HIT3329 / HIT8329
Creating Data Driven Mobile Applications

Lecture 9
Designing an App Around External Data Sources

Presented by Paul Chapman

Adjunct Lecturer F-ICT
Director, Long Weekend LLC
Swinburne University of Technology

SWIN BUR NE
SWINBURNE UNIVERSITY OF TECHNOLOGY

Last Lecture Reviewed

Design Considerations:

1. User Interface (UI)
2. User Experience (UX)

Questions?

What's On For Today?

Architecting a Twitter App:

1. Modeling & Data Flows
2. View Hierarchy
3. Feed Implementation
4. Publishing Implementation
5. Application Logic

1.0 Functional Overview

- Connect to a user's Twitter account
- Read tweet feeds:
  - Home timeline
  - Mentions ("@long_weekend ...")
  - My Tweets, Retweeted
- Display user images next to tweets
- Publish the user's tweets
- Sign user out from Twitter
- Read tweets even when no Internet

1.1 High Level MVC Analysis - Model I

What entities are in the app?

- Tweet
- User

... a good way to complete this step: talk your way through each of the features, paying attention to the nouns: "app will download a list of *tweets* from *Twitter* for a given *user account*"

1.2 High Level MVC Analysis - Model II

What attributes does each entity have?

- Tweet
  - actual text (NSString)
  - date/time (NSDate)
  - user who said it (User)
  - tweet type? (retweet? mention? home feed?)
- User
  - username (NSString)
  - profile image (??!)

1.3 High Level MVC Analysis - Model III

tweet type? (retweet? mention? home feed?)
profile image (??!)

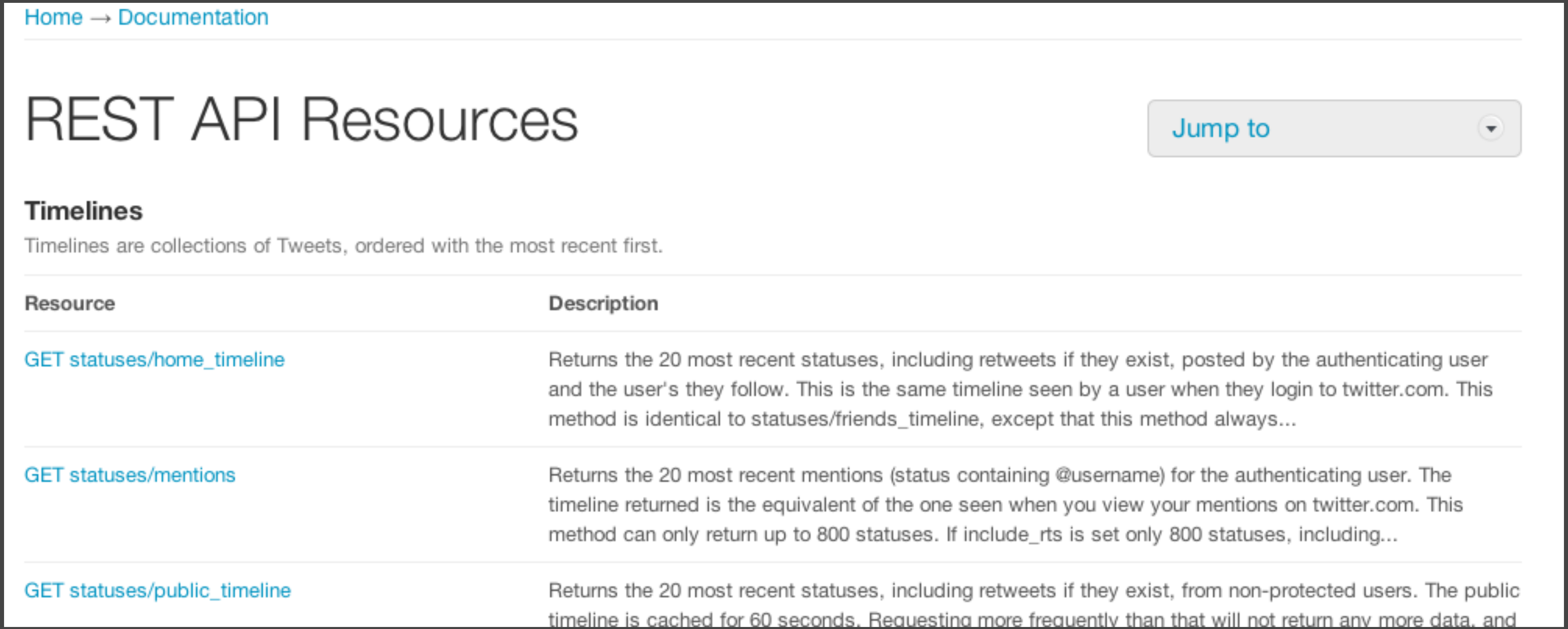
Model design forces questions: often because we don't have enough information yet.

"So, how do I access Twitter in the first place?   Are there separate APIs for mentions, home feeds, etc??!"

Depending on answers, we can finish the model.

1.4 Twitter API - I

Browsing Twitter's developer site, we find what we want at https://dev.twitter.com/docs/api



Home → Documentation

REST API Resources

Jump to

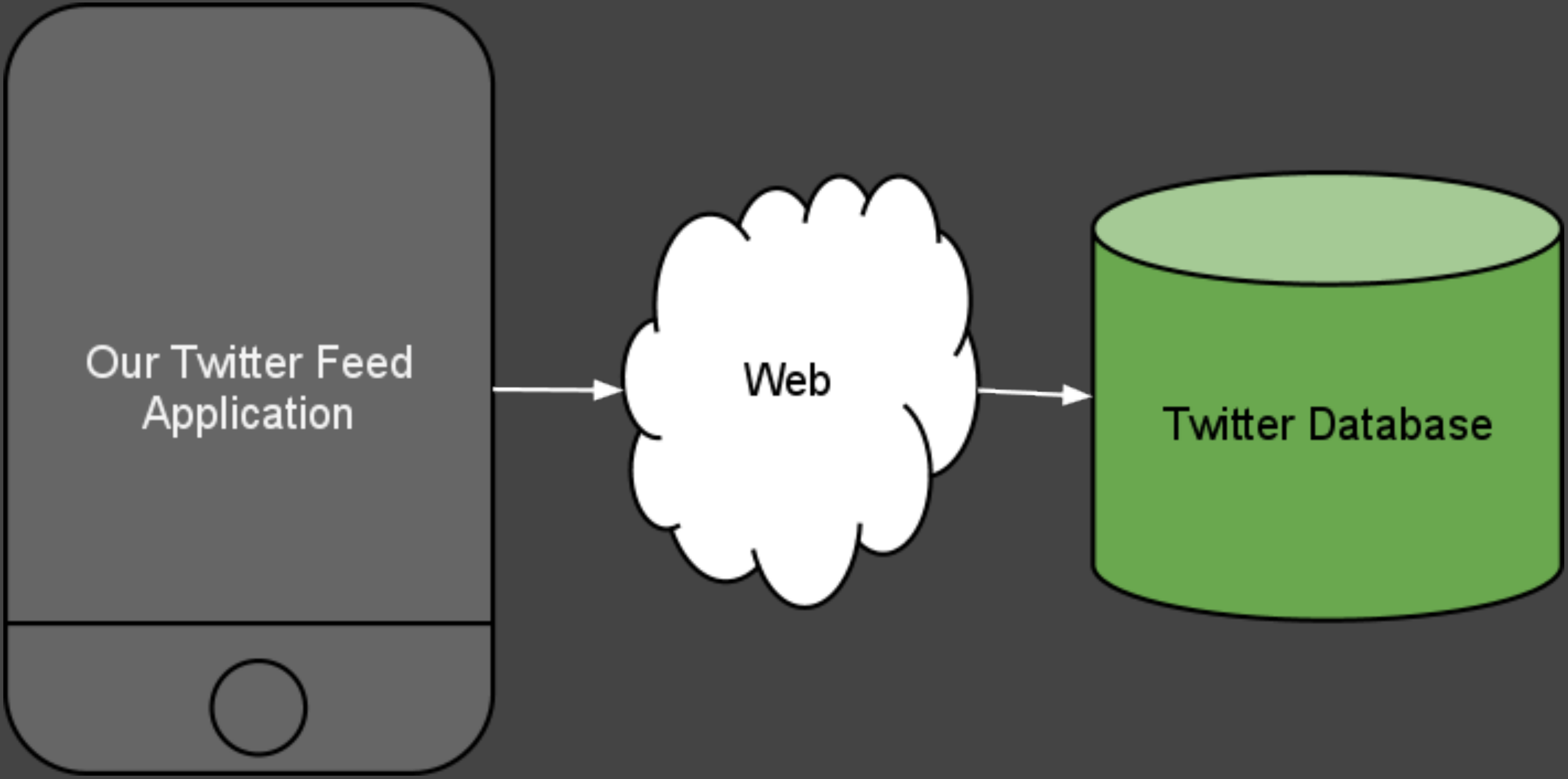
Timelines

Timelines are collections of Tweets, ordered with the most recent first.

