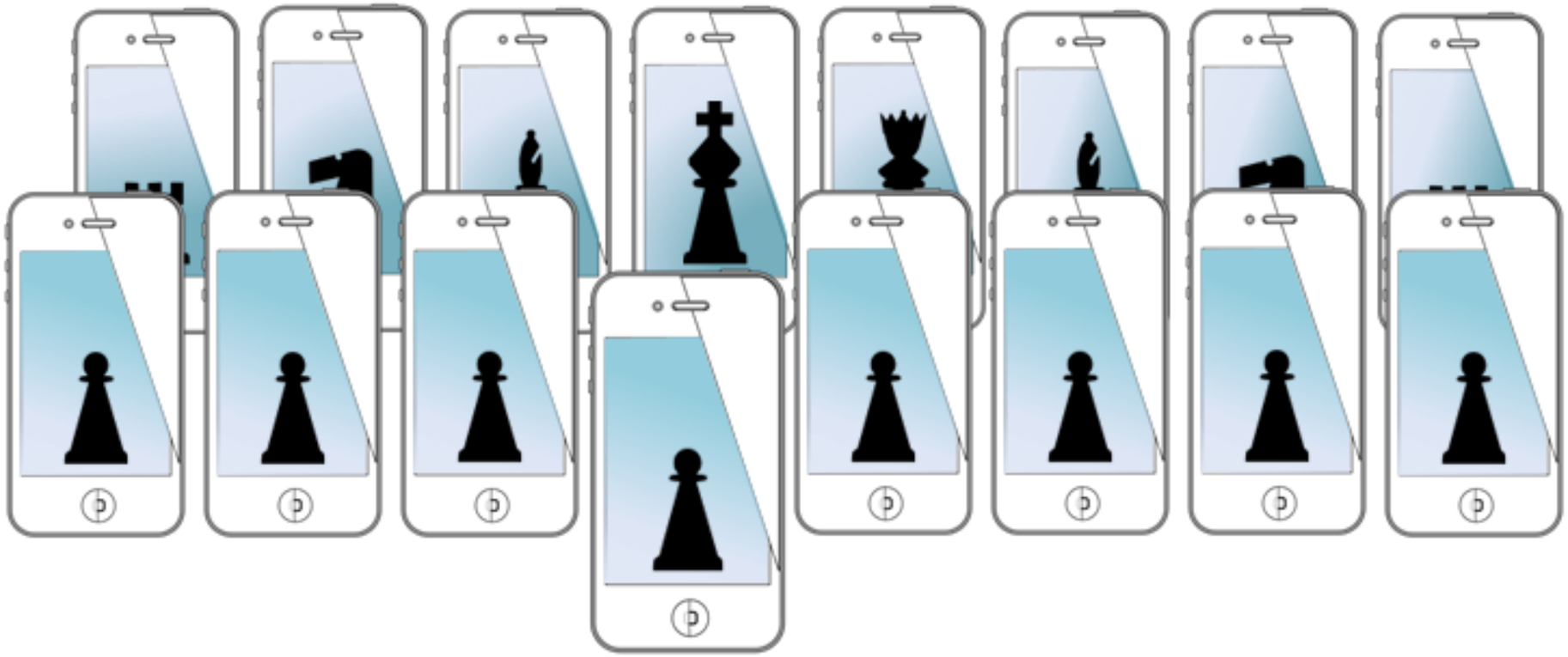
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

core audio

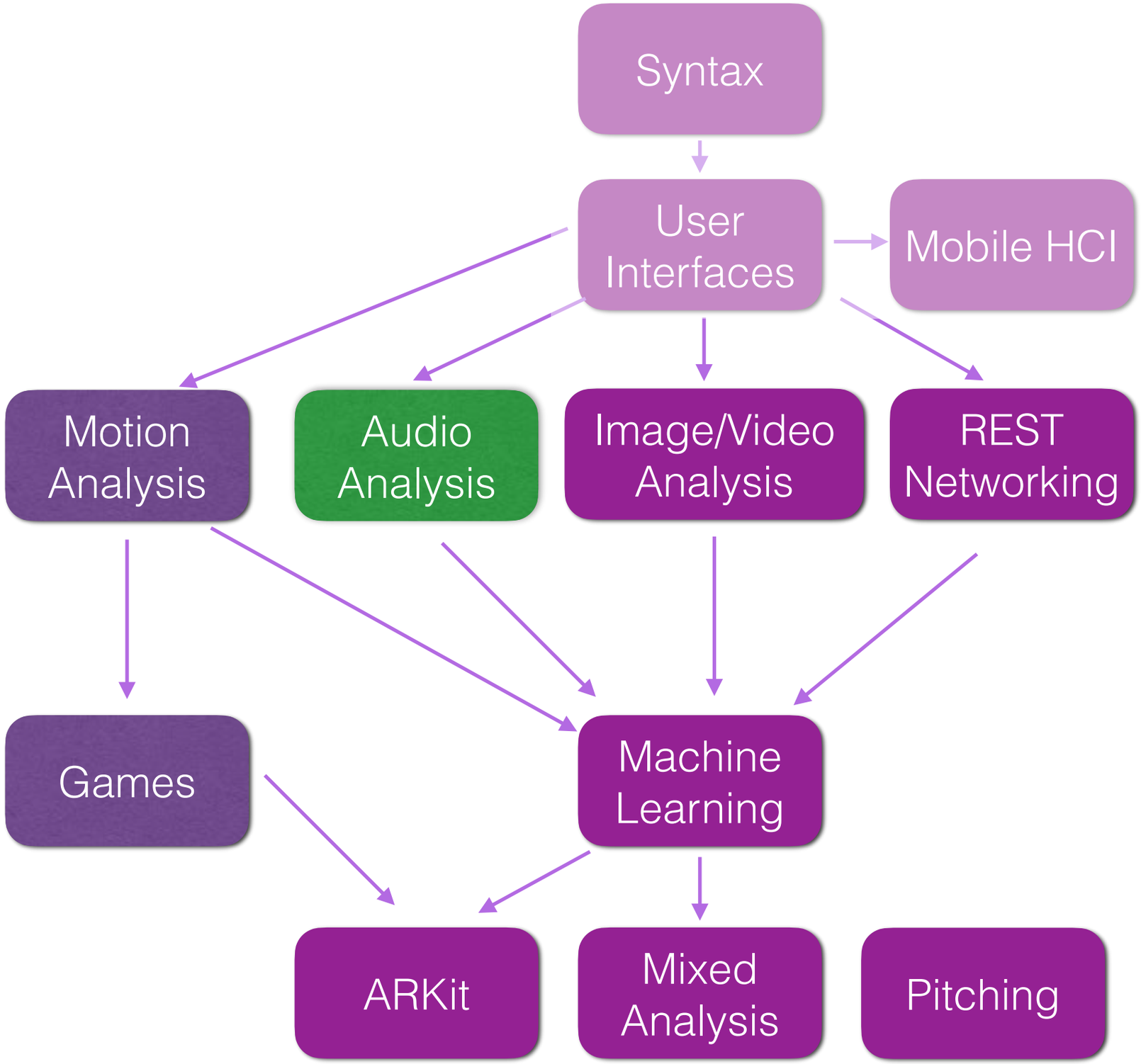
Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University

