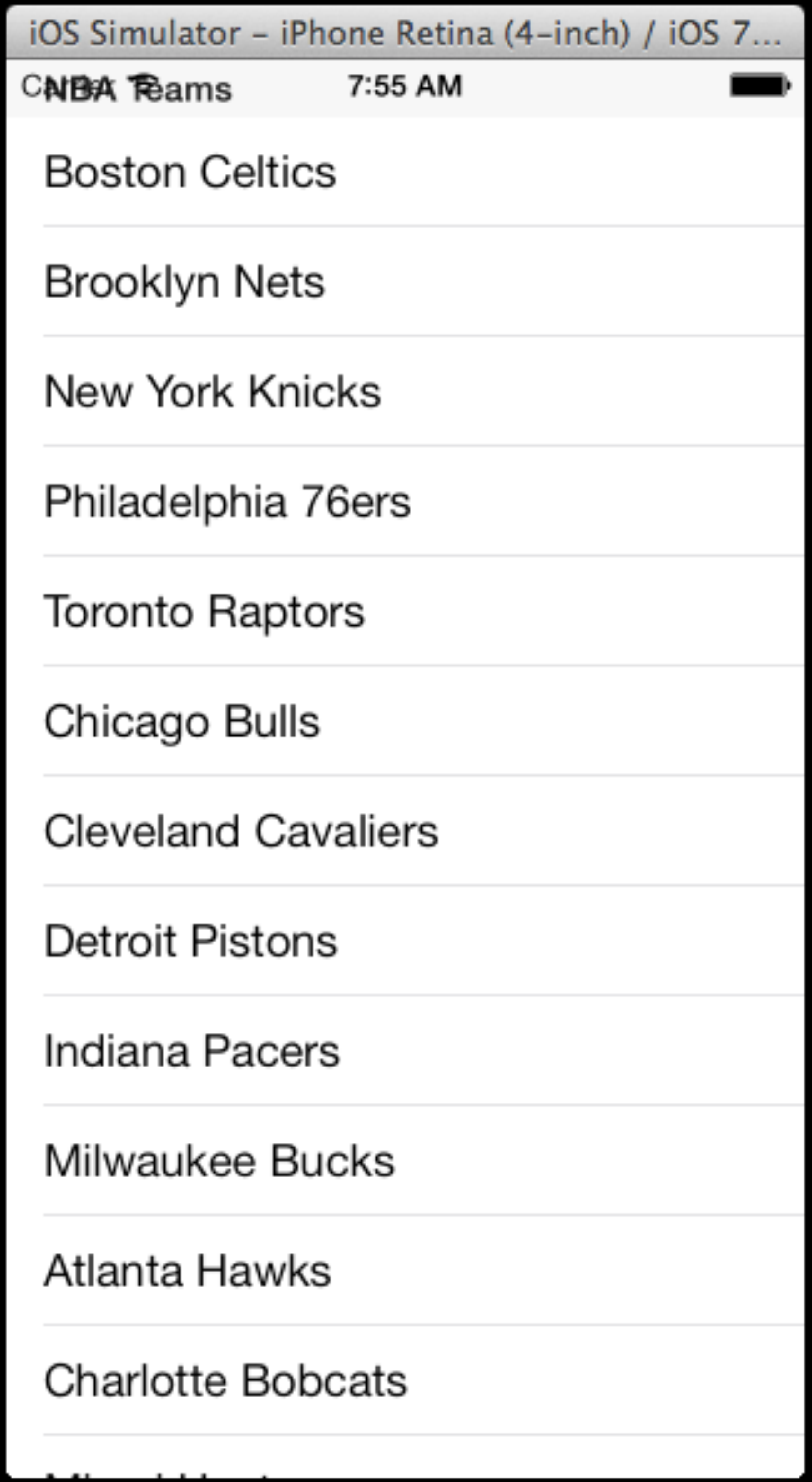
iOS Foundations II
Day 3

- Survey: Canvas / anonymous
- Day 1 HW status check
- Day 2 review
  - ARC/strong/weak
  - Arrays
  - IBOutlet/Action
  - View lifecycle
- Day 2 HW Q & A

- TableViews, Protocols, Delegates
- git mini-lesson, part 2

Happy Belated Independence Day!

UITableView



Boston Celtics
Brooklyn Nets
New York Knicks
Philadelphia 76ers
Toronto Raptors
Chicago Bulls
Cleveland Cavaliers
Detroit Pistons
Indiana Pacers
Milwaukee Bucks
Atlanta Hawks
Charlotte Bobcats

TableViews

- “A Tableview presents data in a scrollable list of multiple rows that may be divided into sections.”
- **One** column and scrolls only **vertically**
- Tableview: Any # of sections; each section can have any # of rows.

  (Often you’ll have only **one** section that contains several rows.)
- Sections shown w/ headers and footers.

  Rows shown with Tableview Cells.

