iPhone Application Programming
Lecture 6: View Controller Programming, Core Graphics and Core Animation

Nur Al-huda Hamdan
Media Computing Group
RWTH Aachen University

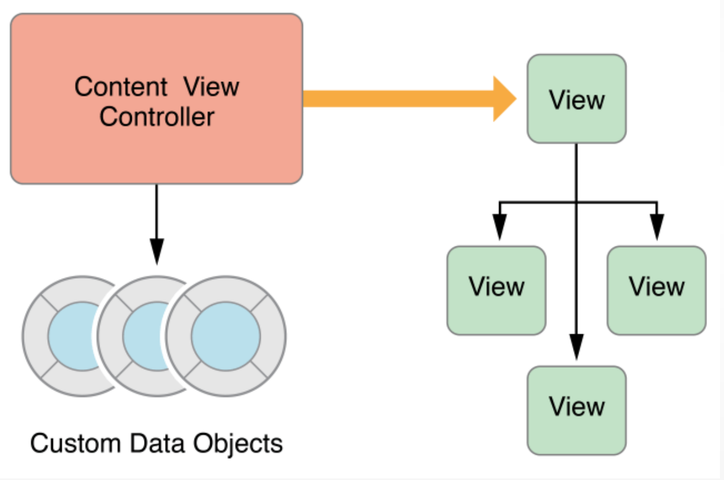
Winter Semester 2015/2016

http://hci.rwth-aachen.de/iphone

Media Computing Group | RWTH AACHEN UNIVERSITY

Role of View Controller

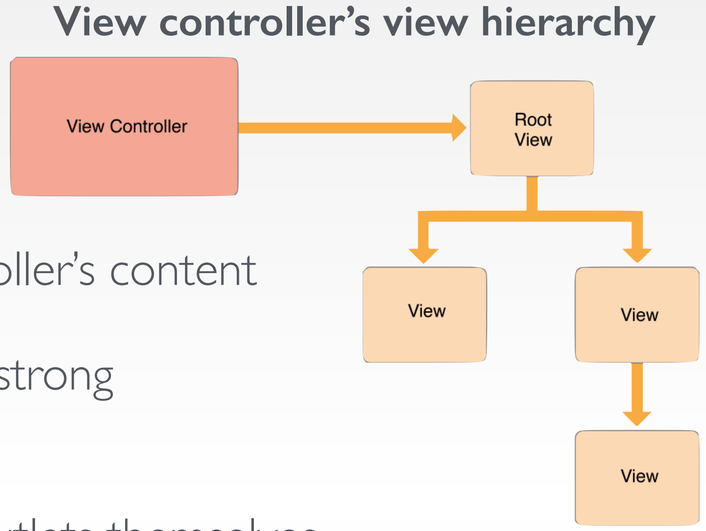
- Every app has at least one view controller
- Manages part of the app's user interface and interaction between the interface and underlying data
- Facilitates transitions between different parts of the user interface
- UIViewController class defines the methods and properties for managing views, handling events, transitioning from one view controller to another, and coordinating with other parts of the app
- An app subclasses UIViewController and adds custom app behavior



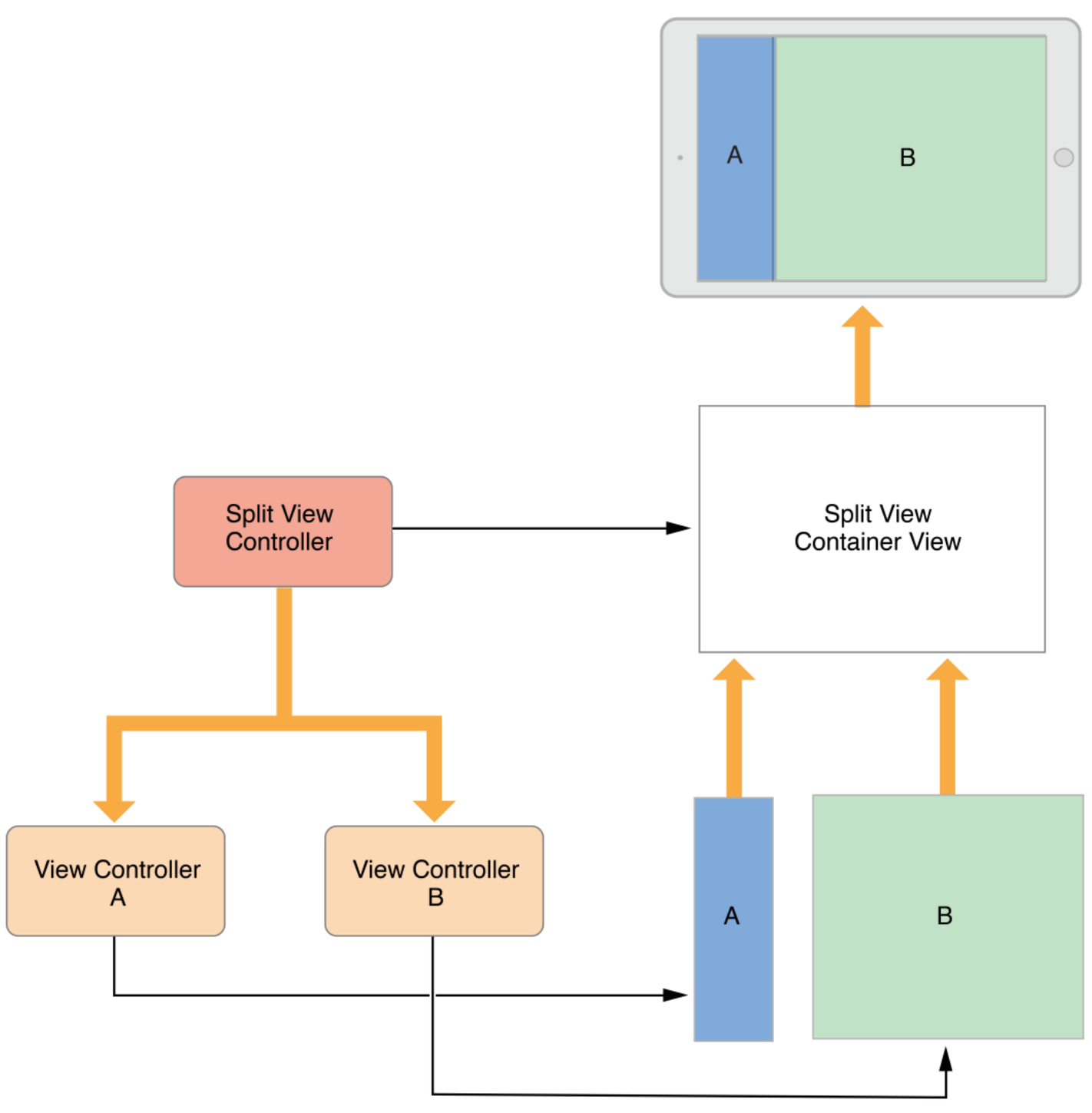
Type of View Controllers

- Two types of view controllers
  - Content view controllers manage a discrete piece of your app's content and are the main type of view controller that you create
  - Container view controllers collect information from other view controllers (known as child view controllers) and present it in a way that facilitates navigation or presents the content of those view controllers differently
- Most apps are a mixture of both types of view controllers

