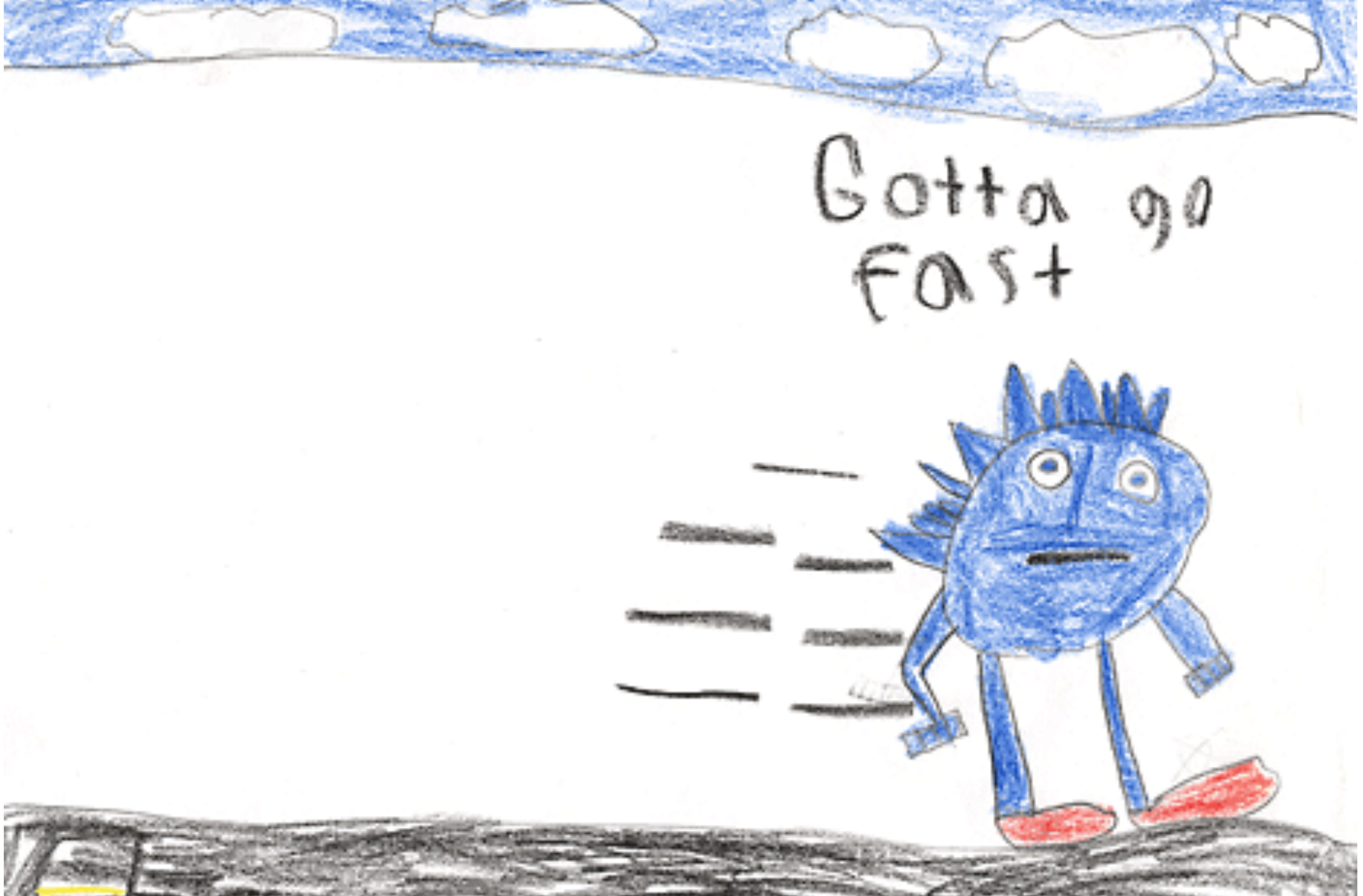
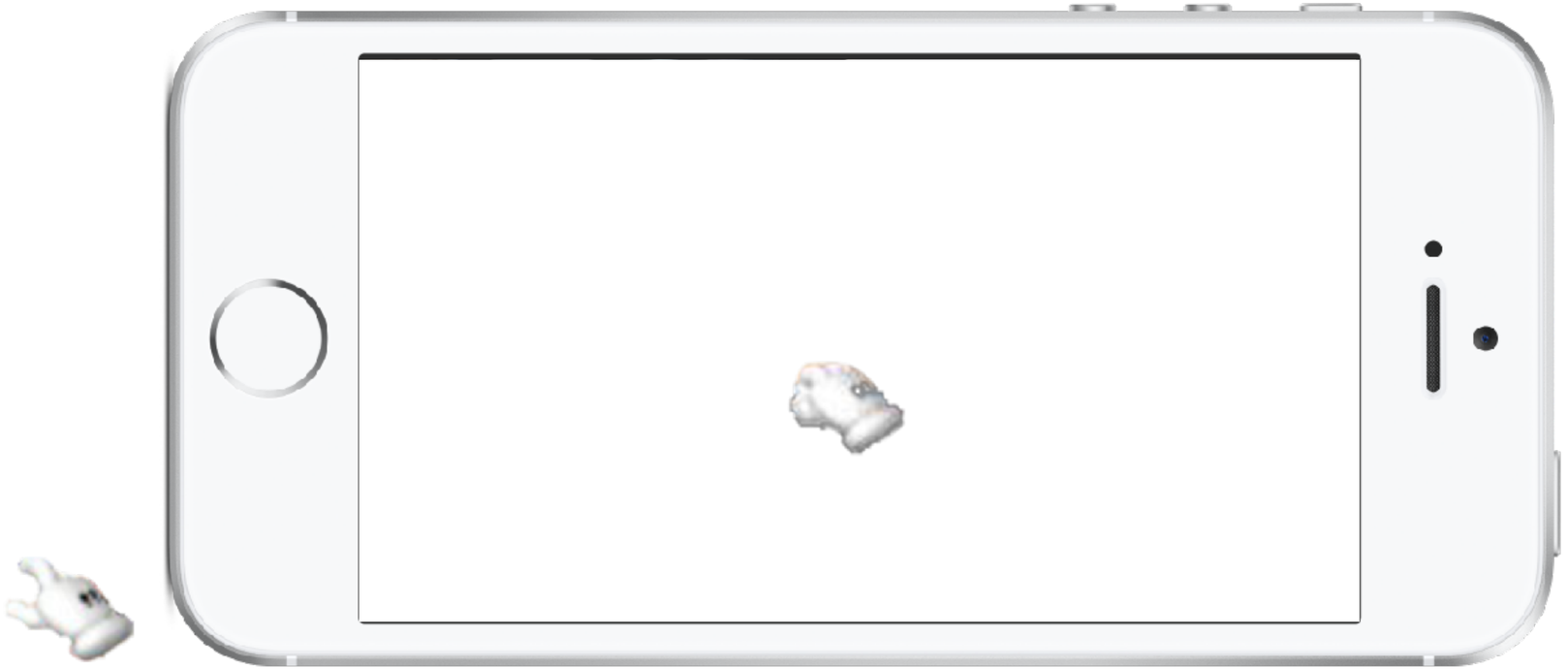
Gotta go fast



Luke Parham

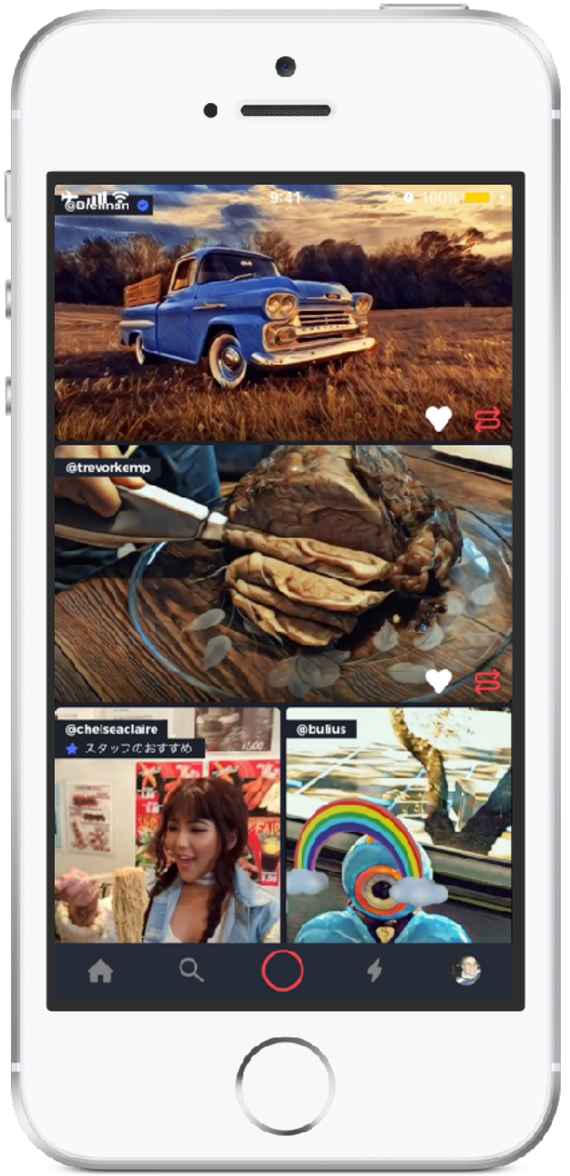
iOS Engineer at Fyusion

Tutorial Writer for RayWenderlich.com



UIKonf

@lukeparham



UIKonf

@lukeparham

Why Care?

UIKonf

"Well, let's say you can shave 10 seconds off of the boot time. Multiply that by five million users and thats 50 million seconds, every single day.

Over a year, that's probably dozens of lifetimes. So if you make it boot ten seconds faster, you've saved a dozen lives. That's really worth it, don't you think?"

The Game Plan

Noticing Dropped Frames

- Using CADisplayLink and Maths

Tools of The Trade

- Time Profiler
- Core Animation Instrument
- Activity Trace

Bonus Level!

- Extra Performance Tips and Tricks

DROPPED FRAMES

Performance on iOS

Get off of main

- The Main Thread is responsible for...
  - Accepting User Input
  - Displaying Results to the screen
- The Goal should be 60 fps on devices you support
  - ~16.67 ms per frame*
- Speed vs Perceived Speed (Responsiveness)
  - Dropped frames and blocking interactions create a perceived slowness
  - Responsiveness is often more important

UIKonf

@lukeparham

Rule #1:

Use a CADisplayLink to track dropped frames

Use a CADisplayLink

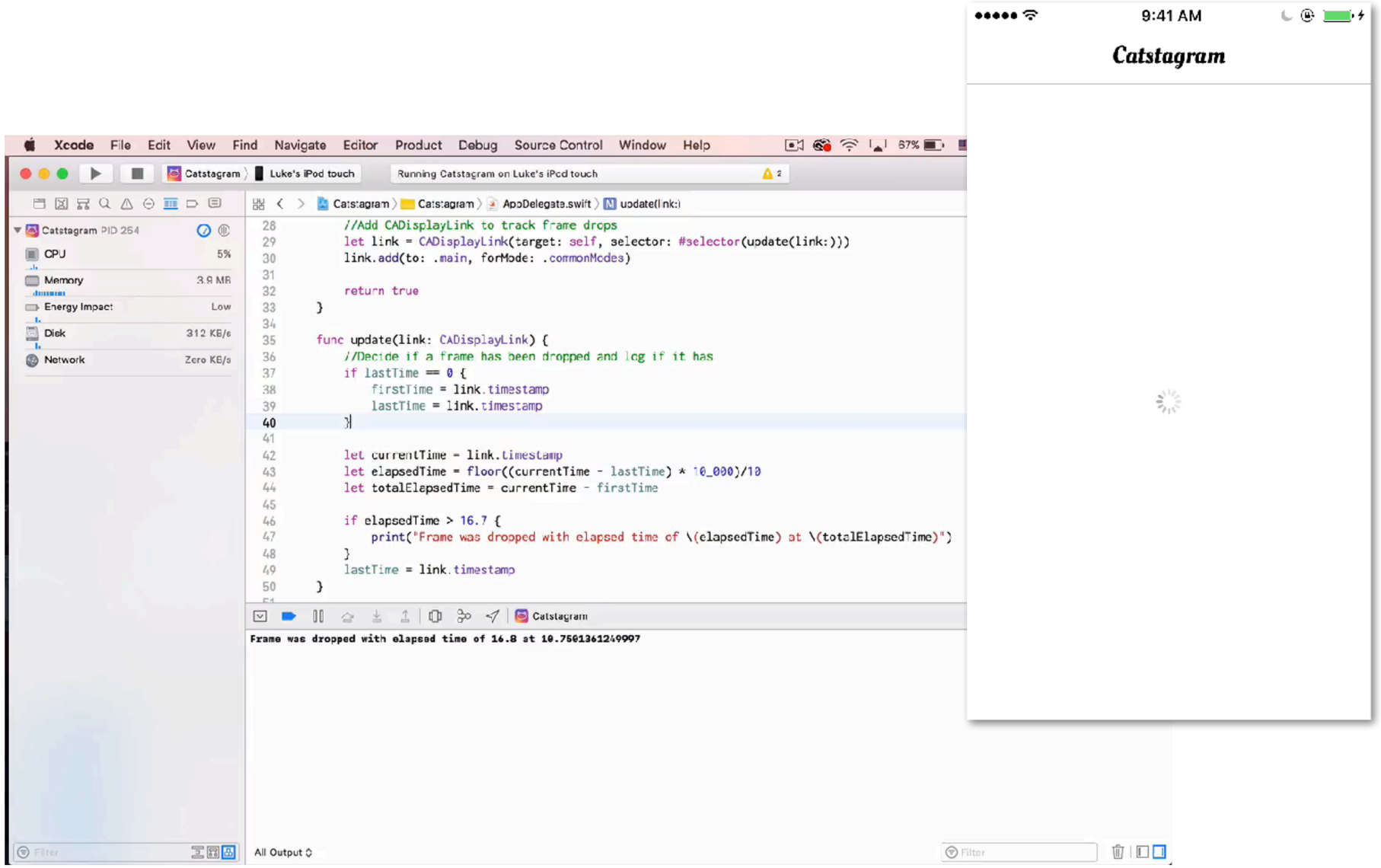
Or else...

A Special Timer that fires on the Vsync

- Aka when the screen refreshes

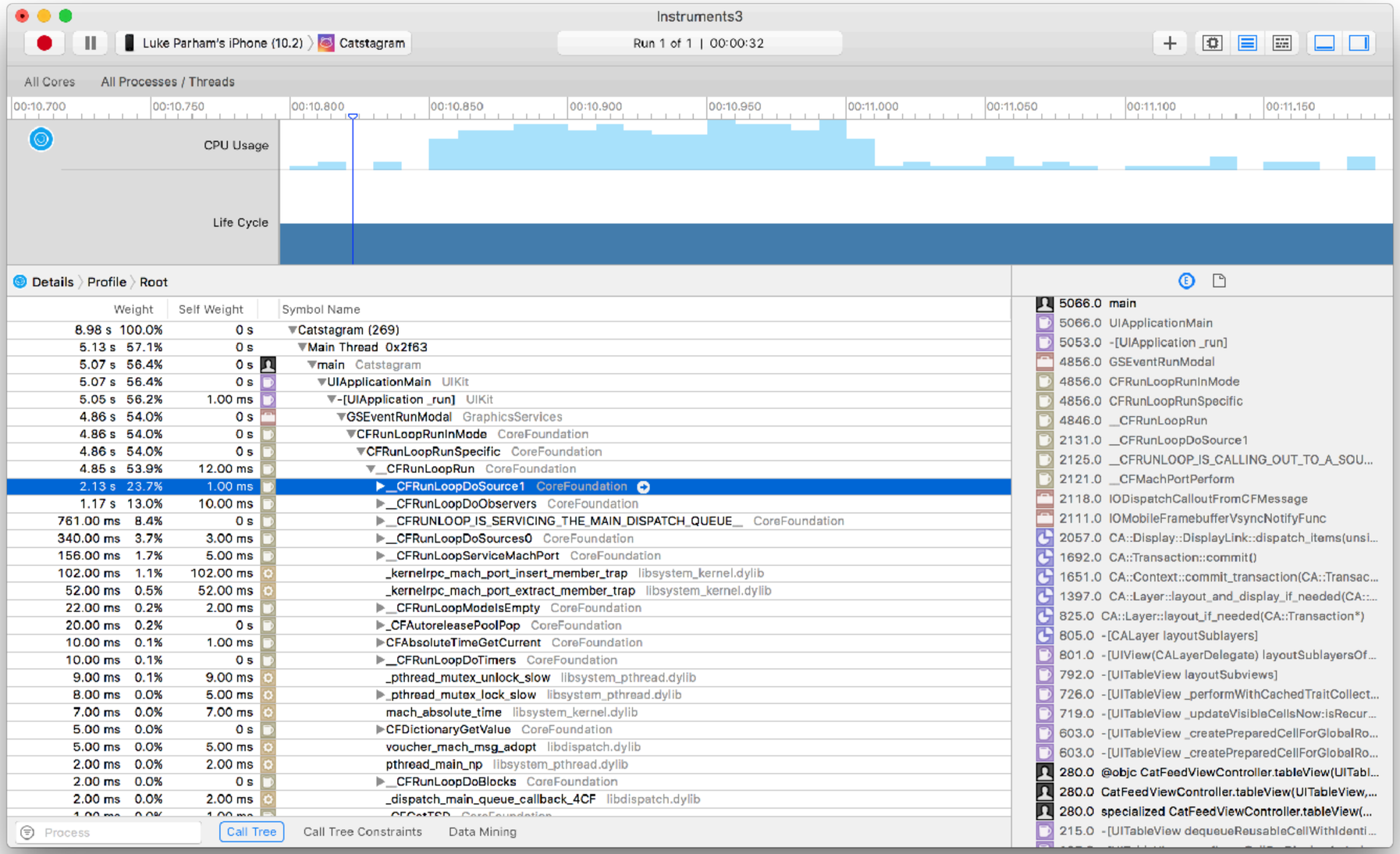
```
let link = CADisplayLink(target: self, selector: #selector(AppDelegate.update(link:)))
link.add(to: RunLoop.main, forMode: .commonModes)


func update(link: CADisplayLink) {
  if lastTime == 0.0 {
    firstTime = link.timestamp
    lastTime = link.timestamp
  }

  let currentTime = link.timestamp
  let elapsedTime = floor((currentTime - lastTime) * 10_000)/10
  let totalElapsedTime = currentTime - firstTime

  if elapsedTime > 16.7 {
    print("Frame was dropped with elapsed time of \(elapsedTime) at \(totalElapsedTime)")
  }
  lastTime = link.timestamp
}
```

