iOS Dev Accelerator
Week 5 Day 4

- Targets
- WatchKit
- HashTables

WatchKit

- 3 ways to present information on Apple Watch:
  - User can manually launch your watch app (required)
  - Provide glances to user (optional)
  - Provide custom UI for your apps remote/local notifications (optional)
- All of this is contained in your watch app, which itself is another target contained inside an already existing iPhone app.

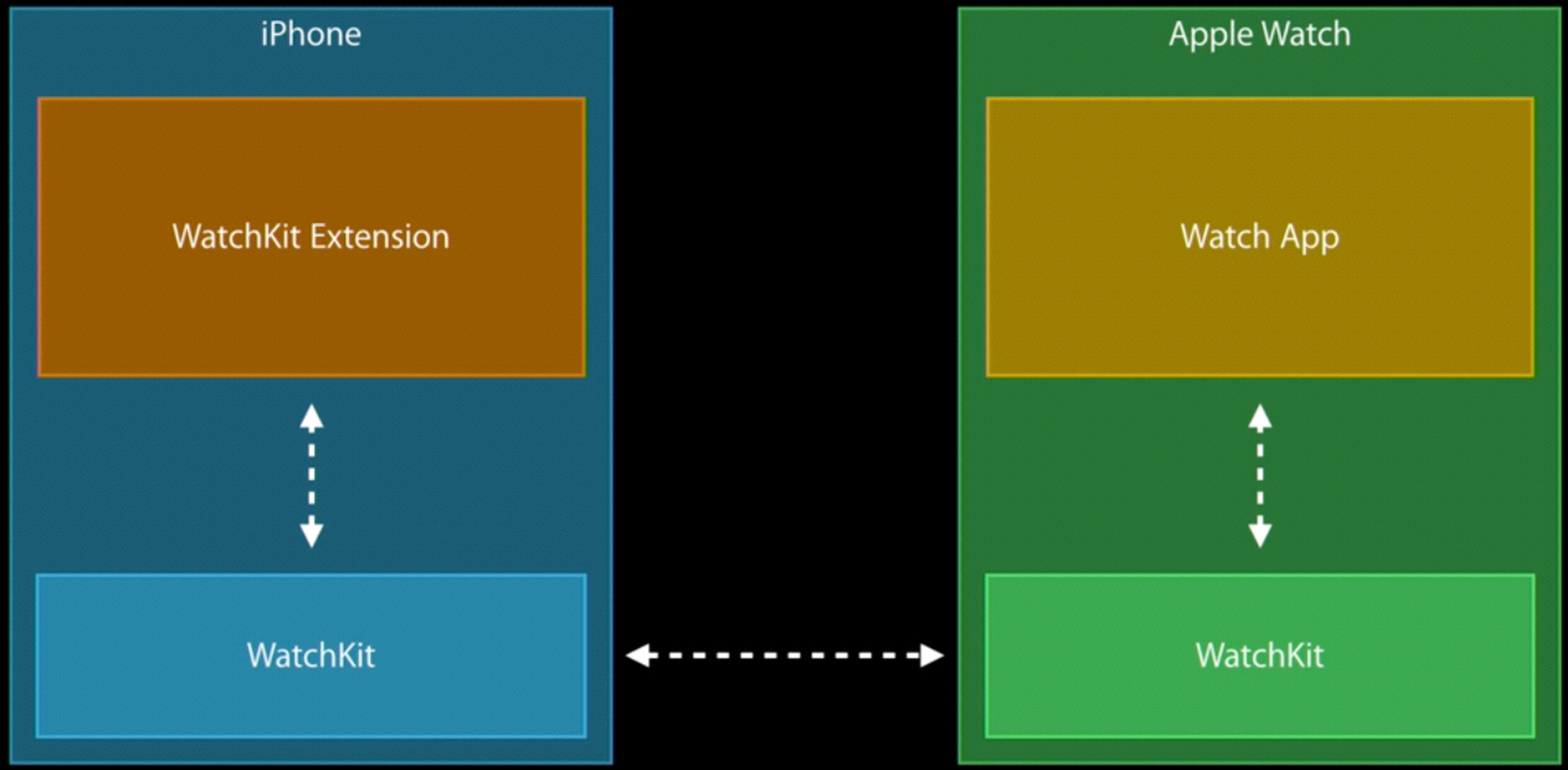
WatchKit app architecture

- Watch apps are comprised of 2 parts
  - WatchKit Extension, runs on iPhone, responds to user actions done on the watch itself. This is your watch app's code.
  - On the watch, your watch's app UI is loaded from a bundle. **No code from your app is actually on the apple watch (or very little).**
- The iPhone and Apple Watch are constantly passing information back and forth

