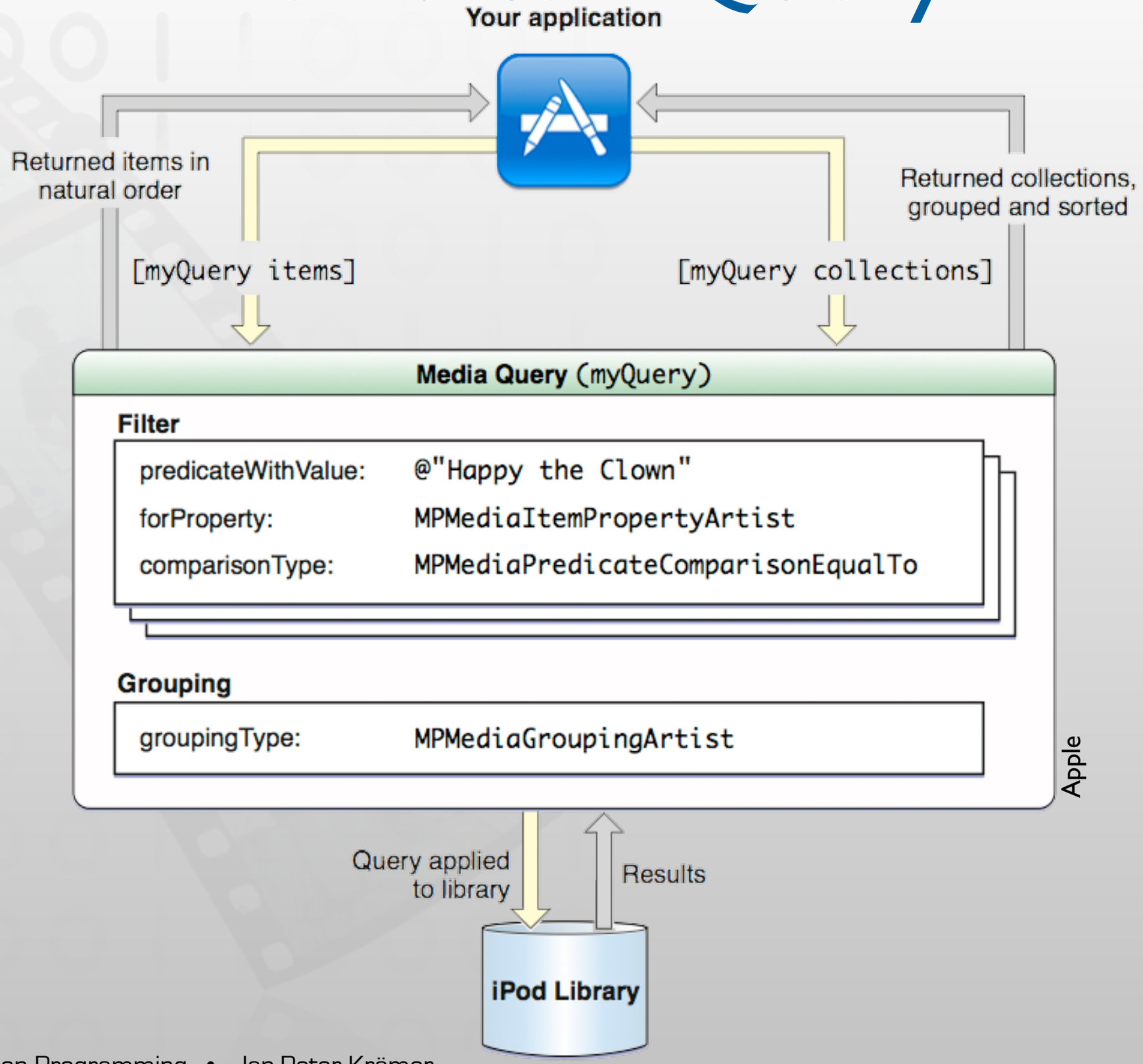
# Music Players

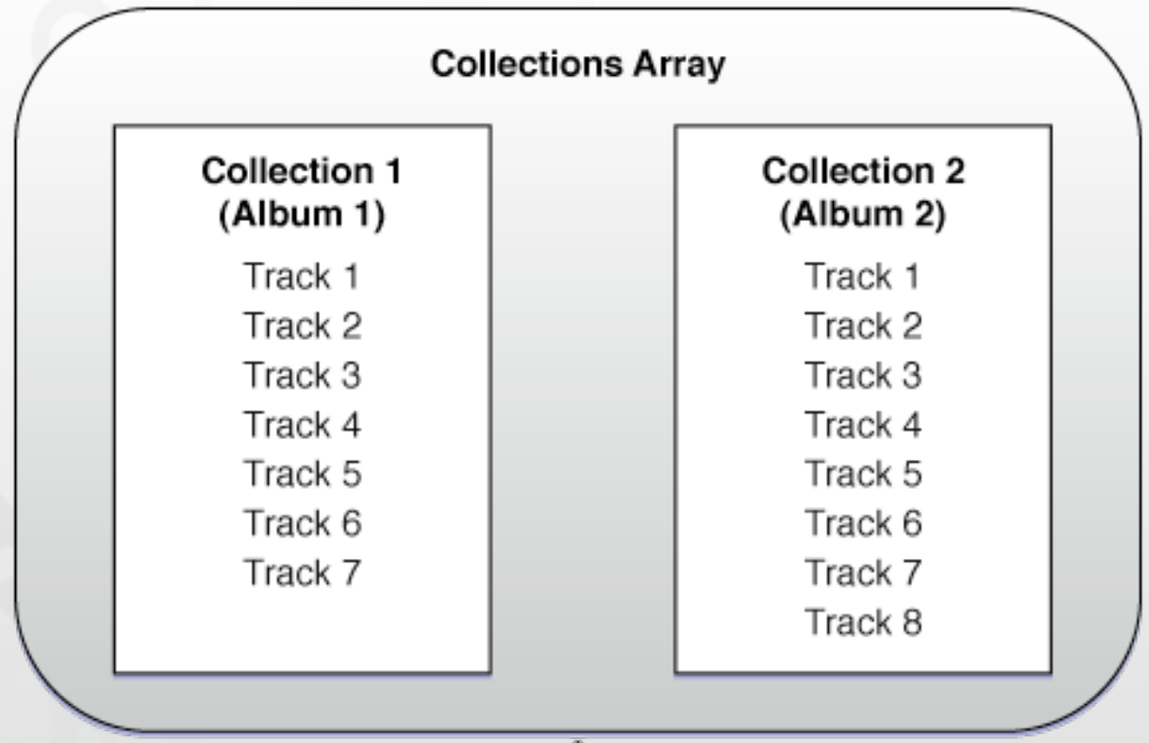


# Music Player Basic Example

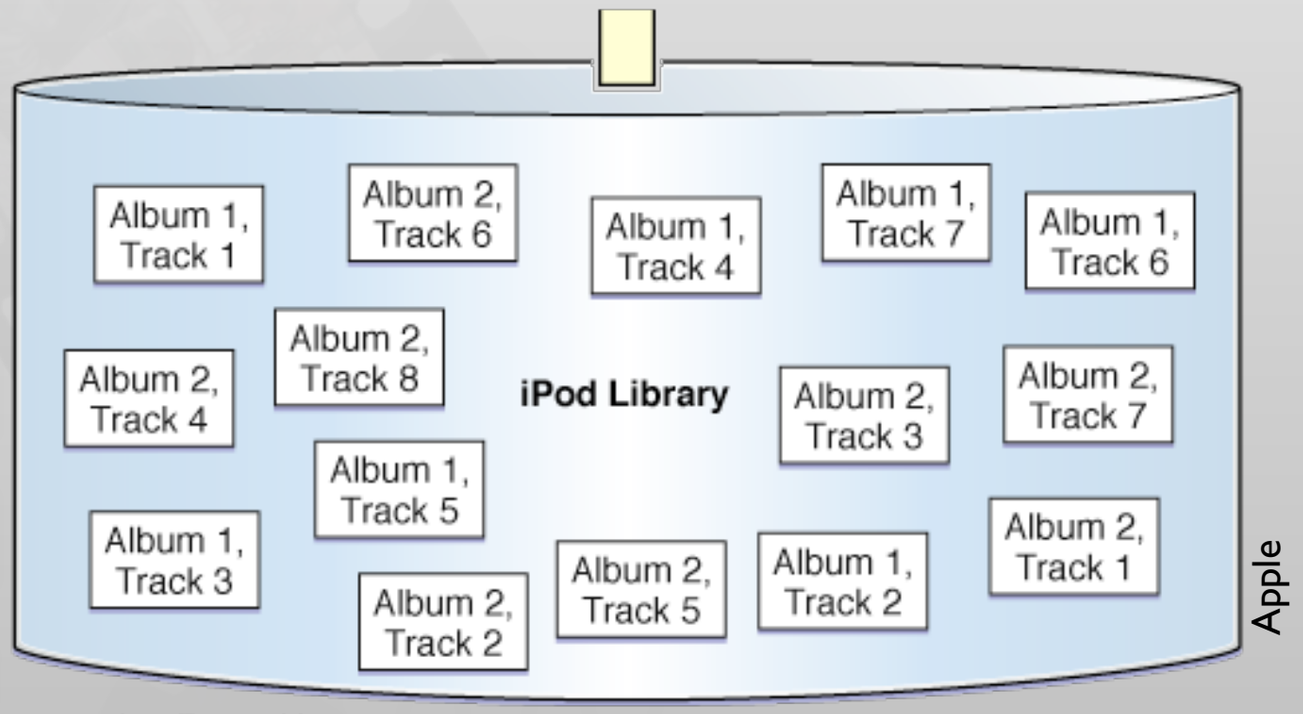
```
@IBAction func pickTracks(sender: AnyObject) {
    let picker = MPMediaPickerController(mediaTypes: .Music)
    picker.allowsPickingMultipleItems = true
    picker.delegate = self
    self.presentViewController(picker, animated: true, completion: nil)
}

extension ViewController : MPMediaPickerControllerDelegate {
   func mediaPicker(mediaPicker: MPMediaPickerController, didPickMediaItems
mediaItemCollection: MPMediaItemCollection) {
    self.dismissViewControllerAnimated(true, completion: nil)

    let player = MPMusicPlayerController.systemMusicPlayer()
    player.setQueueWithItemCollection(mediaItemCollection)

    player.play()
   }
}
```

