

Mastering Modern Media Playback

Best practices for AVFoundation and AVKit

Session 503

Stefan Hafneger

AVKit Engineer





What You Will Learn

Introducing AVKit for iOS

- What's new in AVKit for OS X

AVFoundation and AVKit best practices

Modern Media Stack

iOS 7 and before

Media Player

AVFoundation

Core Media

Modern Media Stack

iOS 8



Media Player

AVFoundation

Core Media

Modern Media Stack

iOS 8



Media Player

AVKit

AVFoundation

Core Media

Modern Media Stack

iOS 8



Media Player

AVKit

AVFoundation

Core Media

Modern Media Stack

Same on iOS and OS X



AVKit

AVFoundation

Core Media

Introducing AVKit for iOS

Why AVKit?

iOS 7 and before

Standardized Playback UI

Access to Media Stack

Media Player

AVFoundation

Why AVKit?

iOS 7 and before

Standardized Playback UI

Access to Media Stack

Media Player



AVFoundation

Why AVKit?

iOS 7 and before

Standardized Playback UI

Access to Media Stack

Media Player



AVFoundation

Why AVKit?

iOS 7 and before

Standardized Playback UI

Access to Media Stack

Media Player



AVFoundation



Why AVKit?

iOS 7 and before

Standardized Playback UI

Access to Media Stack

Media Player



AVFoundation



Why AVKit?

iOS 7 and before

Standardized Playback UI

Access to Media Stack

Media Player








AVFoundation



AVKit + AVFoundation







Why AVKit?

iOS 7 and before

	Standardized Playback UI	Access to Media Stack
Media Player		
AVFoundation		
AVKit + AVFoundation		

Why AVKit?

iOS 7 and before

	Standardized Playback UI	Access to Media Stack
Media Player		
AVFoundation		
AVKit + AVFoundation		

AVKit for iOS



AVKit.framework



Demo

AVPlayerViewController

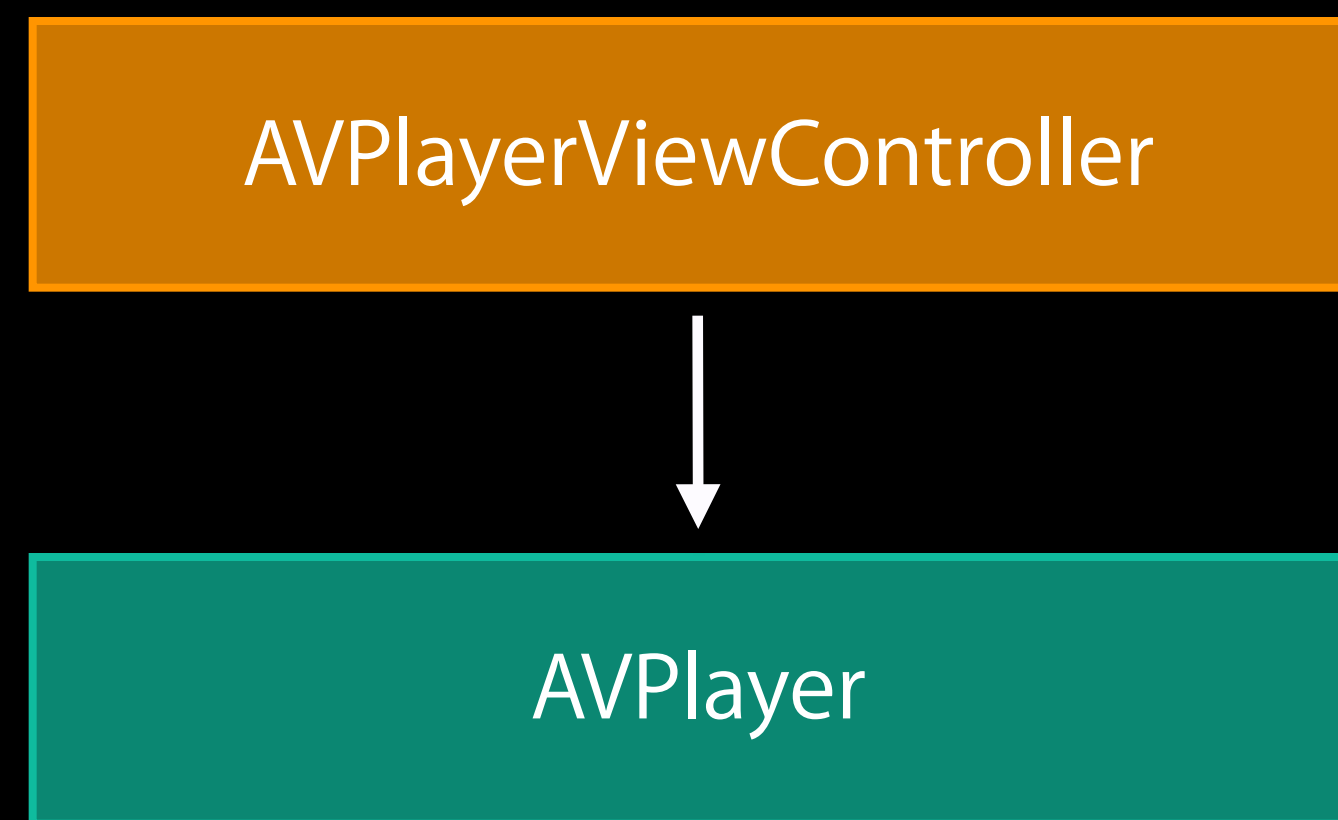
AVPlayerViewController

AVFoundation basics

AVPlayerViewController

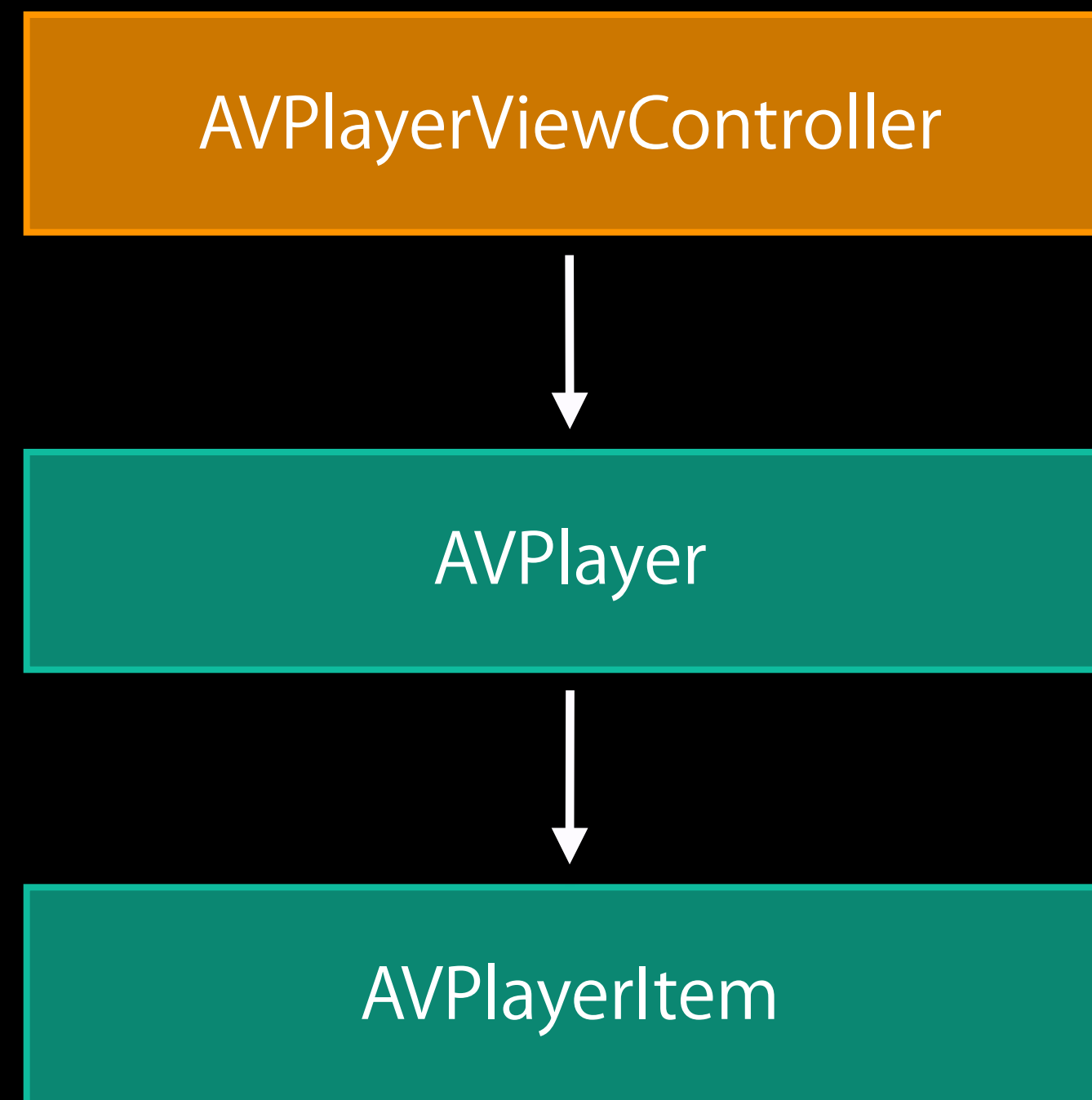
AVPlayerViewController

AVFoundation basics



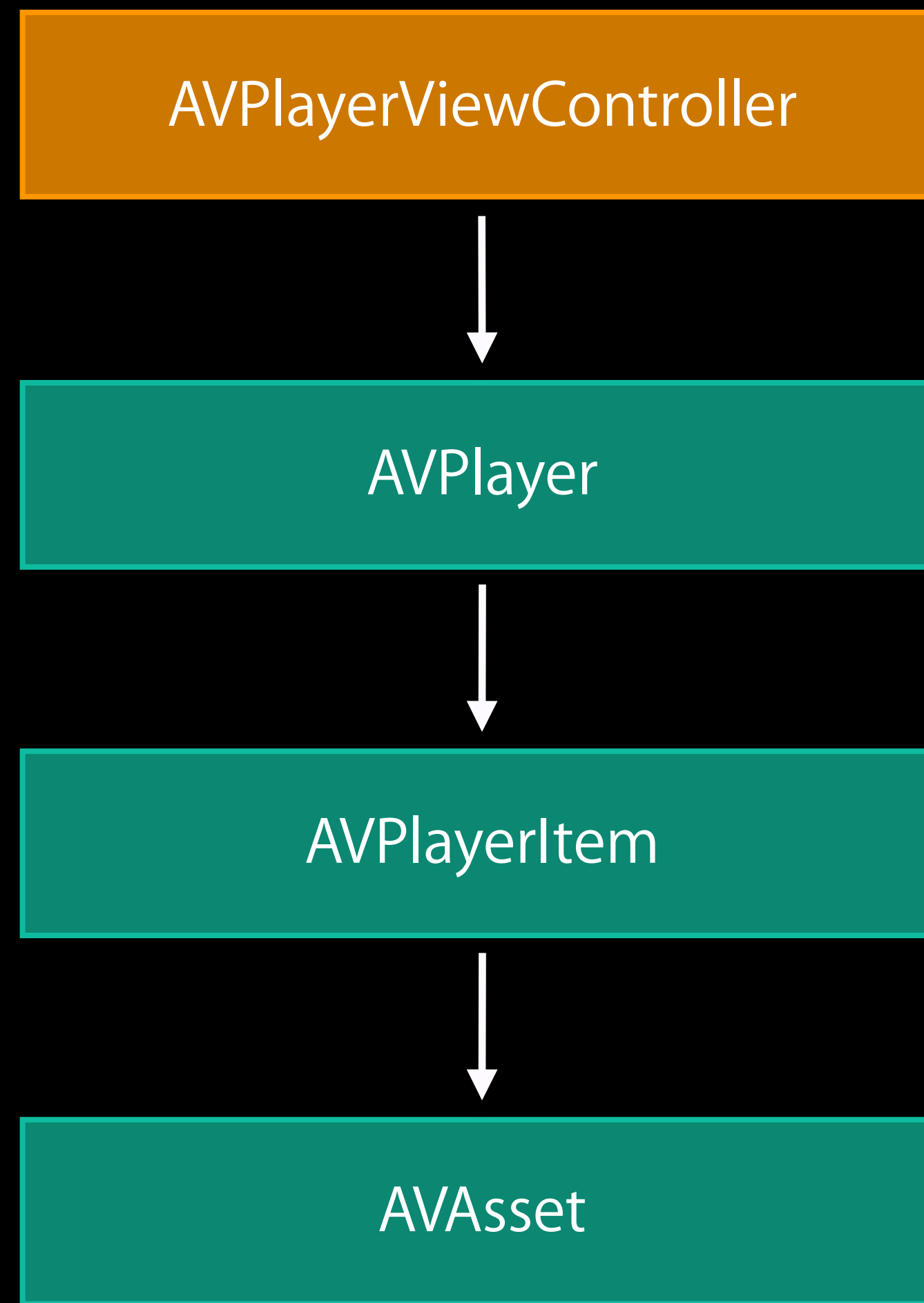
AVPlayerViewController

AVFoundation basics



AVPlayerViewController

AVFoundation basics



Providing Content

Steps to provide content for AVPlayerViewController

```
// 1. Create asset from URL.
```

```
AVAsset *asset = [AVAsset assetWithURL:URL];
```

```
// 2. Create player item for asset.
```

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
// 3. Create player with player item.
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

```
// 4. Associate player with player view controller.
```

```
playerViewController.player = player;
```

Providing Content

Steps to provide content for AVPlayerViewController

// 1. Create asset from URL.

```
AVAsset *asset = [AVAsset assetWithURL:URL];
```

// 2. Create player item for asset.

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

// 3. Create player with player item.

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

// 4. Associate player with player view controller.

```
playerViewController.player = player;
```

Providing Content

Steps to provide content for AVPlayerViewController

// 1. Create asset from URL.

```
AVAsset *asset = [AVAsset assetWithURL:URL];
```

// 2. Create player item for asset.

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

// 3. Create player with player item.

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

// 4. Associate player with player view controller.

```
playerViewController.player = player;
```

Providing Content

Steps to provide content for AVPlayerViewController

// 1. Create asset from URL.

```
AVAsset *asset = [AVAsset assetWithURL:URL];
```

// 2. Create player item for asset.

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

// 3. Create player with player item.

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

// 4. Associate player with player view controller.

```
playerViewController.player = player;
```

Providing Content

One step to provide content for *AVPlayerViewController*

```
// All four steps in one line of code.
```

```
playerViewController.player = [AVPlayer playerWithURL:URL];
```

AVPlayerViewController Features

Mostly identical to *MPMoviePlayerController*

Adaptive playback controls

- Embedded or full screen
- Option to hide playback controls

Dynamic playback controls

- Chapter navigation
- Media selection
- Streaming playback controls
- AirPlay and HDMI



9:41 AM

100%

[← Movies](#)

Vacation 2014

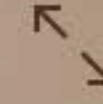
[Edit](#)



3:29



-7:17



Title

Longboarding in Utah

Creator

John Appleseed

Camera

iPhone 5s

Duration

00:10:46

Dimensions

1920 x 1080

Format

H.264, AAC



Favorites



Search



Downloads



More



9:41 AM

100% 

Done 3:29



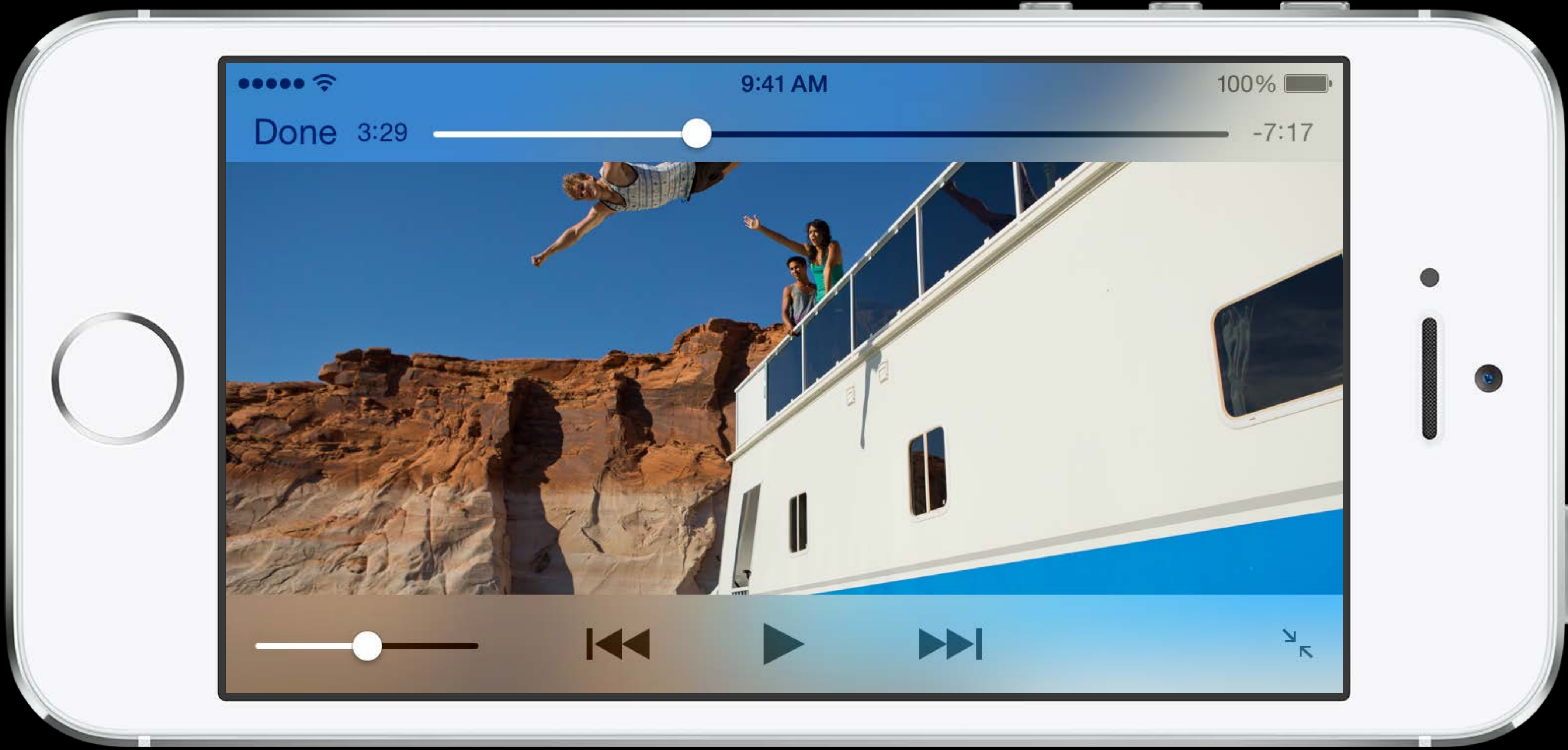
-7:17












9:41 AM

100% 

Done 3:29



-7:17







9:41 AM

100%

Done 3:29



-7:17



Grandiose Aussicht!





