iOS Dev Accelerator
Week 3 Day 1

- UICollectionView
- Photos Framework Intro

UICollectionView

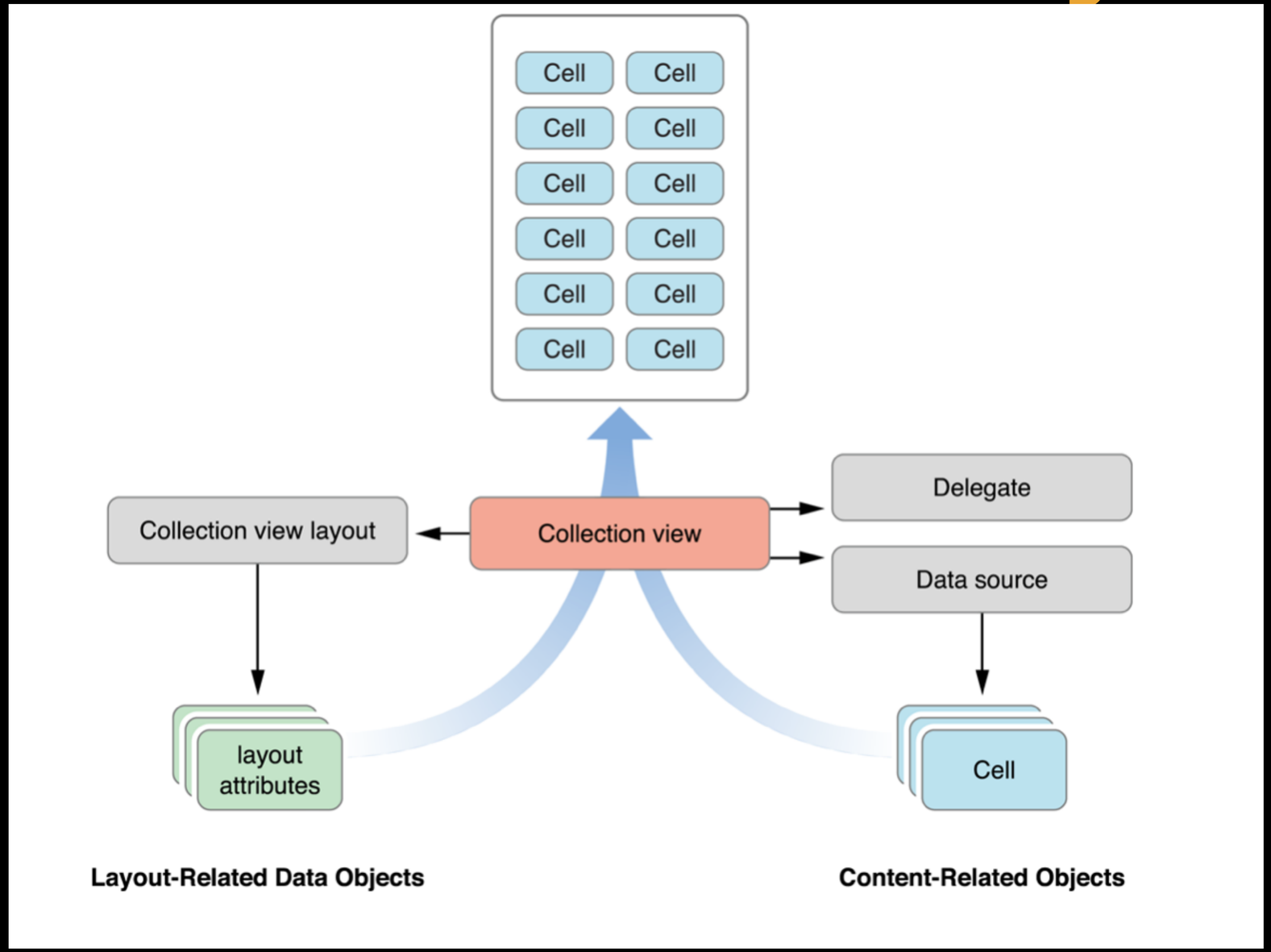
UICollectionView

- “A collection view is a way to present an ordered set of data items using flexible and changeable layout”
- Most commonly used to present items in a grid-like arrangement. (Items to collection views are as rows to table views)
- Creating custom layouts allow the possibility of many different layouts (grids, circular layouts, stacks, dynamic)

Internal WorkFlow

- You provide the data (datasource pattern!)
- The layout object provides the placement information
- The collection view merges the two pieces together to achieve the final appearance.

Collection View Objects



Reusable Views

- Collection views employ the same recycle program that table views do. (Same Queue data structure)
- 3 reusable views involved with collection views:
    1. Cells : Presents the content of a single item
    2. Supplementary views : headers and footers
    3. Decoration views : wholly owned by the layout object and not tied to any data from your data source. Ex : Custom background appearance.
- Unlike tableviews, collection view imposes no styles on your reusable views, they are for the most part blank canvases for you to work with.

