iOS Dev Accelerator Week1 Day4

- Lazy Loading
- TableView HeaderView
- Nib’s/Xib’s
- Image Assets
- Resizing Images
- Queue Data Structure

Homework Review

Lazy Loading

Lazy Loading

- “Lazy Loading is a design pattern commonly used in programming to defer initialization of an object until the point at which it is needed”
- In iOS, lazy loading is often time used in collection views and table views for deferring the loading of resources used by cells until the time those resources are actually needed.
- Swift has introduced a lazy keyword that can make properties lazy, so they wont be instantiated until they are accessed. (more on this later in the course)

Demo

Nib’s/Xib’s

Nib's/Xib's

- Nibs, or Xibs since Xcode 3.1, allowed developers to create interfaces graphically instead of programatically.
- Storyboard are now the best (and Apple recommended) way to create your interface, but xibs still serve a few purposes:
  - Create a single isolated view controller layout
  - Create the layout of a non-view controller related view (like a tableview cell!)
  - Work better with source control

How Xib's work

- Interface objects are what you add to a xib file to layout your interface.
- At runtime, the interface objects are instantiated by the system and inserted into your code.

Xib Lifecycle

1. The system loads contents of the nib file into memory
2. the nib's object graph is unarchived, with our old friend NSKeyedUnarchiver
3. All connections are re-established (outlets and actions)
4. awakeFromNib is sent to all appropriate objects (good place for setup code for your view)

When to use a Xib?

1. Use a Xib to layout a View Controller's interface in insolation (ie not in a storyboard)
2. Use a Xib to layout a Table View or Collection View cell in isolation (maybe because it will be used in multiple view controllers)
3. Use a Xib to layout a custom view that isn't created with a view controller

View Controller and Xib

1. Set your Xib's File's Owner as the View Controller's class (this allows you create the outlets and actions)
2. Use your view controllers constructor that takes in a nib name and bundle (this is inherited from UIViewController)

Demo

Custom Views and Xib

1. Drag out a view into your Nib
2. set its class to be your custom views class (this allows outlets!)
3. Use the NSBundle method loadNibNamed:owner:options: to load th nib file. This will return an array of all the root objects of the nib
4. Grab the appropriate object from the array (it will be the only object if your nib only holds one root view)

Cells and Xibs

1. Drag a table view or collection view cell into your Nib
2. Set the class of the cell to be your custom class (this allows you to drag outlets)
3. In viewDidLoad(), or some other appropriate setup method, call registerNib:ForCellReuseIdentifier: on your table view

Demo

TableViewHeaderView

The tableView's header view

- In addition to header views for each section, you can also have an accessory view that sits on top of the table itself.
- It's just a regular UIView
- Can be set in code by creating the view and setting it to the tableView's tableHeaderView property.
- Set in storyboard by just dragging a view on to the tableview.

Demo

Image Assets

Asset Catalogs

- “Use asset catalogs to simplify management of images that are used by your app”
- Things you can put in your asset catalogs:
  - Image sets
  - App icons
  - Launch Images

Image Sets

- Image sets contain all the versions of an image necessary for the different scale factors of different devices' screens. (ie retina vs non-retina vs retina HD)
- When you drag an image into the XCAsset, an image set will be created for it.
- You will see 1x(non retina), 2x (retina), and 3x (retina HD) slots for each image.
- Image sets can also be configured to be device or size class specific.

1x, 2x, and 3x

- Lets say we have an image view that is going to be constrained to 300 points height and 300 points width.
- We would need a 300x300 image for 1x, 600x600 image for 2x, and 900x900 image for 3x for the image displayed in the image view to scale properly for each device resolution.

1x, 2x, 3x

1x, 2x, 3x

Image size and optimization

- Prior to shipping, if you are using statically sized images and not vector images, it is vital you have correctly sizes images based on their desired point size in the interface.
- If you don’t, the system has to rescale the image for you, which greatly increases the memory footprint. A 30KB image could now take 1MB. It’s brutal.
- In code, you can always check what kind of screen you are working with (1x,2x, or 3x) by checking UIScreen.mainScreen().scale

Image naming

- Based on what device scale the image is for, append @2x or @3x to the end of the name of the image (but before the file type) to help xcode with correctly managing the image.
- Ex: retweet@2x.png and retweet@3x.png
- For non retina, aka @1x, you drop the @1x.

