Beta 2.0

Full Stack iOS Entwicklung mit Swift

WPF im MIM - WS 17/18

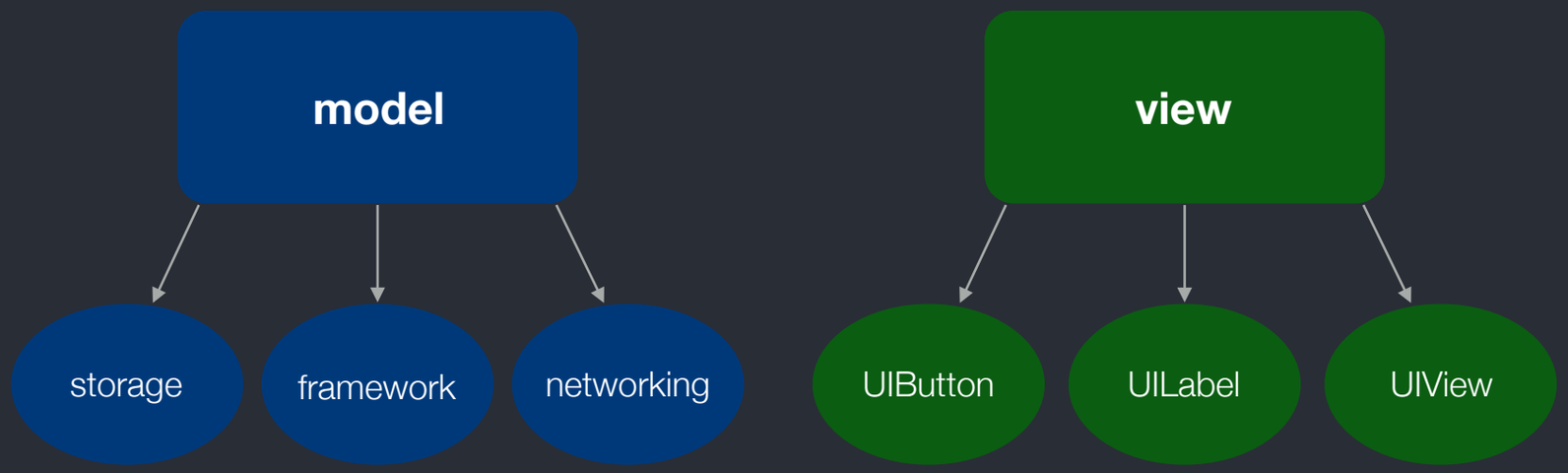
Alexander Dobrynin, M.Sc.

