Full Stack iOS Entwicklung mit Swift

WPF FSIOS

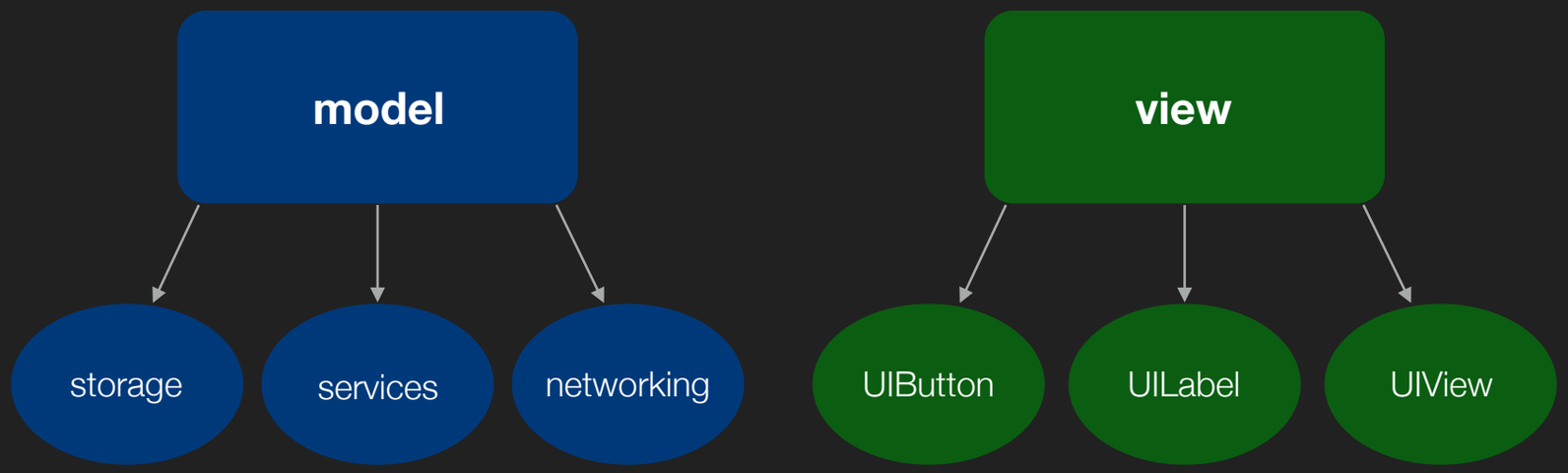
Alexander Dobrynin, M.Sc.

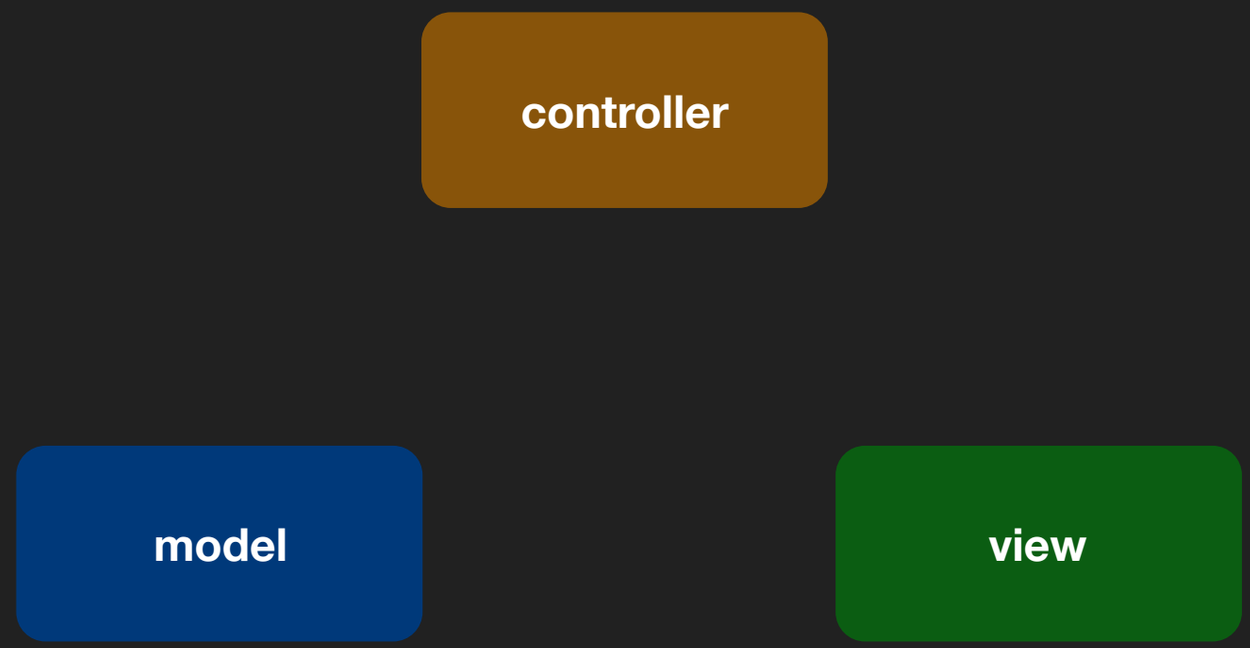
Heute

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

Zusammenfassung

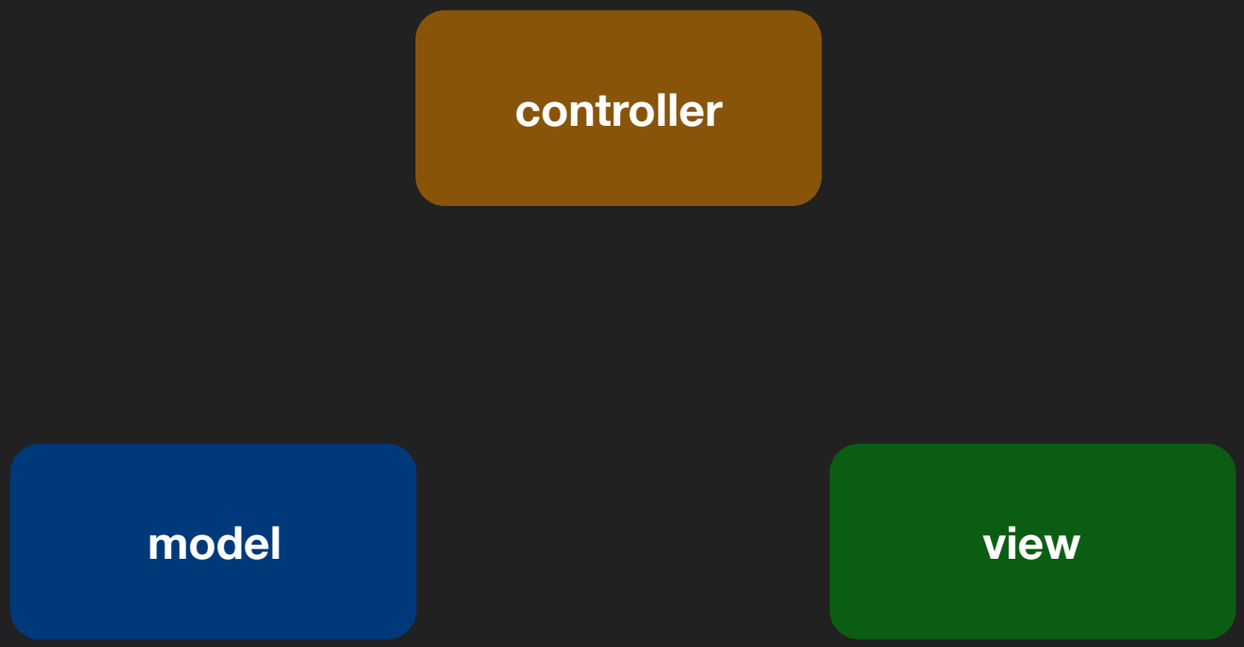
controller


model
storage
services
networking
view
UIButton
UILabel
UIView

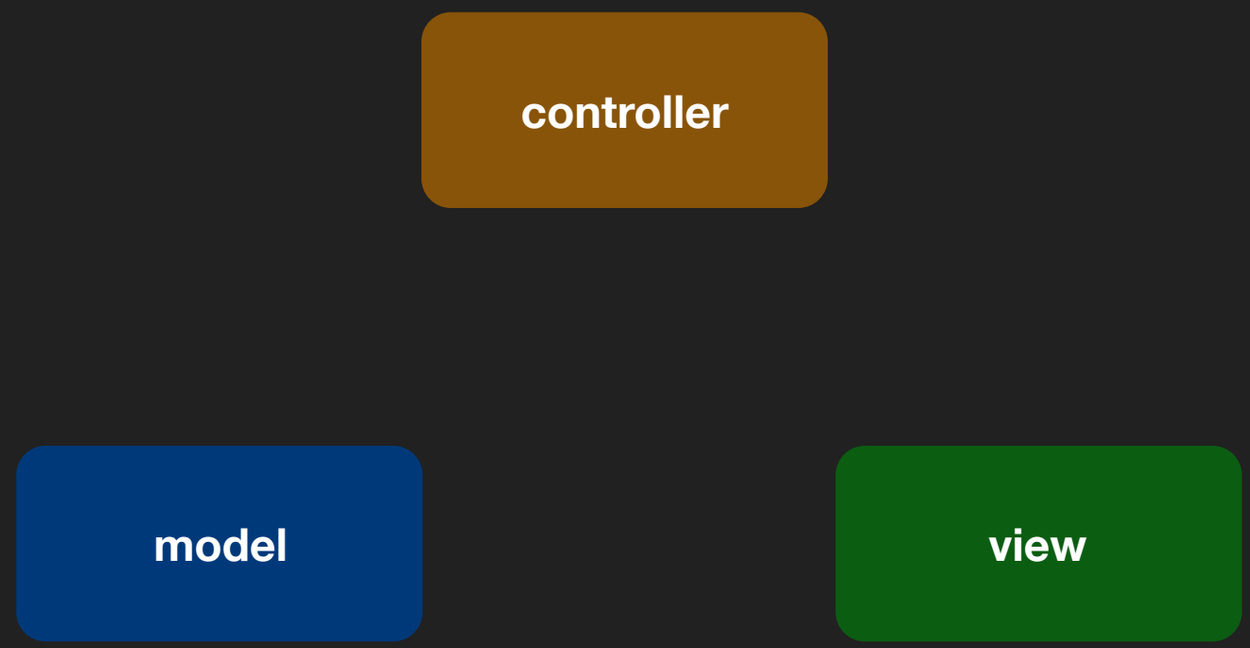


Model enthält die reine Anwendungslogik, ohne UI

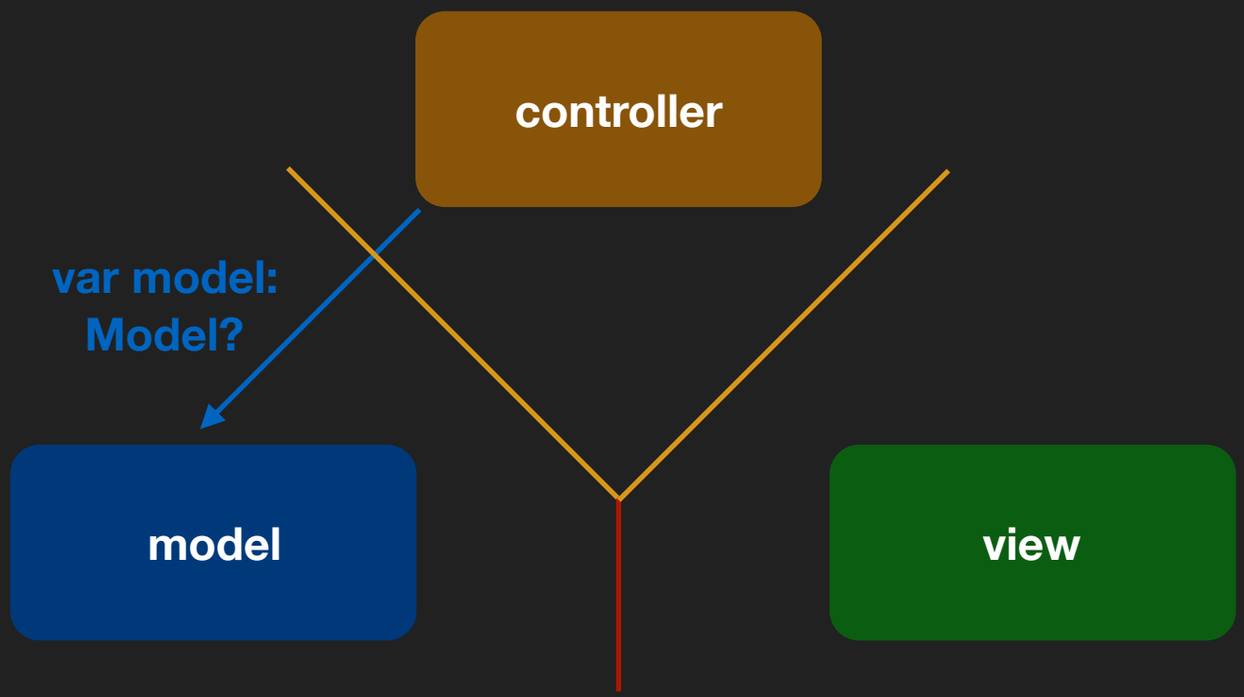
und ist deshalb unabhängig vom Controller, der View und dem Endgerät