View Management

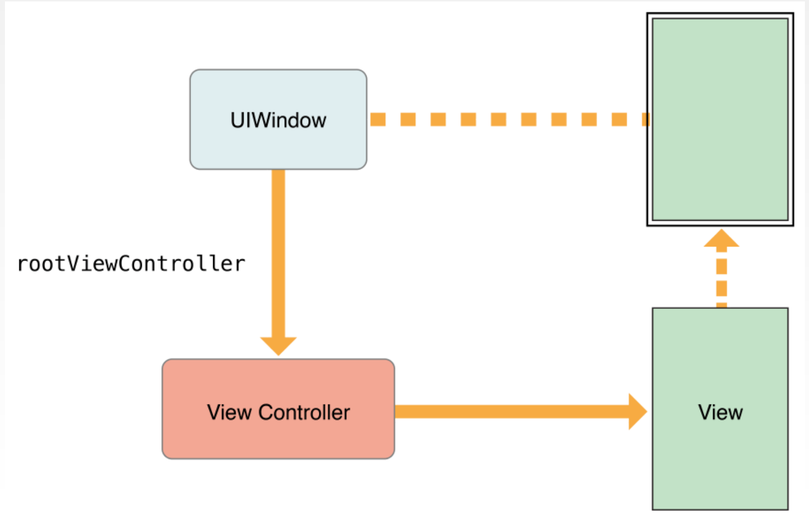


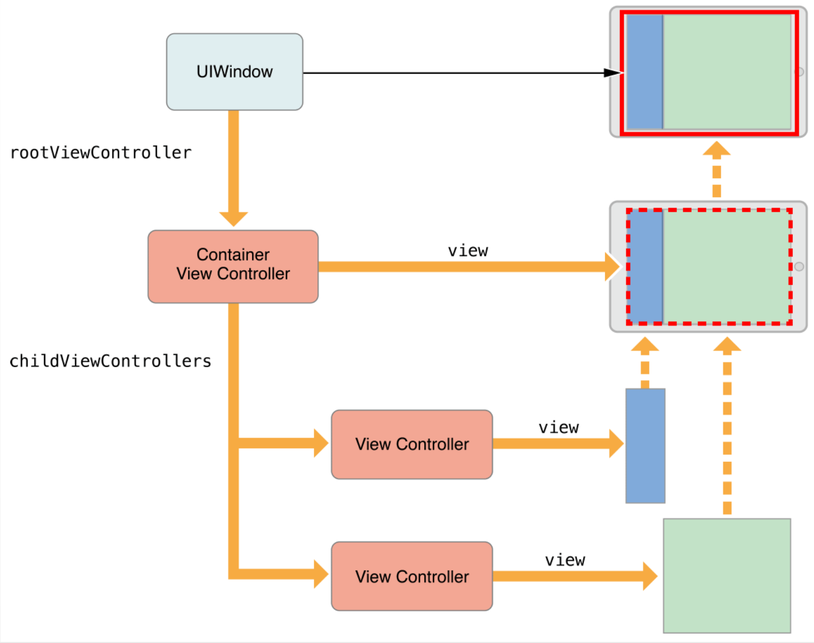
- Most important role of a view controller is to manage a hierarchy of views
- Every view controller has a single root view that encloses all of the view controller's content
- The view controller always has a reference to its root view and each view has strong references to its subviews
- A view controller uses outlets to access different views in the hierarchy. The outlets themselves are connected to the actual view objects automatically when loaded from the storyboard
- A content view controller manages all of its views by itself
- A container view controller manages its own views plus the root views from one or more of its child view controllers
  - It sizes and places root views according to the container's design (root views hierarchies are managed by their owner view controller)



Root View Controller

- The root view controller is the first node in the view controller hierarchy
- Every window has exactly one root view controller whose content fills that window
- The root view controller defines the initial content seen by the user
- A container view controller mixes the content of one or more child view controllers together with optional custom views
  - UINavigationController displays the content from a child view controller, a navigation bar and an optional toolbar, which are managed by the navigation controller
- UIKit includes several container view controllers: UINavigationController, UISplitViewController, UITabBarController, and UIPageViewController
- The container is responsible for positioning and size its child views appropriately



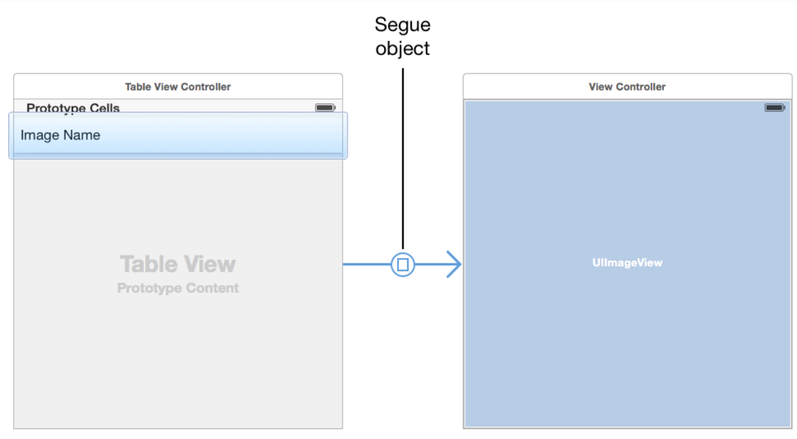


Presenting a View Controller

- There are two ways to display a view controller onscreen: embed it in a container view controller or present it programatically
- Support for presenting view controllers is built in to the UIViewController
  - The showViewController and showDetailViewController methods offer the most adaptive and flexible way to display view controllers. These methods let the presenting view controller decide how best to handle the presentation, e.g., a container view controller might incorporate the view controller as a child instead of presenting it modally
  - The presentViewController method always displays the view controller modally
- Presenting a view controller creates a relationship between the original view controller, known as the presenting view controller, and the new view controller to be displayed, known as the presented view controller (view controller hierarchy)
- UIKit lets you display a new view controller using built-in or custom animations
- You can initiate the presentation of a view controller programmatically or using segues. If you know your app's navigation at design time, segues are the easiest way

