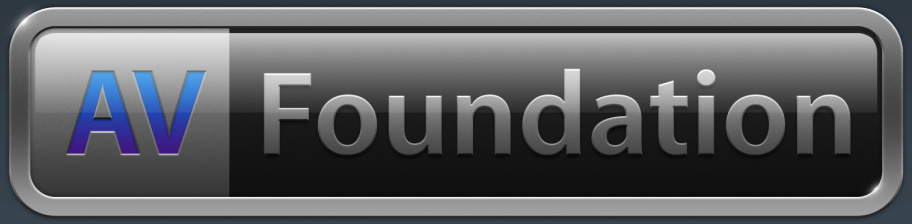
# Exploring AV Foundation

Session 405

**Kevin Calhoun and Simon Goldrei**
Media Systems Engineering

# AVFoundation

- Foundational framework for timed audiovisual media
- Integrated with OS
- Same model for Mac OS X and iOS
- Submitted for your consideration
  - New apps
  - Transition for existing apps

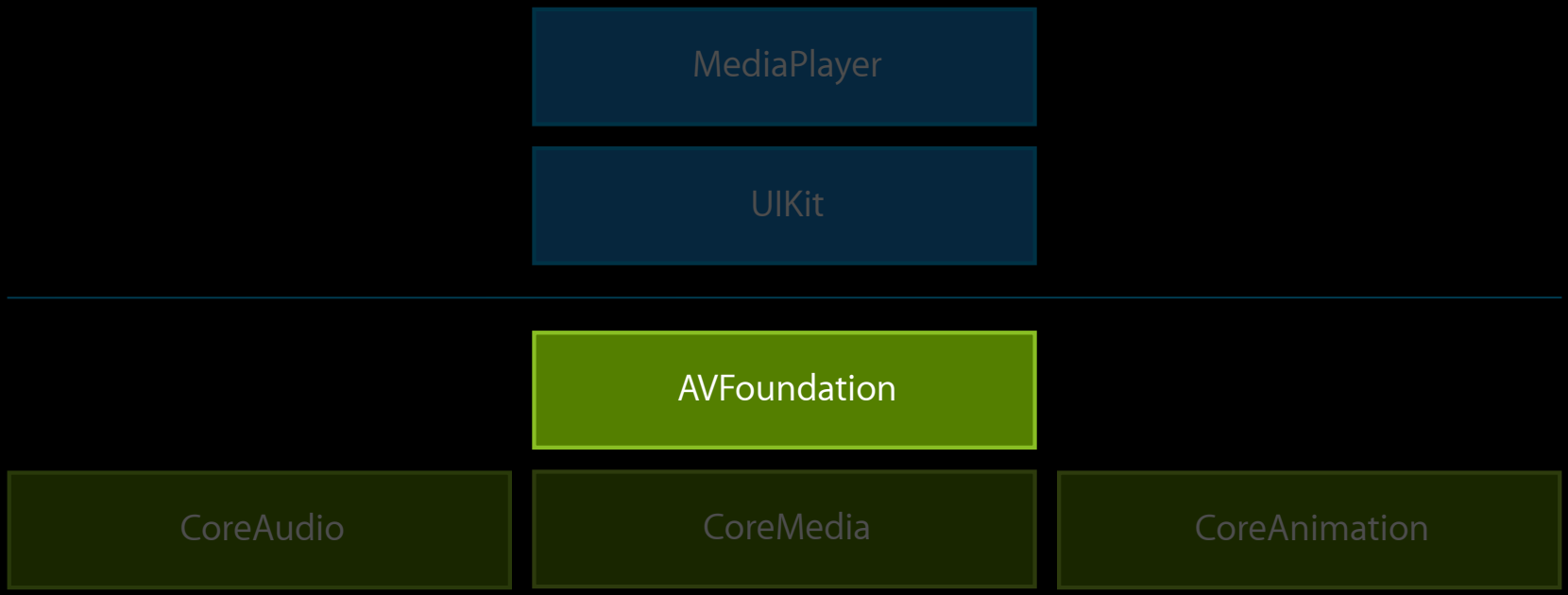

AV Foundation

# Agenda

- Core Concepts
  - Sharing media services in a robust manner
  - Building responsive and efficient media applications
  - Key protocols and interfaces
- How AV Foundation models media
  - Media inspection
  - Mechanics of playback
- New API

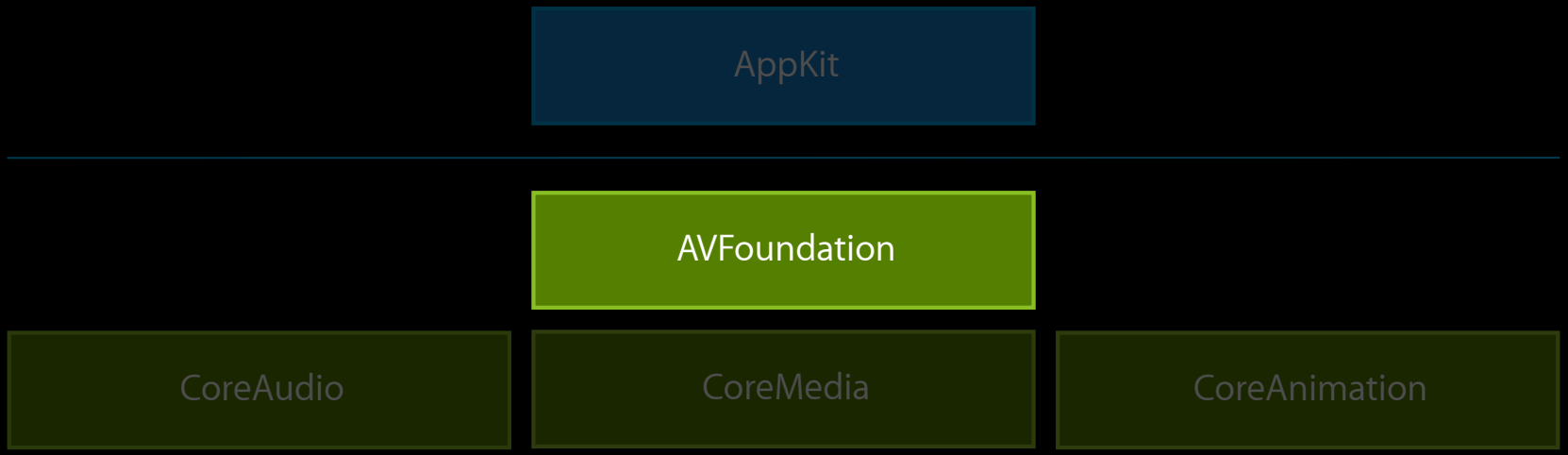
# Where Is AV Foundation

## iOS



# Where Is AV Foundation

## Mac OS X



# When to Use AV Foundation

- Inspection
  - Media properties
- Playback and presentation
  - Media controller for custom behavior and UI
- Composition
  - Compose and combine segments of media
- Export
  - Reauthor, possibly composed, media
- Capture
  - Control camera features, obtain image data, persist to disk

# Why Use AV Foundation

## One approach for both Mac OS X and iOS

- Feature rich-media API
- 64-bit native

- Hardware acceleration comes standard
  - Hardware decode whenever available
    - Nothing to adopt
  - Advanced video pipeline
    - GPU-backed
    - Core Animation integrated

# Why Use AV Foundation

## Solving the challenges of timed media

- Resources are large, stored remotely and take time to parse and decode

- AV Foundation:
  - Does not obscure this reality
  - Performs necessary operations to service your requests
  - Coordinates and resolves requests with a dynamic playback environment

# Why Use AV Foundation

## Solving the challenges of timed media

- AV Foundation
  - Support your media application to remain responsive and efficient

# Using AV Foundation Efficiently

## Classes and protocols for responsive media applications

“The only reason for time is so that everything doesn’t happen at once.”

Albert Einstein

# AV Foundation Themes

## Some things best happen concurrently

- Two media models
  - Static: Loadable and once loaded does not change
    - Does not mutate independently
  - Dynamic: Observable while prepared for, and during, playback
    - Does mutate independently with playback environment
- You want to:
  - Request values asynchronously and await when loaded
  - Key-Value Observe dynamic values where offered

# AV Foundation Themes

## iOS multitasks

Only on iOS

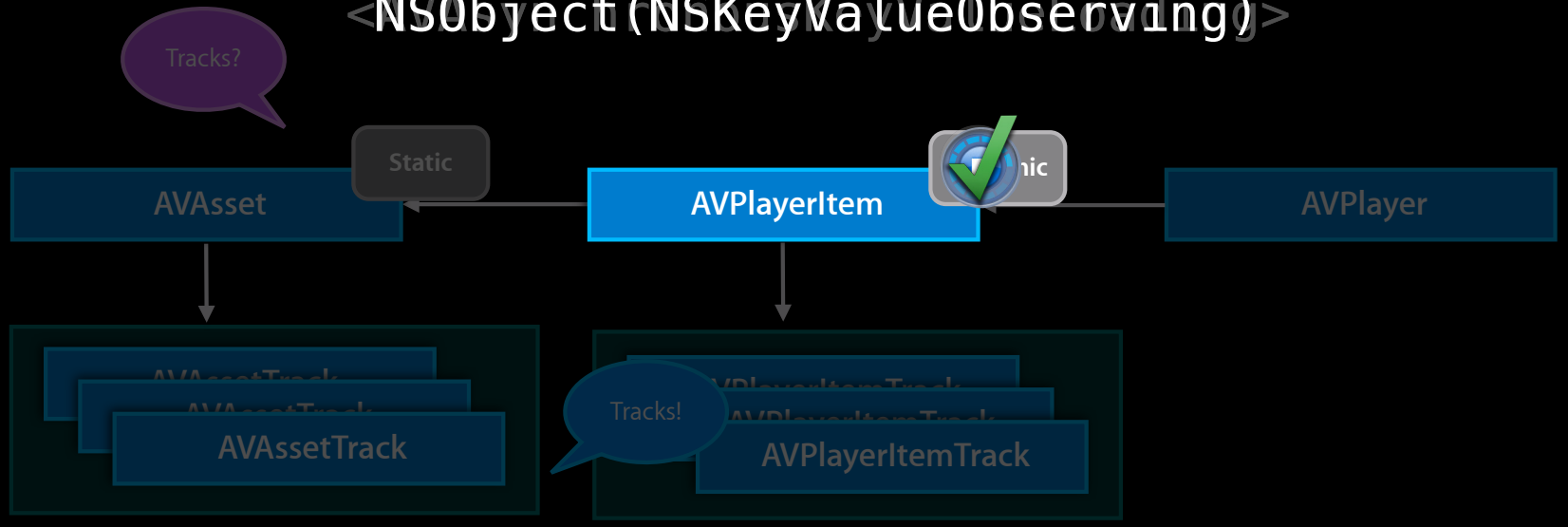
- Be prepared
  - Your media operations may be interrupted
  - Shared media services may be reset
- Be considerate
  - You need to share media services asynchronously…