Resource	Description
GET statuses/home_timeline	Returns the 20 most recent statuses, including retweets if they exist, posted by the authenticating user and the user's they follow. This is the same timeline seen by a user when they login to twitter.com. This method is identical to statuses/friends_timeline, except that this method always...
GET statuses/mentions	Returns the 20 most recent mentions (status containing @username) for the authenticating user. The timeline returned is the equivalent of the one seen when you view your mentions on twitter.com. This method can only return up to 800 statuses. If include_rts is set only 800 statuses, including...
GET statuses/public_timeline	Returns the 20 most recent statuses, including retweets if they exist, from non-protected users. The public timeline is cached for 60 seconds. Requesting more frequently than that will not return any more data, and

1.5 Twitter API - II

Twitter's docs say to access via HTTP web requests.



1.6 Twitter API - III

Twitter uses a "RESTful HTTP API"..... meaning:

- GET data using a URL, just like reading a Web page in a browser

- POST data into Twitter with a URL, just like submitting an online form with a browser

GET and POST are the HTTP terms for these 2 kinds of actions, there are others: DELETE and PUT, but aren't used often - <sadface>

1.7 Twitter API - IV

Twitter API documentation matches exactly what we want for the 3 read functions:

- Home Feed: GET statuses/home_timeline
- Mentions: GET statuses/mentions
- Retweets: GET statuses/retweets_of_me

For publishing, we can use:

- POST statuses/update

1.8 Twitter API - V

Finally, we need to get the images of each user:
GET users/profile_image/:screen_name

The docs also define an important new requirement:
"This method must not be used as the image source URL presented to users of your application."

... in other words, we have to save the images locally!

1.9 Twitter API - VI

One more thing to worry about ... ***signing in and signing out!***

From the API documentation, again:

"Twitter uses the open authentication standard OAuth for authentication."

1.10 OAuth Says What?

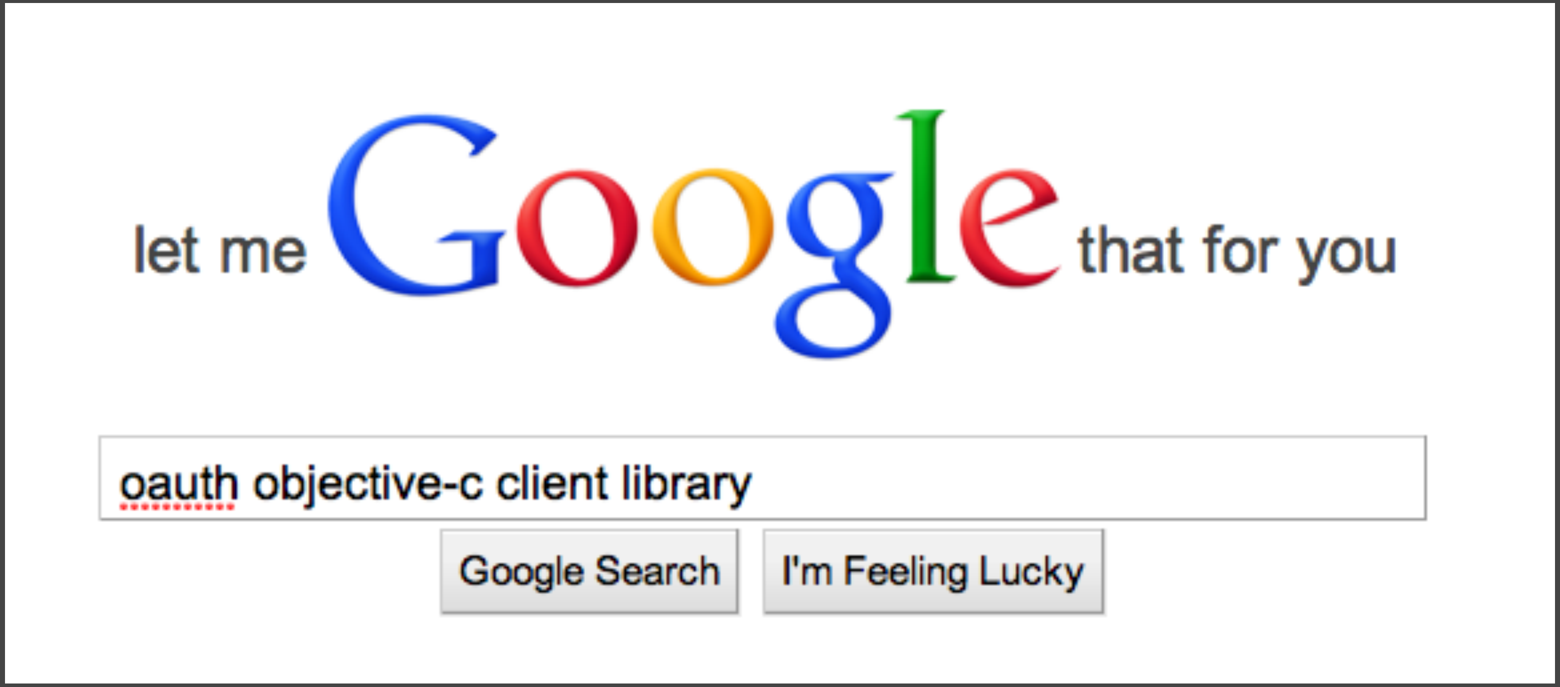
What is OAuth? According to http://oauth.net:

An open protocol to allow secure API authorization in a simple and standard method from desktop and web applications.

To us, that reads ***really big headache***.

1.11 Libraries to the Rescue! (Part One)

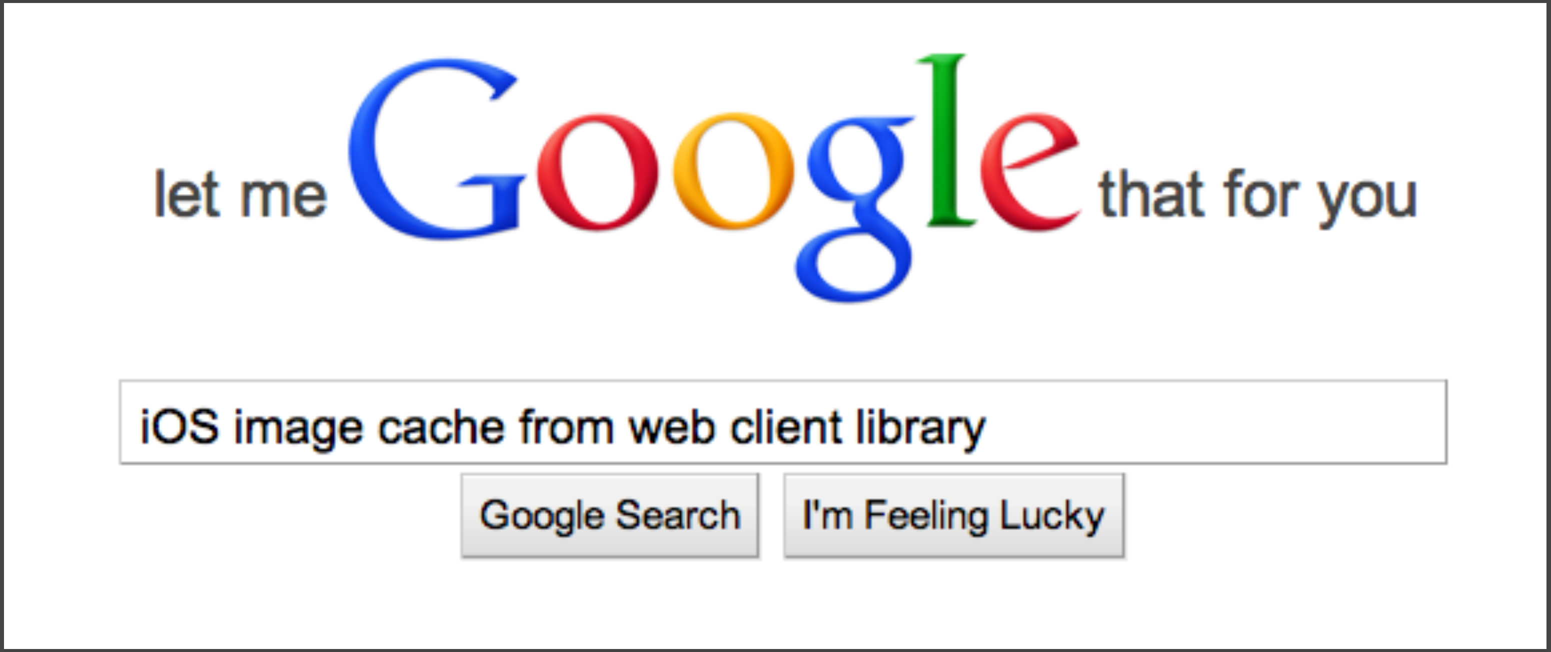
Hasn't someone already solved this problem?



