



apple WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

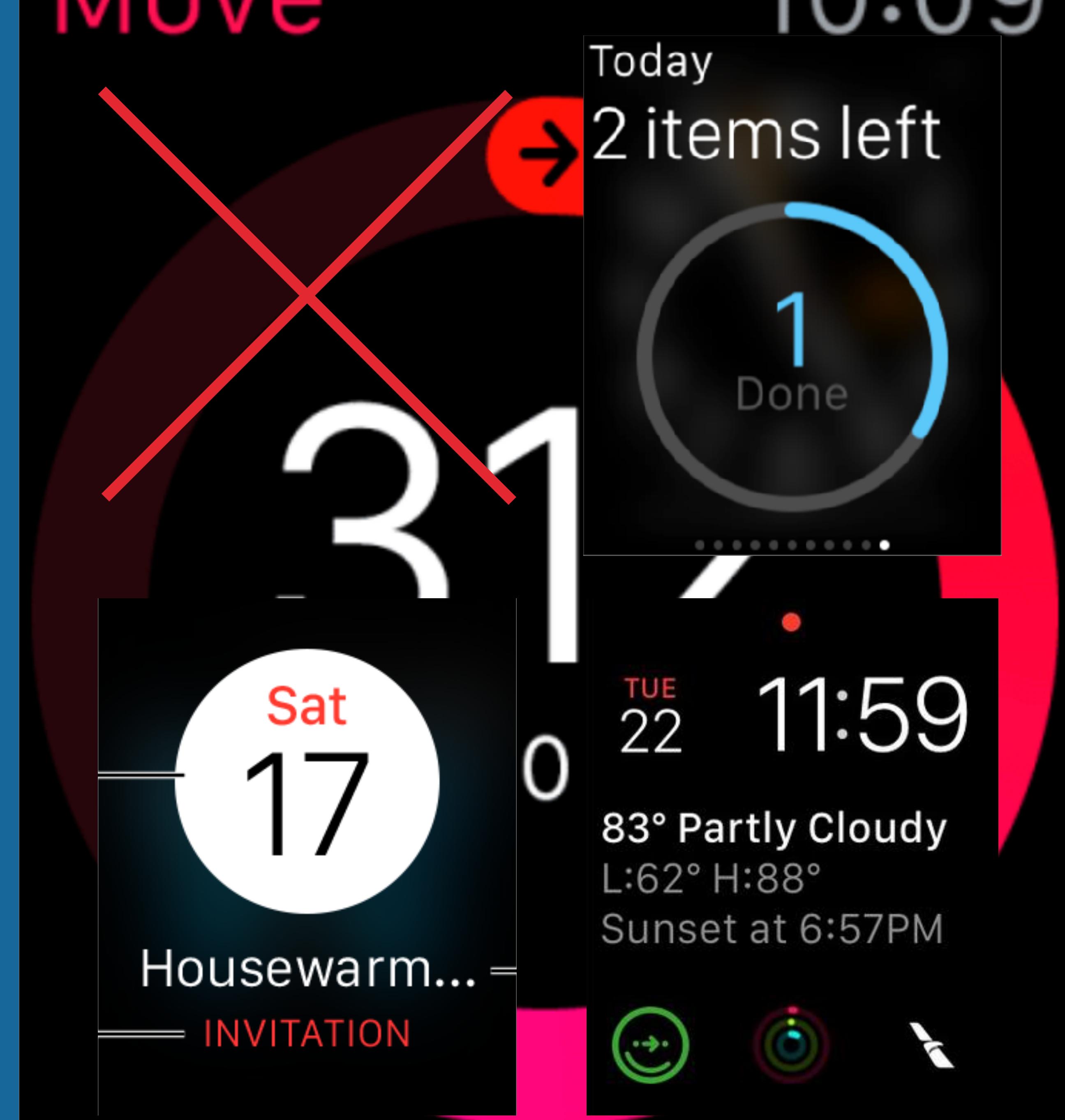
USER INTERACTIONS ON



USER INTERACTIONS

SUBTITLE

- User interactions available
 - WatchKit App
 - Glances
 - Dock
 - Custom notifications
 - Complication (new in 2.0)



USER INTERACTIONS

- What makes up a Watch App?
 - Interface
 - Notification Interface
 - Complication Interface
 - Code for managing all interfaces in extension

The screenshot shows the Xcode interface with a project titled "InterFaces" selected. The project structure is as follows:

- InterFaces (Target)
 - InterFaces (Group)
 - AppDelegate.swift
 - ViewController.swift
 - Main.storyboard
 - Assets.xcassets
 - LaunchScreen.storyboard
 - Info.plist
 - InterFaces WatchKit App (Group)
 - Interface.storyboard
 - Assets.xcassets
 - Info.plist
 - InterFaces WatchKit Extension (Group)
 - InterfaceController.swift
 - ExtensionDelegate.swift
 - NotificationController.swift
 - GlanceController.swift
 - ComplicationController.swift
 - Assets.xcassets
 - Info.plist
 - Supporting Files
 - Products

The code editor on the right displays the content of the selected file, `InterfaceController.swift`:

```
// InterfaceController.swift
// InterFaces WatchKit Extension
// Created by T. Andrew Bi... 2016 The University of Texas at Austin
// Copyright © 2016 The University of Texas at Austin. All rights reserved.

import WatchKit
import Foundation
import WatchConnectivity

class InterfaceController: WKInterfaceController {

    // MARK: - IBOutlets
    @IBOutlet var nameLabel: WKInterfaceLabel

    // MARK: - Connectivity session
    let session = WCSession.default()

    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // Set the name label
        nameLabel.setText("Hello, World!")
        // Check if the session is connected
        if session.isConnected {
            session.sendMessage(["text": "Hello, World!"], replyHandler: { [weak self] (reply) in
                self?.nameLabel.setText(reply["text"] as? String)
            }, errorHandler: { [weak self] (error) in
                print("Error: \(error.localizedDescription)")
            })
        }
    }

    @IBAction func tapToInteractive(_ sender: WKInterfaceButton) {
        print("Tap (interactive)")
        let applicationDict = [
            "text": "Hello, World!"
        ]
        session.sendMessage(applicationDict, replyHandler: { [weak self] (reply) in
            self?.nameLabel.setText(reply["text"] as? String)
        }, errorHandler: { [weak self] (error) in
            print("Error: \(error.localizedDescription)")
        })
    }

    @IBAction func tapToApplication(_ sender: WKInterfaceButton) {
        print("Tap (application)")
        do {
            let applicationDict = [
                "text": "Hello, World!"
            ]
            try session.default().sendMessage(applicationDict)
        } catch {
            // Handle errors here
            print(error)
        }
    }

    // MARK: - Lifecycle
    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // Set the name label
        nameLabel.setText("Hello, World!")
        // Check if the session is connected
        if session.isConnected {
            session.sendMessage(["text": "Hello, World!"], replyHandler: { [weak self] (reply) in
                self?.nameLabel.setText(reply["text"] as? String)
            }, errorHandler: { [weak self] (error) in
                print("Error: \(error.localizedDescription)")
            })
        }
    }
}
```

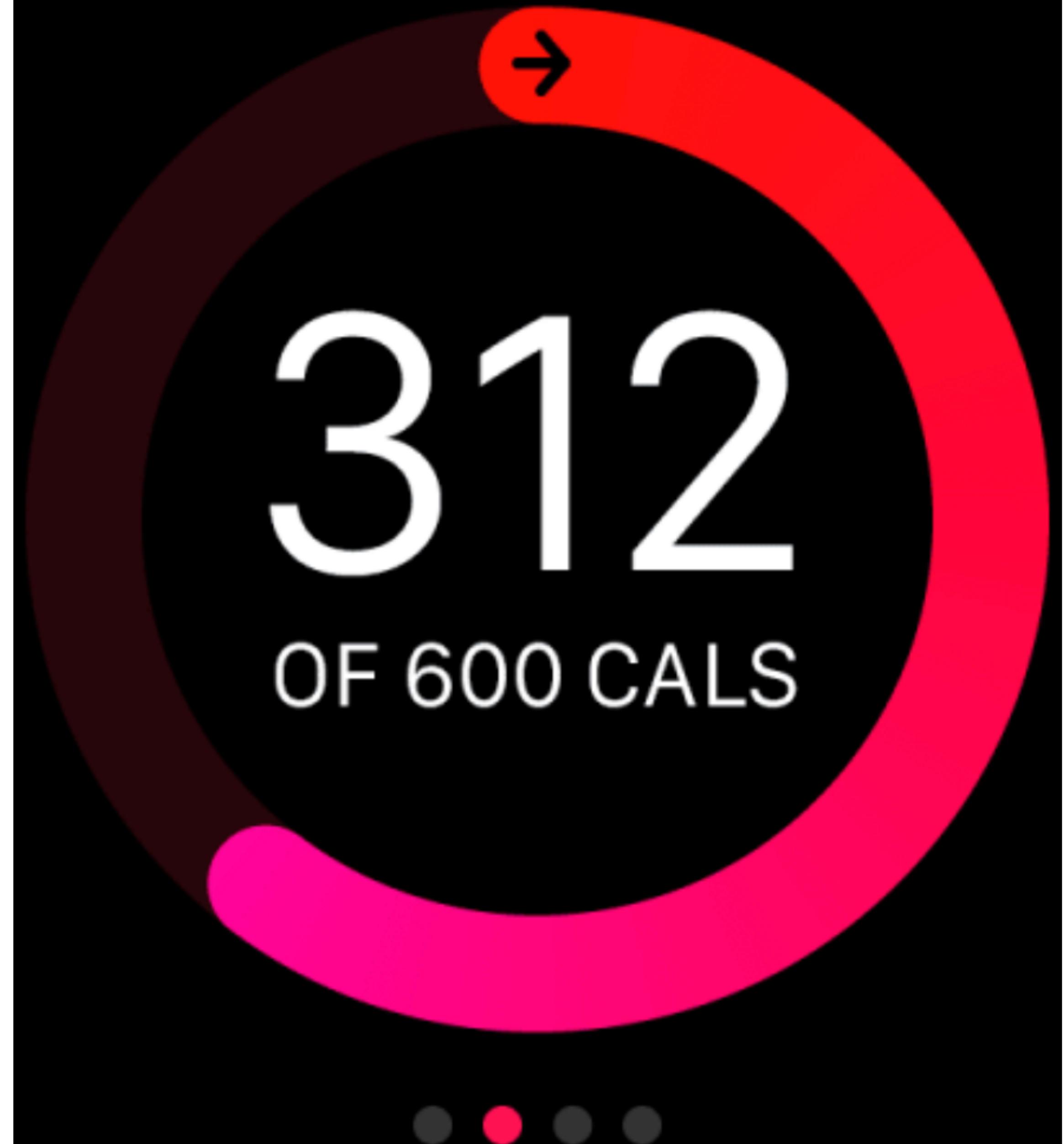
USER INTERACTIONS

SUBTITLE

- WatchKit App
 - Full-app experience which they interact with by opening your app from the Home screen
 - Main way to interact with data
 - Typically, users will only interact with a subset of data/features

Move

10:09



USER INTERFACES ON WATCHOS

USER INTERACTIONS



Glanceable



Actionable

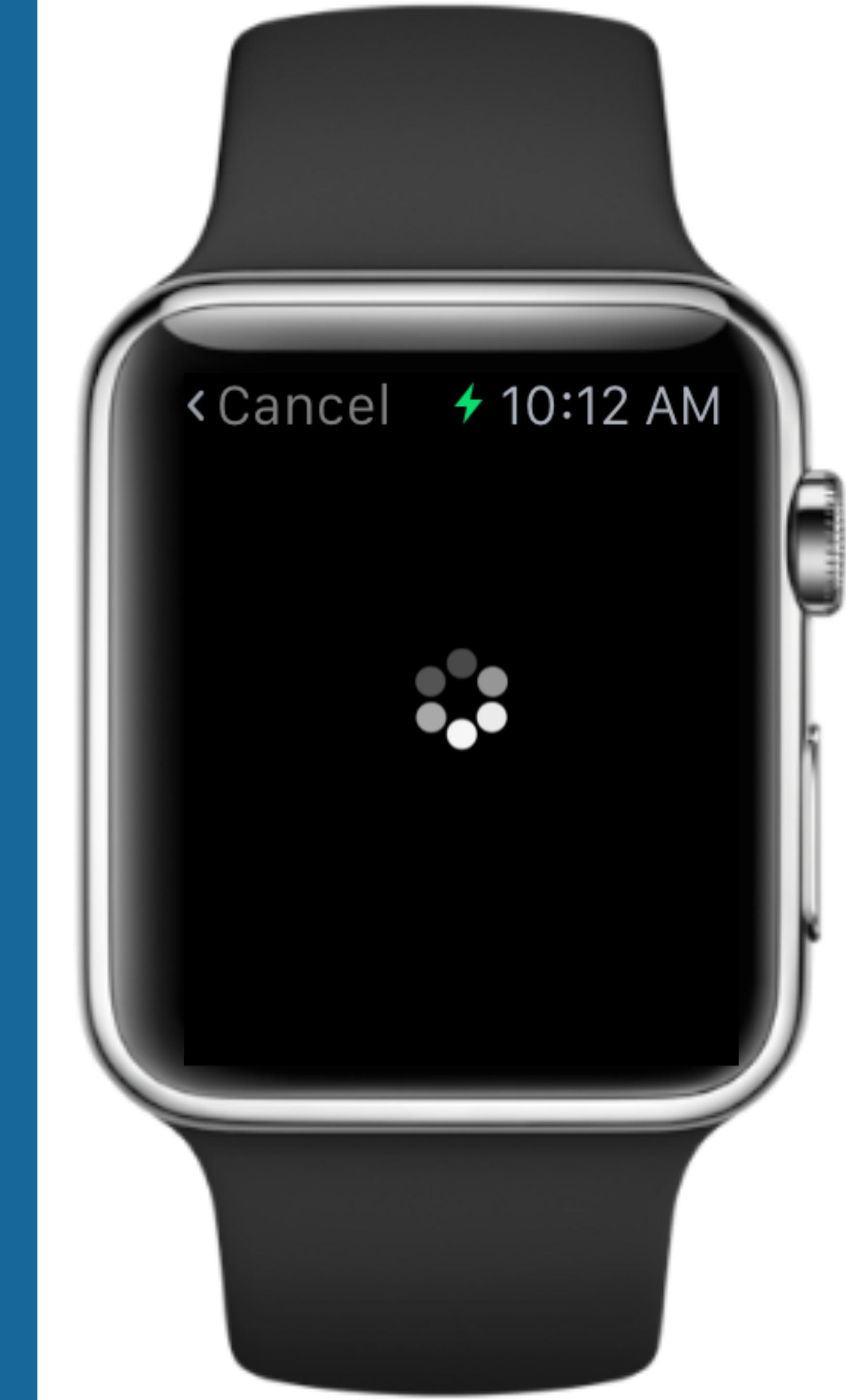


Responsive

- Design for quick interactions

USER INTERACTIONS

- Users will not be interested in seeing this interface in your application



USER INTERACTIONS

- How long is a quick interaction?

. 2 seconds

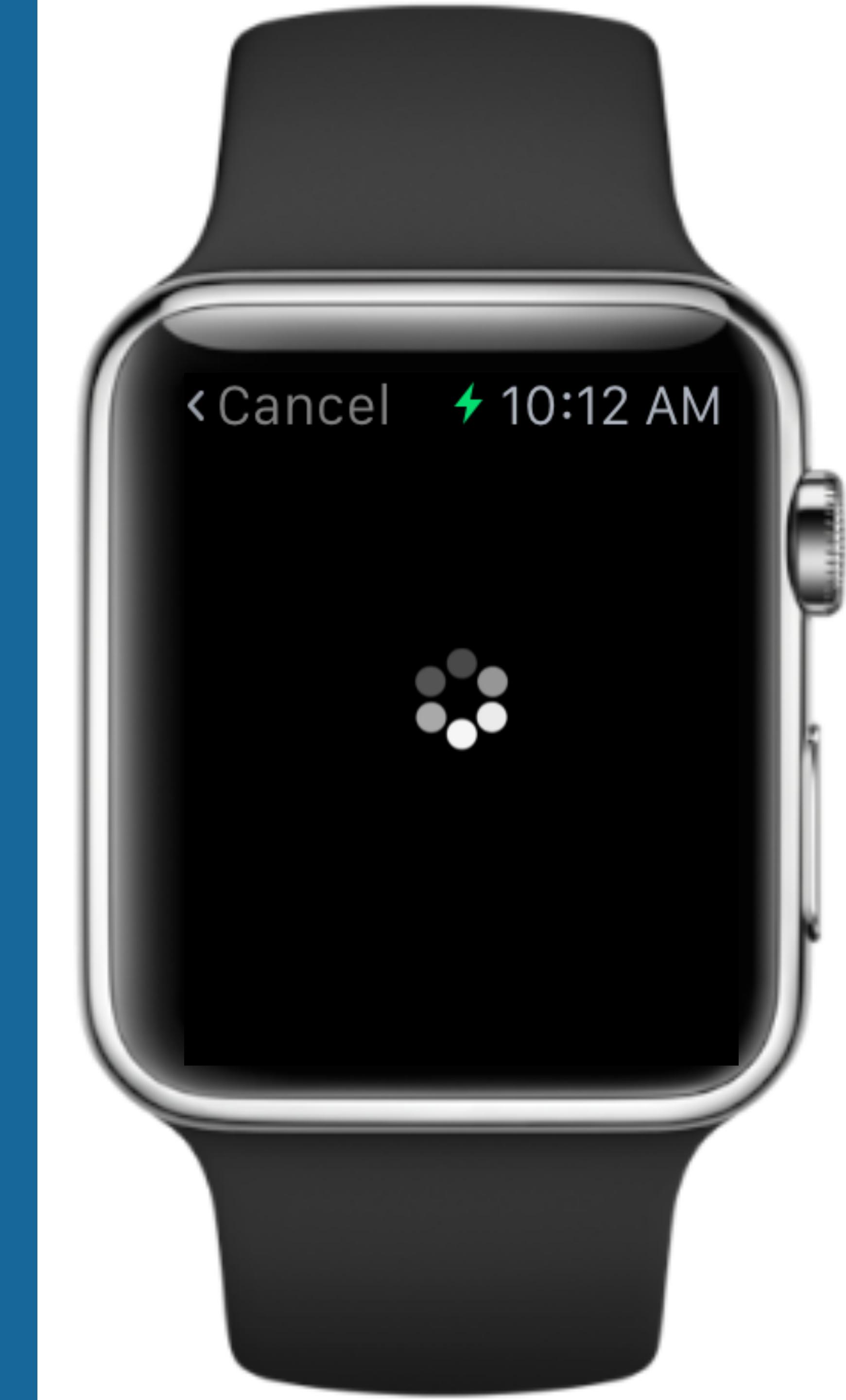
USER INTERACTIONS

- Design for quick and fast interactions
- Be very conscious of load times
- Compliment your iOS app
- Independent watch apps



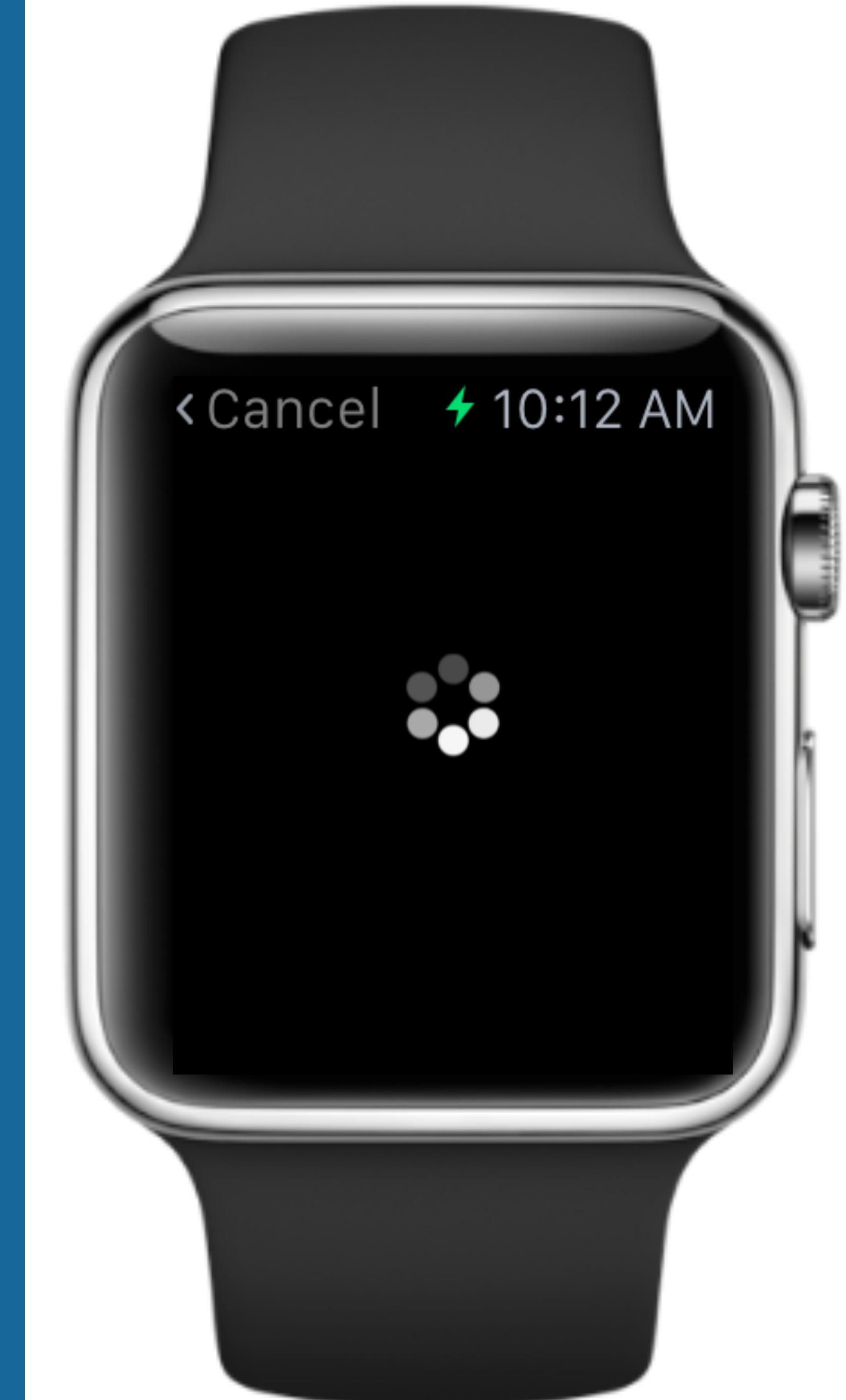
USER INTERACTIONS

- Interacting
 - Gesture recognizers
 - Digital crown rotation



USER INTERACTIONS

- watchOS additions to support displaying and updating information
 - Improved table navigation (vertical paging)
 - Support for new Notifications Framework
 - SceneKit and SpriteKit integration



SOME WATCH APPS

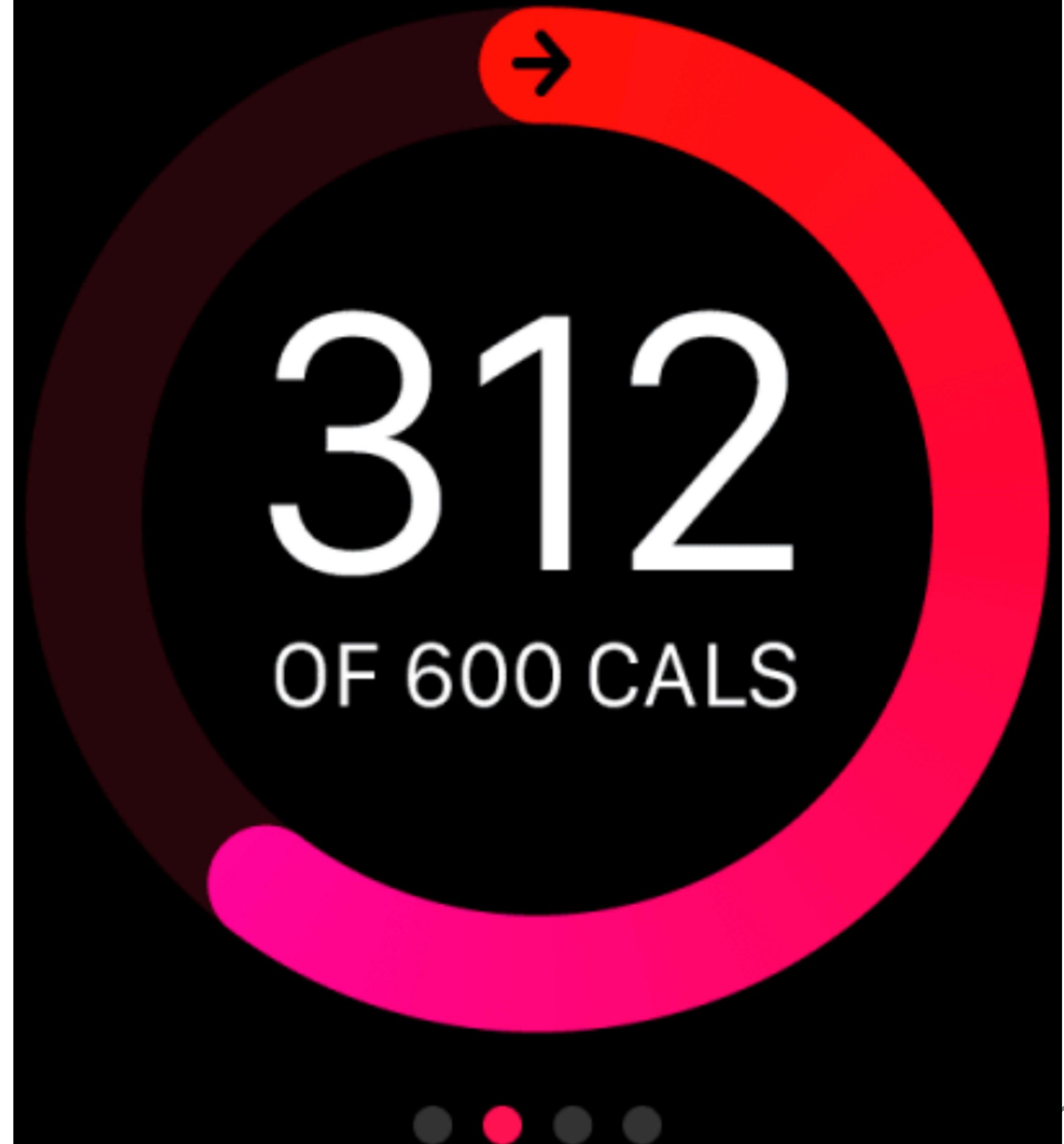
USER INTERACTIONS

SUBTITLE

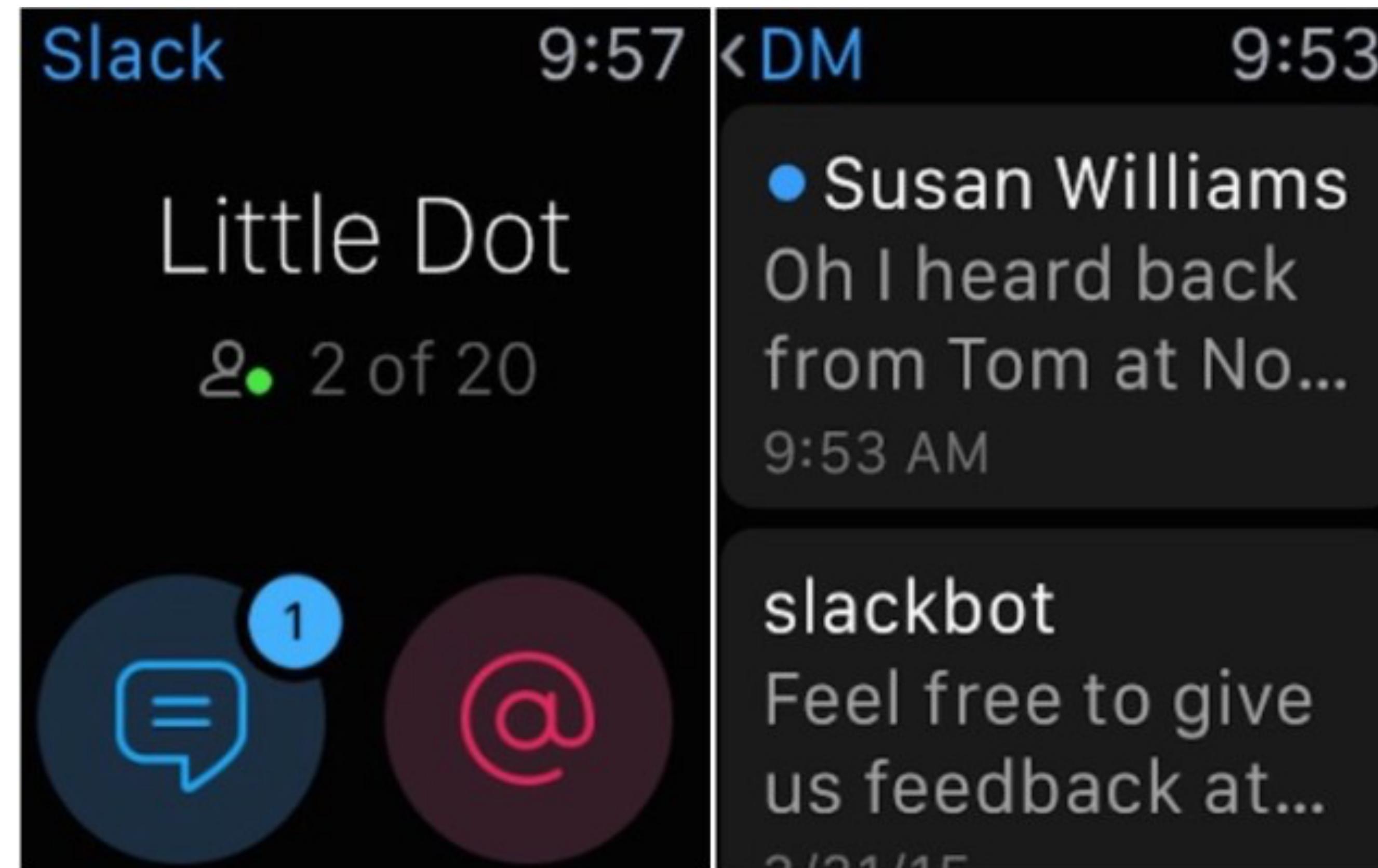
- Watch apps have different navigation and interaction models
 - Health app
 - Music app
 - Game app
 - App app

Move

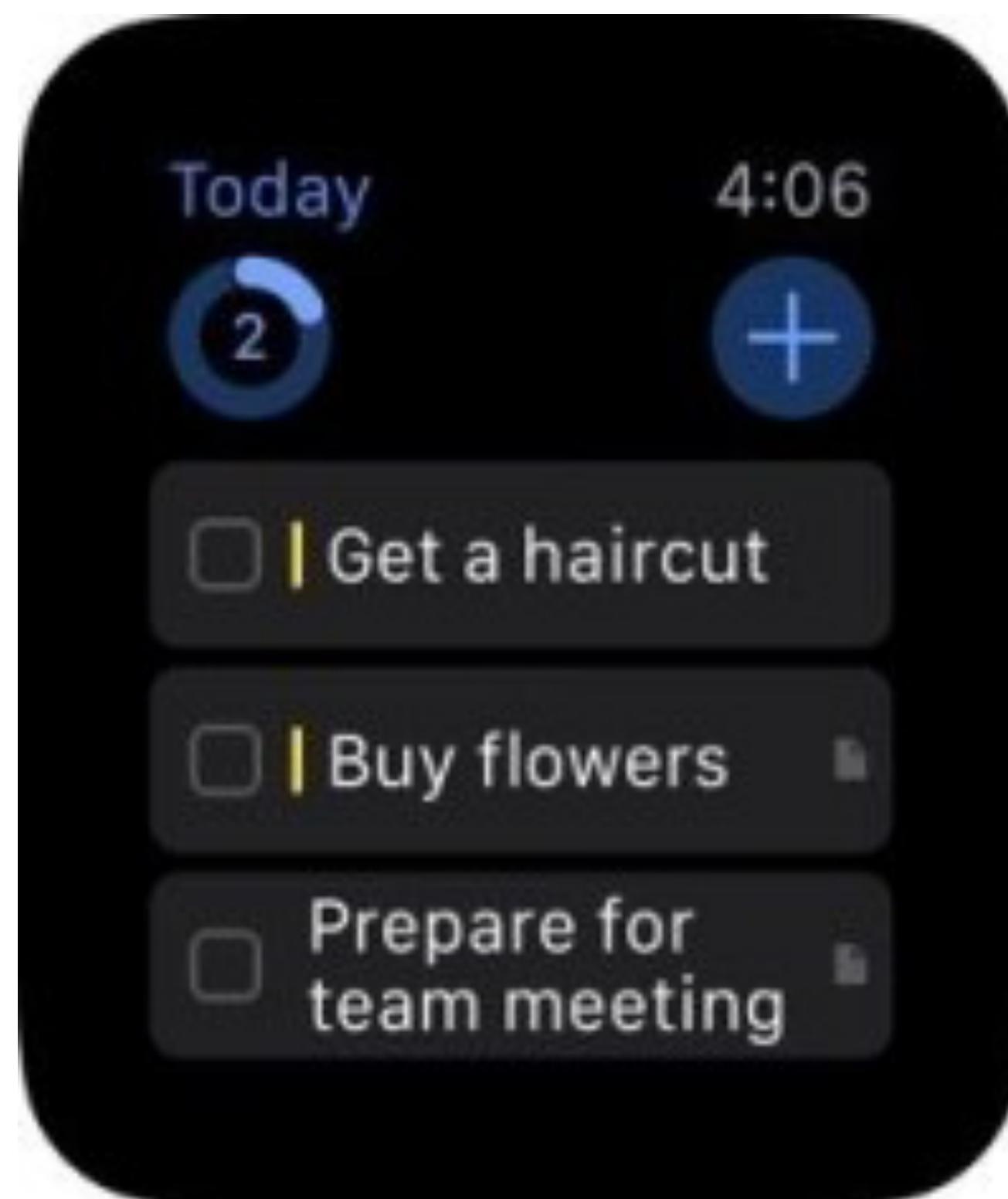
10:09



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS

Deliveries 10:09

Sailor Moon S... Delivered: Nort...

3 DAYS Seconds and... In transit: Toro...

8 DAYS 2 11-inch Ma... Ship date unk... Updated a moment ago

Updated a moment ago

< Seconds an... 10:09

3 days

In transit: Toronto, ON

Delivered by Saturday

A map of the Great Lakes region showing the delivery route. A green dot marks the starting point in Buffalo, NY, and a grey dot marks the destination in Toronto, ON. The route follows the western shore of Lake Erie and the northern shore of Lake Ontario.

USER INTERACTIONS



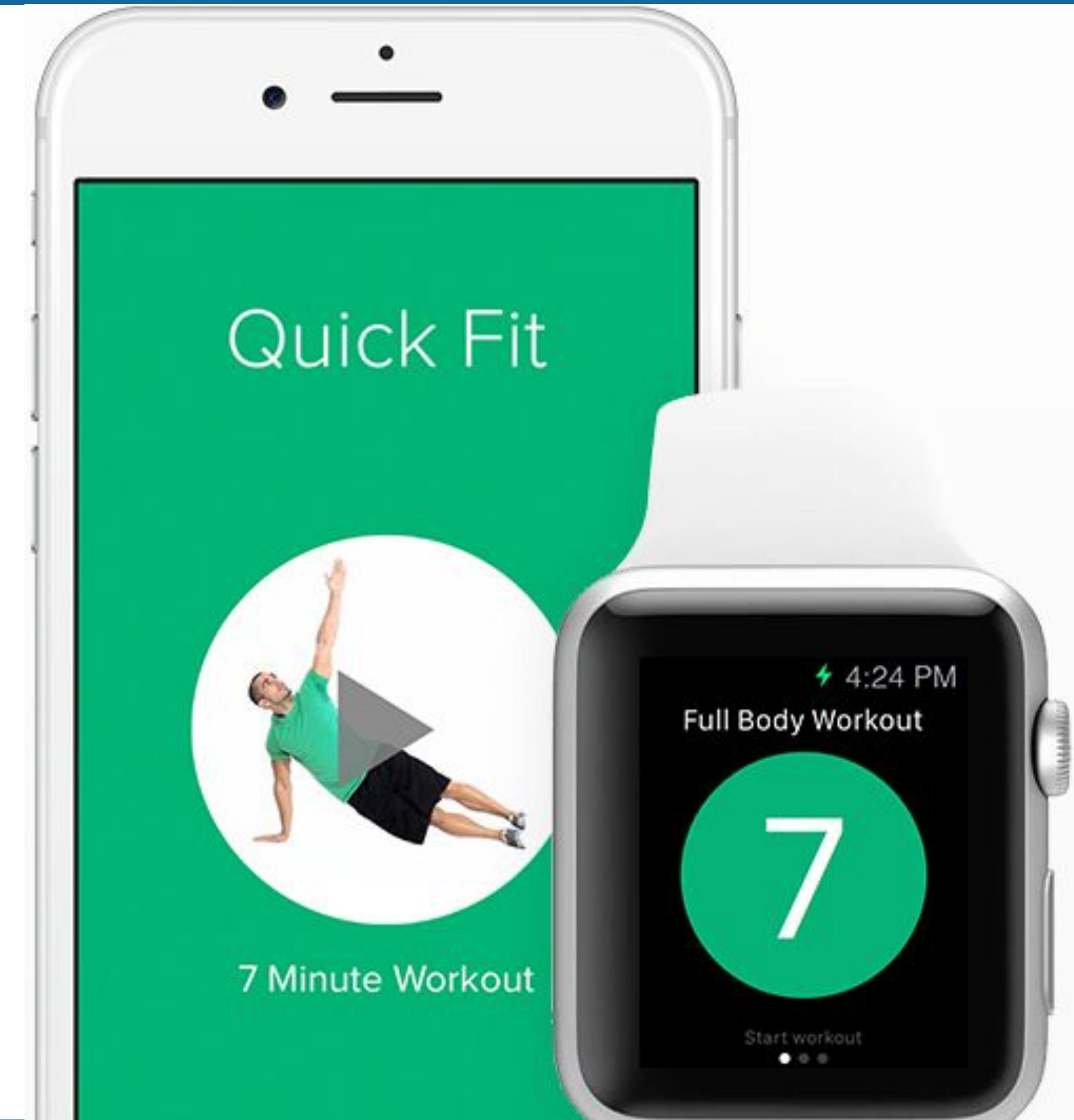
USER INTERACTIONS



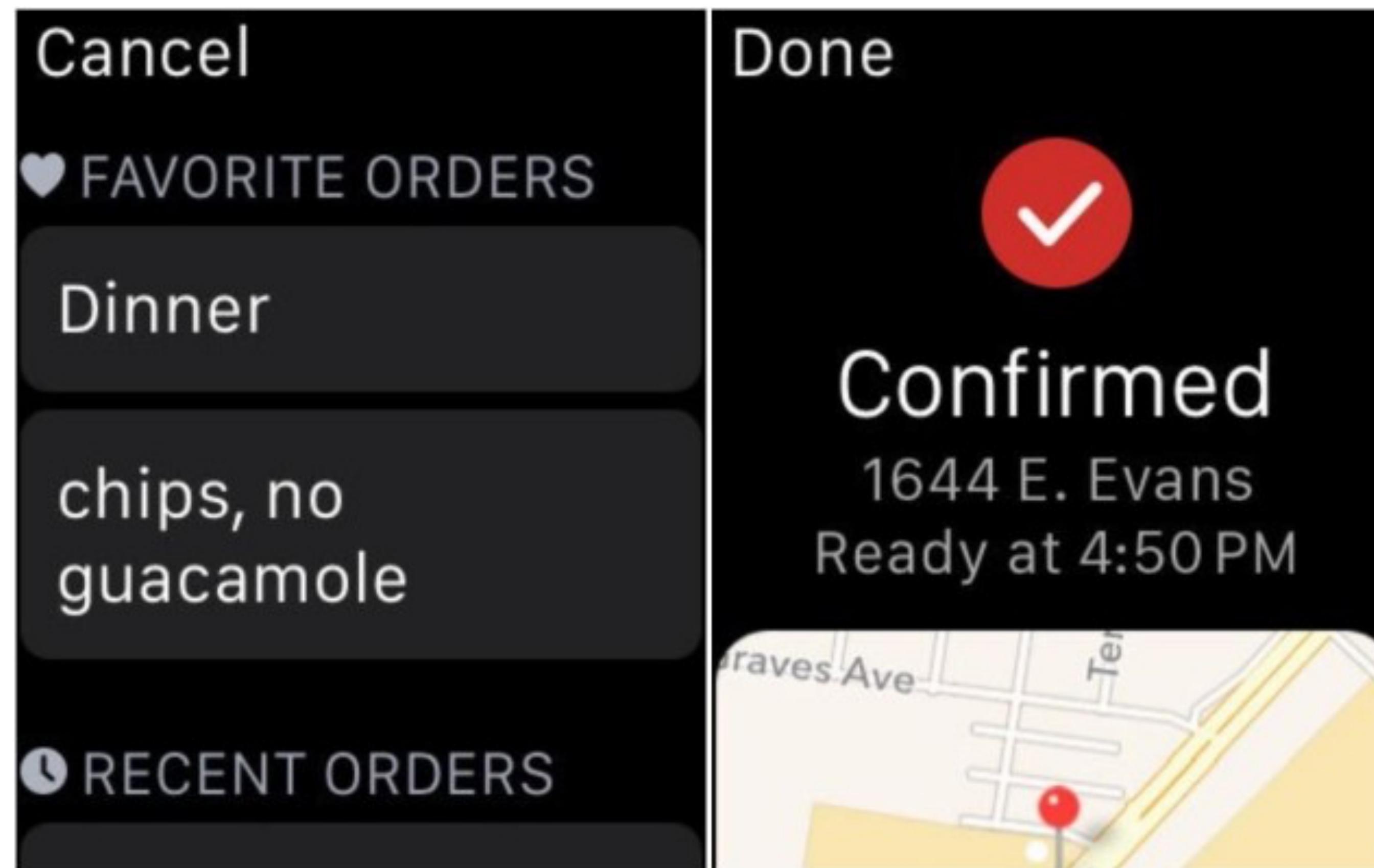
USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS



GLANCES

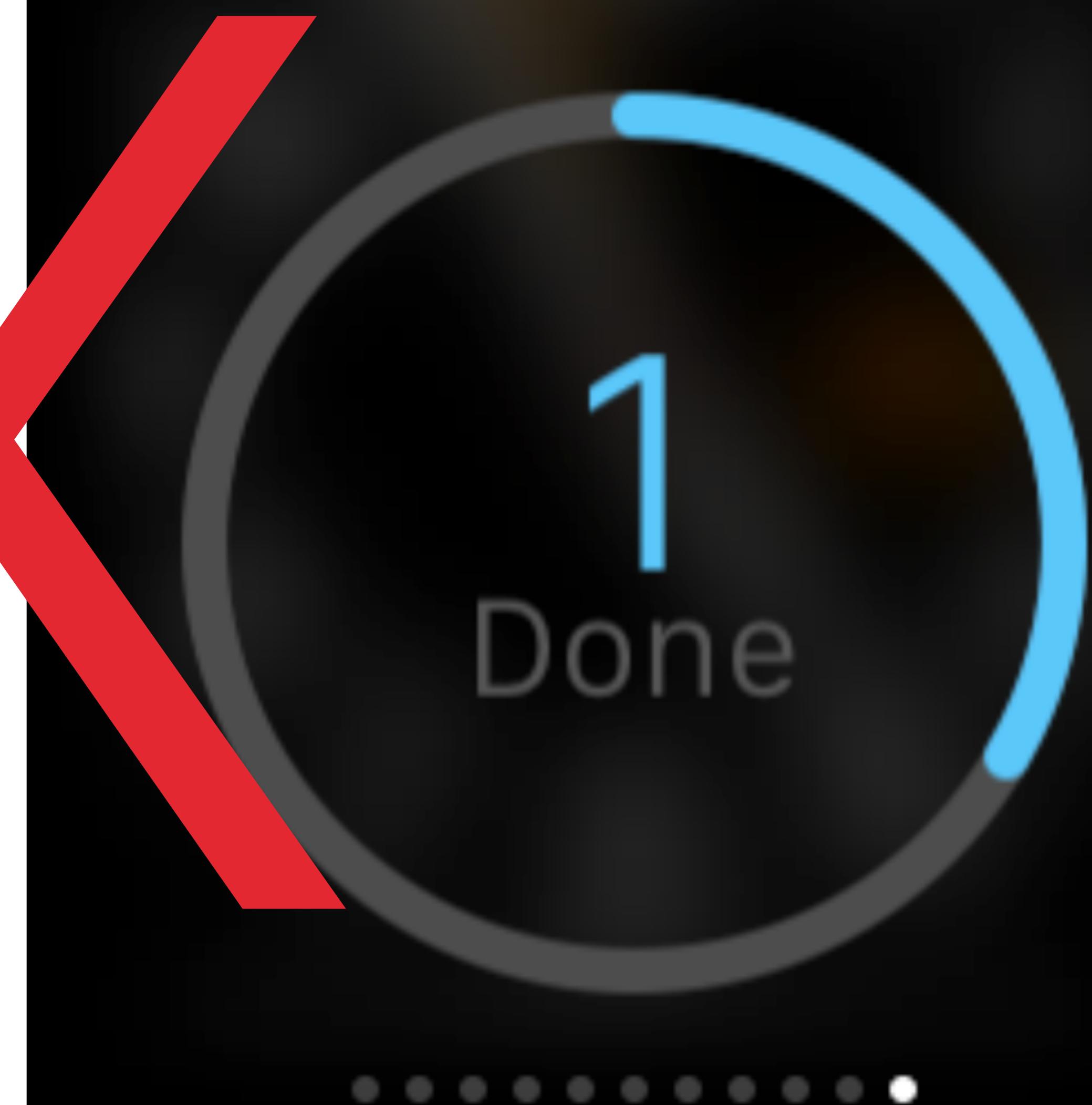


USER INTERACTIONS

SUBTITLE

- Glances
 - A focused interface that you use to display your app's most important information
 - Glances are nonscrolling and read-only
 - Cannot contain buttons, switches, or other interactive controls
 - Tapping a glance launches your WatchKit app
 - “Not to be used as an app launcher”

Today
2 items left



USER INTERACTIONS

GLANCES HAS BEEN REPLACED BY THE DOCK



NOTIFICATIONS

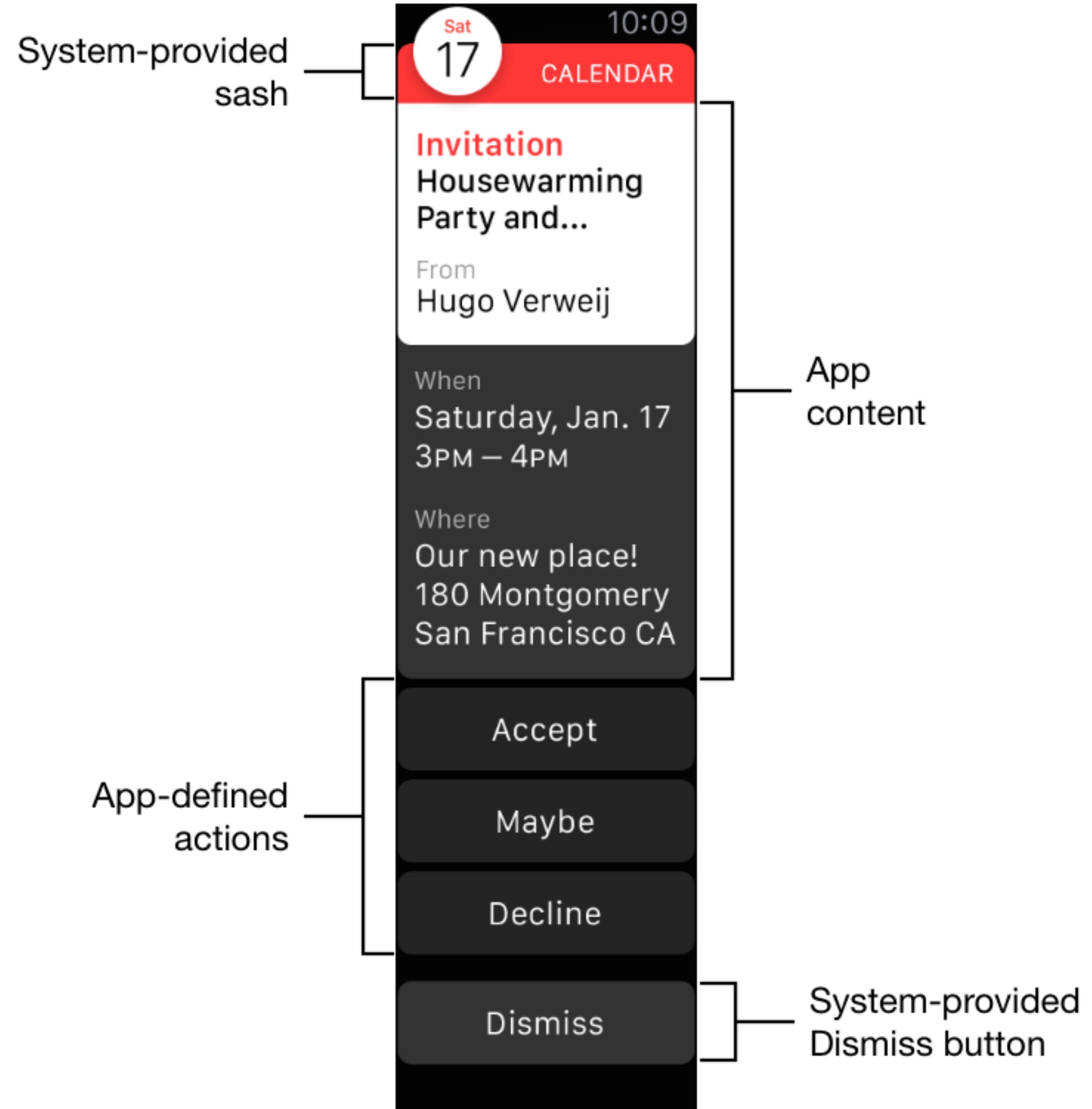
USER INTERACTIONS

- Custom Notifications
 - Works with iPhone to display local and remote notifications
 - Providing custom notification interfaces
 - Short
 - Long



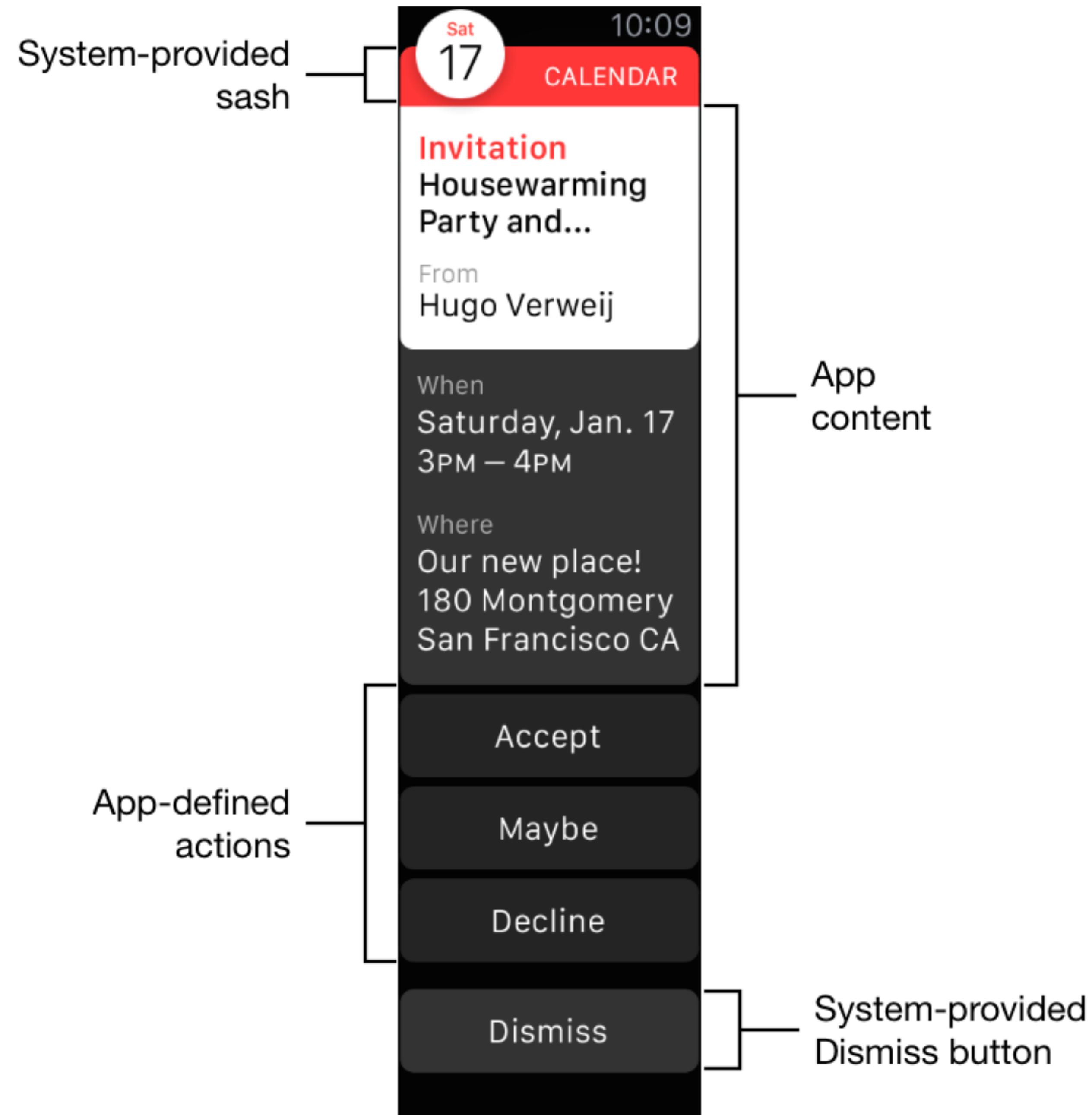
USER INTERACTIONS

- Long looks notifications
- Actionable notifications to increase functionality



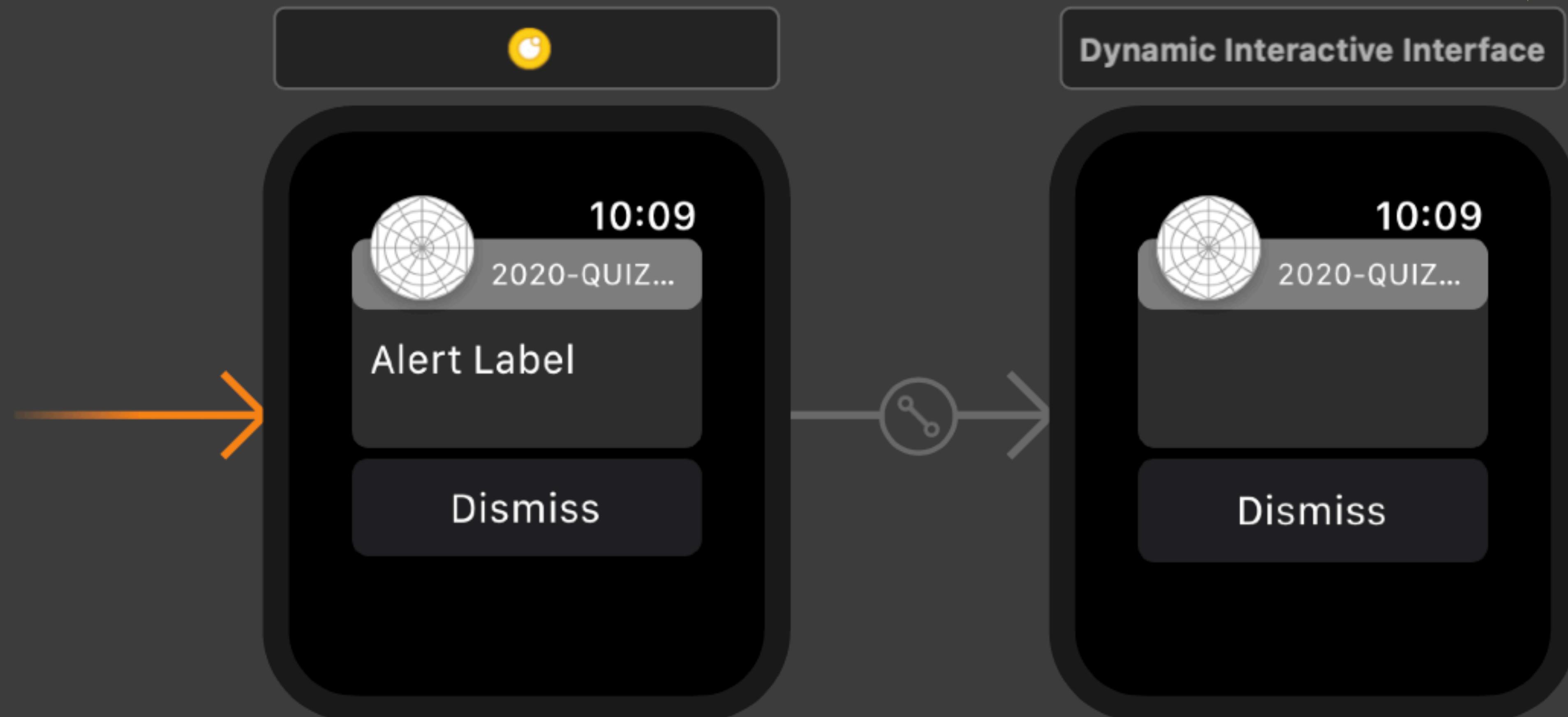
USER INTERACTIONS

- watchOS works with UNNotification framework
- Schedule and handle location notification on watch using extension
 - Time, location based
- Handle remote notifications delivered to watch



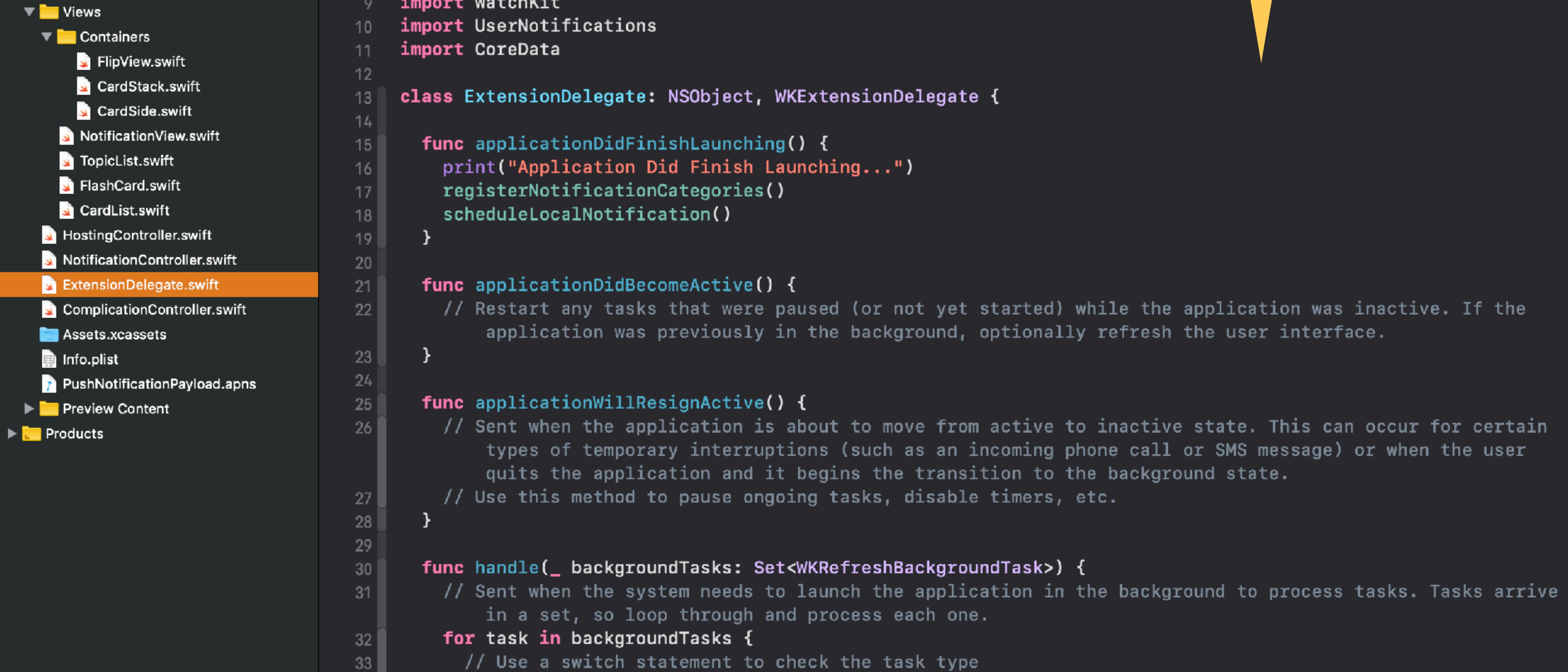
USER INTERACTIONS

Templates from
Xcode



USER INTERACTIONS

Templates from Xcode



```
9 import WatchKit
10 import UserNotifications
11 import CoreData
12
13 class ExtensionDelegate: NSObject, WKExtensionDelegate {
14
15     func applicationDidFinishLaunching() {
16         print("Application Did Finish Launching...")
17         registerNotificationCategories()
18         scheduleLocalNotification()
19     }
20
21     func applicationDidBecomeActive() {
22         // Restart any tasks that were paused (or not yet started) while the application was inactive. If the
23         // application was previously in the background, optionally refresh the user interface.
24     }
25
26     func applicationWillResignActive() {
27         // Sent when the application is about to move from active to inactive state. This can occur for certain
28         // types of temporary interruptions (such as an incoming phone call or SMS message) or when the user
29         // quits the application and it begins the transition to the background state.
30         // Use this method to pause ongoing tasks, disable timers, etc.
31     }
32
33     func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
34         // Sent when the system needs to launch the application in the background to process tasks. Tasks arrive
35         // in a set, so loop through and process each one.
36         for task in backgroundTasks {
37             // Use a switch statement to check the task type
38         }
39     }
40 }
```

COMPLICATIONS

USER INTERACTIONS

COMPLICATIONS



USER INTERACTIONS

COMPLICATIONS

SUBTITLE

- Small visual elements that appear on the watch face
 - Customizable
 - Always visible



USER INTERACTIONS



USER INTERACTIONS

COMPLICATIONS

SUBTITLE

- The term *complication* comes from watch making, where the addition of features added complexity to the watch construction



Customize

USER INTERACTIONS

COMPLICATIONS

Apple recommends that all Watch apps include a complication, even if that complication only acts as a button to launch the app. For information about complications and how to implement them, see [Complication Essentials](#).



USER INTERACTIONS

COMPLICATIONS

- Number of complications varies depending on the watch face

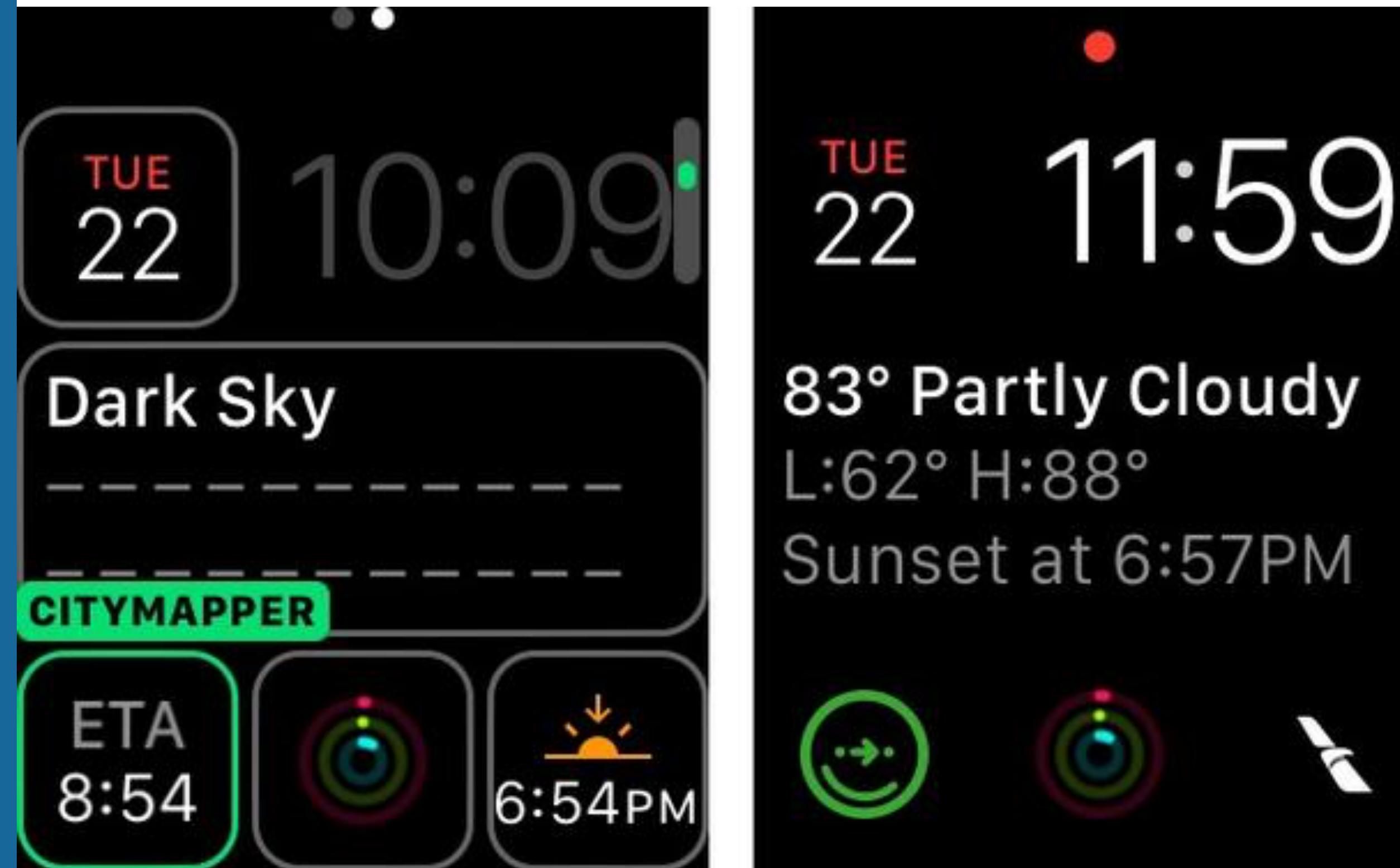


Nike Run Club

USER INTERACTIONS

COMPLICATIONS

- Apps whose complications are selected on the face get VIP treatment
 - Stays in memory; fast launch
 - Receives more time to execute background tasks
 - Receive background updates (at least 2x/hour)



USER INTERACTIONS

COMPLICATIONS

- Different templates for the different watch styles

Graphic Circular

These templates display text, gauges, and full-color images in small circular areas on Infograph and Infograph Modular watch faces. Some of the templates also support multicolor text. The Infograph and Infograph Modular faces are only available on Apple Watch Series 4.



Closed Gauge Image



Closed Gauge Text



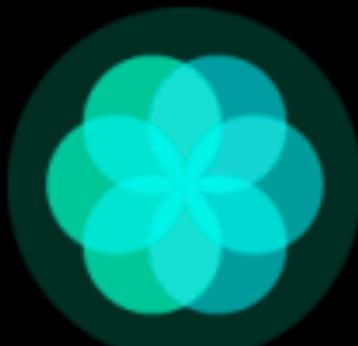
Open Gauge Image



Open Gauge Text



Open Gauge Range



Image

USER INTERACTIONS

COMPLICATIONS

The screenshot shows the Xcode project structure and target settings for a WatchKit Extension named "2020-quiz-starter".

Project Structure:

- Info.plist
- 2020-quiz-starter WatchKit Extension
 - Model
 - FlashCardModel.swift
 - Views
 - Containers
 - FlipView.swift
 - CardStack.swift
 - CardSide.swift
 - NotificationView.swift
 - TopicList.swift
 - FlashCard.swift
 - CardList.swift
 - HostingController.swift
 - NotificationController.swift
 - ExtensionDelegate.swift
 - ComplicationController.swift
 - Assets.xcassets
 - Info.plist
 - PushNotificationPayload.apns
- Preview Content
- Products

TARGETS

- 2020-quiz-starter
- 2020-quiz-starter...
- 2020-quiz-starter...

Display Name: 2020-quiz-starter WatchKit Extension
Bundle Identifier: mobi.uchicago.-020-quiz-starter.watchkitapp.watchkitapp
Version: 1.0
Build: 1

Deployment Info: Deployment Target: 6.2

Complications Configuration:

- Data Source Class: \$(PRODUCT_MODULE_NAME).ComplicationCor
- Supported Families:
 - Modular Small
 - Modular Large
 - Utilitarian Small
 - Utilitarian Small Flat
 - Utilitarian Large
 - Circular Small
 - Extra Large
 - Graphic Corner
 - Graphic Bezel
 - Graphic Circular
 - Graphic Rectangular

Complications Group: Complication

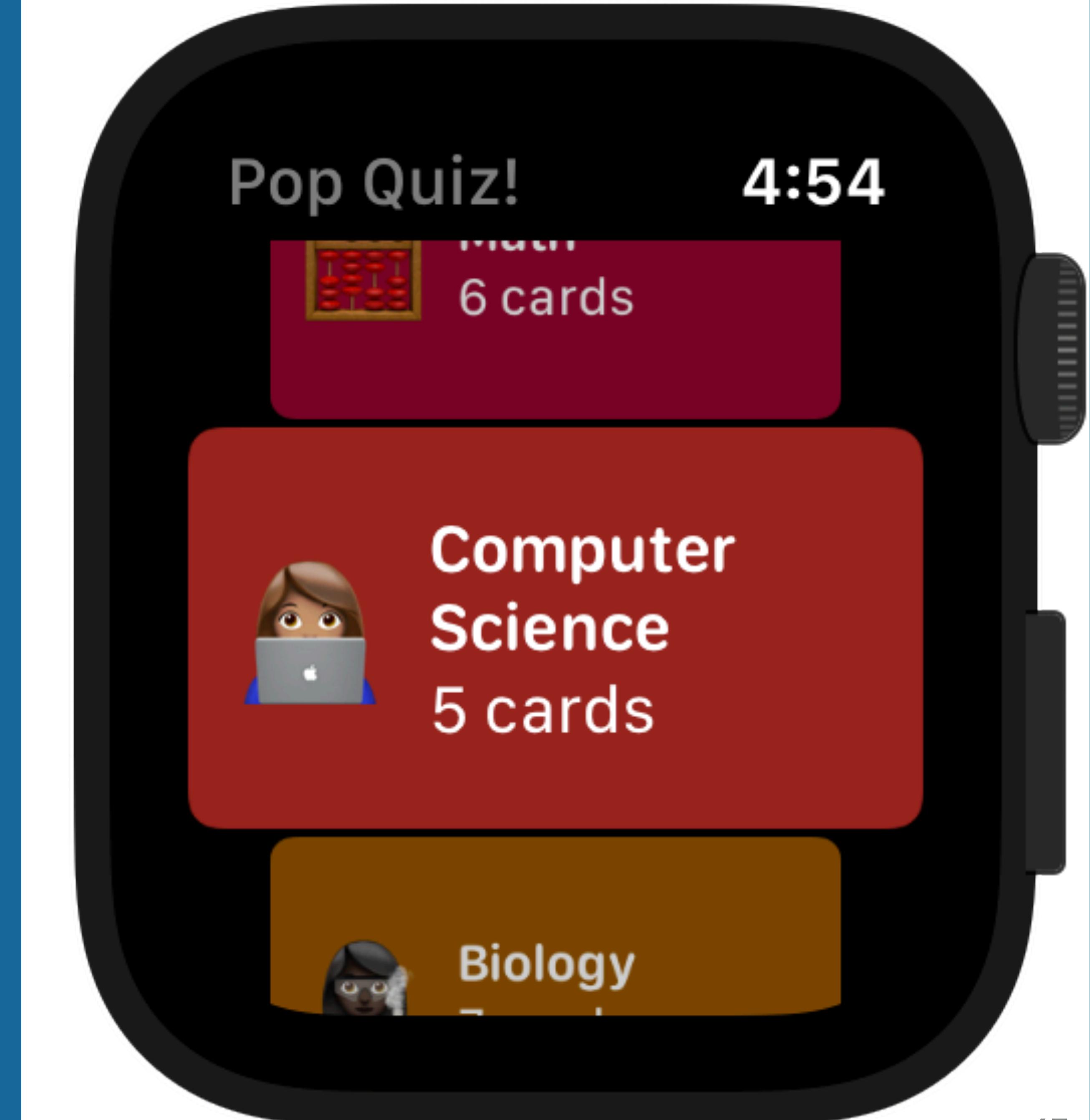
Supported complications

ASSIGNMENT 3

PART 2

ASSIGNMENT 3

- Pop Quiz!
- Independent watchOS application for studying flashcards



ASSIGNMENT 3

SUBTITLE

- Features
 - Core Data model for data persistence
 - Custom notifications
 - From APNS
 - Local notification
 - Complication
 - SwiftUI



ASSIGNMENT 3

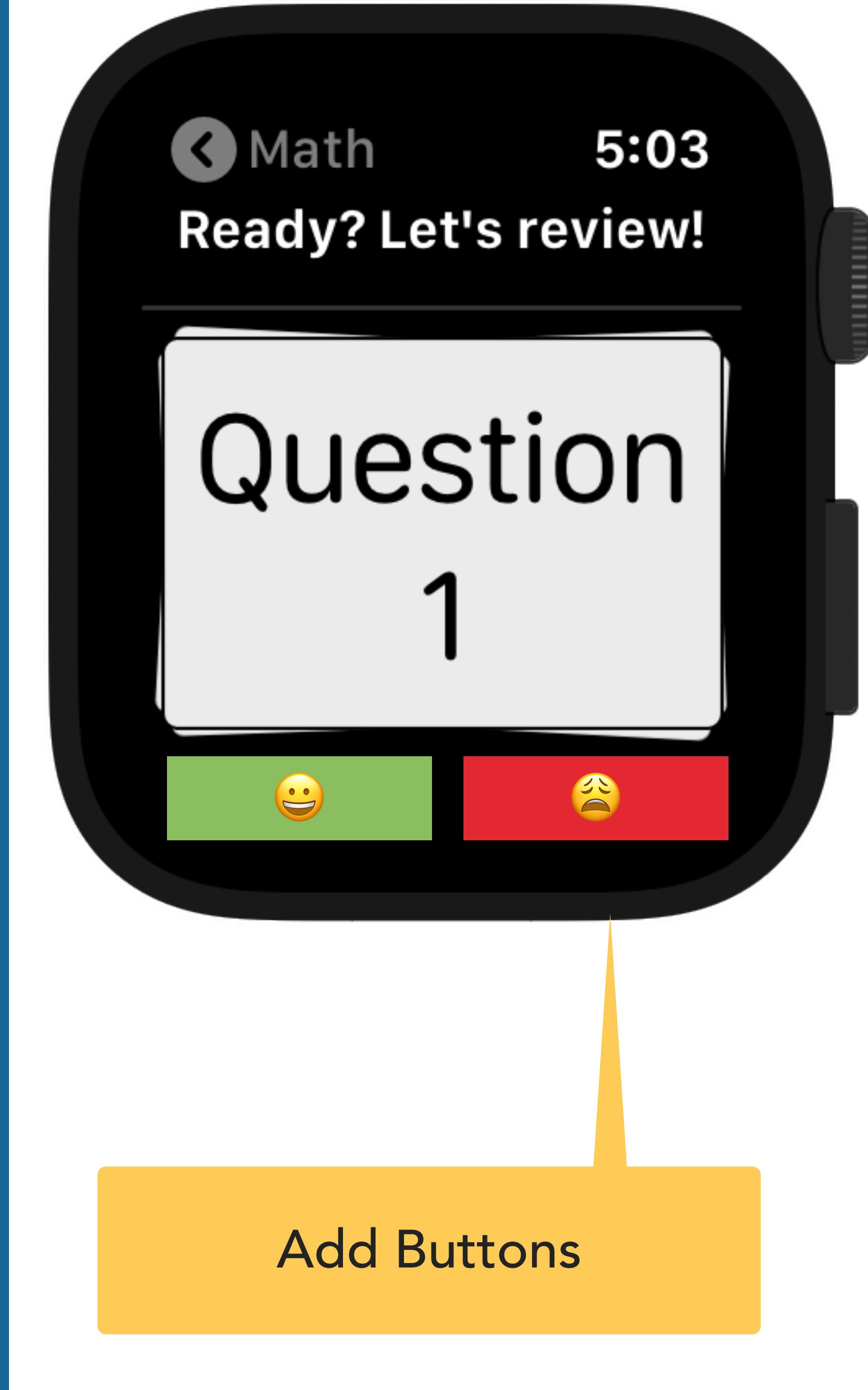
SUBTITLE

- Template app that provides core functionality and design
 - Feel free to change anything



ASSIGNMENT 3

- Change data model to use CoreData
 - 3 subjects worth of data
 - 10 questions per subject
- Add buttons for "Correct" and "Incorrect"
- Keep statistics on correct/incorrect



ASSIGNMENT 3

SUBTITLE

- Update Complication to use a "Graphic Circle Closed Gauge Text"
 - Show percent answered correctly
 - Gauge completion is percent correct
- Add a background update to refresh the percentages



ASSIGNMENT 3

SUBTITLE

- Update Notification to handle CoreData model
- Local notification should randomly select an incorrect question from last session
- Handle statistics on the "Correct/Incorrect" actions



ASSIGNMENT 3

SUBTITLE

- Add all necessary app icons





apple WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

UIKIT WATCHKIT APPS

INSPIRATION

WATCHKIT APPS



WATCHKIT APPS



WATCHKIT APPS



WATCHKIT APPS



USER INTERACTION ELEMENTS

WATCHKIT APPS

SUBTITLE

- Very familiar collection of interface objects
- Objects are just a proxy for the views in your storyboard
 - MVC to the extreme

WKInterfaceController	UIViewController
WKUserNotificationInterfaceController	UIApplicationDelegate + UIAlertController
WKInterfaceDevice	UIDevice
WKInterfaceObject	UIView
WKInterfaceButton	UIButton
WKInterfaceDate	UILabel + NSDateFormatter
WKInterfaceGroup	UIScrollView
WKInterfaceImage	UIImageView
WKInterfaceLabel	UILabel
WKInterfaceMap	MKMapView
WKInterfaceSeparator	UITableView.separatorColor / .separatorStyle
WKInterfaceSlider	UIStepper + UISlider
WKInterfaceSwitch	UISwitch
WKInterfaceTable	UITableView
WKInterfaceTimer	UILabel + NSDateFormatter + NSTimer

WATCHKIT APPS

Labels

Labels use static text to convey short messages and are one of the most common elements in your app. They can span multiple lines and can be updated programmatically.

When you design a label, focus first on legibility. Use lighter colors for label text and use Dynamic Type to ensure that the text is sized appropriately. The built-in text styles offer the best legibility and automatically support Dynamic Type. If custom typefaces are necessary, avoid using ones that are overly stylized.

See [Typography](#) for more information about using text in an Apple Watch app.



WATCHKIT APPS

Images

An image element displays a single image or an animated sequence of images. Images may be in any format supported by iOS, but the preferred format is PNG. Animated sequences can be started and stopped programmatically.

Size images for 38mm and 42mm displays, respectively. You can use a single image resource for both display sizes as long as it maintains clarity.

Create images at 2x resolution. Apple Watch has a Retina display, so there is no need to create standard resolution images. Include "@2x" in image names.



WATCHKIT APPS

Groups

Groups can help you lay out other elements, such as images and labels. A group has attributes for specifying position, size, margins, and other layout-related properties. It also has a background image, color, and corner radius that can transform it into a visual element.

Use groups to arrange items horizontally or vertically. All items in a group are laid out in the same horizontal or vertical line. Use the group's inter-item spacing attribute to add space between items. Use the group's margins to add space between the group and its surrounding items.

Nest groups to mix horizontal and vertical content. You can use nesting to achieve specific layouts for your content. For example, you can place multiple vertical groups inside a single horizontal group.



WATCHKIT APPS

Tables

Tables present rows of content in a single column. A table row is dynamically configurable and its contents can be changed at any time. Tables are inherently scrollable, support various interactions, and can be assigned a background color or image.

Use row types consistently. Although a table can contain multiple row types, it should present a consistent overall appearance. Start with the row type you use for the table content, and add more as needed to support other types of content, such as headers or footers. Always use each row type for its intended purpose. For example, don't display content in a row type designed for headers or footers.

Limit the number of table rows to about 20. Display the most important rows at the top and let the wearer view more on demand. Fewer rows are easier to scan quickly.

Don't embed tables inside groups. Tables resize dynamically based on the number of rows they contain; they ignore height restrictions placed on them by groups.



WATCHKIT APPS

Buttons

A button performs an app-specific action. Buttons have customizable backgrounds and rounded corners. They can contain a label, an image, or a group object. The standard corner radius for a button is six points.

The background of the button is called the platter. You can assign a color or an image to the platter to change its appearance.

Create buttons that span the width of the screen. Full-width buttons look better and are easier to tap. If two buttons must share the same horizontal space, use the same height for both and use images or short text labels for each button's content.

Use the same height for vertical stacks of one- and two-line text buttons. As much as possible, use identical button heights for visual consistency. However, for buttons that contain dramatically different content, using different heights can improve the look of individual buttons.

Use the standard corner radius. It promotes a consistent visual style across apps and reinforces the intent of buttons.



WATCHKIT APPS

Switches

A switch lets people choose between two mutually exclusive choices or states, such as yes/no. A label must always be displayed next to a switch to specify which property the switch affects.



WATCHKIT APPS

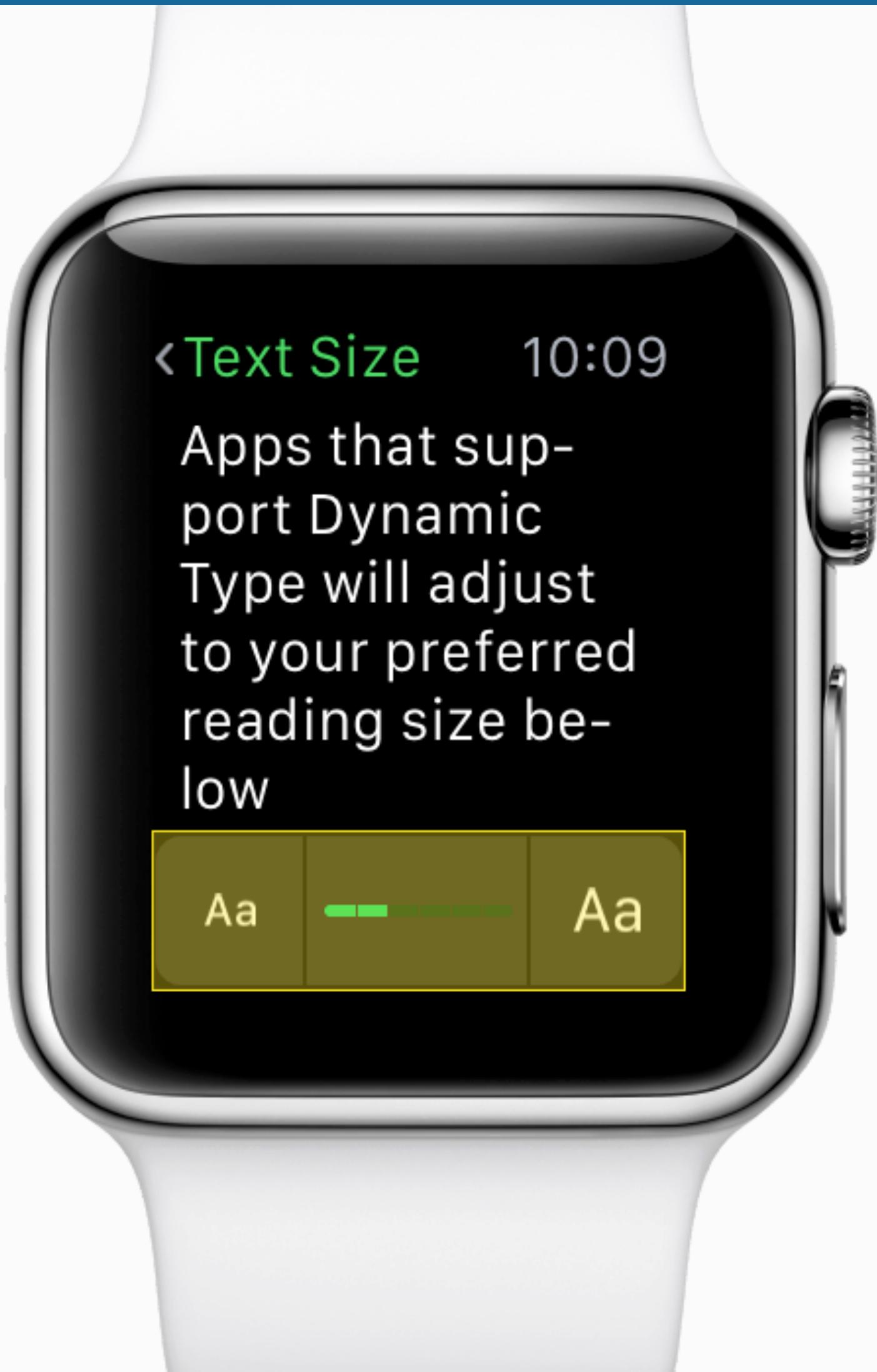
Sliders

A slider lets people make adjustments to a value by tapping images at either end of the slider bar.

A slider displays its value as a set of steps or as a continuous bar. It always increases and decreases its value by a predetermined amount.

Sliders are not meant to display numerical values.

Use custom images to communicate what the slider does. The system displays plus and minus signs by default.

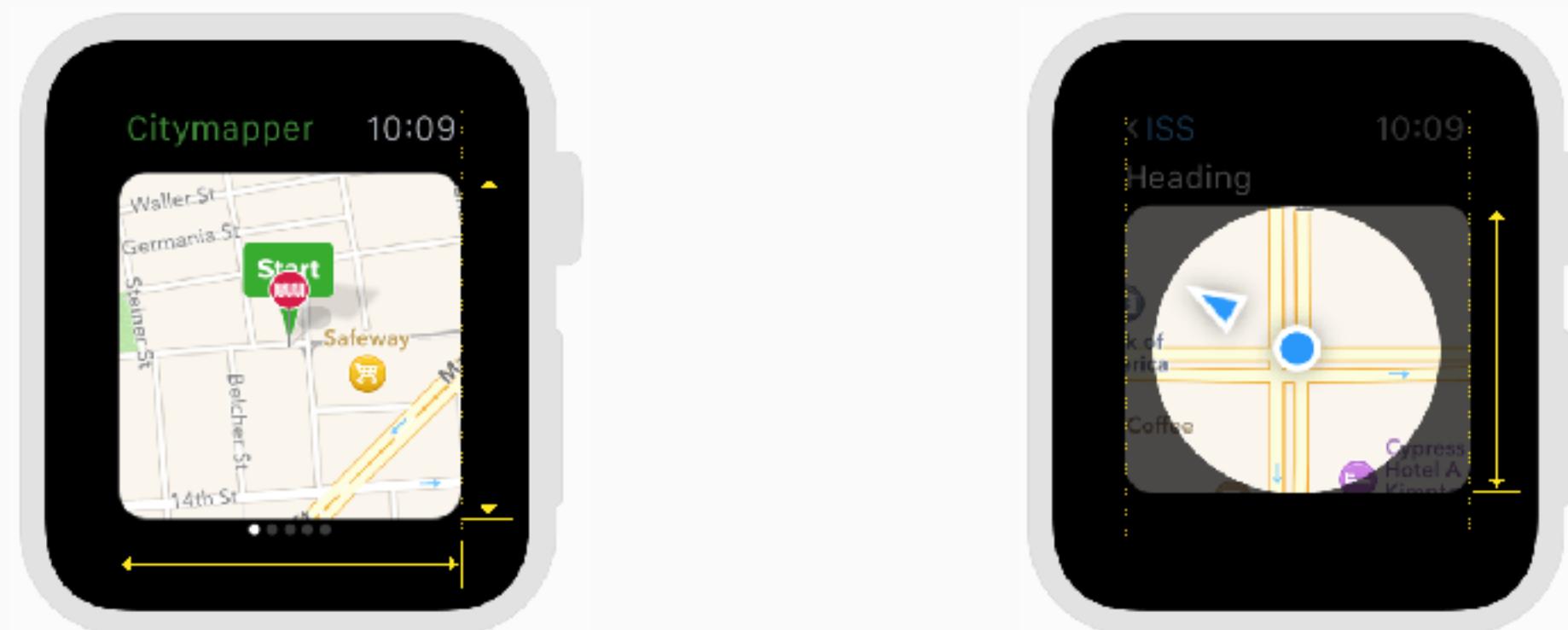


WATCHKIT APPS

Maps

Maps are static snapshots of geographic locations. You place a map in your interface at design time, but you must configure the displayed region at runtime. The displayed region is not interactive, but tapping a map will open the Maps app on Apple Watch.

You can annotate a map to highlight points of interest or other relevant information. A map can display the standard green, red, and purple pins. It can also display custom images. The system lets you add up to five annotations to a map.



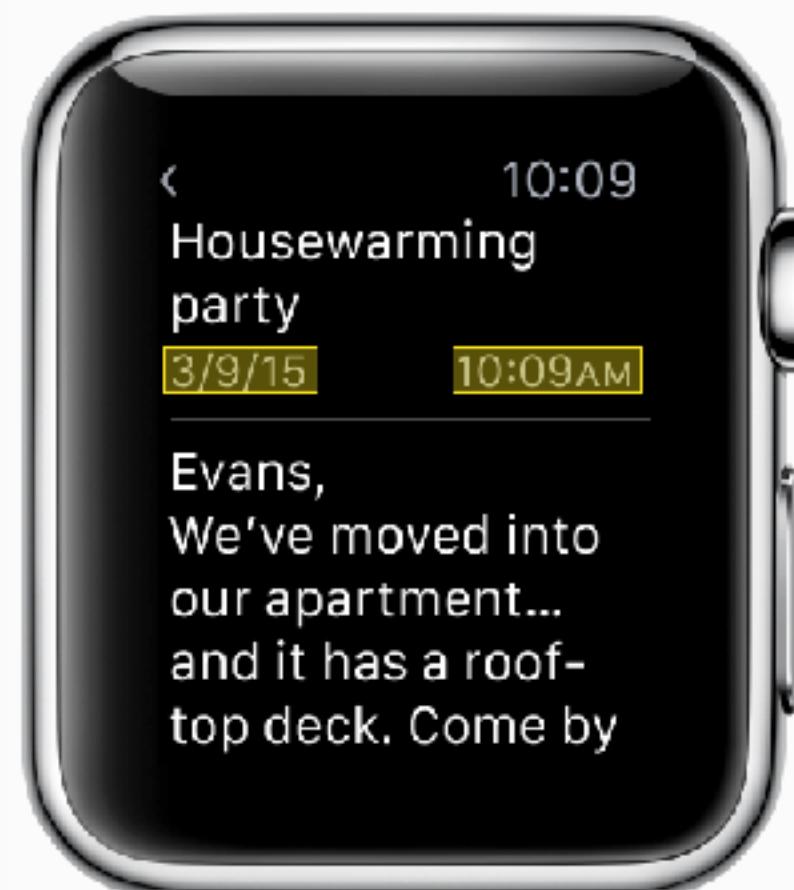
Size the map element to fit the screen. The wearer should be able to see the entire map element on the Apple Watch display without scrolling the screen.

Configure the displayed map region to be the smallest area that encompasses the points of interest. The contents of the map element itself do not scroll, so all intended content must be enclosed by the specified map region.

WATCHKIT APPS

Date & Timer Labels

Date and timer labels display real-time values on Apple Watch.



A date label displays the date, the time, or a combination of both. It can be configured to use a variety of formats, calendars, and time zones. After you configure it, a date label updates its value without further input from your app.



A timer label displays a precise countdown or count-up timer. It can be configured to display its count value in a variety of formats. After you configure it, a timer label counts down or up without further input from your app.

WATCHKIT APPS

Menus

Firm presses on the Apple Watch display cause the current screen's menu (if any) to appear. A menu can display up to four relevant actions for the current screen without taking away space from your interface.

Include a menu when the current screen has relevant actions.

Menus are optional. If no menu is present, the system plays an animation when the wearer presses firmly on the display.

Use a label and an icon to convey the purpose of each menu action.

Both the label and the icon are required. Labels are limited to two lines, so the text should be short.