"There's a protocol for that" ™

# AV Foundation

## Core playback classes

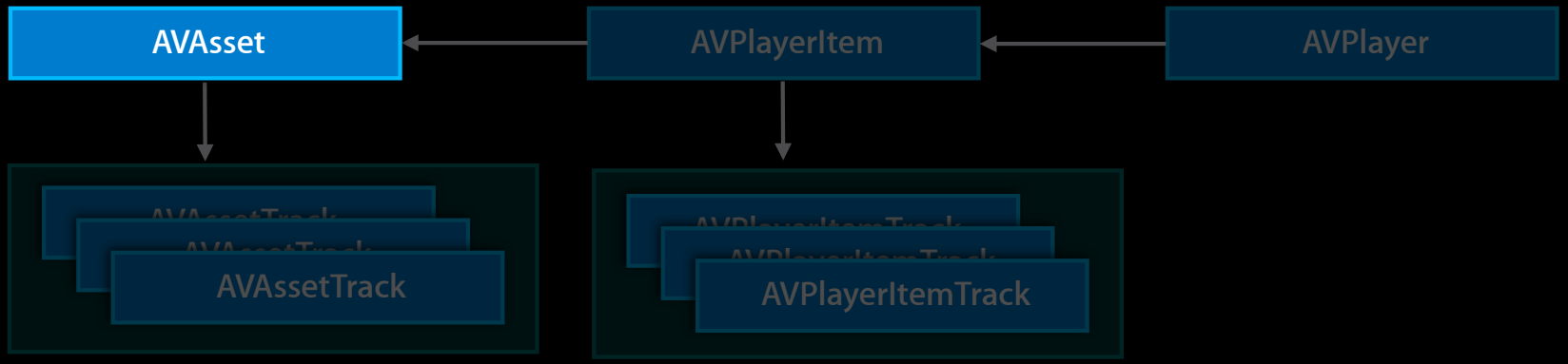
NSObject(NSKeyValueObserving)




AVAssetTrack a resource within a resource during playback

# AV Foundation

## Core playback classes



# AVAsset

## Inspection via <AVAsynchronousKeyValueLoading>

- Models the whole media resource

```
AVAsset *asset = [AVURLAsset URLAssetWithURL:url options:opts];
```

  - A new AVAsset has no values loaded

- Discover AVAsset utility
  - exportable
  - composable
  - readable
  - playable

```
NSArray *keys = [NSArray arrayWithObject:@”playable”];
[asset loadValuesAsynchronouslyForKeys:keys completionHandler: ^{
   NSError *error = nil;
   AVKeyValueStatus playableStatus = [asset statusOfValueForKey:@"playable"
                                                          error:&error];
    switch (playableStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
   }
}];
```

```
NSArray *keys = [NSArray arrayWithObject:@”playable”];
[asset loadValuesAsynchronouslyForKeys:keys completionHandler: ^{
   NSError *error = nil;
   AVKeyValueStatus playableStatus = [asset statusOfValueForKey:@"playable"
                                                          error:&error];
    switch (playableStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
   }
}];
```

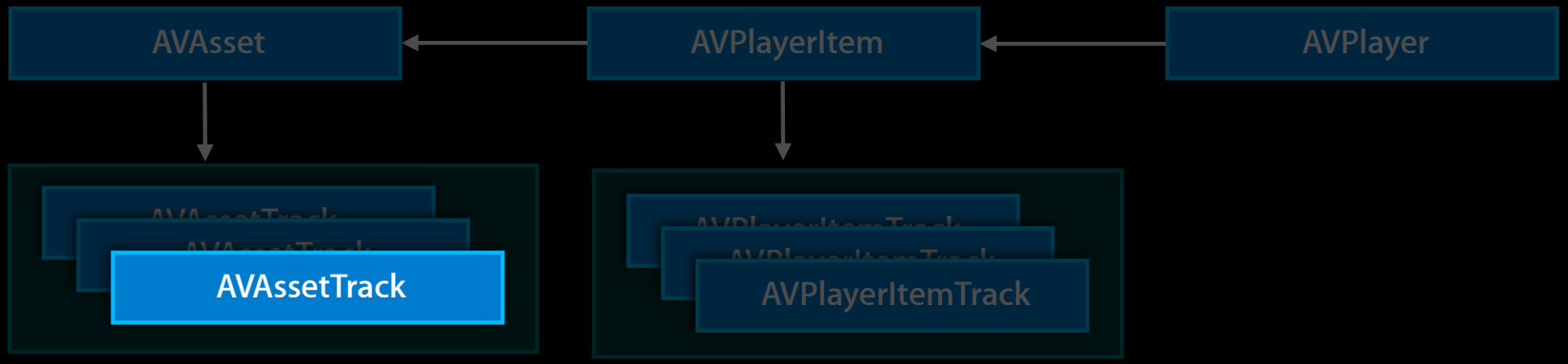
```
NSArray *keys = [NSArray arrayWithObject:@”playable”];
[asset loadValuesAsynchronouslyForKeys:keys completionHandler: ^{
   NSError *error = nil;
   AVKeyValueStatus playableStatus = [asset statusOfValueForKey:@"playable"
                                                          error:&error];
    switch (playableStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
   }
}];
```

```
NSArray *keys = [NSArray arrayWithObject:@”playable”];
[asset loadValuesAsynchronouslyForKeys:keys completionHandler: ^{
   NSError *error = nil;
   AVKeyValueStatus playableStatus = [asset statusOfValueForKey:@"playable"
                                                          error:&error];
    switch (playableStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
   }
}];
```

```
NSArray *keys = [NSArray arrayWithObject:@”playable”];
[asset loadValuesAsynchronouslyForKeys:keys completionHandler: ^{
   NSError *error = nil;
   AVKeyValueStatus playableStatus = [asset statusOfValueForKey:@"playable"
                                                          error:&error];
    switch (playableStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
   }
}];
```

# AV Foundation

## Core playback classes



# AVAssetTrack

## Inspection via <AVAsynchronousKeyValueLoading>

- Models the static state of a media stream

```
AVKeyValueStatus status = [asset statusOfValueForKey:@”tracks” error:nil];
if (AVKeyValueStatusLoaded != status) {
     NSArray *keys = [NSArray arrayWithObject:@”tracks”];
     [asset loadValuesAsynchronouslyForKeys:keys
                          completionHandler:^{ ... }];
}
```

  - Each AVAssetTrack is of a single media type
  - formatDescription, frameRate, dataRate, language tagging, and metadata

# AVAssetTrack

## Inspection via <AVAsynchronousKeyValueLoading>

- Models the static state of a media stream

```
AVKeyValueStatus status = [asset statusOfValueForKey:@”tracks” error:nil];
if (AVKeyValueStatusLoaded != status) {
     NSArray *keys = [NSArray arrayWithObject:@”tracks”];
     [asset loadValuesAsynchronouslyForKeys:keys
                          completionHandler:^{ ... }];
}
```

  - Each AVAssetTrack is of a single media type
  - formatDescription, frameRate, dataRate, language tagging, and metadata

# AVAssetTrack

## Inspection via <AVAsynchronousKeyValueLoading>

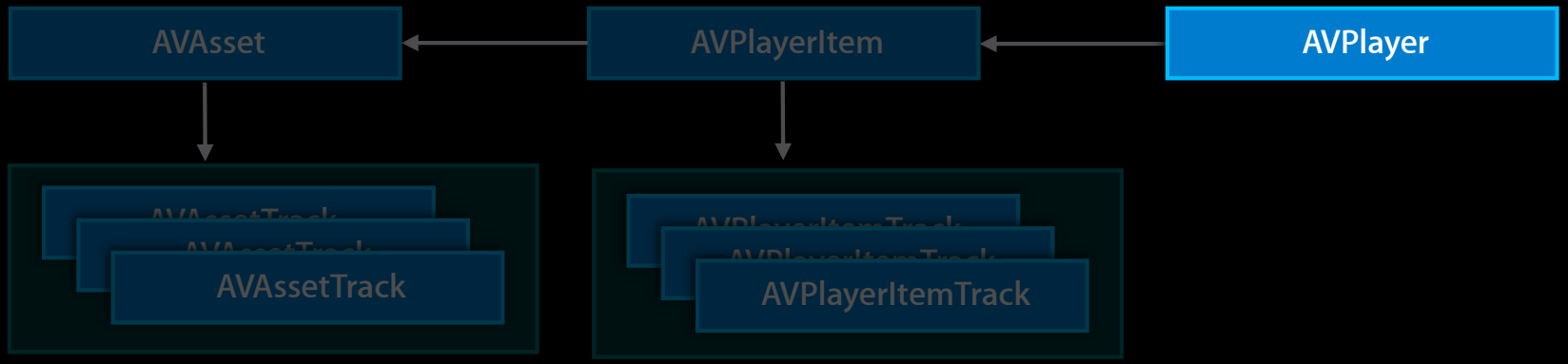
- Models the static state of a media stream

```
AVKeyValueStatus status = [asset statusOfValueForKey:@”tracks” error:nil];
if (AVKeyValueStatusLoaded != status) {
     NSArray *keys = [NSArray arrayWithObject:@”tracks”];
     [asset loadValuesAsynchronouslyForKeys:keys
                          completionHandler:^{ ... }];
}
```

  - Each AVAssetTrack is of a single media type
  - formatDescription, frameRate, dataRate, language tagging, and metadata

# AV Foundation

## Core playback classes



# AVPlayer

## Observation via `NSObject(NSKeyValueObserving)`

- A media controller
  - With just a few AVPlayerItem conveniences

```
AVPlayer *myPlayer = [AVPlayer playerWithPlayerItem:myPlayerItem];
```

