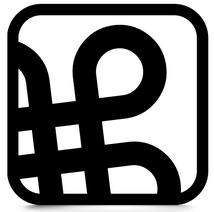


Macoun

Pflegeleichte Software

Christian Tietze
@ctietze

Wie fügt sich die Arbeit,
die ich heute erledigen werde,
ins Projekt ein?

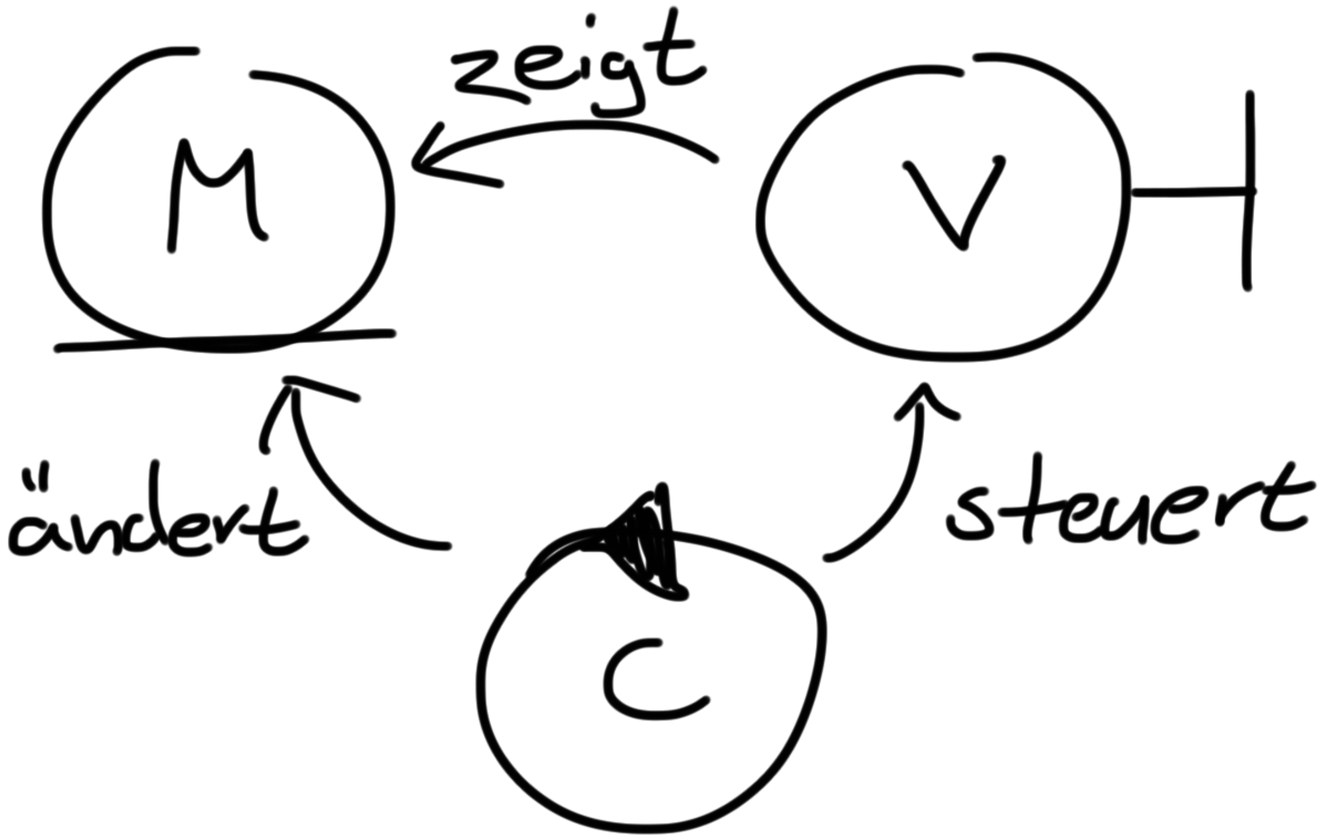
Ablauf

1. Ungepflegte Software
2. Struktur — *Layered Architecture*
3. Prozess — *ReSwift*
4. Anwendung

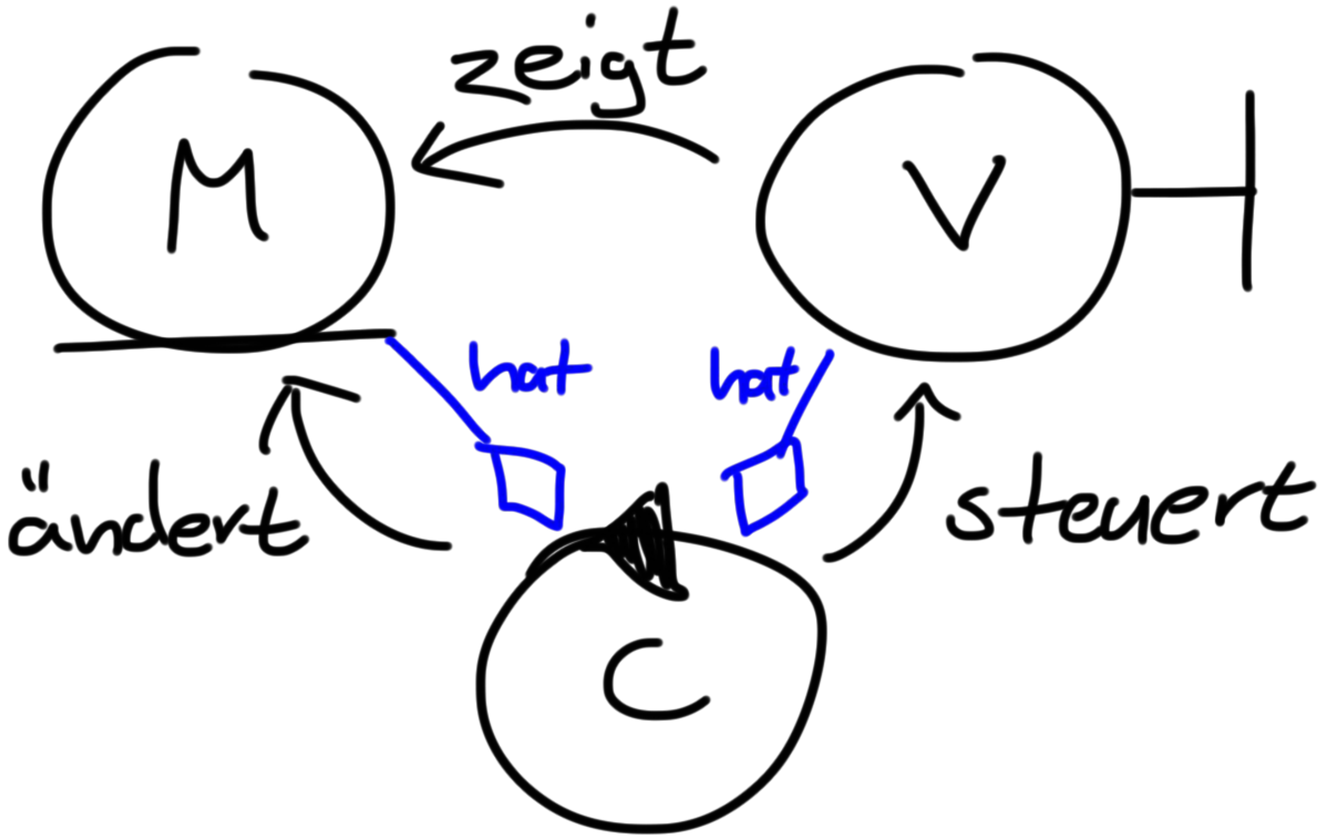
1. Ungepflegte Software

Massive View Controller

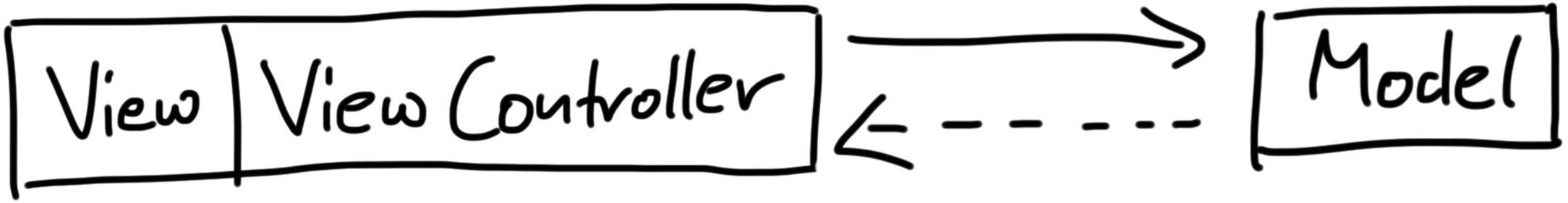
»MVC Sucks!«



»MVC Sucks!«

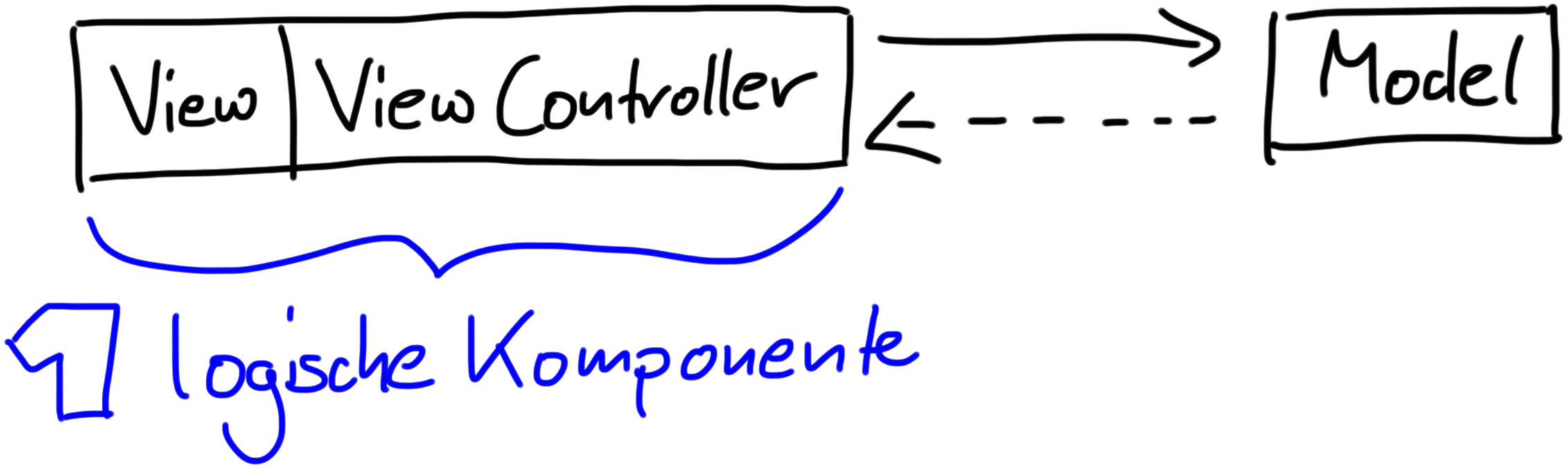


»MVC Sucks!«



Ash Furrow https://www.objc.io/issues/13-architecture/mvvm/

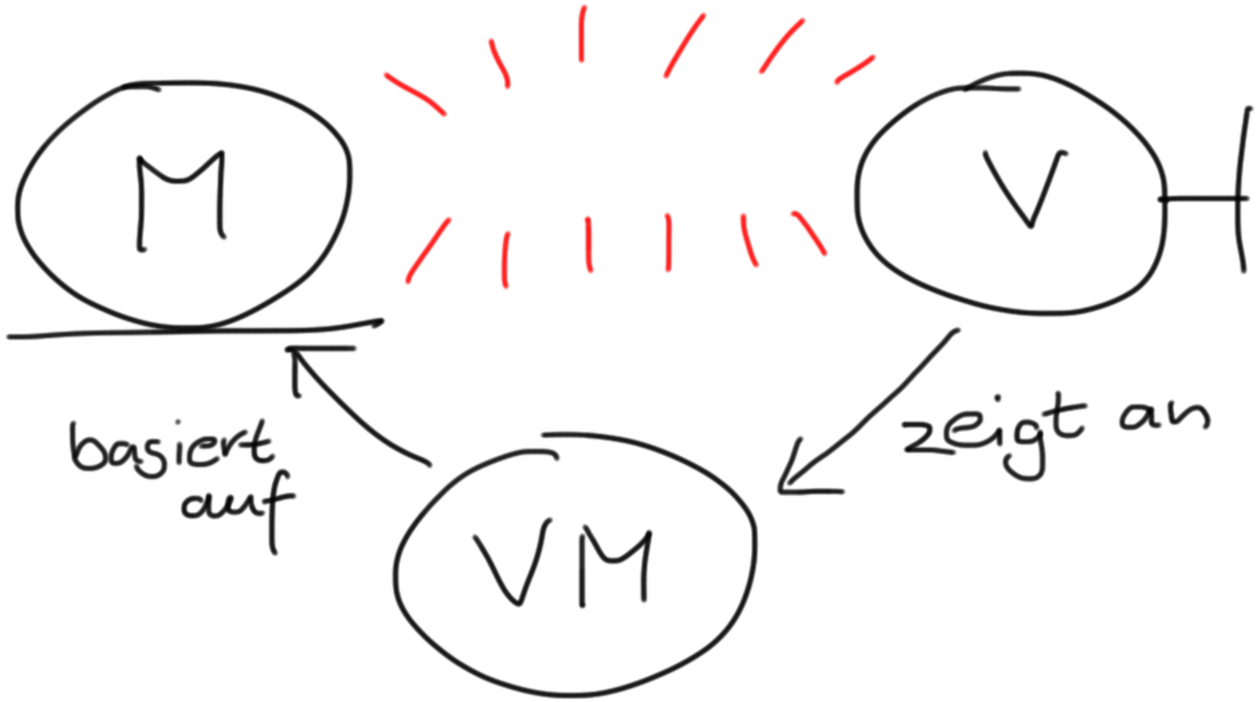
»MVC Sucks!«



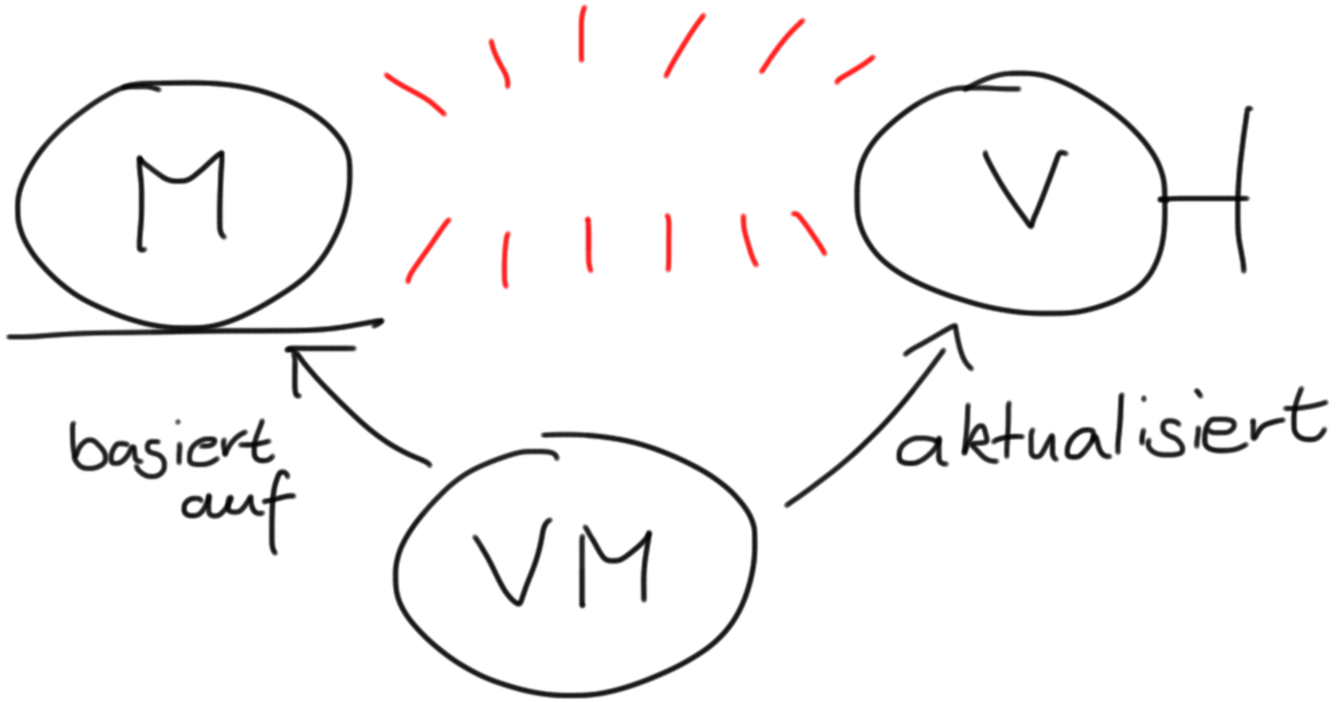
Ash Furrow https://www.objc.io/issues/13-architecture/mvvm/

