

# CS193P - Lecture 6

## iPhone Application Development

Designing iPhone Applications  
Model-View-Controller (Why and How?)  
View Controllers

# Announcements

- Assignment 3 is due tomorrow at 11:59pm
  - Questions?
- Presence 1 is due next Tuesday (4/28)
- Friday's optional section...
  - "Preparing Apps for the App Store"
  - 200-205, 3:15PM

# Announcements

- Online resources for auditors and iTunes U viewers
  - <http://groups.google.com/group/iphone-appdev-auditors>
  - <http://cs193p.com>
  - Not affiliated with Stanford or Apple
  - Don't forget <http://devforums.apple.com>

# Announcements

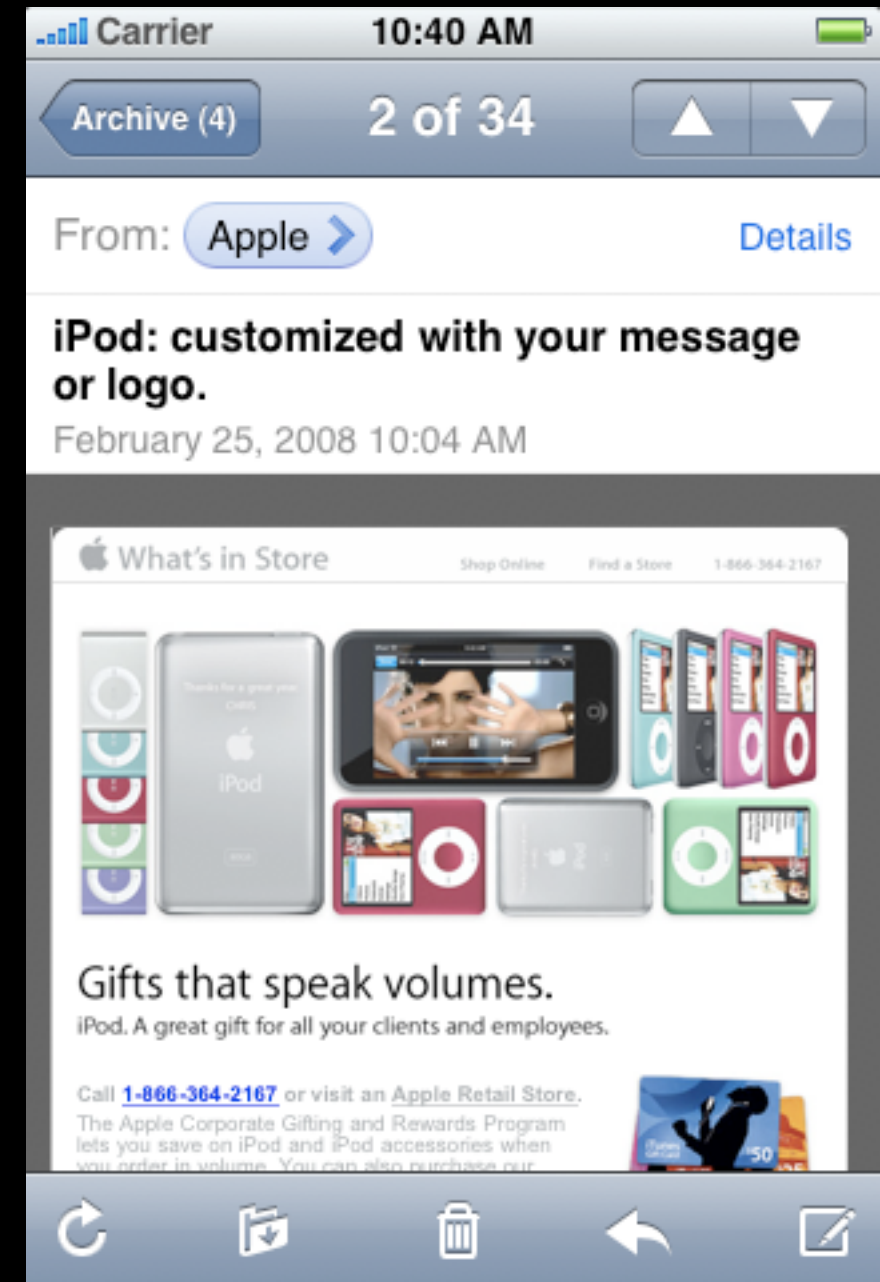
- Many requests for us to post assignment solutions online
  - Short answer: We're lazy
  - Longer answer: There are parts of the course that we reuse from quarter to quarter, so we won't be distributing solutions
  - Discussing assignments is fine
    - If you're a Stanford student, remember the Honor Code
  - We request that you **don't distribute completed assignments**

# Today's Topics

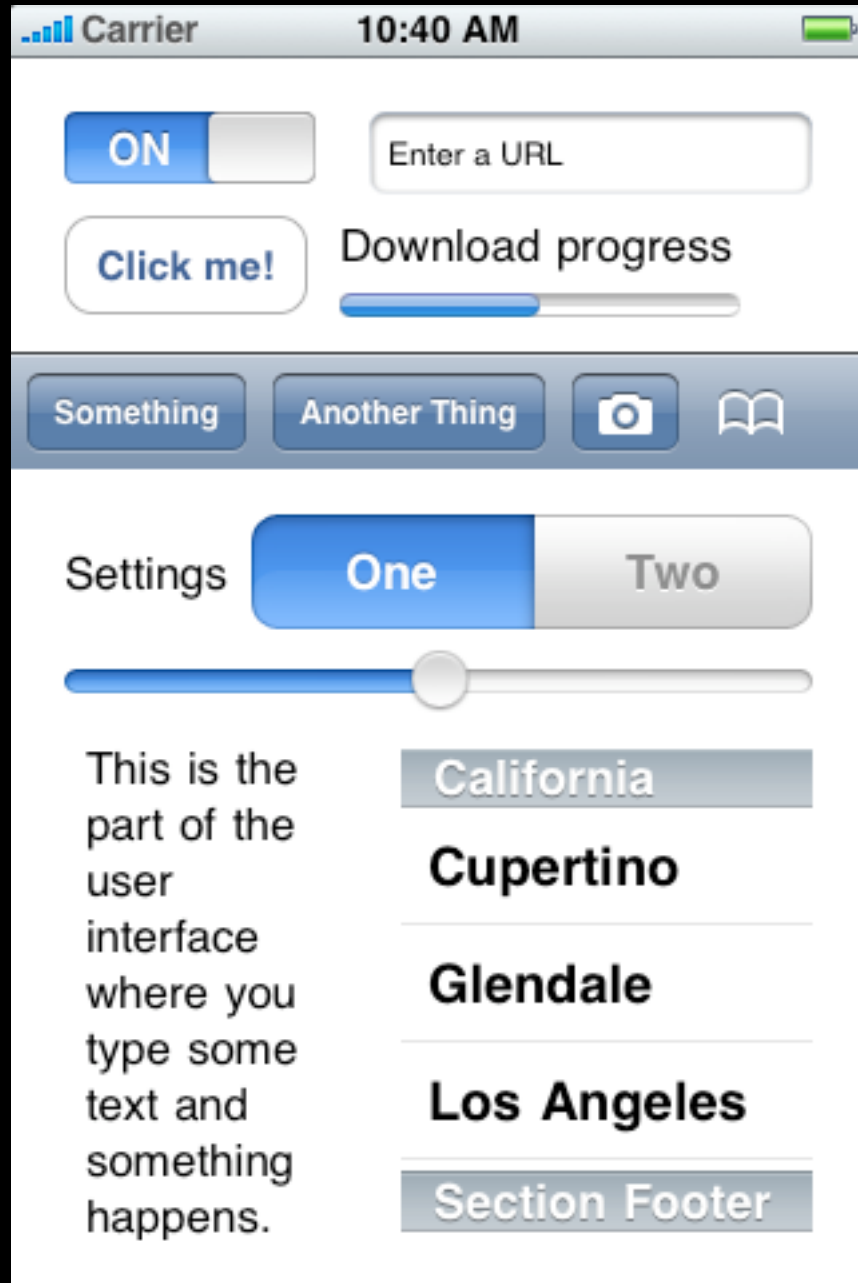
- Designing iPhone Applications
- Model-View-Controller (Why and How?)
- View Controllers
- Presence 1

