iPhone Application Programming
Lecture 13: Audio

Florian Heller
Media Computing Group
RWTH Aachen University

Winter Semester 2013/2014

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

RWTH AACHEN UNIVERSITY

Review

- List all touch phases of an UITouch object and explain briefly what they mean.
- What are possible event types of UIEvent objects?
- Illustrate, how you can implement your own gesture recognizer.

Overview

- Audio
  - Audio basics
  - iPod library access
  - AVFoundation
  - OpenAL
- Video

Audio Basics

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

Two APIs

- AVAudioSession (Obj-C)
  - Get system information on

    Output route

    Hardware capabilities
  - Easy interruption handling
- Audio session services (C)
  - Modify audio session behavior
  - Use callbacks for interruption handling

One API

- AVAudioSession (Obj-C)
  - Get system information on

    Output route

    Hardware capabilities
  - Easy interruption handling
  - Modify audio session behavior
- Audio session services (C)
  - Deprecated in 

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
- AVAudioSessionCategorySoloAmbient

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

```
NSError *setCategoryErr = nil;
NSError *activationErr = nil;

// Set the category
[[AVAudioSession sharedInstance]
                 setCategory:AVAudioSessionCategoryPlayback
                       error:&setCategoryErr];
// and activate it
[[AVAudioSession sharedInstance]
                         setActive:YES
                             error:&activationErr];

// ... deactivate it later on
[[AVAudioSession sharedInstance]
                         setActive:NO
                             error:&activationErr];
```

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: @selector(handleInterruption:)
                       name: AVAudioSessionInterruptionNotification
                     object: [AVAudioSession sharedInstance]];
```

Handling Interruptions

```
- (void)handleInterruption:(NSNotification*)intNotification
{
   AVAudioSessionInterruptionType intType =
[intNotification.userInfo[AVAudioSessionInterruptionTypeKey] intValue];

   if (intType == AVAudioSessionInterruptionTypeBegan)
   {
       // Save state to resume later
       NSLog(@"Interruption began");
   }
   else if (intType == AVAudioSessionInterruptionTypeEnded)
   {
       // Interruption ended
       NSLog(@"Interruption ended");

       // Should we resume?
       AVAudioSessionInterruptionOptions options =
[intNotification.userInfo[AVAudioSessionInterruptionOptionKey] intValue];
       if (options == AVAudioSessionInterruptionOptionShouldResume)
       {
          NSLog(@"Please resume");
       }
   }
}
```

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.

Route Change Reasons

- AVAudioSessionRouteChangeReasonUnknown
- AVAudioSessionRouteChangeReasonNewDeviceAvailable
- AVAudioSessionRouteChangeReasonOldDeviceUnavailable
- AVAudioSessionRouteChangeReasonCategoryChange
- AVAudioSessionRouteChangeReasonOverride
- AVAudioSessionRouteChangeReasonWakeFromSleep
- AVAudioSessionRouteChangeReasonNoSuitableRouteForCategory
- AVAudioSessionRouteChangeReasonRouteConfigurationChange

Reacting to Route Changes

```
// Register for the interruption notifications

[[NSNotificationCenter defaultCenter]
                addObserver: self
                   selector: @selector(handleRouteChange:)
                       name: AVAudioSessionRouteChangeNotification
                     object: [AVAudioSession sharedInstance]];
```

Reacting to Route Changes

```
- (void)handleRouteChange:(NSNotification*)routeNotification
{
   AVAudioSessionRouteChangeReason reason =
[routeNotification.userInfo[AVAudioSessionRouteChangeReasonKey] intValue];
if (reason == AVAudioSessionRouteChangeReasonNewDeviceAvailable)
{
   AVAudioSessionRouteDescription *oldRoute =
routeNotification.userInfo[AVAudioSessionRouteChangePreviousRouteKey];
   AVAudioSessionRouteDescription *currentRoute = [[AVAudioSession
sharedInstance] currentRoute];
   NSLog(@"Something turned up");
   NSLog(@"Old outputs: %@\nNew outputs : %@", oldRoute.outputs,
currentRoute.outputs);

}
else if (reason == AVAudioSessionRouteChangeReasonOldDeviceUnavailable)
{
   NSLog(@"Something disappeared");
}
}
```

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)

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

```
#import <AudioToolbox/AudioToolbox.h>

CFURLRef        _soundFileURLRef;
SystemSoundID  _soundFileRef;

// Get the main bundle for the app
CFBundleRef mainBundle;
mainBundle = CFBundleGetMainBundle ();

// Get the URL to the sound file to play
soundFileURLRef  =   CFBundleCopyResourceURL (
                        mainBundle,
                        CFSTR ("tap"),
                        CFSTR ("aif"),
                        NULL
                    );