Cargo-Kulte

MVVM



MVVM



MVVM

»… is not very good.«
— Soroush Khanlou
http://khanlou.com/2015/12/mvvm-is-not-very-good/

»… is exceptionally ok.«
— Ash Furrow
https://ashfurrow.com/blog/mvvm-is-exceptionally-ok/

»… is quite okay at what it's supposed to do.«
— Ich 😅
https://christiantietze.de/posts/2016/08/mvvm-is-okay-for-what-it-does/

VIPER

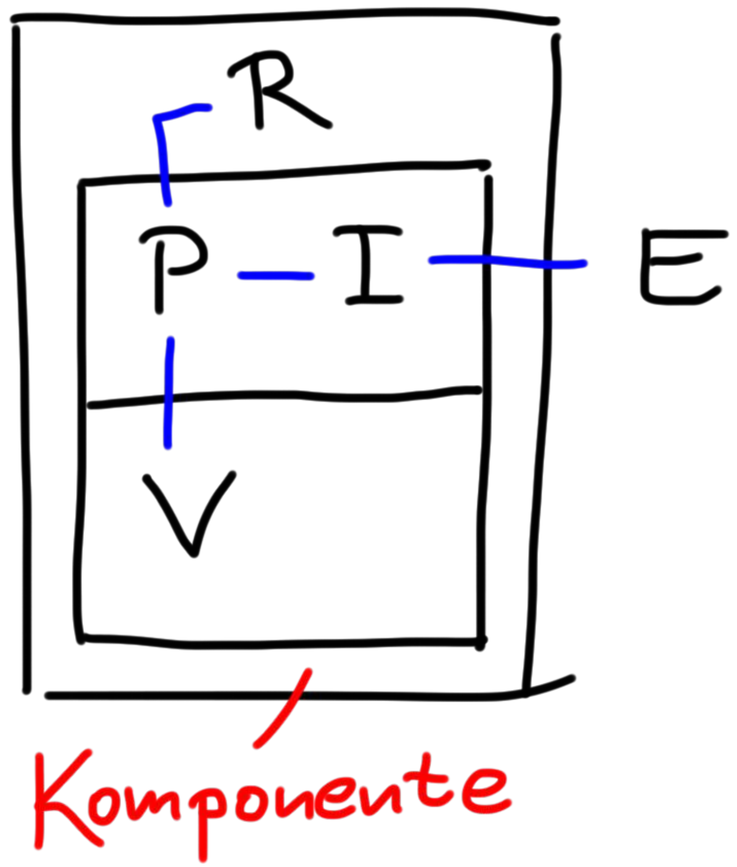
View

Interactor

Presenter

Entity

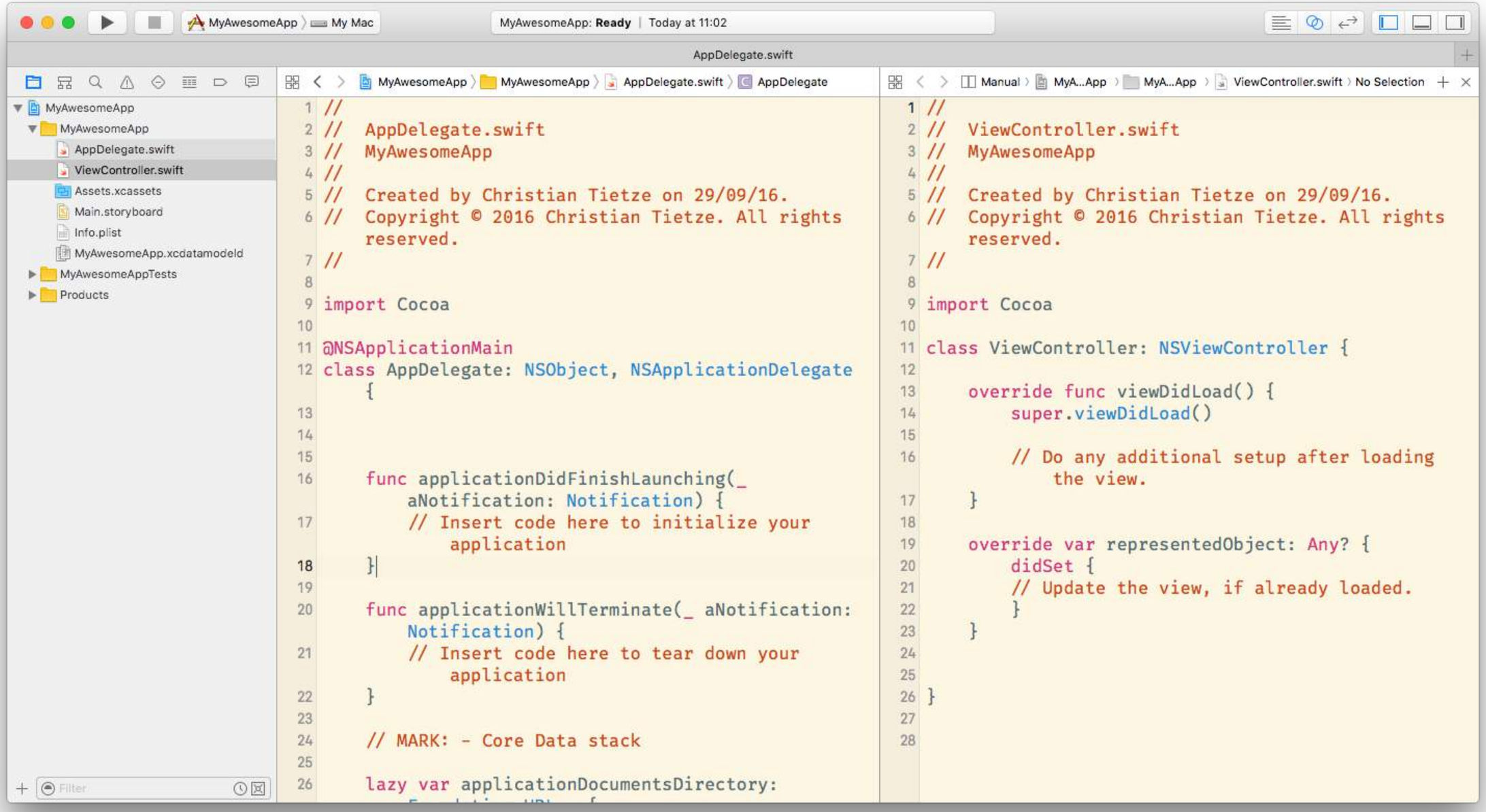
Routing

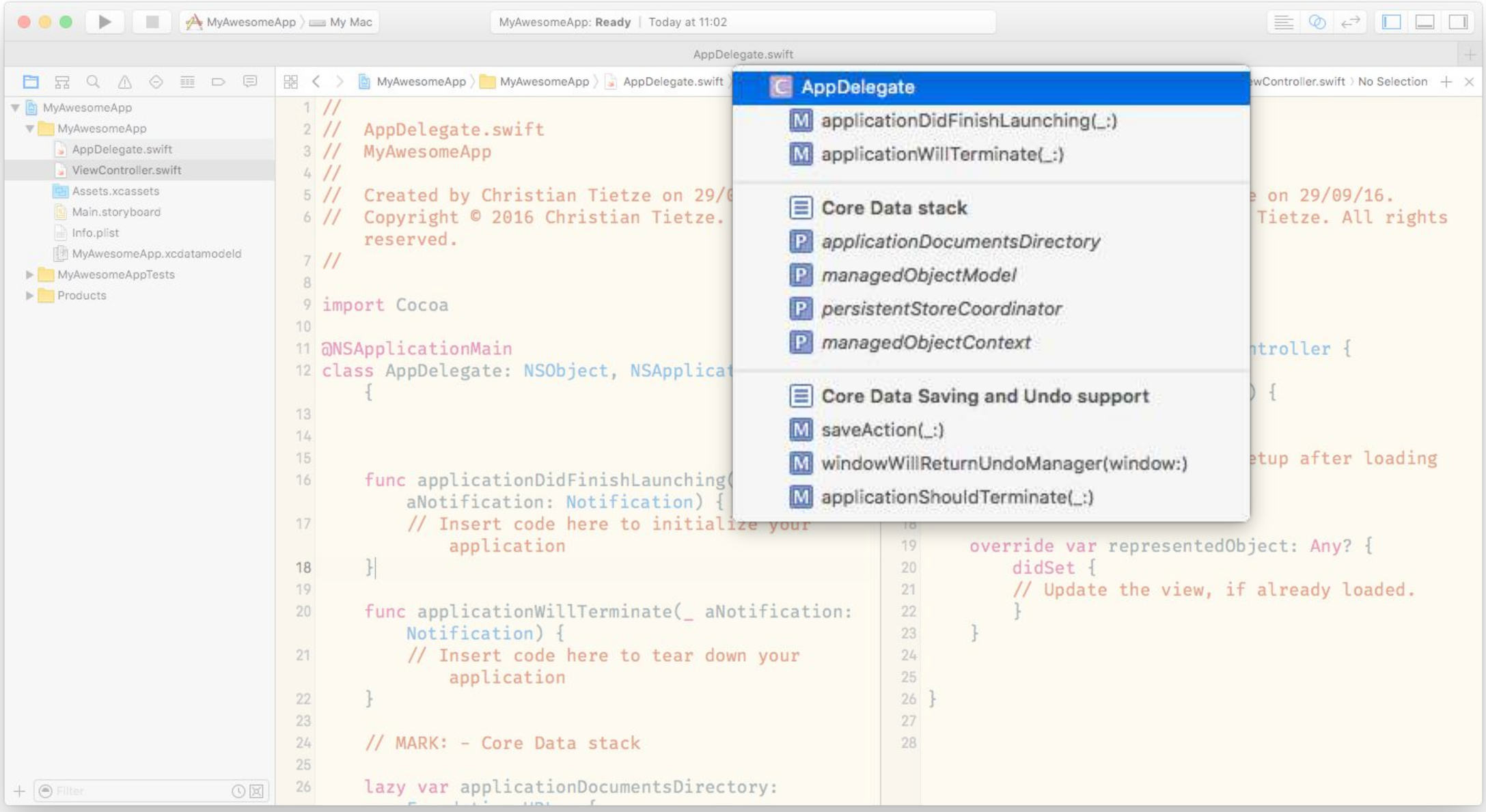


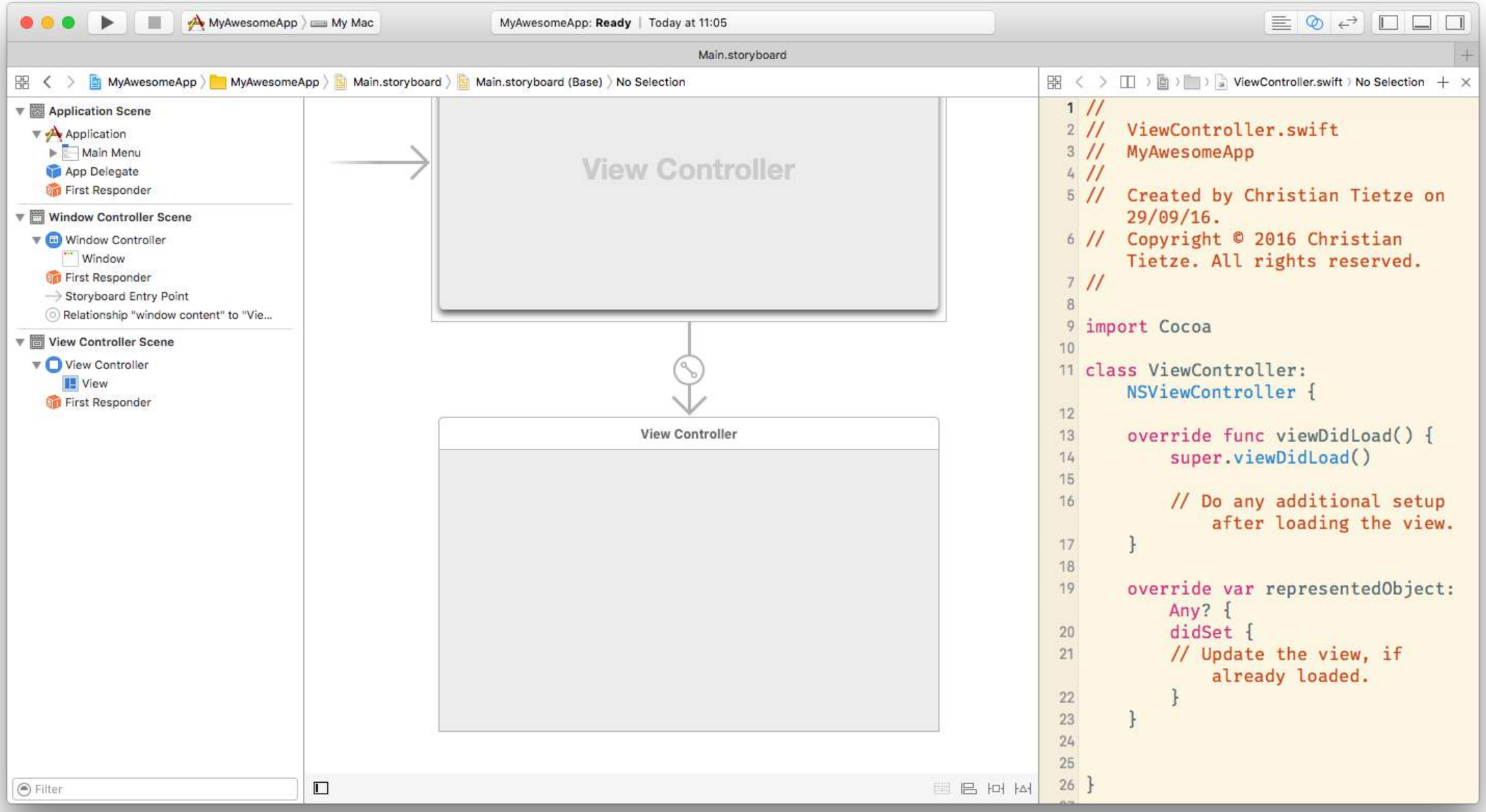
Heilsversprechen

- *»write once«* ist Quatsch
- Code ist Prosa
  - Lesbarkeit
  - Verständlichkeit
  - *mental mapping*

