iPhone Programming

CMSC 498i – Spring 2010

View Controllers (Part 2)

Lecture #10 – Chuck Pisula

Today's Topics

- More on View Controllers
- Modal Controllers
- Rotation, Editing, Customization…

Review

- Application Flow
  - `UITabBarController`
    - One per application
  - `UINavigationController`
    - One per application, or…
    - One per tab item
- Custom Logic
  - `UIViewController`, `UITableViewController`
    - One for each screen

Subclassing

Standard Overrides

Subclassing - Overrides

- `-viewWillAppear:`
  - Perform tasks associated with presenting your view
  - E.g. update a display value, coordinate status bar look, etc.
- `-viewDidAppear:`
  - Called when the view has been fully transitioned onto the screen
  - E.g. `UITableView` flashes its scroll bars

Subclassing - Overrides

- `-viewDidDisappear:`
  - Your view is dismissed, covered or hidden
  - Should stop timers, display updates, etc…
- `-viewWillDisappear:`
  - May want to stop timers and updates at this point
  - May want to save state to disk

Subclassing Overrides

- `-viewDidLoad`
  - Called after a view is loaded from XIB, or by `-loadView`
- `-viewDidUnload`
  - Called whenever UIKit deems it necessary to release the view without deallocating the controller
  - You can release any data easily recreated
  - If your data is expensive to recreate (often the case with caches), maybe wait until `-didReceiveMemoryWarning` to release

Laziness…

- Notice, view creation is very “lazy”
  - Requires some coordination because of laziness
  - Benefit is memory saving

```
- (UIView *)view {
   if (!view_) {
       // load the “view_”
       [self loadView];
   }
   return view_;
}
```

Laziness…

- Notice, view creation is very “lazy”
  - Requires some coordination because of laziness
  - Benefit is memory saving

```
- (NSMutableDictionary *)mapTable {
   if (!mapTable_) {
      mapTable_ = [[NSMutableDictionary alloc] init];
   }
   return mapTable_;
}
```

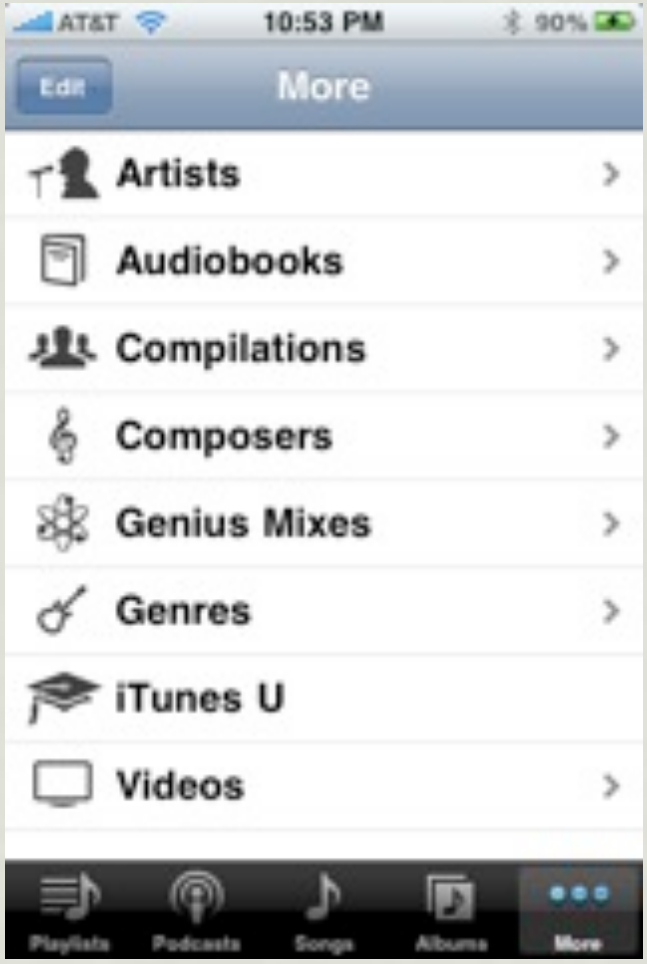
Lazy Loading

- Example – `UITabViewController`
  - tab view created and set up with list of view controllers
  - however, only one view exists
  - view is created on demand when switching to selected tab
- Example – `UINavigationController`
  - Only views on screen must be kept around
  - For good performance, visited items views are kept around
  - …Unless a “memory notification” comes in…

