

# 42. Architektur interaktiver Systeme

1

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 15-0.4, 10.11.15

- 1) Benutzungsoberflächen und Anwendungslogik
- 2) Kopplung von synchronen und formularbasierten Benutzungsoberflächen und Anwendungslogik
- 3) Kopplung von reaktiven, graphischen Benutzungsoberflächen und Anwendungslogik
- 4) Controller als Steuerungsmaschinen
- 5) Implementierung der Konnektoren
- 6) Swing



▶ Obligatorisch:

- [PassiveView] Martin Fowler. Passive View. <http://www.martinfowler.com/eaDev/PassiveScreen.html>. Strikte Schichtung, aktiver Controller und passiver View.

▶ Weitere:

- F. Buschmann, N. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-orientierte Software-Architektur. Addison-Wesley.
  - ◆ Entwurfsmuster und Architekturstile. MVC, Pipes, u.v.m.
- [Herrmann] M. Veit, S. Herrmann. Model-View-Controller and Object Teams: A Perfect Match of Paradigms. Aspect-Oriented System Development (AOSD) 2003, ACM Press
- Mike Potel. MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. VP & CTO Taligent, Inc.
  - ◆ <ftp://www6.software.ibm.com/software/developer/library/mvp.pdf>
- html web frameworks
  - ◆ STRUTS <http://exadel.com/tutorial/struts/5.2/guess/strutsintro.html>
  - ◆ Web Application Component Toolkit [http://www.phpwact.org/pattern/model\\_view\\_controller](http://www.phpwact.org/pattern/model_view_controller)

- ▶ Die Architektur interaktiver Anwendungen ist eines der komplexesten Gebiete der Software-Architektur
- ▶ Um sie zu verstehen, brauchen wir *alle Teile* des Kurses:
  - Kollaborationen und Konnektoren
  - Schichten
  - Steuerungs- und Protokollmaschinen
  - Sequenzdiagramme
  - Entwurfsmuster

Die Bildung kommt nicht vom Lesen, sondern vom  
Nachdenken über das Gelesene.

Carl Hilty, 28.02.1833 - 12.10.1909

Schweizer Richter und Staatsrechtler, Buchautor und christl. Staatsrechts-  
Philosoph

Seine Bücher beeinflussten auch K. Adenauer



# 42.1 Benutzungsoberflächen (UI) und Anwendungslogik

---

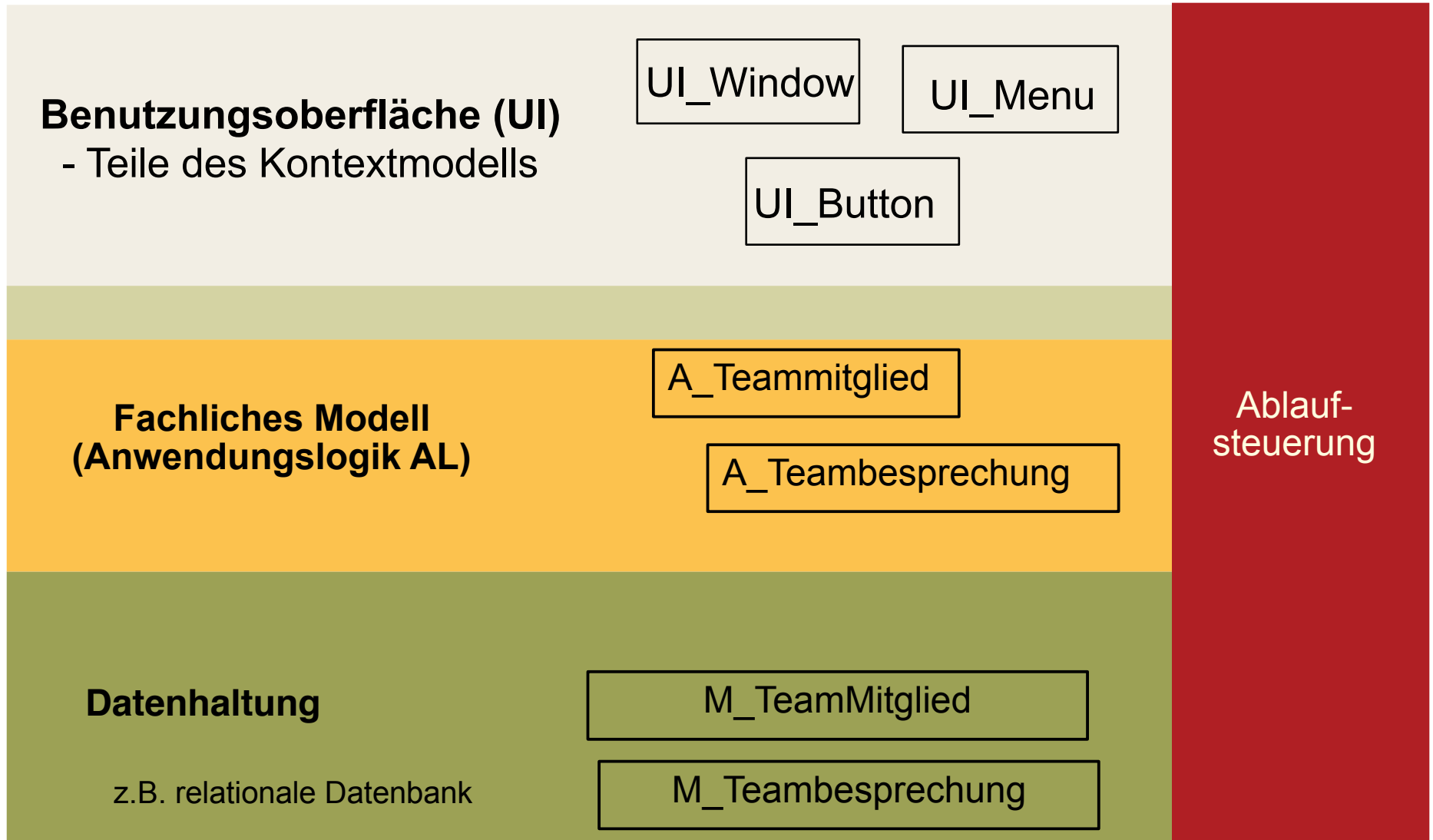
---

4

Verschiedene Arten der Kopplung zwischen Benutzer und Software

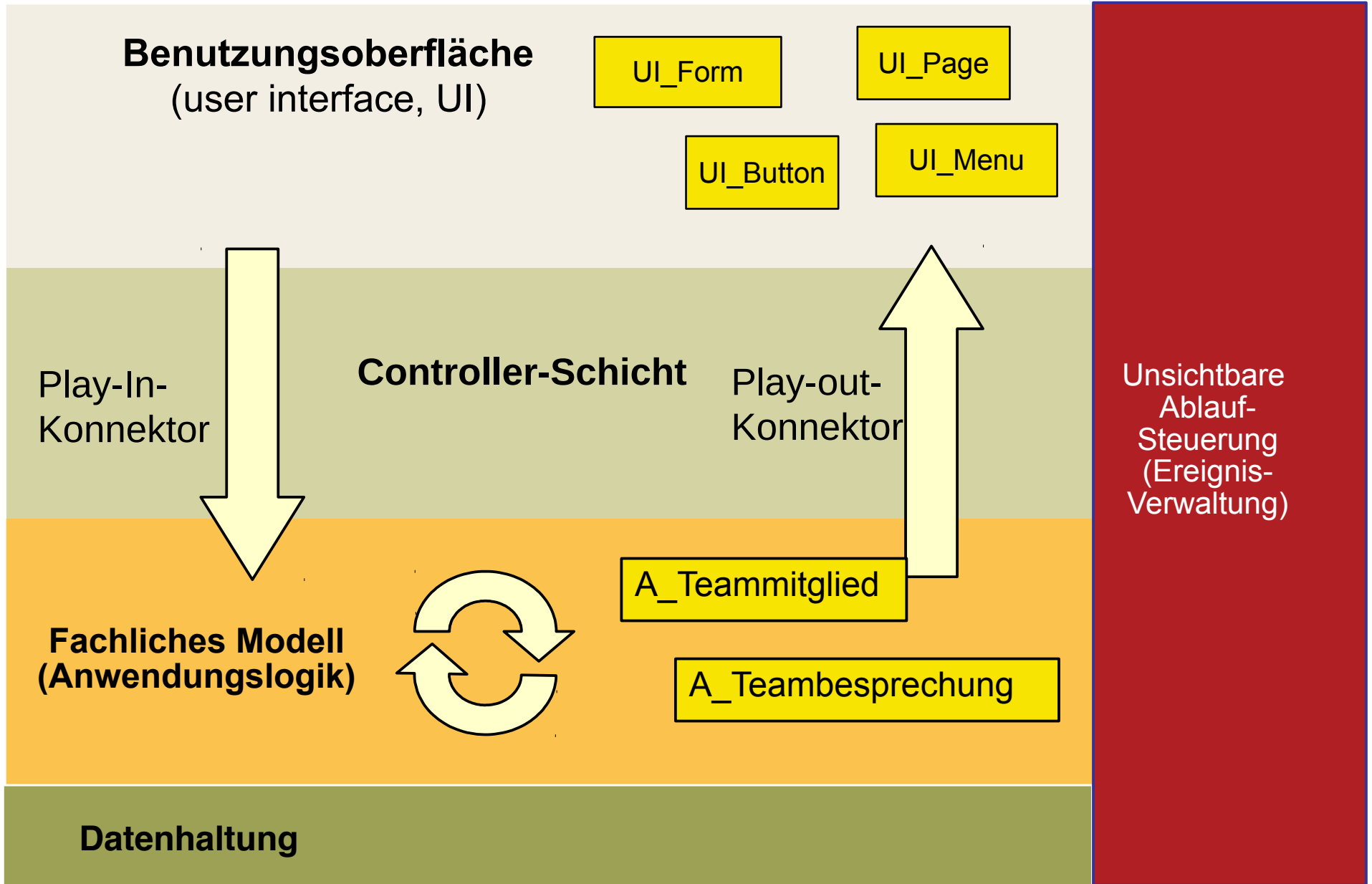
# Schichtenarchitektur, grob

5



# Controller bildet 4. Schicht zwischen der Benutzungsoberfläche (UI) und der Anwendungslogik

6



# Arten von Benutzungsschnittstellen (User Interface, UI)

7

- ▶ **Synchrone UI:** die Anwendungslogik ruft die UI auf und wartet auf Eingaben (Treiber ist die Anwendungslogik)
  - Kommandozeilen-orientiert, textuelle UI (TUI)
  - Maskenorientiert (screen flow) oder formularorientiert (form flow, FUI)  
==> dann kann der Controller entfallen
  - Verteilte UI (Web UI)  
==> dann muss der Controller die verteilte und parallele Verarbeitung steuern

- ▶ **Asynchrone UI:** die Anwendungslogik reagiert auf die UI (Treiber ist die UI)
  - Graphische UI (GUI)
  - Tangible UI (TUI)  
==> dann muss der Controller die parallele Verarbeitung steuern

## 42.2 Kopplung von *synchronen* Benutzeroberflächen mit der Anwendungslogik

8

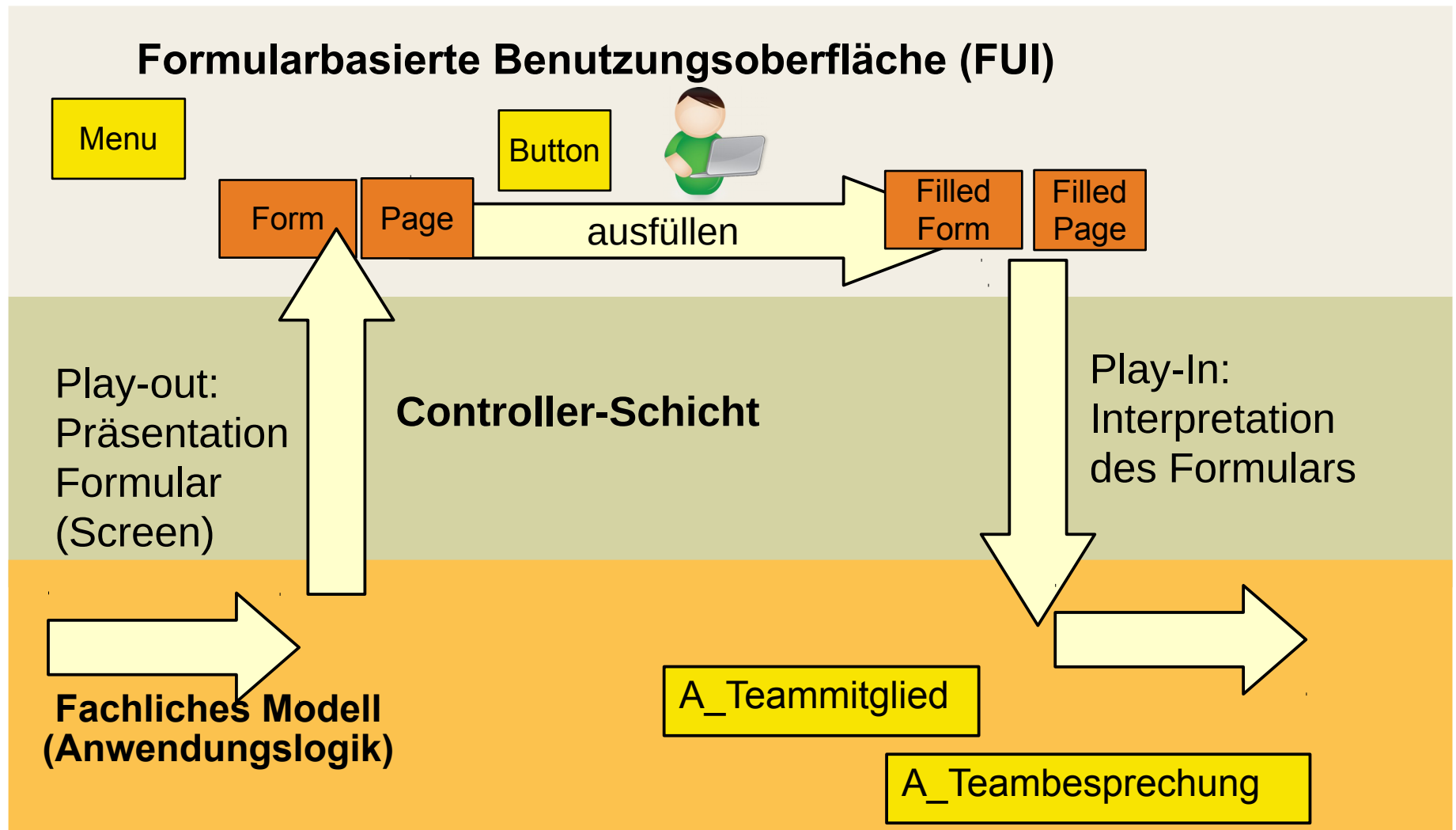
- Text- und Formularbasierte Oberflächen (Form-Based UI, FUI) sind meist synchron mit der Anwendungslogik gekoppelt
- Die Anwendungslogik ruft die Oberfläche auf und wartet auf die Eingaben des Benutzers, z.B. das Ausfüllen von Formularen



