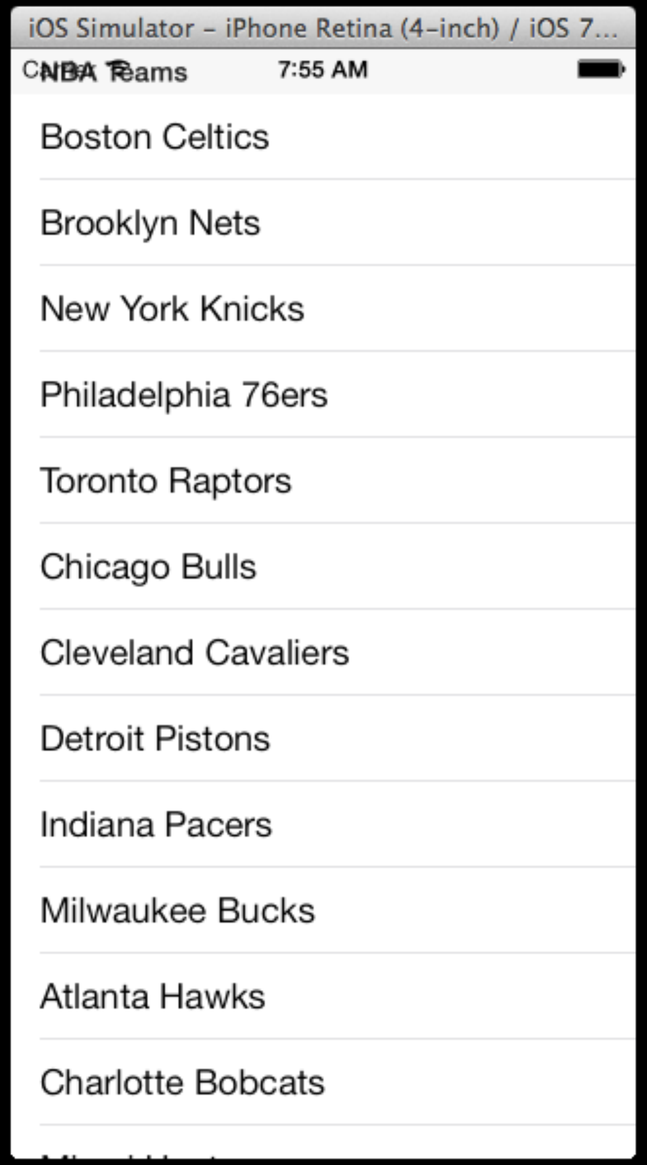
# iOS Foundations II
# Day 3

- TableView
- Delegation & Protocols
- Arrays
- Segues
- More Git!

# UITableView



# TableViews

- “A Tableview presents data in a scrollable list of multiple rows that may be divided into sections.”
- A Tableview only has one column and only scrolls vertically.
- A Tableview has 0 through n-1 sections, and those sections have 0 through n-1 rows. A lot of the time you will just have 1 section and its corresponding rows.
- Sections are displayed with headers and footers, and rows are displayed with Tableview Cells, both of which are just a subclass of UIView.

