App Extensions

App Extensions

App Extensions

- “An app extension lets you extend custom functionality and content beyond your app and make it available to users while they’re using other apps or the system”
- Started with iOS 8.0 and OS X v10.10
- Several types of app extension ‘points’

App Extension Types

- Today Widgets: Give a quick update in the Today view of notification center
- Share: Share content from your app
- Action: work with content originating in a host app
- Photo Editing (iOS only): Edit a photo or video within photos app
- Finder Sync (OS X): Present information about file sync state directly in Finder
- Document Provider (iOS only): Provide access to and manage a repository of files.
- Custom Keyboard (iOS only): Replace the iOS system keyboard with a custom keyboard for use in all apps

App Extensions

- Although an app extension must be contained inside of an app's project in Xcode, it is a separate binary that executes independently from your app
- Creating an extension creates a new target that sits side by side with your app's target.
- So your extension has separate build settings and files, just like any other target.
- You can add multiple extensions to a single app!

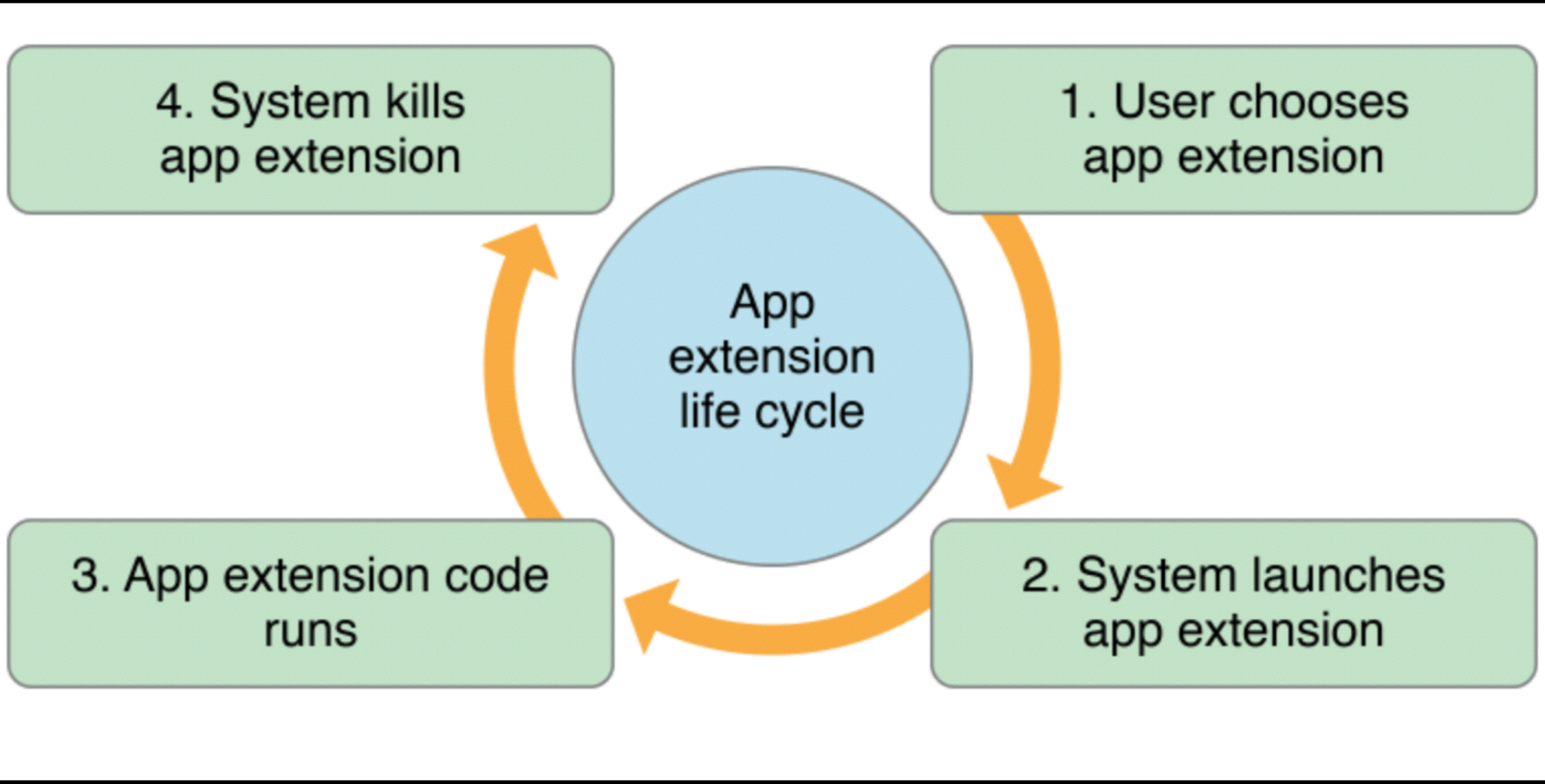
App Extension release

- To release an app extension, it must be submitted to the app store via submission of its containing app.
- Once the user installs the containing app, the app extension(s) is installed with it.
- The user must take additional action to enable the extension. Usually the user will be able to enable the extension within the context of the task the extension was designed for.
- Users can also go to the settings app of their device to enable it as well.