For information

No visual design in IB; need to toggle presses in simulator



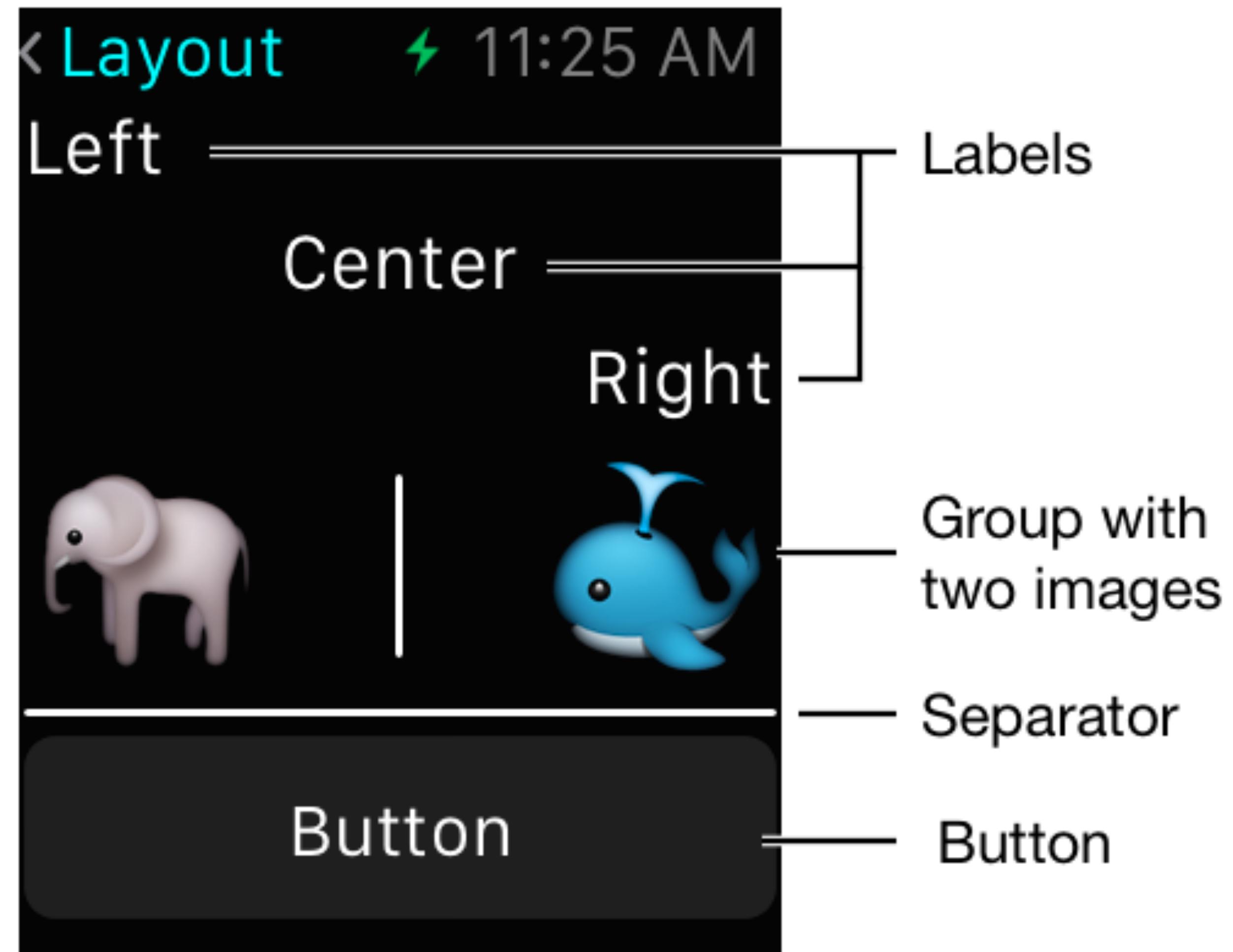
Play

UI ESSENTIALS

WATCHKIT APPS

SUBTITLE

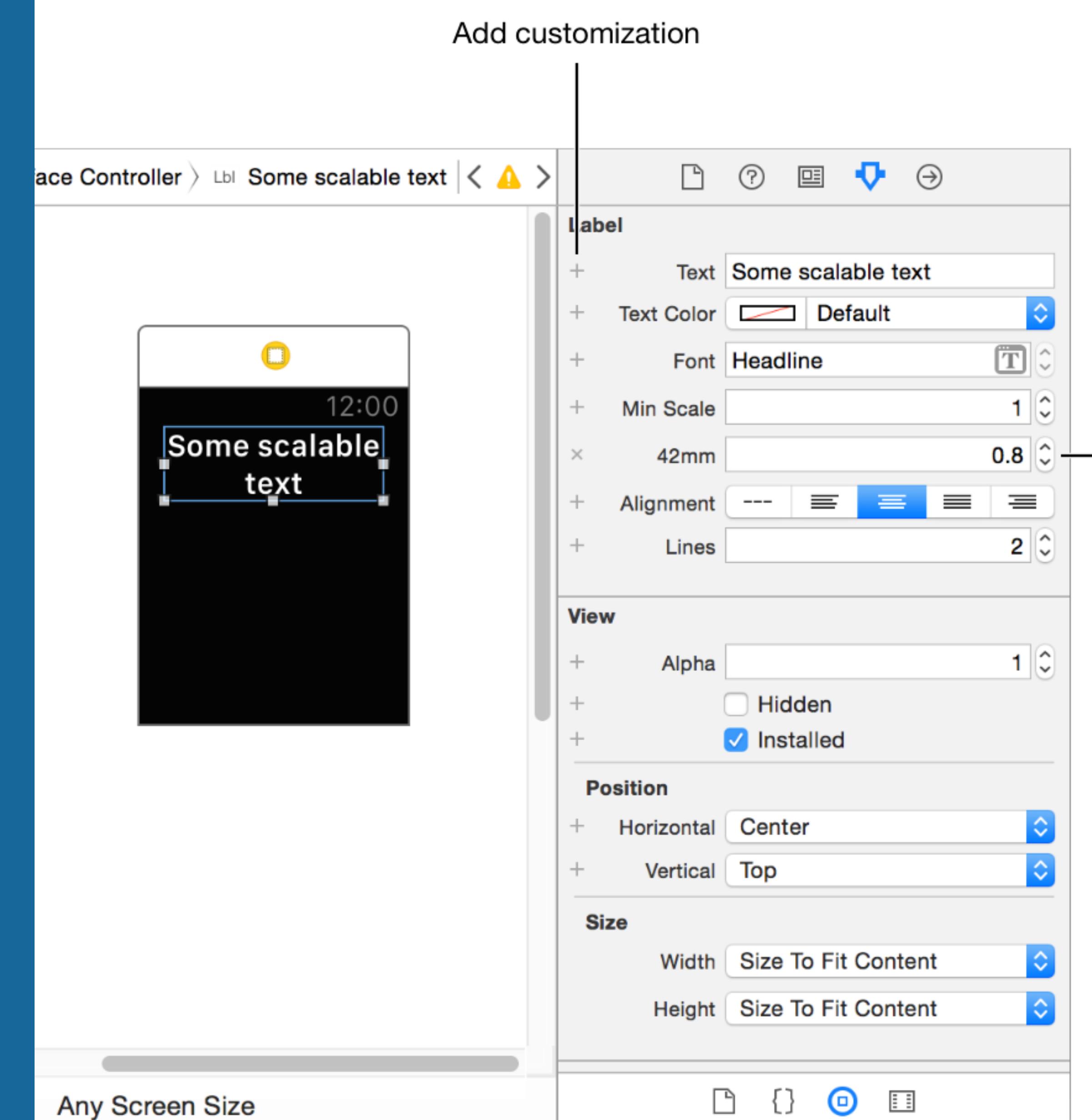
- WatchKit uses a different layout model
 - Xcode arranges items for you
 - Highly constrained, yet liberating
- Interface groups are used to define more complex layouts



WATCHKIT APPS

SUBTITLE

- What's 4mm?
 - Customize interface based on device (if you want)
 - Apple recommends same interface for all watches



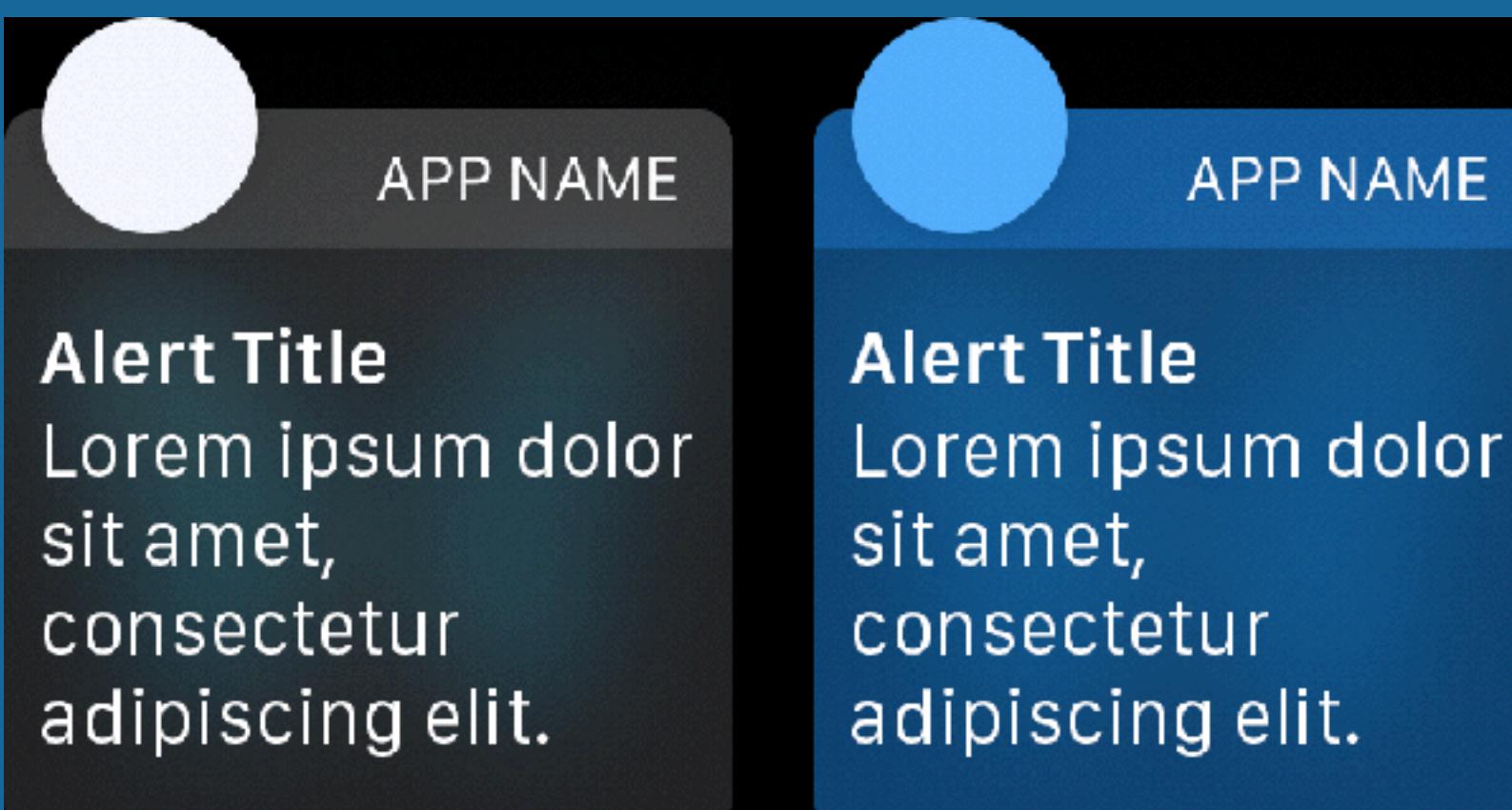
Change display mode

WATCHKIT APPS

- At runtime, an interface controller can make the following modifications to the objects in its corresponding storyboard scene:
 - Set or update data values
 - Change the visual appearance of objects that support such modifications
 - Change the size of an object
 - Change the transparency of an object
 - Show or hide an object
- Note: Objects have no getters, you need to keep track of state on your own

WATCHKIT APPS

- Every WatchKit app has an associated key color, which is applied to the following UI elements
 - The title string in the status bar
 - The app name in short-look notifications
- An app's key color is stored in the Global Tint property of an app's storyboard



WATCHKIT APPS

The screenshot shows the Xcode interface for a WatchKit application. On the left is the Project Navigator with files like AppDelegate.swift, ViewController.swift, Main.storyboard, and various xcassets and nib files. Below it is the WatchKit Extension and App groups, each containing InterfaceController.swift, NotificationController.swift, GlanceController.swift, and Images.xcassets. The center-left shows the Main Entry Point list with Glance Interface Controller, Static Notification Inter..., and Notification Controller... selected. To the right are two storyboard previews: a Main screen with a headerLabel and a Glance screen showing two Groups. The right side of the interface contains the Utilities panel with tabs for Location, Interface Builder Document, Localization, Target Membership, and Text Settings.

Main Entry Point

- Glance Interface Contr...
- Static Notification Inter...
- Notification Controller...

Main

headerLabel

Glance

Glance Interface

Group

Group

Location Relative to Group
Choose Containing Folder
Full Path /Users/tbinkowski/Google Drive/g-Teaching/MPCS51032-2015-Spring/MyDemos/WatchInterface/WatchInterface WatchKit App

Dev Region /Users/tbinkowski/Google Drive/g-Teaching/MPCS51032-2015-Spring/MyDemos/WatchInterface/WatchInterface WatchKit App/Base.iproj/Interface.storyboard

Interface Builder Document

- Opens in Default (6.2)
- Builds for Project Deployment Tar...

Global Tint Default

Localization

- Base
- English

Target Membership

- WatchInterface
- WatchInterfaceTests
- WatchInterface WatchKit Extension
- WatchInterface WatchKit App**

Text Settings

WATCHKIT APPS

R 250
G 17
B 79

17% Opacity

R 255
G 59
B 48

17% Opacity

R 255
G 149
B 0

15% Opacity

R 255
G 230
B 32

14% Opacity

R 4
G 222
B 113

14% Opacity

R 0
G 245
B 234

13% Opacity

R 90
G 200
B 250

15% Opacity

R 32
G 148
B 250

17% Opacity

R 120
G 122
B 255

20% Opacity

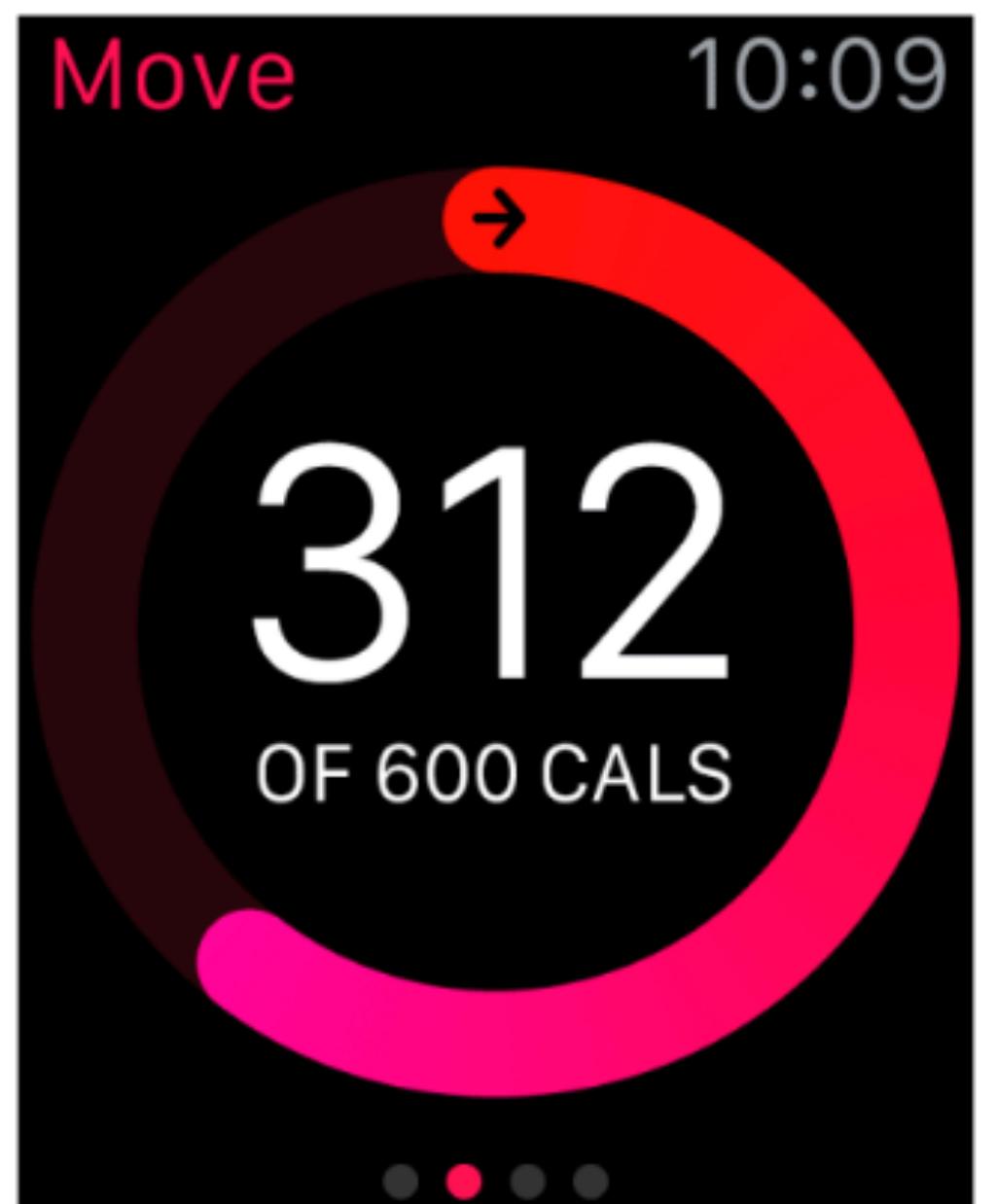
R 242
G 244
B 255

14% Opacity

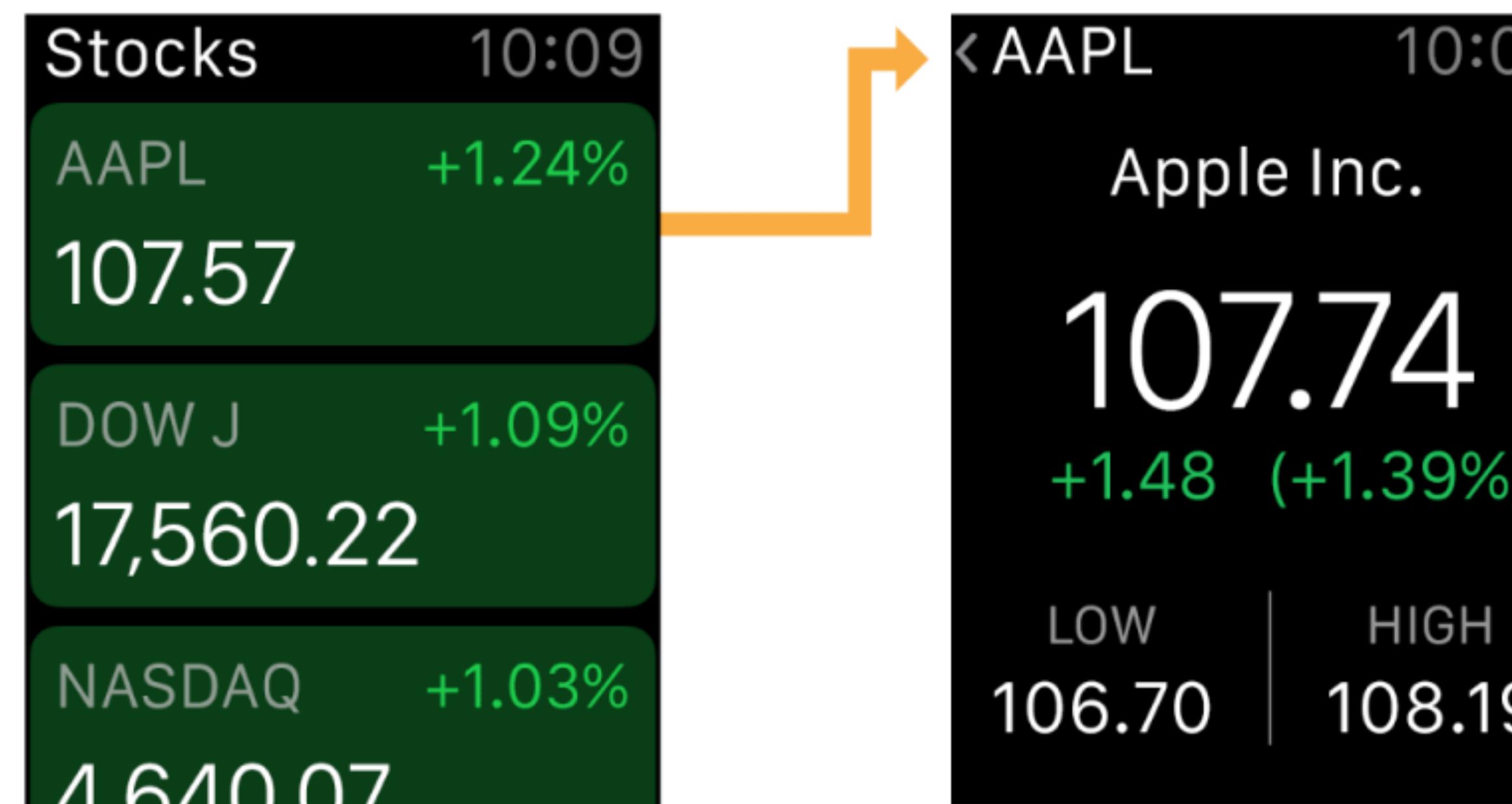
INTERFACE NAVIGATION

WATCHKIT APPS

Page-Based



Hierarchical



WATCHKIT APPS

- Implementing a Page-Based Interface
 - reloadRootControllersWithNames:contexts:
 - becomeCurrentPage
- Implementing a Hierarchical Interface
 - pushControllerWithName:context:
 - popController
- Presenting Interface Controllers Modally
 - presentControllerWithName:context:
 - presentControllerWithNames:contexts:

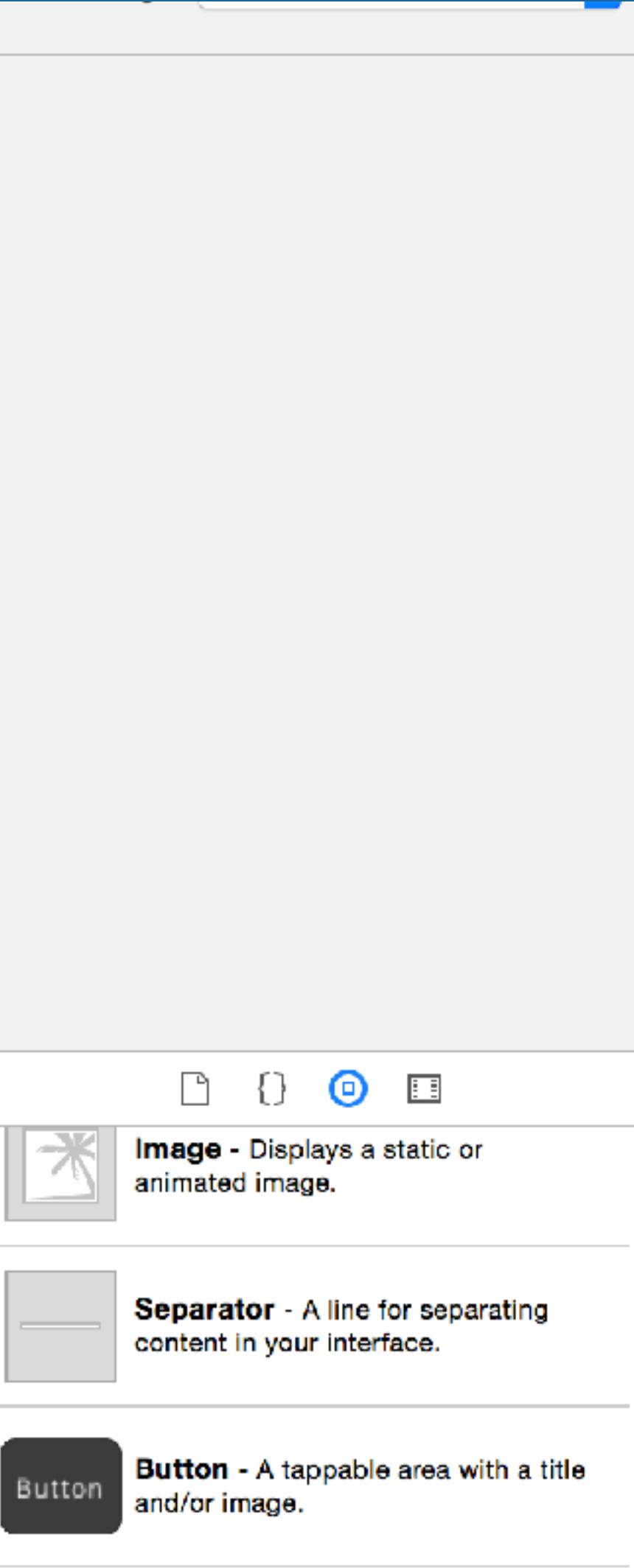
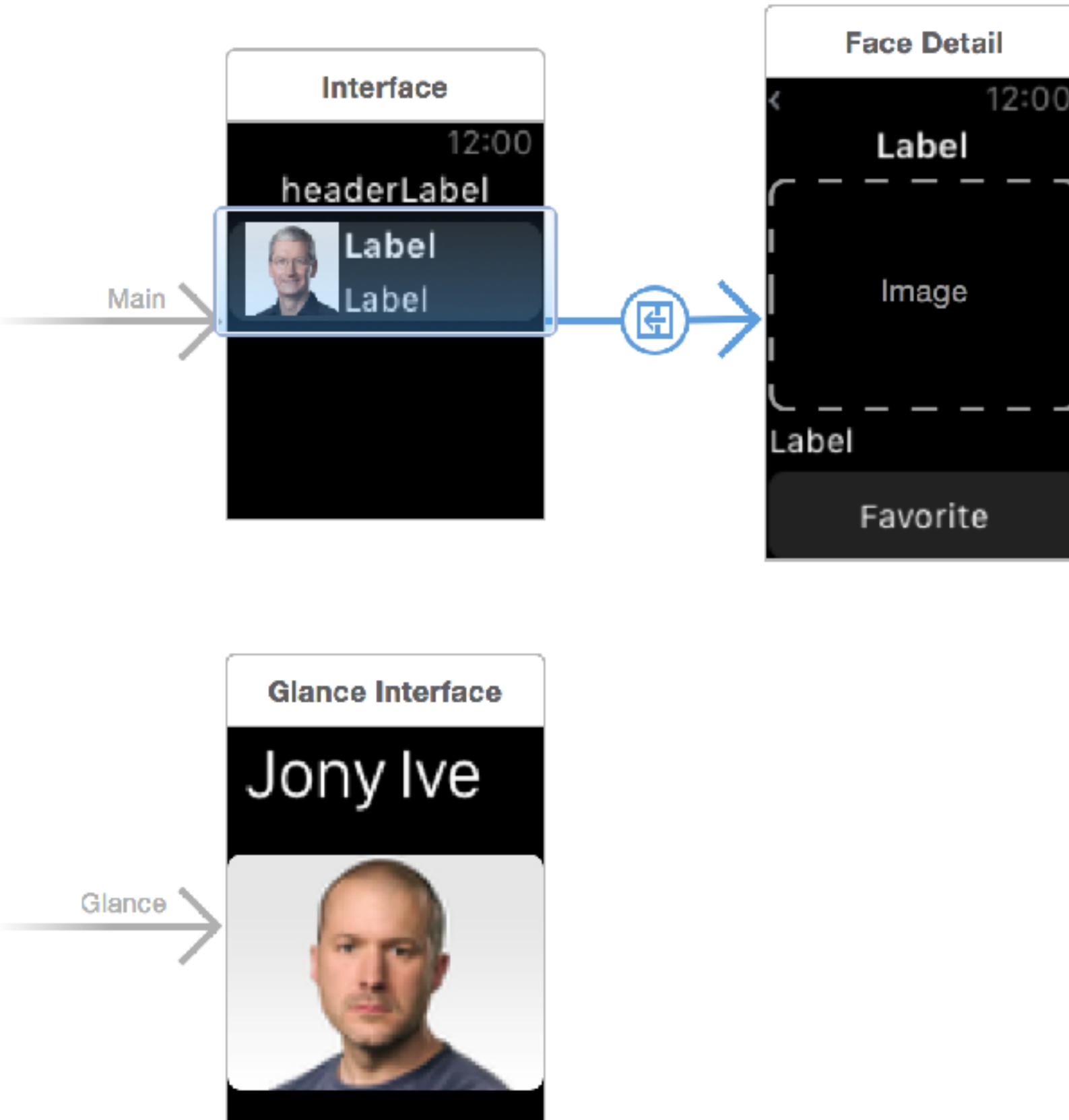
WATCHKIT APPS

The screenshot shows the Xcode interface for a WatchKit app. On the left is the Project Navigator with the following file structure:

- AppDelegate.swift
- ViewController.swift
- Main.storyboard
- Images.xcassets
- LaunchScreen.xib
- Supporting Files
- WatchInterfaceTests
- WatchInterface WatchKit Extension
 - FaceRow.swift
 - InterfaceController.swift
 - FaceDetailController.swift
 - NotificationController.swift
 - GlanceController.swift
 - Images.xcassets
 - Supporting Files
 - Info.plist
 - PushNotificationPayload.apns
- WatchInterface WatchKit App
 - Interface.storyboard
 - Images.xcassets
 - Supporting Files
 - Info.plist
- InterfaceKit
 - InterfaceKit.h
 - Face.swift
 - DefaultsManager.swift
 - AppleExecutives.plist
 - Supporting Files
 - Info.plist
- InterfaceKitTests
- Products

Push segue to Face D...

- Face Detail Controller...
- Glance Interface Contr...
- Static Notification Inter...
- Notification Controller...



WATCHKIT APPS

```
49
50 // MARK: - Notifications
51 override func handleActionWithIdentifier(identifier: String?, forRemoteNotification
52     remoteNotification: [NSObject : AnyObject]) { ... }
53
54 // MARK: - Handoff Support
55 override func handleUserActivity(userInfo: [NSObject : AnyObject]?) { ... }
56
57 // MARK: - Table Support
58 func reloadTable() { ... }
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83 // MARK: - Seuge
84 override func contextForSegueWithIdentifier(segueIdentifier: String, inTable table:
85     WKInterfaceTable, rowIndex: Int) -> AnyObject? {
86
87     if segueIdentifier == "FaceDetailSegue" {
88         let name = faces[rowIndex].name
89         return name
90     }
91     return nil
92 }
```

WATCHKIT APPS

```
//  
8  
9 import Foundation  
10 import WatchKit  
11 import InterfaceKit  
12  
13 class FaceDetailController: WKInterfaceController {  
14  
15     /** Store the name explicitly since we can't get it from the labels */  
16     var cachedName: String = ""  
17  
18     // MARK: - Outlets and Actions  
19     @IBOutlet weak var name: WKInterfaceLabel!  
20     @IBOutlet weak var image: WKInterfaceImage!  
21     @IBOutlet weak var positionTitle: WKInterfaceLabel!  
22  
23     /**  
24         Add current user as favorite  
25     */  
26     @IBAction func tapFavorite() {  
27  
28  
29     // MARK: - Lifecycle  
30     override func awakeWithContext(context: AnyObject?) {  
31         super.awakeWithContext(context)  
32         cachedName = context as! String  
33     }  
34 }
```

You define the context



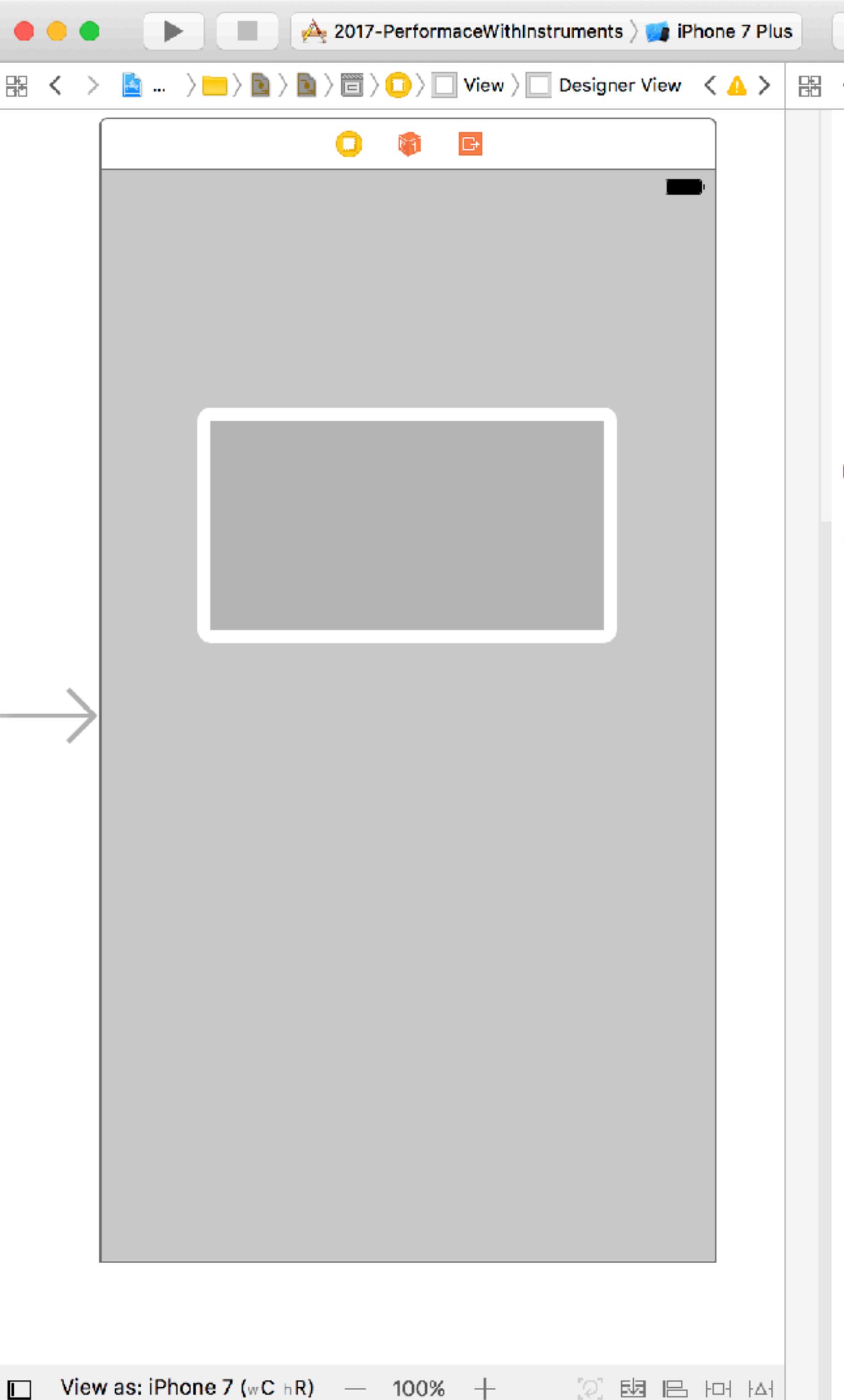
apple WATCH APPLICATION DEVELOPMENT

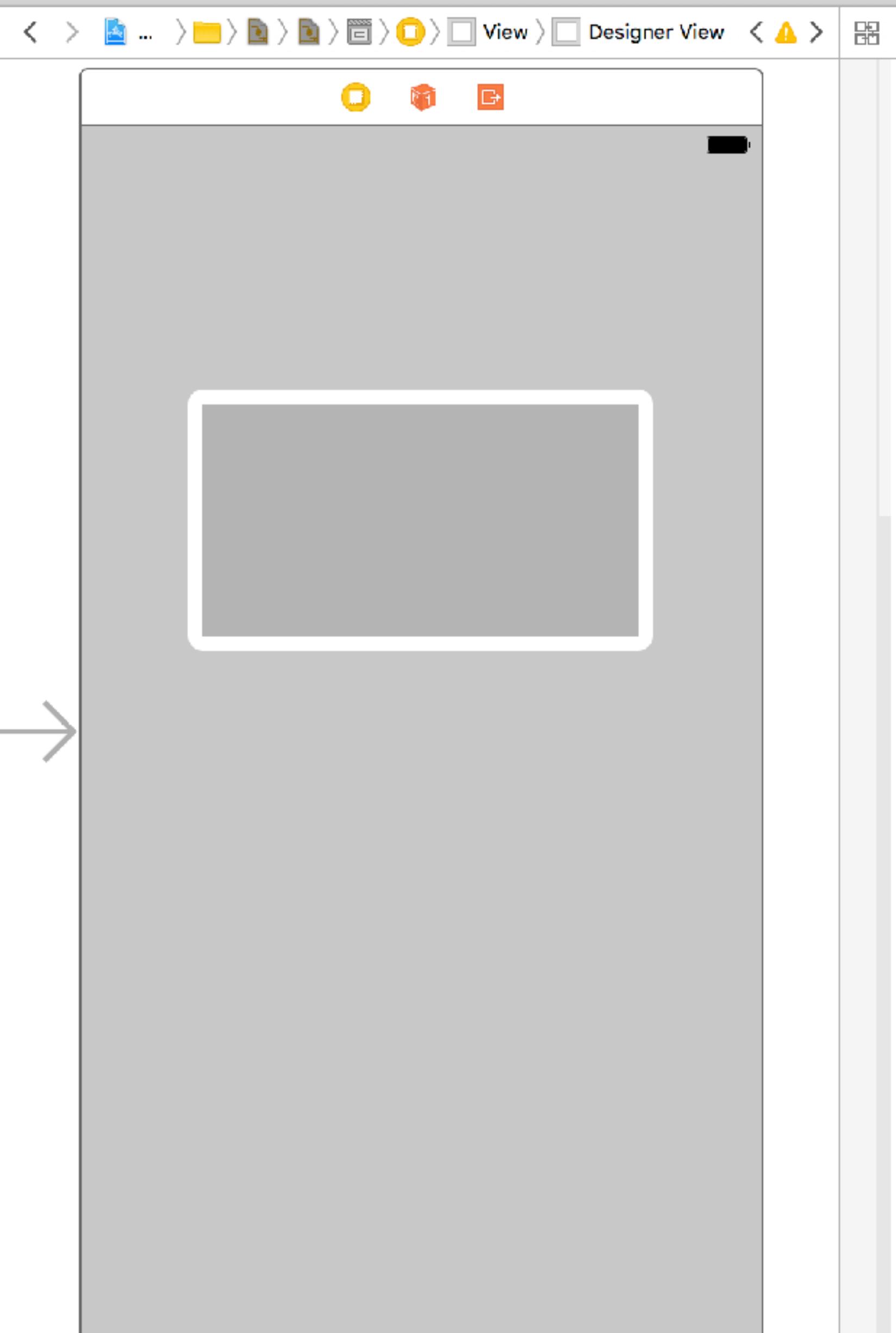
MPCS 51032 • SPRING 2020 • SESSION 5

CUSTOM DESIGNED
VIEWS

CUSTOM DESIGNED VIEWS

- Interface builder declarations that expose your custom attributes in Interface Builder
- `@IBInspectable` - adds interface to interface builder for custom views
- `@IBDesignable` - renders custom views in interface builder at design time





CUSTOM DESIGNED VIEWS

- Allows custom classes to be rendered in real time in Interface Builders without running

CUSTOM DESIGNED VIEWS

- Interface builder declarations that expose your custom attributes in Interface Builder
- `@IBInspectable` - adds interface to interface builder for custom views
- `@IBDesignable` - renders custom views in interface builder at design time

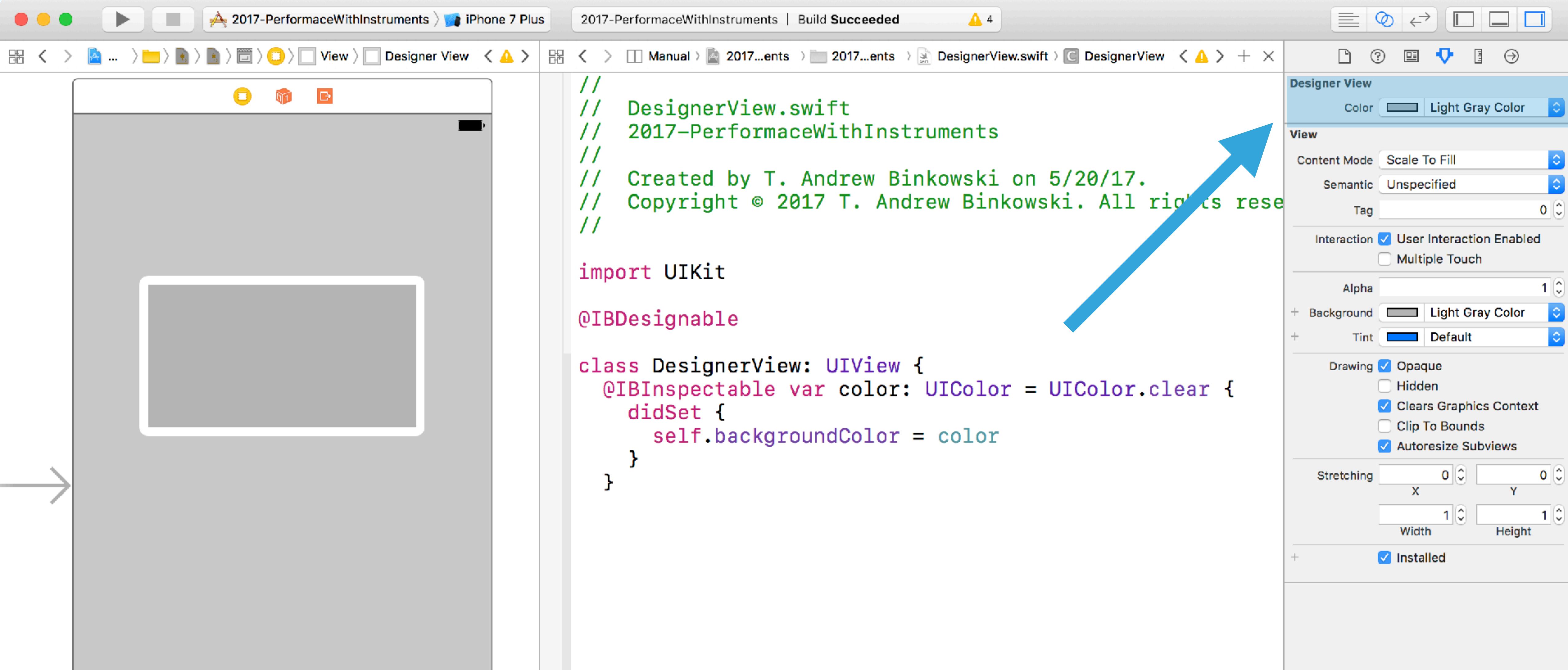
- `Int`
- `CGFloat`
- `Double`
- `String`
- `Bool`
- `CGPoint`
- `CGSize`
- `CGRect`
- `UIColor`
- `UIImage`

CUSTOM DESIGNED VIEWS

The screenshot shows the Xcode interface with the following details:

- Top Bar:** Shows the project name "2017-PerformaceWithInstruments", target "iPhone 7 Plus", and status "Finished running - Profiling 2017-PerformaceWithInstrument.. 2".
- Left Navigator:** Displays file paths: 2...ts > 2... > 2... > 2... > View > Designer View.
- Editor Area:** Shows the code for `DesignerView.swift`. The code includes comments about the file creation date (May 20, 2017) and copyright information, imports `UIKit`, and defines a class `DesignerView: UIView`.
- Right Sidebar (View Inspector):** Contains settings for the selected view:
 - Content Mode:** Scale To Fill
 - Semantic:** Unspecified
 - Tag:** 0
 - Interaction:** User Interaction Enabled (checked), Multiple Touch (unchecked)
 - Alpha:** 1
 - Background:** Light Gray Color
 - Tint:** Default
 - Drawing:** Opaque (checked), Hidden (unchecked), Clears Graphics Context (checked), Clip To Bounds (unchecked), Autoresize Subviews (checked)
 - Stretching:** X: 0, Y: 0
 - Width:** 1, **Height:** 1
 - Installed:** checked

CUSTOM DESIGNED VIEWS



The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files like `2017-PerformanceWithInstruments.xcodeproj`, `2017-PerformanceWithInstruments`, and `DesignerView.swift`.
- Editor:** Displays the `DesignerView.swift` file content:

```
// DesignerView.swift
// 2017-PerformanceWithInstruments
//
// Created by T. Andrew Binkowski on 5/20/17.
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.

import UIKit

@IBDesignable

class DesignerView: UIView {
    @IBInspectable var color: UIColor = UIColor.clear {
        didSet {
            self.backgroundColor = color
        }
    }
}
```
- Assistant Editor:** Shows the `Designer View` interface builder preview, which is a light gray square.
- Attributes Inspector:** On the right, it shows properties for the `Designer View` including:
 - Color:** Light Gray Color
 - View:**
 - Content Mode: Scale To Fill
 - Semantic: Unspecified
 - Tag: 0
 - Interaction:** User Interaction Enabled (checked), Multiple Touch (unchecked)
 - Alpha:** 1
 - Background:** Light Gray Color
 - Tint:** Default
 - Drawing:** Opaque (checked), Hidden (unchecked), Clears Graphics Context (checked), Clip To Bounds (unchecked), Autoresize Subviews (checked)
 - Stretching:** X: 0, Y: 0, Width: 1, Height: 1
 - Installed:** checked

A large blue arrow points from the text "Copyright © 2017 T. Andrew Binkowski. All rights reserved." towards the Attributes Inspector.

CUSTOM DESIGNED VIEWS

- Valid types to be inspectable
- They do not have to be visible

- Int
- CGFloat
- Double
- String
- Bool
- CGPoint
- CGSize
- CGRect
- UIColor
- UIImage

CUSTOM DESIGNED VIEWS

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files like 2017-PerformanceWithInstruments, 2017-PerformanceWithInstruments.xcodeproj, and DesignerView.swift.
- Editor:** Displays the code for `DesignerView.swift`. The code defines a `DesignerView` class that inherits from `UIView`. It includes properties for background color, border color, border width, and corner radius, each annotated with `@IBInspectable`.
- Utilities:** On the right side, the Utilities panel is open, showing settings for the `Designer View`. These include:
 - Color:** Light Gray Color
 - Border Color:** Default
 - Border Width:** 0
 - Corner Radius:** 0
 - View:** Content Mode: Scale To Fill, Semantic: Unspecified, Tag: 0, Interaction: User Interaction Enabled (checked), Alpha: 1, Background: Light Gray Color, Tint: Default.
 - Drawing:** Opaque (checked), Hidden, Clears Graphics Context (checked), Clip To Bounds, Autoresize Subviews (checked).
 - Stretching:** X: 0, Y: 0, Width: 1, Height: 1.
 - Installed:** Checked.

CUSTOM DESIGNED VIEWS

The screenshot shows the Xcode interface with a custom designed view. On the left is the code for the `DesignerView` class. In the center is a preview of the view, which is a light gray rectangle with a black border and rounded corners. On the right is the Attribute Inspector for the view.

```
//  
import UIKit  
  
@IBDesignable  
  
class DesignerView: UIView {  
    @IBInspectable var color: UIColor = UIColor.clear {  
        didSet {  
            self.backgroundColor = color  
        }  
    }  
  
    @IBInspectable var borderColor: UIColor = UIColor.white {  
        didSet {  
            self.layer.borderColor = borderColor.cgColor  
        }  
    }  
  
    @IBInspectable var borderWidth: CGFloat = 1.0 {  
        didSet {  
            self.layer.borderWidth = borderWidth  
        }  
    }  
  
    @IBInspectable  
    public var cornerRadius: CGFloat = 2.0 {  
        didSet {  
            self.layer.cornerRadius = cornerRadius  
        }  
    }  
}
```

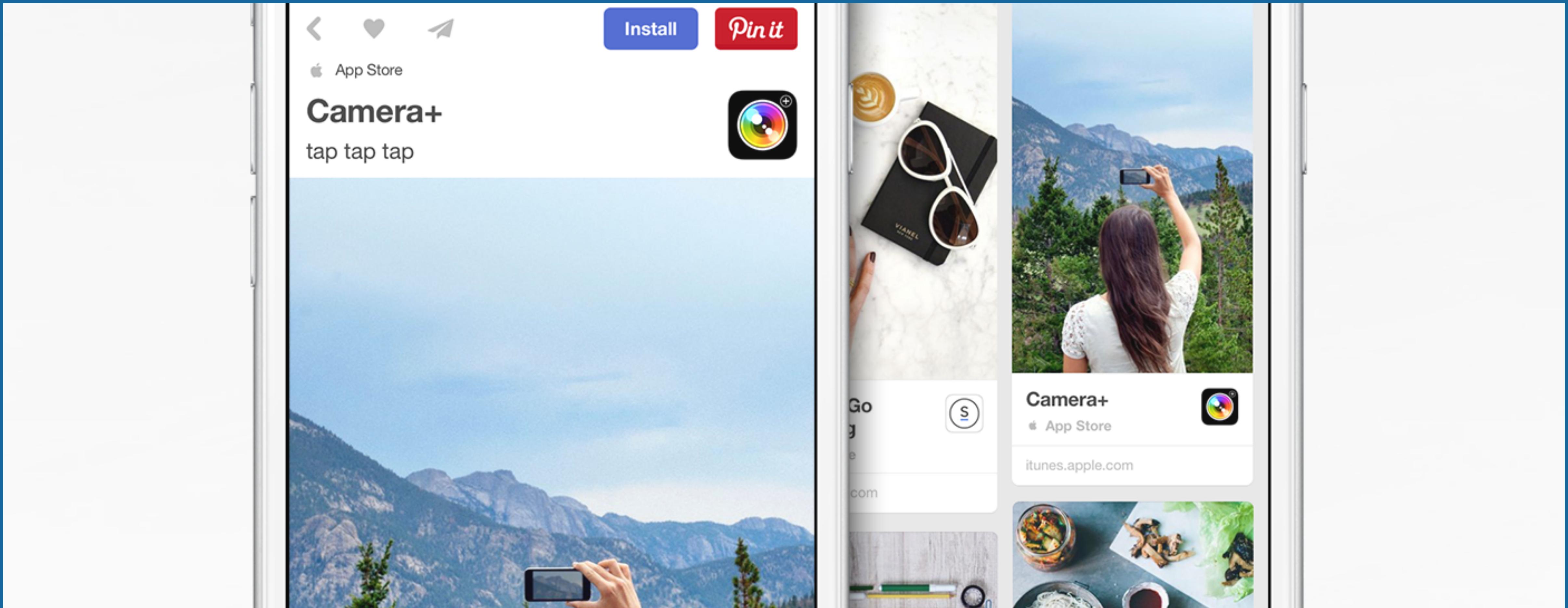
Corner Radius: 0

View

- Content Mode:** Scale To Fill
- Semantic:** Unspecified
- Tag:** 0
- Interaction:** User Interaction Enabled
 Multiple Touch
- Alpha:** 1
- Background:** Light Gray Color
- Tint:** Default
- Drawing:** Opaque
 Hidden
 Clears Graphics Context
 Clip To Bounds
 Autoresizes Subviews
- Stretching:** 0 X, 0 Y
- Width:** 1
Height: 1
- Installed:**

SCREENSHOT IMAGE
CACHE

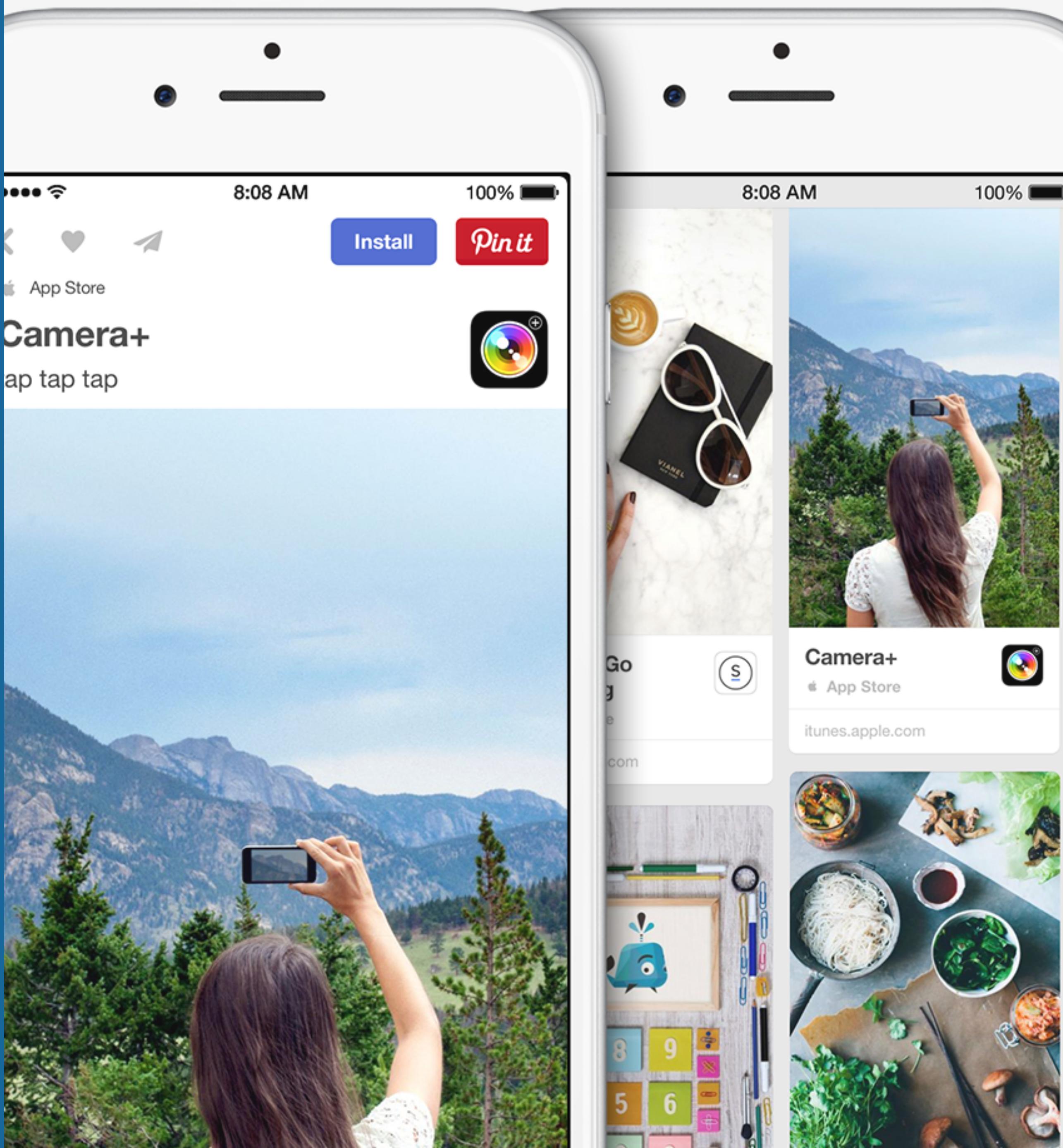
SCREENSHOT IMAGE CACHE



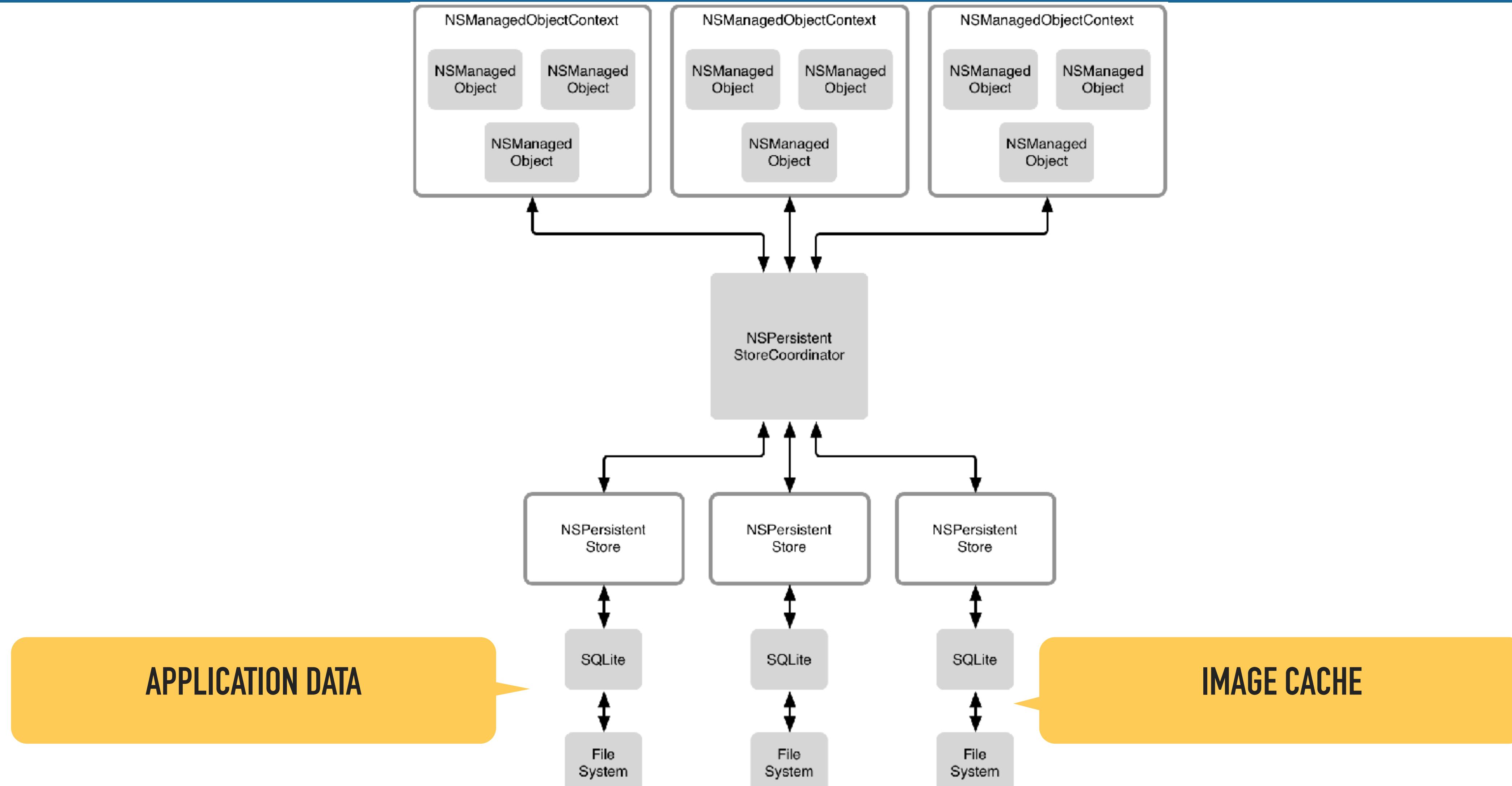
- Screen shot images can be big and they will be scrolled

SCREENSHOT IMAGE CACHE

- Options for caching
 - Memory
 - Documents directory
 - ApplicationSupport/Cache
- Options for storage
 - Write file
 - .sqlite
 - Core Data
- Options for Core Data
 - In-memory only
 - Persist with timestamp



SCREENSHOT IMAGE CACHE



SCREENSHOT IMAGE CACHE

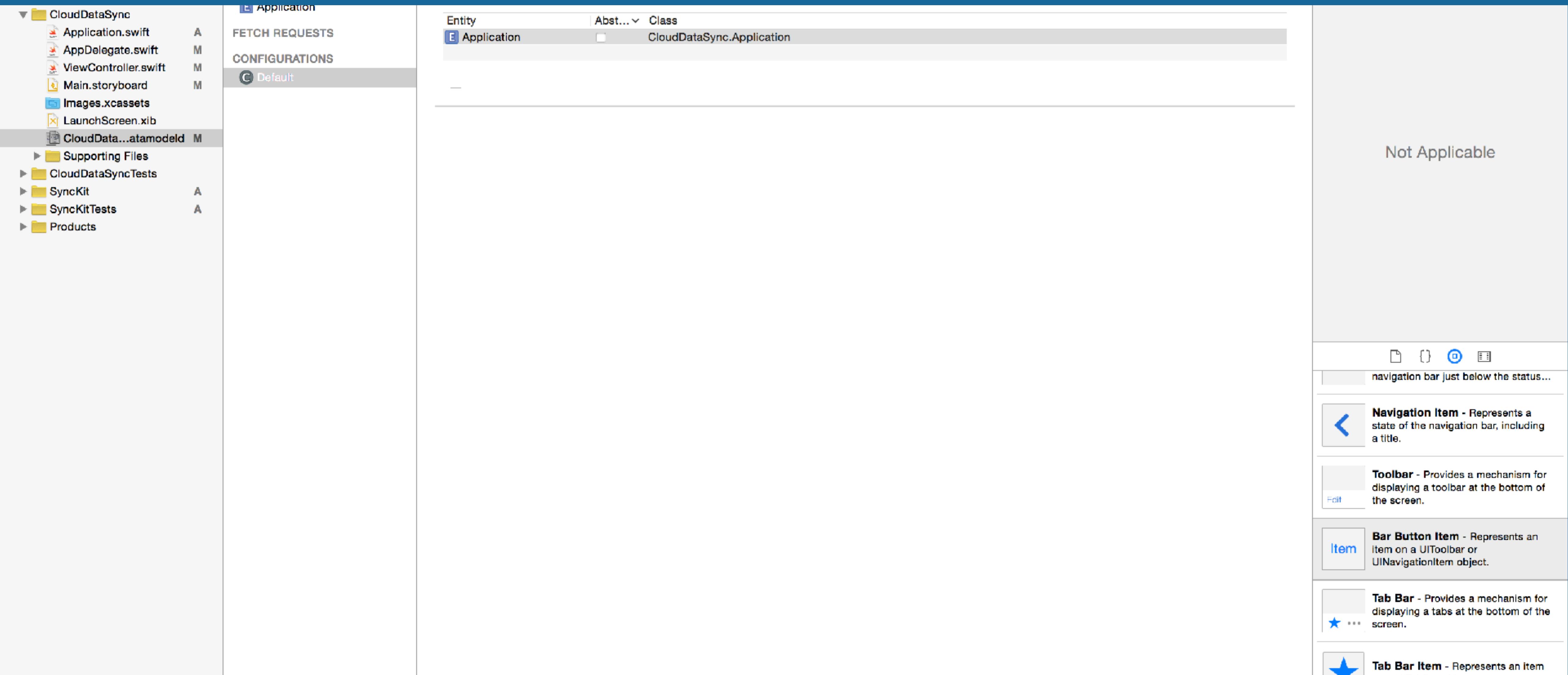
```
lazy var applicationDocumentsDirectory: NSURL = { ... }()
lazy var managedObjectModel: NSManagedObjectModel = { ... }()
lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
    // The persistent store coordinator for the application. This implementation creates and returns a coordinator, having added the store for the application to it. This property is optional since there are legitimate error conditions that could cause the creation of the store to fail.
    // Create the coordinator and store
    let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
    // Create store
    let url = self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqlite")
    var failureReason = "There was an error creating or loading the application's saved data."
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: url, options: nil)
    } catch {
        // Report any error we got.
        var dict = [String: AnyObject]()
        dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
        dict[NSLocalizedFailureReasonErrorKey] = failureReason
        dict[NSErrorUnderlyingErrorKey] = error as NSError
        let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
        NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
        abort()
    }
    // Create the image cache persistant store defined as a separate "Configuration"
    let imageCacheURL = self.applicationDocumentsDirectory.URLByAppendingPathComponent("ImageCache.sqlite")
    print("ImageCache URL: \(url)")
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: "ImageCache", URL: imageCacheURL, options: nil)
    } catch {
        print("Image Cache store problem")
        abort()
    }
}

return coordinator
```

APPLICATION DATA

IMAGE CACHE

SCREENSHOT IMAGE CACHE



SCREENSHOT IMAGE CACHE

The screenshot shows the Xcode interface for managing Core Data entities. On the left, the Project Navigator displays the project structure, including files like Application.swift, AppDelegate.swift, ViewController.swift, Main.storyboard, Images.xcassets, LaunchScreen.xib, CloudDataSyncTests, SyncKit, SyncKitTests, and Products.

The main area is the Core Data editor, which is currently viewing the entity **Image**. The entity has two attributes defined:

- imageData**: Type **Transformable**
- timeStamp**: Type **Date**

Below the attributes, there are sections for **Relationships** and **Fetched Properties**, each with '+' and '-' buttons for adding or removing items.

On the right side of the editor, there are several panels providing detailed information and configuration options for the entity:

- Name**: `imageData`
- Properties**:
 - Transient**
 - Optional**
 - Indexed**
- Attribute Type**: **Transformable**
- Name**: `Value Transformer Name`
- Advanced**:
 - Index in Spotlight**
 - Store in External Record File**
- User Info**: A table with columns **Key** and **Value**.
- Versioning**: Options for **Hash Modifier** (Version Hash Modifier) and **Renaming ID** (Renaming Identifier).
- Navigation Item**: Describes a state of the navigation bar, including a title.
- Toolbar**: Provides a mechanism for displaying a toolbar at the bottom of the screen.
- Item**: Represents an item on a UIToolbar or UINavigationItem object.
- Tab Bar**: Provides a mechanism for displaying tabs at the bottom of the screen.
- Tab Bar Item**: Represents an item.

SCREENSHOT IMAGE CACHE

```
e("ImageCache", inManagedObject  
go") !)
```

- Create image entity with metadata to determine when to purge it
- Core Data handles the management of reading/writing the file to/from memory/disk

SCREENSHOT IMAGE CACHE

Documents	►	Data	►	InternalDaemon	►	3E35DAB5-...B861A21134	►	Library	►	ImageCache.sqlite-shm
Downloads	►	Shared	►	PluginKitPlugin	►	3E440B27-6...8288226B4	►	tmp	►	ImageCache.sqlite-wal
Library	►			System	►	4AADAD79-...2268642839	►		►	SingleViewCoreData.sqlite
Media	►			TempDir	►	5B69A137-...4591A703B1	►		►	SingleViewC...ta.sqlite-shm
Root	►			VPNPlugin	►	5E4EFCFE-9...29E6ED00D	►		►	SingleViewC...ta.sqlite-wal
tmp	►			XPCService	►	6B09B616-F...38E2A1B42	►			
var	►				►	6BF3BB80-...750A6B639E	►			
					►	9AF04614-...66378F3EE5	►			
					►	34A96D6E-...07AF87754B	►			
					►	58F54BFF-0...7819945C4	►			
					►	92FAA192-1...F69515F25	►			
					►	0490E65B-...8BECF54CF8	►			
					►	735EA0A2-...728FD20885	►			
					►	867CDC43-3AOC-4C6B-BDA1-7AEEAFBAFF12				
					►	1489ED51-F...EFB81E7BE				
					►	0959142E-3...F2581B352				
					►	8297456C-...8B6E11B8F0				
					►	56601809-...439F7C70A2				
					►	A8A0D683-...AE28A4D326				
					►	AD2EE5D6-...11B7E9F158				
					►	AFF4A29C-...801D3C1388				
					►	B35C58CA-...E1D8822709				
					►	C06FB3A2-...242A5941E0				
					►	C16AF0E9-...A988B11BBE				
					►	C62EF3B0-...283057C1F2				
					►	D1F67944-...7E937ED6BC				
					►	E4E26518-A...C0F02A0F0				
					►	E8D42EDD-...AA203FAA03				
					►	EDA94372-...3FCEE0168F				

SCREENSHOT IMAGE CACHE

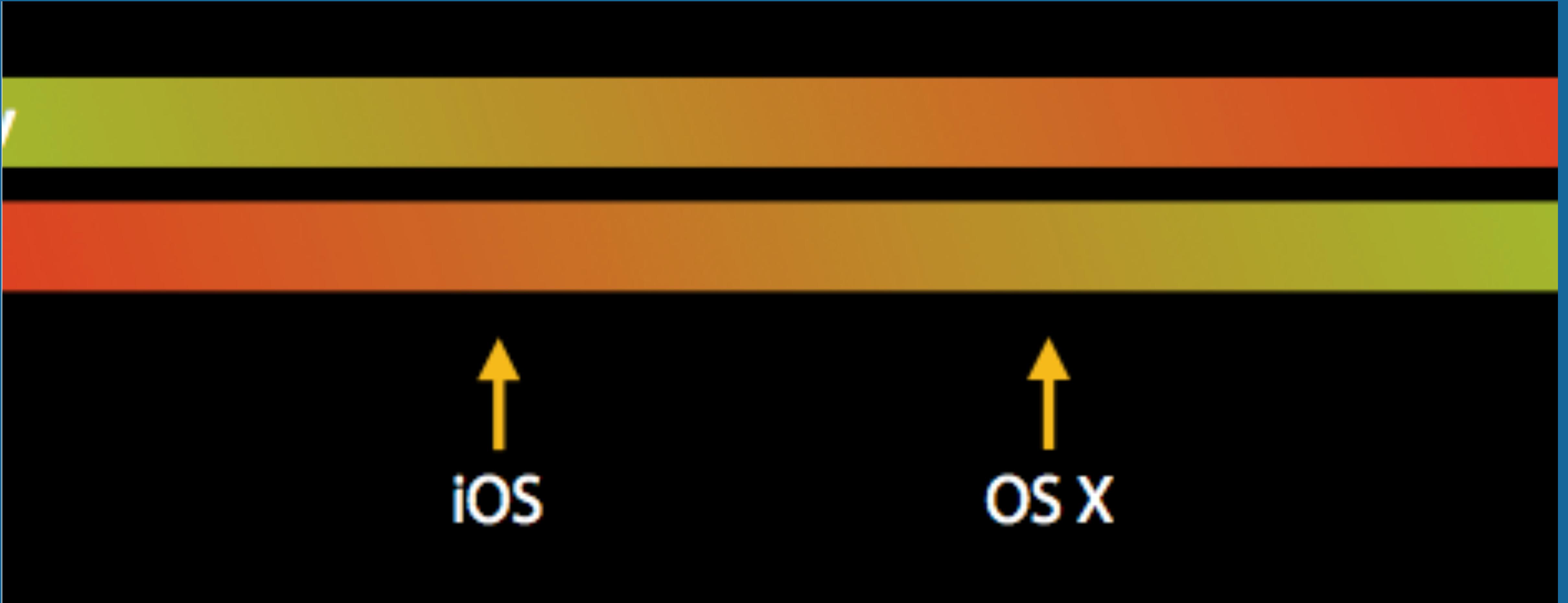
- Transformable vs Binary Data
 - Big SQLite File
 - Transformable is binary data with some metadata
 - <http://stackoverflow.com/questions/22387924/changing-existing-core-data-image-store-from-transformable-to-binary-data-allows>

SCREENSHOT IMAGE CACHE

- Additional configurations could be used for multiple store
 - Application (preloaded)
 - Favorites (user defined)
 - ImageCache

CORE DATA PERFORMANCE

CORE DATA PERFORMANCE



- Core data optimization is a balance
 - Minimize memory usage

CORE DATA PERFORMANCE

- Performance hits
 - Loading too much
 - Firing many faults
 - Frequent cache misses
 - Expensive queries
 - Incurring too many locks during read/write

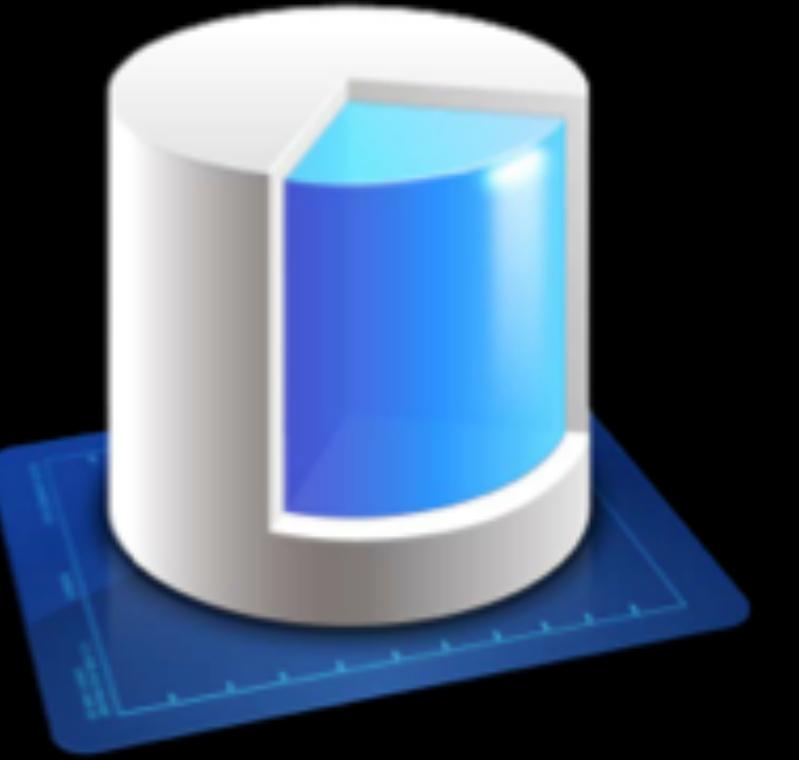


Core Data

MEASURING PERFORMANCE

MEASURING PERFORMANCE

- Instruments
 - What are you looking for?
 - How long should it take?
- Interpreting the results
 - Cache misses?
 - Fetches?
 - Faults firing?
 - Saves?
 - Memory usage?



Core Data



Time Profiler



Allocations



File Activity

MEASURING PERFORMANCE

ENTITIES

- E Application
- FETCH REQUESTS
- CONFIGURATIONS
- C Default

▼ Attributes

Attribute ▲	Type
S applicationId	String
S name	String
N specialNumber	Integer 16

+ -

► Relationships

▼ Fetched Properties

Fetched Property ▲	Predicate
--------------------	-----------

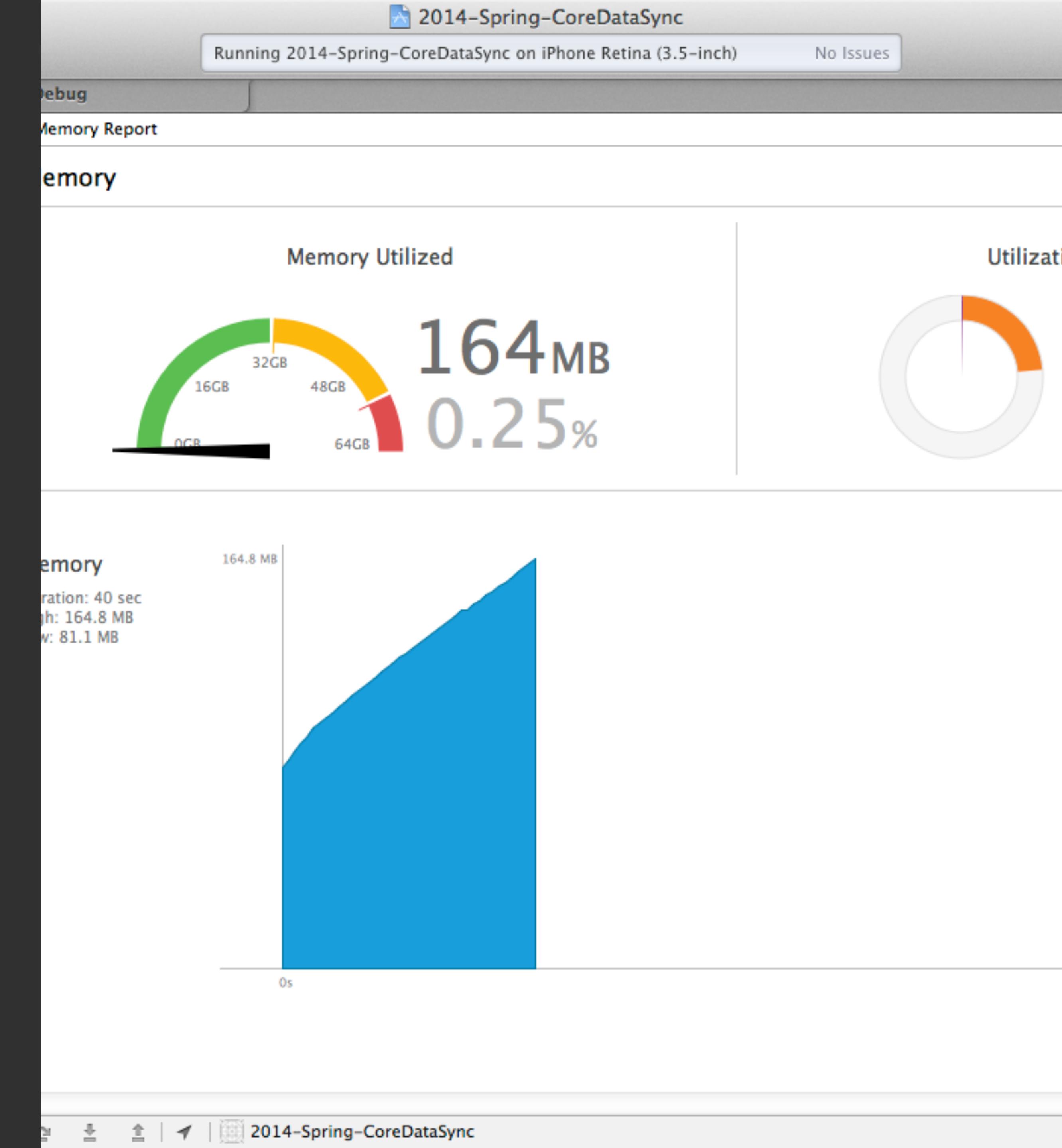
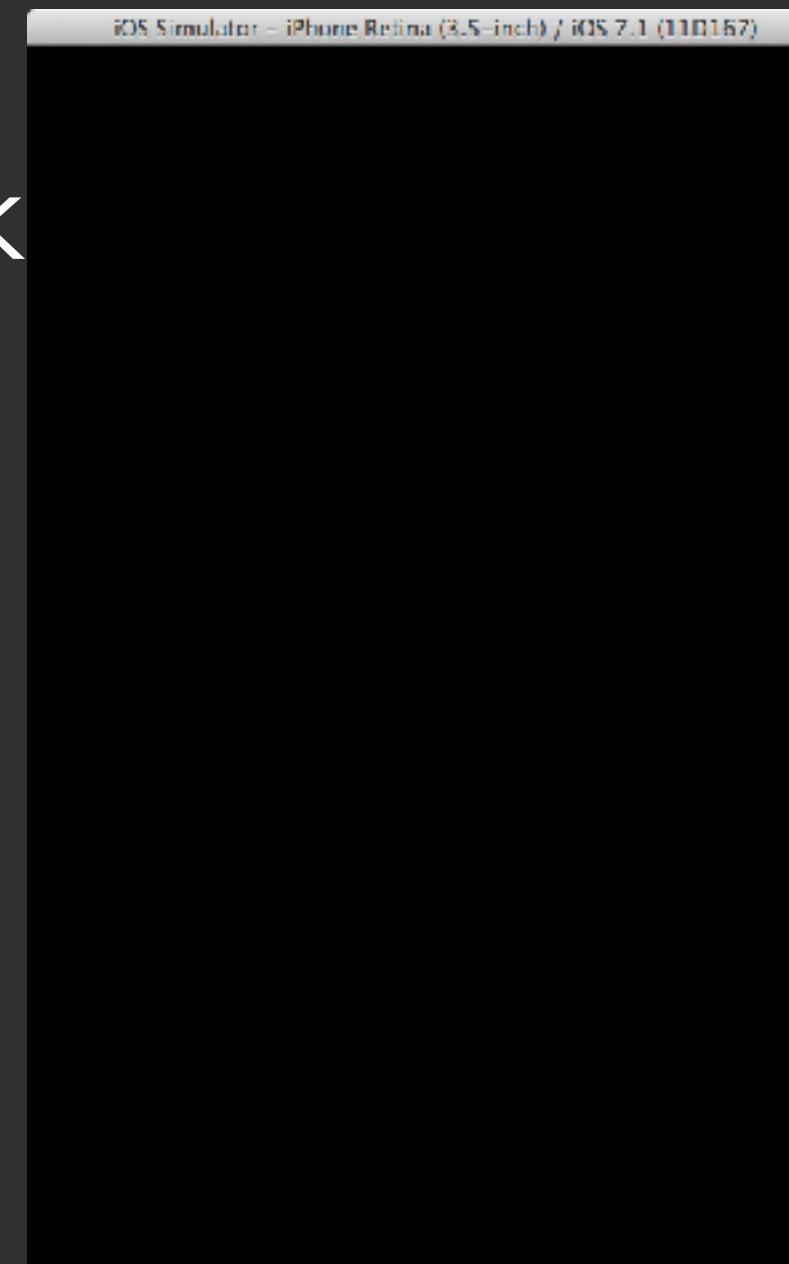
+ -

For demonstration purposes only

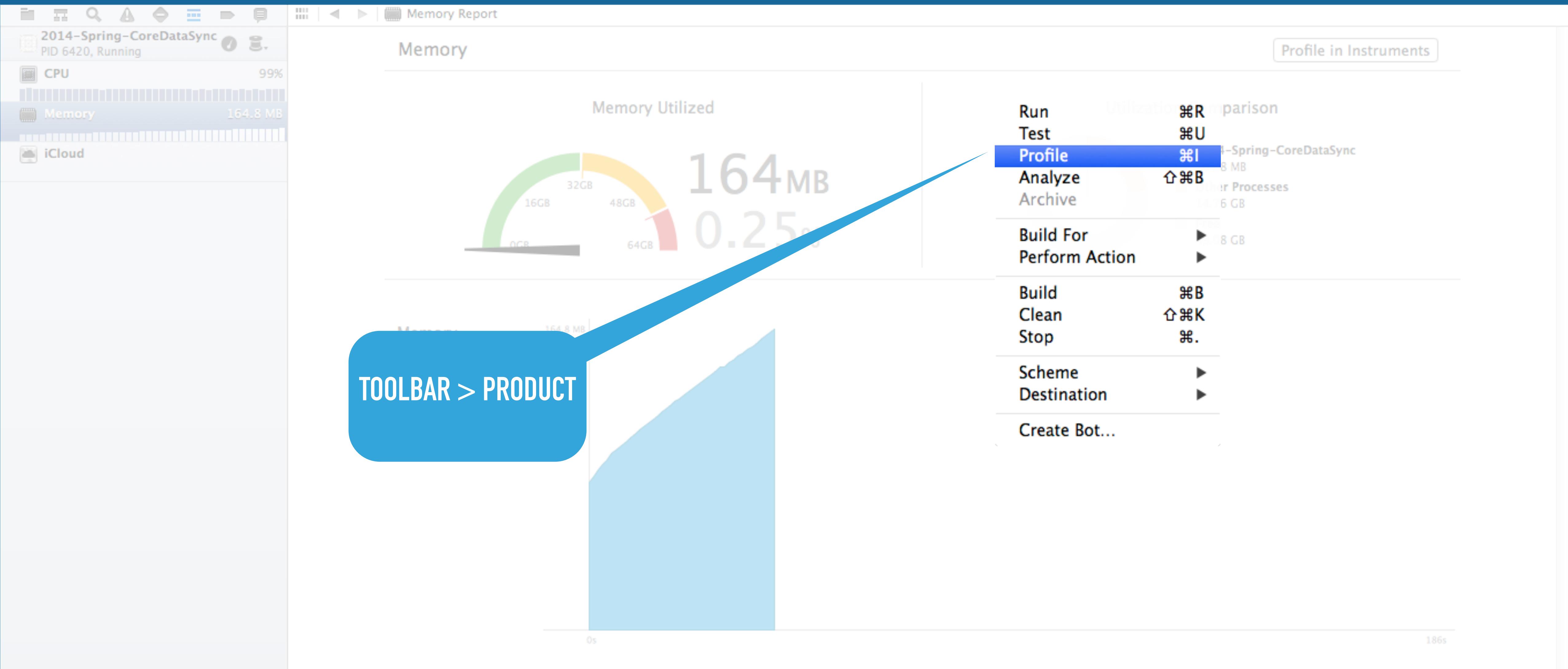
MEASURING PERFORMANCE

- Slow launch = black screen
 - iOS watchdog will kill launch if necessary

IOS WOULD PROBABLY
KILL THIS



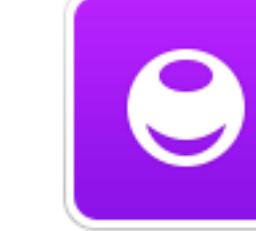
MEASURING PERFORMANCE



MEASURING PERFORMANCE

Choose a profiling template for:  iPhone 6s Plus (9.3) >  2016-CoreDataStack

Standard Custom Recent

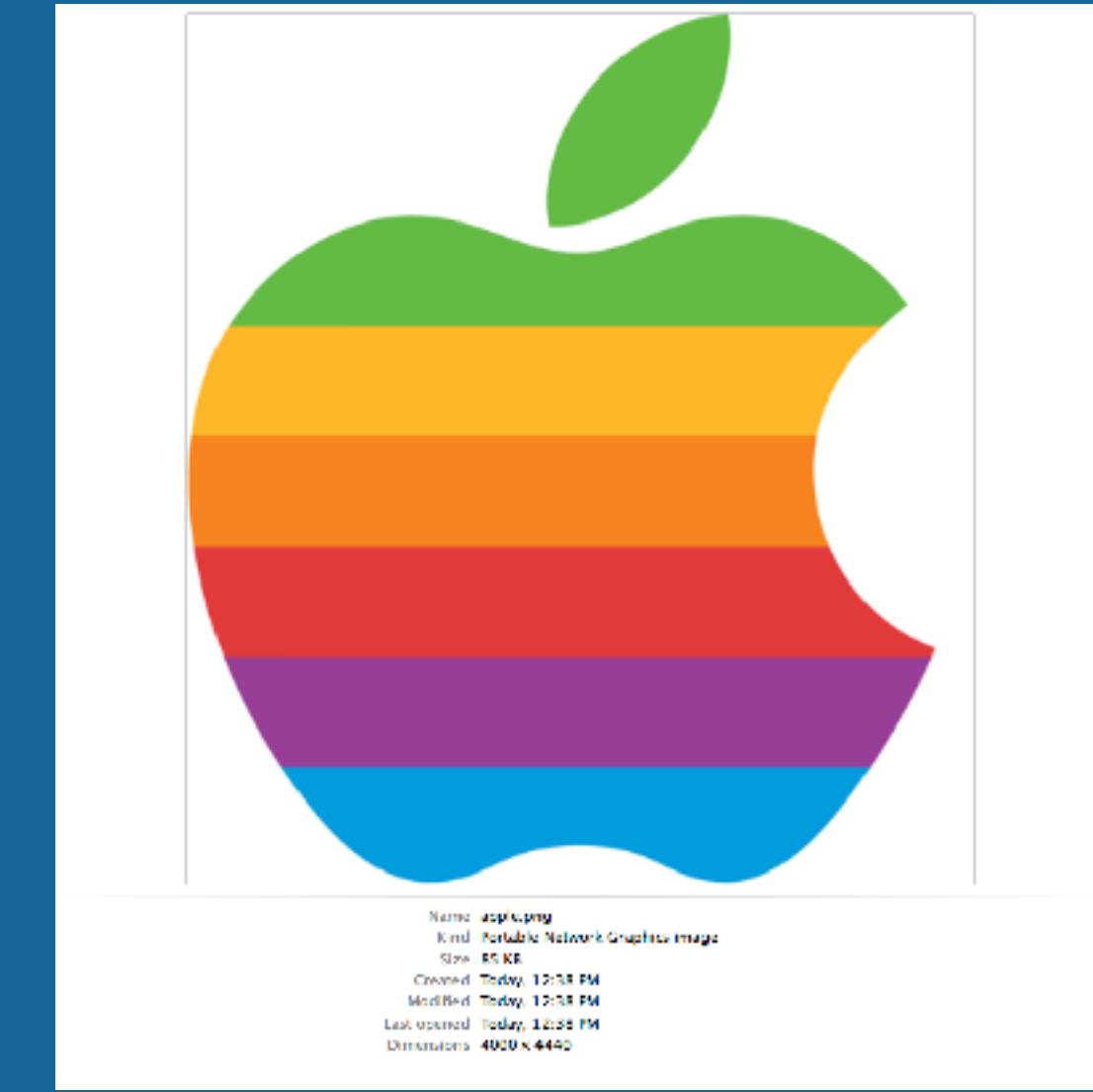
 Blank	 Activity Monitor	 Allocations	 Automation	 Cocoa Layout	 Core Animation
 Core Data	 Counters	 Energy Log	 File Activity	 GPU Driver	 Leaks
 Leaks	 (())	 ES	 I/O	 Timer	

 **Leaks**
Measures general memory usage, checks for leaked memory, and provides statistics on object allocations by class as well as memory address histories for all active allocations and leaked blocks.