Demo

Resizing images

Resizing an Image

- Par for the course with programming, there are a number of different ways to resize an image in iOS
- NSHipster did a great article on a bunch of the different ways, and which way is the fastest.
- Here is the fastest:

```
1 var size = CGSize(width: 100, height: 100)
2 UIGraphicsBeginImageContext(size)
3 originalImage!.drawInRect(CGRect(x: 0, y: 0, width: 100, height: 100))
  let thumbnail = UIGraphicsGetImageFromCurrentImageContext()
  UIGraphicsEndImageContext()
```

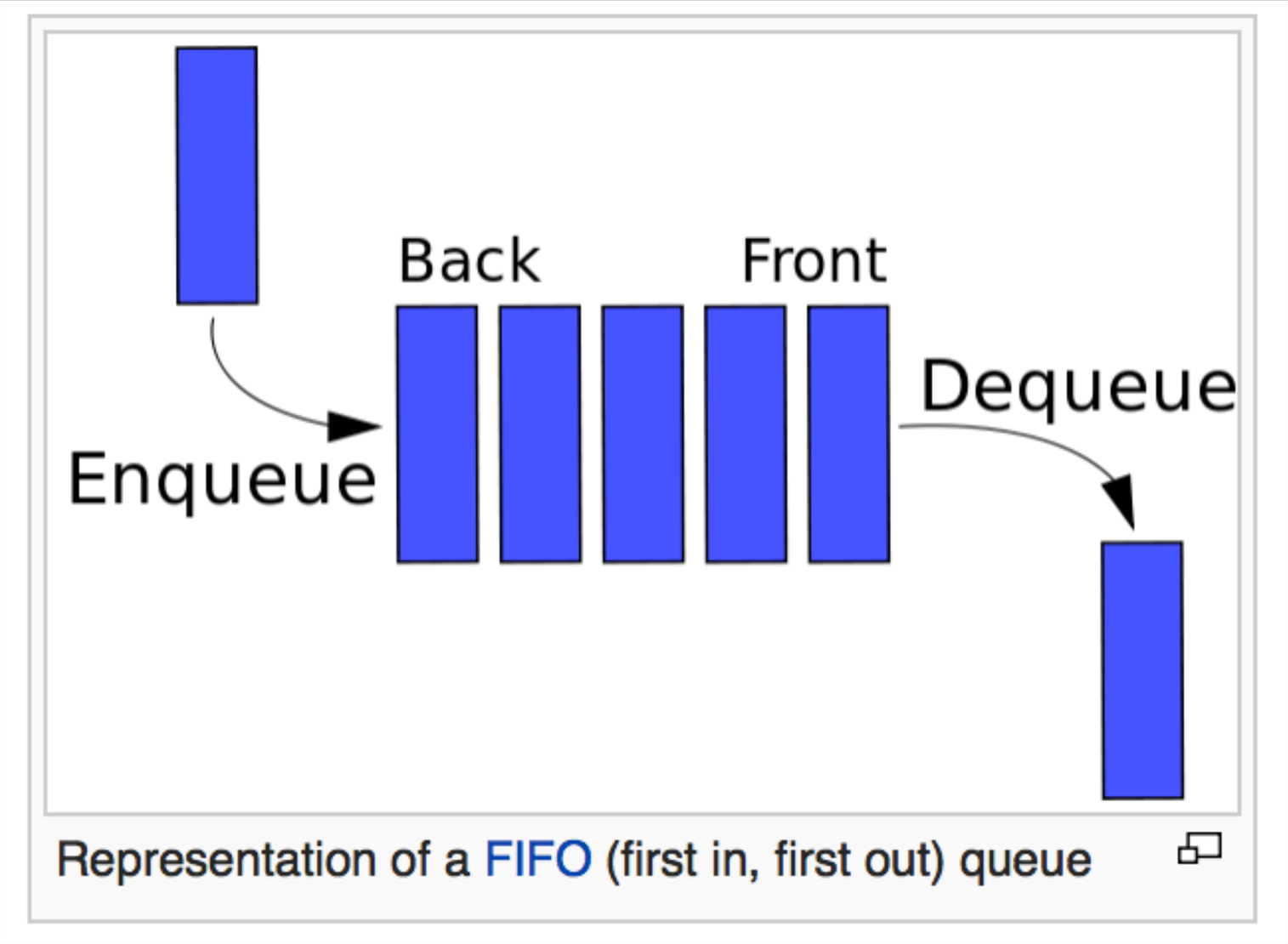
1. UIGraphicsBeginImageContext() creates bitmap-based graphics context for you to draw in and makes it the ‘current context’
2. drawInRect is a method on UIImage which draws the image in the specified rectangle, in the current context, and scales if needed
3. UIGraphicsGetImageFromCurrentImageContext() just pulls the currently drawn image from the context and returns a UIImage

Queue Data Structure

Queues

- From Wikipedia: “In computer science, a queue is a particular kind of data structure in which entities in the collection are kept in order and the principal operations on the collection are addition of entities to the rear (known as enqueue) and removal of entities from the front position (known as dequeue)”

