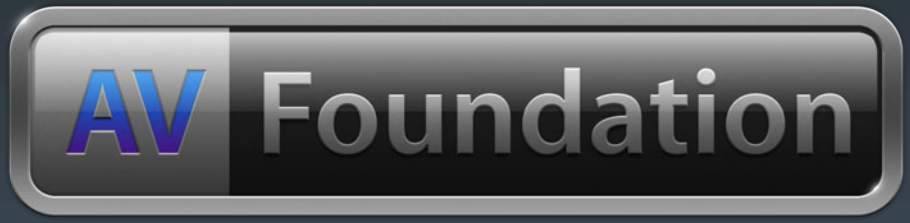
AV Foundation

Real-time media effects and processing during playback

Session 517

Simon Goldrei
Media Systems


AV Foundation

Agenda

Agenda

- Synchronizing an AVPlayer with a custom timeline

Agenda

- Synchronizing an AVPlayer with a custom timeline
- Real-time audio effects and processing

Agenda

- Synchronizing an AVPlayer with a custom timeline
- Real-time audio effects and processing
- Real-time video effects and processing

Agenda

- Synchronizing an AVPlayer with a custom timeline
- Real-time audio effects and processing
- Real-time video effects and processing

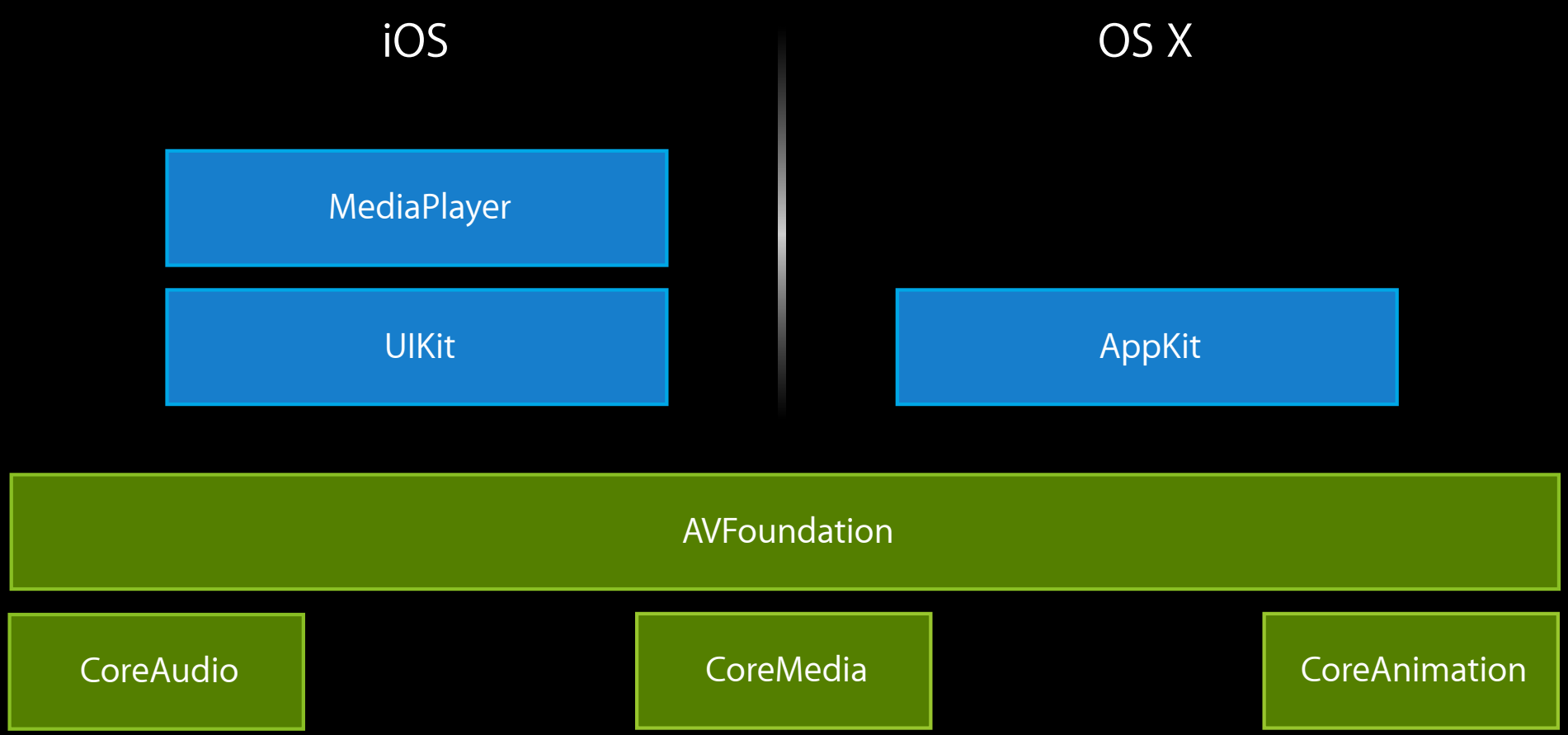
Stay in A/V Sync

Prerequisites

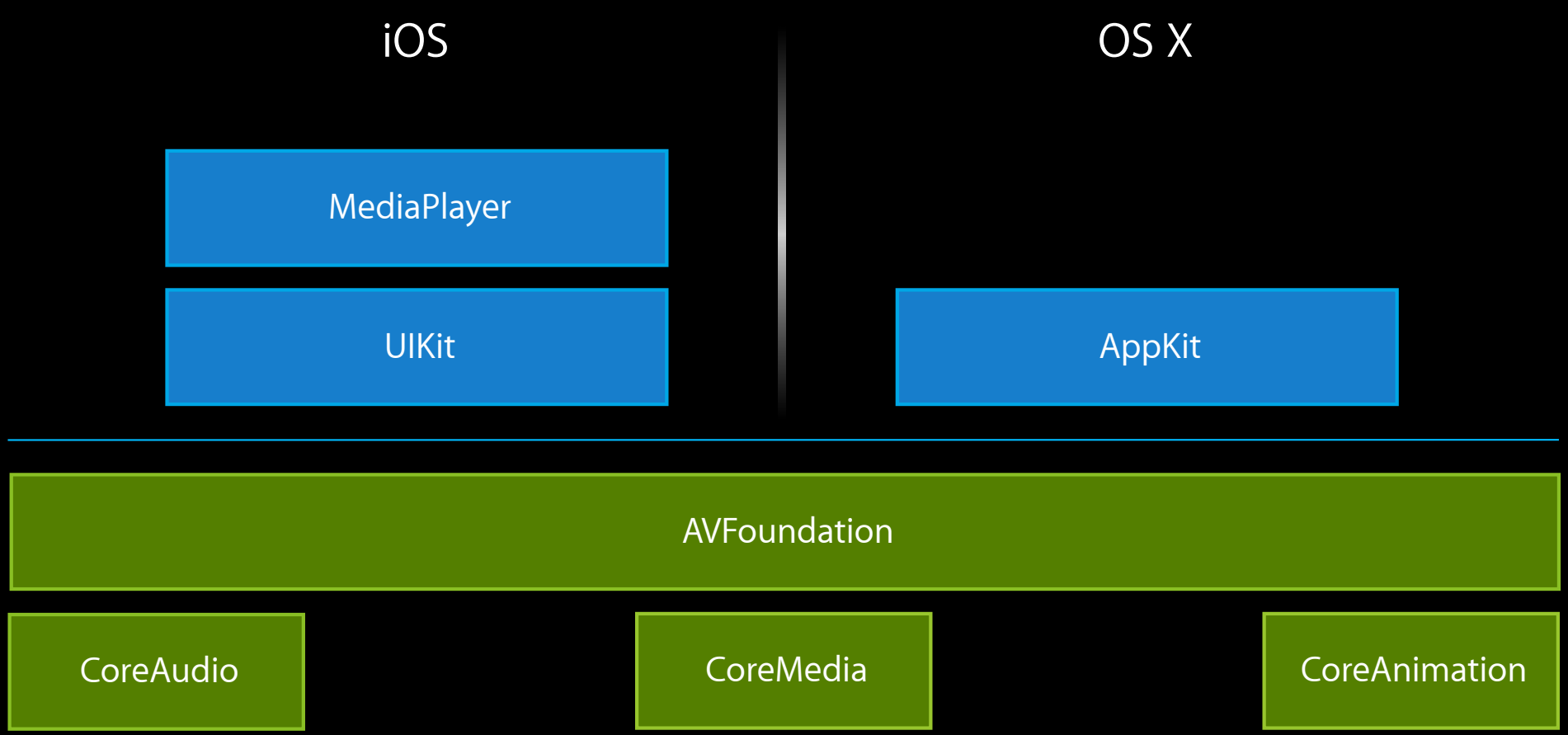
AVFoundation 2010 or 2011

- AVFoundation playback classes
  - AVAsset, AVAssetTrack
  - AVPlayerItem, AVPlayerItemTrack
  - AVPlayer