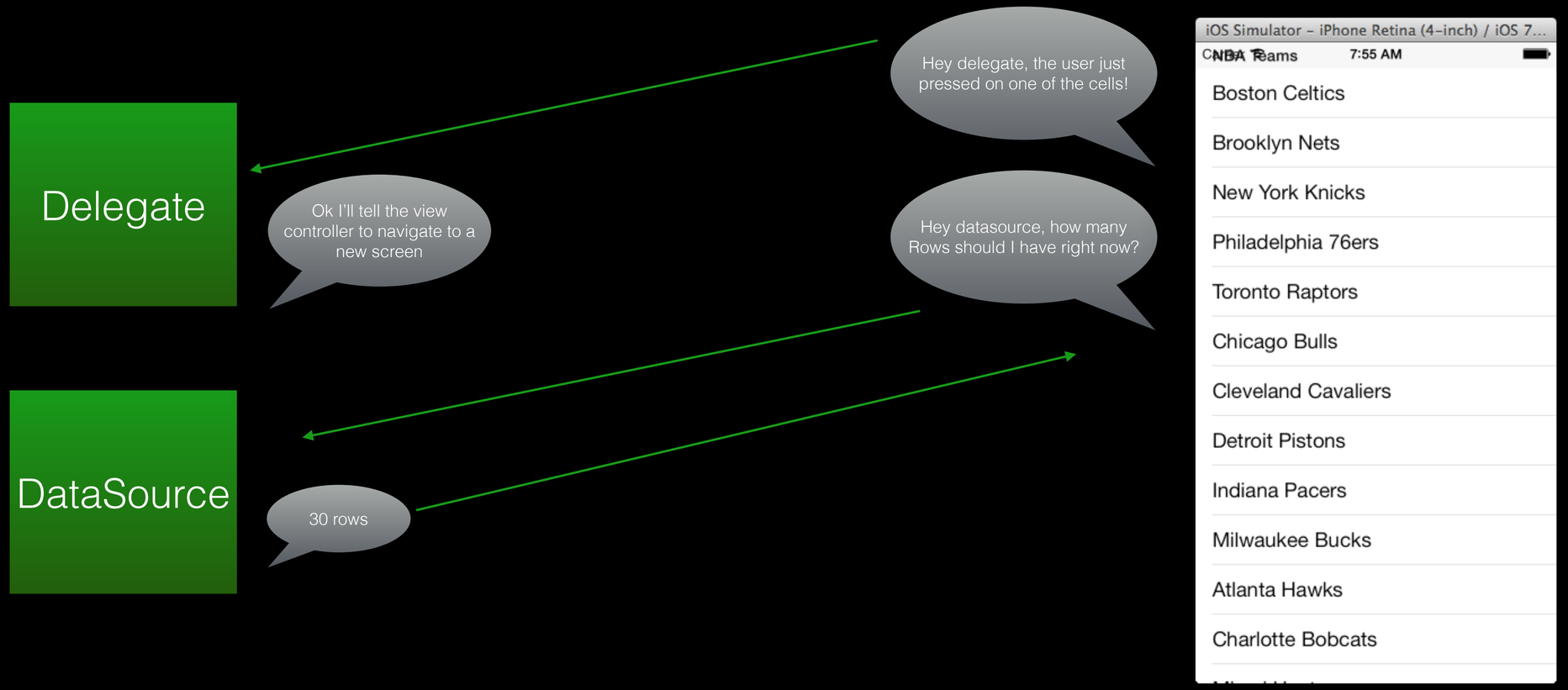
# How do Tableviews work?

- Tableview's rely, or **delegate**, onto other objects in order to function.
- A tableview asks another object how many rows and sections it should display, and what to display in each row.
- A tableview notifies another object when it is interacted with, like when the user selects a certain cell (aka row).
- So what is this pattern of one object relying on another called?

# Delegation

- Tableviews rely on the concept of delegation to get their job done.
- Picture time:

# TableViews and Delegates




A tableview has 2 delegate objects (instances of a class!). One actually called delegate and one called datasource. The datasource pattern is just a specialized form of delegation that is for data retrieval only.

# Delegation

- The whole point of delegation is to allow you to implement custom behavior without having to subclass.
- Apple could have designed UITableView's api so you would have to subclass UITableView, but then you would have to understand UITableviews in a lot more detail(which methods can I override? Do I have to call super? omfg?)
- **Delegates must adopt the protocol of the object they are being delegated from.**
- Delegation is used extensively in a large portion of Apple's frameworks.

# Protocols

- When an object wants to be something's delegate (or datasource), it needs to first conform to a protocol.
- Conforming to a protocol is like saying "hey, i can do all of these things, so let me be your delegate!"
- Sort of like signing a contract.
- To have your custom class conform to a protocol, just add the name of the protocol after the classes superclass:

```
class RootVC: UIViewController, UIPageViewControllerDelegate {
```

# TableViews

- A tableview requires 2 questions to be answered (aka methods to be implemented) and they are all in the datasource. At the very least the tableview needs data to display. It doesn't have to respond to user interaction, so the delegate object is completely optional.
- `tableView(numberOfRowsInSection:) How many rows am I going to display?`
- `tableView(cellForRowAtIndexPath:) What cell do you want for the row at this index?`
- Number of sections is actually optional, and is 1 by default.