9:41 AM

100%

Done

Live Broadcast





AVPlayerViewController Features

Compared to *MPMoviePlayerController*

MPMoviePlayerController

AVPlayerViewController

Embedded and full screen playback controls

Embedded and full screen playback controls

Manual controls style selection

Automatic controls style selection

Hide playback controls

Hide playback controls

Chapter navigation and media selection

Chapter navigation and media selection

AirPlay and HDMI support

AirPlay and HDMI support

Localized and accessible

Localized and accessible

AVPlayerViewController Features

Compared to MPMoviePlayerController

MPMoviePlayerController

AVPlayerViewController

Embedded and full screen playback controls

Embedded and full screen playback controls

Manual controls style selection

Automatic controls style selection

Hide playback controls

Hide playback controls

Chapter navigation and media selection

Chapter navigation and media selection

AirPlay and HDMI support

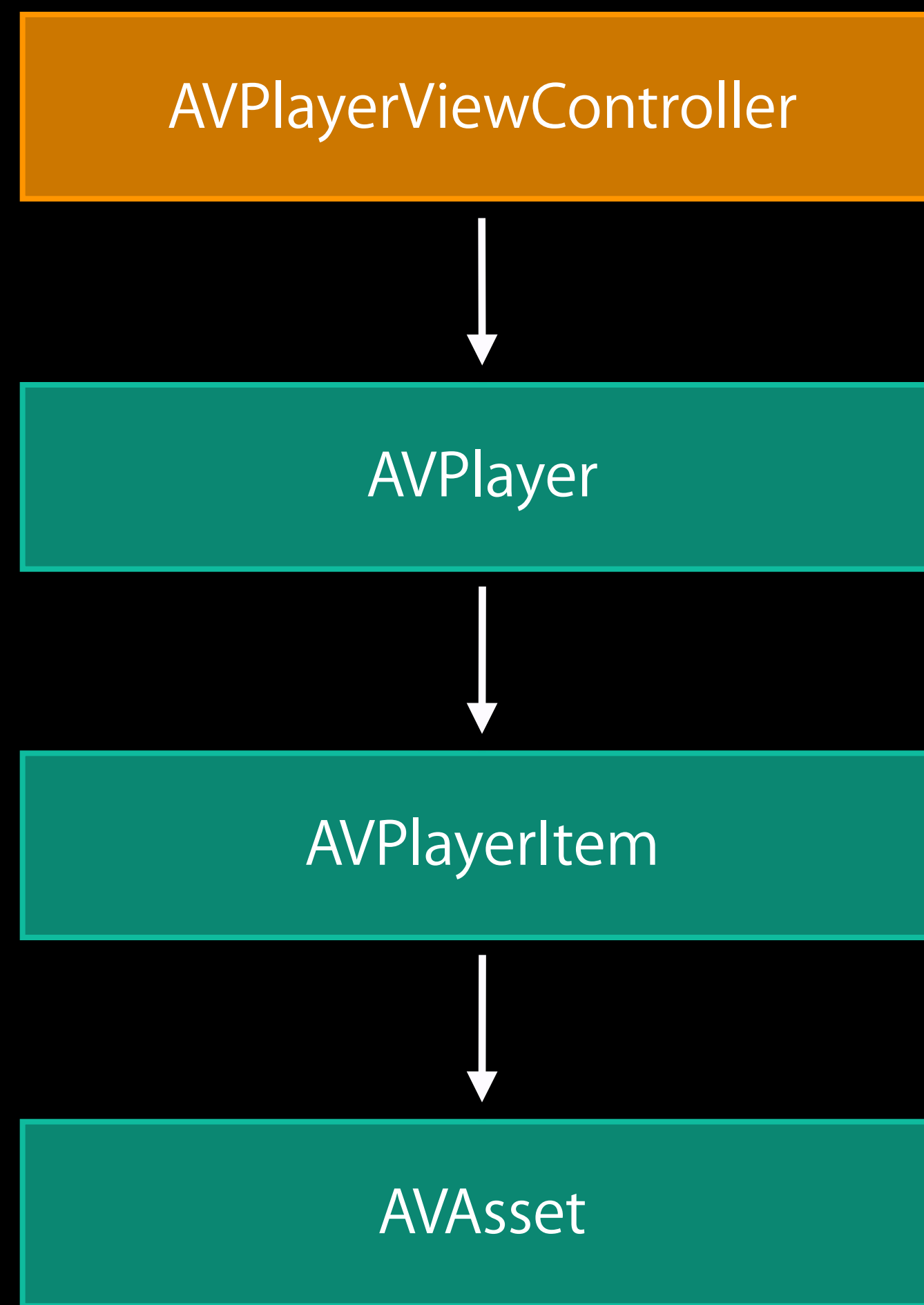
AirPlay and HDMI support

Localized and accessible

Localized and accessible

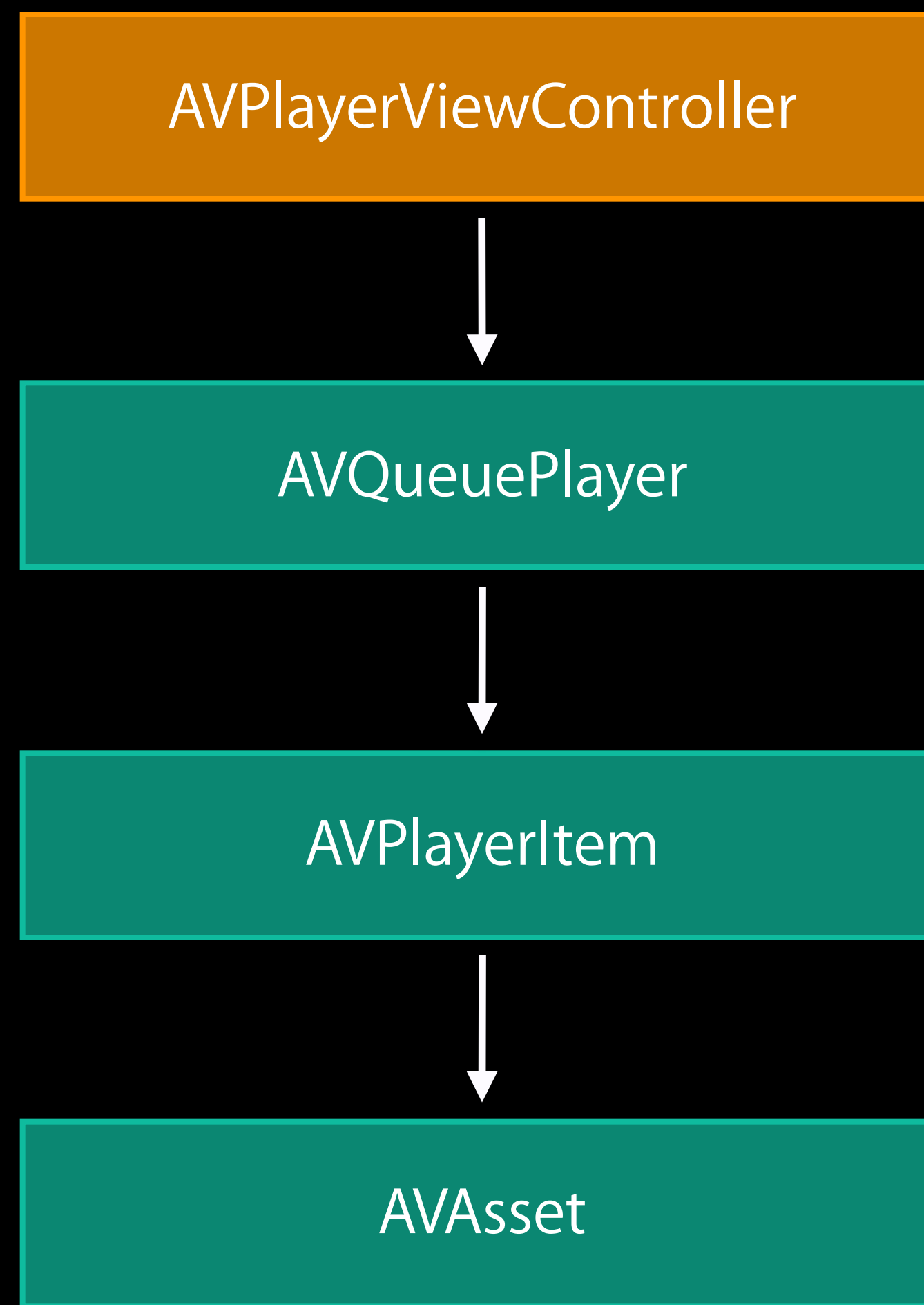
AVPlayerViewController

AVQueuePlayer



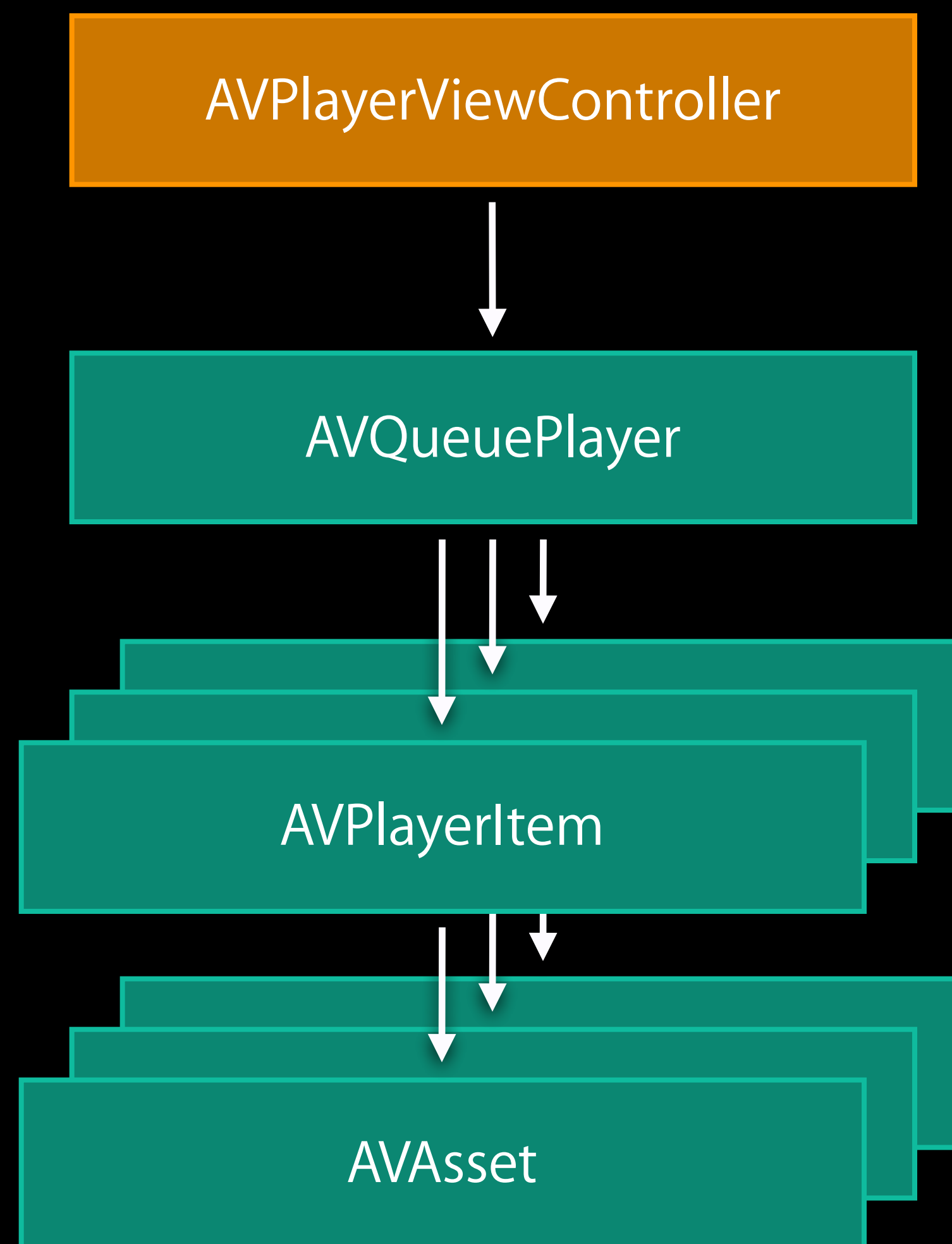
AVPlayerViewController

AVQueuePlayer



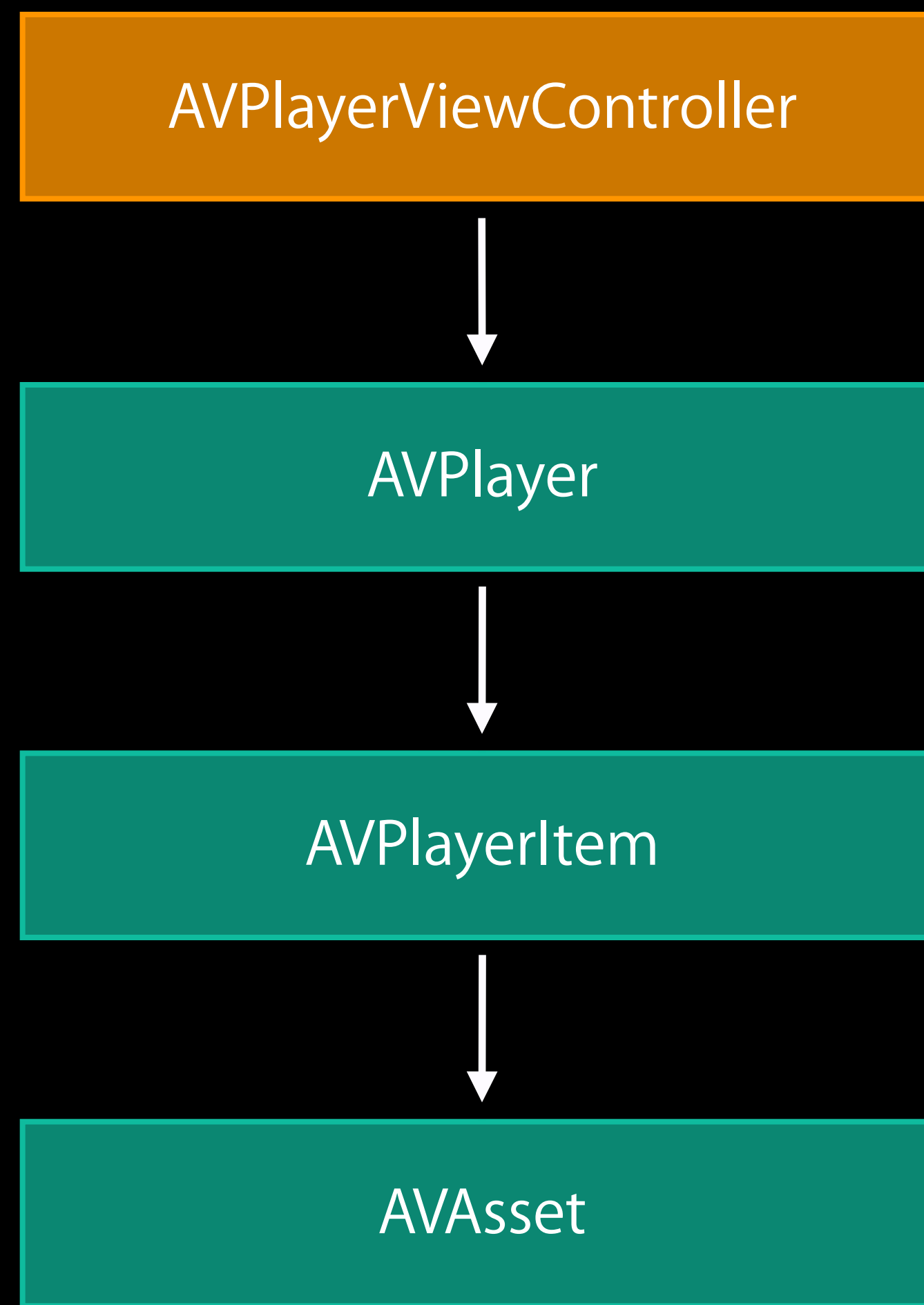
AVPlayerViewController

AVQueuePlayer



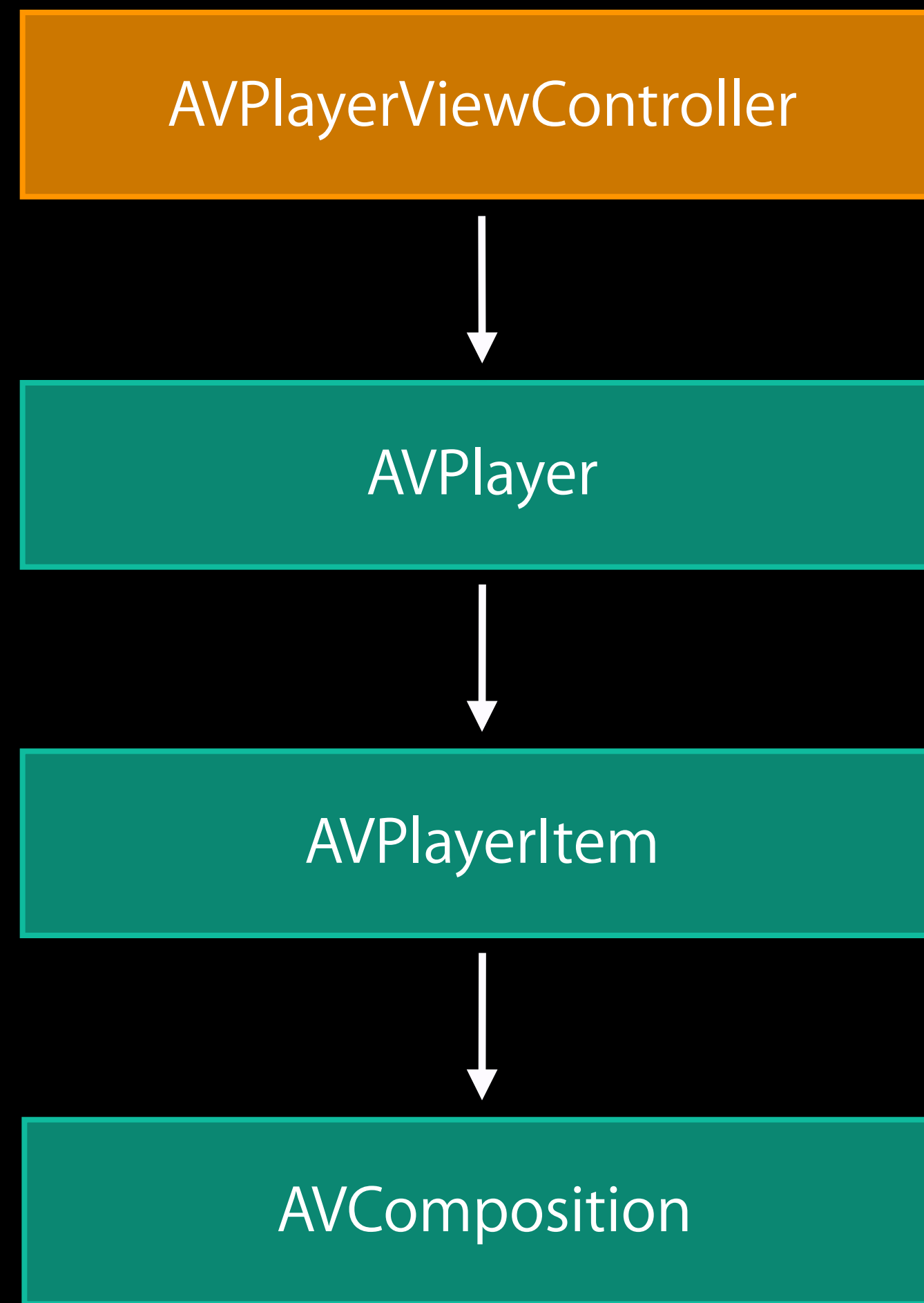
AVPlayerViewController

AVComposition



AVPlayerViewController

AVComposition



AVFoundation

Beyond basic media playback

Audio/video effects

- `AVComposition`
- `AVVideoComposition`
- `AVAudioMix`

Audio analysis/visualization

- `MTAudioProcessingTap`

...

AVFoundation

Beyond basic media playback

Audio/video effects

- `AVComposition`
- `AVVideoComposition`
- `AVAudioMix`

Audio analysis/visualization

- `MTAudioProcessingTap`

...

WWDC 2013 Advanced Editing with AVFoundation

AVPlayerViewController Features

Compared to *MPMoviePlayerController*

MPMoviePlayerController

AVPlayerViewController

Embedded and full screen playback controls

Embedded and full screen playback controls

Manual controls style selection

Automatic controls style selection

Hide playback controls

Hide playback controls

Chapter navigation and media selection

Chapter navigation and media selection

AirPlay and HDMI support

AirPlay and HDMI support

Localized and accessible

Localized and accessible

AVPlayerViewController Features

Compared to *MPMoviePlayerController*

MPMoviePlayerController

AVPlayerViewController

Embedded and full screen playback controls

Embedded and full screen playback controls

Manual controls style selection

Automatic controls style selection

Hide playback controls

Hide playback controls

Chapter navigation and media selection

Chapter navigation and media selection

AirPlay and HDMI support

AirPlay and HDMI support

Localized and accessible

Localized and accessible

AVPlayerViewController Features

Compared to MPMoviePlayerController

MPMoviePlayerController

Embedded and full screen playback controls

Manual controls style selection

Hide playback controls

Chapter navigation and media selection

AirPlay and HDMI support

Localized and accessible

AVPlayerViewController

Embedded and full screen playback controls

Automatic controls style selection

Hide playback controls

Chapter navigation and media selection

AirPlay and HDMI support

Localized and accessible

Full access to media stack

Optimized playback queues

Video effects and audio visualization

Smooth scrubbing UI

Demo

AVMovieLibrary with Effects

Adopting `AVPlayerViewController`

Transitioning from `AVPlayerLayer`

Use `AVPlayerViewController`'s view instead of `AVPlayerLayer`-backed `UIView`

Set `player` property on `AVPlayerViewController` instead of `AVPlayerLayer`

Use `AVFoundation` API as before

Adopting AVPlayerViewController

Transitioning from MPMoviePlayerViewController

```
// Create movie player view controller with content URL.  
MPMoviePlayerViewController *moviePlayerViewController =  
    [[MPMoviePlayerViewController alloc] initWithContentURL:url];
```

...turns into...

```
// Create player view controller and player with URL.  
AVPlayerViewController *playerViewController =  
    [[AVPlayerViewController alloc] init];  
playerViewController.player = [AVPlayer playerWithURL:url];
```


Adopting AVPlayerViewController

Transitioning from MPMoviePlayerViewController

```
// Create movie player view controller with content URL.  
MPMoviePlayerViewController *moviePlayerViewController =  
    [[MPMoviePlayerViewController alloc] initWithContentURL:url];
```

...turns into...

```
// Create player view controller and player with URL.  
AVPlayerViewController *playerViewController =  
    [[AVPlayerViewController alloc] init];  
playerViewController.player = [AVPlayer playerWithURL:url];
```

Adopting AVPlayerViewController

Transitioning from MPMoviePlayerController

View-specific API

- AVPlayerViewController

Controller-specific API

- AVPlayer and AVPlayerItem

Adopting AVPlayerViewController

Transitioning from MPMoviePlayerController

View-specific API

- AVPlayerViewController

Controller-specific API

- AVPlayer and AVPlayerItem

Things to watch out for

- Playback controls style dynamic
- Does not auto play by default

AVKit for iOS

Wrap up

UI-level Cocoa Touch framework for AVFoundation

Standardized playback controls and behaviors

Full access to powerful modern media stack

Consider adopting `AVPlayerViewController`

What's New in AVKit for OS X

Highlights

User interface refresh



What's New in AVKit for OS X

Highlights

User interface refresh

- Automatic for `AVPlayerView` clients



What's New in AVKit for OS X

Highlights

User interface refresh

- Automatic for `AVPlayerView` clients

Updated behaviors to match iOS



What's New in AVKit for OS X

Highlights

User interface refresh

- Automatic for `AVPlayerView` clients

Updated behaviors to match iOS



What's New in AVKit for OS X

Highlights

User interface refresh

- Automatic for `AVPlayerView` clients

Updated behaviors to match iOS

WWDC 2013 Moving to AVKit and AVFoundation



What's New in AVKit for OS X

Highlights

User interface refresh

- Automatic for `AVPlayerView` clients

Updated behaviors to match iOS

New class `AVCaptureView`



AVFoundation and AVKit Best Practices

Shalini Sahoo

AVFoundation Engineer

Motivation

Benefits of best practices

Responsive UI

Robustness

Efficient use of network

Improved battery life

Modern Media Stack

iOS and OS X

An orange rectangular box containing the text "AVKit".

AVKit

An orange rectangular box containing the text "AVFoundation".

AVFoundation

A teal rectangular box containing the text "Core Media".

Core Media

Modern Media Stack

iOS and OS X

AVKit

The diagram illustrates the Modern Media Stack for iOS and OS X. It consists of three stacked layers. The top layer is AVKit, represented by an orange rectangle. Below it is AVFoundation, also represented by an orange rectangle. The bottom layer is Core Media, represented by a teal rectangle. A horizontal dashed white line is positioned between the AVKit and AVFoundation layers.