// Create a system sound object representing the sound file
AudioServicesCreateSystemSoundID (
   _soundFileURLRef,
   &_soundFileRef
);
```

System Sounds

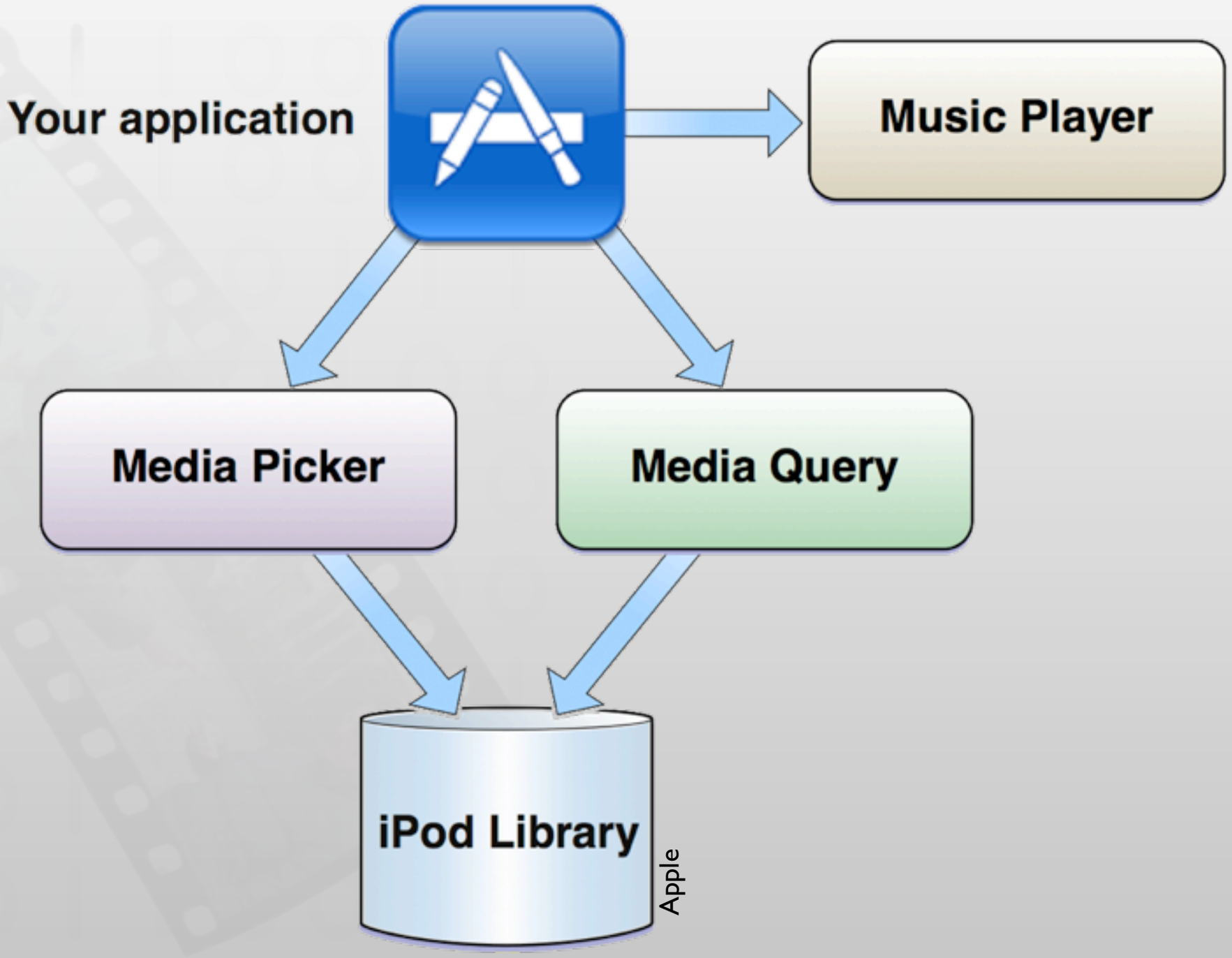
```
// Just play a sound
- (IBAction)playSystemSound {
  AudioServicesPlaySystemSound(self.soundFileRef);
}

// Play a sound and/or vibrate
- (IBAction)playAlertSound {
  AudioServicesPlayAlertSound(self.soundFileRef);
}

// Just vibrate
- (IBAction)vibrate {
  AudioServicesPlaySystemSound(kSystemSoundID_Vibrate);
}
```

Playback Completion Callback

```
#import <AudioToolbox/AudioServices.h>
// Just play a sound
- (IBAction)playSystemSound:(id)sender{
  AudioServicesAddSystemSoundCompletion(soundFileRef,
                                        NULL,
                                        NULL,
                                        completionCallback,
                                        (__bridge void*) self);
  AudioServicesPlaySystemSound(self.soundFileRef);
}