oauth objective-c client library

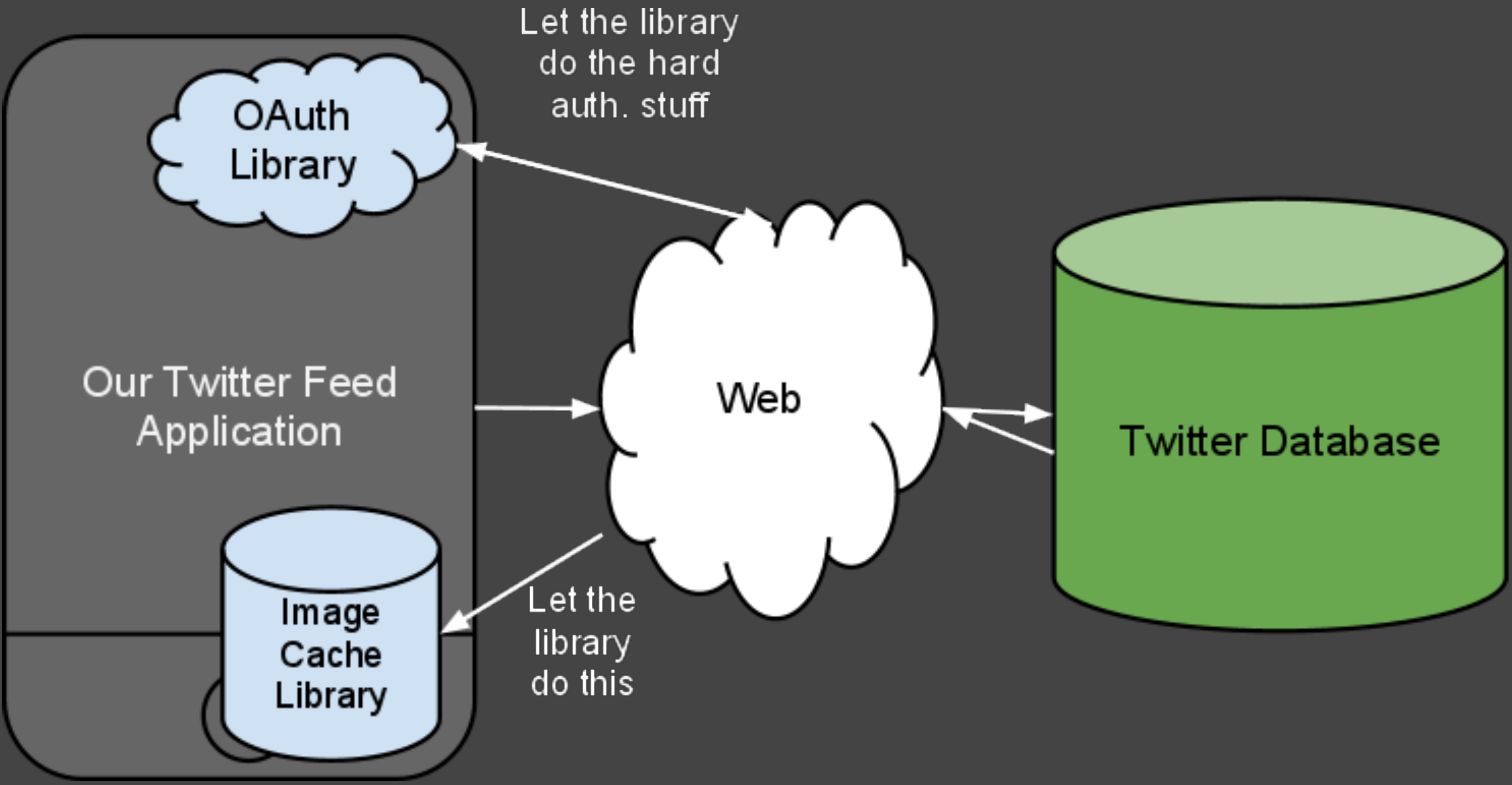
1.12 Libraries to the Rescue! (Part Deux)

And this one too?



iOS image cache from web client library

1.12 Back to High-Level Structure



1.13 Meanwhile, the Model Design

Now, let's answer these questions:

- tweet type (retweet? mention? home feed?)
- profile image (??!)

Answer #1: We need to remember which tweet is associated with which feed -- we can use a simple NSInteger enumeration

Answer #2: We pass a URL to the image cache library (as a string), and it does the rest of the work -- so profile image can just be stored as an NSString.

1.14 Model, Complete - Ready for Core Data

- Tweet
  - actual text (NSString)
  - date/time (NSDate)
  - user who said it (User)
  - tweet type (NSInteger)

- User
  - username (NSString)
  - profile image URL (NSString)