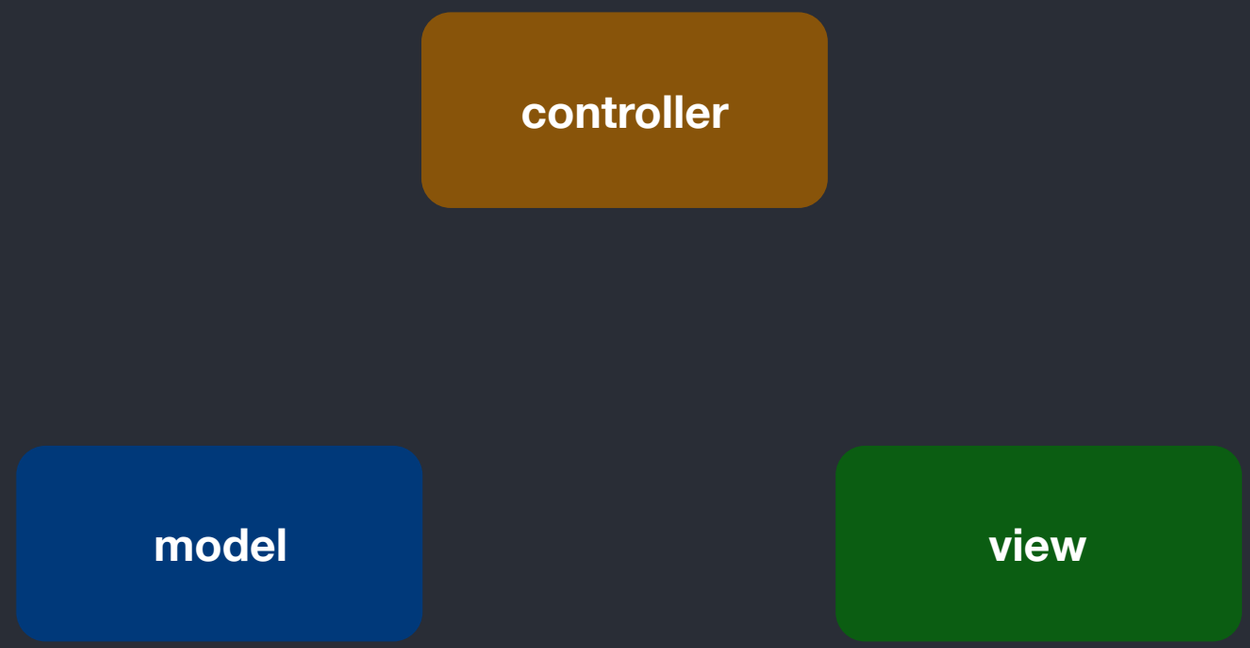
Heute

Model-View-Controller (MVC)
DataSource, Delegate, Target-Action, Notification

Demo

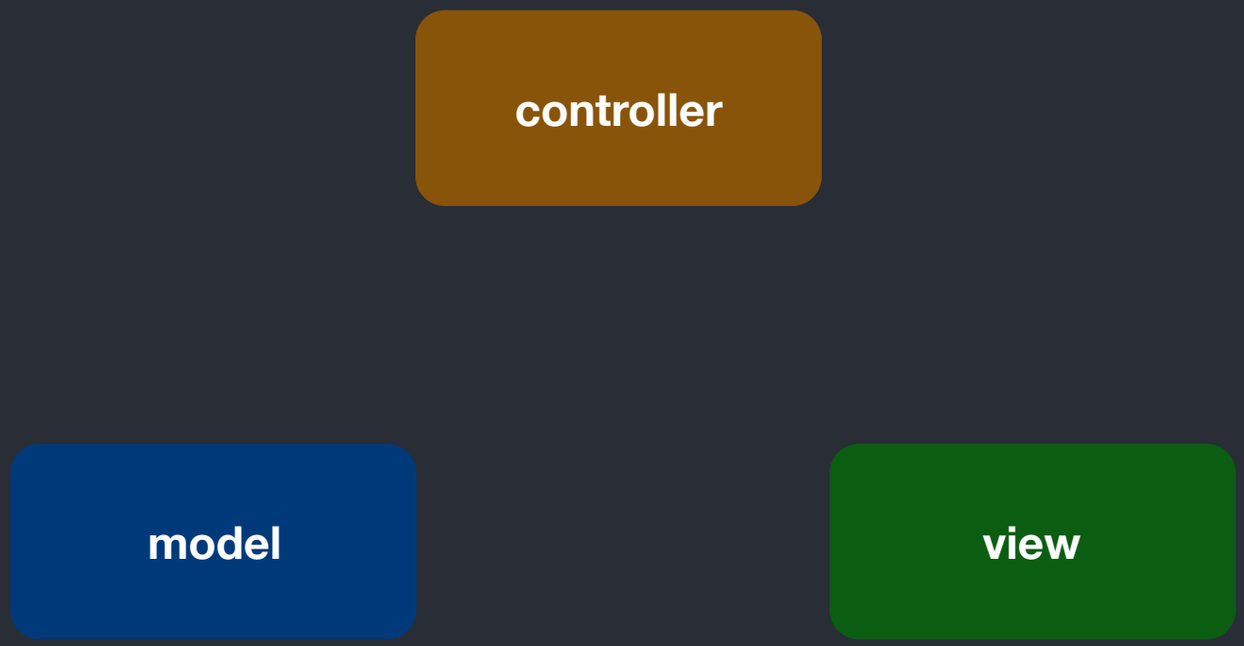
controller


model
storage
framework
networking
view
UIButton
UILabel
UIView

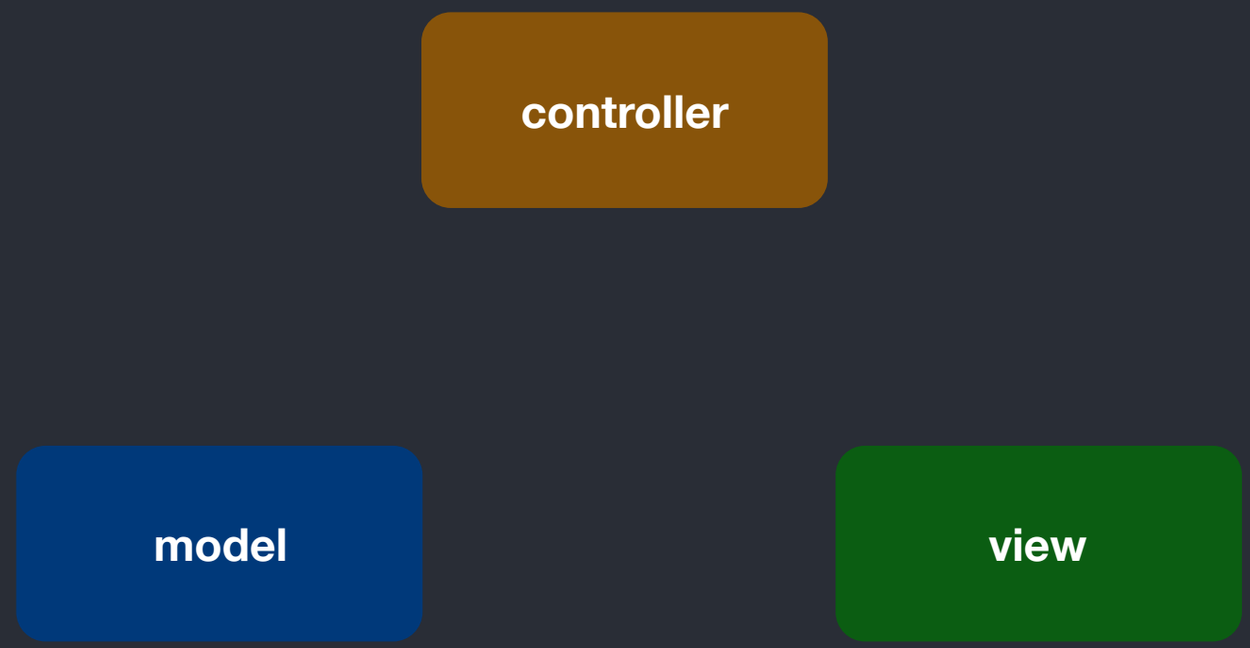


Model enthält die reine Anwendungslogik, ohne UI

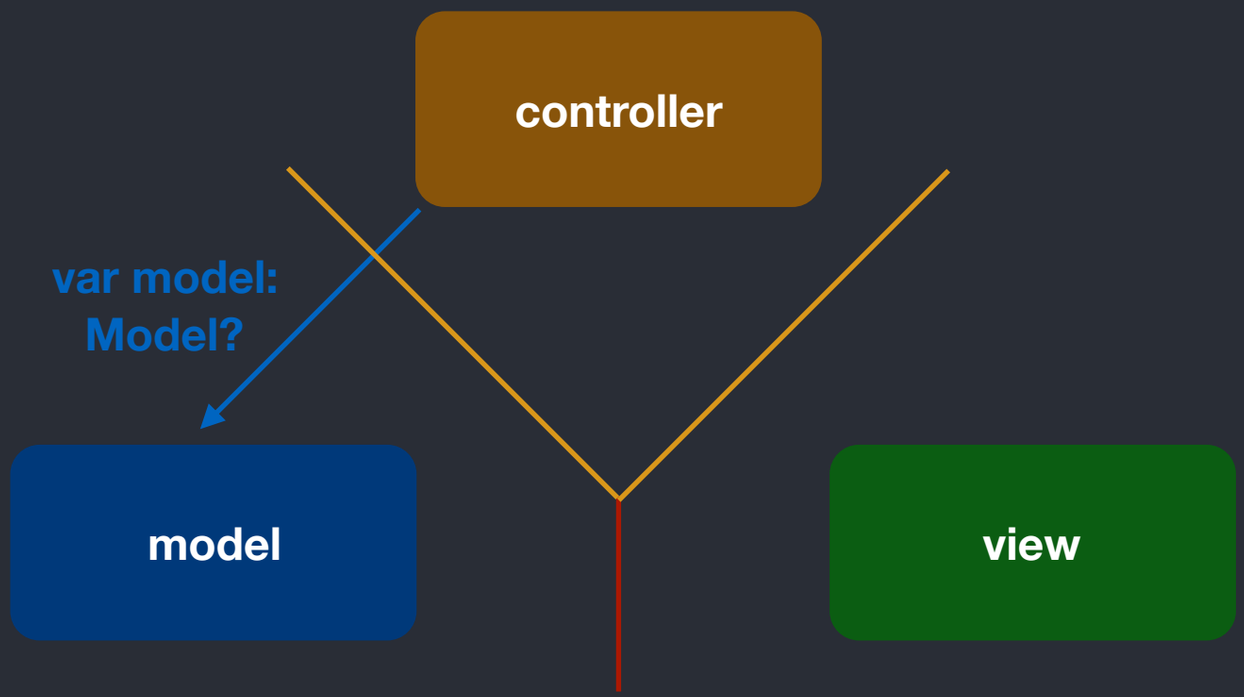
und ist deshalb unabhängig vom Controller, der View und dem Endgerät



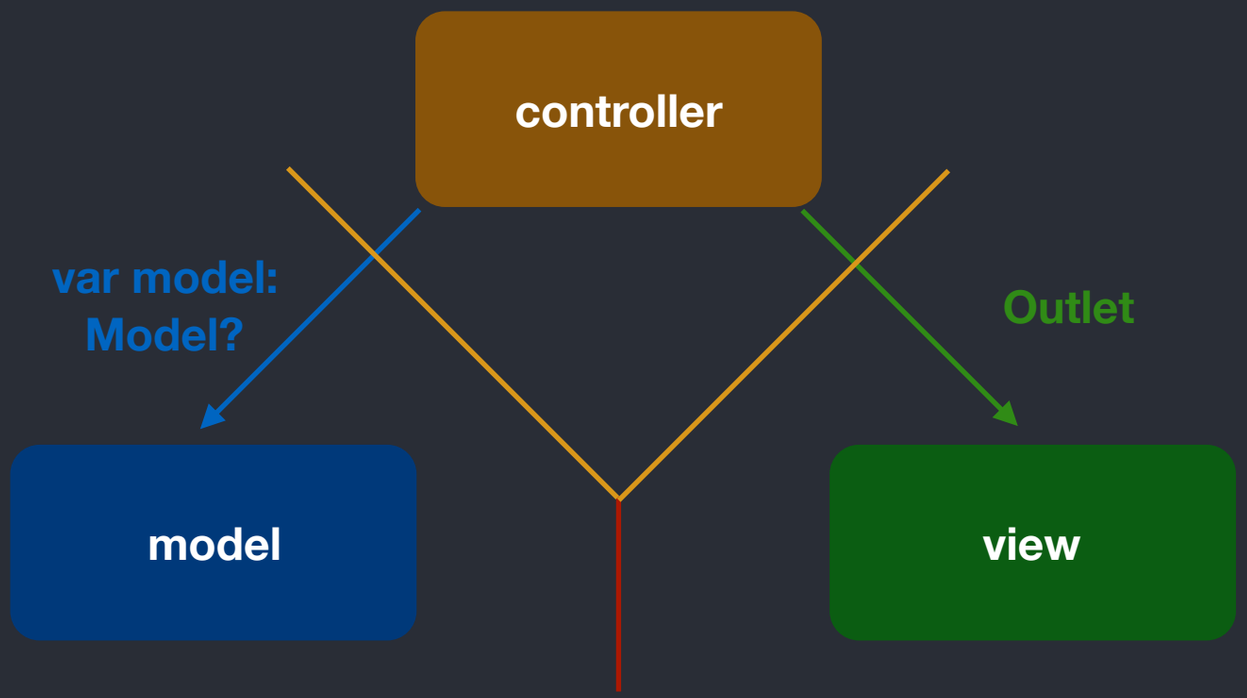
View’s sind generische UI-Elemente, die Informationen darstellen und Interaktionen anbieten



