What's New in Core Audio for iOS

Session 602

Tony Guetta
Manager, Core Audio System Software

Introduction

- Overview of new audio features in iOS 7
- Focus on one exciting new technology

Audio Input Selection

NEW

- Select among available audio inputs
- Choose which microphone on multi-mic devices
- Set microphone polar pattern
    - Achieves directivity through beam forming processing
    - e.g., cardioid, subcardioid
- See `<AVFoundation/AVAudioSession.h>`

Multichannel Audio Enhancements

NEW

- Discover maximum number of input and output channels
- Set preferred number of input and output channels
- Get audio channel labels
- See `<AVFoundation/AVAudioSession.h>`

Open AL Extensions

NEW

- Per-source spatialization rendering quality
  - Improved high-quality rendering algorithm
  - Render to multichannel output hardware
- Output capture
- See `<OpenAL/oalMacOSX_OALExtensions.h>`

Audio Queue Time-Pitch Capabilities

NEW

- Set playback rate
- Adjust playback pitch
- See `<AudioToolbox/AudioQueue.h>`

Audio Recording Permission

NEW

- Recording now requires user approval
- One-time approval remembered for each application
- Changeable in Settings
- Silence until permission is granted
- Triggered by use of AVAudioSession categories that enable recording
- API for application control

```
{
    ...
    [[AVAudioSession sharedInstance] requestRecordPermission];
    ...
}
```

Deprecated Audio Session C API

- Use AVAudioSession
- See `<AVFoundation/AVAudioSession.h>`

New in iOS 7
Summary

- Audio input selection
- Multichannel audio enhancements
- Open AL extensions
- Audio Queue time-pitch capabilities
- Audio recording permission
- Audio Session C API officially deprecated—Use AVAudioSession

New in iOS 7

Summary

- Audio input selection
- Multichannel audio enhancements
- Open AL extensions
- Audio Queue time-pitch capabilities
- Audio recording permission
- Audio Session C API officially deprecated—Use AVAudioSession
- Inter-app audio