// Plain C callback function that just logs to the console
static void completionCallback(SystemSoundID ssID, void *mySelf)
{
  NSLog(@"Completion Callback");
  AudioServicesRemoveSystemSoundCompletion(ssID);
}
```

iPod Library Access

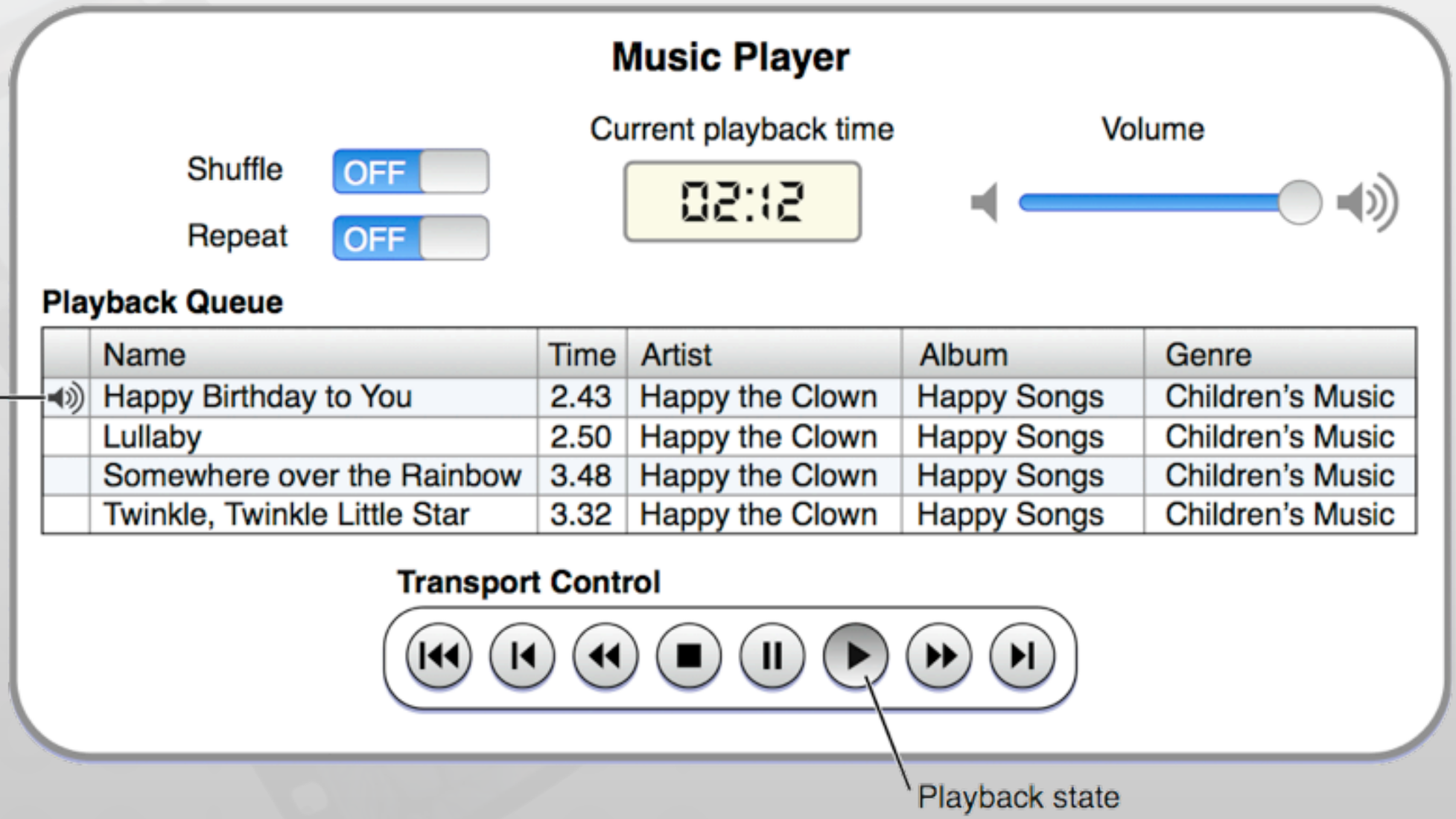
iPod Library Access



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

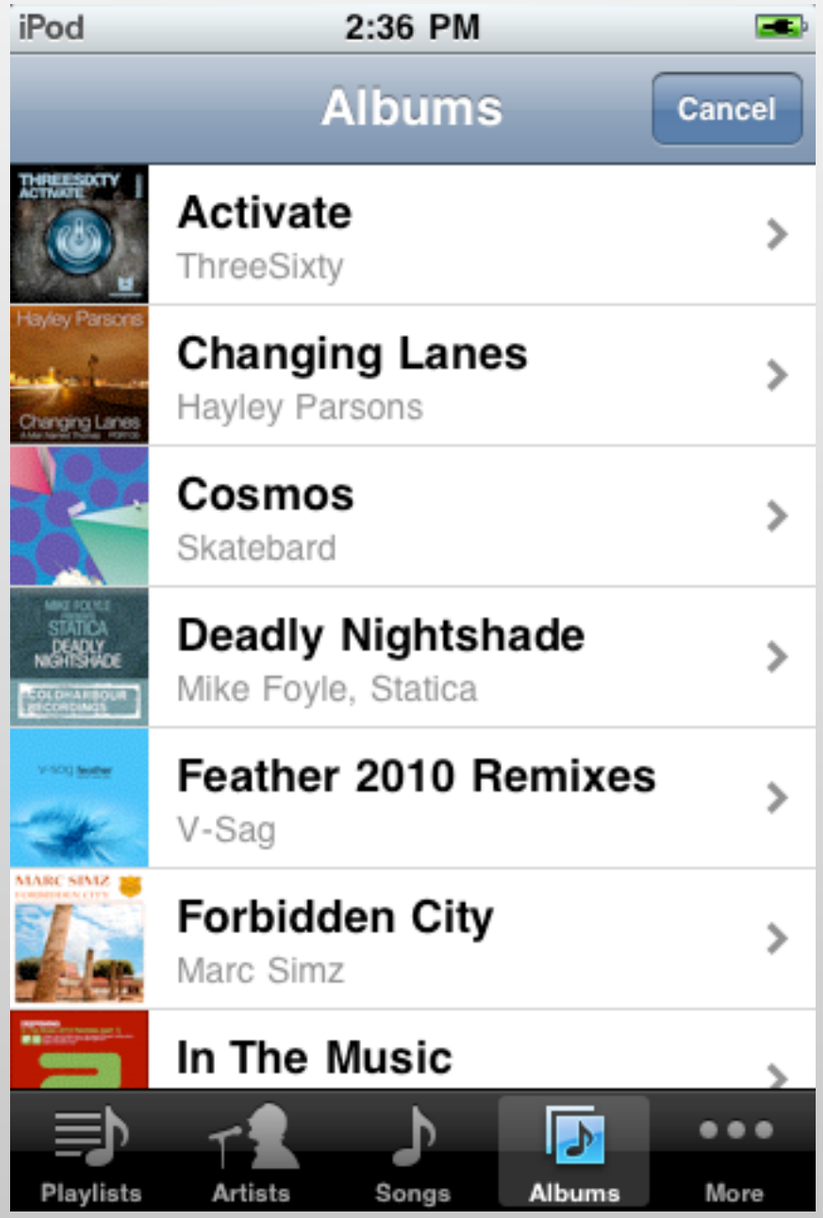


Music Player Basic Example

```
#import <MediaPlayer/MediaPlayer.h>

MPMusicPlayerController *myPlayer;

// You have two options concerning you player
// Just play music in your app
myPlayer = [MPMusicPlayerController applicationMusicPlayer];

// Keep playing if your app terminates
myPlayer = [MPMusicPlayerController iPodMusicPlayer];

//Fill the playback queue
[myPlayer setQueueWithQuery: [MPMediaQuery songsQuery]];

