JGoodies Karsten Lentzsch

DESKTOP-MUSTER & DATENBINDUNG

JGoodies

- Quelloffene Swing-Bibliotheken
- Beispielanwendungen
- Gestalte Oberflächen
- Berate zu Desktop und Swing

- In Expertengruppen zu JSRs 295 und 296
- Eigene Bibliothek: JGoodies Binding

Ziel

Wie organisiere ich Präsentationslogik?

Wie synchronisiere ich Fachdaten mit GUI?

Gliederung

Einleitung

Autonomous View

Model View Controller

Model View Presenter

Presentation Model

Datenbindung

Gliederung

Einleitung

Autonomous View

Model View Controller

Model View Presenter

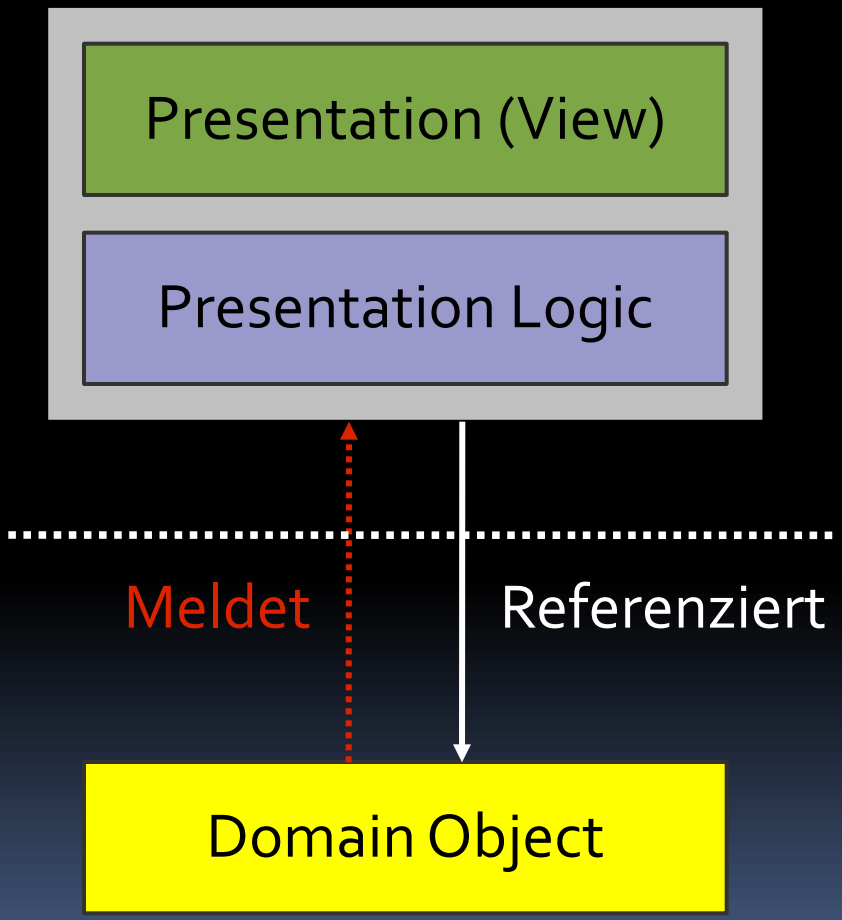
Presentation Model

Datenbindung

Fragen

- Wie strukturiere ich meine Anwendung?
- Wie baue ich einen View?
- Wer handhabt Events?
- Brauche ich einen Controller?
- Wie kann ich meine GUI-Logik testen?

Legende


Presentation (View)
Presentation Logic
Meldet
Referenziert
Domain Object

Legende

- Fach-/Geschäftslogik
- Beispiele:
  - Buch
  - Person
  - Adresse
  - Rechnung
- Allgemein: Graph

Domain Object

Legende

Presentation Logic

- Handler für:
  - Listenauswahländerung
  - CheckBox-Auswahl
  - Drag-Drop-Ende
- UI-Modelle:
  - ListModel
  - TableModel
  - TreeSelectionModel
- Swing Actions

Event Handling vs. Presentation Logic

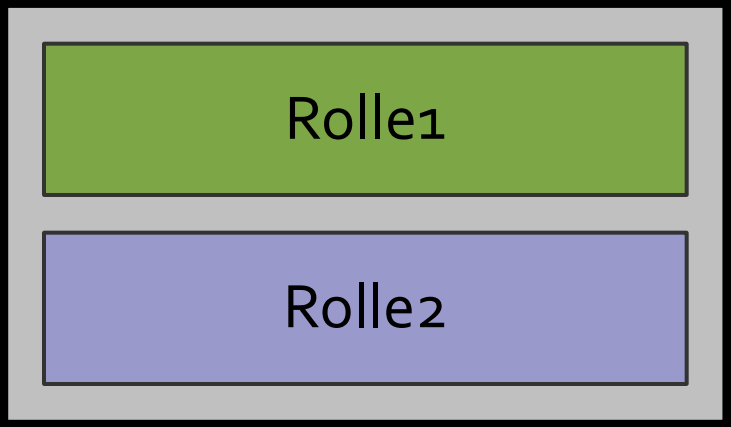
- Toolkit handhabt feinkörnige Events:
  - Mouse entered, exited
  - Mouse pressed
  - RadioButton pressed, armed, rollover
- Anwendung handhabt grobkörnige Events:
  - RadioButton selected
  - Action performed
  - Listenelement zugefügt
  - Facheigenschaft geändert

Legende

Presentation (View)

- Container
  - JPanel, JDialog, JFrame
- Enthält:
  - JTextField, JList, JTable
- Initialisierungscode
- Panel-Baucode
- GUI-Zustand:
  - CheckBox pressed
  - Mouse over

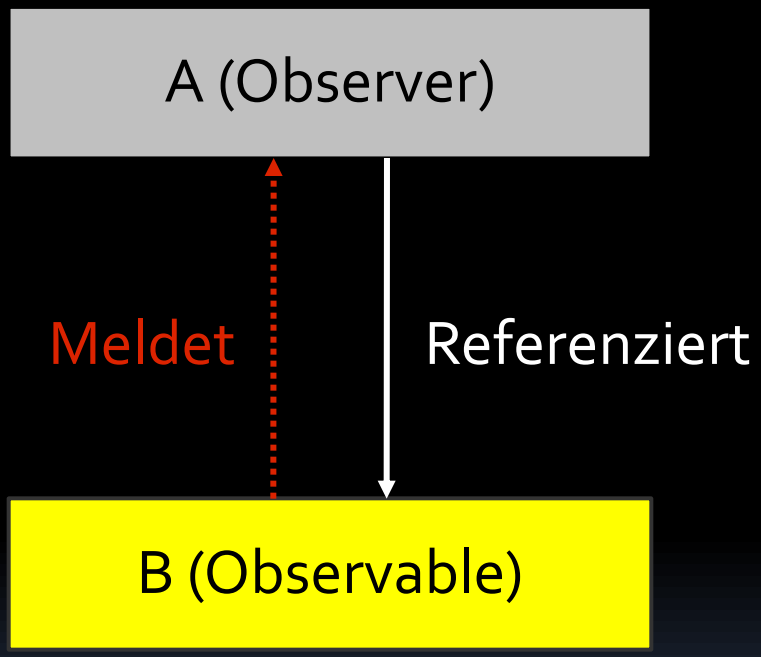
Legende



- Die Rollen "sitzen" in einer Klasse
- Können aufeinander zugreifen

- Getrennte Schichten

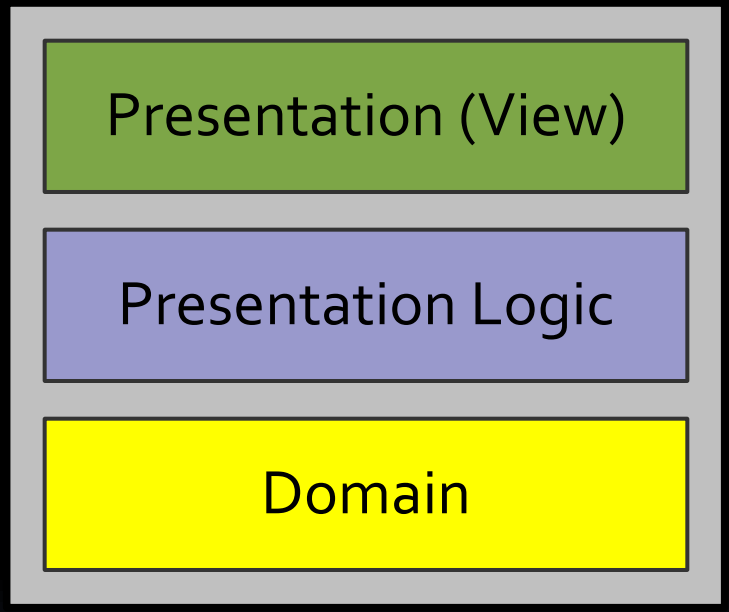
Legende



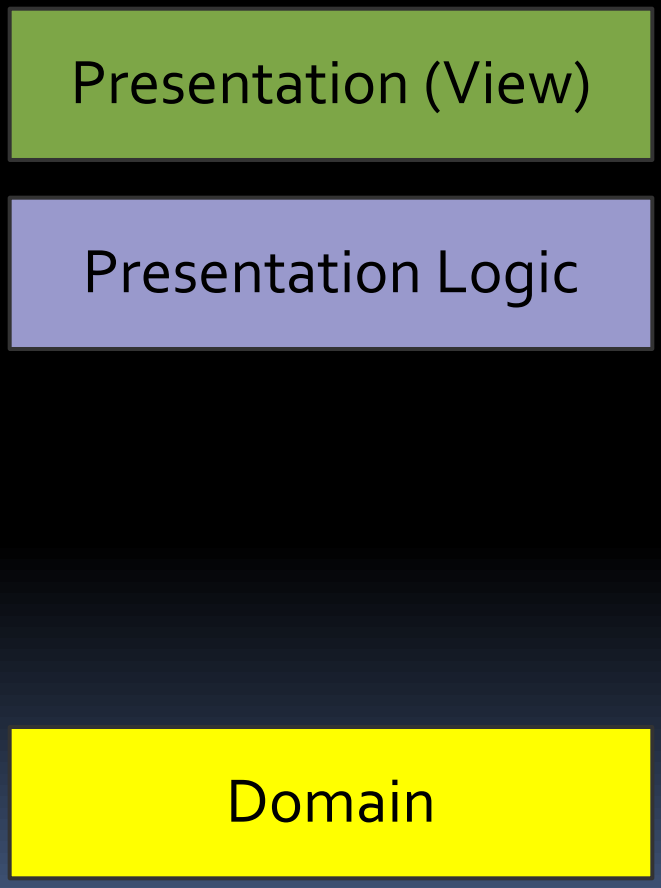
- A beobachtet Änderungen an B

- A ist ein Observer
- B ist ein Observable

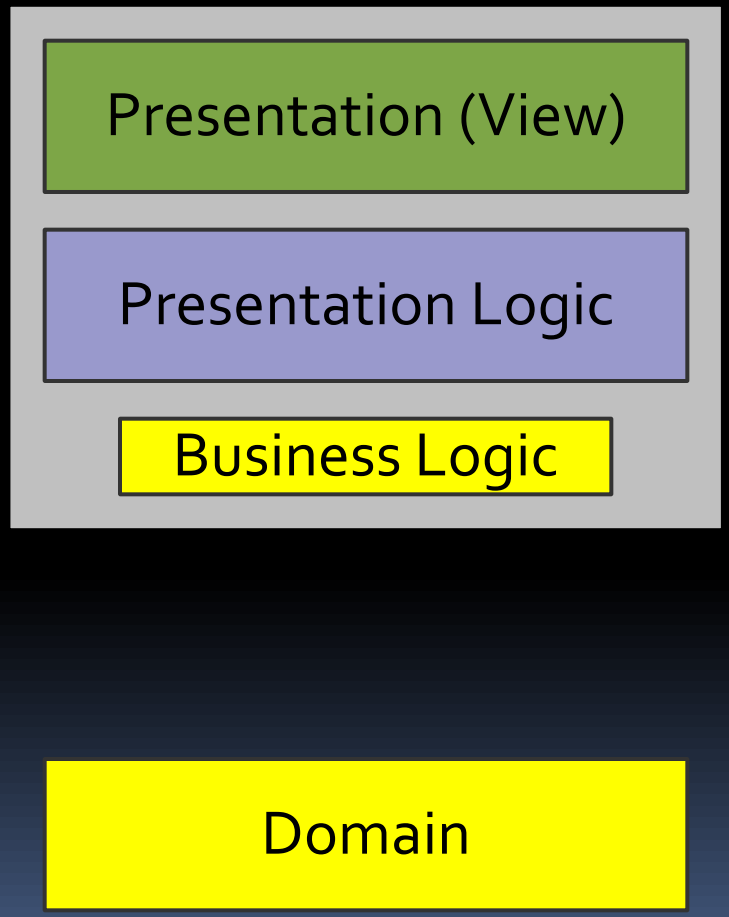
Alle Rollen vermengt



Muster: Separated Presentation



Fachlogik in der Präsentation

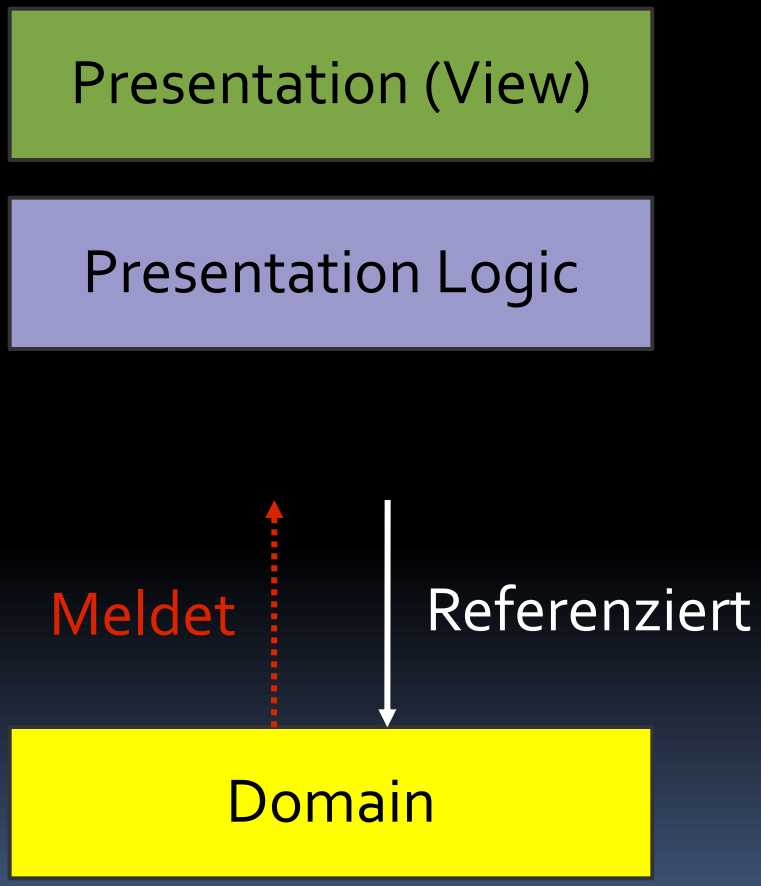


Entkopple Fachdaten und Präsentation

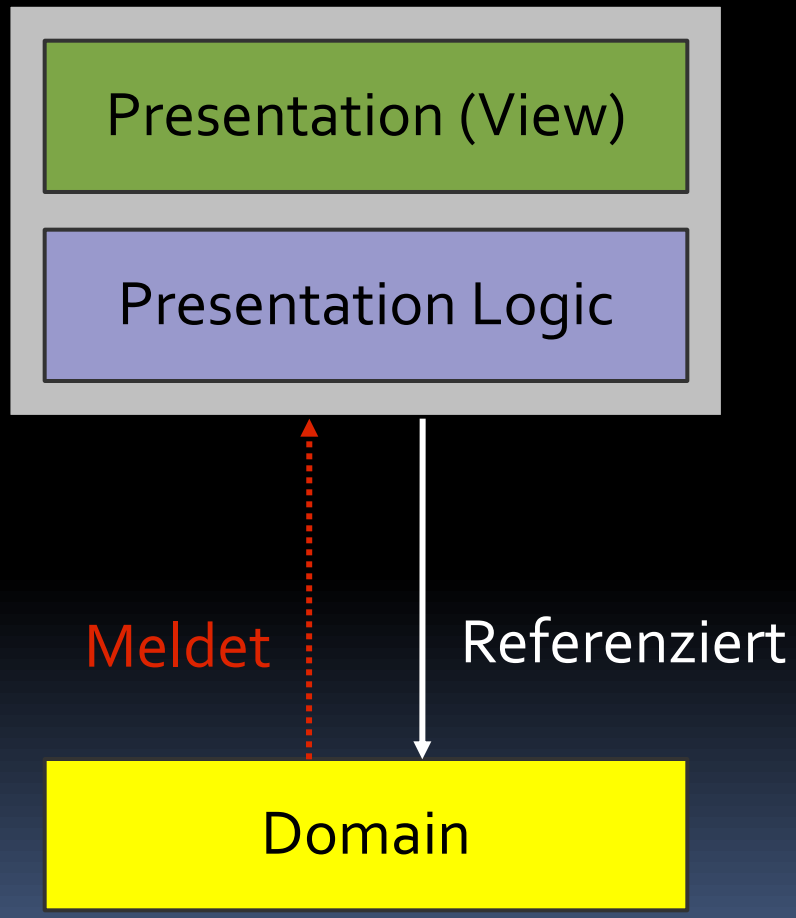
- Fachdaten sollen die Präsentation nicht halten
- Präsentation hält und ändert Fachdaten

