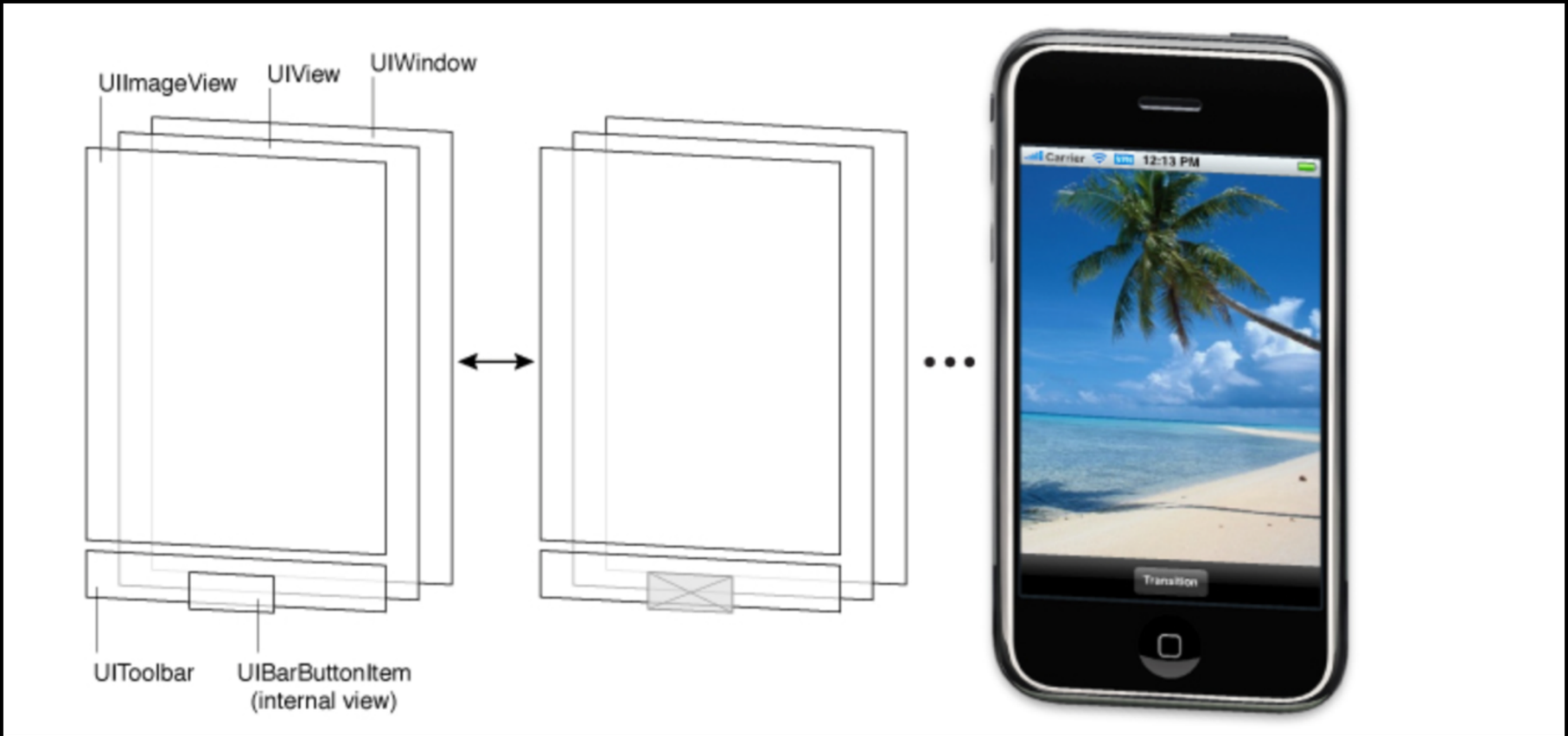
iOS Foundations II
Session 5

- UIView's and the view hierarchy
- AutoLayout AGAIN
- UITextFields
- Table View Reloading

