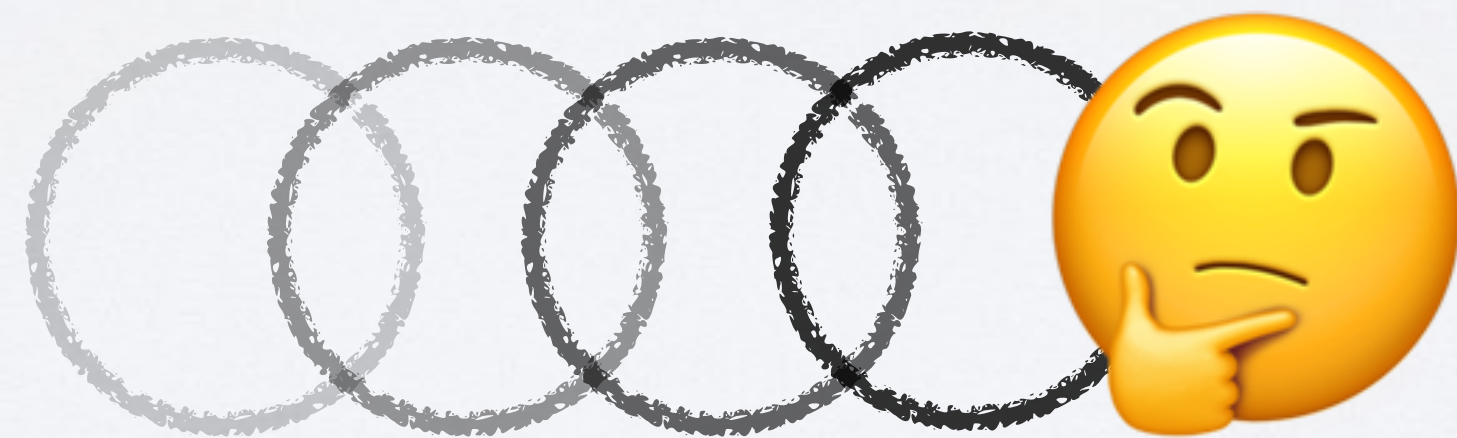


LEARNING FROM IOS ANIMATIONS

Justin Miller • Mapbox



The animation foundations in iOS are well-designed, intuitive, and powerful.

What can we learn about how they are built, the capabilities that they give, and the assumptions that they are built upon, that we can bring to our own software design?

PERSONAL INTRO

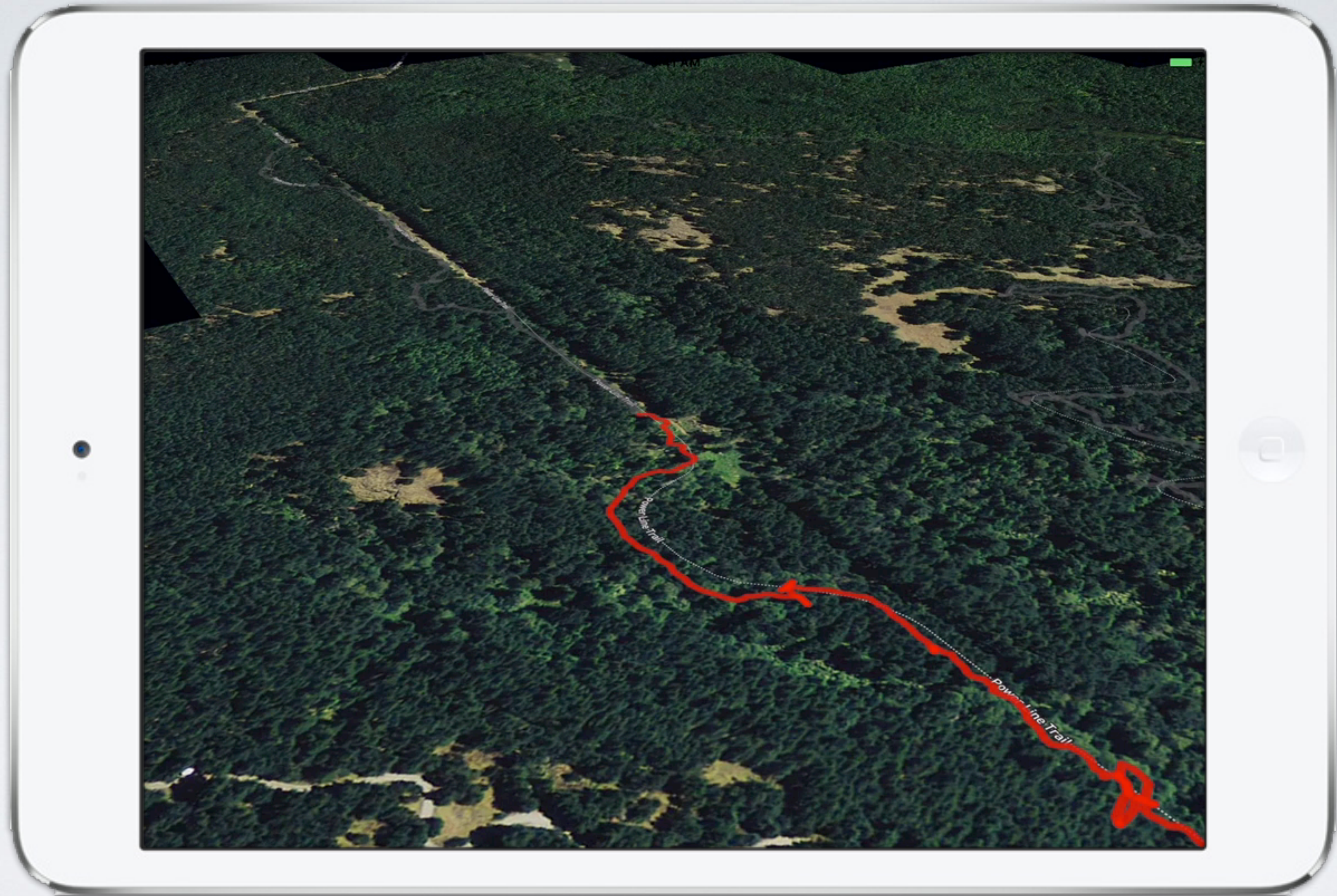
- I've been programming for work for 20 years & have been using Apple technologies for 15 of those
- Perl, PHP, Objective-C, C, C++, Java, Swift (also Bash)
- I've been both an app builder and a tool builder
- I have always admired Apple's API design

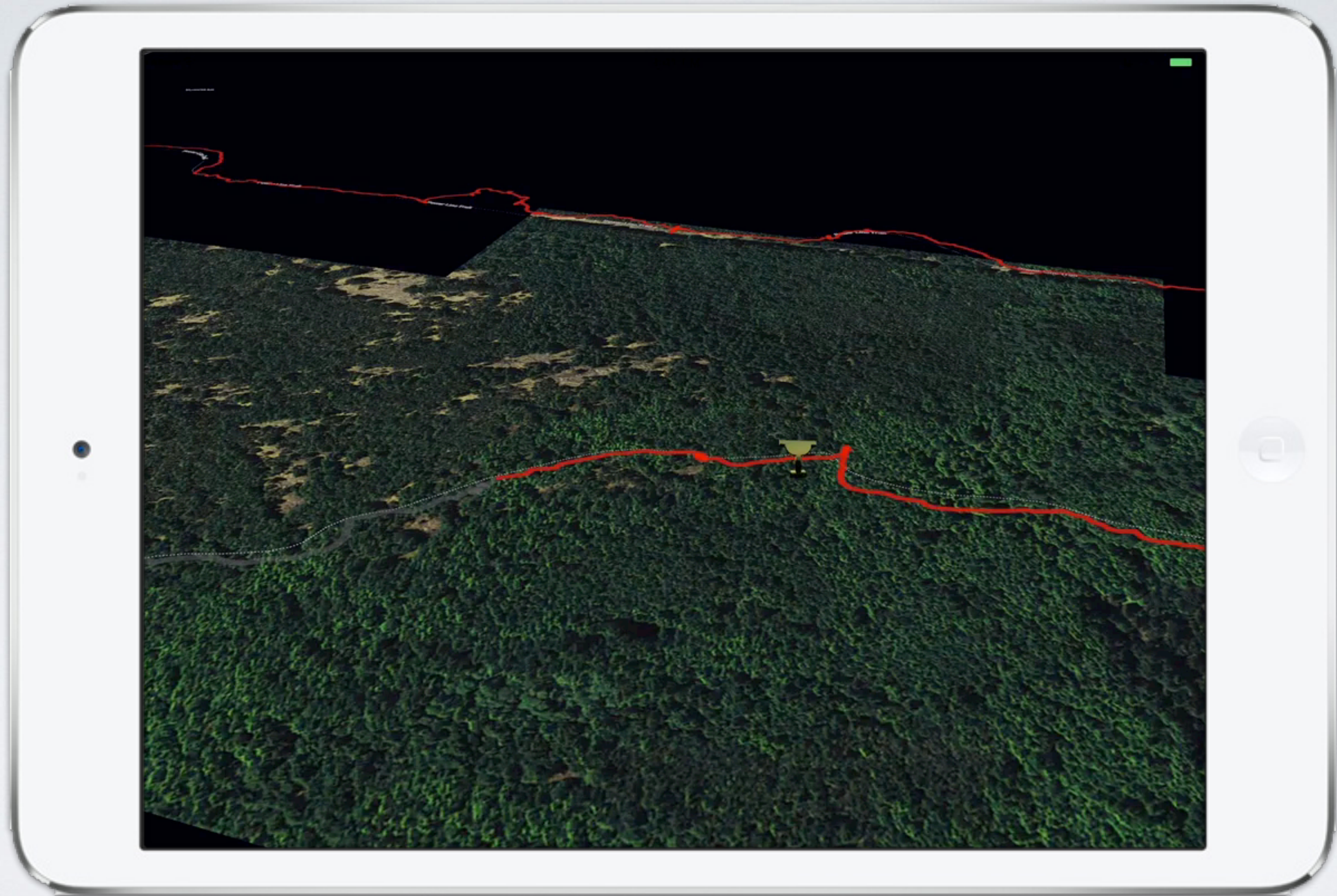
MAPBOX INTRO

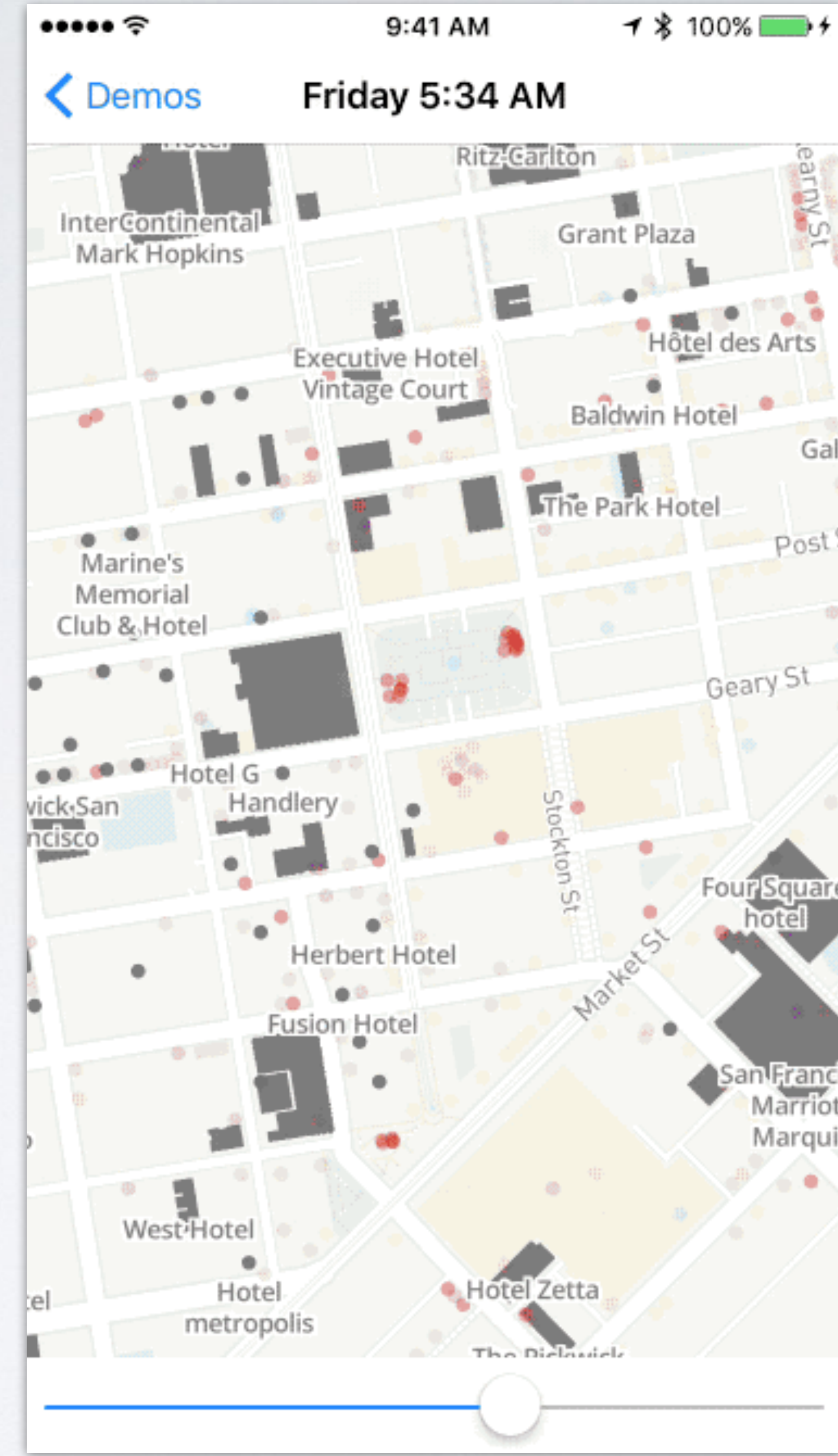
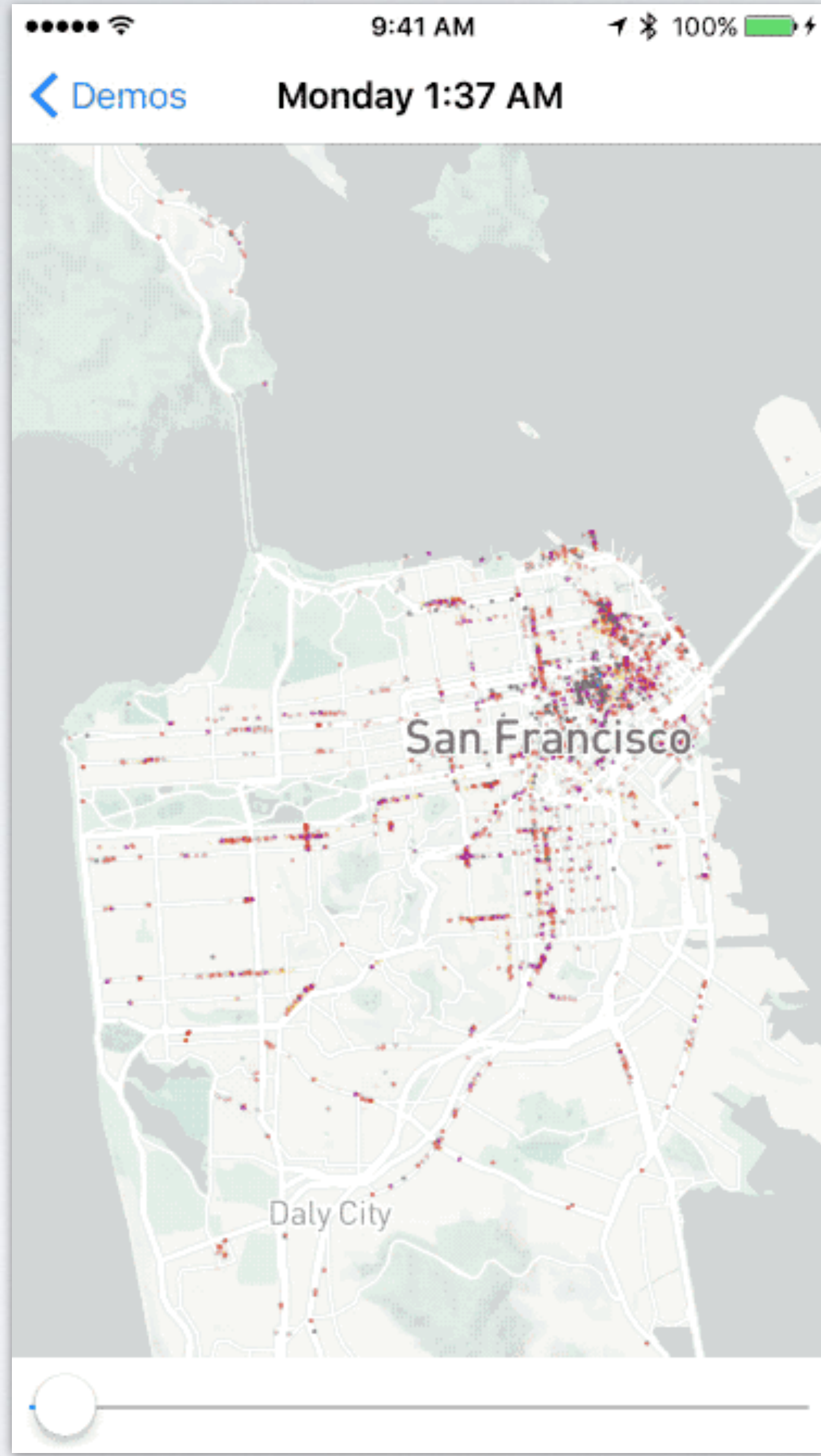
- We're building developer tools for maps & location
 - Customized map appearance & function
- We recently added "runtime styling"
- I have had to think a lot lately about animations

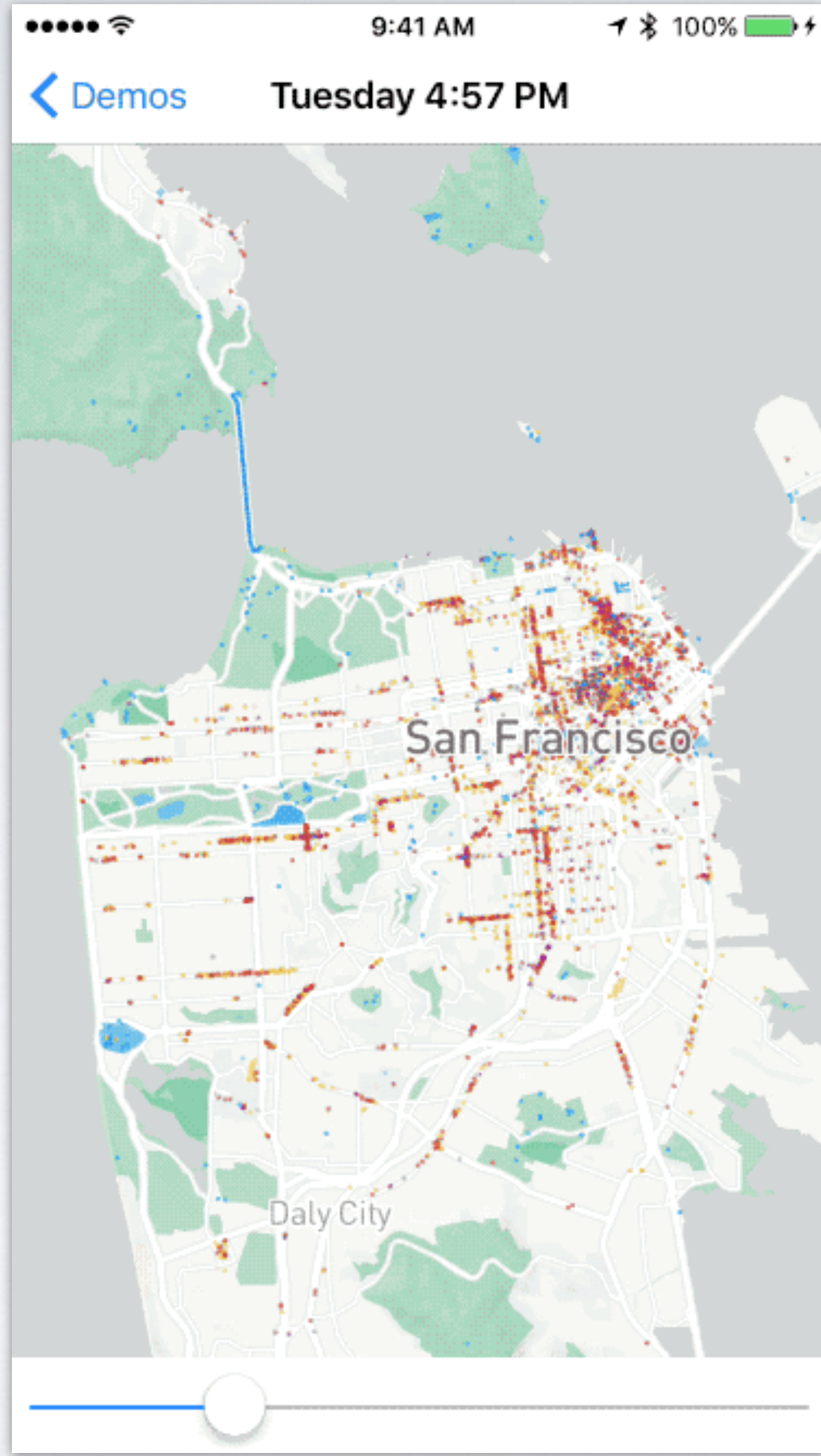












WHAT ARE ANIMATIONS?



