Desktop Patterns and Data Binding

Karsten Lentzsch

Founder
JGoodies
www.JGoodies.com

TS-1074

Goal

Learn how to organize presentation logic and how to bind domain data to views

Agenda

Introduction

Autonomous View

Model View Controller

Model View Presenter

Presentation Model

Data Binding

Agenda

Introduction

Autonomous View

Model View Controller

Model View Presenter

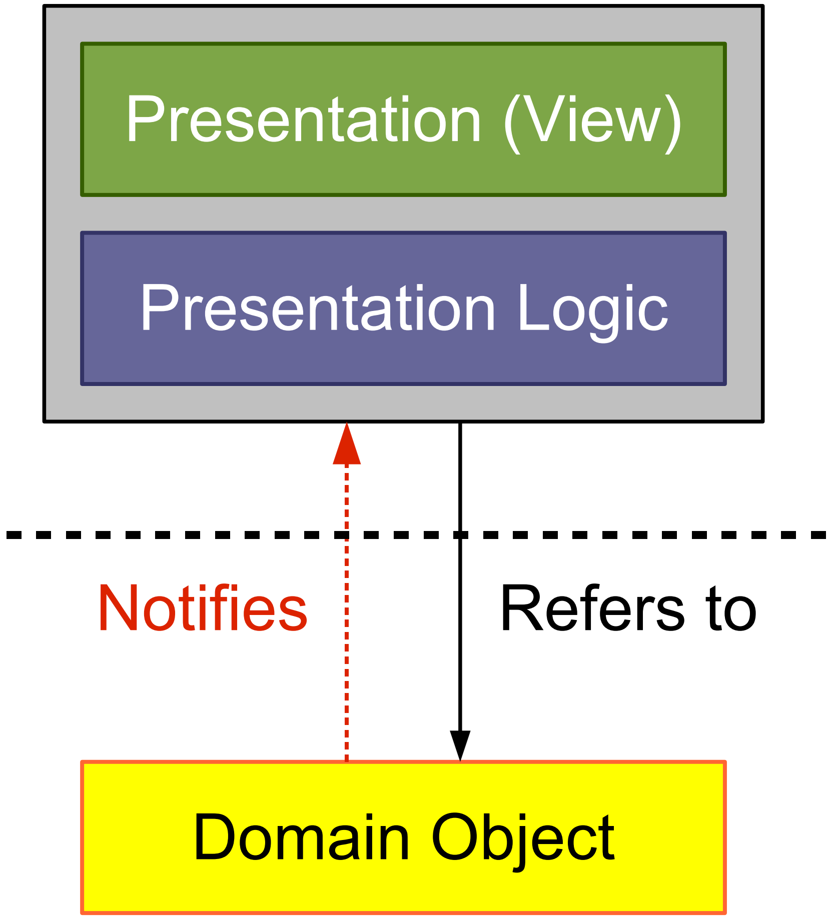
Presentation Model

Data Binding

Questions

- How shall I structure my application?
- How to separate concerns?
- How to build a view?
- Who should handle events?
- Do I need a controller?
- How can I test my GUI logic?

Legend


Presentation (View)
Presentation Logic
Notifies
Refers to
Domain Object

Legend

- Domain/business logic
- Examples:
  - Book
  - Person
  - Address
  - Invoice
- More generally: object graph

Domain Object

Legend

Presentation Logic

- Handlers for:
  - List selection changes
  - Check box selection
  - Drag drop end
- UI models
  - ListModel
  - TableModel
  - TreeSelectionModel
- Swing actions

Event Handling vs. Presentation Logic

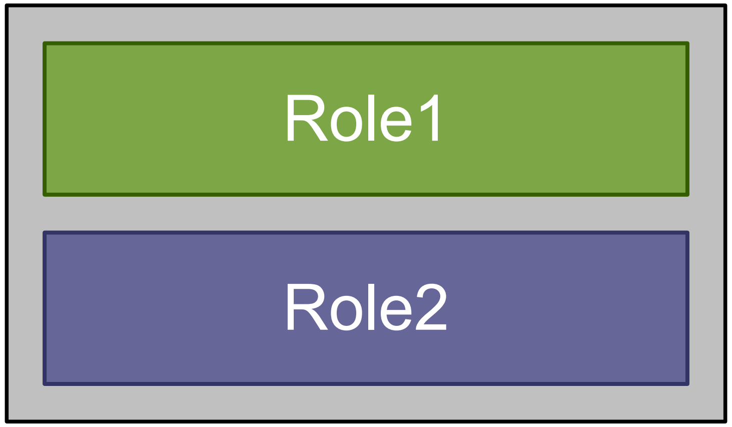
- Toolkit handles fine-grained events:
  - Mouse entered, exited
  - Mouse pressed
  - Radio button pressed, armed, rollover
- Application handles coarse-grained events:
  - Radio button selected
  - Action performed
  - List items added
  - Domain property changed

Legend

Presentation (View)

- Container:
  - JPanel, JDialog, JFrame
- Contains components:
  - JTextField, JList, JTable
- Component initialization
- Panel building code
- GUI state:
  - Check box pressed
  - Mouse over

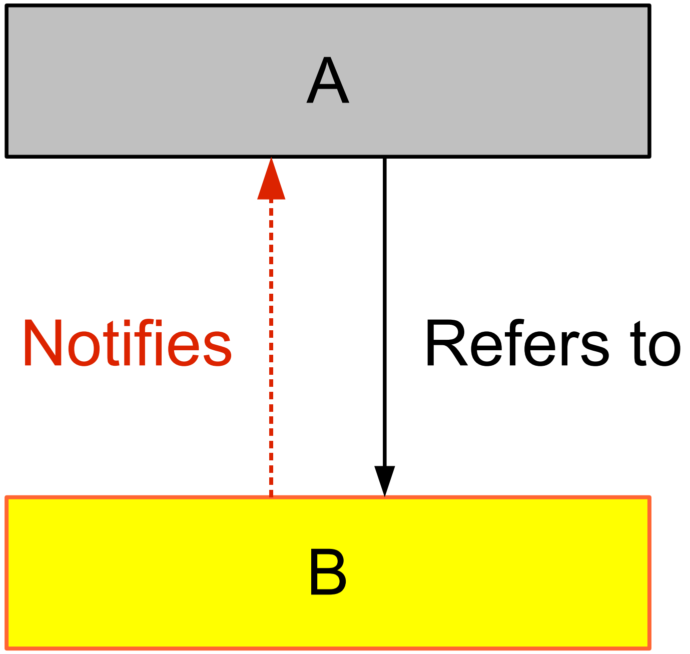
Legend



- Role1 and Role2 **sit together** in a class
- Can access each other

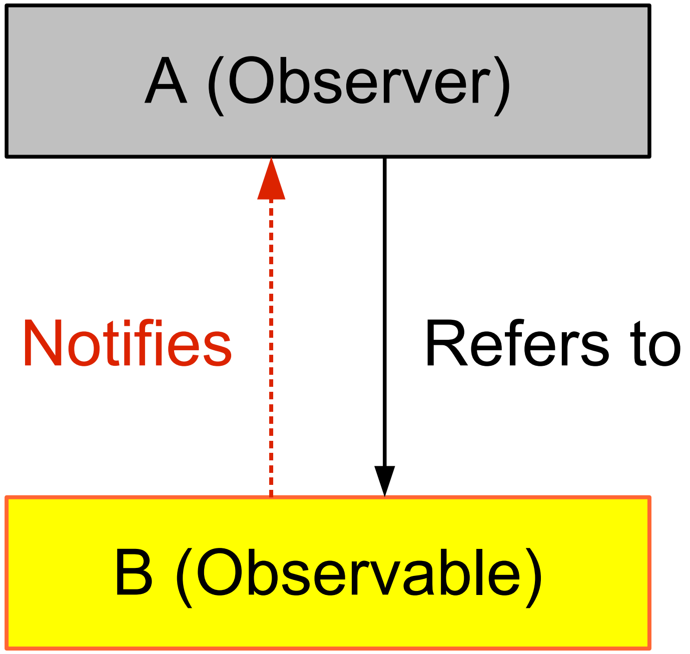
- Separated layers

Legend



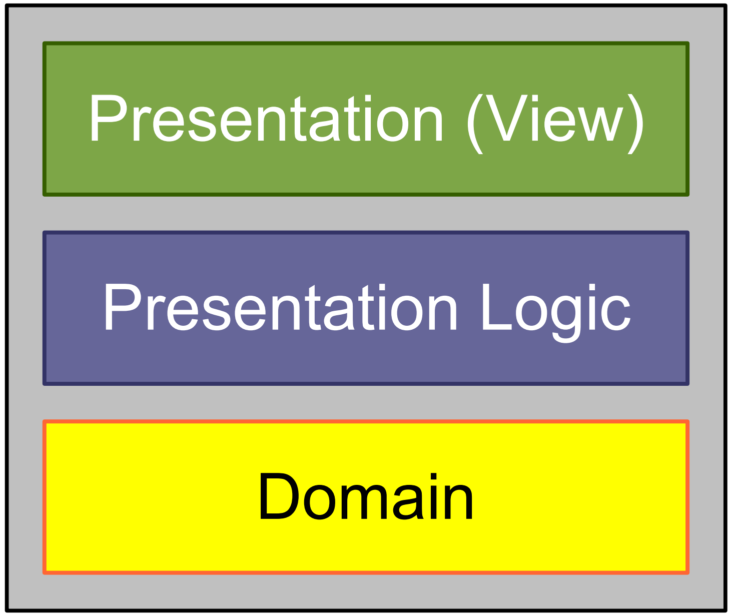
- A refers to B
- A holds a reference to B
- B indirectly refers to A