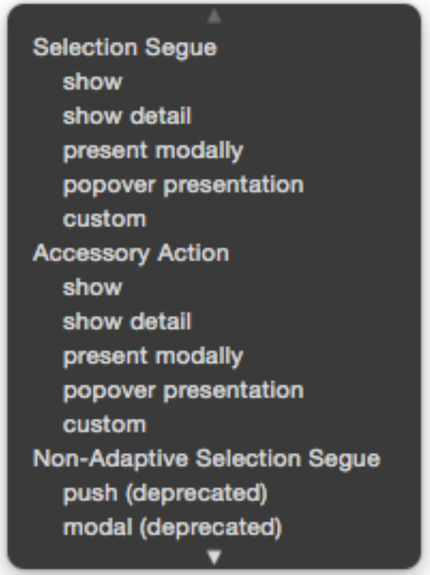
Segues

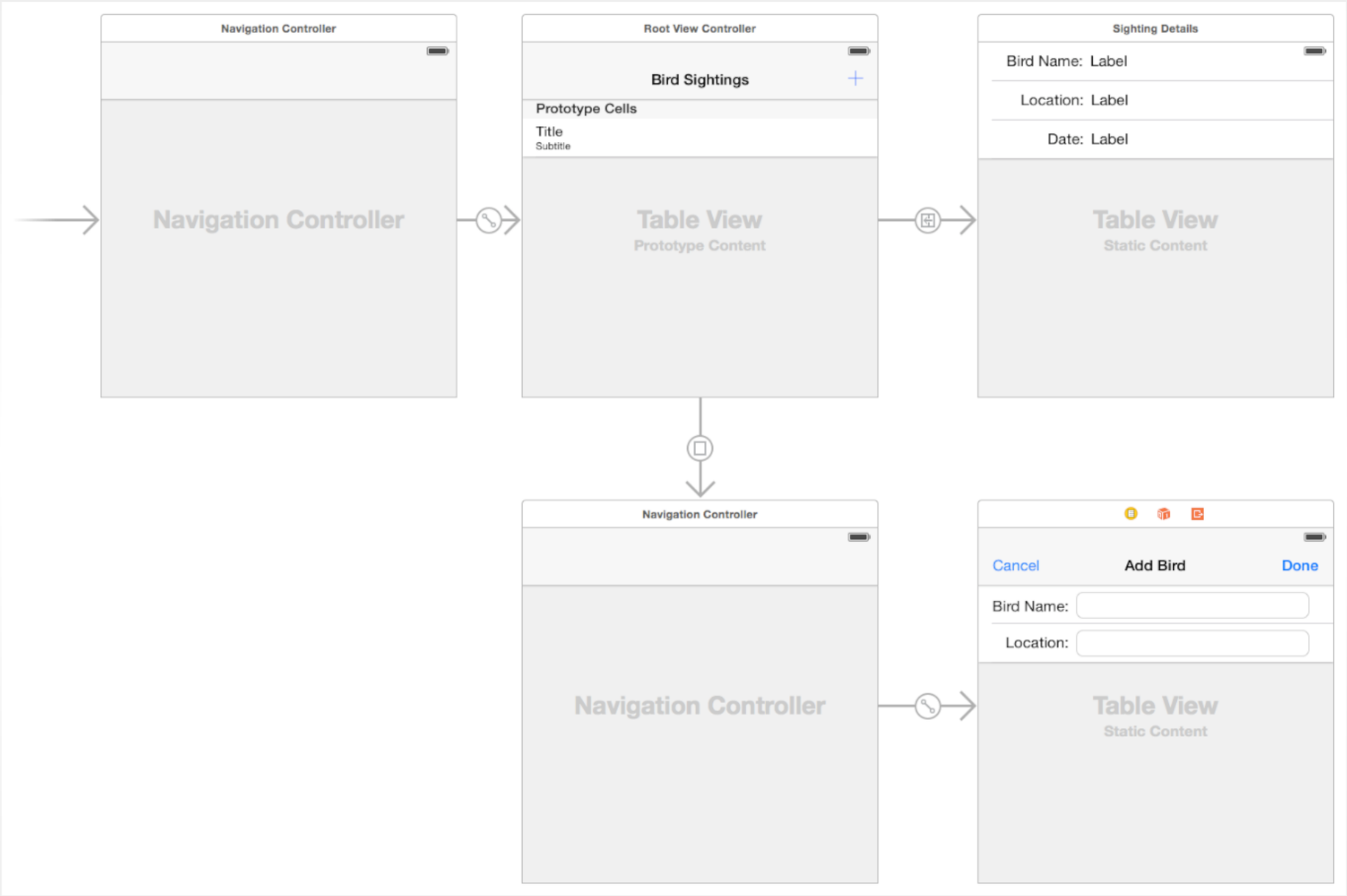
- Define the flow of your app's interface
- A transition between two view controllers in your app's storyboard file
- The starting point of a segue is the button, table row, or gesture recogniser that initiates the segue. The end point of a segue is the view controller you want to display
- A segue always presents a new view controller, but you can also use an unwind segue to dismiss a view controller
- Assign the segue an identifier to access in code
- Segues always create new view controller not existing ones even when you go back–you are creating a new one because once yo move from one its removed totally



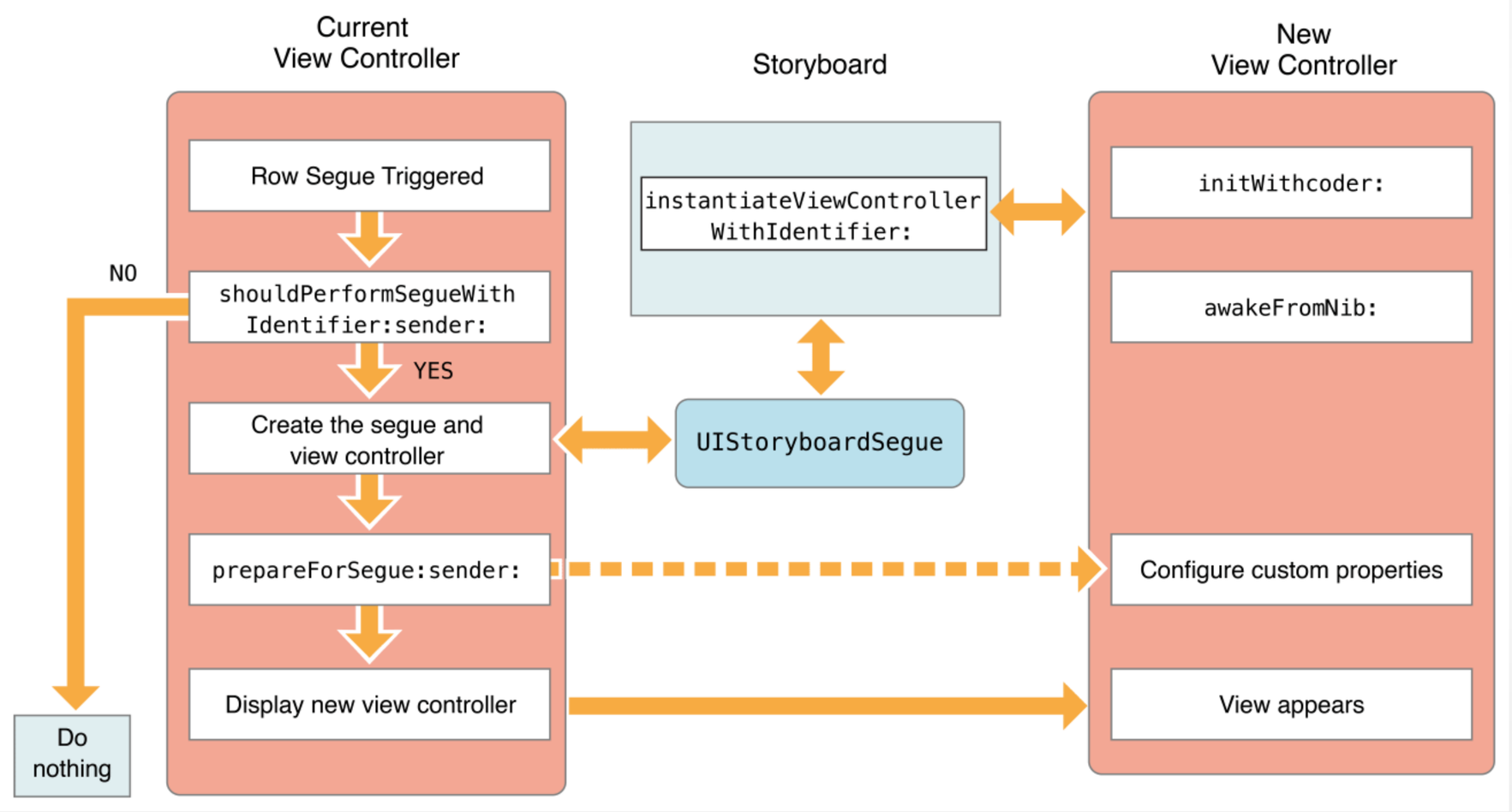
Type of Segues

- Four types and custom
- Show push view controller on top stack if using navigation controller. If not a navigation controller, it displays modally (covers the entire screen)
- Show detail: if navigation controller this is like show, if in split view then you specify what to share the screen with
- Modal takes over the screen (not the best practice)
- Popover is a a bit nicer than modal because you can access the background controller and click on to dismiss the popover
- You can create relationships (parent-child) and transitions