MR. BOJANGLES



Bill "Bojangles" Robinson, 1878-1949

MR. BOJANGLES

- Probably the first programming that I can remember
- Intro to animation for the TI-99/4A computer (1979)
- Absolute simplest possible animation
 - Two frames swapped in time, coincident in position

CHAR Worksheet

First Figure

	LEFT BLOCK	RIGHT BLOCK	CODE	SHORT- HAND CODE	DOTS
ROW 1	1	1	99	0	0000
ROW 2	1	1	5A	1	0001
ROW 3	1	1	3C	2	0010
ROW 4	1	1	3C	3	0011
ROW 5	1	1	3C	4	0100
ROW 6	1	1	3C	5	0101
ROW 7	1	1	44	6	0110
ROW 8	1	1	84	7	0111
				8	1000
				9	1001
				A	1010
				B	1011
				C	1100
INPUT TO CHAR:	"995A3C3C3C3C4484"			D	1101
				E	1110
				F	1111

CHAR Worksheet

Second Figure

	LEFT BLOCK				RIGHT BLOCK				CODE	SHORT-HAND CODE	DOTS
ROW 1				1	1				<u>18</u>	0	0000
ROW 2	1			1	1			1	<u>99</u>	1	0001
ROW 3	1	1	1	1	1	1	1	1	<u>FF</u>	2	0010
ROW 4			1	1	1	1			<u>3C</u>	3	0011
ROW 5			1	1	1	1			<u>3C</u>	4	0100
ROW 6			1	1	1	1			<u>3C</u>	5	0101
ROW 7			1					1	<u>22</u>	6	0110
ROW 8			1					1	<u>21</u>	7	0111
										8	1000
										9	1001
										A	1010
										B	1011
										C	1100
INPUT TO CHAR:	<u>"1899FF3C3C3C2221"</u>									D	1101
										E	1110
										F	1111

LIST

10 CALL CLEAR

20 A\$="995A3C3C3C3C4484"

25 B\$="1899FF3C3C3C2221"

30 CALL CHAR(128,A\$)

35 CALL CHAR(129,B\$)

40 CALL COLOR(13,2,16)

50 CALL VCHAR(12,16,128)

60 FOR DELAY=1 TO 100

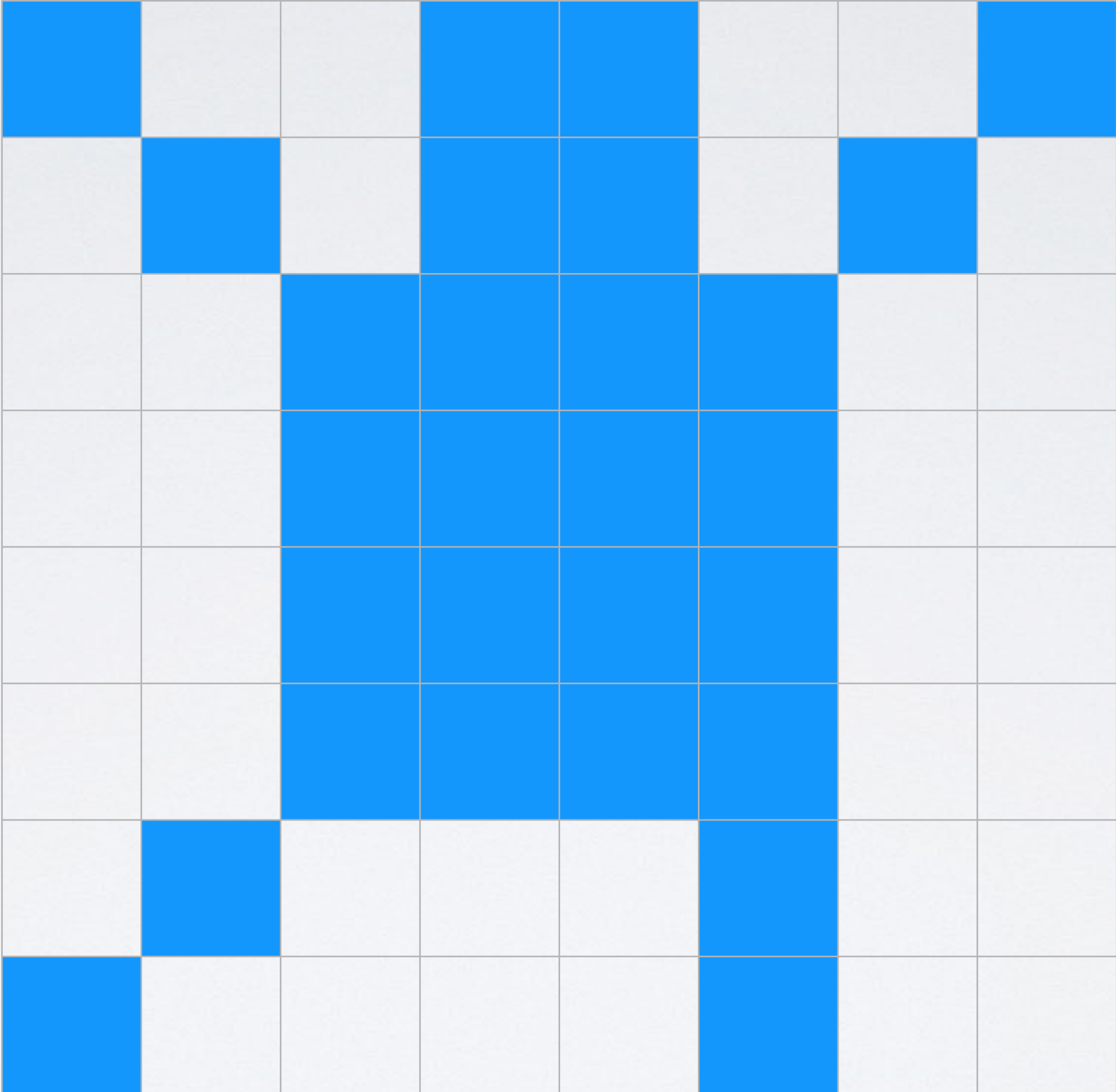
70 NEXT DELAY

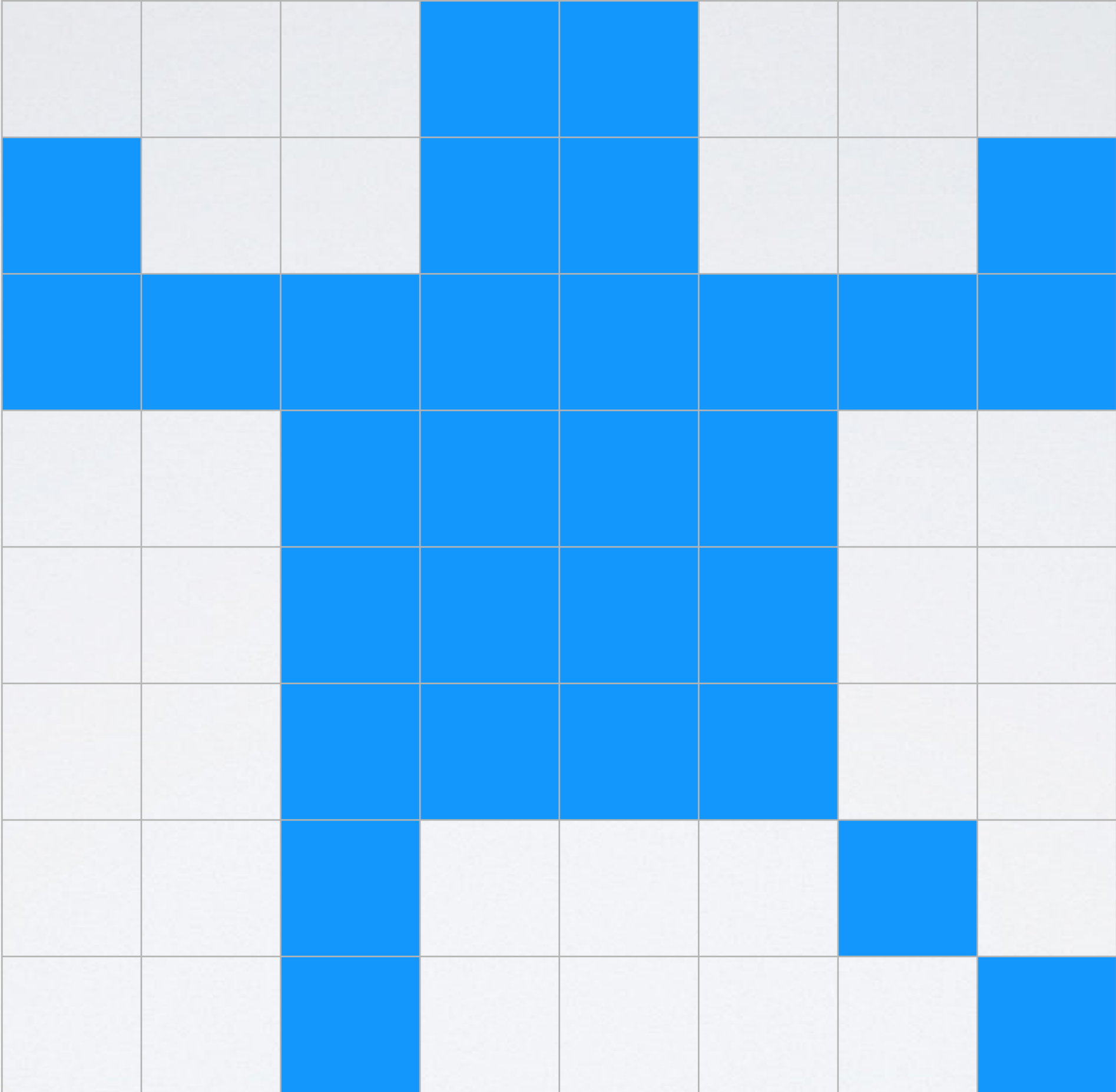
80 CALL VCHAR(12,16,129)

90 FOR DELAY=1 TO 100

100 NEXT DELAY

110 GOTO 50





LIST

10 CALL CLEAR

20 A\$="995A3C3C3C3C4484"

25 B\$="1899FF3C3C3C2221"

30 CALL CHAR(128,A\$)

35 CALL CHAR(129,B\$)

40 CALL COLOR(13,2,16)

50 CALL VCHAR(12,16,128)

60 FOR DELAY=1 TO 100

70 NEXT DELAY

80 CALL VCHAR(12,16,129)

90 FOR DELAY=1 TO 100

100 NEXT DELAY

110 GOTO 50

ANIMATIONS ARE COMMUNICATION

- Hello, friends!
- Merhaba, arkadaşlar!
- We can tell the user with text
- Or we can show the user with animation

Row 1

Row 2

Row 3

Row 4

Row 5

**New
Row!**

ANIMATIONS ARE COMMUNICATION

- Hello, friends!
- Merhaba, arkadaşlar!
- We can tell the user with text
- Or we can show the user with animation

Row 1
Row 2
Row 3
Row 4
Row 5

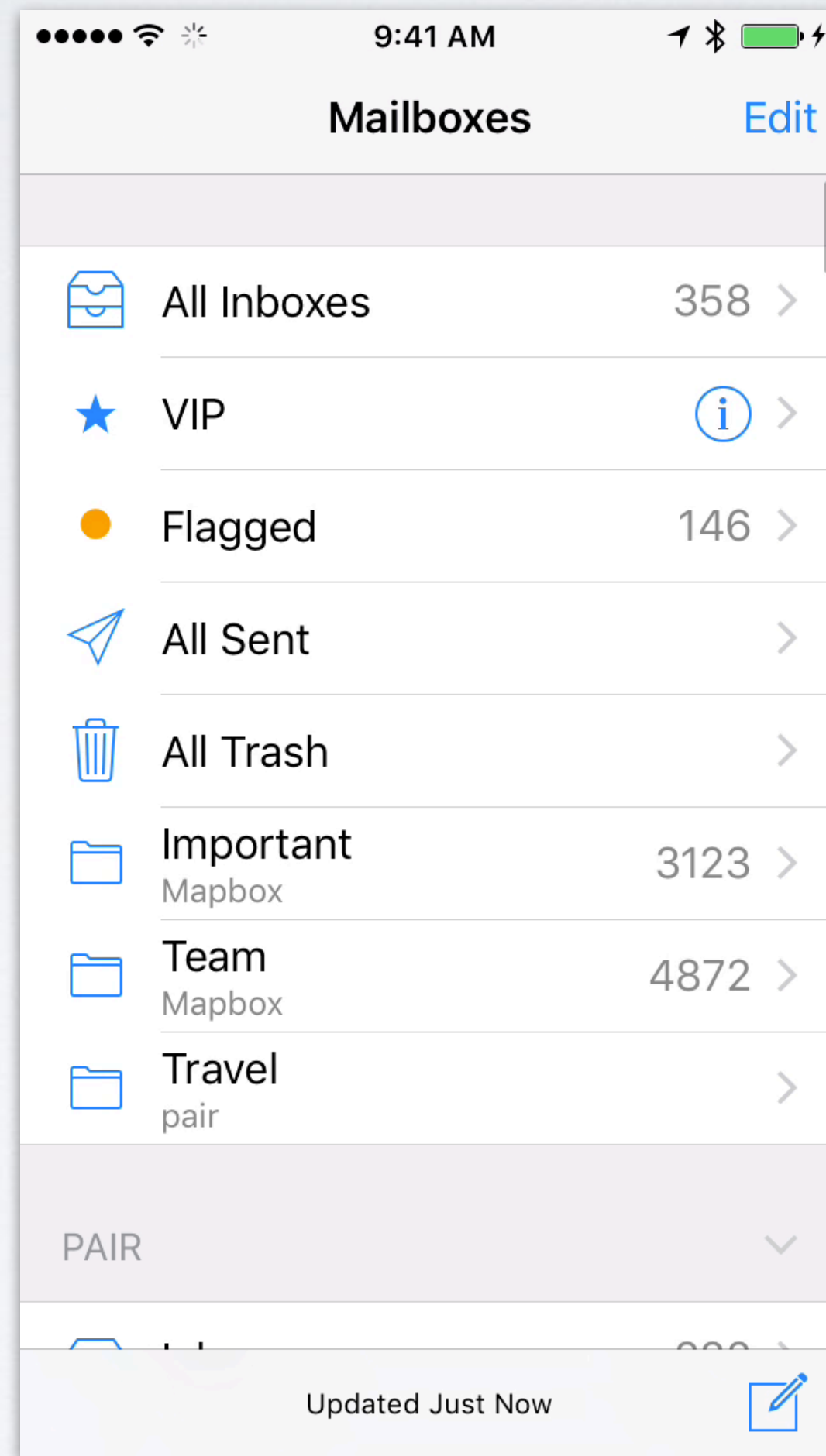
IOS ANIMATION PLATFORM

- iOS has very high animation performance and an obsession with 60FPS
- Mature animation API which debuted for macOS (OS X) and was there from the start for iOS

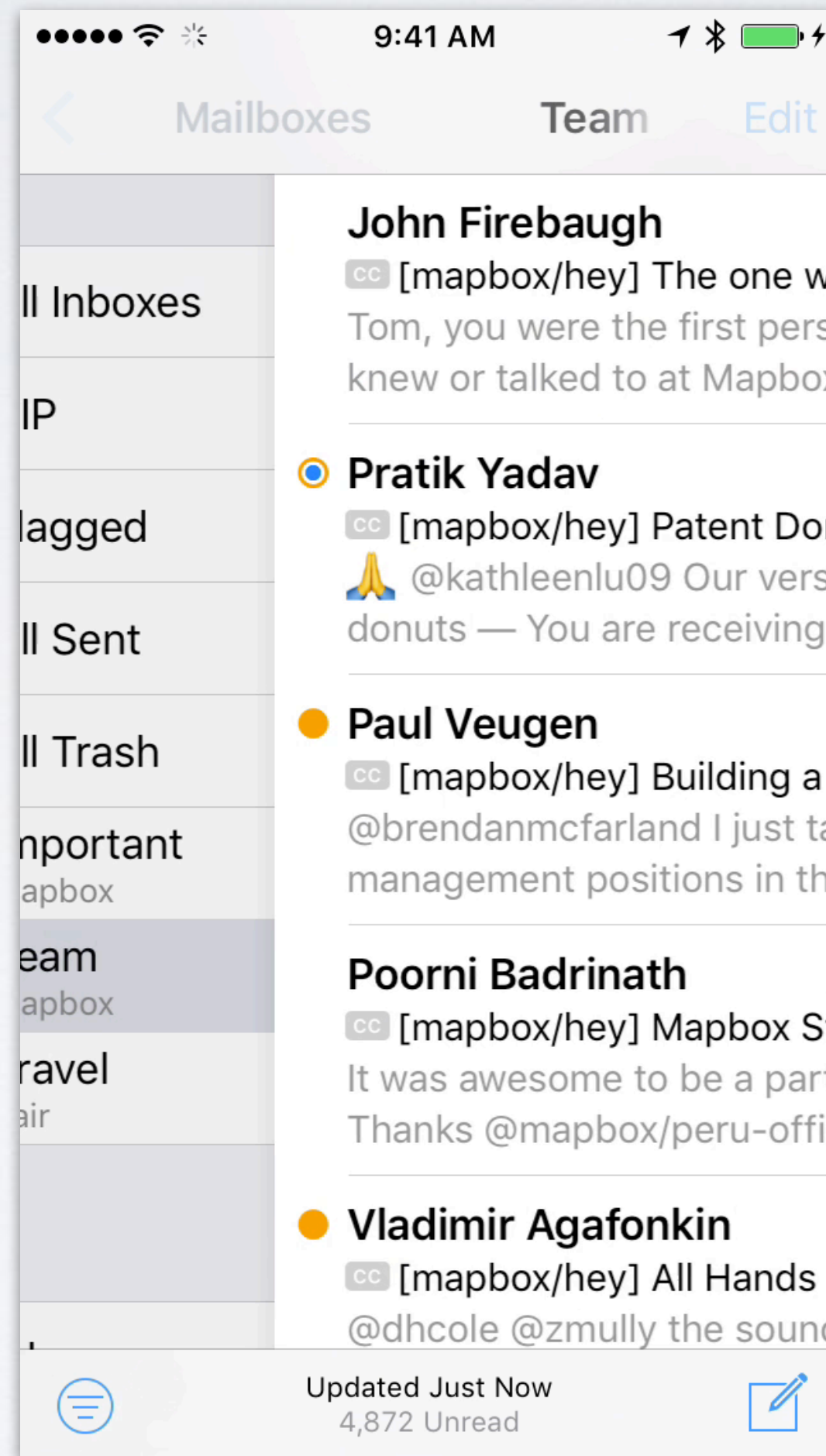
IOS ANIMATIONS ARE PERVASIVE

- You might not even notice many places that they happen
- But they are used through the base OS to give a sense of place, context, and movement