# Synchrone Kopplung zwischen Anwendungslogik, Controllerschicht und FUI

9

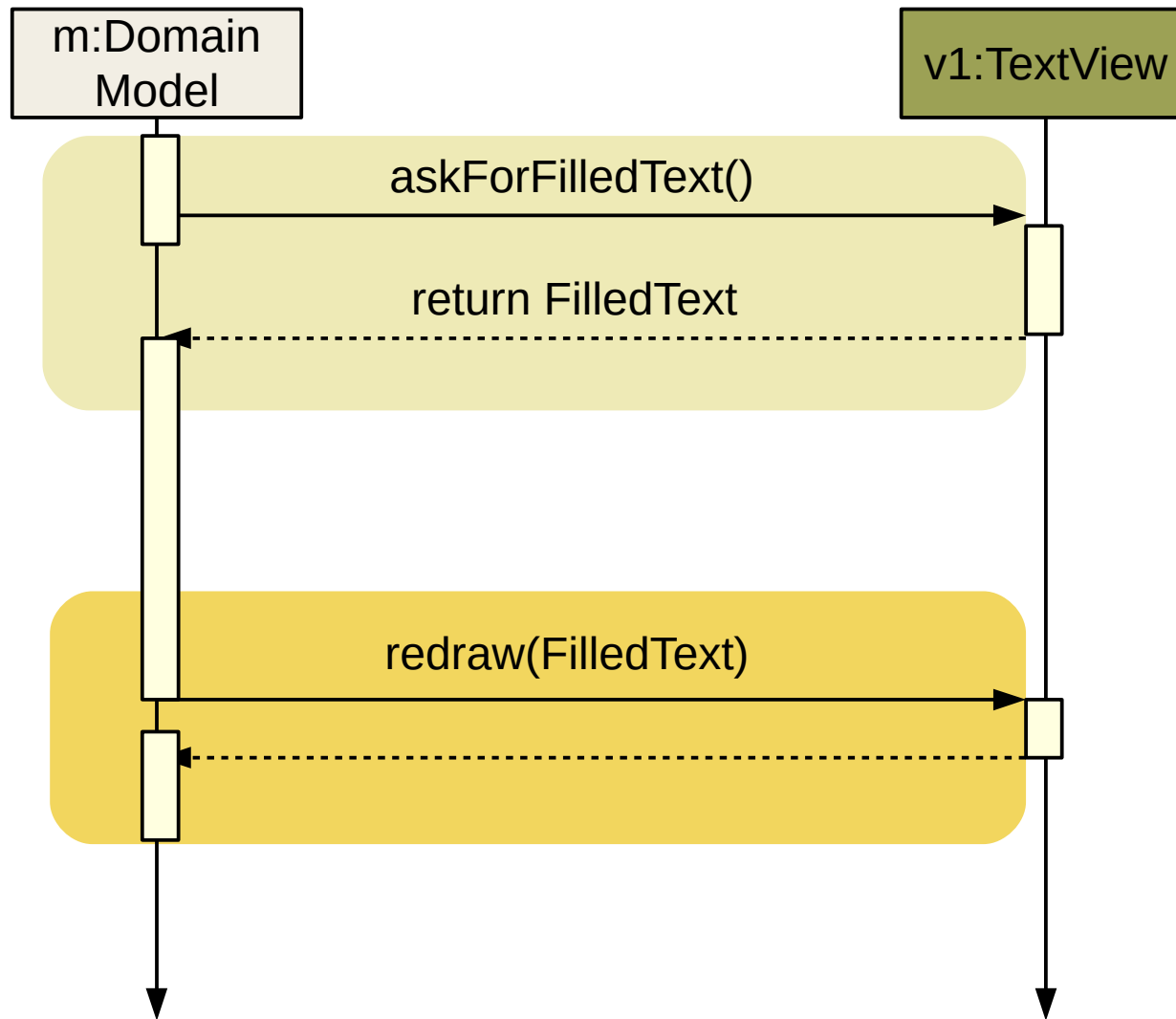
- ▶ Die Anwendungslogik ruft das formularbasierte UI mit einem leeren Formular auf und wartet auf das Ausfüllen des Benutzers (synchron)
- ▶ Die Play-In und Play-Out-Konnektoren sind besonders einfach



# 42.2.1. Textbasierte UI mit synchronem Update (*ein View*)

10

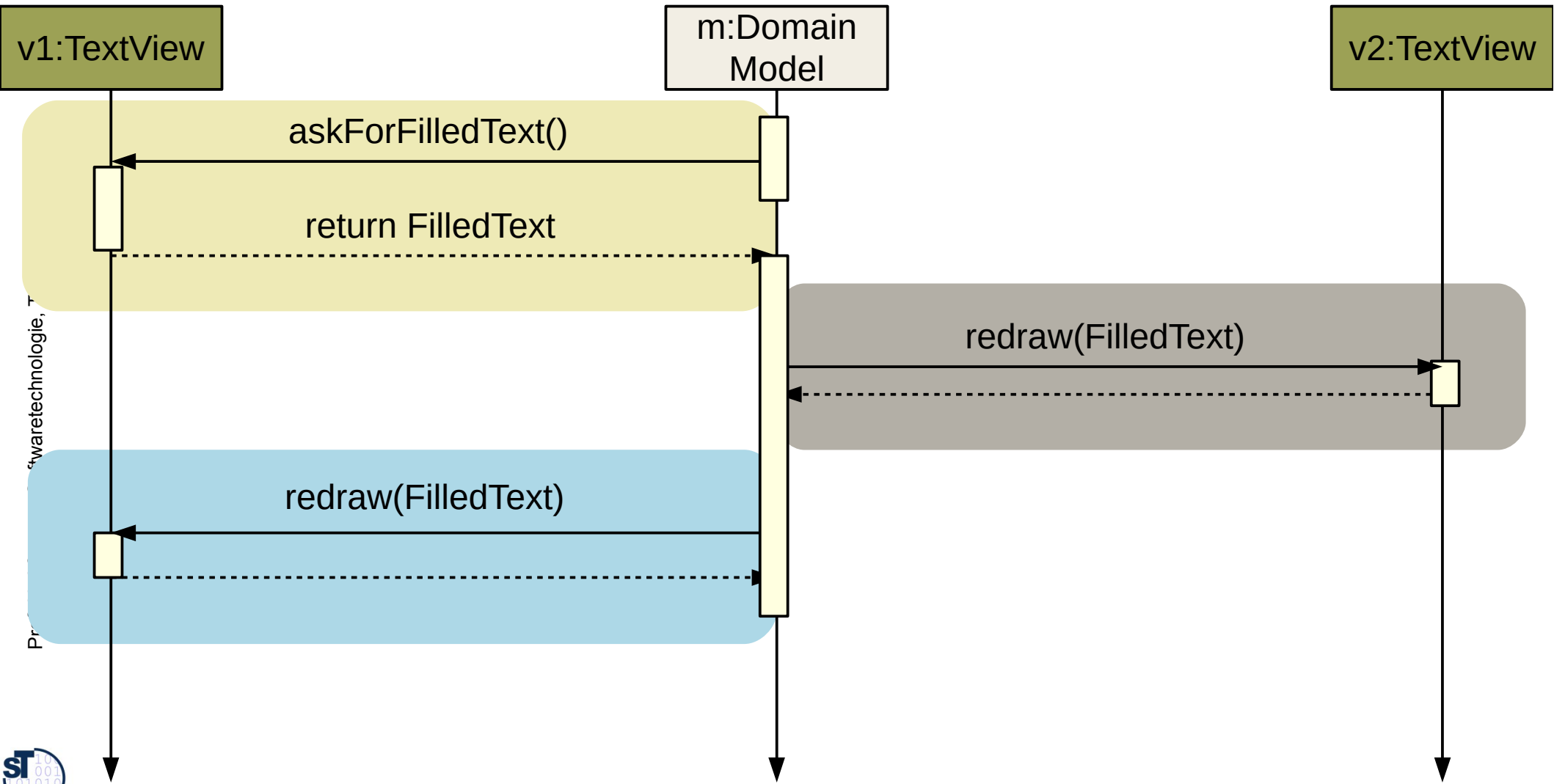
- ▶ In Java: Eingabe mit `System.in`, Ausgabe mit `System.out`
- ▶ Play-In und Play-Out-Konnektoren sind Prozeduraufrufe an die UI



# Textbasierte UI mit synchronem Update (mehrere Text-Views)

11

- ▶ Immer alles schön nacheinander (synchron)



# Einfache textuelle Sichten

12

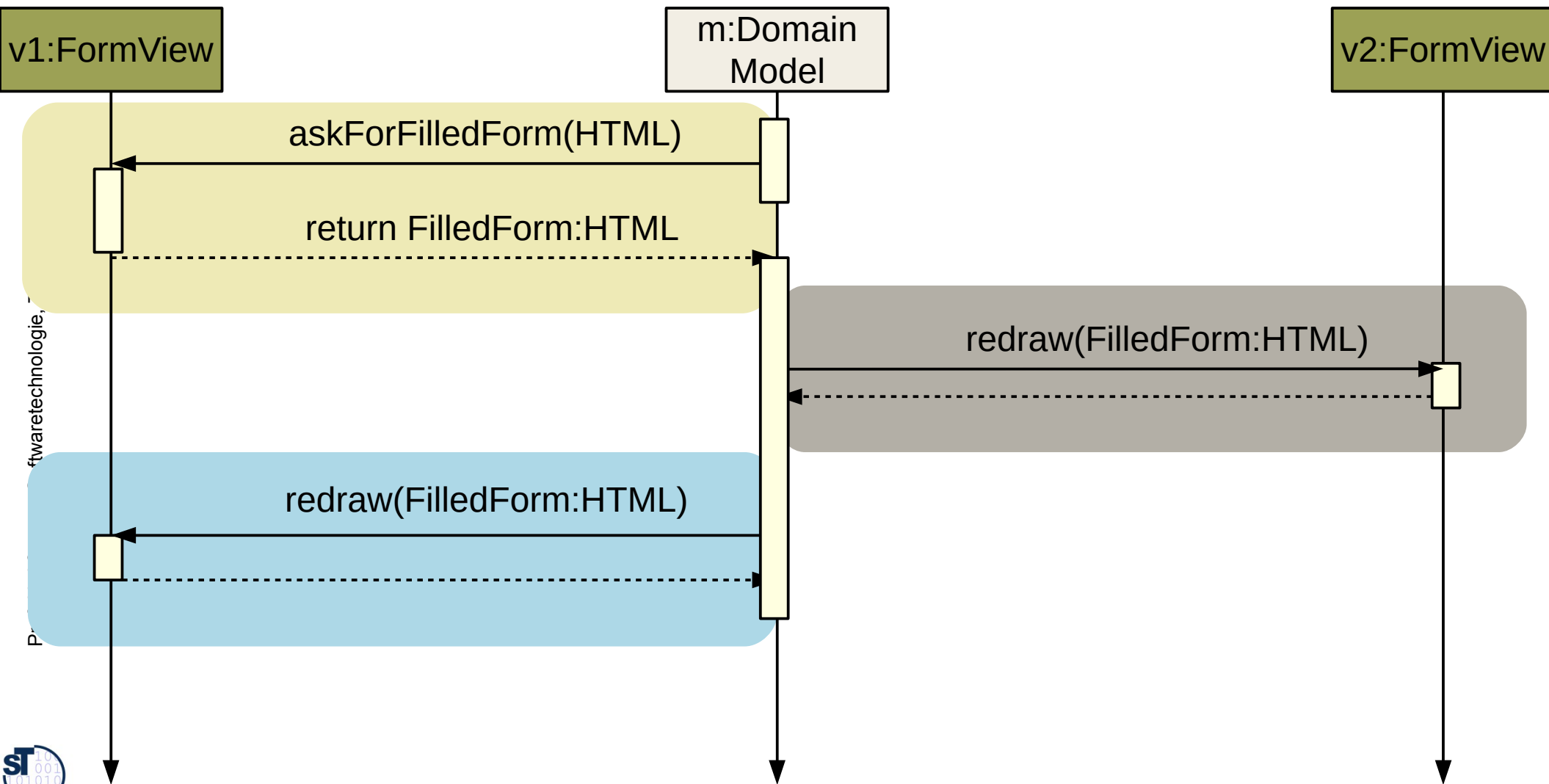
- ▶ Textbasierte UI sind spezielle formularbasierte UI
- ▶ In Java: Aufruf der Objekte `System.in` und `System.out`

```
class PersonModel {  
  
    ... activities of the model ...  
  
    System.out.println(„Enter a number\n“);  
    int num = System.in.read();  
  
    Person p.number = num;  
  
    foreach (view ; model.getViews()) {  
        view.redraw(p);  
    }  
  
    ... further activities of the model ...  
}
```

## 42.2.1. Formularbasierte UI mit XML

13

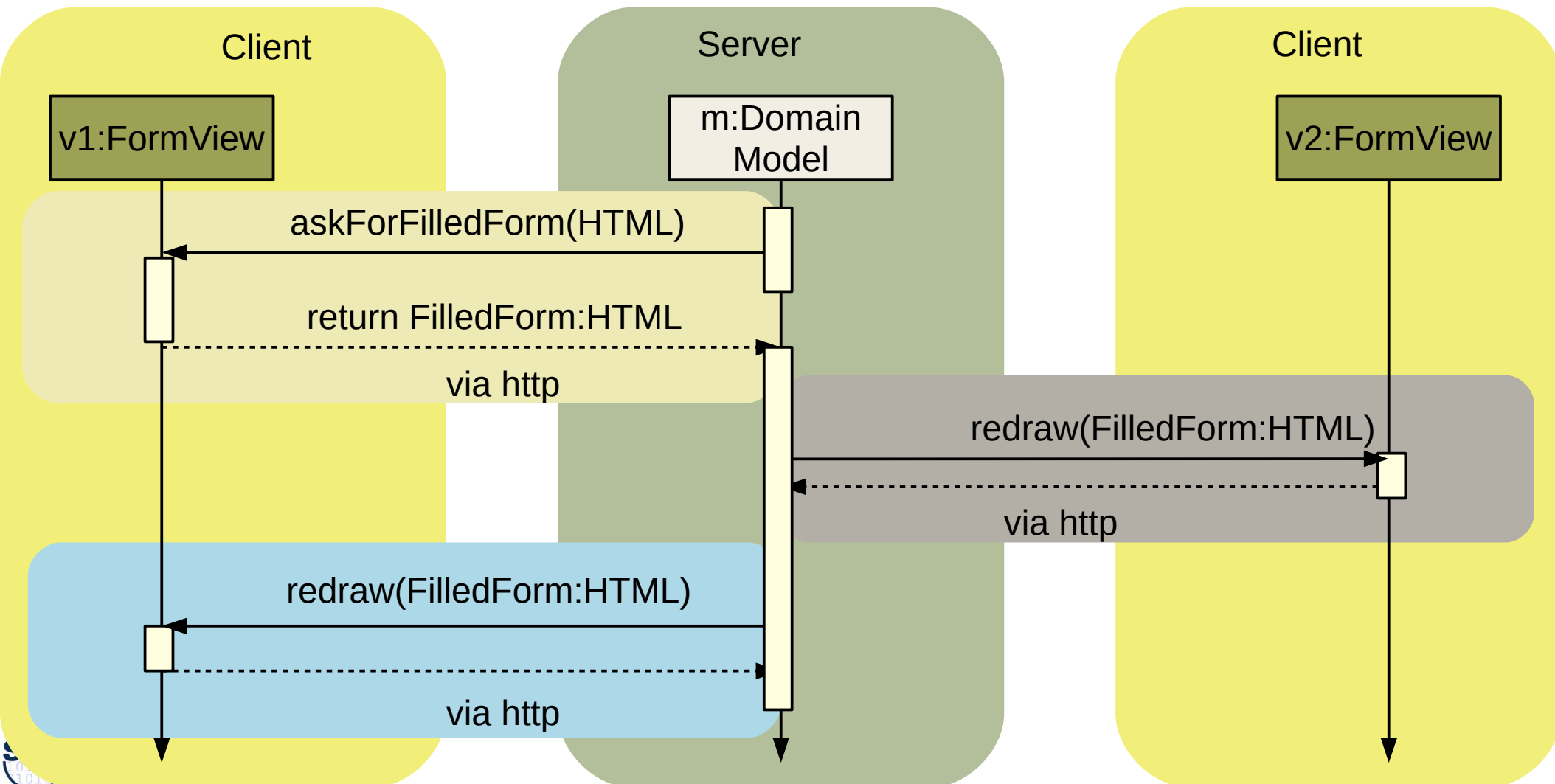
- ▶ HTML und XML bieten standardisierte Formate für Formulare an, die von Browsern dargestellt, interpretiert, und ausgefüllt werden können
- ▶ Die Play-In und Play-Out-Konnektoren transportieren XML-Dokumente