Segues at Runtime



Preparing to Navigate

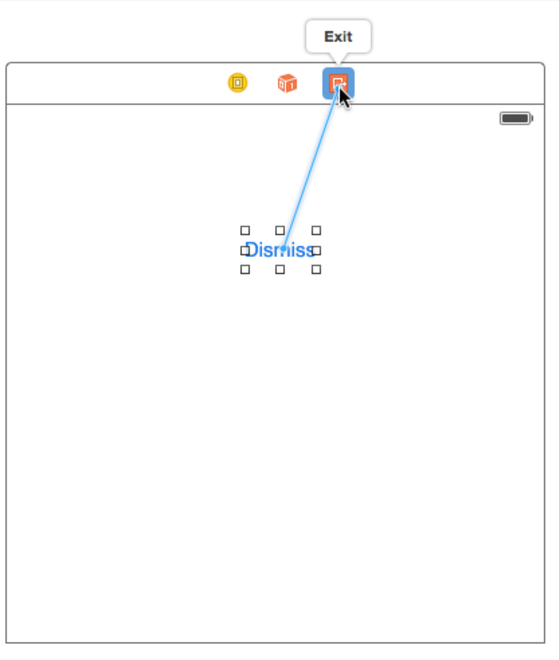
- On the source view controller side
- Do not access the outlets of the destination, they are still not set

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    var dvc = segue.destinationViewController as? UIViewController
    if let nvc = segue.destinationViewController as? UINavigationController
    {
        dvc = nvc.visibleViewController
    }
    if segue.identifier == "Segue identifier" {

