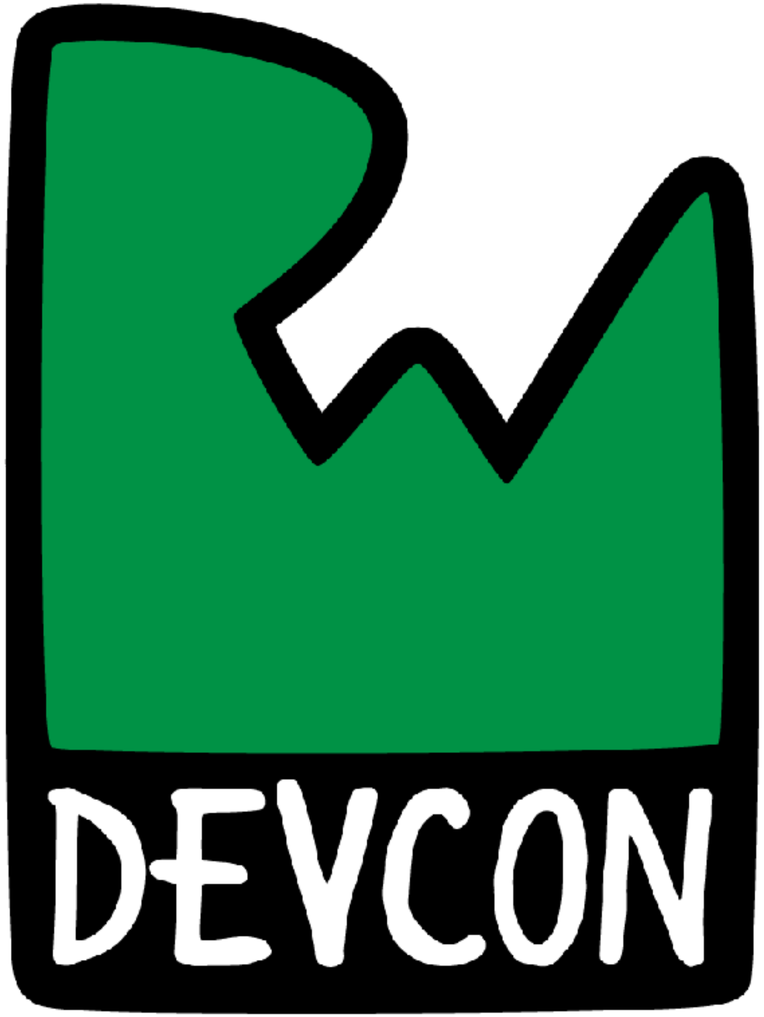
Practical Instruments

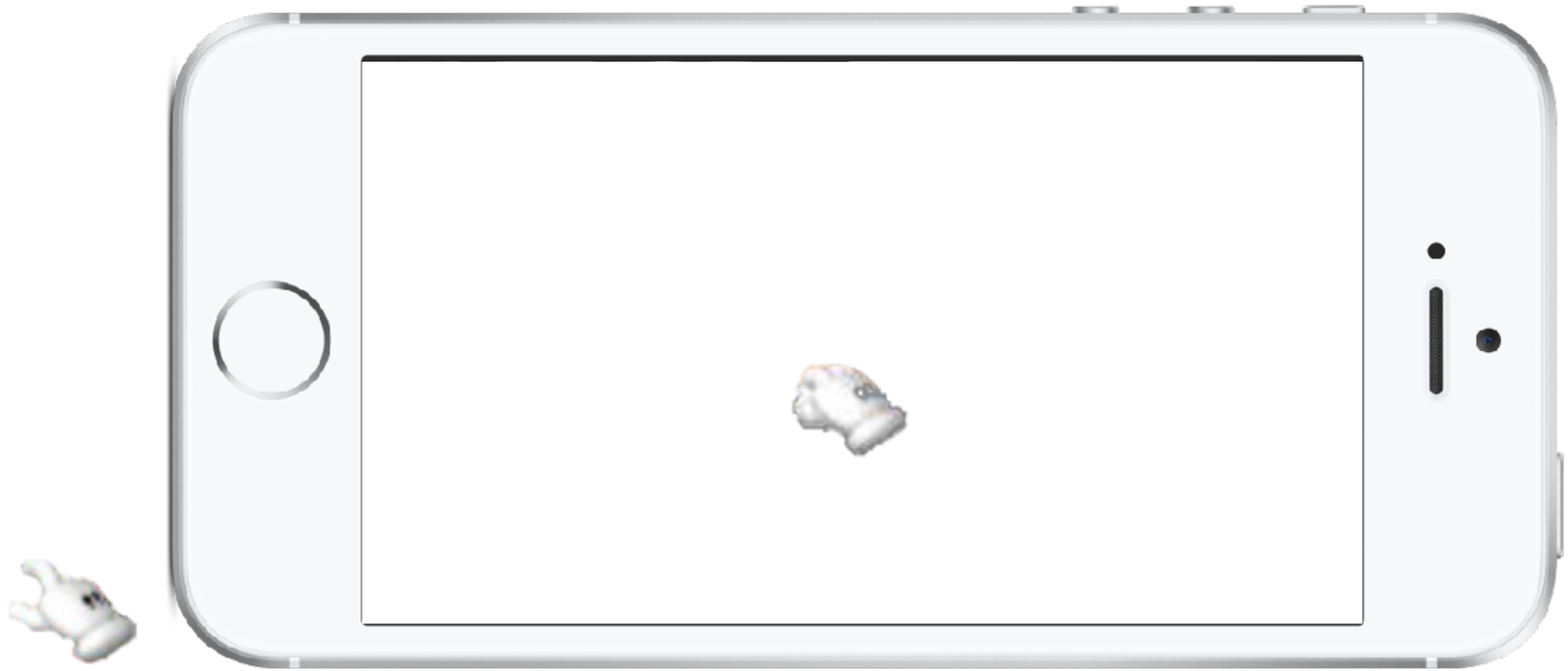


Intro

Luke Parham

iOS Engineer at Fyusion

Semi-Pro Rocket League Player 🚀🏎



Why Care?


Archivio Composizione Esposizione Strumenti
The Macintosh™ Finder
Bruce Horn and Steve Capps
512K
Version 4.1 ©1985 Apple Computer

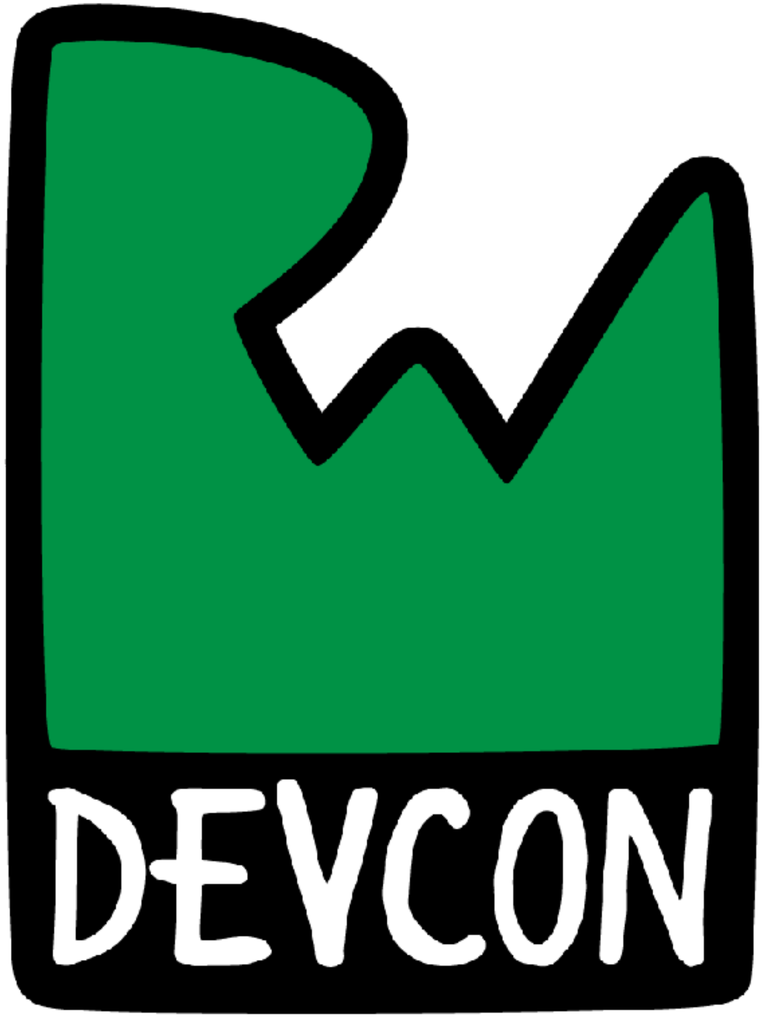
"Let's say you can shave 10 seconds off of the boot time. Multiply that by five million users and thats 50 million seconds, every single day.

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

- **Performance and Rendering in iOS**
- **Time Profiler**
- **The Core Animation Instrument**
- **System Trace**
- **Writing Fast Code**

Practical Instruments



Performance and Rendering in iOS

Performance on iOS

- **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)**
  - **Responsiveness is often more important**
  - **Dropped frames and blocking interactions create a perceived slowness**

Rule #1

Use a CADisplayLink to track dropped frames

Demo #1!

DETECTING DROPPED FRAMES

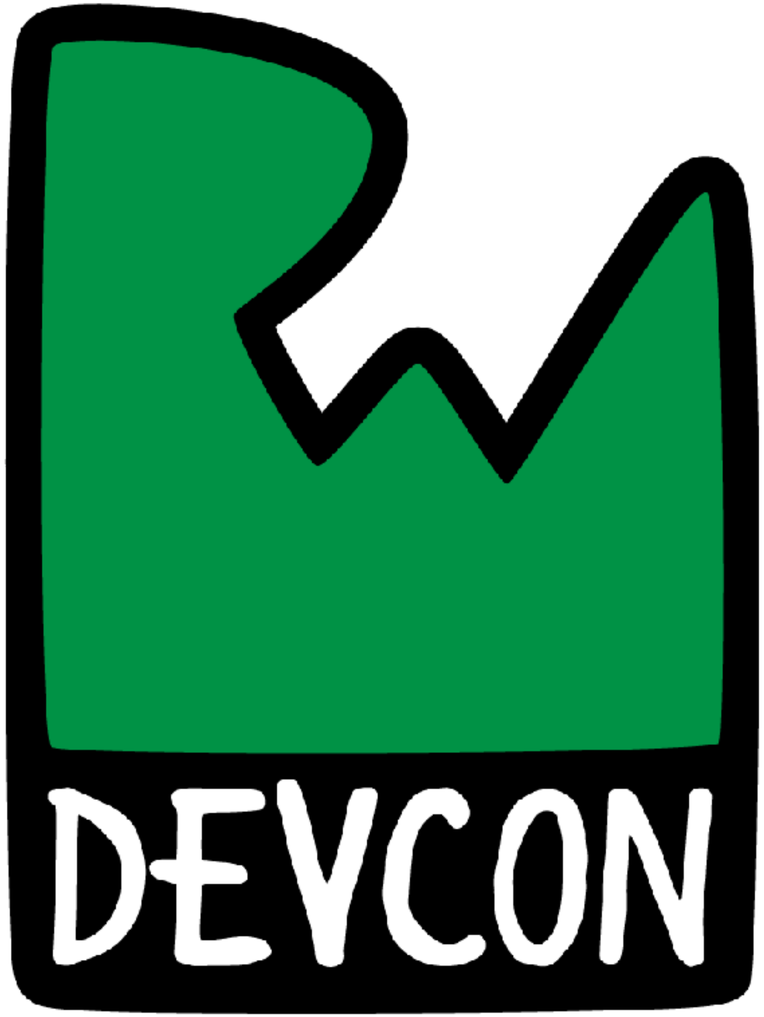
- In this demo, we'll set up Catstagram so you can see every dropped frame.
  - 1) Go to the AppDelegate and add a CADisplayLink to the main run loop
  - 2) Write a method that logs every frame drop and includes:
    - The time since the app started that the frame drop occurred
    - The amount of time since the last vsync event
  - Resources:
    - https://developer.apple.com/documentation/quartzcore/cadisplaylink
    - https://code.facebook.com/posts/456535491190613/delivering-high-scroll-performance/

Using a CADisplayLink

- Tracking when the screen refreshes

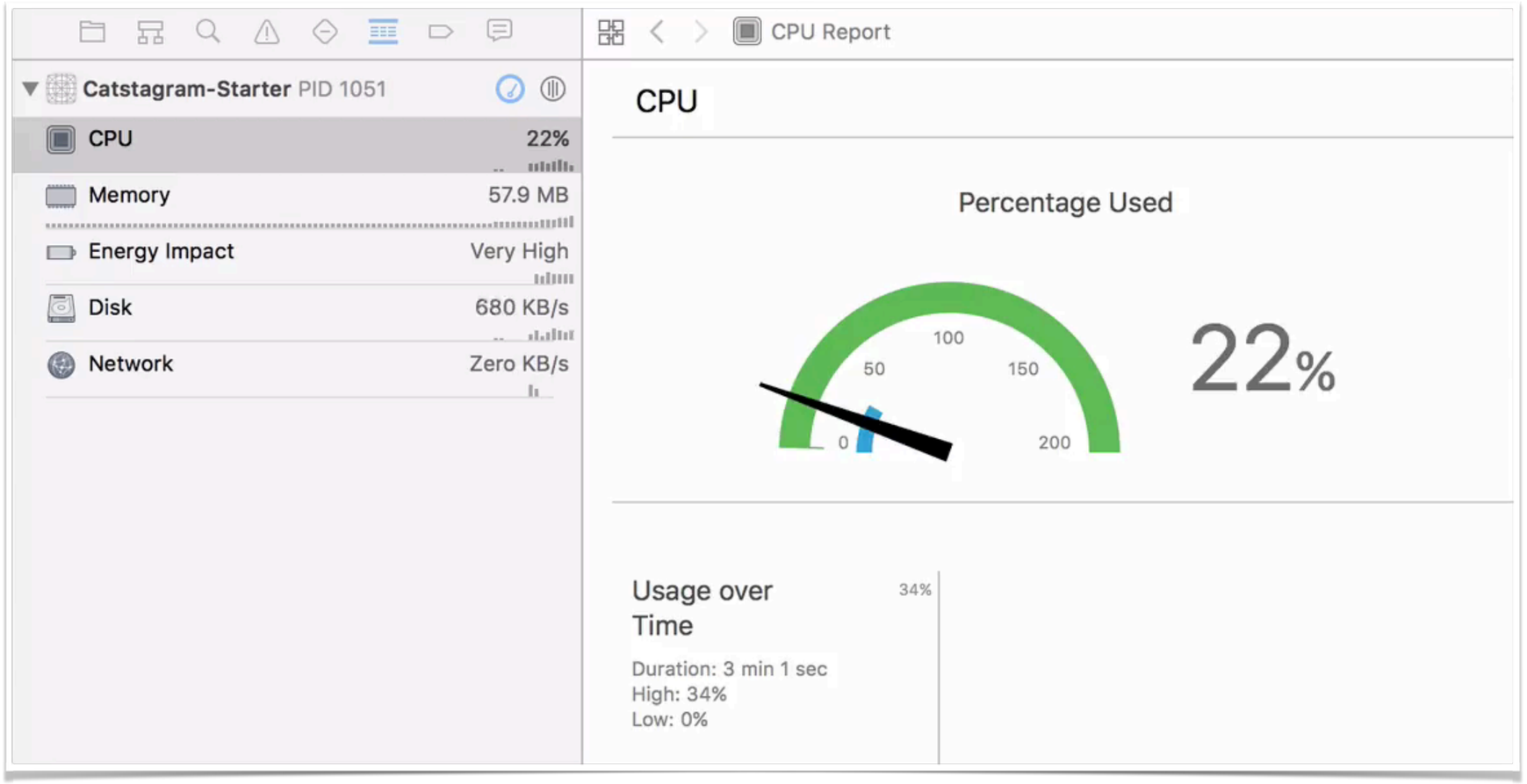
```
let link = CADisplayLink(target: self, selector: #selector(AppDelegate.update(link:)))
link.add(to: RunLoop.main, forMode: .commonModes)


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
}
```