# Media Item Query






```
MPMediaQuery *myQuery = [[MPMediaQuery] alloc] init];
[myQuery setGroupingType: MPMediaGroupingAlbum];
NSArray *myCollections = [myQuery collections];
```



# Advanced Audio

# AVAudioEngine

- Powerful API for real-time audio
- Read and write audio files
- Play and record using files and buffers
- Audio processing
- Spatial audio

# AVAudioEngine

- Works on a processing graph
- The graph consists of connected AVAudioNodes
  - AVAudioOutputNode
  - AVAudioMixerNode
  - AVAudioPlayerNode
  - AVAudioUnit
- The engine sets up the graph by connecting the nodes
- Start/stop the engine to control playback

# AVAudioNode

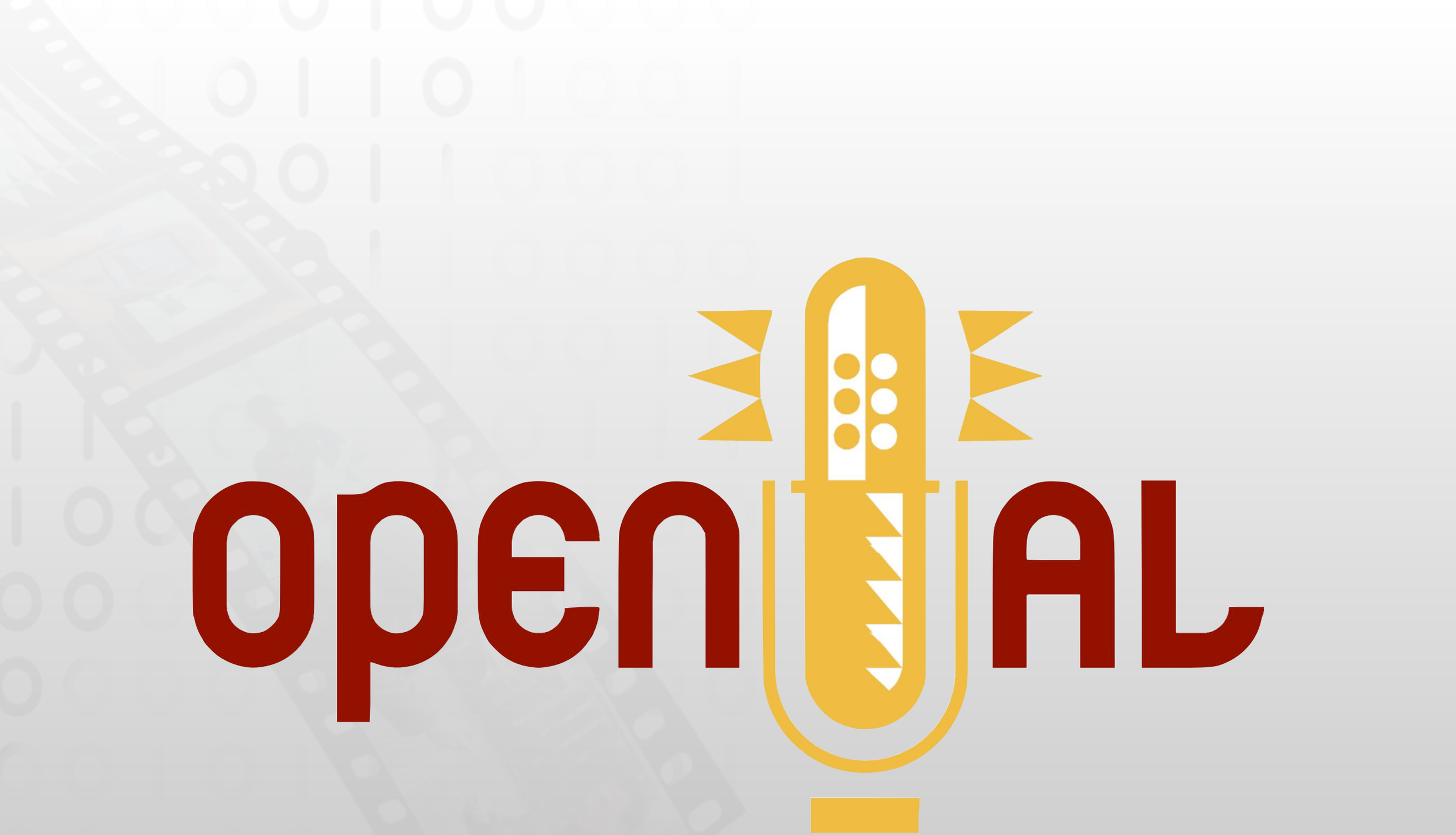
- Source
  - Player, microphone
- Process
  - Mixer, effect
- Destination
  - Speaker

