

# Programmieren fürs iOS

## 5. UIKit: Interface Builder, ViewControllers & SwiftUI



# Inhalt "UIKit"

- Interface Builder, inkl. Verbindung Layout-Code
- ViewController
  - Content vs. Container
  - UINavigationController
  - Übergänge: modal vs. show (push)
  - Storyboards & Segues
- Interoperabilität UIKit-SwiftUI
  - UIHoistingController
  - UIViewControllerRepresentable



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Intro: UIKit vs. SwiftUI

# SW01: UIKit vs. SwiftUI



- Aktuell zwei UI-Technologien
    - UIKit: der alte **imperative** König
      - ViewControllers, Target-Action-Muster, usw.
    - SwiftUI: der junge **deklarative** Kronprinz...
      - Zukunftsträchtiges "modernes" deklaratives UI-Framework (ähnlich zu Flutter, Jetpack Compose, ...)
  - Aktuell in Transitionsphase...
- Modul behandelt beide, mehr Fokus SwiftUI 😊

# SwiftUI

- Neuer deklarativer UI-Syntax
  - Seit Xcode 11 / **ab iOS 13**
- (Fast) alles passiert im Code...
  - Preview-Views
- Siehe Intro die letzten beiden Wochen! :-)

<https://developer.apple.com/xcode/swiftui/>

Programmieren fürs iOS - 5. UIKit: Interface Builder, ViewControllers & SwiftUI



## SwiftUI

```
import SwiftUI
```

```
struct ContentView: View {
```

```
    @State var start: Date = Date()
```

```
    @State var laps: Int = 0
```

```
    var body: some View {
```

```
        Group {
```

```
            ResetView()
```

```
            LapButton()
```

```
        }
    }
```

```
    var lapsSection: Section {
```

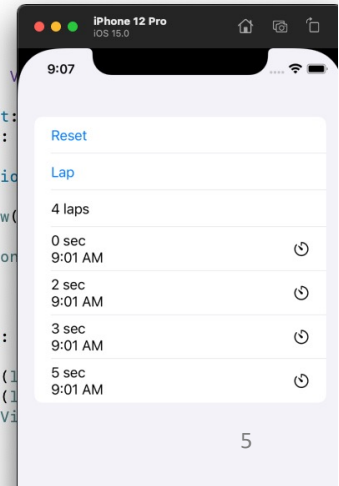
```
        Group {
```

```
            Text("\(laps) laps")
```

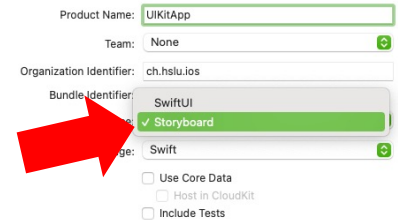
```
            ForEach(1...laps, id: \.laps) { lap, index in
```

```
                RowView(lap, index)
```

```
            }
        }
    }
```



# Heute: UIKit / Storyboard



- Massive Auswirkungen in Bezug auf Erzeugung & Verwendung von GUI
- Wir tauchen ein in die "klassische" UIKit-Welt mit Storyboards, ViewControllers, usw.

Zwischenfazit: Die iOS-App-Welt wird umfangreicher & komplizierter!..

# Kursinhalt (grob)

1. 19.09.: Einführung, Swift Crash Course & Xcode
2. 26.09.: SwiftUI Basics, Layouting, App-Provisioning
3. 03.09.: SwiftUI: States, Bindings, Navigation
4. 10.10.: Kommunikation & Nebenläufigkeit
5. 17.10.: UIKit, ViewControllers, UIKit vs. SwiftUI
6. 24.10.: Fragmentierung, mobile Usability, Widgets
7. 31.10.: Persistenz & Unit-Tests, Property Wrappers
8. 07.11.: Memory-Management, Frameworks



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

# Intro Interface Builder



# Interface Builder vs. Coding

- Bisher: Views "programmieren"
  - Übung 1: 

```
let label = UILabel(frame: frame)  
self.view.addSubview(label)
```

  
...



- Neu: Views "zeichnen"
  - Tool: Interface Builder
    - In Xcode integriert (seit Xcode 4.0)
    - Eng verknüpft mit UIKit
      - **NICHT vorgesehen für SwiftUI**

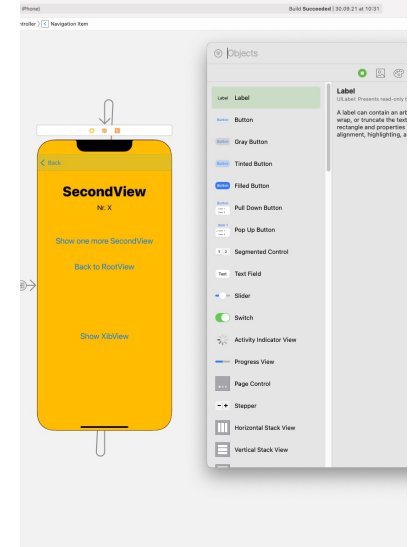
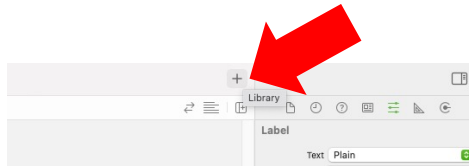


# Interface Builder

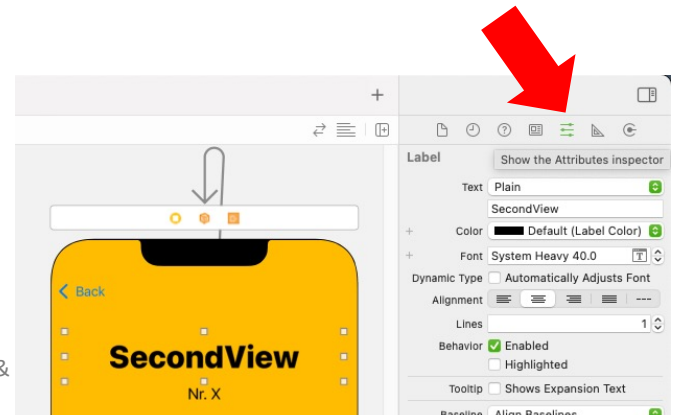
- Interface Builder = Visuelles Tool für GUIs
  - Eng verbunden mit Xcode (liest "automatisch" bestimmte Dinge aus Code, aus einzelnen Klassen)
  - Automatisches Ausrichten: "Apple-Look'n'Feel"
  - Standard-Widgets: UILabel, UIButton, UISlider, ...
  - Dateiformat: .storyboard
    - Früher primär: .nib-Datei (**N**extStep **I**nterface **B**uilder)
      - resp. .xib (x für xml...)
      - Gibt's immer noch, funktioniert immer noch! (Siehe später)
- Relativ einfach zu bedienen, primär graphisch 😊
  - Drag'n'drop, Ziehen, usw.

# Demo: Interface Builder

- (Object) Library
  - Zeigt die im IB verfügbaren Objekte



- Attributes Inspector
  - Attribute inspizieren und setzen





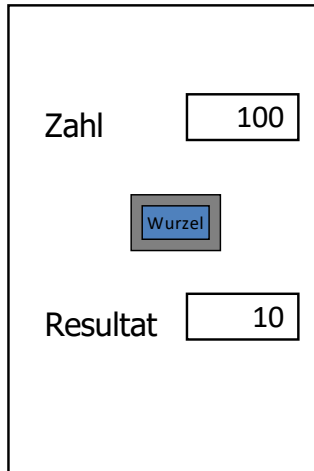
<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

# Interface Builder: Verbindung Layout – Code

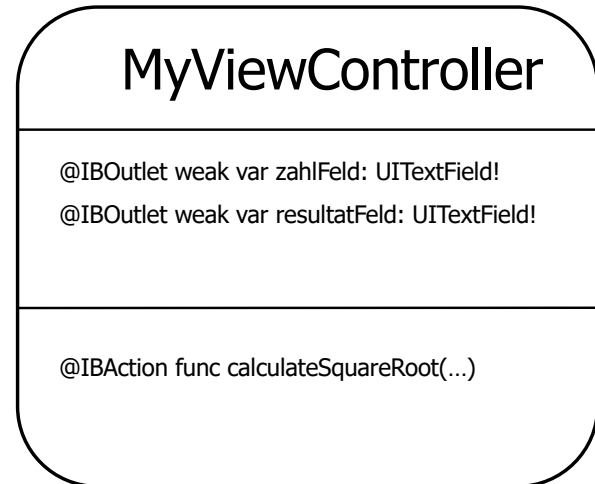
# Verbindung Layout-Code

- Wie Layout (.xib, resp. .storyboard) mit Code zusammenbringen?