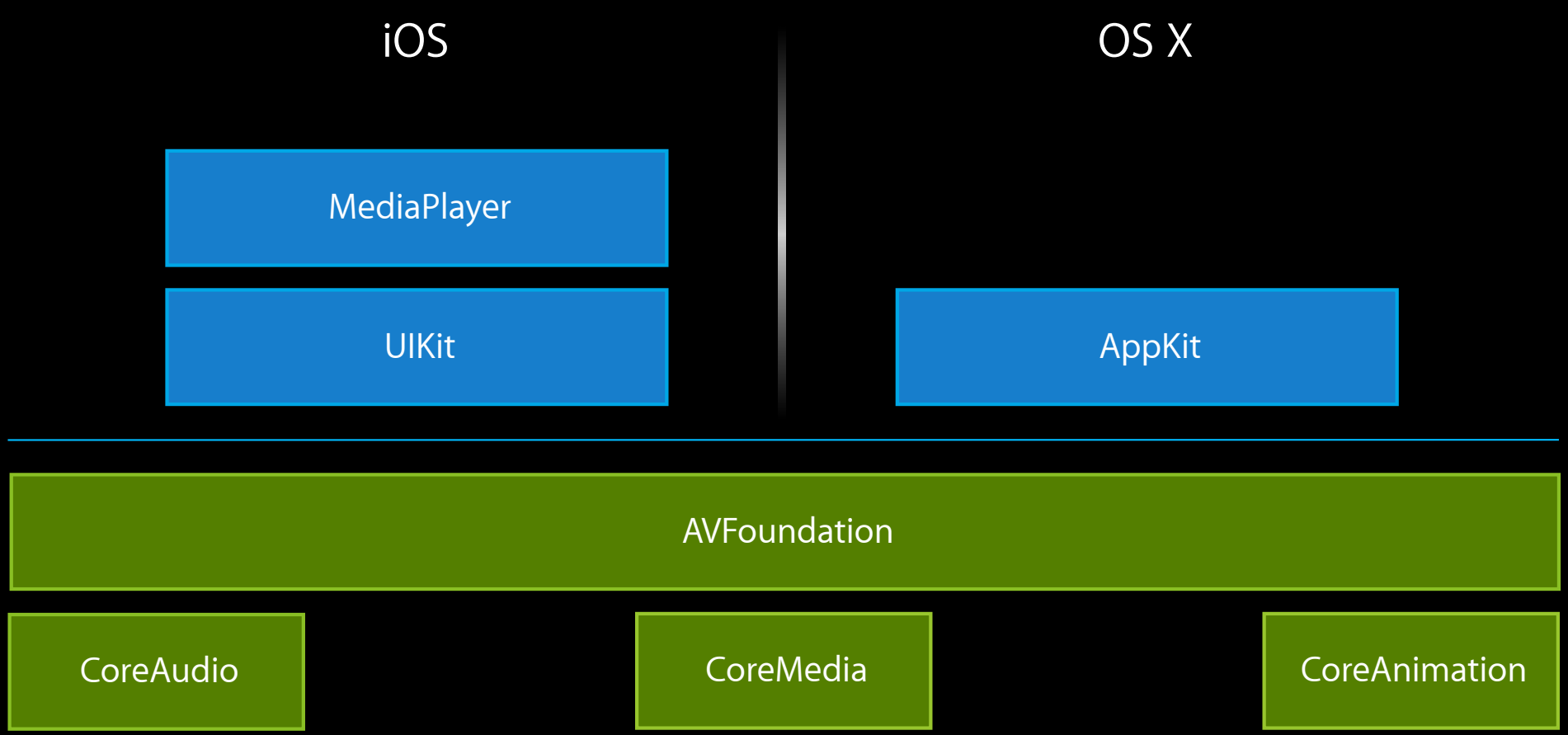
Where Is AV Foundation



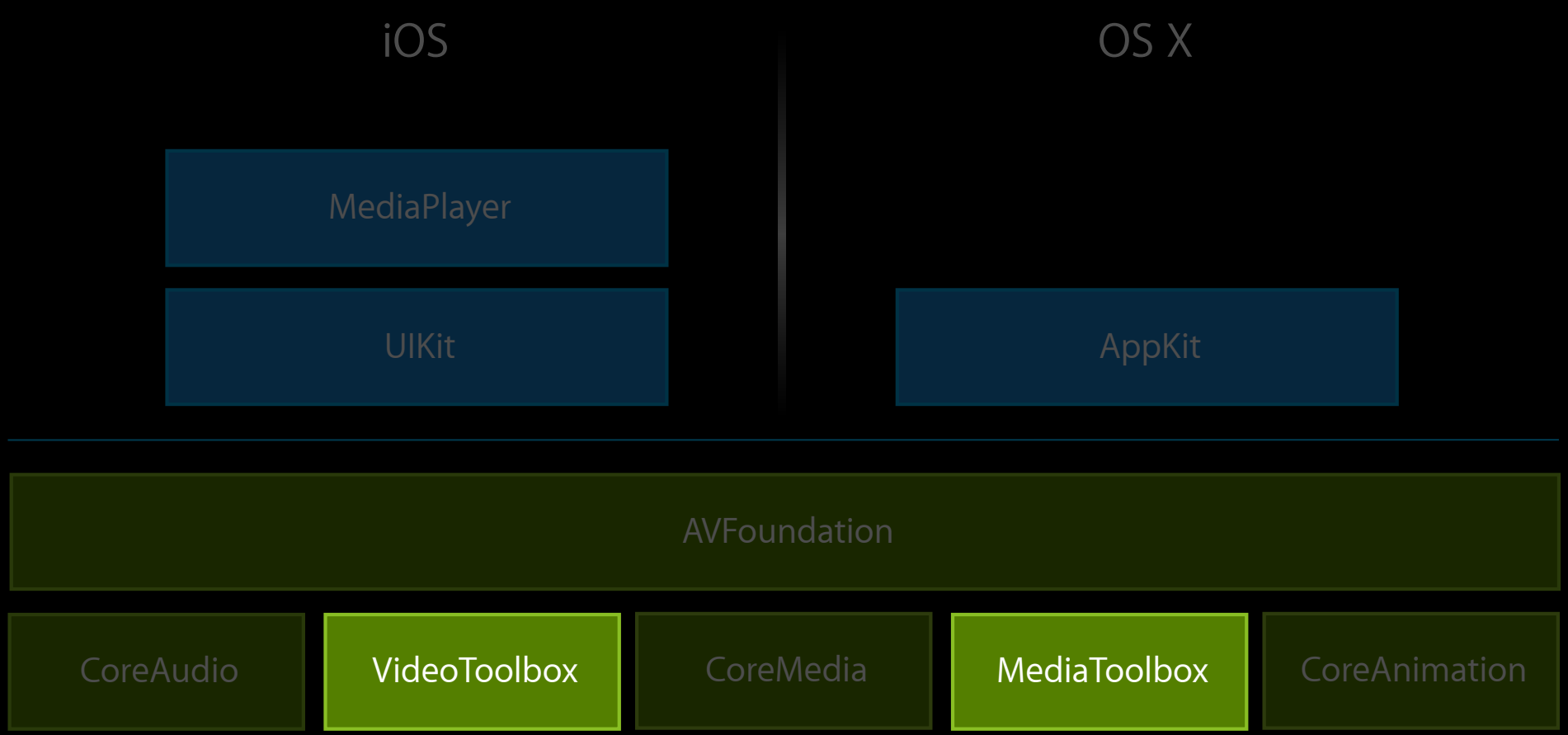
Where Is AV Foundation



Where Is AV Foundation

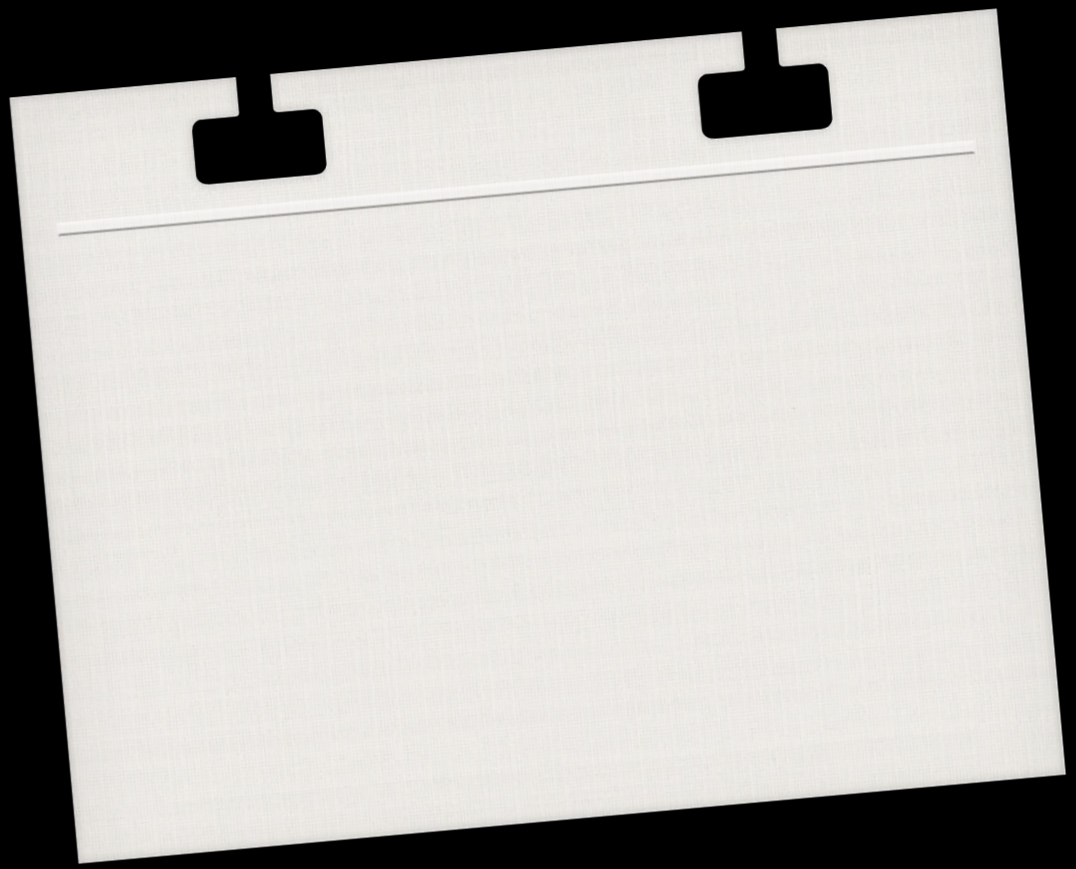


Where Is AV Foundation



Playback Update

Playback in Four Easy Steps



Playback in Four Easy Steps

Playback in Four Easy Steps

- Create your AVURLAsset
- Load @"tracks" asynchronously

Playback in Four Easy Steps

- Create your AVURLAsset
- Load @"tracks" asynchronously
- Create AVPlayerItem

Playback in Four Easy Steps

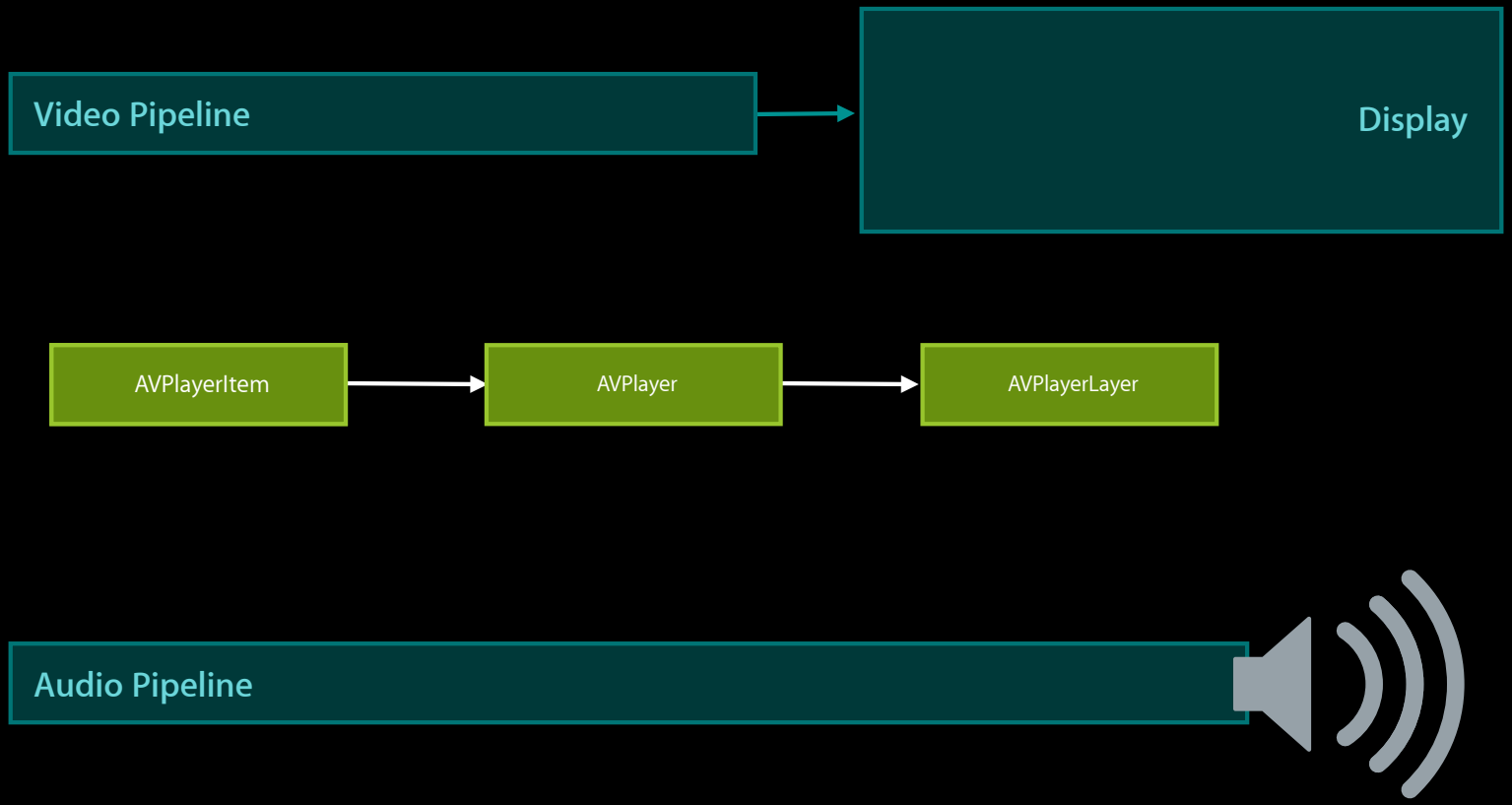
- Create your AVURLAsset
- Load @"tracks" asynchronously
- Create AVPlayerItem
- Create AVPlayer with your item

Playback in Three Easy Steps

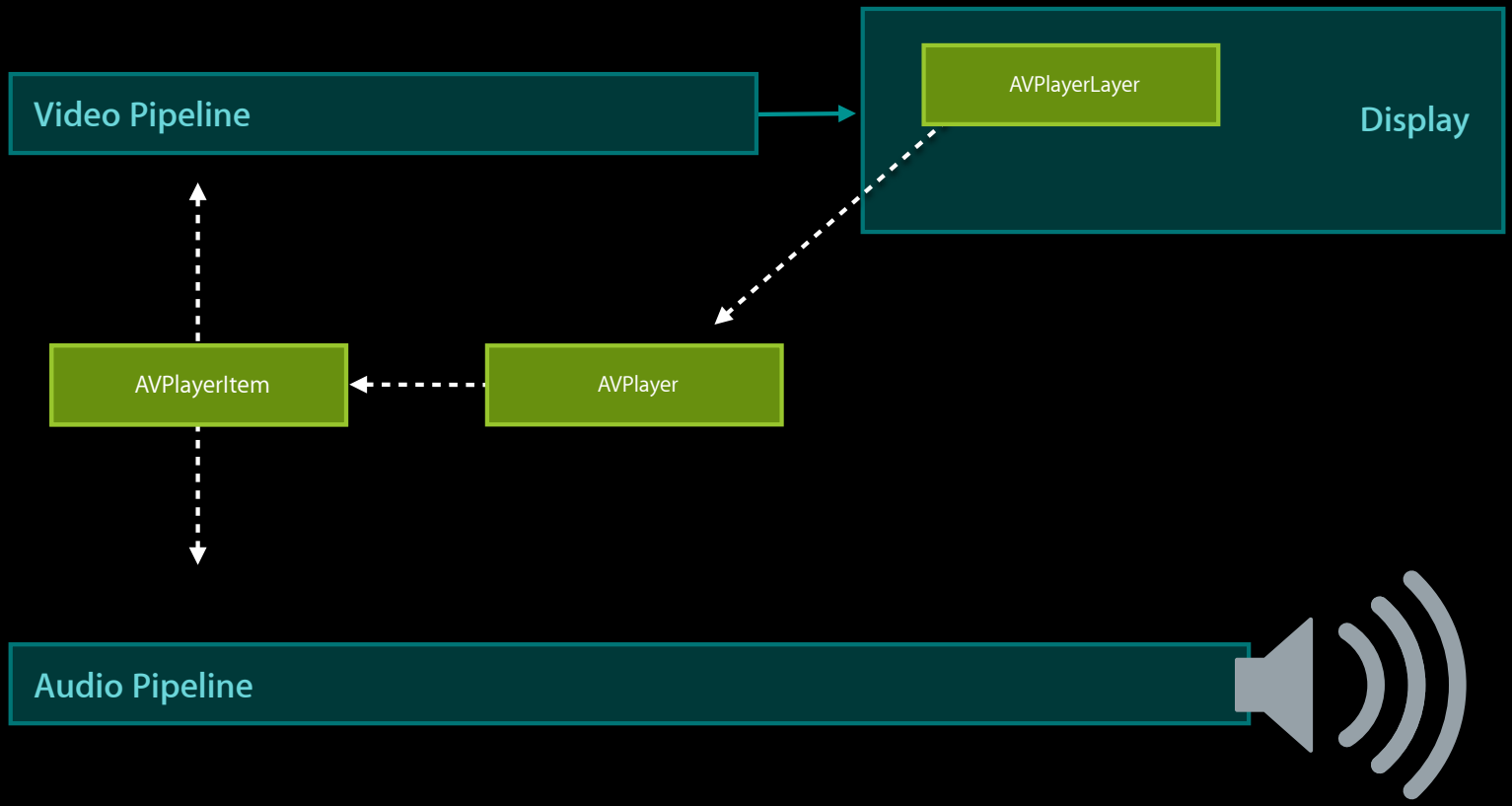
Customizing Playback

Synchronization primitives

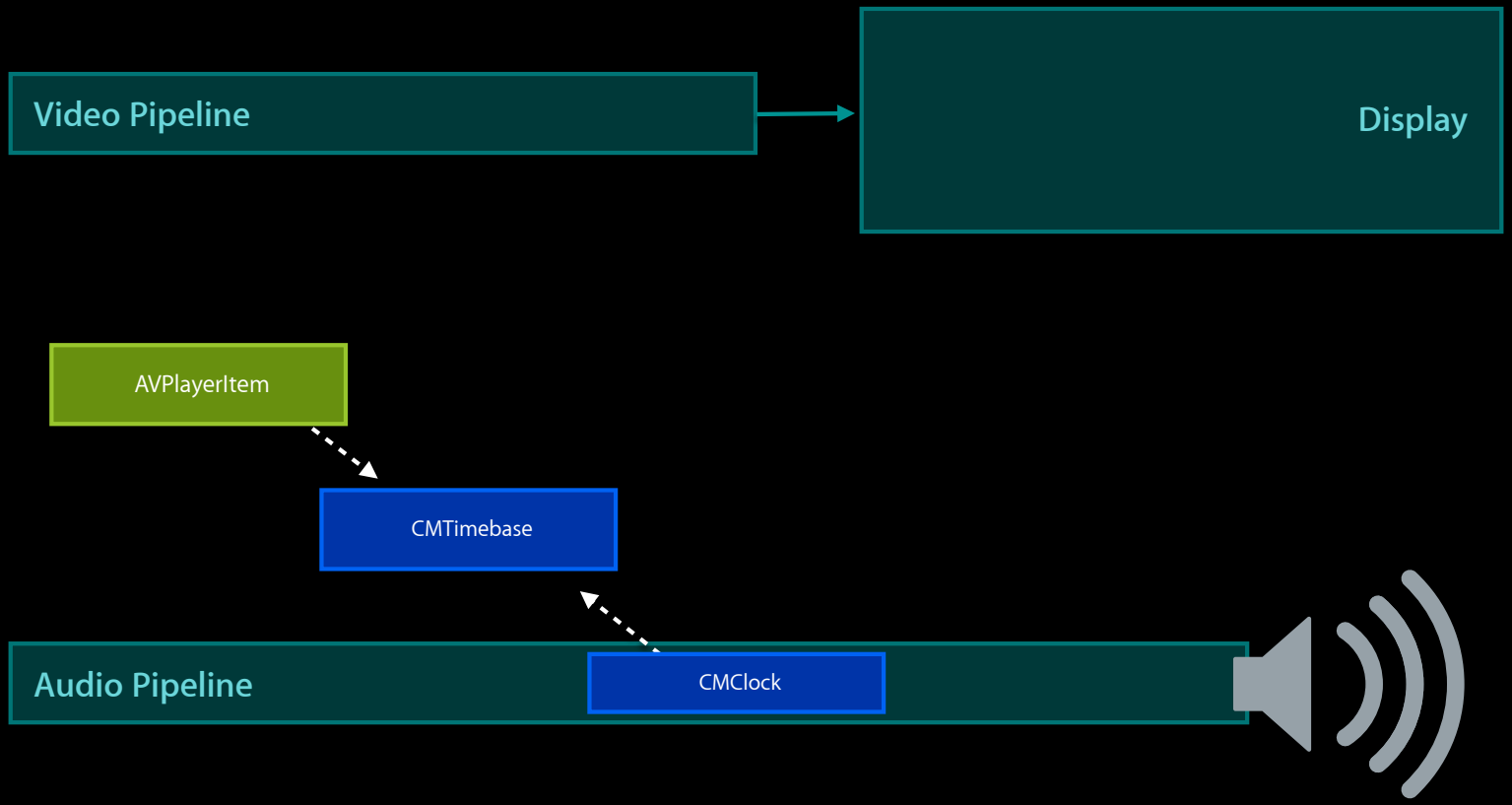
Playback Circa 2011



Playback Circa 2011



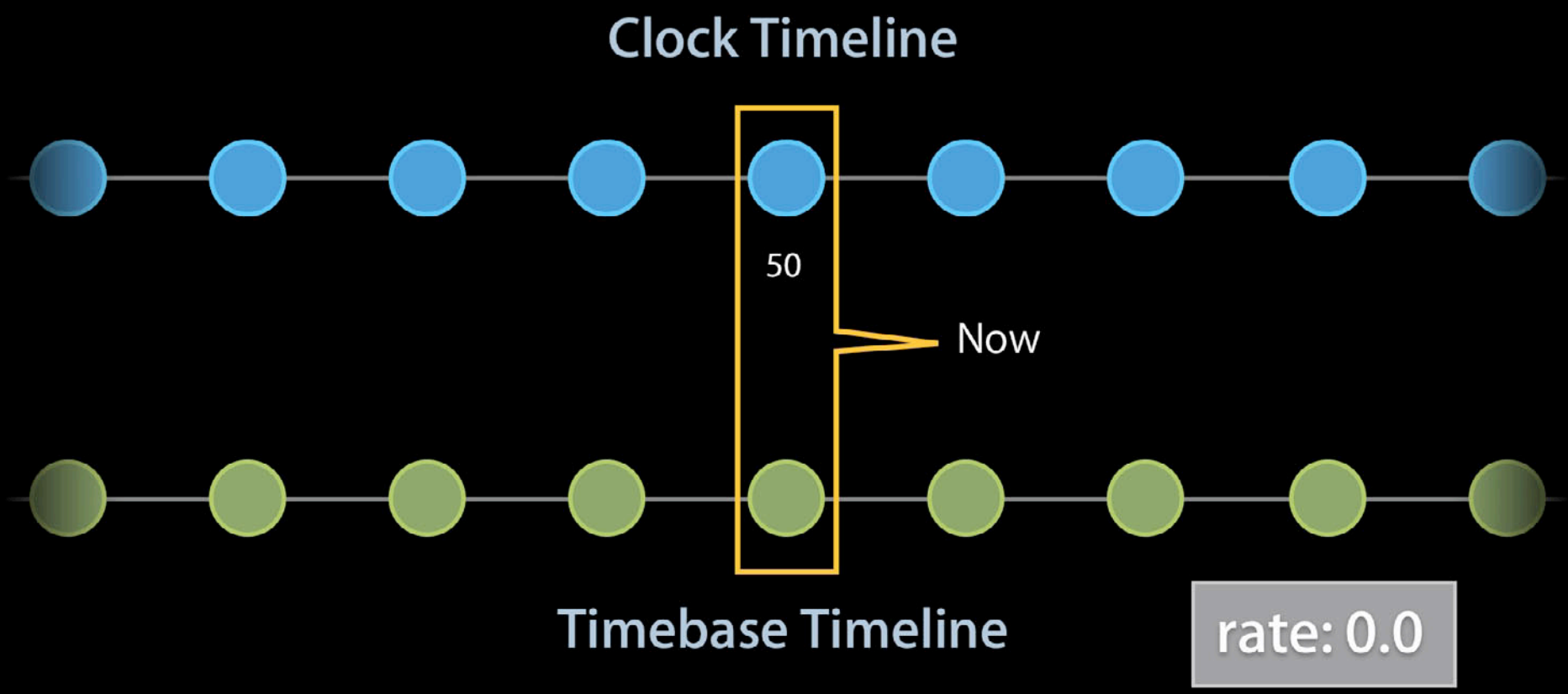
Keep It in Sync



CMClock and CMTimebase

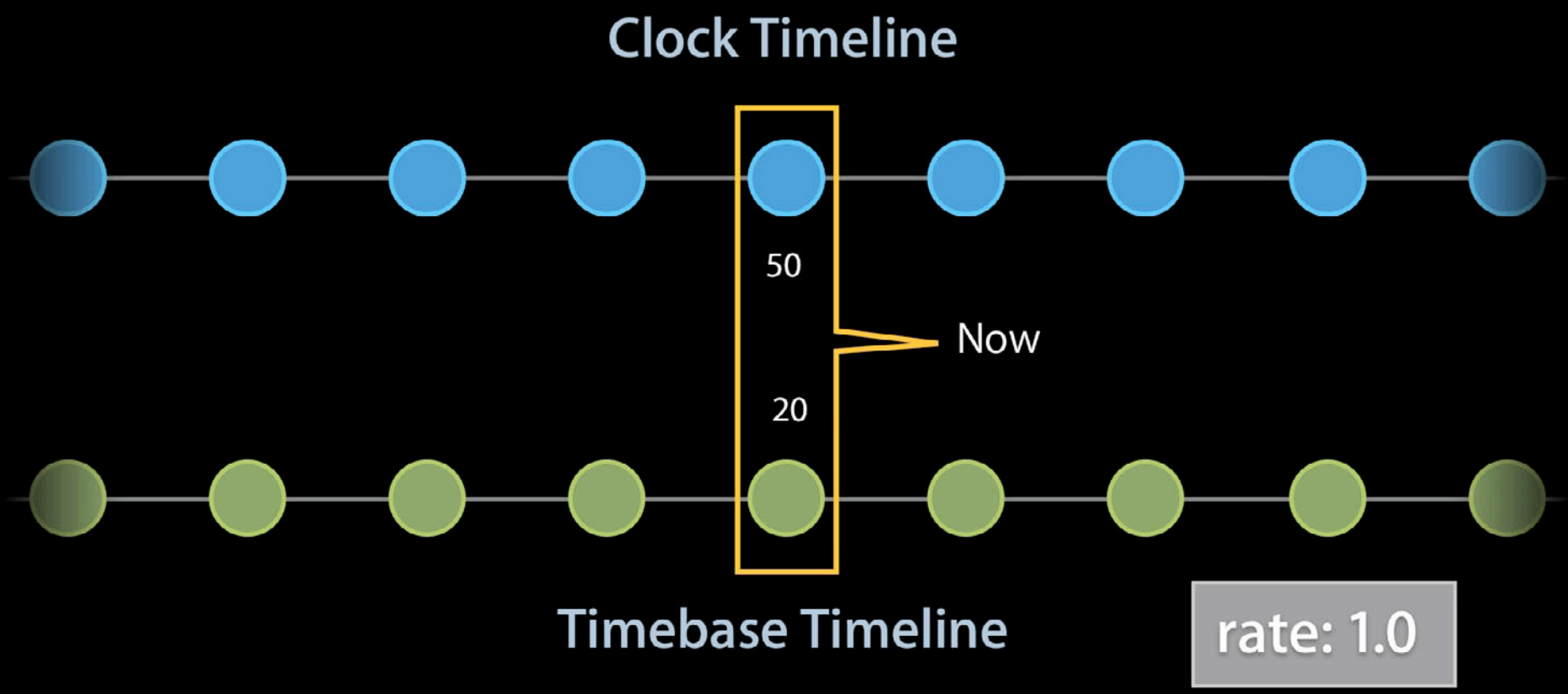
- CMClock
  - A source of time measurement
    - Host time clock
    - Audio device clock
- CMTimebase
  - Rate with respect to time
  - Controlled by your application
  - Slaved to a master