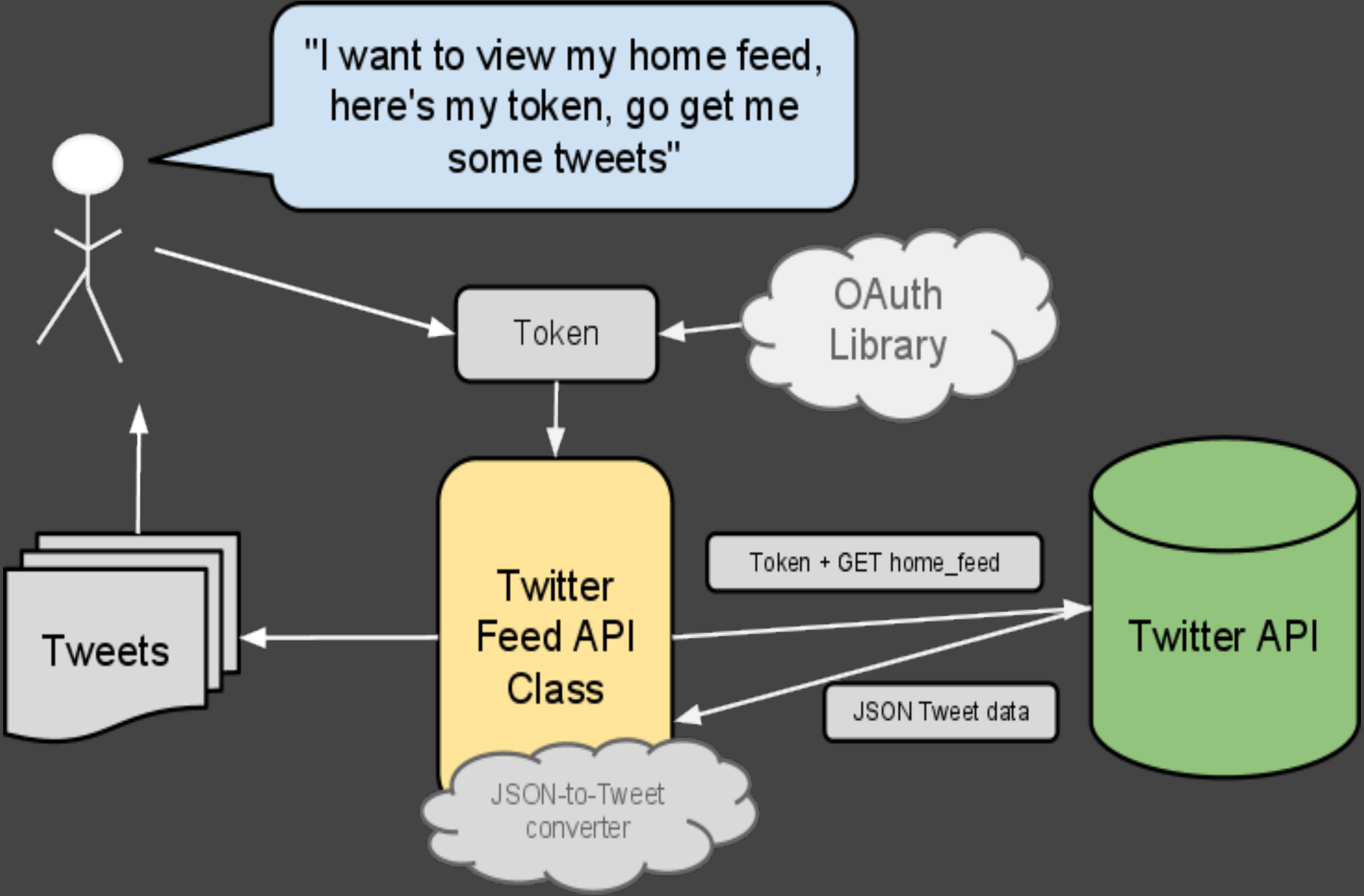
1.15 Getting Tweets from Twitter

We now know what an individual Tweet object looks like ... ***but***

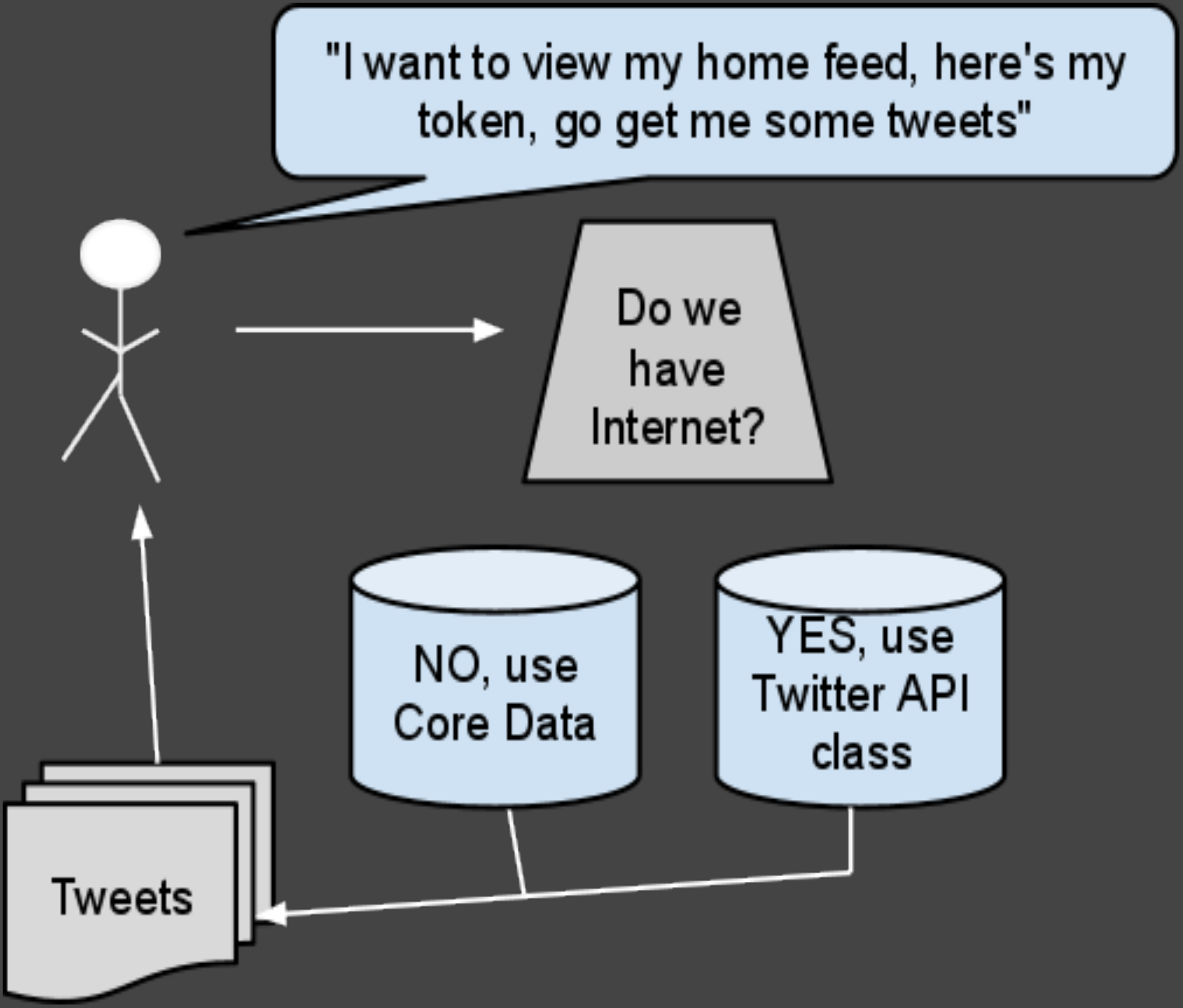
Who's responsibility is it to retrieve a whole stream of tweets?

... we also need an *API object* that acts as an interface between ***OUR*** model (Tweet, User, etc) and ***THEIR*** model (Twitter's data is JSON)

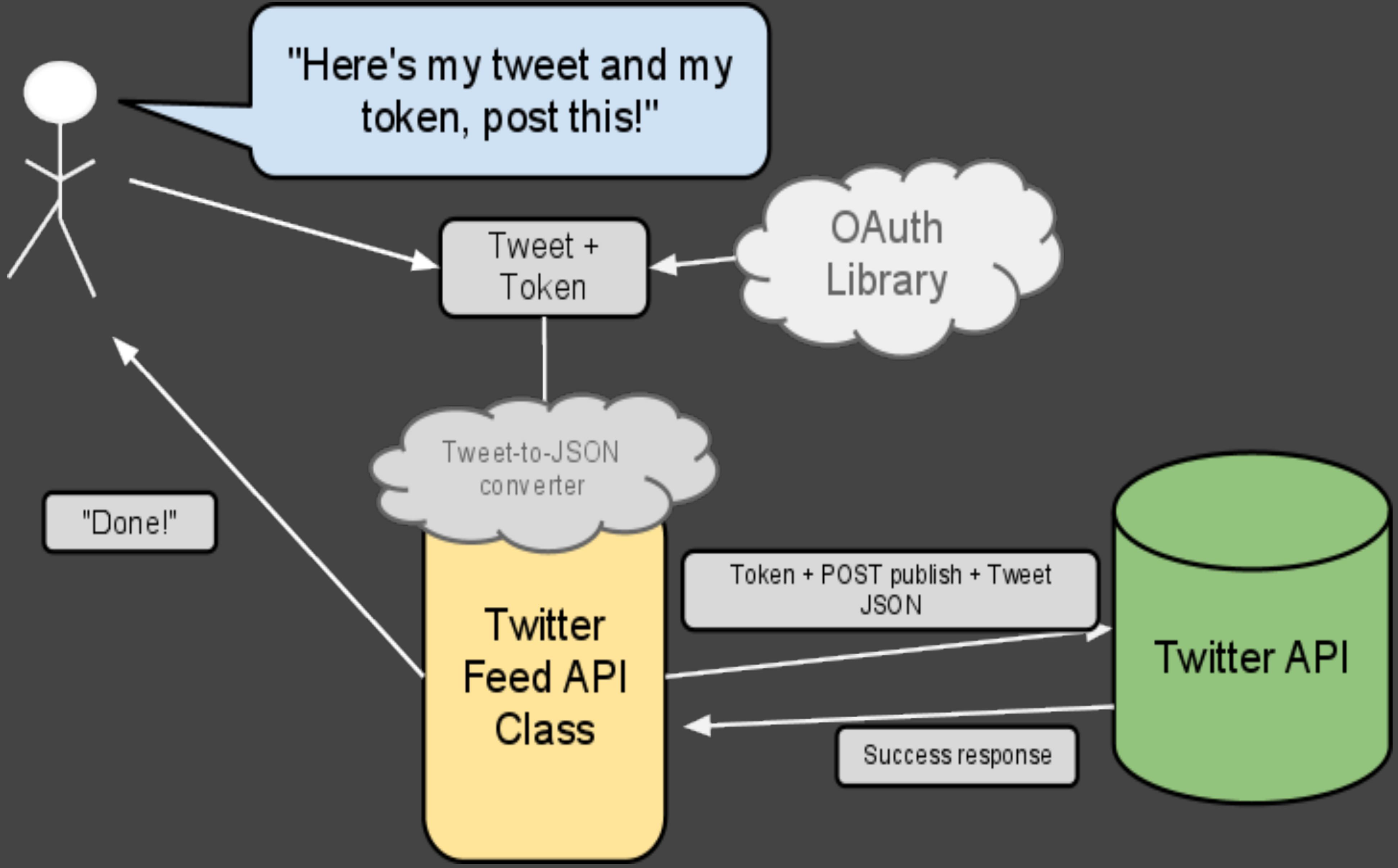
1.16 Getting Tweets from Twitter - Flow



1.17 Getting Tweets from Twitter - Offline


"I want to view my home feed, here's my token, go get me some tweets"
Do we have Internet?
NO, use Core Data
YES, use Twitter API class
Tweets

1.18 Sending Tweets to Twitter - Flow



What's On For Today?