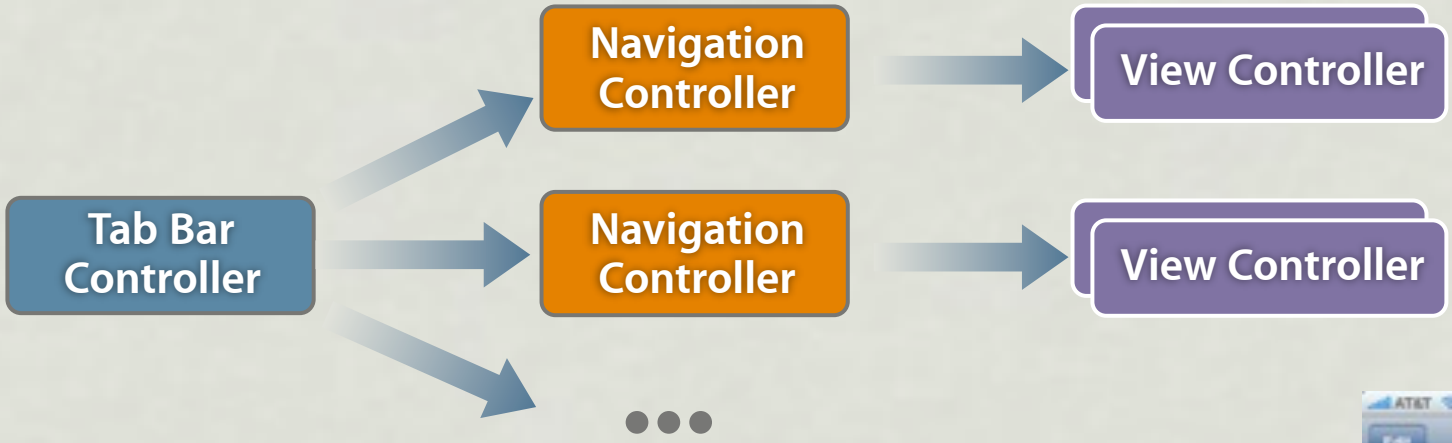
Demo

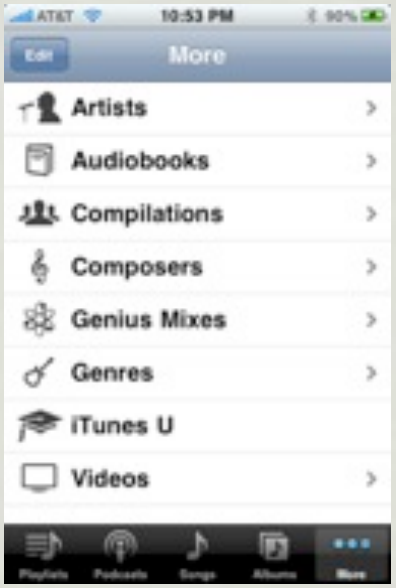
View Controllers

Example Application Flow



Example Application Flow



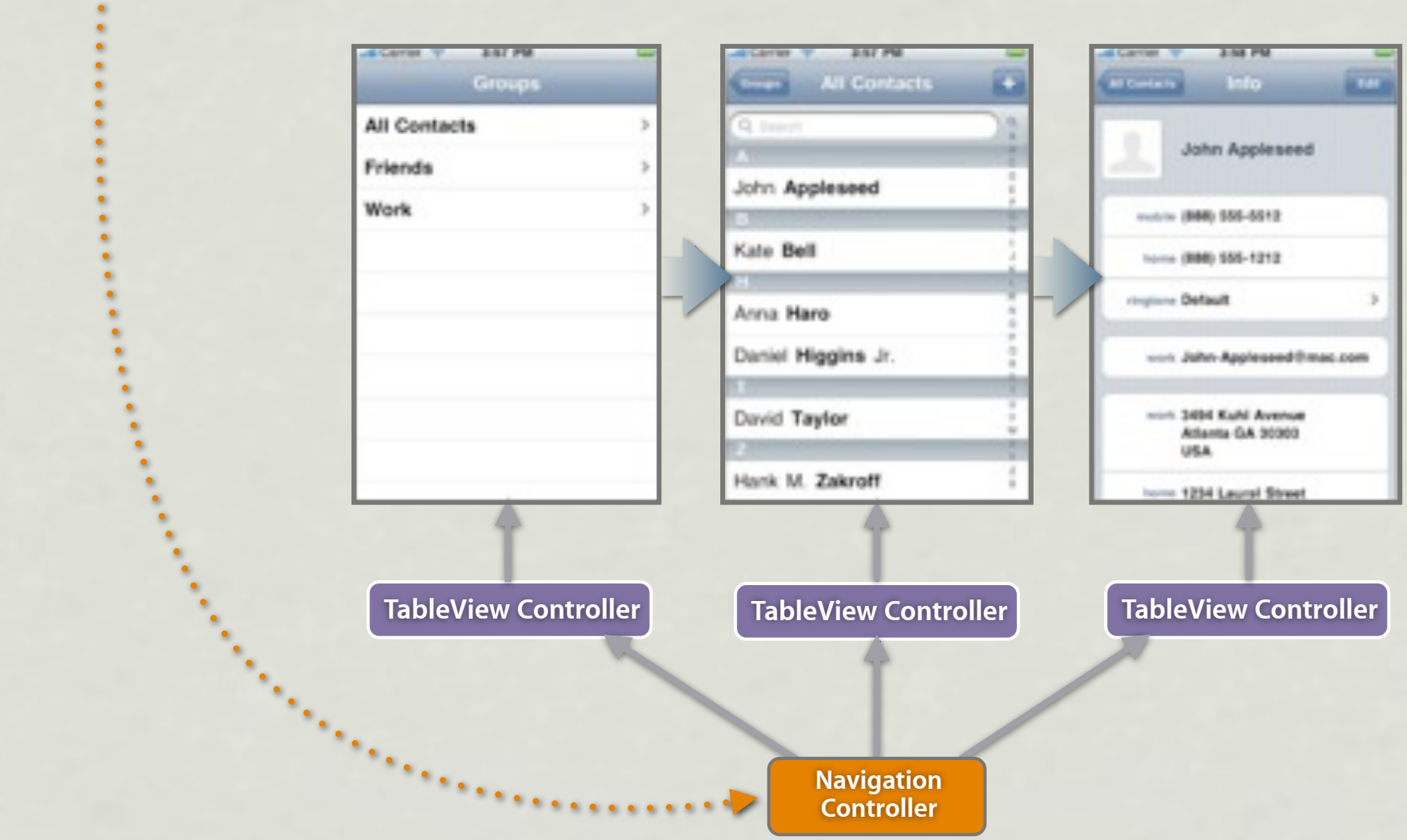


Example Flow – Terms

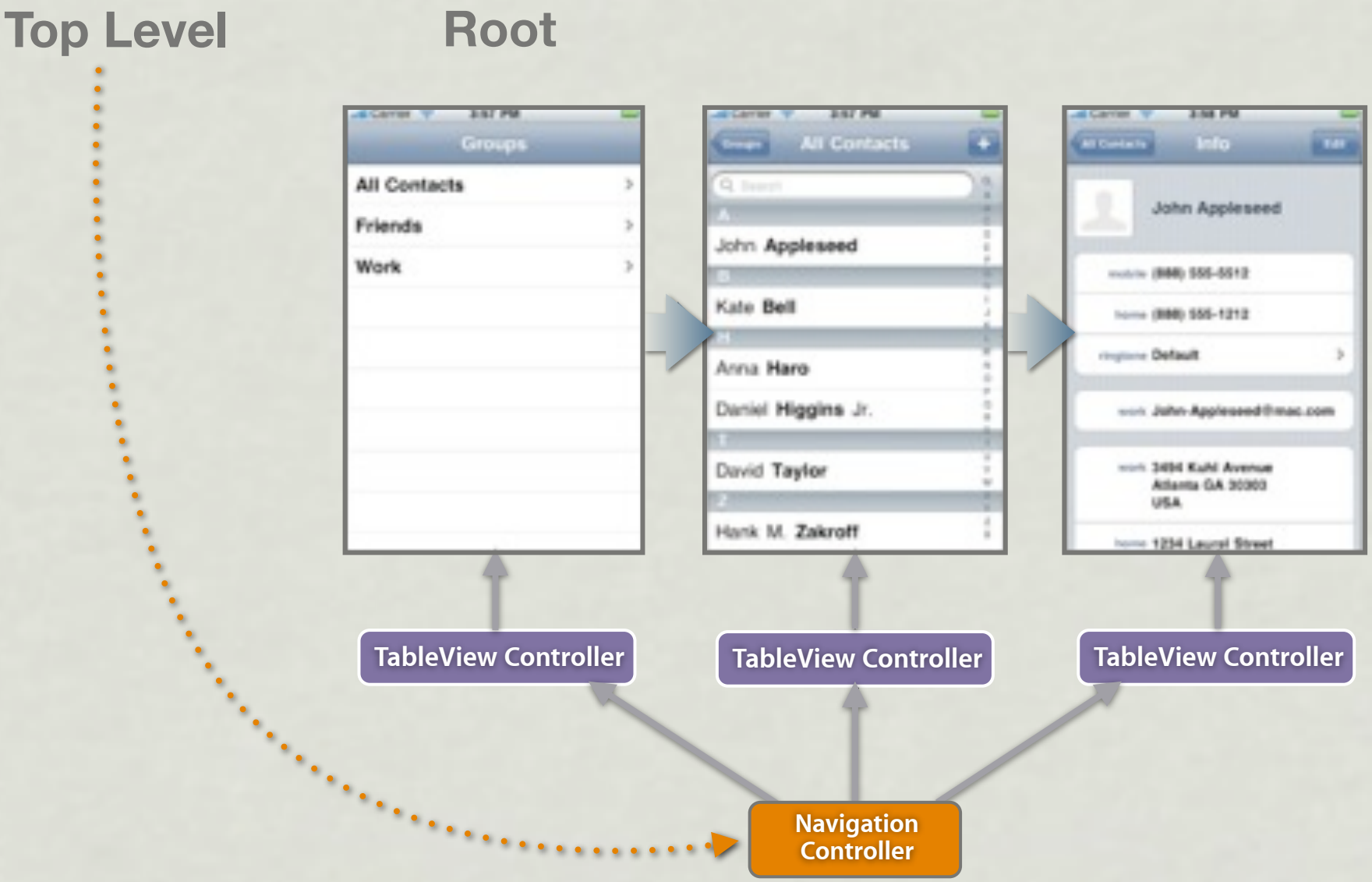
Example Flow – Terms

Example Flow – Terms

Top Level



Example Flow – Terms



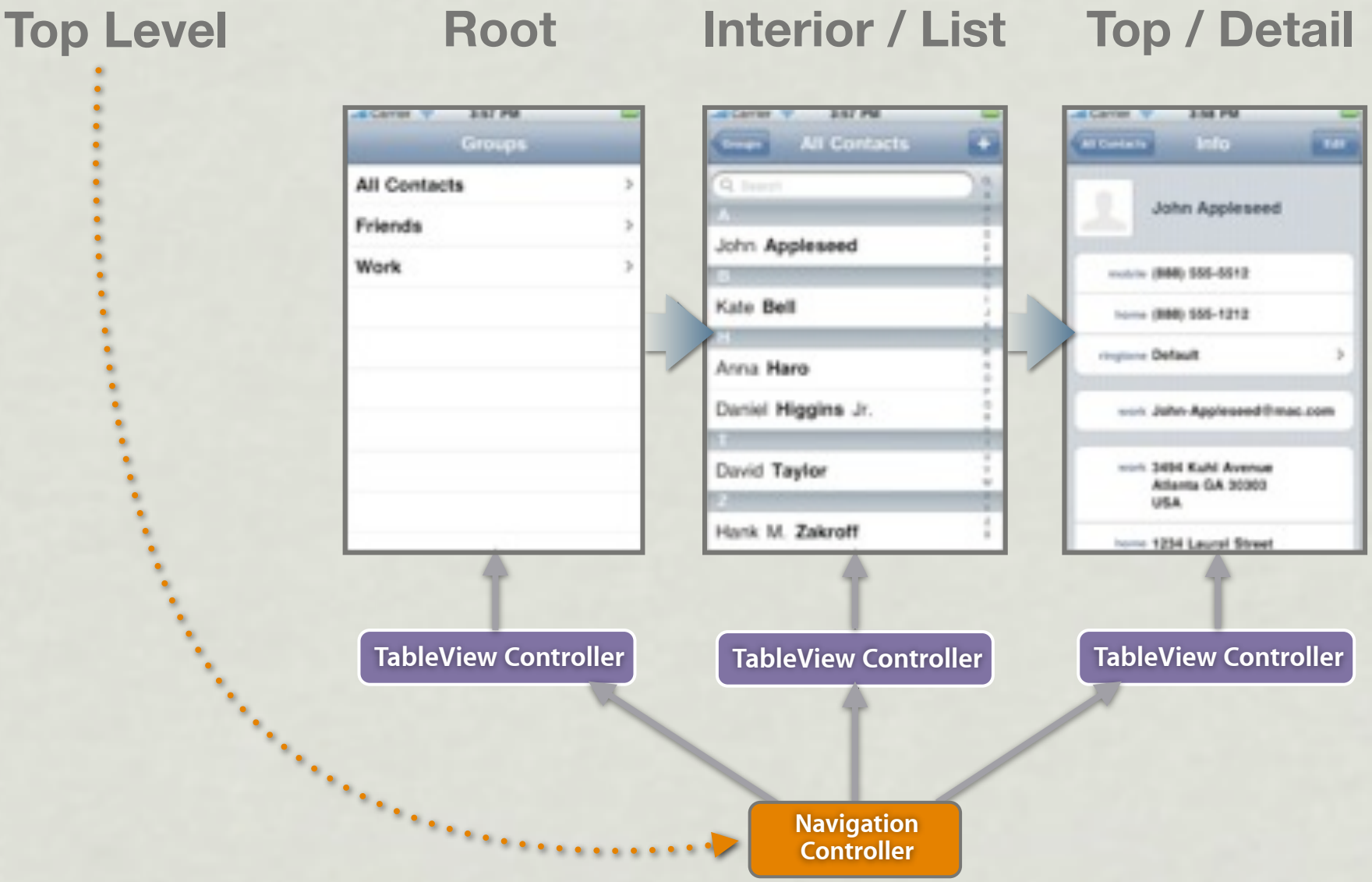
Example Flow – Terms

Top Level

Root

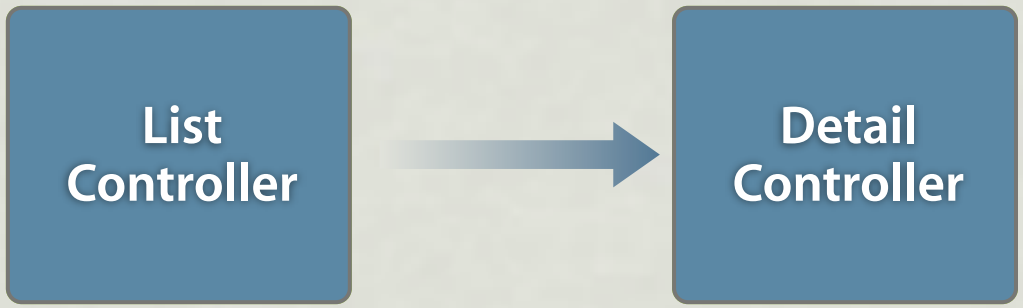
Interior / List

Example Flow – Terms



Controller For Each Screen

- Manage “screenful of content” *
- Small targeted controllers are easy to write, maintain



* Evan Doll, 2008-ish

Typical Navigation

- “Root” view controller
  - Create in Application Delegate
  - Typically one of either…
    - `UITabBarController`
    - `UITabBarController` of `UINavigationControllers`
- Navigation controller interior items
  - Respond to events and create new view controllers
  - Push onto navigation stack, or display modally

Tab As Root View

- Setup in Application Delegate

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {









}
```

Tab As Root View

- Setup in Application Delegate

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Create top level tab bar controller
    tabBarController = [[UITabBarController alloc] init];









}
```

Tab As Root View

- Setup in Application Delegate

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Create top level tab bar controller
    tabBarController = [[UITabBarController alloc] init];


    // Create each individual tab item's controller
    // Initialized with a root controller
    rootCtl1 = [[[SomeTableViewController alloc] init...] autorelease];
    navCtl1 = [[[UINavigationController alloc] initWithRootViewController:rvc1] autorelease]

    // Give the individual tab item's to the tab controller
    NSArray *subitems = [NSArray arrayWithObjects:navController1, navController2, nil];
    [tabBarController setViewControllers: subitems];









}
```

Tab As Root View

- Setup in Application Delegate

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Create top level tab bar controller
    tabBarController = [[UITabBarController alloc] init];


    // Create each individual tab item's controller