AVFoundation

Core Media

WWDC 2011 Exploring AVFoundation

Inspection vs. Playback

AVAsset

Loads property values on demand

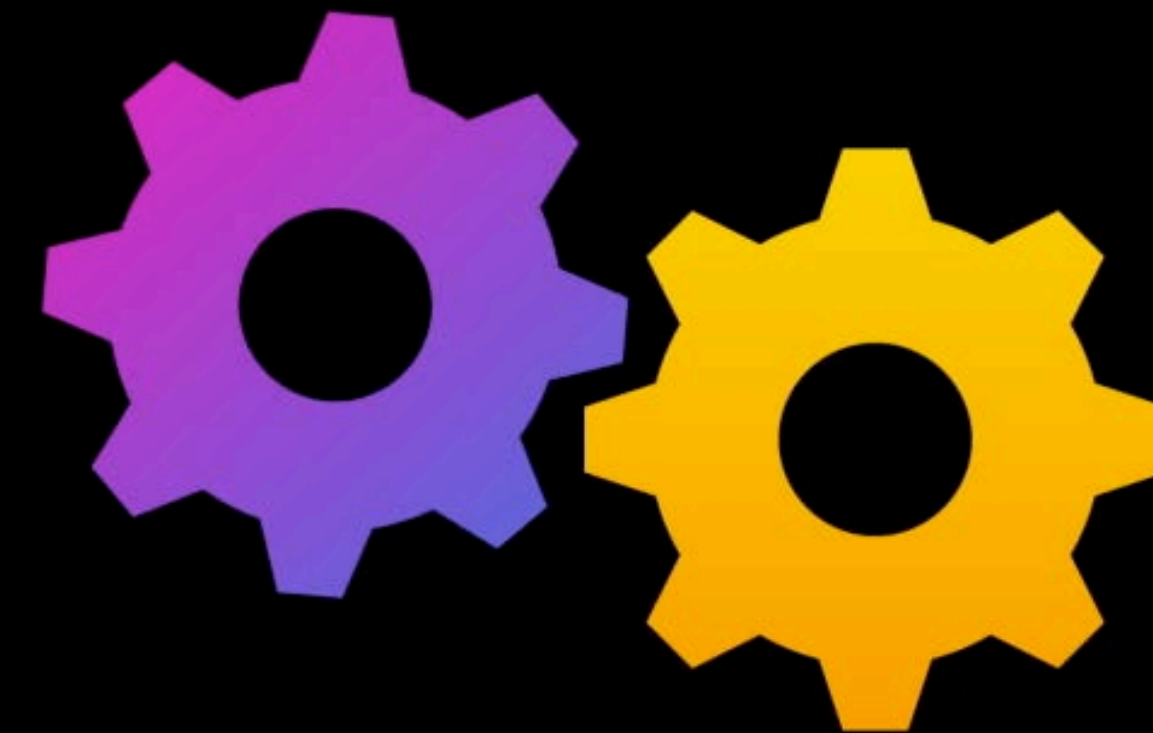


Inspection vs. Playback

AVAsset



Loads property values on demand



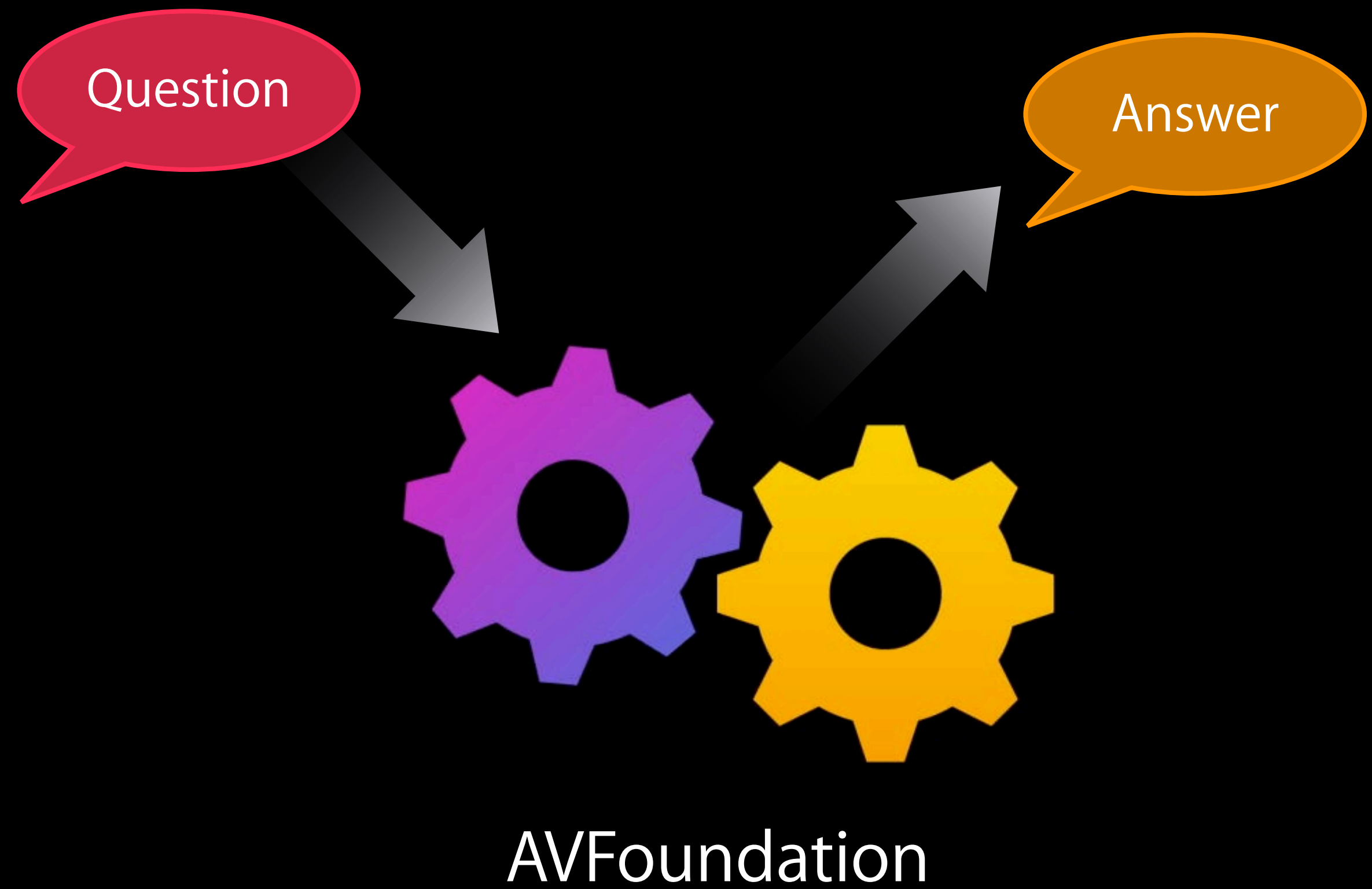
AVFoundation

Inspection vs. Playback

AVAsset



Loads property values on demand



Inspection vs. Playback

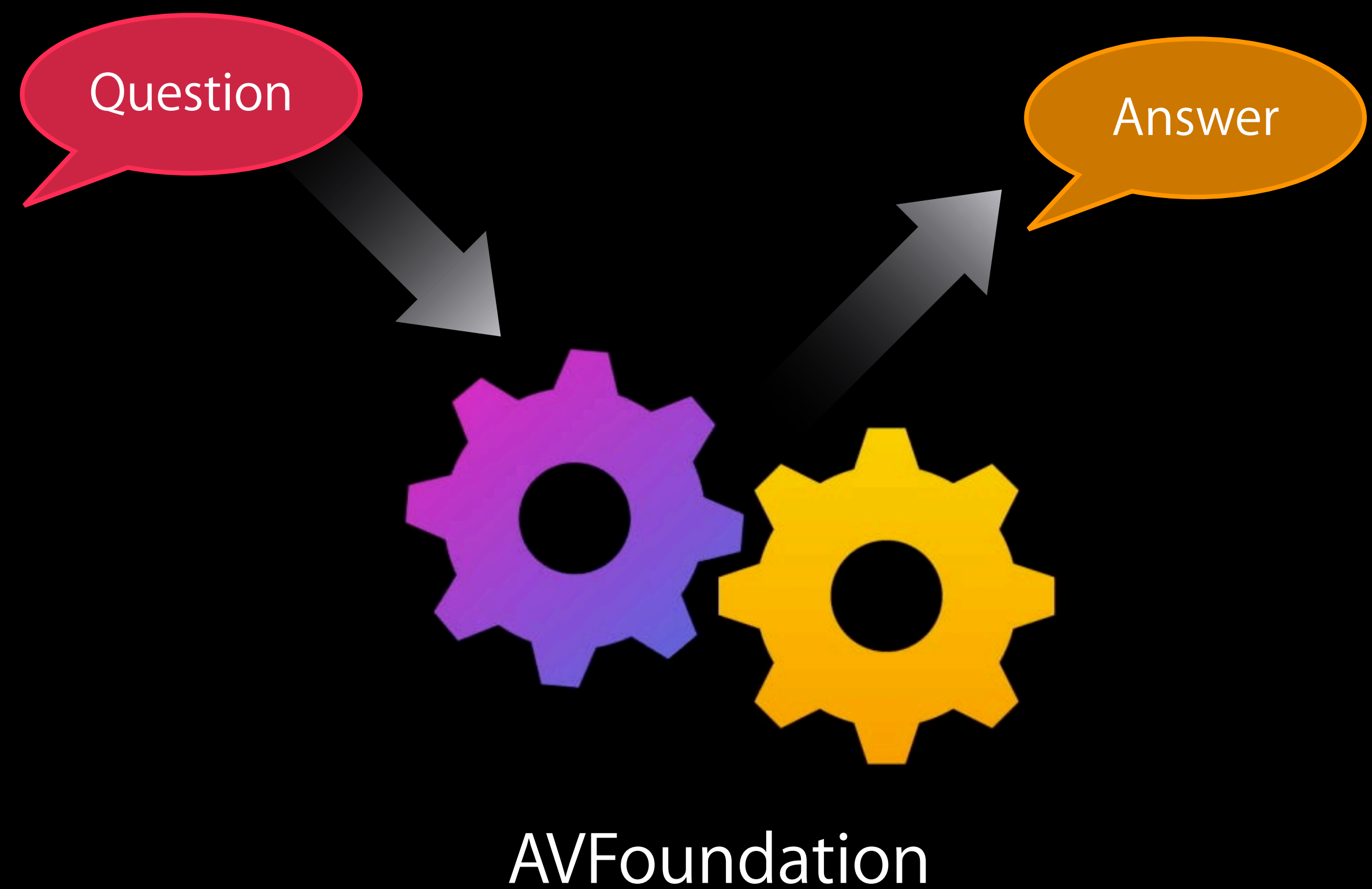
AVAsset



Loads property values on demand

Best practice

- Request values asynchronously first



Inspection vs. Playback

AVPlayer and AVPlayerItem



Playback engine loads and changes values on its own

Once prepared for playback you can read properties

Inspection vs. Playback

AVPlayer and AVPlayerItem



Playback engine loads and changes values on its own

Once prepared for playback you can read properties



AVFoundation

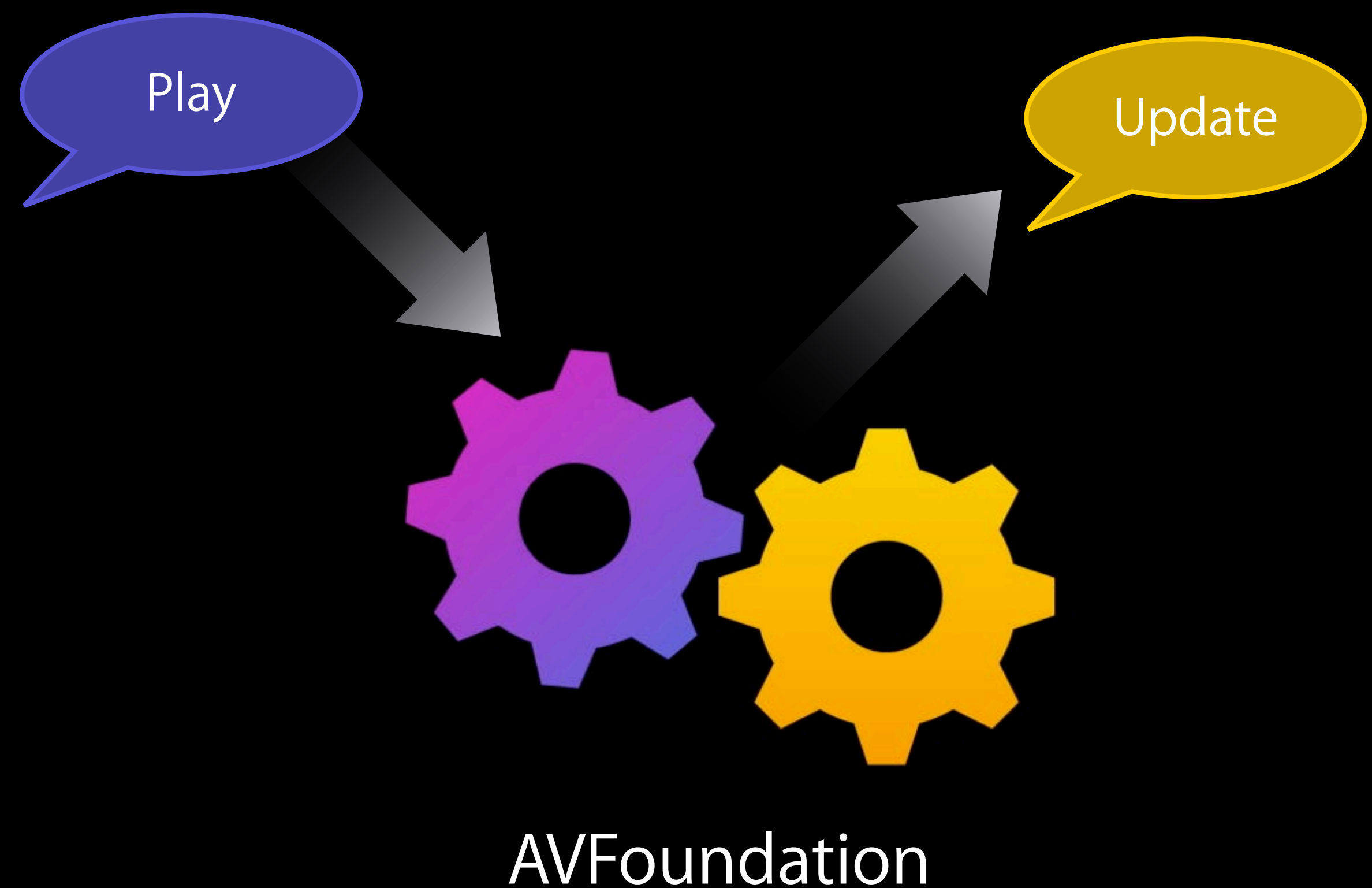
Inspection vs. Playback

AVPlayer and AVPlayerItem



Playback engine loads and changes values on its own

Once prepared for playback you can read properties



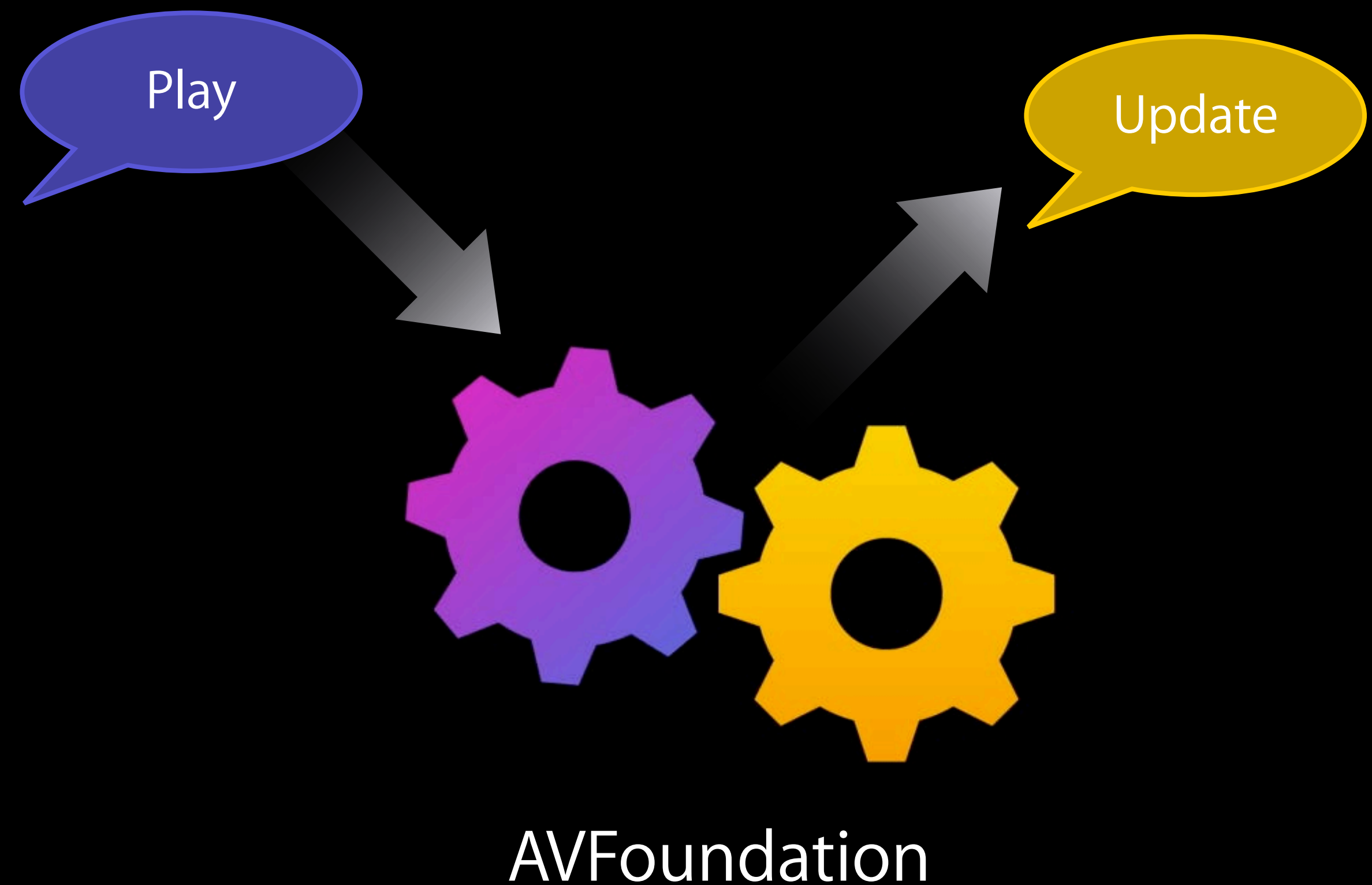
Inspection vs. Playback

AVPlayer and AVPlayerItem



Playback engine loads and changes values on its own

Once prepared for playback you can read properties



Inspection vs. Playback

AVPlayer and AVPlayerItem

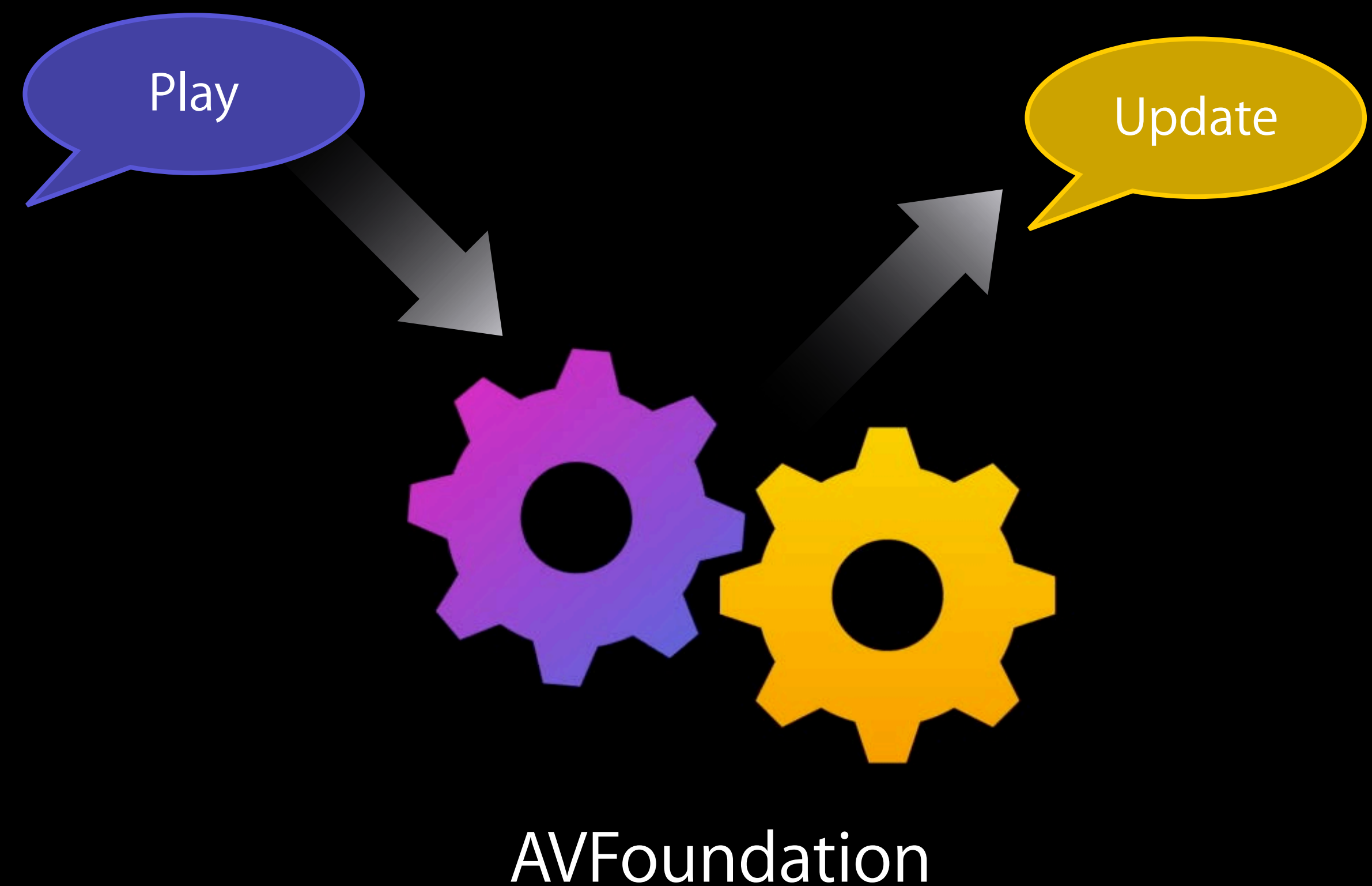


Playback engine loads and changes values on its own

Once prepared for playback you can read properties

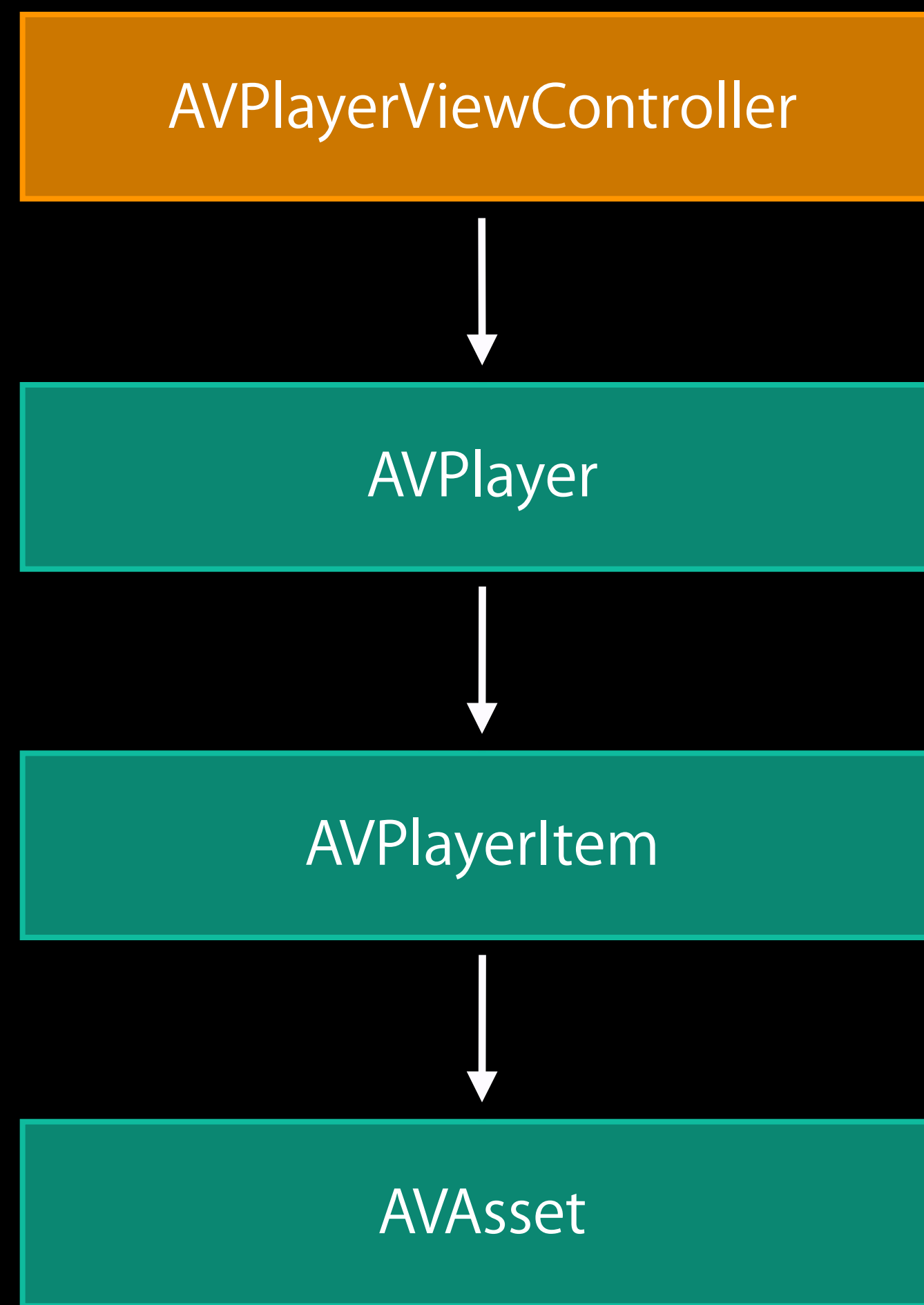
Best practice

- Use KVO to observe changing values



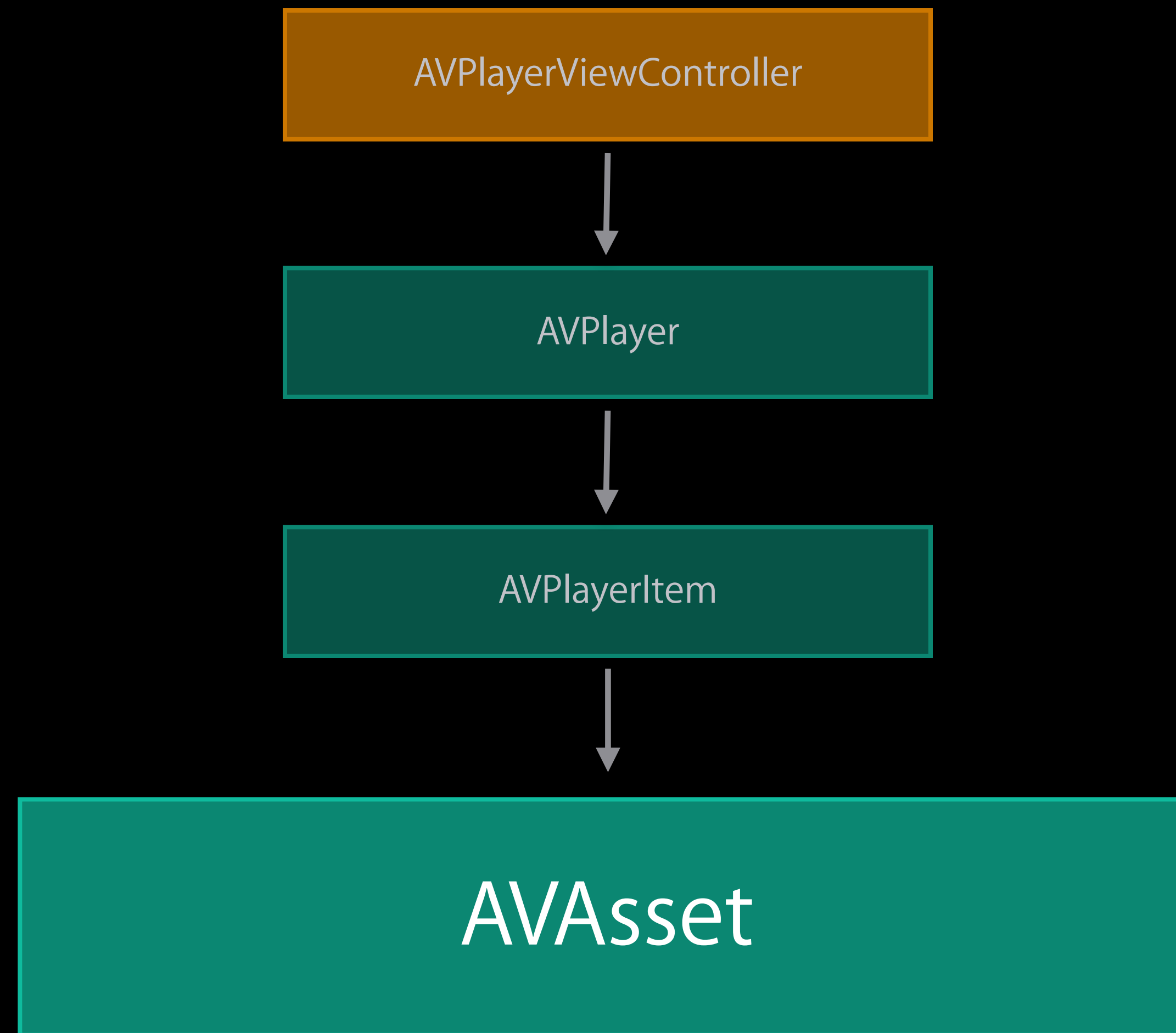
Best Practices

Roadmap



Best Practices

Roadmap



AVAsset

Inspection via *AVAsynchronousKeyValueLoading*

Load properties and wait to be loaded before calling a getter

AVAsset

Inspection via *AVAsynchronousKeyValueLoading*

Load properties and wait to be loaded before calling a getter



Duration?

AVAsset

Inspection via `AVAsynchronousKeyValueLoading`