  Both are subclasses of UIView (note: class *inheritance*!)

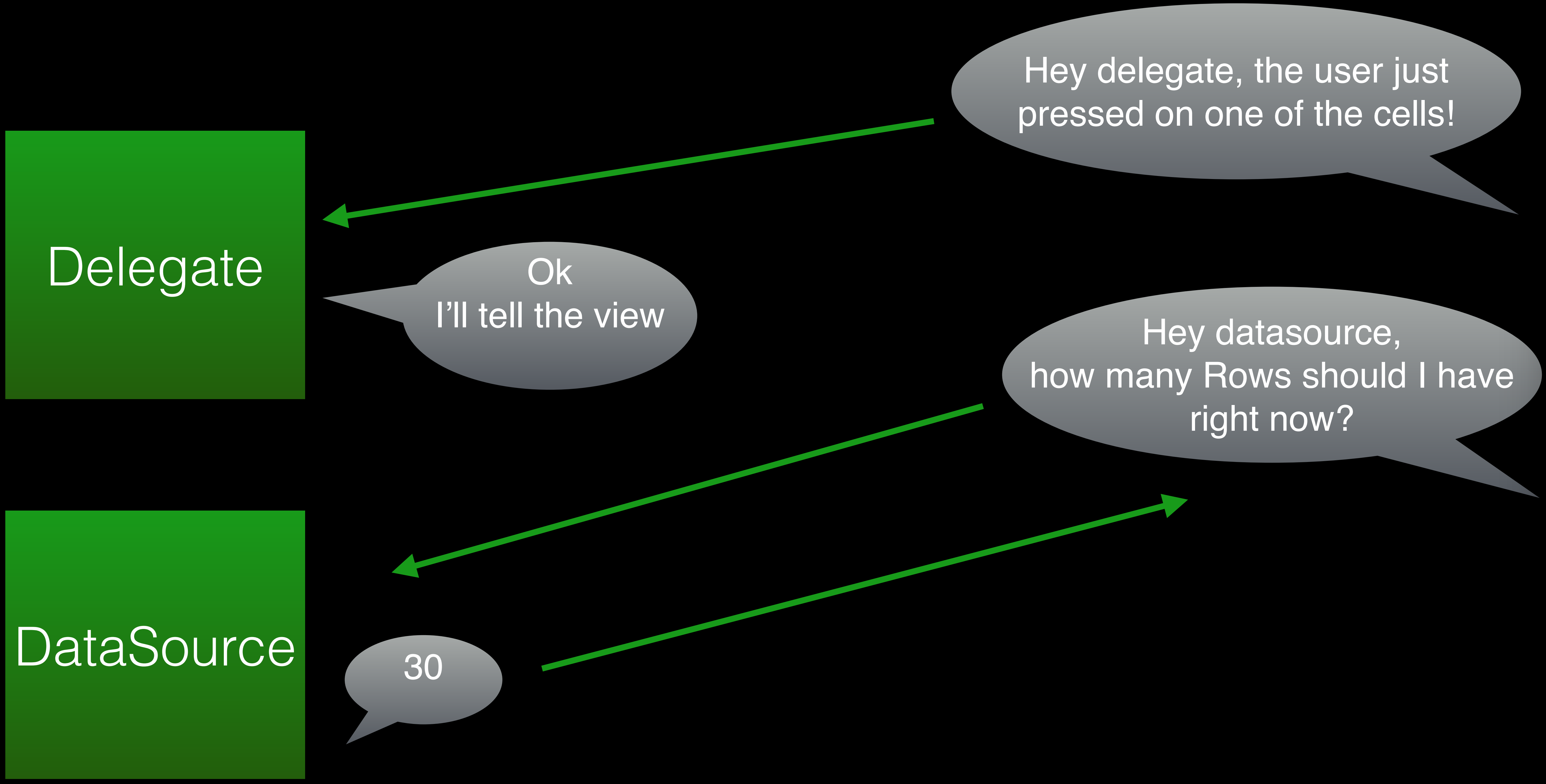
How do Tableviews work?