Views

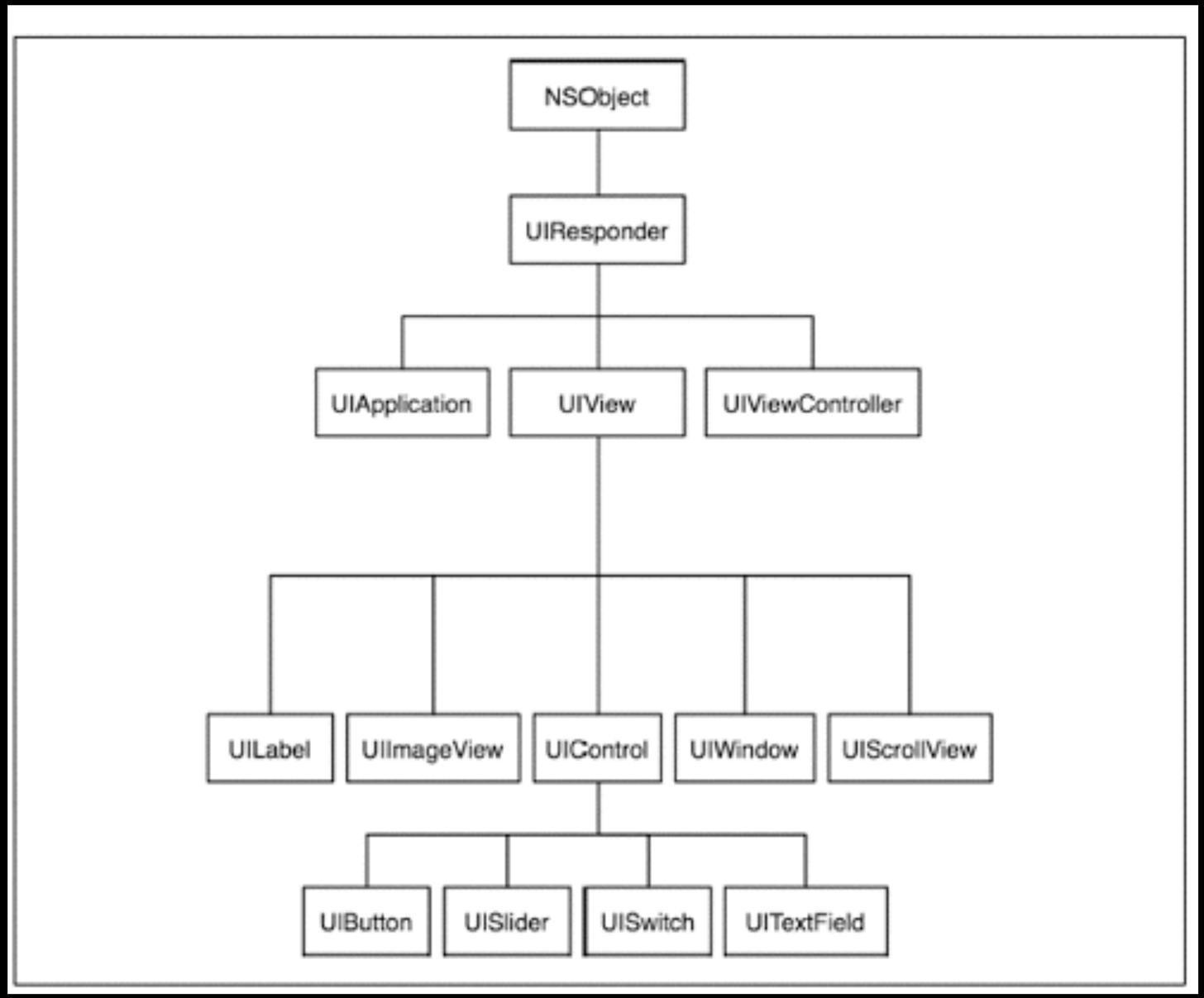


- “A view is an instance of the `UIView` class (or one of its subclasses) and manages a rectangular area in your application window”
- “Views are responsible for drawing content, handling multitouch events, and managing the layout of any subviews. “