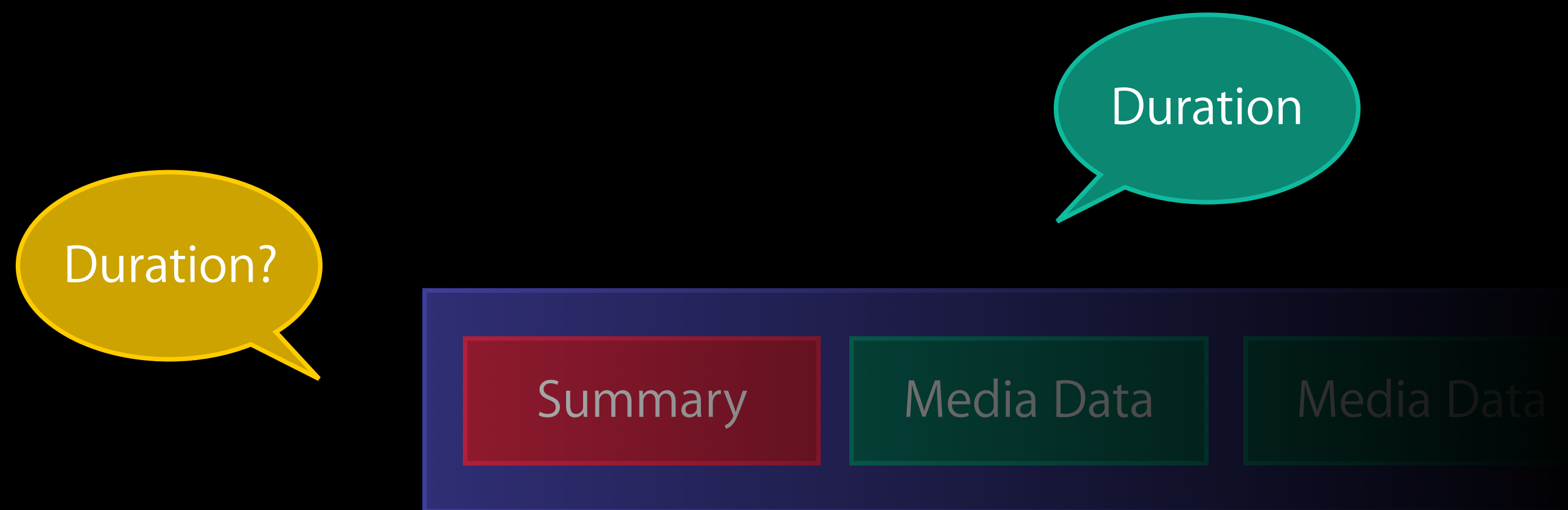
Load properties and wait to be loaded before calling a getter



AVAsset

Inspection via AVAsynchronousKeyValueLoading

Load properties and wait to be loaded before calling a getter



AVAsset

Inspection via *AVAsynchronousKeyValueLoading*

Load properties and wait to be loaded before calling a getter

AVAsset

Inspection via *AVAsynchronousKeyValueLoading*

Load properties and wait to be loaded before calling a getter



Duration?

AVAsset

Inspection via `AVAsynchronousKeyValueLoading`

Load properties and wait to be loaded before calling a getter

Duration?

Media Data

Media Data

Media Data

Media Data

Media Data

AVAsset

Inspection via `AVAsynchronousKeyValueLoading`

Load properties and wait to be loaded before calling a getter

Duration?

Media Data

Media Data

Media Data

Media Data

Media Data

Duration

Why Load Asynchronously?



Why Load Asynchronously?

On OS X

- Loading synchronously on the main thread makes app unresponsive



Why Load Asynchronously?

On OS X

- Loading synchronously on the main thread makes app unresponsive

On iOS

- Loading synchronously on **any thread** can cause mediaserver to time out and terminate
- Affecting your app and every other app



AVAsset

Inspection via *AVAsynchronousKeyValueLoading*

Load properties and wait to be loaded before calling a getter

Only load properties you need

Request properties needed in batches

AVAsset

Inspection via AVAsynchronousKeyValueLoading

Load properties and wait to be loaded before calling a getter

Only load properties you need

Request properties needed in batches

```
– (void)loadValuesAsynchronouslyForKeys:(NSArray *)keys  
    completionHandler:(void (^)(void))handler;
```

AVAsset

Inspection via AVAsynchronousKeyValueLoading

Load properties and wait to be loaded before calling a getter

Only load properties you need

Request properties needed in batches

```
- (void)loadValuesAsynchronouslyForKeys:(NSArray *)keys  