# AVPlayer

## Observing media time

- Building transport controls
  - Scrubber
  - Coordinating events with media-time epochs

- Time observation is very dynamic
  - Key-Value Observing and changeDictionary is not appropriate

```
(id)addPeriodicTimeObserverForInterval:queue:usingBlock:

(id)addBoundaryTimeObserverForTimes:queue:usingBlock:
```

# Observing Media Time

```
- (void) setUpTransportUI {
    CMTime interval = CMTimeMakeWithSeconds(0.5);
    id myObserver = [[myPlayer addPeriodicTimeObserverForInterval:interval
                                queue:dispatch_get_main_queue()
                                usingBlock:^{
                                            [self movePlayheadUI];
                                        }] retain];
}


- (void) cleanUp {
    [myPlayer removeTimeObserver:myObserver];
    [myObserver release];
}
```

# Observing Media Time

```
- (void) setUpTransportUI {
    CMTime interval = CMTimeMakeWithSeconds(0.5);
    id myObserver = [[myPlayer addPeriodicTimeObserverForInterval:interval
                                queue:dispatch_get_main_queue()
                                usingBlock:^{
                                            [self movePlayheadUI];
                                        }] retain];
}


- (void) cleanUp {
    [myPlayer removeTimeObserver:myObserver];
    [myObserver release];
}
```

# Observing Media Time

```
- (void) setUpTransportUI {
    CMTime interval = CMTimeMakeWithSeconds(0.5);
    id myObserver = [[myPlayer addPeriodicTimeObserverForInterval:interval
                                queue:dispatch_get_main_queue()
                                usingBlock:^{
                                            [self movePlayheadUI];
                                        }] retain];
}


- (void) cleanUp {
    [myPlayer removeTimeObserver:myObserver];
    [myObserver release];
}
```

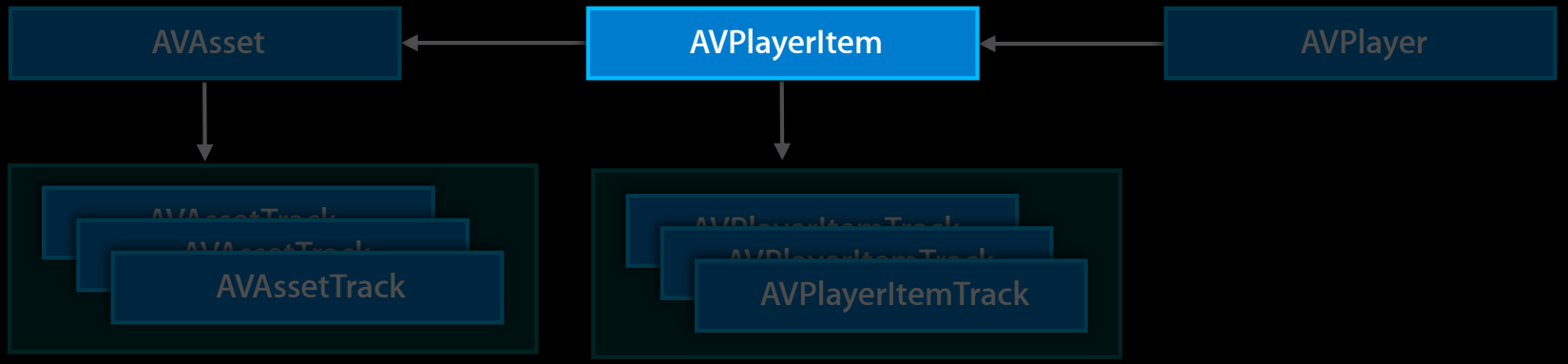
# Observing Media Time

```
- (void) setUpTransportUI {
    CMTime interval = CMTimeMakeWithSeconds(0.5);
    id myObserver = [[myPlayer addPeriodicTimeObserverForInterval:interval
                                queue:dispatch_get_main_queue()
                                usingBlock:^{
                                            [self movePlayheadUI];
                                        }] retain];
}


- (void) cleanUp {
    [myPlayer removeTimeObserver:myObserver];
    [myObserver release];
}
```

# AV Foundation

## Core playback classes



# AVPlayerItem

## Observation via `NSObject(NSKeyValueObserving)`

- Models the dynamic, or playback, state of an AVAsset

- One AVAsset may be used to create many AVPlayerItems

```
AVPlayerItem *myItem = [AVPlayerItem playerItemWithAsset:myAsset];
```

- Or

```
AVPlayerItem *myItem = [AVPlayerItem playerItemWithURL:myURL];
```

# AVPlayerItem

## Observation via `NSObject(NSKeyValueObserving)`

- Media properties
  - duration, tracks, presentationSize, audioMix
  - New iOS 5: canPlayFastForward, canPlayFastReverse

# AVPlayerItem

## Observation via `NSObject(NSKeyValueObserving)`

- Manipulate time
  - seekToTime, stepByCount, forwardPlaybackEndTime, reversePlaybackEndTime

# AVPlayerItem

## Observation via `NSObject(NSKeyValueObserving)`

- Playability hints
  - playbackLikelyToKeepUp, playbackBufferFull, playbackBufferEmpty, loadedTimeRanges, seekableTimeRanges

# AVPlayerItem

## Observation via `NSObject(NSKeyValueObserving)`

- Services
  - New: accessLog, errorLog
  - status
    - Eligibility for playback
    - Shared media services termination
    - Unknown, ReadyForPlayback, or Failed

```
[myItem addObserver:self
         forKeyPath:@”status”
            options:NSKeyValueObservingOptionNew
            context:myStatusObservationContext];
```

# AVPlayerItem

## Observation via `NSObject(NSKeyValueObserving)`

- Services
  - New: accessLog, errorLog
  - status
    - Eligibility for playback
    - Shared media services termination
    - Unknown, ReadyForPlayback, or Failed

```
[myItem addObserver:self
         forKeyPath:@”status”
            options:NSKeyValueObservingOptionNew
            context:myStatusObservationContext];
```

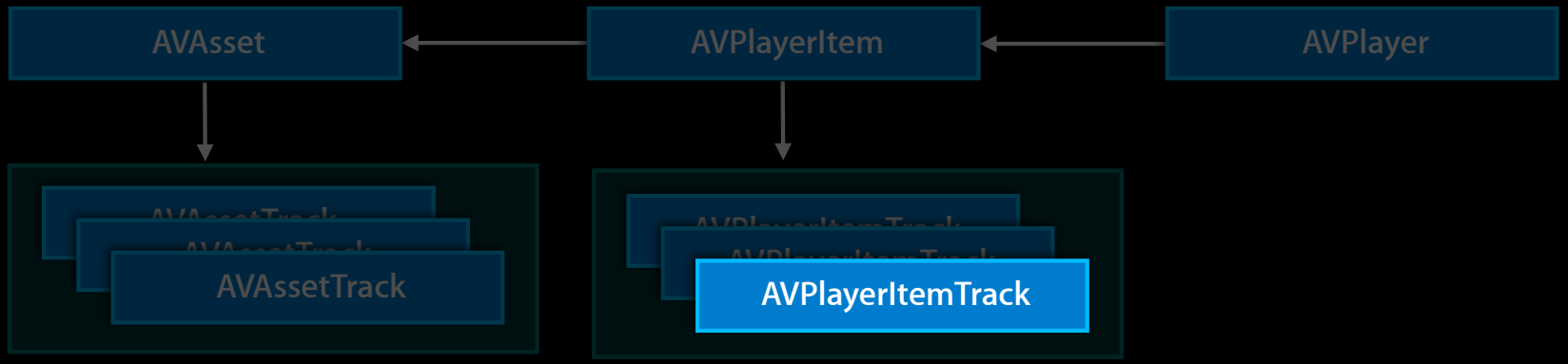
```
- (void) observeValueForKeyPath:(NSString *)keyPath
                        ofObject:(id)object
                          change:(NSDictionary *)change
                         context:(void *)context {
   if (context == myStatusObservationContext) {
      AVPlayerItem item = (AVPlayerItem *)object;
      AVPlayerItemStatus status = [item status];
      switch (status) {
         case AVPlayerItemStatusReadyToPlay:
            [myPlayer play];
            break;
         case AVPlayerItemStatusFailed:
         ...
      }
   }
}
```

```
- (void) observeValueForKeyPath:(NSString *)keyPath
                        ofObject:(id)object
                          change:(NSDictionary *)change
                         context:(void *)context {
   if (context == myStatusObservationContext) {
      AVPlayerItem item = (AVPlayerItem *)object;
      AVPlayerItemStatus status = [item status];
      switch (status) {
         case AVPlayerItemStatusReadyToPlay:
            [myPlayer play];
            break;
         case AVPlayerItemStatusFailed:
         ...
      }
   }
}
```

```
- (void) observeValueForKeyPath:(NSString *)keyPath
                        ofObject:(id)object
                          change:(NSDictionary *)change
                         context:(void *)context {
   if (context == myStatusObservationContext) {
      AVPlayerItem item = (AVPlayerItem *)object;
      AVPlayerItemStatus status = [item status];
      switch (status) {
         case AVPlayerItemStatusReadyToPlay:
            [myPlayer play];
            break;
         case AVPlayerItemStatusFailed:
         ...
      }
   }
}
```

# AV Foundation

## Core playback classes



# AVPlayerItemTrack

## Observation via `NSObject(NSKeyValueObserving)`

- A model for the dynamic state of an AVAssetTrack during playback
  - Playability
    - enabled
  - Access to backing AVAssetTrack

```
[myPlayerItem addObserver:self
                forKeyPath:@”tracks”
                   options:NSKeyValueObservingOptionNew
                   context:myPlayerItemObservationContext];
...
- (void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                     change:(NSDictionary *)change context:(void *)context {
   if (context == myPlayerItemObservationContext) {
      NSArray *tracks =
                (NSArray *)[change objectForKey:NSKeyValueChangeNewKey];
      for (AVPlayerItemTrack *track in tracks) {
            if ([[track assetTrack] mediaType] == AVMediaTypeVideo)
                [track setEnabled:NO];
      }
   }
}
```