# Demo

# Arrays

- **An Array stores multiple values of the same type in an ordered list.**
- Arrays are very important to understand
- Arrays are used in virtually every app ever made!!!!
- You typically use arrays any time you have a collection of similar objects or data and you want to perform similar operations on them and/or keep them in order.
- Arrays are considered a collection type.

# An Array's type in Swift

- The type of an array can be written the long way or short 'literal' way:
  - Long way:

    Array<String>

  - Short way:

    [String]

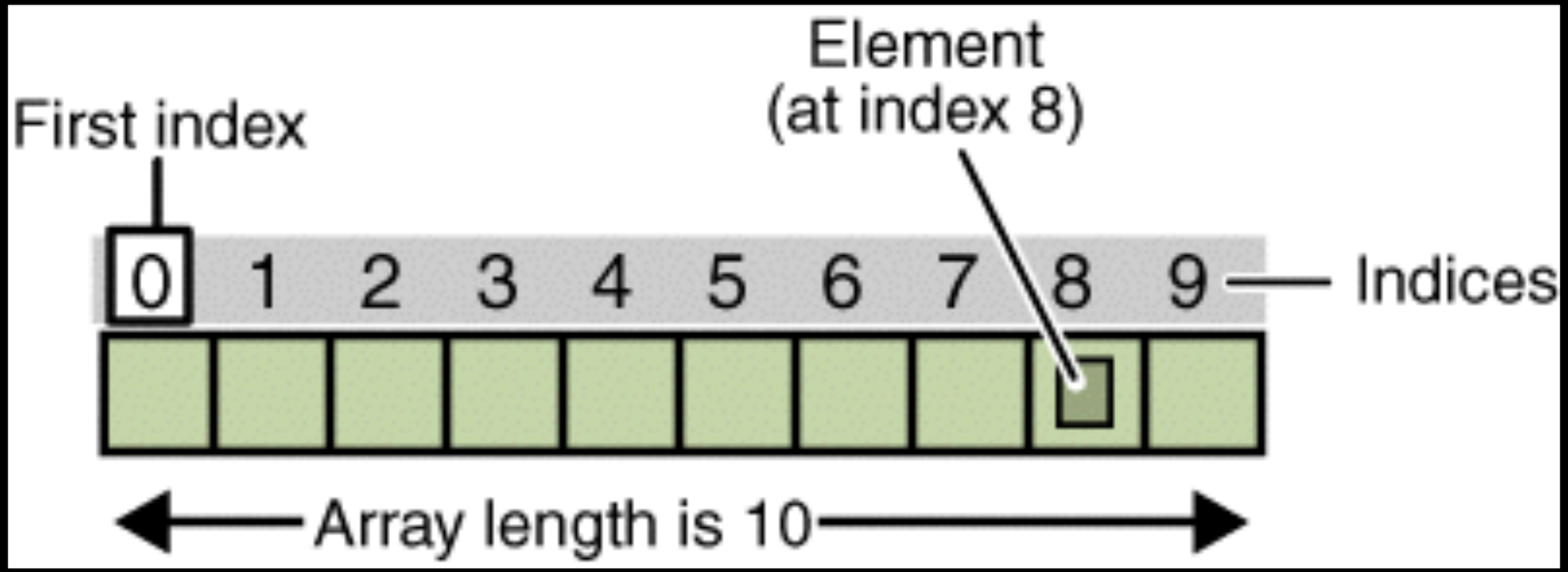
- Both ways are functionally identical, and the **short hand way is always preferred.**

# Creating an array in Swift

- We can use array literals to initialize an array.
- Array literals are a short hand, and the preferred way to create an array with one or more values.
- An array is written as a comma separated list of values surrounded by a pair of square brackets:

```
var names = ["Brad","Andy","John"]
```

# Array Visual



# Accessing an item in the array

- You can retrieve a value from the array by using **subscript syntax**.
- You do this by passing the index of the value you want to retrieve inside square brackets immediately after the name of the array:

```
var names = ["Brad","Andy","John"]
var person = names[1]
println(person) //prints Andy
var anotherPerson = names[2]
println(anotherPerson) //prints John
```

- You can access the total number of values in an array with the .count property on Arrays.