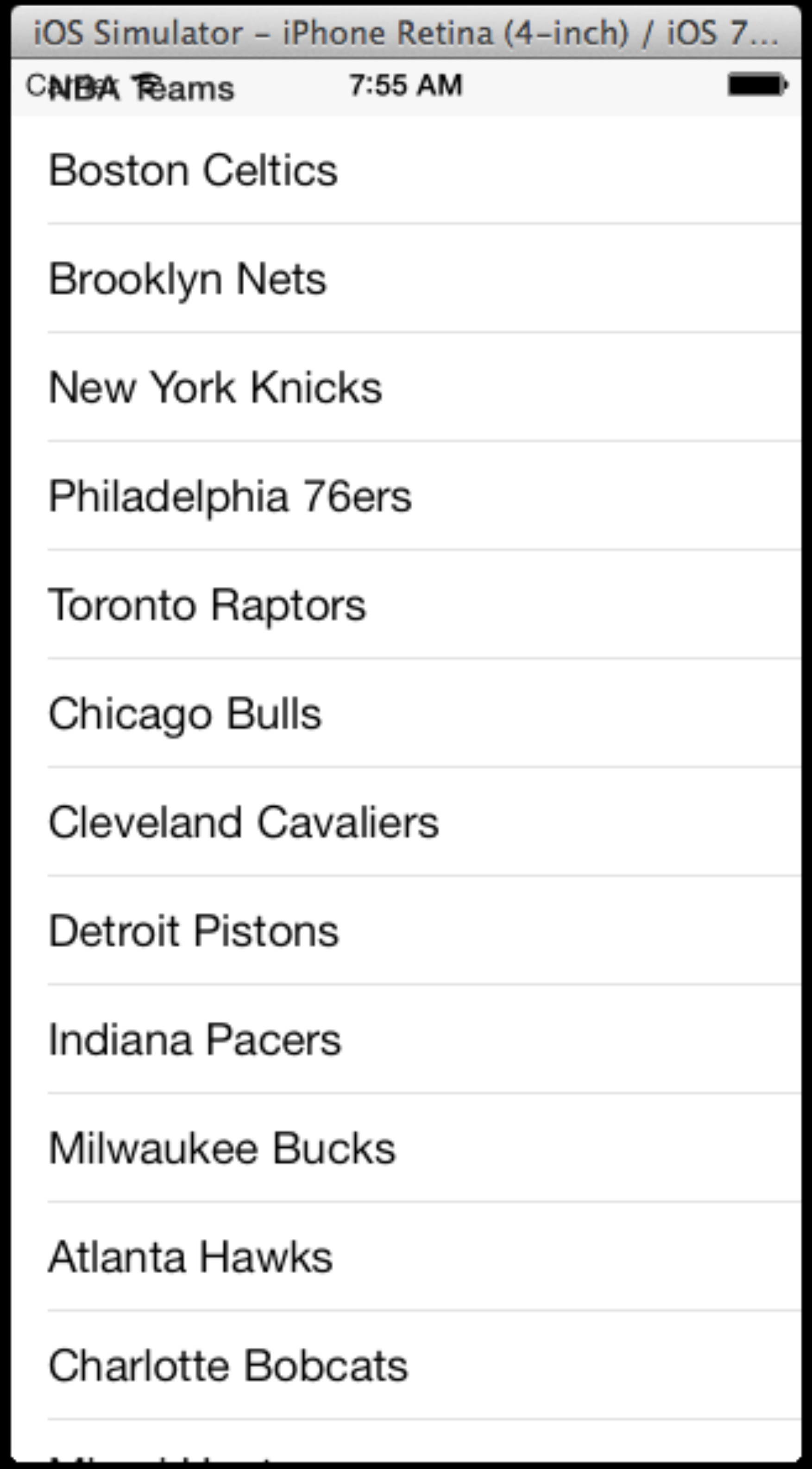
- Tableview’s rely on, or **delegate**, onto other objects in order to function.
- A tableview asks another object how many rows and sections it should display, and what to display in each row.
- A tableview notifies another object when it is interacted with, like when the user selects a certain cell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

Delegation

- Tableviews rely on the design pattern of **delegation** to get the job done.
- Picture time:

TableViews and Delegates


Hey delegate, the user just pressed on one of the cells!
Delegate
Ok
I'll tell the view
Hey datasource,
how many Rows should I have right now?
DataSource
30


iOS Simulator – iPhone Retina (4-inch) / iOS 7...

NBA Teams 7:55 AM

Boston Celtics
Brooklyn Nets
New York Knicks
Philadelphia 76ers
Toronto Raptors
Chicago Bulls
Cleveland Cavaliers
Detroit Pistons
Indiana Pacers
Milwaukee Bucks
Atlanta Hawks
Charlotte Bobcats

A tableview uses two delegate objects:

. One is actually called **delegate**, the other is called **data source**.

. The data source is a special delegate for **data retrieval** only.

Delegation vs datasource

- Design pattern: “**Delegation”.** Definition: “A simple and powerful design pattern in which one object in a program acts on behalf, or in coordination with, another object.”
- The **delegate** object acts on behalf (or coordinates with) another object.
- E.g., table view’s **delegate** acts on behalf of the table view whenever the user interacts with the table view.
- A **data source** is almost identical to a delegate. The key difference: the **data source** is in control of data, while a **delegate** is normally in control of a user interface attribute or event.