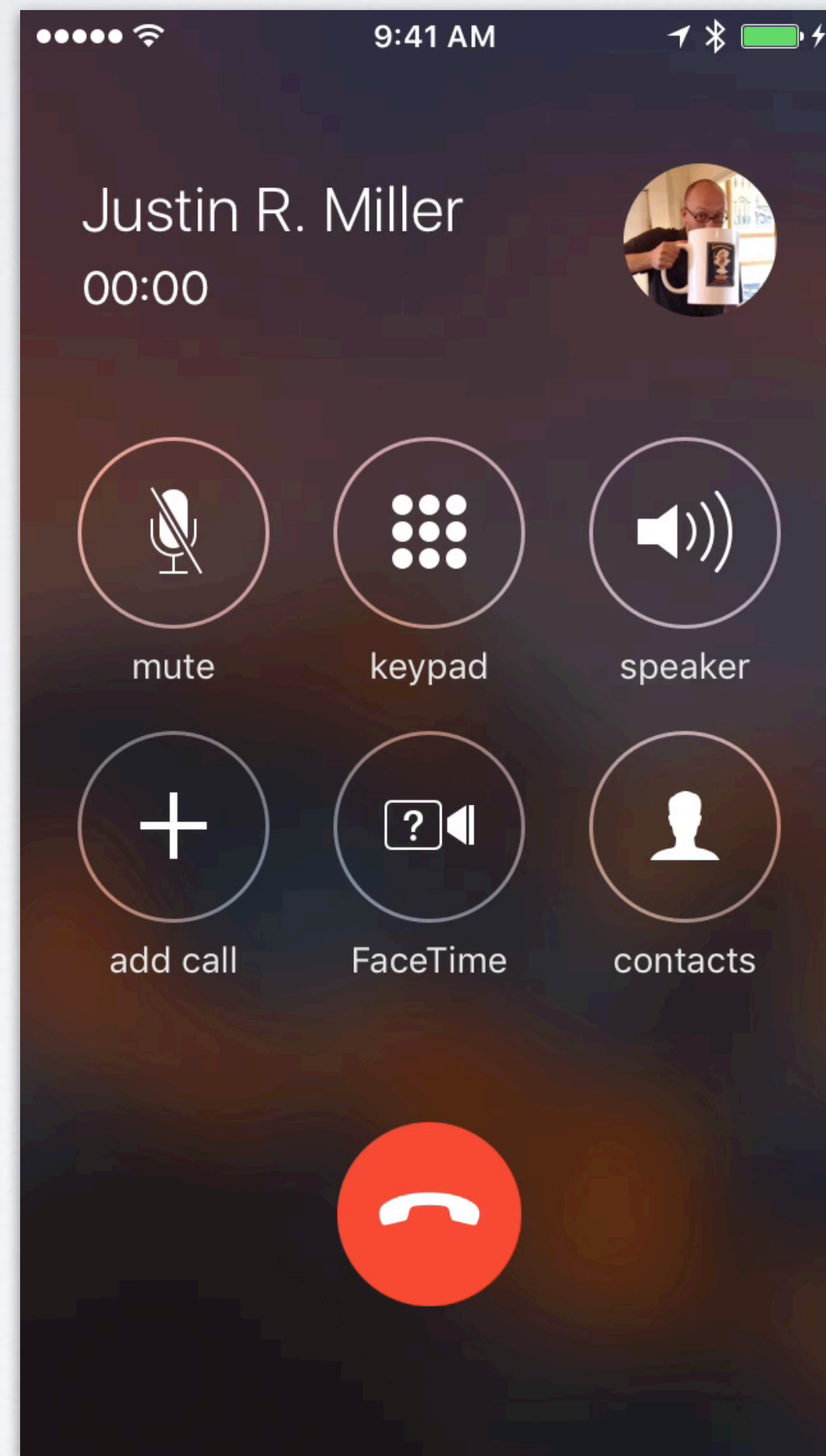
APP NAVIGATION



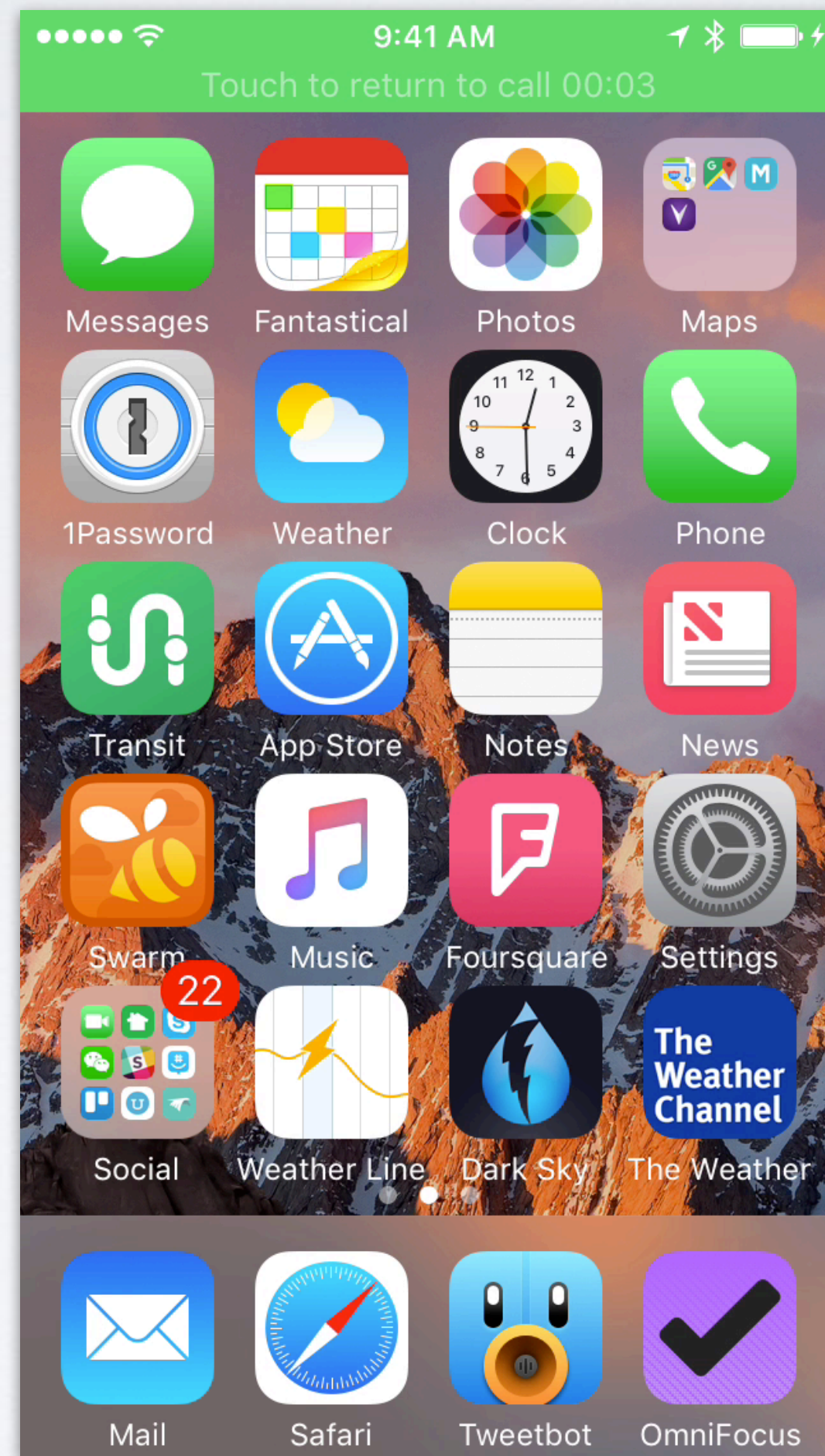
APP NAVIGATION



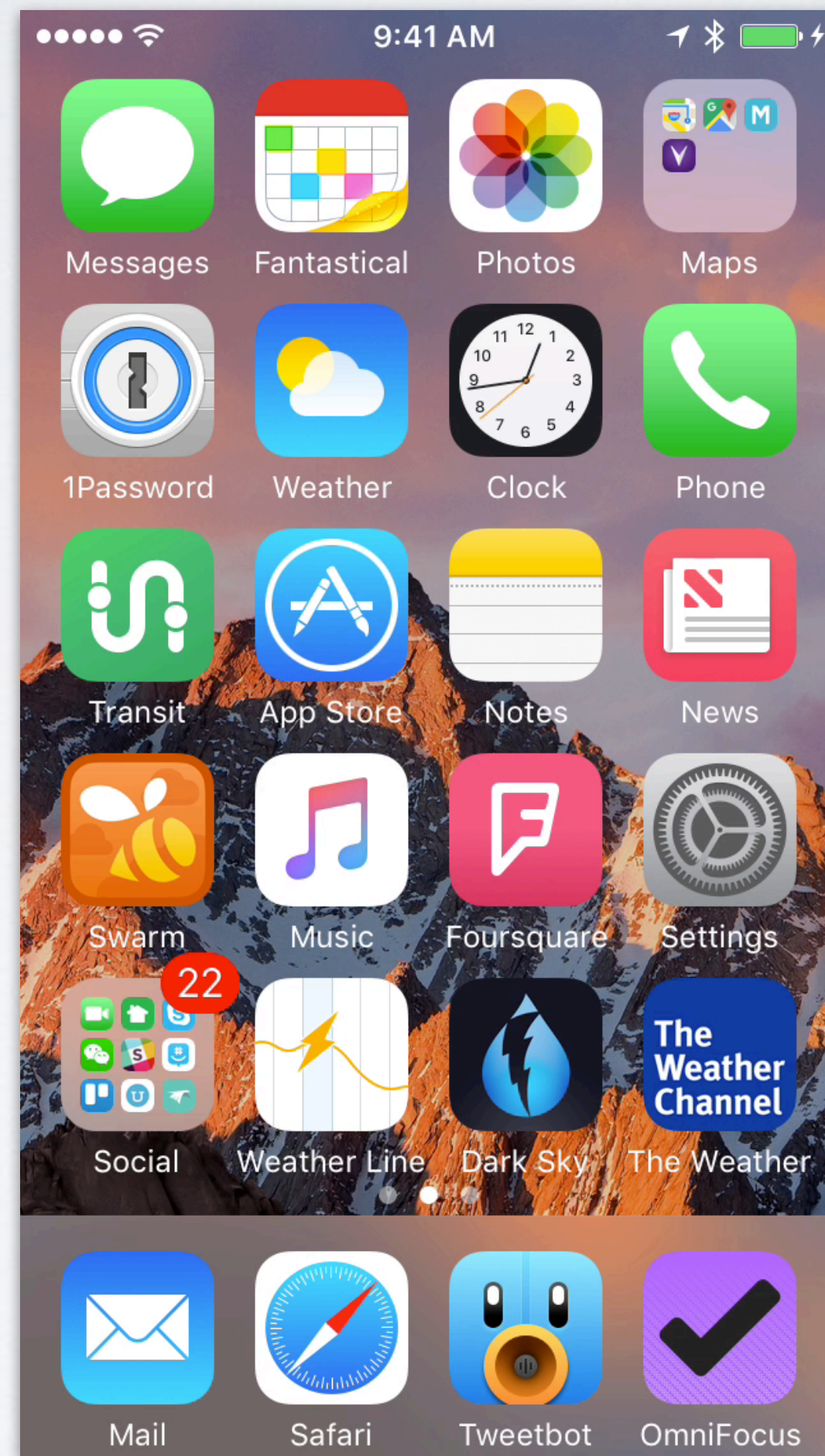
INDICATING MOTION OR ACTIVITY



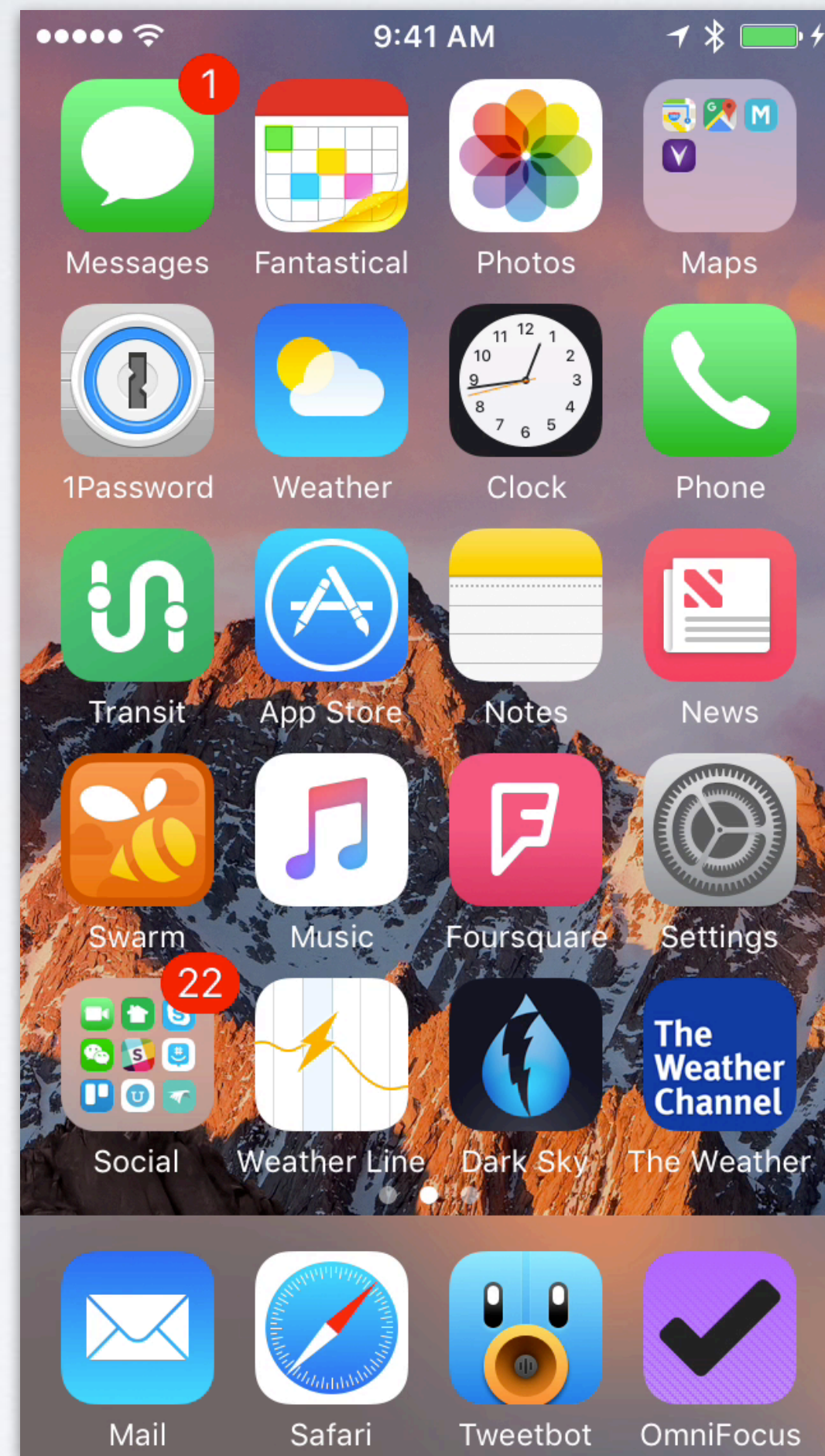
INDICATING MOTION OR ACTIVITY



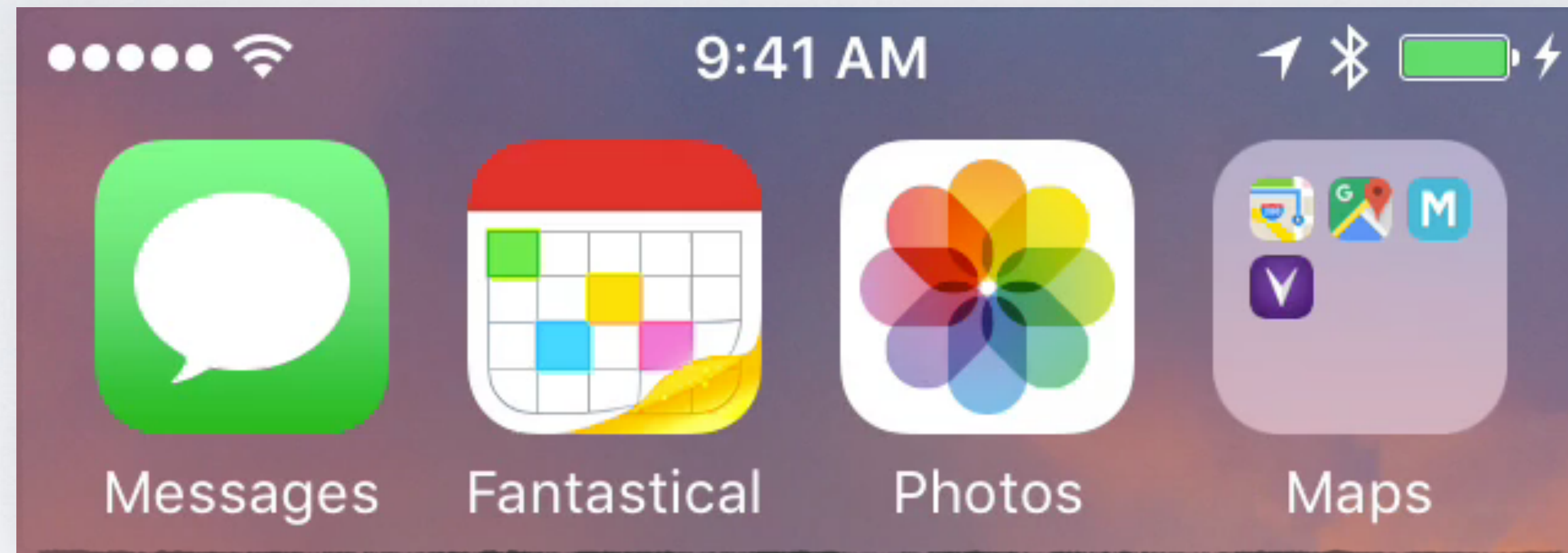
SOFTENING ROUTINES



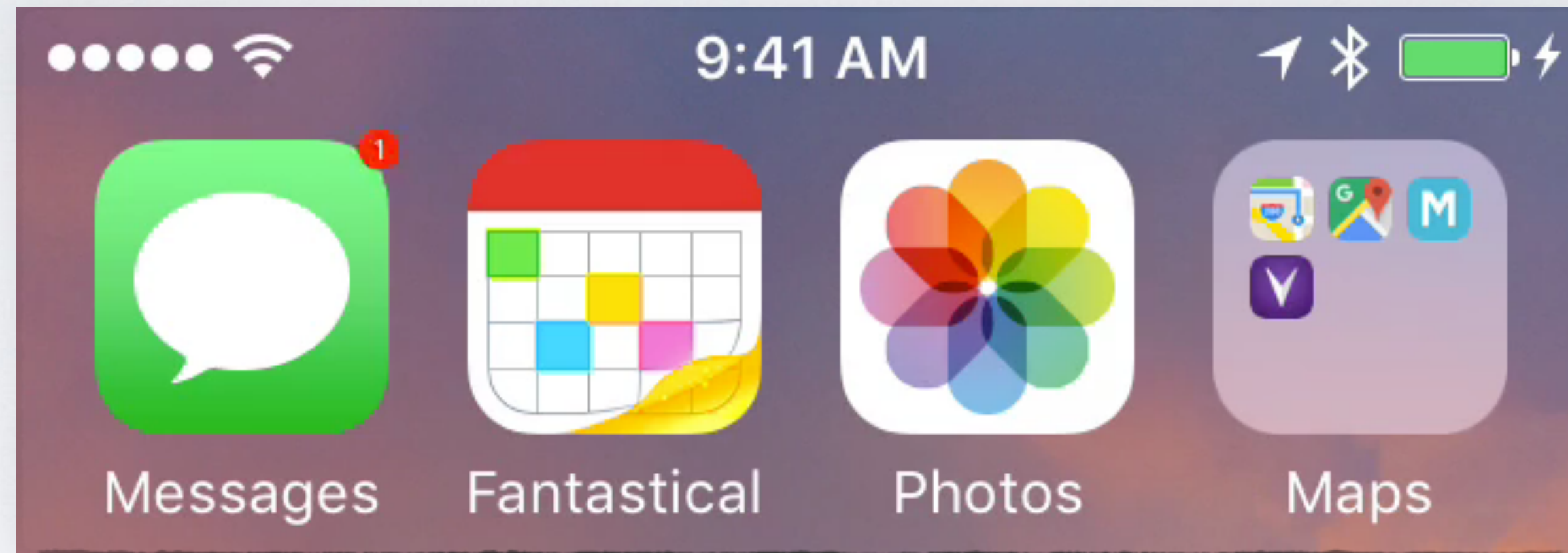
SOFTENING ROUTINES



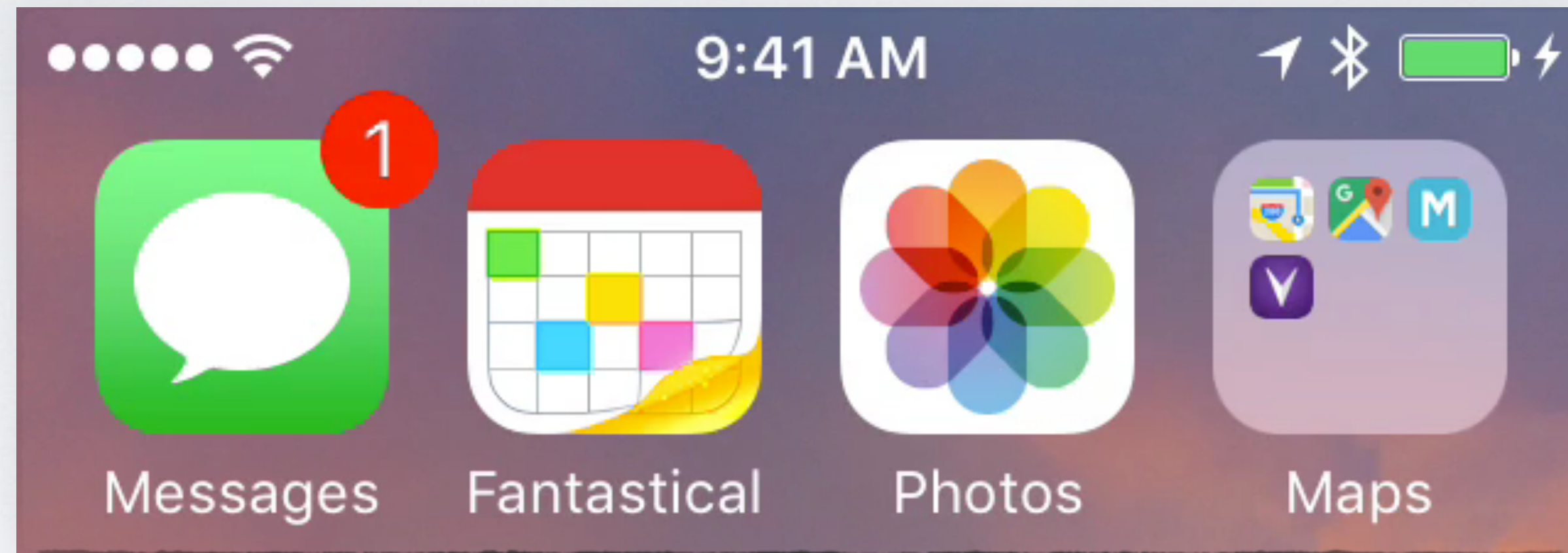
SOFTENING ROUTINES



SOFTENING ROUTINES



SOFTENING ROUTINES



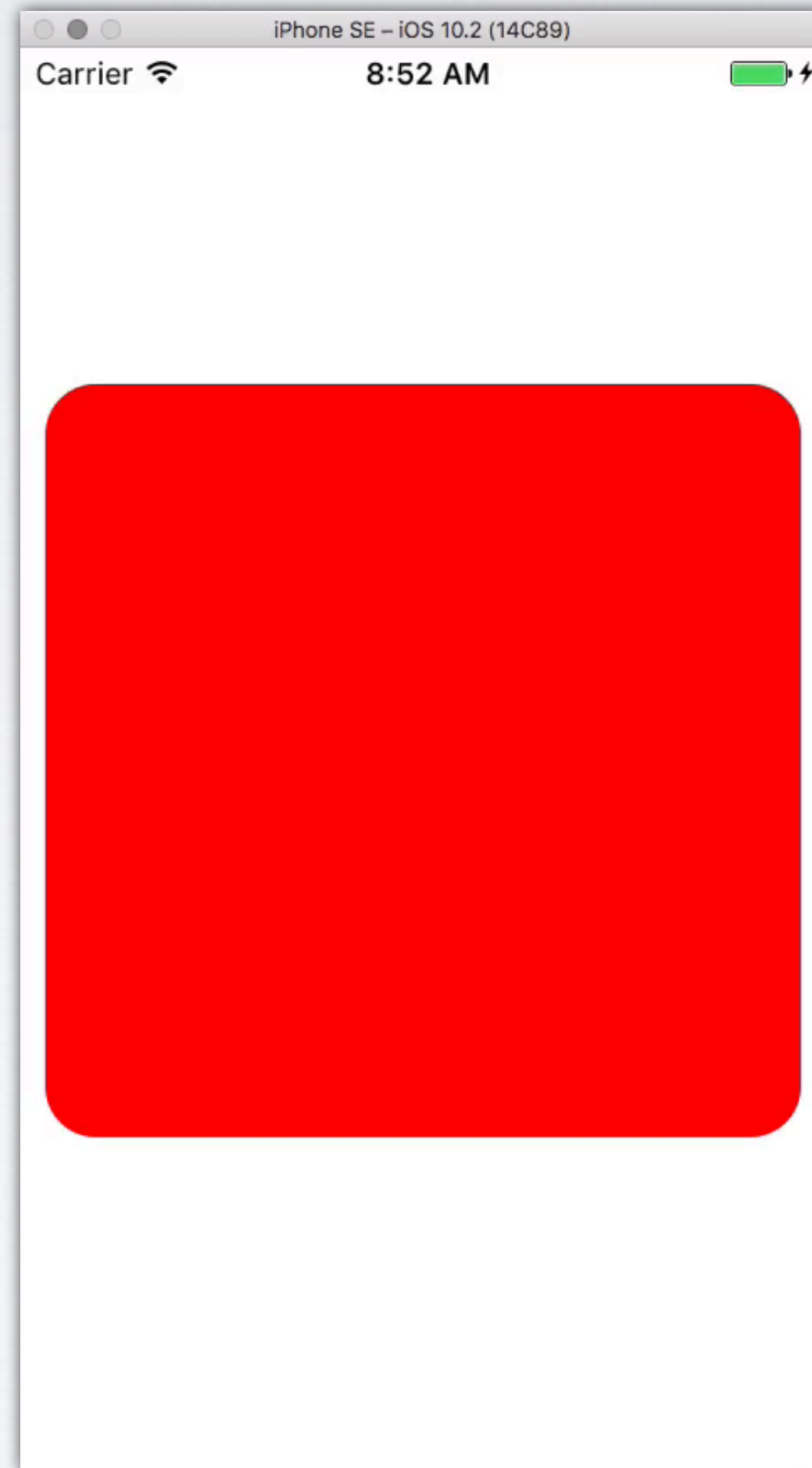
CORE ANIMATION

- Introduced in OS X 10.5 (“Leopard”, 2007) via (then-secret) iPhone team
- Implicit animation model
 - Don't have to build animation objects
 - Interpolation is handled automatically

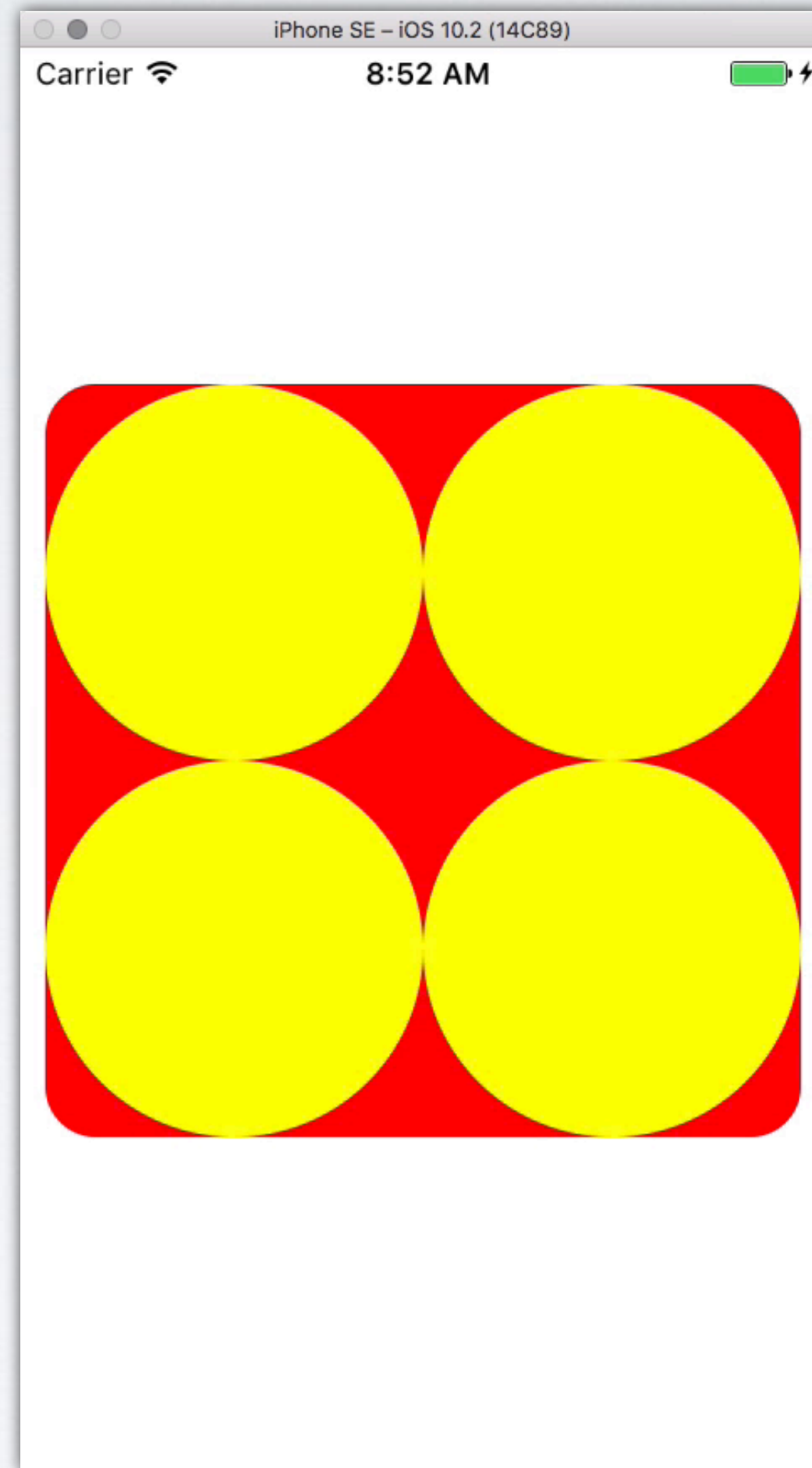
LET'S SEE HOW IT WORKS

- Visual building block (the view) exists on screen
- A view can contain anything
- Views are backed by **layers**, which are their bitmap representations—a sort of snapshot of their contents