logistics and agenda

Wed.	Audio Access
Mon.	Fast Audio Plotting
Wed.	Flipped Module 2: FFT

- logistics:
  - Xcode updates
  - **next lecture**: more audio and graphing audio
  - **next week**: second flipped module, the FFT
- agenda:
  - blocks and closures
  - core audio intro

class overview



blocks and closures

- a block of code that you want to run at another time and perhaps pass to other classes to run
  - created at runtime
  - acts like an object that can be passed as an argument or created on the fly
  - once created, can be called repeatedly
  - can access variables from scope where defined
  - syntax is slightly different in swift and objective-c
  - common to define when calling a method that uses block
- swift calls these **closures**, objective-c says **blocks**

block/closure syntax

most common usage is as input into a function

```
myArray.enumerateObjects( {(obj, idx, ptr) in
    print("\(obj) is at index \(idx)")
})
```

```
{ ( parameters ) -> return type in
    statements
}
```

```
myArray.enumerateObjects(){(obj, idx, ptr) in
    print("\(obj) is at index \(idx)")
}
```

Also valid if closure is last input

```
var elapsedTime:Int = 0

let timer:Timer = Timer(timeInterval: 1, repeats: true) { tmp in
    elapsedTime += 1
    self.feedbackLabel.text = "Elapsed: \(elapsedTime)"
}

RunLoop.main.add(timer, forMode: .common)
```

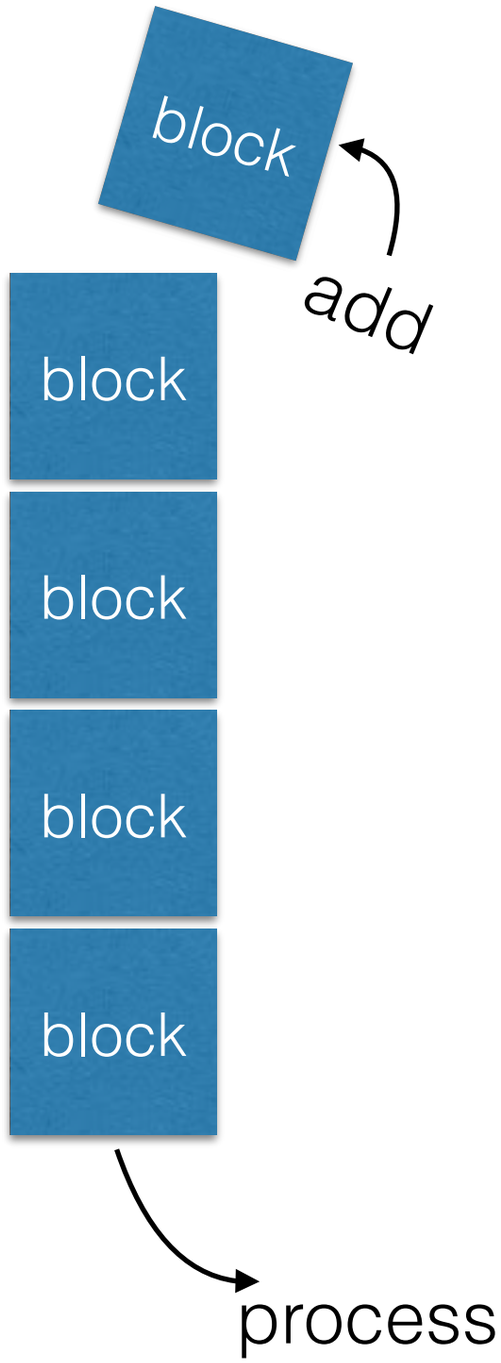
in scope and editable!