Views

- To get onscreen, a UIView must be added as a subview of a parent view. All views can be a child view of a parent view, and all views can have children views of their own. This is called the view hierarchy.
- A UIView has many properties that dictate its appearance: backgroundColor, alpha, hidden, opaque, tintColor, layer, clipsToBounds, etc.
- But the most important property of a UIView have to do with its location and size: its called ‘frame’

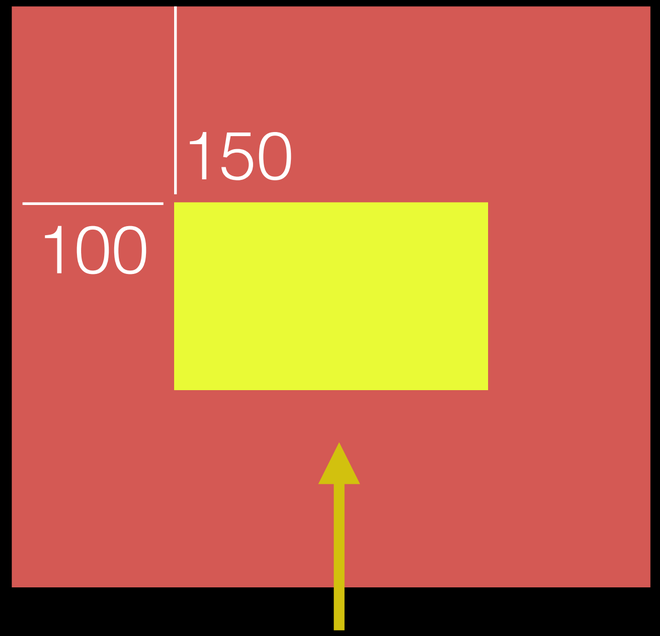
UIViews are EVERYWHERE



Frame

- **Frame** - Describes the view's location and size **in its superview's coordinate system.**
- This property is of type CGRect.
- A CGRect is a struct that has 2 properties of its own: Origin and Size
- Origin is a point that specifies the coordinates of the rectangle's origin (top left corner).
- Size specifies the height and width of the rectangle.
- Demo

Frame



yellowView’s Frame

- x = 100
- y = 130
- Width = 150
- Height = 100

Demo

Autolayout

- A constraint-based layout system for making user interfaces.
- AutoLayout works by you bossing it around.
- Specifically, you can tell it 2 things about every view in your interface:
    1. Size - How big the view is going to be
    2. Location - Where the object is going to be located in its super view
- Once told ‘the rules’, autolayout will enforce the rules you setup.
- These rules are setup using constraints.

Constraints

- Constraints are the fundamental building block of autolayout.
- Constraints contain rules for the layout of your interface's elements.
- You could give a 50 point height constraint to an imageView, which constrains that view to always have a 50 point height. Or you give it a constraint to always be 20 points from the bottom of its superview.
- Constraints can work together, but sometimes they conflict with other constraints.
- At runtime Autolayout considers all constraints, and then calculates the positions and sizes that bests satisfies all the constraints.