Architecting a Twitter App:

1. Modeling & Data Flows
2. View Hierarchy
3. Feed Implementation
4. Publishing Implementation
5. Application Logic

2.0 View Hierarchy

The UX interactions for this app:

- View a feed (list of Tweet objects)
- Compose a new tweet
- Login or log out of Twitter

2.1 View Hierarchy - Feeds

For each feed, use a UITableViewController

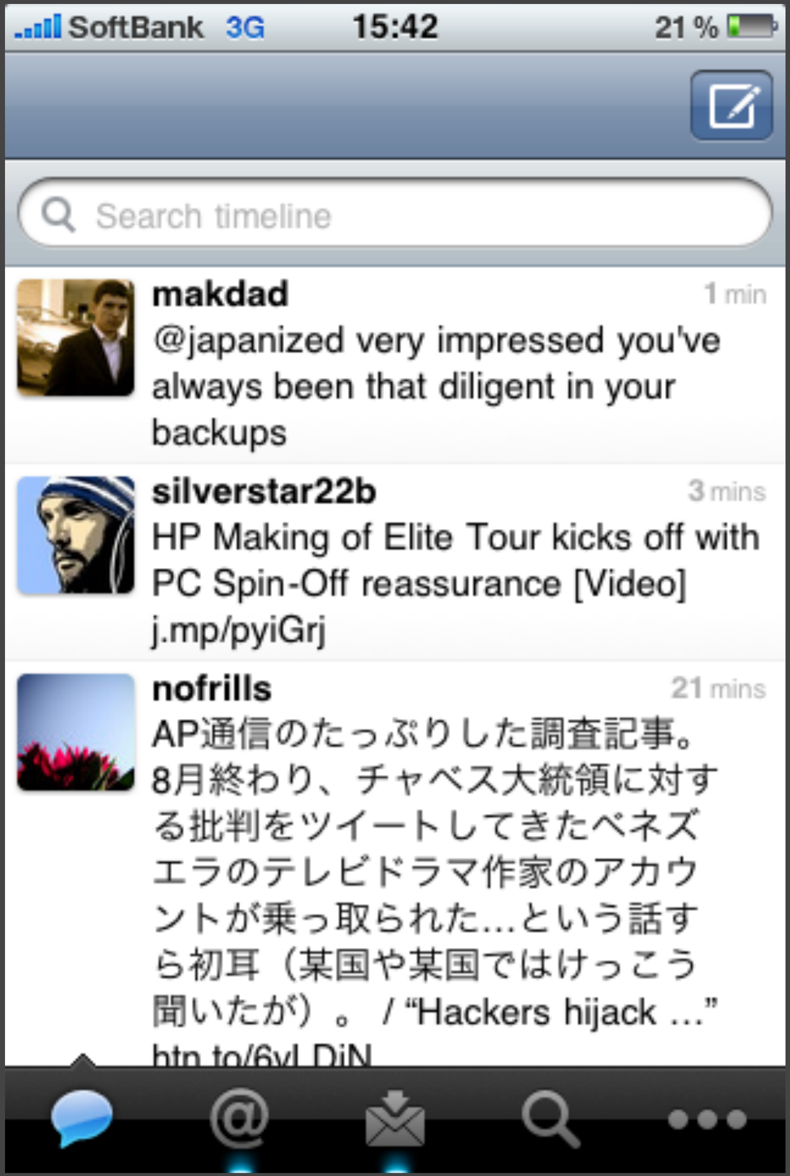
For navigation between feeds ... there are (*at least*) 2 ways to implement

- UITabBarController with 3 tabs
- Drill-down "main menu" table view
- What would you do? Think like a user

2.2 View Hierarchy - Feeds

We think UITabBarController is best for this case...

... so does the actual Twitter app >>

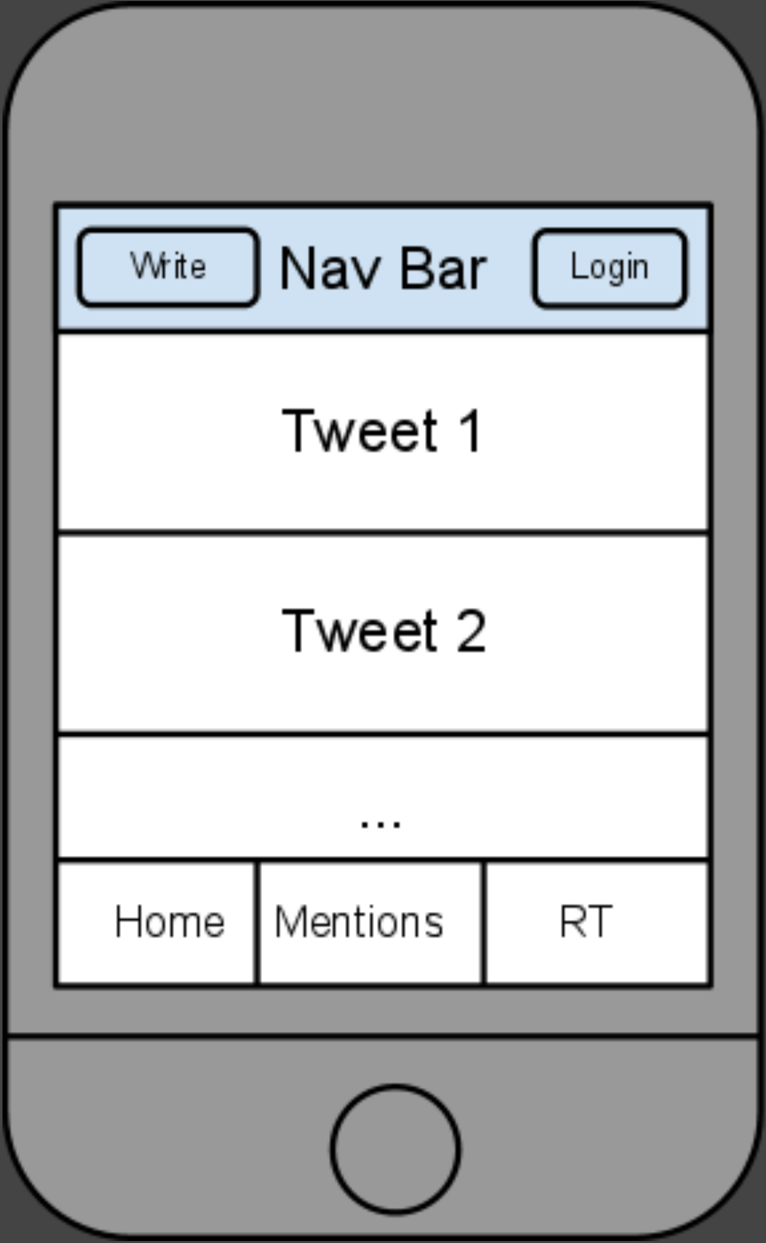


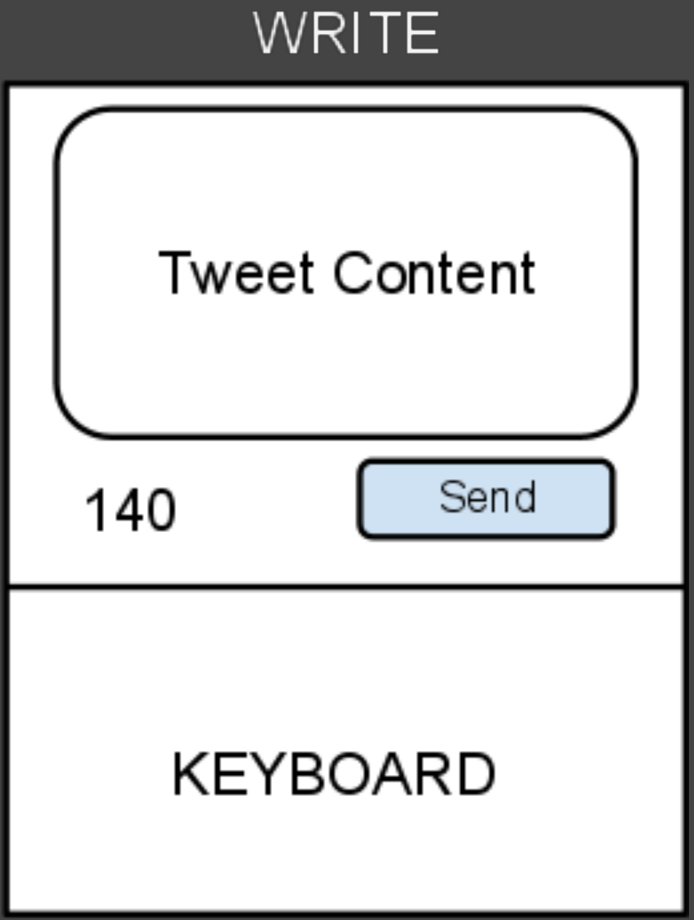
makdad 1 min
@japanized very impressed you've always been that diligent in your backups

silverstar22b 3 mins
HP Making of Elite Tour kicks off with PC Spin-Off reassurance [Video] j.mp/pyiGrj