Bad Performance on iOS



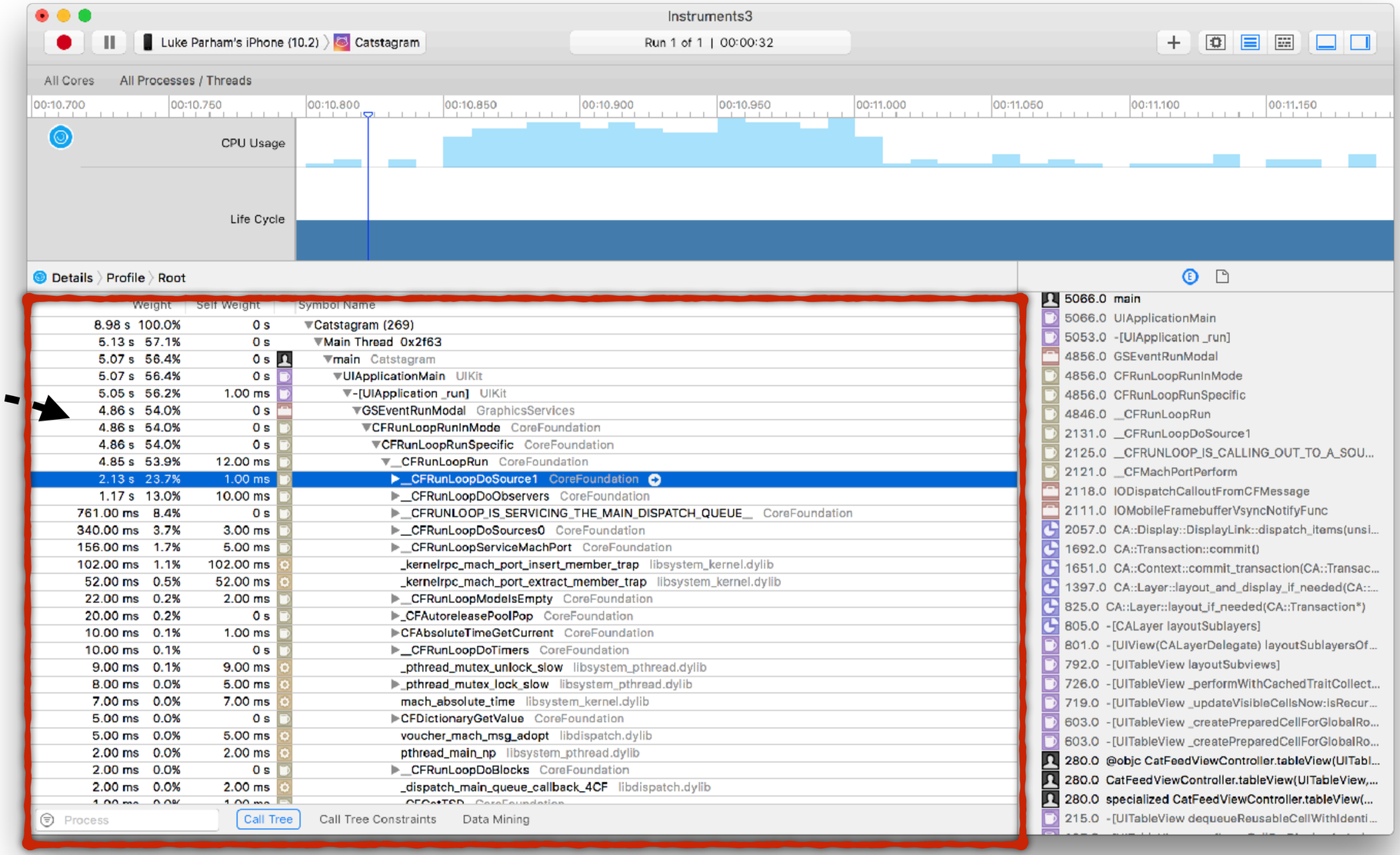
TIME PROFILER

An Example Trace

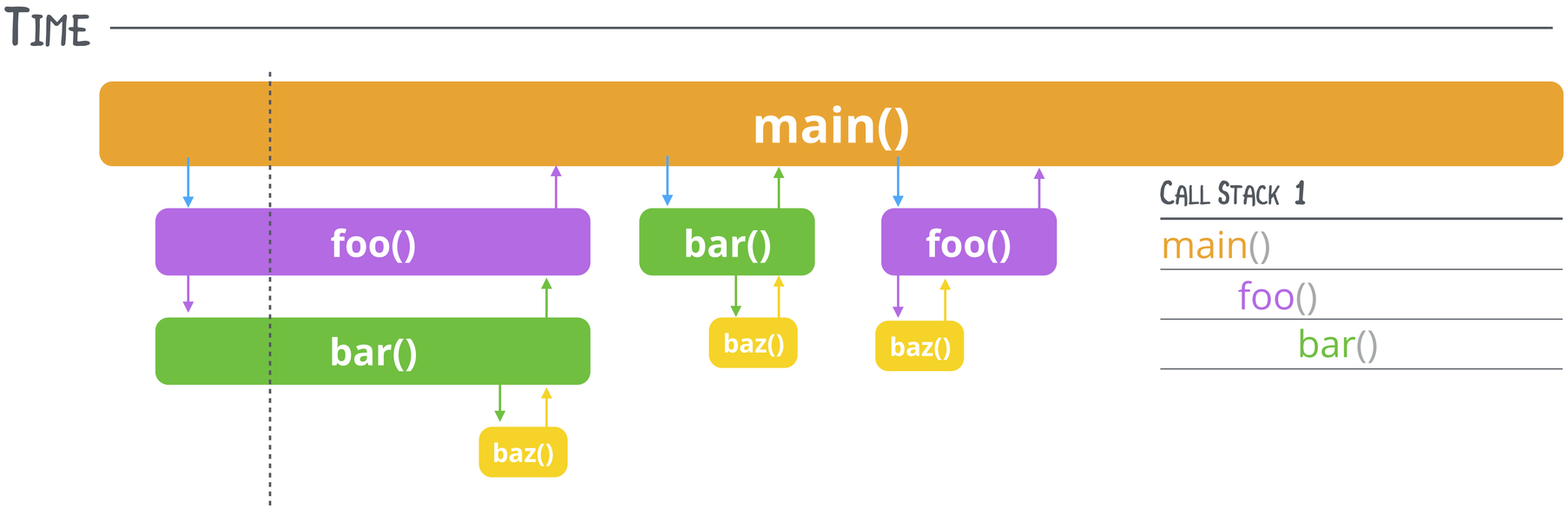


An Example Trace

Call Tree



The Call Tree

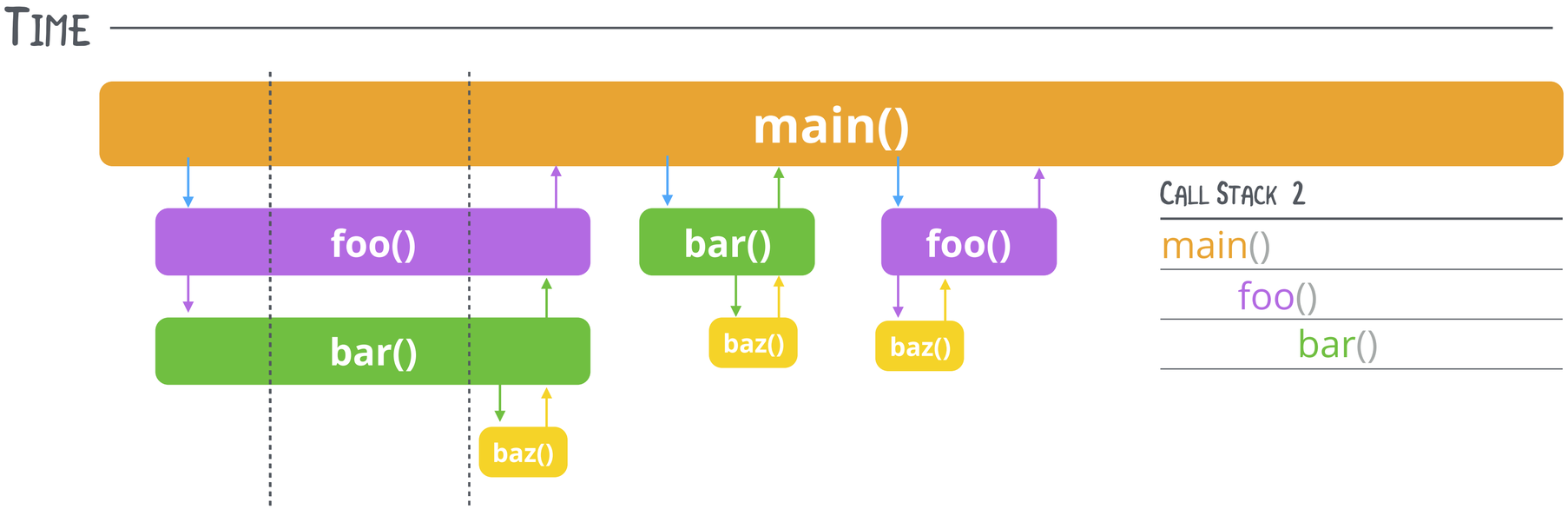


CALL TREE

1
1
1

UIKonf

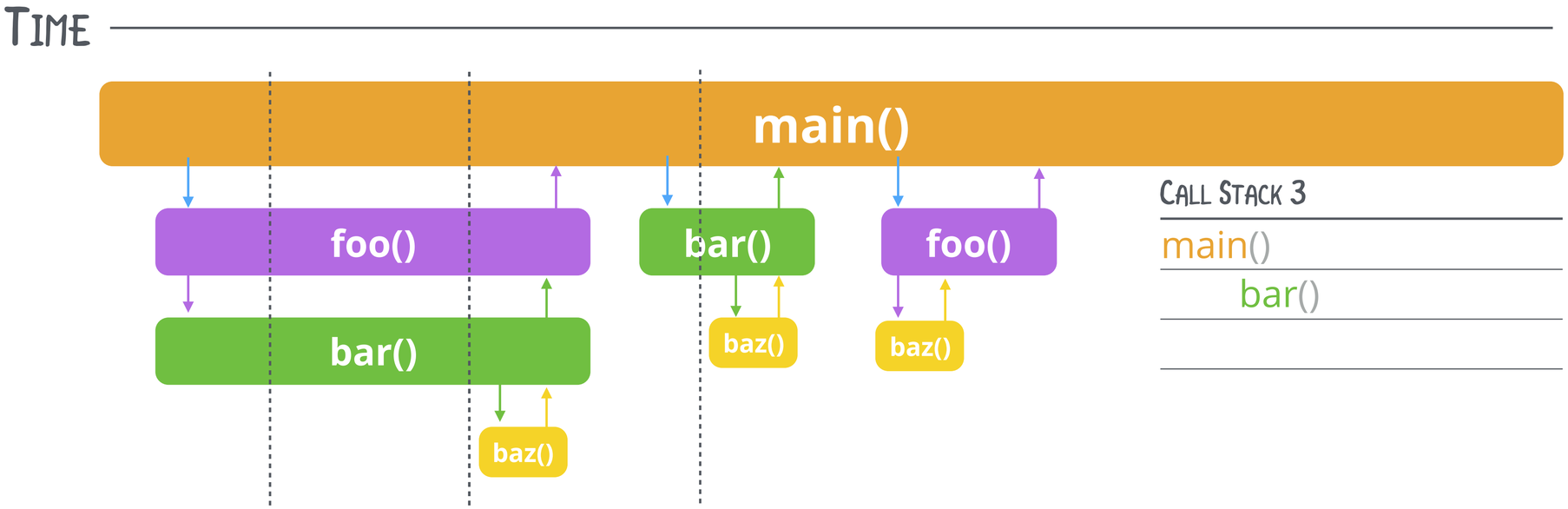
The Call Tree



CALL TREE

1	main()
1	foo()
1	bar()

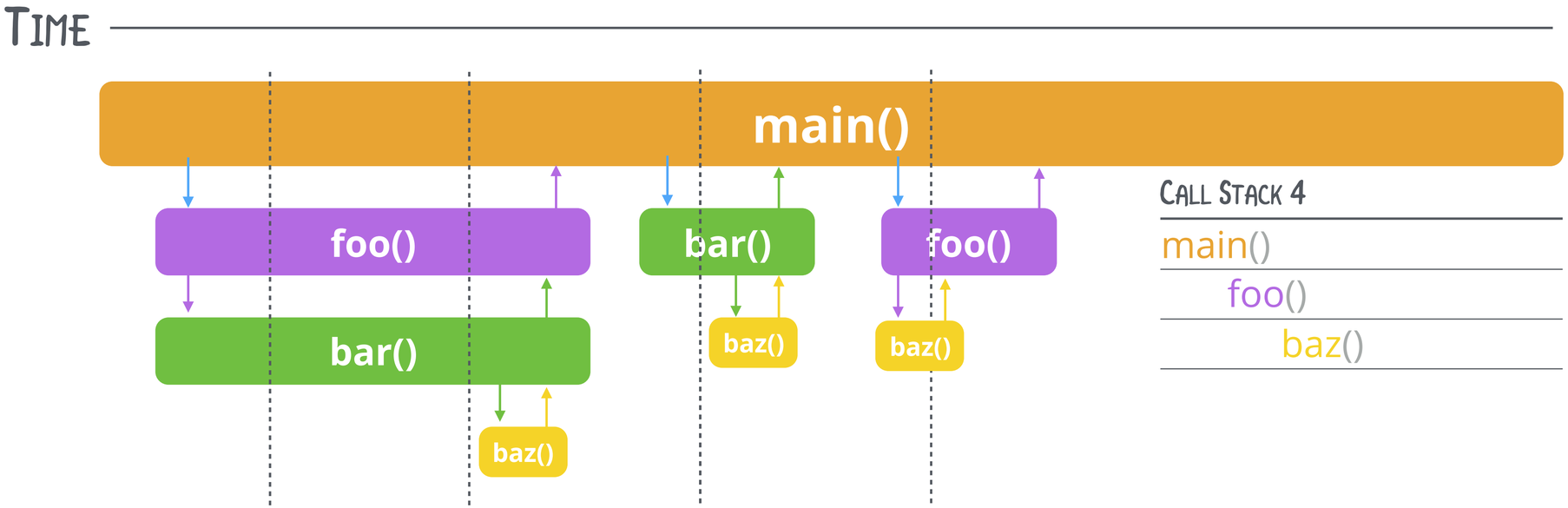
The Call Tree



CALL TREE

3	main()
2	foo()
2	bar()
1	bar()

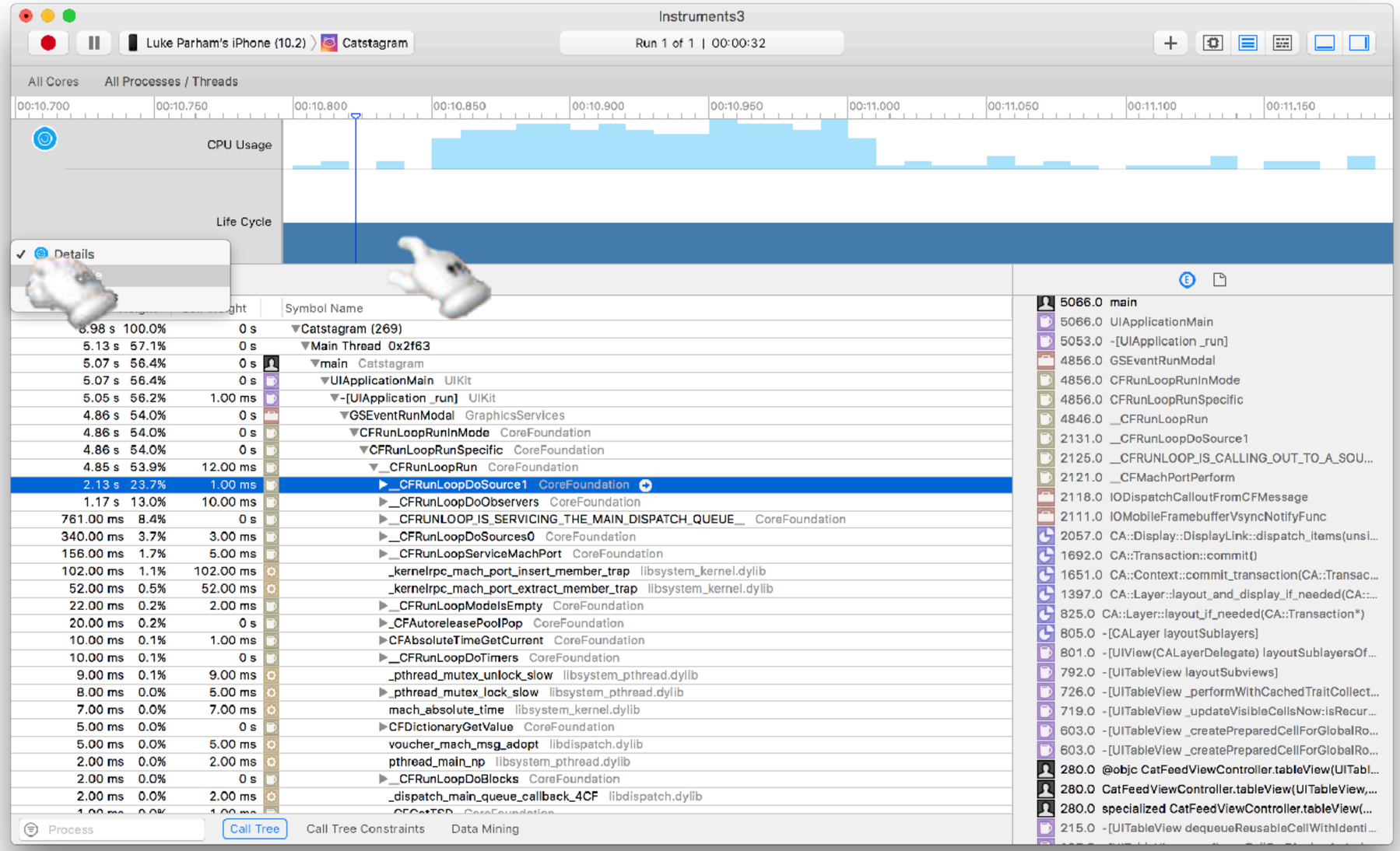
The Call Tree



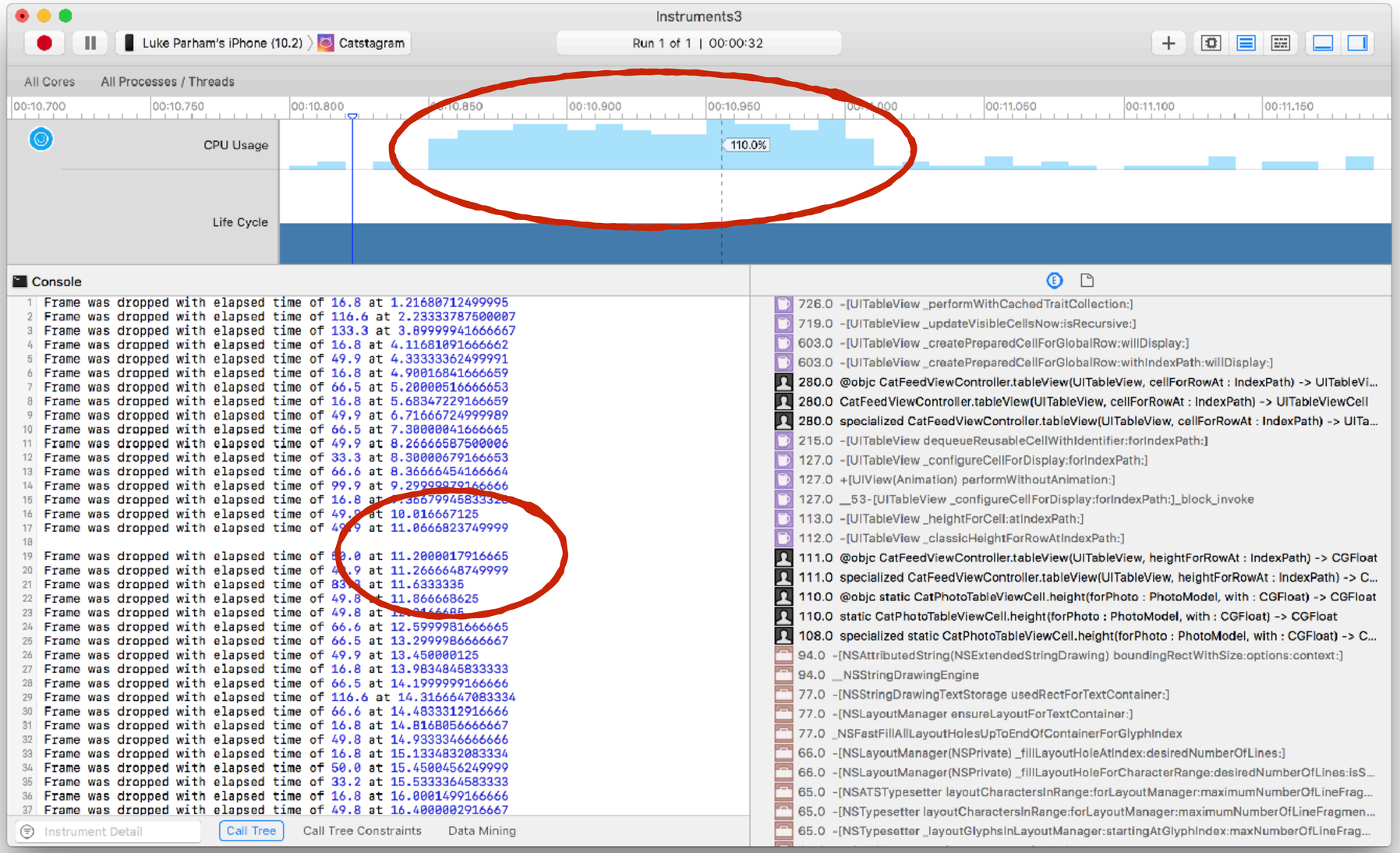
CALL TREE

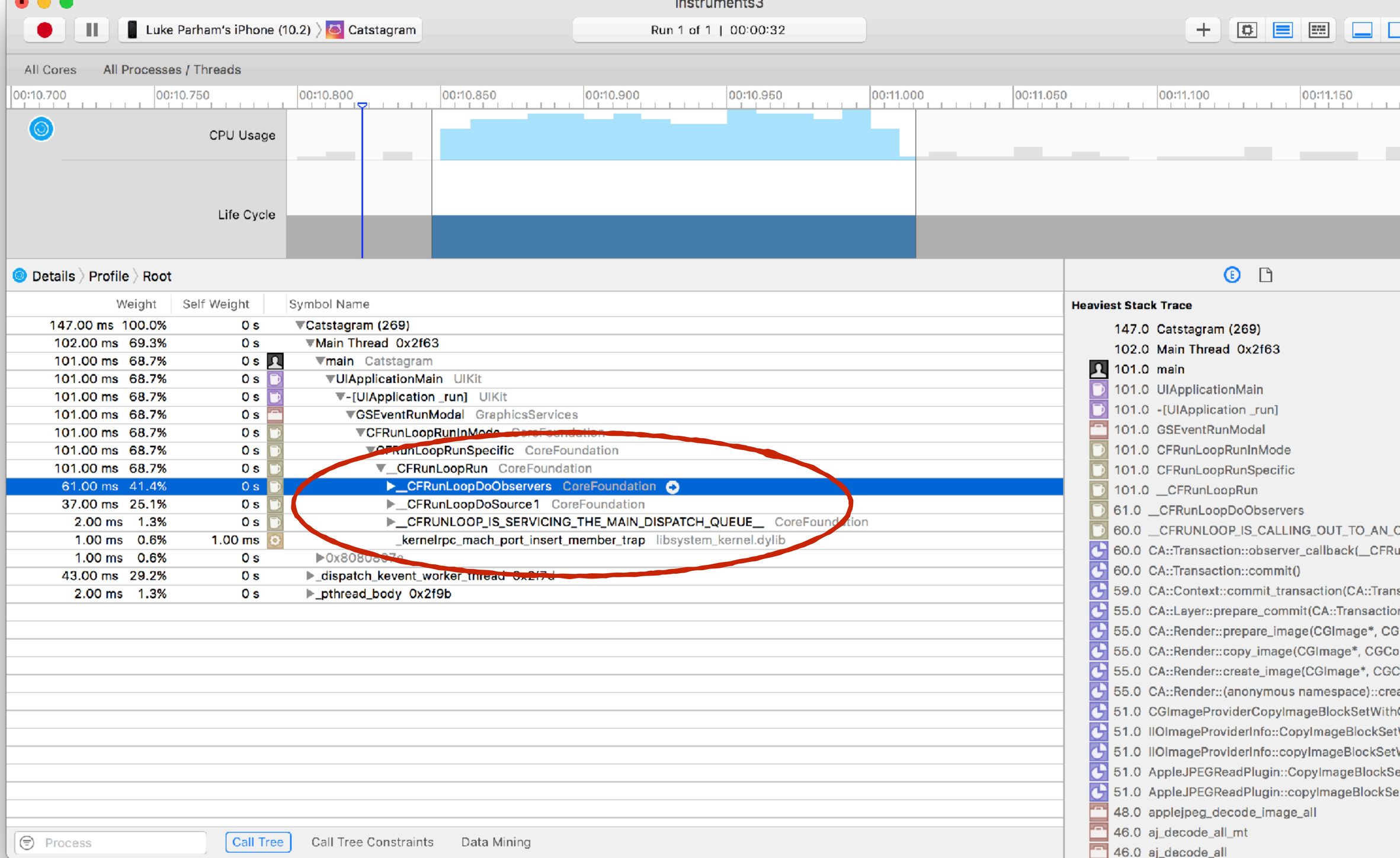
3	main()
2	foo()
2	bar()
1	bar()