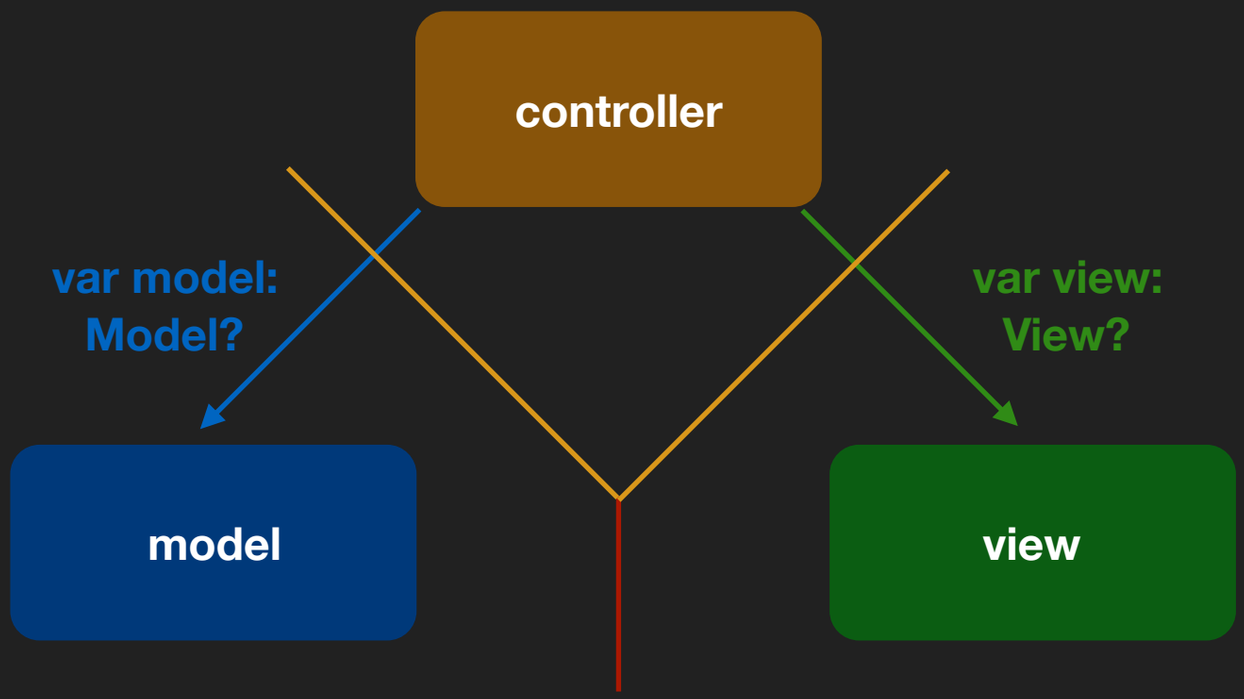
View’s sind generische UI-Elemente, die Informationen darstellen und Interaktionen anbieten



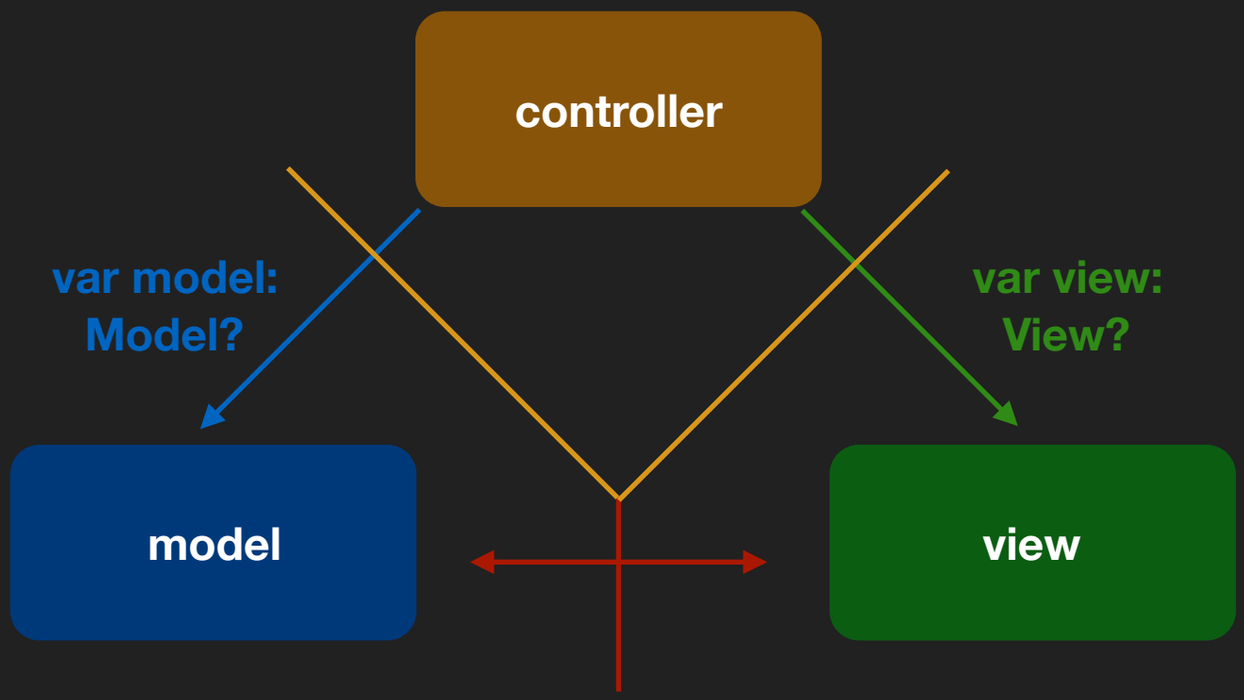
Controller interpretieren View und/oder Model und übersetzen jeweils in die andere Richtung



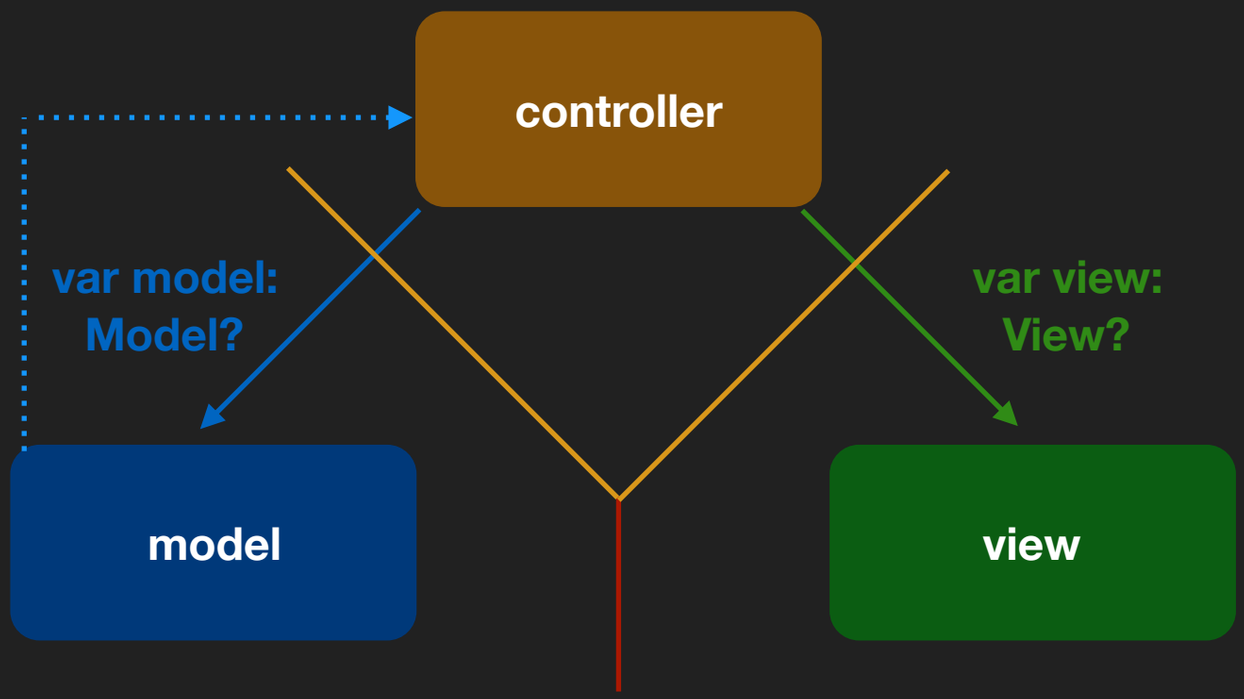
Controller haben eine property des Models, welches sie präsentieren



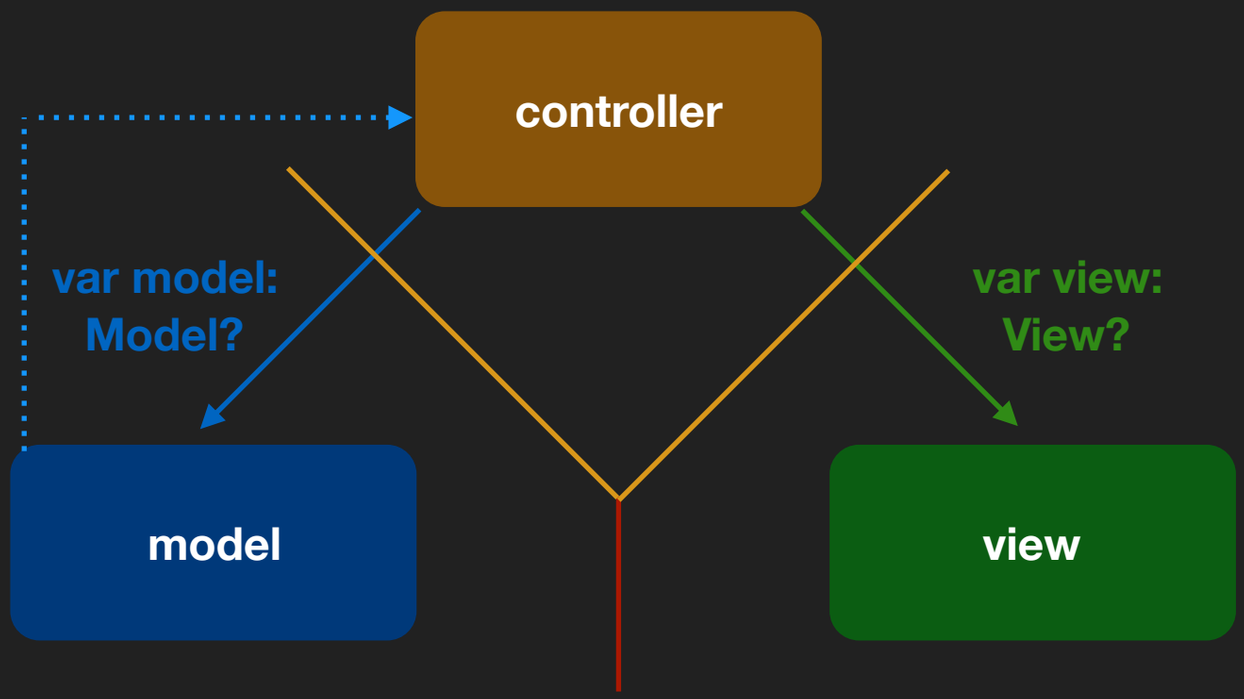
Controller haben eine Outlet-Verbindung zur View, z.B. die visuelle Repräsentation des Models



Model und View sprechen niemals direkt miteinander

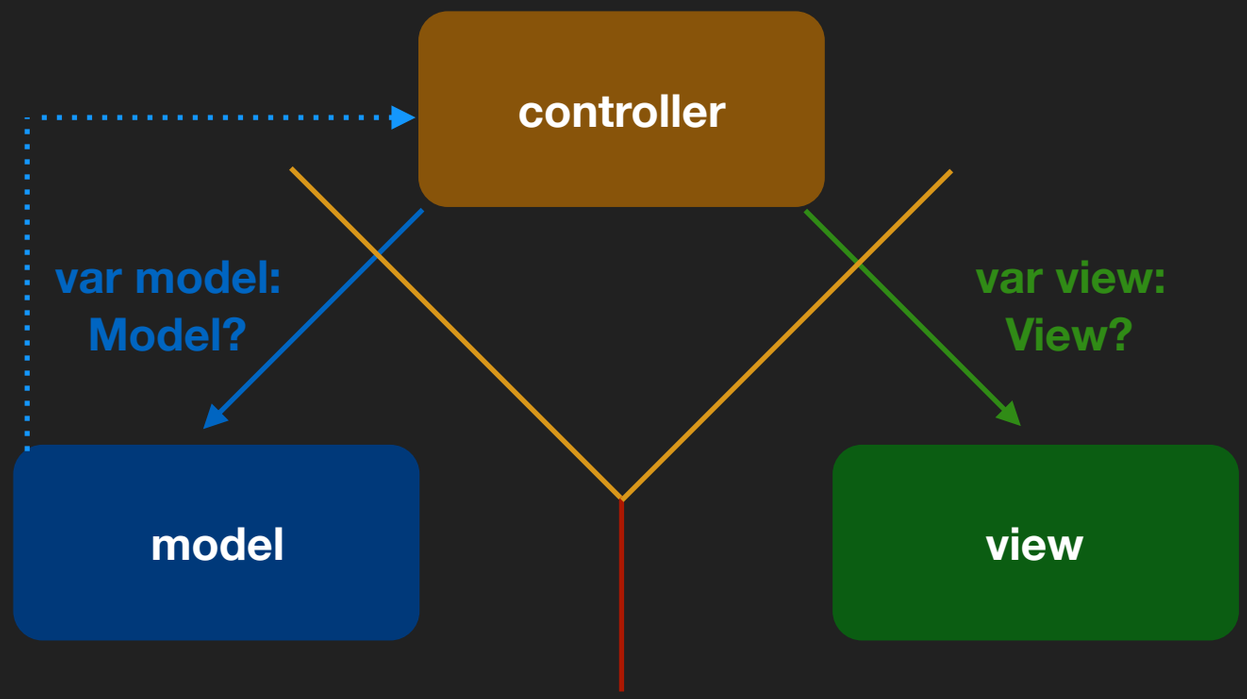


Was passiert, wenn das Model sich ändert oder neue Daten hat? Es ist unabhängig von der View und dem Controller