CMClock and CMTimebase



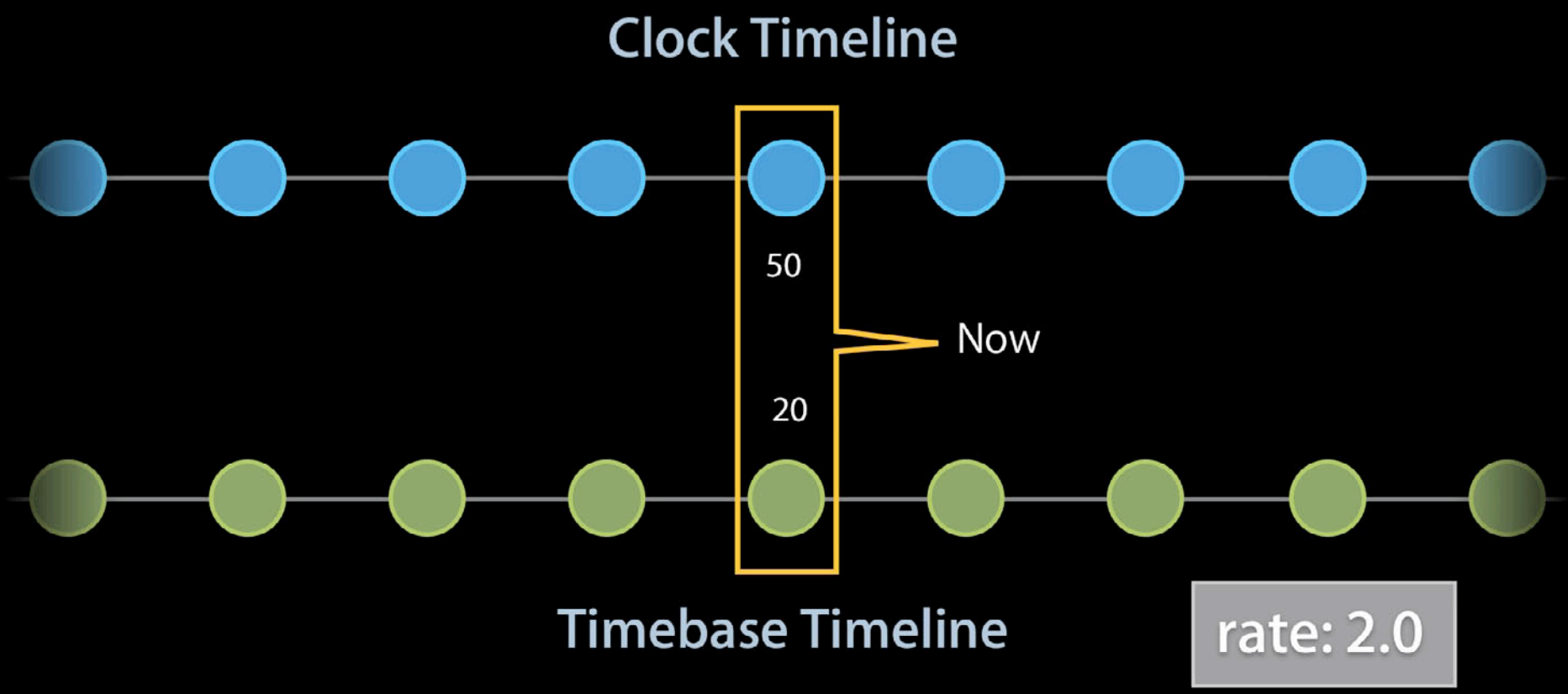
CMClock and CMTimebase

CMClock and CMTimebase



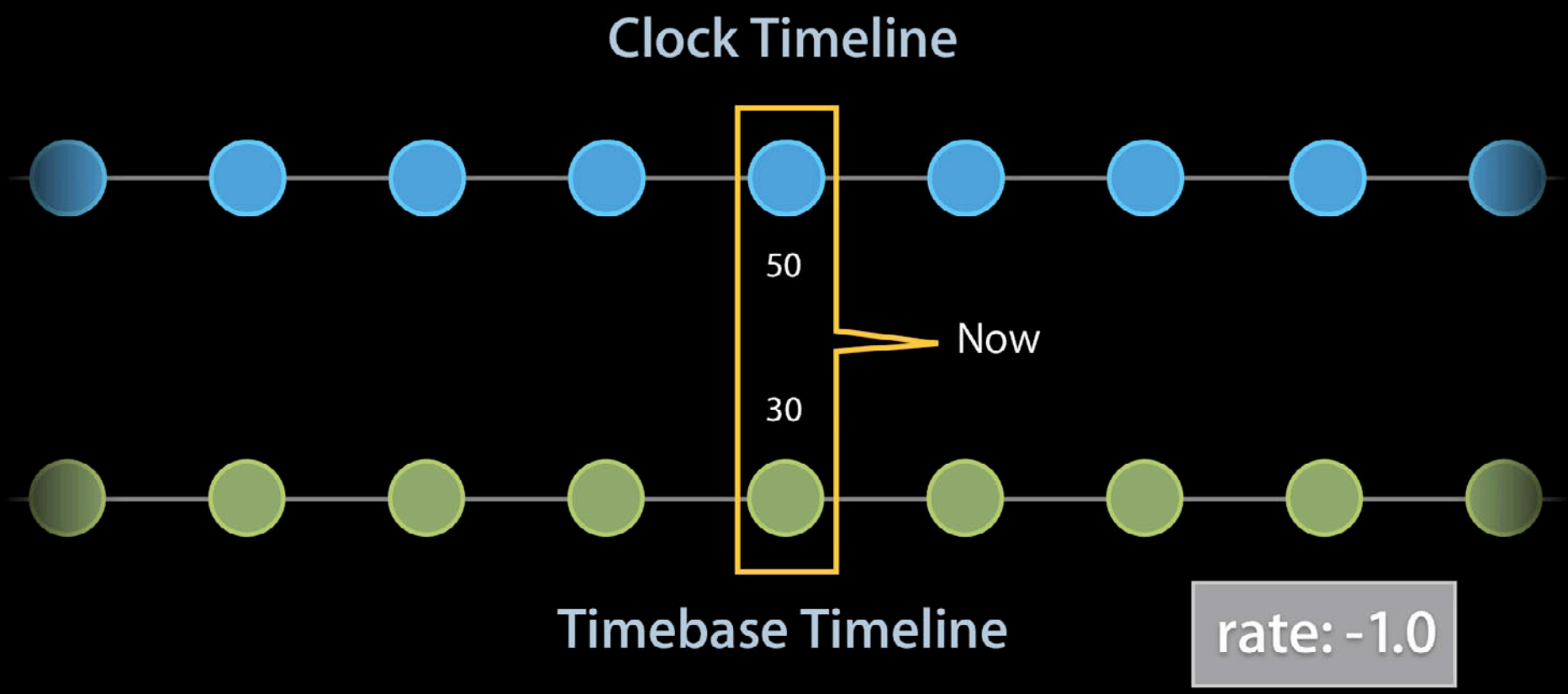
CMClock and CMTimebase

CMClock and CMTimebase



CMClock and CMTimebase

CMClock and CMTimebase



CMClock and CMTimebase

Audio Clock

CMAudioClock.h

```
OSStatus CMAudioClockCreate( CFAllocatorRef allocator,
                             CMClockRef *clockOut );
```

Audio Device Clock

CMAudioDeviceClock.h

```
OSStatus CMAudioDeviceClockCreate( CFAllocatorRef allocator,
                                   CFStringRef deviceUID,
                                   CMClockRef *clockOut );
```

Host Time Clock

CMSync.h

```
CMClockRef CMClockGetHostTimeClock( void );
```

CMTimebase

AVPlayerItem.h

```
@property (nonatomic, readonly) CMTimebaseRef timebase;
```

CMTimebase

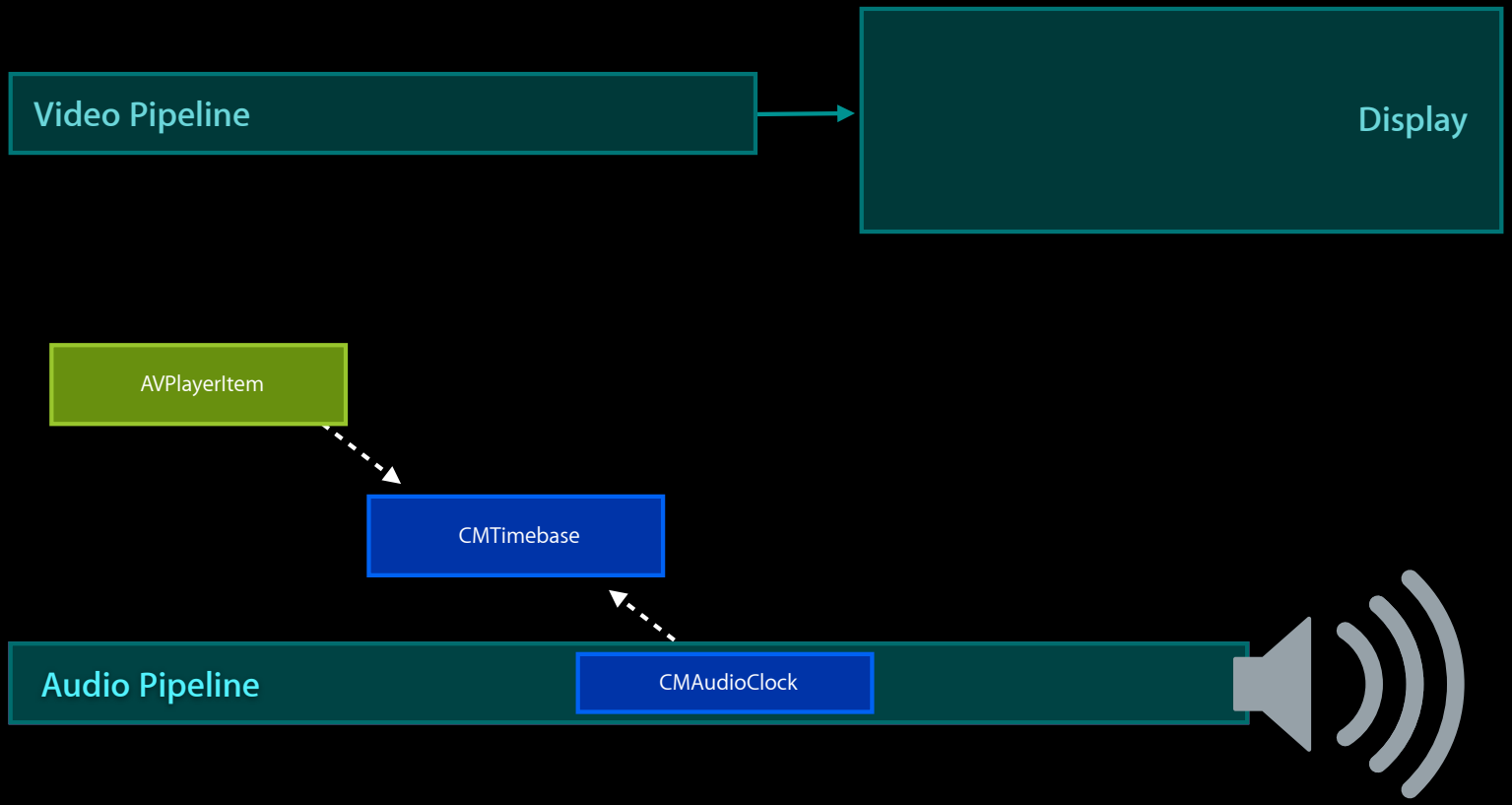
CMSync.h

```
OSStatus CMTimebaseCreateWithMasterClock( CFAllocatorRef allocator,
                                          CMClockRef masterClock,
                                          CMTimebaseRef *timebaseOut );
```

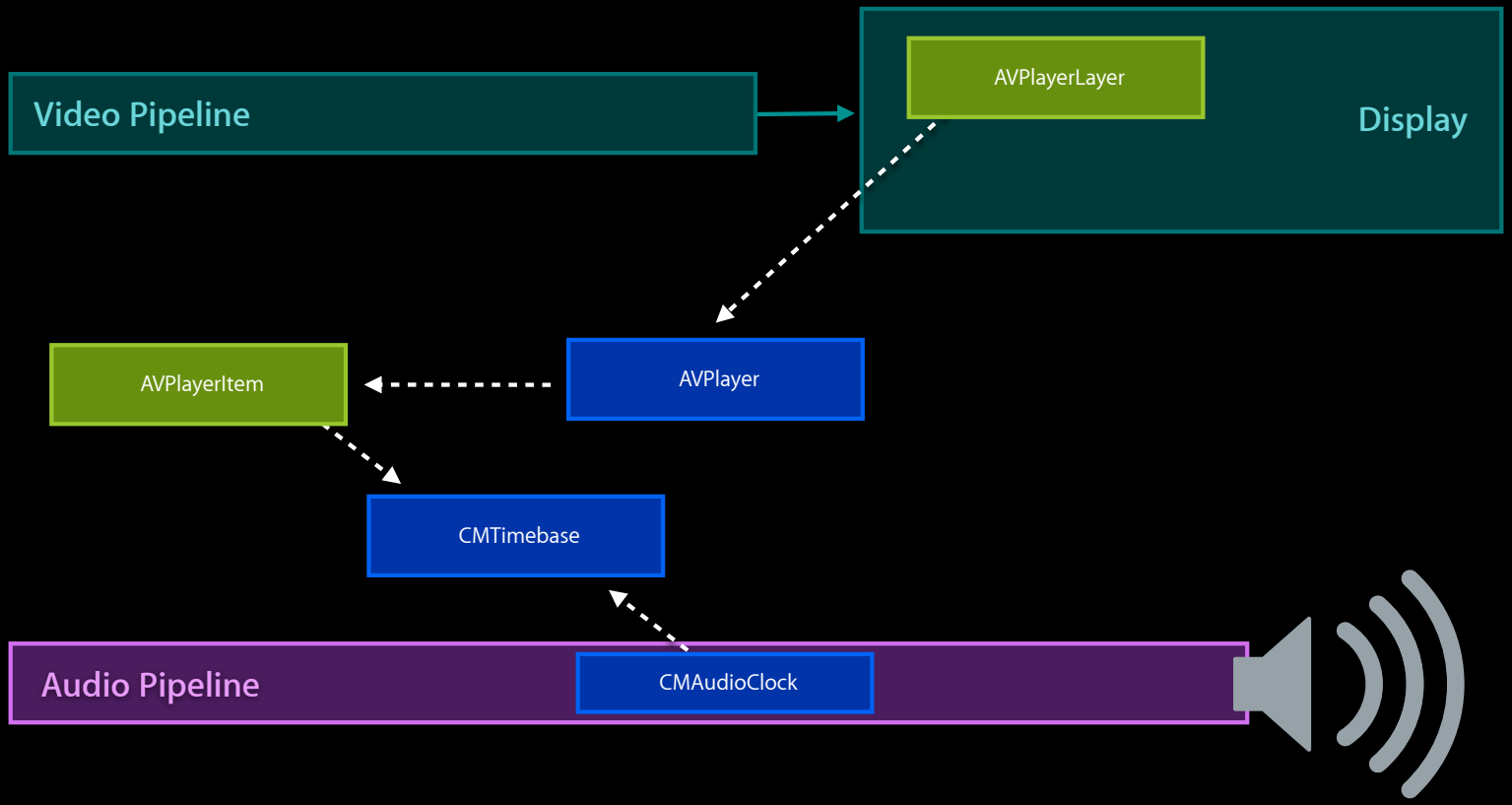
Synchronizing AVPlayer

Synchronizing AVPlayer with a custom timeline

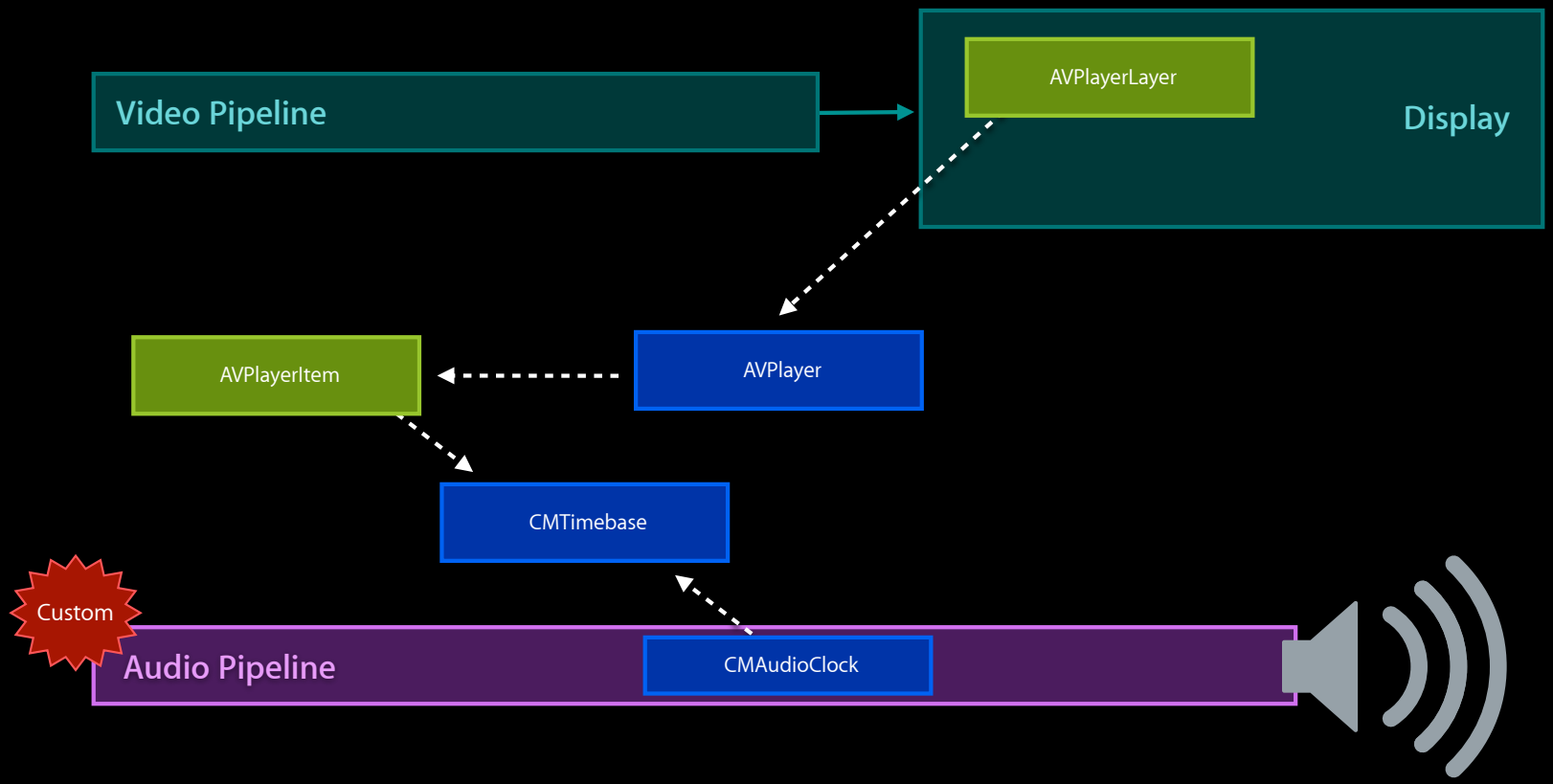
Synchronizing Video to Custom Audio



Synchronizing Video to Custom Audio

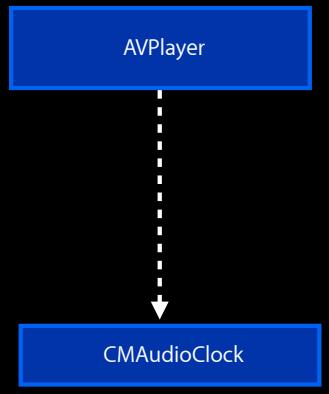


Synchronizing Video to Custom Audio



Synchronizing Video to Custom Audio

Synchronizing Video to Custom Audio



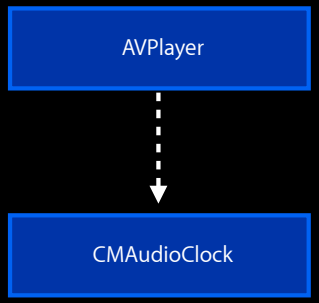
Synchronizing with AVPlayer

- Stay in sync

```
@property (nonatomic, retain) CMClockRef masterClock
```

- Start in sync

```
– setRate:(float)rate time:(CMTime)itemTime atHostTime:
(CMTime)hostClockTime
```



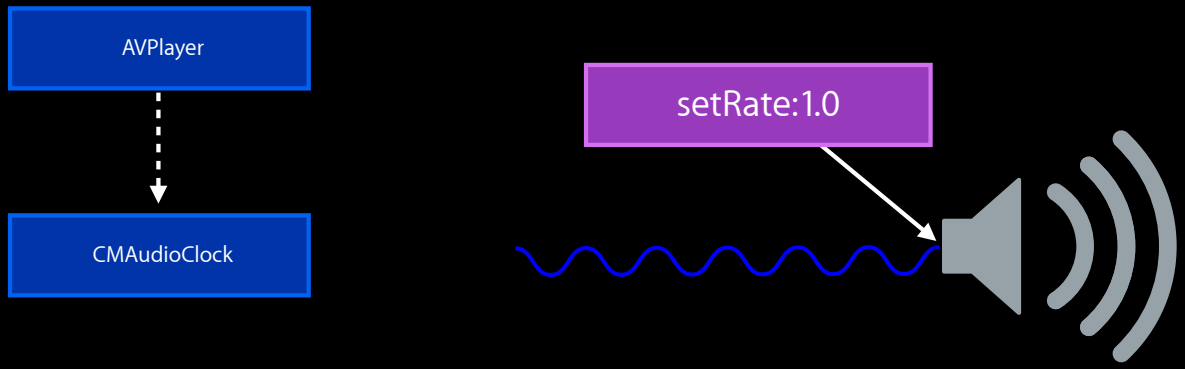
Synchronizing with AVPlayer

- Stay in sync

```
@property (nonatomic, retain) CMClockRef masterClock
```

- Start in sync

```
– setRate:(float)rate time:(CMTime)itemTime atHostTime:
(CMTime)hostClockTime
```

