Music in iOS and Mac OS X

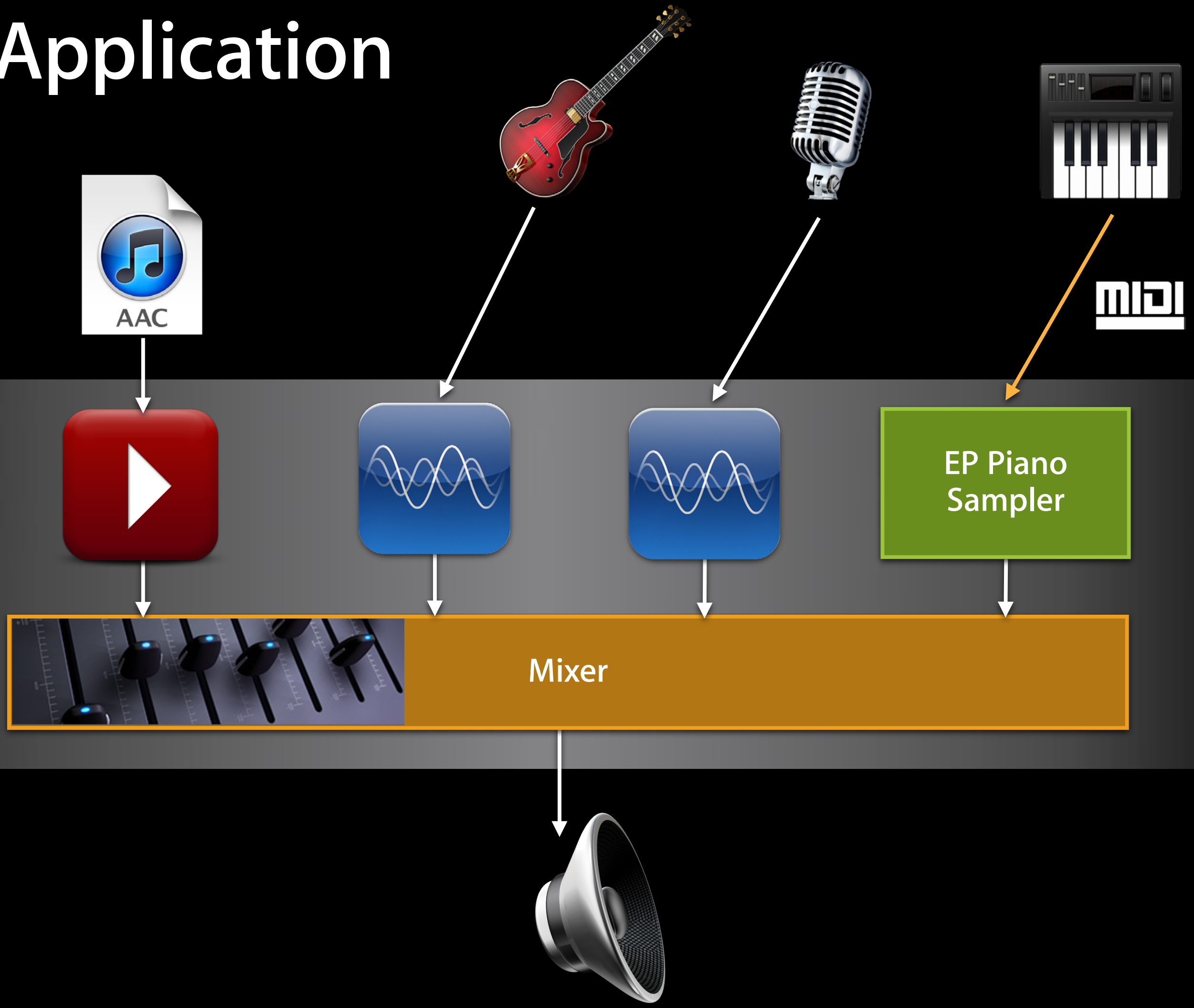
Session 411

Michael Hopkins
Core Audio Engineering

What You Will Learn

- Using Audio Units
- Introduction to the AUSampler
- CoreMIDI on iOS
- Playing of music sequences

Music Application



What Is an Audio Unit?

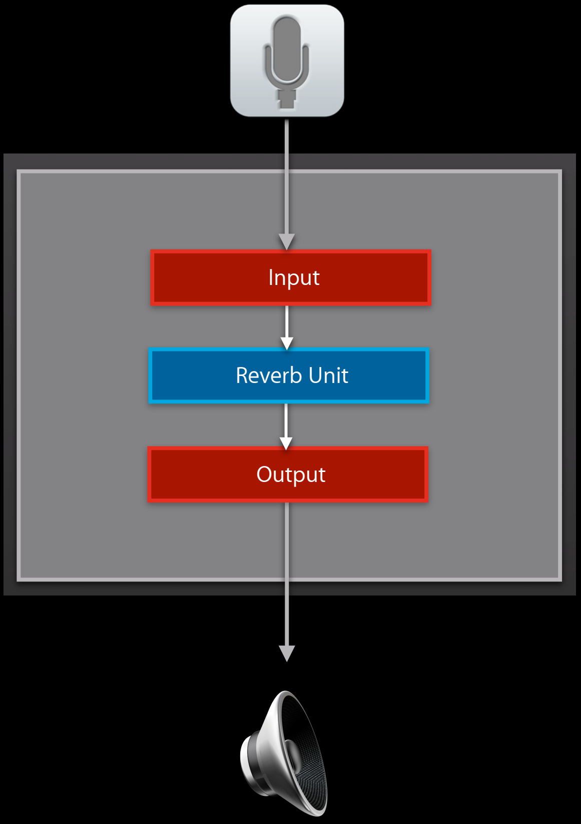
- Plug-in for processing audio
- Supports real-time input, output, or simultaneous I/O
- Organized in an audio processing graph
- Controlled by properties and parameters
- Can have a view (Mac OS X)

Types of Audio Units

- Effects
- Music effects
- Instruments
- Generators
- Panners
- Converters
- Mixers
- Offline effects
- Output units

Using Audio Units

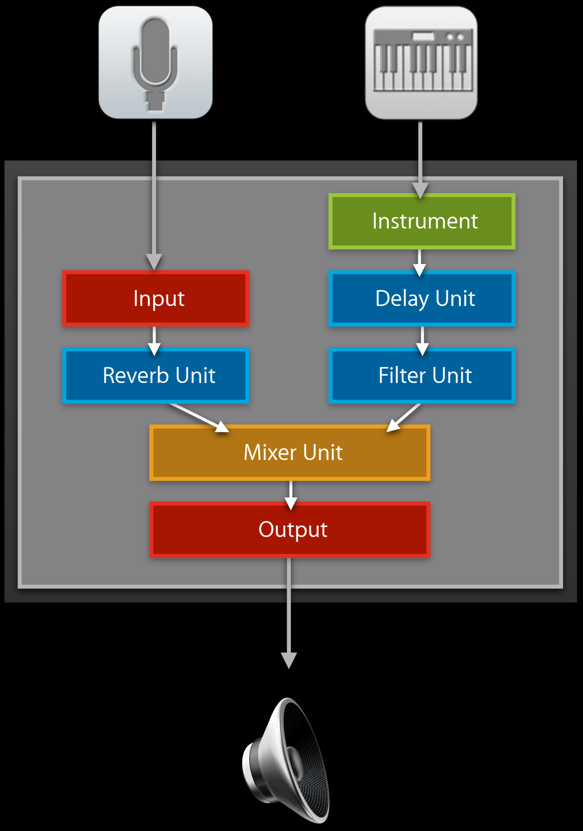
Organizing Audio Units in a graph



- Audio Units can be added to an audio processing network called an AUGraph
  - Each item in the graph is an AUNode
  - Graph defines connections between nodes
  - Signal flow goes from input to output
  - Graph ends with a single output unit

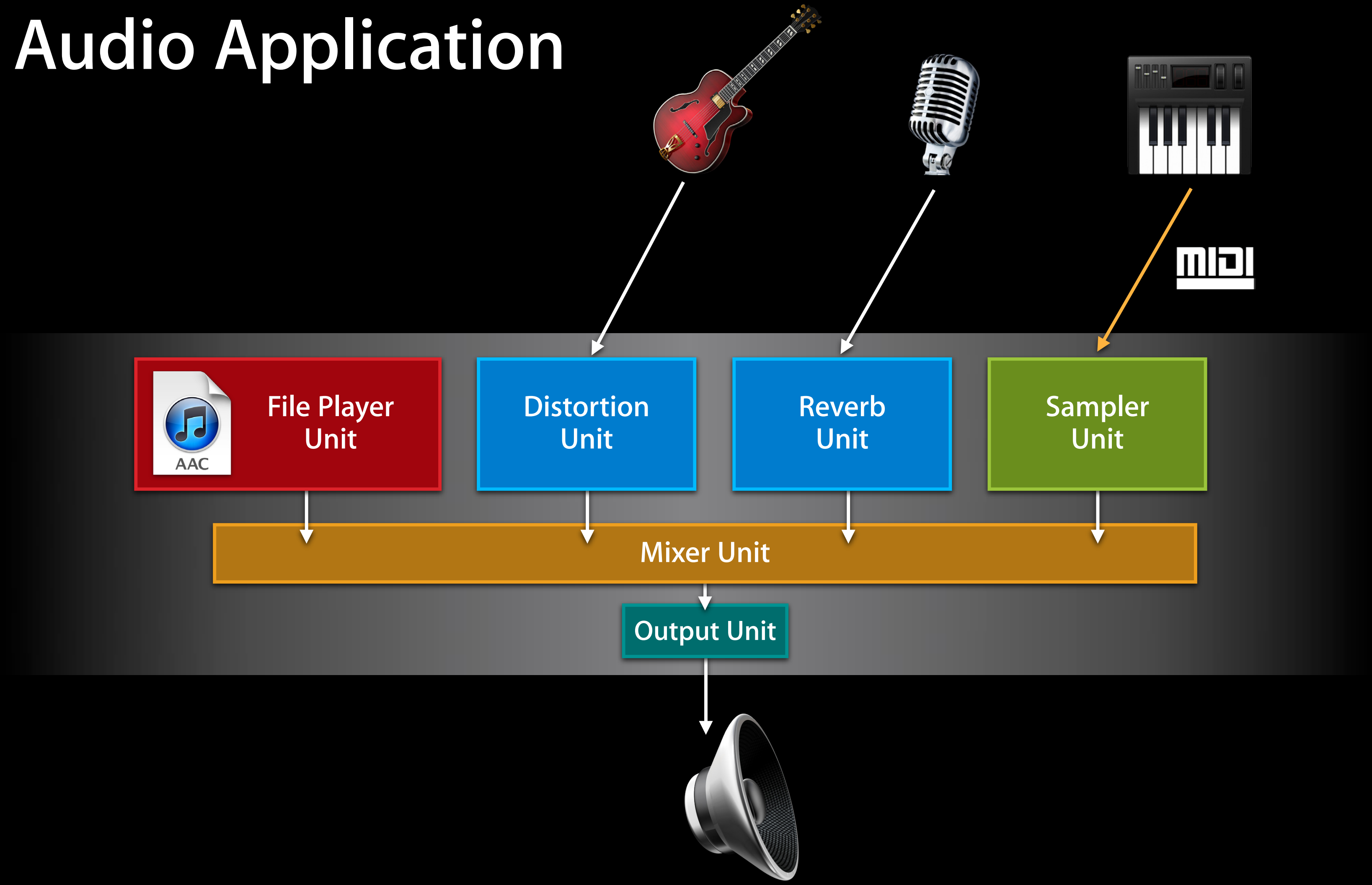
Using Audio Units

AU graphs (cont.)



- Graphs can use a mixer unit to combine separate chains prior to the output

Audio Application



Interacting with Audio Units

Properties

- Key-value pairs
- Configure state that is managed by the host
  - Sample rate
  - Stream format
  - Number of input buses on a mixer
- Set on Audio Unit and take effect at initialization

Interacting with Audio Units

Parameters

- Key-value pairs
- Intended to be used during processing
    - Delay time
    - Feedback amount
    - Stereo panning position
- Generally set through a UI

Demo

Exploring Audio Units in AULab

Michael Hopkins
Core Audio Engineering

New Audio Units in iOS

- Effects
  - Filters
    - Highpass
    - Lowpass
    - Bandpass
    - Highshelf
    - Lowshelf
    - Parametric EQ
  - Peak limiter
  - Dynamics processor
  - Reverb

New Audio Units in iOS

- Effects
  - Varispeed
  - SimpleTime
  - NotQuiteSoSimpleTime
- Generators
  - AudioFilePlayer
  - ScheduledSlicePlayer
- Instruments
  - Sampler

Audio Components

Finding, loading, and instantiating Audio Units

Michael Hopkins
Core Audio Engineering

Audio Unit Basics

- Audio Unit instances are created from Audio Components
- Uniquely identified by an AudioComponentDescription
    - Each value is a four-character OSType

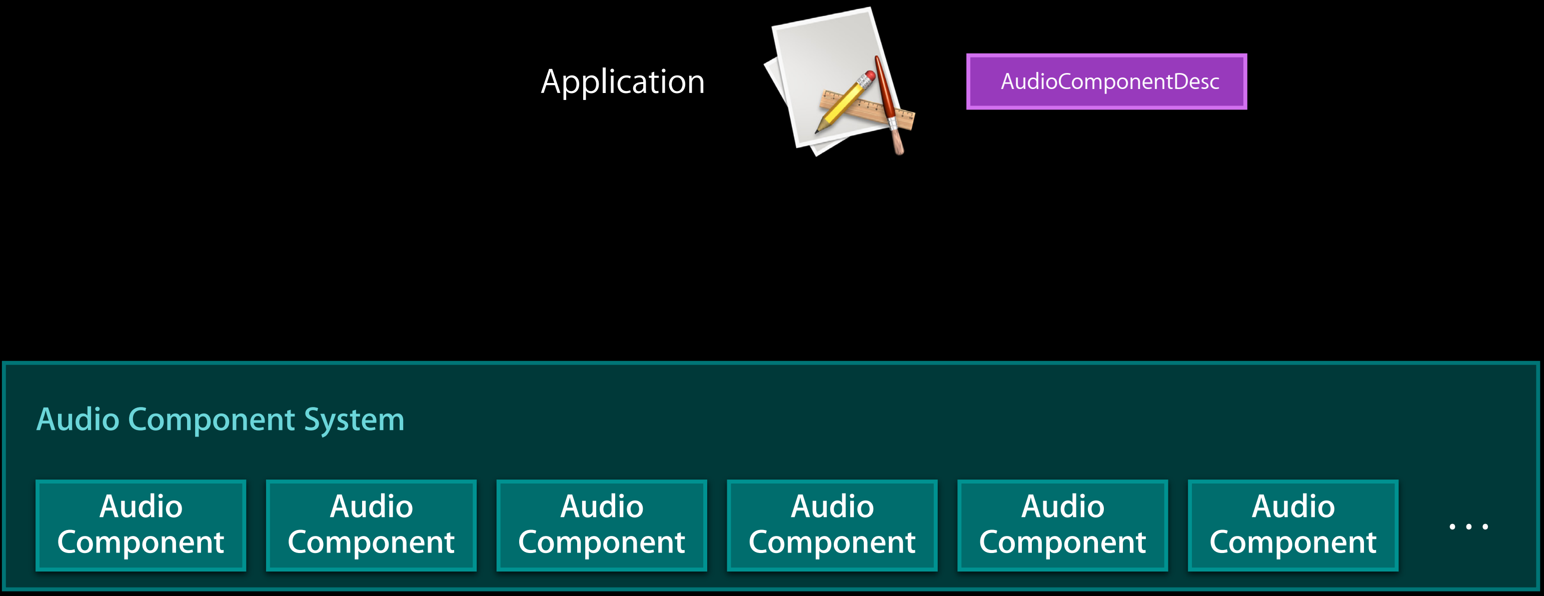
Type	`‘aufx’`
Subtype	`‘dlay’`
Manufacturer	`‘acme’`

- Name
- Version

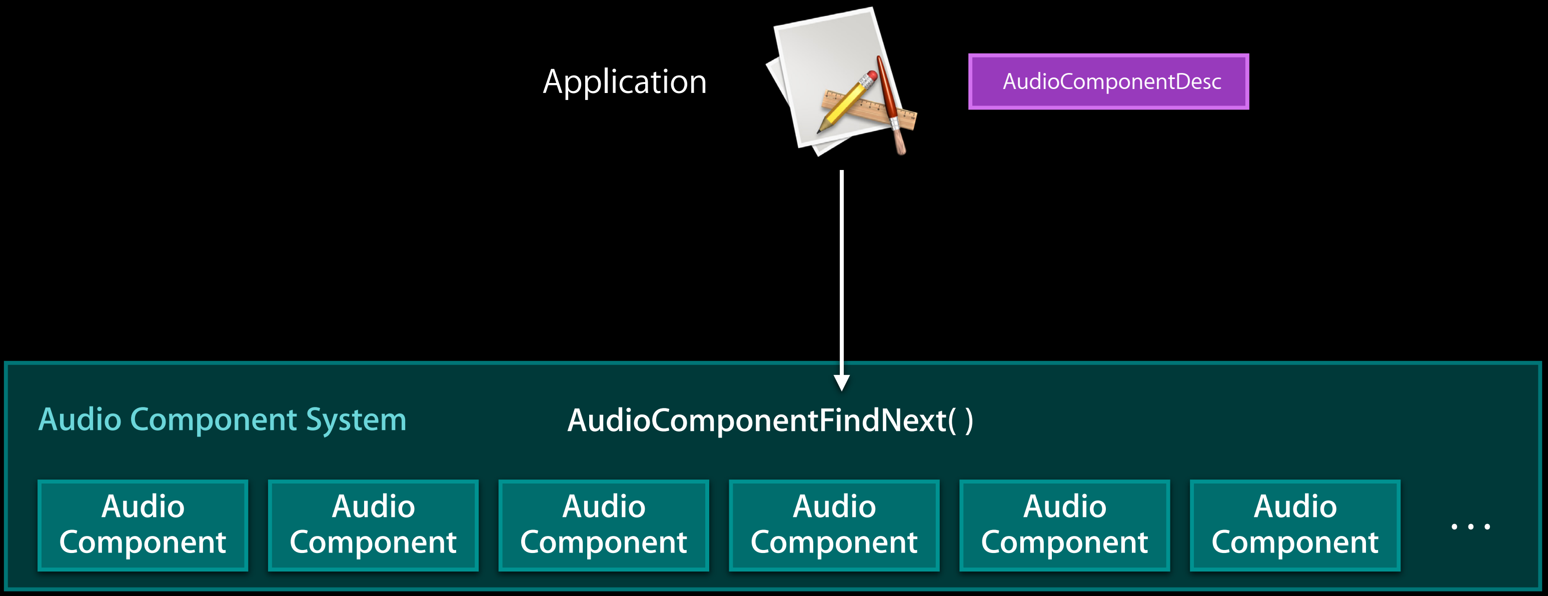
Introducing the AudioComponent API

- Provides facilities for Audio Components
  - Finding
  - Loading
  - State management
- Available in iOS and Mac OS X
- Replaces Component Manager calls on Mac OS X