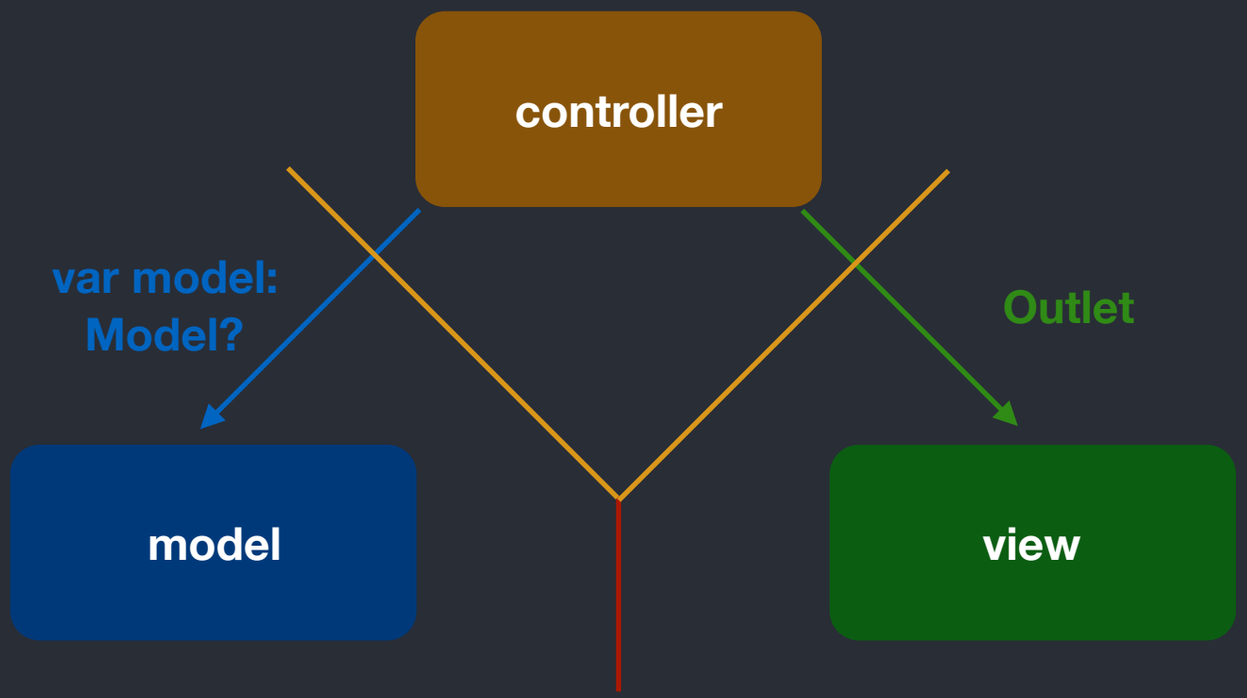
Controller interpretieren View und/oder Model und übersetzen jeweils in die andere Richtung



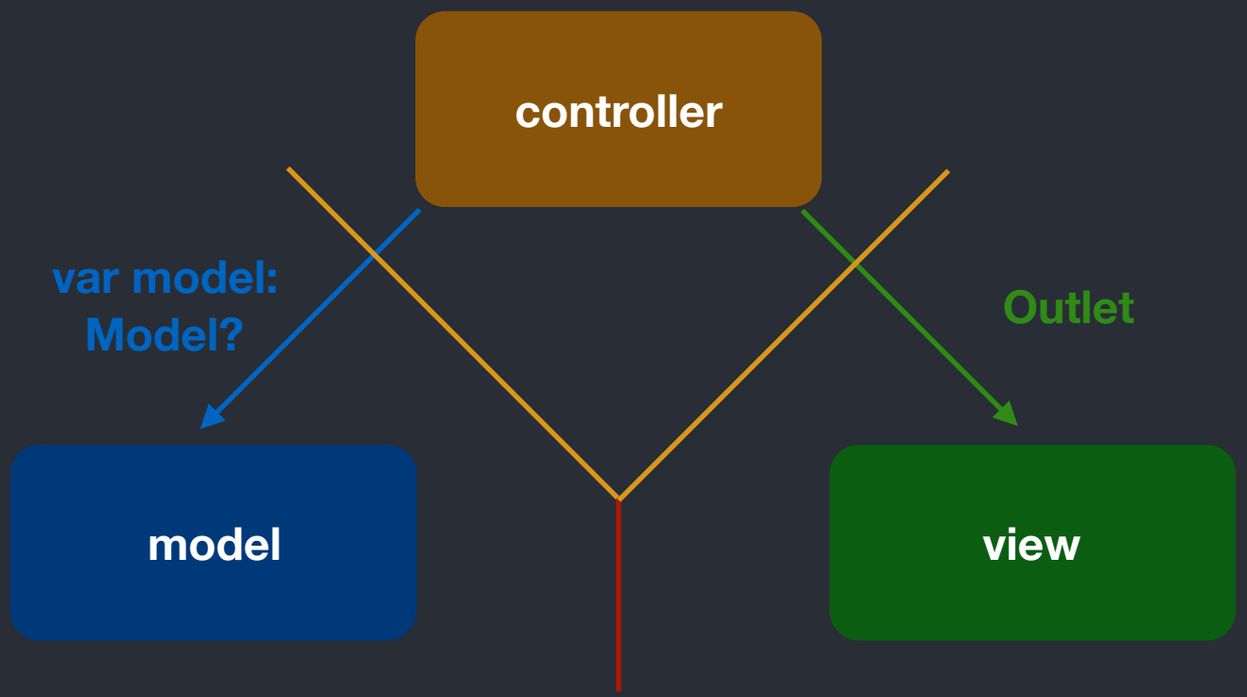
Controller haben eine property des Models, welches sie präsentieren



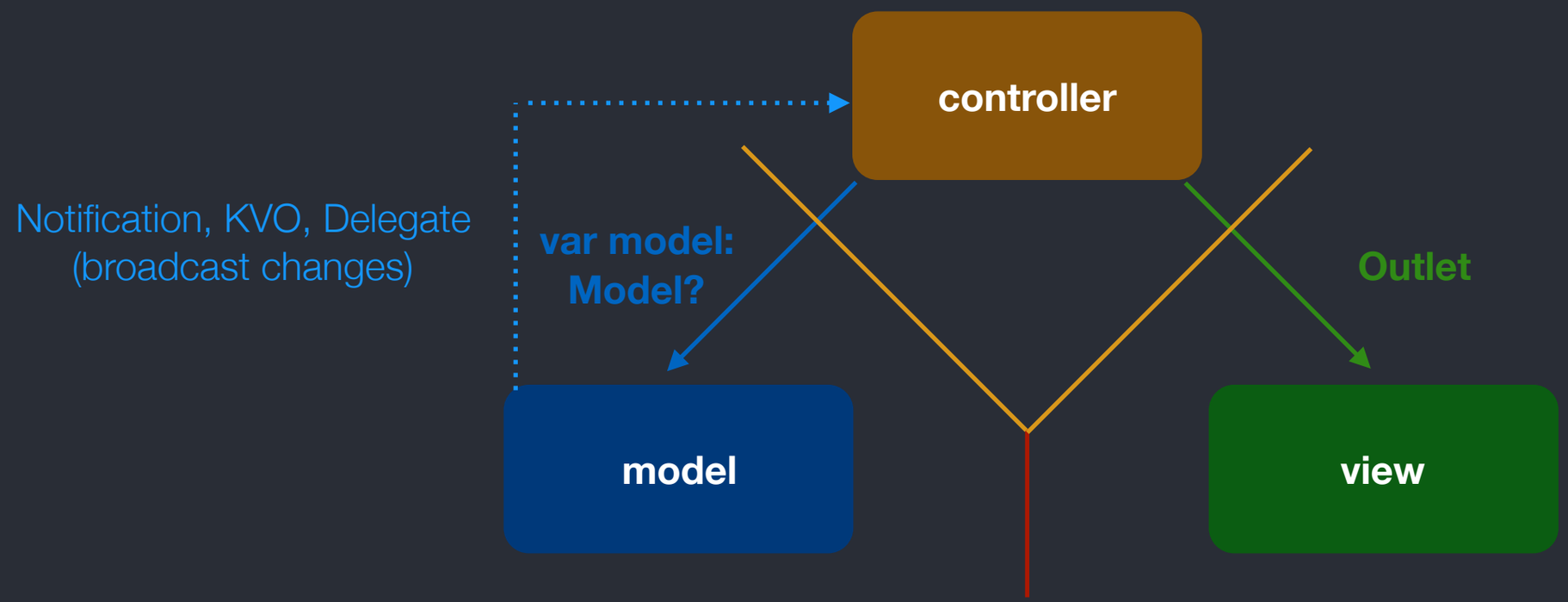
Controller haben eine Outlet-Verbindung zur View, welche sie mit dem Model synchronisieren