Practical Instruments

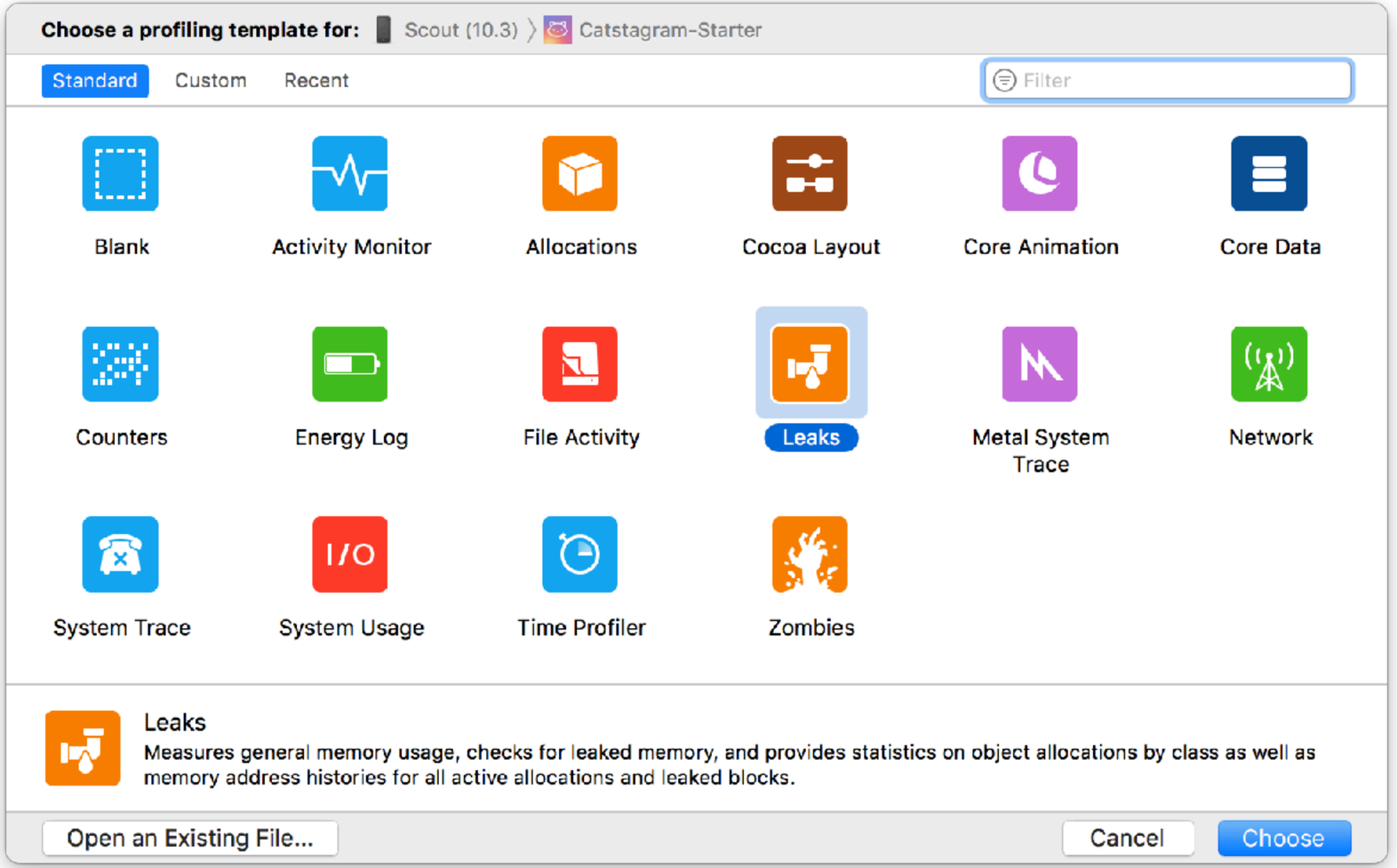


Intro to Instruments

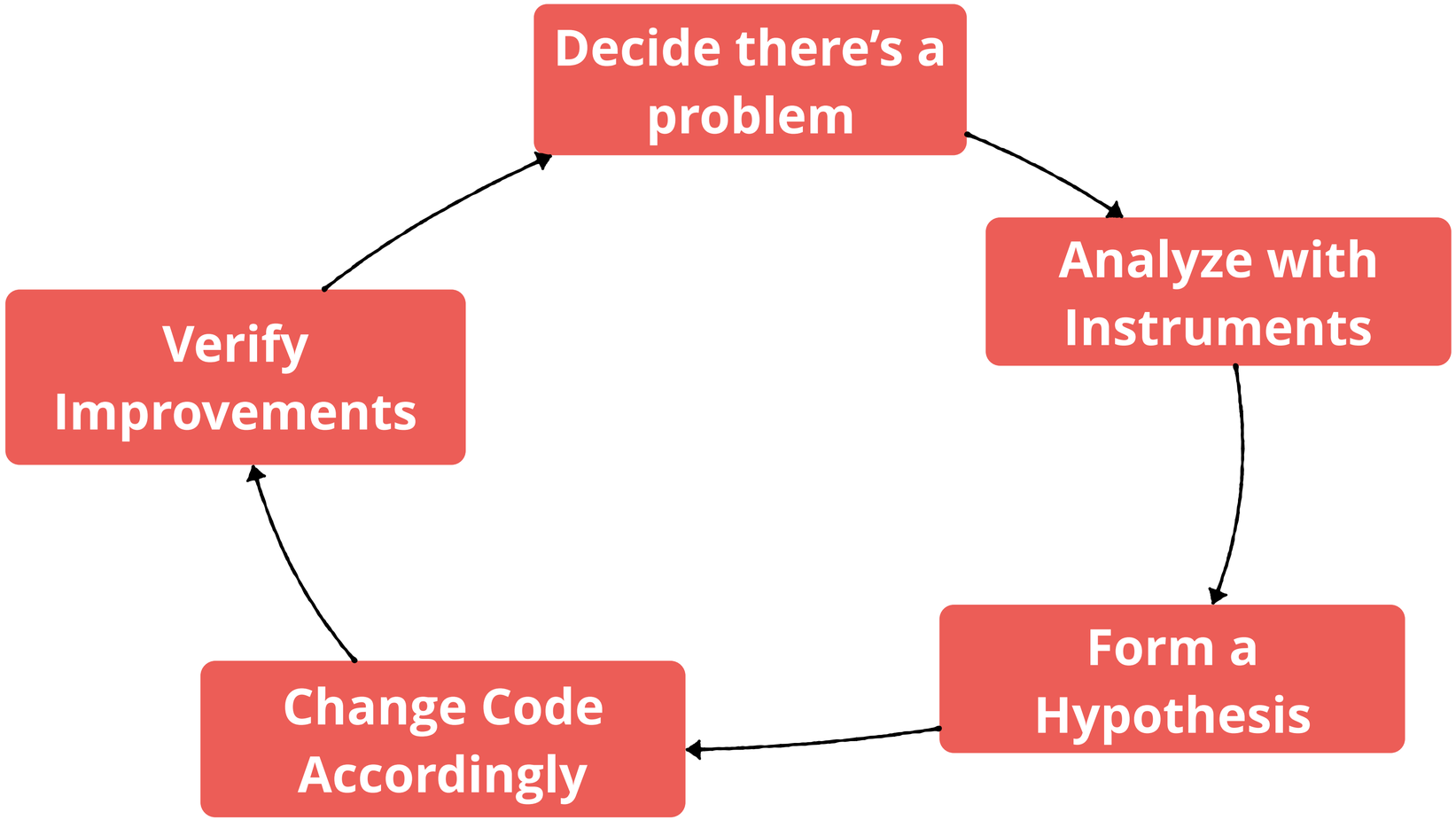
Debug Gauges



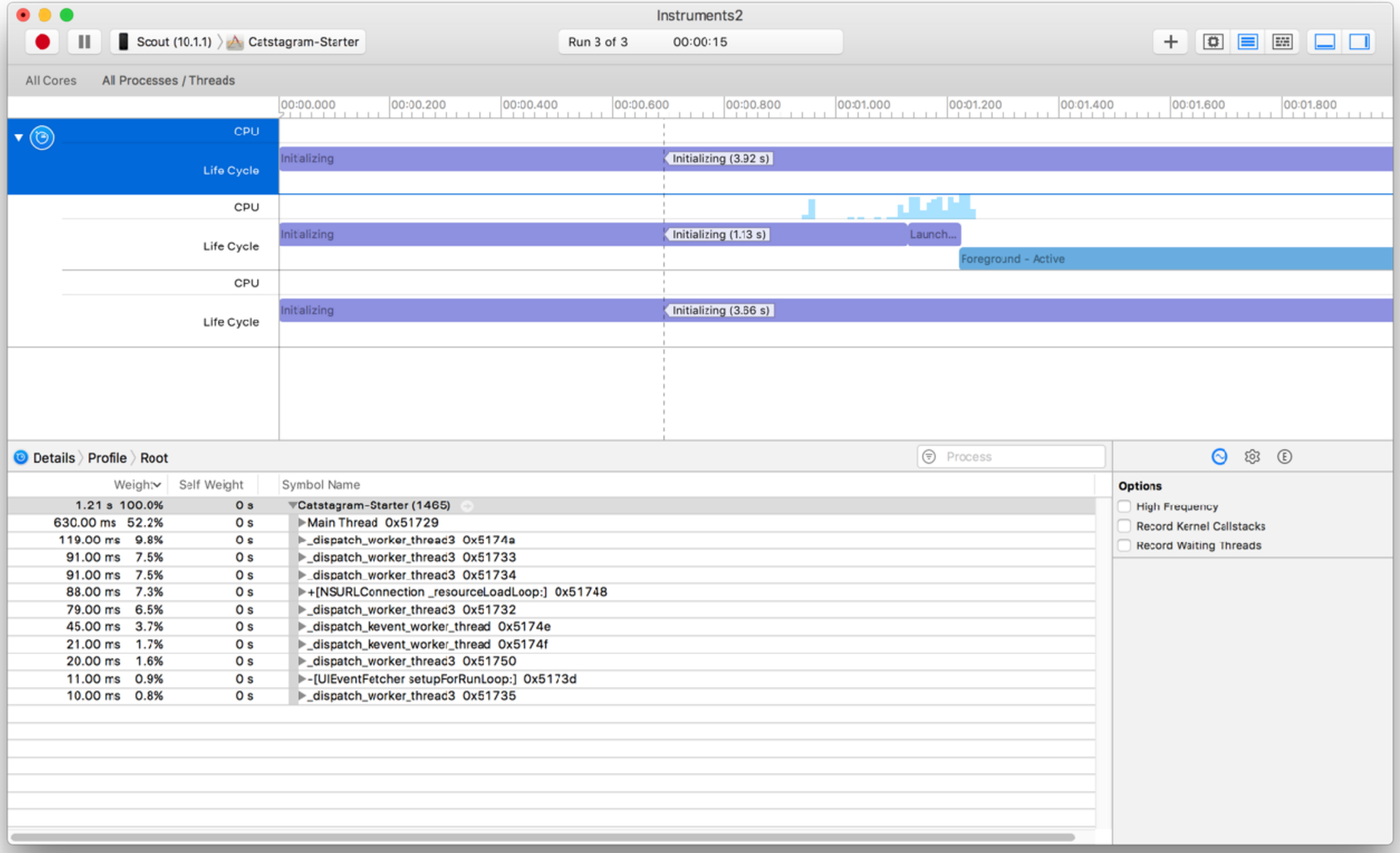
Profiling Templates



Instruments Workflow

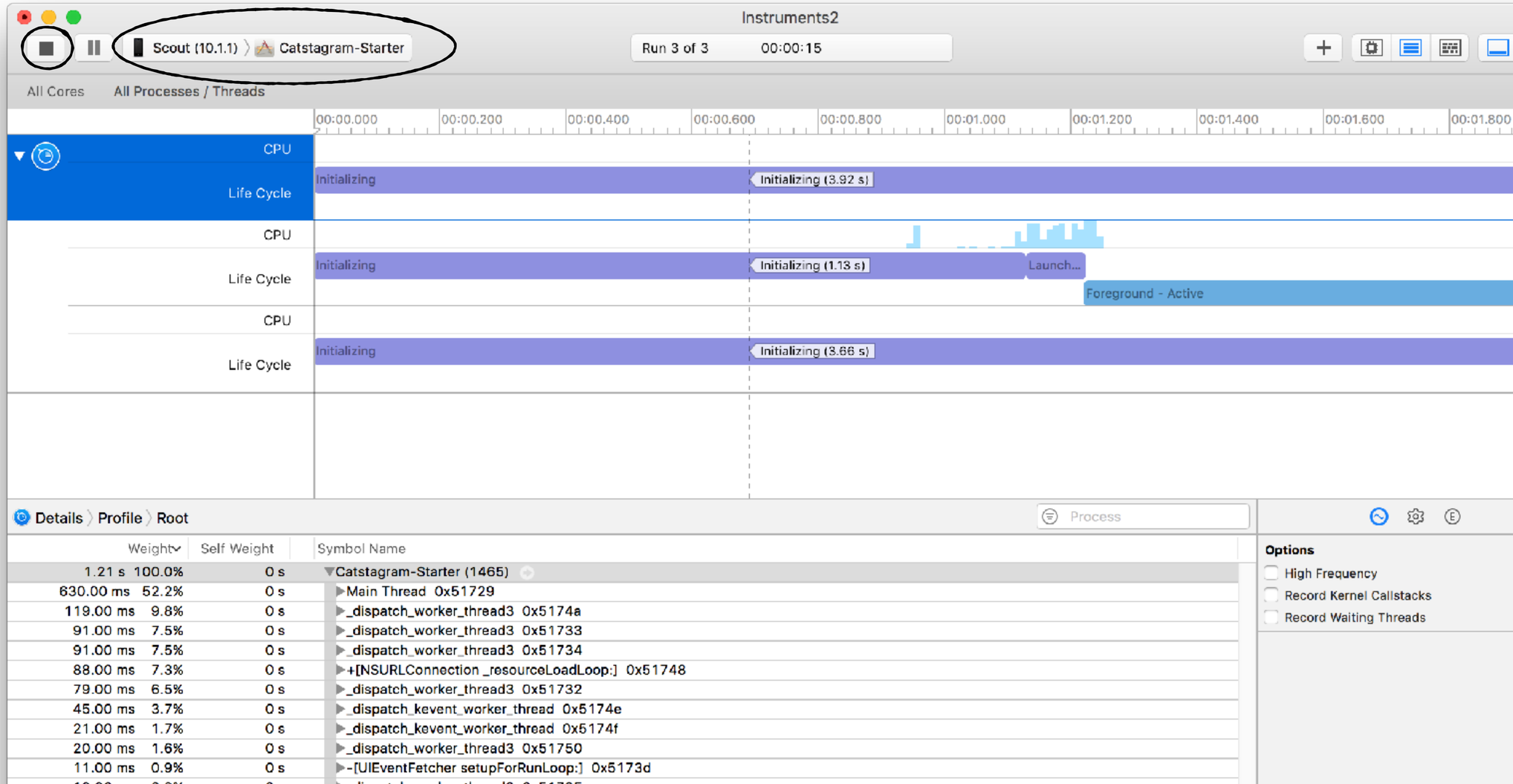


Anatomy of a Trace

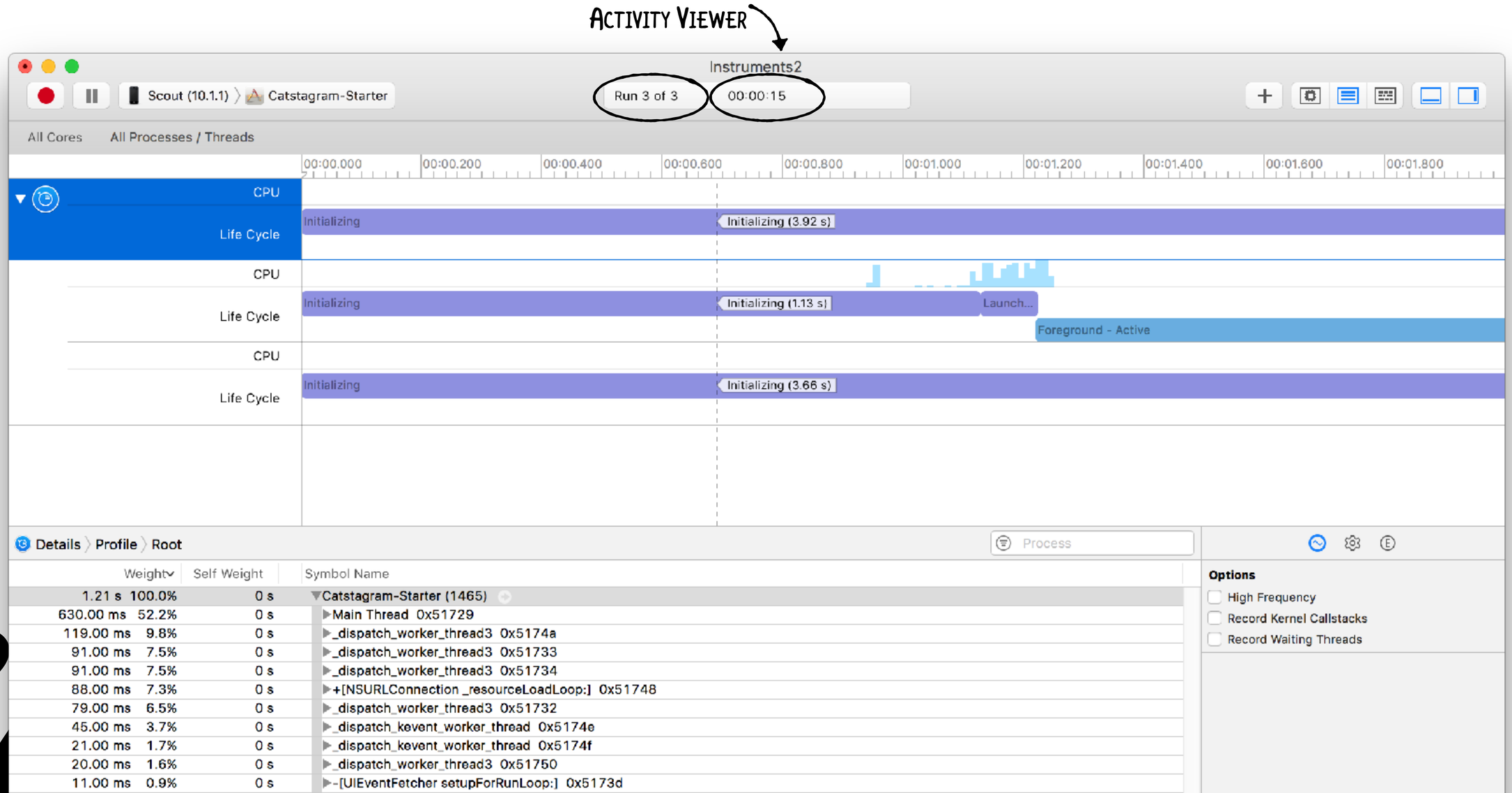


Anatomy of a Trace

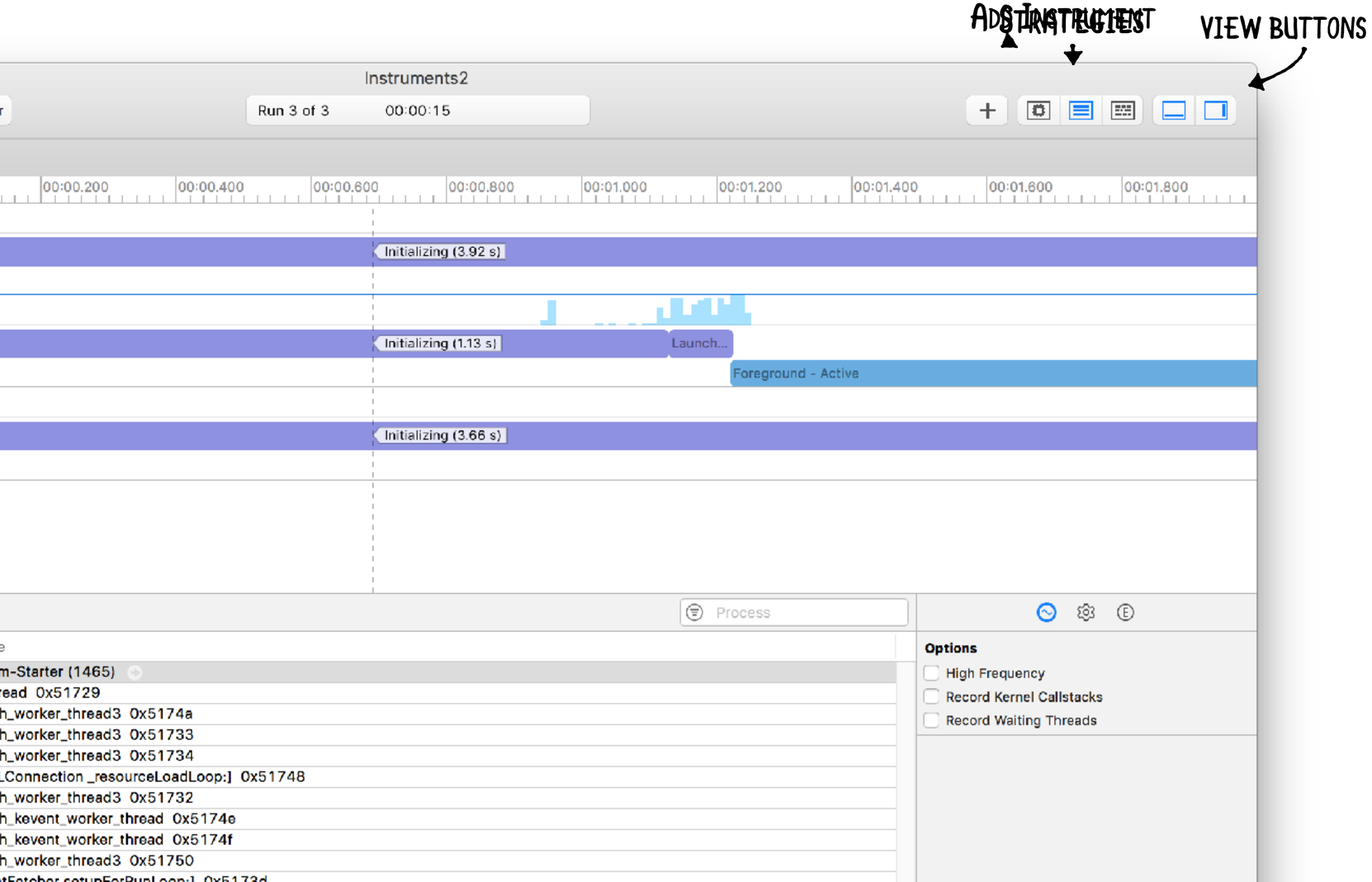
Toolbar



Anatomy of a Trace

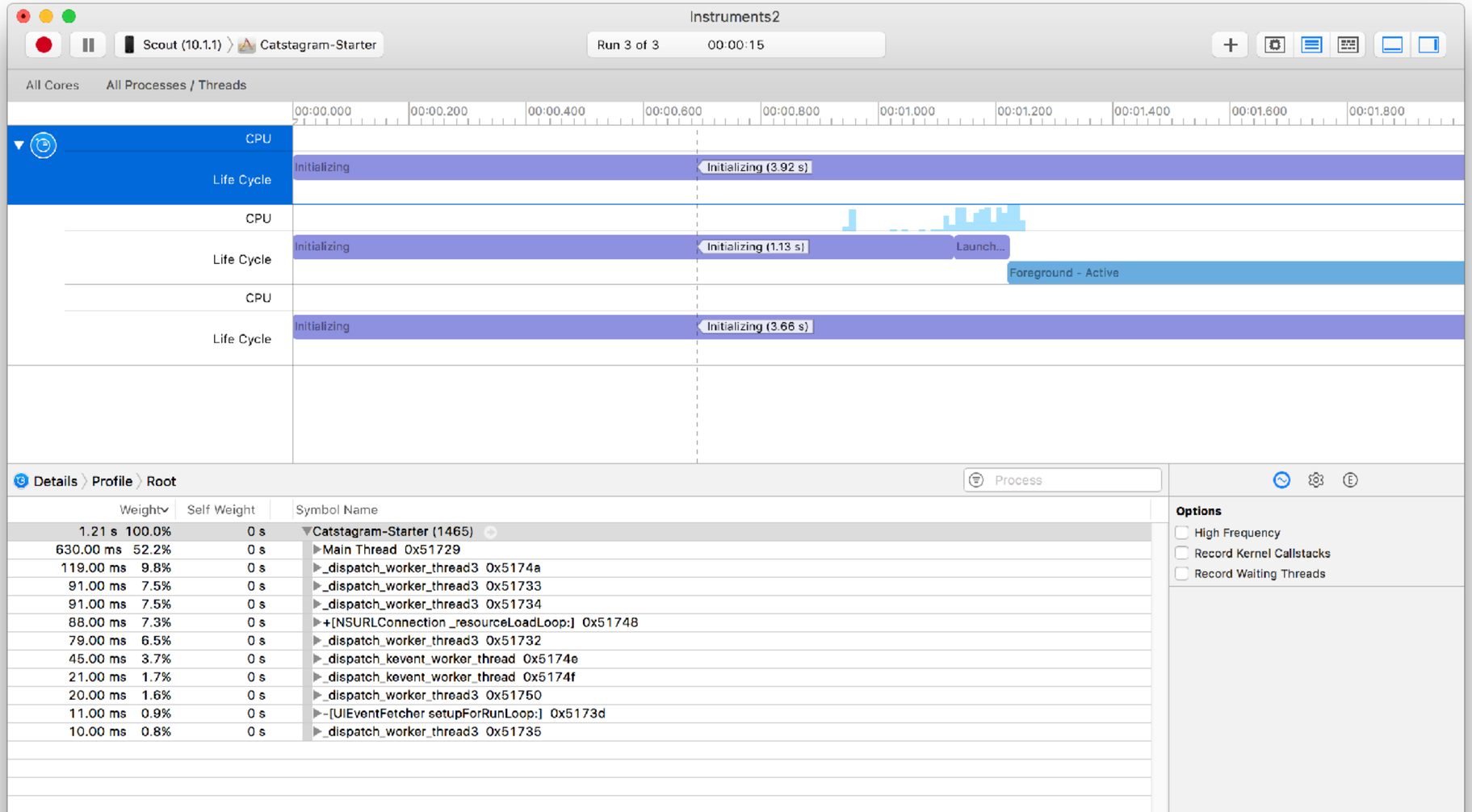


Anatomy of a Trace

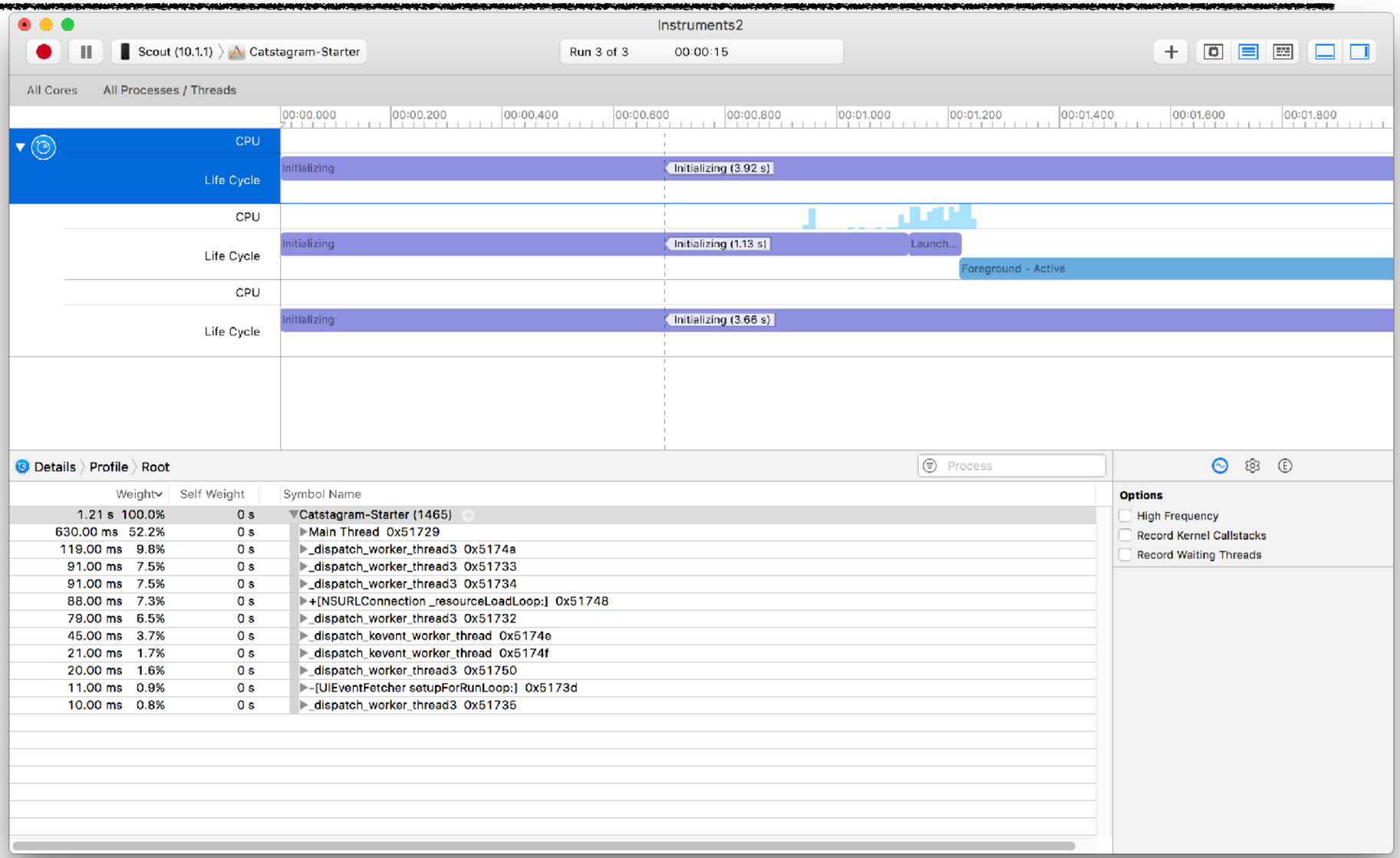


Anatomy of a Trace

TimeLine



Anatomy of a Trace



Detail Pane

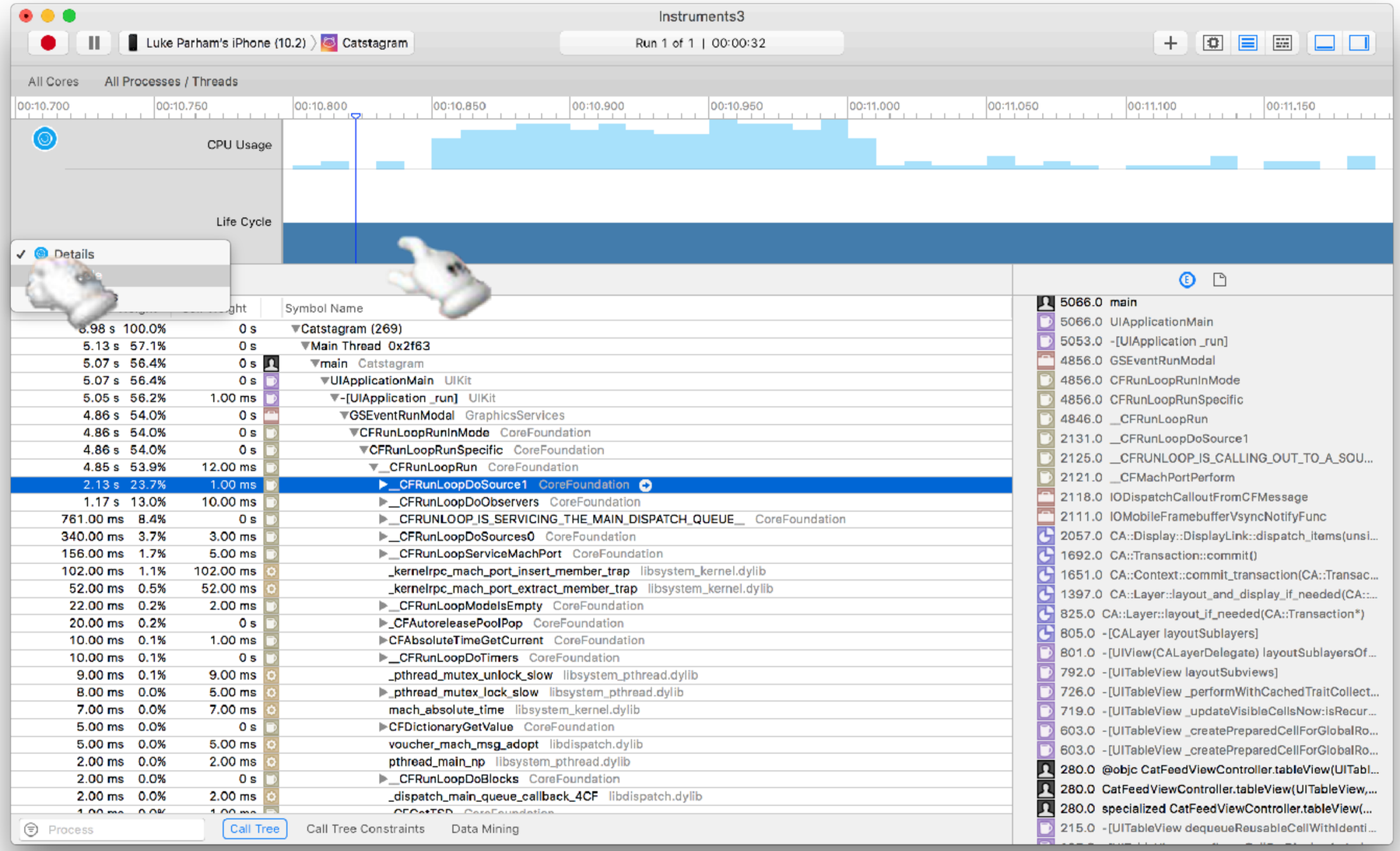
Challenge #1!

10 MINUTES