Developing a WatchKit app

- As simple as adding a new target to your existing iPhone app project in Xcode.
- Use all the same tools in Xcode as you have used for iPhone/iPad apps (storyboard, debugger, etc)

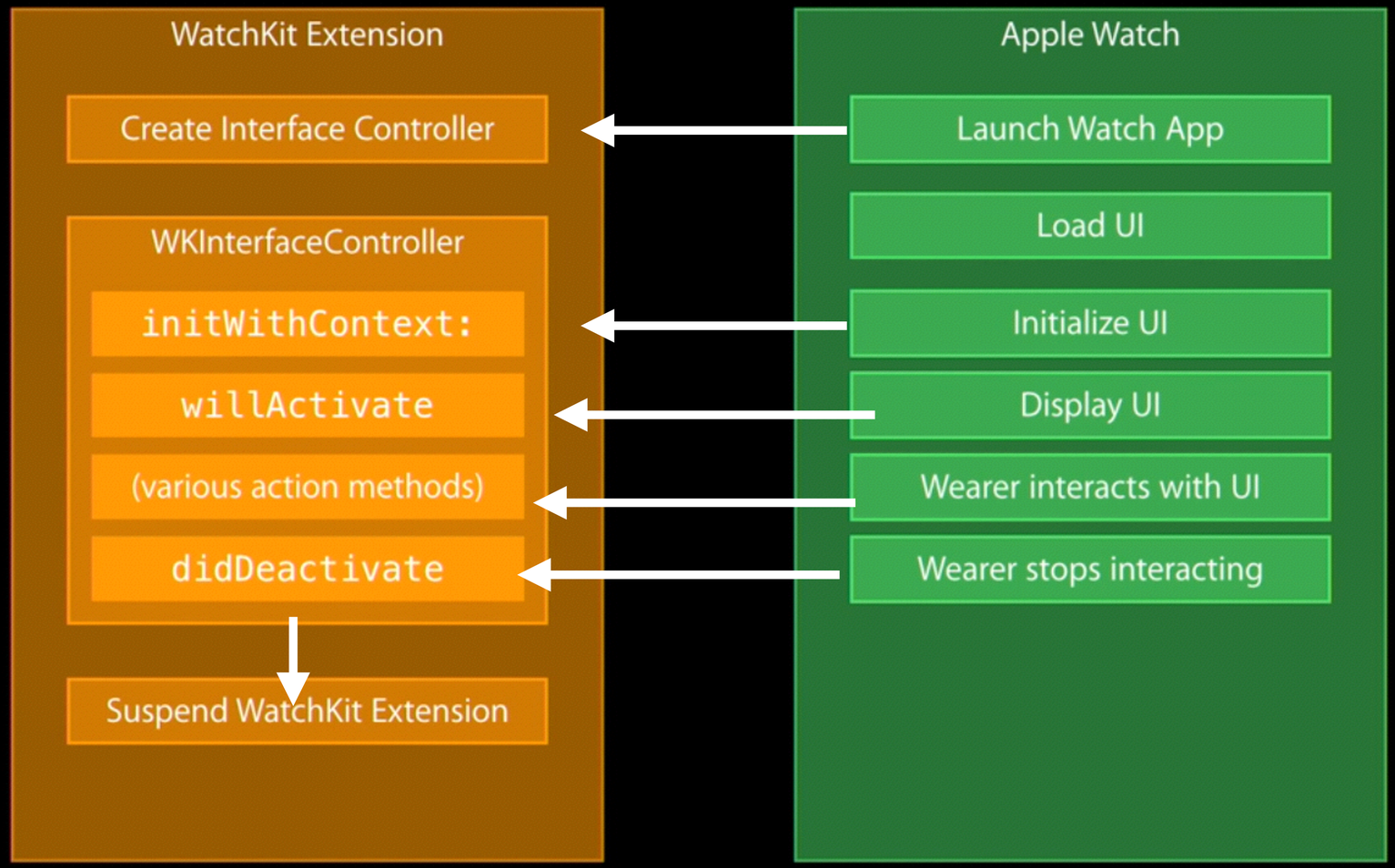
Watch app architecture



Watch app internal workflow

1. When your watch app is launched on Apple watch, the watch extension is launched on your iPhone app
2. The storyboard in your watch app bundle is loaded on apple watch, and the initial interface controller is created on the extension
3. Once the UI is initialized, your interface controller will receive the initWithContext call.
4. When the UI is in the process of getting displayed, willActivate is called on your interface controller
5. As the wear interacts with UI elements, WatchKit calls the IBAction methods you defined on your Interface controller. This is your chance to respond to the actions and update interface elements appropriately.
6. Finally, didDeactivate will tell your interface controller when it is no longer on screen. An interface controller can be de-allocated at anytime after this method as returned, so use this time to save any state and do any cleanup.

Watch app internal workflow



Simulating your WatchKit app

- When you create your watch kit app, Xcode automatically configures a build scheme for running and debugging your Xcode app.
- An Xcode scheme defines a collection of targets to build, a configuration to use when building, and a collection of tests to execute.

Demo

WatchKit Framework

- The WatchKit framework provides everything you need to create your Watch app's UI.
- Its the equivalent of UIKit for iPhone/ iPad apps.

WKInterfaceController

- Is the equivalent of UIViewController in UIKit
- Instead of managing UIView's in a scene, WKInterfaceController manages elements in a scene.
- One controller per screen of content
- Manage UI elements through outlets
- Use IBAction to trigger events based on UI interactions

WKInterfaceController LifeCycle

- Just like UIViewController, WKInterfaceController has life cycle methods! Just 2:
  - willActivate()
  - didDeactivate()
- Also provides one designated initializer:
    - initWithContext:
- Context can be anything! Pretty cool, much easier to pass information to the next interface controller in the stack this way vs UIViewController. Thanks, Android.

WKInterfaceController & Navigation

- Watch App's have 2 and only 2 navigation architectures:
- Hierarchal - Navigation stack based. Just like an app with UINagivationController. InterfaceControllers have the methods pushControllerWithName:Context: and popController for this purpose.
- Page Based - Act just like a scroll view, displaying a predetermined number of scenes each managed by an interface controller.
- These 2 are not allowed to be mixed. You must choose one. The one caveat: You can modally present a page based controller from a hierarchal based controller.

WKInterfaceGroup

- Used for layout either horizontally or vertically
- In WatchKit, objects automatically flow downward from the lop left corner of the screen, filling the available space.
- Groups let you manipulate the spacing and arrangement of UI Elements to achieve the look that you want.
- Groups don't have a visual appearance by default, but they do have a backgroundColor and image property you can set to give it an appearance.
- Autolayout is not available with WatchKit, so groups is one of the primary ways you can achieve the custom layout you want.

WatchKit UI Elements

- All UI Elements have these 4 configurable properties that dicate its appearance:
  - Height
  - Width
  - Alpha
  - Hidden
- Notice how there is no frame. Woah! The location of UI Elements are always dictated by whatever container element they are added to (Table, Group, etc)

Demo

WKInterfaceMap

- Non-interactive snapshot
- Configured dynamically
- Up to 5 annotations
- Tapping map opens maps app on Apple Watch

WKInterfaceTable

- Used to achieve the same thing as UITableView, but are implemented differently.
- No sections, headers, footers, editing, searching, datasources, or delegates
- Still a single column of dynamically created rows
- Multiple row types (content, header, footer)
- **Rows are backed by a row controller object**

Row Controller

- Every prototype row you add to the table interface element on storyboard has a Table Row Controller.
- That Table Row Controller has a Group inside of it. It is this group you can place interface elements and then create outlets to in your custom table row controller class.
- Your table row controller class is just a subclass of NSObject.

Row Controller Workflow

- Row Controllers are very similar to a custom tableview/collectionview cell:
    1. Create a custom row controller class which is a subclass of NSObject
    2. Set the class of the table row controller that is generated for you when you drag a table out to match your new custom class
    3. Set an identifier on your row controller via storyboard
    4. Add interface elements to to the row controller's group and then create your outlets.