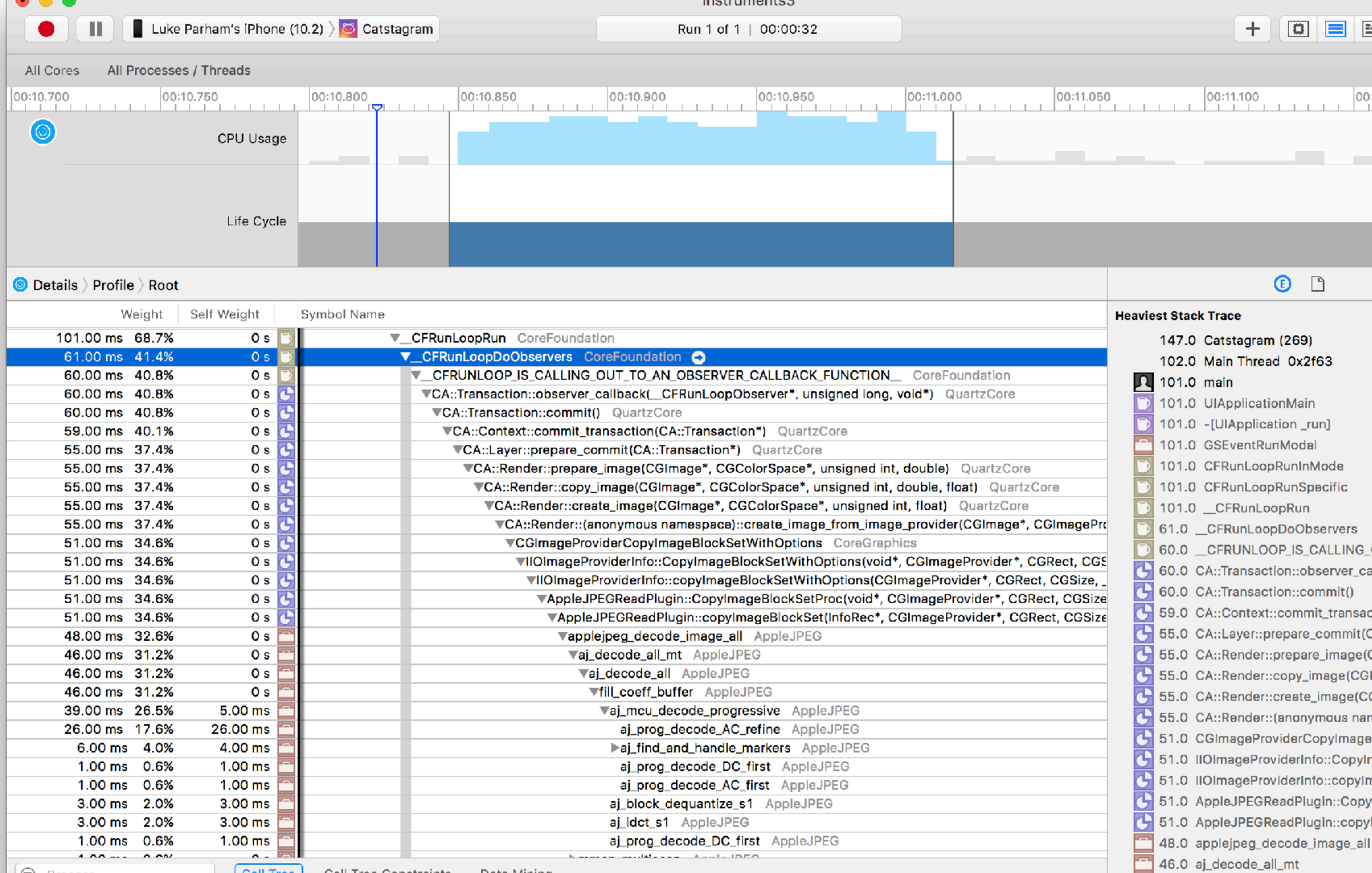
An Example Trace



Dropped Frames in Action




Instruments3
Luke Parham's iPhone (10.2)
Catstagram
Run 1 of 1 | 00:00:32
All Cores
All Processes / Threads
00:10.700
00:10.750
00:10.800
00:10.850
00:10.900
00:10.950
00:11.000
00:11.050
00:11.100
00:11.150
CPU Usage
Life Cycle
Details
Profile
Root
Weight
Self Weight
Symbol Name
147.00 ms 100.0% 0 s Catstagram (269)
102.00 ms 69.3% 0 s Main Thread 0x2f63
101.00 ms 68.7% 0 s main Catstagram
101.00 ms 68.7% 0 s UIApplicationMain UIKit
101.00 ms 68.7% 0 s -[UIApplication _run] UIKit
101.00 ms 68.7% 0 s GSEventRunModal GraphicsServices
101.00 ms 68.7% 0 s CFRunLoopRunInMode
101.00 ms 68.7% 0 s CFRunLoopRunSpecific CoreFoundation
101.00 ms 68.7% 0 s __CFRunLoopRun CoreFoundation
61.00 ms 41.4% 0 s __CFRunLoopDoObservers CoreFoundation
37.00 ms 25.1% 0 s __CFRunLoopDoSource1 CoreFoundation
2.00 ms 1.3% 0 s __CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__ CoreFoundation
1.00 ms 0.6% 1.00 ms _kernelrpc_mach_port_insert_member_trap libsystem_kernel.dylib
1.00 ms 0.6% 0 s 0x8080837a
43.00 ms 29.2% 0 s _dispatch_kevent_worker_thread
2.00 ms 1.3% 0 s _pthread_body 0x2f9b
Heaviest Stack Trace
147.0 Catstagram (269)
102.0 Main Thread 0x2f63
101.0 main
101.0 UIApplicationMain
101.0 -[UIApplication _run]
101.0 GSEventRunModal
101.0 CFRunLoopRunInMode
101.0 CFRunLoopRunSpecific
101.0 __CFRunLoopRun
61.0 __CFRunLoopDoObservers
60.0 __CFRUNLOOP_IS_CALLING_OUT_TO_AN_O
60.0 CA::Transaction::observer_callback(__CFRu
60.0 CA::Transaction::commit()
59.0 CA::Context::commit_transaction(CA::Trans
55.0 CA::Layer::prepare_commit(CA::Transaction
55.0 CA::Render::prepare_image(CGImage*, CG
55.0 CA::Render::copy_image(CGImage*, CGCo
55.0 CA::Render::create_image(CGImage*, CGC
55.0 CA::Render::(anonymous namespace)::crea
51.0 CGImageProviderCopyImageBlockSetWith
51.0 IIOImageProviderInfo::CopyImageBlockSet
51.0 IIOImageProviderInfo::copyImageBlockSet
51.0 AppleJPEGReadPlugin::CopyImageBlockSe
51.0 AppleJPEGReadPlugin::copyImageBlockSe
48.0 applejpeg_decode_image_all
46.0 aj_decode_all_mt
46.0 aj_decode_all
Process
Call Tree
Call Tree Constraints
Data Mining


Luke Parham's iPhone (10.2)
Catstagram
Run 1 of 1 | 00:00:32
All Cores
All Processes / Threads
00:10.700
00:10.750
00:10.800
00:10.850
00:10.900
00:10.950
00:11.000
00:11.050
00:11.100
CPU Usage
Life Cycle
Details
Profile
Root
Weight
Self Weight
Symbol Name
101.00 ms 68.7% 0 s __CFRunLoopRun CoreFoundation
61.00 ms 41.4% 0 s __CFRunLoopDoObservers CoreFoundation
60.00 ms 40.8% 0 s __CFRUNLOOP_IS_CALLING_OUT_TO_AN_OBSERVER_CALLBACK_FUNCTION__ CoreFoundation
60.00 ms 40.8% 0 s CA::Transaction::observer_callback(__CFRunLoopObserver*, unsigned long, void*) QuartzCore
60.00 ms 40.8% 0 s CA::Transaction::commit() QuartzCore
59.00 ms 40.1% 0 s CA::Context::commit_transaction(CA::Transaction*) QuartzCore
55.00 ms 37.4% 0 s CA::Layer::prepare_commit(CA::Transaction*) QuartzCore
55.00 ms 37.4% 0 s CA::Render::prepare_image(CGImage*, CGColorSpace*, unsigned int, double) QuartzCore
55.00 ms 37.4% 0 s CA::Render::copy_image(CGImage*, CGColorSpace*, unsigned int, double, float) QuartzCore
55.00 ms 37.4% 0 s CA::Render::create_image(CGImage*, CGColorSpace*, unsigned int, float) QuartzCore
55.00 ms 37.4% 0 s CA::Render::(anonymous namespace)::create_image_from_image_provider(CGImage*, CGImagePr
51.00 ms 34.6% 0 s CGImageProviderCopyImageBlockSetWithOptions CoreGraphics
51.00 ms 34.6% 0 s IIOImageProviderInfo::CopyImageBlockSetWithOptions(void*, CGImageProvider*, CGRect, CGS
51.00 ms 34.6% 0 s IIOImageProviderInfo::copyImageBlockSetWithOptions(CGImageProvider*, CGRect, CGSize, _
51.00 ms 34.6% 0 s AppleJPEGReadPlugin::CopyImageBlockSetProc(void*, CGImageProvider*, CGRect, CGSize
51.00 ms 34.6% 0 s AppleJPEGReadPlugin::copyImageBlockSet(InfoRec*, CGImageProvider*, CGRect, CGSize
48.00 ms 32.6% 0 s applejpeg_decode_image_all AppleJPEG
46.00 ms 31.2% 0 s aj_decode_all_mt AppleJPEG
46.00 ms 31.2% 0 s aj_decode_all AppleJPEG
46.00 ms 31.2% 0 s fill_coeff_buffer AppleJPEG
39.00 ms 26.5% 5.00 ms aj_mcu_decode_progressive AppleJPEG
26.00 ms 17.6% 26.00 ms aj_prog_decode_AC_refine AppleJPEG
6.00 ms 4.0% 4.00 ms aj_find_and_handle_markers AppleJPEG
1.00 ms 0.6% 1.00 ms aj_prog_decode_DC_first AppleJPEG
1.00 ms 0.6% 1.00 ms aj_prog_decode_AC_first AppleJPEG
3.00 ms 2.0% 3.00 ms aj_block_dequantize_s1 AppleJPEG
3.00 ms 2.0% 3.00 ms aj_idct_s1 AppleJPEG
1.00 ms 0.6% 1.00 ms aj_prog_decode_DC_first AppleJPEG
Heaviest Stack Trace
147.0 Catstagram (269)
102.0 Main Thread 0x2f63
101.0 main
101.0 UIApplicationMain
101.0 -[UIApplication _run]
101.0 GSEventRunModal
101.0 CFRunLoopRunInMode
101.0 CFRunLoopRunSpecific
101.0 __CFRunLoopRun
61.0 __CFRunLoopDoObservers
60.0 __CFRUNLOOP_IS_CALLING_
60.0 CA::Transaction::observer_ca
60.0 CA::Transaction::commit()
59.0 CA::Context::commit_transac
55.0 CA::Layer::prepare_commit(C
55.0 CA::Render::prepare_image(C
55.0 CA::Render::copy_image(CG
55.0 CA::Render::create_image(C
55.0 CA::Render::(anonymous nam
51.0 CGImageProviderCopyImage
51.0 IIOImageProviderInfo::CopyIm
51.0 IIOImageProviderInfo::copyIm
51.0 AppleJPEGReadPlugin::Copy
51.0 AppleJPEGReadPlugin::copyI
48.0 applejpeg_decode_image_all
46.0 aj_decode_all_mt

Details 〉 Profile 〉 Root

Weight	Self Weight	Symbol Name
101.00 ms 68.7%	0 s	▼__CFRunLoopRun CoreFoundation
61.00 ms 41.4%	0 s	▼__CFRunLoopDoObservers CoreFoundation
60.00 ms 40.8%	0 s	▼__CFRUNLOOP_IS_CALLING_OUT_TO_AN_OBSERVER_CALLBACK_FUNCTION__ CoreFoundation
60.00 ms 40.8%	0 s	▼CA::Transaction::observer_callback(__CFRunLoopObserver*, unsigned long, void*) QuartzCore
60.00 ms 40.8%	0 s	▼CA::Transaction::commit() QuartzCore
59.00 ms 40.1%	0 s	▼CA::Context::commit_transaction(CA::Transaction*) QuartzCore
55.00 ms 37.4%	0 s	▼CA::Layer::prepare_commit(CA::Transaction*) QuartzCore
55.00 ms 37.4%	0 s	▼CA::Render::prepare_image(CGImage*, CGColorSpace*, unsigned int, double) QuartzCore
55.00 ms 37.4%	0 s	▼CA::Render::copy_image(CGImage*, CGColorSpace*, unsigned int, double, float) QuartzCore
55.00 ms 37.4%	0 s	▼CA::Render::create_image(CGImage*, CGColorSpace*, unsigned int, float) QuartzCore
55.00 ms 37.4%	0 s	▼CA::Render::(anonymous namespace)::create_image_from_image_provider(CGImage*, CGImagePro
51.00 ms 34.6%	0 s	▼CGImageProviderCopyImageBlockSetWithOptions CoreGraphics
51.00 ms 34.6%	0 s	▼IIOImageProviderInfo::CopyImageBlockSetWithOptions(void*, CGImageProvider*, CGRect, CGS
51.00 ms 34.6%	0 s	▼IIOImageProviderInfo::copyImageBlockSetWithOptions(CGImageProvider*, CGRect, CGSize, _
51.00 ms 34.6%	0 s	▼AppleJPEGReadPlugin::CopyImageBlockSetProc(void*, CGImageProvider*, CGRect, CGSize
51.00 ms 34.6%	0 s	▼AppleJPEGReadPlugin::copyImageBlockSet(InfoRec*, CGImageProvider*, CGRect, CGSize
48.00 ms 32.6%	0 s	▼applejpeg_decode_image_all AppleJPEG
[illegible]	0 s	▼aj_decode_all_mt AppleJPEG
46.00 ms 31.2%	0 s	▼aj_decode_all AppleJPEG
46.00 ms 31.2%	0 s	▼fill_coeff_buffer AppleJPEG
39.00 ms 26.5%	5.00 ms	▼aj_mcu_decode_progressive AppleJPEG
26.00 ms 17.6%	26.00 ms	aj_prog_decode_AC_refine AppleJPEG
6.00 ms 4.0%	4.00 ms	▶aj_find_and_handle_markers AppleJPEG
1.00 ms 0.6%	1.00 ms	aj_prog_decode_DC_first AppleJPEG
1.00 ms 0.6%	1.00 ms	aj_prog_decode_AC_first AppleJPEG
3.00 ms 2.0%	3.00 ms	aj_block_dequantize_s1 AppleJPEG
3.00 ms 2.0%	3.00 ms	aj_idct_s1 AppleJPEG
1.00 ms 0.6%	1.00 ms	aj_prog_decode_DC_first AppleJPEG

Heaviest Stack Trace

147.0 Catsta
102.0 Main
101.0 main
101.0 UIApp
101.0 -[UIAp
101.0 GSEve
101.0 CFRun
101.0 CFRun
101.0 __CFR
61.0 __CFRu
60.0 __CFRU
60.0 CA::Tra
60.0 CA::Tra
59.0 CA::Co
55.0 CA::Lay
55.0 CA::Re
55.0 CA::Re
55.0 CA::Re
55.0 CA::Re
51.0 CGImag
51.0 IIOImag
51.0 IIOImag
51.0 AppleJ
51.0 AppleJ
48.0 applejp
46.0 aj_deco
46.0 aj_deco

Process | Call Tree | Call Tree Constraints | Data Mining

Rule #2:

Decode JPEGs in the Background

Manually Decoding JPEGs

```
extension UIImage {

  func decodedImage() -> UIImage? {
    guard let newImage = self.cgImage else { return nil }

    let colorSpace = CGColorSpaceCreateDeviceRGB()
    let context = CGContext(data: nil,
                            width: newImage.width,
                            height: newImage.height,
                            bitsPerComponent: 8,
                            bytesPerRow: newImage.width * 4,
                            space: colorSpace,
                            bitmapInfo: CGImageAlphaInfo.noneSkipFirst.rawValue)

    context?.draw(newImage, in: CGRect(x: 0, y: 0,
                                       width: newImage.width,
                                       height: newImage.height))

    let decodedImage = context?.makeImage()

    if let decodedImage = decodedImage {
      return UIImage(cgImage: decodedImage)
    }
    return nil
  }
}
```

Manually Decoding JPEGs

- Technically less efficient than letting the imageView do things
  - But...

```
DispatchQueue.global(qos: .userInitiated).async {
    let decodedImage = image.decodedImage()
    DispatchQueue.main.async {
        imageView.layer.contents = decodedImage?.cgImage
    }
}
```

Not Crashing

If you're into that sort of thing

- Memory warnings are handled on main
  - Allocating in the background can get your app killed…
  - Instead, wait on main just in case

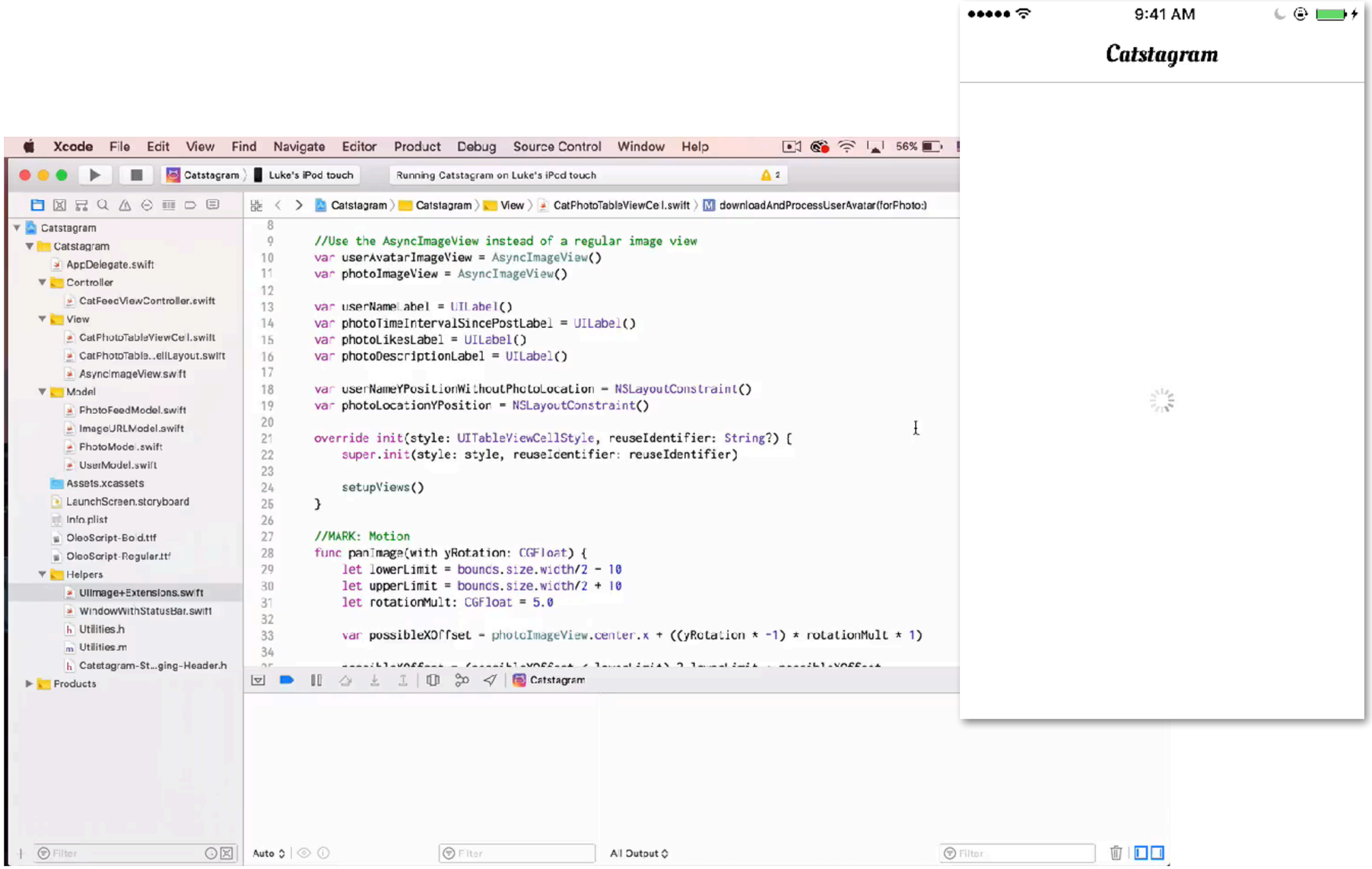
Objective-C

```
dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INTERACTIVE, 0), ^{
    [self performSelectorOnMainThread:@selector(waitWhileProcessingMemoryWarnings)
                           withObject:nil
                        waitUntilDone:YES];
    //load images...
});
```

Swift

```
DispatchQueue.global(qos: .userInitiated).async {
    DispatchQueue.main.sync { }
    //load images...
}
```

Better Performance on iOS



The Usual Suspects

Nothing new under the sun

JPEG Decoding

- UIImageViews will do this for you lazily

Text Measurement

- NSAttributedString's -boundingRectWithSize:

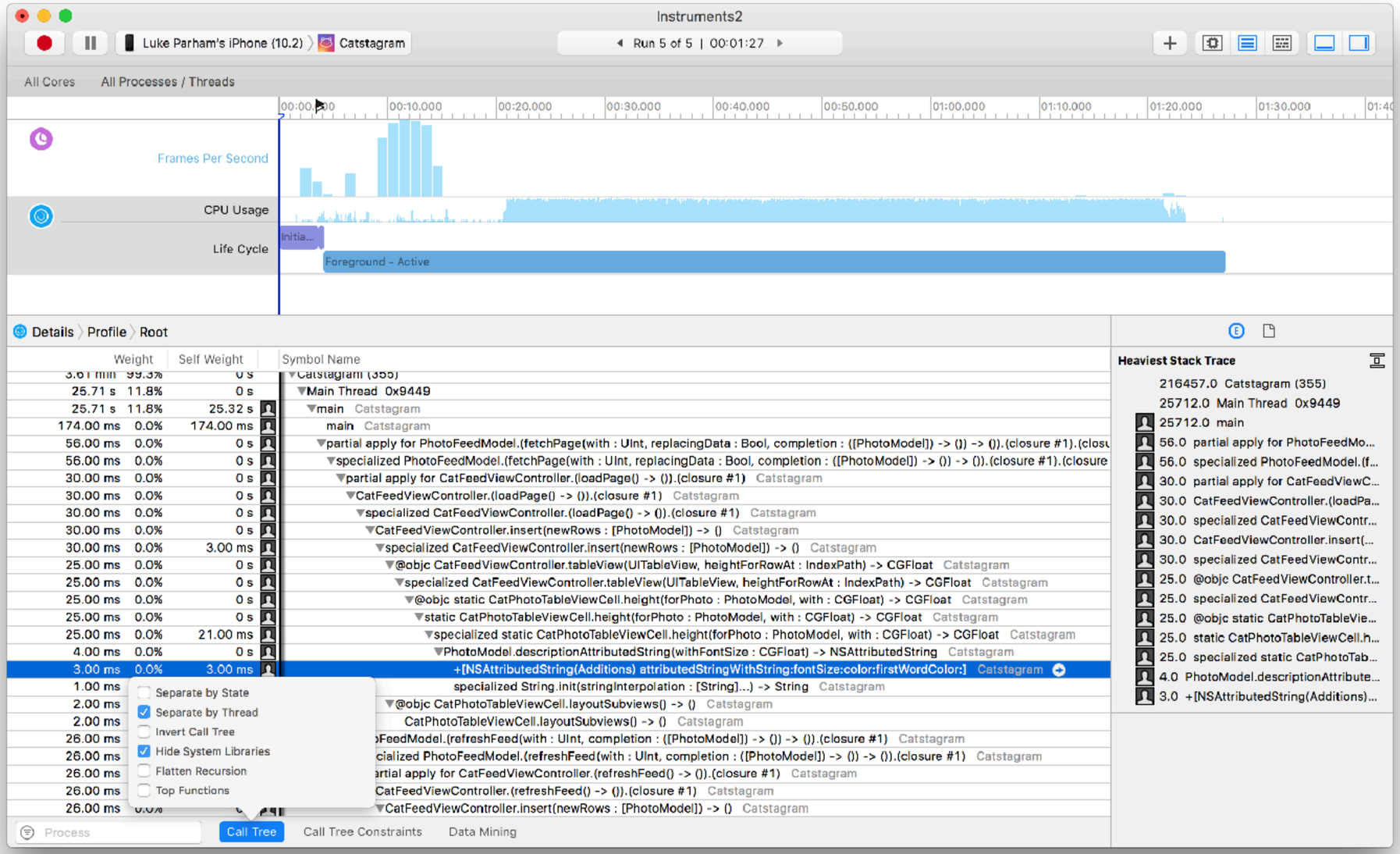
Hierarchy Layout

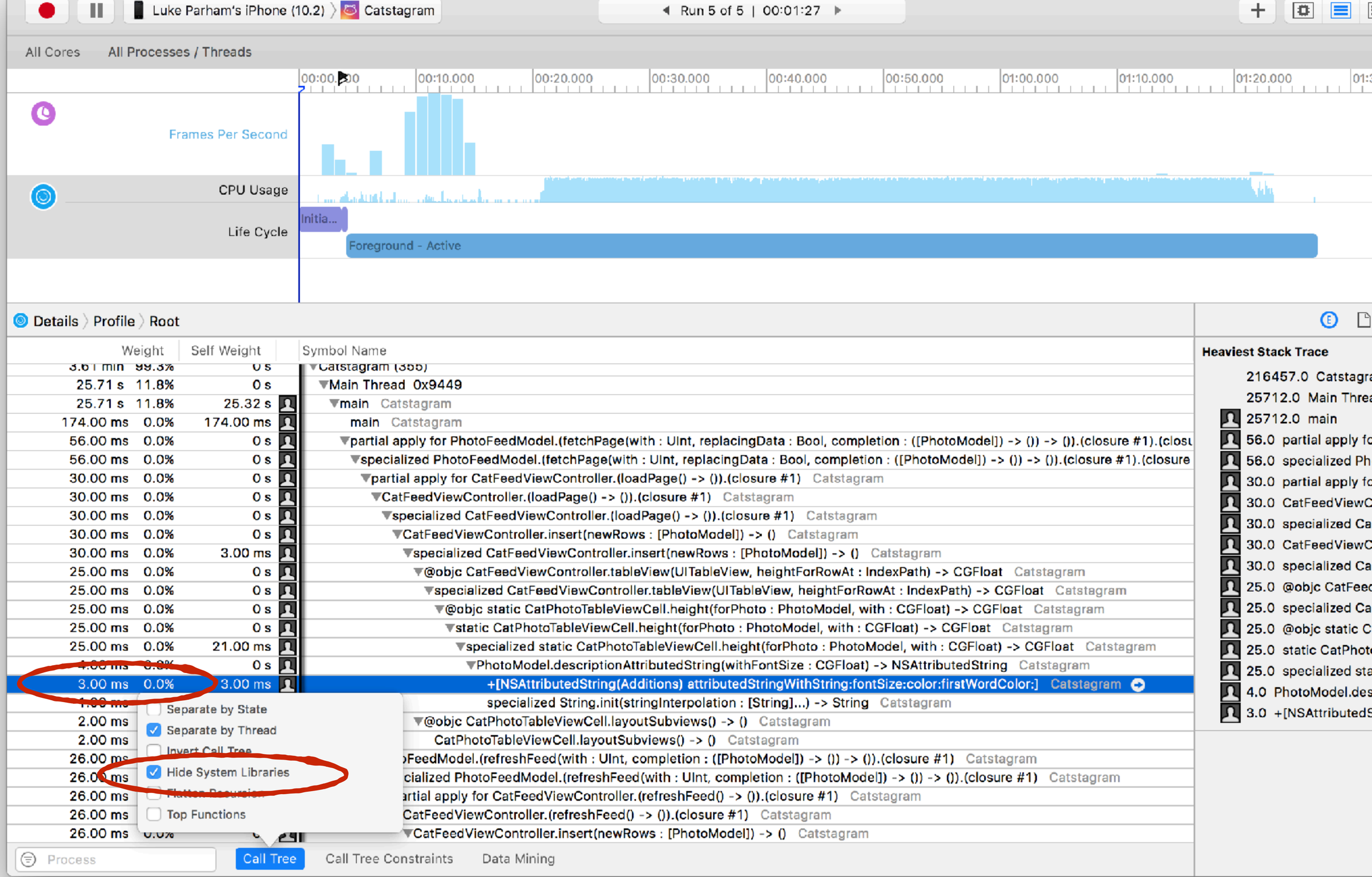
- Especially when using AutoLayout

Object Allocation

- Especially in very tight loops

Text Measurement




Luke Parham's iPhone (10.2)
Catstagram
Run 5 of 5 | 00:01:27
All Cores
All Processes / Threads
00:00.000
00:10.000
00:20.000
00:30.000
00:40.000
00:50.000
01:00.000
01:10.000
01:20.000
Frames Per Second
CPU Usage
Life Cycle
Initia...
Foreground - Active
Details
Profile
Root
Weight
Self Weight
Symbol Name
Heaviest Stack Trace
Separate by State
Separate by Thread
Invert Call Tree
Hide System Libraries
Top Functions
Process
Call Tree
Call Tree Constraints
Data Mining

Summary

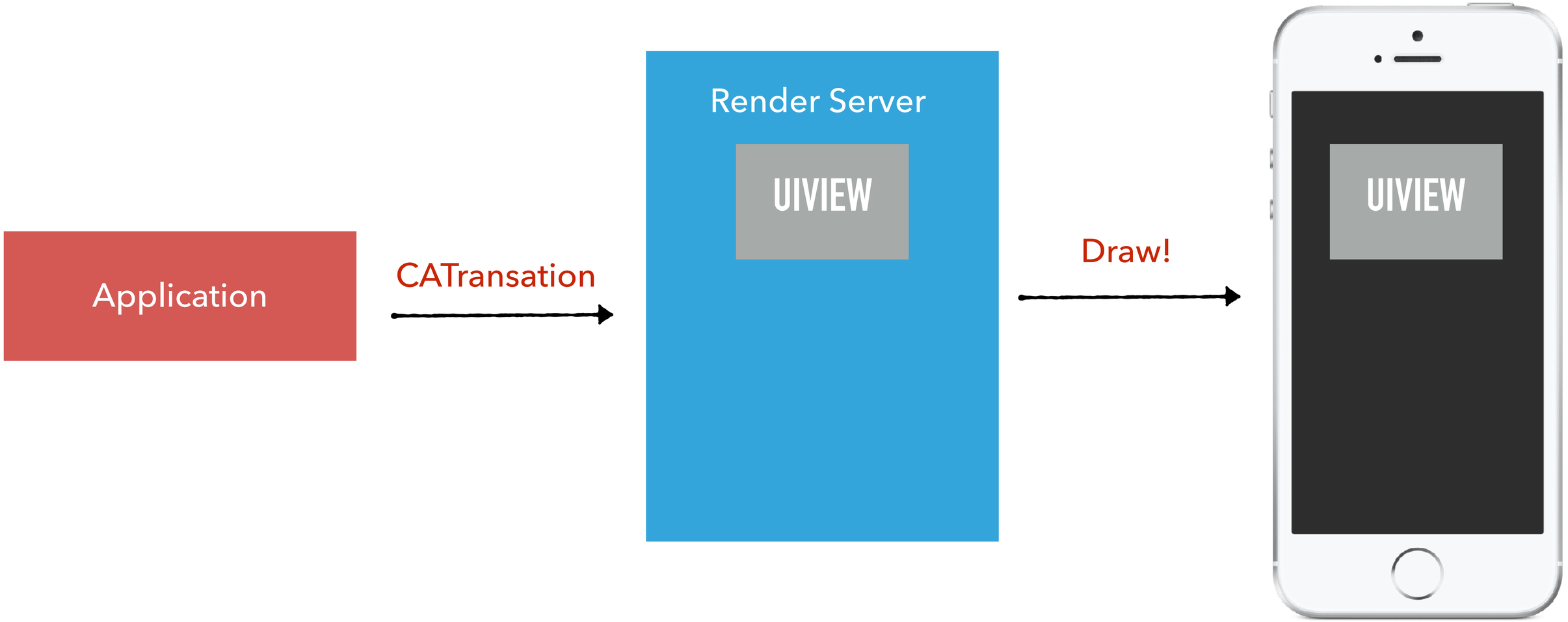
- If you can do work in the background, you probably should
  - Just make sure it’s not UIKit related
  - In that case use CoreGraphics etc.
- Ignore Apple’s code at your own peril
- Don’t forget about Memory Warnings

CORE ANIMATION

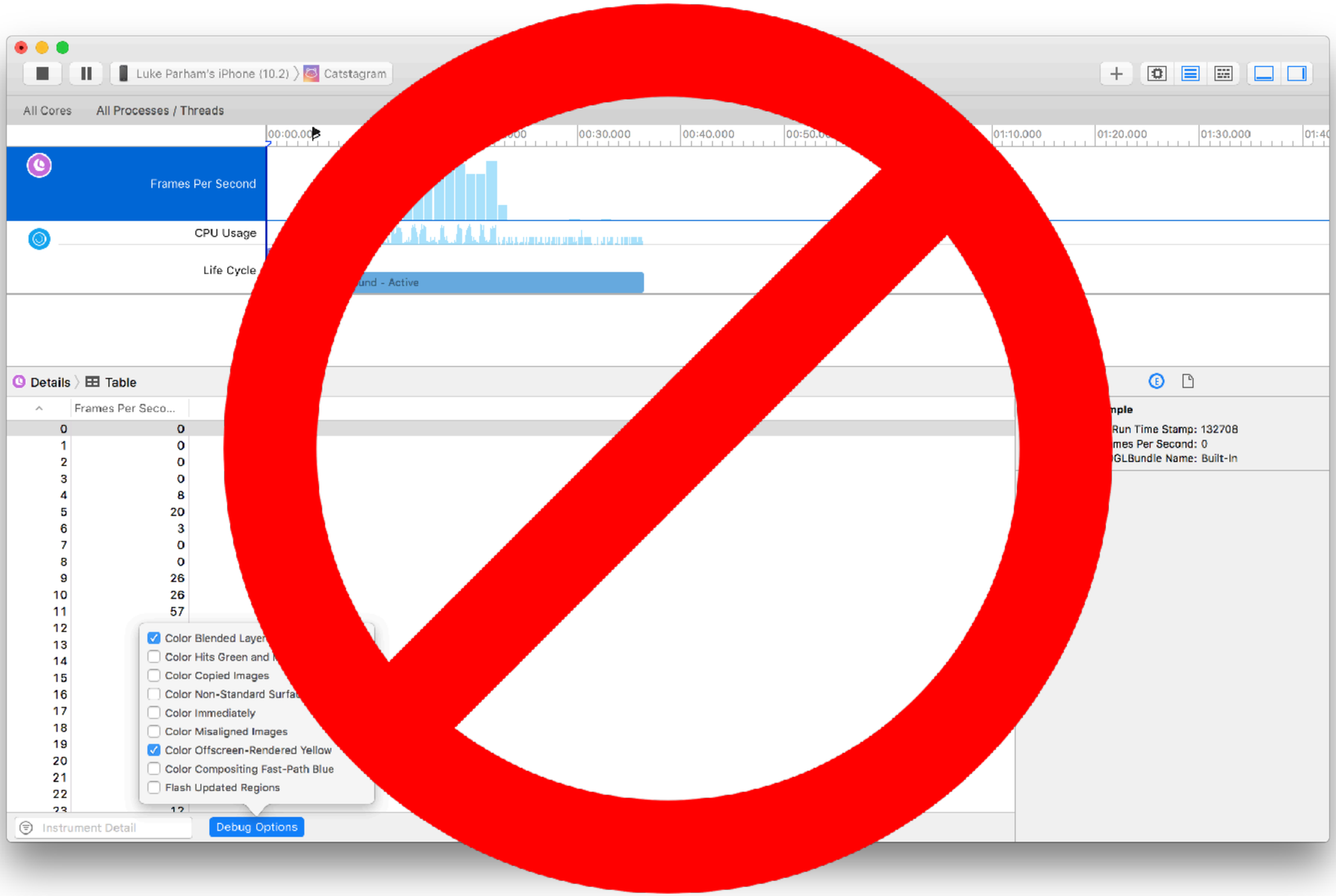
The Render Server

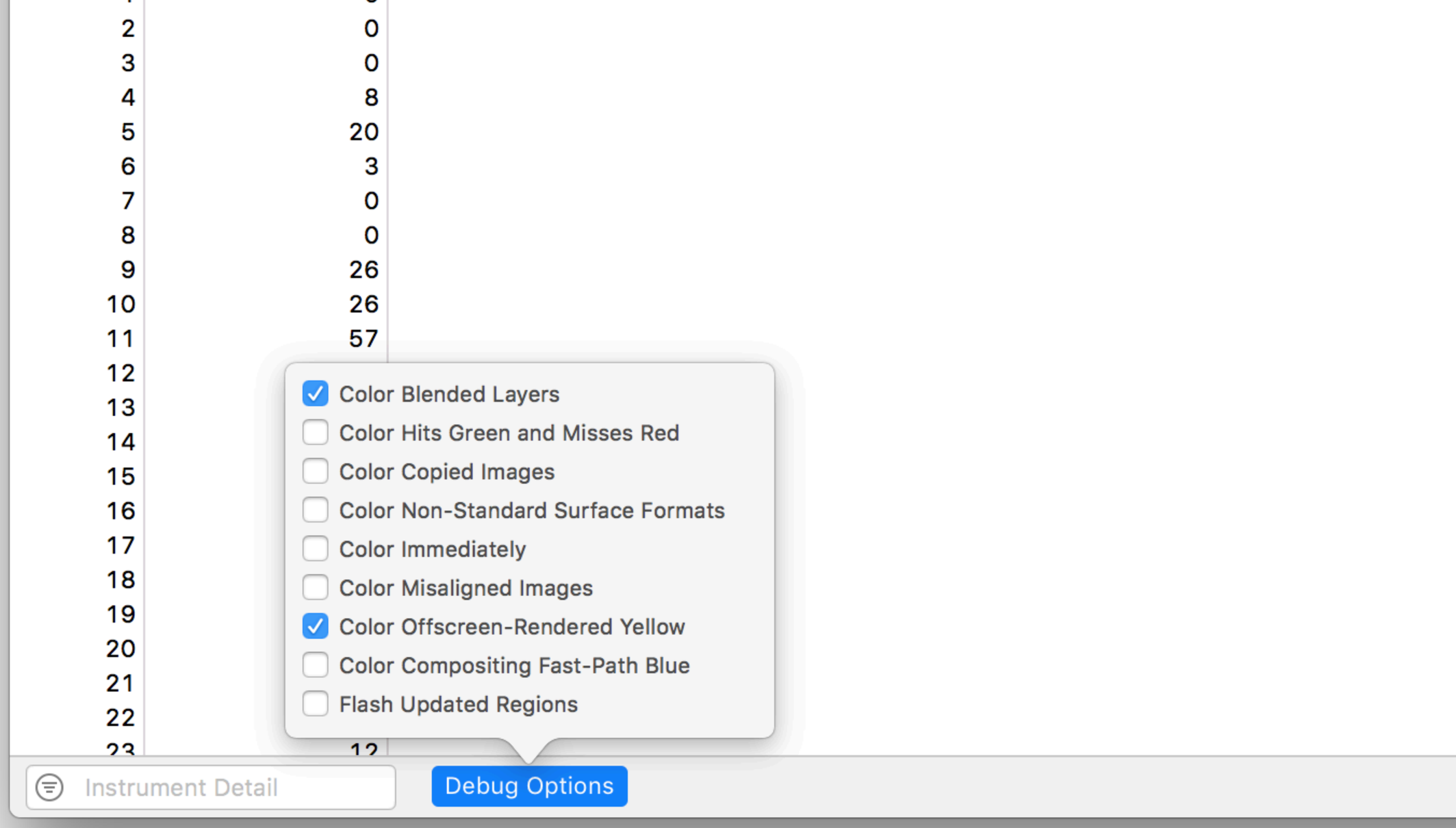
- All UIViews implicitly cause a **CATransaction**.
- The **Render Server** is an out-of-process service that maintains a copy of your view hierarchy and draws to the screen.