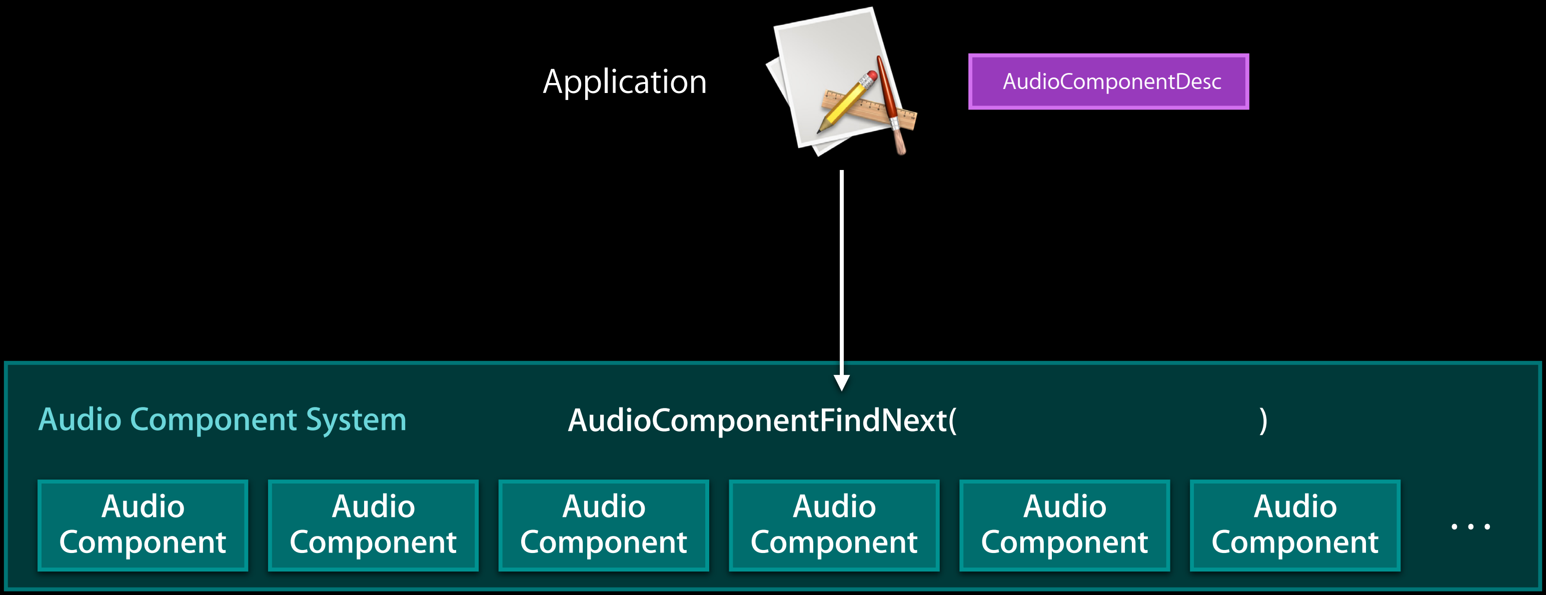
Finding and Loading an Audio Unit



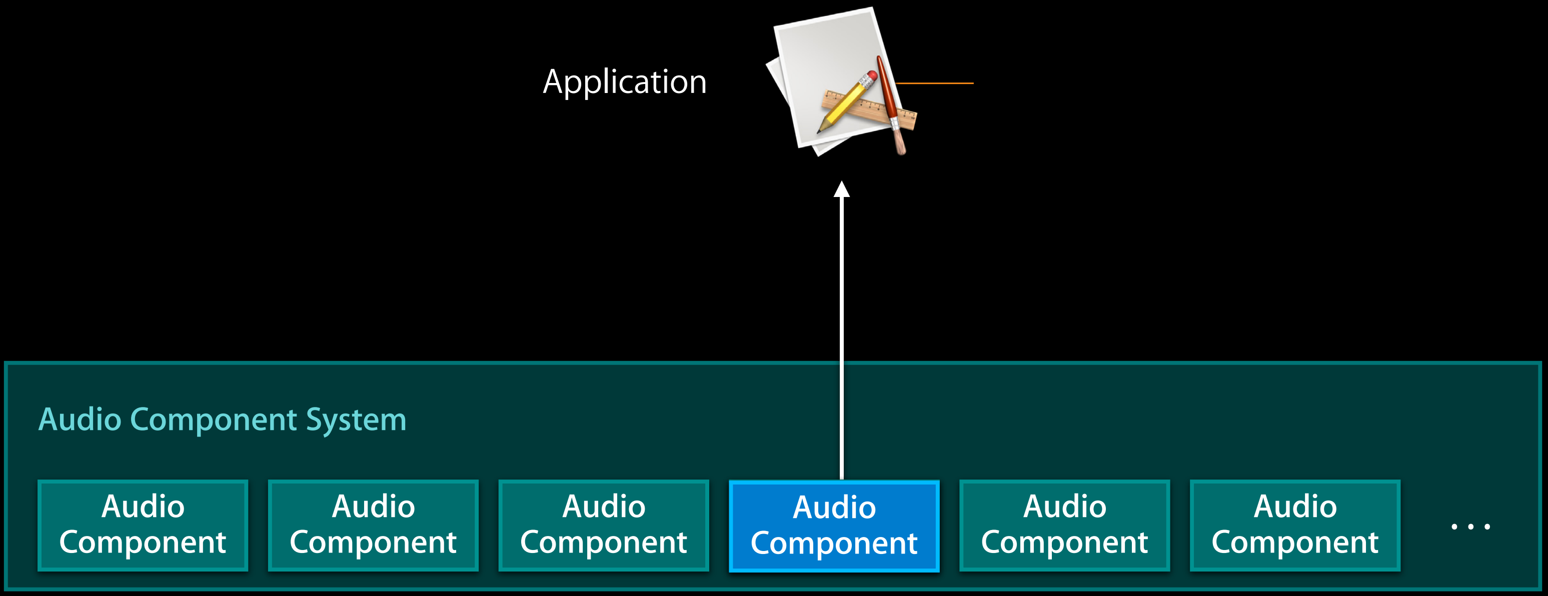
Finding and Loading an Audio Unit



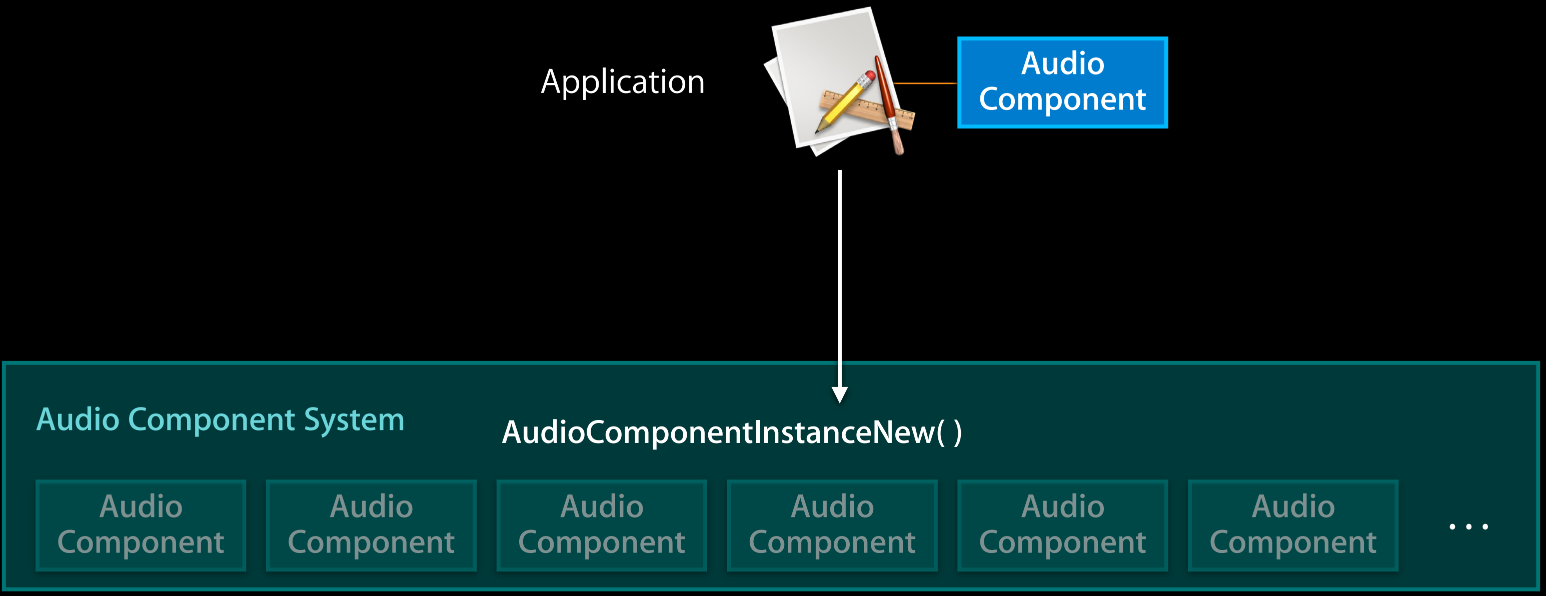
Finding and Loading an Audio Unit



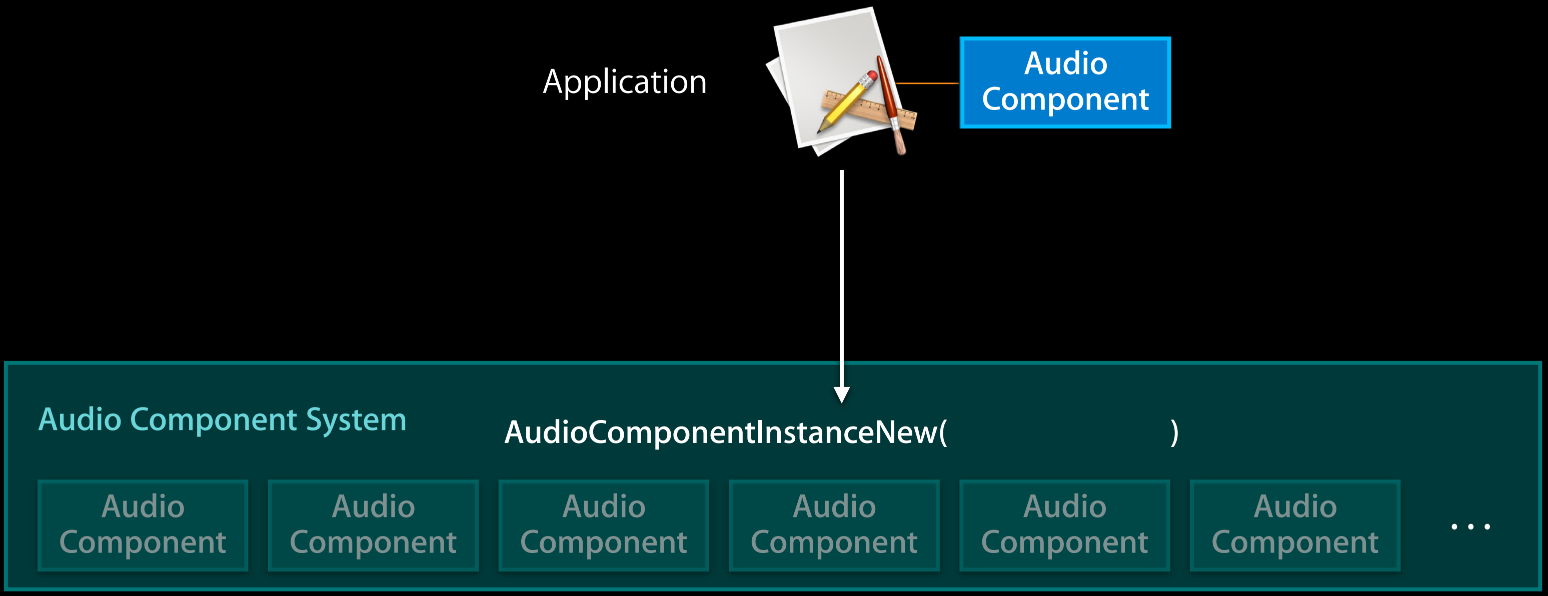
Finding and Loading an Audio Unit

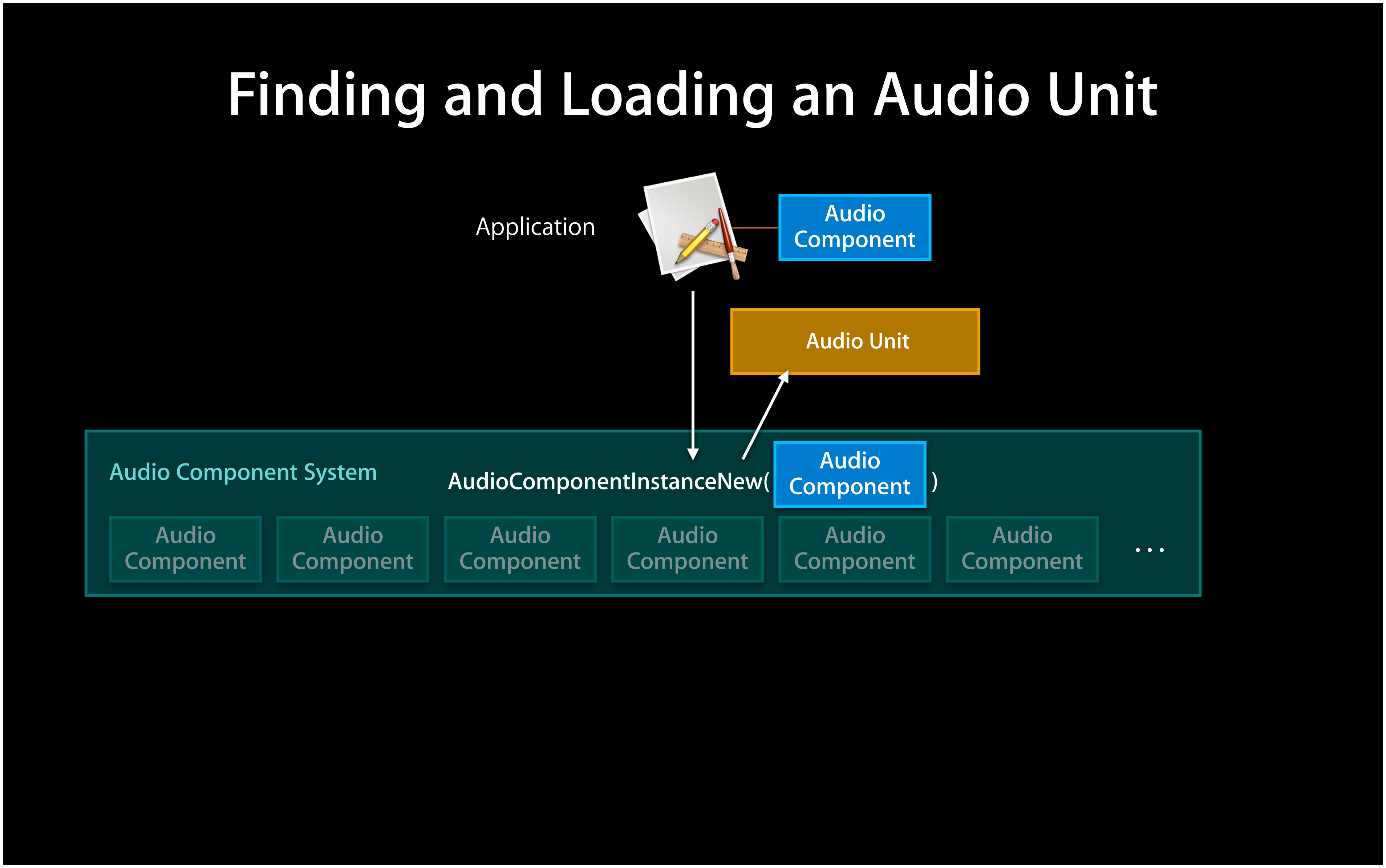


Finding and Loading an Audio Unit



Finding and Loading an Audio Unit





Finding and Loading an Audio Unit

Finding and Creating an Audio Unit

- Use AudioComponentFindNext() to find a specific Audio Component

```
AudioComponent AudioComponentFindNext(AudioComponent *inComp,
                                      const AudioComponentDescription *inDesc)
```

  - First argument can be NULL to start at beginning of component list
  - Pass a specific Audio Component to retrieve next match
  - Wildcard searches allowed

Registering Audio Components

- System scans directories for bundles

```
~/Library/Audio/Plug-Ins/Components
/Library/Audio/Plug-Ins/Components
/System/Library/Components
```

- Bundles have a specific extension
  - .audiocomp (registers with Audio Component System)
  - .component (registers with Component Manager and Audio Component System)

Registering Audio Components

Components can be registered at runtime

```
AudioComponent AudioComponentRegister(AudioComponentDescription *desc,
                                      CFStringRef name,
                                      UInt32 version,
                                      AudioComponentFactoryFunction func)
```

- Component only available in application process

Audio Components

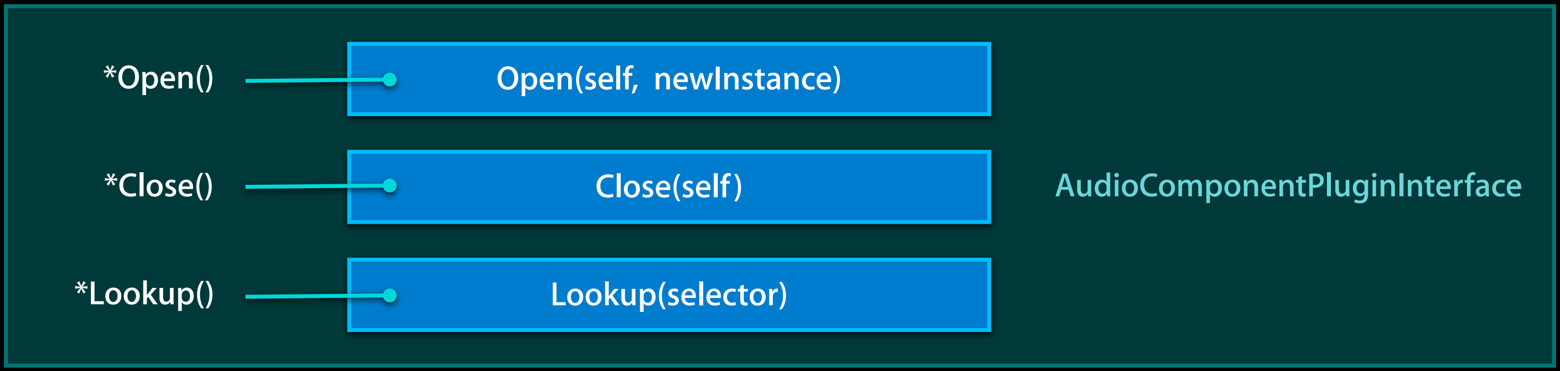
A look under the hood

- All Audio Components have an AudioComponentFactoryFunction
  - Used to create instances of the Component
  - Returns a pointer to an AudioComponentPluginInterface

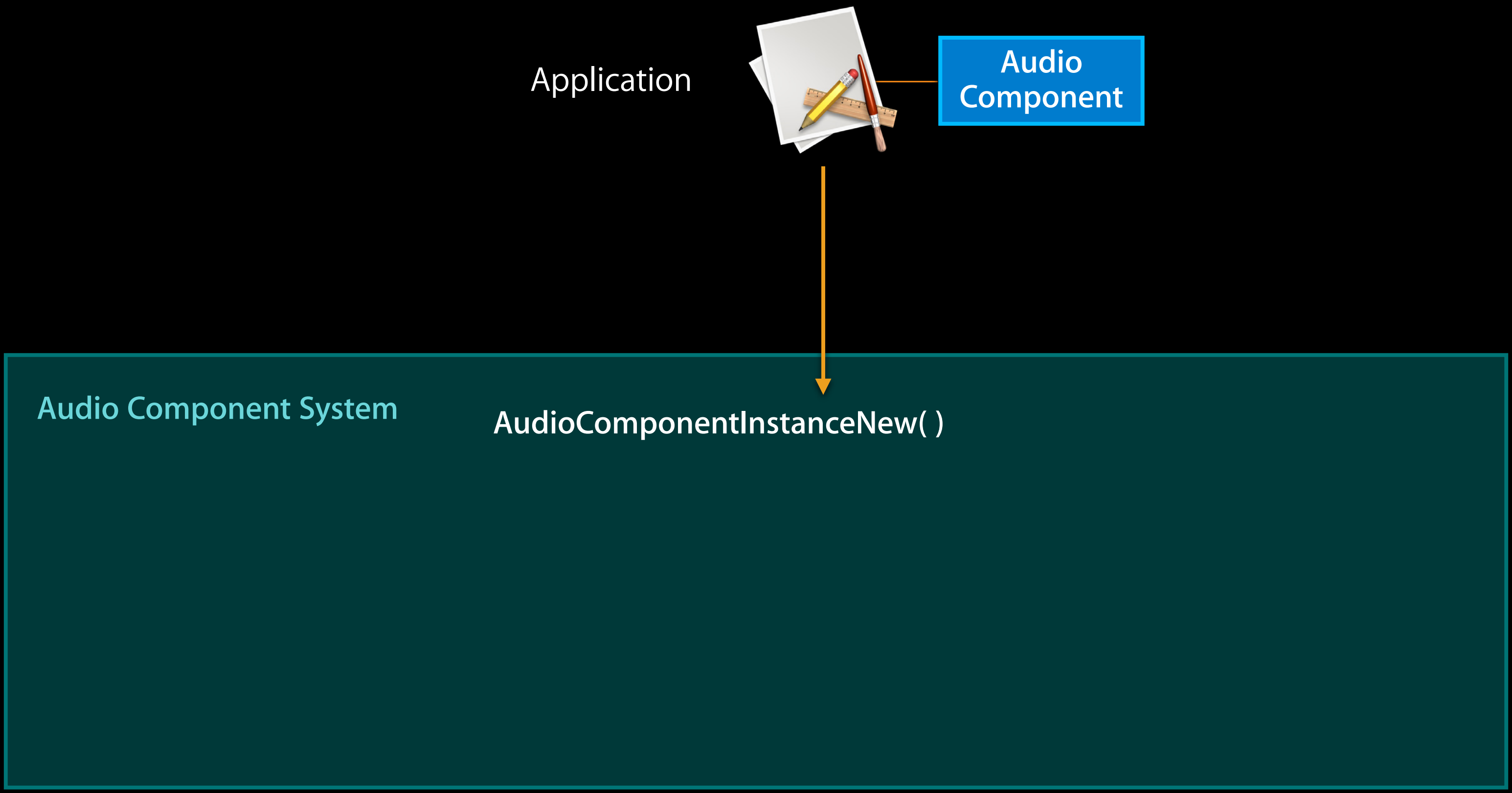
Audio Units

A look under the hood

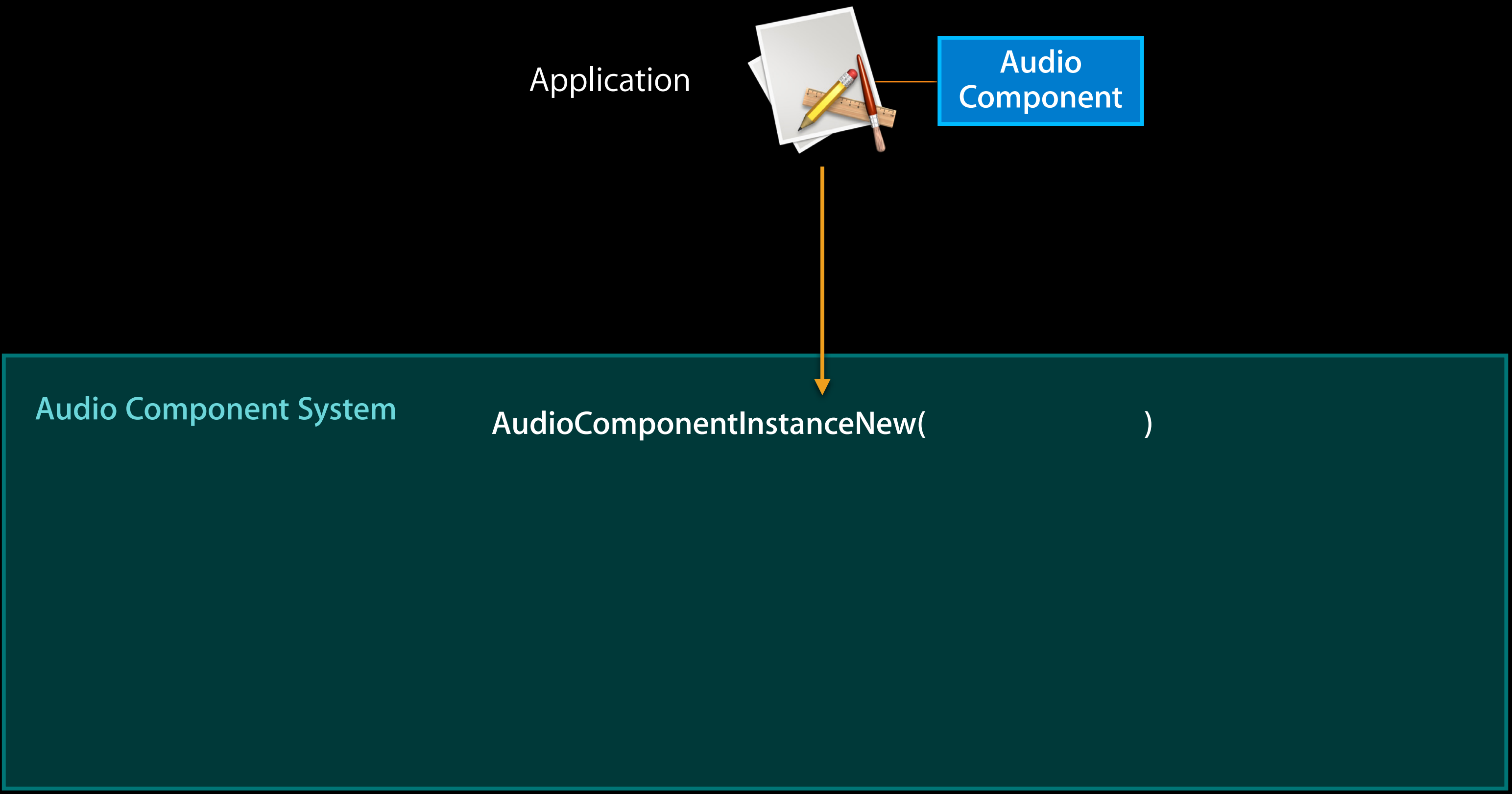
- All Audio Components have an AudioComponentFactoryFunction
  - Used to create instances of the Component
  - Returns a pointer to an AudioComponentPluginInterface