Queue Visual



Representation of a FIFO (first in, first out) queue

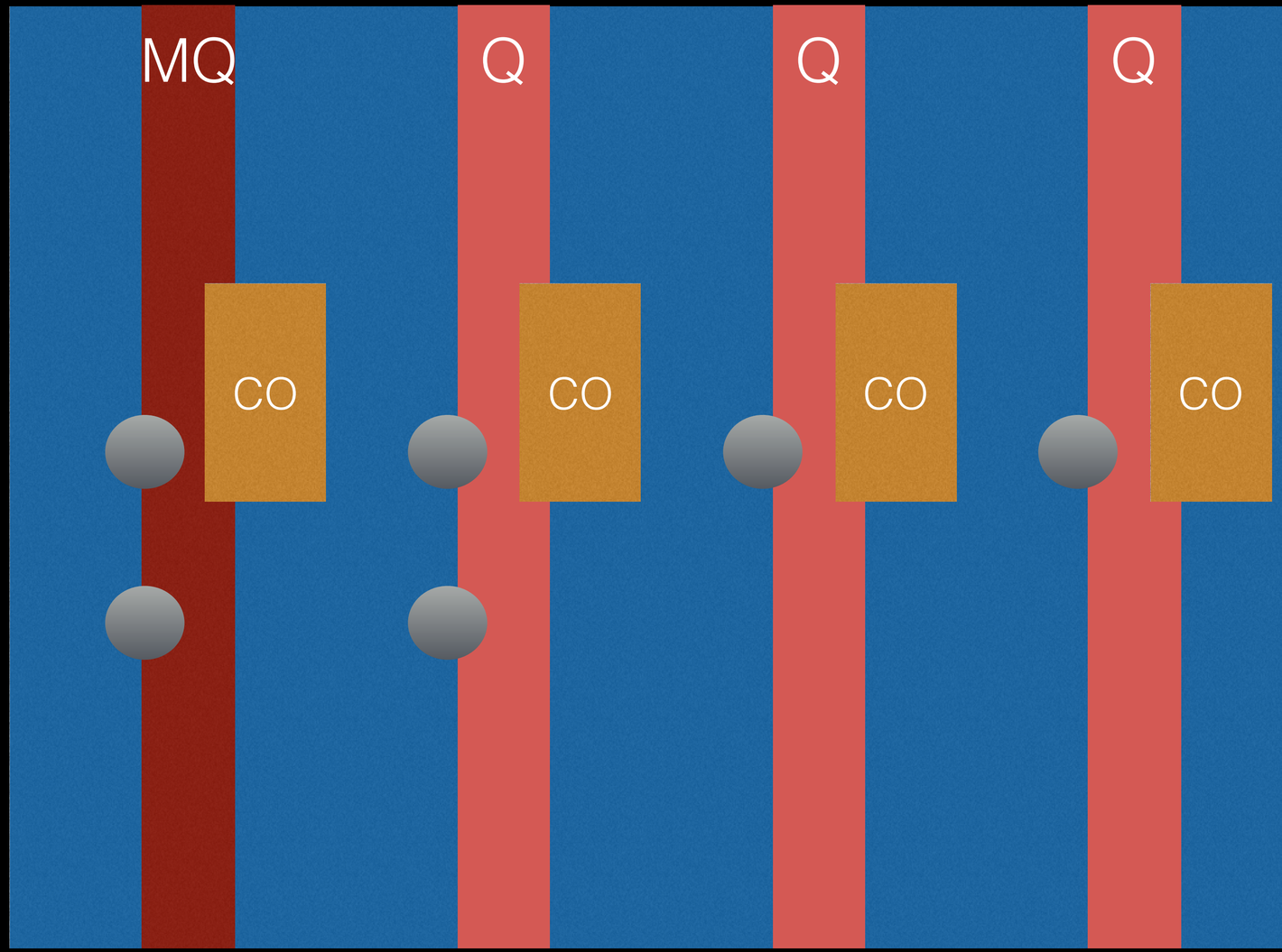
What are Queues used for?

- A queue is great for any situation where you need to **efficiently maintain a First-In-First-Out order of some grouping of entities.**
- Servers use queues as a fair policy when responding to requests. They might be getting 10,000's requests a second, and they respond to those requests in a first come first serve fashion using a queue.
- CPU's use queues to properly schedule operations to run based on priority

What are Queues used for?

- A queue is great for any situation where you need to **efficiently maintain a First-In-First-Out order of some grouping of entities.**
- Servers use queues as a fair policy when responding to requests. They might be getting 10,000's requests a second, and they respond to those requests in a first come first serve fashion using a queue.
- CPU's use queues to properly schedule operations to run based on priority

NSOperationQueue



Using Operation Queues, we can run code on other queues, not just the main queue.

Just like a grocery store,
the more lines that are open, the less time the customers need to wait.

Implementing a Queue

- Implementing a queue can be different based on what data structure you are using under the hood.
- Can be implemented using an array or linked list
- Just need to implement dequeue (removing) and enqueue (adding)
- Queues can have a peek operation as well (sometimes called front), which simply returns the front most entity.

Demo