\*.storyboard/\*.xib:



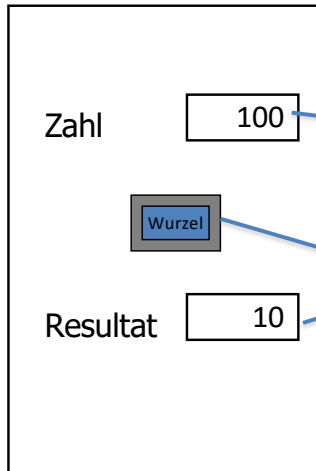
Objekt zur Laufzeit:



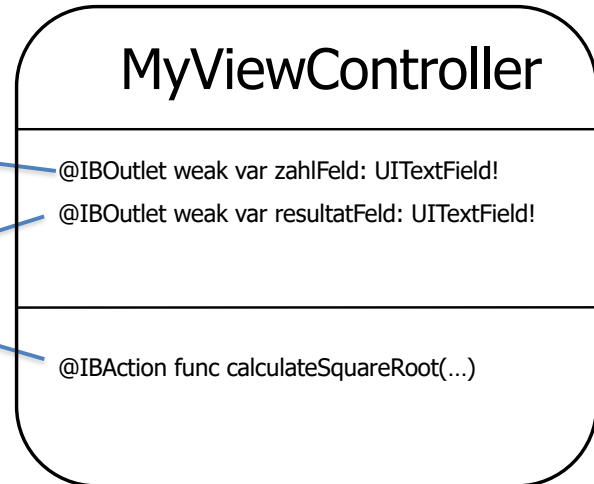
# Verbindung Layout-Code

- Wie Layout (.xib, resp. .storyboard) mit Code zusammenbringen?
- Schlüssel: **IBOutlet** & **IBAction**

\*.xib / \*.storyboard:



Objekt zur Laufzeit:



# IBOutlet

- IBOutlet verbinden Layout (.xib/.storyboard) mit Code
  - Verbindung ziehen (mit der Maus!) im Interface Builder
- Schlüsselwort "IBOutlet" bei der Deklaration
  - **@IBOutlet** weak var myLabel: UILabel!
- Enge Bindung zwischen IB und Code
  - IB merkt z.B. wenn in Xcode neue IBOutlets oder IBActions deklariert werden
  - "Mausverbindung" vom IB in den Code

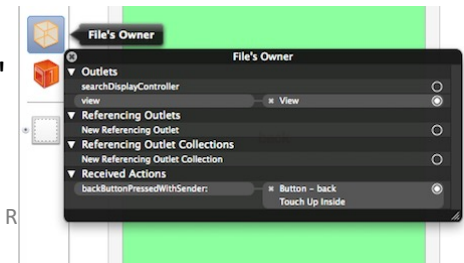
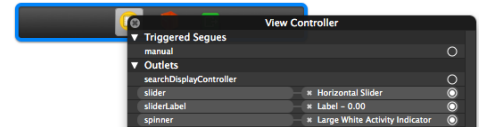
# Deklaration von IBOutlets

- Im Code (MyViewController.swift)
  - @IBOutlet weak var myLabel: UILabel!
- Verbindung zwischen Code und Layout mit der Maus (Methoden "Rechtsklick" oder "ctrl")
  - Falls nicht vorhanden, wird ggf. IBOutlet im Code erzeugt

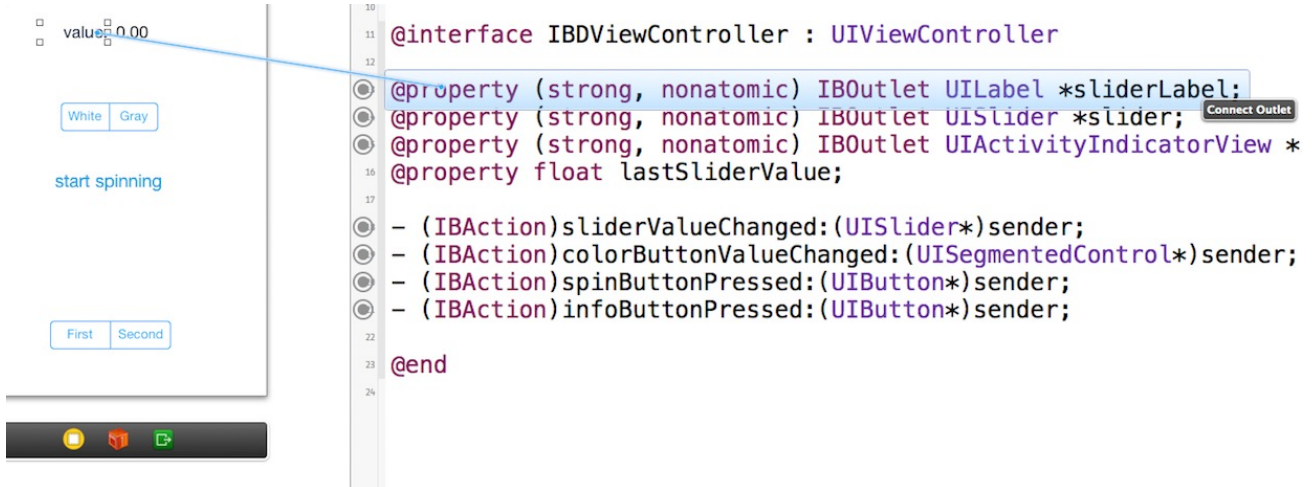


# Demo: IBOutlet

- Deklaration IBOutlet im Code
- Verschiedene Varianten Maus-Verbindung
  - Maus: Ctrl-Taste oder rechter Mausklick
  - Siehe nächste Folien...
- Referenz auf ViewController?
  - .storyboard: ViewController
  - .xib: File's Owner = Platzhalter für entspr. ViewController (mehr zu MVC später...)
    - d.h. +- : "File's Owner = MyViewContorller"



# Mit deklariertem Outlet verbinden



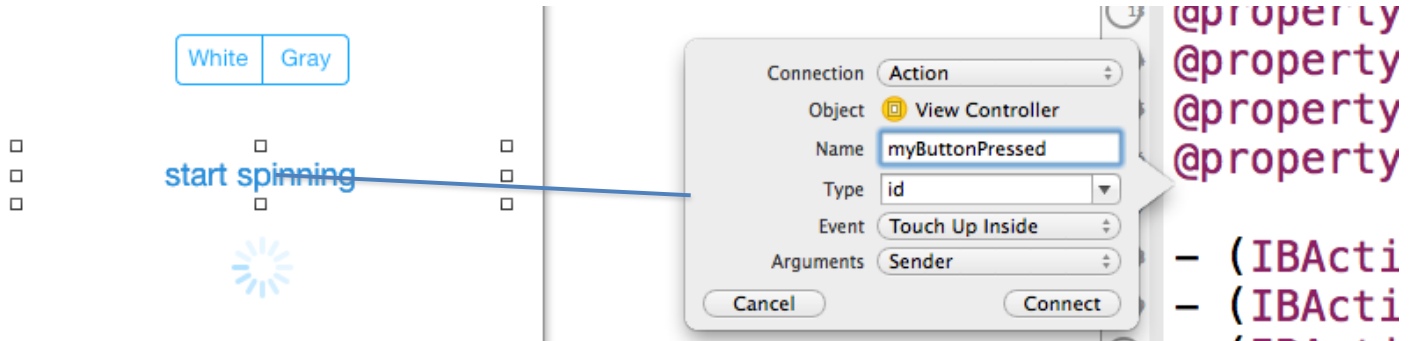
# IBAction

- Target/Action-Muster: Verbindet "Action" mit einem Ziel
  - "Action" z.B. Knopfdruck, oder neuer Wert (Methode)
  - Ziel ist ein Objekt, welches die Nachricht erhält
    - typischerweise ein ViewController (mehr dazu nächste Woche)
- Syntax (MyViewController.swift)
  - @IBAction func sliderValueChanged(sender: UISlider)
  - Konvention: Exakt ein Argument "sender"
    - "sender" = Auslöser der Action
- Verbindung Layout-Code mit "Maus" analog zu IBOutlet
  - Auch programmatisch möglich
    - Methode addTarget:action:forControlEvents: von UIControl

# Demo: IBAction

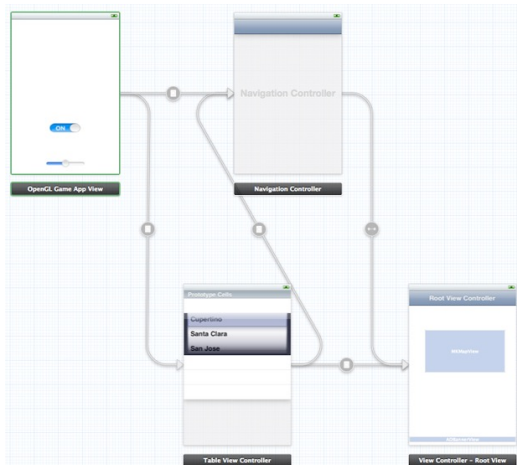
- Deklaration IBAction im Code
- Verschiedene Varianten "Maus-Verbindung" wie bei IBOutlet
- Target/Action live...
  - z.B. Action auf Knopfdruck

# IBAction-Methode generieren



# Ausblick: Storyboards

- Mehrere/alle Views in einer Datei, inkl. Modellierung der Übergänge: Storyboard – behandeln wir gleich vertiefter