    // Initialized with a root controller
    rootCtl1 = [[[SomeTableViewController alloc] init...] autorelease];
    navCtl1 = [[[UINavigationController alloc] initWithRootViewController:rvc1] autorelease]

    // Give the individual tab item's to the tab controller
    NSArray *subitems = [NSArray arrayWithObjects:navController1, navController2, nil];
    [tabBarController setViewControllers: subitems];



    // Add the tab controller's view
    // This will cause -loadView to execute, or the XIB to load
    [window addSubview:tabBarController.view];

    // Put it all on screen
    [window makeKeyAndVisible];
}
```

Drilling Into

- Respond to an action, or table selection

```
- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath {




}
```

Drilling Into

- Respond to an action, or table selection

```
- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

    // Find info to display in the detail view
    // We will pass this data to the detail view controller
    id info = [self infoForRowAtIndexPath:indexPath];
    if (info) {









    }
}
```

Drilling Into

- Respond to an action, or table selection

```
- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

    // Find info to display in the detail view
    // We will pass this data to the detail view controller
    id info = [self infoForRowAtIndexPath:indexPath];
    if (info) {

       // Create a DetailViewController
       DetailViewController *dvc = [[DetailViewController alloc] init];

       // Pass it the info it needs
       [dvc setDetailInfo:info];

       // Animate it in the current navigation stack
       [[self navigationController] pushViewController:dvc animated:YES];

       // The navigation controller owns the DetailViewController
       // as a result of it being on the current stack
       [dvc release];

    }
}
```

Navigating Back

- Back button – leave it up to the user
- Programmatically

```
[navController popViewControllerAnimated:YES]
```

Data Flow

- Determine what data needs to be communicated

Data Flow

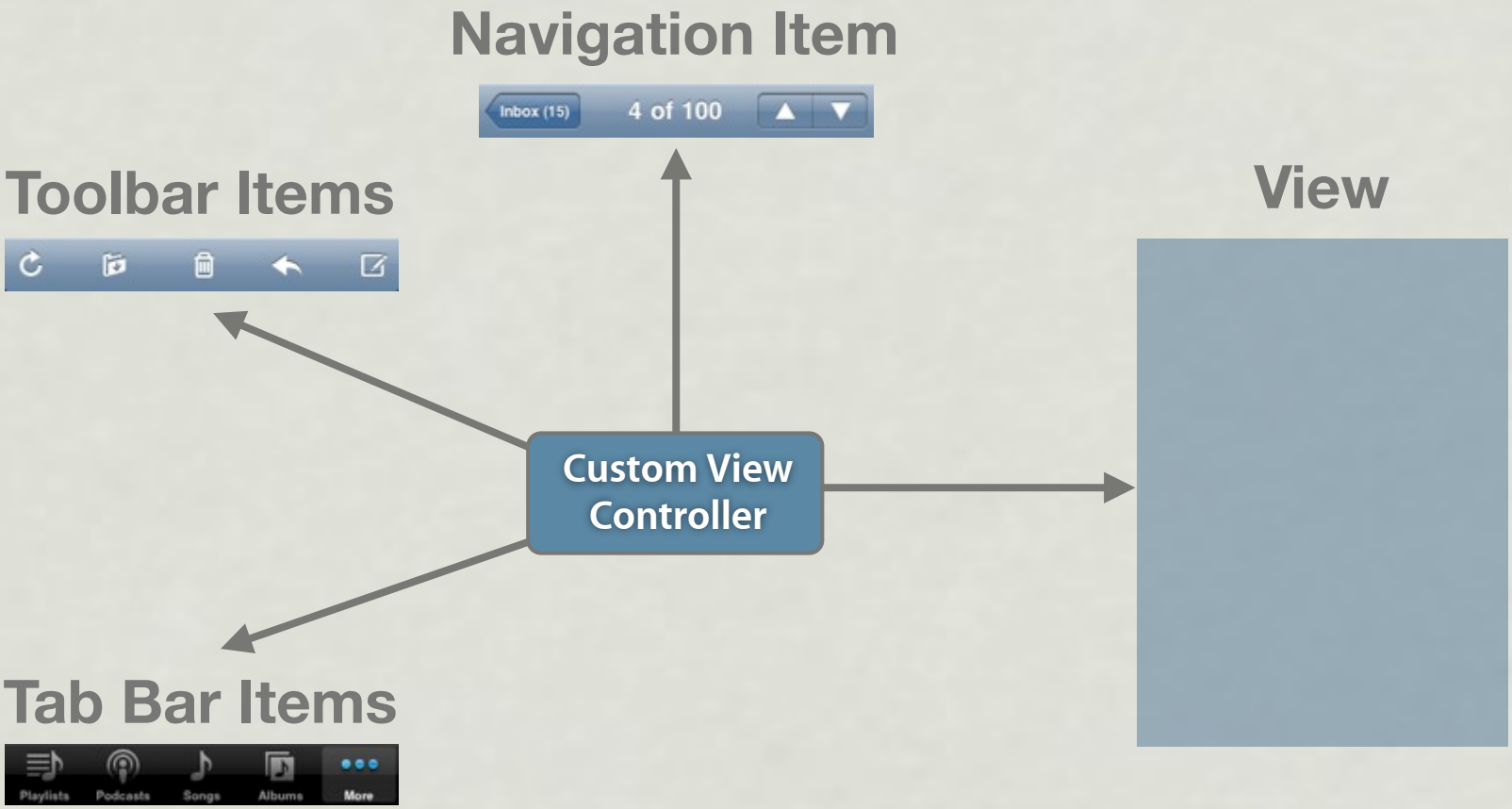
- Determine what data needs to be communicated
- Provide mechanism for passing data to sub controllers
  - Setter methods, properties
  - Parameters to your own designated initializer