OPTIMIZING SAVES

MEASURING PERFORMANCE

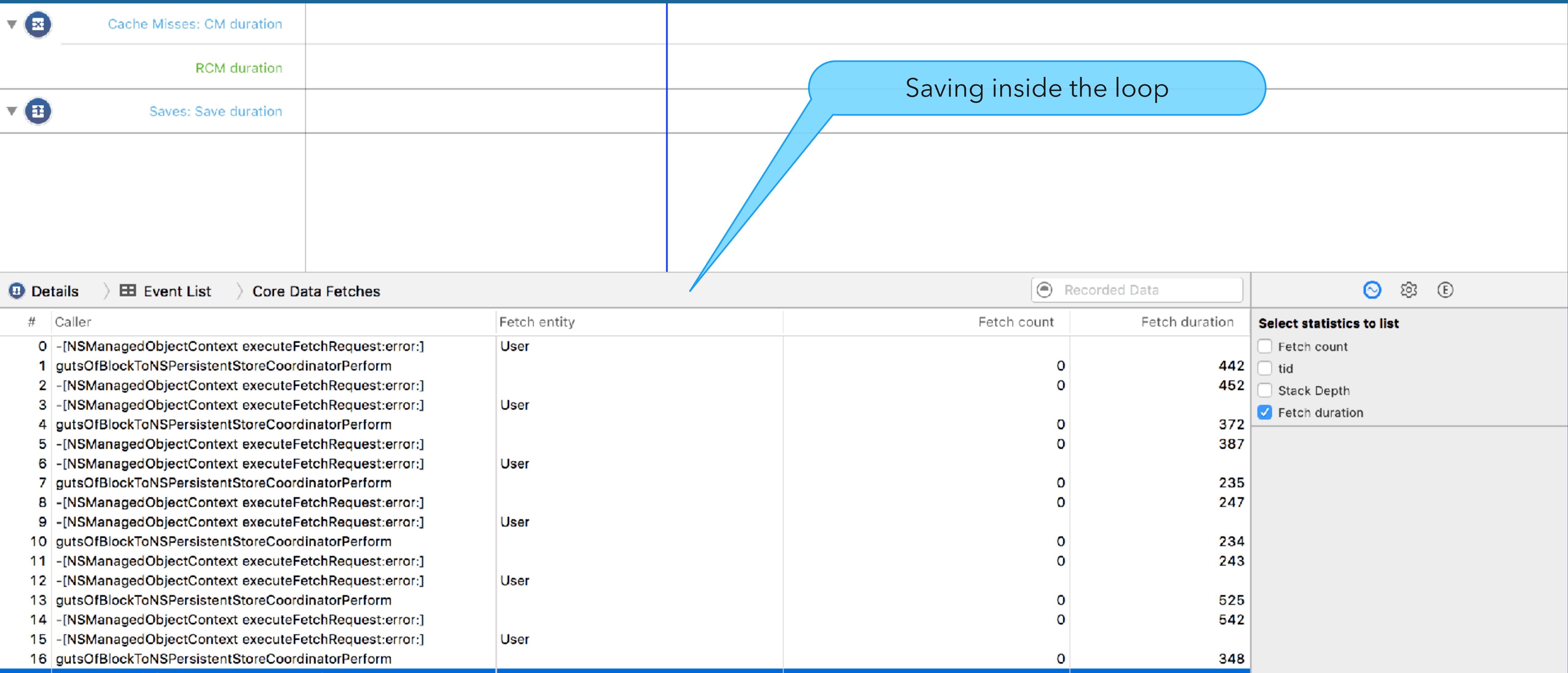
- Add and save some big data



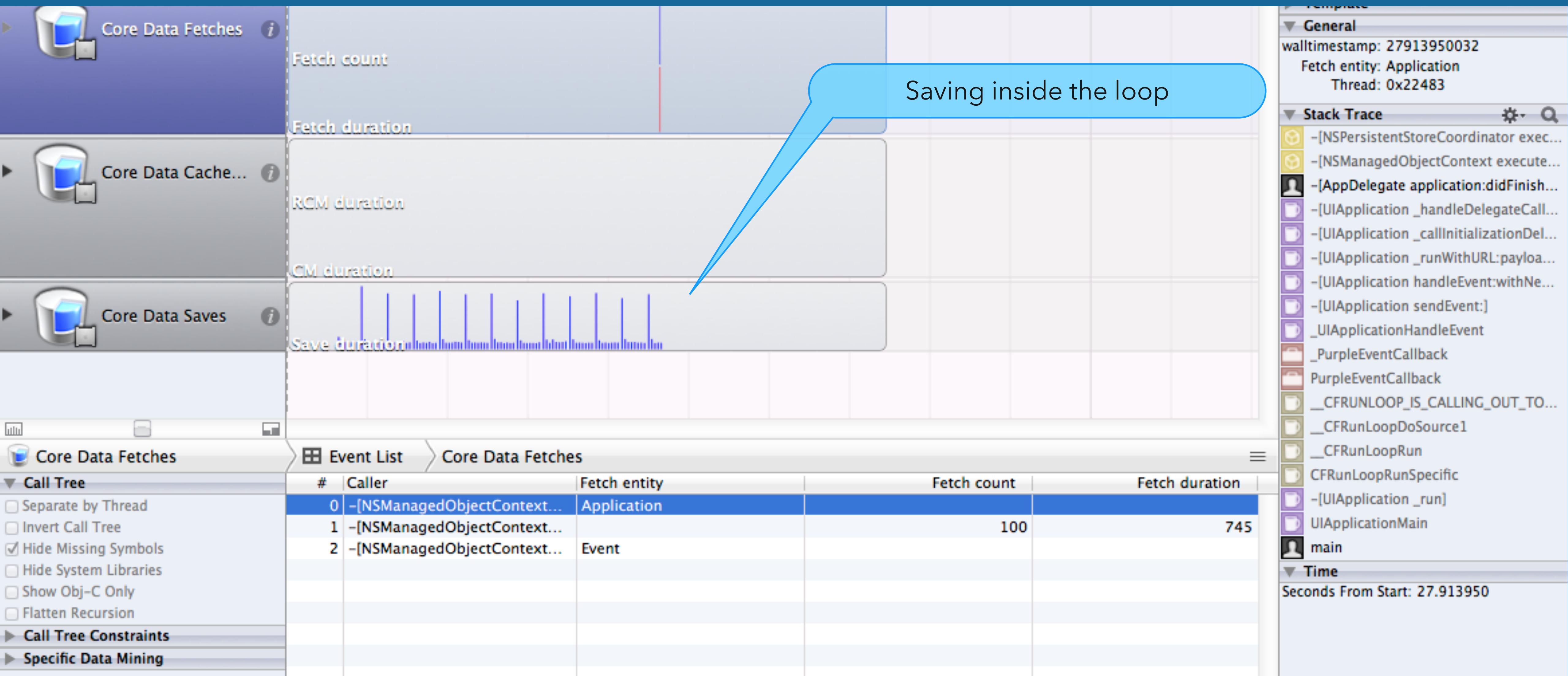
```
for (int specialNumber=0; specialNumber < 500; specialNumber++) {  
    NSManagedObject *newManagedObject = [NSEntityDescription insertNewObjectForEntityForName:@"Application"  
                                         inManagedObjectContext:[StoreManager  
                                                sharedStoreManager].managedObjectContext];  
    [newManagedObject setValue:[NSDate date] description forKey:@"applicationId"];  
    [newManagedObject setValue:[NSDate date] description forKey:@"name"];  
    [newManagedObject setValue:[NSNumber numberWithInt:specialNumber] forKey:@"specialNumber"];  
    UIImage *apple = [UIImage imageNamed:@"ClassicAppleLogo"];  
    NSData *imageData = UIImagePNGRepresentation(apple);  
    [newManagedObject setValue:imageData forKey:@"image"];  
  
    [[StoreManager sharedStoreManager] saveContext];  
}
```

Saving in the loop

MEASURING PERFORMANCE

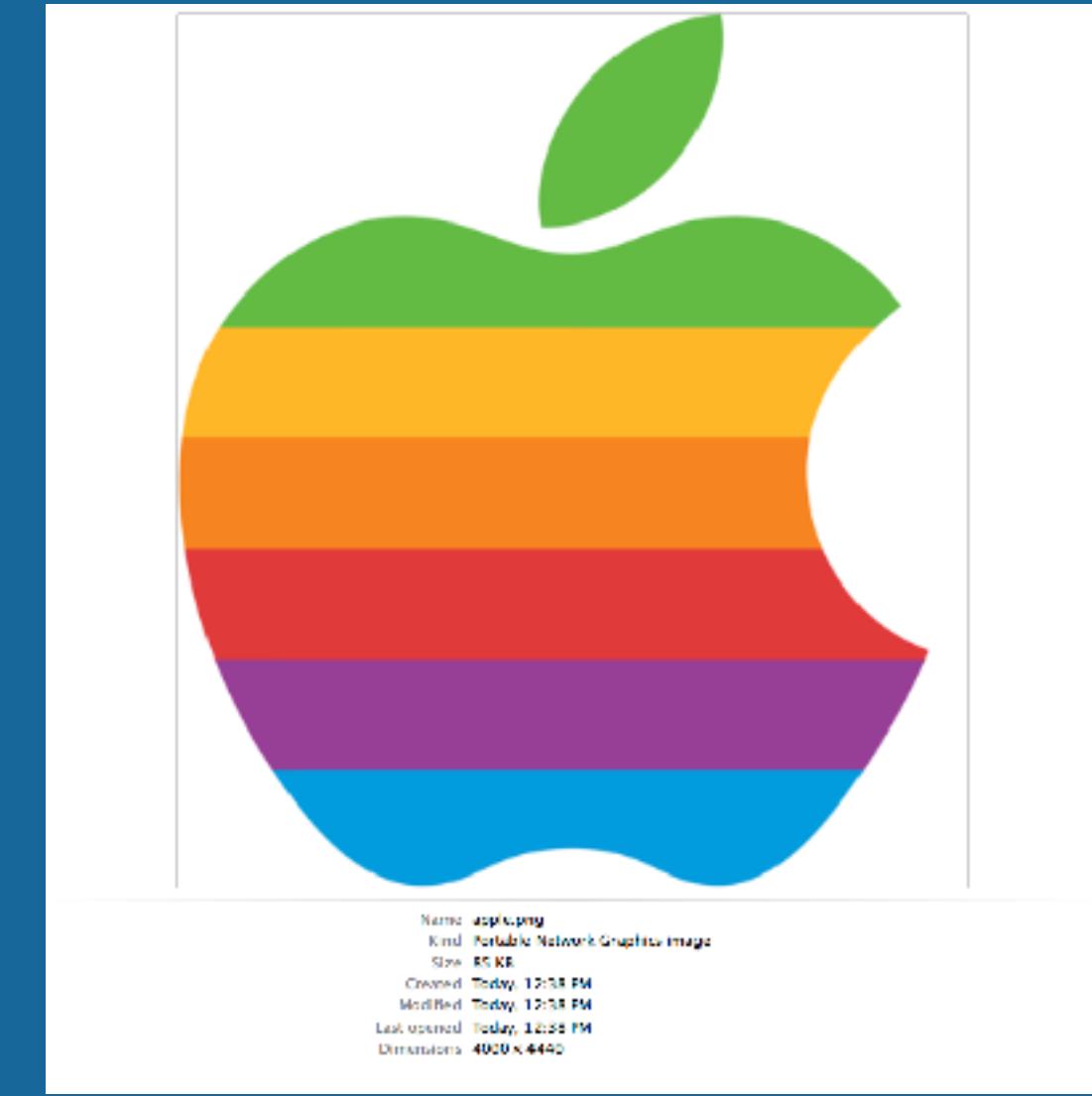


MEASURING PERFORMANCE



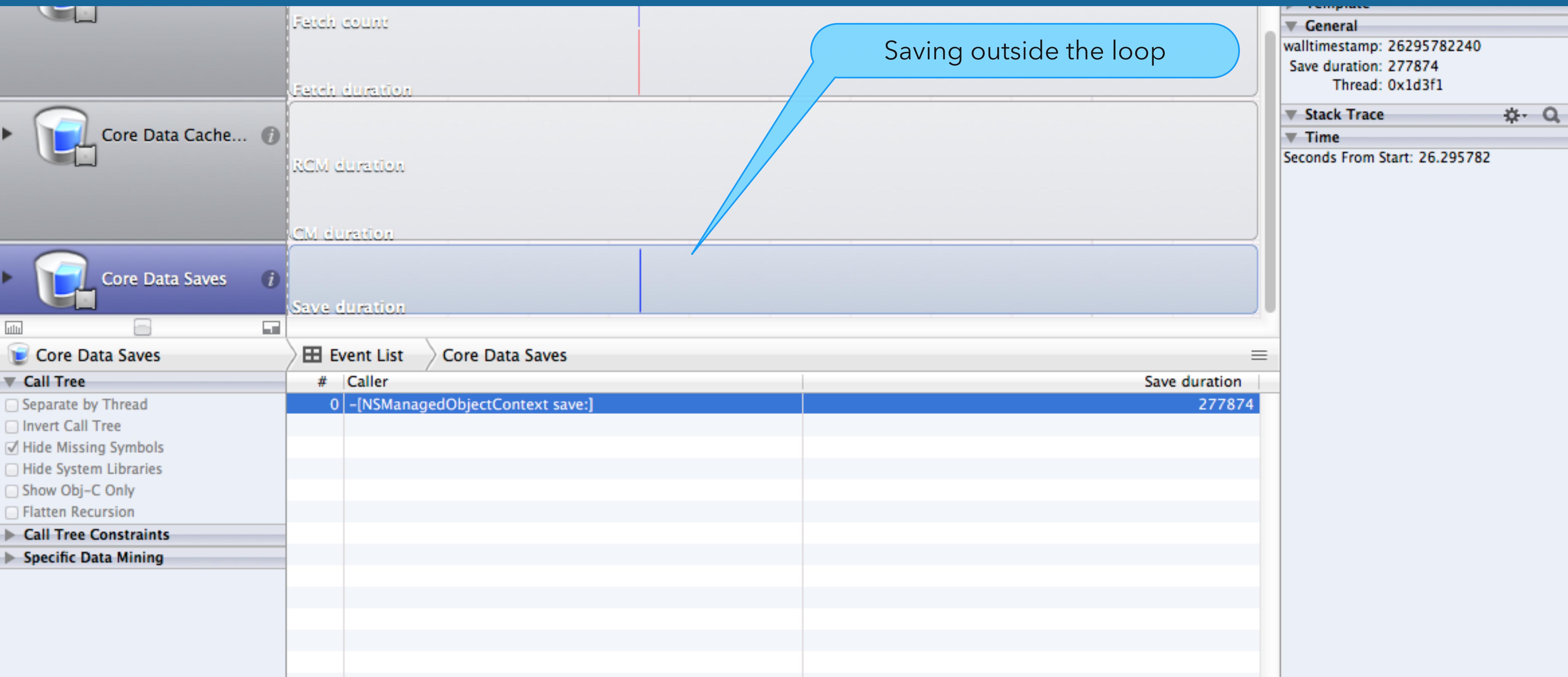
MEASURING PERFORMANCE

- Add and save some big data



Saving outside the loop

MEASURING PERFORMANCE



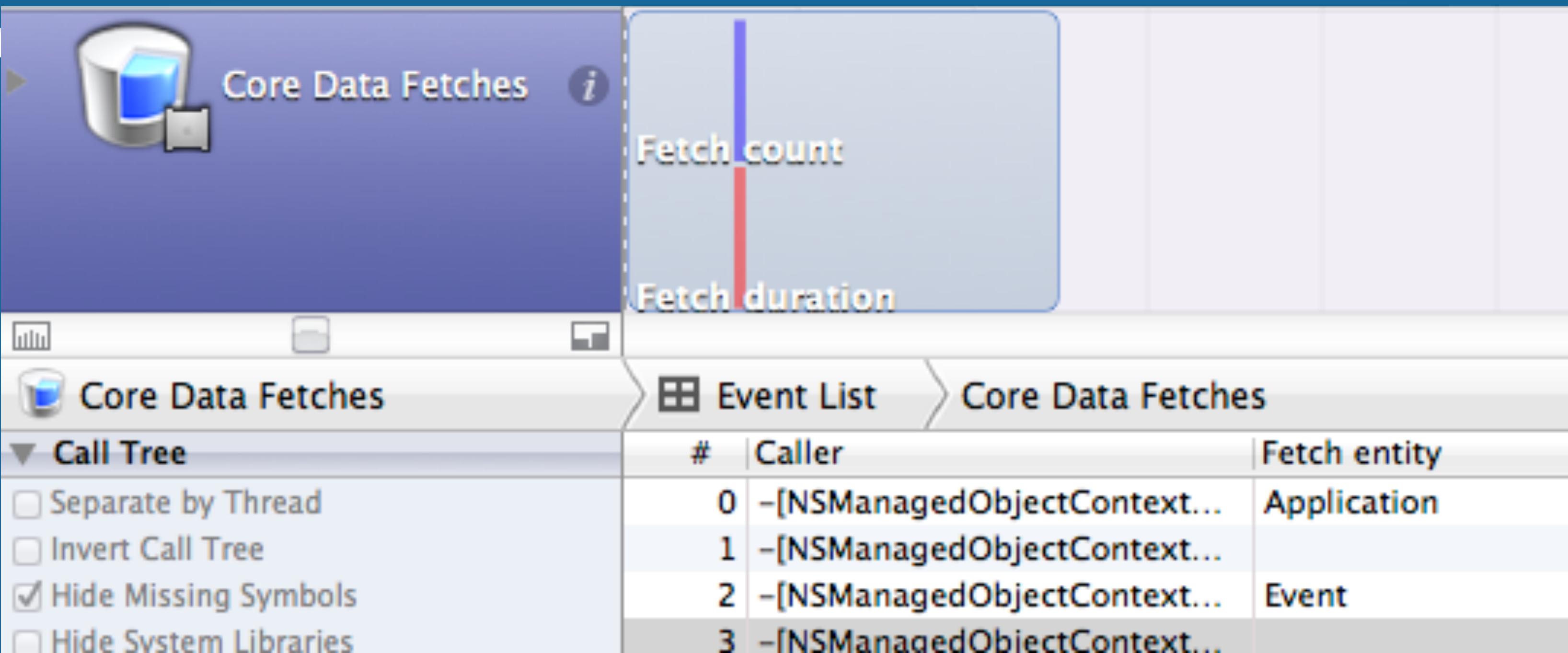
OPTIMIZING FETCH REQUESTS

MEASURING PERFORMANCE

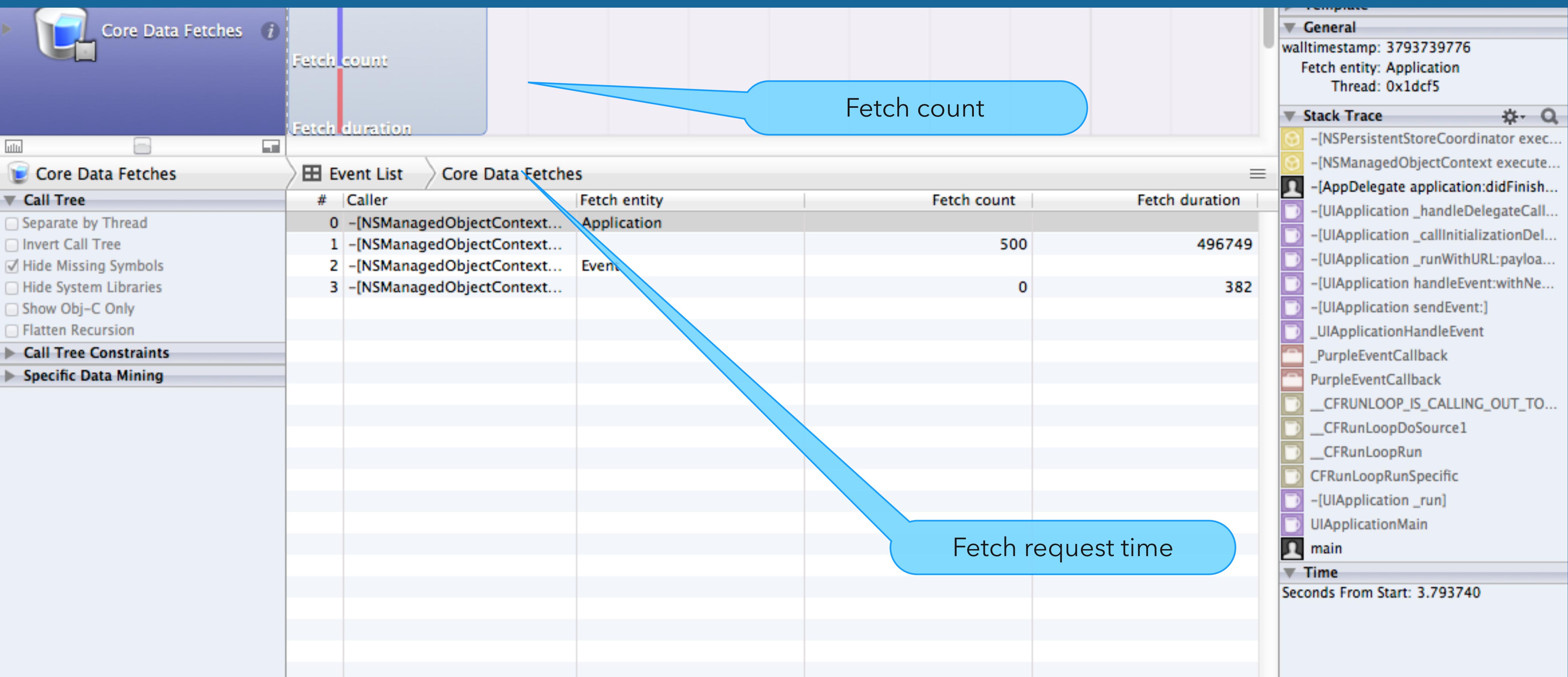
```
// Fetch the Application objects (just for demonstration)
NSFetchRequest *request = [[NSFetchRequest alloc] initWithEntityName:@"Application"];

NSArray *results = [[StoreManager sharedStoreManager].managedObjectContext
                     executeFetchRequest:request error:nil];
```

- Optimizing Core Data Fetches



MEASURING PERFORMANCE



MEASURING PERFORMANCE

- Optimizing Fetch Requests

- Don't fetch more than you need

- Only

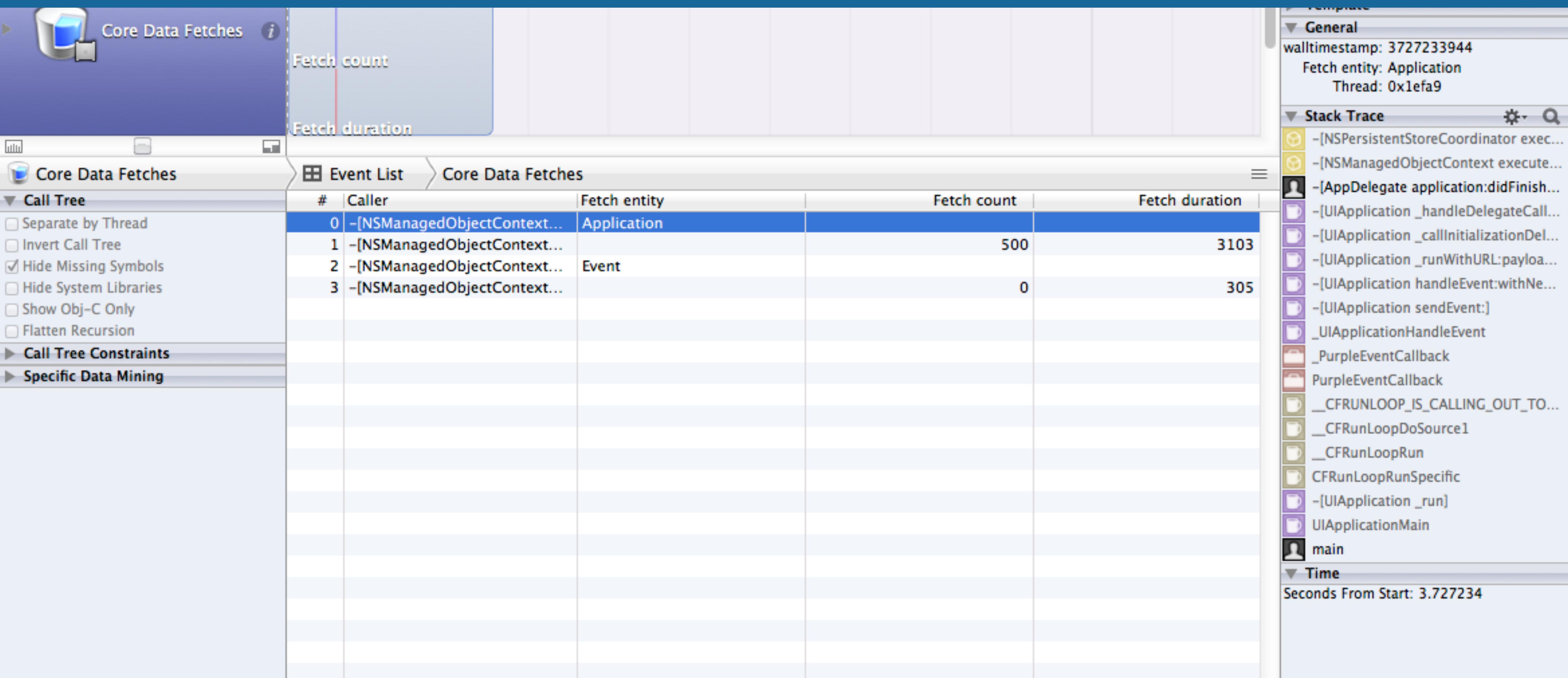
```
// Fetch the Application objects (just for demonstration)
NSFetchRequest *request = [[NSFetchRequest alloc] initWithEntityName:@"Application"];
request.fetchBatchSize = 20;

NSArray *results = [[StoreManager sharedStoreManager].managedObjectContext
                    executeFetchRequest:request error:nil];
```

- Don

- Use fetch batch size

MEASURING PERFORMANCE



MEASURING PERFORMANCE

- Optimizing Fetch Requests
 - Don't fetch more than you need
 - Only limited number on screen
 - Don't fetch everything
 - Use fetch batch size

```
// Fetch the Application objects (just for demonstration)
NSFetchRequest *request = [[NSFetchRequest alloc] initWithEntityName:@"Application"];
request.fetchBatchSize = 20;

NSArray *results = [[StoreManager sharedStoreManager].managedObjectContext
                     executeFetchRequest:request error:nil];
```

- Returns a special array
 - CoreData will automatically fetch additional objects as needed

MEASURING PERFORMANCE

The screenshot shows the Xcode Instruments application running the "Core Data Fetches" template. The main interface displays performance metrics for Core Data fetch operations, including "Fetch count" and "Fetch duration". A detailed "Event List" table provides a breakdown of these fetches by caller, entity, count, and duration. A blue callout bubble highlights the "Core Data fetching" section of the table. To the right, a "Call Tree" sidebar shows various filtering options, and the "General" and "Stack Trace" panes provide additional context about the application's execution environment.

#	Caller	Fetch entity	Fetch count	Fetch duration
0	-[NSManagedObjectContext ...]	Application		
1	-[NSManagedObjectContext ...]	Event	500	3035
2	-[NSManagedObjectContext ...]		0	339
3	-[NSManagedObjectContext ...]	Application	500	2565
193	-[NSManagedObjectContext ...]	Application	20	68173
192	-[NSManagedObjectContext ...]	Application	20	55481
191	-[NSManagedObjectContext ...]	Application	20	55625
190	-[NSManagedObjectContext ...]	Application	20	55025
189	-[NSManagedObjectContext ...]	Application	20	67521
188	-[NSManagedObjectContext ...]			
187	-[NSManagedObjectContext ...]			
186	-[NSManagedObjectContext ...]			
185	-[NSManagedObjectContext ...]			
184	-[NSManagedObjectContext ...]			

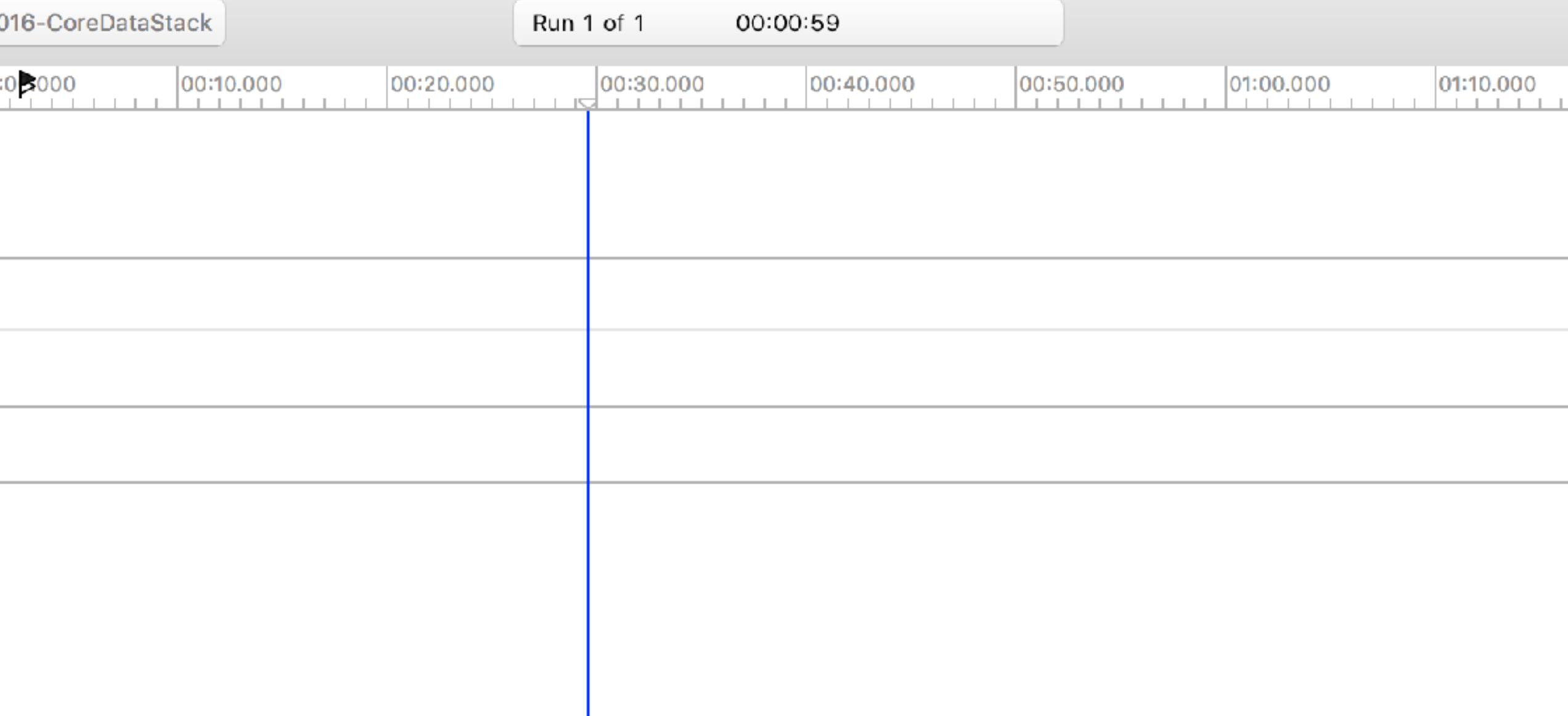
Core Data fetching

General
walltimestamp: 3363740112
Fetch entity: Application
Thread: 0x20e3d

Stack Trace

Time
Seconds From Start: 3.363740

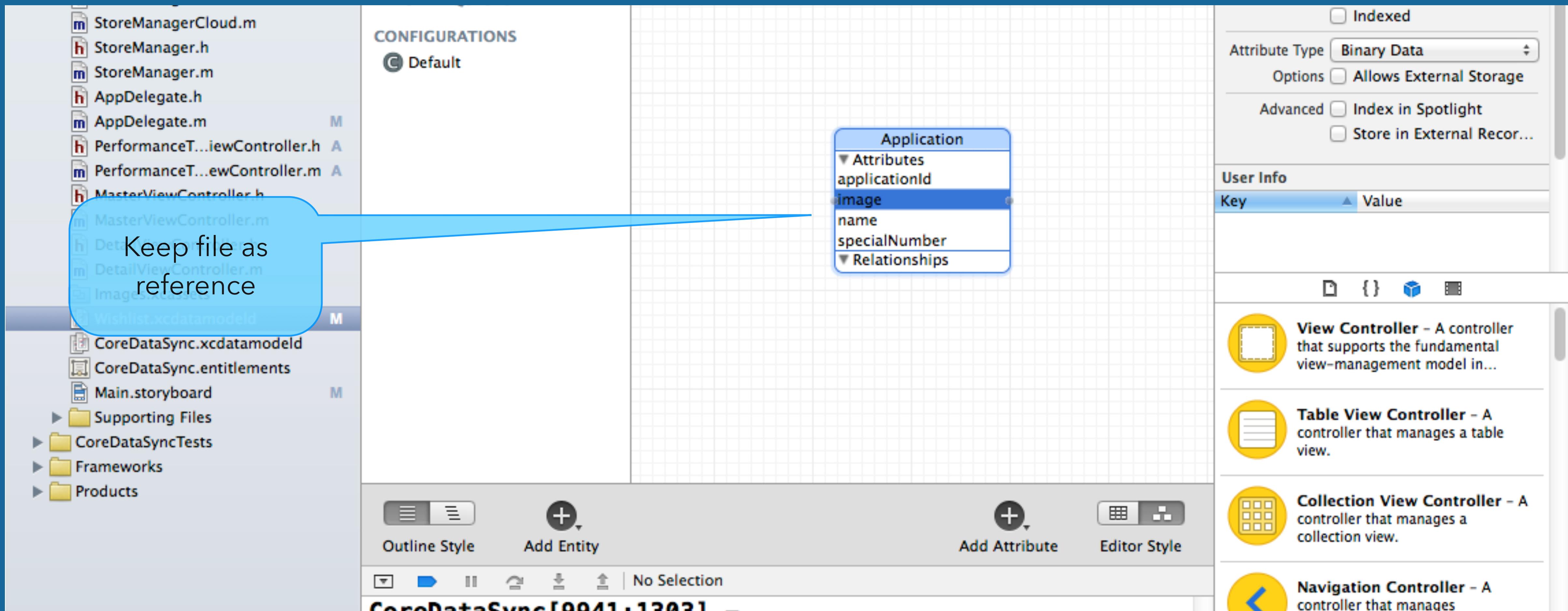
OPTIMIZING THE DATA MODEL



OPTIMIZING THE DATA MODEL

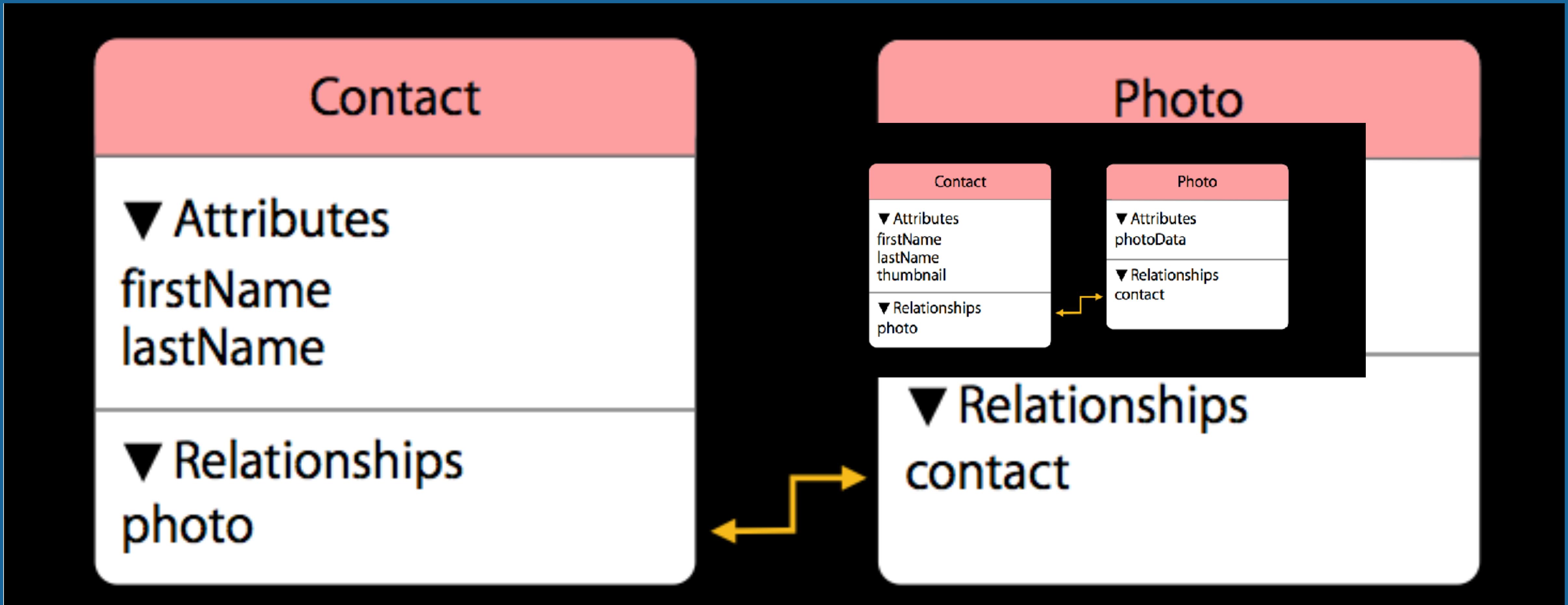
- Typically you would normalize data
 - Don't duplicate data in your application
 - Easier to maintain
 - Less storage
- In CoreData, this may not be the best performing option
 - Balance fetches vs. time vs. memory

OPTIMIZING THE DATA MODEL



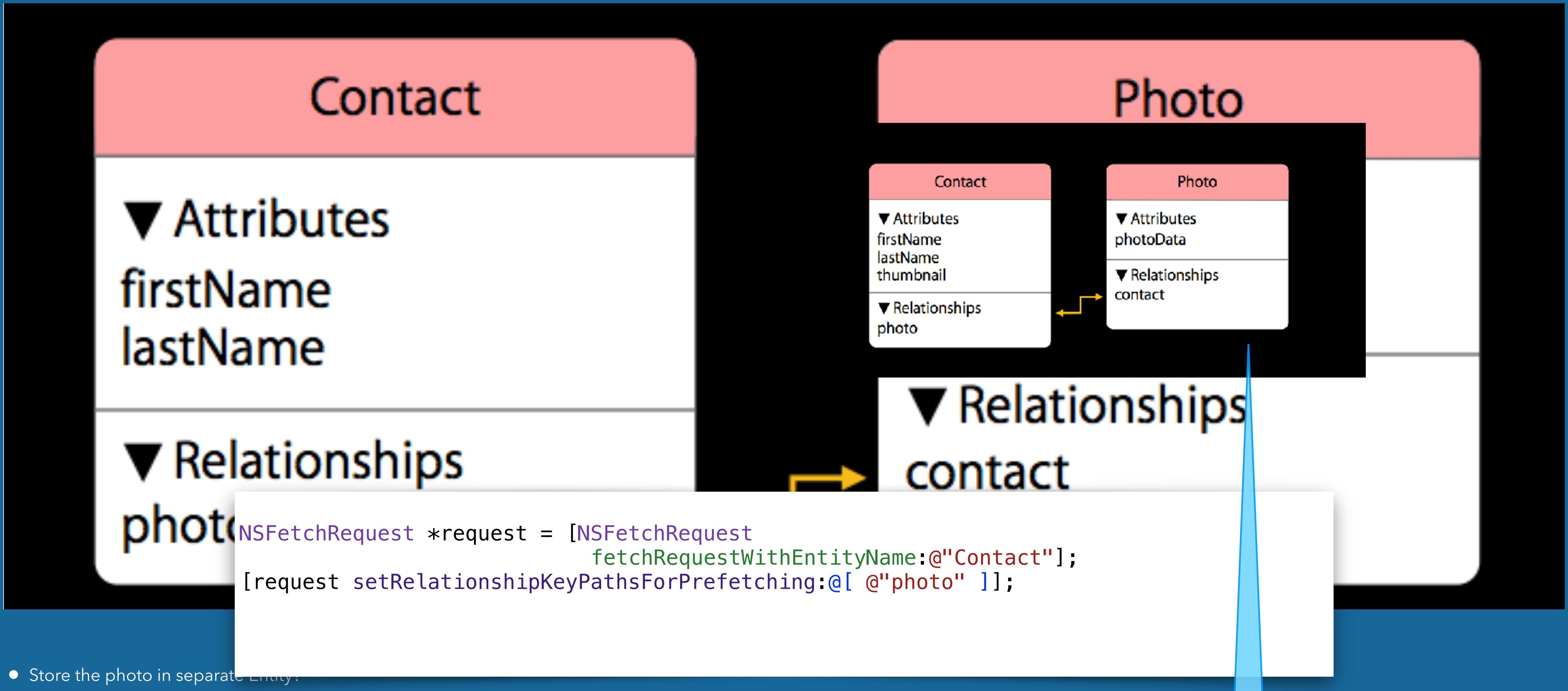
- Keep photo as a separate Entity or as an attribute?

OPTIMIZING THE DATA MODEL



- Store the photo in separate Entity?
- Pros:
 - Photo is not loaded in fetch

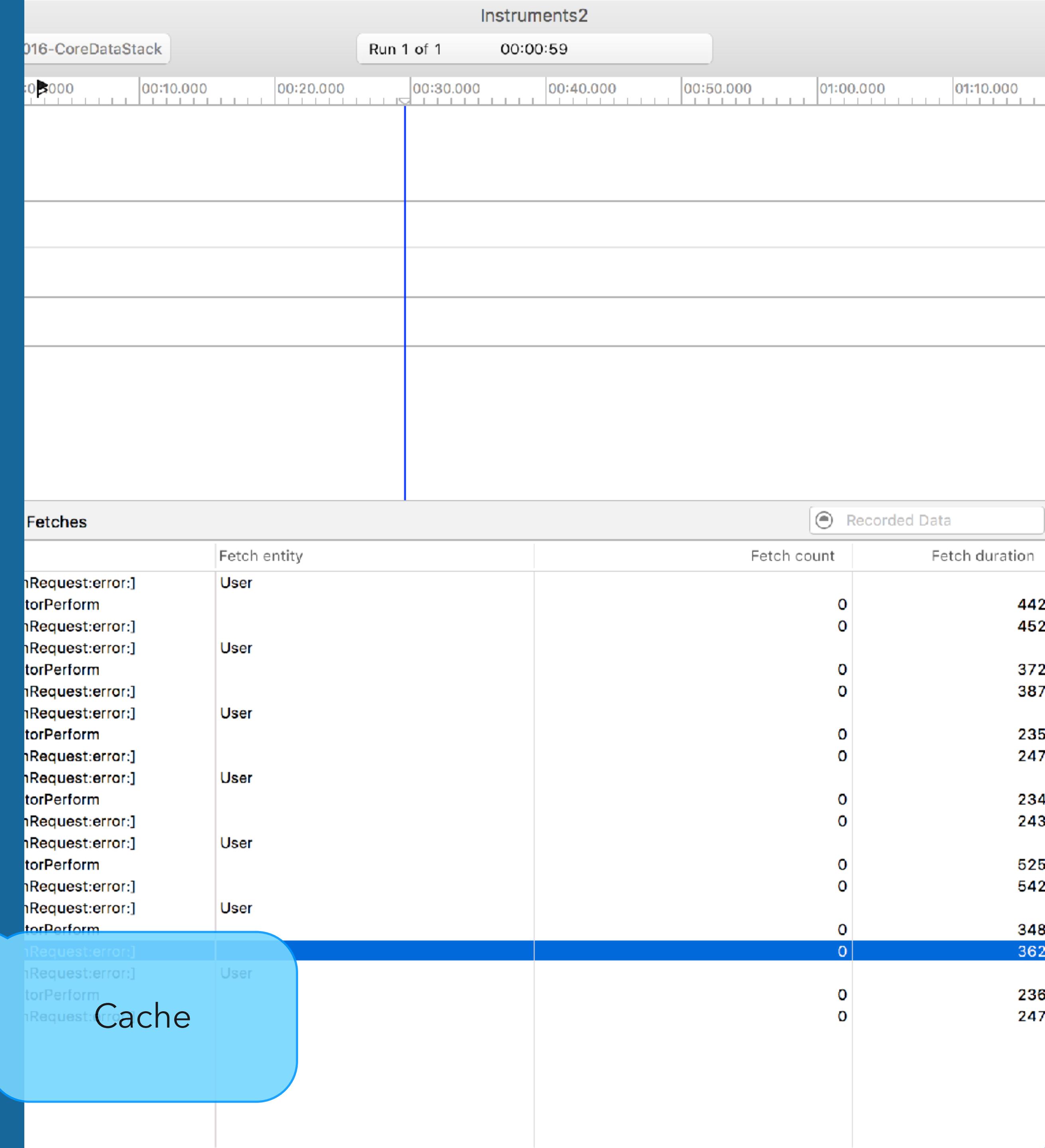
OPTIMIZING THE DATA MODEL



Best option for performance

OPTIMIZING THE DATA MODEL

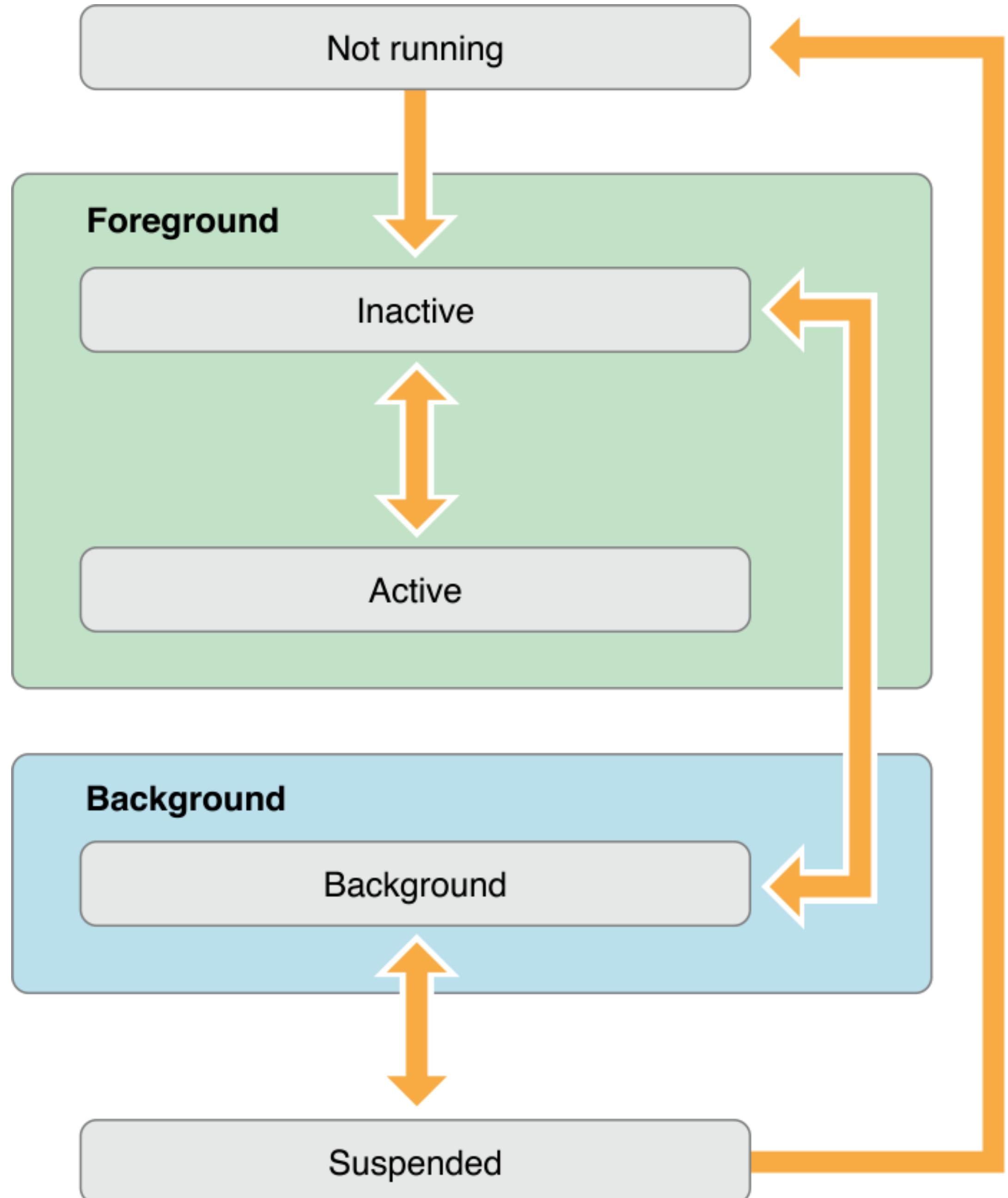
- A faulted Core Data object may already be in memory; it may be held in its `NSPersistentStoreCoordinator`'s cache
- If an object that isn't in the cache (a "cache miss"), the object has to be freshly read from the database.
- You want to minimize the effect of cache faults by preloading the objects when it doesn't impair user experience



BACKGROUND TASKS

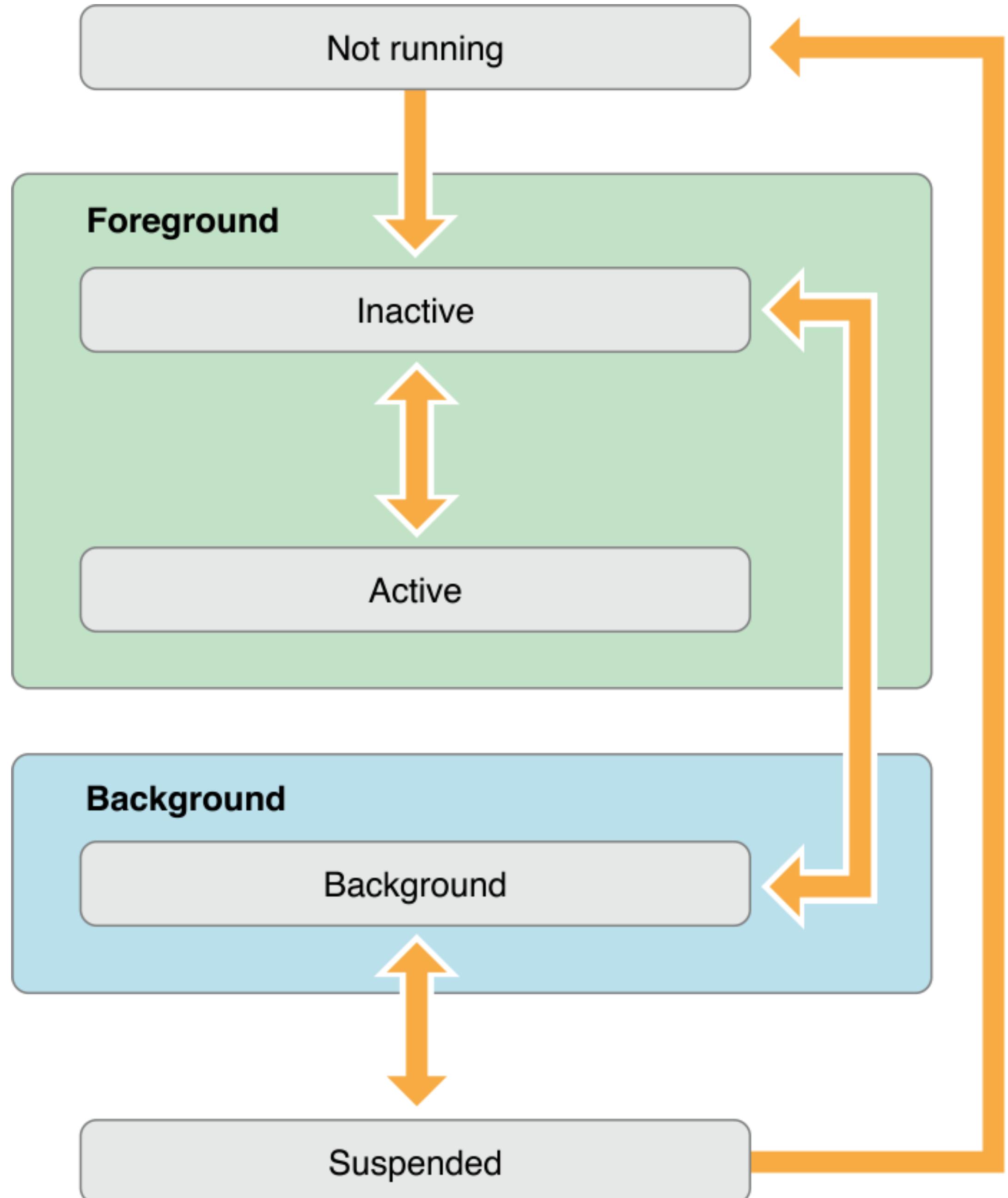
BACKGROUND TASKS

- State changes in iOS
- Background is a quick stop on way to suspended state
- Applications can be allotted additional time to run tasks in the background



BACKGROUND TASKS

- Legitimate background tasks
 - Track location for fitness app
 - Play music over lock screen
 - Download/update data so that it is ready for launch



BACKGROUND TASKS

- Three techniques on iOS
 - Finite-length tasks
 - Start a short task in the foreground and ask for time to finish that task when moved to background
 - URLSession background tasks
 - Initiate downloads in the foreground; hand off management of those downloads to the system
 - Background Execution modes
 - Support specific tasks can declare their support for one or more background execution modes

BACKGROUND TASKS

- Apple is strict about utility for background modes
- Some techniques have been abused, but don't count on it working for long

Apps Facebook

Facebook Says It Fixed A Bug That Caused Silent Audio To Vampire Your iPhone Battery

Posted Oct 22, 2015 by Matthew Panzarino (@panzer)

[Next Story](#)

1000 INVESTORS 200 COUNTRIES

VIVA TECHNOLOGY JUNE 15-16-17 / PARIS 2017

30% OFF ON YOUR PASS

AdChoices ▾

Crunchbase

Facebook

FOUNDED 2004

OVERVIEW

Facebook is an online social networking service that allows its users to connect with friends and family as well as make new connections. It provides its users

Facebook engineering manager Ari Grant has [posted an explanation](#) for the recent issues with the app gorging itself on power. Grant says that the battery issues were caused by a series of bugs including one that ate up extra CPU cycles, causing Facebook to use up more of your iPhone battery than it should.

BACKGROUND FETCH



BACKGROUND FETCH

- Special mode that will opportunistically fetch content
- Prevent "blinking data"
- ~ 30 seconds

The screenshot shows the Xcode Capabilities tab with the following configuration:

- Background Modes:**
 - Audio, AirPlay, and Picture in Picture
 - Location updates
 - Voice over IP
 - Newsstand downloads
 - External accessory communication
 - Uses Bluetooth LE accessories
 - Acts as a Bluetooth LE accessory
 - Background fetch
 - Remote notifications
- Steps:** ✓ Add the Required Background Modes key to your info.plist file

Below the capabilities list, there are several other sections:

- **Inter-App Audio**
- **Keychain Sharing**
- **Associated Domains**
- **App Groups**
- **Data Protection**
- **HomeKit**
- **Human Interface Guidelines**

BACKGROUND FETCH

- When a good opportunity arises, the system wakes or launches your app into the background and calls the app delegate's `application:performFetchWithCompletionHandler:` method
- Subject to OS scheduling (and punishment)
 - Following guidelines gives preferences to future runs

BACKGROUND FETCH

- Set background mode

- ▶  **Maps**
- ▶  **Personal VPN**
- ▶  **Network Extensions**
- ▼  **Background Modes**

- Modes:
- Audio, AirPlay, and Picture in Picture
 - Location updates
 - Voice over IP
 - Newsstand downloads
 - External accessory communication
 - Uses Bluetooth LE accessories
 - Acts as a Bluetooth LE accessory
 - Background fetch
 - Remote notifications

Steps: ✓ Add the Required Background Modes key to your info plist file

- ▶  **Inter-App Audio**
- ▶  **Keychain Sharing**
- ▶  **Associated Domains**
- ▶  **App Groups**
- ▶  **Data Protection**

BACKGROUND FETCH

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
    // Minimum size defined by the system  
    //UIApplication.shared.setMinimumBackgroundFetchInterval(UIApplicationBackgroundFetchIntervalMinimum)  
  
    // At least 10 seconds  
    UIApplication.shared.setMinimumBackgroundFetchInterval(10)  
  
    return true  
}
```

- Set intervals (constants or explicit)

BACKGROUND FETCH

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var backgroundSessionCompletionHandler: (() -> Void)?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool { }

    // Support for background fetch
    func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {

        // A. Find the view controller we want in the hierarchy
        if let tabBarController = window?.rootViewController as? UITabBarController {
            if let fetchViewController = tabBarController.viewControllers?[0] as? BackgroundFetchViewController {
                // B. Execute function in fetch view controller
                fetchViewController.fetchData {
                    // C. In completion block, update the UI
                    print("In completion block")
                    fetchViewController.updateInterface()
                    // D. Tell OS we're done so that it can update the multitasking
                    // snapshot
                    completionHandler(.newData)
                }
            }
        }
    }
}
```

Called for
background
fetch

BACKGROUND FETCH

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var backgroundSessionCompletionHandler: (() -> Void)?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool { ... }

    // Support for background fetch
    func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {

        // A. Find the view controller we want in the hierarchy
        if let tabBarController = window?.rootViewController as? UITabBarController {
            if let fetchViewController = tabBarController.viewControllers?[0] as? BackgroundFetchViewController {
                // B. Execute function in fetch view controller
                fetchViewController.fetchData {
                    // C. In completion block, update the UI
                    print("In completion block")
                    fetchViewController.updateInterface()
                    // D. Tell OS we're done so that it can update the multitasking
                    // snapshot
                    completionHandler(.newData)
                }
            }
        }
    }
}
```

Tell UI to
update itself

BACKGROUND FETCH

```
//  
//  ViewController.swift  
//  WatchYourBack  
//  
//  Created by T. Andrew Binkowski on 5/7/17.  
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.  
  
import UIKit  
  
class BackgroundFetchViewController: UIViewController {  
  
    var dateFormatter: DateFormatter = ...  
  
    @IBOutlet weak var timeLabel: UILabel!  
  
    var currentTime: Date? {  
        didSet {  
            if let currentTime = self.currentTime { ... }  
        }  
    }  
    //  
    // MARK: -  
    //  
    override func viewDidLoad() {...}  
  
    override func didReceiveMemoryWarning() {...}  
  
    func fetchData(_ completion: () -> Void) {...}  
    func updateInterface() {...}  
}
```

View controller that has functions called in background task

BACKGROUND FETCH

```
class BackgroundFetchViewController: UIViewController {  
  
    var dateFormatter: DateFormatter = ...  
  
    @IBOutlet weak var timeLabel: UILabel!  
  
    var currentTime: Date? {  
        didSet {  
            if let currentTime = self.currentTime {  
                self.timeLabel.text = dateFormatter.string(from: currentTime)  
            }  
        }  
    }  
  
    //  
    // MARK: -  
    //  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        self.updateInterface()  
    }  
  
    //  
    // MARK: - Fetching and Updating  
    //  
    func fetchData(_ completion: () -> Void) {  
        // Some long running task  
        for i in 0...100 {  
            print(i)  
        }  
        completion()  
    }  
  
    func updateInterface() {  
        print("UpdateUI")  
        self.currentTime = Date()  
    }  
}
```

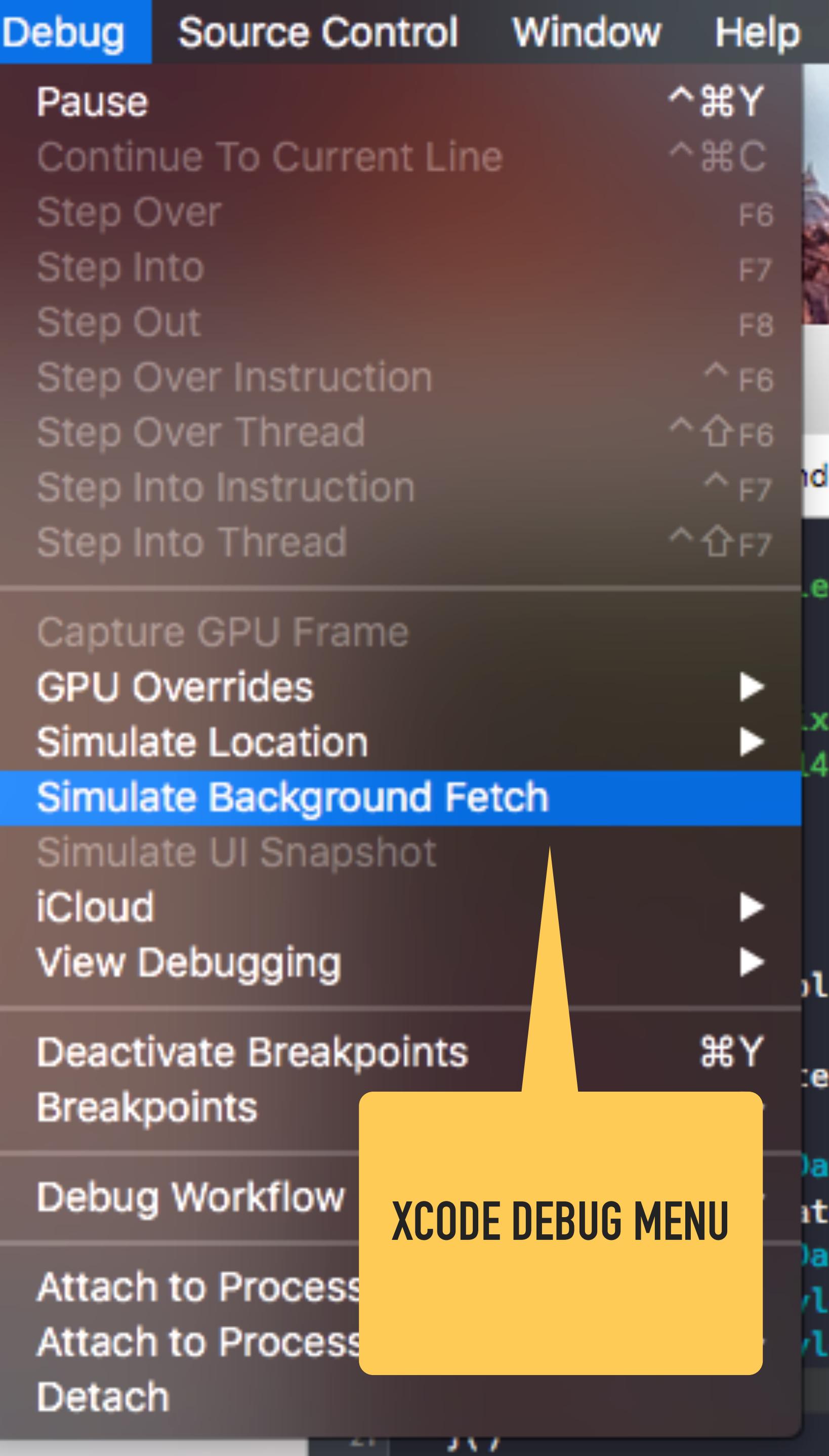
Keep track of time and update the UI when changed

Long-running task (for show)

Update the label

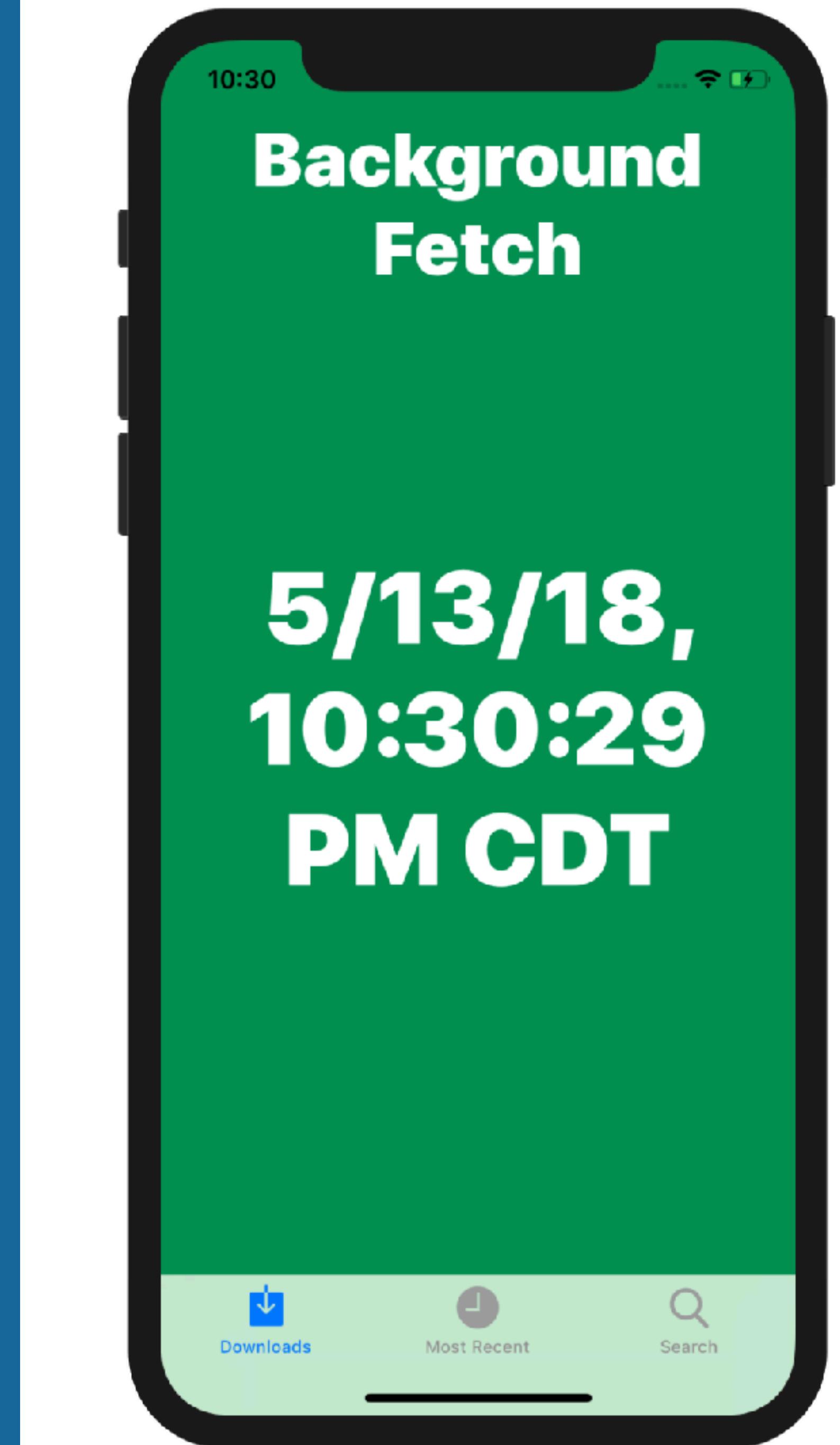
BACKGROUND FETCH

- Testing on device is unpredictable
- For once, testing on simulator is easier



BACKGROUND FETCH

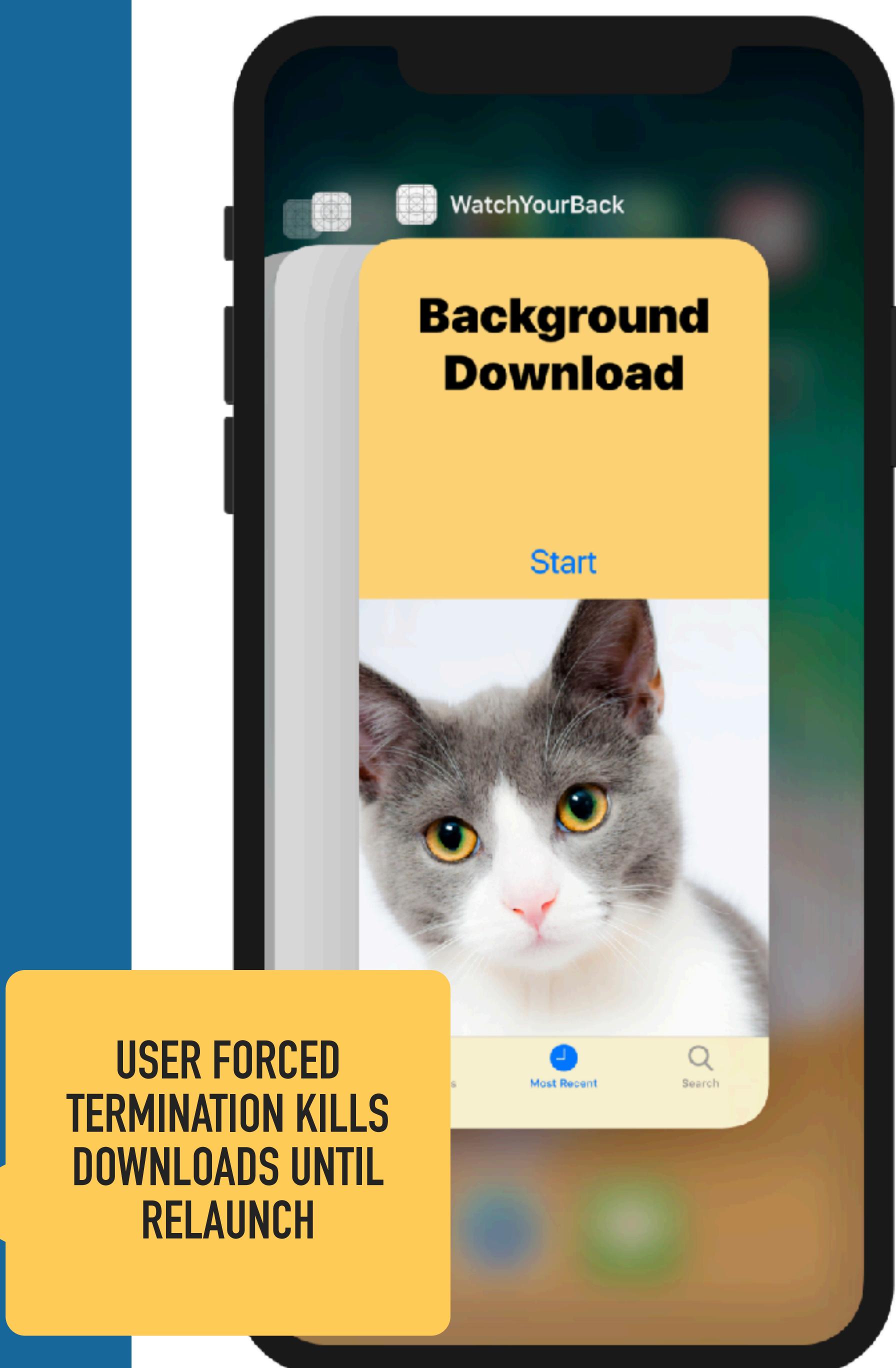
- Screenshot is updated in multitasking view



BACKGROUND DOWNLOAD

BACKGROUND DOWNLOAD

- URLSession provides built-in support for background downloads
- System manages them through suspension or termination
 - Will restart app when download complete



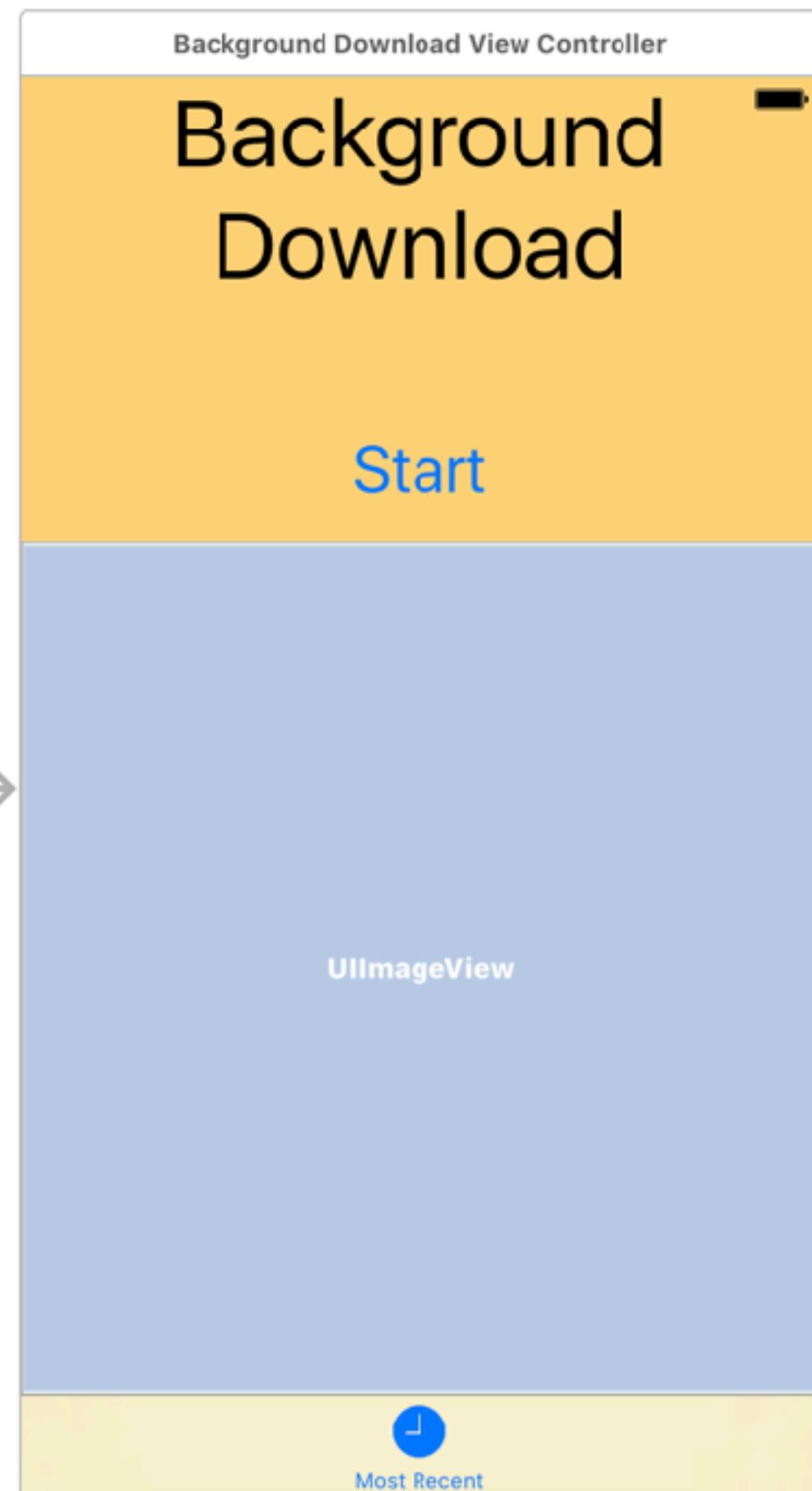
BACKGROUND DOWNLOAD

Set up URLSession

- Setting up a background download
 - Create the configuration object using the `backgroundSessionConfigurationWithIdentifier:` method of `NSURLSessionConfiguration`
 - Set the value of the configuration object's `sessionSendsLaunchEvents` property to true
 - If your app starts transfers while it is in the foreground, it is recommend that you also set the `discretionary` property of the configuration object to YES
 - Configure any other properties of the configuration object as appropriate.
 - Use the configuration object to create your `NSURLSession` object

BACKGROUND DOWNLOAD

```
//  
// BackgroundDownloadViewController.swift  
// WatchYourBack  
//  
// Created by T. Andrew Binkowski on 5/7/17.  
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.  
  
import UIKit  
  
class BackgroundDownloadViewController: UIViewController {  
  
    /// Start a download that will persist through moving to background  
    @IBAction func tapDownload(_ sender: Any) {  
        //  
    }  
  
    /// ImageView to show our downloaded image  
    @IBOutlet weak var imageView: UIImageView!  
}  
  
///  
///  
///  
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {  
  
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,  
                   didWriteData bytesWritten: Int64,  
                   totalBytesWritten: Int64,  
                   totalBytesExpectedToWrite: Int64) {  
        //  
    }  
  
    func urlSession(_ session: URLSession,  
                   task: URLSessionTask,  
                   didCompleteWithError error: Error?) {  
        //  
    }  
  
    func urlSession(_ session: URLSession,  
                   downloadTask: URLSessionDownloadTask,  
                   didFinishDownloadingTo location: URL){  
        //  
    }  
}
```



BACKGROUND DOWNLOAD

```
class BackgroundDownloadViewController: UIViewController {  
  
    /// Start a download that will persist through moving to background  
    @IBAction func tapDownload(_ sender: Any) {  
        print("Tap download")  
        let config = URLSessionConfiguration.background(withIdentifier: "mobi.uchicago.download")  
        config.sessionSendsLaunchEvents = true  
        let session = URLSession(configuration: config,  
                               delegate: self,  
                               delegateQueue: OperationQueue())  
        let url = URL(string: "https://static.pexels.com/photos/104827/cat-pet-animal-domestic-104827.jpeg")  
        let downloadTask = session.downloadTask(with: url!)  
        downloadTask.resume()  
    }  
  
    /// ImageView to show our downloaded image  
    @IBOutlet weak var imageView: UIImageView!  
}  
  
///  
///  
///  
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {
```

BACKGROUND DOWNLOAD

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var backgroundSessionCompletionHandler: (() -> Void)?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool { ... }

    func application(_ application: UIApplication,
                    handleEventsForBackgroundURLSession identifier: String,
                    completionHandler: @escaping () -> Void) {
        print("handleEventsForBackgroundURLSession: \(identifier)")
        self.backgroundSessionCompletionHandler = completionHandler
    }
}
```

Assign the completion
handler to a property to
keep it alive

BACKGROUND DOWNLOAD

```
import UIKit

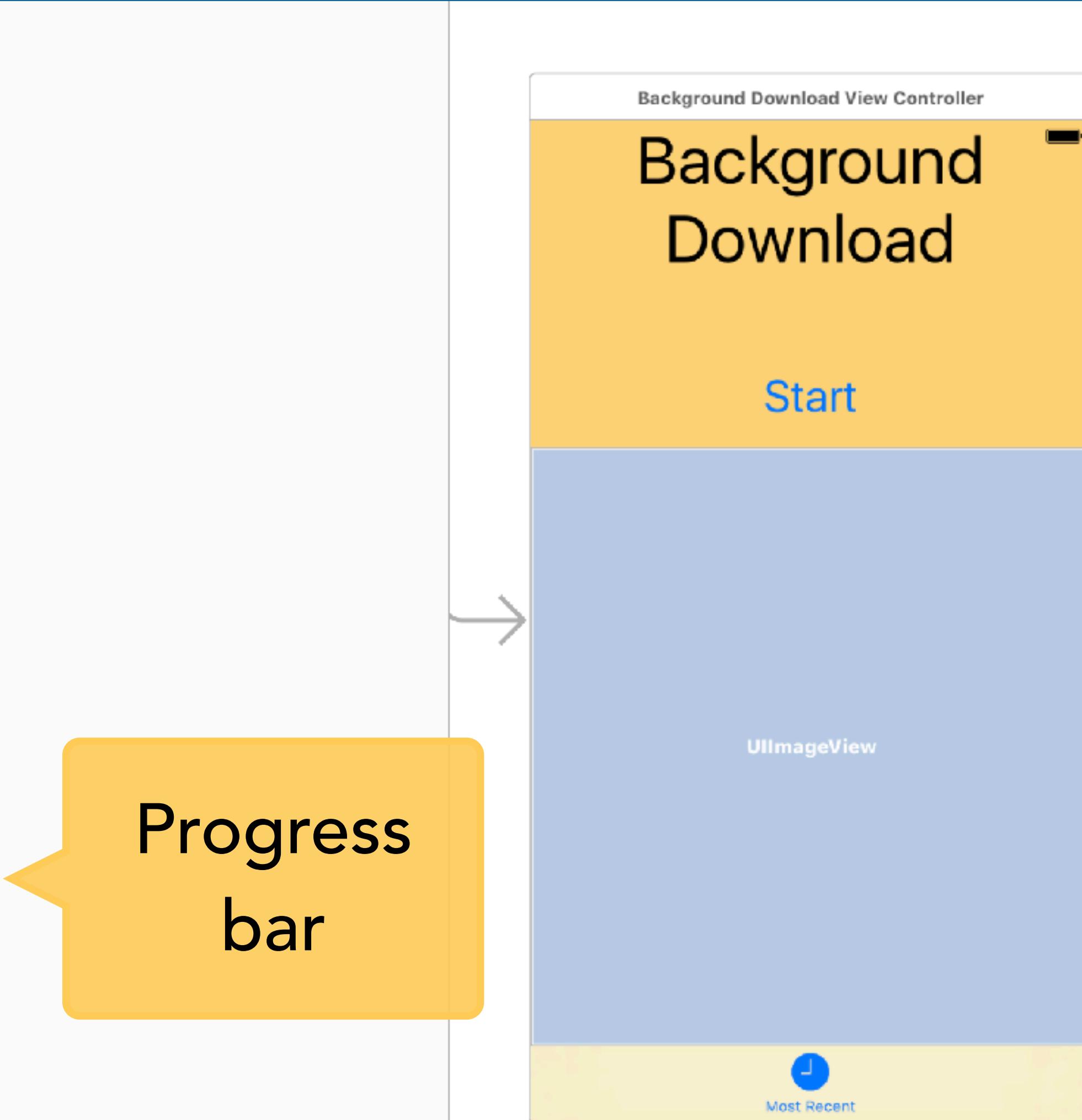
class BackgroundDownloadViewController: UIViewController {
    // Start a download that will persist through moving to background
    @IBAction func tapDownload(_ sender: Any) { ... }

    // ImageView to show our downloaded image
    @IBOutlet weak var imageView: UIImageView!
}

/// 
/// 
/// extension BackgroundDownloadViewController: URLSessionDownloadDelegate {

    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,
                    didWriteData bytesWritten: Int64,
                    totalBytesWritten: Int64,
                    totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            print("Progress \(downloadTask) \(progress)")
        }
    }

    func urlSession(_ session: URLSession,
                    task: URLSessionTask,
                    didCompleteWithError error: Error?) { ... }
}
```



BACKGROUND DOWNLOAD

```
///  
func urlSession(_ session: URLSession,  
                 task: URLSessionTask,  
                 didCompleteWithError error: Error?) {  
    print("Task completed: \(task), error: \(String(describing: error))")  
}  
  
///  
func urlSession(_ session: URLSession,  
                 downloadTask: URLSessionDownloadTask,  
                 didFinishDownloadingTo location: URL){  
  
    print("Download finished: \(location)")  
  
    let path = NSSearchPathForDirectoriesInDomains(FileManager.SearchPathDirectory.documentDirectory,  
                                                FileManager.SearchPathDomainMask.userDomainMask,  
                                                true)[0]  
    let fileManager = FileManager()  
    let destinationURLOfFile = URL(fileURLWithPath: path.appendingFormat("/downloaded.jpg"))  
    try? fileManager.removeItem(at: destinationURLOfFile)  
  
    do {  
        try fileManager.moveItem(at: location, to: destinationURLOfFile)  
        let image = UIImage(contentsOfFile: path.appendingFormat("/downloaded.jpg"))  
        DispatchQueue.main.async {  
            self.imageView.image = image  
        }  
    } catch {  
        print("An error occurred while moving file to destination url")  
    }  
}
```

Called when done



BACKGROUND DOWNLOAD

```
func urlSession(_ session: URLSession,
                downloadTask: URLSessionDownloadTask,
                didFinishDownloadingTo location: URL){

    print("Download finished: \(location)")

    let path = NSSearchPathForDirectoriesInDomains(FileManager.SearchPathDirectory.documentDirectory,
                                                FileManager.SearchPathDomainMask.userDomainMask,
                                                true)[0]
    let fileManager = FileManager()
    let destinationURLOfFile = URL(fileURLWithPath: path.appendingFormat("/downloaded.jpg"))
    try? fileManager.removeItem(at: destinationURLOfFile)

    do {
        try fileManager.moveItem(at: location, to: destinationURLOfFile)
        updateInterface()
    } catch {
        print("An error occurred while moving file to destination url")
    }
}

/// Update the UI. Run from background to trigger snapshot refresh
func updateInterface() {
    let path = NSSearchPathForDirectoriesInDomains(FileManager.SearchPathDirectory.documentDirectory,
                                                FileManager.SearchPathDomainMask.userDomainMask,
                                                true)[0]
    if let image = UIImage(contentsOfFile: path.appendingFormat("/downloaded.jpg")) {
        DispatchQueue.main.async {
            print("Updating image")
            self.imageView.image = image
        }
    }
}
```

Update UI

Get file from /tmp and
update image

BACKGROUND DOWNLOAD

- Supported by URLSession on iOS and watchOS



FINITE-LENGTH TASKS

FINITE-LENGTH TASKS

- Execute a task in the background

```
import UIKit

class BackgroundTaskViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        longRunningTask()
    }

    func longRunningTask() {
        var bti : UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid
        bti = UIApplication.shared.beginBackgroundTask(expirationHandler: {
            // Clean up after run (or denial)

            // Call the completion handler
            UIApplication.shared.endBackgroundTask(bti)
        })

        // Do work
        for i in 0...10000000 {
            print(i)
        }

        DispatchQueue.main.async {
            UIApplication.shared.endBackgroundTask(bti)
        }
    }
}
```

FINITE-LENGTH TASKS

```
func longRunningTask() {  
    var bti : UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid  
    bti = UIApplication.shared.beginBackgroundTask(expirationHandler: {  
        // Clean up after run (or denial)  
  
        // Call the completion handler  
        UIApplication.shared.endBackgroundTask(bti)  
    })  
  
    // Do work  
    for i in 0...10000000 {  
        print(i)  
    }  
  
    DispatchQueue.main.async {  
        UIApplication.shared.endBackgroundTask(bti)  
    }  
}
```

0

Called if system cancels job

Work

Tell OS you're done



apple WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5



apple WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

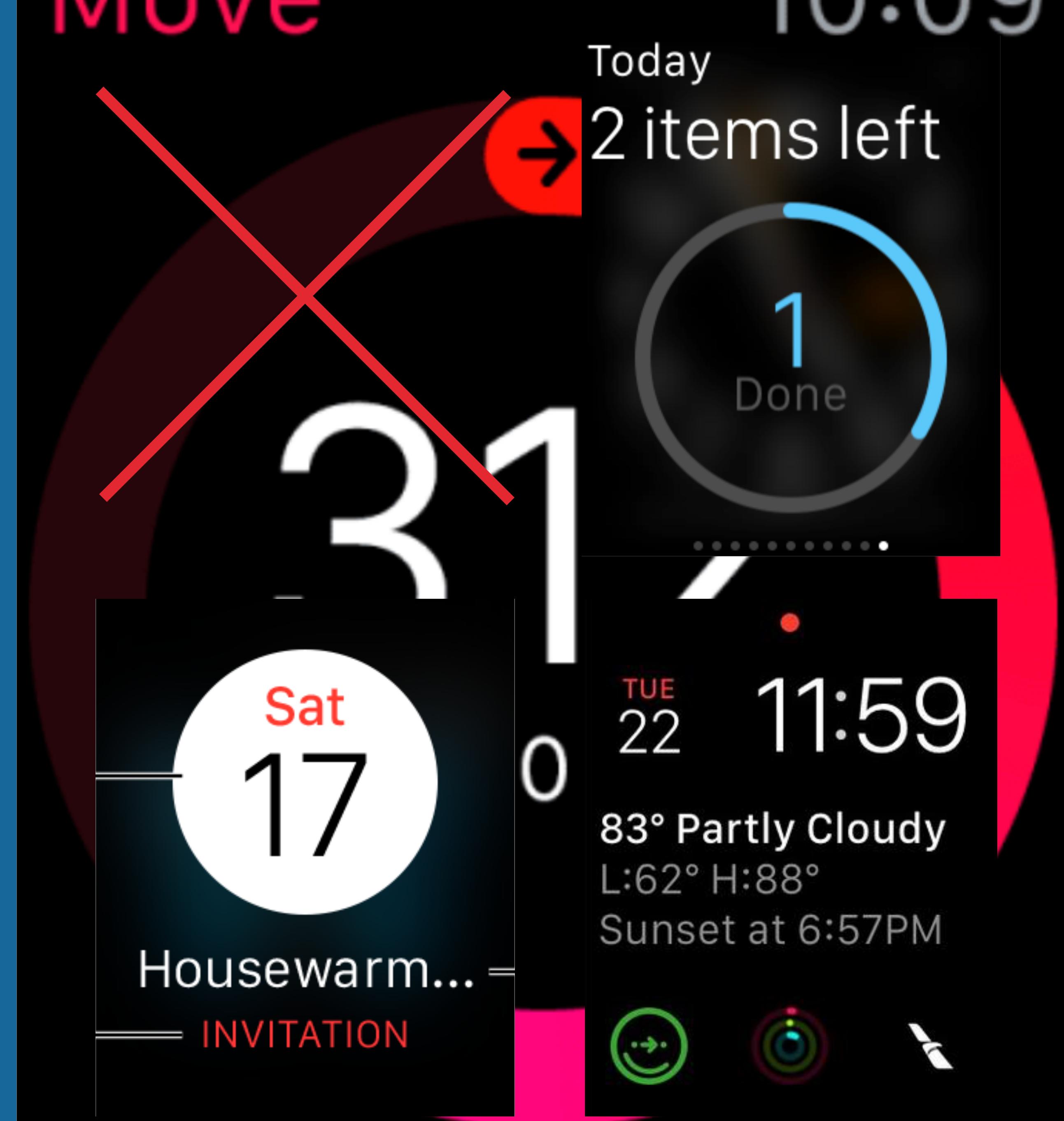
USER INTERACTIONS ON



USER INTERACTIONS

SUBTITLE

- User interactions available
 - WatchKit App
 - Glances
 - Dock
 - Custom notifications
 - Complication (new in 2.0)



USER INTERACTIONS

- What makes up a Watch App?
 - Interface
 - Notification Interface
 - Complication Interface
 - Code for managing all interfaces in extension

The screenshot shows the Xcode interface with a project titled "InterFaces" selected. The project structure is as follows:

- InterFaces (Target)
 - InterFaces (Group)
 - AppDelegate.swift
 - ViewController.swift
 - Main.storyboard
 - Assets.xcassets
 - LaunchScreen.storyboard
 - Info.plist
 - InterFaces WatchKit App (Group)
 - Interface storyboard
 - Assets.xcassets
 - Info.plist
 - InterFaces WatchKit Extension (Group)
 - InterfaceController.swift
 - ExtensionDelegate.swift
 - NotificationController.swift
 - GlanceController.swift
 - ComplicationController.swift
 - Assets.xcassets
 - Info.plist
 - Supporting Files
 - Products

The code editor on the right displays the content of the `InterfaceController.swift` file:

```
// InterfaceController.swift
// InterFaces WatchKit Extension
// Created by T. Andrew Bi... 2016 The University of Texas at Austin
// Copyright © 2016 The University of Texas at Austin. All rights reserved.

import WatchKit
import Foundation
import WatchConnectivity

class InterfaceController: WKInterfaceController {

    // MARK: - IBOutlets
    @IBOutlet var nameLabel: WKInterfaceLabel

    // MARK: - Connectivity session
    let session = WCSession.default()

    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // Set the name label
        nameLabel.setText("Hello, World!")
        // Check if the session is connected
        if session.isConnected {
            session.sendMessage(["text": "Hello, World!"], replyHandler: { [weak self] (reply) in
                self?.nameLabel.setText(reply["text"] as? String)
            }, errorHandler: { [weak self] (error) in
                print("Error: \(error.localizedDescription)")
            })
        }
    }

    @IBAction func tapToInteractive(_ sender: WKInterfaceButton) {
        print("Tap (interactive)")
        let applicationDict = [
            "text": "Hello, World!"
        ]
        session.sendMessage(applicationDict, replyHandler: { [weak self] (reply) in
            self?.nameLabel.setText(reply["text"] as? String)
        }, errorHandler: { [weak self] (error) in
            print("Error: \(error.localizedDescription)")
        })
    }

    @IBAction func tapToApplication(_ sender: WKInterfaceButton) {
        print("Tap (application)")
        do {
            let applicationDict = [
                "text": "Hello, World!"
            ]
            try session.default().sendMessage(applicationDict)
        } catch {
            // Handle errors here
            print(error)
        }
    }

    // MARK: - Lifecycle
    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // Set the name label
        nameLabel.setText("Hello, World!")
        // Check if the session is connected
        if session.isConnected {
            session.sendMessage(["text": "Hello, World!"], replyHandler: { [weak self] (reply) in
                self?.nameLabel.setText(reply["text"] as? String)
            }, errorHandler: { [weak self] (error) in
                print("Error: \(error.localizedDescription)")
            })
        }
    }
}
```

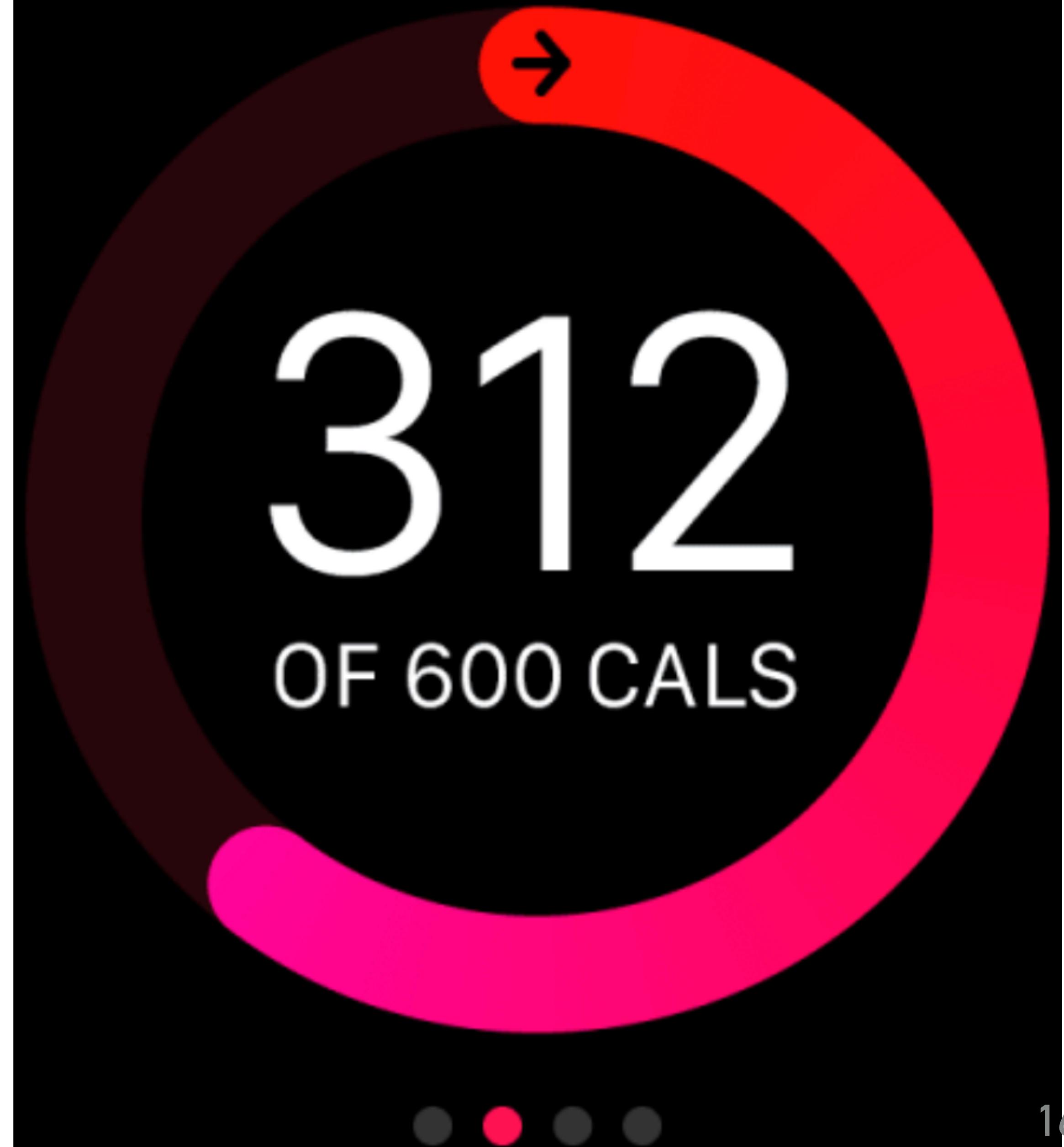
USER INTERACTIONS

SUBTITLE

- WatchKit App
 - Full-app experience which they interact with by opening your app from the Home screen
 - Main way to interact with data
 - Typically, users will only interact with a subset of data/features

Move

10:09



WATCH APP

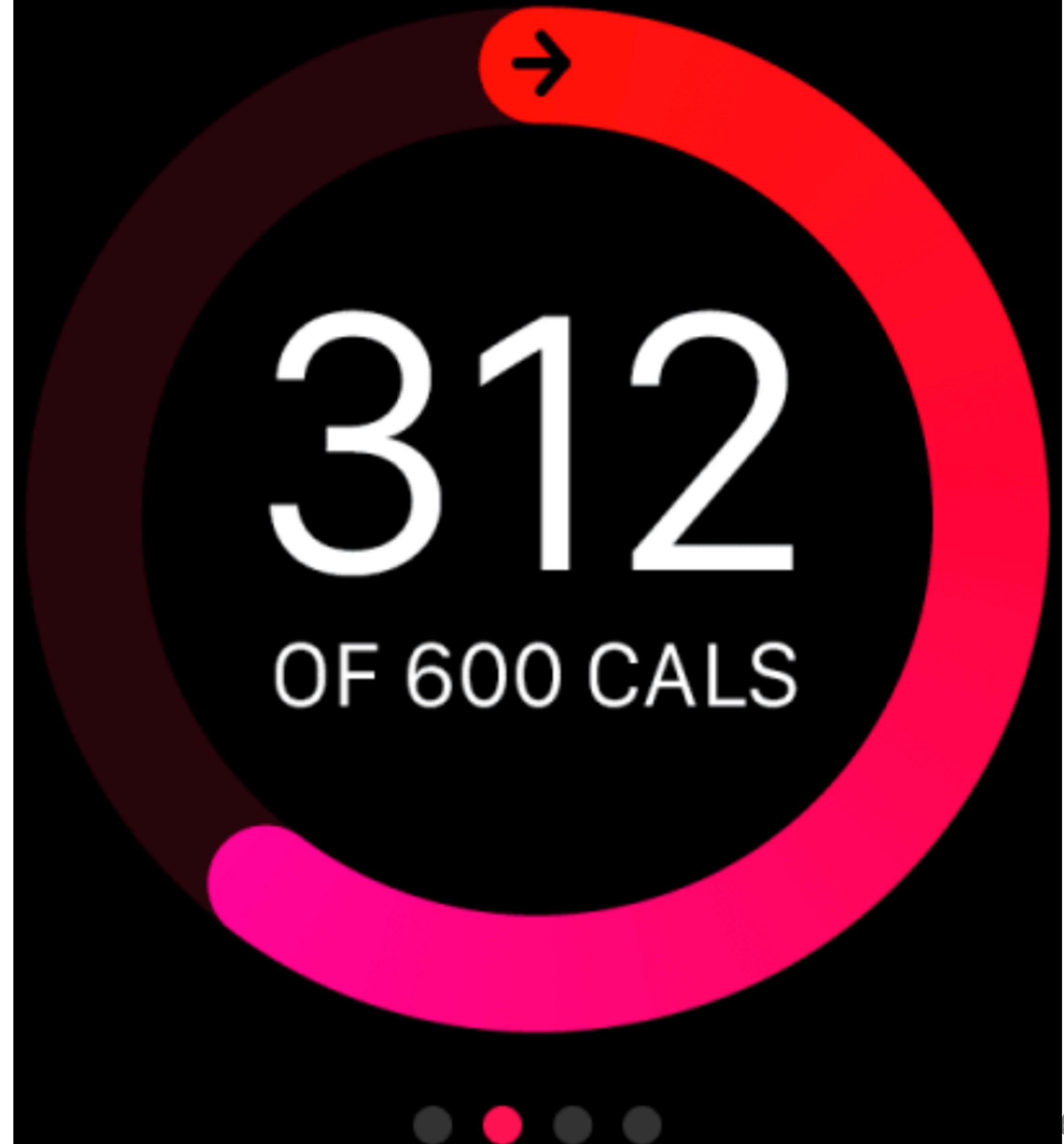
USER INTERACTIONS

SUBTITLE

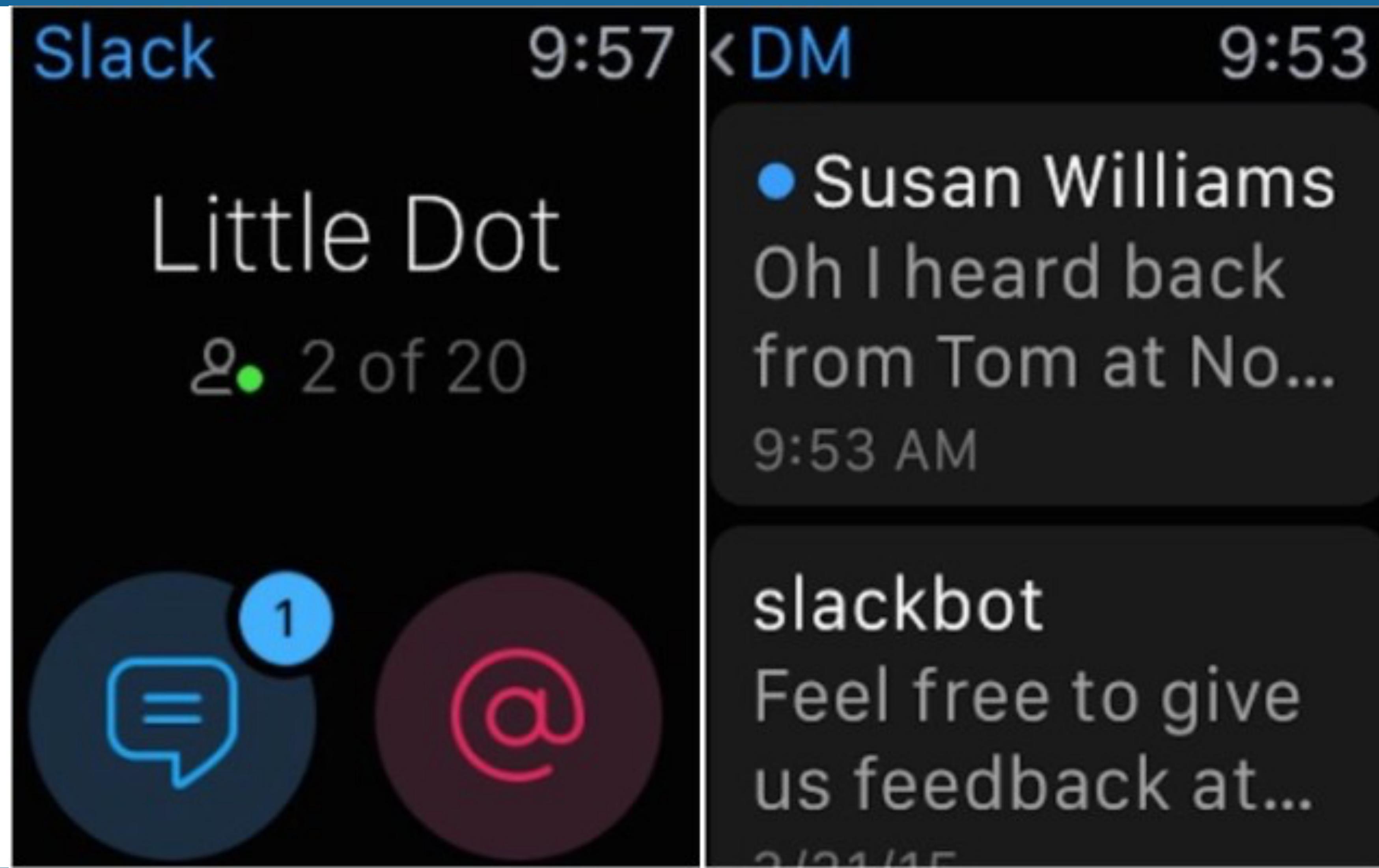
- Watch apps have different navigation and interaction models
 - Health app
 - Music app
 - Game app
 - App app

Move

10:09



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS

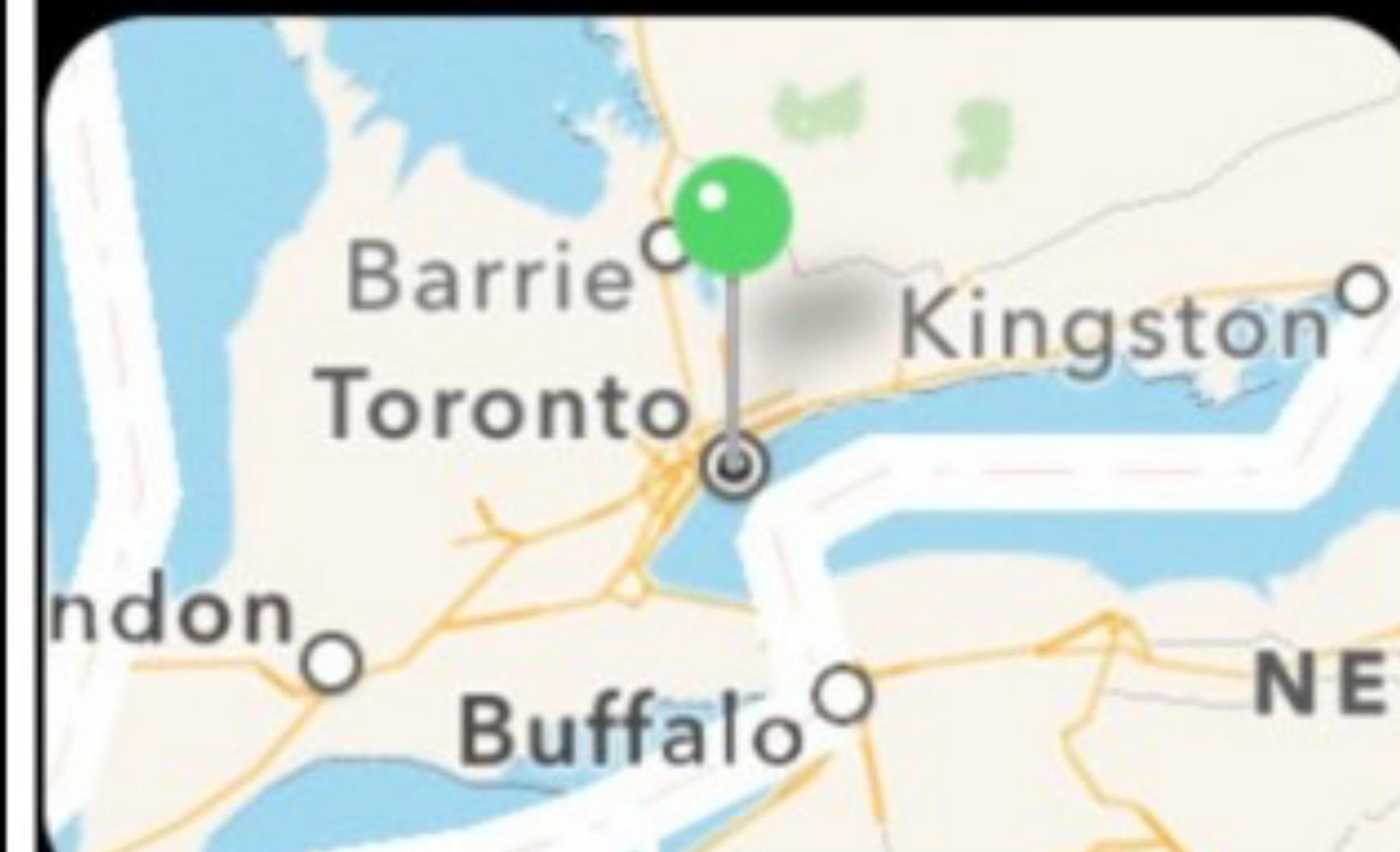
Deliveries 10:09

 Sailor Moon S...
Delivered: Nort...