Legend

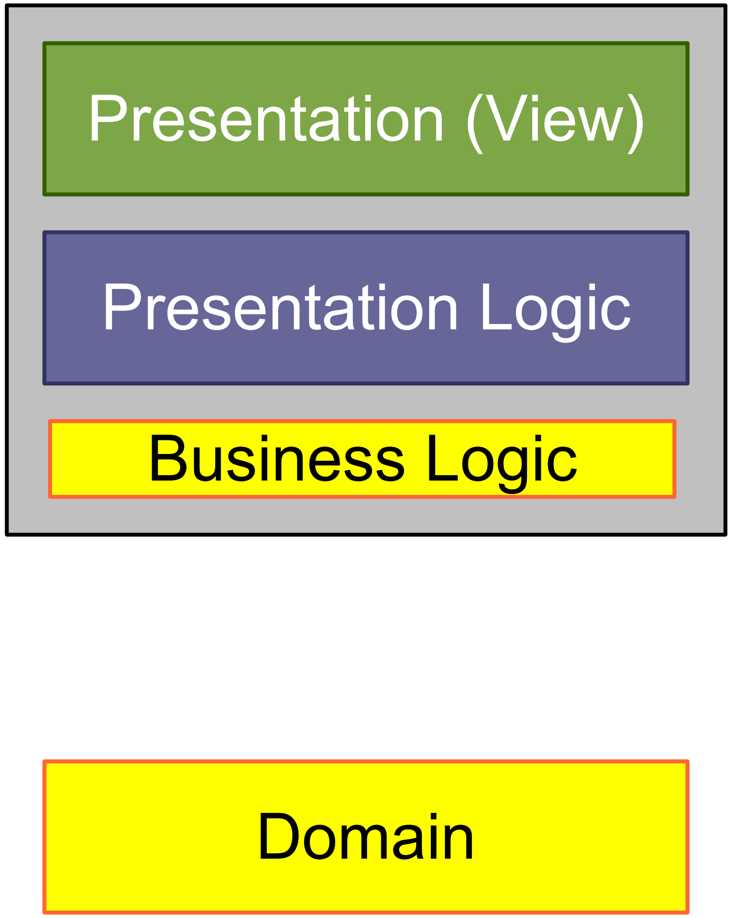


- A observes B’s changes
- A is an **Observer**
- B is an **Observable**

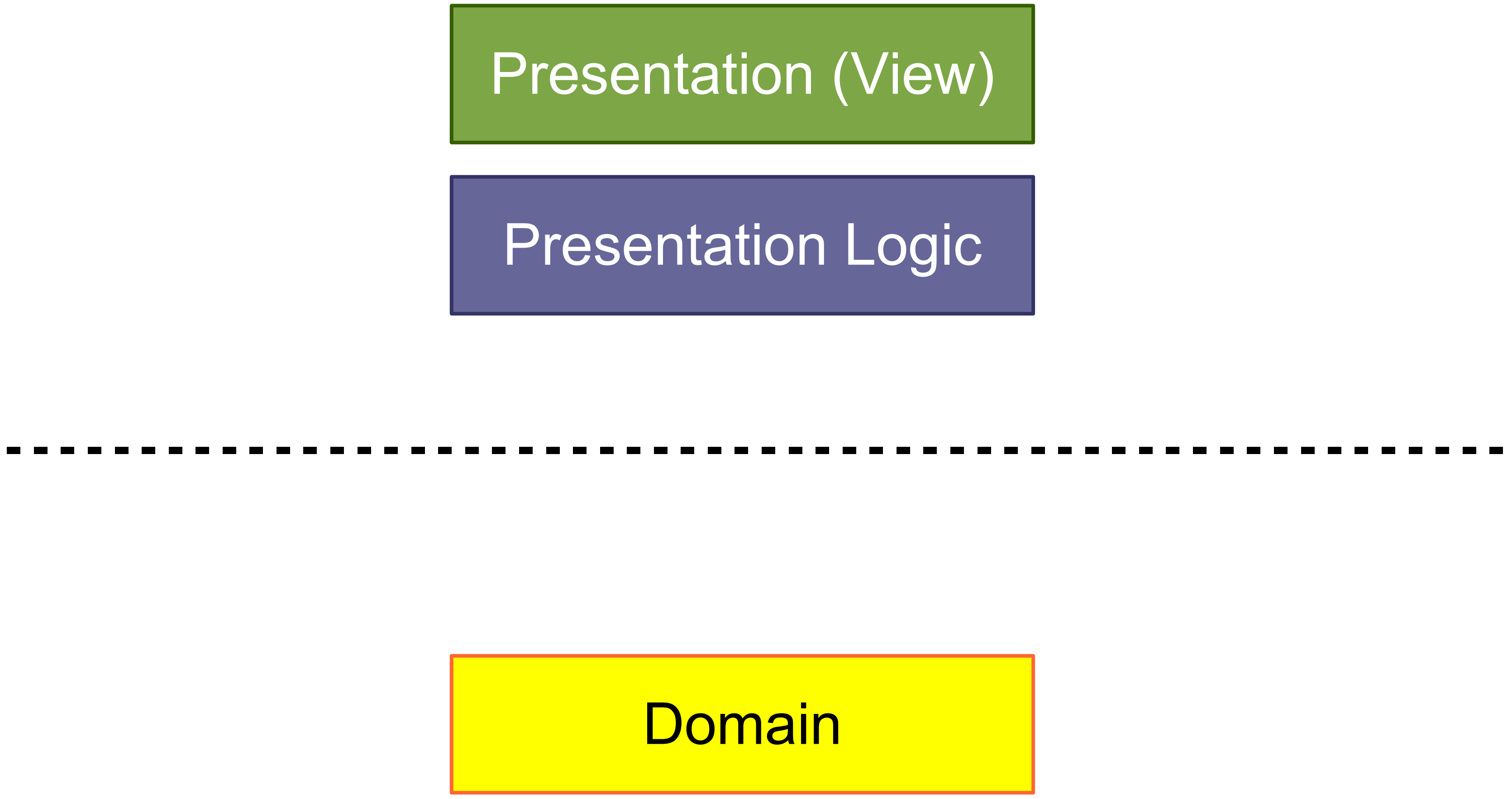
All Mixed Together



Business Logic in the Presentation



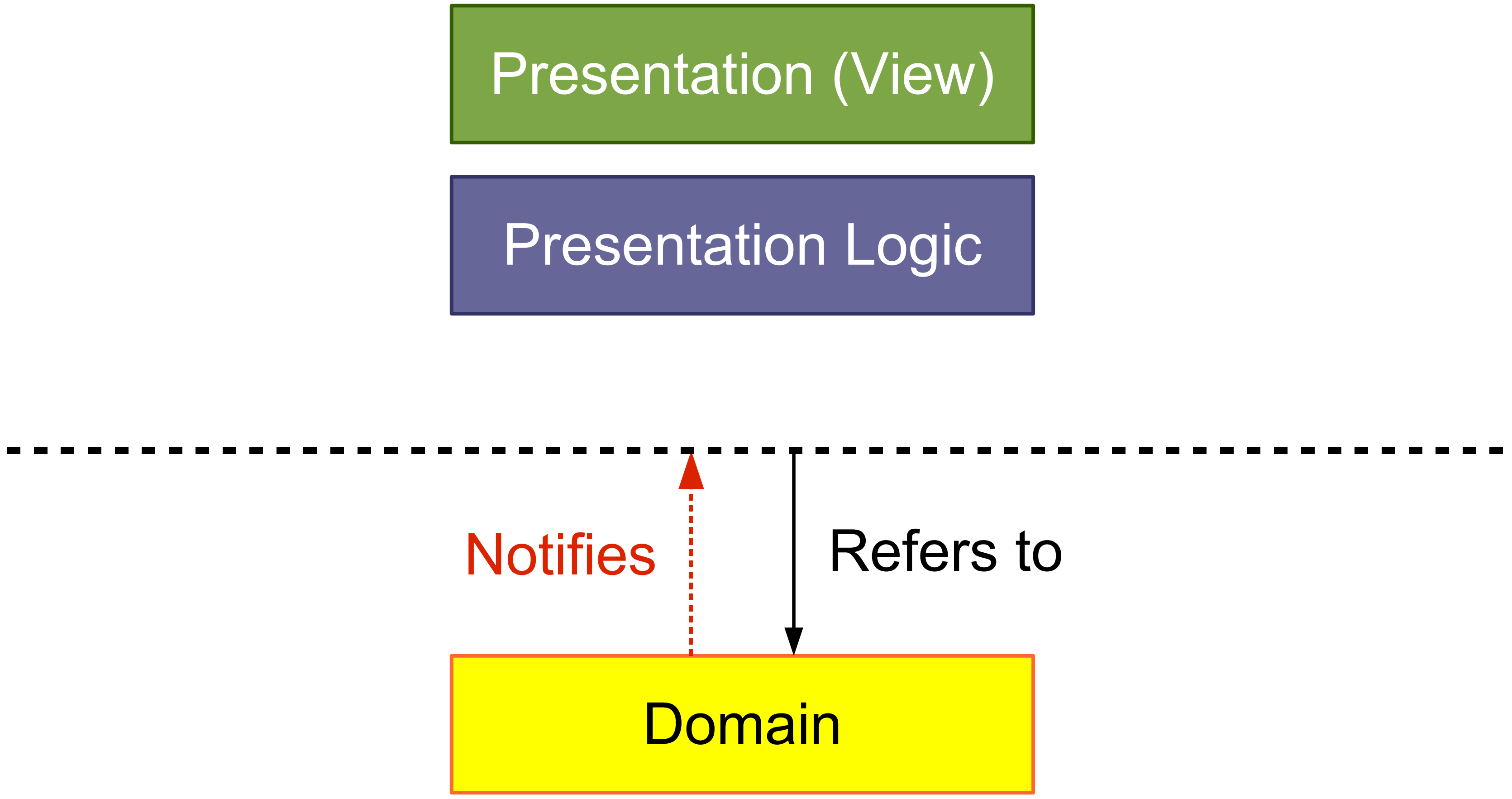
Pattern: Separated Presentation



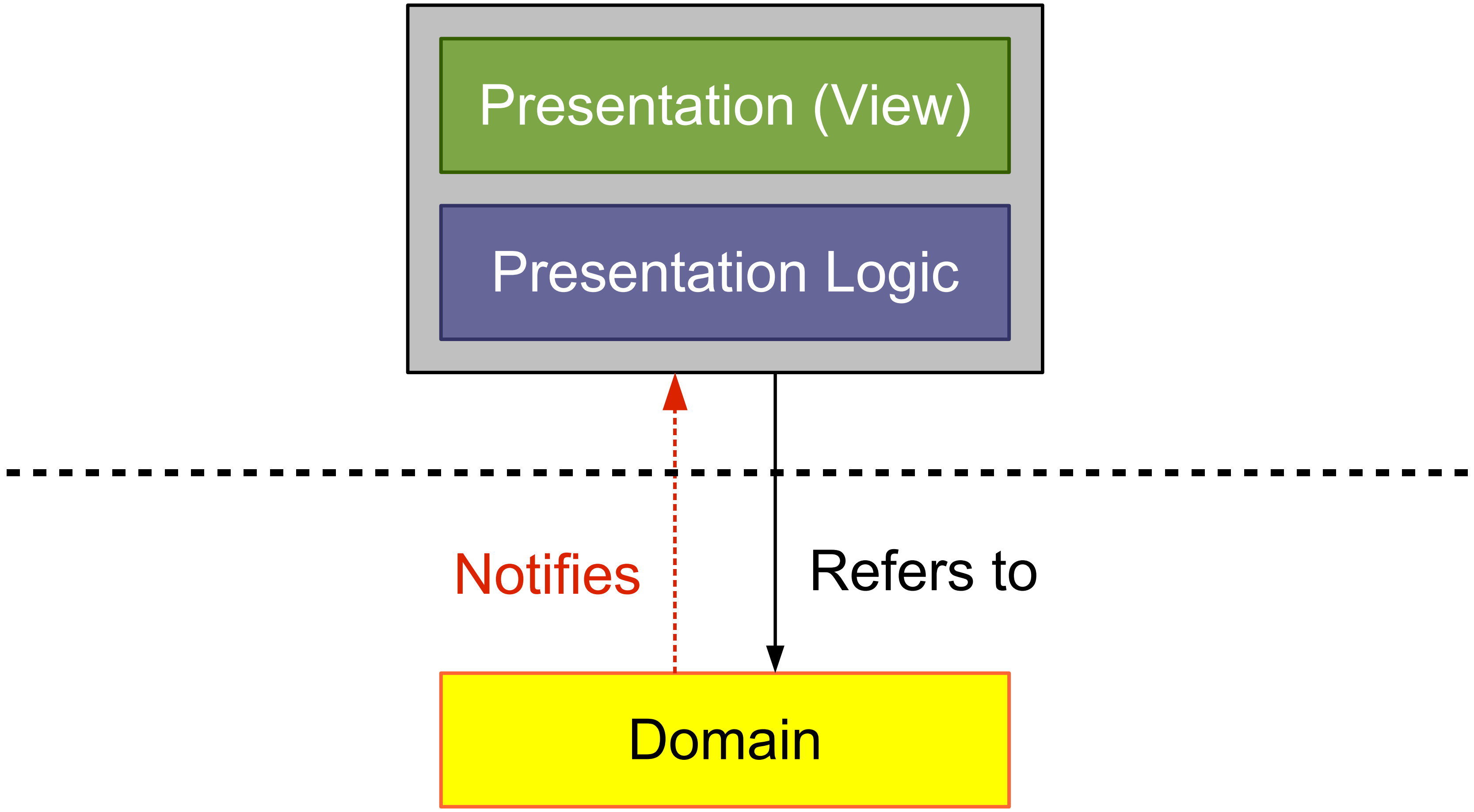
Decouple Domain From Presentation

- The domain shall not reference the presentation
- Presentation refers to domain and modifies it
- Advantages:
  - Reduces complexity
  - Multiple presentations

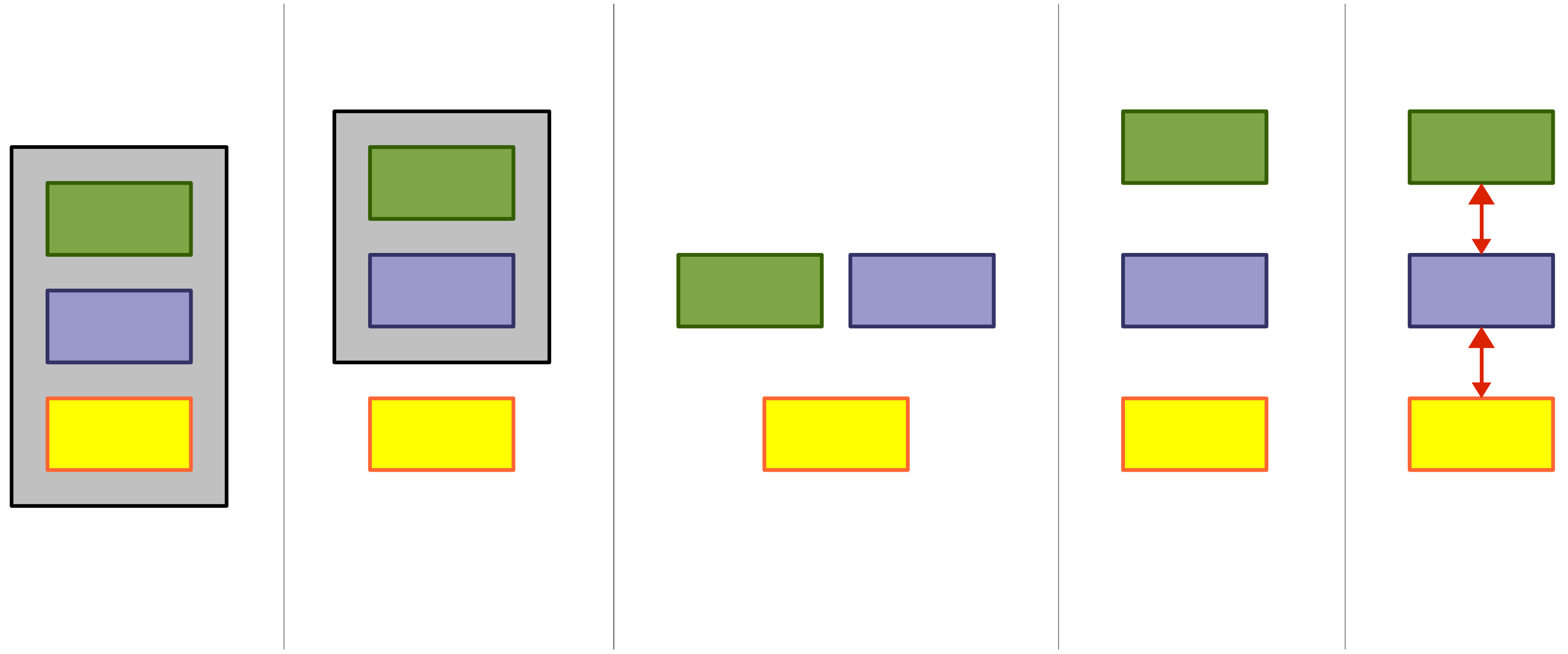
Separated Presentation with Observer



Separated Presentation



Visual Agenda



Agenda

Introduction

Autonomous View

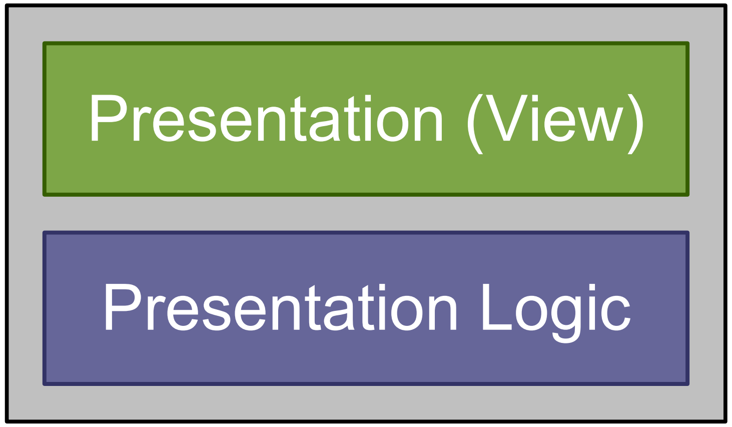
Model View Controller

Model View Presenter

Presentation Model

Data Binding

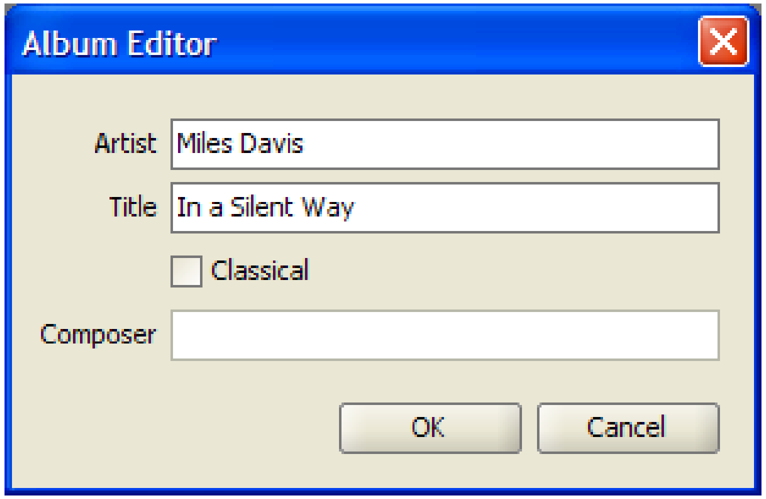
Pattern: Autonomous View



Autonomous View

- Often one class per window or screen
- Often a subclass of JDialog, JFrame, JPanel
- Contains:
  - Fields for UI components
  - Component initialization
  - Panel building/layout
  - Model initialization
  - Presentation logic: listeners, operations

Example GUI

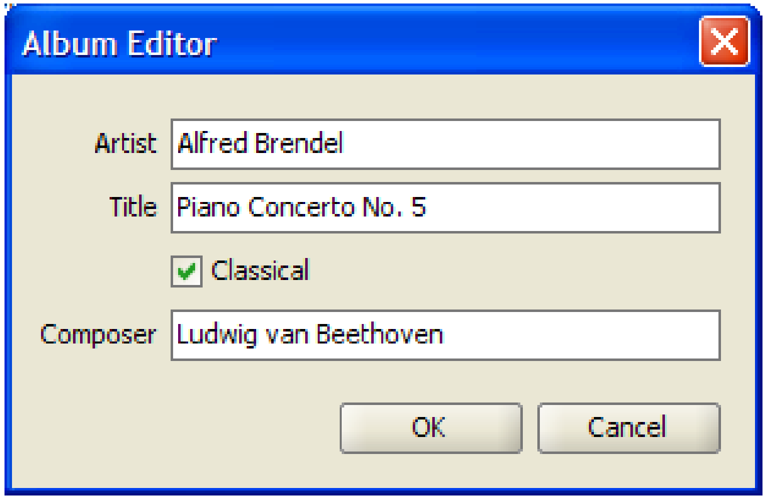

Album Editor

Artist: Miles Davis

Title: In a Silent Way

☐ Classical

Composer:


Album Editor

Artist: Alfred Brendel

Title: Piano Concerto No. 5

✓ Classical

Composer: Ludwig van Beethoven

Composer field is **enabled,** if classical is **selected**

Autonomous View Sample (1/2)

```
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 Sample (2/2)

