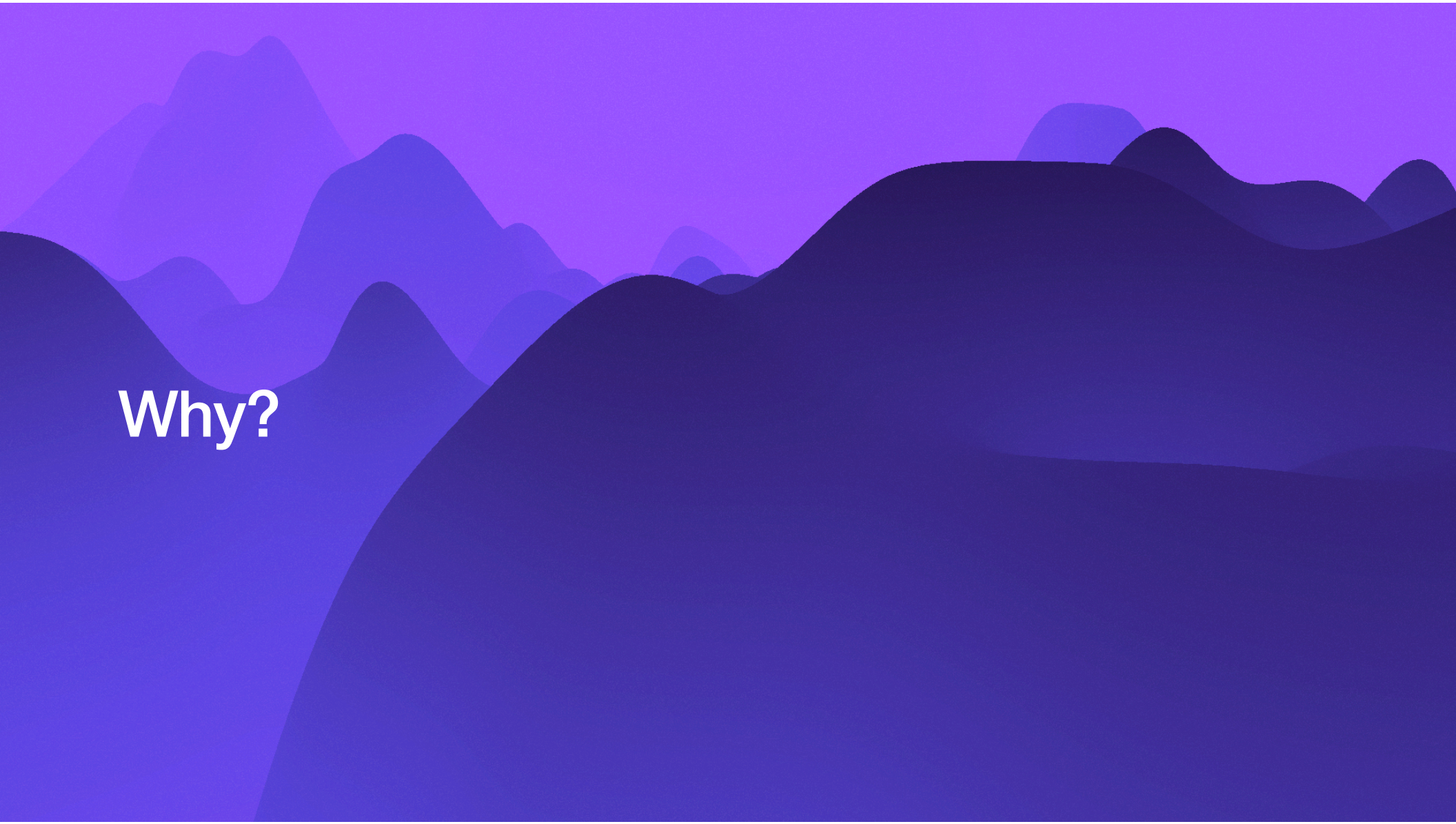
Insights into SwiftUI Performance Optimization

DC iOS

Mac Bellingrath, October 2023


Why?


What is performance?
The Render Loop
View Identity and Lifetime
Instruments
Questions


What do I mean by performance?

What is performance?

- We want our app to be responsive to user input
- Free of hitches and hangs
- Stable
- Efficient

What is performance?

- We want our app to be responsive to user input
- Free of hitches and hangs
- Stable
- Efficient

SwiftUI abstracts its render loop but there are practical scenarios where understanding it is beneficial.

There is a reason the two SwiftUI performance talks include “Demystify” in their titles 🔮


The SwiftUI Render Loop

onAppear

Will we ever see the text “initial?”

Is this view “rendered” twice?

```
@Observable
class Model {
    var status: String = "initial"

    func fetch() {
        status = "fetching"
        // ...
    }
}

struct ContentView: View {
    @Bindable var model = Model()

    var body: some View {
        Text(model.status)
            .onAppear { model.fetch() }
    }
}
```

onAppear

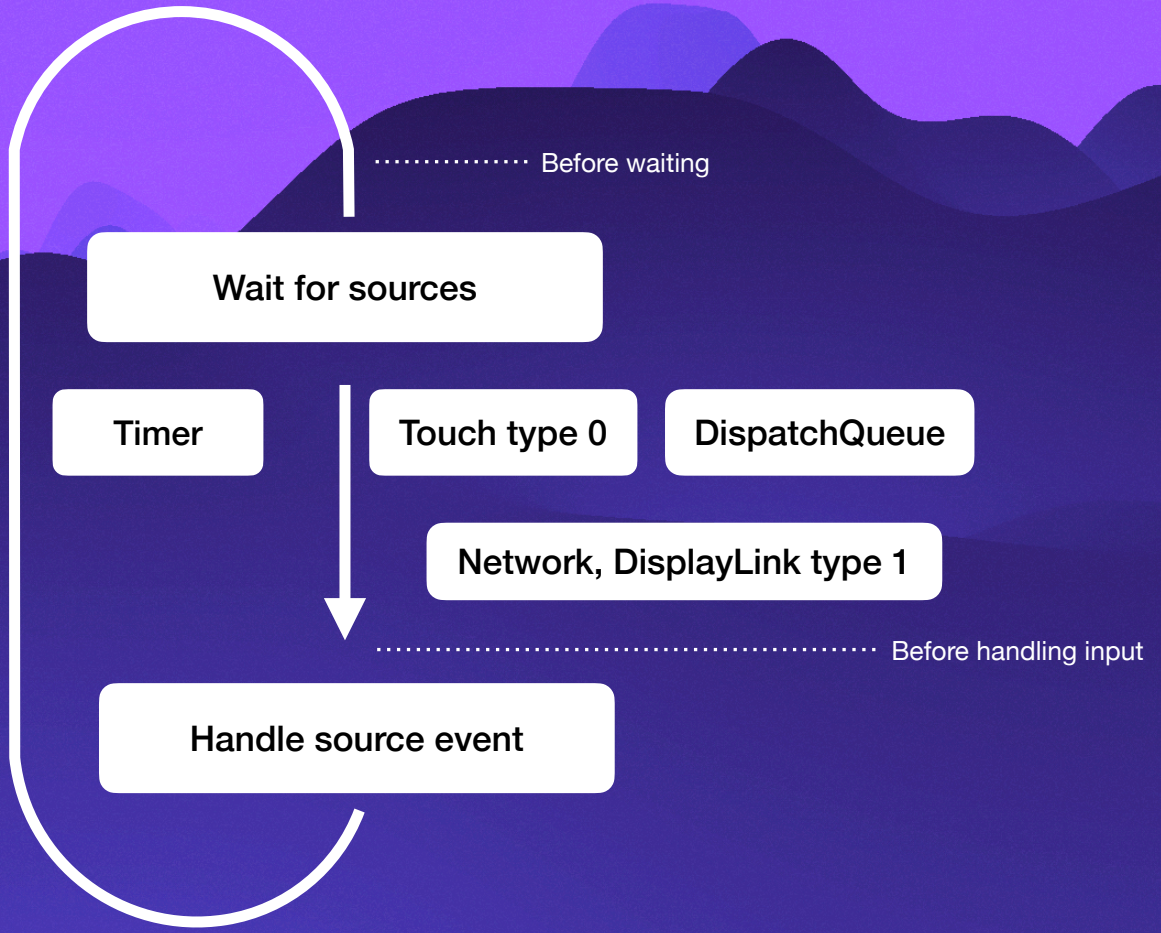
We will never see “initial” and the view will only be rendered once.

To understand why, must look at render loop

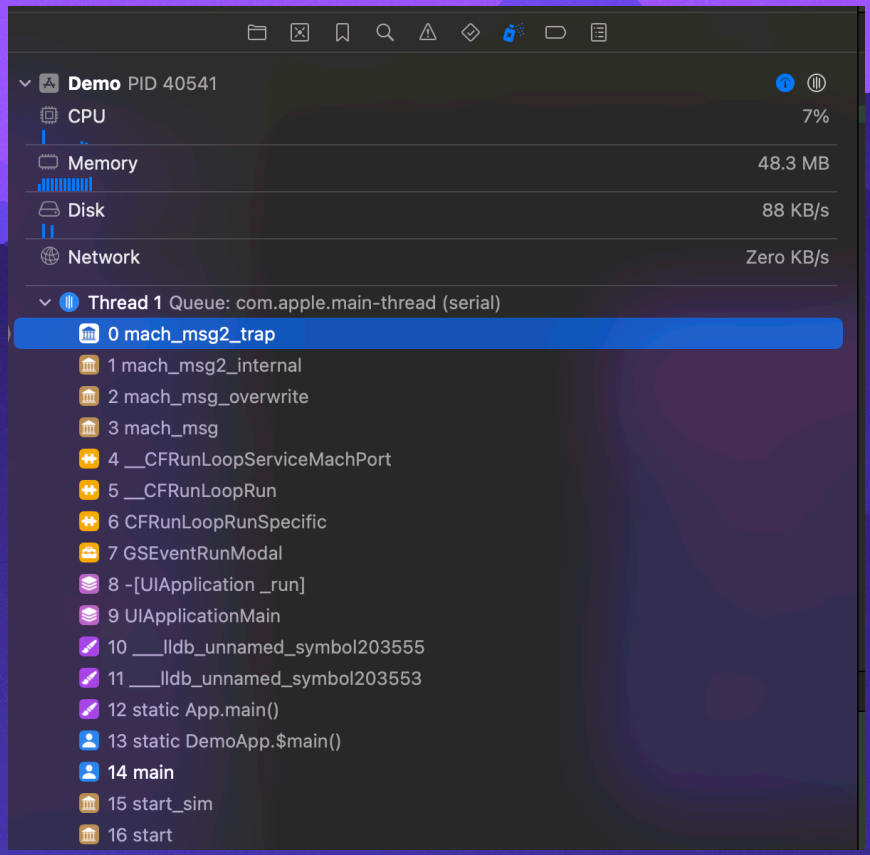
```
@Observable
class Model {
    var status: String = "initial"

    func fetch() {
        status = "fetching"
        // ...
    }
}

struct ContentView: View {
    @Bindable var model = Model()

    var body: some View {
        Text(model.status)
            .onAppear { model.fetch() }
    }
}
```


Run Loop
Main Queue
Main Actor
SwiftUI
UIKit
CFRunLoop

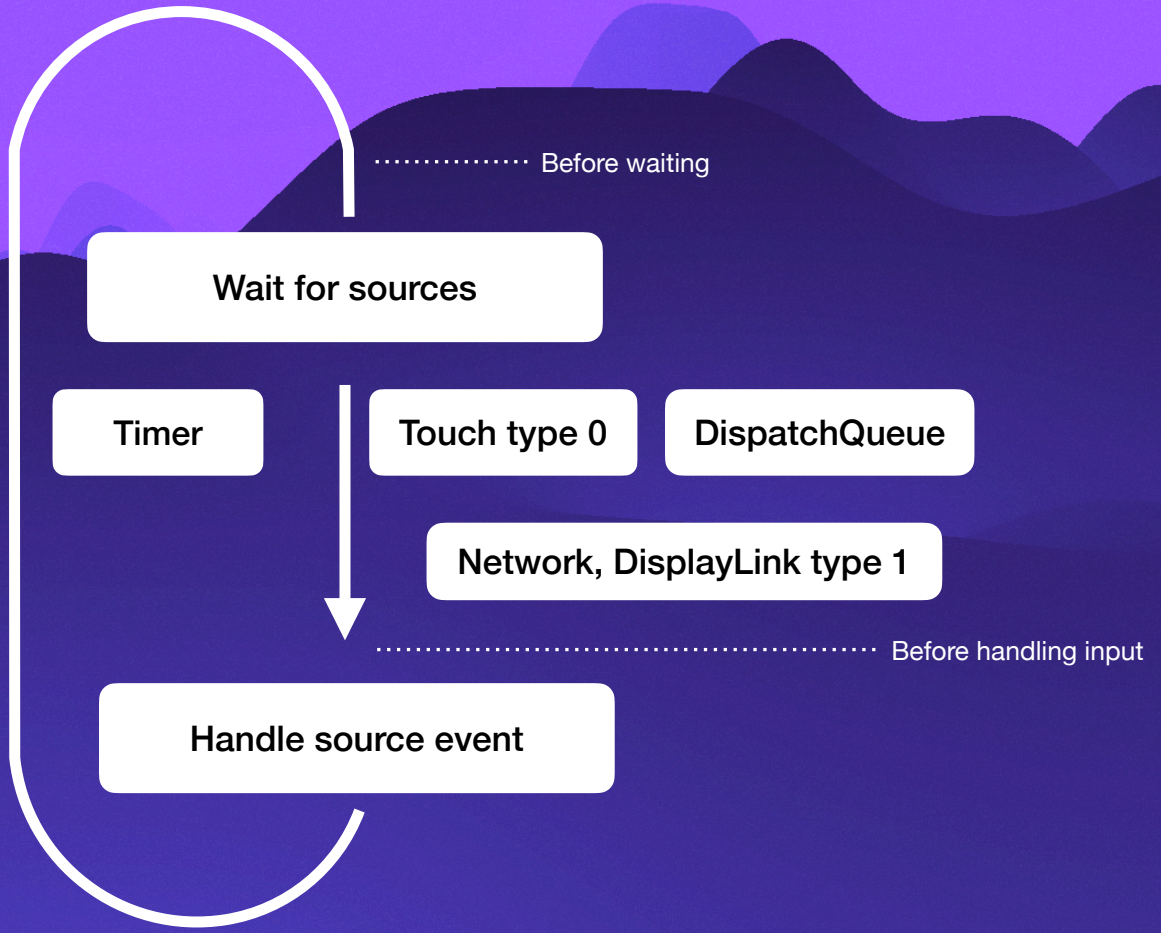
Run Loop


Before waiting
Wait for sources
Timer
Touch type 0
DispatchQueue
Network, DisplayLink type 1
Before handling input
Handle source event

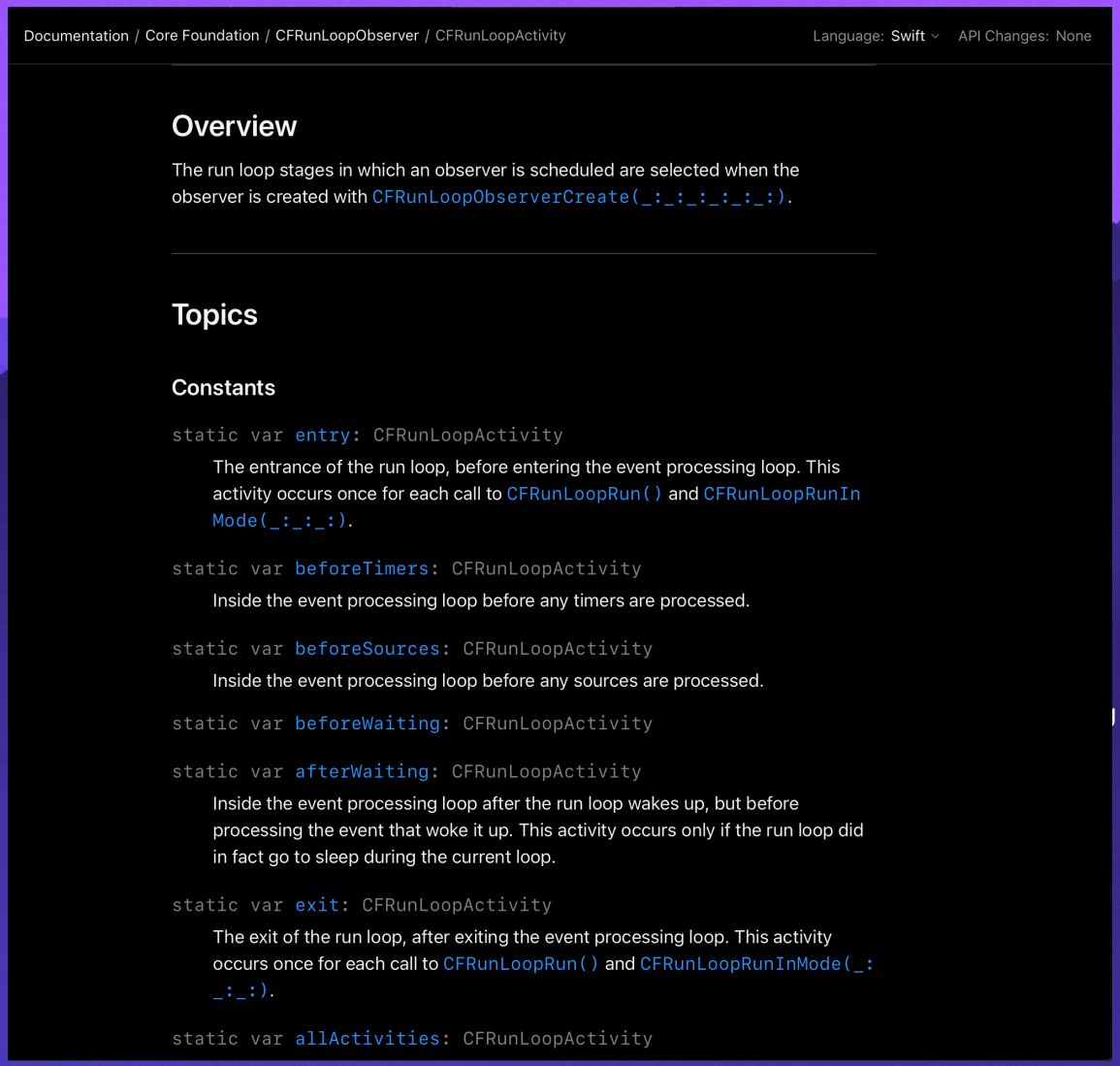
Run Loop


Demo PID 40541
CPU 7%
Memory 48.3 MB
Disk 88 KB/s
Network Zero KB/s
Thread 1 Queue: com.apple.main-thread (serial)
0 mach_msg2_trap
1 mach_msg2_internal
2 mach_msg_overwrite
3 mach_msg
4 __CFRunLoopServiceMachPort
5 __CFRunLoopRun
6 CFRunLoopRunSpecific
7 GSEventRunModal
8 -[UIApplication _run]
9 UIApplicationMain
10 ___lldb_unnamed_symbol203555
11 ___lldb_unnamed_symbol203553
12 static App.main()
13 static DemoApp.$main()
14 main
15 start_sim
16 start

Run Loop


Before waiting
Wait for sources
Timer
Touch type 0
DispatchQueue
Network, DisplayLink type 1
Before handling input
Handle source event

Run Loop Observers



Documentation / Core Foundation / CFRunLoopObserver / CFRunLoopActivity

Language: Swift API Changes: None

Overview

The run loop stages in which an observer is scheduled are selected when the observer is created with `CFRunLoopObserverCreate(_:_:_:_:_:_:)`.

Topics

Constants

`static var entry: CFRunLoopActivity`

The entrance of the run loop, before entering the event processing loop. This activity occurs once for each call to `CFRunLoopRun()` and `CFRunLoopRunInMode(_:_:_:)`.

`static var beforeTimers: CFRunLoopActivity`

Inside the event processing loop before any timers are processed.

`static var beforeSources: CFRunLoopActivity`

Inside the event processing loop before any sources are processed.

`static var beforeWaiting: CFRunLoopActivity`

`static var afterWaiting: CFRunLoopActivity`

Inside the event processing loop after the run loop wakes up, but before processing the event that woke it up. This activity occurs only if the run loop did in fact go to sleep during the current loop.

`static var exit: CFRunLoopActivity`

The exit of the run loop, after exiting the event processing loop. This activity occurs once for each call to `CFRunLoopRun()` and `CFRunLoopRunInMode(_:_:_:)`.