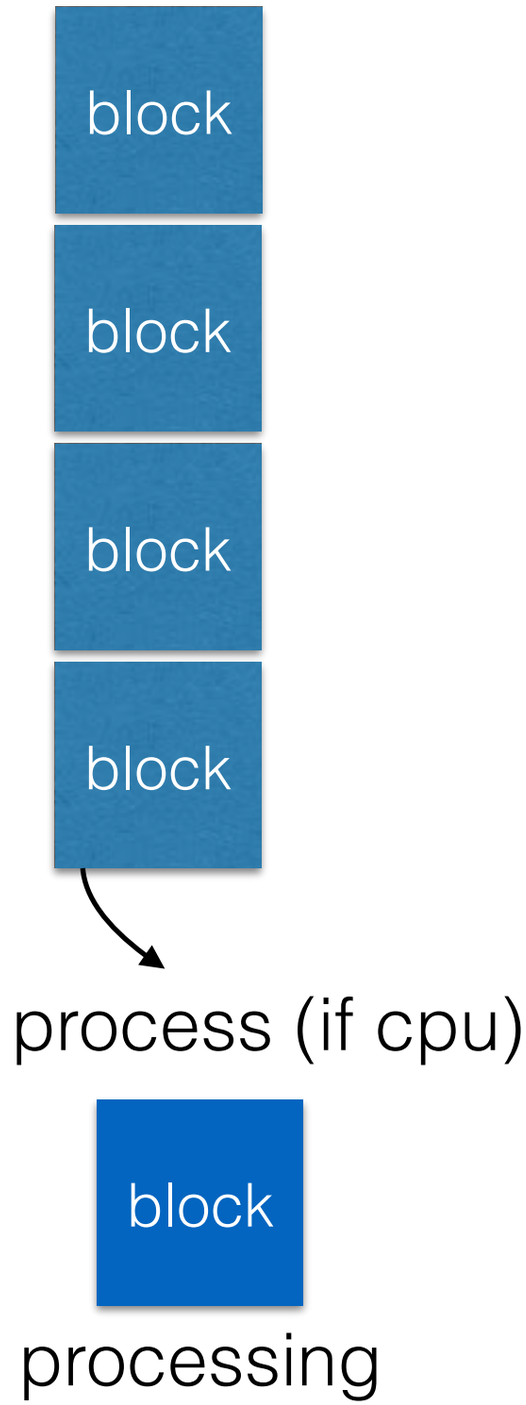
concurrency in iOS

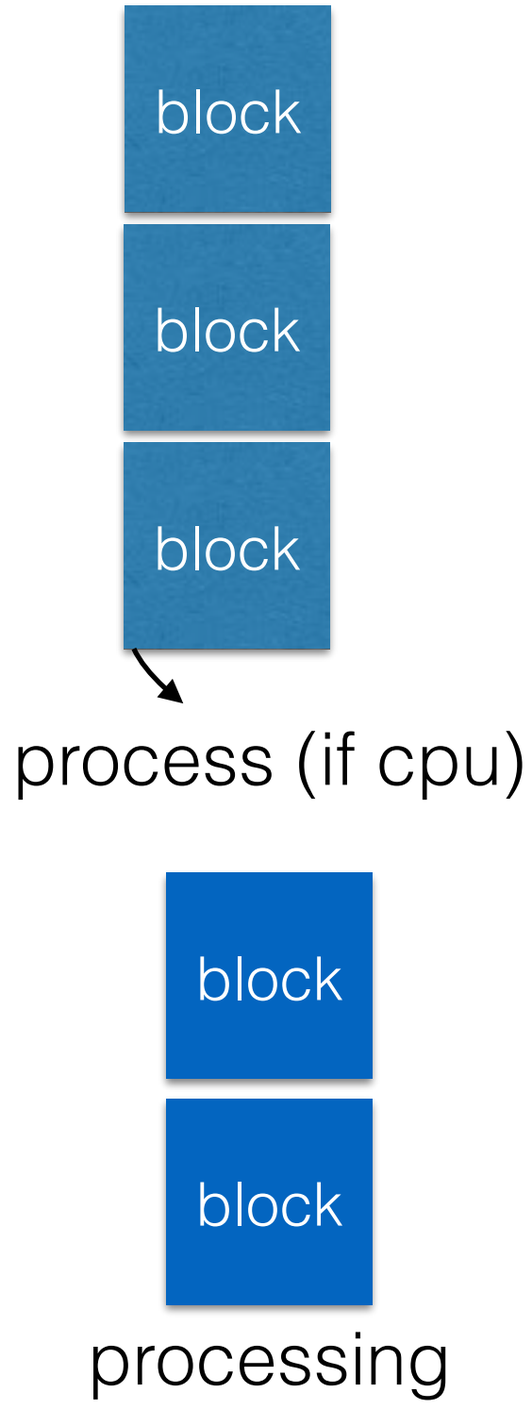
optional content

- grand central dispatch (GCD) handles all operations
  - GCD looks at “queues” of **blocks** that need to be run
  - GCD and the Xcode compiler work deep inside the OS, actually in the kernel — they are optimized
  - for a **serial queue** each block is run sequentially
  - for **concurrent queues** the first block is dequeued
    - if CPU is available, then the next block is also dequeued, but could finish any time
- the **main queue handles all UI operations** (and no other queue should generate UI changes!!)
  - so, **no updating of** the views, labels, buttons, (image views*) **except from the main queue**