Data Flow

- Determine what data needs to be communicated
- Provide mechanism for passing data to sub controllers
  - Setter methods, properties
  - Parameters to your own designated initializer
- For communicating back up the hierarchy, use loose coupling
  - Define a generic interface
    - Your own notifications
    - You own delegation or data source protocol

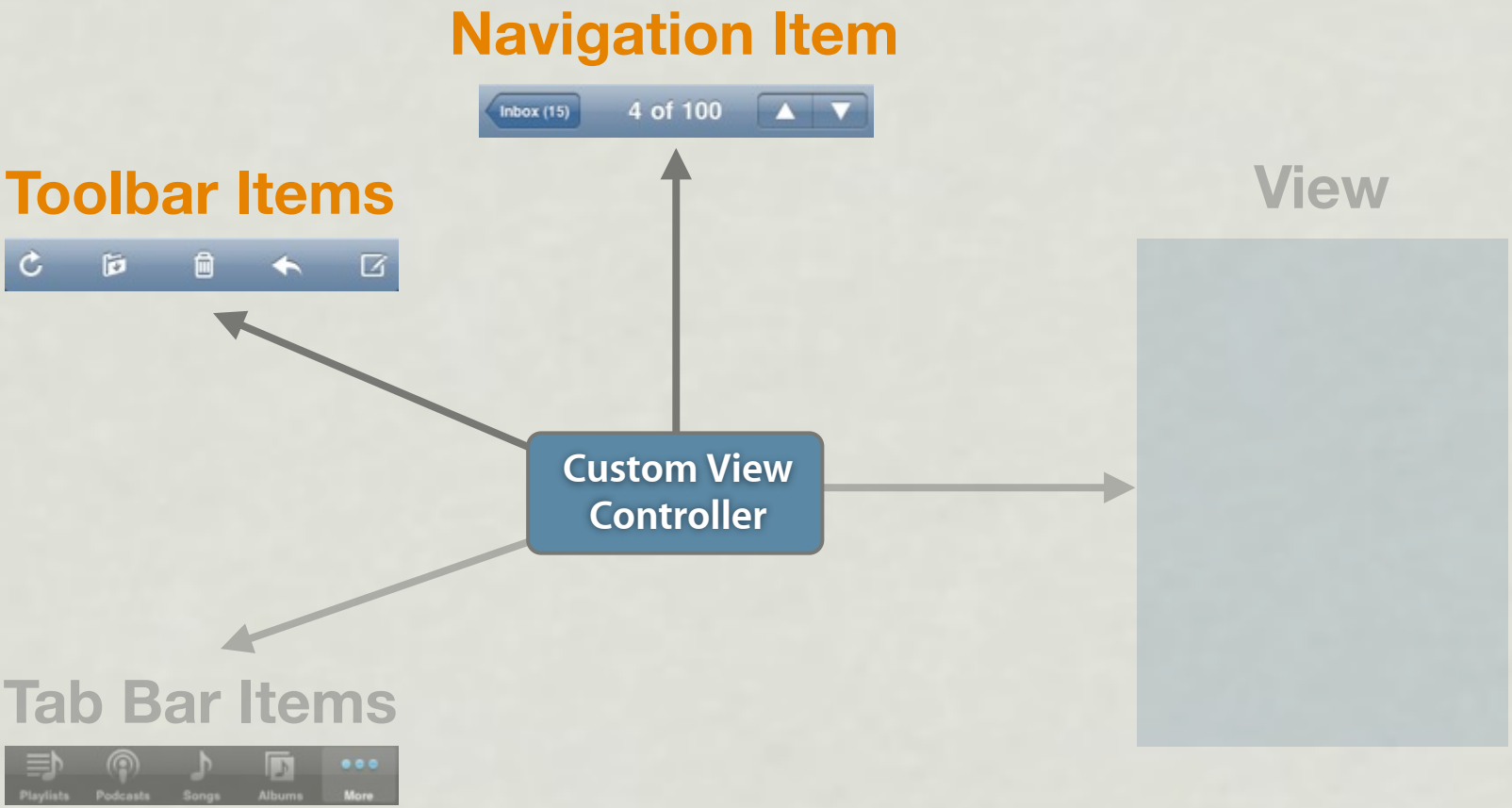
Fitting In

- Your custom view controllers within a nav / tab controller
- `UIViewController` provides access to container adornments



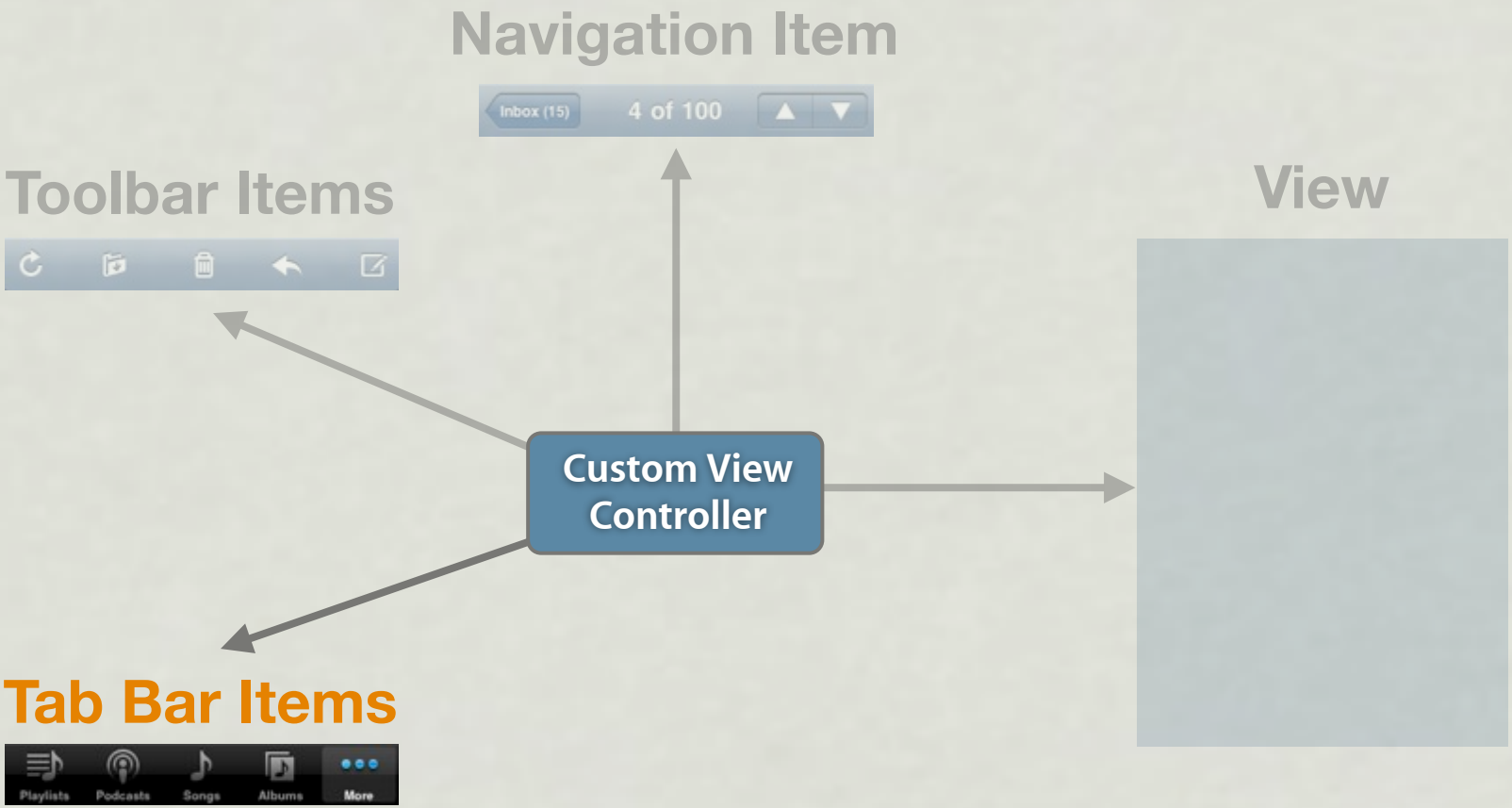
Fitting In

- Your custom view controllers within a nav / tab controller
- `UIViewController` provides access to container adornments



Fitting In

- Your custom view controllers within a nav / tab controller
- `UIViewController` provides access to container adornments



Fitting In To Tab Bars

- UITabBarController.h defines a `UIViewController` category
- API to configure tab item, access containing controller

```
@interface UIViewController (UITabBarControllerItem)

// Returns the nearest containing tab bar controller
@property(nonatomic,readonly,retain) UITabBarController *tabBarController;

// Default is an item that displays the view controller's title.
// Created lazily, so don't use this if not in a tab bar
@property(nonatomic,retain) UITabBarItem *tabBarItem;

@end
```