Source: <http://developer.apple.com/>

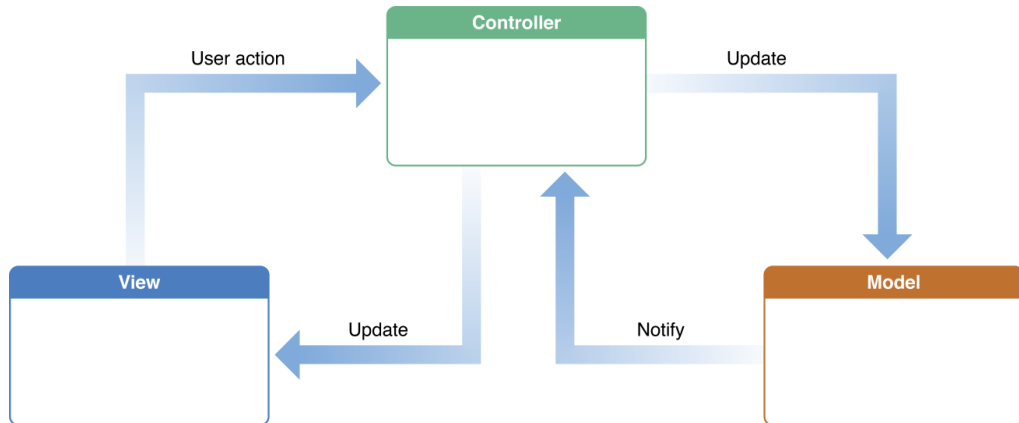


<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

# UIKit: View- Controller

# MVC: View Controller

- Model-View-Controller Design-Muster
  - Model: "Daten"
  - View: "Ansicht"
  - Controller: "Vermittler"



Source: <http://developer.apple.com/>

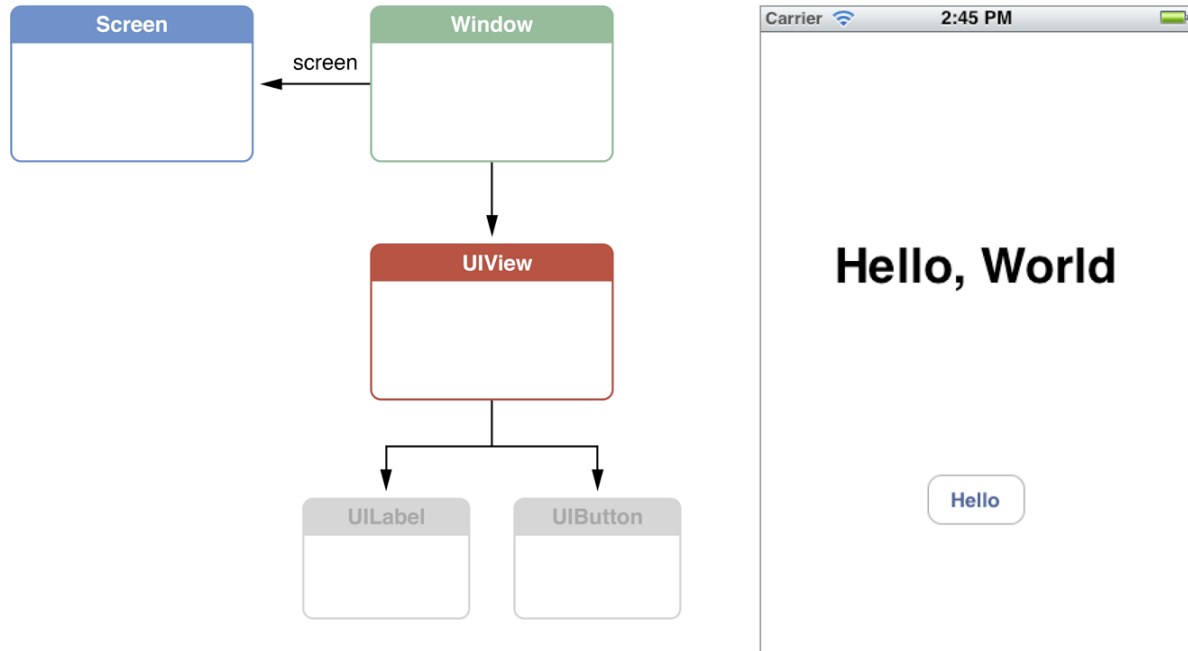


# ViewControllers sind bei UIKit zentral wichtig...

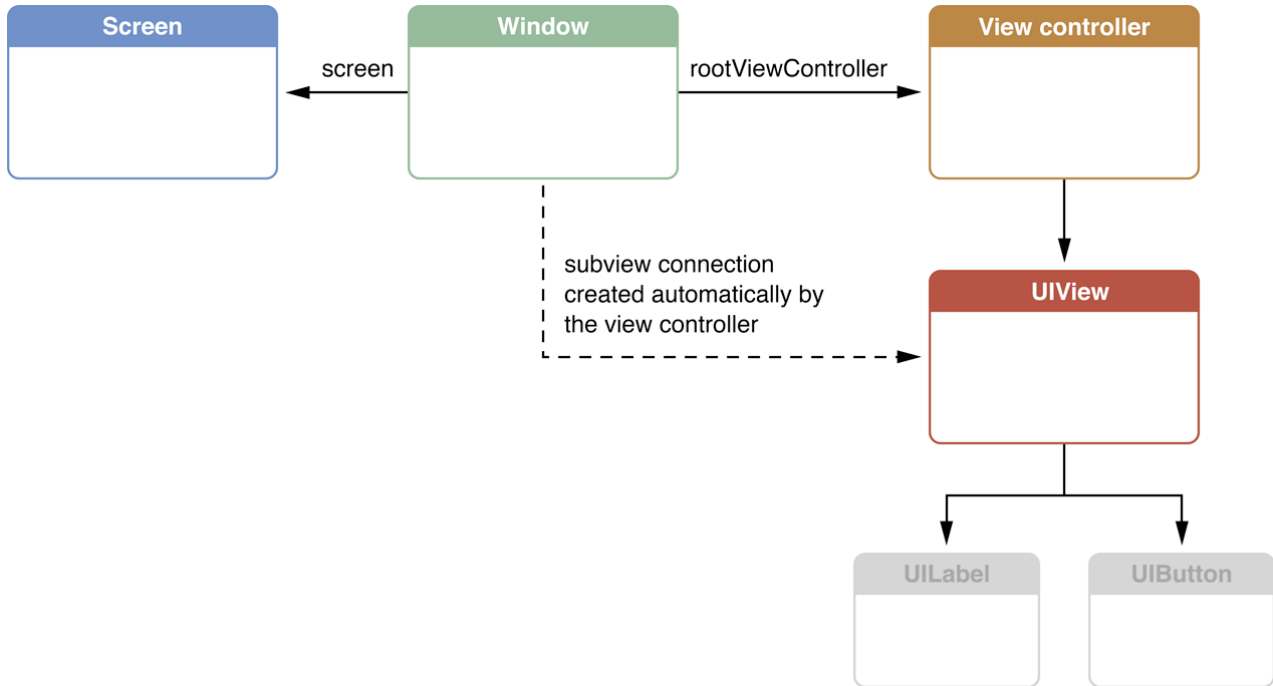
View controllers are the foundation of your app's internal structure. Every app has at least one view controller, and most apps have several. Each view controller manages a portion of your app's user interface as well as the interactions between that interface and the underlying data. View controllers also facilitate transitions between different parts of your user interface.

[https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/index.html#//apple\\_ref/doc/uid/TP40007457-CH2-SW1](https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/index.html#//apple_ref/doc/uid/TP40007457-CH2-SW1)

# Screen, Window & Views



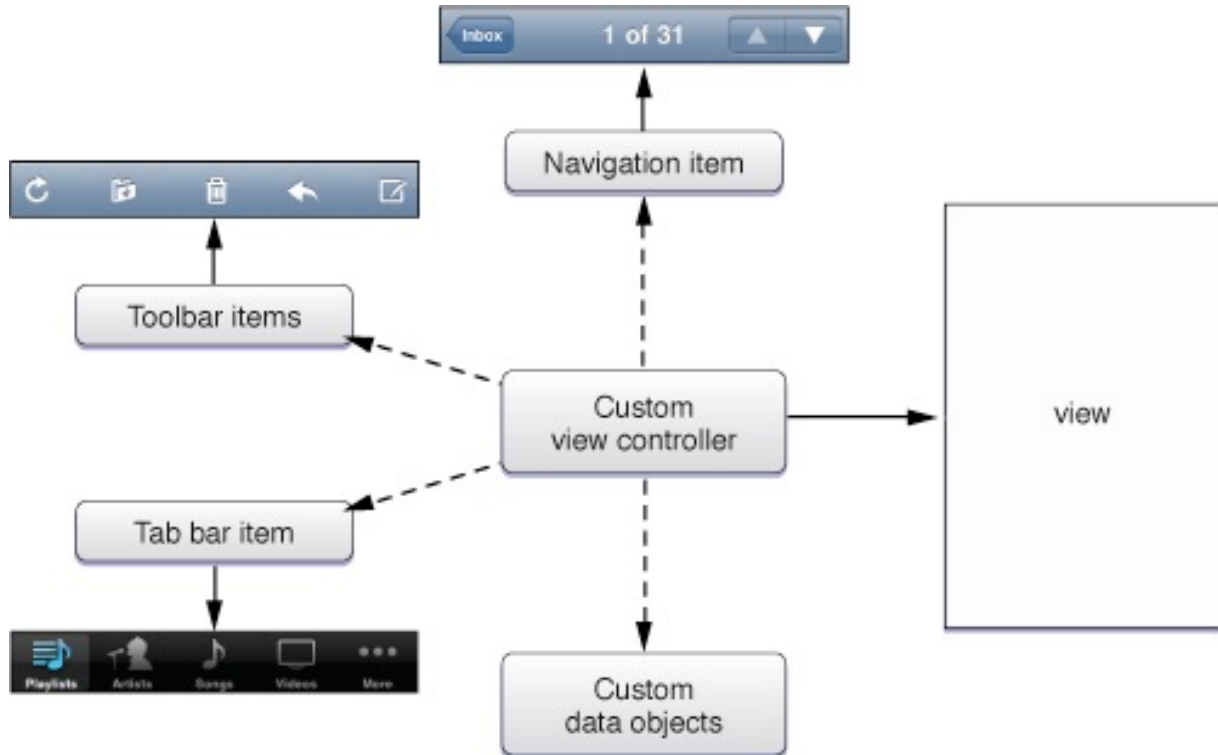
# View Controllers manage Views



# UIViewController

- "DIE" View-Controller-Klasse
  - Controller einer "Bildschirmseite" in iOS (UIKit)
- Basisklasse für alle iOS-ViewController
- Funktionalität für: Modal-View, Navigation, ...