# Designing iPhone Applications

# Two Flavors of Mail

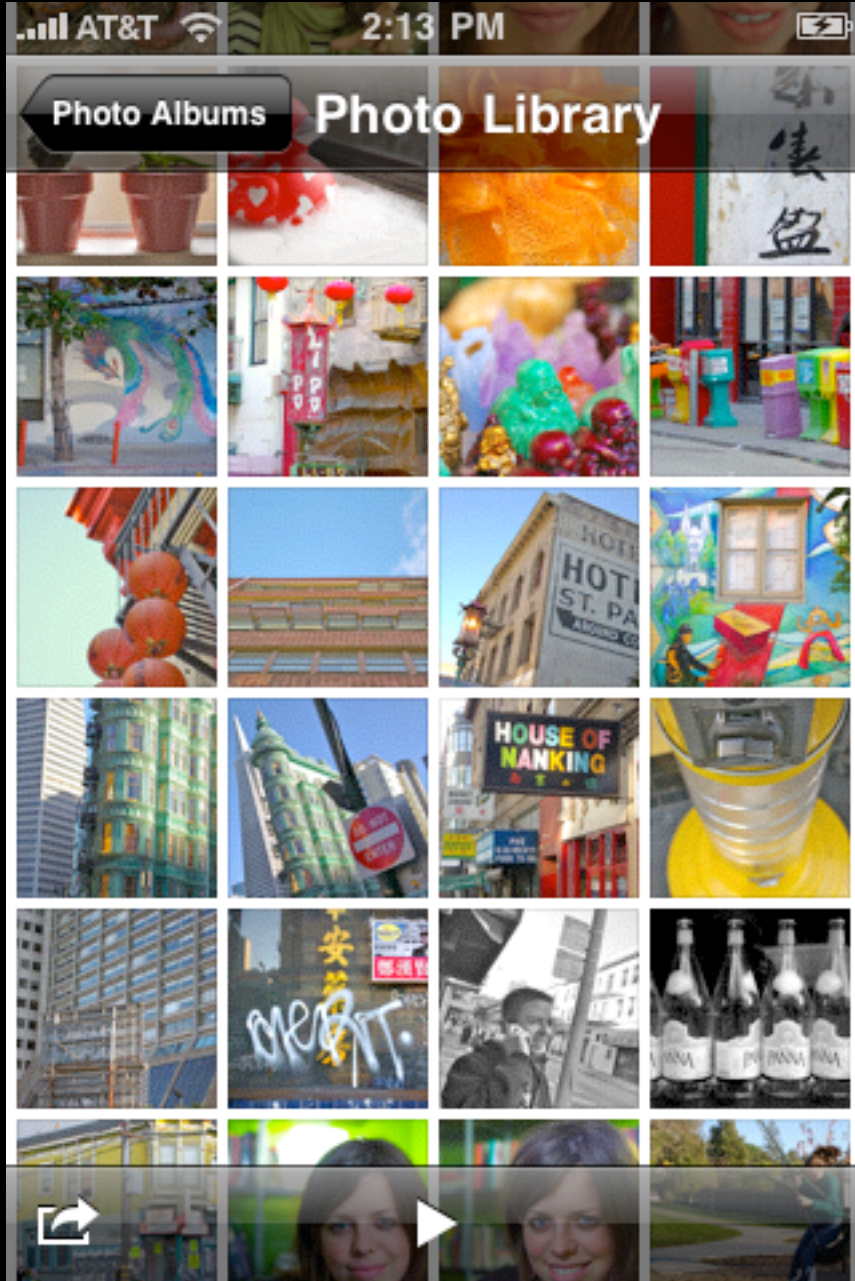


# Organizing Content





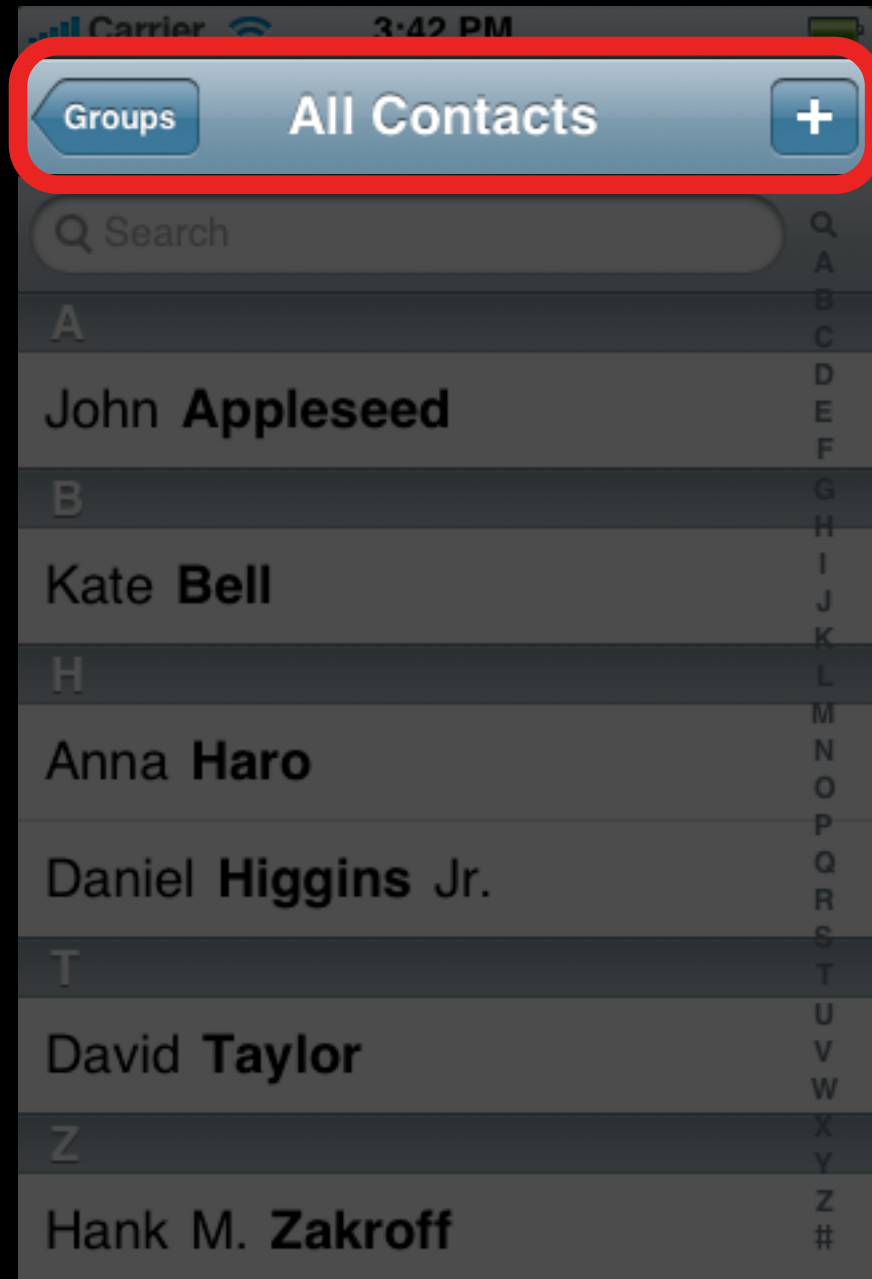
# Organizing Content



- Focus on your user's data
- One thing at a time
- Screenfuls of content

# Patterns for Organizing Content

Navigation Bar

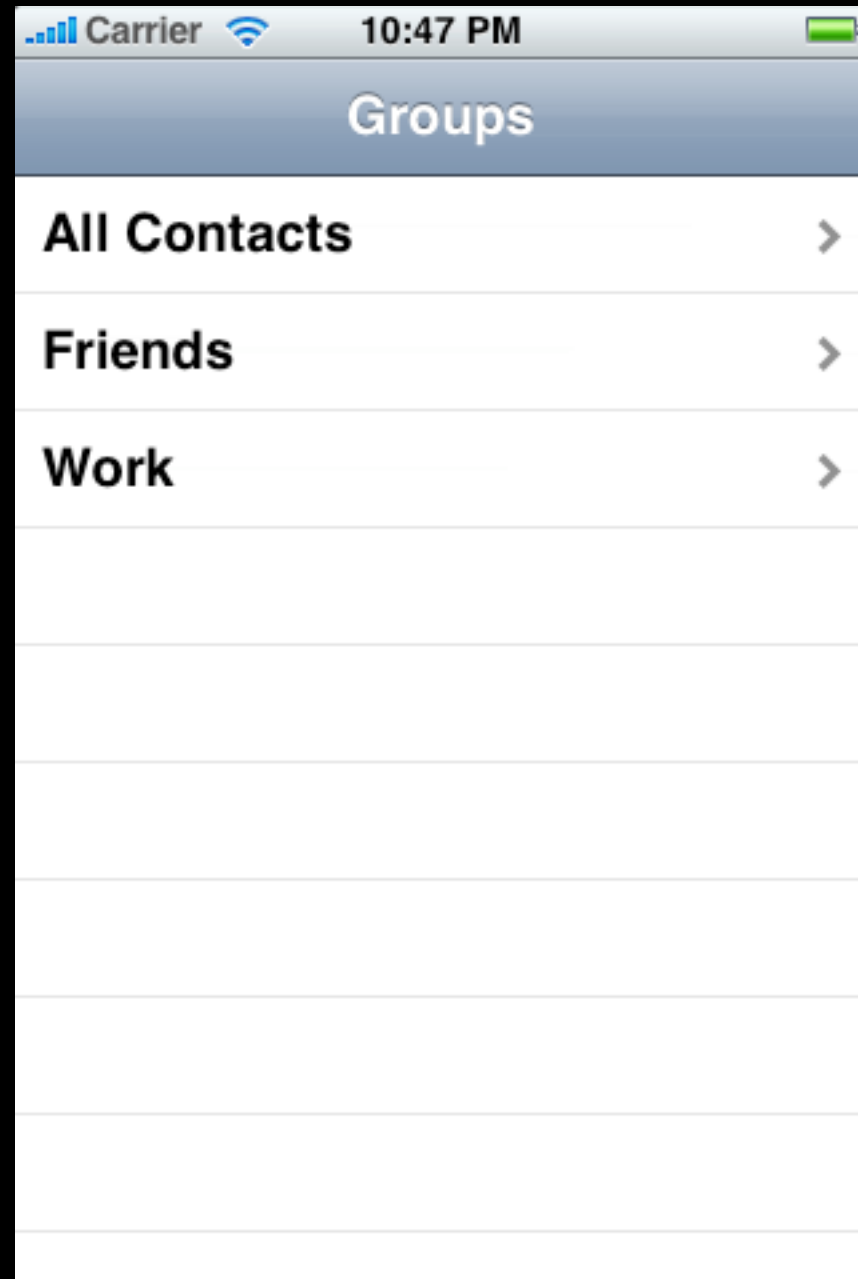


Tab Bar



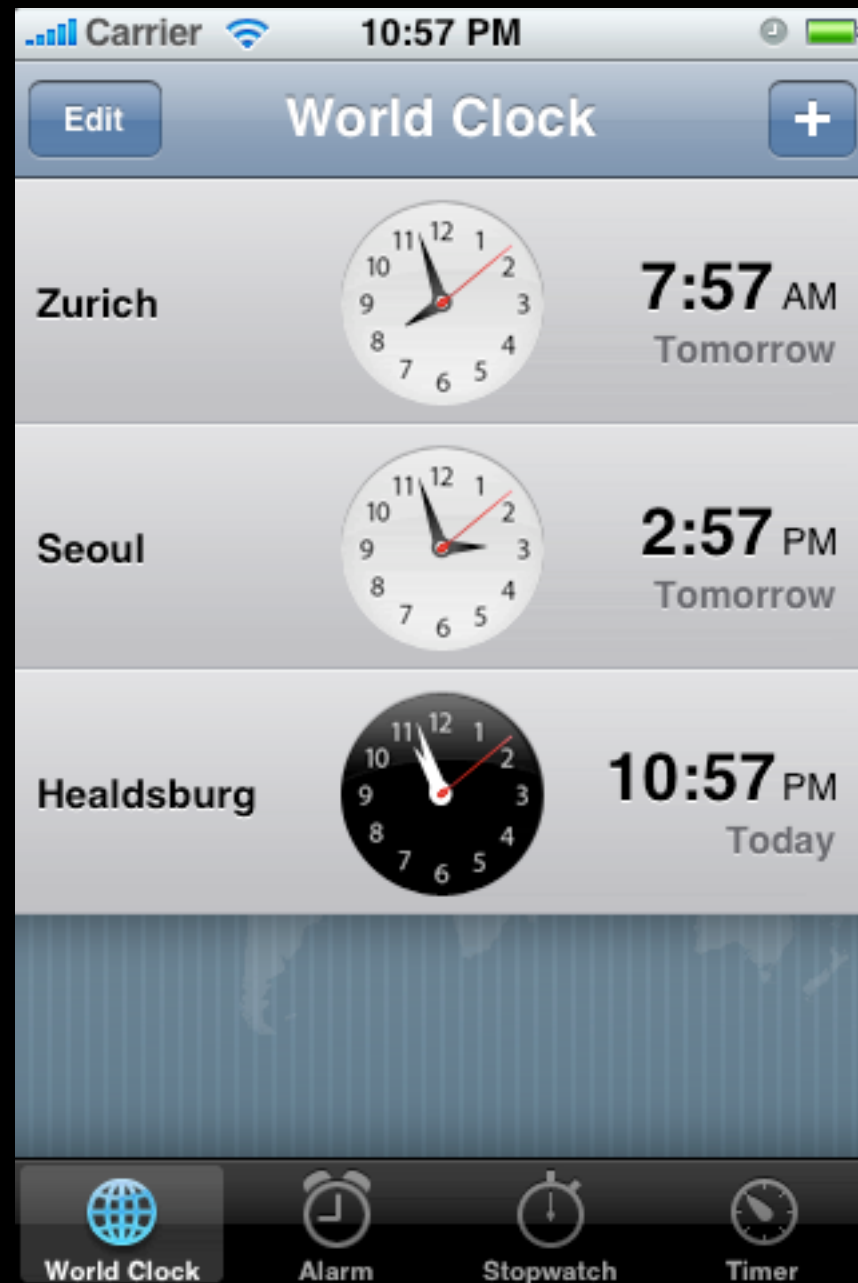
# Navigation Bar

- Hierarchy of content
