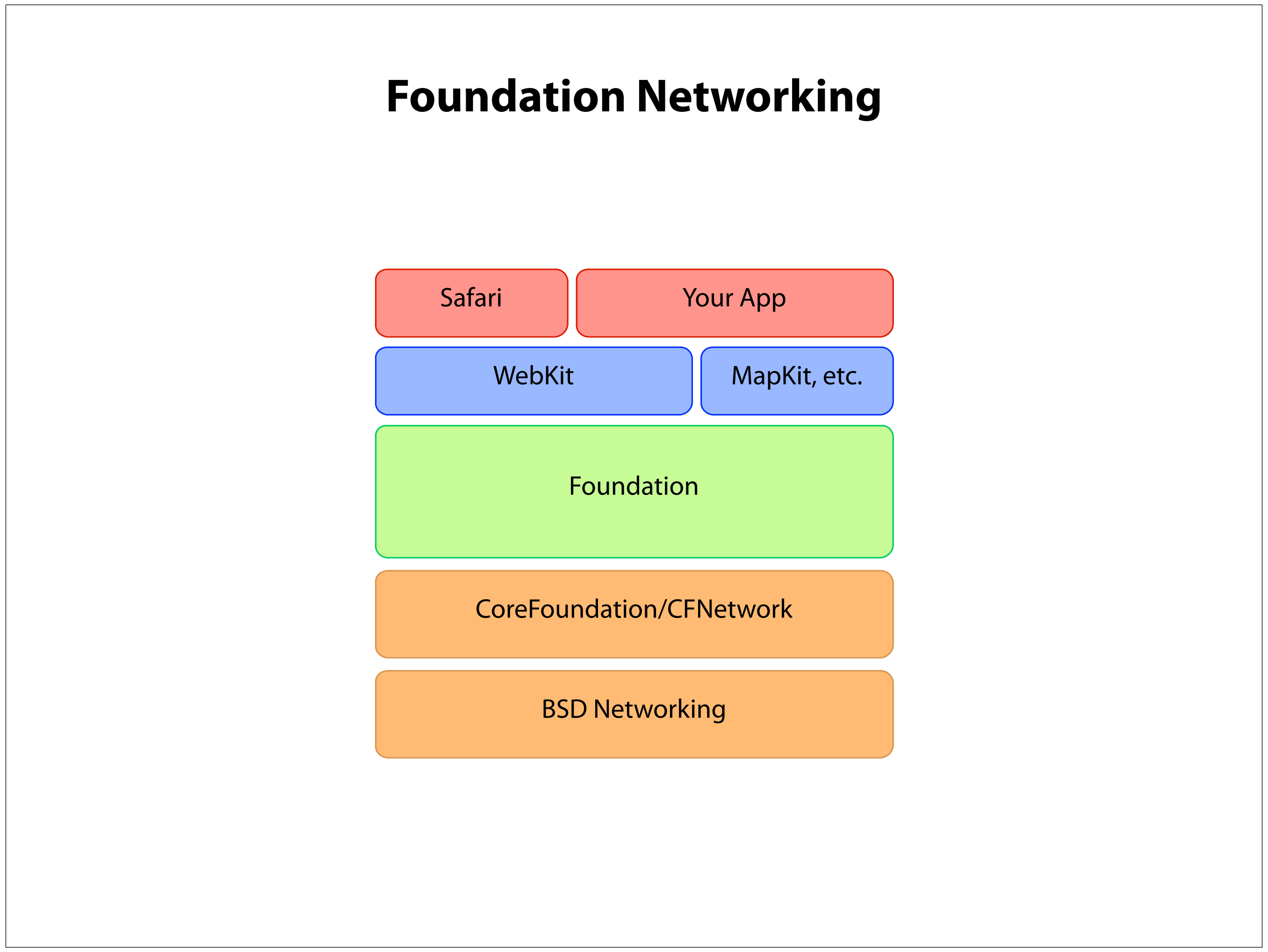
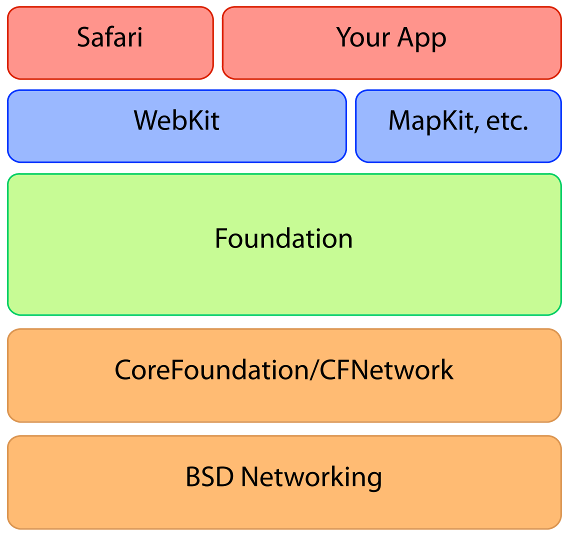
Networking with NSURLConnection

Alex Vollmer
http://alexvollmer.com
@alexvollmer

pluralsight
hardcore dev and IT training



Foundation Networking



The Foundation Framework is a feature-rich collection of APIs to use in your applications.

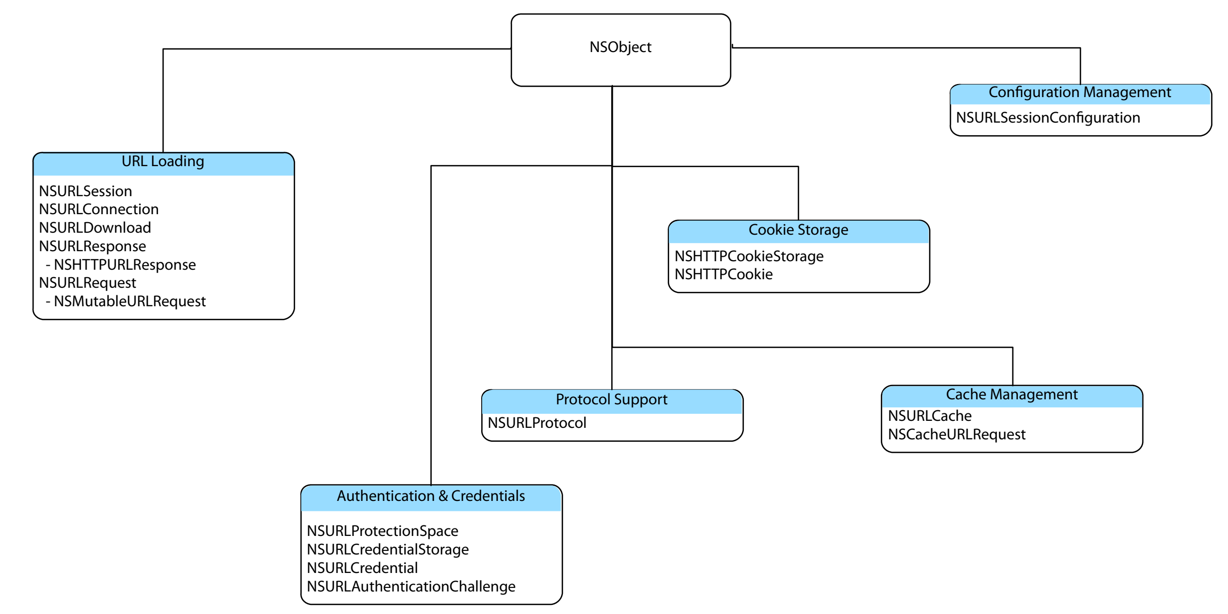
As far as networking goes, the technology stack starts at the bottom with good old-fashioned BSD sockets.