concurrent queues





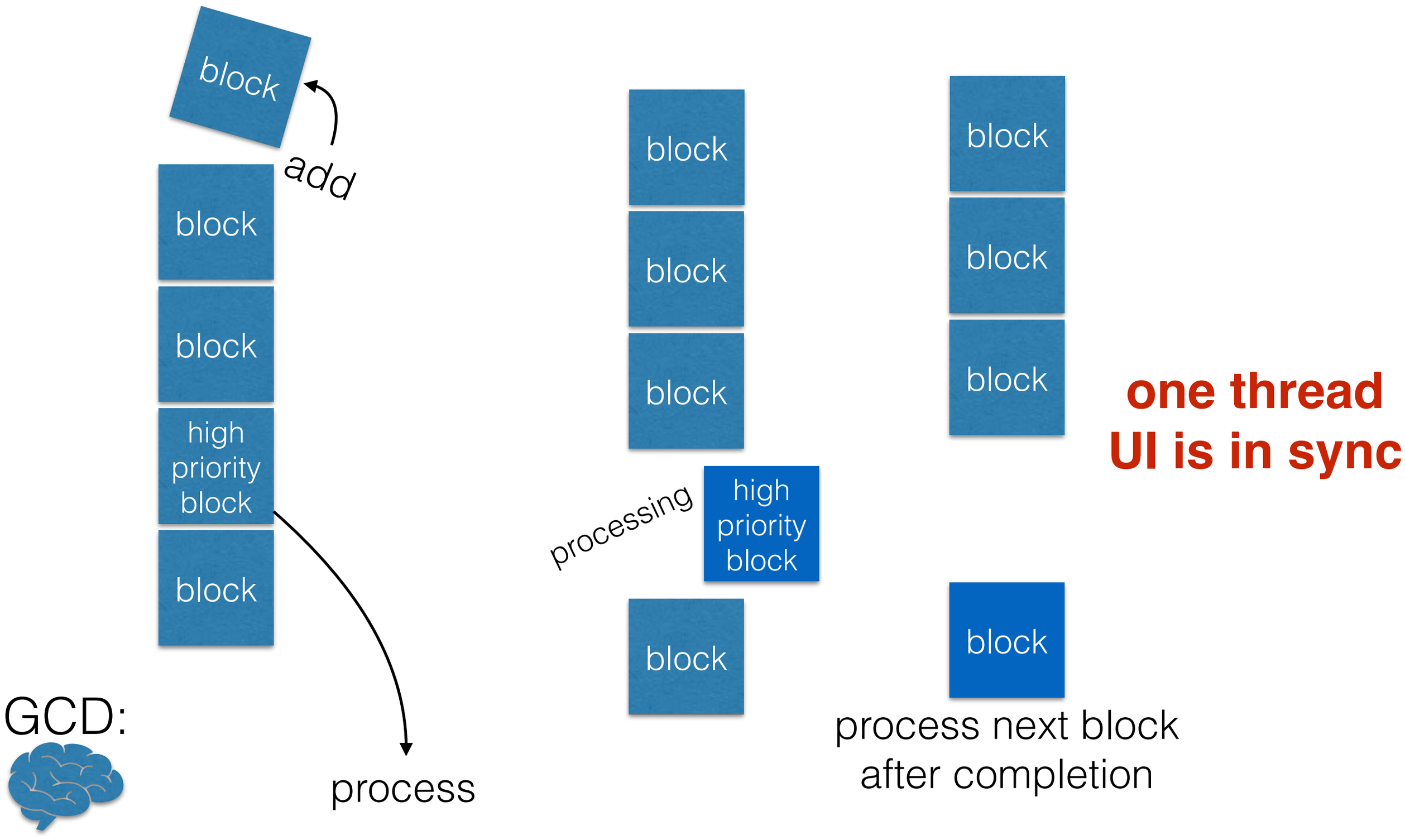


threads?

either could finish

GCD:

the main queue



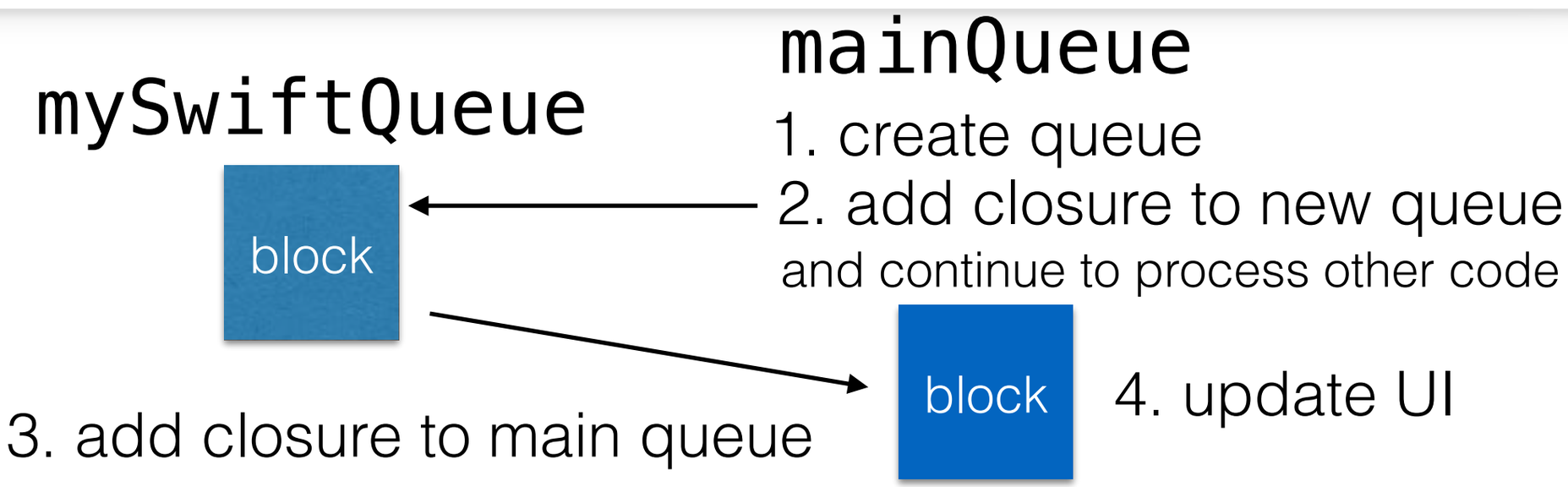
create your own queue!

```
var queue:DispatchQueue = DispatchQueue(label: “mySwiftQueue”)

queue.async() {

    //code to execute in block
    for _ in 0..<3{
        print(" I am being executed from a custom queue")
    }

    // now we go to the main queue
    DispatchQueue.main.async() {
        print("Running from main queue!")
    }
}
```