- Drill down into greater detail



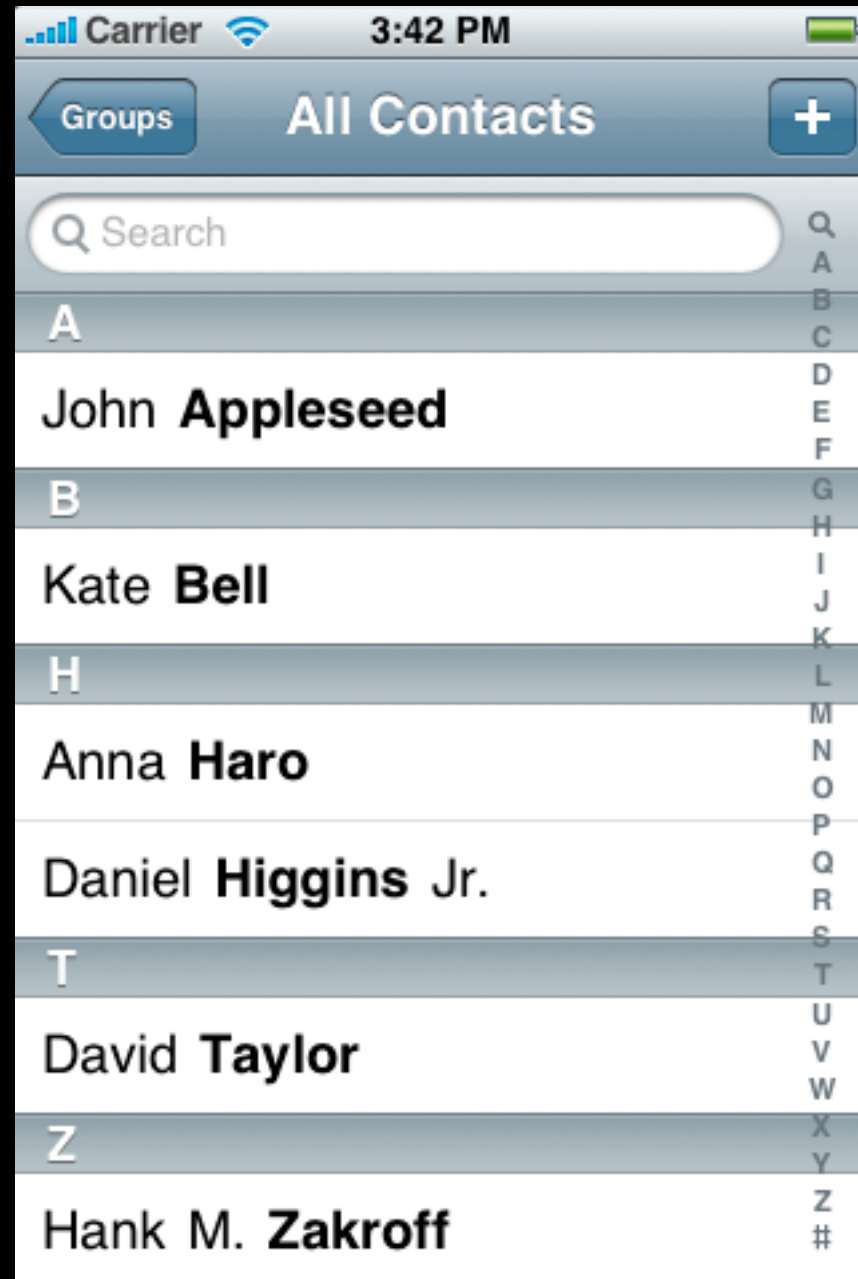
# Tab Bar

- Self-contained modes

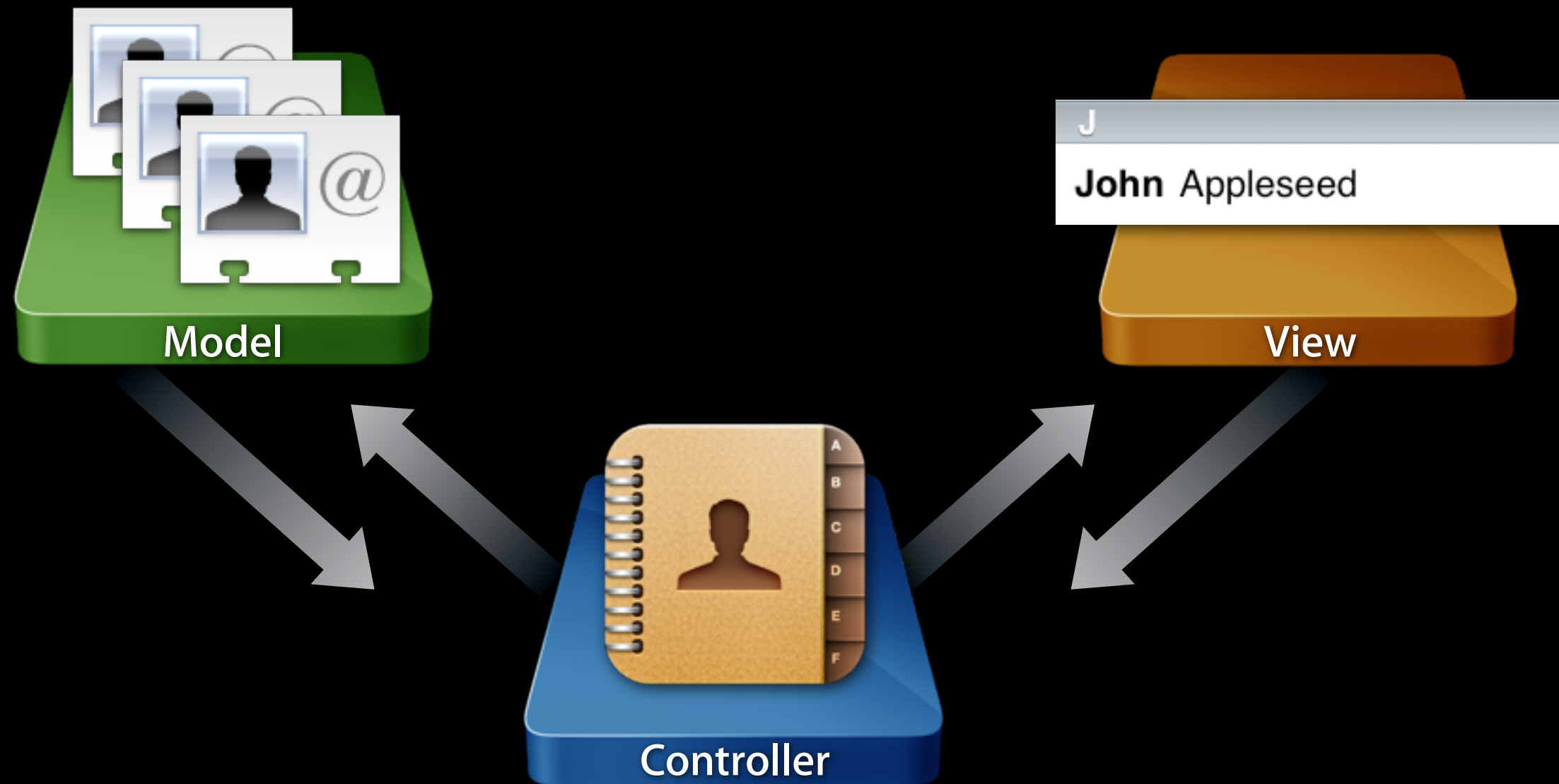


# A Screenful of Content

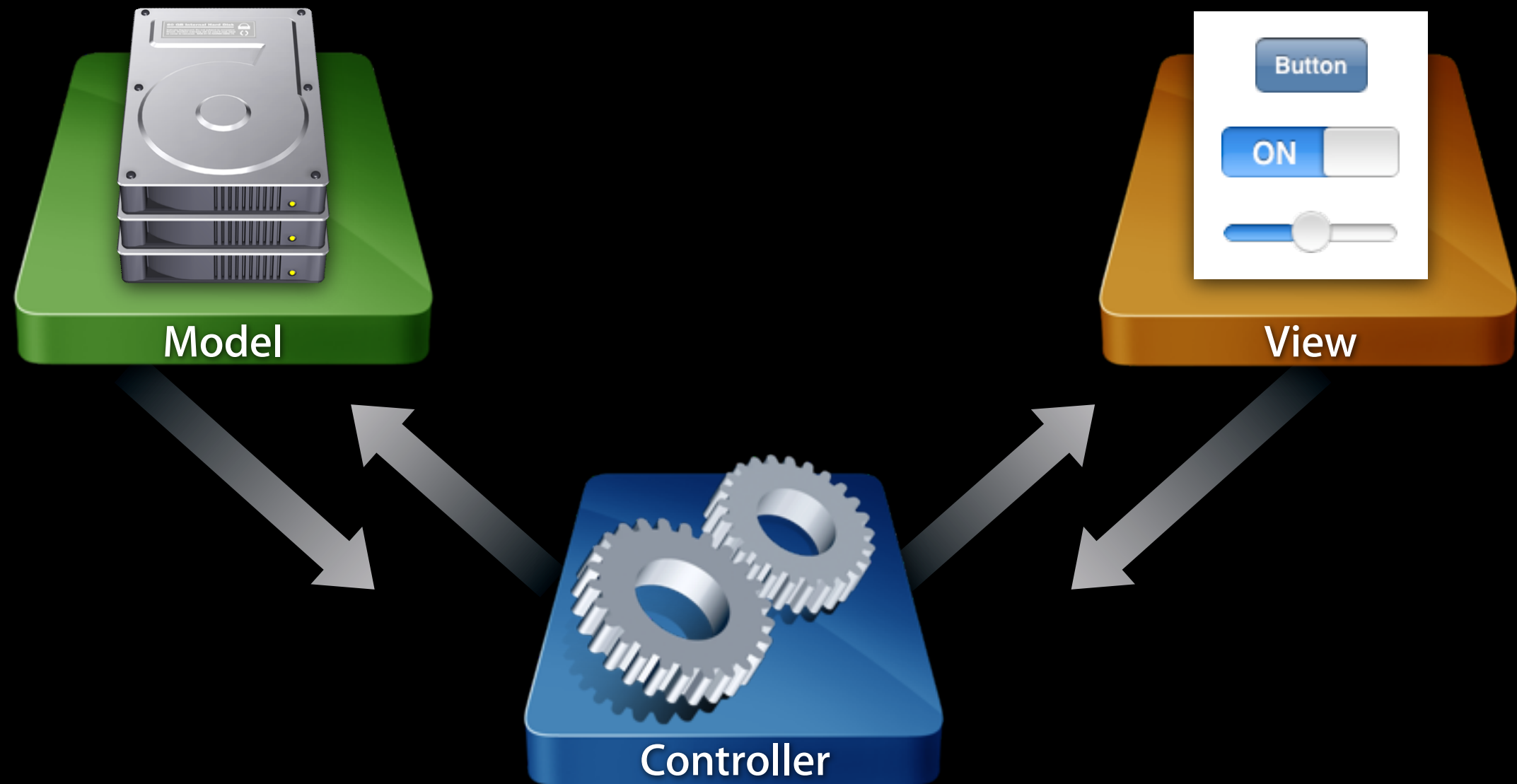
- Slice of your application
- Views, data, logic



# Parts of a Screenful



# Parts of a Screenful



# Model-View-Controller (Why and How?)



# Why Model-View-Controller?

- Ever used the word “spaghetti” to describe code?
- Clear responsibilities make things **easier to maintain**
- Avoid having one monster class that does everything



# Why Model-View-Controller?

- Separating responsibilities also leads to **reusability**
- By minimizing dependencies, you can take a model or view class you've already written and use it elsewhere
- Think of ways to **write less code**

# Communication and MVC

- How should objects communicate?
- Which objects know about one another?