Inter-App Audio

NEW

- Stream audio between apps in real-time
- Built on familiar APIs

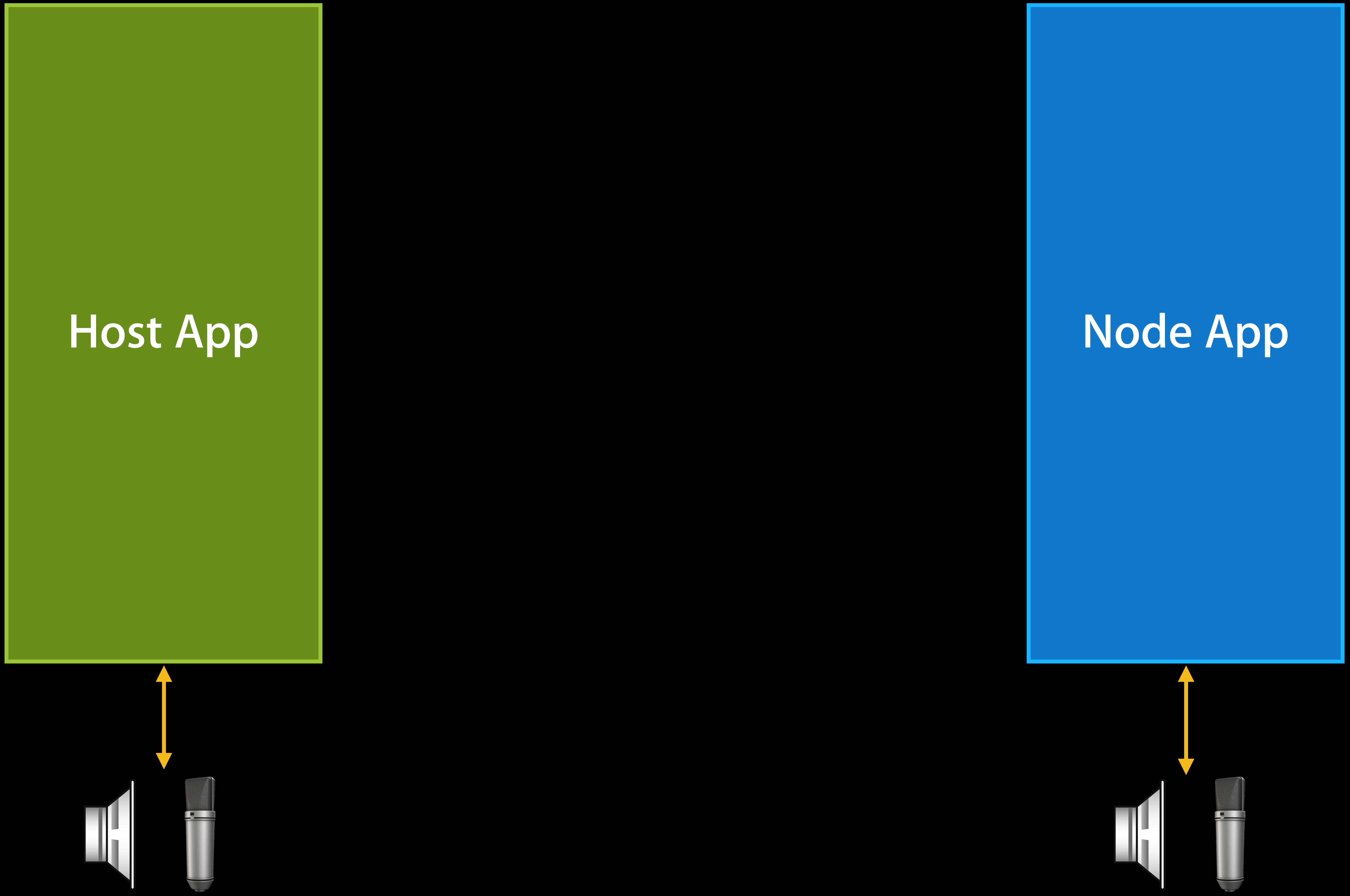
Demo

Alec Little
GarageBand

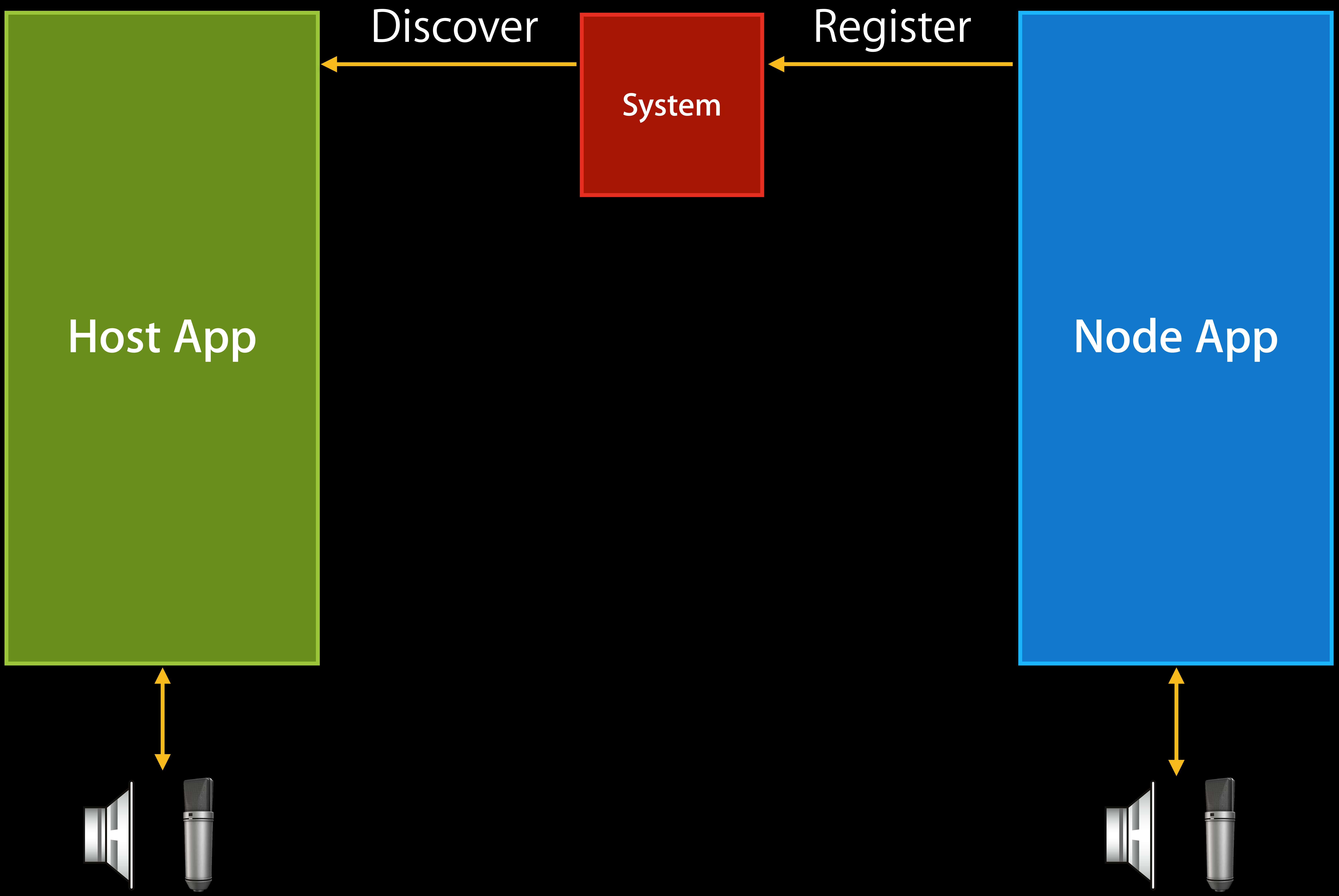
Inter-App Audio in Detail

Doug Wyatt
Core Audio Plumber

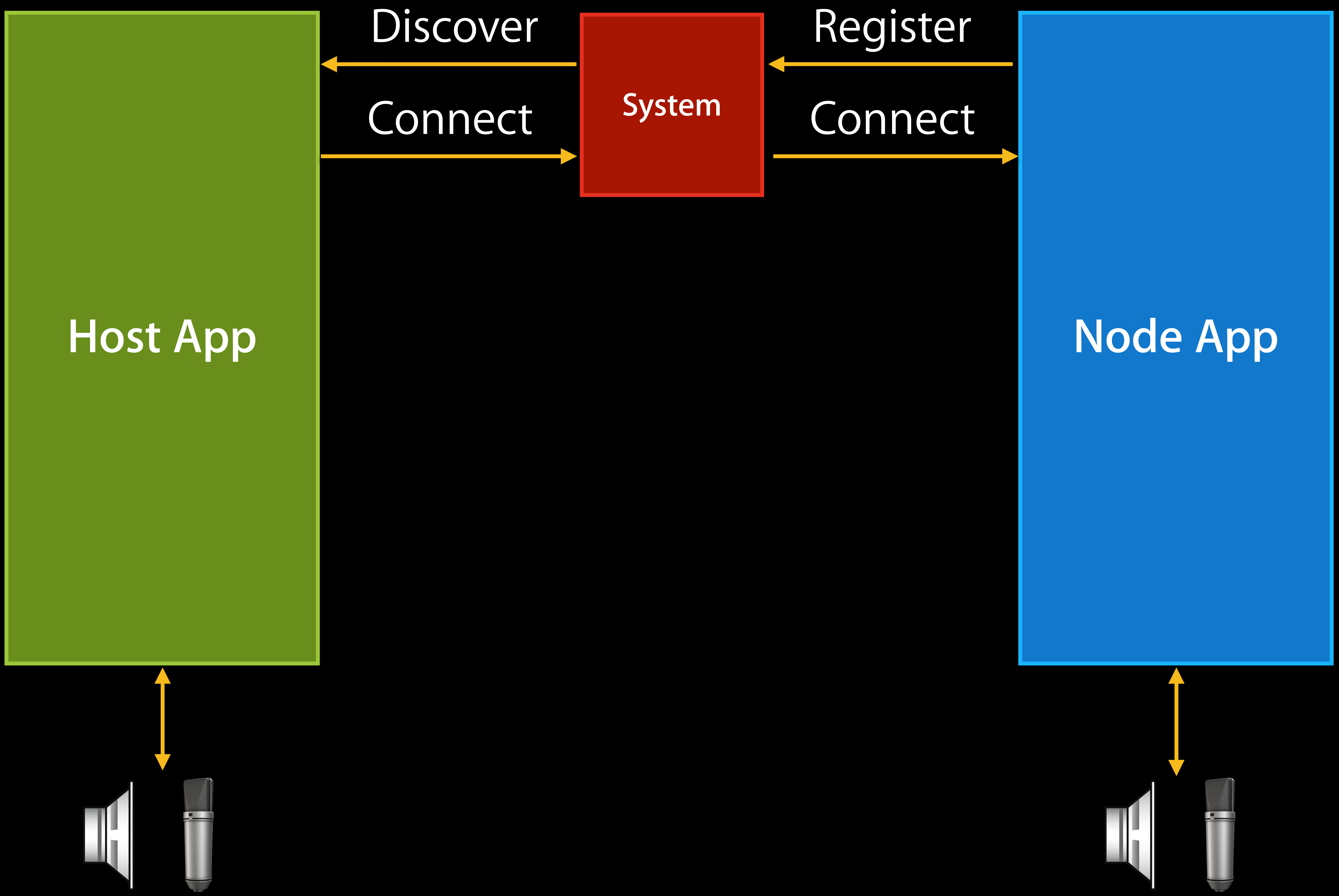
API Overview



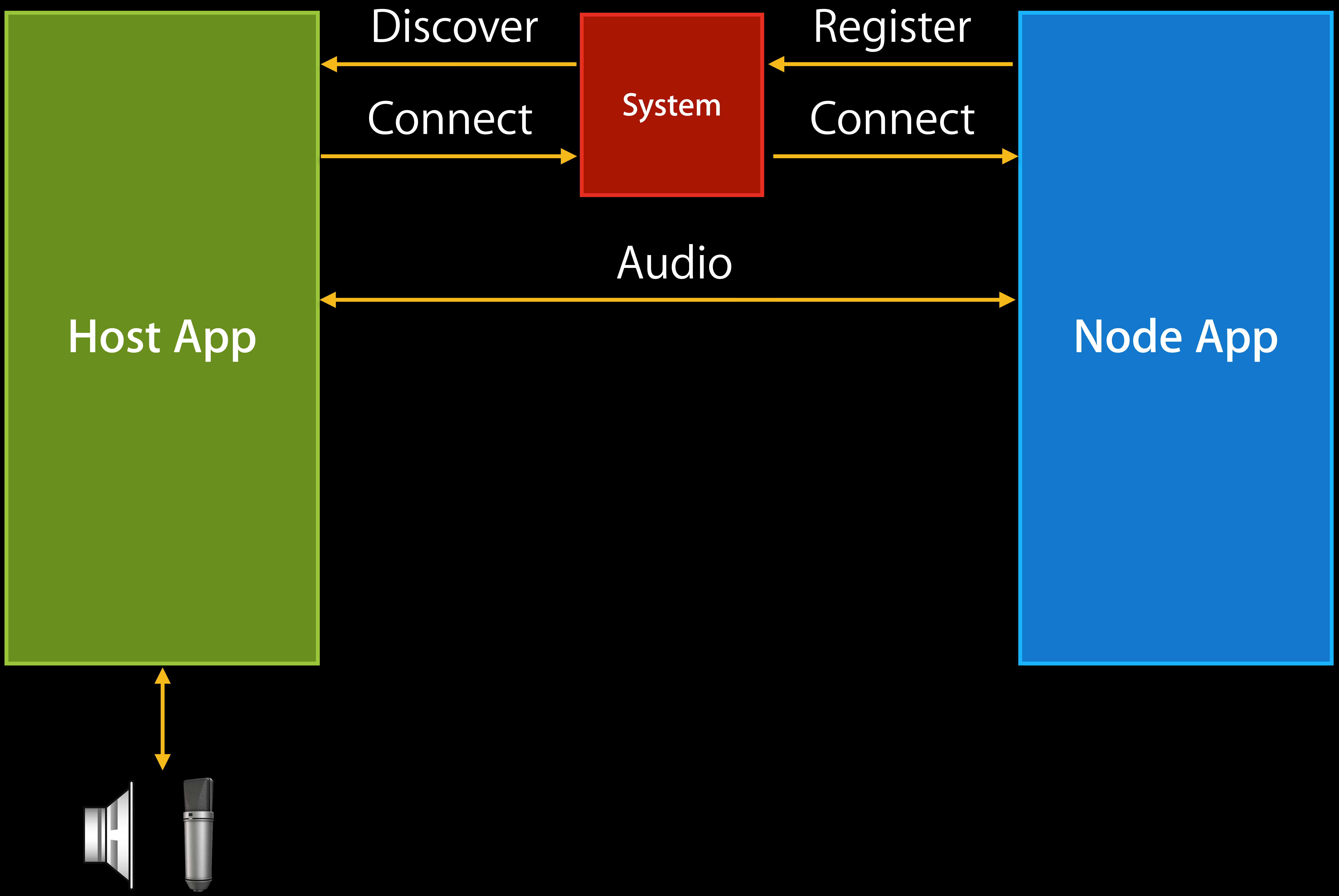
API Overview



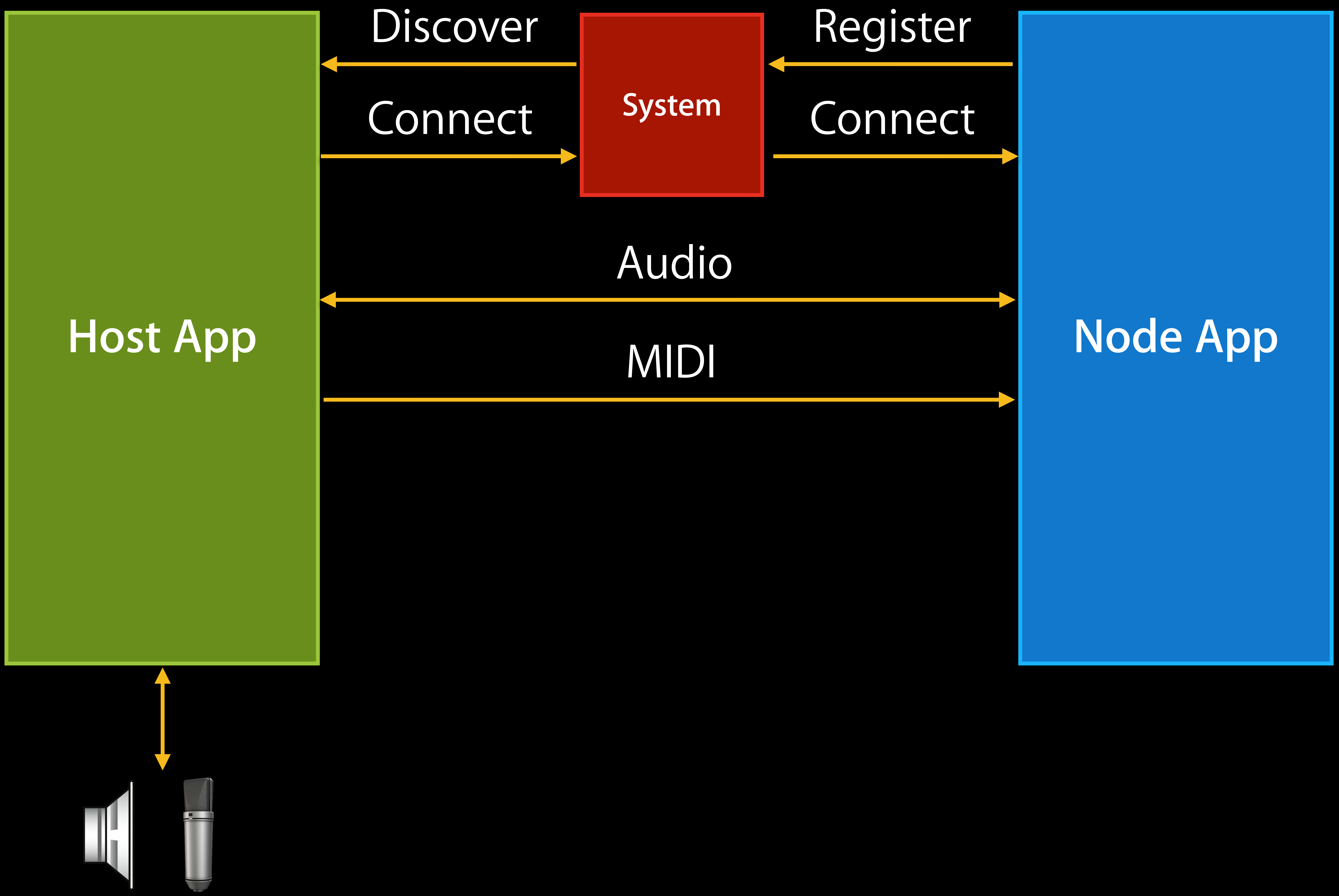
API Overview



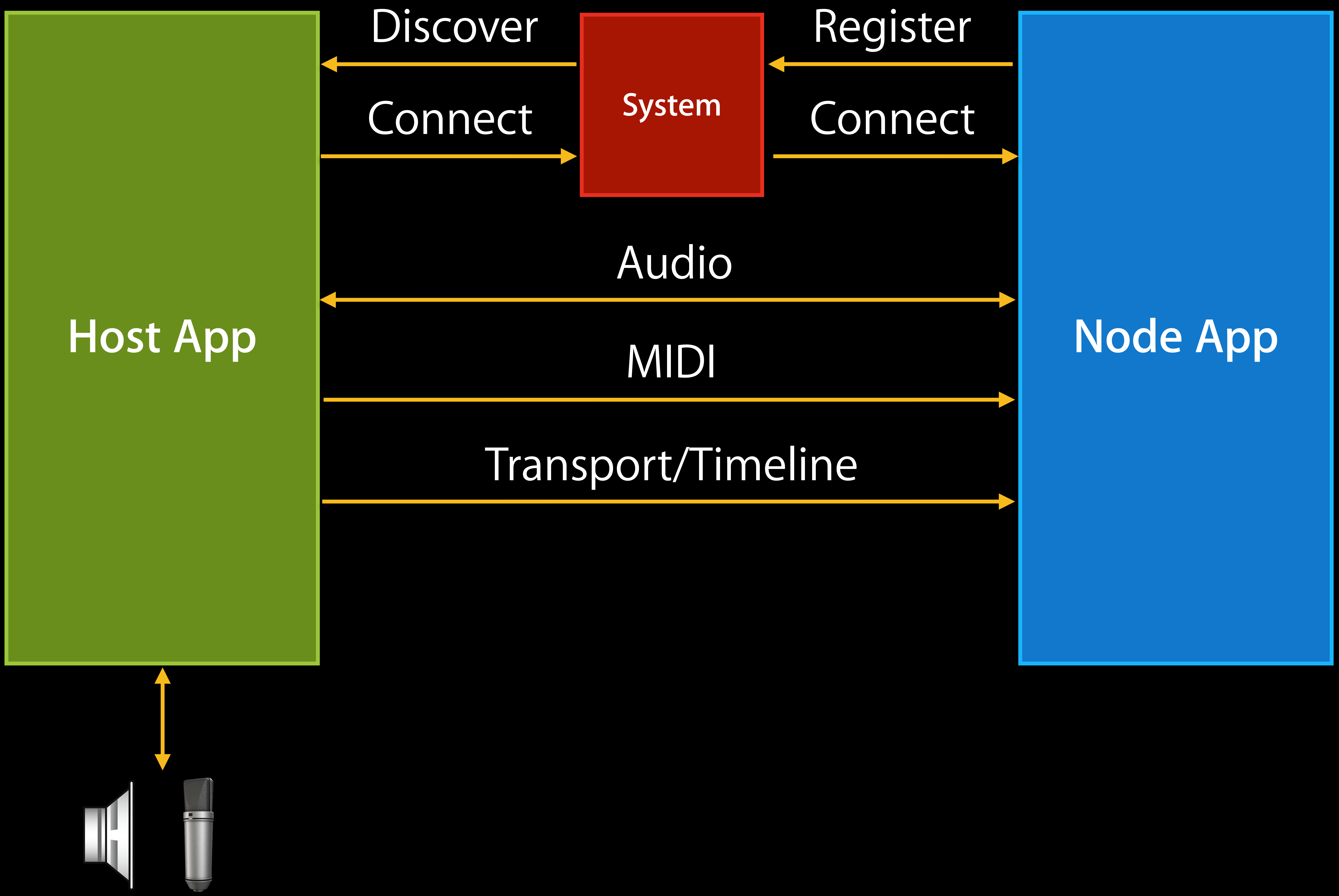
API Overview



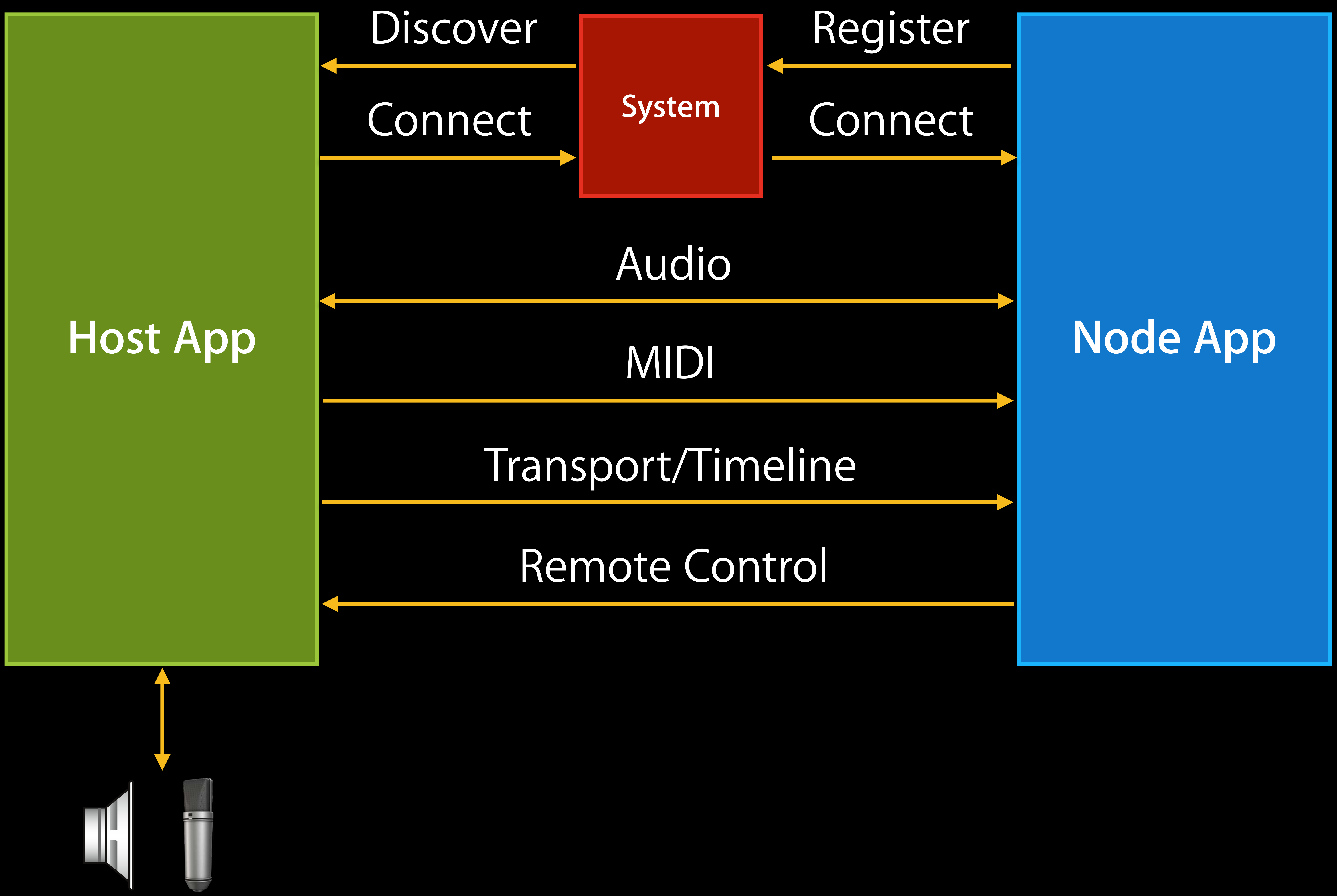
API Overview



API Overview

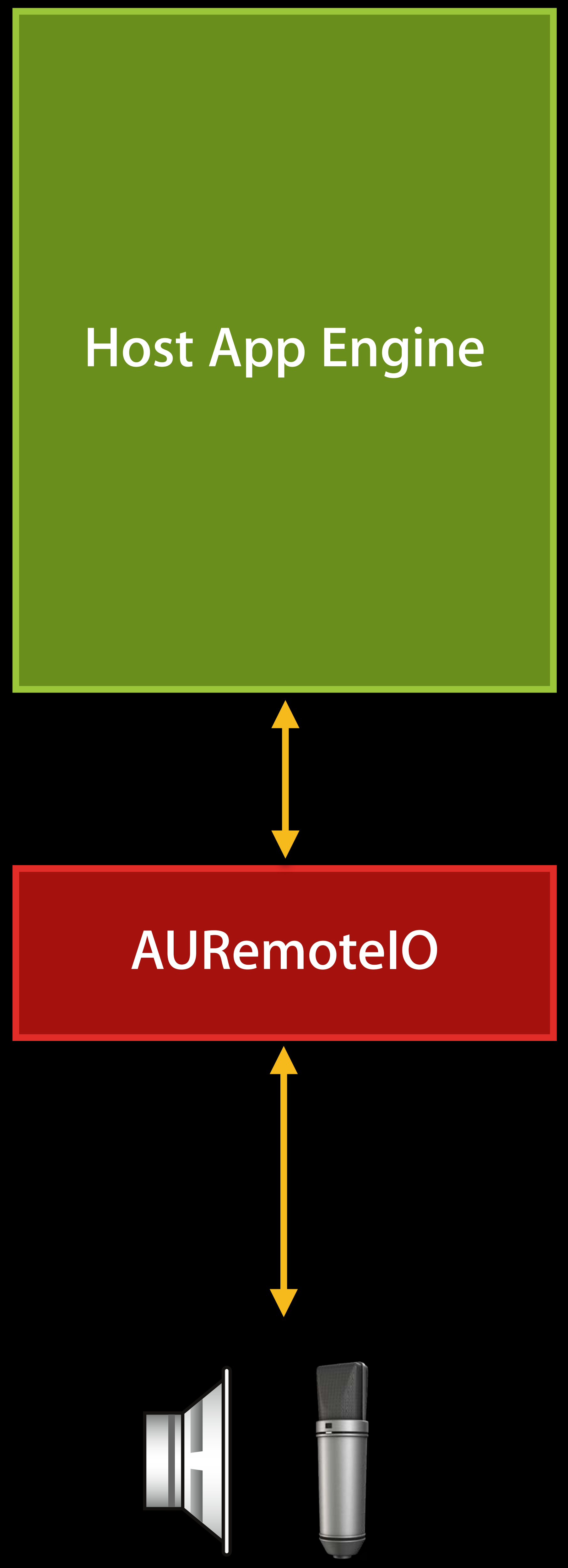


API Overview