```
class ClassicalChangeHandler
                         implements ChangeListener {

   public void stateChanged(ChangeEvent e) {
      // Enable or disable the composer field.
   }
}

class OKActionHandler implements ActionListener {

   public void actionPerformed(ActionEvent e) {
      // Commit changes and close the dialog.
   }
}
```

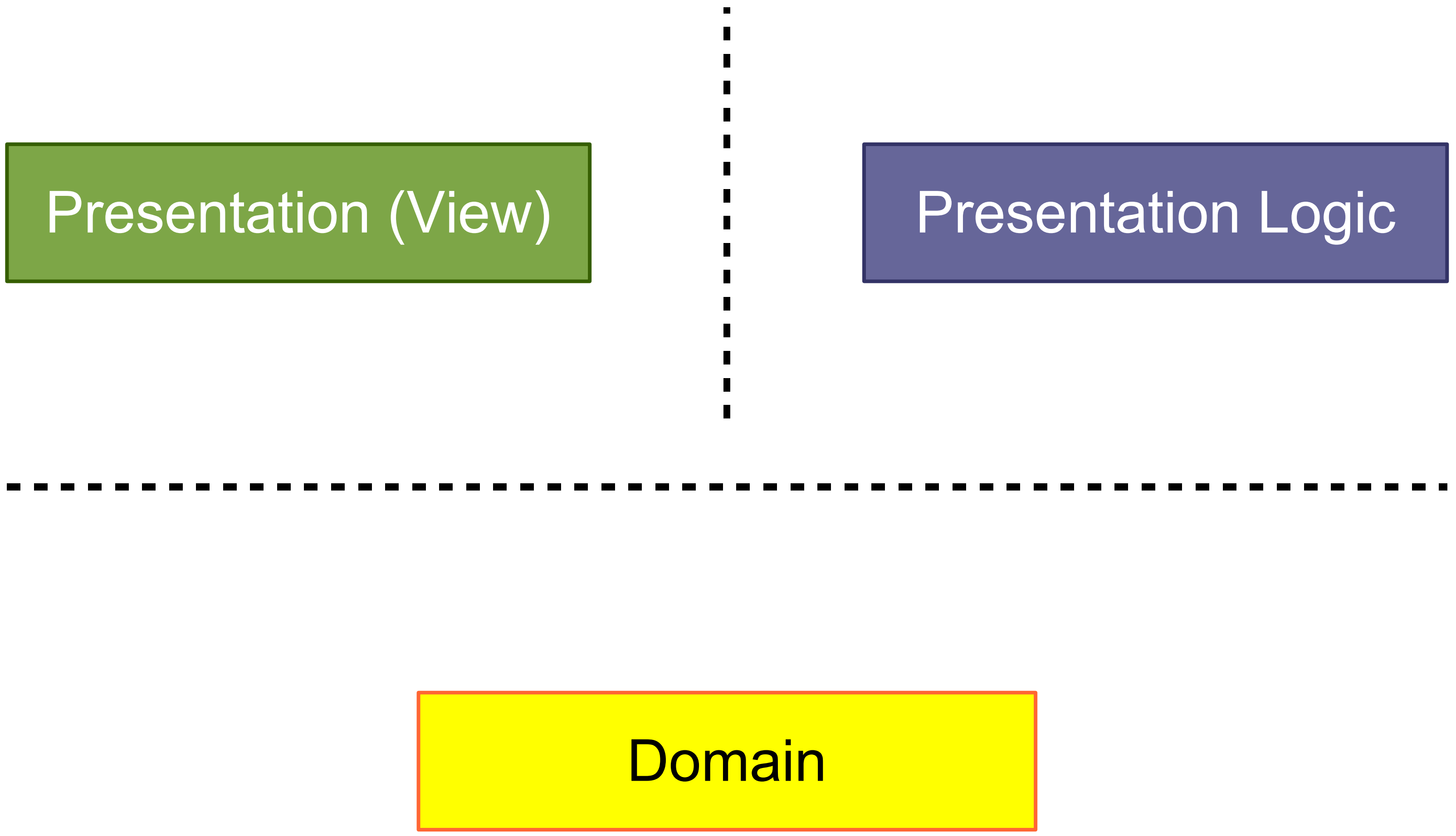
Autonomous View: Tips

- **Build** dialogs, frames, panels
- **Extend** JDialog, JFrame, JPanel if necessary
  Do you extend or use HashMap?
- Compose large screens from small panels
  - In simple cases use build methods like: #buildMainPanel, #buildButtonBar, etc
  - Otherwise nest subpanels

Autonomous View

- Common and workable
- Has disadvantages:
  - Difficult to test logically
  - Difficult to overview, manage, maintain, and debug, if the view or logic is complex
- Consider to separate the logic from the view

Presentation Logic Separated



Separated Logic: Advantages I

- Allows to test the presentation logic logically
- Simplifies team synchronization
- Each part is smaller and easier to overview
- Allows to build **forbidden zones**
    - For team members
    - Before you ship a new release
        - Layout changes allowed
        - Design is done, but bug fixes in the logic are still allowed

Separated Logic: Advantages II

- Thin GUI:
  - Easier to build, understand, maintain
  - Can follow syntactical patterns
  - More team members can work with it
- Logic can ignore presentation details, e.g. component types (JTable vs. JList)
- Logic can be reused for different views