## Model

- Example: **Polygon class**
- Not aware of views or controllers
- Typically the **most reusable**
- Communicate generically using...
  - Key-value observing
  - Notifications

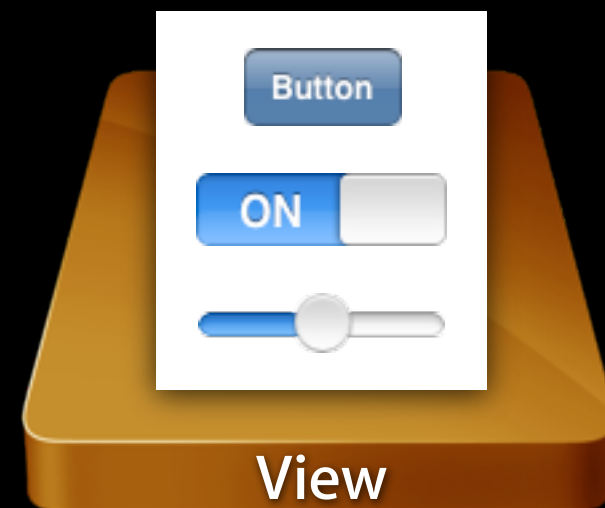


# Communication and MVC

- How should objects communicate?
- Which objects know about one another?

## View

- Example: **PolygonView** class
- Not aware of controllers, may be aware of relevant model objects
- Also **tends to be reusable**
- Communicate with controller using...
  - Target-action
  - Delegation



# Communication and MVC

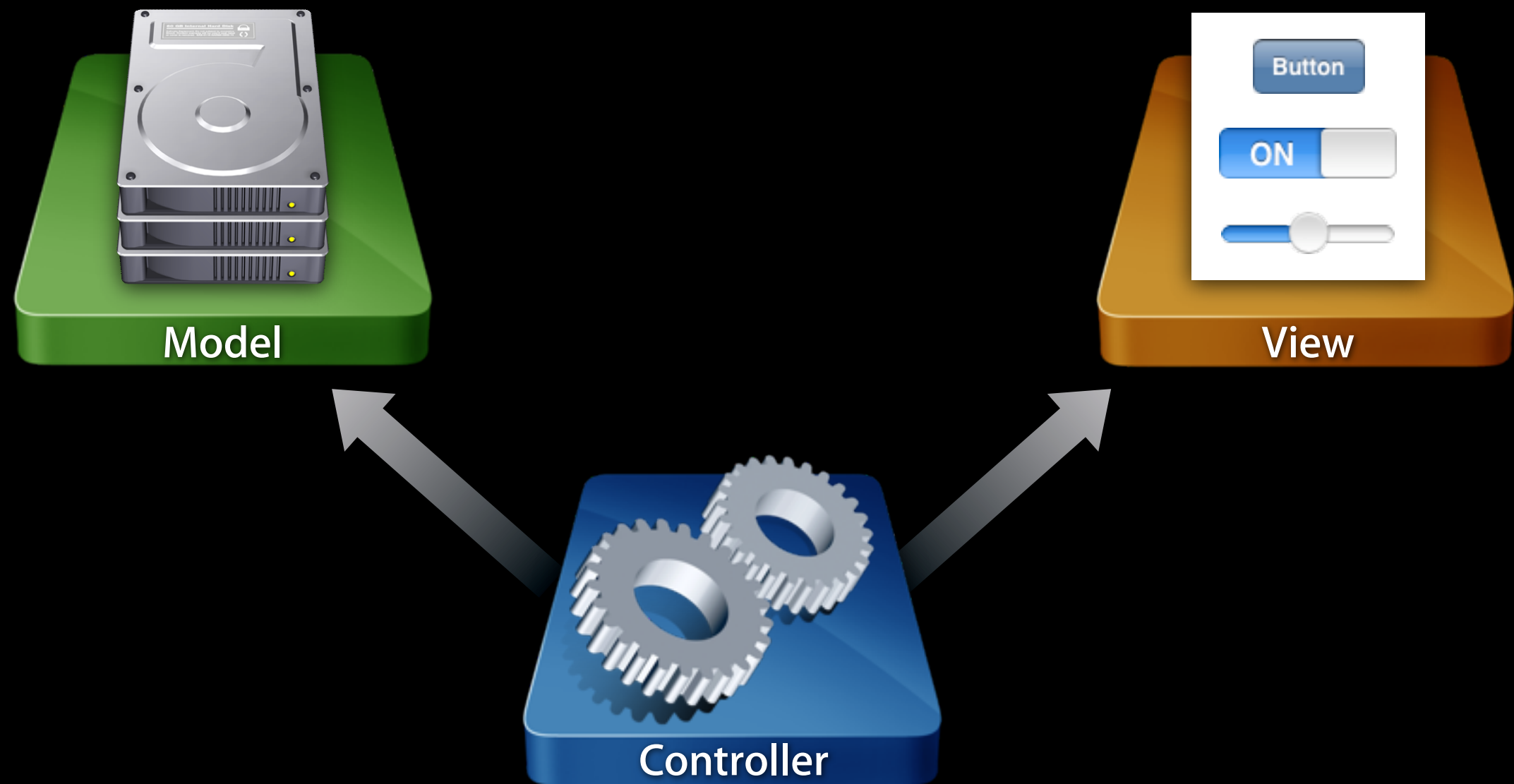
- How should objects communicate?
- Which objects know about one another?

## Controller

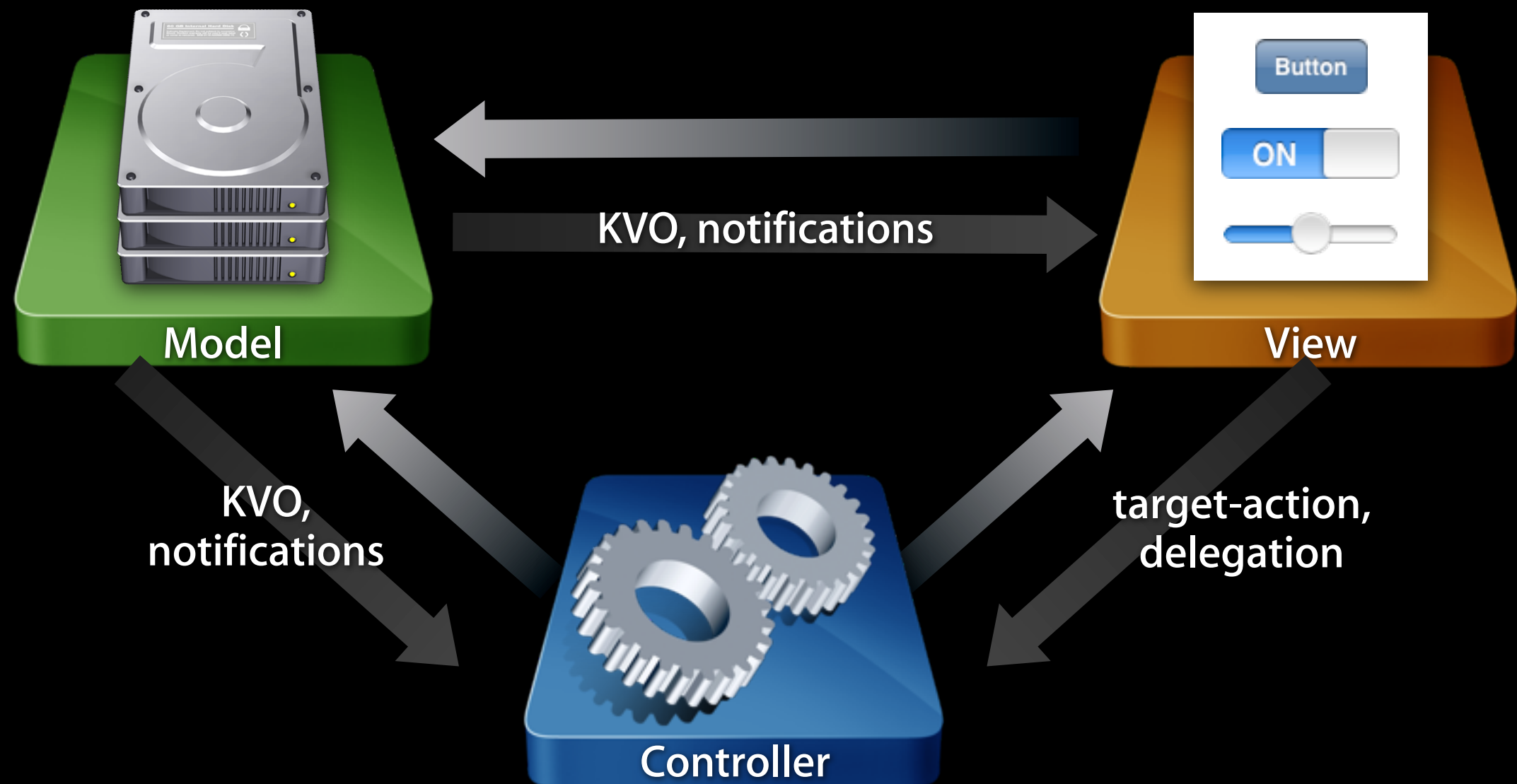
- Knows about model and view objects
- The brains of the operation
- Manages relationships and data flow
- Typically app-specific,  
so **rarely reusable**