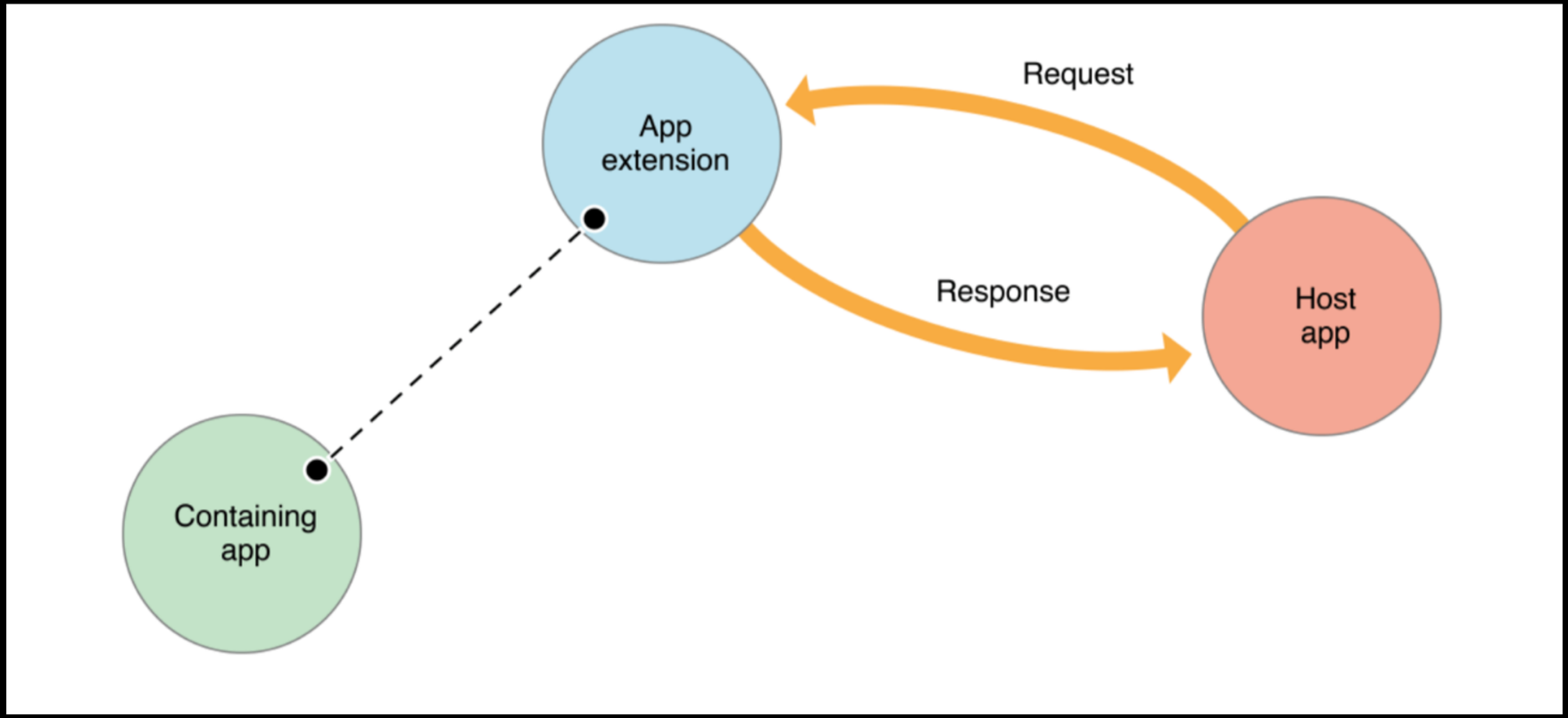
App Extension life cycle

- An app that a user employs to choose an app extension is called a **host app**.
- The host app provides an **extension context** to the extension and kicks off the extensions life cycle via sending a request in response to the user's action.
- The extension will usually terminate after the request is complete

App Extension life cycle



App Extension communication



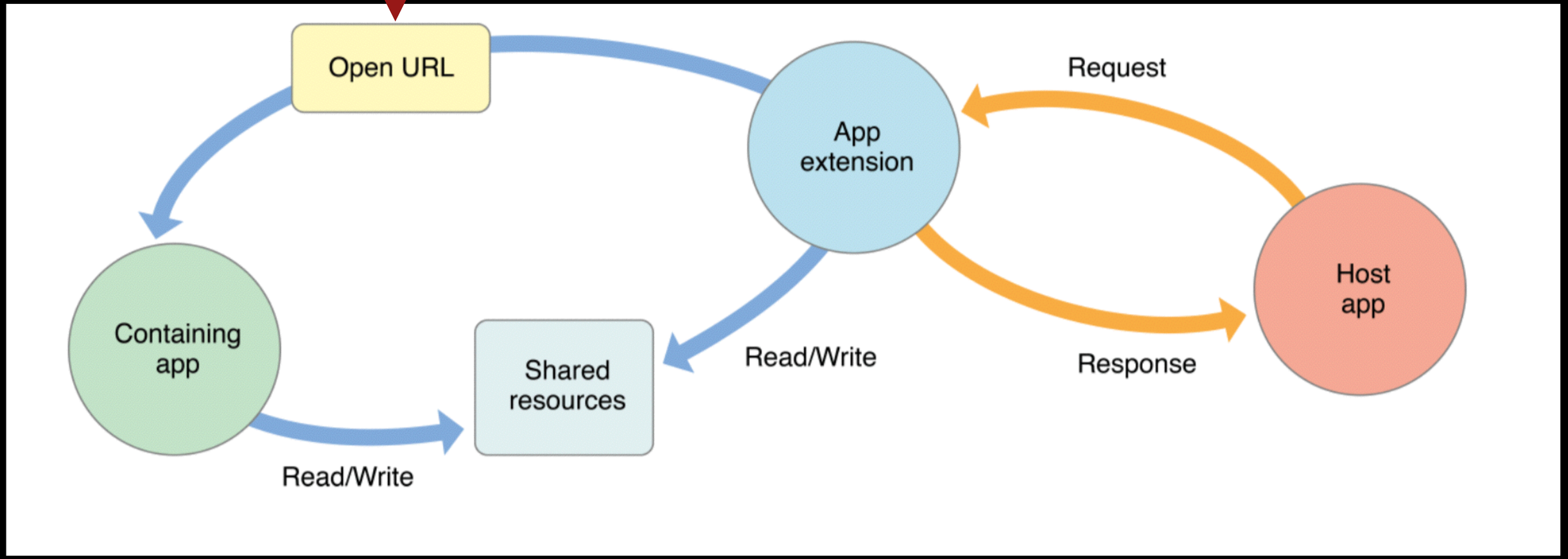
- The app extension primarily communicates with the host app (which is different from the containing app!)
- The format of the communication is: request generated by host, response generated by extension
- There is no direct communication between the extension and the containing app, typically the containing app isn't even running

App Extension

- Although the app extension cannot communicate directly with its containing app (the app it was submitted with) it can:
    1. access shared data with the containing app in a privately defined shared container
    2. In a today widget only, can open the containing app by calling `openURL:completionHandler:` method of the NSExtensionContext class

App Extension

(only in today widget!)