//Start playing
[myPlayer play];
```

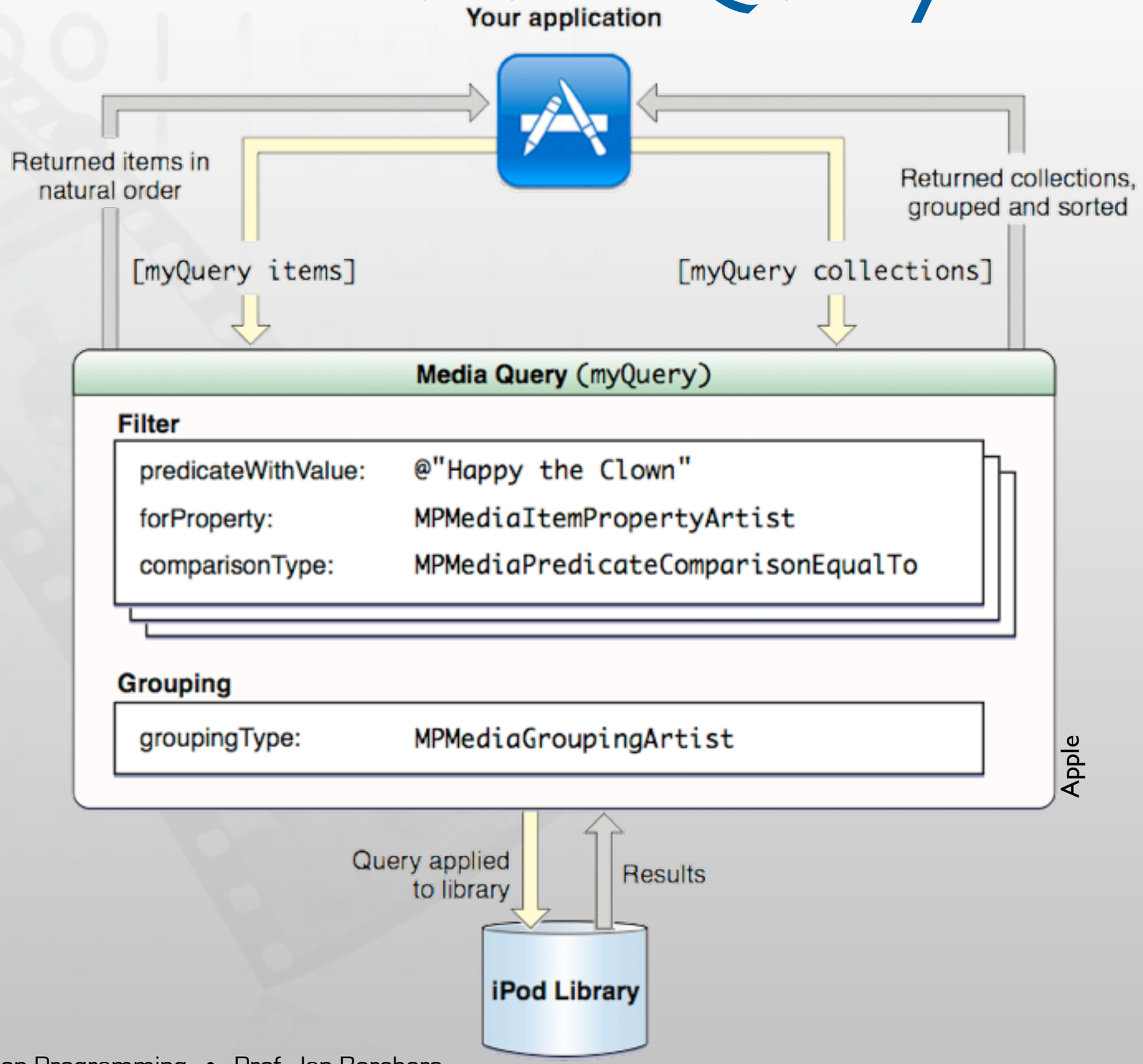
Media Items Picker

- Modal view controller
- Same interface as the iPod app
- No creation of playlists
- You have to store created collections yourself



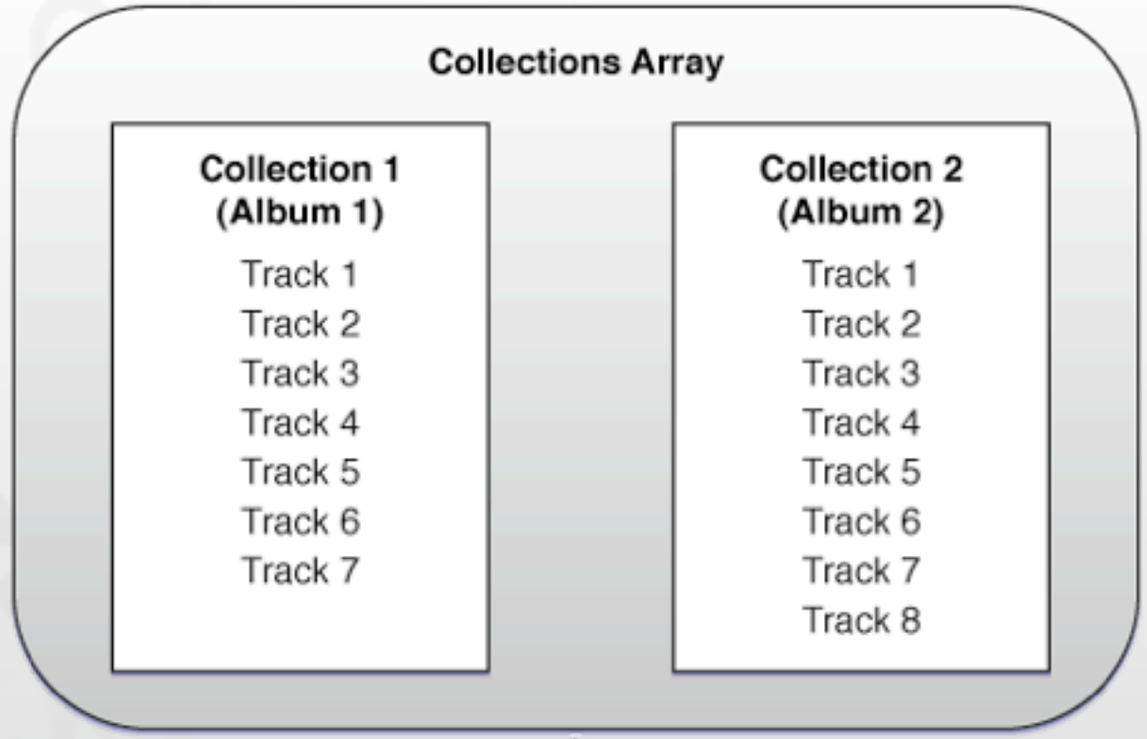
```objc
#pragma mark IBActions
- (IBAction)selectTrack:(id)sender {
    MPMediaPickerController *picker = [[MPMediaPickerController alloc] initWithMediaTypes:
                                       (MPMediaTypeMusic | MPMediaTypePodcast)];
    picker.allowsPickingMultipleItems = YES;
    picker.delegate = self;
    [self presentViewController:picker animated:YES completion:nil];
}

#pragma mark MediaItemPicker delegate methods
- (void)mediaPicker:(MPMediaPickerController *)mediaPicker
  didPickMediaItems:(MPMediaItemCollection *)mediaItemCollection {
    // Remove the picker
    [self dismissViewControllerAnimated:YES completion:nil];

    // Keep playing if your app terminates
    MPMusicPlayerController *myPlayer = [MPMusicPlayerController iPodMusicPlayer];

    //Fill the playback queue
    [myPlayer setQueueWithItemCollection:mediaItemCollection];