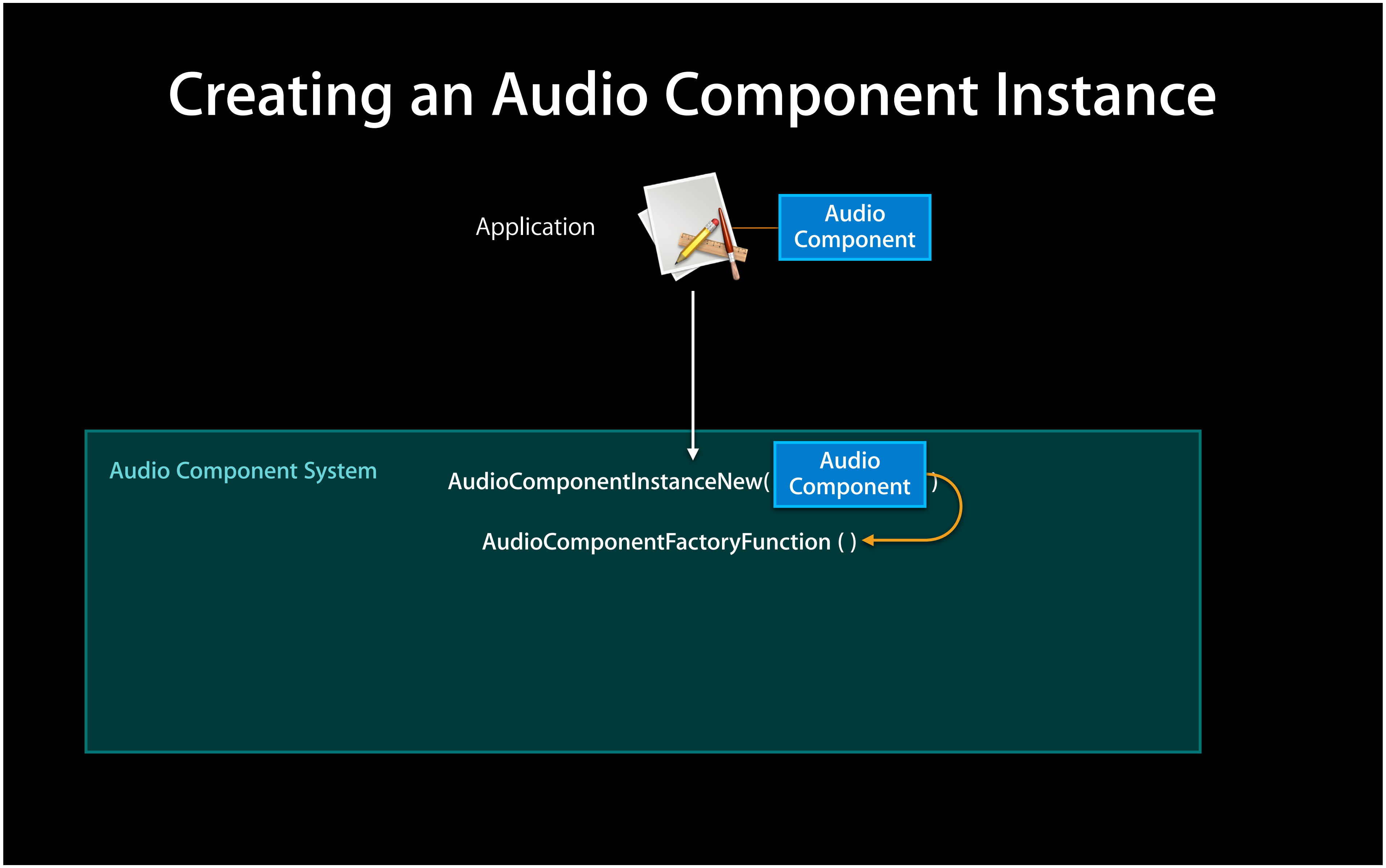
Creating an Audio Component Instance



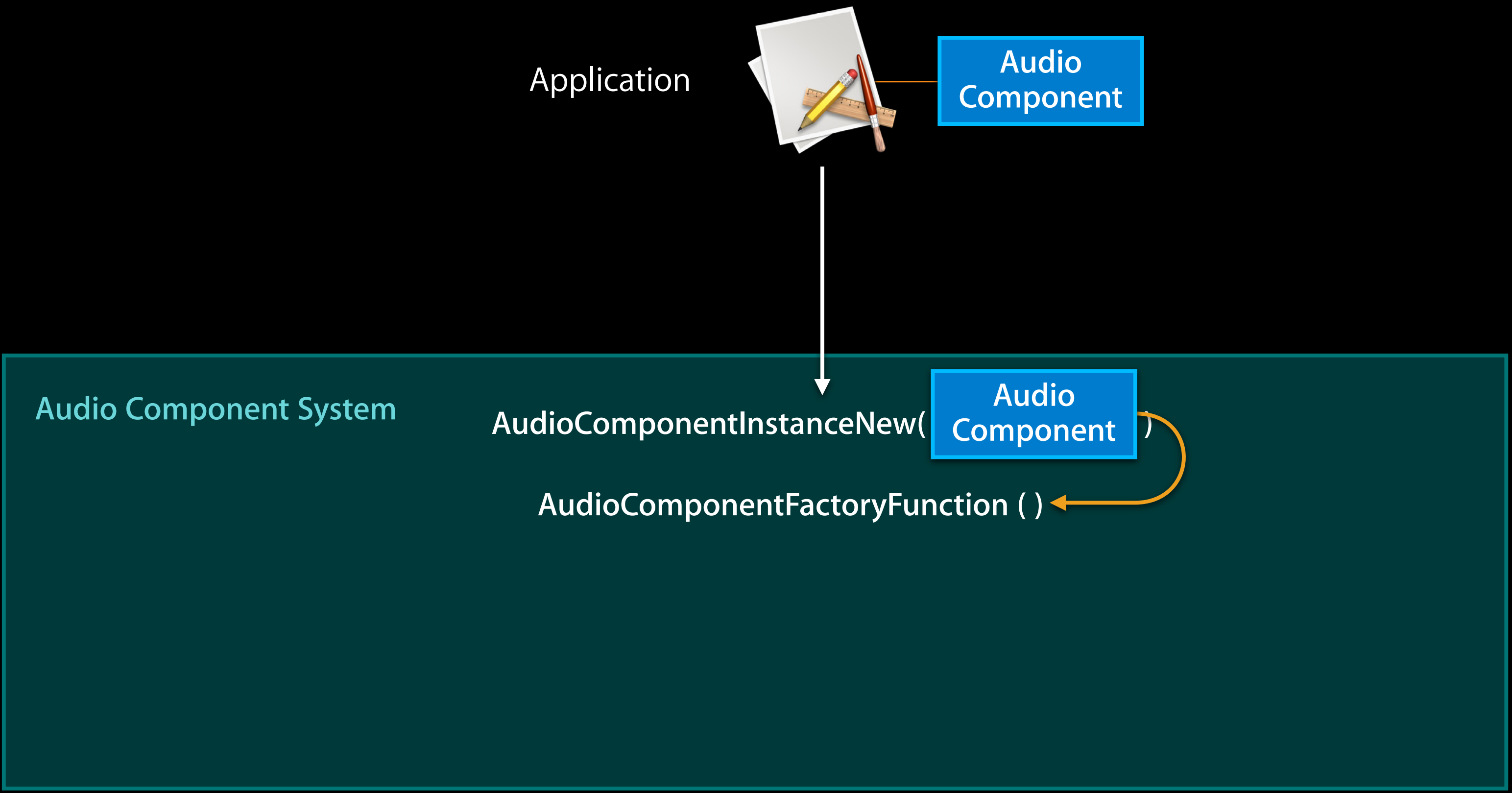
Creating an Audio Component Instance



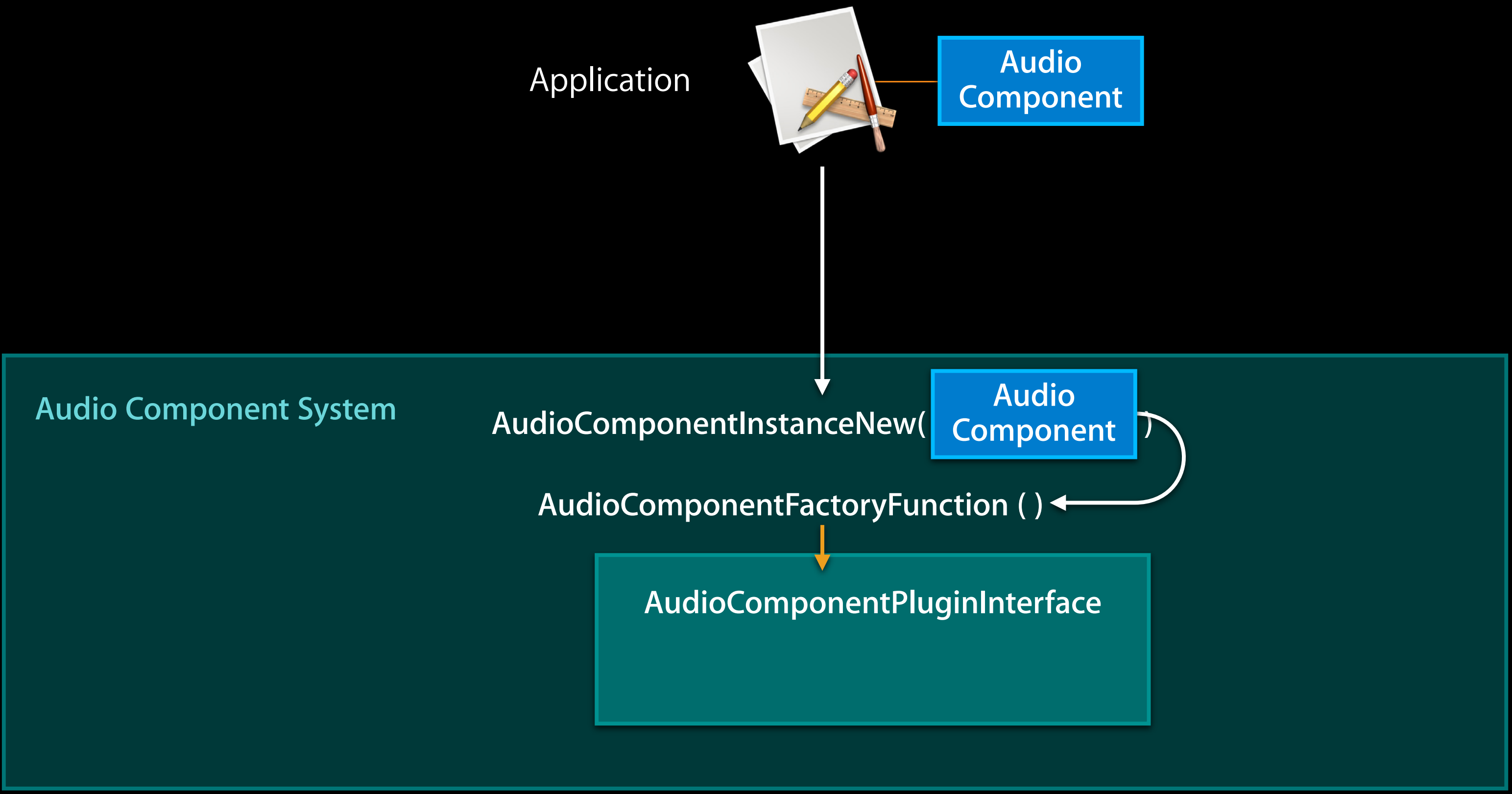
AudioComponentInstanceNew( )



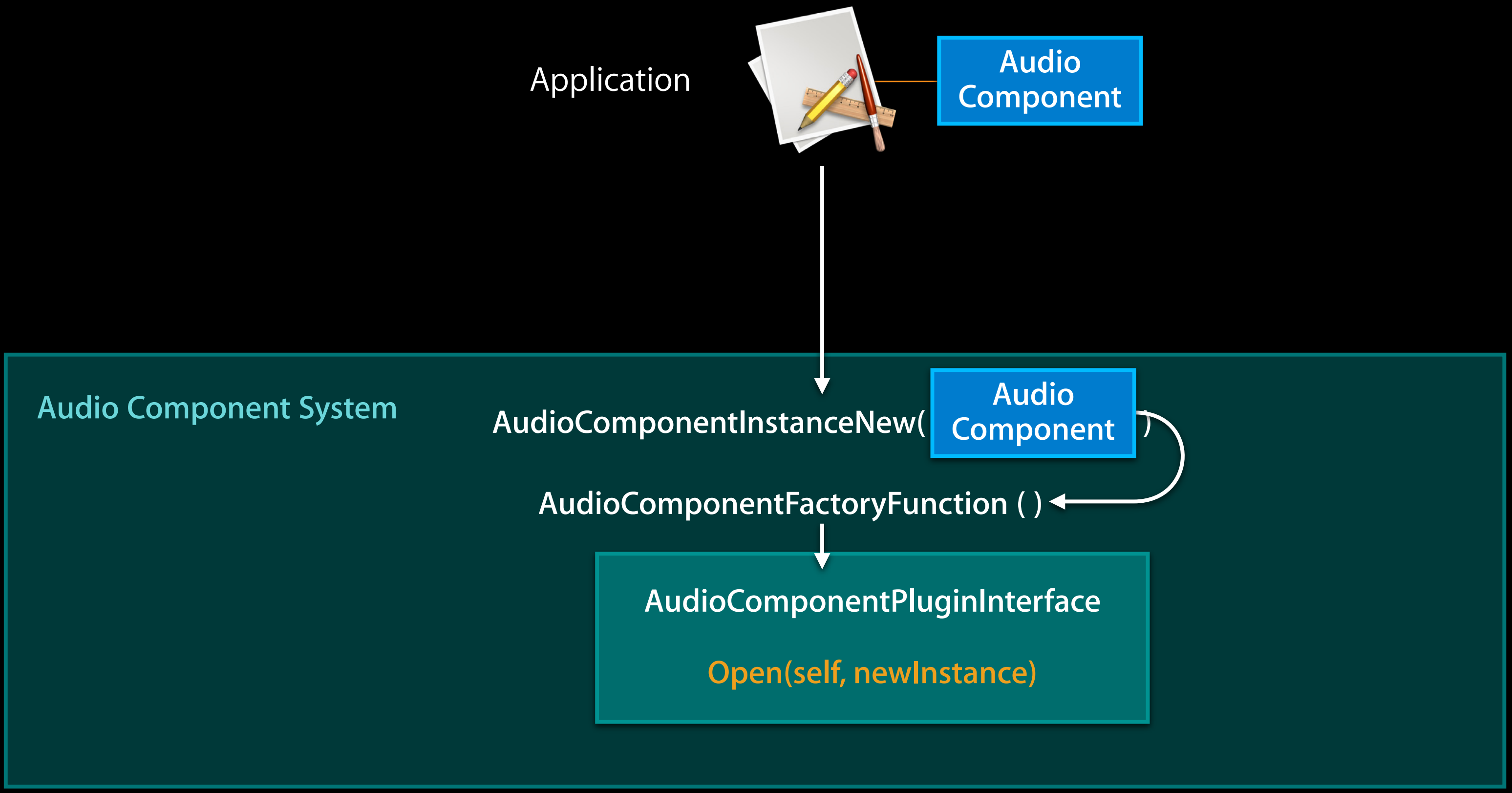
Creating an Audio Component Instance


Application
Audio Component
Audio Component System
AudioComponentInstanceNew(Audio Component)
AudioComponentFactoryFunction ( )

Creating an Audio Component Instance

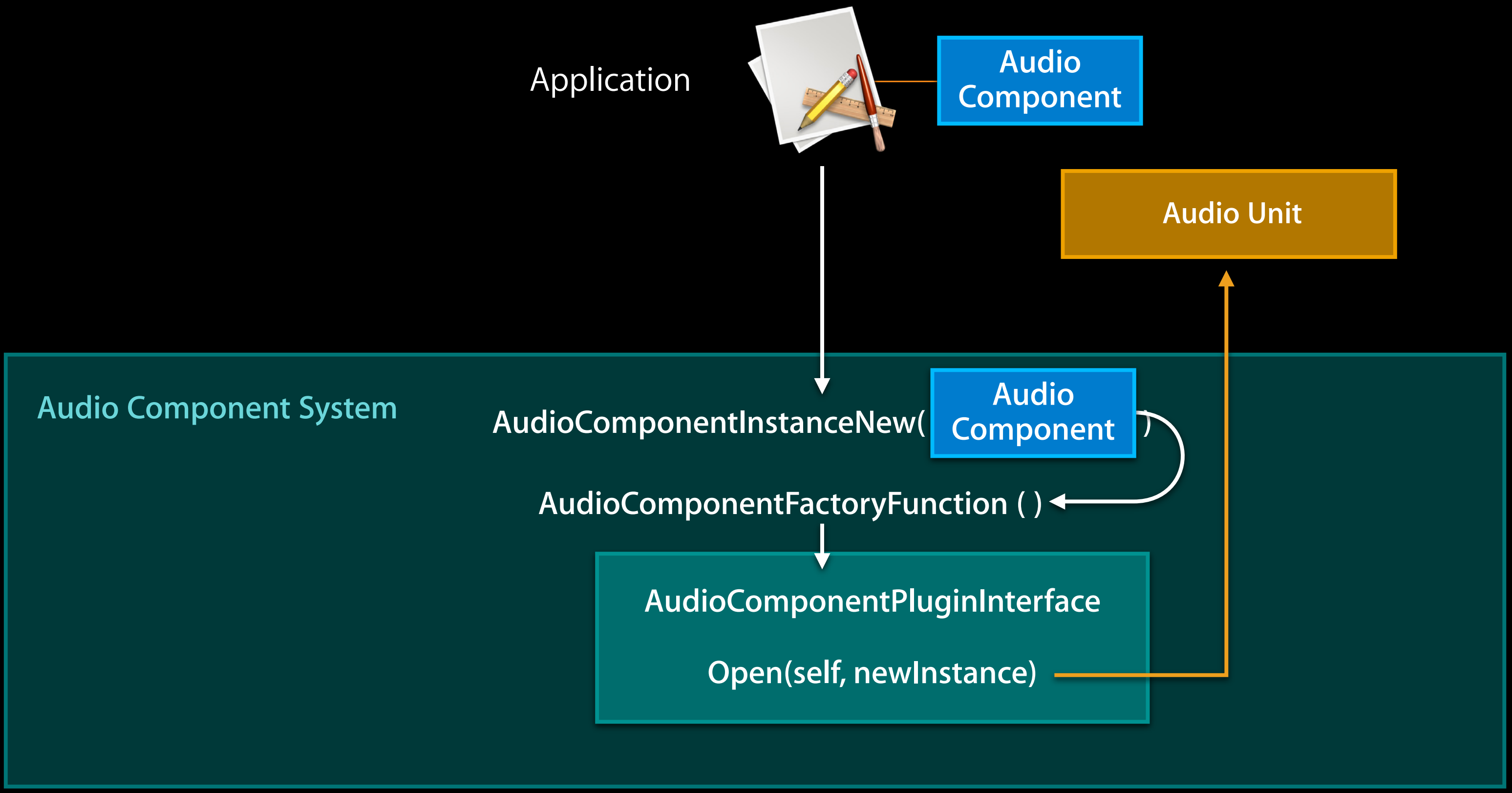

Application
Audio Component
Audio Component System
AudioComponentInstanceNew(Audio Component)
AudioComponentFactoryFunction ( )
AudioComponentPluginInterface

Creating an Audio Component Instance

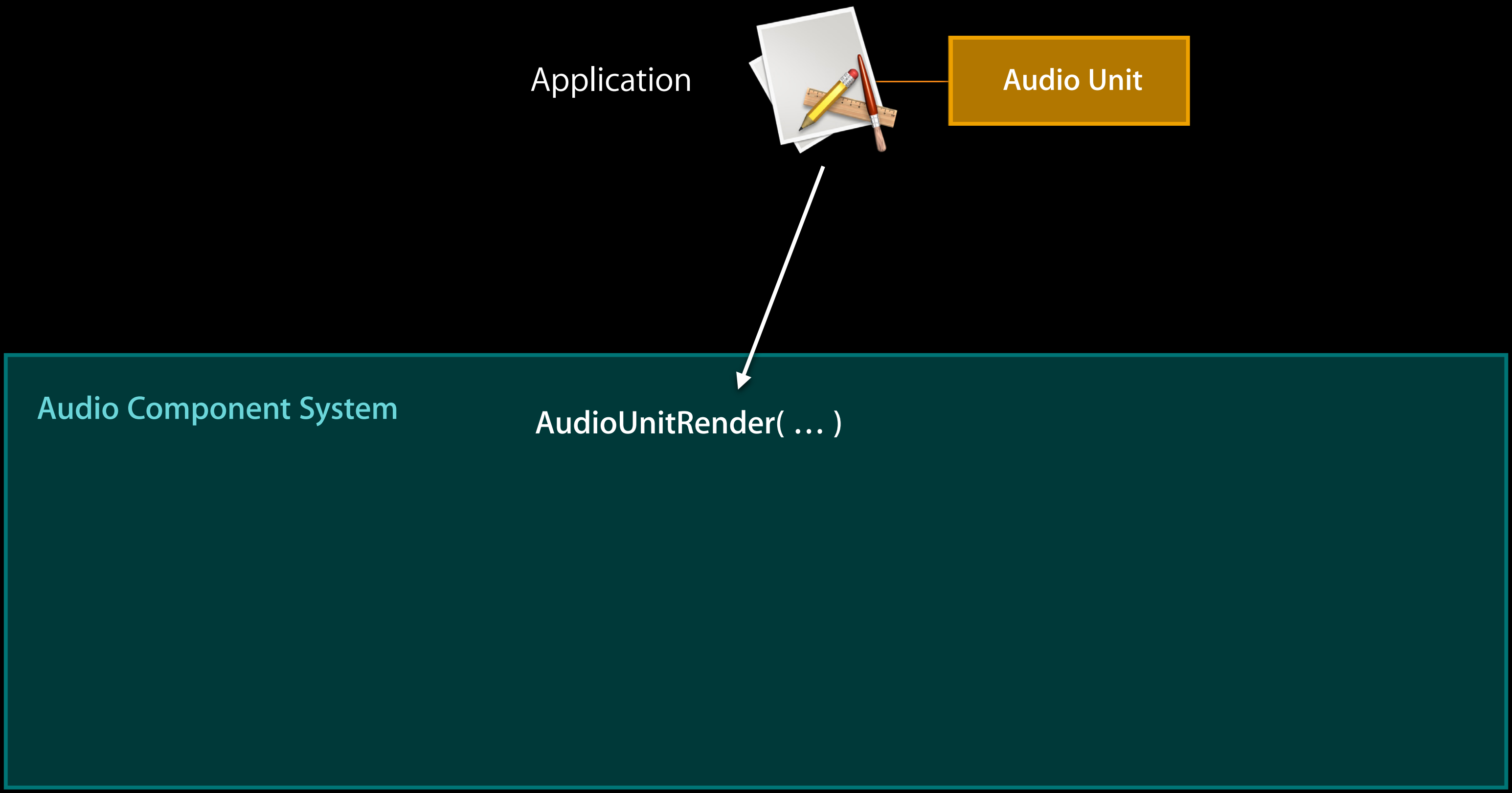


Application
Audio Component
Audio Component System
AudioComponentInstanceNew(Audio Component)
AudioComponentFactoryFunction ( )
AudioComponentPluginInterface
Open(self, newInstance)

