MOBILE SENSING LEARNING

CS5323 & 7323
Mobile Sensing and Learning

core audio

Eric C. Larson, Lyle School of Engineering,
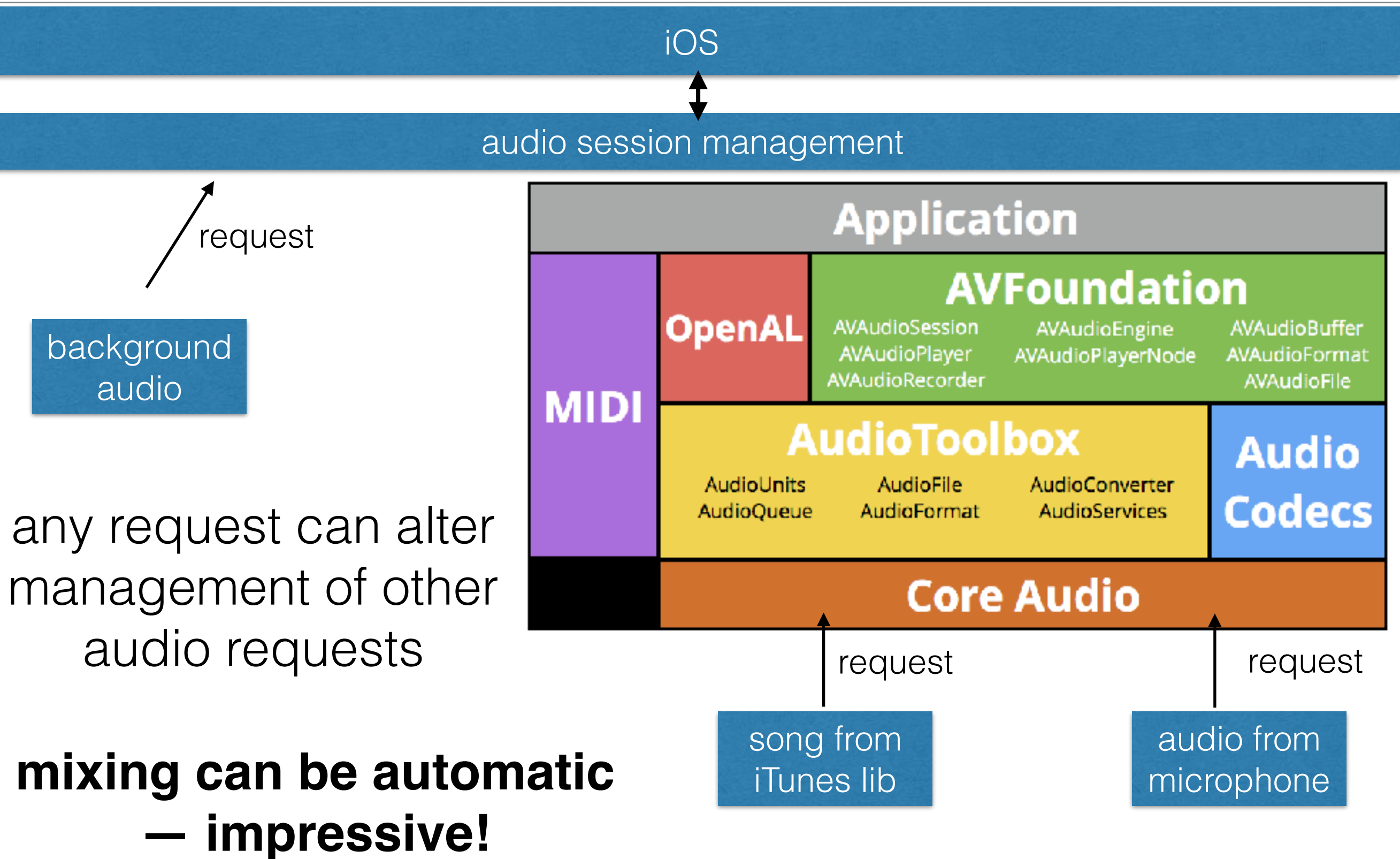Computer Science, Southern Methodist University