# Formularbasierte UI mit XML übers Web

14

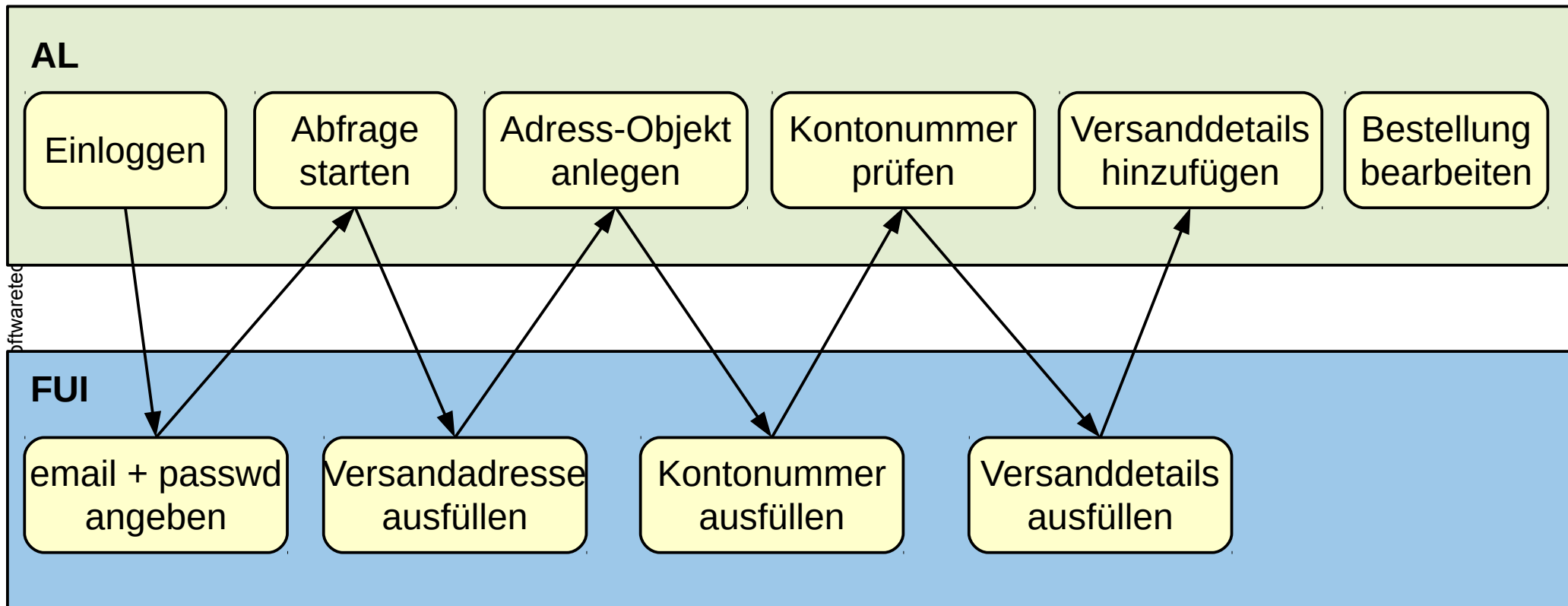
- ▶ HTML und XML können vom Client zum Server übertragen werden
- ▶ Kanalprotokoll http oder https



# Screen-Flow

15

- ▶ Der Fluss von Daten zwischen AL und FUI wird als **Screen Flow** bezeichnet und kann durch ein Aktivitätendiagramm mit zwei Swimlanes beschrieben werden
- ▶ Die Initiative liegt in der AL: Der FUI wird jeweils von der AL beauftragt, die Daten einzuholen



# 42.3 Überblick zu *reaktiven* graphischen Benutzeroberflächen (GUI)

Kopplung der GUI und Anwendungslogik durch Controller

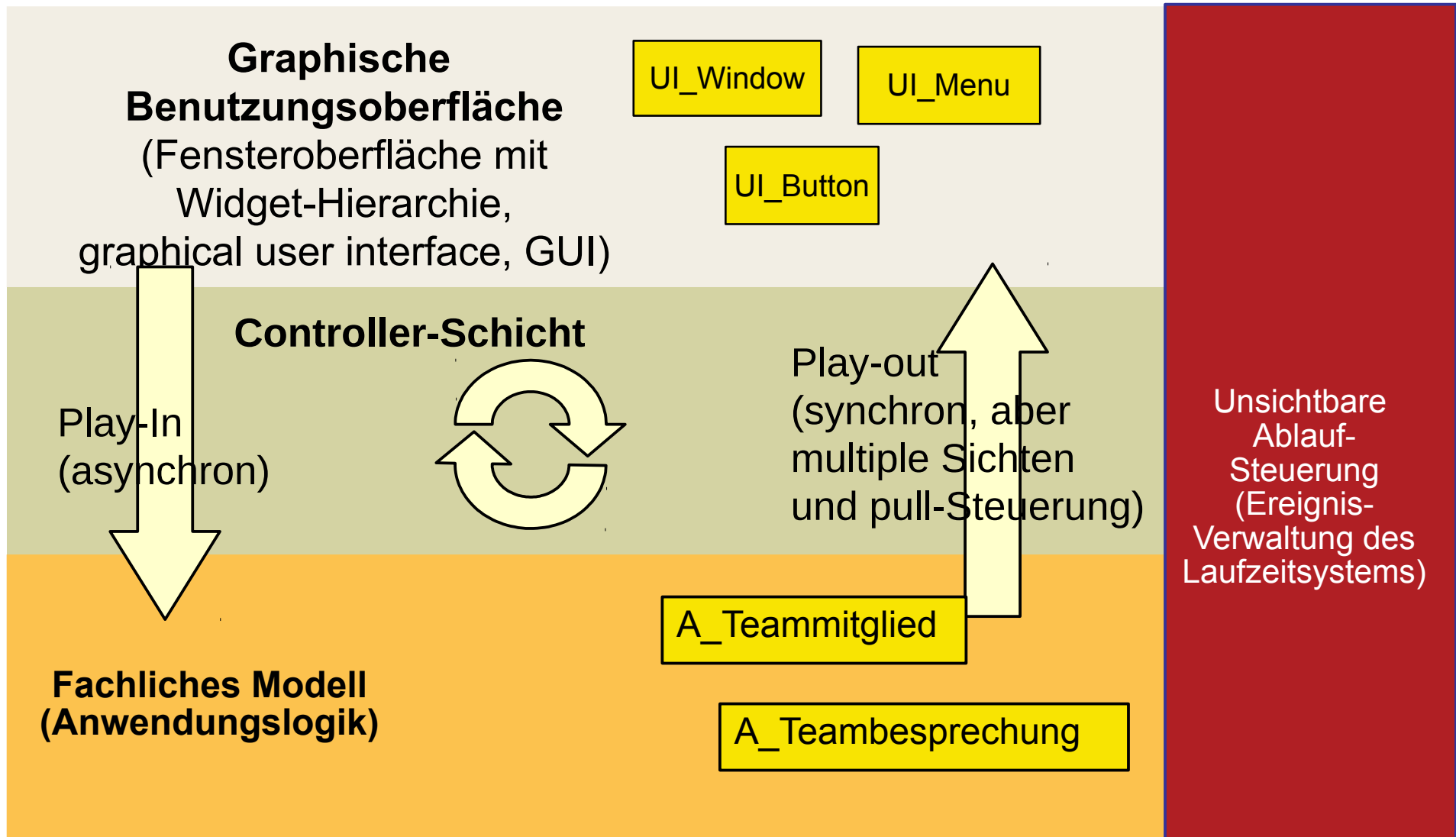
16

- Bistlang war es einfach, aber auch unflexibel
- Jetzt bringt ein *Controller* bzw. eine *Controllerschicht* die Ereignisse, “auslösenden” Fensterelemente (Sicht) und Modell asynchron zusammen
  - Der Controller beherrscht und kapselt die Interaktion, die Initiative geht von ihm aus
  - View und Modell sind gegenüber ihm passiv



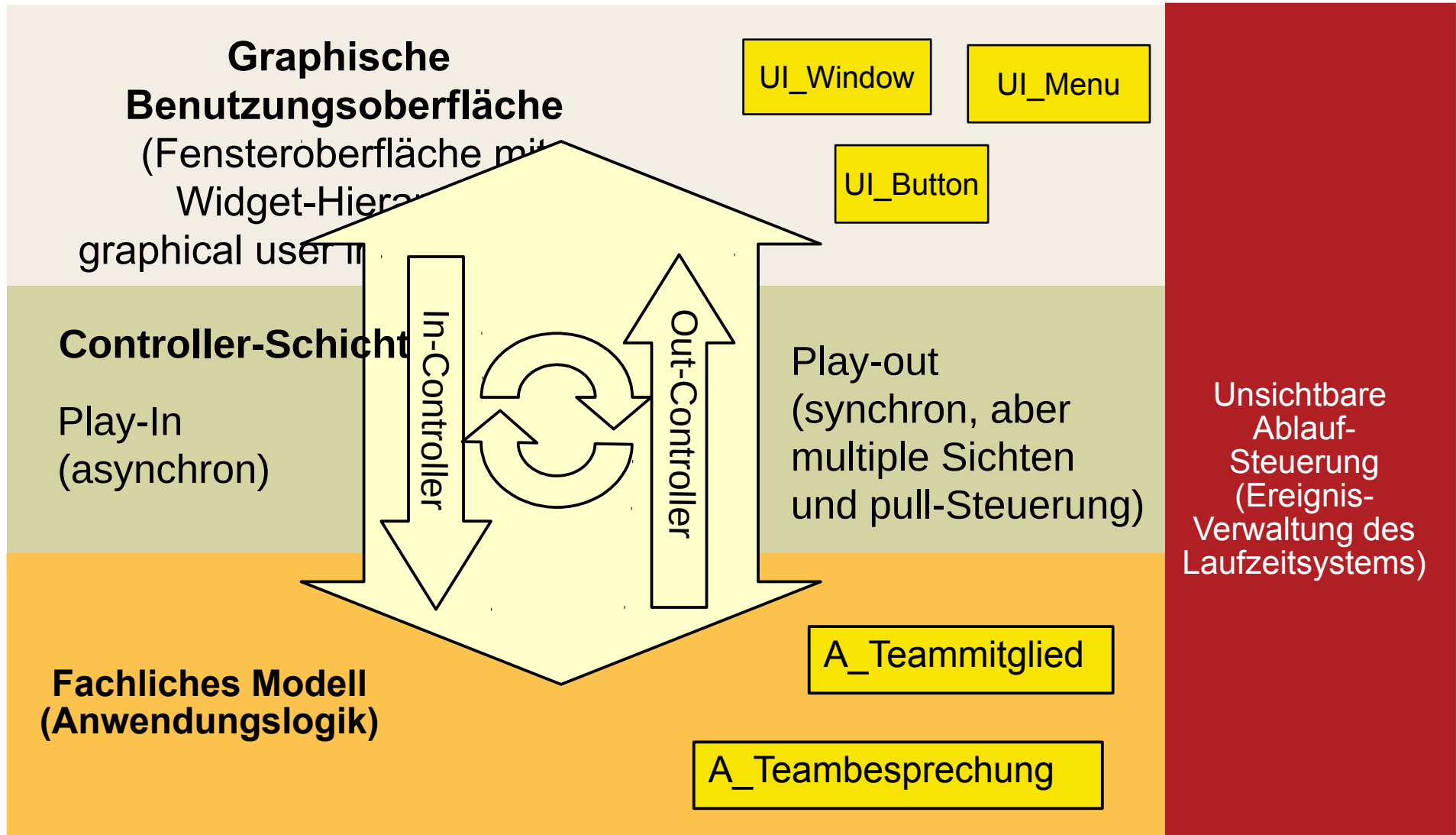
# Schichtenarchitektur der reaktiven Benutzungsoberfläche (GUI)

17



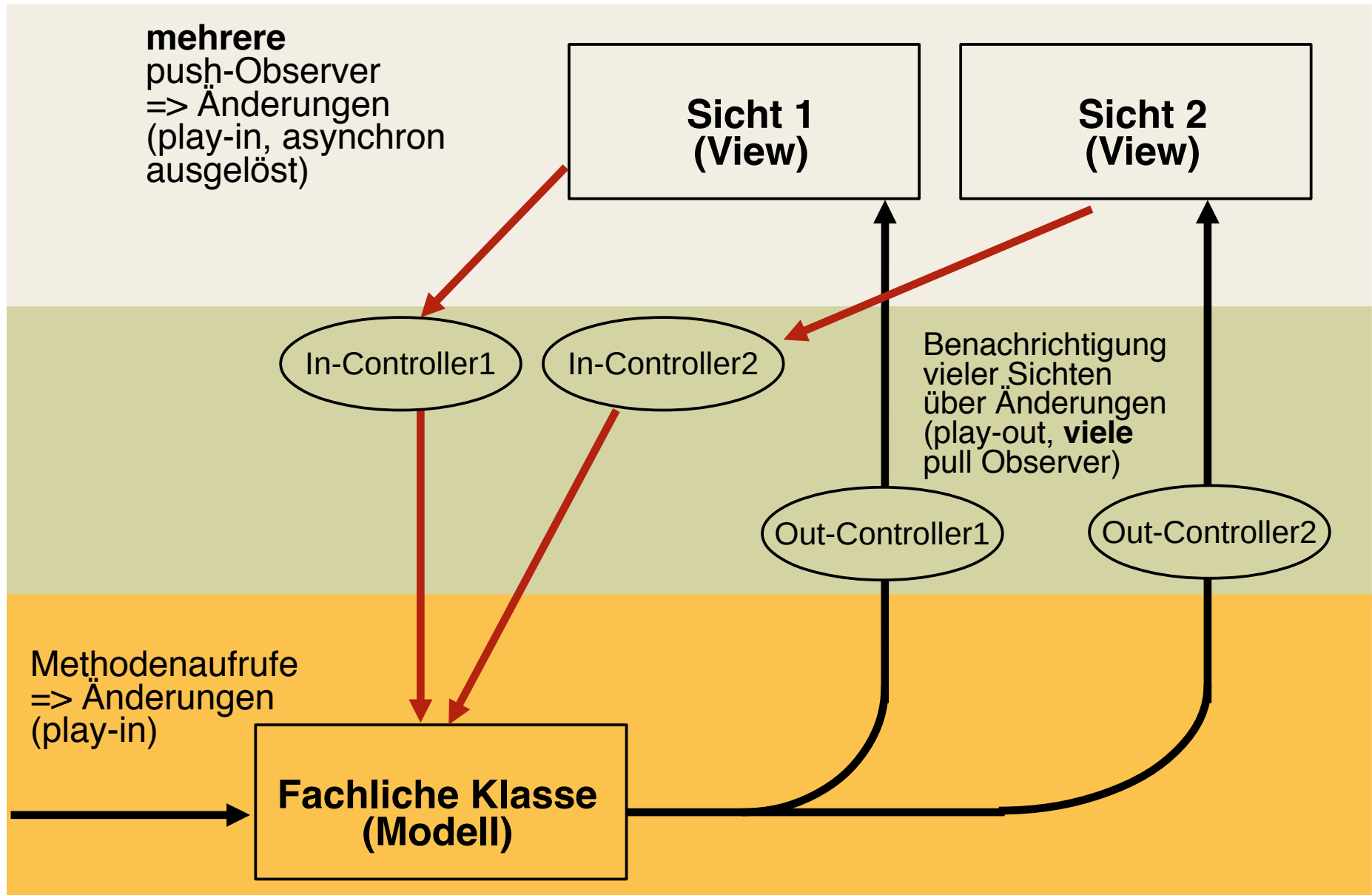
# Schichtenarchitektur der reaktiven Benutzungsoberfläche (GUI)