    completionHandler:(void (^)(void))handler;
```

No longer need to load @"tracks" before playback

AVAsset

Inspection via AVAsynchronousKeyValueLoading

```
[asset loadValuesAsynchronouslyForKeys:@[@"playable"] completionHandler:^(
    switch ([asset statusOfValueForKey:@"playable" error:&error]) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
    }
)];
```


AVAsset

Inspection via AVAsynchronousKeyValueLoading

```
[asset loadValuesAsynchronouslyForKeys:@[@"playable"] completionHandler:^(
    switch ([asset statusOfValueForKey:@"playable" error:&error]) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
    }
}];
```

AVAsset

Inspection via AVAsynchronousKeyValueLoading

```
[asset loadValuesAsynchronouslyForKeys:@[@"playable"] completionHandler:^(
    switch ([asset statusOfValueForKey:@"playable" error:&error]) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
    }
}];
```

AVAsset

Inspection via AVAsynchronousKeyValueLoading

```
[asset loadValuesAsynchronouslyForKeys:@[@"playable"] completionHandler:^(
    switch ([asset statusOfValueForKey:@"playable" error:&error]) {
        case AVKeyValueStatusLoaded:
            [self updateUIForAsset];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        ...
    }
}];
```

AVAsset

Best practices



Load only keys you are interested in

Use `-loadValuesAsynchronouslyForKeys:completionHandler:`

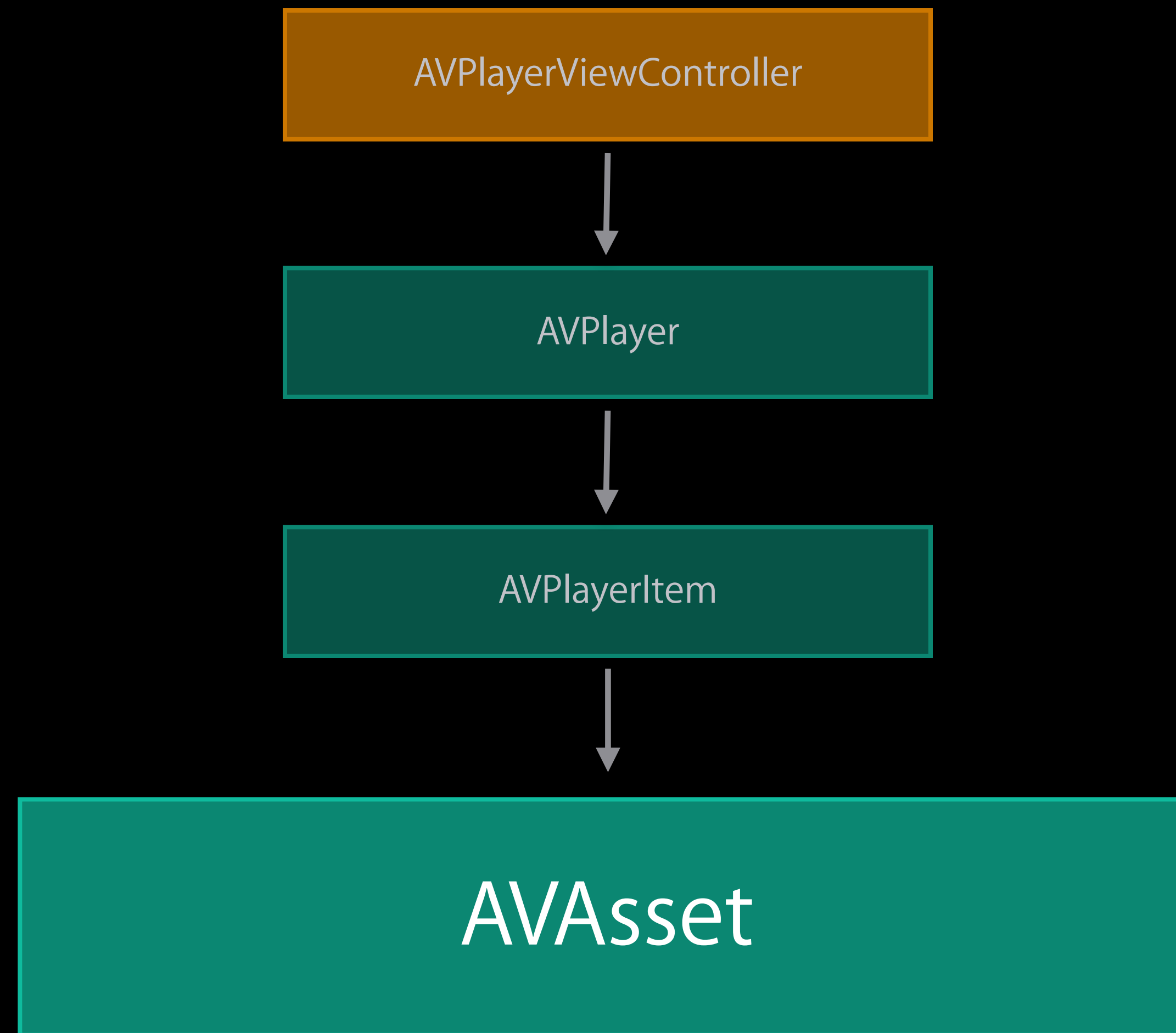
Check for status of keys in the completion handler

Access the properties only if loaded

Be prepared for synchronous callback

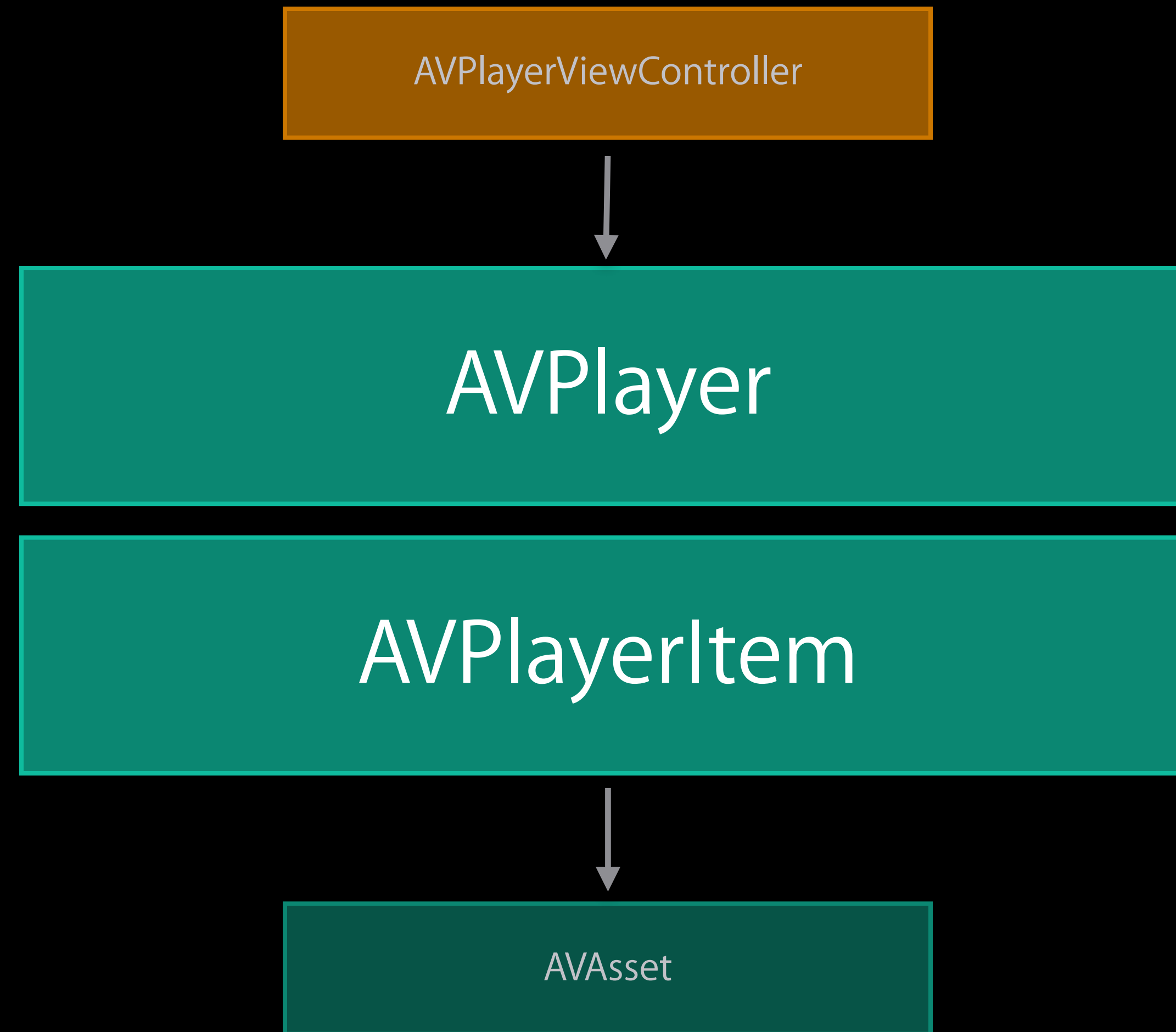
Best Practices

Roadmap



Best Practices

Roadmap



AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range

AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range



AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range



AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range

AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range
- Example—Playback interruption by a phone call

AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range
- Example—Playback interruption by a phone call
- Example—Media services reset

AVPlayer and AVPlayerItem

Observation via NSKeyValueObserving

Properties can change with playback

- Example—Progressive download item's loaded range
- Example—Playback interruption by a phone call
- Example—Media services reset

Update state during playback using KVO

AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
NSArray *videoTracks = [asset tracksWithMediaType:AVMediaTypeVideo];
```

```
if (videoTracks.count > 0) {  
    videoTrack = videoTracks.firstObject;  
    // Update UI to reflect audio-video.  
} else {  
    // Update UI to reflect audio-only.  
}
```

AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
NSArray *videoTracks = [asset tracksWithMediaType:AVMediaTypeVideo];
```

```
if (videoTracks.count > 0) {  
    videoTrack = videoTracks.firstObject;  
    // Update UI to reflect audio-video.  
} else {  
    // Update UI to reflect audio-only.  
}
```

AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
NSArray *videoTracks = [asset tracksWithMediaType:AVMediaTypeVideo];
```

```
if (videoTracks.count > 0) {  
    videoTrack = videoTracks.firstObject;  
    // Update UI to reflect audio-video.  
} else {  
    // Update UI to reflect audio-only.  
}
```


AVPlayer and AVPlayerItem

Deciding when to show audio-only UI



```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
NSArray *videoTracks = [asset tracksWithMediaType:AVMediaTypeVideo];
```

```
if (videoTracks.count > 0) {  
    videoTrack = videoTracks.firstObject;  
    // Update UI to reflect audio-video.  
} else {  
    // Update UI to reflect audio-only.  
}
```

AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
[playerItem addObserver:self  
                forKeyPath:@"presentationSize"  
                options:NSKeyValueObservingOptionNew  
                context:presentationSizeObservationContext];
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
// Inside -observeValueForKeyPath:ofObject:change:context: implementation...
if (presentationSizeObservationContext == context) {
    // Check if new presentation size is CGSizeZero.
    CGSize size = change[NSKeyValueChangeNewKey].sizeValue;
    if (CGSizeEqualToSize(size, CGSizeZero)) {
        for (AVPlayerItemTrack *playerItemTrack in playerItem.tracks) {
            AVAssetTrack *track = playerItemTrack.assetTrack;
            if ([track hasMediaCharacteristic:AVMediaCharacteristicAudible]) {
                // Show audio-only UI.
            }
        }
    }
}
}
```

AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
// Inside -observeValueForKeyPath:ofObject:change:context: implementation...
if (presentationSizeObservationContext == context) {
    // Check if new presentation size is CGSizeZero.
    CGSize size = change[NSKeyValueChangeNewKey].sizeValue;
    if (CGSizeEqualToSize(size, CGSizeZero)) {
        for (AVPlayerItemTrack *playerItemTrack in playerItem.tracks) {
            AVAssetTrack *track = playerItemTrack.assetTrack;
            if ([track hasMediaCharacteristic:AVMediaCharacteristicAudible]) {
                // Show audio-only UI.
            }
        }
    }
}
}
```


AVPlayer and AVPlayerItem

Deciding when to show audio-only UI

```
// Inside -observeValueForKeyPath:ofObject:change:context: implementation...
if (presentationSizeObservationContext == context) {
    // Check if new presentation size is CGSizeZero.
    CGSize size = change[NSKeyValueChangeNewKey].sizeValue;
    if (CGSizeEqualToSize(size, CGSizeZero)) {
        for (AVPlayerItemTrack *playerItemTrack in playerItem.tracks) {
            AVAssetTrack *track = playerItemTrack.assetTrack;
            if ([track hasMediaCharacteristic:AVMediaCharacteristicAudible]) {
                // Show audio-only UI.
            }
        }
    }
}
}
```

AVPlayer and AVPlayerItem

KVO recipe



1. Create a player item with asset
2. Register for key value observing a property
3. Create a player with the item
4. Check for new value in the observation callback

AVPlayer and AVPlayerItem

Do not assume the order of events

Client

```
playerWithPlayerItem:
```

```
addObserver:self
```

```
forKeyPath:@"status"
```

```
options:NSKeyValueObservingOptionNew
```

```
...
```

AVFoundation



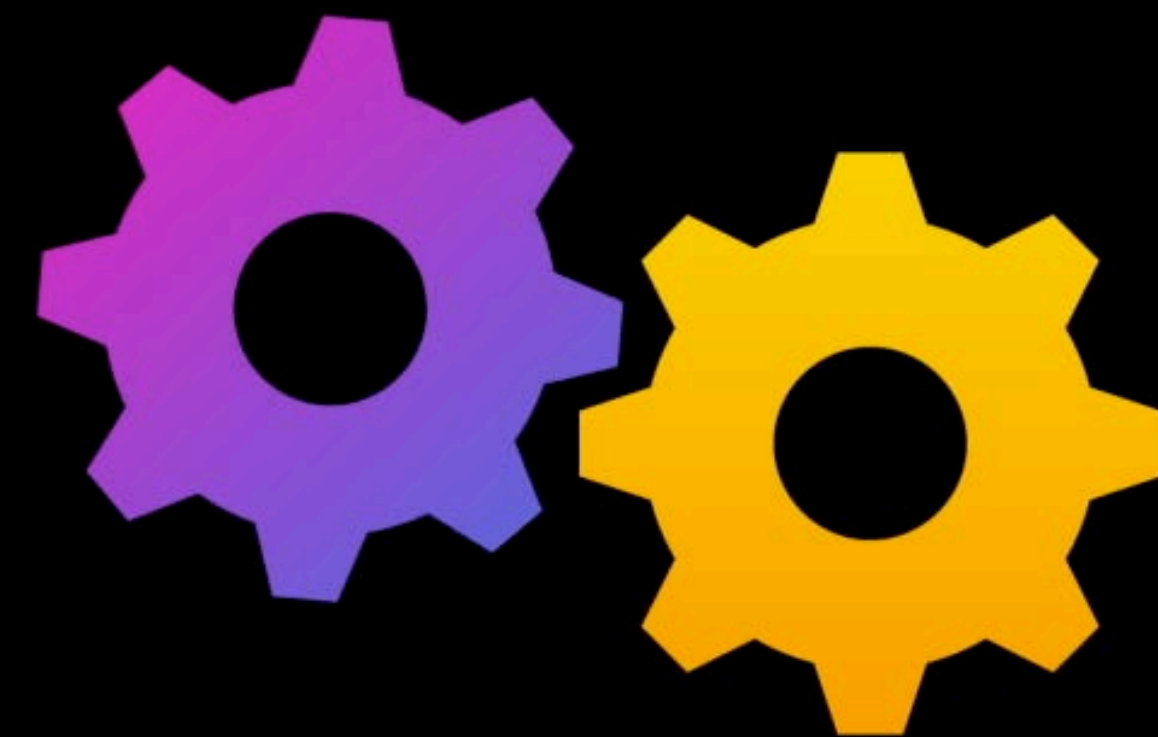
AVPlayer and AVPlayerItem

Do not assume the order of events

Client

AVFoundation

playerWithPlayerItem:



Status Changed

addObserver:self

forKeyPath:@"status"

options:NSKeyValueObservingOptionNew

...

AVPlayer and AVPlayerItem

Do not assume the order of events

Client

```
addObserver:self  
forKeyPath:@"status"  
options:NSKeyValueObservingOptionNew  
...
```

```
playerWithPlayerItem:
```

AVFoundation



AVPlayer and AVPlayerItem

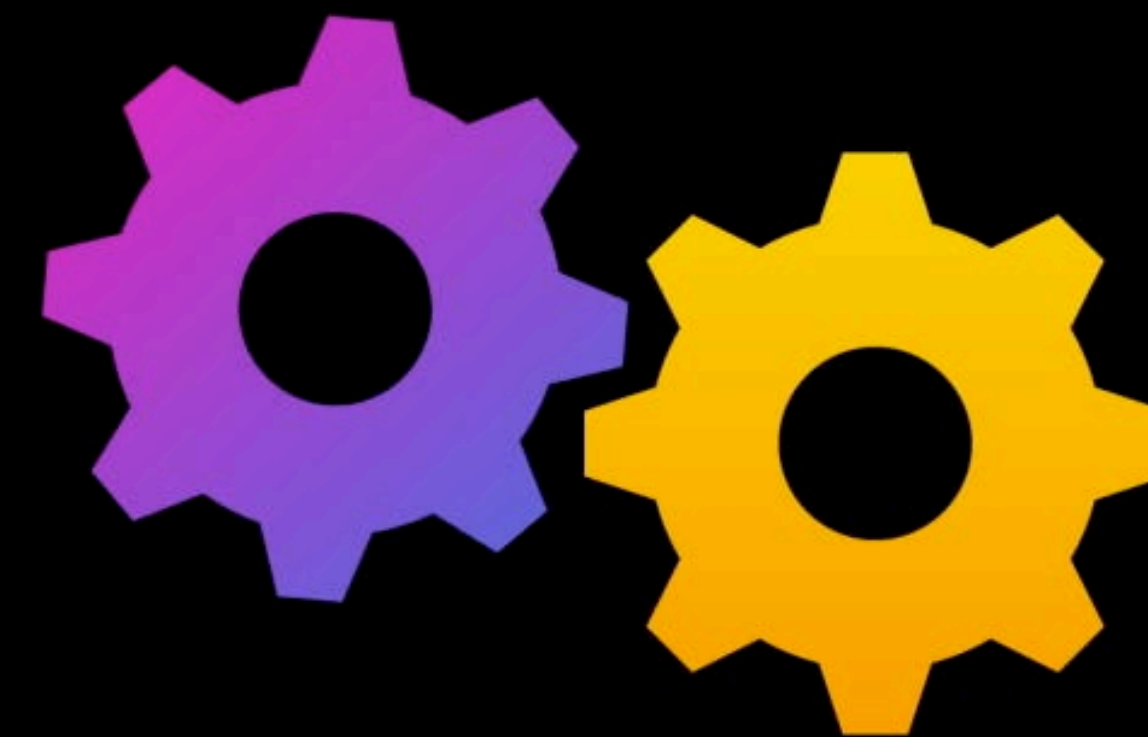
Do not assume the order of events

Client

AVFoundation

```
addObserver:self  
forKeyPath:@"status"  
options:NSKeyValueObservingOptionNew  
...
```

playerWithPlayerItem: 



Status Changed

AVPlayer and AVPlayerItem

Do not assume the order of events

Alternatively, ask for initial value in the options for KVO

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:item];
```

```
[item addObserver:self  
         forKeyPath:@"status"  
         options:NSKeyValueObservingOptionNew | NSKeyValueObservingOptionInitial  
         context:statusObservationContext];
```

AVPlayer and AVPlayerItem

Safely accessing properties

Serialize access on main queue

Register and unregister observers on main queue

Avoids possible race conditions when accessing properties

Does not affect the end-user responsiveness

AVPlayer and AVPlayerItem

Set up player item before adding it to the player

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
// Add outputs.
```

```
[output setDelegate:self queue:myDispatchQueue];
```

```
[playerItem addOutput:output];
```

```
// Select media options.
```

```
[playerItem selectMediaOption:option inMediaSelectionGroup:group];
```

```
// Set end time.
```

```
playerItem.forwardPlaybackEndTime = endTime;
```

```
// Once configured, add to player.
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

AVPlayer and AVPlayerItem

Set up player item before adding it to the player

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
// Add outputs.
```

```
[output setDelegate:self queue:myDispatchQueue];
```

```
[playerItem addOutput:output];
```

```
// Select media options.
```

```
[playerItem selectMediaOption:option inMediaSelectionGroup:group];
```

```
// Set end time.
```

```
playerItem.forwardPlaybackEndTime = endTime;
```

```
// Once configured, add to player.
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

AVPlayer and AVPlayerItem

Set up player item before adding it to the player

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
// Add outputs.
```

```
[output setDelegate:self queue:myDispatchQueue];
```

```
[playerItem addOutput:output];
```

```
// Select media options.
```

```
[playerItem selectMediaOption:option inMediaSelectionGroup:group];
```

```
// Set end time.
```

```
playerItem.forwardPlaybackEndTime = endTime;
```

```
// Once configured, add to player.
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

AVPlayer and AVPlayerItem

Set up player item before adding it to the player

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
// Add outputs.
```

```
[output setDelegate:self queue:myDispatchQueue];
```

```
[playerItem addOutput:output];
```

```
// Select media options.
```

```
[playerItem selectMediaOption:option inMediaSelectionGroup:group];
```

```
// Set end time.
```

```
playerItem.forwardPlaybackEndTime = endTime;
```

```
// Once configured, add to player.
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```

AVPlayer and AVPlayerItem

Set up player item before adding it to the player

```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset];
```

```
// Add outputs.
```

```
[output setDelegate:self queue:myDispatchQueue];
```

```
[playerItem addOutput:output];
```

```
// Select media options.
```

```
[playerItem selectMediaOption:option inMediaSelectionGroup:group];
```

```
// Set end time.
```

```
playerItem.forwardPlaybackEndTime = endTime;
```

```
// Once configured, add to player.
```

```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
```