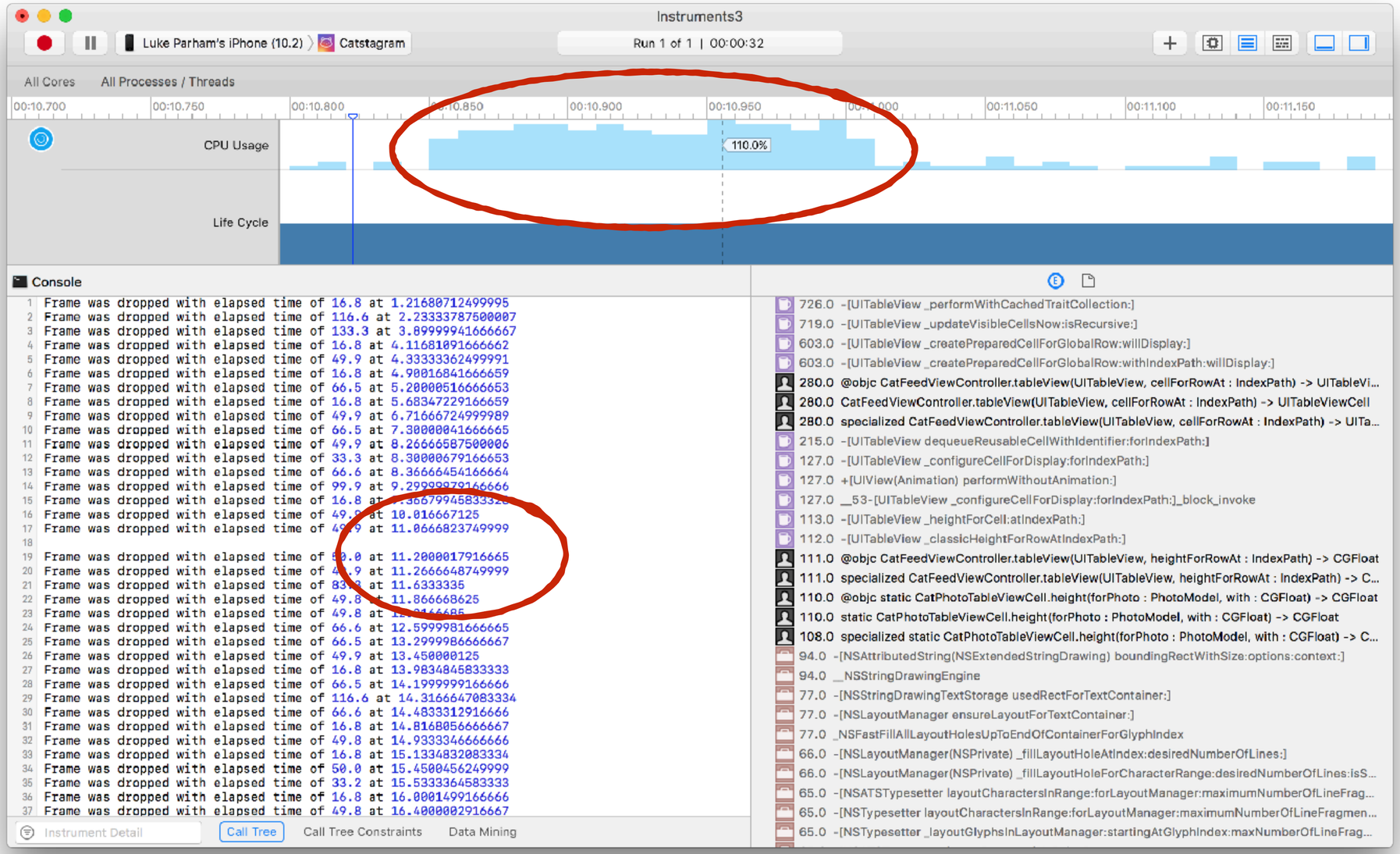
PROFILING CATSTAGRAM

- For this challenge, you'll get more comfortable with exploring a profiling trace
  - 1) Start the sample app in Profiling Mode and try out a few of the Instruments
  - 2) Start a Time Profiler session and find where your dropped frame logs are being printed
  - 3) Zoom in on one dropped frame in particular in the timeline pane

AN EXAMPLE TRACE



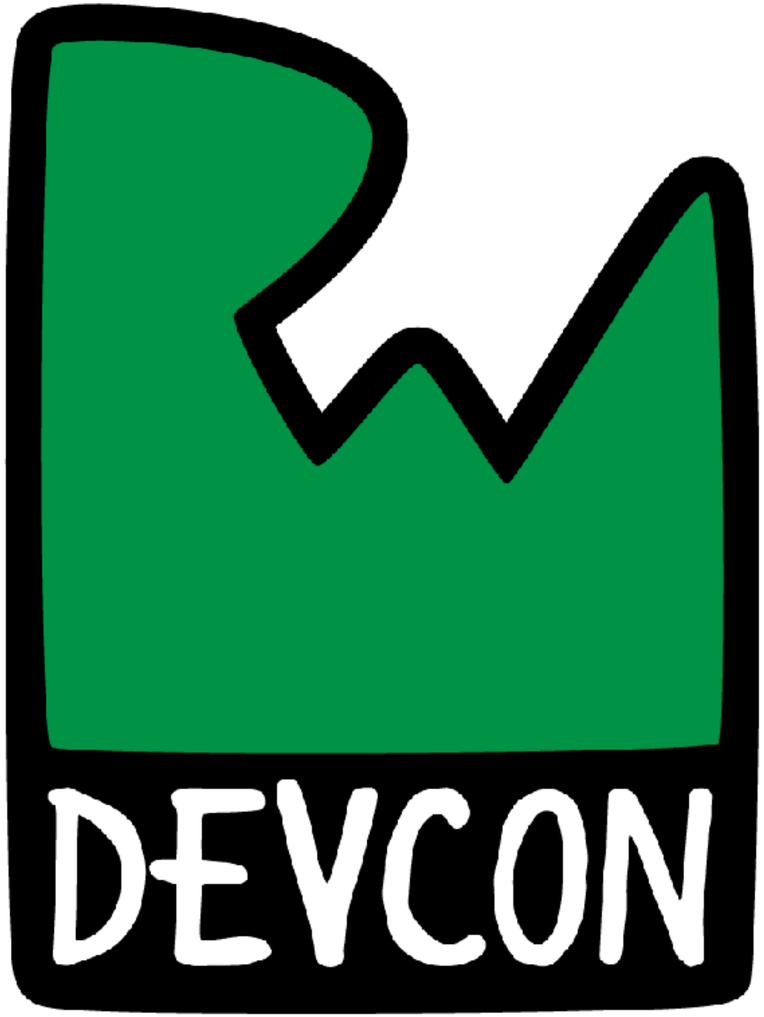
DROPPED FRAMES IN ACTION



BREAK TIME!

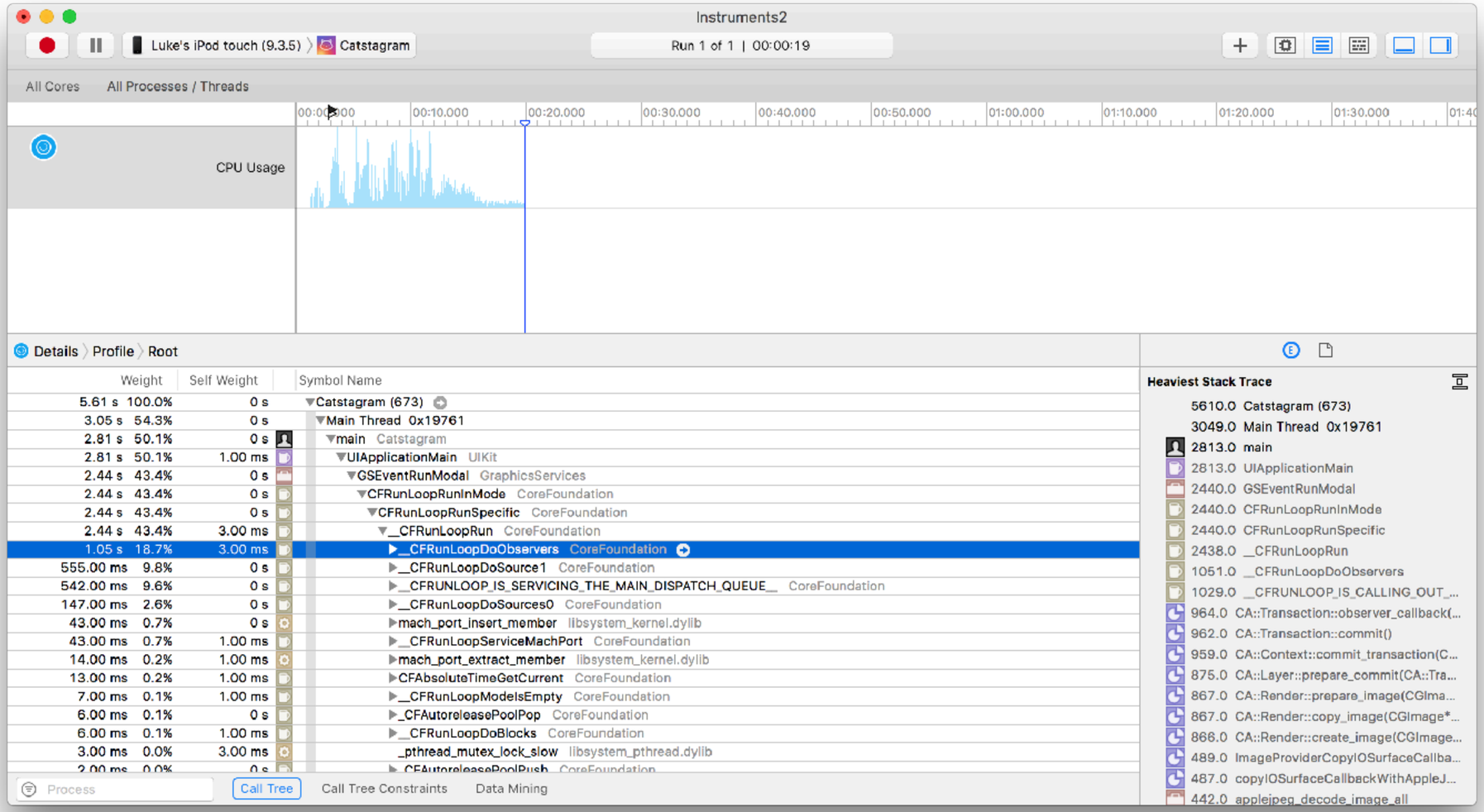
10 MINUTES

Practical Instruments

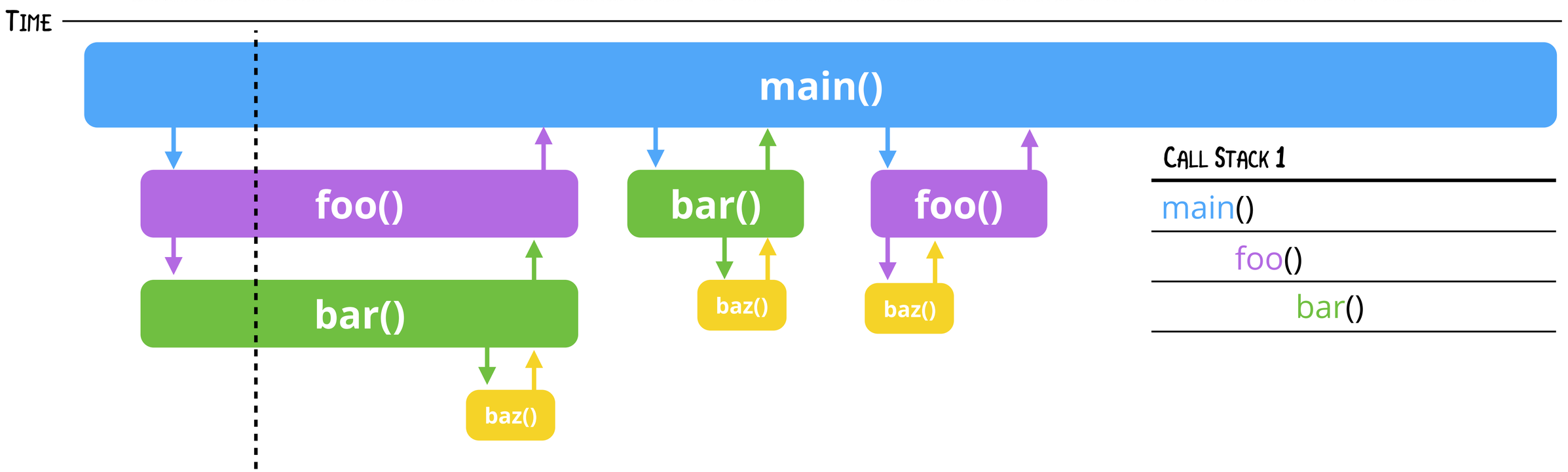


TIME PROFILER

The Call Tree



The Call Tree



Call Stack 1

main()

foo()