# Communication and MVC



# Communication and MVC



# View Controllers



# Problem: Managing a Screenful

- Controller manages views, data and application logic
- Apps are made up of many of these
- Would be nice to have a well-defined starting point
  - A la UIView for views
  - Common language for talking about controllers

# Problem: Building Typical Apps

- Some application flows are very common
  - Navigation-based
  - Tab bar-based
  - Combine the two
- Don't reinvent the wheel
- Plug individual screens together to build an app

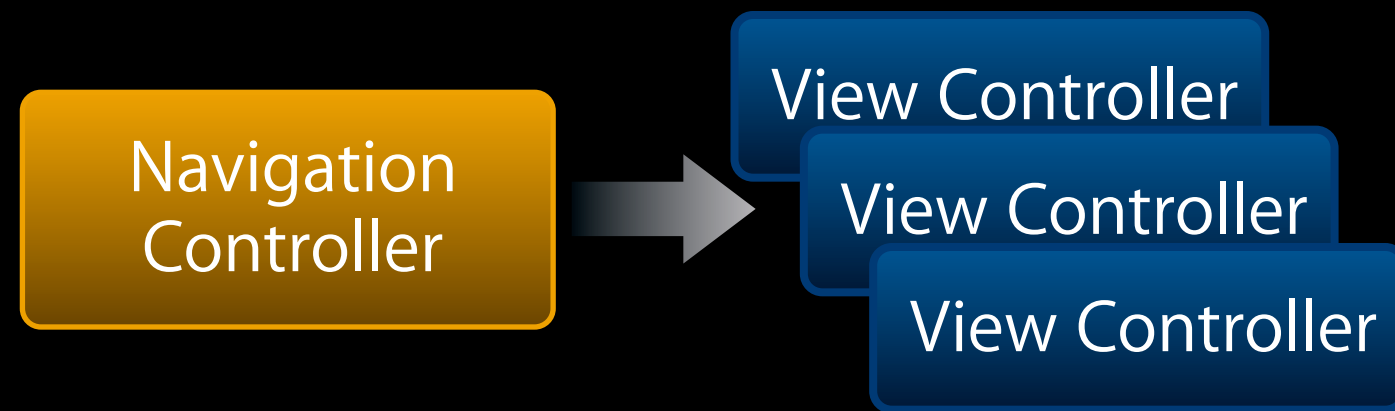
# UIViewController

- Basic building block
- Manages a screenful of content
- Subclass to add your application logic



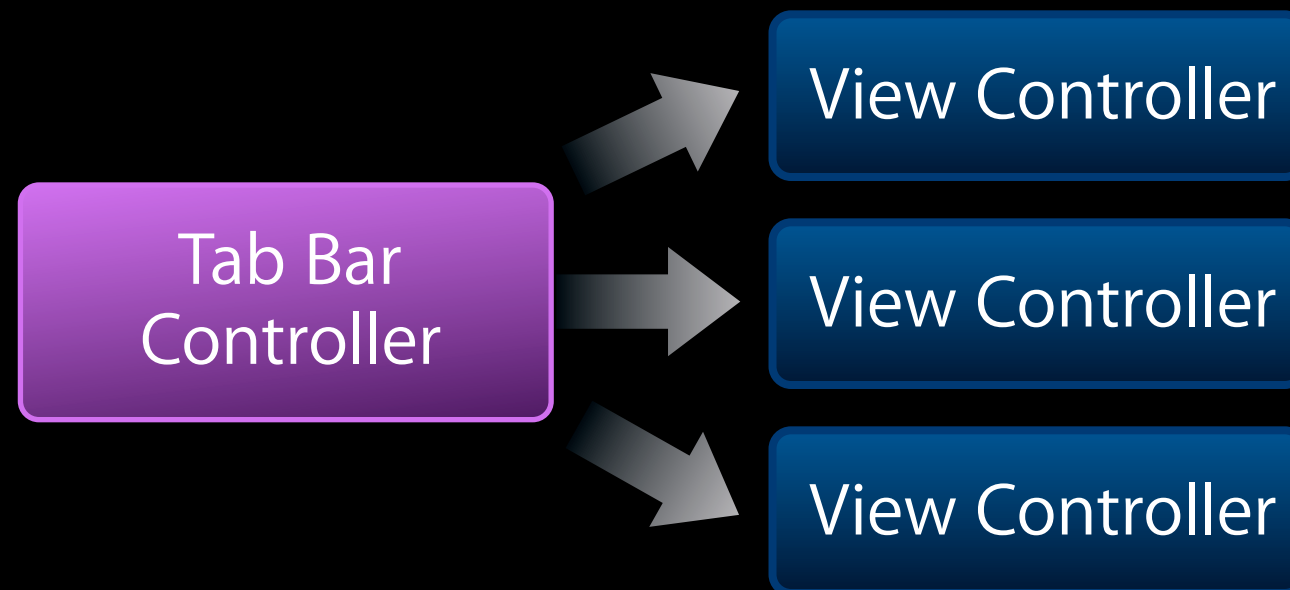
# “Your” and “Our” View Controllers

- **Create your own** UIViewController subclass for each screenful
- Plug them together using existing **composite** view controllers



# “Your” and “Our” View Controllers

- **Create your own** UIViewController subclass for each screenful
- Plug them together using existing **composite** view controllers



# Your View Controller Subclass

```
#import <UIKit/UIKit.h>
```

```
@interface MyViewController : UIViewController {  
    // A view controller will usually  
    // manage views and data  
    NSMutableArray *myData;  
    UILabel *myLabel;  
}
```

```
// Expose some of its contents to clients
```

```
@property (readonly) NSArray *myData;
```

```
// And respond to actions
```

```
- (void)doSomeAction:(id)sender;
```

# The “View” in “View Controller”

- UIViewController superclass has a view property
  - @property (retain) UIView \*view;
- Loads lazily
  - On demand when requested
  - Can be purged on demand as well (low memory)
- Sizing and positioning the view?
  - Depends on where it's being used
  - Don't make assumptions, be flexible

# When to call -loadView?

- **Don't do it!**
- Cocoa tends to embrace a lazy philosophy
  - Call -release instead of -dealloc
  - Call -setNeedsDisplay instead of -drawRect:
- Allows work to be deferred or coalesced
  - Performance!



# Creating Your View in Code

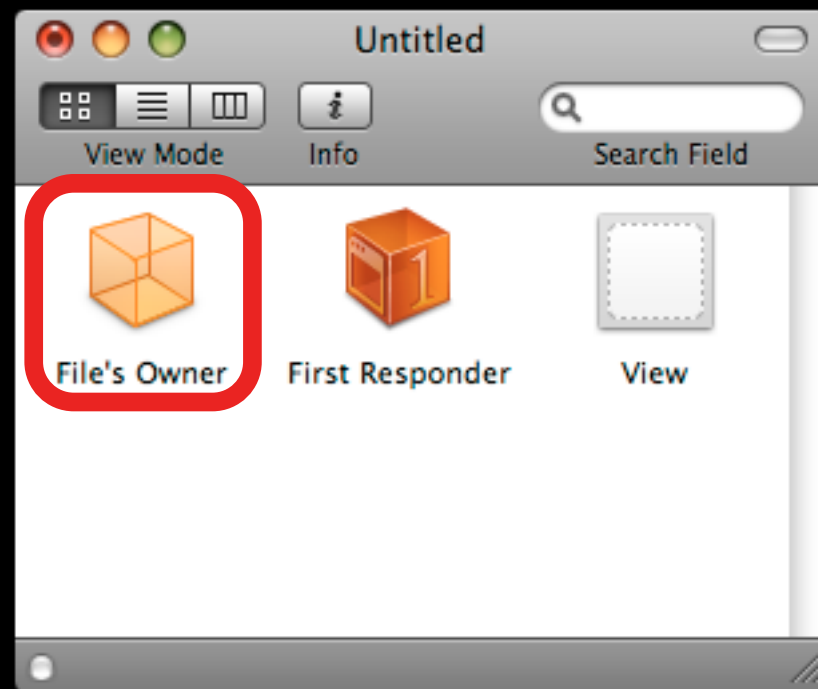
- Override -loadView
  - Never call this directly
- Create your views
- Set the view property
- Create view controller with -init



```
// Subclass of UIViewController
- (void)loadView
{
    MyView *myView = [[MyView alloc] initWithFrame:frame];
    self.view = myView; // The view controller now owns the view
    [myView release];
}
```