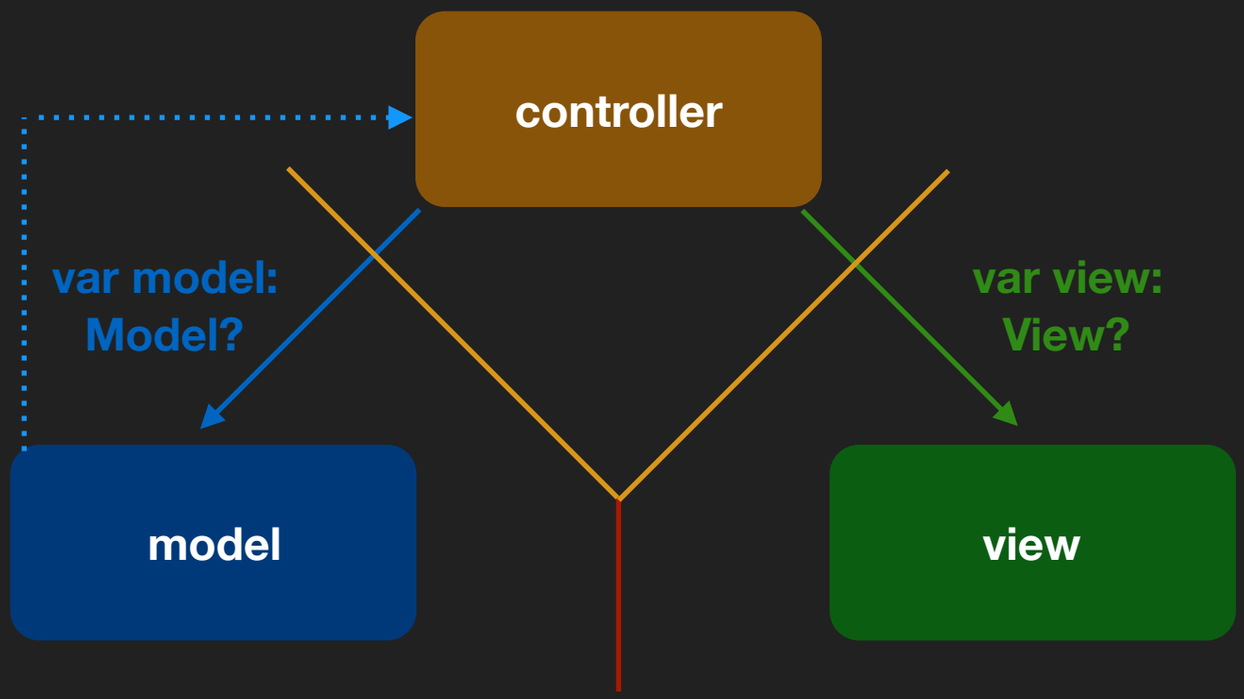
Wie kommuniziert das Model, dass sich die View an anpassen muss, ohne den Controller zu kennen?

[1] Broadcast changes via Notification

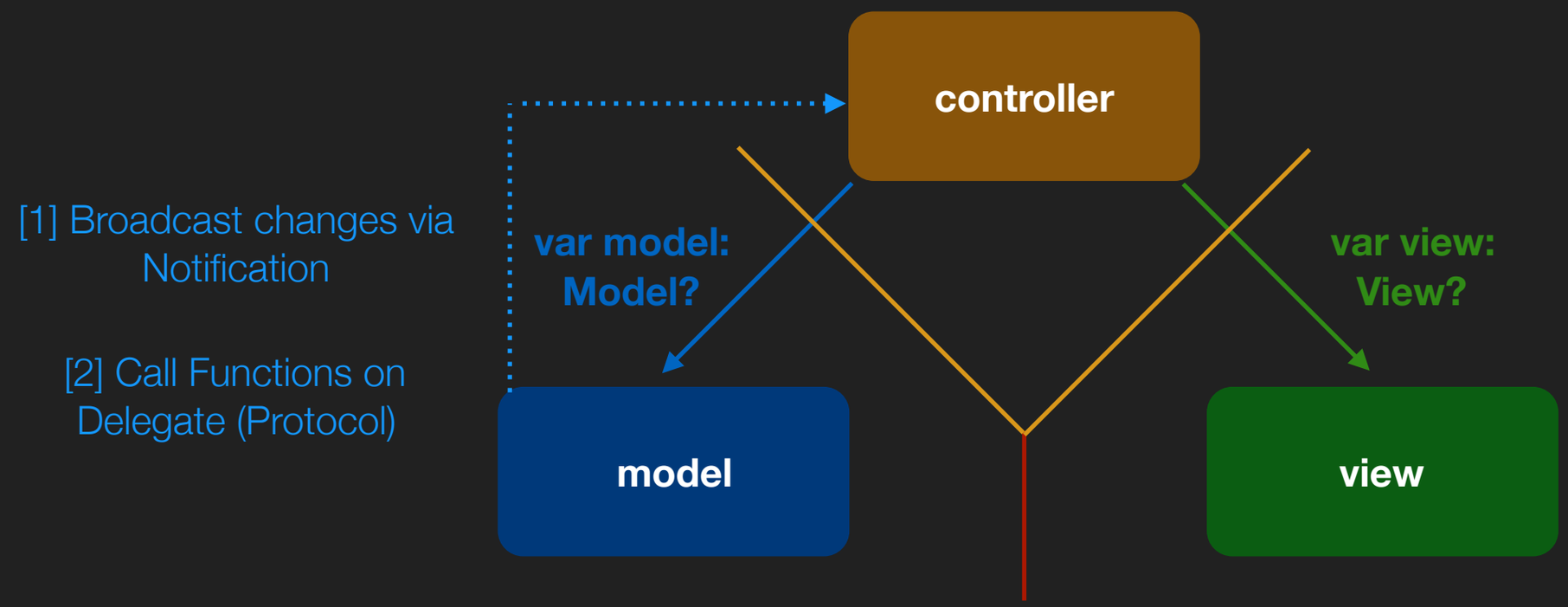


[1] Das Model broadcastet seine Änderungen über das Notification-Pattern an alle Adressaten (z.B. Controller)

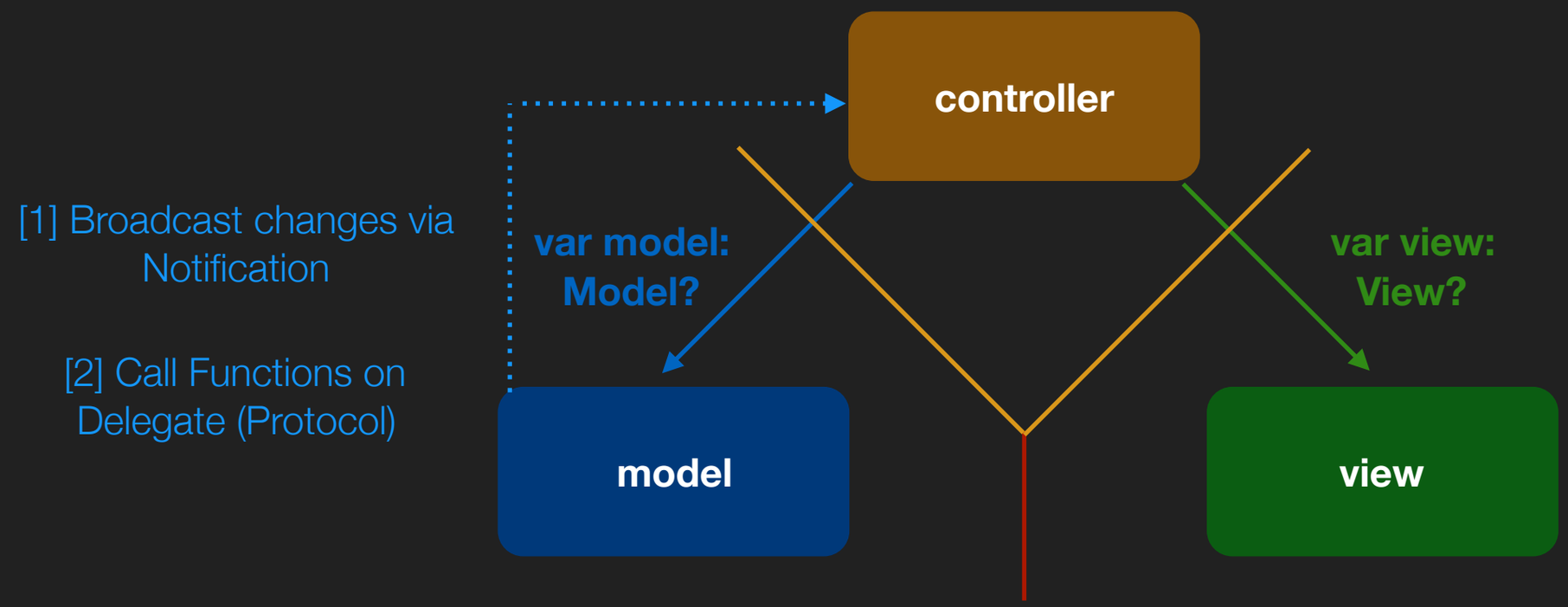
[1] Broadcast changes via Notification



[1] Der Controller registriert sich als Adressat und aktualisiert die View, wenn er vom Model benachrichtig wird



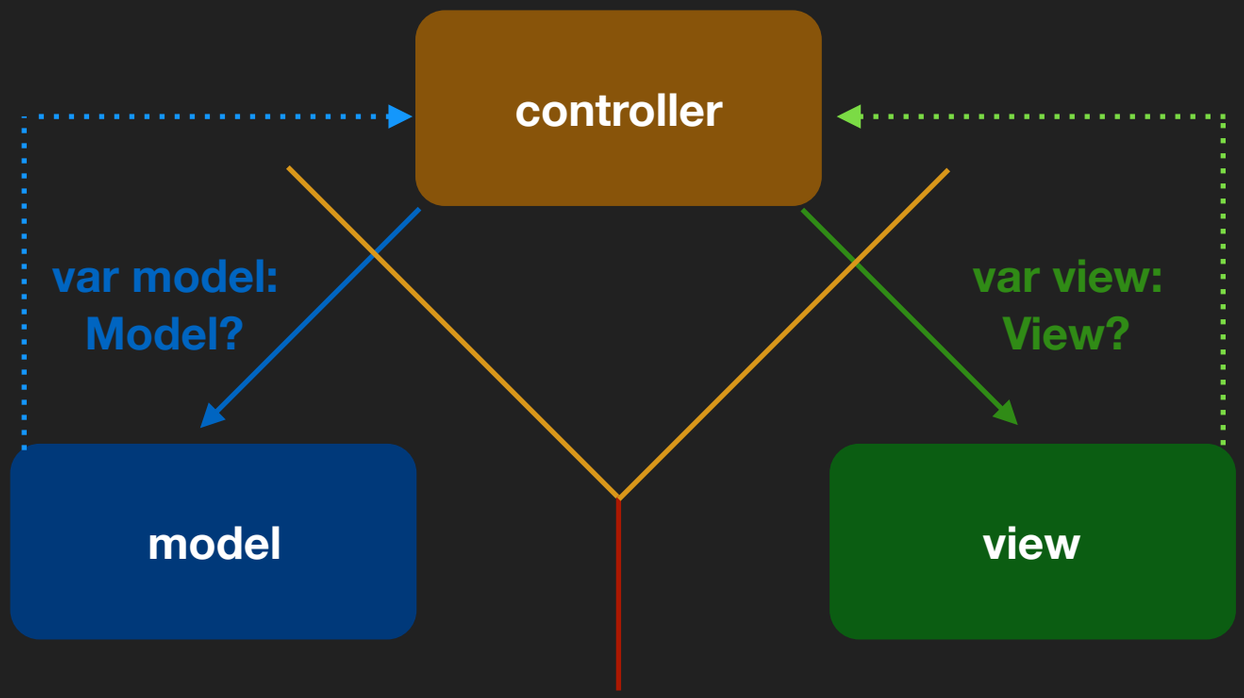
[2] Das Model kommuniziert über eine lose gekoppelte Schnittstelle (Protocol), welche der Controller implementiert