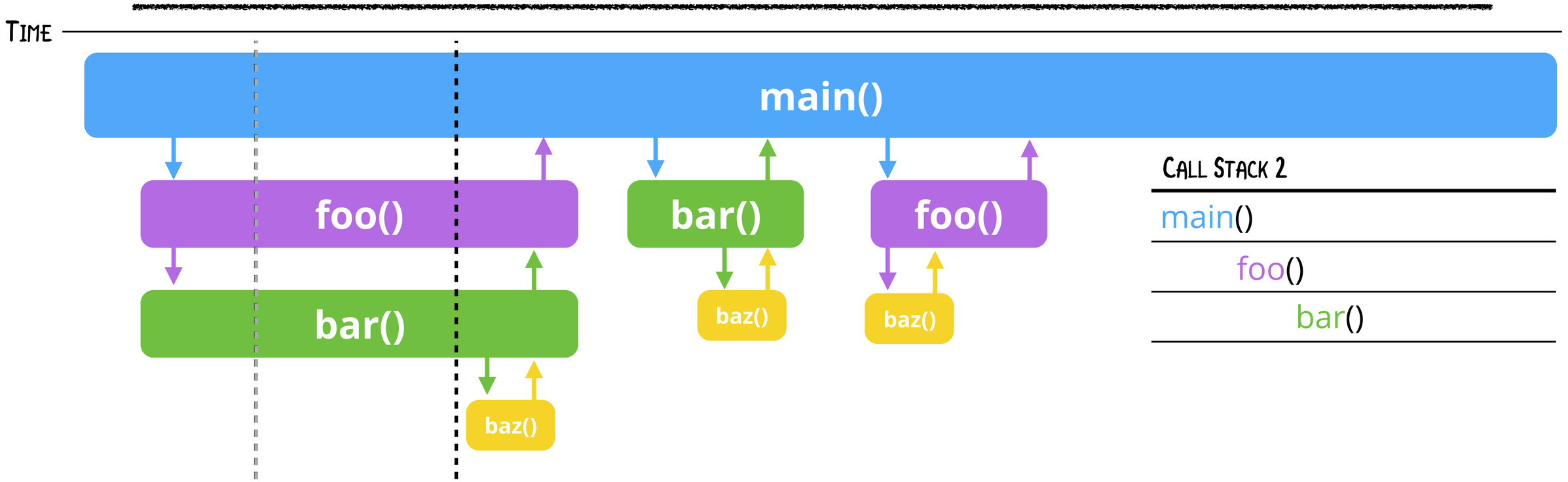
bar()

1

1

1

The Call Tree



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

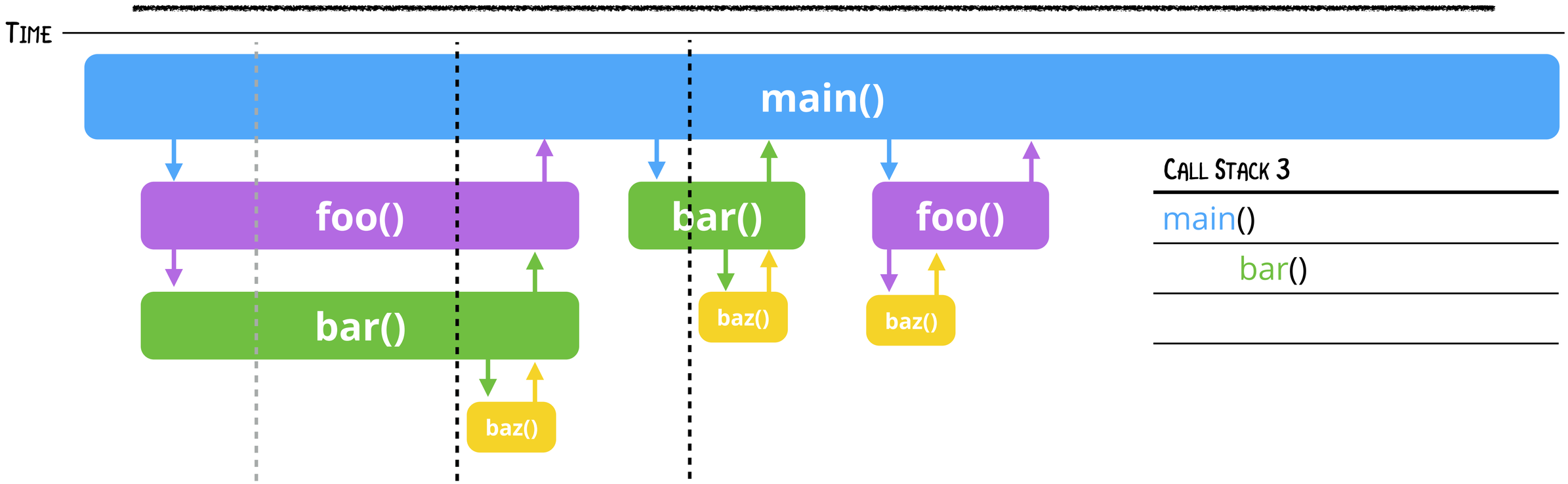
Call Stack 2

main()

foo()

bar()

The Call Tree



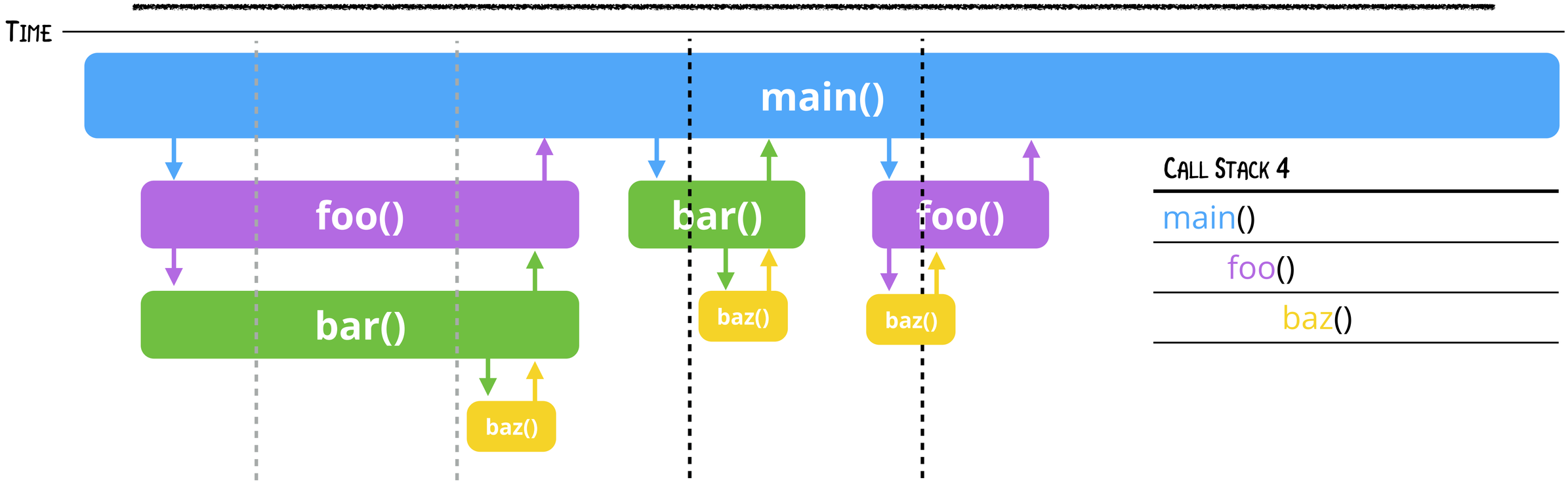
3	main()
2	foo()
2	bar()
1	bar()

Call Stack 3

main()

bar()

The Call Tree



~~3~~ 4	main()
~~2~~ 3	foo()
2	bar()
1	bar()

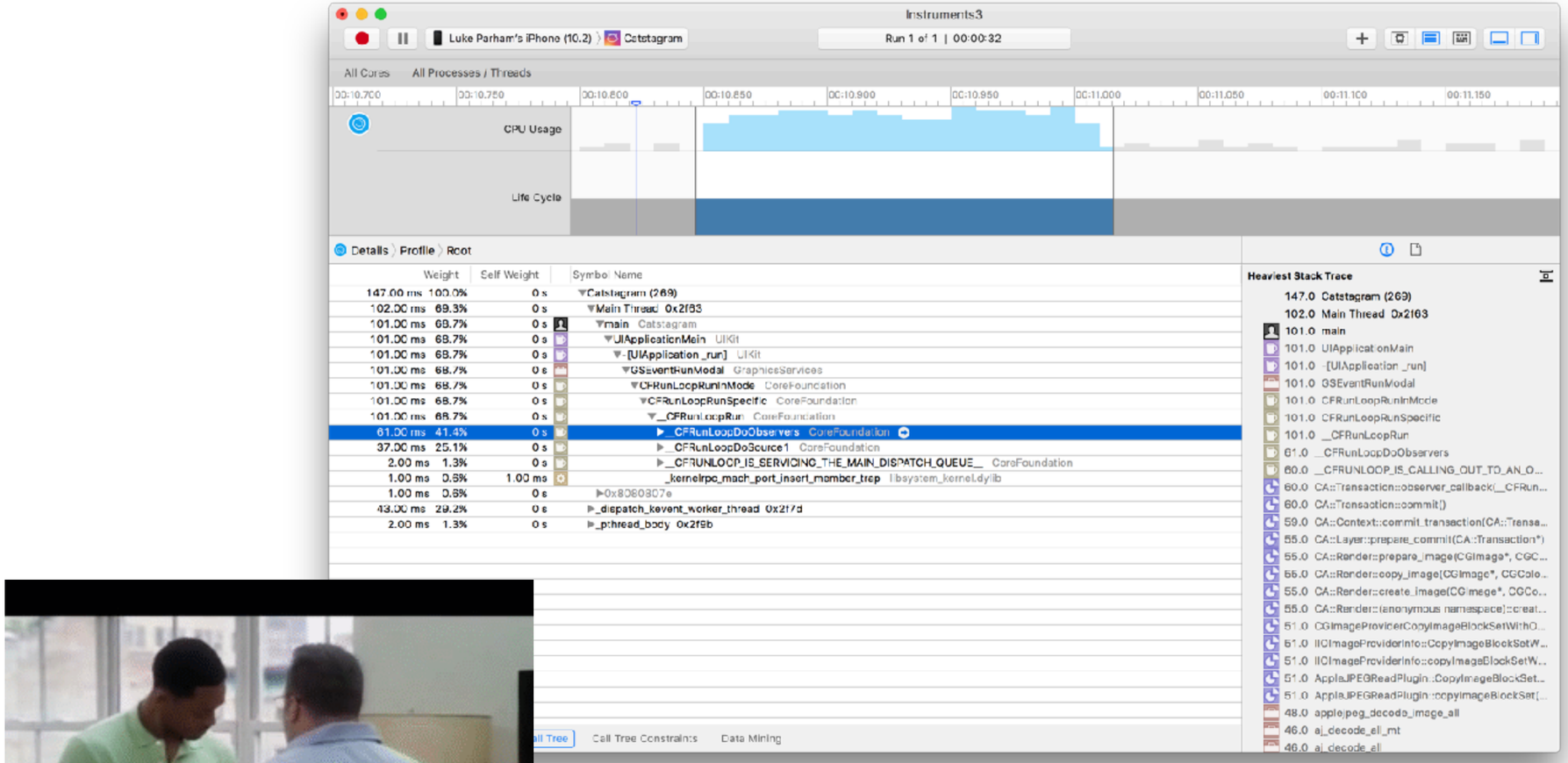
Call Stack 4

main()

foo()

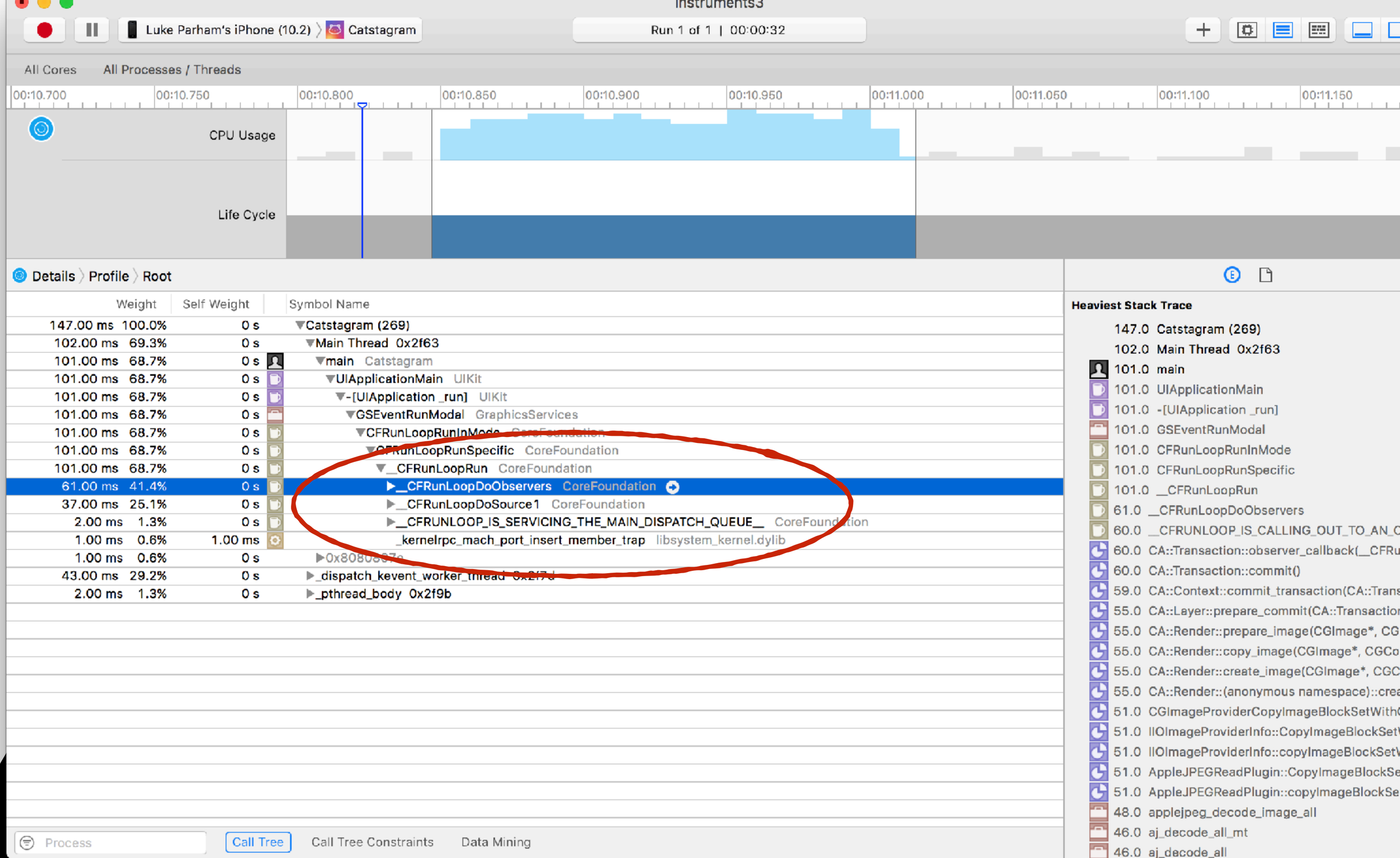
baz()

This is Where You Live



Quick Explanation of Run Loops

- Figure this out


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
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 CA::Transaction::commit()
Process
Call Tree
Call Tree Constraints
Data Mining

Challenge #2!