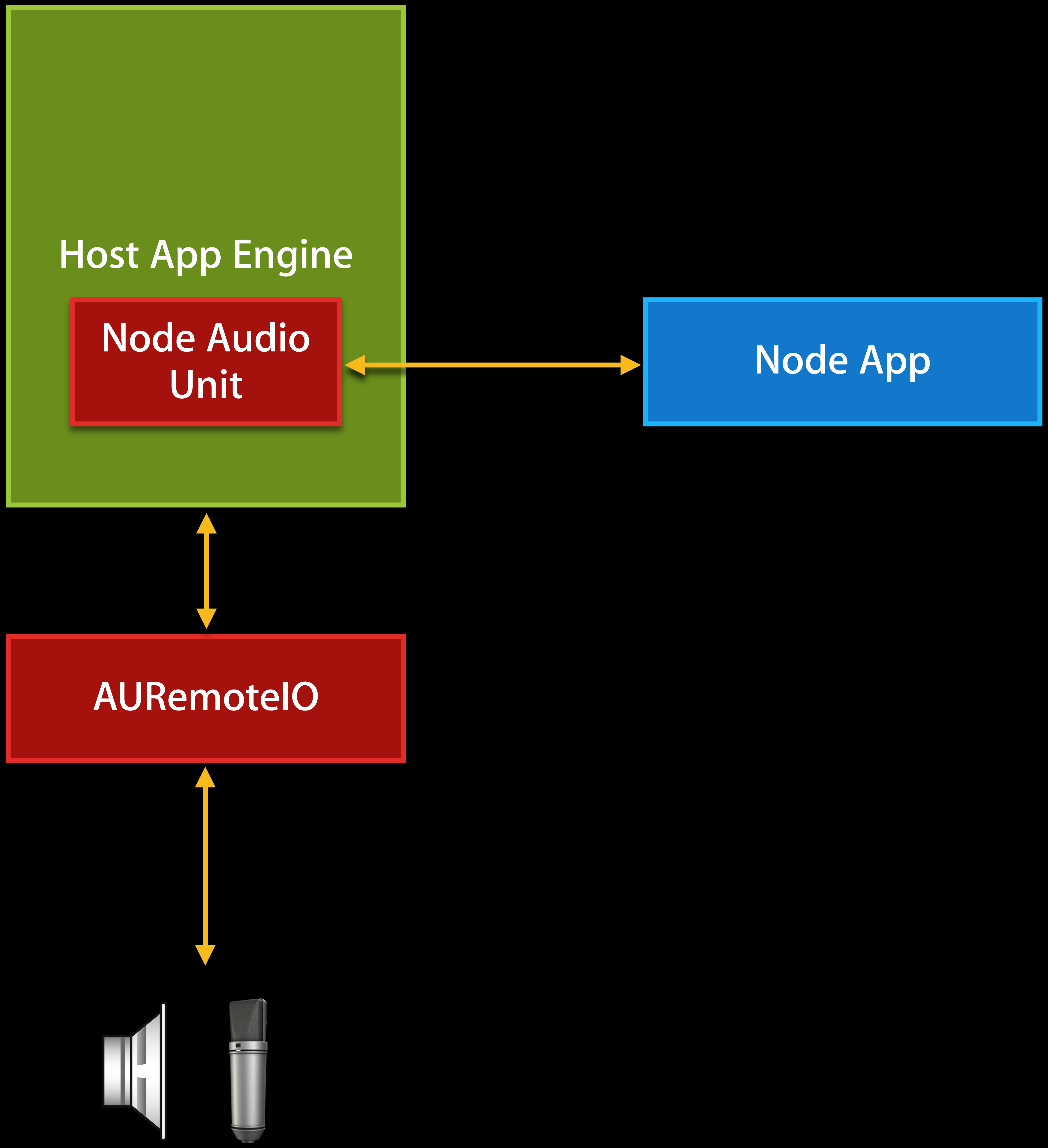
Introduction

Host app



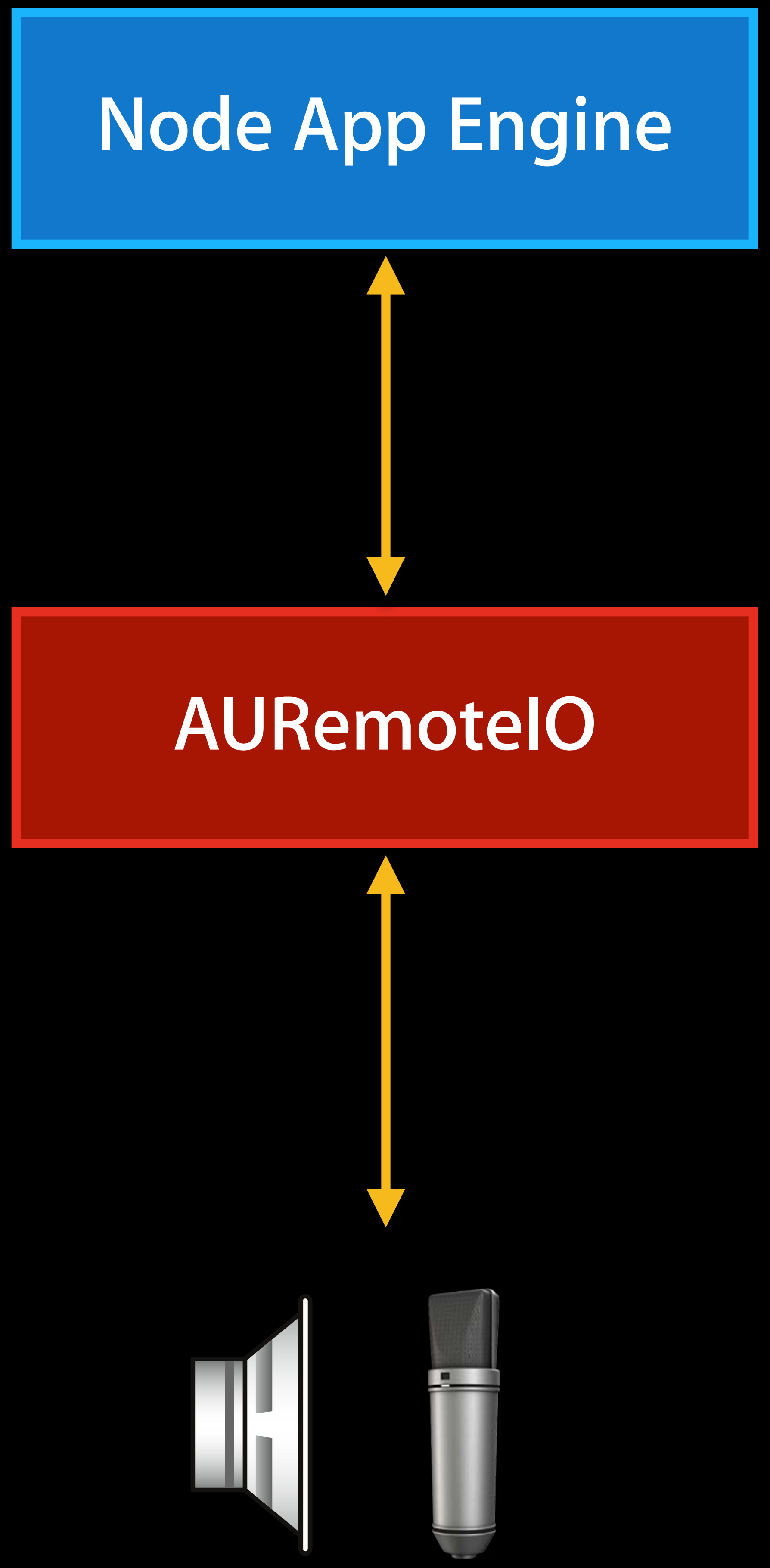
Introduction

Host app



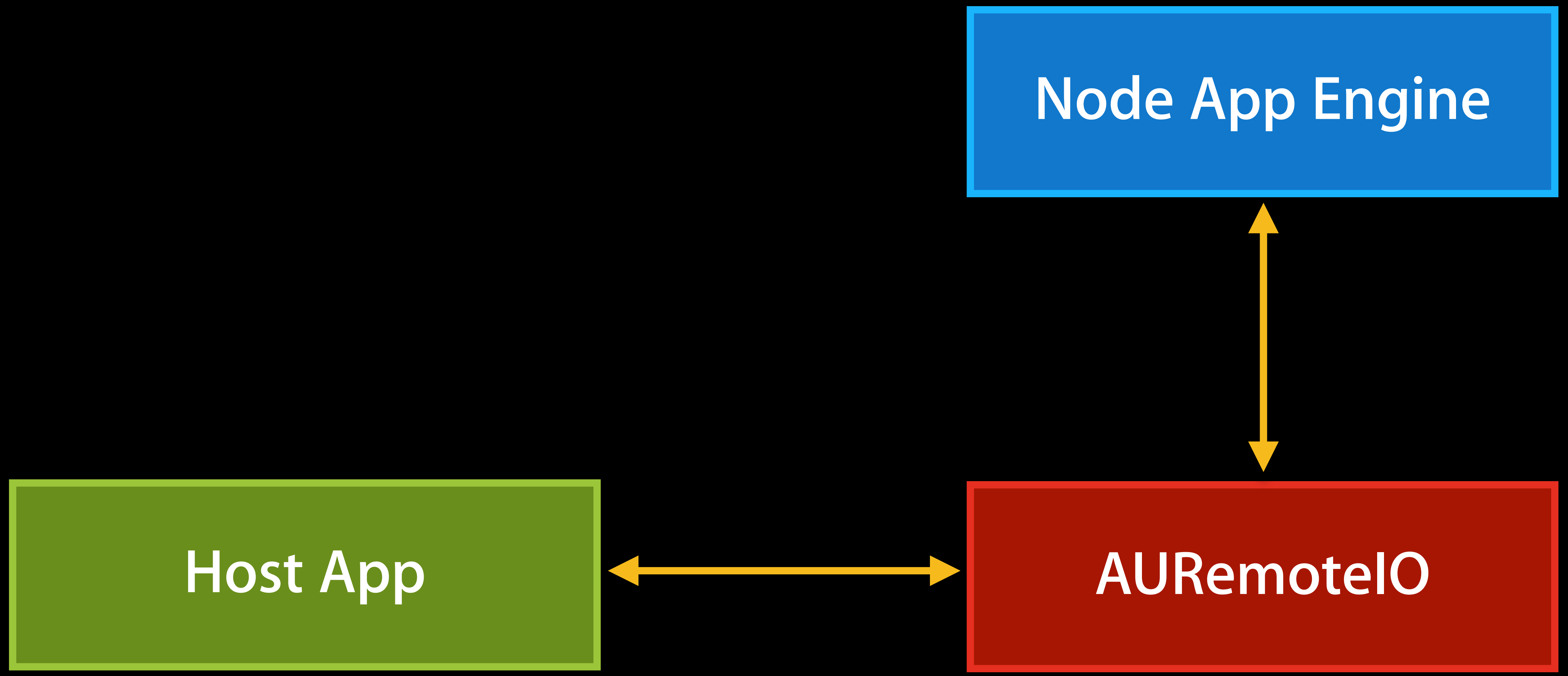
Introduction

Node app



Introduction

Node app



Introduction

- Extensions to AudioUnit.framework API
- Host sees node as AudioUnit
- Node’s I/O unit redirected to host

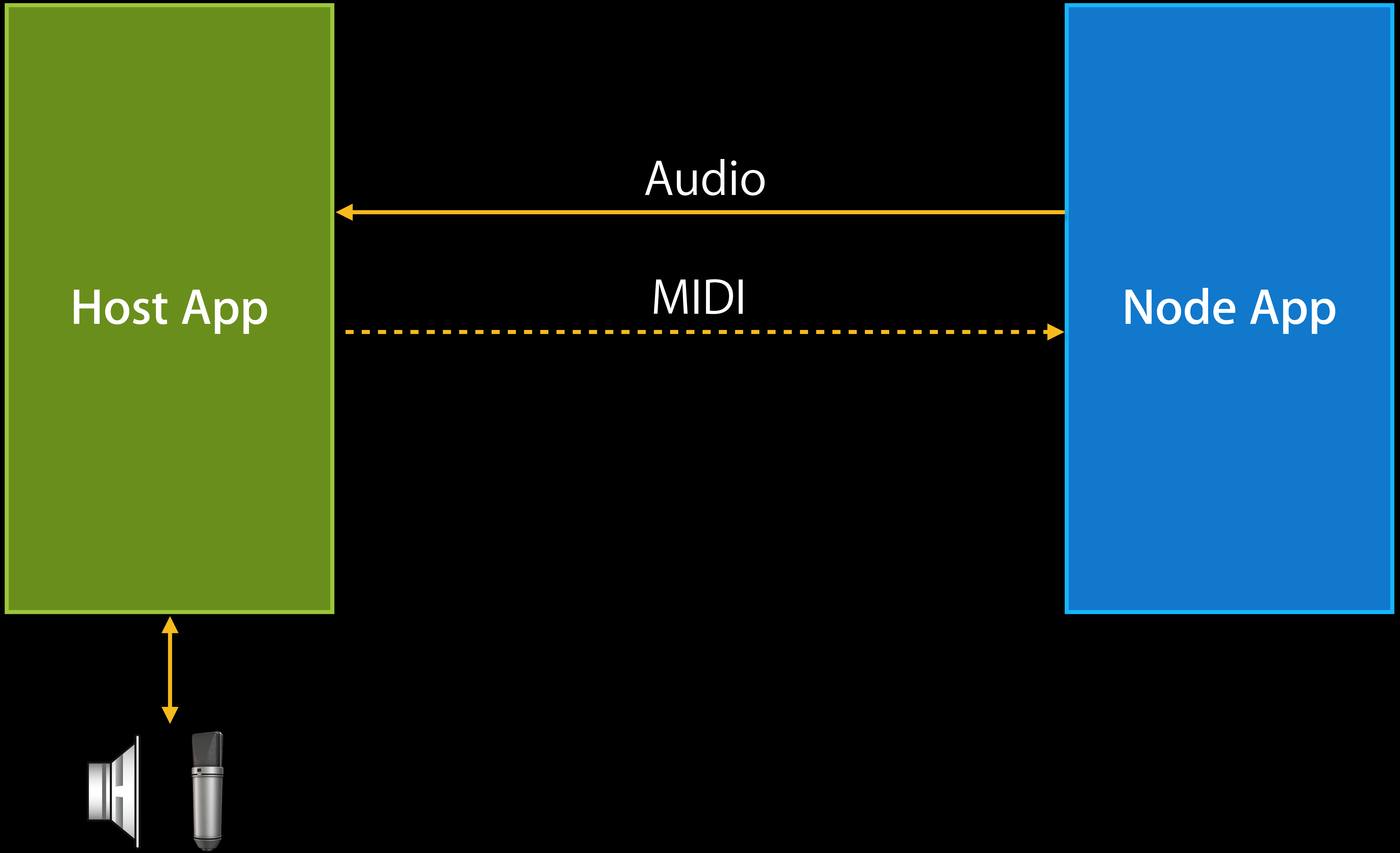
Introduction

New AudioUnit types

Audio Component Type	Input (from Host)	Output
kAudioUnitType_RemoteGenerator	-	audio
kAudioUnitType_RemoteInstrument	MIDI	audio
kAudioUnitType_RemoteEffect	audio	audio
kAudioUnitType_RemoteMusicEffect	audio and MIDI	audio