- Vorteile
  - verringert Komplexität
  - erleichtert mehrere Views

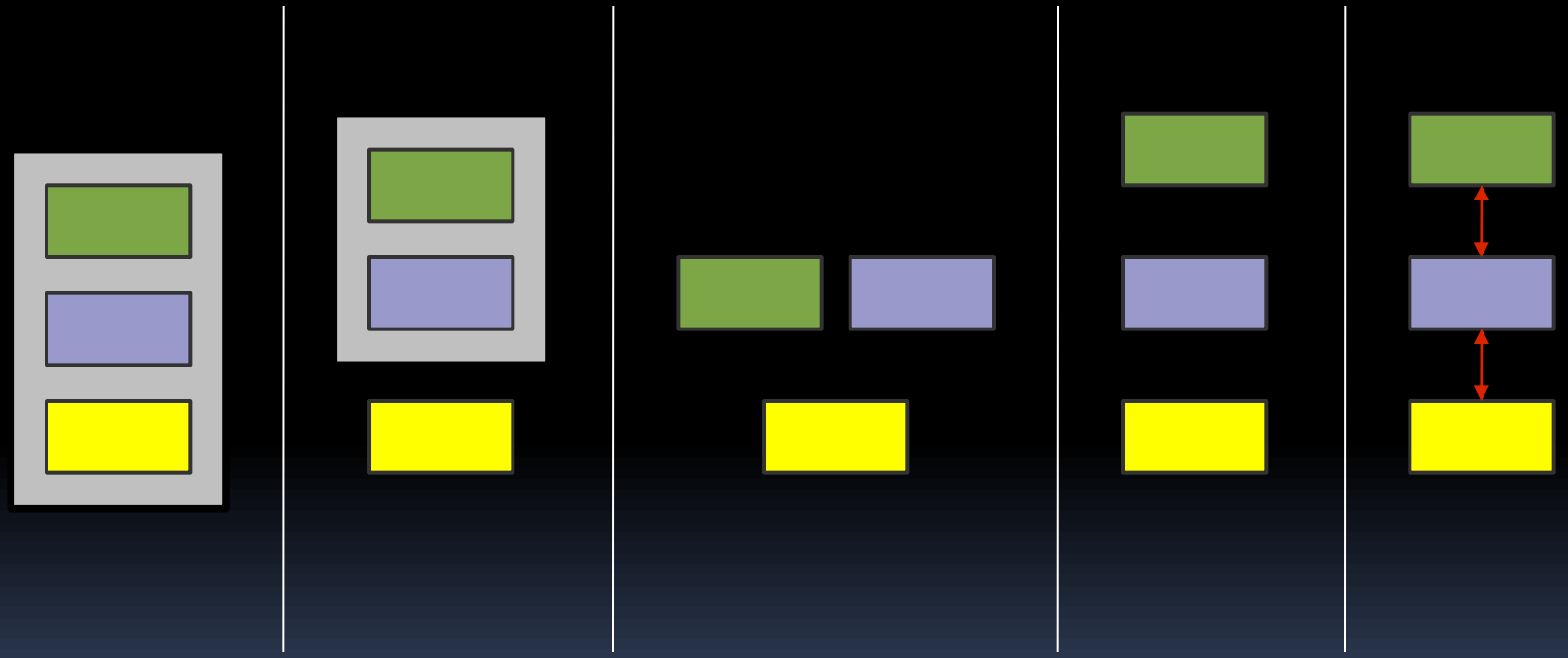
Separated Presentation mit Observer



Separated Presentation (Beispiel)



Visuelle Gliederung



Gliederung

Einleitung

Autonomous View

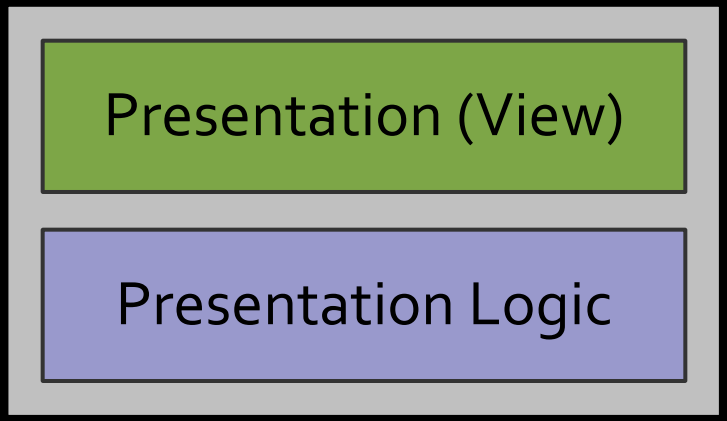
Model View Controller

Model View Presenter

Presentation Model

Datenbindung

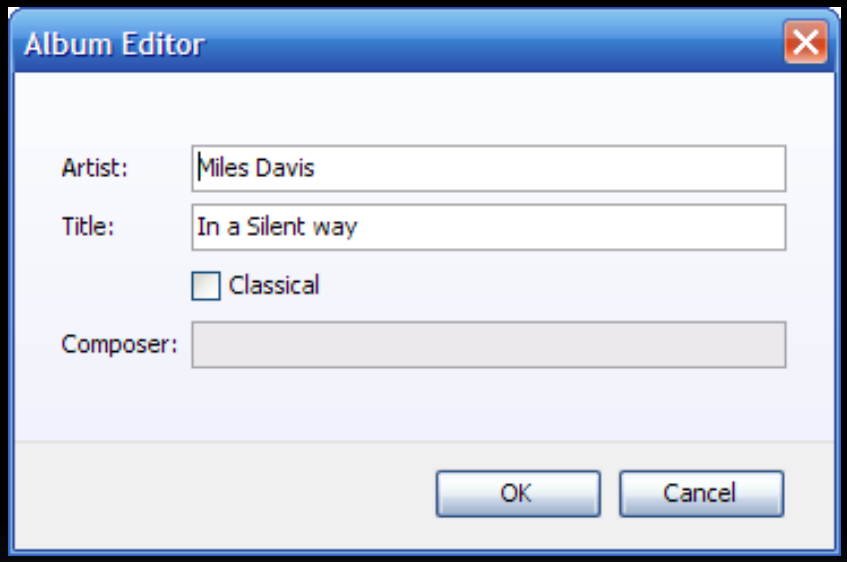
Muster: Autonomous View

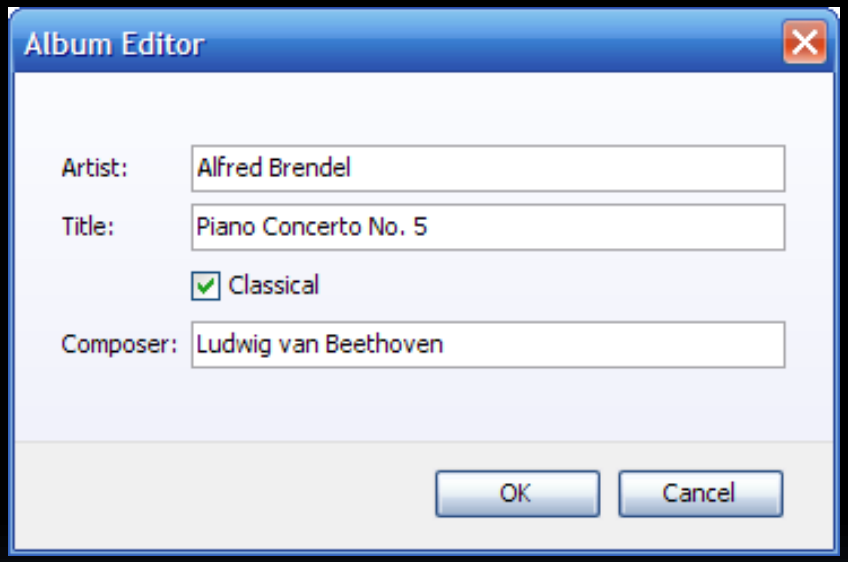


Autonomous View

- Häufig eine Klasse pro Fenster/Screen
- Häufig Unterklassen von JDialog, JFrame, JPanel
- Enthält:
  - Felder für UI-Komponenten
  - Komponenten-Initialisierung
  - Panel-Bau und Layout
  - Modell-Initialisierung
  - Präsentationslogik: Listener, Operationen

Beispiel-GUI





Composer ist enabled, wenn Classical selected ist

Autonomous View-Beispiel I

```
public class AlbumDialog extends JDialog {

    private final Album album;

    private JTextField artistField;
    ...

    public AlbumDialog(Album album) { ... }

    private void initComponents() { ... }

    private void initPresentationLogic() { ... }

    private JComponent buildContent() { ... }
```

Autonomous View-Beispiel II

```
class ClassicalChangeHandler
                          implements ChangeListener {

   public void stateChanged(ChangeEvent e) {
      // Prüfe den classical-Zustand.
     boolean classical = classicalBox.isSelected();

     // Aktualisiere das composer-Feld-enablement.
     composerField.setEnabled(classical);
   }

}
```

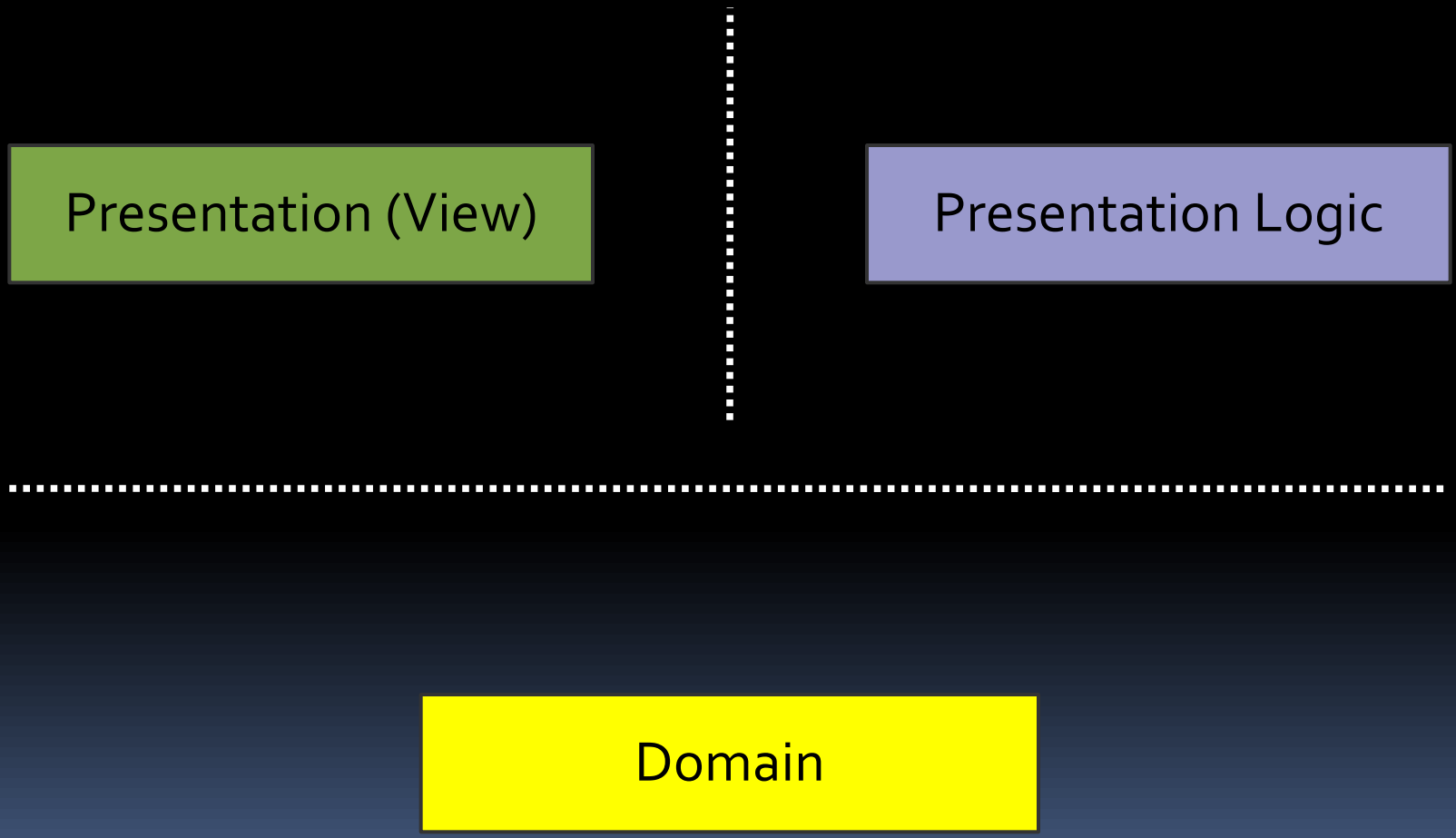
Autonomous View: Tipps

- Baue Dialoge, Frames, Panels
- Erweitere JDialog, JFrame, JPanel wenn nötig. Erweiterst oder nutzt du HashMap?

Autonomous View

- Üblich und brauchbar
- Hat Nachteile:
  - Schwer logisch zu testen
  - Schwer zu überblicken, pflegen, debuggen, wenn der View oder die Logik komplex ist
- Erwäge, die Logik vom View zu trennen

Präsentationslogik abgetrennt



Separated Logic: Vorteile I

- Logische Tests der Präsentationslogik
- Team-Synchronisation
- Kleinere Teile, leichter zu überblicken
- Erleichtert "verbotene" Bereiche
  - für Entwickler
  - vor einem neuen Release
    - Layout-Änderungen erlaubt
    - Design fertig, Logik-Korrekturen erlaubt

Separated Logic: Vorteile II

- Dünne, "dumme" GUI
  - einfacher zu bauen, verstehen, warten
  - kann syntaktischen Mustern folgen
  - Mehr Team-Mitglieder können damit arbeiten
- Logik kann Präsentationsdetails ignorieren, etwa Komponententypen (JTable vs. JList)
- Logik kann geteilt werden zwischen Views

Separated Logic: Nachteile

- Extra-Mechanismus für die Trennung
- Mehraufwand um mehrere Quellen zu lesen und zu bearbeiten

Logik vom View trennen

- Kann vereinfachen oder verkomplizieren
- Trennkosten hängen vom Muster ab
- Meinung: meistens lohnt es, zu trennen

Meine Empfehlung für Projekte:

- Nutze Autonomous View für Nachrichtendialoge
- Trenne alle anderen Fälle

Gliederung

Einleitung

Autonomous View

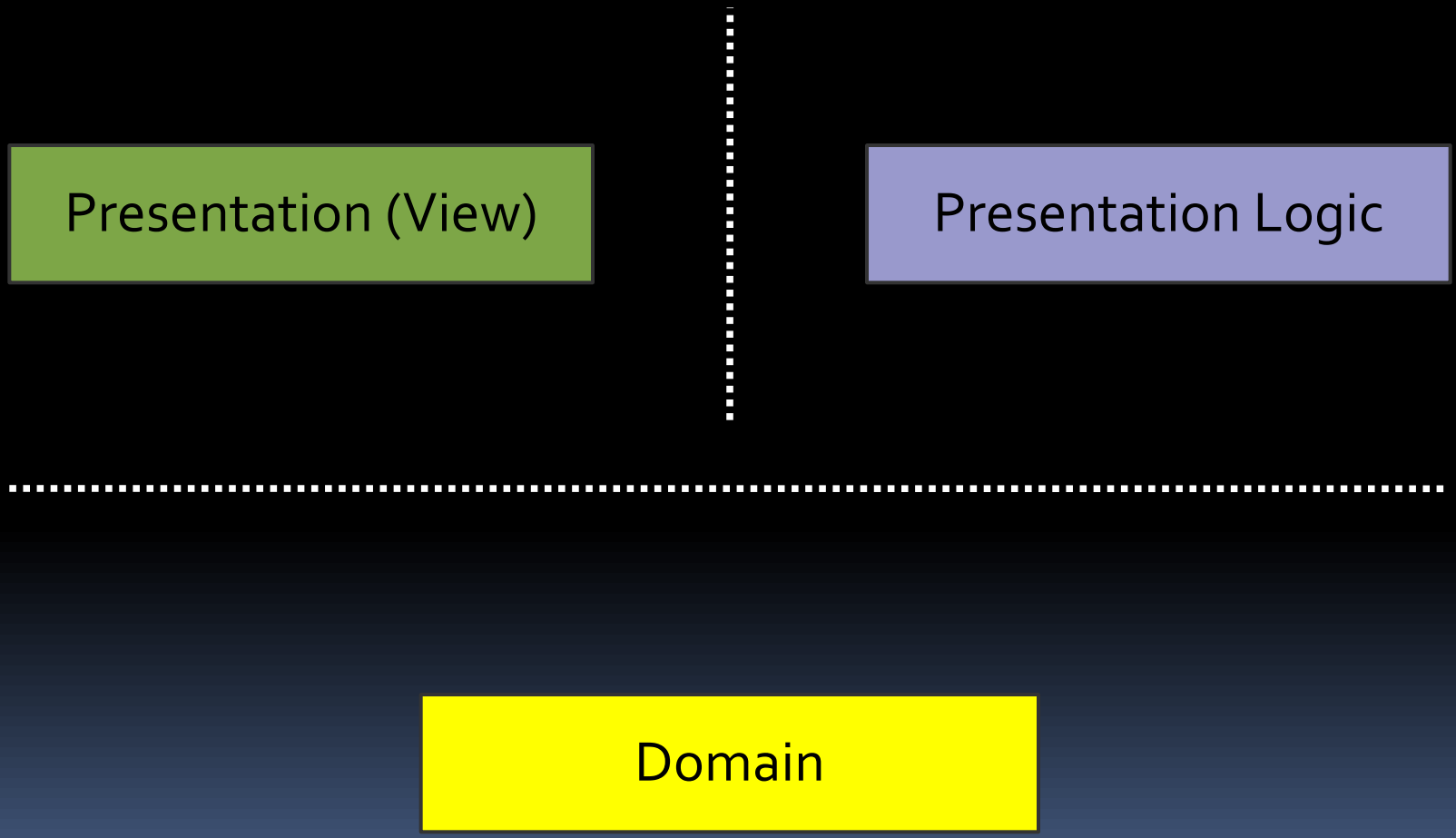
Model View Controller

Model View Presenter

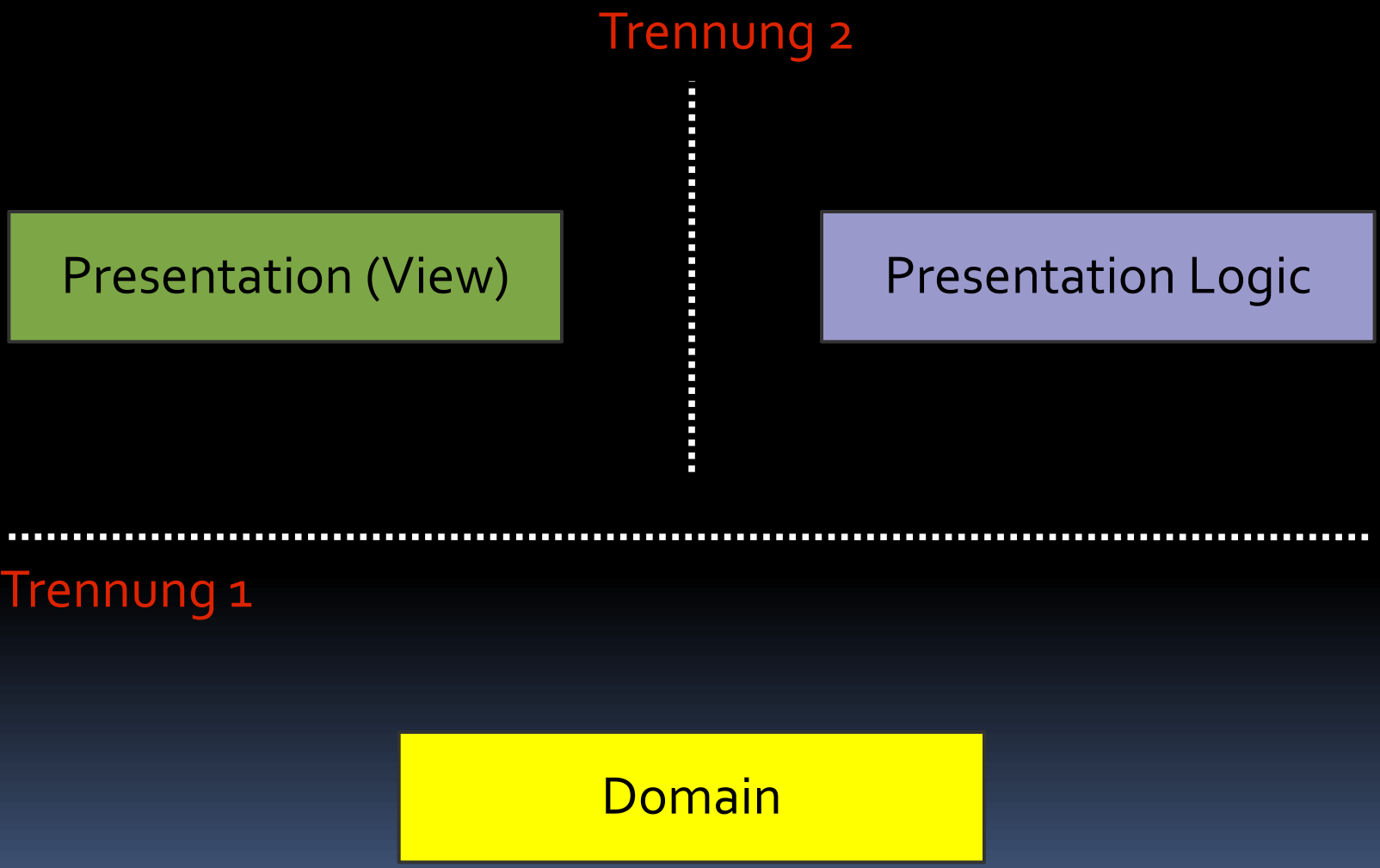
Presentation Model

Datenbindung

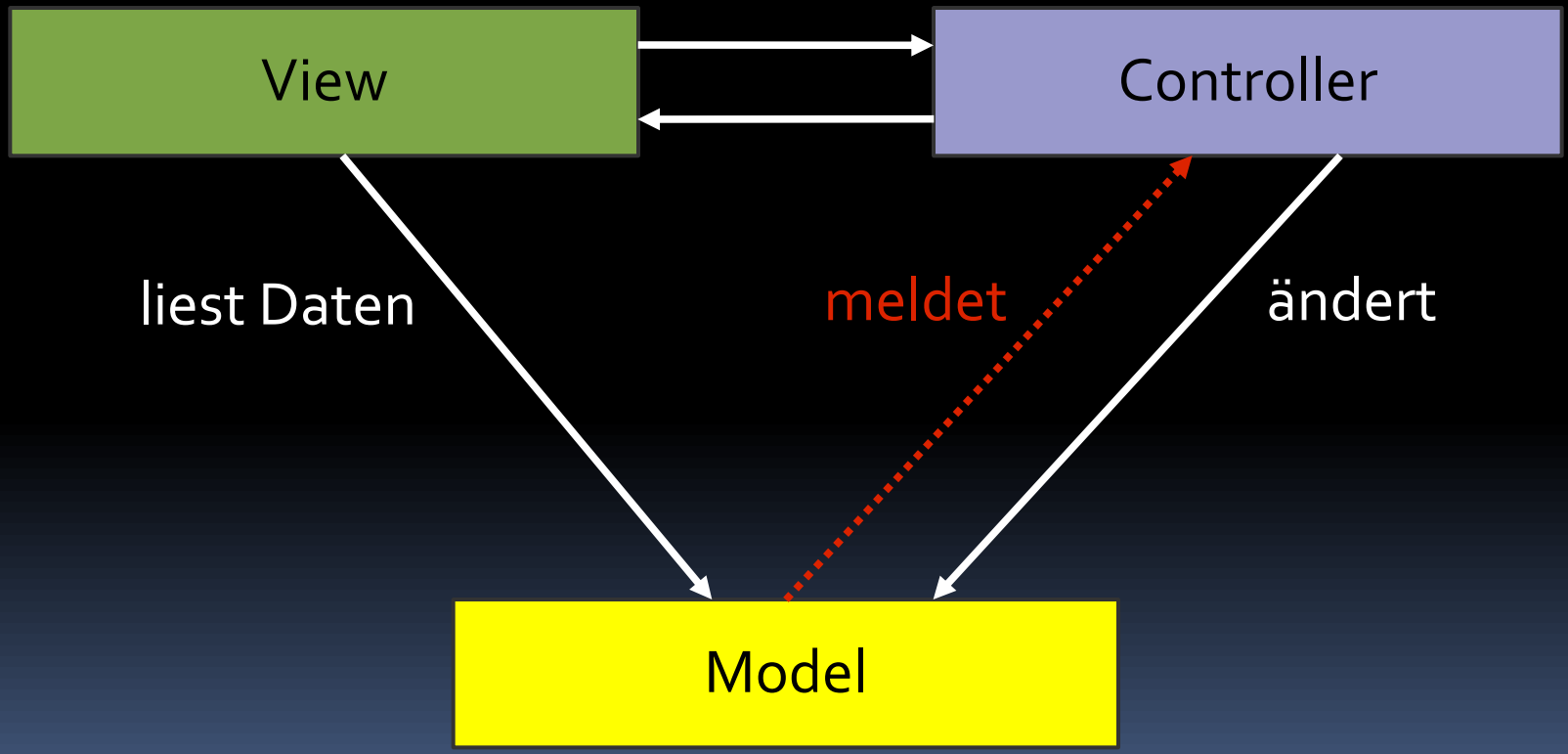
Präsentationslogik abgetrennt



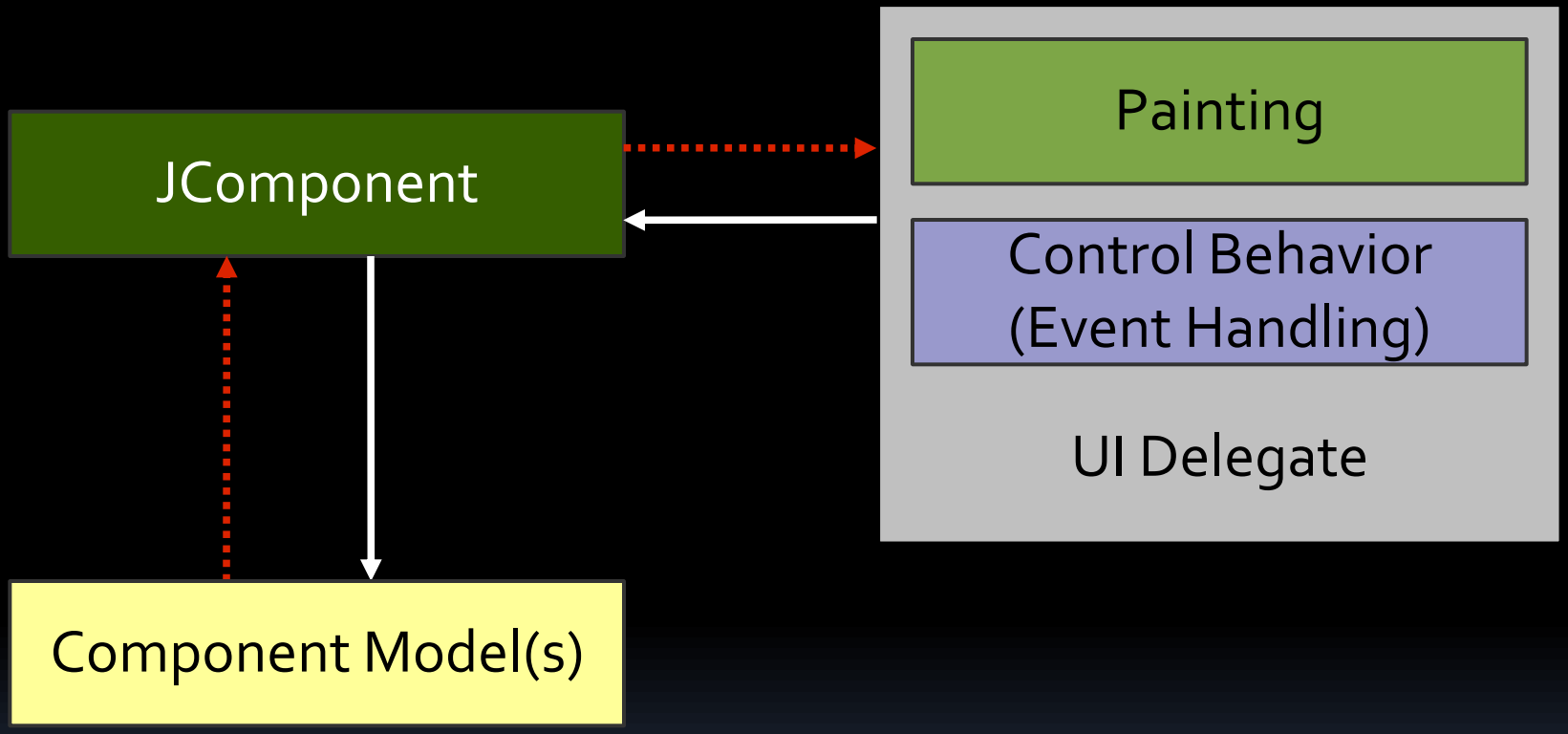
Motivation für MVC



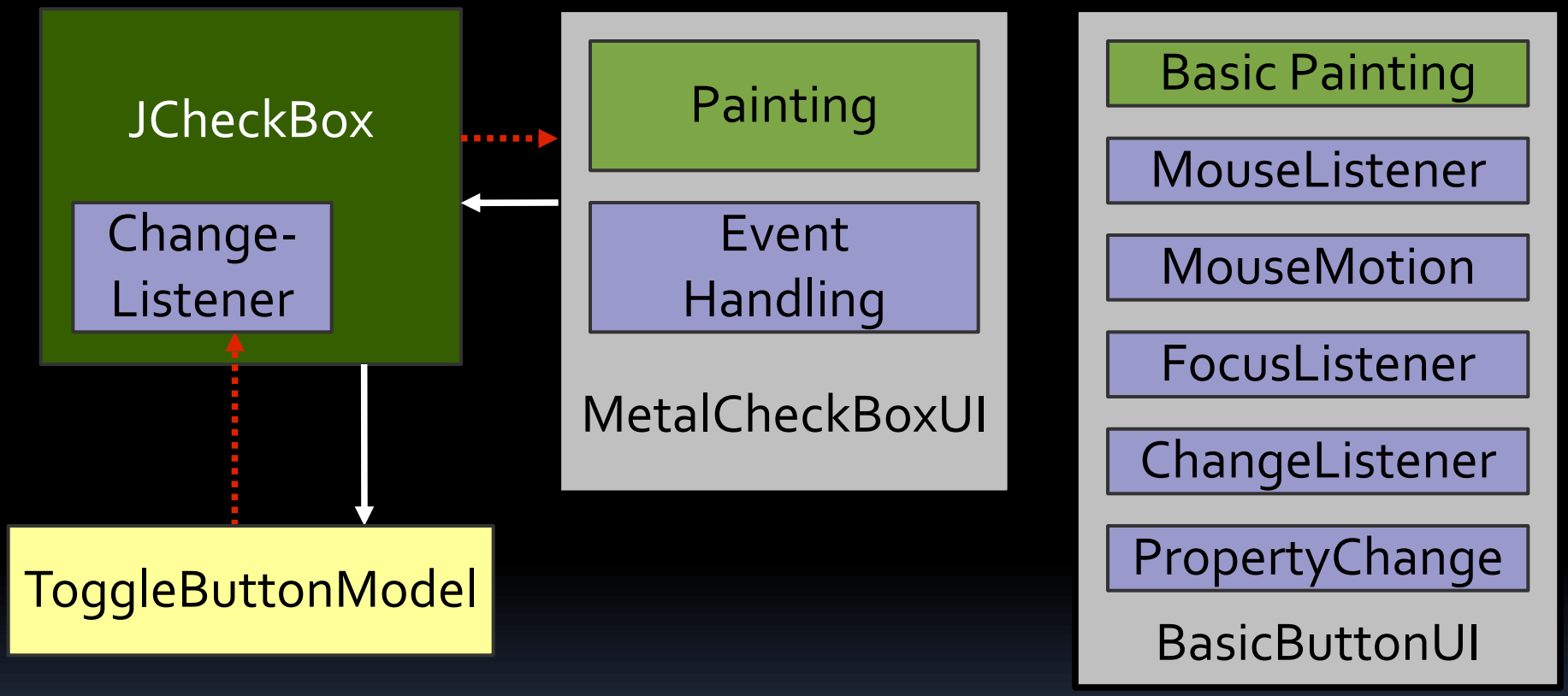
Muster: Model-View-Controller



Swing: M-JComponent-(VC)



JCheckBox


JCheckBox
Change-
Listener
Painting
Event
Handling
MetalCheckBoxUI
ToggleButtonModel
Basic Painting
MouseListener
MouseMotion
FocusListener
ChangeListener
PropertyChange
BasicButtonUI

MVC vs. Swing

- MVC trennt View und Controller
- Swing vereint View mit Controller
- UI-Delegates malen und behandeln Events

- MVC geht für Komponenten und Anwendungen
- Swing nutzt MVC nicht für Komponenten