3 DAYS Seconds and...
In transit: Toro...

8 DAYS 2 11-inch Ma...
Ship date unkn...
Updated a moment ago

< Seconds an... 10:09
3 days
In transit: Toronto, ON
Delivered by Saturday



USER INTERACTIONS



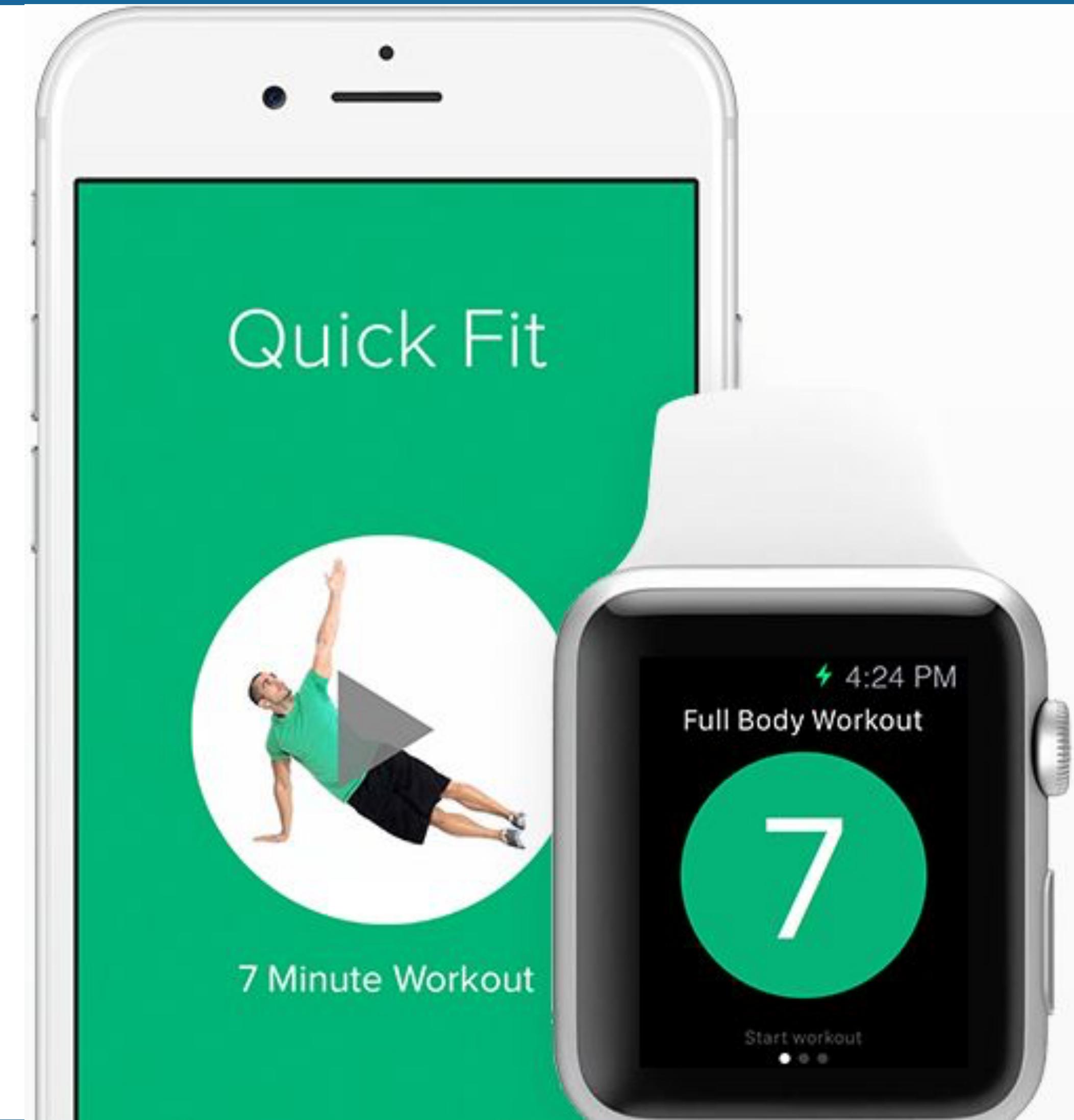
USER INTERACTIONS



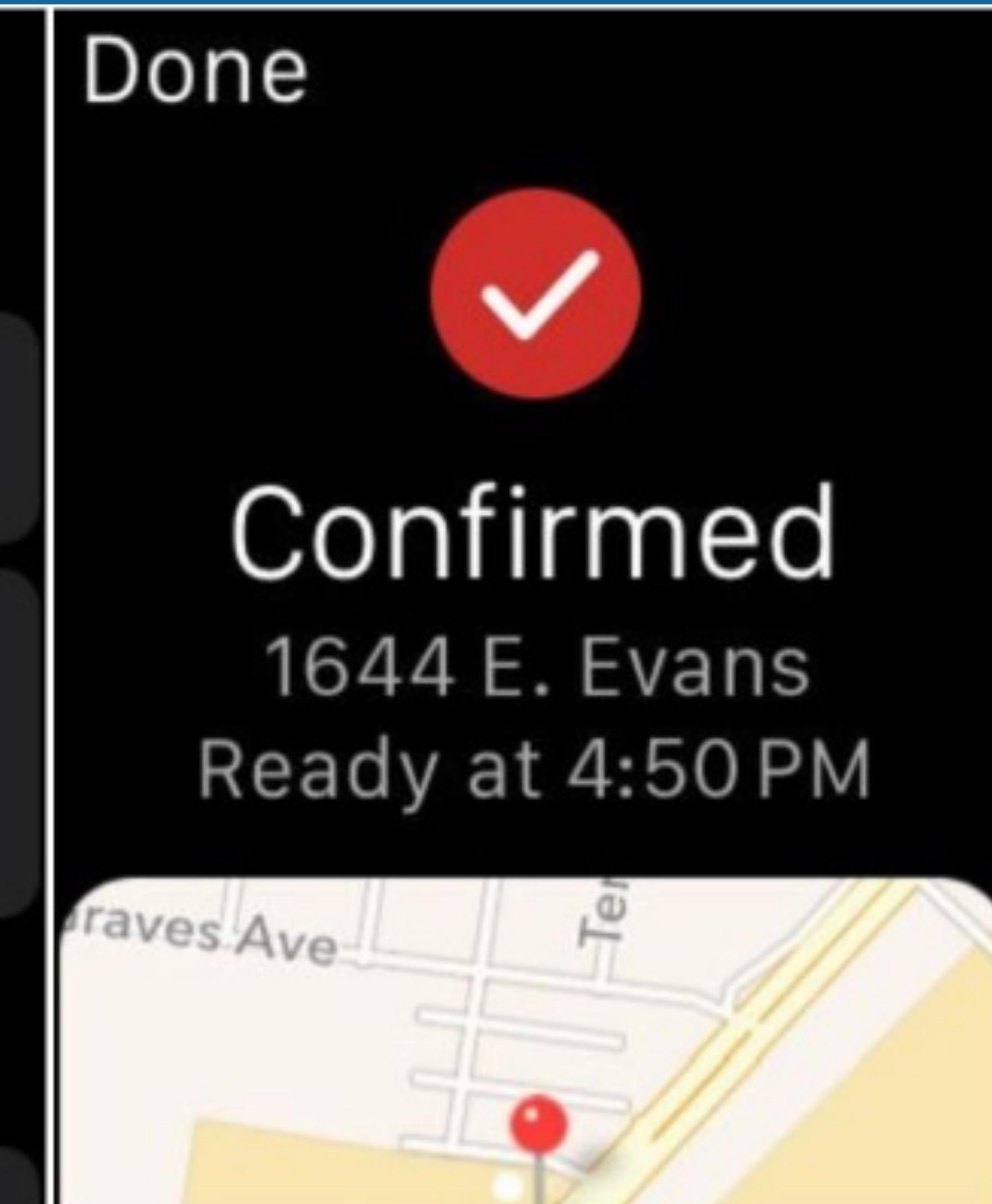
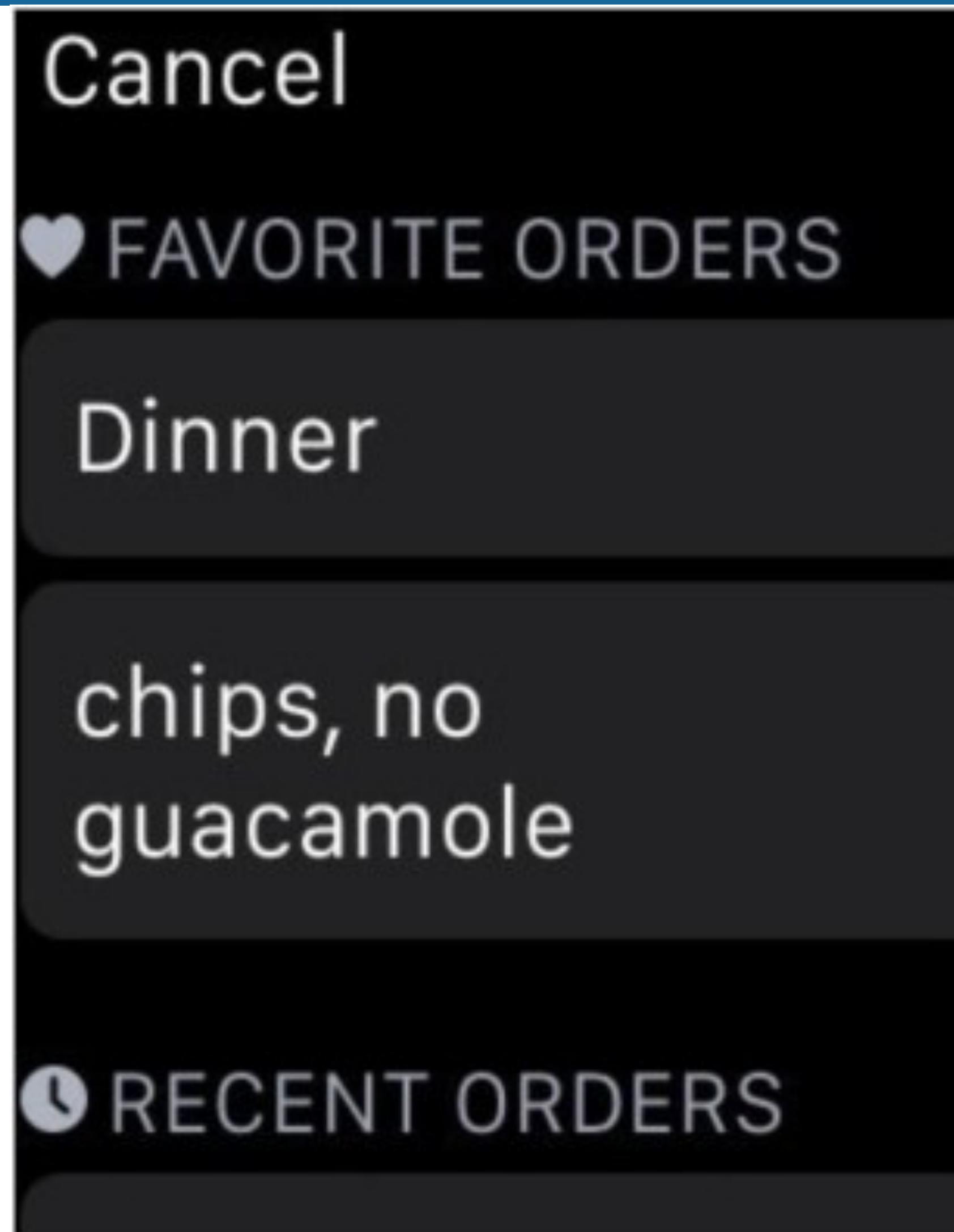
USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS



GLANCES

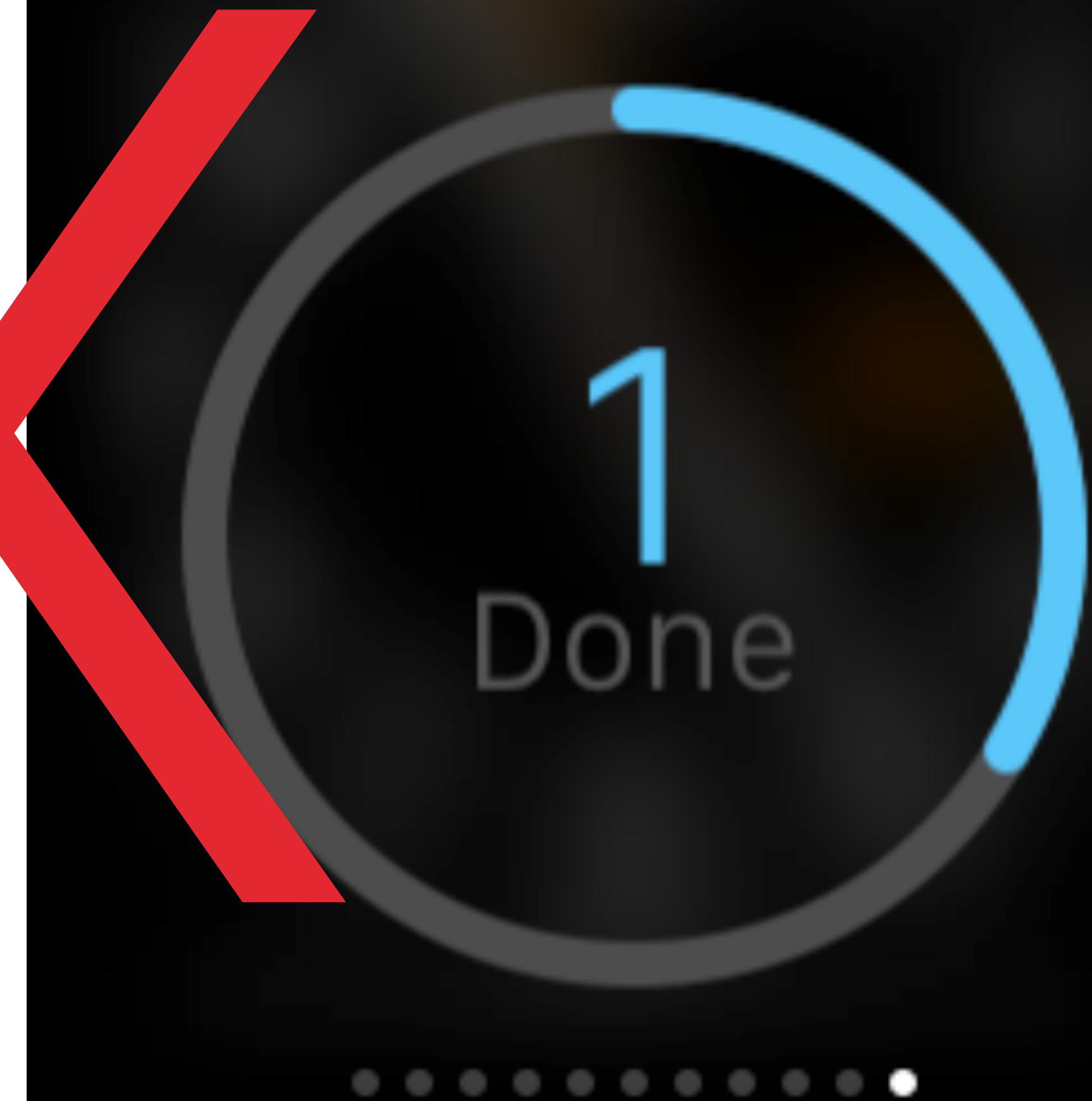


USER INTERACTIONS

SUBTITLE

- Glances
 - A focused interface that you use to display your app's most important information
 - Glances are nonscrolling and read-only
 - Cannot contain buttons, switches, or other interactive controls
 - Tapping a glance launches your WatchKit app
 - “Not to be used as an app launcher”

Today
2 items left



USER INTERACTIONS

GLANCES HAS BEEN REPLACED BY THE DOCK



NOTIFICATIONS

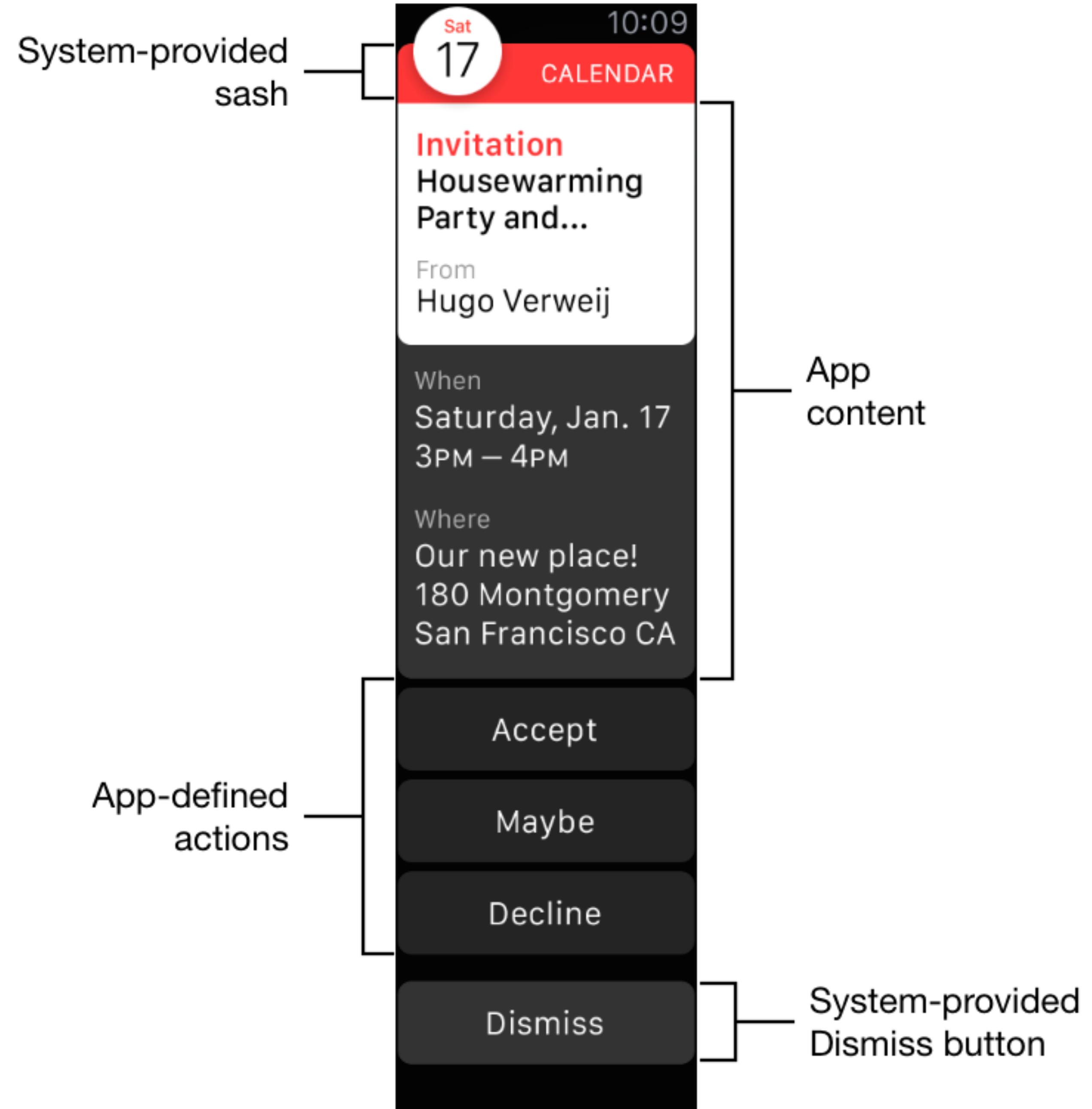
USER INTERACTIONS

- Custom Notifications
 - Works with iPhone to display local and remote notifications
 - Providing custom notification interfaces
 - Short
 - Long



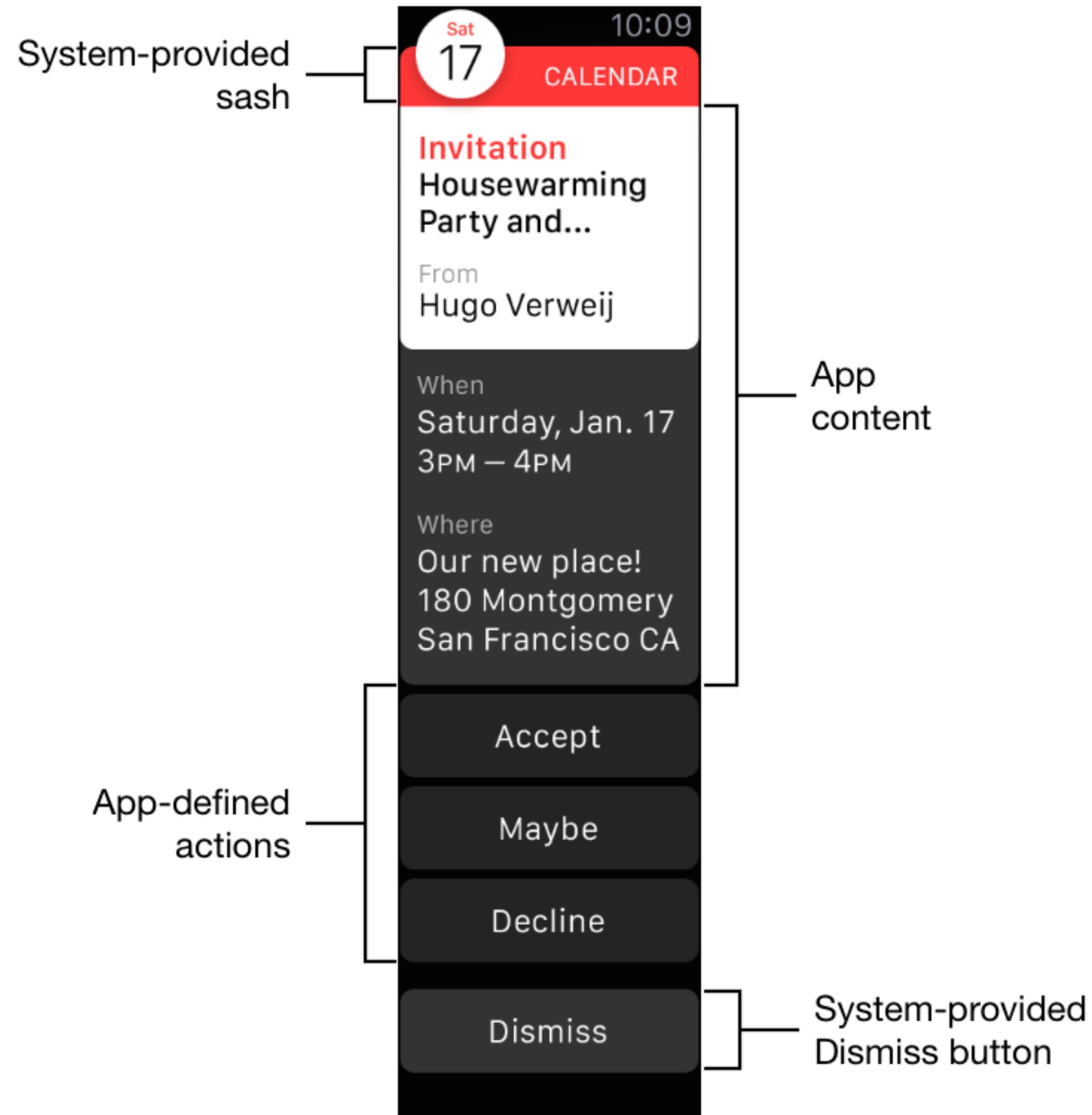
USER INTERACTIONS

- Long looks notifications
- Actionable notifications to increase functionality



USER INTERACTIONS

- watchOS works with UNNotification framework
- Schedule and handle location notification on watch using extension
 - Time, location based
- Handle remote notifications delivered to watch

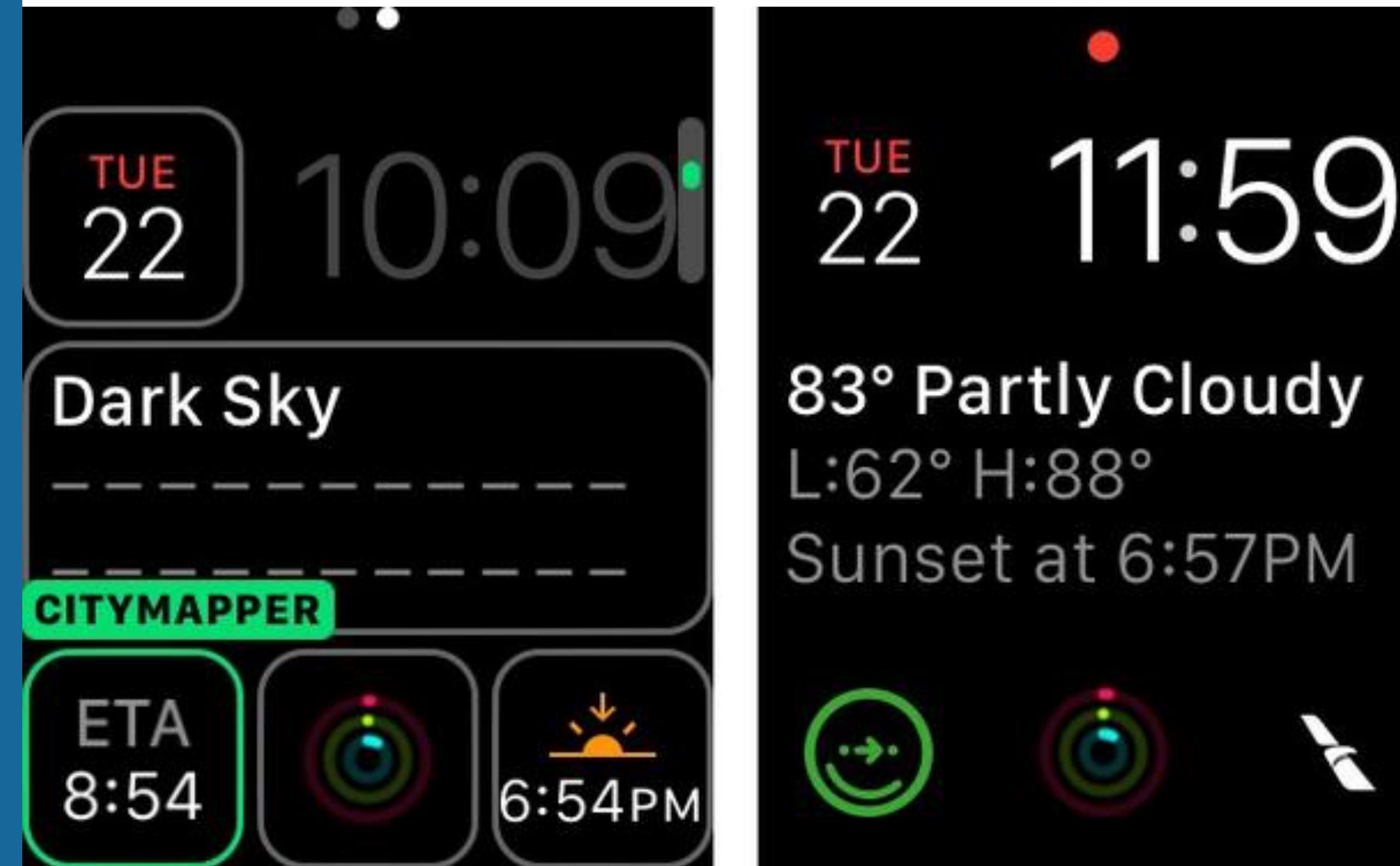


COMPLICATIONS

USER INTERACTIONS

COMPLICATIONS SUBTITLE

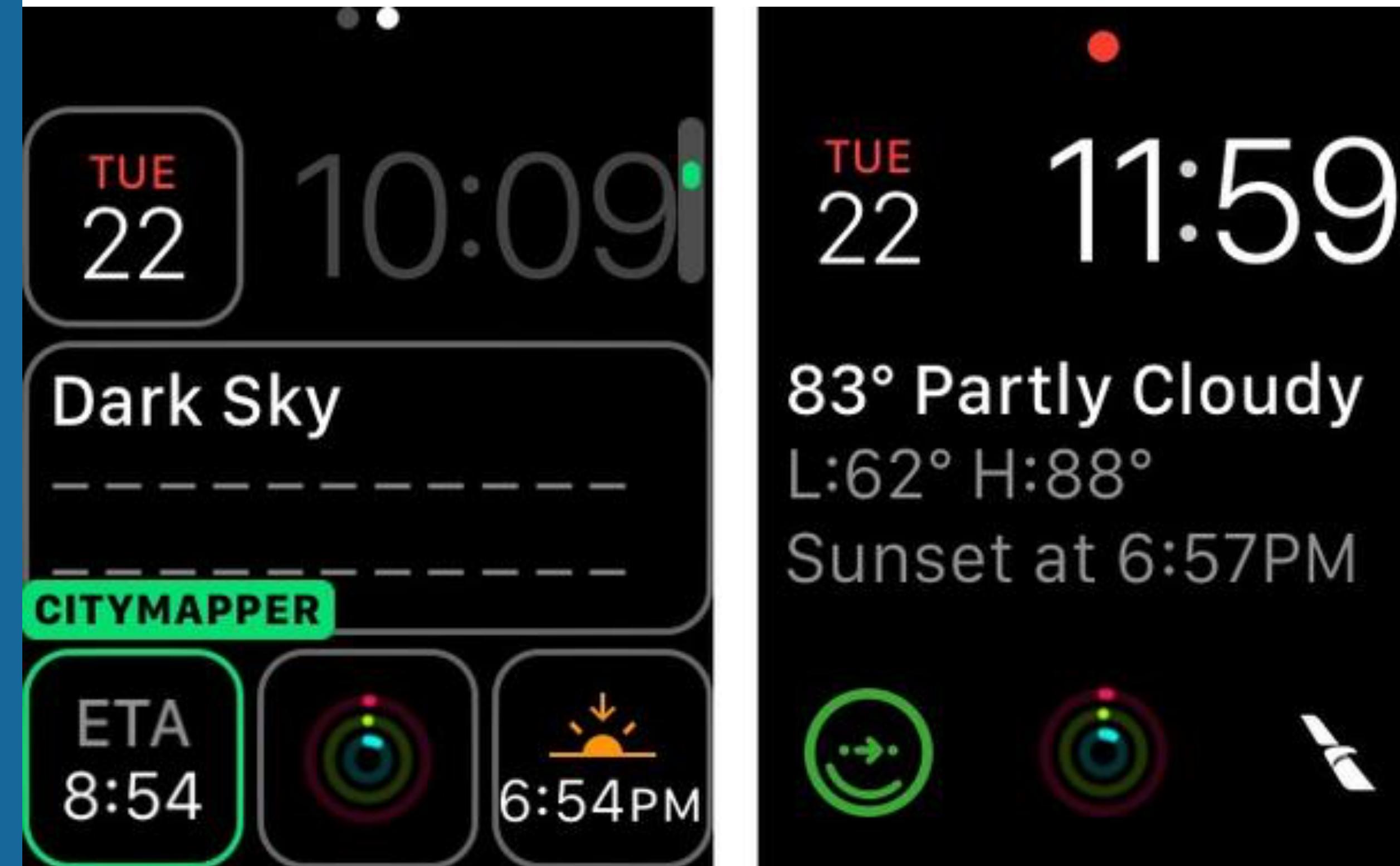
- Small visual elements that appear on the watch face
- Customizable
- Always visible



USER INTERACTIONS

COMPLICATIONS SUBTITLE

- The term *complication* comes from watch making, where the addition of features added complexity to the watch construction



USER INTERACTIONS



USER INTERACTIONS



USER INTERACTIONS

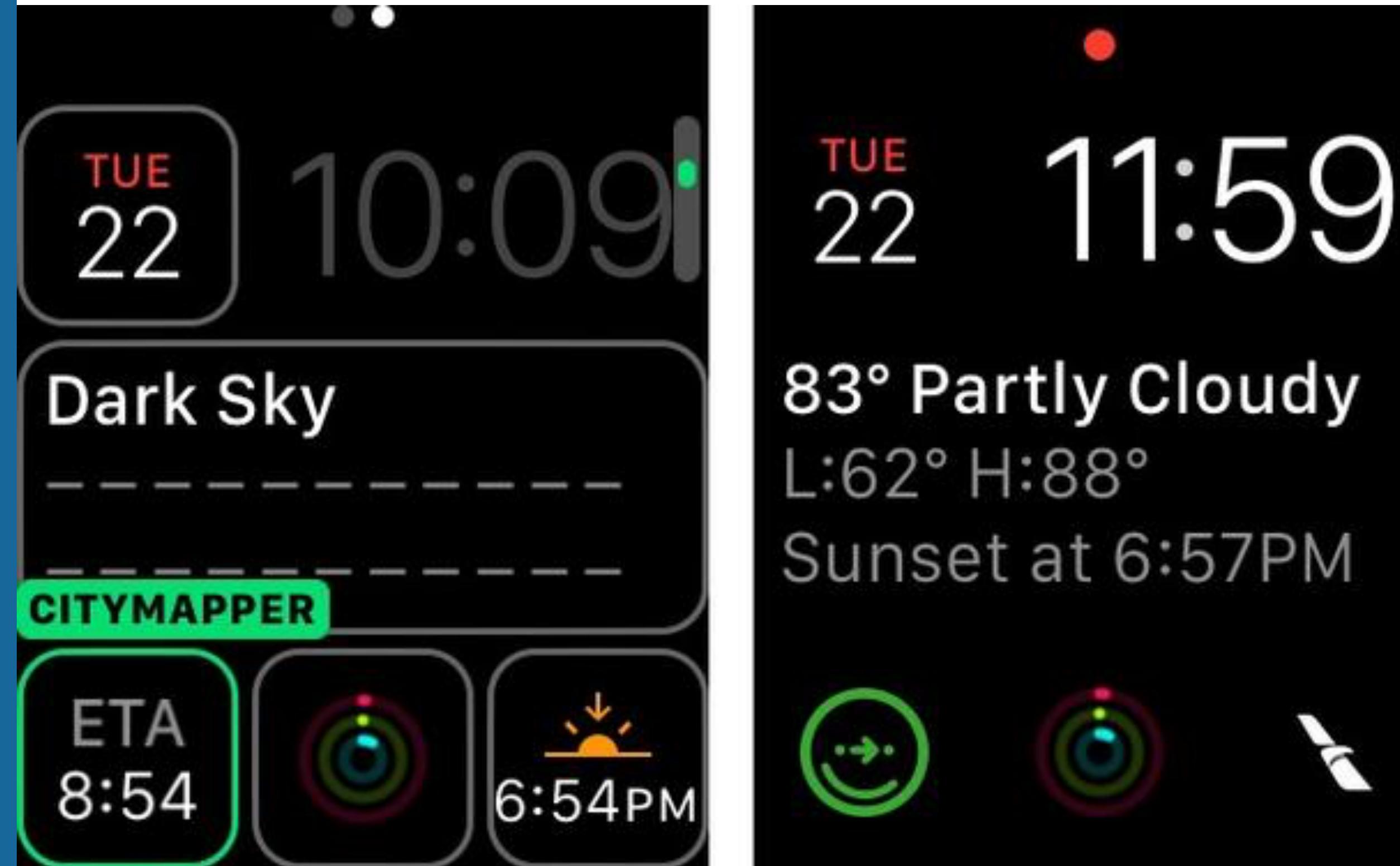


USER INTERACTIONS

COMPLICATIONS

SUBTITLE

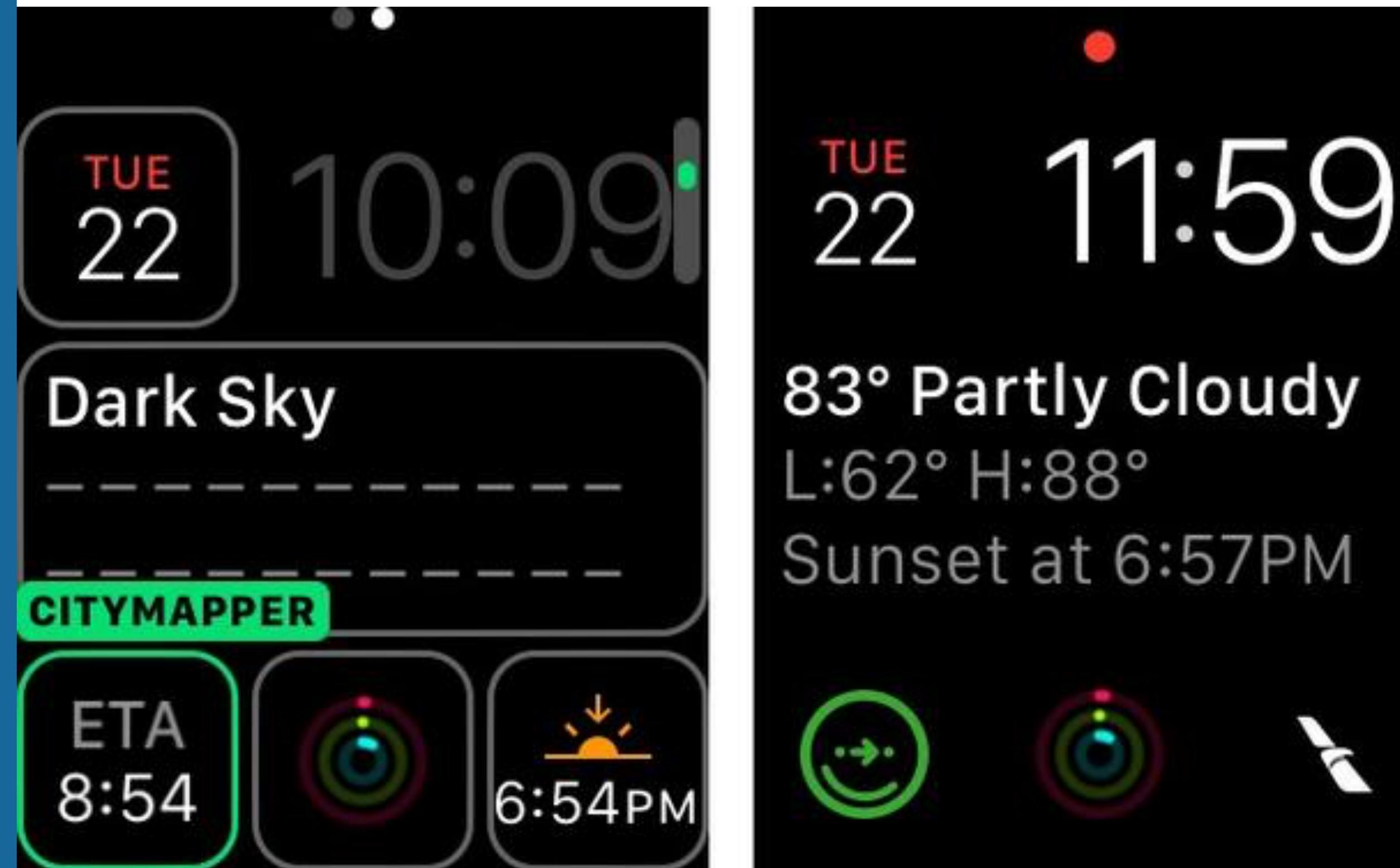
- Number of complications varies depending on the watch face
- At least 2



USER INTERACTIONS

COMPLICATIONS

- Apps whose complications are selected on the face get VIP treatment
 - Stays in memory; fast launch
 - Receives more time to execute background tasks
 - Receive background updates (at least 2x/hour)



USER INTERACTIONS

COMPLICATIONS

Apple recommends that all Watch apps include a complication, even if that complication only acts as a button to launch the app. For information about complications and how to implement them, see [Complication Essentials](#).



USER INTERFACES ON WATCHOS

USER INTERACTIONS



Glanceable



Actionable

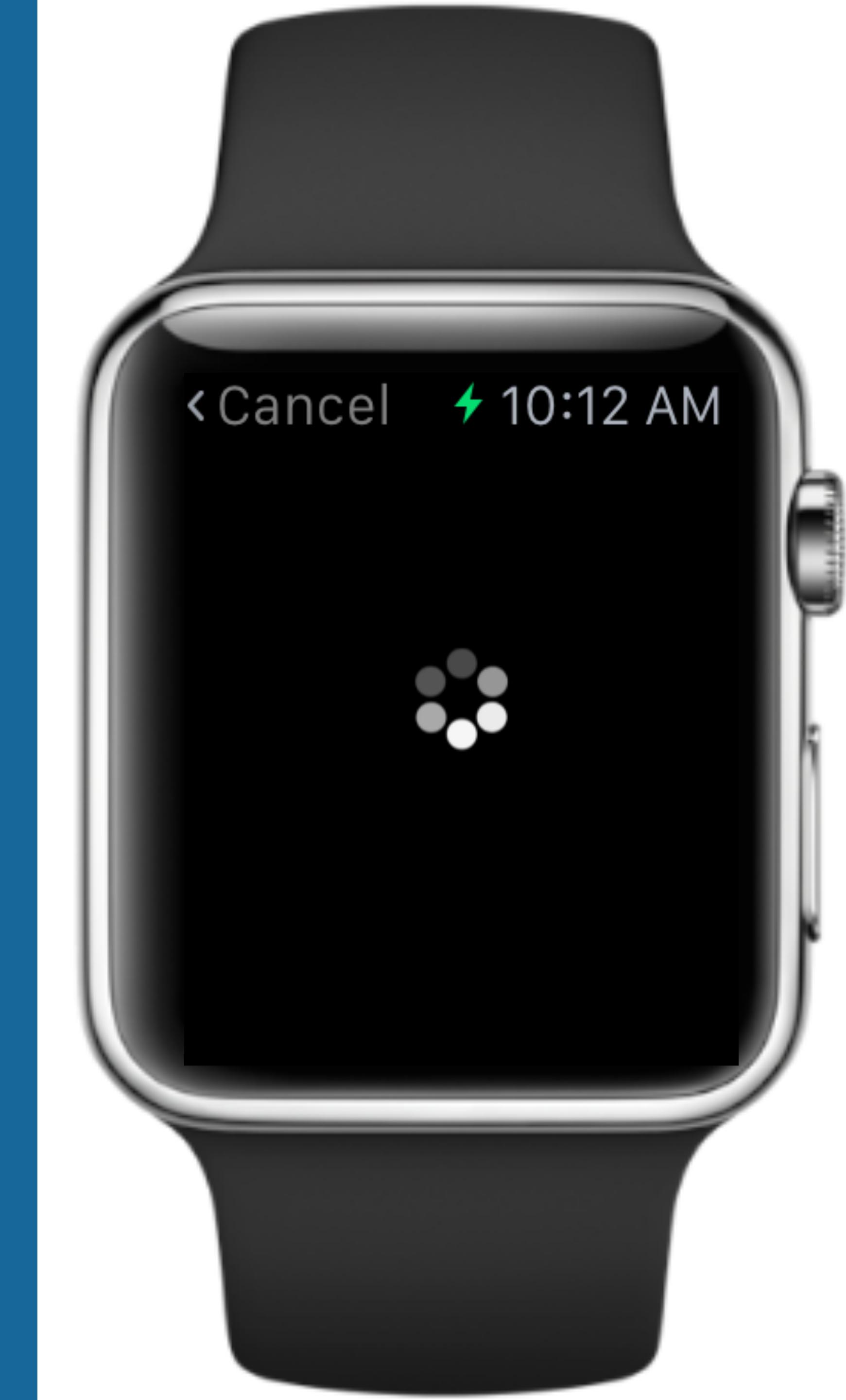


Responsive

- Design for quick interactions

USER INTERACTIONS

- Users will not be interested in seeing this interface in your application



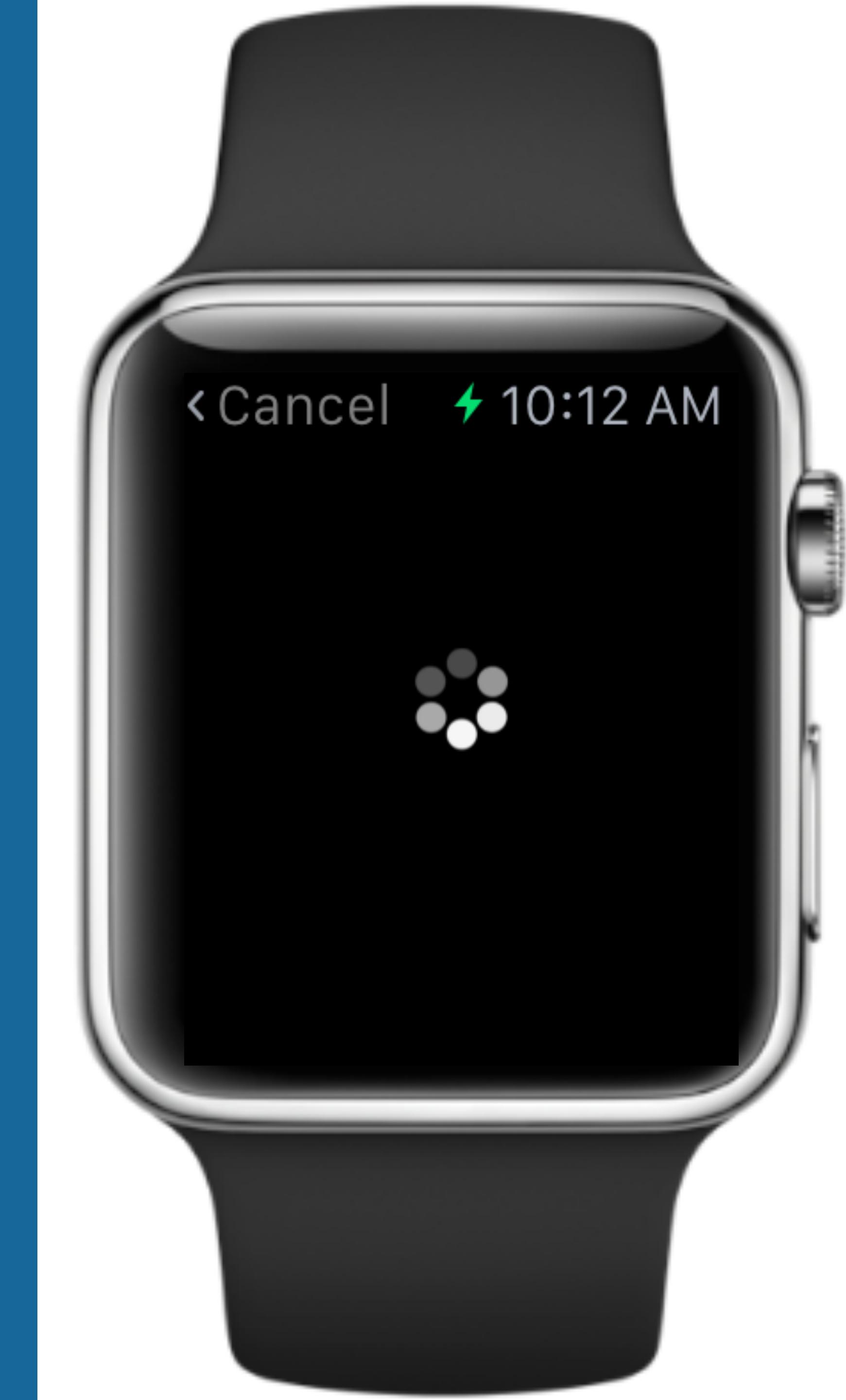
USER INTERACTIONS

- How long is a quick interaction?

. 2 seconds

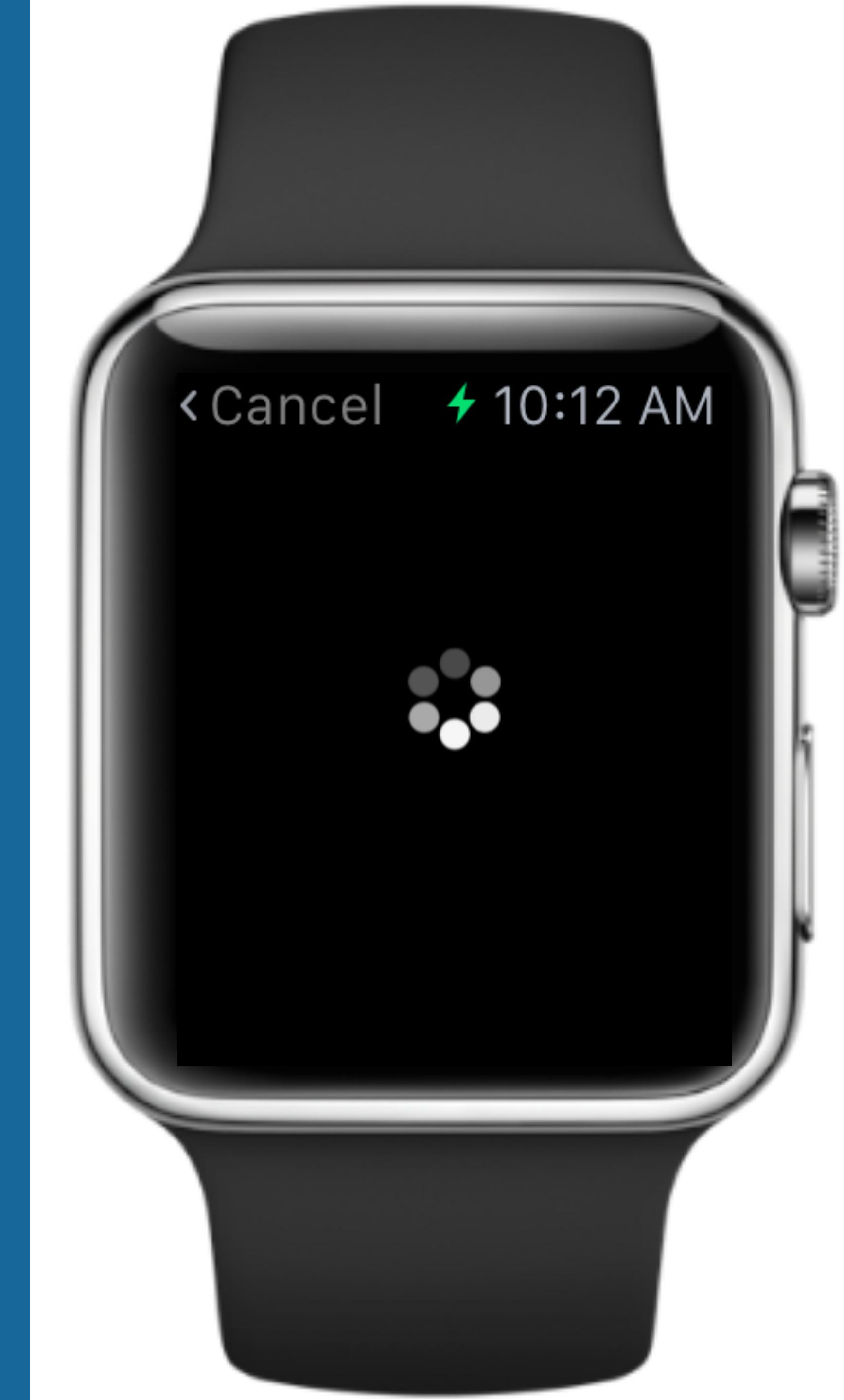
USER INTERACTIONS

- Design for quick and fast interactions
- Compliment your iOS app
- Be very conscious of load times



USER INTERACTIONS

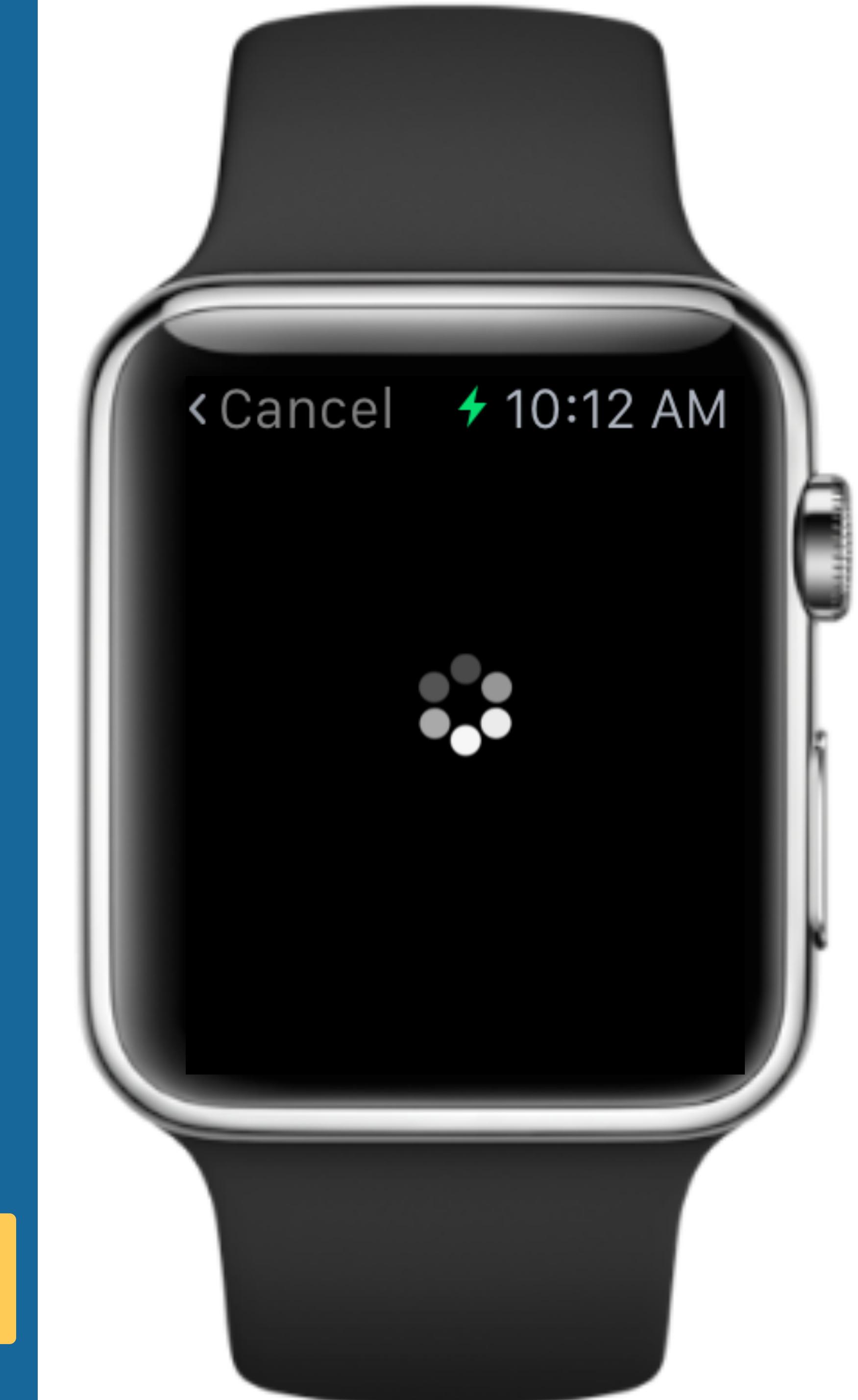
- watchOS3 additions to support quick interactions
 - Gesture recognizers
 - Digital crown rotation



USER INTERACTIONS

- watchOS additions to support displaying and updating information
 - Improved table navigation (vertical paging)
 - Support for new Notifications Framework
 - SceneKit and SpriteKit integration

YES, SPRITEKIT





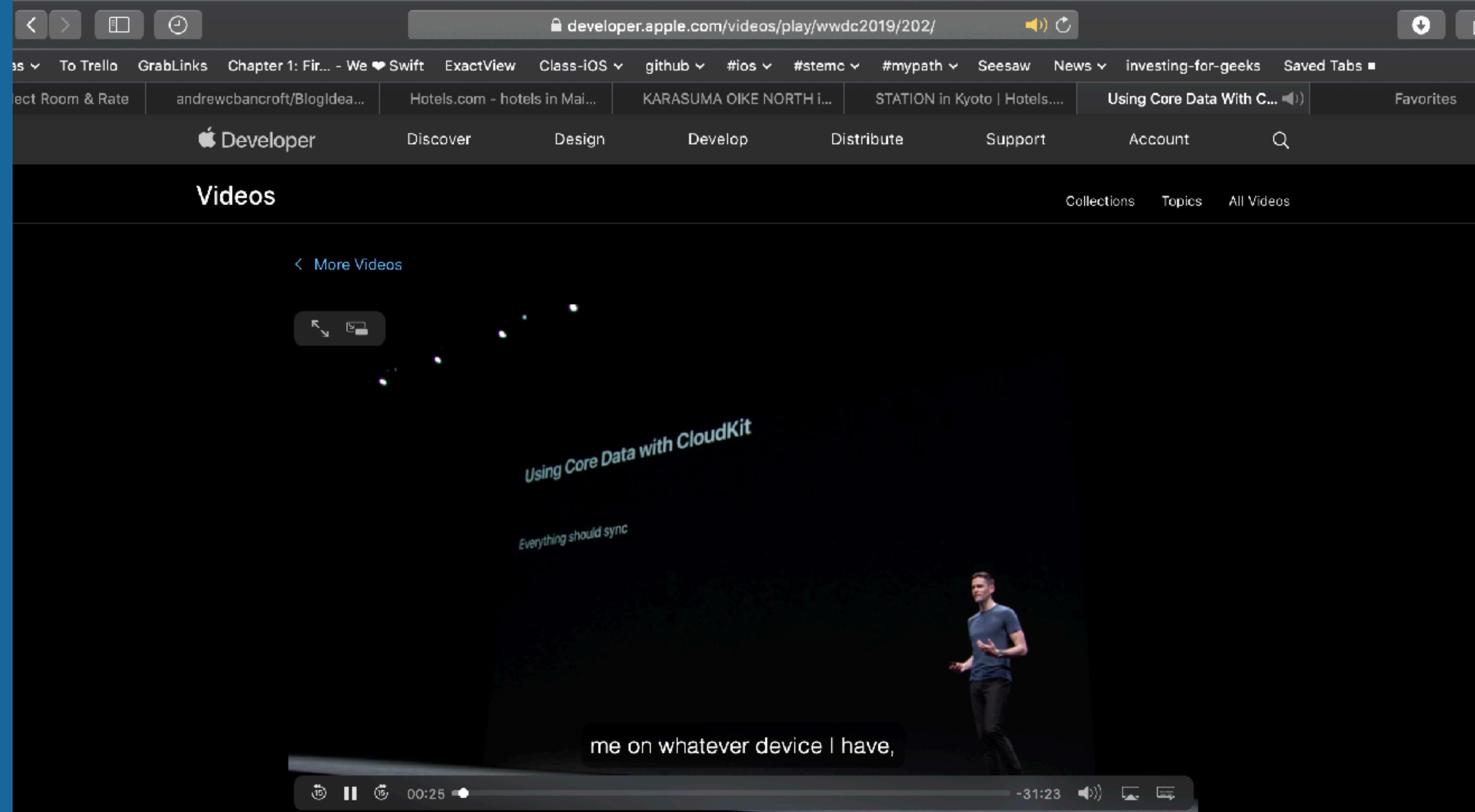
apple WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

BUT WAIT...

CLOUDKIT AND CORE DATA

- WWDC 2019 announcement



Overview Transcript

Using Core Data With CloudKit

CloudKit offers powerful, cloud-syncing technology while Core Data provides extensive data modeling and persistence APIs. Learn about combining these complementary technologies to easily build cloud-backed applications. See how new Core Data APIs make it easy to manage the flow of data through your application, as well as in and out of CloudKit. Join us to learn more about combining these frameworks to provide a great experience across all your customers' devices.

Resources

CLOUDKIT AND CORE DATA

- There are actually 2 ways to use CloudKit
 - Manually
 - Fully managed with Core Data Sync

etting Up Core Data with CloudKit

Set up the classes and capabilities that sync your store to CloudKit.

Framework
Core Data

Overview

To sync your Core Data store to CloudKit, you enable the CloudKit capability for your app. You also set up the Core Data stack with a persistent container that is capable of managing one or more local persistent stores that are backed by a CloudKit private database.

Configure a New Xcode Project

When you create a new project, you specify whether you want to add support for Core Data with CloudKit directly from the project setup interface. The resulting project instantiates an `PersistentCloudKitContainer` in your app's delegate. Once you enable CloudKit in your project, you use this container to manage one or more local stores that are backed with a CloudKit database.

1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point for your project and click Next.
3. Select the Use Core Data and Use CloudKit checkboxes.
4. Enter any other project details and click Next.
5. Specify a location for your project and click Create.

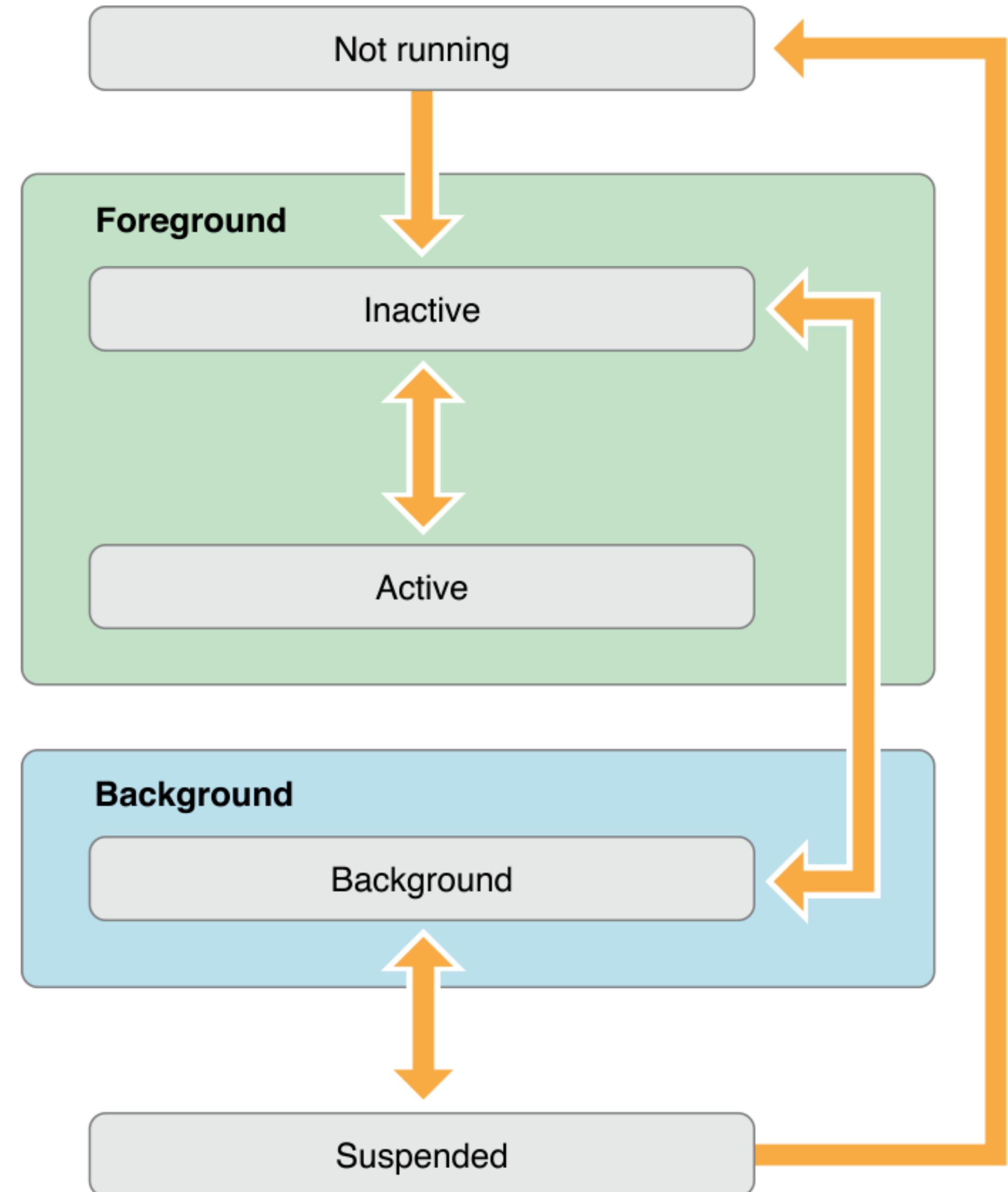
THIS METHOD IS MORE CORE DATA THAN

BACKGROUND TASKS

BACKGROUND TASKS

SUBTITLE

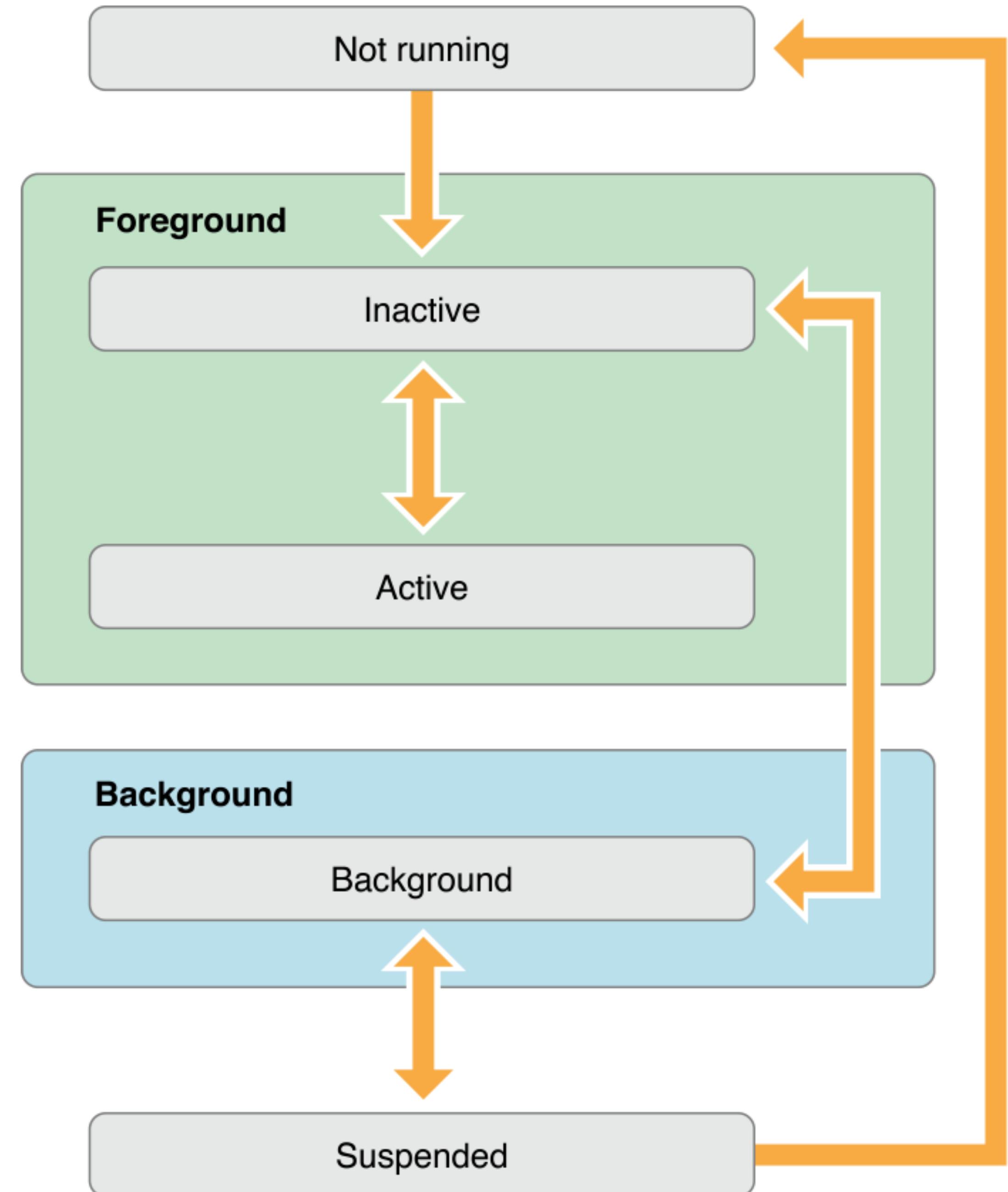
- State changes in iOS
- Background is a quick stop on way to suspended state
- Applications can be allotted additional time to run tasks in the background



BACKGROUND TASKS

SUBTITLE

- Legitimate background tasks
 - Track location for fitness app
 - Play music over lock screen
 - Download/update data so that it is ready for launch



BACKGROUND TASKS

- Three techniques on iOS
 - Finite-length tasks
 - Start a short task in the foreground and ask for time to finish that task when moved to background
 - URLSession background tasks
 - Initiate downloads in the foreground; hand off management of those downloads to the system
 - Background Execution modes
 - Support specific tasks can declare their support for one or more background execution modes

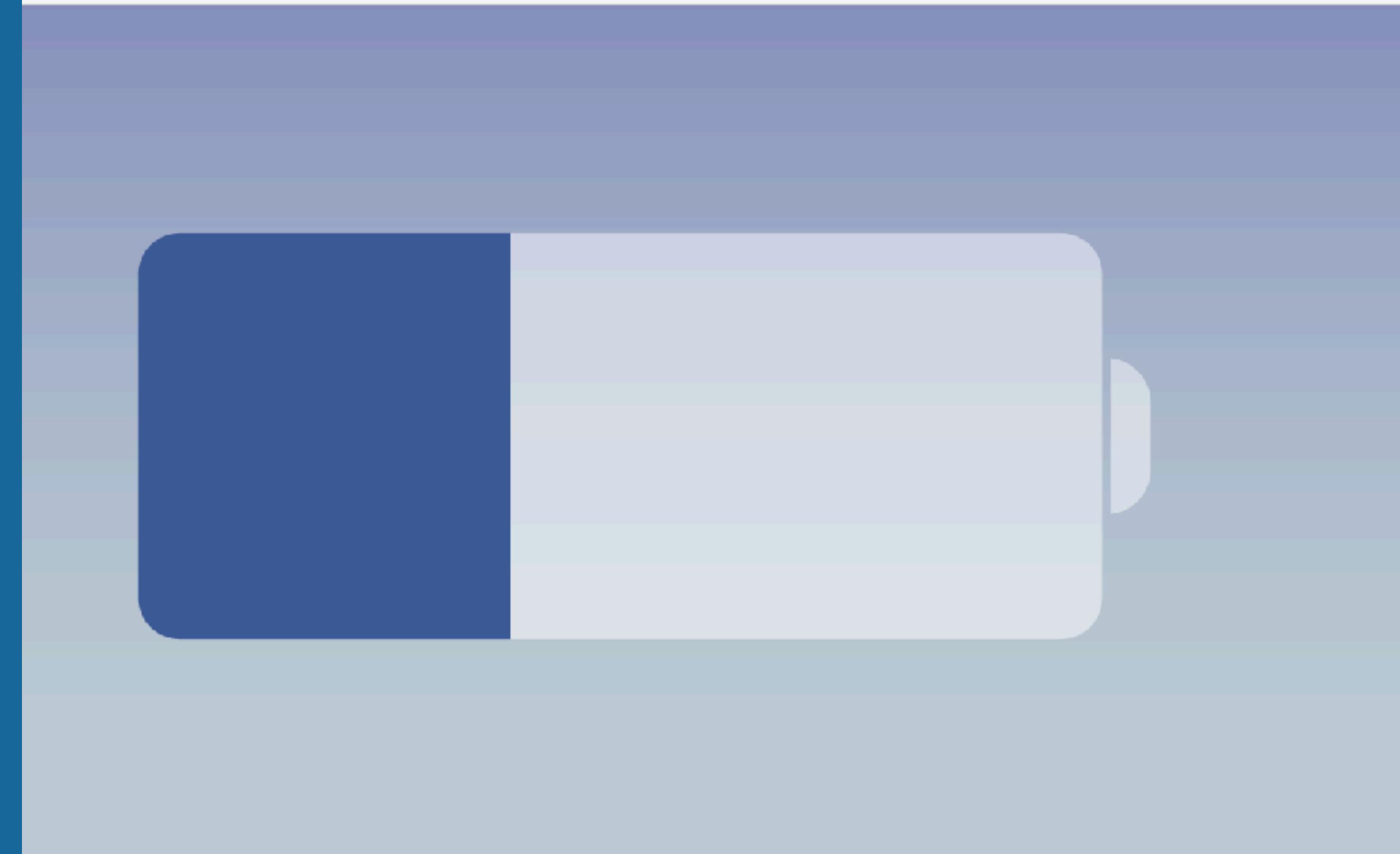
BACKGROUND TASKS

- Apple is strict about utility for background modes
- Some techniques have been abused, but don't count on it working for long

Facebook

ebook Says It Fixed A Bug That Caused Silent Audio Vampire Your iPhone Battery

Oct 22, 2015 by Matthew Panzarino (@panzer)



Facebook engineering manager Ari Grant has [posted an explanation](#) for the recent issues with gorging itself on power. Grant says that the battery issues were caused by a series of bugs, including one that ate up extra CPU cycles, causing Facebook to use up more of your iPhone's battery than it should.



VIVA TECHNOLOGY
JUNE 15-16-17 / PARIS 2017
30% OFF ON YOUR PASS

Crunchbase

Facebook

FOUNDED 2004

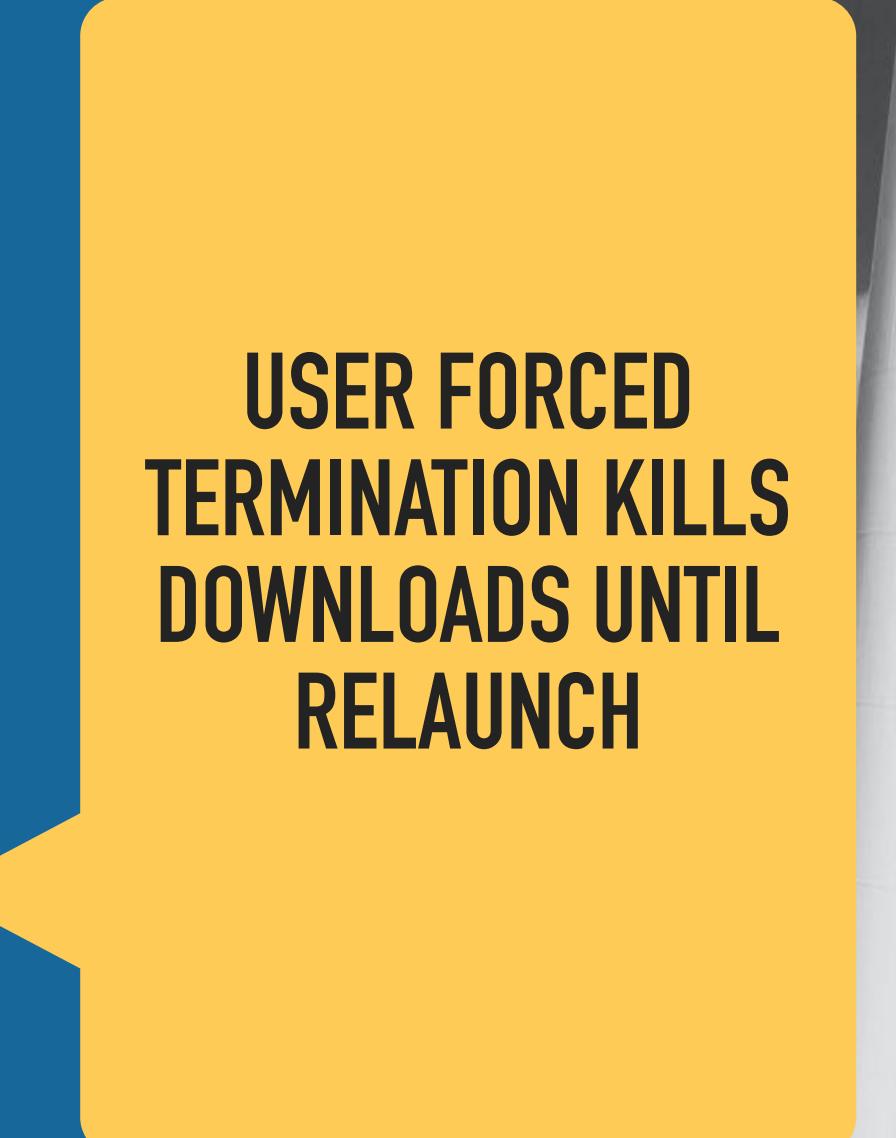
OVERVIEW

Facebook is an online social networking service that allows its users to connect with friends and family, as well as make new connections. It provides its users with a platform to share information, photos, and videos with their network.

BACKGROUND DOWNLOAD

BACKGROUND DOWNLOAD

- `NSURLSession` provides built-in support for background downloads
- System manages them through suspension or termination
 - Will restart app when download complete



USER FORCED
TERMINATION KILLS
DOWNLOADS UNTIL
RELAUNCH



BACKGROUND DOWNLOAD

- Setting up a background download
 - Create the configuration object using the `backgroundSessionConfigurationWithIdentifier:` method of `NSURLSessionConfiguration`
 - Set the value of the configuration object's `sessionSendsLaunchEvents` property to true
 - If your app starts transfers while it is in the foreground, it is recommend that you also set the `discretionary` property of the configuration object to YES
 - Configure any other properties of the configuration object as appropriate.
 - Use the configuration object to create your `NSURLSession` object

BACKGROUND DOWNLOAD

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var backgroundSessionCompletionHandler: (() -> Void)?