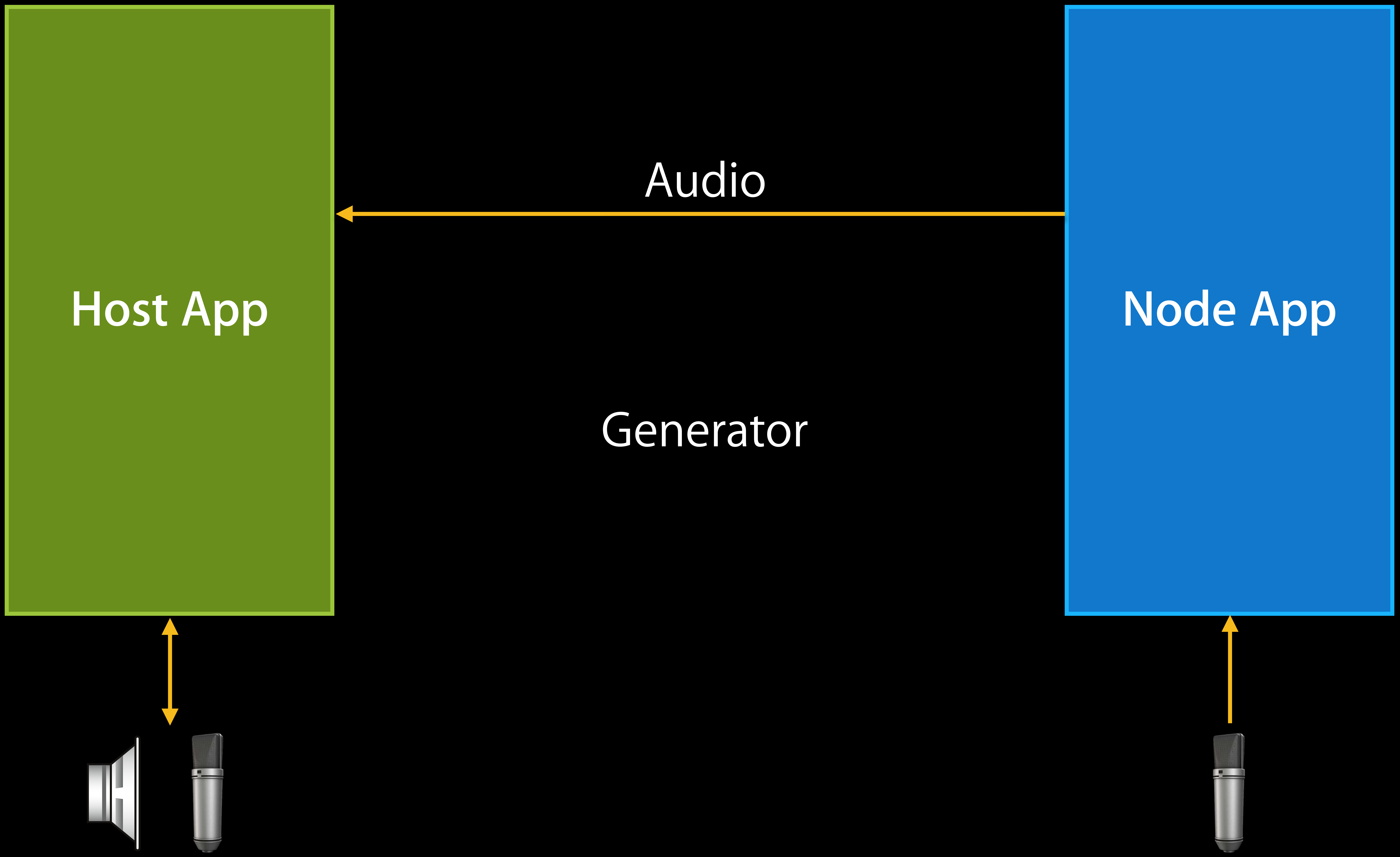
One Node App, Multiple Components

Generator or instrument



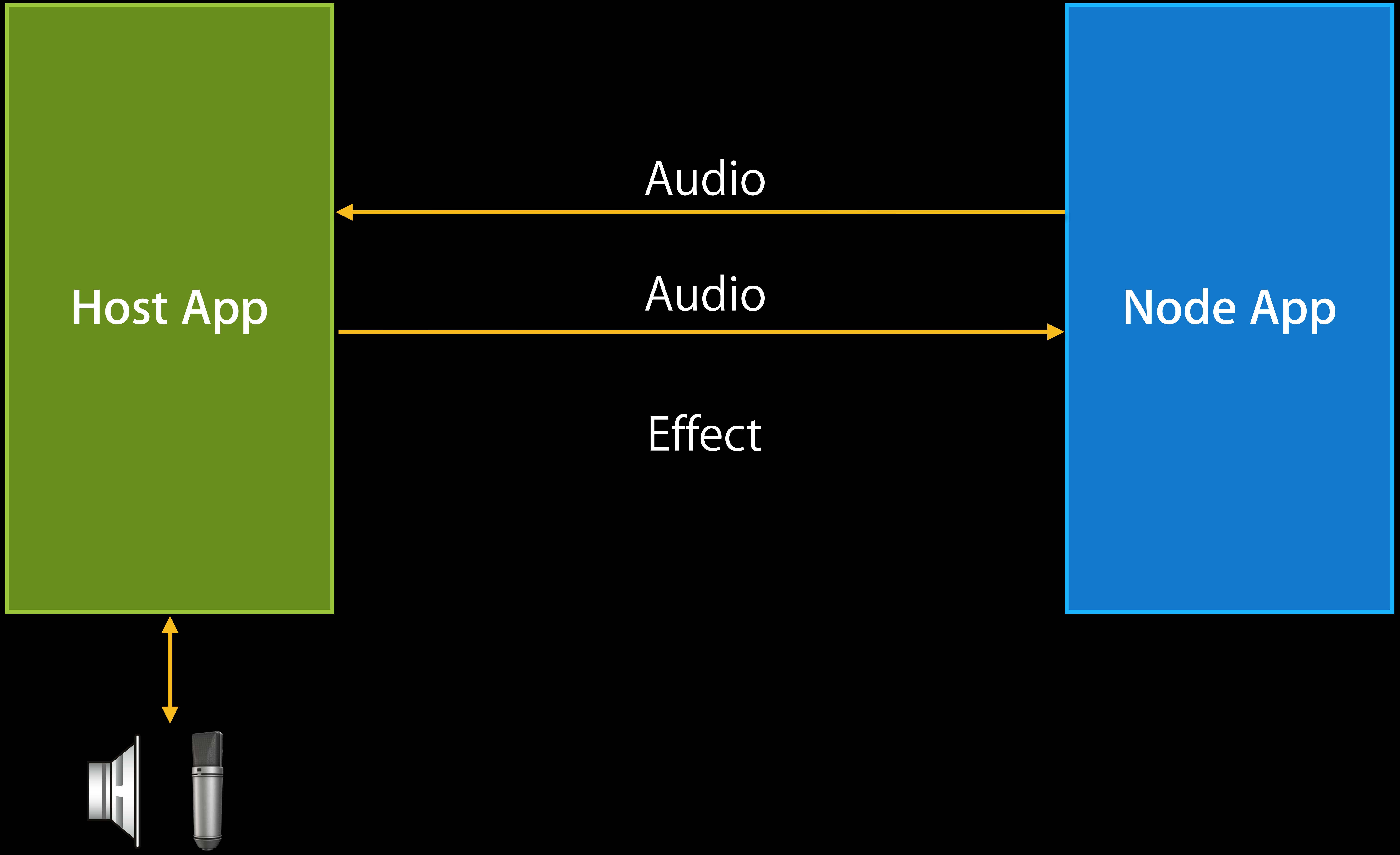
One Node App, Multiple Components

Generator or effect



One Node App, Multiple Components

Generator or effect

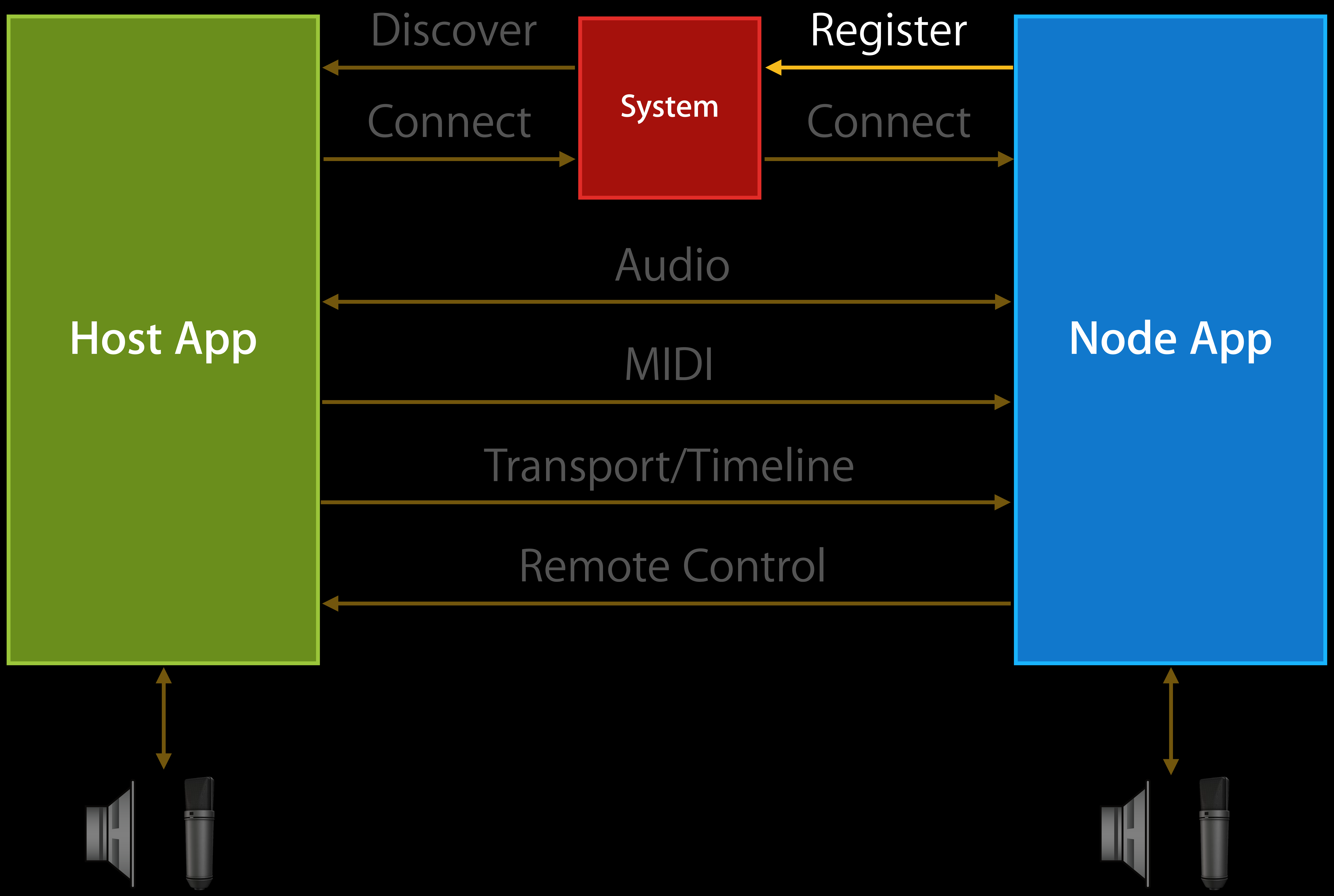


Introduction

Requirements

- Available on most iOS 7-compatible devices
    - (Except iPhone 4: API’s fail quietly)
- “inter-app-audio” entitlement (all)
- “audio” in UIBackgroundModes (hosts, nodes that access mic)
- AVAudioSessionCategoryOptionMixWithOthers (nodes)

Registering a Node App



Registering a Node App

- Register via Info.plist “AudioComponents”
    - Makes app launchable
- AudioOutputUnitPublish
    - “Checks in” the registration

Registering a Node App

AudioComponents entry in Info.plist

```
<key>AudioComponents</key>
<array>
    <dict>
        <key>type</key>
        <string>aurg</string>
        <key>subtype</key>
        <string>ACgn</string>
        <key>manufacturer</key>
        <string>ACME</string>
        <key>name</key>
        <string>Acme: SineGenerator</string>
        <key>version</key>
        <integer>1</integer>
    </dict>
</array>
```

Registering a Node App

Create and publish the I/O unit

```
  // create the AURemoteIO I/O unit

AudioComponentDescription ioUnitDesc = { kAudioUnitType_Output,
kAudioUnitSubType_RemoteIO, kAudioUnitManufacturer_Apple, 0, 0 };

AudioComponent comp = AudioComponentFindNext(NULL, &ioUnitDesc);

err = AudioComponentInstanceNew(comp, &_ioUnit);

// publish the AURemoteIO

AudioComponentDescription generatorDesc = { kAudioUnitType_RemoteGenerator,
‘ACgn’, ‘ACME’, 0, 0 };

CFStringRef name = CFSTR(“Acme: SineGenerator”);

const UInt32 version = 1;

err = AudioOutputUnitPublish(&generatorDesc, name, version, _ioUnit);
```

Registering a Node App

Create and publish the I/O unit

```
  // create the AURemoteIO I/O unit

AudioComponentDescription ioUnitDesc = { kAudioUnitType_Output,
kAudioUnitSubType_RemoteIO, kAudioUnitManufacturer_Apple, 0, 0 };

AudioComponent comp = AudioComponentFindNext(NULL, &ioUnitDesc);

err = AudioComponentInstanceNew(comp, &_ioUnit);

// publish the AURemoteIO

AudioComponentDescription generatorDesc = { kAudioUnitType_RemoteGenerator,
‘ACgn’, ‘ACME’, 0, 0 };

CFStringRef name = CFSTR(“Acme: SineGenerator”);

const UInt32 version = 1;

err = AudioOutputUnitPublish(&generatorDesc, name, version, _ioUnit);
```

Registering a Node App

Create and publish the I/O unit

```
  // create the AURemoteIO I/O unit

AudioComponentDescription ioUnitDesc = { kAudioUnitType_Output,
kAudioUnitSubType_RemoteIO, kAudioUnitManufacturer_Apple, 0, 0 };

AudioComponent comp = AudioComponentFindNext(NULL, &ioUnitDesc);

err = AudioComponentInstanceNew(comp, &_ioUnit);
```

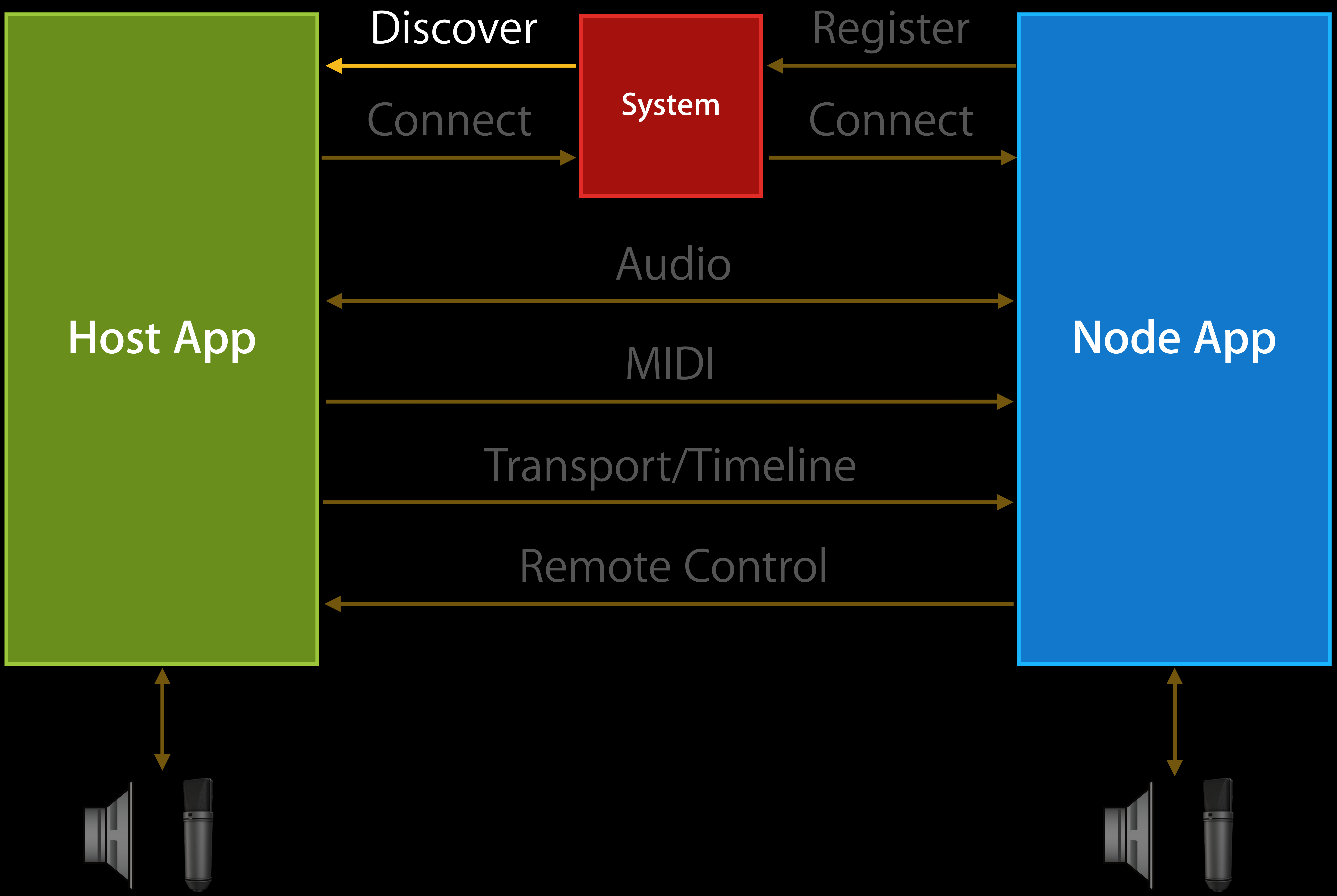
```
// publish the AURemoteIO

AudioComponentDescription generatorDesc = { kAudioUnitType_RemoteGenerator,
‘ACgn’, ‘ACME’, 0, 0 };

CFStringRef name = CFSTR(“Acme: SineGenerator”);

const UInt32 version = 1;

err = AudioOutputUnitPublish(&generatorDesc, name, version, _ioUnit);
```

Registering a Node App

Publishing the I/O unit

- Must publish on launch
- Component descriptions, names, and versions must match
- Component name should be “Manufacturer: App name”

Node App Discovery (for Hosts)



Node App Discovery

For hosts

```
AudioComponentDescription searchDesc = { 0, 0, 0, 0, 0 }, foundDesc;
AudioComponent comp = NULL;

while (true) {
   comp = AudioComponentFindNext(comp, &searchDesc);
   if (comp == NULL) break;

   if (AudioComponentGetDescription(comp, &foundDesc) != noErr) continue;

   switch (foundDesc.componentType) {
      case kAudioUnitType_RemoteEffect:
      case kAudioUnitType_RemoteGenerator:
      case kAudioUnitType_RemoteInstrument:
      case kAudioUnitType_RemoteMusicEffect:
      // found a node
      ...
      break;
   }
}
```

Node App Discovery

For hosts

```
AudioComponentDescription searchDesc = { 0, 0, 0, 0, 0 }, foundDesc;
AudioComponent comp = NULL;

while (true) {
   comp = AudioComponentFindNext(comp, &searchDesc);
   if (comp == NULL) break;

   if (AudioComponentGetDescription(comp, &foundDesc) != noErr) continue;

   switch (foundDesc.componentType) {
      case kAudioUnitType_RemoteEffect:
      case kAudioUnitType_RemoteGenerator:
      case kAudioUnitType_RemoteInstrument:
      case kAudioUnitType_RemoteMusicEffect:
      // found a node
      ...
      break;
   }
}
```

Node App Discovery

For hosts

```
AudioComponentDescription searchDesc = { 0, 0, 0, 0, 0 }, foundDesc;
AudioComponent comp = NULL;

while (true) {
   comp = AudioComponentFindNext(comp, &searchDesc);
   if (comp == NULL) break;

   if (AudioComponentGetDescription(comp, &foundDesc) != noErr) continue;

   switch (foundDesc.componentType) {
      case kAudioUnitType_RemoteEffect:
      case kAudioUnitType_RemoteGenerator:
      case kAudioUnitType_RemoteInstrument:
      case kAudioUnitType_RemoteMusicEffect:
      // found a node
      ...
      break;
   }
}
```

Node App Discovery

Getting node information

```
RemoteAU *rau = [[RemoteAU alloc] init];
rau->_desc = foundDesc;
rau->_comp = comp;

AudioComponentCopyName(comp, (CFStringRef *)&rau->_name);

rau->_image = [AudioComponentGetIcon(comp, 48) retain];

rau->_lastActiveTime = AudioComponentGetLastActiveTime(comp);

[_audioUnits addObject: rau];
```

Node App Discovery

Getting node information

```
RemoteAU *rau = [[RemoteAU alloc] init];
rau->_desc = foundDesc;
rau->_comp = comp;

AudioComponentCopyName(comp, (CFStringRef *)&rau->_name);

rau->_image = [AudioComponentGetIcon(comp, 48) retain];

rau->_lastActiveTime = AudioComponentGetLastActiveTime(comp);

[_audioUnits addObject: rau];
```

Node App Discovery

Getting node information

```
RemoteAU *rau = [[RemoteAU alloc] init];
rau->_desc = foundDesc;
rau->_comp = comp;

AudioComponentCopyName(comp, (CFStringRef *)&rau->_name);

rau->_image = [AudioComponentGetIcon(comp, 48) retain];

rau->_lastActiveTime = AudioComponentGetLastActiveTime(comp);

[_audioUnits addObject: rau];
```

Node App Discovery

Getting node information

```
RemoteAU *rau = [[RemoteAU alloc] init];
rau->_desc = foundDesc;
rau->_comp = comp;

AudioComponentCopyName(comp, (CFStringRef *)&rau->_name);

rau->_image = [AudioComponentGetIcon(comp, 48) retain];

rau->_lastActiveTime = AudioComponentGetLastActiveTime(comp);

[_audioUnits addObject: rau];
```

Node App Discovery

Getting node information

```
RemoteAU *rau = [[RemoteAU alloc] init];
rau->_desc = foundDesc;
rau->_comp = comp;

AudioComponentCopyName(comp, (CFStringRef *)&rau->_name);

rau->_image = [AudioComponentGetIcon(comp, 48) retain];

rau->_lastActiveTime = AudioComponentGetLastActiveTime(comp);

[_audioUnits addObject: rau];
```

Node App Discovery

Getting node information

```
RemoteAU *rau = [[RemoteAU alloc] init];
rau->_desc = foundDesc;
rau->_comp = comp;

AudioComponentCopyName(comp, (CFStringRef *)&rau->_name);

rau->_image = [AudioComponentGetIcon(comp, 48) retain];

rau->_lastActiveTime = AudioComponentGetLastActiveTime(comp);

[_audioUnits addObject: rau];
```