agenda

- core audio intro

- next lecture: more audio and graphing audio

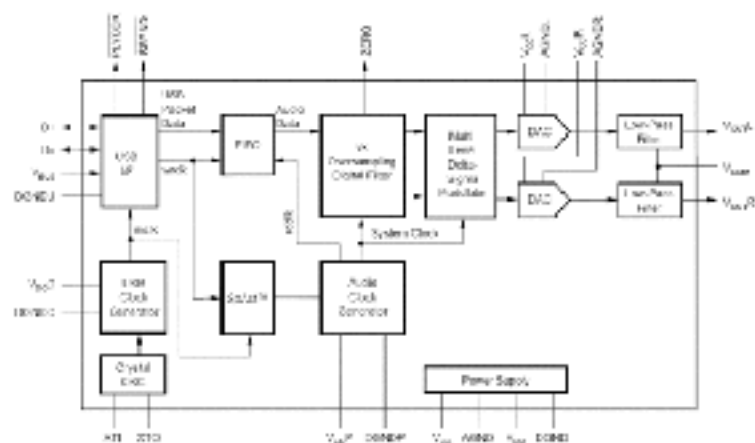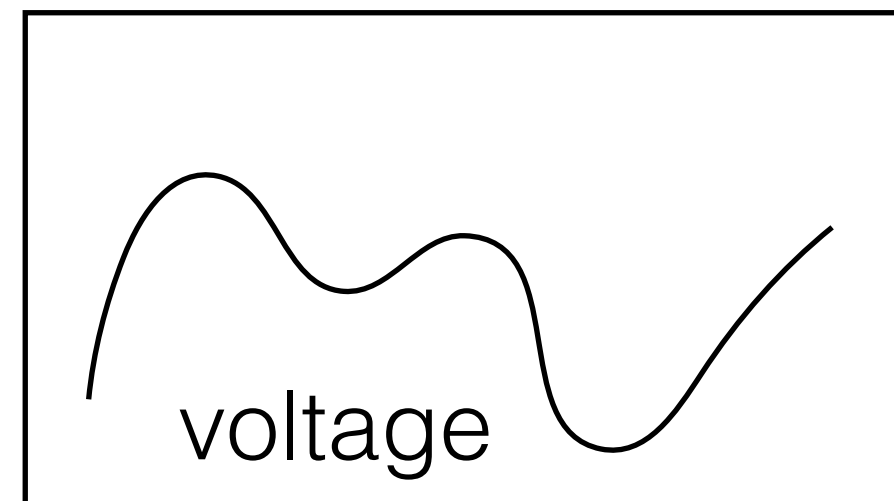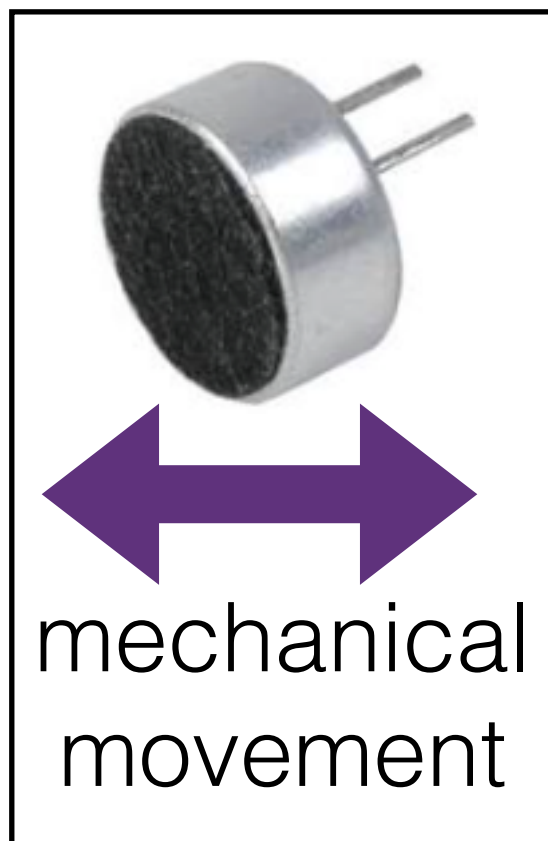- **next week**: second flipped module, the FFT

Core Audio

- many **audio packages** exist, but we want **low level signals**

- **Audio Sessions** (high level, completely overhauled starting iOS7)

  - shared instance (for all applications)

  - set category (play, record, both)