15 MINUTES

FINDING BOTTLENECKS

- In this challenge, you'll dive a little more deeply into the Time Profiler trace of Catstagram.
  - 1) Your only goal is to make a list of potential bottlenecks and how you think you could fix them.
    - Try using the timeline pane to zoom in on times when a lot of frames have been dropped
    - Try toggling a few of the Call Tree options on and off to see how they affect your search

* Make sure you have DSyms turned on!

Demo #2!

10 MINUTES

REMOVING BOTTLENECKS TOGETHER

- In this demo, we'll walk through what you found, and try to speed up Catstagram!
- You can follow along with me by looking at the instructions in **Demo 2**

Rule #2

Never do I/O on the main thread

Core Animation Lifecycle

- Explain the phases core animation uses from that WWDC video and also the blog post

Rule #3

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 itself
  - But...

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

DEMO #3!

20 MINUTES

DECODING IN THE BACKGROUND

- In this demo, we'll drastically improve Catstagram's responsiveness by decoding large jpegs on a background queue.
  - 1) Create a new class called AsyncImageView that's a subclass of UIView
  - 2) Make an "image" property of type UIImage with a custom setter
  - 3) Write a method that takes in images and returns a decoded version
  - 4) Finally, use the cgImage of the decoded image as the contents of the view

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, "ping" main first just in case