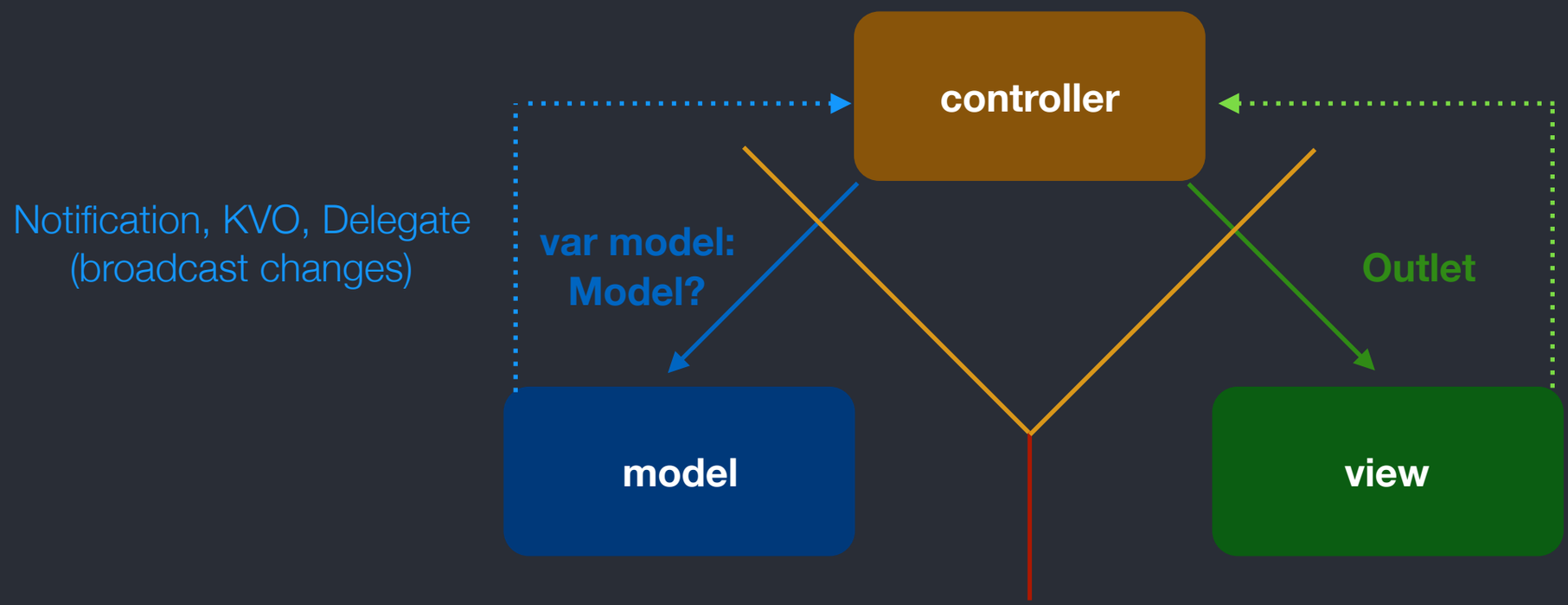
Model und View sprechen niemals direkt miteinander



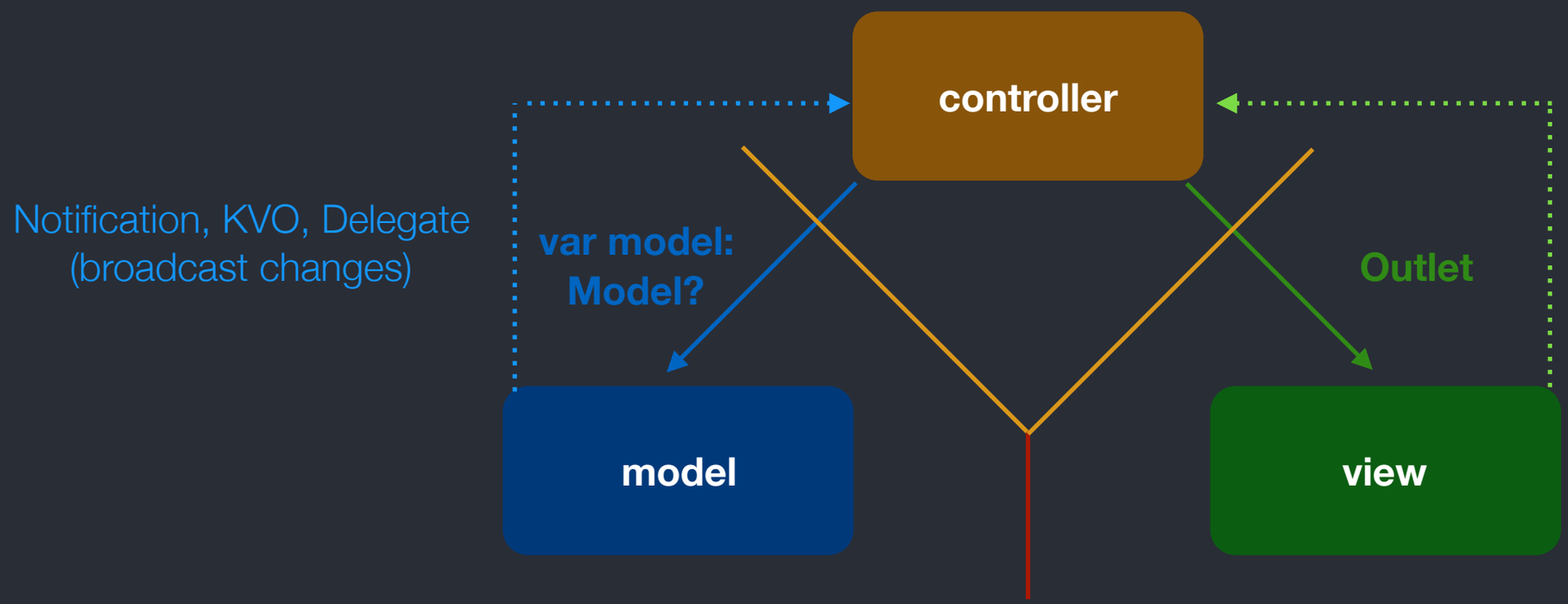
Was passiert, wenn das Model sich ändert oder neue Daten hat? Es ist unabhängig von der View und dem Controller



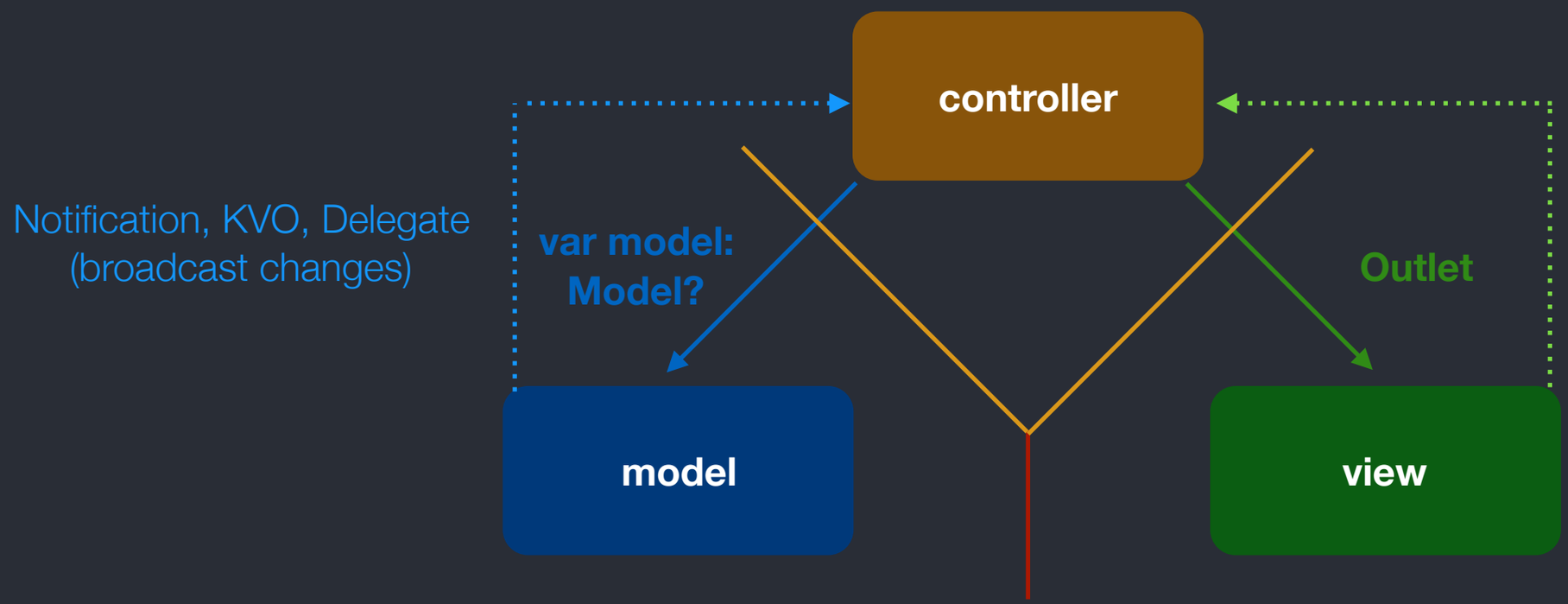
Das Model broadcastet (entkoppelt) die Änderungen über das Notification-, KVO- oder Delegate-Pattern an den Controller