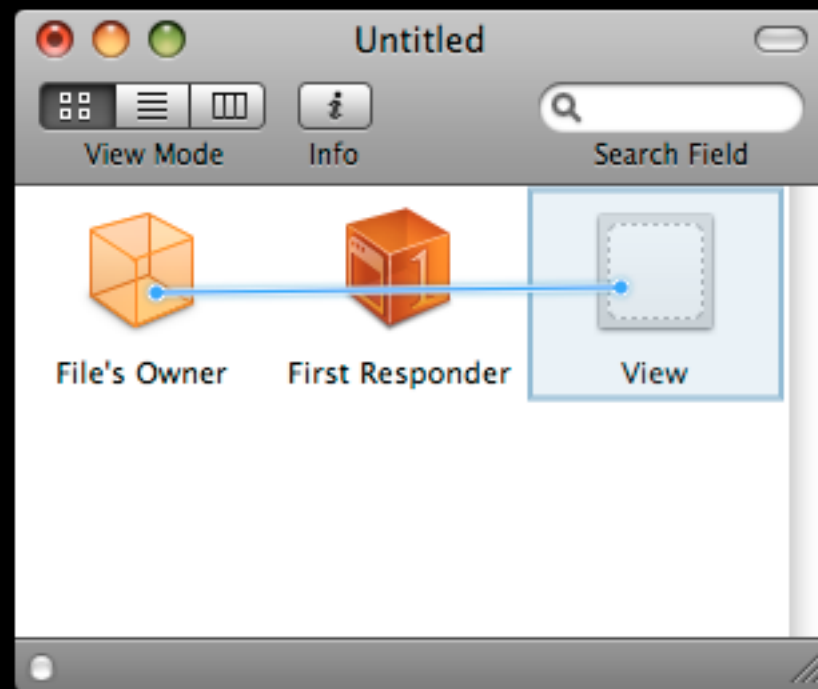
# Creating Your View with Interface Builder

- Lay out a view in Interface Builder
- File's owner is view controller class



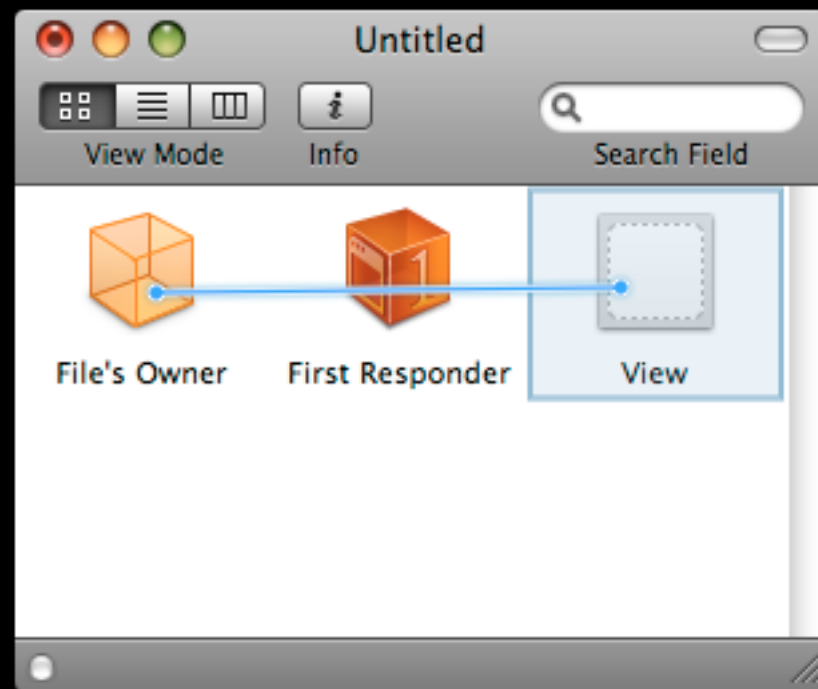
# Creating Your View with Interface Builder

- Lay out a view in Interface Builder
- File's owner is view controller class
- Hook up view outlet



# Creating Your View with Interface Builder

- Lay out a view in Interface Builder
- File's owner is view controller class
- Hook up view outlet
- Create view controller with `-initWithNibName:bundle:`



# Demo:

## View Controllers with IB

# View Controller Lifecycle

```
- (id)initWithNibName:(NSString *)nibName  
bundle:(NSBundle *)bundle  
{  
    if (self == [super init...]) {  
        // Perform initial setup, nothing view-related  
        myData = [[NSMutableArray alloc] init];  
        self.title = @"Foo";  
    }  
    return self;  
}
```

# View Controller Lifecycle

```
- (void)viewDidLoad  
{  
    // Your view has been loaded  
    // Customize it here if needed  
    view.someWeirdProperty = YES;  
}
```

# View Controller Lifecycle

```
- (void)viewWillAppear:(BOOL)animated  
{  
    [super viewWillAppear:animated];  
  
    // Your view is about to show on the screen  
    [self beginLoadingDataFromTheWeb];  
    [self startShowingLoadingProgress];  
}
```



# View Controller Lifecycle

```
- (void)viewWillDisappear:(BOOL)animated  
{  
    [super viewWillDisappear:animated];  
  
    // Your view is about to leave the screen  
    [self rememberScrollPosition];  
    [self saveDataToDisk];  
}
```

# Loading & Saving Data

- Lots of options out there, depends on what you need
  - UserDefaults
  - Property lists
  - SQLite
  - Web services
- Covering in greater depth in Lecture 9 on 4/29

# Demo:

# Loading & Saving Data

# More View Controller Hooks

- Automatically rotating your user interface
- Low memory warnings

# Supporting Interface Rotation

```
- (BOOL)shouldAutorotateToInterfaceOrientation:  
    (UIInterfaceOrientation)interfaceOrientation  
{  
    // This view controller only supports portrait  
    return (interfaceOrientation ==  
            UIInterfaceOrientationPortrait);  
}
```

# Supporting Interface Rotation

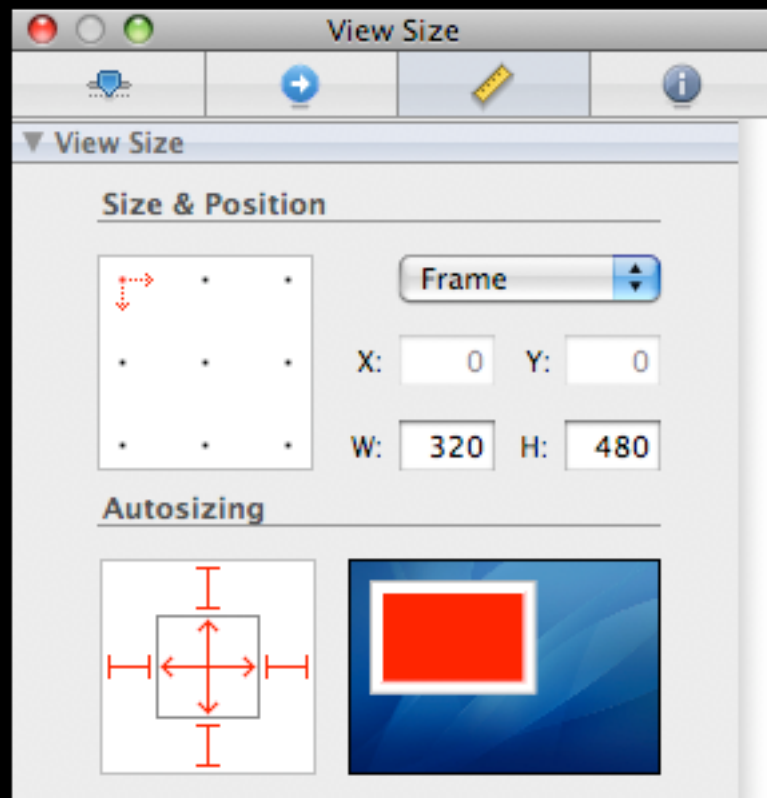
```
- (BOOL)shouldAutorotateToInterfaceOrientation:  
    (UIInterfaceOrientation)interfaceOrientation  
{  
    // This view controller supports all orientations  
    // except for upside-down.  
    return (interfaceOrientation !=  
            UIInterfaceOrientationPortraitUpsideDown);  
}
```

# Demo:

# Rotating Your Interface

# Autoresizing Your Views

```
view.autoresizingMask = UIViewAutoresizingFlexibleWidth |  
                        UIViewAutoresizingFlexibleHeight;
```