        dvc!.view.backgroundColor = UIColor.redColor()
    }
}
```

Unwinding Segues

- Choose the destination view Drag to the Exit object at the top of the view controller scene
  - Define an action: @IBAction func myUnwindAction(unwindSegue: UIStoryboardSegue)
- From the source view controller drag to exit, and hook up with the above action
- You do not need to dismiss the view controller in this action
- Usr the segue object to fetch the view controller being dismissed so that you can retrieve data from it

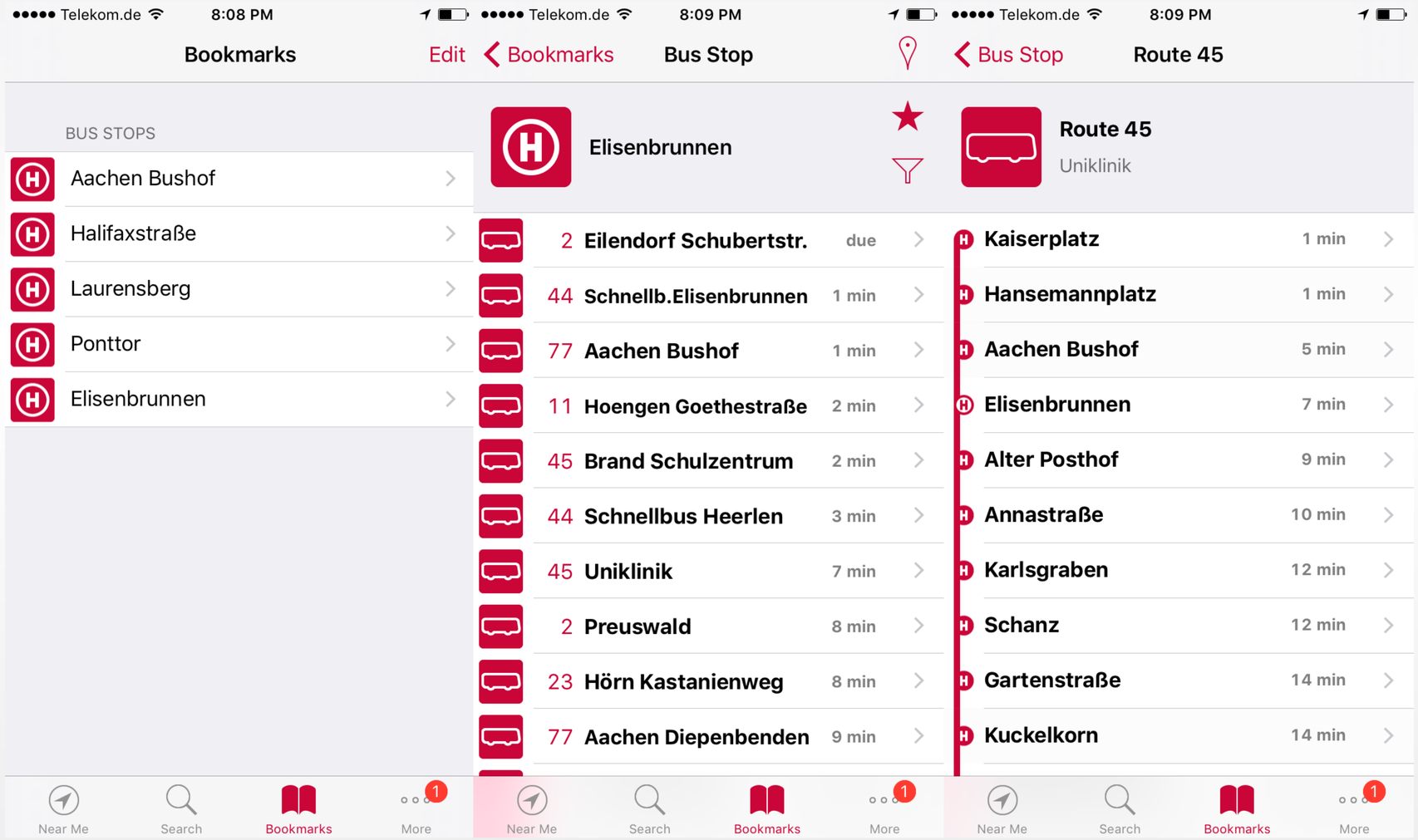


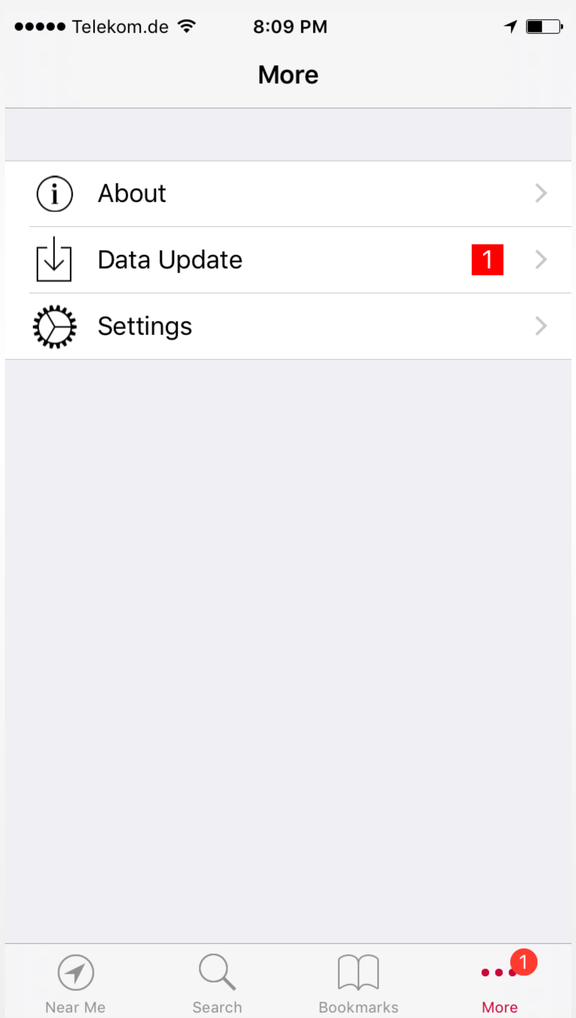
The source view controller

Navigation

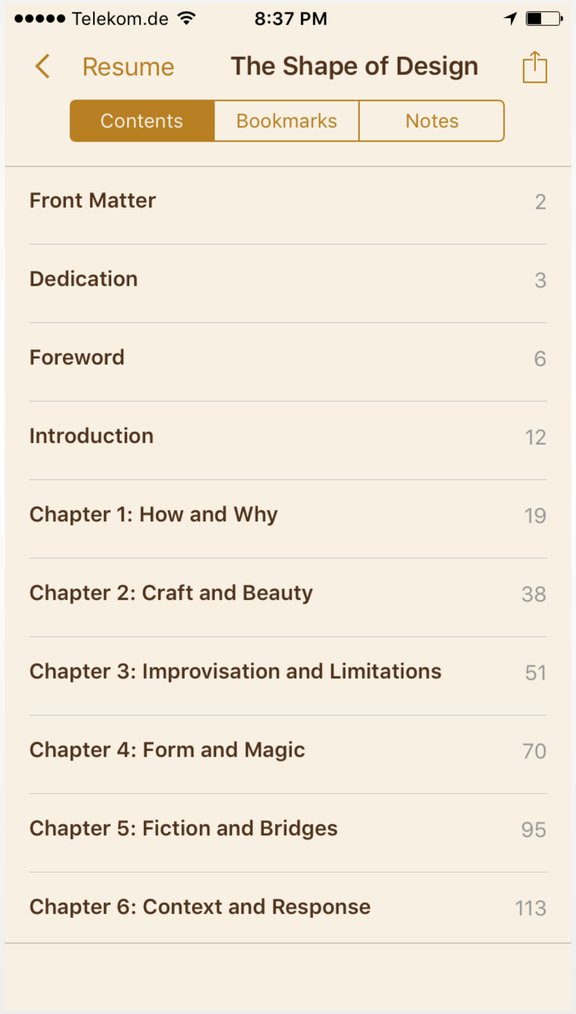
- Three main styles of navigation
  - Hierarchical: one choice per screen, retract one step at a time or to the beginning, e.g., navigation bar (title and back button)
  - Flat: all main categories are accessible from any screen, e.g., tab bar
  - Content or experience-driven
- Can combine more than one navigation style in an app

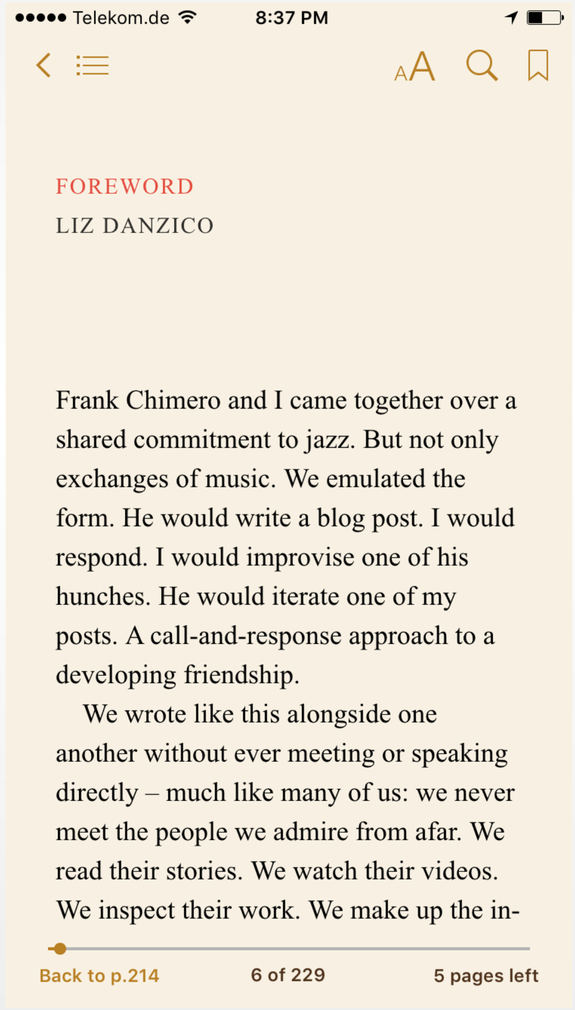
Hierarchical and Flat Nav.

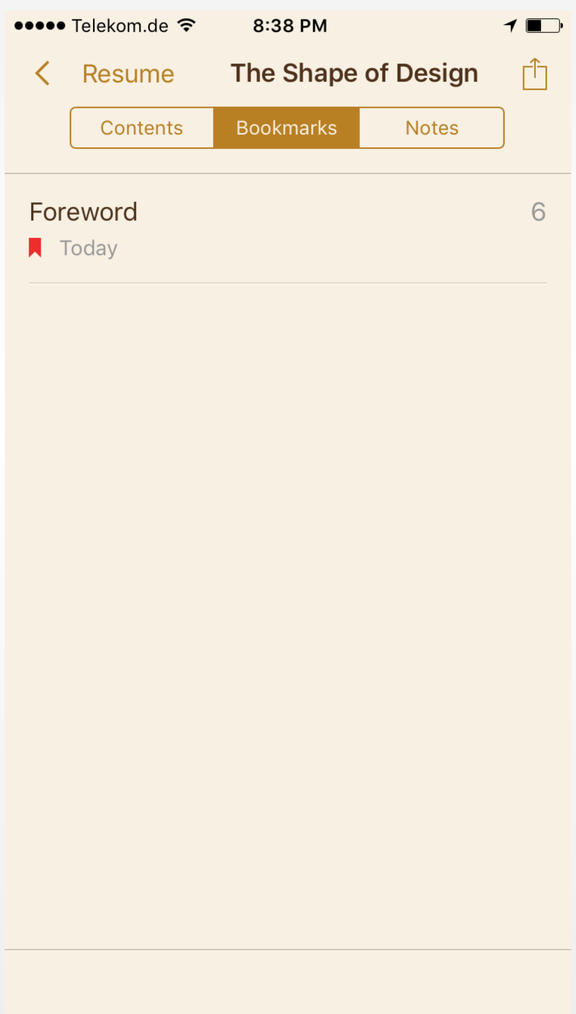




Content or Experience Driven Nav.

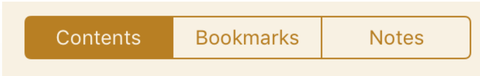




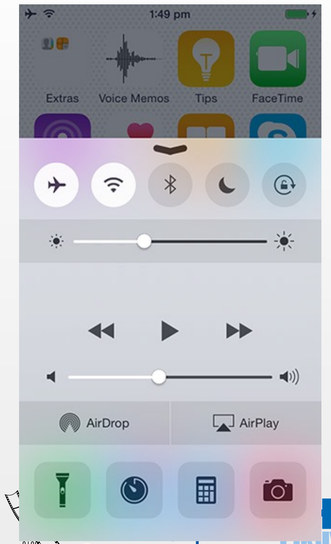


Pseudo Nav.

- Page control allows you to move between screens, each representing an instance of the same type or page. Indicates number of pages, and the selected one



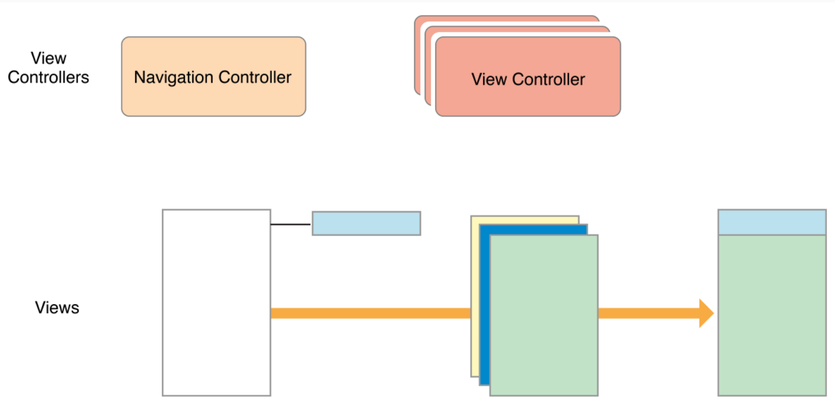
- Segmented control allows you to see different categories of the content on the screen (it doesn't enable navigation to a new screen)



- Use a *temporary view* for a screen that users want to access from different contexts

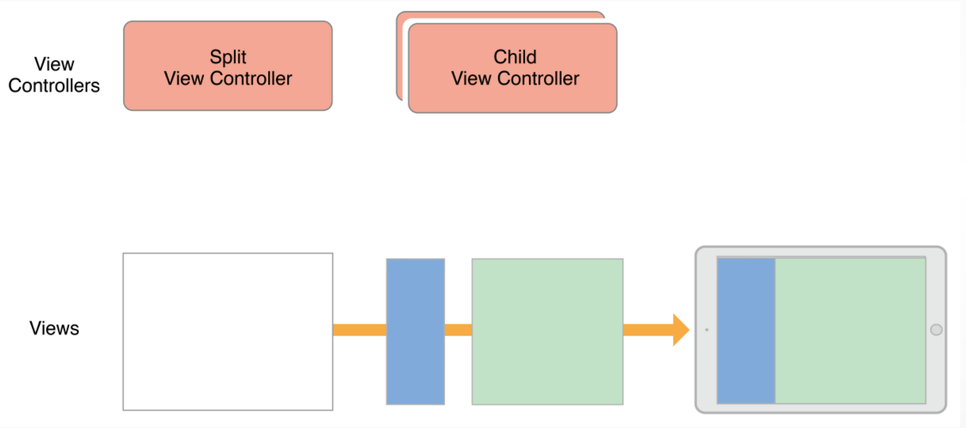
UINavigationController

- Supports navigation through a hierarchical data set
- Presents one child view controller at a time
- A navigation bar at the top of the interface displays the current position in the data hierarchy and displays a back button
- When the user interacts with a button or table row of a child view controller, the child asks the navigation controller to push a new view controller into view
- The child handles the configuration of the new view controller's contents, but the navigation controller manages the transition animations
- The navigation controller also manages the navigation bar, which displays a back button for dismissing the topmost view controller, as well as a toolbar
- The navigation controller resizes its child to fit the available space (orientation)



UISplitViewController

- Displays the content of two view controllers in a master-detail arrangement
- Adapts to iPad and iPhone
- The content of one view controller (the master) determines what details are displayed by the other view controller
- The size of the child views is configurable, as is the visibility of the master view

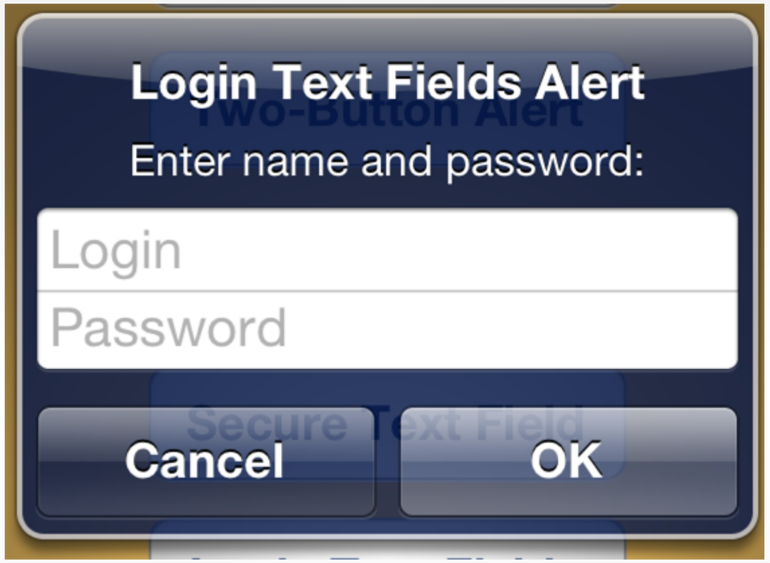


Temporary Views

- Modal context: Alerts, action sheets, and modal views
- Give users a way to complete a task or get information without distractions, but it does so by temporarily preventing them from interacting with the rest of the app.
- Use sparsely, when it is critical to get the user's attention
- An alert interrupts the user's experience and requires a tap to dismiss, so it's important for users to feel that the alert's message warrants the intrusion
- Provide clear exists: people should always be able to predict the fate of their work when they dismiss a modal view