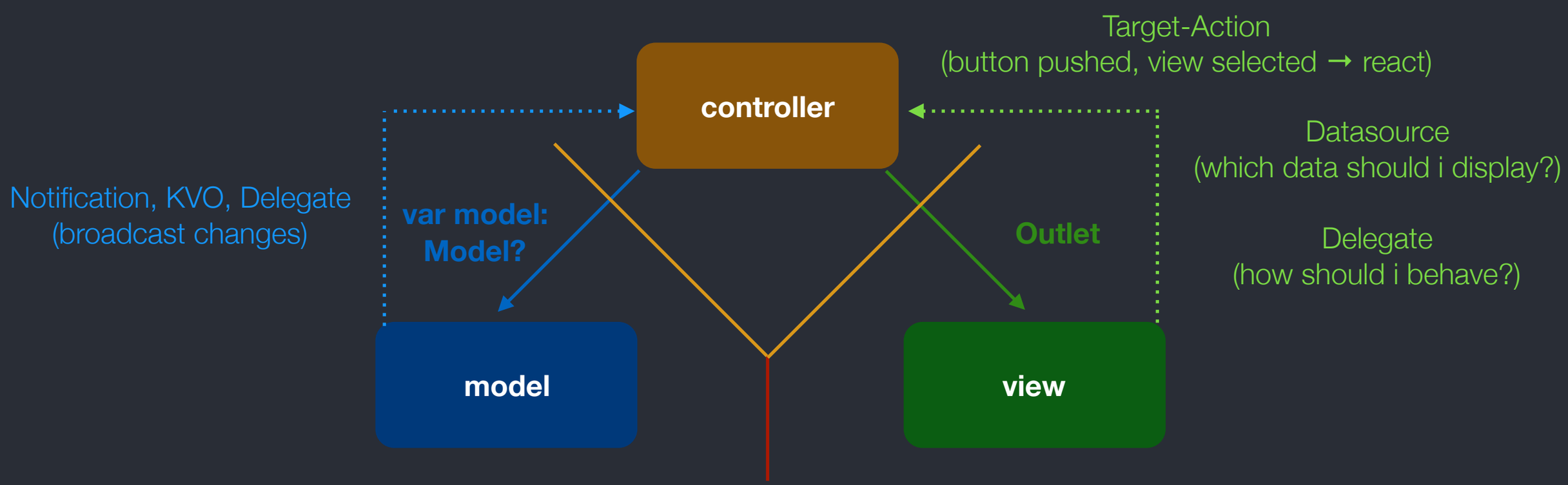
Die generische View muss eine Datenquelle erhalten, um zu wissen, welche Informationen angezeigt werden sollen



Zudem muss die generische View nachfragen, wie sie sich in bestimmten Situationen verhalten soll



Zuletzt kann die generische View Interaktionen/Aktionen weiterleiten, weil sie nicht weiss, wie sie damit umgehen soll



Auch hier verwendet die View ein lose gekoppeltes Prinzip für die Kommunikation zum Controller

controller

controll

oller

model

view

cont

roller

view

model

view

iPhone 7 – iOS 10.3 (14E269)

Carrier 16:40

News

F10 | Allgemein | Informatik

120 Gäste kamen zur Eröffnung des Polymer-Labors

Lebensgrundlage Wasser: Abschlussarbeiten helfen Kosten sparen und nützen der Umwelt

News | Kontakte | Mensa | Stundenplan | Einstellungen

iPhone 7 – iOS 10.3 (14E269)

Carrier 16:41

News

120 Gäste kamen zur Eröffnung des Polymer-Labors

Neubau für den Campus Gummersbach der TH Köln erweitert das Angebot für Lehre, Forschung und Transfer

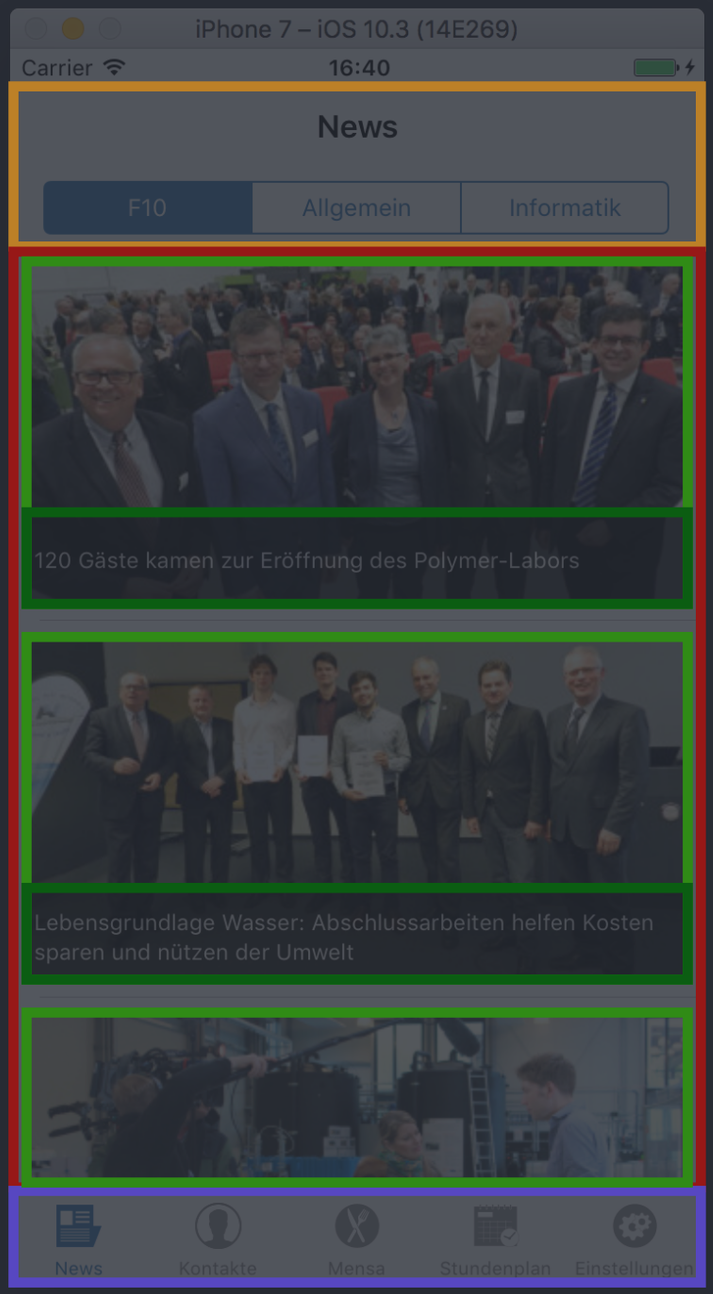
24.3.2017 - Unter den Hochschulen für Angewandte Wissenschaften hat die TH Köln in Deutschland eine Sonderstellung, nicht nur durch ihre Größe, sondern auch durch ihr breites Angebot im Bereich Chemie: Die TH Köln an ihren drei Standorten Köln, Leverkusen und Gummersbach über die komplette

News | Kontakte | Mensa | Stundenplan | Einstellungen

iPhone 7 – iOS 10.3 (14E269)

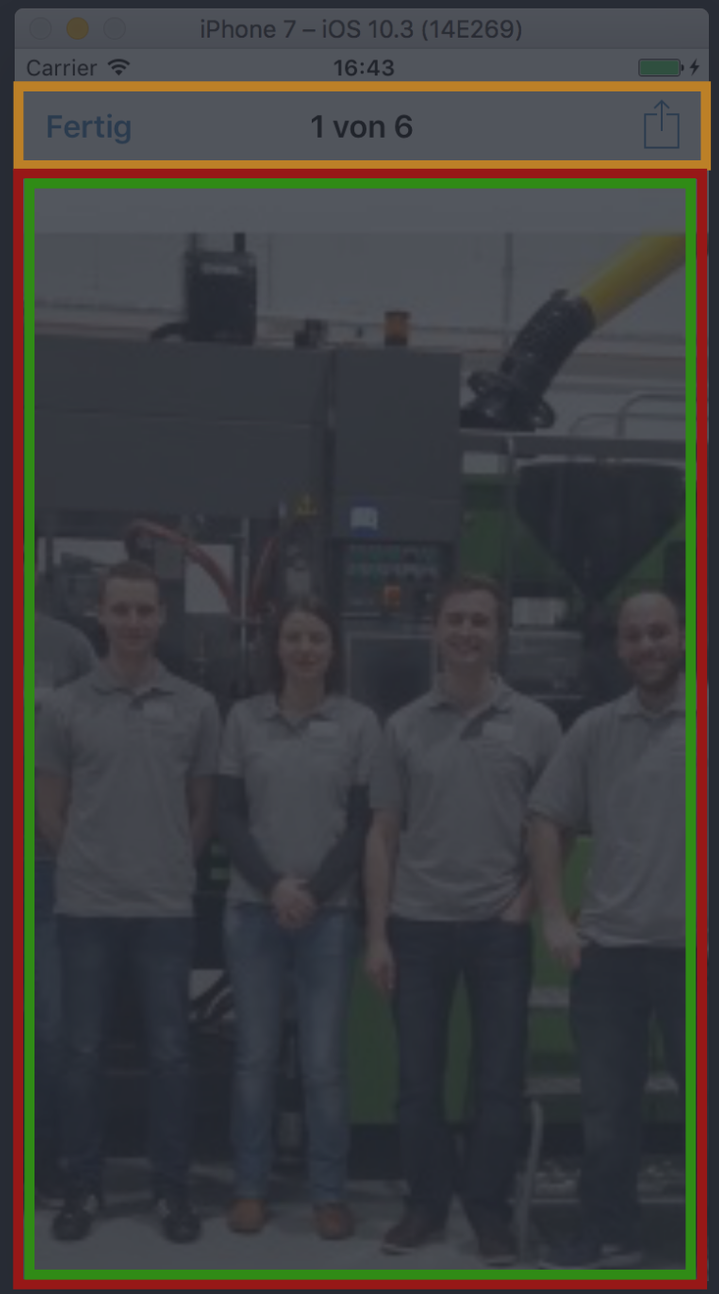
Carrier 16:43

Fertig 1 von 6



- NavigationController
  - NavigationBar*
- NewsTableViewController
  - NewsCell
    - UIImageView
    - UILabel
- TabBarController
  - TabBarButton*

- NavigationController
  - NavigationBar*
    - BarButtonItems
- DetailNewsViewController
  - ContainerView
    - PageViewController
      - UIImageView
    - UIPageLabel
  - StackView
    - UILabel x 3
- ...