The CoreFoundation library and CFNetwork functions provide some higher-level abstractions with a C-based API.

The Foundation framework includes Objective-C equivalents to the functionality found in the CoreFoundation layer.

Layers like WebKit, MapKit, and Mobile Safari are built on top of the Foundation's networking layer. This is also where we'll be integrating networking into our sample application.

URL Loading System



The URL Loading System is a part of Foundation used for accessing resources via URL. This system is composed of Objective-C classes that help with Authentication & Credentials, Cache Management, Cookie Storage, Configuration Management and even custom URL protocol support.

URL Support

http://

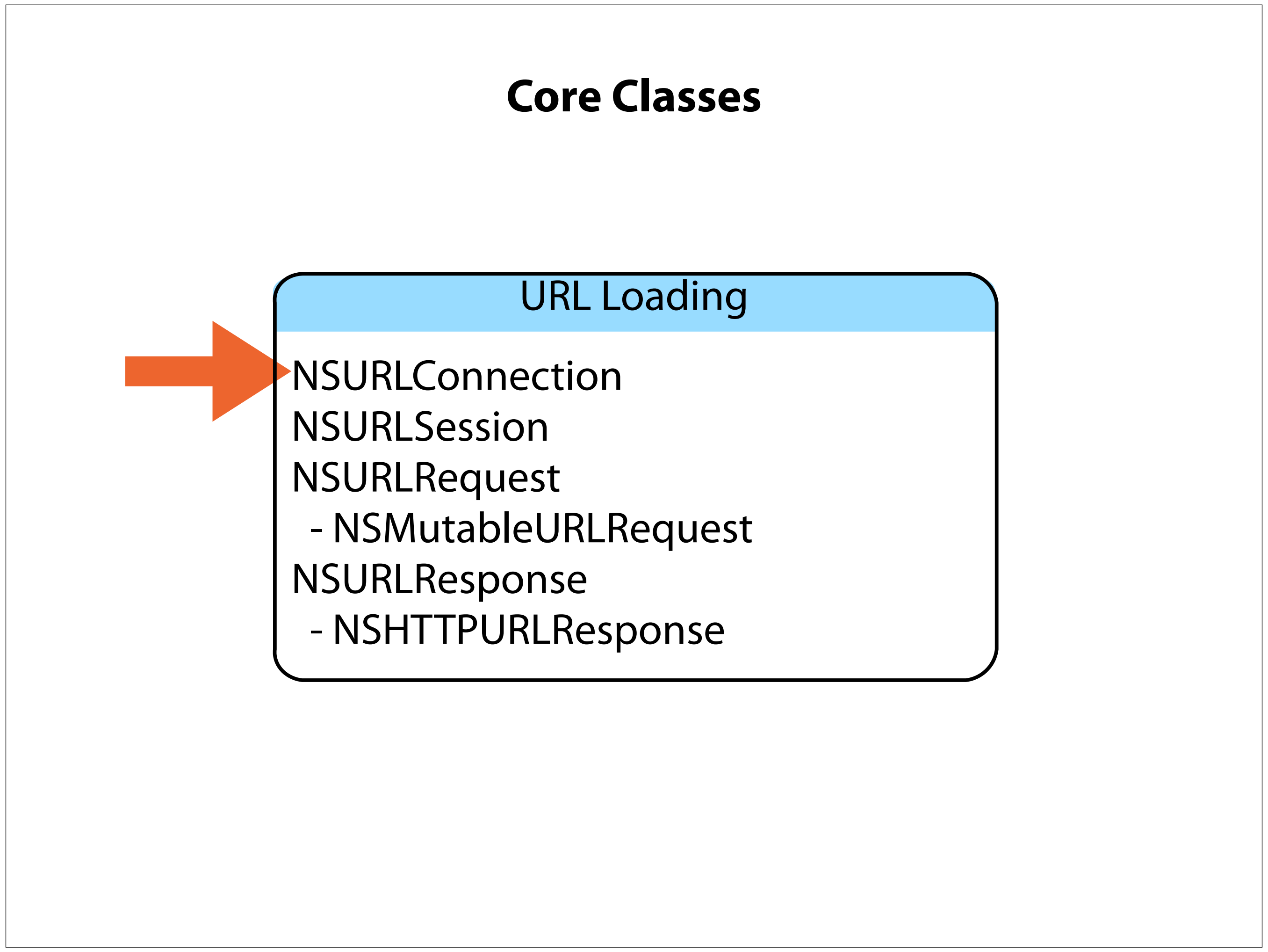
https://

ftp://

file:///

data://

By default, the URL Loading system supports these URL schemes…



Core Classes

URL Loading

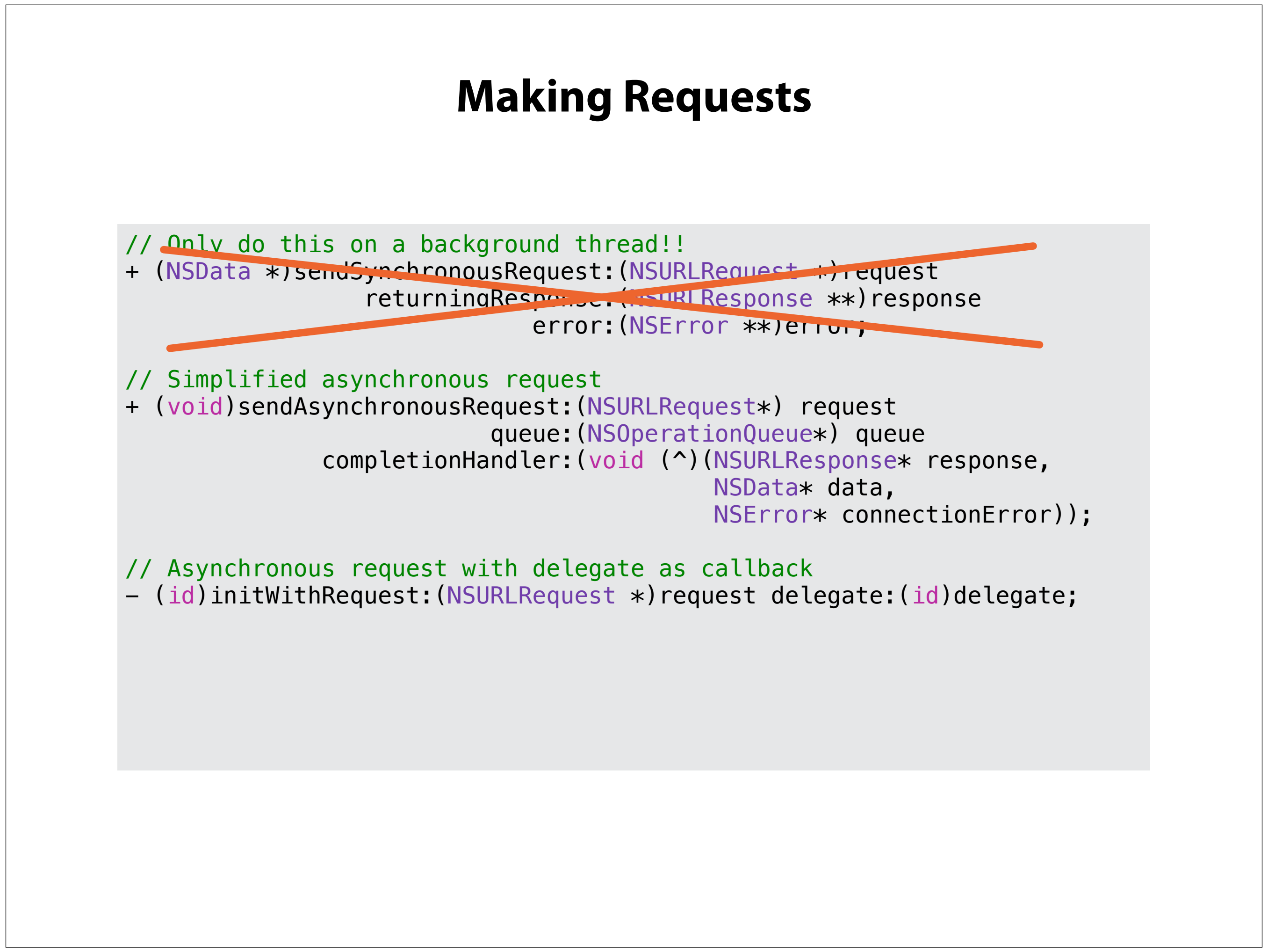
NSURLConnection
NSURLSession
NSURLRequest
 - NSMutableURLRequest
NSURLResponse
 - NSHTTPURLResponse