WatchKit table Workflow

1. Drag a table onto your interface controller in storyboard
2. Create and setup your custom table row controller
3. Create an outlet to your table in your interface controller class
4. Call setNumberOfRows on your table, passing in the count of your backing array (or some number to denote how many rows you want)
5. enumerate through your backing array, use rowControllerAtIndex: on your table to grab a reference to each row controller, and set your interface outlets accordingly.

Demo

WatchKit Glances

- A Glance is a supplemental way for the user to view important information from your app.
- Not all apps need a glance!
- Should be used to provide immediately relevant information in a timely manner
- Glances are accessed by the user by swiping up on the watch face of the apple watch.
- Glances do not support any interactivity (buttons, switches), **except tapping a glance automatically launches your watch app.**

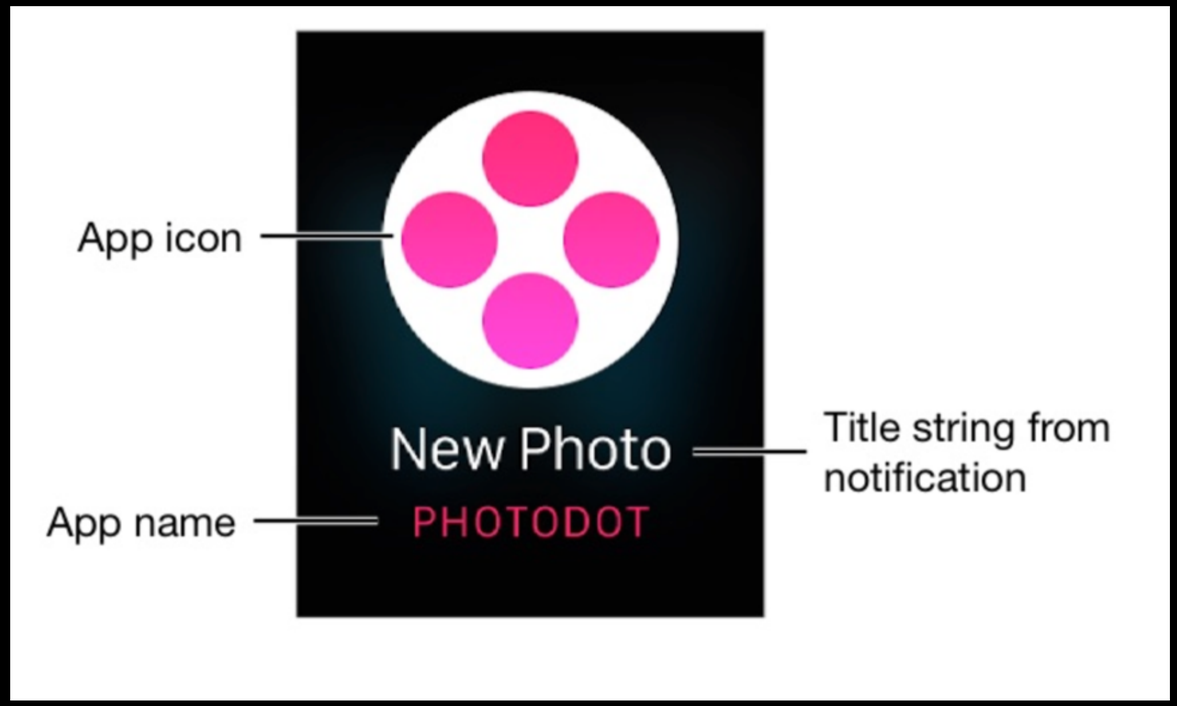
Demo

WatchKit Notifications

- If your iOS app supports local or push notifications, Apple Watch can display those notifications too
- When a notification arrives on the user's iPhone, iOS decides whether to display that notification on the iPhone or Apple Watch.
- The system first lets the user know subtly that a notification is available (some small vibration)
- If the user raises the watch to look at it, an abbreviated 'Shortlook' form of the notification is displayed
- If the user continues to view the notification, or taps it, the system shows a longer 'Long-look' version of the notification. The longer version of the notification is the one you get to customize.
- From there the user can dismiss the notification, or act on the notification by tapping an action button or launching the watch app

Watchkit ‘Shortlook’ interface

- The interface the short look notification is a non-scrolling screen that cannot be customized
- The system uses a predefined template to display the app name and icon with the title string stored in the local notification or remote notification payload.
- If the user continues to look at the short look notification, the system transitions quickly from the short-look interface to the long-look interface.