- NavigationController
  - NavigationBar*
    - BarButtonItems
- ImageViewController
  - UIImageView
  - UITapGestureRecognizer

*ist nicht ganz korrekt

Heute

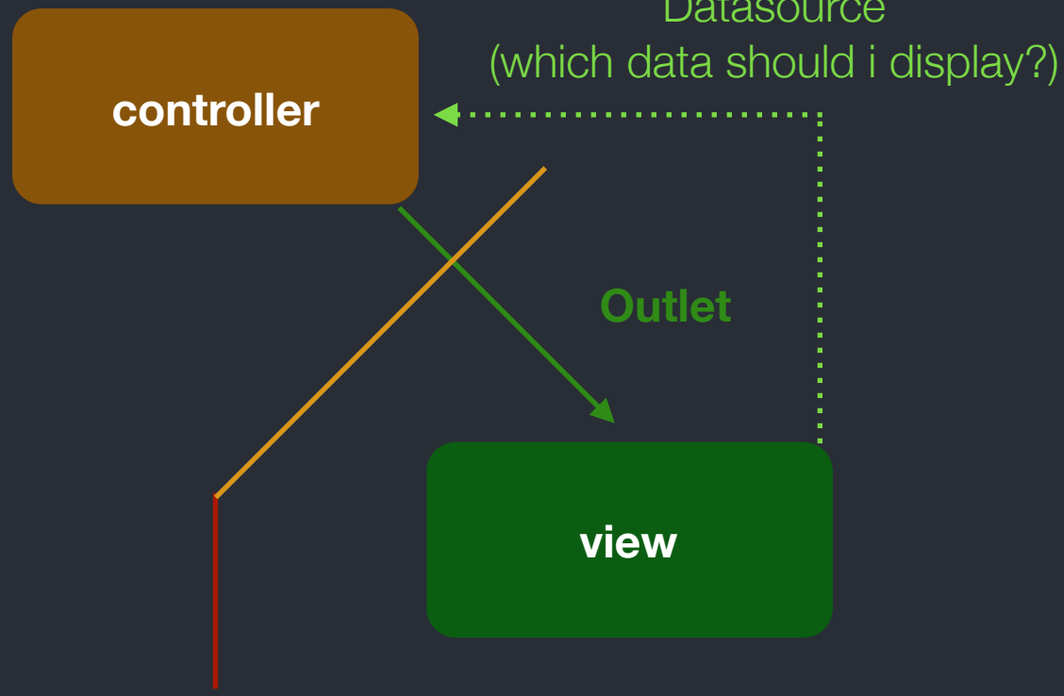
Model-View-Controller (MVC)

DataSource, Delegate, Target-Action, Notification

Demo

DataSource

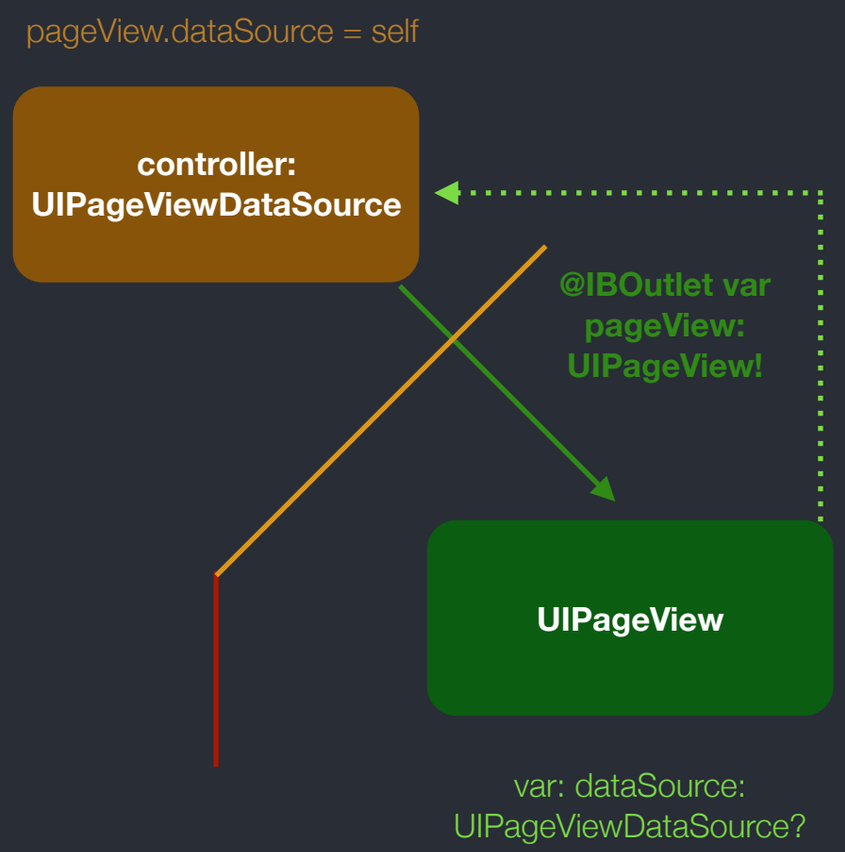
- Bei vielen (eher statischen) Views reicht das Setzen von Properties bereits aus
- Andere (eher dynamischere) Views, wie TableView, CollectionView und PageView, benötigen die Daten erst im Laufe der Interaktion
- Eine DataSource ist ein Protocol und enthält alle Funktionen, um Daten anzufragen
  - View hat ein optionales Property vom Typ der DataSource
  - Controller setzt sich selbst als DataSource der View
  - Controller muss alle Funktionen des Protocols implementieren
  - View fragt über das Property die Daten an
  - Der Controller liefert die Daten, weil er die Funktionen implementiert hat



Die generische View muss eine Datenquelle erhalten,
um zu wissen, welche Informationen angezeigt werden sollen

DataSource

- Bei vielen (eher statischen) Views reicht das Setzen von Properties bereits aus
- Andere (eher dynamischere) Views, wie TableView, CollectionView und PageView, benötigen die Daten erst im Laufe der Interaktion
- Eine DataSource ist ein Protocol und enthält alle Funktionen, um Daten anzufragen
  - View hat ein optionales Property vom Typ der DataSource
  - Controller setzt sich selbst als DataSource der View
  - Controller muss alle Funktionen des Protocols implementieren
  - View fragt über das Property die Daten an
  - Der Controller liefert die Daten, weil er die Funktionen implementiert hat



DataSource

Protocol

```
public protocol UIPageViewControllerDataSource : NSObjectProtocol {
    public func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore viewController:
UIViewController) -> UIViewController?

    public func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter viewController:
UIViewController) -> UIViewController?

    optional public func presentationCount(for pageViewController: UIPageViewController) -> Int
}
```