Delegation

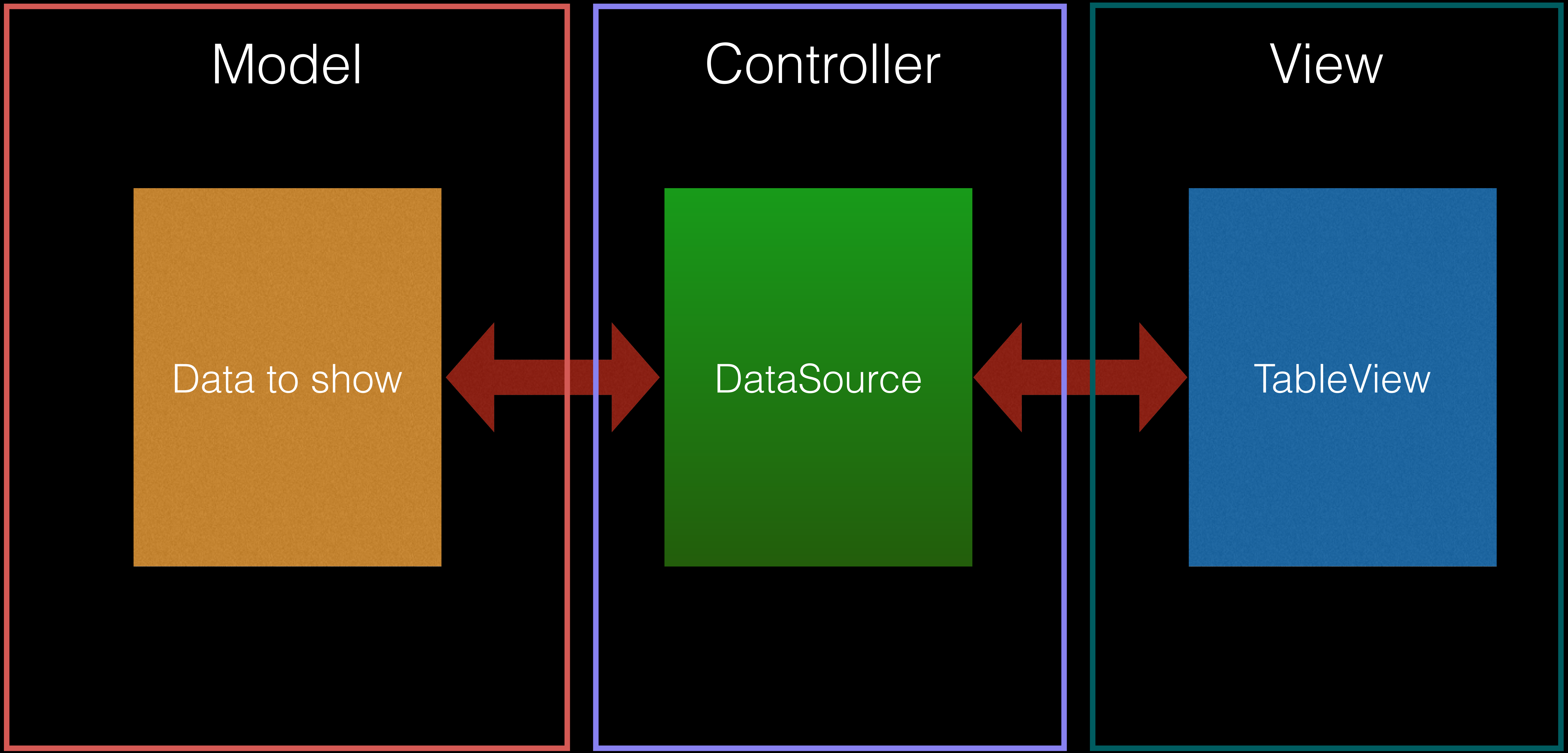
- Another benefit of delegation is to allow you to implement custom behavior without having to subclass.
- Apple could have designed UITableView's api so you would have to subclass UITableView, but then you would have to understand UITableviews in a lot more detail(which methods can I override? Do I have to call super? omfg?)
- Instead, you can just create a custom delegate/datasource object, which will do the customizations for the table view.
- Delegation is used extensively in a large portion of Apple's frameworks.
- **Delegates/Datasources must adopt the protocol of the object they are being delegated from.**

Protocols & Delegates

- Protocol
    - List of methods (func names & *signatures*)
    - Used as an interface, like an API
- Delegate
    - Variable that conforms to a protocol