Gliederung

Einleitung

Autonomous View

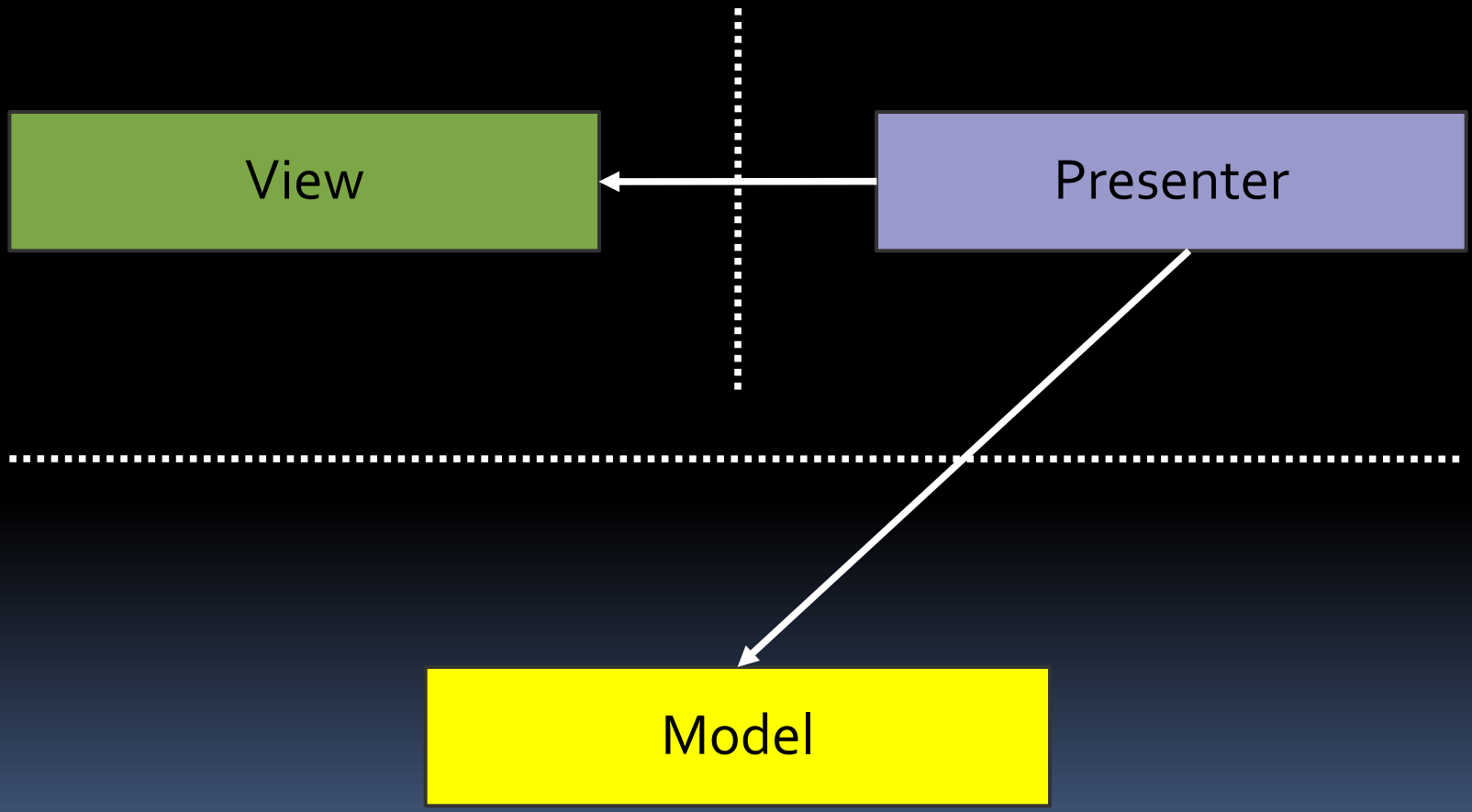
Model View Controller

Model View Presenter

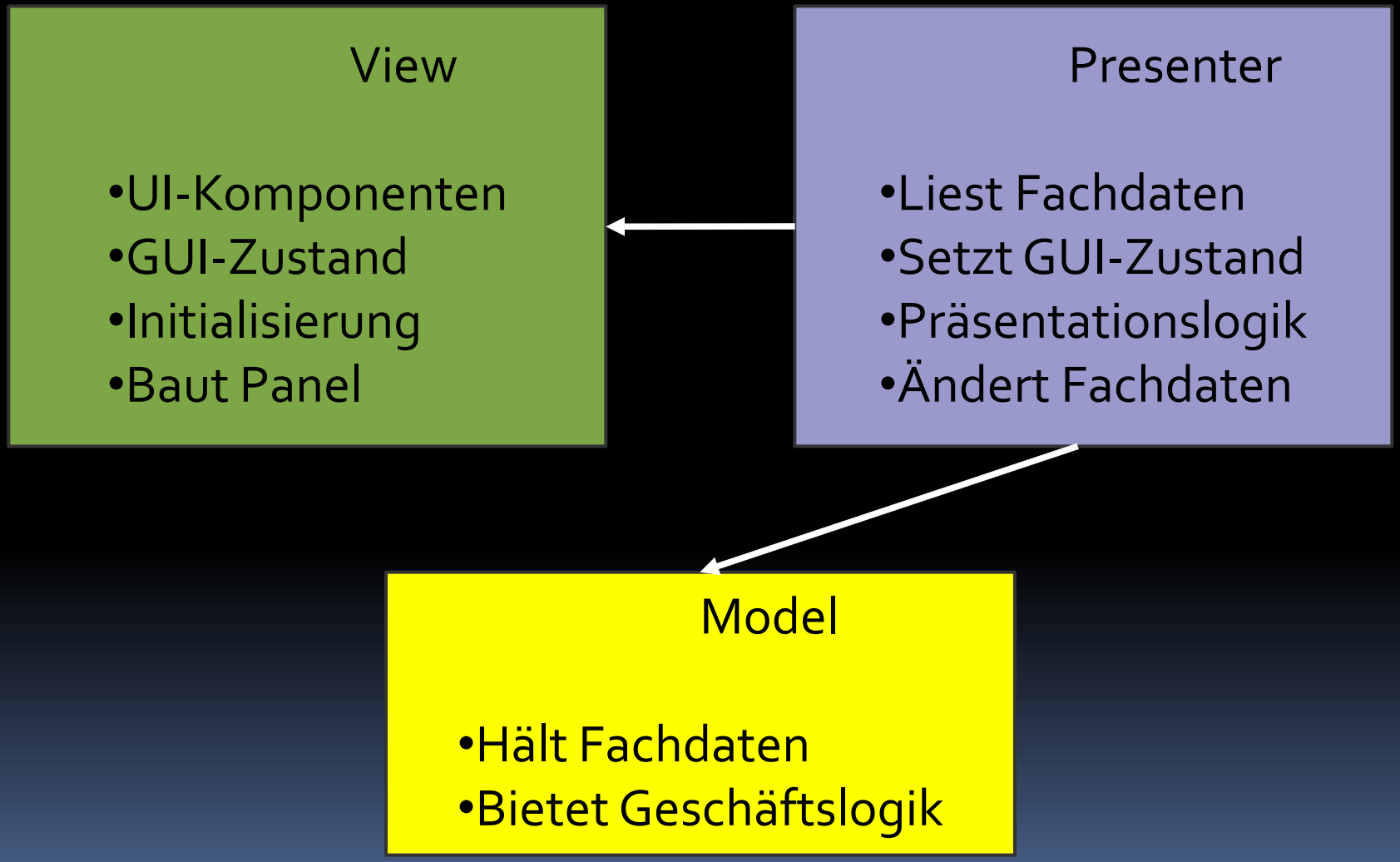
Presentation Model

Datenbindung

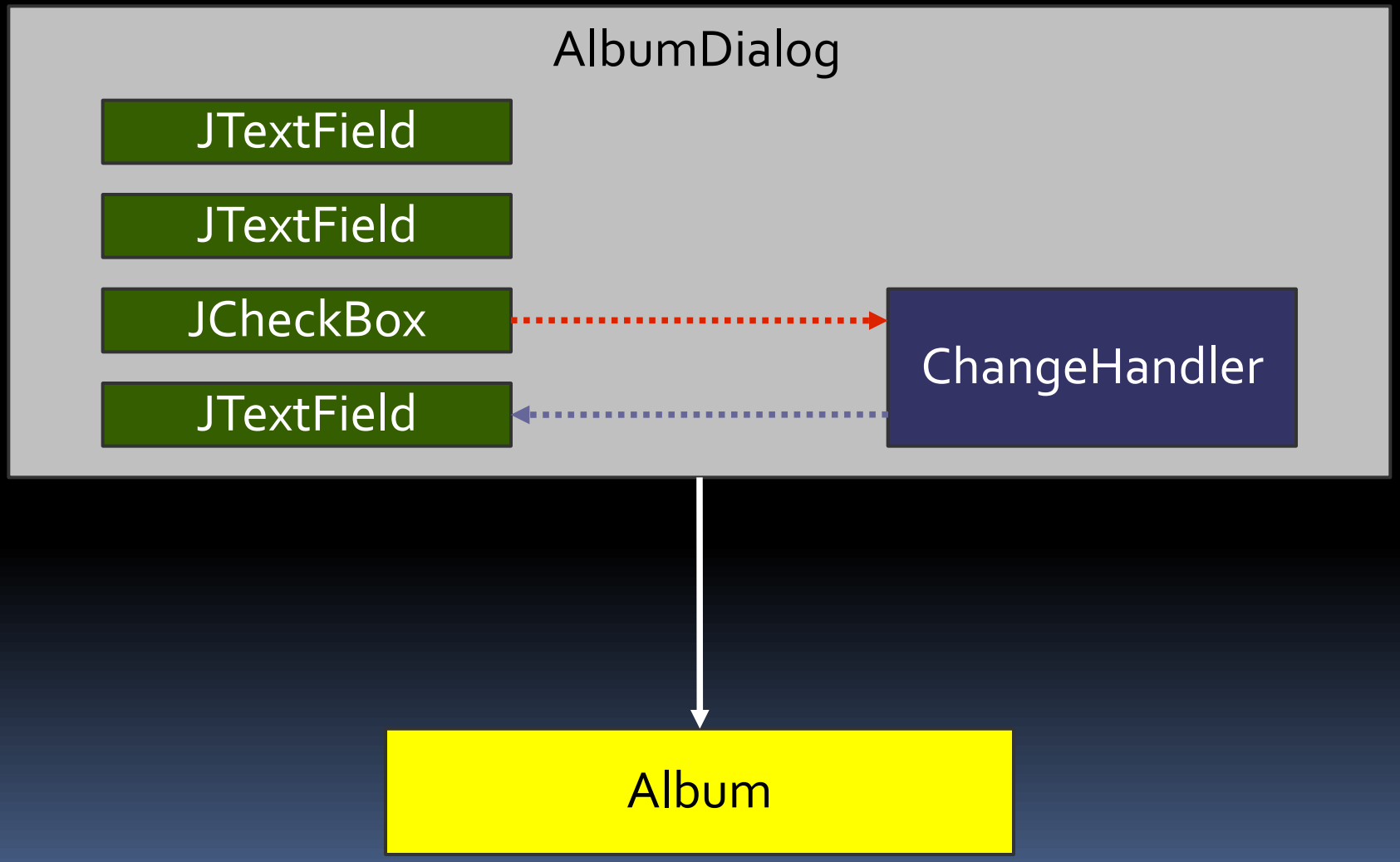
Muster: Model-View-Presenter



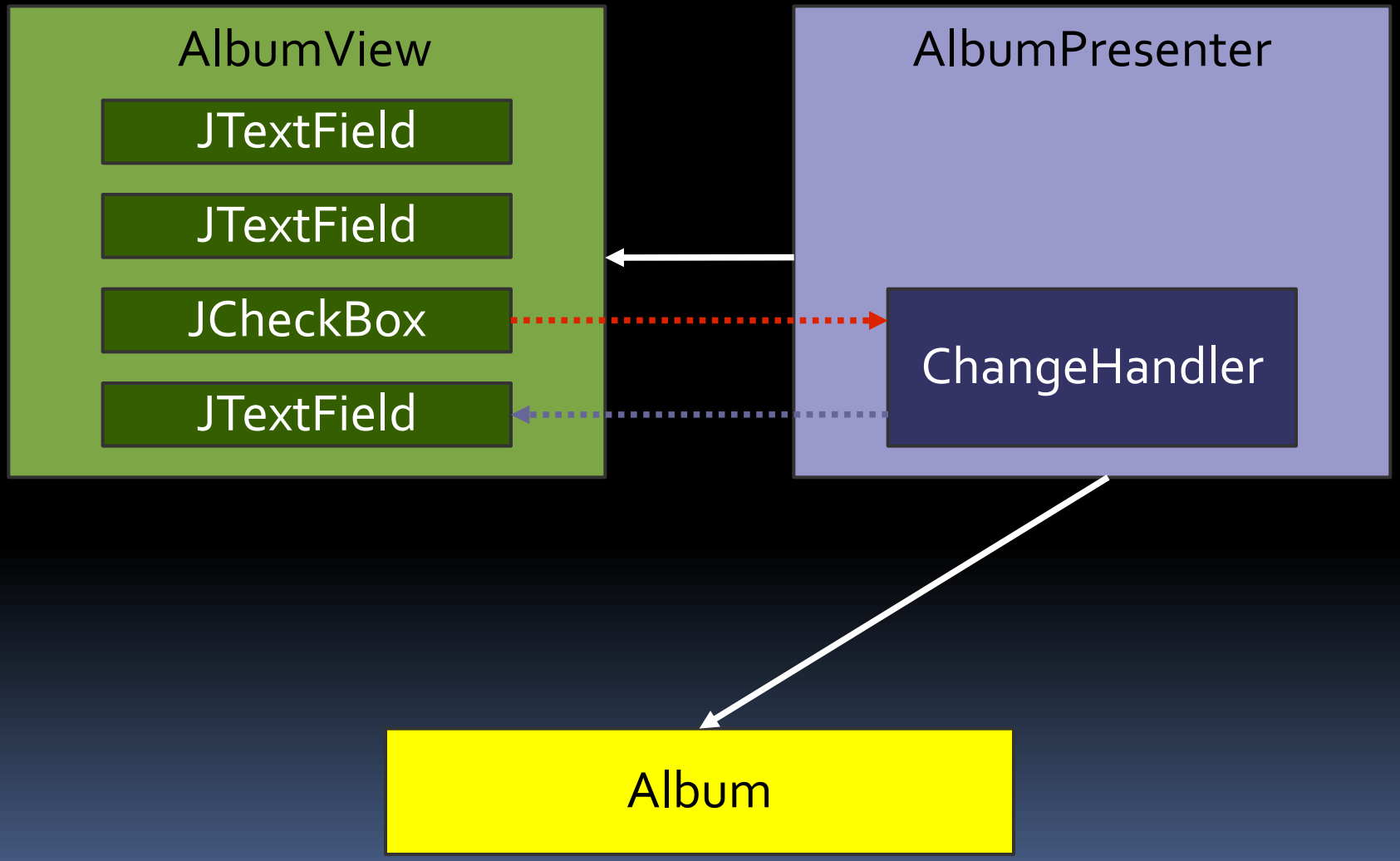
Model-View-Presenter (MVP)



Album mit Autonomous View



Album mit MVP



Von Autonomous View …

```
public class AlbumDialog extends JDialog {
private JTextField artistField;
public AlbumDialog(Album album) { ... }
private void initComponents() { ... }
private JComponent buildContent() { ... }

private final Album album;
private void initPresentationLogic() { ... }
private void readGUIStateFromDomain() { ... }
private void writeGUIStateToDomain() { ... }
class ClassicalChangeHandler implements ...
class OKActionHandler implements ...
}
```

… zu Model-View-Presenter

```
class AlbumView extends JDialog {
JTextField artistField;
public AlbumView() { ... }
private void initComponents() { ... }
private JComponent buildContent() { ... }
}
public class AlbumPresenter {
private final AlbumView view;
private Album album;
private void initPresentationLogic() { ... }
private void readGUIStateFromDomain() { ... }
private void writeGUIStateToDomain() { ... }
class ClassicalChangeHandler implements ...
class OKActionHandler implements ...
}
```

… zu Model-View-Presenter

```
class AlbumView extends JDialog {
JTextField artistField;
public AlbumView() { ... }
private void initComponents() { ... }
private JComponent buildContent() { ... }
}
public class AlbumPresenter {
private final AlbumView view;
private Album album;
private void initPresentationLogic() { ... }
private void readGUIStateFromDomain() { ... }
private void writeGUIStateToDomain() { ... }
class ClassicalChangeHandler implements ...
class OKActionHandler implements ...
}
```

Presenter: Beispiellogik

```
class ClassicalChangeHandler
                        implements ChangeListener {

  public void stateChanged(ChangeEvent e) {
    // Prüfe den classical-Zustand des Views.
    boolean classical =
              view.classicalBox.isSelected();

    // Ändere dessen composer-Feld-Enablement.
    view.composerField.setEnabled(classical);
  }

}
```