`static var allActivities: CFRunLoopActivity`

input

Run Loop

__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__

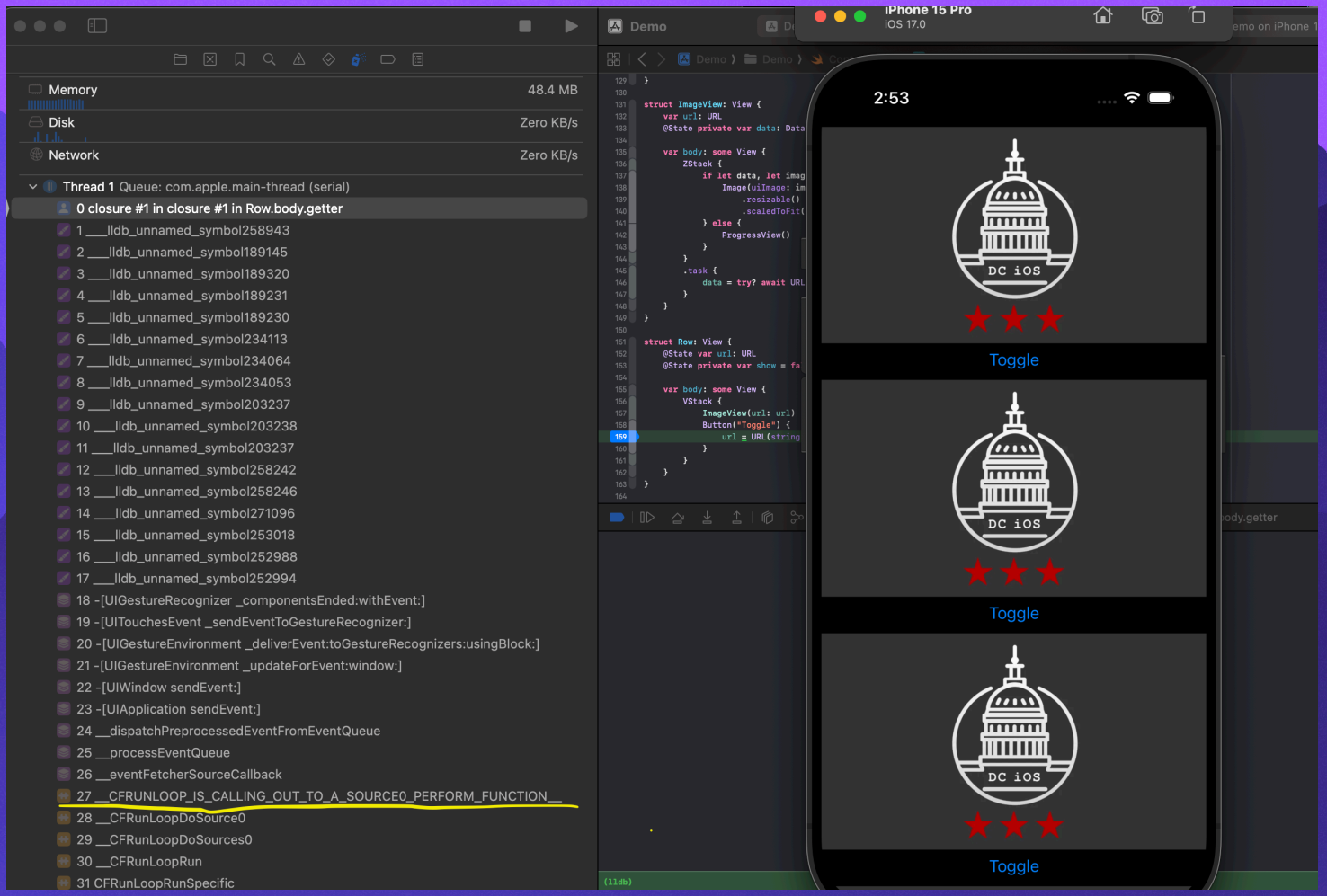
__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__

__CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__

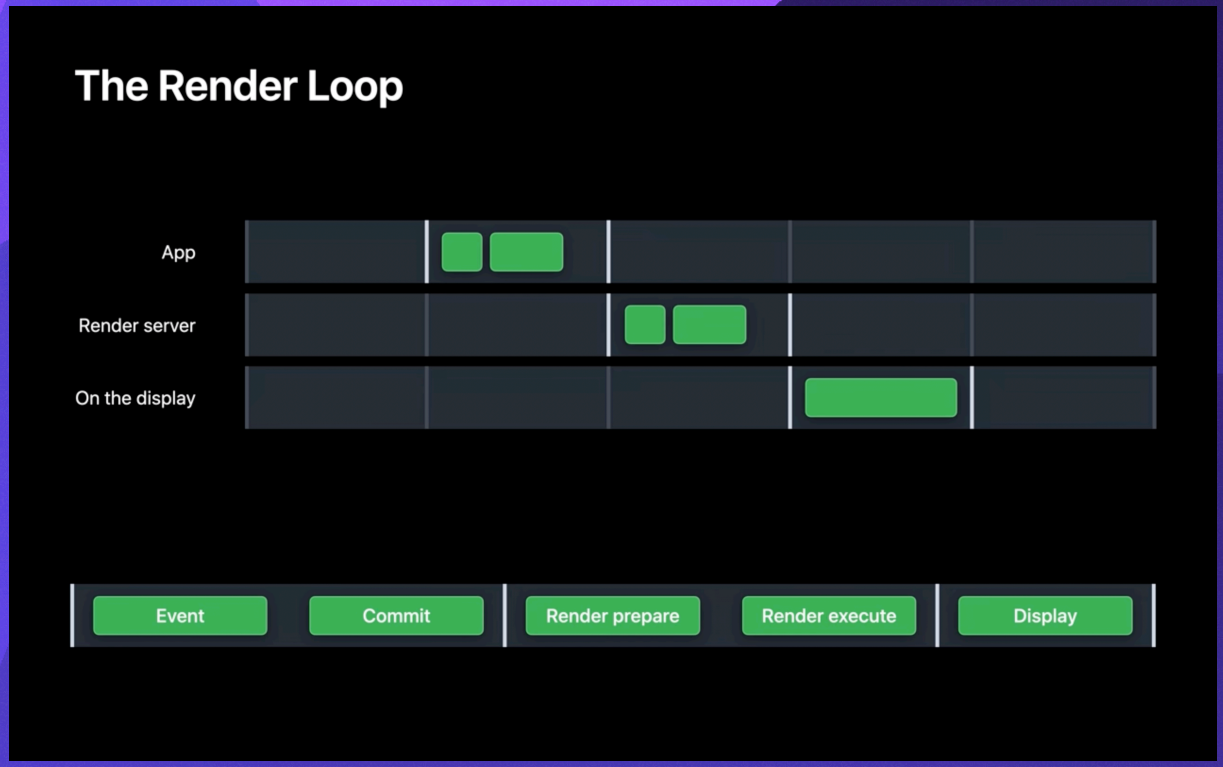
__CFRUNLOOP_IS_CALLING_OUT_TO_A_TIMER_CALLBACK_FUNCTION__

__CFRUNLOOP_IS_CALLING_OUT_TO_AN_OBSERVER_CALLBACK_FUNCTION__

Run Loop


Memory 48.4 MB
Disk Zero KB/s
Network Zero KB/s
Thread 1 Queue: com.apple.main-thread (serial)
0 closure #1 in closure #1 in Row.body.getter
1 ___lldb_unnamed_symbol258943
2 ___lldb_unnamed_symbol189145
3 ___lldb_unnamed_symbol189320
4 ___lldb_unnamed_symbol189231
5 ___lldb_unnamed_symbol189230
6 ___lldb_unnamed_symbol234113
7 ___lldb_unnamed_symbol234064
8 ___lldb_unnamed_symbol234053
9 ___lldb_unnamed_symbol203237
10 ___lldb_unnamed_symbol203238
11 ___lldb_unnamed_symbol203237
12 ___lldb_unnamed_symbol258242
13 ___lldb_unnamed_symbol258246
14 ___lldb_unnamed_symbol271096
15 ___lldb_unnamed_symbol253018
16 ___lldb_unnamed_symbol252988
17 ___lldb_unnamed_symbol252994
18 -[UIGestureRecognizer _componentsEnded:withEvent:]
19 -[UITouchesEvent _sendEventToGestureRecognizer:]
20 -[UIGestureEnvironment _deliverEvent:toGestureRecognizers:usingBlock:]
21 -[UIGestureEnvironment _updateForEvent:window:]
22 -[UIWindow sendEvent:]
23 -[UIApplication sendEvent:]
24 __dispatchPreprocessedEventFromEventQueue
25 __processEventQueue
26 __eventFetcherSourceCallback
27 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__
28 __CFRunLoopDoSource0
29 __CFRunLoopDoSources0
30 __CFRunLoopRun
31 CFRunLoopRunSpecific
Demo
iPhone 15 Pro
iOS 17.0
2:53
DC iOS
Toggle
Toggle
Toggle

The Render Loop

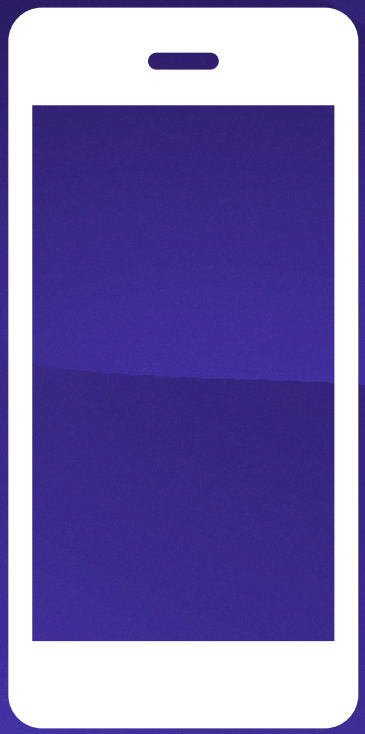


Explore UI animation hitches and the render loop, Apple

Hardware

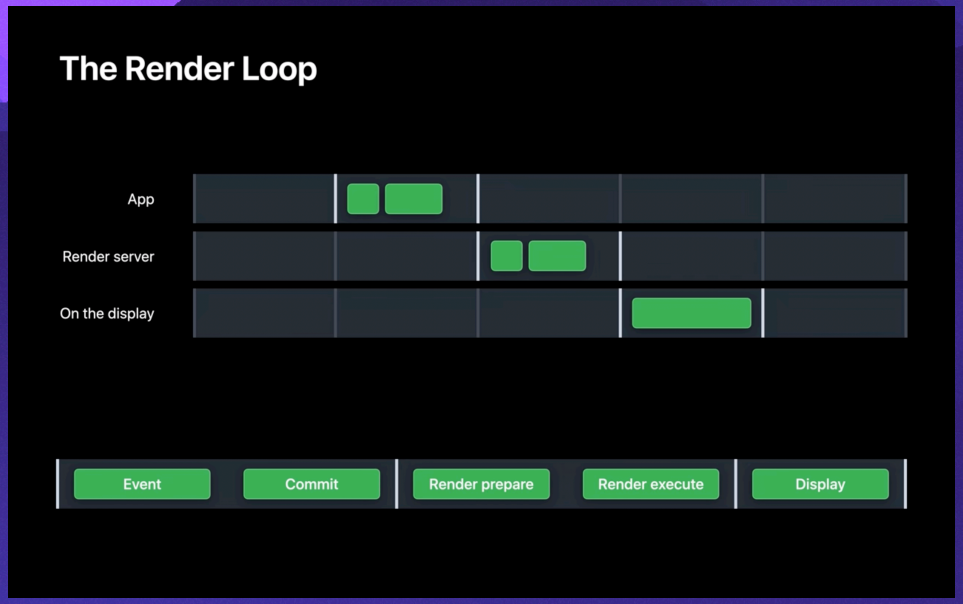
Display
60hz
***dynamic refresh 120hz**
screen refreshes

Input
Touch input may be at a higher rate than display but we do not receive them separately



The Render Loop

Event Phase



The Render Loop

Event Phase

Receive events - touches, networking, keyboard, timers, etc.

```swift
class UIKitView: UIView {
    private var isOn = false
    private var height: NSLayoutConstraint!

    @objc
    private func buttonTapped(_ sender: UIButton) {
        isOn.toggle()
        backgroundColor = isOn ? .blue : .green // setNeedsDisplay()
        height.constant = isOn ? 50 : 100 // setNeedsLayout()
    }
}

struct SwiftUIView: View {
    @State private var isOn = false

    var body: some View {
        Button("Toggle") {
            isOn.toggle() // Update Attribute Graph
        }
        .background(isOn ? .blue : .green)
        .frame(height: isOn ? 50 : 100)
    }
}
```

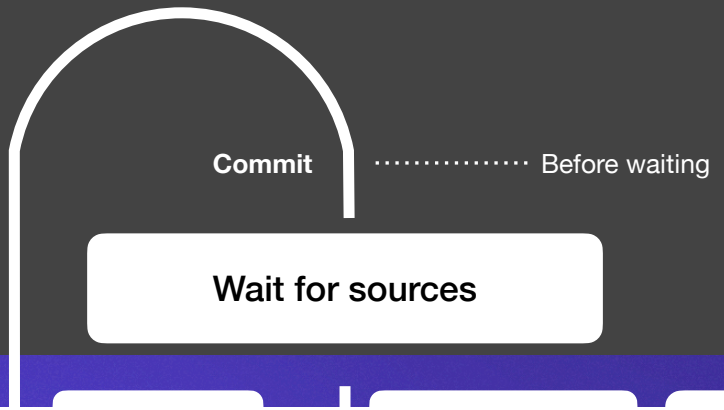
CATransaction

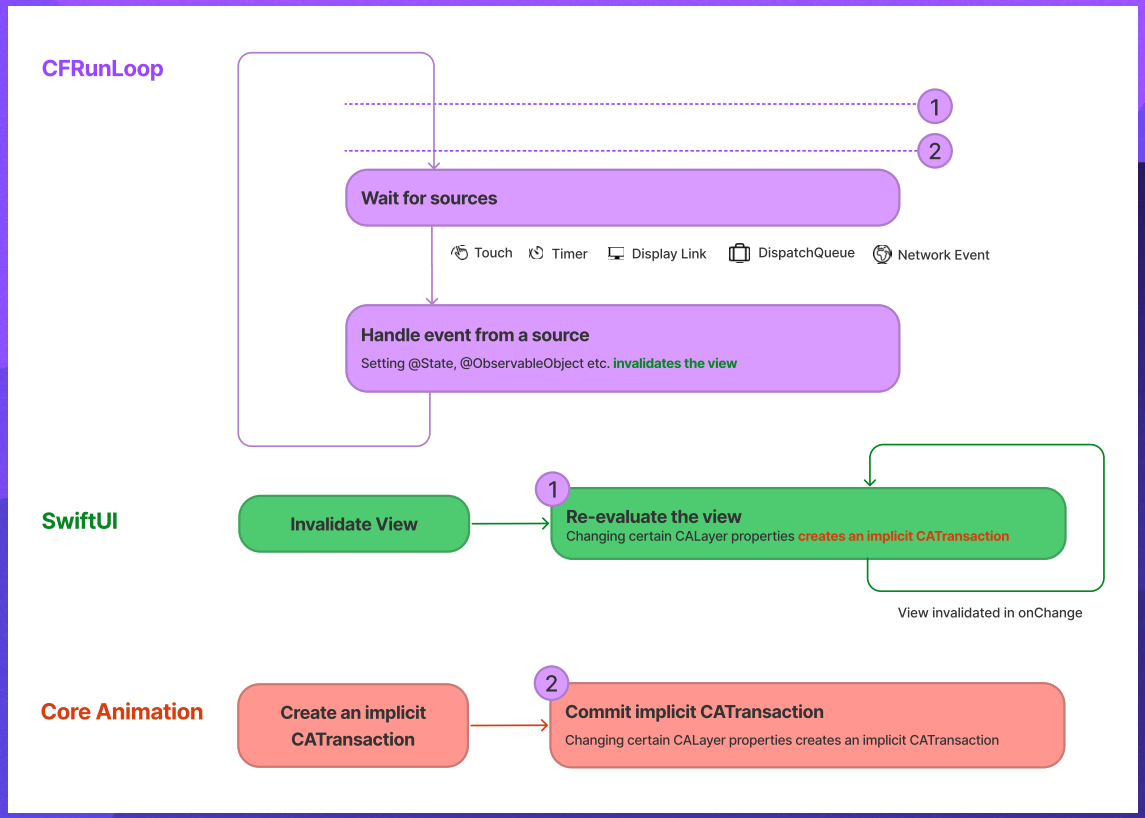
Core Animation talks to the render server.

Implicit transactions started automatically

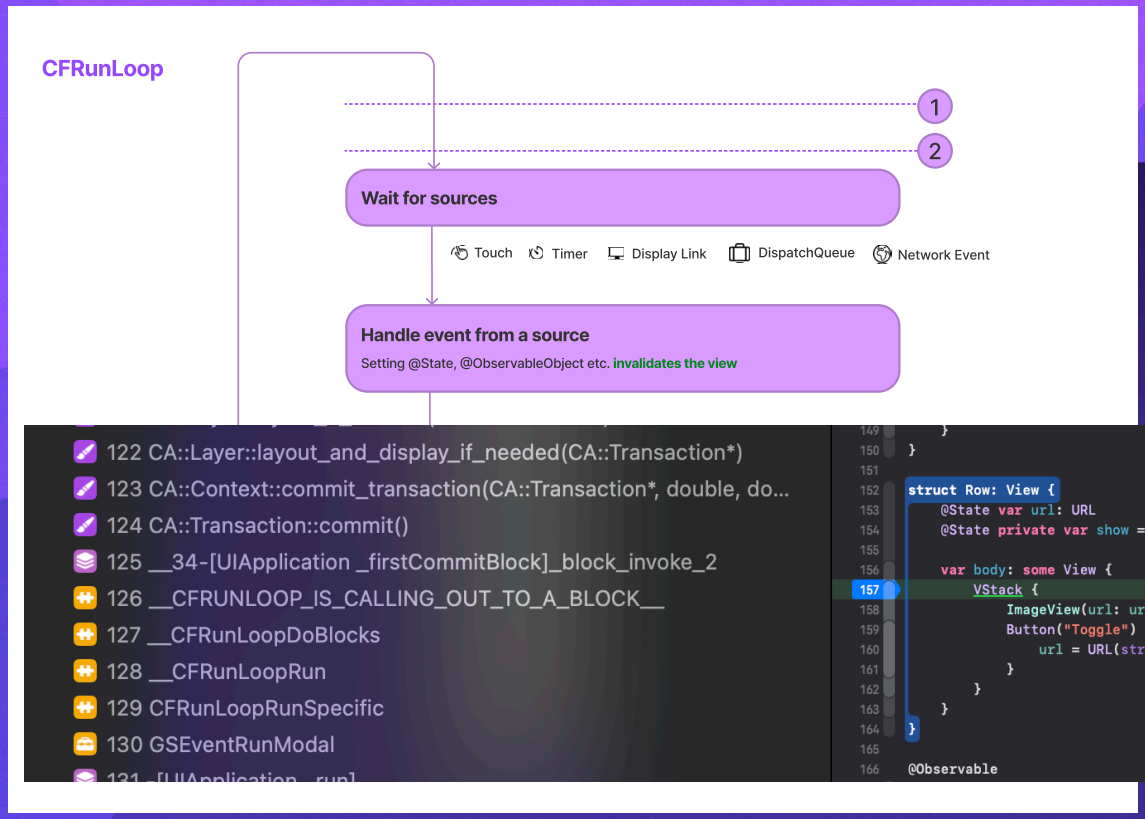
```
class UIKitView: UIView {
    private var isOn = false
    private var height: NSLayoutConstraint!

    @objc
    private func buttonTapped(_ sender: UIButton) {
        isOn.toggle()
        backgroundColor = isOn ? .blue : .green // setNeedsDisplay()
        sleep(1) //
        height.constant = isOn ? 50 : 100 // setNeedsLayout()
    }
}
```

What is the rendered background color?