Demo 3.1:
Protocols & Delegates

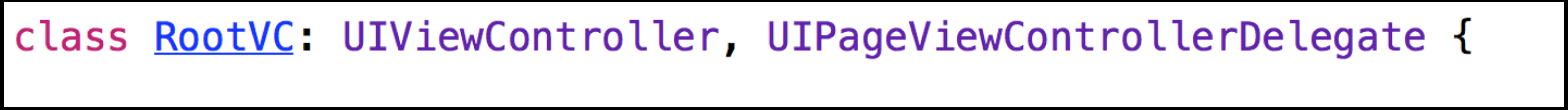
TableViews and its data source



Delegation helps greatly with the separation of concerns in our MVC environment

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/datasource!”
- 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 {
```

`TableView Datasource Protocol`

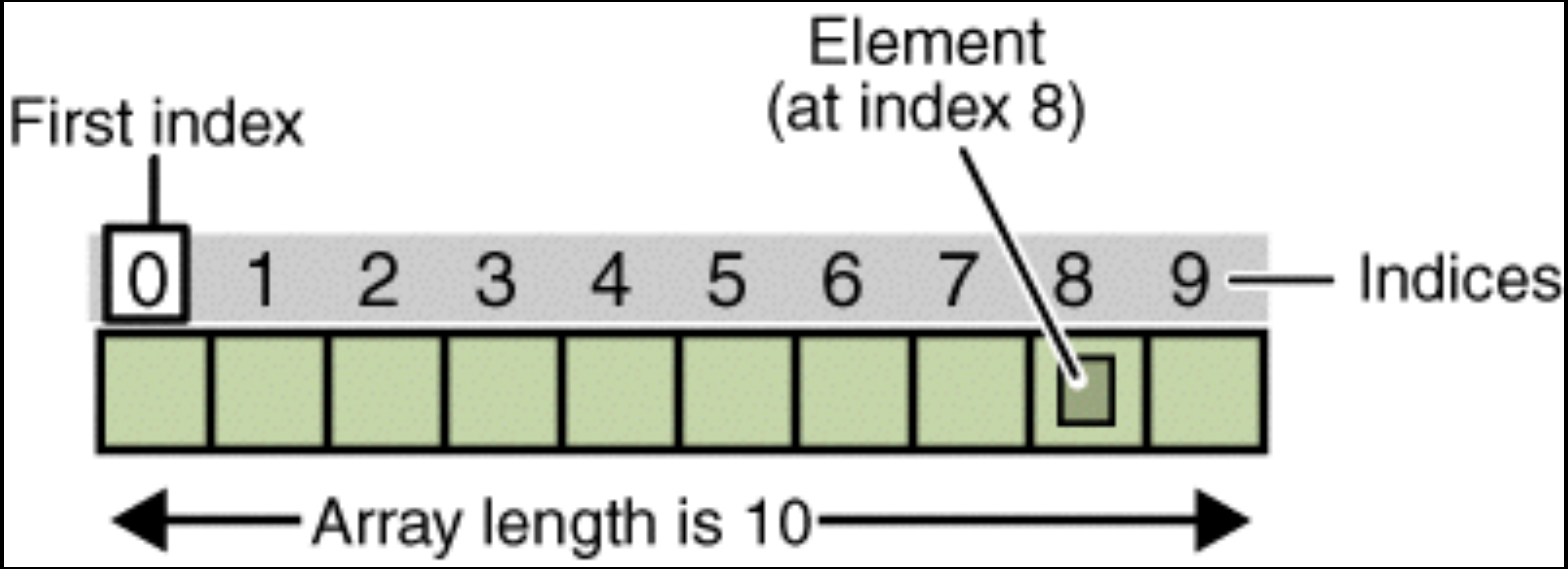
- A tableview needs two questions to be answered (i.e., two methods implemented) by their **data source**. At the very least, the tableview needs data to display. It doesn't have to respond to user interaction, so the "delegate" object is optional.
- `tableView(numberOfRowsInSection:) How many rows am 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.