create new queue

define closure

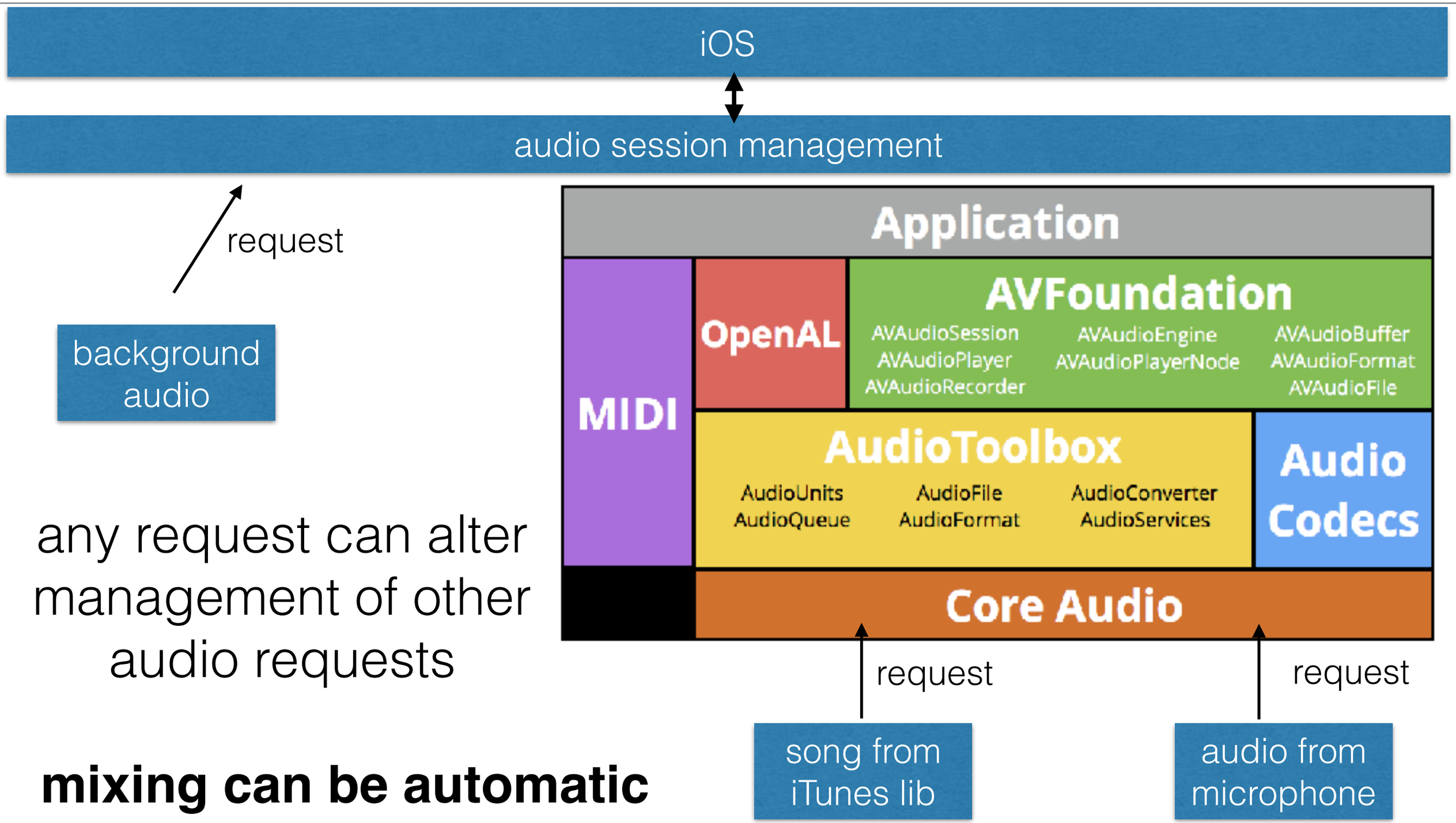
update UI, another closure



Core Audio

- many **audio packages** exist, but we want **low level signals**
- **Audio Sessions** (high level, setup audio hardware)
  - access the shared instance (for all applications)
  - access/set category (play, record, both)
    - choose options: like mixing with ambient sources
  - access/set audio route (mostly we will ignore this)
    - set specific hardware within audio route
  - **Audio Units** (once hardware is ready handle the input/output)
    - set stream format, buffer sizes, sampling rate,
    - initialize memory for audio buffers
    - set callback rendering procedure

audio sessions



any request can alter management of other audio requests

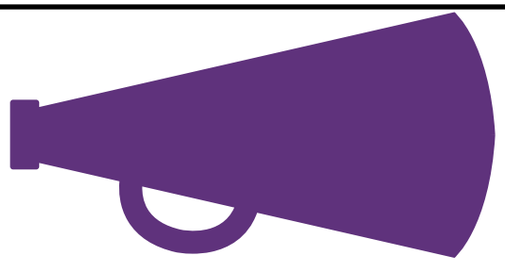
mixing can be automatic — impressive!