# Anatomie eines View Controllers



# UIViewController:

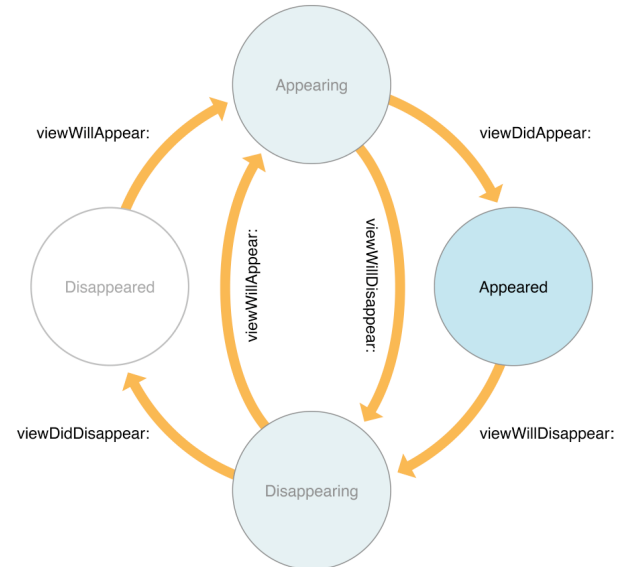
## einige Properties...

- `nibName`
  - kann nil sein (default: nib und Controller heissen gleich)
- `view`
  - DIE view, welcher dieser Controller steuert
- `title`
  - wichtig bei Navigation: Default für Titel NavigationBar & Back-Button
- Für ModalViews
  - `modalViewController`
  - `modalTransitionStyle`
- Für Navigation & Tabs
  - `navigationController`, `navigationItem`, `toolbarItems`, `tabBarController`, `tabBarItem`

# UIViewController:

## einige Methoden...

- initWithNibName:bundle:
  - Initialisierung aus einer Nib-Datei
  - Werden wir kaum brauchen: passiert automatisch, wenn nib und Controller gleich heissen
- loadView
  - 3 Optionen, siehe nächste Folie
  - Danach wird viewDidLoad aufgerufen
- "Vor- und Nachwarnungen" wenn View angezeigt wird / verschwindet
  - viewWillAppear: / viewDidAppear:
  - viewWillDisappear: / viewDidDisappear:
- Speicherwarnung
  - didReceiveMemoryWarning





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

# View- Controller: Content vs. Container

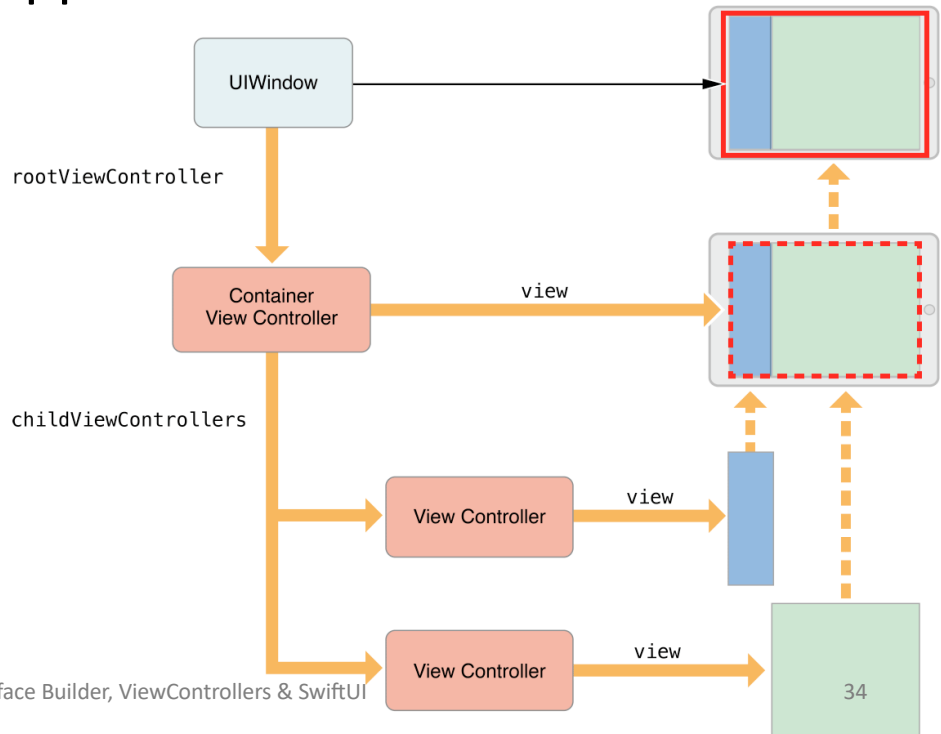


# Content & Container Controller

- Es gibt zwei Arten von View Controllers
  - Content View Controller
    - Stellen Inhalt dar: "Views"
    - z.B.: UIViewController, UITableViewController
  - Container View Controllers
    - Arrangieren Inhalt von anderen ViewControllers (typischerweise Content ViewControllers)
    - z.B.: UINavigationController, UITabViewController, UISplitViewController, UIPageViewController, UIPopoverController