2. Struktur


MyAwesomeApp 〉 My Mac
MyAwesomeApp: Ready | Today at 11:02
AppDelegate.swift
MyAwesomeApp
MyAwesomeApp
AppDelegate.swift
ViewController.swift
Assets.xcassets
Main.storyboard
Info.plist
MyAwesomeApp.xcdatamodeld
MyAwesomeAppTests
Products
MyAwesomeApp 〉 MyAwesomeApp 〉 AppDelegate.swift 〉 AppDelegate
//
// AppDelegate.swift
// MyAwesomeApp
//
// Created by Christian Tietze on 29/09/16.
// Copyright © 2016 Christian Tietze. All rights reserved.
//
import Cocoa
@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {
func applicationDidFinishLaunching(_ aNotification: Notification) {
// Insert code here to initialize your application
}
func applicationWillTerminate(_ aNotification: Notification) {
// Insert code here to tear down your application
}
// MARK: - Core Data stack
lazy var applicationDocumentsDirectory:
Manual 〉 MyA...App 〉 MyA...App 〉 ViewController.swift 〉 No Selection
//
// ViewController.swift
// MyAwesomeApp
//
// Created by Christian Tietze on 29/09/16.
// Copyright © 2016 Christian Tietze. All rights reserved.
//
import Cocoa
class ViewController: NSViewController {
override func viewDidLoad() {
super.viewDidLoad()
// Do any additional setup after loading the view.
}
override var representedObject: Any? {
didSet {
// Update the view, if already loaded.
}
}
}
Filter


MyAwesomeApp 〉 My Mac
MyAwesomeApp: Ready | Today at 11:02
AppDelegate.swift
MyAwesomeApp
MyAwesomeApp
AppDelegate.swift
ViewController.swift
Assets.xcassets
Main.storyboard
Info.plist
MyAwesomeApp.xcdatamodeld
MyAwesomeAppTests
Products
MyAwesomeApp 〉 MyAwesomeApp 〉 AppDelegate.swift
ewController.swift 〉 No Selection
AppDelegate
applicationDidFinishLaunching(_:)
applicationWillTerminate(_:)
Core Data stack
applicationDocumentsDirectory
managedObjectModel
persistentStoreCoordinator
managedObjectContext
Core Data Saving and Undo support
saveAction(_:)
windowWillReturnUndoManager(window:)
applicationShouldTerminate(_:)
//
// AppDelegate.swift
// MyAwesomeApp
//
// Created by Christian Tietze on 29/0
// Copyright © 2016 Christian Tietze.
reserved.
//
import Cocoa
@NSApplicationMain
class AppDelegate: NSObject, NSApplicat
{
func applicationDidFinishLaunching(
aNotification: Notification) {
// Insert code here to initialize your
application
}
func applicationWillTerminate(_ aNotification:
Notification) {
// Insert code here to tear down your
application
}
// MARK: - Core Data stack
lazy var applicationDocumentsDirectory:
e on 29/09/16.
Tietze. All rights
ntroller {
) {
etup after loading
override var representedObject: Any? {
didSet {
// Update the view, if already loaded.
}
}
}


MyAwesomeApp 〉 My Mac
MyAwesomeApp: Ready | Today at 11:05
Main.storyboard
MyAwesomeApp 〉 MyAwesomeApp 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection
Application Scene
Application
Main Menu
App Delegate
First Responder
Window Controller Scene
Window Controller
Window
First Responder
Storyboard Entry Point
Relationship "window content" to "Vie...
View Controller Scene
View Controller
View
First Responder
Filter
View Controller
View Controller
ViewController.swift 〉 No Selection
1 //
2 // ViewController.swift
3 // MyAwesomeApp
4 //
5 // Created by Christian Tietze on 29/09/16.
6 // Copyright © 2016 Christian Tietze. All rights reserved.
7 //
8
9 import Cocoa
10
11 class ViewController: NSViewController {
12
13 override func viewDidLoad() {
14 super.viewDidLoad()
15
16 // Do any additional setup after loading the view.
17 }
18
19 override var representedObject: Any? {
20 didSet {
21 // Update the view, if already loaded.
22 }
23 }
24
25
26 }

Das *Was-kommt-in-den-View-Controller* Spiel

- Core Data `perform(_:)`
- `URLRequest`, `URLSession` & Callbacks
- Dateioperationen
- `UserDefaults`

Heuristiken

a.k.a.

Die 7 Killer-Tipps für Bug-freie Controller

Code-Heuristiken

- kurze Klassen (~100 Zeilen)
- kurze Methoden (~5 Zeilen)
- <4 Parameter pro Methode
- Symmetrie

```
func eatBanana(banana: Banana) {
  let peeledBanana = banana.peeled
  guard isEdible(peeledBanana)
    else { return }
  self.assimilateNutrients(peeledBanana)
}
```

```
func eatBanana(banana: Banana) {
  self.prepare(food: banana)
    |> self.checkEdibility
    |> self.assimilate
}
```

Ziel: Objektkohäsion

Heuristik: *Single Responsibility Principle*

- Methoden passen nicht zueinander
- Nachrichtenpfade konfus

Kohäsionsarten

- Zufällig
- Logisch
- Zeitlich

} Code

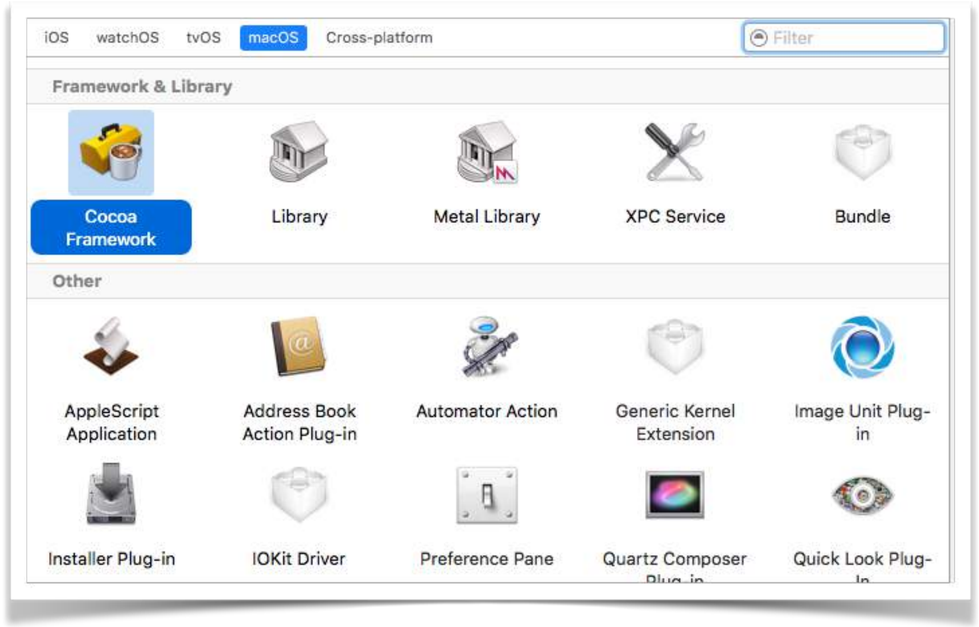
- *communicational*
- Sequenziell
- Funktional

} Problem(domäne)

Edward Yourdon: *Structured Design*, 1979, S. 108–132

Modulebene

- (Swift) Frameworks
- API Design



S

O

L

I

D

Robert C. Martin: *Design Principles and Design Patterns*, 2000

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Robert C. Martin: *Design Principles and Design Patterns*, 2000

Single Responsibility Principle

`open`/`final` in Swift 3

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Robert C. Martin: *Design Principles and Design Patterns*, 2000

Wie lerne ich,
Probleme zu erkennen,
wenn sie auftreten?

Modellieren

Problemdomäne

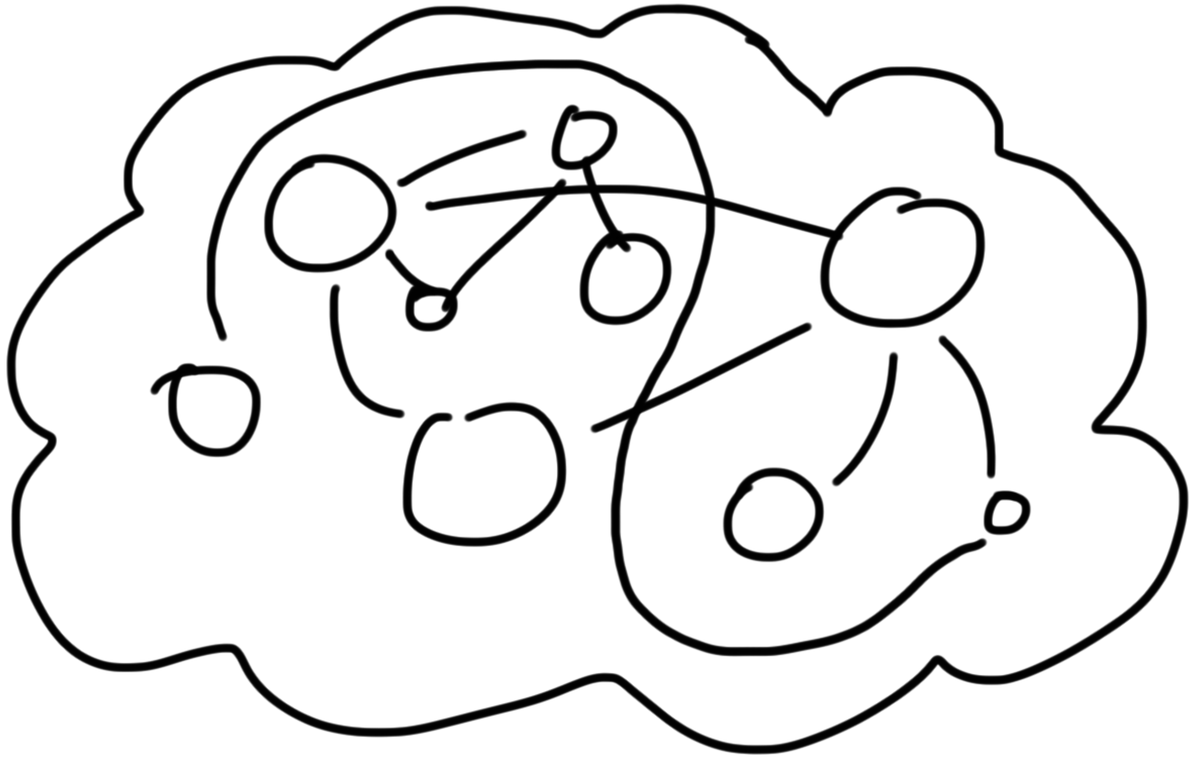
Code

Softwarearchitektur

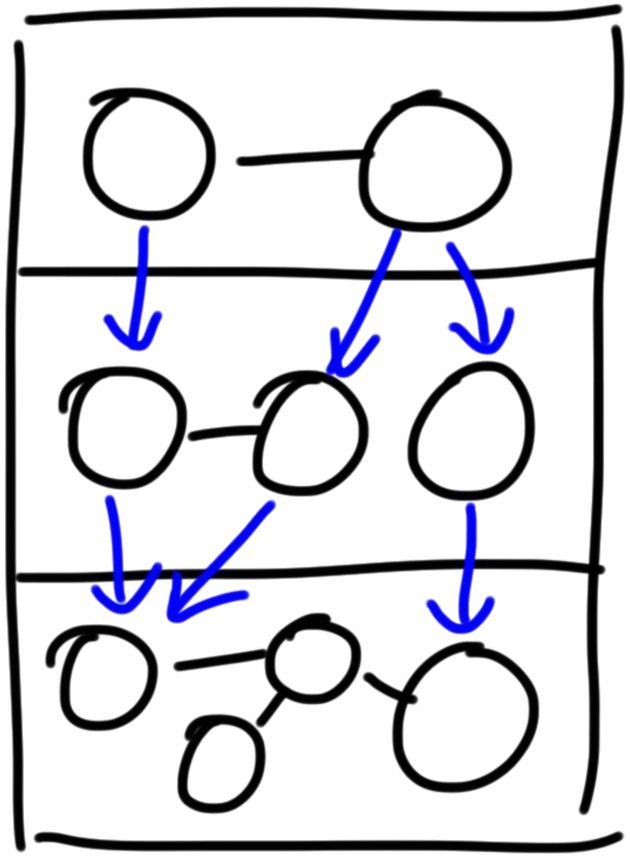
Programmieren

Layered Architecture

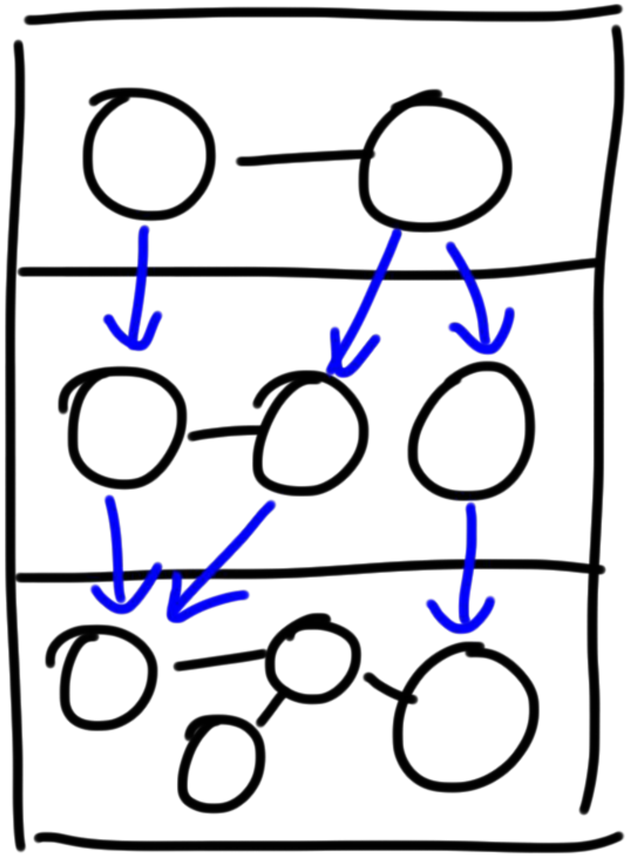
Big Ball of Mud



Schichtenarchitektur

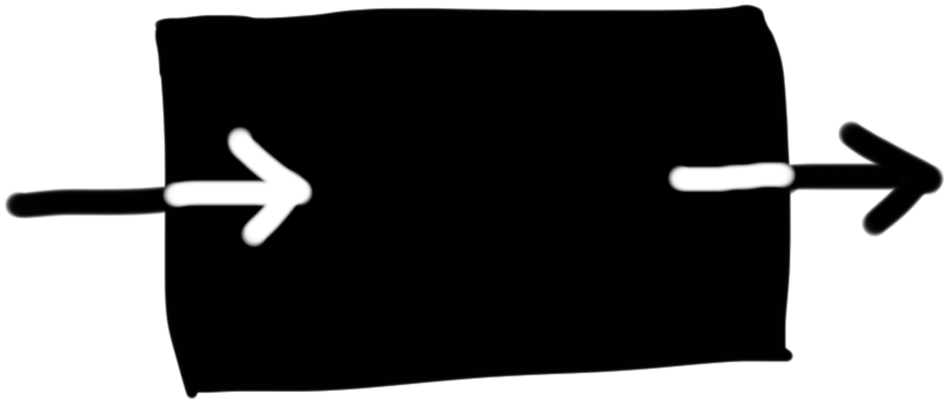


Schichtenarchitektur



- Gruppieren in Modulen
- Module in Schichten anordnen
- nur eine Richtung erlaubt