At the heart of the URL Loading system is the NSURLConnection class. This class represents a connection to a resource identified by a URL. In iOS 7 and OS X 10.9, the NSURLSession class was introduced which is the recommended API to use going forward.

Requests are made by instantiating and configuring an instance of NSURLRequest or NSMutableURLRequest.

Responses are encapsulated in NSURLResponse or NSHTTPURLResponse instances.

In this module we'll implement the networking layer using NSURLConnection. While NSURLSession is the more modern system, if you need to support older versions of iOS you need to know how this API works. It will also help you appreciate the design changes made with NSURLSession.



Making Requests

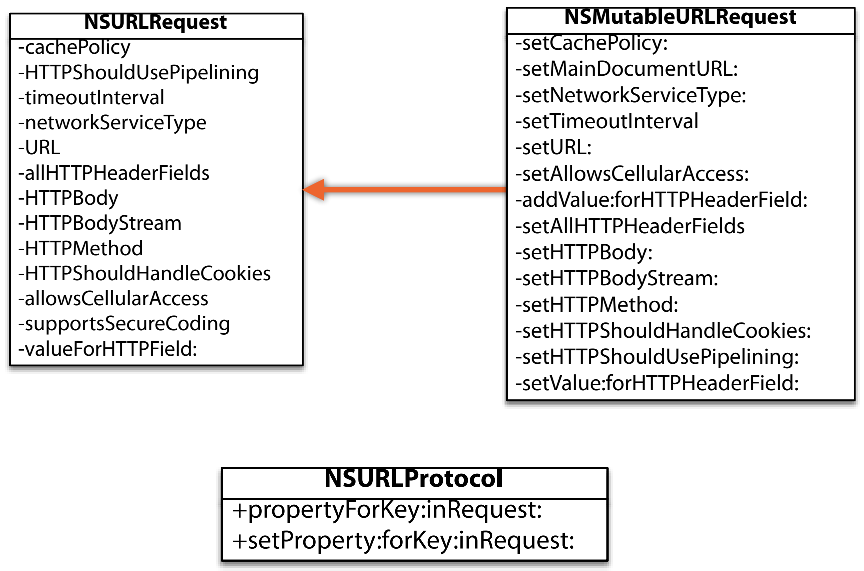
```
~~// Only do this on a background thread!!
+ (NSData *)sendSynchronousRequest:(NSURLRequest *)request
                 returningResponse:(NSURLResponse **)response
                             error:(NSError **)error;~~

// Simplified asynchronous request
+ (void)sendAsynchronousRequest:(NSURLRequest*) request
                          queue:(NSOperationQueue*) queue
              completionHandler:(void (^)(NSURLResponse* response,
                                          NSData* data,
                                          NSError* connectionError));

// Asynchronous request with delegate as callback
- (id)initWithRequest:(NSURLRequest *)request delegate:(id)delegate;
```

You have three basic ways to make a request using NSURLConnection. The first is to make the request synchronously using +sendSynchronousRequest:returningResponse:error:. Don't ever use this method on the main thread since it will block further execution until a response is returned. Since we almost never need to manage threads ourselves, we won't use this method.

To make asynchronous requests, you can use the +sendAsychronousRequest:queue:completionHandler: class method or you can alloc/init a NSURLConnection instance with a delegate and a NSURLRequest.

The first way is easier, but doesn't have all of the same functionality as the second option. These convenience methods don't allow you to customize behavior with a delegate, so the second option is what we will be using.

Requests

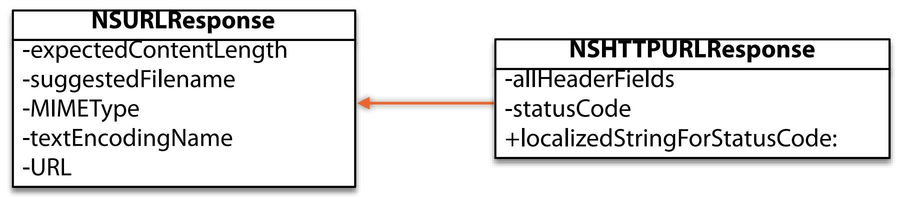


To retrieve a resource via URL, you need to create an instance of NSURLRequest. By default, this is an immutable object so if you need to customize the request beyond the initial URL and cache policy, create an instance of NSMutableURLRequest.

Note that the request object supports several properties that are specific to HTTP despite the fact that the URL system can support other protocols. The NSURLRequest class can be extended in a key-value coding style by using the -propertyForKey:inRequest: and -setProperty:forKey:inRequest: in the NSURLProtocol class.

For our purposes, we won’t need to worry about this customization.

Responses

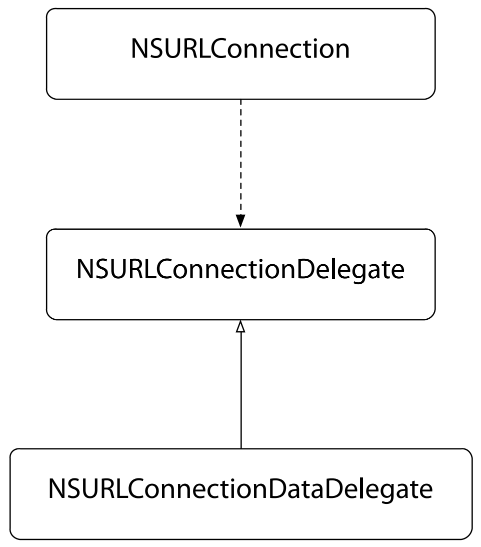


Responses are encapsulated within instances of the NSURLResponse class. The Foundation framework provides the NSHTTPURLResponse which is a specialized subclass of NSURLResponse. Anytime you make a request via HTTP, you will get an instance of NSHTTPURLResponse.