    //Start playing
    [myPlayer play];
}

- (void)mediaPickerDidCancel:(MPMediaPickerController *)mediaPicker
{
    //Remove the picker
    [self dismissViewControllerAnimated:YES completion:nil];
    NSLog(@"Selection cancelled");
}
```

Media Item Query

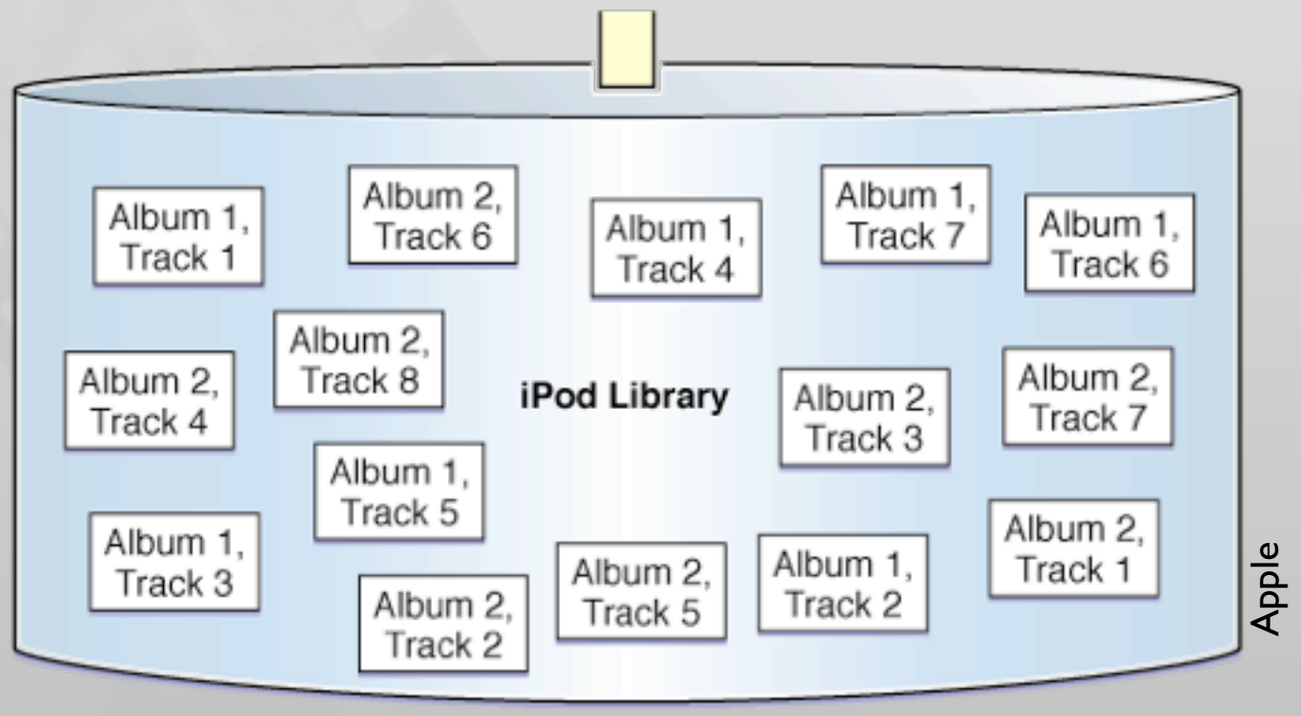


MPMediaQuery

```
    //Set up the query predicates
MPMediaPropertyPredicate *predicate =
 [MPMediaPropertyPredicate predicateWithValue:@"Skatebard"
                                  forProperty:MPMediaItemPropertyArtist

comparisonType:MPMediaPredicateComparisonEqualTo];

NSSet *predicates = [NSSet setWithObject:predicate];

// Set up the query
MPMediaQuery *query =
            [[MPMediaQuery alloc] initWithFilterPredicates:predicates];


// Group by album
query.groupingType = MPMediaGroupingAlbum;

// Get an array with MPMediaItemCollections containing the albums
NSArray *albums = [query collections];
```




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



Keys and Types

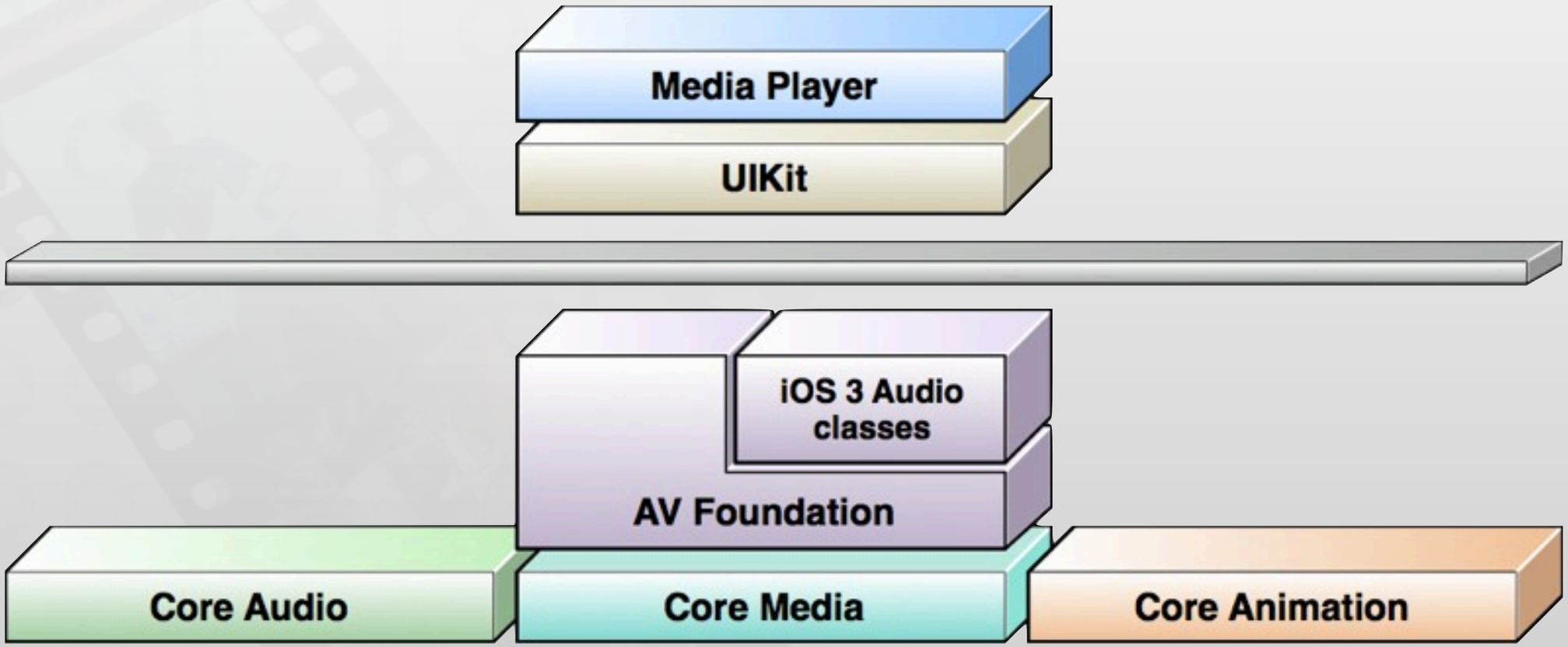
- General media item property keys
  - MPMediaItemPropertyPersistentID
  - MPMediaItemPropertyMediaType
  - MPMediaItemPropertyTitle
  - MPMediaItemPropertyAlbumTitle
  - MPMediaItemPropertyArtist
  - And many more
- Predicate Comparison types
  - MPMediaPredicateComparisonEqualTo
  - MPMediaPredicateComparisonContains

Using Metadata