Alert

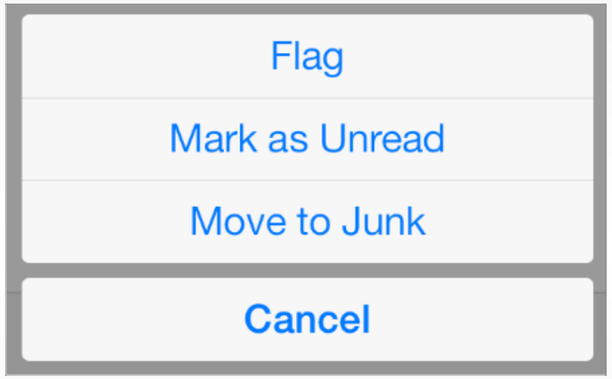
- UIAlert, UIAlertViewDelegate (optional) (UIAlertController)
- Display a concise and informative alert message to the user
- Displays a required title and an optional message
- Contains one or more buttons
- Appears on top of app content
- Used for users to make a decision about a course of action



- You can control the number of buttons and their titles (if more than 2 buttons need choose an action sheet), displayed text, and inclusion of one or two text fields (plain or secure ****) Every alert has a Cancel button so that the user can dismiss the alert.
- An alert view can be canceled at any time by the system, when the user taps the Home button
- No IB just programmatic support

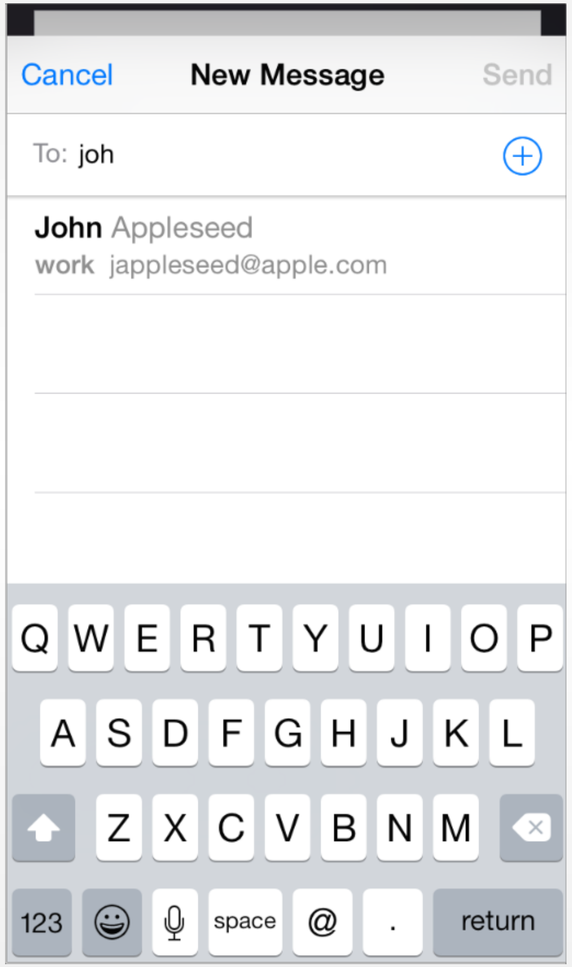
Action Sheet

- UIActionSheet, UIActionSheetDelegate
- Appears as the result of a user action (Action button)
- Displays two or more buttons representing several alternative choices to complete a task initiated by the user
- Confirm or cancel an action
  - cancel button
  - destructive button, e.g., delete image (defaults to red)
  - other buttons
- No IB just programmatic support



Modal View

- UIPresentationController
- Provides self-contained functionality in the context of the current task or workflow
- Can occupy the entire screen or a portion of the screen
- Contains the text and controls that are necessary to complete the task
- Usually displays a button that completes the task and dismisses the view a
- Choose an appropriate transition style for revealing the modal view
  - Vertical.: the modal view slides up from the bottom edge of the screen and slides back down when dismissed (this is the default transition style)
  - Flip: the current view flips horizontally from right to left to reveal the modal view



Event Handling

- View controllers are responder objects and are capable of handling events
- But usually, view controllers handle touch events indirectly using delegate methods or action methods
- Views handle their own touch events and report the results to a method of an associated delegate or target object, which is usually the view controller
- View controllers should release unneeded resources
- When the available free memory is running low, UIKit asks apps to free up any resources that they no longer need by calling the didReceiveMemoryWarning method of a view controller

Event Handling

- View controllers implement methods to respond to specific control events. Controls and some views call an action method to report specific interactions
- View controllers can attach gesture recognizers and call an action method to report the current status of a gesture
- View controllers observe notifications sent by the system or other objects. Notifications report changes and are a way for the view controller to update its state
- View controllers act as a data source or delegate for another object, such as a CLLocationManager object, which sends updated location values to its delegate
- Responding to events often involves updating the content of views, which requires having references to those views (outlets)

View Controller Life Cycle

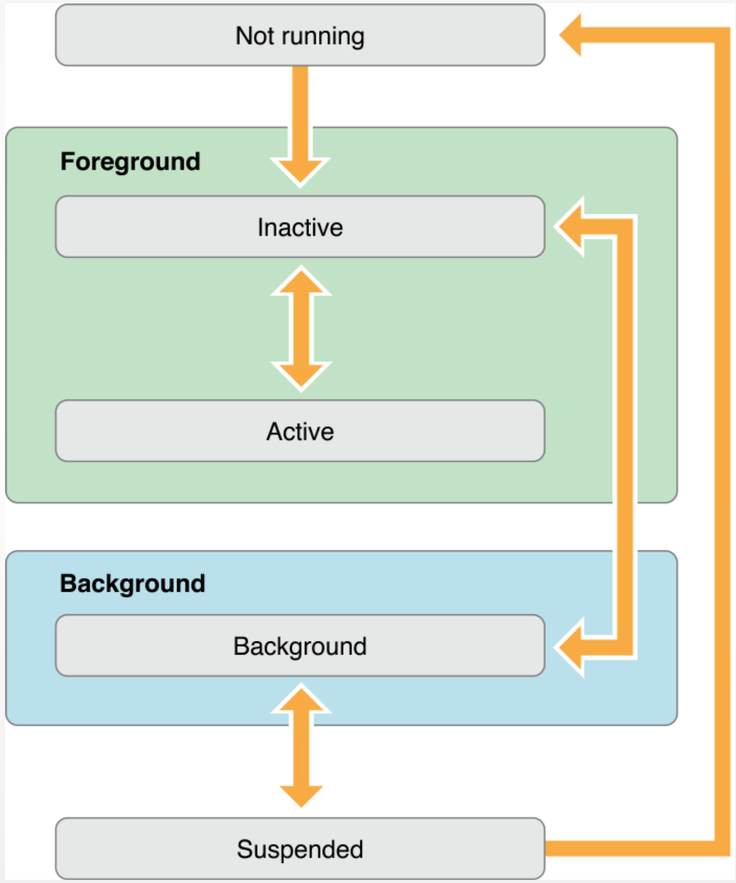
- Sequence of transition states announced by calling methods
- We override these methods to set up the view controller