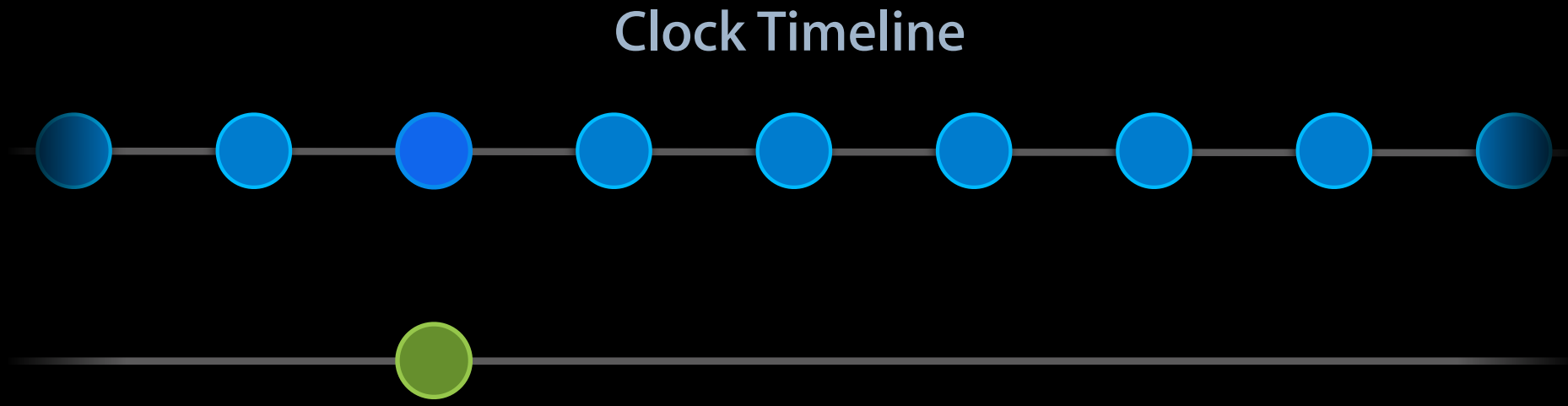


Synchronizing with AVPlayer

```
CMTime CMClockMakeHostTimeFromSystemUnits( uint64_t hostTime );

- setRate:(float)rate time:(CMTime)itemTime
                atHostTime:(CMTime)hostClockTime;
```

Synchronizing with AVPlayer

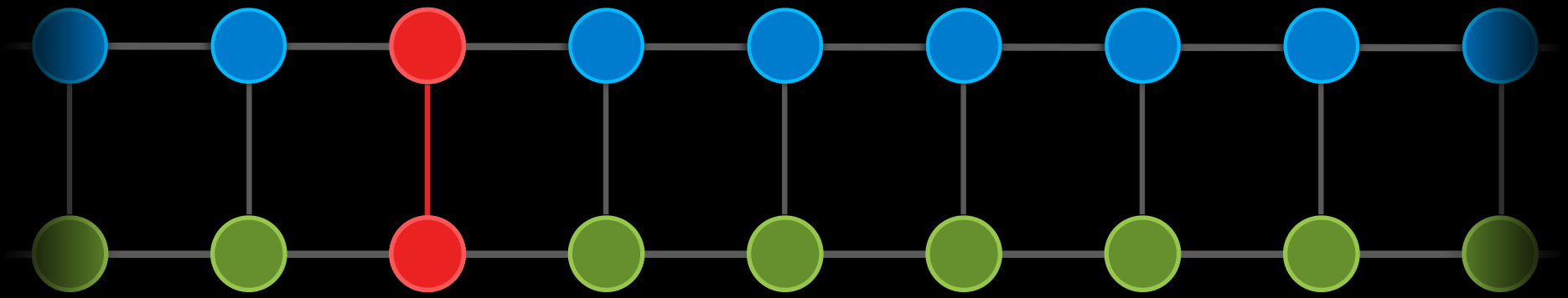
```
CMTime CMClockMakeHostTimeFromSystemUnits( uint64_t hostTime );

- setRate:(float)rate time:(CMTime)itemTime
                atHostTime:(CMTime)hostClockTime;
```



Synchronizing with AVPlayer

```
CMTime CMClockMakeHostTimeFromSystemUnits( uint64_t hostTime );

- setRate:(float)rate time:(CMTime)itemTime
                atHostTime:(CMTime)hostClockTime;
```

Clock Timeline



Timebase Timeline

Synchronizing with AVPlayer

Priming ahead of some future time

```
- prerollAtRate:(float)rate
        completionHandler:(void (^)(BOOL finished))completionHandler

- cancelPendingPrerolls
```

Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished){
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished){
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished) {
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished){
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished){
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished){
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

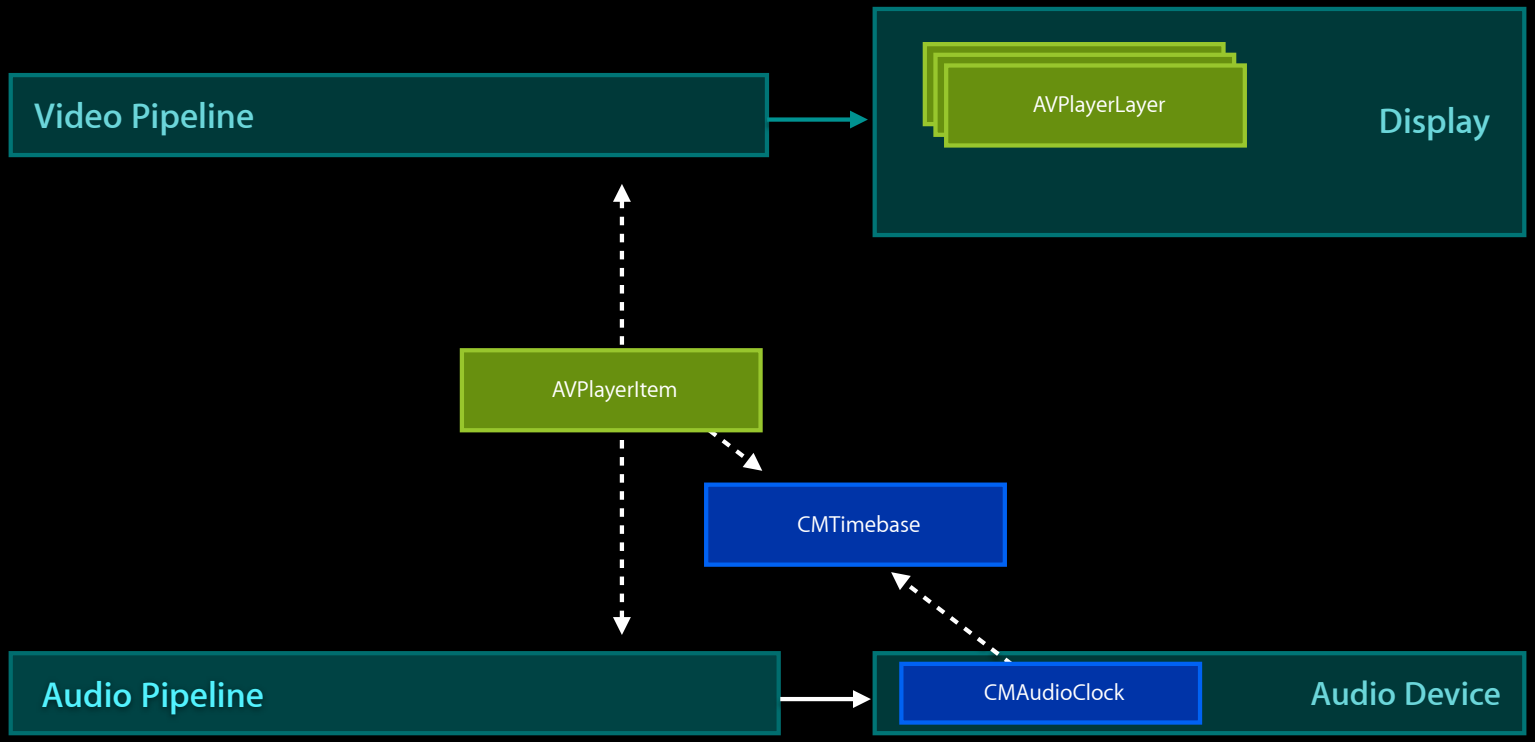
Synchronizing Video to Custom Audio

```
CMClockRef audioClock = NULL;
OSStatus err = CMAudioClockCreate(kCFAllocatorDefault, &audioClock);
if (err == noErr) {
    [myPlayer setMasterClock:audioClock];
    [myPlayer prerollAtRate:1.0 completionHandler:^(BOOL finished){
        if (finished) {
            // Calculate future host time here
            [myPlayer setRate:1.0 time:newItemTime atHostTime:hostTime];
        }
        else {
                // Preroll interrupted or cancelled
        }
    }];
}
```

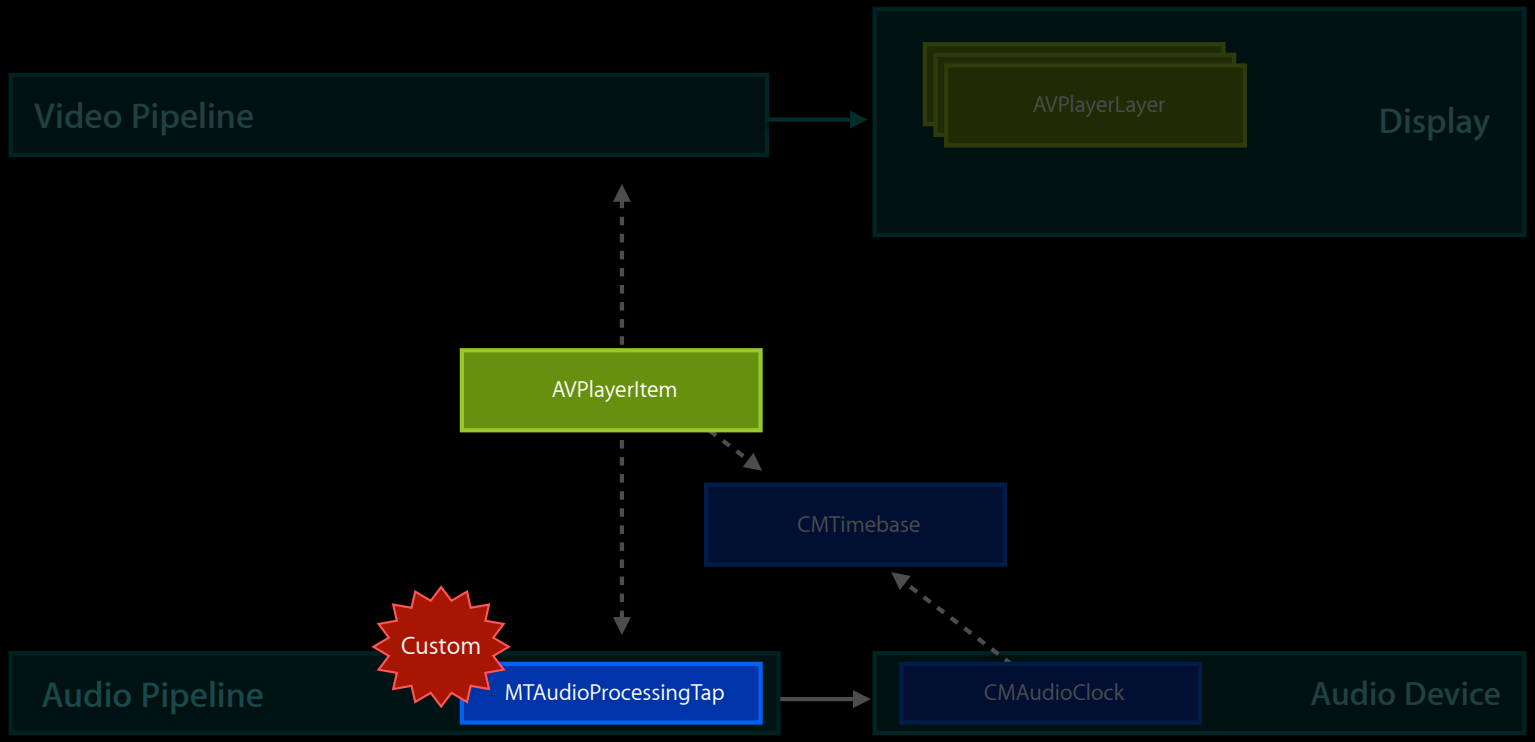
Real-Time Audio Effects and Processing

Audio effects for audiovisual media

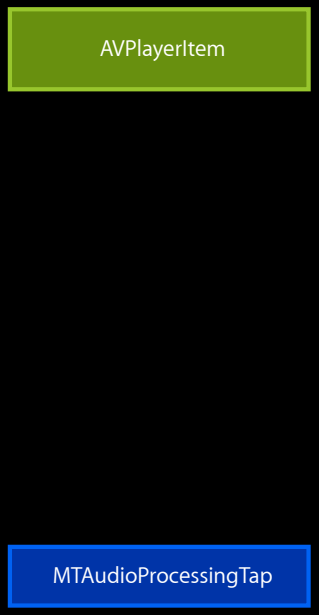
Real-Time Audio Processing



Real-Time Audio Processing

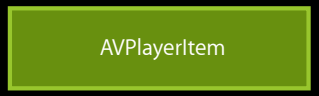


Real-Time Audio Processing



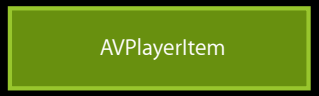
Real-Time Audio Processing

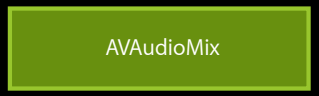
MTAudioProcessingTap



Real-Time Audio Processing

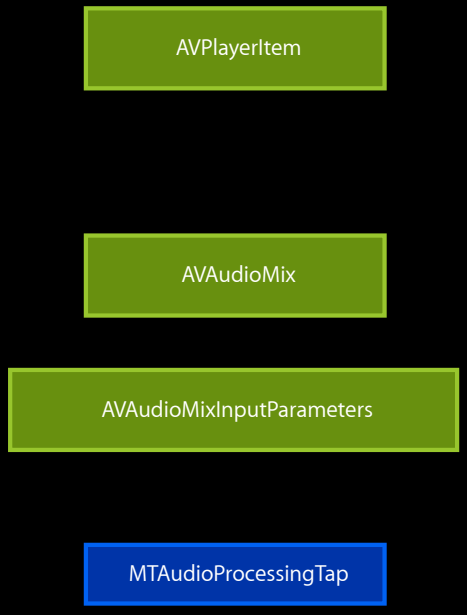
MTAudioProcessingTap





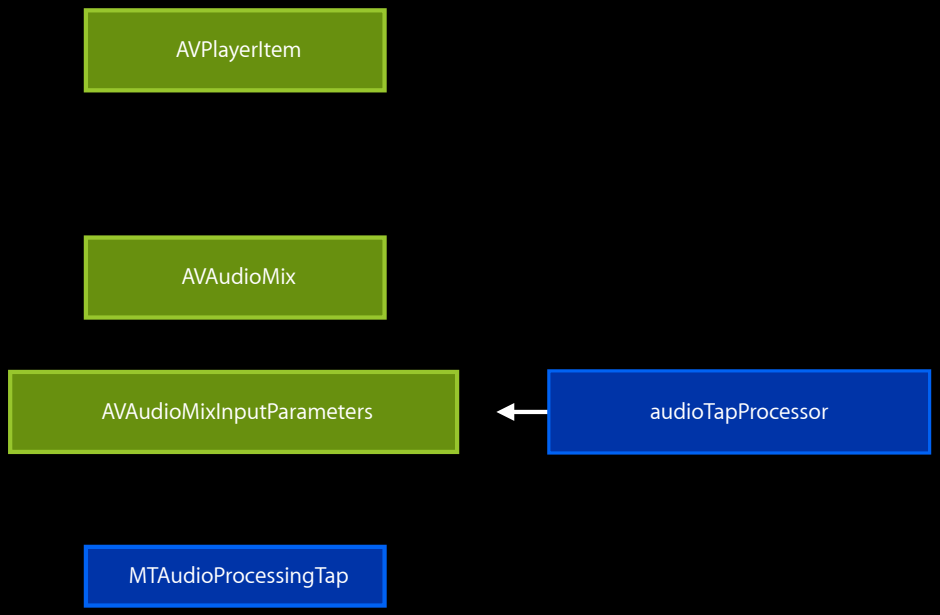
Real-Time Audio Processing

MTAudioProcessingTap



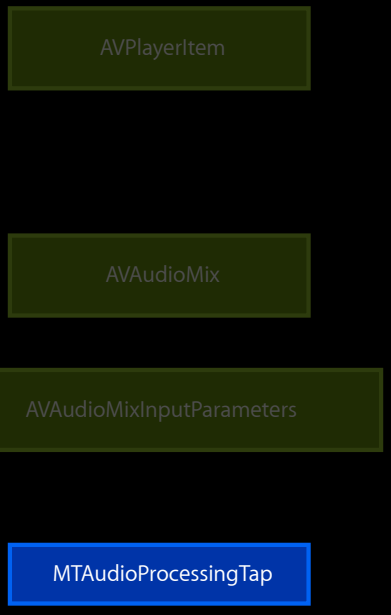
Real-Time Audio Processing

MTAudioProcessingTap



Real-Time Audio Processing

MTAudioProcessingTap



MTAudioProcessingTap

Creation

```
OSStatus
MTAudioProcessingTapCreate( CFAllocatorRef allocator,
                            const MTAudioProcessingTapCallbacks *callbacks,
                            MTAudioProcessingTapCreationFlags flags,
                            MTAudioProcessingTapRef *tapOut );
```

- Manage with CFRetain and CFRelease

MTAudioProcessingTap

Creation

```
OSStatus
MTAudioProcessingTapCreate( CFAllocatorRef allocator,
                            const MTAudioProcessingTapCallbacks *callbacks,
                            MTAudioProcessingTapCreationFlags flags,
                            MTAudioProcessingTapRef *tapOut );


typedef struct {
    int version;
    void *clientInfo;
    MTAudioProcessingTapInitCallback init;
    MTAudioProcessingTapFinalizeCallback finalize;
    MTAudioProcessingTapPrepareCallback prepare;
    MTAudioProcessingTapUnprepareCallback unprepare;
    MTAudioProcessingTapProcessCallback process;
} MTAudioProcessingTapCallbacks;
```

MTAudioProcessingTap

Lifecycle callbacks

```
typedef void
( *MTAudioProcessingTapPrepareCallback ) (
  MTAudioProcessingTapRef tap,
  CMItemCount maxFrames,
  const AudioStreamBasicDescription
    *processingFormat );
```

MTAudioProcessingTapCallbacks
version
clientInfo
init
finalize
prepare
unprepare
process

MTAudioProcessingTap

Lifecycle callbacks

```
typedef void
( *MTAudioProcessingTapUnprepareCallback ) (
  MTAudioProcessingTapRef tap );
```

MTAudioProcessingTapCallbacks

- version
- clientInfo
- init
- finalize
- prepare
- unprepare
- process

MTAudioProcessingTap

Lifecycle callbacks

```
typedef void
( *MTAudioProcessingTapProcessCallback ) (
  MTAudioProcessingTapRef tap,
  CMItemCount numberFrames,
  MTAudioProcessingTapFlags flags,
  AudioBufferList *bufferListInOut,
  CMItemCount *numberFramesOut,
  MTAudioProcessingTapFlags *flagsOut );
```

MTAudioProcessingTapCallbacks
version
clientInfo
init
finalize
prepare
unprepare
process

MTAudioProcessingTap

Lifecycle callbacks

```
typedef void
( *MTAudioProcessingTapProcessCallback ) (
  MTAudioProcessingTapRef tap,
  CMItemCount numberFrames,
  MTAudioProcessingTapFlags flags,
  AudioBufferList *bufferListInOut,
  CMItemCount *numberFramesOut,
  MTAudioProcessingTapFlags *flagsOut );
```

MTAudioProcessingTapCallbacks
version
clientInfo
init
finalize
prepare
unprepare
process

MTAudioProcessingTap

```
MTAudioProcessingTapGetSourceAudio( MTAudioProcessingTapRef tap,
                                    CMItemCount numberFrames,
                                    AudioBufferList *bufferListInOut,
                                    MTAudioProcessingTapFlags *flagsOut,
                                    CMTimeRange *timeRangeOut,
                                    CMItemCount *numberFramesOut );
```

MTAudioProcessingTap

- Processing callback constraints
  - Don’t:
    - Acquire locks
    - Allocate or deallocate memory
  - Do:
    - Be efficient
    - Consider CMSimpleQueue