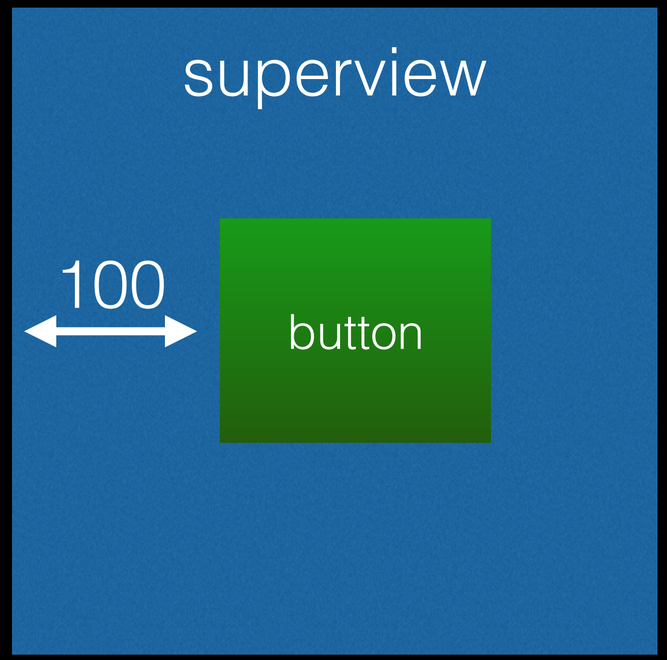
Demo

Attributes

- When you attach constraints to a view, you attach them using attributes.
- The attributes are: **left/leading, right/trailing, top, bottom, width, height, centerX, centerY.**
- Attributes tell the constraint what part of the view(s) they should be manipulating

Attributes

- So if you attach a constraint of 100 points from a button's left attribute to its container's left attribute, thats saying "I want the left side of this button to be 100 points over from its super view's left side"



Demo

Constraints + Attributes = Math time

- “You can think of a constraint as a mathematical representation of a human-expressable statement”
- So if you say “the left edge should be 20 points from the left edge of its containing view”
- This translates to button.left = container.left x 1.0 + 20
- which is in the form of y = mx+b
- **first attribute = second attribute * multiplier + constant**
- In the case of an absolute value, like pinning height, width, centerX, or centerY, the second attribute is nil.
- You can change the constants in code as an easy way to programmatically adjust your interface.

Demo

Storyboard and Autolayout

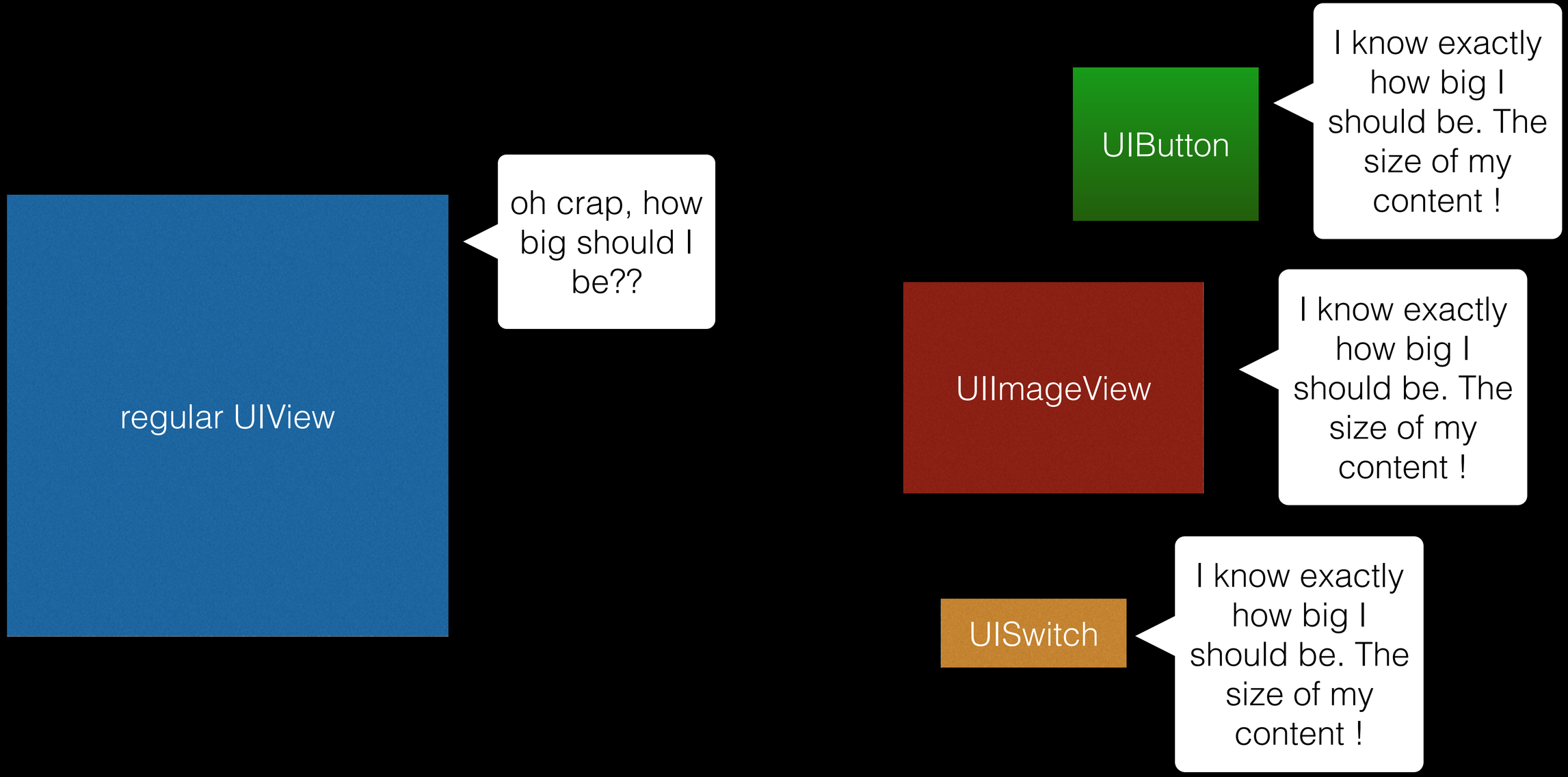
- Storyboard makes setting up autolayout pretty intuitive and painless, and even though you can setup autolayout completely in code, Apple strongly recommends doing it in storyboard.
- Xcode will let you build your app even if you have constraints that are conflicting and incorrect, but Apple says you should never ship an app like that.
- **When you drag a object onto your interface, it starts out with no constraints.**
- **Once you apply one constraint, that view needs to have constraints dictating its size AND location**