audio hardware

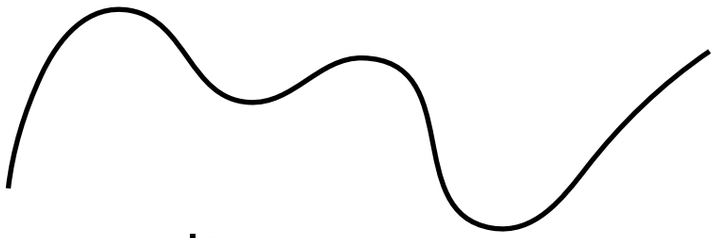
Input

sound

mechanical movement



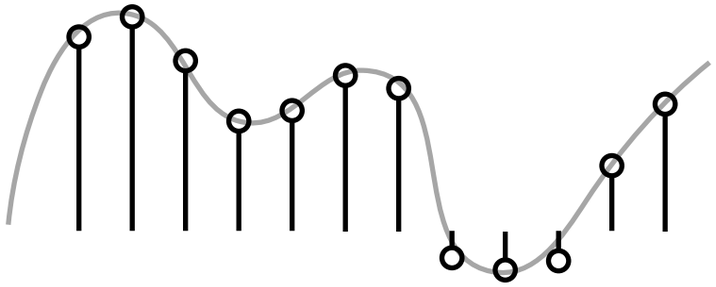
amplifier



voltage

audio card

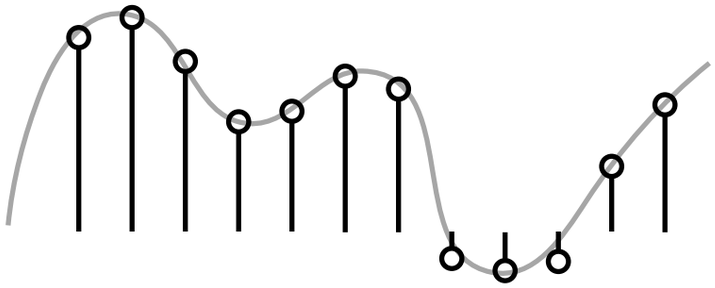
buffer and ADC



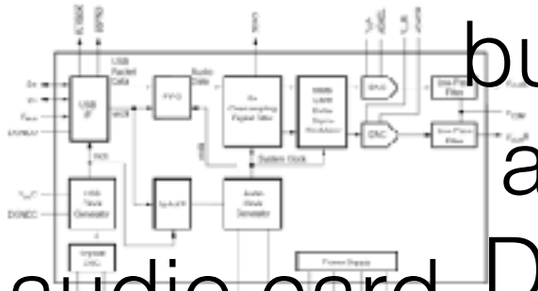
array of audio

notify software a buffer is ready

Output

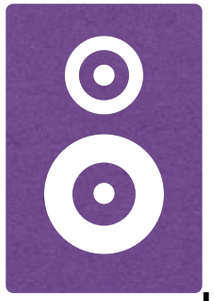


array of audio to play



audio card

buffer and DAC

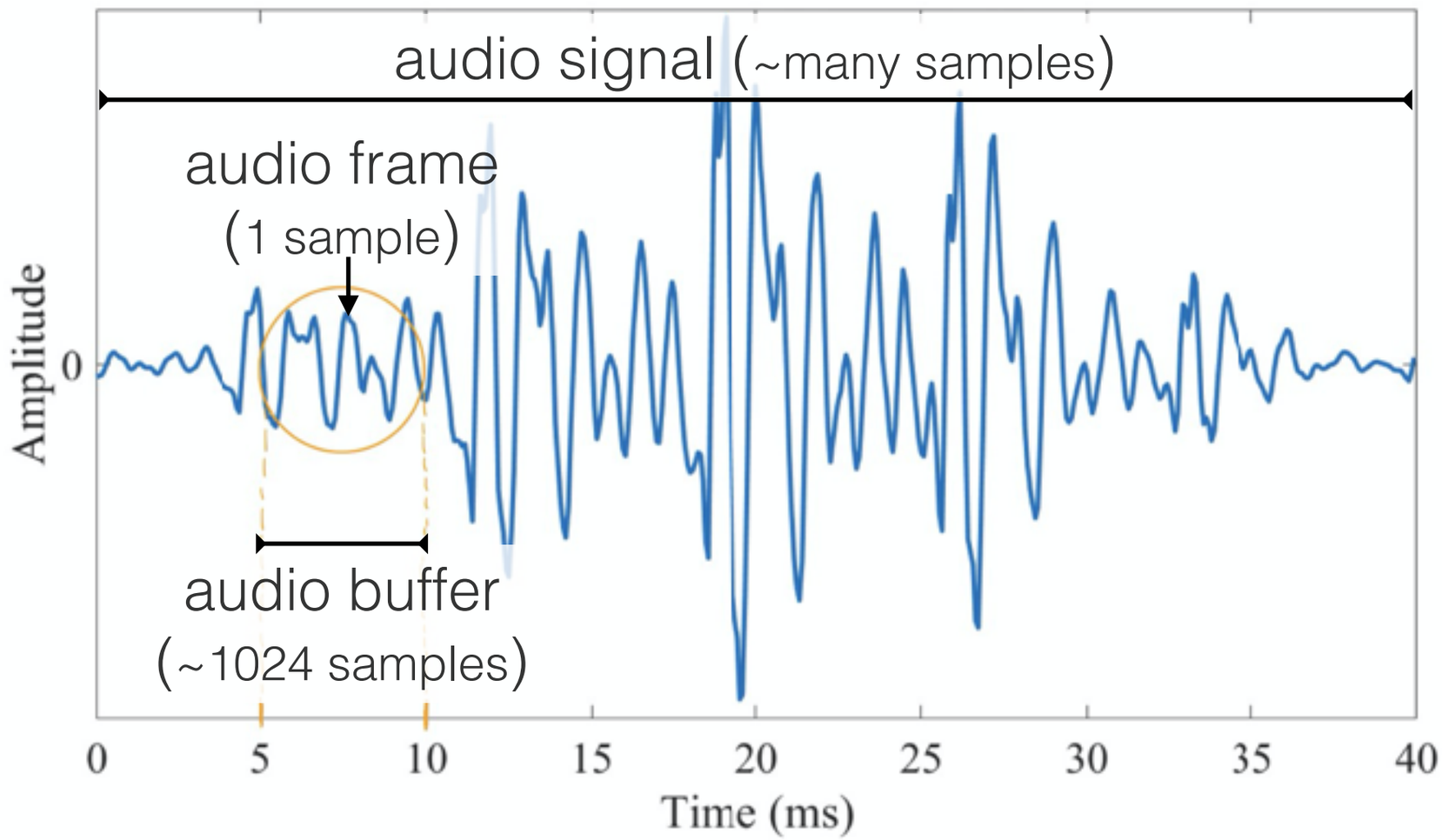


speaker

sound

audio buffering

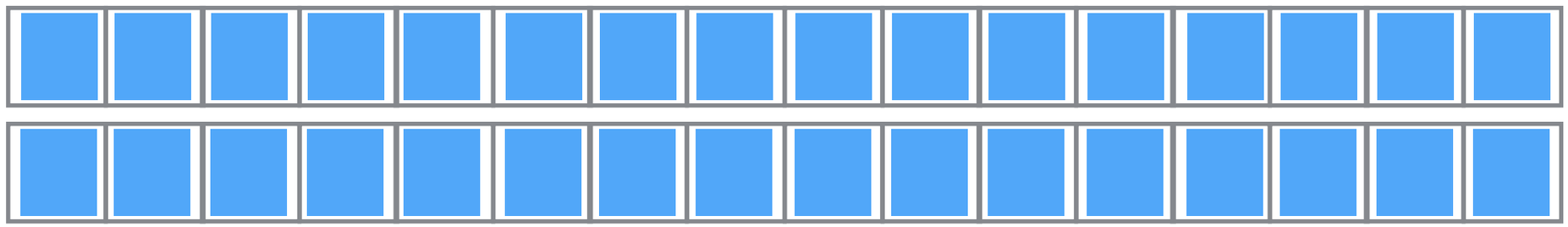
- audio card buffers up audio samples before sending to the main processor



https://medium.com/better-programming/audio-visualization-in-swift-using-metal-accelerate-part-1-390965c095d7

audio units

audio input buffer procedure, double buffer shown



Audio Card (memory allocated on card)

common language

C

sent to audio session closure

CPU

(memory in RAM)

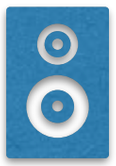
copy over samples, convert

exit from call as soon as possible!

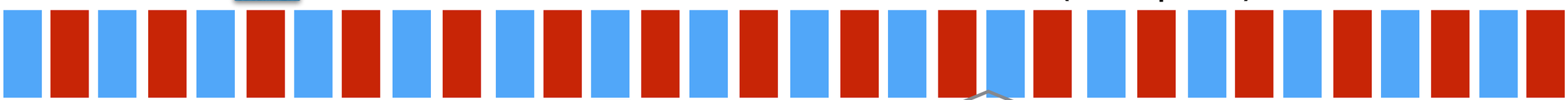
do not allocate memory, take locks, or waste time!!

audio unit formats

microphone (input)