Black Box



- Grenze
- Input
- Output

4 Bewährte Ebenen

Model	Daten/Zustand
User Interface	AppKit/UIKit-Komponenten
Infrastruktur	Kontext, Außenwelt (I/O, Web)
»Die App«	Zusammenspiel

SOLID auf Schichtebene

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

SOLID auf Schichtebene

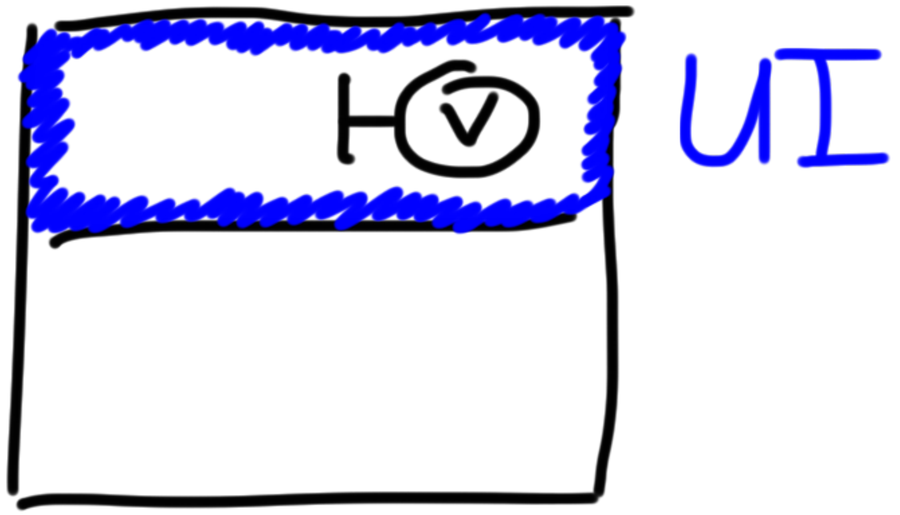
Open/Closed Principle
Liskov Substitution Principle

Objektdesign

Single Responsibility Principle
Interface Segregation Principle
Dependency Inversion Principle

Modulebene

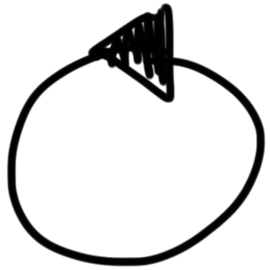
SRP auf Schichtebene

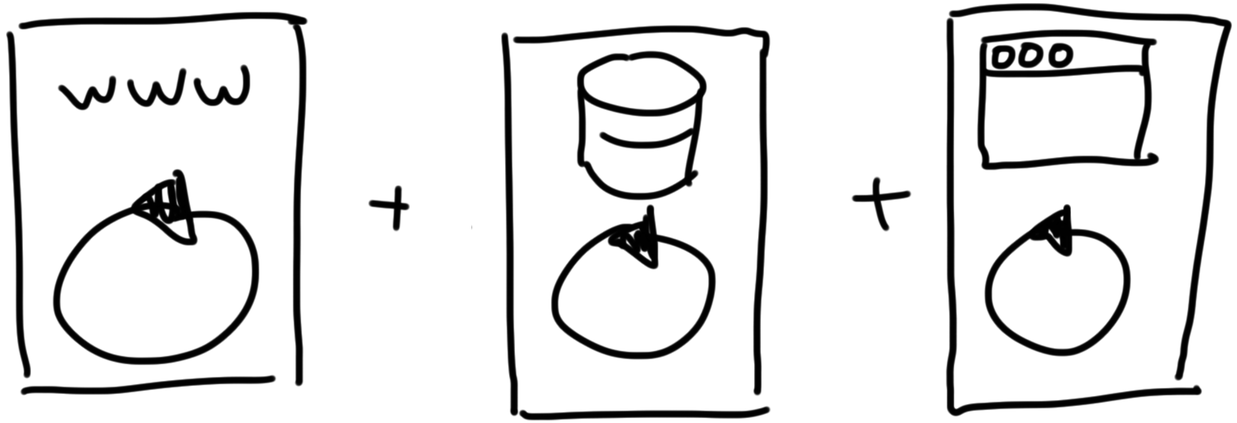


- Modulebene: Verantwortlichkeit des Moduls/der Schicht selbst
- Objektebene im Modul: Kontext der besitzenden Schicht