Not Crashing

If you're into that sort of thing

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...
}
```

Challenge #3!

15 MINUTES

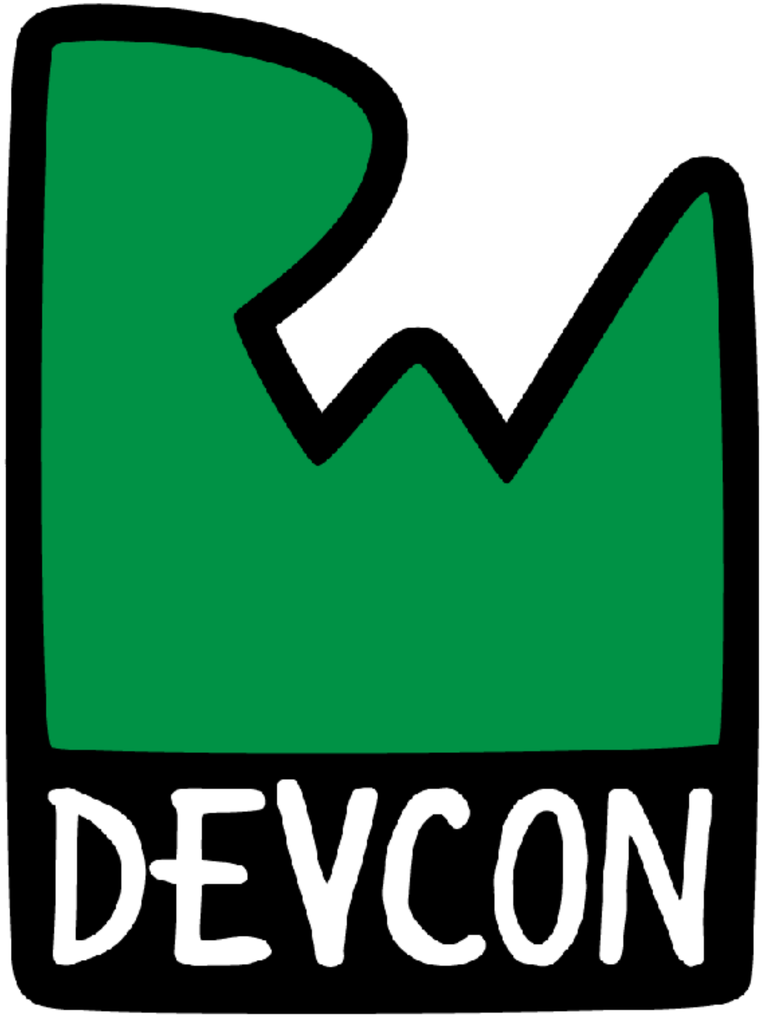
SOME QUICK IMPROVEMENTS

- In this challenge, you'll put a little more polish on your AsyncImageView.
  - 1) Add an NSCache to store and retrieve decoded images.
  - 2) Avoid crashes from memory pressure by waiting on main while allocating in the background

BREAK TIME!

10 MINUTES

Practical Instruments

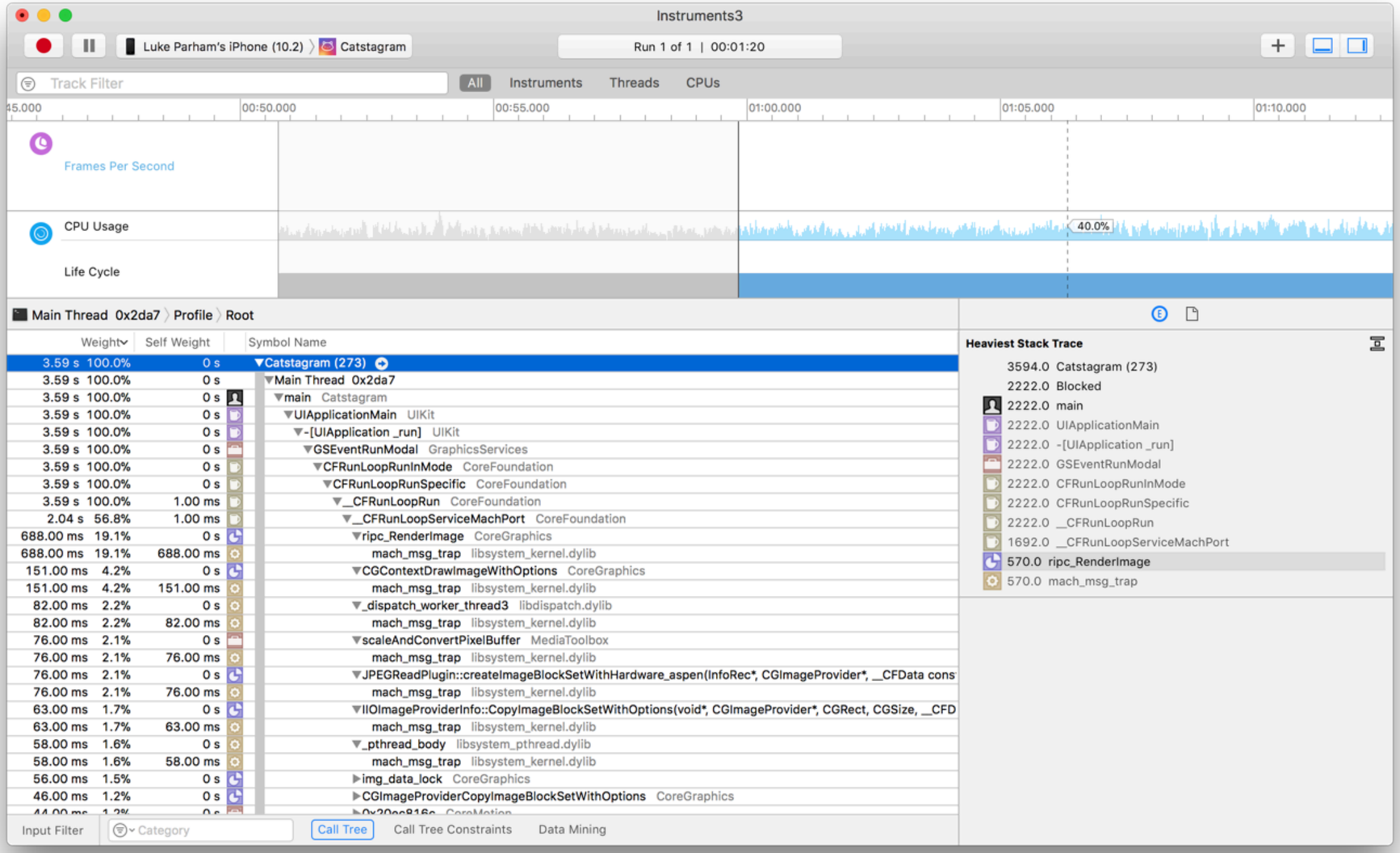


The Core Animation Instrument

Still Dropping Frames!

- When we run, we see that we're still dropping frames here and there depending on our device
- The trace doesn't tell us much, and we keep bottoming out at mach_msg_trap

MACH_MSG_TRAP



Main Thread 0x2da7 〉 Profile 〉 Root

Weight	Self Weight	Symbol Name
3.59 s 100.0%	0 s	▼Catstagram (273)
3.59 s 100.0%	0 s	▼Main Thread 0x2da7
3.59 s 100.0%	0 s	▼main Catstagram
3.59 s 100.0%	0 s	▼UIApplicationMain UIKit
3.59 s 100.0%	0 s	▼-[UIApplication _run] UIKit
3.59 s 100.0%	0 s	▼GSEventRunModal GraphicsServices
3.59 s 100.0%	0 s	▼CFRunLoopRunInMode CoreFoundation
3.59 s 100.0%	0 s	▼CFRunLoopRunSpecific CoreFoundation
3.59 s 100.0%	1.00 ms	▼__CFRunLoopRun CoreFoundation
2.04 s 56.8%	1.00 ms	▼__CFRunLoopServiceMachPort CoreFoundation
688.00 ms 19.1%	0 s	▼ripc_RenderImage CoreGraphics
688.00 ms 19.1%	688.00 ms	mach_msg_trap libsystem_kernel.dylib
151.00 ms 4.2%	0 s	▼CGContextDrawImageWithOptions CoreGraphics
151.00 ms 4.2%	151.00 ms	mach_msg_trap libsystem_kernel.dylib
82.00 ms 2.2%	0 s	▼_dispatch_worker_thread3 libdispatch.dylib
82.00 ms 2.2%	82.00 ms	mach_msg_trap libsystem_kernel.dylib
76.00 ms 2.1%	0 s	▼scaleAndConvertPixelBuffer MediaToolbox
76.00 ms 2.1%	76.00 ms	mach_msg_trap libsystem_kernel.dylib
76.00 ms 2.1%	0 s	▼JPEGReadPlugin::createImageBlockSetWithHardware_aspen(InfoRec*, CGImageProvider*, __CFData cons
76.00 ms 2.1%	76.00 ms	mach_msg_trap libsystem_kernel.dylib
63.00 ms 1.7%	0 s	▼IIOImageProviderInfo::CopyImageBlockSetWithOptions(void*, CGImageProvider*, CGRect, CGSize, __CFD
63.00 ms 1.7%	63.00 ms	mach_msg_trap libsystem_kernel.dylib
58.00 ms 1.6%	0 s	▼_pthread_body libsystem_pthread.dylib
58.00 ms 1.6%	58.00 ms	mach_msg_trap libsystem_kernel.dylib
56.00 ms 1.5%	0 s	▶img_data_lock CoreGraphics
46.00 ms 1.2%	0 s	▶CGImageProviderCopyImageBlockSetWithOptions CoreGraphics
44.00 ms 1.2%	0 s	▶0x20ec816c CoreMotion

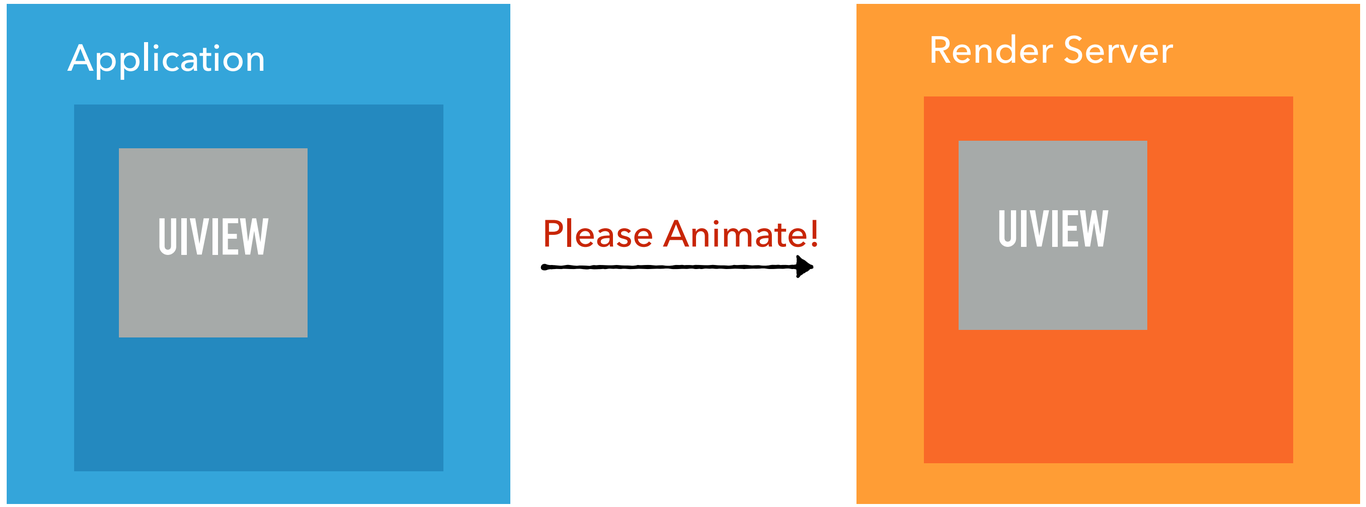
Heaviest Stack Trace

3594.0	Catstagram
2222.0	Blocked
2222.0	main
2222.0	UIApplicatio
2222.0	-[UIApplica
2222.0	GSEventRu
2222.0	CFRunLoop
2222.0	CFRunLoop
2222.0	__CFRunLoo
1692.0	__CFRunLoo
570.0	ripc_RenderI
570.0	mach_msg_t

Input Filter | Category | Call Tree | Call Tree Constraints | Data Mining