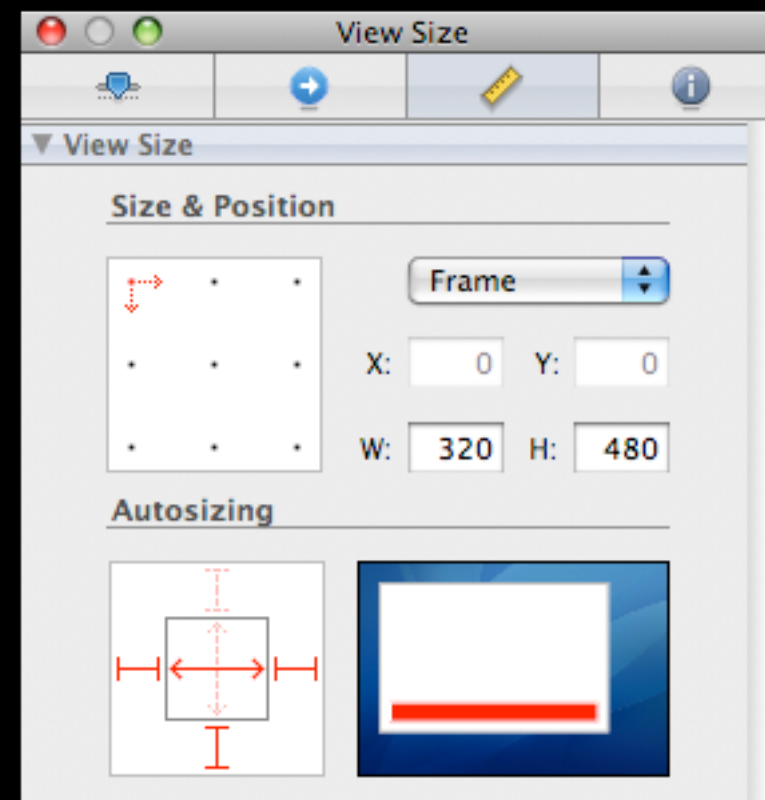
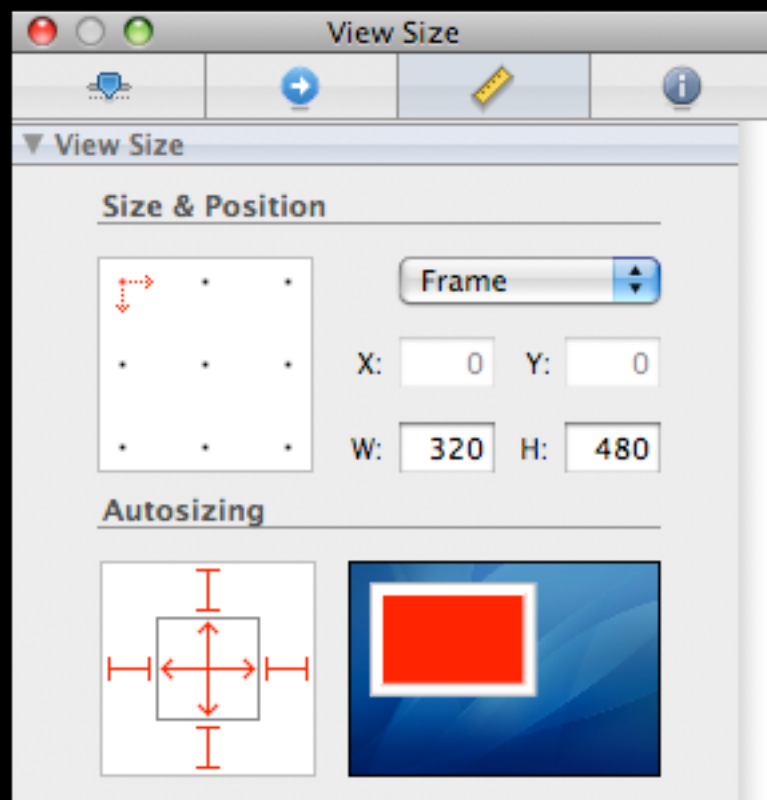




# Autoresizing Your Views

```
view.autoresizingMask = UIViewAutoresizingFlexibleWidth |  
                        UIViewAutoresizingFlexibleHeight;
```

```
view.autoresizingMask = UIViewAutoresizingFlexibleWidth |  
                        UIViewAutoresizingFlexibleTopMargin;
```



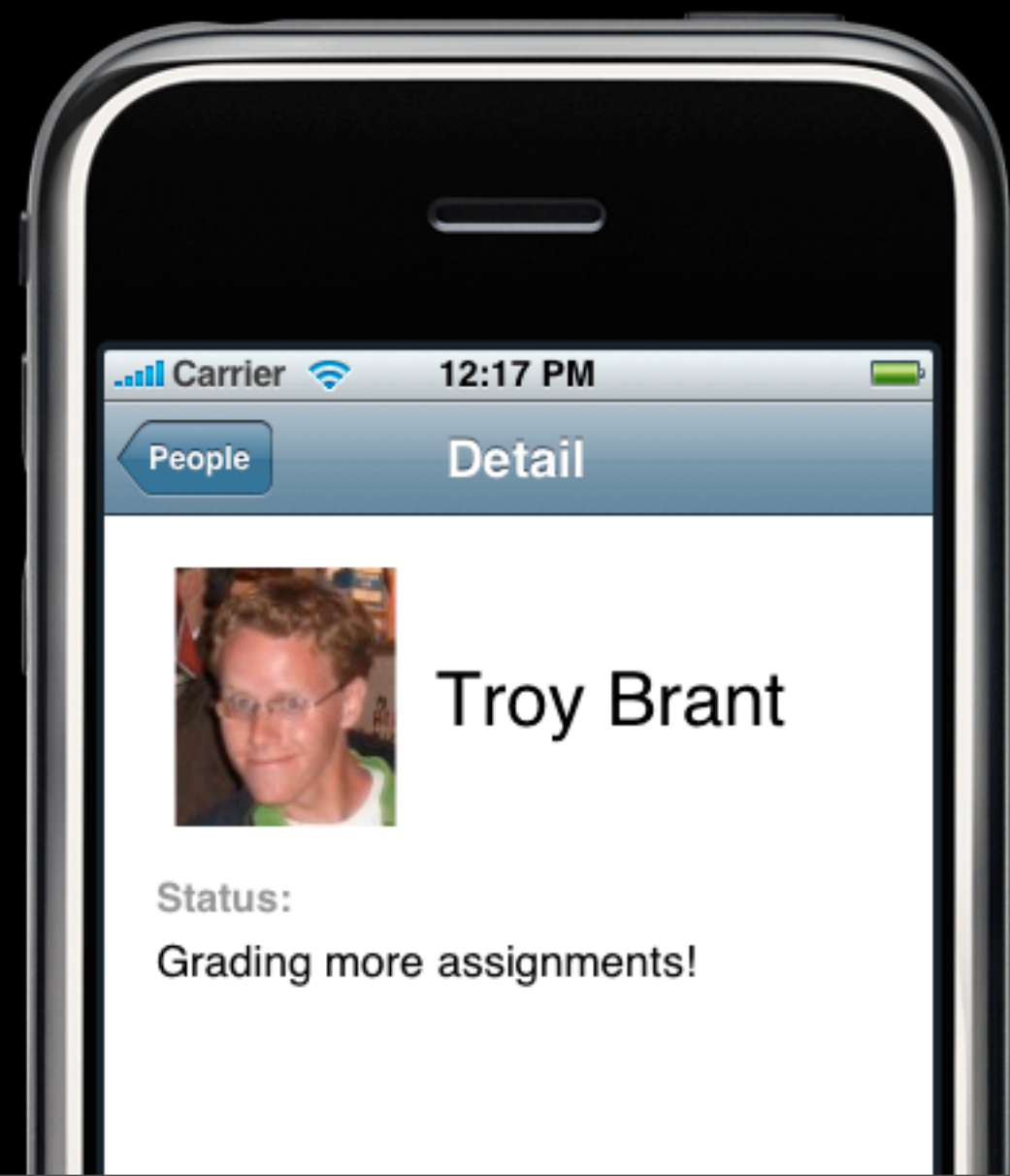
# Presence

# Presence

- Building an iPhone app for viewing online status updates
  - “What are you doing right now?”
- Our assignments will be using Twitter API
  - Could extend to Facebook updates, IM status, RSS feeds...
- **Four parts**, each week builds on the previous one
  - Part 1: Using view controllers & navigation
  - Part 2: Managing and displaying real data
  - Part 3: Threading, text input, modal content
  - Part 4: Search, Address Book and more...

# Presence - Part 1

- Goals
  - Create your own view controller subclasses
  - Present a hierarchy using UINavigationController (next lecture)



# Demo: Presence 1

Questions?