    - choose options: like mixing with ambient sources

  - set audio route (new starting in iOS7)

    - set specific hardware within audio route

- **Audio Units** (more low level, output, input)

  - set stream format, buffer sizes, sampling rate,

  - initialize memory for audio buffers

  - set callback rendering procedure

audio sessions

iOS

$\updownarrow$

audio session management

request

background
audio

any request can alter
management of other
audio requests

**mixing can be automatic
— impressive!**

request

request

song from
iTunes lib

audio from
microphone

audio hardware

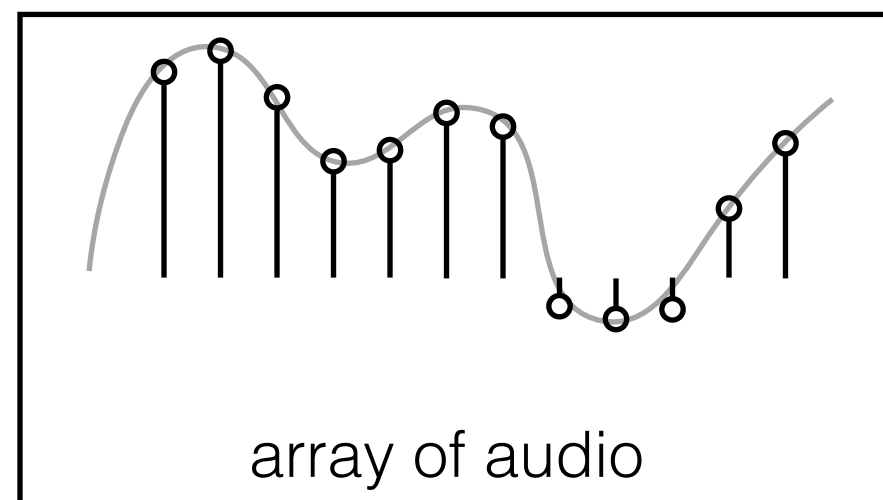sound

mechanical movement

amplifier

voltage

buffer and ADC

audio card

array of audio
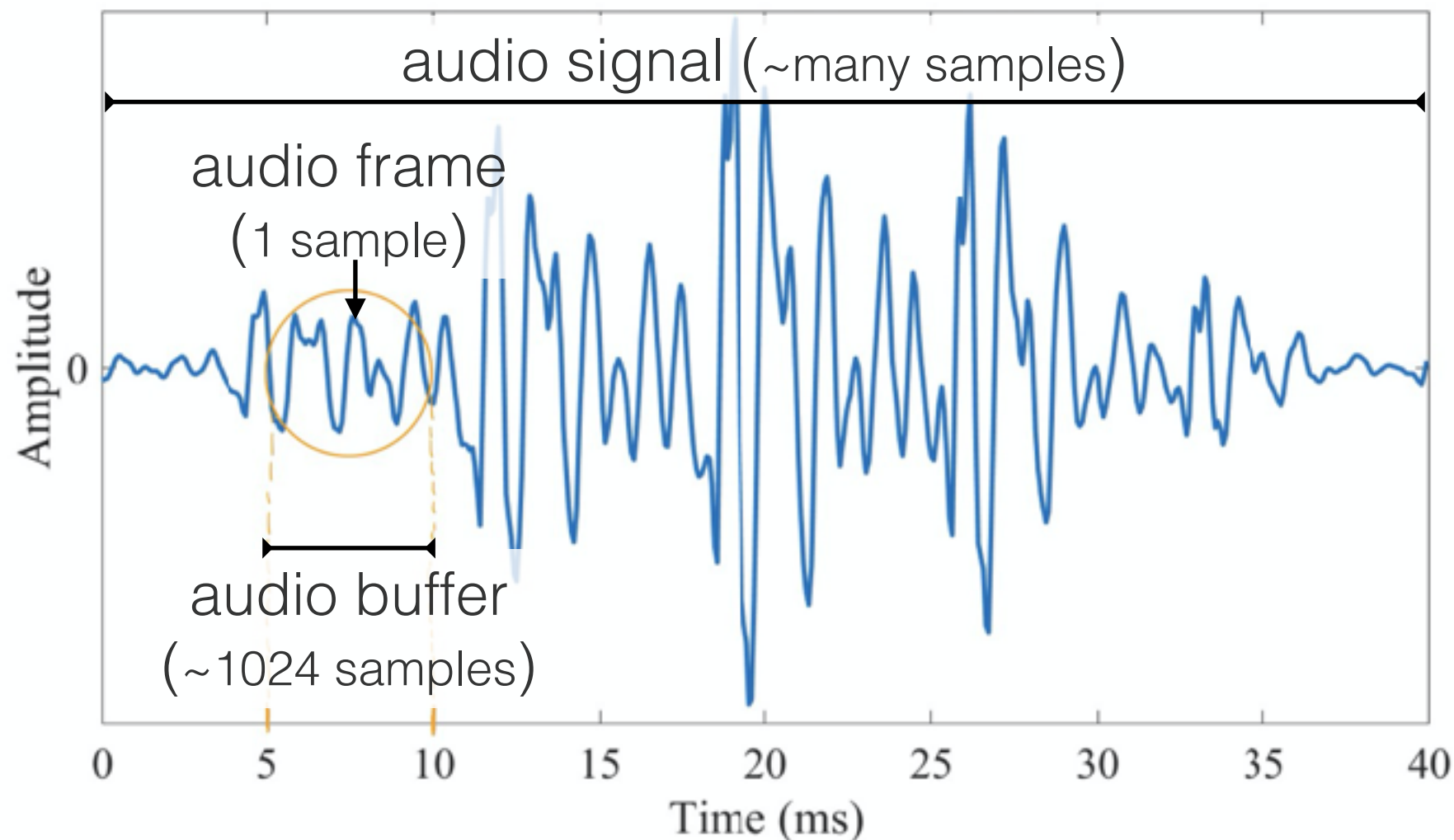
notify software a buffer is ready