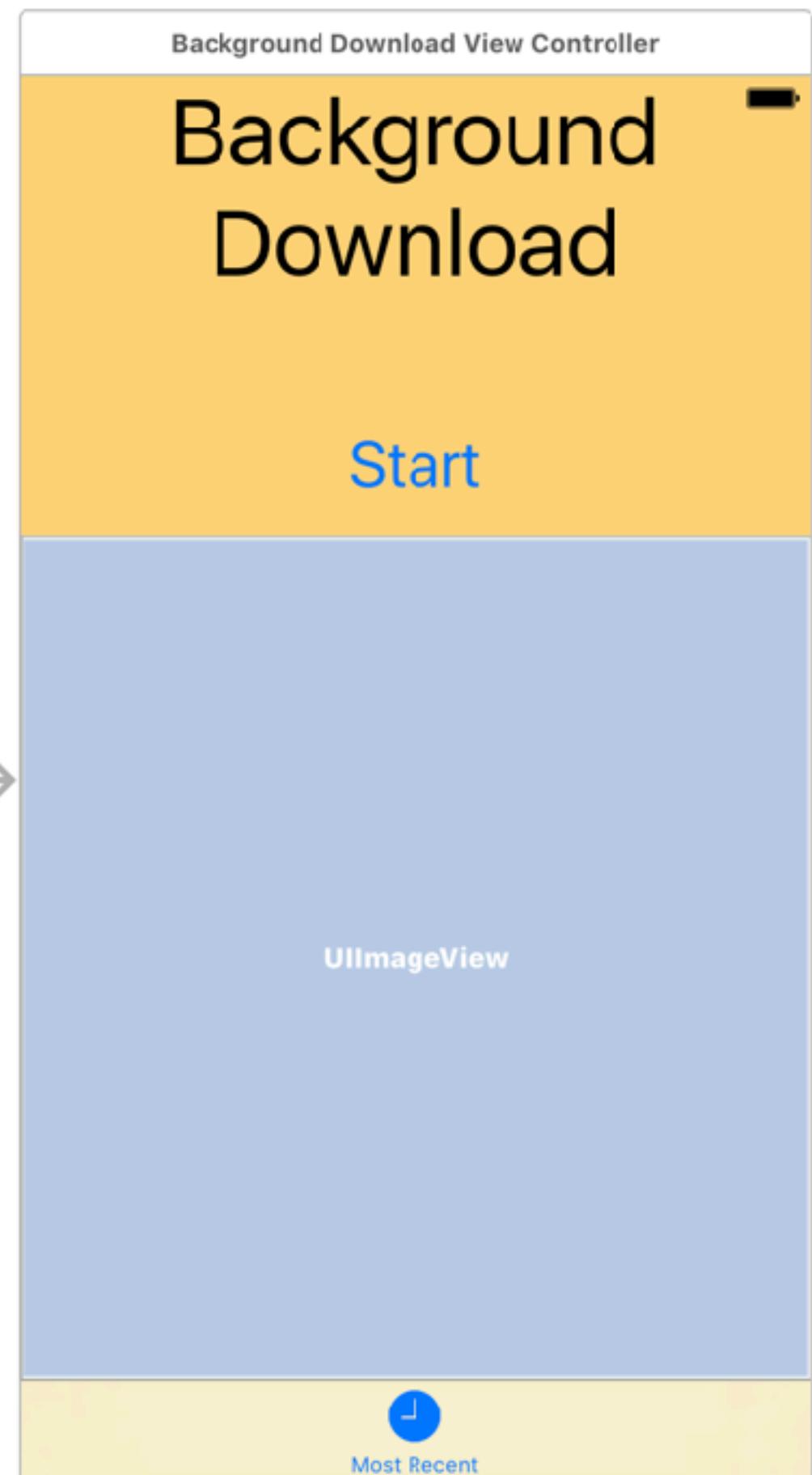
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool { ... }

    func application(_ application: UIApplication,
                    handleEventsForBackgroundURLSession identifier: String,
                    completionHandler: @escaping () -> Void) {
        print("handleEventsForBackgroundURLSession: \(identifier)")
        self.backgroundSessionCompletionHandler = completionHandler
    }
}
```

ASSIGN THE COMPLETION HANDLER
TO A PROPERTY TO KEEP IT ALIVE

BACKGROUND DOWNLOAD

```
//  
// BackgroundDownloadViewController.swift  
// WatchYourBack  
//  
// Created by T. Andrew Binkowski on 5/7/17.  
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.  
  
import UIKit  
  
class BackgroundDownloadViewController: UIViewController {  
  
    /// Start a download that will persist through moving to background  
    @IBAction func tapDownload(_ sender: Any) {  
        //  
    }  
  
    /// ImageView to show our downloaded image  
    @IBOutlet weak var imageView: UIImageView!  
}  
  
///  
///  
///  
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {  
  
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,  
                   didWriteData bytesWritten: Int64,  
                   totalBytesWritten: Int64,  
                   totalBytesExpectedToWrite: Int64) {  
        //  
    }  
  
    func urlSession(_ session: URLSession,  
                   task: URLSessionTask,  
                   didCompleteWithError error: Error?) {  
        //  
    }  
  
    func urlSession(_ session: URLSession,  
                   downloadTask: URLSessionDownloadTask,  
                   didFinishDownloadingTo location: URL){  
        //  
    }  
}
```



BACKGROUND DOWNLOAD

```
// Watchtower
//
// Created by T. Andrew Binkowski on 5/7/17.
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

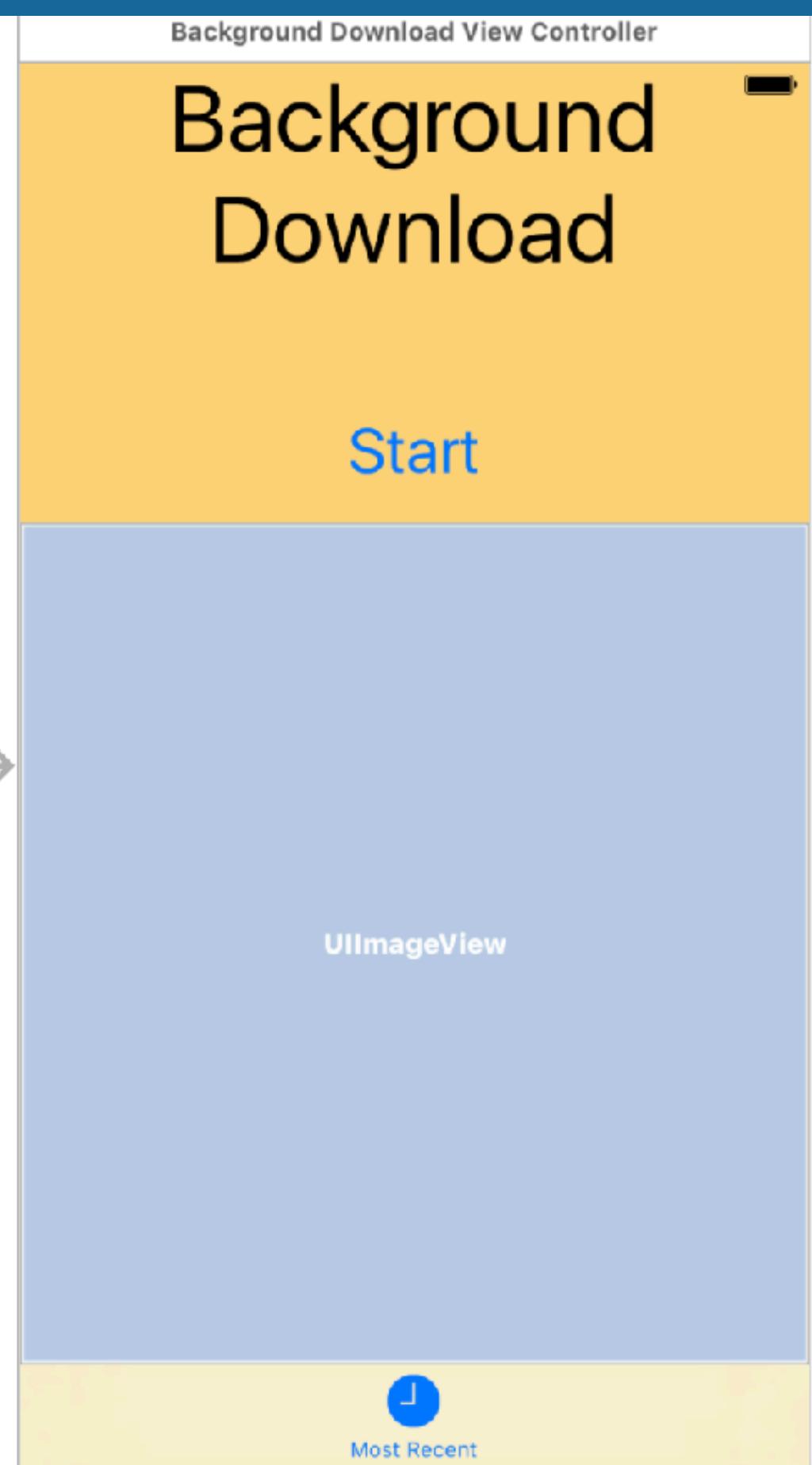
import UIKit

class BackgroundDownloadViewController: UIViewController {

    /// Start a download that will persist through moving to background
    @IBAction func tapDownload(_ sender: Any) {
        print("Tap download")
        let config = URLSessionConfiguration.background(withIdentifier: "mobi.uchicago.download")
        config.sessionSendsLaunchEvents = true
        let session = URLSession(configuration: config,
                               delegate: self,
                               delegateQueue: OperationQueue())
        let url = URL(string: "https://static.pexels.com/photos/104827/cat-pet-animal-domestic-104827.jpeg")
        let downloadTask = session.downloadTask(with: url!)
        downloadTask.resume()
    }

    /// ImageView to show our downloaded image
    @IBOutlet weak var imageView: UIImageView!
}

///
///
///
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {
```



BACKGROUND DOWNLOAD

```
import UIKit

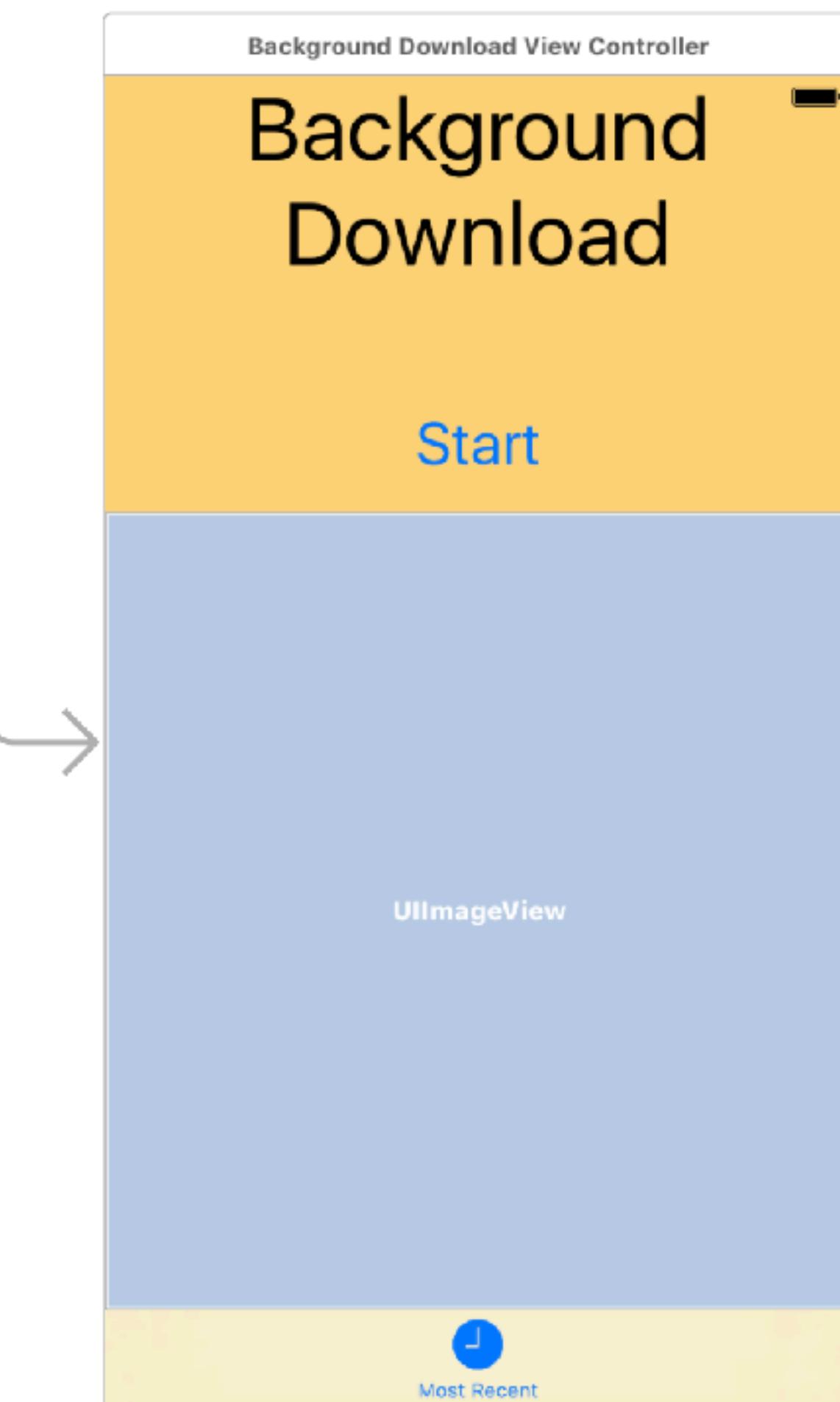
class BackgroundDownloadViewController: UIViewController {
    // Start a download that will persist through moving to background
    @IBAction func tapDownload(_ sender: Any) { ... }

    // ImageView to show our downloaded image
    @IBOutlet weak var imageView: UIImageView!
}

/// 
/// 
/// extension BackgroundDownloadViewController: URLSessionDownloadDelegate {

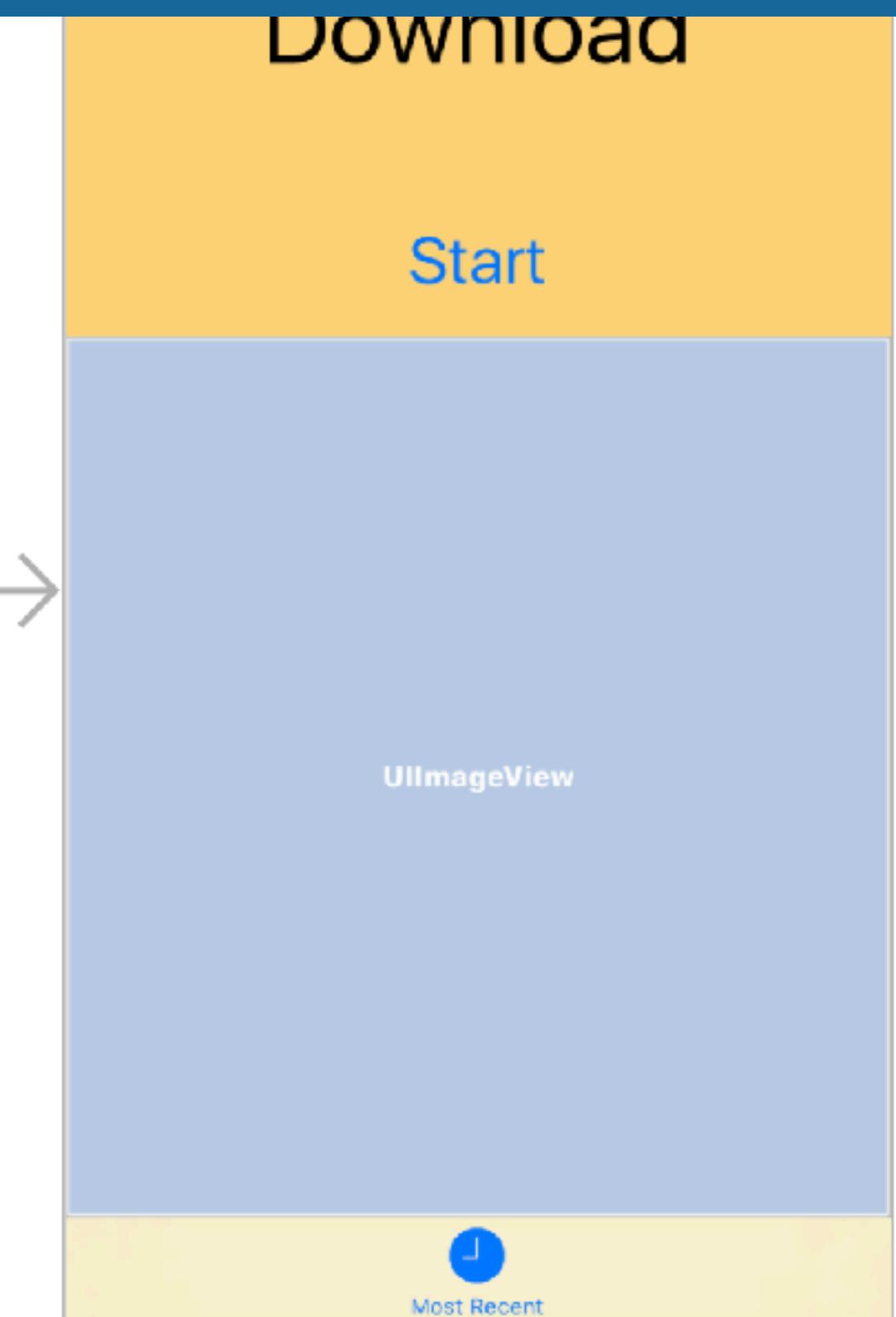
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,
                    didWriteData bytesWritten: Int64,
                    totalBytesWritten: Int64,
                    totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            print("Progress \(downloadTask) \(progress)")
        }
    }

    func urlSession(_ session: URLSession,
                    task: URLSessionTask,
                    didCompleteWithError error: Error?) { ... }
}
```



BACKGROUND DOWNLOAD

```
///  
func urlSession(_ session: URLSession,  
                 task: URLSessionTask,  
                 didCompleteWithError error: Error?) {  
    print("Task completed: \(task), error: \(String(describing: error))")  
}  
  
///  
func urlSession(_ session: URLSession,  
                 downloadTask: URLSessionDownloadTask,  
                 didFinishDownloadingTo location: URL){  
  
    print("Download finished: \(location)")  
  
    let path = NSSearchPathForDirectoriesInDomains(FileManager.SearchPathDirectory.documentDirectory,  
                                                FileManager.SearchPathDomainMask.userDomainMask,  
                                                true)[0]  
    let fileManager = FileManager()  
    let destinationURLOfFile = URL(fileURLWithPath: path.appendingFormat("/downloaded.jpg"))  
    try? fileManager.removeItem(at: destinationURLOfFile)  
  
    do {  
        try fileManager.moveItem(at: location, to: destinationURLOfFile)  
        let image = UIImage(contentsOfFile: path.appendingFormat("/downloaded.jpg"))  
        DispatchQueue.main.async {  
            self.imageView.image = image  
        }  
    } catch {  
        print("An error occurred while moving file to destination url")  
    }  
}
```



BACKGROUND DOWNLOAD

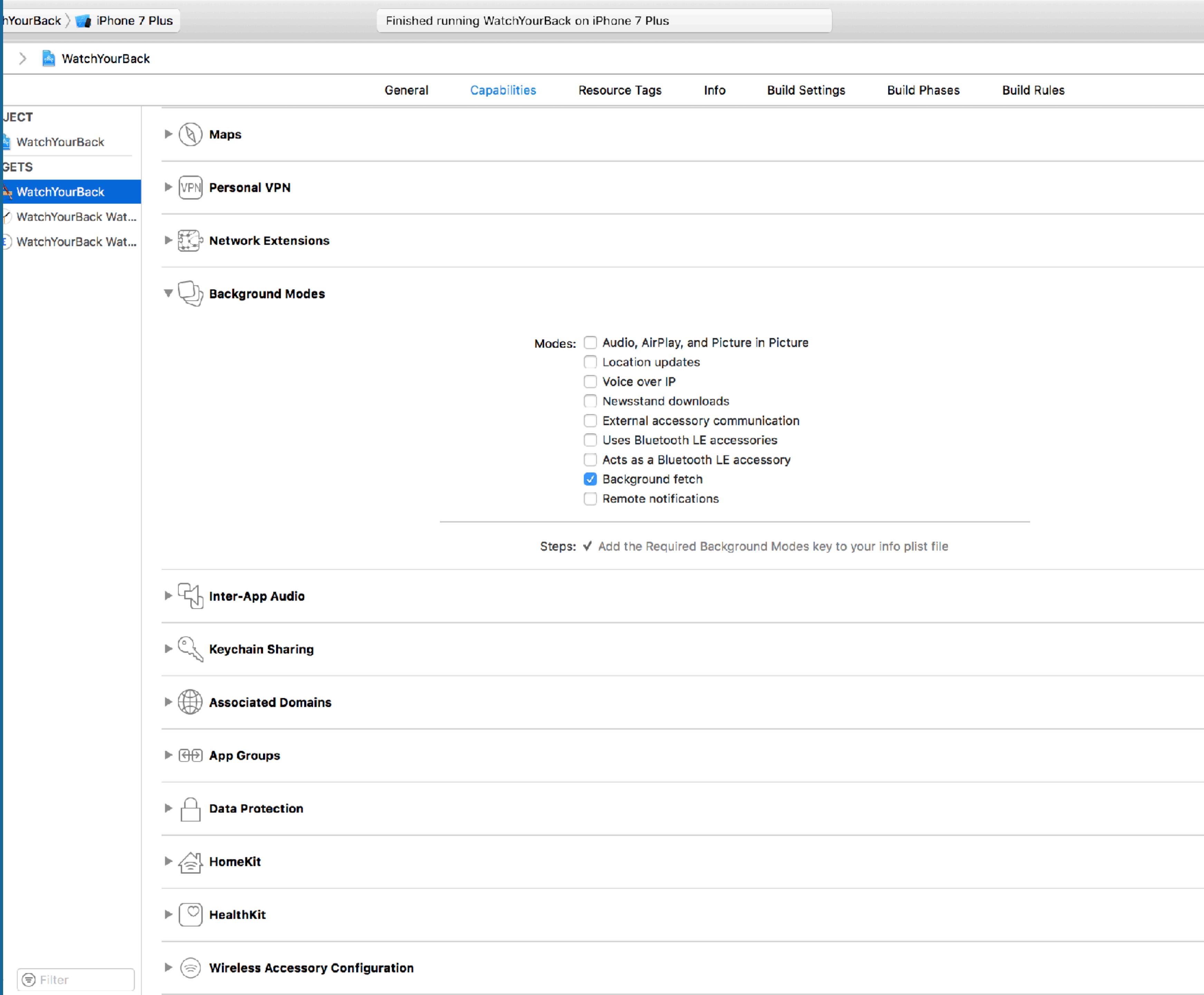
- Supported by URLSession on iOS and watchOS

BACKGROUND FETCH



BACKGROUND FETCH

- Special mode that will opportunistically fetch content
- Prevent "blinking data"
- ~ 30 seconds



BACKGROUND FETCH

- When a good opportunity arises, the system wakes or launches your app into the background and calls the app delegate's `application:performFetchWithCompletionHandler:` method
- Subject to OS scheduling (and punishment)
 - Following guidelines gives preferences to future runs

BACKGROUND FETCH

- Set background mode

The screenshot shows the Xcode interface with the project 'WatchYourBack' selected. The 'Capabilities' tab is active. On the left, there's a sidebar with sections like 'PROJECT', 'GETS', and 'WatchYourBack'. Under 'WatchYourBack', 'Background Modes' is expanded, showing a list of available modes: 'Maps', 'Personal VPN', 'Network Extensions', and 'Background Modes'. Below this, a list of checked modes is shown, with 'Background fetch' being the only one selected. A note at the bottom says 'Steps: ✓ Add the Required Background Modes key to your info.plist file'.

WatchYourBack > iPhone 7 Plus

Finished running WatchYourBack on iPhone 7 Plus

WatchYourBack

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT

WatchYourBack

GETS

WatchYourBack

WatchYourBack Wat...

WatchYourBack Wat...

Background Modes

Modes:

- Audio, AirPlay, and Picture in Picture
- Location updates
- Voice over IP
- Newsstand downloads
- External accessory communication
- Uses Bluetooth LE accessories
- Acts as a Bluetooth LE accessory
- Background fetch
- Remote notifications

Steps: ✓ Add the Required Background Modes key to your info.plist file

Inter-App Audio

Keychain Sharing

Associated Domains

App Groups

Data Protection

HomeKit

HealthKit

Wireless Accessory Configuration

Filter

BACKGROUND FETCH

```
on, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey :  
        backgroundFetchInterval(UIApplicationBackgroundFetchIntervalMinimum)  
        setFetchInterval(10)
```

- Set intervals

BACKGROUND FETCH

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var backgroundSessionCompletionHandler: (() -> Void)?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool { }

    // Support for background fetch
    func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {

        // A. Find the view controller we want in the hierarchy
        if let tabBarController = window?.rootViewController as? UITabBarController {
            if let fetchViewController = tabBarController.viewControllers?[0] as? BackgroundFetchViewController {
                // B. Execute function in fetch view controller
                fetchViewController.fetchData {
                    // C. In completion block, update the UI
                    print("In completion block")
                    fetchViewController.updateInterface()
                    // D. Tell OS we're done so that it can update the multitasking
                    // snapshot
                    completionHandler(.newData)
                }
            }
        }
    }
}
```

BACKGROUND FETCH

```
//  
//  ViewController.swift  
//  WatchYourBack  
//  
//  Created by T. Andrew Binkowski on 5/7/17.  
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.  
  
import UIKit  
  
class BackgroundFetchViewController: UIViewController {  
  
    var dateFormatter: DateFormatter = ...  
  
    @IBOutlet weak var timeLabel: UILabel!  
  
    var currentTime: Date? {  
        didSet {  
            if let currentTime = self.currentTime { ... }  
        }  
    }  
    //  
    // MARK: -  
    //  
    override func viewDidLoad() {...}  
  
    override func didReceiveMemoryWarning() {...}  
  
    func fetchData(_ completion: () -> Void) {...}  
    func updateInterface() {...}  
}
```

View controller that has functions called in background task

BACKGROUND FETCH

```
class BackgroundFetchViewController: UIViewController {  
  
    var dateFormatter: DateFormatter = ...  
  
    @IBOutlet weak var timeLabel: UILabel!  
  
    var currentTime: Date? {  
        didSet {  
            if let currentTime = self.currentTime {  
                self.timeLabel.text = dateFormatter.string(from: currentTime)  
            }  
        }  
    }  
  
    //  
    // MARK: -  
    //  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        self.updateInterface()  
    }  
  
    //  
    // MARK: - Fetching and Updating  
    //  
    func fetchData(_ completion: () -> Void) {  
        // Some long running task  
        for i in 0...100 {  
            print(i)  
        }  
        completion()  
    }  
  
    func updateInterface() {  
        print("UpdateUI")  
        self.currentTime = Date()  
    }  
}
```

Keep track of time and update the UI when changed

Long-running task

Update the label

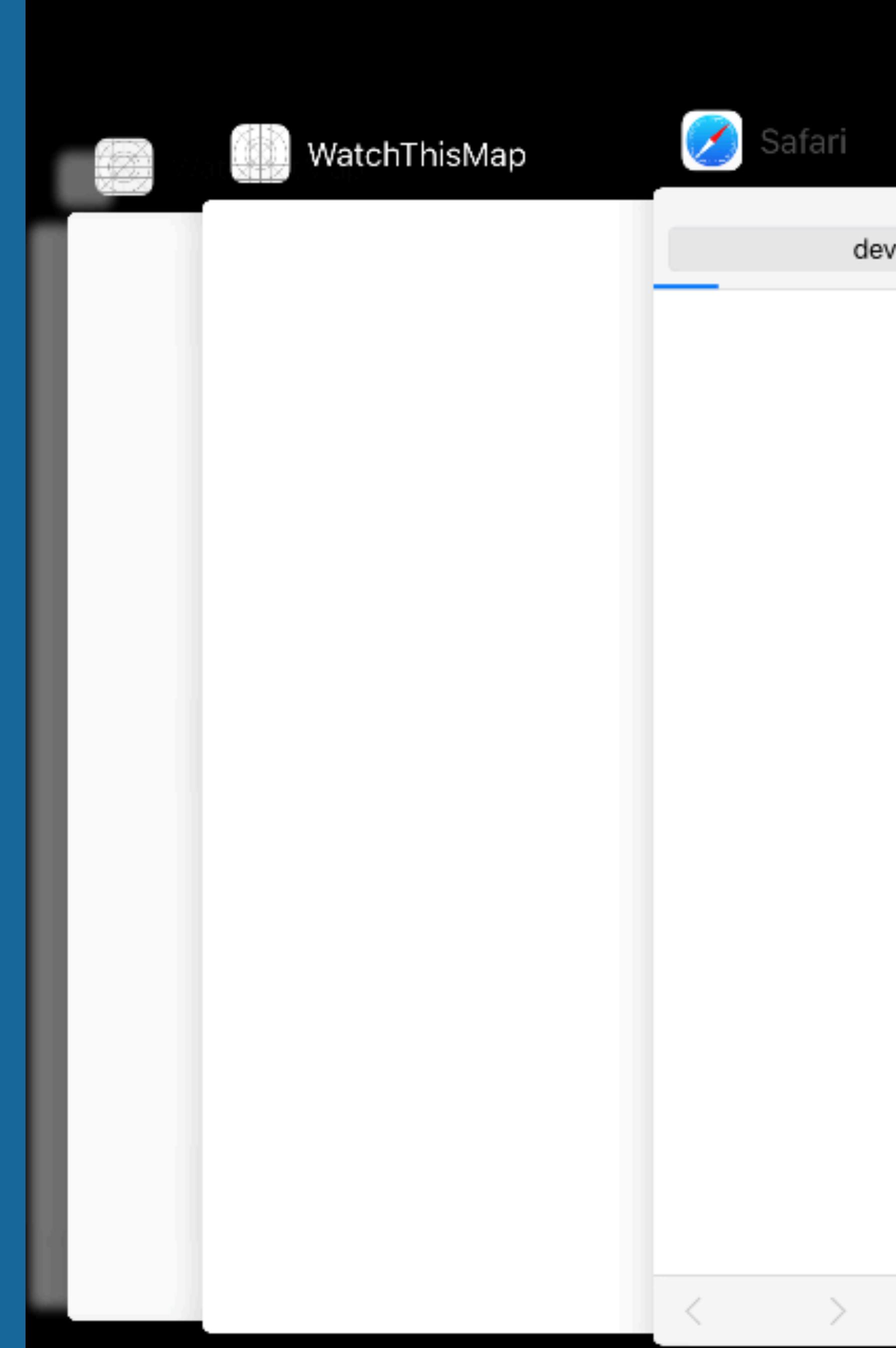
BACKGROUND FETCH

- Testing on device is unpredictable
- For once, testing on simulator is easier

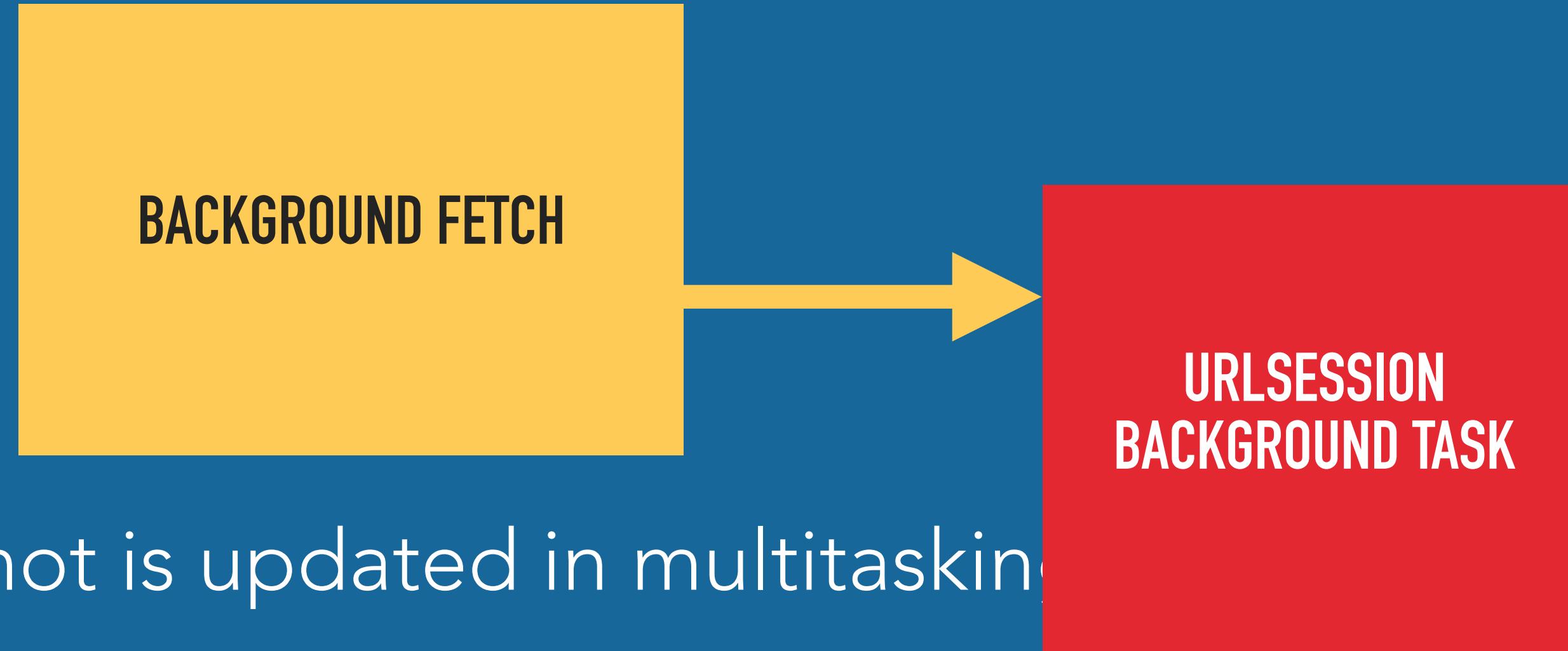


BACKGROUND FETCH

- Screenshot is updated in multitasking view



BACKGROUND FETCH



- Screenshot is updated in multitasking

FINITE-LENGTH TASKS

FINITE-LENGTH TASKS

- Execute a task in the background

```
import UIKit

class BackgroundTaskViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        longRunningTask()

    }

    func longRunningTask() {
        var bti : UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid
        bti = UIApplication.shared.beginBackgroundTask(expirationHandler:
            // Clean up after run (or denial)

            // Call the completion handler
            UIApplication.shared.endBackgroundTask(bti)
        )

        // Do work
        for i in 0...100000000 {
            print(i)
        }

        DispatchQueue.main.async {
            UIApplication.shared.endBackgroundTask(bti)
        }
    }
}
```

FINITE-LENGTH TASKS

```
super.viewDidLoad()
longRunningTask()
}

func longRunningTask() {
    var bti : UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid
    bti = UIApplication.shared.beginBackgroundTask(expirationHandler: {
        // Clean up after run (or denial)

        // Call the completion handler
        UIApplication.shared.endBackgroundTask(bti)
    })

    // Do work
    for i in 0...100000000 {
        print(i)
    }
}
```

The diagram illustrates the execution flow of a finite-length task. It starts with the call to `beginBackgroundTask`, which is represented by a yellow speech bubble containing the number "0". This leads to the "Work" step, shown in a yellow box with a callout pointing to the loop body. After the work is done, the "Tell OS you're done" step is shown in a yellow box with a callout pointing to the call to `endBackgroundTask`. Finally, the "What happens when finished" step is shown in a large yellow box with a callout pointing to the completion handler block.

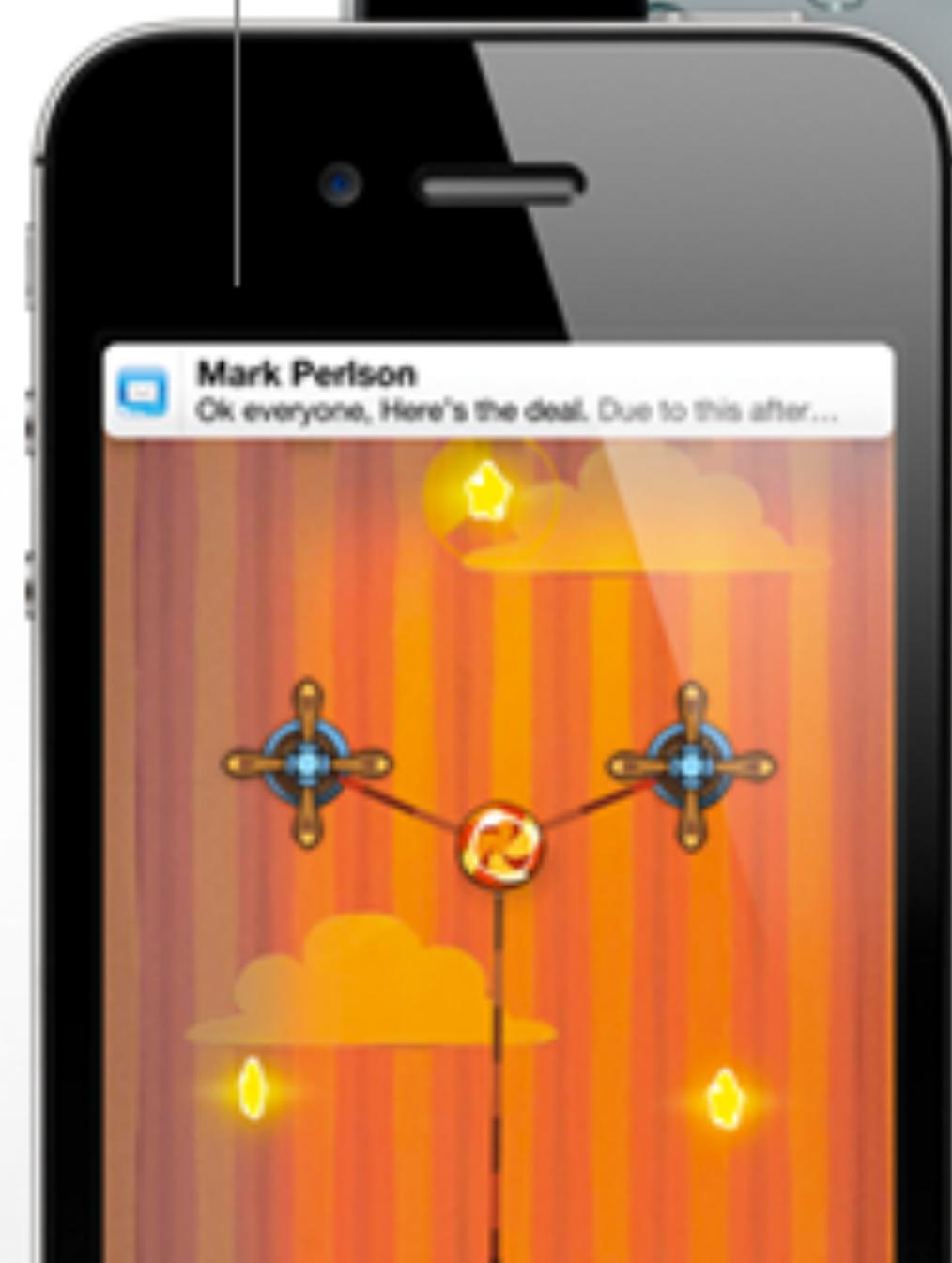
REVIEW

NOTIFICATIONS

NOTIFICATIONS

No more interruptions

Notifications appear at the top of your screen and disappear quickly.



One-swipe access

Swipe down to reveal Notification Center.



See more at a glance
View stocks and weather, too.

NOTIFICATIONS

No more interruptions

Notifications appear at the top of your screen and disappear quickly.

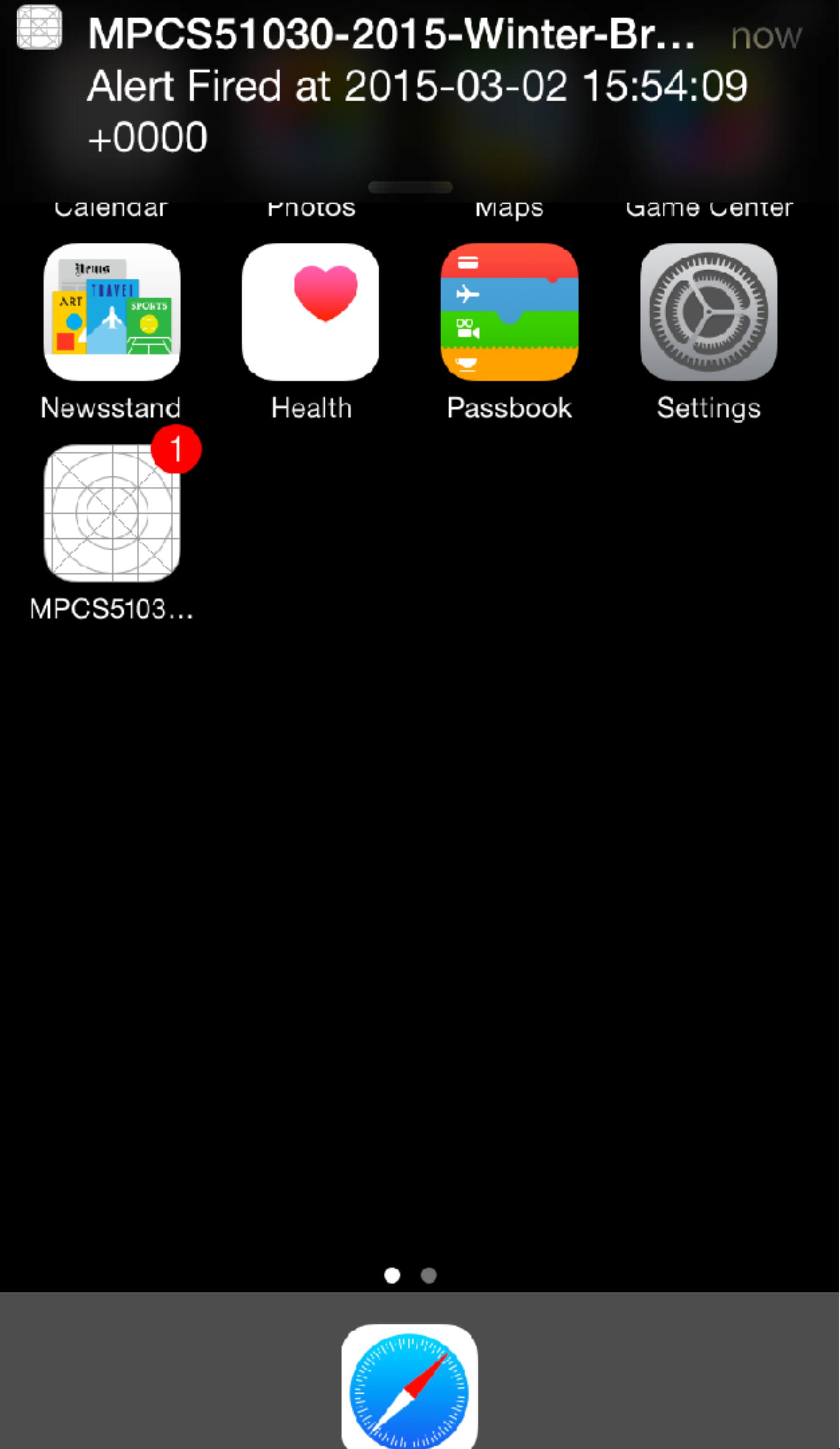


"User Notification"
not broadcast
notifications

See more
at a glance
View stocks and
weather, too.

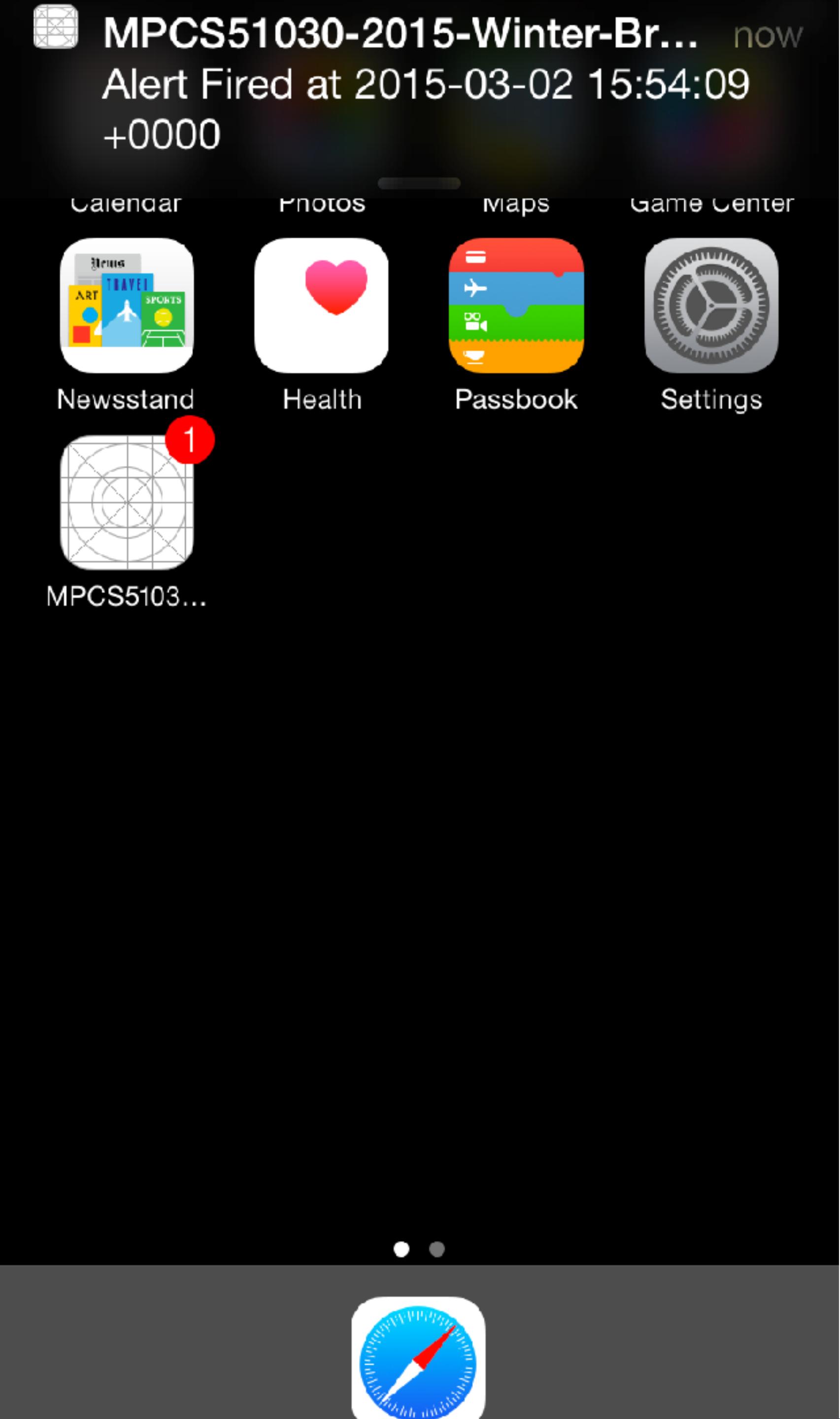
NOTIFICATIONS

- Only one application can run in the foreground
- Notifications give you flexibility in how to interact with users when they're not running your app
 - Messages
 - Game turn
 - Updates available



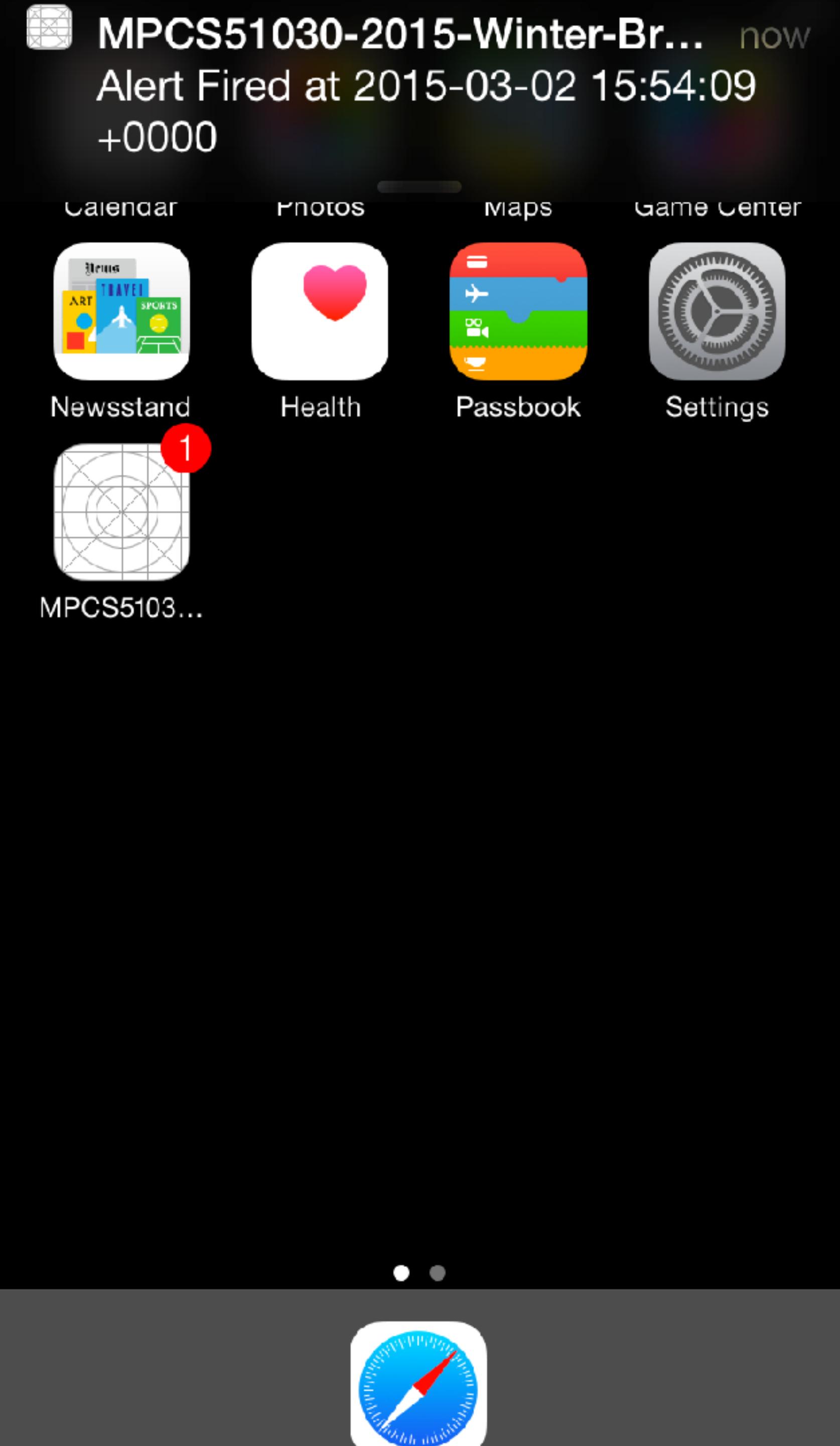
NOTIFICATIONS

- Re-engage your users
 - Notifications can launch your app



NOTIFICATIONS

- Notification styles
 - Modal
 - Alert
 - Sounds (default, custom)
 - Badges
 - Silent
 - Happens in background
 - “Trigger event”



NOTIFICATIONS

- Local Notifications
 - Scheduled by an application
 - Delivered by the iOS on the device
 - “Schedule a notification”

Carrier

9:56 AM

Back

Notifications

Show in Notification Center 5 >

Sounds



Badge App Icon



Show on Lock Screen

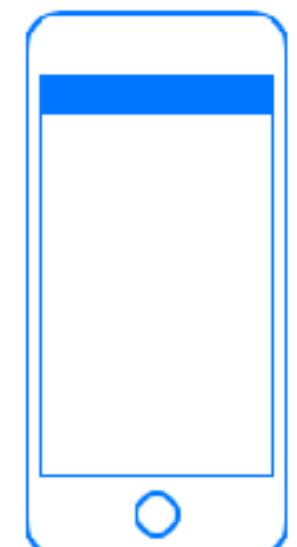


Show alerts on the lock screen, and in
Notification Center when it is accessed from
the lock screen.

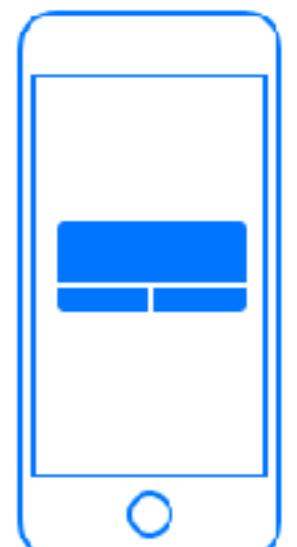
ALERT STYLE WHEN UNLOCKED



None



Banners



Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and
go away automatically.

NOTIFICATIONS

- Push Notifications
 - Remote notifications
 - Sent by application's remote server to Apple Push Notification Service (APNS)
 - "Register a notification"

Carrier

9:56 AM

Back

Notifications

Show in Notification Center 5 >

Sounds



Badge App Icon



Show on Lock Screen

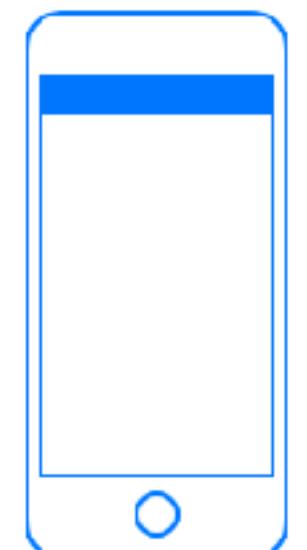


Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

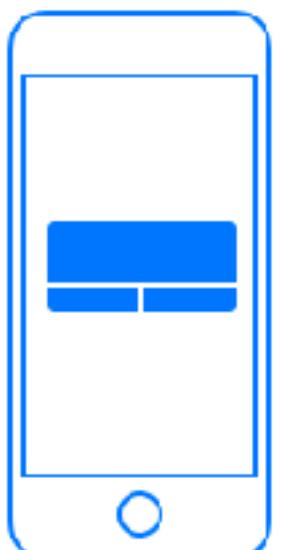
ALERT STYLE WHEN UNLOCKED



None



Banners



Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and go away automatically.

NOTIFICATIONS

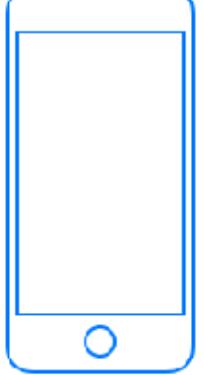
Sounds

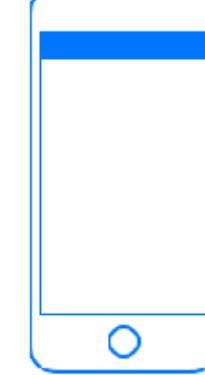
Badge App Icon

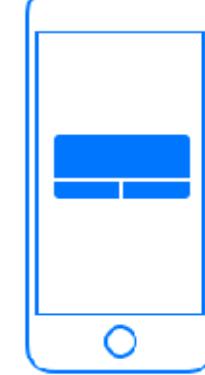
Show on Lock Screen

Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

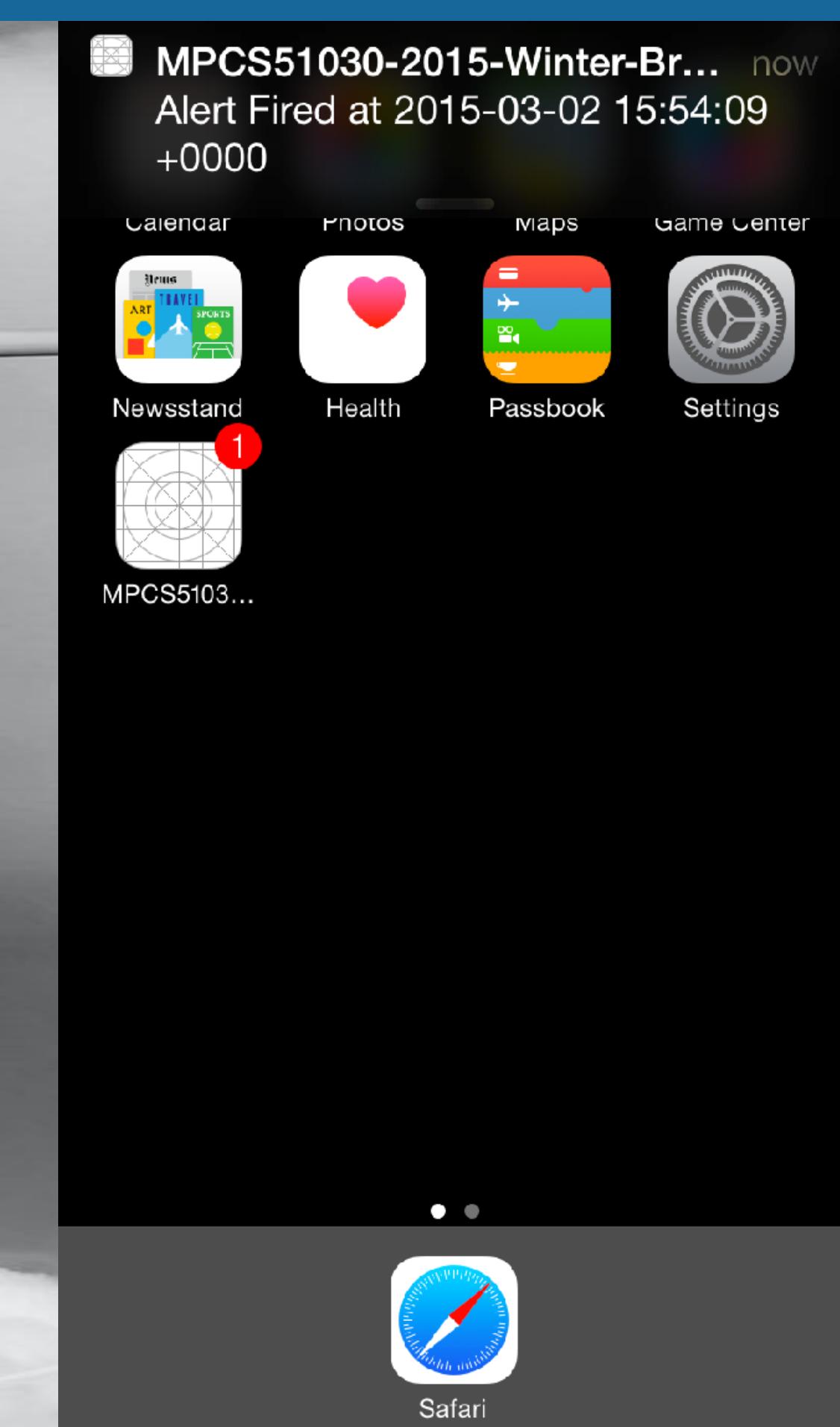
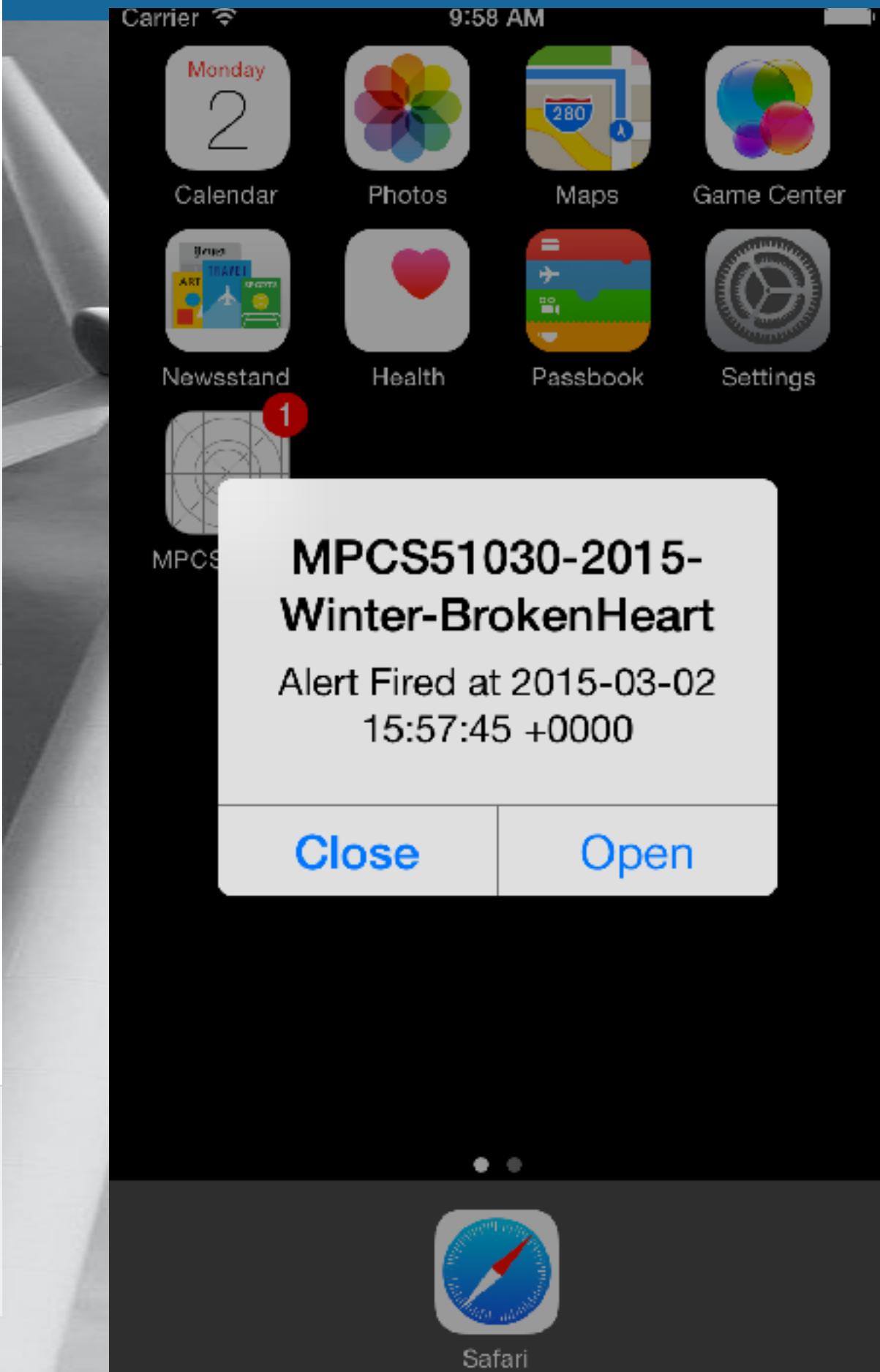
ALERT STYLE WHEN UNLOCKED

 None

 Banners

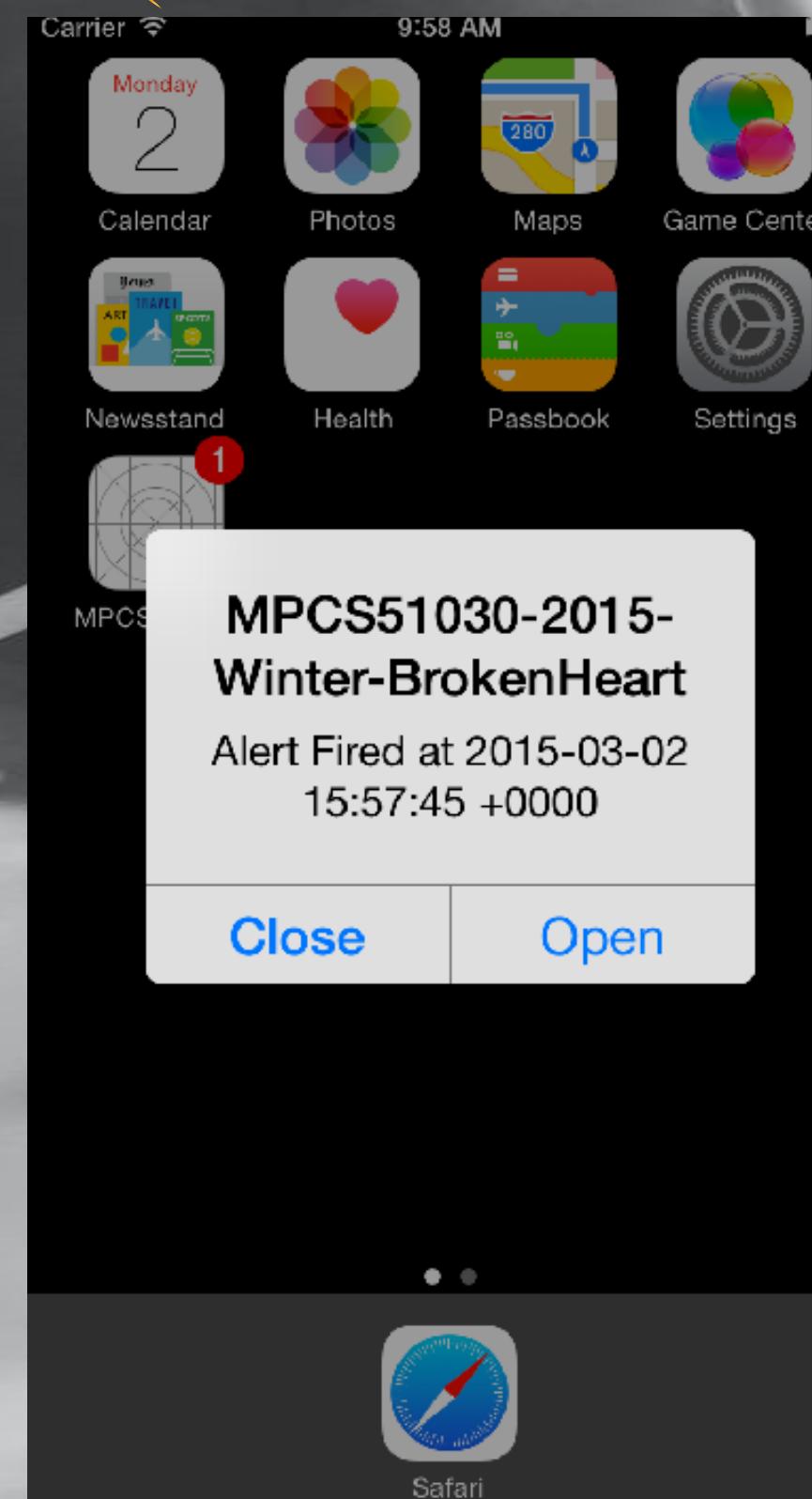
 Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and go away automatically.

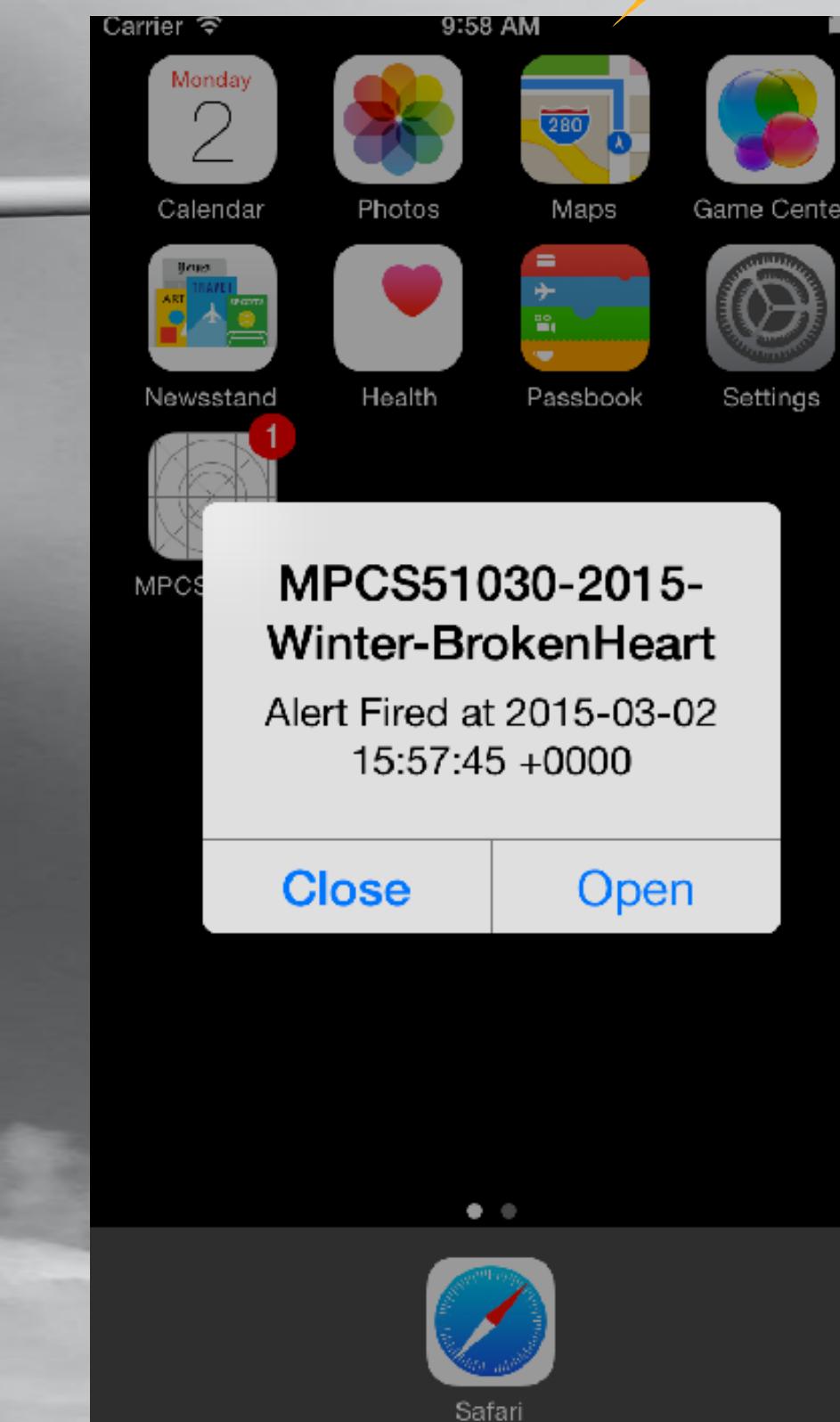


NOTIFICATIONS

Local



Push



Appears the same on devices

NOTIFICATIONS

- Handling notifications in app delegate
 - application:didFinishLaunchingWithOptions:
 - application:didReceiveRemoteNotification:
 - application:didReceiveLocalNotification:

Options: local or push

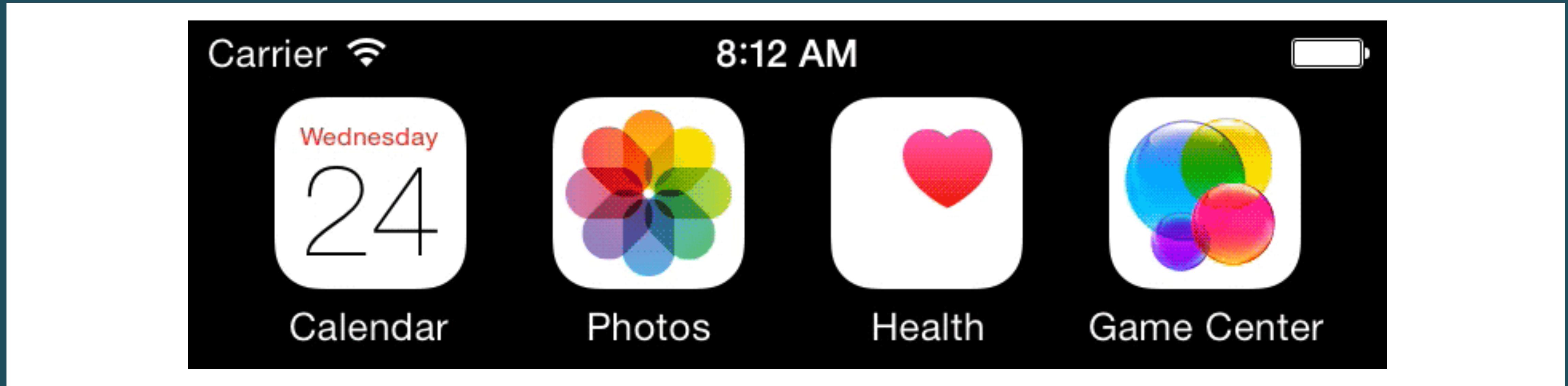
`UIApplicationLaunchOptionsLocalNotificationKey`

NOTIFICATIONS

- iOS8 introduced interactive notifications
 - Engage with application without opening it



NOTIFICATIONS



- iOS8 introduced interactive notifications
 - Engage with application without opening it
- iOS9 allows text input (eg. messages) from interactive notifications
- iOS10 unified the API for working with UserNotification Framework

PERMISSIONS

PERMISSIONS

- Remember that users are always in control
 - Keep them informed
 - Do not bother them
 - Of course you are trying to get their permission to bombard them with notifications)
 - Make sure your app still functions without notifications
 - Inform the user of consequences
 - “To be informed on the latest info, enable notifications in Settings” don’t forget to show a link to settings

Carrier

9:56 AM

Back

Notifications

Show in Notification Center 5 >

Sounds



Badge App Icon



Show on Lock Screen

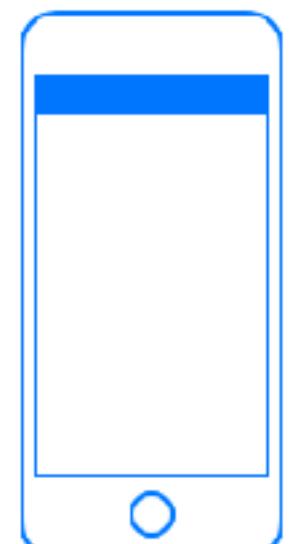


Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

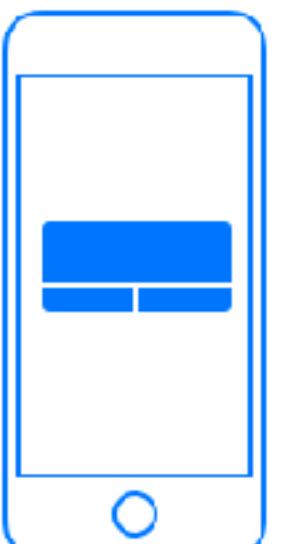
ALERT STYLE WHEN UNLOCKED



None



Banners



Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and go away automatically.

PERMISSIONS

```
let options: UNAuthorizationOptions = [.alert, .sound];
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: options) {
    (granted, error) in
    if !granted {
        print("Something went wrong")
    }
}
```

- Registering for notifications

PERMISSIONS

"HeyYou" Would Like to Send You Notifications

Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.

Standard Dialogue

Don't Allow

OK

PERMISSIONS

- Consideration on notifications
 - Users may not accept to receive notifications
 - Users can change preferences in settings
- Be prepared to handle all potential use cases

```
center.getNotificationSettings { (settings) in
    if settings.authorizationStatus != .authorized {
        // Notifications not allowed
    }
}
```

PERMISSIONS

- The old way
 - “Go to Settings > Privacy > Location > OUR_APP”



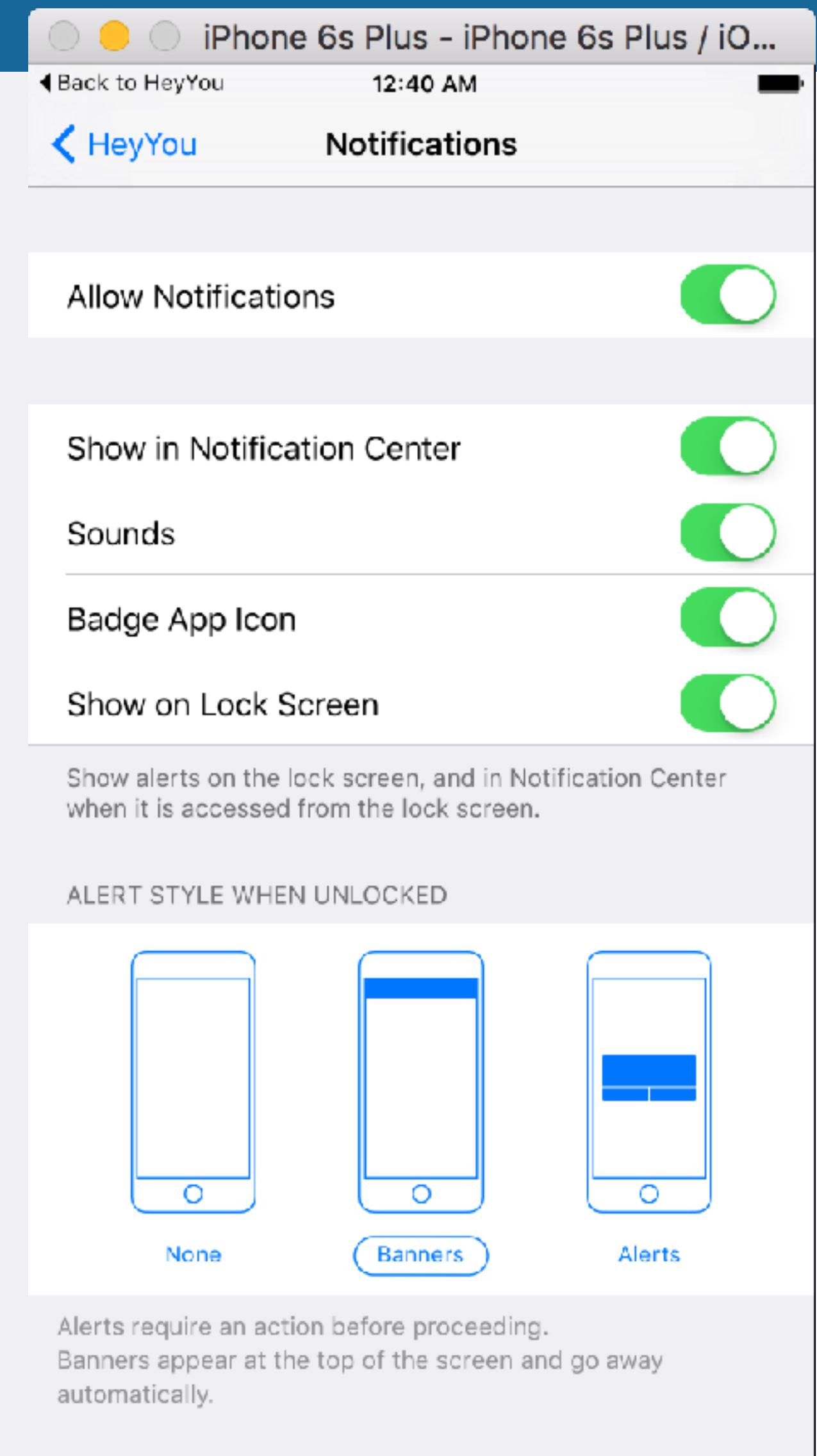
VERIZON 4G LTE 8:20 AM

< Settings Swarm

ALLOW SWARM TO ACCESS

	Location	While Using >
	Contacts	<input type="checkbox"/>
	Photos	<input checked="" type="checkbox"/>
	Camera	<input checked="" type="checkbox"/>
	Notifications >	
	Background App Refresh	<input checked="" type="checkbox"/>
	Use Cellular Data	<input checked="" type="checkbox"/>

PERMISSIONS



PERMISSIONS

```
e have authorization to send notifications
settings = UIApplication.sharedApplication().currentUserNotificationSettings() else { return }

t, let the user know and give them the option of going directly
to the settings screen for the application
types == .None {

n alert view controller
controller = UIAlertController(title: "⚠",
                                message: "The notification permission was not authorized. Please enable it in Settings to continue.",
                                preferredStyle: .Alert)

o go to settings
settingsAction = UIAlertAction(title: "Settings", style: .Default) { (alertAction) in
    appSettings = NSURL(string: UIApplicationOpenSettingsURLString) {
        UIApplication.sharedApplication().openURL(appSettings)
    }
}

ller.addAction(settingsAction)

cancel action that does nothing
cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)
ller.addAction(cancelAction)

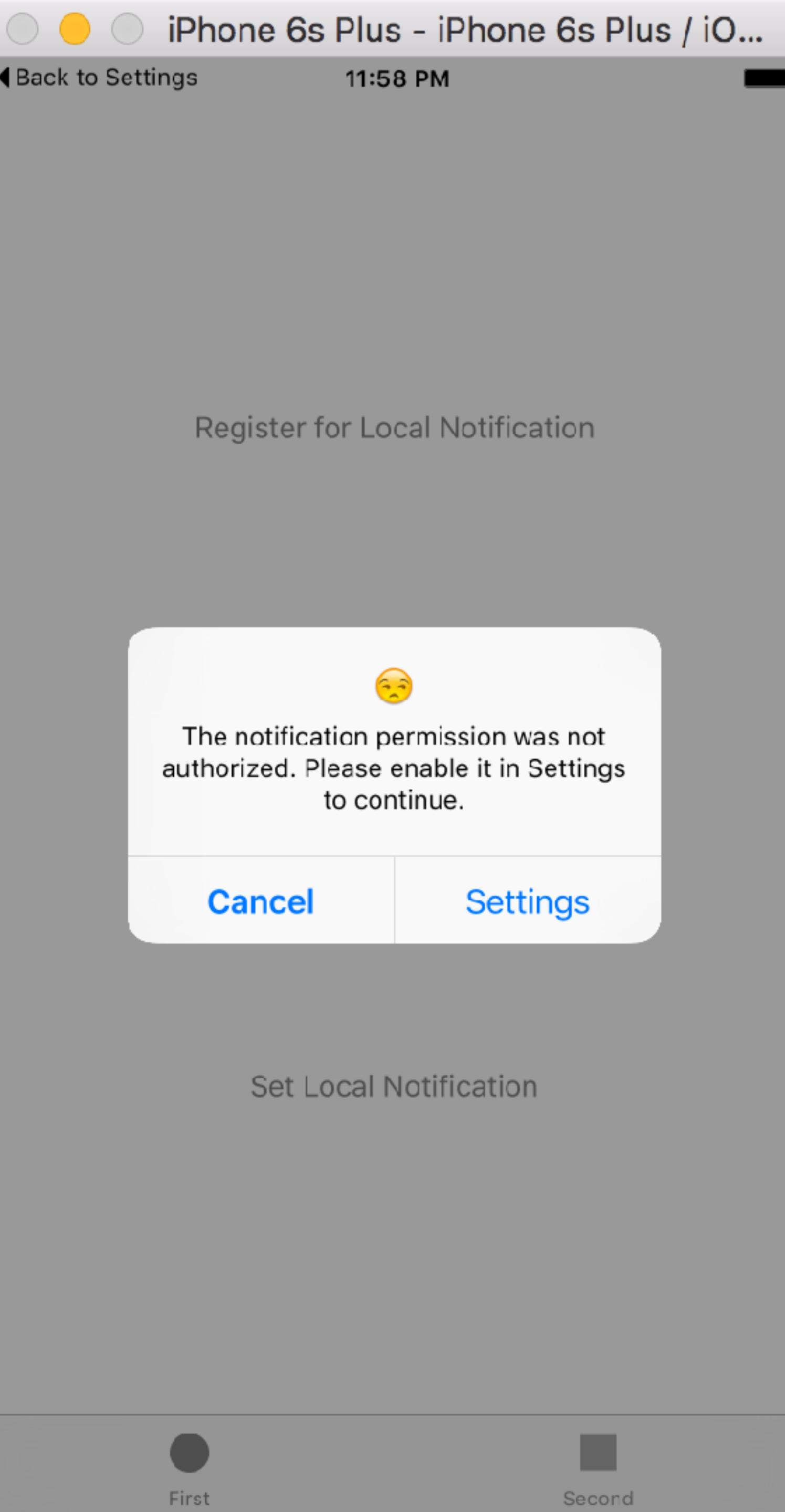
alert and exit early
dismissViewControllerAnimated(alertController, animated: true, completion: nil)
```



New
Way

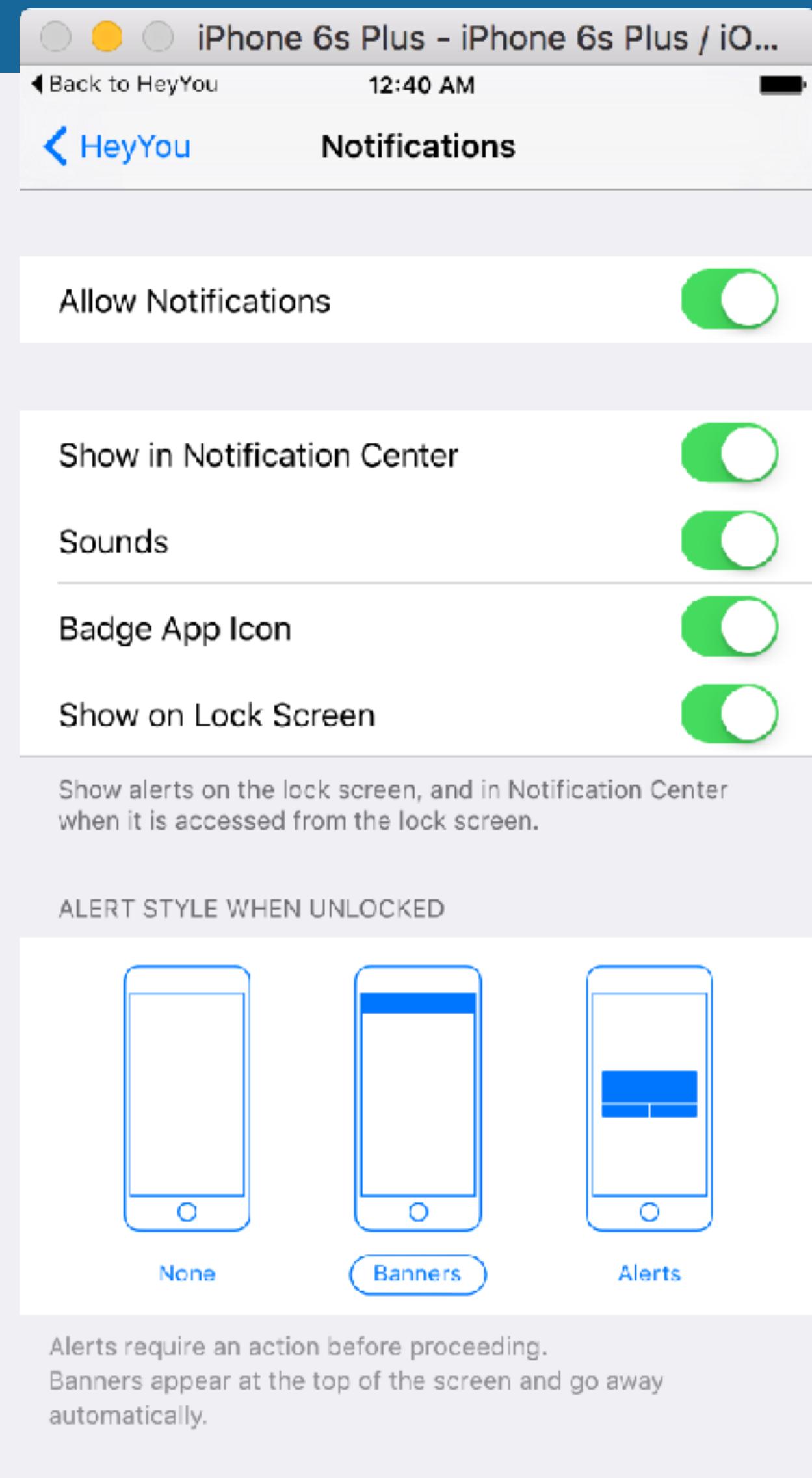
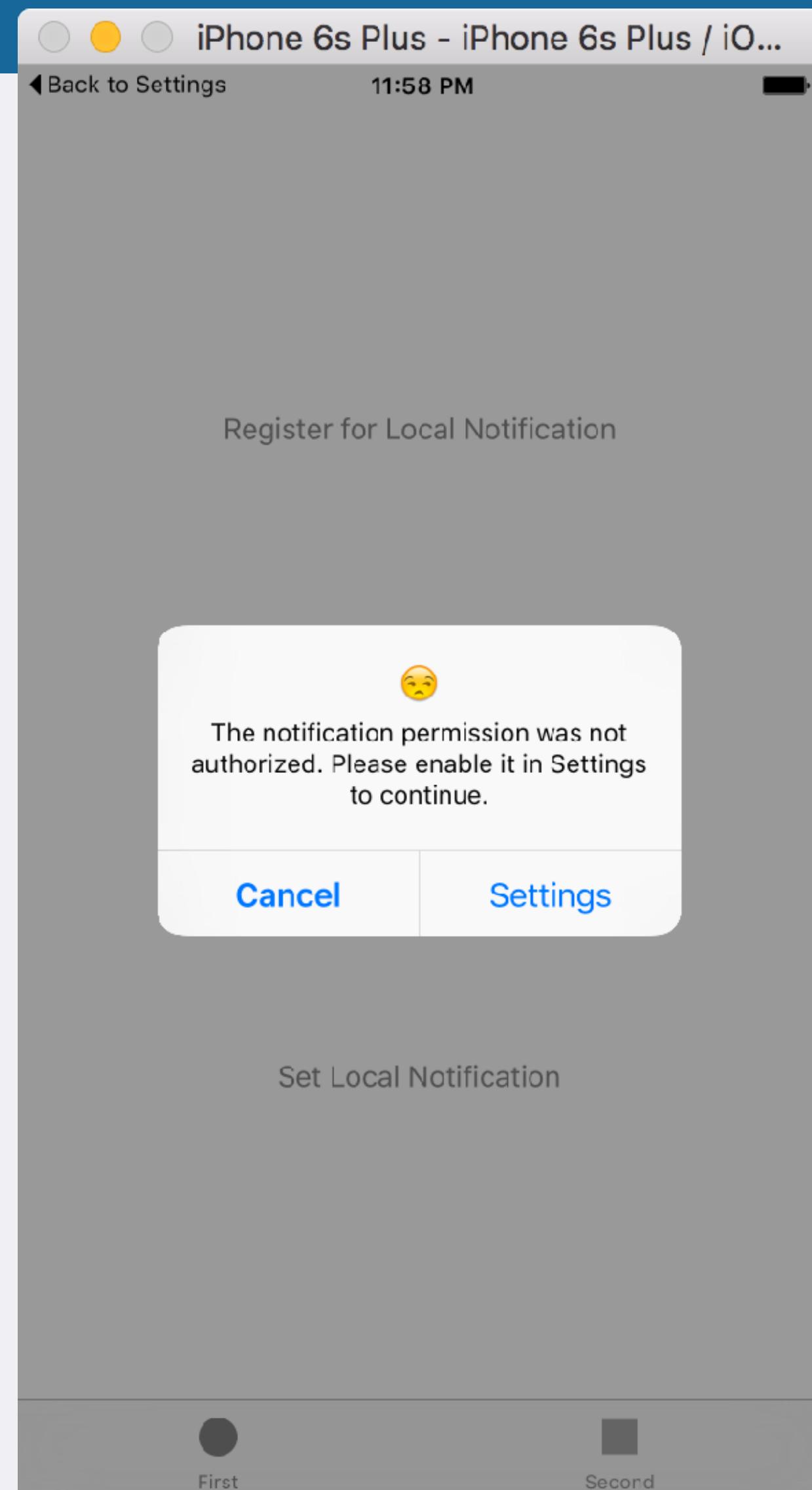
PERMISSIONS

```
zation to send notifications  
icication.sharedApplication().currentU  
{  
    alertController(title: "😢", message:  
        preferredStyle: .Alert)  
  
    letAction(title: "Settings", style:  
        UIAlertActionStyle.default){  
        url(string: UIApplicationOpenSettingsURLString)  
        application().openURL(url!)  
  
(settingsAction)  
  
    letAction(title: "Cancel", style: .Cancel){  
        (cancelAction)  
  
        alertController, animated: true, comple
```



```
{ return }  
as not authorized. Please enable it
```

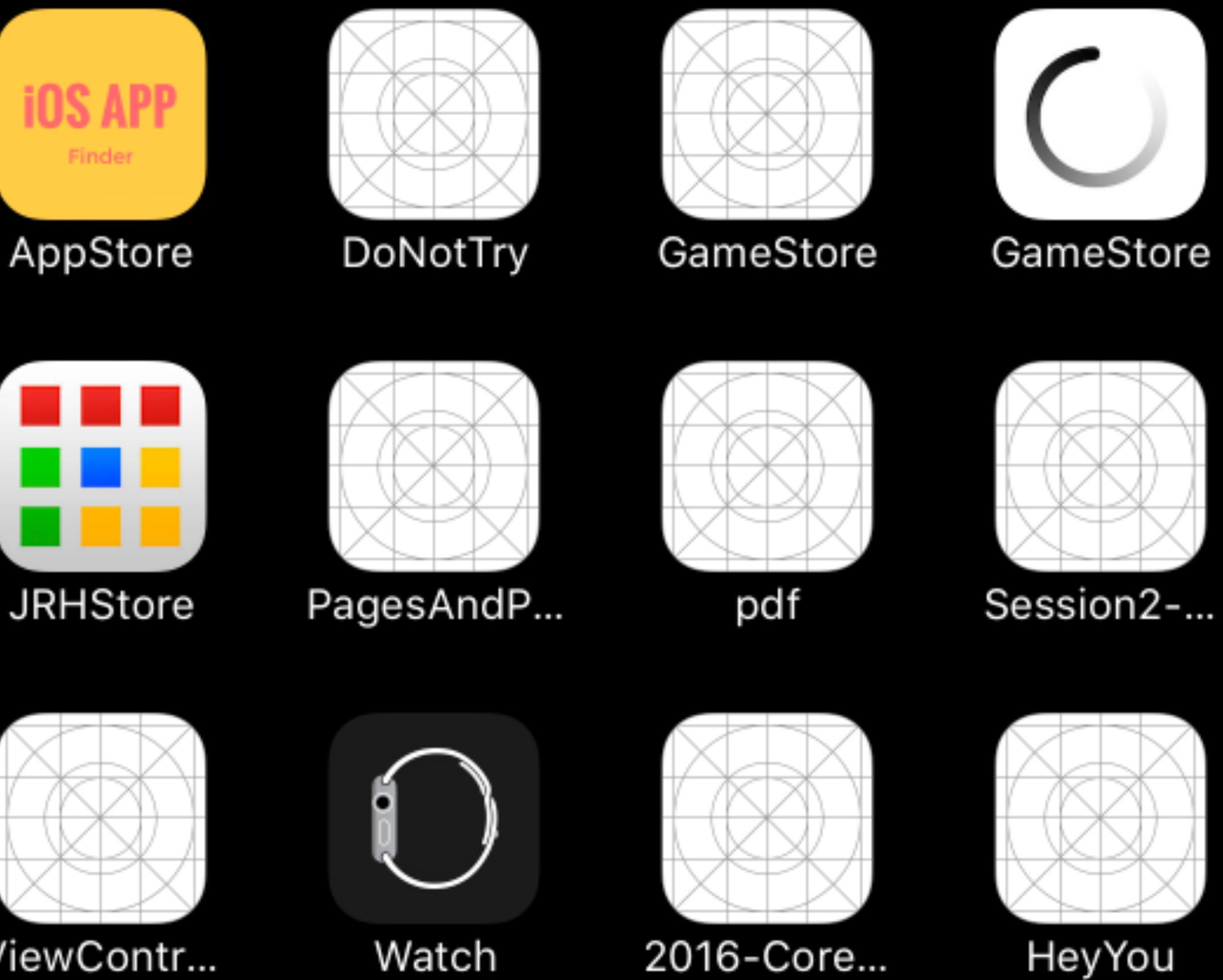
PERMISSIONS



LOCAL NOTIFICATIONS

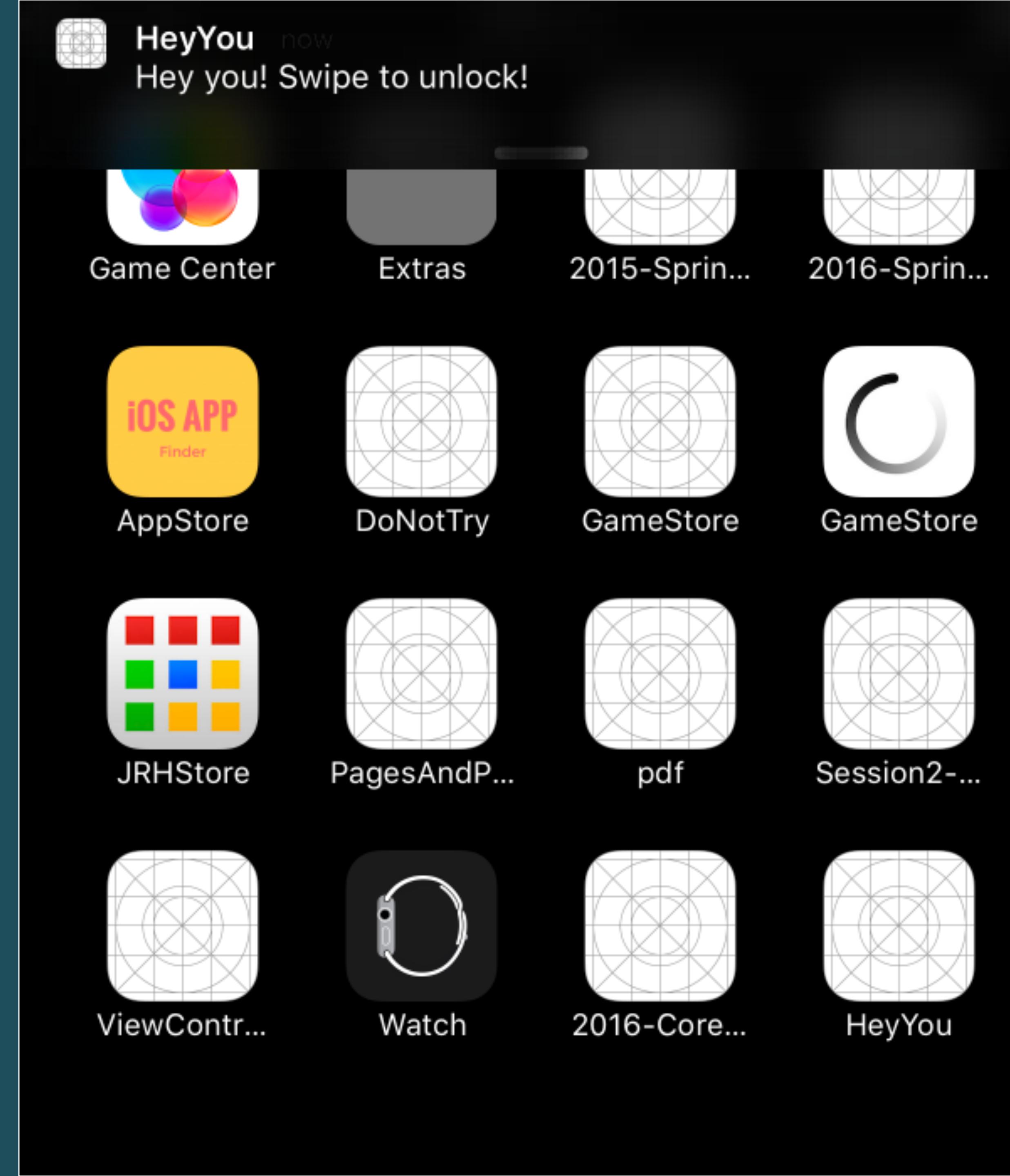
LOCAL NOTIFICATIONS

- Local notifications are delivered by the iOS device



LOCAL NOTIFICATIONS

- Messages are handled differently depending on the state of the application
 - Running
 - Background
 - Terminated
- Developer responsibility to handle each case



LOCAL NOTIFICATIONS

Launch app from notification

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {  
    if let notification:UILocalNotification = launchOptions?[UIApplicationLaunchOptionsLocalNotificationKey] as? UILocalNotification {  
        print("Launch from local notification: \(notification)")  
    }  
  
    return true  
}
```

- Test for type of notification when launching from terminated state
- Different behavior
 - Go to a view controller
 - Set some date

LOCAL NOTIFICATIONS

Handle
notification from
suspended

```
led when the app is in the background
userNotificationCenter(_ center: UNUserNotificationCenter,
                      didReceive response: UNNotificationResponse,
                      withCompletionHandler completionHandler: @escaping () ->
(response)
if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    The user dismissed the notification without taking action
}
if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    The user launched the app
    print("AppDelegate: Did Receive: \(response)")
}
completionHandler()
```



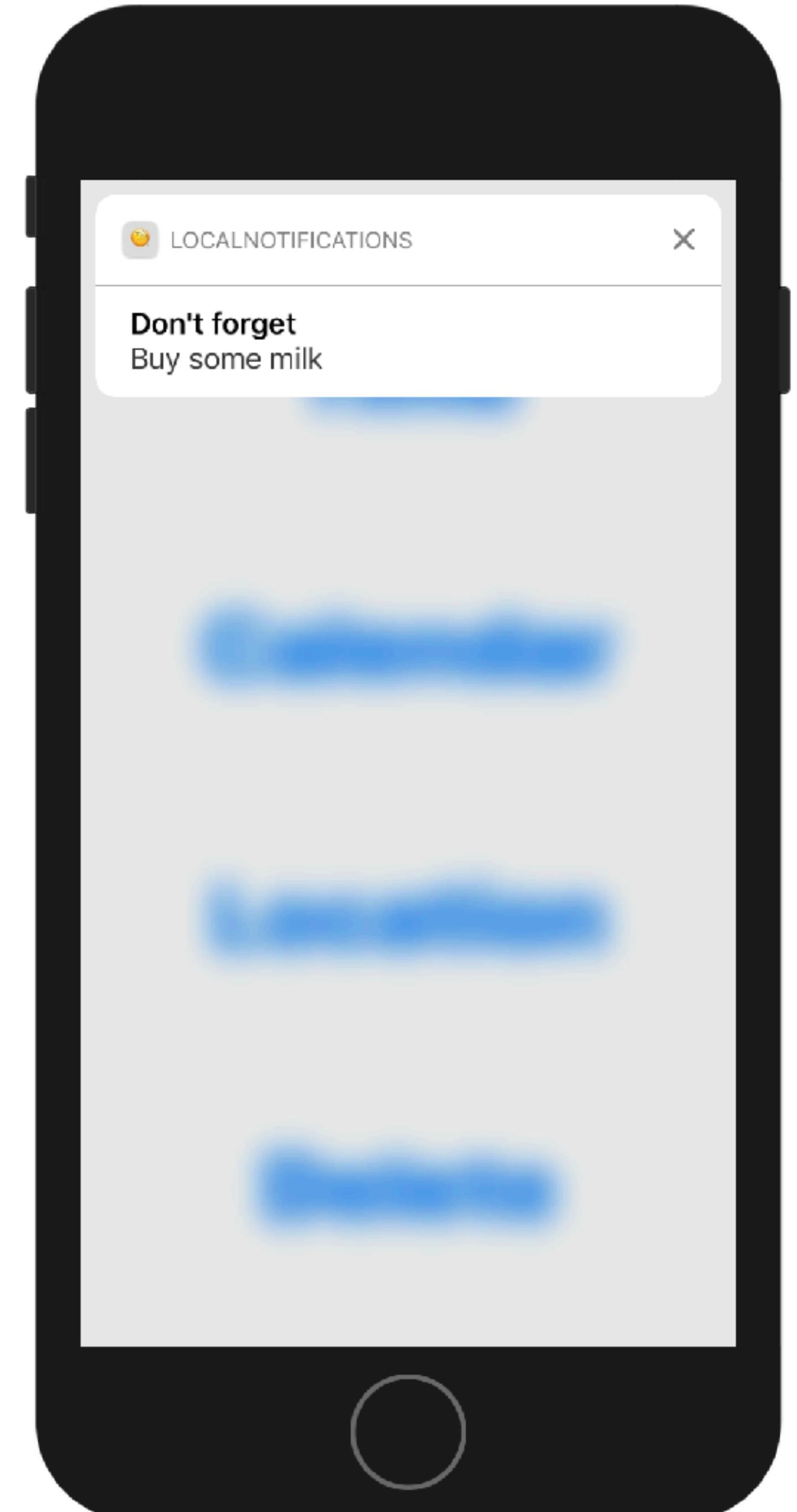
LOCAL NOTIFICATIONS

Have to
set
delegate
early

```
led when the app is in the background
    userNotificationCenter(_ center: UNUserNotificationCenter,
        didReceive response: UNNotificationResponse,
        withCompletionHandler completionHandler: @escaping () ->
(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        The user dismissed the notification without taking action
    } else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```

LOCAL NOTIFICATIONS

- Remember to handle all the states (if necessary)
 - App in foreground
 - App suspended
 - App terminated



SCHEDULING LOCAL NOTIFICATIONS

SCHEDULING NOTIFICATIONS

- Notification request contain content and a trigger

- `title` : String containing the primary reason for the alert.
- `subtitle` : String containing an alert subtitle (if required)
- `body` : String containing the alert message text
- `badge` : Number to show on the app's icon.
- `sound` : A sound to play when the alert is delivered. Use `UNNotificationSound.default()` or create a custom sound from a file.
- `launchImageName` : name of a launch image to use if your app is launched in response to a notification.
- `userInfo` : A dictionary of custom info to pass in the notification
- `attachments` : An array of `UNNotificationAttachment` objects. Use to include audio, image or video content.