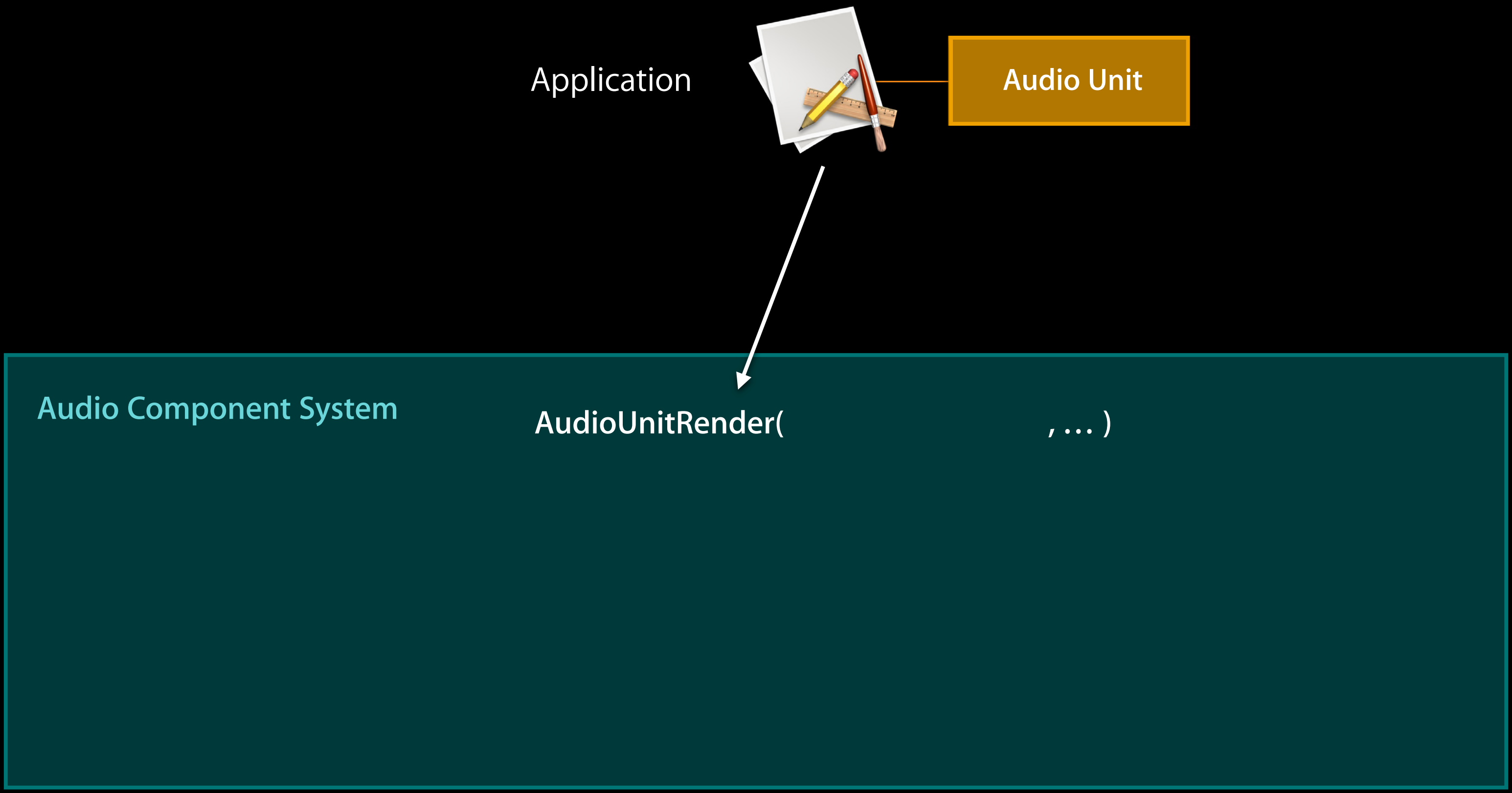
Creating an Audio Component Instance

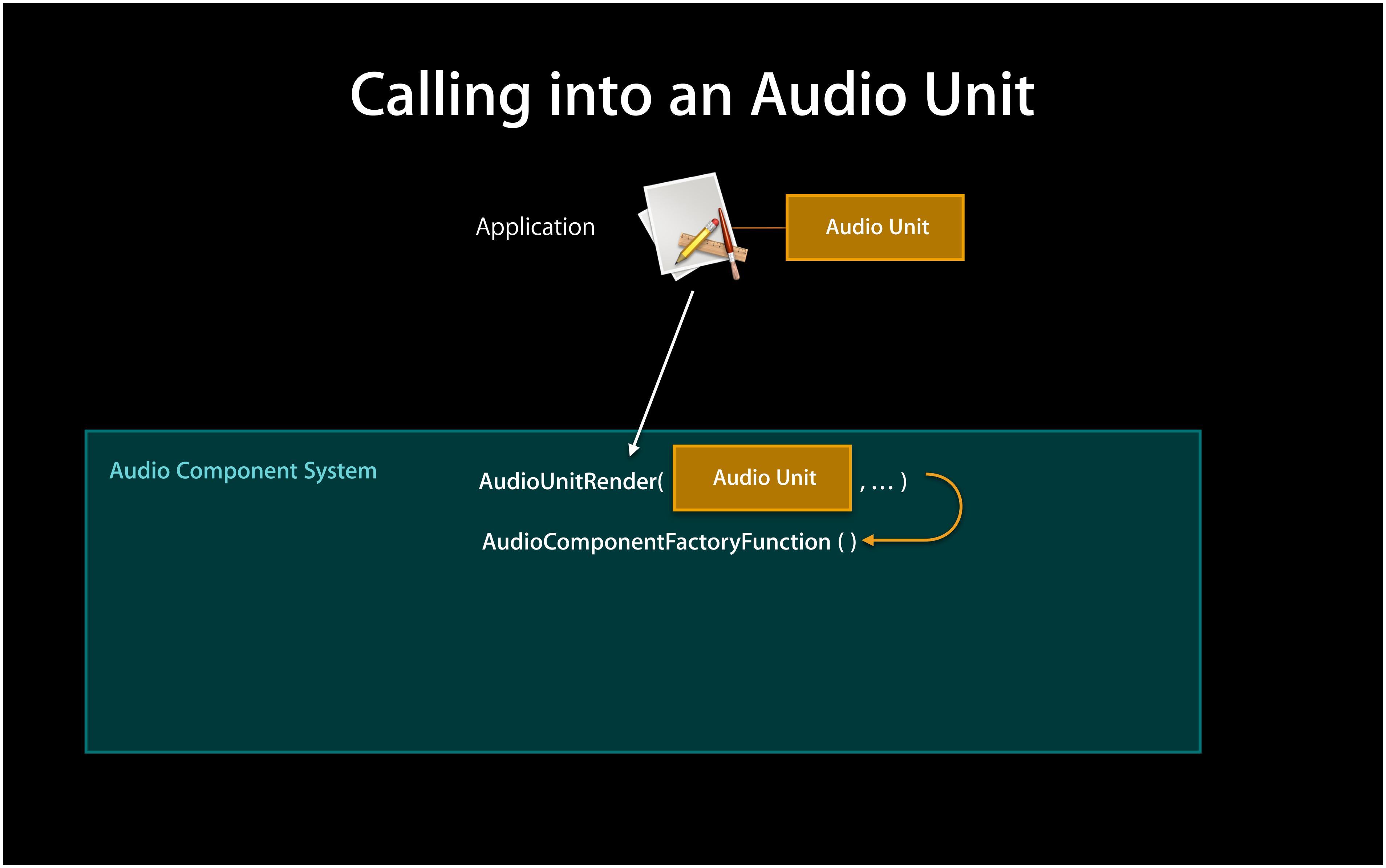

Application
Audio Component
Audio Unit
Audio Component System
AudioComponentInstanceNew(Audio Component)
AudioComponentFactoryFunction ( )
AudioComponentPluginInterface
Open(self, newInstance)

Calling into an Audio Unit



Calling into an Audio Unit

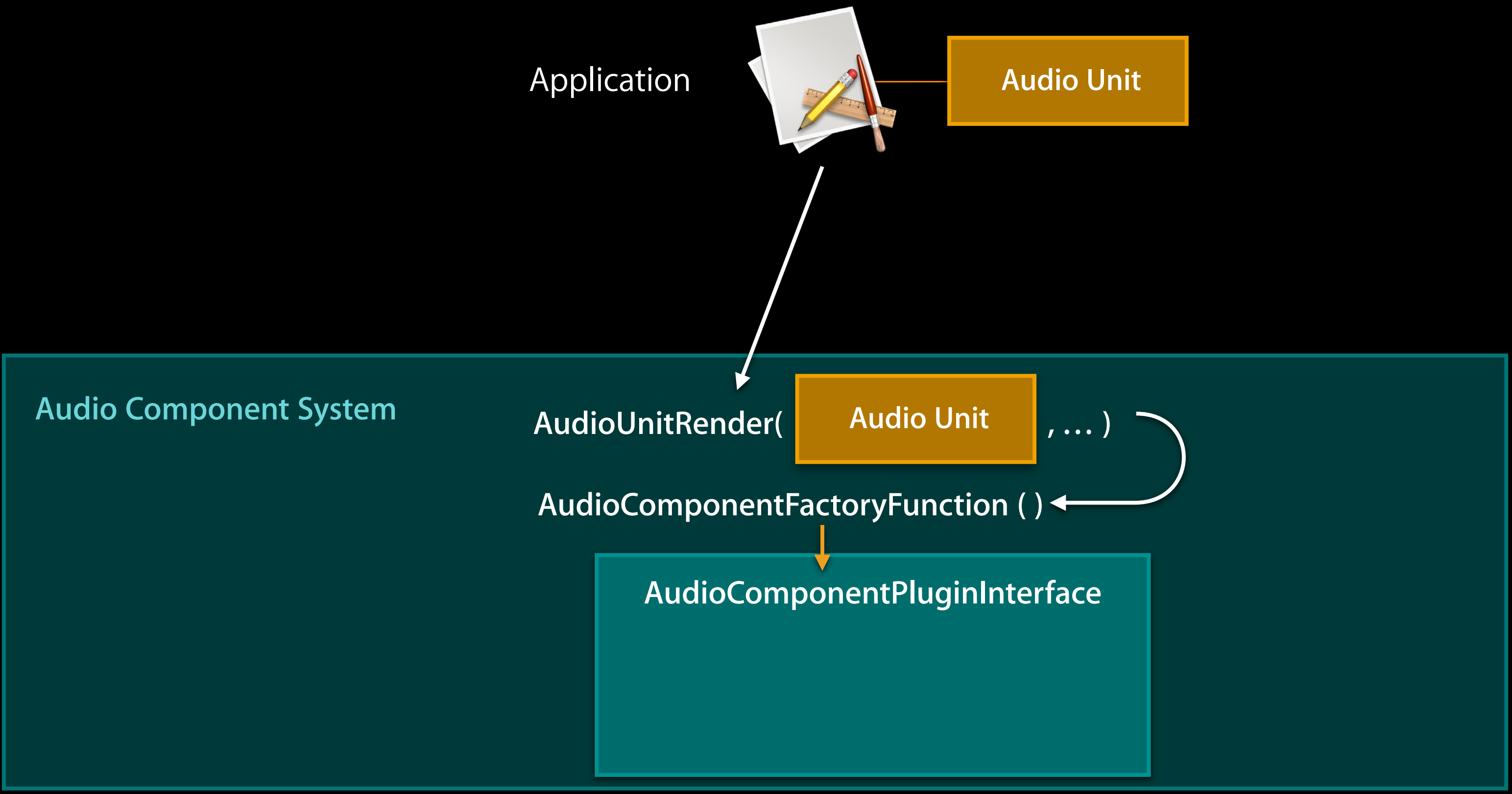




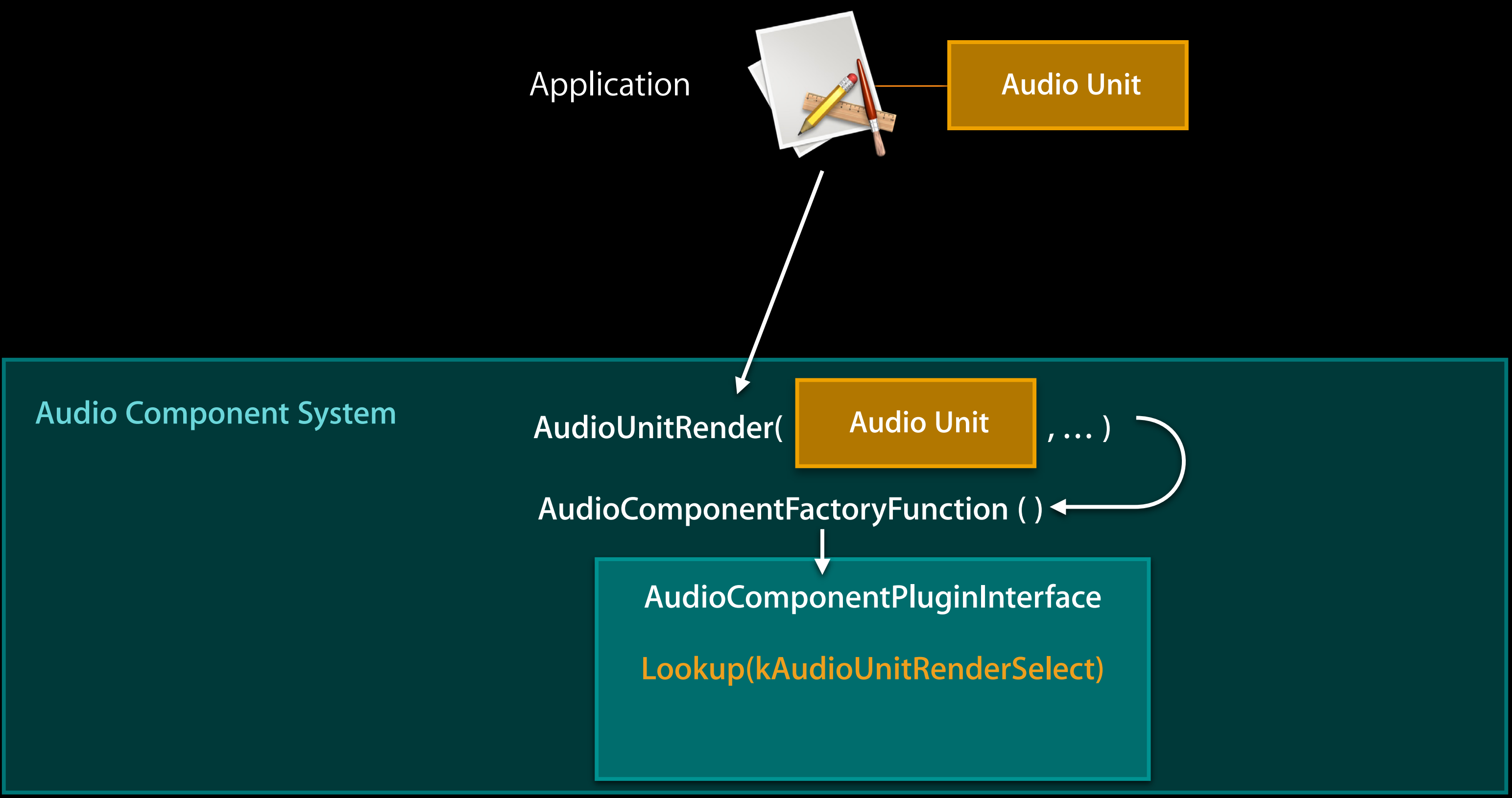
Calling into an Audio Unit

Application
Audio Unit
Audio Component System
AudioUnitRender(Audio Unit , …)
AudioComponentFactoryFunction ( )

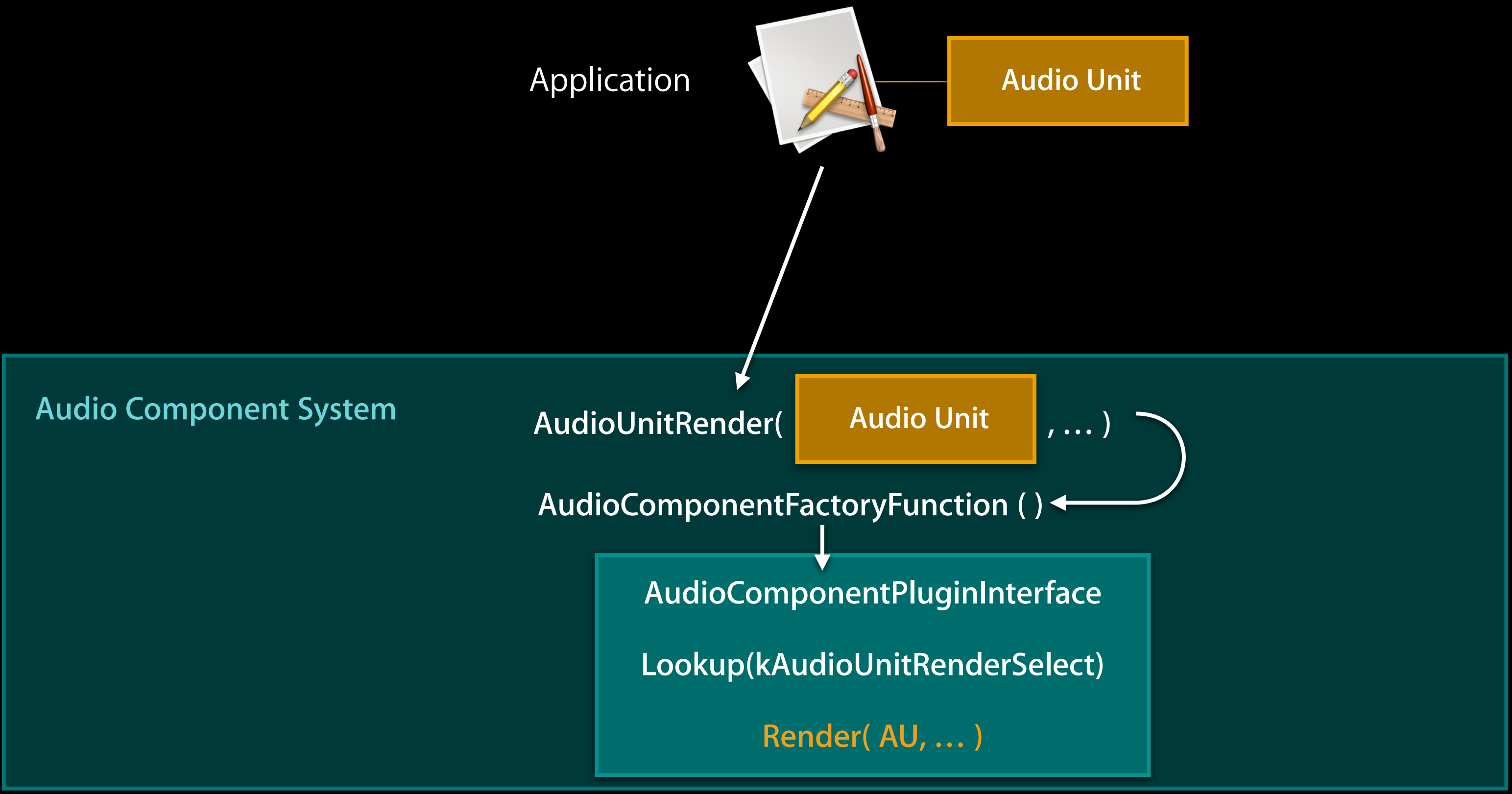
Calling into an Audio Unit


Application
Audio Unit
Audio Component System
AudioUnitRender(Audio Unit , ...)
AudioComponentFactoryFunction ( )
AudioComponentPluginInterface

Calling into an Audio Unit


Application
Audio Unit
Audio Component System
AudioUnitRender(Audio Unit , …)
AudioComponentFactoryFunction ( )
AudioComponentPluginInterface
Lookup(kAudioUnitRenderSelect)

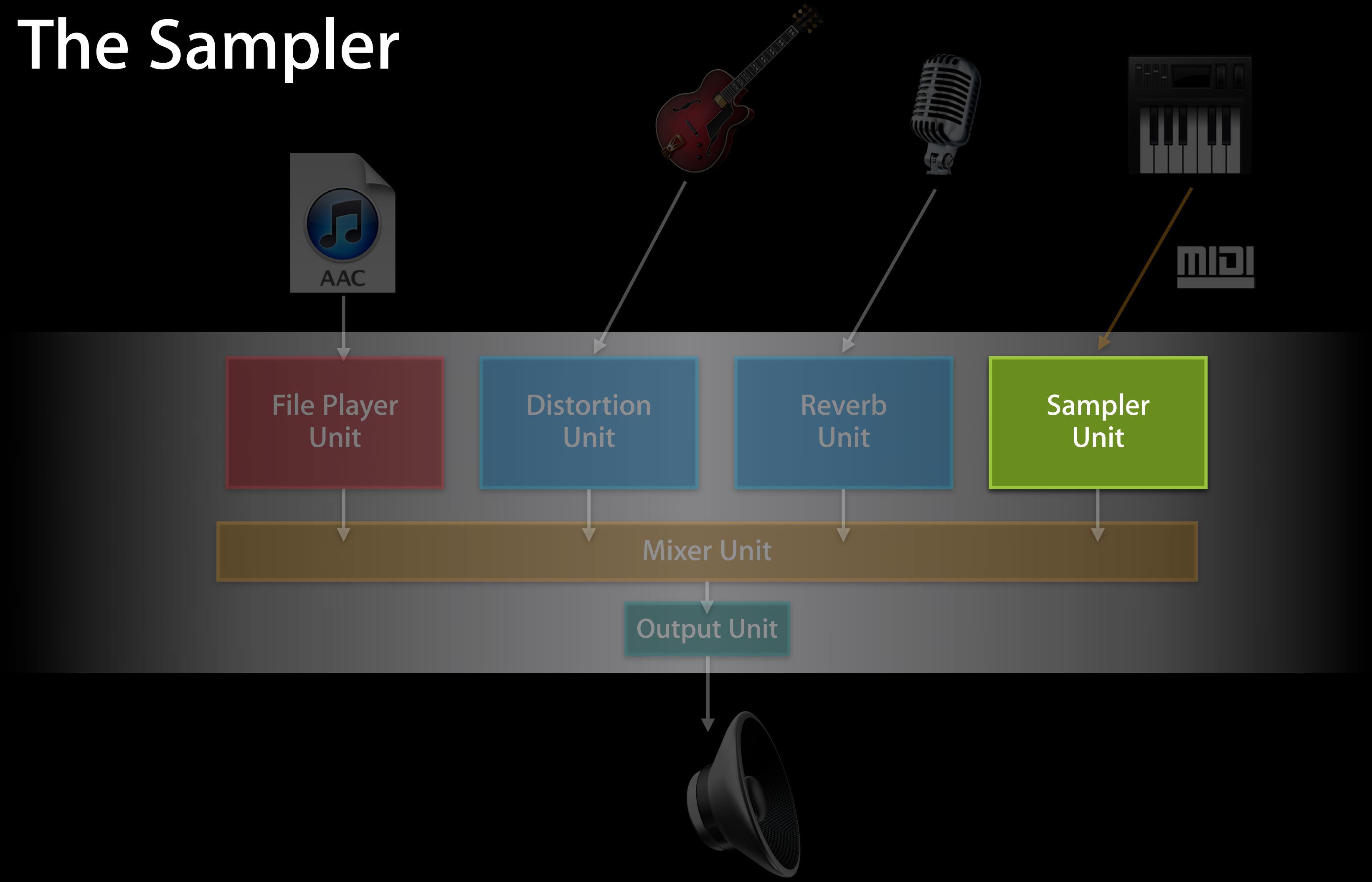
Calling into an Audio Unit


Application
Audio Unit
Audio Component System
AudioUnitRender(Audio Unit , …)
AudioComponentFactoryFunction ( )
AudioComponentPluginInterface
Lookup(kAudioUnitRenderSelect)
Render(AU, …)

