

# Safer Swift Code with Value Types

# What do I mean by safety?

## Medium


By A Medium Corporation

Open iTunes to buy and download apps.

[View More by This Developer](#)



[View in iTunes](#)

 This app is designed for both iPhone and iPad

### Description

Welcome to Medium for iOS, a simple app that lets you read and write the stories that matter most to you.

Every day thousands of new voices publish their unique experiences, views, and reflections to Medium.com, creating

[A Medium Corporation Web Site](#) ▶ [Medium Support](#) ▶

[...More](#)

### What's New in Version 1.14.1121

- Peter let a couple nasty bugs slip into our last release, these have been fixed
- Fired Peter

# Agenda

1. **What** are Value Types vs. Reference Types
2. **Why** is this topic relevant now?
3. **How:** A Practical Example of a "Value Oriented" Architecture

# Values vs. References

# Reference Types

```
class PersonRefType {  
    let name:String  
    var age:Int  
  
    // ..  
}  
// 1  
let peter = PersonRefType(name: "Peter", age: 36)  
// 2  
let peter2 = peter  
// 3  
peter2.age = 25  
// peter {"Peter", 25}  
// peter2 {"Peter", 25}
```

# Value Types

```
struct Person {  
    let name:String  
    var age:Int  
}  
// 1  
let petra = Person(name:"Petra", age:25)  
// 2  
var petra2 = petra  
// 3  
petra2.age = 20  
// petra    {"Petra", 25}  
// petra2   {"Petra", 20}
```

# Why now?

# Foundation / C Types

## Reference Types:

- NSArray
- NSSet
- NSData

## Value Types:

- NSInteger
- Struct



# Swift Standard Library

## Reference Types:

- `ManagedBuffer(?)`
- `NonObjectiveCBase(??)`

## Value Types:

- `Array`
- `String`
- `Optional`

# Enums and Structs in Swift are Powerful

- Can have variables
- Can have functions
- Can implement protocols

# So What Can't They Do?

“Indeed, in contrast to structs, Swift classes support implementation inheritance, (limited) reflection, deinitializers, and multiple owners.” - Andy Matushak (<http://www.objc.io/issue-16/swift-classes-vs-structs.html>)

# We've already been doing this!

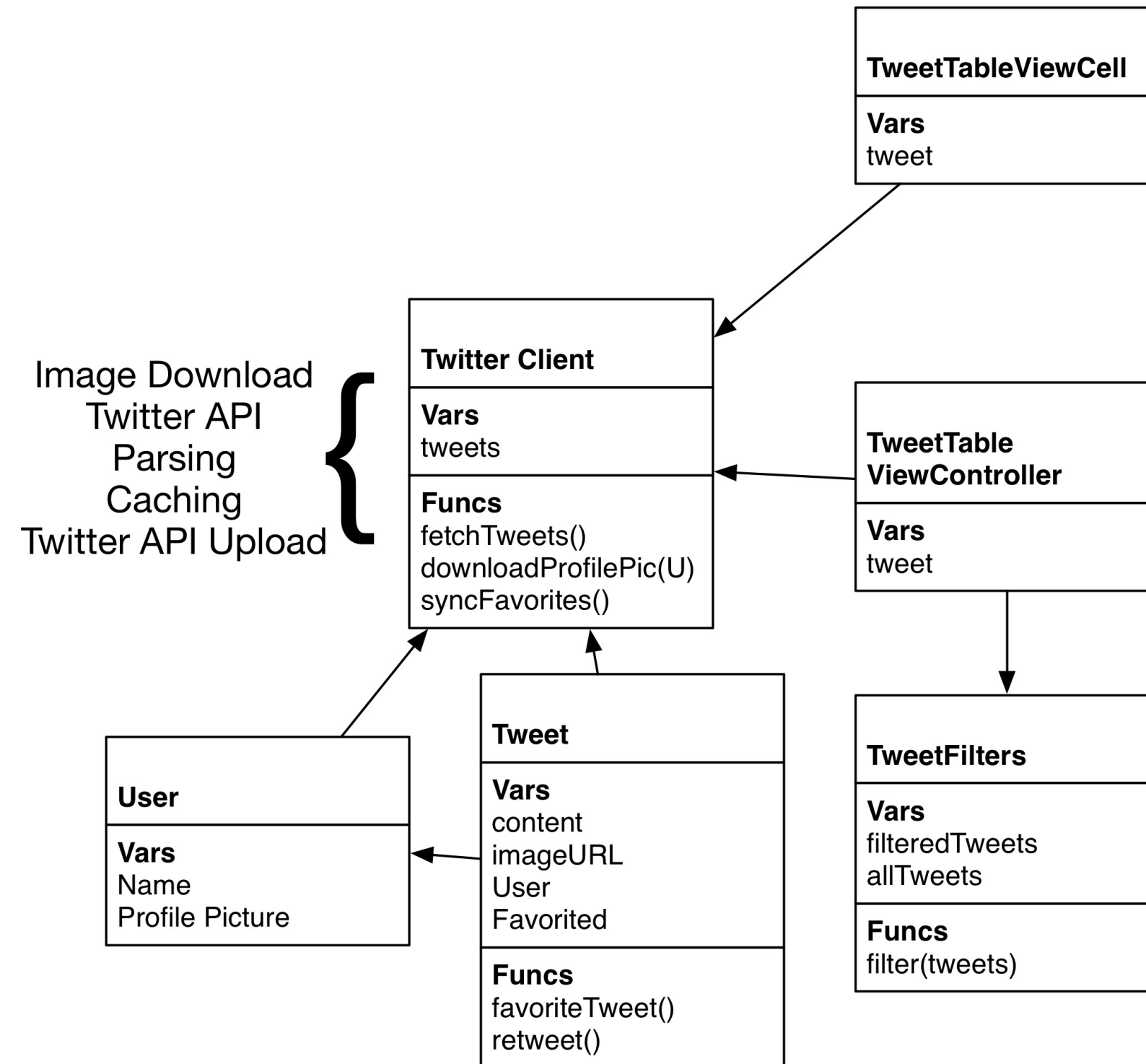
```
@property (copy) NSString *userName;
```