- Core Animation animates layer property changes by default

FADE DEMO



FADE DEMO



THE CODE

```
let sublayer = CALayer()  
sublayer.frame = CGRect(x: x, y: y, width: size, height: size)  
sublayer.backgroundColor = UIColor.yellow.cgColor  
sublayer.cornerRadius = size / 2  
sublayer.opacity = 0  
view.layer.addSublayer(sublayer)
```

```
layer.opacity = (layer.opacity == 1 ? 0 : 1)
```

THAT'S IT?

LESSON # 1: DISCOVERABILITY

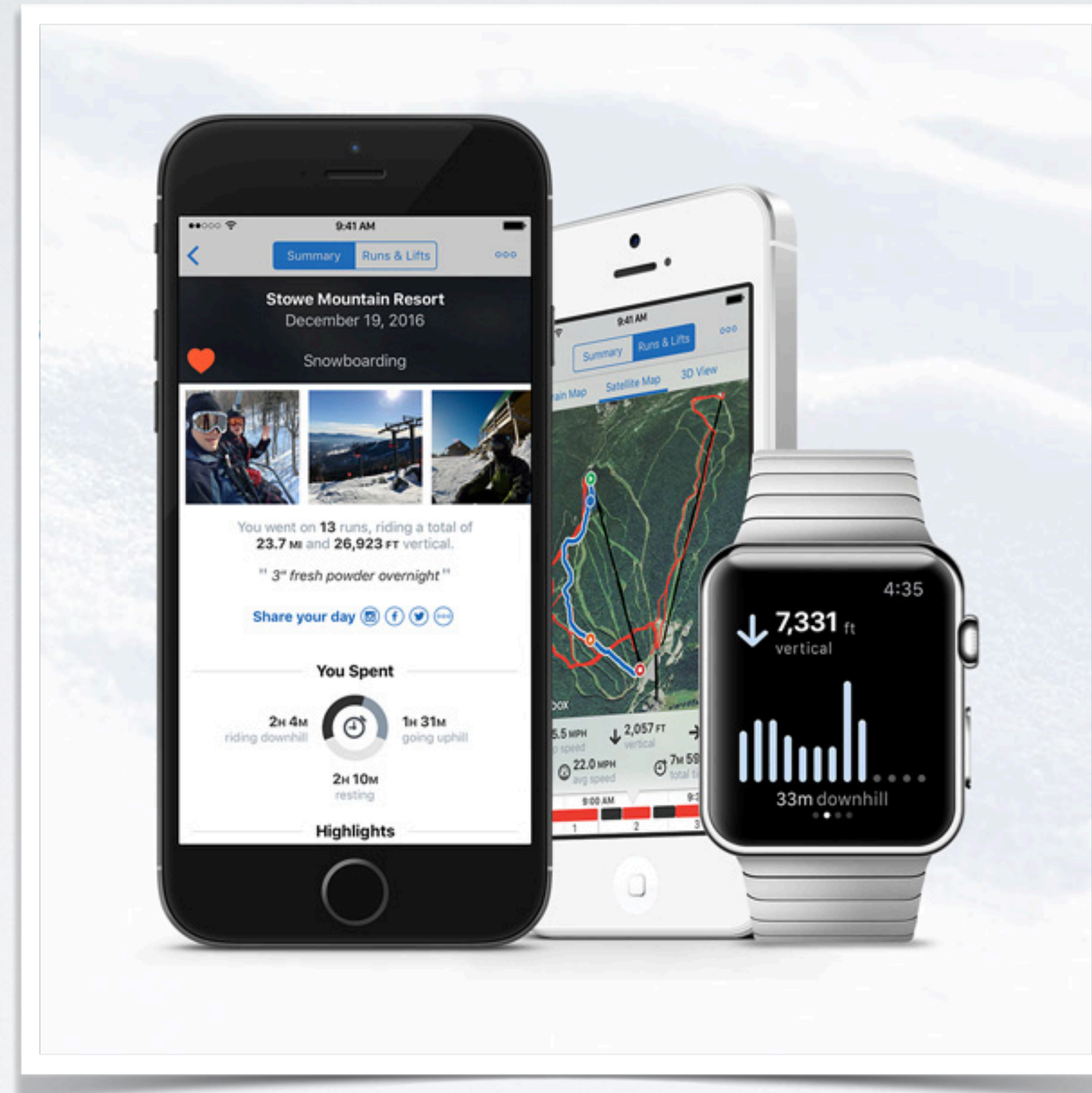
ENHANCING DISCOVERABILITY

- Piggyback on things you are doing in nearby APIs
(here, layer property changes)
- Consider opting in to a behavior by default

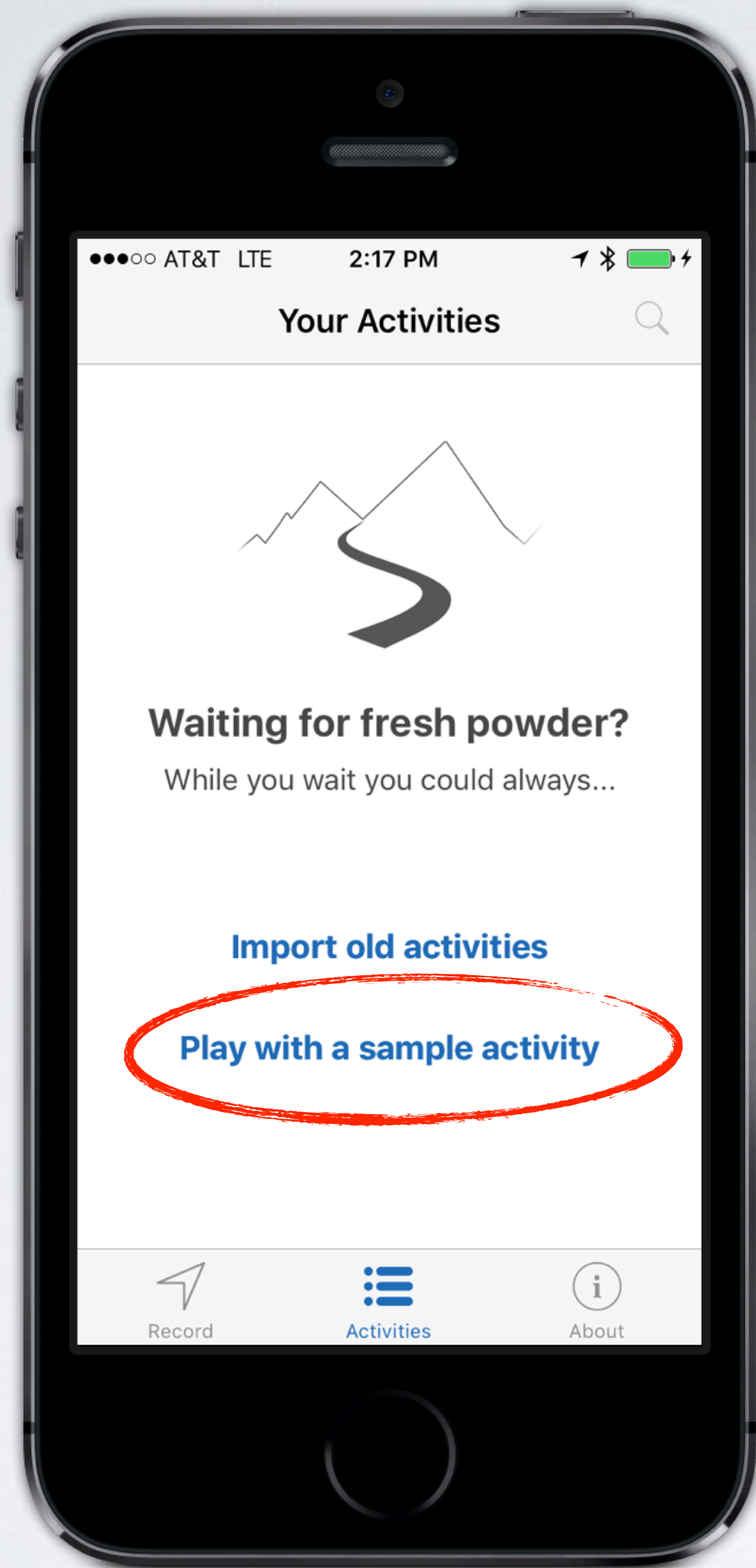
ENHANCING DISCOVERABILITY

- Build SDKs?
 - Database initial data
 - View default background color
 - First run demo
- Build apps?
 - Default populated data
 - Partially-hidden content to encourage gestures

SLOPES



getslopes.com



●●●● AT&T LTE 2:17 PM 🔍 🔋

Your Activities






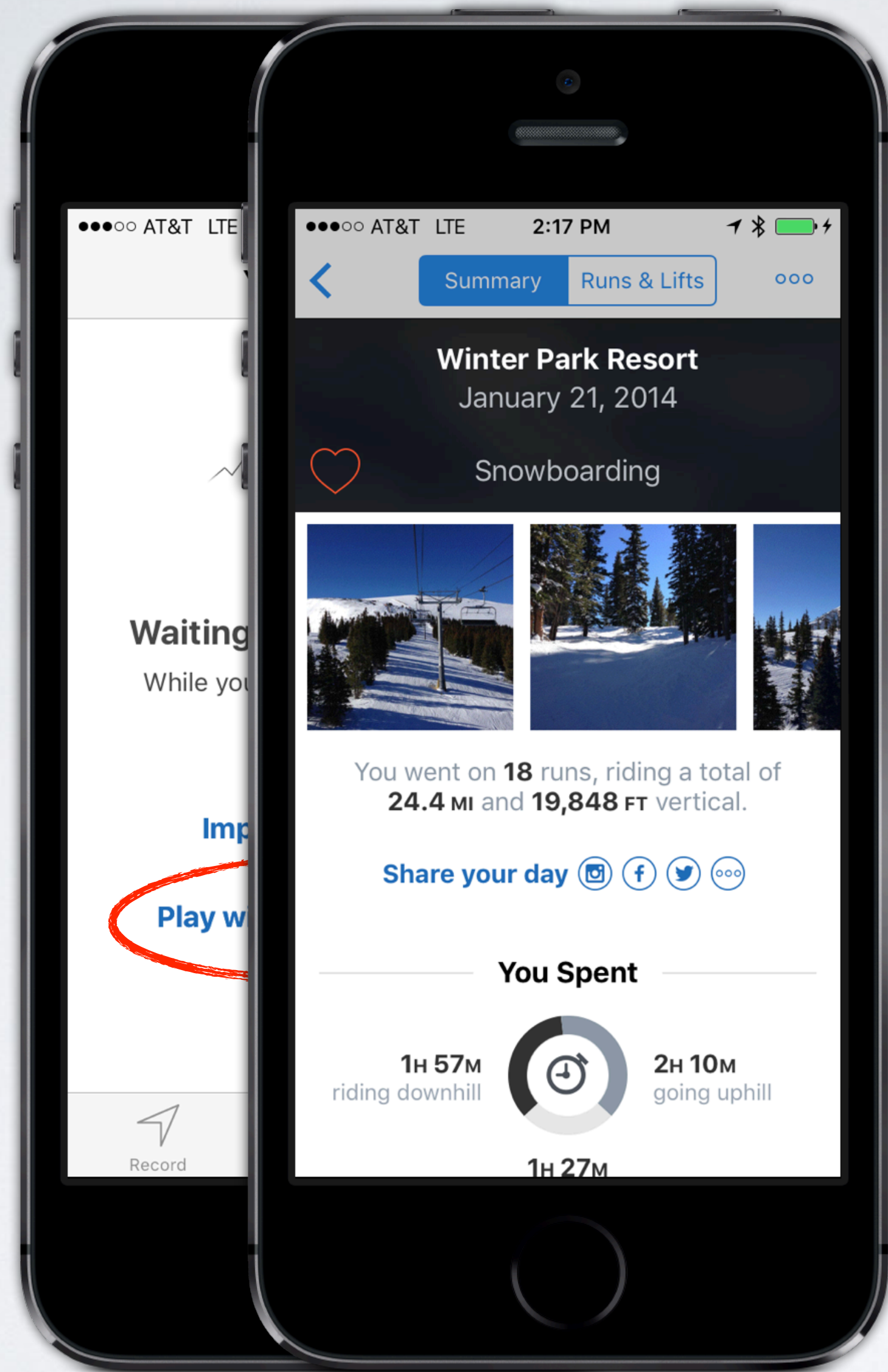
Waiting for fresh powder?