The Render Server

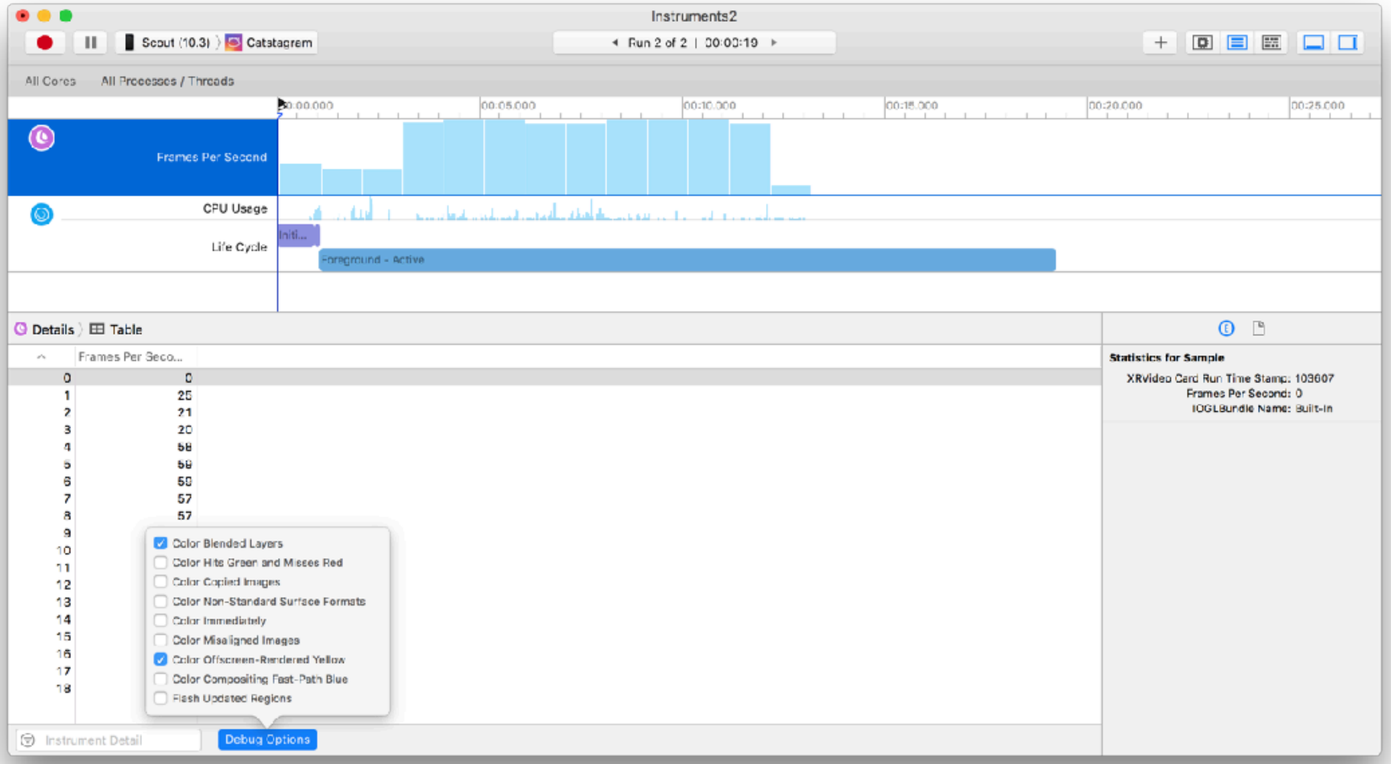
- In iOS, most everything you see on-screen happens as a result of Core Animation
  - The Render Server is an out of process service that maintains a copy of your view hierarchy and updates when CATransactions are committed.
  - Any time you create a UIView or do an animation using the UIView animation API, a CATransaction is implicitly created and committed at the bottom of the run loop.

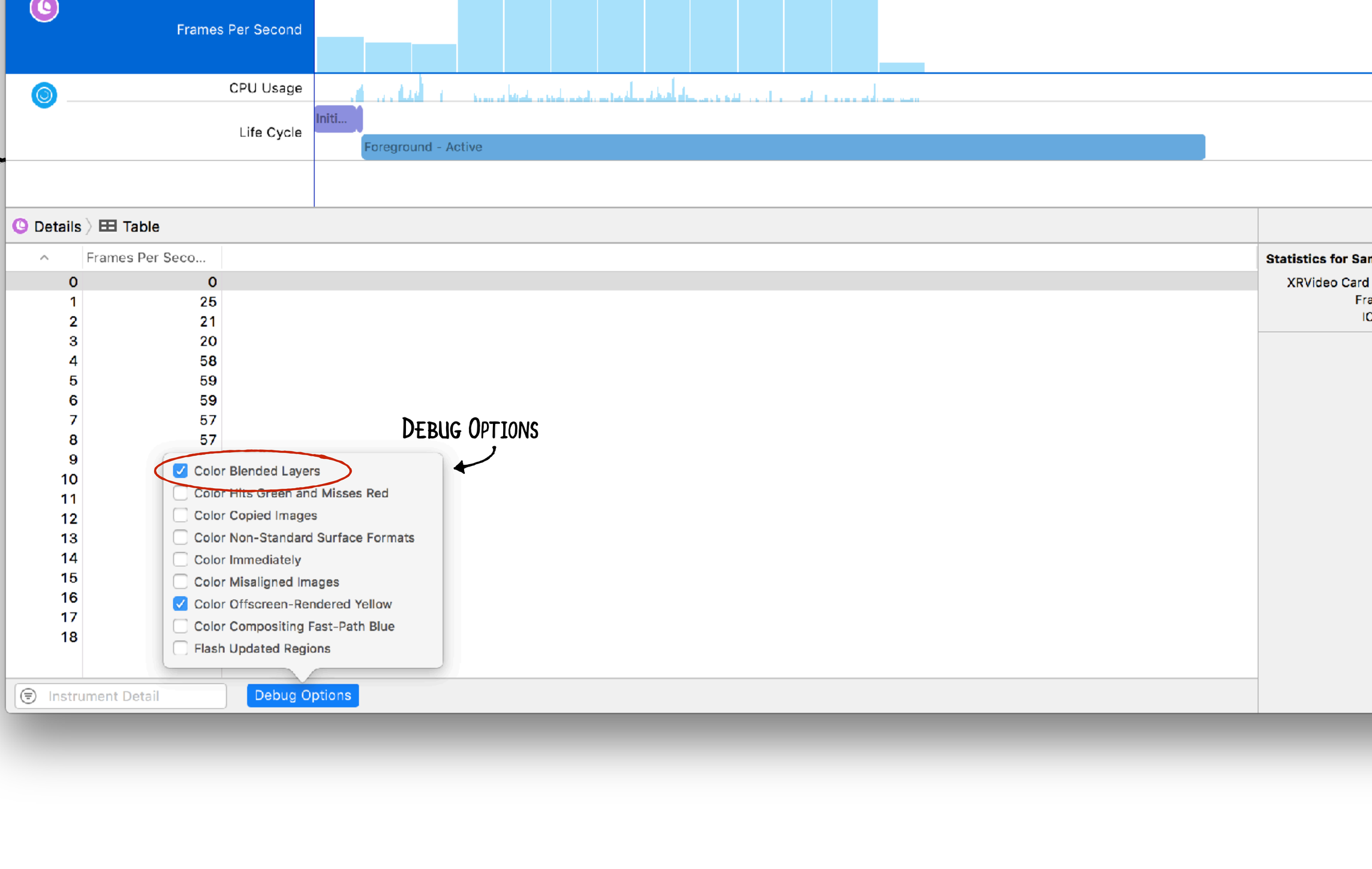


The Core Animation Instrument

CA Timeline

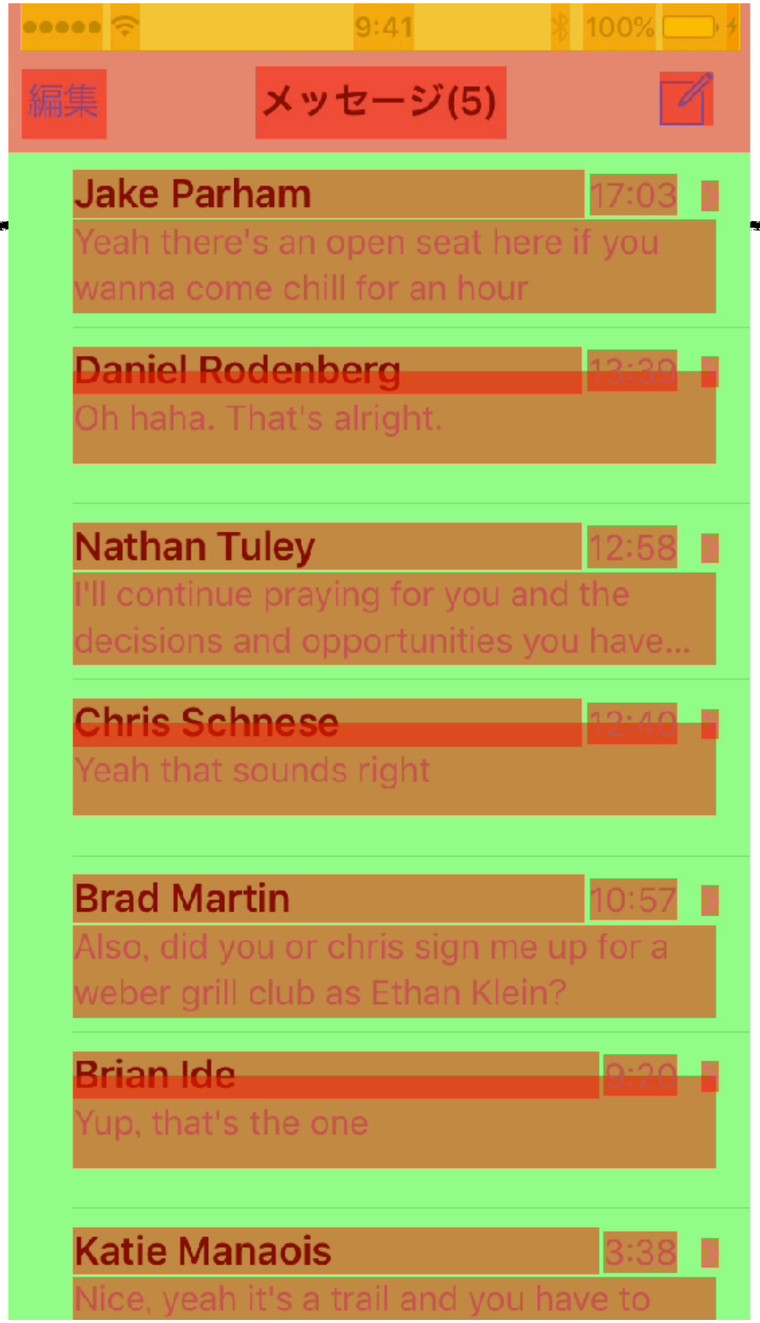
Time Profiler



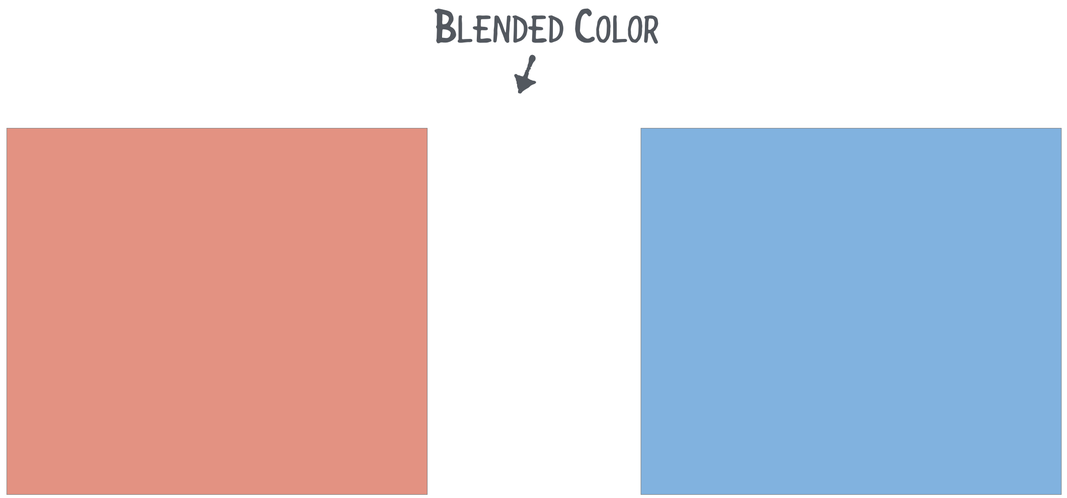

Frames Per Second
CPU Usage
Life Cycle
Initi...
Foreground - Active
Details
Table
Frames Per Seco...
0 0
1 25
2 21
3 20
4 58
5 59
6 59
7 57
8 57
9
10
11
12
13
14
15
16
17
18
Statistics for San
XRVideo Card
Fra
IO
DEBUG OPTIONS
Color Blended Layers
Color Hits Green and Misses Red
Color Copied Images
Color 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

Performance Police

Performance Police



Alpha Blending



Rule #5

Avoid unnecessary alpha blending by using opaque layers

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

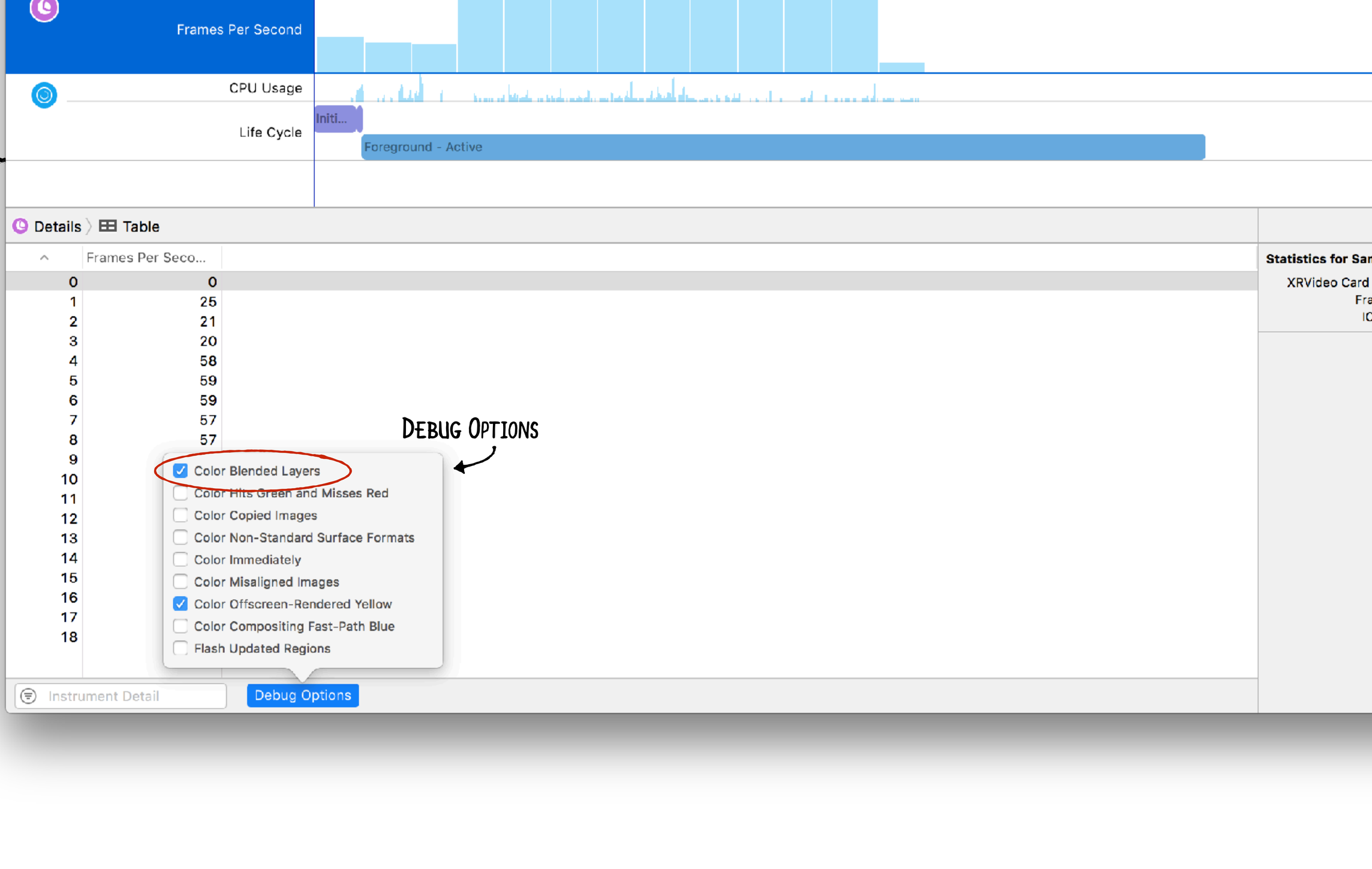
```
userNameLabel.layer.opacity = 1.0
userNameLabel.backgroundColor = .white
```

Challenge #4!

10 MINUTES

REMOVING UNNECESSARY ALPHA BLENDING

- In this challenge, you'll lighten the computational load of the render server by removing some alpha blending.
  - 1) Use the Core Animation Instrument to see which views have alpha blending.
  - 2) For each view that's being blended, if the view beneath it is the same color, then make the view on top opaque.


Frames Per Second
CPU Usage
Life Cycle
Initi...
Foreground - Active
Details
Table
Frames Per Seco...
0 0
1 25
2 21
3 20
4 58
5 59
6 59
7 57
8 57
9
10
11
12
13
14
15
16
17
18
Statistics for San
XRVideo Card
Fra
IO
DEBUG OPTIONS
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

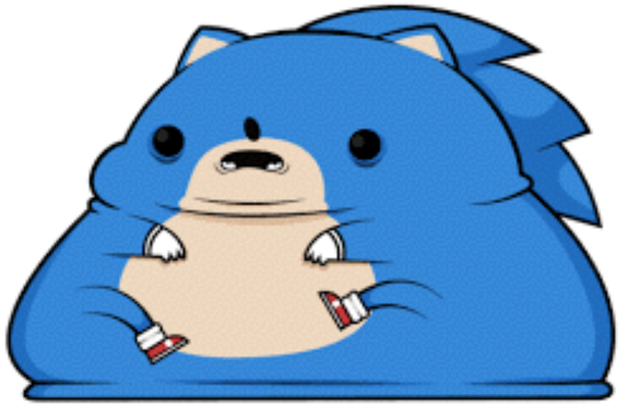
Offscreen Rendering

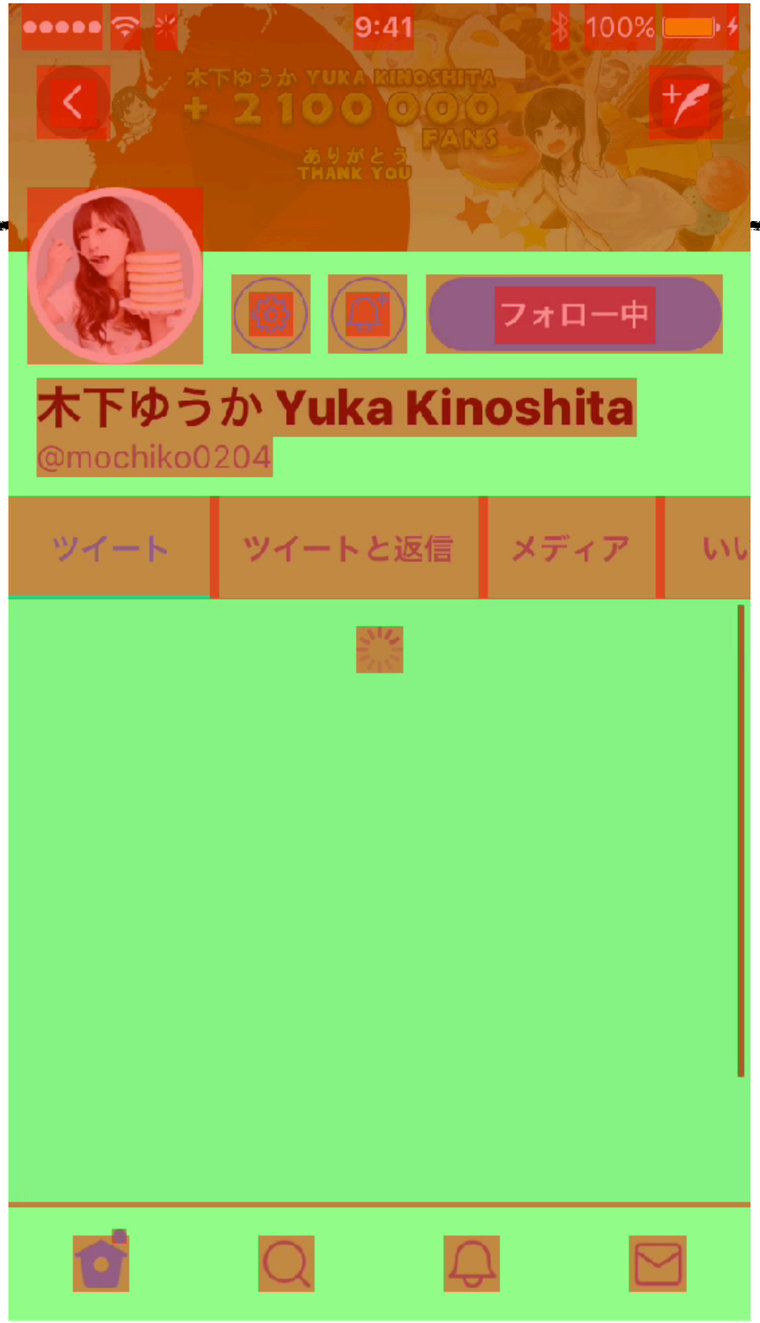
What Causes It

- **Offscreen Rendering**
  - Occurs when a layer doesn't fully know how to draw something you've told it to.
  - Happens when you:
    - Use the **cornerRadius** property
    - Use the **shouldRasterize** property
    - Render **shadows** "incorrectly"
    - Use **UIEffectsViews**

Performance Police

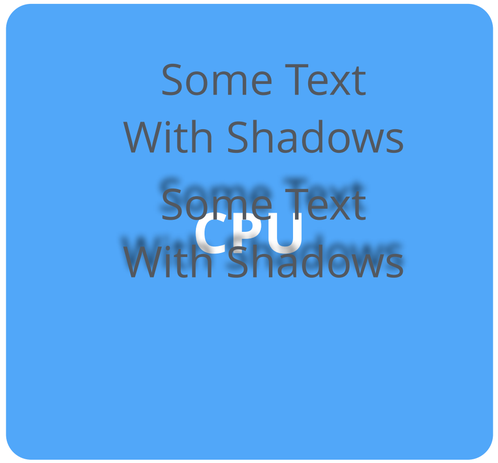
Performance Police

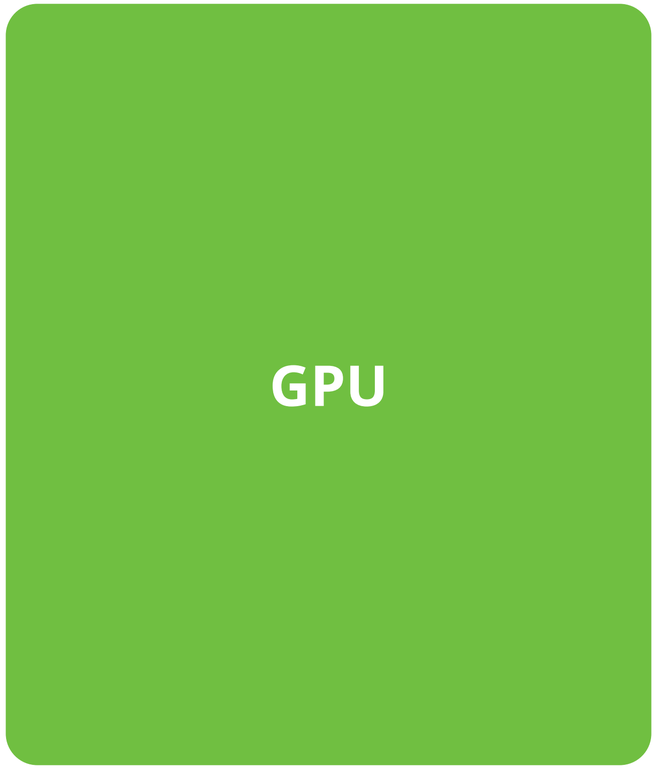


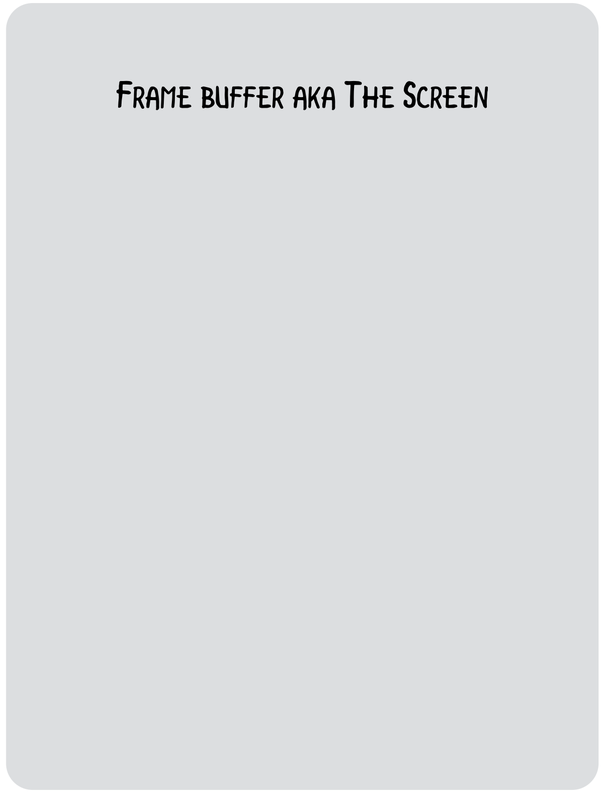


Offscreen Rendering - Shadows

Some Text
With Shadows







Rule #6

Don’t use the **cornerRadius** property

Offscreen Rendering - Corner Radius

Just don't

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!
}
```