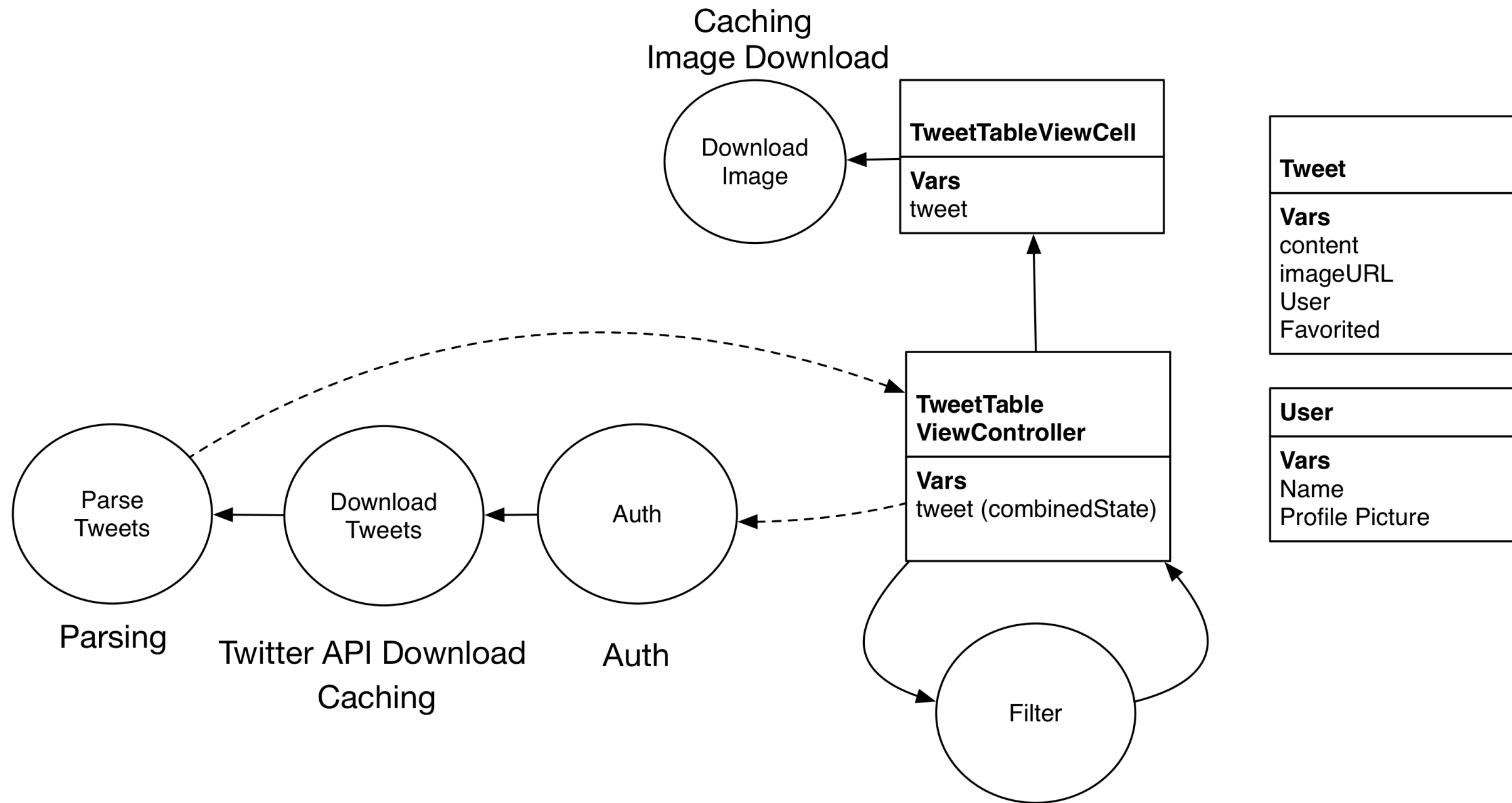
# Case Study

A Twitter Client Built on Immutable Value Types

# Twitter Client

1. Download the latest 200 tweets and display them
2. Allow to filter tweets (RT only, favorited tweets only, etc.)
3. Allow user to favorite tweets (should be synced with server)







# How can we favorite Tweets?

# It Is Very Simple with OOP

```
tweet.favorited = true
```

# It Is Simple with OOP

```
let lockQueue = dispatch_queue_create("com.happylocking", nil)
dispatch_sync(lockQueue) {
    tweet.favorited = true
}
```

# Is It Simple with OOP?

```
let lockQueue = dispatch_queue_create("com.happylocking", nil)
dispatch_sync(lockQueue) {
    tweet.favorited = true
    NotificationCenter.defaultCenter().
        postNotificationName("Tweet Changed", object: tweet)
}
```

# Modeling Change is Hard!

```
let lockQueue = dispatch_queue_create("com.happylocking", nil)
dispatch_sync(lockQueue) {
    tweet.favorited = true
    NotificationCenter.defaultCenter().
        postNotificationName("Tweet Changed", object: tweet)
    tweetAPIClient.markFavorited(tweet.identifier)
}
```

# Modeling Change is Hard!

- Protect against unwanted updates
- Distribute new value throughout application
- Understand what the underlying *identity* of an object is and perform update accordingly

# Modeling Change is Hard!

- Protect against unwanted updates
  - Distribute new value
  - Understand what the underlying *identity* of an object is and perform update accordingly
- > We need to this in all places where we mutate values!**