- Insatiate from storyboard, prepared (if segue), outlets set, view controller appears on the screen (and disappear), change size, have memory wants
- ViewDidLoad is the best place to setup your hierarchy, update the UI, and allocate resources (your outlets are set by now). This only happens once in the life of view controller
- ViewWillLayoutSubView and viewDidLayout. Called when a view's frame change, e.g., in autorotation which changes your bounds

View Controller Life Cycle

- ViewWillAppear happens every time your view controller takes the screen. Get expensive resources on request
- ViewDidAppear after your visible
- ViewWillDisappear get rid of unneeded resources (but nothing time consuming)
- ViewDidDisappear
- didReceiveMemoryWaring get rid of large resources (audio and images)
- awakeFromNib happens just after installation, use it for delegate assignment

App Life Cycle

- The app is launched
- Foreground the user is using the app
  - Inactive: no UI events yet
  - Active: the view controller on screen and can receive events
- Background (briefly) the user switched from your app
  - You can ask a background task to run your code briefly, e.g., updates from the net
- Suspended your code is not running, not killed yet
- Killed if the battery is running out or memory is full
- Transitions on the orange lines the app sends the app notifications



App Delegate

- In the appDelegate the method didFinishLaunchingWithOptions is called
- You get a dictionary of options telling the app why it was launched, e.g., as an activity, or document reader, open URL, map notification push notifications, etc
- You can observe theses transitions using notifications
- applicationWillResignActive, pause your UI (phone call)
- applicationDidBecomeActive, play your UI
- applicationDidEnterBackground, prepare to be killed
- applicationWillEnterBackground, undo what you did in DidEnterBackground

App Delegate

- Store the state of UI
- Encrypt the data when the the screen is locked
- Open file types the app supports
- Background task, e.g., music app, voip apps
- Provide notifications with timers to wake up your app periodically

UIApplication

- UIApplication.sharedApplication() get a global instance of your app
- Delegates the work of the appDelegate
- Can open URL sent by other apps to its own app
- Register of local and push notification
- Network in use spinner
- Ask for a background task
- Ask about the state of the app

Core Graphics

Drawing in Views– Core Graphics

- 2D drawing engine
- Path-based drawing
- Transparency, shading, shadows, layers
- Core Graphics is extremely fast
  - Hardware acceleration
- Can work on a background thread
- At a lower technical level than UIKit (C-based)
  - Cannot use UIColor and UIBezierPath, but CGColor and CGPath

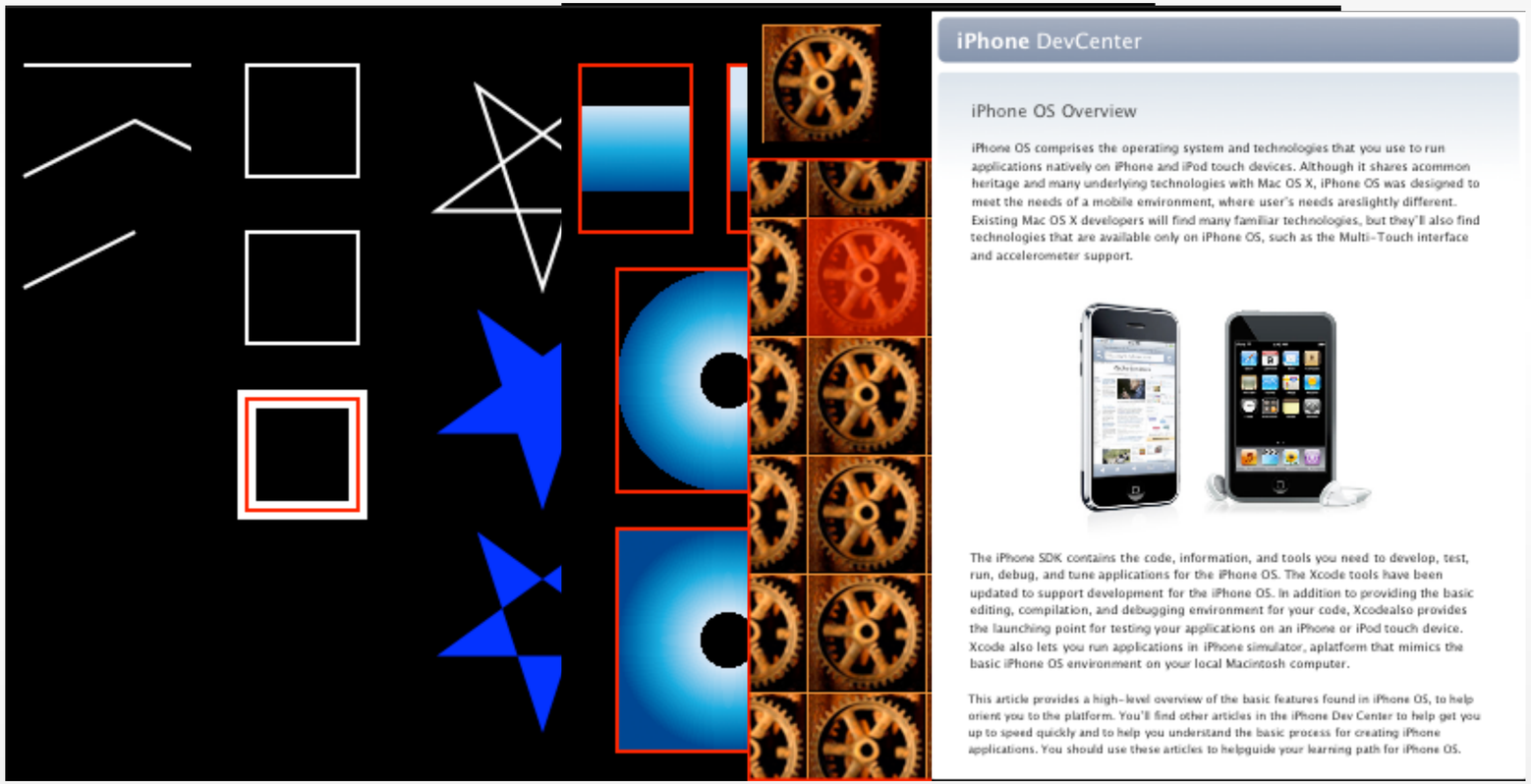
CoreGraphics Primitives

- Graphics context, is the drawing destination
- Paths
- Transformations
- Colors & Fonts
- Images & PDF

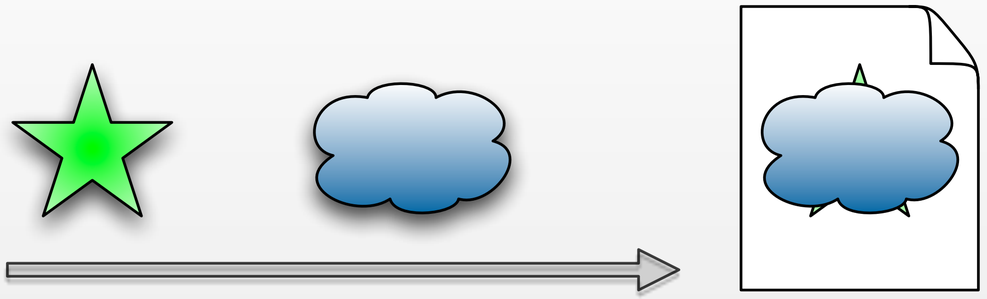
The Graphics Context

- Opaque data type (CGContext)
- Can be a view, PDF, a bitmap image, offscreen location of a layer
- Encapsulates drawing
  - Color
  - Line width
  - Other drawing parameters
- Push contexts and pop context to change drawing parameters
- Obtain the context for a view from the drawRect method

CoreGraphics Examples



Painters Drawing Model



Drawing

- Create the drawing context: UIGraphicsBeginImageContextWithOptions() also PDF and Bitmap