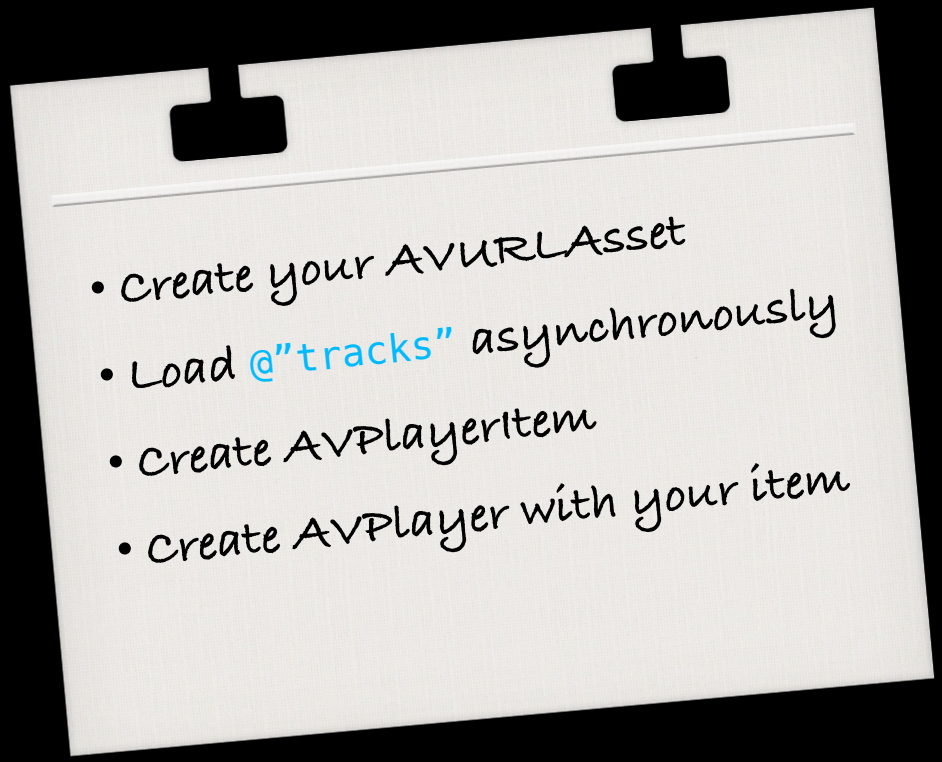
```
[myPlayerItem addObserver:self
                forKeyPath:@”tracks”
                   options:NSKeyValueObservingOptionNew
                   context:myPlayerItemObservationContext];
...
- (void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                     change:(NSDictionary *)change context:(void *)context {
   if (context == myPlayerItemObservationContext) {
      NSArray *tracks =
                (NSArray *)[change objectForKey:NSKeyValueChangeNewKey];
      for (AVPlayerItemTrack *track in tracks) {
            if ([[track assetTrack] mediaType] == AVMediaTypeVideo)
                [track setEnabled:NO];
      }
   }
}
```

```
[myPlayerItem addObserver:self
                forKeyPath:@”tracks”
                   options:NSKeyValueObservingOptionNew
                   context:myPlayerItemObservationContext];
...
- (void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                     change:(NSDictionary *)change context:(void *)context {
   if (context == myPlayerItemObservationContext) {
      NSArray *tracks =
                (NSArray *)[change objectForKey:NSKeyValueChangeNewKey];
      for (AVPlayerItemTrack *track in tracks) {
            if ([[track assetTrack] mediaType] == AVMediaTypeVideo)
                [track setEnabled:NO];
      }
   }
}
```

```
[myPlayerItem addObserver:self
                forKeyPath:@”tracks”
                   options:NSKeyValueObservingOptionNew
                   context:myPlayerItemObservationContext];
...
- (void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                     change:(NSDictionary *)change context:(void *)context {
   if (context == myPlayerItemObservationContext) {
      NSArray *tracks =
                (NSArray *)[change objectForKey:NSKeyValueChangeNewKey];
      for (AVPlayerItemTrack *track in tracks) {
            if ([[track assetTrack] mediaType] == AVMediaTypeVideo)
                [track setEnabled:NO];
      }
   }
}
```

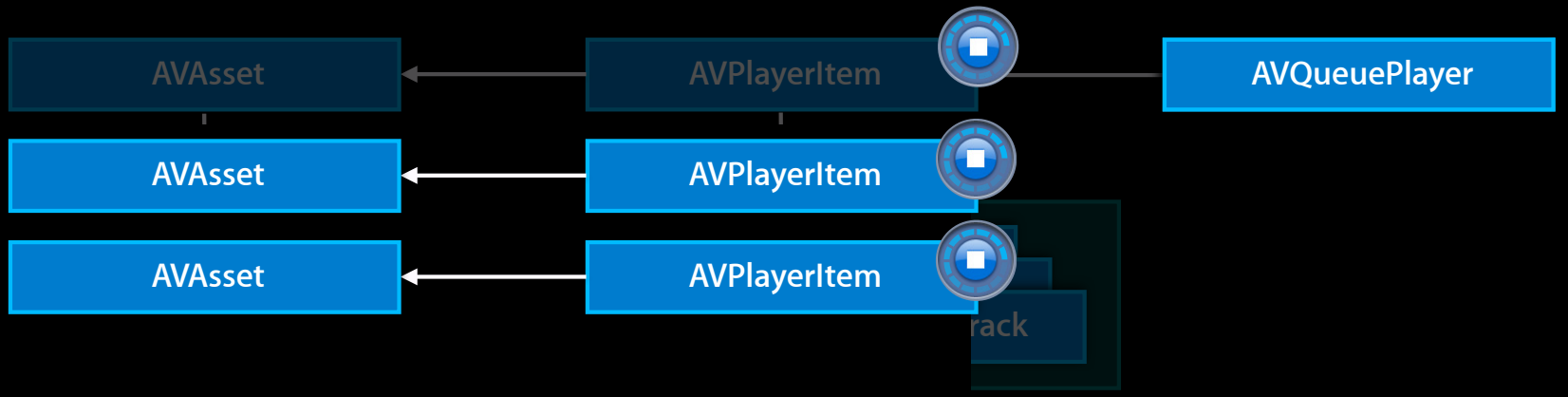
```
[myPlayerItem addObserver:self
                forKeyPath:@”tracks”
                   options:NSKeyValueObservingOptionNew
                   context:myPlayerItemObservationContext];
...
- (void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                     change:(NSDictionary *)change context:(void *)context {
   if (context == myPlayerItemObservationContext) {
      NSArray *tracks =
                (NSArray *)[change objectForKey:NSKeyValueChangeNewKey];
      for (AVPlayerItemTrack *track in tracks) {
            if ([[track assetTrack] mediaType] == AVMediaTypeVideo)
               [track setEnabled:NO];
      }
   }
}
```