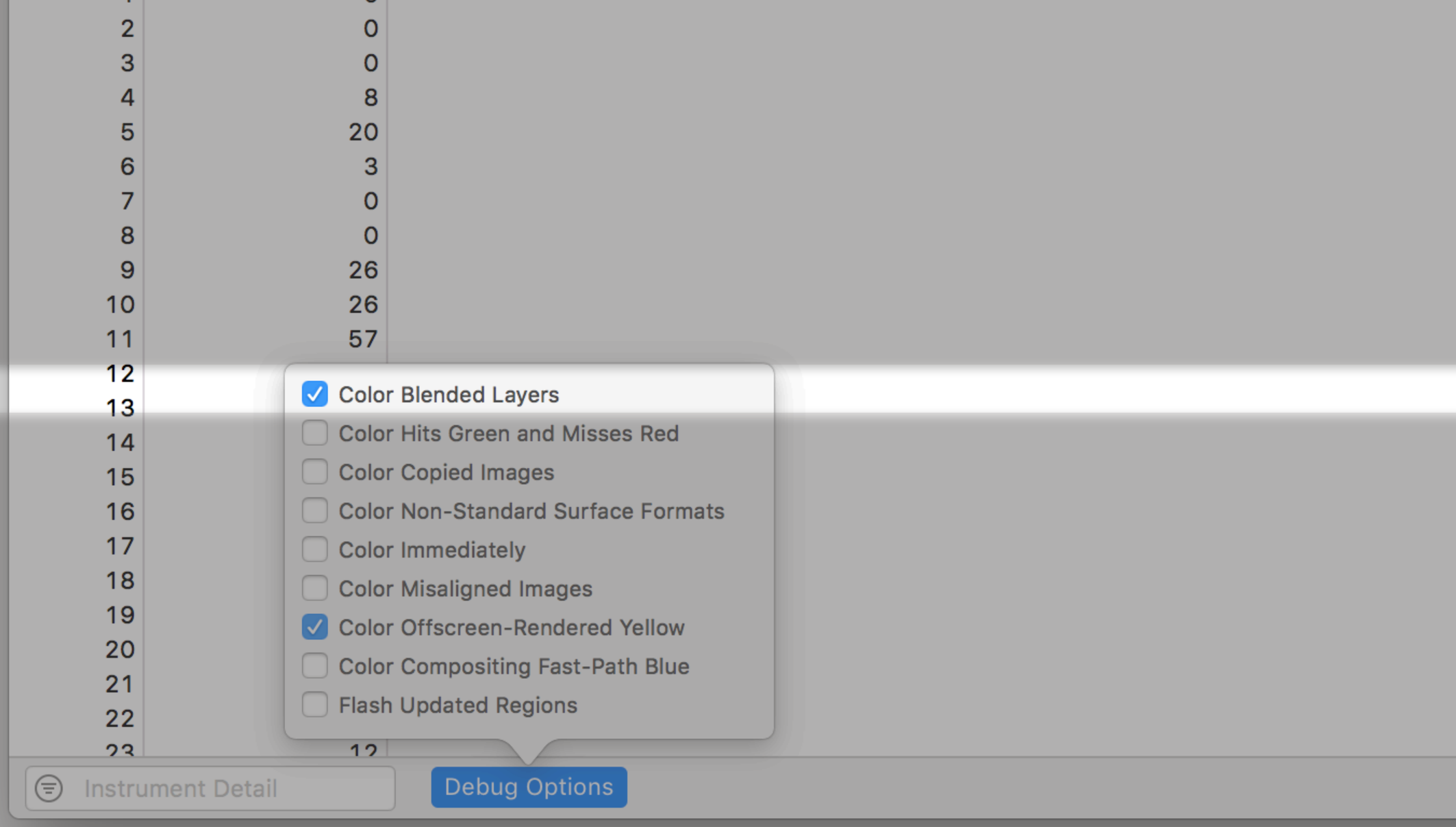
The Render Server



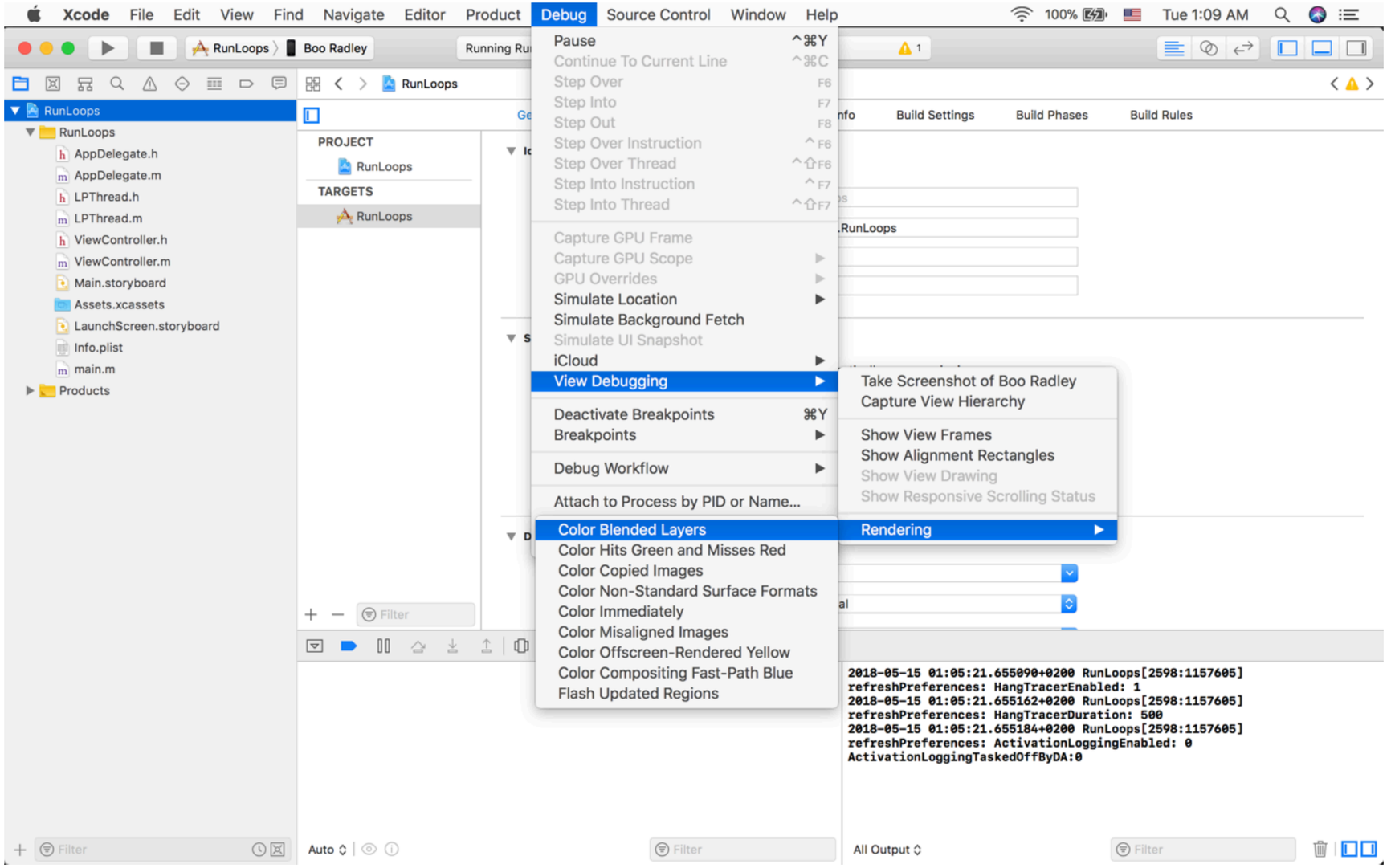
The Core Animation Instrument

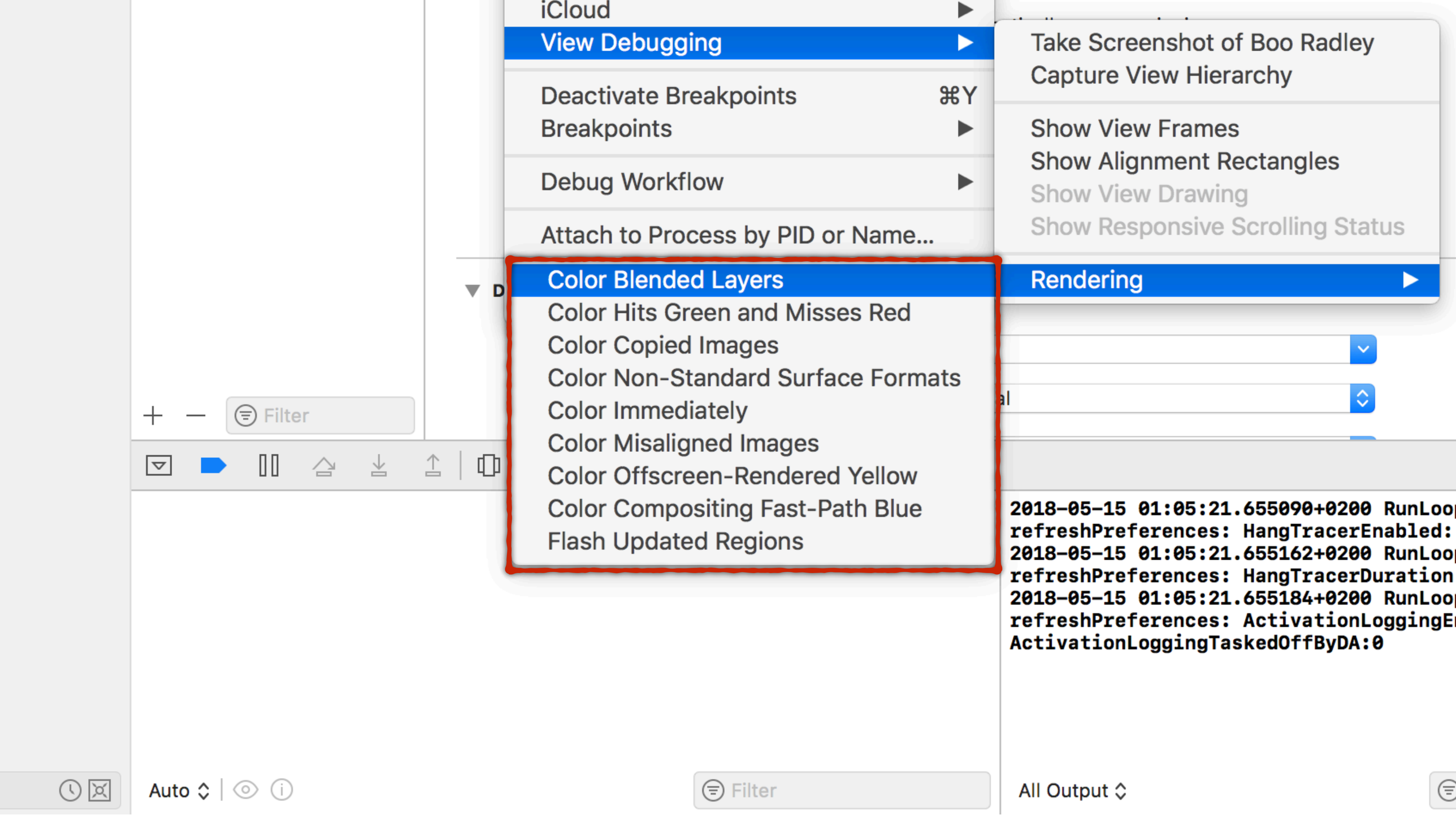



2 0
3 0
4 8
5 20
6 3
7 0
8 0
9 26
10 26
11 57
12
13
14
15
16
17
18
19
20
21
22
23 12
Color Blended Layers
Color Hits Green and Misses Red
Color Copied Images
Color Non-Standard Surface Formats
Color Immediately
Color Misaligned Images
Color Offscreen-Rendered Yellow
Color Compositing Fast-Path Blue
Flash Updated Regions
Instrument Detail
Debug Options

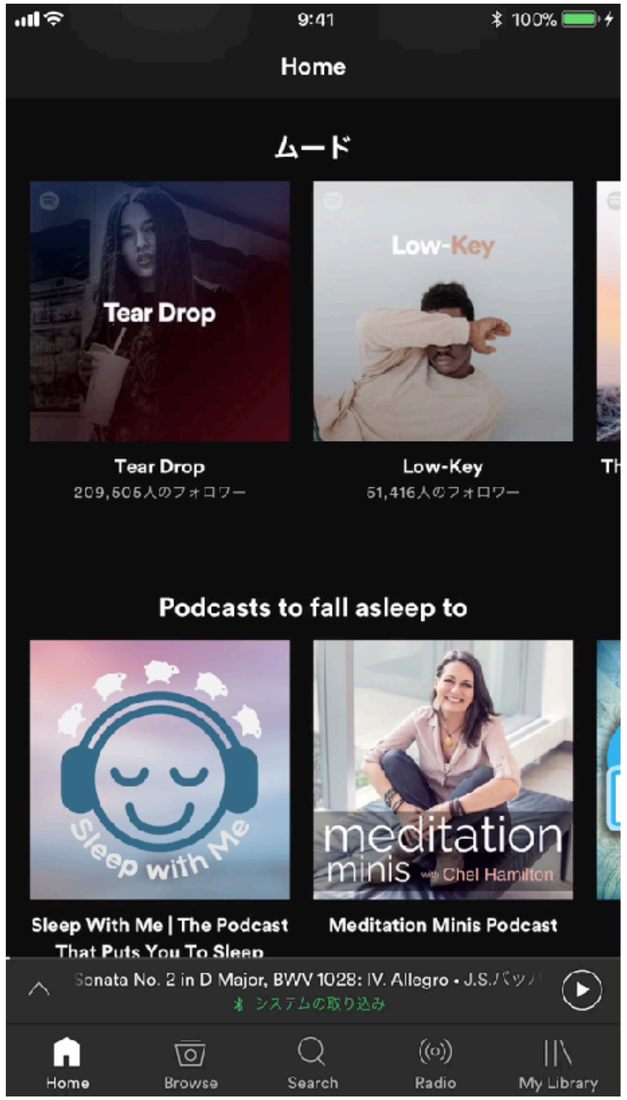

2 0
3 0
4 8
5 20
6 3
7 0
8 0
9 26
10 26
11 57
12
13
14
15
16
17
18
19
20
21
22
23 12
Color Blended Layers
Color Hits Green and Misses Red
Color Copied Images
Color Non-Standard Surface Formats
Color Immediately
Color Misaligned Images
Color Offscreen-Rendered Yellow
Color Compositing Fast-Path Blue
Flash Updated Regions
Instrument Detail
Debug Options

Debug Options in Xcode



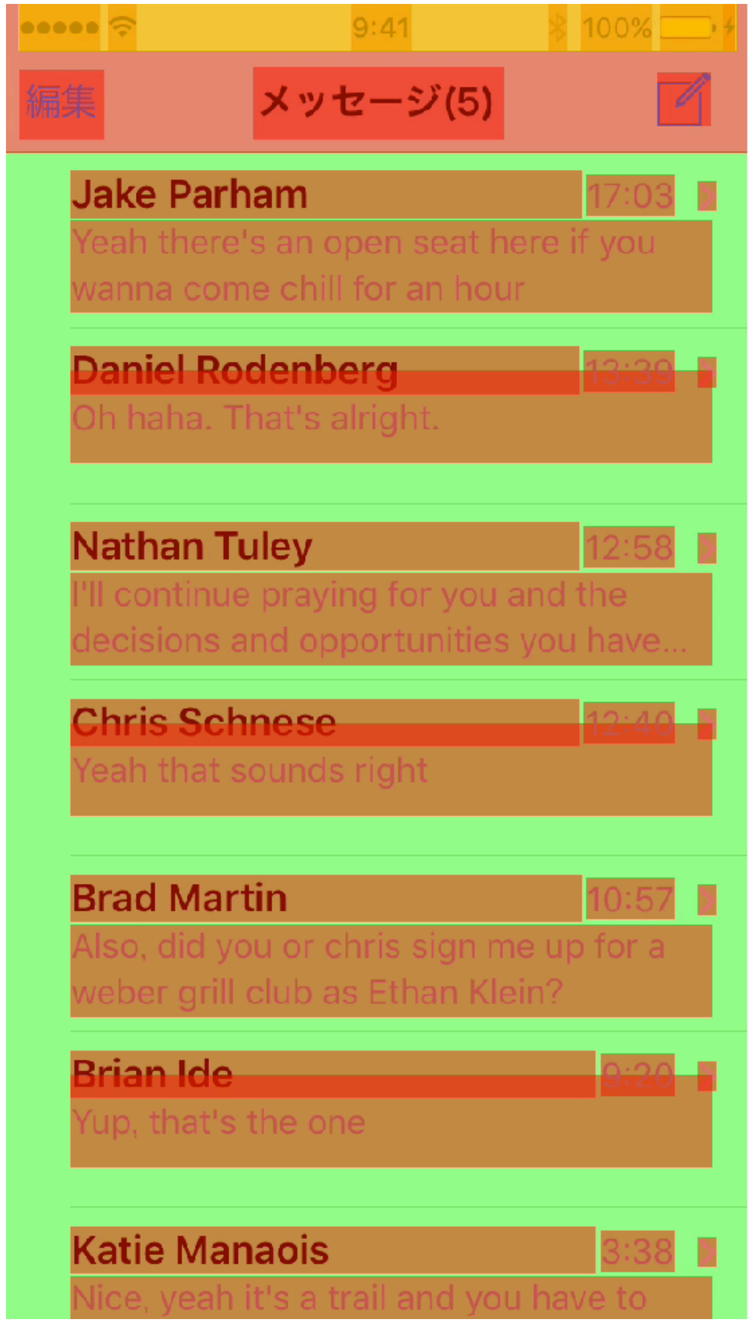

iCloud
View Debugging
Deactivate Breakpoints ⌘Y
Breakpoints
Debug Workflow
Attach to Process by PID or Name...
Take Screenshot of Boo Radley
Capture View Hierarchy
Show View Frames
Show Alignment Rectangles
Show View Drawing
Show Responsive Scrolling Status
Rendering
Color Blended Layers
Color Hits Green and Misses Red
Color Copied Images
Color Non-Standard Surface Formats
Color Immediately
Color Misaligned Images
Color Offscreen-Rendered Yellow
Color Compositing Fast-Path Blue
Flash Updated Regions
Filter
2018-05-15 01:05:21.655090+0200 RunLoo
refreshPreferences: HangTracerEnabled:
2018-05-15 01:05:21.655162+0200 RunLoo
refreshPreferences: HangTracerDuration
2018-05-15 01:05:21.655184+0200 RunLoo
refreshPreferences: ActivationLoggingE
ActivationLoggingTaskedOffByDA:0
Auto
Filter
All Output

Debug Options in Xcode



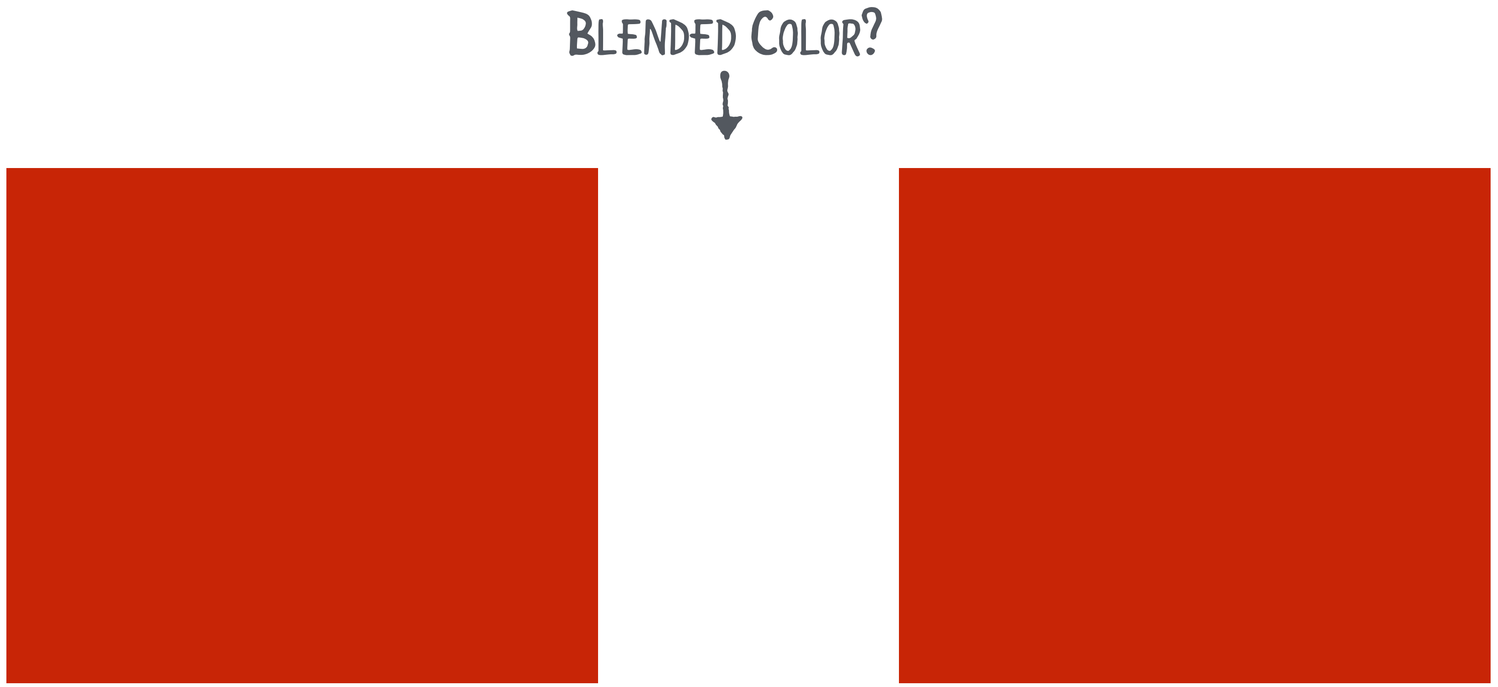
Performance Police

Perf Police


9:41
100%
編集
メッセージ(5)
Jake Parham
17:03
Yeah there's an open seat here if you wanna come chill for an hour
Daniel Rodenberg
13:39
Oh haha. That's alright.
Nathan Tuley
12:58
I'll continue praying for you and the decisions and opportunities you have...
Chris Schnese
12:40
Yeah that sounds right
Brad Martin
10:57
Also, did you or chris sign me up for a weber grill club as Ethan Klein?
Brian Ide
9:20
Yup, that's the one
Katie Manaois
3:38
Nice, yeah it's a trail and you have to

Alpha Blending

Alpha Blending



Rule #3:

Avoid unnecessary alpha blending

Alpha Blending

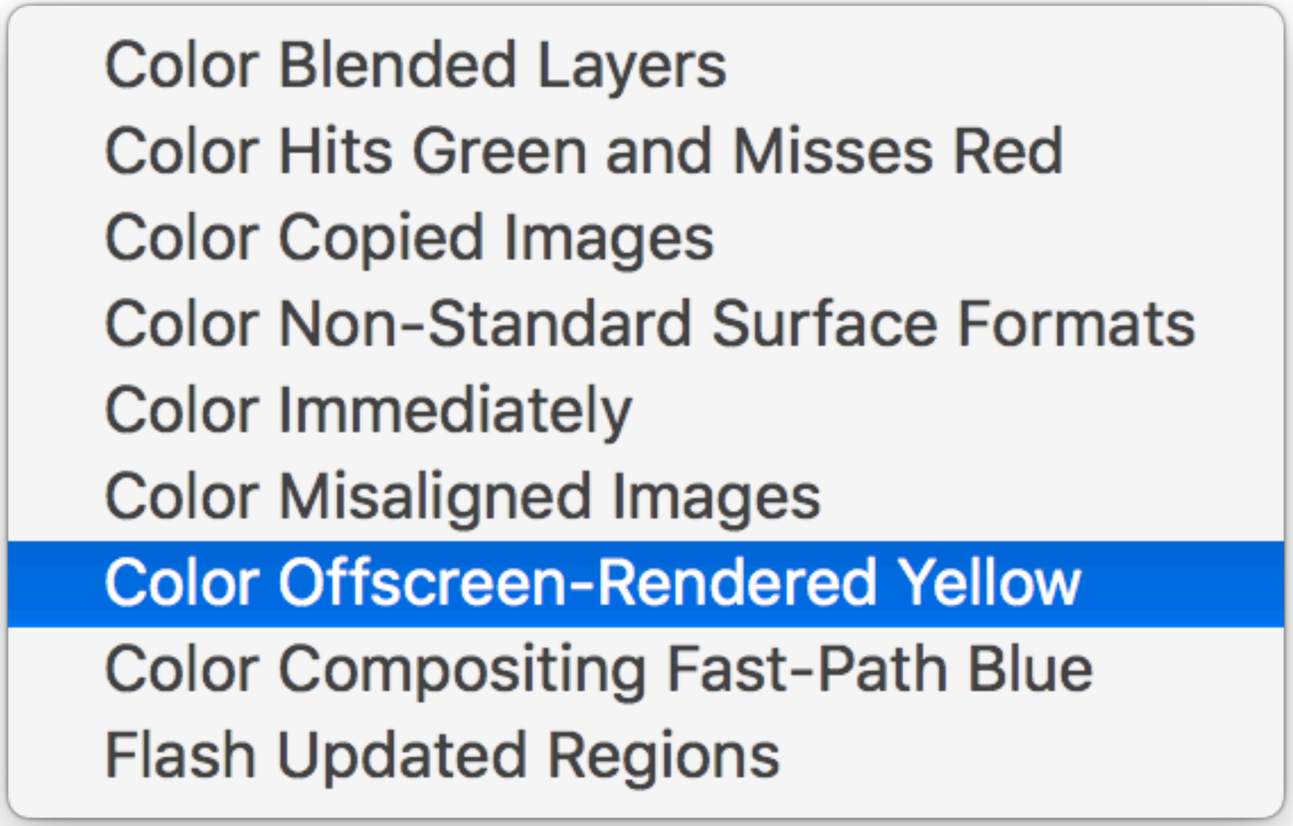
Don’t, unless you really wanna

If the subview has the same background color then blending is unnecessary

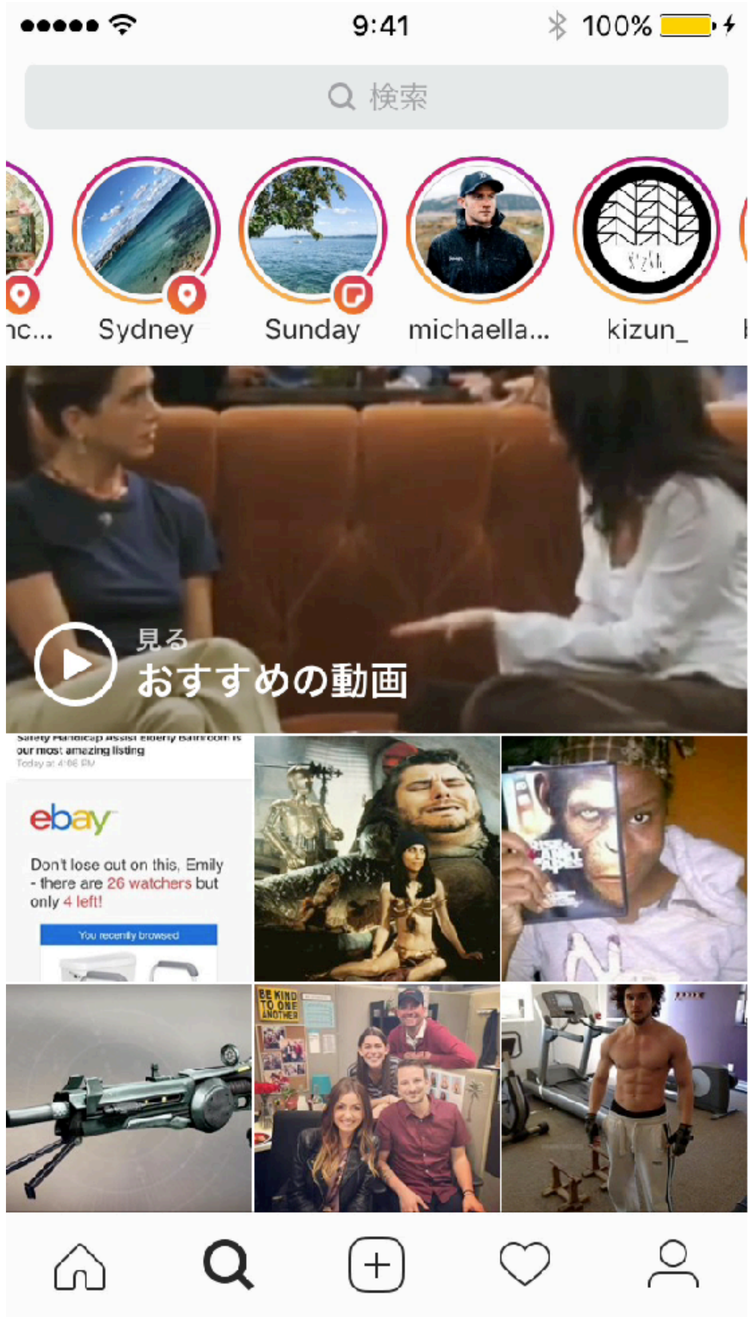
- Use Opacity and Background color to avoid it

```
userNameLabel.layer.opacity = 1.0
userNameLabel.backgroundColor = .white
```

Offscreen Rendering



Perf Police


9:41
100%
検索
Sydney
Sunday
michaella...
kizun_
見る
おすすめの動画
ebay
Don't lose out on this, Emily - there are 26 watchers but only 4 left!
You recently browsed

Rule #4:

Don’t use the cornerRadius property

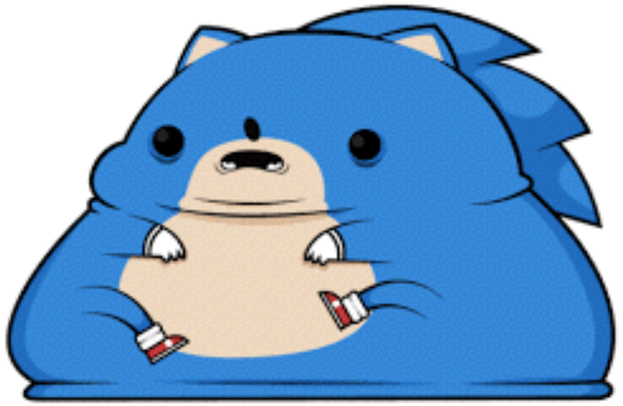
Offscreen Rendering

Round Corners

Instead, use a bezier path

```
func circularImage(withSize size: CGSize) -> UIImage {
  let scale = UIScreen.main.scale
  let circleRect = CGRect(x: 0, y: 0,
                          width: size.width * scale,
                          height: size.height * scale)

  UIGraphicsBeginImageContextWithOptions(circleRect.size, true, scale)

  let circlePath = UIBezierPath(roundedRect: circleRect,
                                cornerRadius: circleRect.width/2.0)
  circlePath.addClip()
  draw(in: circleRect)

  let roundedImage = UIGraphicsGetImageFromCurrentImageContext()
  return roundedImage!
}
```

Perf Police




9:41
100%
木下ゆうか YUKA KINOSHITA
+ 2100000
FANS
ありがとう
THANK YOU
フォロー中
木下ゆうか Yuka Kinoshita
@mochiko0204
ツイート
ツイートと返信
メディア

Rule #5:

For the love of G-d, don’t use the shouldRasterize property

Off-Screen Rendering

This is not the way

👍

Summary

Avoid Blending and Offscreen Rendering

- **Blending**
  - Make views opaque when possible
- **Offscreen Rendering**
  - Don’t use the **cornerRadius** property
  - Don’t use **shouldRasterize** unless you really know it helps
  - **Shadows** can also cause offscreen rendering
  - **UIVisualEffectsView**s can be even more expensive

ACTIVITY TRACE

Suggestion #1

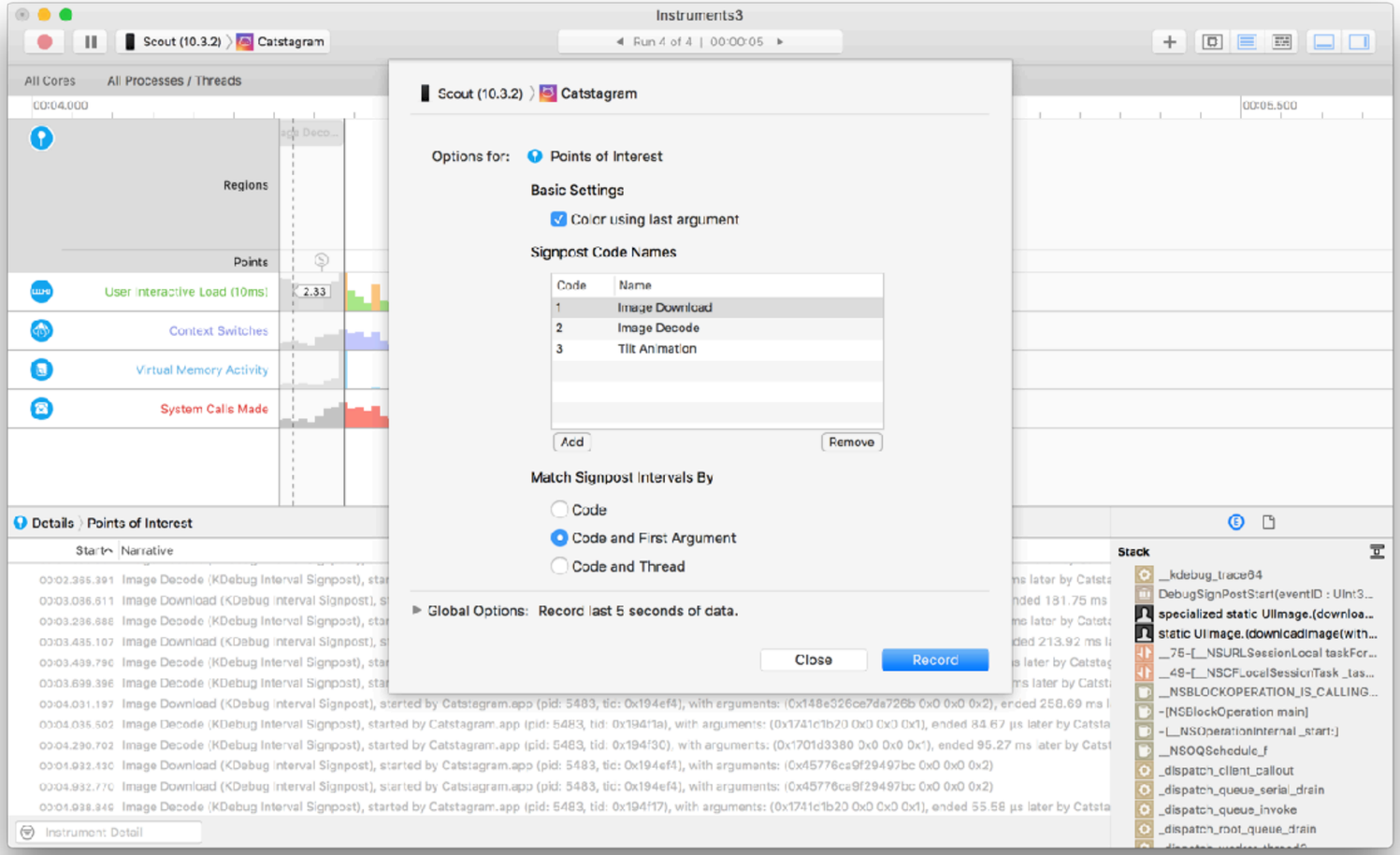
Use System Trace to track times for specific events

System Trace

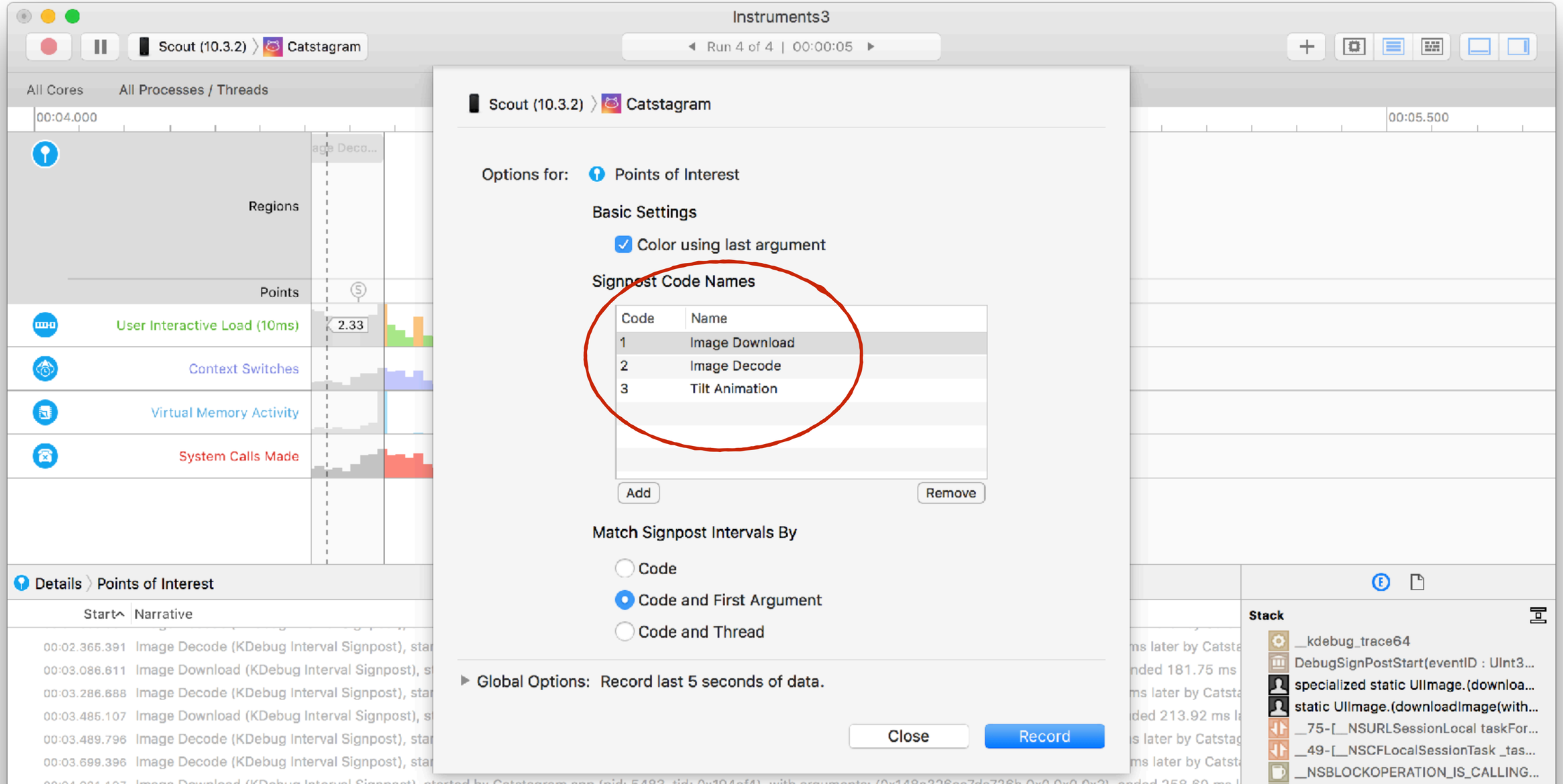
- Allows you to get fine grained information about what's happening in your system.
  - Use "Sign Posts" to signal when something important happens.
    - **Points** are single events
    - **Regions** have a beginning and ending and can track chunks of work
  - Shows how your code is interacting with the rest of the system

Getting Set Up

- By going to File > Recording Options… you can set up specific named events



Getting Set Up



Getting Set Up

- First, import the headers (objc only)

```
#import <sys/kdebug_signpost.h>
```

- You can set up points for significant events

```
kdebug_signpost(11, (uintptr_t)_fyuseModel, 0, 0, 2);
```

-

- And regions of interest (rendering a fyuse frame)

```
kdebug_signpost_start(11, (uintptr_t)currentFyuseModel, 0, 0, 1);
FYFrame *frame = [_fyuseModel frameForID:frameID];
kdebug_signpost_end(11, (uintptr_t)currentFyuseModel, 0, 0, 1);
```

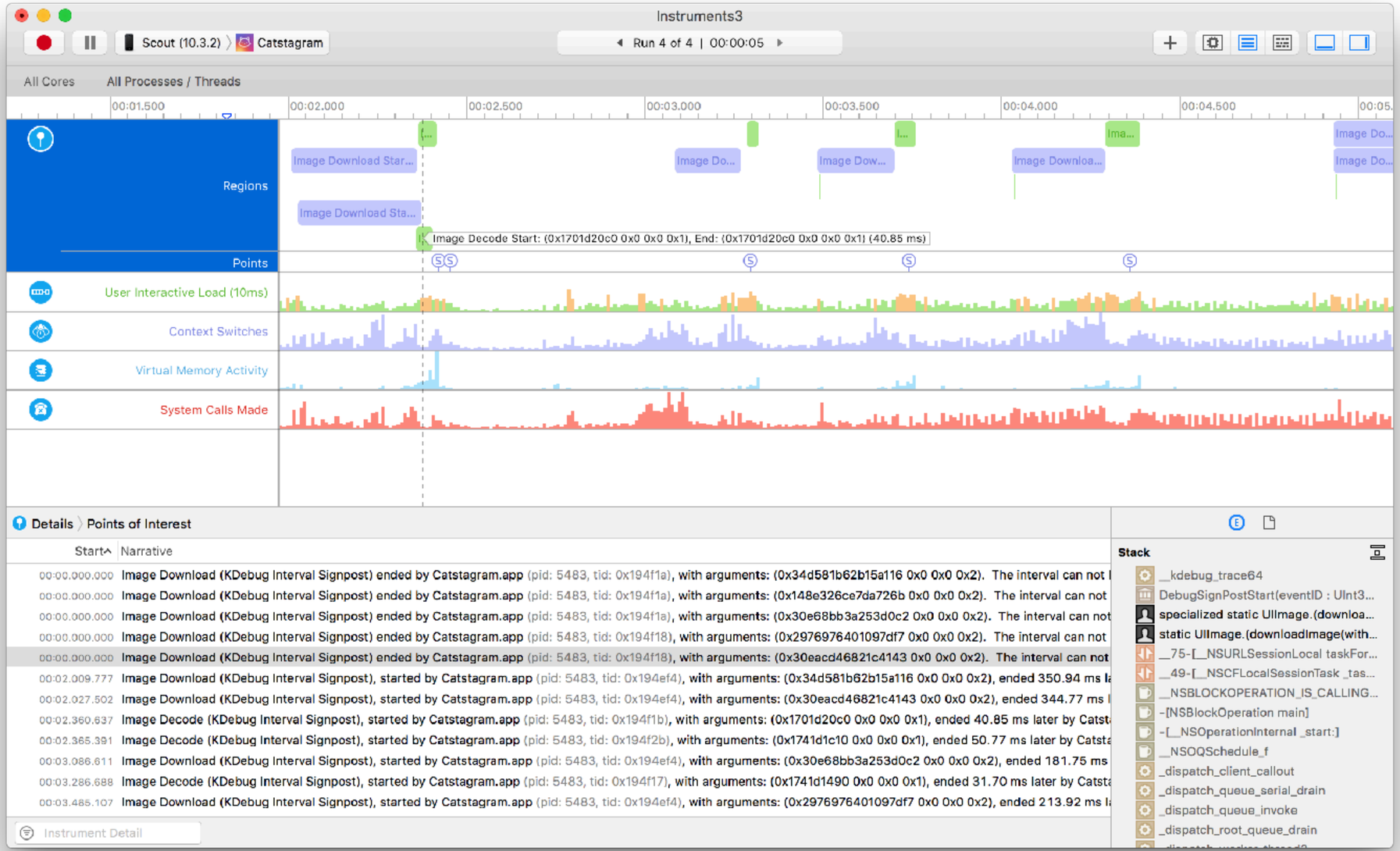
Sign Posts

For this sample app, I've written a wrapper class to make this stuff a little bit more readable. You can check it out on Github at:

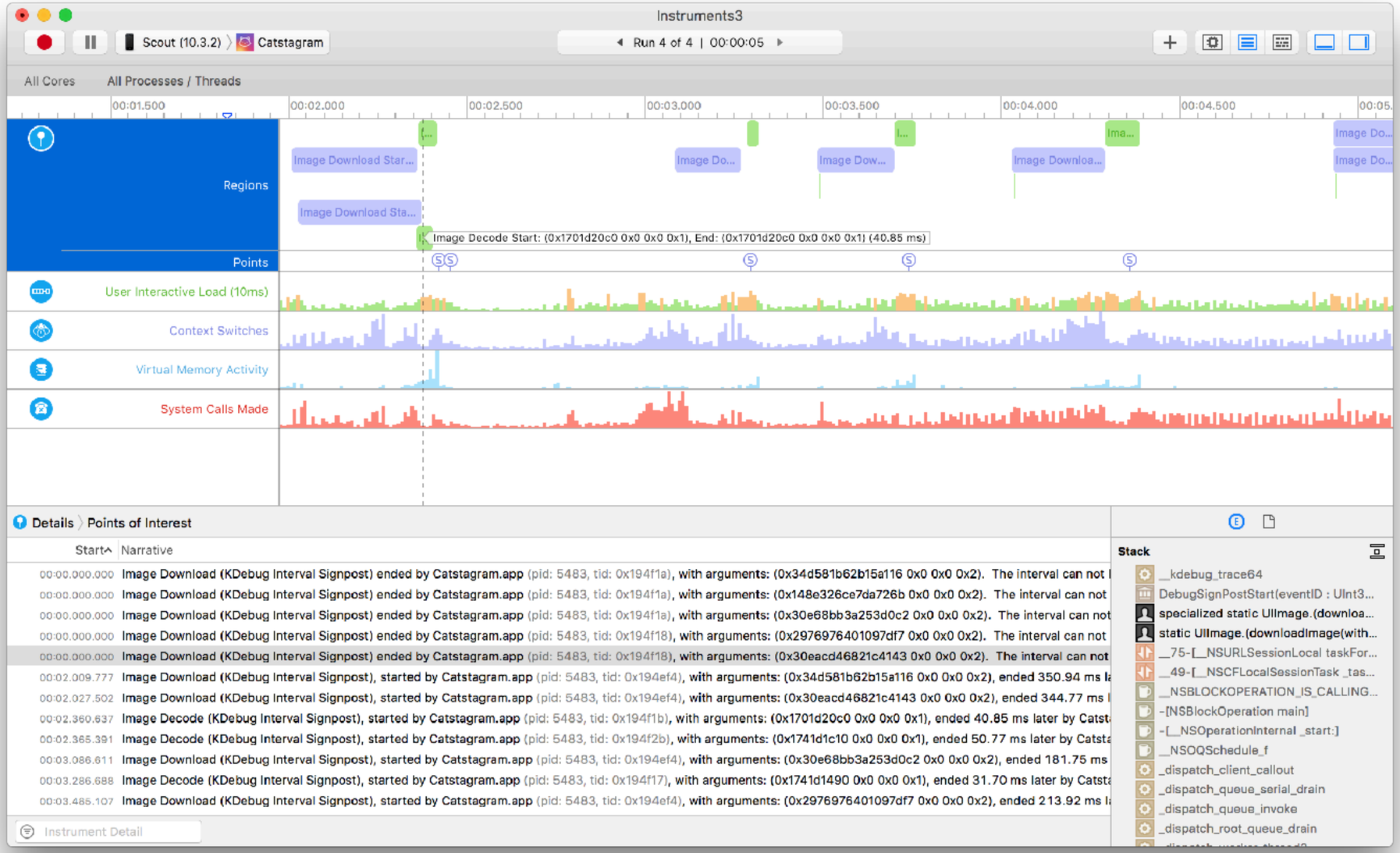
https://github.com/lappp9/GottaGoFastTalk

```
DebugSignPostStart(event: .imageDecode, eventObject: image, signPostColor: .green)
let decodedImage = self.decodedImage(image)
DebugSignPostEnd(event: .imageDecode, eventObject: image, signPostColor: .green)
```