Separated Logic: Disadvantages

- Extra machinery to support the separation
- Extra effort to read and manage multiple sources

Separating Logic from the View

- Can simplify or add complexity
- Separation costs vary with the pattern used
- **Opinion:** typically you benefit from the separation

My personal guideline for team projects:

- Use Autonomous View for message dialogs
- Otherwise separate the logic from the view

Agenda

Introduction

Autonomous View

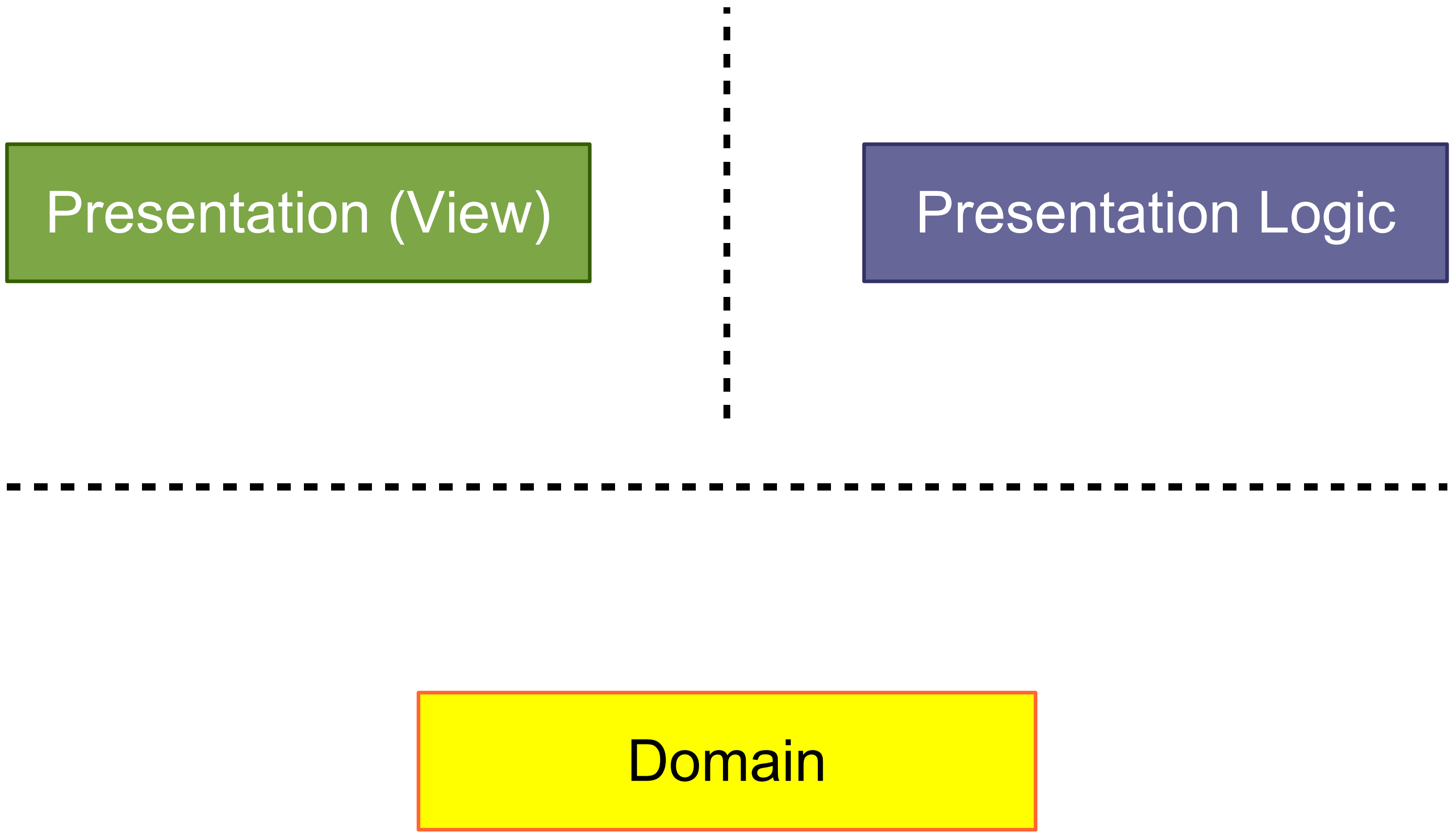
Model View Controller

Model View Presenter

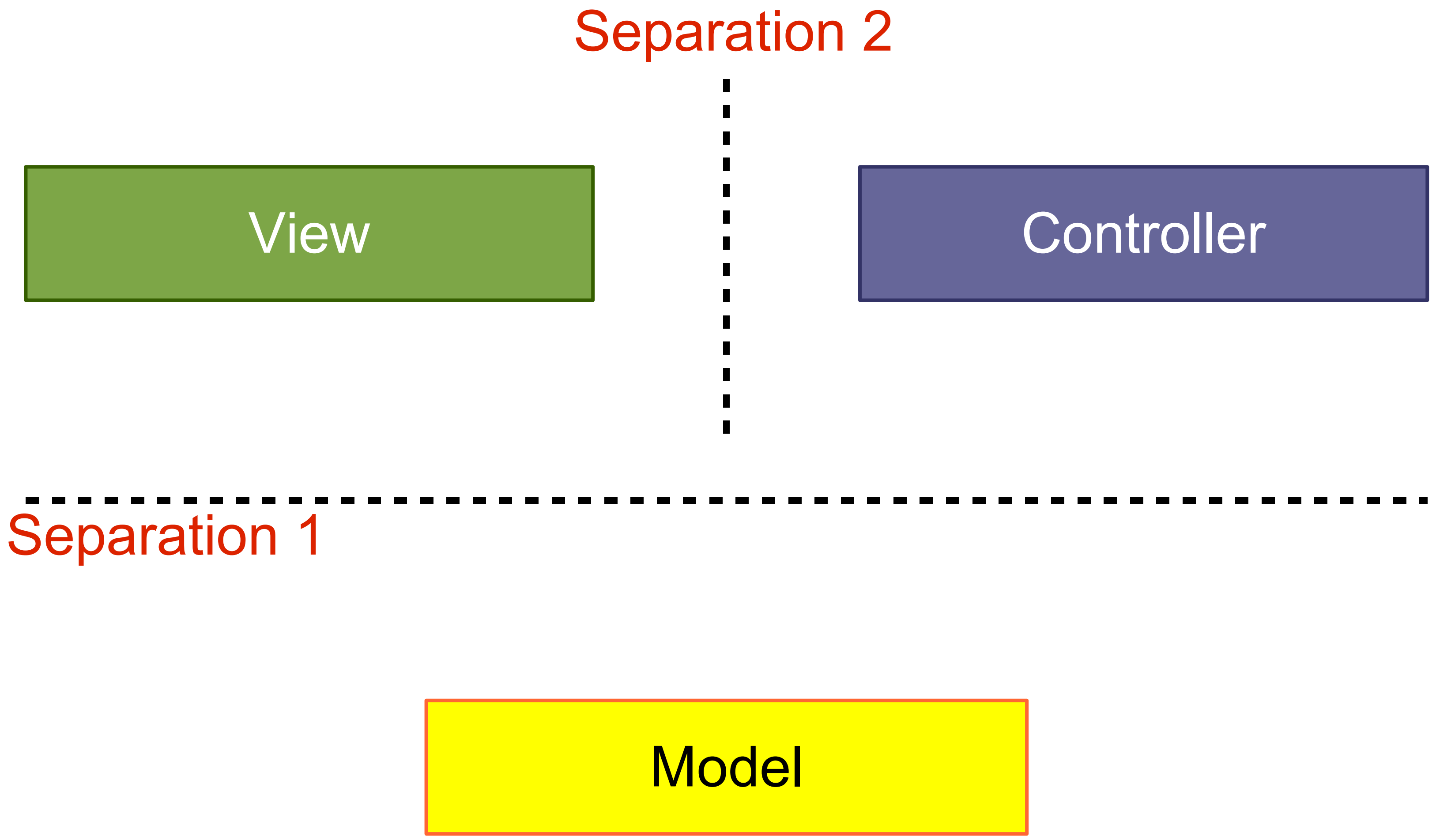
Presentation Model

Data Binding

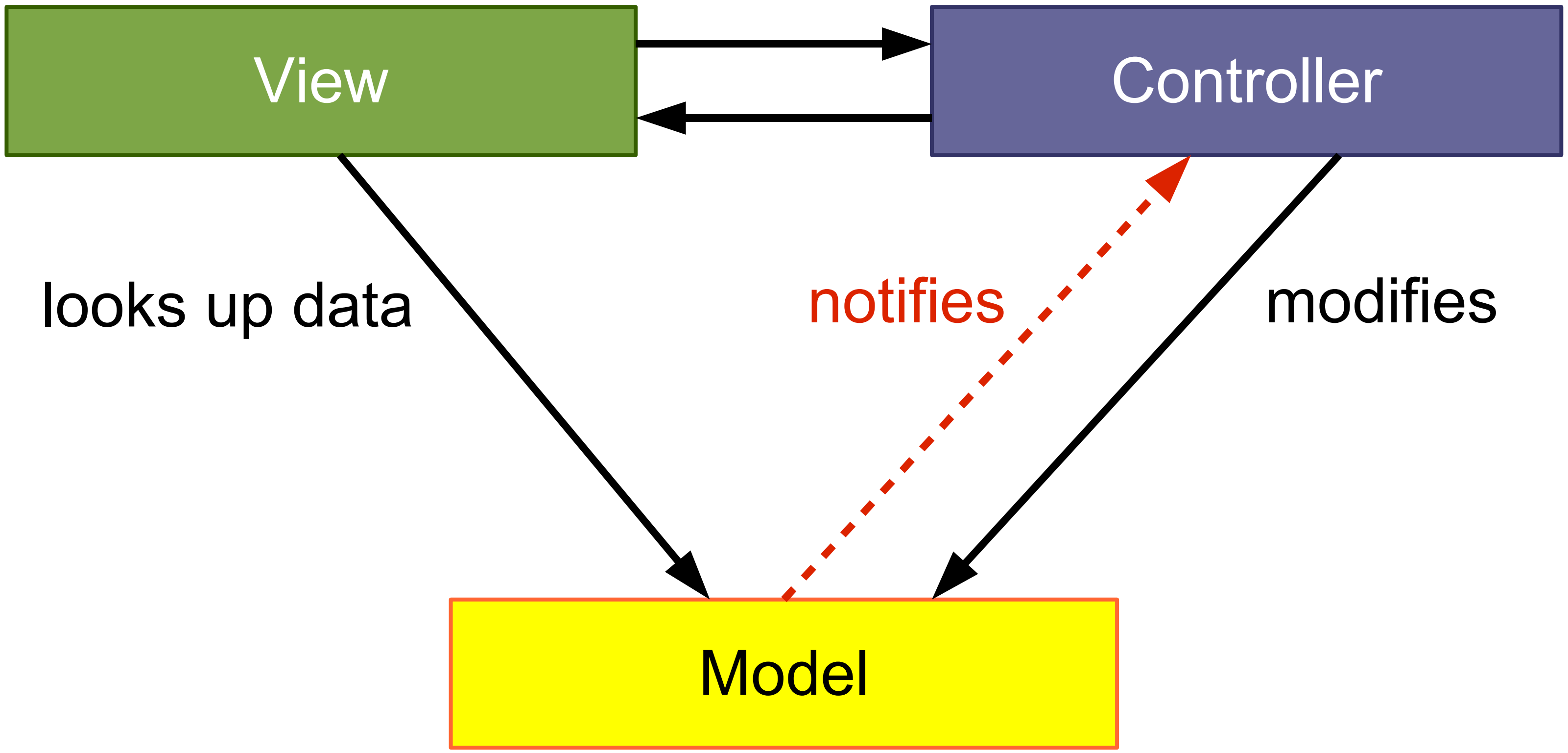
Presentation Logic Separated



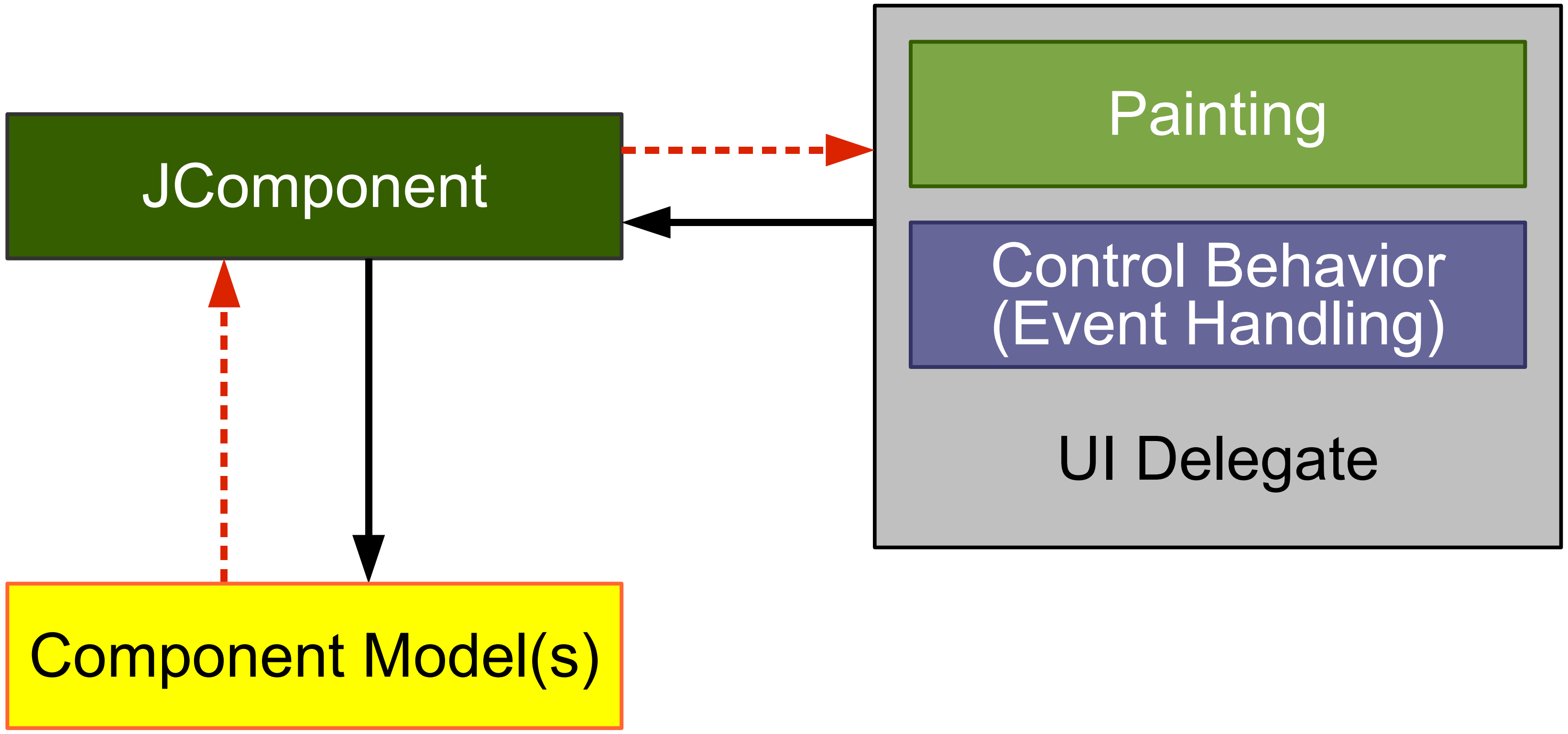
Motivation for Model-View-Controller



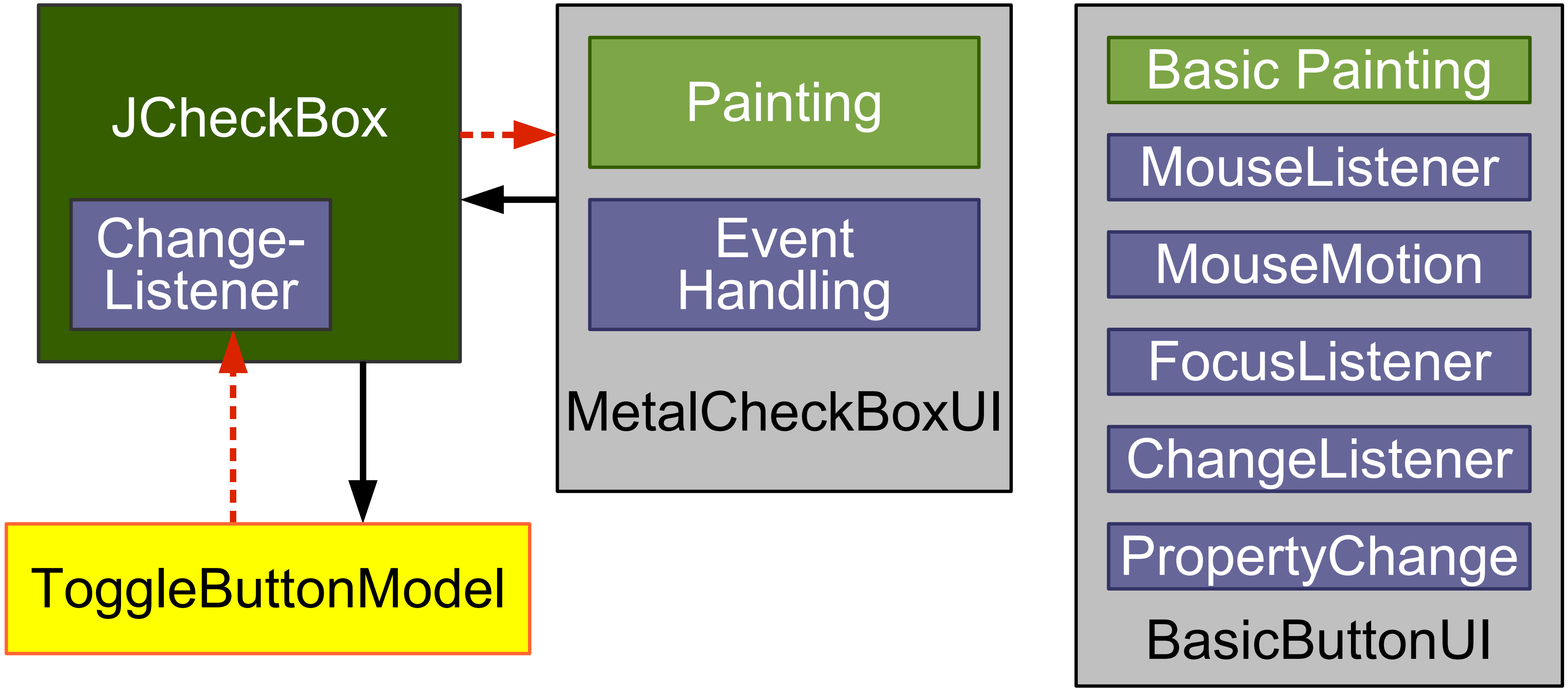
Pattern: Model-View-Controller



Swing: M-JComponent-(VC)



JCheckBox


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

MVC vs. Swing

- MVC separates the View and Controller
- Swing does **not** separate the View and Controller
- UI delegates display data **and** handle events
- MVC useful for component level and app level
- Swing does not use MVC at the component level

Agenda

Introduction

Autonomous View

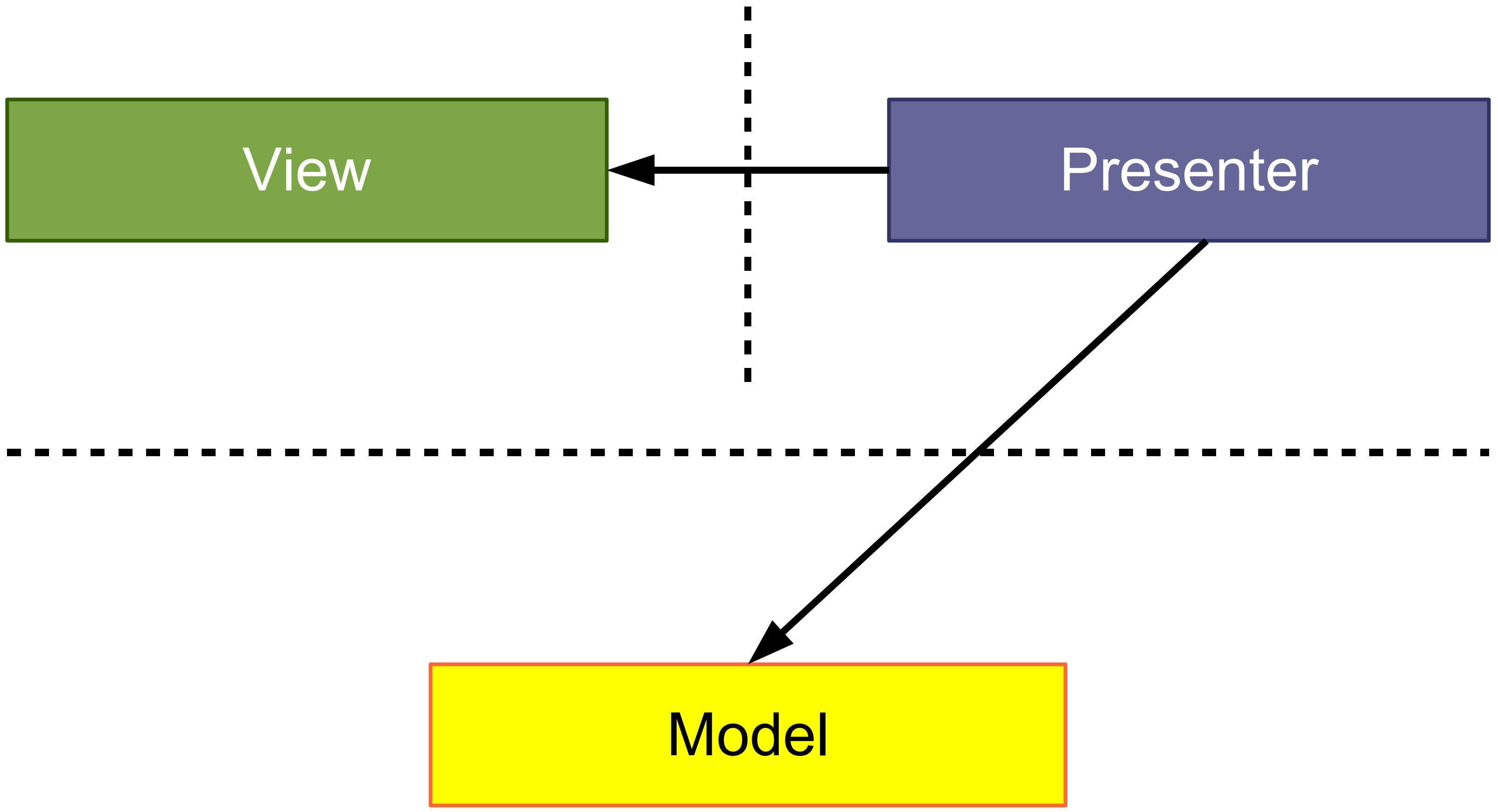
Model View Controller

Model View Presenter

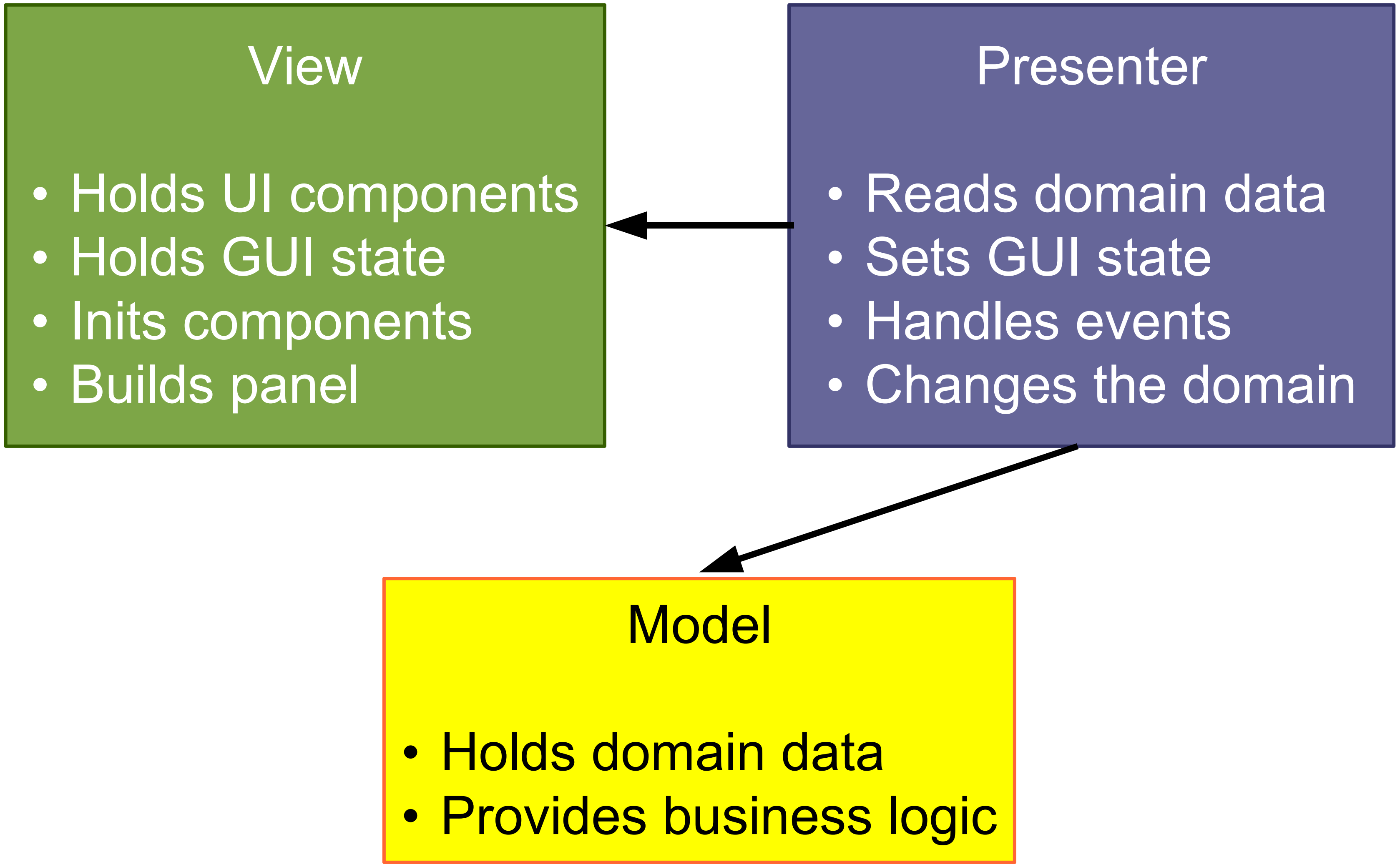
Presentation Model

Data Binding

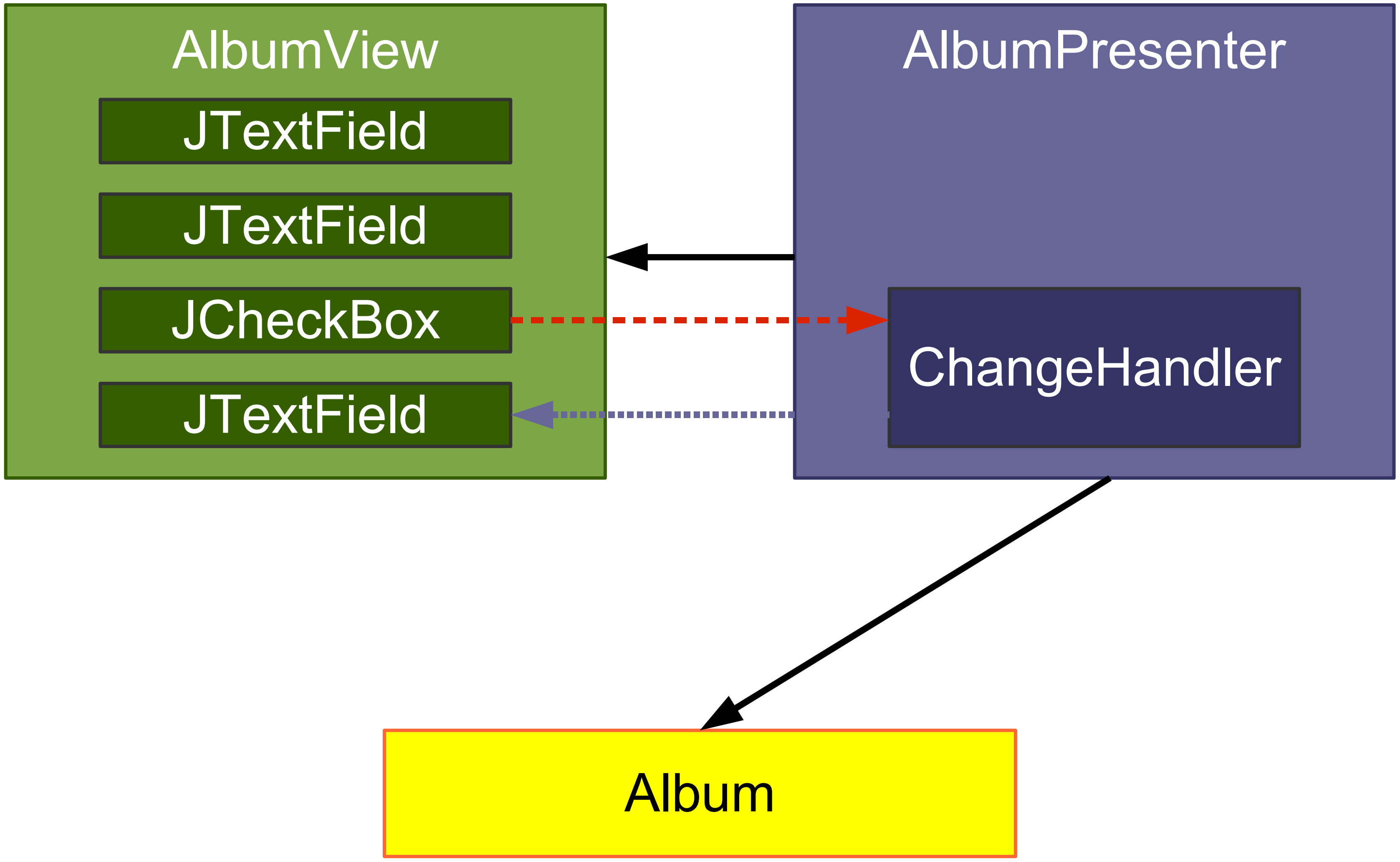
Pattern: Model View Presenter (MVP)



Model View Presenter



Model View Presenter



From 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 ...
}
```