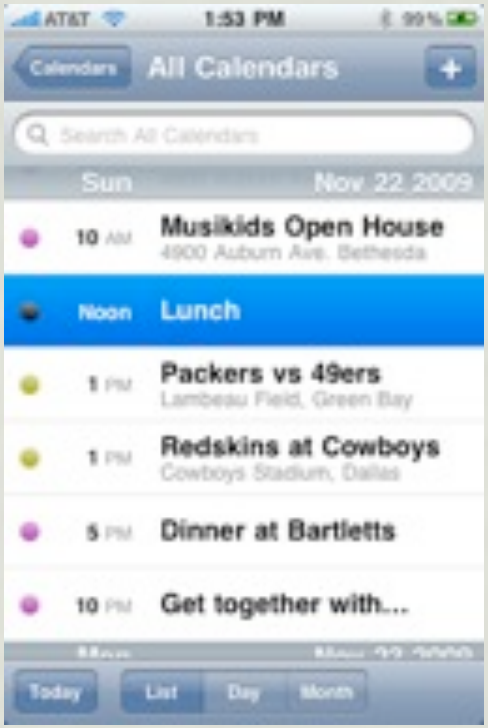
Fitting In To Navigation

- `UINavigationController.h` defines a `UIViewController` category
- API to configure containing controller's navigation bar

```
@interface UIViewController (UINavigationControllerItem)

// Default is an item that uses the view controller's title (title, back button)
// Created lazily, so don't use this if not in a navigation controller
@property(nonatomic,readonly,retain) UINavigationItem *navigationItem;

// Default is NO. If YES, pushing this view controller hides any bottom bar
@property(nonatomic) BOOL hidesBottomBarWhenPushed;

// Returns the nav controller containing this view controller.
@property(nonatomic,readonly,retain) UINavigationController *navigationController;

@end
```

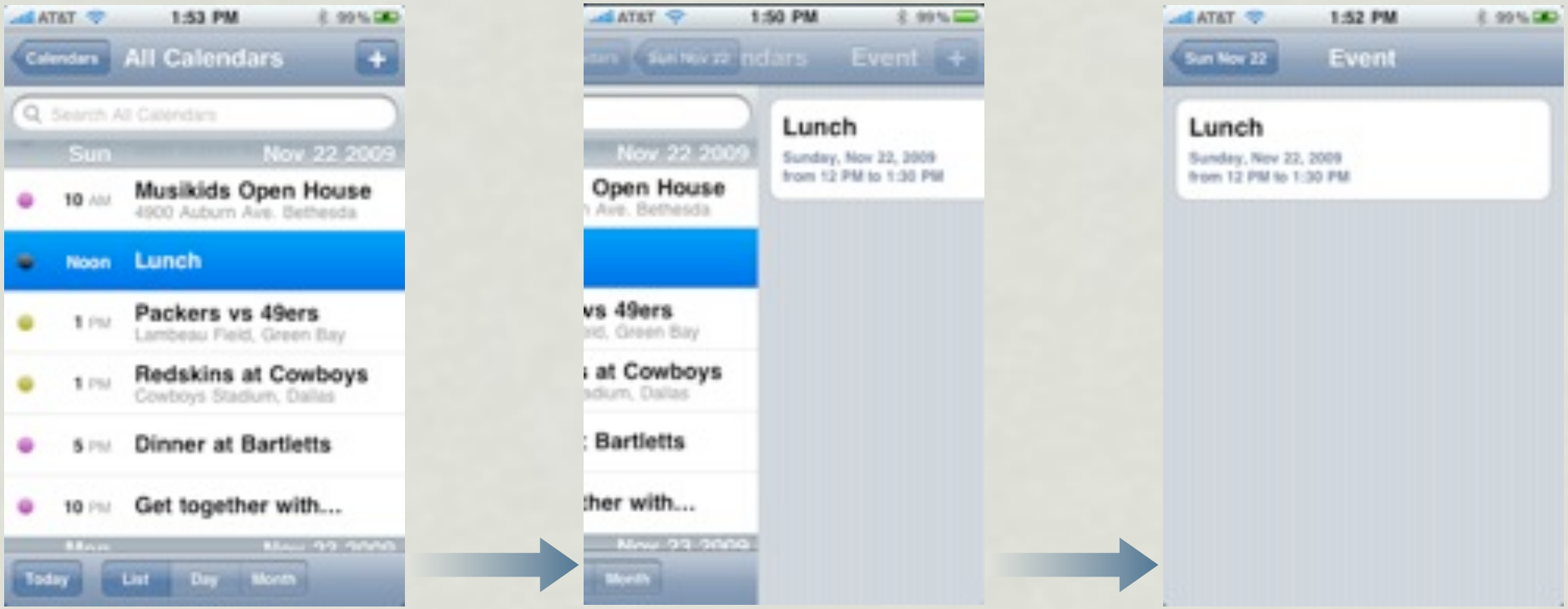
- `navigationItem` – default settings usually fine
- `hidesBottomBarWhenPushed` – hides toolbars, button bars