The Sampler Audio Unit

Doug Scott
Core Audio Engineering

The Sampler


AAC
MIDI
File Player Unit
Distortion Unit
Reverb Unit
Sampler Unit
Mixer Unit
Output Unit

A New Instrument for iOS and Mac OS X

- What makes this an instrument?
    - Generates audio output
    - Music events trigger notes and change behavior

A New Instrument for iOS and Mac OS X

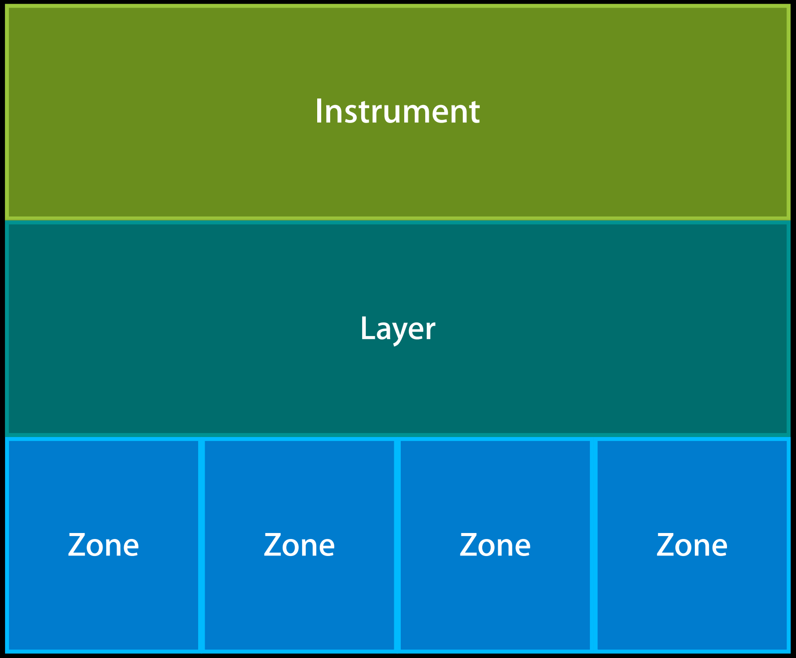
- What makes this an instrument?
  - Generates audio output
  - Music events trigger notes and change behavior
- What makes this a sampler?
  - Audio files organized as a playable instrument
    - Drum kit
    - Acoustic piano
    - Sound effects

Sampler Features

- Accepts samples in multiple formats
- Shares resources
- Streams large audio files
- Lightweight native presets
- Flexible instrument preset design
- Translates DLS and SoundFont2 instrument presets

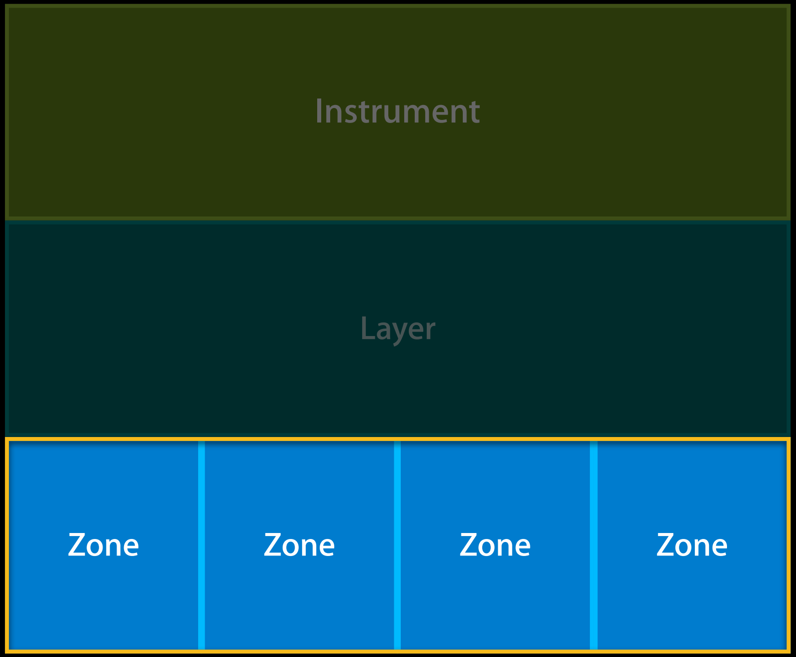
How a Sampler Patch Is Organized

A hierarchy of zones and layers



Zones

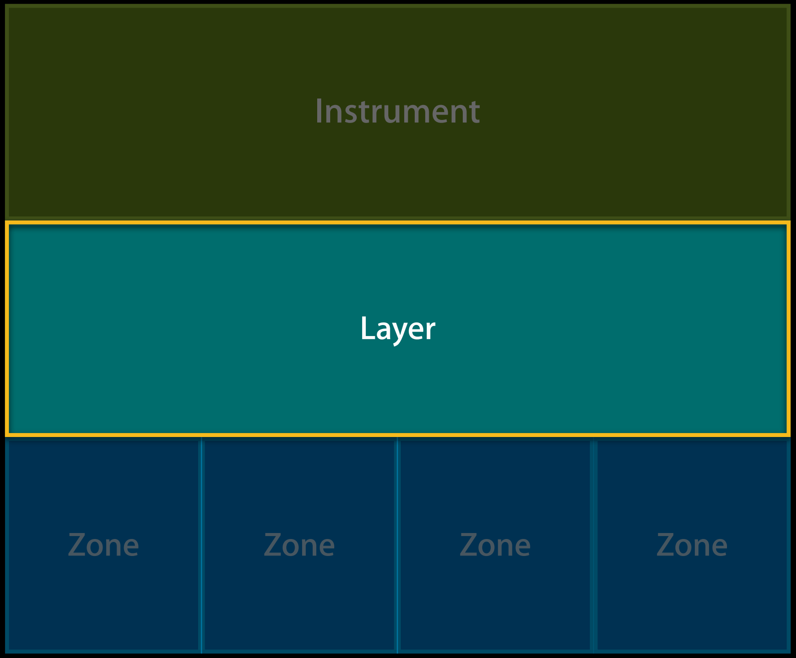
How to map each sample



- Inherit from parent layer
- Root key
- Key number range
- Key velocity range
- Waveform looping
- Gain
- Detune
- etc.

Layers

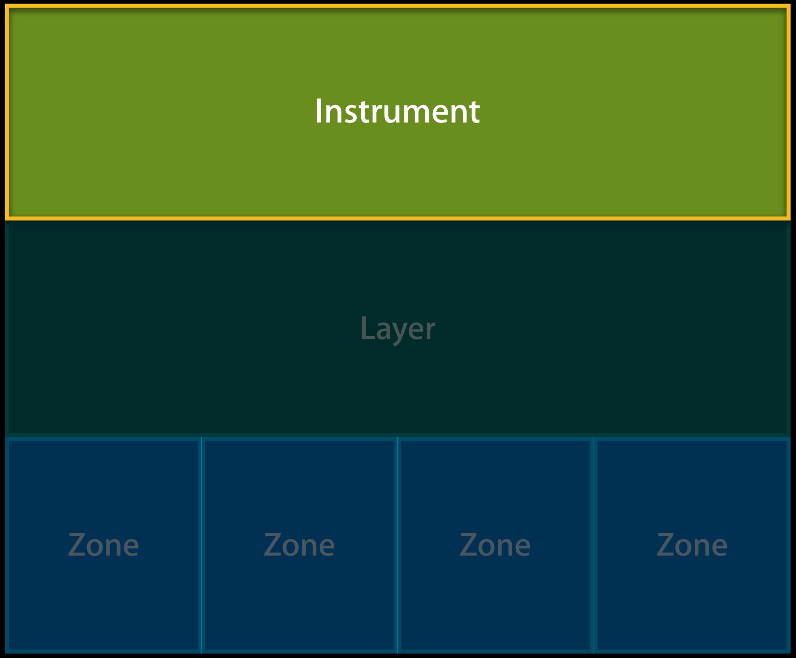
Allow zones to share common settings



- Collection of zones
- Settings for filters, LFOs, envelopes, etc.
- Modulation connections
- Zone selection
- Key offset
- etc.

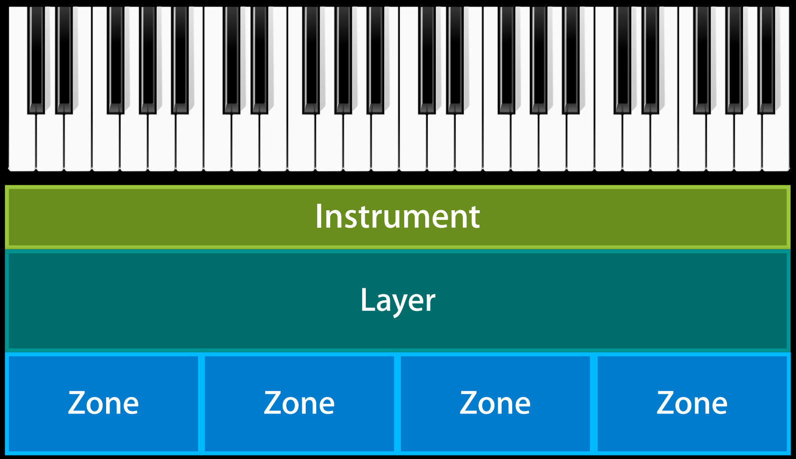
Instrument

A collection of layers



A Simple Patch

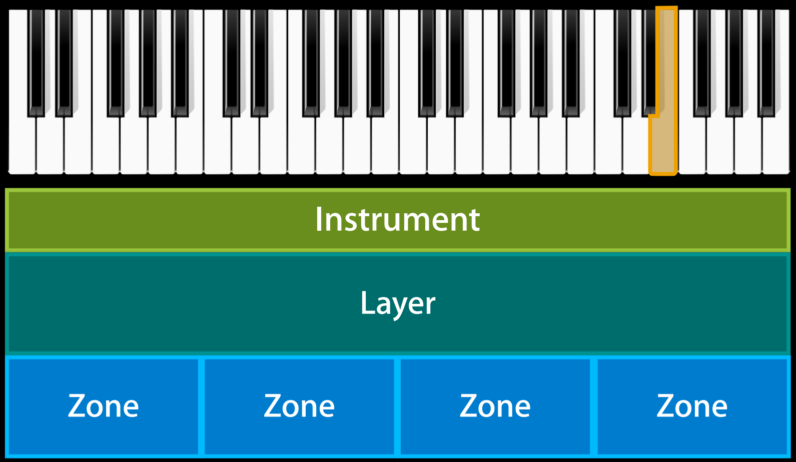
A single layer with multiple zones



- Layer spans entire keyboard
- Divided into four zones

Sampler Behavior

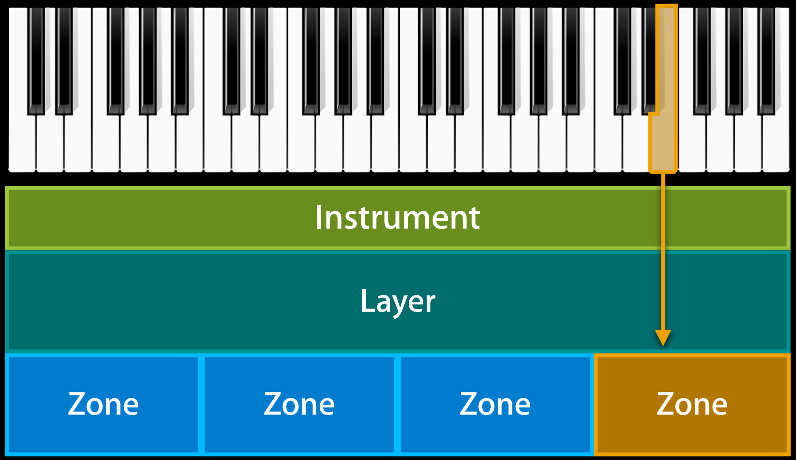
A single layer with multiple zones



- MIDI note on received

Sampler Behavior

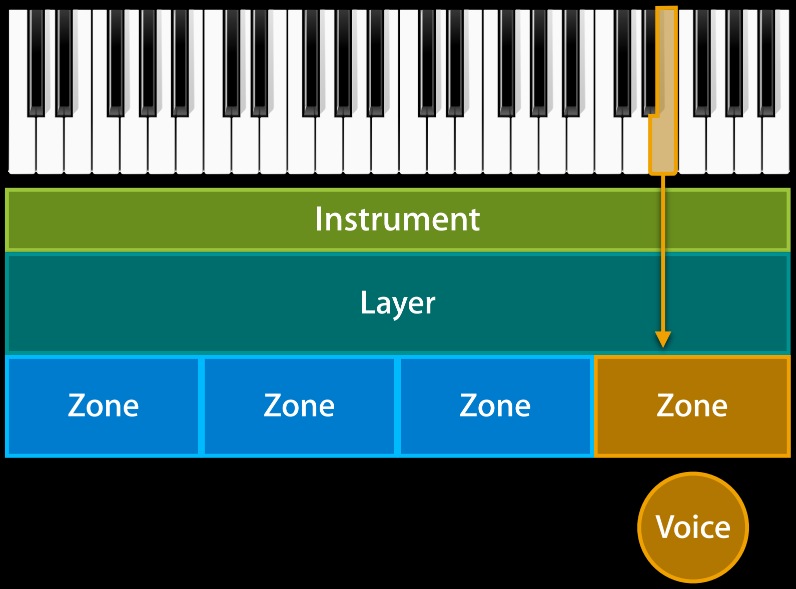
A single layer with multiple zones



- MIDI note on received
- Zone selected

Sampler Behavior

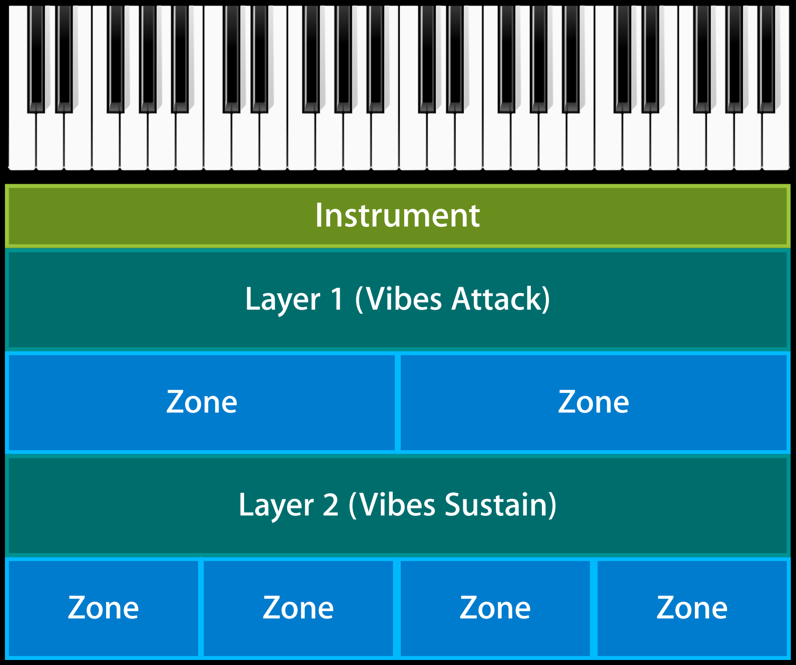
A note-on generates a single voice



- MIDI note on received
- Zone selected
- Voice is activated

A Layered Patch

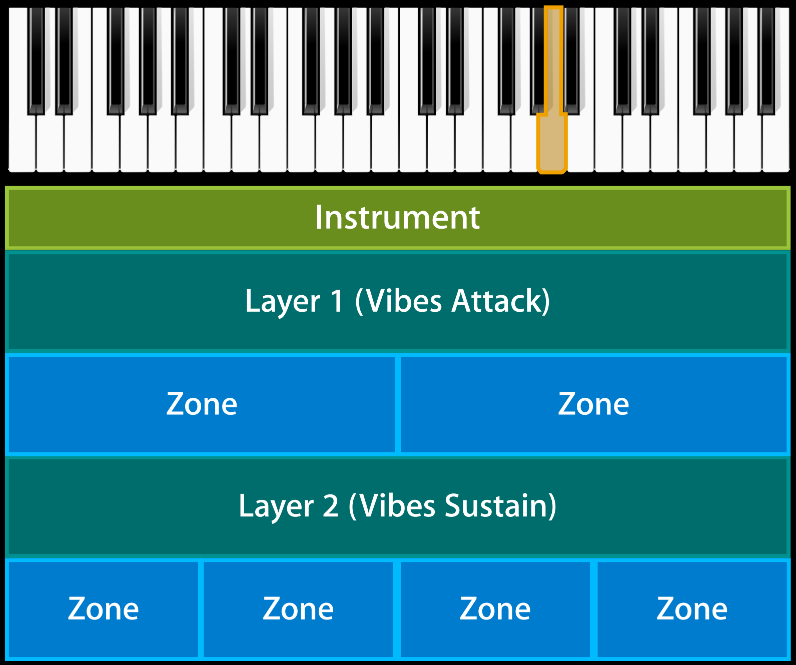
Two layers which overlap to produce a complex instrument



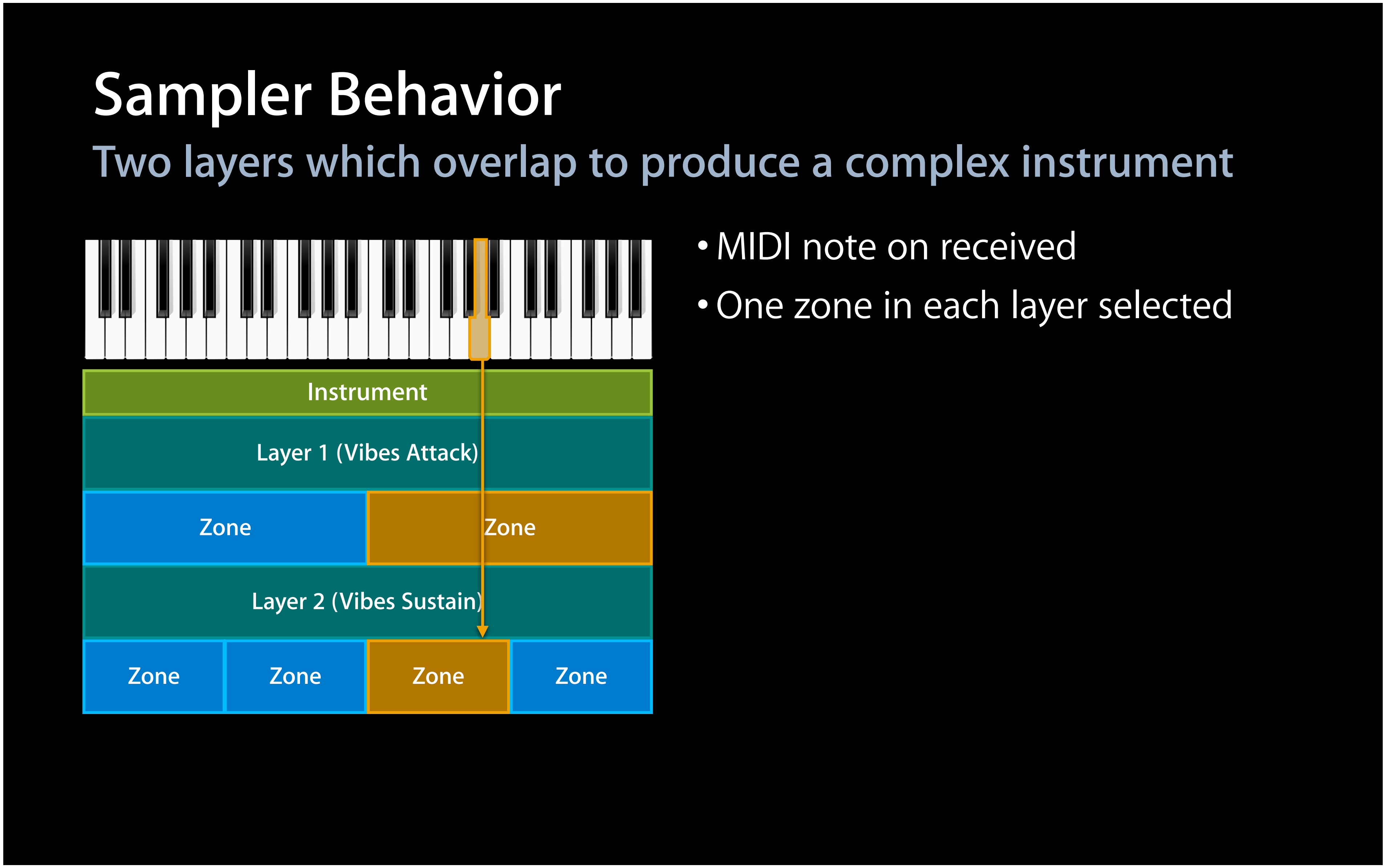
- Both layers span entire keyboard
- First layer divided into two zones
- Second layer divided into four zones