...to Model View Presenter

```
class AlbumView extends JDialog {
   private 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 ...
}
```

...to Model View Presenter

```
class AlbumView extends JDialog {
   private 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: Example Logic

```
class ClassicalChangeHandler
                    implements ChangeListener {

  public void stateChanged(ChangeEvent e) {
    // Check the view's classical state.
    boolean classical =
             view.classicalBox.isSelected();

    // Update the composer field enablement.
    view.composerField.setEnabled(classical);
  }

}
```

Agenda

Introduction

Autonomous View

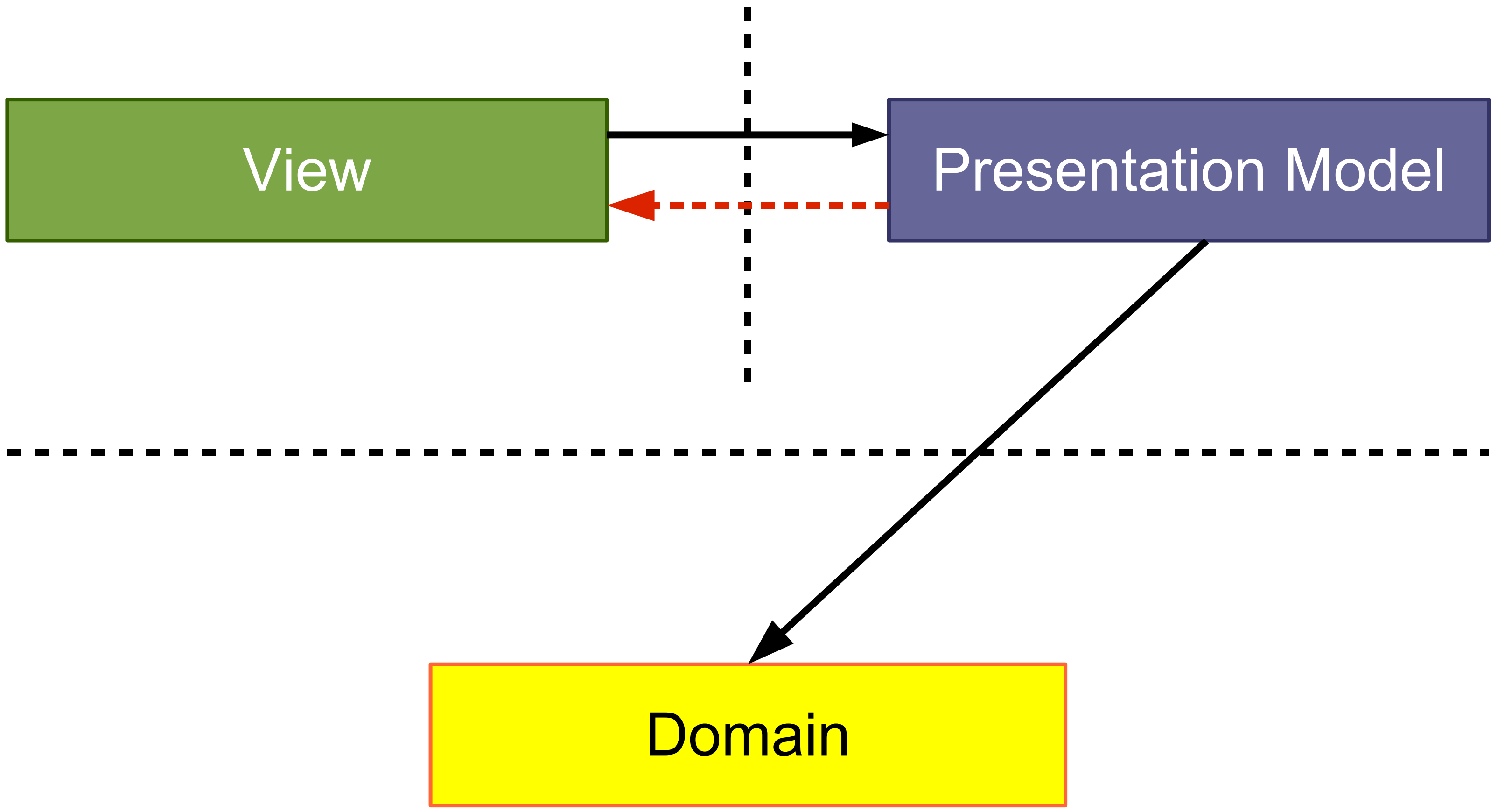
Model View Controller

Model View Presenter

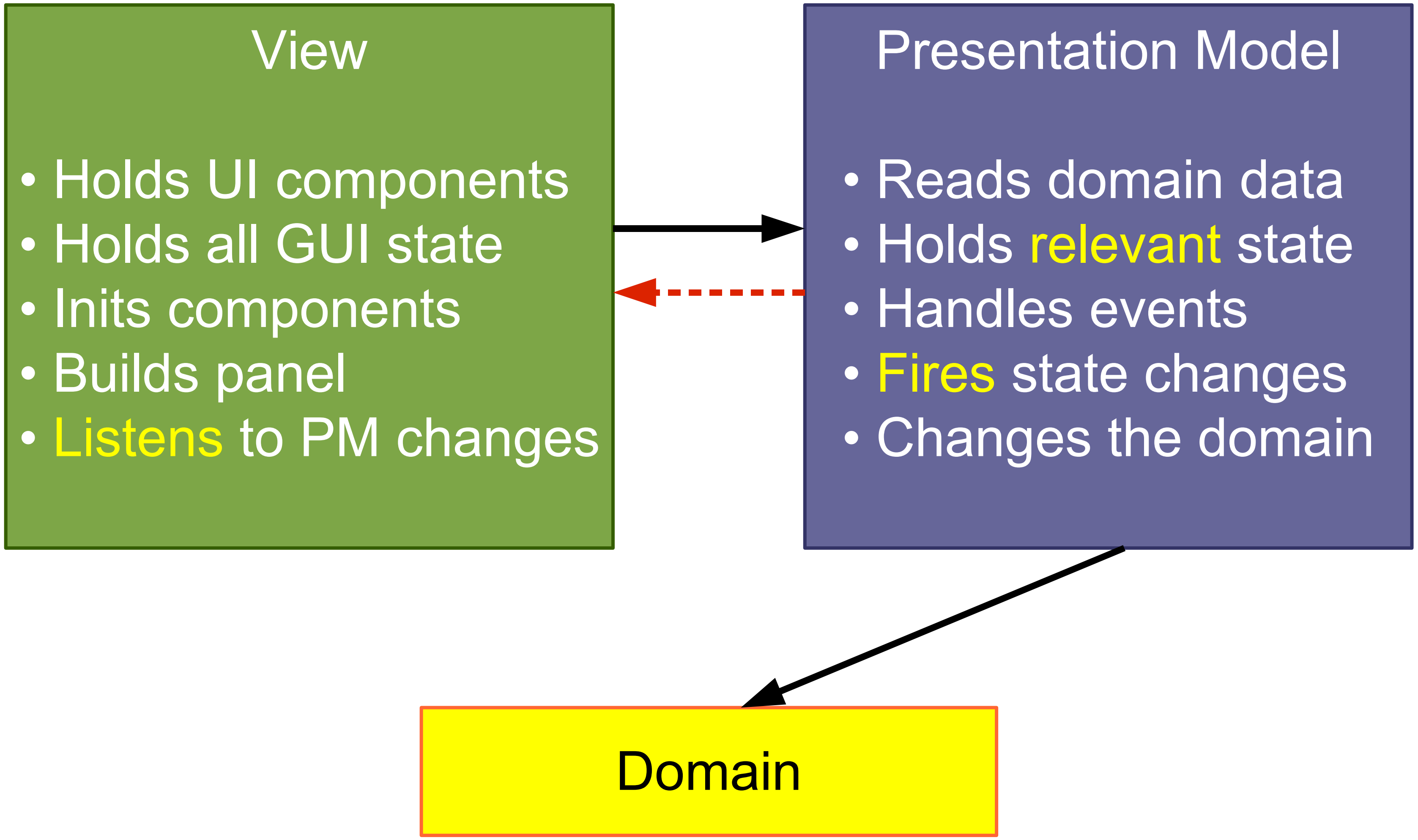
Presentation Model

Data Binding

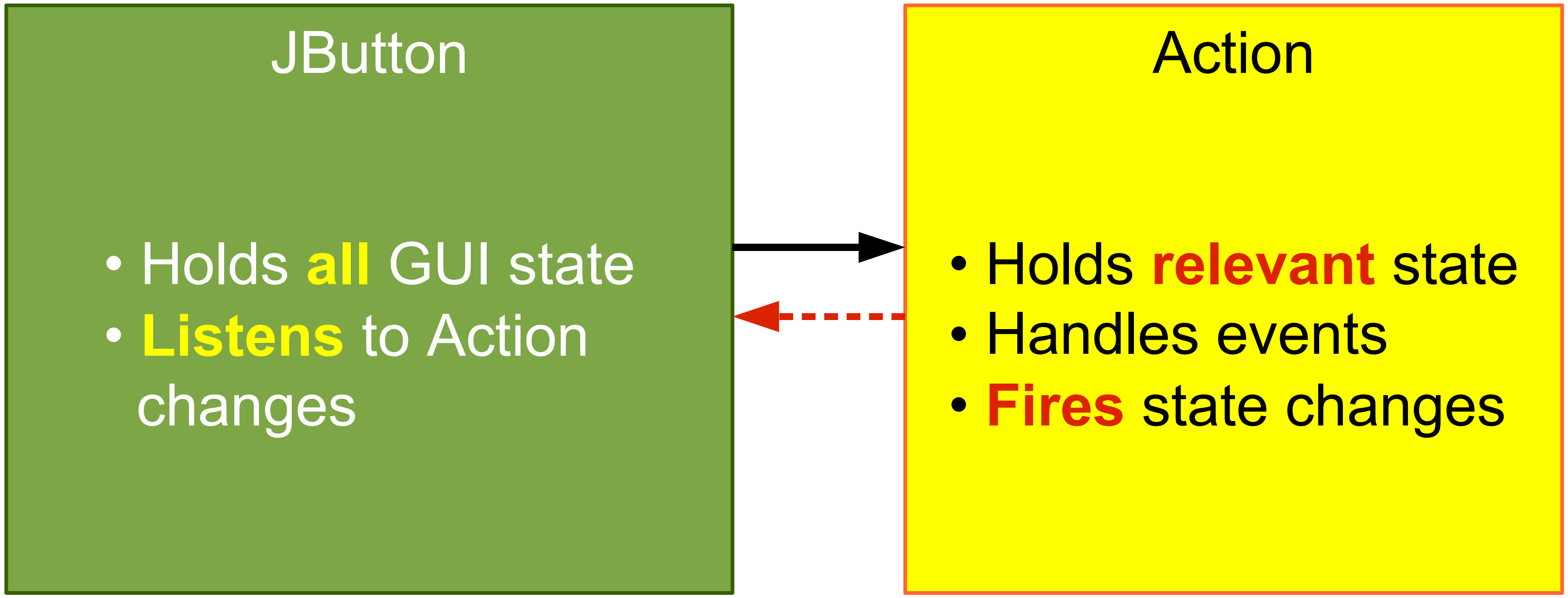
Pattern: Presentation Model



Presentation Model



Reminder: Swing Actions



JButton

- Holds **all** GUI state
- **Listens** to Action changes

Action

- Holds **relevant** state
- Handles events
- **Fires** state changes

From Autonomous View...

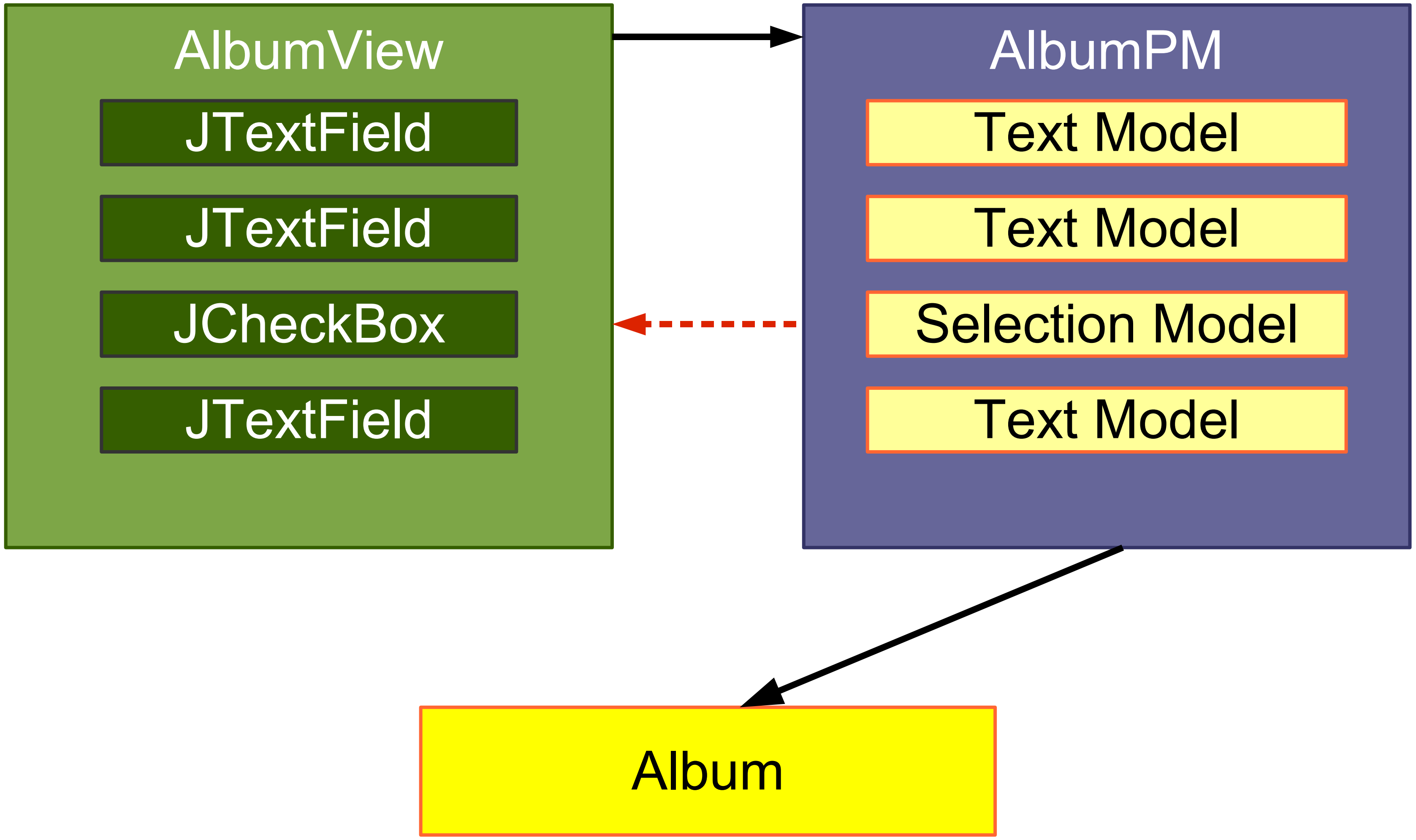
```
public class AlbumDialog extends JDialog {
   private JTextField artistField;
   public AlbumDialog(Album album) { ... }
   private void initComponents() { ... }
   private JComponent buildContent() { ... }