# Playback in Four Easy Steps



# AV Foundation

## Core playback classes

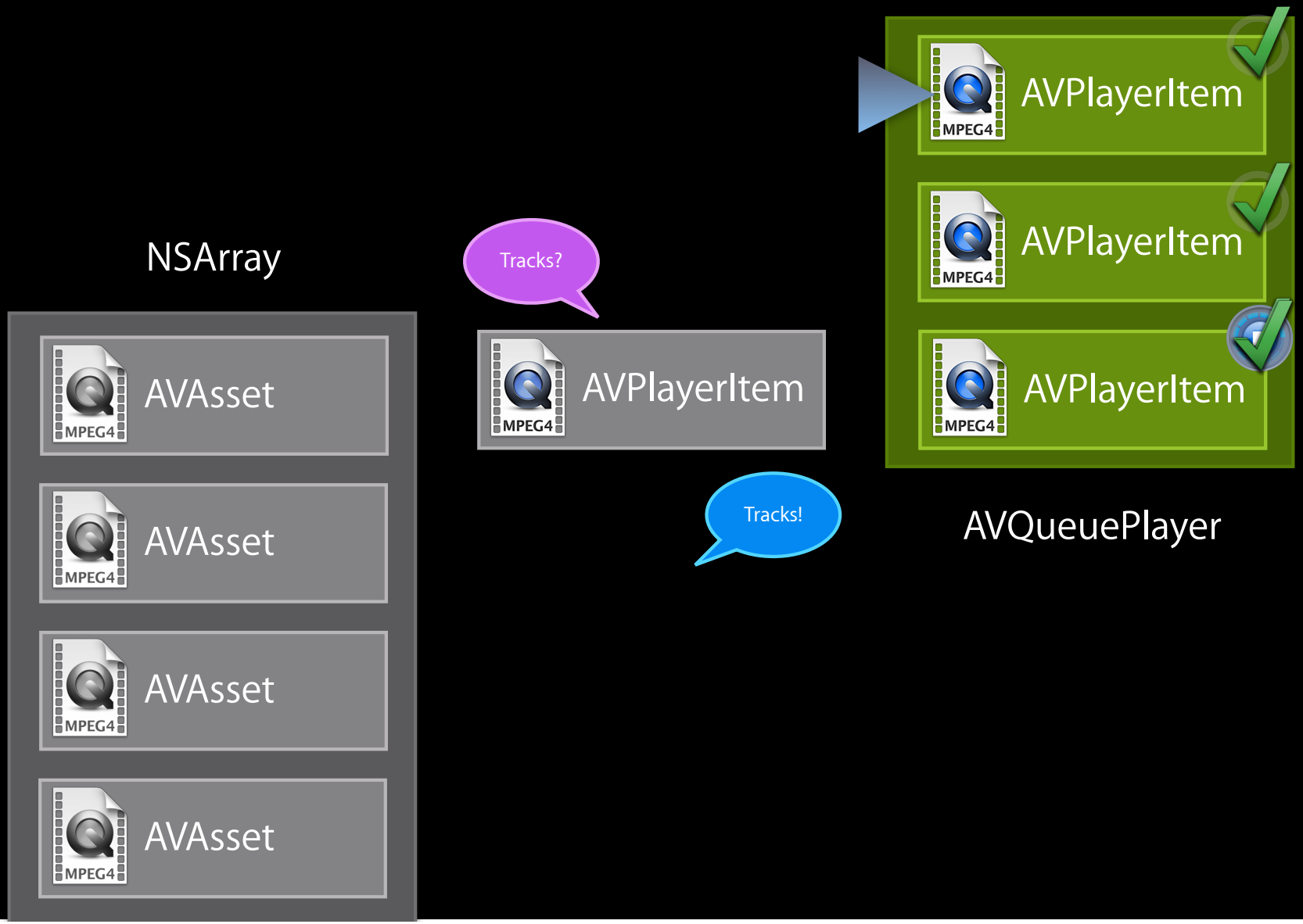


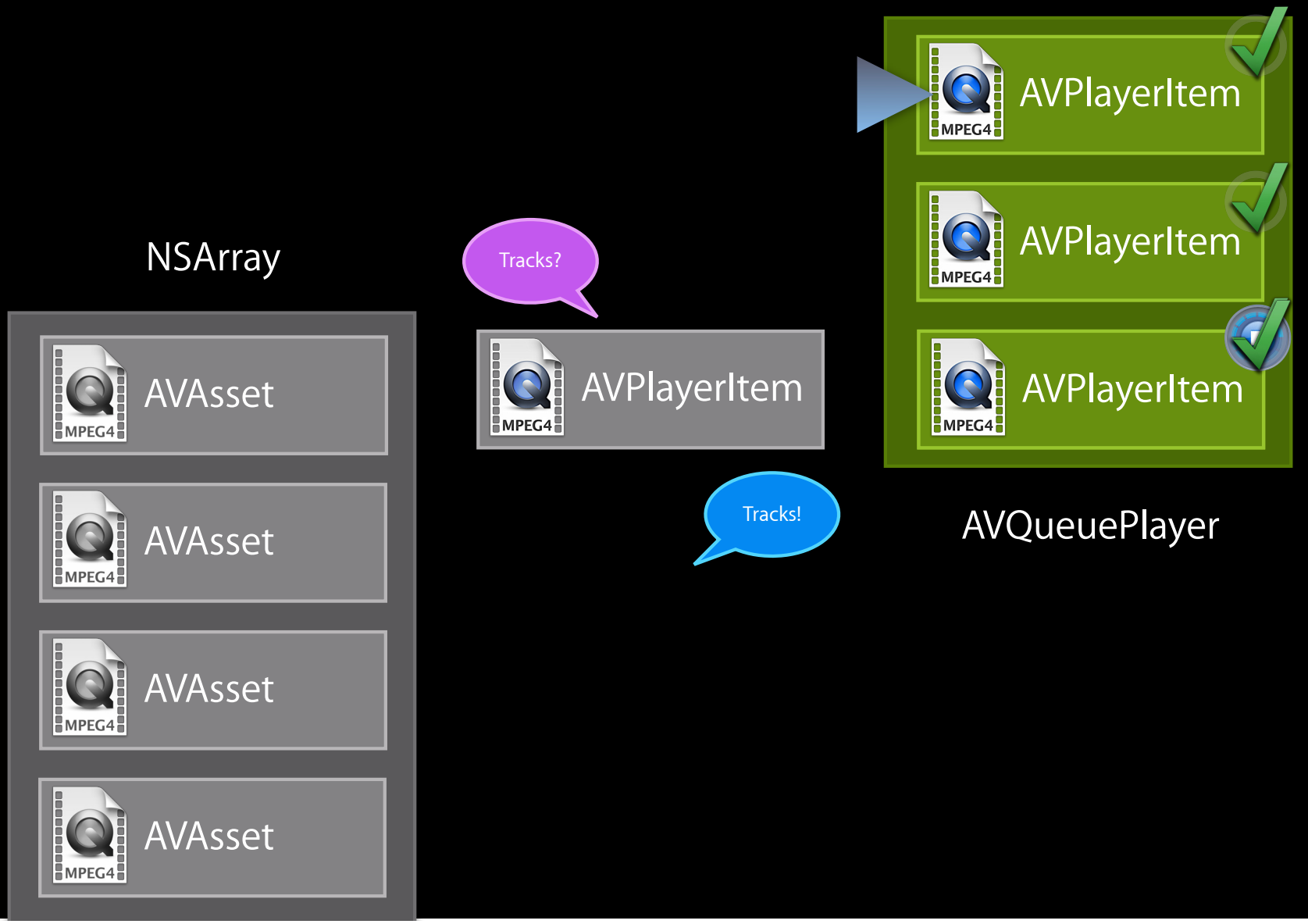
# AVQueuePlayer

## Observation via `NSObject(NSKeyValueObserving)`

- A media controller
  - Not a playlist model
    - AVQueuePlayer will prepare all media resources in the queue
  - Provides gapless audio item transition
  - Mechanism for looping

```
AVQueuePlayer *myPlayer = [AVQueuePlayer queuePlayerWithItems:myItemArray];
```


NSArray
AVAsset
AVAsset
AVAsset
AVAsset
Tracks?
AVPlayerItem
Tracks!
AVPlayerItem
AVPlayerItem
AVPlayerItem
AVQueuePlayer


NSArray
AVAsset
AVAsset
AVAsset
AVAsset
Tracks?
AVPlayerItem
Tracks!
AVPlayerItem
AVPlayerItem
AVPlayerItem
AVQueuePlayer

```
NSMutableArray *myPlayList =
                    [NSMutableArray arrayWithContentsOfFile:myPlayListFile];
...
AVPlayerItem *lastItem = nil;
for (int i = 0; i < 3; ++i) {
  if ([playList count]) {
    AVAsset *nextAsset = [myPlayList removeLastObject];
    AVKeyValueStatus status =
                      [nextAsset statusOfValueForKey:@"tracks" error:nil];
    if (status == AVKeyValueStatusLoaded) {
      AVPlayerItem *item = [AVPlayerItem playerItemWithAsset: nextAsset];
      [myPlayer insertItem:item afterItem:lastItem];
      lastItem = item;
    } // else loadValuesAsynchronouslyForKeys
  }
}
```

```
NSMutableArray *myPlayList =
                    [NSMutableArray arrayWithContentsOfFile:myPlayListFile];
...
AVPlayerItem *lastItem = nil;
for (int i = 0; i < 3; ++i) {
  if ([playList count]) {
    AVAsset *nextAsset = [myPlayList removeLastObject];
    AVKeyValueStatus status =
                      [nextAsset statusOfValueForKey:@"tracks" error:nil];
    if (status == AVKeyValueStatusLoaded) {
      AVPlayerItem *item = [AVPlayerItem playerItemWithAsset: nextAsset];
      [myPlayer insertItem:item afterItem:lastItem];
      lastItem = item;
    } // else loadValuesAsynchronouslyForKeys
  }
}
```

```
NSMutableArray *myPlayList =
                    [NSMutableArray arrayWithContentsOfFile:myPlayListFile];
...
AVPlayerItem *lastItem = nil;
for (int i = 0; i < 3; ++i) {
  if ([playList count]) {
    AVAsset *nextAsset = [myPlayList removeLastObject];
    AVKeyValueStatus status =
                      [nextAsset statusOfValueForKey:@"tracks" error:nil];
    if (status == AVKeyValueStatusLoaded) {
      AVPlayerItem *item = [AVPlayerItem playerItemWithAsset: nextAsset];
      [myPlayer insertItem:item afterItem:lastItem];
      lastItem = item;
    } // else loadValuesAsynchronouslyForKeys
  }
}
```

```
NSMutableArray *myPlayList =
                    [NSMutableArray arrayWithContentsOfFile:myPlayListFile];
...
AVPlayerItem *lastItem = nil;
for (int i = 0; i < 3; ++i) {
  if ([playList count]) {
    AVAsset *nextAsset = [myPlayList removeLastObject];
    AVKeyValueStatus status =
                      [nextAsset statusOfValueForKey:@"tracks" error:nil];
    if (status == AVKeyValueStatusLoaded) {
      AVPlayerItem *item = [AVPlayerItem playerItemWithAsset: nextAsset];
      [myPlayer insertItem:item afterItem:lastItem];
      lastItem = item;
    } // else loadValuesAsynchronouslyForKeys
  }
}
```

```
NSMutableArray *myPlayList =
                    [NSMutableArray arrayWithContentsOfFile:myPlayListFile];
...
AVPlayerItem *lastItem = nil;
for (int i = 0; i < 3; ++i) {
  if ([playList count]) {
    AVAsset *nextAsset = [myPlayList removeLastObject];
    AVKeyValueStatus status =
                      [nextAsset statusOfValueForKey:@"tracks" error:nil];
    if (status == AVKeyValueStatusLoaded) {
      AVPlayerItem *item = [AVPlayerItem playerItemWithAsset: nextAsset];
      [myPlayer insertItem:item afterItem:lastItem];
      lastItem = item;
    } // else loadValuesAsynchronouslyForKeys
  }
}
```

```
[myPlayer addObserver:self
           forKeyPath:@"currentItem"
              options:0
              context:myCurrentItemChangedContext];
```

```
[myPlayer addObserver:self
           forKeyPath:@"currentItem"
              options:0
              context:myCurrentItemChangedContext];
```

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {
   if ( context == myCurrentItemChangedContext ) {
     if ([playList count]) {
       AVPlayerItem *lastItem = [[myPlayer items] lastObject];
       AVAsset *nextAsset = [myPlayList removeLastObject];
       AVKeyValueStatus status =
                        [nextAsset statusOfValueForKey:@"tracks" error:nil];
       if (status == AVKeyValueStatusLoaded) {
         AVPlayerItem *item = [AVPlayerItem playerItemWithAsset:nextAsset];
         [myPlayer insertItem:item afterItem:lastItem];
       } // else loadValuesAsynchronouslyForKeys
     }
  }
}
```

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {
    if ( context == myCurrentItemChangedContext ) {
      if ([playList count]) {
        AVPlayerItem *lastItem = [[myPlayer items] lastObject];
        AVAsset *nextAsset = [myPlayList removeLastObject];
        AVKeyValueStatus status =
                        [nextAsset statusOfValueForKey:@"tracks" error:nil];
        if (status == AVKeyValueStatusLoaded) {
          AVPlayerItem *item = [AVPlayerItem playerItemWithAsset:nextAsset];
          [myPlayer insertItem:item afterItem:lastItem];
        } // else loadValuesAsynchronouslyForKeys
      }
  }
}
```