Konsument

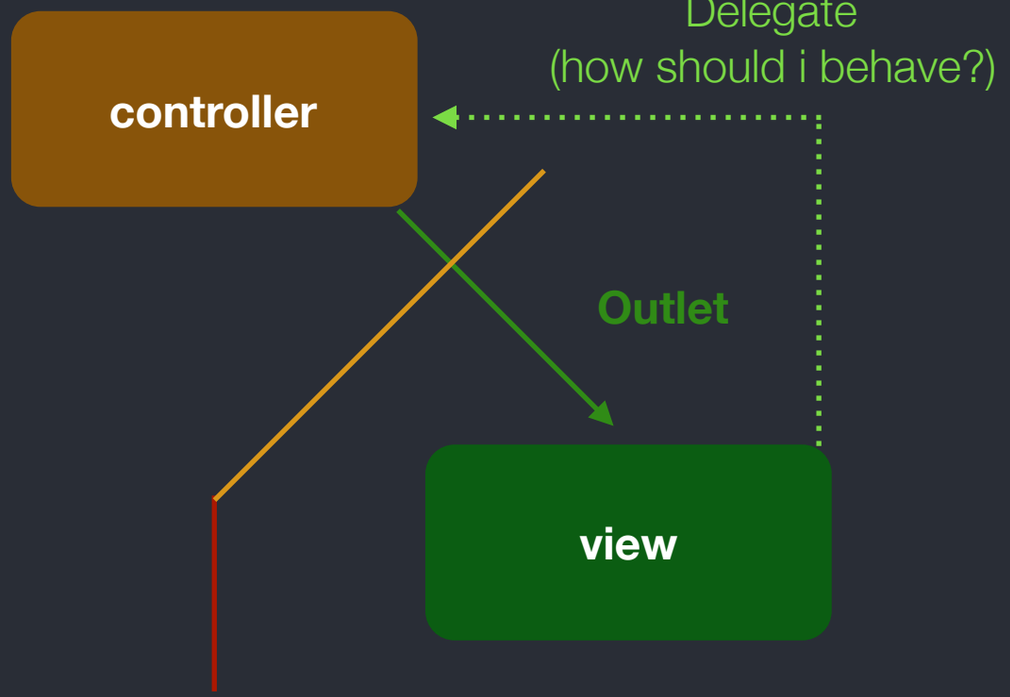
```
class GamePageViewController: UIPageViewController {
   var dataSource: UIPageViewControllerDataSource?

   func foo() {
      self.dataSource?.pageViewController(self, viewControllerAfter: viewController)
   }
}
```

Anbieter

```
class SomeViewController: UIPageViewController {
    var gameViewController: GameViewController?

    override func viewDidLoad() {
        super.viewDidLoad()

        gameViewController.dataSource = self
    }
}

extension SomeViewController: UIPageViewControllerDataSource {
    func pageViewController(
        _ pageViewController: UIPageViewController,
        viewControllerAfter viewController: UIViewController) -> UIViewController? {
        if viewController is A { return fooController } else { return nil }
    }
}
```

Delegate

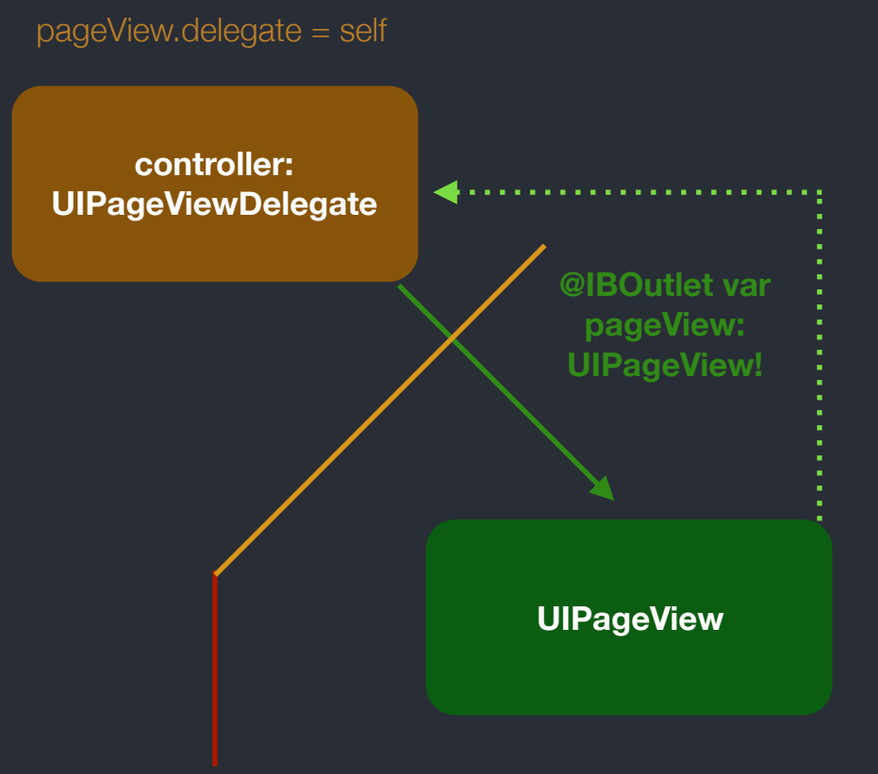
- Eine View kommuniziert an seine aufrufende Instanz zurück
- Der Controller interpretiert die Änderungen an der View und übersetzt diese bspw. in ein Model
- Ein Delegate ist ein Protocol und enthält alle Funktionen, um Informationen zu kommunizieren
  - View hat ein optionales Property vom Typ der Delegate
  - Controller setzt sich selbst als Delegate der View
  - Controller muss alle Funktionen des Protocols implementieren
  - View kommuniziert seine Schritte, Änderungen usw. über das Property
  - Der Controller bekommt die Chance auf diese Informationen zu reagieren, weil er die Funktionen implementiert hat



Zudem muss die generische View nachfragen,
wie sie sich in bestimmten Situationen verhalten soll

Delegate

- Eine View kommuniziert an seine aufrufende Instanz zurück
- Der Controller interpretiert die Änderungen an der View und übersetzt diese bspw. in ein Model
- Ein Delegate ist ein Protocol und enthält alle Funktionen, um Informationen zu kommunizieren
  - View hat ein optionales Property vom Typ der Delegate
  - Controller setzt sich selbst als Delegate der View
  - Controller muss alle Funktionen des Protocols implementieren
  - View kommuniziert seine Schritte, Änderungen usw. über das Property
  - Der Controller bekommt die Chance auf diese Informationen zu reagieren, weil er die Funktionen implementiert hat



Delegate

Protocol

```
public protocol UIPageViewControllerDelegate : NSObjectProtocol {

    optional public func pageViewController(_ pageViewController: UIPageViewController, willTransitionTo
pendingViewControllers: [UIViewController])

    optional public func pageViewController(_ pageViewController: UIPageViewController, didFinishAnimating finished:
Bool, previousViewControllers: [UIViewController], transitionCompleted completed: Bool)
}
```

Konsument

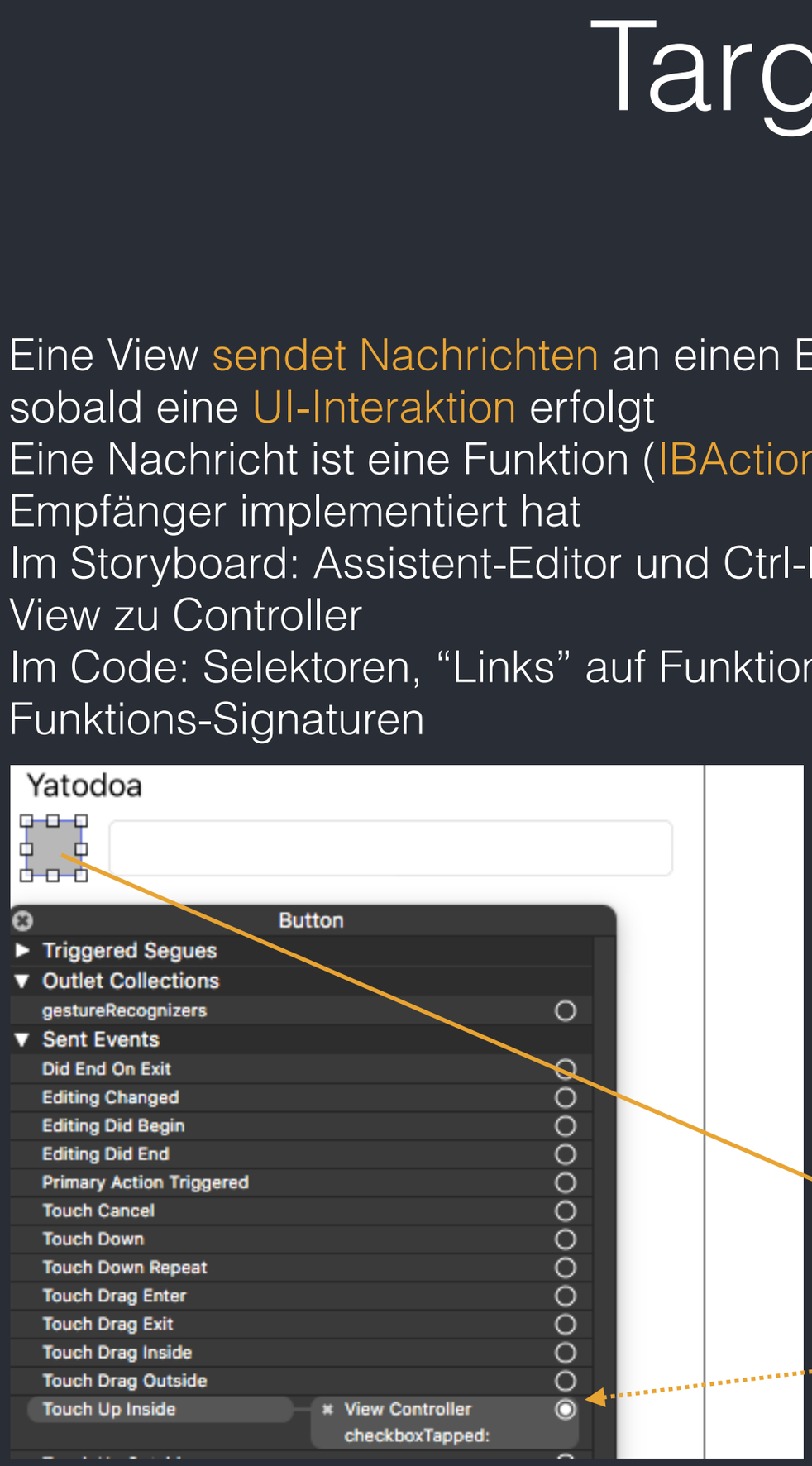
```
class GamePageViewController: UIPageViewController {
   var delegate: UIPageViewControllerDelegate?

   func foo() {
      self.delegate?.pageViewController(self, willTransitionTo: [viewController])
   }
}
```