Sampler Behavior

Two layers which overlap to produce a complex instrument

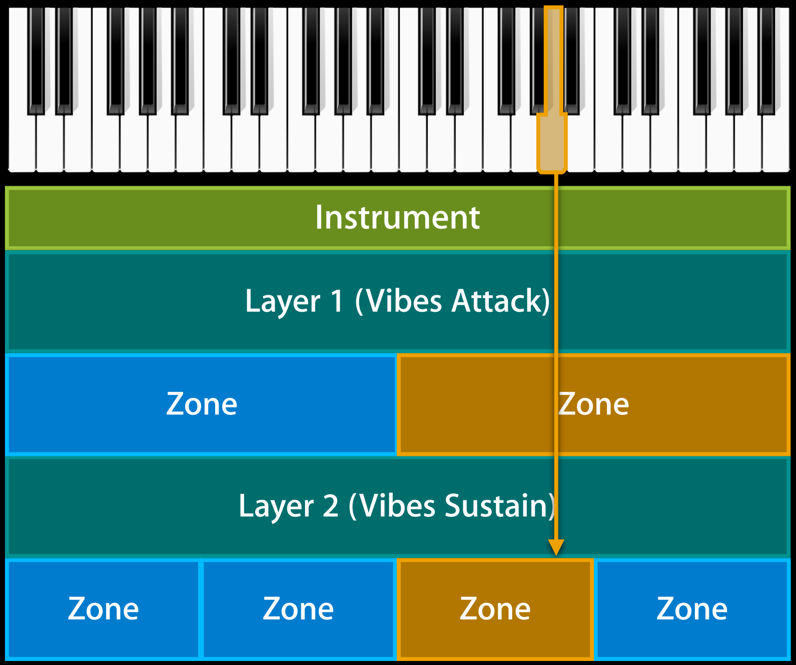


- MIDI note on received



Sampler Behavior

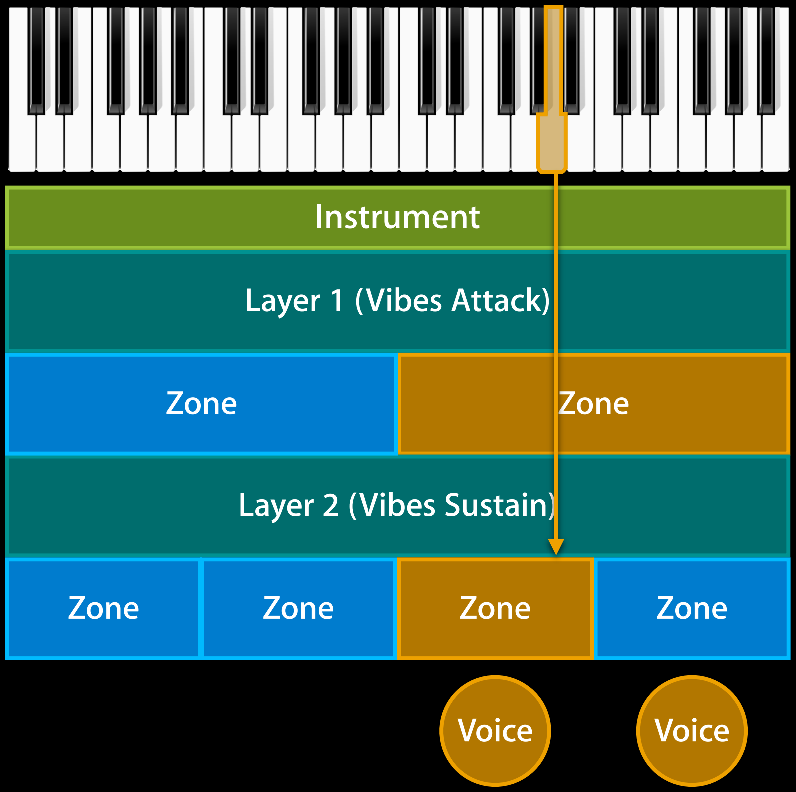
Two layers which overlap to produce a complex instrument



- MIDI note on received
- One zone in each layer selected

Sampler Behavior

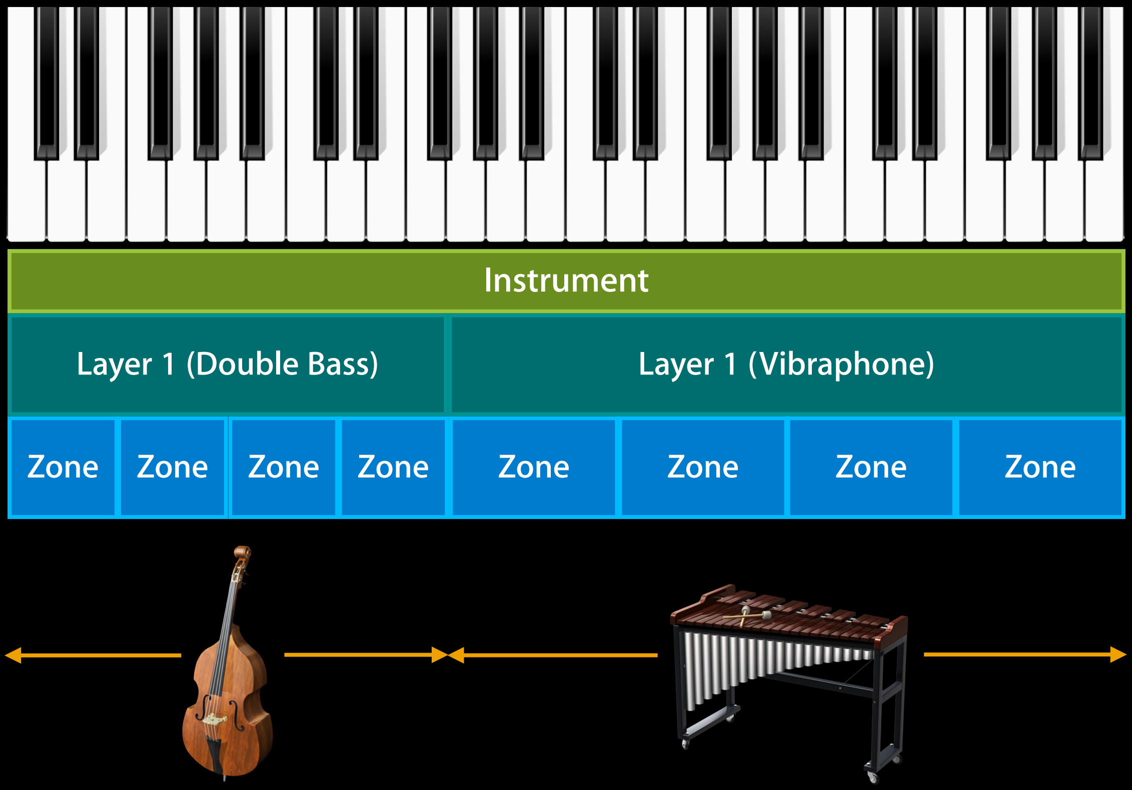
A note on generates two voices



- MIDI note on received
- One zone in each layer selected
- Two voices are activated

Example of a Complex Patch

Keyboard split into two instrumental timbres



Demo

The Sampler’s Custom View and Presets

Doug Scott
Core Audio Engineering

Configuring the Sampler

Loading a patch

- AUPreset file
- Build from set of audio files
- DLS bank or SoundFont 2 files

Loading a Patch Using an AUPreset File

- Audio file assets in app bundle’s resource directory (required on iOS)
- Convert the preset file into a PropertyList
- Load using the ClassInfo property

```
OSStatus result;
result = AudioUnitSetProperty(mySamplerUnit,
                              kAudioUnitProperty_ClassInfo,
                              kAudioUnitScope_Global,
                              0,
                              &presetPropertyList,
                              sizeof(CFPropertyListRef));
```

Creating a Patch from Audio Files

A new custom instrument

- Audio files in app bundle's resource directory (required on iOS)
- Audio file's instrument chunk
  - Sample loop
  - Key range
  - etc.

Loading a Patch from a Sound Bank

Sampler translates DLS and SoundFont2 patches

- Bank file in app bundle's resource directory (required on iOS)
- Select a preset
    - Bank ID
    - Instrument ID
- Load using AudioUnitSetProperty

Demo

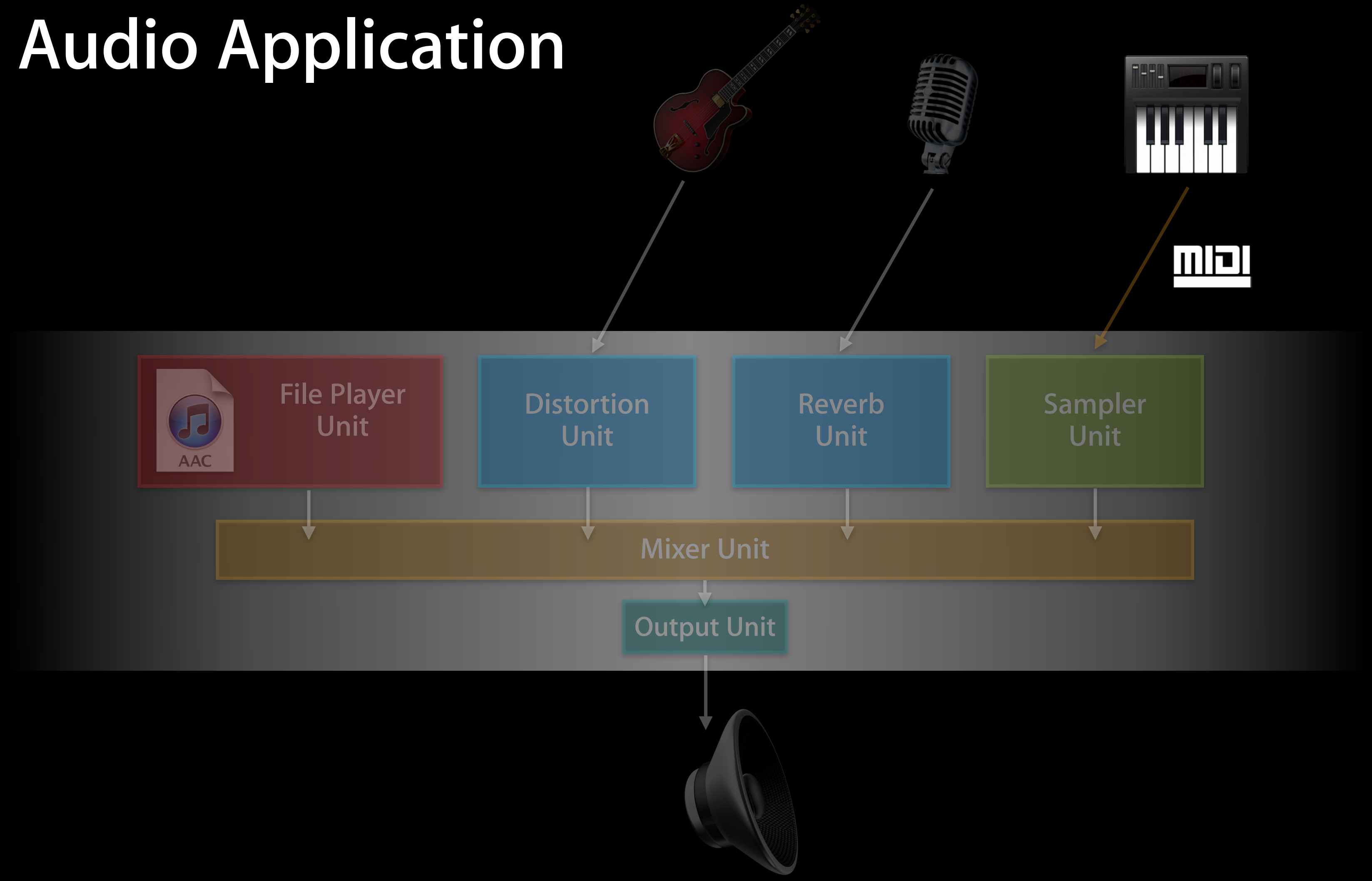
Using a Sampler AUPreset in Your App

Doug Scott
Core Audio Engineering

Introduction to CoreMIDI in iOS

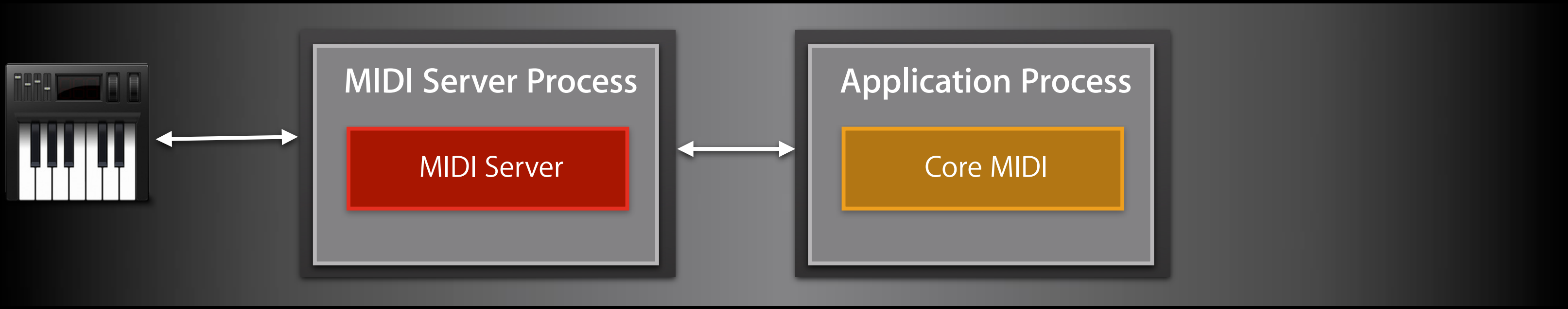
Michael Hopkins
Core Audio Engineering

Audio Application



What Is CoreMIDI?

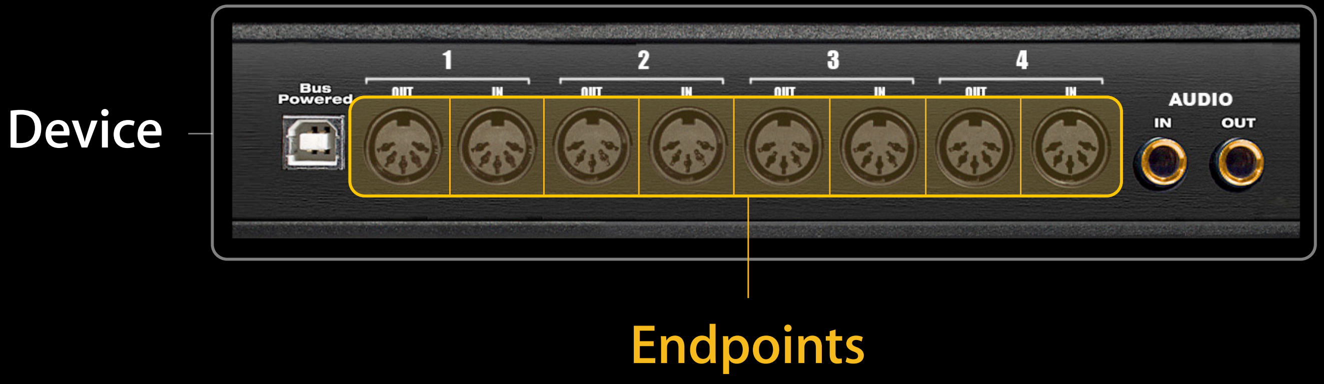
- A set of services that applications can use to communicate with MIDI devices
- Provides abstractions for interacting with a MIDI network



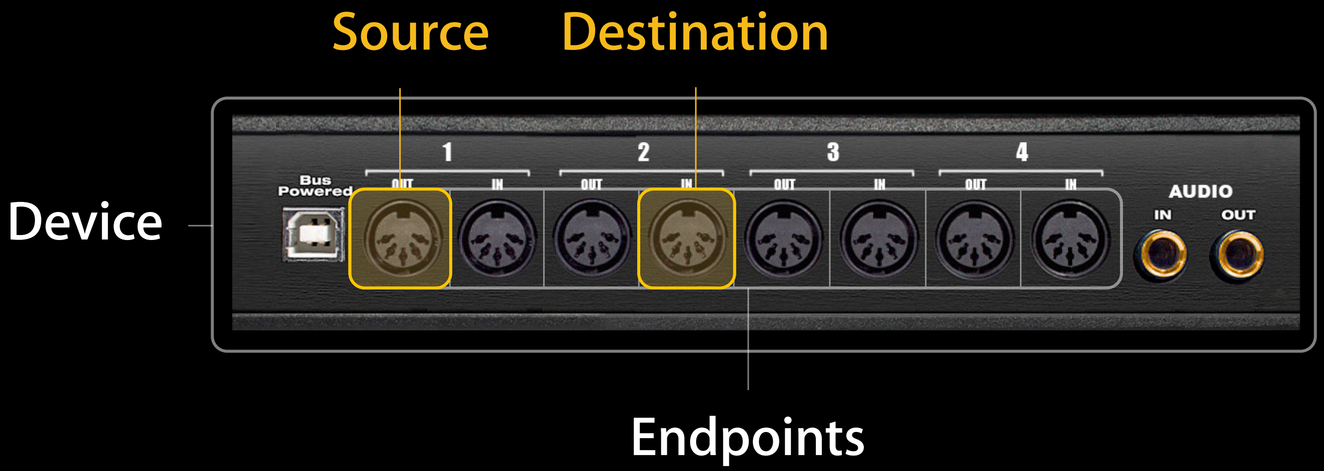
MIDI Devices

Device

MIDI Devices



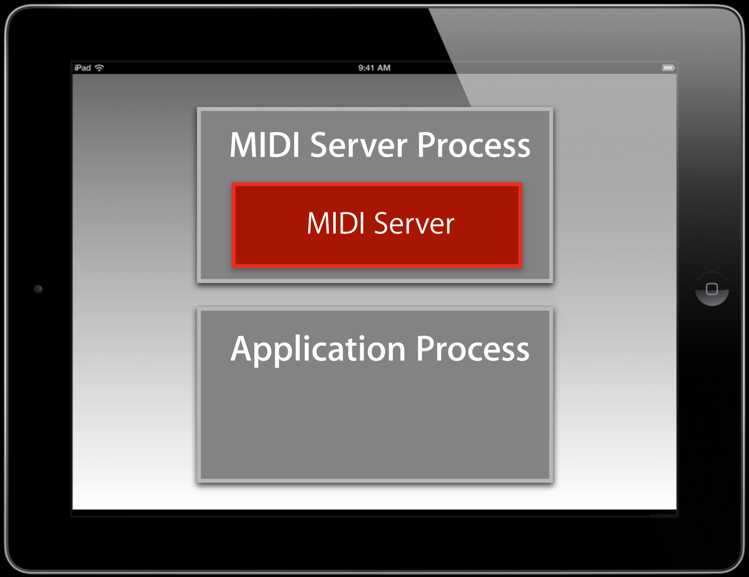
MIDI Devices



Handling Device Notifications

Plugging in a new device

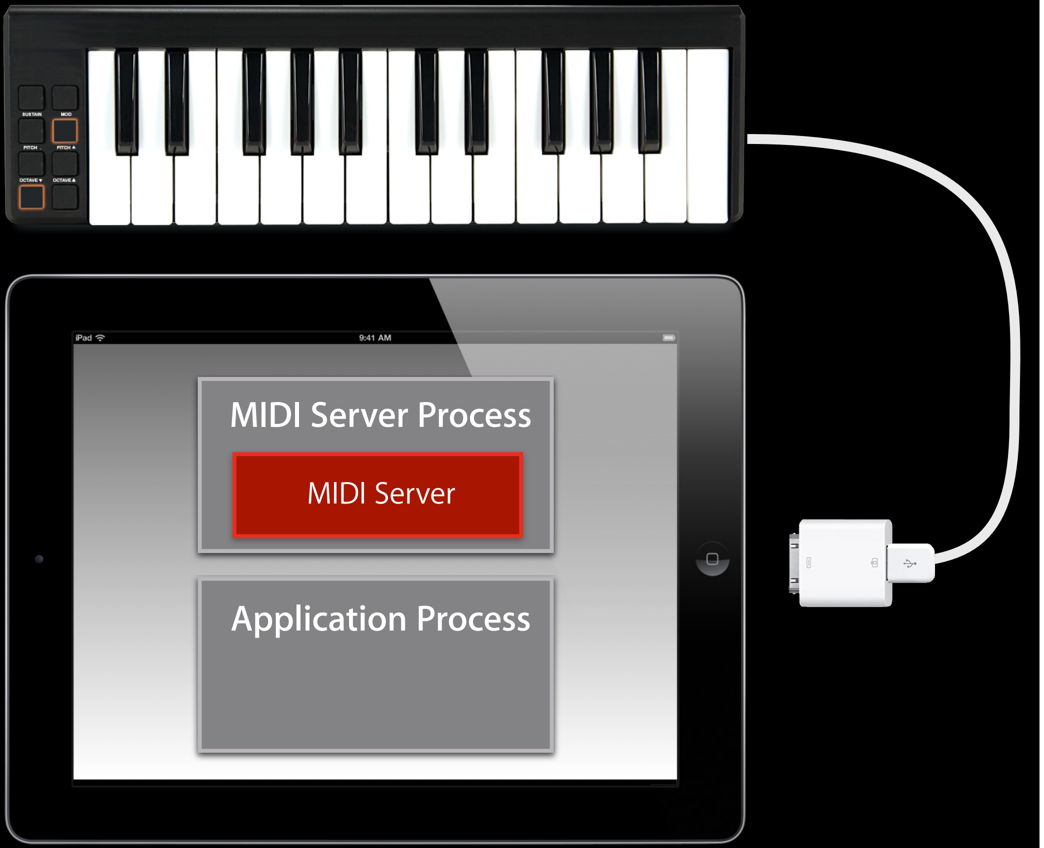
- Application needs to create a client object
- MIDI Server calls application's MIDINotifyProc() when
  - Device changes
  - Property changes
  - Setup changes



Handling Device Notifications

Plugging in a new device

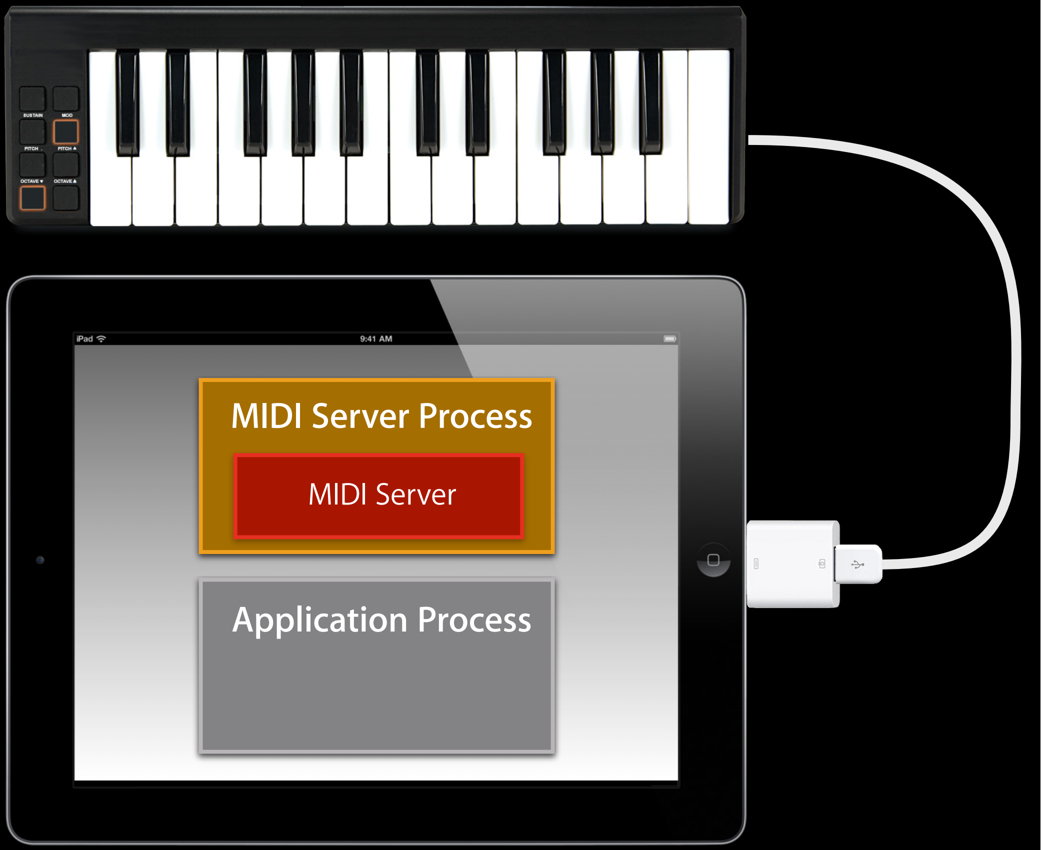
- Application needs to create a client object
- MIDI Server calls application's MIDINotifyProc() when
    - Device changes
    - Property changes
    - Setup changes