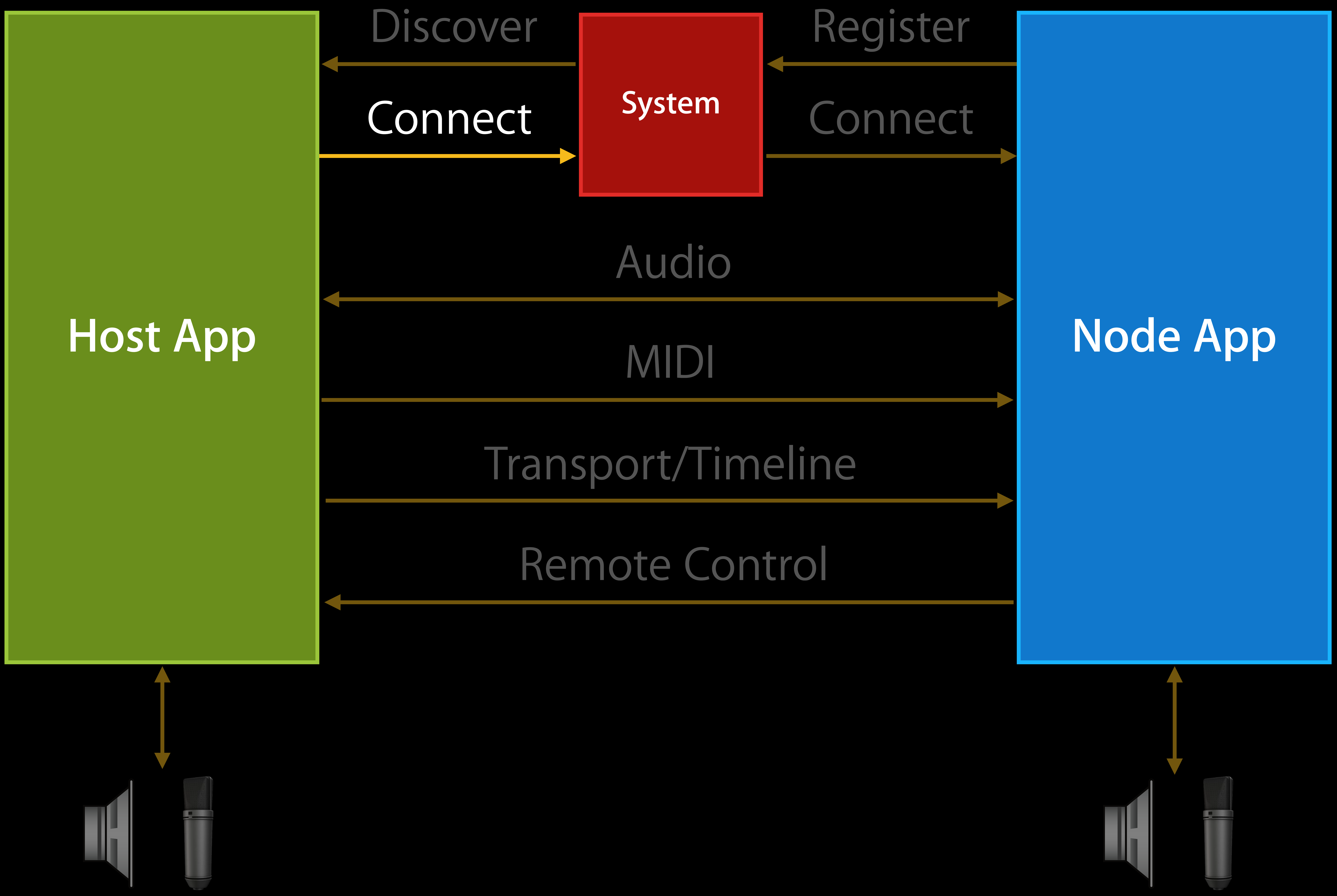
Node App Discovery

Observing registration changes

- Registrations change dynamically
  - When apps installed/deleted
  - When media services reset

```
NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
[nc addObserverForName:
  (NSString *) kAudioComponentRegistrationsChangedNotification
  object: nil queue: nil
  usingBlock:
    ^(NSNotification *) {
      [self refreshAUList];
    }
];
```

Connecting to a Node



Connecting to a Node

```
AudioUnit myAudioUnit;
err = AudioComponentInstanceNew(comp, &myAudioUnit);
```

- Node app will be launched into background

Preparing a Node
Activate audio session

```
[[AVAudioSession sharedSession] setActive: YES];
```

- Sample rate, channel count

Preparing a Node

Set stream formats

```
AudioStreamBasicDescription format;

format.mChannelsPerFrame = 2; // stereo
format.mSampleRate = [AVAudioSession sharedSession].sampleRate;
format.mFormatID = kAudioFormatLinearPCM;
format.mFormatFlags = kAudioFormatFlagsNativeFloatPacked |
                      kAudioFormatFlagIsNonInterleaved;
format.mBytesPerFrame = format.mBytesPerPacket = sizeof(Float32);
format.mBitsPerChannel = 32;
format.mFramesPerPacket = 1;

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Output, 0, &format, sizeof(format));

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input, 0, &format, sizeof(format));
```

Preparing a Node

Set stream formats

```
AudioStreamBasicDescription format;

format.mChannelsPerFrame = 2; // stereo
format.mSampleRate = [AVAudioSession sharedSession].sampleRate;
format.mFormatID = kAudioFormatLinearPCM;
format.mFormatFlags = kAudioFormatFlagsNativeFloatPacked |
                      kAudioFormatFlagIsNonInterleaved;
format.mBytesPerFrame = format.mBytesPerPacket = sizeof(Float32);
format.mBitsPerChannel = 32;
format.mFramesPerPacket = 1;

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Output, 0, &format, sizeof(format));

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input, 0, &format, sizeof(format));
```

Preparing a Node

Set stream formats

```
AudioStreamBasicDescription format;

format.mChannelsPerFrame = 2; // stereo
format.mSampleRate = [AVAudioSession sharedSession].sampleRate;
format.mFormatID = kAudioFormatLinearPCM;
format.mFormatFlags = kAudioFormatFlagsNativeFloatPacked |
                      kAudioFormatFlagIsNonInterleaved;
format.mBytesPerFrame = format.mBytesPerPacket = sizeof(Float32);
format.mBitsPerChannel = 32;
format.mFramesPerPacket = 1;

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Output, 0, &format, sizeof(format));

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input, 0, &format, sizeof(format));
```

Preparing a Node

Set stream formats

```
AudioStreamBasicDescription format;

format.mChannelsPerFrame = 2; // stereo
format.mSampleRate = [AVAudioSession sharedSession].sampleRate;
format.mFormatID = kAudioFormatLinearPCM;
format.mFormatFlags = kAudioFormatFlagsNativeFloatPacked |
                      kAudioFormatFlagIsNonInterleaved;
format.mBytesPerFrame = format.mBytesPerPacket = sizeof(Float32);
format.mBitsPerChannel = 32;
format.mFramesPerPacket = 1;

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Output, 0, &format, sizeof(format));

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input, 0, &format, sizeof(format));
```

Preparing a Node

Set stream formats

```
AudioStreamBasicDescription format;

format.mChannelsPerFrame = 2; // stereo
format.mSampleRate = [AVAudioSession sharedSession].sampleRate;
format.mFormatID = kAudioFormatLinearPCM;
format.mFormatFlags = kAudioFormatFlagsNativeFloatPacked |
                      kAudioFormatFlagIsNonInterleaved;
format.mBytesPerFrame = format.mBytesPerPacket = sizeof(Float32);
format.mBitsPerChannel = 32;
format.mFramesPerPacket = 1;

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Output, 0, &format, sizeof(format));

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input, 0, &format, sizeof(format));
```

Preparing a Node

Set stream formats

```
AudioStreamBasicDescription format;

format.mChannelsPerFrame = 2; // stereo
format.mSampleRate = [AVAudioSession sharedSession].sampleRate;
format.mFormatID = kAudioFormatLinearPCM;
format.mFormatFlags = kAudioFormatFlagsNativeFloatPacked |
                      kAudioFormatFlagIsNonInterleaved;
format.mBytesPerFrame = format.mBytesPerPacket = sizeof(Float32);
format.mBitsPerChannel = 32;
format.mFramesPerPacket = 1;

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Output, 0, &format, sizeof(format));

AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input, 0, &format, sizeof(format));
```

Preparing a Node

Connect input

- For effects
- From AudioUnit:
  - AUGraphConnectNodeInput
  - kAudioUnitProperty_MakeConnection
- From callback:
  - kAudioUnitProperty_SetRenderCallback
- Output

Preparing a Node

Disconnection

- Can happen automatically
  - Node terminates
  - Host fails to render
- Instance becomes “zombie”
- kAudioComponentErr_InstanceInvalidated

Preparing a Node

Disconnection callback

```
AudioUnitAddPropertyListener(myAudioUnit,
  kAudioUnitProperty_IsInterAppConnected,
  NodeConnectionListener, self);


void NodeConnectionListener(void *userData, AudioUnit myAudioUnit,
  AudioUnitPropertyID, AudioUnitScope, AudioUnitElement)
{
   UInt32 connected = 0, size = sizeof(connected);
   OSStatus err =
   AudioUnitGetProperty(myAudioUnit, kAudioUnitProperty_IsInterAppConnected,
     kAudioUnitScope_Global, 0,
     &connected, &size);
   if (err != noErr || connected == 0) {
      // Node disconnected
   }
}
```

Preparing a Node

Initialization

```
// Prepare for rendering
AudioUnitInitialize(myAudioUnit);
```

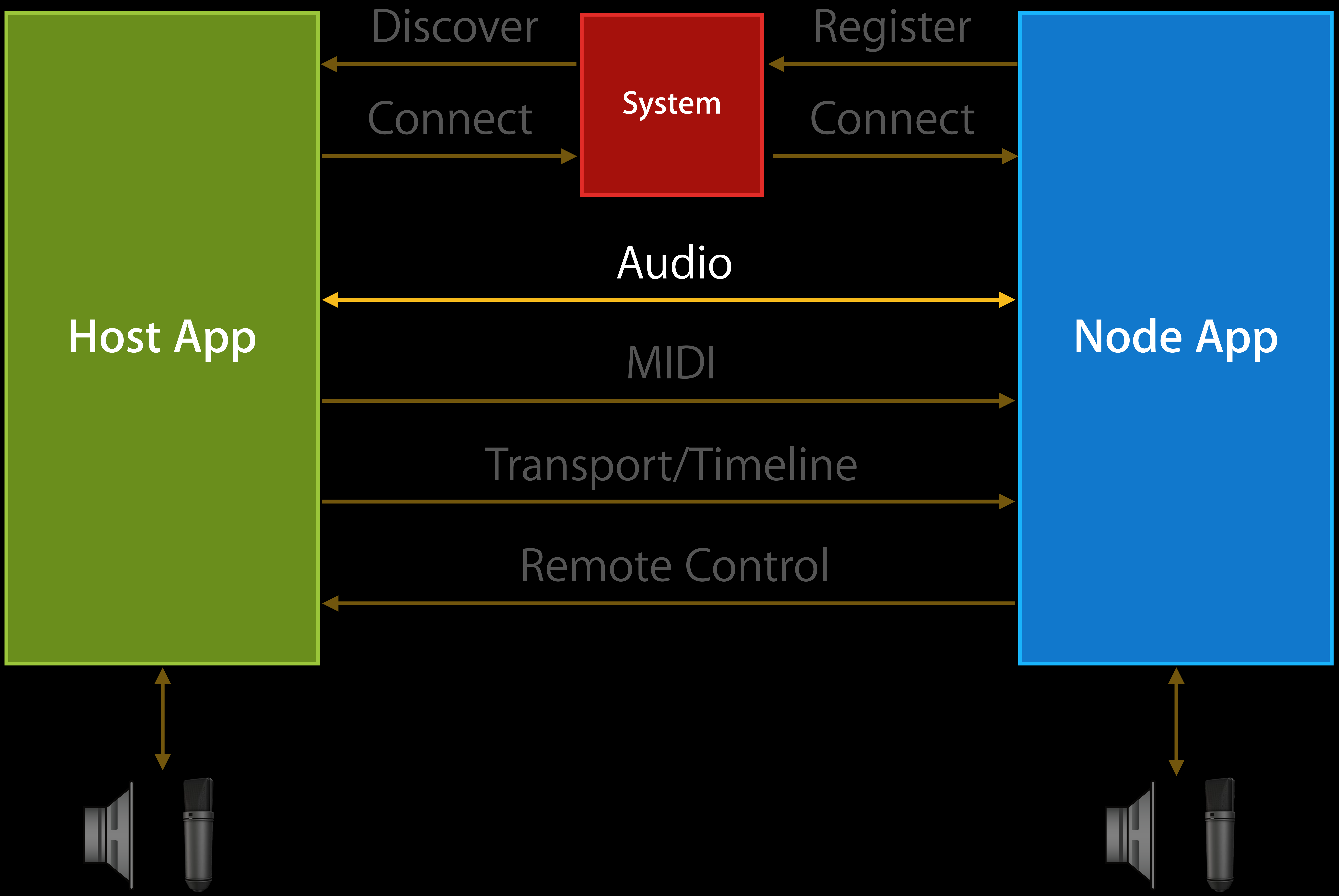
- Host must call AudioUnitRender regularly

Preparing a Node

Summary

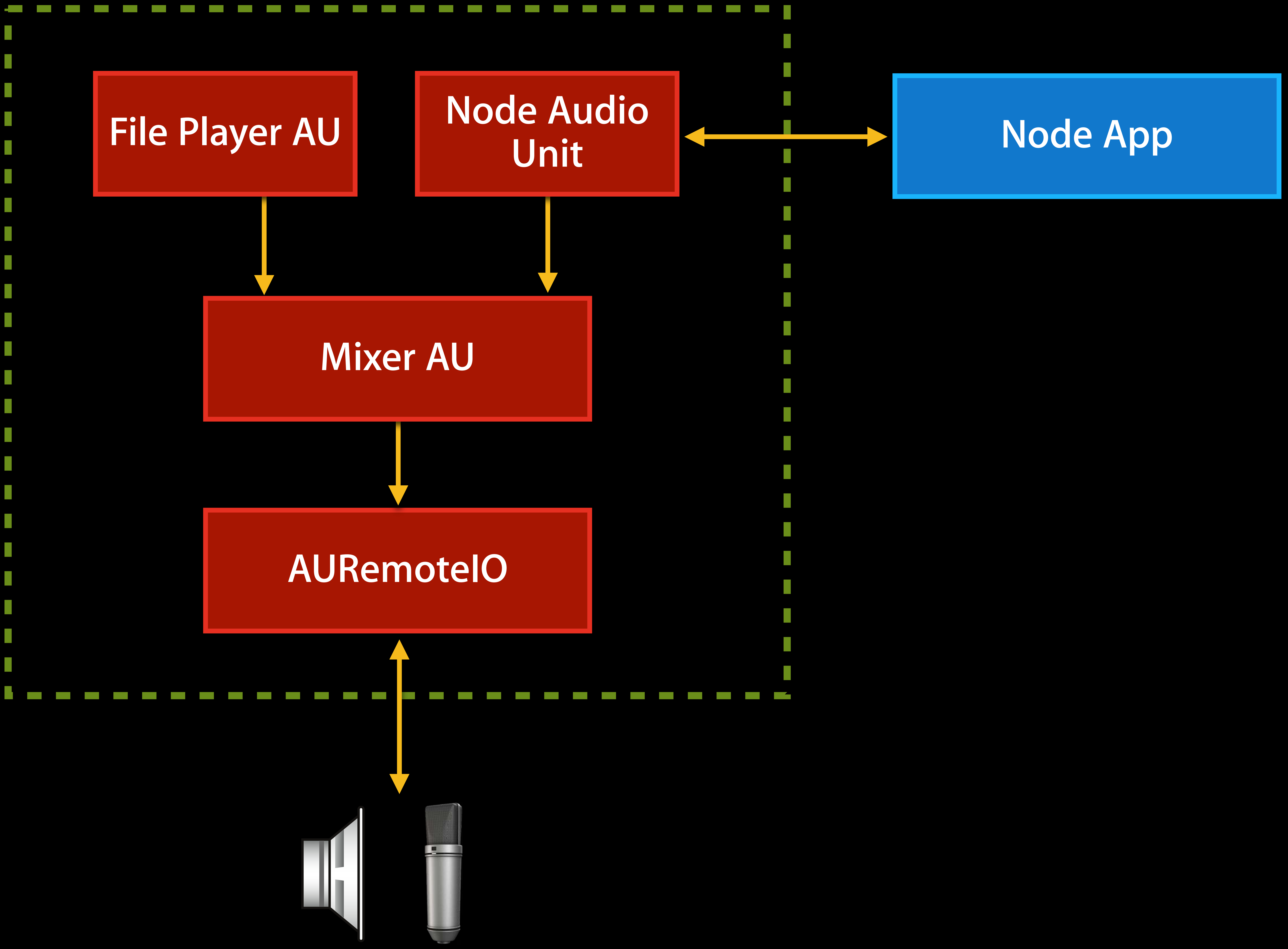
- Activate audio session
- Set stream formats
- Connect audio input
- Add disconnection listener
- Initialize