There are a couple of things to note here. Unlike the request objects, the HTTP specifics are modeled in a specific sub-class. The other important point to notice is that the response objects only describe *envelope* data, not *body* content.

To access the body of a response requires some interaction with the NSURLConnection's delegate.

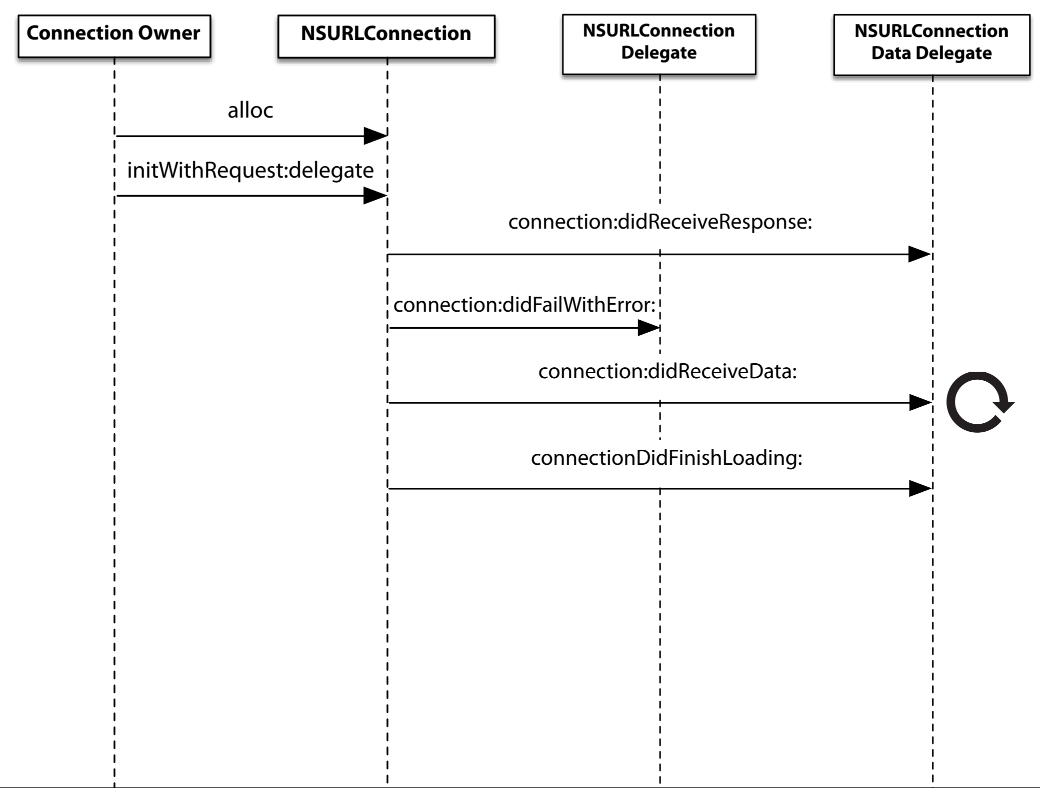
NSURLConnection & Delegate



When you make a request with a NSURLConnection class, you typically instantiate it with a delegate that implements the NSURLConnectionDelegate protocol. If you need further interaction with the connection, your delegate can implement any of the optional methods from the NSURLConnectionDataDelegate protocol.

To access the body of a response, you need to implement a handful of optional methods from the NSURLConnectionDataDelegate protocol.

Data Loading



Connection Owner
NSURLConnection
NSURLConnection Delegate
NSURLConnection Data Delegate
alloc
initWithRequest:delegate
connection:didReceiveResponse:
connection:didFailWithError:
connection:didReceiveData:
connectionDidFinishLoading:

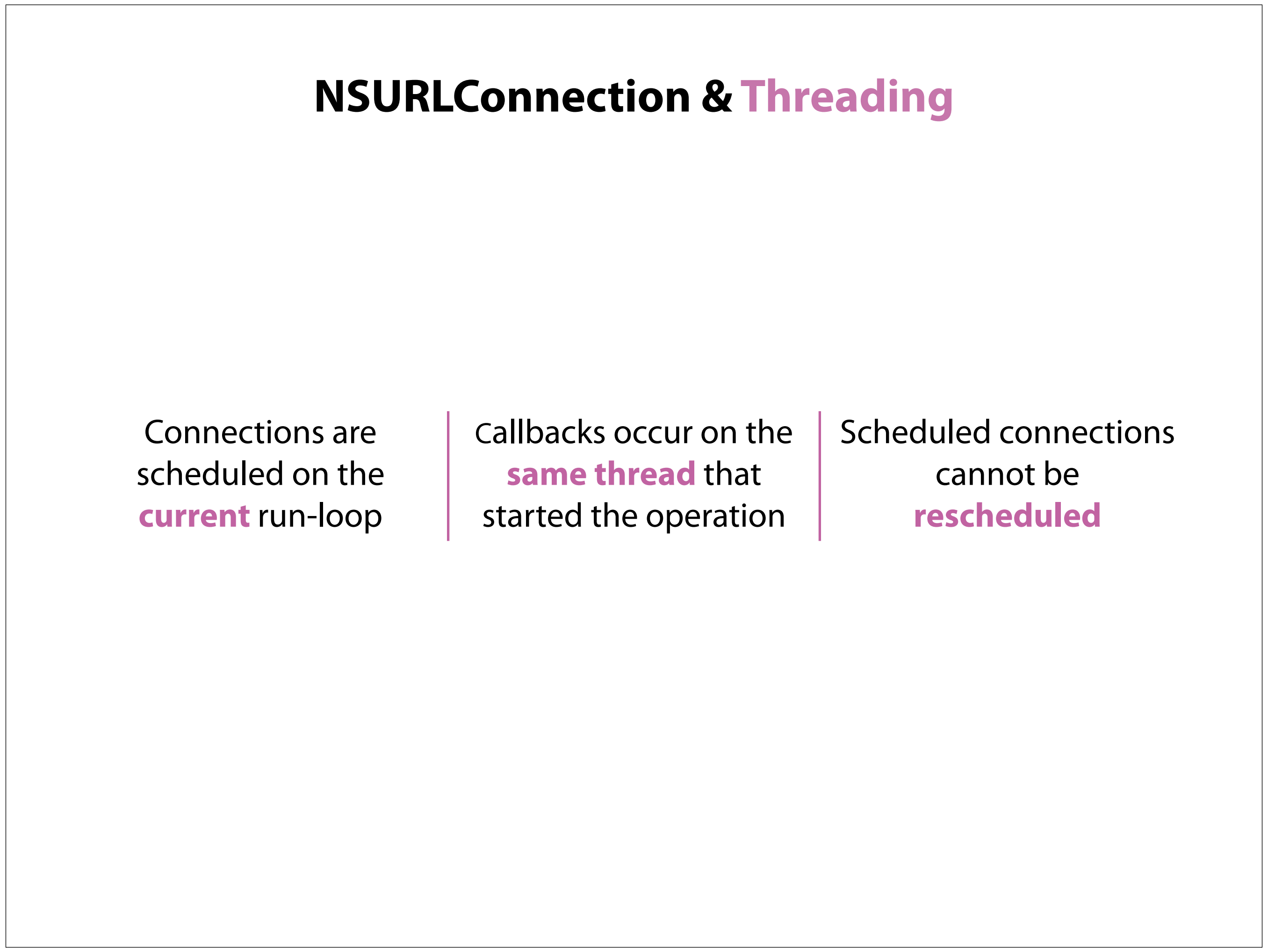
When you make a request with NSURLConnection, you first need to create a new instance and designate its delegate. By default, this initializer will immediately start the request.