Gliederung

Einleitung

Autonomous View

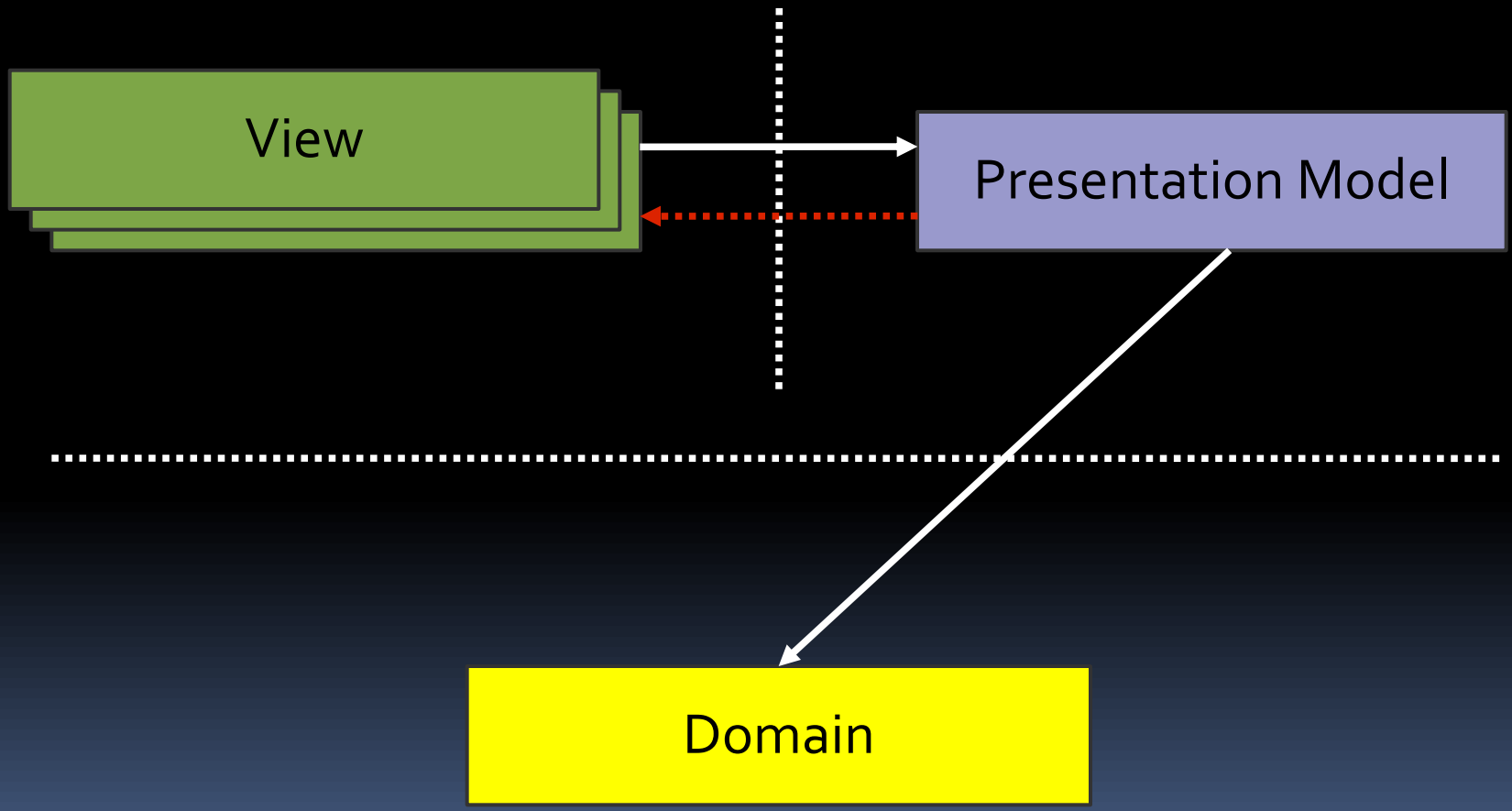
Model View Controller

Model View Presenter

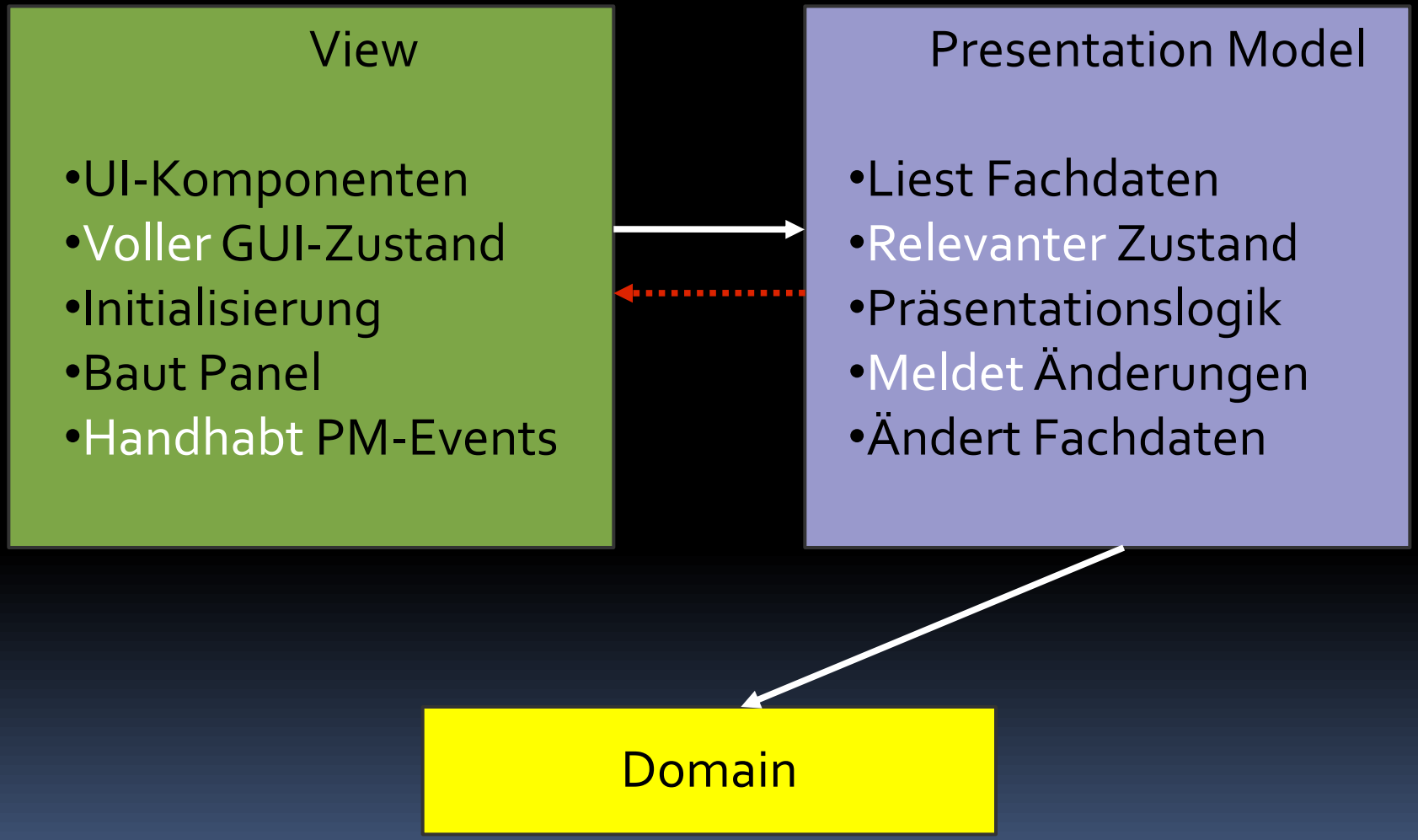
Presentation Model

Datenbindung

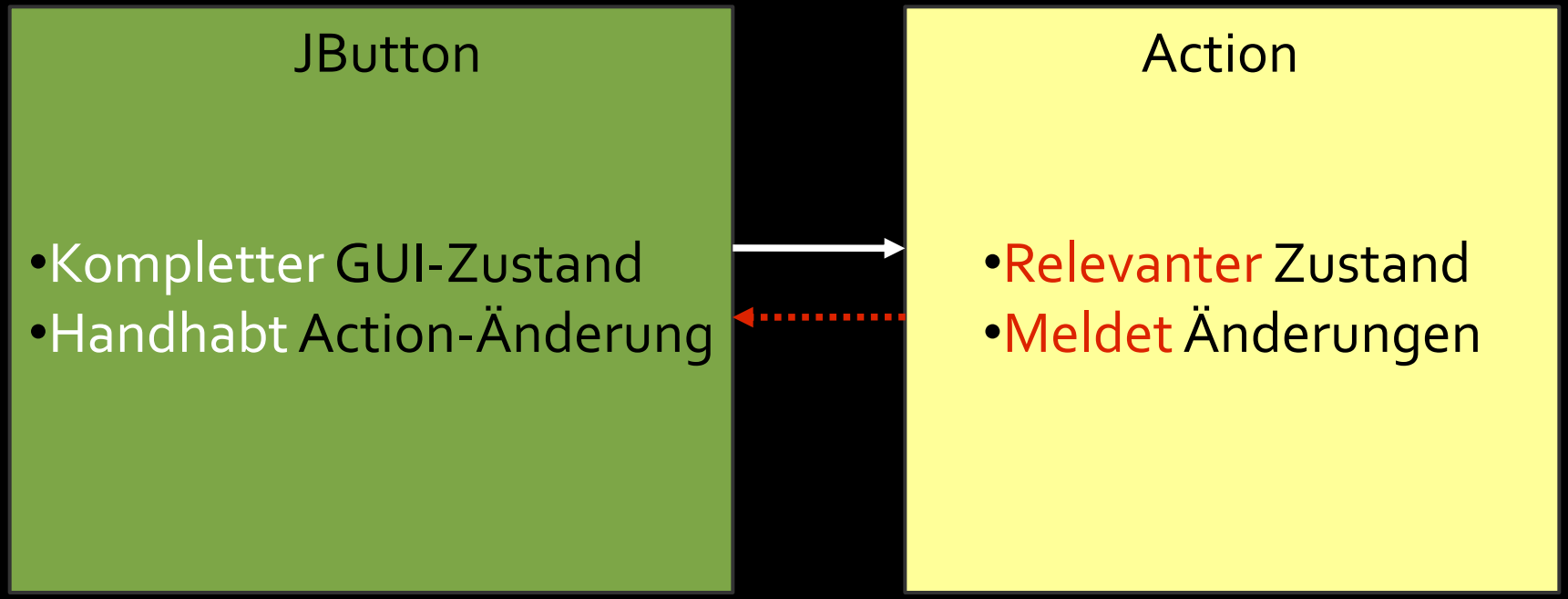
Muster: Presentation Model



Presentation Model (PM)



Erinnerung: Swing-Actions



Von Autonomous View …

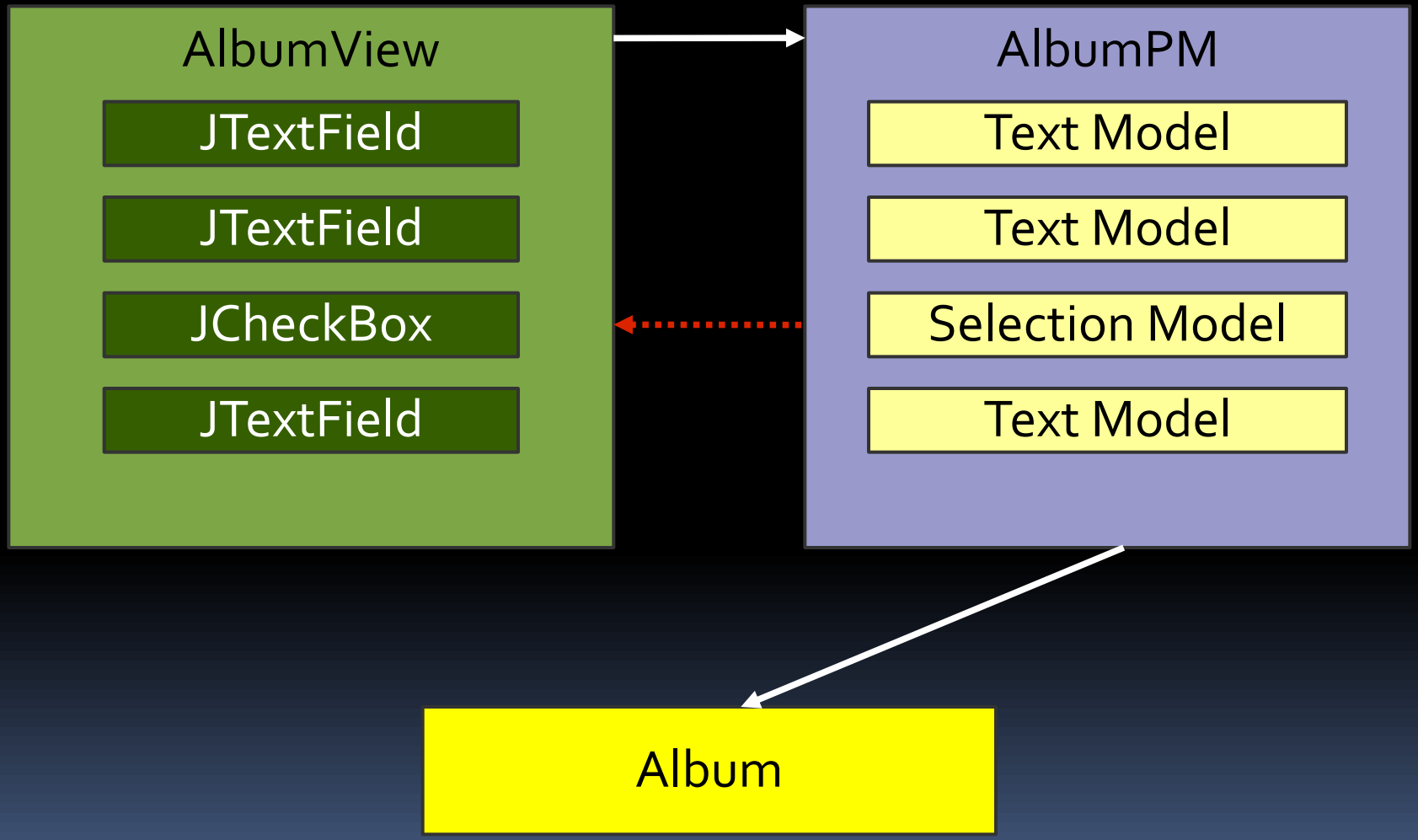
```
public class AlbumDialog extends JDialog {
private JTextField artistField;
public AlbumDialog(Album album) { ... }
private void initComponents() { ... }
private JComponent buildContent() { ... }

private final Album album;
private void initPresentationLogic() { ... }
private void readGUIStateFromDomain() { ... }
private void writeGUIStateToDomain() { ... }
class ClassicalChangeHandler implements ...
class OKActionHandler implements ...
}
```

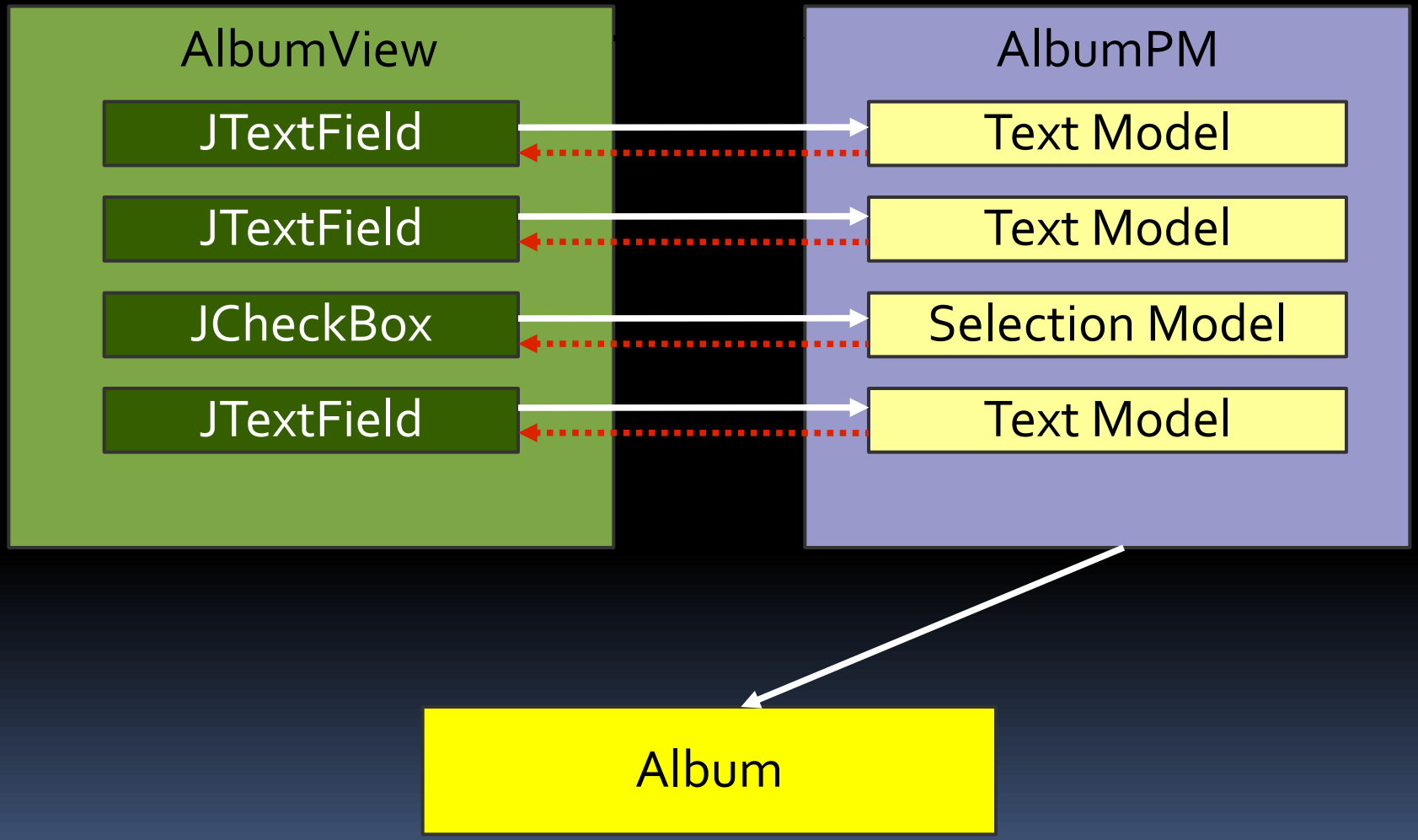
… zu Presentation Model

```
class AlbumView extends JDialog {
   private final AlbumPresentationModel model;
private JTextField artistField;
public AlbumView(AlbumPM model) { ... }
private void initComponents() { ... }
private JComponent buildContent() { ... }
}
public class AlbumPresentationModel {
private Album album;
private void initPresentationLogic() { ... }
private void readPMStateFromDomain() { ... }
private void writePMStateToDomain() { ... }
class ClassicalChangeHandler implements ...
class OKActionHandler implements ...
}
```