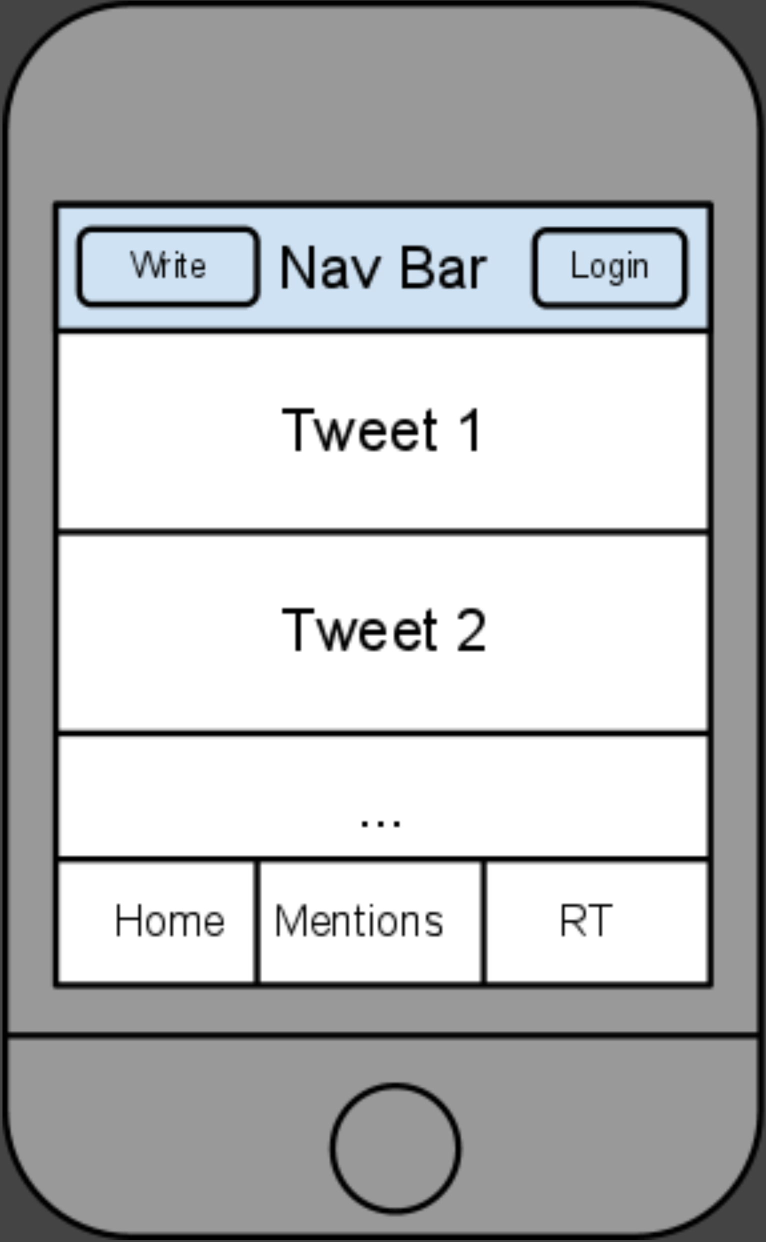
nofrills 21 mins
AP通信のたっぷりした調査記事。8月終わり、チャベス大統領に対する批判をツイートしてきたベネズエラのテレビドラマ作家のアカウントが乗っ取られた…という話すら初耳（某国や某国ではけっこう聞いたが）。 / "Hackers hijack …" htp.to/6yLDiN

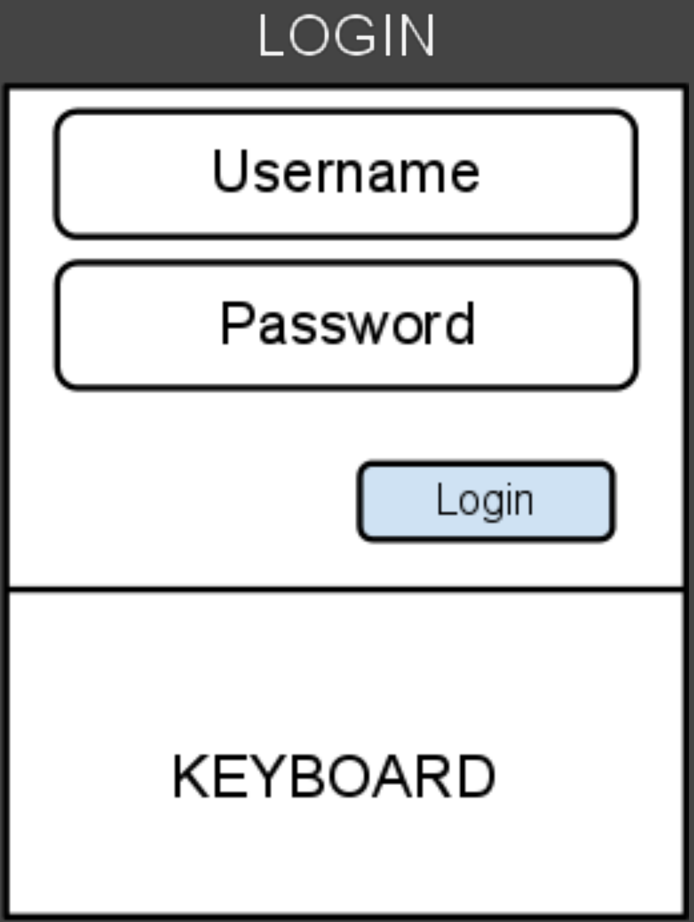
2.3 Our View Layout, I





2.3 Our View Layout, II





2.4 What Does The UI Hierarchy Look Like?

- App Delegate Object
  - UITabBarController
    - UINavigationController (Home)
      - UIButton (New Tweet / Compose)
      - UIButton (Login/Logout)
      - FeedViewController
        - Feed API object
    - UINavigationController (Mentions)
      - UIButton x 2
      - FeedViewController
        - Feed API object
    - UINavigationController (RT)
      - UIButton x 2
      - FeedViewController
        - Feed API object

2.5 Save Yourself Time

...All of that hierarchy can be set up in MainWindow.xib using Interface Builder!

Next, supply the IBAction methods, @property declarations, and set some things on -viewDidLoad to make it work!

2.6 Get Tweets When the View Loads

FeedViewController's -viewDidLoad is a good place to make our API call:

```
- (void) viewDidLoad {
  [super viewDidLoad];
  self.api = [[[FeedAPI alloc] init] autorelease];
  [api loadTweetsOfType:kTweetMentions delegate:self];
}

// This would be a delegate callback - reload the table
- (void) tweetApiDidLoad:(FeedAPI*)api {
  self.tweets = [self.api loadedTweets];
  [self.tableView reloadData];
}
```

2.6 Feed Implementation

But what would this call look like on the inside?

```
[api loadTweetsOfType:kTweetMentions
                 delegate:self];
```