Demo

Intrinsic Content Size

- Intrinsic content size is the minimum size a view needs to display its content.
- Its available for certain UIView subclasses:
  - UIButton & UILabel: these views are as large as they need to display their full text
  - UIImageView: image views have a size big enough to display their entire image. This can change with its content mode.
- You will know a view has an intrinsic content size if autolayout doesn't require its size to be described with constraints.

Intrinsic Content Size



Demo

UITextField

- “A `UITextField` object is a control that displays editable text and sends an action message to a target object when the user presses the return button.”
- A text field can have a delegate to handle editing related notifications.
- When a user taps into a text field, the text field becomes the first responder and it brings the keyboard on screen.
- You are responsible for making sure the text field you are editing is not covered by the keyboard.
- **A UITextField is a subclass of UIView**

UITextField

- UITextfield has an intrinsic content size!
- It takes the size of the place holder text inside of it. There are two types of placeholder text:
  - Text: regular text that the textfield starts off with
  - Placeholder text: greyed out place holder text, used to tell the user what should go inside the field. Once a single character of real text is added to the text field, the place holder text vanishes.
- If you provide no text or placeholder text, the textfield intrinsically takes a very small width of 26 points.

Demo

UITextField Keyboard Dismissal

- By default the textfield does not dismiss the keyboard once the user hits return.
- The delegate of the textfield can be notified when the user hits return by implementing this method:
  - textFieldShouldReturn(textField : UITextField)
- In the implementation of this method, call resignFirstResponder() on the textfield passed in
- You can return true in this method, since pressing return in a textfield doesn't do anything, since it only supports one line.