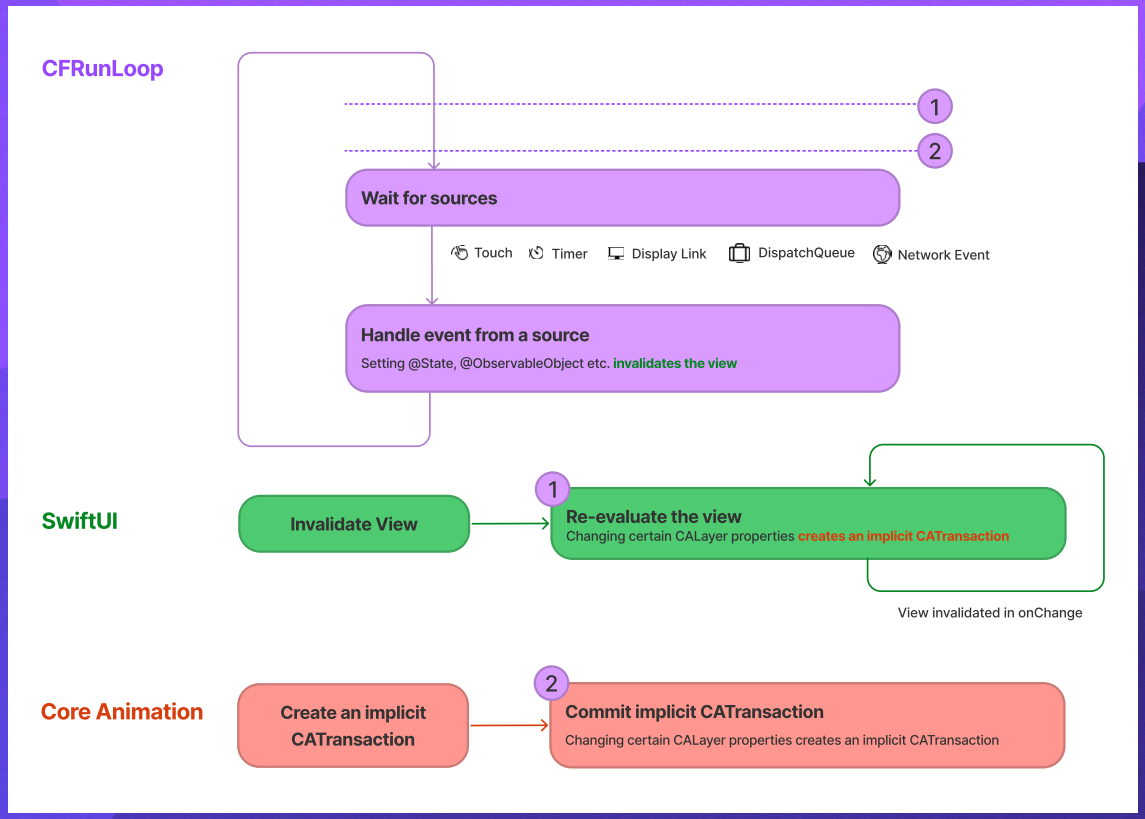


The SwiftUI render loop, Rens Breur



The SwiftUI render loop, Rens Breur

onChange(of: _) action tried to update multiple times per frame



Evaluate body of changed views

```
onChange()
onPreferenceChange
onAppear()
```

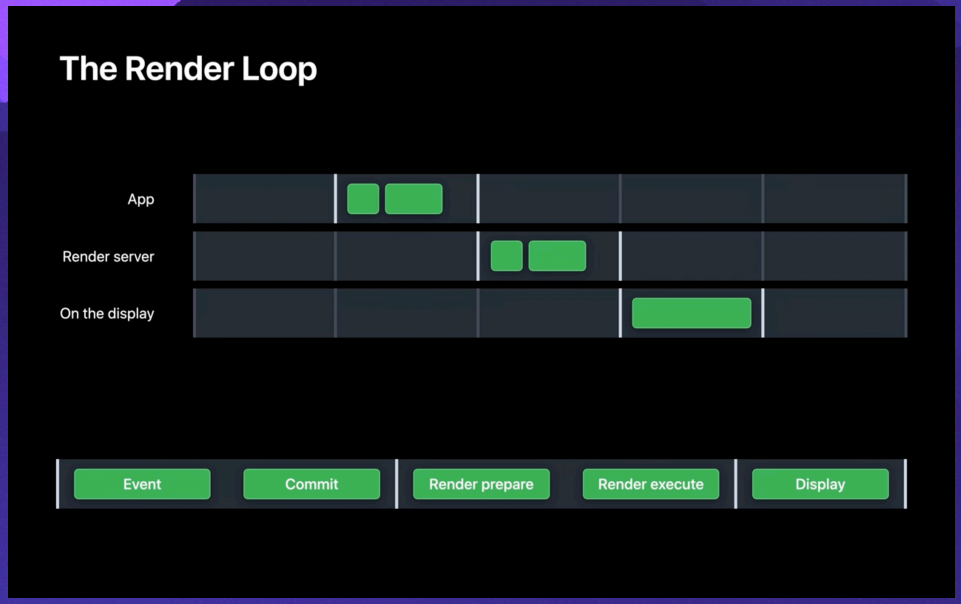
The SwiftUI render loop, Rens Breur

onChange(of: _) action tried to update multiple times per frame

The Render Loop

Commit Phase

Layout and drawing



The Render Loop

Commit Phase

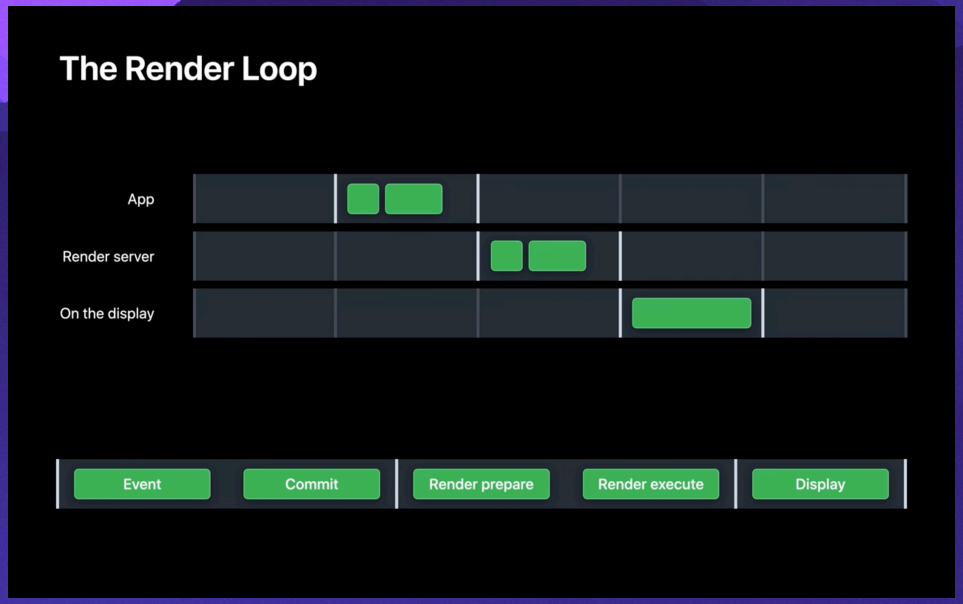
Layout and drawing

```
private struct BasicVStack: Layout {
     func sizeThatFits(
          proposal: ProposedViewSize,
          subviews: Subviews,
          cache: inout ()
     ) -> CGSize {
          subviews.reduce(CGSize.zero) { result, subview in
              let size = subview.sizeThatFits(.unspecified)
              return CGSize(
                  width: max(result.width, size.width),
                  height: result.height + size.height)
          }
     }

    func placeSubviews(
        in bounds: CGRect,
        proposal: ProposedViewSize,
        subviews: Subviews,
        cache: inout Cache
    ) {
        // for illustration only
        subviews.forEach { $0.place(at: ..., proposal: proposal)
    }
}

class UIKitView: UIView {

    override func layoutSubviews() {
        super.layoutSubviews()
        // …

    }

    override func draw(_ rect: CGRect) {
        super.draw(rect)
        // ...
    }
}
```

The Render Loop

Render Prepare/Execute

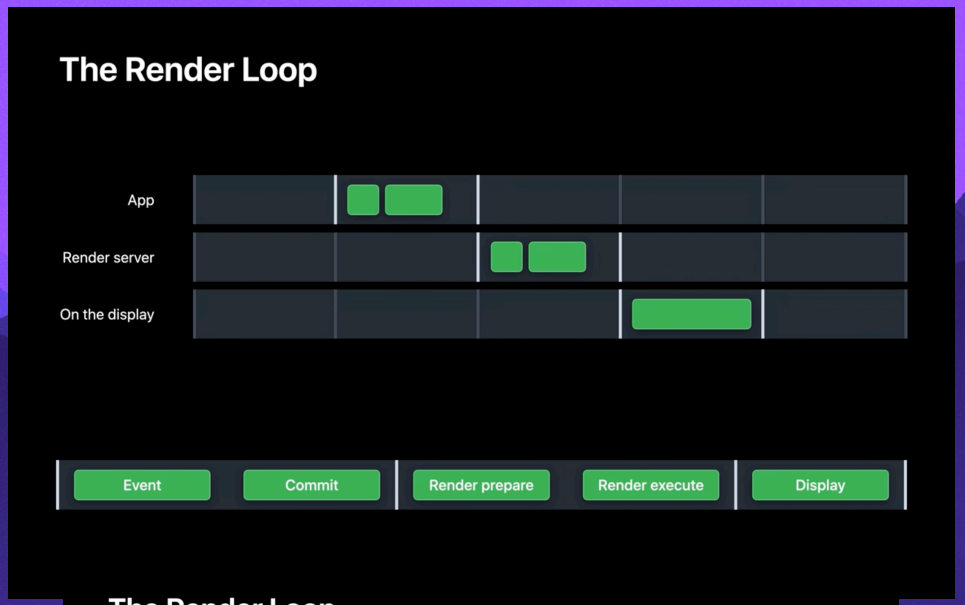
Layer tree sent to render server

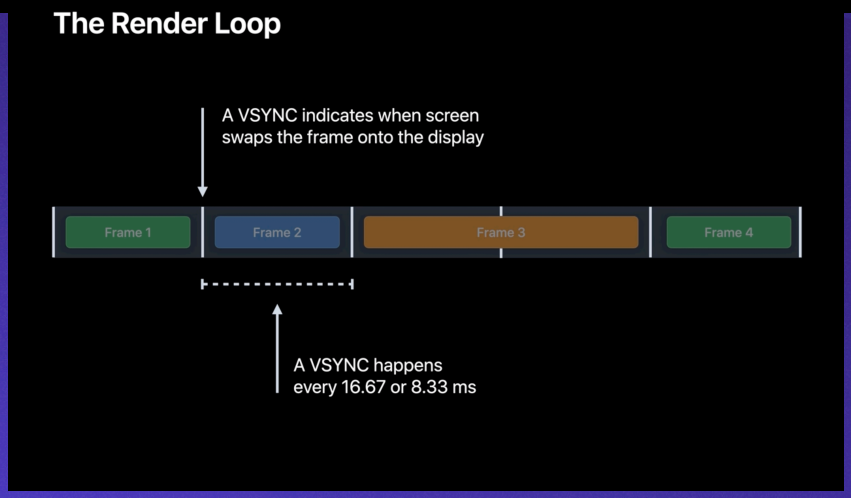


The Render Loop

Display

GPU executes and renders final output to be displayed on next VSYNC



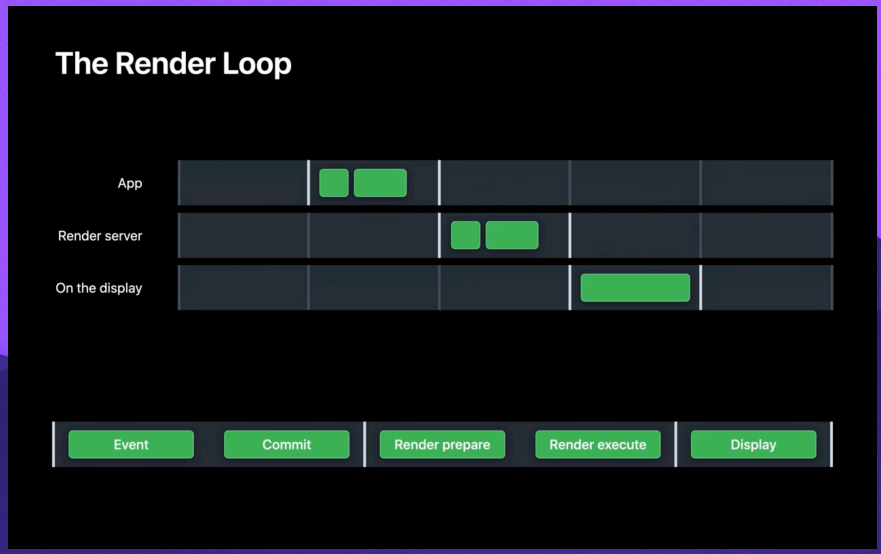


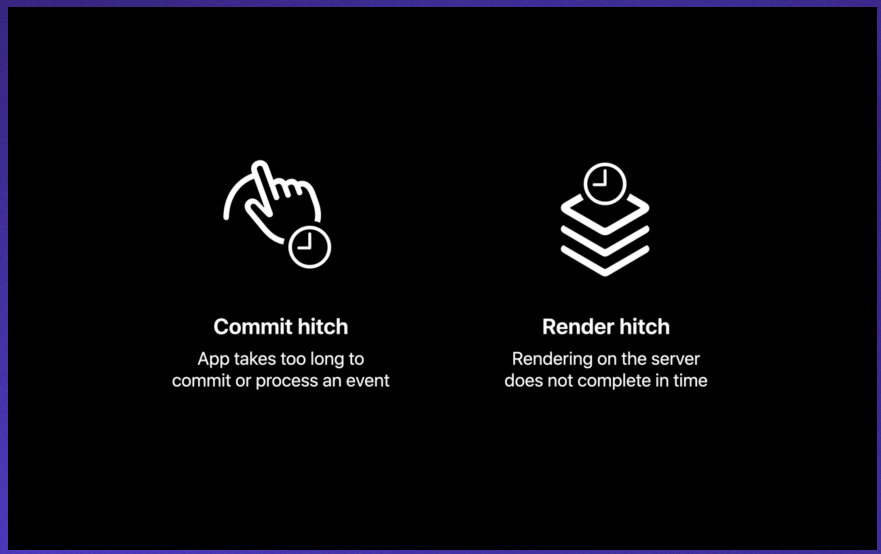
The Render Loop

Hitches

Commit - too long to process events or commit

Render - too long on the render server






View Identity and Lifetime


Identity Lifetime Dependencies


Identity


Identity
Implicit
Explicit

Identity

Implicit

```
SwiftUI._ConditionalContent<SwiftUI.Text, SwiftUI.Text>
```

```
var body: some View {
    if expand {
        expandedView
    } else {
        collapsedView
    }
}
```


Identity
Explicit
var body: some View {
VStack { ... }
.id(url)
}
ForEach(records, id: \.id) {
Text($0.content)
}


Lifetime

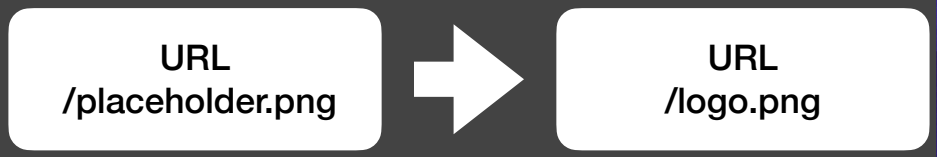

Lifetime
Lifetime of a View is the duration of its identity

Lifetime

```
struct ImageView: View {
    var url: URL
    @State private var data: Data?

    var body: some View {
        ZStack {
            if let data, let image = UIImage(data: data) {
                Image(uiImage: image)
            } else {
                ProgressView()
            }
        }
        .task {
            data = try? await URLSession.shared.data(from: url).0
        }
    }
}
```

Lifetime

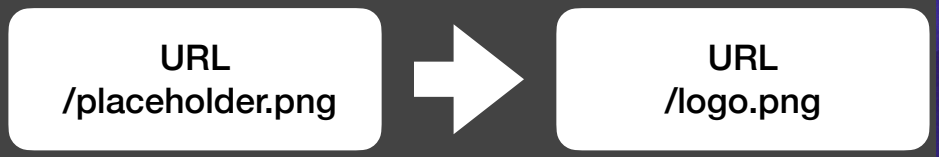
```
struct ImageView: View {
    var url: URL
    @State private var data: Data?

    var body: some View {
        ZStack {
            if let data, let image = UIImage(data: data) {
                Image(uiImage: image)
            } else {
                ProgressView()
            }
        }
        .task {
            data = try? await URLSession.shared.data(from: url).0
        }
    }
}
```



What happens here when URL changes? Does the task run again?

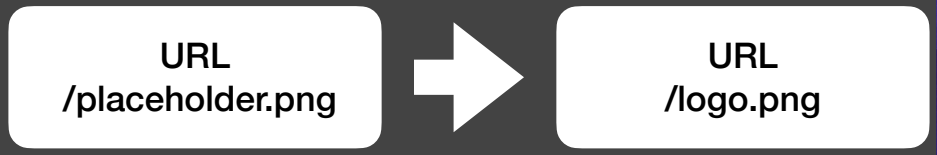
Lifetime

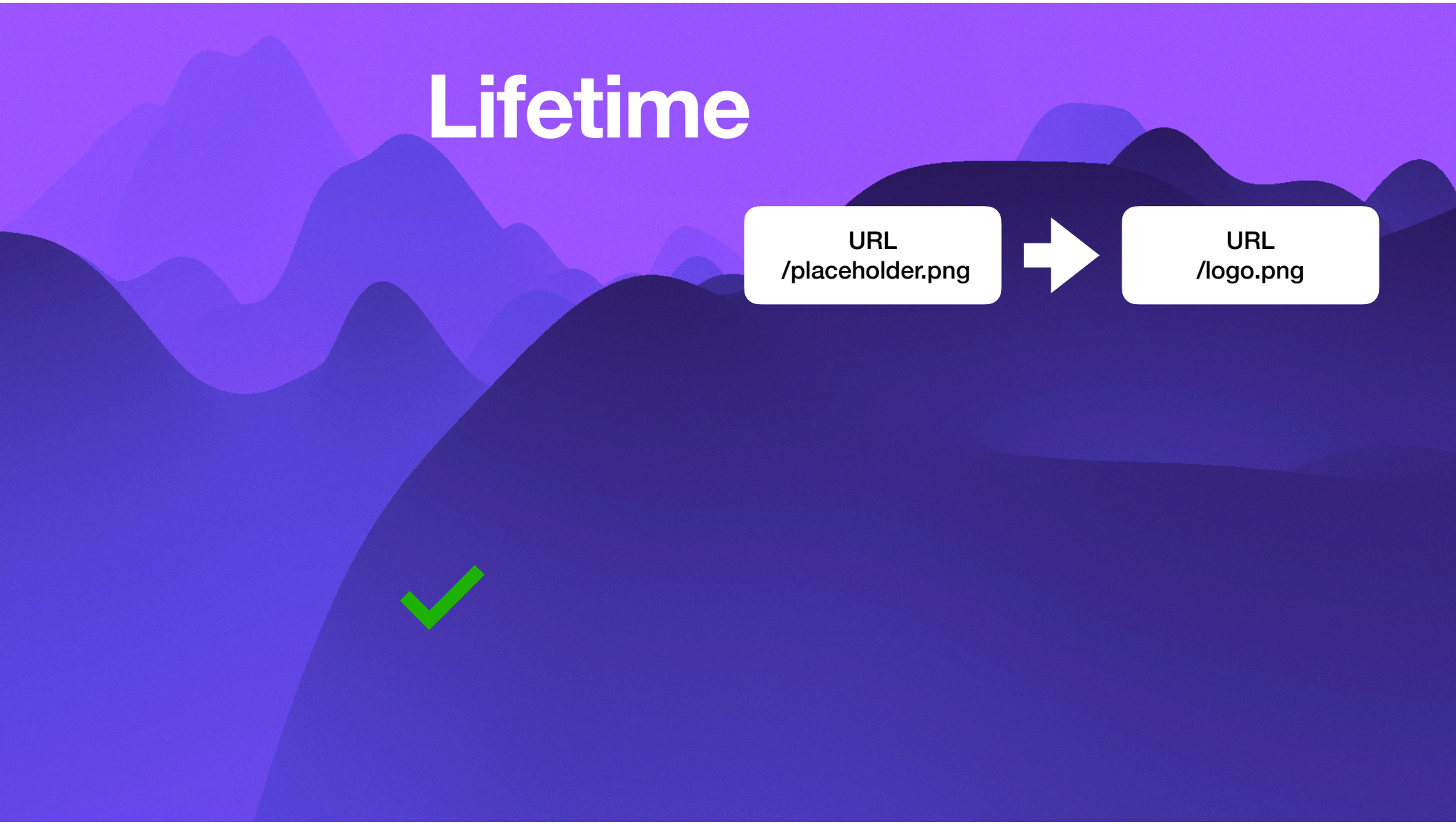
```
struct ImageView: View {
    var url: URL
    @State private var data: Data?

    var body: some View {
        ZStack {
            if let data, let image = UIImage(data: data) {
                Image(uiImage: image)
            } else {
                ProgressView()
            }
        }
        .task {
            data = try? await URLSession.shared.data(from: url).0
        }
    }
}
```



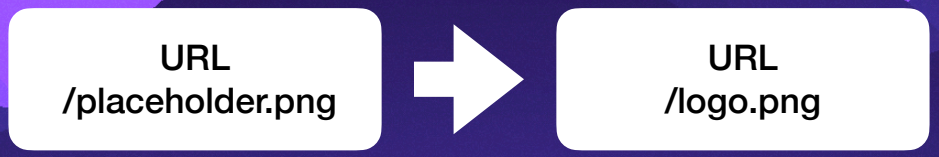
Lifetime

```swift
struct ImageView: View {
    var url: URL
    @State private var data: Data?

    var body: some View {
        ZStack {
            if let data, let image = UIImage(data: data) {
                Image(uiImage: image)
            } else {
                ProgressView()
            }
        }
        .task {
            data = try? await URLSession.shared.data(from: url).0
        }
        .id(url)
    }
}
```

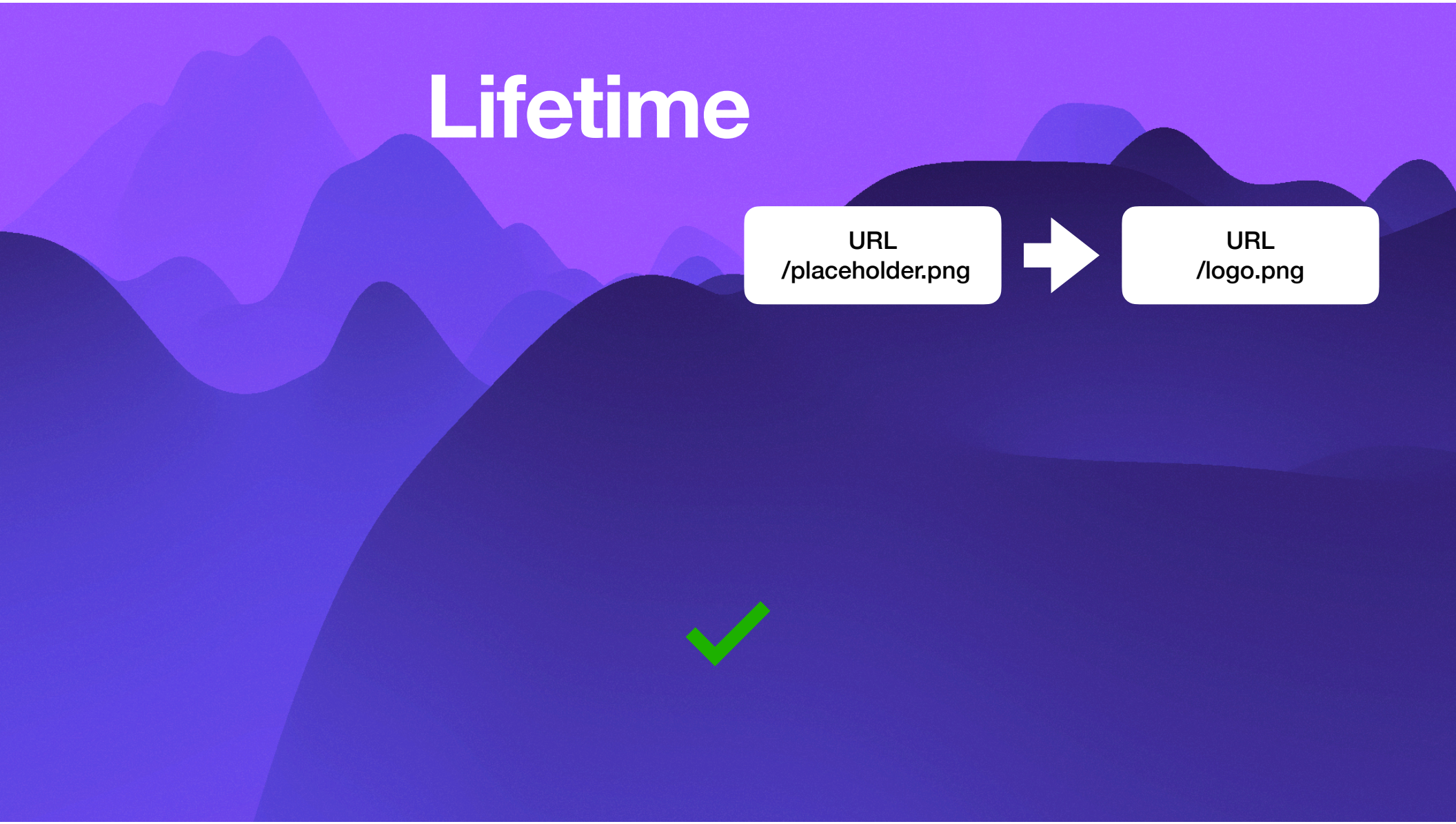



Lifetime
URL
/placeholder.png
URL
/logo.png

Lifetime



Can we do better?