AVPlayer and AVPlayerItem

Best practices



Key value observe property changes

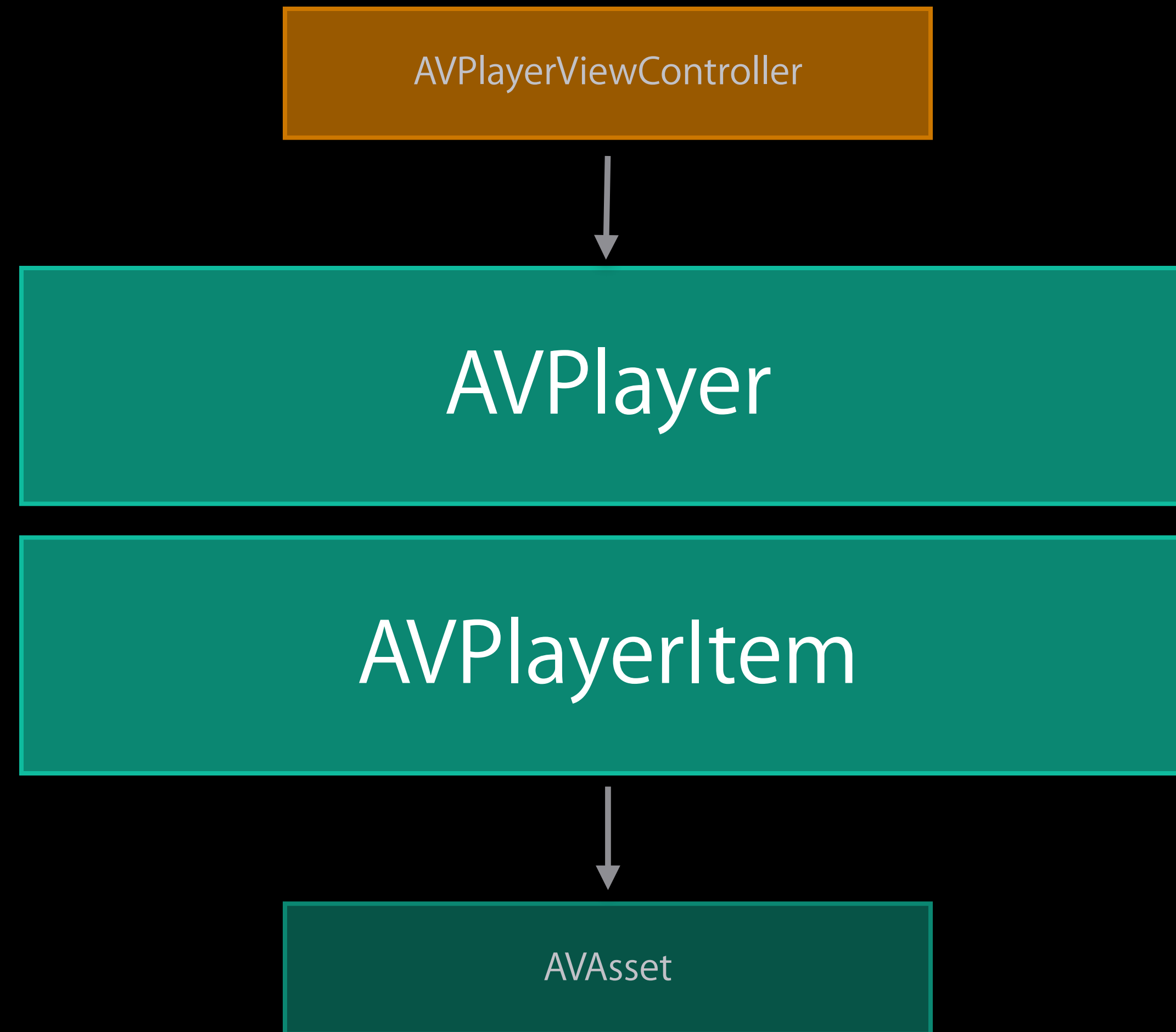
Do not rely on the ordering of events

Serialize access to objects on main queue

Set up player item before adding it to player

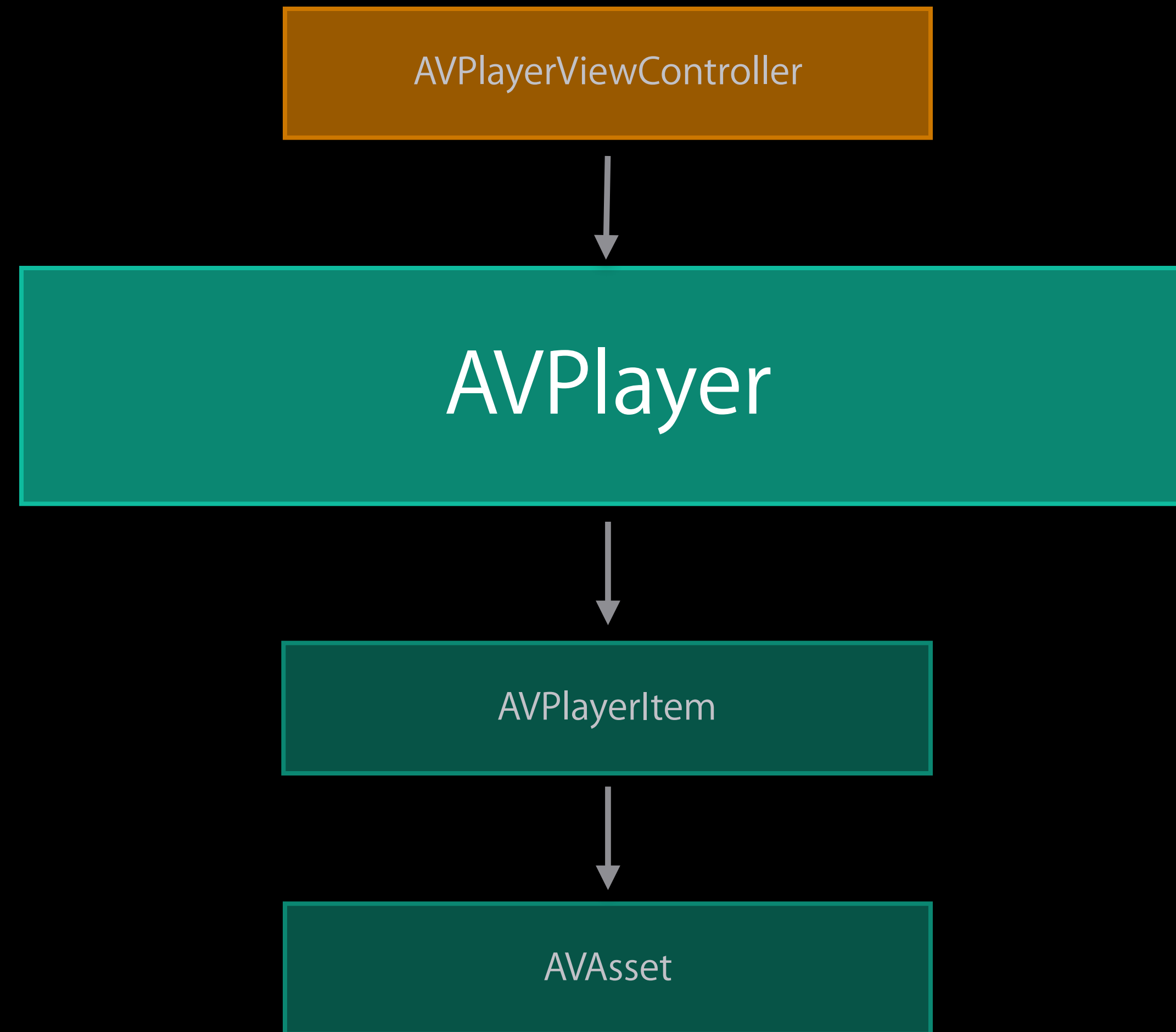
Best Practices

Roadmap



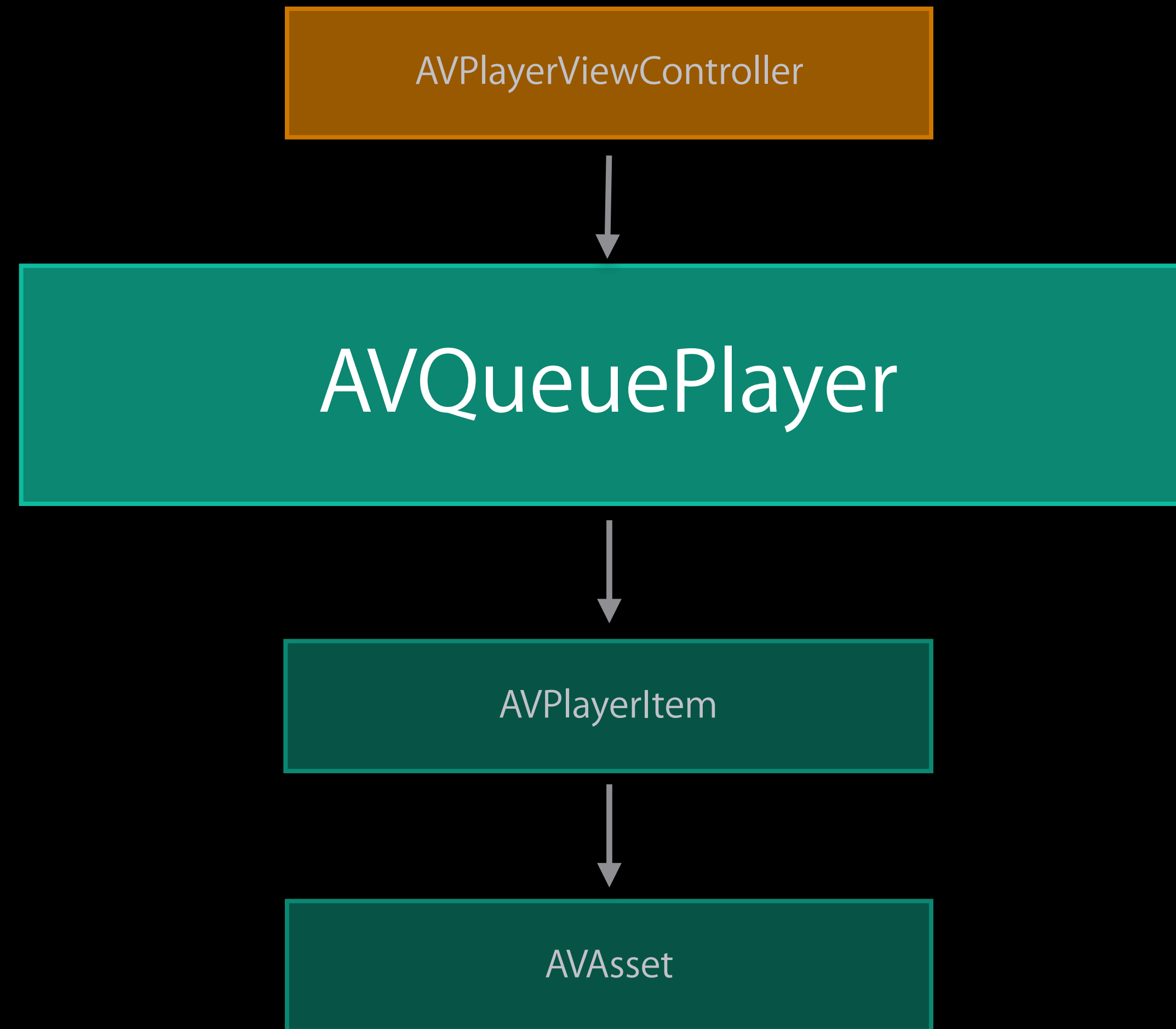
Best Practices

Roadmap



Best Practices

Roadmap



AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well

AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well



AVQueuePlayer

AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well

AVPlayerItem

AVPlayerItem

AVPlayerItem

AVPlayerItem

AVPlayerItem

AVQueuePlayer

AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well



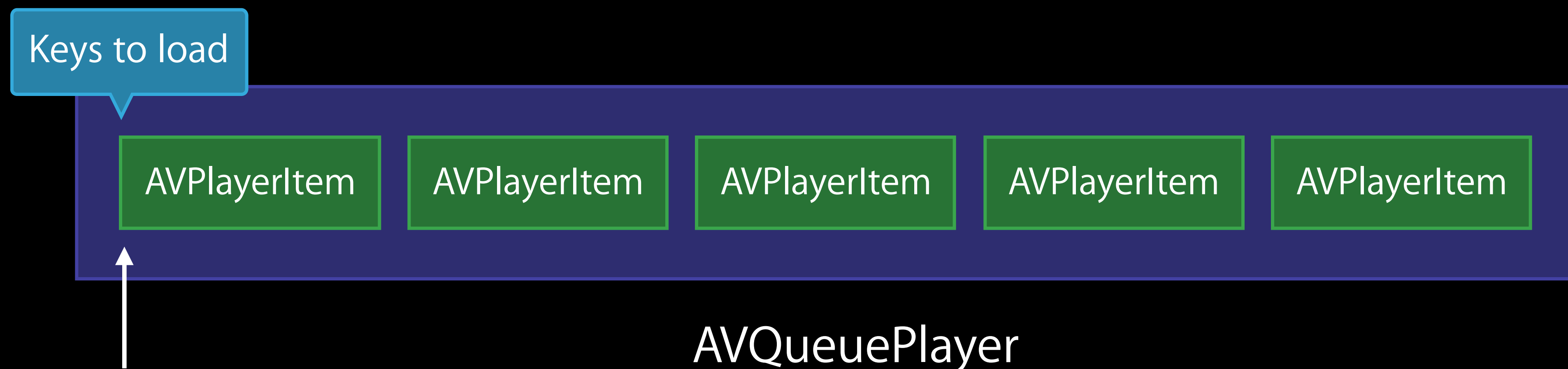
AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well



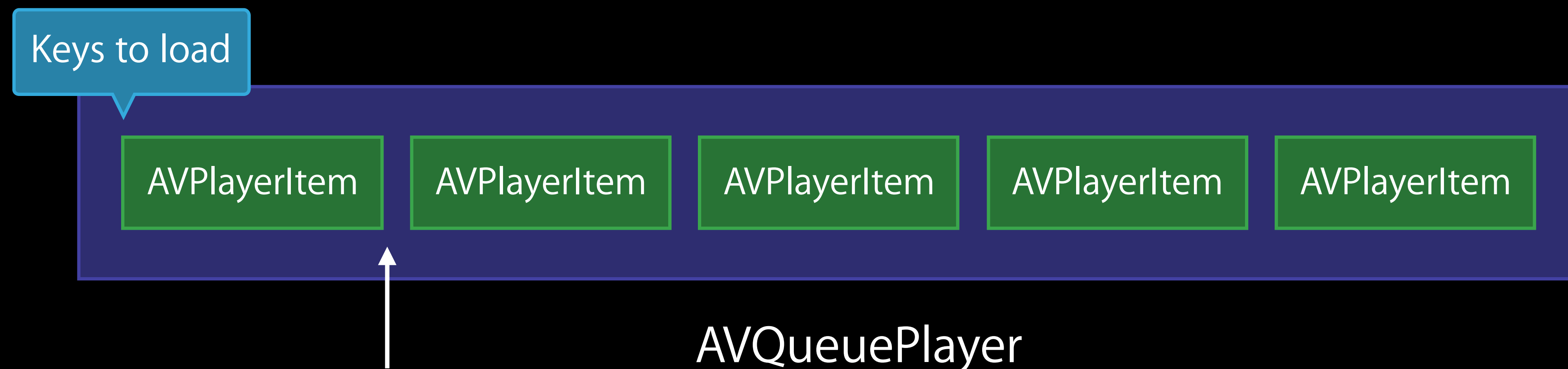
AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well



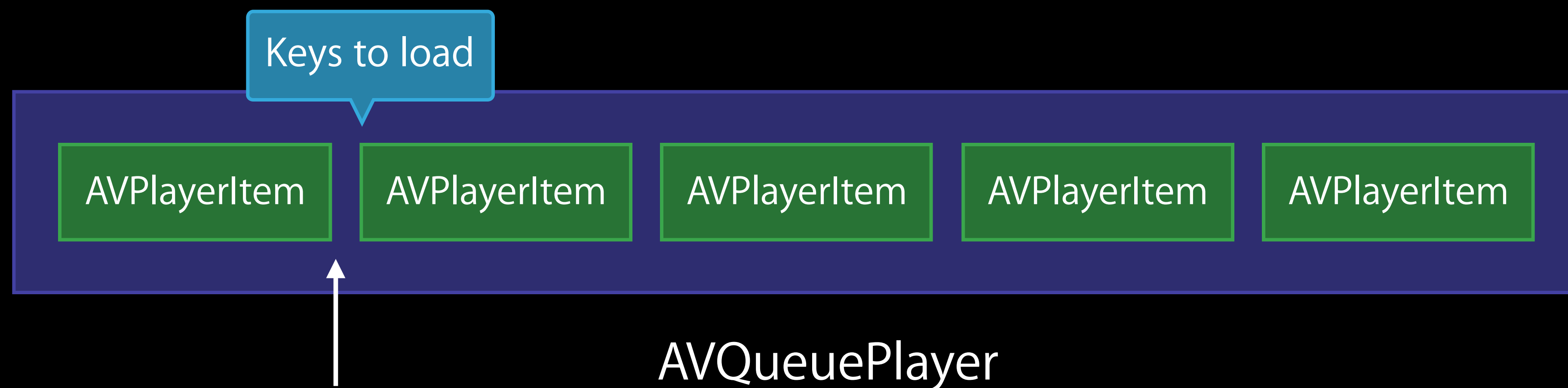
AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well



AVQueuePlayer

Best practices



`[AVPlayerItem initWithAsset:asset automaticallyLoadedAssetKeys:keys]`

- Automatically loads keys specified in combination with other keys loaded internally
- Valid for AVPlayer as well