Your Processing Callback

```
void myProcessCallback(MTAudioProcessingTapRef tap,
                       CMItemCount numberFrames,
                       MTAudioProcessingTapFlags flags,
                       AudioBufferList *bufferListInOut,
                       CMItemCount *numberFramesOut,
                       MTAudioProcessingTapFlags *flagsOut)
{
 CMTimeRange timeRange;
 OSStatus err = MTAudioProcessingTapGetSourceAudio( tap,
                                                    numberFrames,
                                                    bufferListInOut,
                                                    flagsOut,
                                                    &timeRange,
                                                    numberFramesOut );
```

Your Processing Callback

```
void myProcessCallback(MTAudioProcessingTapRef tap,
                       CMItemCount numberFrames,
                       MTAudioProcessingTapFlags flags,
                       AudioBufferList *bufferListInOut,
                       CMItemCount *numberFramesOut,
                       MTAudioProcessingTapFlags *flagsOut)
{
 CMTimeRange timeRange;
 OSStatus err = MTAudioProcessingTapGetSourceAudio( tap,
                                                    numberFrames,
                                                    bufferListInOut,
                                                    flagsOut,
                                                    &timeRange,
                                                    numberFramesOut );
```

```
    ...

    for (CMItemCount i = 0; i < bufferListInOut->mNumberBuffers; i++)
    {
        AudioBuffer* pBuffer = &bufferListInOut->mBuffers[i];
        UInt32 cSamples = numberFrames *
                  ( _isNonInterleaved ? 1 : pBuffer->mNumberChannels);

        float* pData = (float *) pBuffer->mData;

        for (UInt32 j = 0; j < cSamples; j++) {

               // do something with each sample

      }
    }
}
```

```
    ...

    for (CMItemCount i = 0; i < bufferListInOut->mNumberBuffers; i++)
    {
        AudioBuffer* pBuffer = &bufferListInOut->mBuffers[i];
        UInt32 cSamples = numberFrames *
                  ( _isNonInterleaved ? 1 : pBuffer->mNumberChannels);

        float* pData = (float *) pBuffer->mData;

        for (UInt32 j = 0; j < cSamples; j++) {

              // do something with each sample

      }
    }
}
```

```
    ...

    for (CMItemCount i = 0; i < bufferListInOut->mNumberBuffers; i++)
    {
        AudioBuffer* pBuffer = &bufferListInOut->mBuffers[i];
        UInt32 cSamples = numberFrames *
                  ( _isNonInterleaved ? 1 : pBuffer->mNumberChannels);

        float* pData = (float *) pBuffer->mData;

        for (UInt32 j = 0; j < cSamples; j++) {

               // do something with each sample

      }
    }
}
```

```
    ...

    for (CMItemCount i = 0; i < bufferListInOut->mNumberBuffers; i++)
    {
        AudioBuffer* pBuffer = &bufferListInOut->mBuffers[i];
        UInt32 cSamples = numberFrames *
                  ( _isNonInterleaved ? 1 : pBuffer->mNumberChannels);

        float* pData = (float *) pBuffer->mData;

        for (UInt32 j = 0; j < cSamples; j++) {

              // do something with each sample

      }
    }
}
```

```
    ...

    for (CMItemCount i = 0; i < bufferListInOut->mNumberBuffers; i++)
    {
        AudioBuffer* pBuffer = &bufferListInOut->mBuffers[i];
        UInt32 cSamples = numberFrames *
                  ( _isNonInterleaved ? 1 : pBuffer->mNumberChannels);

        float* pData = (float *) pBuffer->mData;

        for (UInt32 j = 0; j < cSamples; j++) {

               // do something with each sample

      }
    }
}
```

Demo

AudioTapProcessor: using Audio Units

AudioProcessingTap Demo

Using Audio Units

- Discover and create audio unit using

```
AudioComponentDescription
```

- Set the render callback on the audio unit

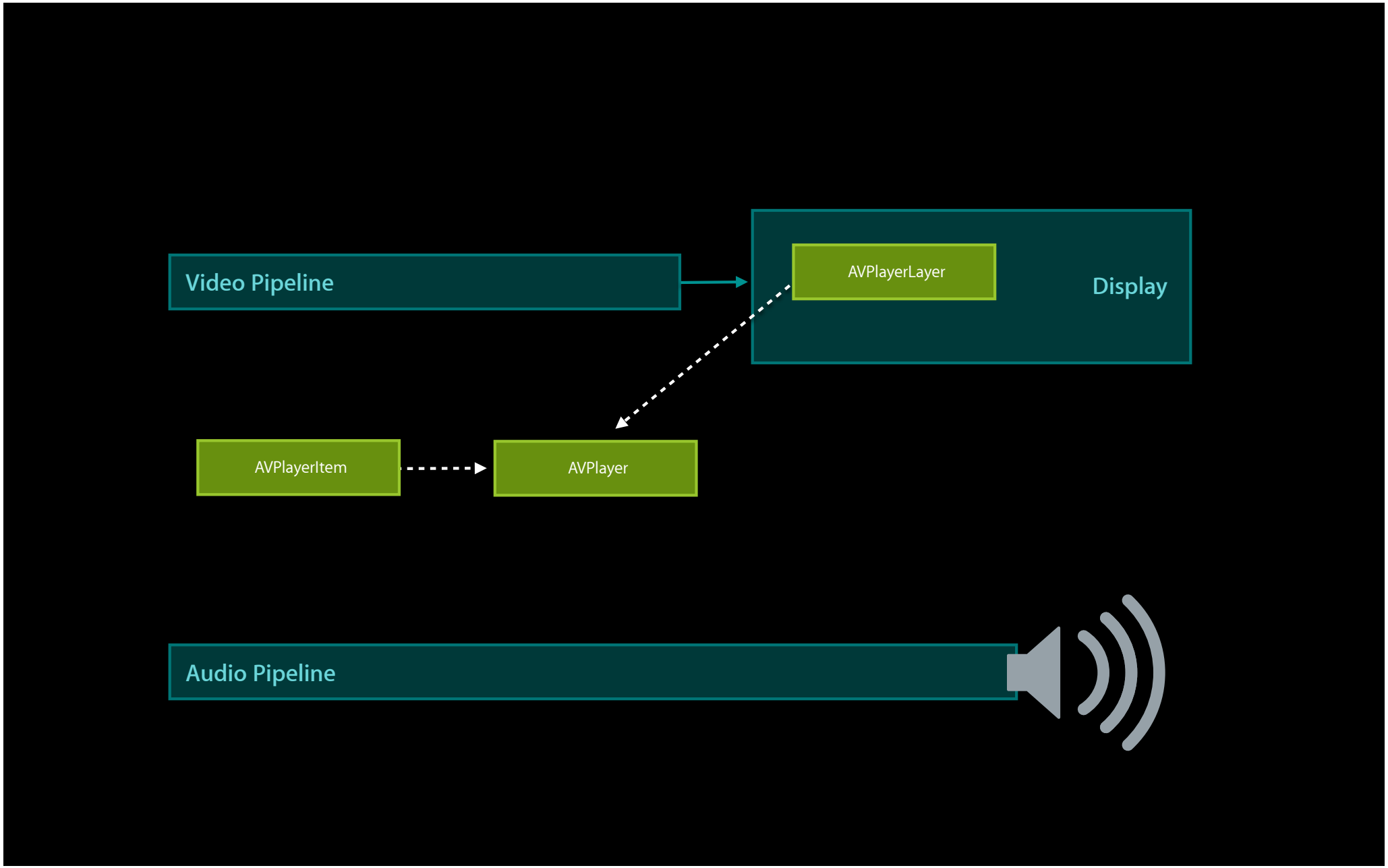
```
kAudioUnitProperty_SetRenderCallback

OSStatus myAudioUnitRenderCallback(
      void *inRefCon, ..., UInt32 inNumberFrames, AudioBufferList *ioData)
{
  return MTAudioProcessingTapGetSourceAudio( inRefCon,
                                             inNumberFrames,
                                             ioData,
                                            NULL, NULL, NULL);
}
```

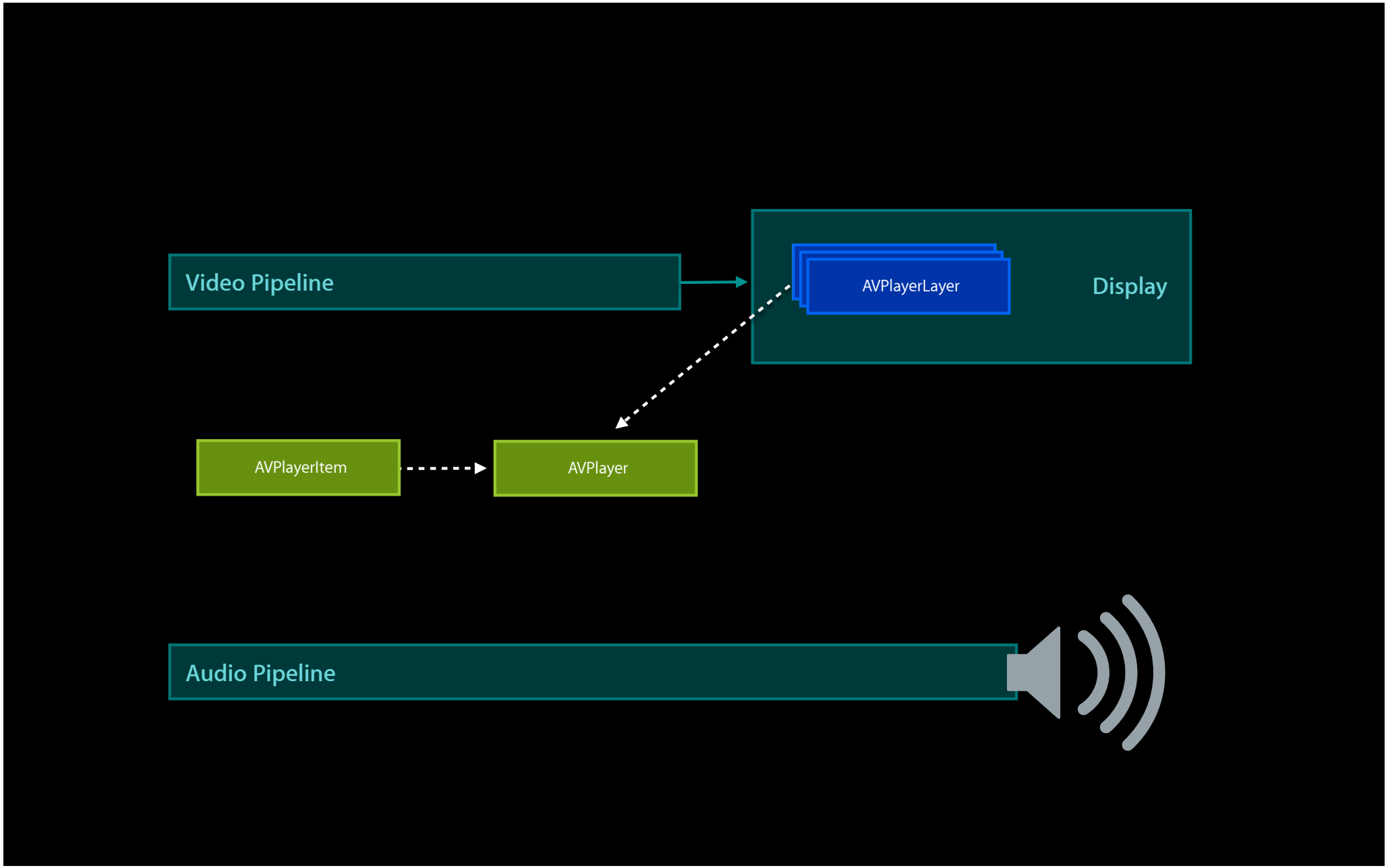
Real-Time Video Effects and Processing

Real-Time Video Effects and Processing

Goals of real-time video output

- Decoded video images during playback
- Output ahead of display-time
  - Custom manipulation
- Maintain A/V sync
  - Timestamped images


Video Pipeline
AVPlayerLayer
Display
AVPlayerItem
AVPlayer
Audio Pipeline


Video Pipeline
AVPlayerLayer
Display
AVPlayerItem
AVPlayer
Audio Pipeline

AVPlayerLayer

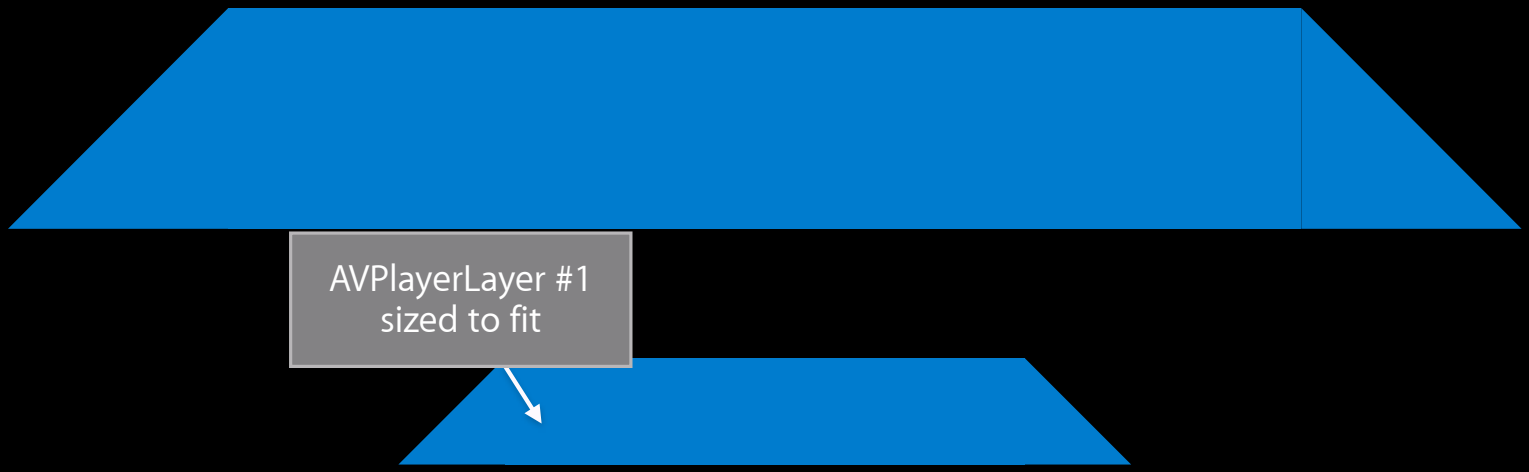
- New support
  - Multiple AVPlayerLayers to one AVPlayer
- Advantages
  - Automatic A/V sync
  - Easy to use