System Trace

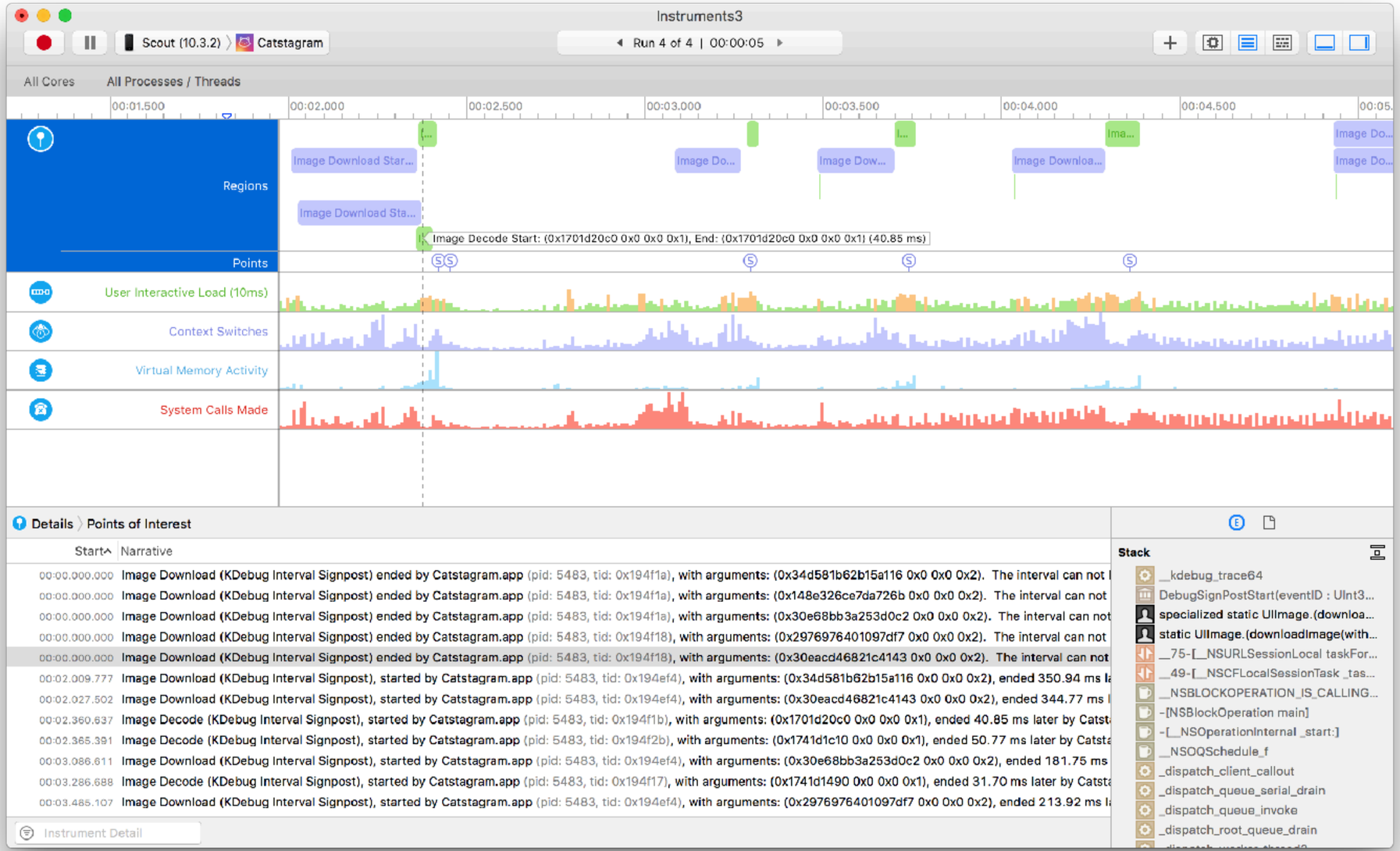


System Trace



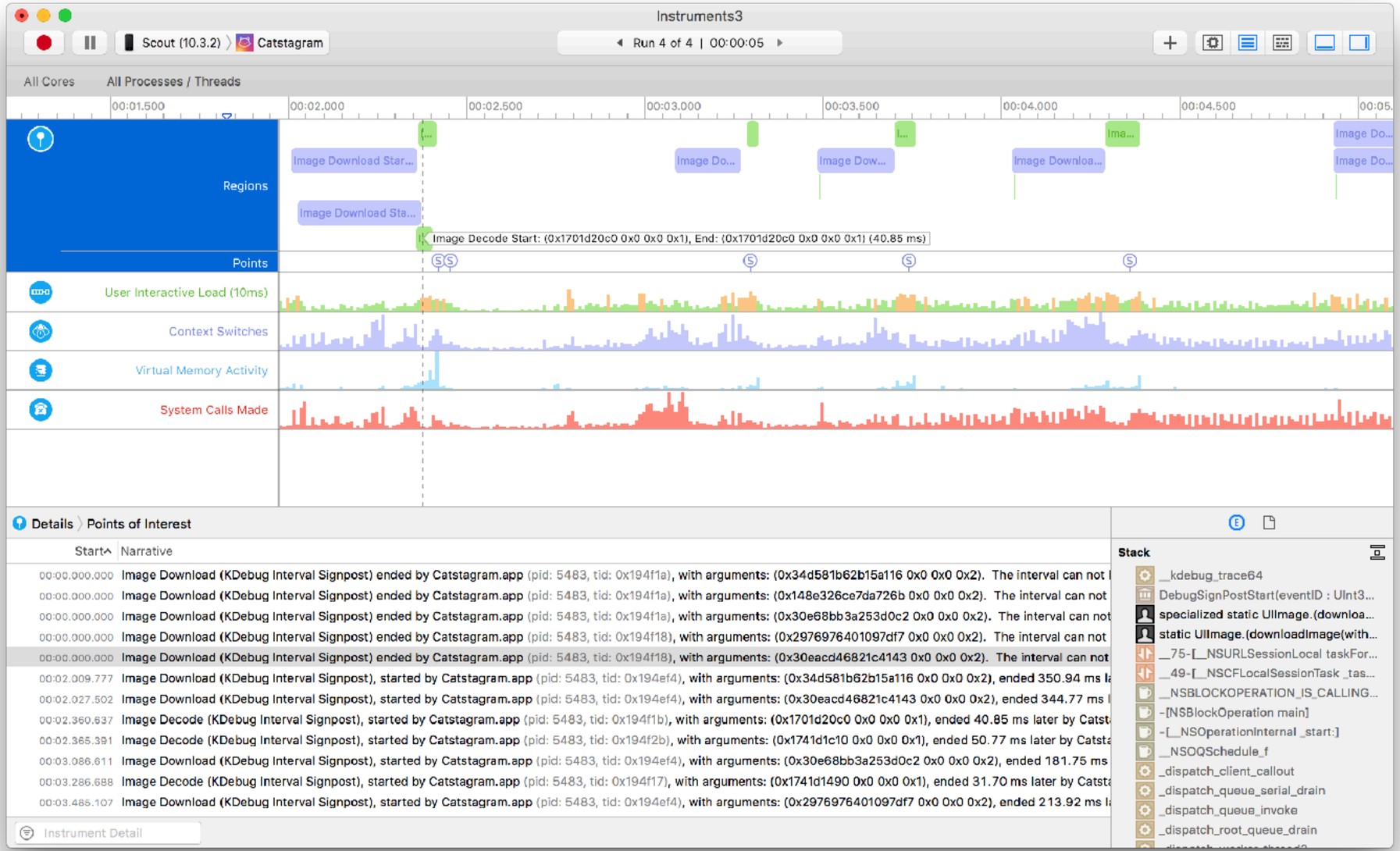
System Trace

User Interactive Load ↘

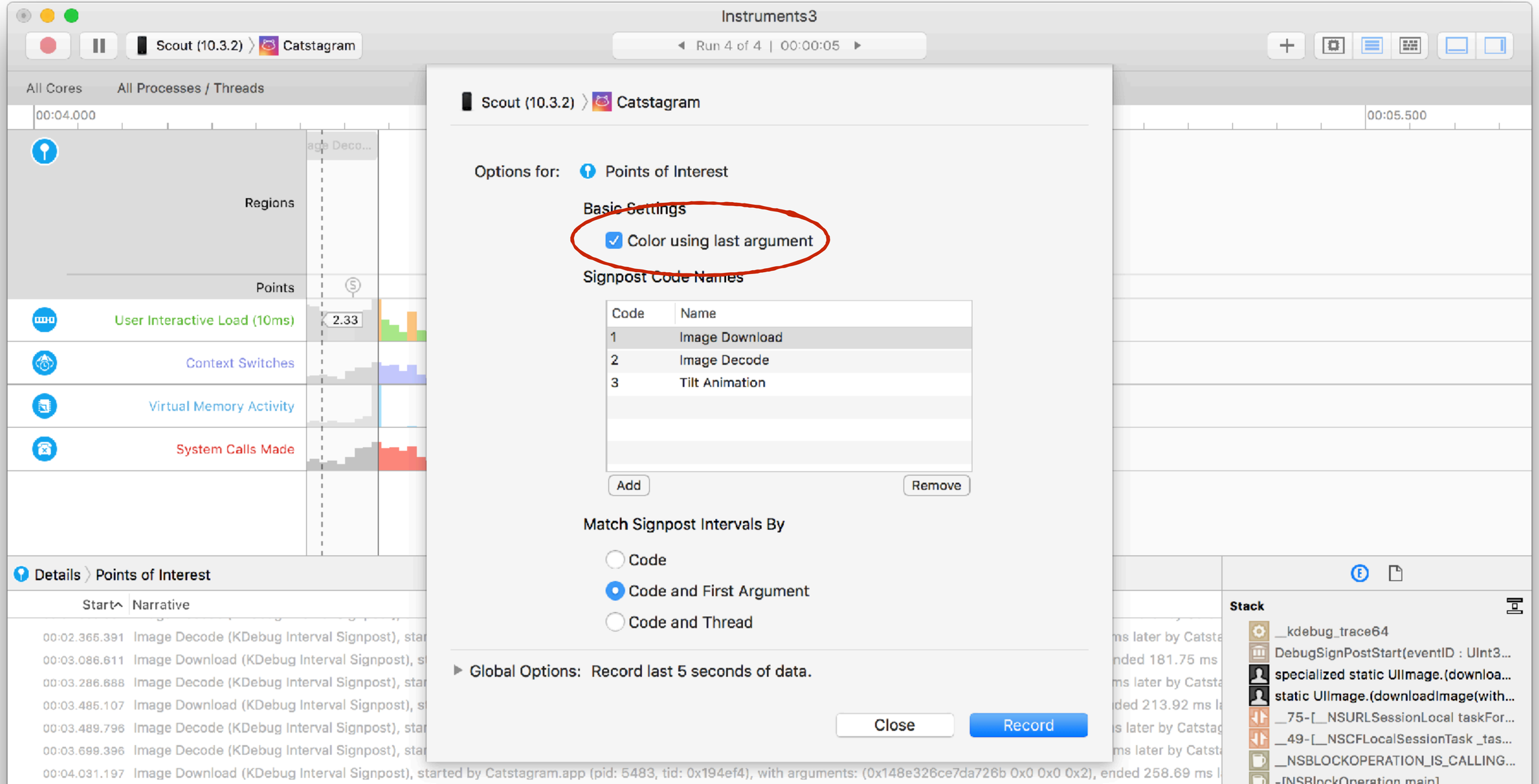


System Trace

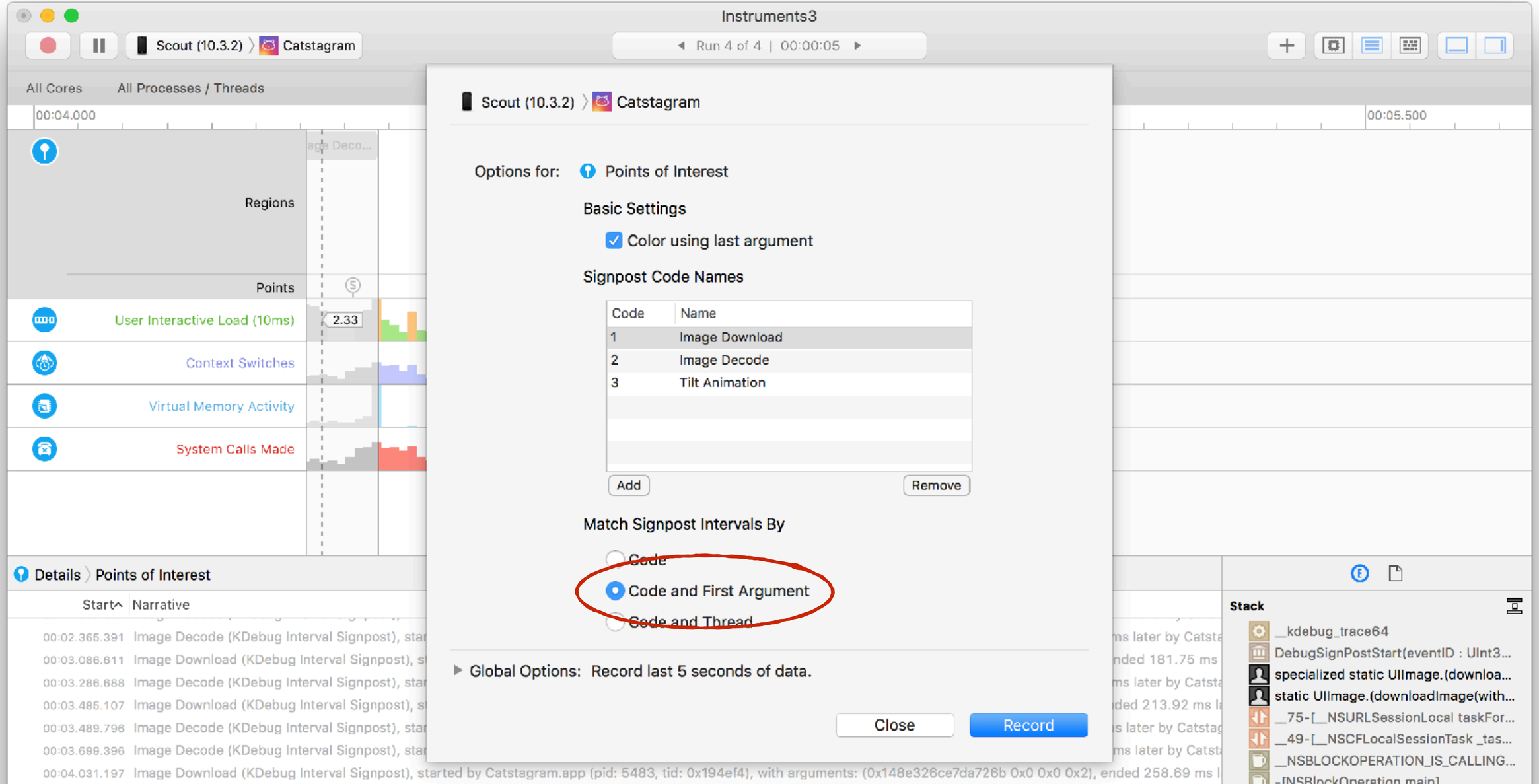
SYSTEM CALLS ↘



Getting Set Up

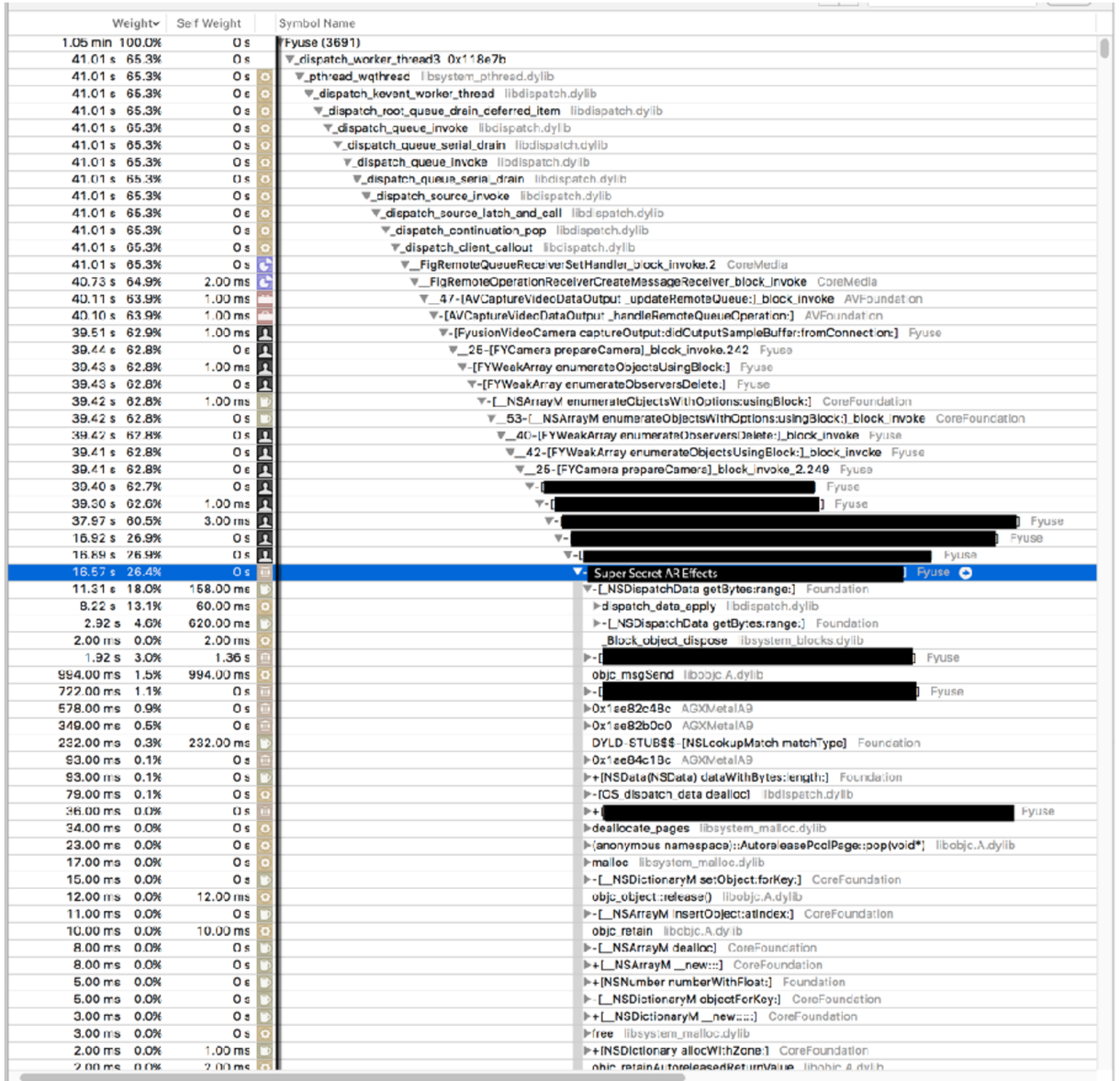


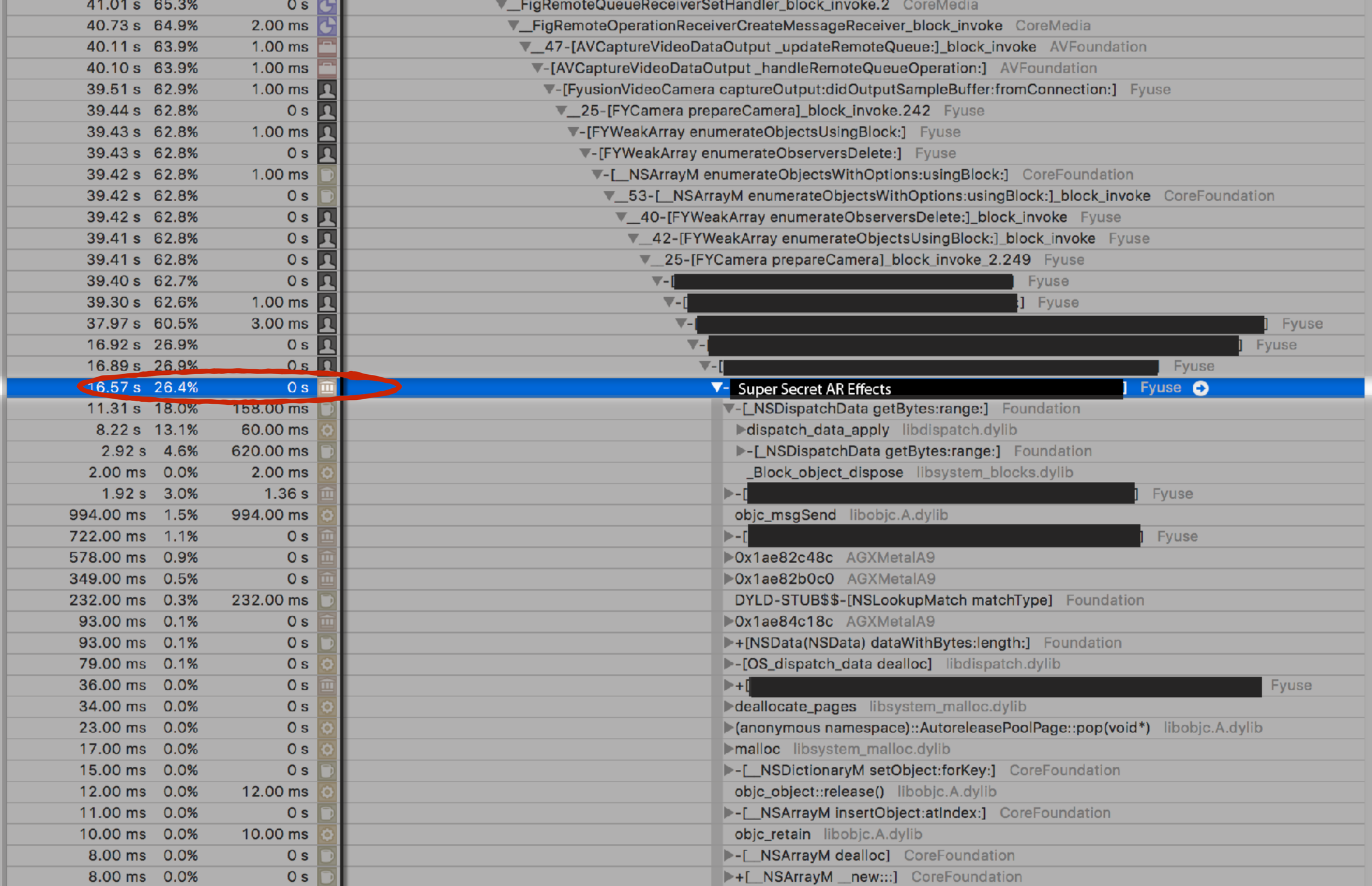
Getting Set Up

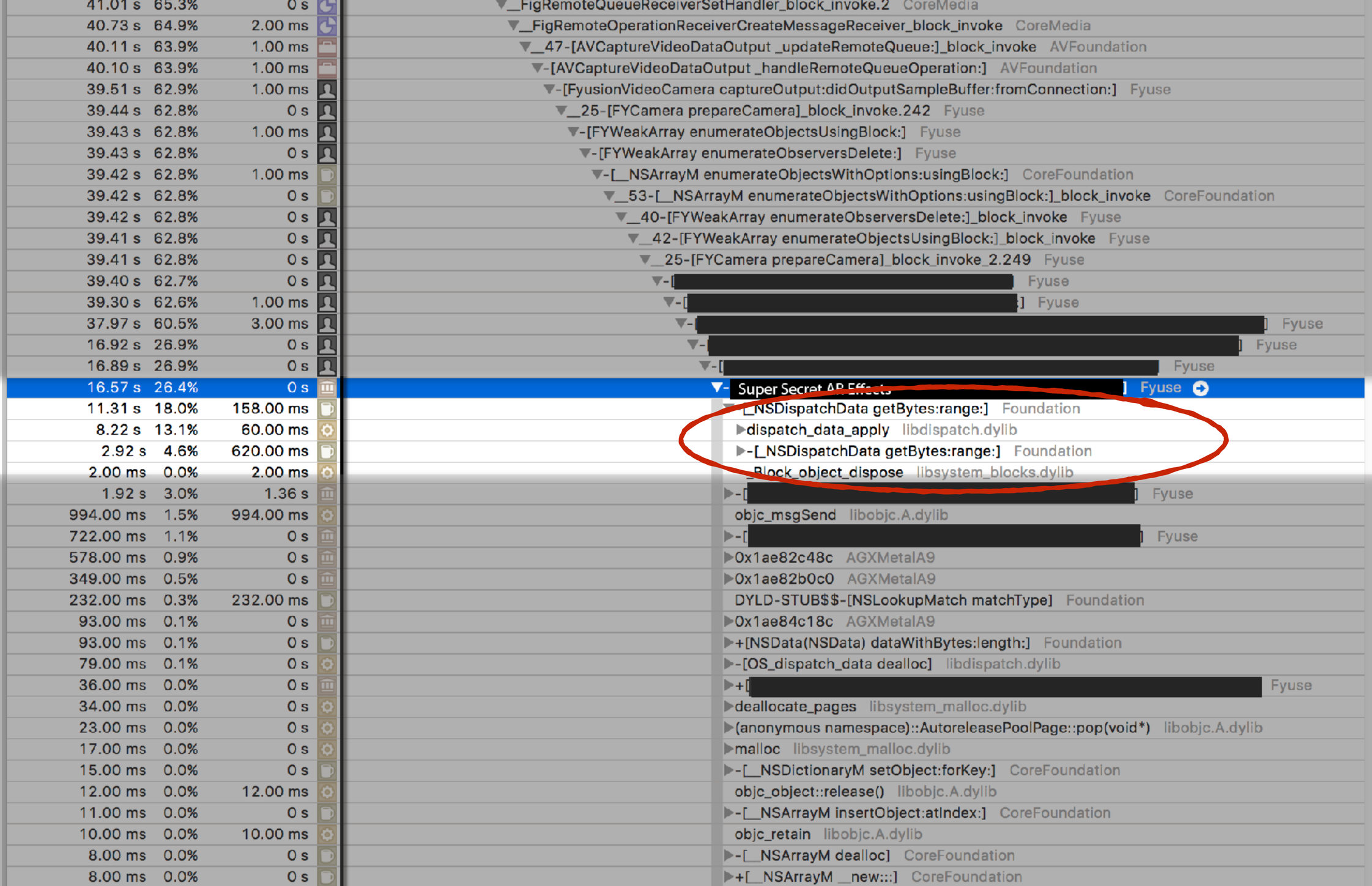



SONIC
RINGS 0
TOTAL
0
"TAILS"
RINGS 0
BONUS!

Camera Slowdown Example




41.01 s 65.3% 0 s
__FigRemoteQueueReceiverSetHandler_block_invoke.2 CoreMedia
40.73 s 64.9% 2.00 ms
__FigRemoteOperationReceiverCreateMessageReceiver_block_invoke CoreMedia
40.11 s 63.9% 1.00 ms
__47-[AVCaptureVideoDataOutput _updateRemoteQueue:]_block_invoke AVFoundation
40.10 s 63.9% 1.00 ms
-[AVCaptureVideoDataOutput _handleRemoteQueueOperation:] AVFoundation
39.51 s 62.9% 1.00 ms
-[FyusionVideoCamera captureOutput:didOutputSampleBuffer:fromConnection:] Fyuse
39.44 s 62.8% 0 s
__25-[FYCamera prepareCamera]_block_invoke.242 Fyuse
39.43 s 62.8% 1.00 ms
-[FYWeakArray enumerateObjectsUsingBlock:] Fyuse
39.43 s 62.8% 0 s
-[FYWeakArray enumerateObserversDelete:] Fyuse
39.42 s 62.8% 1.00 ms
-[__NSArrayM enumerateObjectsWithOptions:usingBlock:] CoreFoundation
39.42 s 62.8% 0 s
__53-[__NSArrayM enumerateObjectsWithOptions:usingBlock:]_block_invoke CoreFoundation
39.42 s 62.8% 0 s
__40-[FYWeakArray enumerateObserversDelete:]_block_invoke Fyuse
39.41 s 62.8% 0 s
__42-[FYWeakArray enumerateObjectsUsingBlock:]_block_invoke Fyuse
39.41 s 62.8% 0 s
__25-[FYCamera prepareCamera]_block_invoke_2.249 Fyuse
39.40 s 62.7% 0 s
Fyuse
39.30 s 62.6% 1.00 ms
Fyuse
37.97 s 60.5% 3.00 ms
Fyuse
16.92 s 26.9% 0 s
Fyuse
16.89 s 26.9% 0 s
Fyuse
16.57 s 26.4% 0 s
Super Secret AR Effects
Fyuse
11.31 s 18.0% 158.00 ms
-[_NSDispatchData getBytes:range:] Foundation
8.22 s 13.1% 60.00 ms
dispatch_data_apply libdispatch.dylib
2.92 s 4.6% 620.00 ms
-[_NSDispatchData getBytes:range:] Foundation
2.00 ms 0.0% 2.00 ms
_Block_object_dispose libsystem_blocks.dylib
1.92 s 3.0% 1.36 s
Fyuse
994.00 ms 1.5% 994.00 ms
objc_msgSend libobjc.A.dylib
722.00 ms 1.1% 0 s
Fyuse
578.00 ms 0.9% 0 s
0x1ae82c48c AGXMetalA9
349.00 ms 0.5% 0 s
0x1ae82b0c0 AGXMetalA9
232.00 ms 0.3% 232.00 ms
DYLD-STUB$$-[NSLookupMatch matchType] Foundation
93.00 ms 0.1% 0 s
0x1ae84c18c AGXMetalA9
93.00 ms 0.1% 0 s
+[NSData(NSData) dataWithBytes:length:] Foundation
79.00 ms 0.1% 0 s
-[OS_dispatch_data dealloc] libdispatch.dylib
36.00 ms 0.0% 0 s
Fyuse
34.00 ms 0.0% 0 s
deallocate_pages libsystem_malloc.dylib
23.00 ms 0.0% 0 s
(anonymous namespace)::AutoreleasePoolPage::pop(void*) libobjc.A.dylib
17.00 ms 0.0% 0 s
malloc libsystem_malloc.dylib
15.00 ms 0.0% 0 s
-[__NSDictionaryM setObject:forKey:] CoreFoundation
12.00 ms 0.0% 12.00 ms
objc_object::release() libobjc.A.dylib
11.00 ms 0.0% 0 s
-[__NSArrayM insertObject:atIndex:] CoreFoundation
10.00 ms 0.0% 10.00 ms
objc_retain libobjc.A.dylib
8.00 ms 0.0% 0 s
-[__NSArrayM dealloc] CoreFoundation
8.00 ms 0.0% 0 s
+[__NSArrayM __new:::] CoreFoundation


41.01 s 65.3% 0 s
__FigRemoteQueueReceiverSetHandler_block_invoke.2 CoreMedia
40.73 s 64.9% 2.00 ms
__FigRemoteOperationReceiverCreateMessageReceiver_block_invoke CoreMedia
40.11 s 63.9% 1.00 ms
__47-[AVCaptureVideoDataOutput _updateRemoteQueue:]_block_invoke AVFoundation
40.10 s 63.9% 1.00 ms
-[AVCaptureVideoDataOutput _handleRemoteQueueOperation:] AVFoundation
39.51 s 62.9% 1.00 ms
-[FyusionVideoCamera captureOutput:didOutputSampleBuffer:fromConnection:] Fyuse
39.44 s 62.8% 0 s
__25-[FYCamera prepareCamera]_block_invoke.242 Fyuse
39.43 s 62.8% 1.00 ms
-[FYWeakArray enumerateObjectsUsingBlock:] Fyuse
39.43 s 62.8% 0 s
-[FYWeakArray enumerateObserversDelete:] Fyuse
39.42 s 62.8% 1.00 ms
-[__NSArrayM enumerateObjectsWithOptions:usingBlock:] CoreFoundation
39.42 s 62.8% 0 s
__53-[__NSArrayM enumerateObjectsWithOptions:usingBlock:]_block_invoke CoreFoundation
39.42 s 62.8% 0 s
__40-[FYWeakArray enumerateObserversDelete:]_block_invoke Fyuse
39.41 s 62.8% 0 s
__42-[FYWeakArray enumerateObjectsUsingBlock:]_block_invoke Fyuse
39.41 s 62.8% 0 s
__25-[FYCamera prepareCamera]_block_invoke_2.249 Fyuse
39.40 s 62.7% 0 s
Fyuse
39.30 s 62.6% 1.00 ms
Fyuse
37.97 s 60.5% 3.00 ms
Fyuse
16.92 s 26.9% 0 s
Fyuse
16.89 s 26.9% 0 s
Fyuse
16.57 s 26.4% 0 s
Fyuse
11.31 s 18.0% 158.00 ms
-[_NSDispatchData getBytes:range:] Foundation
8.22 s 13.1% 60.00 ms
dispatch_data_apply libdispatch.dylib
2.92 s 4.6% 620.00 ms
-[_NSDispatchData getBytes:range:] Foundation
2.00 ms 0.0% 2.00 ms