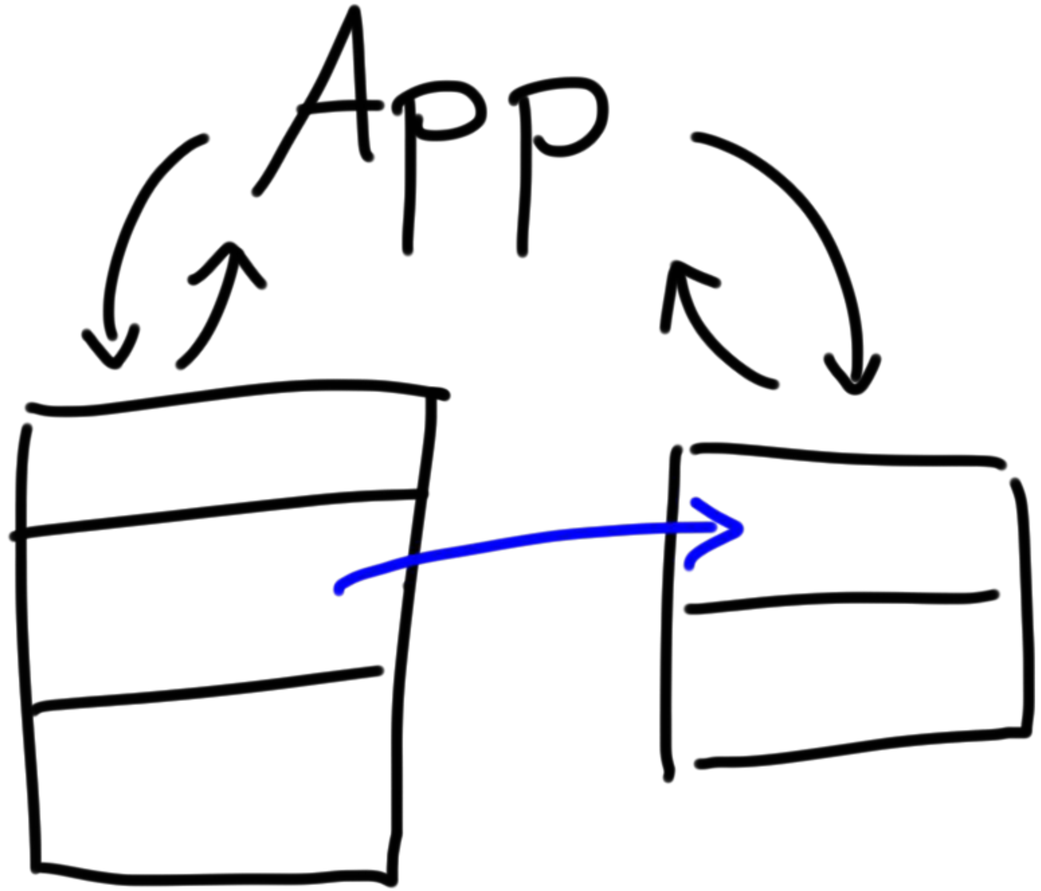
ServerDataSyncer

- JSON-Daten anfordern
- Model-Speicher aktualisieren
- neue Inhalte anzeigen

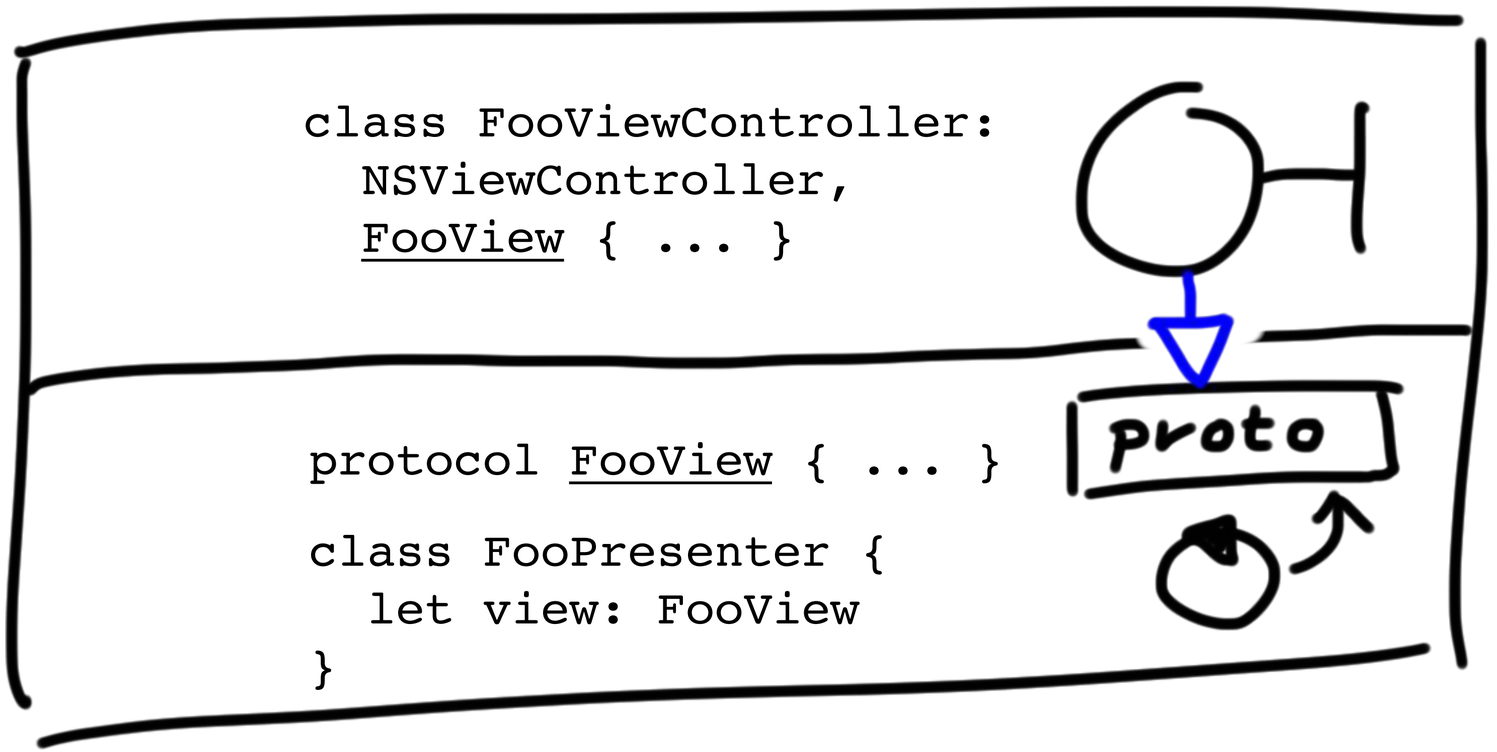




ISP auf Schichtebene



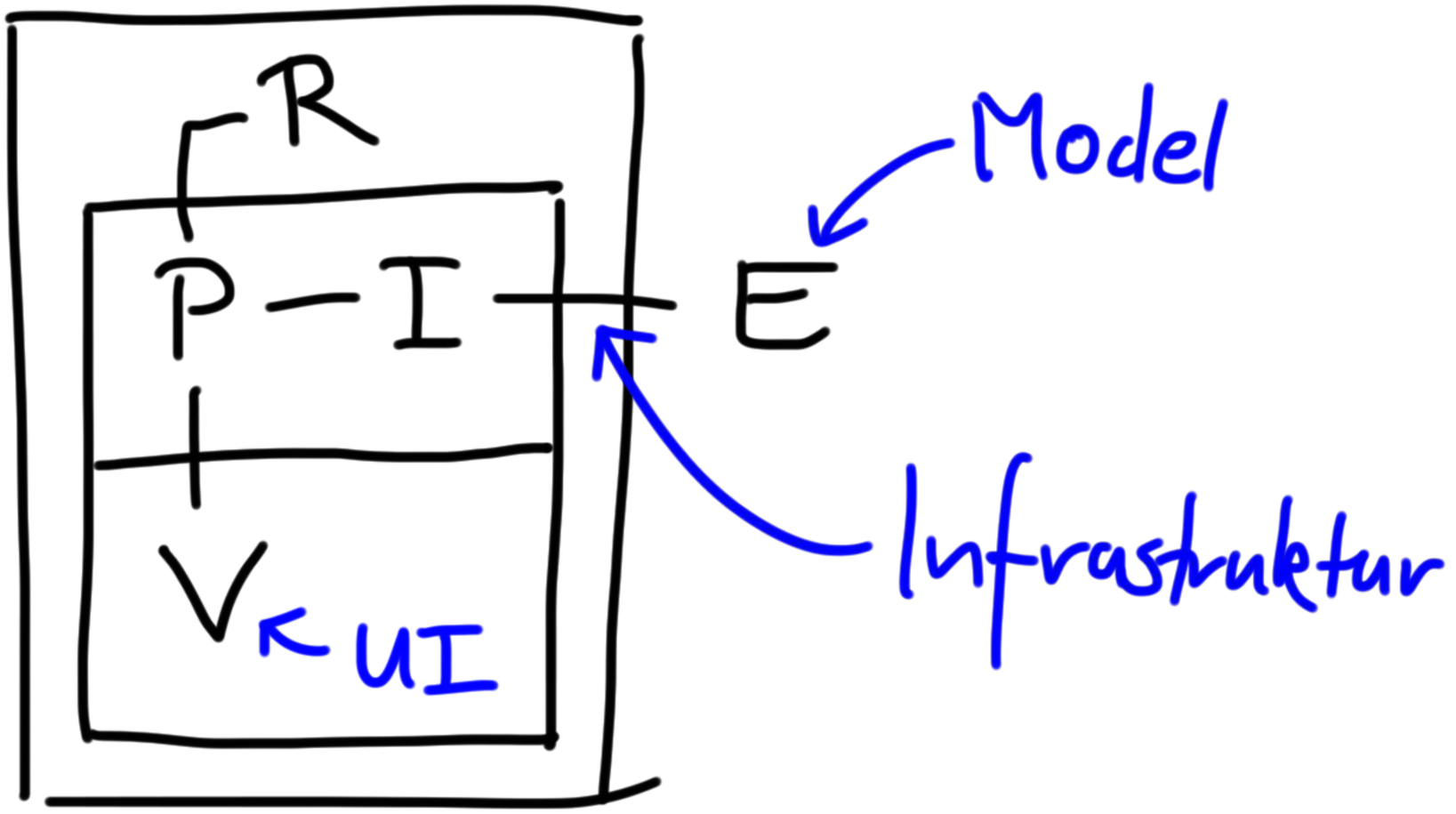
DIP auf Schichtebene



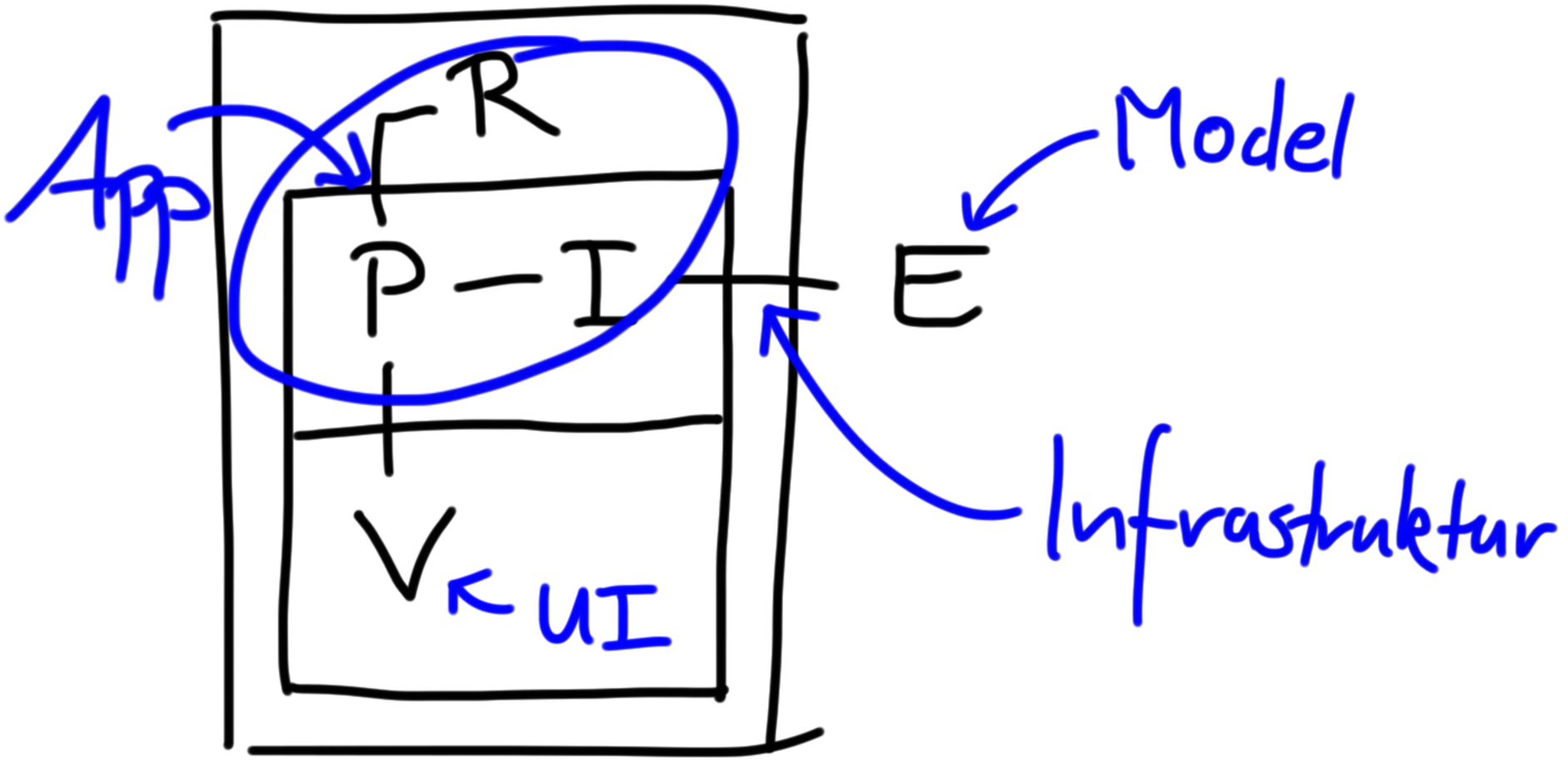
Cargo-Kulte

unter der Lupe

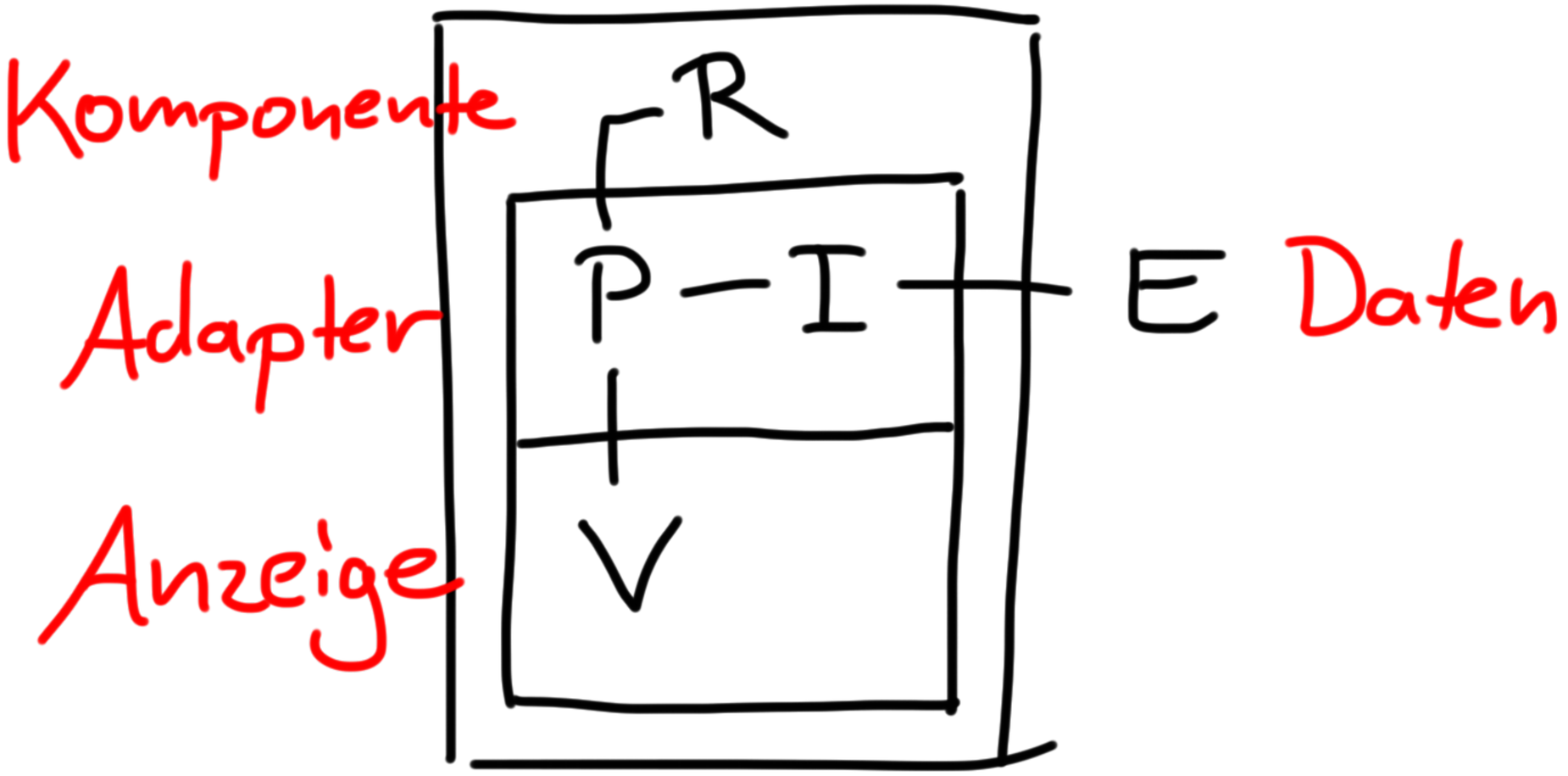
VIPER



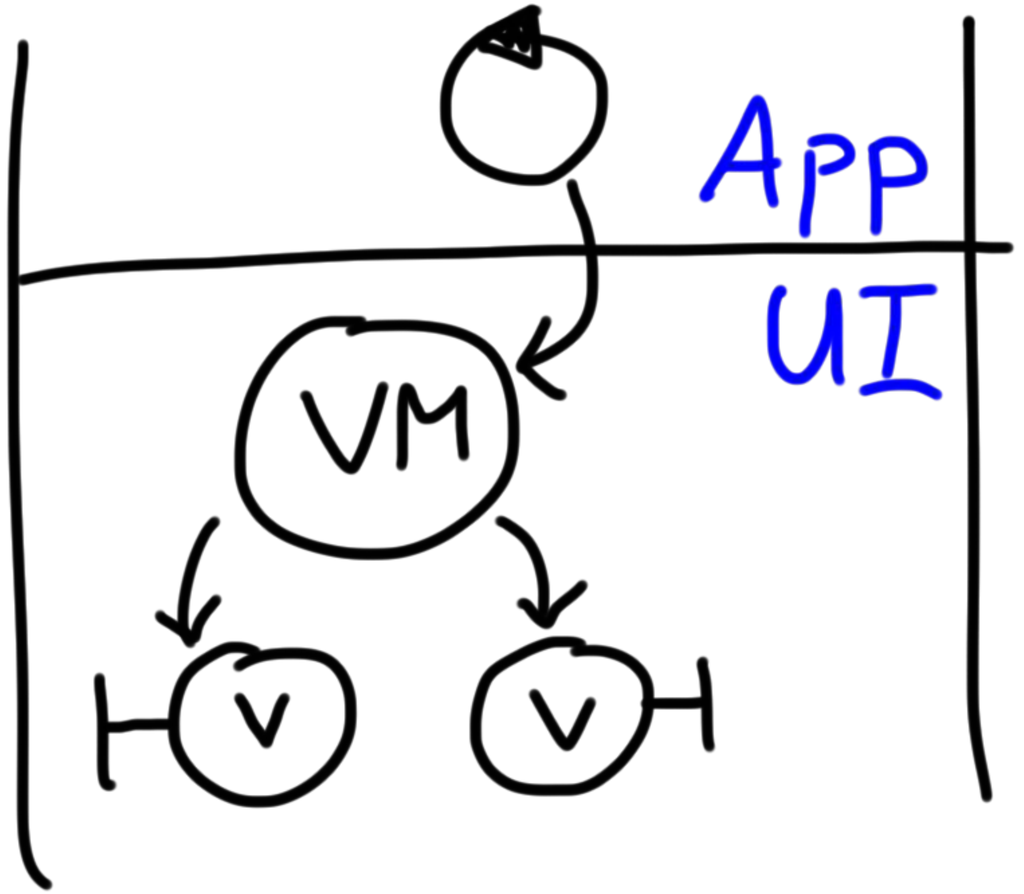
VIPER



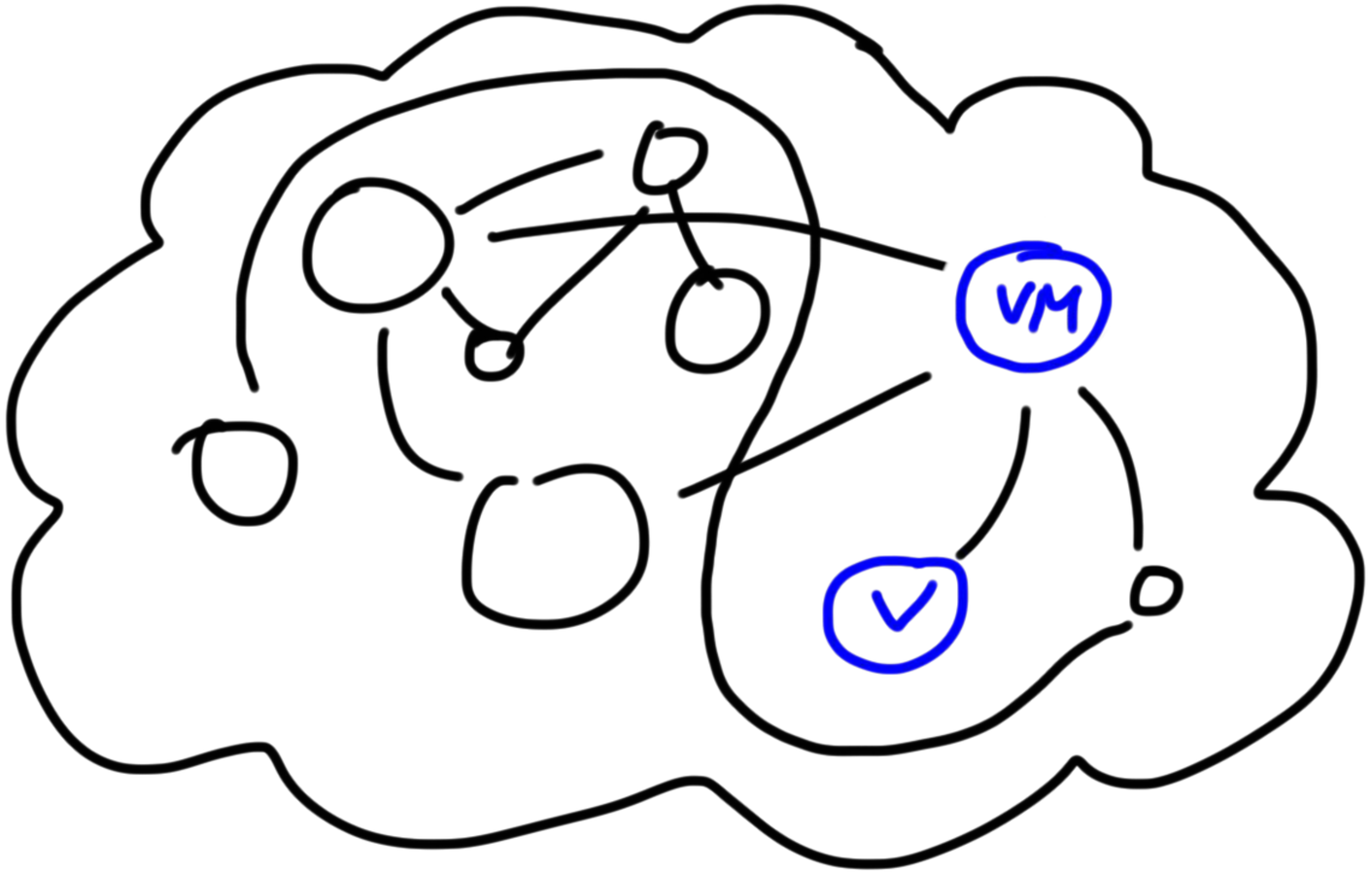
VIPER



MVVM



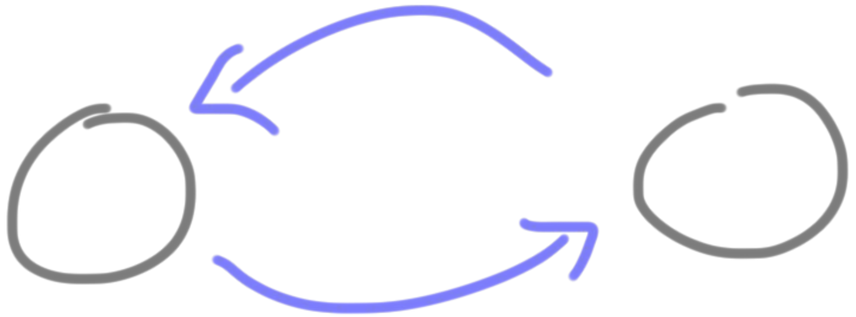
MVVM



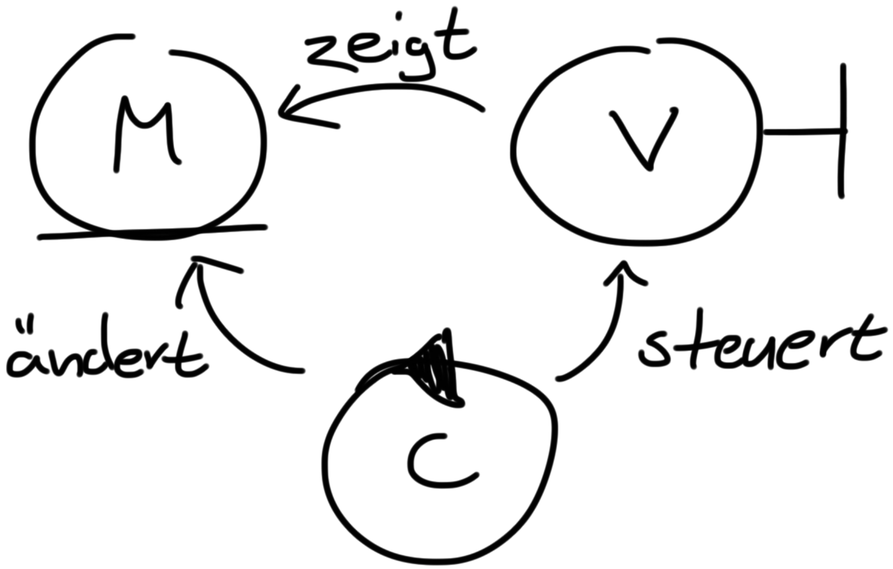
Zusammenfassung »Struktur«

- Modellieren
- Struktur durch Grenzziehung