Demo

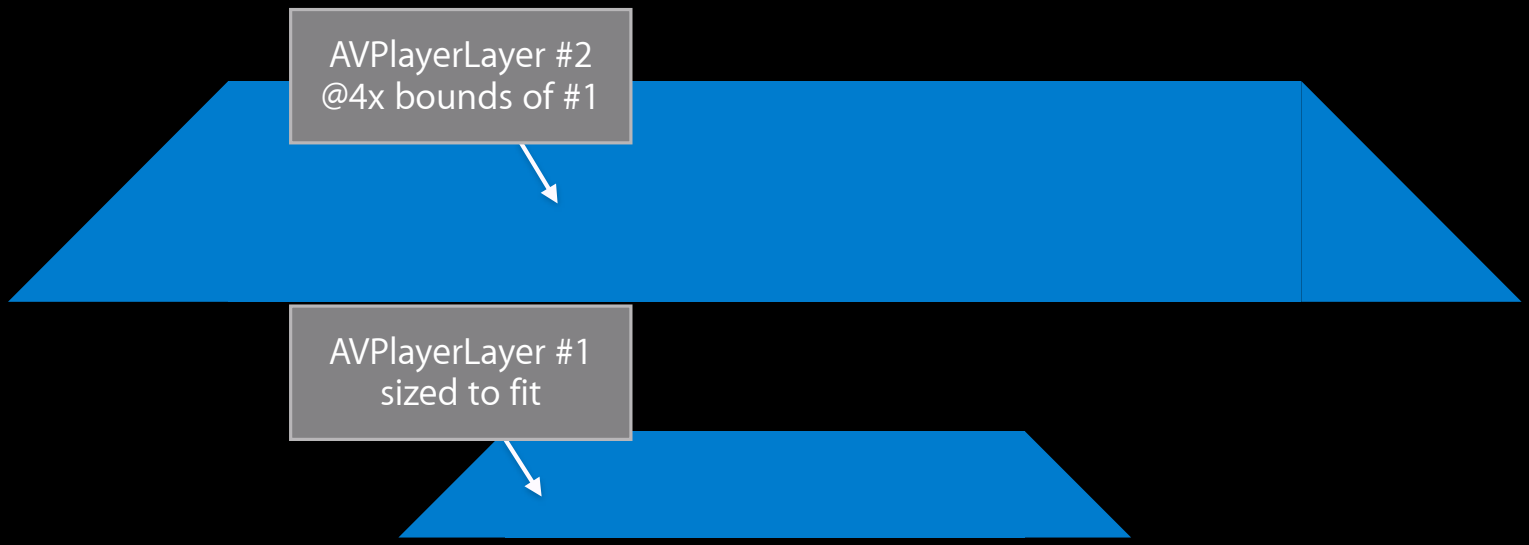
AVLoupe

AVLoupe: How It Works

AVLoupe: How It Works

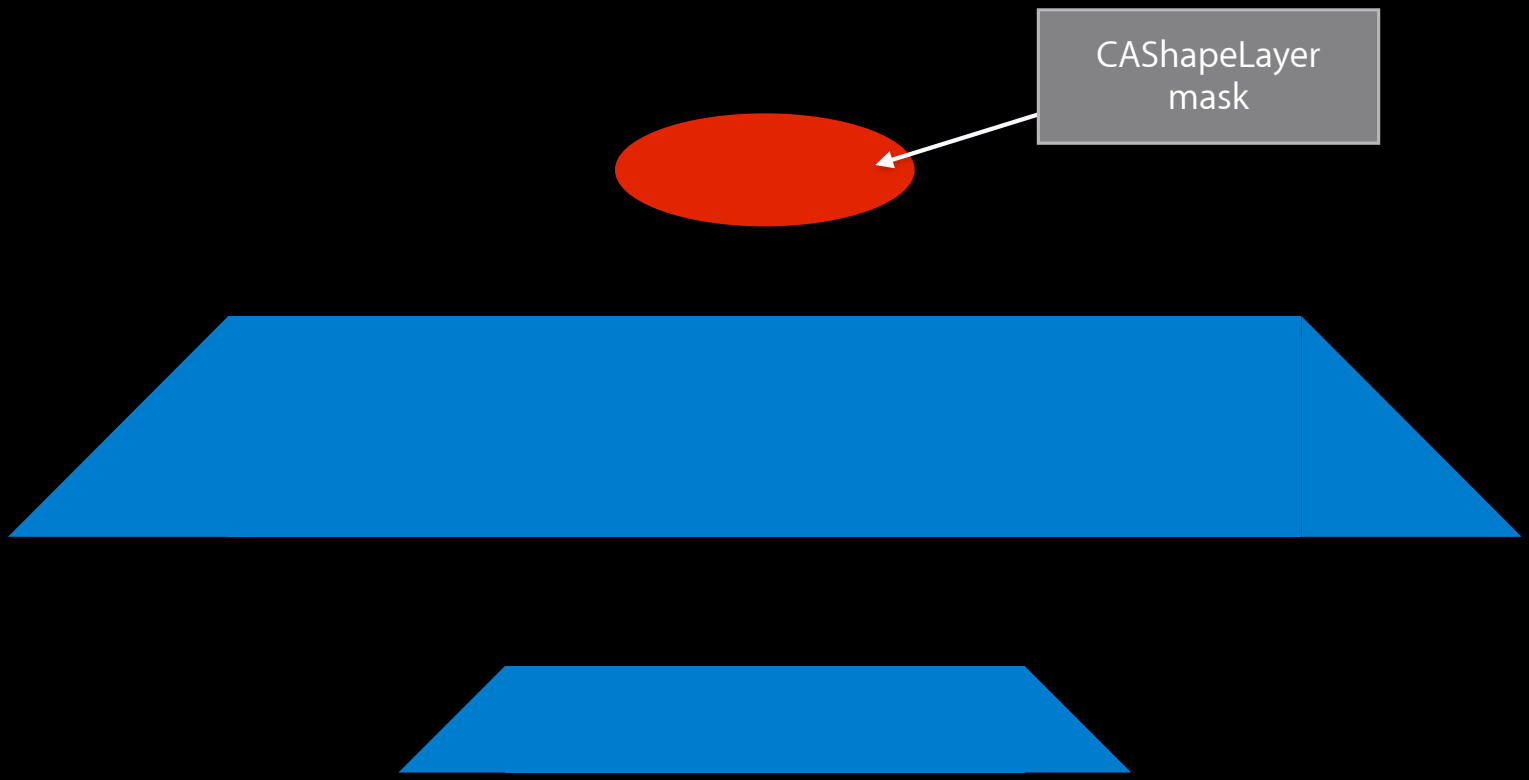


AVLoupe: How It Works

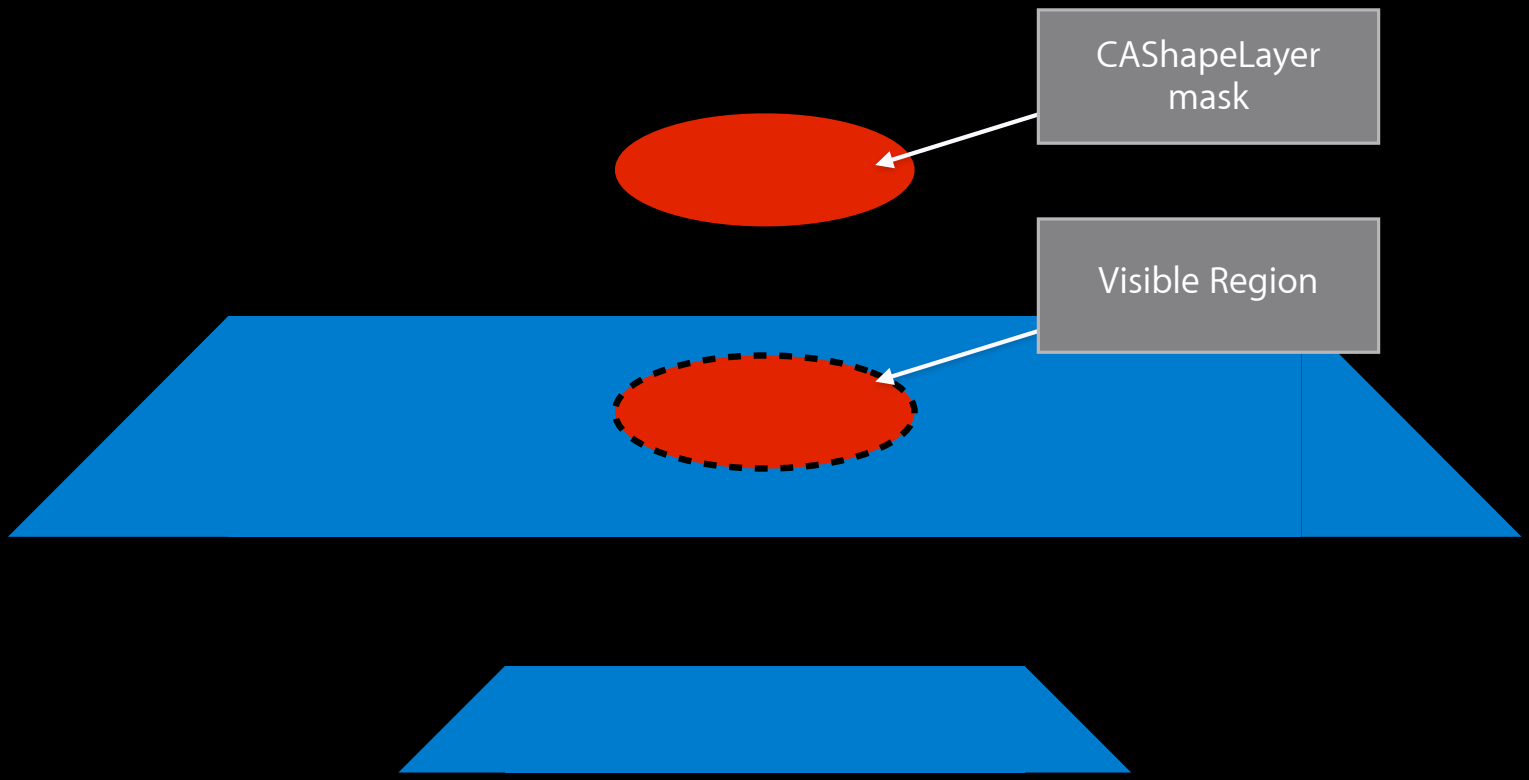


AVLoupe: How It Works

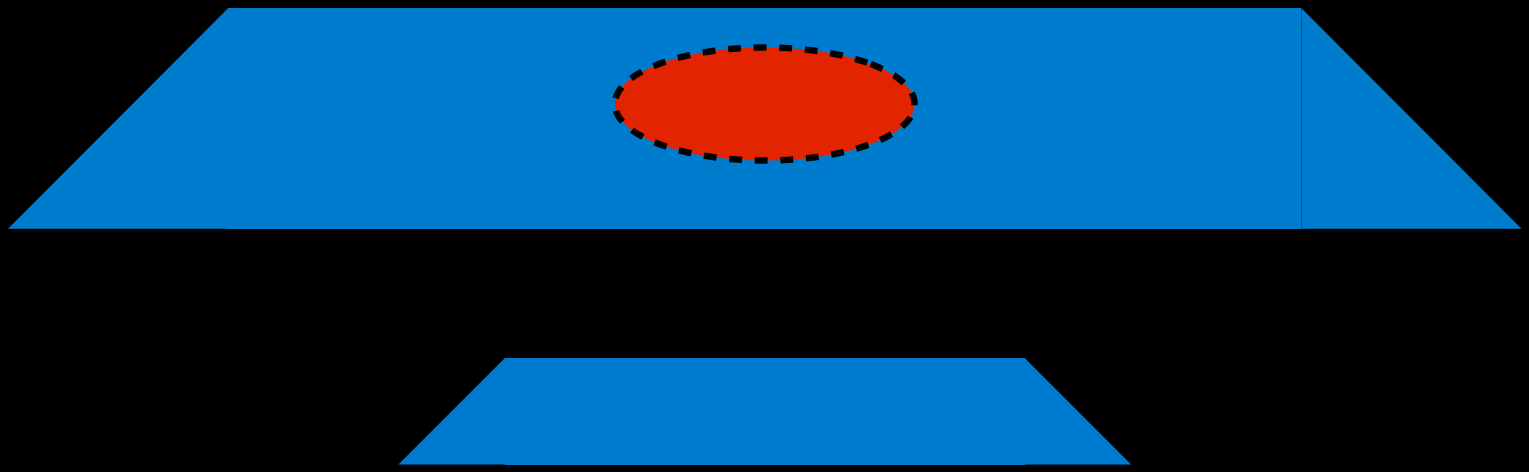
AVLoupe: How It Works



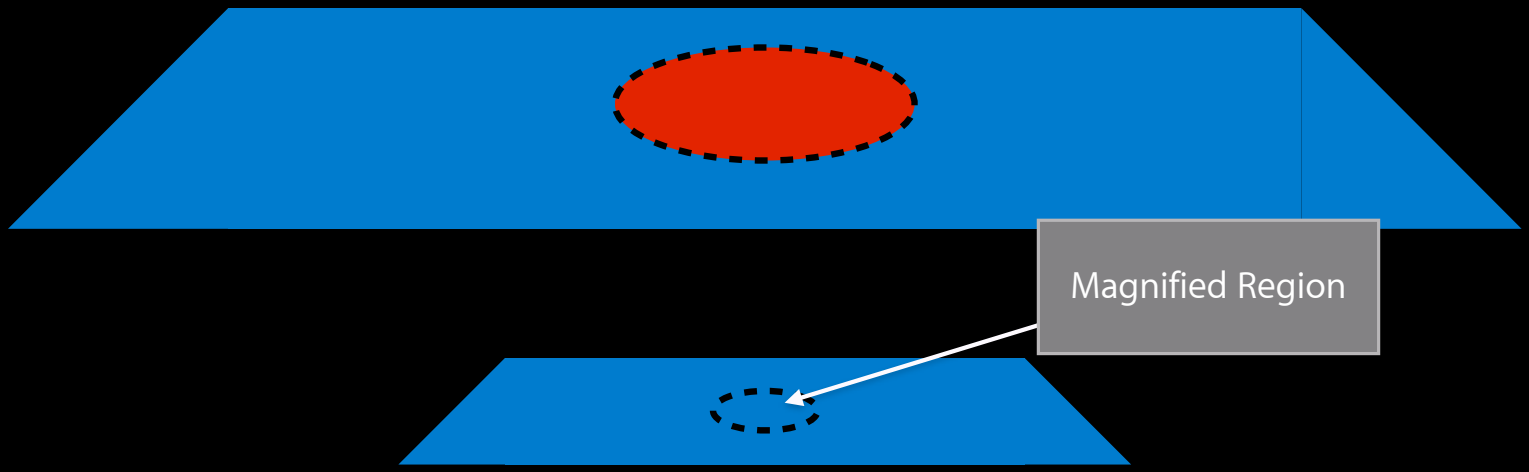
AVLoupe: How It Works



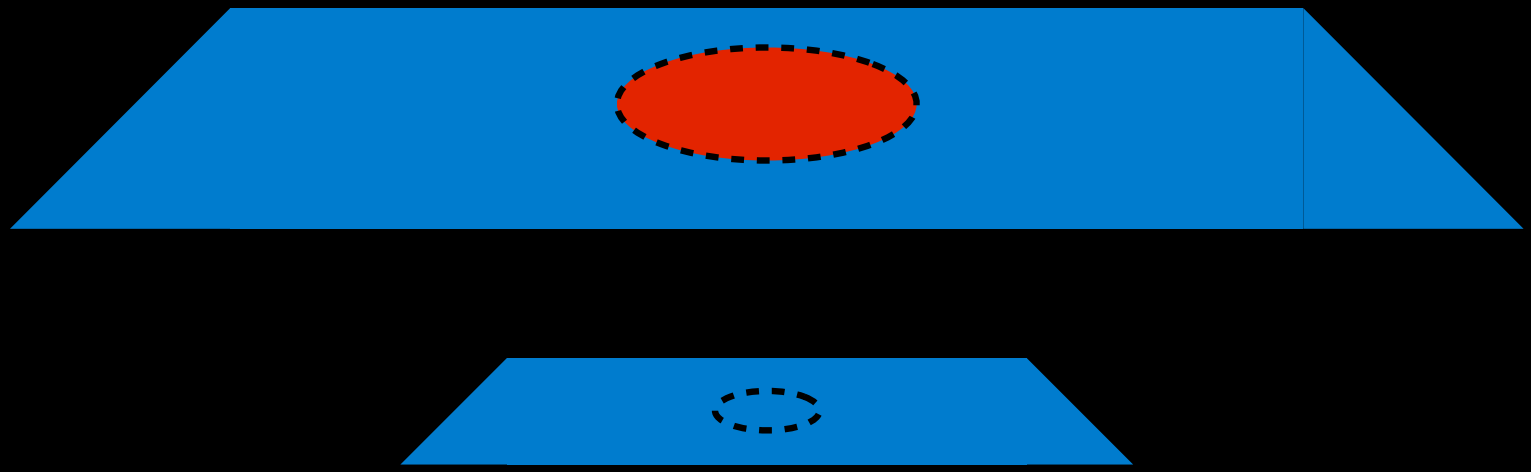
AVLoupe: How It Works



AVLoupe: How It Works

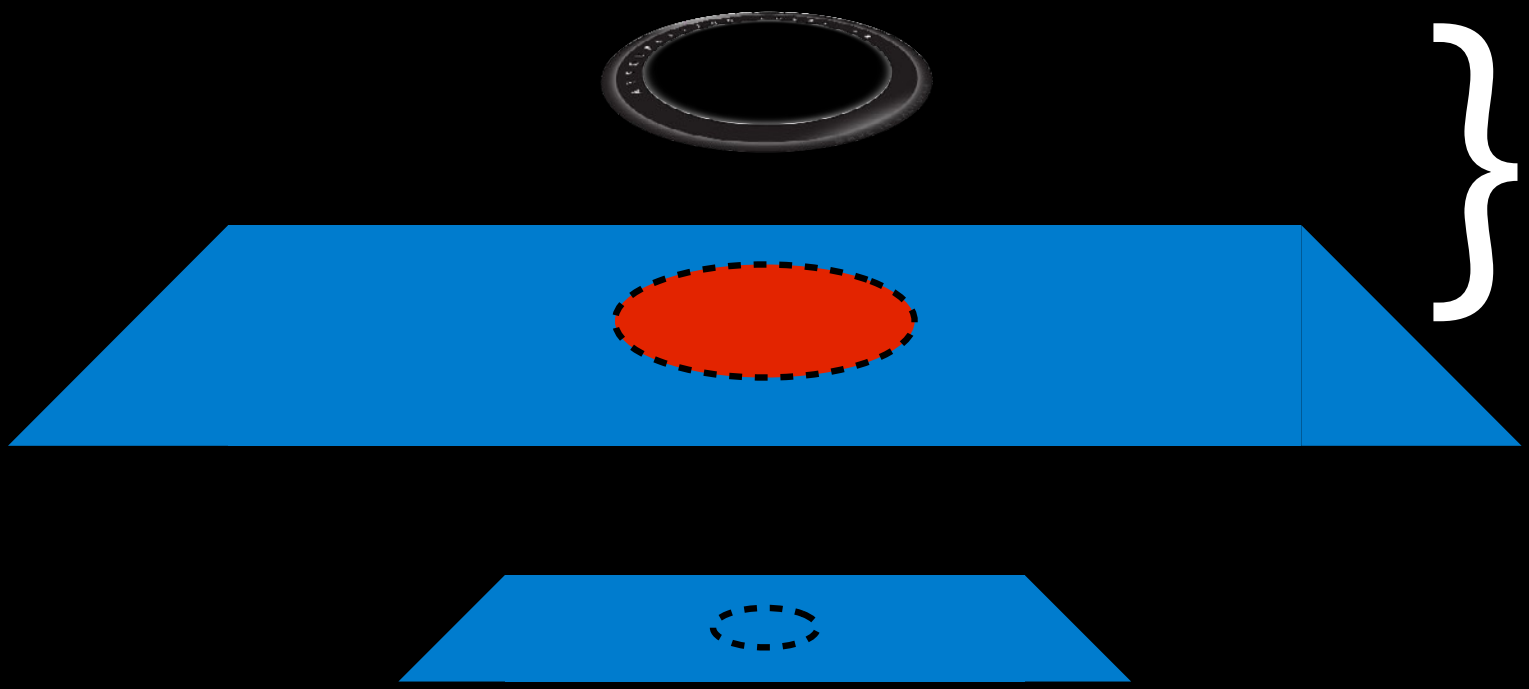


AVLoupe: How It Works

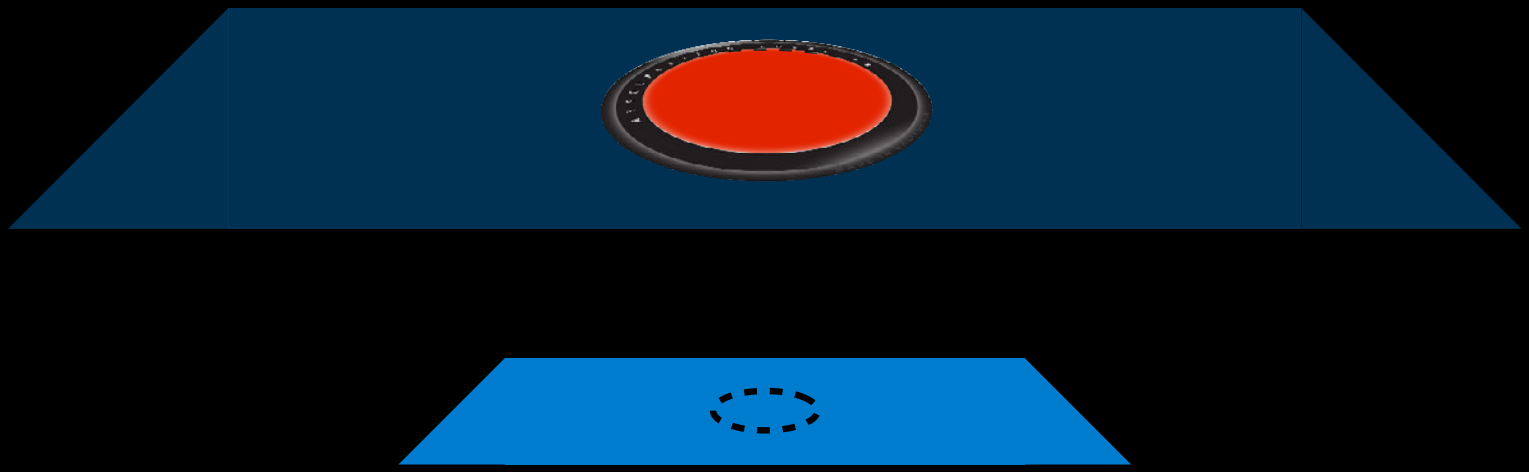


AVLoupe: How It Works

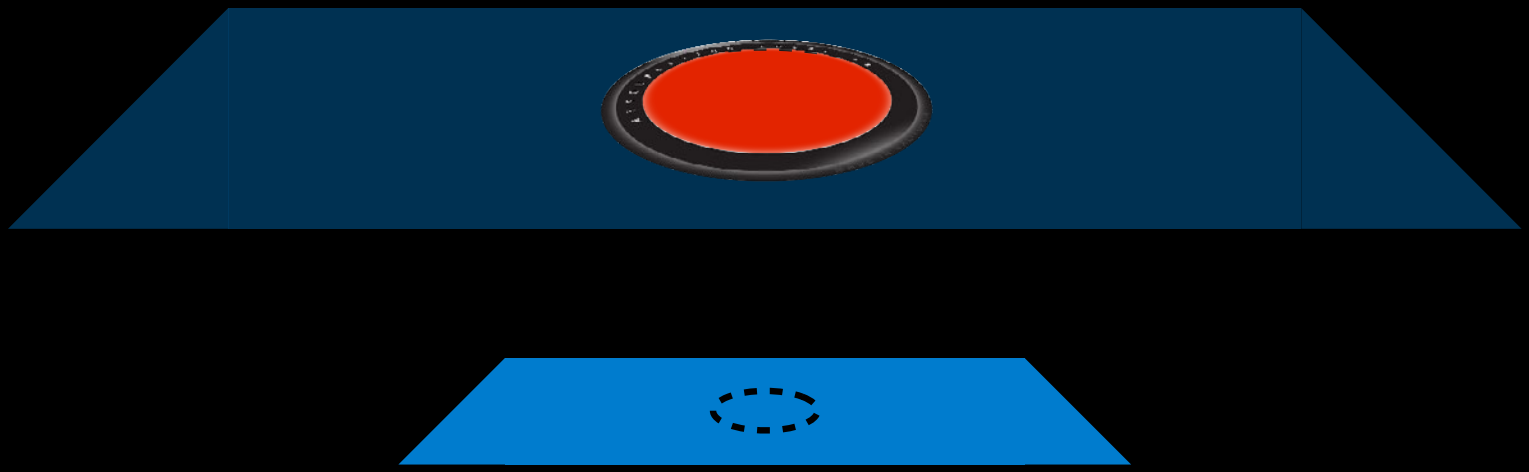
UIImageView
+
UIPanGestureRecognizer



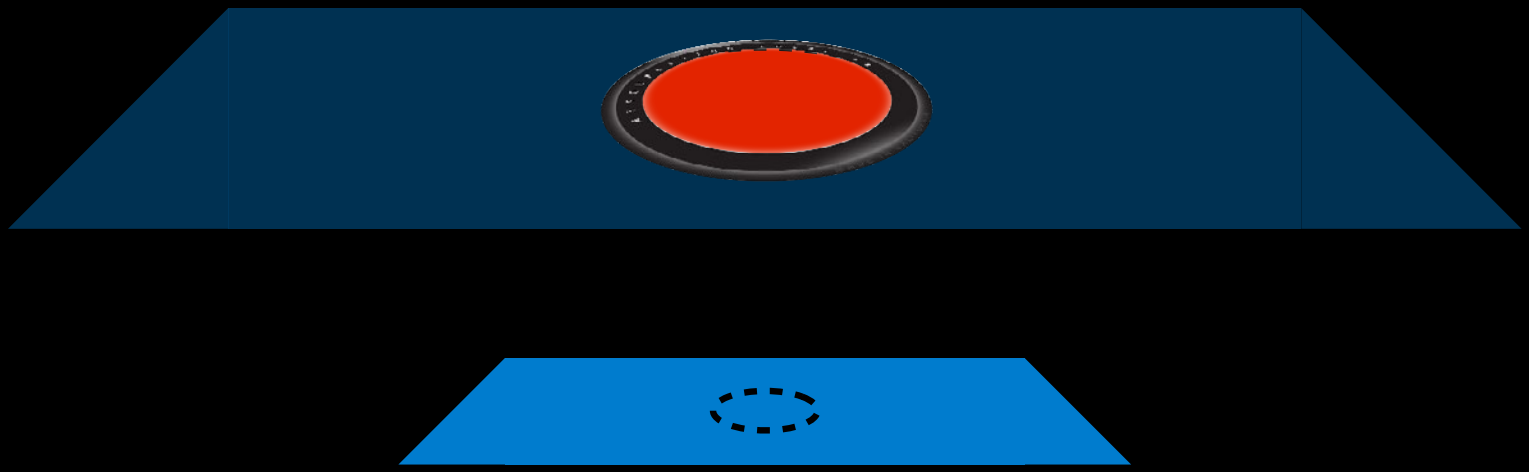
AVLoupe: How It Works



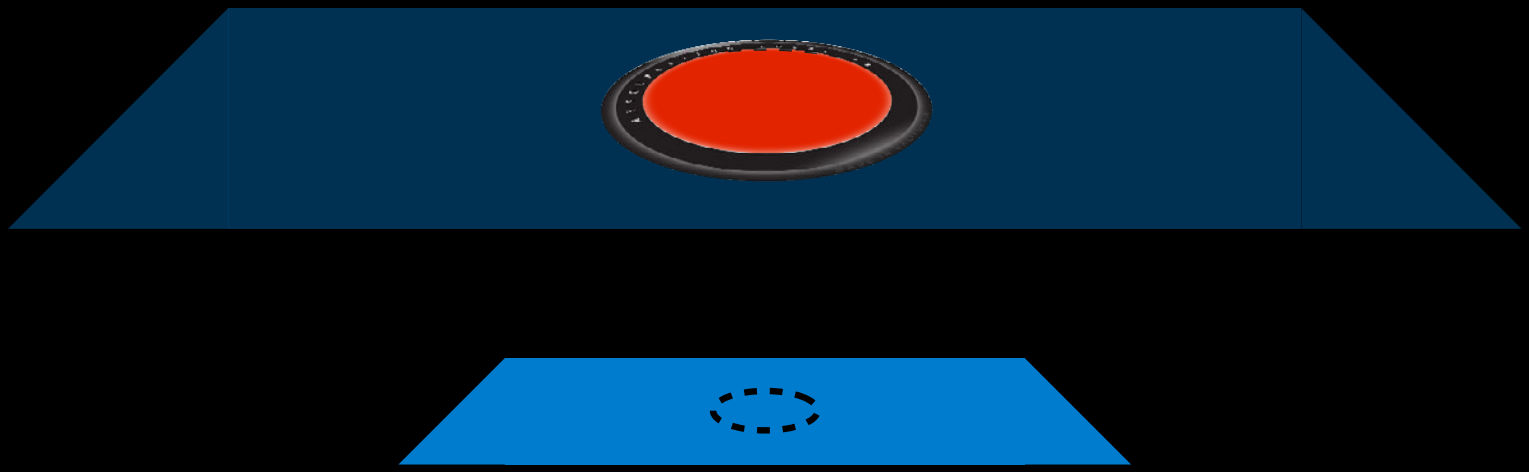
AVLoupe: How It Works



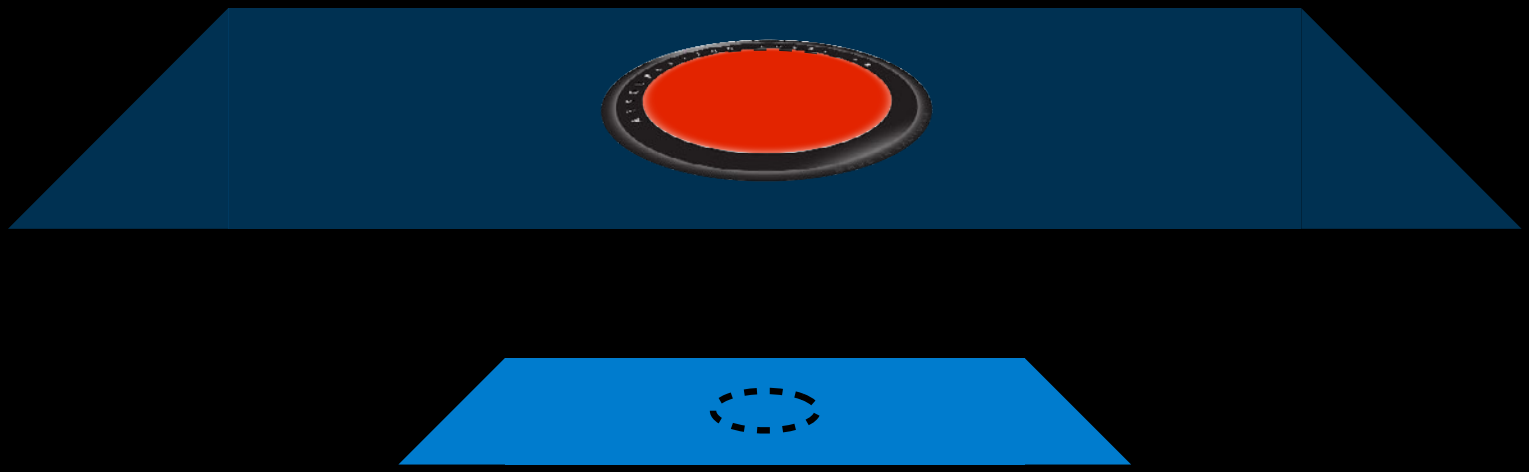
AVLoupe: How It Works



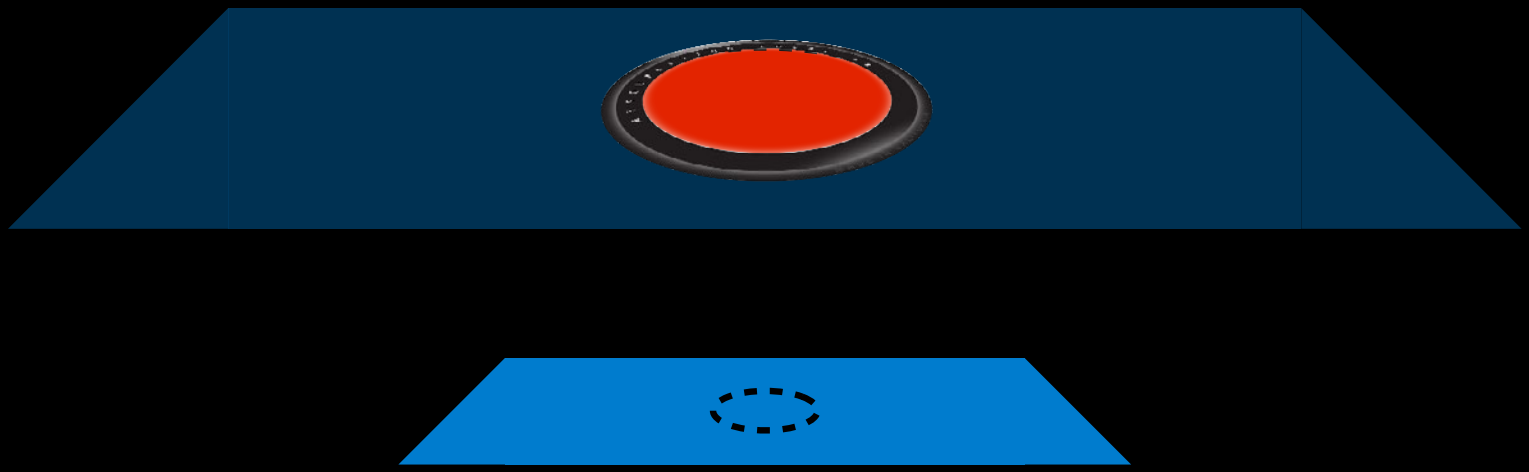