While you wait you could always...

Import old activities

Play with a sample activity

-  Record
-  Activities
-  About



AT&T LTE

AT&T LTE 2:17 PM

Summary Runs & Lifts

Winter Park Resort

January 21, 2014

Heart icon Snowboarding

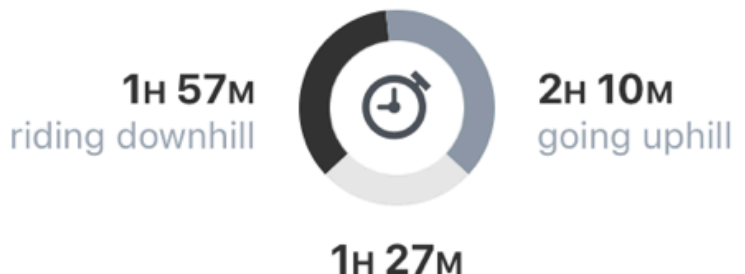


You went on 18 runs, riding a total of 24.4 MI and 19,848 FT vertical.

Share your day



You Spent



Waiting

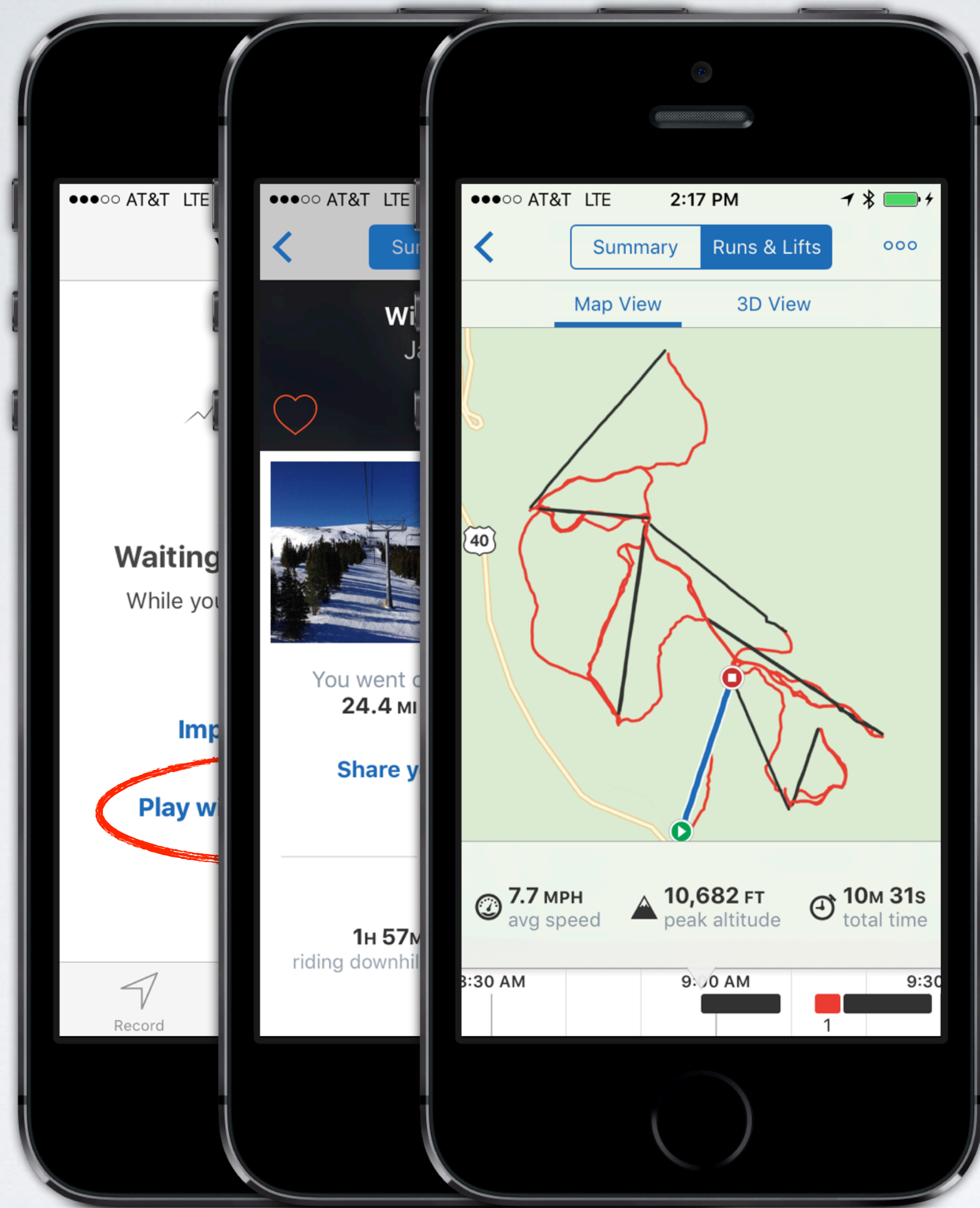
While you

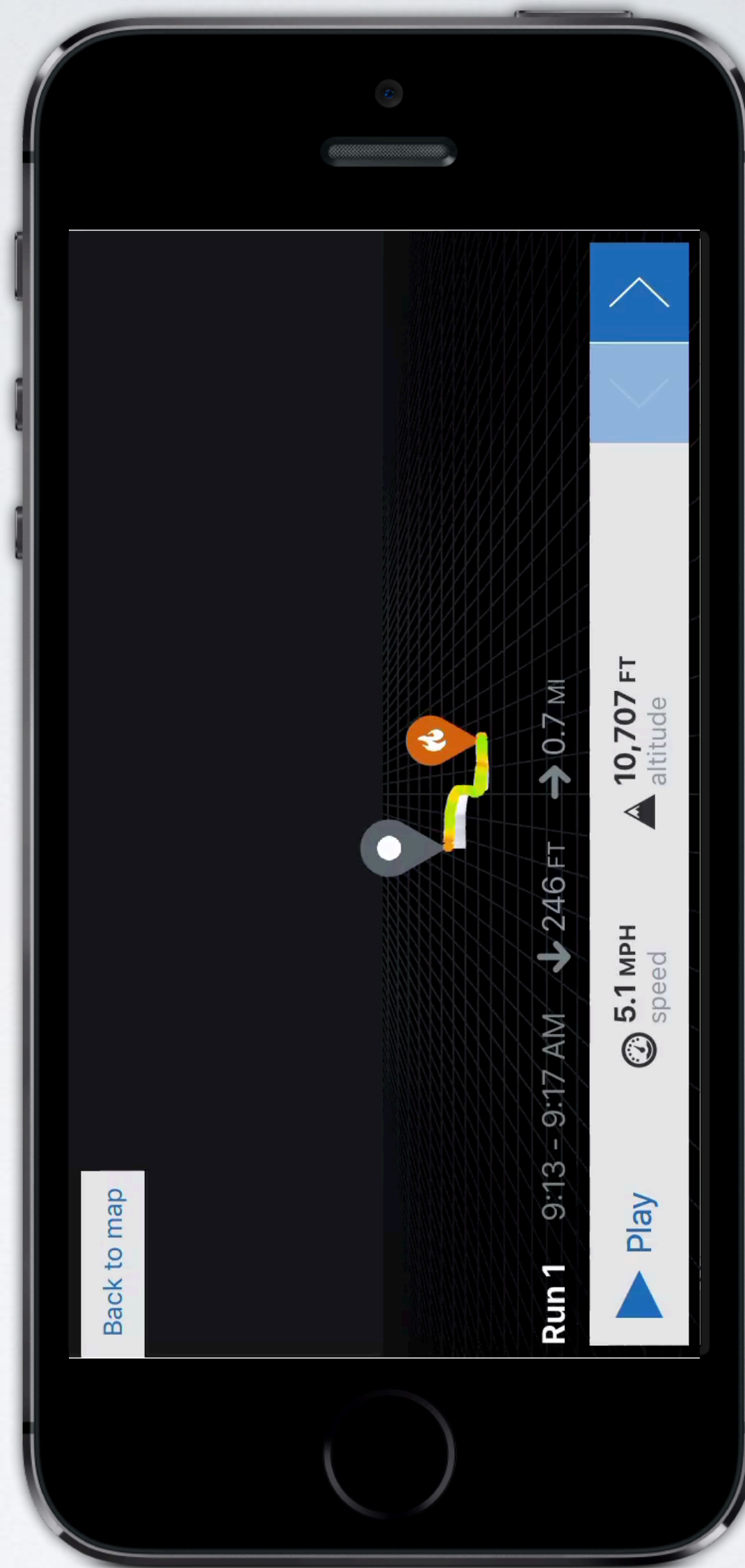
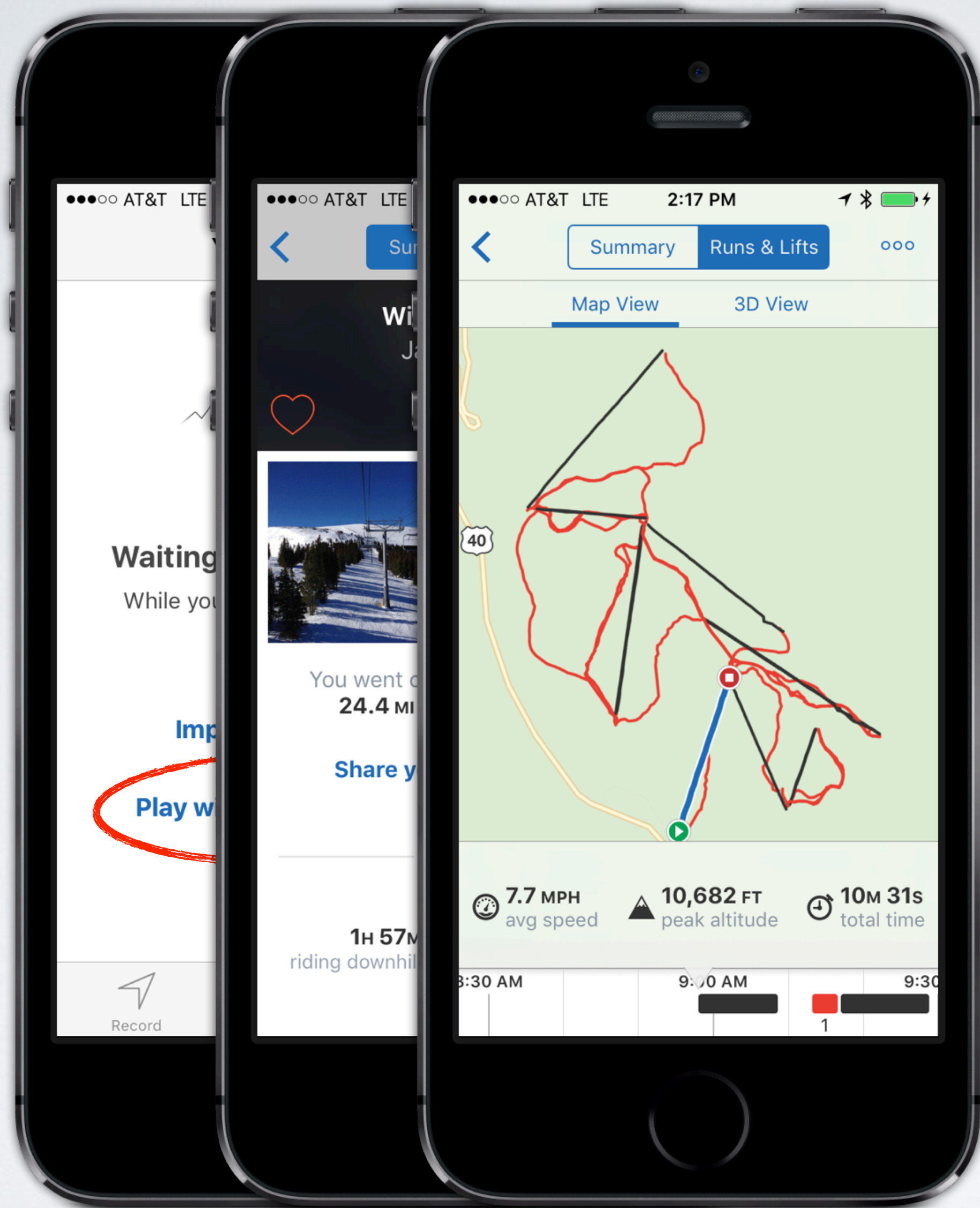
Imp

Play w



Record





Back to map

Run 1 9:13 - 9:17 AM ↓ 246 FT → 0.7 MI

▶ Play

🕒 5.1 MPH
speed

⚓ 10,707 FT
altitude



Back to map

↓ 792 FT → 0.9 MI

|| Pause

🕒 26.3 MPH
speed

⚓ 10,907 FT
altitude



THERE IS A **LOT** MORE!

- Explicit animations
- Complex animation building blocks
- Keyframe animations
- Custom properties
- Per-property runtime checks for actions
- Replicated layers
- Nested transactions
- Transitions between views and layers

LESSON #2: FLEXIBILITY

FLEXIBILITY

- Allows for uses beyond the primary use case
- But doesn't bog down the primary use case

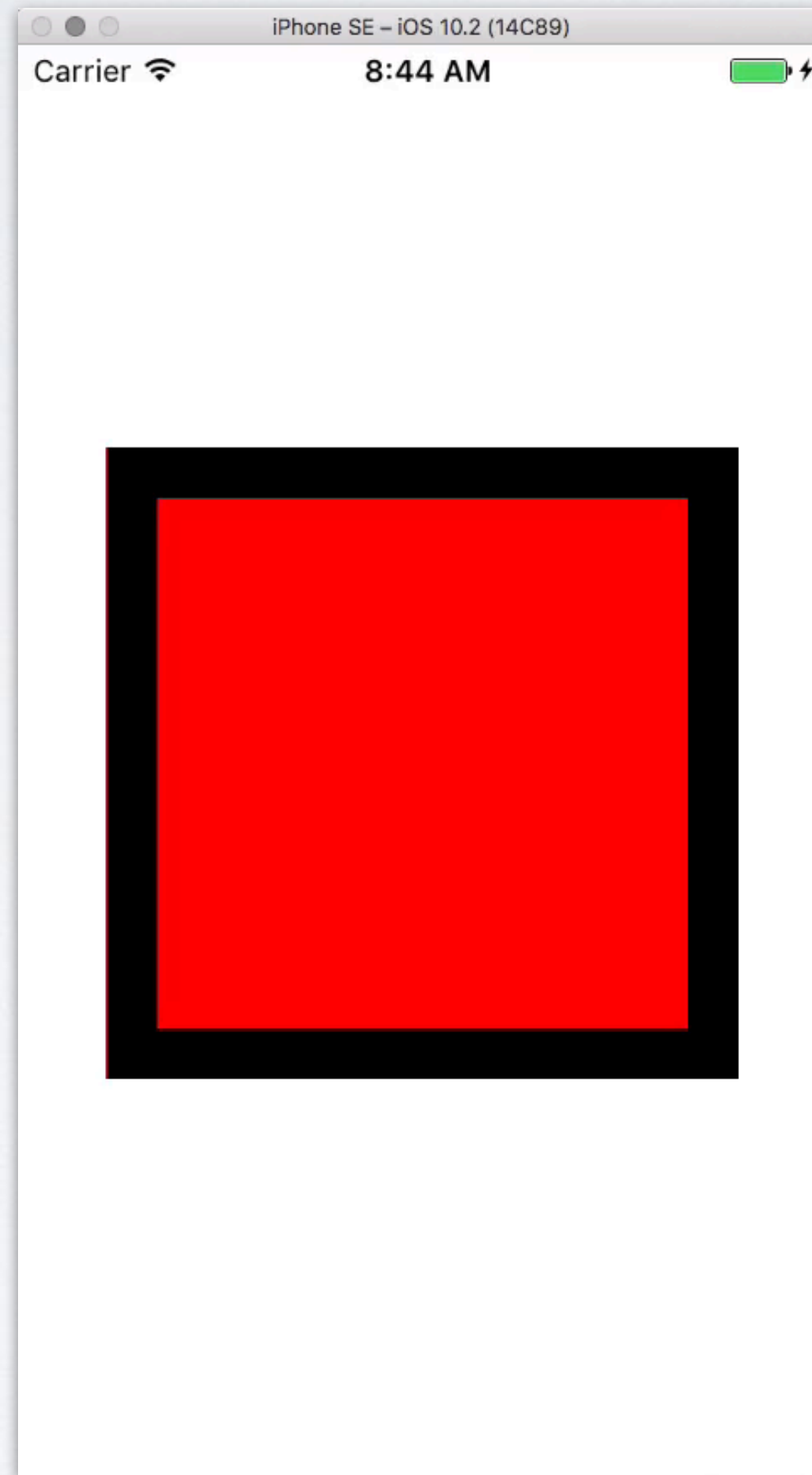
EXPLICIT ANIMATIONS

- Instead of just setting property values, you construct animation objects
- Able to customize parameters on individual animations

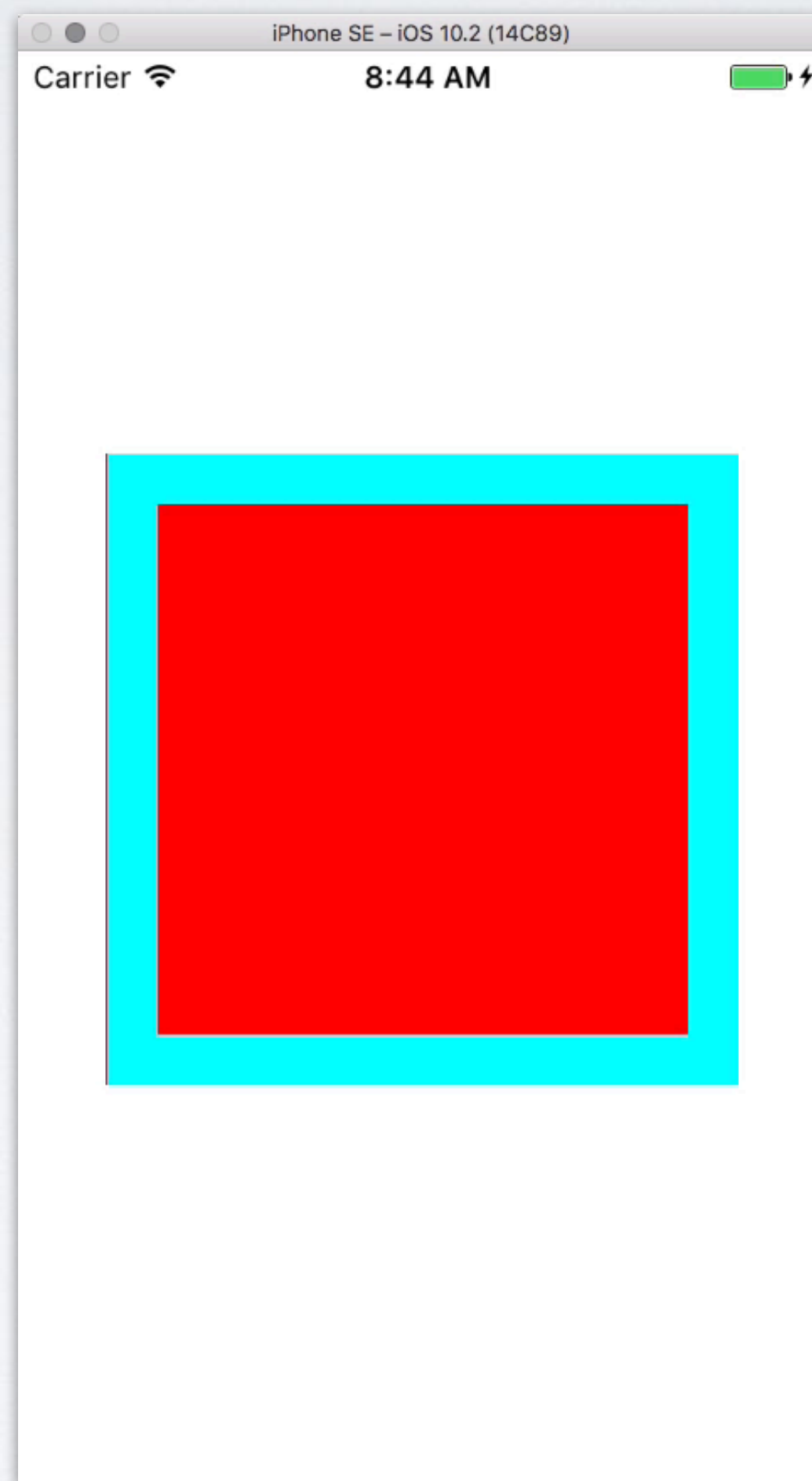
EXPLICIT ANIMATIONS

```
let animation = CABasicAnimation(keyPath: "borderColor")
animation.duration = 2
animation.timingFunction = CAMediaTimingFunction(
    name: kCAMediaTimingFunctionLinear
)
animation.fromValue = layer.borderColor
animation.toValue = UIColor.cyan.cgColor
animation.isRemovedOnCompletion = false
animation.fillMode = kCAFillModeForwards
layer.add(animation, forKey: "borderColor")
```