audio buffering

- audio card buffers up audio samples before sending to the main processor

audio units

audio input buffer procedure, double buffer shown
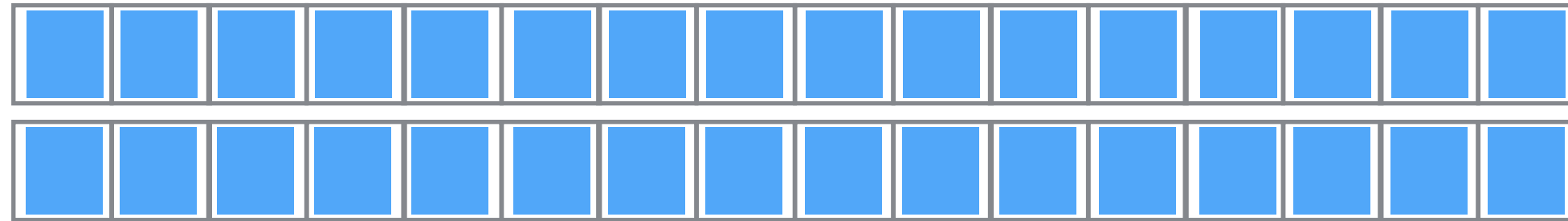
Audio Card (memory allocated on card)

common language

C

sent to audio session callback

CPU
(memory in RAM)

copy over samples, convert

exit from call as soon as possible!

do not allocate memory, take locks, or waste time!!

audio unit formats

microphone (input)
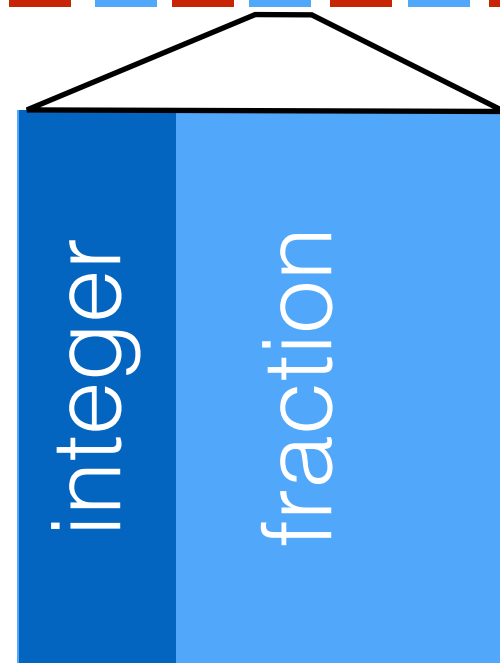

stereo could be interleaved (output)