  - Options include: size, opaque or not, pixel to point scale
- Get the drawing context: UIGraphicsGetCurrentContext()
- Drawing
  - First you create a path: CGContextMoveToPoint, CGContextAddLineToPoint, CGContextAddRect
  - Last you drawing a path: CGContextStrokePath
- Converts context to UIImage: UIGraphicsGetImageFromCurrentImageContext()
- Terminate context: UIGraphicsEndImageContext()

And a Lot More

- Building blocks: Points Lines Arcs Curves Ellipses Rectangles
- Use can also translate, rotate, draw images and text, shadows, etc
- Several blending modes available
- Clipping along paths
- Patterns
- Gradients
- Transparency layers

Core Animation

Core Animation

- High level of abstraction to apply animations to views
  - Dynamic (animatable) attributes
  - CAAnimation class

List of Animatable Properties

- Geometric: frame, bounds, position, transform...
- Background: backgroundColor, backgroundFilters
- Border: borderColor, borderWidth
- Content: contents, contentsGravity
- Sublayers: sublayers, sublayerTransform...
- Filters, Shadow, Compositing, Masks

CALayer

- UIView equivalent for animation
  - All animation is performed in CALayers
- All UIViews are backed up by CALayers
  - (only Cocoa Touch, on demand for Cocoa)
  - Layer hierarchy in parallel to view hierarchy
  - view.layer
- You can create and animate your own layers
  - No need for a view

Implicit Animations

- Layers offer many animatable properties
- Changing their value creates an implicit animation
  - The presented value is changed over time (0.25s)
- Every layer has a presentation and a model layer
  - Presentation Layer: currently displayed values
  - Model Layer: target values

Animating Views

- Delays: gradually fade from view over a period of 3 seconds
- Animation curves define whether an animation is performed at a constant speed, whether it starts out slow and speeds up and so on
  - UIViewAnimationOptions.CurveLinear
  - UIViewAnimationOptions.CurveEaseOut – fast then slow
  - UIViewAnimationOptions.CurveEaseIn – slow then fast
  - UIViewAnimationOptions.CurveEaseInOut – slow fast slow
- Adding a code block to the completion to execute when the animation was completed
- Transformations allow changes to be made to the coordinate system of a screen (rotate, resize and translate a view)

```
UIView.animateWithDuration(3.0, delay: 5.0,
    options:
UIViewAnimationOptions.CurveLinear,
    animations: {
        self.button.alpha = 0
    },
    completion: ({finished in
        if (finished) {
            UIView.animateWithDuration(3.0,
animations: {
                self.button.alpha = 1.0
            })
        }
    }))
```

```
let scaleTrans = CGAffineTransformMakeScale(2, 2)
let angle = CGFloat(45 * M_PI / 180)
let rotateTrans =
CGAffineTransformMakeRotation(angle)

self.button.transform =
CGAffineTransformConcat(scaleTrans, rotateTrans)
```

Explicit Animation

- Create animation object
  - CABasicAnimation
  - CAKeyframeAnimation
- Configure animation
  - Duration
  - Timing function
- Configure animation target
  - Key path of animated property
  - fromValue: and toValue:

Combining Animations

- Multiple animations can be added to a layer
  - But: only one per key
- Animations will be played in parallel

Next Time

- The slides from this lecture will be uploaded to our website
- This week's reading assignment will be on the website today
- Next week we'll talk about Rendering in iOS