18



# Modell, Controller und Views in strikter Schichtung

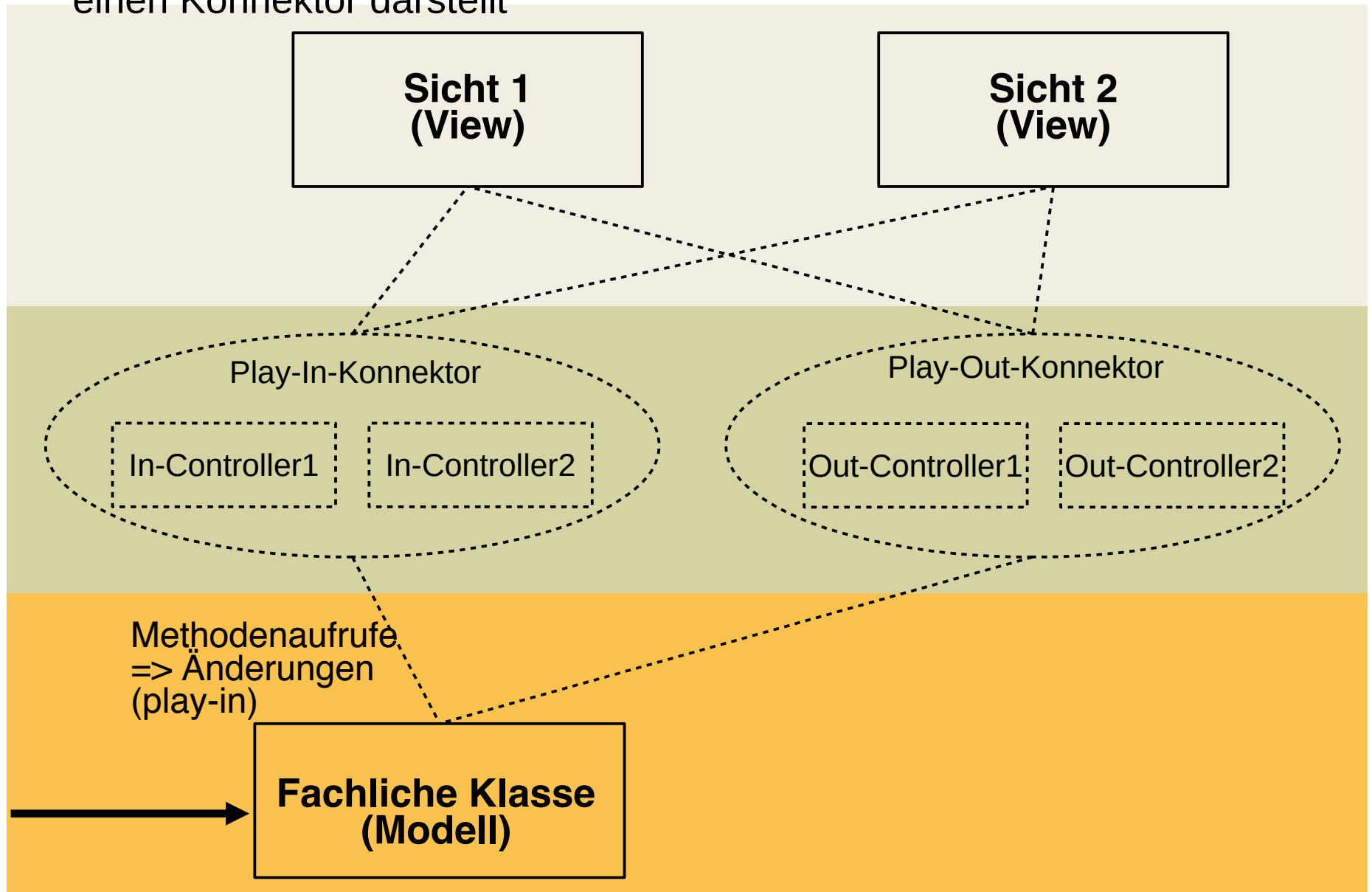
19



# Controller sind Konnektoren zwischen Model und View

20

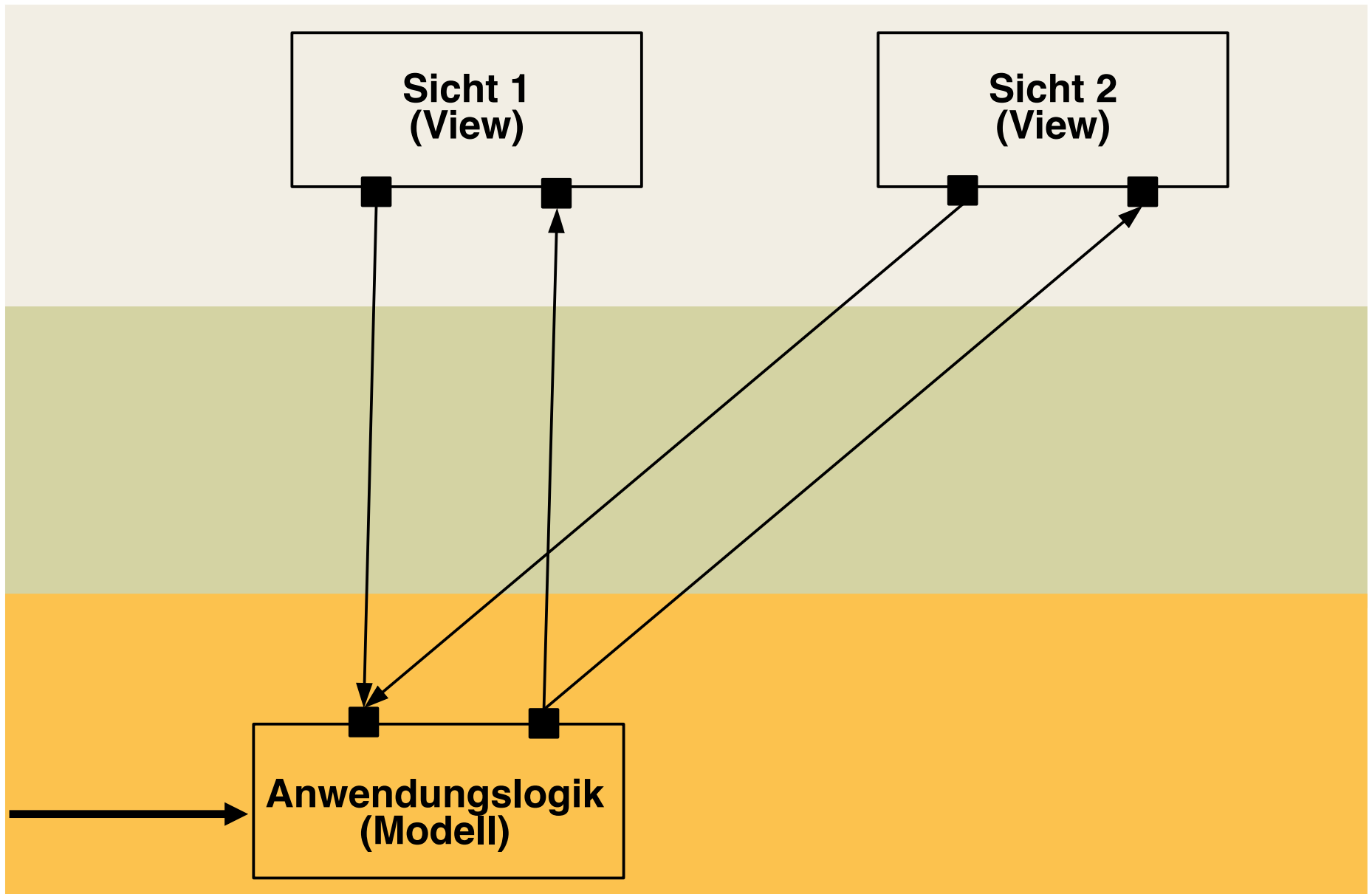
- ▶ Meist existiert ein Hauptobjekt in der Kollaboration, womit der Controller einen Konnektor darstellt



# Controller sind Konnektoren zwischen Model und View

21

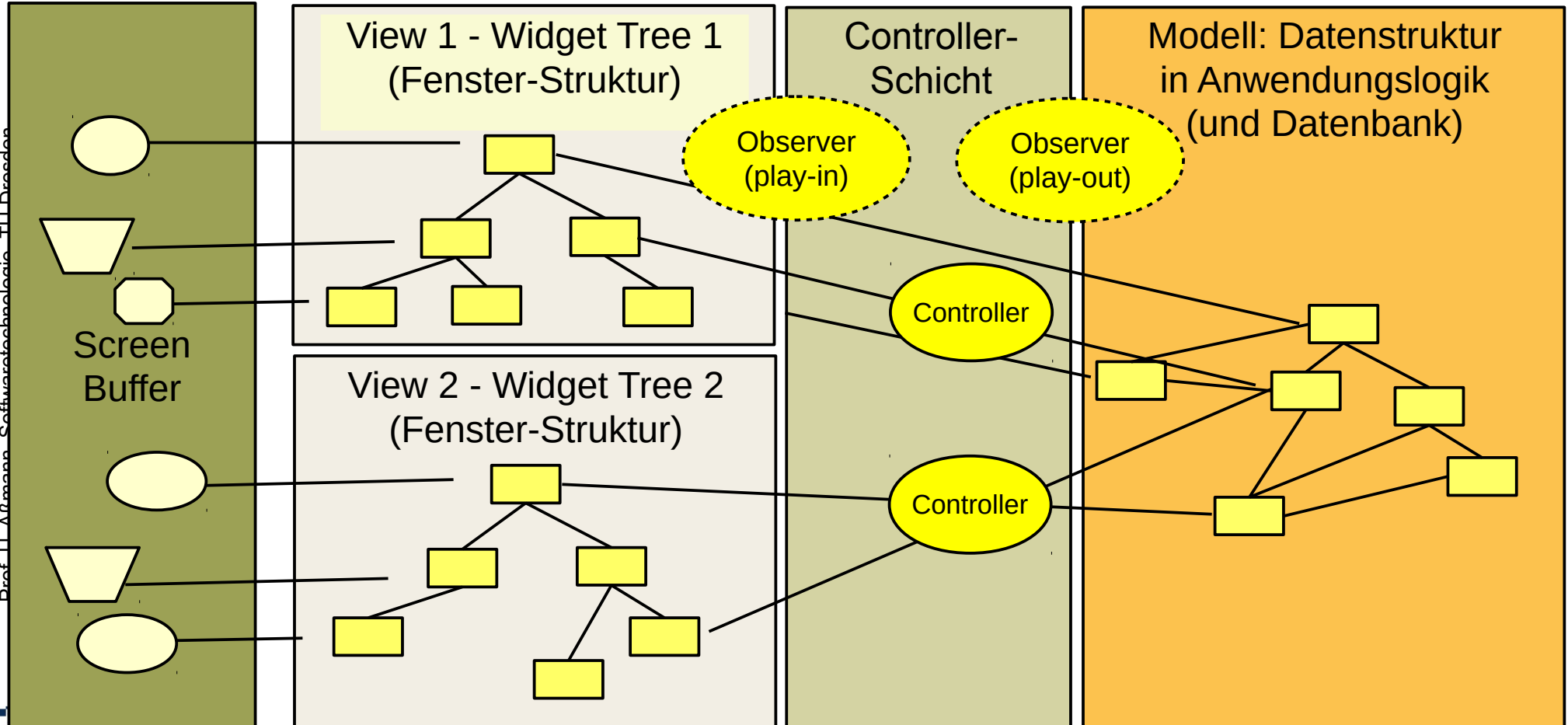
- ▶ Konnektoren, die Ports verbinden, abstrahieren die Kollaborationen



# Widgets und Datenstrukturen in asynchronen GUI

22

- ▶ Fensterstrukturen sind hierarchisch (Einkapselung von Widgets)
- ▶ Datenstruktur in Anwendung wird den Widget-Hierarchien zugeordnet
- ▶ Screen-Buffer zeigt die Widget-Struktur bitweise (`paint()`)
- ▶ Pro View ein Controller



# Programme mit asynchronen GUI laufen in 3 Phasen

23

## ► Interaktive Anwendungen mit GUI laufen in Phasen:

### 1) Aufbau der Schichten: Aufbau der Datenstrukturen

- a) Aufbau der Anwendungslogik
- b) Aufbau der Controllerschicht (Aufbau der Konnektoren)
- c) Aufbau der Widget-Schicht (widget hierarchies): Hierarchischer Aufbau der Fensteroberfläche durch Konstruktoraufrufe an Widgets und Einfügen in Fensterhierarchie (widget embodiment)

### 2) Netzaufbau

- a) **Vernetzung der Fensteroberfläche mit der Anwendungslogik** über die *Controller-Konnektoren*, um Reaktionen der Anwendungslogik zu ermöglichen
  - a) **Play-Out-Kollaboration:** Anschluß des GUI-Reaktionscodes auf Veränderungen der Modellstruktur (View wird pull-Observer des Controller, indirekt des Modells, Vorbereitung des Play-Out)
  - b) **Play-In-Kollaboration:** Anschluß des Modell-Reaktionscode auf Benutzereingaben (Controller ist push-Observer der Widgets, Vorbereitung des Play-In)

### 3) Reaktionsphase (Reaktive, asynchrone Phase)

s. nächste Folie

# Zusammenspiel der Widget-Struktur und der Anwendungslogik

24

## 3) Reaktionsphase (Reaktive, asynchrone Phase)

- **Play-In:** bei der die Benutzeraktionen vom System (Ereignisverwaltung) als Ereignisobjekte ins Programm gegeben werden
    - ◆ **Event notification:** Ereignismeldung, dass Benutzer etwas getan hat
    - ◆ **Data transmission:** etwaiger Transfer der Daten
  - **Play-Out:** Bei der in der Anwendungslogik durchgeführten Aktionen die Fensteroberfläche auf den neuesten Stand gebracht wird
    - ◆ **Event notification:** Ereignismeldung, dass Anwendung etwas getan hat
    - ◆ **Data transmission:** Transfer der Daten zum GUI
    - ◆ **Visualization:** Neuzeichnen des GUI
- ▶ Der Steuerfluß eines GUI-Programms wird *nie* explizit spezifiziert, sondern ergibt sich aus den Aktionen des Benutzers oder des Modells
- Die Controllerschicht hat die Kontrolle über das Verhalten
  - *reagiert* auf die Ereignisse im View und im Anwendungsmodell (reaktives System)
  - *steuert* Redraw und Aktionen auf Modell



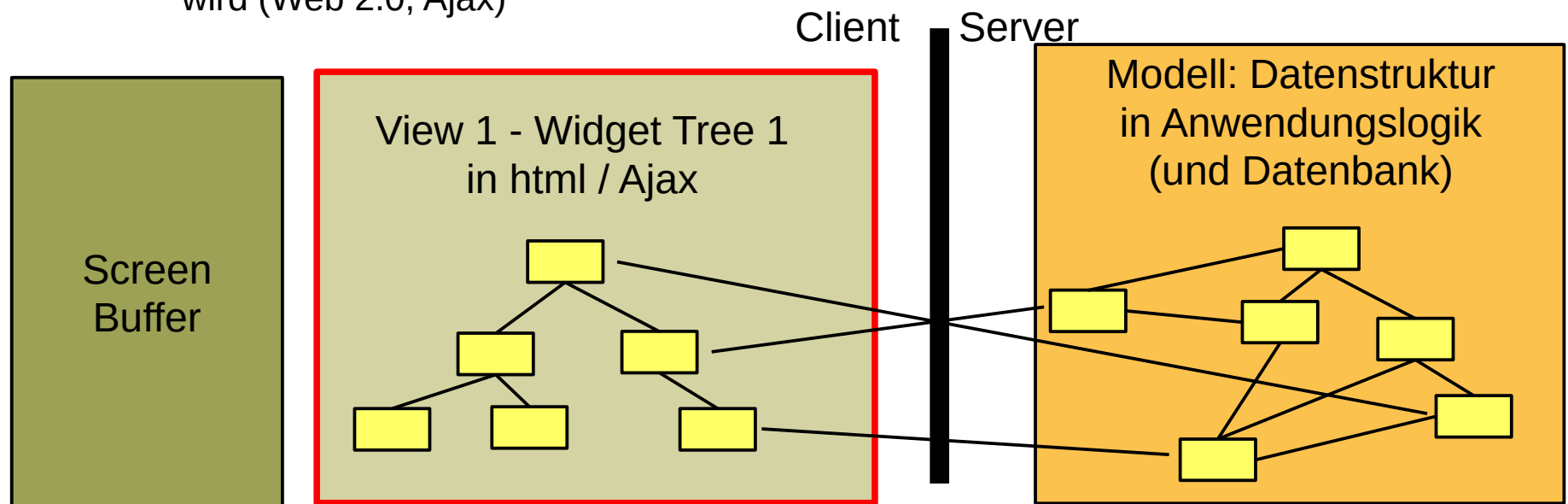
# 1) Aufbauphase Schichten:

## Aufbau der Widget-Struktur und fachl. Modell

25

Verschiedene Techniken für den Aufbau der Datenstrukturen:

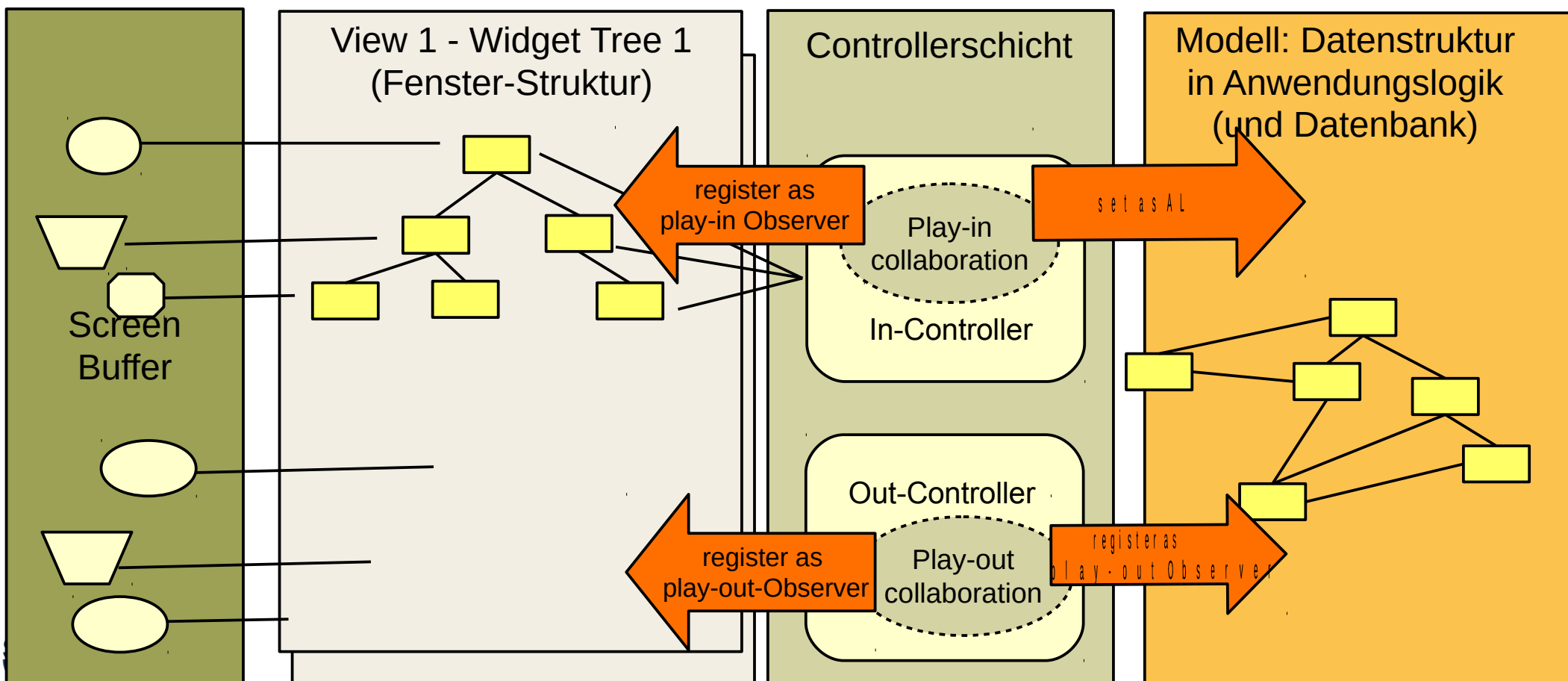
- **Rich Client:** Durch Konstruktoraufrufe und Additionen von Unterwidgets zu Oberwidgets (encapsulation)
  - rein in Java-AWT/Swing, mit expliziter Konstruktion der Widget-Hierarchien
- **App:** App-Frameworks wie Android oder iOS
- **Web:** z.B. Durch einen HTML-Baum, der von einem Brauser interpretiert wird (für Webanwendungen)
  - Durch einen XML-Baum, der von einem XML-Parser eingelesen und als Objekt-Struktur im Speicher abgelegt wird (XUL - Firefox, XAML - Vista)
  - Durch einen HTML-Baum, der bei Veränderungen inkrementell im Brauser nachgeladen wird (Web 2.0, Ajax)



# Phase 2) Netzaufbauphase: Aufbau der Verbindung

26

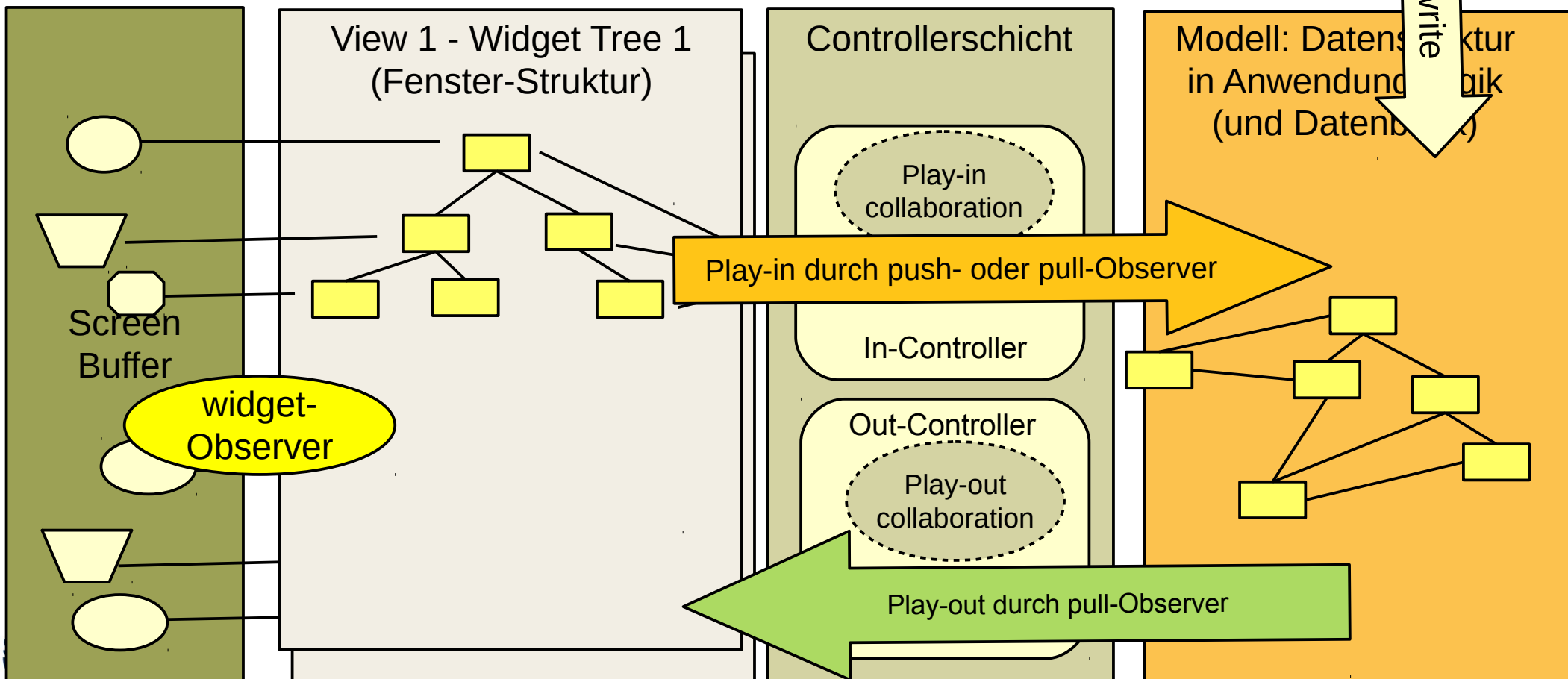
- ▶ Die Netzaufbauphase verbindet mit Kollaborationen
  - GUI, Input-Controller und AL für Play-In (In-Connector)
  - AL, Output-Controller und GUI für Play-Out (Out-Connector)



# Phase 3) Überblick MVC Dynamik

27

- ▶ **Model** ist passiv. Der Controller interpretiert die Eingaben und schreibt das Modell entsprechend
- ▶ **View** ist weitg. passiv. Controller benachrichtigt View, wenn sich was im Modell geändert hat
- ▶ **In-Controller** ist ein Observer, der wenig Daten (Events) zu transferieren hat, kann also als push-Observer oder pull-Observer implementiert werden; meist push-Observer
- ▶ **Out-Controller** muss u.U. große Datenmengen transferieren und wird meist als pull-Observer realisiert



# Die Dynamik der Phase 3 (“Life”)

28

Phase 3 (Dynamik) trennt zwischen Ereignisverarbeitung und Datentransport

Die Konnektoren setzen push- oder pull-Observer-Muster ein

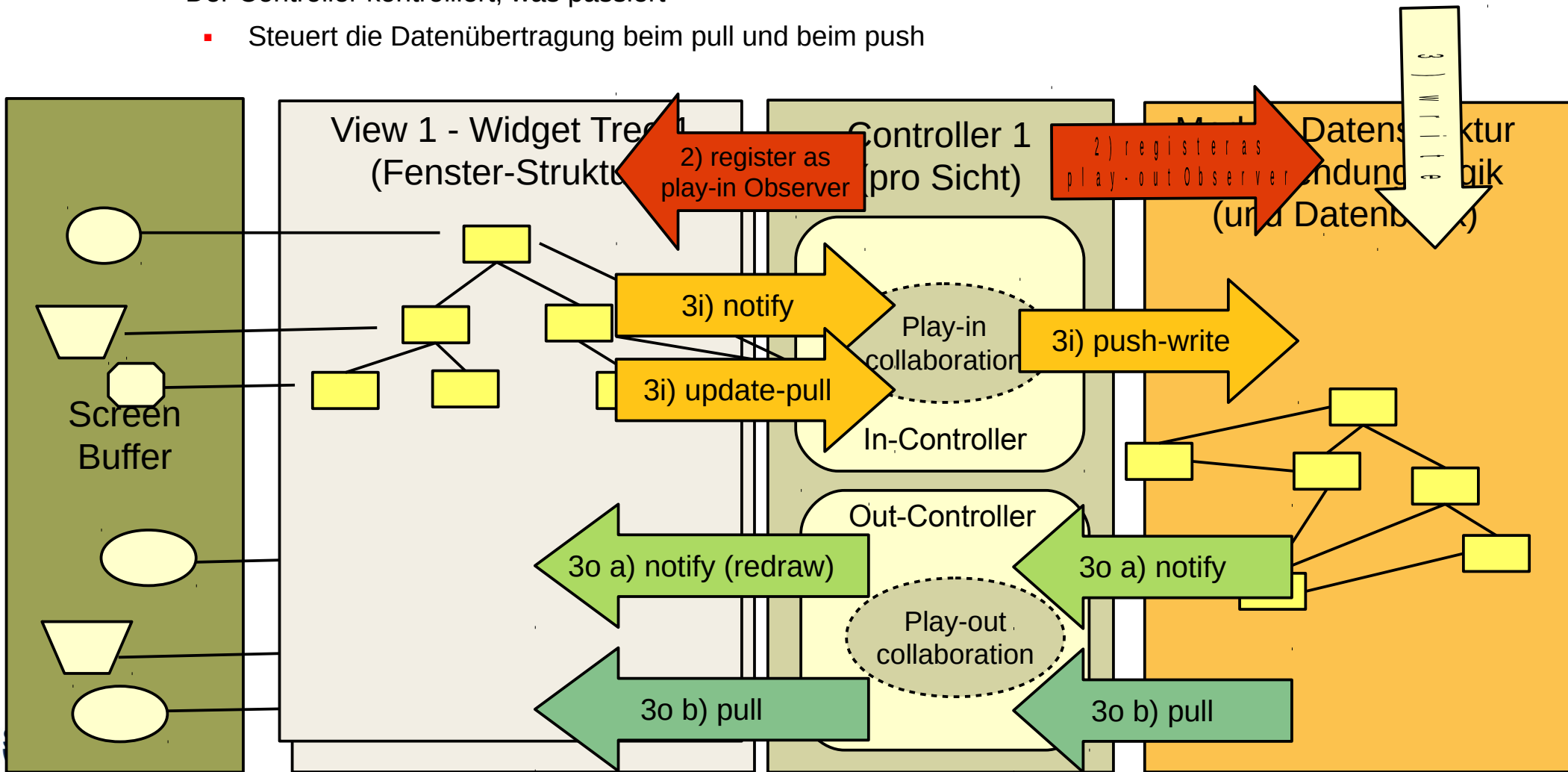
Phase 3 behandelt Verteilung (Web) mit unterschiedlichen Controller-Architekturen

Frameworks geben die Architektur vor (z.B. *Spring*, Grails, Ruby on Rails)

# Gesamte MVC-Dynamik (indirektes Play-In und Play-Out)

29

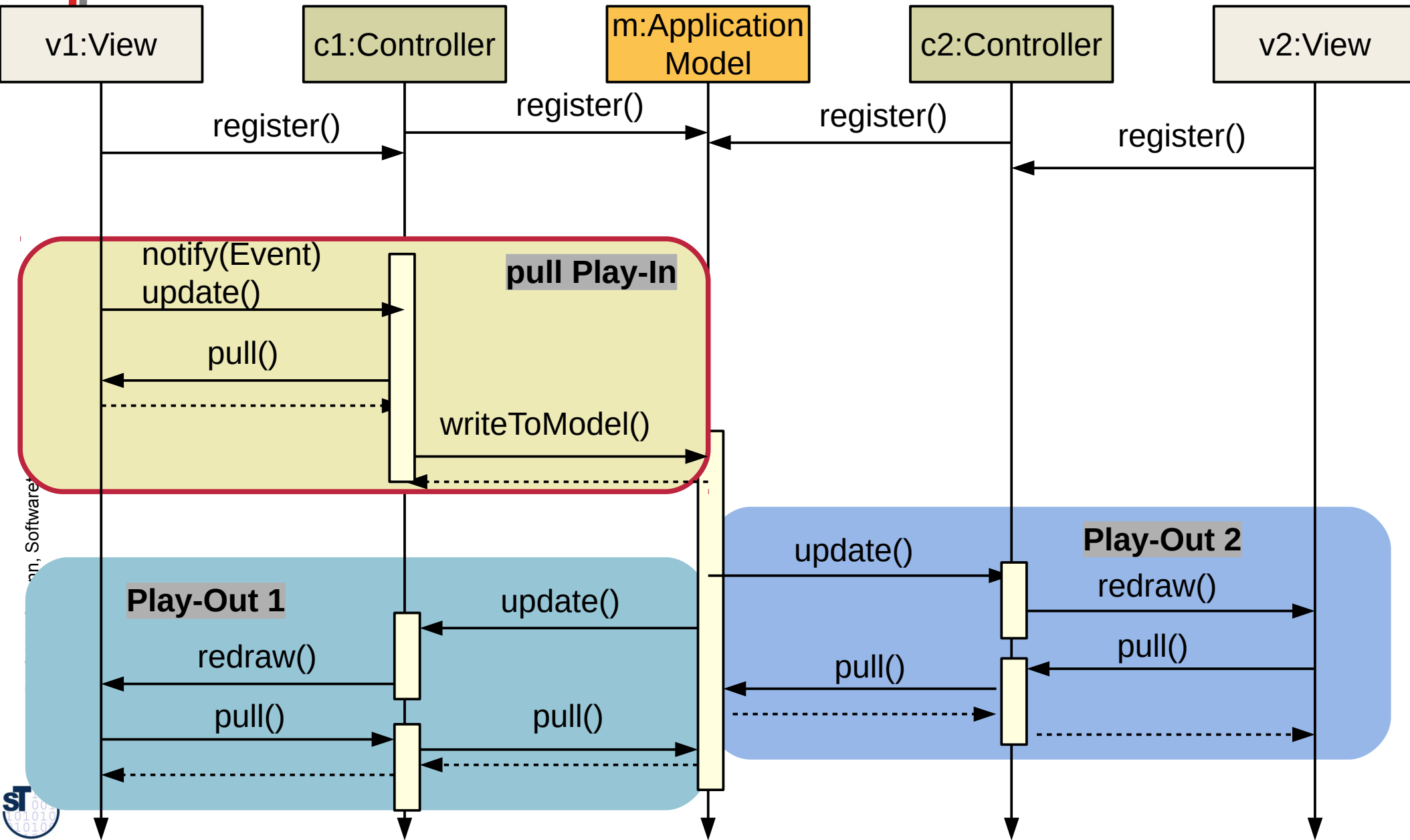
- ▶ Model ist völlig passiv, wird vom Controller geschrieben
- ▶ View is ebenfalls passiv, wird vom Controller aktiviert und gelesen
- ▶ Play-Out Observers indirektem play-out:
  - greift *indirekt* über den Observer auf das Modell zu (update, data-pull)
- ▶ Der Controller kontrolliert, was passiert
  - Steuert die Datenübertragung beim pull und beim push



# Play-In mit passivem View und pull-In-Controller; Passives Play-Out mit indirektem pull-Out-View

[PassiveView]

30



Software



# 42.4 Controller als Steuerungsmaschinen in Konnektoren

32

# Implementierung der Controller

33

Ein Controller-Konnektor wird durch eine Steuerungsmaschine implementiert, die die Ereignisse auf der Fensterhierarchie (UI) in die Aufrufe an die Anwendungslogik *übersetzt* (u.u.)