Host Rendering Node Audio



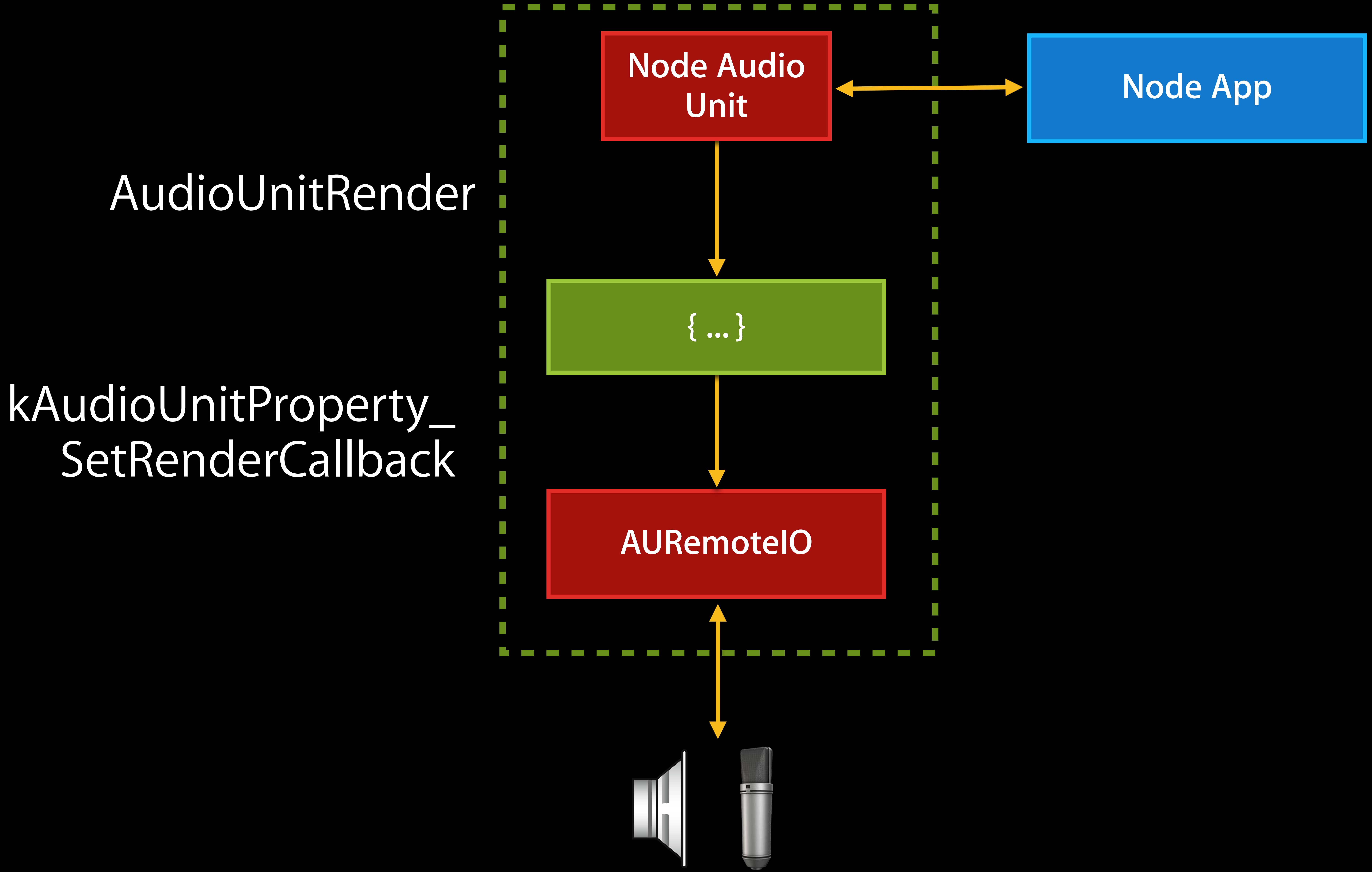
Host Rendering Node Audio

Using AUGraph



Host Rendering Node Audio

Using AudioUnits directly



Switching to a Node

- Do this after user opens the node
- Icon; tap to switch

```
void SwitchToNode(AudioUnit node)
{
    NSURL *url = NULL;
    UInt32 propertySize = sizeof(url);
    // property is CFURLRef but NSURL is toll-free-bridged.
    OSStatus err = AudioUnitGetProperty(peerAudioUnit,
  kAudioUnitProperty_PeerURL, kAudioUnitScope_Global, 0, &url,
  &propertySize);
    if (err) return;
    [[UIApplication sharedApplication] openUrl: url];
    [url release];
}
```

Switching to a Node

- Do this after user opens the node
- Icon; tap to switch

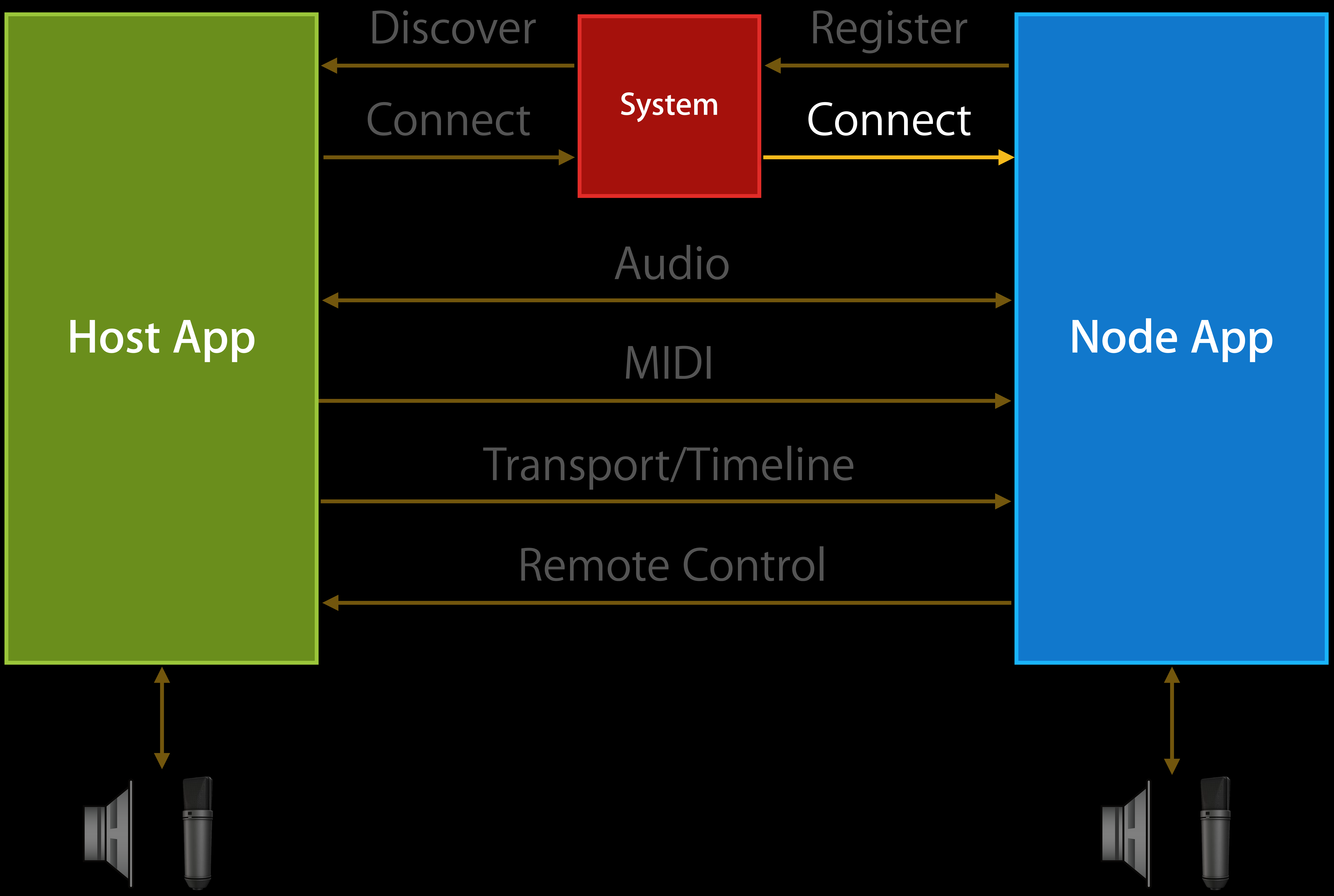
```
void SwitchToNode(AudioUnit node)
{
    NSURL *url = NULL;
    UInt32 propertySize = sizeof(url);
    // property is CFURLRef but NSURL is toll-free-bridged.
    OSStatus err = AudioUnitGetProperty(peerAudioUnit,
  kAudioUnitProperty_PeerURL, kAudioUnitScope_Global, 0, &url,
  &propertySize);
    if (err) return;
    [[UIApplication sharedApplication] openUrl: url];
    [url release];
}
```

Switching to a Node

- Do this after user opens the node
- Icon; tap to switch

```
void SwitchToNode(AudioUnit node)
{
    NSURL *url = NULL;
    UInt32 propertySize = sizeof(url);
    // property is CFURLRef but NSURL is toll-free-bridged.
    OSStatus err = AudioUnitGetProperty(peerAudioUnit,
  kAudioUnitProperty_PeerURL, kAudioUnitScope_Global, 0, &url,
  &propertySize);
    if (err) return;
    [[UIApplication sharedApplication] openUrl: url];
    [url release];
}
```

Being Connected (for Nodes)



Being Connected (for Nodes)

On launch

- Can be launched into the background
- Can’t start running from background
- Must publish I/O unit

```
UIApplicationState appstate =
    [UIApplication sharedApplication].applicationState;
_inForeground = (appstate != UIApplicationStateBackground);
```

Being Connected (for Nodes)

Handling connection

- kAudioUnitProperty_IsInterAppConnected
- When value becomes 1:
  - Output unit has been initialized
  - Set audio session active (if accessing mic)
  - Start running
  - Even if in the background
- AudioOutputUnitGetHostIcon

Being Connected (for Nodes)

Handling disconnection

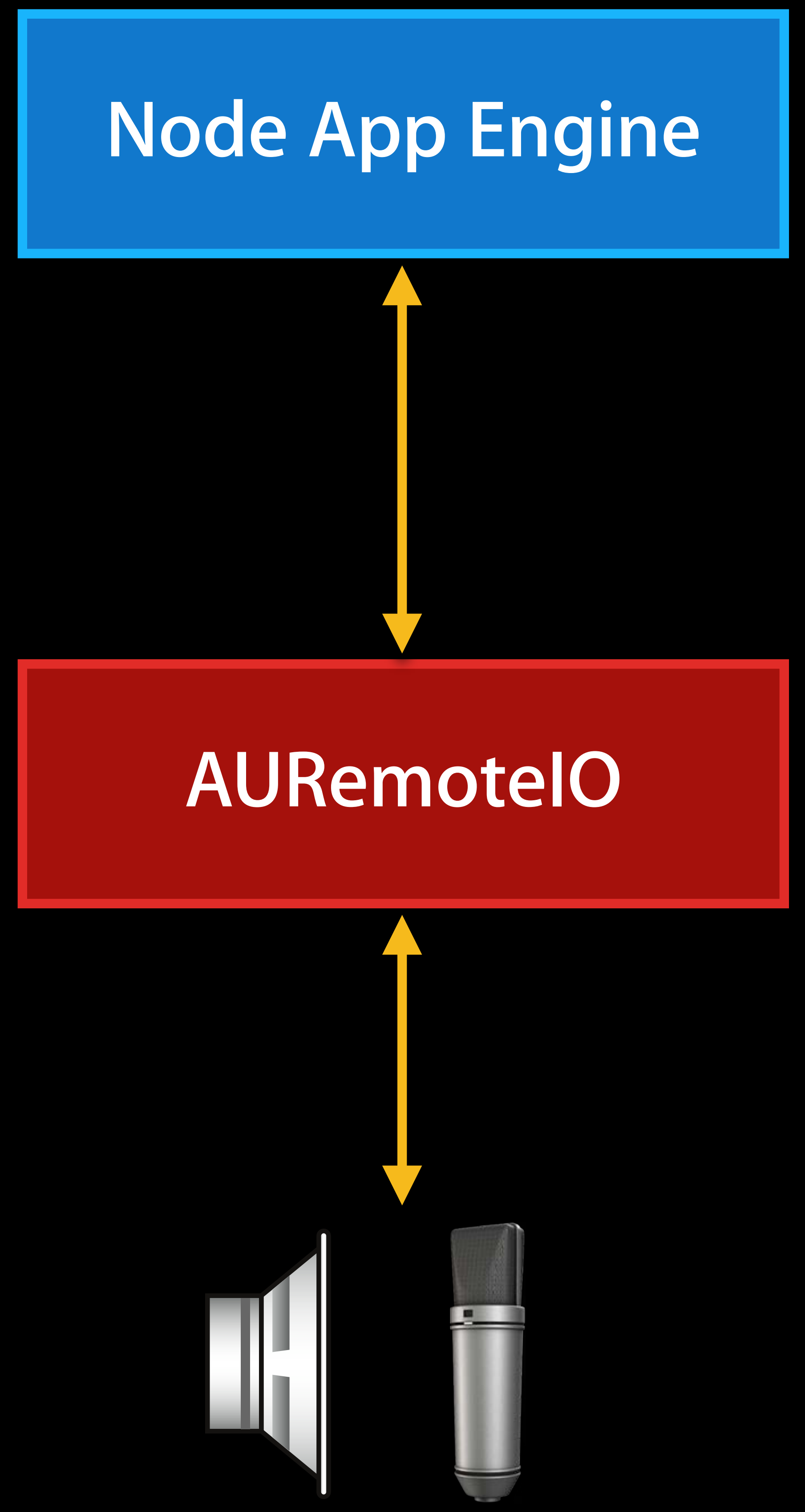
- When kAudioUnitProperty_IsInterAppConnected becomes 0:
    - Output unit has been uninitialized and stopped
    - Set audio session inactive (if using mic)
    - May set session active and resume running, but only if in the foreground

Running State (for Nodes)

- CanStart = Connected or InForeground
- Running = Connected or AppSpecificConditions