When the request first gets envelope response information for things like HTTP headers and status code the -connection:didReceiveResponse: delegate method will be called with an instance of NSURLResponse.

If the connection fails for some reason like losing network connectivity, the -connection:didFailWithError: method is called with a NSError instance.

If you expect to handle the body of a response, implement the -connection:didReceiveData: delegate method. Note that this method may be called several times as data arrives in chunks so it’s your responsibility to handle this.

Once the response has finally been retrieved the -connectionDidFinishLoading: delegate method is called.



NSURLConnection & Threading

Connections are scheduled on the **current** run-loop

Callbacks occur on the **same thread** that started the operation

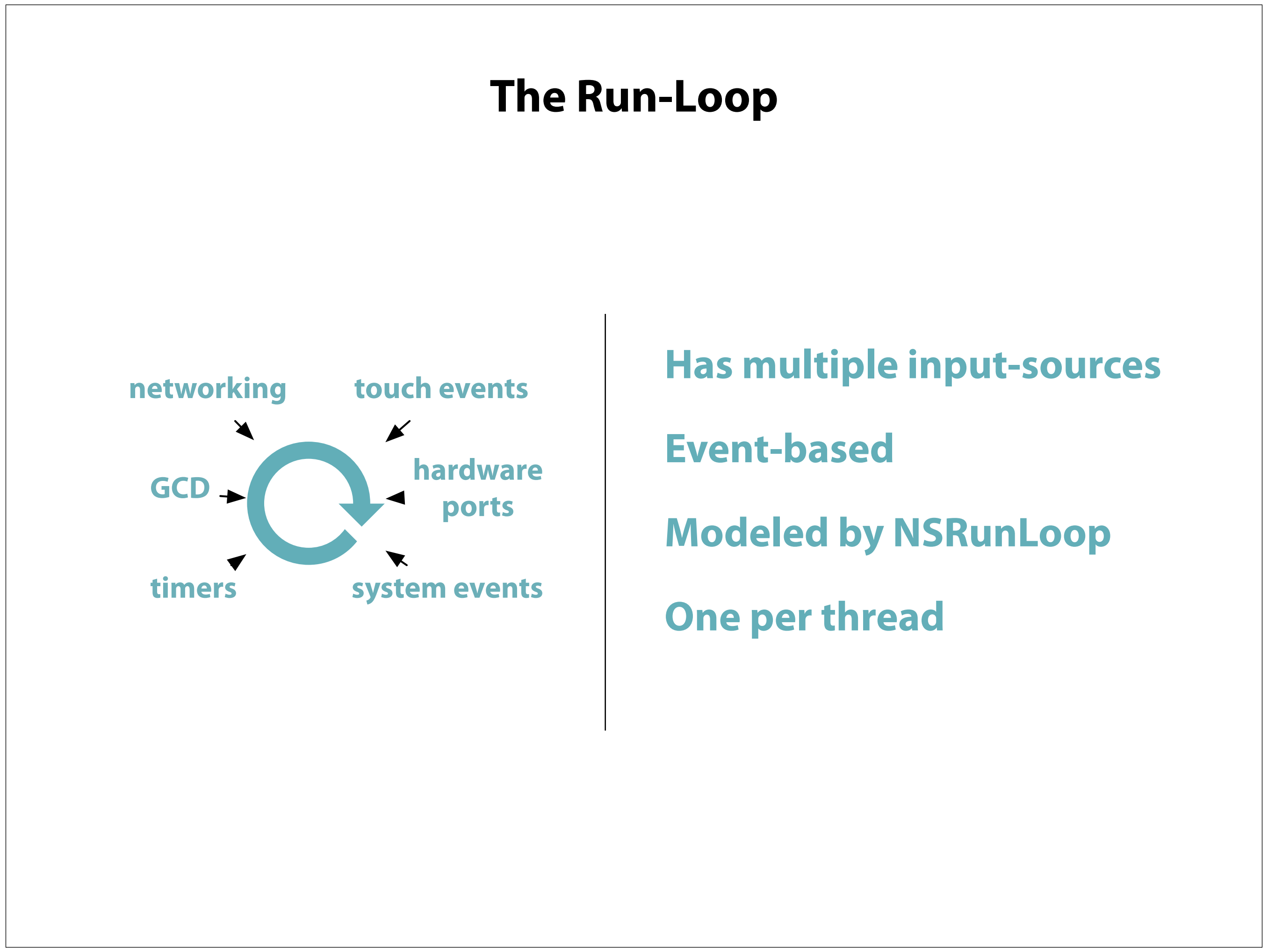
Scheduled connections cannot be **rescheduled**

You may be wondering how NSURLConnection work with threads.

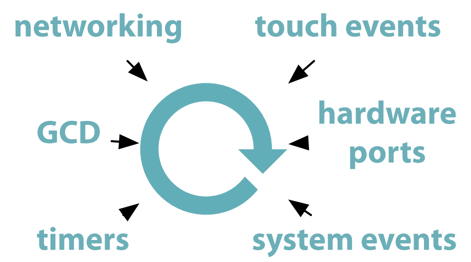
The first thing you should know is that NSURLConnections are scheduled in the current run-loop.

Callbacks to the delegate happen on the same thread that started the operation.

Once a connection begins, you cannot reschedule it.



The Run-Loop



Has multiple input-sources

Event-based

Modeled by NSRunLoop

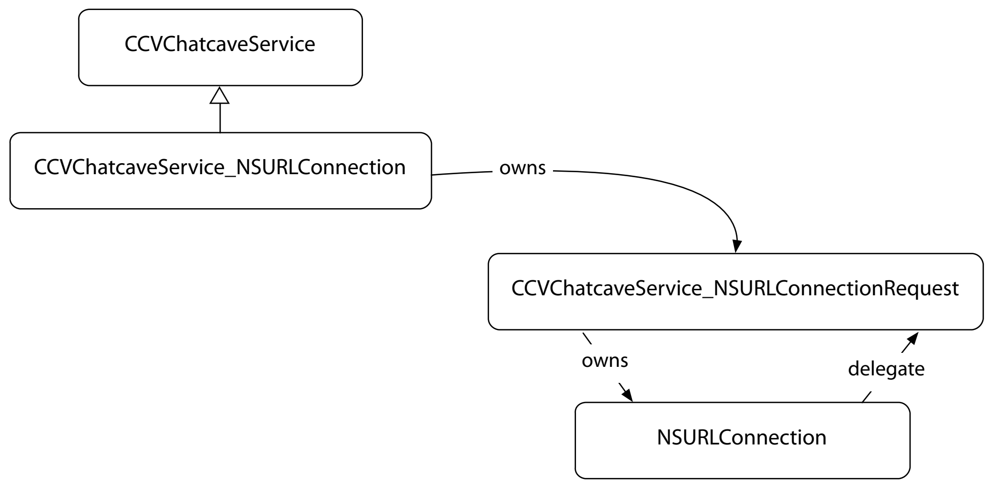
One per thread

What is the run loop? At the core of every Mac & iOS application is a central loop that runs throughout the lifetime of the application. This loop listens to multiple input sources and dispatches events to the appropriate code in your application. It's modeled by an instance of the NSRunLoop class. A run-loop is associated with a single thread.