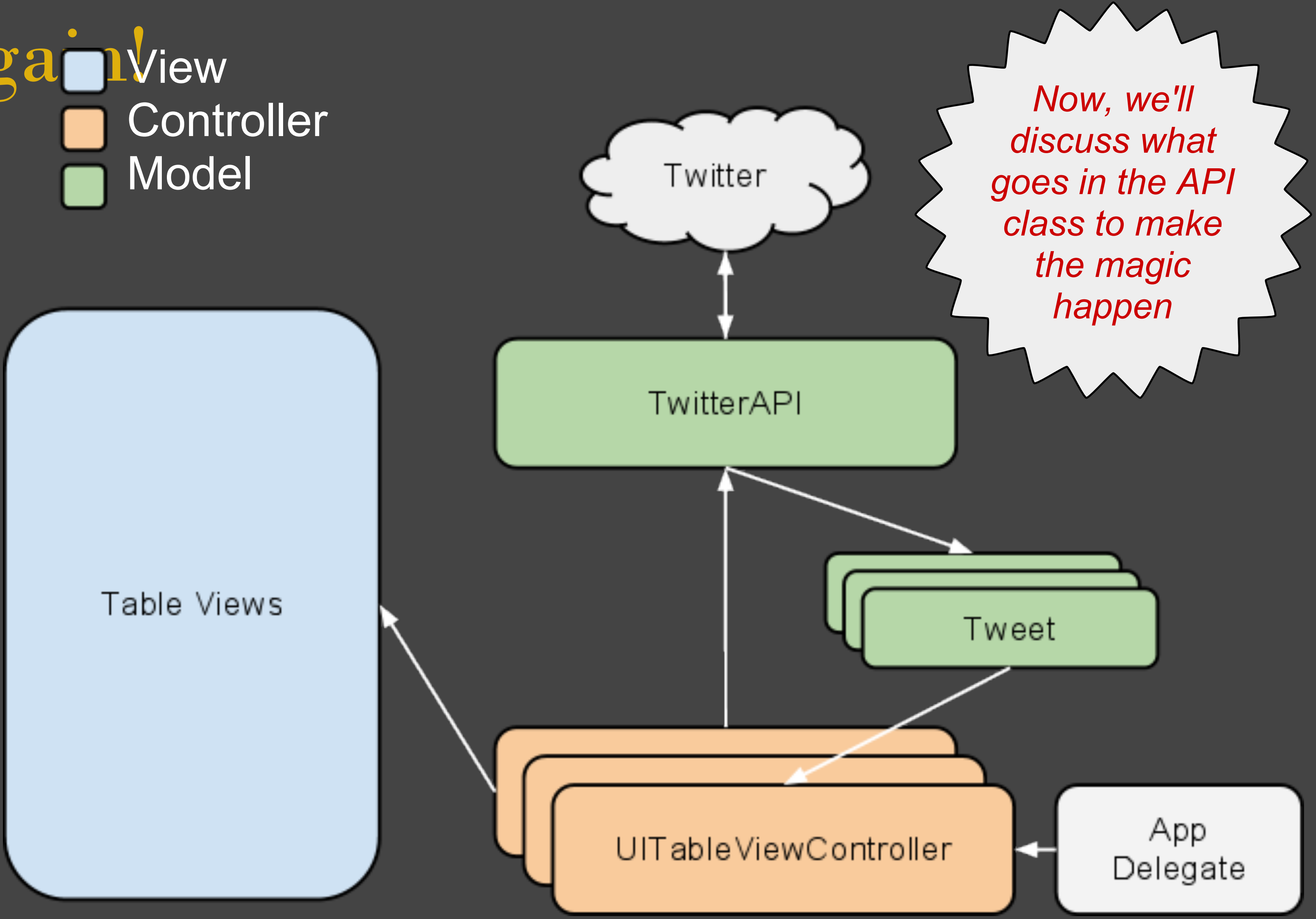
... think back to the end of lecture 6 -- we make a @protocol & delegate to have the view controller "told" when the network call finishes.

What's On For Today?

Architecting a Twitter App:

1. Modeling & Data Flows
2. View Hierarchy
3. Feed Implementation
4. Publishing Implementation
5. Application Logic

3.0 Here's Lecture 6, Slide 4.8 - MVC again!



Now, we'll discuss what goes in the API class to make the magic happen

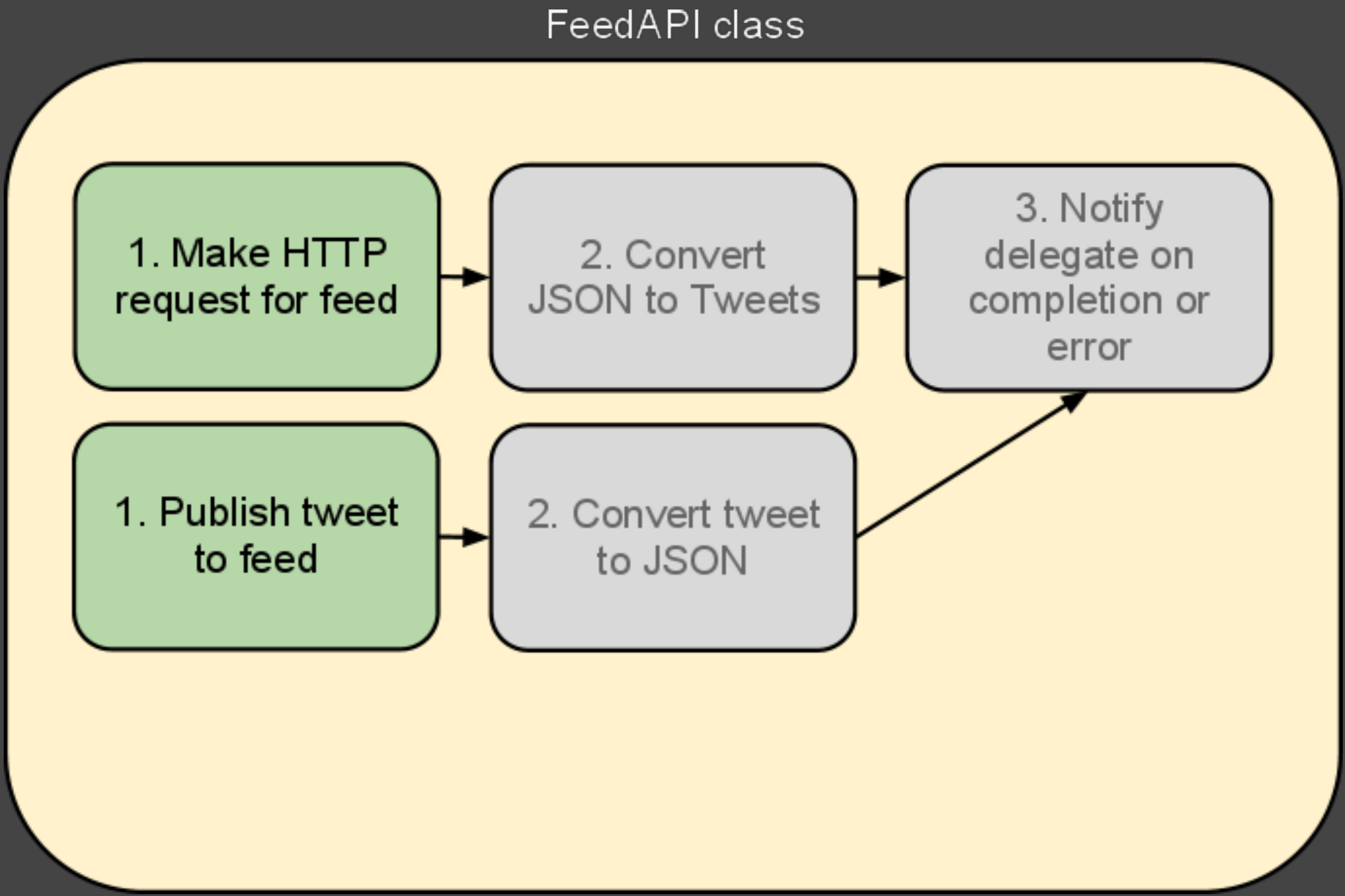
3.1 What Does FeedAPI look like?

The API class will serve 2 purposes:

- Initiate all network HTTP calls
- Handle HTTP responses, converting between JSON data and Tweet objects

...(you could argue a class with 2 jobs should be 2 classes.)

3.2 What Does FeedAPI look like?


FeedAPI class
1. Make HTTP request for feed
2. Convert JSON to Tweets
3. Notify delegate on completion or error
1. Publish tweet to feed
2. Convert tweet to JSON

3.2 Network Call Code - Making the URL

```
- (void) loadTweetsOfType:(kTweetTypes)type
                 delegate:(id<FeedAPIDelegate>aDelegate {
  self.delegate = aDelegate;
  NSString *feed = nil;

  if (type == kTweetTypeMentions) {
      feed = @"mentions.json";
  } else if (type == kTweetTypeRetweets) {
      feed = @"retweets_of_me.json";
  } else {
      feed = @"home_timeline.json";
  }

  NSString *url = [NSString stringWithFormat:
      @"https://api.twitter.com/1/statuses/%@",feed];
```

3.3 Different Strokes for Different Folks

Instead of the if...then...else statement, we could make 3 subclasses of the same base class that handles the request -- each with the feed URL set as a string in -init

However, for readability & simplicity, we choose to have a single API class handle all 3 feeds + publishing.

3.4 Network Call Code - Making the request

... (cont'd from 3.2 slide) ...

```
  NSString *url = [NSString stringWithFormat:
     @"https://api.twitter.com/1/statuses/%@",feed];

  NSURLRequest *request = [NSURLRequest requestWithURL:
                          [NSURL URLWithString:url]];

  self.connection = [NSURLConnection
                       connectionWithRequest:request
                       delegate:self];
}
```

FeedAPI is the delegate for NSURLConnection -- and the view controller is the delegate of the FeedAPI. Such "delegate chaining" is common in interactions between code of different "depths".

3.5 Network Call Code - Delegate Callback

```
- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
  // (assume we implemented connectionDidReceiveData:)

  // Next we have to make this method:
  self.tweets = [self tweetsFromJSONData:self.receivedData];

  // Now tell *our* delegate that we're done getting tweets
  if ([self.delegate respondsToSelector:
                          @selector(didFinishLoadingTweets:)]) {
      // We need to make this @protocol too
      [self.delegate didFinishLoadingTweets:self];
  }
}
```

Notice the "delegate value chain" -- each delegate callback is "adding value" to the data before passing it along to its delegate

3.6 Converting JSON to Tweets & vice-versa

Twitter sends us data in JSON format... but our view controllers will use model objects (namely Tweet, User).