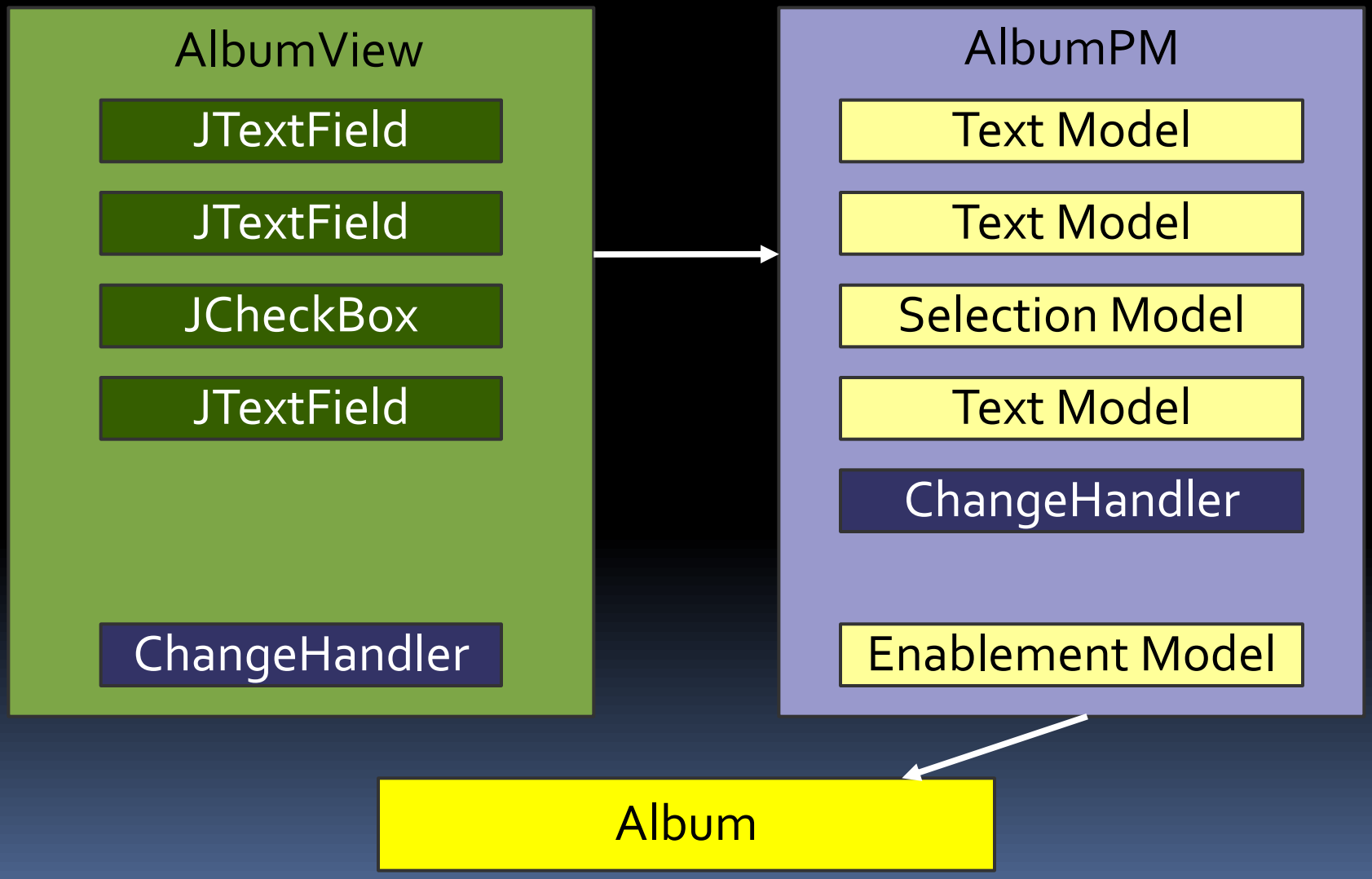
Album mit Presentation Model



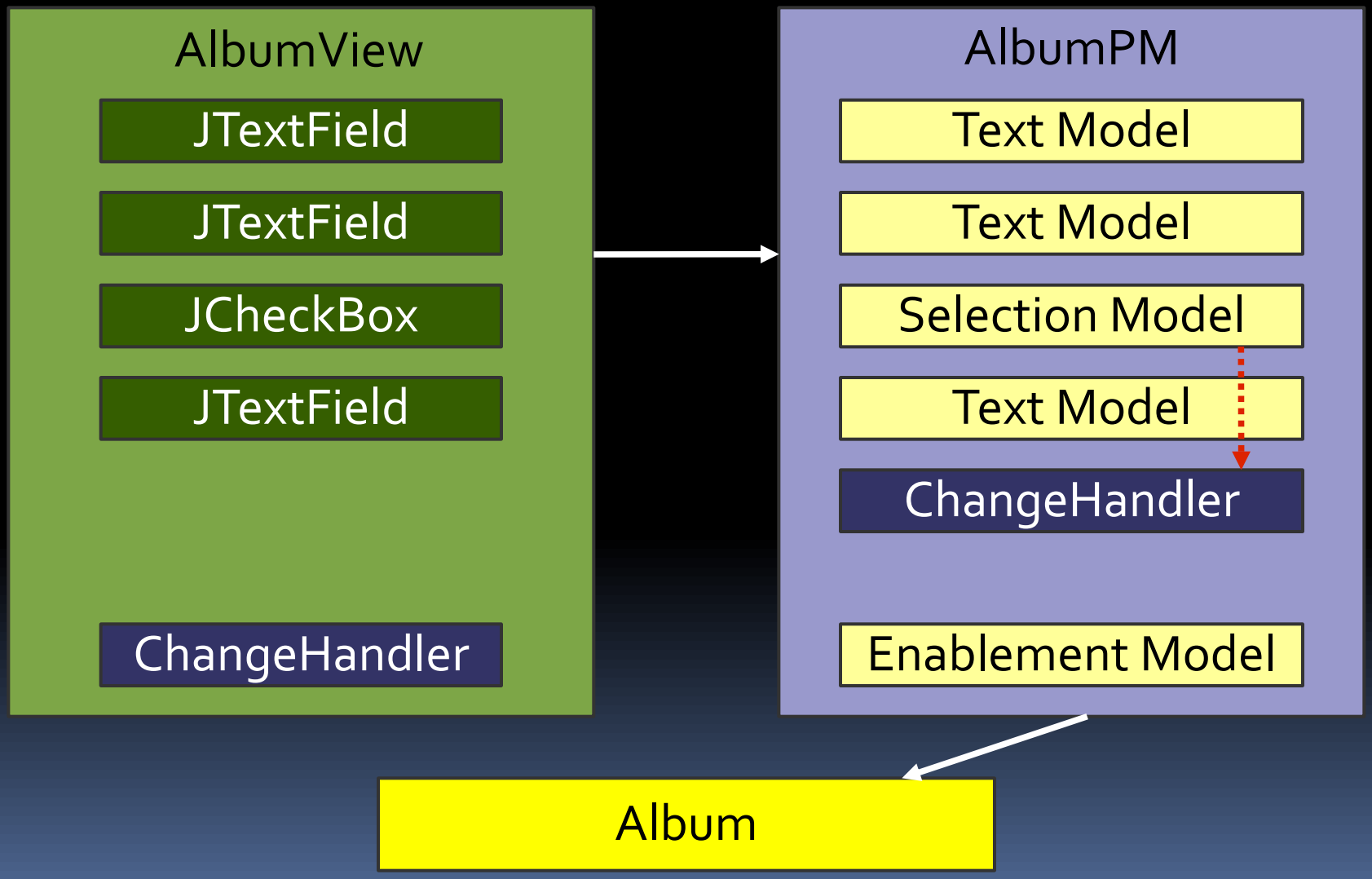
Album mit Presentation Model



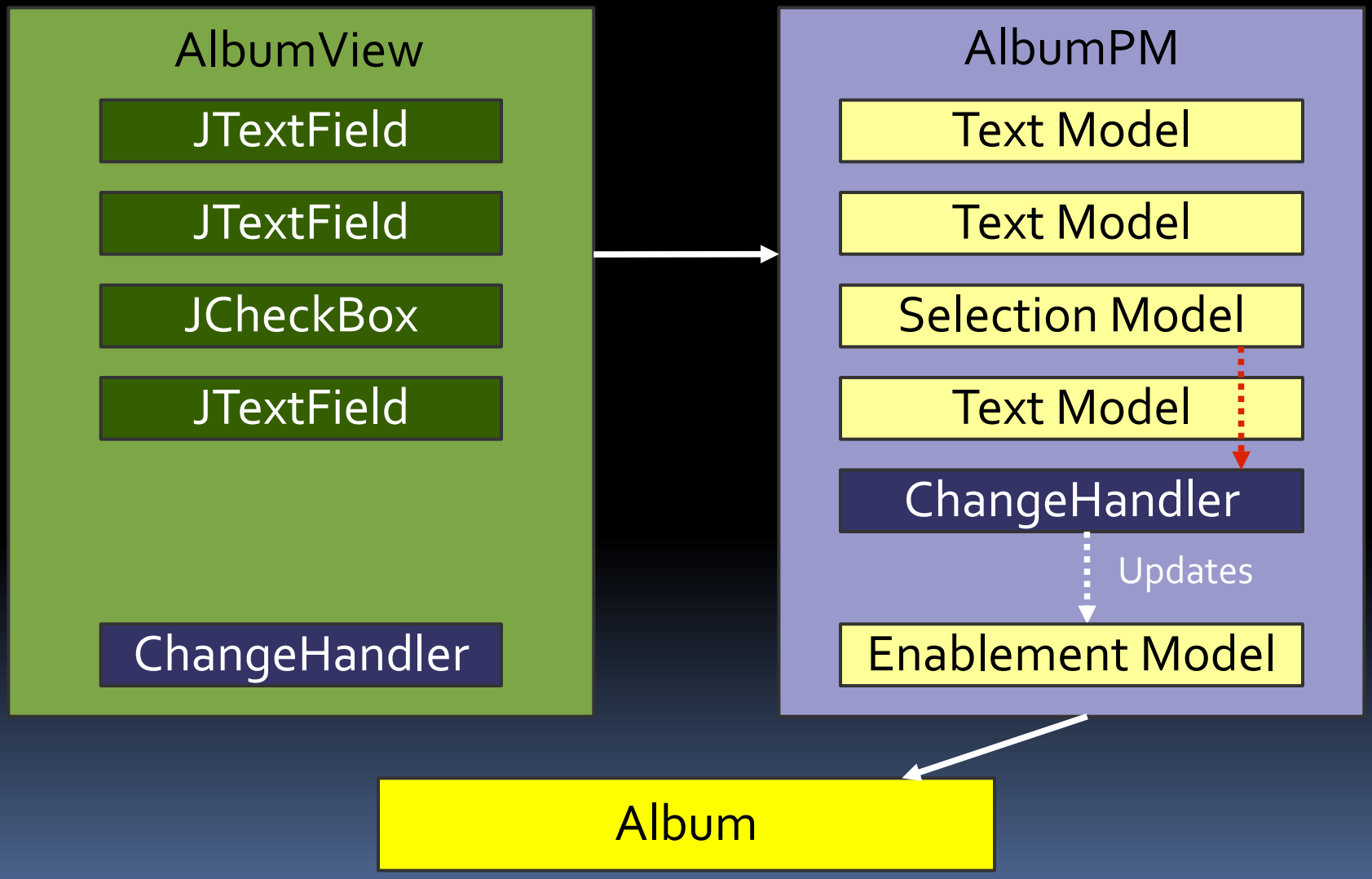
AlbumPresentationModel: Logik



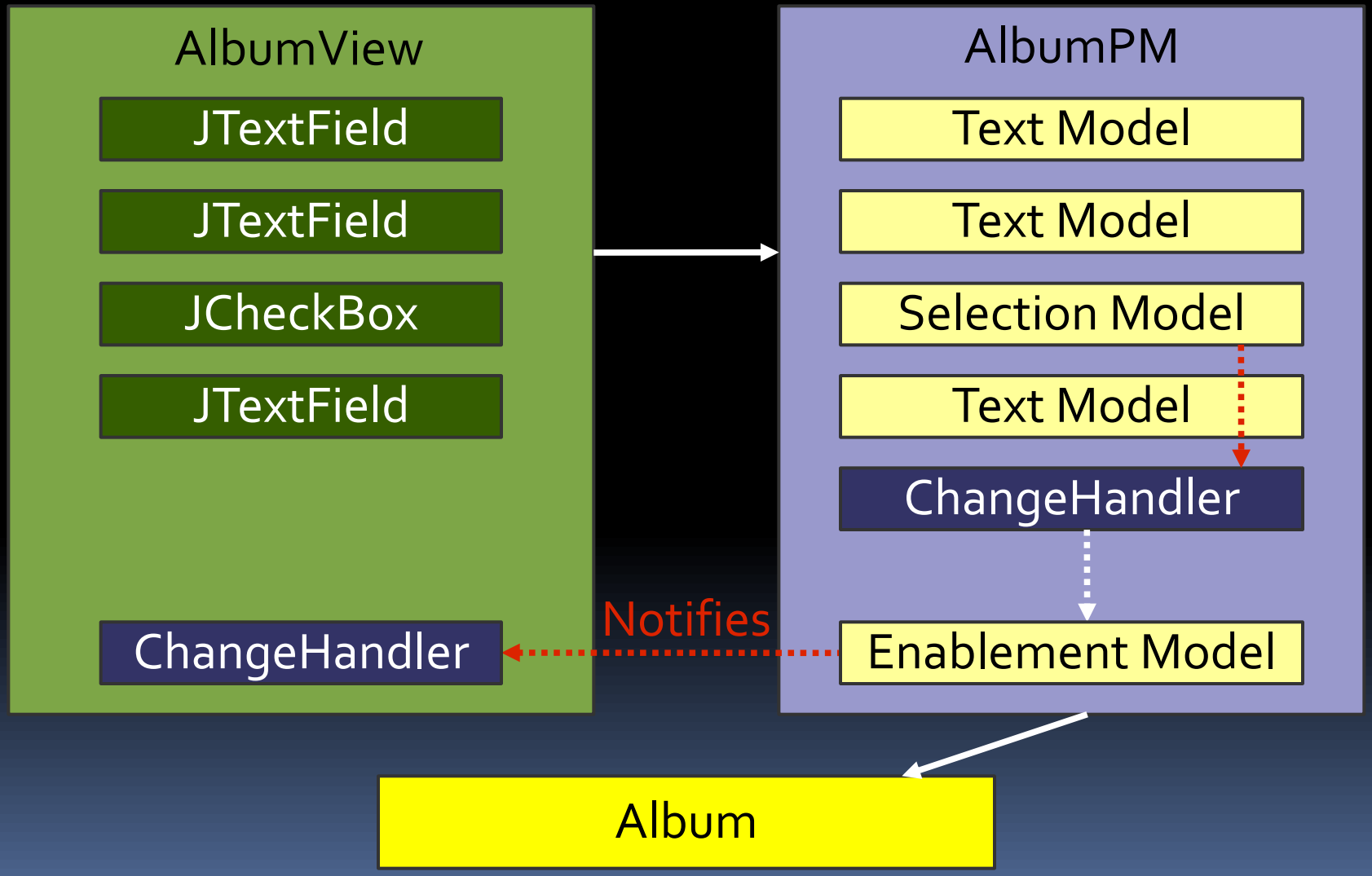
AlbumPresentationModel: Logik



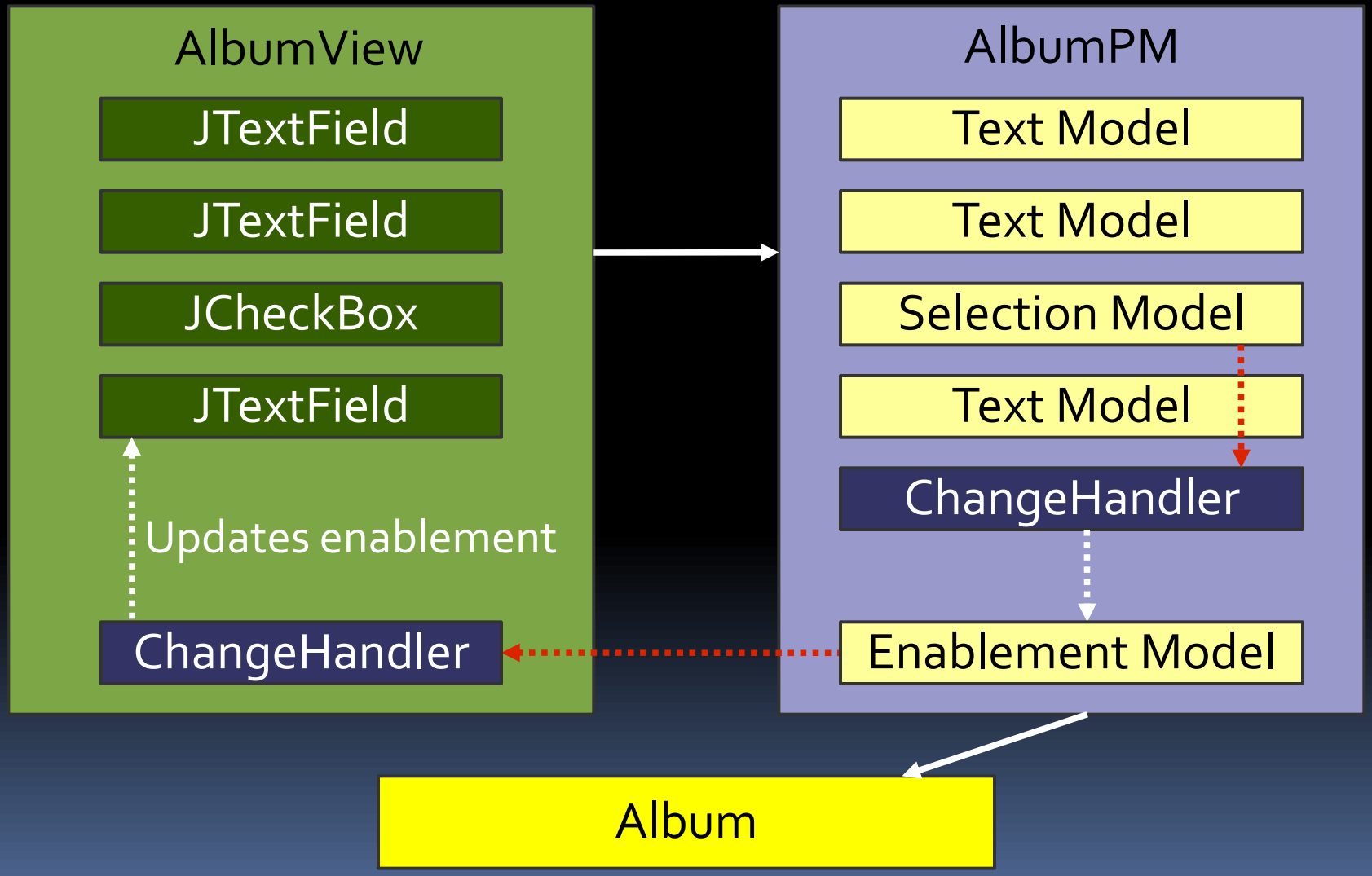
AlbumPresentationModel: Logik



AlbumPresentationModel: Logik



AlbumPresentationModel: Logik

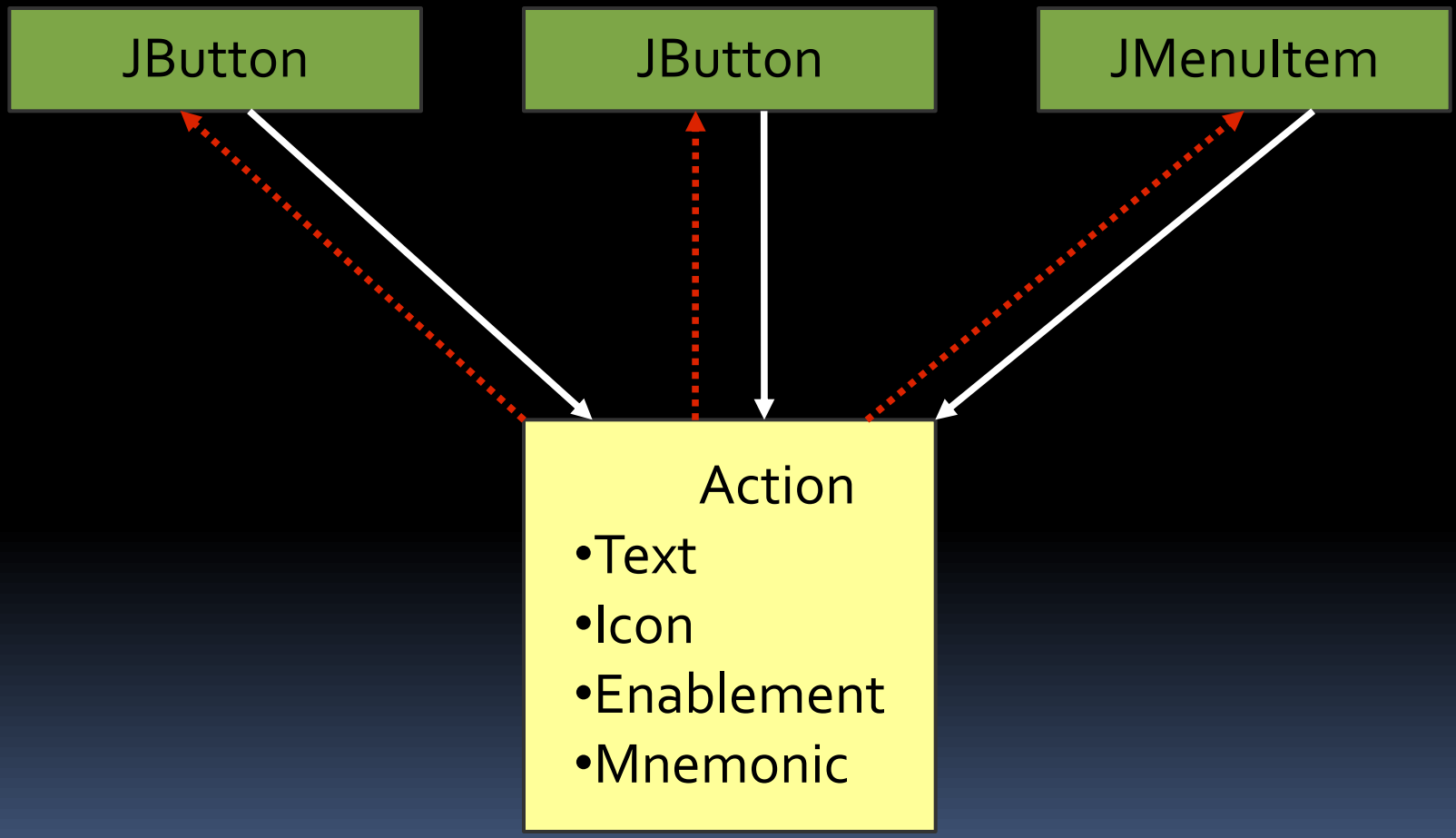


Keine Sorge: Nochmal Actions

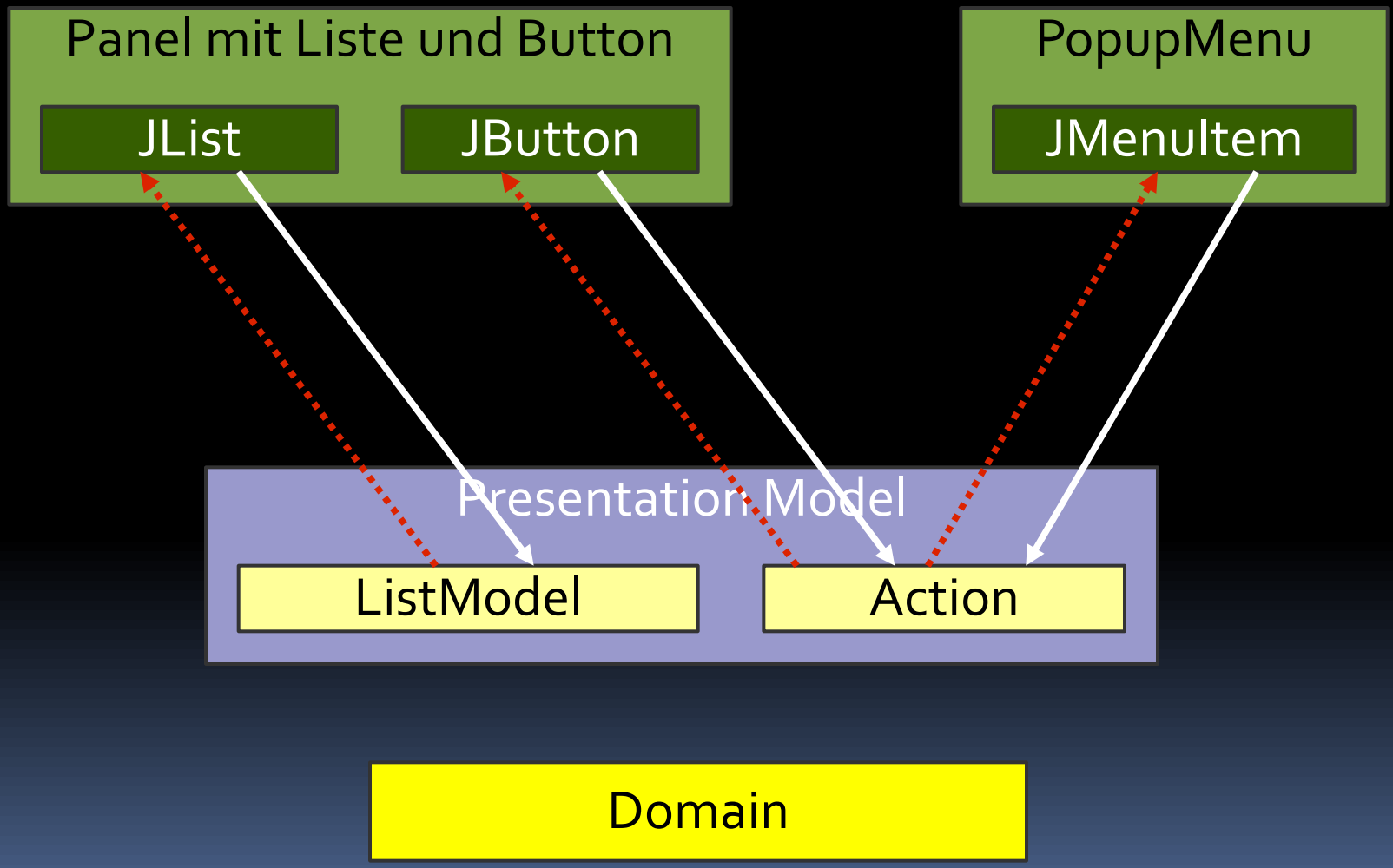
- Swing nutzt soetwas für Actions
- Actions feuern PropertyChangeEvents
- JButton beobachtet seine Action und aktualisiert seinen Zustand