When a NSURLConnection is scheduled in the main run-loop, that doesn't mean it's blocking the main thread. NSURLConnection manages its own threads internally for requests. The only question is how it will deliver events to its delegate. By default, NSURLConnection will call back to its delegate on the same thread that the connection was scheduled.

All of this is to say, you should almost never have to create your own threads or worry about NSURLConnection's threading implementation.

Service Classes


CCVChatcaveService
CCVChatcaveService_NSURLConnection
owns
CCVChatcaveService_NSURLConnectionRequest
owns
delegate
NSURLConnection

Before we dive into the code, let’s get a handle on the classes we’ll be working with.

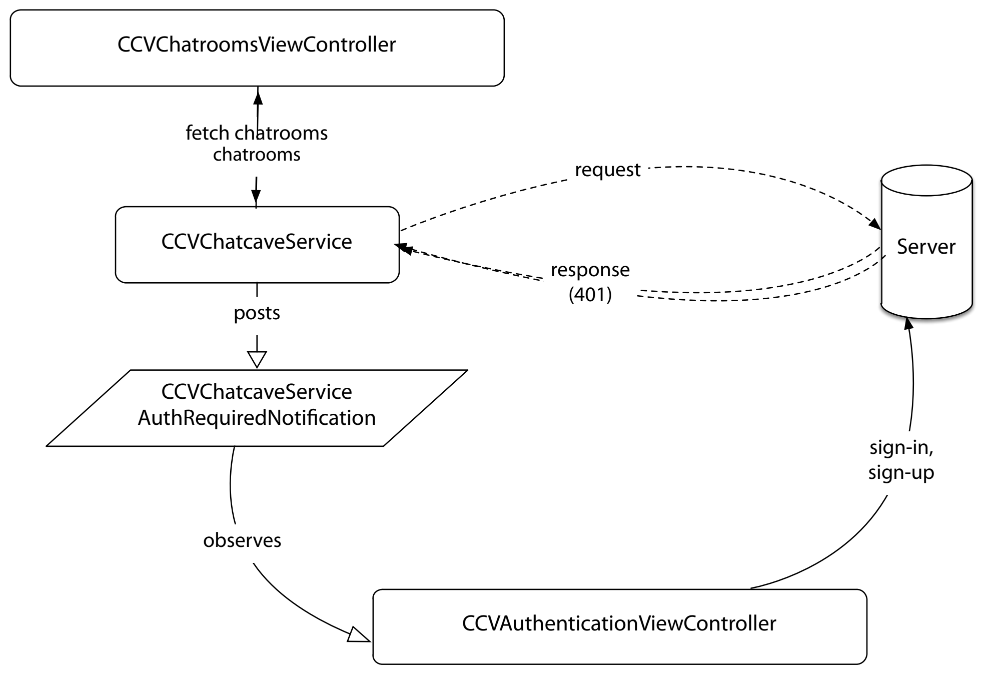
We’ll start by creating the CCVChatcaveService class which will encapsulate all access to the ChatCave API. We can put quite a bit of functionality in this class so we’ll treat it as an abstract superclass that we’ll re-implement and refine throughout this course.

Next, we’ll create a CCVChatcaveService sub-class that is built on NSURLConnection. We’ll give it the name CCVChatcaveService_NSURLConnection.

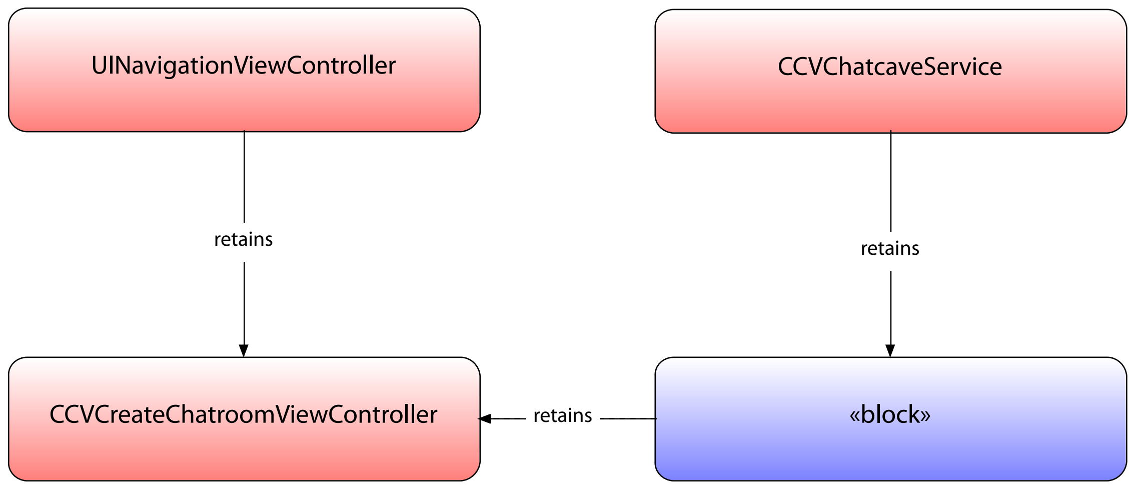
This class will create instances of CCVChatcaveService_NSURLConnectionRequest which will create a NSURLConnection instance and be its delegate.