Content sent with
notification

SCHEDULING NOTIFICATIONS

```
let content = UNMutableNotificationContent()  
content.title = "Don't forget"  
content.body = "Buy some milk"  
content.sound = UNNotificationSound.default()
```

- Notification request contain content and a trigger

SCHEDULING NOTIFICATIONS

- Triggers
 - Time
 - Calendar
 - Location

SCHEDULING NOTIFICATIONS

- Schedule a notification for a number of seconds later

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

_ = UNTimeIntervalNotificationTrigger(timeInterval: 300,
                                      repeats: false)
```

Time trigger

SCHEDULING NOTIFICATIONS

- Trigger at a specific date and time
- The trigger is created using a date components object which makes it easier for certain repeating intervals

Calendar trigger

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

let date = Date(timeIntervalSinceNow: 3600)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,.second],  
from: date)

let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate,  
repeats: false)
```

SCHEDULING NOTIFICATIONS

```
let triggerDaily = Calendar.current.dateComponents([.hour,.minute,.second], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDaily, repeats: true)
```

```
let triggerWeekly = Calendar.current.dateComponents([.weekday,.hour,.minute,.second], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerWeekly, repeats: true)
```

- Repeating notifications

SCHEDULING NOTIFICATIONS

Location trigger

```
let trigger = UNLocationNotificationTrigger(triggerWithRegion:region, repeats:false)
```

- Trigger when a user enters or leaves a geographic region
- The region is specified through a CoreLocation `CLRegion`
- You need to get permissions just like `CoreLocation`



SCHEDULING NOTIFICATIONS

```
UNUserNotificationCenter.current().  
    getPendingNotificationRequests {  
        requests in  
        print(requests)  
    }
```

- List all the current notifications

SCHEDULING NOTIFICATIONS

```
let center = UNUserNotificationCenter.current()  
center.removeAllPendingNotificationRequests()
```

- Delete notifications

SCHEDULING NOTIFICATIONS

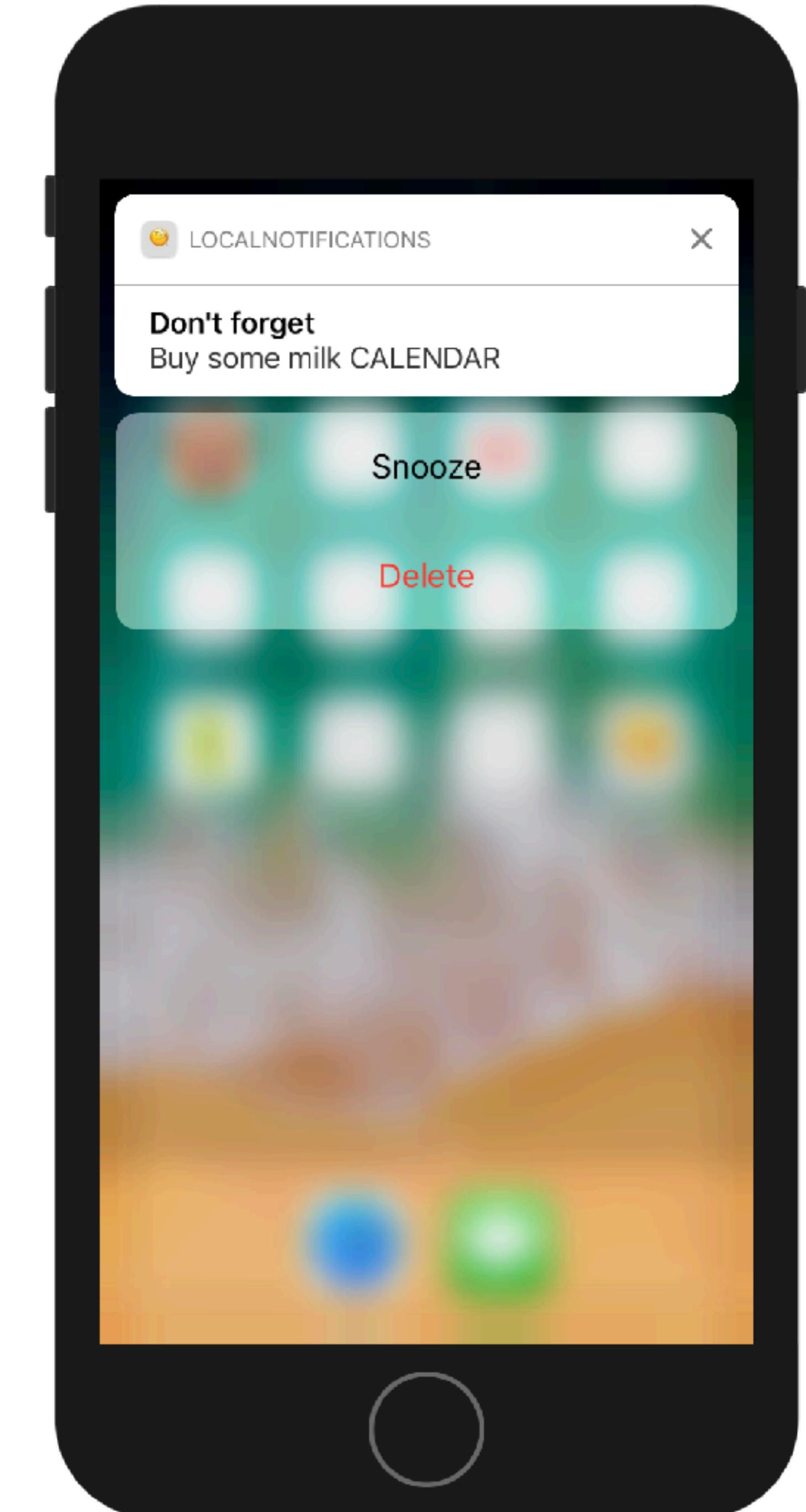
```
@IBAction func tapDelete(_ sender: Any) {  
    listNotification()  
  
    let center = UNUserNotificationCenter.current()  
    center.removeAllPendingNotificationRequests()  
  
    listNotification()  
}
```

- Delete notifications

NOTIFICATION ACTIONS

NOTIFICATION ACTIONS

- Define custom actions for your notifications



NOTIFICATION ACTIONS

```
notificationAction(identifier: "Snooze", title: "S  
notificationAction(identifier: "DeleteAction", tit  
  
cationCategory(identifier: "CalendarCategory",  
actions: [snoozeAction, deleteActi  
intentIdentifiers: [], options: [  
.current().setNotificationCategories([category]
```

NOTIFICATION ACTIONS

```
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,.second], from:  
date)  
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)  
  
// Actions  
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])  
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:  
[.destructive])  
let category = UNNotificationCategory(identifier: "CalendarCategory",  
                                     actions: [snoozeAction,deleteAction],  
                                     intentIdentifiers: [], options: [])  
UNUserNotificationCenter.current().setNotificationCategories([category])  
  
// Add a notification  
let identifier = "Time Notification"  
let request = UNNotificationRequest(identifier: identifier,  
                                    content: content,  
                                    trigger: trigger)
```

NOTIFICATION ACTIONS

```
didReceive response: UNNotificationResponse,  
withCompletionHandler completionHandler: @escaping () -> Void) {  
  
print("AppDelegate: Did Receive: \(response)")  
  
// Determine the user action  
switch response.actionIdentifier {  
case UNNotificationDismissActionIdentifier:  
    print("Dismiss Action")  
case UNNotificationDefaultActionIdentifier:  
    print("Default")  
case "Snooze":  
    print("Snooze")  
case "DeleteAction":  
    print("Delete")  
default:  
    print("Unknown action")  
}
```

Switch on the
actionIdentifier

PUSH NOTIFICATIONS

PUSH NOTIFICATIONS

- Sent from Apple's servers to device
- Apple Push Notification Service (APNS) is the gateway for push notification
 - SSL certificate
 - Provisioning
 - Networking code
 - Device tokens

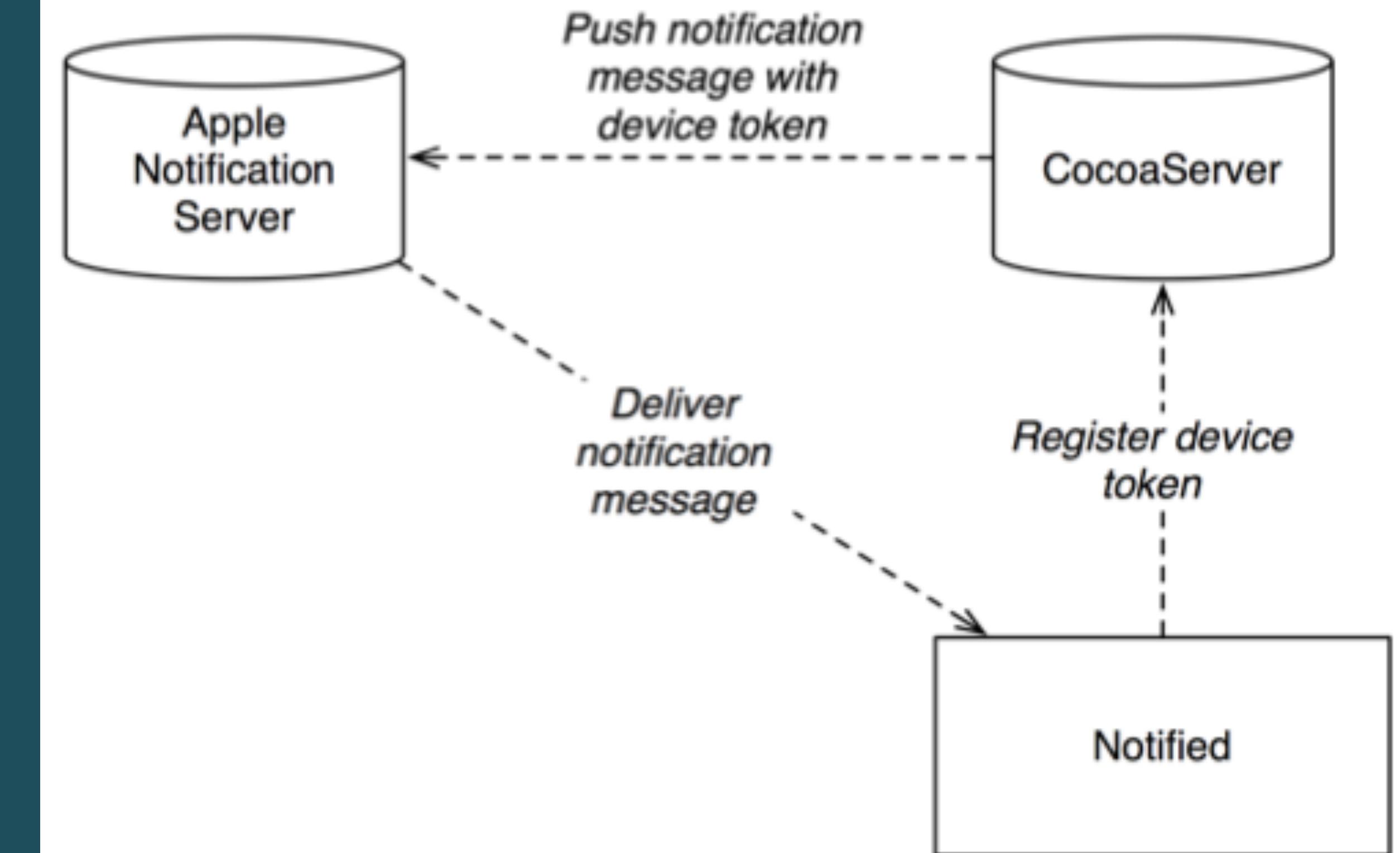
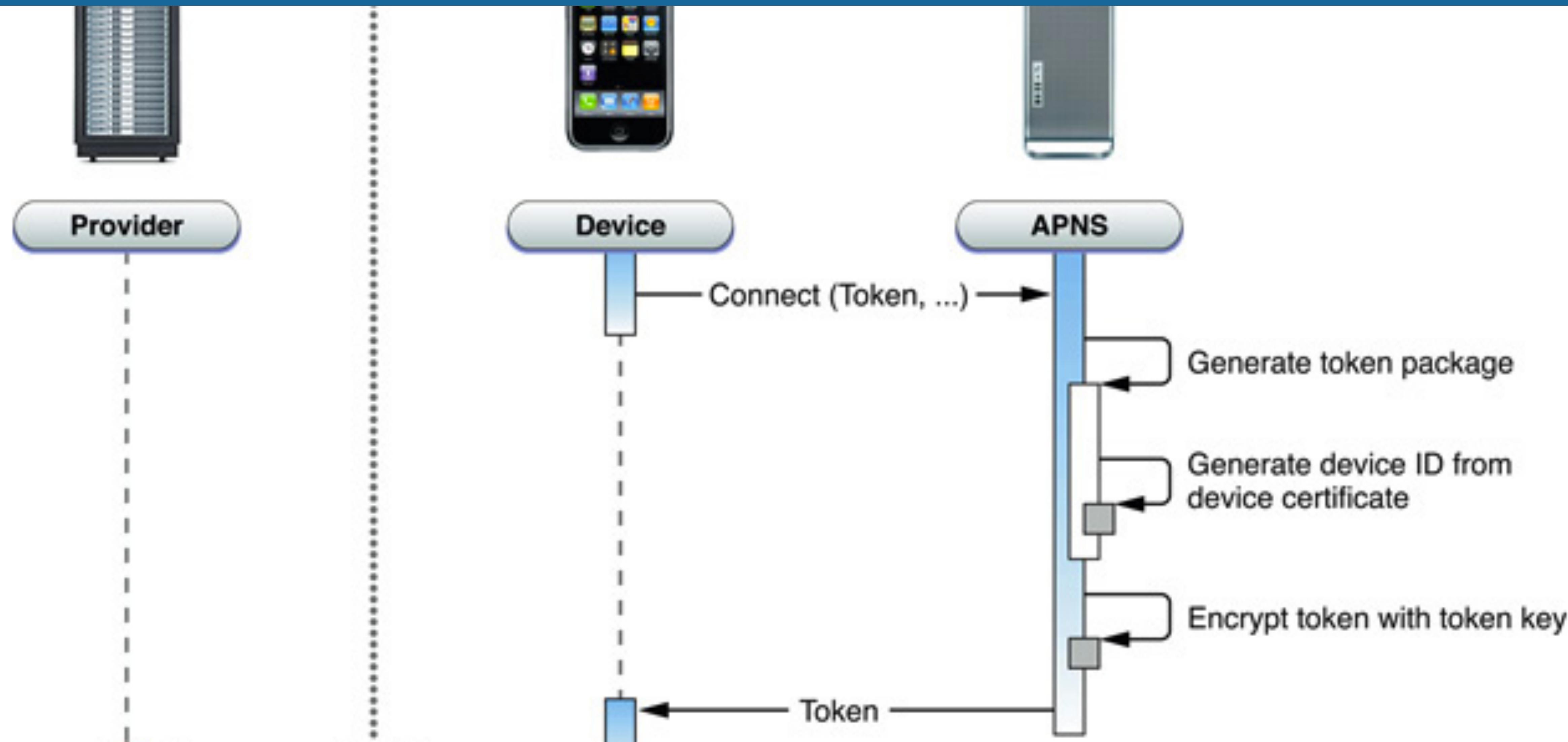


image via Big Nerd Ranch iOS Programming

PUSH NOTIFICATIONS

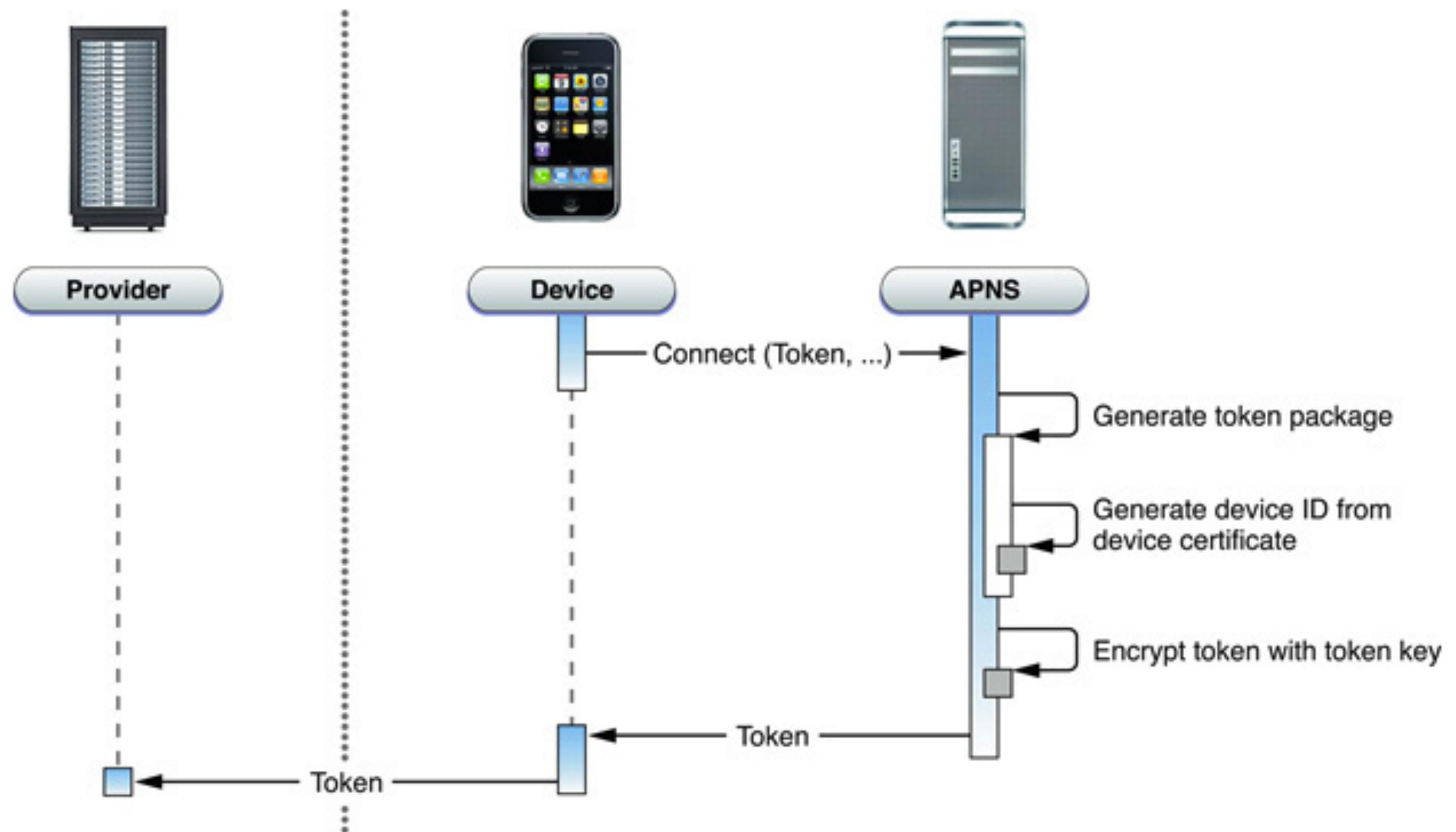
1. An app enables push notification (the user has to confirm that he wishes to receive these notifications)
2. The app receives a “device token”
3. The app sends the device token to your server
4. The server sends a push notification to the Apple Push Notification Service
5. APNS sends the push notification to the user’s device

PUSH NOTIFICATIONS



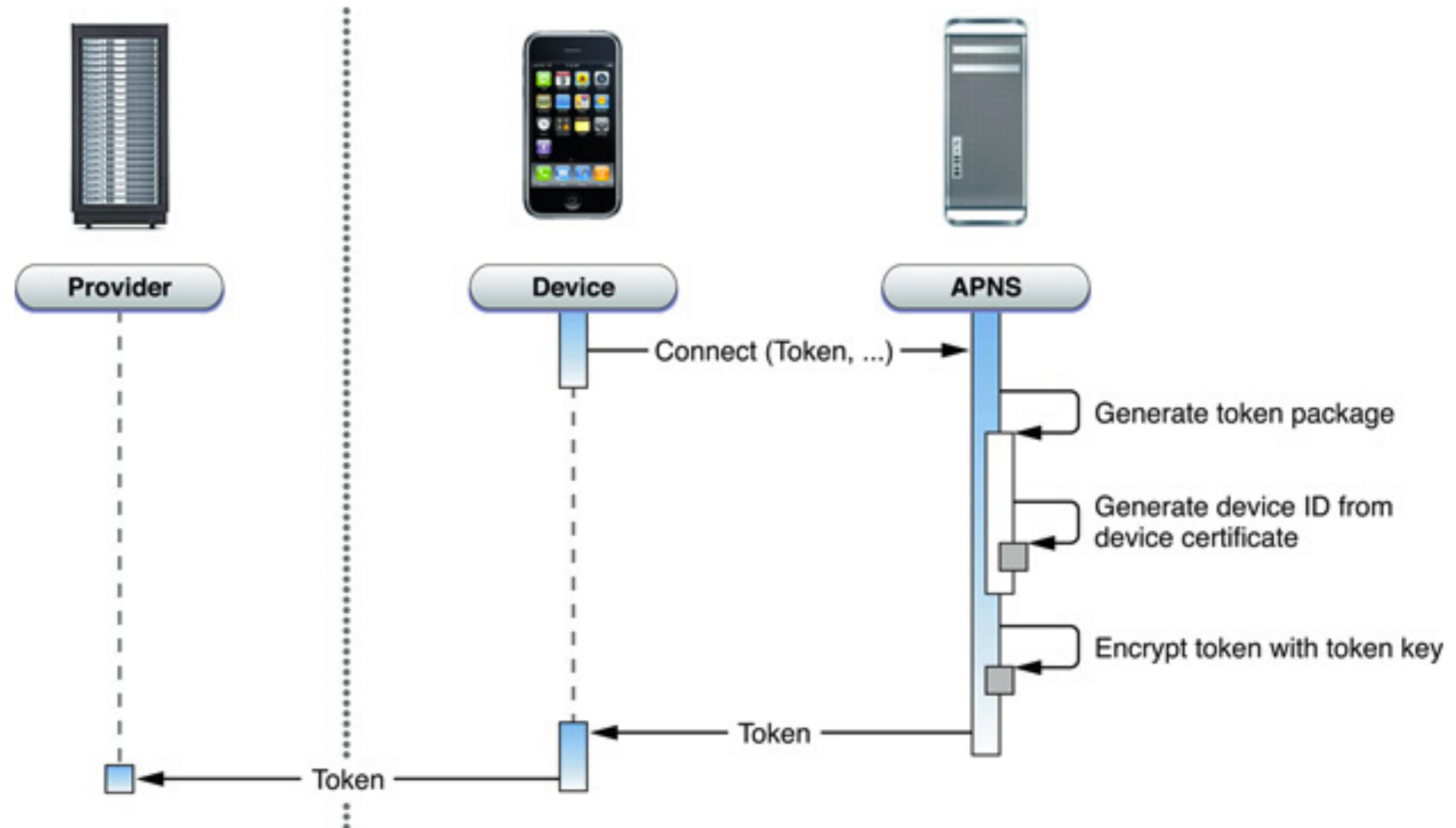
PUSH NOTIFICATION

- Requirements for push notifications
 - iOS device
 - Paid iOS Developer membership
 - Server
 - Ability to run background processes
 - Install SSL certificates
 - Make outgoing TLS connections on non-standard ports



PUSH NOTIFICATION

- Many third party solutions for push notifications
 - Parse
 - Firebase
 - App Engine
 - ...



PUSH NOTIFICATIONS

```
{"aps": {  
    "badge": 100,  
    "alert": "Push Notification Test",  
    "sound": "SoundFile.aif"},  
    "type": "websiteUrl",  
    "data": "http://mySite.com"  
}
```

Manage the badges in your code (not incremental)

Include sound files in your bundle

- Push notification payload
 - JSON dictionary
 - <256 bytes

Pass any key-values in the notification

PUSH NOTIFICATION

- Receiving notification require app delegate methods to be implemented
- Code path is the same for local notification

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: @escaping () -> Void) {
    print(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        // The user dismissed the notification without taking action
    }
    else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        // The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

Urban Airship

..... powering modern mobile

Products Customers Resources

The image shows the Urban Airship platform interface. On the left, there's a screenshot of the web-based dashboard with a sidebar containing 'Push Composer', 'Rich Push Composer', 'New Message', 'Drafts (0)', 'History', and 'Reports'. The main area is titled 'Create your message' with a 'Give Your Message a Title' field containing '\$1 OFF Any Classic Espresso Drink'. Below it is a WYSIWYG editor with various styling options. A preview of the message is shown with a large '\$1 OFF' coupon graphic. On the right, there are two smartphones displaying the resulting push notification. The first phone shows the full coupon message from 'TIMBER RIDGE COFFEE ROASTERS'. The second phone shows a portion of the inbox with messages like 'Getting warm \$1 off an ic...', 'music download of the...', '1 lb. of whole bean cof...', 'punch card is full; free...', 'location opening', and 'punch card is nearly full'. A red button at the bottom right says 'Explore Rich Push'.

Rich Push

It just got easier to create rich experiences. Use WYSIWYG editor to add surveys and more.

Explore Rich Push

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

nomad / houston

Code Issues 27 Pull requests 5 Projects 0 Wiki Insights

Apple Push Notifications; No Dirigible Required <http://nomad-cli.com>

houston notifications ruby cli nomad apns

157 commits 1 branch 19 releases 32 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

mpvosseller authored and dankimio committed on Jan 24 Set default passphrase to nil instead of the empty string. (#158) Latest commit 592bbeb on Jan 24

bin	Set default passphrase to nil instead of the empty string. (#158)	a month ago
lib	Set default passphrase to nil instead of the empty string. (#158)	a month ago
spec	Mutable content: update README, add spec (#137)	a year ago
.gitignore	Update .gitignore	a year ago
Gemfile	Update code style (#134)	a year ago
LICENSE	Updating copyright	3 years ago
README.md	README: use unregistered_devices instead of devices (#156)	5 months ago
Rakefile	Rakefile: add Bundler tasks, add default task	a year ago
houston.gemspec	Update dependencies	a year ago
README.md		



HOUSTON

NOTIFICATIONS WRAP-UP

- Notifications keep users engage by opening your app
- Notifications are not guaranteed to be delivered by Apple
- Tips and tricks:
 - Push was important before multi-tasking in iOS4, but many uses can be implemented as local notifications
 - Third-party option for remote notifications
 - Use CloudKit for remote notifications via subscriptions
- Don't abuse notifications...this is one thing that Apple seems to regularly enforce in the App Store Guidelines
 - "Push notifications can not be used for advertising"

REVIEW

NOTIFICATIONS

NOTIFICATIONS

No more interruptions

Notifications appear at the top of your screen and disappear quickly.



One-swipe access

Swipe down to reveal Notification Center.



See more at a glance
View stocks and weather, too.

NOTIFICATIONS

No more interruptions

Notifications appear at the top of your screen and disappear quickly.

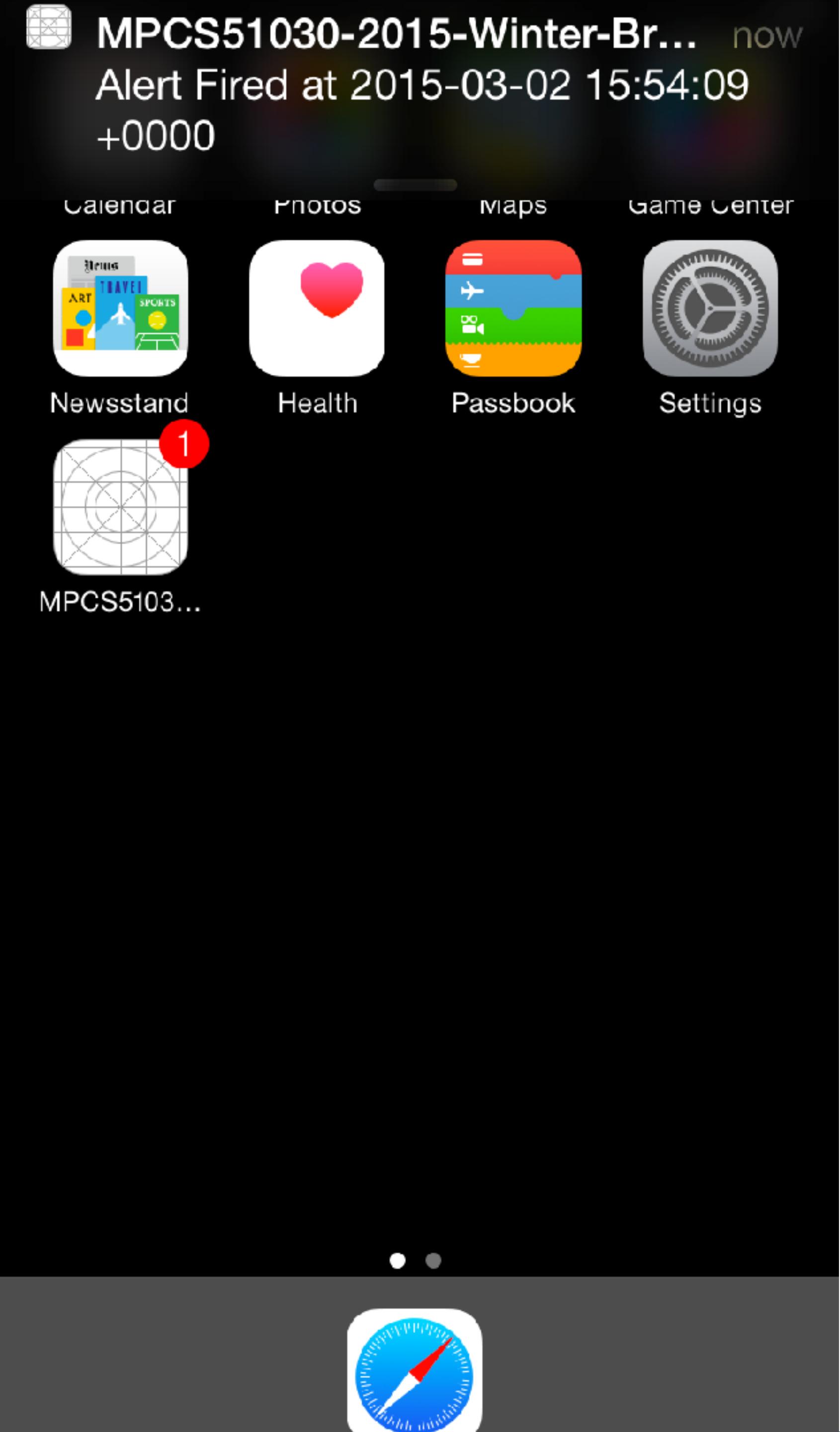


"User Notification"
not broadcast
notifications

See more
at a glance
View stocks and
weather, too.

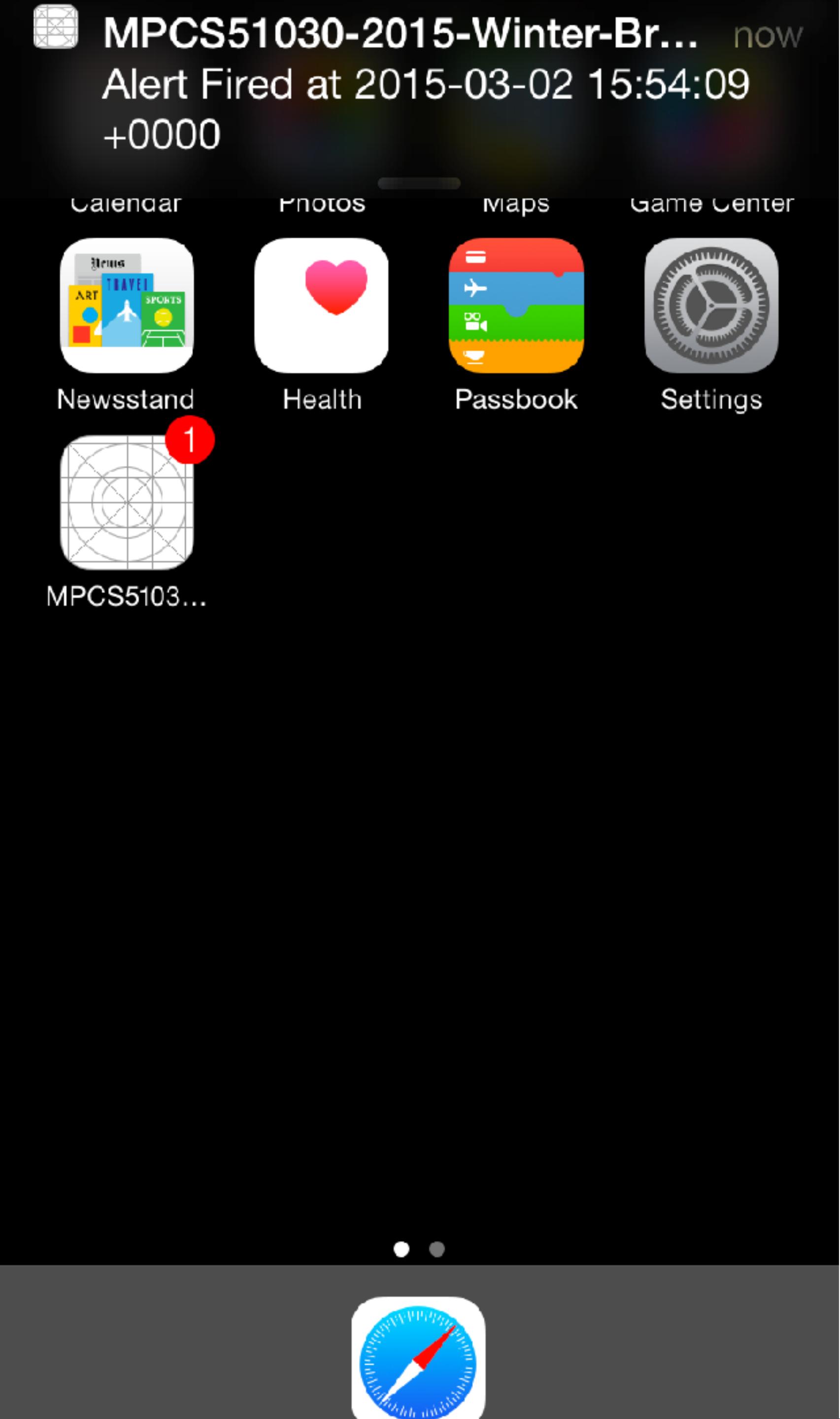
NOTIFICATIONS

- Only one application can run in the foreground
- Notifications give you flexibility in how to interact with users when they're not running your app
 - Messages
 - Game turn
 - Updates available



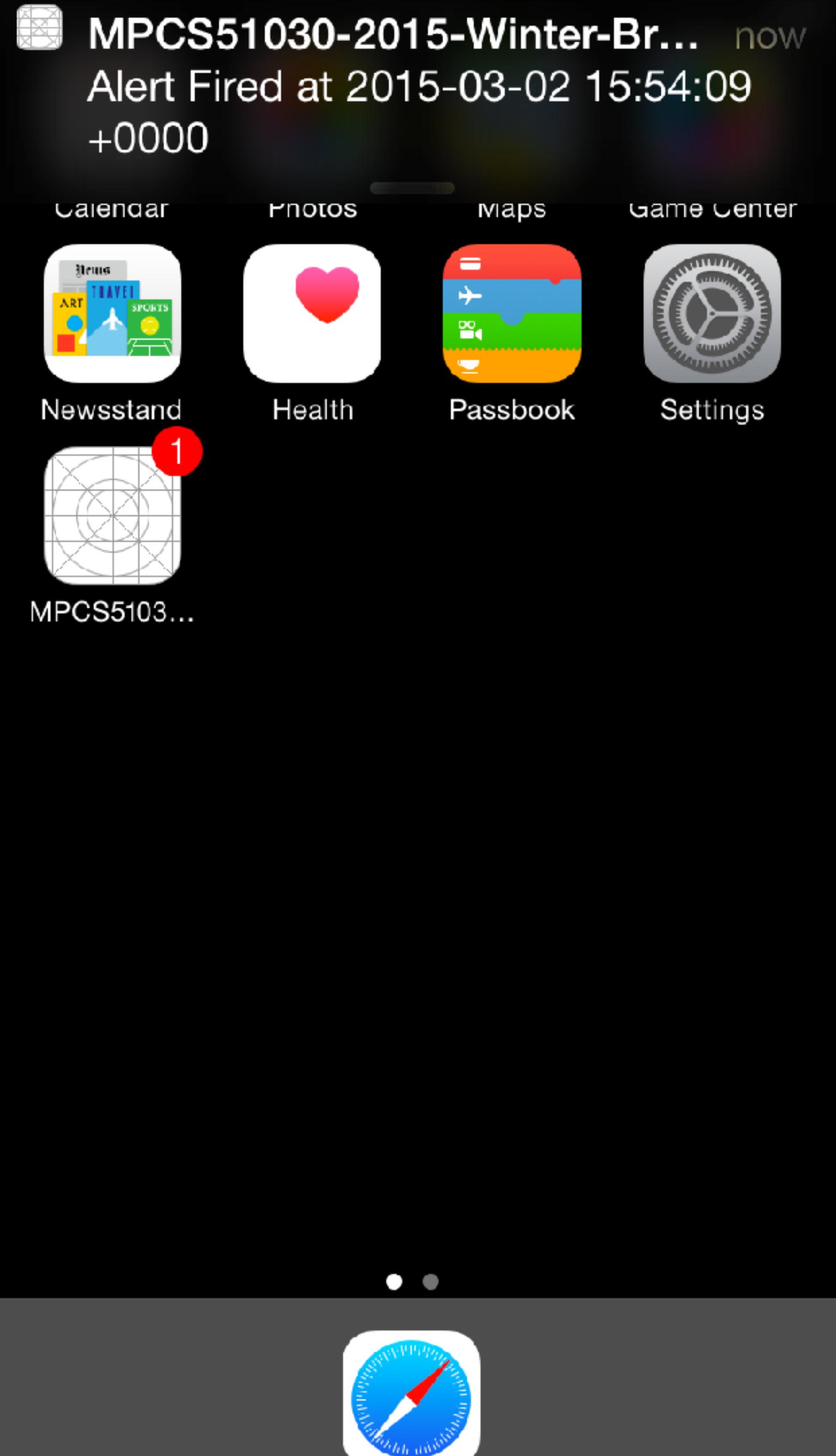
NOTIFICATIONS

- Re-engage your users
 - Notifications can launch your app



NOTIFICATIONS

- Notification styles
 - Modal
 - Alert
 - Sounds (default, custom)
 - Badges
 - Silent
 - Happens in background
 - “Trigger event”



NOTIFICATIONS

- Local Notifications
 - Scheduled by an application
 - Delivered by the iOS on the device
 - “Schedule a notification”

Carrier

9:56 AM

Back

Notifications

Show in Notification Center 5 >

Sounds



Badge App Icon



Show on Lock Screen

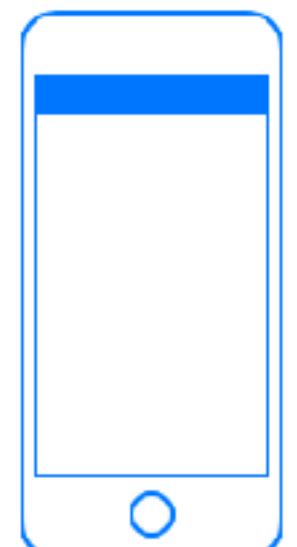


Show alerts on the lock screen, and in
Notification Center when it is accessed from
the lock screen.

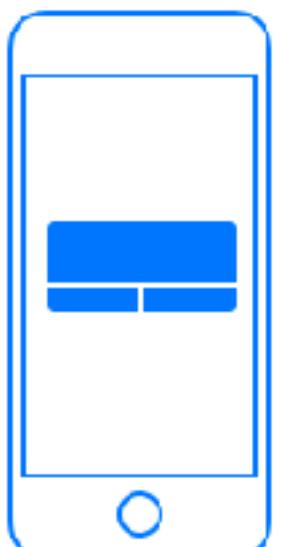
ALERT STYLE WHEN UNLOCKED



None



Banners



Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and
go away automatically.

NOTIFICATIONS

- Push Notifications
 - Remote notifications
 - Sent by application's remote server to Apple Push Notification Service (APNS)
 - "Register a notification"

Carrier

9:56 AM

Back

Notifications

Show in Notification Center 5 >

Sounds



Badge App Icon



Show on Lock Screen

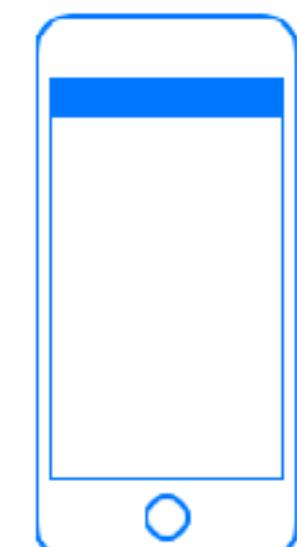


Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

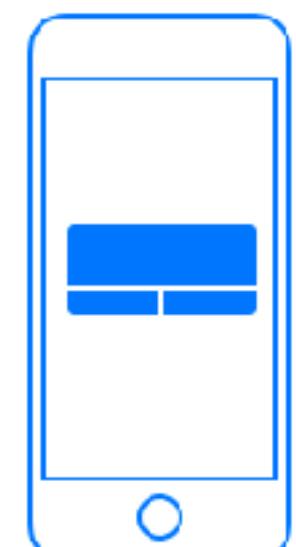
ALERT STYLE WHEN UNLOCKED



None



Banners



Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and go away automatically.

NOTIFICATIONS

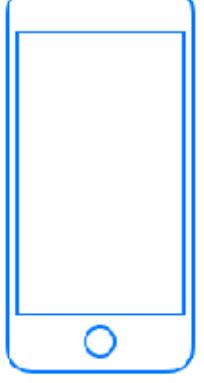
Sounds

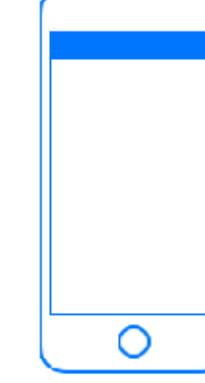
Badge App Icon

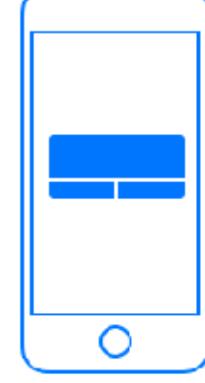
Show on Lock Screen

Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

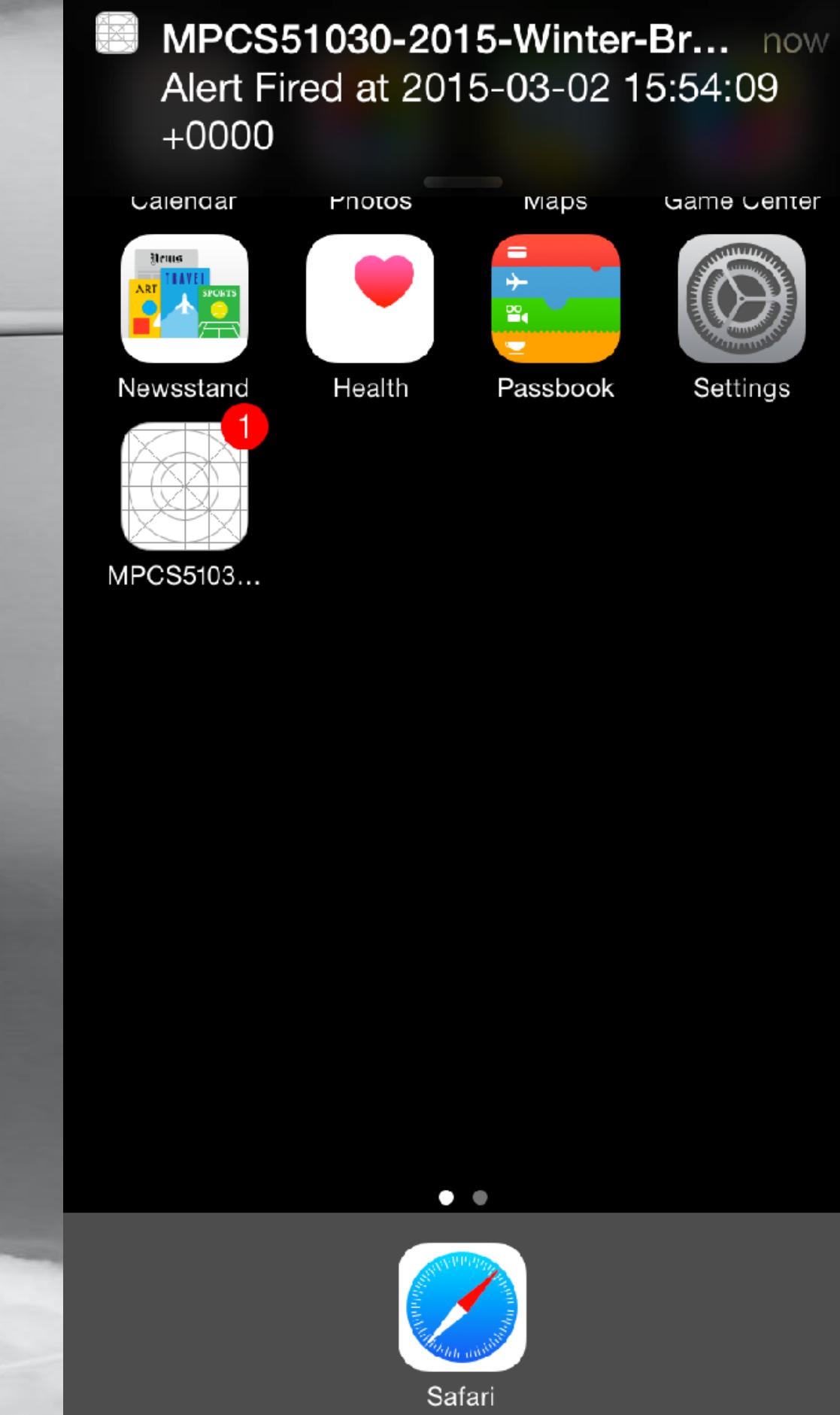
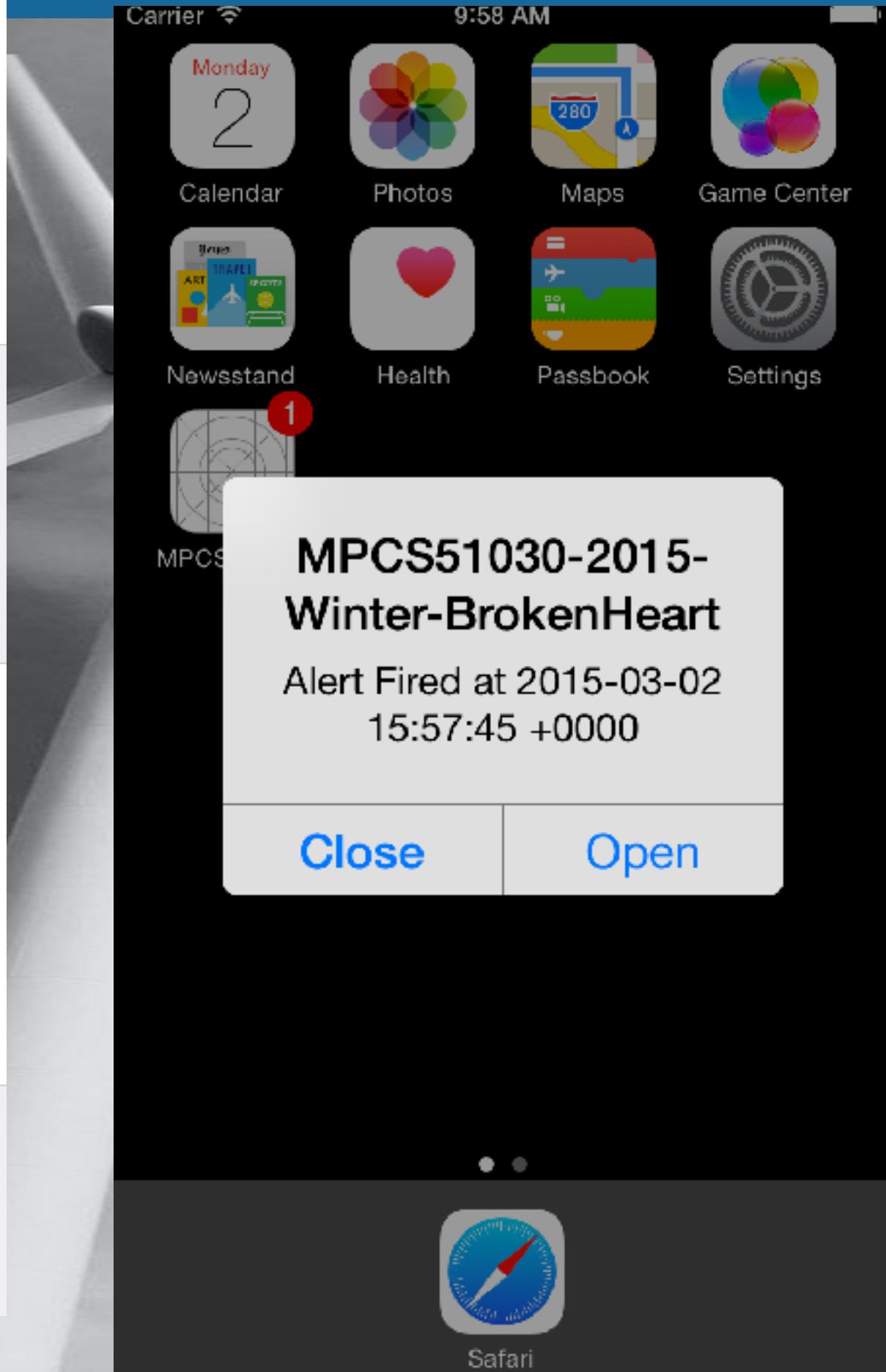
ALERT STYLE WHEN UNLOCKED

 None

 Banners

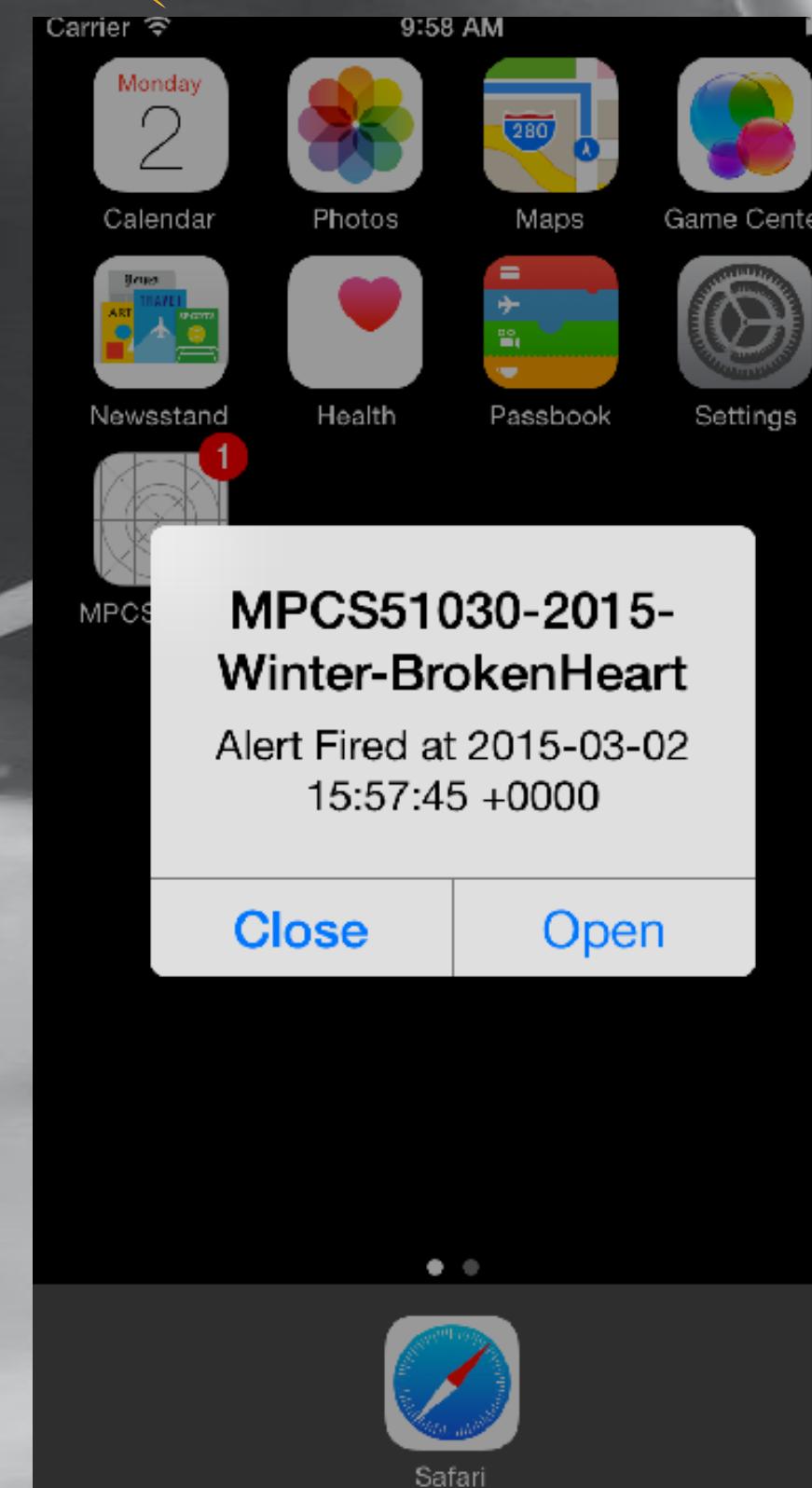
 Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and go away automatically.

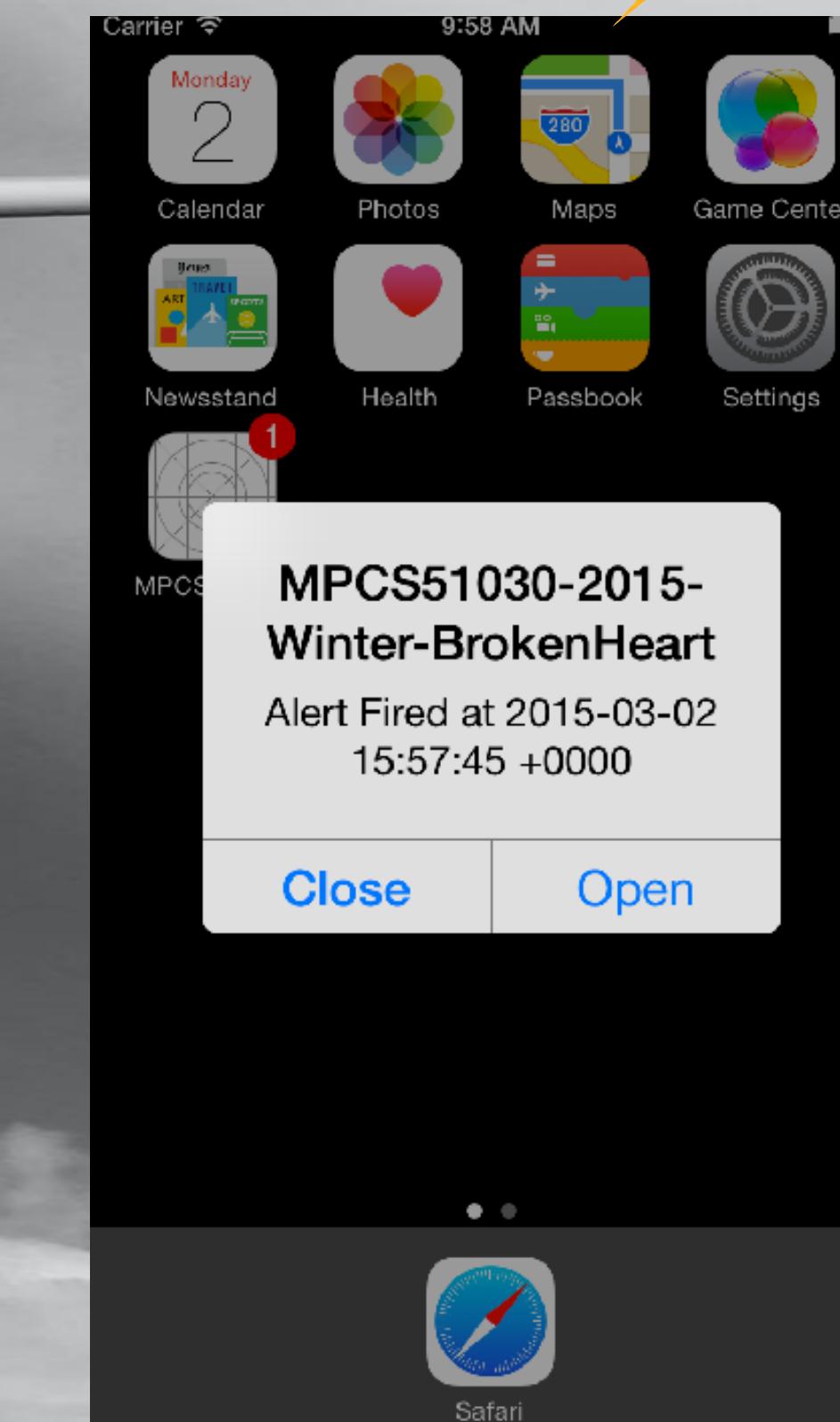


NOTIFICATIONS

Local



Push



Appears the same on devices

NOTIFICATIONS

- Handling notifications in app delegate
 - application:didFinishLaunchingWithOptions:
 - application:didReceiveRemoteNotification:
 - application:didReceiveLocalNotification:

Options: local or push

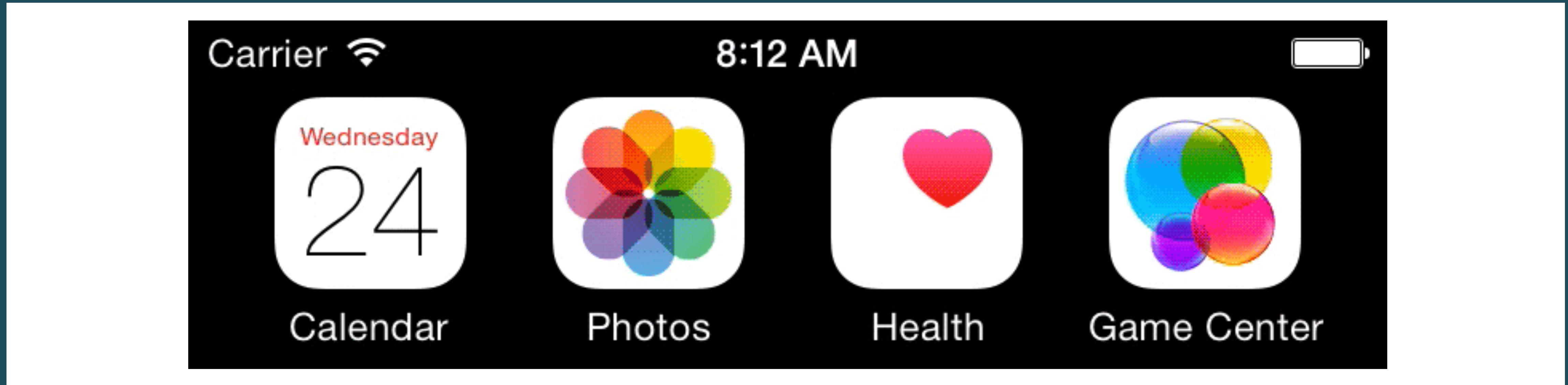
`UIApplicationLaunchOptionsLocalNotificationKey`

NOTIFICATIONS

- iOS8 introduced interactive notifications
 - Engage with application without opening it



NOTIFICATIONS



- iOS8 introduced interactive notifications
 - Engage with application without opening it
- iOS9 allows text input (eg. messages) from interactive notifications
- iOS10 unified the API for working with UserNotification Framework

PERMISSIONS

PERMISSIONS

- Remember that users are always in control
 - Keep them informed
 - Do not bother them
 - Of course you are trying to get their permission to bombard them with notifications)
 - Make sure your app still functions without notifications
 - Inform the user of consequences
 - “To be informed on the latest info, enable notifications in Settings” don’t forget to show a link to settings

Carrier

9:56 AM

Back

Notifications

Show in Notification Center 5 >

Sounds



Badge App Icon



Show on Lock Screen

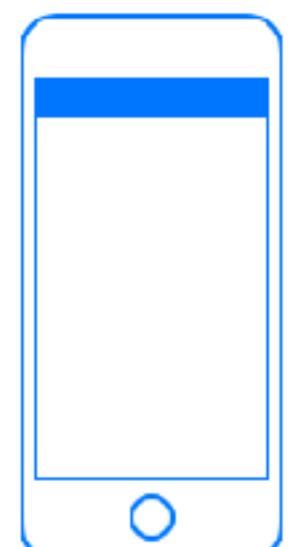


Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

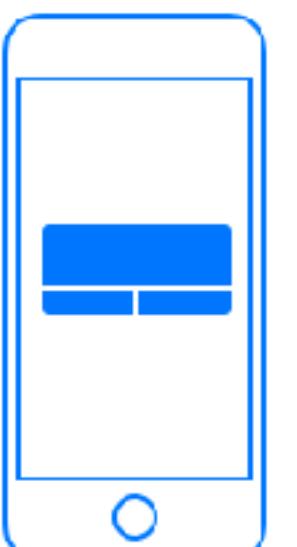
ALERT STYLE WHEN UNLOCKED



None



Banners



Alerts

Alerts require an action before proceeding.
Banners appear at the top of the screen and go away automatically.

PERMISSIONS

```
let options: UNAuthorizationOptions = [.alert, .sound];
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: options) {
    (granted, error) in
    if !granted {
        print("Something went wrong")
    }
}
```

- Registering for notifications

PERMISSIONS

"HeyYou" Would Like to Send You Notifications

Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.

Standard Dialogue

Don't Allow

OK

PERMISSIONS

- Consideration on notifications
 - Users may not accept to receive notifications
 - Users can change preferences in settings
- Be prepared to handle all potential use cases

```
center.getNotificationSettings { (settings) in
    if settings.authorizationStatus != .authorized {
        // Notifications not allowed
    }
}
```

PERMISSIONS

- The old way
 - “Go to Settings > Privacy > Location > OUR_APP”



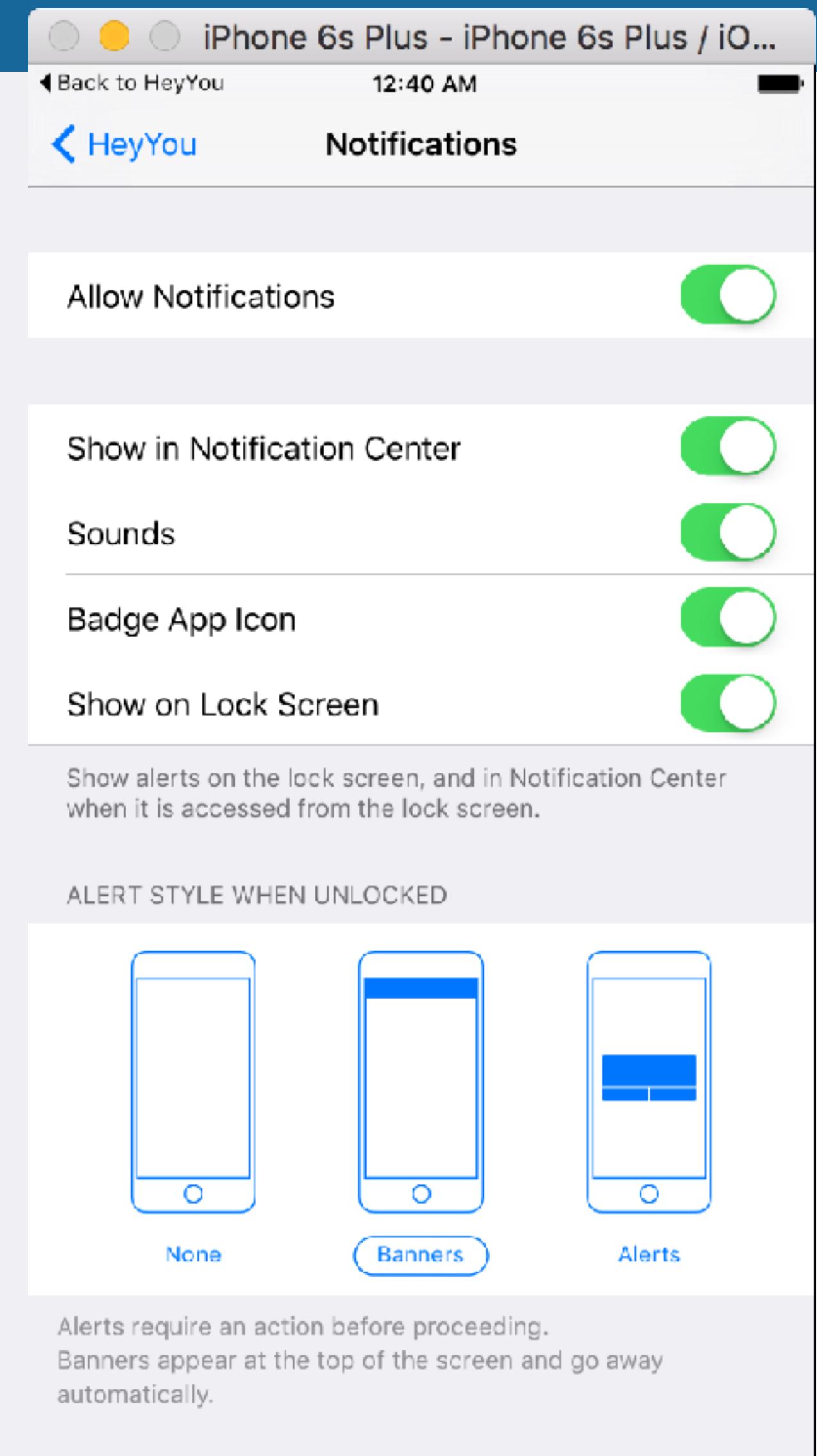
VERIZON 4G LTE 8:20 AM

< Settings Swarm

ALLOW SWARM TO ACCESS

	Location	While Using >
	Contacts	<input type="checkbox"/>
	Photos	<input checked="" type="checkbox"/>
	Camera	<input checked="" type="checkbox"/>
	Notifications >	
	Background App Refresh	<input checked="" type="checkbox"/>
	Use Cellular Data	<input checked="" type="checkbox"/>

PERMISSIONS



PERMISSIONS

```
e have authorization to send notifications
settings = UIApplication.sharedApplication().currentUserNotificationSettings() else { return }

t, let the user know and give them the option of going directly
to the settings screen for the application
types == .None {

n alert view controller
controller = UIAlertController(title: "⚠",
                               message: "The notification permission was not authorized. Please enable it in Settings to continue.",
                               preferredStyle: .Alert)

o go to settings
settingsAction = UIAlertAction(title: "Settings", style: .Default) { (alertAction) in
    appSettings = NSURL(string: UIApplicationOpenSettingsURLString) {
        UIApplication.sharedApplication().openURL(appSettings)
    }
}

ller.addAction(settingsAction)

cancel action that does nothing
cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)
ller.addAction(cancelAction)

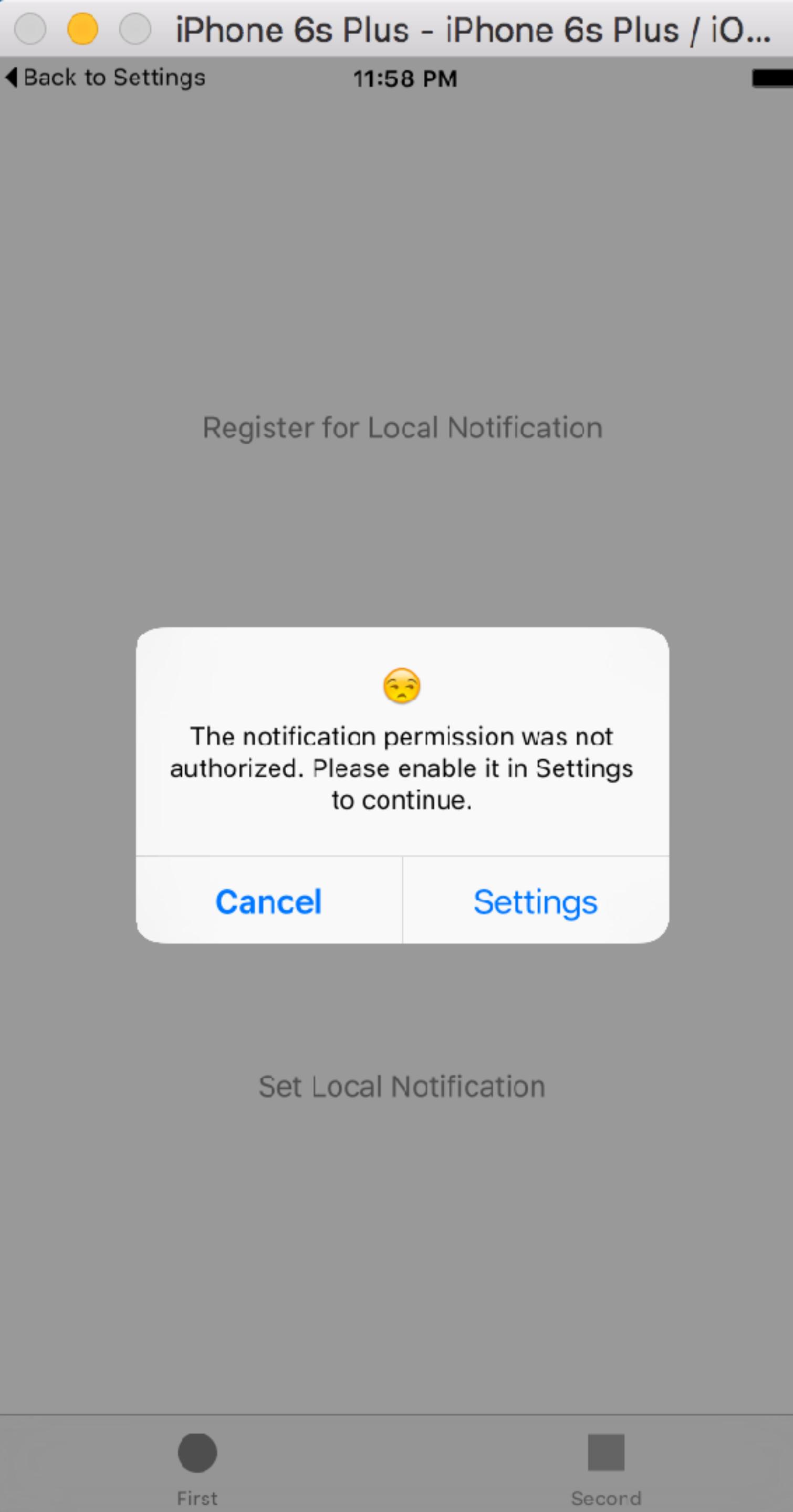
alert and exit early
dismissViewControllerAnimated(alertController, animated: true, completion: nil)
```



New
Way

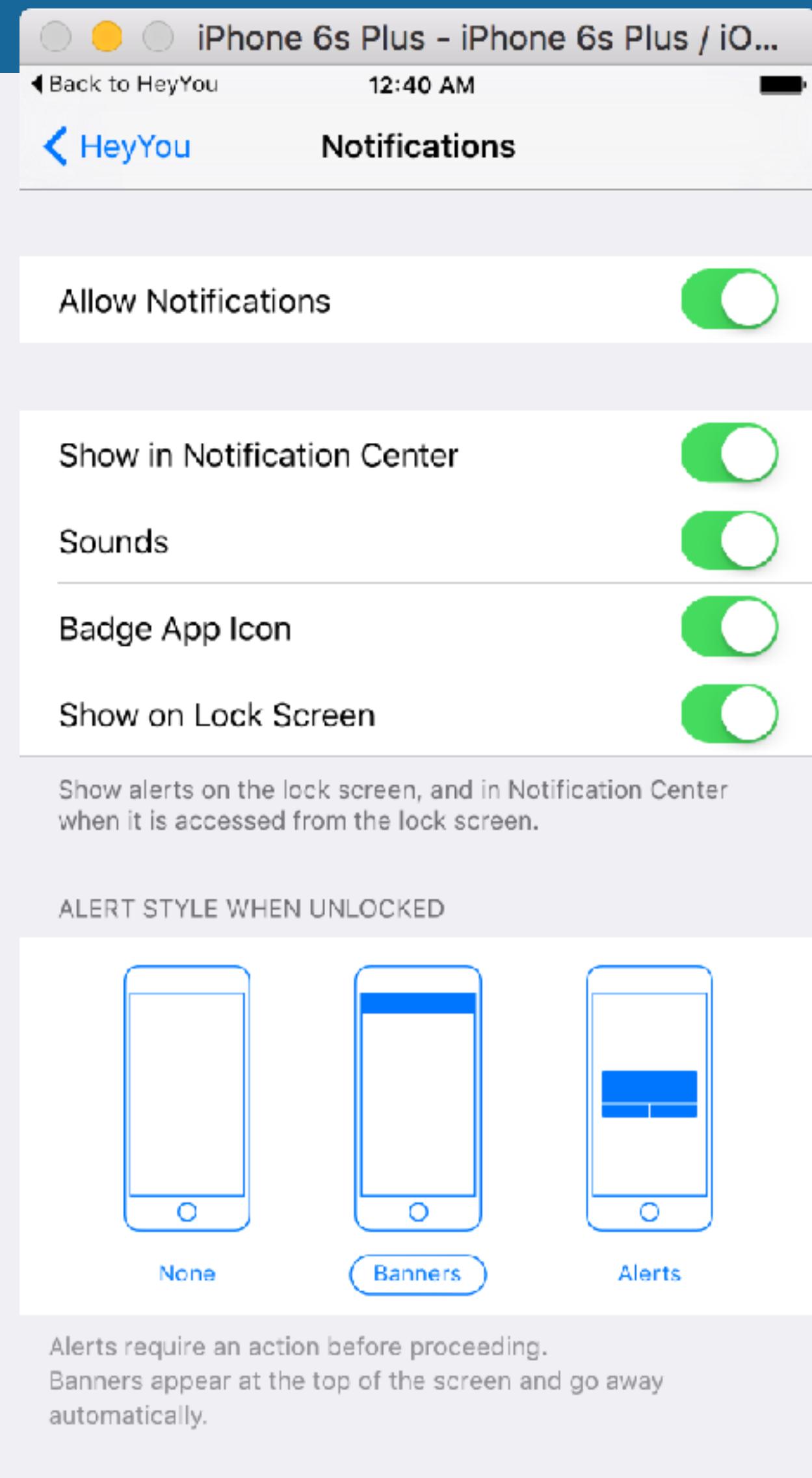
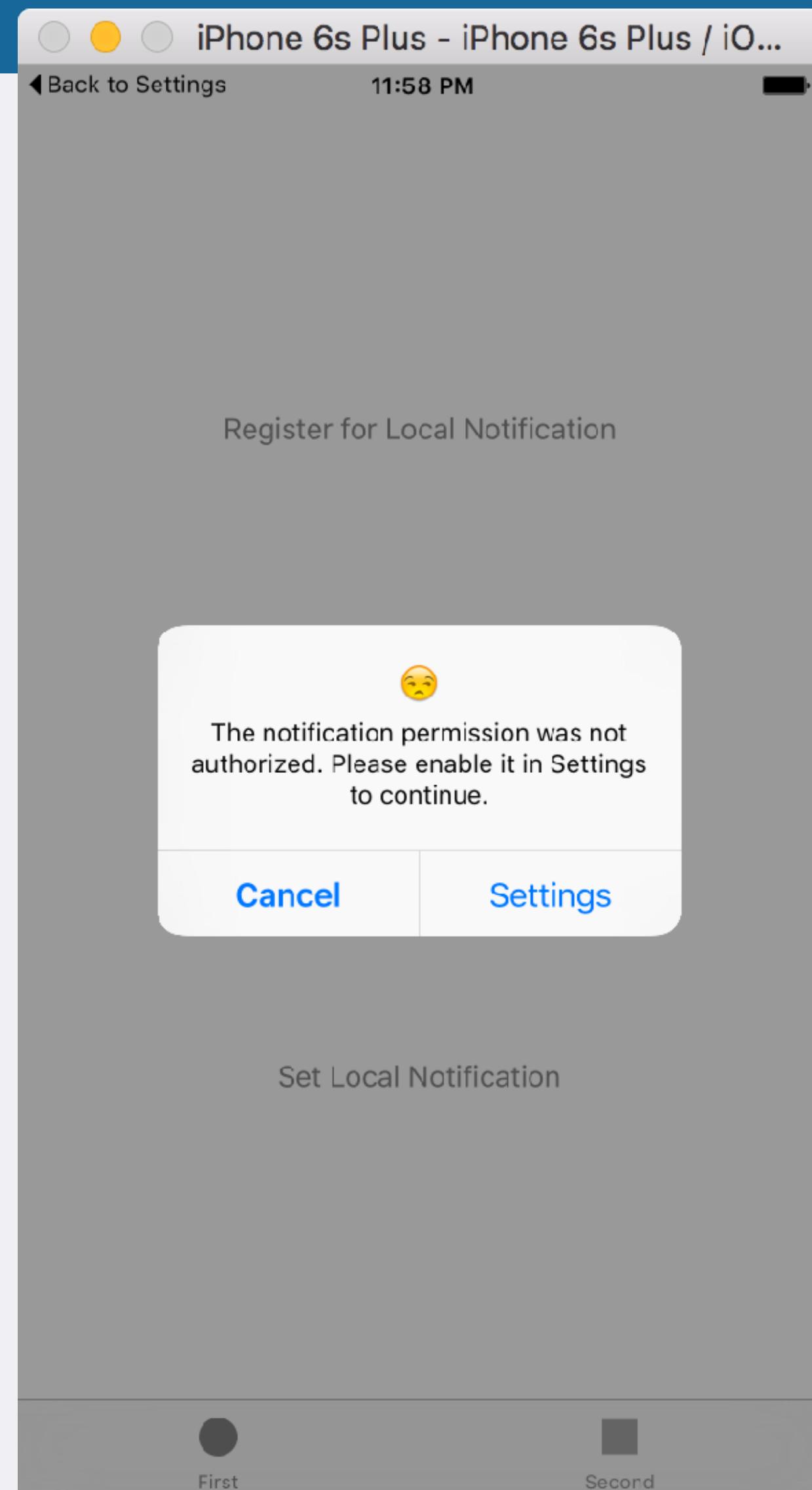
PERMISSIONS

```
zation to send notifications  
icication.sharedApplication().currentU  
{  
    lertController(title: "😢", message:  
        preferredStyle: .Alert)  
  
    ertAction(title: "Settings", style:  
        URL(string: UIApplicationOpenSetting  
pplication().openURL(appSettings)  
  
(settingsAction)  
  
tAction(title: "Cancel", style: .Ca  
(cancelAction)  
  
rtController, animated: true, comple
```



```
{ return }  
as not authorized. Please enable it
```

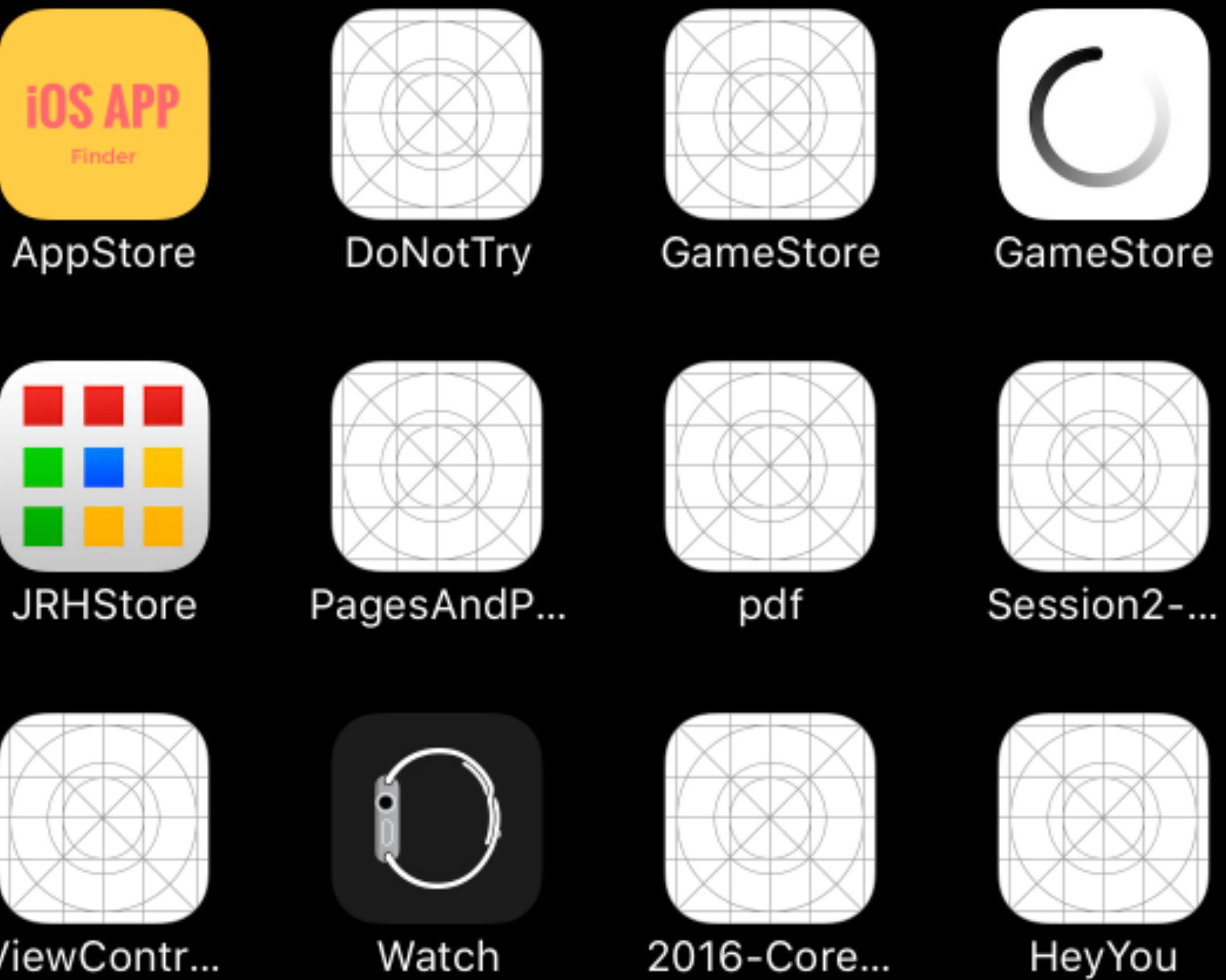
PERMISSIONS



LOCAL NOTIFICATIONS

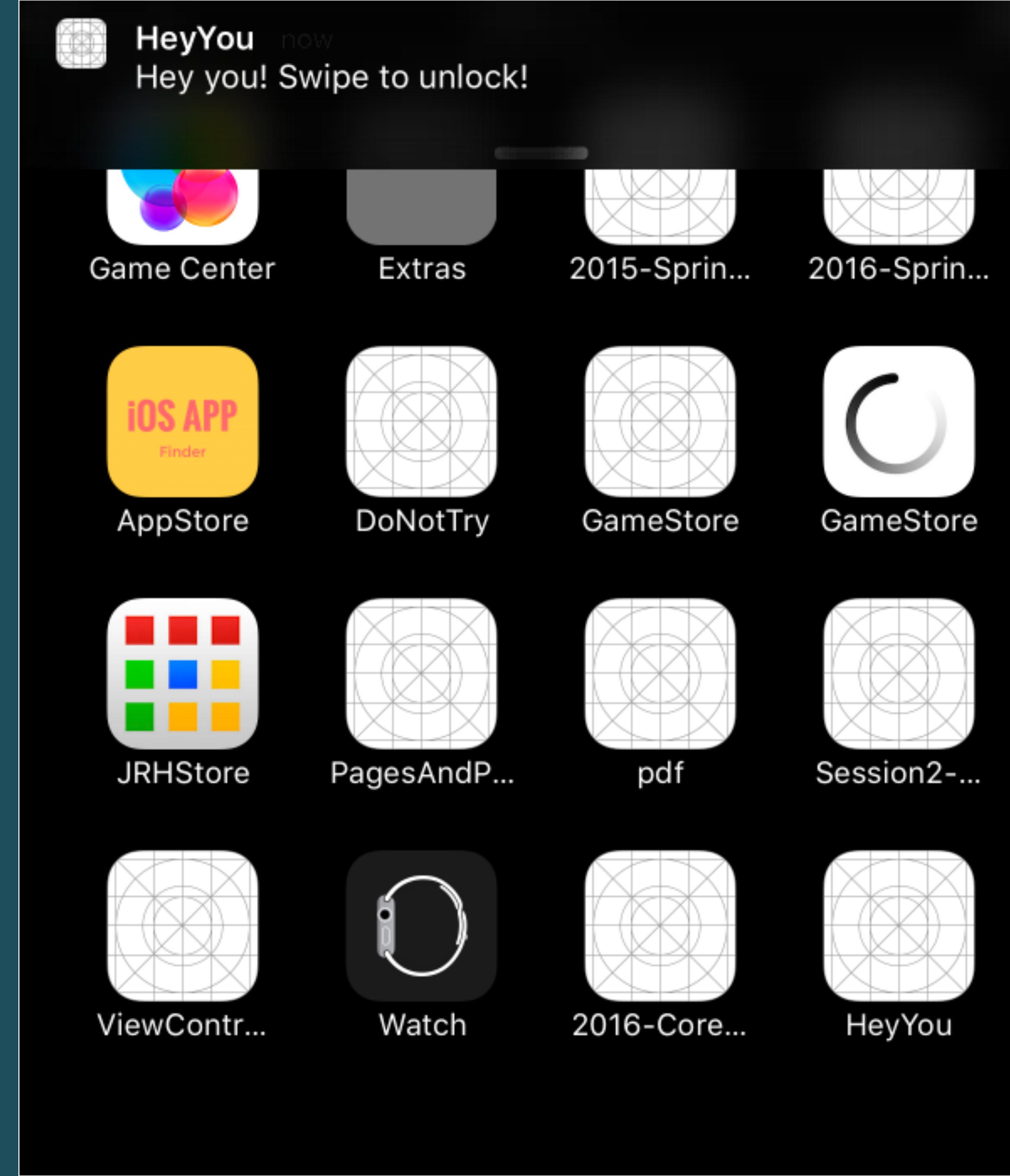
LOCAL NOTIFICATIONS

- Local notifications are delivered by the iOS device



LOCAL NOTIFICATIONS

- Messages are handled differently depending on the state of the application
 - Running
 - Background
 - Terminated
- Developer responsibility to handle each case



LOCAL NOTIFICATIONS

Launch app from notification

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {  
    if let notification:UILocalNotification = launchOptions?[UIApplicationLaunchOptionsLocalNotificationKey] as? UILocalNotification {  
        print("Launch from local notification: \(notification)")  
    }  
  
    return true  
}
```

- Test for type of notification when launching from terminated state
- Different behavior
 - Go to a view controller
 - Set some date

LOCAL NOTIFICATIONS

Handle
notification from
suspended

```
led when the app is in the background
userNotificationCenter(_ center: UNUserNotificationCenter,
                      didReceive response: UNNotificationResponse,
                      withCompletionHandler completionHandler: @escaping () ->
(response)
if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    The user dismissed the notification without taking action
}
if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    The user launched the app
    print("AppDelegate: Did Receive: \(response)")
}
completionHandler()
```



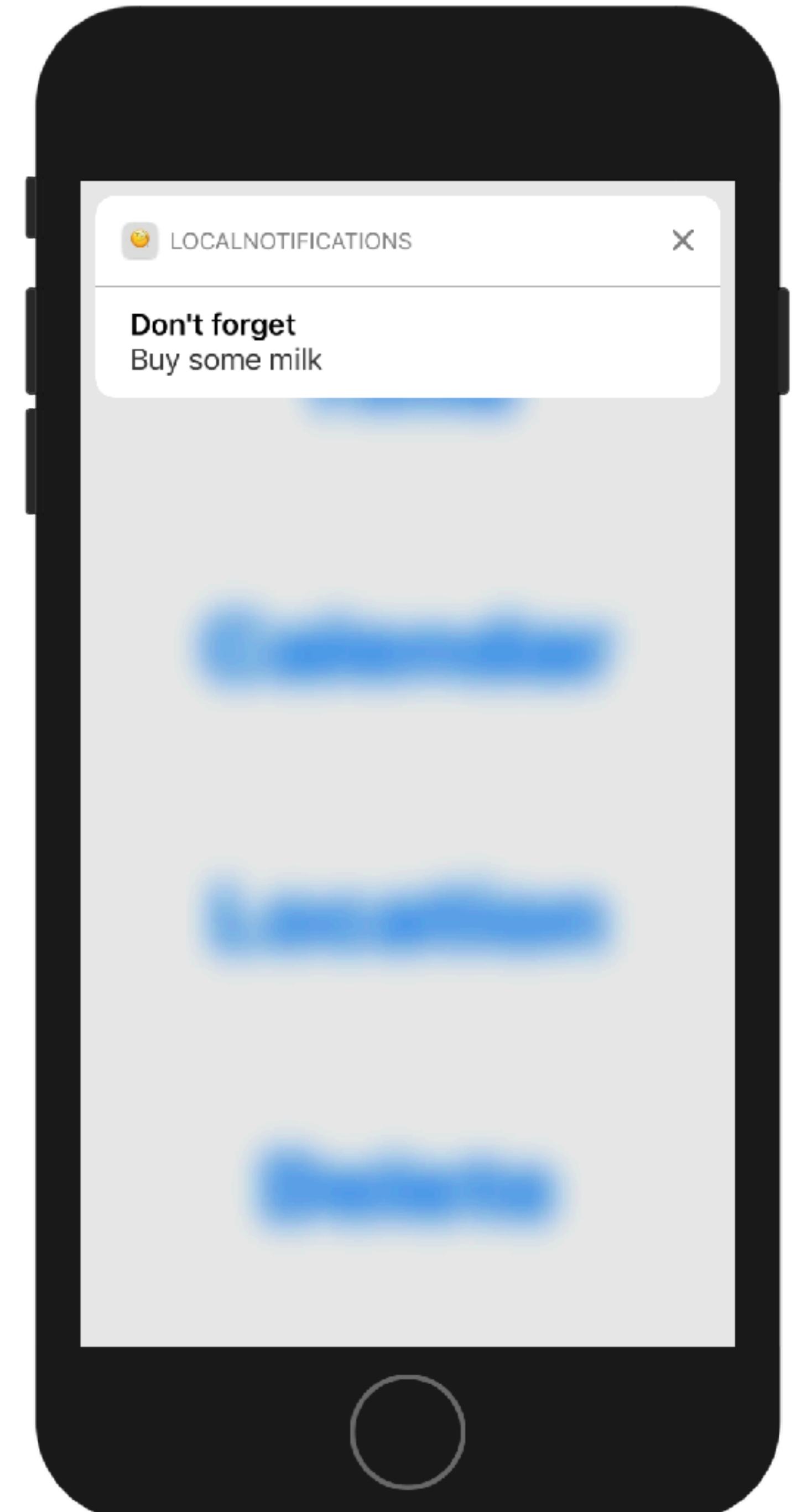
LOCAL NOTIFICATIONS

Have to
set
delegate
early

```
led when the app is in the background
    userNotificationCenter(_ center: UNUserNotificationCenter,
        didReceive response: UNNotificationResponse,
        withCompletionHandler completionHandler: @escaping () ->
(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        The user dismissed the notification without taking action
    } else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```

LOCAL NOTIFICATIONS

- Remember to handle all the states (if necessary)
 - App in foreground
 - App suspended
 - App terminated



SCHEDULING LOCAL NOTIFICATIONS

SCHEDULING NOTIFICATIONS

- Notification request contain content and a trigger

- `title` : String containing the primary reason for the alert.
- `subtitle` : String containing an alert subtitle (if required)
- `body` : String containing the alert message text
- `badge` : Number to show on the app's icon.
- `sound` : A sound to play when the alert is delivered. Use `UNNotificationSound.default()` or create a custom sound from a file.
- `launchImageName` : name of a launch image to use if your app is launched in response to a notification.
- `userInfo` : A dictionary of custom info to pass in the notification
- `attachments` : An array of `UNNotificationAttachment` objects. Use to include audio, image or video content.