Handling Device Notifications

Plugging in a new device

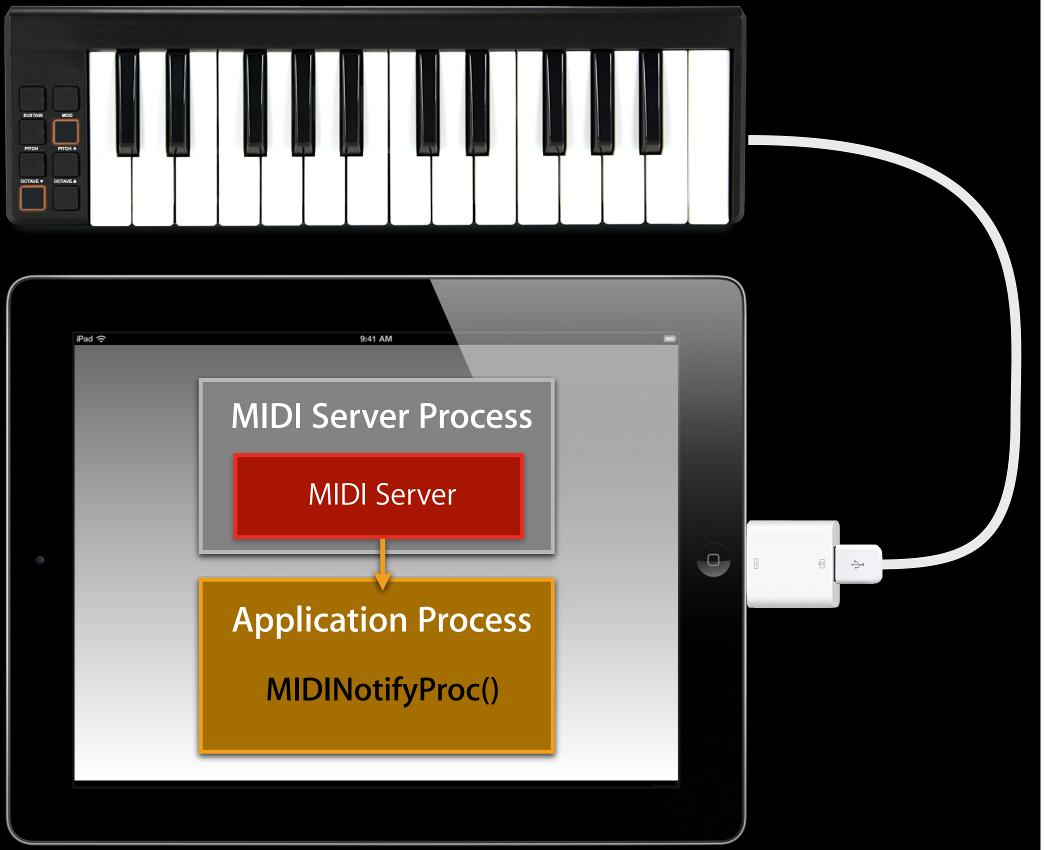
- Application needs to create a client object
- MIDI Server calls application's MIDINotifyProc() when
  - Device changes
  - Property changes
  - Setup changes



Handling Device Notifications

Plugging in a new device

- Application needs to create a client object
- MIDI Server calls application's MIDINotifyProc() when
    - Device changes
    - Property changes
    - Setup changes



CoreMIDI Properties

- Properties can get information about devices, entities, or endpoints
  - Name
  - Manufacturer
  - Unique ID
  - Offline state
  - Receive and transmit channels
  - Current patch
  - MIDI settings
    - Supports General MIDI
    - Supports MMC
    - Receives/transmits clock

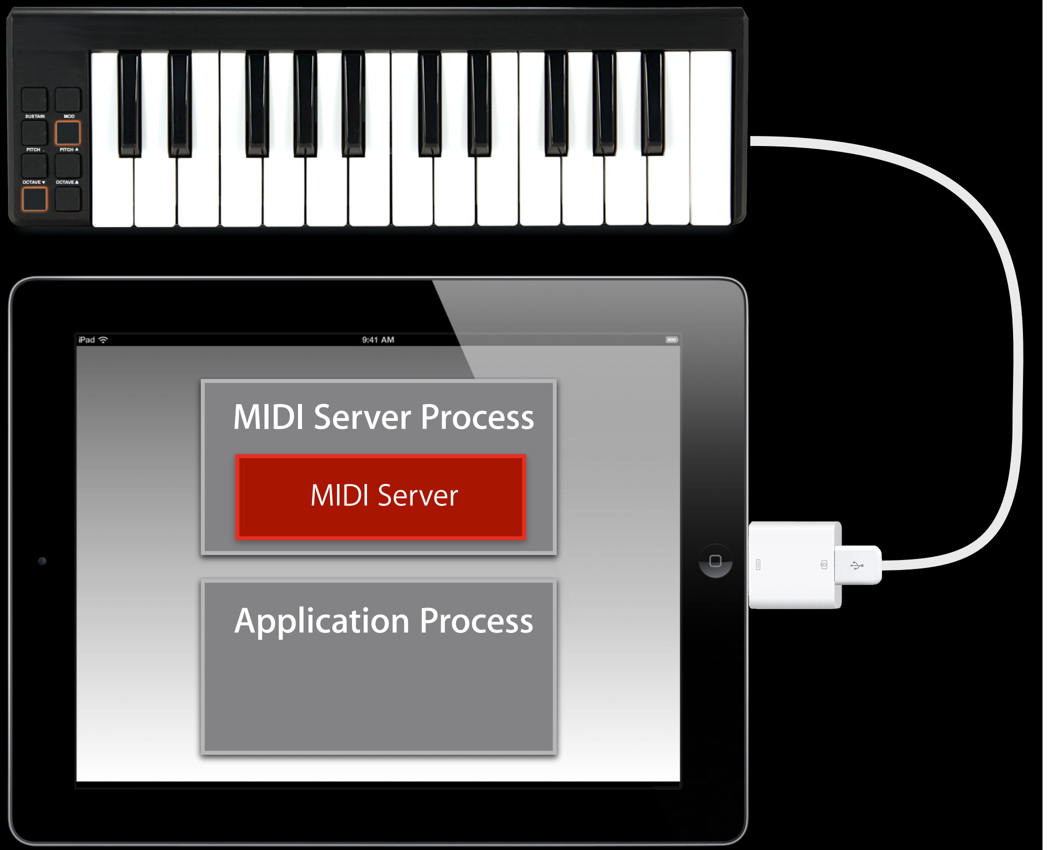
Creating a MIDIClient

```
void MIDINotificationHandler(const MIDINotification *inMsg, void *inRefCon)
{
    switch (inMsg->messageID) {
        case kMIDIMsgPropertyChanged:
            ...
        break;
    }
}
```

```
MIDIClientRef midiClient;
OSStatus result = MIDIClientCreate(CFSTR(“My Client”),
                                   MIDINotificationHandler,
                                   NULL, &midiClient);
```

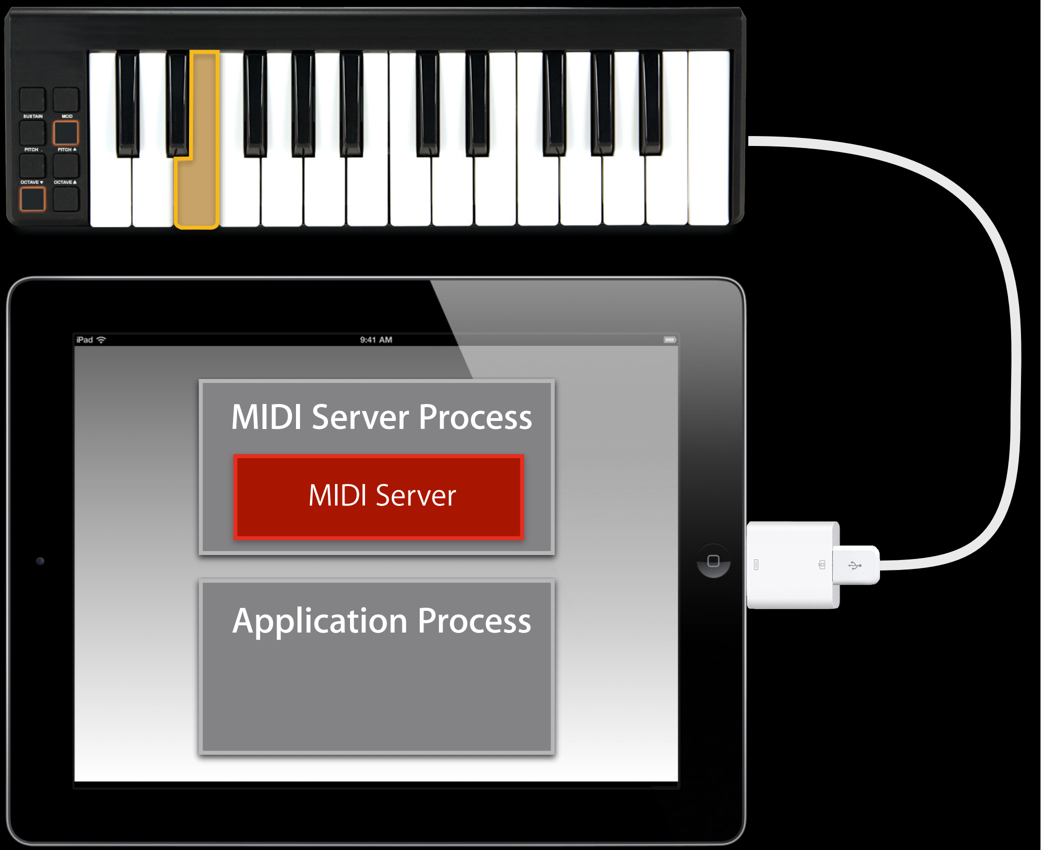
Getting MIDI Data

- Application needs to create a MIDI input port
- MIDI Server calls application's MIDIReadProc() when MIDI messages are received
- Provides a packet list of MIDI events



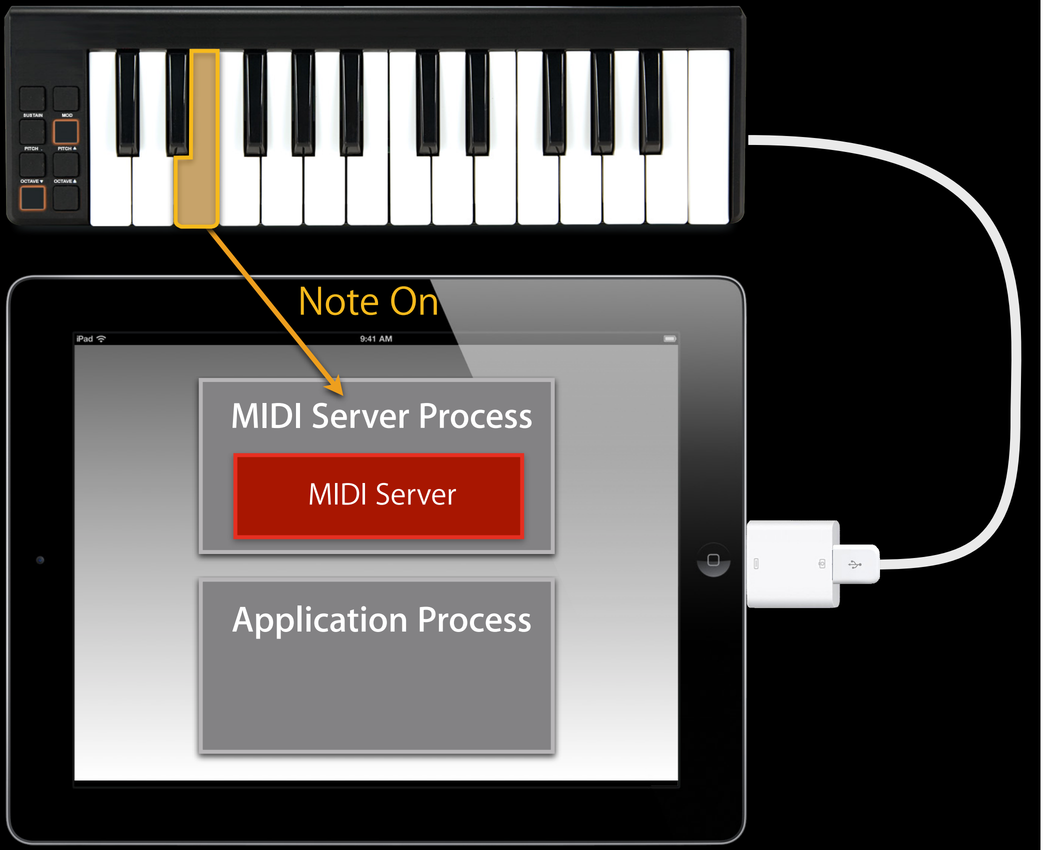
Getting MIDI Data

- Application needs to create a MIDI input port
- MIDI Server calls application's MIDIReadProc() when MIDI messages are received
- Provides a packet list of MIDI events



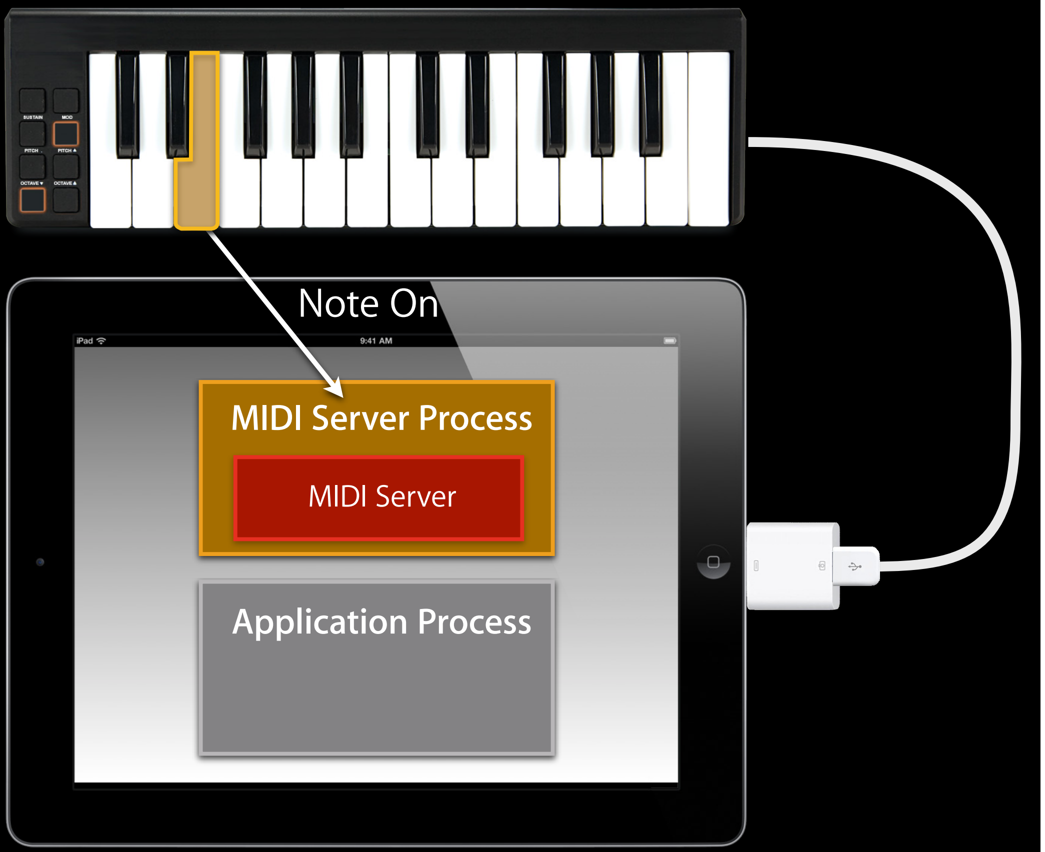
Getting MIDI Data

- Application needs to create a MIDI input port
- MIDI Server calls application's MIDIReadProc() when MIDI messages are received
- Provides a packet list of MIDI events



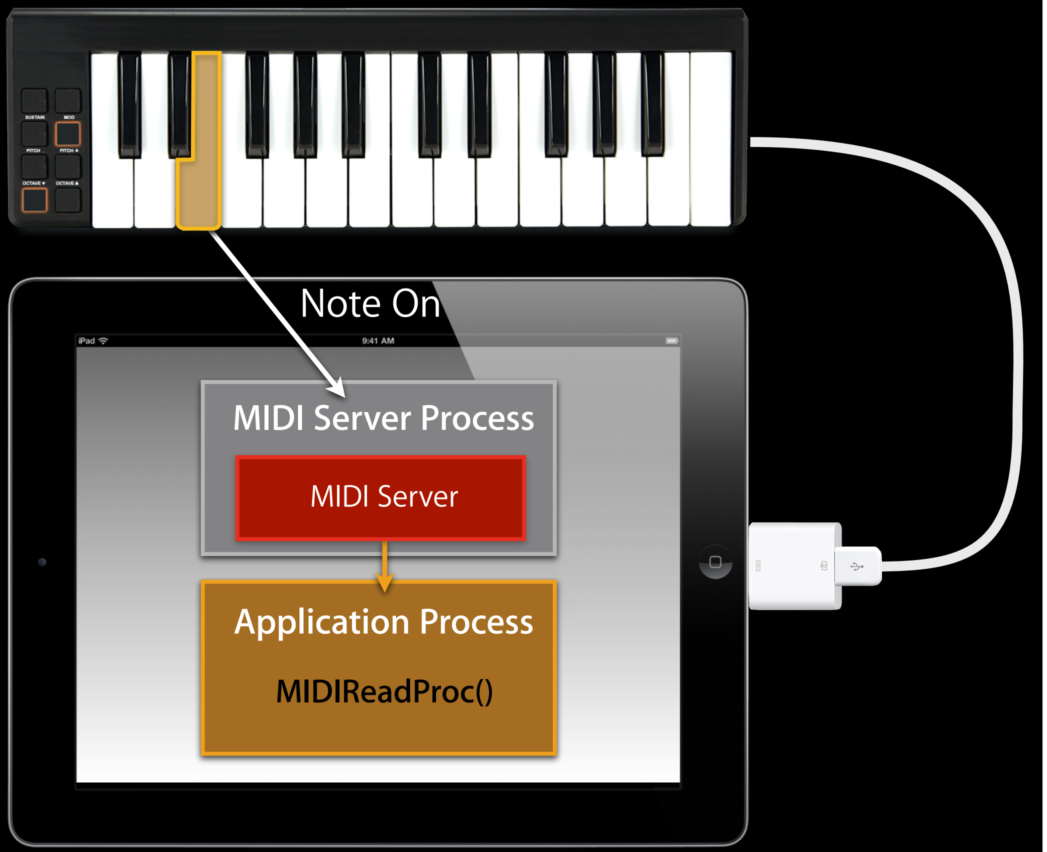
Getting MIDI Data

- Application needs to create a MIDI input port
- MIDI Server calls application's MIDIReadProc() when MIDI messages are received
- Provides a packet list of MIDI events



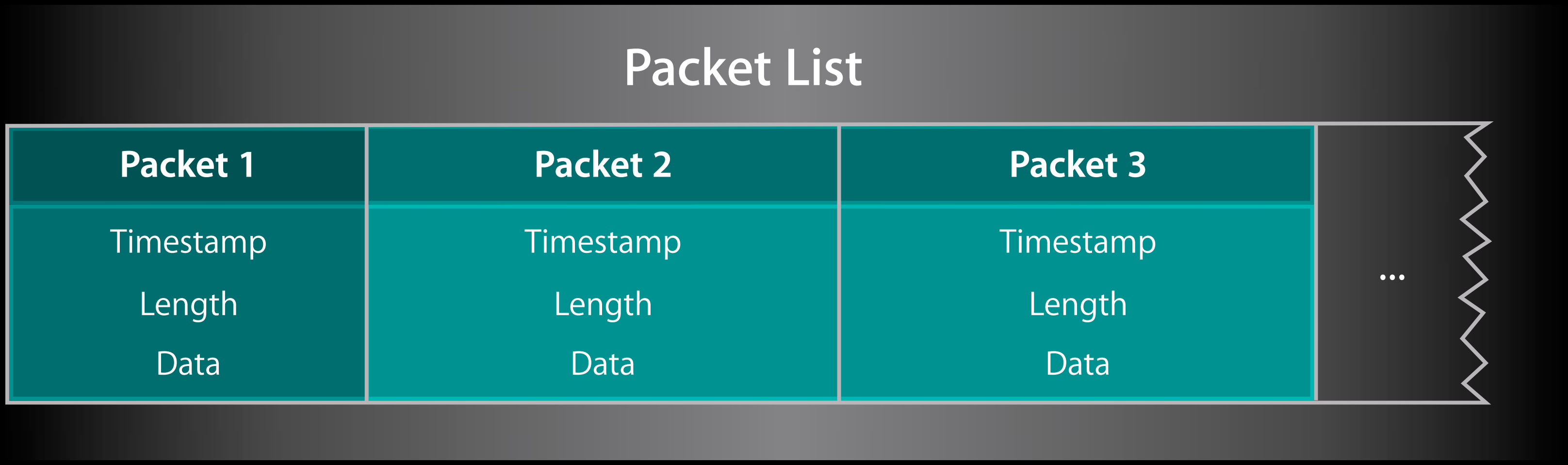
Getting MIDI Data

- Application needs to create a MIDI input port
- MIDI Server calls application's MIDIReadProc() when MIDI messages are received
- Provides a packet list of MIDI events



MIDI Packets

Anatomy of a MIDIPacket



```
p1=PacketList[0]   p2=MIDIPacketNext(p1)  p3=MIDIPacketNext(p2)
```

Getting MIDI Data

Creating a port

```
void MIDIInputPortHandler(const MIDIPacketList *inPackets,
                          void *inRefCon, void *srcRefCon) {
    const MIDIPacket *pkt = &(inPackets->packet[0];
    for (int i = 0; i < inPackets->numPackets; pkt = MIDIPacketNext(pkt)) {
        ...  // process packets
    }
}
```

```
MIDIPortRef midiInputPort;
OSStatus result = MIDIInputPortCreate(midiClient, CFSTR(“InputPort”),
                              MIDIInputPortHandler, NULL, &midiInputPort);
```

Sending MIDI Data

- Use MIDISend( ) to send a packet list of MIDI Data
- Use MIDIOutputPortCreate( ) to create an output port
- Specify the destination for the data

```
MIDISend(MIDIPortRef port,
         MIDIEndpointRef dest,
         const MIDIPacketList *packets)
```