[2] Dadurch ruft das Model Funktionen des Controllers auf, ohne konkret zu wissen, dass es ein Controller ist

[1] Broadcast changes via Notification

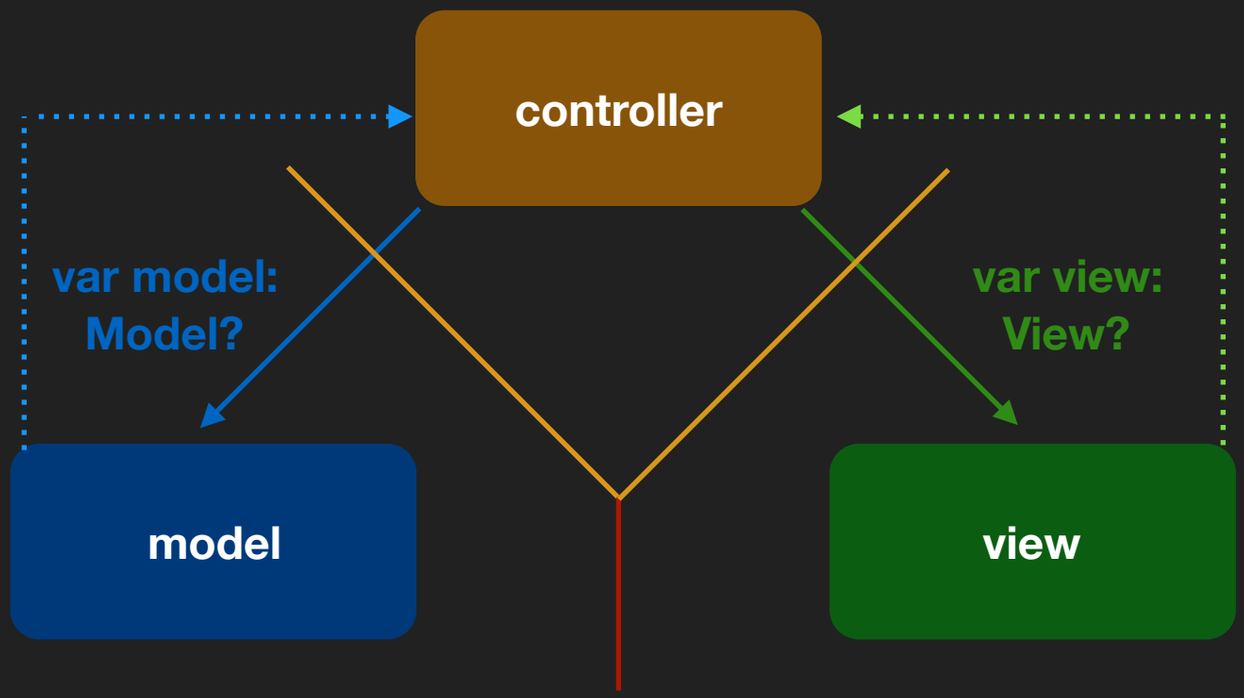
[2] Call Functions on Delegate (Protocol)



Wie leitet die View Interaktionen und Zustandsveränderungen an den Controller weiter, sodass das Model reagieren kann?

[1] Broadcast changes via Notification

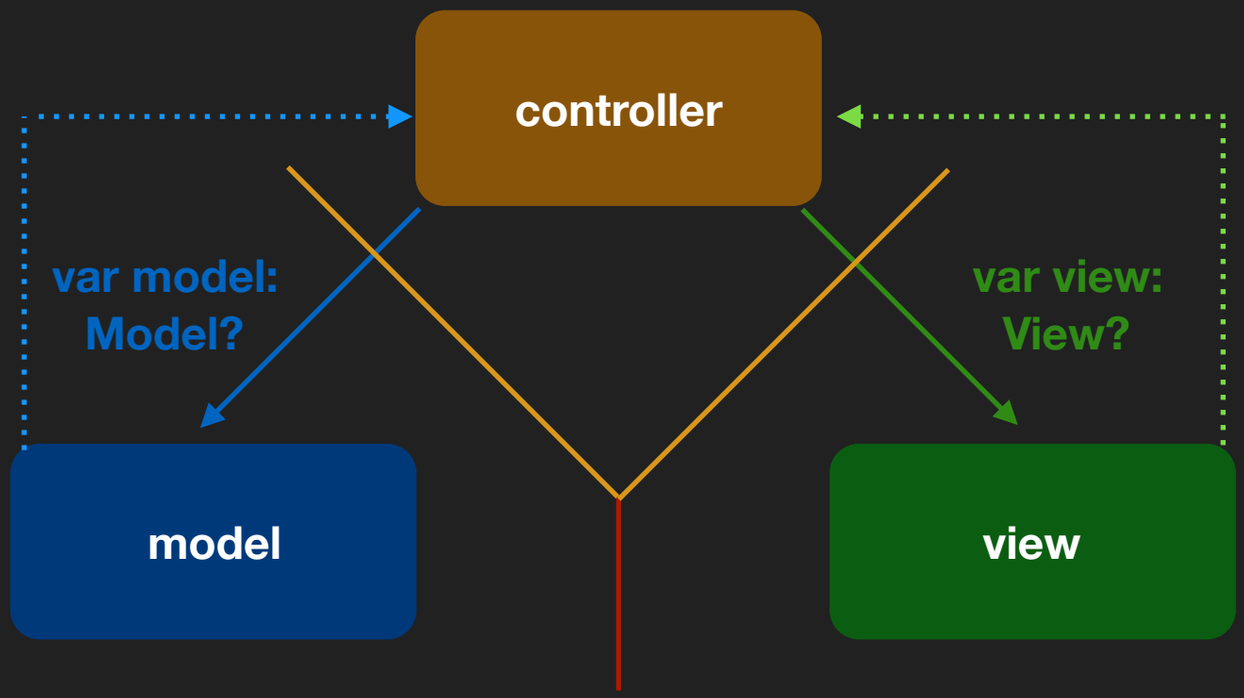
[2] Call Functions on Delegate (Protocol)



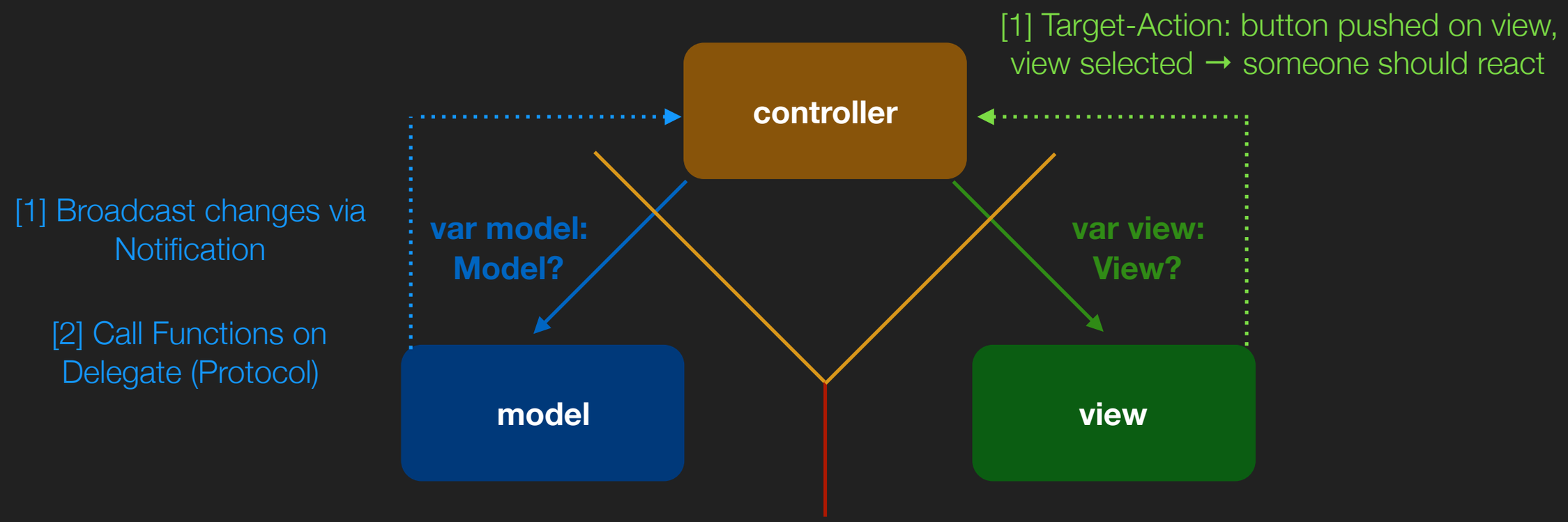
Auch hier verwendet die View lose gekoppelte Prinzipien für die Kommunikation zum Controller

[1] Broadcast changes via Notification

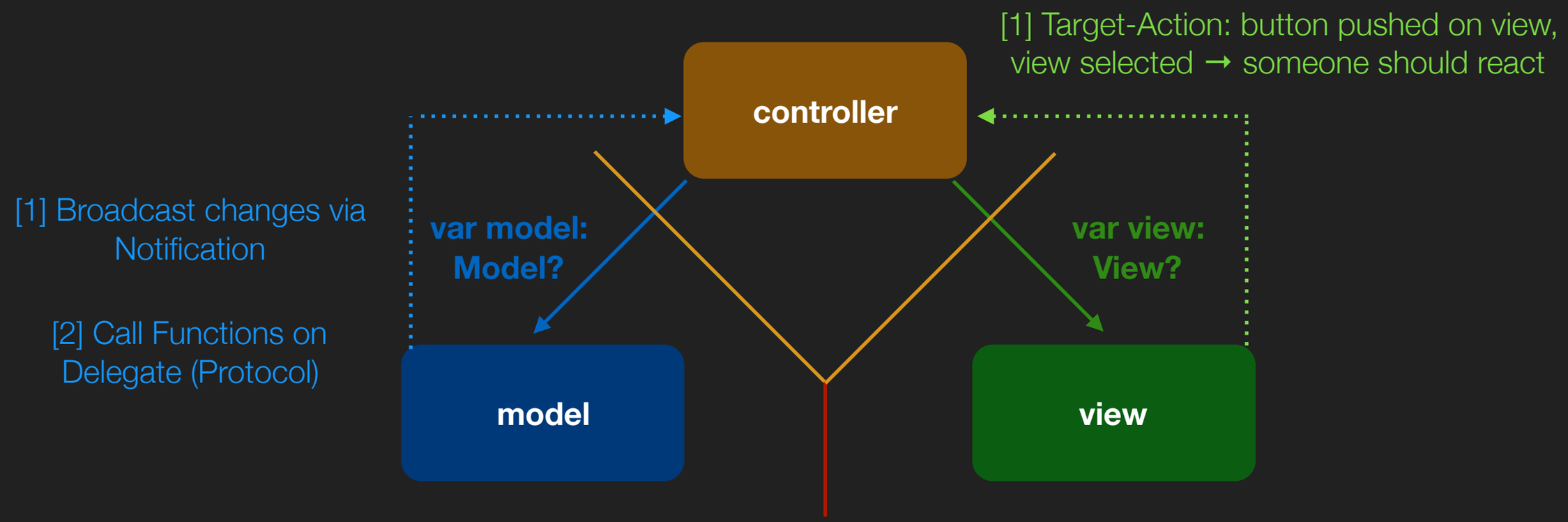
[2] Call Functions on Delegate (Protocol)



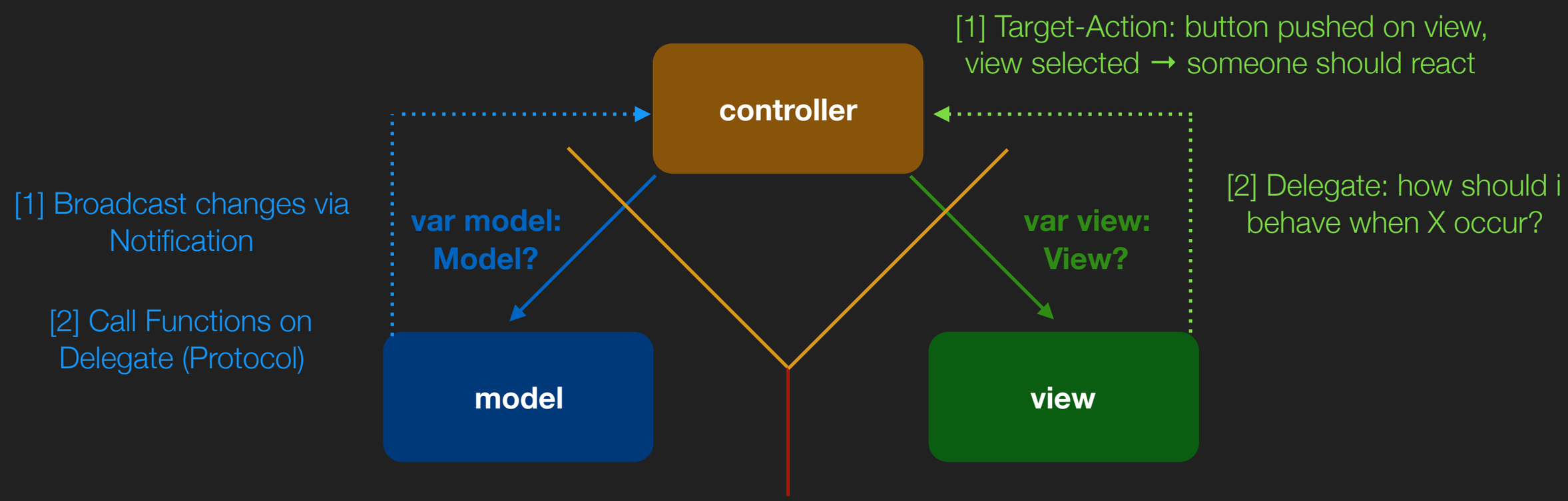
[1] Die generische View muss Interaktionen/Aktionen weiterleiten, weil sie nicht weiss, wie sie damit umgehen soll