Content sent with
notification

SCHEDULING NOTIFICATIONS

```
let content = UNMutableNotificationContent()  
content.title = "Don't forget"  
content.body = "Buy some milk"  
content.sound = UNNotificationSound.default()
```

- Notification request contain content and a trigger

SCHEDULING NOTIFICATIONS

- Triggers
 - Time
 - Calendar
 - Location

SCHEDULING NOTIFICATIONS

- Schedule a notification for a number of seconds later

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

_ = UNTimeIntervalNotificationTrigger(timeInterval: 300,
                                      repeats: false)
```

Time trigger

SCHEDULING NOTIFICATIONS

- Trigger at a specific date and time
- The trigger is created using a date components object which makes it easier for certain repeating intervals

Calendar trigger

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

let date = Date(timeIntervalSinceNow: 3600)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,.second],  
from: date)

let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate,  
repeats: false)
```

SCHEDULING NOTIFICATIONS

```
let triggerDaily = Calendar.current.dateComponents([.hour,.minute,.second], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDaily, repeats: true)
```

```
let triggerWeekly = Calendar.current.dateComponents([.weekday,.hour,.minute,.second], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerWeekly, repeats: true)
```

- Repeating notifications

SCHEDULING NOTIFICATIONS

Location trigger

```
let trigger = UNLocationNotificationTrigger(triggerWithRegion:region, repeats:false)
```

- Trigger when a user enters or leaves a geographic region
- The region is specified through a CoreLocation `CLRegion`
- You need to get permissions just like `CoreLocation`



SCHEDULING NOTIFICATIONS

```
UNUserNotificationCenter.current().  
    getPendingNotificationRequests {  
        requests in  
        print(requests)  
    }
```

- List all the current notifications

SCHEDULING NOTIFICATIONS

```
let center = UNUserNotificationCenter.current()  
center.removeAllPendingNotificationRequests()
```

- Delete notifications

SCHEDULING NOTIFICATIONS

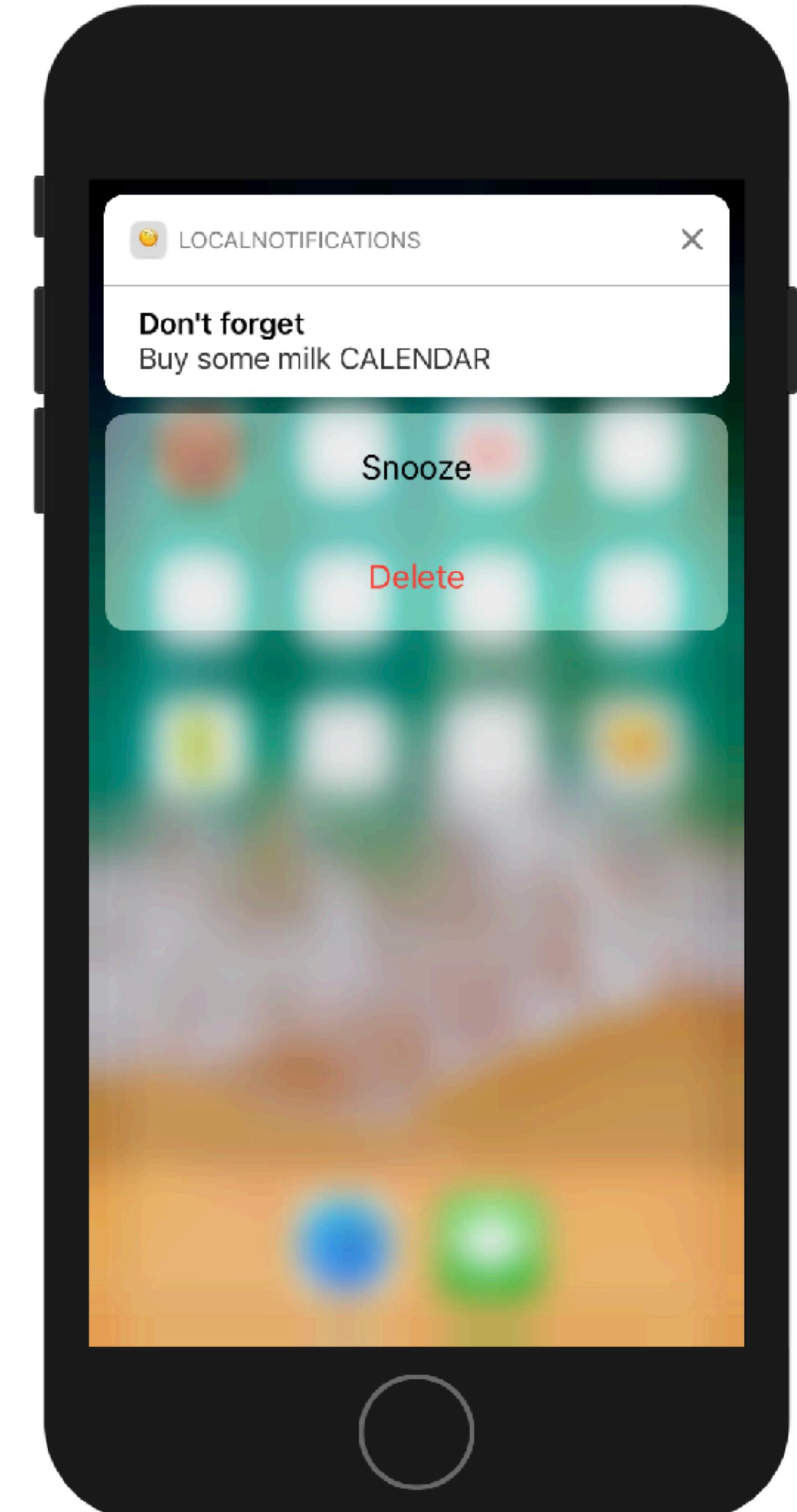
```
@IBAction func tapDelete(_ sender: Any) {  
    listNotification()  
  
    let center = UNUserNotificationCenter.current()  
    center.removeAllPendingNotificationRequests()  
  
    listNotification()  
}
```

- Delete notifications

NOTIFICATION ACTIONS

NOTIFICATION ACTIONS

- Define custom actions for your notifications



NOTIFICATION ACTIONS

```
notificationAction(identifier: "Snooze", title: "S  
notificationAction(identifier: "DeleteAction", tit  
  
cationCategory(identifier: "CalendarCategory",  
actions: [snoozeAction, deleteActi  
intentIdentifiers: [], options: [  
.current().setNotificationCategories([category]
```

NOTIFICATION ACTIONS

```
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,.second], from:  
date)  
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)  
  
// Actions  
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])  
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:  
[.destructive])  
let category = UNNotificationCategory(identifier: "CalendarCategory",  
                                     actions: [snoozeAction,deleteAction],  
                                     intentIdentifiers: [], options: [])  
UNUserNotificationCenter.current().setNotificationCategories([category])  
  
// Add a notification  
let identifier = "Time Notification"  
let request = UNNotificationRequest(identifier: identifier,  
                                    content: content,  
                                    trigger: trigger)
```

NOTIFICATION ACTIONS

```
didReceive response: UNNotificationResponse,  
withCompletionHandler completionHandler: @escaping () -> Void) {  
  
print("AppDelegate: Did Receive: \(response)")  
  
// Determine the user action  
switch response.actionIdentifier {  
case UNNotificationDismissActionIdentifier:  
    print("Dismiss Action")  
case UNNotificationDefaultActionIdentifier:  
    print("Default")  
case "Snooze":  
    print("Snooze")  
case "DeleteAction":  
    print("Delete")  
default:  
    print("Unknown action")  
}
```

Switch on the
actionIdentifier

PUSH NOTIFICATIONS

PUSH NOTIFICATIONS

- Sent from Apple's servers to device
- Apple Push Notification Service (APNS) is the gateway for push notification
 - SSL certificate
 - Provisioning
 - Networking code
 - Device tokens

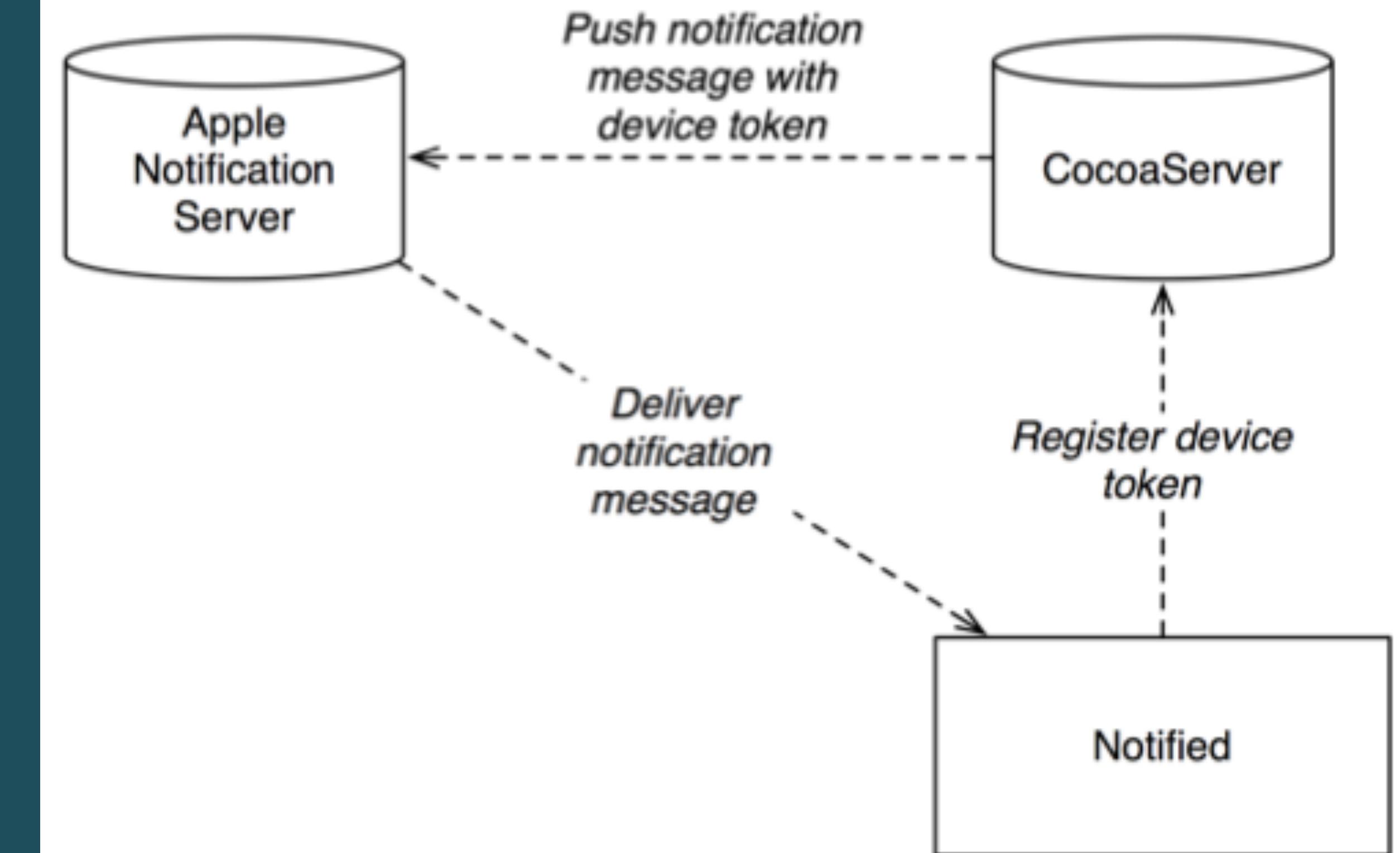
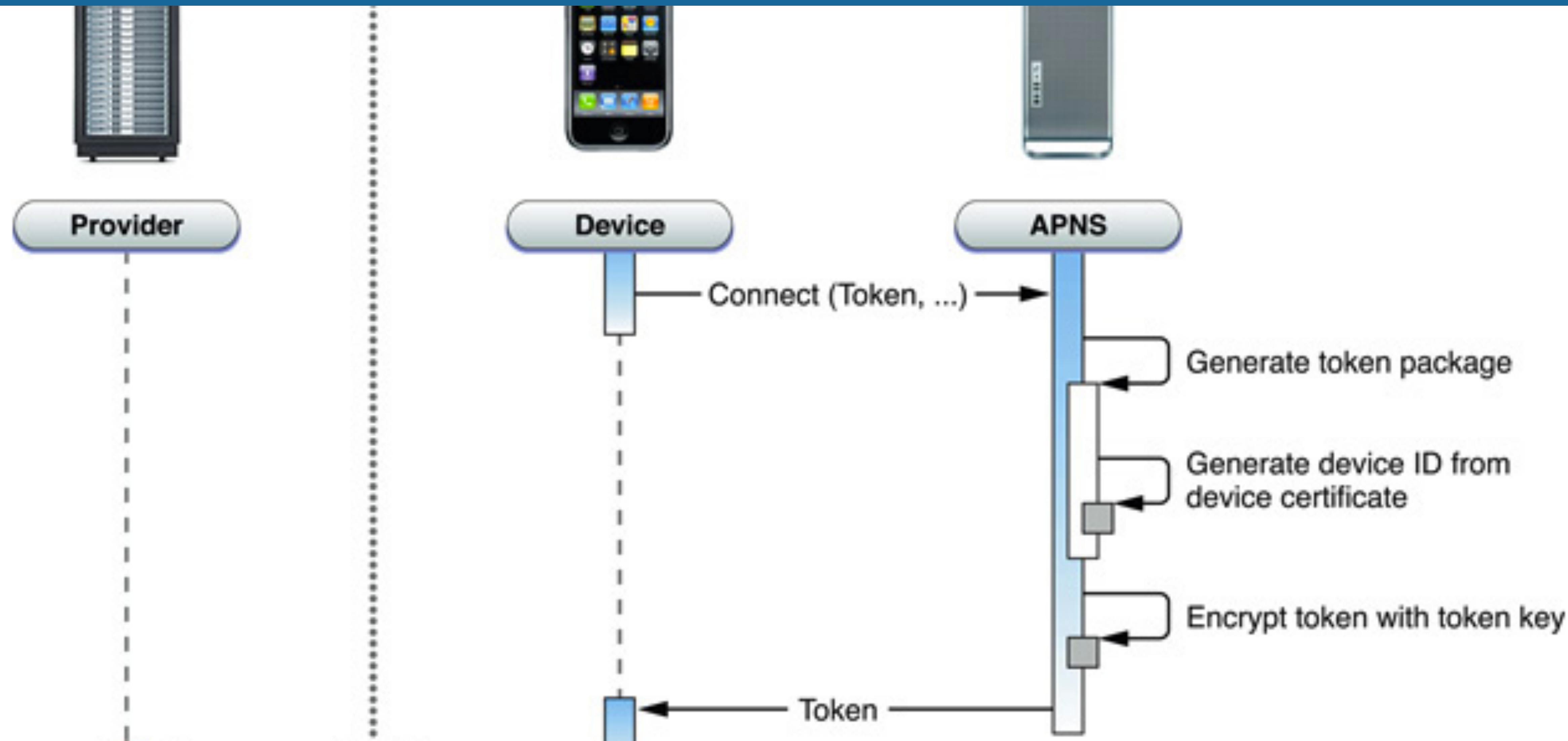


image via Big Nerd Ranch iOS Programming

PUSH NOTIFICATIONS

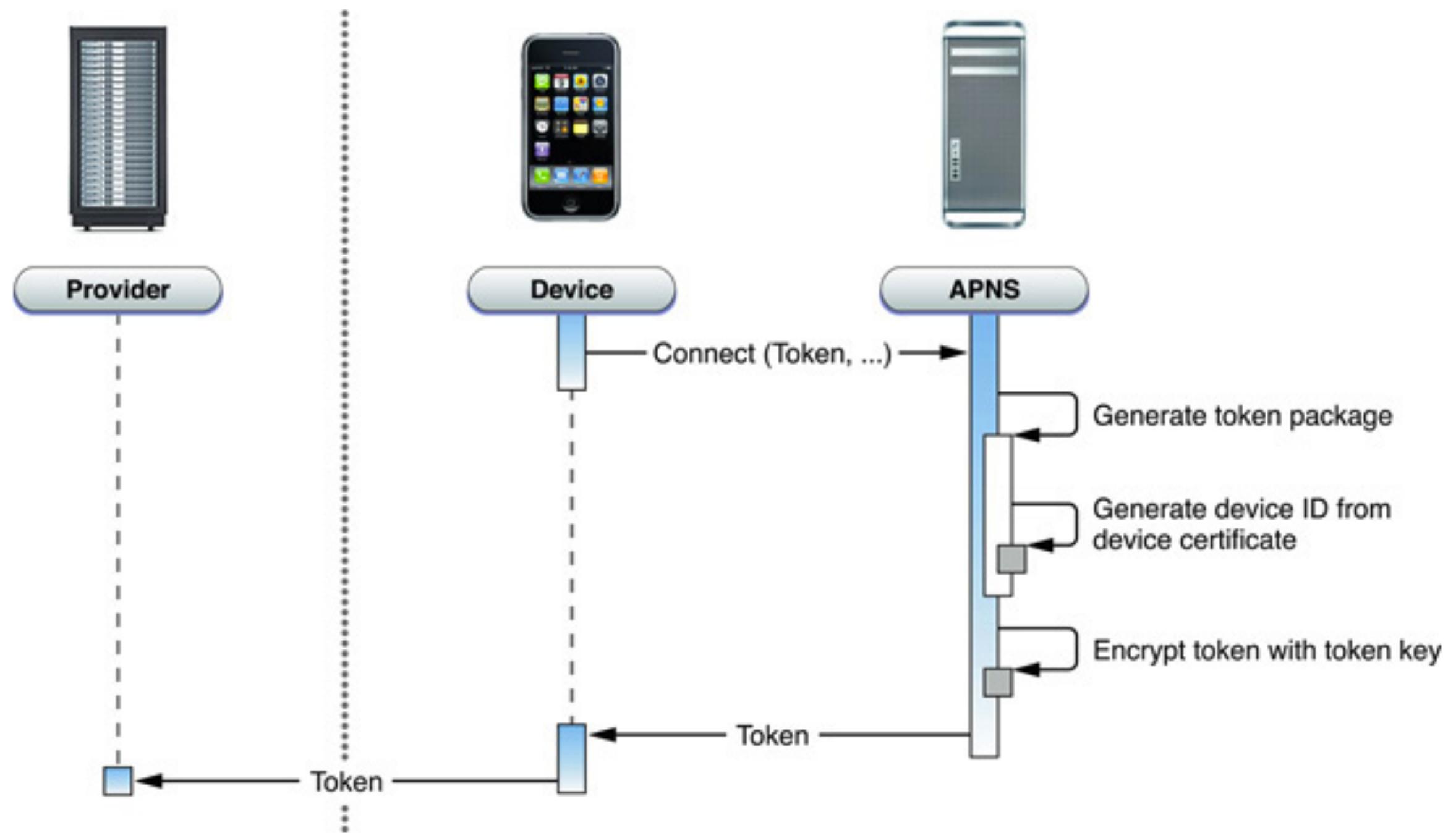
1. An app enables push notification (the user has to confirm that he wishes to receive these notifications)
2. The app receives a “device token”
3. The app sends the device token to your server
4. The server sends a push notification to the Apple Push Notification Service
5. APNS sends the push notification to the user’s device

PUSH NOTIFICATIONS



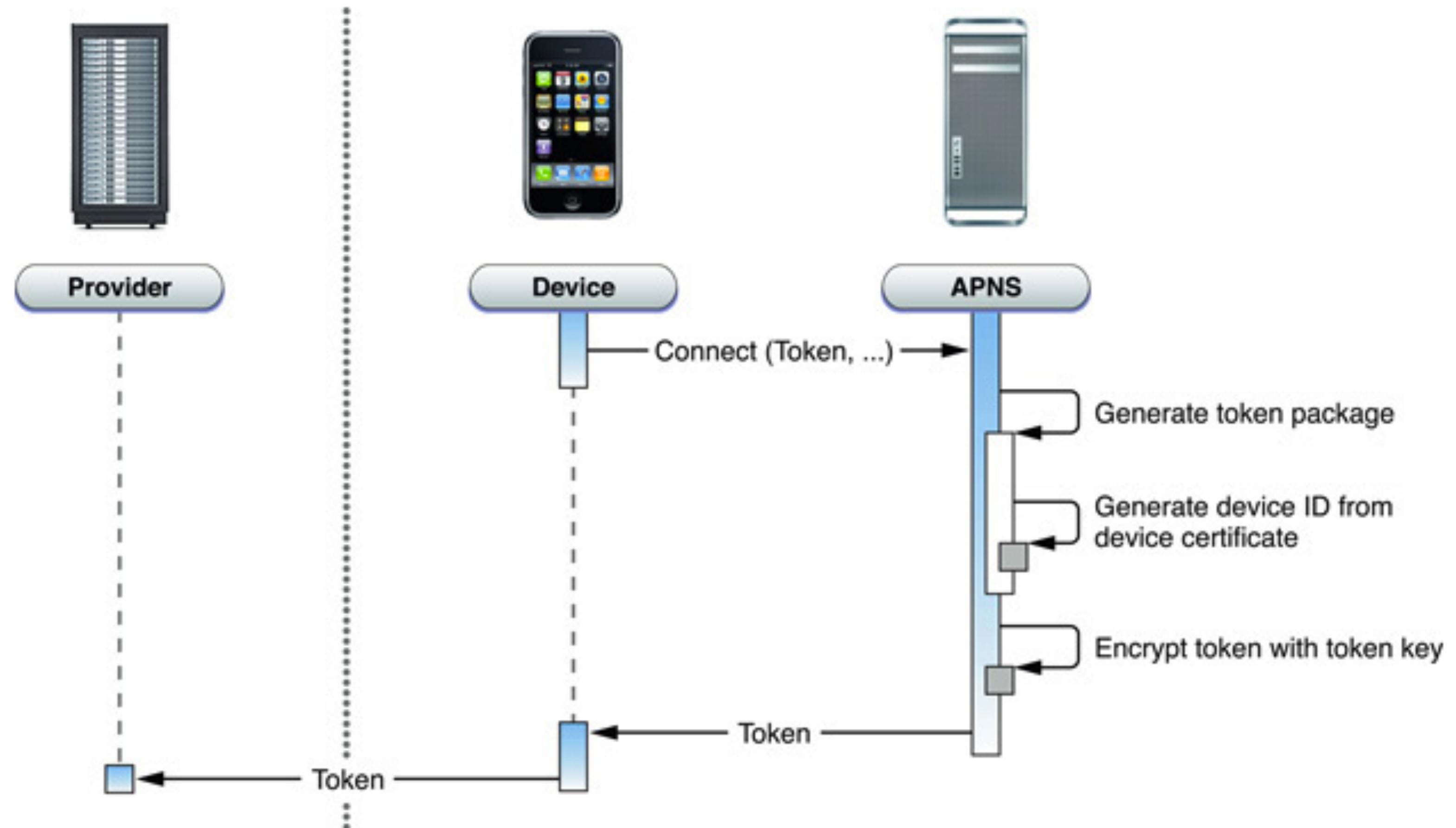
PUSH NOTIFICATION

- Requirements for push notifications
 - iOS device
 - Paid iOS Developer membership
 - Server
 - Ability to run background processes
 - Install SSL certificates
 - Make outgoing TLS connections on non-standard ports



PUSH NOTIFICATION

- Many third party solutions for push notifications
 - Parse
 - Firebase
 - App Engine
 - ...



PUSH NOTIFICATIONS

```
{"aps": {  
    "badge": 100,  
    "alert": "Push Notification Test",  
    "sound": "SoundFile.aif"},  
    "type": "websiteUrl",  
    "data": "http://mySite.com"  
}
```

Manage the badges in your code (not incremental)

Include sound files in your bundle

- Push notification payload
 - JSON dictionary
 - <256 bytes

Pass any key-values in the notification

PUSH NOTIFICATION

- Receiving notification require app delegate methods to be implemented
- Code path is the same for local notification

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: @escaping () -> Void) {
    print(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        // The user dismissed the notification without taking action
    }
    else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        // The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

Urban Airship

..... powering modern mobile

Products Customers Resources

The image shows the Urban Airship platform interface. On the left, there's a screenshot of the web-based dashboard with a sidebar containing 'Push Composer', 'Rich Push Composer', 'New Message', 'Drafts (0)', 'History', and 'Reports'. The main area is titled 'Create your message' with fields for 'Title' (set to '\$1 OFF Any Classic Espresso Drink') and a WYSIWYG editor. A preview of the message is shown, featuring a 'COUPON' button and a large '\$1 OFF' text. On the right, two smartphones are displayed. The one in the foreground shows a notification from 'TIMBER RIDGE COFFEE ROASTERS' with the same '\$1 OFF' offer. The second smartphone shows a list of notifications in its inbox, including messages like 'Getting warm \$1 off an ic...', 'music download of the...', '1 lb. of whole bean cof...', 'punch card is full: free...', 'location opening', and 'punch card is nearly full'.

Rich Push

It just got easier to experiences. Use WYSIWYG editor to surveys and more

Explore Rich Push

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

nomad / houston

Code Issues 27 Pull requests 5 Projects 0 Wiki Insights

Apple Push Notifications; No Dirigible Required <http://nomad-cli.com>

houston notifications ruby cli nomad apns

157 commits 1 branch 19 releases 32 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

mpvosseller authored and dankimio committed on Jan 24 Set default passphrase to nil instead of the empty string. (#158) Latest commit 592bbeb on Jan 24

bin	Set default passphrase to nil instead of the empty string. (#158)	a month ago
lib	Set default passphrase to nil instead of the empty string. (#158)	a month ago
spec	Mutable content: update README, add spec (#137)	a year ago
.gitignore	Update .gitignore	a year ago
Gemfile	Update code style (#134)	a year ago
LICENSE	Updating copyright	3 years ago
README.md	README: use unregistered_devices instead of devices (#156)	5 months ago
Rakefile	Rakefile: add Bundler tasks, add default task	a year ago
houston.gemspec	Update dependencies	a year ago
README.md		



HOUSTON

NOTIFICATIONS WRAP-UP

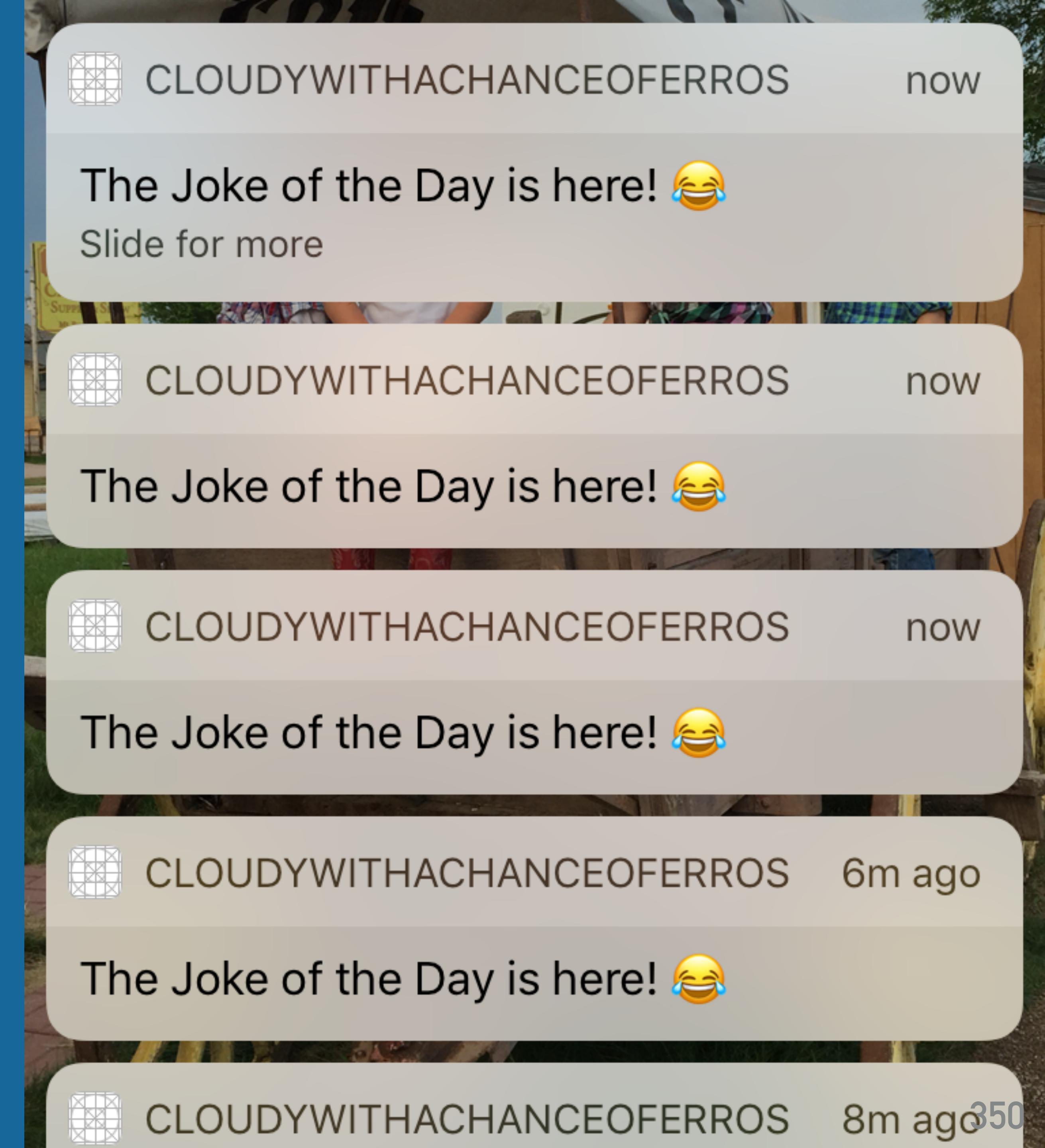
- Notifications keep users engage by opening your app
- Notifications are not guaranteed to be delivered by Apple
- Tips and tricks:
 - Push was important before multi-tasking in iOS4, but many uses can be implemented as local notifications
 - Third-party option for remote notifications
 - Use CloudKit for remote notifications via subscriptions
- Don't abuse notifications...this is one thing that Apple seems to regularly enforce in the App Store Guidelines
 - "Push notifications can not be used for advertising"

CUSTOM NOTIFICATIONS FROM SUBSCRIPTIONS

CUSTOM NOTIFICATIONS

SUBTITLE

- Generic notifications from subscriptions
- This may not be the great user experience we want



CUSTOM NOTIFICATIONS

SUBTITLE

- Generic notifications from subscriptions
- This may not be the great user experience we want

1:08 ↗



TWENTY-NINETEEN-CLOUDKIT

now

😂 Joke of the Day

Urgent Message

Spend more money on in-app purchases...or else.



CUSTOM NOTIFICATIONS



CUSTOM NOTIFICATIONS

Record Types ▾

System Types

Users

Custom Types

alert

joke

joke_of_the_day



New Type



Delete Type

Field Name	Field Type	Indexes
System Fields		
recordName	Reference	None
createdBy	Reference	None
createdAt	Date/Time	None
modifiedBy	Reference	None
modifiedAt	Date/Time	None
changeTag	String	None
Custom Fields		
message	String	None
⊕ Add Field		Edit Indexes

CUSTOM NOTIFICATIONS

```
func registerSilentAlertSubscription() {  
    let uuid: UUID = UUID()  
    let identifier = "\(uuid)-alert"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKSubscription.NotificationInfo()  
    notificationInfo.shouldSendContentAvailable = true  
    notificationInfo.desiredKeys = ["message"]  
  
    // Create the subscription  
    let subscription = CKQuerySubscription(recordType: "alert",  
                                            predicate: NSPredicate(value: true),  
                                            subscriptionID: identifier,  
                                            options: [CKQuerySubscription.Options.firesOnRecordCreation])  
  
    subscription.notificationInfo = notificationInfo  
    CKContainer.default().publicCloudDatabase.save(subscription,  
                                                   completionHandler: {returnRecord, error in  
        if let err = error {  
            print("ALERT: subscription failed \(err.localizedDescription)")  
        } else {  
            print("ALERT: subscription set up")  
        }  
    })  
}
```

MESSAGE IS PASSED ALL THE WAY THROUGH TO THE IOS APP

CUSTOM NOTIFICATIONS



CLOUDKIT HELPER

CUSTOM NOTIFICATIONS

- alert.py
- Prompt user for a message and then adds a new record to the public database
 - Triggers the alert

```
alert.py > ...
1 import sys
2 import json
3
4 import cloudkit_helper as ck
5
6 def add_alert(message):
7     """Create a new alert with a custom message"""
8
9     data = {
10         'operations': [
11             {
12                 'operationType': 'create',
13                 'record': {
14                     'recordType': 'alert',
15                     'fields': {
16                         'message': {
17                             'value': message,
18                         }
19                     }
20                 }
21             }
22
23         print('Posting operation to create quote...')
24         result = ck.cloudkit_request(
25             '/development/public/records/modify', json.dumps(data))
26         print(result['content'])
27
28
29     def main():
30         message = input("What would you like to say to your users?\n")
31         add_alert(message)
32
33
34     if __name__ == '__main__':
35         main()
36
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

2: Python + □ ^ ×

019-autumn/mpcs51033-2019-autumn-code/PyCloudKit-master/alert.py"/Google Drive/g-Teaching/uchicago.cloud/.mpcs51033-2
What would you like to say to your users?
Buy more in app purchases...or else[]

CUSTOM NOTIFICATIONS

```
def add_alert(message):
    """Create a new alert with a custom message"""

    data = {
        'operations': [
            {
                'operationType': 'create',
                'record': {
                    'recordType': 'alert',
                    'fields': {
                        'message': {
                            'value': message,
                        }
                    }
                }
            }
        ]
    }

    print('Posting operation to create quote...')
    result = ck.cloudkit_request(
        '/development/public/records/modify', json.dumps(data))
    print(result['content'])
```

CUSTOM NOTIFICATIONS

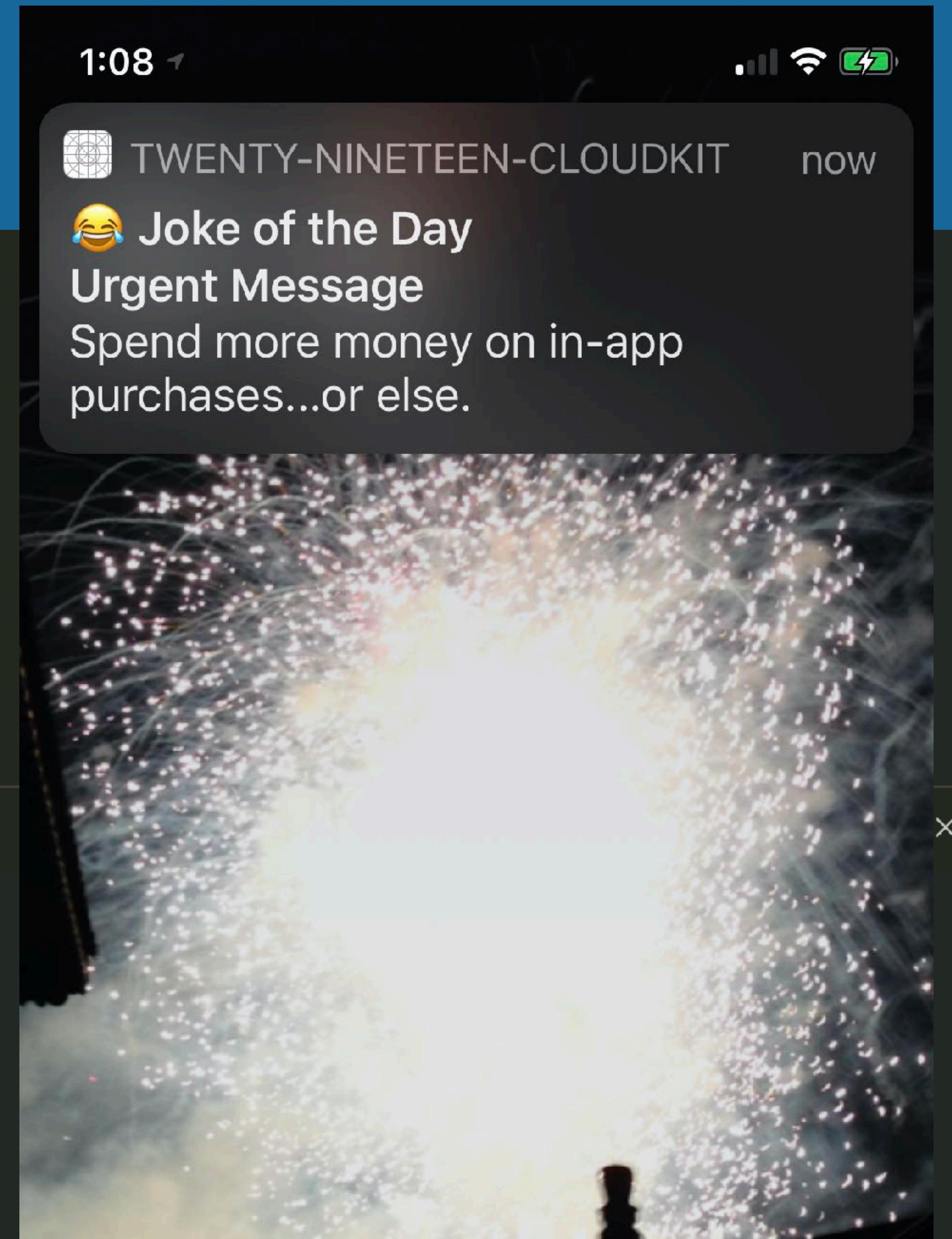
```
12     record : [
13         'recordType': 'alert',
14         'fields': {
15             'message': {
16                 'value': message,
17             }
18         }
19     }
20 }
21
22
23 print('Posting operation to create quote...')
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

3-2

What would you like to say to your users?

Buy more in app purchases...or else



CUSTOM NOTIFICATIONS

What would you like to say to your users?

 You can even send emojis!!!!!!

Posting operation to create quote..

URL: <https://api.apple-cloudkit.com/database/1/iCloud.mobi.uchicago.twenty-n>

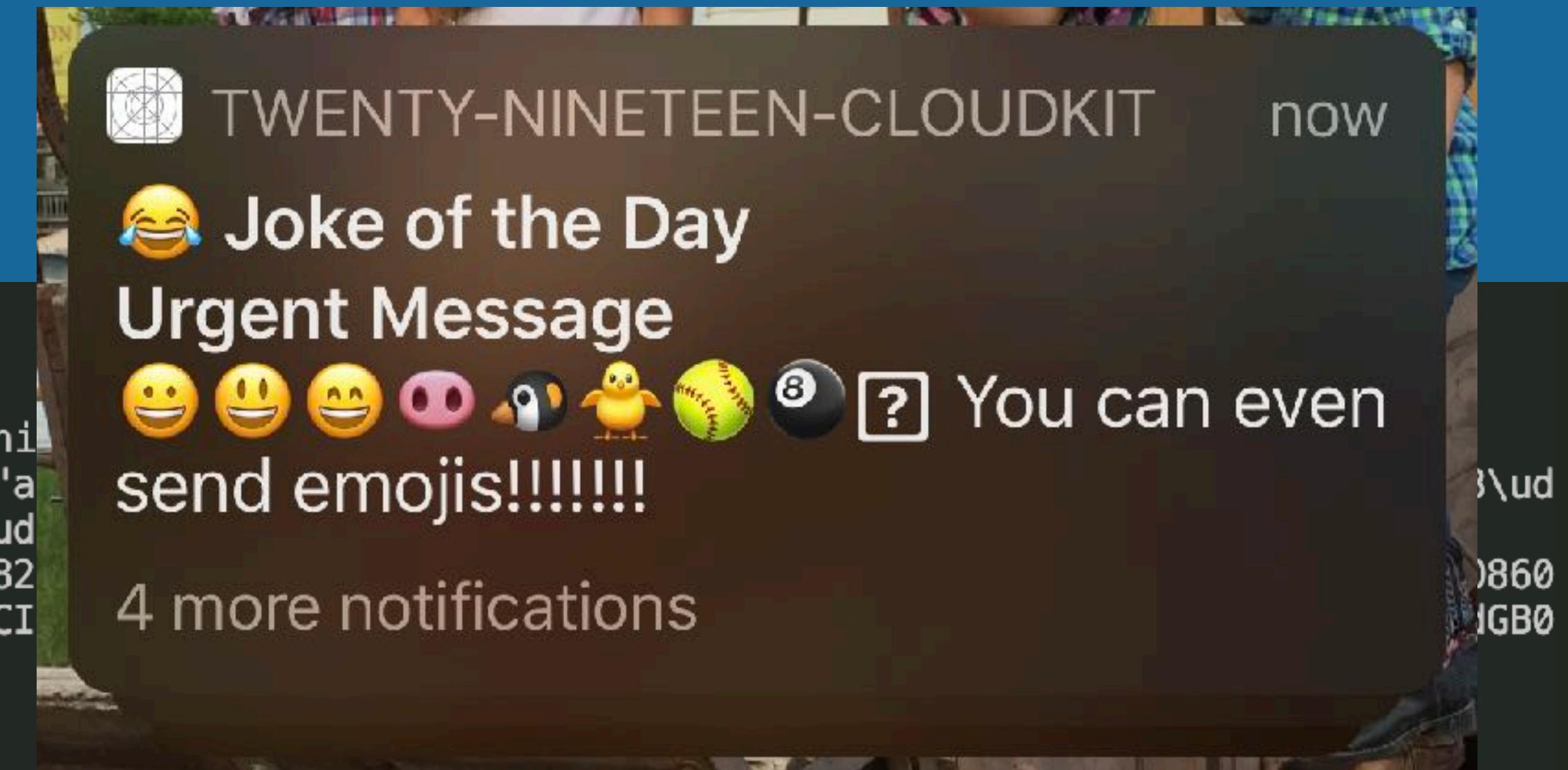
Headers: {'X-Apple-CloudKit-Request-KeyID': '99e04a281f141f4bce822833bd8e8782', 'Date': '2018-11-12T10:29:27Z', 'X-Apple-CloudKit-Request-SignatureV1': 'IMEY63...'}
Body: {

10ate': '2019-11-12T19:29:21.000Z',
B_mJJE6fNLf2UmpUZQsVC9s+' }]

Para
{

```
"records" : [ {
    "recordName" : "0B361BB6-5E9D-4D10-8531-F607A96C480B",
    "recordType" : "alert",
    "fields" : {
        "message" : {
            "value" : "\uD83D\uDE00\uD83D\uDE03\uD83D\uDE04\uD83D\uDE05mojis!!!!!!",
            "type" : "STRING"
        }
    },
    "pluginFields" : { },
    "recordChangeTag" : "k2w91wdk",
    "created" : {
        "timestamp" : 1573586969406,
        "userRecordName" : "_8a76cb86f6390ecf95f548796a270cc8",
        "deviceID" : "12345678901234567890123456789012"
    }
} ]
```

EMOJIS SURVIVE



CUSTOM NOTIFICATIONS

- Create an amazing form that takes some text for the custom message

```
'record': {
    'recordType': 'Alert',
    'fields': {
        'message': {'value': message},
    }
}]}
```

The screenshot shows the CloudKit Dashboard interface. At the top, there's a navigation bar with three colored dots (red, yellow, green), back and forward arrows, and other icons. Below the bar, it says "CloudKit Dashboard". The main area has a title "Alert Notifcation Message" and a large input field. Below the input field is a "Submit" button. At the bottom, there's some red text: "Alert Notifcation Message
".

```
result_modify_jokes = ck.cloudkit_request(
    '/development/public/records/modify',
    json.dumps(new joke of the day data))
def
```

**Alert Notifcation Message
**

IOS APP

CUSTOM NOTIFICATIONS

- Handle the remote notification
 - Read the APNS data
 - Extract the message
 - Create local notification

```
h notifications
we are getting the changed record from cloudkit and then creating a new notification
on(_ application: UIApplication,
    didReceiveRemoteNotification userInfo: [AnyHashable : Any],
    fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) ->

PS notification data
fo)
erInfo["aps"] as! [String: AnyObject]

ing with the content available data
available = aps["content-available"] as! Int
ailable == 1 {
: \("aps")
itInfo = userInfo["ck"] as! [String: AnyObject]
Id = cloudKitInfo["qry"]?["rid"] as! String
= cloudKitInfo["qry"]?["af"] as! [String: AnyObject]

ge indicates that we have a received a silent notification
sage = field["message"] as? String {
tification_from_silent(message: message, recordId: recordId)

Handler(.newData)

Handler(.noData)
```

CUSTOM NOTIFICATIONS

SILENT NOTIFICATION CONTENT

```
[AnyHashable("aps"): {  
    "content-available" = 1;  
}, AnyHashable("ck"): {  
    ce = 2;  
    cid = "iCloud.cloud.uchicago.Cloudy";  
    nid = "2b661f39-5617-4a1c-a889-09cd9eae7a";  
    qry = {  
        af = {  
            message = "Where are my emojis?";  
        };  
        dbs = 2;  
        fo = 1;  
        rid = "A0915CF5-E4E7-4C76-856E-48CE01ADCACE";  
        sid = "14725FE6-9C8F-4BDC-96FF-3815E5FE1E3B-alert";  
        zid = "_defaultZone";  
        zoid = "_defaultOwner";  
    };  
}]
```

FIELD I REQUESTED

RECORD THAT MATCHED
THE QUERY

CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {
    print("Creating local notification")

    // Create notification content
    let content = UNMutableNotificationContent()
    content.title = "😂 Joke of the Day"
    content.subtitle = "Urgent Message"
    content.body = message
    content.sound = UNNotificationSound.default

    // Set up trigger
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)

    // Create the notification request
    let center = UNUserNotificationCenter.current()
    let identifier = recordId
    let request = UNNotificationRequest(identifier: identifier,
                                         content: content, trigger: trigger)
    center.add(request, completionHandler: { (error) in
        if let error = error {
            print("Something went wrong: \(error)")
        }
    })
}
```

CREATE A
NOTIFICATION

CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {  
    print("Creating local notification")  
  
    // Create notification content  
    let content = UNMutableNotificationContent()  
    content.title = "😂 Joke of the Day"  
    content.subtitle = "Urgent Message"  
    content.body = message  
    content.sound = UNNotificationSound.default  
  
    // Set up trigger  
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)  
  
    // Create the notification request  
    let center = UNUserNotificationCenter.current()  
    let identifier = recordId  
    let request = UNNotificationRequest(identifier: identifier,  
                                         content: content, trigger: trigger)  
    center.add(request, completionHandler: { (error) in  
        if let error = error {  
            print("Something went wrong: \(error)")  
        }  
    })
```

CREATE A TRIGGER

TIME, 1 SEC

CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {  
    print("Creating local notification")  
  
    // Create notification content  
    let content = UNMutableNotificationContent()  
    content.title = "😂 Joke of the Day"  
    content.subtitle = "Urgent Message"  
    content.body = message  
    content.sound = UNNotificationSound.default  
  
    // Set up trigger  
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)  
  
    // Create the notification request  
    let center = UNUserNotificationCenter.current()  
    let identifier = recordId  
    let request = UNNotificationRequest(identifier: identifier,  
                                         content: content, trigger: trigger)  
    center.add(request, completionHandler: { (error) in  
        if let error = error {  
            print("Something went wrong: \(error)")  
        }  
    })  
}
```

ADD TO
NOTIFICATION
CENTER

IOS NEWS

CLASS NEWS

- Get on Apple's good side

Latest App Store redesign for iOS shows 800% increase in downloads for Featured apps

Peter Cao - Apr. 20th 2018 11:49 am PT [Twitter](#) [@iPeterCao](#)

IOS 11

APP STORE



CLASS NEWS

- Sign-up for presentations send me a date and idea in Slack



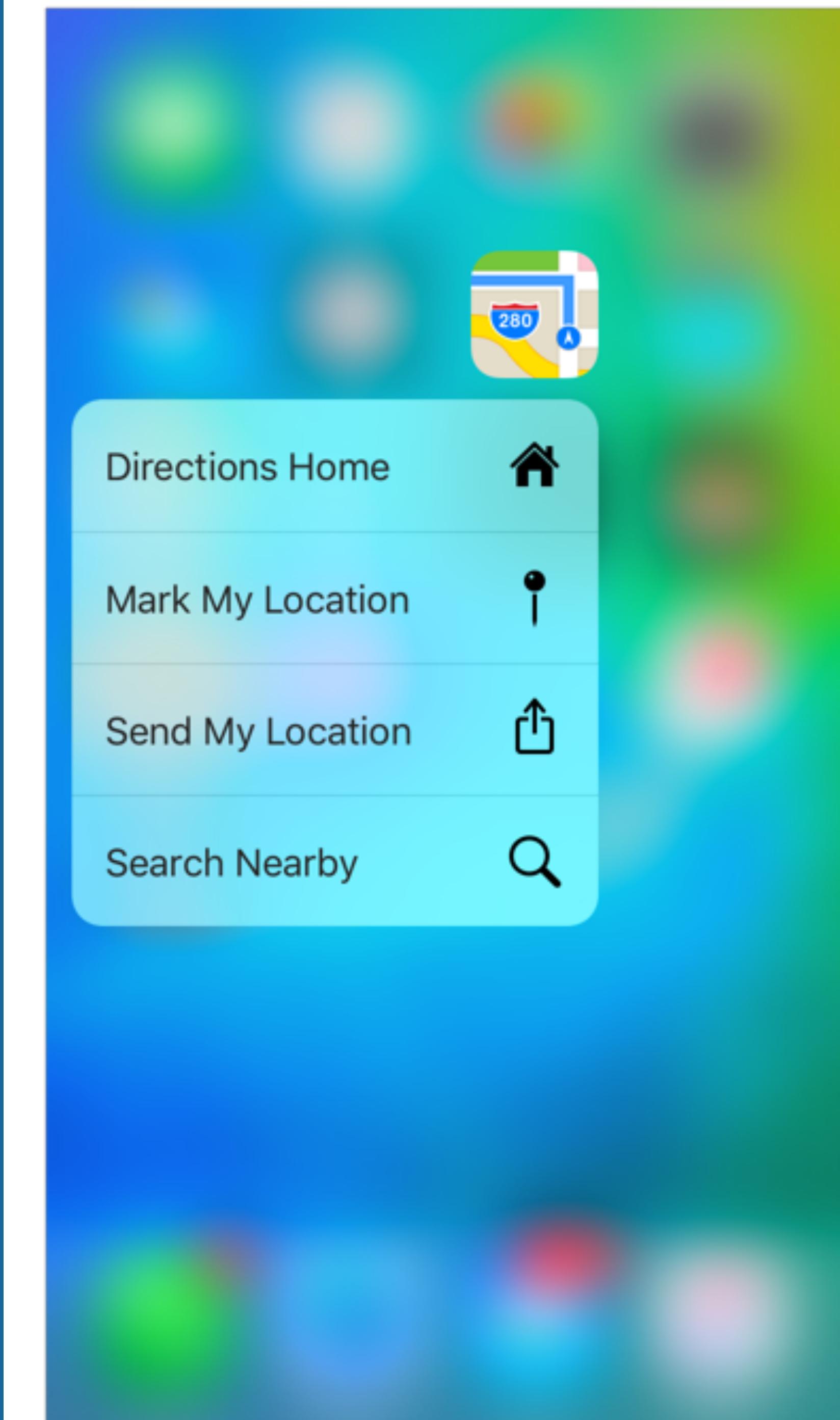
ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

3D TOUCH

3D TOUCH

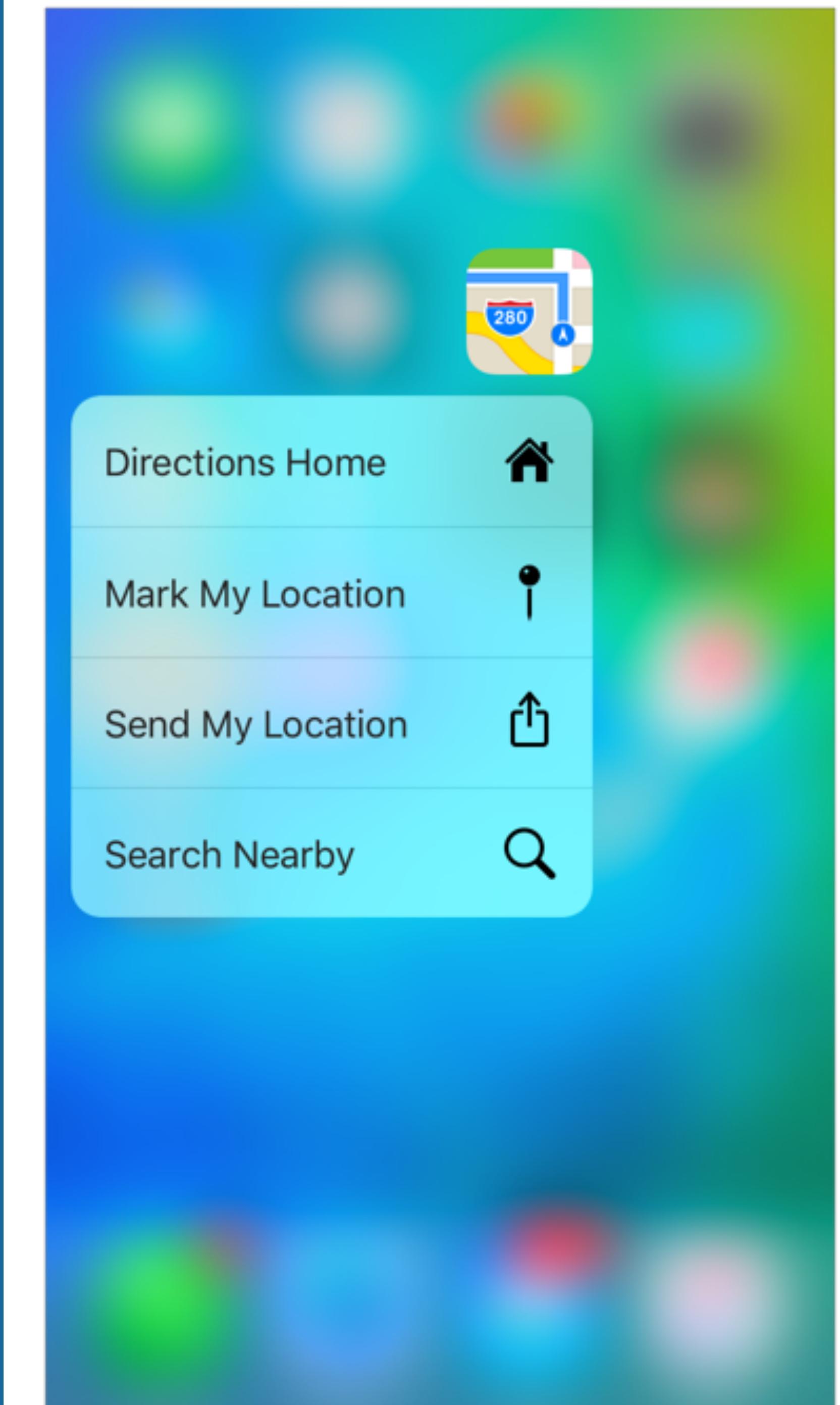
- iOS 9+, new iPhone models add a third dimension to the user interface
 - A user can now press your Home screen icon to immediately access functionality provided by your app
 - Within your app, a user can now press views to see previews of additional content and gain accelerated access to features



3D TOUCH

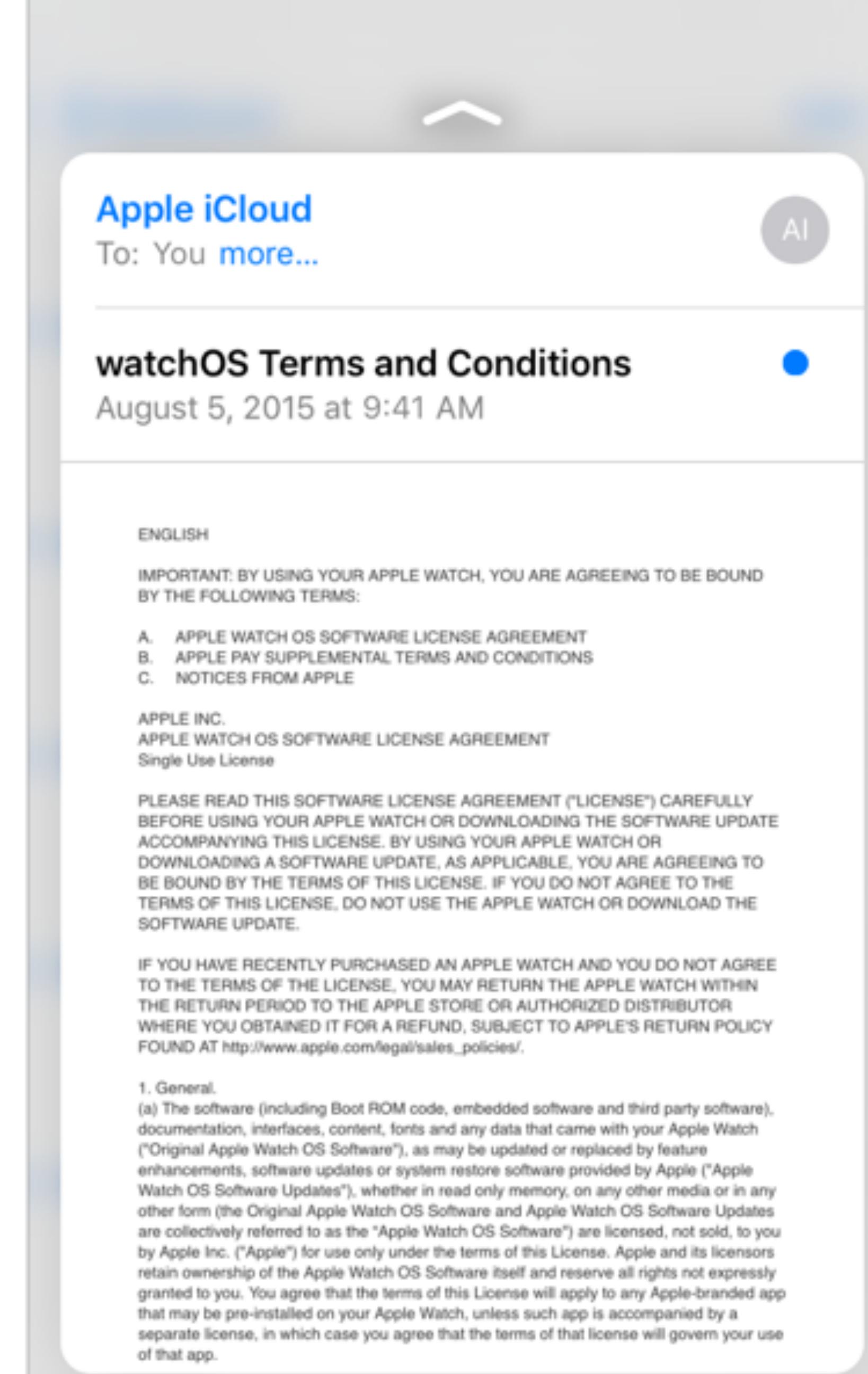
APIS

- The Home screen quick action API is for adding shortcuts to your app icon that anticipate and accelerate a user's interaction with your app
 - Sometimes at the expense of opening your application



3D TOUCH APIS

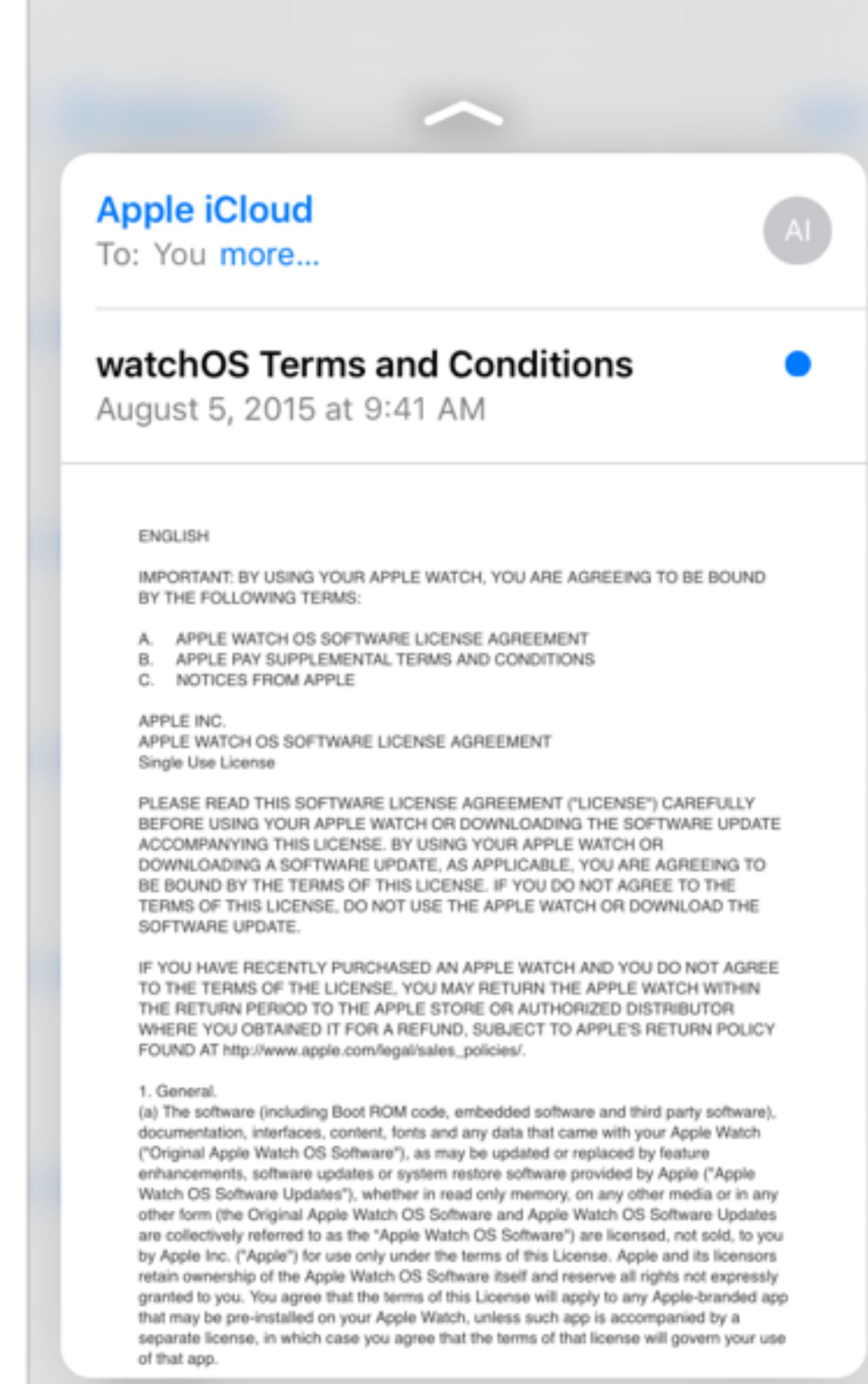
- The Web view peek and pop API lets you enable system-mediated previews of HTML link destinations



3D TOUCH

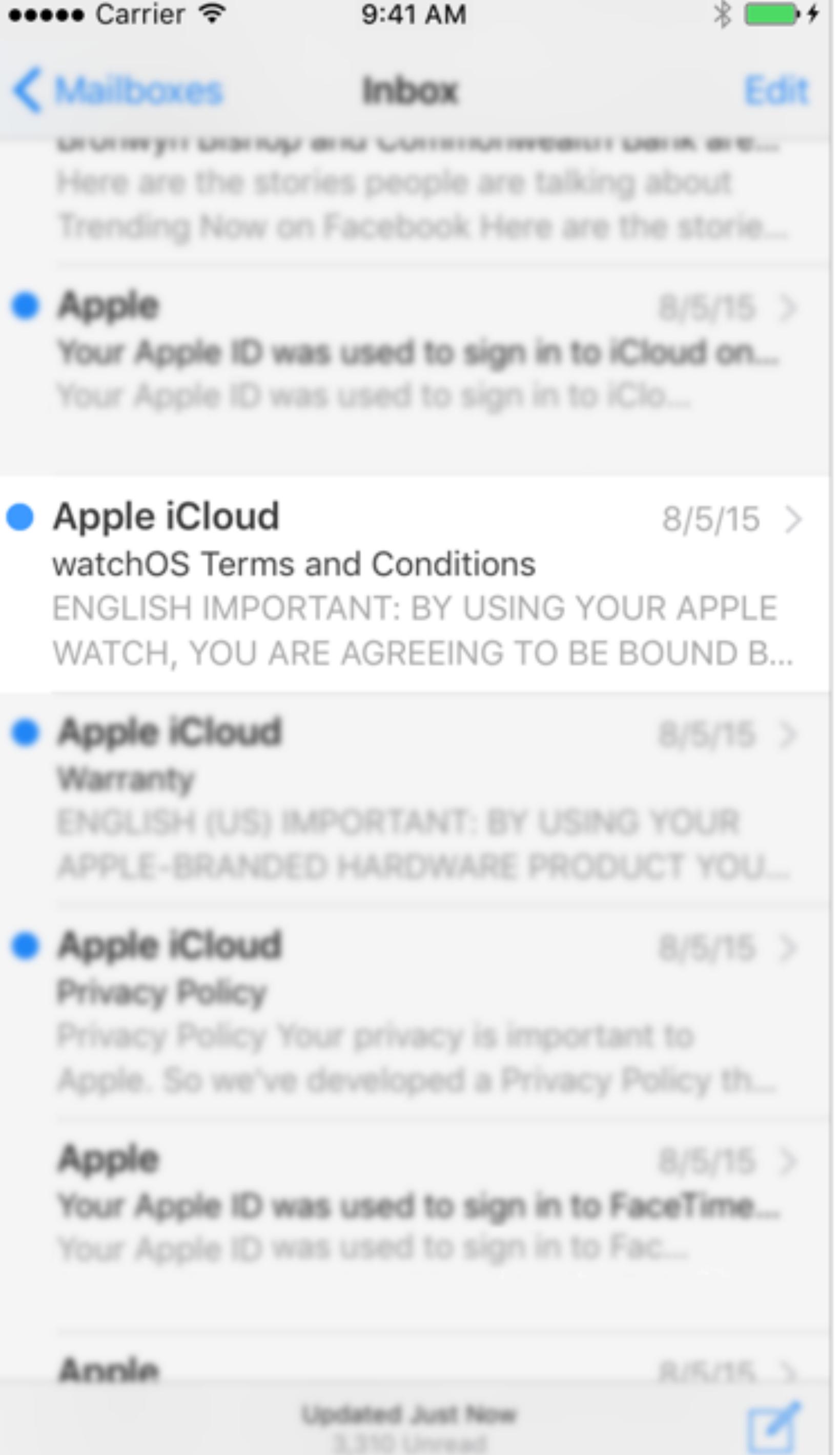
APIS

- Peek and Pop
 - UIKit
 - Web view peek and pop API lets you enable system-mediated previews of HTML link destinations



3D TOUCH APIS

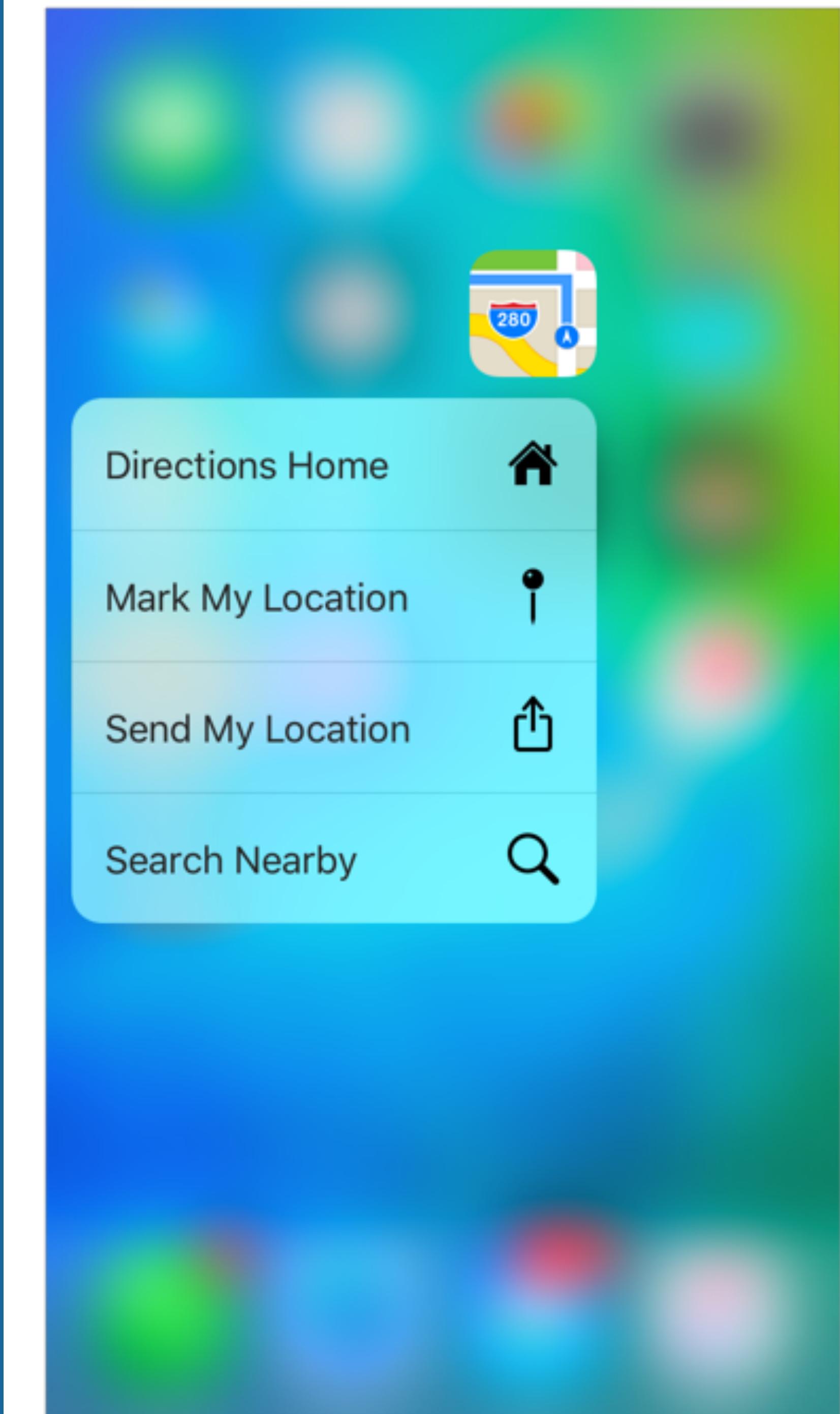
- UITapGestureRecognizer properties let you add customized force-based user interaction to your app



3D TOUCH

APIS

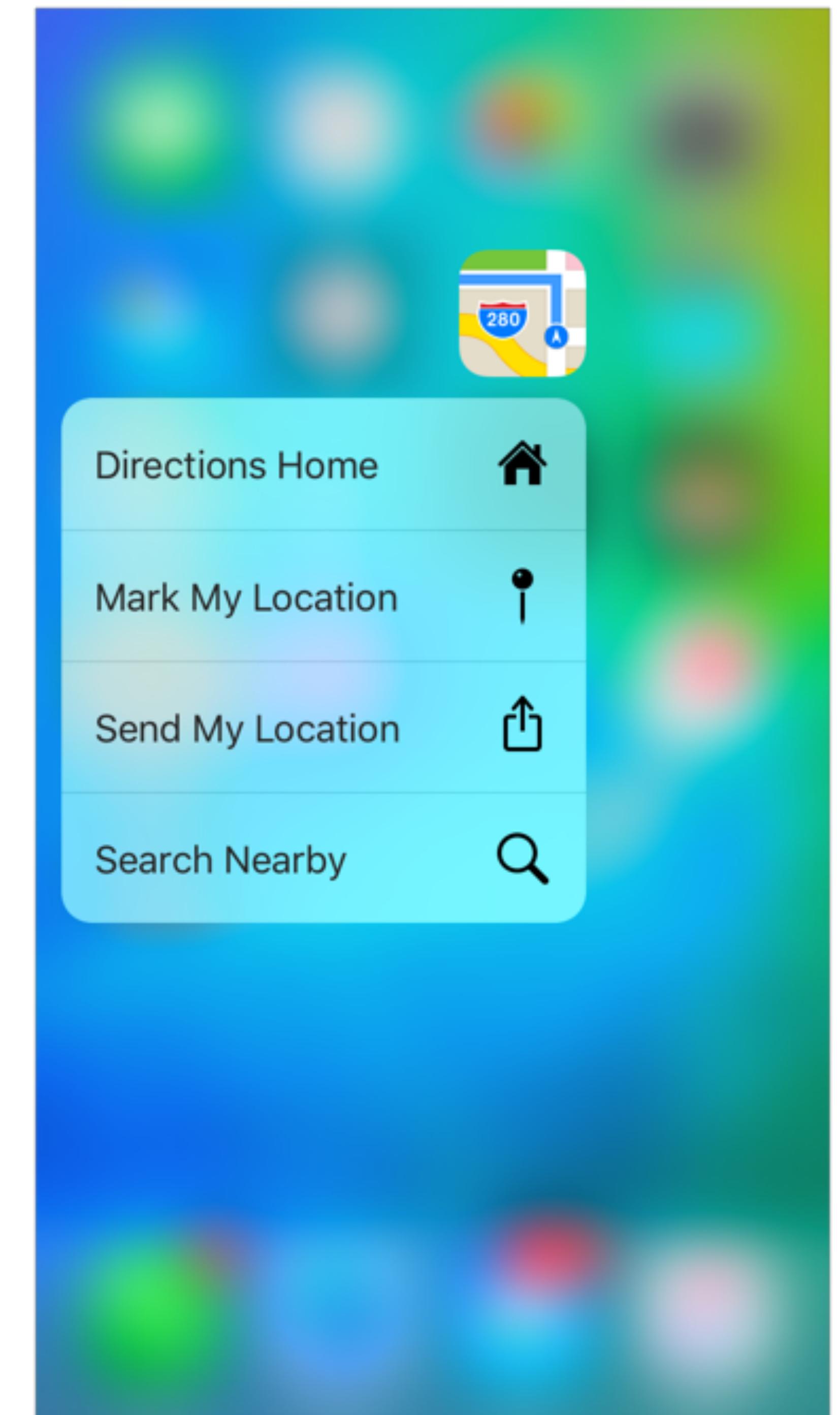
- No matter which of these APIs you adopt, your app must check the availability of 3D Touch at runtime
 - iOS6+ and higher
 - iPad Pro



HOME SCREEN

HOME SCREEN

- Support two types of quick actions (static and dynamic)
- Static quick actions
 - Available to the user immediately upon app installation
 - Define Home screen static quick actions in your app's Info.plist file in the UIApplicationShortcutItems array



HOME SCREEN

Info.plist	Bundle name	String	\$(PRODUCT_NAME)
	Bundle OS Type code	String	APPL
	Bundle versions string, short	String	1.2
	Bundle creator OS Type code	String	????
	Bundle version	String	1
	Application requires iPhone enviro...	Boolean	YES
	► UIApplicationShortcutItems	Array	(2 items)
	▼ Item 0	Dictionary	(5 items)
	UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeSearch
	UIApplicationShortcutItemSubtit...	String	shortcutSubtitle1
	UIApplicationShortcutItemTitle	String	shortcutTitle1
	UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).First
	► UIApplicationShortcutItemUserl...	Dictionary	(1 item)
	▼ Item 1	Dictionary	(5 items)
	UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeShare
	UIApplicationShortcutItemSubtit...	String	shortcutSubtitle2
	UIApplicationShortcutItemTitle	String	shortcutTitle2
	UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).Second
	► UIApplicationShortcutItemUserl...	Dictionary	(1 item)
	Launch screen interface file base...	String	LaunchScreen
	Main storyboard file base name	String	Main
	► Required device capabilities	Array	(1 item)
	► Status bar tinting parameters	Dictionary	(1 item)
	► Supported interface orientations	Array	(3 items)

HOME SCREEN

▼ Item 0	Dictionary	(5 items)	CAN ALSO USE CUSTOM IMAGE
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIcon	
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle1	
UIApplicationShortcutItemTitle	String	shortcutTitle1	
UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).First	
▼ UIApplicationShortcutItemUserl...	Dictionary	(1 item)	
firstShorcutKey1	String	firstShortcutKeyValue1	DATA DICT TO PASS
▼ Item 1	Dictionary	(5 items)	
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeShare	
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle2	
UIApplicationShortcutItemTitle	String	shortcutTitle2	
UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).Second	
▼ UIApplicationShortcutItemUserl...	Dictionary	(1 item)	

- Appearance properties and identifiers

HOME SCREEN

The user activates your application by selecting a shortcut on the home screen. Implement `application(_:willFinishLaunchingWithOptions:)` or `application(_:didFinishLaunchingWithOptions:)` to handle the shortcut in those callbacks and return `false` if possible. In the latter case, the system will not launch your application. This is useful if your application is already launched in the background.

```
func application(_ application: UIApplication, performActionFor shortcutItem: UIApplicationShortcutItem, completionHandler: @escaping (Bool) -> Void) {
    let handledShortCutItem = handleShortCutItem(shortcutItem)
    completionHandler(handledShortCutItem)
```

- Handle the shortcut item by parsing the `shortcutItem.type` to identify

UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).Second
-------------------------------	--------	--------------------------------------

HOME SCREEN

- Custom logic to handle the shortcut
- You have to take it from here

```
func handleShortCutItem(_ shortcutItem: UIApplicationShortcutItem) -> Bool {  
    handled = false  
  
    // Verify that the provided `shortcutItem`'s `type` is one handled by the application.  
    if let identifier = ShortcutIdentifier(fullType: shortcutItem.type) != nil else { return false }  
  
    let shortCutType = shortcutItem.type as String? ?? ""  
  
    switch shortCutType {  
        case ShortcutIdentifier.First.type:  
            // Handle shortcut 1 (static).  
            handled = true  
            break  
        case ShortcutIdentifier.Second.type:  
            // Handle shortcut 2 (static).  
            handled = true  
            break  
        case ShortcutIdentifier.Third.type:  
            // Handle shortcut 3 (dynamic).  
            handled = true  
            break  
        case ShortcutIdentifier.Fourth.type:  
            // Handle shortcut 4 (dynamic).  
            handled = true  
            break  
        default:  
            break  
  
    }  
  
    // Construct an alert using the details of the shortcut used to open the application.  
    let alertController = UIAlertController(title: "Shortcut Handled", message: "\(shortcutItem.localizedTitle ?? "", preferredStyle: .alert)  
", preferredStyle: .alert)  
    let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)  
    alertController.addAction(okAction)  
  
    // Display an alert indicating the shortcut selected from the home screen.  
    if let rootViewController = window?.rootViewController {  
        rootViewController.present(alertController, animated: true, completion: nil)  
    }  
}  
}
```

HOME SCREEN

SUBTITLE

- Dynamic quick actions
 - Available to the user after first launch
 - Define Home screen dynamic quick actions with the `UIApplicationShortcutItem`, `UIMutableApplicationShortcutItem`, and `UIApplicationShortcutIcon` classes
 - Add dynamic quick actions to your app's shared `UIApplication` object using the `shortcutItems` property

Class

UIApplicationShortcutItem

An application shortcut item, also called a Home screen dynamic quick action, specifies a user-initiated action for your app.

Overview

On a device that supports 3D Touch, a user invokes the quick action by pressing your app's icon on the Home screen and then selecting the quick action's title. Your app delegate receives and handles the quick action.

You must specify the characteristics of your `UIApplicationShortcutItem` instances during initialization, before you register them with your app object. The quick actions you've registered with your app object are immutable.

Registering an Array of Dynamic Quick Actions With Your App

To register an array of Home screen dynamic quick actions, set the value of your shared app object's `shortcutItems` property with an `NSArray` instance containing your defined dynamic Home screen quick actions.

Changing Your App's Dynamic Quick Actions

To change your app's Home screen dynamic quick actions, replace your app object's `shortcutItems` array by setting a new value for the property. As a convenience for working with registered quick actions, this class has a mutable subclass, `UIMutableApplicationShortcutItem`. The following code snippet illustrates one way to use the `mutableCopy()` method, along with mutable quick actions, to change the title of a dynamic Home screen quick action:

HOME SCREEN

```
ationShortcutItem(type: "mobi.uchicago.s  
localizedTitle: "的笑容 Shortcut Item"  
localizedSubtitle: "Dynamic Shortcuts  
icon: UIApplicationShortcutIconImage(systemName:  
userInfo: nil)  
ortcutItems = [shortcut]
```

- Create and update the shortcut items

HOME SCREEN

```
from the home screen  
application, performActionFor shortcut  
ol) -> Void) {
```

- Static and dynamic shortcuts are handled the same way
- The `userInfo` dictionary/name should define the logic

HOME SCREEN

APPLICATION SHORTCUT PROJECT

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files like README.md, ApplicationShortcuts, AppDelegate.swift, ShortcutsTableViewController.swift, ShortcutDetailViewController.swift, Info.plist, Assets.xcassets, Base.lproj, and Products.
- Editor:** Displays the `AppDelegate.swift` file content. The code handles application launch options, specifically dealing with application shortcuts. It includes logic to handle different shortcut types (Fourth.type) and construct an alert to inform the user which shortcut was selected. It also overrides the application life cycle methods `applicationDidBecomeActive` and `application(_:didFinishLaunchingWithOptions:)`.
- Identity and Type:** Shows the file is named `AppDelegate.swift`, is a `Default - Swift Source`, and is located relative to the project at `ApplicationShortcuts/AppDelegate.swift`. The full path is listed as `/Users/tabinkowski/Downloads/ApplicationShortcutsUsingUI/ApplicationShortcutItem/ApplicationShortcuts/AppDelegate.swift`.
- On Demand Resource Tags:** A note stating "Only resources are taggable".
- Target Membership:** The file is associated with the target `ApplicationShortcuts`.
- Text Settings:** Configuration for text encoding (Unicode (UTF-8)), line endings (Default - macOS / Unix (LF)), indent using (Spaces), widths (Tab width 2, Indent width 2), and wrap lines.

```
handled = true
break
case ShortcutIdentifier.Fourth.type:
// Handle shortcut 4 (dynamic).
handled = true
break
default:
break
}

// Construct an alert using the details of the shortcut used to open the application.
let alertController = UIAlertController(title: "Shortcut Handled", message: "\"\\"(\shortcutItem.localizedTitle)
\"", preferredStyle: .alert)
let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
alertController.addAction(okAction)

// Display an alert indicating the shortcut selected from the home screen.
window!.rootViewController?.present(alertController, animated: true, completion: nil)

return handled
}

// MARK: - Application Life Cycle

func applicationDidBecomeActive(_ application: UIApplication) { }

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    var shouldPerformAdditionalDelegateHandling = true

    // If a shortcut was launched, display its information and take the appropriate action
    if let shortcutItem = launchOptions?[UIApplicationLaunchOptionsKey.shortcutItem] as? UIApplicationShortcutItem {
        launchedShortcutItem = shortcutItem

        // This will block "performActionForShortcutItem:completionHandler" from being called.
        shouldPerformAdditionalDelegateHandling = false
    }
}
```

HOME SCREEN

DEMO: EXTEND THIS APP SHOW SIMPLE

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows files like 2017-ExtendThisApp.entitlements, AppDelegate.swift, ViewController.swift, Main.storyboard, ForceViewController.swift, Assets.xcassets, LaunchScreen.storyboard, Info.plist, DataKitManager.swift, Constants.swift, CoreDataManager.swift, DataKit.h, Info.plist, Model.xcdatamodeld, and Products (2017-ExtendThisApp.app, DataKit.framework).
- Code Editor:** Displays the AppDelegate.swift code. The code handles application lifecycle events such as applicationDidEnterBackground, applicationWillResignActive, applicationWillEnterForeground, applicationWillBecomeActive, and applicationWillTerminate.
- Search Bar:** Located at the top right, with the placeholder "Search Documentation".
- Documentation View:** A yellow speech bubble-like box containing the text "DEMO: EXTEND THIS APP SHOW SIMPLE".
- Utilities Area:** On the right side, there are sections for "No Quick Help", "Search Documentation", and three items in the "Search bar connected to a search dis..." dropdown: "Fixed Space Bar Button Item", "Flexible Space Bar Button Item", and "View".

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        print(NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).last as Any);

        // Print out the app group (for fun)
        let sharedAppGroup: String = "group.2017-extend-this-app"
        let directory: NSURL = FileManager.default.containerURL(forSecurityApplicationGroupIdentifier: sharedAppGroup)! as NSURL
        print("App Group URL: \(directory)")

        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
    }

    func applicationWillBecomeActive(_ application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.

        // Handle the application shortcuts from the home screen
        func application(_ application: UIApplication, performActionFor shortcutItem: UIApplicationShortcutItem, completionHandler: @escaping
```

HOME SCREEN

DEMO: APPLICATION SHORTCUT PROJECT

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files like README.md, ApplicationShortcuts, AppDelegate.swift, ShortcutsTableViewController.swift, ShortcutDetailViewController.swift, Info.plist, Assets.xcassets, Base.lproj, and Products.
- Editor:** Displays the `AppDelegate.swift` file content. The code handles application launch options, specifically dealing with application shortcuts. It includes logic to handle different shortcut types (Fourth.type) and construct an alert to inform the user which shortcut was selected.
- Run Script:** Shows the build log: "Finished running AppShortcuts on iPhone 7" with 2 warnings.
- Identity and Type:** Shows the file is named `AppDelegate.swift`, is a `Default - Swift Source`, and is located relative to the project at `ApplicationShortcuts/AppDelegate.swift`.
- On Demand Resource Tags:** A note stating "Only resources are taggable".
- Target Membership:** The file is associated with the target `ApplicationShortcuts`.
- Text Settings:** Configuration for text encoding (Unicode (UTF-8)), line endings (Default - macOS / Unix (LF)), indent using (Spaces), widths (Tab width 2, Indent width 2), and wrap lines.

```
handled = true
break
case ShortcutIdentifier.Fourth.type:
// Handle shortcut 4 (dynamic).
handled = true
break
default:
break
}

// Construct an alert using the details of the shortcut used to open the application.
let alertController = UIAlertController(title: "Shortcut Handled", message: "\"\\"(\shortcutItem.localizedTitle)
\"", preferredStyle: .alert)
let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
alertController.addAction(okAction)

// Display an alert indicating the shortcut selected from the home screen.
window!.rootViewController?.present(alertController, animated: true, completion: nil)

return handled
}

// MARK: - Application Life Cycle

func applicationDidBecomeActive(_ application: UIApplication) { }

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    var shouldPerformAdditionalDelegateHandling = true

    // If a shortcut was launched, display its information and take the appropriate action
    if let shortcutItem = launchOptions?[UIApplicationLaunchOptionsKey.shortcutItem] as? UIApplicationShortcutItem
    {

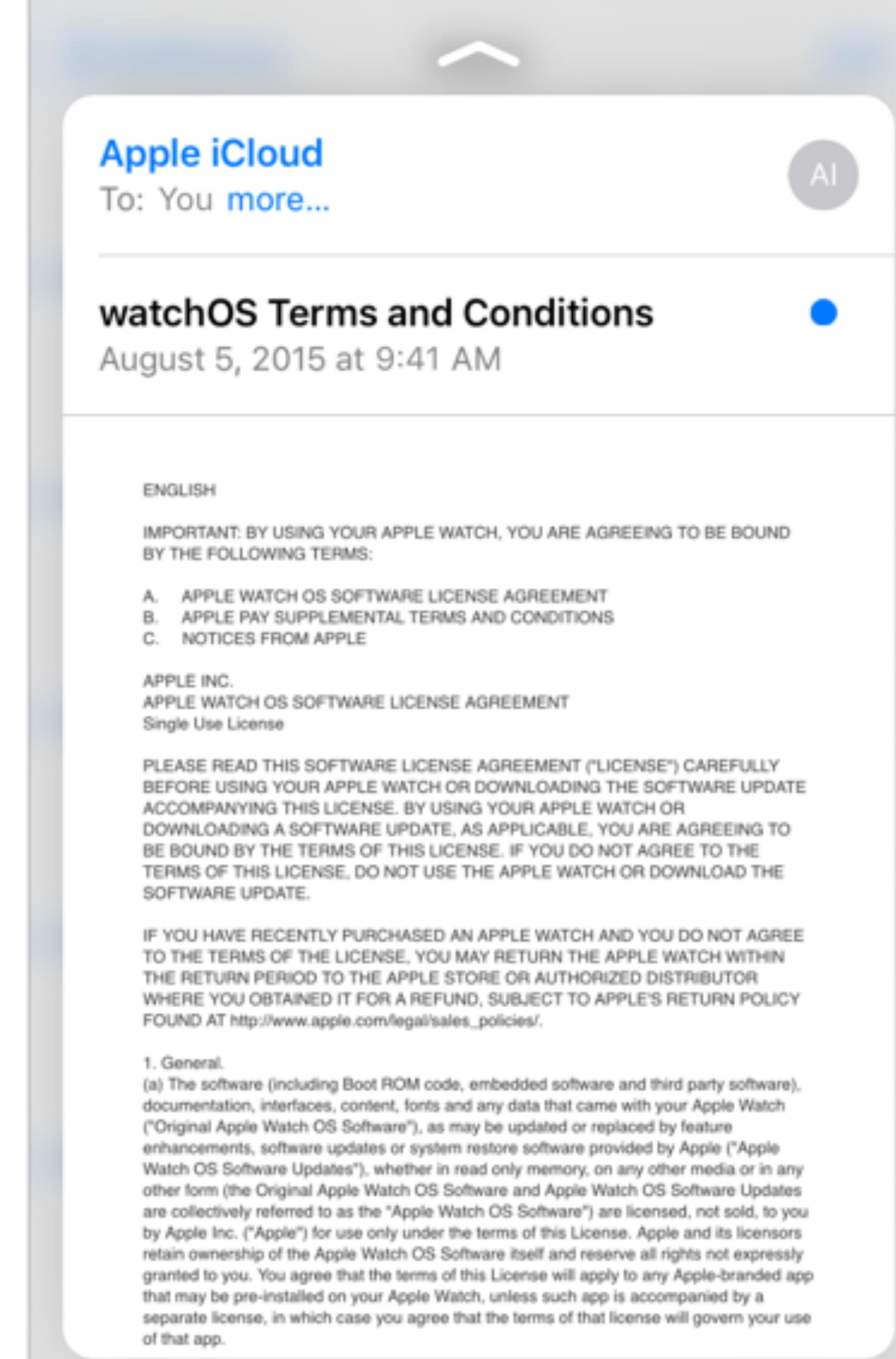
        launchedShortcutItem = shortcutItem

        // This will block "performActionForShortcutItem:completionHandler" from being called.
        shouldPerformAdditionalDelegateHandling = false
    }
}
```

**PEEK AND POP IN
STORYBOARD**

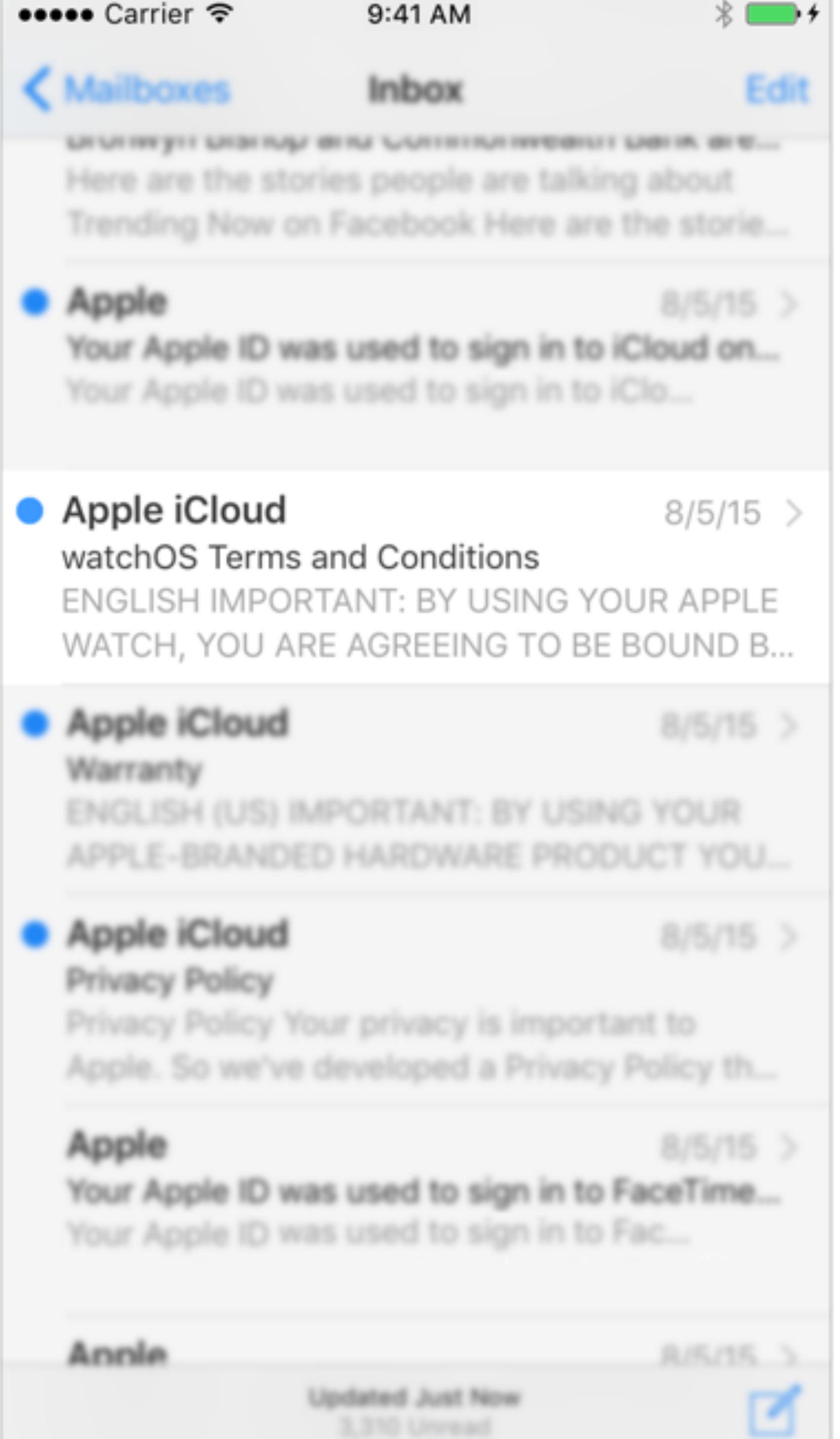
PEEK AND POP

- Peek
 - Preview of the content
 - "preview"
- Pop
 - Go to the content
 - "commit"



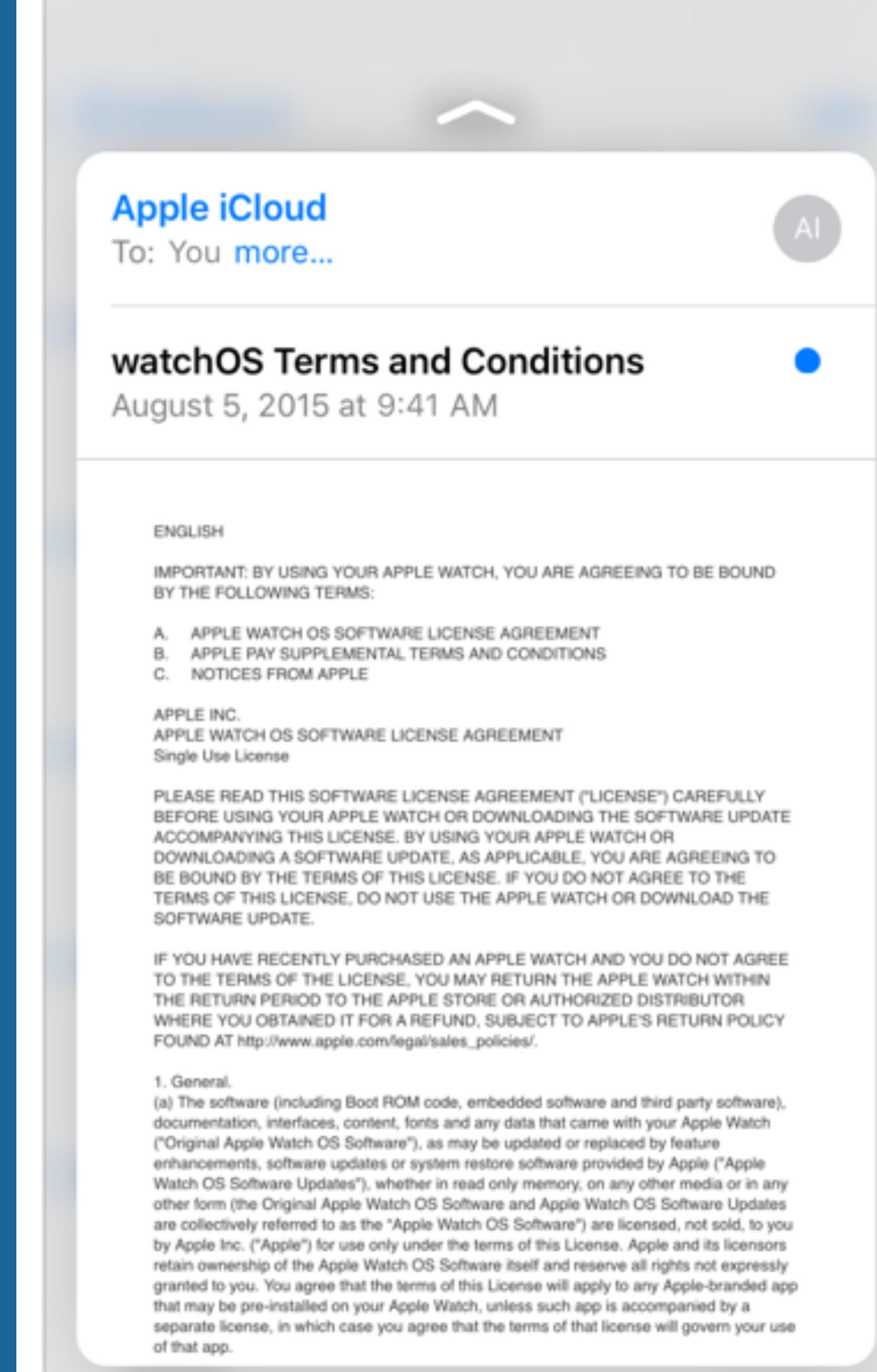
PEEK AND POP

- Indication of peek availability
 - With a light press, surrounding content blurs to tell the user a preview of additional content—the peek—is available



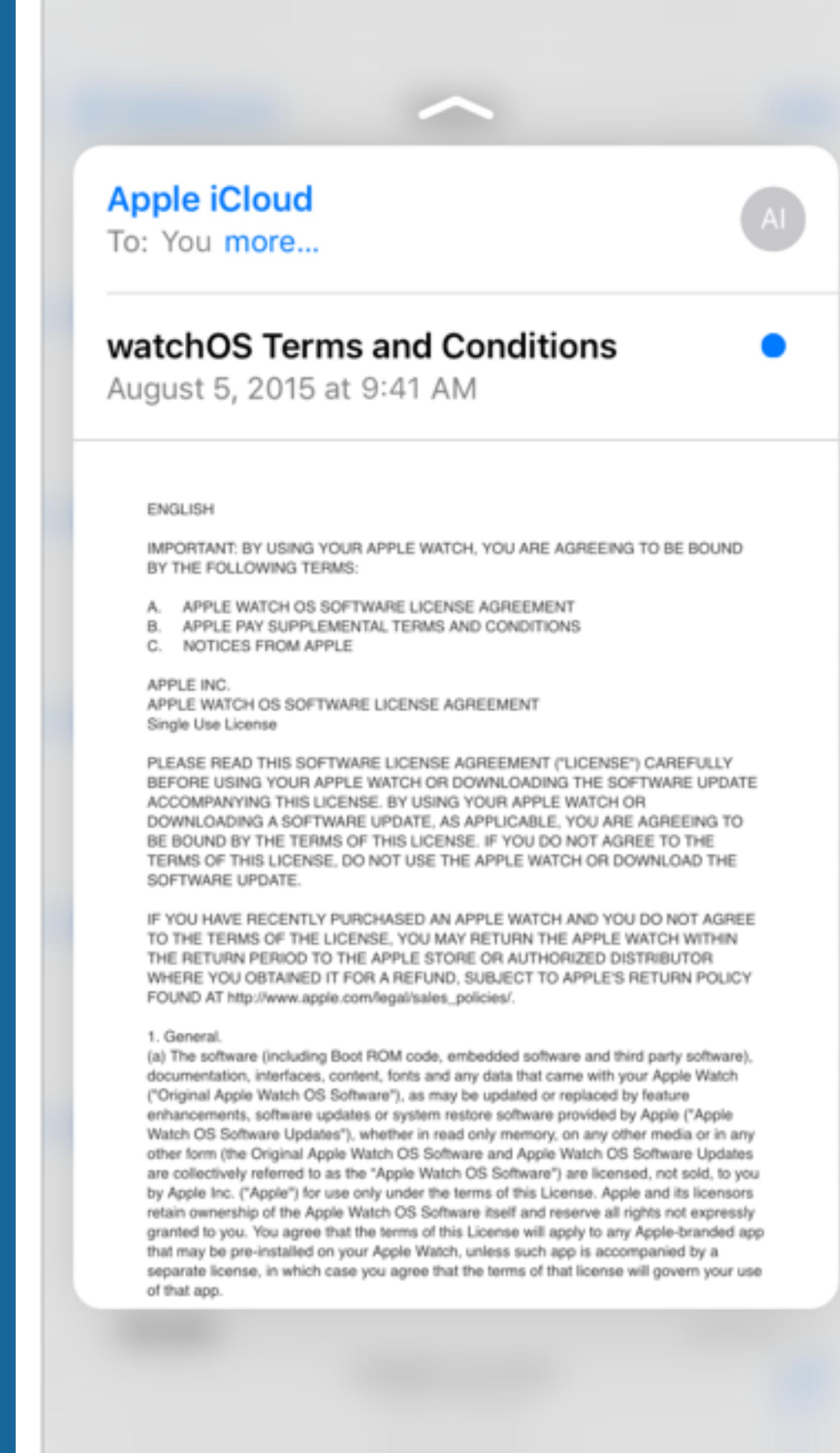
PEEK AND POP

- Peek
 - Press a bit more deeply and the view transitions to show the peek
 - If the user ends the touch at this point, the peek disappears and the app returns to its state before the interaction started



PEEK AND POP

- Pop
 - The user can press deeper still on the peek itself to navigate, using the system-provided pop transition, to the view being previewed as a peek
 - The pop view then fills your app's root view and displays a button to navigate back to where the interaction began



PEEK AND POP

- Peek quick actions
 - If the user swipes the peek upward, the system shows the peek quick actions
 - Each peek quick action is a deep link into your app
 - 

IMPORTANT: BY USING YOUR APPLE WATCH, YOU ARE AGREEING TO BE BOUND BY THE FOLLOWING TERMS:

- A. APPLE WATCH OS SOFTWARE LICENSE AGREEMENT
- B. APPLE PAY SUPPLEMENTAL TERMS AND CONDITIONS
- C. NOTICES FROM APPLE

APPLE INC.
APPLE WATCH OS SOFTWARE LICENSE AGREEMENT
Single Use License

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY BEFORE USING YOUR APPLE WATCH OR DOWNLOADING THE SOFTWARE UPDATE ACCOMPANYING THIS LICENSE. BY USING YOUR APPLE WATCH OR DOWNLOADING A SOFTWARE UPDATE, AS APPLICABLE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, DO NOT USE THE APPLE WATCH OR DOWNLOAD THE SOFTWARE UPDATE.