```
// Set up the query
MPMediaQuery *query = [[MPMediaQuery alloc] initWithFilterPredicates:p];
// Group by album
query.groupingType = MPMediaGroupingAlbum;

// Get an array with MPMusicCollections containing the albums
NSArray *albums = [query collections];

// Iterate over all returned collections
for (MPMediaItemCollection *collection in albums){
   //and over all songs in these collections
   for (MPMediaItem *song in [collection items]) {
      NSString *title = [song valueForProperty:MPMediaItemPropertyTitle];
      NSLog(@"%@", title);
   }
}
```

Album Artwork

```
UIImageView *albumImageView;
MPMediaItemArtwork *artwork =
          [mediaItem valueForProperty: MPMediaItemPropertyArtwork];
UIImage *artworkImage = [artwork imageWithSize: albumImageView.bounds.size];
if (artworkImage) {
   albumImageView.image = artworkImage;
}
else {
   albumImageView.image = [UIImage imageNamed: @"noArtwork.png"];
}
```


Media Player
UIKit
iOS 3 Audio classes
AV Foundation
Core Audio
Core Media
Core Animation

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

Setting up an AVAudioPlayer

```
#import <AVFoundation/AVFoundation.h>

@property (strong) AVAudioPlayer *player;


NSString *filePath =
          [[NSBundle mainBundle] pathForResource:@"sample" ofType:@"m4a"];
NSURL *fileURL = [[NSURL alloc] initFileURLWithPath:filePath];

self.player =
   [[AVAudioPlayer alloc] initWithContentsOfURL:fileURL error:nil];

// Play sound only once
self.player.numberOfLoops = 0;
self.player.delegate = self;

[self.player prepareToPlay];
```

Playback

```
- (IBAction)playSound:(id)sender {
    // Play back the audio file
    [self.player play];
}

- (IBAction)pauseSound:(id)sender {
    // Just pause playback, do not release the acquired audio hardware
    [self.player pause];
}
```

```
- (IBAction)stopSound:(id)sender {
    // Release audio hardware.
    // However, currentTime is not reset, so it behaves like pause.
    [self.player stop];
}
```

AVAudioPlayer Delegate Protocol

```
#pragma mark AVAudioPlayer delegate methods

// Sound playback completion
- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)p successfully:
(BOOL)flag;


// Decoding error
- (void)playerDecodeErrorDidOccur:(AVAudioPlayer *)p error:(NSError *)error;


// Interruption handling
- (void)audioPlayerBeginInterruption:(AVAudioPlayer *)p;
- (void)audioPlayerEndInterruption:(AVAudioPlayer *)p;
```

Synchronize Playback

```
- (void) startSynchronizedPlayback {

    NSTimeInterval shortStartDelay = 0.01;                // seconds
    NSTimeInterval now = player.deviceCurrentTime;

    [player       playAtTime: now + shortStartDelay];
    [secondPlayer playAtTime: now + shortStartDelay];

    // Here, update state and user interface for
    // each player, as appropriate
}
```

AVAudioPlayer Properties

```
// Volume ranges from 0.0 to 1.0
@property float volume
// Pan ranges from -1.0 to 1.0
@property float pan
@property(readonly, getter=isPlaying) BOOL playing
@property NSInteger numberOfLoops

@property(readonly) NSDictionary *settings
Channel layout (AVChannelLayoutKey)
Encoder bit rate (AVEncoderBitRateKey)
Audio data format (AVFormatIDKey)
Channel count (AVNumberOfChannelsKey)
Sample rate (AVSampleRateKey)

// Audio file information
@property(readonly) NSUInteger numberOfChannels
@property(readonly) NSTimeInterval duration
@property(readonly) NSURL *url
@property NSTimeInterval currentTime
@property(readonly) NSTimeInterval deviceCurrentTime
```

Metering

```
// Metering is off by default
@property(getter=isMeteringEnabled) BOOL meteringEnabled

// Refreshes the average and peak power values for
// all channels of an audio player.
// Needs to be called to get current values for peak and average
power
- (void)updateMeters

- (float)averagePowerForChannel:(NSUInteger)channelNumber
- (float)peakPowerForChannel:(NSUInteger)channelNumber
```

Demo

AVAudioRecorder

```
// Get a reference to a writable directory
NSString *tempDir = NSTemporaryDirectory ();
NSString *soundFilePath = [tempDir stringByAppendingString:
@"sound.caf"];
self.soundFileURL = [NSURL fileURLWithPath: soundFilePath];

// assign ourselves as audio session delegate to get notifications
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
audioSession.delegate = self;
[audioSession setActive:YES error: nil];
```

AVAudioRecorder

```
[[AVAudioSession sharedInstance] setCategory: AVAudioSessionCategoryRecord
                                       error: nil];