■ right speaker

■ left speaker

integer | fraction

32 bits

callback preallocates buffers
developer fills the output buffer
OS handles playing the buffer
if you don't fill fast enough,
audio is choppy

audio units solution…

```swift
// The audio engine used to record input from the microphone.
private let audioEngine = AVAudioEngine()

// setup audio
let audioSession = AVAudioSession.sharedInstance()
do{
    try audioSession.setCategory(AVAudioSession.Category.record)
    try audioSession.setMode(AVAudioSession.Mode.measurement)
    try audioSession.setActive(true, options: .notifyOthersOnDeactivation)
}
catch { fatalError("Audio engine could not be setup") }

let inputNode = audioEngine.inputNode
let recordingFormat = inputNode.outputFormat(forBus: 0)
```

Setup Audio

```swift
inputNode.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat)
                    { (buffer: AVAudioPCMBuffer, when: AVAudioTime) in
```

…some APIs need the AVAudioBuffer Object…

```swift
}

audioEngine.prepare()
do{ try audioEngine.start() }
catch { fatalError("Audio engine could not start") }
```

Get Samples

but, audio unit taps are slower than using core audio…

wouldn't it be **great** if there was a module that **handled** all the specifics of **audio units for us**?

Alex Wiltschko
alexbw

- Twitter
- Boston, MA
- alex.bw@gmail.com
- Joined on Dec 4, 2009

Novocaine: takes the pain out of audio processing

Originally developed by **Alex Wiltschko**

Heavily manipulated by **eclarson**

novocaine

- Novocaine needs callbacks

declare properties and setup manager

init circular buffer (written for you)

```swift
private lazy var audioManager:Novocaine? = {
        return Novocaine.audioManager()
    }()
private lazy var circBuffer:CircularBuffer? = {
        return CircularBuffer.init(
                        numChannels: Int64(self.audioManager!.numInputChannels),
                        andBufferSize: Int64(BUFFER_SIZE))
    }()
```

declare properties

setup audio and init buffer

```objc
@property (strong, nonatomic) Novocaine *audioManager;
@property (strong, nonatomic) CircularBuffer *buffer;

_audioManager = [Novocaine audioManager];
_buffer = [[CircularBuffer alloc]initWithNumChannels:1 andBufferSize:BUFFER_SIZE];
```

the novocaine in/out block

input

```
self.audioManager?.inputBlock = self.handleMicrophone

private func handleMicrophone (data:Optional<UnsafeMutablePointer<Float>>,
                                numFrames:UInt32,
                                numChannels: UInt32) {
        // copy samples from the microphone into circular buffer
        self.circBuffer?.addNewFloatData(data, withNumSamples: Int64(numFrames))
    }
```

output

```
self.audioManager?.outputBlock = self.handleSpeakerQueryWithAudioFile

private func handleSpeakerQueryWithAudioFile(data:Optional<UnsafeMutablePointer<Float>>,
                                numFrames:UInt32,
                                numChannels: UInt32){
        self.outputBuffer?.fetchInterleavedData(data, withNumSamples:Int64(numFrames))
    }
```

> setup function to give novocaine for callback

> microphone samples as float array

> data to write to speakers

```
[self.audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels){


    [weakSelf.buffer addNewFloatData:data withNumSamples:numFrames];
}];



[self.audioManager setOutputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)
{
    [weakSelf.buffer fetchInterleavedData:data withNumSamples:numFrames];

}];
```