- Swing synchronisiert Action- mit GUI-Zustand
- Wir schreiben nur:
  `new JButton(anAction)`

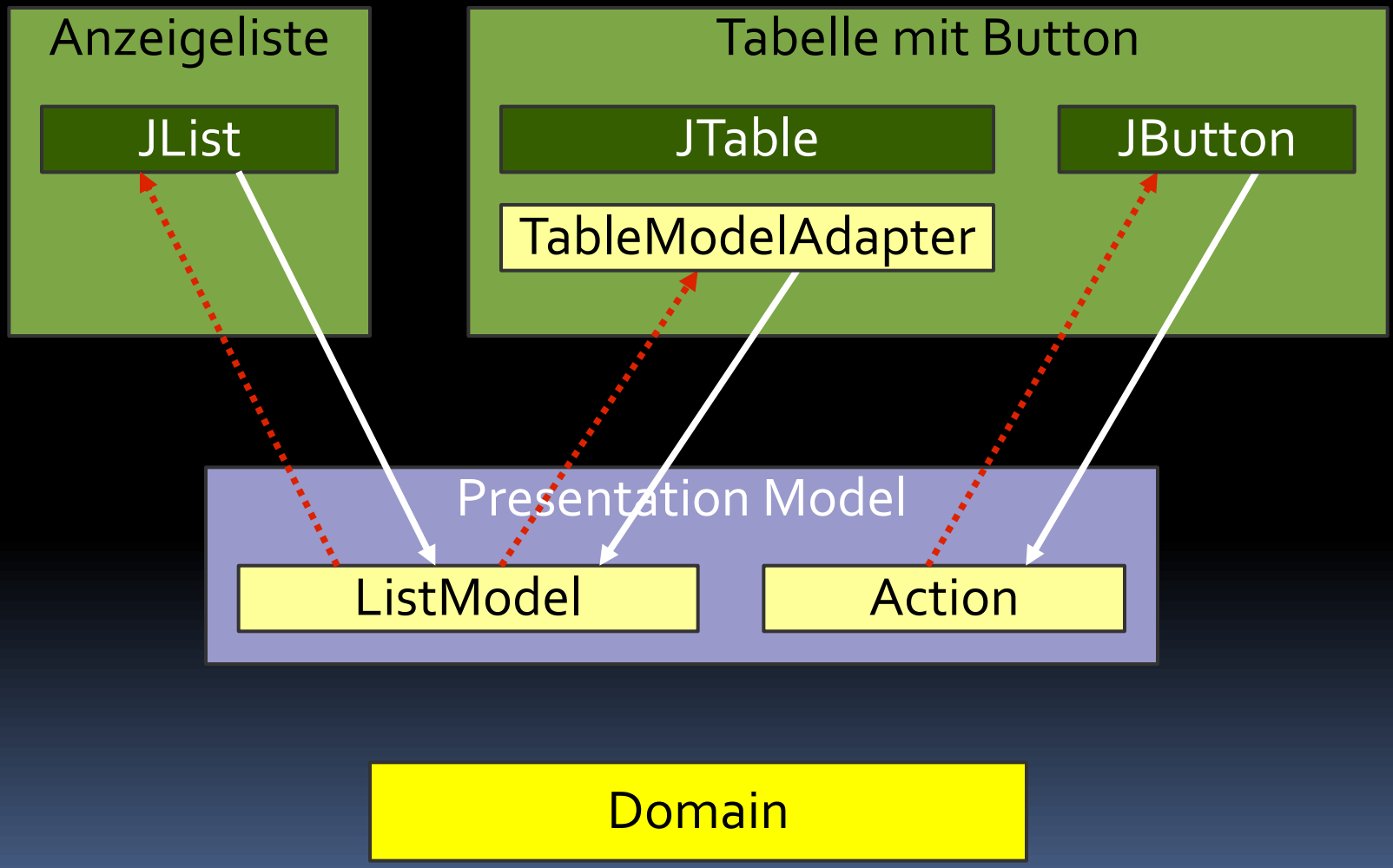
Action mit mehreren Views



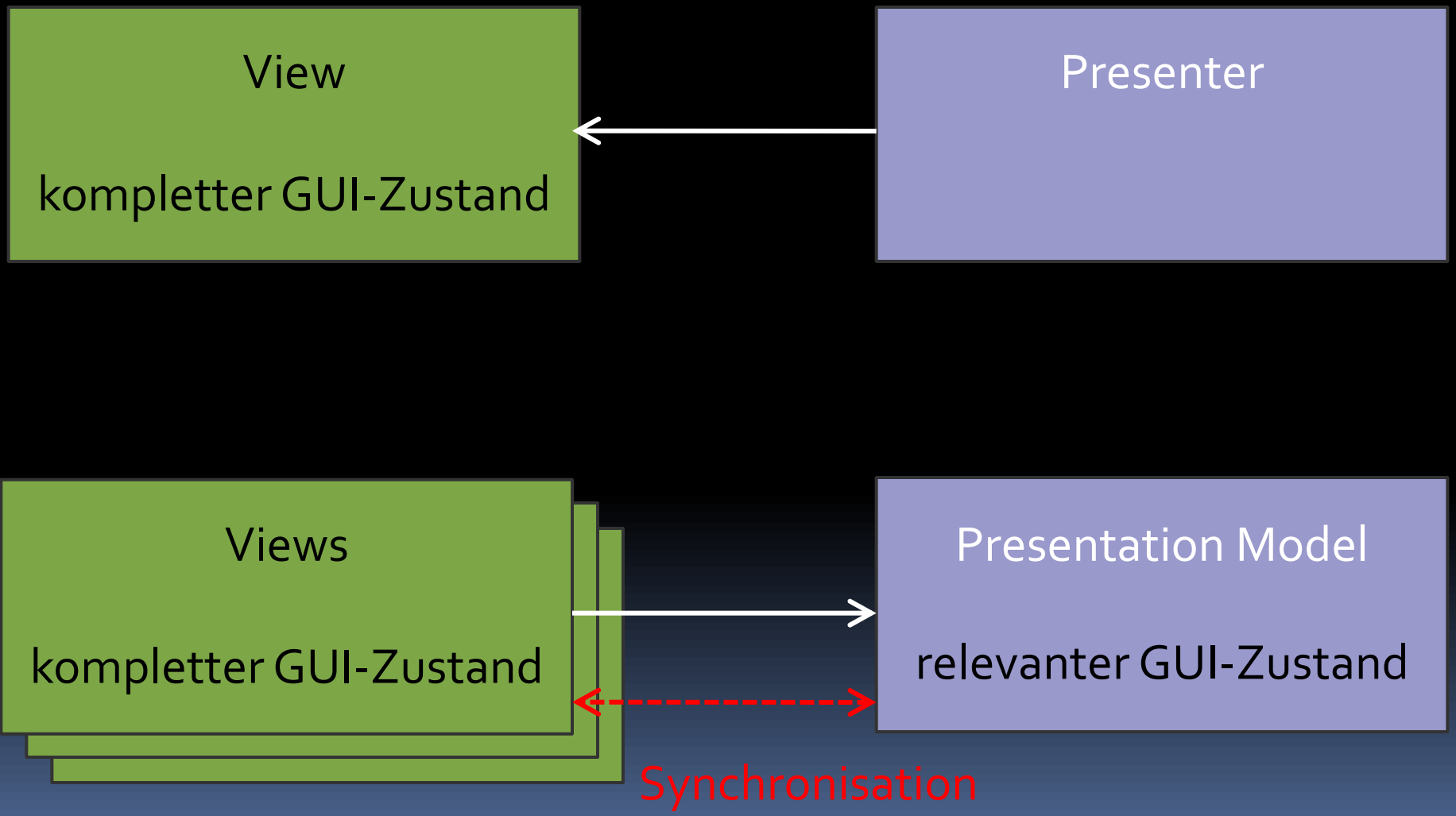
PM: Mehrere Views



PM: Mehrere Views



MVP vs. PM: GUI-Zustand



MVP vs. PM: GUI-Zustand

- MVP
  - View hält den GUI-Zustand
  - Presenter hält keinen Zustand
  - GUI-Zustand wird nicht synchronisiert
- Presentation Model
  - View hält kompletten GUI-Zustand
  - PM hält den relevanten GUI-Zustand
  - Muss PM-Zustand mit View-Zustand synchronisieren

Testen

- MVP
  - Presenter-Test braucht View-Stub
  - View-Vorschau ohne Presenter
- Presentation Model
  - PM-Test ohne View(s)
  - View-Vorschau braucht PM-Stub

MVP vs. PM: Transformation

- Einige Autonomous Views nutzen GUI-Details
- Presenter kann "unsauberen" Code beibehalten
  - Auftrennung in MVP ist leichter
  - Umbau zu MVP kostet weniger
- Auftrennung in PM erfordert Extraarbeit
  - Finde geeignete GUI-Zustandsabstraktionen
  - Schreibe Handler in den Views
  - Räume auf
- Evtl. profitiert man vom Aufräumen

MVP vs. PM: Allgemeines

- Entwickler sind eher gewöhnt, direkt mit GUI-Zustand zu arbeiten
- Presenter hängt ab von GUI-Komponententypen
- MVP: mechanische Trennung
- PM: gedankliche Trennung
- MVP geht Probleme an, die viele mit einer alten Implementierung von PM hatten

Gliederung

Einleitung

Autonomous View

Model View Controller

Model View Presenter

Presentation Model

Datenbindung

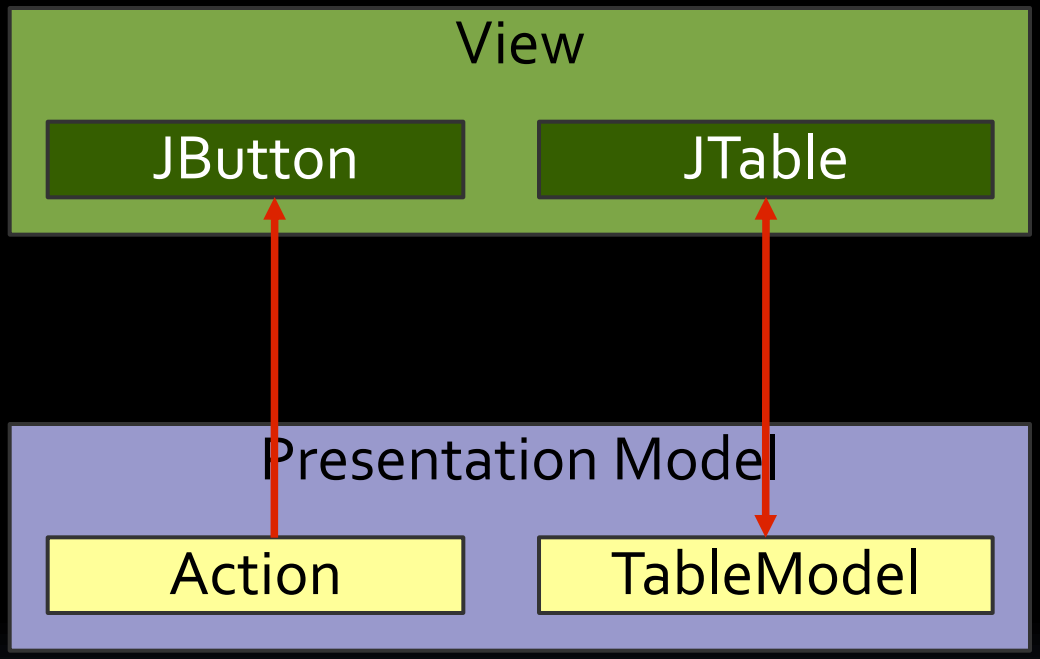
Datenbindung

- Synchronisiert zwei Datenquellen
- In einer oder zwei Richtungen
- Kann häufig Typen wandeln
- Integriert optional eine (Vor-)Validierung

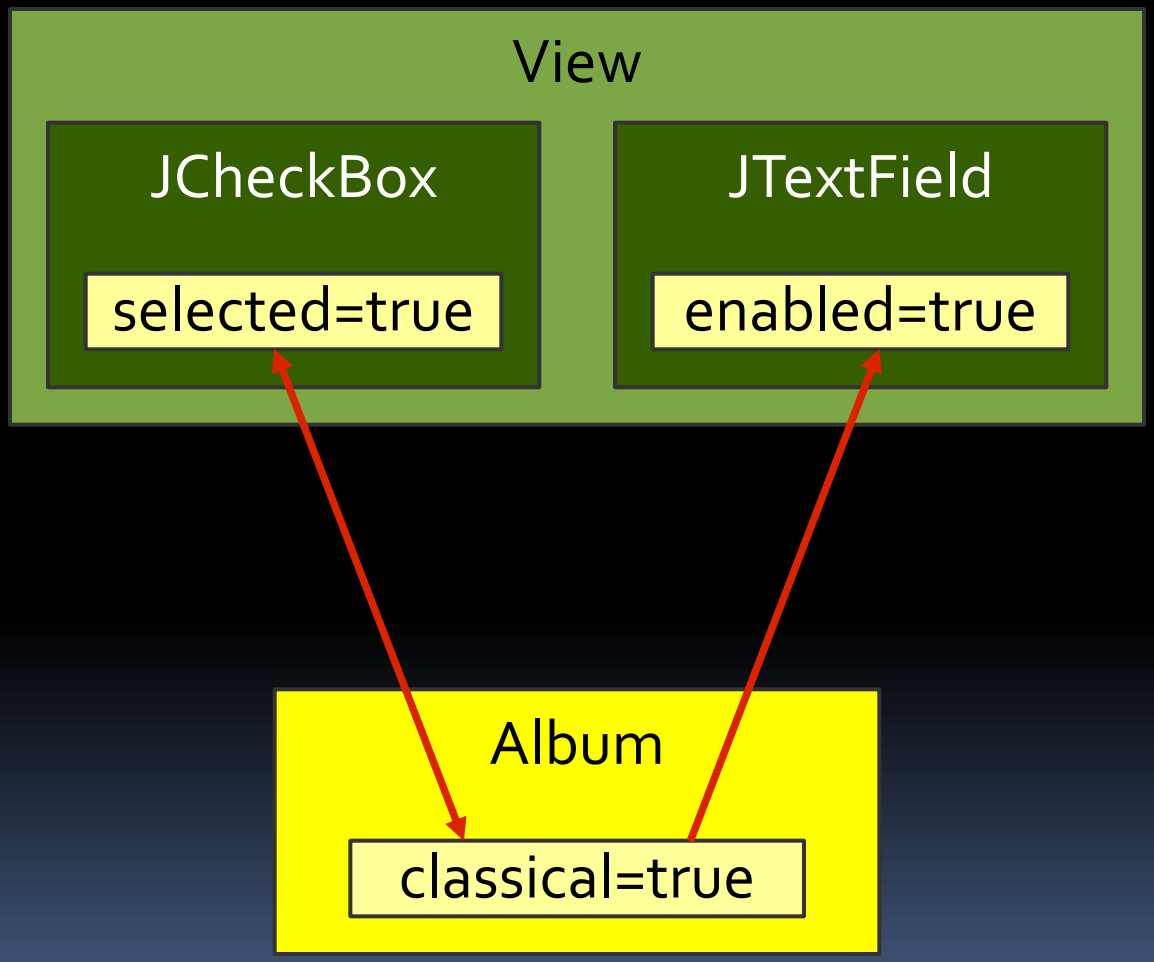
Beispiele

- Action → JButton
- TableModel ↔ JTable
- Album.classical ↔ Classical JCheckBox
- Album.classical → Composer JTextField.enabled
- Database ↔ GUI-Formular
- Web Service → JTable