NSDictionary *recordSettings =
[[NSDictionary alloc] initWithObjectsAndKeys:
 [NSNumber numberWithFloat: 44100.0], AVSampleRateKey,
 [NSNumber numberWithInt: kAudioFormatAppleLossless], AVFormatIDKey,
 [NSNumber numberWithInt: 1], AVNumberOfChannelsKey,
 [NSNumber numberWithInt: AVAudioQualityMax],
 AVEncoderAudioQualityKey, nil];

AVAudioRecorder *newRecorder =
[[AVAudioRecorder alloc] initWithURL: soundFileURL
                            settings: recordSettings
                               error: nil];
[recordSettings release];
self.soundRecorder = newRecorder;
[newRecorder release];

soundRecorder.delegate = self;
[soundRecorder prepareToRecord];
[soundRecorder record];
```

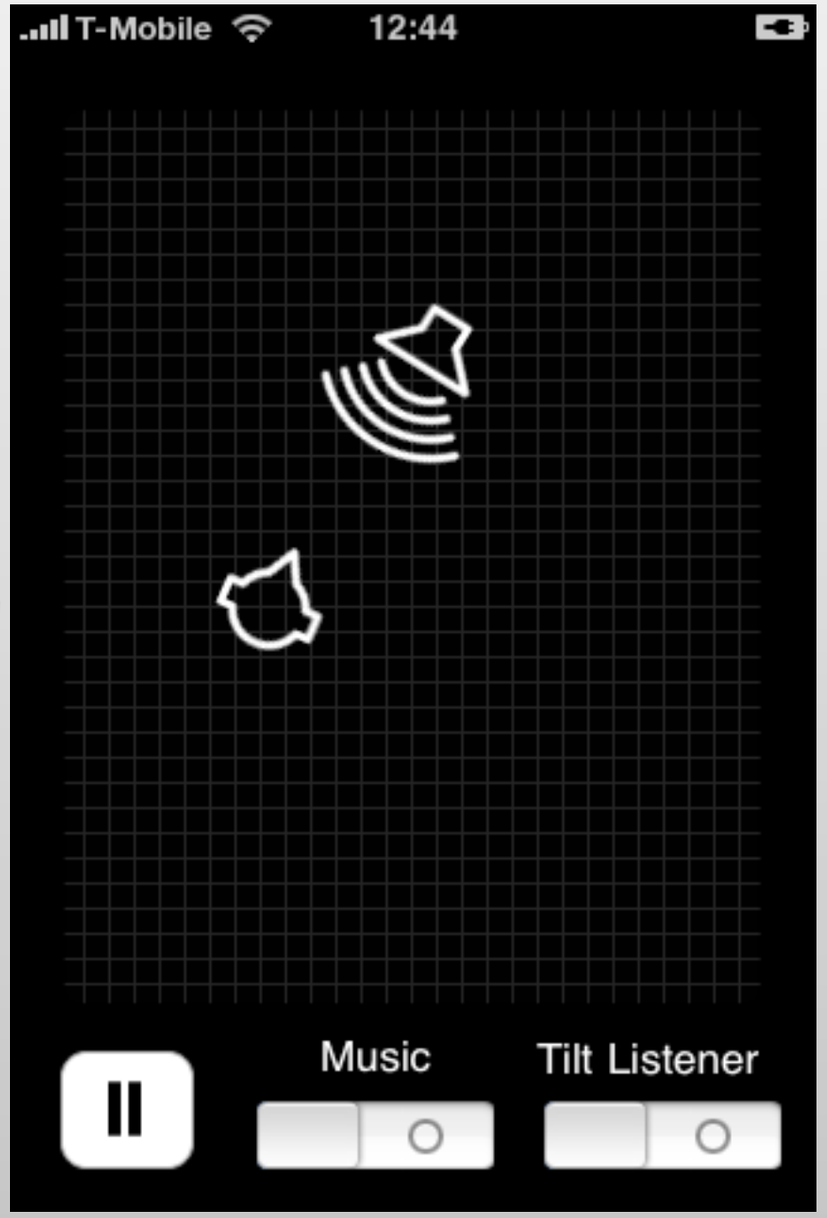
Demo

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



OpenAL on iOS

- Simple stereo panning
- High quality rendering: spherical head model, ILD, ITD, distance filtering
- Enhance realism yourself
- Supports only linear PCM data
  - Use Extended Audio File Services to convert
- Buffer playback completion callback

OpenALExample

Listener (Context)

Gain (0.0 : 1.0) 0.50

Stereo Rendering

High Quality (stereo only)

Distance Model: Inverse Clamped

Orientation X Y Z

Position: -49 0 -43

Velocity: -0.0 -0.0

Elevation (Y) :

Doppler Factor (0.0 : 1.0) 1.00

Speed Of Sound (0 : 686) 343.3

Velocity (Listener Vector) 0

Sources

	Play	Position X	Y	Z	Distance Ref	Max	Rolloff
Car	☑	175	0	-175	5	1,000	1.0
Voices	☑	175	0	175	5	1,000	1.0
Bubbles	☑	-148	0	-118	5	1,000	1.0
Electric	☑	-175	0	175	5	1,000	1.0
Capture	☑				5	1,000	1.0

	Gain (0.0 : 1.0)	Pitch (0.5x : 4.0x)
Car	0.75	1.00
Voices	1.00	1.00
Bubbles	1.00	1.00
Electric	1.00	1.00
Capture	1.00	1.00

Capture Captured Samples:

	Direction/Velocity Angle	Velocity X	Z	Velocity (Source Vector)
Car		0	0	0
Voices		0	0	0
Bubbles		0	0	0
Electric		0	0	0
Capture		0	0	0

	Use Cones	Inner Cone	Outer Cone	Outer Cone Gain
Car	☑	90.0	180.0	0.00
Voices	☑	100.1	270.0	0.00
Bubbles	☐	90.0	180.0	0.00
Electric	☑	180.0	71.6	0.00
Capture	☐	90.0	180.0	0.00

Click and drag objects

Z

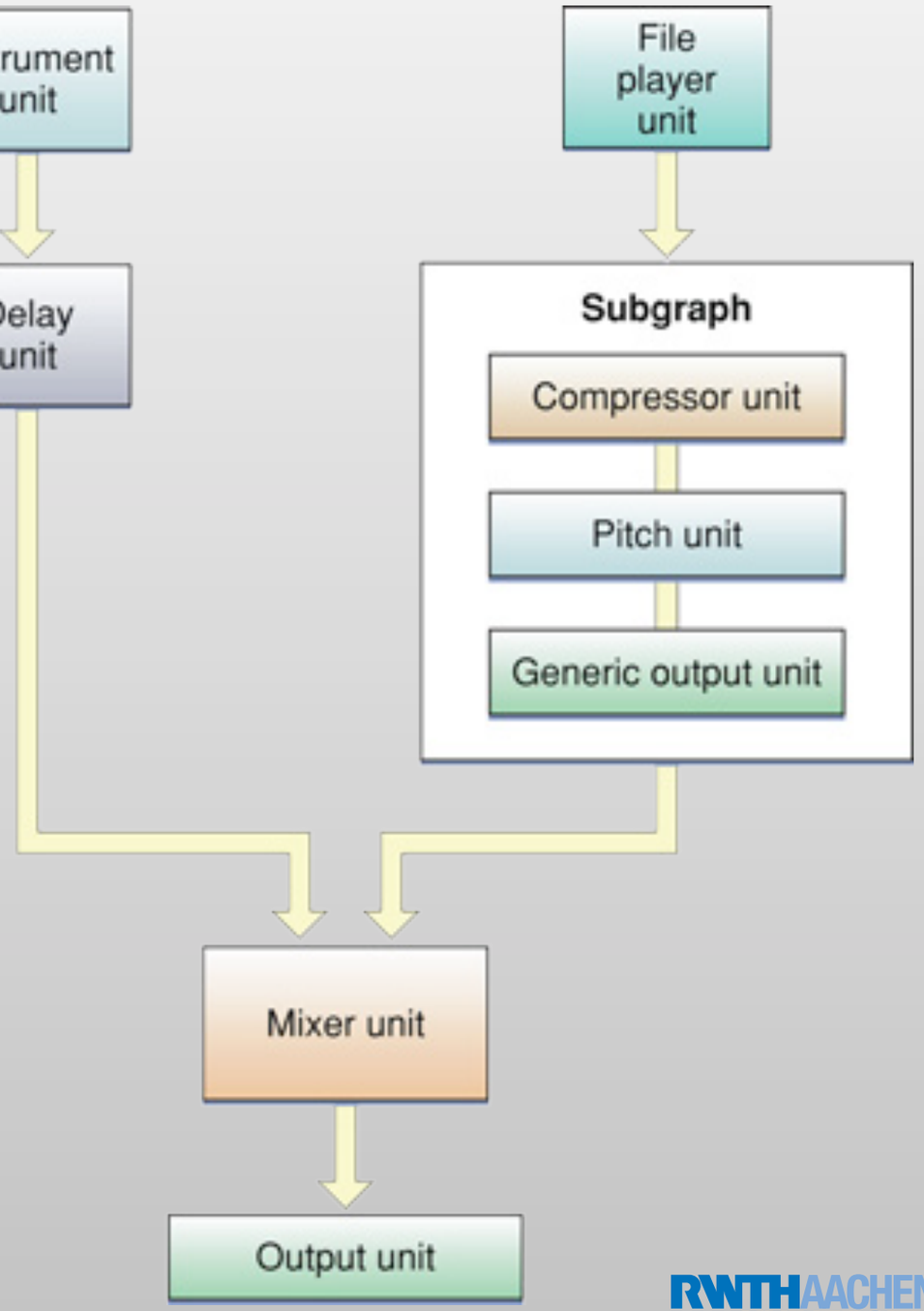
\+

\- X \+

Advanced Audio

Audio Units

- Audio processing
- Plug-in architecture
- Low latency
- No user interface



Audio Units

- Converter unit
- iPod Equalizer unit
- 3D Mixer unit
- Multichannel Mixer unit
- Generic Output unit
- Remote I/O unit
- Voice Processing I/O unit

Using Audio Units

- Create an AUGraph
- Add the nodes to the graph
  - Don't forget the output node
- Connect the nodes to each other
- Open the graph
- Set parameters of the nodes
- Initialize the graph

Using Audio Units