FeedAPI needs to convert between the two!

Vice-versa applies for tweet publishing: FeedAPI has to convert a Tweet to JSON

3.7 Converting JSON to Tweets

```
- (NSArray*) tweetsFromJSONData:(NSString*)jsonData {
  NSMutableArray *tweets = [NSMutableArray array];

  // Awesomely, jsonframework library makes this 1 line!
  NSArray *tweetArray = [jsonData JSONValue];

  for (NSDictionary *singleTweet in tweetArray)
  {
    Tweet *newTweet = [[Tweet alloc] init];
    newTweet.text = [singleTweet objectForKey:@"text"];
    // ... (set other attributes here, including User)
    [tweets addObject:newTweet];
    [newTweet release];
  }
  return (NSArray*)tweets;
}
```

What's On For Today?

Architecting a Twitter App:

1. Modeling & Data Flows
2. View Hierarchy
3. Feed Implementation
4. Publishing Implementation
5. Application Logic

4.0 Converting a Tweet to JSON

```
- (NSString*) JSONDataFromTweet:(Tweet*)tweet
{
  // We need to make a dictionary or array to convert to
  // JSON value
  NSMutableDictionary *dict =
                      [NSMutableDictionary dictionary];

  // Set keys for whatever the Twitter API requires...
  [dict setObject:tweet.text forKey:@"text"];
  [dict setObject:tweet.timestamp forKey:@"pubDate"];

  // jsonframework library ALSO makes this 1 line!
  NSString *jsonTweet = [dict JSONRepresentation];

  return jsonTweet;
}
```

4.1 Publishing Implementation

Publishing is the exact same as getting tweets, just in a different order:

1. Convert Tweet object to JSON
2. Make Twitter API call via HTTP request
3. Check result on completion

4.2 Sending Queue

Our app displays tweets that have already downloaded (from Core Data) when there is no network connection.

...when we send a tweet without I have no network connection, what is a good UX?

Hint: Apple's Mail application

4.3 Periodic Network Check

Use an NSTimer to check periodically if any tweets to send.

If so, try to send them again.
If no network access again, wait again.

All of this can happen transparent to the user -- but they should know about the "Outbox"

4.4 How Long Is A Piece of String?, I

Imagine the following:

1. User presses "Send" right as they get into a tunnel on the train: tweet enters "send queue".
2. User closes the app.
3. User re-opens the app ***3 months*** later.

Is it appropriate to still post that tweet when they re-open? Our app will try!

4.5 How Long Is A Piece of String?, II

Every new feature adds such ***edge cases*** (like the ***3 months later*** problem).

Handling edge cases well can be an *awesome* UX, but it also increases ***code complexity*** and ***testing requirements***.

It is up to you to design "what's reasonable" for your apps: what will you handle? what will you not handle?

What's On For Today?

Architecting a Twitter App:

1. Modeling & Data Flows
2. View Hierarchy
3. Feed Implementation
4. Publishing Implementation
5. Application Logic

5.0 So....What's Left?

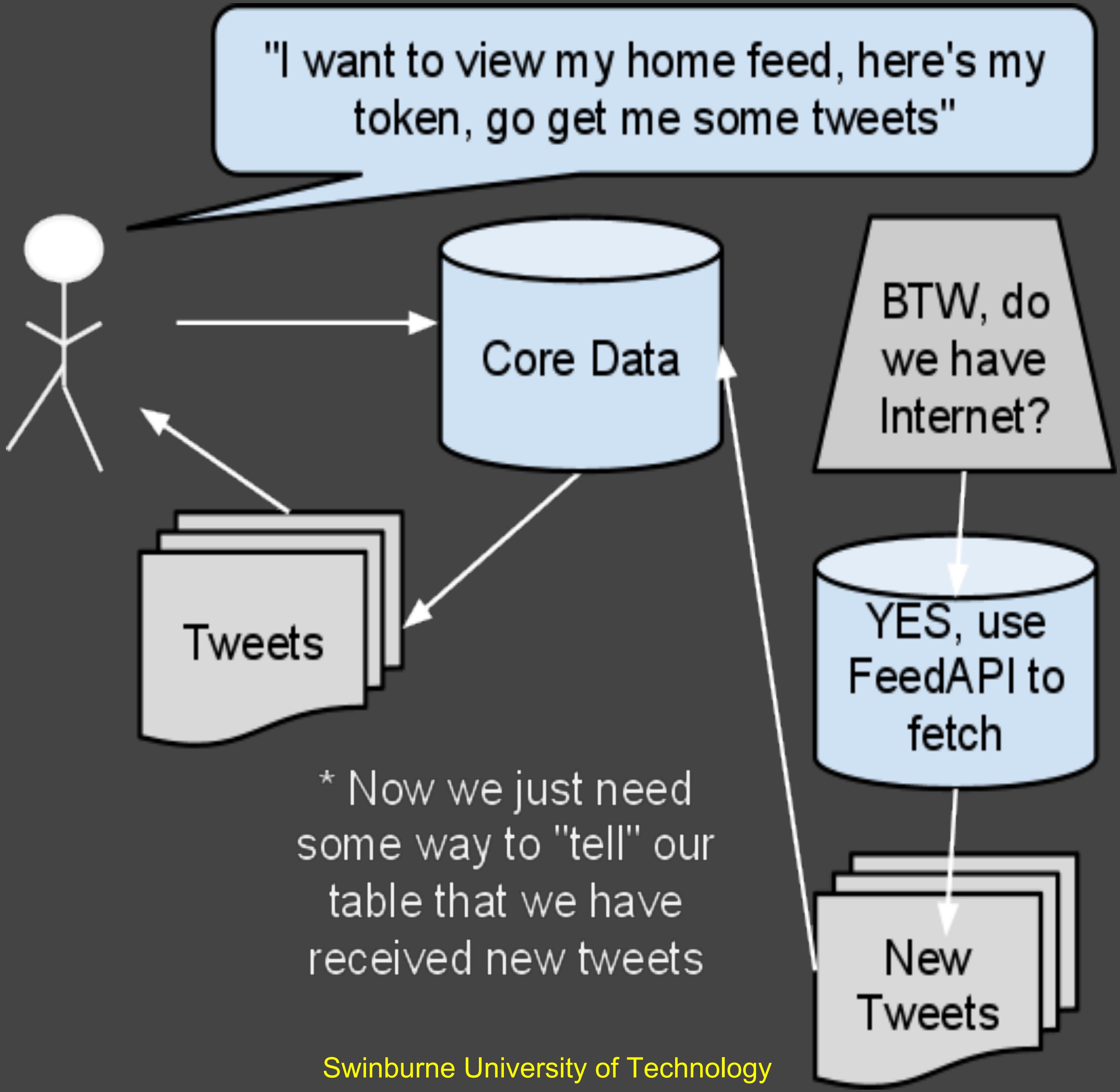
- FeedAPI is handling all getting & sending tweets
- UITableViewControllers display tweet lists
- Application view hierarchy set in MainWindow.xib

5.1 Core Data Logic

- Retrieve tweets from Core Data when Internet unavailable

...it makes more sense to put □***new tweets*** into Core Data, and then set up UITableViewController to read from Core Data all the time.

5.2 Time To Refactor Slides 1.16 and 1.17


"I want to view my home feed, here's my token, go get me some tweets"
Core Data
BTW, do we have Internet?
YES, use FeedAPI to fetch
Tweets
New Tweets

* Now we just need some way to "tell" our table that we have received new tweets

5.4 Notifications to the Rescue

Whenever NSManagedObjectContext is saved, it posts a notification:
NSManagedObjectContextDidSaveNotification
If UITableViewController listens for this notification, we can reload our table when we have new tweets, automatically.

5.5 User Logged In?

We need to hook up the "logging in" UI with the OAuth library.

Very easy, but when OAuth gives us a token - where do we put it?

5.6 NSUserDefaults

We can use NSUserDefaults to store the token across the user's sessions

```
NSUserDefaults *settings =
            [NSUserDefaults standardDefaults];
[settings setValue:token forKey:@"token"];
[settings synchronize];
```

When the user logs out, we just clear out the token:

```
[settings setValue:nil forKey:@"token"];
```

What We Covered Today

Architecting a Twitter App:

1. Modeling & Data Flows
2. View Hierarchy
3. Feed Implementation
4. Publishing Implementation
5. Application Logic

End of Lecture 9

1. Lab
2. Assignments
3. Get Coding