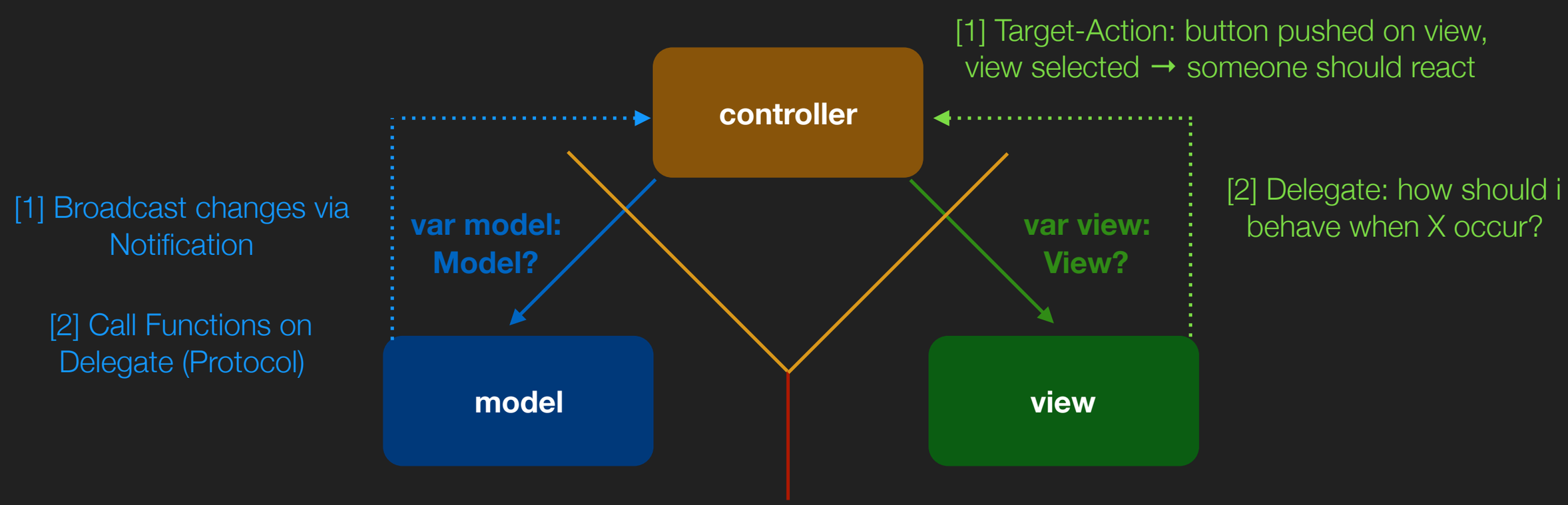
[1] Die View feuert auf ein bestimmtes Target (z.B. Controller) eine Action (Function), wenn ein bestimmtes Event stattfindet



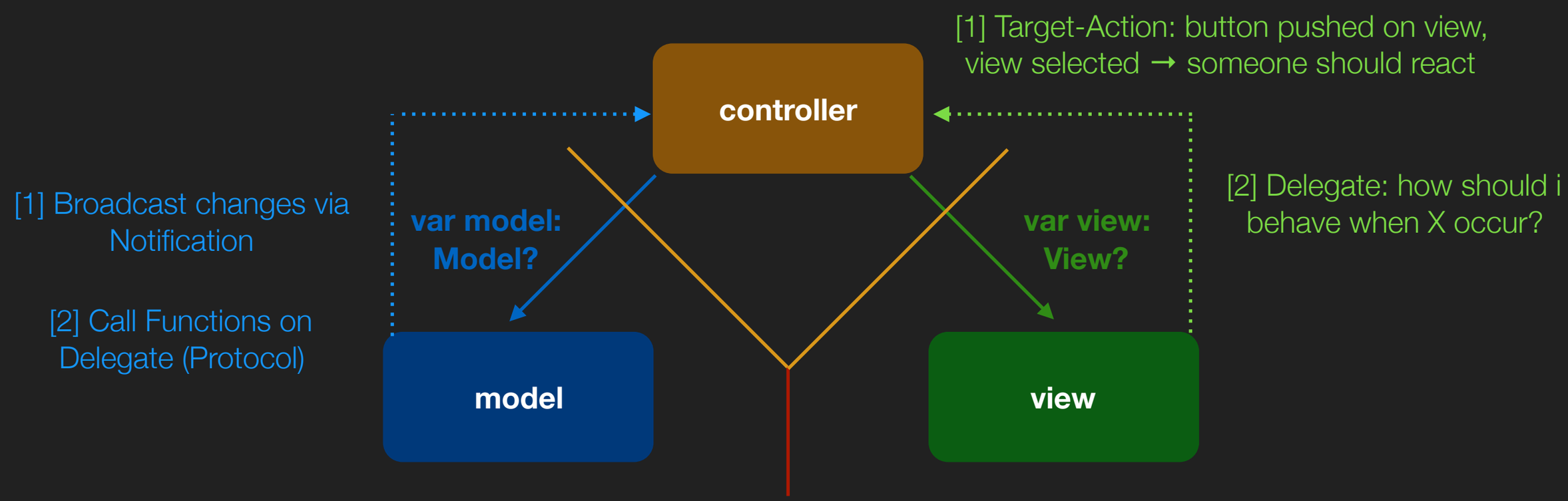
[2] Die generische View muss den Controller fragen, wie sie sich in bestimmten Situationen verhalten soll



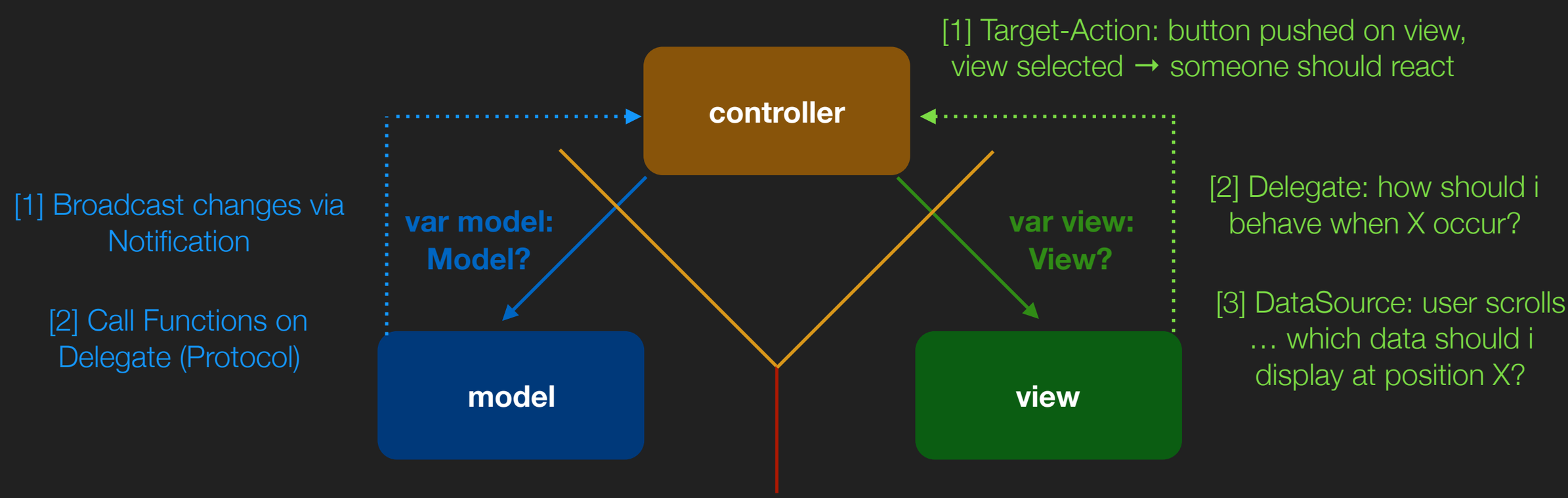
[2] Die View teilt über eine Delegate Protocol mit, dass eine bestimmte Situation eingetreten ist …



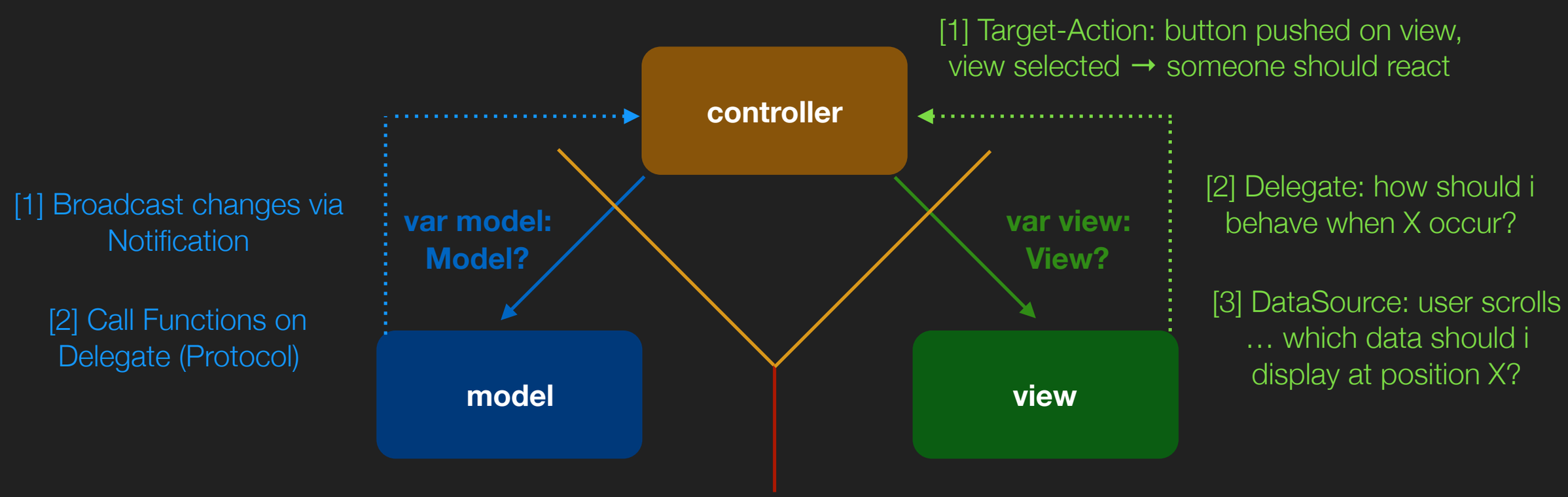
[2] … oder fragt nach wie sie sich in bestimmten Situation Verhalten soll, während der User mit der View interagiert



[3] Manchmal muss die View eine Datenquelle erhalten, um zu wissen, welche Informationen angezeigt werden sollen



[3] Auch hier fragt die View über ein DataSource Protocol an, welche Daten an welcher Stelle angezeigt werden sollen



Controller kennt Model und View. Model und View kennen sich weder untereinander noch den 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