Table View and arrays

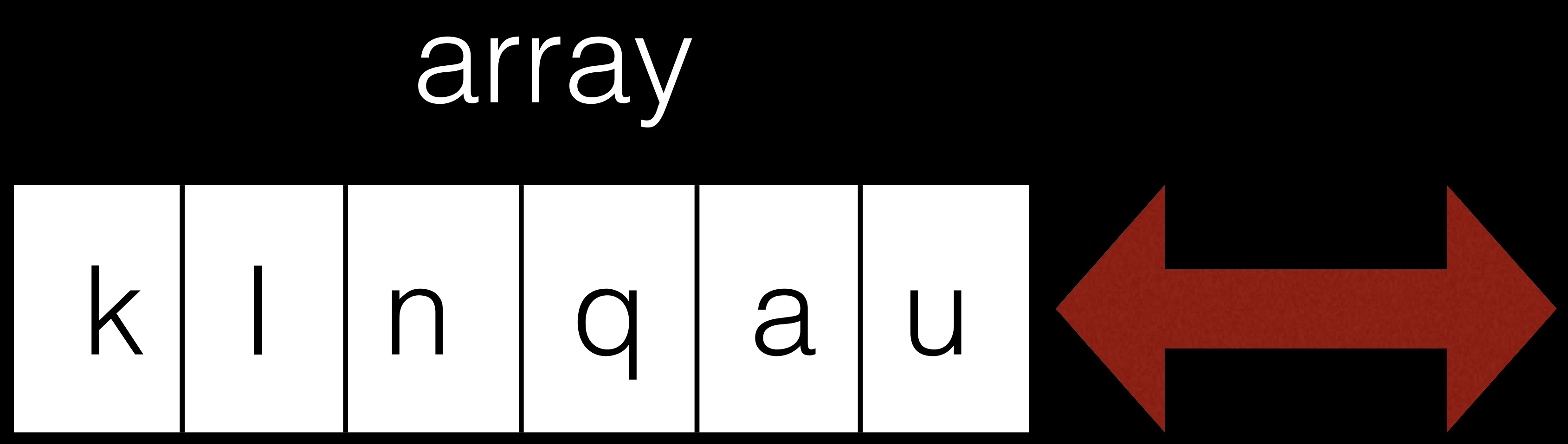
TableView+Array

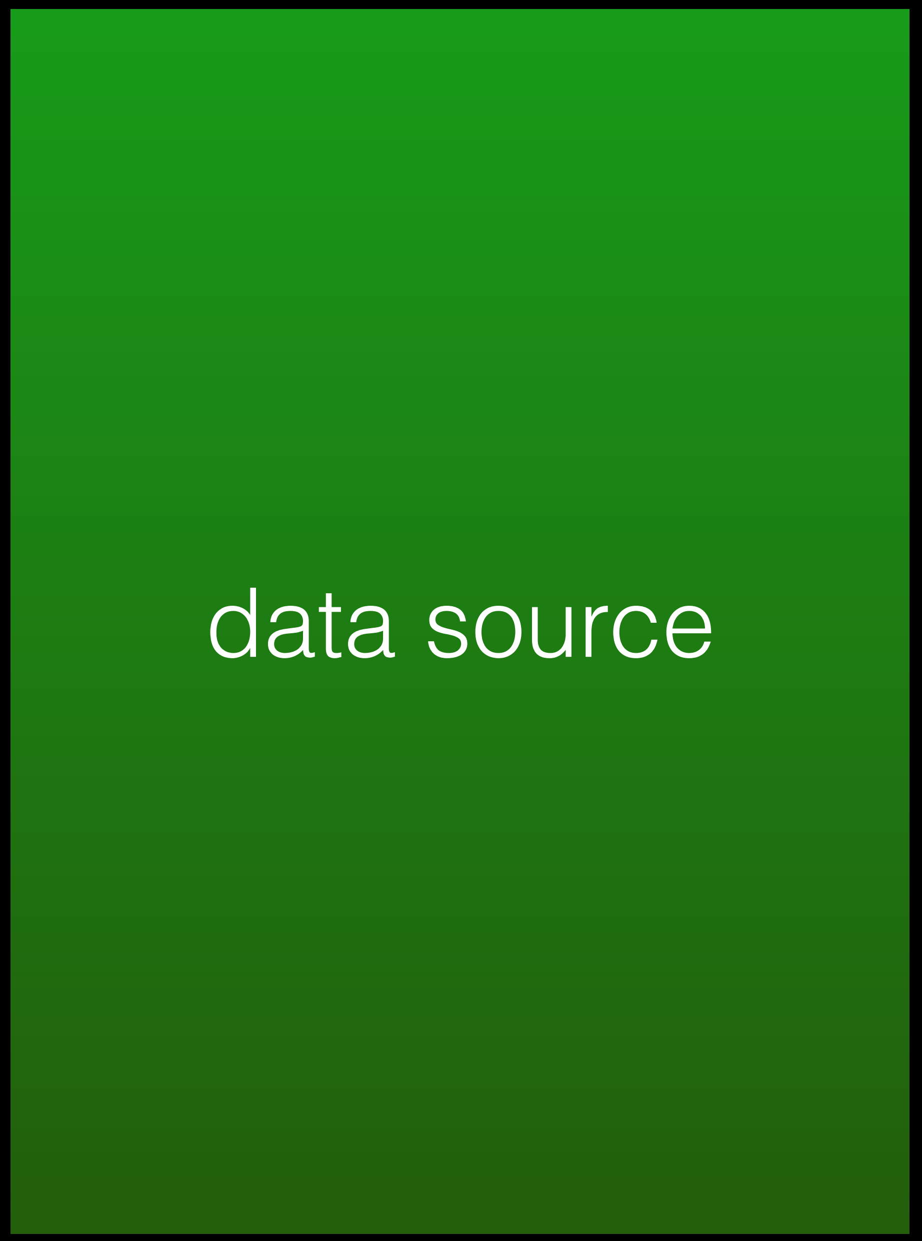
- 99% of the time, table views are ‘backed’ by an array.
- So basically the table view’s data source is the middle man between the array, and where its going to be shown, the table view
- The table view uses the count of the array to figure out how many cells its going to display
- And then for each cell, it displays whatever is at that index of the array

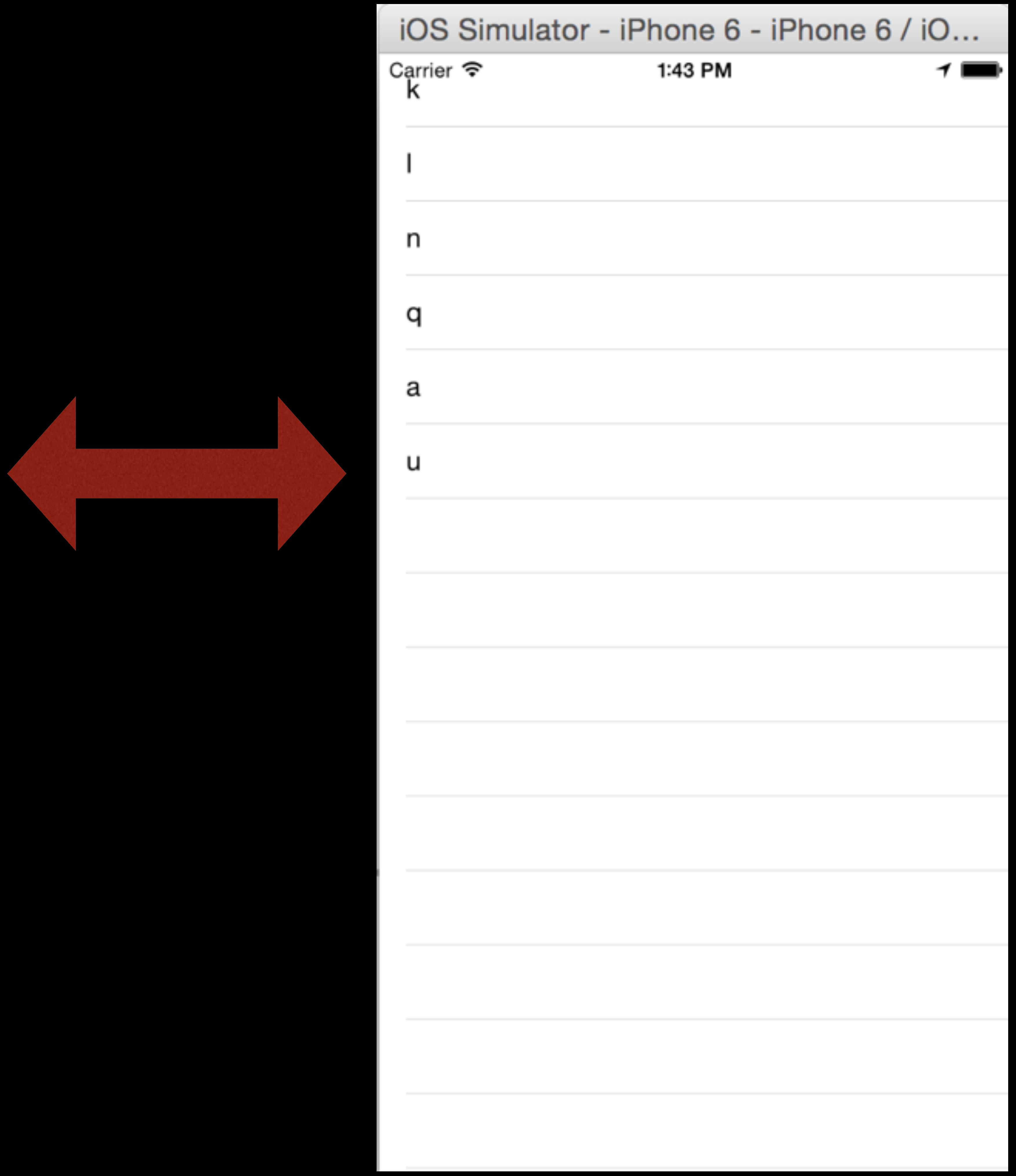
Array Visual



TableView+Array







Demo 3.2:
Xcode UI, interactive

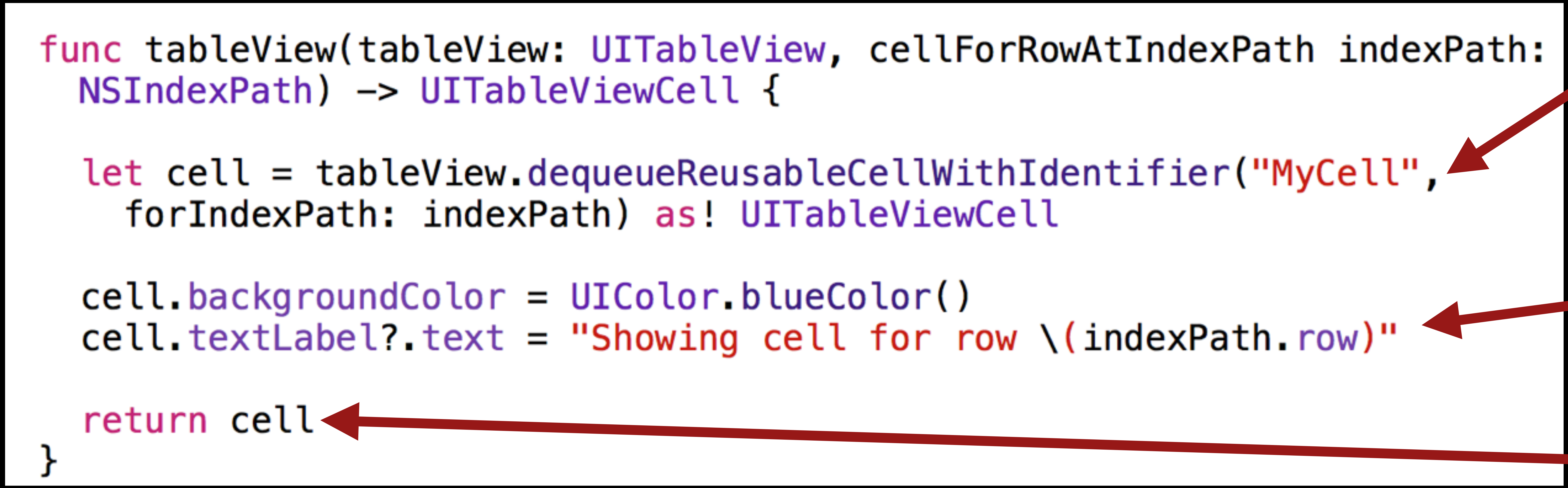
Cell Reuse

Reusing Cells

- Table views are very efficient
- They can be asked to show lists of thousands, and even millions of objects
- They do this by only generating enough cells to fill the screen, and then reusing cells as they are scrolled off the screen, immediately placing them back on screen.
- So showing an array of ten thousand strings, the table view only has to generate 7 or 8 cells (enough to fill the screen, depends on screen size and row height)

tableView(cellForRowAtIndexPath:)

- lets look at the data source cellForRow method in depth:



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

   let cell = tableView.dequeueReusableCellWithIdentifier("MyCell",
     forIndexPath: indexPath) as! UITableViewCell

   cell.backgroundColor = UIColor.blueColor()
   cell.textLabel?.text = "Showing cell for row \(indexPath.row)"

   return cell
}
```