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

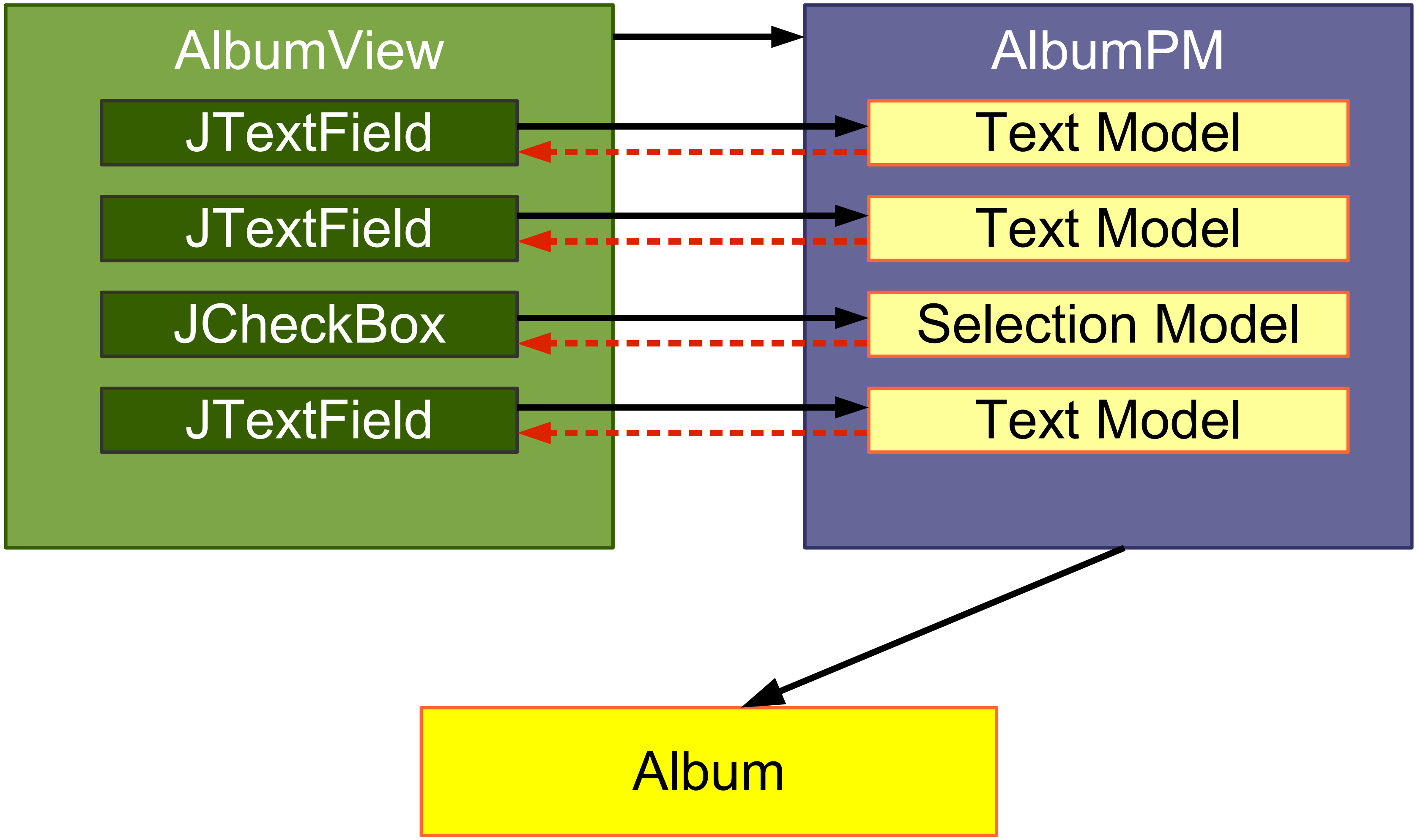
...to Presentation Model

```
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 ...
}
```