# Demo

# TableViews and Arrays

- When a tableview asks its datasource what it should display in each cell, it does so by sending a method for each cell that is going to be on screen.
- Think about how this would work without an array:

```
func tableView(tableView: UITableView!, cellForRowAtIndexPath indexPath: NSIndexPath!) -> UITableViewCell! {
    if indexPath.row == 0 {
       // return "Brad"
        }
    else if indexPath.row == 1 {
       // return "John"
        }
    else if indexPath.row == 2 {
        //return "Russell"
        }
    else if indexPath.row == 3{
            //return "Sherman"
        }
    else if indexPath.row == 4 {
        //return "Pete"
        }
}
```

# TableViews and Arrays

- now the achieving the exact same functionality with an array:

```
func tableView(tableView: UITableView!, cellForRowAtIndexPath indexPath: NSIndexPath!) -> UITableViewCell! {

    // return self.names[indexPath.row]
}
```

# TableViews Gotchas

1. Did you give the tableview a data source? (by setting up an outlet to it and then settings it datasource property, or wiring it up via storyboard)
2. Does your view controller conform to the data source protocol?
3. Did you implement the required 2 methods? (how many rows, and what cell for each row)
4. Did you drag out a tableview cell, and then set its reuse identifier?
5. Does the reuse identifier in your storyboard match the one in your code? “Did you forget your quotes in your code?” -Vick

# Demo

# Segues

- Segues are provided by the storyboard to help you easily transition from one view controller to another.
- There are 2 primary segues that are used : Show and Present
- Show refers to pushing a view controller onto the navigation stack, it usually will slide in from the right or left.
- Present refers to modally presenting a view controller, it usually slides up from the bottom.
- You can create your own custom segue to customize the behavior to match your needs.

# Segues

- 2 primary methods for dealing with segues in code:
    1. performSegueWithIdentifier:
    2. prepareForSegue:sender:

# performSegueWithIdentifier

- Call this method in your View Controller to trigger a segue in code.
- If you your segue isn't hooked up to be triggered by an action in your interface, you will need to call this method to trigger the segue.

# prepareForSegue:sender:

- This method is called on the source view controller of the segue, right before the segue is actually performed.
- The first paramater is the segue itself, which is an instance of the UISegue class.
- The most important property of a UISegue instance is the destinationViewController property, which gives you a reference to the view controller you are about to segue to.
- This is a great spot to pass information to the next screen.

# Demo

# GitHub

- Github is a repository web-based hosting service.
- Github hosts people's repositories, and those repositories can be public or private.
- The repositories on Github are just like the repositories that you have on your own machine, except Github's web application has additional features that makes working in a team much easier.

- https://help.github.com/articles/set-up-git/

# Remotes

- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- So when you have a repository on Github, it is considered a remote repository.
- When you import a directory into a git project with git init, eventually you may want to create a remote repository on Github and push to it. This gives you a backup in the cloud, and also lets your work be seen by others.
- Or you can clone an already existing remote repository (your own or someone elses) to get a local copy of the remote repository on your machine.

# GitHub and Git

- Github and your local git install have 2 ways of communicating:
  - https (recommended)
  - ssh
- Both of these forms of communication require authentication.
- For https, it is recommended you use a credential helper so you don't have to enter in your credentials every time you interact with a remote repository.
- For SSH, you can generate SSH keys on your computer and then register those SSH keys to your Github account.
- Lets all follow the steps at https://help.github.com/articles/set-up-git/ together to get our github & git properly setup.

# demo

# Git Remote Commands

- git remote -v shows you all the remotes you have configured for your local repository on your machine
- use git remote add <nickname> <url> to add a remote repo
- after committing, use git push <nickname> <branch> to push your committed changes to your remote
- use git pull to automatically fetch and merge a remote branch into your current local branch

# Demo