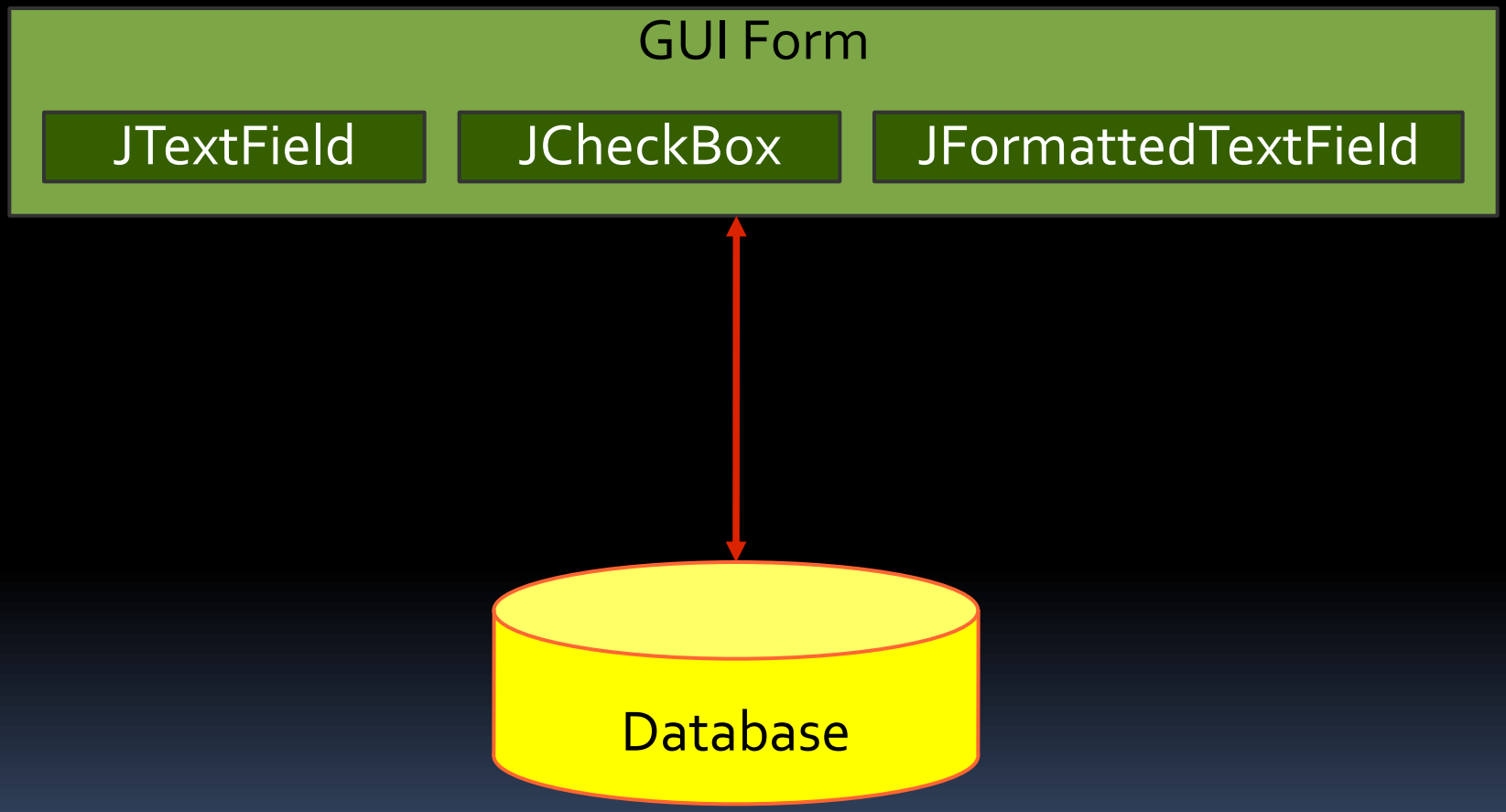
Beispiel: GUI-Modell mit View



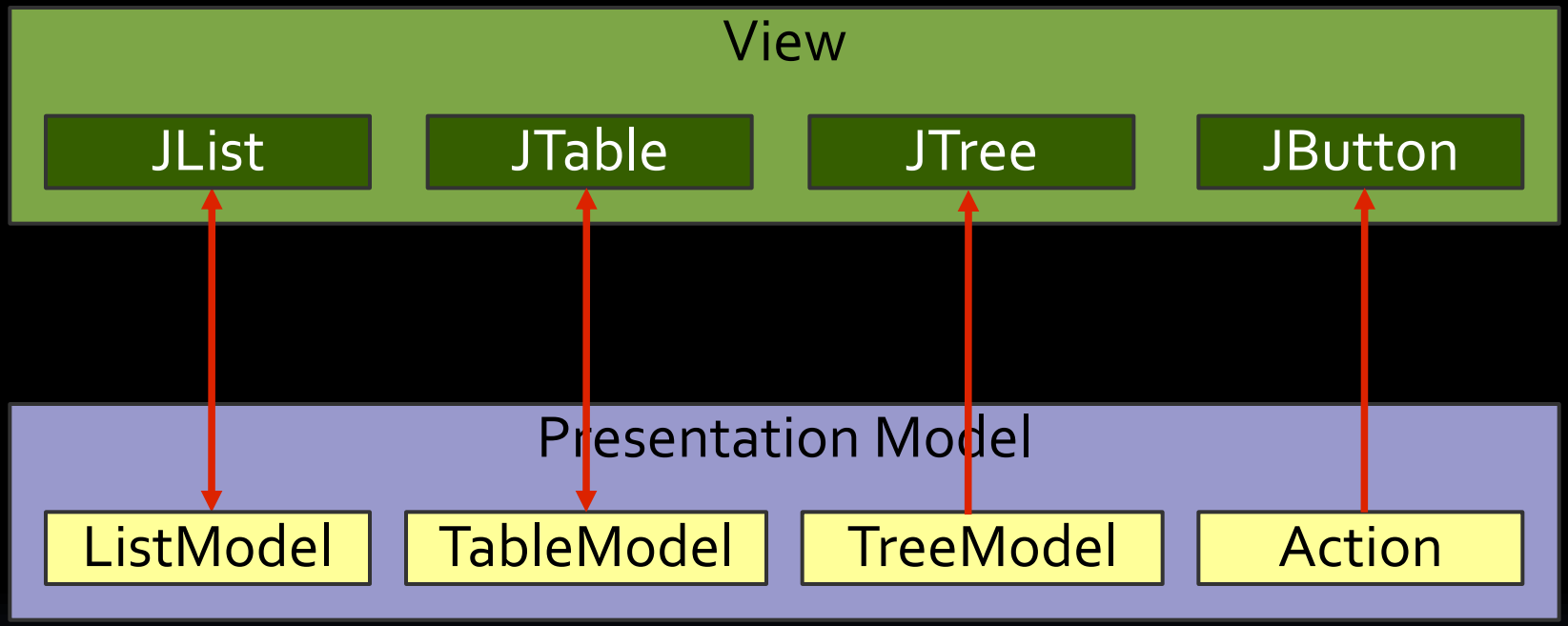
Bsp.: UI- mit Fachzustand



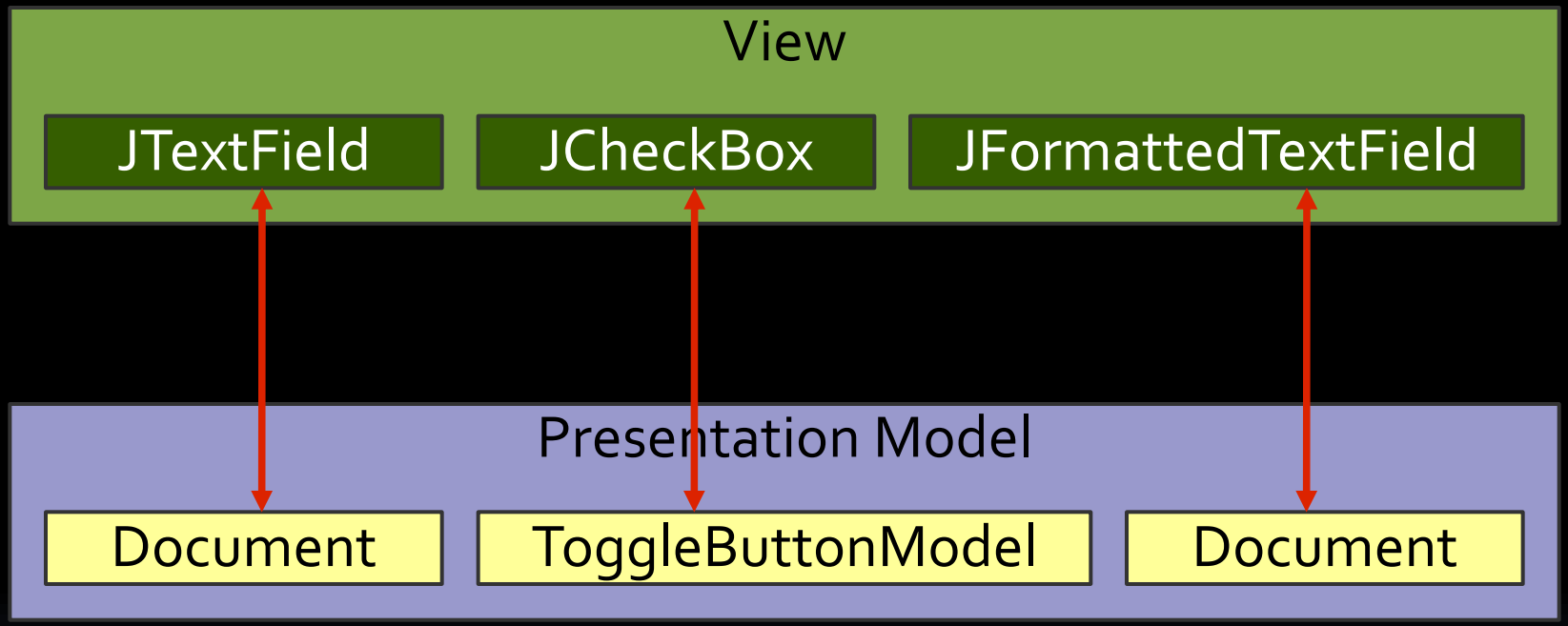
Bsp.: Formularwerte mit DB



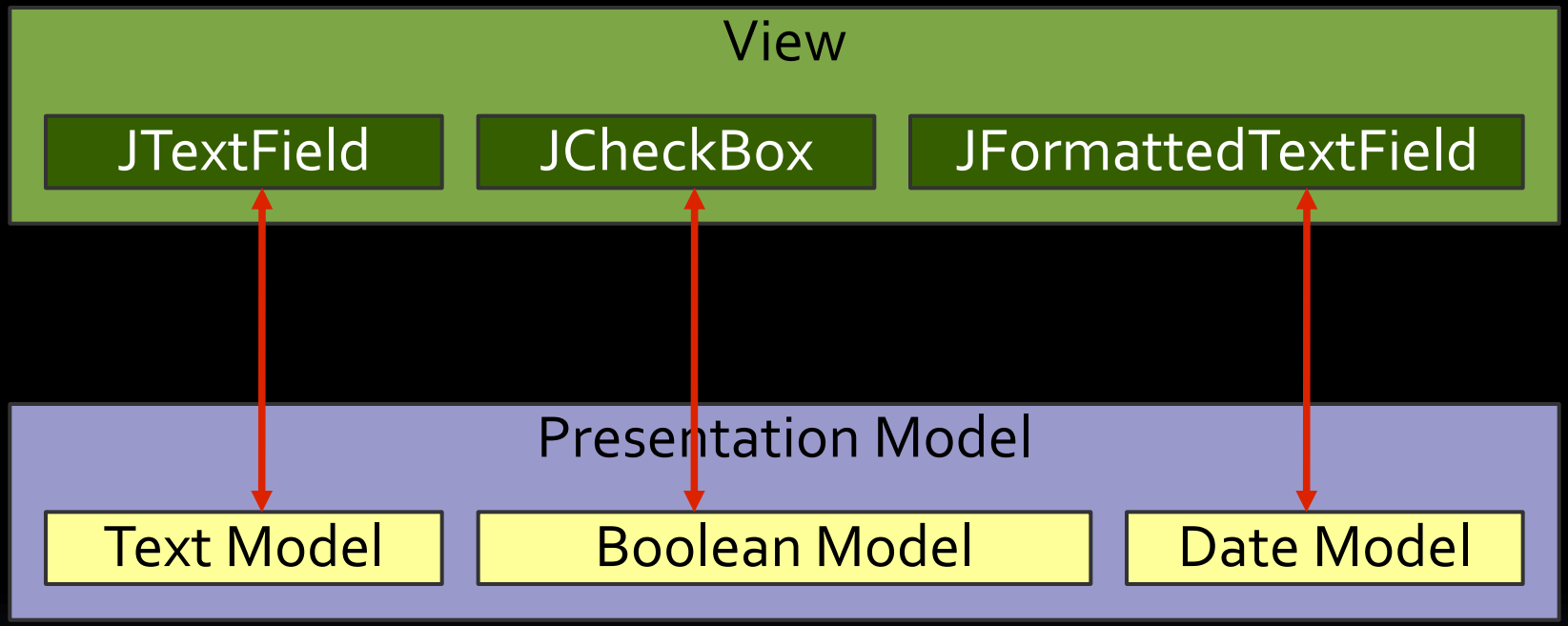
Nützliche Swing-Bindungen



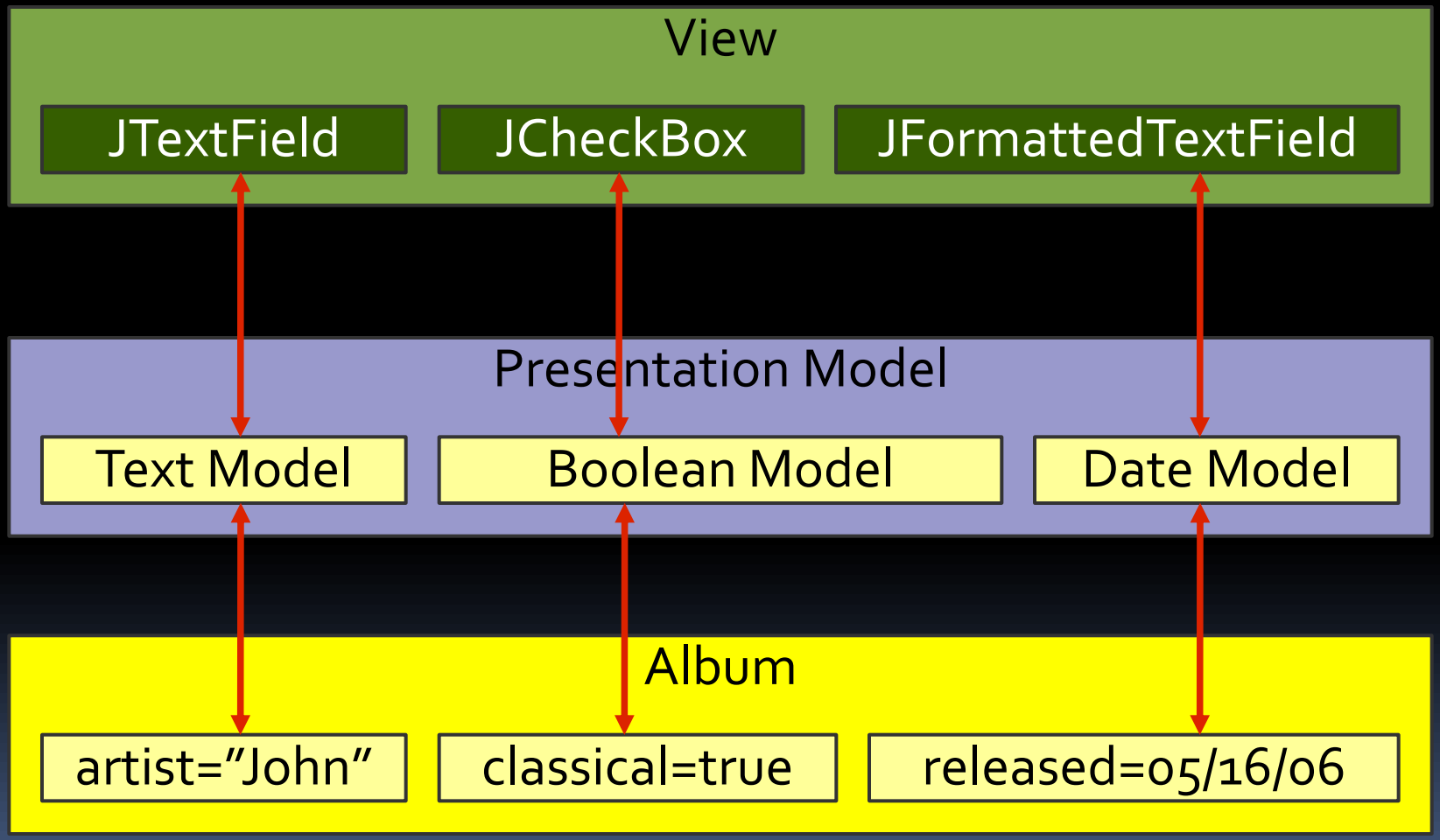
Low-Level-Modelle in Swing



Gesucht: Geeignete Modelle



Gesucht: Synchronisationspfad



JGoodies Binding

- Nutzt Swing-Bindungen für
  - JList, JTable, JComboBox, JTree, JButton
- Springt ein, wo geeignete Modelle fehlen
  - JTextField, JCheckBox, …
- Wandelt Bean-Eigenschaften in einheitliche Modelle (ValueModel)

- Macht Kniffliges möglich
- Macht Einfaches ein bisschen einfacher

AlbumView: Init mit binden

```
private void initComponents() {
  artistField = Factory.createTextField(
     presentationModel.getModel("artist"));

  classicalBox = Factory.createCheckBox(
     presentationModel.getModel("classical"));

  songList = Factory.createList(
     presentationModel.getSongsAndSelection());

  okButton = new JButton(
  presentationModel.getOKAction());
}
```

AlbumView: EnablementHandler

```
private void initPresentationLogic() {

    // Synchronisiere das Feld-Enablement
    // mit dem PresentationModel-Zustand.
    PropertyConnector.connect(
        presentationModel,
        "composerEnabled",
        composerField,
        "enabled");

}
```

JSR 295: Beans Binding

- Synchronisiert eine Datenquelle mit einem Ziel, typischerweise zwei bound Bean-Eigenschaften
- Soll Typwandlung und Validierung können
- Erster Code ist vorhanden
- JSR hat noch keinen Draft, inaktiv
- Spec lead weg
- Projekt tot, Abspaltung geplant

JavaFX

JGoodies Binding vs. JSR 295

- Property-Pfade
- Bindeobjekte vs. Expression language
- Kombinieren der Bindeobjekte
- Synchron zu einem Teilgraphen vs. teilweise synchron zu einem großen Graphen
- Puffern
- Verzögern

Kopieren …

- Einfach zu verstehen
- Geht fast immer
- Einfach zu debuggen; alle Operationen explizit

- Macht's schwer, Views zu synchronisieren
- Erfordert Disziplin im Team
- Eher für grobkörnige Updates
- ~~Führt zu immer gleichem Code-Bläh~~

… vs. automatische Bindung

- Feinkörnige Aktualisierungen
- Erleichtert Synchronisation
- Deutlich schwerer zu verstehen und zu debuggen
- Mehraufwand beim Umbenennen und für Obfuscator

Kosten automatischer Bindung

- Erhöht Lernkosten
- Senkt Produktionskosten
- Kann Änderungskosten erheblich senken

Zusammenfassung

- Ausgangspunkt: Separated Presentation
- Üblich und OK: Autonomous View
- MVP arbeitet mit View-GUI-Zustand
- PM kopiert Zustand und braucht Synchronisation
- Swing unterstützt Presentation Model bereits

Ratschläge

- Nutze Separated Presentation wenn möglich
- Teile Autonomous Views wenn geeignet
- Lies Fowlers "Organizing Presentation Logic"

- Nutze automatische Datenbindung nur
  - wenn sie zuverlässig und flexibel ist
  - mindestens einer im Team sie beherrscht

Sonstiges

- Event Bus
- JSR 296 – Swing Application Framework
  - organisiert, vereinfacht, standardisiert
  - Ressourcen-Management
  - Action-Management
  - Hintergrundaufgaben

Weitere Informationen

- Martin Fowler: *Further P of EAA*
  - google "Organizing Presentation Logic"
- Scott Delap: *Desktop Java Live*
- JGoodies-Artikel - www.jgoodies.com/articles
  - Swing Data Binding
  - JSR 296

Datenbindungssysteme

- JFace Data Binding
  - siehe auch UFaceKit
- Beans Binding
- BetterBeansBinding
  - Fabrizio Giucis BeansBinding-Abspaltung
- JGoodies Binding
  - Tutorial enthält PM-Beispiele
- Oracle ADF otn.oracle.com
  - suche "JClient"

Swing-Überlebenshilfe

Desktop-Muster

Datenbinding

JSR 296

Erste Hilfe für Swing

Layout-Management

Meta-Design

FRAGEN UND ANTWORTEN

JGoodies Karsten Lentzsch

DESKTOP-MUSTER & DATENBINDUNG