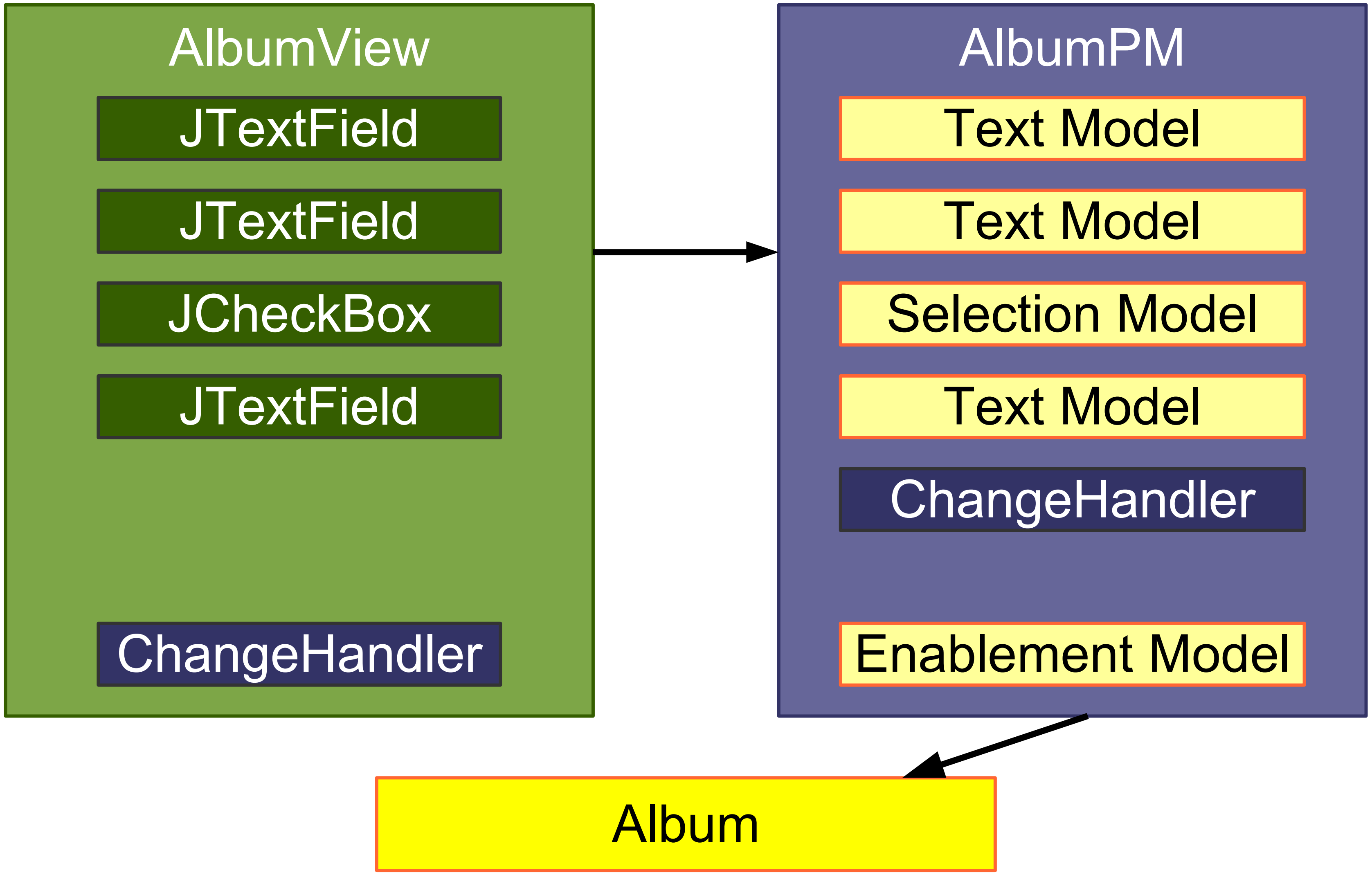
AlbumPresentationModel



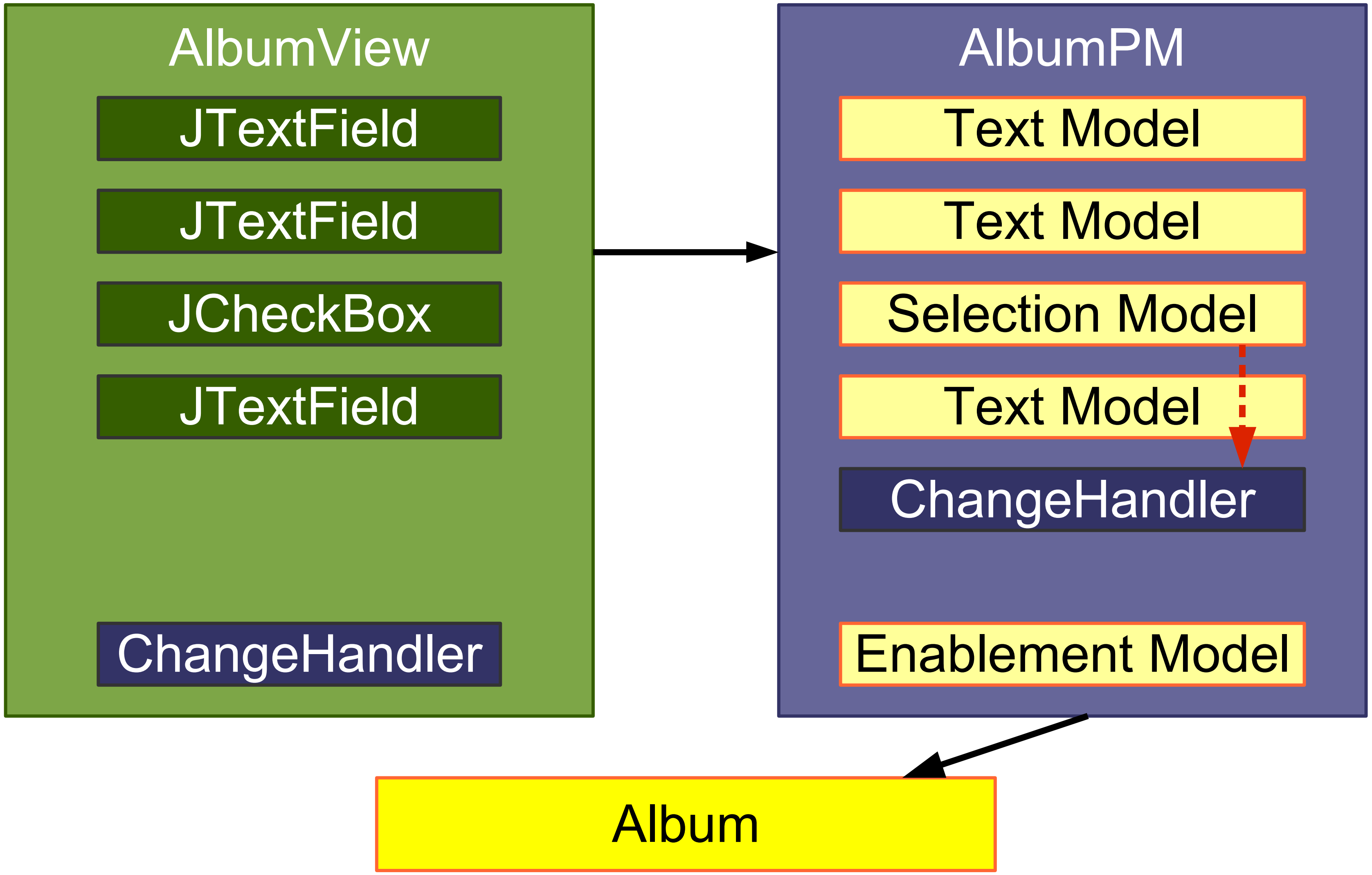
AlbumPresentationModel



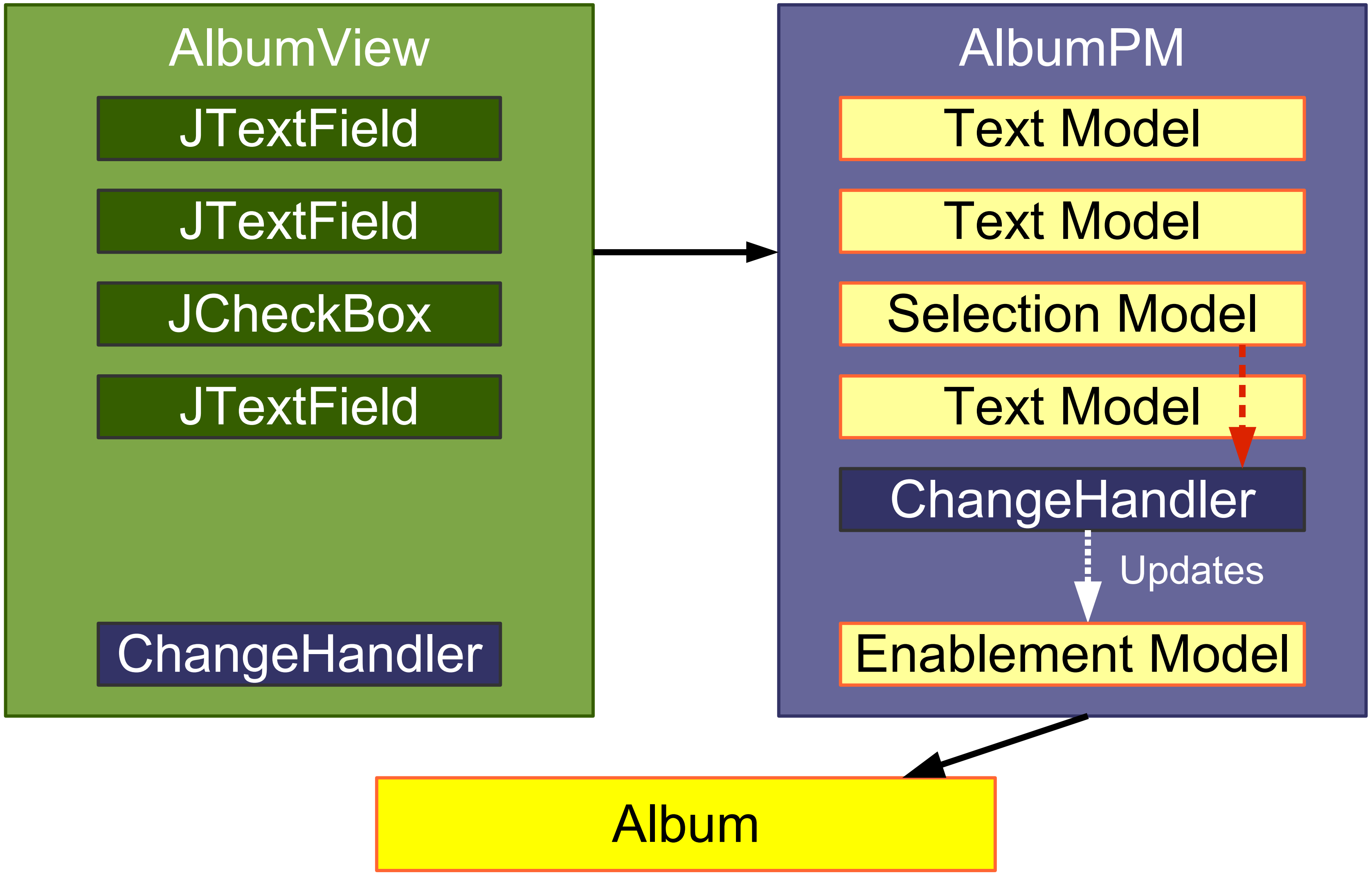
AlbumPresentationModel: Logic



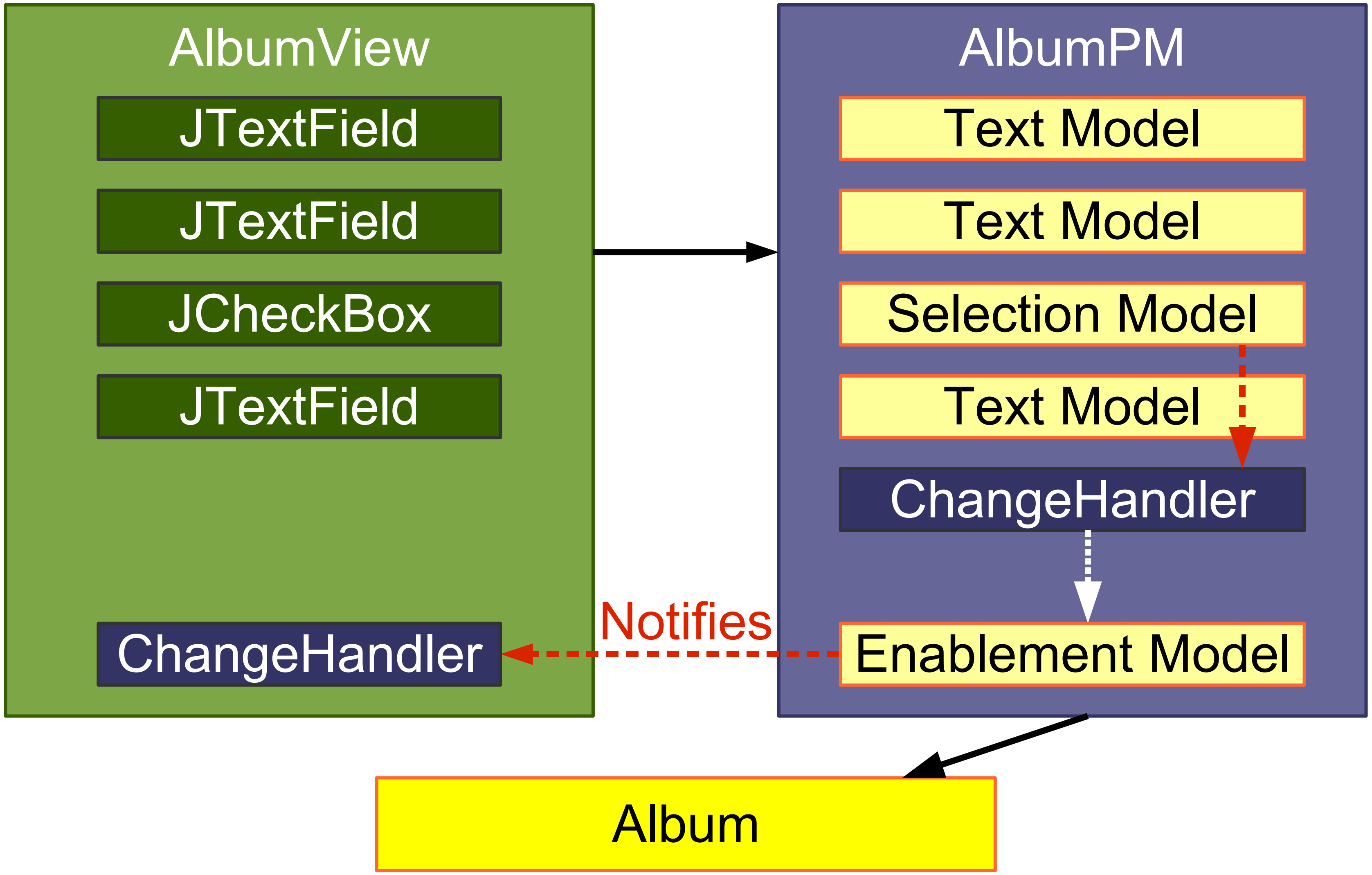
AlbumPresentationModel: Logic



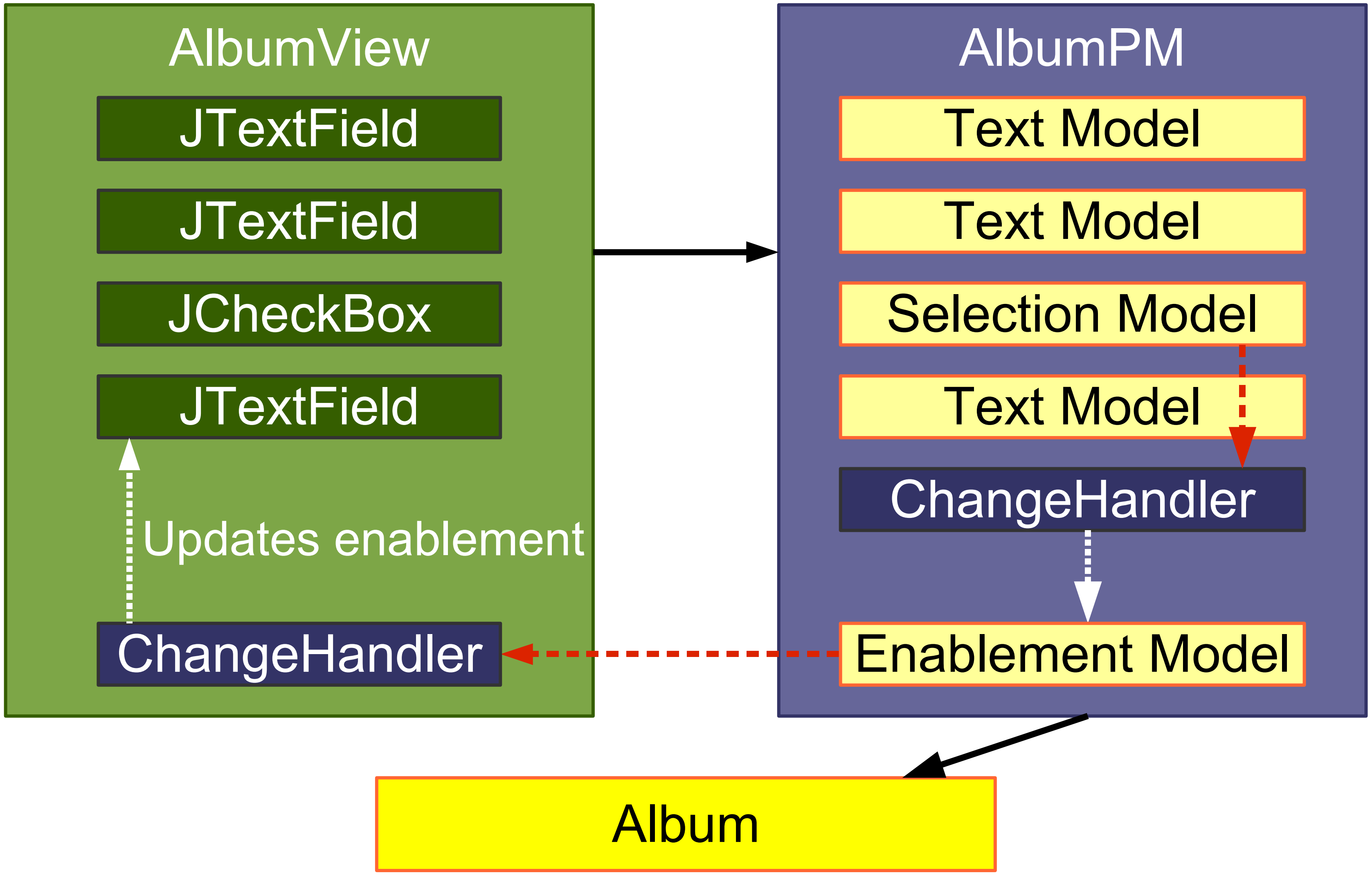
AlbumPresentationModel: Logic



AlbumPresentationModel: Logic



AlbumPresentationModel: Logic

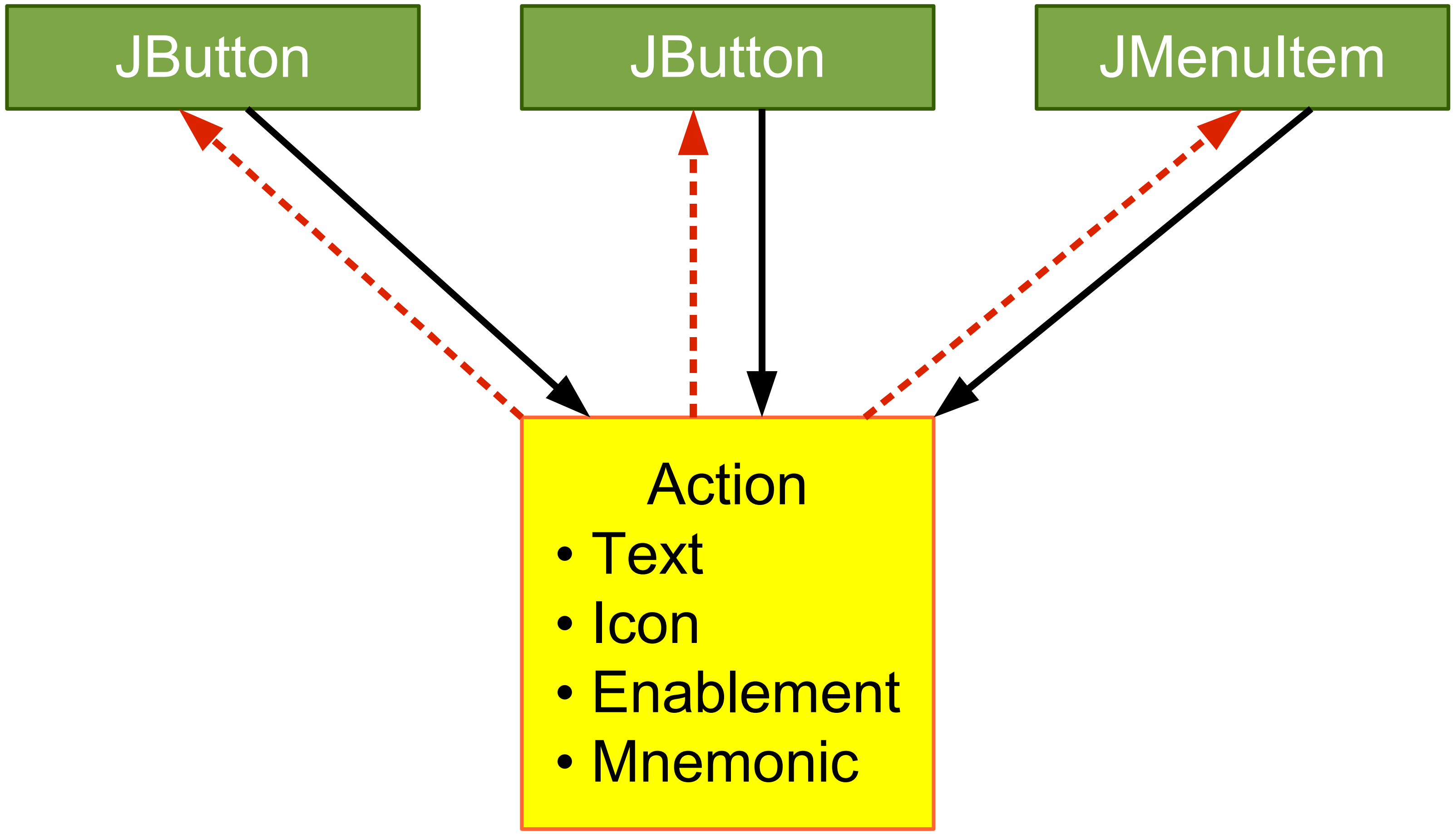


No Worries: Actions Again

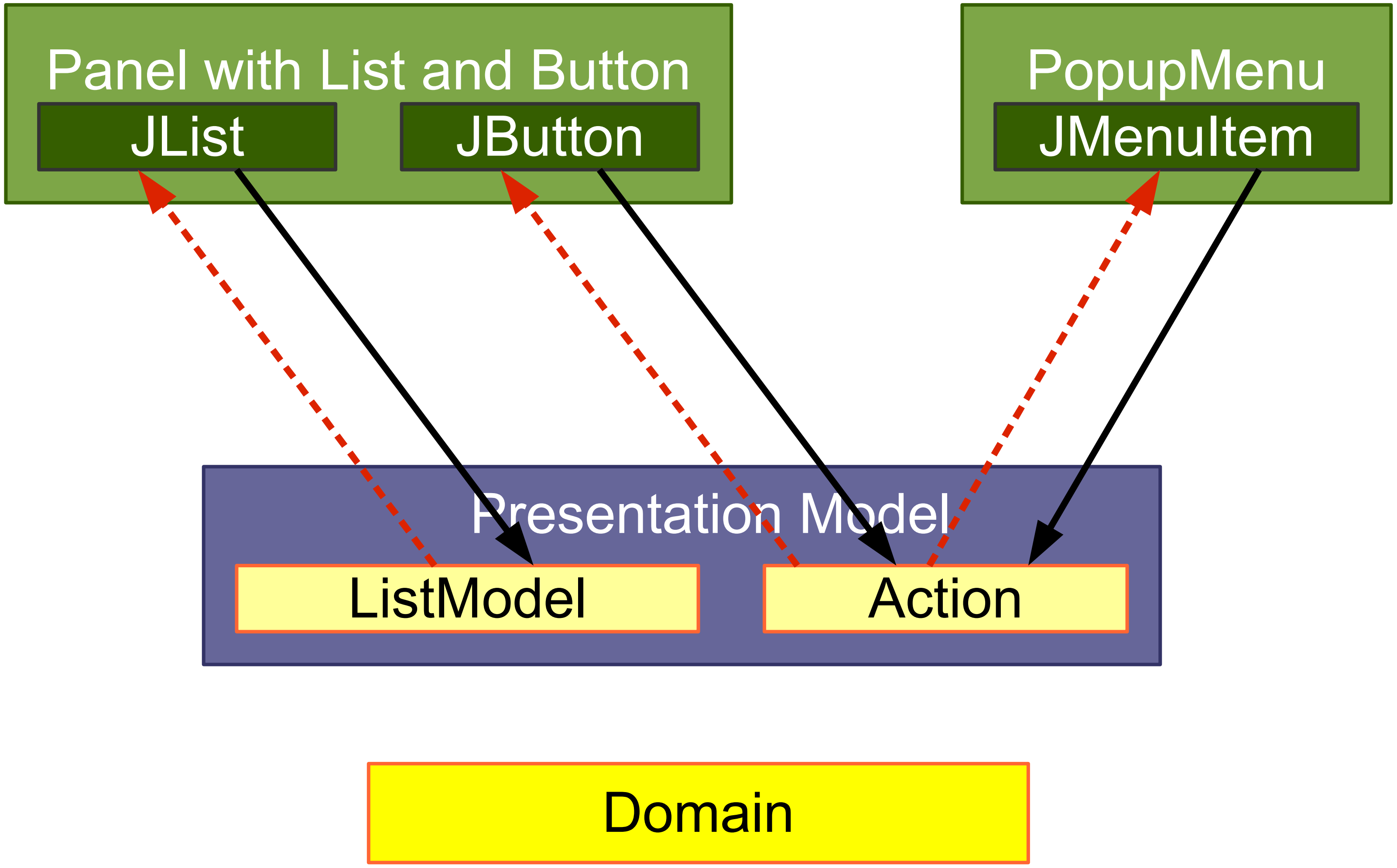
- Swing uses a similar machinery for Actions
- Actions fire PropertyChangeEvents
- JButton listens to the Action and updates its state
- Swing synchronizes Action state and GUI state
- All **you** need to write is:

```
new JButton(anAction)
```

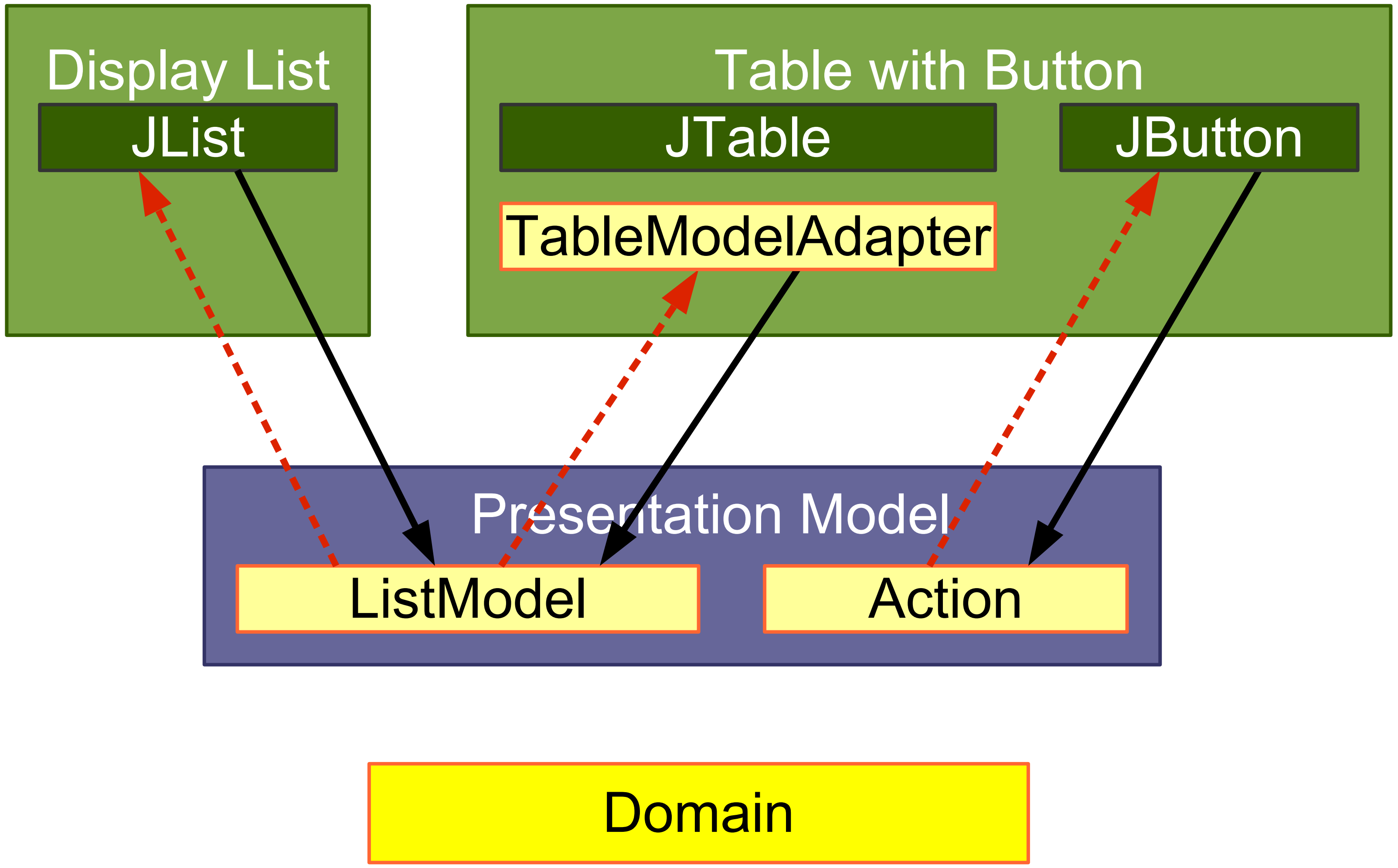
Action with Multiple Views



Presentation Model: Multiple Views I



Presentation Model: Multiple Views II



MVP vs. Presentation Model: GUI State

- MVP
  - View holds **the** GUI state
  - Presenter holds **no** state
  - Avoids having to synchronize copied GUI state
- Presentation Model
  - View holds **all** GUI state
  - PM holds the **relevant** GUI state
  - Must synchronize PM state and View state