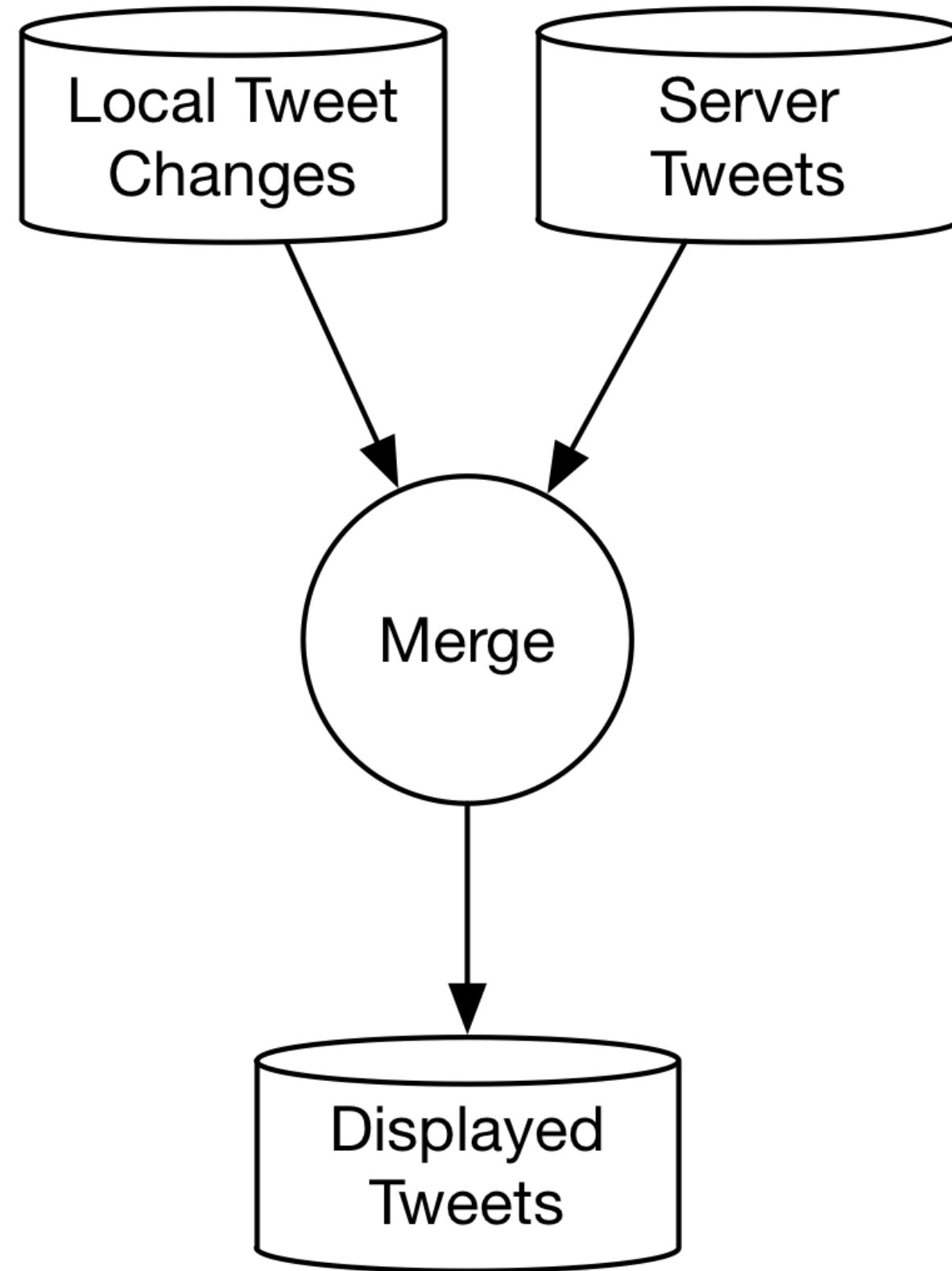
Modelling Change  
is Hard!



# How Can We Model Change With Immutable Value Types?

## Model change to values as values

- Create a new `Tweet` for every change
- Save these changes in a *Store*
- *Store* saves local changes and server state
- *Store* provides a merged view on list of tweets
- *Store* can trigger sync of local state to server



# Favoriting a Tweet

```
let currentTweet = tweetTableViewCell.tweet!
```

```
let newTweet = Tweet(  
    content: currentTweet.content,  
    identifier: currentTweet.identifier,  
    user: currentTweet.user,  
    type: currentTweet.type,  
    favoriteCount: currentTweet.favoriteCount,  
    isFavorited: !currentTweet.isFavorited  
)
```

```
store.addTweetChangeToLocalStorage(newTweet)
```