CollectionViewDataSource

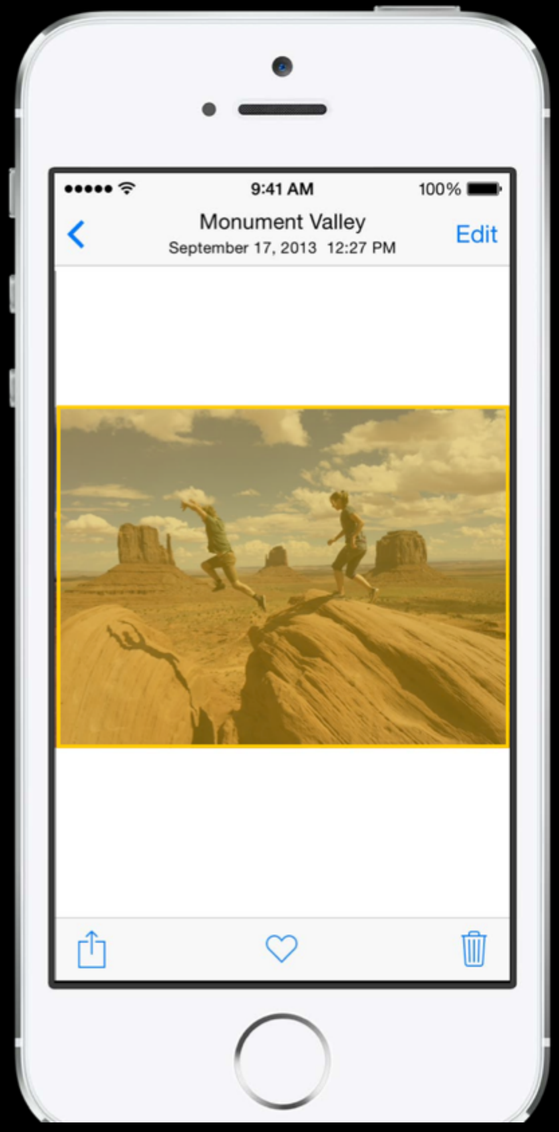
- Very similar to tableview's datasource. It is required.
- Answers these questions:
  - How many sections does the collection view contain?
  - For a given section, how many items does a section contain?
  - For a given section or item, what views should be used to display the corresponding content?

Photos Framework

- New Apple Framework that allows access to photos and videos from the photo library.
- Also used for creating photo editing app extensions, a new feature with iOS8
- First-class citizen, you can create a full-featured photo library browser and editor on par with Apple's Photos App.
- Intended to supersede ALAssetsLibrary

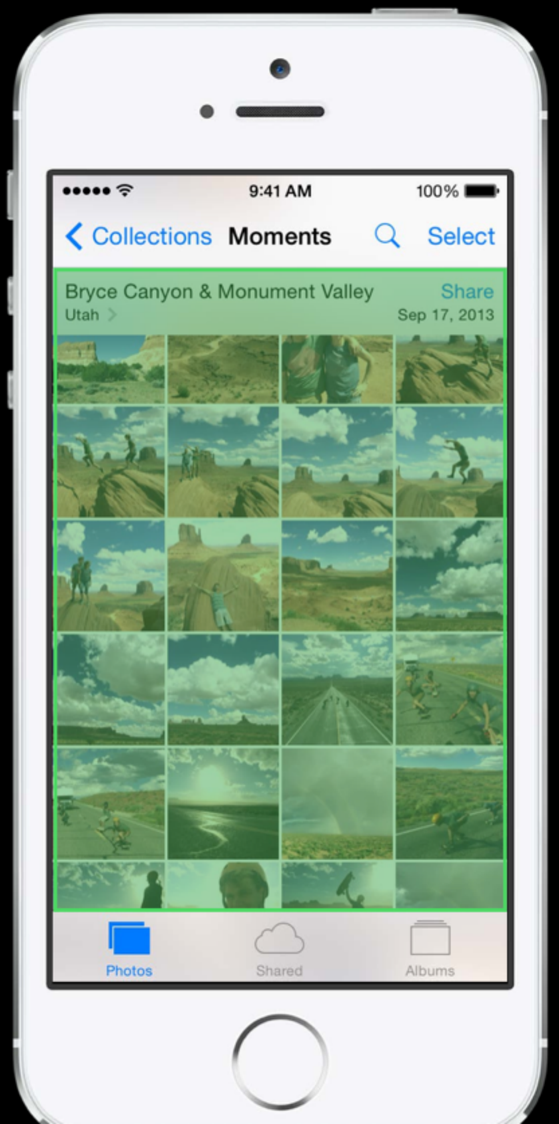
PHAsset

- The Photos framework model object that represents a single photo or video.
- Has properties for Media type, Creation date, Location, and Favorite.