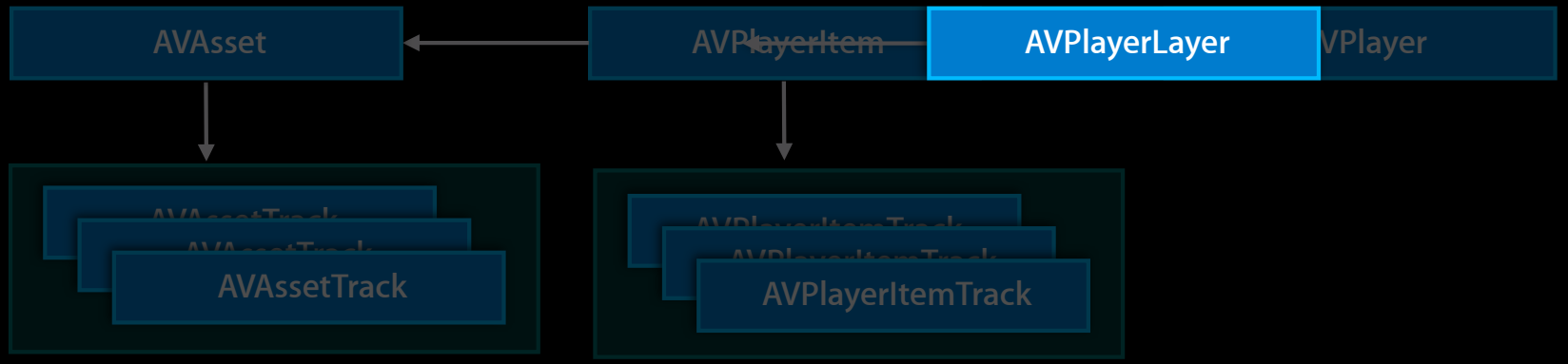
```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {
    if ( context == myCurrentItemChangedContext ) {
      if ([playList count]) {
        AVPlayerItem *lastItem = [[myPlayer items] lastObject];
        AVAsset *nextAsset = [myPlayList removeLastObject];
        AVKeyValueStatus status =
                        [nextAsset statusOfValueForKey:@"tracks" error:nil];
        if (status == AVKeyValueStatusLoaded) {
          AVPlayerItem *item = [AVPlayerItem playerItemWithAsset:nextAsset];
          [myPlayer insertItem:item afterItem:lastItem];
        } // else loadValuesAsynchronouslyForKeys
      }
    }
}
```

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {
    if ( context == myCurrentItemChangedContext ) {
      if ([playList count]) {
        AVPlayerItem *lastItem = [[myPlayer items] lastObject];
        AVAsset *nextAsset = [myPlayList removeLastObject];
        AVKeyValueStatus status =
                        [nextAsset statusOfValueForKey:@"tracks" error:nil];
        if (status == AVKeyValueStatusLoaded) {
          AVPlayerItem *item = [AVPlayerItem playerItemWithAsset:nextAsset];
          [myPlayer insertItem:item afterItem:lastItem];
        } // else loadValuesAsynchronouslyForKeys
      }
    }
}
```

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {
    if ( context == myCurrentItemChangedContext ) {
      if ([playList count]) {
        AVPlayerItem *lastItem = [[myPlayer items] lastObject];
        AVAsset *nextAsset = [myPlayList removeLastObject];
        AVKeyValueStatus status =
                        [nextAsset statusOfValueForKey:@"tracks" error:nil];
        if (status == AVKeyValueStatusLoaded) {
          AVPlayerItem *item = [AVPlayerItem playerItemWithAsset:nextAsset];
          [myPlayer insertItem:item afterItem:lastItem];
        } // else loadValuesAsynchronouslyForKeys
      }
    }
}
```

# AV Foundation

## Core playback classes



# AVPlayerLayer

## Observation via `NSObject(NSKeyValueObserving)`

- A media presentation object
  - How to display media onscreen

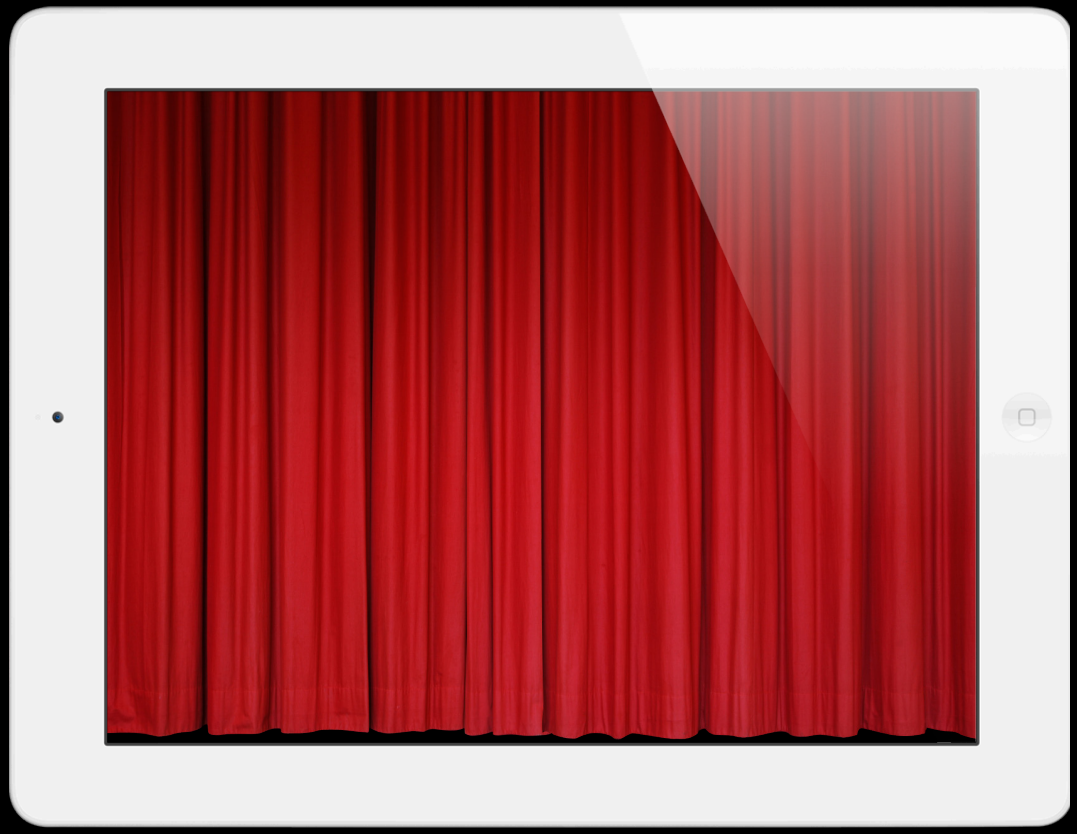
- Attach an AVPlayerLayer to your:
  - UIView
  - NSView

    `wantsLayer`
  - Or CALayer hierarchy

```
[AVPlayerLayer playerLayerWithPlayer:myPlayer];
```

# AVPlayerLayer

`readyForDisplay`



# AVPlayerLayer

## videoGravity

AVLayerVideoGravityResizeAspect AVLayerVideoGravityResize AVLayerVideoGravityResizeAspectFill

# Using AV Foundation Efficiently

## Summary of key points

# Static Models

## <AVAsynchronousKeyValueLoading>

- Useful without a media controller
- AVAsset
  - AVAssetTrack

Performs loading only as requested
and once loaded does not change

# Dynamic Models

## NSObject(NSKeyValueObserving)

- Most useful with a media controller
- AVPlayerItem
  - AVPlayerItemTrack

Once associated with a media controller
dynamic models are able to mutate, operate, and
load independently of your interaction

# Matters of protocol

## And platform etiquette

- Mac OS X: Your app experiencing the SPOD
  - Unresponsive and poor performing
- iOS: Shared media services termination
  - Affecting your app and all other running apps

# Matters of protocol

## And platform etiquette

- `<AVAsynchronousKeyValueLoading>`

  `loadValuesAsynchronouslyForKeys:completionHandler:`

  `statusOfValueForKey:error:`

- `NSObject(NSKeyValueObserving)`

  `addObserver:forKeyPath:options:context:`

  `removeObserver:forKeyPath:`

  `observeValueForKeyPath:ofObject:change:context:`

# New API

# AirPlay

# AirPlay

## Stream your content wirelessly

Only on iOS

- To enable

```
[myPlayer setAllowsAirPlayVideo:YES];
```

- KV-Observable property

```
airPlayVideoActive
```

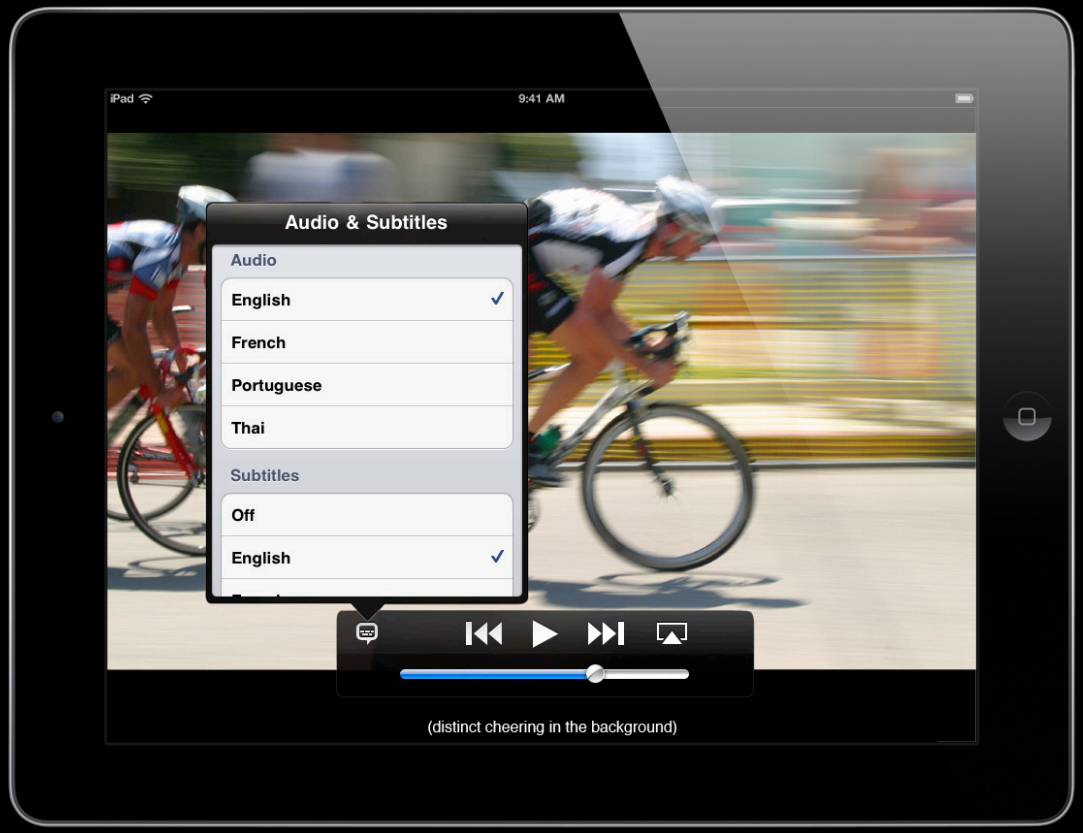
- AirPlay video during AirPlay screen

```
[myPlayer setUsesAirPlayVideoWhileAirPlayScreenIsActive:YES];
```

| AirPlay and External Displays in iOS apps | Presidio<br>Tuesday 3:15PM |
|---|---|