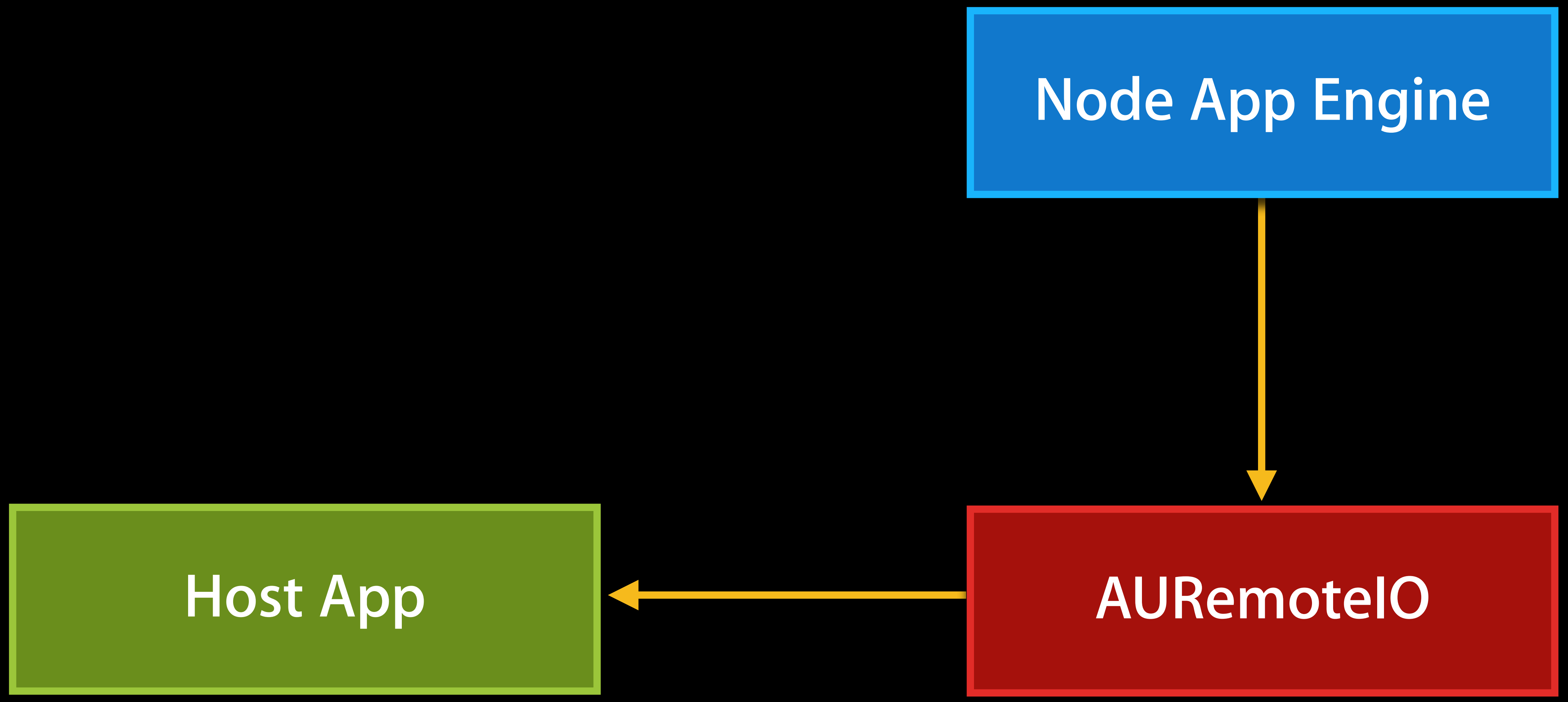
Node Rendering

Standalone



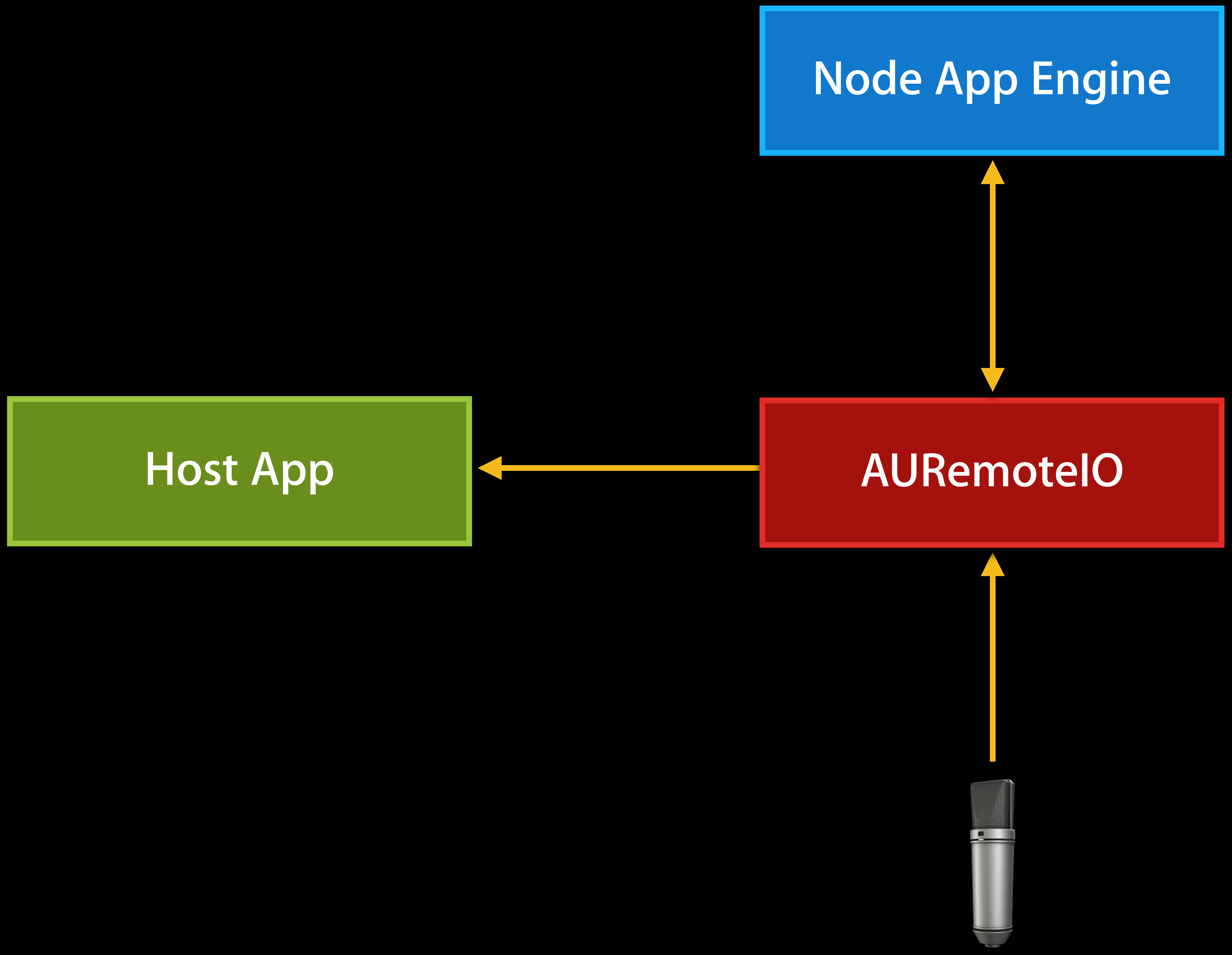
Node Rendering

Generator or instrument



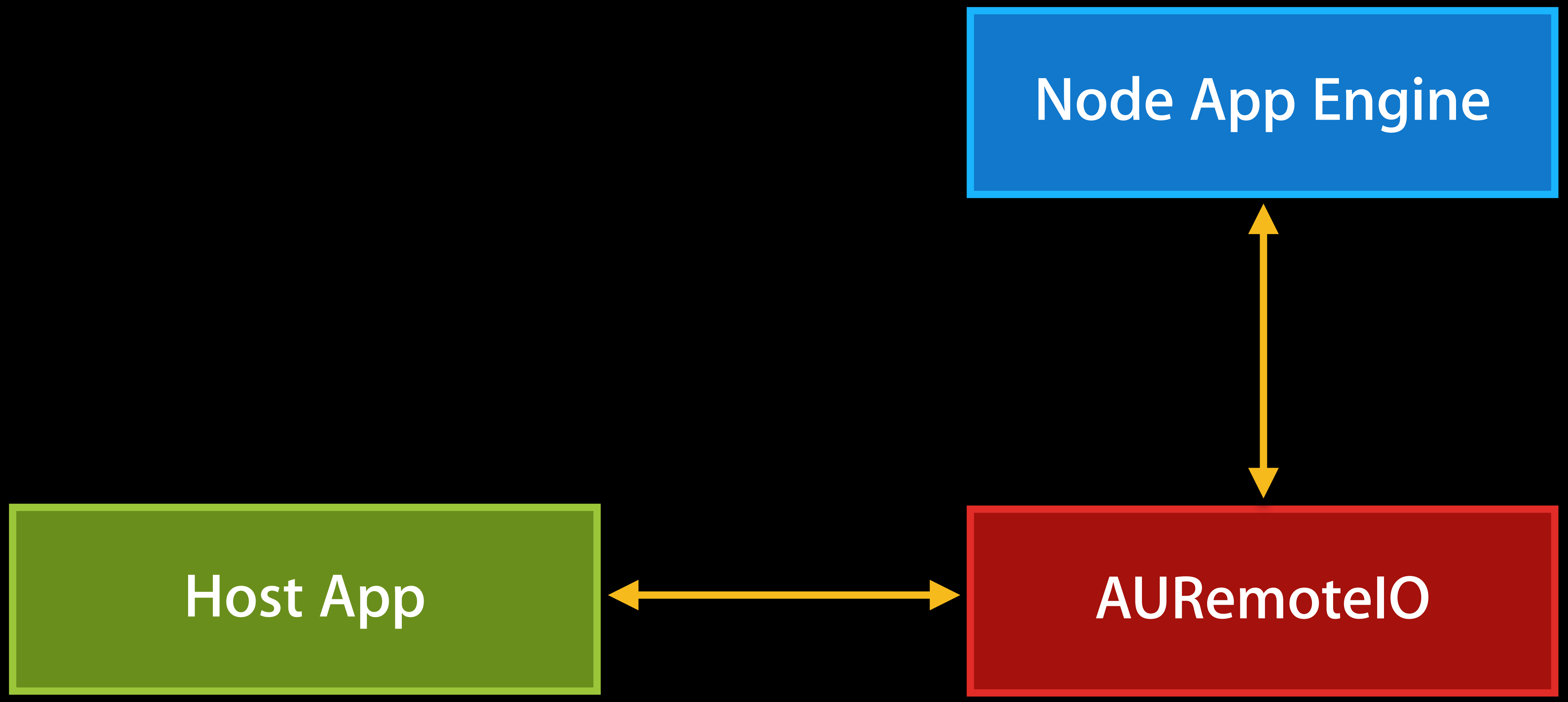
Node Rendering

Generator or instrument with audio input



Node Rendering

Effect



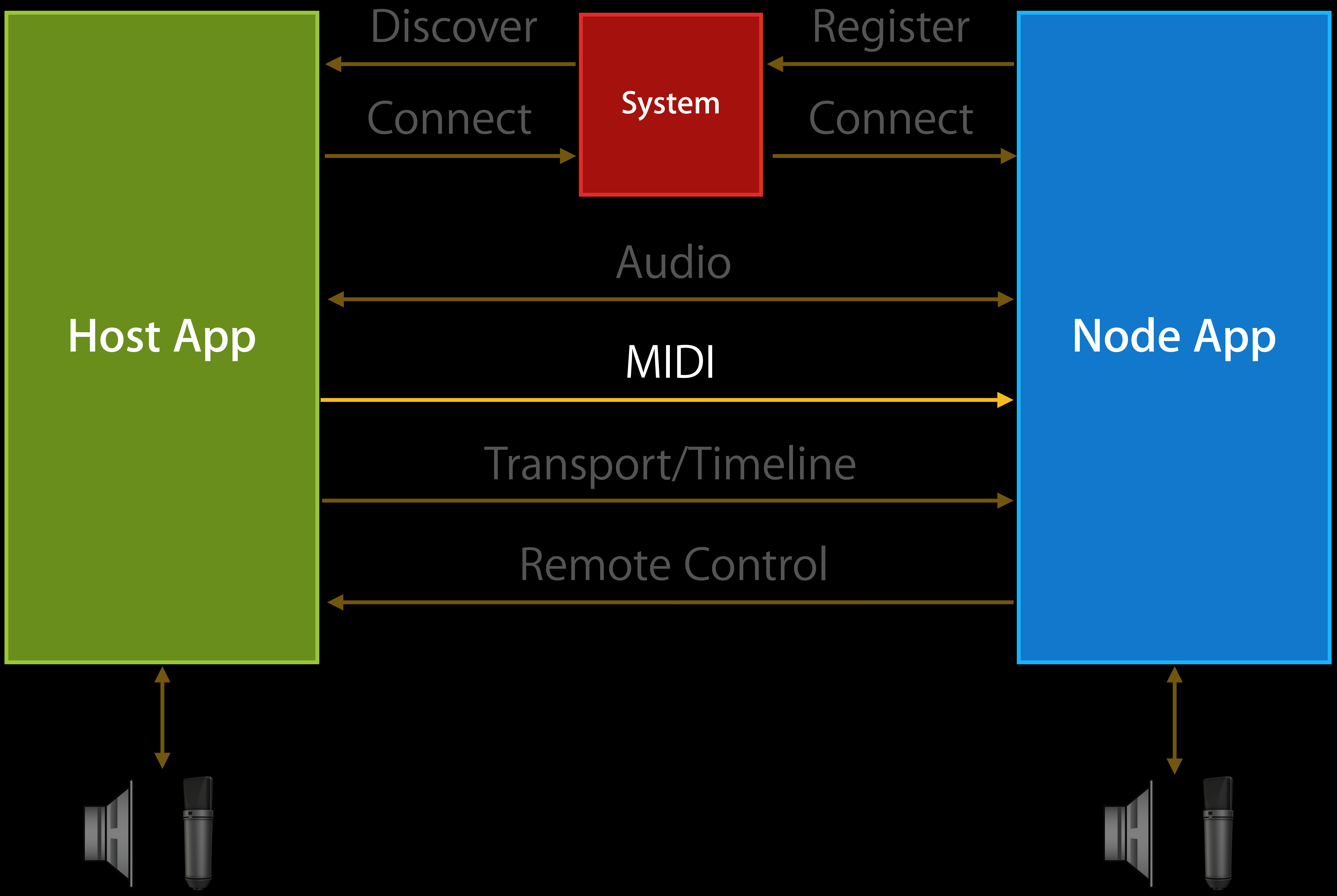
Switching from Node to Host

- Same as with host: Use `kAudioUnitProperty_PeerURL`

Stopping Audio Rendering (Host)

- AudioOutputUnitStop/AUGraphStop
- AudioUnitUninitialize
    - Release resources
    - Can re-initialize
- AudioComponentInstanceDispose
    - When completely finished
    - After node invalidated

MIDI



Host App
Discover
Connect
System
Register
Connect
Audio
MIDI
Transport/Timeline
Remote Control
Node App

MIDI

- For RemoteInstrument and RemoteMusicEffect nodes
    - When MIDI events tied to audio
    - Sample-accurate scheduling
    - Not for sync (clock/timecode)
- Complements CoreMIDI
    - USB/network
    - Apps that don’t support inter-app audio

Sending MIDI Events (Host)

Immediate, unscheduled

```
UInt32 offsetSampleFrames = 0;

const UInt8 kMIDINoteOn = 0x90;

const UInt8 kMiddleC = 60;

const UInt8 kVelocity = 64;

MusicDeviceMIDIEvent(myAudioUnit, kMIDINoteOn, kMiddleC, kVelocity,
  offsetSampleFrames);
```

Sending MIDI Events (Host)

Scheduled

```
double offsetSeconds = ...;
UInt32 offsetSampleFrames = offsetSeconds * sampleRate;
const UInt8 kMIDINoteOn = 0x90;
MusicDeviceMIDIEvent(myAudioUnit, kMIDINoteOn, 60 /* middle C */, 64,
  offsetSampleFrames);
AudioUnitRender(myAudioUnit, ...);
```

Sending MIDI Events (Host)

Scheduled, with AUGraph

```
AUGraphAddRenderNotify(myGraph, MyRenderNotify, self);

...

OSStatus MyRenderNotify(...
  const AudioTimeStamp *inTimeStamp
  ...
  UInt32 inNumberFrames ...)
{
     double offsetSeconds = ...;
     UInt32 offsetSampleFrames = offsetSeconds * sampleRate;
     const UInt8 kMIDINoteOn = 0x90;
     MusicDeviceMIDIEvent(myAudioUnit, kMIDINoteOn, 60 /* middle C */, 64,
       offsetSampleFrames);
     ...
}
```

Receiving MIDI Events (Node)

MIDI Callbacks

```
typedef struct {
   void *userData;
   // see MusicDeviceMIDIEvent, MusicDeviceSysEx
   void (*MIDIEventProc)(void *userData, UInt32 inStatus,
              UInt32 inData1, UInt32 inData2, UInt32 inOffsetSampleFrame);
   void (*MIDISysExProc)(void *userData, const UInt8* inData,
              UInt32 inLength);
} AudioOutputUnitMIDICallbacks;
```

Receiving MIDI Events (Node)

```
void MyMIDIEventProc(void *userData, UInt32 inStatus,
   UInt32 inData1, UInt32 inData2, UInt32 inOffsetSampleFrame)
{
   MyEngine *engine = (MyEngine *)userData;
   ...
}

void InstallMIDICallbacks(MyEngine *engine)
{
   AudioOutputUnitMIDICallbacks callbacks;
   callbacks.userData = engine;
   callbacks.MIDIEventProc = MyMIDIEventProc;
   callbacks.MIDISysExProc = NULL;
   AudioUnitSetProperty(myOutputUnit, kAudioOutputUnitProperty_MIDICallbacks,
     kAudioUnitScope_Global, 0,
     &callbacks, sizeof(callbacks));
}
```

Receiving MIDI Events (Node)

```
void MyMIDIEventProc(void *userData, UInt32 inStatus,
   UInt32 inData1, UInt32 inData2, UInt32 inOffsetSampleFrame)
{
   MyEngine *engine = (MyEngine *)userData;
   ...
}
```

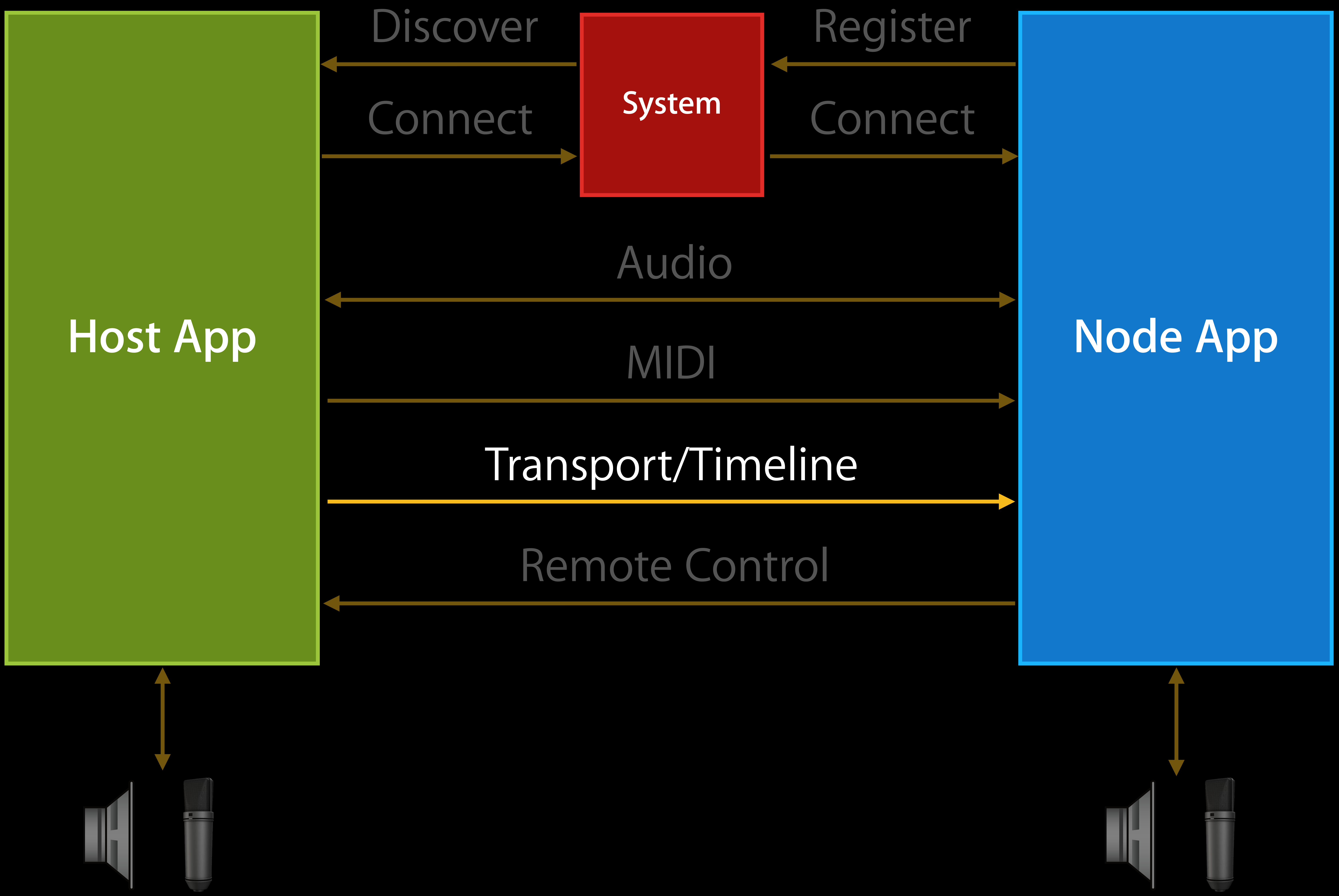
```
void InstallMIDICallbacks(MyEngine *engine)
{
   AudioOutputUnitMIDICallbacks callbacks;
   callbacks.userData = engine;
   callbacks.MIDIEventProc = MyMIDIEventProc;
   callbacks.MIDISysExProc = NULL;
   AudioUnitSetProperty(myOutputUnit, kAudioOutputUnitProperty_MIDICallbacks,
     kAudioUnitScope_Global, 0,
     &callbacks, sizeof(callbacks));
}
```