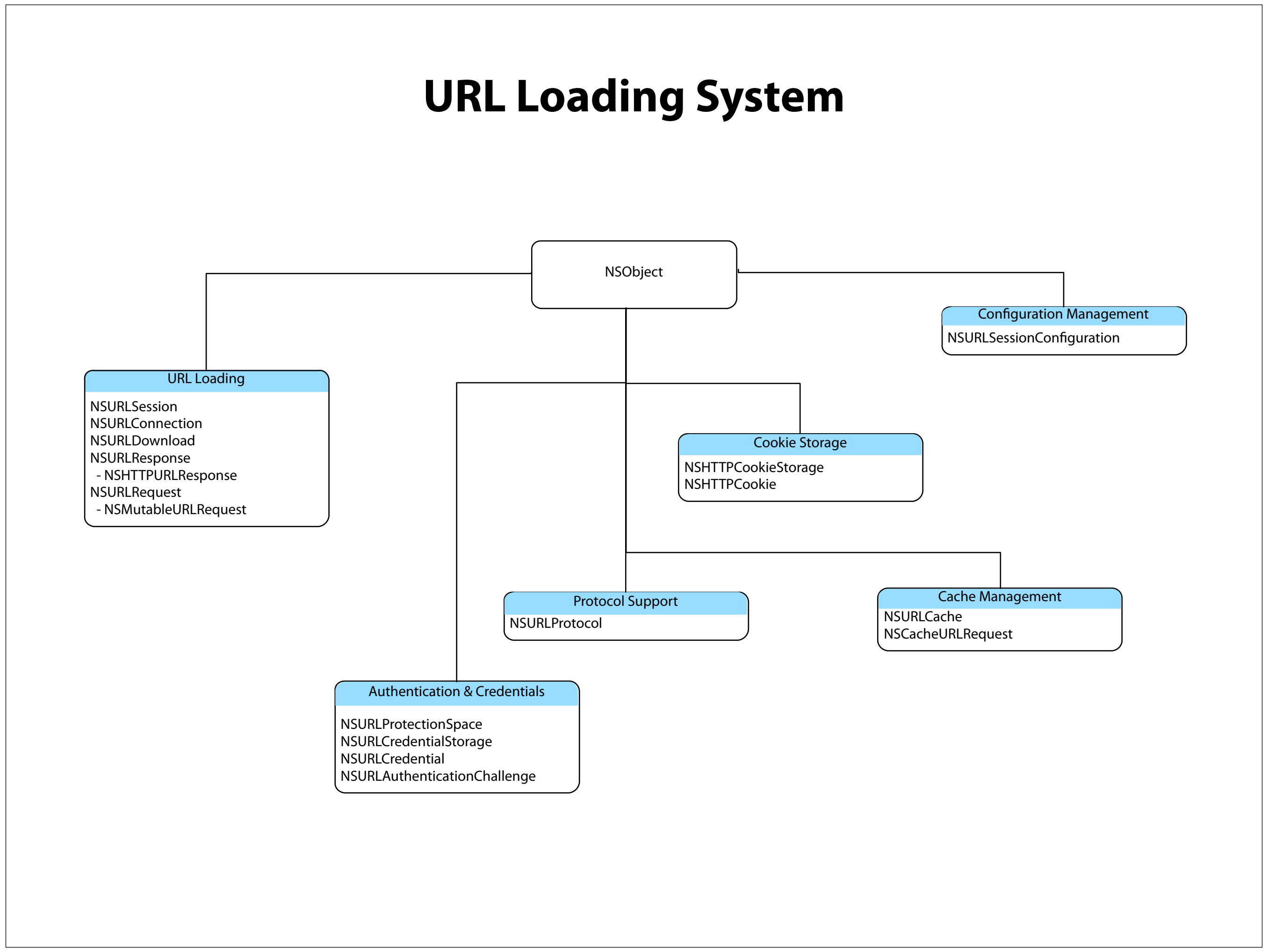
Let’s get started!

Authentication



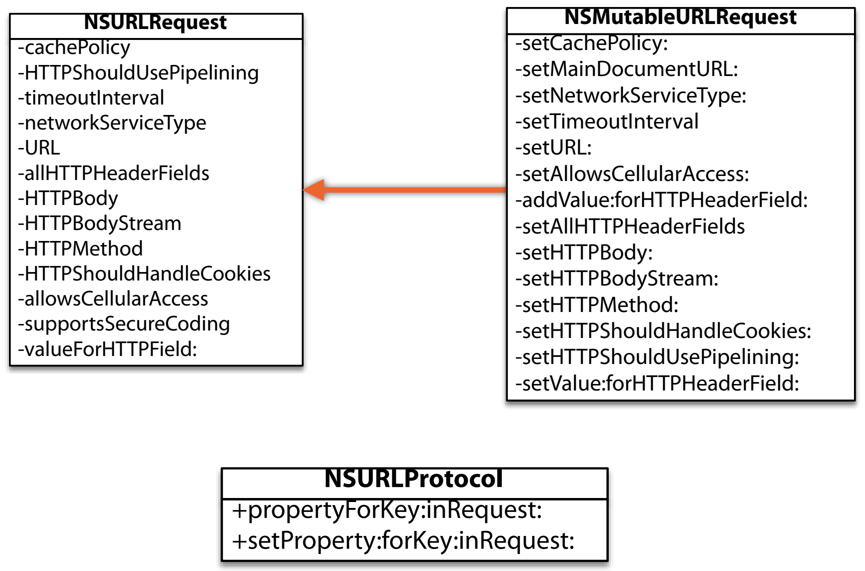
Retain Cycles





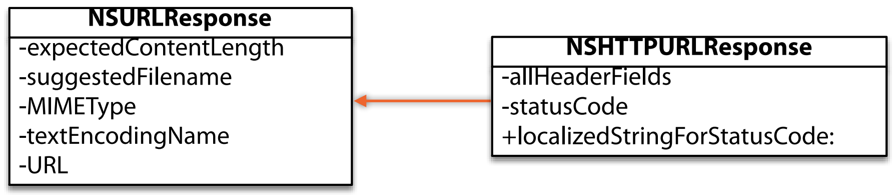
In this module, you were introduced to the URL Loading System. You learned that NSURLConnection is the traditional way of loading networking resources via URL.

Requests



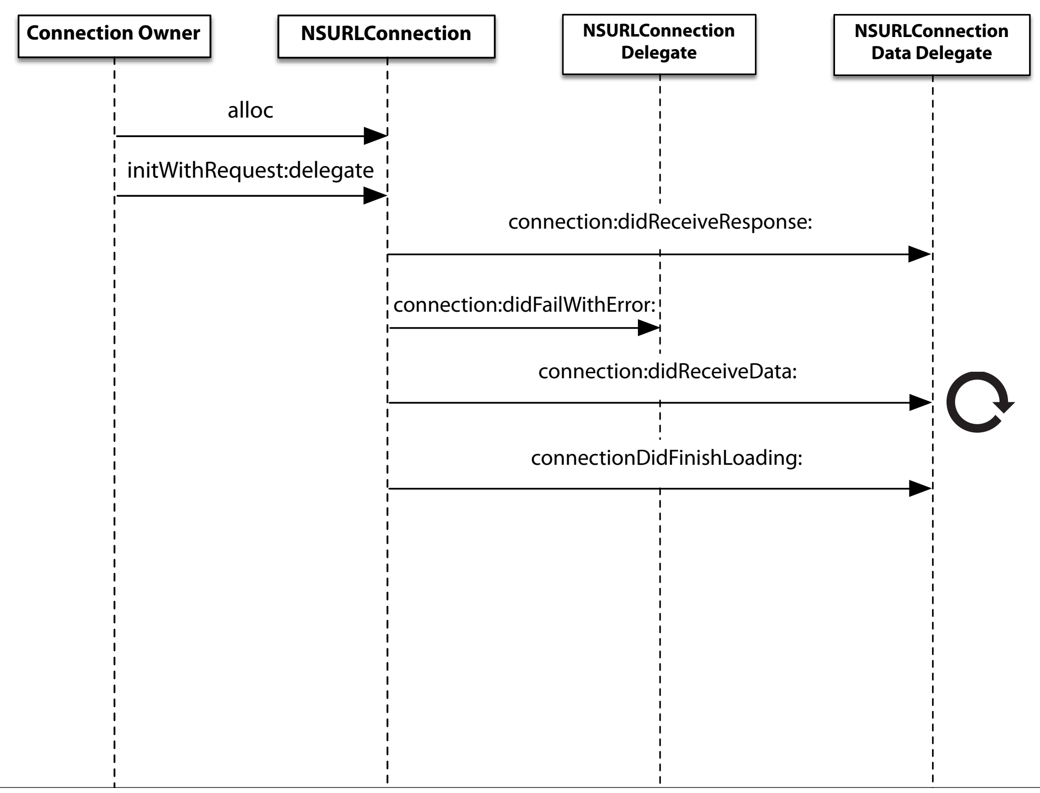
We learned how the NSURLRequest class encapsulates requests for network resources and how the NSMutableURLRequest sub-class allows us to refine those requests.