Lifetime
URL
/placeholder.png
URL
/logo.png

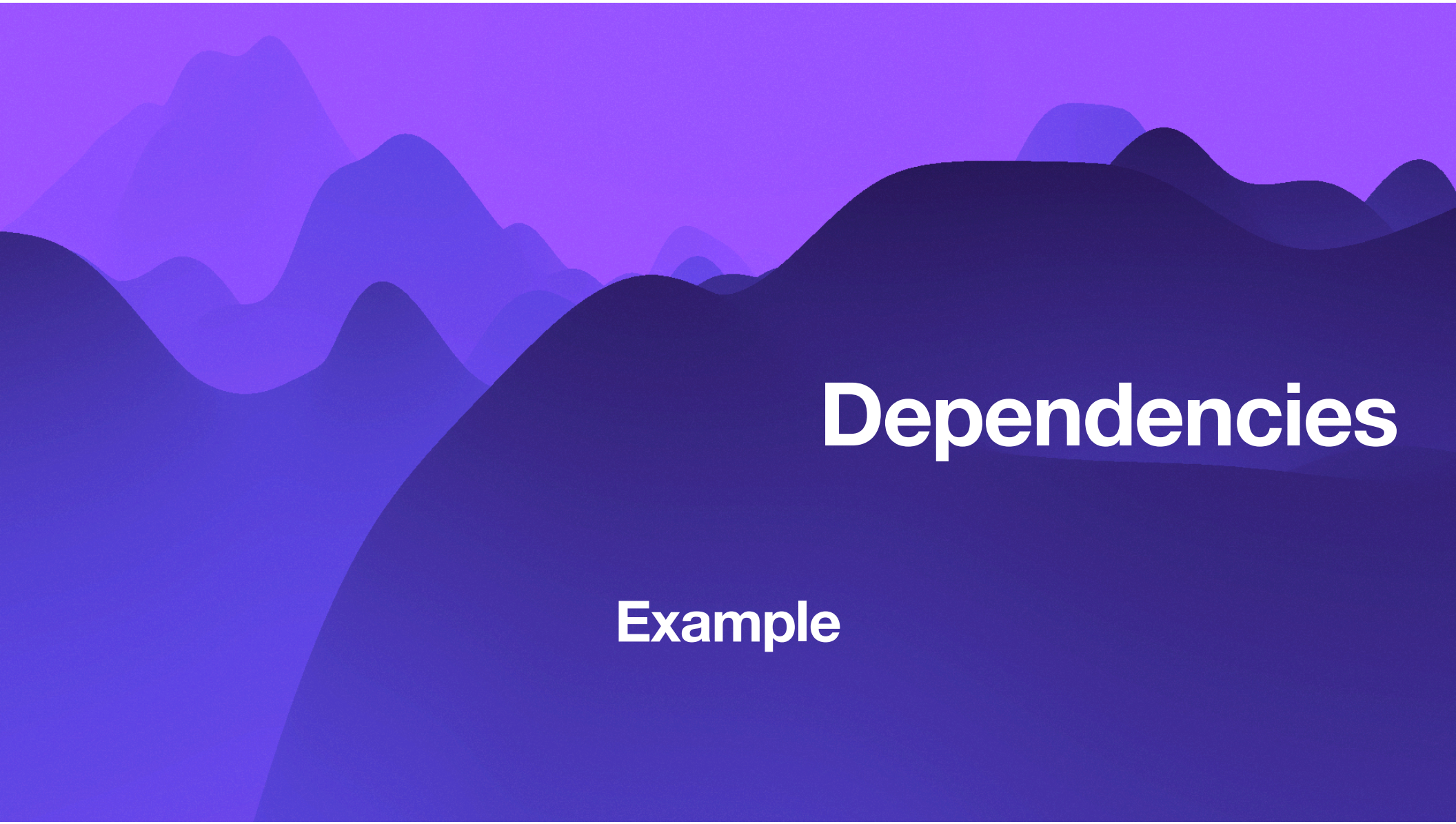

Dependencies

Dependencies

```
struct ContentView: View {

    @State var record: Record = Record() // Dependency
    var condition = false // Dependency
```

SwiftUI tracks dependencies and calls for new view bodies when dependencies change

Dependencies

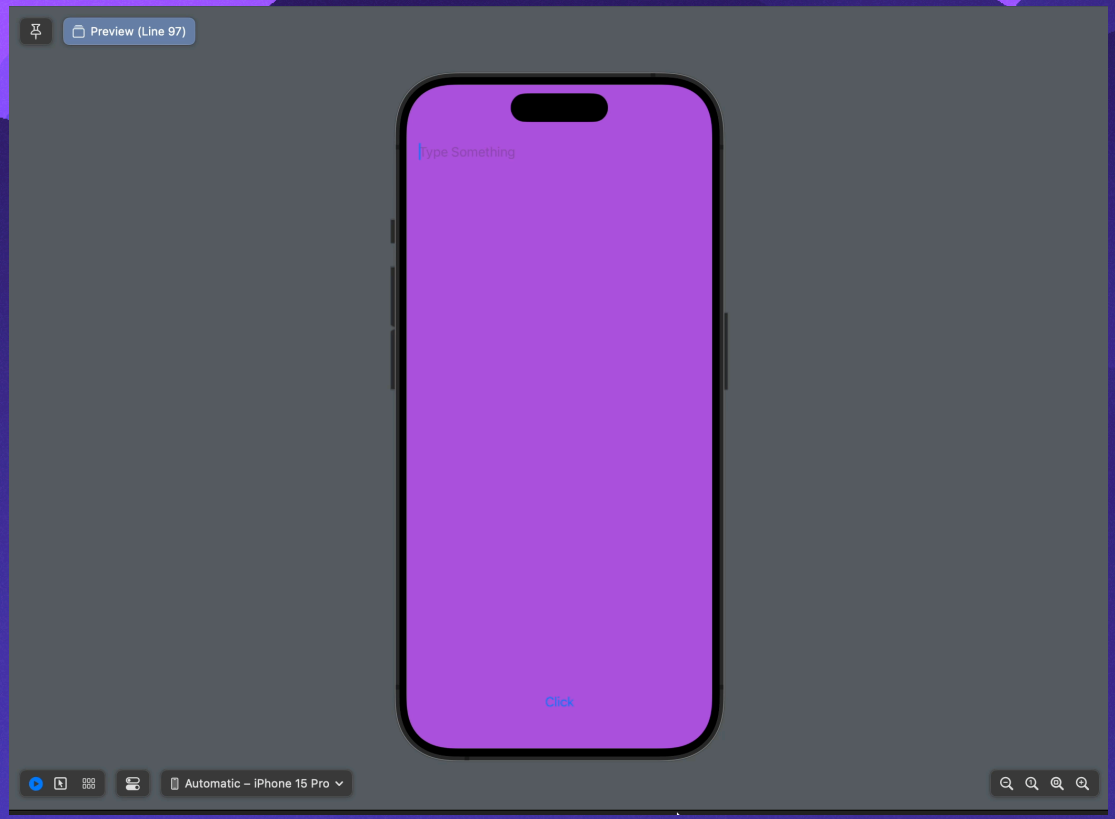

Dependencies
Example


Dependencies


Dependencies

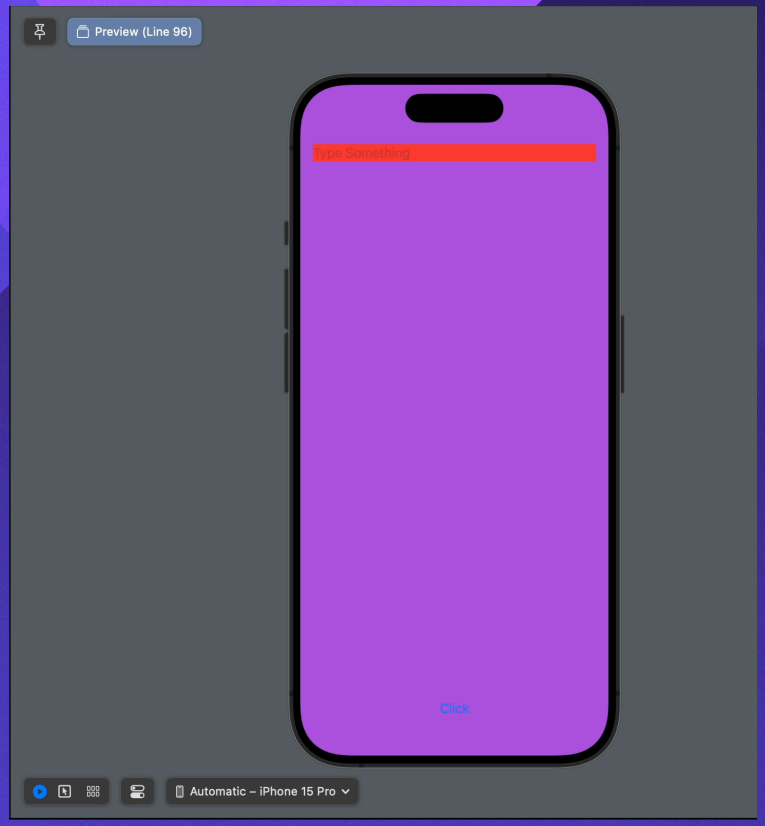

Dependencies

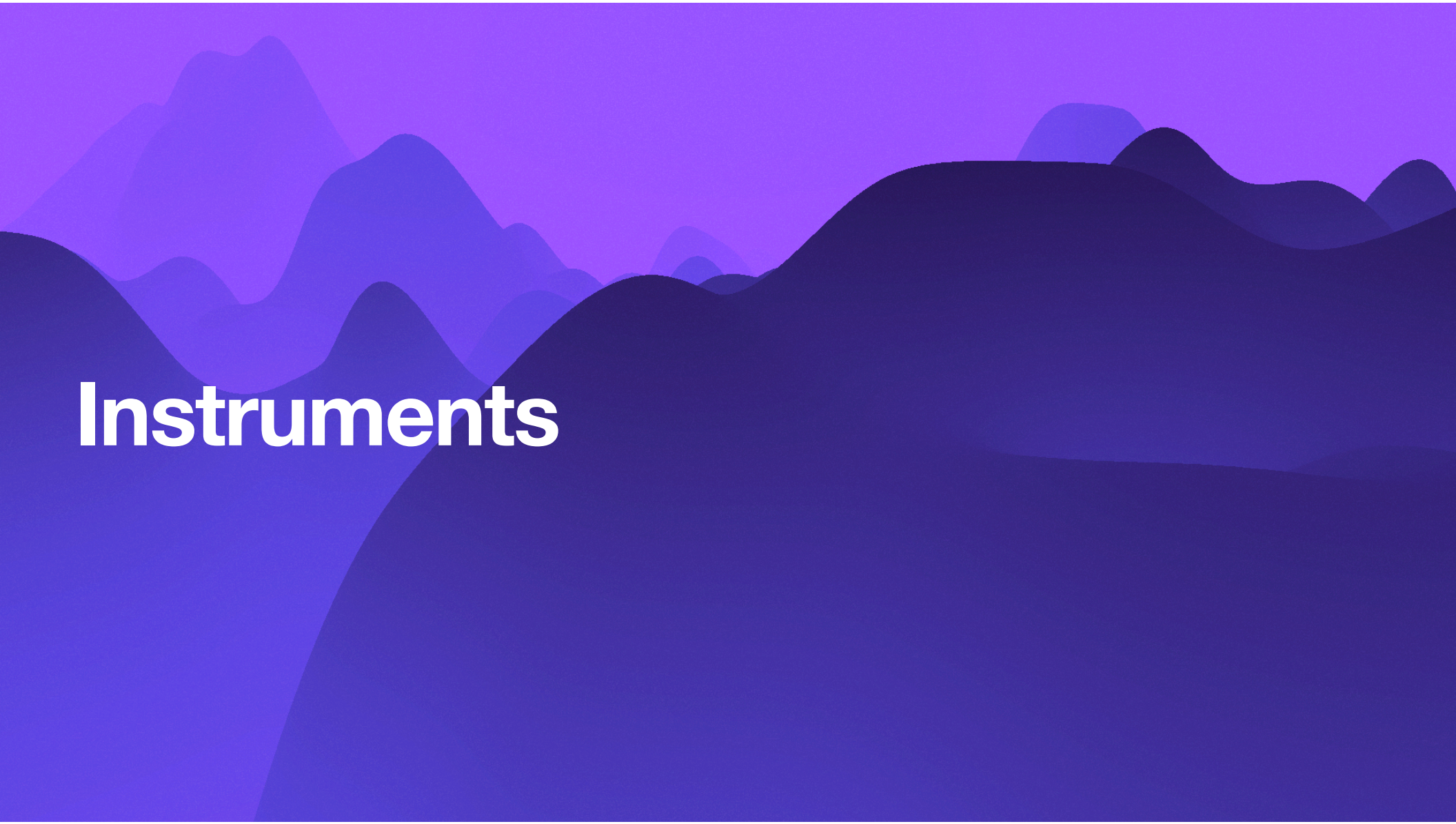
Dependencies




Dependencies

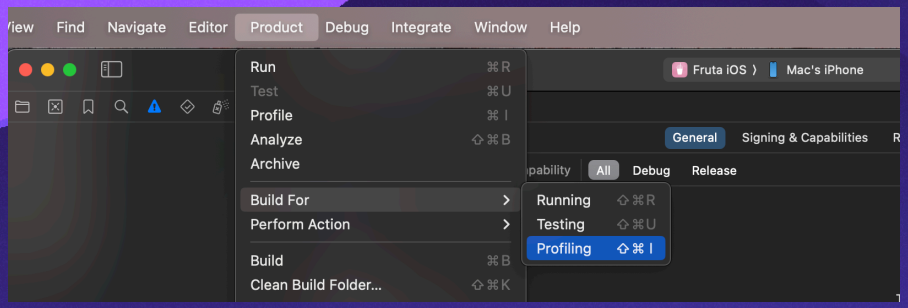
Dependencies

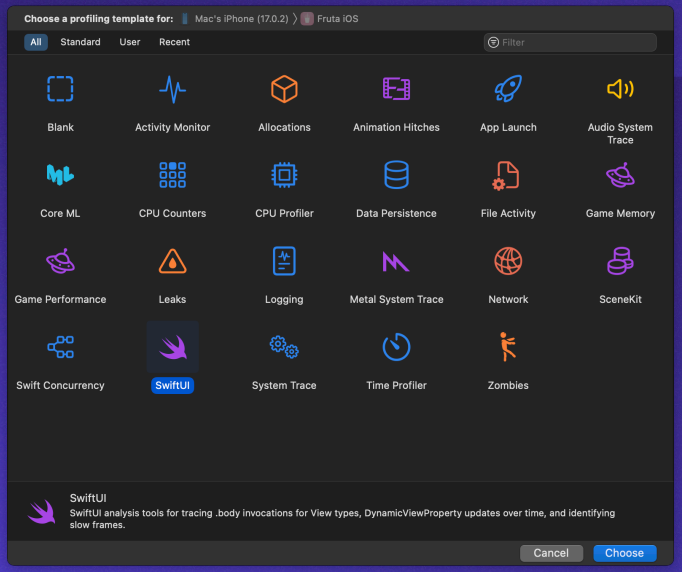


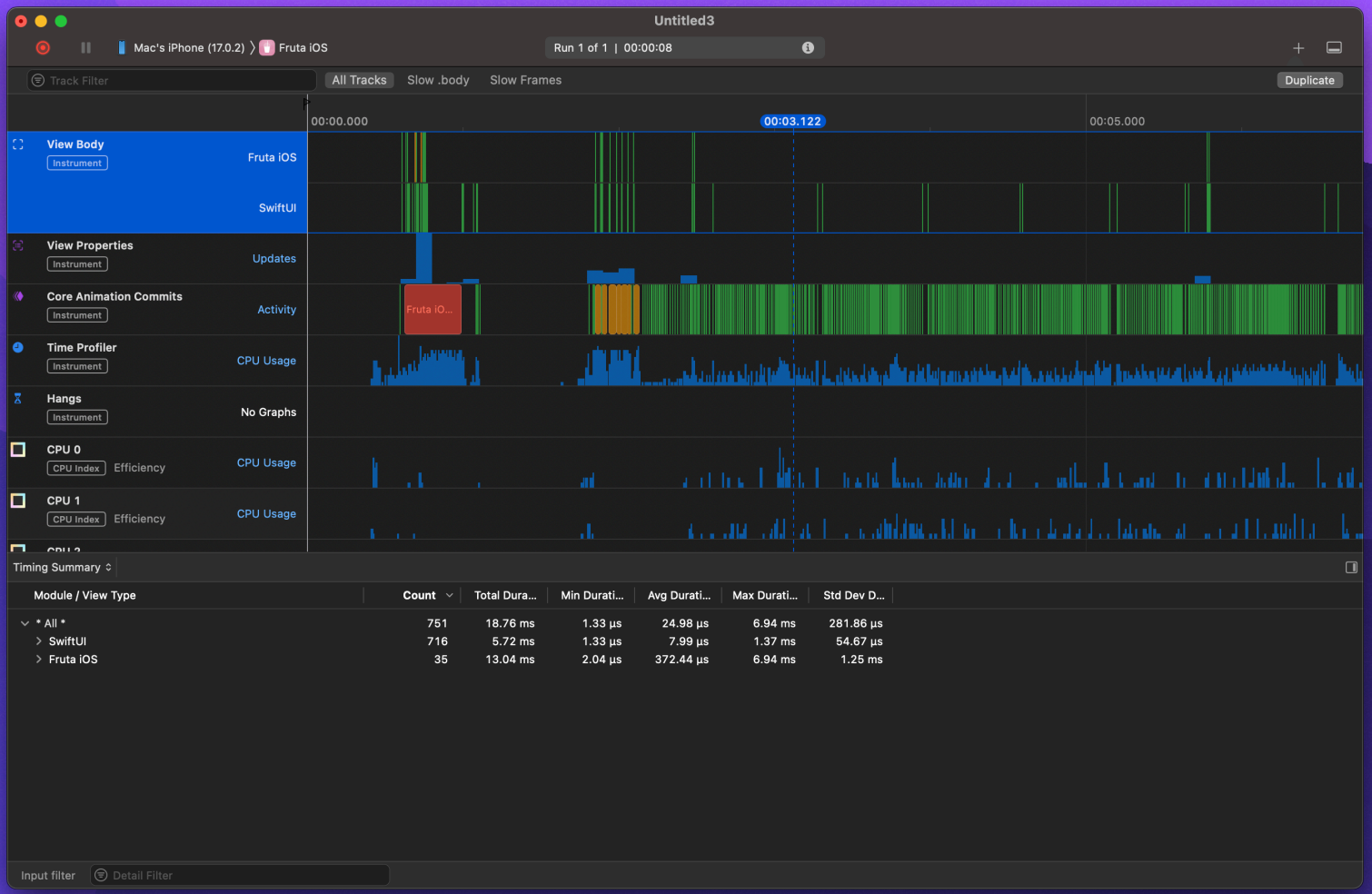

Instruments

How To Use The SwiftUI Profiling Template

- Build your app for profiling
- Profile on physical device






Untitled3
Mac's iPhone (17.0.2)
Fruta iOS
Run 1 of 1 | 00:00:08
Track Filter
All Tracks
Slow .body
Slow Frames
Duplicate
00:00.000
00:03.122
00:05.000
View Body
Instrument
Fruta iOS
SwiftUI
View Properties
Instrument
Updates
Core Animation Commits
Instrument
Activity
Fruta iO...
Time Profiler
Instrument
CPU Usage
Hangs
Instrument
No Graphs
CPU 0
CPU Index
Efficiency
CPU Usage
CPU 1
CPU Index
Efficiency
CPU Usage
Timing Summary
Module / View Type
Count
Total Dura...
Min Durati...
Avg Durati...
Max Durati...
Std Dev D...
* All *
751
18.76 ms
1.33 µs
24.98 µs
6.94 ms
281.86 µs
SwiftUI
716
5.72 ms
1.33 µs
7.99 µs
1.37 ms
54.67 µs
Fruta iOS
35
13.04 ms
2.04 µs
372.44 µs
6.94 ms
1.25 ms
Input filter
Detail Filter