AVLoupe: How It Works



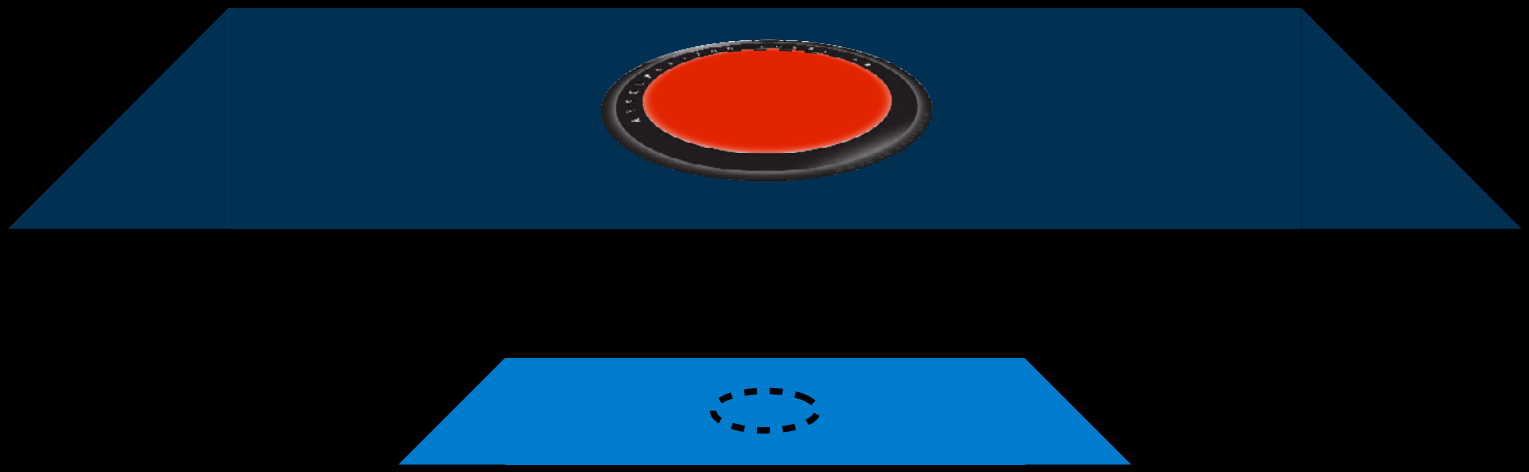
AVLoupe: How It Works

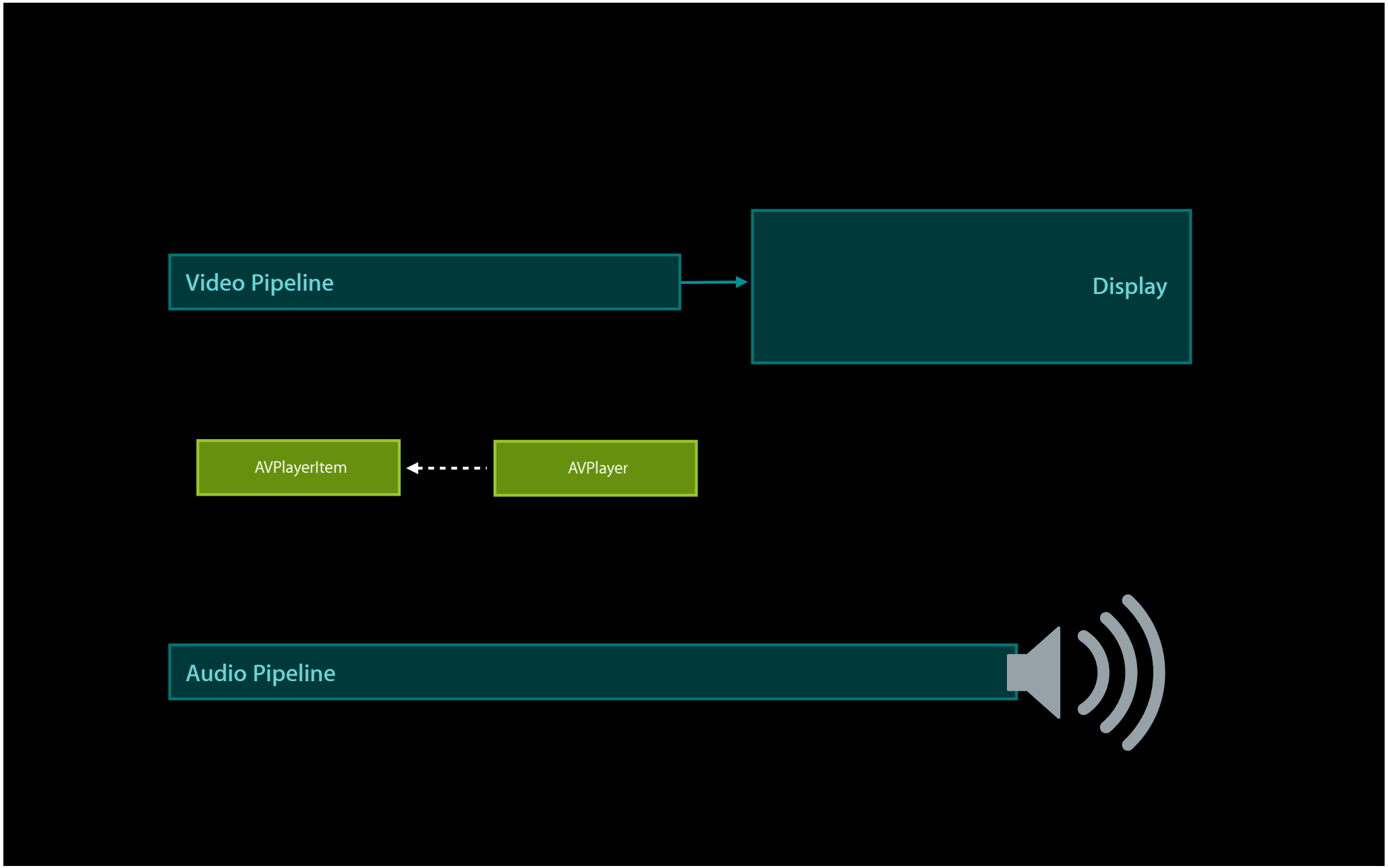


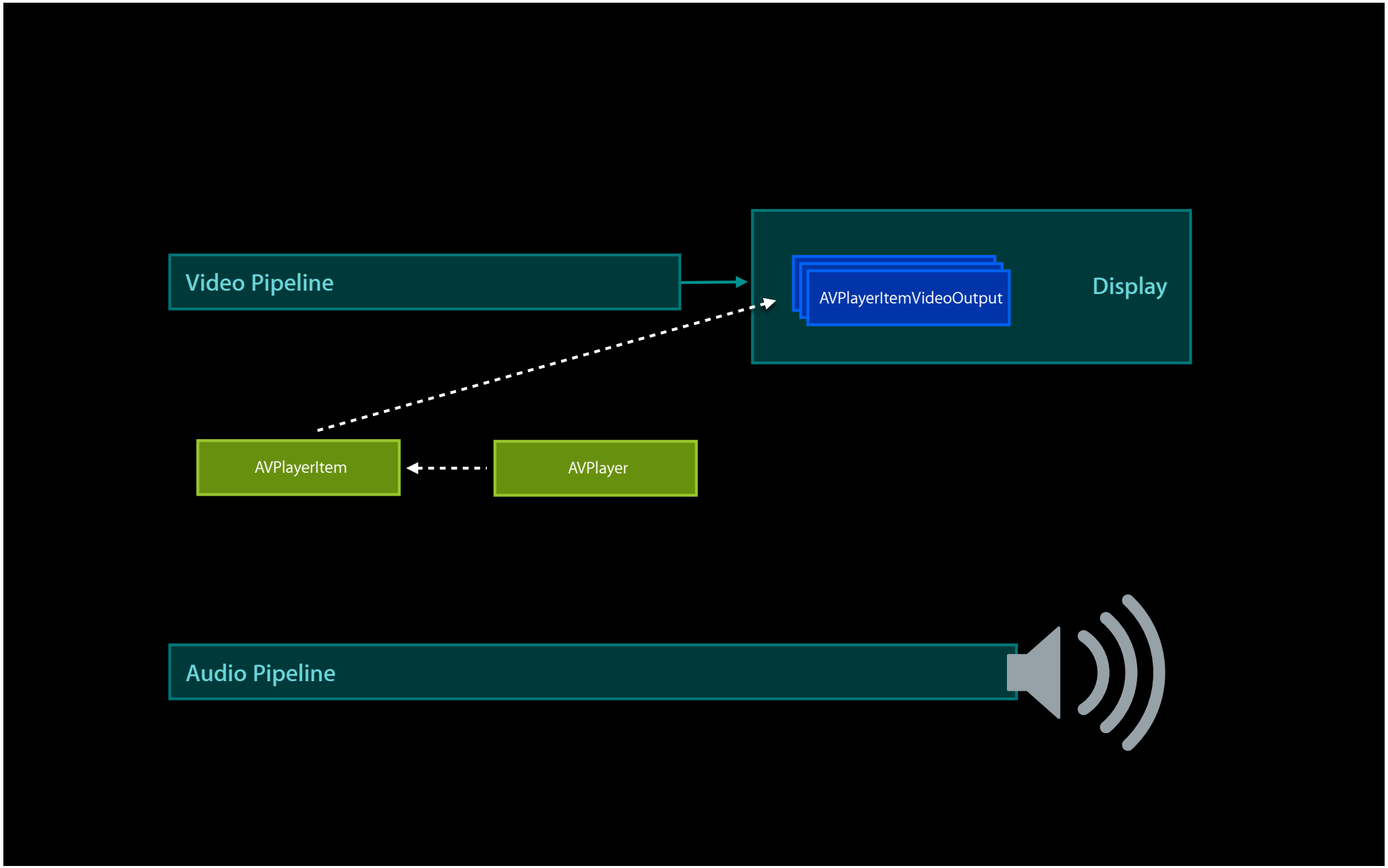
AVLoupe: How It Works



AVLoupe: How It Works




Video Pipeline
Display
AVPlayerItem
AVPlayer
Audio Pipeline


Video Pipeline
AVPlayerItemVideoOutput
Display
AVPlayerItem
AVPlayer
Audio Pipeline

AVPlayerItemVideoOutput

Getting video images in real time

- Custom presentation
  - OpenGL
  - CoreGraphics
- Video processing
  - Effects
  - Analysis

AVPlayerItemVideoOutput

Pulling images from a dynamic model

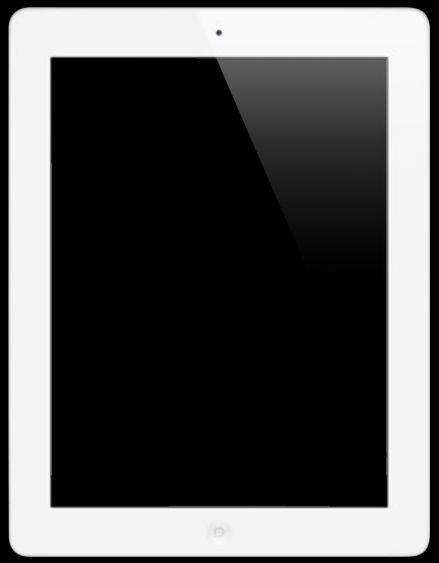
- Emits best available image for an item time

```
hasNewPixelBufferForItemTime:
copyPixelBufferForItemTime:itemTimeForDisplay:
```

- Call for each vertical sync
  - OS X
    - CVDisplayLink
    - CAOpenGLLayer
  - iOS
    - CADisplayLink
    - GLKitView or GLKitViewController