# Effect nodes

- Effect
  - Delay
  - Distortion
  - EQ
  - Reverb
- Time effects
  - Varispeed
  - TimePitch

# AVAudioEnvironmentNode

- Rendering algorithms
  - Equal power panning
  - Spherical head
  - HRTF
  - Sound field
- Input
  - Position
  - Orientation
  - Obstruction, occlusion
  - Distance attenuation
  - Reverberation


openAL

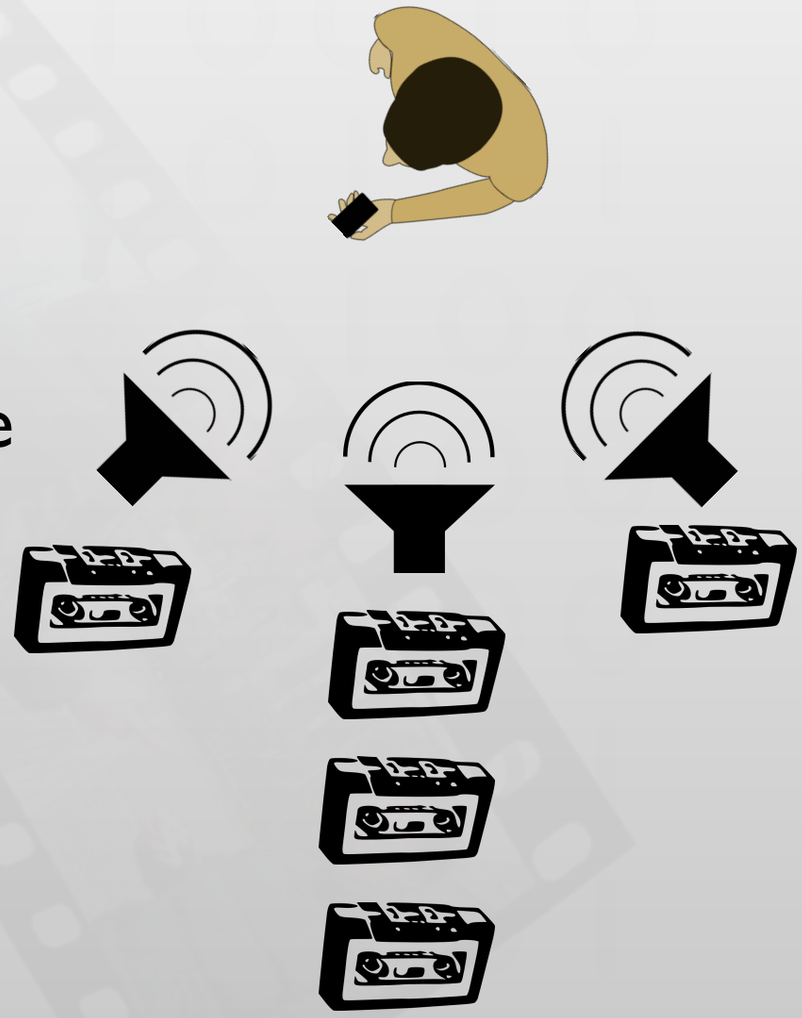
# Demo

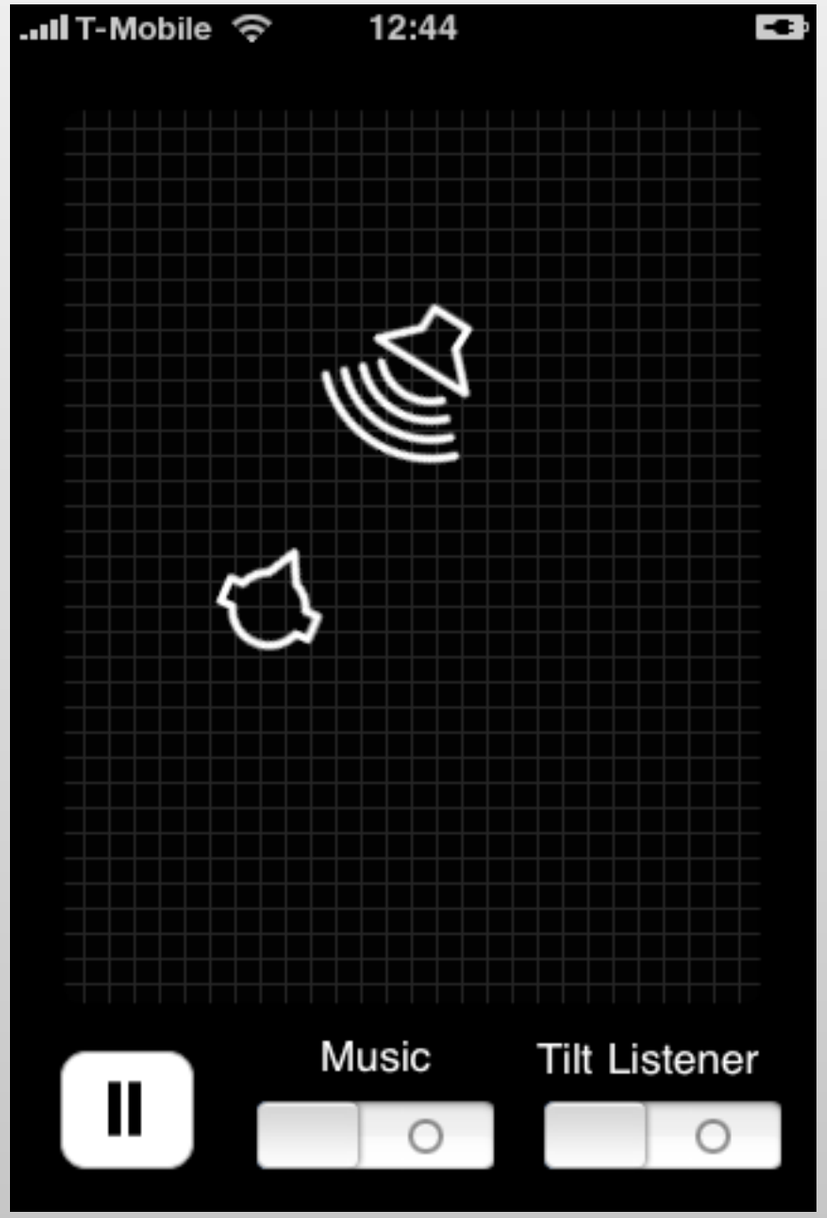
# OpenAL

- Spatial audio rendering framework
- Similar to OpenGL
- Define the scene, let the system do the rest
- Simultaneous playback of several sources