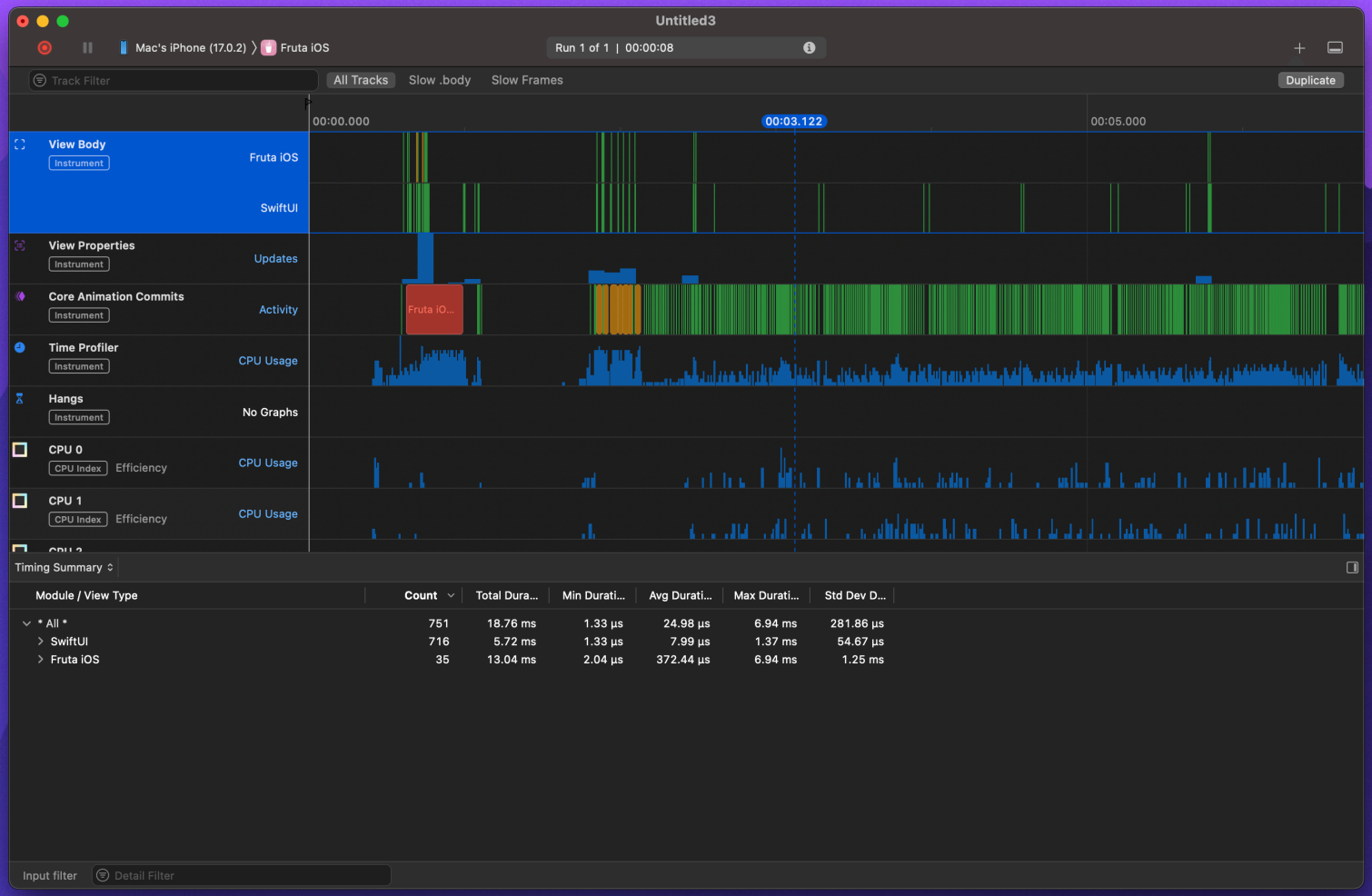
Body

Properties

Commits

Time Profiler

Hangs



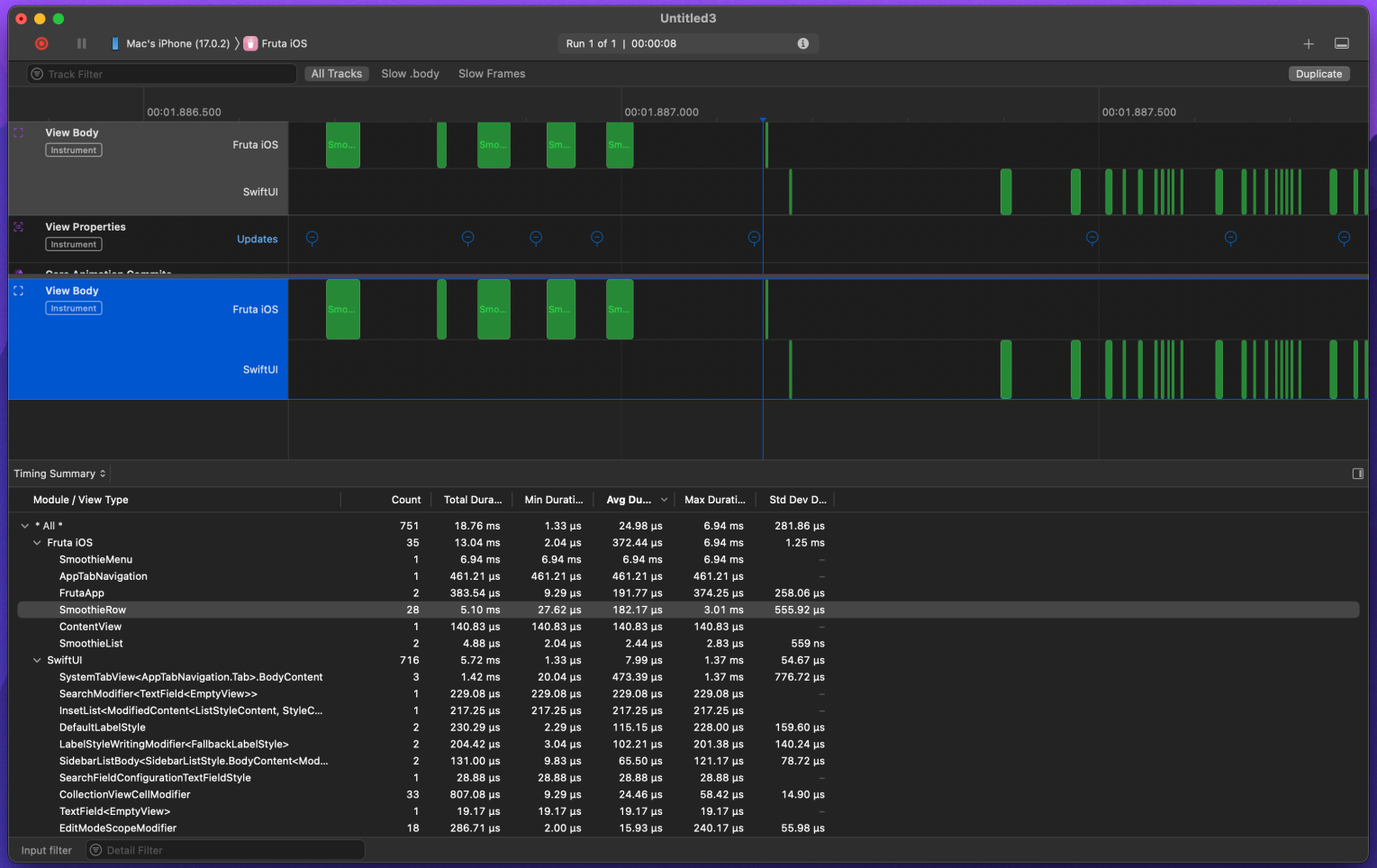
Body

Properties

Commits

Time Profiler

Hangs



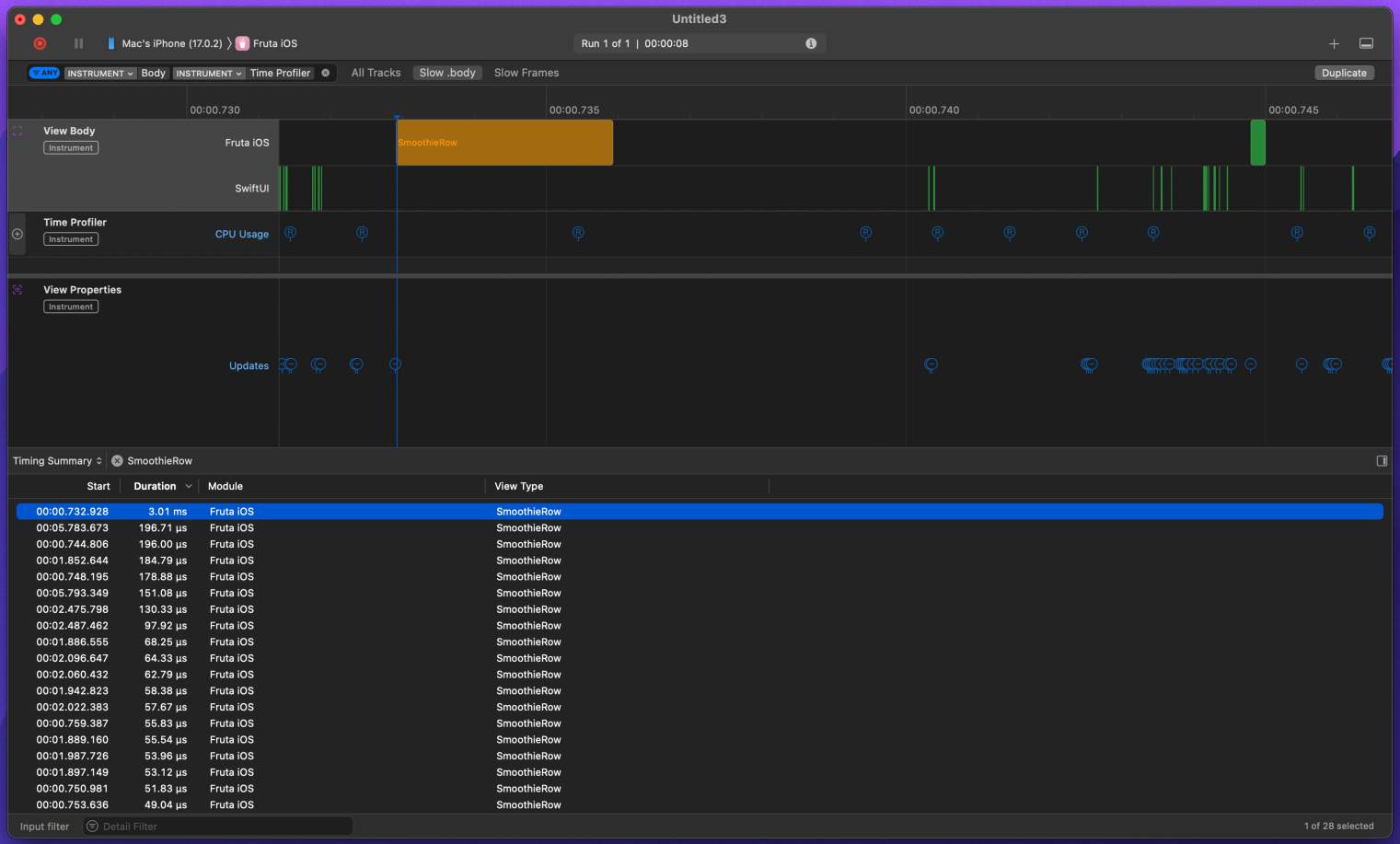
Body

Properties

Commits

Time Profiler

Hangs



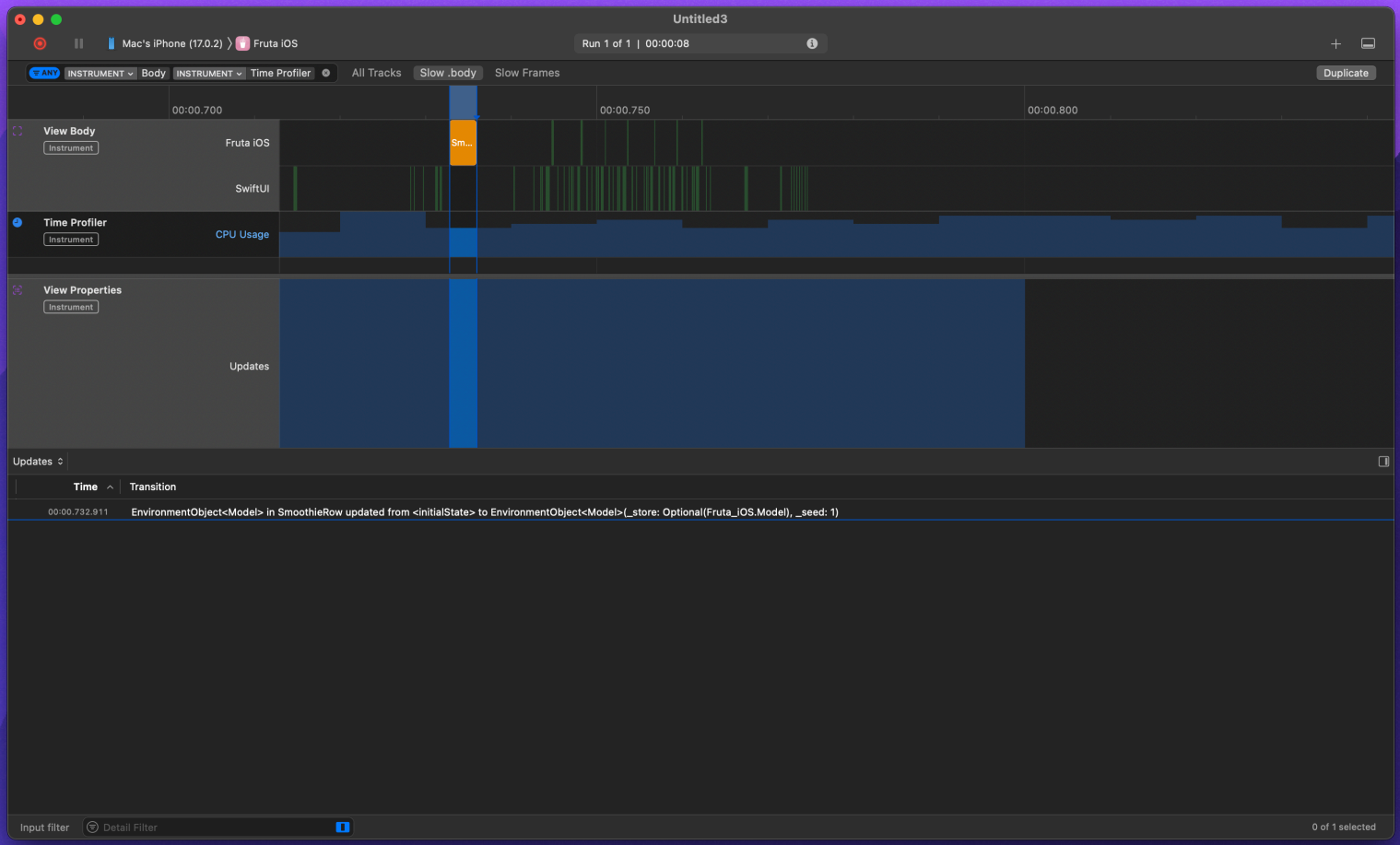
Body

Properties

Commits

Time Profiler

Hangs



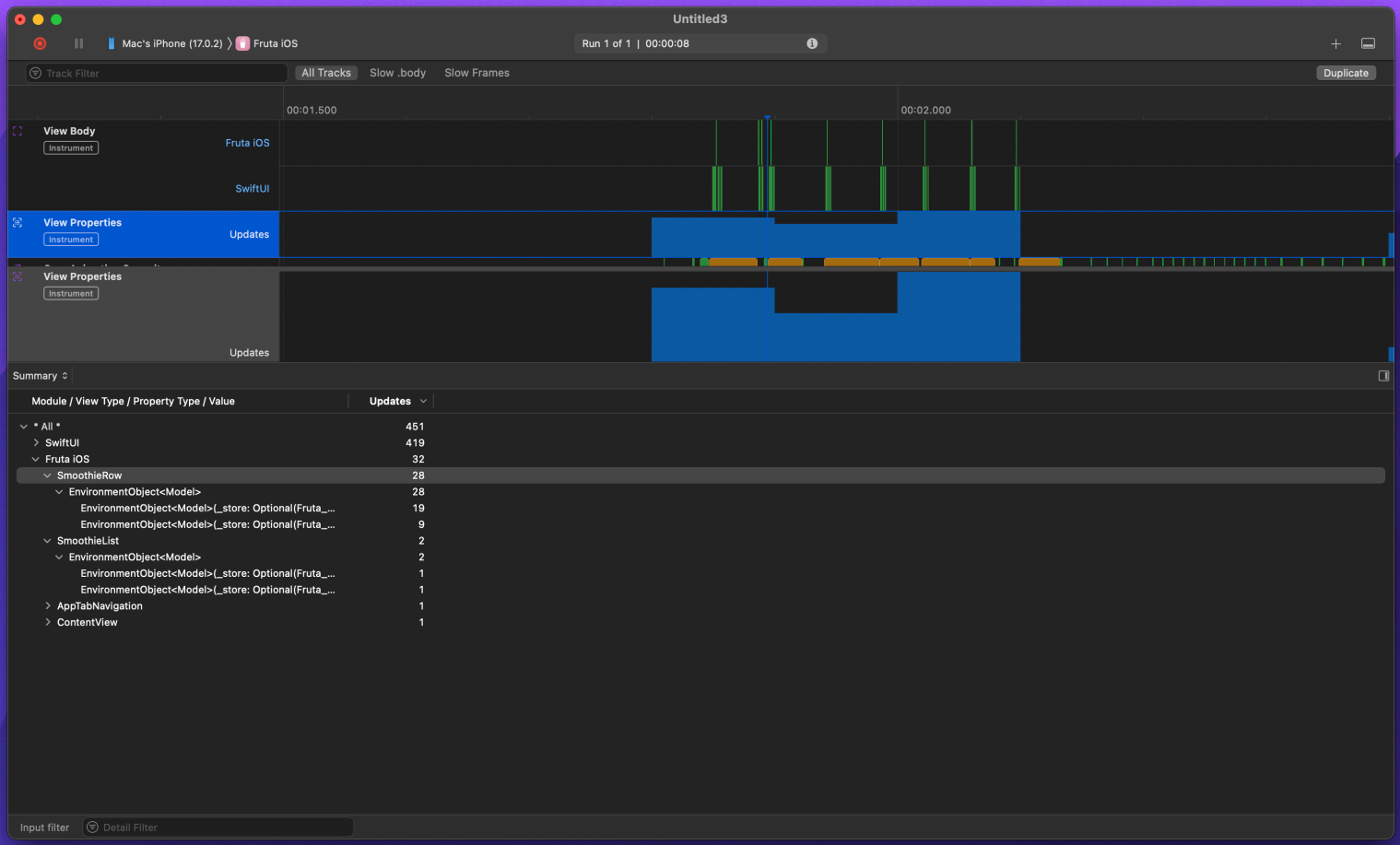
Body

Properties

Commits

Time Profiler

Hangs



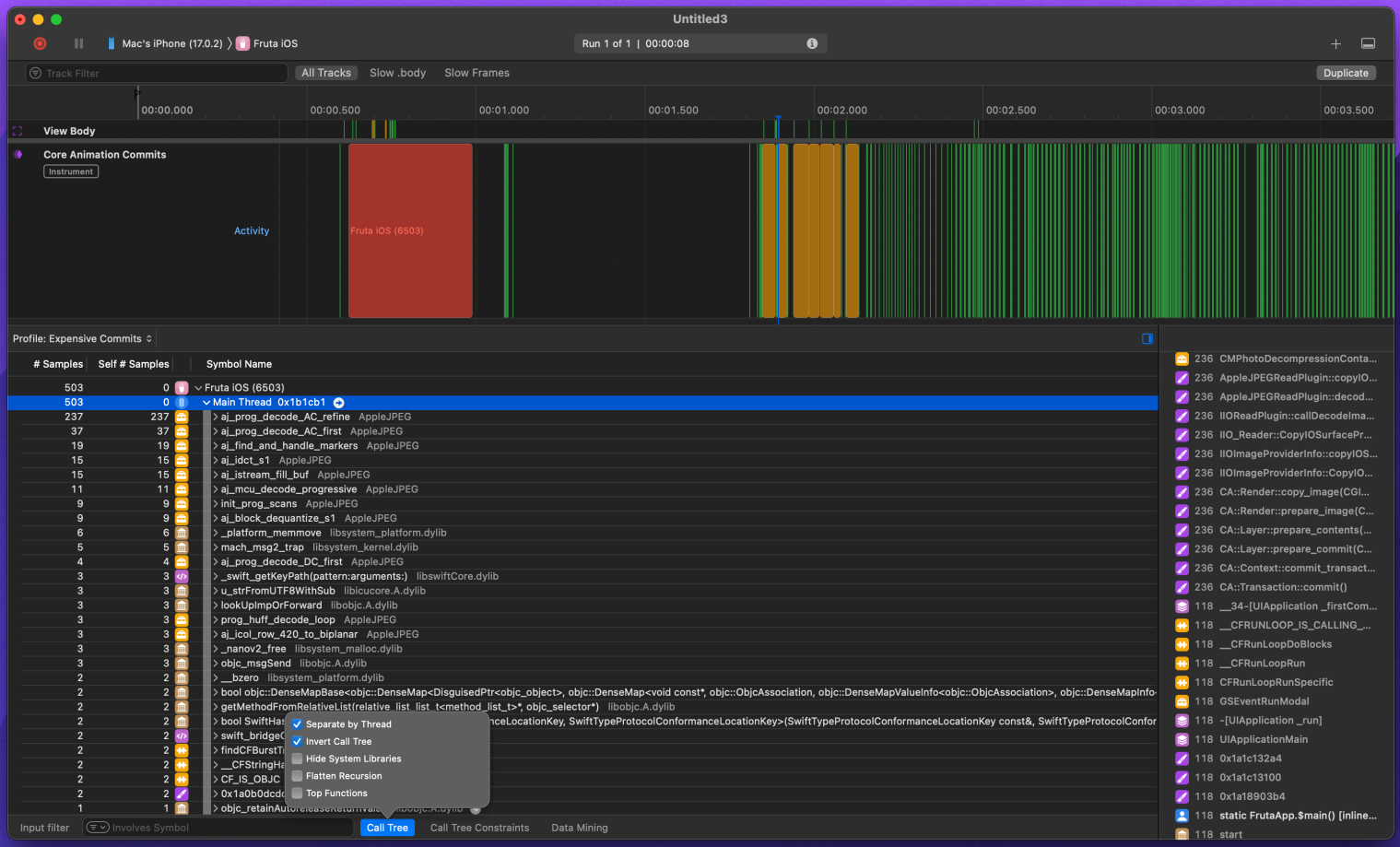
Body

Properties

Commits

Time Profiler

Hangs



CATransaction

A mechanism for grouping multiple layer-tree operations into atomic updates to the render tree.