Demo

Table View Reloading

- If the data the table view is showing is ever changed, you must manually tell the table view to reload itself.
- You can either tell the table view to reload all of its visible rows (easy but more expensive), or reload individual rows (cheaper, but a bit more work)

Table View Reloading

- reloading the entire table view (easy but not most efficient):

```
self.tableView.reloadData()
```

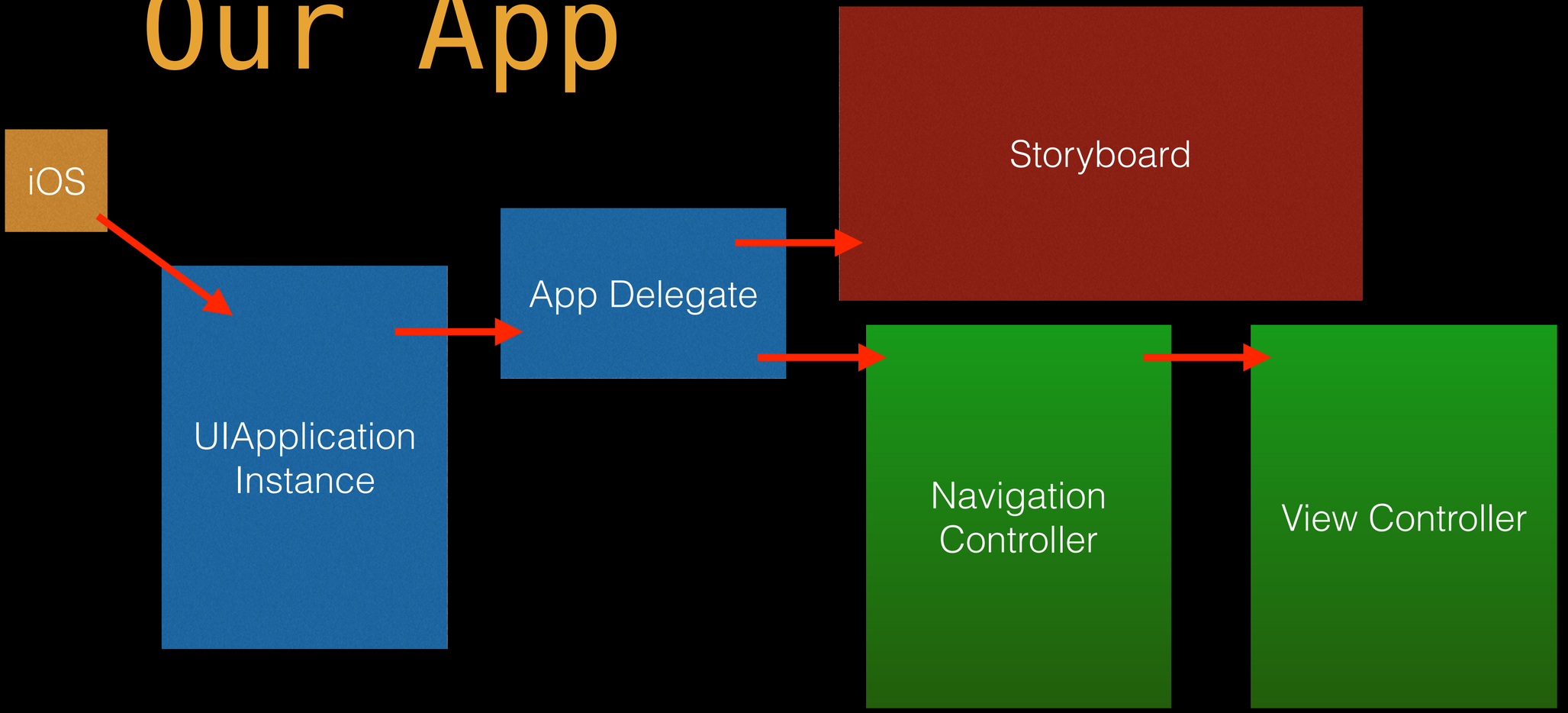
- reloading select indexPaths(harder but more efficient):