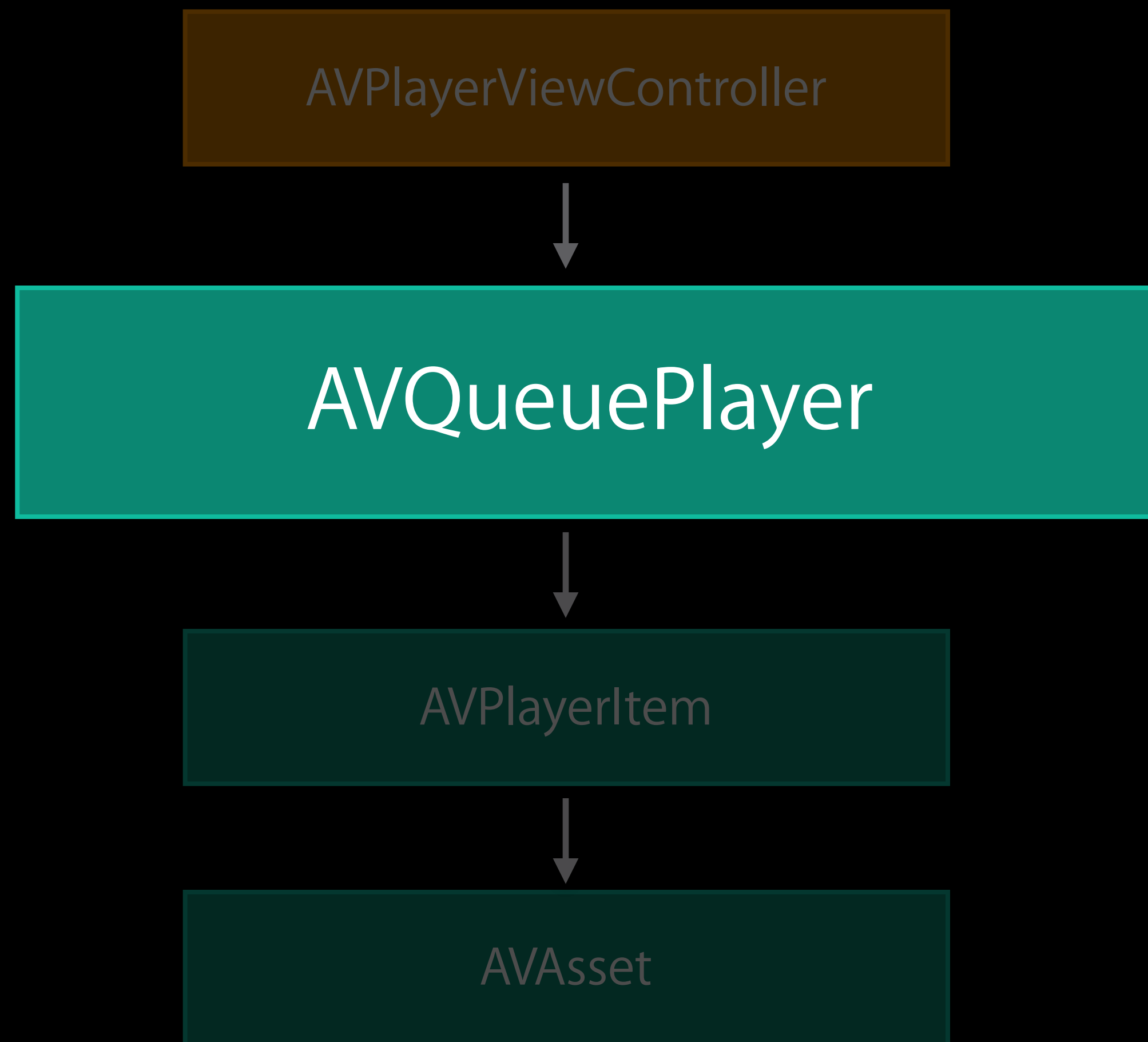
API contract

- When AVPlayerItemStatusReadyToPlay, asset keys will be loaded or have failed

Observe `AVQueuePlayer.currentItem.status` for error checking

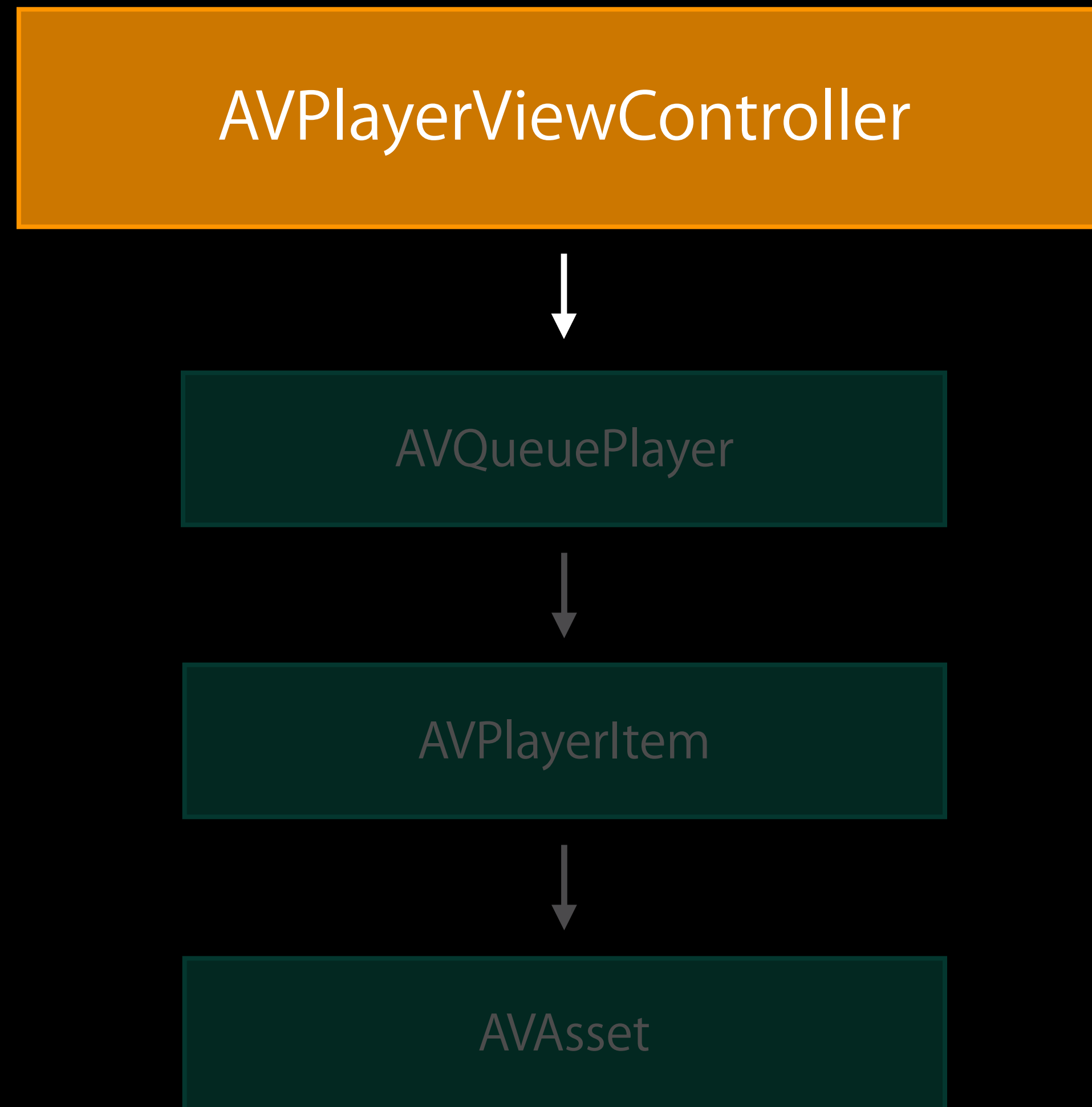
Best Practices

Roadmap



Best Practices

Roadmap



AVPlayerViewController

Show when ready for display

```
// Observe AVPlayerViewController.readyForDisplay.  
[playerViewController addObserver:self forKeyPath:@"readyForDisplay"  
options:0 context:readyForDisplayObservationContext];
```

```
// Inside observeValueForKeyPath:ofObject:change:context: implementation..  
if (readyForDisplayObservationContext == context) {  
    if (playerViewController.readyForDisplay) {  
        // Show AVPlayerViewController when first video frame is decoded.  
        playerViewController.view.hidden = NO;  
    }  
}
```


AVPlayerViewController

Show when ready for display

```
// Observe AVPlayerViewController.readyForDisplay.
[playerViewController addObserver:self forKeyPath:@"readyForDisplay"
options:0 context:readyForDisplayObservationContext];

// Inside observeValueForKeyPath:ofObject:change:context: implementation...
if (readyForDisplayObservationContext == context) {
    if (playerViewController.readyForDisplay) {
        // Show AVPlayerViewController when first video frame is decoded.
        playerViewController.view.hidden = NO;
    }
}
}
```

AVPlayerViewController

Content overlay view



```
// Content view frame matches player view bounds.  
[playerViewController.contentOverlayView addSubview:myCustomView];
```

```
// Use video rect to draw at right position.  
myCustomView.frame = playerViewController.videoBounds;
```

AVPlayerViewController

Content overlay view



```
// Content view frame matches player view bounds.  
[playerViewController.contentOverlayView addSubview:myCustomView];
```

```
// Use video rect to draw at right position.  
myCustomView.frame = playerViewController.videoBounds;
```

iPad 9:41 AM 100%

Done 0:29 -0:31

0:29 -0:31

Volume icon

Play/Pause, Stop, Next icons

iPad 9:41 AM 100%

Done 0:29 -0:31

The video player interface includes a progress bar at the top with a white dot indicating the current position. The video content shows two skateboarders on a paved road under a clear blue sky. The skateboarder in the foreground is wearing a red helmet and an orange patterned shirt. The skateboarder in the background is wearing a teal shirt. A yellow rectangular box highlights a thumbnail image in the bottom right corner of the video frame, which appears to be a smaller version of the video content. The bottom control bar contains a volume slider, a play/pause button, a skip back button, a skip forward button, and a full screen button.

AVPlayerView

Chapter navigation



```
- (void)seekToChapterAtTime:(CMTime)time
    chapterNumber:(NSUInteger)chapterNumber
    chapterTitle:(NSString *)chapterTitle {
    // Seek to chapter.
    [playerItem seekToTime:time completionHandler:^(BOOL finished) {
        if (finished) {
            // Flash chapter number and chapter title.
            [playerView flashChapterNumber:chapterNumber
                               chapterTitle:chapterTitle];
        }
    }];
}
```

AVPlayerView

Chapter navigation



```
- (void)seekToChapterAtTime:(CMTime)time
    chapterNumber:(NSUInteger)chapterNumber
    chapterTitle:(NSString *)chapterTitle {
    // Seek to chapter.
    [playerItem seekToTime:time completionHandler:^(BOOL finished) {
        if (finished) {
            // Flash chapter number and chapter title.
            [playerView flashChapterNumber:chapterNumber
                               chapterTitle:chapterTitle];
        }
    }];
}
```

AVPlayerViewController

Best practices



Show when ready for display

Use content overlay view to draw on top of video

Flash chapter number and title when seeking

Best Practices

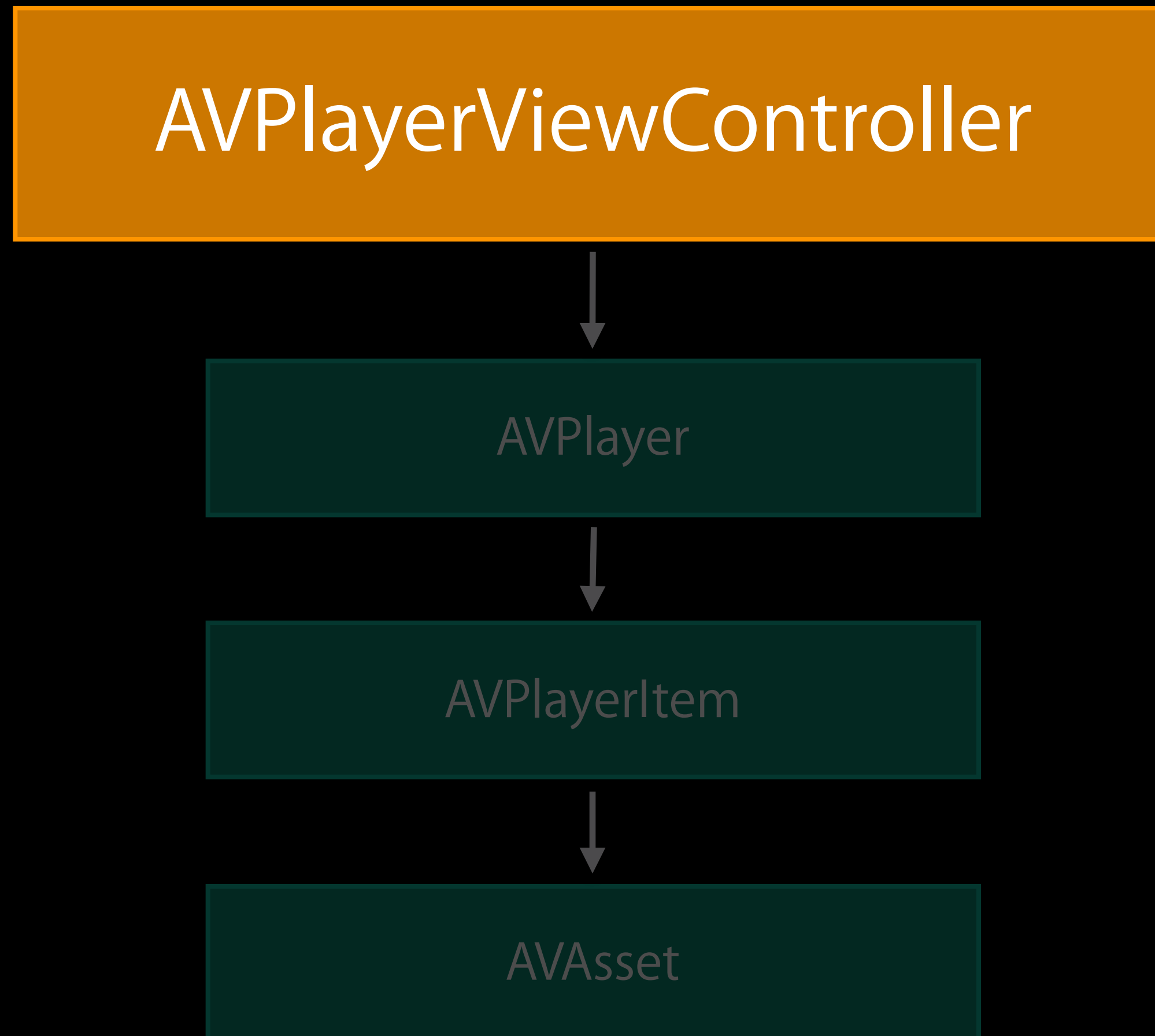
Roadmap

AVPlayerViewController

AVPlayer

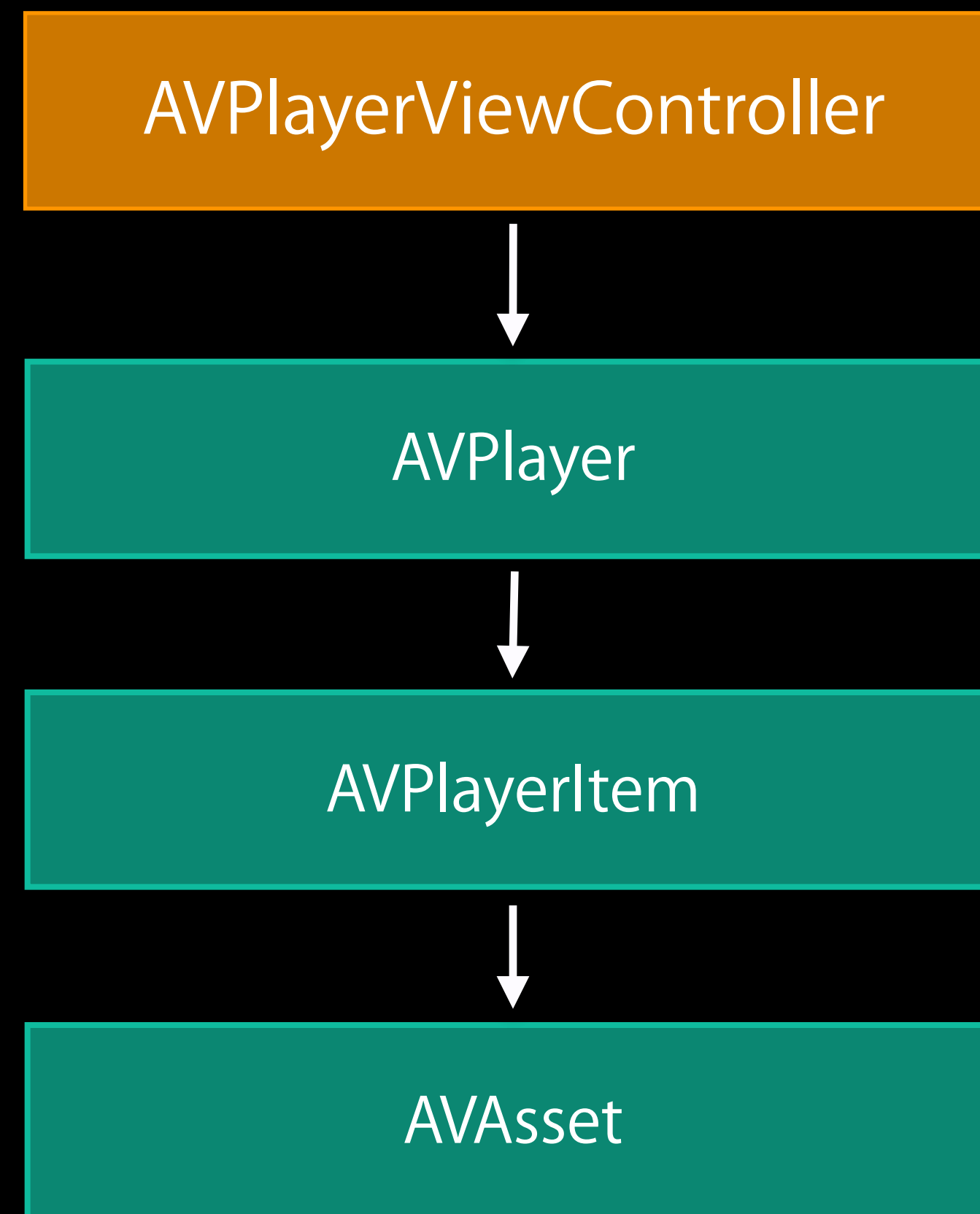
AVPlayerItem

AVAsset



Best Practices

Roadmap



AVFoundation and AVKit Best Practices

Wrap up

AVAsset inspection via `AVAsynchronousKeyValueLoading`

`AVPlayerItem`, `AVPlayer` observation via `NSKeyValueObserving`

Tips for using `AVQueuePlayer`

Observe `readyForDisplay` for display purposes

Customizing player view with content overlay view and chapter number display

Summary

AVKit available now for iOS and OS X

Automatic user interface refresh with *AVPlayerView*

AVFoundation has powerful API for effects and visualization

Adopt modern media frameworks

Optimize existing code by embracing best practices

More Information

Evangelism

evangelism@apple.com

Documentation

AVFoundation Programming Guide

<http://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/AVFoundationPG/>

Apple Developer Forums

<http://devforums.apple.com>

Related Sessions

-
- *Harnessing Metadata in Audiovisual Media* Pacific Heights Tuesday 2:00PM
 - *Camera Capture: Manual Controls* Marina Wednesday 11:30AM
 - *Direct Access to Video Encoding and Decoding* Nob Hill Thursday 11:30AM
-

Labs

-
- | | | |
|---------------------------------------|-------------|-------------------|
| ● AirPlay Lab | Media Lab B | Tuesday 2:00PM |
| ● AVFoundation Lab | Media Lab A | Tuesday 3:15PM |
| ● AVFoundation and Camera Capture Lab | Media Lab A | Wednesday 12:45PM |
| ● HTTP Live Streaming Lab | Media Lab A | Thursday 9:00AM |
| ● AVFoundation and Camera Capture Lab | Media Lab A | Thursday 2:00PM |
-

 WWDC14