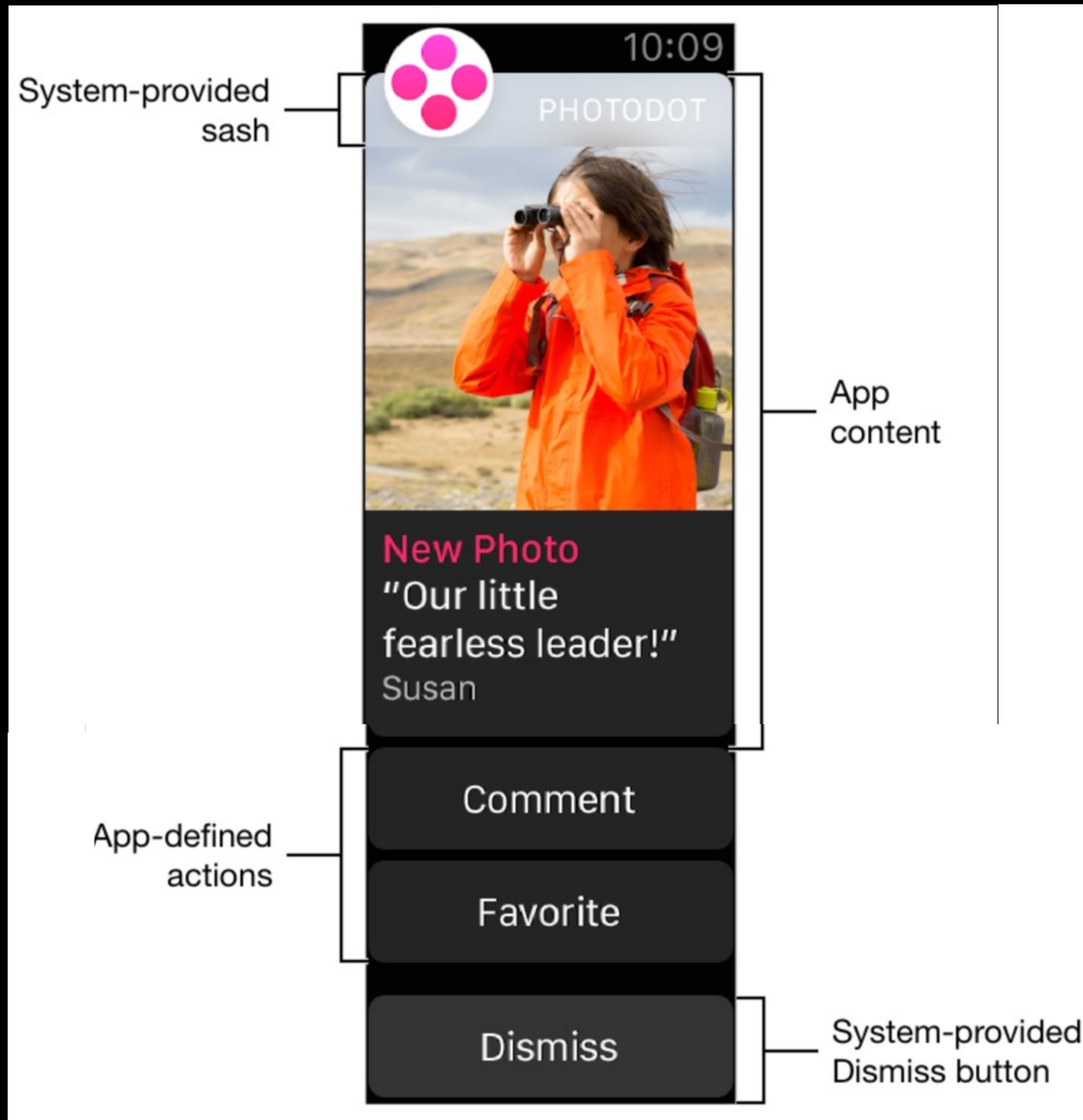
Watchkit ‘Long-look’ interface

- The long-look interface is a scrollable screen that displays the notifications content and any associated action buttons.
- If you do not provide a custom notification interface, Apple watch displays a pre-built one that includes your app icon, notification string, and alert message.
- If you do provide a custom interface, Apple Watch uses that one instead.