EXPLICIT ANIMATIONS



EXPLICIT ANIMATIONS



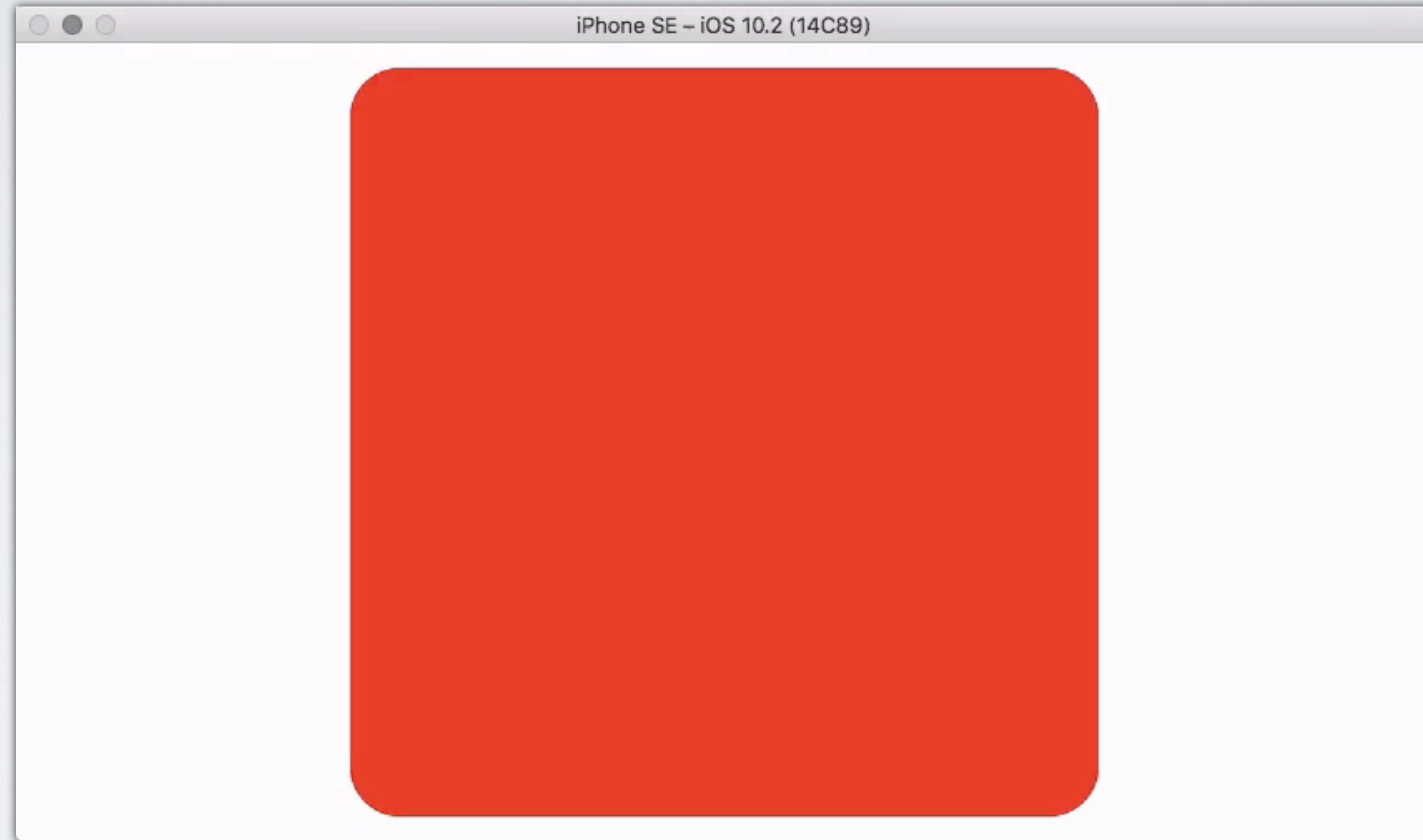
TRANSACTIONS

- Allow you to specify custom animation parameters
- Allow you to precisely control time, acceleration, and completion actions across multiple animations

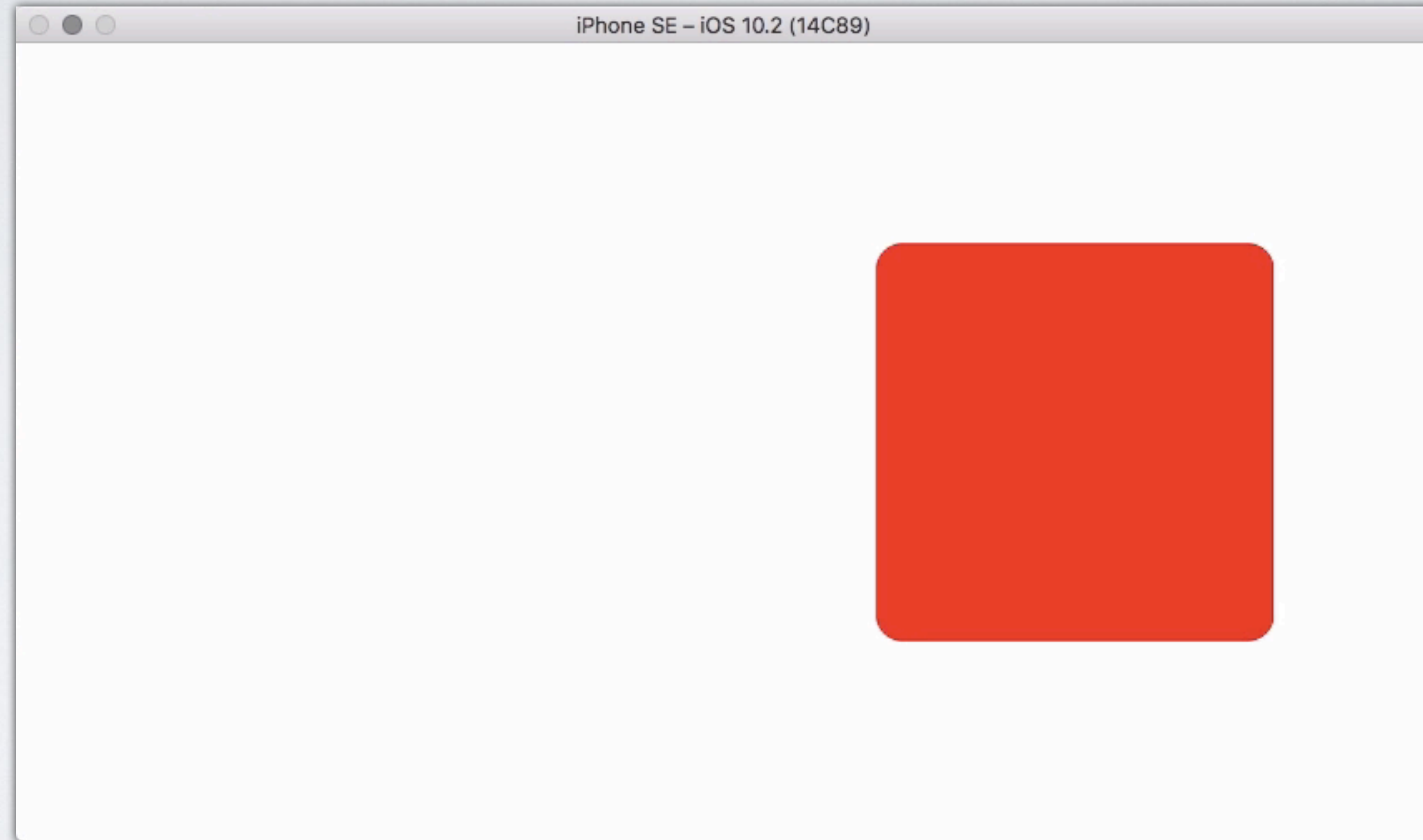
TRANSACTION EXAMPLE

```
CATransaction.begin()  
CATransaction.setAnimationDuration(0.1)  
CATransaction.setAnimationTimingFunction(CAMediaTimingFunction(name:  
    kCAMediaTimingFunctionEaseInEaseOut))  
  
layer.position = CGPoint(x: layer.position.x + 150, y: layer.position.y)  
layer.transform = CATransform3DMakeScale(0.5, 0.5, 1)  
  
CATransaction.commit()
```

TRANSACTION EXAMPLE



TRANSACTION EXAMPLE

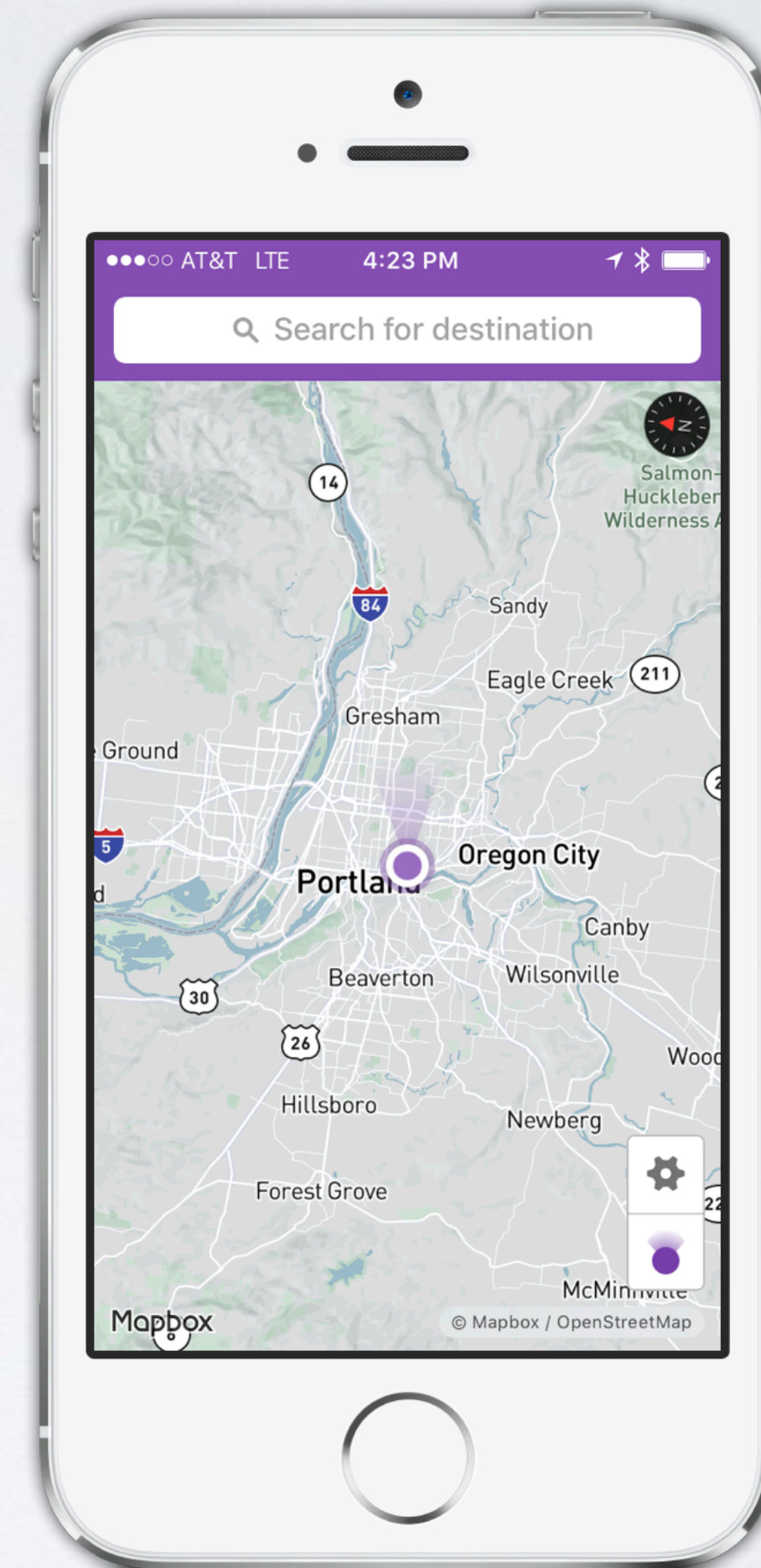
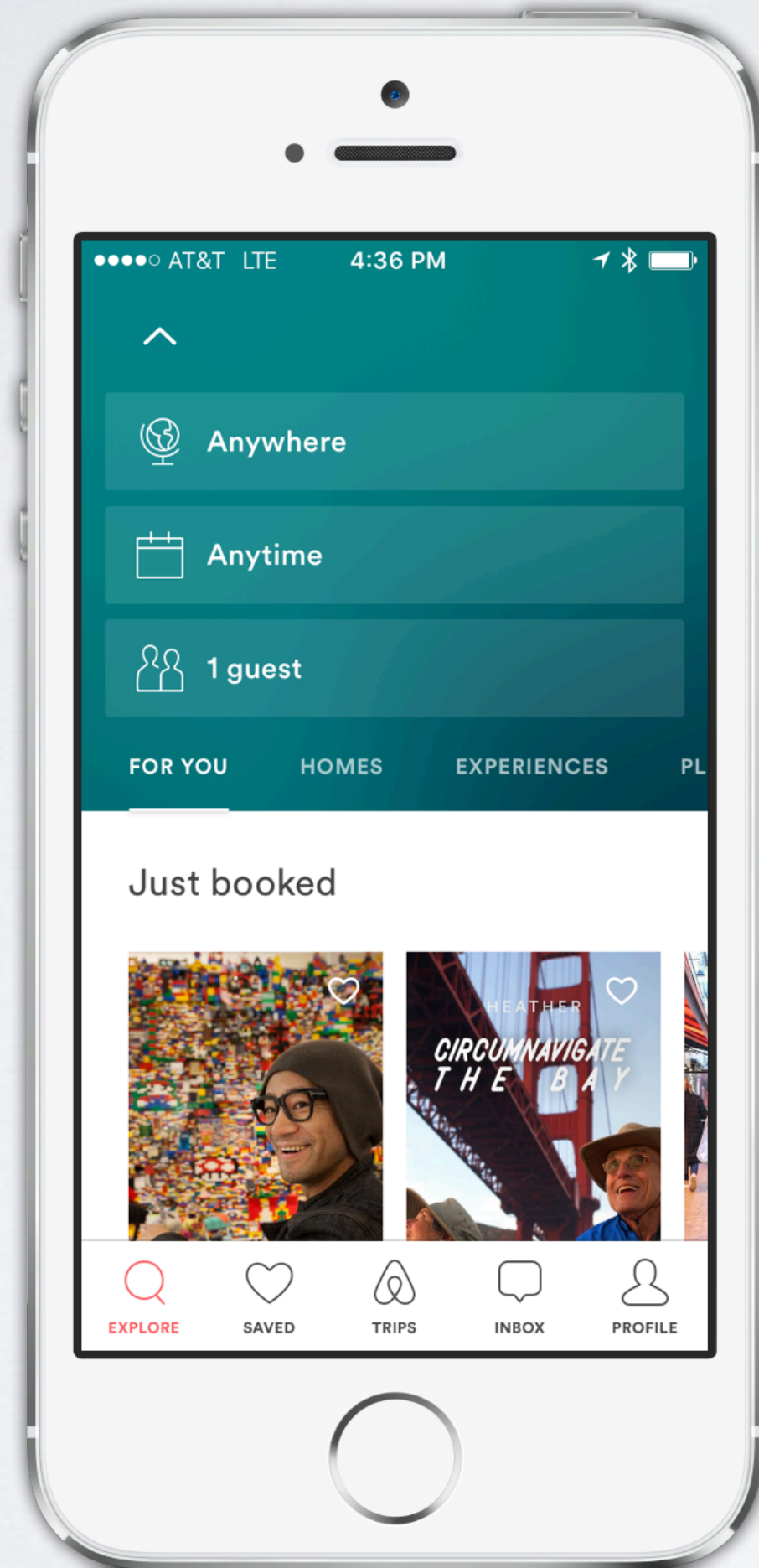


LESSON #3: MAKE IT INTUITIVE

TRANSACTION EXAMPLE REVISITED

```
CATransaction.begin()  
CATransaction.setAnimationDuration(0.1)  
CATransaction.setAnimationTimingFunction(CAMediaTimingFunction(name:  
    kCAMediaTimingFunctionEaseInEaseOut))  
  
layer.position = CGPoint(x: layer.position.x + 150, y: layer.position.y)  
layer.transform = CATransform3DMakeScale(0.5, 0.5, 1)  
  
CATransaction.commit()
```

GRADIENTS



GRADIENTS

```
let layer = CAGradientLayer()  
layer.colors = [UIColor.red.cgColor, UIColor.purple.cgColor]
```