Hides Bottom Bar



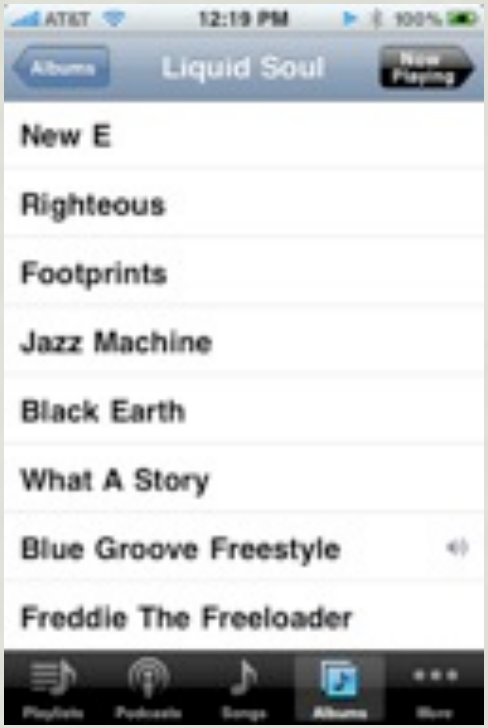
Example: Toolbar

Hides Bottom Bar



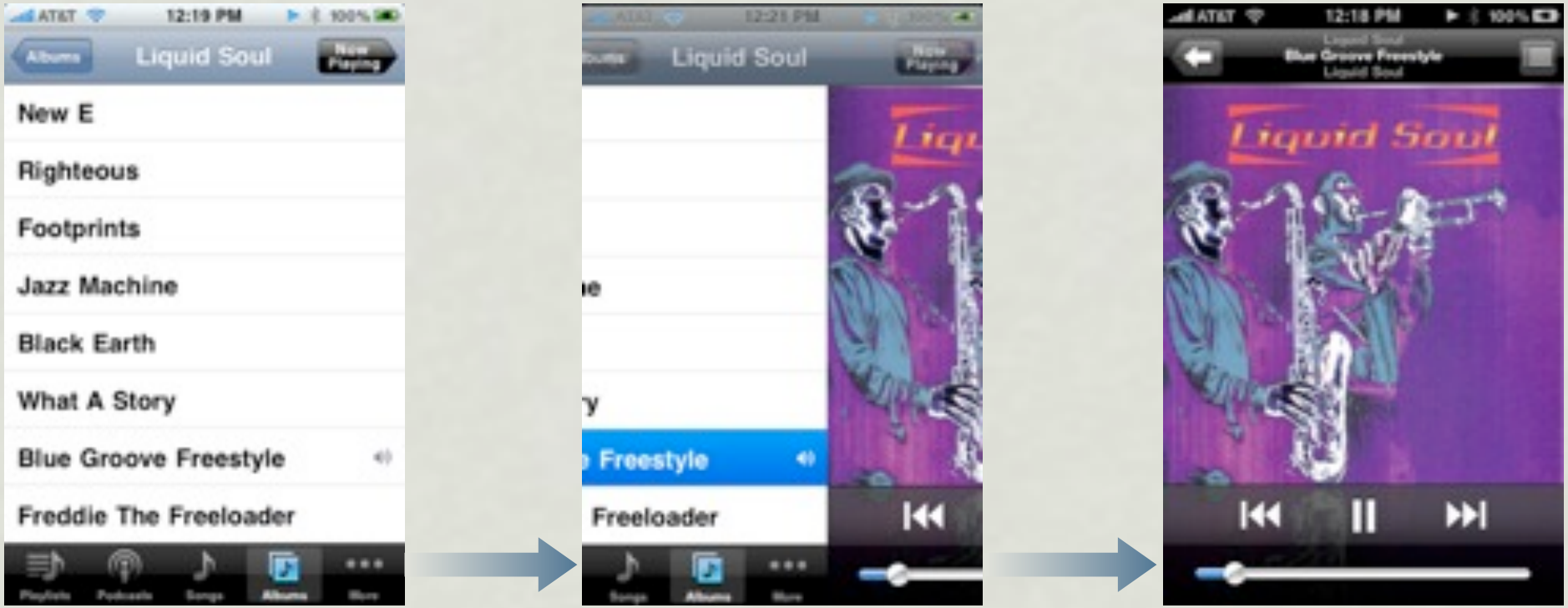
Example: Toolbar

Hides Bottom Bar



Example: Tab Bar

Hides Bottom Bar



Example: Tab Bar

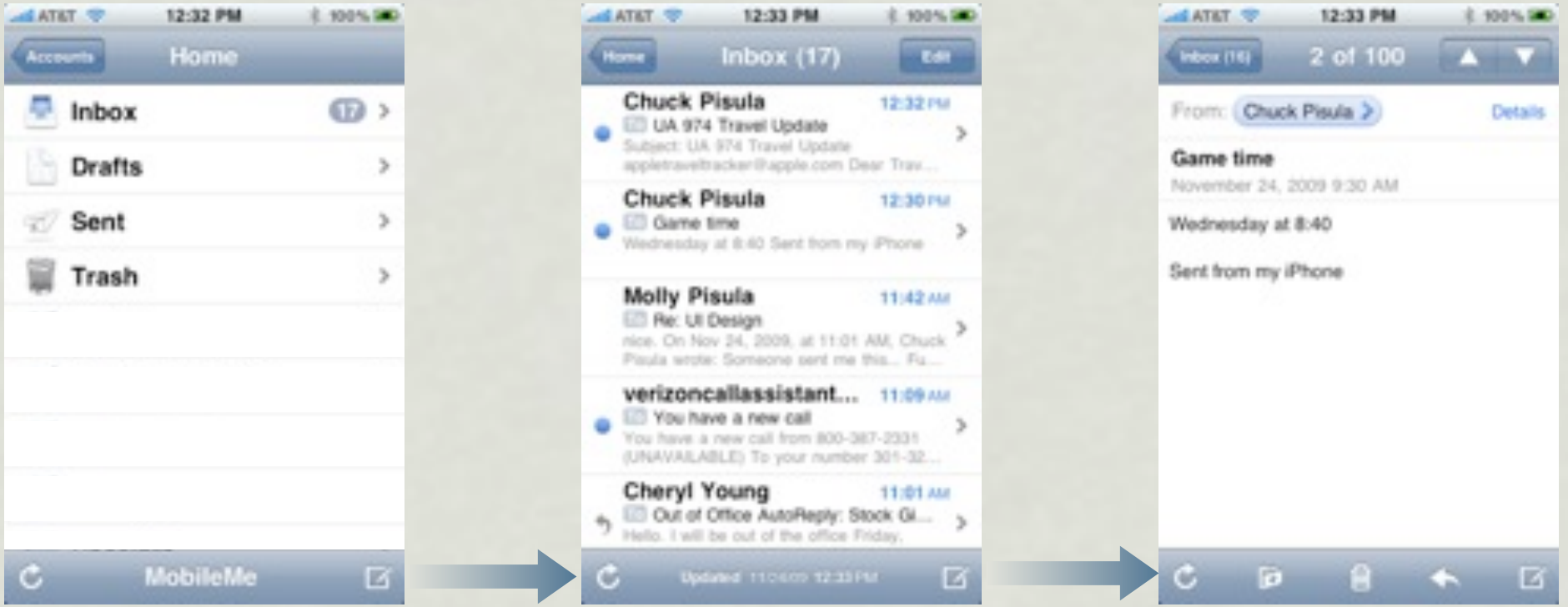
Fitting In To Navigation

- More `UIViewController` categories focused on…
- API to configure toolbar

```
@interface UIViewController (UINavigationControllerContextualToolbarItems)

// The list of UIBarButtonItems to display in a navigation toolbar
// Configure before display or any time after…
@property (nonatomic, retain) NSArray *toolbarItems;

@end
```

Toolbar Example

- Per-View Controller toolbar items



Modality

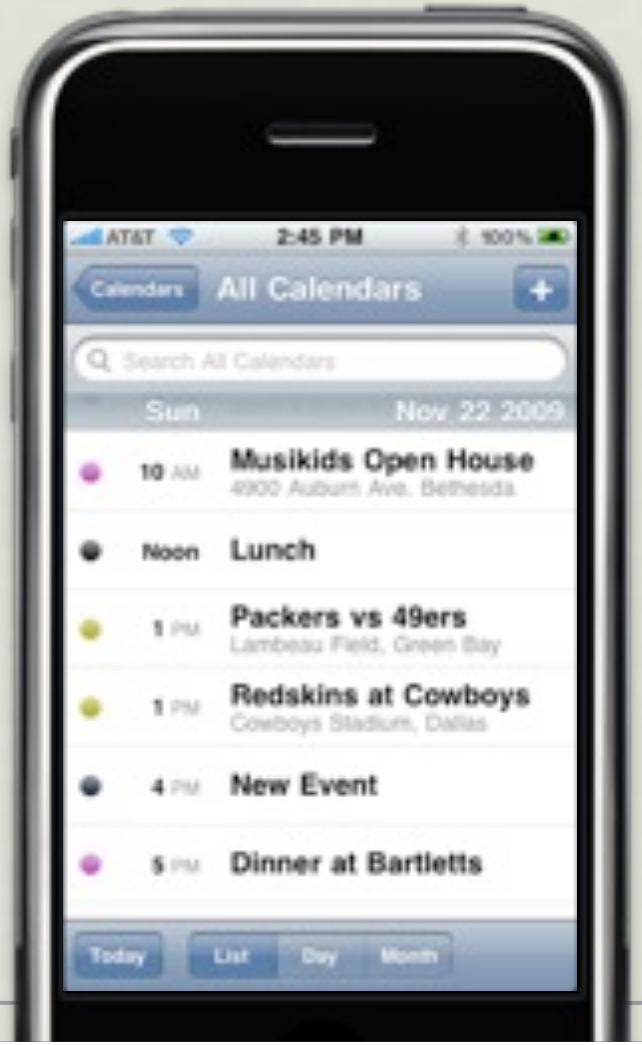
Modal

- Modal Interfaces prevent user interaction
  - On desktop, a modal alert disables interacting with rest of app
  - iPhone modal controllers cover screen, thereby disabling interaction with all but the modal controller chain

Modal

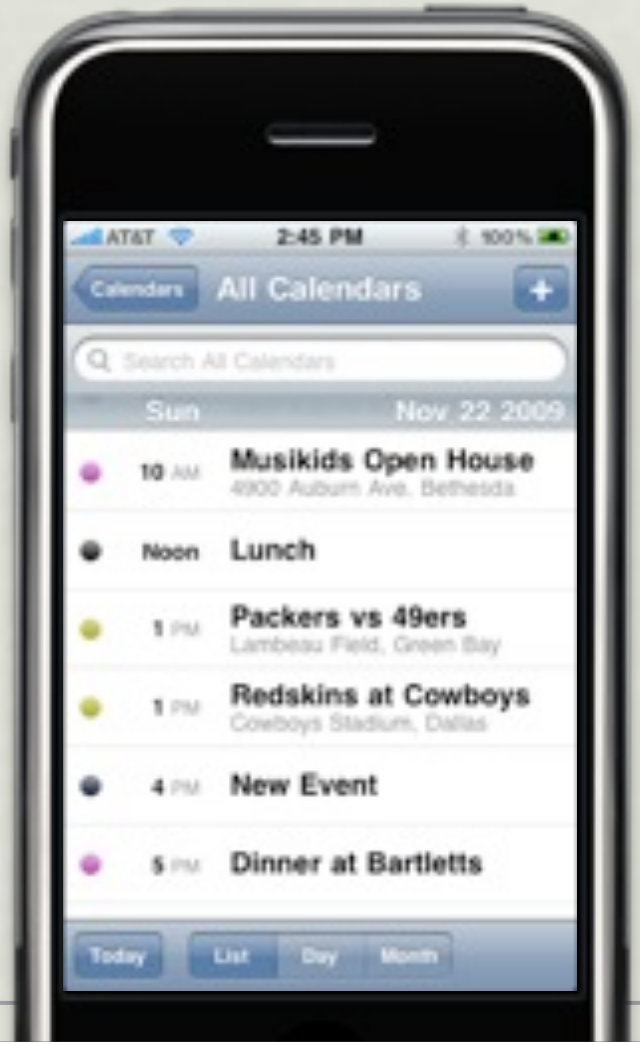
- Modal Interfaces prevent user interaction
  - On desktop, a modal alert disables interacting with rest of app
  - iPhone modal controllers cover screen, thereby disabling interaction with all but the modal controller chain
- When to use
  - user attention is required before proceeding
    - Request Input
    - Present Temporary Info
  - Rotation UI – surprising but true! More later…