Body

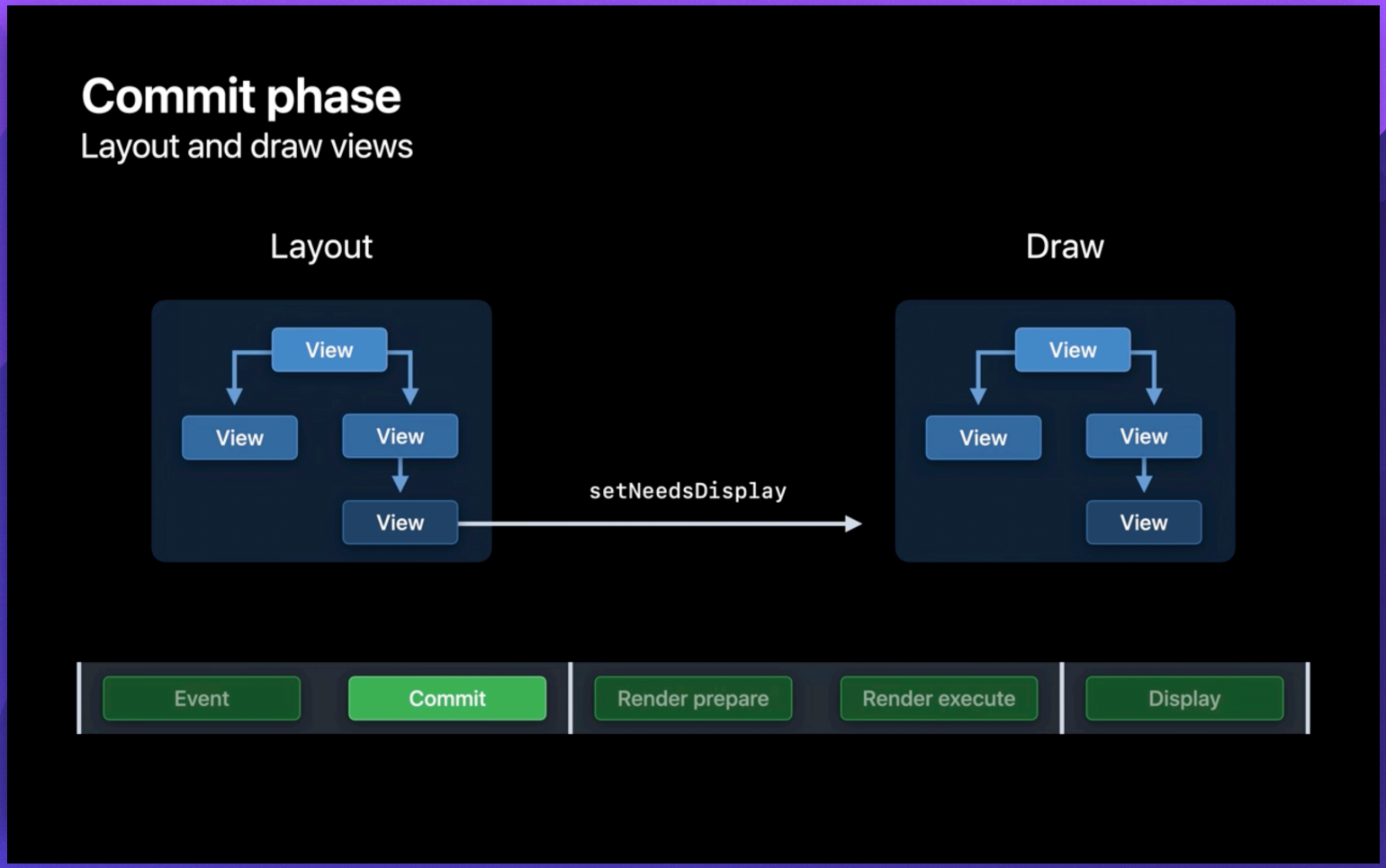
Properties

Commits

Time Profiler

Hangs

Explore UI animation hitches and the render loop, WWDC 21



CATransaction

A mechanism for grouping multiple layer-tree operations into atomic updates to the render tree.

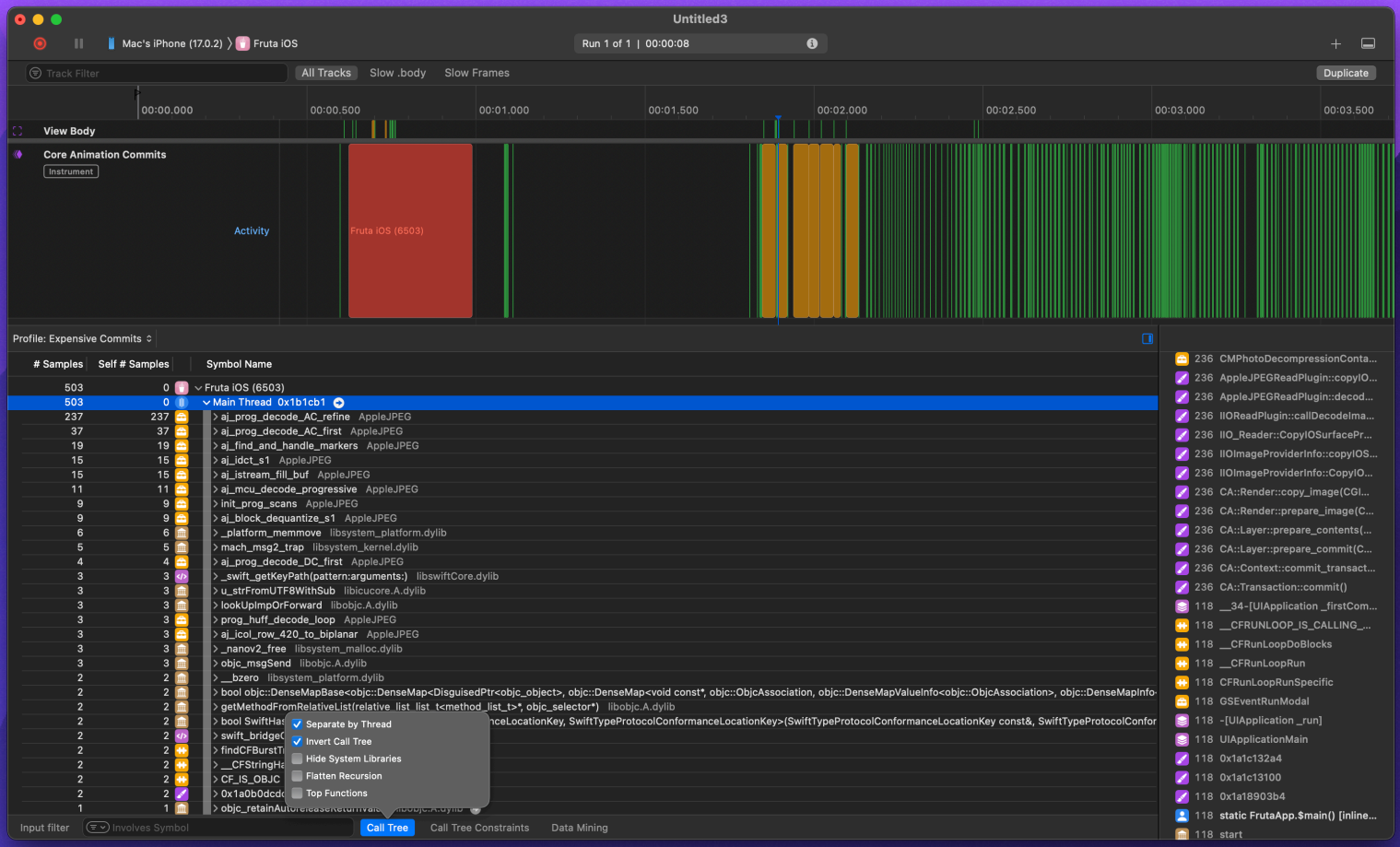
Body

Properties

Commits

Time Profiler

Hangs



CATransaction

A mechanism for grouping multiple layer-tree operations into atomic updates to the render tree.