- Grenzen auf allen Ebenen
- Design Patterns ≠ Architekturansätze

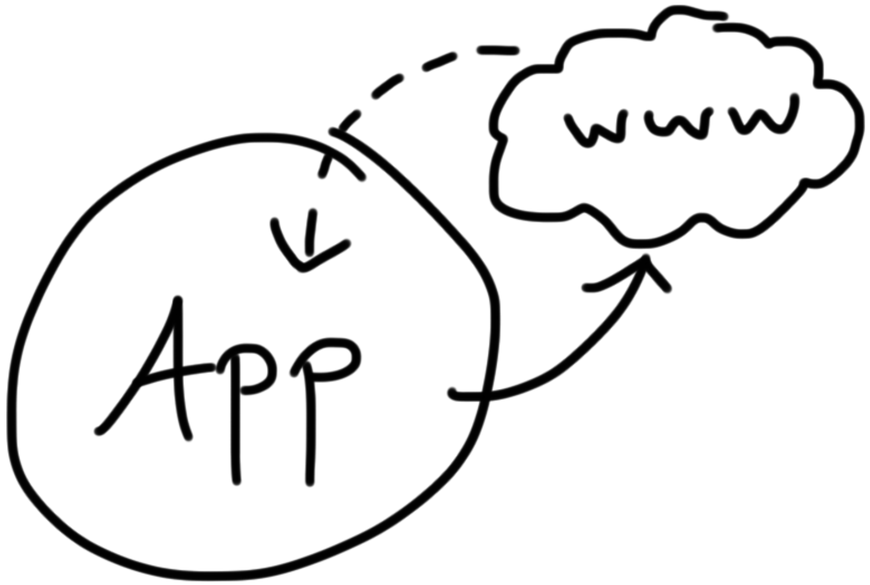
3. Prozess



Wie kriege ich Daten
von einem View Controller
zum anderen übergeben?



- Wie update ich das Model?
- Wie zeige ich Updates dann wieder an?



- Wie update ich die App mit neuen Server-Daten?
- Wie zeige ich Updates dann wieder an?

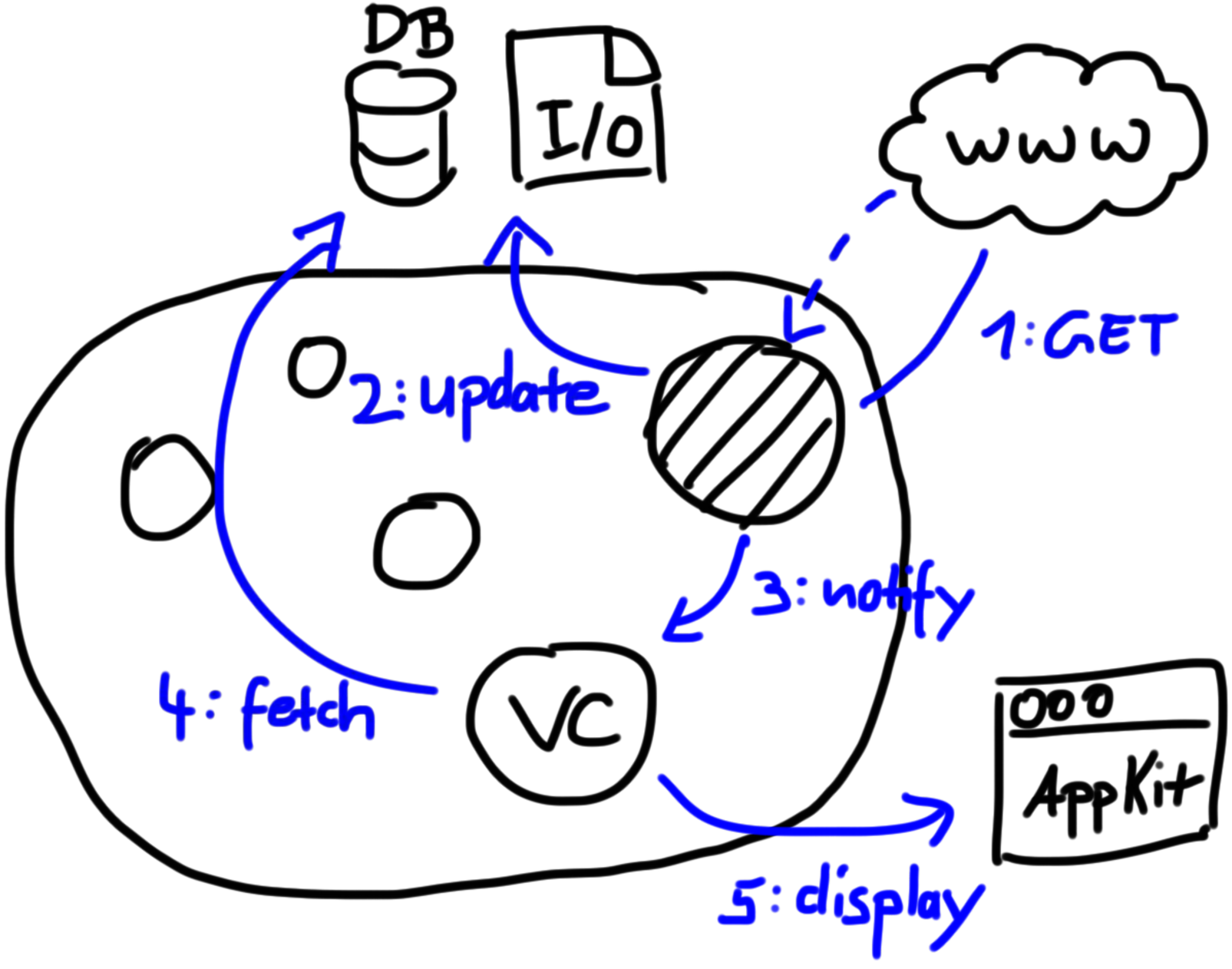
Naiver Ansatz I

¯_(ツ)_/¯

- keine Sync
- keine Updates
- kein Hintergrundservice

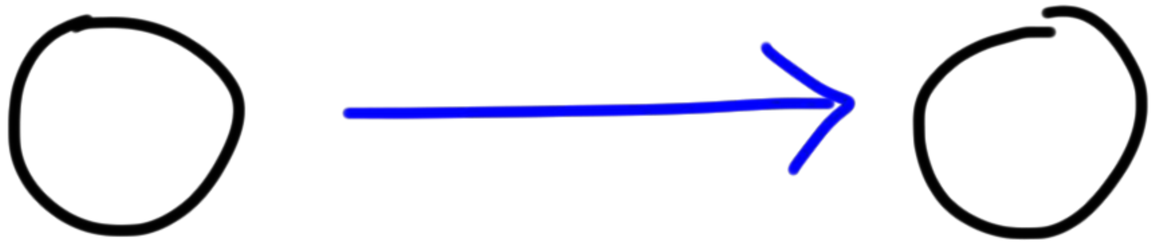
Naiver Ansatz II

- GET Serverdaten
- Update lokaler Daten
- Benachrichtigen der Anzeige
  - Fordere Änderungen an
  - Zeige neue Inhalte an