# OpenAL Primitives

- Listener
- Sound Source
- Sound Buffer



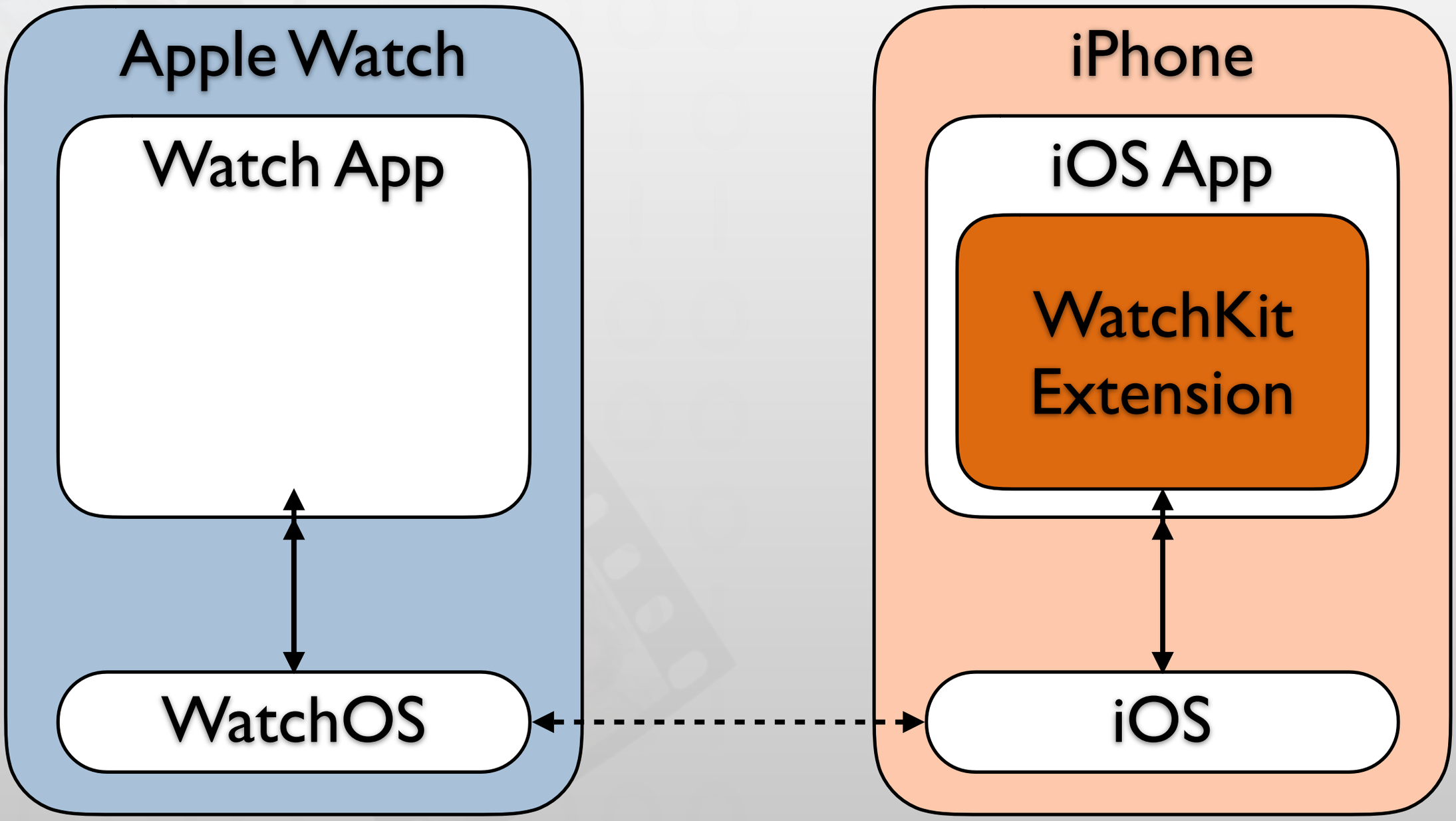


# More Audio

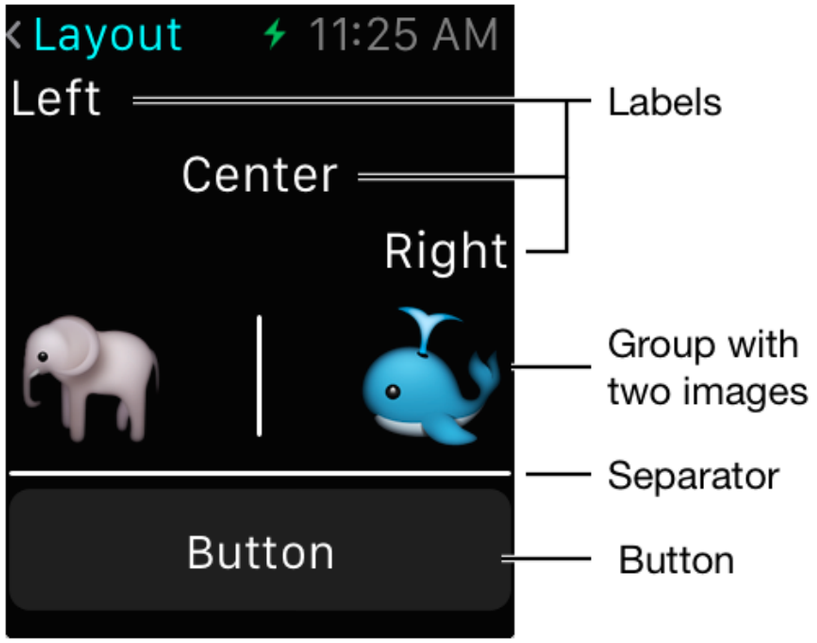
- Audio Units
- Inter-App Audio
- Audio queue services
- Core MIDI