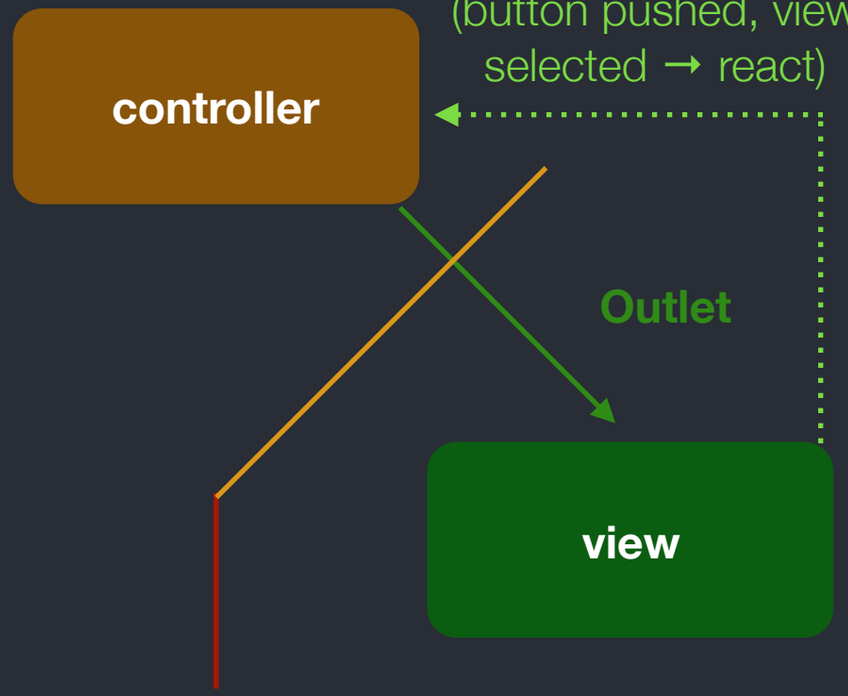
Anbieter

```
class SomeViewController: UIPageViewController {
    var gameViewController: GameViewController?

    override func viewDidLoad() {
        super.viewDidLoad()

        gameViewController.delegate = self
    }
}

extension SomeViewController: UIPageViewControllerDelegate {

    func pageViewController(_ pageViewController: UIPageViewController, willTransitionTo pendingViewControllers:
[UIViewController]) {
        animate()
    }
}
```

Target-Action

- Eine View sendet Nachrichten an einen Empfänger, sobald eine UI-Interaktion erfolgt
- Eine Nachricht ist eine Funktion (IBAction), die der Empfänger implementiert hat
- Im Storyboard: Assistent-Editor und Ctrl-Drag von View zu Controller
- Im Code: Selektoren, “Links” auf Funktionen über Funktions-Signaturen



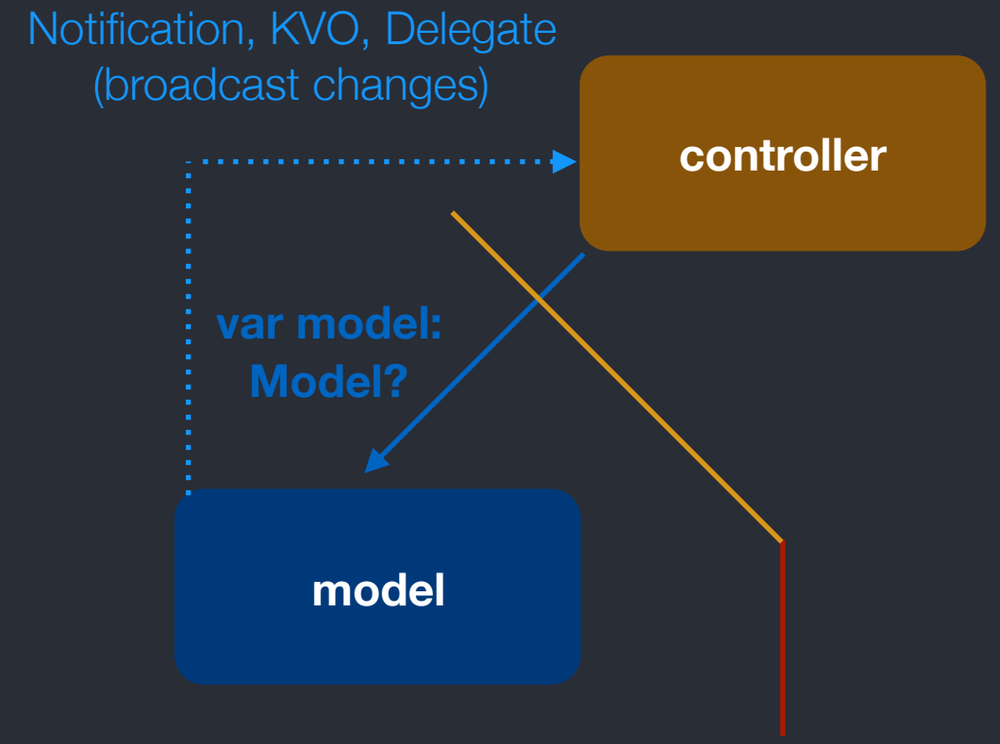


Zuletzt kann die generische View Interaktionen/Aktionen weiterleiten, weil sie nicht weiss, wie sie damit umgehen soll

```
@IBAction func checkboxTapped(_ sender: UIButton) {
    if let text = todoTextField.text, !text.isEmpty {
        if sender.currentTitle == nil {
            sender.setTitle("✓", for: UIControlState.
```

Notification

- Ein Objekt (i.d.R. Model) broadcastet Nachrichten zu einem Thema
- Interessenten registrieren sich für ein Thema und erhalten Nachrichten
- Zudem sollten sich Interessanten wieder deregistrieren, um Memory-Leaks zu vermeiden
- Beides erfolgt i.d.R. in Lifecycle-Methoden wie viewWillAppear und viewWillDisappear
- Die Nachrichten sind Selektoren mit einem Parameter vom Typ Notification
- Eine Notification enthält u.a. den Sender und Payload (userInfo) vom Typ Dictionary<AnyHashable, Any>
- iOS selbst sendet Notifications bei bestimmten Events unter dem Namespace Notification.Name.*
- Notifications werden für 1:n verwendet, während Delegates für 1:1 Beziehungen verwendet werden



Das Model broadcastet (entkoppelt) die Änderungen über das Notification-, KVO- oder Delegate-Pattern an den Controller

Notifications

Sender

```
class SomeModel {
    func action() {
        let center = NotificationCenter.default
        center.post(name: Notification.Name.SomethingDidChange, object: self, userInfo: ["Data": data])
    }
}
```

Empfänger 1

```
class A: UIViewController {
    func viewWillAppear() {
        let center = NotificationCenter.default
        center.addObserver(self, selector: #selector(updateSomething), name: Notification.Name.SomethingDidChange,
object: nil)
    }

    @objc func updateSomething(notification: Notification) {
        if let data = notification.userInfo["Data"] { … }
    }

    func viewWillDisappear() {
        NotificationCenter.default.removeObserver(self)
    }
}
```

Empfänger 2

```
class B: UIViewController {
    func viewWillAppear() {
        let center = NotificationCenter.default
        center.addObserver(self, selector: #selector(updateSomething), name: Notification.Name.SomethingDidChange, object: nil)
    }

    @objc func updateSomething(notification: Notification) {
        if let data = notification.userInfo["Data"] { … }
    }
```

Demo - Yatodoa

MVC
Delegate
Custom Views
Swift Features

Yatodoa - Assignment

- Starke Verbindung zwischen View und Controller mit Delegate lösen
- Mindestens eine weitere Property im Model definieren und in der UI verwenden, z.B. letzte Änderung
- Synchronisierung der Properties zwischen Model und View beachten
- Sonstige Änderungen und Verbesserungen sind Willkommen
- Bis zum 14:11, 13:59 Uhr per Pull-Request einreichen