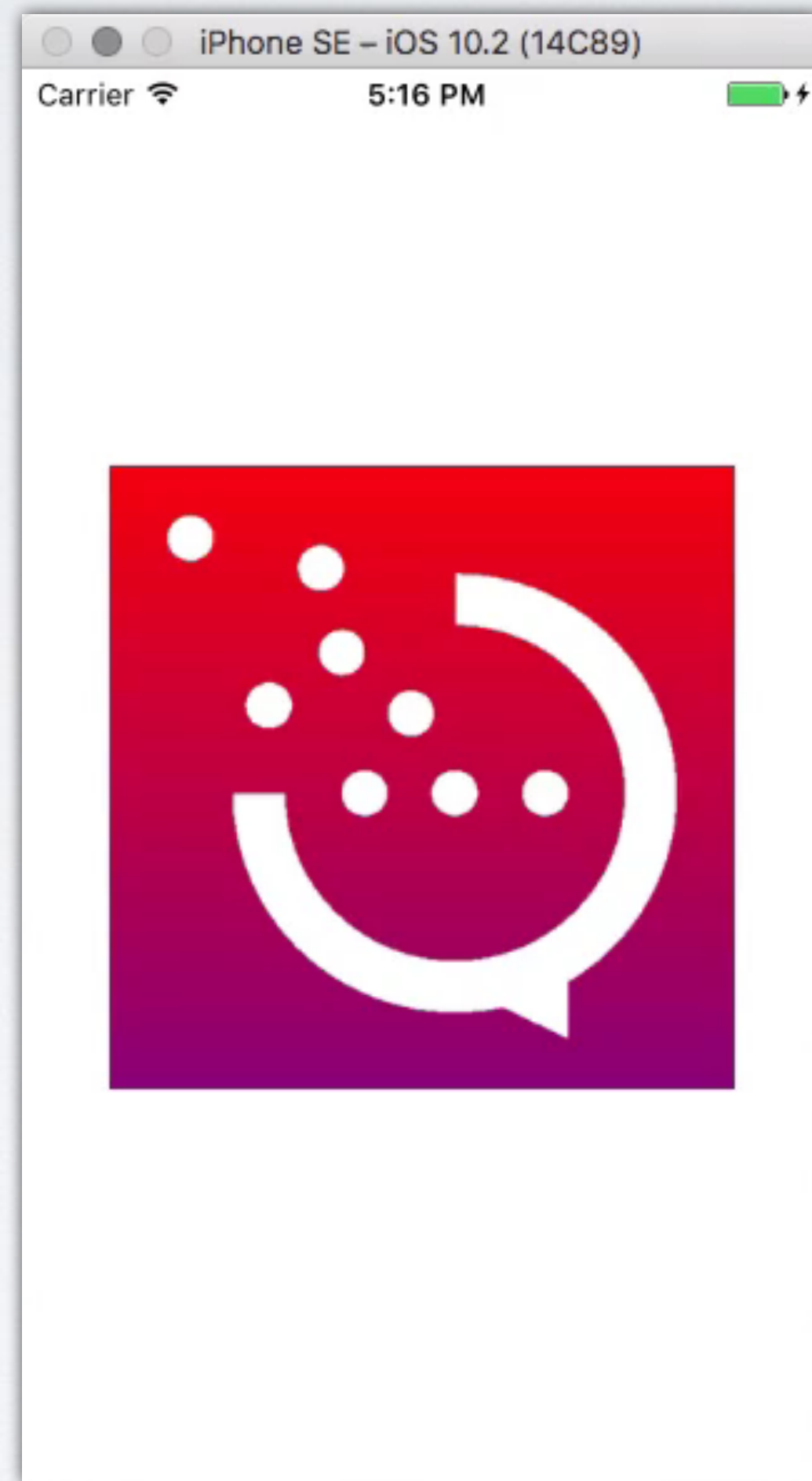

MASKS



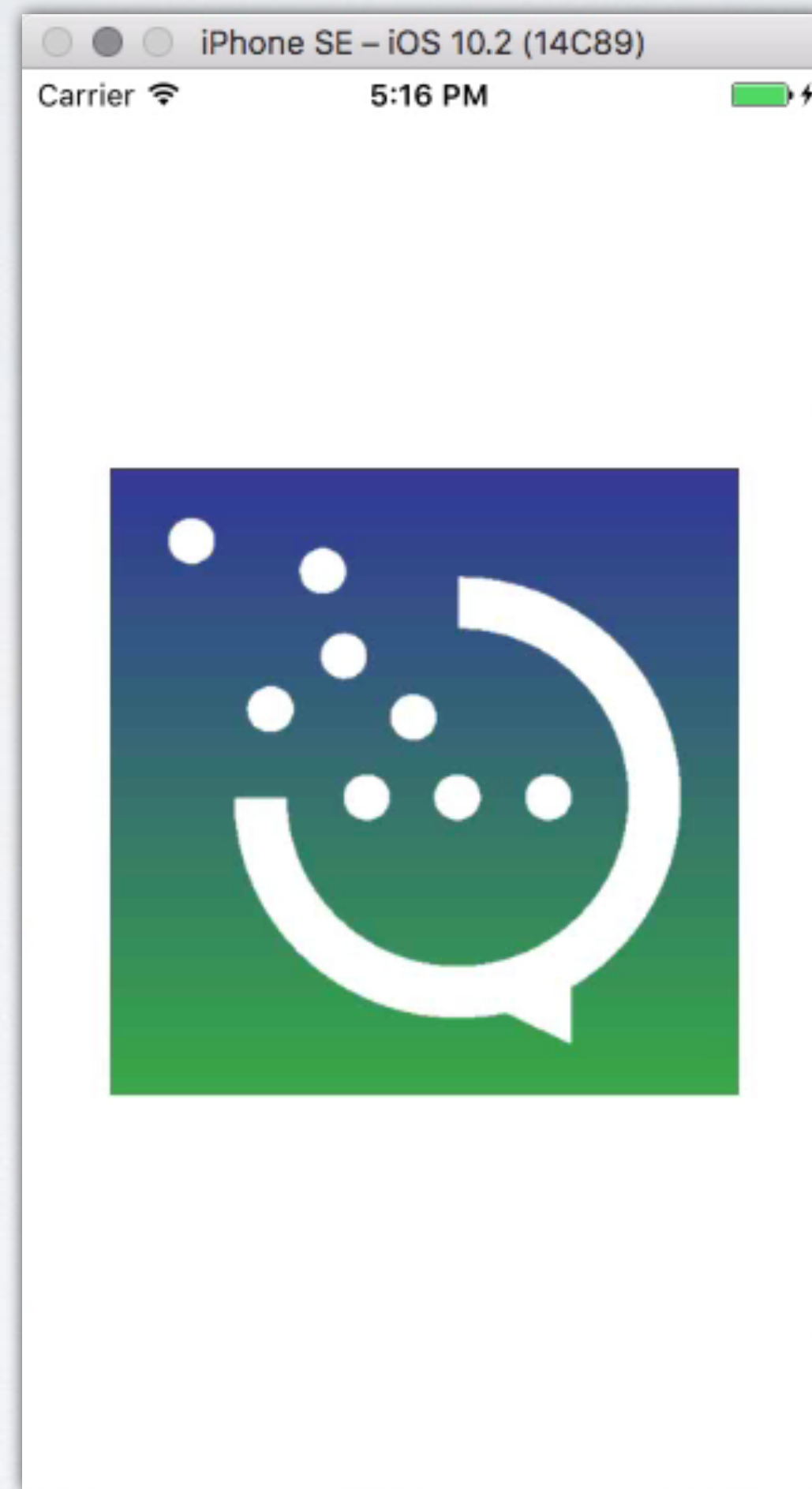
MASKS

```
let logo = CALayer()  
logo.contents = UIImage(named: "itt.png")!.cgImage  
layer.mask = logo
```

GRADIENT & MASK COMBINED



GRADIENT & MASK COMBINED

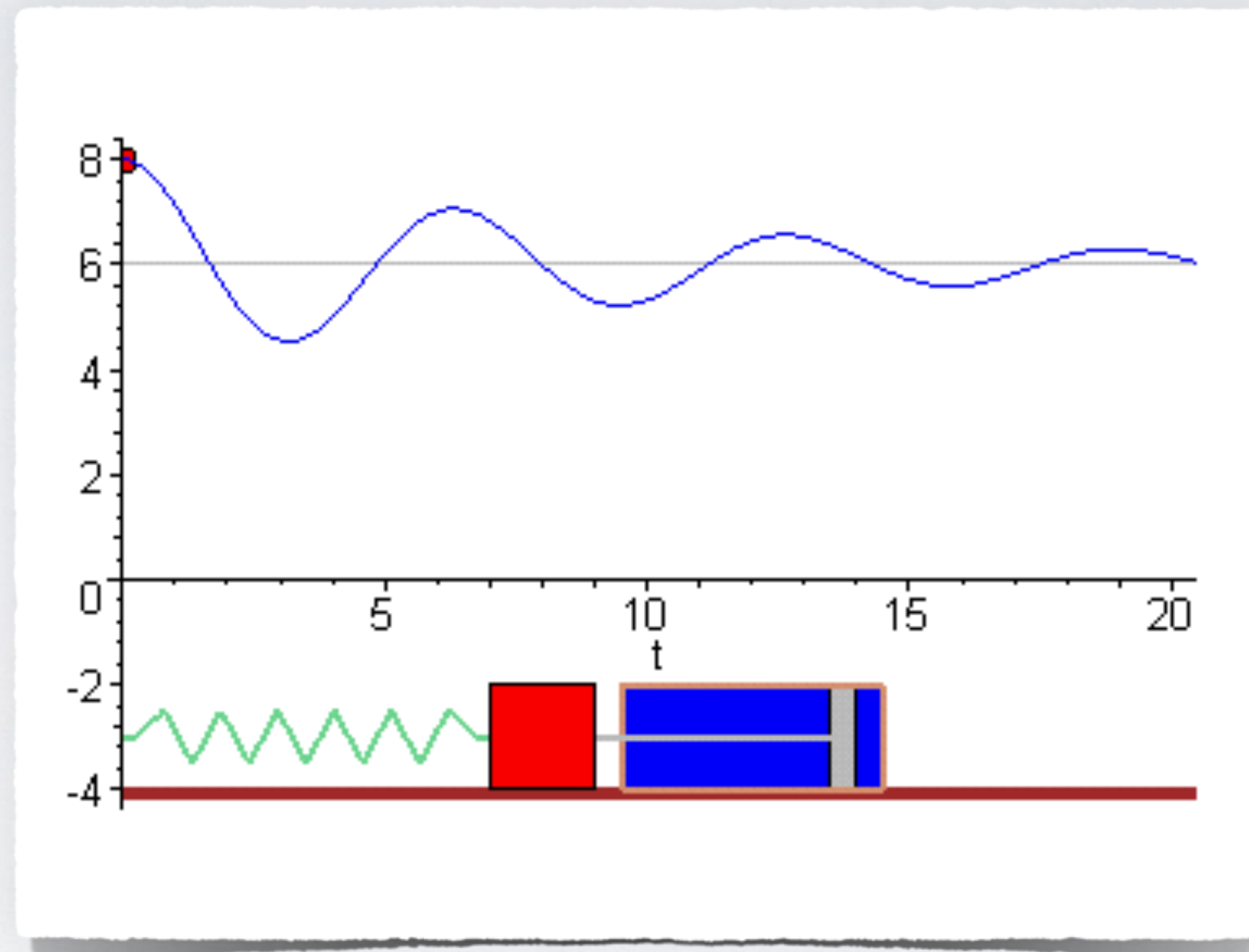


LESSON #4: HIDE COMPLEXITY

HIDE COMPLEXITY

- We use **.mask** to set a mask
- We use **.colors** to set gradient colors
- Complexity of drawing and animating is hidden

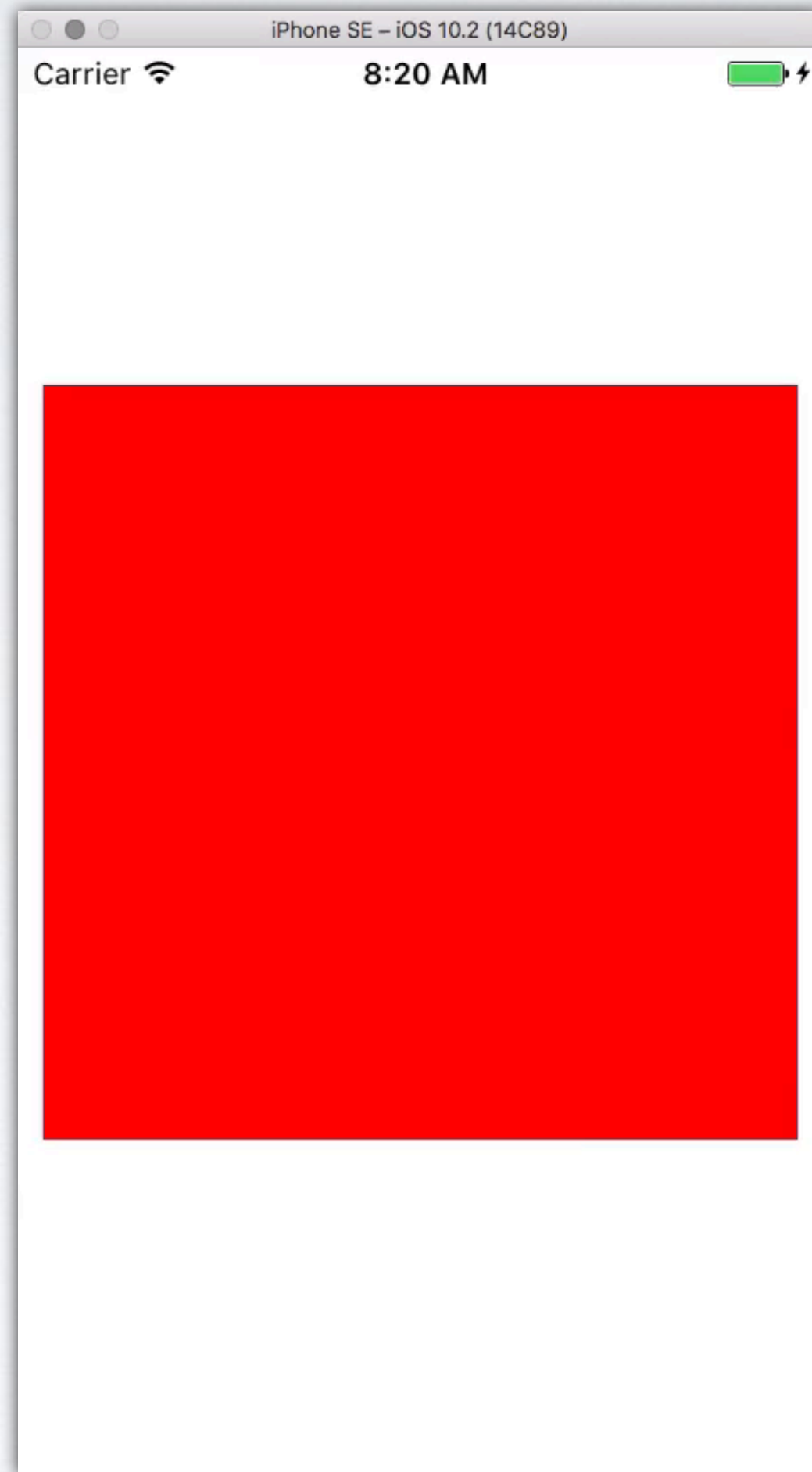
SPRING ANIMATIONS



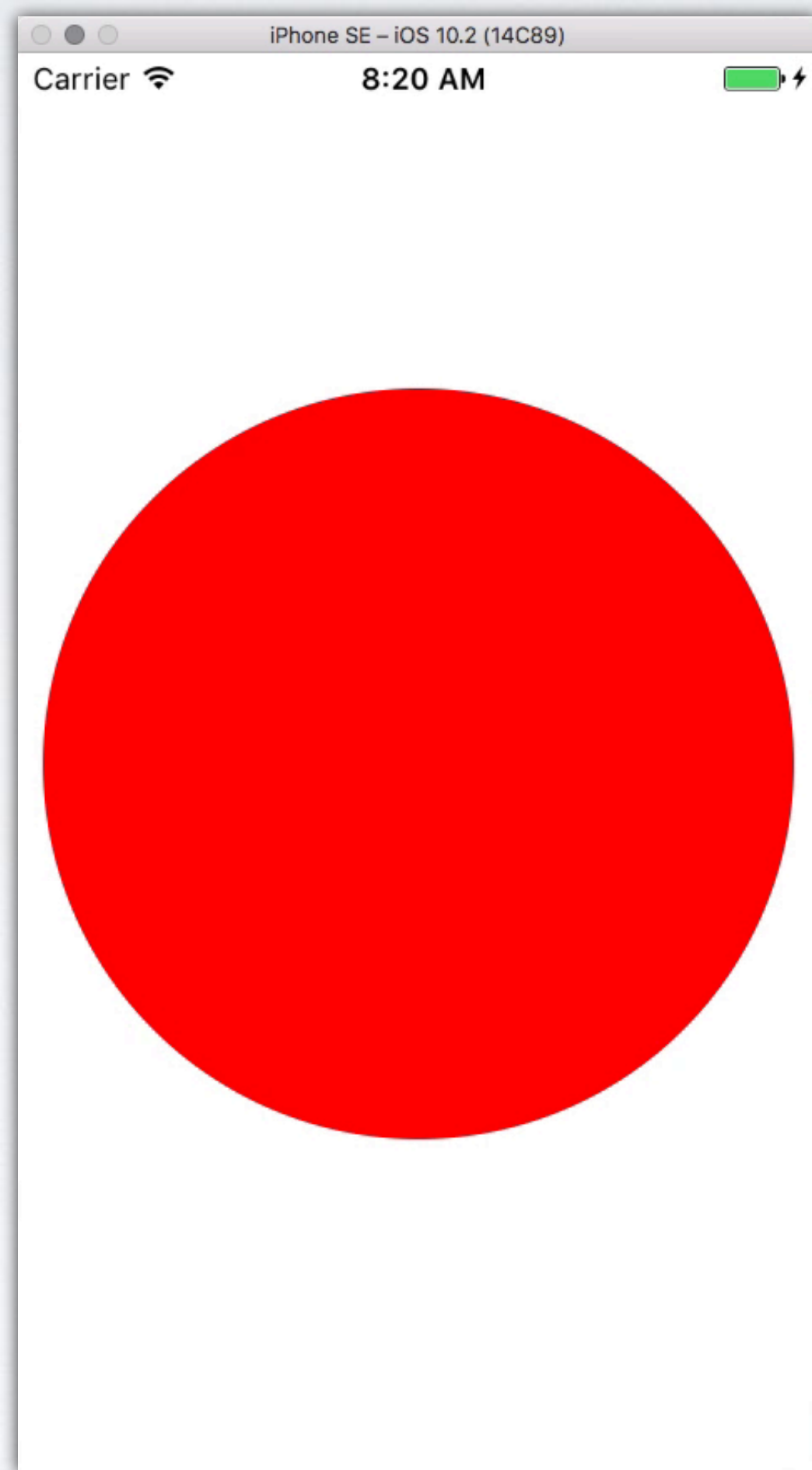
AS YOU MIGHT EXPECT...

- This is trivial in Core Animation
- Accomplished with **CASpringAnimation**
- Useful for more organic-feeling interfaces

SPRING ANIMATIONS



SPRING ANIMATIONS



SPRING ANIMATIONS

```
let spring = CASpringAnimation()  
spring.damping = 5  
spring.duration = 5  
spring.fromValue = layer.cornerRadius  
spring.toValue = layer.bounds.size.width / 2  
spring.fillMode = kCAFillModeForwards  
spring.isRemovedOnCompletion = false  
layer.add(spring, forKey: "cornerRadius")
```

LESSON #5: MAKE IT FUN

FUN

- Core Animation has a playfulness to it
- This reflects the potential for playful interactions in your apps
- Admittedly, you don't have to dress up animations very much—it's not an API like string encoding or task queuing

LESSON #6: MAKE IT UNSURPRISING

MAKE APIS UNSURPRISING

- Consider if the default implicit animation duration was *zero seconds*
- You wouldn't be able to see animations, even though they were the *default behavior!*

“If a necessary feature has a high astonishment factor, it may be necessary to redesign the feature.”

–Principle of Least Astonishment (PoLA)

SURPRISING APIS

- Classic example: a list or array **add()** or **insert()** that sorts
- Consider what is least surprising to the user, rather than the expected behavior given knowledge of the inner workings

LESSON #7: EXTENSIBILITY

EXTENSIBLE

- Core Animation supports custom properties
- Not just the predefined ones like **opacity** & **position**

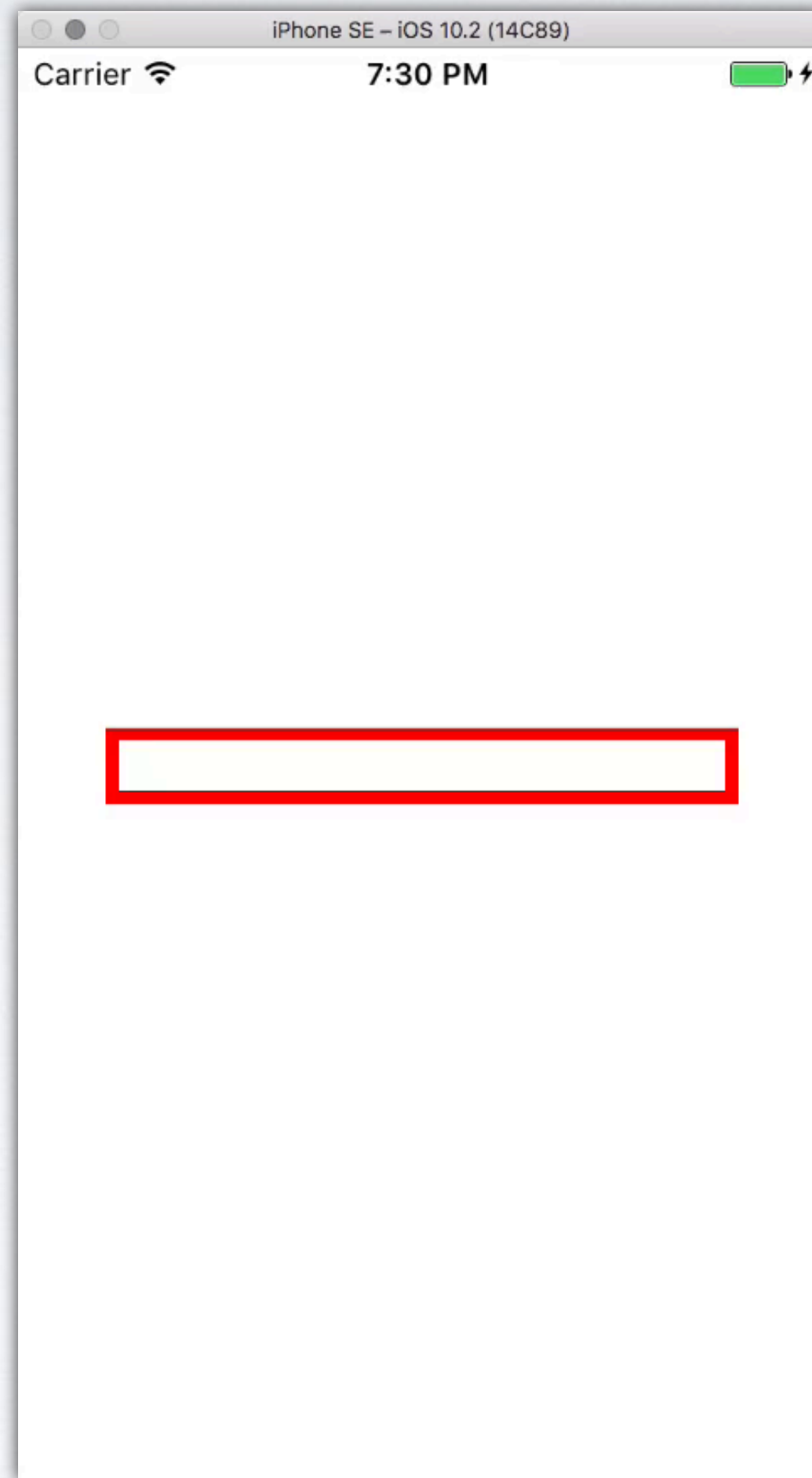
EXTENSIBLE

```
class ProgressLayer: CALayer {  
    @NSManaged var progress: CGFloat  
  
    override class func needsDisplay(forKey key: String) -> Bool {  
        if key == "progress" {  
            return true  
        }  
        return super.needsDisplay(forKey: key)  
    }  
  
    override func draw(in ctx: CGContext) {  
        ctx.setFillColor(UIColor.red.cgColor)  
        ctx.addRect(  
            CGRect(  
                x: 0,  
                y: 0,  
                width: presentation()!.progress * bounds.size.width,  
                height: bounds.size.height  
            )  
        )  
        ctx.fillPath()  
    }  
}
```