- ▶ Ereignisse auf der Fensterhierarchie (UI)
  - Button-Pressed, WindowClosed, MenuItemSelected, etc.
- ▶ Aufrufe an die Anwendungslogik:
  - Erzeugen von Kommandoobjekten
  - Schreiben auf Materialien (Domänenobjekte)
  - Aufrufen von Tools und Workflows

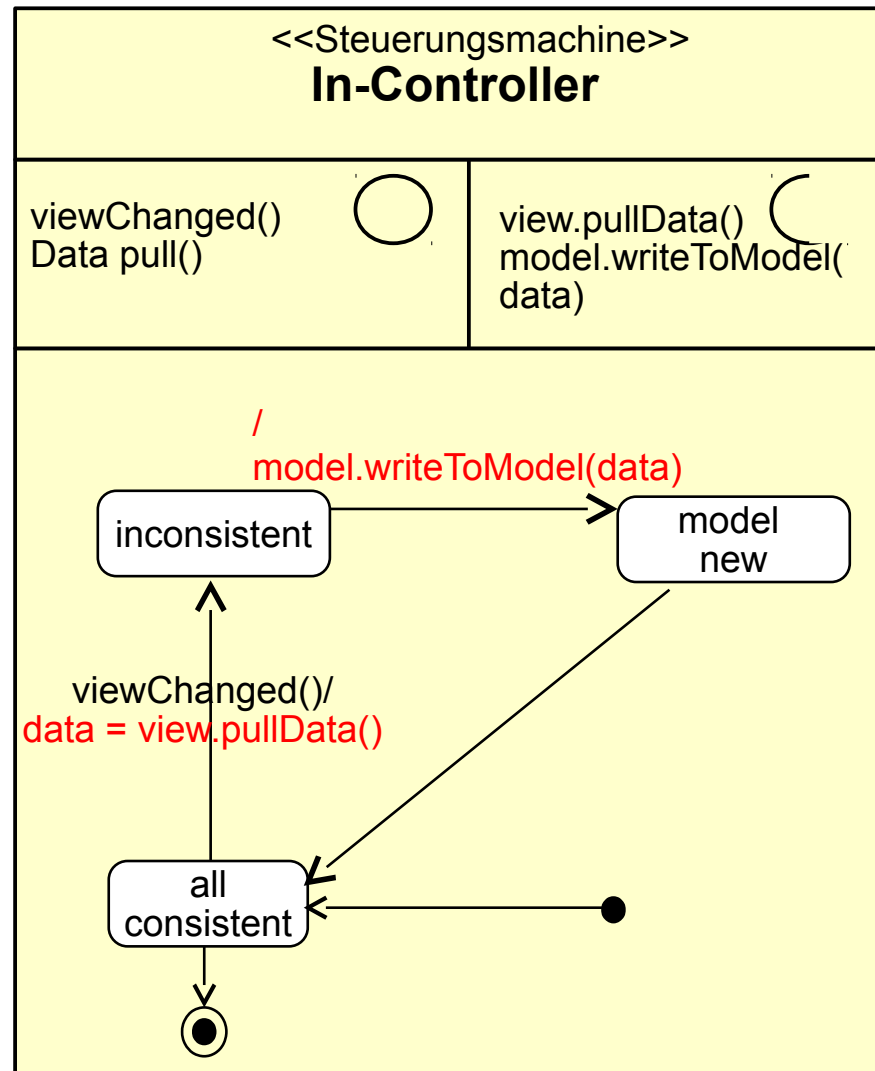
Ein In-Controller übersetzt die Ereignisse des UI in die Ereignisse der AL.  
Ein Out-Controller übersetzt die Ereignisse der AL in die Ereignisse der UI.  
Beide können kombiniert sein.



# Input-Controller mit drei Zuständen

34

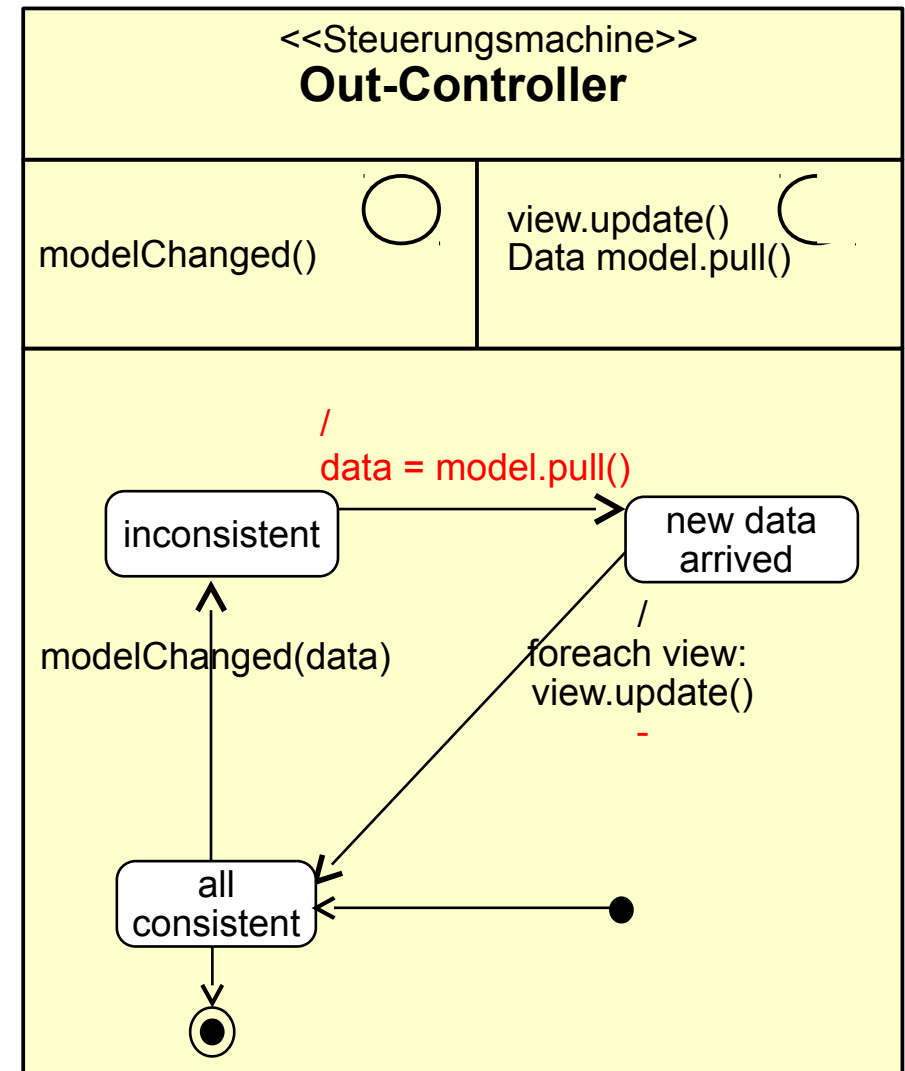
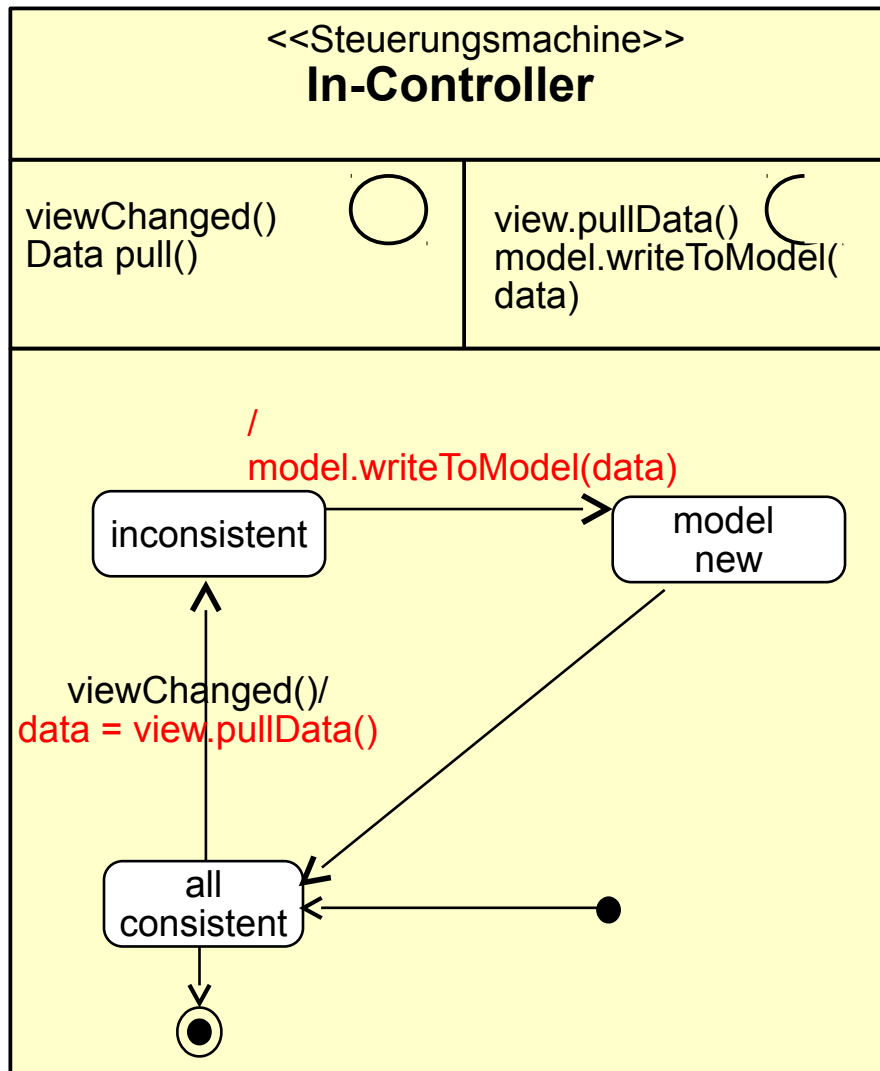
- ▶ Die entstehende Steuerungsmaschine steuert View und Model an (“beherrscht” sie)
- ▶ Getriggert wird sie durch die Ereignisse `viewChanged`. Sie löst `pullData` und `writeToModel` aus



# Paare von Controller-Steuerungsmaschinen

35

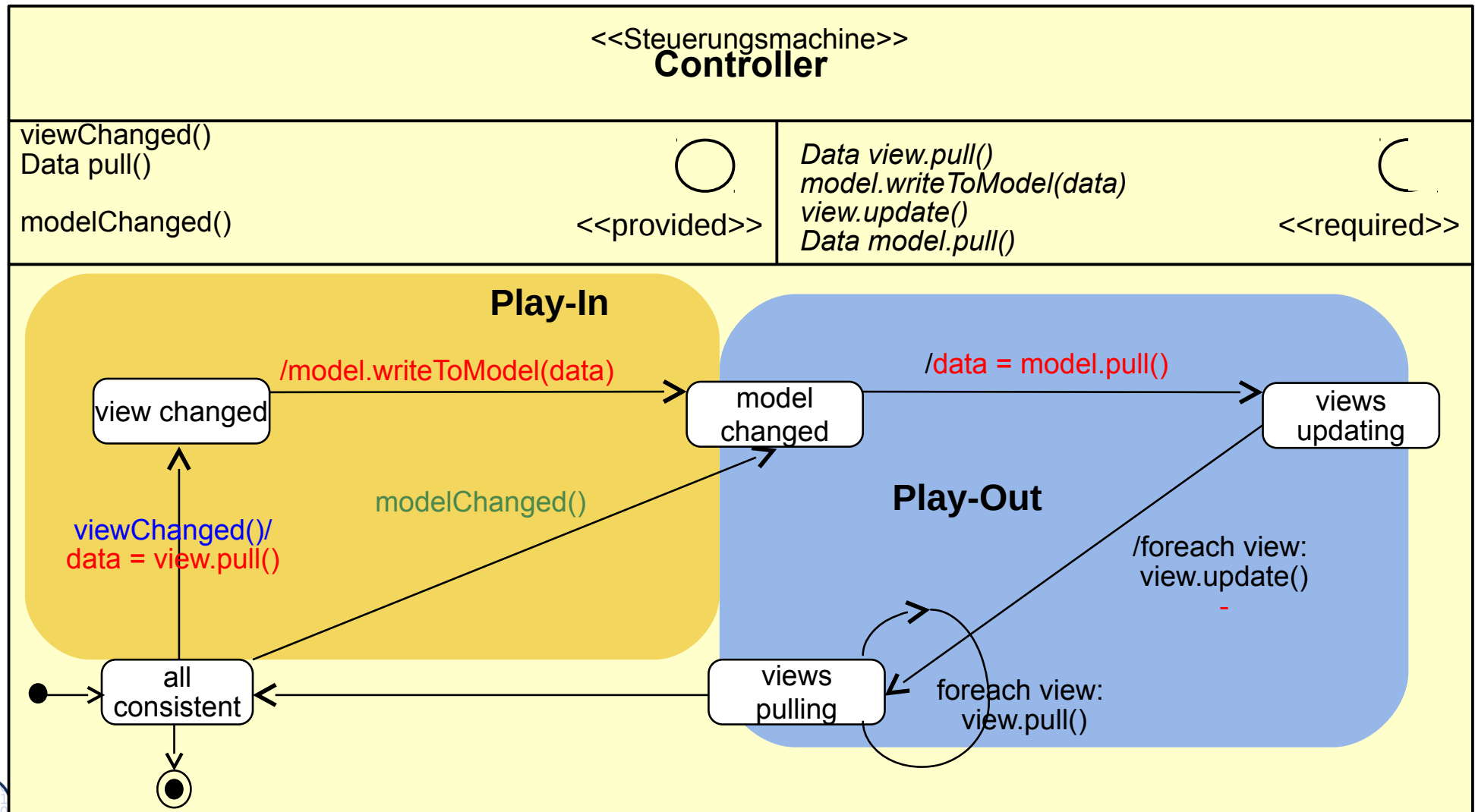
- ▶ In- und Out-Steuerungsmaschinen bilden Elemente einer oder mehrerer In- und Out-Kollaborationen



# Controller als bidirektionale Konnektoren

36

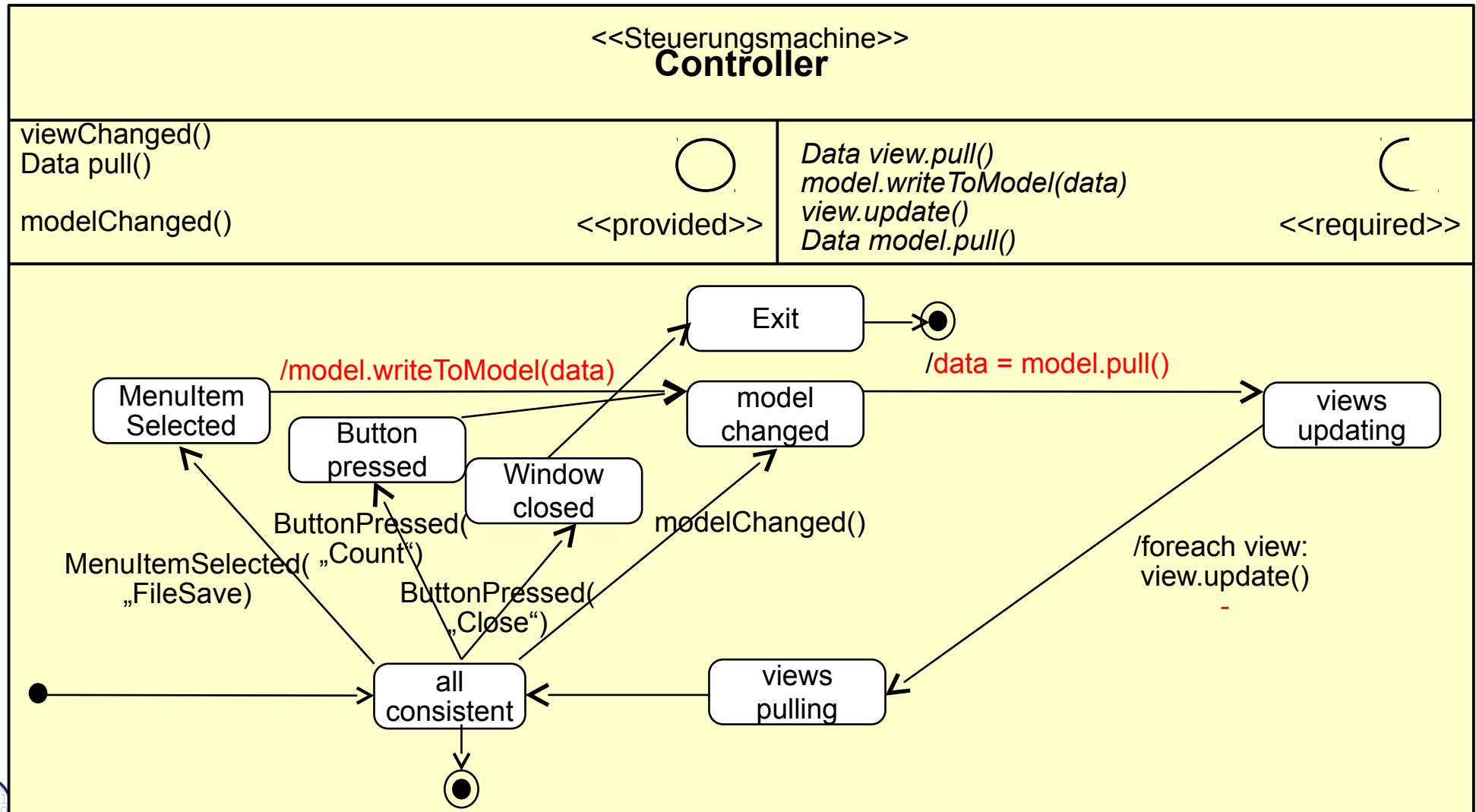
- ▶ In- und Out-Controller können auch *zusammengelegt* sein (z.B. in Spring)
- ▶ Die entstehende Steuerungsmaschine steuert View und Model an (“beherrscht” sie)
- ▶ Getriggert wird sie durch die Ereignisse **viewChanged** und **modelChanged**