```
// create the AUGraph
OSStatus result = NewAUGraph(&mGraph);

// add the nodes to the graph
result = AUGraphAddNode(mGraph, &output_desc, &outputNode);
result = AUGraphAddNode(mGraph, &mixer_desc, &mixerNode );

// connect the nodes to each other
result = AUGraphConnectNodeInput(mGraph, mixerNode, 0, outputNode,
0);

// open the graph
result = AUGraphOpen(mGraph);

// configure some properties
result = AudioUnitSetProperty(mMixer,
kAudioUnitProperty_ElementCount,
                    kAudioUnitScope_Input, 0, &numbuses,
sizeof(UInt32));

// initialize the graph
result = AUGraphInitialize(mGraph);
```

More Audio

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

Video

MPMoviePlayer

- Plays movies from file or network stream
- Fullscreen or custom view
- MPMoviePlayerViewController
  - Add sublayers on top of the video
  - Custom background content
- Standard user interface
- Programmatic control

MPMoviePlayer

```
NSString *filePath = [[NSBundle mainBundle]
                                pathForResource:@"Movie" ofType:@"m4v"];

NSURL *fileURL = [NSURL fileURLWithPath:filePath];

self.moviePlayer =
        [[MPMoviePlayerController alloc] initWithContentURL: fileURL];

[self.moviePlayer.view setFrame: [self.view bounds]];
```

```
[self.view addSubview:self.moviePlayer.view];
[[NSNotificationCenter defaultCenter]
                    addObserver: self
                       selector: @selector(playbackCompletionCallback:)
                           name: MPMoviePlayerPlaybackDidFinishNotification
                         object: nil];

[self.moviePlayer play];
```

MPMoviePlayer Callback

```
// When the movie is finished, release the controller.
- (void)playbackCompletionCallback:(NSNotification*)aNotification
{
   [[NSNotificationCenter defaultCenter]
                  removeObserver: self
                            name: MPMoviePlayerPlaybackDidFinishNotification
                          object: nil];
   // Release the player and the movie
   self.moviePlayer = nil;
}
```

Demo

MPMoviePlayer

- Scaling mode
- Fullscreen
- Repeat mode
- Autoplay
- Generate thumbnail images

Summary

- Playing sound
- Accessing iPod library
- OpenAL
- Video
- Reading Assignment:
  - Multimedia Programming Guide
  - iPod Library Access Programming Guide
  - Audio Session Programming Guide
  - AVFoundation Programming Guide