> microphone samples as float array

> data to write to speakers

The program, using MVC

interacts with audio card

interacts with Novocaine

Setup Core audio and data buffer conversion

Set code for reading or writing audio samples

Use callbacks from model, as fast as possible

Manage access to samples for controllers

callbacks

Novocaine, low level c

audio model, swift

model properties

called each interval

Query audio model
for updated
buffer properties

Manage behavior of model

Use buffers for interaction or display

in view controller, swift or objective-c

novocaine setup demo

source code on GitHub

rolling stones, if time

and now its time for a demo
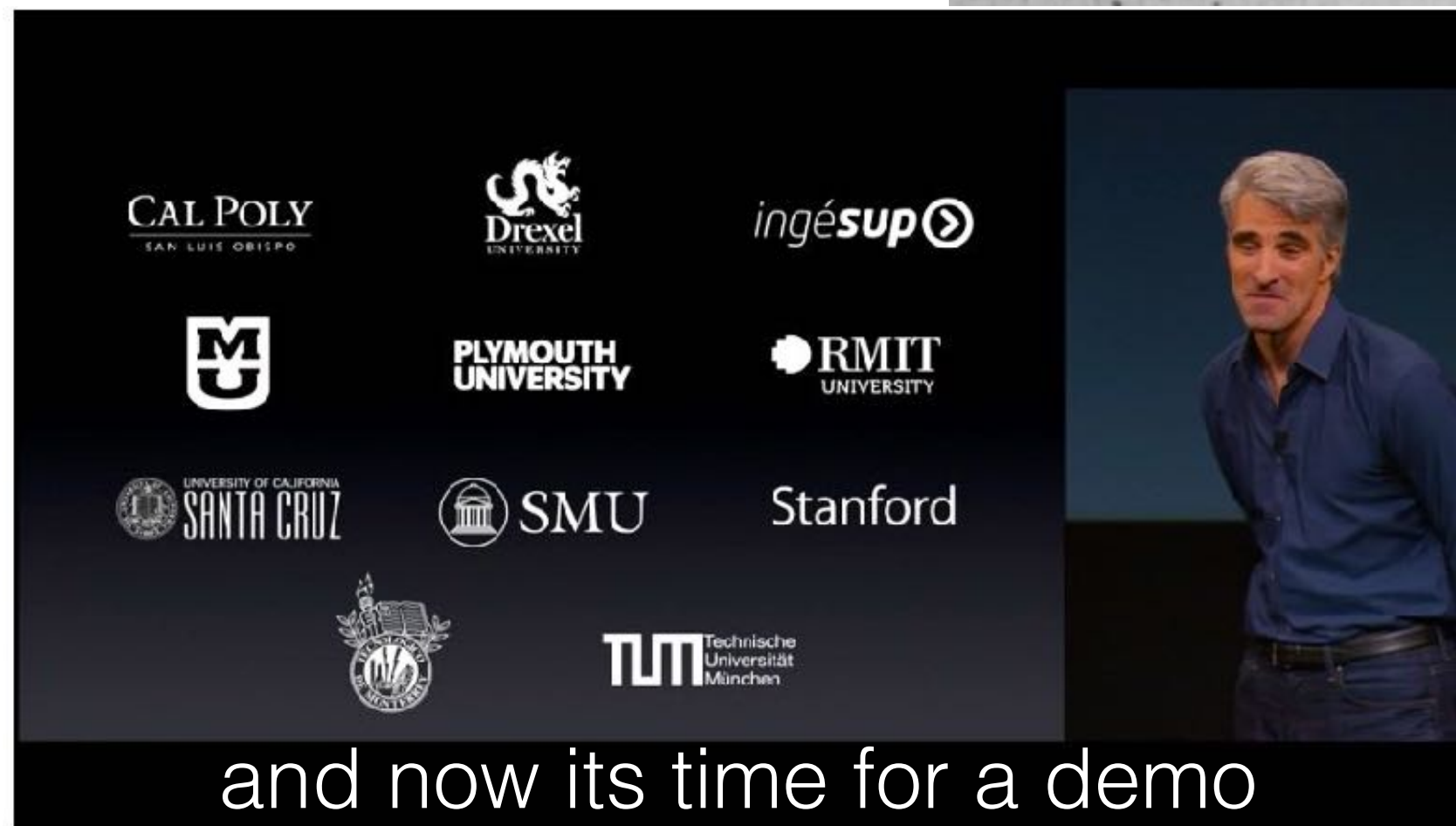
Declare in info.plist

Application requires it none environment	✓	Boolean	YES
Privacy - Microphone Usage Description	⇕	String	This App requires microphone access.
Application Scene Manifest	⌃	Dictionary	(2 items)

for next time…

- more core audio

  - playing songs (if not covered today)

  - getting samples from microphone

    - showing samples with Metal

  - working with sampled data

  - the accelerate framework

MOBILE SENSING LEARNING

CS5323 & 7323

Mobile Sensing and Learning

audio session

Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University