# Prinzipieller Aufbau von Controllern

37

- Die Steuerungsmaschine des Controllers kennt viele verschiedene Ereignisse des UI und kann sie in spezifischen Zuständen behandeln



# Implementierung der Controller

38

- ▶ Die Controllerschicht wird realisiert als
  - Konnektor mit einer Steuerungsmaschine
  - Menge von In-/Out-Konnektoren mit kommunizierenden Steuerungsmaschinen
- ▶ Controller können kombiniert, oder auch als Paare von kommunizierenden Steuerungsmaschinen auftreten:
  - Der **Input-Controller** ist eine Steuerungsmaschine, die die Ereignisse auf der Fensterhierarchie in die Aufrufe an die Anwendungslogik übersetzt
  - Der **Output-Controller** ist eine Steuerungsmaschine, die die Ereignisse in der Anwendungslogik in die Aufrufe an die Fensterhierarchie übersetzt

# 42.5 Implementierung mit Konnektoren

39



# Ein einfacher MVC-Konnektor (als Team)

```
class MVCConnector<Model,View,Controller>{
  List<myView> views;
  myModel model;
  myController controller;
  // Phase 1: creation of layers
  MVCConnector<View,Model,View,Controller>
  () {
    views = new ArrayList<myViews>();
    model = new myModel();
    connector = new myController();
  }
  class myView extends View {
    // Inherit the View methods
  }
  class myModel extends Model {
    // Inherit the Model methods
  }
}
```

```
class myController extends Controller {
  // phase 2:
  wireNet() {
    registerView(); registerModel();
  }
  registerView() { .. }
  registerModel() { .. }

  // Phase 3: dynamics
  run() {
    .. Controller state machines ..
  }
}
```

# 42.6. MVC Frameworks

41

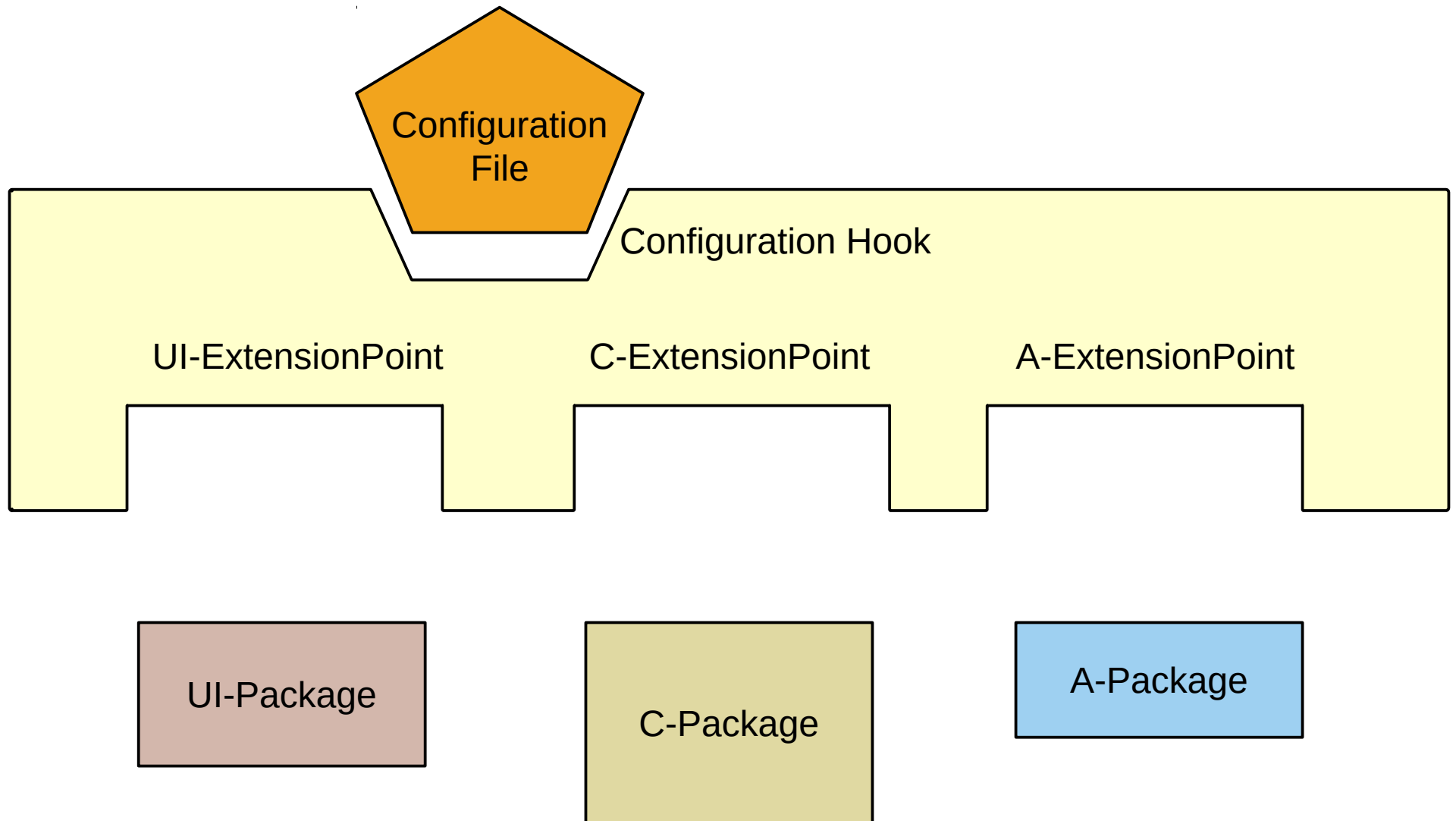




- ▶ Die Struktur einer Controllerschicht kann sich von Anwendungsklasse zu Anwendungsklasse sehr unterscheiden.
- ▶ Ein **MVC-Framework** gibt eine Struktur der Controllerschicht vor, definiert Protokolle für die Ereignismeldung und den Datenaustausch vor und kann durch den Entwickler erweitert werden.
  - MVC Frameworks benötigen Konfiguration und “Plugins”
  - **Oft folgt man dem Prinzip “Convention over configuration”:** Konventionen über Dateiverzeichnisse und Konfigurationsdateien vereinfachen dem MVC-Framework das Auffinden von Controller-, View-, Anwendungsklassen, sowie Hinweise zu ihrer Verdrahtung
  - Konfigurationsdateien meist in XML oder Java-Property-Lists
- ▶ Berühmte Beispiele:
  - Java: Spring, Struts
  - Ruby: Ruby on Rails
  - Groovy: Grails

# MVC Frameworks kennen "Plugins"

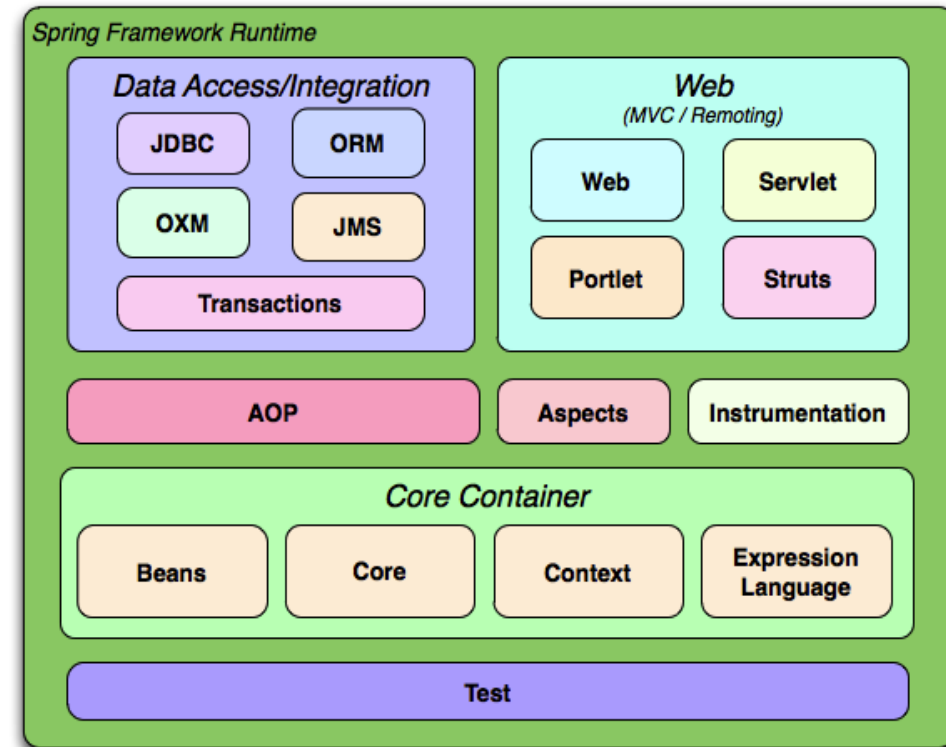
43



# Spring Framework

44

- ▶ *Spring* ist das im Praktikum im WS verwendete MVC-Framework
  - Webbasiert, d.h. Controllerschicht ist auf Client und Server verteilt implementiert
  - Konfigurierbar durch XML-Dateien und Java Property Files
  - Erweiterbar
- ▶ Das Salespoint-Framework nutzt als Konnektor zum GUI das Konnektor-Framework SPRING
  - Main controller, subcontroller
  - Web-MVC Frameworks brauchen *starke Schichtung*
  - Bietet sehr viele verschiedene Pakete, nicht nur für Web-UIs

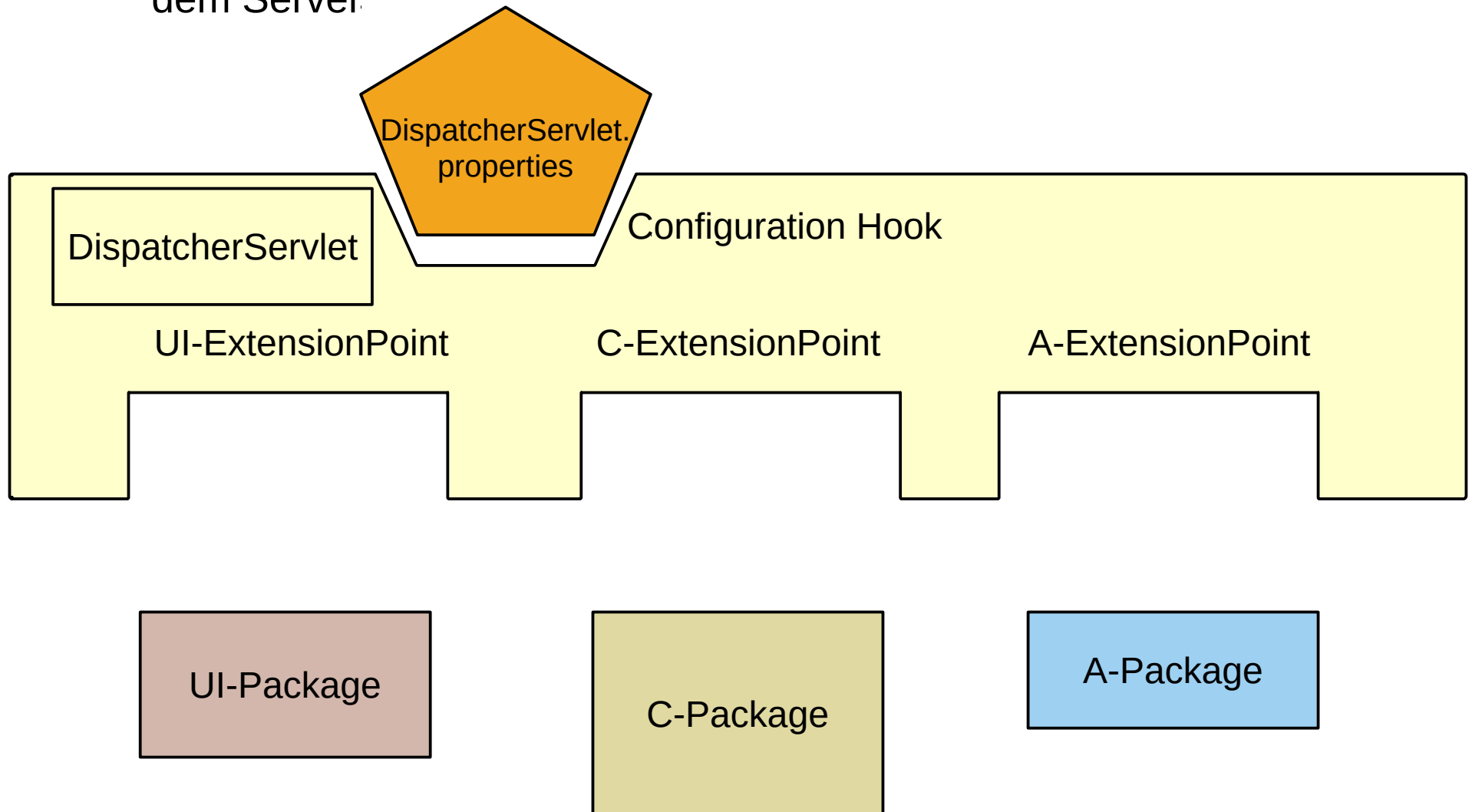


- <http://spring.io/guides>

# Spring Konfiguration

45

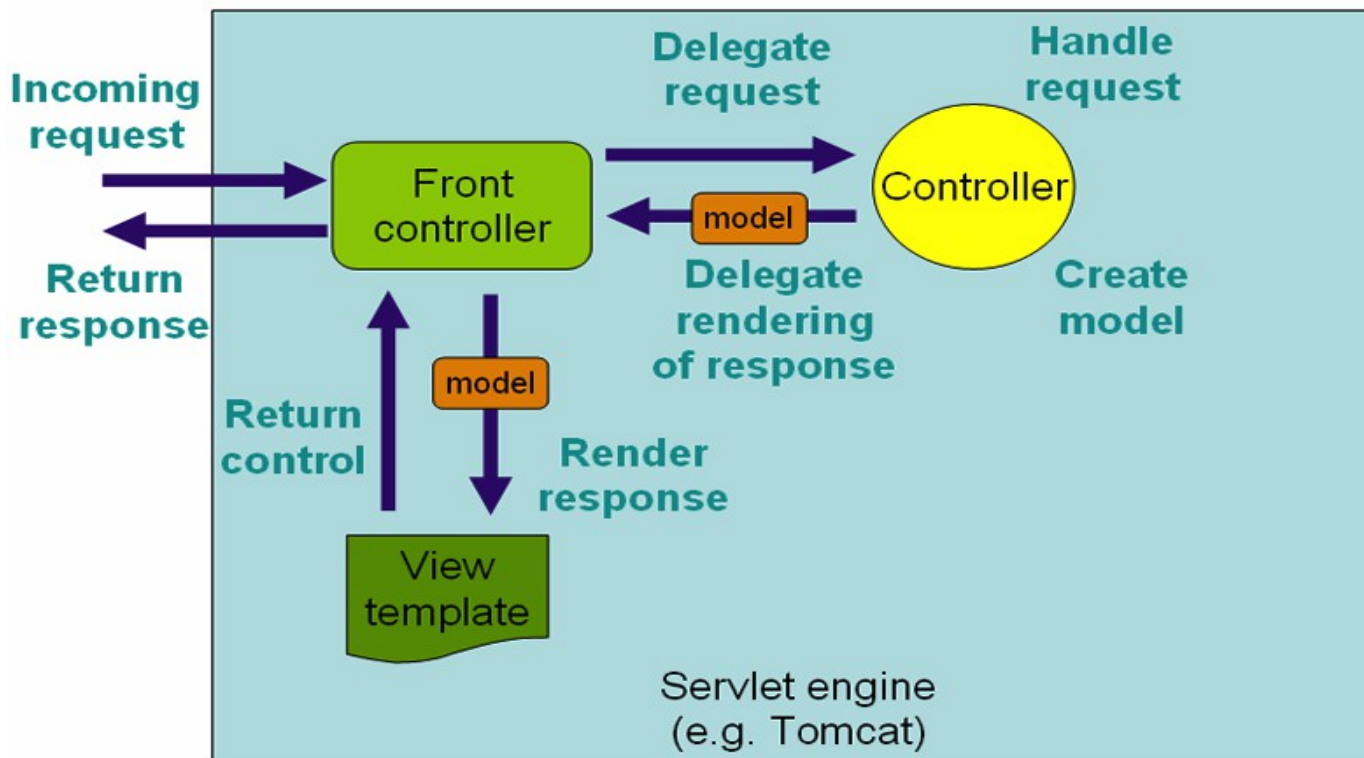
- ▶ Spring übernimmt das Management der Verteilung
  - Das Zusammenspiel zwischen Browser, Server und Anwendungslogik auf dem Server.