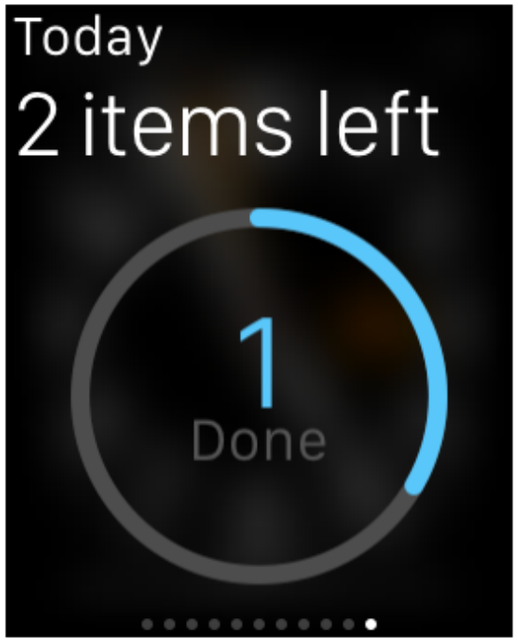
# WatchOS

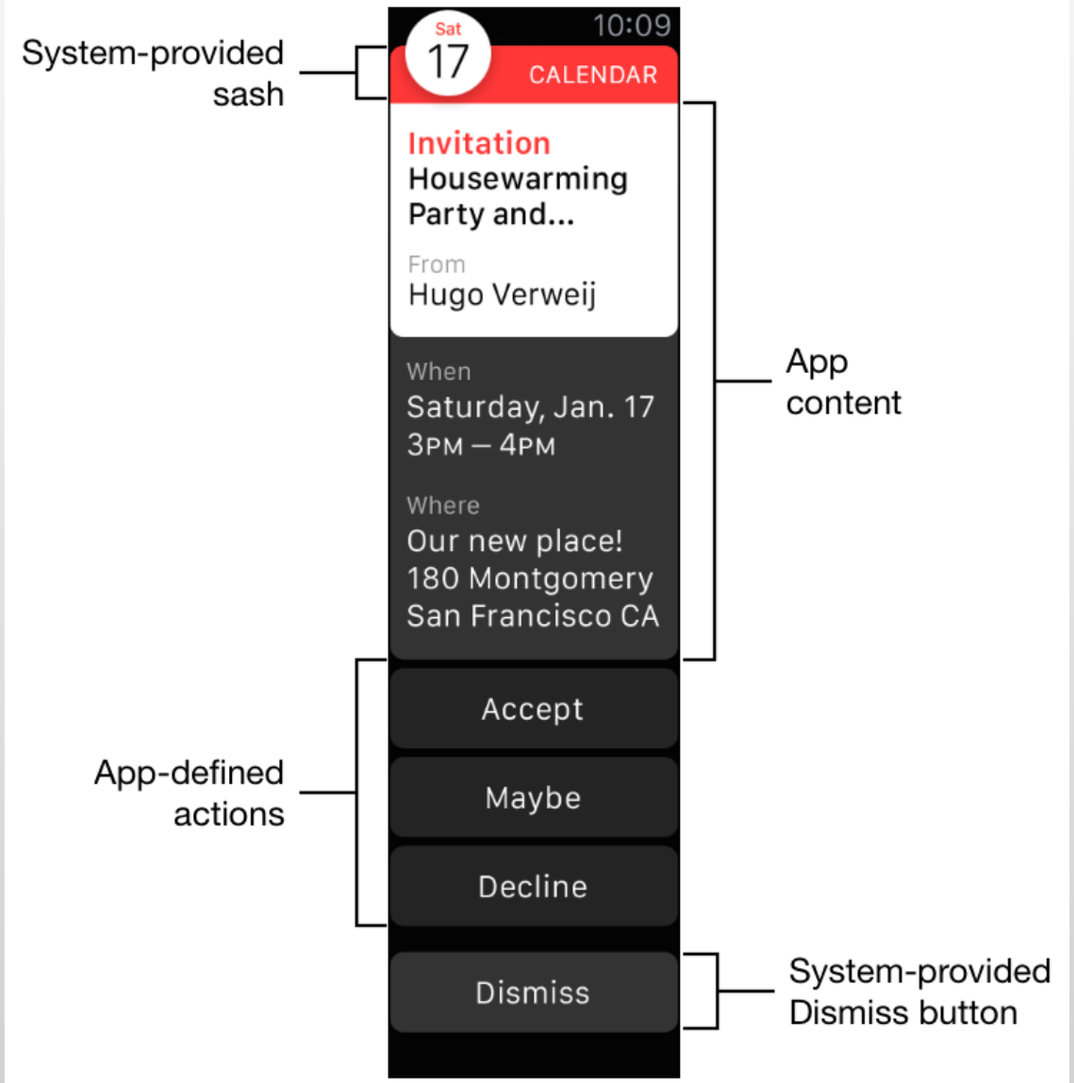
# Architecture



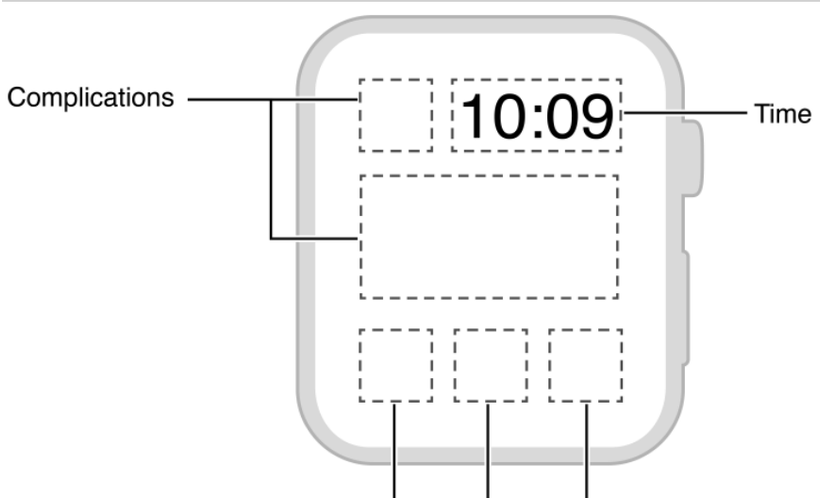
# Parts of a WatchKit App







# Communicate with iOS

- No Background execution
- Permission requests sometimes need to be answered on the iPhone
- Handoff long-running tasks to the iPhone