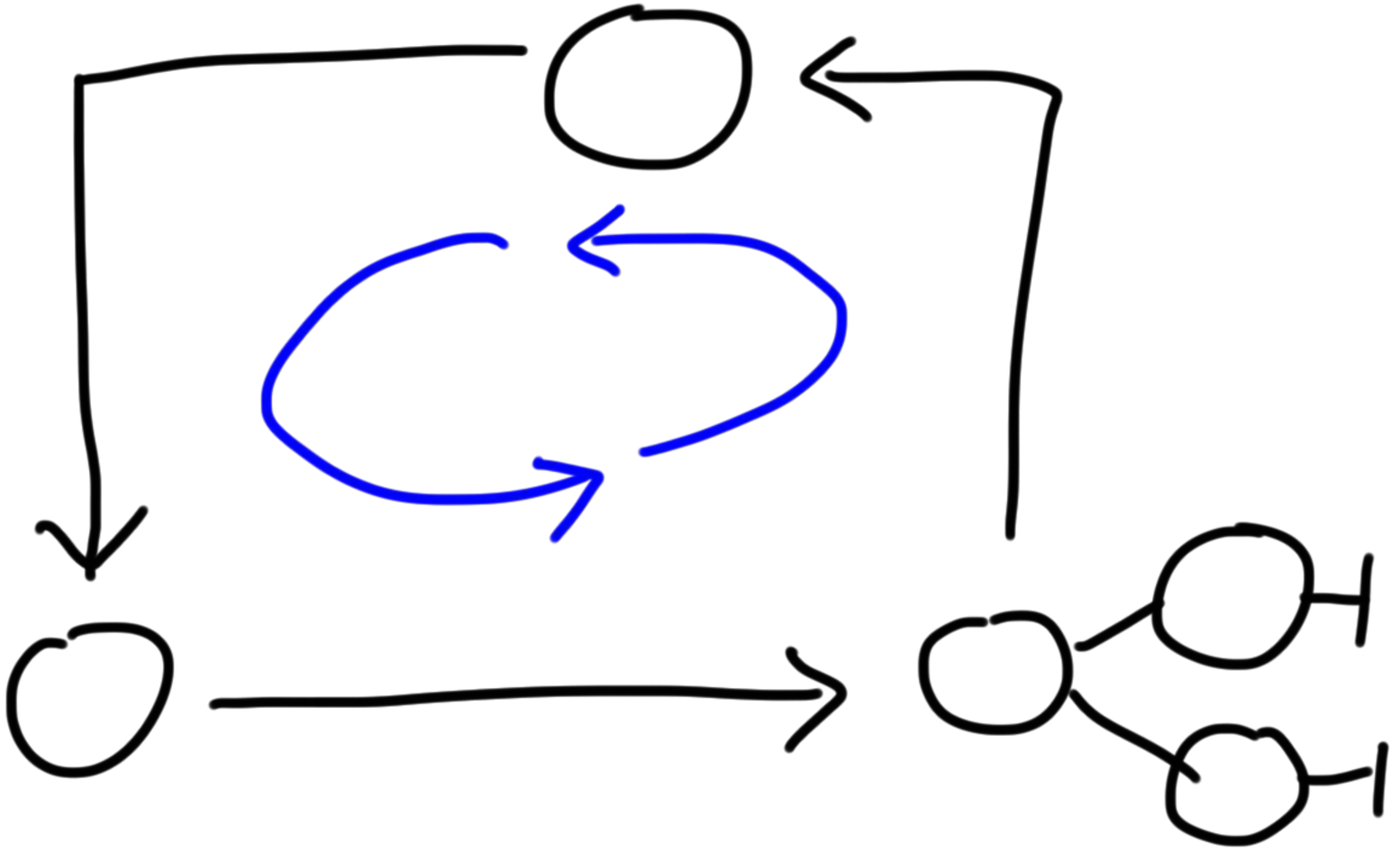

DB
I/O
www
1:GET
2:update
3:notify
4:fetch
VC
AppKit
5:display

Unidirectional Data Flow

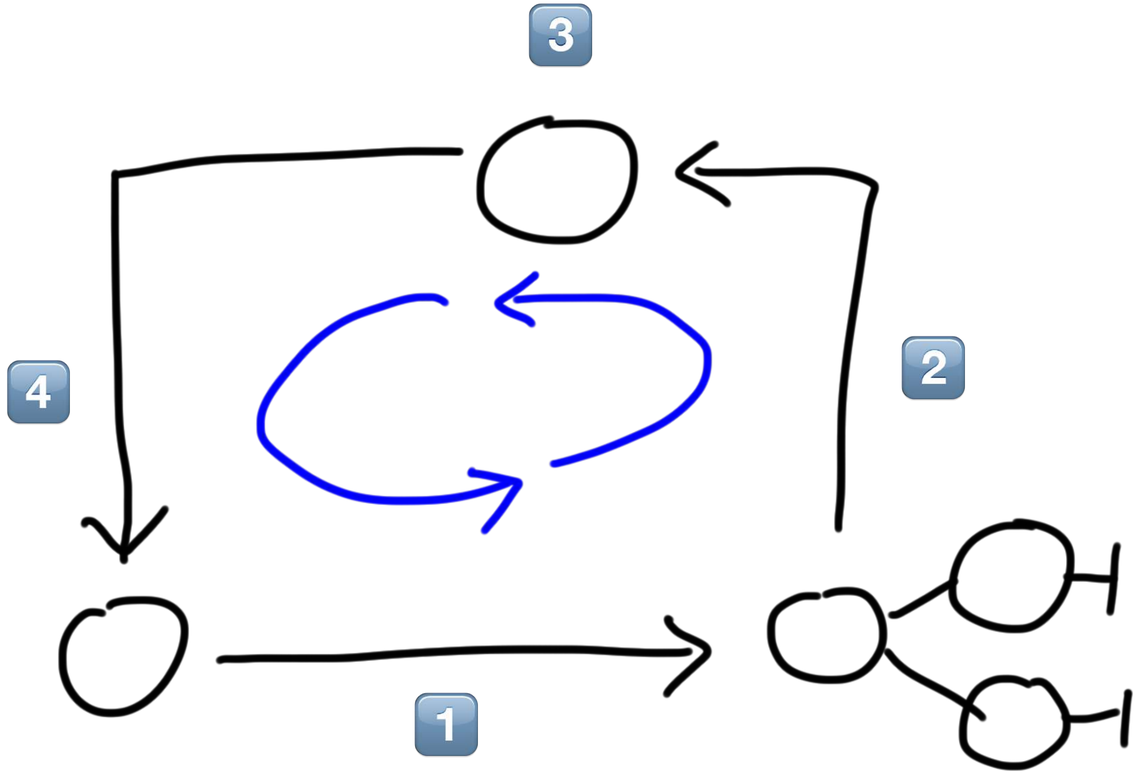
Unidirectional Flow



Unidirectional Flow



Unidirectional Flow



1. Leite Zustand an *subscriber* weiter
2. Verwende den Zustand
3. Schick Änderungen als Events
4. Verarbeite Events zu neuem Zustand
5. gehe zu 1

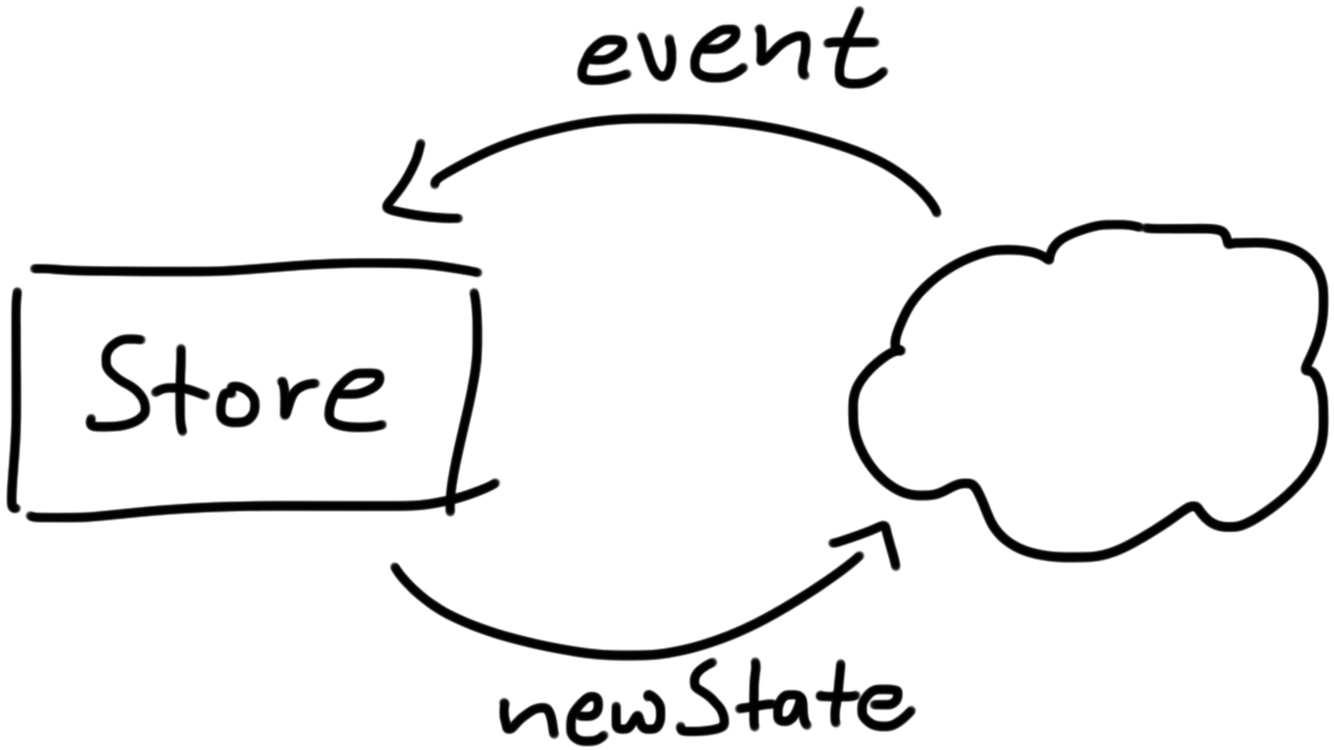
»Wozu? Hab ich doch schon!«

- Events
- Cocoa Bindings
- `Foundation.Notification`
- Methodenaufrufe

ReSwift

ReSwift

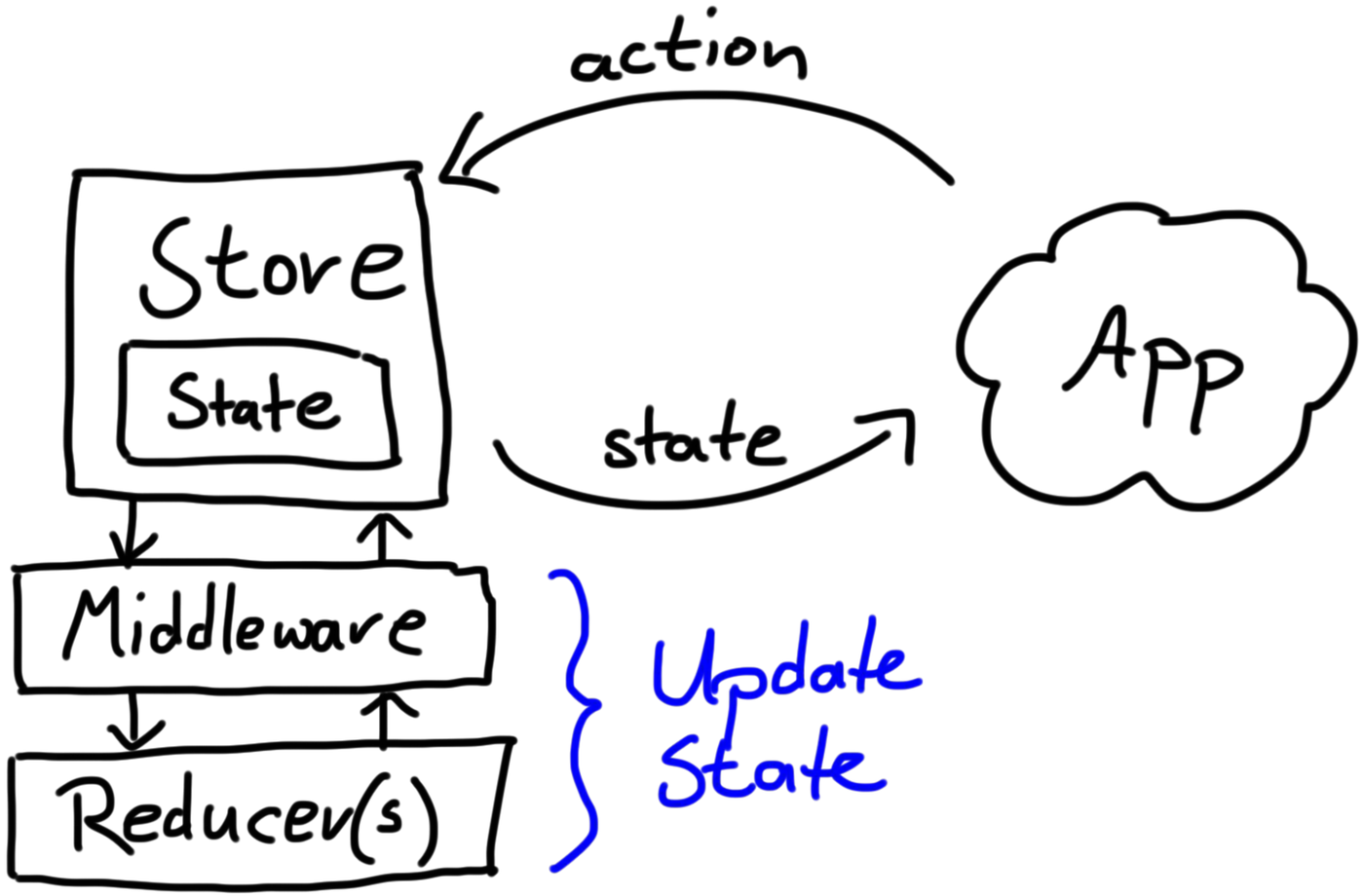
ReSwift.Action



ReSwift.Store<T : ReSwift.StateType>

```
ReSwift.StateSubscriber<T : ReSwift.StateType> {
  func newState(state: T)
}
```

ReSwift


action
Store
State
App
state
Middleware
Reducer(s)
Update
State

Store

```
let store = ToDoListStore(
    reducer: ToDoListReducer(),
    state: nil,
    middleware: [
        loggingMiddleware
    ])
```

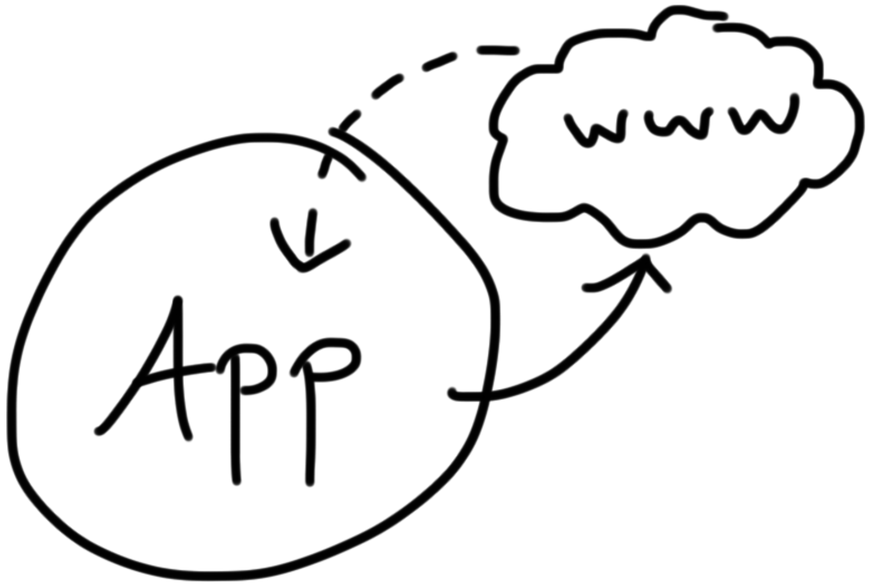
Reducer

```
class ToDoListReducer: ReSwift.Reducer {

  func handleAction(action: Action,
    state: ToDoListState?) -> ToDoListState {

    guard var state = state else { return ToDoListState() }

    state = updateStateSomehow()
    return state
  }
}
```



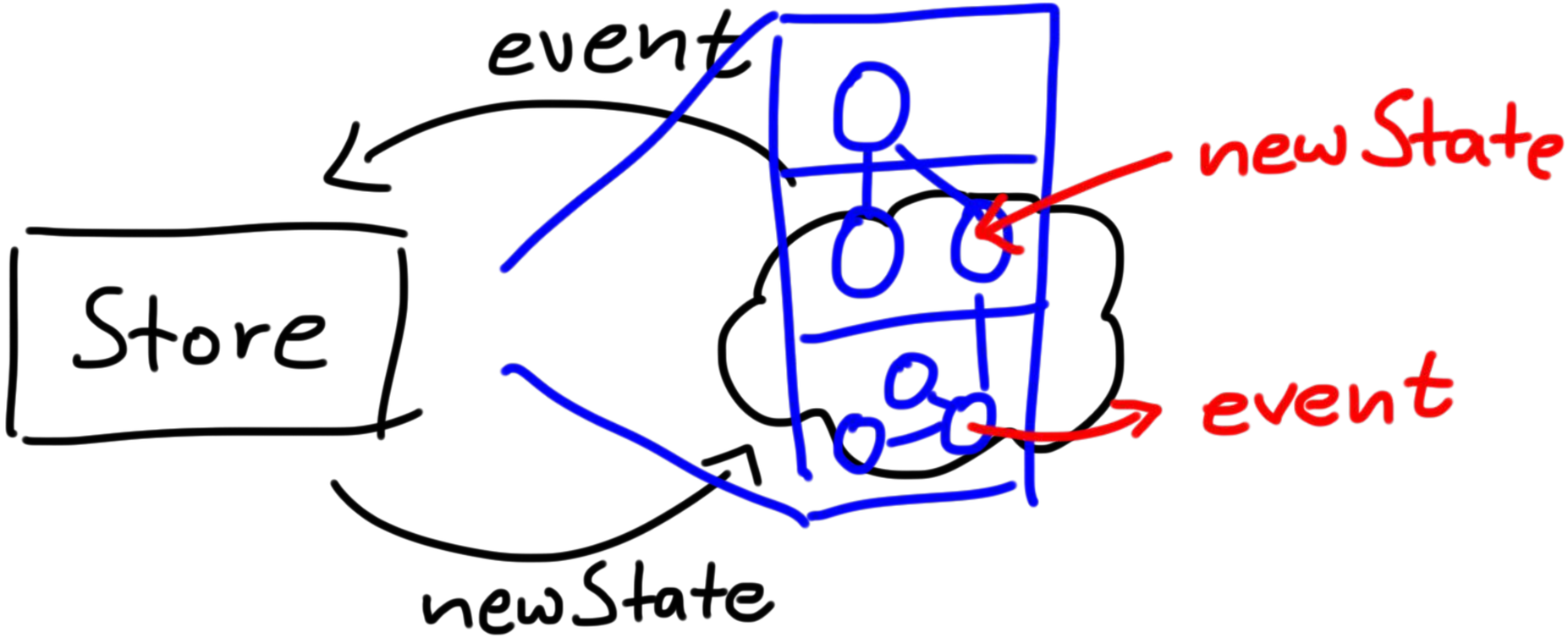
- Wie update ich die App mit neuen Server-Daten?
- Wie zeige ich Updates dann wieder an?