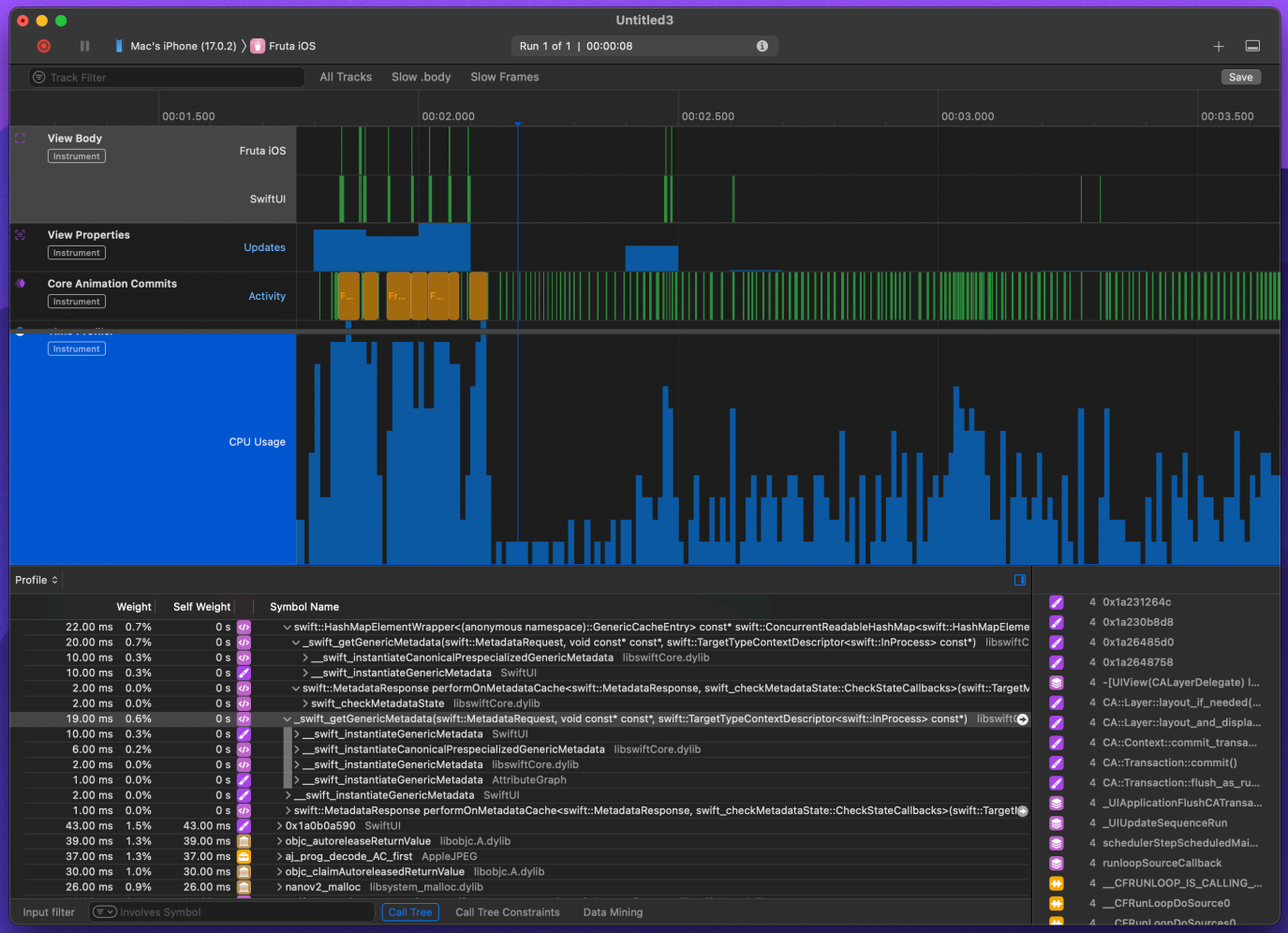
Body

Properties

Commits

Time Profiler

Hangs



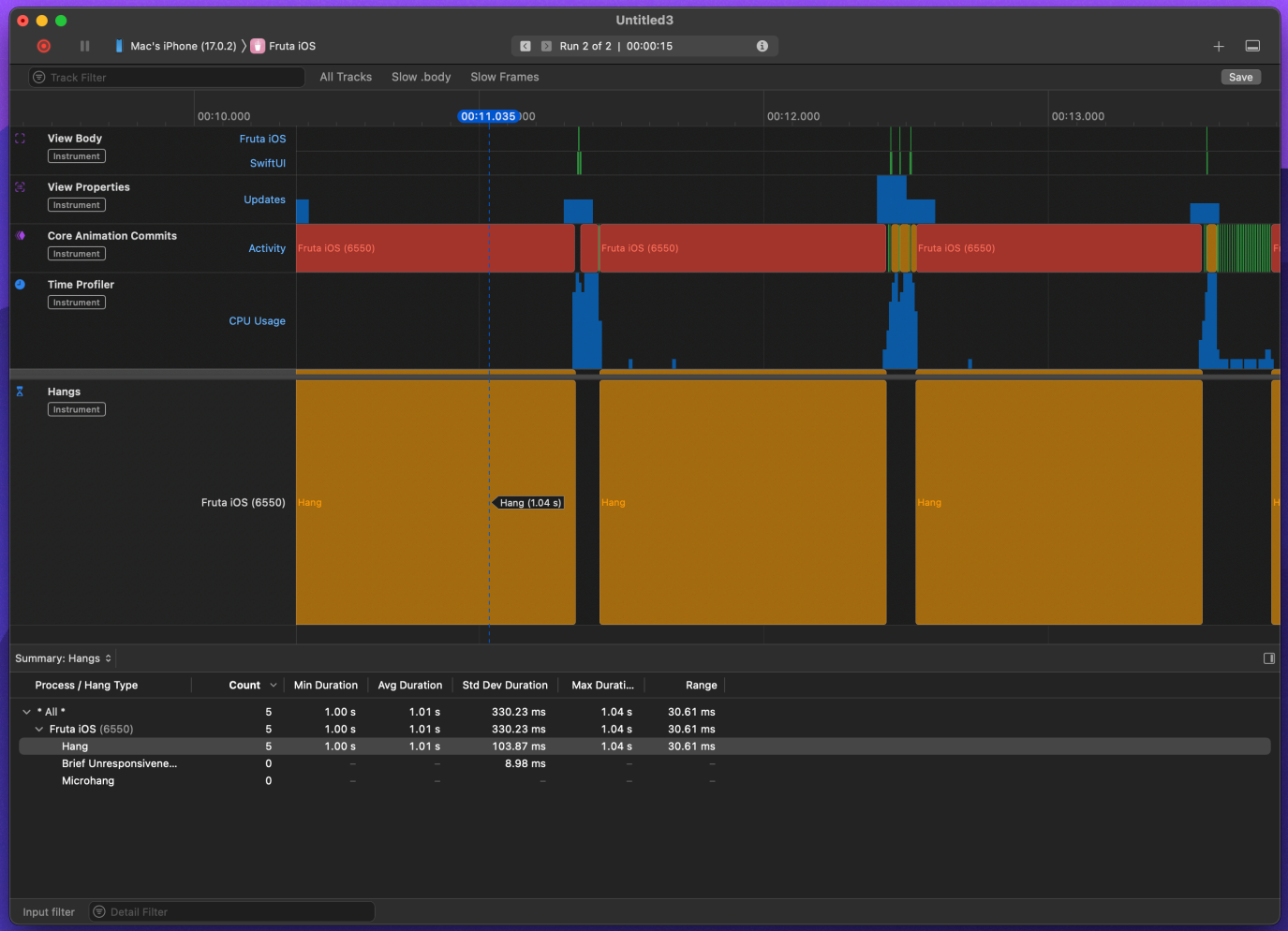
Body

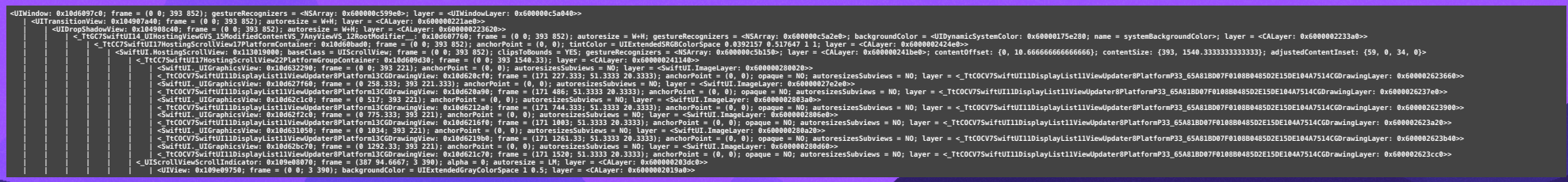
Properties

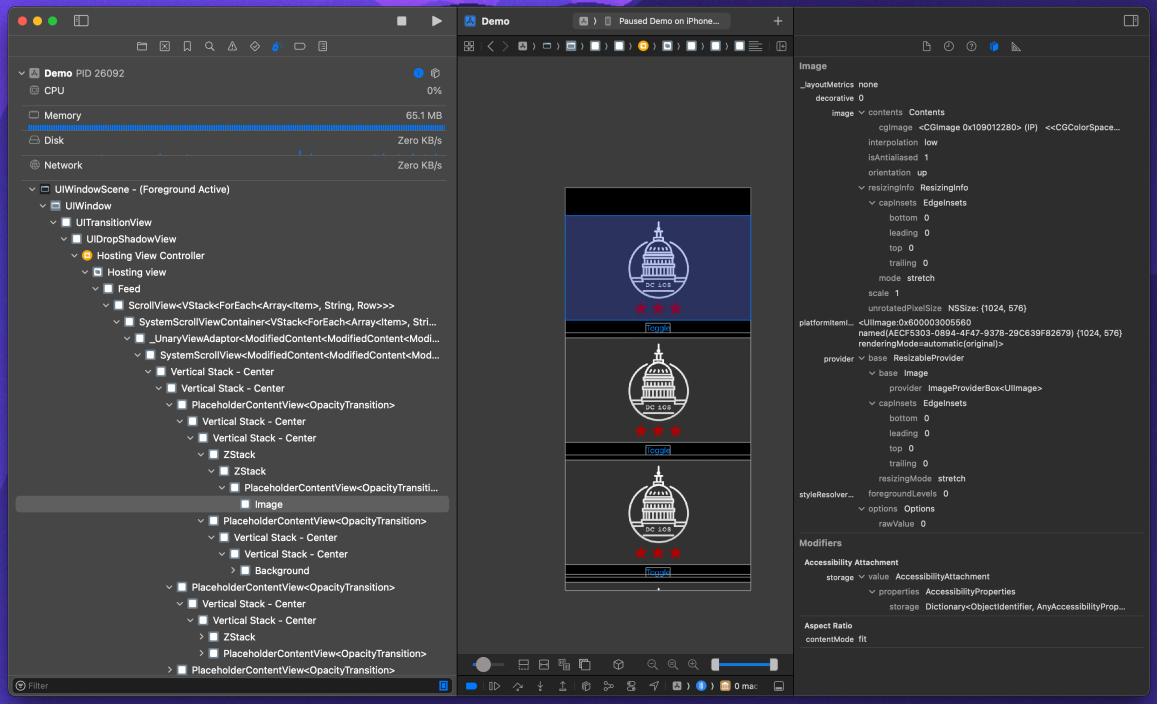
Commits

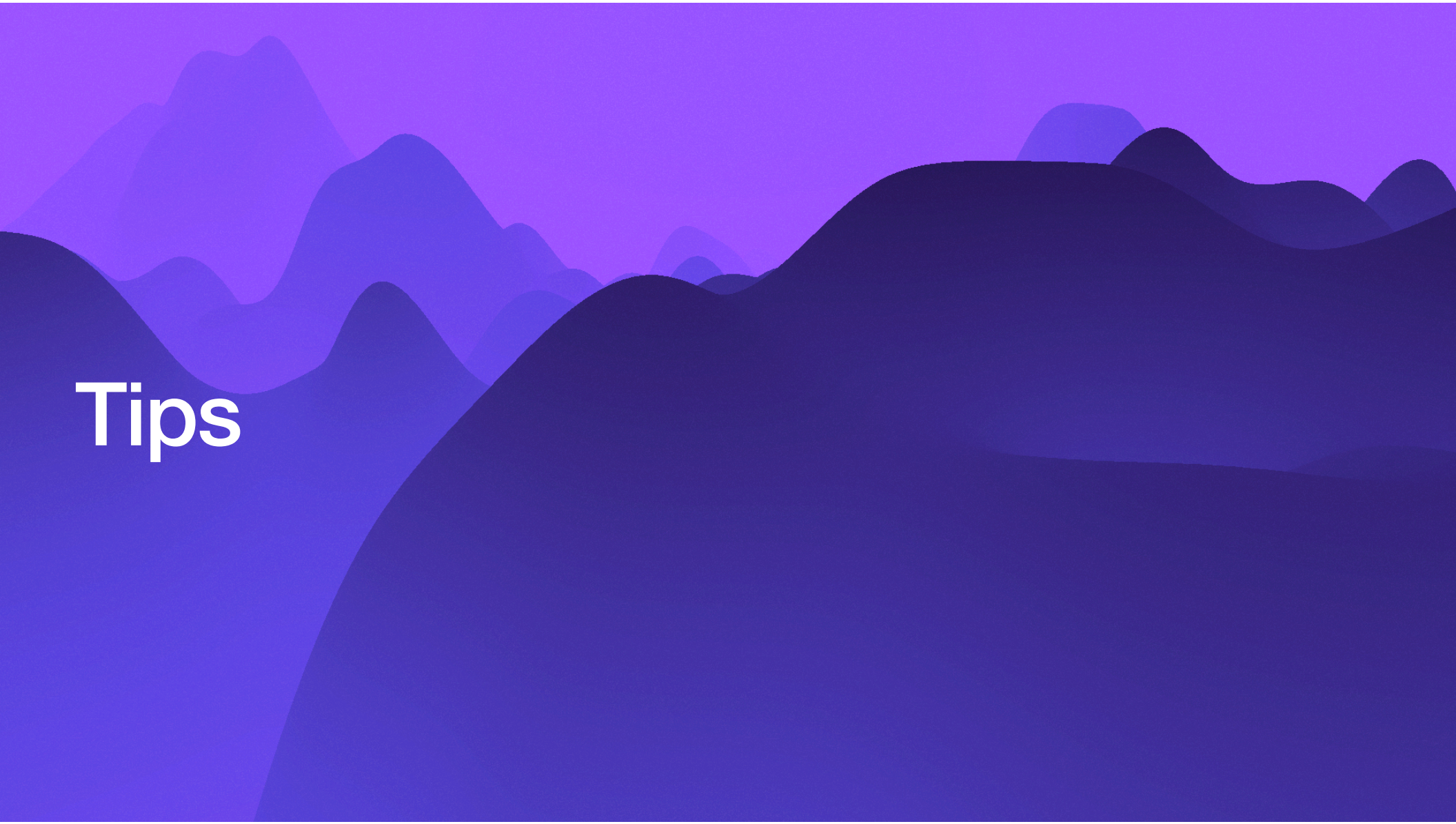
Time Profiler

Hangs




<UIWindow: 0x10d6097c0; frame = (0 0; 393 852); gestureRecognizers = <NSArray: 0x600000c599e0>; layer = <UIWindowLayer: 0x600000c5a040>>
| <UITransitionView: 0x104907a40; frame = (0 0; 393 852); autoresize = W+H; layer = <CALayer: 0x60000221ae0>>
| | <UIDropShadowView: 0x104908c40; frame = (0 0; 393 852); autoresize = W+H; layer = <CALayer: 0x600000223620>>
| | | <_TtGC7SwiftUI14_UIHostingViewGVS_15ModifiedContentVS_7AnyViewVS_12RootModifier__: 0x10d607760; frame = (0 0; 393 852); autoresize = W+H; gestureRecognizers = <NSArray: 0x600000c5a2e0>; backgroundColor = <UIDynamicSystemColor: 0x60000175e280; name = systemBackgroundColor>; layer = <CALayer: 0x6000002233a0>>
| | | | <_TtCC7SwiftUI17HostingScrollView17PlatformContainer: 0x10d60bad0; frame = (0 0; 393 852); anchorPoint = (0, 0); tintColor = UIExtendedSRGBColorSpace 0.0392157 0.517647 1 1; layer = <CALayer: 0x6000002424e0>>
| | | | | <SwiftUI.HostingScrollView: 0x113019000; baseClass = UIScrollView; frame = (0 0; 393 852); clipsToBounds = YES; gestureRecognizers = <NSArray: 0x600000c5b150>; layer = <CALayer: 0x600000241be0>; contentOffset: {0, 10.6666666666666666}; contentSize: {393, 1540.3333333333333}; adjustedContentInset: {59, 0, 34, 0}>
| | | | | | <_TtCC7SwiftUI17HostingScrollView22PlatformGroupContainer: 0x10d609d30; frame = (0 0; 393 1540.33); layer = <CALayer: 0x600000241140>>
| | | | | | | <SwiftUI._UIGraphicsView: 0x10d632290; frame = (0 0; 393 221); anchorPoint = (0, 0); autoresizesSubviews = NO; layer = <SwiftUI.ImageLayer: 0x60000280020>>
| | | | | | | <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8Platform13CGDrawingView: 0x10d620cf0; frame = (171 227.333; 51.3333 20.3333); anchorPoint = (0, 0); opaque = NO; autoresizesSubviews = NO; layer = <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8PlatformP33_65A81BD07F0108B0485D2E15DE104A7514CGDrawingLayer: 0x600002623660>>
| | | | | | | <SwiftUI._UIGraphicsView: 0x10d627f60; frame = (0 258.333; 393 221.333); anchorPoint = (0, 0); autoresizesSubviews = NO; layer = <SwiftUI.ImageLayer: 0x60000027e2e0>>
| | | | | | | <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8Platform13CGDrawingView: 0x10d620a90; frame = (171 486; 51.3333 20.3333); anchorPoint = (0, 0); opaque = NO; autoresizesSubviews = NO; layer = <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8PlatformP33_65A81BD07F0108B0485D2E15DE104A7514CGDrawingLayer: 0x6000026237e0>>
| | | | | | | <SwiftUI._UIGraphicsView: 0x10d62c1c0; frame = (0 517; 393 221); anchorPoint = (0, 0); autoresizesSubviews = NO; layer = <SwiftUI.ImageLayer: 0x6000002803a0>>
| | | | | | | <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8Platform13CGDrawingView: 0x10d6212a0; frame = (171 744.333; 51.3333 20.3333); anchorPoint = (0, 0); opaque = NO; autoresizesSubviews = NO; layer = <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8PlatformP33_65A81BD07F0108B0485D2E15DE104A7514CGDrawingLayer: 0x600002623900>>
| | | | | | | <SwiftUI._UIGraphicsView: 0x10d62f2c0; frame = (0 775.333; 393 221); anchorPoint = (0, 0); autoresizesSubviews = NO; layer = <SwiftUI.ImageLayer: 0x6000002806e0>>
| | | | | | | <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8Platform13CGDrawingView: 0x10d6216f0; frame = (171 1003; 51.3333 20.3333); anchorPoint = (0, 0); opaque = NO; autoresizesSubviews = NO; layer = <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8PlatformP33_65A81BD07F0108B0485D2E15DE104A7514CGDrawingLayer: 0x600002623a20>>
| | | | | | | <SwiftUI._UIGraphicsView: 0x10d631050; frame = (0 1034; 393 221); anchorPoint = (0, 0); autoresizesSubviews = NO; layer = <SwiftUI.ImageLayer: 0x60000280a20>>
| | | | | | | <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8Platform13CGDrawingView: 0x10d6219b0; frame = (171 1261.33; 51.3333 20.3333); anchorPoint = (0, 0); opaque = NO; autoresizesSubviews = NO; layer = <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8PlatformP33_65A81BD07F0108B0485D2E15DE104A7514CGDrawingLayer: 0x600002623b40>>
| | | | | | | <SwiftUI._UIGraphicsView: 0x10d62bc70; frame = (0 1292.33; 393 221); anchorPoint = (0, 0); autoresizesSubviews = NO; layer = <SwiftUI.ImageLayer: 0x60000280d60>>
| | | | | | | <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8Platform13CGDrawingView: 0x10d621c70; frame = (171 1520; 51.3333 20.3333); anchorPoint = (0, 0); opaque = NO; autoresizesSubviews = NO; layer = <_TtCOCV7SwiftUI11DisplayList11ViewUpdater8PlatformP33_65A81BD07F0108B0485D2E15DE104A7514CGDrawingLayer: 0x600002623cc0>>
| | | | | <_UIScrollViewScrollIndicator: 0x109e08070; frame = (387 94.6667; 3 390); alpha = 0; autoresize = LM; layer = <CALayer: 0x600000203dc0>>
| | | | | | <UIView: 0x109e09750; frame = (0 0; 3 390); backgroundColor = UIExtendedGrayColorSpace 1 0.5; layer = <CALayer: 0x6000002019a0>>


Demo PID 26092
CPU 0%
Memory 65.1 MB
Disk Zero KB/s
Network Zero KB/s
UIWindowScene - (Foreground Active)
UIWindow
UITransitionView
UIDropShadowView
Hosting View Controller
Hosting view
Feed
ScrollView<VStack<ForEach<Array<Item>, String, Row>>>
SystemScrollViewContainer<VStack<ForEach<Array<Item>, Stri...
_UnaryViewAdaptor<ModifiedContent<ModifiedContent<Modi...
SystemScrollView<ModifiedContent<ModifiedContent<Mod...
Vertical Stack - Center
Vertical Stack - Center
PlaceholderContentView<OpacityTransition>
Vertical Stack - Center
Vertical Stack - Center
ZStack
ZStack
PlaceholderContentView<OpacityTransiti...
Image
PlaceholderContentView<OpacityTransition>
Vertical Stack - Center
Vertical Stack - Center
Background
PlaceholderContentView<OpacityTransition>
Vertical Stack - Center
Vertical Stack - Center
ZStack
PlaceholderContentView<OpacityTransition>
PlaceholderContentView<OpacityTransition>
Filter
Demo
Paused Demo on iPhone...
Toggle
Toggle
Toggle
Image
_layoutMetrics none
decorative 0
image contents Contents
cgImage <CGImage 0x109012280> (IP) <<CGColorSpace...
interpolation low
isAntialiased 1
orientation up
resizingInfo ResizingInfo
capInsets EdgeInsets
bottom 0
leading 0
top 0
trailing 0
mode stretch
scale 1
unrotatedPixelSize NSSize: {1024, 576}
platformItemI... <UIImage:0x600003005560 named(AECF5303-0894-4F47-9378-29C639F82679) {1024, 576} renderingMode=automatic(original)>
provider base ResizableProvider
base Image
provider ImageProviderBox<UIImage>
capInsets EdgeInsets
bottom 0
leading 0
top 0
trailing 0
resizingMode stretch
styleResolver... foregroundLevels 0
options Options
rawValue 0
Modifiers
Accessibility Attachment
storage value AccessibilityAttachment
properties AccessibilityProperties
storage Dictionary<ObjectIdentifier, AnyAccessibilityProp...
Aspect Ratio
contentMode fit


Tips

Tips

Avoid AnyView

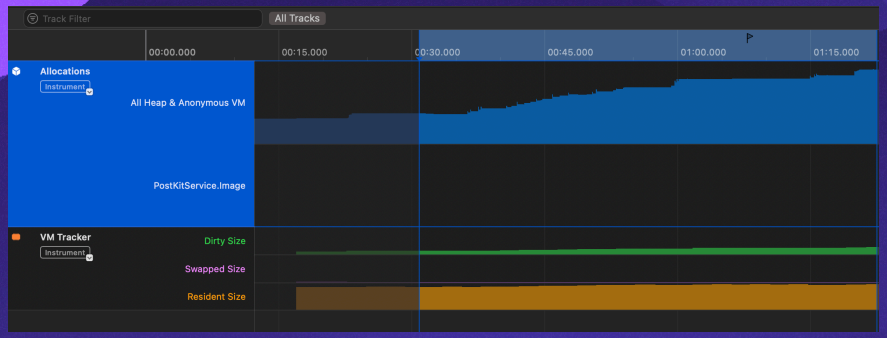
- Erases view type, results in more redraws as SwiftUI cannot compare identities
- Use @ViewBuilder instead

```
/// A type-erased view.
///
/// An `AnyView` allows changing the type of view used in a given view
/// hierarchy. Whenever the type of view used with an `AnyView` changes, the old
/// hierarchy is destroyed and a new hierarchy is created for the new type.
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
@frozen public struct AnyView : View { … }
```

Tips

Leverage SwiftUI's Lazy Loading

- LazyVStack, LazyHStack, LazyVGrid
- **Beware of expensive state though**




```
struct ContentView: View {

    @State private var image: UIImage?
    var body: some View {... }
}
```

Tips

Common sources of slow bodies

- Heap allocation
- String interpolation
- Bundle lookup (e.g. Image initializer)
- Expensive identification

```
struct LoadingImageView<Placeholder: View>: View {

    let placeholder: () -> Placeholder
    var user: User

    public init(_ user: User, @ViewBuilder placeholder: @escaping () -> Placeholder) { ... }

    var body: some View {... }
        Text(AttributedString("Loading \(user.name)", attributes: attributes))
        Bundle(for: type(of: ...))
    }
}
```

Tips

Drawing Group

```
func drawingGroup(
    opaque: Bool = false,
    colorMode: ColorRenderingMode = .nonLinear
) -> some View
```

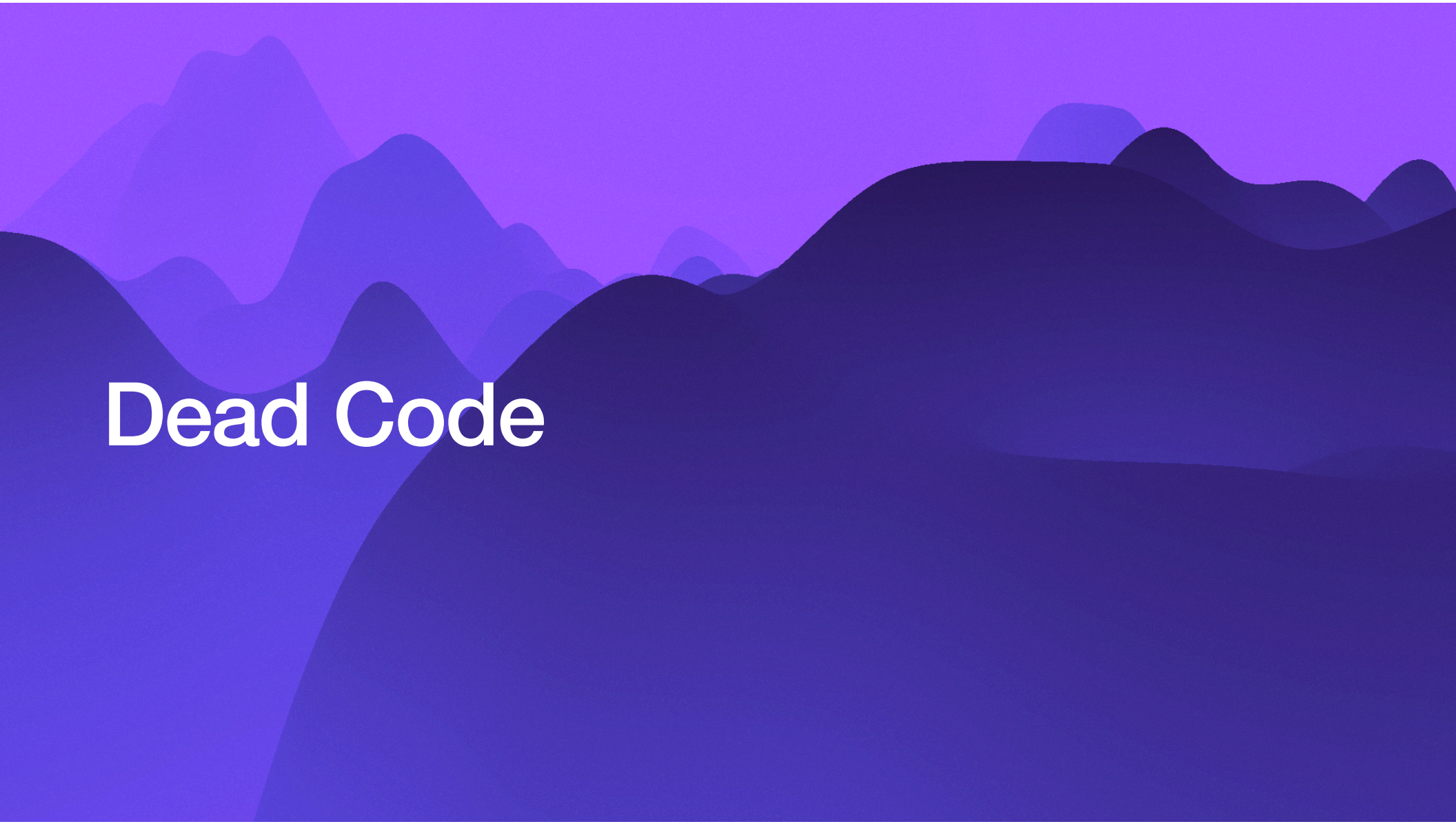

Wrapping Up and What’s Next

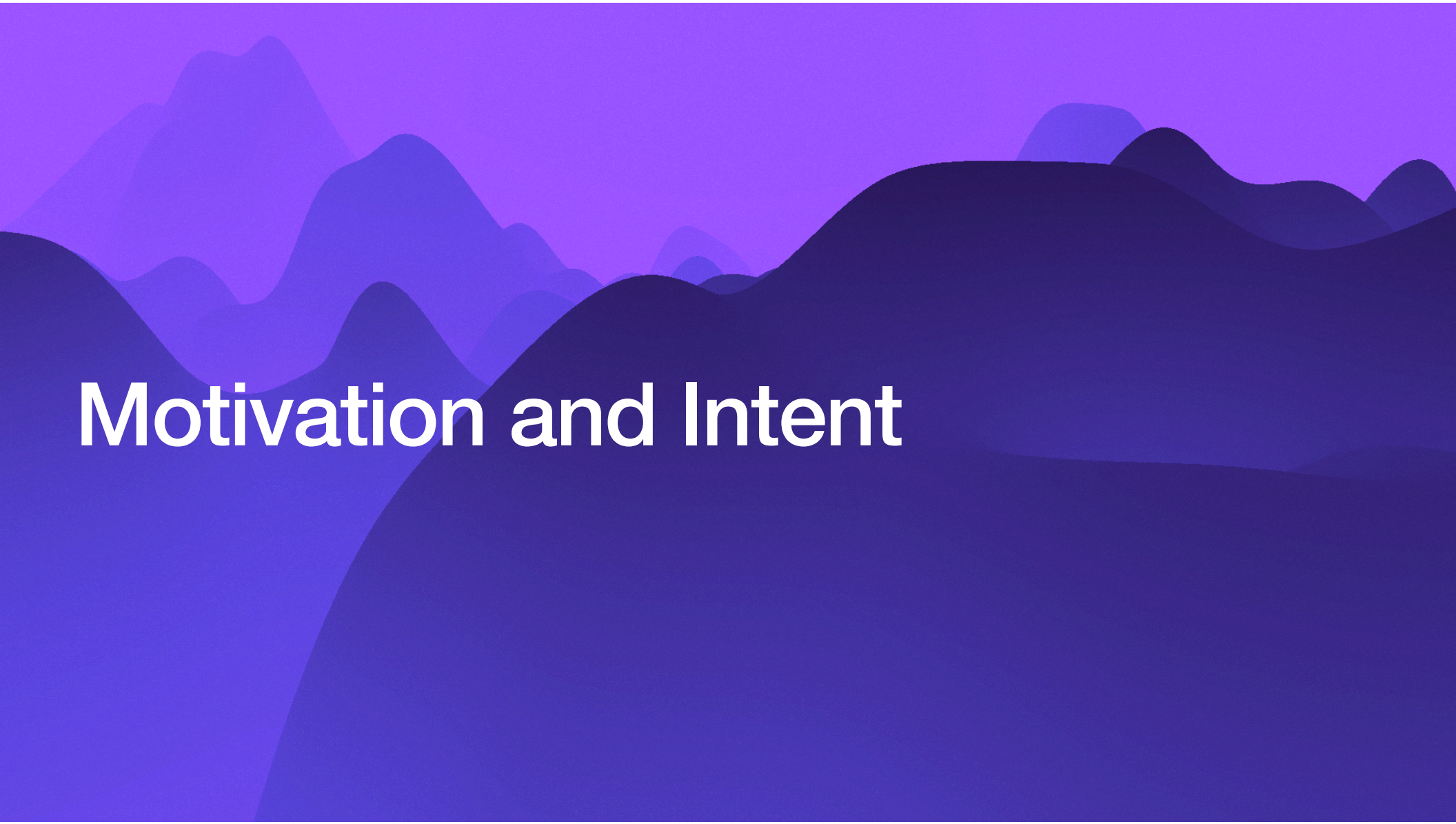

Resources

Resources

- **Demystify SwiftUI, WWDC 21**
  - https://developer.apple.com/wwdc21/10022
- **Demystify SwiftUI Performance, WWDC 23**
  - https://developer.apple.com/videos/play/wwdc2023/10160
- **Explore UI animation hitches and the render loop**
    - https://developer.apple.com/videos/play/tech-talks/10855/
- **A Day in the Life of a SwiftUI View**
  - https://chris.eidhof.nl/presentations/day-in-the-life/
- **The SwiftUI render loop**
  - https://rensbr.eu/blog/swiftui-render-loop/