Presenting Modal Controllers



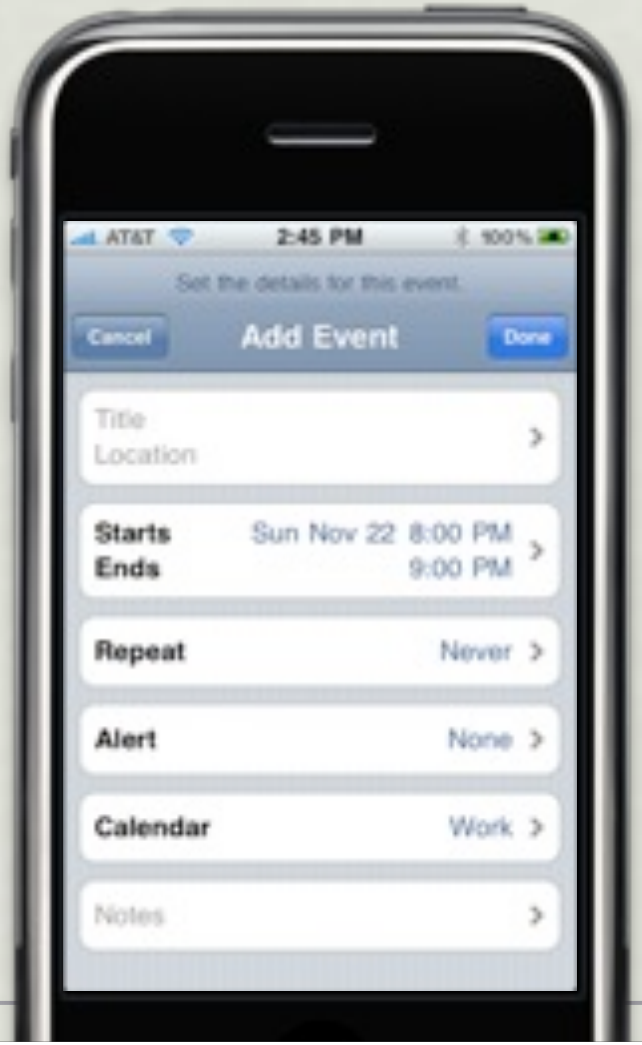
Presenting Modal Controllers

```
AddEventViewController *modalVC = ...;
[calVC presentModalViewController:modalVC animated:YES];
```



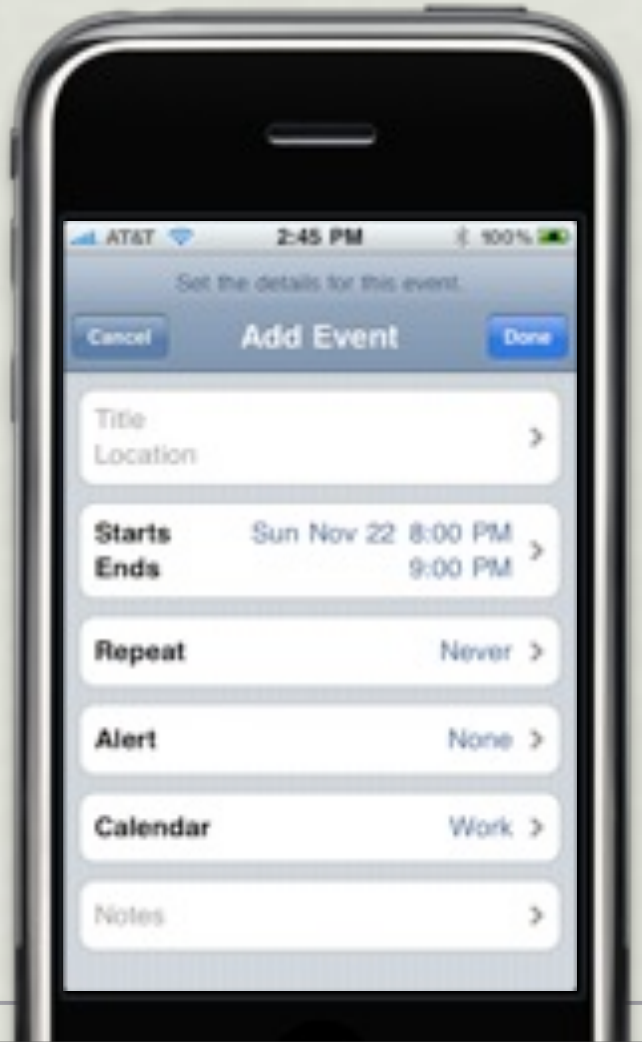
Presenting Modal Controllers

```
AddEventViewController *modalVC = ...;
[calVC presentModalViewController:modalVC animated:YES];
```



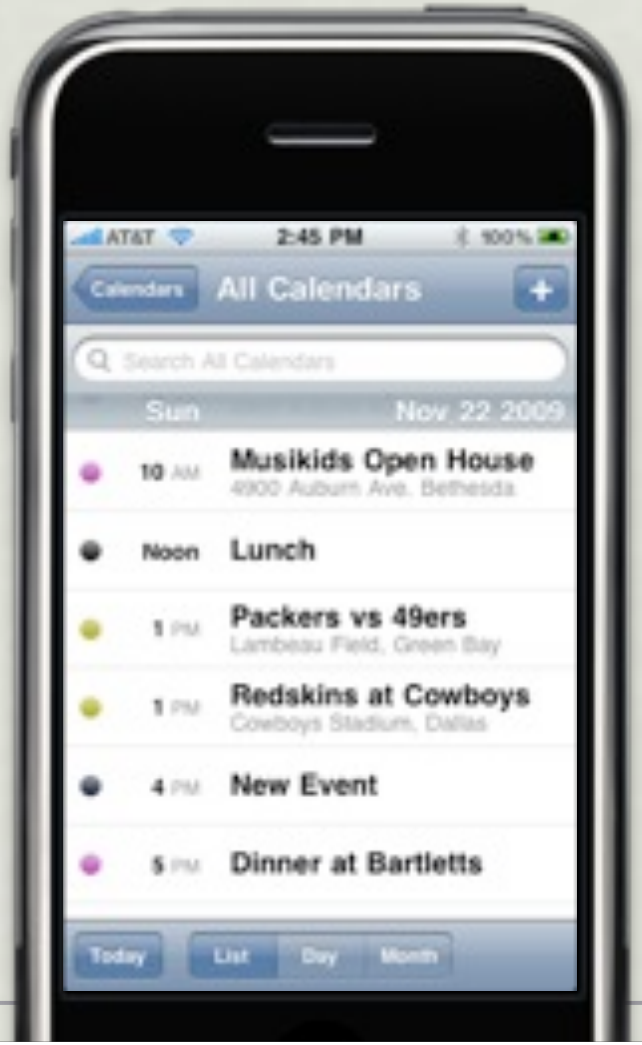
Presenting Modal Controllers

```
[calVC dismissModalViewControllerAnimated:YES];
```



Presenting Modal Controllers

```
[calVC dismissModalViewControllerAnimated:YES];
```

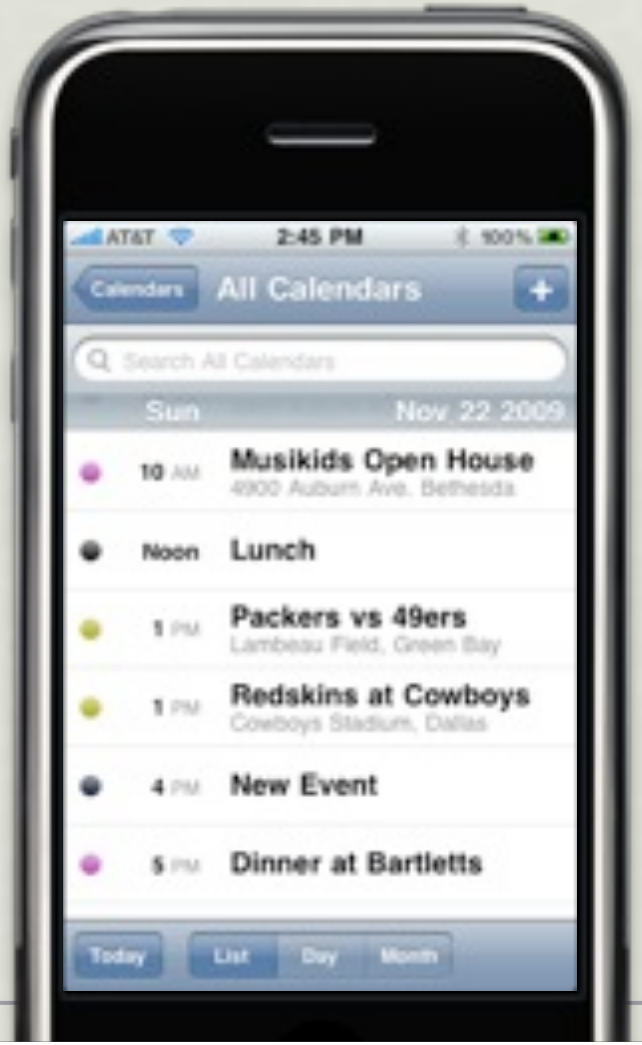


Presenting Modal Controllers

- Presenting
  - Attach to the topmost view controller to ensure screen is covered
- Dismissing
  - Usually in response to a delegate “done” callback