# Modelling Change

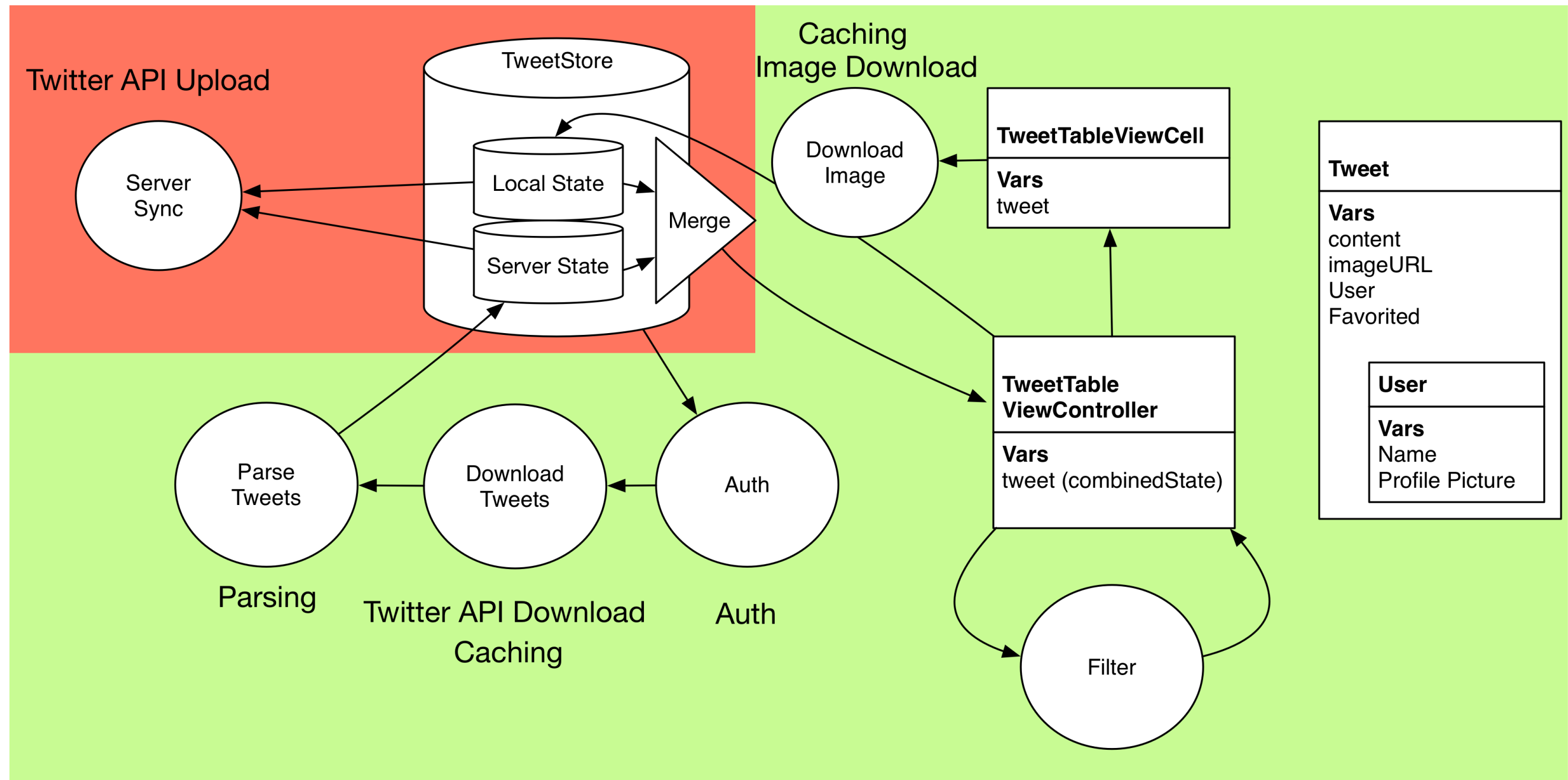
```
class TweetStore {  
    var tweets: [Tweet]? {  
        get {  
            var mergedList: [Tweet]? = nil  
  
            dispatch_sync(storeQueue) {  
                mergedList = mergeListIntoListLeftPriority(self.localState, self.serverTweets)  
            }  
  
            return mergedList  
        }  
    }  
  
    func addTweetChangeToLocalState(tweet: Tweet) {  
        dispatch_sync(storeQueue) {  
            let index = find(self.localState, tweet)  
            if let index = index {  
                self.localState[index] = tweet  
            } else {  
                self.localState.append(tweet)  
            }  
        }  
    }  
  
    // ...  
}
```

# Syncing Change

1. Iterate over each local change
2. Generate API request that syncs that local change to server
3. Upon each API response:
  - If success: remove tweet from local change set
  - If failure: leave tweet in local change set

# Syncing Change

```
struct StateMerge <T> {  
    let serverState: [T]  
    let localState: [T]  
}  
  
enum SyncResult <T> {  
    case Success(StateMerge<T>)  
    case Error(StateMerge<T>)  
}  
  
protocol StoreSync {  
    typealias StoreType  
  
    static func syncLocalState(merge: StateMerge<StoreType>)  
        -> Promise<SyncResult<StoreType>>  
}
```



# Benefits of "Value Oriented" Architecture

- Confidence that no one will change our data under the covers
- Change propagation needs to be handled explicitly
- Modeling change as data opens opportunities:
  - Undo Functionality
  - Sophisticated conflict resolution
  - ...



# Thank you!

**Example project:**

<https://github.com/Ben-G/TwitterSwift>

**Get in touch:**

@benjaminencz