# Container View Controller

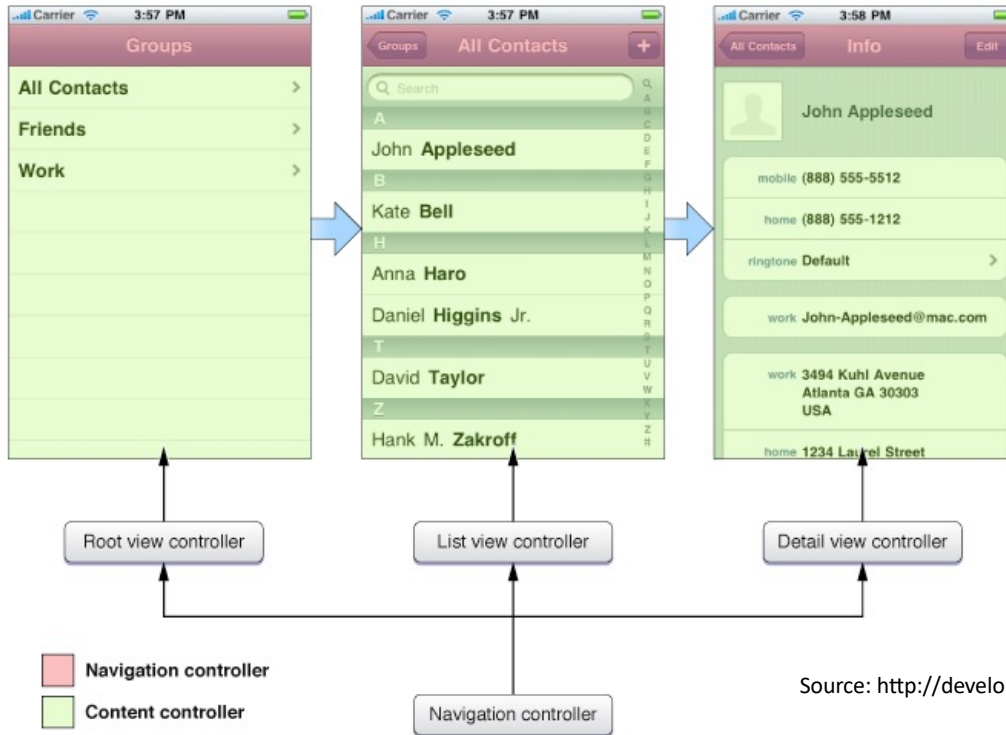
- Bild aus der Apple-Doku



# UITableViewController

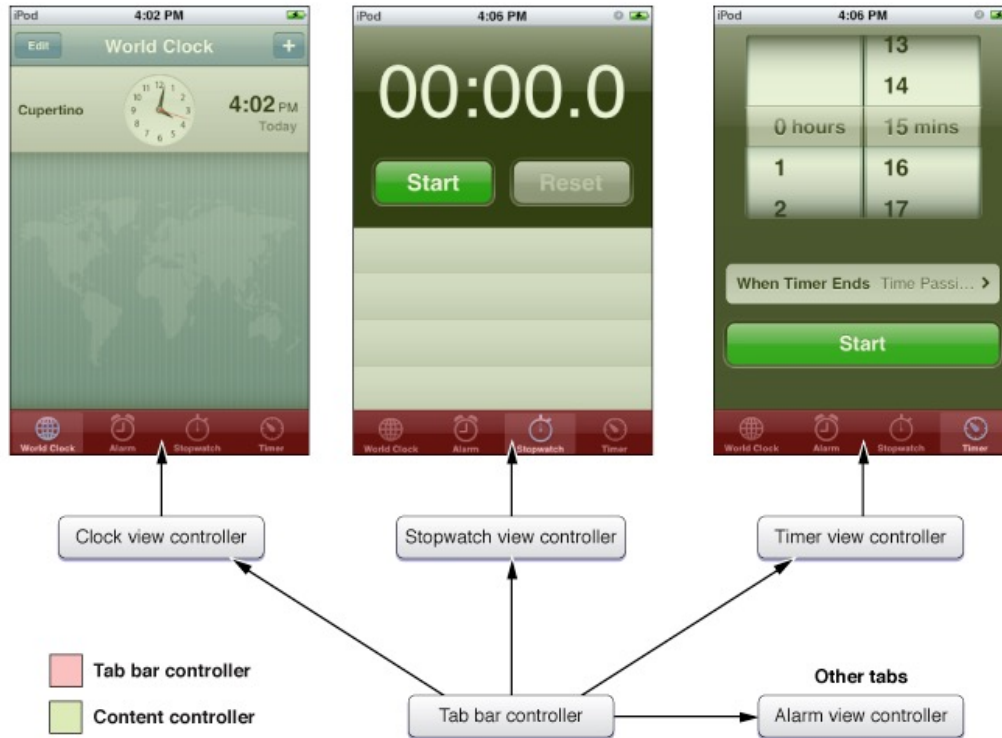


# UINavigationController



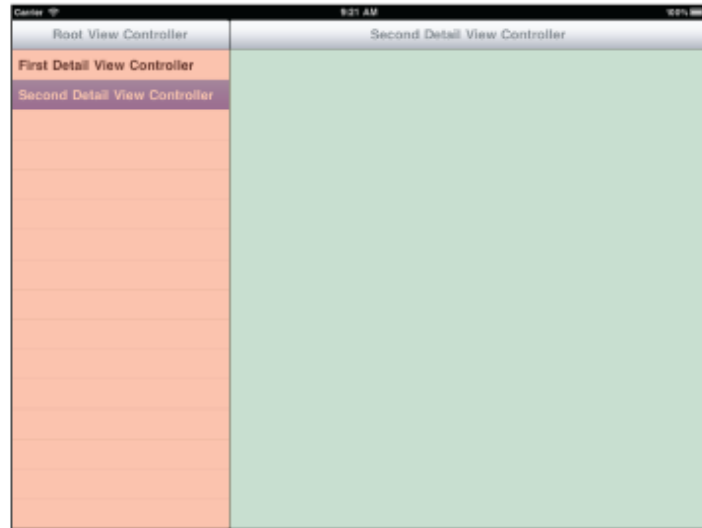
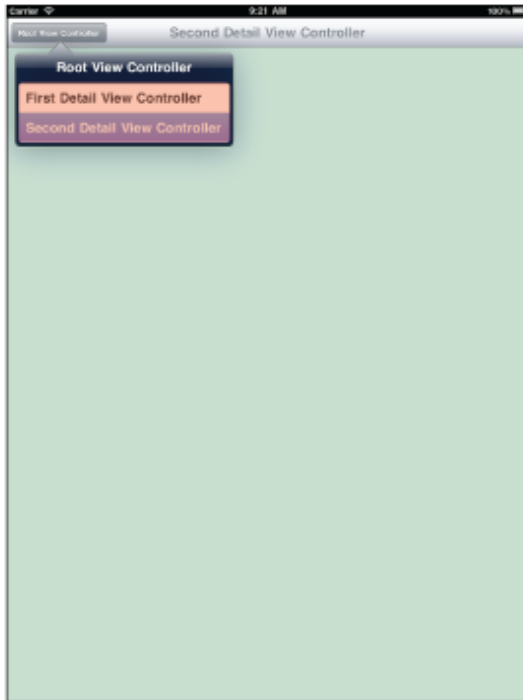
Source: <http://developer.apple.com/>

# UITabBarController



Source: <http://developer.apple.com/>

# UISplitViewController (iPad)



**Master controller**

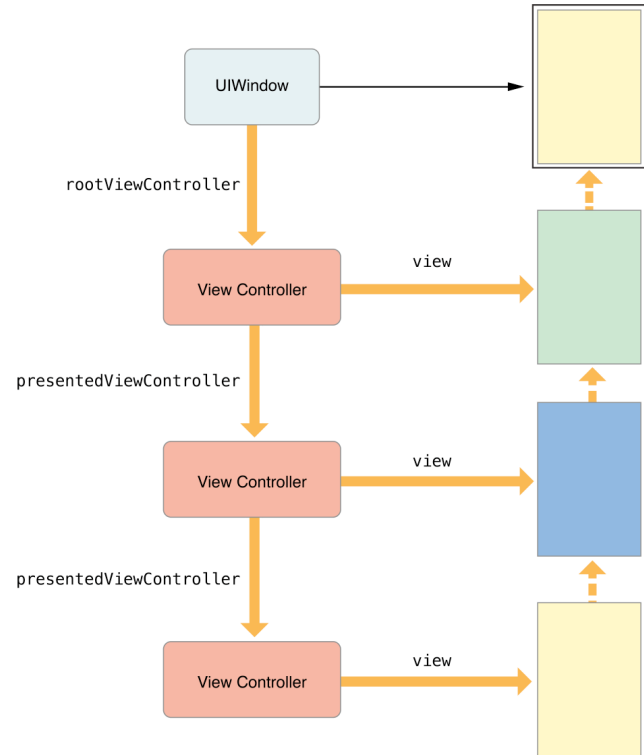


**Detail controller**

Source: <http://developer.apple.com/>

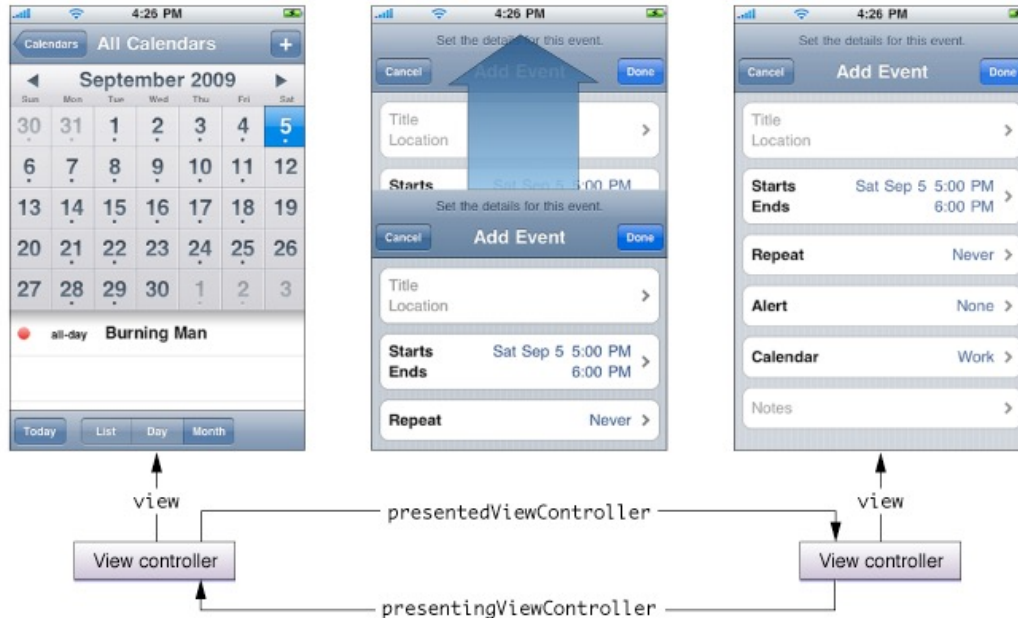
# Modal View Controller

- Teil der Klasse UINavigationController
  - d.h. UINavigationController kann auch eine Art "Container View Controller" sein und andere Views (resp. ViewController) modal präsentieren



# Modal View Controller

- Altes Bild aus der Apple-Doku:





# Demo XibViewController

- Knopf "Show XibView" -> IBAction-Methode

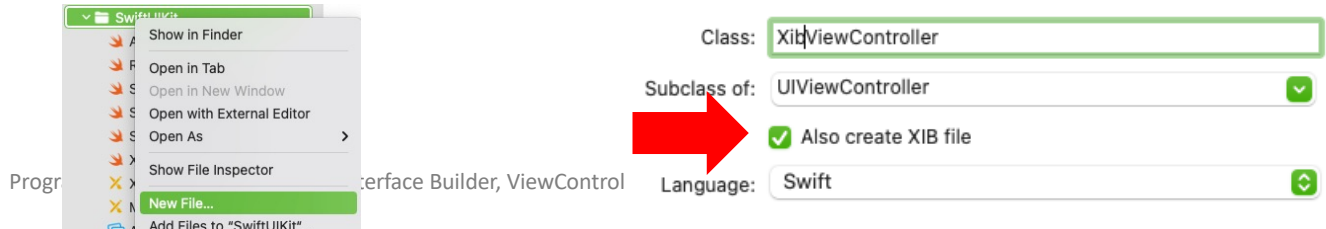
- UIViewController.present(...)

```
self.present(XibViewController(), animated: true, completion: nil)
```