[Reply](#)

[Forward](#)

[Mark...](#)

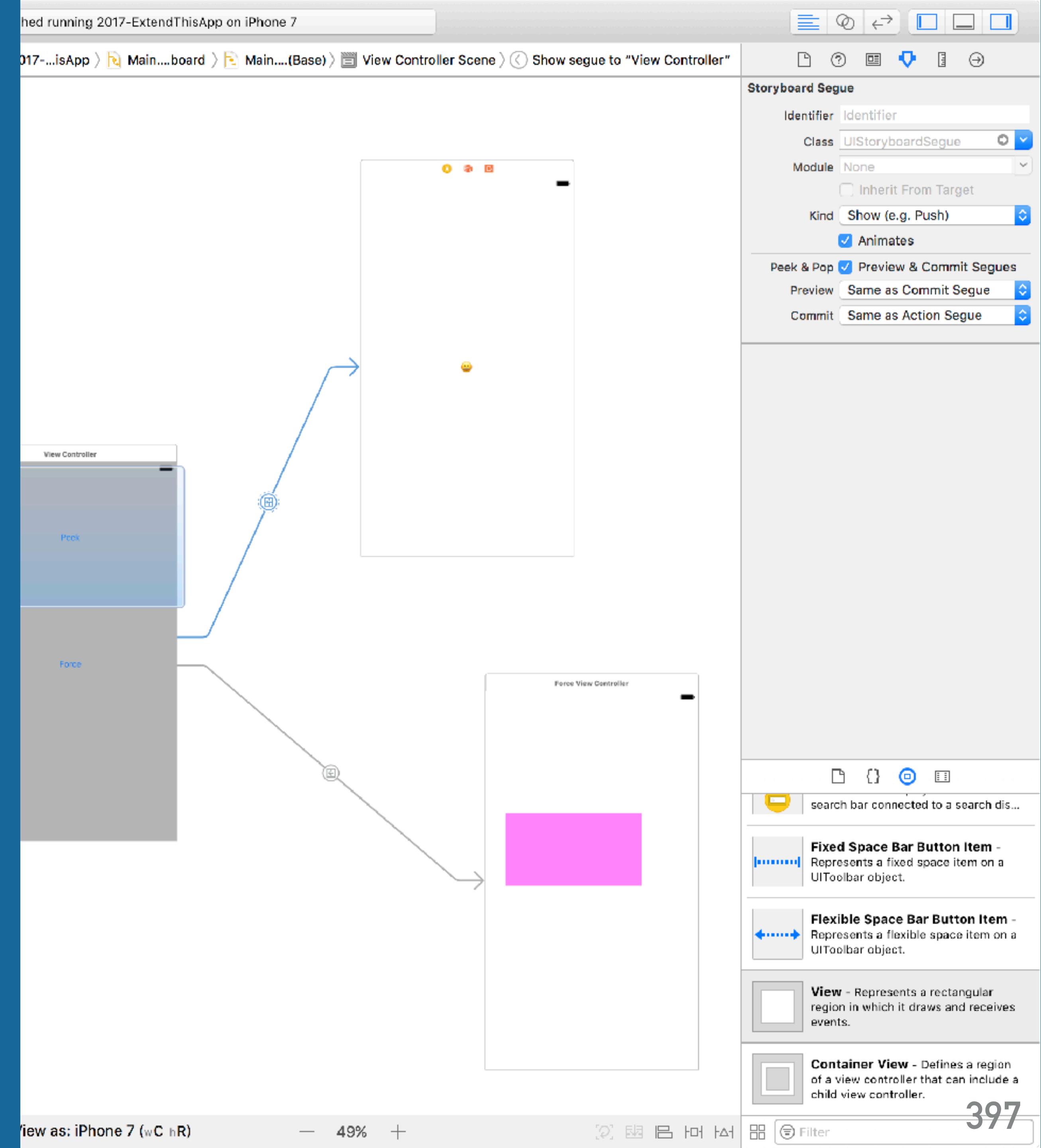
[Notify Me...](#)

[Move Message...](#)

PEEK AND POP

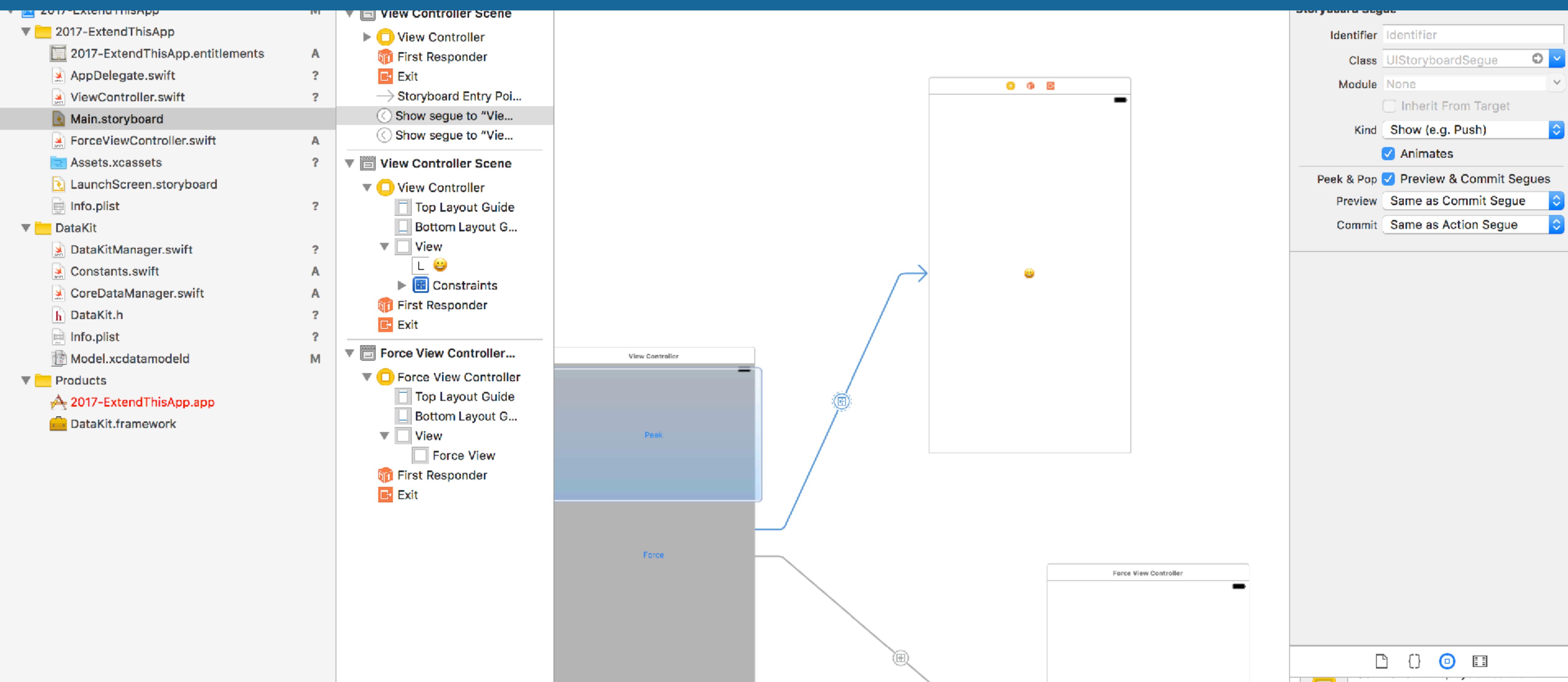
SUBTITLE

- Implement Peek and Pop programmatically or in Storyboard



PEEK AND POP

DEMO: EXTEND THIS APP



FORCE PROPERTIES

FORCE PROPERTIES



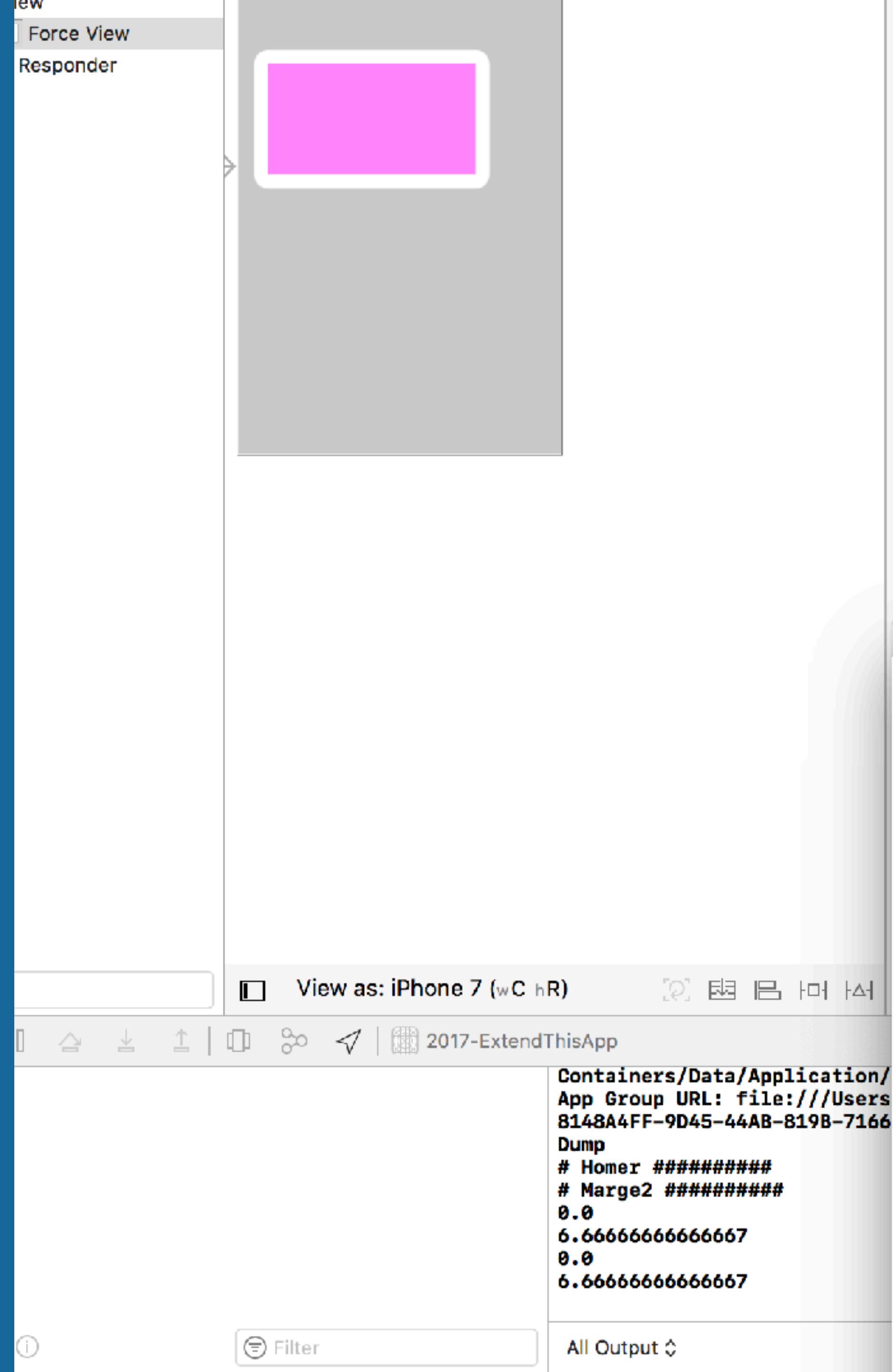
FORCE PROPERTIES

SUBTITLE

- UITapGestureRecognizer class has two new properties to support custom implementation of 3D Touch in your app
 - force
 - maximumPossibleForce
- Detect and respond to touch pressure in the UIEvent objects your app receives

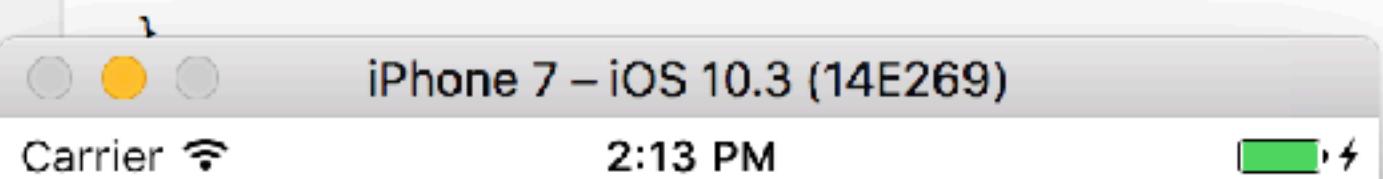


FORCE PROPERTIES



```
class ForceViewController: UIViewController {  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        // Do any additional setup after loading the view.  
    }  
  
    override func didReceiveMemoryWarning() {  
        super.didReceiveMemoryWarning()  
        // Dispose of any resources that can be recreated.  
    }  
  
}
```

```
class ForceView: UIView {  
  
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {  
        if let touch = touches.first {  
            print(touch.force)  
            print(touch.maximumPossibleForce)  
        }  
    }  
}
```



4AB9-8A66-C

BREAK TIME



ASSIGNMENT 3

BUILD A BETTER APP STORE

SUBTITLE

● Search iTunes API

- *Step 1. Query iTunes API*
 - Return JSON
 - *Step 2. Loop through JSON*
 - Create new instance 'Application' Entity
 - Copy the important data from JSON to your model
 - Run JSON['description'] through NSLinguistic tagger to extract the key terms you choose

Assignment

Discoverability in the App Store is one of the biggest problems facing developers (that and Core Data and iCloud syncing). In this assignment, we will create a subset of the App Store that is focused on games and implement a search engine using core iOS technologies. In addition, you will allow users to create a wishlist of games they are interested in. They will be able to add games to the wishlist from within the app and from a share extension from the App Store application. They will also be able to view their wishlist from a Today extension.

Application Overview

The application will come bundled with a Core Data store representing app data that is taken from the iTunes search API. You should select a subset of approximately 100 apps from the Game category of the AppStore. The user will launch the app and be presented with a default list of application.

A search bar will be used to query the local Core Data store of games. The results should be returned presented in a stylized table view.

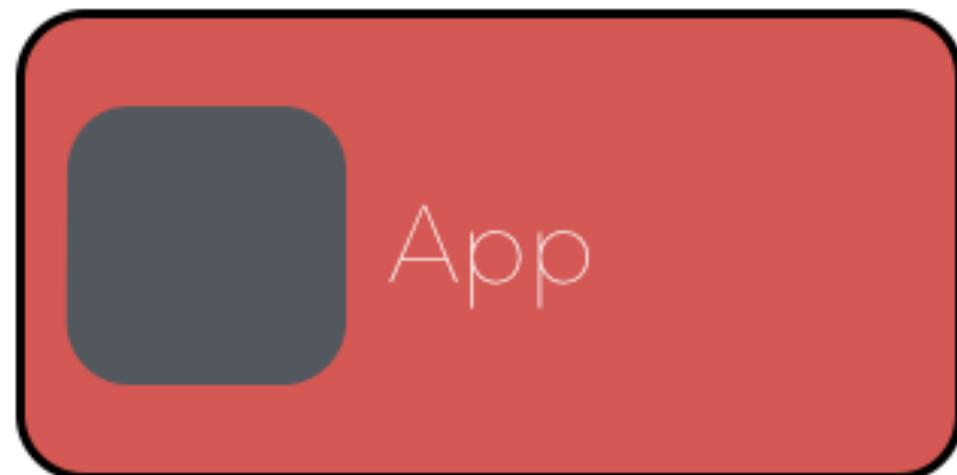
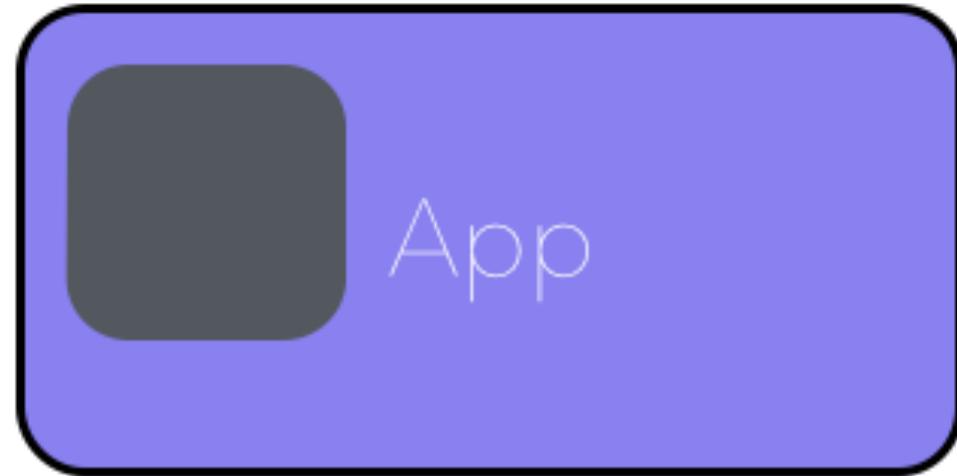
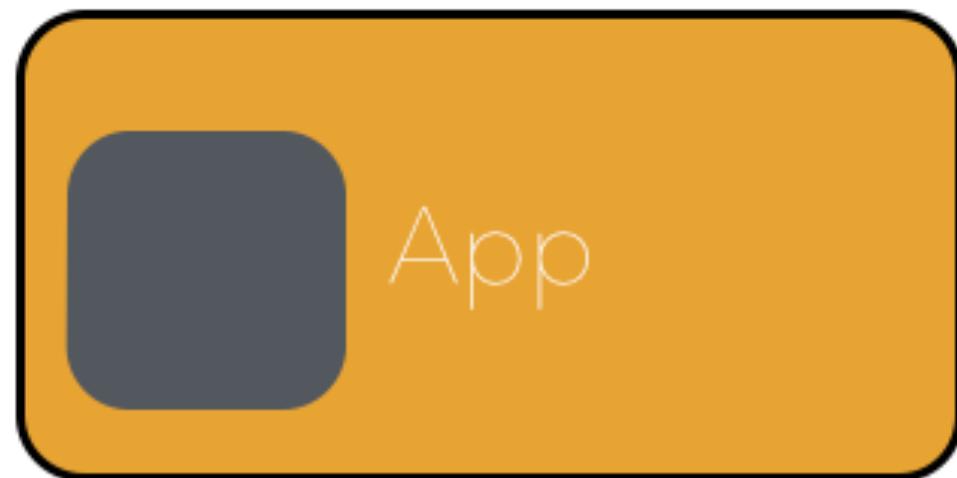
Tip: Look at the layout of the App Store for a guideline on what information to display for each application. See Figure 1.

BUILD A BETTER APP STORE

Search



Search Results

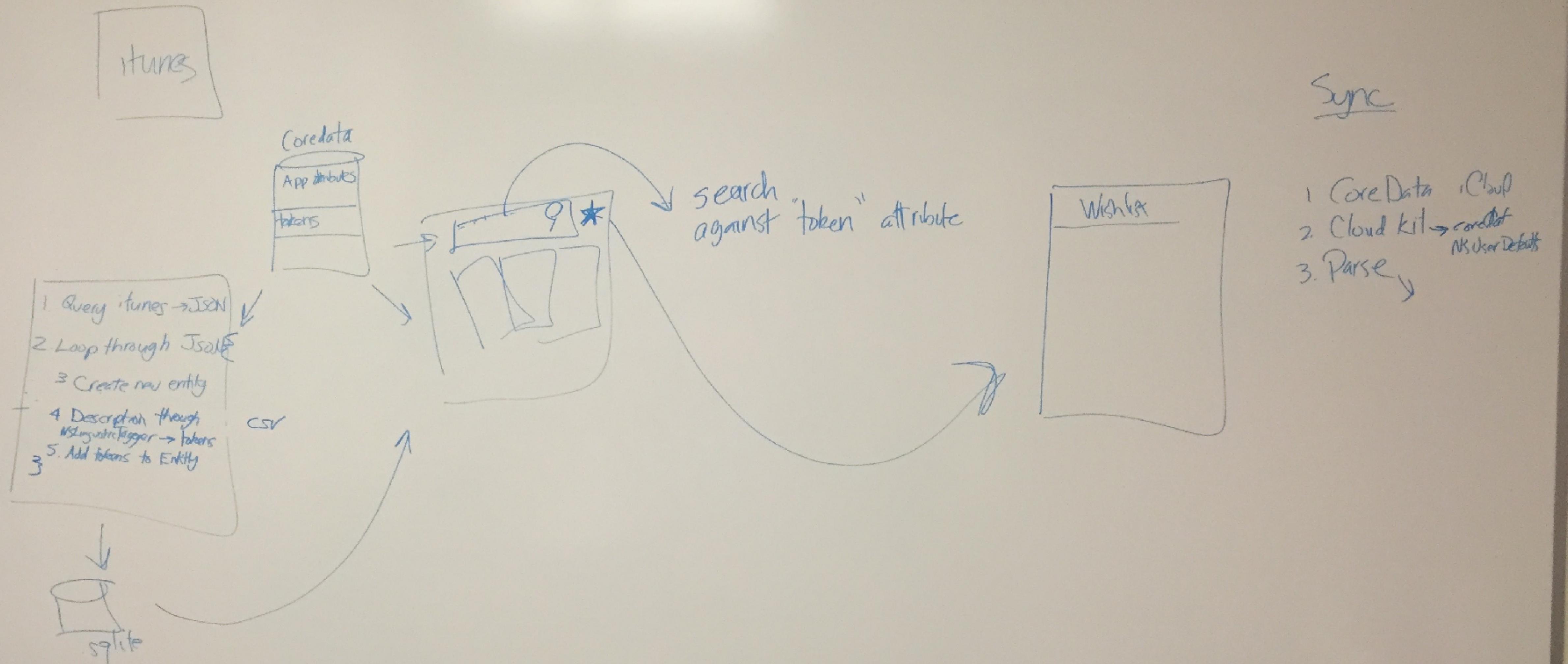


Flappy Twitter
Insta-book

Screen shots



APP ARCHITECTURE



APP ARCHITECTURE

- This Week
 - iTunes API query and parsing
 - NSLinguisticTagger
 - Core Data model
 - Load Core Data into table
 - Style table cells (downloading and showing screenshots)
 - Live search
 - SKStoreProductViewController
 - Networking and Core Data Stack into Embedded Framework



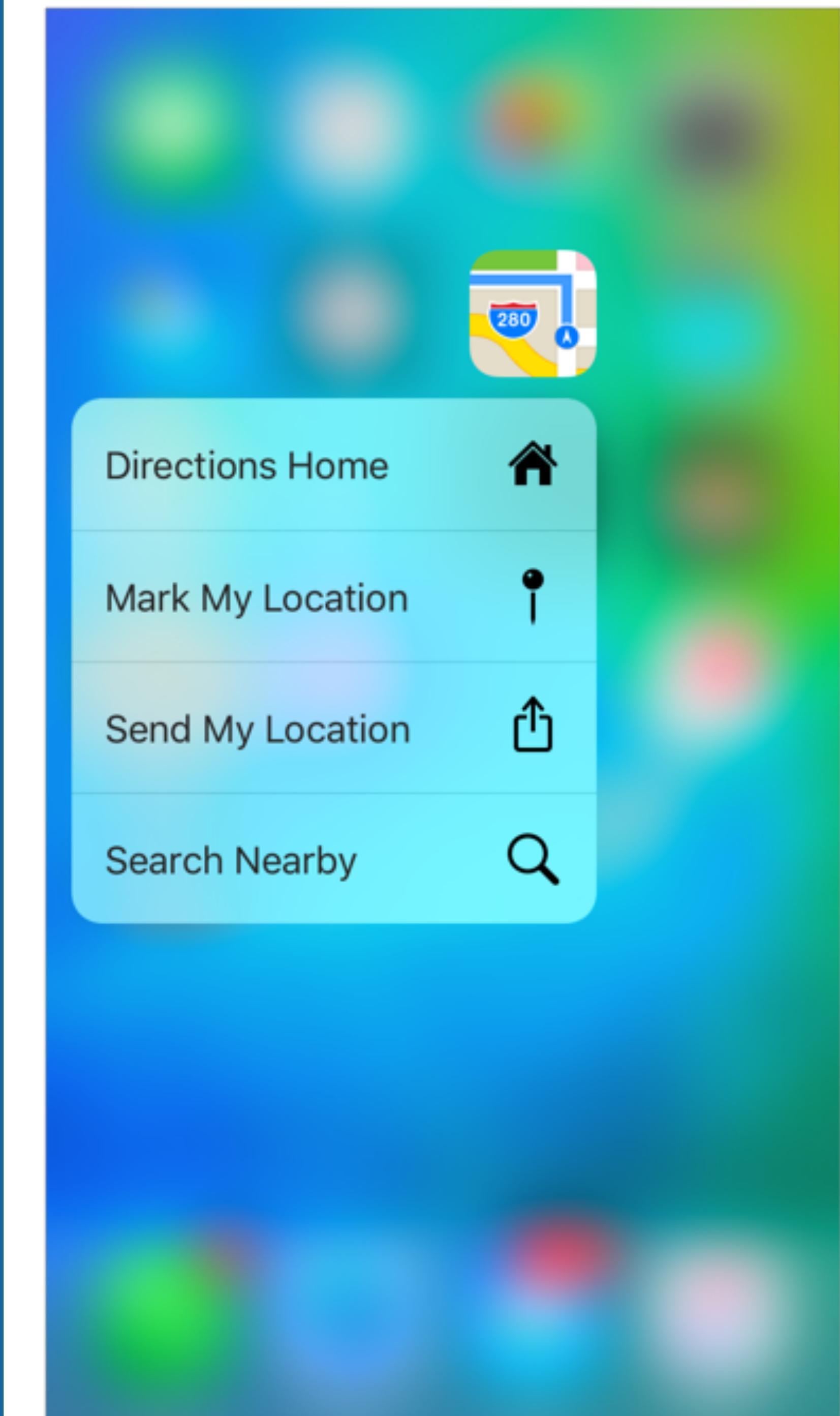
apple WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

3D TOUCH

3D TOUCH

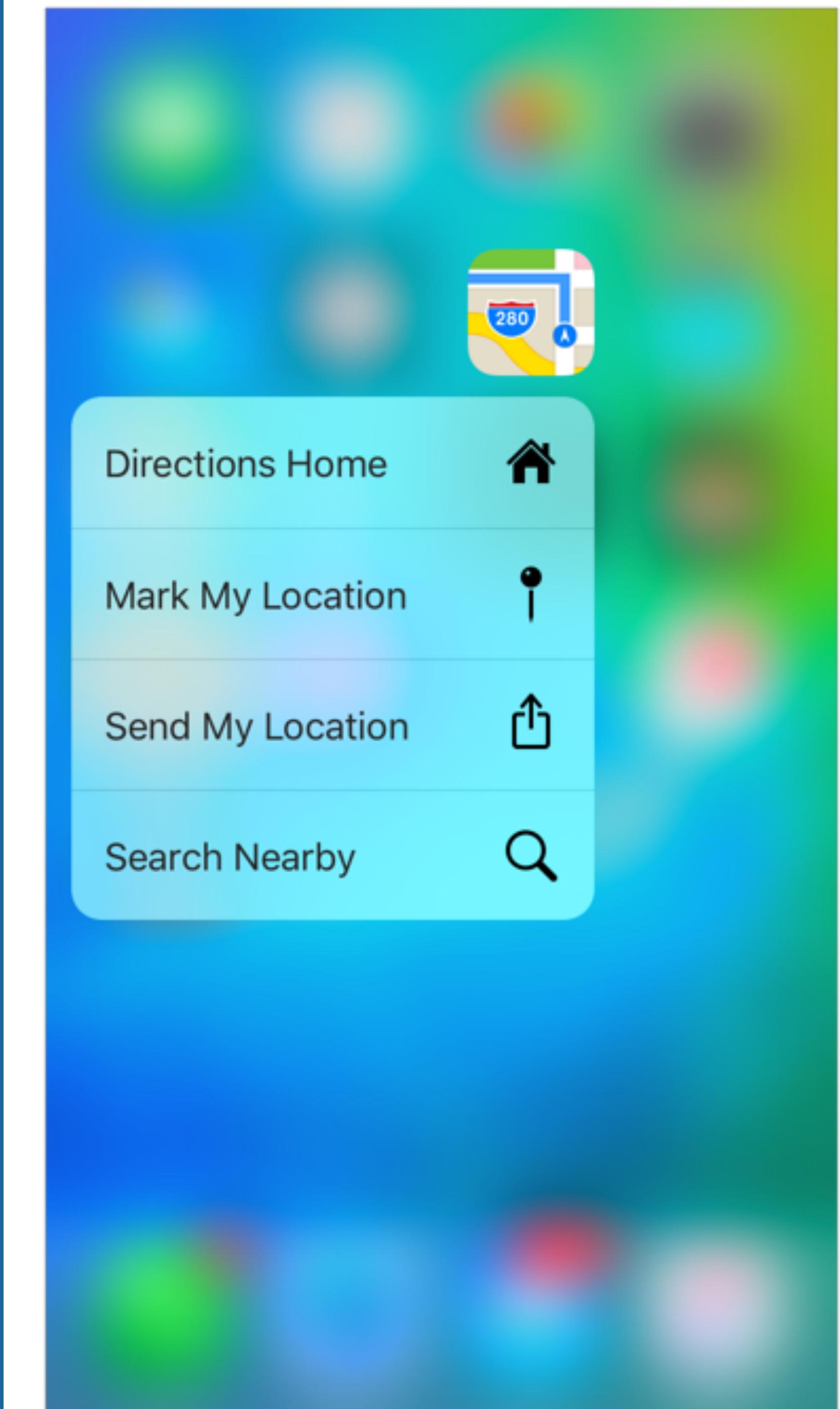
- iOS 9+, new iPhone models add a third dimension to the user interface
 - A user can now press your Home screen icon to immediately access functionality provided by your app
 - Within your app, a user can now press views to see previews of additional content and gain accelerated access to features



3D TOUCH

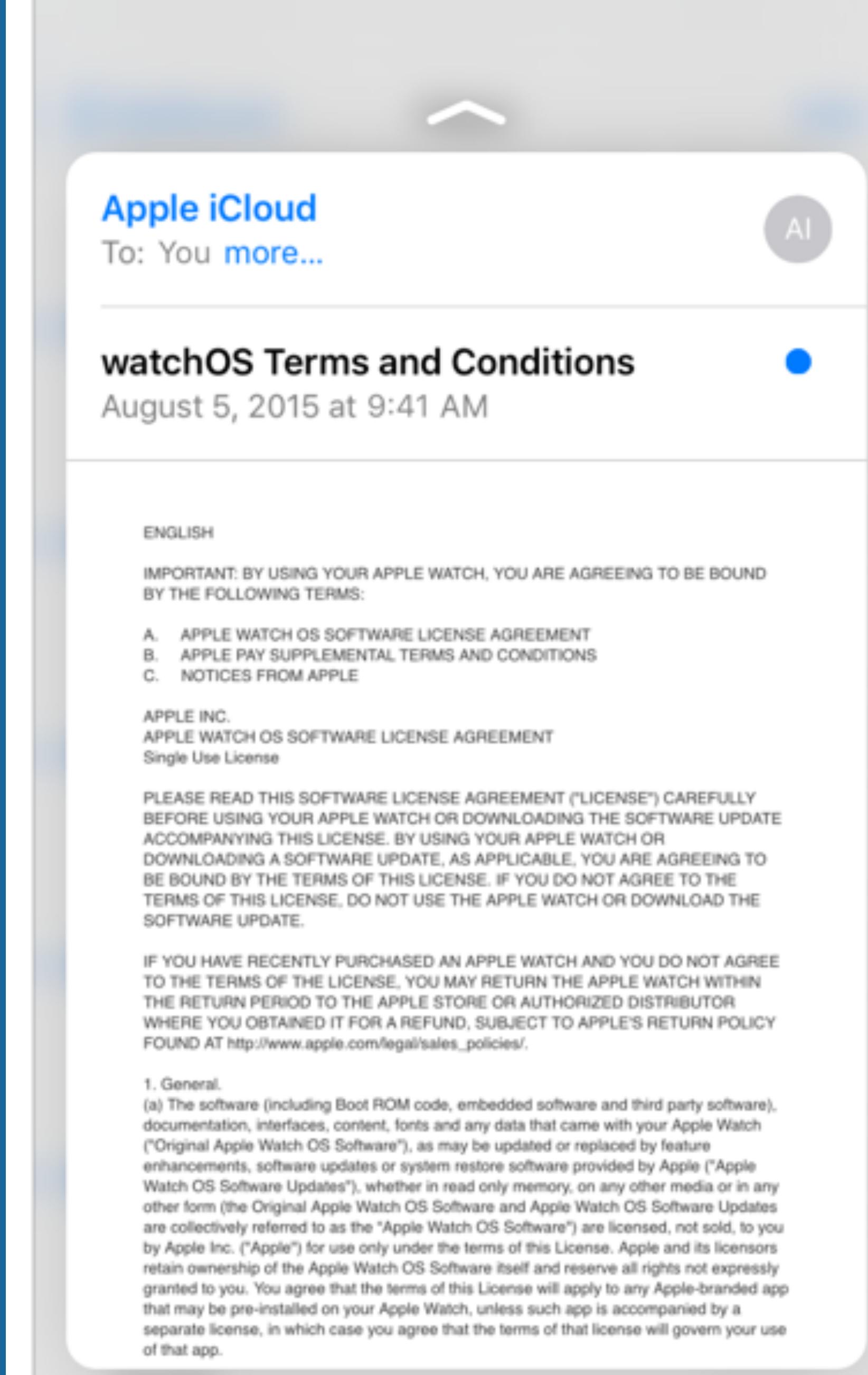
APIS

- The Home screen quick action API is for adding shortcuts to your app icon that anticipate and accelerate a user's interaction with your app
 - Sometimes at the expense of opening your application



3D TOUCH APIS

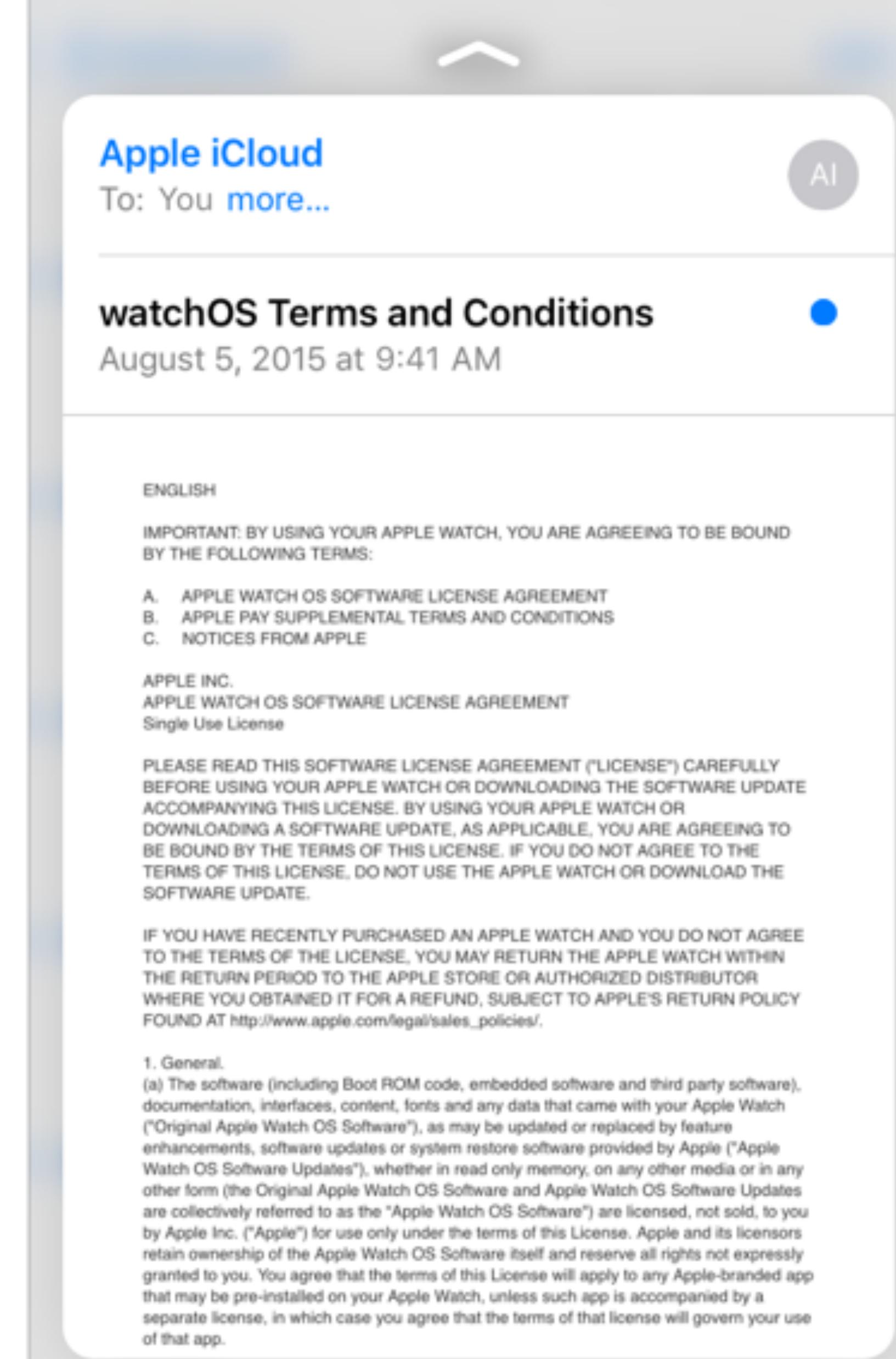
- The Web view peek and pop API lets you enable system-mediated previews of HTML link destinations



3D TOUCH

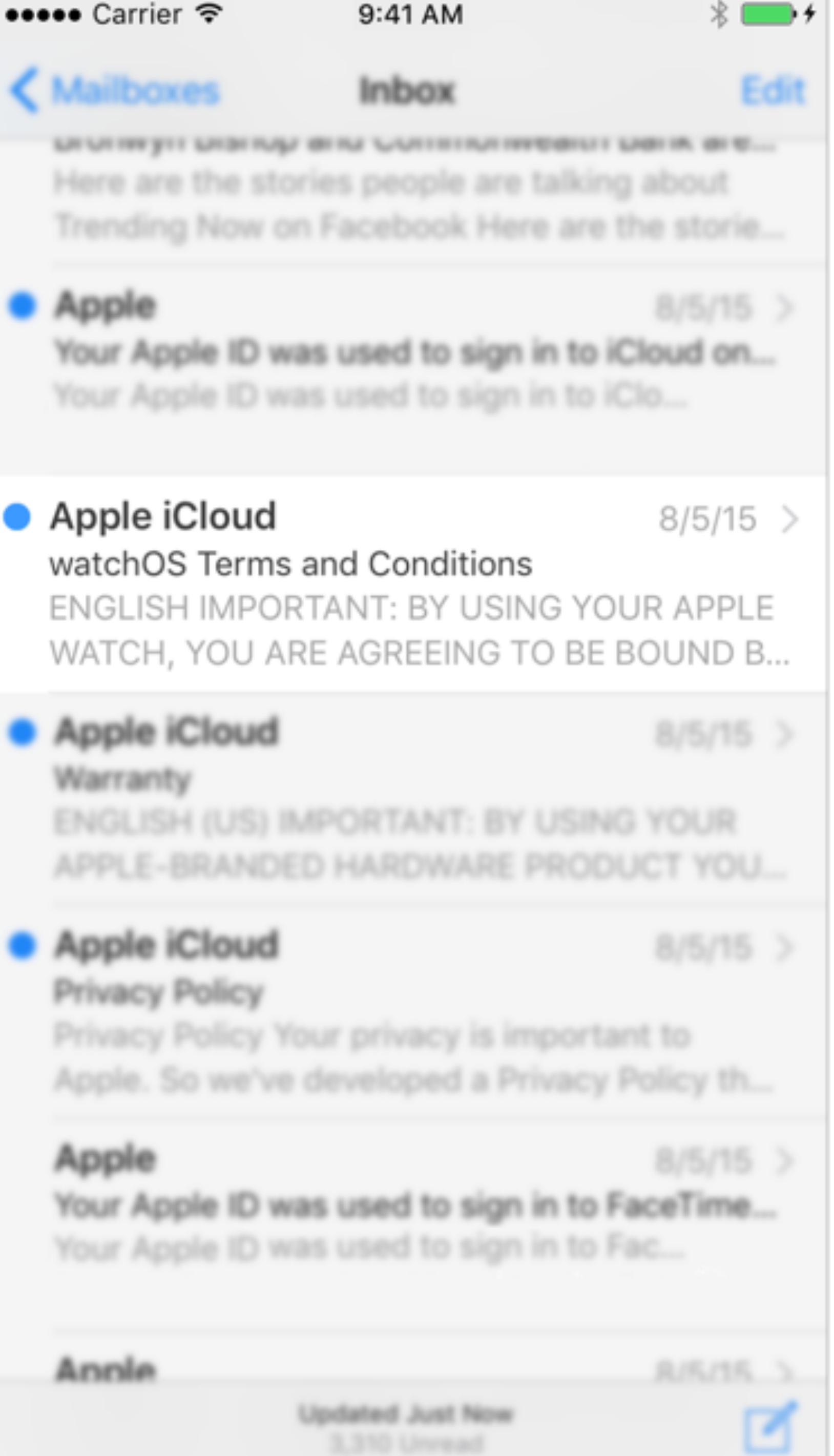
APIS

- Peek and Pop
 - UIKit
 - Web view peek and pop API lets you enable system-mediated previews of HTML link destinations



3D TOUCH APIS

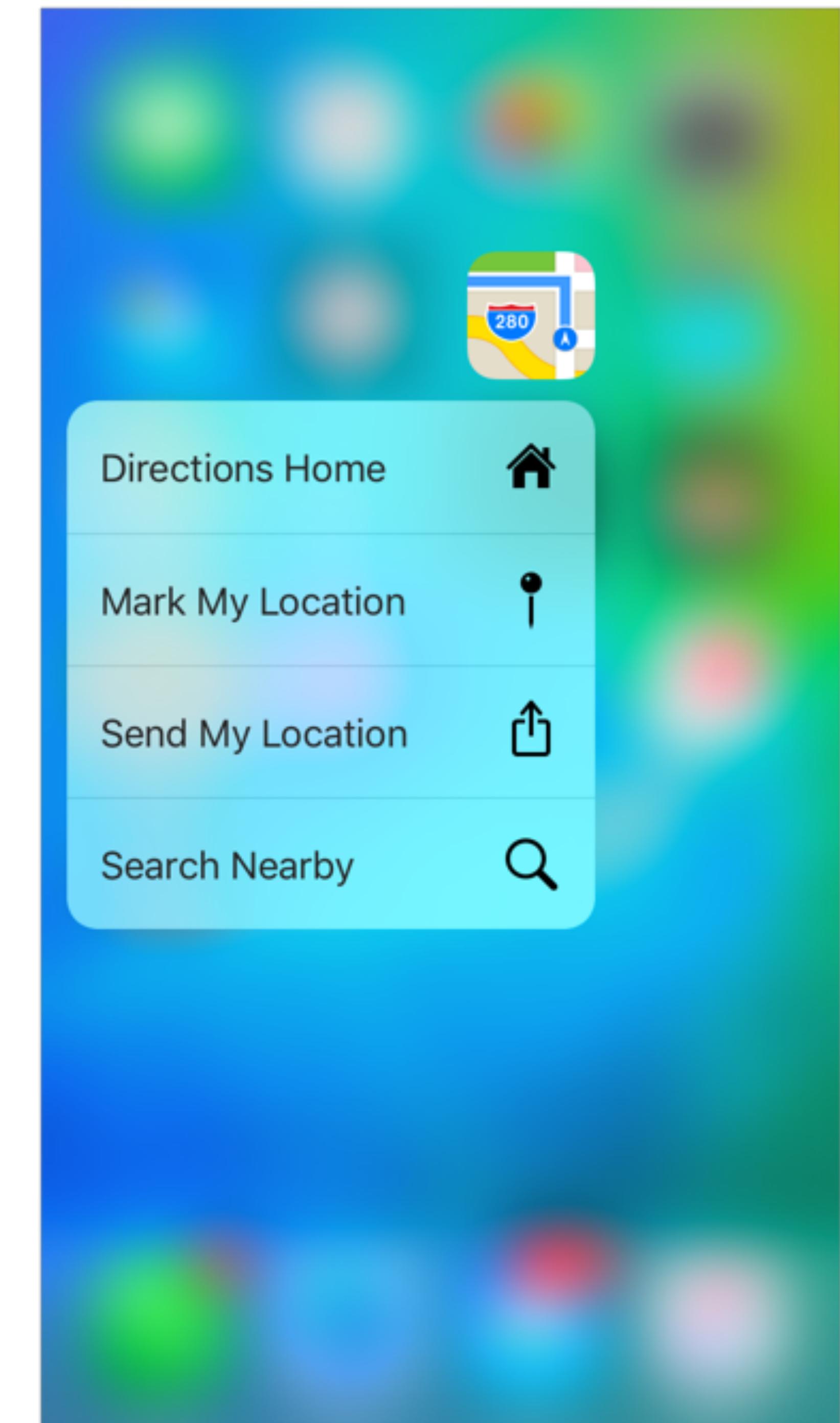
- UITapGestureRecognizer properties let you add customized force-based user interaction to your app



3D TOUCH

APIS

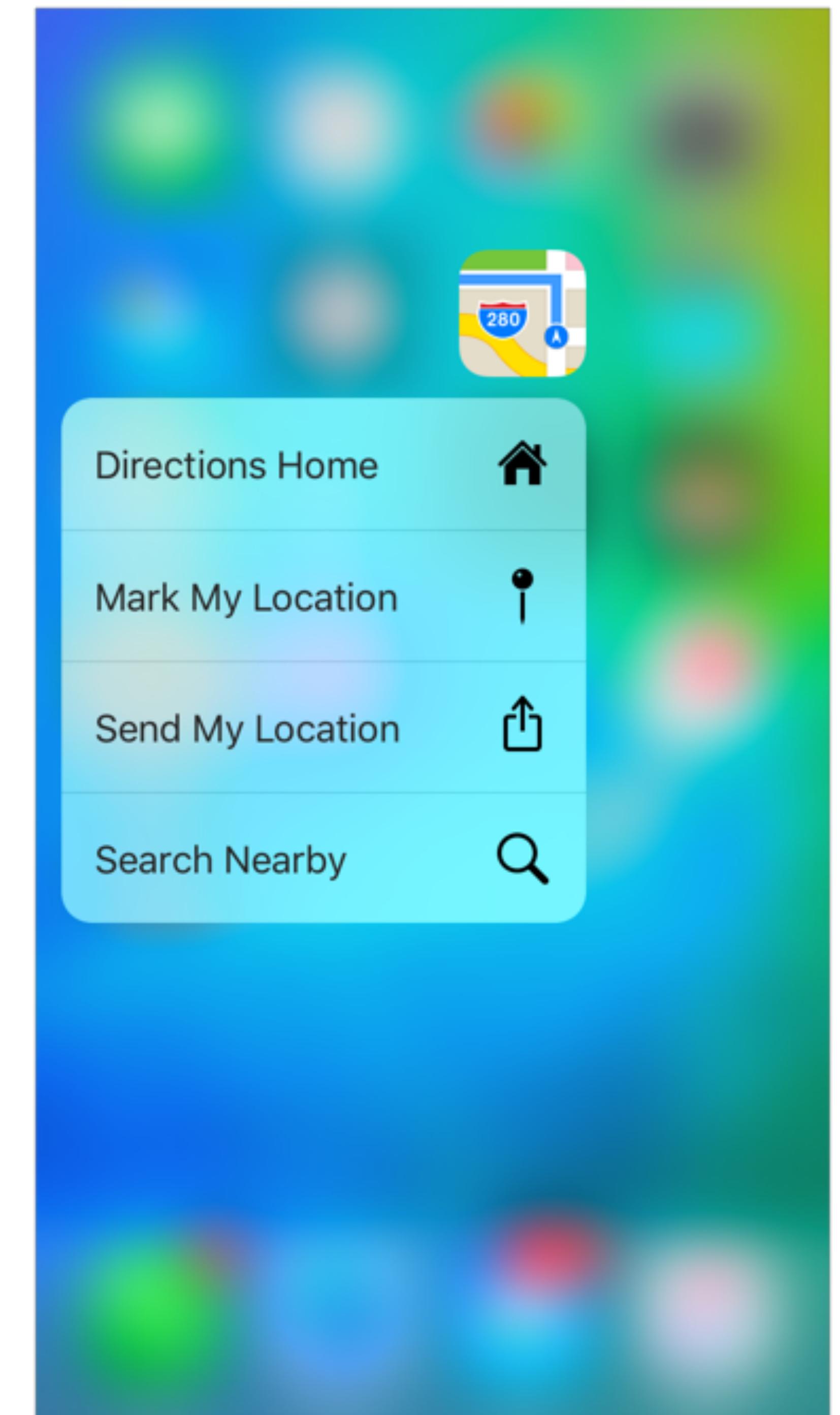
- No matter which of these APIs you adopt, your app must check the availability of 3D Touch at runtime
 - iOS6+ and higher
 - iPad Pro



HOME SCREEN

HOME SCREEN

- Support two types of quick actions (static and dynamic)
- Static quick actions
 - Available to the user immediately upon app installation
 - Define Home screen static quick actions in your app's Info.plist file in the UIApplicationShortcutItems array



HOME SCREEN

Info.plist	Bundle name	String	\$(PRODUCT_NAME)
	Bundle OS Type code	String	APPL
	Bundle versions string, short	String	1.2
	Bundle creator OS Type code	String	????
	Bundle version	String	1
	Application requires iPhone enviro...	Boolean	YES
	► UIApplicationShortcutItems	Array	(2 items)
	▼ Item 0	Dictionary	(5 items)
	UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeSearch
	UIApplicationShortcutItemSubtit...	String	shortcutSubtitle1
	UIApplicationShortcutItemTitle	String	shortcutTitle1
	UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).First
	► UIApplicationShortcutItemUserl...	Dictionary	(1 item)
	▼ Item 1	Dictionary	(5 items)
	UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeShare
	UIApplicationShortcutItemSubtit...	String	shortcutSubtitle2
	UIApplicationShortcutItemTitle	String	shortcutTitle2
	UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).Second
	► UIApplicationShortcutItemUserl...	Dictionary	(1 item)
	Launch screen interface file base...	String	LaunchScreen
	Main storyboard file base name	String	Main
	► Required device capabilities	Array	(1 item)
	► Status bar tinting parameters	Dictionary	(1 item)
	► Supported interface orientations	Array	(3 items)

HOME SCREEN

▼ Item 0	Dictionary	(5 items)	CAN ALSO USE CUSTOM IMAGE
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIcon	
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle1	
UIApplicationShortcutItemTitle	String	shortcutTitle1	
UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).First	
▼ UIApplicationShortcutItemUserl...	Dictionary	(1 item)	
firstShorcutKey1	String	firstShortcutKeyValue1	DATA DICT TO PASS
▼ Item 1	Dictionary	(5 items)	
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeShare	
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle2	
UIApplicationShortcutItemTitle	String	shortcutTitle2	
UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).Second	
▼ UIApplicationShortcutItemUserl...	Dictionary	(1 item)	

- Appearance properties and identifiers

HOME SCREEN

The user activates your application by selecting a shortcut on the home screen. Implement `application(_:willFinishLaunchingWithOptions:)` or `application(_:didFinishLaunchingWithOptions:)` to handle the shortcut in those callbacks and return `false` if possible. In the latter case, the system will not launch your application. This is useful if your application is already launched in the background.

```
func application(_ application: UIApplication, performActionFor shortcutItem: UIApplicationShortcutItem, completionHandler: @escaping (Bool) -> Void) {
    let handledShortCutItem = handleShortCutItem(shortcutItem)
    completionHandler(handledShortCutItem)
```

- Handle the shortcut item by parsing the `shortcutItem.type` to identify

UIApplicationShortcutItemType	String	\$(PRODUCT_BUNDLE_IDENTIFIER).Second
-------------------------------	--------	--------------------------------------

HOME SCREEN

- Custom logic to handle the shortcut
- You have to take it from here

```
func handleShortCutItem(_ shortcutItem: UIApplicationShortcutItem) -> Bool {  
    handled = false  
  
    // Verify that the provided `shortcutItem`'s `type` is one handled by the application.  
    if let identifier = ShortcutIdentifier(fullType: shortcutItem.type) != nil else { return false }  
  
    let shortCutType = shortcutItem.type as String? ?? ""  
  
    switch shortCutType {  
        case ShortcutIdentifier.First.type:  
            // Handle shortcut 1 (static).  
            handled = true  
            break  
        case ShortcutIdentifier.Second.type:  
            // Handle shortcut 2 (static).  
            handled = true  
            break  
        case ShortcutIdentifier.Third.type:  
            // Handle shortcut 3 (dynamic).  
            handled = true  
            break  
        case ShortcutIdentifier.Fourth.type:  
            // Handle shortcut 4 (dynamic).  
            handled = true  
            break  
        default:  
            break  
  
    }  
  
    // Construct an alert using the details of the shortcut used to open the application.  
    let alertController = UIAlertController(title: "Shortcut Handled", message: "\(shortcutItem.localizedTitle ?? "", preferredStyle: .alert)  
", preferredStyle: .alert)  
    let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)  
    alertController.addAction(okAction)  
  
    // Display an alert indicating the shortcut selected from the home screen.  
    if let rootViewController = window?.rootViewController {  
        rootViewController.present(alertController, animated: true, completion: nil)  
    }  
}  
}
```

HOME SCREEN

SUBTITLE

- Dynamic quick actions
 - Available to the user after first launch
 - Define Home screen dynamic quick actions with the `UIApplicationShortcutItem`, `UIMutableApplicationShortcutItem`, and `UIApplicationShortcutIcon` classes
 - Add dynamic quick actions to your app's shared `UIApplication` object using the `shortcutItems` property

Class

UIApplicationShortcutItem

An application shortcut item, also called a Home screen dynamic quick action, specifies a user-initiated action for your app.

Overview

On a device that supports 3D Touch, a user invokes the quick action by pressing your app's icon on the Home screen and then selecting the quick action's title. Your app delegate receives and handles the quick action.

You must specify the characteristics of your `UIApplicationShortcutItem` instances during initialization, before you register them with your app object. The quick actions you've registered with your app object are immutable.

Registering an Array of Dynamic Quick Actions With Your App

To register an array of Home screen dynamic quick actions, set the value of your shared app object's `shortcutItems` property with an `NSArray` instance containing your defined dynamic Home screen quick actions.

Changing Your App's Dynamic Quick Actions

To change your app's Home screen dynamic quick actions, replace your app object's `shortcutItems` array by setting a new value for the property. As a convenience for working with registered quick actions, this class has a mutable subclass, `UIMutableApplicationShortcutItem`. The following code snippet illustrates one way to use the `mutableCopy()` method, along with mutable quick actions, to change the title of a dynamic Home screen quick action:

HOME SCREEN

```
ationShortcutItem(type: "mobi.uchicago.s  
localizedTitle: "的笑容 Shortcut Item"  
localizedSubtitle: "Dynamic Shortcuts  
icon: UIApplicationShortcutIconImage(systemName:  
userInfo: nil)  
ortcutItems = [shortcut]
```

- Create and update the shortcut items

HOME SCREEN

```
from the home screen  
application, performActionFor shortcut  
ol) -> Void) {
```

- Static and dynamic shortcuts are handled the same way
- The `userInfo` dictionary/name should define the logic

HOME SCREEN

APPLICATION SHORTCUT PROJECT

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files like README.md, ApplicationShortcuts, AppDelegate.swift, ShortcutsTableViewController.swift, ShortcutDetailViewController.swift, Info.plist, Assets.xcassets, Base.lproj, and Products.
- Editor:** Displays the `AppDelegate.swift` file content. The code handles application launch options, specifically dealing with application shortcuts. It includes logic to handle different shortcut types (Fourth.type) and construct an alert to inform the user which shortcut was selected.
- Identity and Type:** Shows the file is named `AppDelegate.swift`, is a `Default - Swift Source`, and is located relative to the project at `ApplicationShortcuts/AppDelegate.swift`.
- On Demand Resource Tags:** A note stating "Only resources are taggable".
- Target Membership:** The file is associated with the target `ApplicationShortcuts`.
- Text Settings:** Configuration for text encoding (Unicode (UTF-8)), line endings (Default - macOS / Unix (LF)), indent using (Spaces), widths (Tab width 2, Indent width 2), and wrap lines.

```
handled = true
break
case ShortcutIdentifier.Fourth.type:
// Handle shortcut 4 (dynamic).
handled = true
break
default:
break
}

// Construct an alert using the details of the shortcut used to open the application.
let alertController = UIAlertController(title: "Shortcut Handled", message: "\"\\"(\shortcutItem.localizedTitle)
\"", preferredStyle: .alert)
let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
alertController.addAction(okAction)

// Display an alert indicating the shortcut selected from the home screen.
window!.rootViewController?.present(alertController, animated: true, completion: nil)

return handled
}

// MARK: - Application Life Cycle

func applicationDidBecomeActive(_ application: UIApplication) { }

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    var shouldPerformAdditionalDelegateHandling = true

    // If a shortcut was launched, display its information and take the appropriate action
    if let shortcutItem = launchOptions?[UIApplicationLaunchOptionsKey.shortcutItem] as? UIApplicationShortcutItem {
        launchedShortcutItem = shortcutItem

        // This will block "performActionForShortcutItem:completionHandler" from being called.
        shouldPerformAdditionalDelegateHandling = false
    }
}
```

HOME SCREEN

DEMO: EXTEND THIS APP SHOW SIMPLE

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows files like 2017-ExtendThisApp.entitlements, AppDelegate.swift, ViewController.swift, Main.storyboard, ForceViewController.swift, Assets.xcassets, LaunchScreen.storyboard, Info.plist, DataKitManager.swift, Constants.swift, CoreDataManager.swift, DataKit.h, Info.plist, Model.xcdatamodeld, and Products (2017-ExtendThisApp.app, DataKit.framework).
- Code Editor:** Displays the AppDelegate.swift code. The code handles application lifecycle events such as applicationDidEnterBackground, applicationWillResignActive, applicationWillEnterForeground, applicationWillBecomeActive, and applicationWillTerminate.
- Search Bar:** Located at the top right, with the placeholder "Search Documentation".
- Documentation:** A yellow speech bubble at the top right says "DEMO: EXTEND THIS APP SHOW SIMPLE".
- Help:** A sidebar on the right lists items: "No Quick Help", "Search Documentation", and three UI elements: "Fixed Space Bar Button Item", "Flexible Space Bar Button Item", and "View".

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        print(NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).last as Any);

        // Print out the app group (for fun)
        let sharedAppGroup: String = "group.2017-extend-this-app"
        let directory: NSURL = FileManager.default.containerURL(forSecurityApplicationGroupIdentifier: sharedAppGroup)! as NSURL
        print("App Group URL: \(directory)")

        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
    }

    func applicationWillBecomeActive(_ application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }
}
```

HOME SCREEN

DEMO: APPLICATION SHORTCUT PROJECT

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files like README.md, ApplicationShortcuts, AppDelegate.swift, ShortcutsTableViewController.swift, ShortcutDetailViewController.swift, Info.plist, Assets.xcassets, Base.lproj, and Products.
- Editor:** Displays the `AppDelegate.swift` file content. The code handles application launch options, specifically dealing with application shortcuts. It includes logic to handle different shortcut types (Fourth.type) and construct an alert to inform the user which shortcut was selected.
- Run Script:** Shows the build log: "Finished running AppShortcuts on iPhone 7" with 2 warnings.
- Identity and Type:** Shows the file is named `AppDelegate.swift`, type is "Default - Swift Source", location is "Relative to Project", and full path is `/Users/tabinkowski/Downloads/ApplicationShortcutsUsingUIApplicationShortcutItem/ApplicationShortcuts/AppDelegate.swift`.
- On Demand Resource Tags:** Shows the message "Only resources are taggable".
- Target Membership:** Shows the target is "ApplicationShortcuts".
- Text Settings:** Shows text encoding is "Unicode (UTF-8)", line endings are "Default - macOS / Unix (LF)", indent using "Spaces", widths for Tab and Indent are both set to 2, and wrap lines is checked.

```
handled = true
break
case ShortcutIdentifier.Fourth.type:
// Handle shortcut 4 (dynamic).
handled = true
break
default:
break
}

// Construct an alert using the details of the shortcut used to open the application.
let alertController = UIAlertController(title: "Shortcut Handled", message: "\"\\"(\shortcutItem.localizedTitle)
\"", preferredStyle: .alert)
let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
alertController.addAction(okAction)

// Display an alert indicating the shortcut selected from the home screen.
window!.rootViewController?.present(alertController, animated: true, completion: nil)

return handled
}

// MARK: - Application Life Cycle

func applicationDidBecomeActive(_ application: UIApplication) { }

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    var shouldPerformAdditionalDelegateHandling = true

    // If a shortcut was launched, display its information and take the appropriate action
    if let shortcutItem = launchOptions?[UIApplicationLaunchOptionsKey.shortcutItem] as? UIApplicationShortcutItem
    {

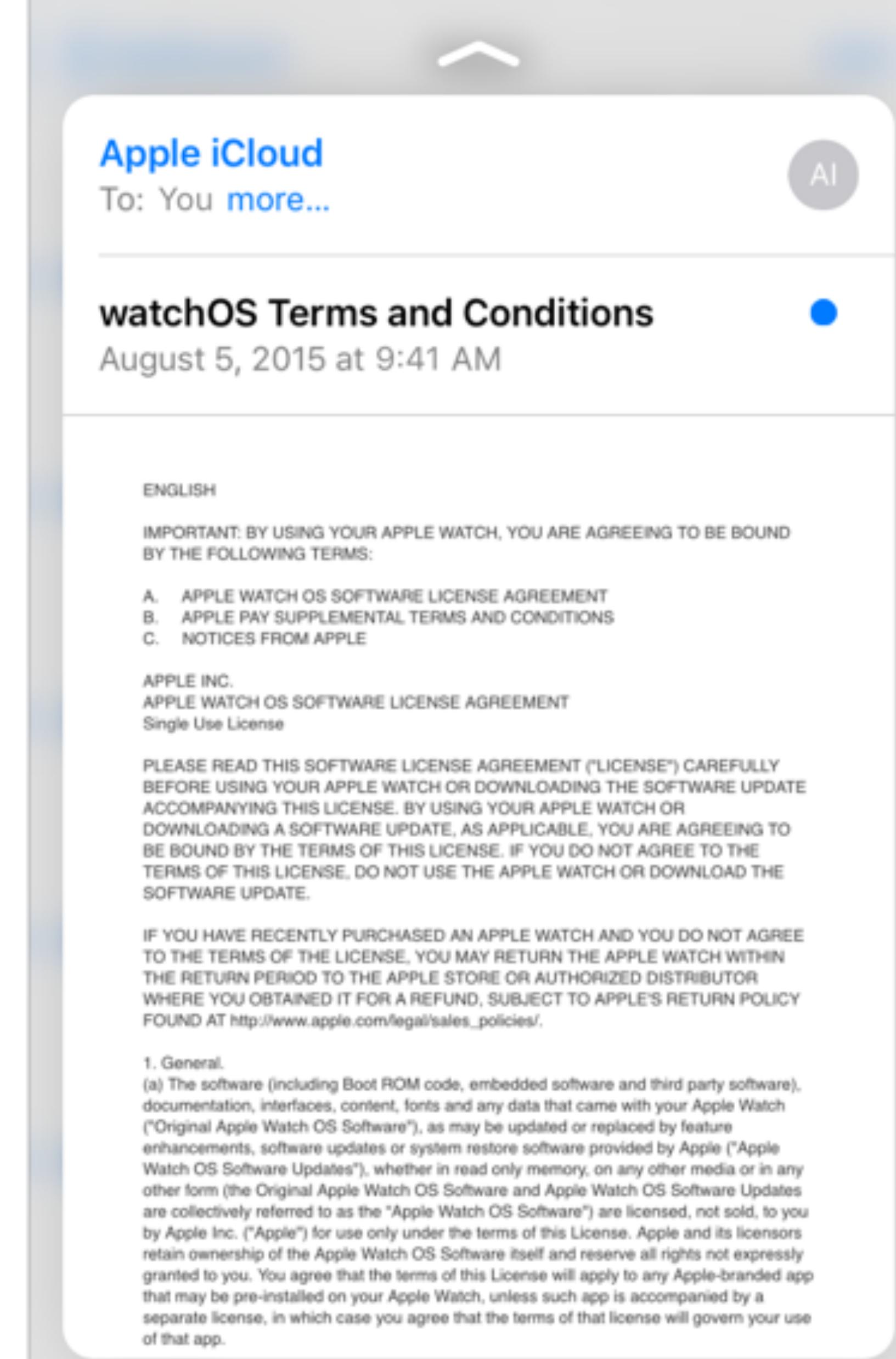
        launchedShortcutItem = shortcutItem

        // This will block "performActionForShortcutItem:completionHandler" from being called.
        shouldPerformAdditionalDelegateHandling = false
    }
}
```

**PEEK AND POP IN
STORYBOARD**

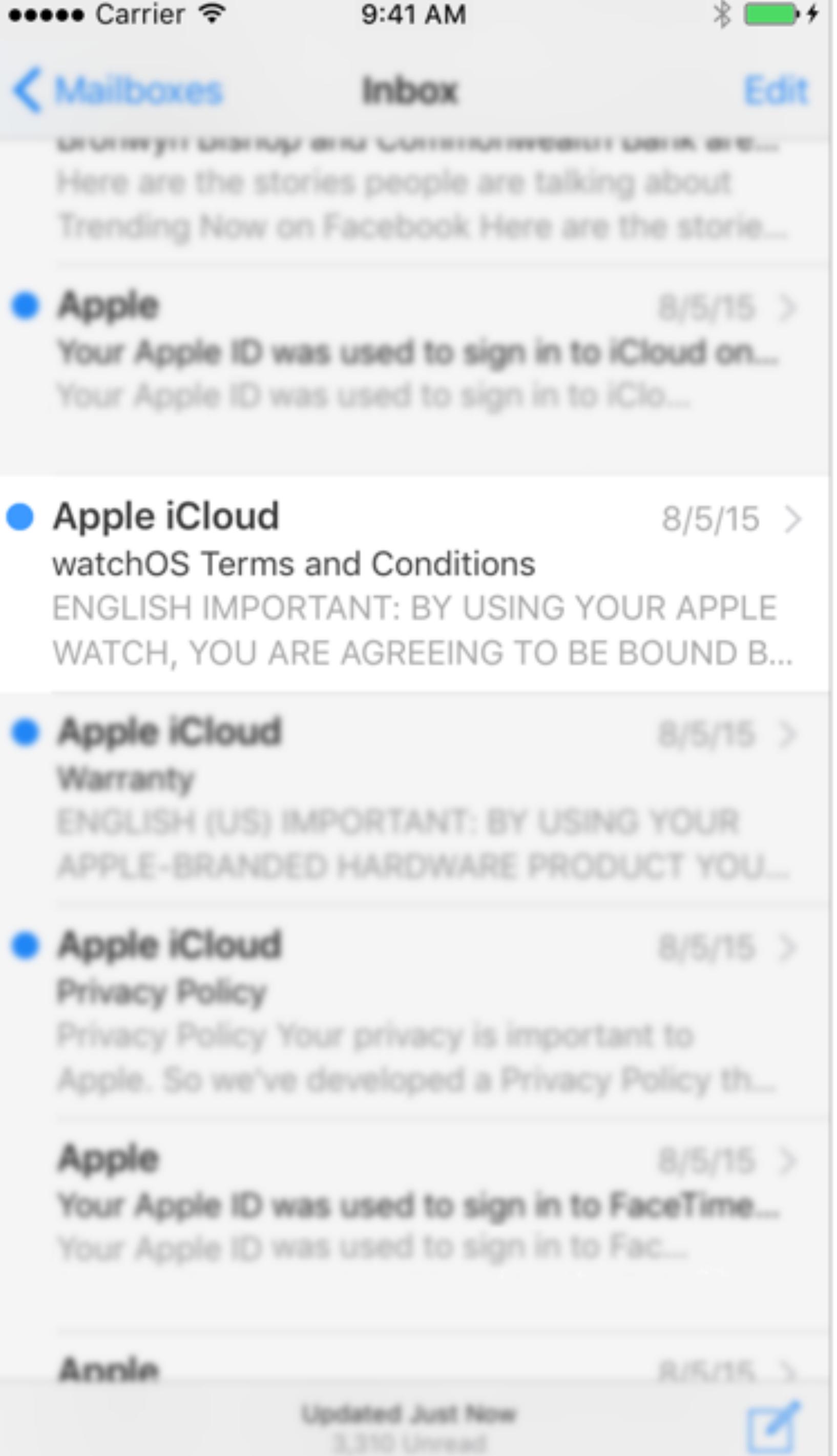
PEEK AND POP

- Peek
 - Preview of the content
 - "preview"
- Pop
 - Go to the content
 - "commit"



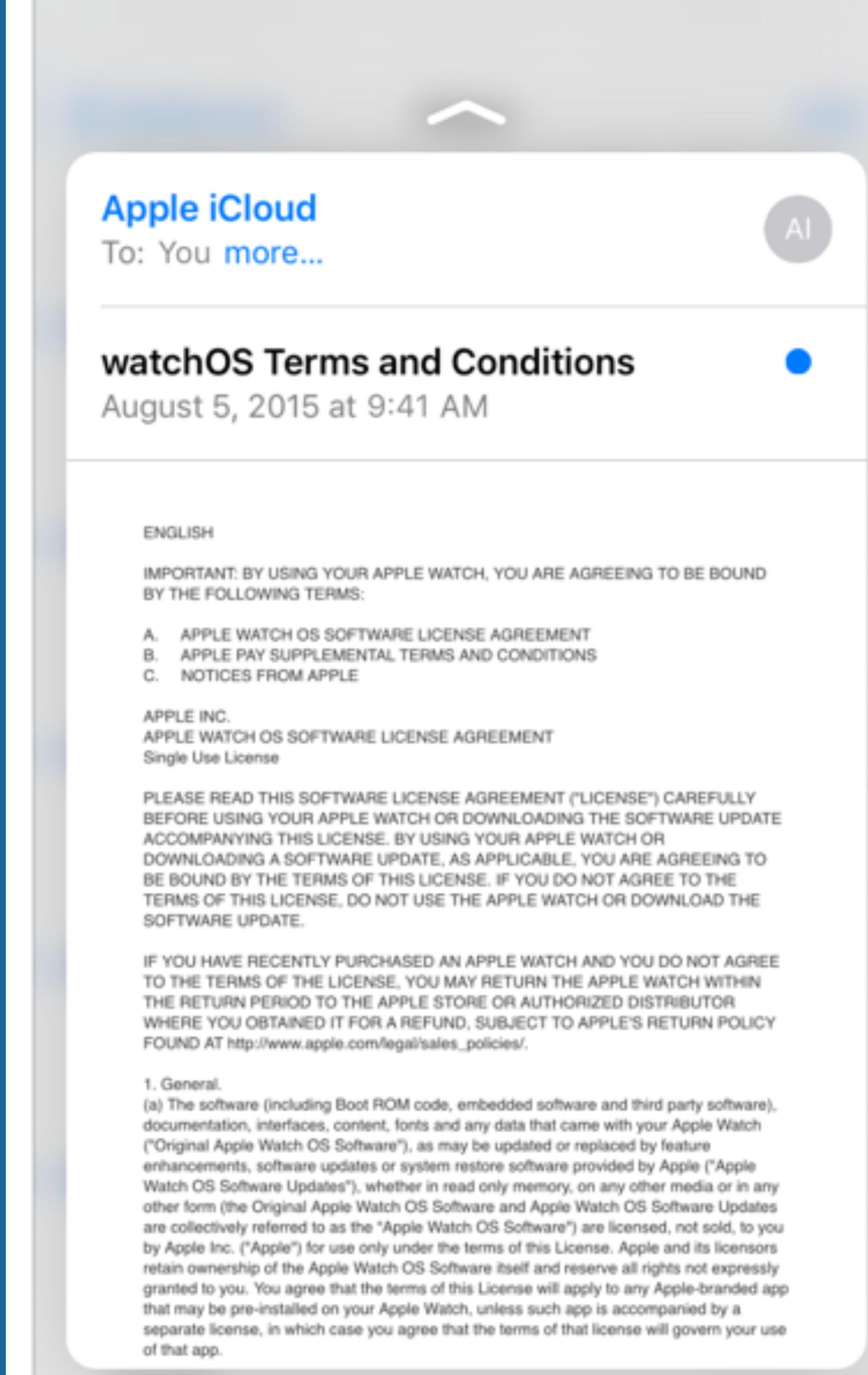
PEEK AND POP

- Indication of peek availability
 - With a light press, surrounding content blurs to tell the user a preview of additional content—the peek—is available



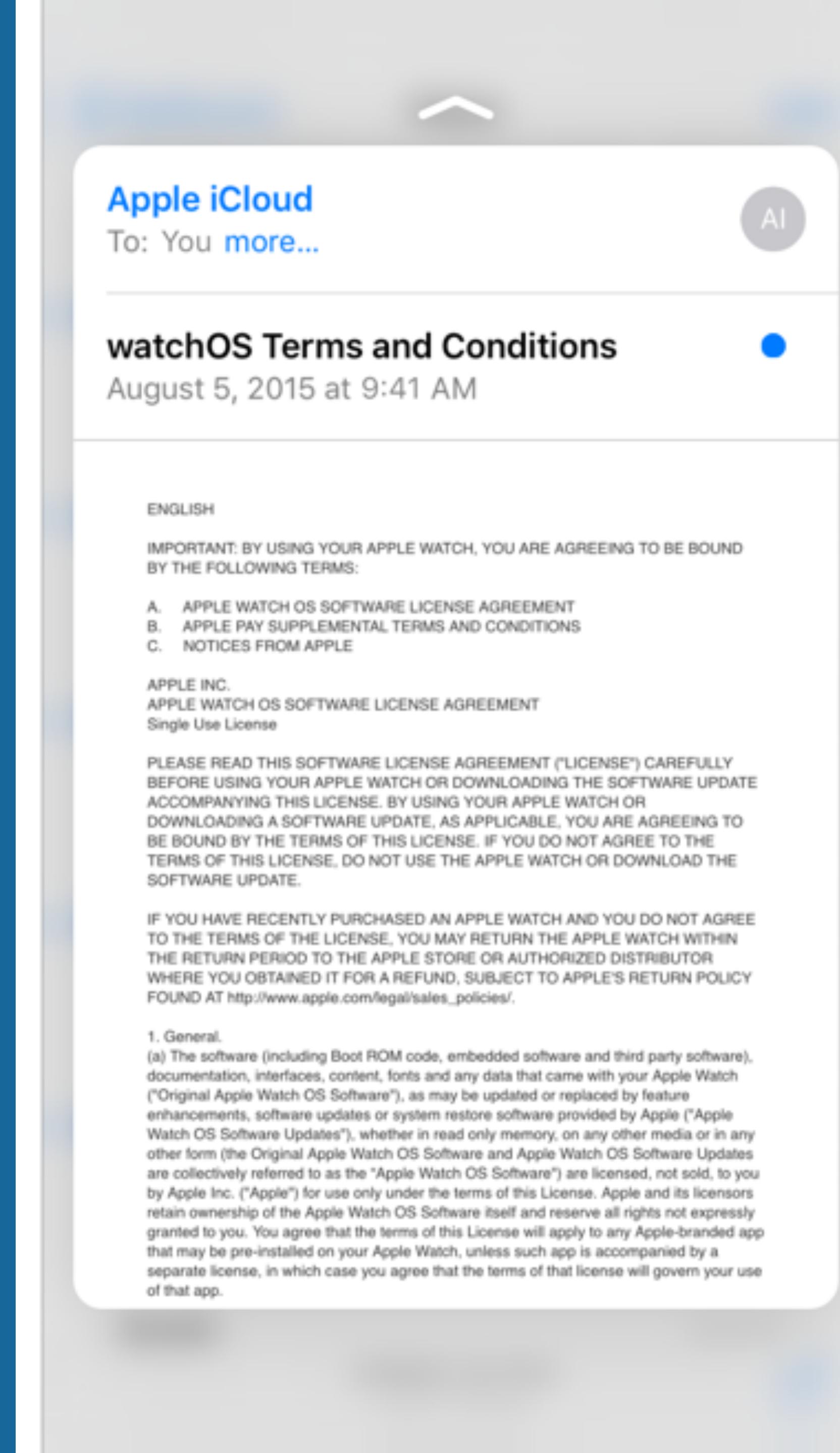
PEEK AND POP

- Peek
 - Press a bit more deeply and the view transitions to show the peek
 - If the user ends the touch at this point, the peek disappears and the app returns to its state before the interaction started



PEEK AND POP

- Pop
 - The user can press deeper still on the peek itself to navigate, using the system-provided pop transition, to the view being previewed as a peek
 - The pop view then fills your app's root view and displays a button to navigate back to where the interaction began



PEEK AND POP

- Peek quick actions
 - If the user swipes the peek upward, the system shows the peek quick actions
 - Each peek quick action is a deep link into your app
 - 

IMPORTANT: BY USING YOUR APPLE WATCH, YOU ARE AGREEING TO BE BOUND BY THE FOLLOWING TERMS:

- A. APPLE WATCH OS SOFTWARE LICENSE AGREEMENT
- B. APPLE PAY SUPPLEMENTAL TERMS AND CONDITIONS
- C. NOTICES FROM APPLE

APPLE INC.
APPLE WATCH OS SOFTWARE LICENSE AGREEMENT
Single Use License

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY BEFORE USING YOUR APPLE WATCH OR DOWNLOADING THE SOFTWARE UPDATE ACCOMPANYING THIS LICENSE. BY USING YOUR APPLE WATCH OR DOWNLOADING A SOFTWARE UPDATE, AS APPLICABLE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, DO NOT USE THE APPLE WATCH OR DOWNLOAD THE SOFTWARE UPDATE.

[Reply](#)

[Forward](#)

[Mark...](#)

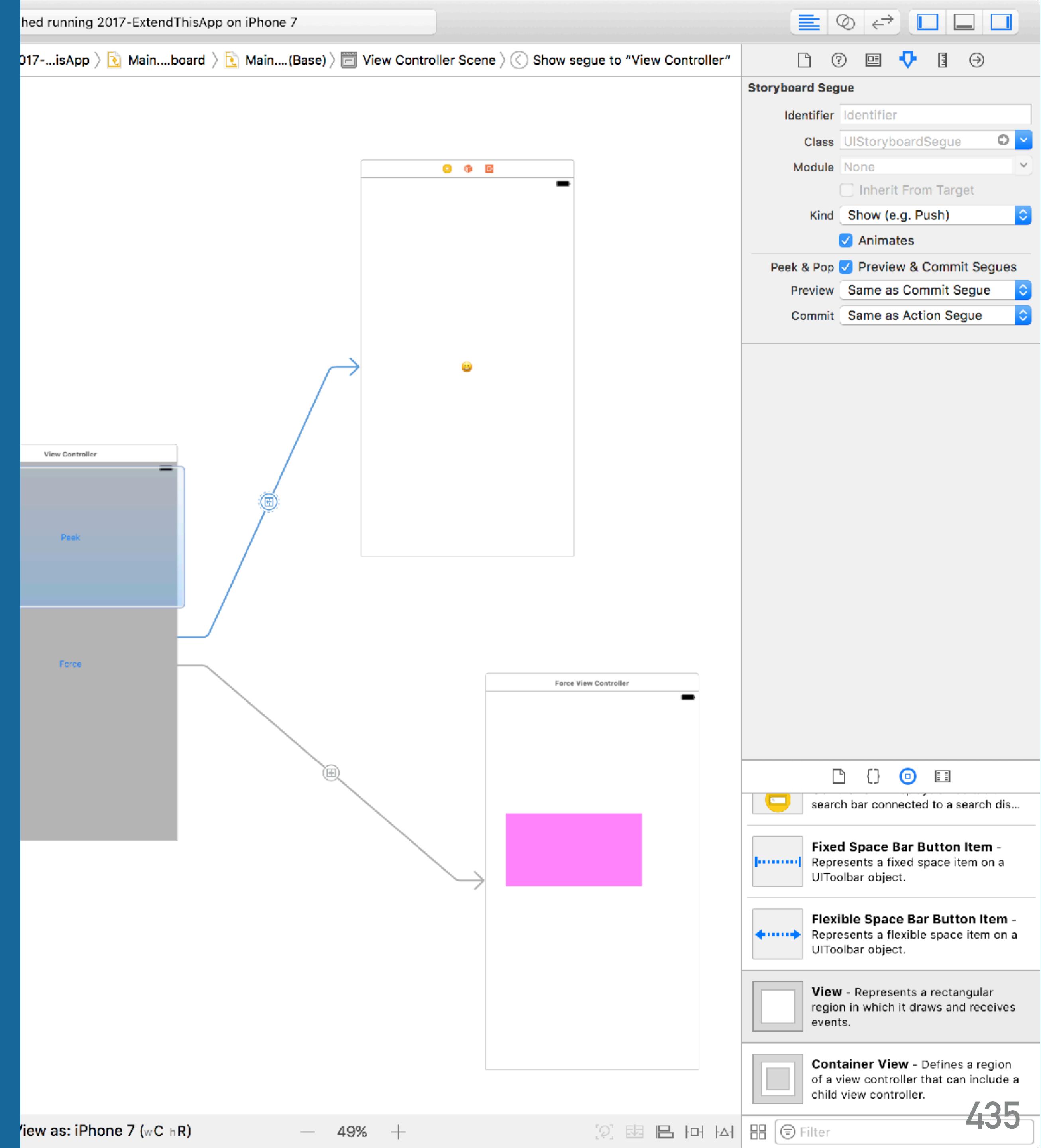
[Notify Me...](#)

[Move Message...](#)

PEEK AND POP

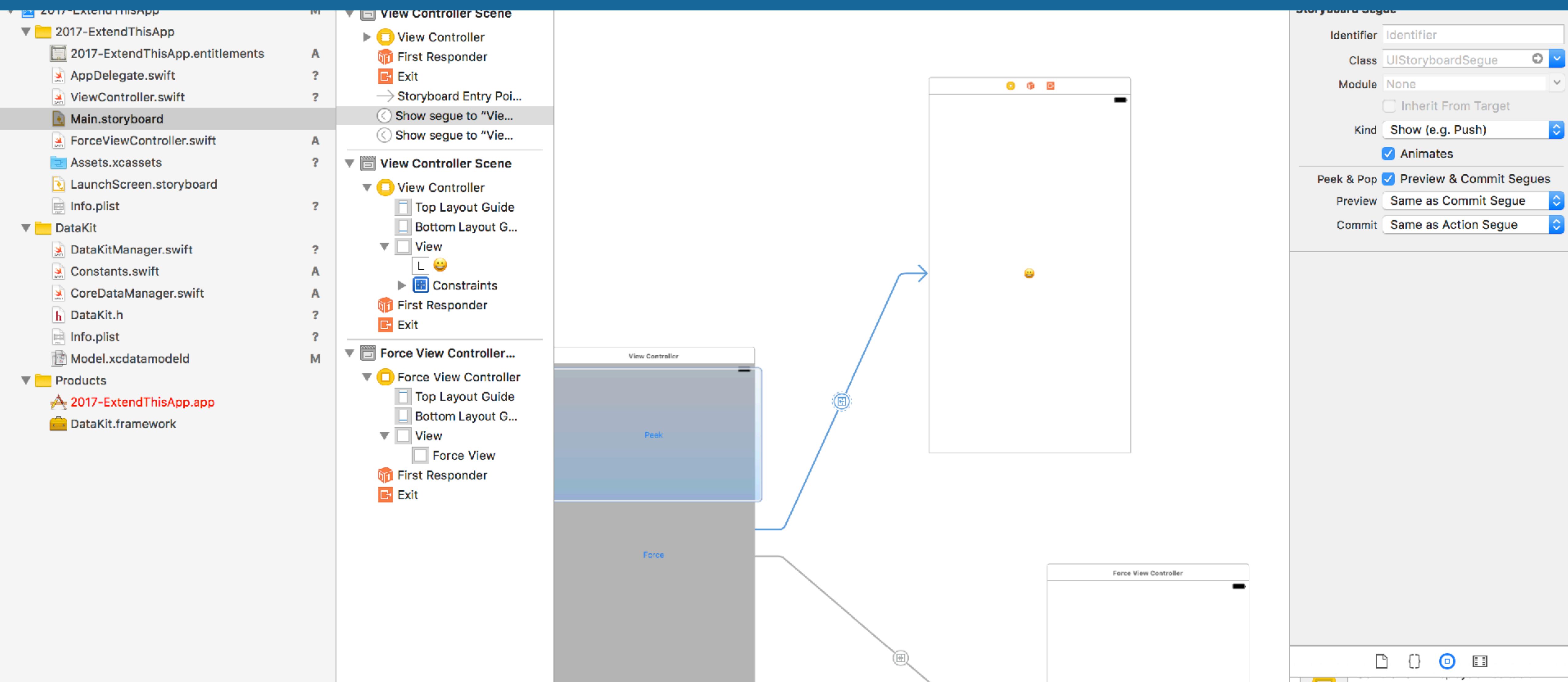
SUBTITLE

- Implement Peek and Pop programmatically or in Storyboard



PEEK AND POP

DEMO: EXTEND THIS APP



FORCE PROPERTIES

FORCE PROPERTIES



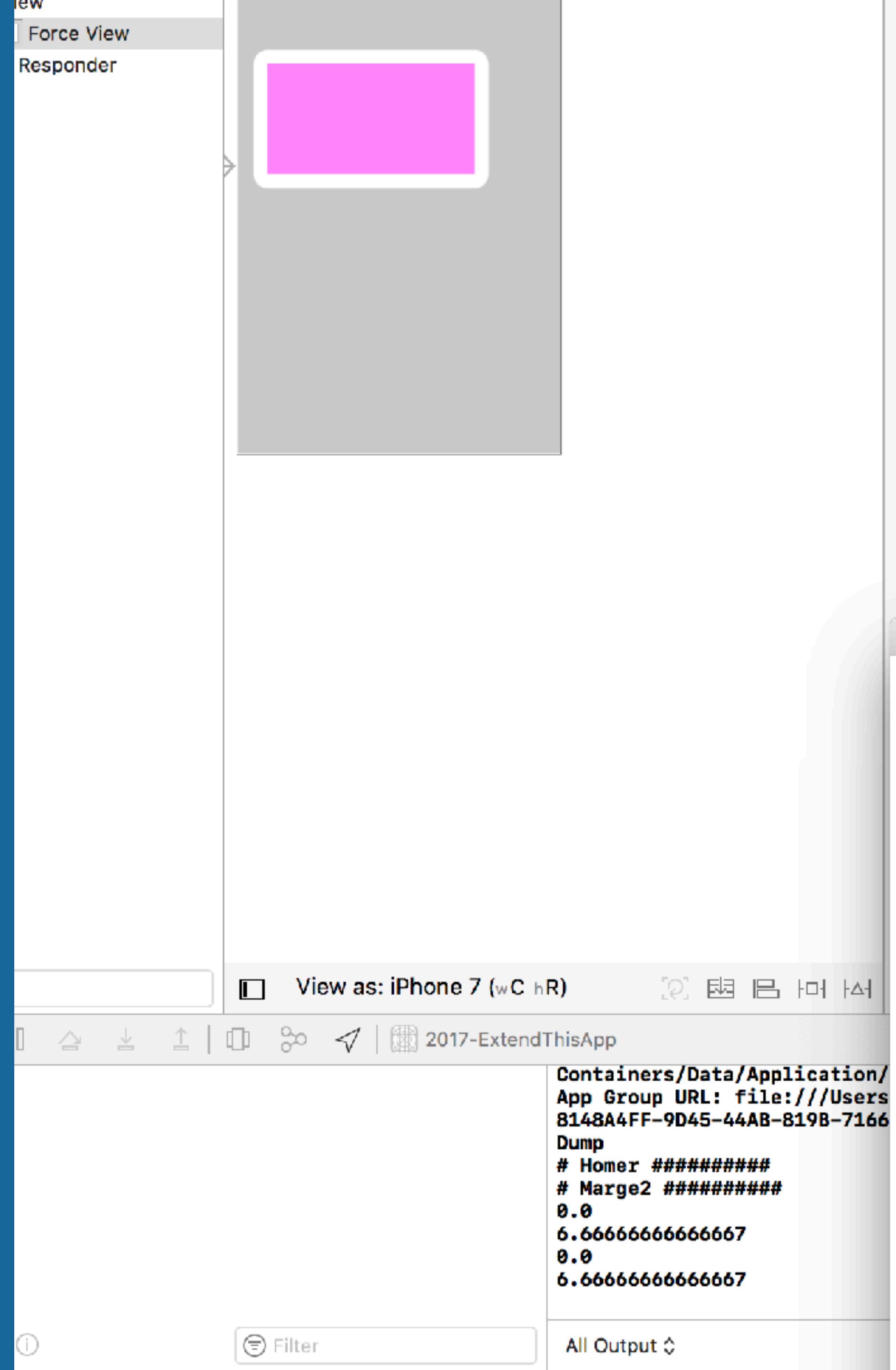
FORCE PROPERTIES

SUBTITLE

- UITapGestureRecognizer class has two new properties to support custom implementation of 3D Touch in your app
 - force
 - maximumPossibleForce
- Detect and respond to touch pressure in the UIEvent objects your app receives



FORCE PROPERTIES



```
class ForceViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

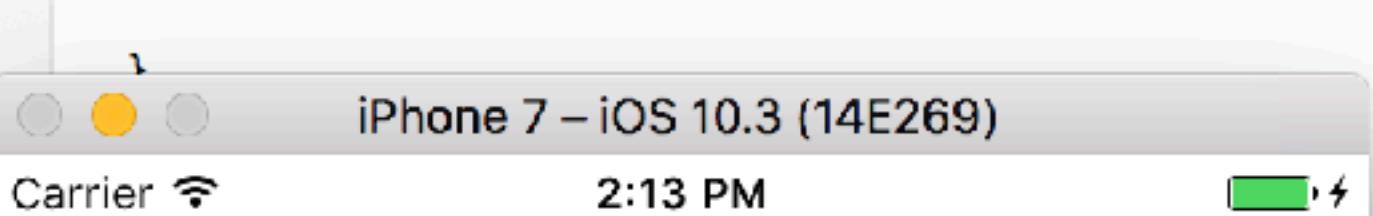
        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}

class ForceView: UIView {

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {
            print(touch.force)
            print(touch.maximumPossibleForce)
        }
    }
}
```



4AB9-8A66-C