AVPlayerItemVideoOutput

Pulling images from a dynamic model

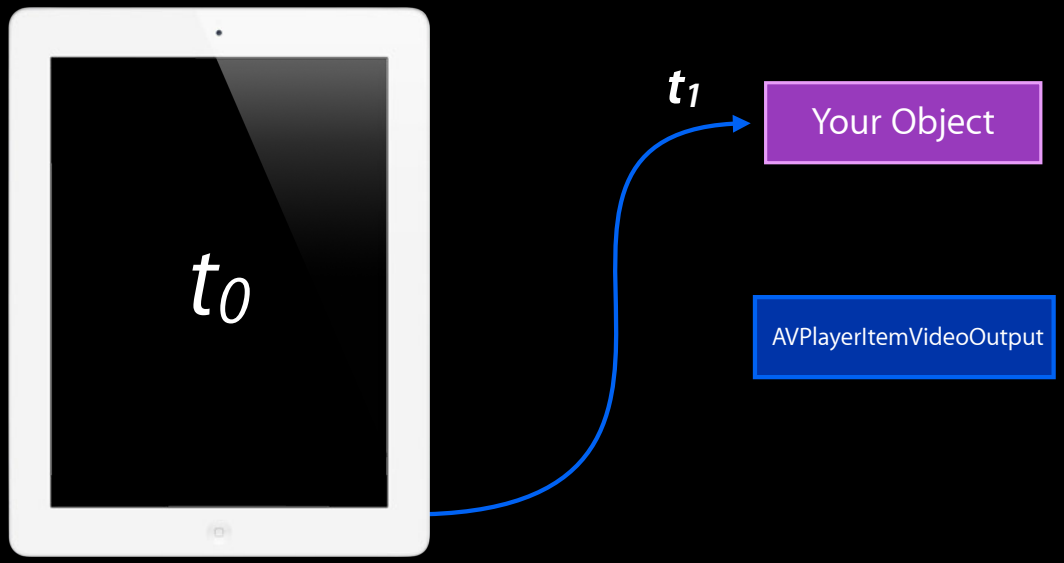


Your Object

AVPlayerItemVideoOutput

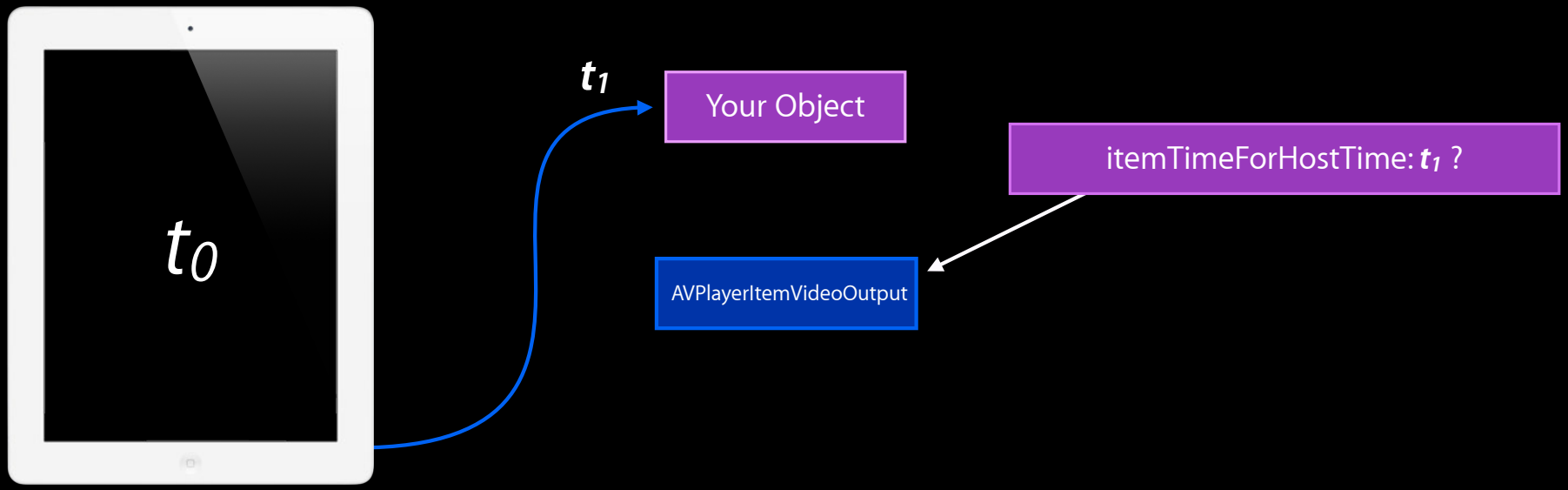
AVPlayerItemVideoOutput

Pulling images from a dynamic model



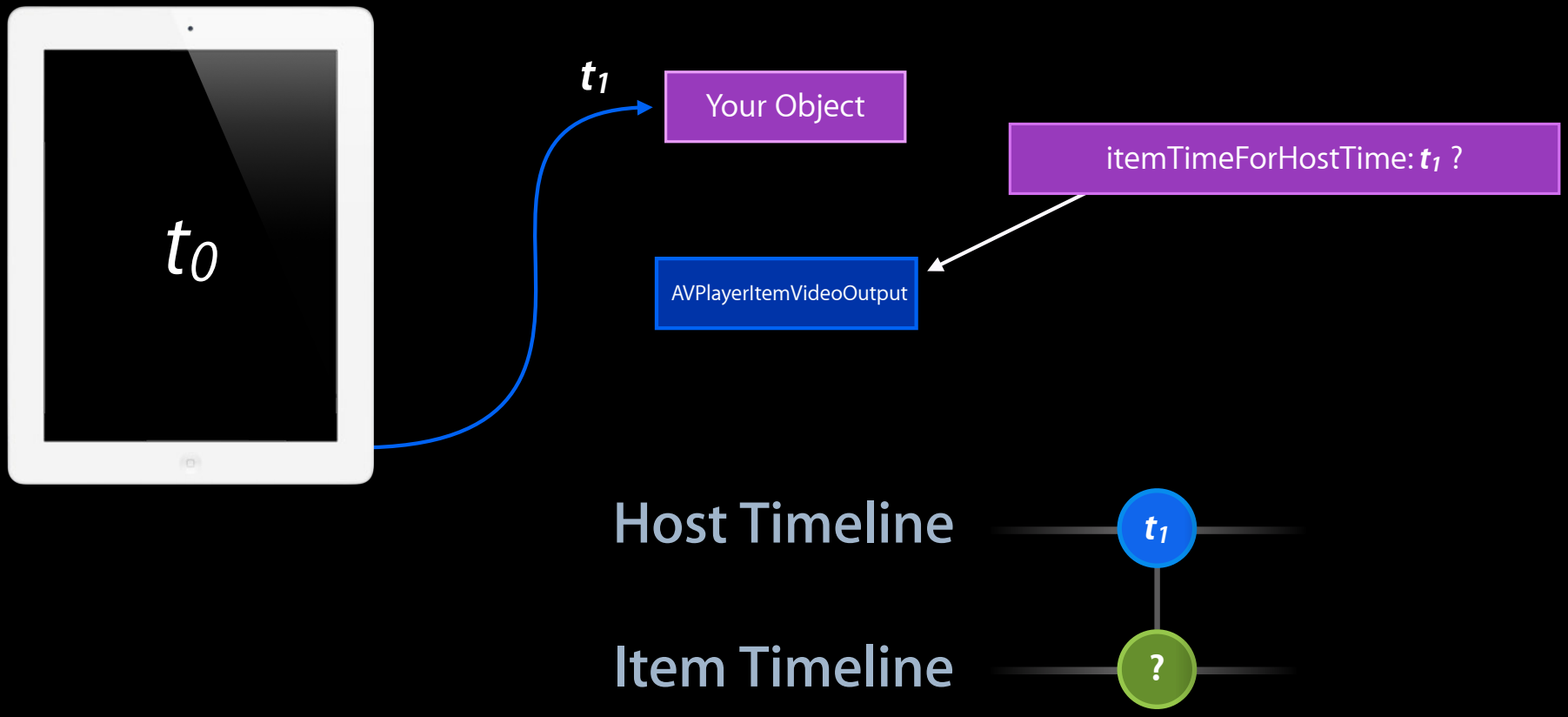
AVPlayerItemVideoOutput

Pulling images from a dynamic model



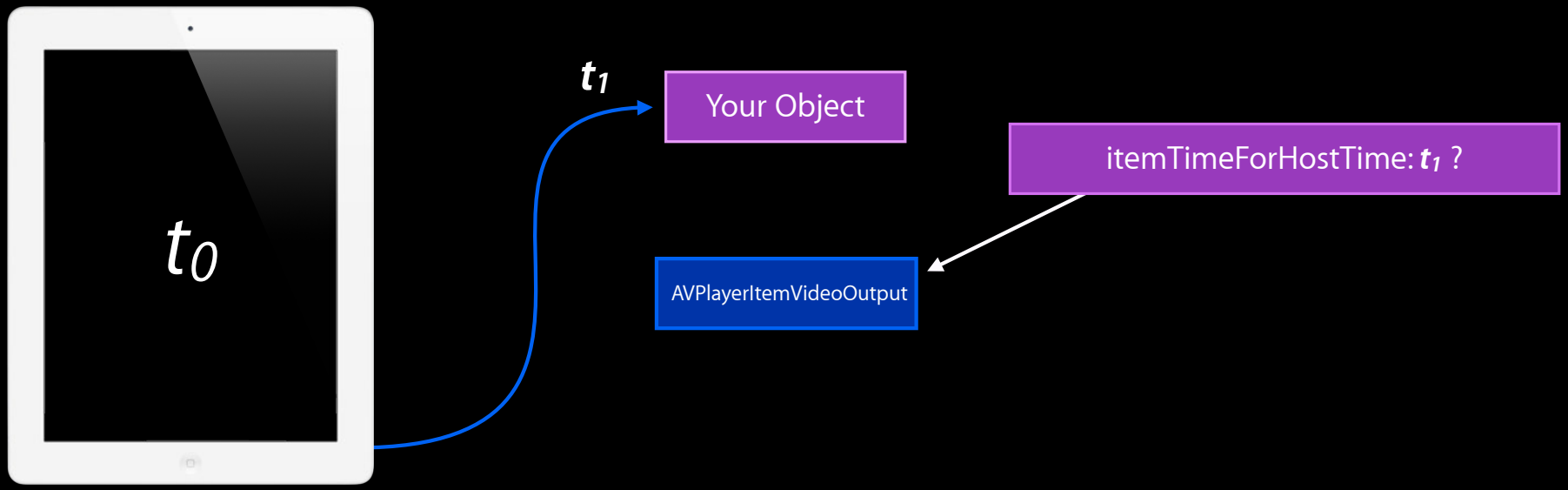
AVPlayerItemVideoOutput

Pulling images from a dynamic model



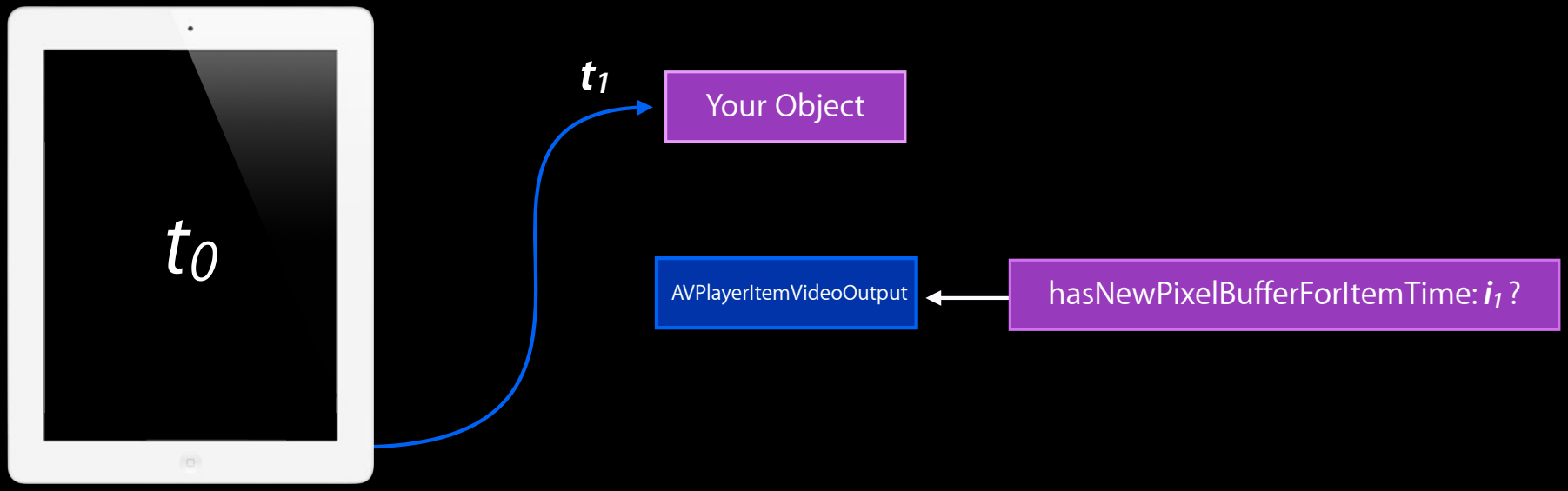
AVPlayerItemVideoOutput

Pulling images from a dynamic model



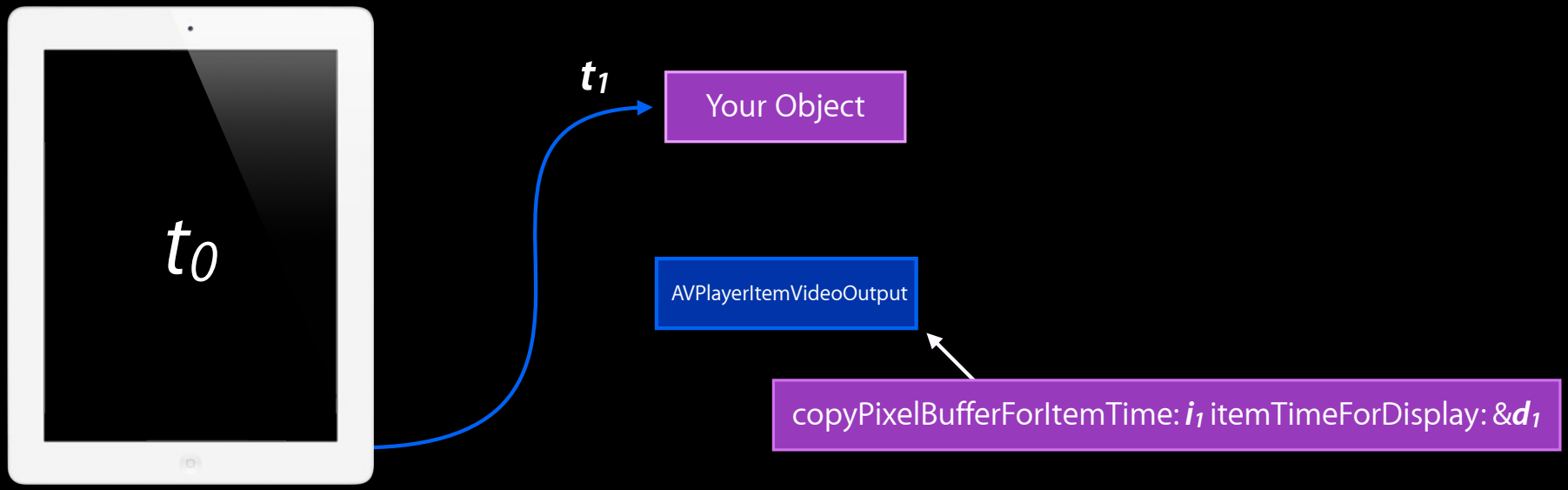
AVPlayerItemVideoOutput

Pulling images from a dynamic model



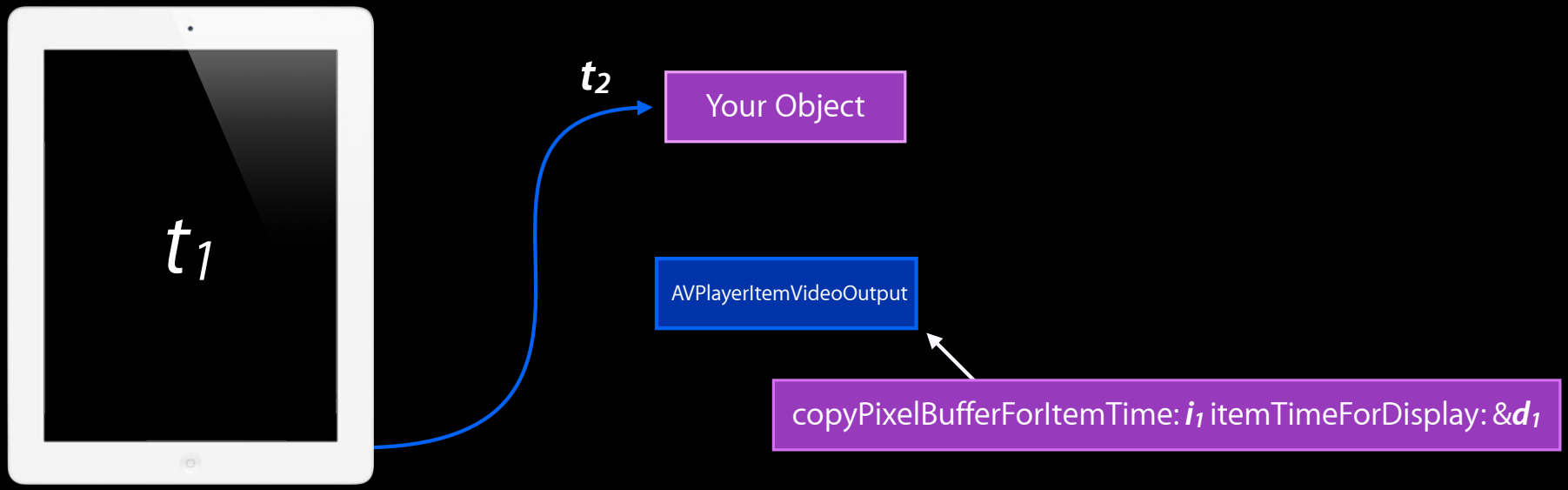
AVPlayerItemVideoOutput

Pulling images from a dynamic model



AVPlayerItemVideoOutput

Pulling images from a dynamic model



CADisplayLink

```
_displayLink = [CADisplayLink displayLinkWithTarget:self
                                           selector:@selector(mySelector:)];

[_displayLink setFrameInterval:1];
[_displayLink setPaused:YES];
[_displayLink addToRunLoop:[NSRunLoop currentRunLoop]
                   forMode:NSDefaultRunLoopMode];
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

CADisplayLink

```
- (void)mySelector:(CADisplayLink *)sender
{
  CFTimeInterval nextOutputTime = [sender timestamp] + [sender duration];
  CMTime itemTime = [_pixelBufferOutput itemTimeForTimeStamp:nextOutputTime];
  if ([_pixelBufferOutput isNewOutputAvailable:itemTime]) {
    CVPixelBufferRef pixBuff = NULL;
    [_pixelBufferOutput copyPixelBuffer:&pixBuff forTime:outputMediaTime];

    glActiveTexture(GL_TEXTURE0);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_lumaTexture );
    glBindTexture(...);

    glActiveTexture(GL_TEXTURE1);
    CVOpenGLESTextureCacheCreateTextureFromImage(..., &_chromaTexture);
    glBindTexture(...);

    CVBufferRelease(pixBuff);
   }
}
```

AVPlayerItemVideoOutput

Meeting the deadline

- You can pull ahead of playhead
- Images not pulled before deadline are discarded

AVPlayerItemVideoOutput

Save power when paused

- Go to sleep
- Request wakeup when output will change
  - `requestNotificationOfMediaDataChangeWithAdvanceInterval:`
  - Delegate callback
    - `outputMediaDataWillChange:`

Video on a Texture

- AVPlayerItemVideoOutput
- OpenGL ES 2.0 game with GLKit
- Real-time chroma key effect
  - Fragment shader
- Video texture atlas

Demo

MagicCube

Summary

Summary

- A/V Sync is the key goal
  - Tell AVFoundation which clock to sync to
  - Be efficient with the audio processing callback
  - Use system services to remain in sync when using an AVPlayerItemVideoOutput

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

Sample Code
Available on session website

Related Sessions

What's New in Camera Capture	Presidio Thursday 10:15AM
Multiplayer Gaming with Game Center	Pacific Heights Thursday 10:15AM
Building Game Center Games for OS X	Pacific Heights Thursday 11:30AM

Labs

Core Animation Lab	Graphics Media & Games Lab C Thursday 11:30AM
AV Foundation Lab	Graphics Media & Games Lab C Thursday 2:00PM
Game Design Lab	Graphics Media & Games Lab A Thursday 2:00PM
iOS Camera Capture Lab	Graphics Media & Games Lab D Thursday 2:00PM


WWDC2012