- Klasse XibViewController generieren lassen

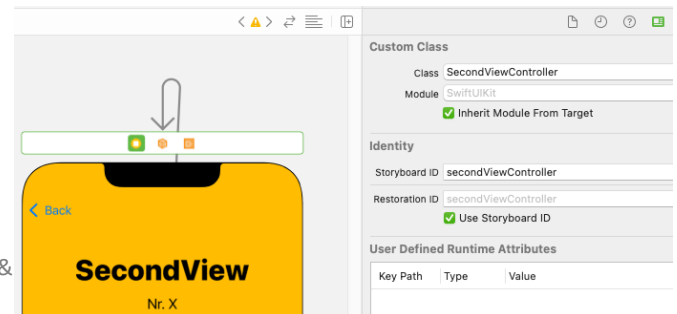
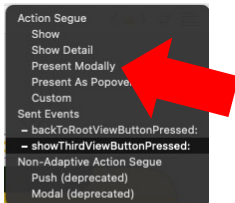
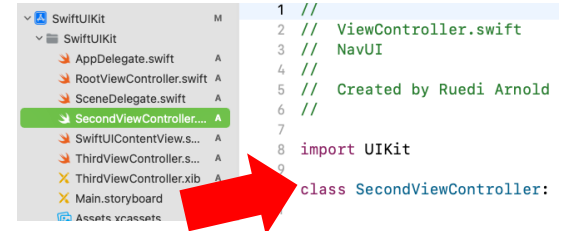
- Unterklasse von UIViewController

- Häcken "Also create XIB file" unter "New File..." –  
"iOS : Cocoa Touch Class"



# Demo modaler SecondVC

- Storyboard mit SecondViewController
  - Subklasse von UIViewController
- Recap IBOutlet
- Knopf mit IBAction
  - Segue: Present Modally





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

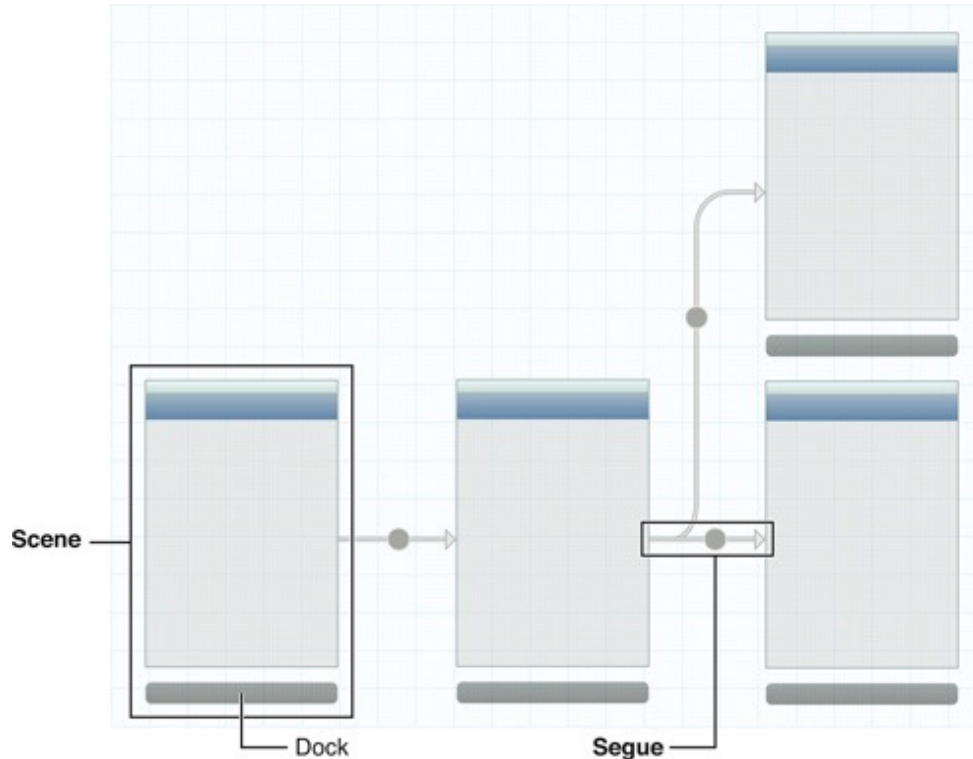
# Storyboard

# Storyboard: Text...

A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views; scenes are connected by segue objects, which represent a transition between two view controllers.

<https://developer.apple.com/library/ios/#documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html>

# ...und Bild dazu ("Storyboard II")



<https://developer.apple.com/library/ios/documentation/general/conceptual/Devpedia-CocoaApp/Storyboard.html>

# "Scene = 1 Screen"

- iPhone: "1 Szene = 1 voller Bildschirm"
  - "each scene corresponds to a full screen's worth of content"
  - iPad: multiple scenes can appear on screen at once - for example, using popover view controllers
- Hinter jeder Szene steht ein ViewController

# Segue = der Übergang

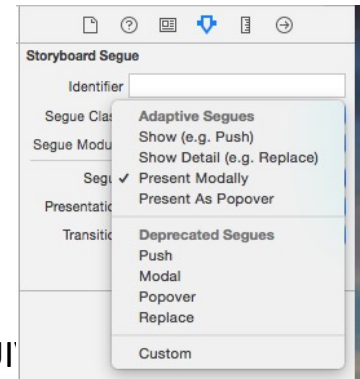
- Ausgesprochen wie "Segway"
  - Hier keine Scooter für Menschen... sondern Übergänge für Szenen! ;-)



<http://www.segway-point.ch/>

# UIStoryboardSegue

- Segue = Übergang zwischen zwei Szenen
  - A UIStoryboardSegue object is responsible for performing the visual transition between two view controllers
  - Verschiedene "Styles", siehe nächste Folie...
- 2 Methoden
  - (id)initWithIdentifier:(NSString \*)identifier source:(UIViewController \*)source destination:(UIViewController \*)destination
  - (void)perform





# Styles von Segues

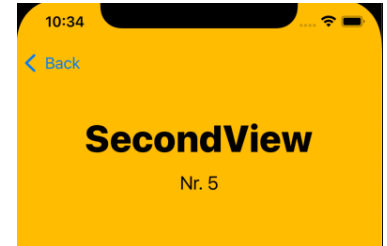
- Show (Push): Modal oder Push
- Show Detail (Replace): innerhalb von SplitViewController (sonst modal)
- Present Modally: Modaler "Overlay"
- Present as Popover: selbstsprechend...

# UIViewController & Segues

- UIViewController: 2 Methoden im Zusammenhang mit Segues:
  - (void)prepareForSegue:(UIStoryboardSegue \*)segue sender:(id)sender
    - Vor Segue-Ausführung, z.B. um Daten zu übergeben 😊
  - (void)performSegueWithIdentifier:(NSString \*)identifier sender:(id)sender
    - Übergang programmatisch auslösen
    - Verknüpfung typischerweise im Interface Builder (Widget -> Action)
- Wird typischerweise vom System / Storyboard aufgerufen

# Demo Storyboard & Segue

- SecondViewController
  - Subklasse von UIViewController
- Datenübergabe in von MainView zu SecondView in UIViewController-Methode performSegueWithIdentifier:sender:
  - Property auf VC-Klasse



```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.destination is SecondViewController {  
        let secondVC = segue.destination as! SecondViewController  
        secondVC.number = self.number + 1  
    }  
}
```



# Swift: Typcheck (is) & Casting (as)

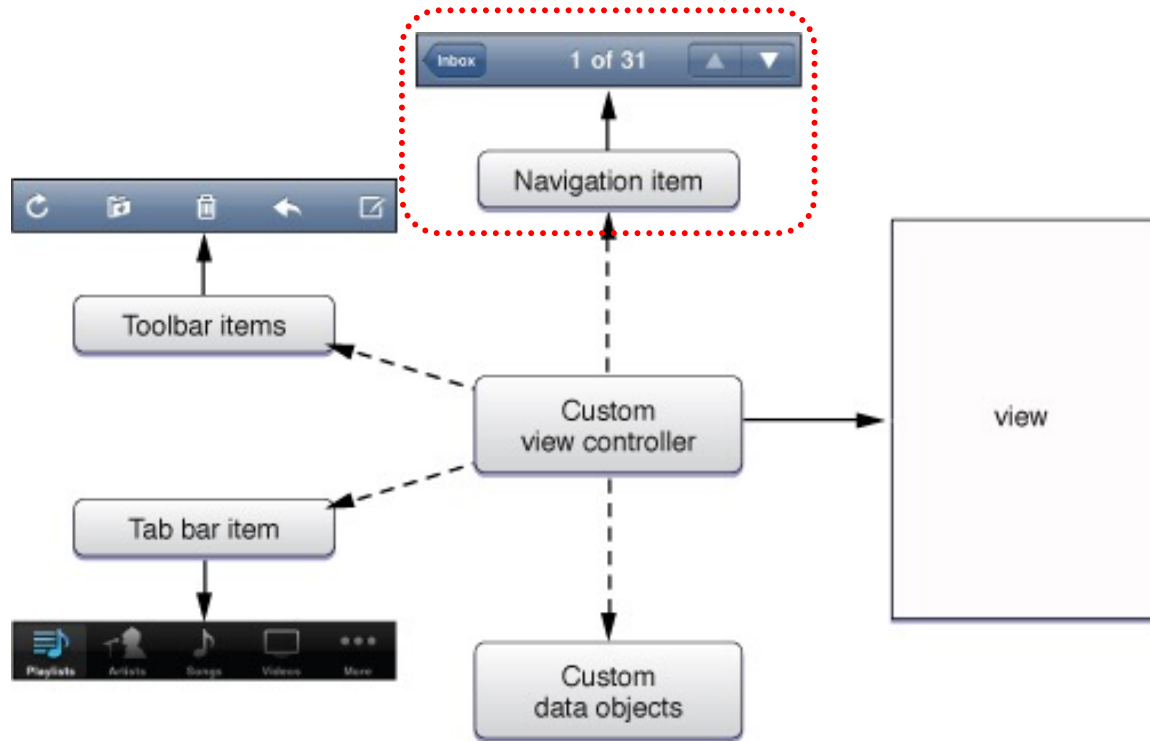
- **is: Test auf Typ** (Klasse, Protokoll, Enum, Struct)
  - Bsp.: `if anyObject is String { ...`
- **as: Cast-Operator**
  - Bsp.: `var x = anyObject as String`
  - Zwei Varianten: `as!` und `as?`
    - **as!:** erzwingt Cast, (falls nicht möglich: Laufzeit-Fehler)
    - **as?:** Liefert nil zurück, falls Cast nicht möglich