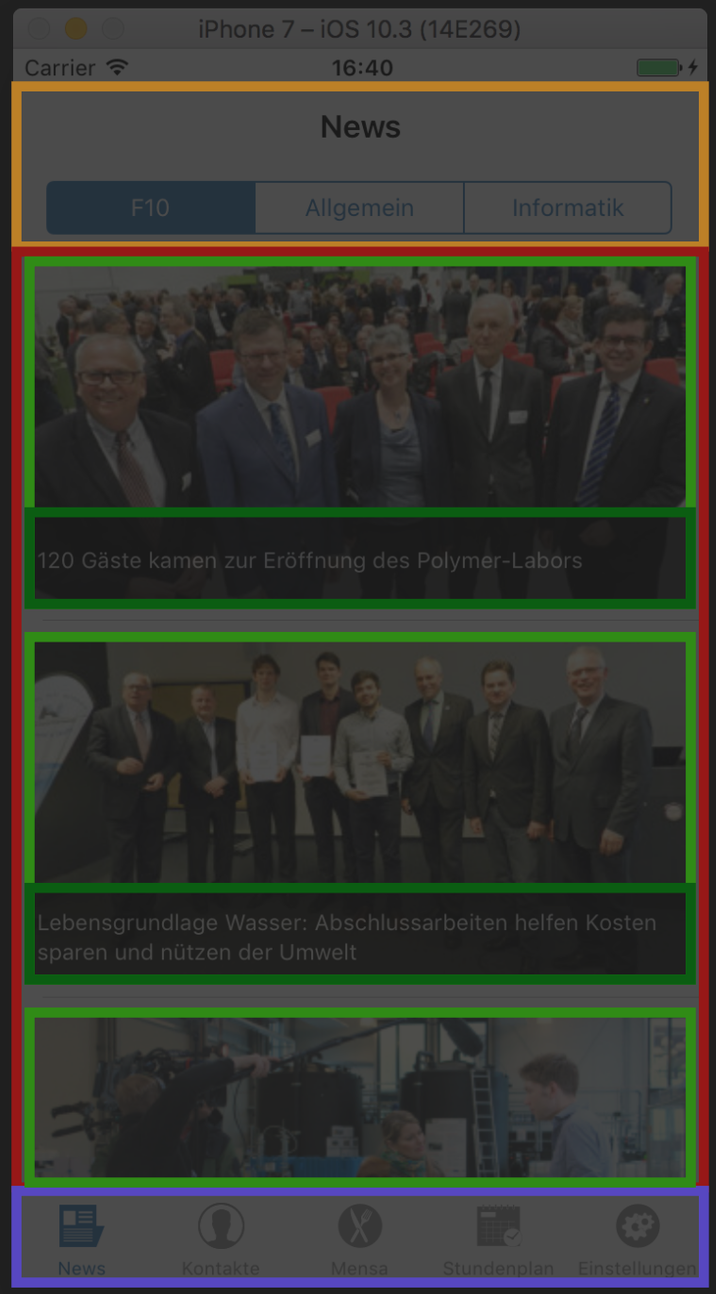
24.3.2017 - Unter den Hochschulen für Angewandte Wissenschaften hat die TH Köln in Deutschland eine Sonderstellung, nicht nur durch ihre Größe, sondern auch durch ihr breites Angebot im Bereich Chemie: Die TH Köln an ihren drei Standorten Köln, Leverkusen und Gummersbach über die komplette

News | Kontakte | Mensa | Stundenplan | Einstellungen

iPhone 7 – iOS 10.3 (14E269)

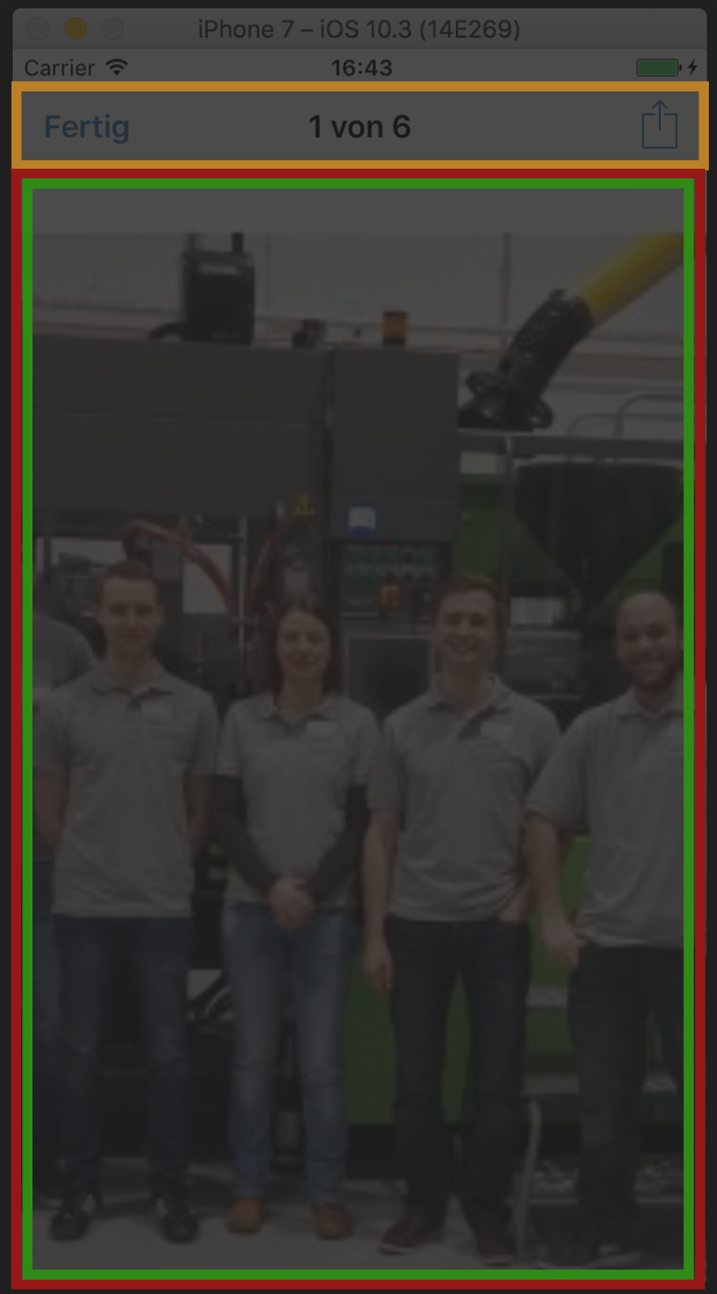
Carrier 16:43

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

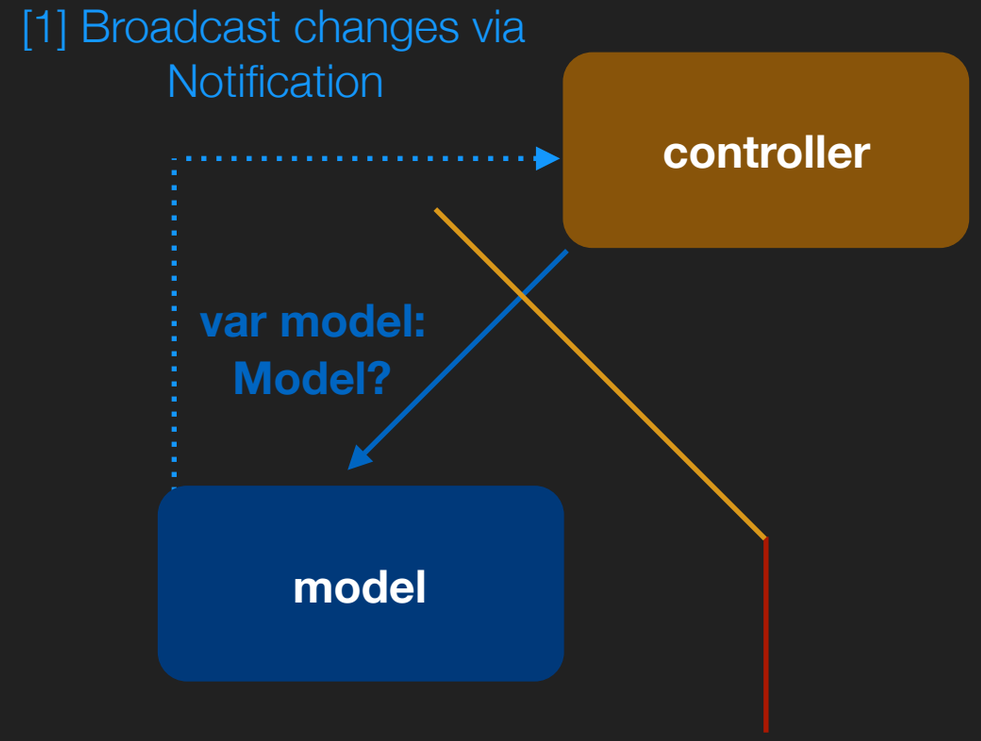
Heute

Model-View-Controller (MVC)

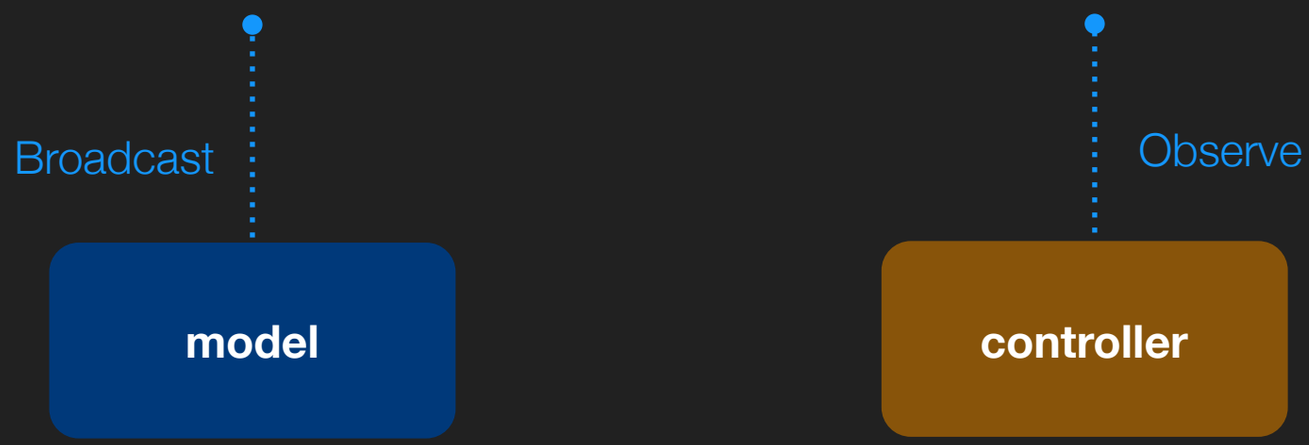
NotificationCenter, Target-Action, Delegate, DataSource

Zusammenfassung

NotificationCenter



NotificationCenter



- Create/Get Notification.Name
- Post Notification by Name [with Payload]

NotificationCenter

```swift
extension NSNotification.Name { // create own topic of type 'Notification.Name'
    static let MyModelDidChange = Notification.Name("MyModelDidChange")
}

struct MyModel {

    func modelHasChanged(to data: Data) {
        NotificationCenter.default.post( // post (broadcast) notification
            name: NSNotification.Name.MyModelDidChange, // under which topic to broadcast
            object: self, // sender
            userInfo: ["DataKey": data] // payload, accessible by 'DataKey'
        )
    }
}
```

NotificationCenter

Broadcast

model

- Create/Get Notification.Name
- Post Notification by Name [with Payload]

Observe

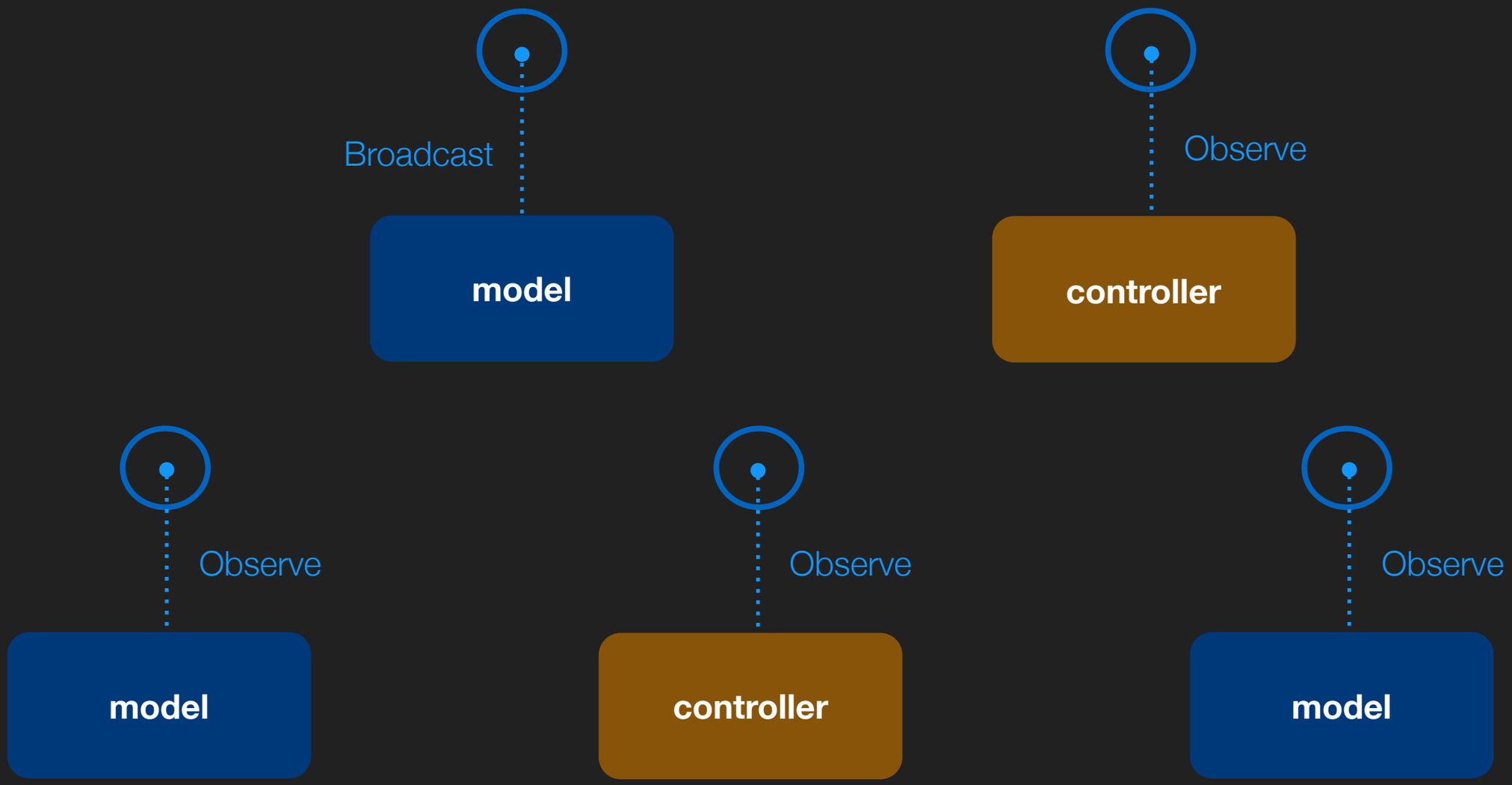
controller

- Start observing Notification for Name [by sender] [on queue]
- Stop observing Notification