Receiving MIDI Events (Node)

```
void MyMIDIEventProc(void *userData, UInt32 inStatus,
   UInt32 inData1, UInt32 inData2, UInt32 inOffsetSampleFrame)
{
  MyEngine *engine = (MyEngine *)userData;
  ...
}
```

```
void InstallMIDICallbacks(MyEngine *engine)
{
  AudioOutputUnitMIDICallbacks callbacks;
  callbacks.userData = engine;
  callbacks.MIDIEventProc = MyMIDIEventProc;
  callbacks.MIDISysExProc = NULL;
  AudioUnitSetProperty(myOutputUnit, kAudioOutputUnitProperty_MIDICallbacks,
    kAudioUnitScope_Global, 0,
    &callbacks, sizeof(callbacks));
}
```

Transport and Timeline Information



Transport and Timeline Information

- Host is master; node can synchronize
- Musical position
- Transport state
- Called at render time

Transport and Timeline Information

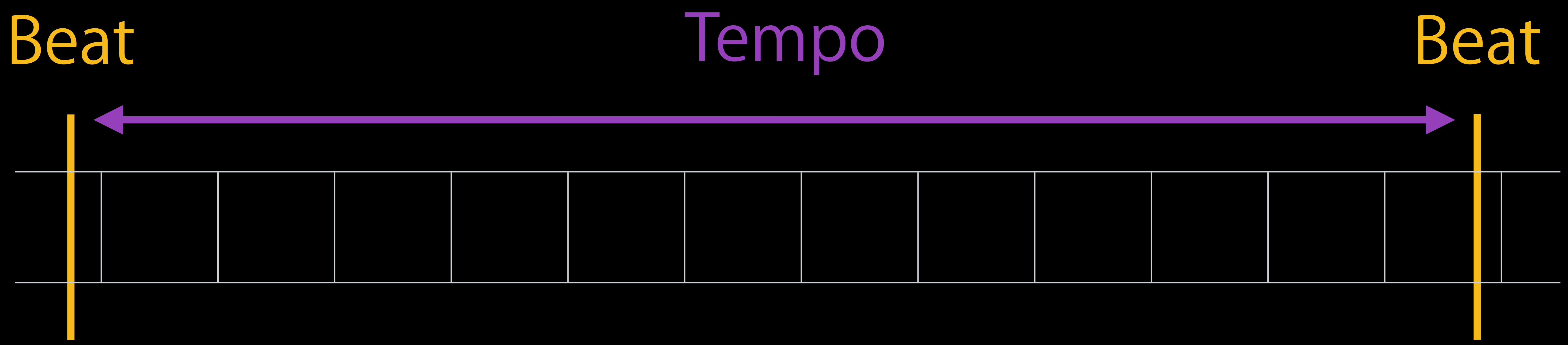
HostCallbackInfo

```
typedef struct HostCallbackInfo {
    void *                                  hostUserData;
    HostCallback_GetBeatAndTempo            beatAndTempoProc;
    HostCallback_GetMusicalTimeLocation     musicalTimeLocationProc;
    HostCallback_GetTransportState          transportStateProc;
    HostCallback_GetTransportState2         transportStateProc2;
} HostCallbackInfo;
```

Providing Host Callbacks

Beat and tempo

```
OSStatus MyBeatAndTempo(...)
{
   *outCurrentBeat = ...;  // position in track (float)
   *outCurrentTempo = ...; // beats per minute
   return noErr;
}
```



Providing Host Callbacks

Musical time location

```
OSStatus MyMusicalTimeLocation(...)
{
   *outDeltaSampleOffsetToNextBeat = ...;
   *outTimeSigNumerator = ...;
   *outTimeSigDenominator = ...;
   *outCurrentMeasureDownbeat = ...;
   return noErr;
}
```

Providing Host Callbacks

Transport state

```
OSStatus MyGetTransportState(...)
{
   *outIsPlaying = ...;
   *outIsRecording = ...;
   *outTransportStateChanged = ...;
   *outCurrentSampleInTimeline = ...;
   *outIsCycling = ...;
   *outCycleStartBeat = ...;
   *outCycleEndBeat = ...;
   return noErr;
}
```

Installing Host Callbacks

```
void InstallHostCallbacks(MyEngine *engine, AudioUnit myAudioUnit)
{
   HostCallbackInfo hci;

   hci.hostUserData = engine;
   hci.beatAndTempoProc = MyBeatAndTempo;
   hci.musicalTimeLocationProc = MyMusicalTimeLocation;
   hci.transportStateProc = NULL;
   hci.transportStateProc2 = MyGetTransportState;

   AudioUnitSetProperty(myAudioUnit, kAudioUnitProperty_kHostCallbacks,
     kAudioUnitScope_Global, 0,
     &hci, sizeof(hci));
}
```

Using Host Callbacks (Node)

- Get kAudioUnitProperty_HostCallbacks at connection time

```
HostCallbackInfo myHostCallbacks;

UInt32 propertySize = sizeof(myHostCallbacks);

AudioUnitGetProperty(myIOUnit, kAudioUnitProperty_kHostCallbacks,
  kAudioUnitScope_Global, 0,
  &myHostCallbacks, &propertySize);
```

- Call the desired host callback(s) at render time
- Caution: Thread-safety

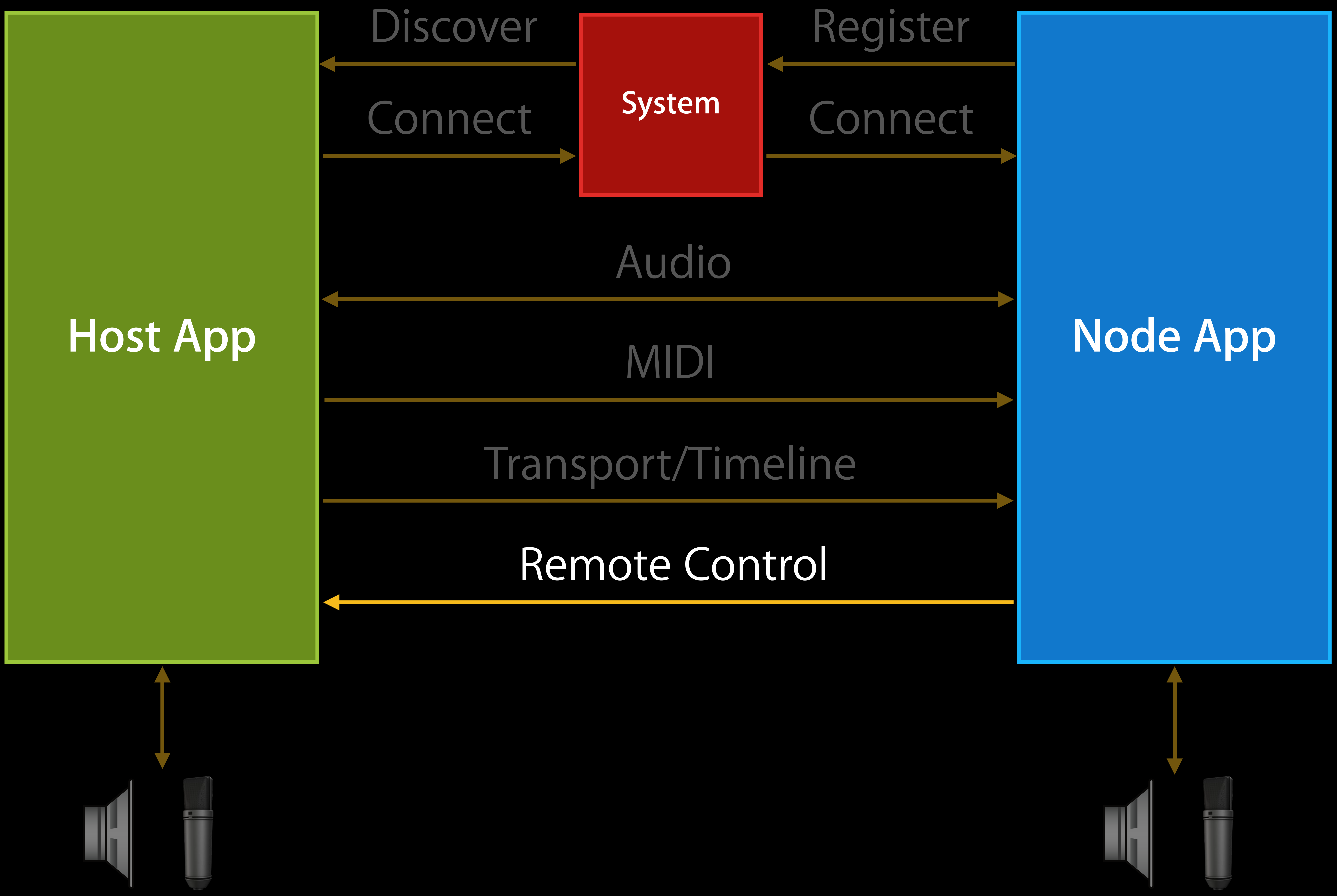
Using Host Callbacks (Node)

Observing transport state changes

- Transport state listener
- Called on non-render thread

```
AudioUnitAddPropertyListener(myIOUnit,
  kAudioOutputUnitProperty_HostTransportState,
  MyTransportStateListener, self);
```

Remote Control



Audio Unit Remote Control Events

- Distinct from UIKit remote control events
- Node can control host's transport

```
enum {
    kAudioUnitRemoteControlEvent_TogglePlayPause = 1,
    kAudioUnitRemoteControlEvent_ToggleRecord    = 2,
    kAudioUnitRemoteControlEvent_Rewind          = 3
};
typedef UInt32 AudioUnitRemoteControlEvent;
```

- Node: Use the recommended transport controls in the sample app

Audio Unit Remote Control Events

From node

- Query whether host supports them:

```
UInt32 eventsSupported = 0, propertySize = sizeof(eventsSupported);
AudioUnitGetProperty(myIOUnit,
  kAudioOutputUnitProperty_HostReceivesRemoteControlEvents,
  kAudioUnitScope_Global, 0,
  &eventsSupported, &propertySize);
```

- Send an event:

```
AudioUnitRemoteControlEvent theControl =
  kAudioUnitRemoteControlEvent_ToggleRecord;
AudioUnitSetProperty(myIOUnit,
  kAudioOutputUnitProperty_RemoteControlToHost,
  kAudioUnitScope_Global, 0,
  &theControl, sizeof(theControl));
```

Audio Unit Remote Control Events

In host

- Set kAudioUnitProperty_RemoteControlEventListener

```
AudioUnitRemoteControlEventListener listenerBlock =
  ^(AudioUnitRemoteControlEvent event) {
    [self handleRemoteEvent: event];
  };

AudioUnitSetProperty(myAudioUnit,
  kAudioOutputUnitProperty_RemoteControlEventListener,
  kAudioUnitScope_Global, 0,
  &listenerBlock, sizeof(listenerBlock));
```

Demo

Inter-app host, instrument and effect, with remote control

Harry Tormey

Audio Session Interruptions

- The usual rules apply
  - Your remote I/O has already been stopped underneath you
- In hosts, system has uninitialized your node audio units

Handling Media Services Reset

- AVAudioSessionMediaServicesWereResetNotification
- All inter-app audio connections are broken (component instances invalidated)
- Host: Dispose node AudioUnit and AURemoteIO
- Node: Dispose AURemoteIO
- Proceed as if app has been launched

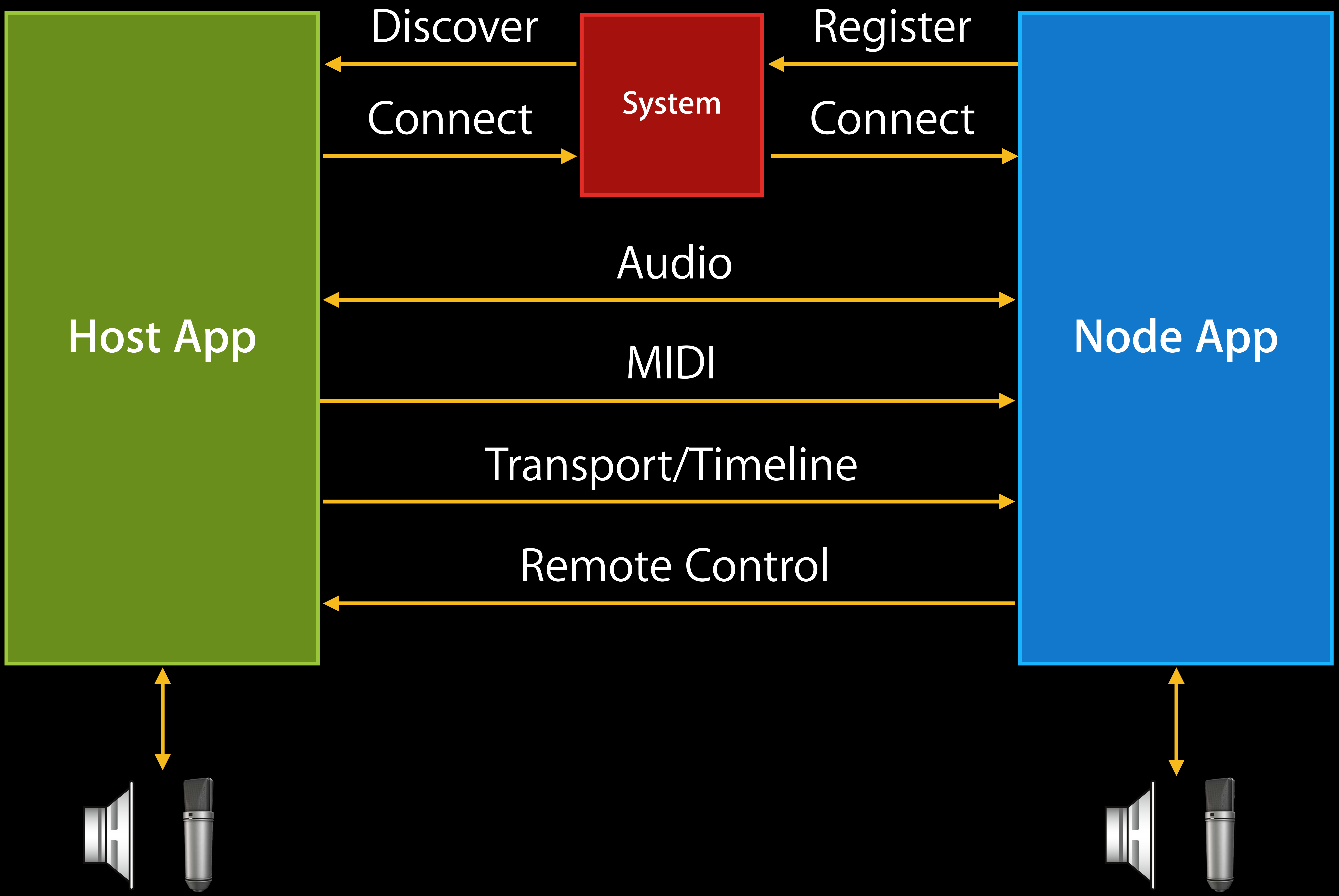
Odds and Ends

- Multiple hosts?
  - If all mixable, yes
- Multiple nodes?
  - Yes

Tips

- Debug node registration
    - Watch console log
- Error -12985
    - “Operation denied”
    - Can’t start playing from the background

Review


Discover
System
Register
Connect
Connect
Audio
Host App
Node App
MIDI
Transport/Timeline
Remote Control

Conclusion

- Existing apps can be converted to nodes fairly simply
- Creating a host is a bit more work, but use AudioUnit APIs for power and flexibility
- Use sample code from WWDC library
- Make great music apps!

Apple Evangelists
Contact information

John Geleynse
Director, Technology Evangelist
geleynse@apple.com

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

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

Related Sessions

Moving to AV Kit and AV Foundation	Pacific Heights Tuesday 4:30PM
Preparing and Presenting Media for Accessibility	Nob Hill Wednesday 10:15AM
What's New in Camera Capture	Nob Hill Wednesday 11:30AM
Advanced Editing with AV Foundation	Marina Thursday 9:00AM

Labs

Audio Lab	Media Lab B Tuesday 2:00PM
Audio Lab	Media Lab B Wednesday 9:00AM

WWDC2013