Demo #5!

15 MINUTES

CORNER ROUNDING THE RIGHT WAY

- In this demo, we'll fix the GPU stalls caused by offscreen rendering from corner rounding.
  - 1) Launch Catstagram in the Core Animation Instrument and look for offscreen rendering due to corner rounding.
  - 2) Go to UIImage+Extensions.swift and add the circularImage(withSize:) method.
  - 2) Go to CatPhotoTableViewCell.swift and start using this method instead of cornerRadius
    - Make sure to do the rounding on a background thread and then come back to main
    - Make sure to remove the lines that set the cornerRadius property
  - 3) Verify your changes in the Core Animation Instrument

The "shouldRasterize" Flag

- Explain how this flag works and end by saying how it's dangerous and should be used with extreme caution. Or, if you're like me...

Rule #7

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

Offscreen Rendering - "Should Rasterize"

You're doing too much, do less

```
//n00b alert
```

👍

Offscreen Rendering - Shadows

```
label.layer.shadowColor = UIColor.black.cgColor
label.layer.shadowOffset = CGSize(width: 2.0, height: 2.0)
label.layer.shadowRadius = 5.0
label.layer.shadowOpacity = 0.5
```

Rule #8

Only use CALayer's shadow properties if you can use 'shadowPath'

DEMO #6!

10 MINUTES

CREATING SHADOWS

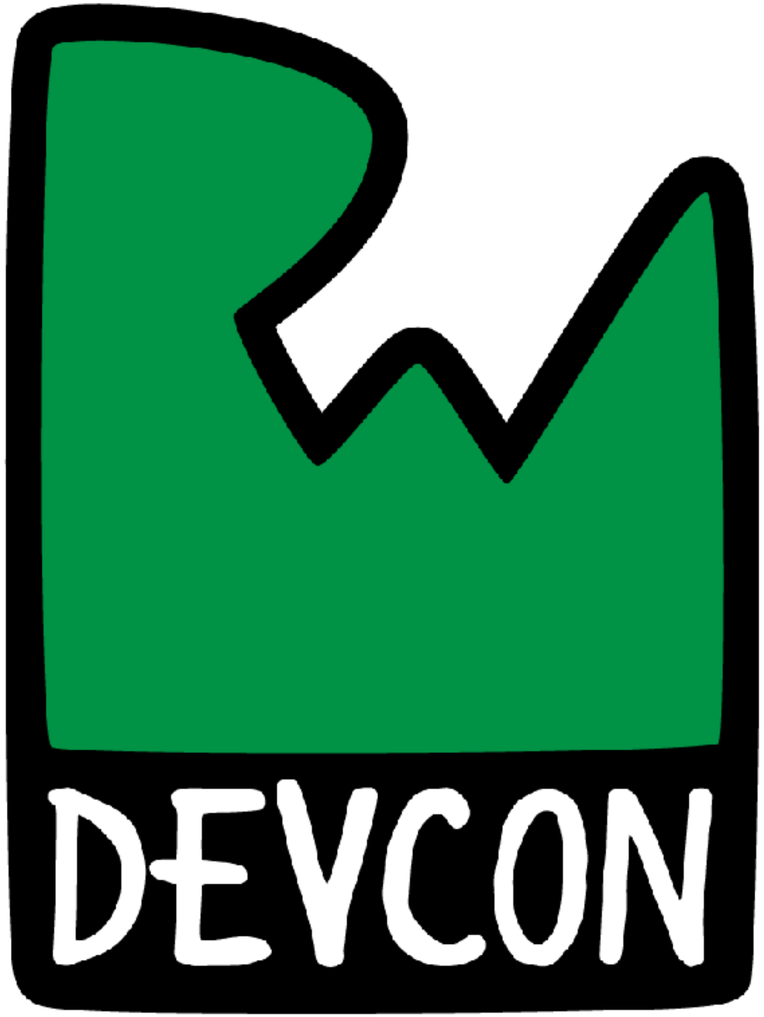
- In this demo we'll fix some more offscreen rendering by choosing a different strategy for drawing shadows.
  - In general, you can use **NSShadow** and **NSAttributedString**s to draw text shadows efficiently.
  - If you need shadows behind a normal square UIView, you can just use the **shadowPath** property with an appropriate **UIBezierPath.**

Summary

Avoid Blending and Offscreen Rendering

- **Blending**
  - Make views opaque when possible
- **Offscreen Rendering**
  - Don’t use the cornerRadius property
  - Shadows can also cause offscreen rendering
  - Don’t use shouldRasterize unless you really know it helps
    - Use NSShadow with attributed strings for labels
    - Make sure to use the shadowPath property for other types of views

Conclusions



Where We've Been and Where You'll Go

What We've Learned

- You should be tracking performance in your app.
  - You can use display links as well as performance tests which we didn't get into.
- Use Time Profiler to get an overview of which methods are taking the most CPU time in your app.
- Use the Core Animation template to see when your code is indirectly causing extra work for the GPU.

Rules of Good Performance

1. Use a CADisplayLink to track dropped frames
2. Never do I/O on the Main Thread
3. Decode JPEGs in the background as well
4. Avoid unnecessary blending
5. Don’t use the cornerRadius property
6. Don’t use shouldRasterize
7. Use shadowPath or NSShadow for shadow rendering

Where To Go From Here

- **More Instruments**
  - Leaks and Allocations instruments are great for debugging memory issues.
  - Energy instrument can show when you're doing work that may not be slow but is causing unnecessary battery drain.
  - Check out my Practical Instruments video course on RayWenderlich.com to get a look at these other instruments
- **More Performance**
  - Check out Marcel Weiher's great book iOS and macOS Performance Tuning to see some advanced techniques for really tightening up an apps' performance.
  - (Heads up it's mostly about Objective-C, for good reason)