Using Networked MIDI Connections

```
#import <CoreMIDI/MIDINetworkSession.h>

MIDINetworkSession *session = [MIDINetworkSession defaultSession];
session.enabled = YES;
session.connectionPolicy = MIDINetworkConnectionPolicy_Anyone;
```

```
MIDINetworkHost *host = [MIDINetworkHost hostWithName: @“My Session”
                                              address: @“myhost.acme.com”
                                                 port: 5004];


[MIDINetworkConnection connectWithHost: host];
```

The Music Sequencer

Doug Scott
Core Audio Engineering

Demo

Playing MIDI files in your app

Doug Scott
Core Audio Engineering

The Music Sequencing API

- Declared in <AudioToolbox/MusicPlayer.h>

The Music Sequencing API

- Declared in <AudioToolbox/MusicPlayer.h>
- MusicSequence
    - Tempo track
    - Event tracks

The Music Sequencing API

- Declared in <AudioToolbox/MusicPlayer.h>
- MusicSequence
  - Tempo track
  - Event tracks
- MusicTrack
  - MIDI events
  - AU Parameter automation data
  - User data events

The Music Sequencing API

- Declared in <AudioToolbox/MusicPlayer.h>
- MusicSequence
  - Tempo track
  - Event tracks
- MusicTrack
  - MIDI events
  - AU Parameter automation data
  - User data events
- MusicPlayer

The Music Sequencing API

MusicSequence

- Add, remove, merge MusicTracks
- Read and write MIDI files
- Beats/time conversion

The Music Sequencing API

MusicTrack

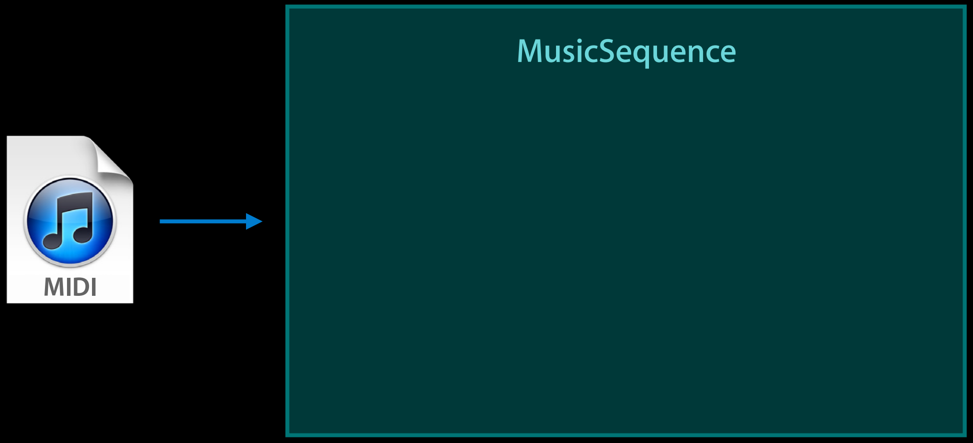
- Add, move, clear music events
- Mute, solo, looping, etc.
- Associate events with a destination
    - Audio Units (instruments, sound effects, etc.)
    - MIDI devices
- Supports event iteration

The Music Sequencing API

MusicPlayer

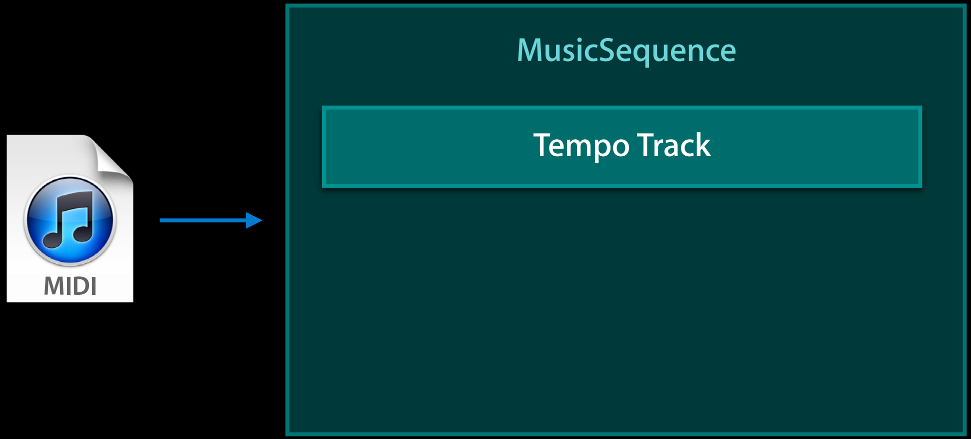
- Playback controls
- Host time to beats and vice versa

Loading a MIDI File—An Easy Use Pattern



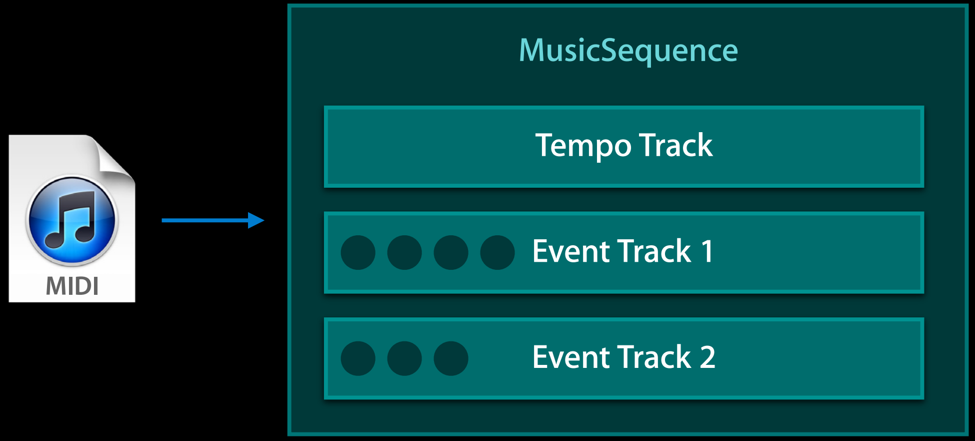
```
MusicSequence mySequence = ...
CFURLRef inPathToMIDIFile = ...
MusicSequenceFileLoad(mySequence, inPathToMIDIFile, 0,
                      kMusicSequenceLoadSMF_ChannelsToTracks);
```

Loading a MIDI File

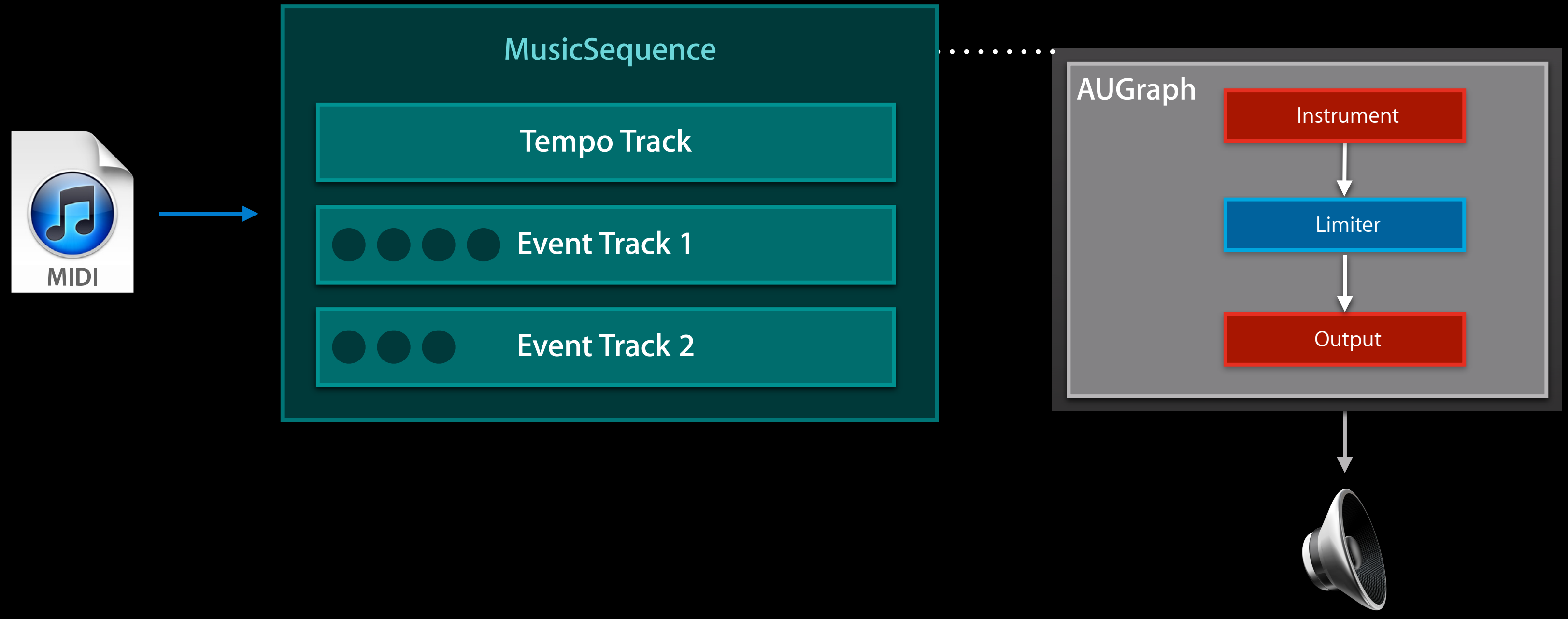


- Loads file’s tempo events if present
- Otherwise, creates a default tempo

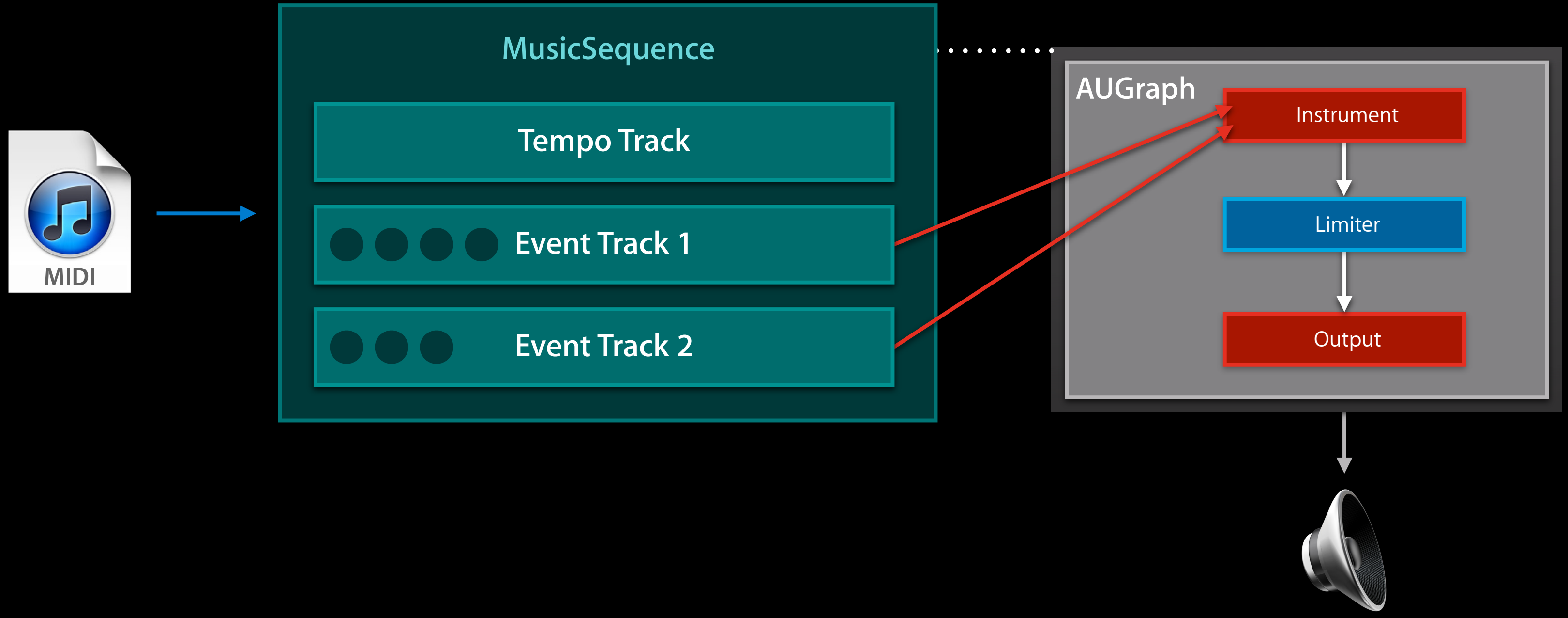
Loading a MIDI File



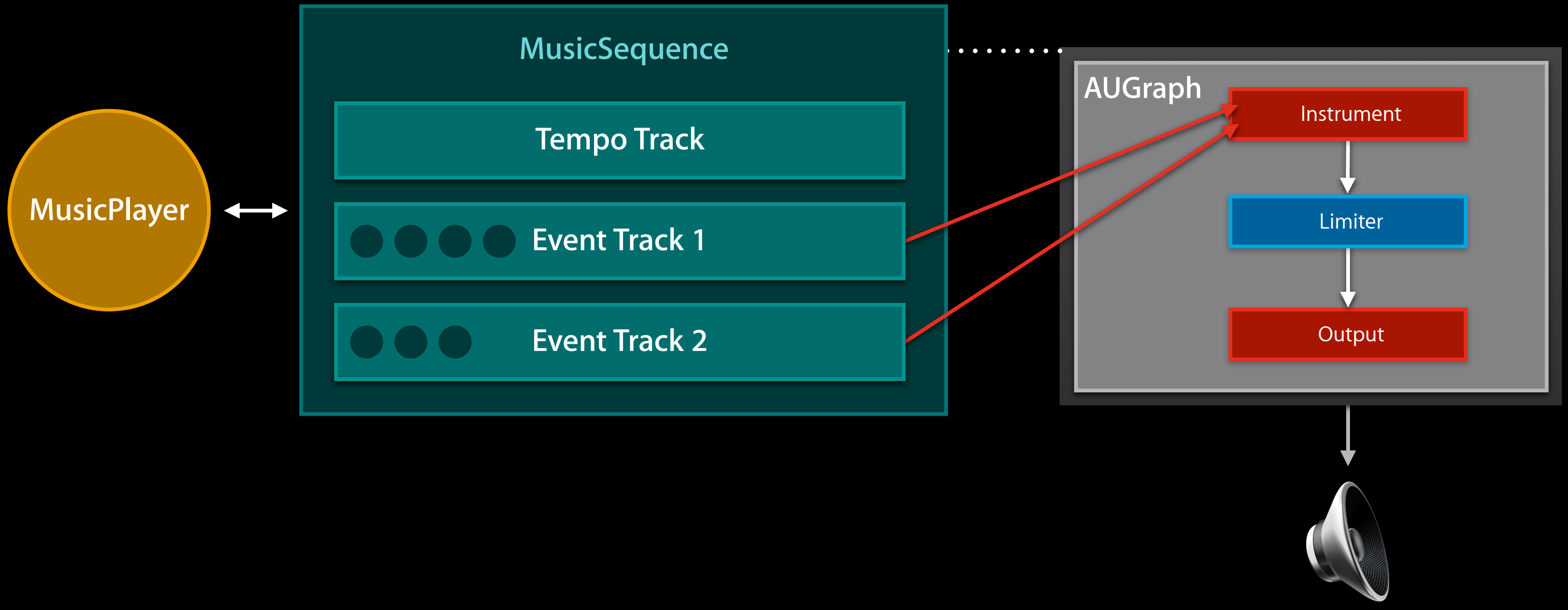
Loading a MIDI File



Loading a MIDI File



Playing the Sequence



```
MusicPlayerSetSequence(myMusicPlayer, mySequence);
MusicPlayerStart(myMusicPlayer);
MusicPlayerSetPlayRateScalar(myMusicPlayer, 1.0);
```

Creating a Custom Sequence

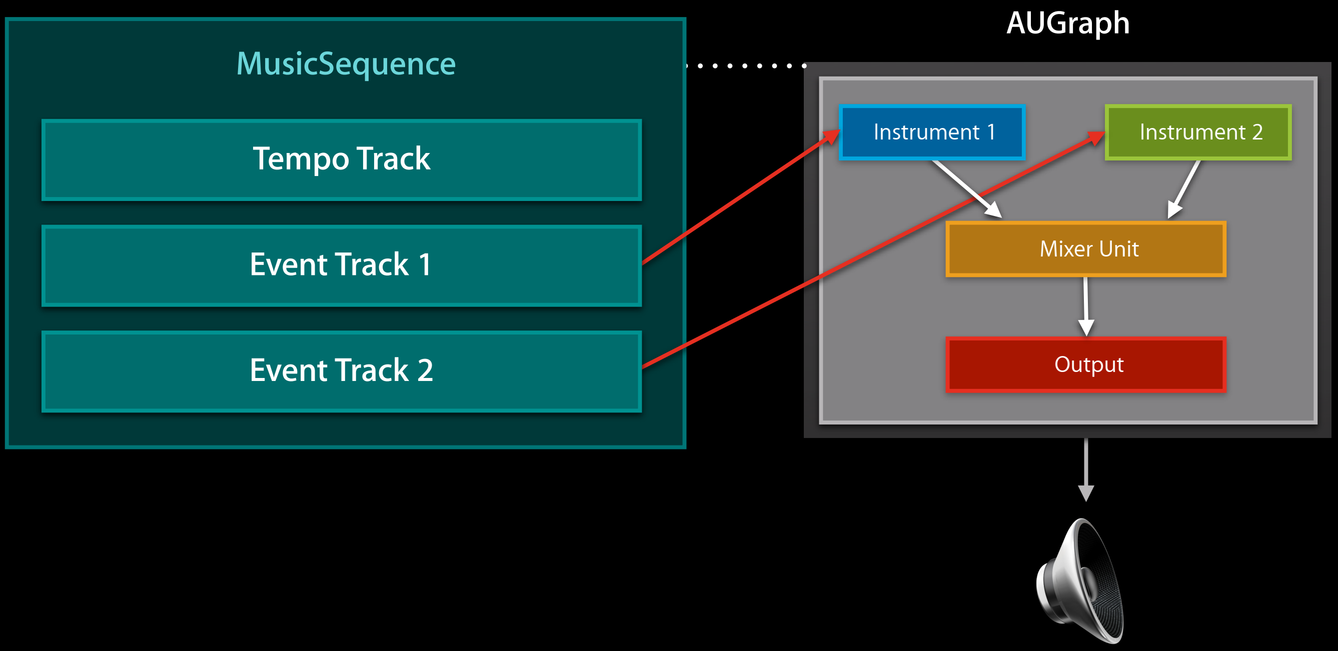
Complex usage pattern

- MIDI recorder/sequencer app
- Play a MIDI sequence with multiple instruments
- AU parameter automation data

Creating a Custom Sequence

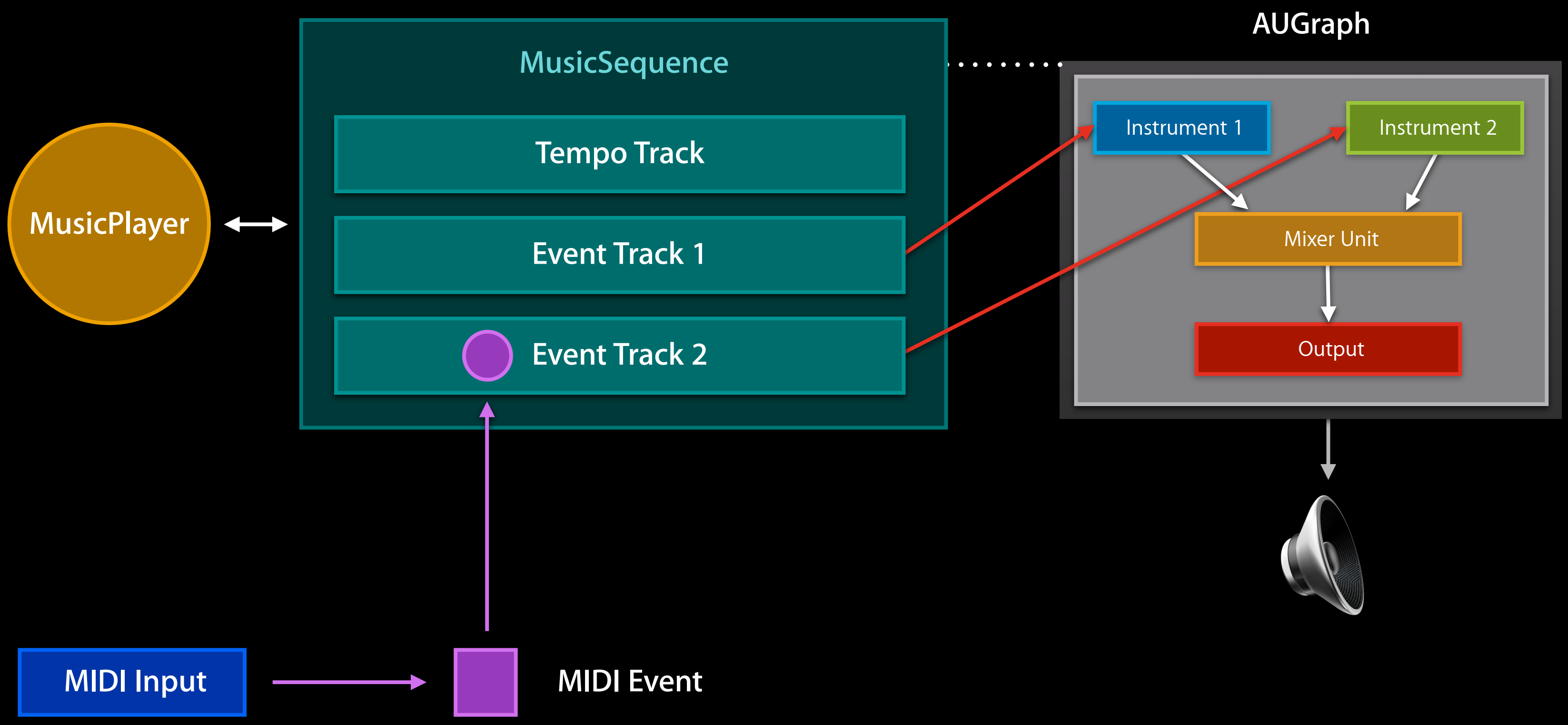
- Create an empty sequence
- Create a custom AUGraph
- Add tracks to sequence
- Add events to tracks
- Target tracks to graph nodes or MIDI endpoints

An Example of a Custom Sequence



• The two event tracks send their events to different instruments

Adding a Live Event to a Track



Summary

Technologies for music applications

- Audio Units
- AUSampler
- CoreMIDI
- Music sequencing

Related Sessions

Audio Session Management for iOS	Marina Wednesday 11:30 AM

Labs

Audio Lab	Graphics & Media Lab C Wednesday 2:00PM

More Information

Eryk Vershen
Media Technologies Evangelist
evershen@apple.com

Documentation and Sample Code
iPhone Dev Center
http://developer.apple.com/iphone

Audio Unit Hosting Guide for iPhone OS
WWDC attendee website

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

WWDC2011