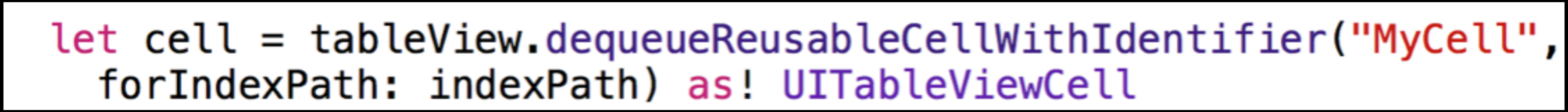
1) dequeue the cell

2) configure the cell

3) return the cell

These are the 3 basic steps you will always follow when dequeueing a cell

Properly reusing cells



```
let cell = tableView.dequeueReusableCellWithIdentifier("MyCell",
    forIndexPath: indexPath) as! UITableViewCell
```

- When you dequeue a cell, you are asking the table view for a cell. The table view pulls a cell from its ‘reuse’ queue.
- The identifier you pass in must match the reuse identifier you set on storyboard
- You need the as! UITableViewCell because the method dequeueReusableCellWithIdentifier returns a type of AnyObject. as! is a forced downcast, which is like saying “I know this method is actually going to return something of this type. In fact, I’m so confident, crash the app if it doesn't't”

Demo 3.3:
Xcode UI, interactive

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?

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.

GitHub

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

Remote Repositories

- Remote repositories are copies of a project hosted external to your local machine (e.g. local or remote server). Virtual or physical separation.
- Your repository on github (e.g. for submitting homework) is a remote repository.
- After you import a folder into a git project using **`git init`**, you may eventually want to create a remote repository on Github and push to it. This backs up your code to the Cloud, and lets others see your work.
- Or you can clone an existing remote repository (your own or someone else's) to make a local copy of the remote on your machine.

Mini-lesson on git

- Specify the text editor you'll use for commit messages

```
$ git config --global core.editor emacs
```

- Add workspace file to local repo:

```
$ git add [file] # "Staged" status; marked for repo, but not in yet
$ git commit # Copy file from workspace to local repo and update metadata
```

- Synchronize local repo with remote repo

```
$ git push
```

Mini-lesson on git

- Remove file from remote repo:

```
$ git rm [file] # Deletes local file
$ git add -A # Removes file from local
  repo
$ git push # Removes file from remote
  repo
```

Mini-lesson on git

- Edit `gitTest/student_names.txt` by adding your name to the file (anywhere in the file)

```
$ git commit
$ git status
```

- Push local (committed) changes to the same remote repo you cloned. On your **first** "git push", git will prompt you for your github email & password. After first push, git should remember your login info.

```
$ git push
```