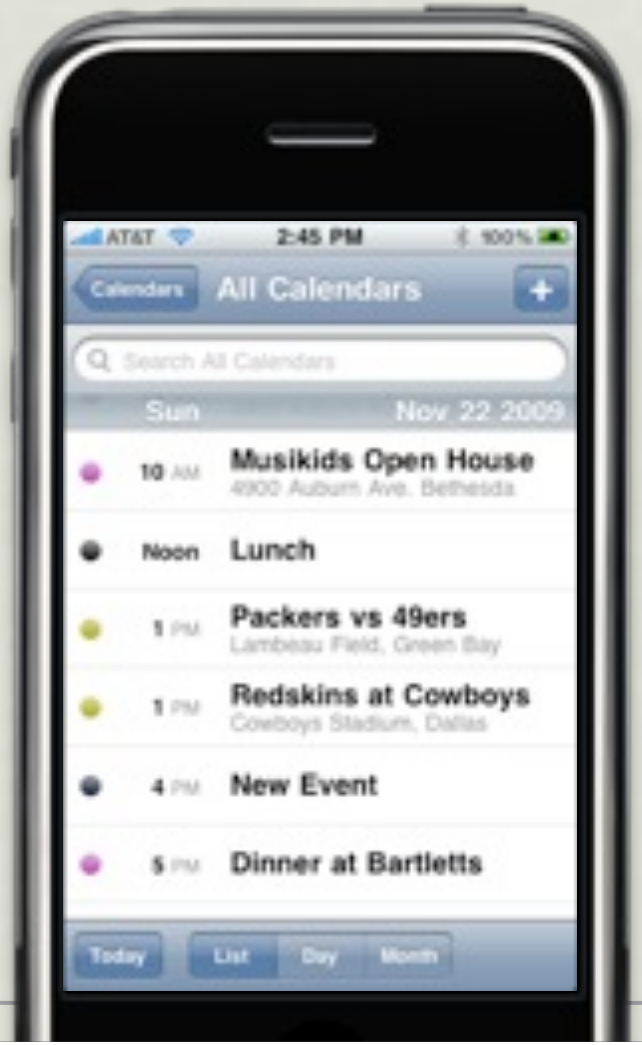
Modal Transition Styles

```
viewController.modalTransitionStyle = UIModalTransitionStyleVertical;
```



Modal Transition Styles

```
viewController.modalTransitionStyle = UIModalTransitionStyleVertical;
```



Modal Transition Styles

```
viewController.modalTransitionStyle = UIModalTransitionStyleCrossFlipHorizontal;
```

Modal Transition Styles

```
viewController.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
```

Modal Transition Styles

```
viewController.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
```

Prompt

- Modal views often display explanatory text
- Prompt attribute available in `UINavigationItem`

```
UINavigationItem *item = [self navigationItem];
[item setPrompt: @”Set the details for this event.”];
```

Modal Controller Chain

- `UIViewController` modal view controller hierarchy

```
// Reference to the modal view controller starting point
@property(nonatomic,readonly) UIViewController *modalViewController;

// If inside a navigation or tab controller returns that view controller
// If a modal view controller, returns view controller it is attached to…
@property(nonatomic,readonly) UIViewController *parentViewController;
```

- Can be starting point for a chain of view controllers
  - Modal controllers can even present other modal view controllers
- Many system features exposed as view controllers
  - `UIImagePickerController`
  - `ABPeoplePickerNavigationController`

Rotation

Rotation

- Checklist for Custom View Controllers
  - Return YES from `-shouldAutorotateToInterfaceOrientation:`
  - Setup `autoresizing` masks to handle supported orientations
  - Implement orientation change steps
- Orientation change steps
  - One step process
  - Two step process

One Step Rotation

- Your view controller receives

```
- (void)willRotateToInterfaceOrientation:...
```

  - Hide any views necessary
  - Make changes to view layout before rotation

- Then you receive

```
- (void)didRotateFromInterfaceOrientation:...
```

Two Step Rotation

- In addition to the override points...
- Additional override points
- First Half

```
-(void)willAnimateFirstHalfOfRotationToInterfaceOrientation:
-(void)didAnimateFirstHalfOfRotationToInterfaceOrientation:
```

- Second Half

```
-(void)willAnimateSecondHalfOfRotationToInterfaceOrientation:
```

- Use
  - Typically used to hide a portrait version of some UI
  - E.g. Video Player controls fade out, then back in

Orientation Based Interface

- Example
  - Calculator application
  - Portrait – simple calculator
  - Landscape – scientific calculator
- Easiest Solution
  - Use multiple view controllers
  - One view controller for each orientation

Orientation Based Interface

- `MyPortraitViewController`
  - Declare support for only portrait
  - Using `UIDevice`, watch for orientation changes **notifications**
    - Normally you implement overrides, but you've disabled the auto rotation
- `MyLandscapeViewController`
  - Managed by the portrait view controller
- Use modal presentation!
  - Present landscape view modally when necessary

Orientation Based Interface

- How To...

```
- (void)orientationChanged:(NSNotification *)notification

{
    UIDeviceOrientation deviceOrientation = [UIDevice currentDevice].orientation;

    if (UIDeviceOrientationIsLandscape(deviceOrientation) &&
        !isShowingLandscapeView)
    {
        [self presentModalViewController:self.landscapeViewController
                                animated:YES];
        isShowingLandscapeView = YES;
    }
    else if (deviceOrientation == UIDeviceOrientationPortrait &&
             isShowingLandscapeView)
    {
        [self dismissModalViewControllerAnimated:YES];
        isShowingLandscapeView = NO;
    }
}
```

Orientation Based Interface

- How To...

```
- (void)orientationChanged:(NSNotification *)notification

{
    UIDeviceOrientation deviceOrientation = [UIDevice currentDevice].orientation;

    if (UIDeviceOrientationIsLandscape(deviceOrientation) &&
        !isShowingLandscapeView)
    {
        [self presentModalViewController:self.landscapeViewController
                                 animated:YES];
        isShowingLandscapeView = YES;
    }
    else if (deviceOrientation == UIDeviceOrientationPortrait &&
             isShowingLandscapeView)
    {
        [self dismissModalViewControllerAnimated:YES];
        isShowingLandscapeView = NO;
    }
}
```

Editing Support

Editing

- `UIViewController` provides hooks to deal with editing
- Access to the configured “edit” bar button

```
- (UIBarButtonItem *)editButtonItem;
```

- Override point, called by standard “edit” bar button

```
- (void)setEditing:(BOOL)editing animated:(BOOL)animated;
```

- Standard behavior
  - `UIViewController` toggles button UI
  - `UITableViewController` forwards edit action to its table

Tab Bar Customization

- `UITabBarController` provides API to support user reordering

```
// If not empty, the “More” screen will show an “edit” button
@property(nonatomic,copy) NSArray *customizableViewControllers;


@protocol UITabBarControllerDelegate <NSObject>

// viewControllers represents the new display order chosen by the user
- (void)tabBarController:(UITabBarController *)tabBarController
      didEndCustomizingViewControllers:(NSArray *)viewControllers
      changed:(BOOL)changed;
@end
```

Search

- We'll discuss search more later in the semester
- For now…
  - Search UI provided by attaching a `UISearchDisplayController`

```
@interface UIViewController (UISearchDisplayControllerSupport)

@property(...) UISearchDisplayController *searchDisplayController;

@end
```

Review

Review

- Custom View Controller Checklist
  - Create UI in `-loadViews`, or in IB (and implement `-nibName`)
  - Define methods for passing data
  - Define delegate methods / notifications for communicating changes
  - Memory efficiency (`-viewDidUnload`, `-didReceiveMemoryWarning`)
  - Rotation Support
  - Define actions, implement to push new `controllers` on the stack
  - Configure UI - `nav item`, `tab bar item`, `toolbar items`

Review

- Application Flow
  - Create top level controller
  - For Tab Bar Controller
    - Create controllers for each tab, and call `setViewControllers:`
  - For Navigation Controller
    - Create and connect `root controller`, call `setRootController:`
- Saving application state
  - Need to remember stack of nav items, etc...
  - Will discuss in lectures dealing with “data persistence”

Reading

- "View Controller Programming Guide For iPhone OS"
  - ViewControllerPGforiPhoneOS.pdf
  - Read "Custom View Controllers" - p.21 - 52
  - Read "Combined View Controllers" - p.97 - 103
- iPhone OS Reference Library
  - UINavigationController Class Reference
    - Supporting Classes UINavigationItem, UIBarButtonItem