ReSwift-Ansatz

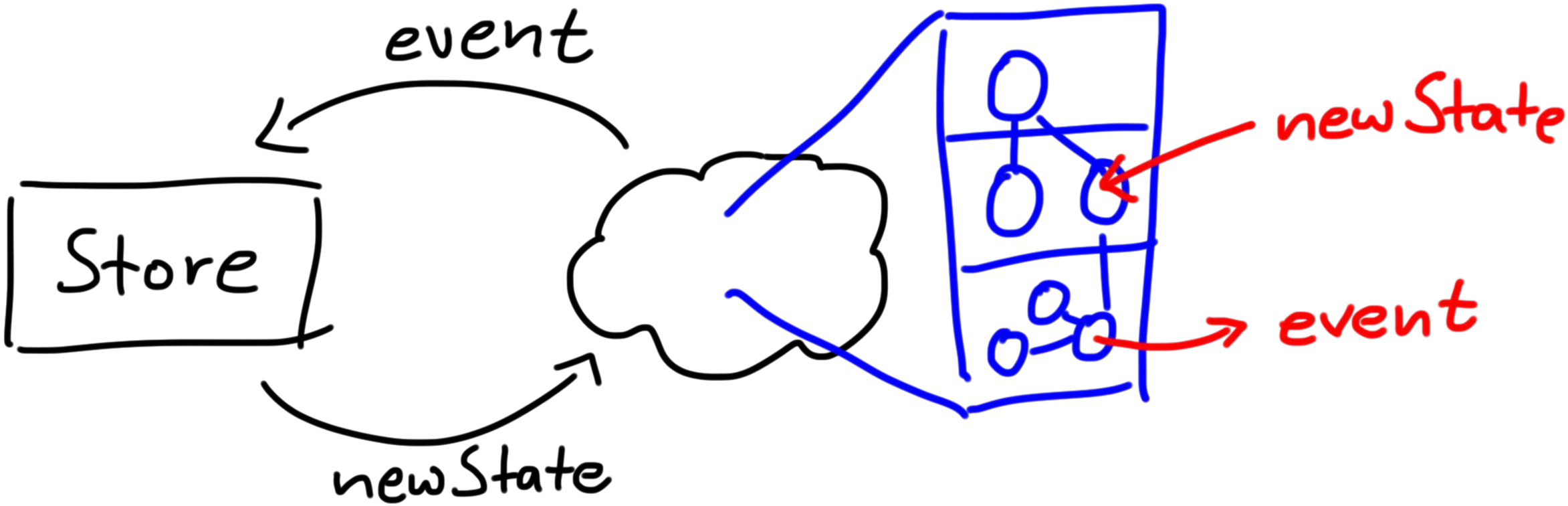
BG Queue
GET Serverdaten
Update Store mit Events
(Rest macht ReSwift)

Main Queue	
`newState` Callbacks	
I/O	UI
Speichere Daten	Zeige Daten an

Architekturvorgaben



Architekturvorgaben



Event Sourcing

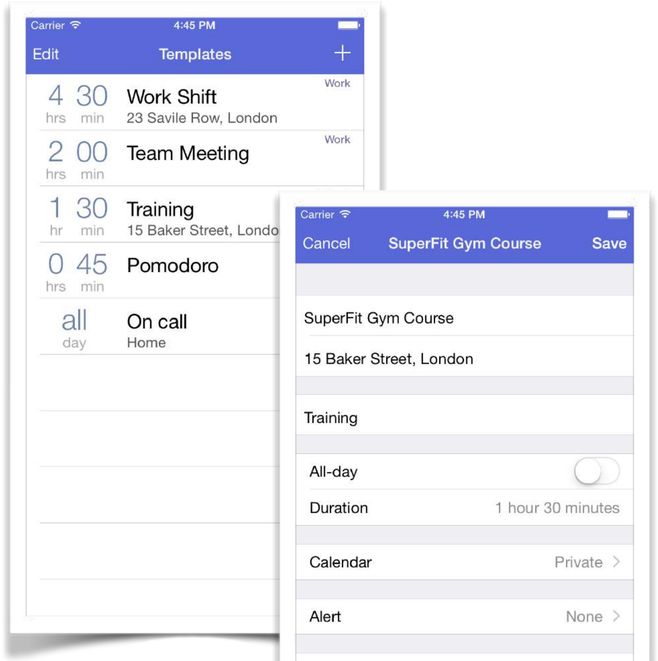
Enterprise Web Services nutzen dies:

- Events statt State in Datenbank speichern
- Geschichte aller Interaktionen
  - Fehleranalyse
  - Zurückrollen von Änderungen

4. Anwendung

… in meinen Apps

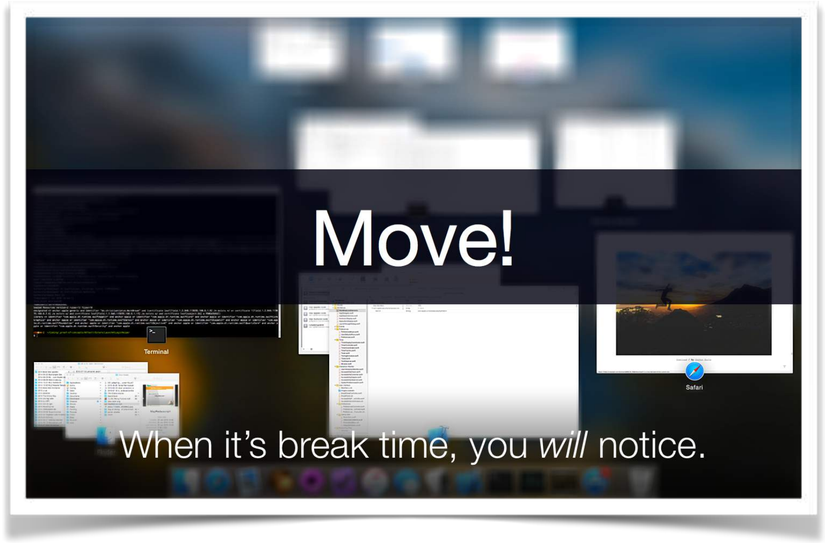
CalendarPaste (iOS)



- State: CoreData & Settings
- `CreateTemplateAction` statt temporären `NSManagedObjectContext`?

➡ lohnt sich nicht: nur Settings relevant

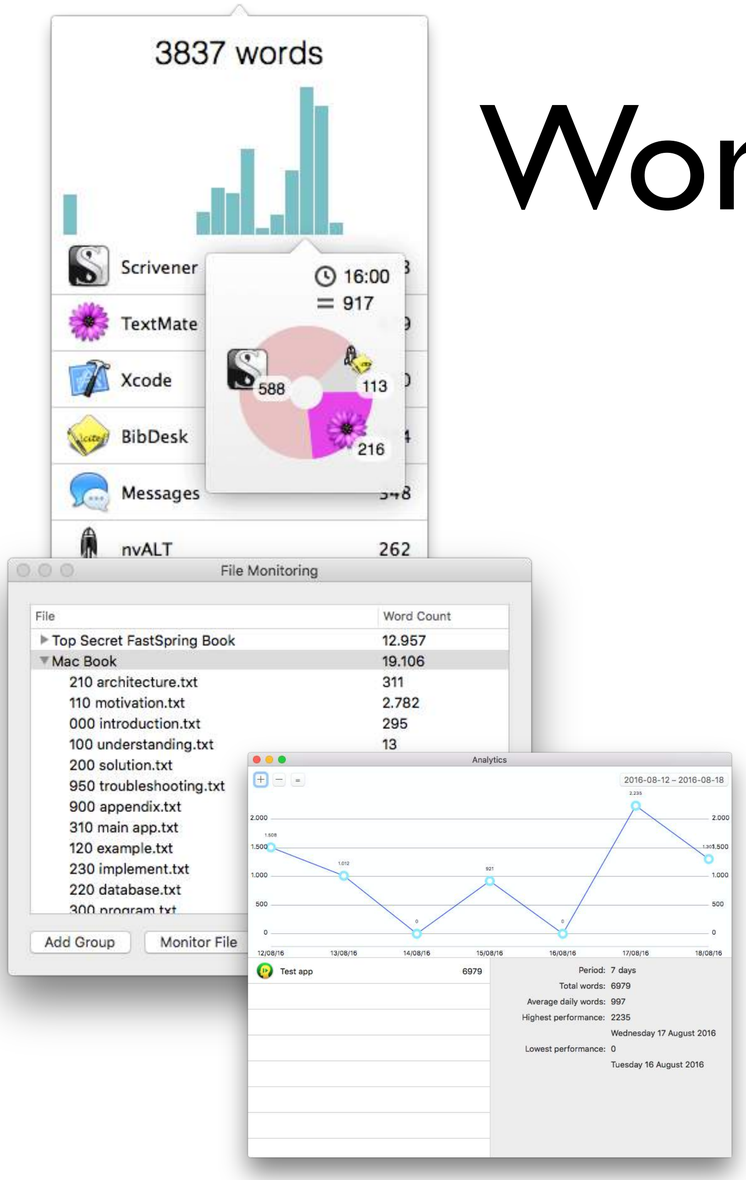
Move! (macOS)



- State: Settings

➡ Timeränderungen vereinfachen

➡ ViewController stärker entkoppeln

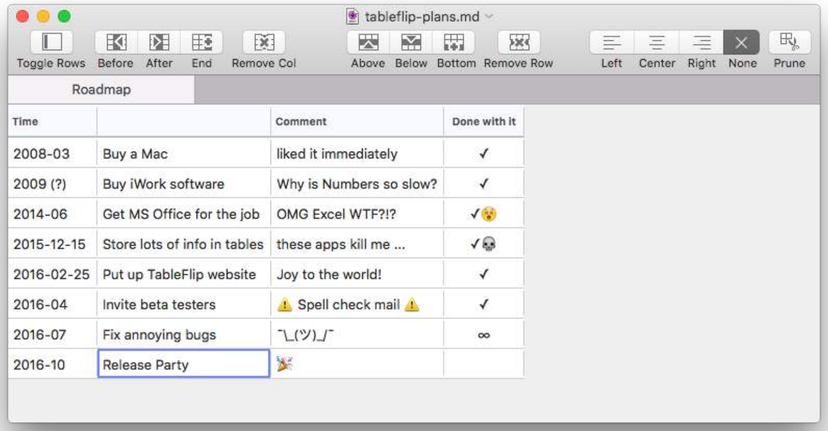


Word Counter (macOS)

- State: Überwachte Apps, überwachte Dateien & Aufzeichnungen
- Events: Tastaturanschlag & Dateiveränderungen
- Anzeige komplex

➡ lohnt sich bei den meisten Modulen

TableFlip (macOS)



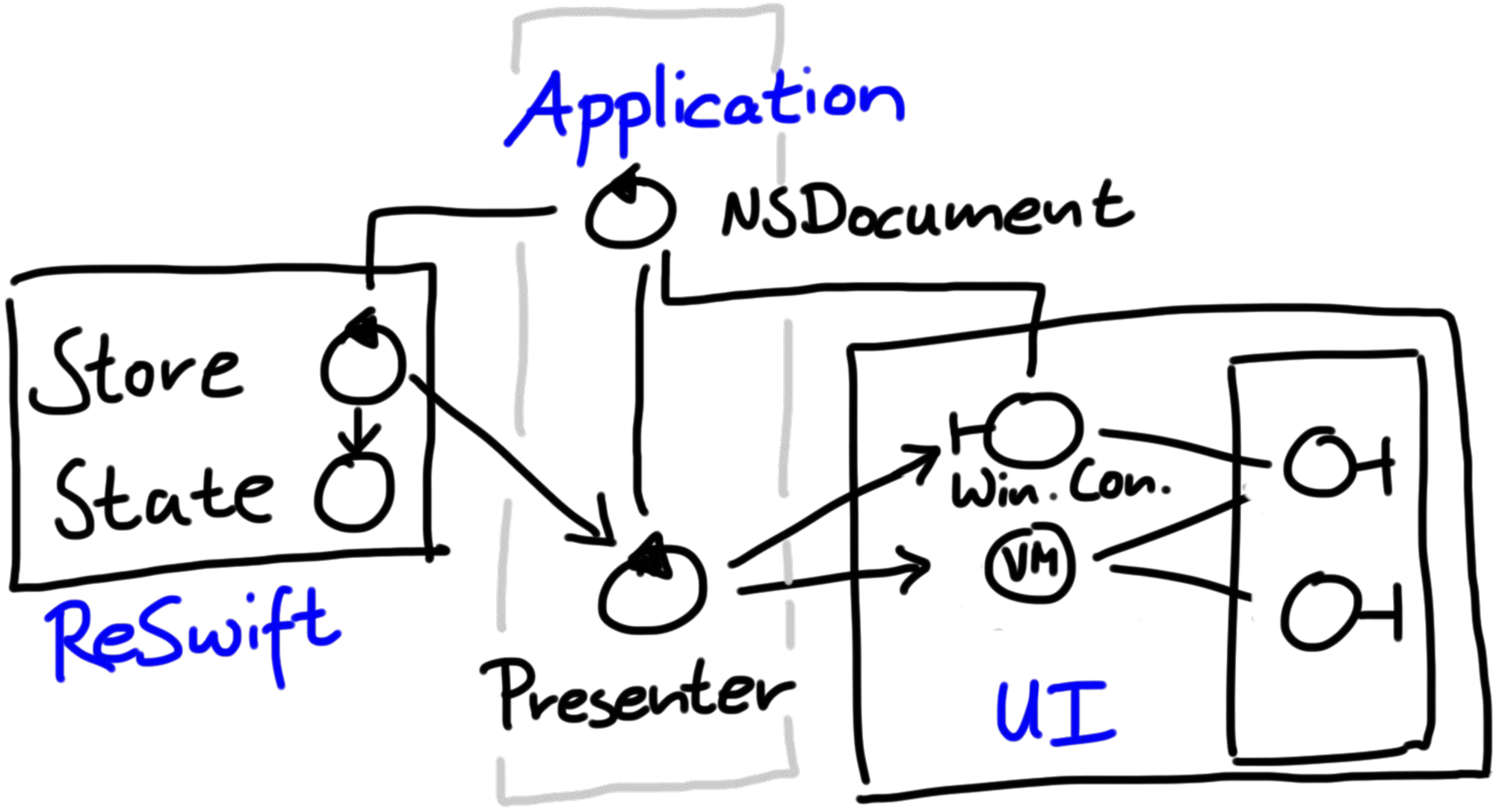
- State: Dateiinhalt
- Events: Benutzereingabe & Dateiupdates
- Anzeige & Interaktion komplex

➡ lohnt sich

⚠️ Update triggert reload, optimierbar

Anwendung in einer exklusiven Macoun Beispielapp

Demo

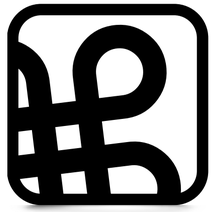

Application
NSDocument
Store
State
ReSwift
Presenter
Win. Con.
VM
UI

Fragen & Anmerkungen

Merci Beaucoup!

Christian Tietze
@ctietze

»Clean Cocoa« Projekt & Blog & Bücher:
http://christiantietze.de/



Macoun