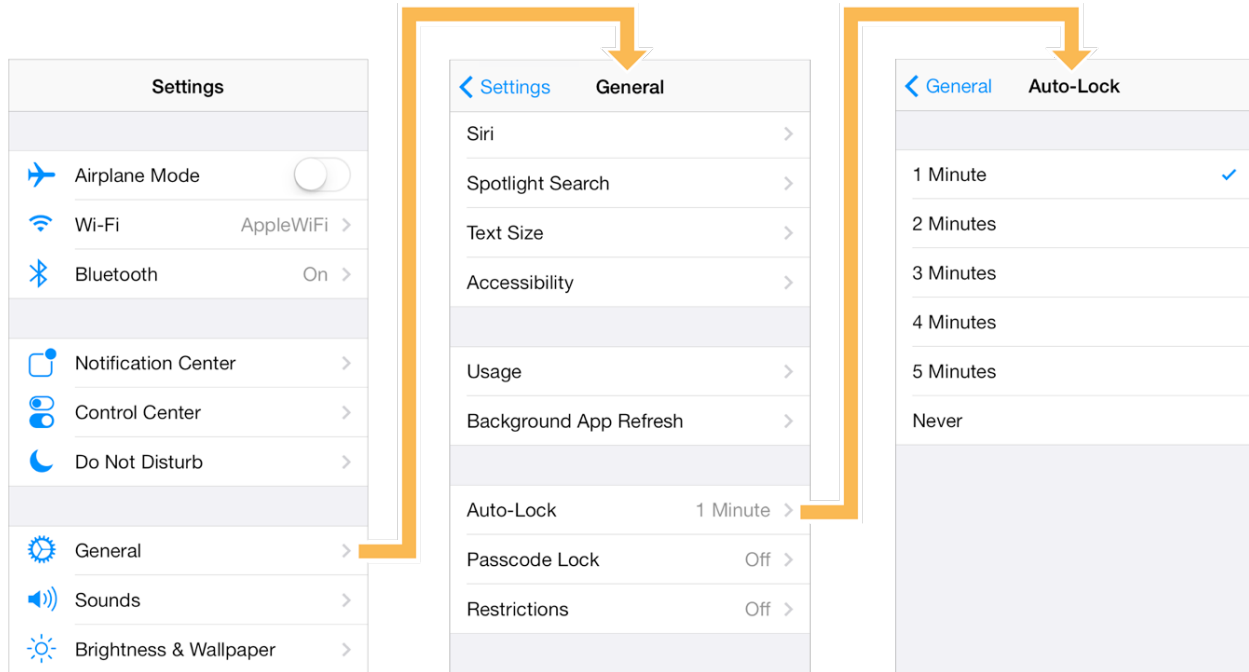
⌘ <http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

# UINavigationController

# Anatomie eines View Controllers



# Bsp. Navigation



# UINavigationController

- Container View Controller
- Verwaltet Hierarchie von Views
  - Hat einen Root-View-Controller
- Viele "praktische" Defaults
  - Navigation-Bar
  - Back-Button
- Wichtigste Methoden
  - `pushViewController:animated:`
  - `popViewControllerAnimated:`
  - `show:`

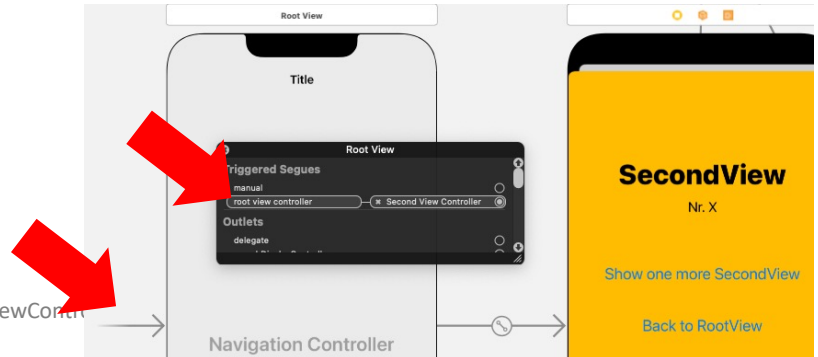
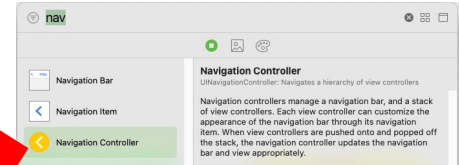


# Ein Stapel von Views...

- Grundidee
  - **Push:** Neue View (d.h. entsprechender ViewController) wird auf einen Navigation Controller gedrückt
    - `pushViewController:animated:`
  - **Pop:** Alte View (d.h. entsprechender ViewController) wird wieder von einem Navigation Controller entfernt
    - 3 Varianten:
      - `popViewControllerAnimated:`
      - `popToRootViewControllerAnimated:`
      - `popToViewController:animated:`

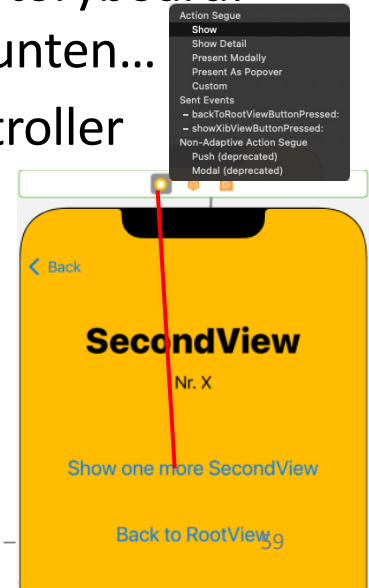
# Demo: Einbau UINavigationController

1. Drag'n'Drop auf Object Library
2. Auf NavCon RootViewController setzen (mit ctrl-Maus)
3. StartPfeil auf RootView setzen

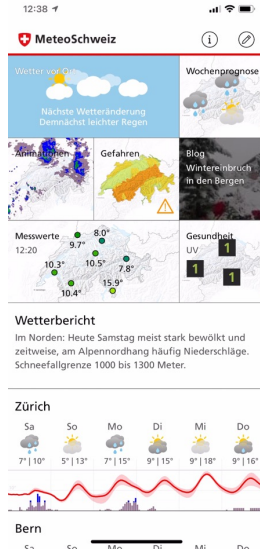


# Demo UINavigationController

- SecondViewController mit Knopf "Show one more SecondView"
  - Verbindung Knopf – Segue Action im Storyboard: siehe rote Linie im Screenshot rechts unten...
  - Show (= Push) neuer SecondViewController
- Knopf "Back to RootView"
  - Eigene IBAction-Methode in SecondViewController-Klasse: Aufruf `VON navigationController.popToRootViewController(...)`



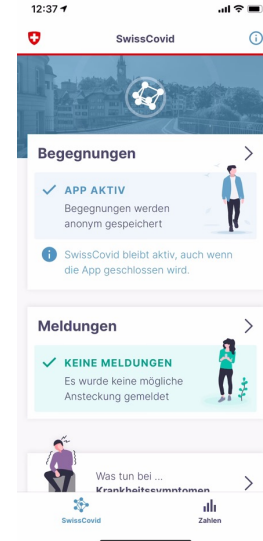
# Verschiedene Konzepte für verschiedene Anwendungen 😊



Navigation



Modal




Tabs



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>  
& SwiftUI

# Interoperabilität UIKit - SwiftUI

# UIKit & SwiftUI

- Aktuell also zwei UI-Technologien für iOS
  - Q: Lassen sich UIKit und SwiftUI kombinieren?
  - A: Ja, in beide Richtungen!
    - Wir schauen je einen einfachen Fall an (ohne komplizierte Navigation, Datenübergabe usw.)
    - Beide Richtungen potentiell sinnvoll
-  Frage in die Runde: Typische Szenarien?

# SwiftUI in einem UIViewController

- Klasse UIHostingController
  - Unterklasse von UIViewController
  - `init(rootView: Content)`: "Creates a hosting controller object that wraps the specified SwiftUI view."
- Wichtig: SwiftUI-Framework importieren
  - `import SwiftUI`

## Summary

A UIKit view controller that manages a SwiftUI view hierarchy.