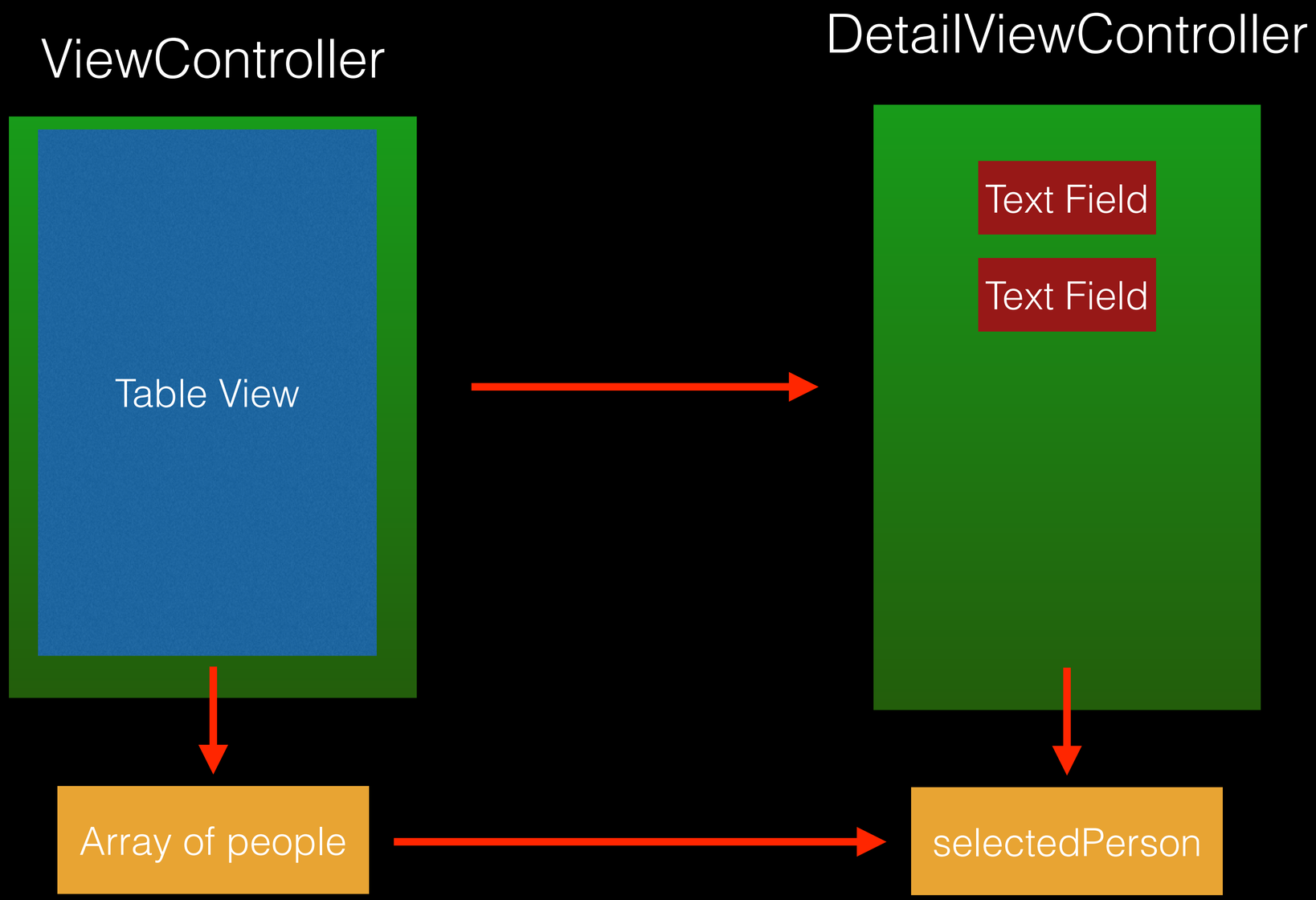
```
let indexPathToReload = NSIndexPath(forItem: 3, inSection: 0)
let indexPaths = [indexPathToReload]

self.tableView.reloadRowsAtIndexPaths(indexPaths, withRowAnimation:
  UITableViewRowAnimation.Fade)
```

Demo

Our App


iOS
UIApplication Instance
App Delegate
Storyboard
Navigation Controller
View Controller


ViewController
DetailViewController
Table View
Text Field
Text Field
Array of people
selectedPerson

Any changes made to your selectedPerson object are automatically reflected in our array, since Person is a class, and objects are pass by reference