App Extension Limitations

- An extension cannot:
  - Access the sharedApplication object
  - Use any API marked in the header file with NS_EXTENSION_UNAVAILABLE (ex: HealthKit and EventKit)
  - Access the camera or microphone
  - Perform long running background tasks (although they can use an NSURLSession object to initiate downloads or uploads..)
  - Receive data using air drop (but can send)

Creating your extension

- Begin by choosing the most suitable app extension type (or ‘point) that fits with what you are trying to accomplish
- Make this choice wisely (or don’t and just use trial and error) because it determines which APIs are available to your app extension, and in some cases how those APIs behave.
- Once you are ready, add a new target to your app and choose your extension point. (File > New > Target)
- Building an app extension target creates a bundle with the file type of .appex

What comes in the app extension template?

- Your new app extension comes with an Info.plist, a view controller class, and a default user interface storyboard.
- The view controller will contain stubs for extension point methods you need to implement
- The info.plist identifies which type of extension you chose and specific details about your extension.

Important info.plist keys

- NSExtensionAttributes: a dictionary of extension point-specific attributes, such as which media types your photo editing extension supports
- NSExtensionPrincipalClass: The name of the view controller class your extension will instantiate when a host app launches your extension
- NSExtensionMainStoryboard (iOS only): the default storyboard to use, usually called MainInterface

NSExtensionContext

- When a host app sends a request to an app extension, it passes in an extension context.
- It's sort of like an NSNotification, it contains the relevant data the extension needs to perform its intended task.
- Your principle view controller has a property called extensionContext which provides access to the passed extension context from the host app
- The context has a property called inputItems, which is an array of NSExtensionItems your app extension needs to use.

NSExtensionItem

- NSExtensionItem contains a number of properties that describe the item:
  - title: an optional title for the item
  - content text: an optional string describing the extension item content
  - attachments: an optional array of media data associated with the extension item
  - user info: an optional dictionary of keys and values corresponding to the extension’s item’s properties

Completing the action

- Depending on the user's choice, you can:
  - Complete the request by calling completeRequestReturningItems:completionHandler: method on the extension context
  - Cancel the request by calling cancelRequestWithError: on the context and return an error code if need be

General Extension Workflow

1. Inspect the extension context and prepare your UI on your principle view controller appropriately
2. User interacts with your extension to perform a task
3. completeRequest or cancelRequest is called based on the user's choices.

Today Widget

- Today Widgets: Give a quick update in the Today view of notification center
- In order for the system to periodically update your today widget's view, be sure your view controller conforms to NCWidgetProviding protocol
- You then implement the method widgetPerformUpdateWithCompletionHandler:. In this method try to update your widgets view, and then call the completionHandler with one of the following constants:
  - NCUpdateResultNewData
  - NCUpdateResultNoData
  - NCUpdateResultFailed

Demo

Photo Extension

- Photo Editing (iOS only): Edit a photo or video within photos app
- Photos app always keeps a version of the original so the user can always revert back
- For each asset, Photos stores the original version, the current version, and adjustment data
- Your view controller that will be used for the extension must adopt the PHContentEditingController protocol

PHContentEditingController Protocol

- 4 required methods:
- startContentEditingWithInput:placeholderImage: - This method is called by the Photos app on your extension before your extension appears. The input parameter is of type PHContentEditingInput, which contains properties for the original image's URL, adjustment data, and meta data of the image
- finishContentEditingWithCompletionHandler: - Photos calls this method when the user chooses to end the editing session. This is time for you to make your final render
- shouldShowCancelConfirmation - Decides if you should show a prompt when canceling
  - cancelContentEditing - Can be called at anytime, clean up any resources

Demo