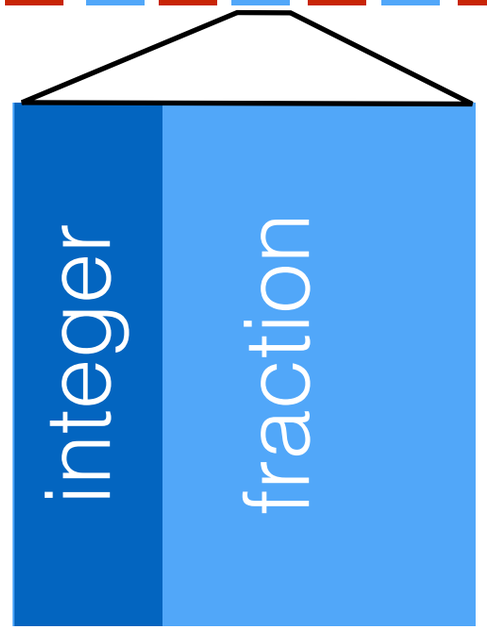


stereo could be interleaved (output)



right speaker

left speaker



32 bits

setup preallocates buffers
developer fills the output buffer
OS handles playing the buffer

if you don't fill fast enough, audio is choppy

audio nodes solution…

```
// 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

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

}

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

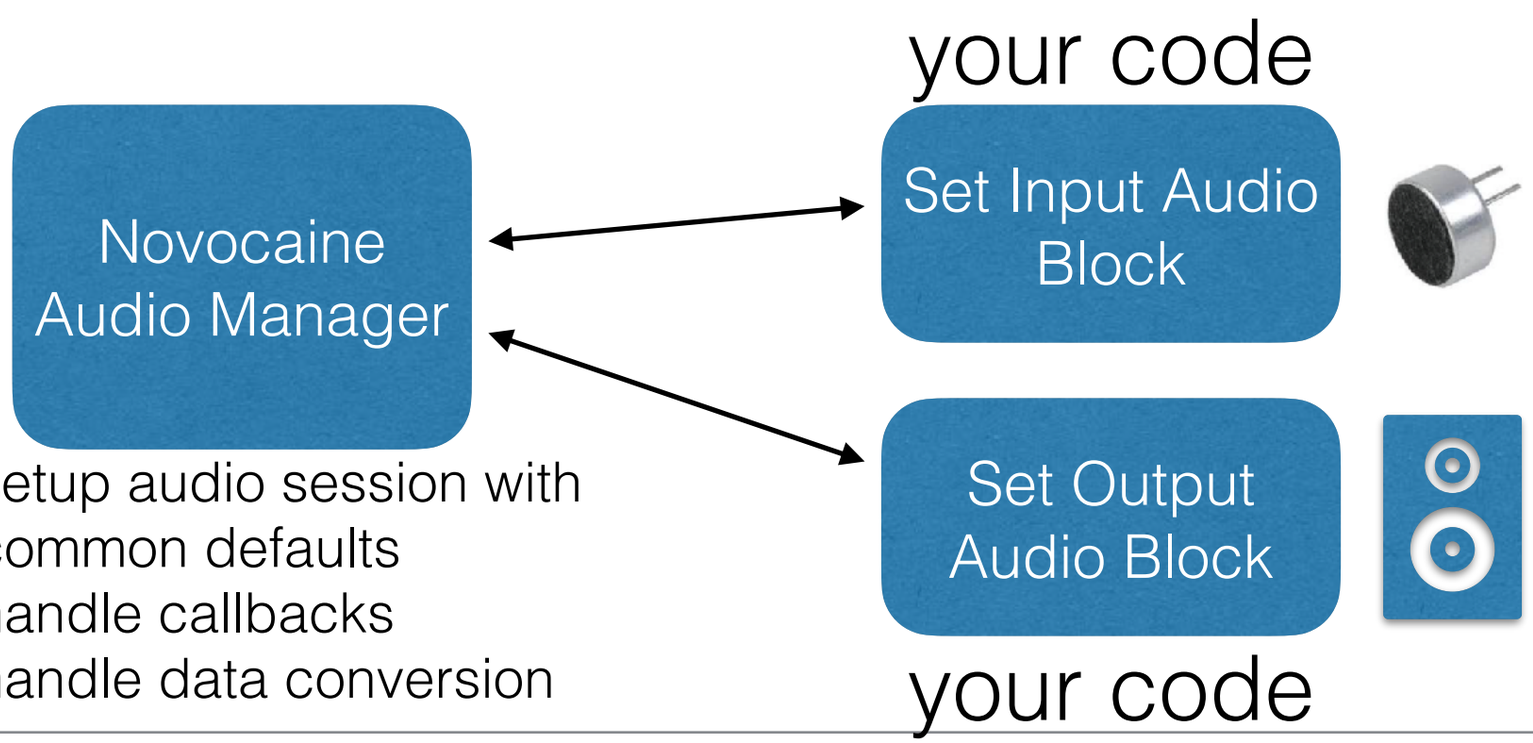
…some APIs need the AVAudioBuffer Object…

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**?

Novocaine: takes the pain out of audio processing
Originally developed by **Alex Wiltschko**
Heavily manipulated by **eclarson**



- setup audio session with common defaults
- handle callbacks
- handle data conversion

Alex Wiltschko
alexbw

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

novocaine

- Novocaine needs callbacks

```
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 and setup manager

init circular buffer (written for you)