## Declaration

```
class UIHostingController<Content> where Content : View
```

## Discussion

Create a `UIHostingController` object when you want to integrate SwiftUI views into a UIKit view hierarchy. At creation time, specify the SwiftUI view you want to use as the root view for this view controller; you can change that view later using the `rootView` property. Use the hosting controller like you would any other view controller, by presenting it or embedding it as a child view controller in your interface.

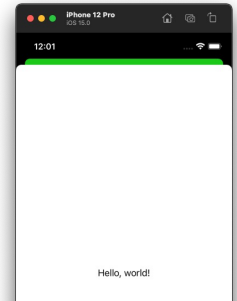
# Demo: UIHostingController

- SwiftUI-View erzeugen
  - z.B. copy/paste ContentView von SwiftUI-Projekt-Template
- Eigene @IBAction
  - HoistingController erzeugen und (modal) anzeigen

```
let swiftUIVC = UIHostingController(rootView: ContentView())
self.present(swiftUIVC, animated: true)
```

- Show (push) ginge natürlich auch, siehe Übung 😊

<https://developer.apple.com/documentation/swiftui/uihostingcontroller>



```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, world!")
                .padding()
        }
    }
}
```



#### Summary

A view that represents a UIKit view controller.

#### Declaration

```
protocol UIViewControllerRepresentable :  
Never
```

# UIViewController in SwiftUI

- UIViewController aus SwiftUI anzeigen:  
UIViewControllerRepresentable = Protokoll  
mit zwei relevante Funktionen:

```
func makeUIView(context: Self.Context) -> Self.UIViewType
```

Creates the view object and configures its initial state.

Required.

```
func updateUIView(Self.UIViewType, context: Self.Context)
```

Updates the state of the specified view with new information from SwiftUI.

Required.

<https://developer.apple.com/documentation/swiftui/uiviewrepresentable>

# Demo: UIViewControllerRepresentable

- Eigene struct erzeugen, welche gewünschte Subklasse von UIViewController zurück gibt:

```
struct UIKitVC: UIViewControllerRepresentable {  
    func updateUIViewController(_ viewController: XibViewController,  
        // not used in this simple example.  
    )  
    func makeUIViewController(context: Context) -> XibViewController {  
        return XibViewController()  
    }  
}
```

- Und dann z.B.  
modal anzeigen:
- ```
Button("Show XibView") {  
    modal = true  
}.sheet(isPresented: $modal,  
    onDismiss: { modal = false },  
    content: { UIKitVC() })
```



& <http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>  
SwiftUI

# Kurs-Ausblick & Übung 5

# Kurs-Ausblick

- SW06: Fragmentierung, mobile Usability, Widgets
- SW07: Persistenz, Testen, Property Wrappers
- SW08: Memory Management, Frameworks
- SW10-14: Teamprojekt (+ ggf. Gastvorträge)
  - Zu zweit je eine eigene App umsetzen!
  - Idee - Papier-Prototyp - Umsetzung - Präsentation - Demo 😊

# Zwischenfazit nach 5 Wochen...

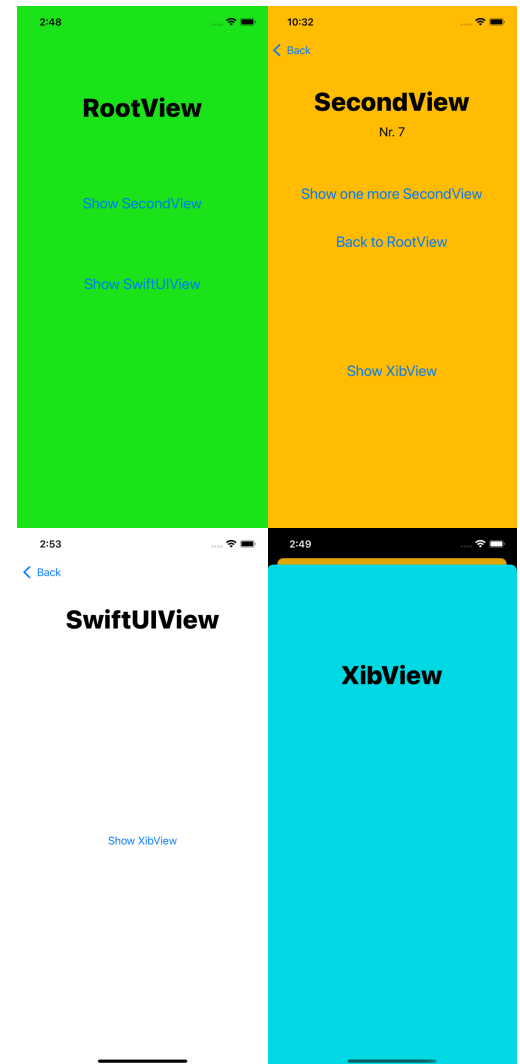
- Fragen / Anmerkungen?
- Inhalt soweit ok?
- Modul-Modus inkl. Präsenzunterricht passt?
- Übungen?
- Sonstwas?..

Thx4feedback... spontan jetzt oder später!

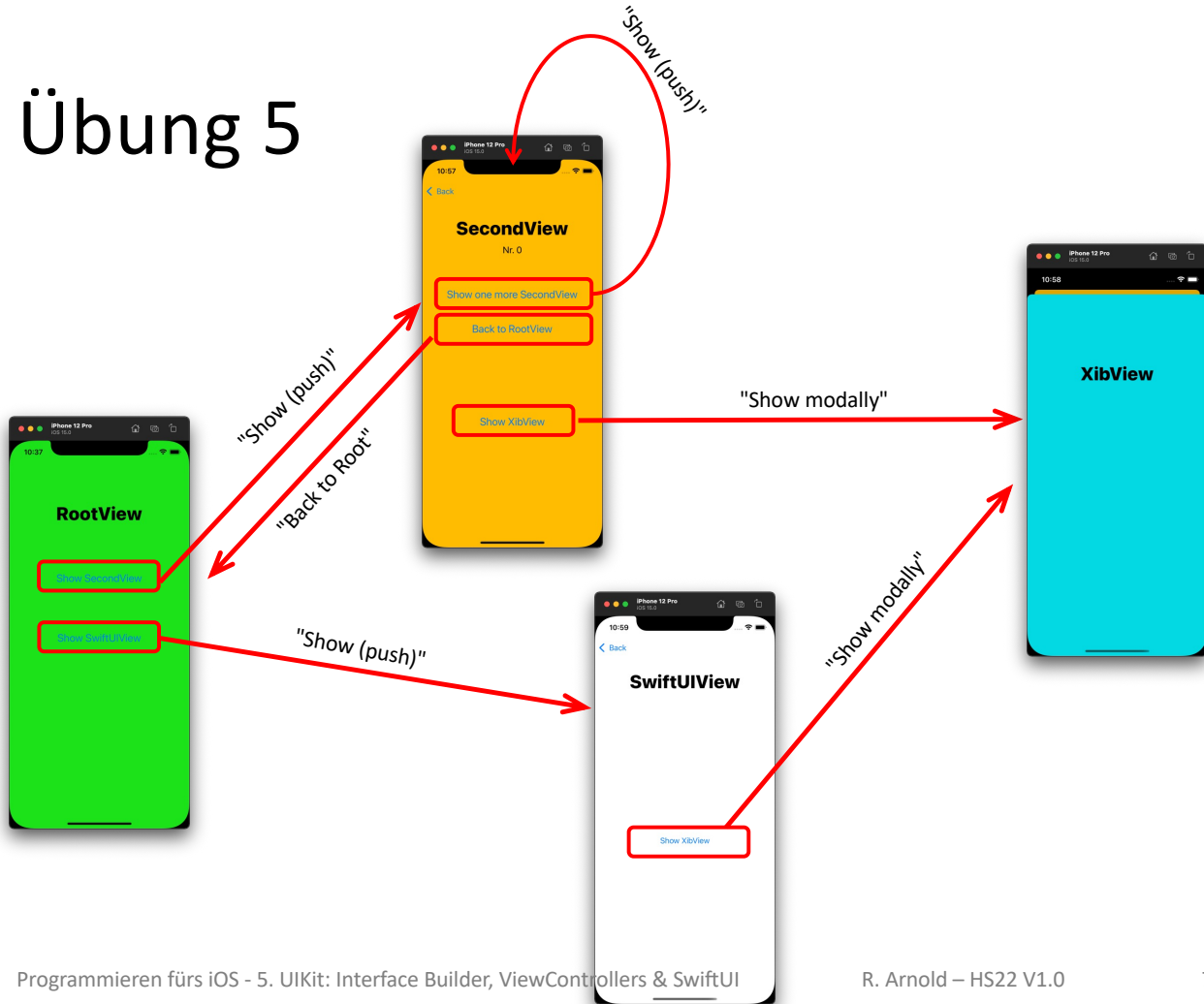
...dieses Modul findet  
zum 11. Mal statt und wir  
geben uns dabei Mühe und  
möchten es gerne weiter verbessern! 😊

# Übung 5

- Interface Builder & Widgets
  - IBOutlets & IBAction
- Mehrere ViewControllers
  - mit .xib, resp. im .storyboard
- Übergänge
  - Modal & Show (Push)
  - Programmatisch & im Storyboard
- Interop SwiftUI
  - UIViewController@SwiftUI
  - SwiftUI@ViewController



# Übung 5



# Ü5: Setup Storyboard



- Dazu je 1 Scene aus xib- und SwiftUI-Datei 😊