Long-View Elements

- **The Sash** is an overlay that contains the app icon and app name
- The content area contains detailed information about the notification
- The bottom area contains a dismiss button (automatically provided) and any action buttons defined by your app.

Clicking anywhere in the Sash or content area launches your app.



Demo

Adding action buttons

- Action buttons save time for the user by offering canned responses for a notification.
- In iOS 8 and later, apps are required to register the types of notification-generated alerts they display using a UIUserNotificationSettings object.
- When creating that object, you can also specify custom notification categories. WatchKit uses these categories to add action buttons to your apps long-look notification interface.

Adding action buttons

```
var categories = NSMutableSet()

var acceptAction = UIMutableUserNotificationAction()
acceptAction.title = NSLocalizedString("Accept", comment: "Accept invitation")
acceptAction.identifier = "accept"
acceptAction.activationMode = UIUserNotificationActivationMode.Background
acceptAction.authenticationRequired = false

var declineAction = UIMutableUserNotificationAction()
declineAction.title = NSLocalizedString("Decline", comment: "Decline invitation")
declineAction.identifier = "decline"
declineAction.activationMode = UIUserNotificationActivationMode.Background
declineAction.authenticationRequired = false
```

```
// Configure other actions and categories and add them to the set...
var settings = UIUserNotificationSettings(forTypes: (.Alert | .Badge | .Sound),
                                          categories: categories)

UIApplication.sharedApplication().registerUserNotificationSettings(settings)
```

Responding to action buttons

- When the user taps an action button for a notification, the system uses the information in the registered UIUserNotificationAction object to determine how to process the action.
- Actions can be processed in the foreground or background:
  - Foreground actions launch your watch kit app and deliver the ID of the tapped button to the main interface controller
  - Background actions launch the containing iOS app in the background so that it can process the action

Responding to action buttons cont.