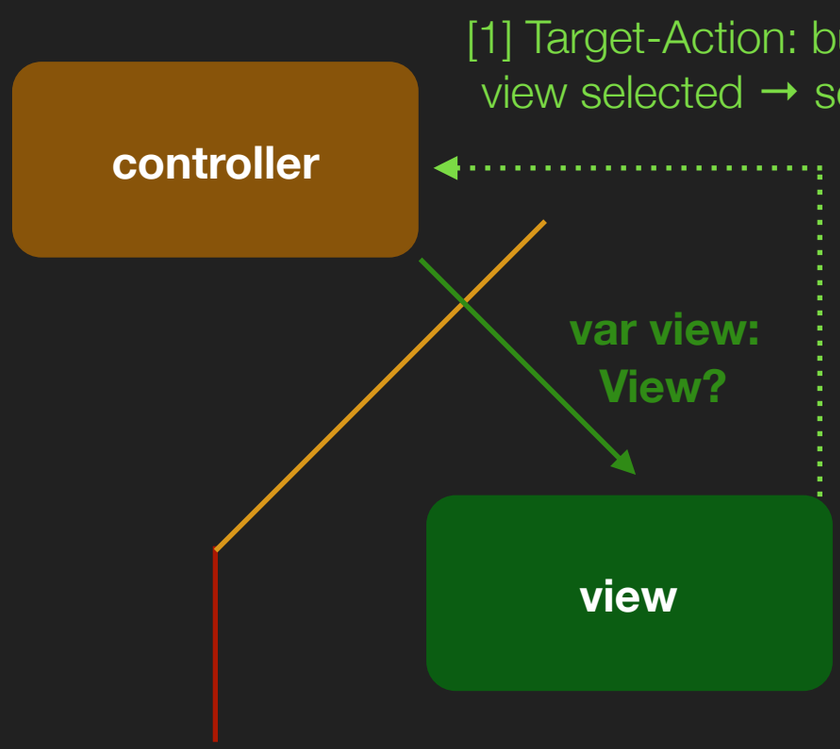
NotificationCenter

```swift
class SomeViewController: UIViewController {

    var observer: NSObjectProtocol? // 'cookie' of observation. used for unobserving when finished

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        observer = NotificationCenter.default.addObserver(
            forName: .MyModelDidChange, // topic to observe for
            object: nil, // who should be the sender. nil for 'anyone'
            queue: OperationQueue.main, // queue where the following closure should be executed on. nil for 'same queue as sender'
            using: { notification in
                let data = notification.userInfo?["DataKey"] as? Data
                print(data ?? "no data")
            }
        )
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        NotificationCenter.default.removeObserver(observer!) // remove observation to prevent memory leaks
    }
}
```

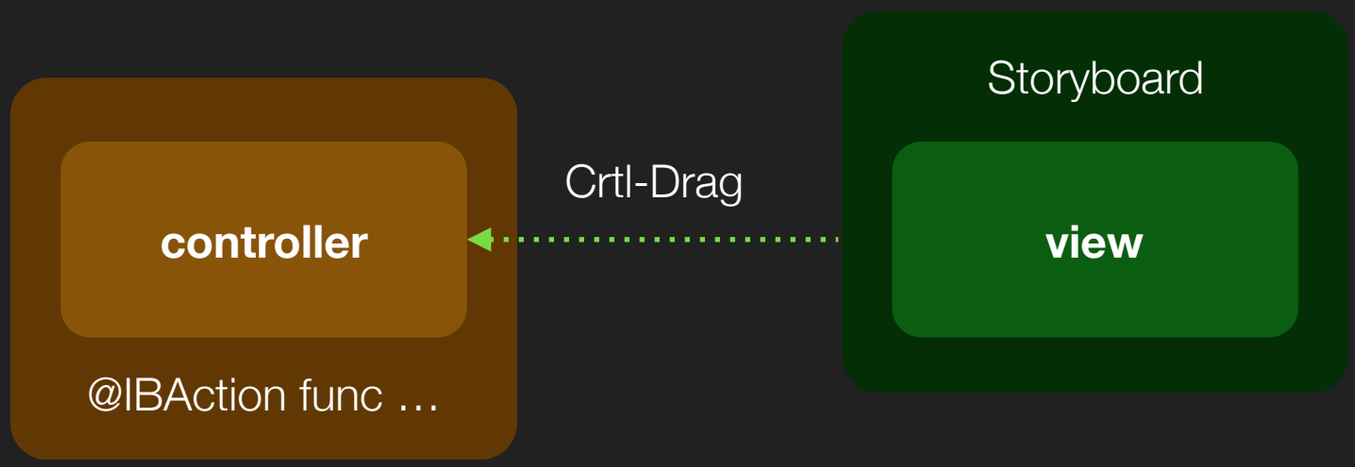
NotificationCenter



Target-Action



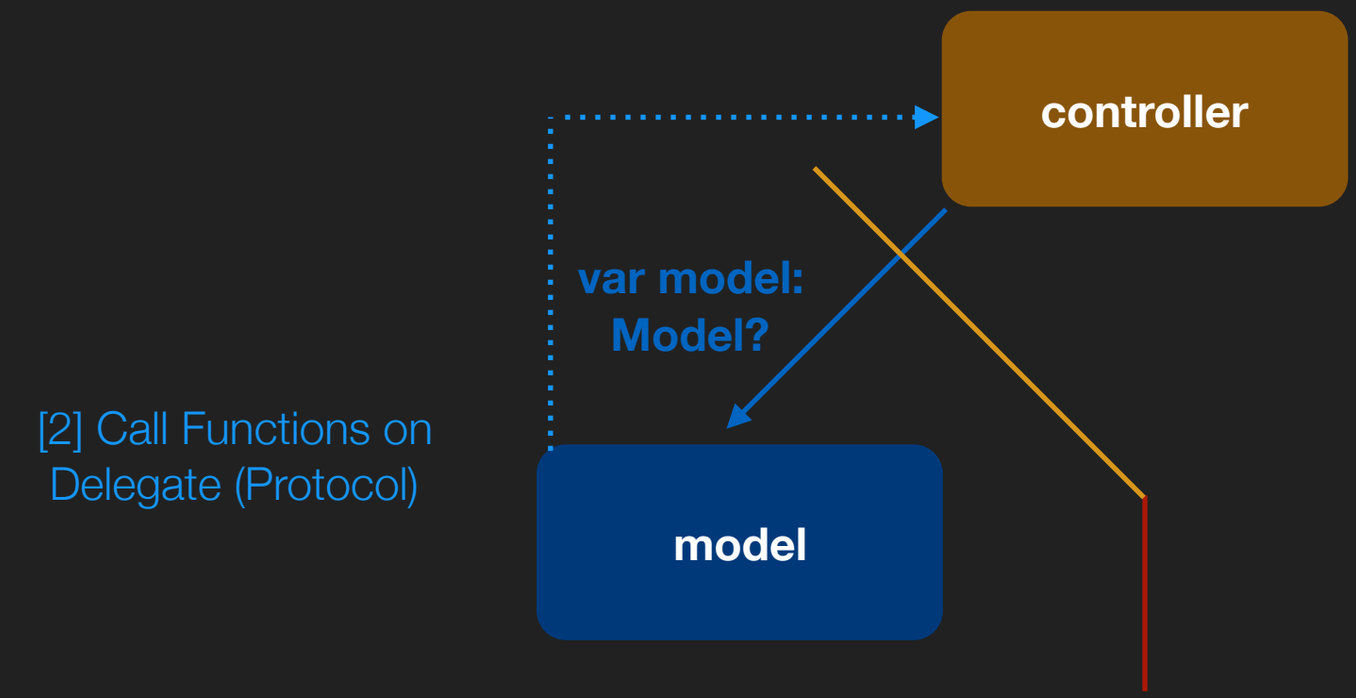
Target-Action



Target-Action

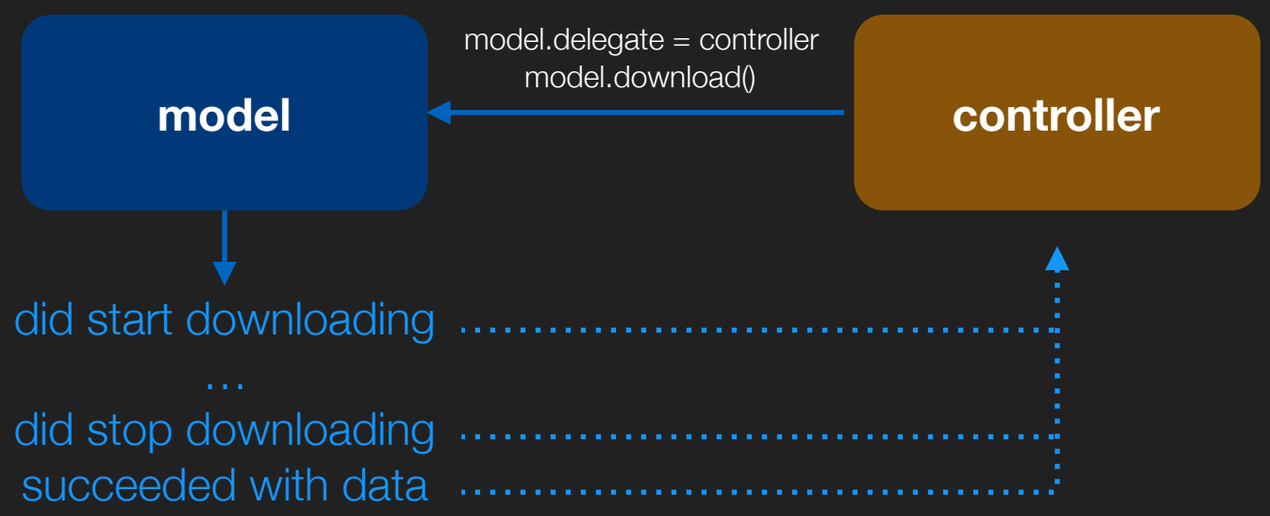
```
class TargetActionViewController: UIViewController {

    let button: UIButton = UIButton() // code version of IBOutlet

    override func viewDidLoad() {
        super.viewDidLoad()
        button.addTarget( // code version of 'crtl-drag' from storyboard to viewController
            self, // target
            action: #selector(buttonTapped(_:)), // function which will be executed
            for: .touchUpInside // action which will fire
        )
    }

    @objc func buttonTapped(_ sender: UIButton) { // code version of IBAction
        print("button tapped by target action")
    }
}
```

Delegate



[2] Call Functions on
Delegate (Protocol)