## WatchConnectivity

# Watch Connectivity

## 1. Check if WC is available

```
WCSession.isSupported()
```

## 2. Activate Session

```
let session = WCSession.defaultSession()
session.delegate = self
session.activateSession()
```

# Watch Connectivity

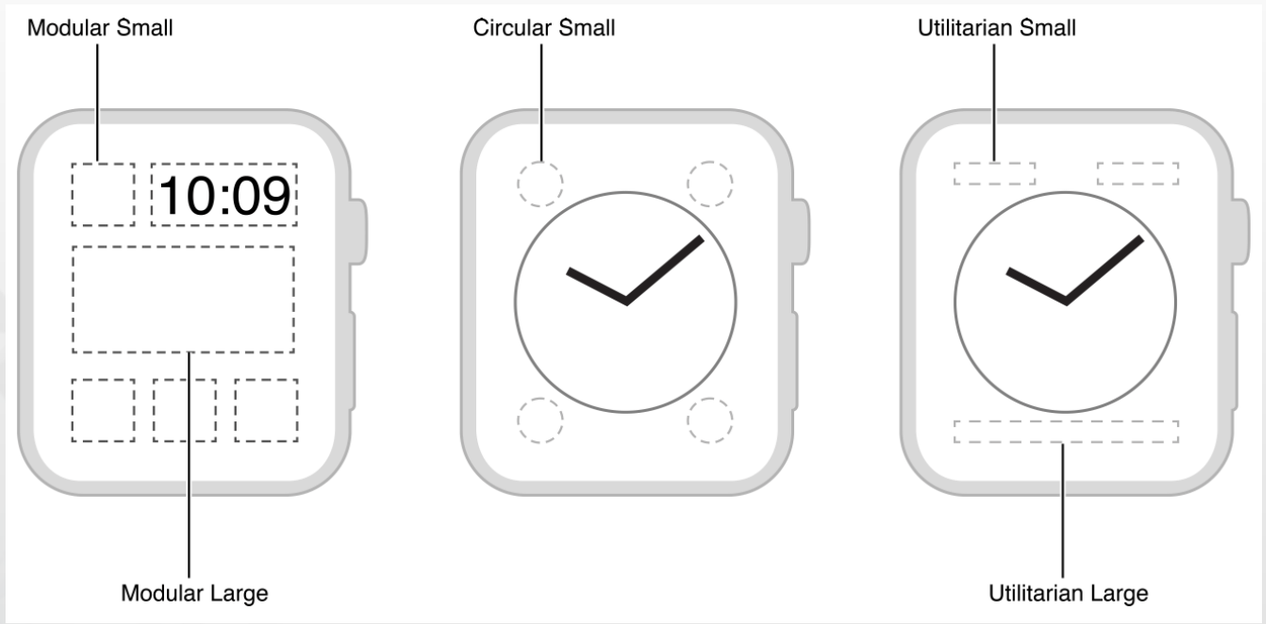
## Send Messages

```
session.sendMessage(["info": "Hello"], replyHandler: {
    (response) -> Void in
    }, errorHandler: { (ErrorType) -> Void in
    })
```

## Background transfer

```
session.transferUserInfo([String: AnyObject])
session.transferFile(url, metadata: [String: AnyObject])
session.transferCurrentComplicationUserInfo([String: AnyObject])
```

# Complications




- Implement one class conforming to CLKComplicationDataSource
- Implement at least:

```
func getCurrentTimelineEntryForComplication(complication: CLKComplication,
withHandler handler: (CLKComplicationTimelineEntry?) -> Void)
func getSupportedTimeTravelDirectionsForComplication(complication: CLKComplication,
withHandler handler: (CLKComplicationTimeTravelDirections) -> Void)
func getTimelineEntriesForComplication(complication: CLKComplication, afterDate date:
NSDate, limit: Int, withHandler handler: ([CLKComplicationTimelineEntry]?) -> Void)
```

# Complications show Entries on a Timeline


⚽ 11:00AM
Haight – Twin Peaks
Group D

⚽ 1:30PM
Bernal – Presidio
Group C

⚽ 4:00PM
SOMA – Castro
Group A

⚽ 6:00PM
Sunset – Marina
Group B

# Complications show Entries on a Timeline


eaks

⚽ 1:30PM
Bernal – Presidio
Group C

⚽ 6:00PM
Sunset – Marina
Group B

# Complications show Entries on a Timeline

# Demo

http://www.raywenderlich.com/117298/watchos-2-tutorial-part-3-animation

# Summary

- WatchOS apps comprised of
  - App
  - Glances
  - Notifications
  - Complications
- Further Reading
  - WatchKit Programming Guide
  - watchOS 2 Transition Guide
  - Watch Connectivity Framework Reference