PHAssetCollection

- A Photos framework model object representing an ordered collection of assets.
- Albums, moments, and smart albums.
- Has properties for Type, Title, and Start and End Date.



PHCollectionList

- A Photos framework model object representing an ordered collection of collections.
- This can be a folder, moment, or year
- Has properties for Type, Title, and Start and End Date.

Fetching Model Objects

- You fetch via class methods on the models:
  - PHAsset.fetchAssetsWIthMediaType(PHAssetMediaType.Photo, options:nil)
  - PHAssetCollection.fetchMomentsWithOptions(nil)
- Collections do not cache their contents, so you still have to fetch all the assets inside of it. This is because the results of a fetch can be very large, and you dont want all of those objects in memory at once.

PFFetchResult

- Results returned in a PHFetchResult
- Similar to an Array.

Making Changes

- You can favorite a photo and add an asset to an album
- You cannot directly mutate an asset, they are read only (thread safe!)
- To make a change, you have to make a change request.
- Theres a request class for each model class:

```
PHAssetChangeRequest
PHAssetCollectionChangeRequest
PHCollectionListChangeRequest
```

Making Changes

```
func toggleFavorite(asset : PHAsset) {

    PHPhotoLibrary.sharedPhotoLibrary().performChanges({

        //create a change request object for the asset
        var changeRequest = PHAssetChangeRequest(forAsset: asset) as
      PHAssetChangeRequest
        //make your change
        changeRequest.favorite = !changeRequest.favorite

        }, completionHandler: { ( success : Bool,error : NSError!) -> Void in

        //asset change complete
        })

}
```

Making New Objects

Create via creation request

```
var request = PHAssetChangeRequest.creationRequestForAssetFromImage(UIImage())
```

Placeholder objects

```
var placeHolder = request.placeholderForCreatedAsset
```

- Reference to a new, unsaved object
- Add to collections
- Can provide unique, persistent `localIdentifier`

Getting to the actual data

- Many different sizes of an image may be available or different formats of a video
- Use PHImageManager to request images/videos
- Request an image based on target size for displaying
- Request a video based on the usage
- Asynchronous API, because you dont know how long it will take to load the data, it could be very expensive
- Will optionally retrieve the data from the network if its only on iCloud

Requesting an Image

```
let manager = PHImageManager.defaultManager()

manager.requestImageForAsset(photo,
                             targetSize: cellSize,
                             contentMode: PHImageContentMode.AspectFill,
                             options: nil,
                             resultHandler: {(result : UIImage!, [NSObject : AnyObject]!) -> Void in
                                 if result {
                                     imageView.image = result
                                 } else {
                                     //tell user something went wrong
                                 }
    })
```

Advanced Image Request

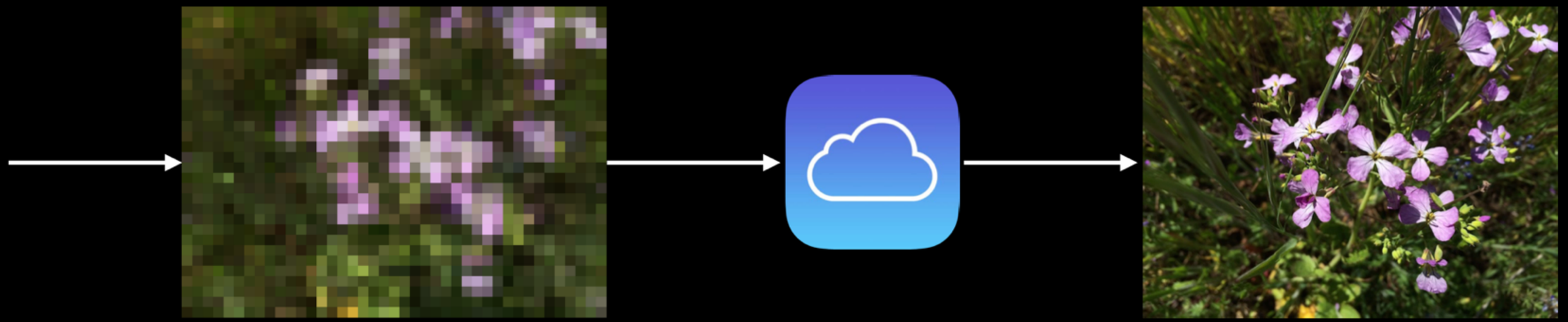
```
var options = PHImageRequestOptions()

options.networkAccessAllowed = true
options.progressHandler = {(progress : Double, error : NSError!, degraded : UnsafePointer<ObjCBool>,
[NSObject : AnyObject]!) in
    //update visible progress UI
    }

//use your options to control the request behavior
manager.requestImageForAsset(photo,
    targetSize: cellSize,
    contentMode: PHImageContentMode.AspectFill,
    options: options,
    resultHandler: {(result : UIImage!, [NSObject : AnyObject]!) -> Void in
```

Advanced Image Request

```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```



First callback synchronous

Second callback asynchronous