_Block_object_dispose libsystem_blocks.dylib
1.92 s 3.0% 1.36 s
Fyuse
994.00 ms 1.5% 994.00 ms
objc_msgSend libobjc.A.dylib
722.00 ms 1.1% 0 s
Fyuse
578.00 ms 0.9% 0 s
0x1ae82c48c AGXMetalA9
349.00 ms 0.5% 0 s
0x1ae82b0c0 AGXMetalA9
232.00 ms 0.3% 232.00 ms
DYLD-STUB$$-[NSLookupMatch matchType] Foundation
93.00 ms 0.1% 0 s
0x1ae84c18c AGXMetalA9
93.00 ms 0.1% 0 s
+[NSData(NSData) dataWithBytes:length:] Foundation
79.00 ms 0.1% 0 s
-[OS_dispatch_data dealloc] libdispatch.dylib
36.00 ms 0.0% 0 s
Fyuse
34.00 ms 0.0% 0 s
deallocate_pages libsystem_malloc.dylib
23.00 ms 0.0% 0 s
(anonymous namespace)::AutoreleasePoolPage::pop(void*) libobjc.A.dylib
17.00 ms 0.0% 0 s
malloc libsystem_malloc.dylib
15.00 ms 0.0% 0 s
-[__NSDictionaryM setObject:forKey:] CoreFoundation
12.00 ms 0.0% 12.00 ms
objc_object::release() libobjc.A.dylib
11.00 ms 0.0% 0 s
-[__NSArrayM insertObject:atIndex:] CoreFoundation
10.00 ms 0.0% 10.00 ms
objc_retain libobjc.A.dylib
8.00 ms 0.0% 0 s
-[__NSArrayM dealloc] CoreFoundation
8.00 ms 0.0% 0 s
+[__NSArrayM __new:::] CoreFoundation

Bonus Rule #1:

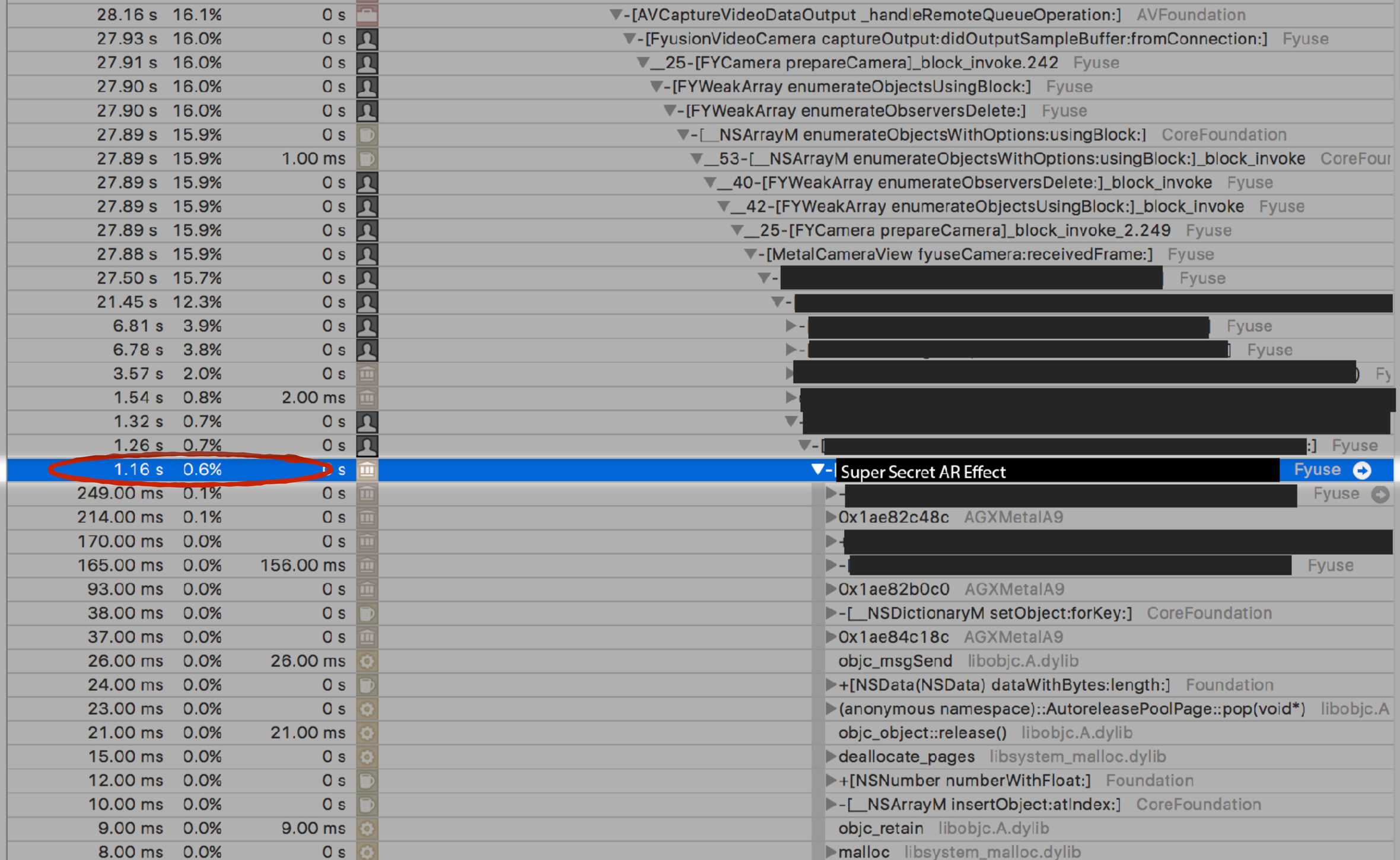
Use C/C++

Use C

No seriously

- Since the bottleneck was around one call to grab a float value from a chunk of memory in an NSData, you can just use **memcpy** instead

```
for (int i = 0; i < imageByteCount * numSlices / sizeof(float); i++) {
    float myFloat;
    [data getBytes:&myFloat range:NSMakeRange(i * sizeof(float), sizeof(float))];
    memcpy(&myFloat, imageBytes + (i * sizeof(float)), sizeof(float));
```


28.16 s 16.1% 0 s -[AVCaptureVideoDataOutput _handleRemoteQueueOperation:] AVFoundation
27.93 s 16.0% 0 s -[FyusionVideoCamera captureOutput:didOutputSampleBuffer:fromConnection:] Fyuse
27.91 s 16.0% 0 s __25-[FYCamera prepareCamera]_block_invoke.242 Fyuse
27.90 s 16.0% 0 s -[FYWeakArray enumerateObjectsUsingBlock:] Fyuse
27.90 s 16.0% 0 s -[FYWeakArray enumerateObserversDelete:] Fyuse
27.89 s 15.9% 0 s -[__NSArrayM enumerateObjectsWithOptions:usingBlock:] CoreFoundation
27.89 s 15.9% 1.00 ms __53-[__NSArrayM enumerateObjectsWithOptions:usingBlock:]_block_invoke CoreFour
27.89 s 15.9% 0 s __40-[FYWeakArray enumerateObserversDelete:]_block_invoke Fyuse
27.89 s 15.9% 0 s __42-[FYWeakArray enumerateObjectsUsingBlock:]_block_invoke Fyuse
27.89 s 15.9% 0 s __25-[FYCamera prepareCamera]_block_invoke_2.249 Fyuse
27.88 s 15.9% 0 s -[MetalCameraView fyuseCamera:receivedFrame:] Fyuse
27.50 s 15.7% 0 s Fyuse
21.45 s 12.3% 0 s
6.81 s 3.9% 0 s Fyuse
6.78 s 3.8% 0 s Fyuse
3.57 s 2.0% 0 s
1.54 s 0.8% 2.00 ms
1.32 s 0.7% 0 s
1.26 s 0.7% 0 s Fyuse
1.16 s 0.6% s Super Secret AR Effect Fyuse
249.00 ms 0.1% 0 s Fyuse
214.00 ms 0.1% 0 s 0x1ae82c48c AGXMetalA9
170.00 ms 0.0% 0 s
165.00 ms 0.0% 156.00 ms Fyuse
93.00 ms 0.0% 0 s 0x1ae82b0c0 AGXMetalA9
38.00 ms 0.0% 0 s -[__NSDictionaryM setObject:forKey:] CoreFoundation
37.00 ms 0.0% 0 s 0x1ae84c18c AGXMetalA9
26.00 ms 0.0% 26.00 ms objc_msgSend libobjc.A.dylib
24.00 ms 0.0% 0 s +[NSData(NSData) dataWithBytes:length:] Foundation
23.00 ms 0.0% 0 s (anonymous namespace)::AutoreleasePoolPage::pop(void*) libobjc.A
21.00 ms 0.0% 21.00 ms objc_object::release() libobjc.A.dylib
15.00 ms 0.0% 0 s deallocate_pages libsystem_malloc.dylib
12.00 ms 0.0% 0 s +[NSNumber numberWithFloat:] Foundation
10.00 ms 0.0% 0 s -[__NSArrayM insertObject:atIndex:] CoreFoundation
9.00 ms 0.0% 9.00 ms objc_retain libobjc.A.dylib
8.00 ms 0.0% 0 s malloc libsystem_malloc.dylib

Bonus Rule #2:

Avoid overdraw

Use a CADisplayLink Again

Seriously this time

Use the preferredFramesPerSecond property

- Available on iOS 10 or higher
- Just do it manually for older OSes

```
_link = [CADisplayLink displayLinkWithTarget:self
                                    selector:@selector(renderIfNecessary:)];
if (FY_AT_LEAST_IOS10) {
  _link.preferredFramesPerSecond = [[UIDevice currentDevice] desiredFrameRate];
} else {
  _needsManualThrottling = YES;
}
[_link addToRunLoop:[NSRunLoop mainRunLoop] forMode:NSRunLoopCommonModes];
```

Bonus Rule #2:

Use IMP Caching

IMP Caching

Sometimes you'll see a significant number of **objc_msgSend**() calls in traces

```
if (!_frameImp) {
  _frameImp = (FYFrame *(*)(id, SEL, CGFloat))[_fyuseModel methodForSelector:@selector(frameForID:)];
}
_frameImp(_fyuseModel, @selector(frameForID:), _curFrameId);
```

Bonus Rule #3:

Don’t use ARC

or Swift…

WHERE TO GO FROM HERE?

Books

iOS and macOS Performance Tuning

Videos

Practical Instruments Course

Years worth of WWDC Videos

QUESTIONS?

Blog: lukeparham.com

Twitter: @lukeparham

UIKonf

@lukeparham