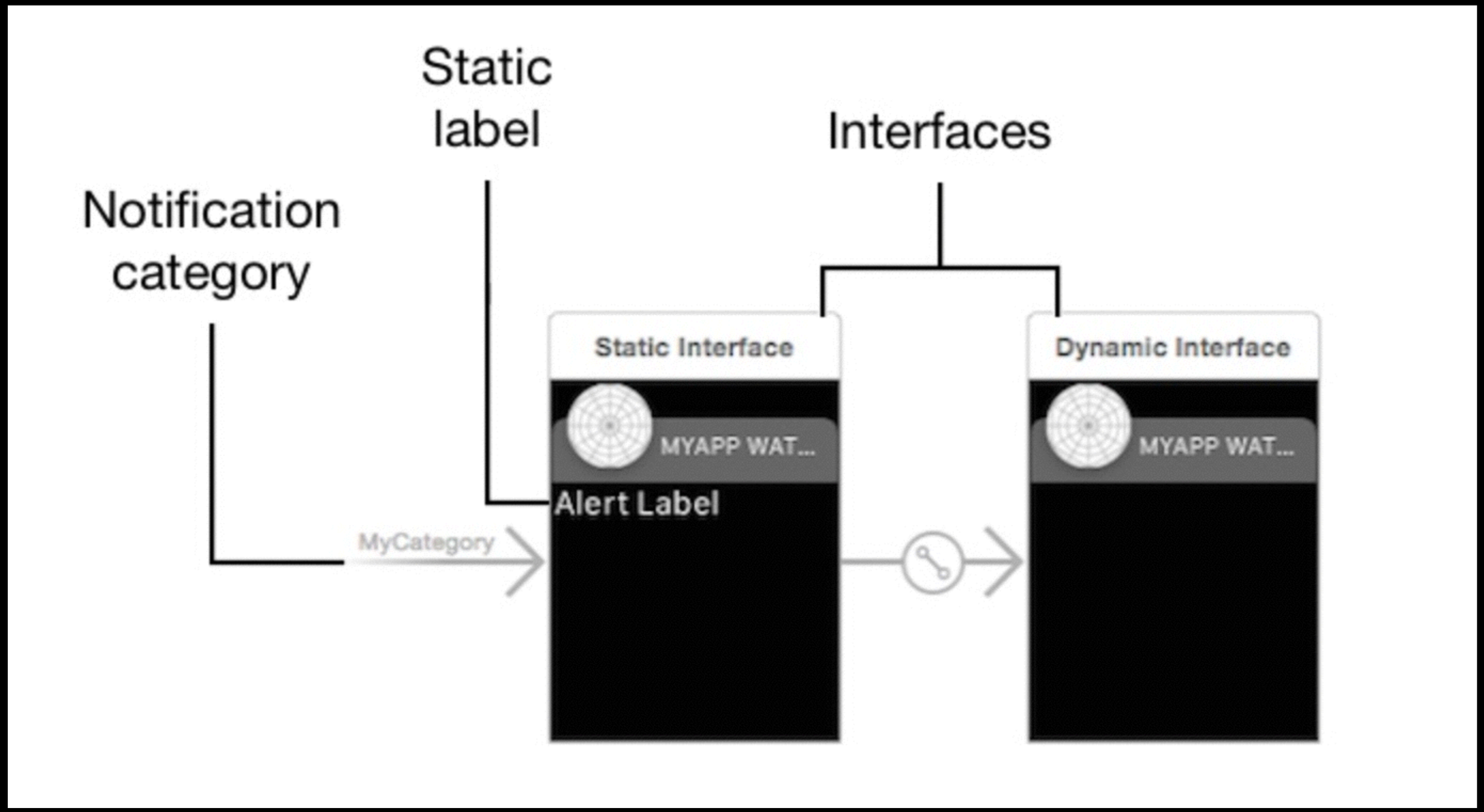
- For the foreground action, its important to note that it isn't your notification interface controller that processes the action, its the actual main interface controller of the watch kit app.
- So it needs to implement handleActionWithIdentifier:forPushNotification: and handleActionWithIdentifier:forLocalNotification:
- For background, information about the notification is passed to the app delegate, so be sure to implement application:handleActionWithIdentifier:ForRemoteNotification:completionHandler: or the local version of that same method.

Demo

Customizing your notification interface

- A custom notification interface consists of two separate interfaces: one static and one dynamic
- The dynamic interface is a fully customized and can contain custom artwork and content provided by your extension
- The static interface is more simple, and only contains the notification's alert message, static images, and text you configure at design time.
- Each notification interface must have an assigned notification type that tells Apple Watch when to use it.

Customizing your notification interface



Demo

Hash Tables

Hash Tables

- A hash table is a data structure that can map keys to values. A dictionary is a hash table.
- The key component to a hash table is a hash function. Given a key, the hash function computes an index to store the values in a backing array.
- On average, hash tables have an O(1) constant time look up. It's amazing!
- Ideally the hash function will assign each key to unique index, but usually in practice you have collisions. In this case we use our old friend linked list to help us out.

Hash Tables Workflow

- This is what happens when you add a value to a hash table/dictionary:
    1. The key passed in with the value to add is run through the hash function, which returns an index number
    2. The hash table creates a new bucket, with the value and key
    3. The hash table goes to the index of its backing array which matches the index returned from the hash function, and inserts the bucket at the head of the linked list of buckets.