# Media Options

# Media Options



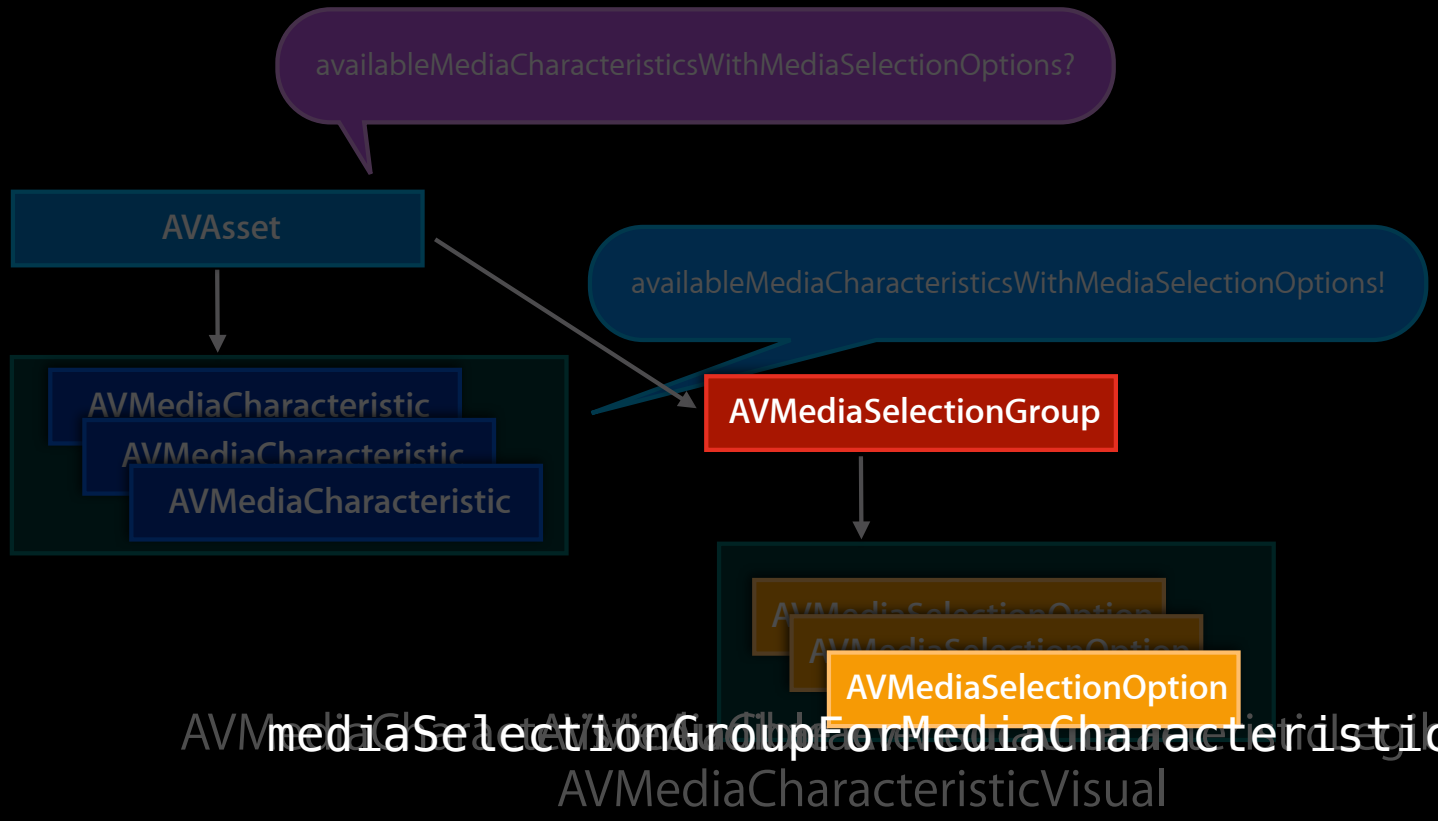
# AVMediaSelectionGroup
# AVMediaSelectionOption

Only on iOS

- Discovery and Selection of
  - Audio Language Options
  - Accessibility Options
  - Visual Media Options

# AV Foundation

## Media selection options



# AVMediaSelectionGroup
# AVMediaSelectionOption

Only on iOS

- AVMediaSelectionGroup of AVMediaSelectionOption
  - AVMediaSelectionGroup offers option filtering
    - e.g., withCharacteristics, withoutCharacteristics, playable
  - AVMediaSelectionOption details the media properties
    - e.g., mediaType, mediaCharacteristic, playable, locale
  - To select, at playback, an option within a group:

```
[myAVPlayerItem selectMediaOption:  inMediaSelectionGroup:  ];
```

```
NSArray *keys = [NSArray
arrayWithObject:@"availableMediaCharacteristicsWithMediaSelectionOptions"];
[myAsset loadValuesAsynchronouslyForKeys:keys
                       completionHandler: ^{
    AVKeyValueStatus optionsStatus = [asset statusOfValueForKey:
                 @"availableMediaCharacteristicsWithMediaSelectionOptions"
                                                        error:&error];
    switch (optionsStatus) {
        case AVKeyValueStatusLoaded:
            dispatch_async(dispatch_get_main_queue(), ^{
               [self mySelectAudioOption];
            });
            break;
        case AVKeyValueStatusFailed:
        ...
}];
```

```
NSArray *keys = [NSArray
arrayWithObject:@"availableMediaCharacteristicsWithMediaSelectionOptions"];
[myAsset loadValuesAsynchronouslyForKeys:keys
                       completionHandler: ^{
    AVKeyValueStatus optionsStatus = [asset statusOfValueForKey:
                @"availableMediaCharacteristicsWithMediaSelectionOptions"
                                                          error:&error];
    switch (optionsStatus) {
        case AVKeyValueStatusLoaded:
            dispatch_async(dispatch_get_main_queue(), ^{
               [self mySelectAudioOption];
            });
            break;
        case AVKeyValueStatusFailed:
        ...
}];
```

```
NSArray *keys = [NSArray
arrayWithObject:@"availableMediaCharacteristicsWithMediaSelectionOptions"];
[myAsset loadValuesAsynchronouslyForKeys:keys
                       completionHandler: ^{
    AVKeyValueStatus optionsStatus = [asset statusOfValueForKey:
                 @"availableMediaCharacteristicsWithMediaSelectionOptions"
                                                        error:&error];
    switch (optionsStatus) {
        case AVKeyValueStatusLoaded:
            dispatch_async(dispatch_get_main_queue(), ^{
               [self mySelectAudioOption];
            });
            break;
        case AVKeyValueStatusFailed:
        ...
}];
```

```
- (void) mySelectAudioOption {
   NSArray *availableOptions = [myAsset
                  availableMediaCharacteristicsWithMediaSelectionOptions];
   if ([availableOptions containsObject:AVMediaCharacteristicAudible]) {
      AVMediaSelectionGroup audibleGroup = [asset
   mediaSelectionGroupForMediaCharacteristic:AVMediaCharacteristicAudible];
      if (audibleGroup) {
         NSArray *options = [AVMediaSelectionGroup
                  mediaSelectionOptionsFromArray:[audibleGroup options]
                                      withLocale:myMatchingLocale];
         if ([options count])
            [myAVPlayerItem selectMediaOption:[options objectAtIndex:0]
                        inMediaSelectionGroup:audibleGroup];
      }
   }
}
```