Responses



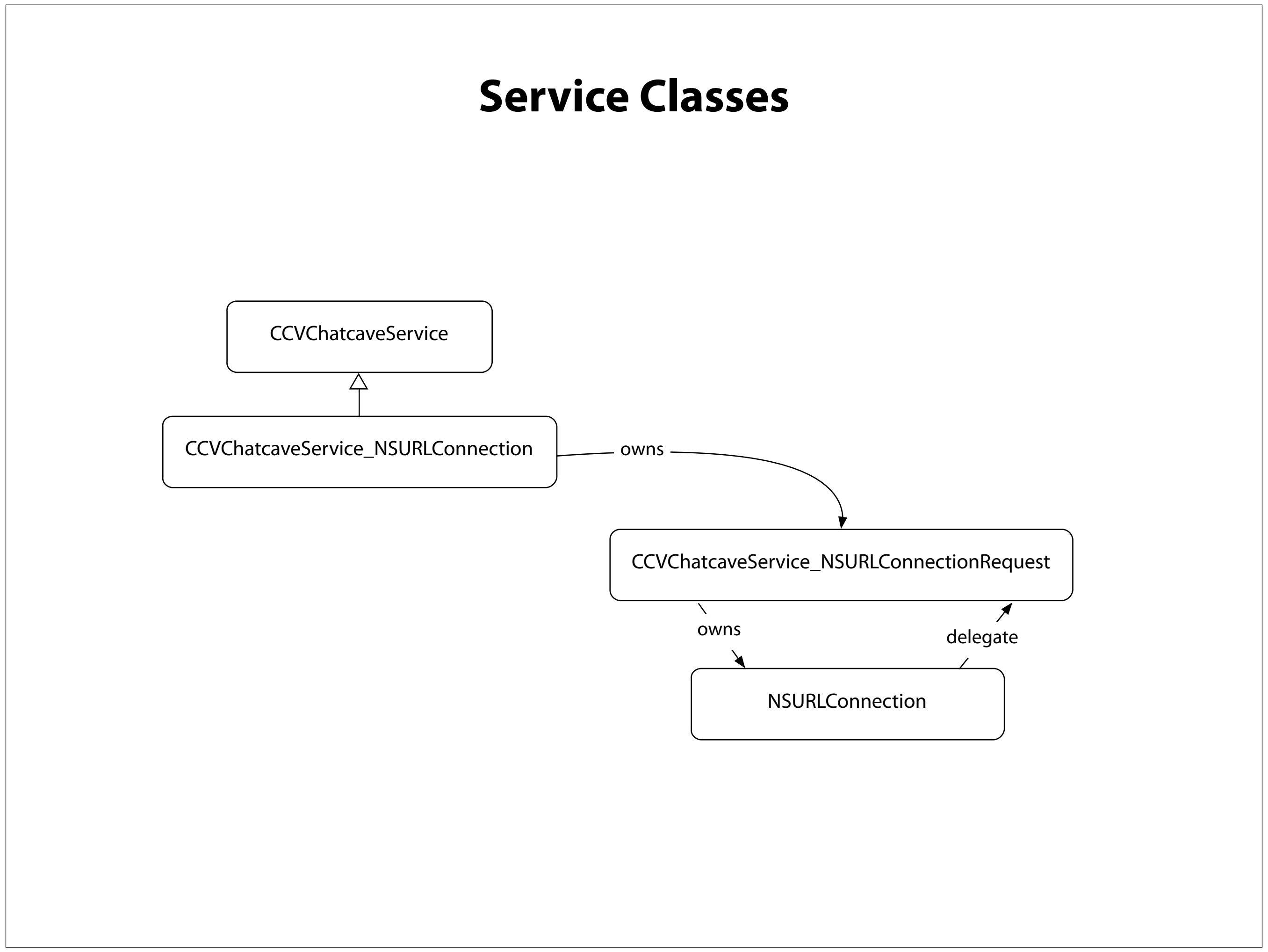
We learned how the NSURLResponse class encapsulates response envelopes and how the NSHTTPURLResponse subclass exposes additional HTTP-specific information.

Data Loading

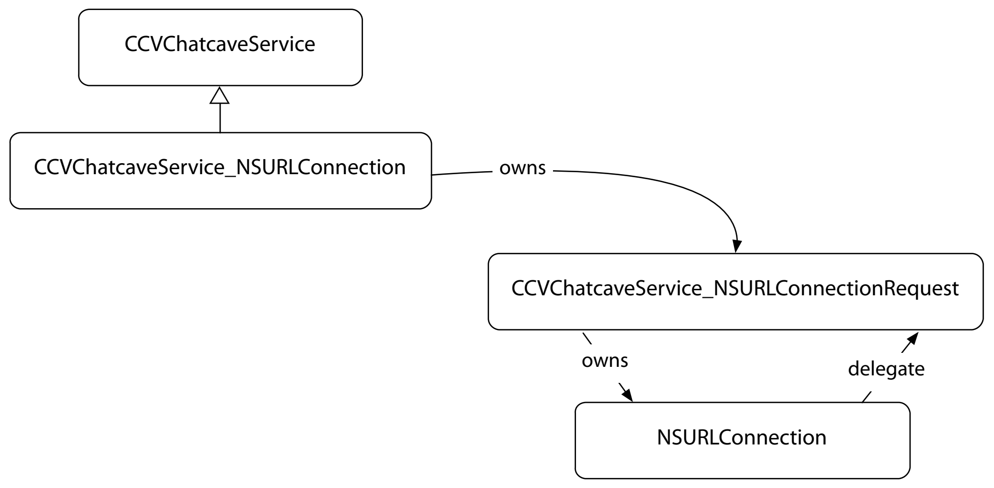


Connection Owner
NSURLConnection
NSURLConnection Delegate
NSURLConnection Data Delegate
alloc
initWithRequest:delegate
connection:didReceiveResponse:
connection:didFailWithError:
connection:didReceiveData:
connectionDidFinishLoading:

We learned how NSURLConnection interacts with its delegate to handle resource-loading. We also learned about the NSURLConnectionDelegate and NSURLConnectionDataDelegate protocols and how we can retrieve the payload from a response.



Service Classes


CCVChatcaveService
CCVChatcaveService_NSURLConnection
owns
CCVChatcaveService_NSURLConnectionRequest
owns
delegate
NSURLConnection

We also performed our first implementation of the networking layer using NSURLConnection. As we built out our ChatcaveService implementation, we saw how much functionality could exist in the abstract parent class before we ever had to deal with a NSURLConnection.