Questions

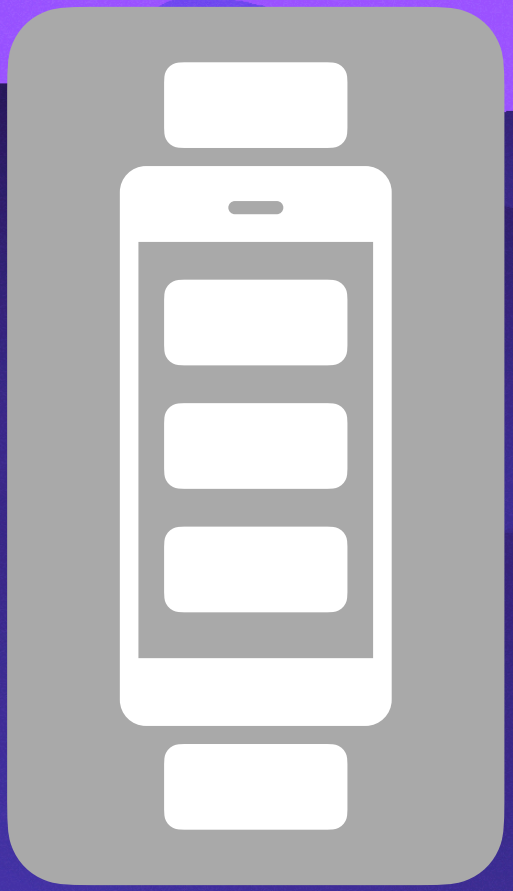

Dead Code


Motivation and Intent

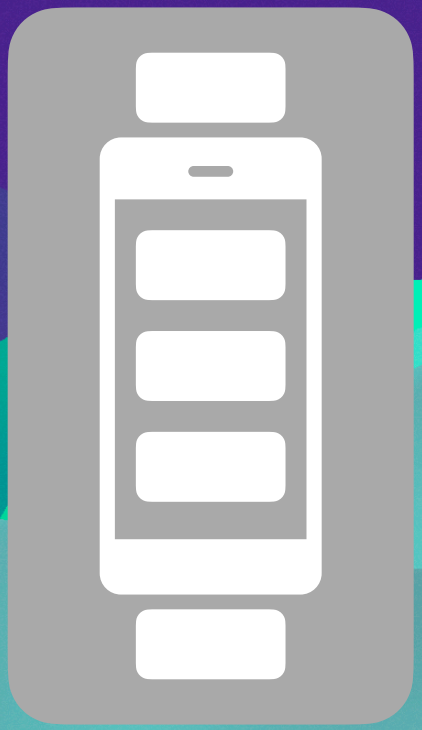
Motivation and Intent

- **Motivation:** Improving my own knowledge and ability to articulate it
- **Motivation:** Giving back to DC iOS
- **My Intention:** You learn at least one thing you didn't know before about SwiftUI

Scenario



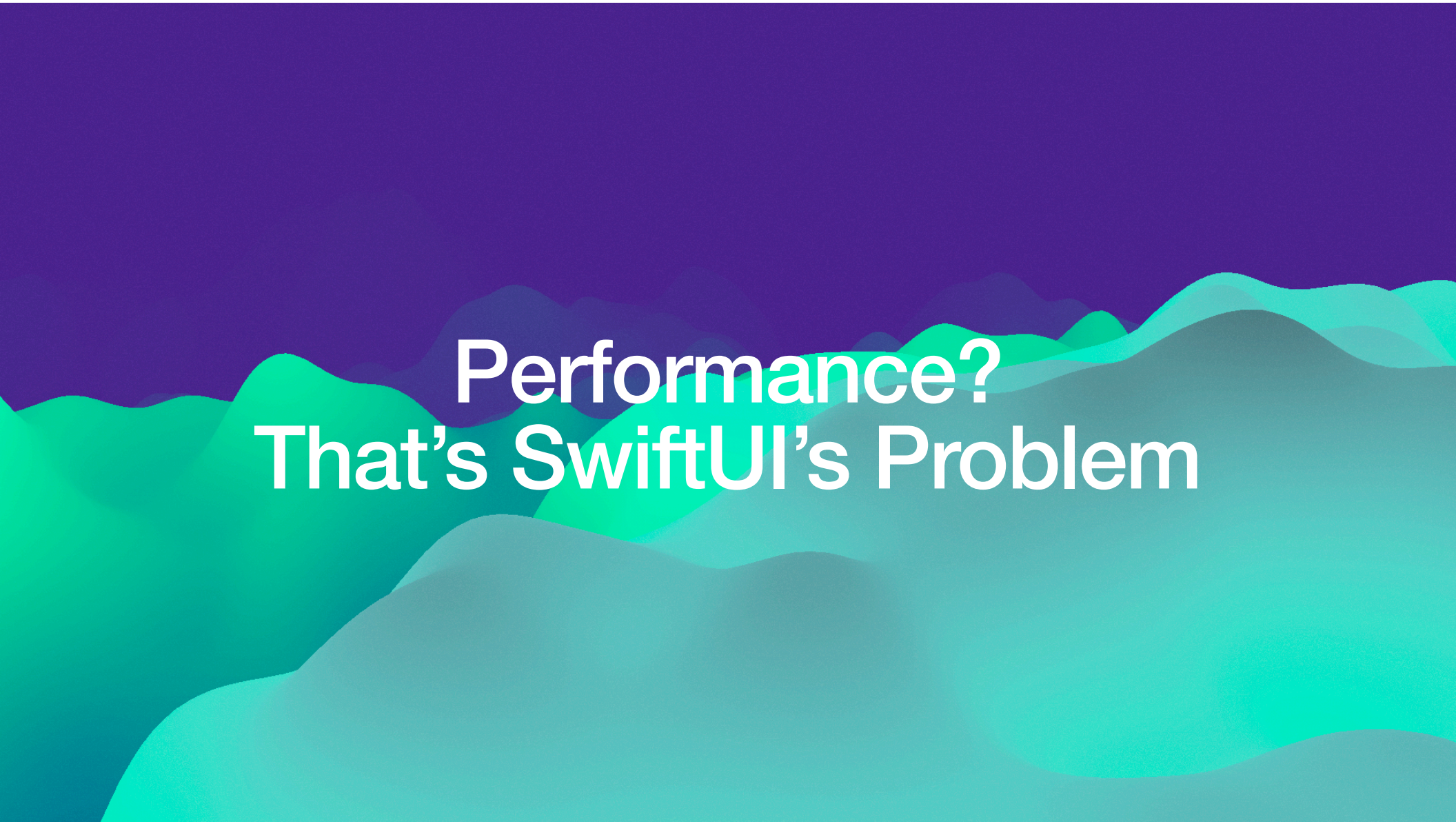
Scenario



- Unresponsive
- Hitches and Hangs
- Broken Animations

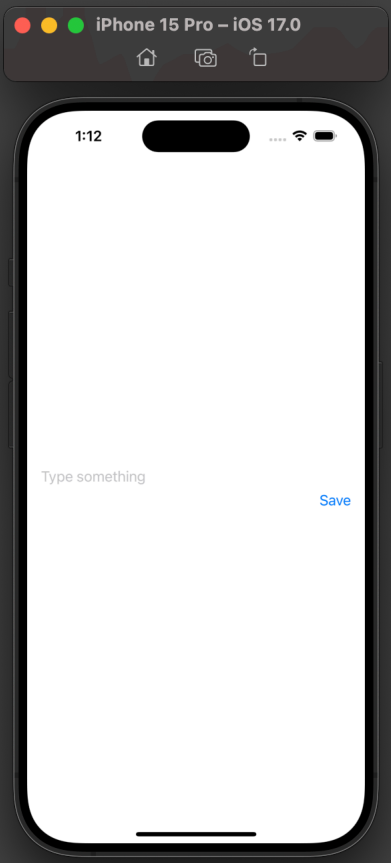
Scenario

- Unresponsive
- Hitches and Hangs
- Broken Animations

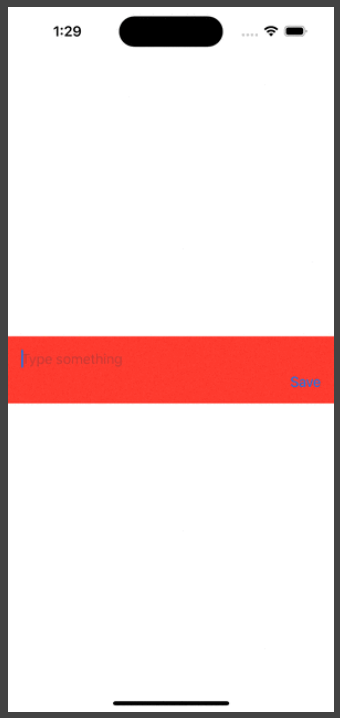

Performance?
That’s SwiftUI’s Problem


SwiftUI Can Be Optimized

```swift
struct Record {
    var content: String
}

struct ContentView: View {

    @State private var record: Record

    var body: some View {
        VStack(alignment: .center) {
            TextField(text: $record.content) {
                Text("Type something")
            }
            HStack {
                Spacer()
                Button("Save") {
                    print("saved")
                }
            }
        }
        .padding()
    }
}
```

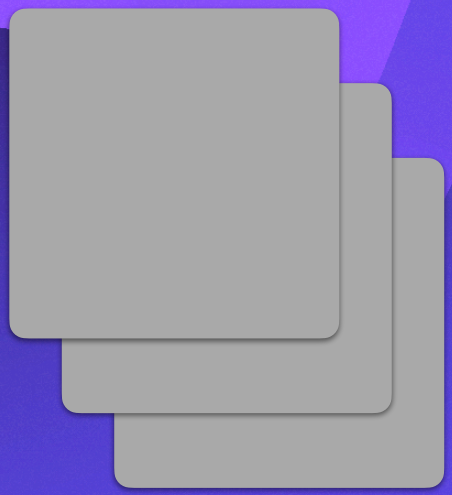


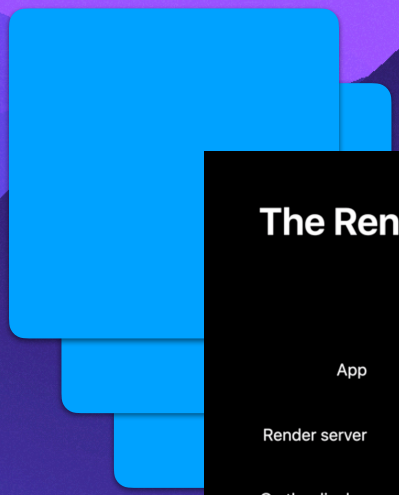
```swift
struct Record {
    var content: String = ""
}

struct ContentView: View {

    @State var record: Record

    var body: some View {
        VStack(alignment: .center) {
            TextField(text: $record.content) {
                Text("Type something")
            }
            HStack {
                Spacer()
                Button("Save") {
                    print("saved")
                }
            }
        }
        .padding()
    }
}
```



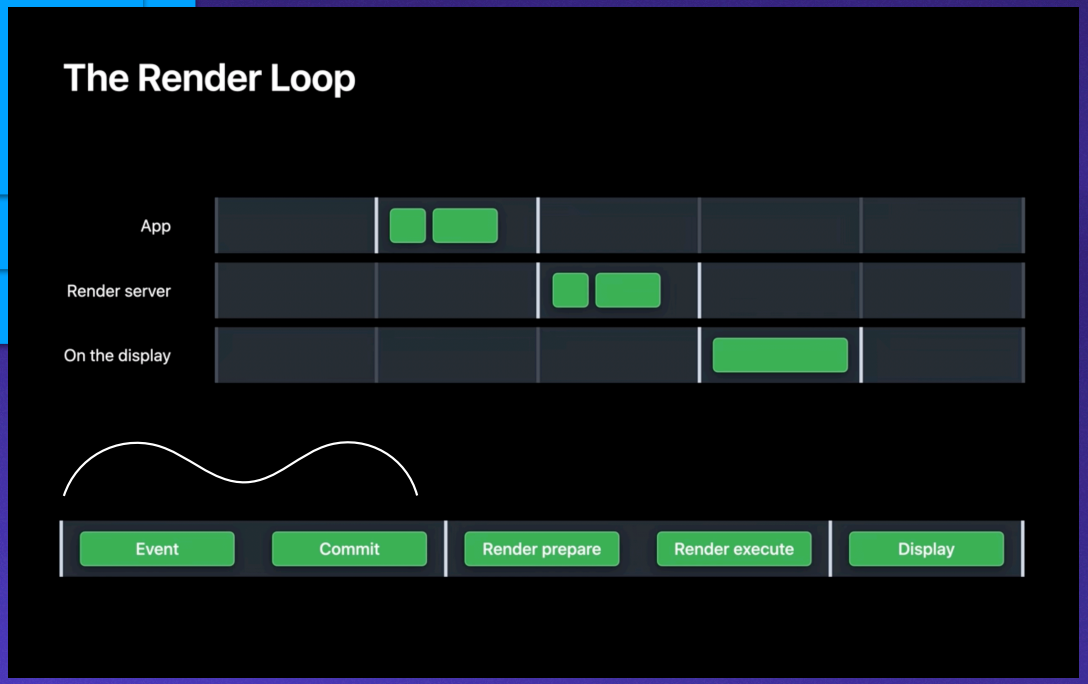
View Tree

Render Tree
(Attribute Graph)





SwiftUI

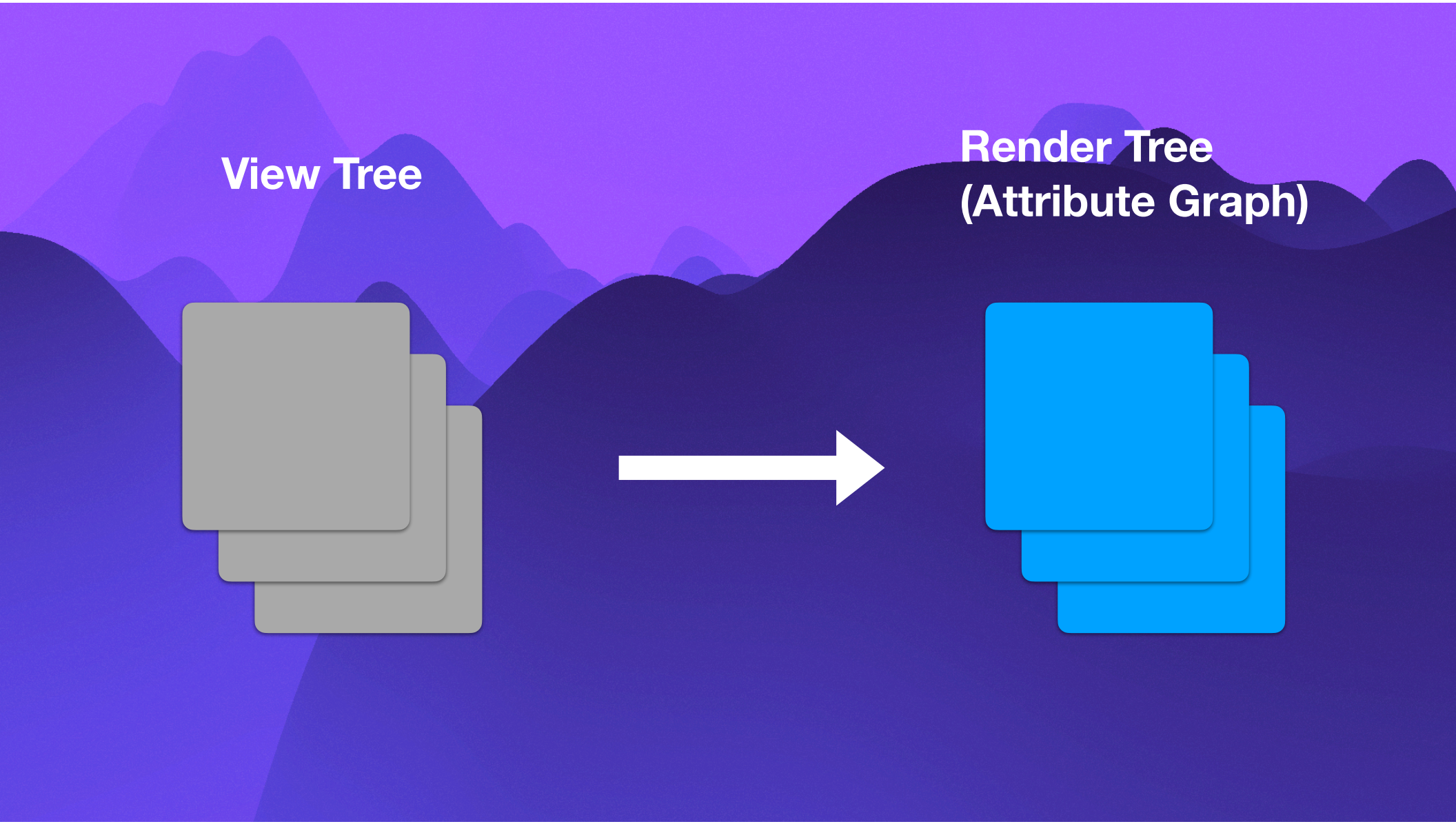


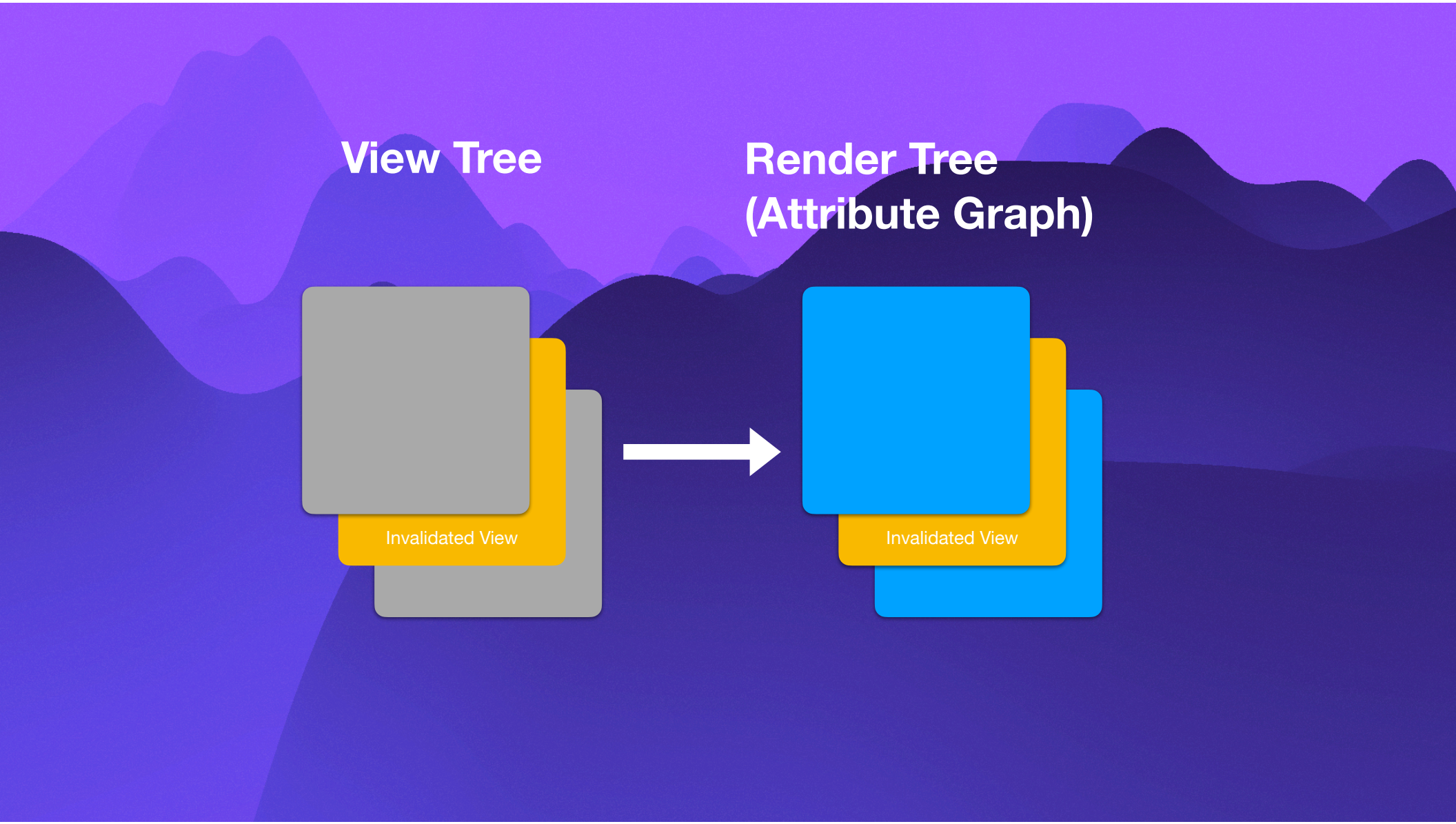
3 | Fine Tuning Techniques

Use Layouts and Frames

- Implement optimized layout using Layout protocol
- Use frame modifier to provide sizes for your views

```
public protocol Layout : Animatable {
   func sizeThatFits(proposal: ProposedViewSize, subviews: Self.Subviews, cache: inout Self.Cache) -> CGSize
   func placeSubviews(in bounds: CGRect, proposal: ProposedViewSize, subviews: Self.Subviews, cache: inout Self.Cache)
   public func updateCache(_ cache: inout Self.Cache, subviews: Self.Subviews)
}
…

   Text(AttributedString("...", attributes: attributes)))
        .frame(width: 300, height: 200)
```


View Tree
Render Tree
(Attribute Graph)


View Tree
Render Tree
(Attribute Graph)
Invalidated View
Invalidated View

Rendering Performance