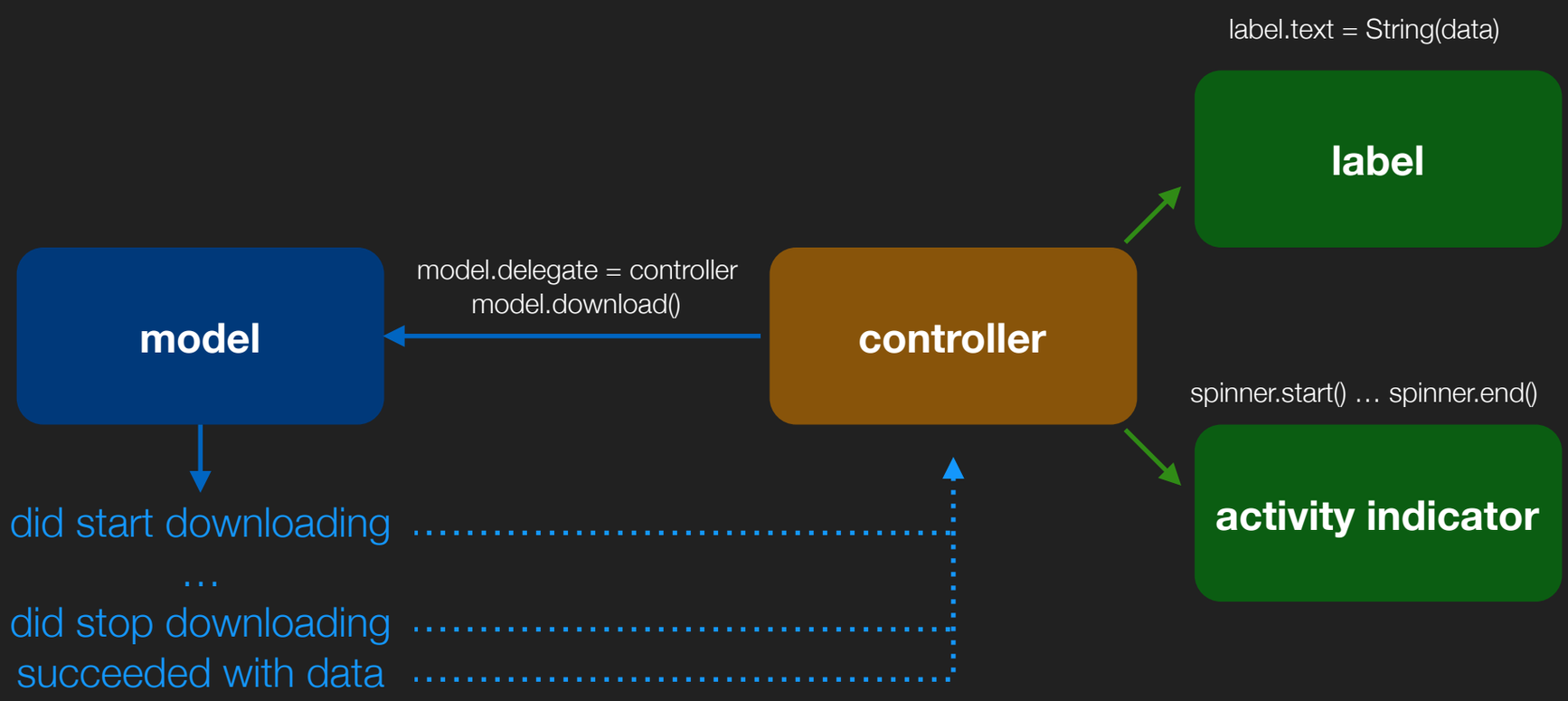
Delegate


model.delegate = controller
model.download()
model
controller
did start downloading
...
did stop downloading
succeeded with data

Delegate

```swift
protocol NetworkClientDelegate {
    func networkClientDidStartDownloading()
    func networkClientDidFinishDownloading()
    func networkClientSucceeded(with data: Data)
    func networkClientFailed(with error: Error)
}

class NetworkClient {
    var delegate: NetworkClientDelegate? // anyone who is interested in can set himself as a delegate

    func download() { // 'NetworkClient' will talk to his delegate (if set) whenever something happens
        delegate?.networkClientDidStartDownloading() // hey delegate, i did start downloading. do something if you will
        // downloading ....
        delegate?.networkClientDidFinishDownloading() // hey delegate, i finished downloading. do something if you will

        let error: Error? = nil // suppose there is no error ...
        let data: Data = Data() // ... but data

        if let error = error {
            delegate?.networkClientFailed(with: error) // hey delegate, an error occurred. do something if you will
        } else {
            delegate?.networkClientSucceeded(with: data) // hey delegate, there is your result. do something if you will
        }
    }
}
```

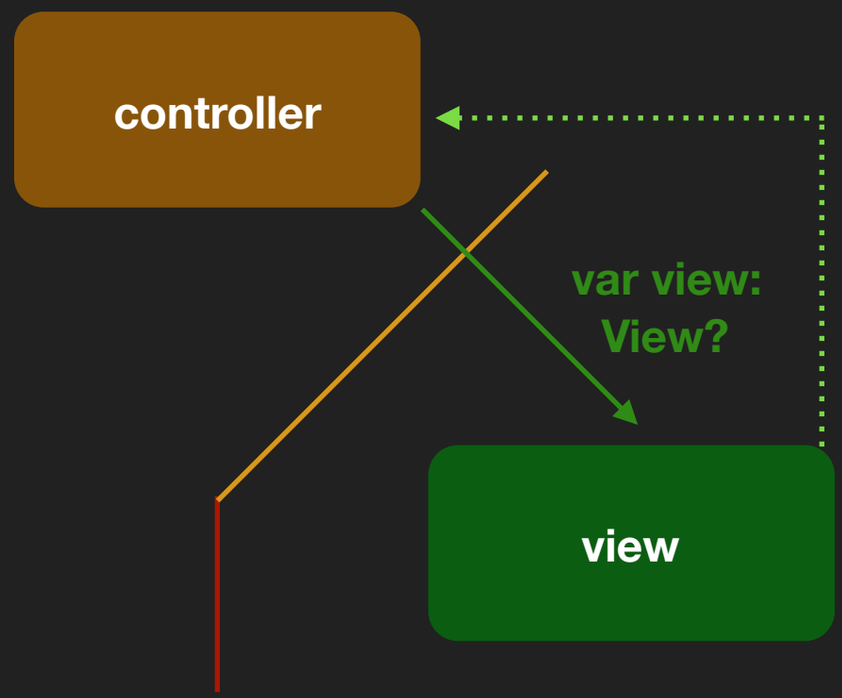
Delegate


label.text = String(data)
label
model.delegate = controller
model.download()
model
controller
spinner.start() … spinner.end()
activity indicator
did start downloading
…
did stop downloading
succeeded with data

Delegate

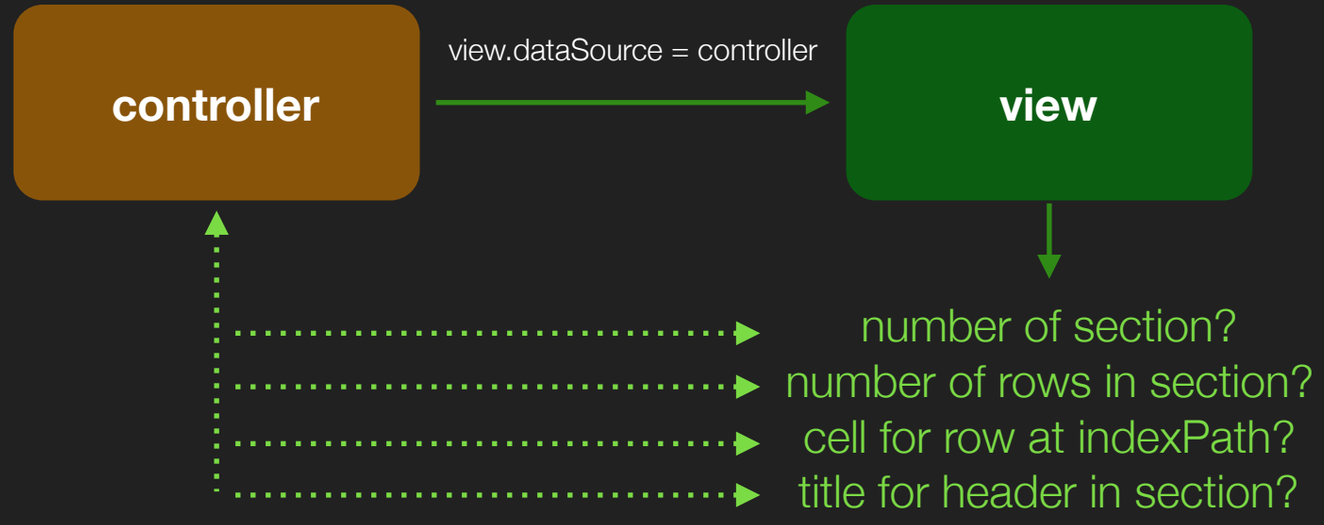
```
class DownloadViewController: UIViewController, NetworkClientDelegate {

    let networkClient = NetworkClient()
    let spinner = UIActivityIndicatorView()
    let label = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()

        networkClient.delegate = self // set my'self' to networkClient's delegate. i will get the calls now
        networkClient.download()
    }

    // have to implement them now ...
    func networkClientDidStartDownloading() {
        spinner.startAnimating() // show activity spinner when download starts
    }

    func networkClientDidFinishDownloading() {
        spinner.hidesWhenStopped = true
        spinner.stopAnimating() // hide activity spinner when download ends
    }

    func networkClientSucceeded(with data: Data) {
        label.text = String(describing: data)
    }

    func networkClientFailed(with error: Error) {
        print("show alert with \(error)")
    }
}
```

DataSource



[3] DataSource: user scrolls
… which data should i
display at position X?

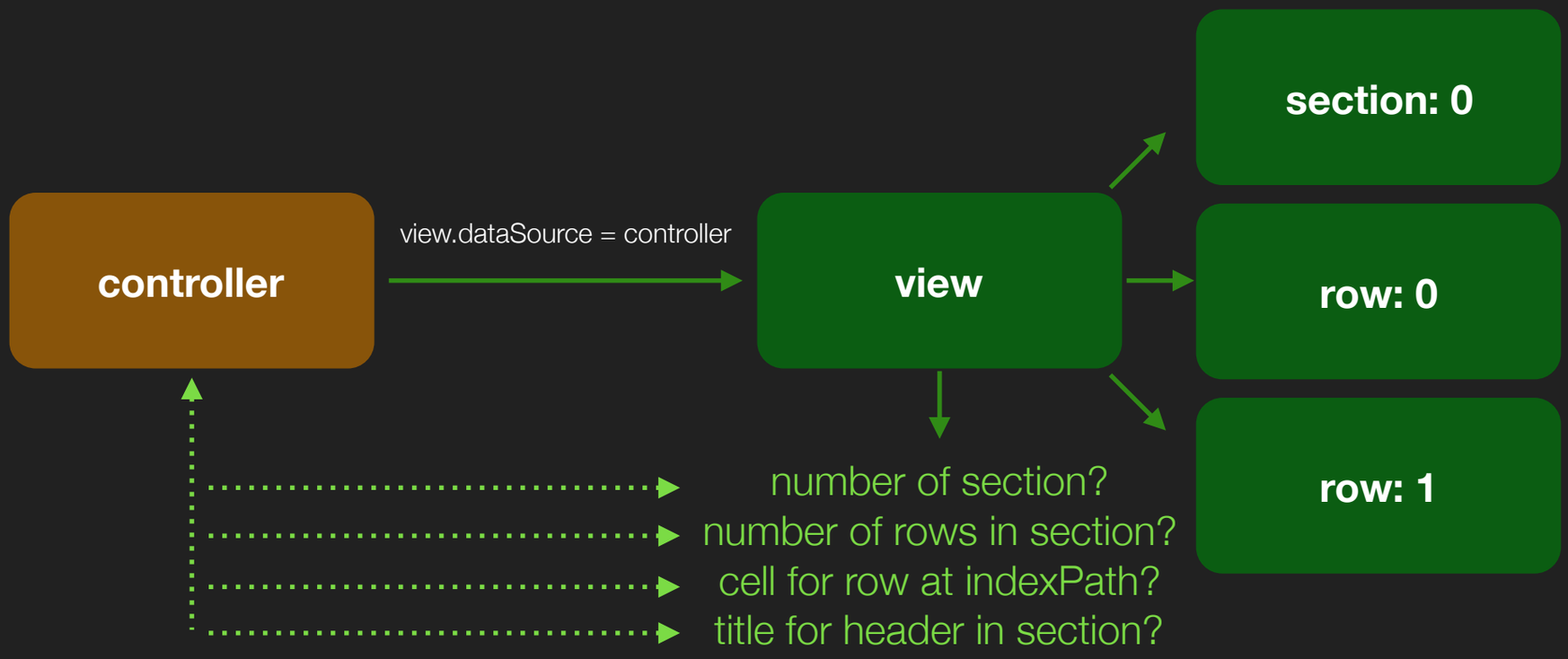
DataSource


controller
view.dataSource = controller
view
number of section?
number of rows in section?
cell for row at indexPath?
title for header in section?

DataSource

```
class MyTableViewController: UIViewController, UITableViewDataSource { // we have to implement 'UITableViewDataSource'

    let tableView: UITableView = UITableView()

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.dataSource = self // make 'self' respond to 'tableView'. thus, tableView will now called certain functions which 'self' can implement
    }

    func numberOfSections(in tableView: UITableView) -> Int { // hey dataSource, how many sections?
        return 1
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 2
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = UITableViewCell()
        cell.textLabel?.text = "row: \(indexPath.row)"
        return cell
    }

    func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
        return "section: \(section)"
    }
}
```

DataSource


section: 0
controller
view.dataSource = controller
view
row: 0
row: 1
number of section?
number of rows in section?
cell for row at indexPath?
title for header in section?

Heute

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

Zusammenfassung

Zusammenfassung

- Aufteilung in MVC und Reglementierung der Kommunikationsrichtung
  - Controller kennt Model und View
  - Model und View kennen sich weder untereinander noch den Controller
- Ein MVC-Stack repräsentiert i.d.R. einen Screen oder einen Teil des Screens (multiple MVCs)
- Model und View kommunizieren lose gekoppelt (“blind”)
  - NotificationCenter: 1 zu n broadcasting von Notifications zu einem Thema (Notification.Name.*) mit optionalem Payload (userInfo)
  - Target-Action: View befeuert das Target (z.B. Controller) wenn eine Action (z.B. Button touched) passiert
  - Delegate: View oder Model teilen dem Delegaten etwas mit oder Fragen nach Informationen
  - DataSource: View oder Model fragen die DataSource nach Daten die sie präsentieren sollen