input

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

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

setup function to give novocaine for callback

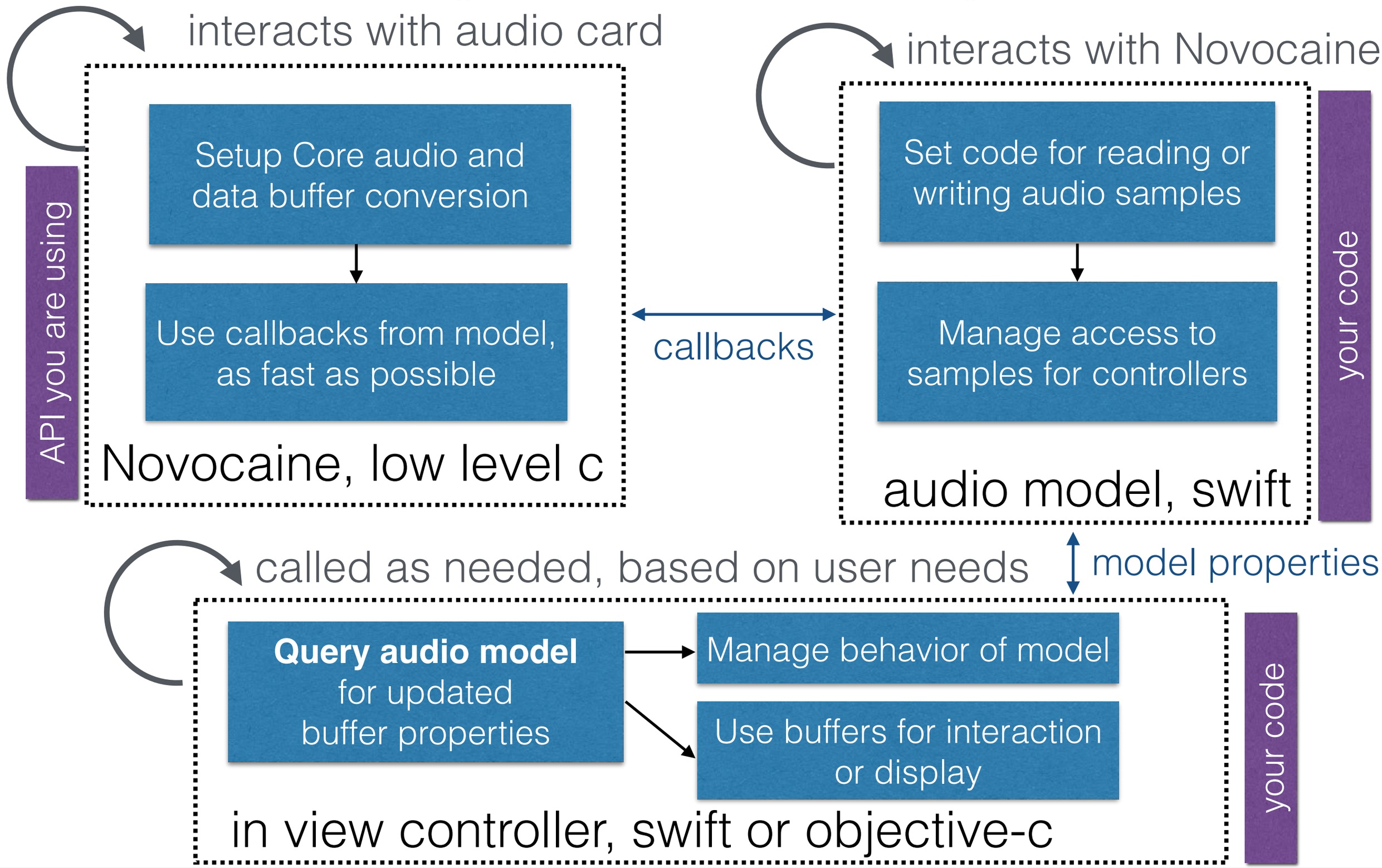
microphone samples as float array

output

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

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

data to write to speakers

The program, using MVC



novocaine setup demo

source code on GitHub

rolling stones, if time

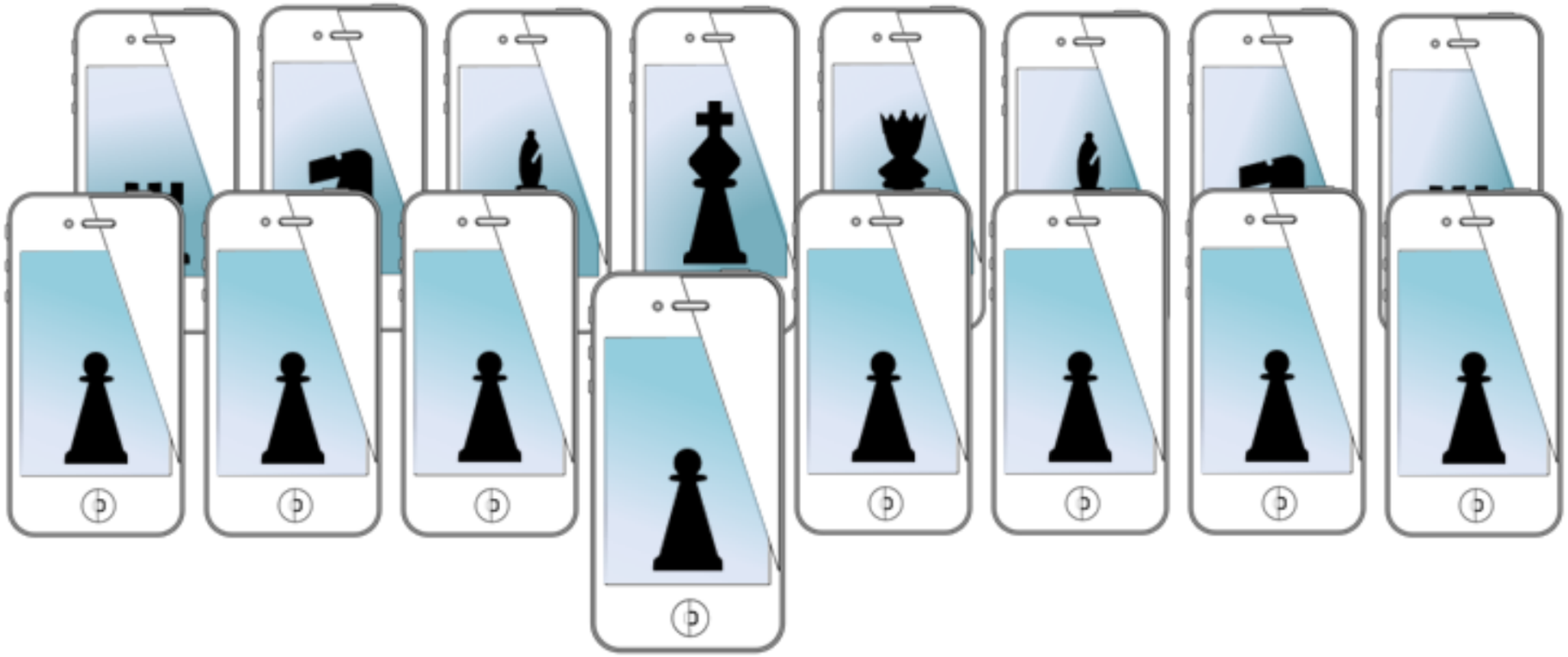
Declare in info.plist

Privacy - Microphone Usage Description	String	This App requires microphone access.

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