# Chapters

# Chapters

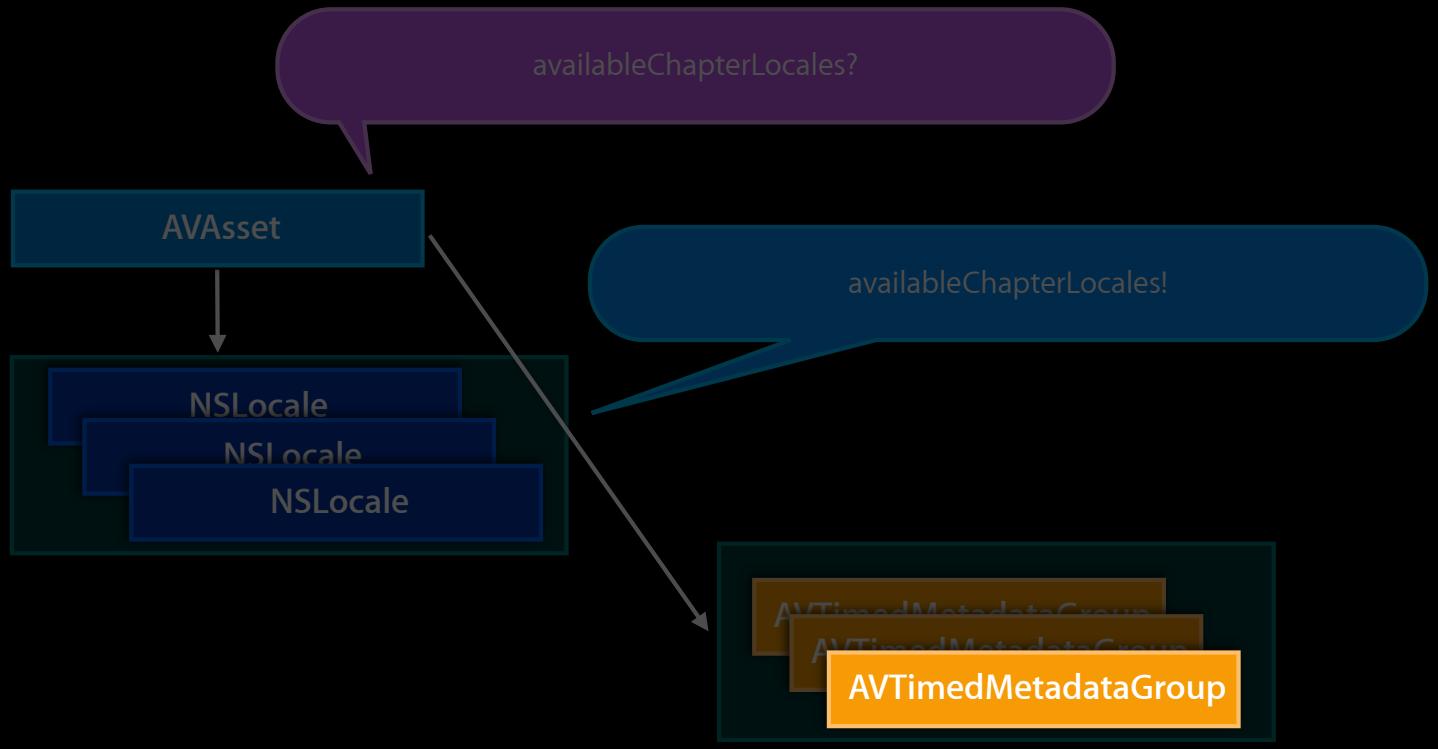

iPad
9:41 AM
Rentals
26 Days Left
Info
Chapters
The Coffee Shop (2011)
R, Appleseed, 2011 HD
Chapter 1
00:00
Chapter 2
05:33
Chapter 3
10:05
Chapter 4
17:46
Chapter 5
26:00
Chapter 6
34:35
Chapter 7
42:41
THE COFFEE SHOP
danielle BISHOP
jacob KENNEDY

# AVTimedMetadataGroup

- Discovery and selection of:
  - Chapters
  - Associated artwork

# AV Foundation

## AVTimedMetadataGroup

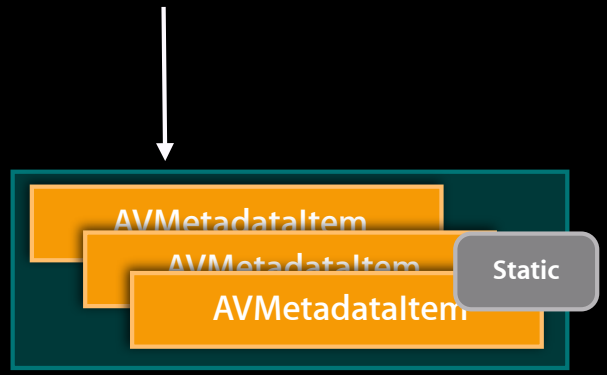



```
chapterMetadataGroups = [asset chapterMetadataGroupsBestMatchingPreferredLanguages:
```

Each AVTimedMetadataGroup instance represents a chapter

# AV Foundation

## AVTimedMetadataGroup

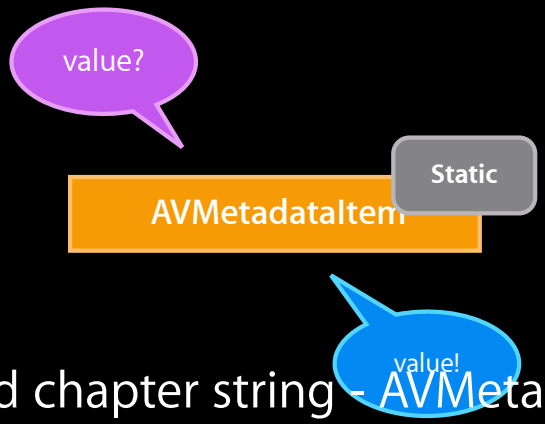



AVTimedMetadataGroup

Each AVTimedMetadataGroup instance represents a chapter

# AV Foundation

## AVTimedMetadataGroup




Value may be the localized chapter string - AVMetadataCommonKeyTitle

Value may be image data

# Media Sources

## AssetsLibrary and MediaPlayer

# iPod Library

## MediaPlayer.framework

```
MPMediaQuery *songsQuery = [MPMediaQuery songsQuery];

// Play the first song found
MPMediaItem *item = [[songsQuery items] objectAtIndex:0];
NSURL *url = (NSURL *)[item valueForProperty:MPMediaItemPropertyAssetURL];
if (url) {
   // 1. Create AVURLAsset With URL
   // 2. Load Asynchronously Value For Key @”tracks”
   // 3. Create an AVPlayerItem with AVURLAsset
   // 4. Create/Enqueue AVPlayerItem
   ...
}
```

# iPod Library

## MediaPlayer.framework

```
MPMediaQuery *songsQuery = [MPMediaQuery songsQuery];

// Play the first song found
MPMediaItem *item = [[songsQuery items] objectAtIndex:0];
NSURL *url = (NSURL *)[item valueForProperty:MPMediaItemPropertyAssetURL];
if (url) {
   // 1. Create AVURLAsset With URL
   // 2. Load Asynchronously Value For Key @"tracks"
   // 3. Create an AVPlayerItem with AVURLAsset
   // 4. Create/Enqueue AVPlayerItem
   ...
}
```

# iPod Library

## MediaPlayer.framework

```
MPMediaQuery *songsQuery = [MPMediaQuery songsQuery];

// Play the first song found
MPMediaItem *item = [[songsQuery items] objectAtIndex:0];
NSURL *url = (NSURL *)[item valueForProperty:MPMediaItemPropertyAssetURL];
if (url) {
   // 1. Create AVURLAsset With URL
   // 2. Load Asynchronously Value For Key @"tracks"
   // 3. Create an AVPlayerItem with AVURLAsset
   // 4. Create/Enqueue AVPlayerItem
   ...
}
```

# Camera Roll

## AssetsLibrary.framework

```
ALAssetsLibrary *lib = [[ALAssetsLibrary alloc] init];
[lib enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
                   usingBlock: ^(ALAssetsGroup *group, BOOL *stop) {
   [group setAssetsFilter:[ALAssetsFilter allVideos]];
   [group enumerateAssetsUsingBlock:
            ^(ALAsset *asset, NSUInteger idx, BOOL *stop) {
      NSURL *url = [[asset defaultRepresentation] url];
      // AVURLAsset creation ...
    }];
}
                failureBlock: ^(NSError *error) {
   // User access was denied
}];
```

# Camera Roll

## AssetsLibrary.framework

```
ALAssetsLibrary *lib = [[ALAssetsLibrary alloc] init];
[lib enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
                   usingBlock: ^(ALAssetsGroup *group, BOOL *stop) {
   [group setAssetsFilter:[ALAssetsFilter allVideos]];
   [group enumerateAssetsUsingBlock:
            ^(ALAsset *asset, NSUInteger idx, BOOL *stop) {
      NSURL *url = [[asset defaultRepresentation] url];
      // AVURLAsset creation ...
    }];
}
                failureBlock: ^(NSError *error) {
   // User access was denied
}];
```

# Camera Roll

## AssetsLibrary.framework

```
ALAssetsLibrary *lib = [[ALAssetsLibrary alloc] init];
[lib enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
                   usingBlock: ^(ALAssetsGroup *group, BOOL *stop) {
   [group setAssetsFilter:[ALAssetsFilter allVideos]];
   [group enumerateAssetsUsingBlock:
            ^(ALAsset *asset, NSUInteger idx, BOOL *stop) {
      NSURL *url = [[asset defaultRepresentation] url];
      // AVURLAsset creation ...
    }];
}
                failureBlock: ^(NSError *error) {
   // User access was denied
}];
```

# Camera Roll

## AssetsLibrary.framework

```
ALAssetsLibrary *lib = [[ALAssetsLibrary alloc] init];
[lib enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
                   usingBlock: ^(ALAssetsGroup *group, BOOL *stop) {
   [group setAssetsFilter:[ALAssetsFilter allVideos]];
   [group enumerateAssetsUsingBlock:
            ^(ALAsset *asset, NSUInteger idx, BOOL *stop) {
      NSURL *url = [[asset defaultRepresentation] url];
      // AVURLAsset creation ...
    }];
}
                failureBlock: ^(NSError *error) {
   // User access was denied
}];
```

# Camera Roll

## AssetsLibrary.framework

```
ALAssetsLibrary *lib = [[ALAssetsLibrary alloc] init];
[lib enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
                   usingBlock: ^(ALAssetsGroup *group, BOOL *stop) {
   [group setAssetsFilter:[ALAssetsFilter allVideos]];
   [group enumerateAssetsUsingBlock:
            ^(ALAsset *asset, NSUInteger idx, BOOL *stop) {
      NSURL *url = [[asset defaultRepresentation] url];
      // AVURLAsset creation ...
    }];
}
                failureBlock: ^(NSError *error) {
   // User access was denied
}];
```

# Related Sessions

| | |
|---|---|
| AirPlay and External Displays in iOS apps | Presidio<br>Tuesday 3:15PM |
| HTTP Live Streaming Update | Nob Hill<br>Tuesday 4:30PM |
| Working with Media in AV Foundation | Pacific Heights<br>Wednesday 2:00PM |
| Introducing AV Foundation Capture for Lion | Pacific Heights<br>Wednesday 3:15PM |
| Capturing from the Camera Using AV Foundation on iOS 5 | Pacific Heights<br>Wednesday 4:30PM |

# Labs

| | |
|---|---|
| AirPlay Lab | Graphics, Media & Games Lab B<br>Wednesday 9:00AM |
| AV Foundation Lab | Graphics, Media & Games Lab C<br>Wednesday 9:00AM |
| HTTP Live Streaming Lab | Graphics, Media & Games Lab D<br>Wednesday 9:00AM |
| QT Kit Lab | Graphics, Media & Games Lab A<br>Wednesday 9:00AM |

| | |
|---|---|
| AV Foundation Lab | Graphics, Media & Games Lab B<br>Thursday 9:00AM |
| QuickTime Lab | Graphics, Media & Games Lab D<br>Thursday 9:00AM |
| DAL Lab | Graphics, Media & Games Lab C<br>Thursday 9:00AM |

# More Information

**Eryk Vershen**
Media Technologies Evangelist
eversheN@apple.com

**Documentation**
AV Foundation Programming Guide
http://developer.apple.com/library/ios/#documentation/AudioVideo/Conceptual/AVFoundationPG/

**Apple Developer Forums**
http://devforums.apple.com


WWDC2011