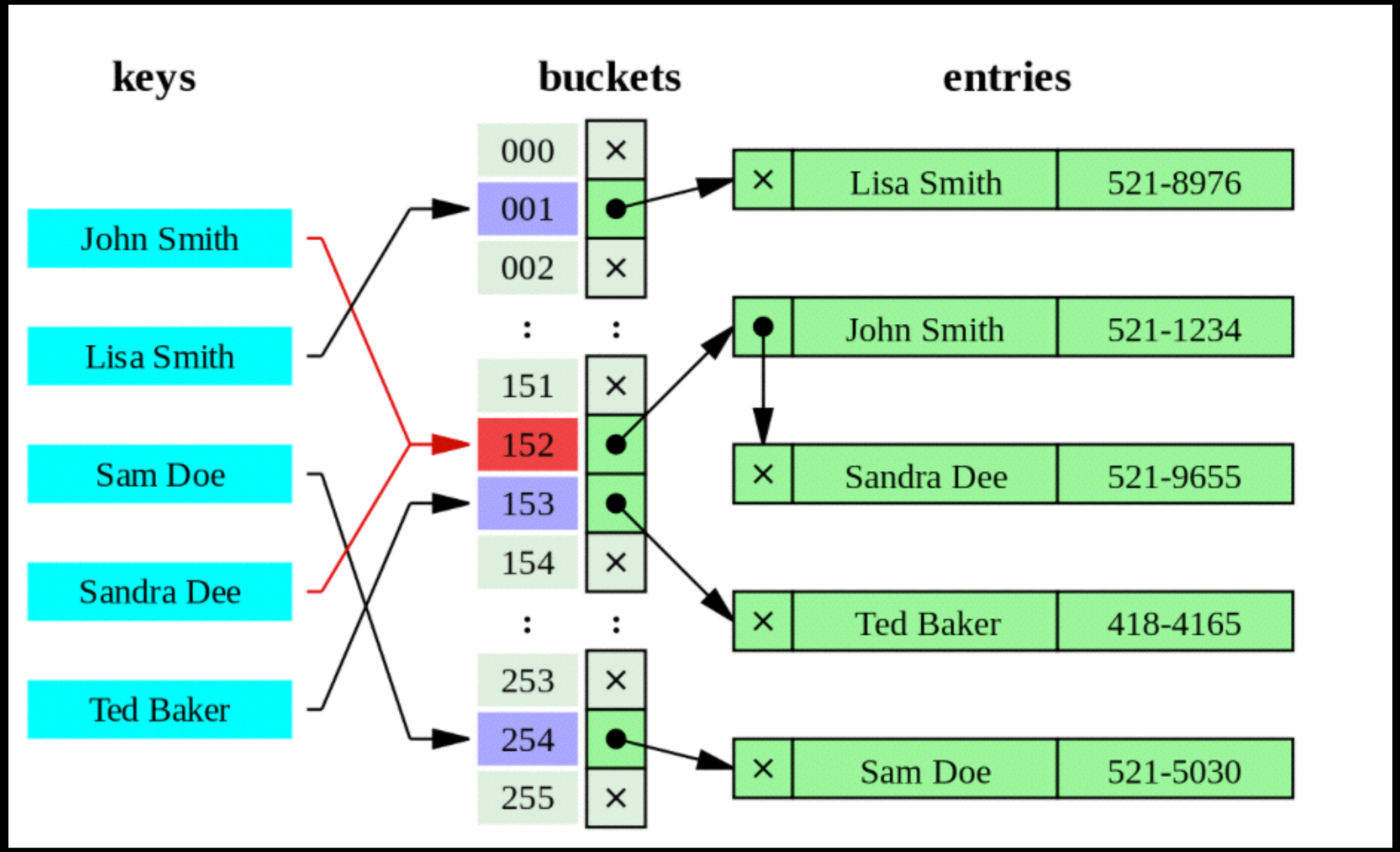
Hash Tables Collisions

- Whenever you have more than one bucket in the linked list at each index of the backing array, that is considered a collision.
- Collisions are bad, because it takes the lookup time of a Hash Table from O(1) constant time to O(n) linear time. :(
- It’s linear because you have to search through the linked list of buckets to find the exact key value pairing you were looking for, and worst case its at the end of the linked list.
- Better hash functions improve the rate of collisions, but they are never completely eliminated.

Hash Tables Workflow

- This is what happens when you try to retrieve a value for a key from a hash table:
    1. The key passed in with the value to add is run through the hash function, which returns an index number
    2. The hash table traverses through the linked list of buckets at that index of its backing array, and returns the value if it finds a bucket with the given key.
    3. If you have no collisions, this is constant time, if you have collisions its linear time.

Hash Tables


keys
buckets
entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

Modulus Operator

- We are going to use the modulus operator in our simple hash function.
- The modulus operator finds the remainder of division of one number by another.
- so 10 % 3 is 1 because 3 goes into 10 3 times, and then a 1 is left.

Demo