# Struktur des String Controllers in Web-Systemen (Server Side)

46

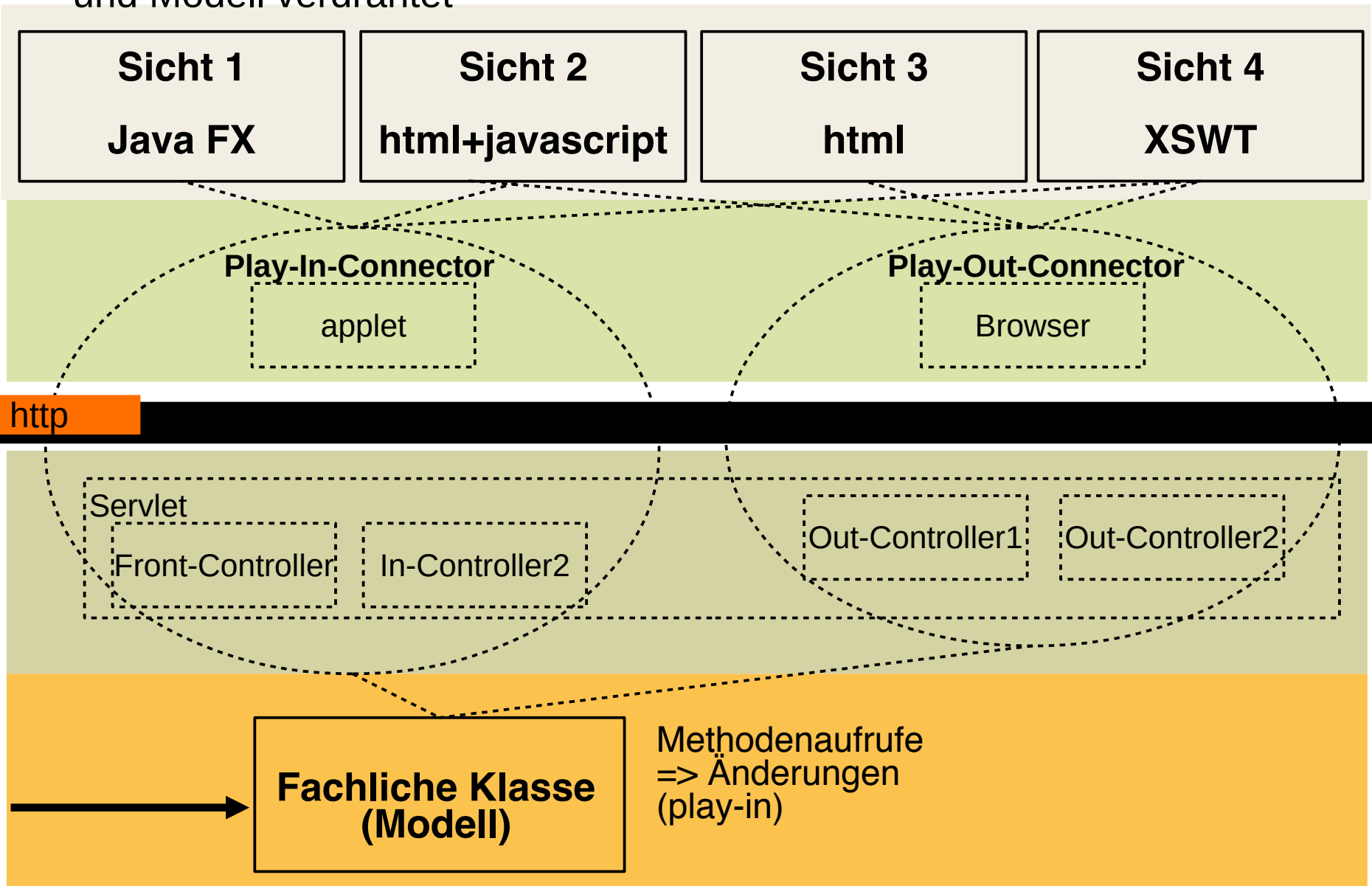
- ▶ Der Spring Controller ist ein komposites Programm auf dem Server:
  - das Spring-DispatcherServlet enthält einen “FrontController”, der das ankommende Ereignis interpretiert (Steuerungsmaschine) und an untergeordnete Controller bzw. Steuerungsmaschinen weiter leitet



# Controller sind Konnektoren zwischen Model und View

47

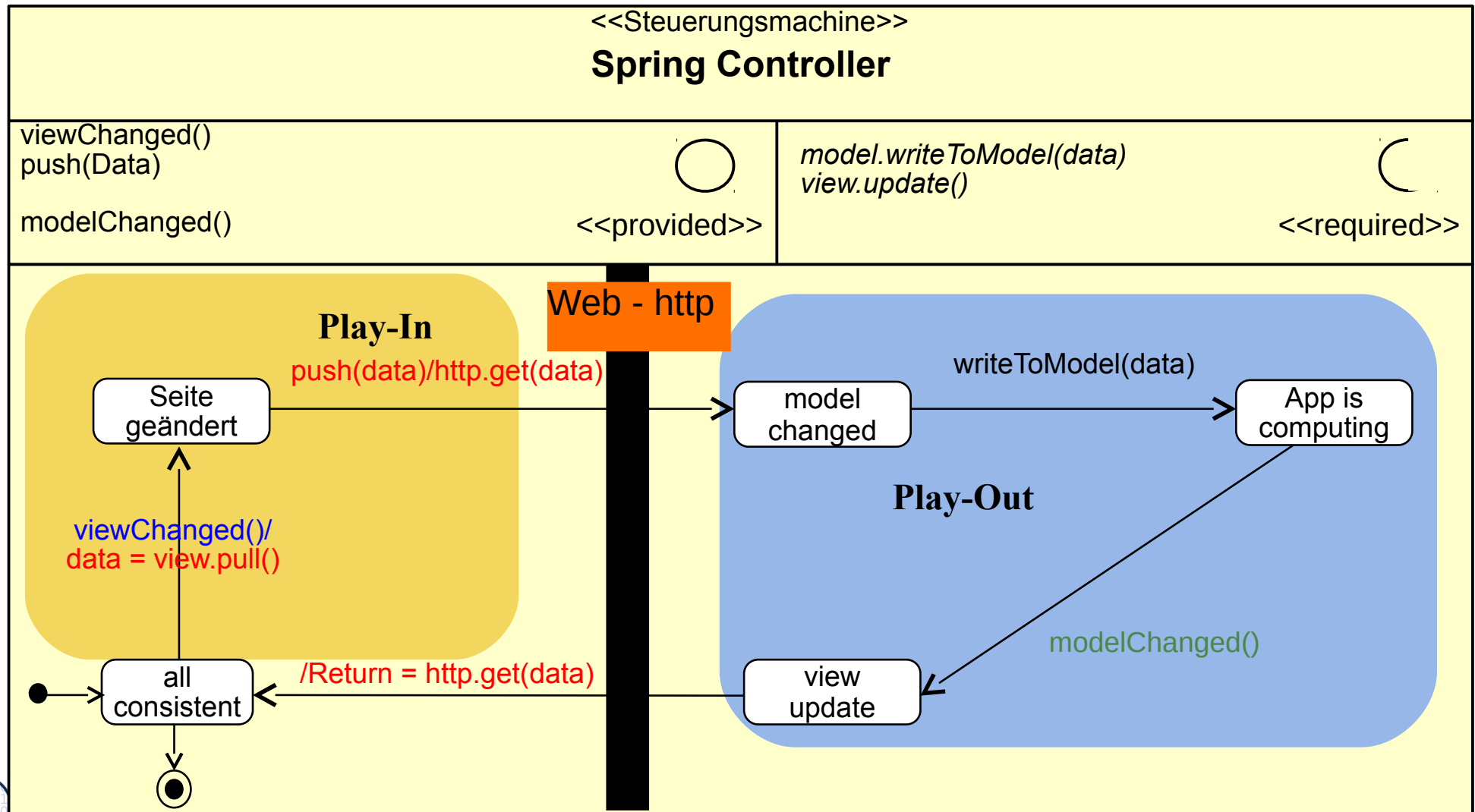
- ▶ Im Folgenden gibt es ein Hauptobjekt, den Konnektor, der View, Controller und Modell verdrahtet



# Server-seitiger Spring-Controller

48

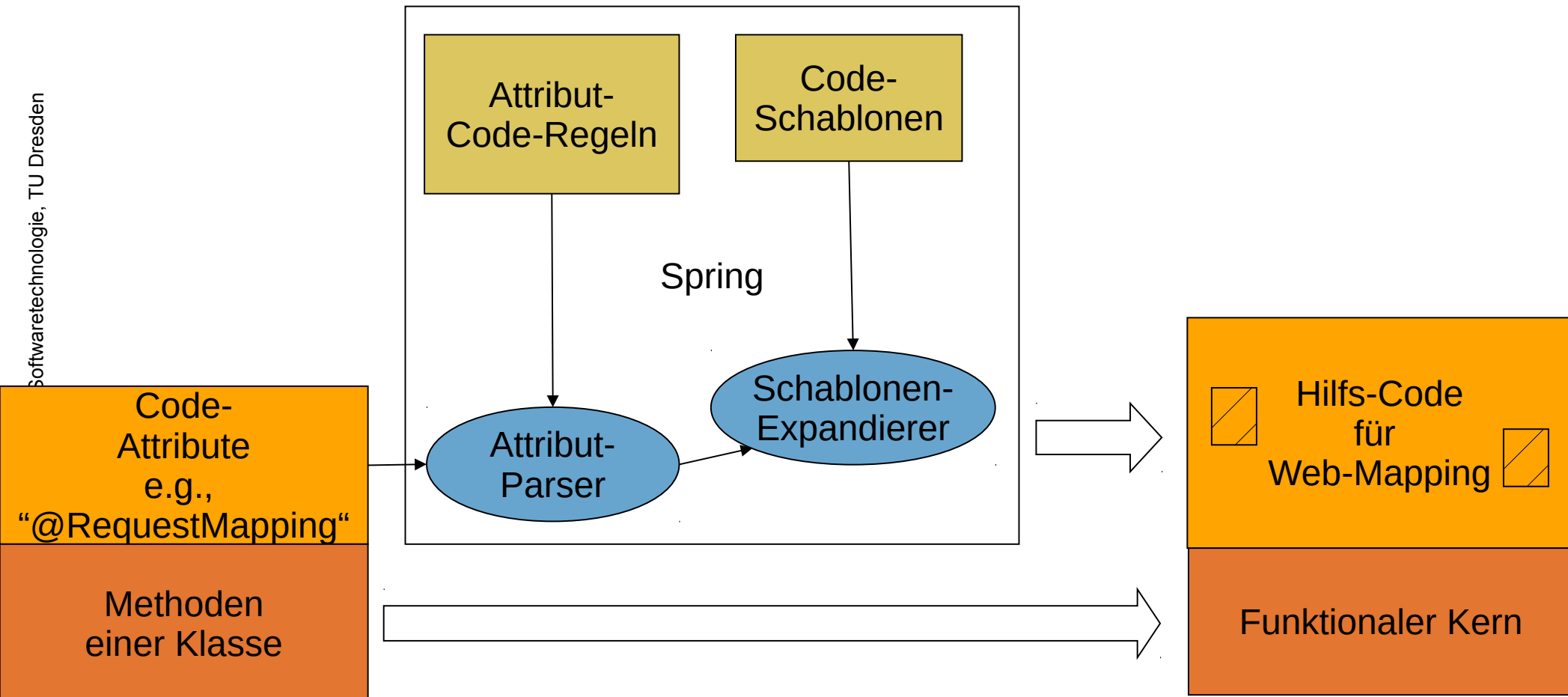
- ▶ In- und Out-Controller können auch **verteilt** sein (z.B. in Spring)
- ▶ Die Steuerungsmaschine ist verteilt auf client und server
- ▶ Getriggert wird sie durch die Ereignisse **viewChanged** und **modelChanged**



# Spring nutzt @attribut-basierte Codegenerierung

49

- ▶ Spring wandet Java-@Attribute (sog. Metadaten) in Code um
- Attribute parameterisieren Schablonen (“templates”): Template-gesteuerte Codegenerierung
- Siehe auch Xdoclet, [xdoclet.sf.net](http://xdoclet.sf.net)





# @RequestMapping

50

- ▶ Ein **REST-Webservice** bildet URL (Web-Dateinamen) auf *aktive Methoden eines Webservice-Objekts* ab
  - `@RequestMapping("<relative URL>")`
- ▶ Wird die URL im Browser aufgerufen, wird die Methode aufgerufen und ihr Resultat als String im JSON-Format zurückgegeben

<http://st.inf.tu-dresden.de>

**/showProduct**

**/layIntoBasket**

**/order**

**@RequestMapping(**  
**"/showProduct")**

**@RequestMapping(**  
**"/layIntoBasket")**

**@RequestMapping(**  
**"/order")**

**Webserver <http://st.inf.tu-dresden.de>**

**otto:Katalog**

zeigeProdukt()

legeProduktinWarenkorb()

bestelle()

# Was haben wir gelernt?

52

- ▶ GUI-Programme laufen in 3 Phasen:
  - 1) Aufbau der Fensterfronten (widget hierarchies) durch Konstruktoraufrufe und Additionen (embodiment) (auch im Browser)
  - 2) Netzaufbau, Aufbau der Controller-Kollaborationen
    - Vorbereitung Play-Out: Anschluß des View-Reaktionscodes als jdk-Observer des Modells
    - Vorbereitung Play-In: Anschluß des Controller als widget-Observer der Views
  - 3) Reaktionsphase, bei der die Benutzeraktionen vom System als Ereignisobjekte ins Programm gegeben werden:
    - der Controller als Listener benachrichtigt und ausgeführt werden (Play-In)
    - die Views bzw. der Controller als Listener des Modells benachrichtigt werden (Play-Out)
- ▶ Der Kontrollfluß eines GUI-Programms wird *nie* explizit spezifiziert, sondern ergibt sich aus den Aktionen des Benutzers
  - Die Views reagieren auf Ereignisse im Screenbuffer, die von der Ablaufsteuerung gemeldet werden
  - Der Controller auf Widget-Veränderungen im View und Änderungen im Modell
  - Der Controller wird als Steuerungsmaschine implementiert und steuert alles (aktiver Konnektor)
- ▶ Spring enthält eine stark geschichtete GUI-Anwendungskopplung
  - Enthält einen kompositen Controller (komposite Steuerungsmaschine)
  - Regelt den Verkehr zwischen Browser, Server, Servlet, Webservice-Methoden

# The End

53

- ▶ Diese Folien sind eine stark überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. used by permission. Verbreitung, Kopieren nur mit Zustimmung der Autoren.