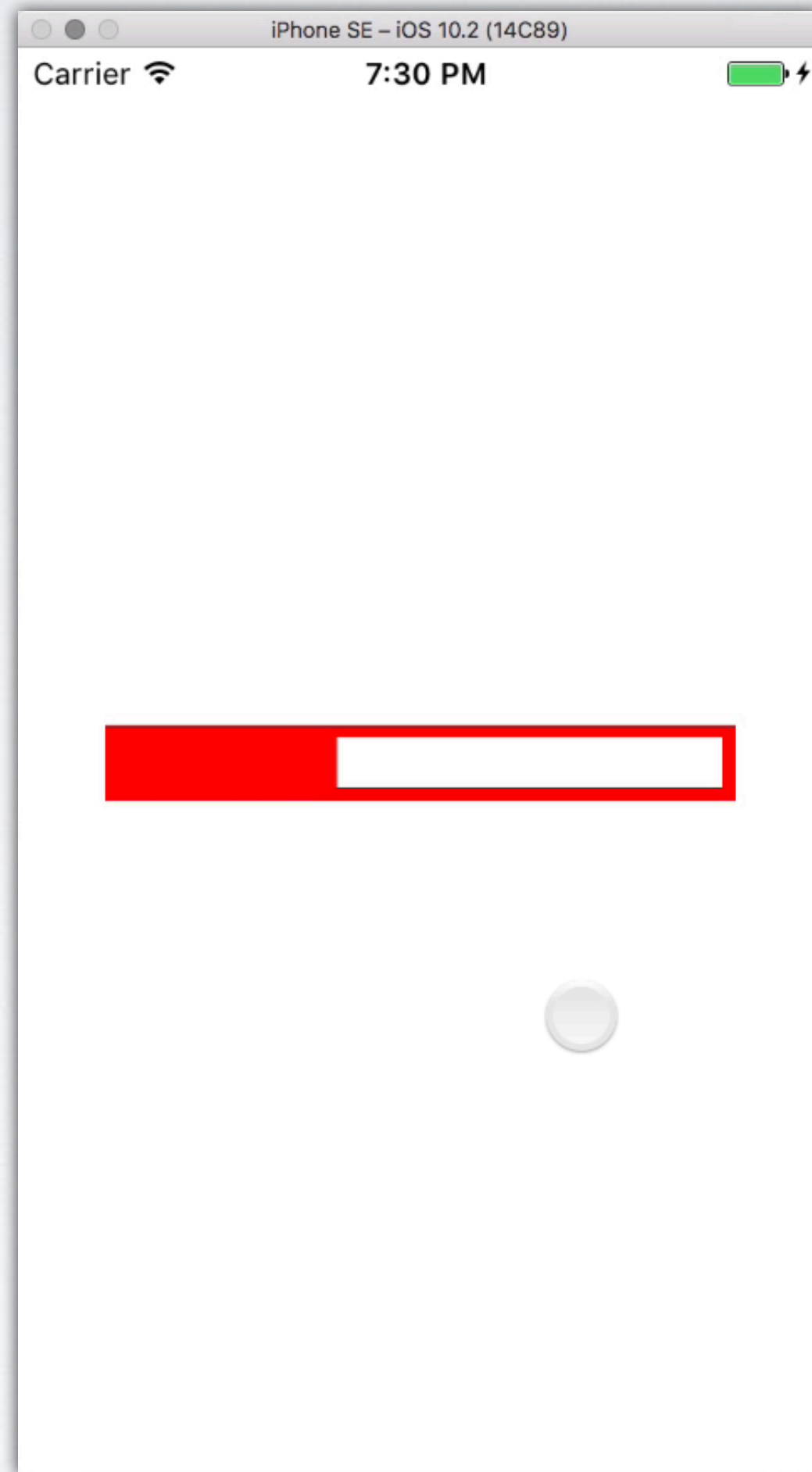
EXTENSIBLE

```
let progress = touch.location(in: view).x / view.bounds.size.width
let animation = CABasicAnimation(keyPath: "progress")
animation.timingFunction = CAMediaTimingFunction(
    name: kCAMediaTimingFunctionEaseInEaseOut
)
animation.fromValue = layer.presentation()!.progress
animation.toValue = progress
animation.isRemovedOnCompletion = false
animation.fillMode = kCAFillModeForwards
animation.duration = 1
layer.add(animation, forKey: "progress")
```

EXTENSIBLE



EXTENSIBLE



LESSON #8: DOCUMENT IT

DOCUMENTATION

- Not the most glamorous thing, but very important
- This is important even for your future self
- But especially important for other consumers

Apple Inc. developer.apple.com/library/content/doc

Guides and Sample Code

Developer

Core Animation Programming Guide

Table of Contents

- Introduction
- ▶ Core Animation Basics
- ▶ Setting Up Layer Objects
- ▶ Animating Layer Content
- ▶ Building a Layer Hierarchy
- ▶ Advanced Animation Tricks
- ▶ Changing a Layer's Default Behavior
- ▶ Improving Animation Performance
- ▶ Appendix A: Layer Style Property Animations
- ▶ Appendix B: Animatable Properties
- ▶ Appendix C: Key-Value Coding Extensions
- Revision History

About Core Animation [Next](#)

Core Animation is a graphics rendering and animation infrastructure available on both iOS and OS X that you use to animate the views and other visual elements of your app. With Core Animation, most of the work required to draw each frame of an animation is done for you. All you have to do is configure a few animation parameters (such as the start and end points) and tell Core Animation to start. Core Animation does the rest, handing most of the actual drawing work off to the onboard graphics hardware to accelerate the rendering. This automatic graphics acceleration results in high frame rates and smooth animations without burdening the CPU and slowing down your app.

If you are writing iOS apps, you are using Core Animation whether you know it or not. And if you are writing OS X apps, you can take advantage of Core Animation with extremely little effort. Core Animation sits beneath AppKit and UIKit and is integrated tightly into the view workflows of Cocoa and Cocoa Touch. Of course, Core Animation also has interfaces that extend the capabilities exposed by your app's views and give you more fine-grained control over your app's animations.

```
graph TD; UIKit[UIKit / AppKit] --- CoreAnimation[Core Animation]; CoreAnimation --- OpenGL[OpenGL ES / OpenGL]; CoreAnimation --- CoreGraphics[Core Graphics];
```

Feedback

Core Animation Programming Guide

Table of Contents

Introduction

Core Animation Basics

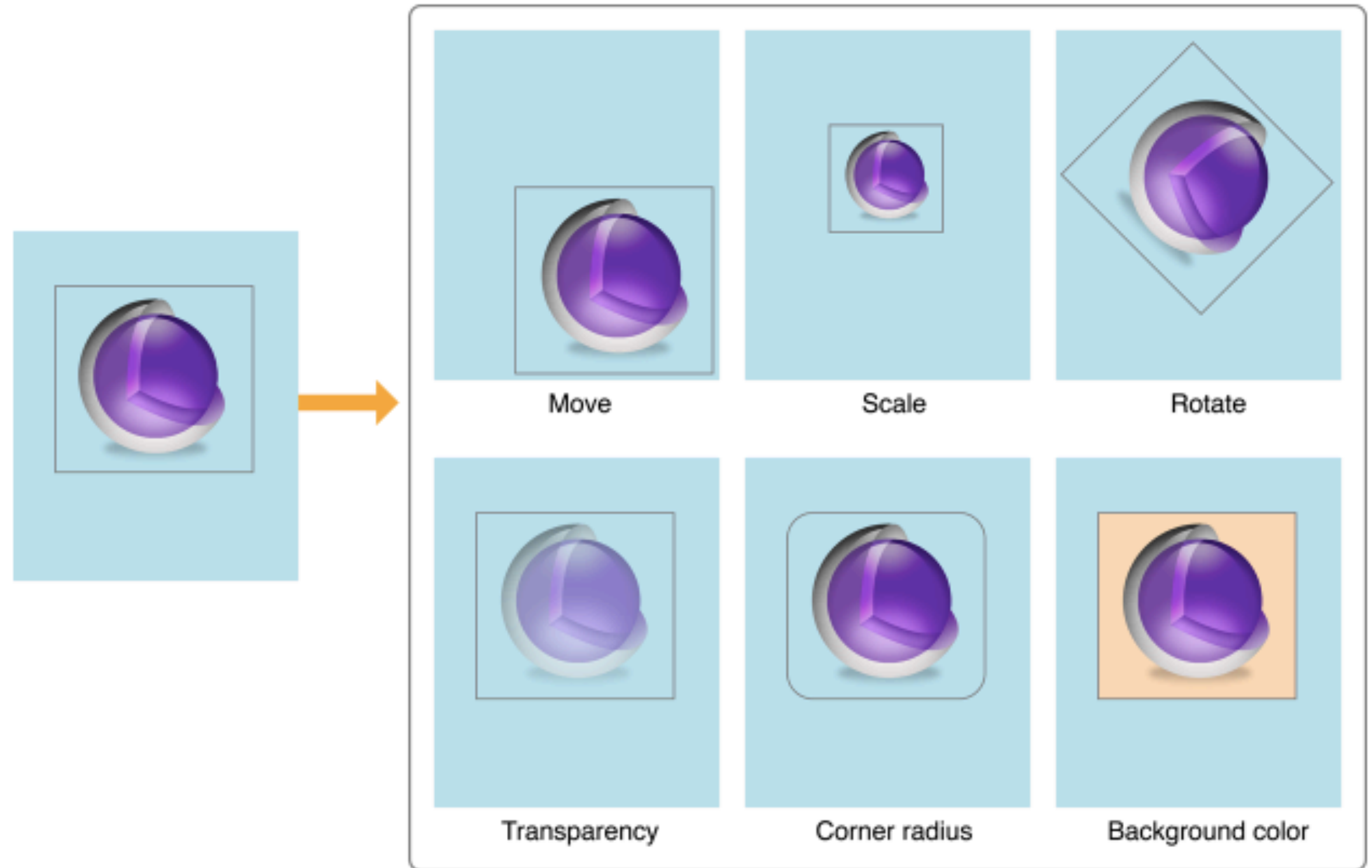
- Layers Provide the Basis for Drawing and Animations
- Layer Objects Define Their Own Geometry
 - Layer Trees Reflect Different Aspects of the Animation State
 - The Relationship Between Layers and Views

Setting Up Layer Objects

- Animating Layer Content
- Building a Layer Hierarchy
- Advanced Animation Tricks
- Changing a Layer's Default Behavior
- Improving Animation Performance
- Appendix A: Layer Style Property Animations
- Appendix B: Animatable Properties
- Appendix C: Key-Value Coding Extensions

Revision History

Figure 1-2 Examples of animations you can perform on layers



Feedback

Apple Inc. developer.apple.com/library/content/doc

Guides and Sample Code

Core Animation Programming Guide

- Table of Contents
- Introduction
- Core Animation Basics
 - Layers Provide the Basis for Drawing and Animations
 - Layer Objects Define Their Own Geometry
 - Layer Trees Reflect Different Aspects of the Animation State
 - The Relationship Between Layers and Views
- Setting Up Layer Objects
 - Enabling Core Animation Support in Your App
 - Changing the Layer Object Associated with a View
 - Providing a Layer's Contents
 - Adjusting a Layer's Visual Style and Appearance
 - The Layer Redraw Policy for OS X Views Affects Performance
 - Adding Custom Properties to a Layer
 - Printing the Contents of a Layer-Backed View
 - Animating Layer Content
 - Building a Layer Hierarchy

background color is rendered behind the layer's contents image and the border is rendered on top of that image, as shown in Figure 2-3. If the layer contains sublayers, they also appear underneath the border. Because the background color sits behind your image, that color shines through any transparent portions of your image.

Figure 2-3 Adding a border and background to a layer

Listing 2-5 shows the code needed to set the background color and border for a layer. All of these properties are animatable.

Feedback

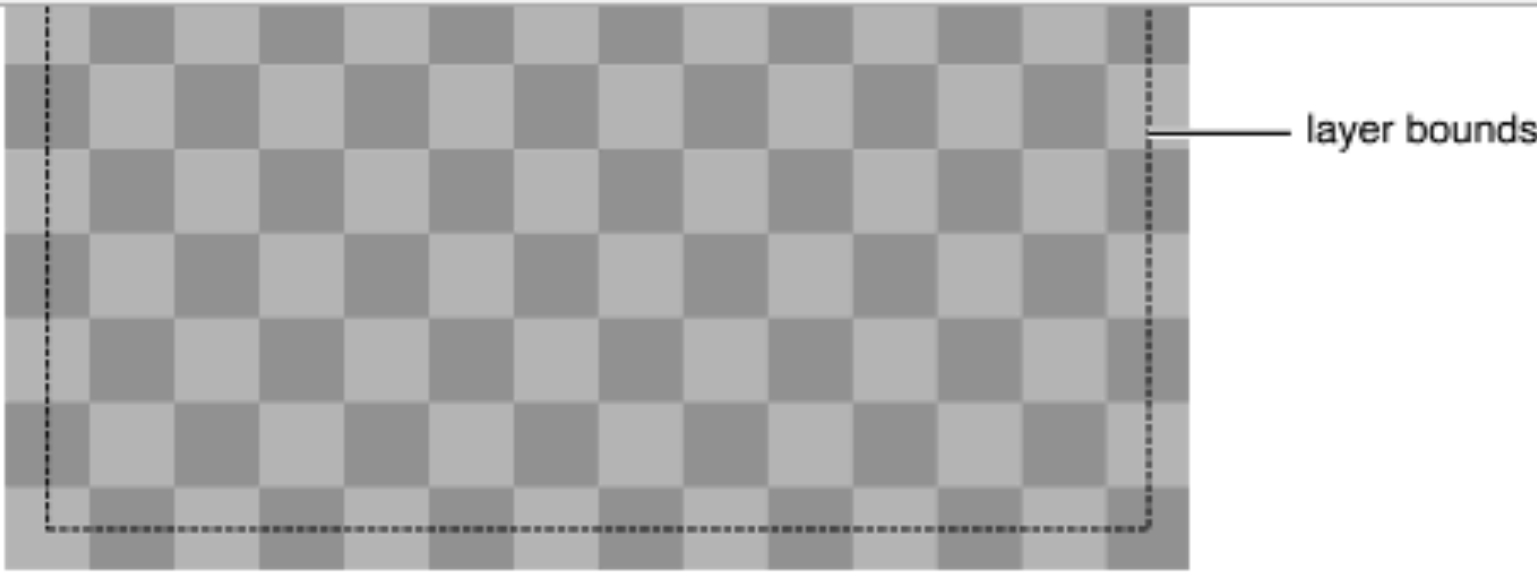
Apple Inc. developer.apple.com/library/content/documentation

Guides and Sample Code

Core Animation Programming Guide

Table of Contents

- Introduction
- Core Animation Basics
 - Layers Provide the Basis for Drawing and Animations
 - Layer Objects Define Their Own Geometry
 - Layer Trees Reflect Different Aspects of the Animation State
 - The Relationship Between Layers and Views
- Setting Up Layer Objects
 - Enabling Core Animation Support in Your App
 - Changing the Layer Object Associated with a View
 - Providing a Layer's Contents
 - Adjusting a Layer's Visual Style and Appearance
 - The Layer Redraw Policy for OS X Views Affects Performance
 - Adding Custom Properties to a Layer
 - Printing the Contents of a Layer-Backed View
 - Animating Layer Content
 - Building a Layer Hierarchy



The following `CALayer` properties specify a layer's geometry:

- `bounds`
- `position`
- `frame` (computed from the `bounds` and `position` and is not animatable)
- `anchorPoint`
- `cornerRadius`
- `transform`
- `zPosition`

iOS Note: The `cornerRadius` property is supported only in iOS 3.0 and later.

Feedback

Apple Inc. developer.apple.com/library/content/documentation

Guides and Sample Code

Developer

Core Animation Programming Guide

Table of Contents

- Introduction
- Core Animation Basics
 - Layers Provide the Basis for Drawing and Animations
 - Layer Objects Define Their Own Geometry
 - Layer Trees Reflect Different Aspects of the Animation State
 - The Relationship Between Layers and Views
- Setting Up Layer Objects
 - Enabling Core Animation Support in Your App
 - Changing the Layer Object Associated with a View
 - Providing a Layer's Contents
 - Adjusting a Layer's Visual Style and Appearance
 - The Layer Redraw Policy for OS X Views Affects Performance
 - Adding Custom Properties to a Layer
 - Printing the Contents of a Layer-Backed View
- Animating Layer Content
- Building a Layer Hierarchy

Previous Next

Animatable Properties

Many of the properties in `CALayer` and `CIFilter` can be animated. This appendix lists those properties, along with the animation used by default.

CALayer Animatable Properties

Table B-1 lists the properties of the `CALayer` class that you might consider animating. For each property, the table also lists the type of default animation object that is created to execute an implicit animation.

Table B-1 Layer properties and their default animations

Property	Default animation
<code>anchorPoint</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2.
<code>backgroundColor</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2.
<code>backgroundFilters</code>	Uses the default implied <code>CATransition</code> object, described in Table B-3. Sub-properties of the filters are animated using the default implied <code>CABasicAnimation</code> object, described in Table B-2.

Feedback

Apple Inc. developer.apple.com/library/content/documentation

Guides and Sample Code Developer

Core Animation Programming Guide

▼ Table of Contents

- Introduction
- ▼ Core Animation Basics
 - ▶ Layers Provide the Basis for Drawing and Animations
 - ▶ Layer Objects Define Their Own Geometry
 - Layer Trees Reflect Different Aspects of the Animation State
 - The Relationship Between Layers and Views
 - ▼ Setting Up Layer Objects
 - Enabling Core Animation Support in Your App
 - ▶ Changing the Layer Object Associated with a View
 - ▶ Providing a Layer's Contents
 - ▶ Adjusting a Layer's Visual Style and Appearance
 - The Layer Redraw Policy for OS X Views Affects Performance
 - Adding Custom Properties to a Layer
 - Printing the Contents of a Layer-Backed View
 - ▶ Animating Layer Content
 - ▶ Building a Layer Hierarchy

<code>cornerRadius</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>doubleSided</code>	There is no default implied animation.
<code>filters</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 . Sub-properties of the filters are animated using the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>frame</code>	This property is not animatable. You can achieve the same results by animating the bounds and position properties.
<code>hidden</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>mask</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>masksToBounds</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>opacity</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>position</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>shadowColor</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>shadowOffset</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .
<code>shadowOpacity</code>	Uses the default implied <code>CABasicAnimation</code> object, described in Table B-2 .

[Feedback](#)

Core Animation Programming Guide

Table of Contents

Introduction

Core Animation Basics

Layers Provide the Basis for Drawing and Animations

Layer Objects Define Their Own Geometry

Layer Trees Reflect Different Aspects of the Animation State

The Relationship Between Layers and Views

Setting Up Layer Objects

Enabling Core Animation Support in Your App

Changing the Layer Object Associated with a View

Providing a Layer's Contents

Adjusting a Layer's Visual Style and Appearance

The Layer Redraw Policy for OS X Views Affects Performance

Adding Custom Properties to a Layer

Printing the Contents of a Layer-Backed View

Animating Layer Content

Building a Layer Hierarchy

Table B-2 Default Implied Basic Animation

Description	Value
Class	CABasicAnimation
Duration	0.25 seconds, or the duration of the current transaction
Key path	Set to the property name of the layer.

Table B-3 lists the animation object configuration for default transition-based animations.

Table B-3 Default Implied Transition

Description	Value
Class	CATransition
Duration	0.25 seconds, or the duration of the current transaction
Type	Fade (<code>kCATransitionFade</code>)
Start progress	0.0

Feedback

RECAP

1. Discoverable
2. Flexible
3. Intuitive
4. Hide Complexity
5. Make it fun
6. Keep it unsurprising
7. Allow extensibility
8. Document it

THANK YOU!



Q&A

- Twitter: [incanus77](#)
- GitHub: [incanus](#)
- Web: justinmiller.io
- Mapbox: mapbox.com/blog