MVP vs. Presentation Model: Testing

- MVP
  - Allows to test the Presenter with a View stub
  - Allows to preview the View without the Presenter
- Presentation Model
  - Allows to test the Presentation Model without the View
  - Allows to preview the View with a PM stub

MVP vs. Presentation Model: Transformation Differences

- Some Autonomous Views use low-level GUI state
- Presenter can keep **dirty** low-level ops
  - Split to MVP is easier to do
  - Split to MVP may costs less
- Split to PM may require extra work
  - Find and add GUI state abstractions
  - Add handlers to the view
- You may benefit from the extra cleaning

MVP vs. Presentation Model: General

- Developers are used to operate on view state
- Presenter depends on GUI component types
- MVP addresses problems many faced with PM

Agenda

Introduction

Autonomous View

Model View Controller

Model View Presenter

Presentation Model

Data Binding

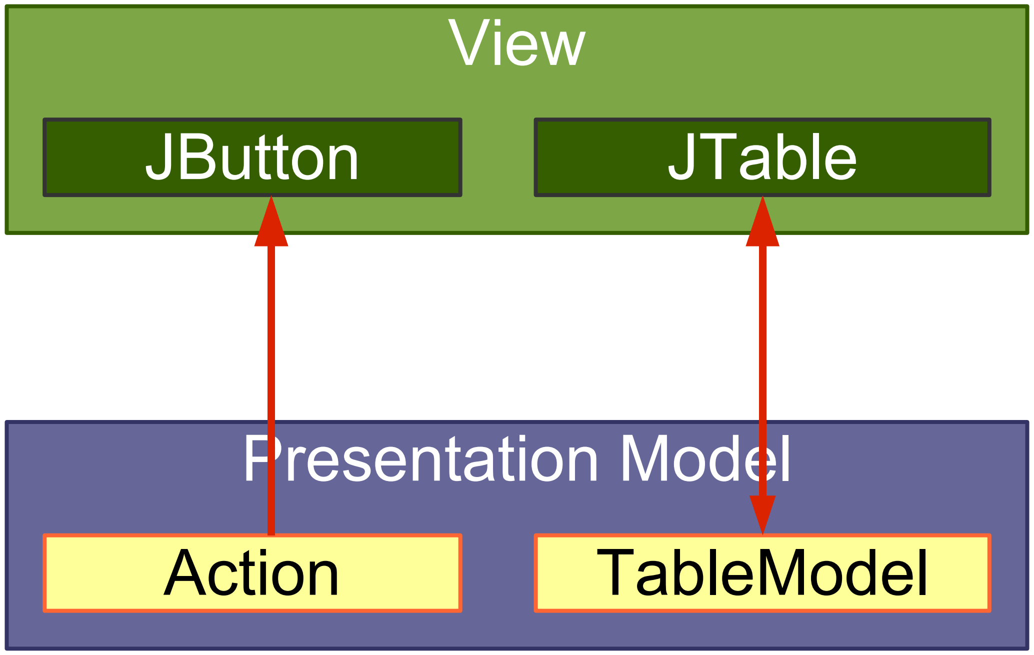
Data Binding

- Synchronizes two data sources
- One-way or two-way

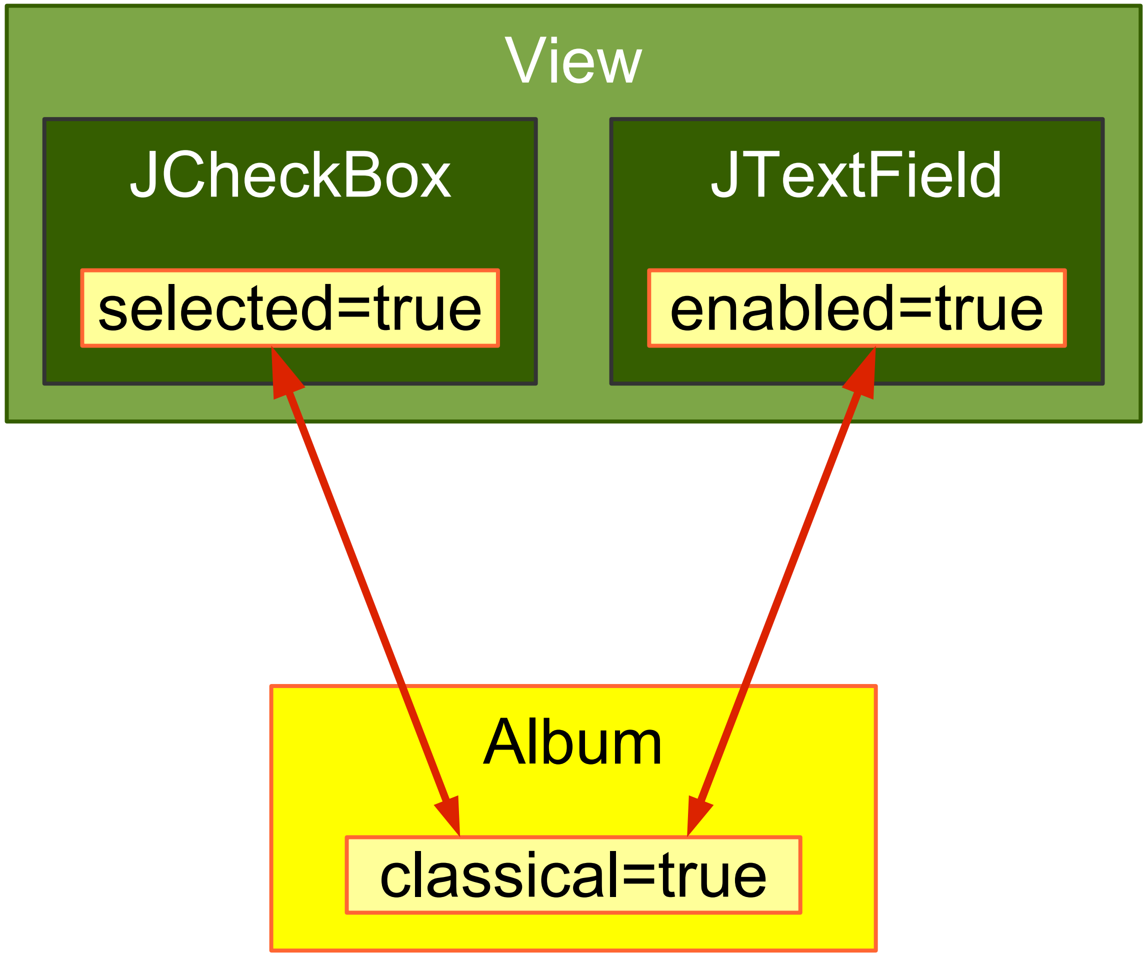
Binding Examples

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

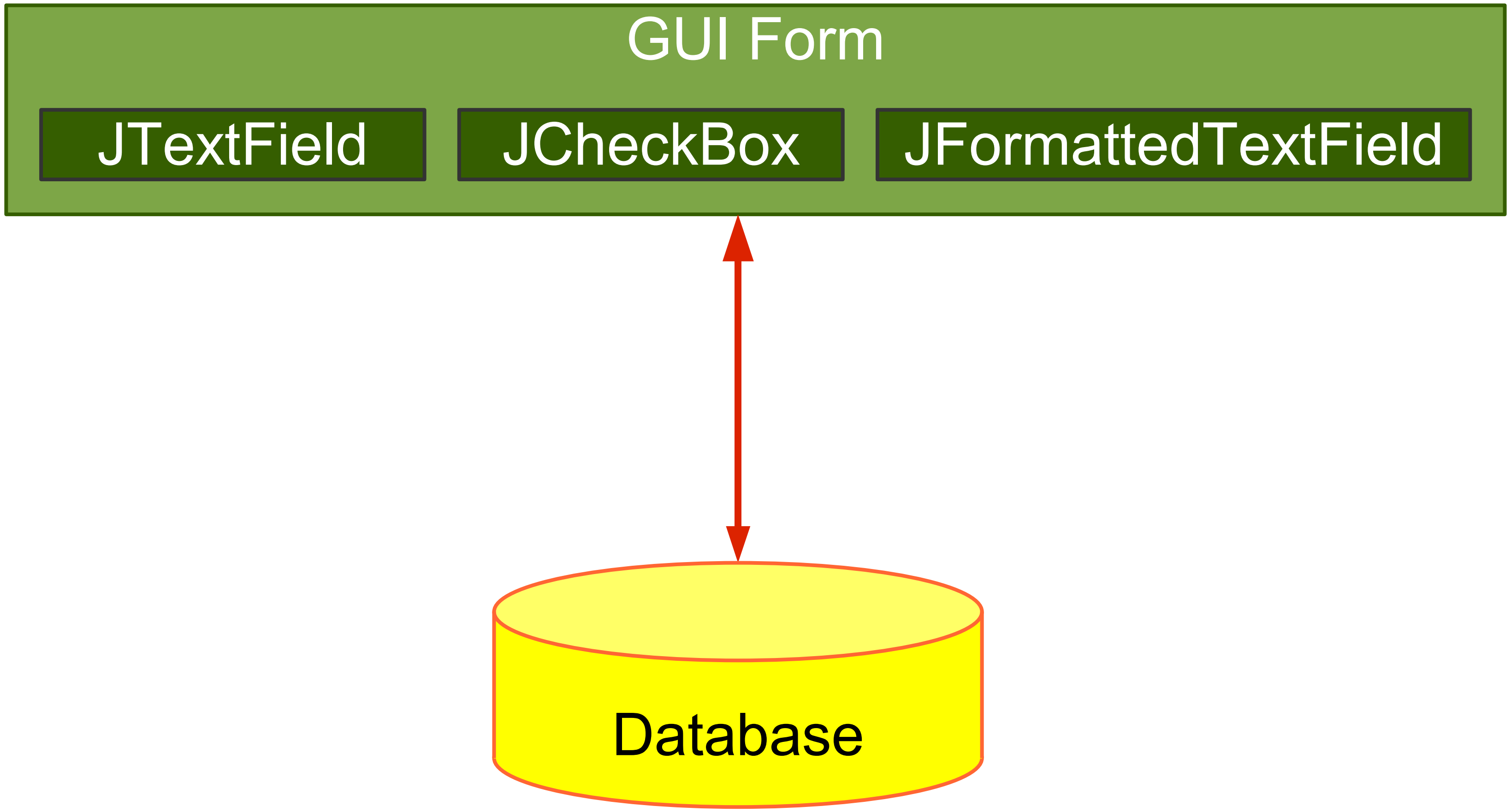
Binding Levels



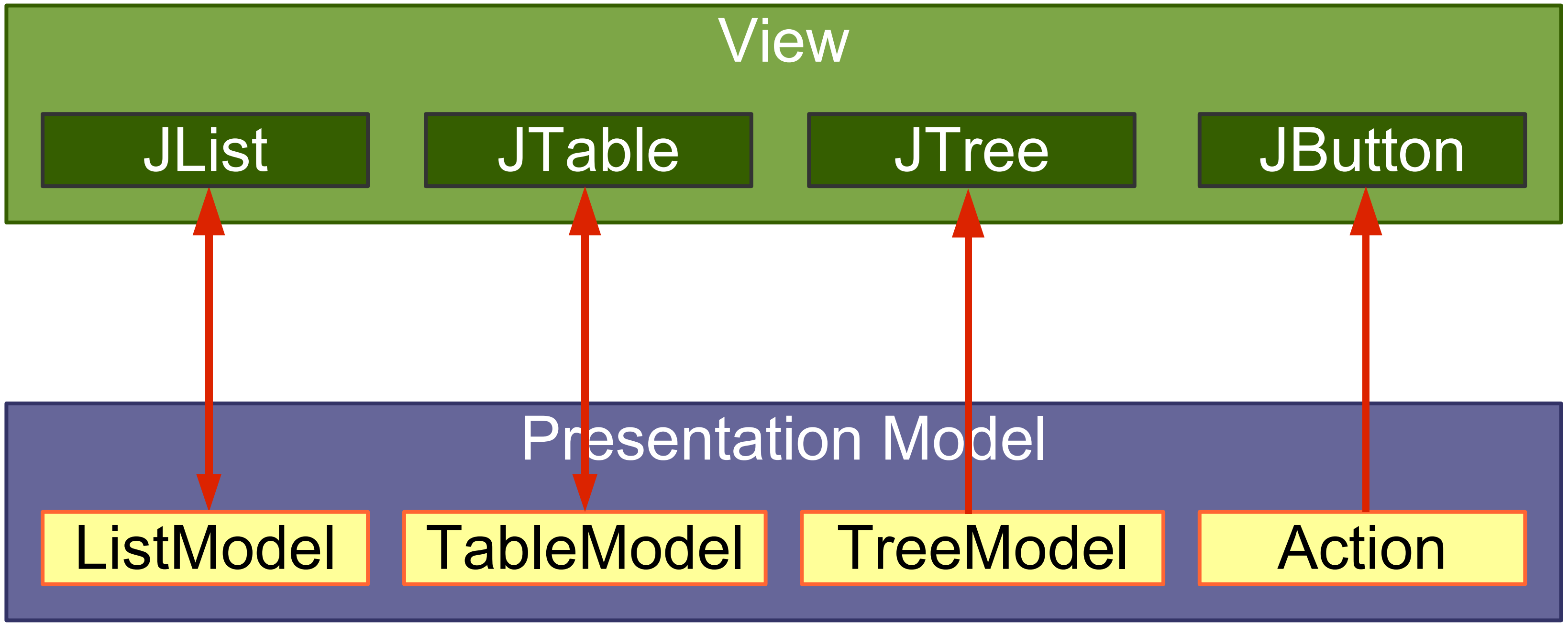
Binding Levels



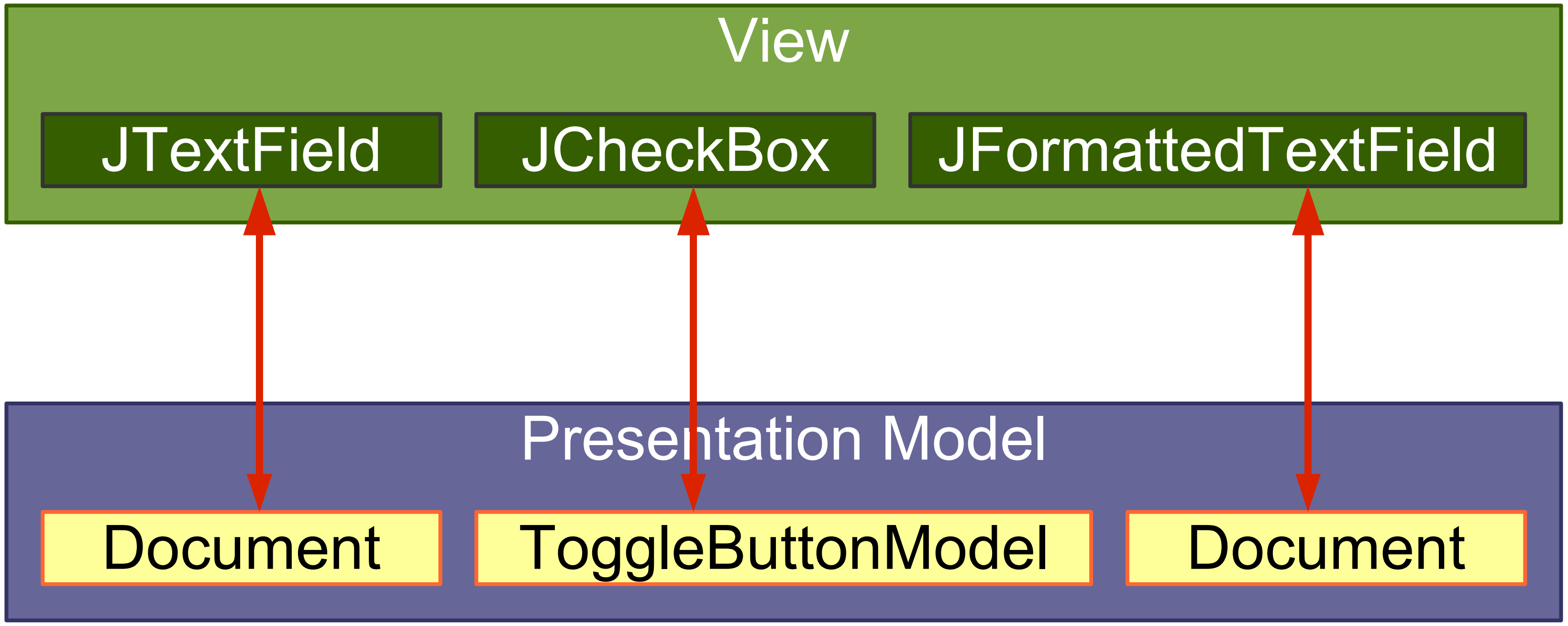
Binding Levels



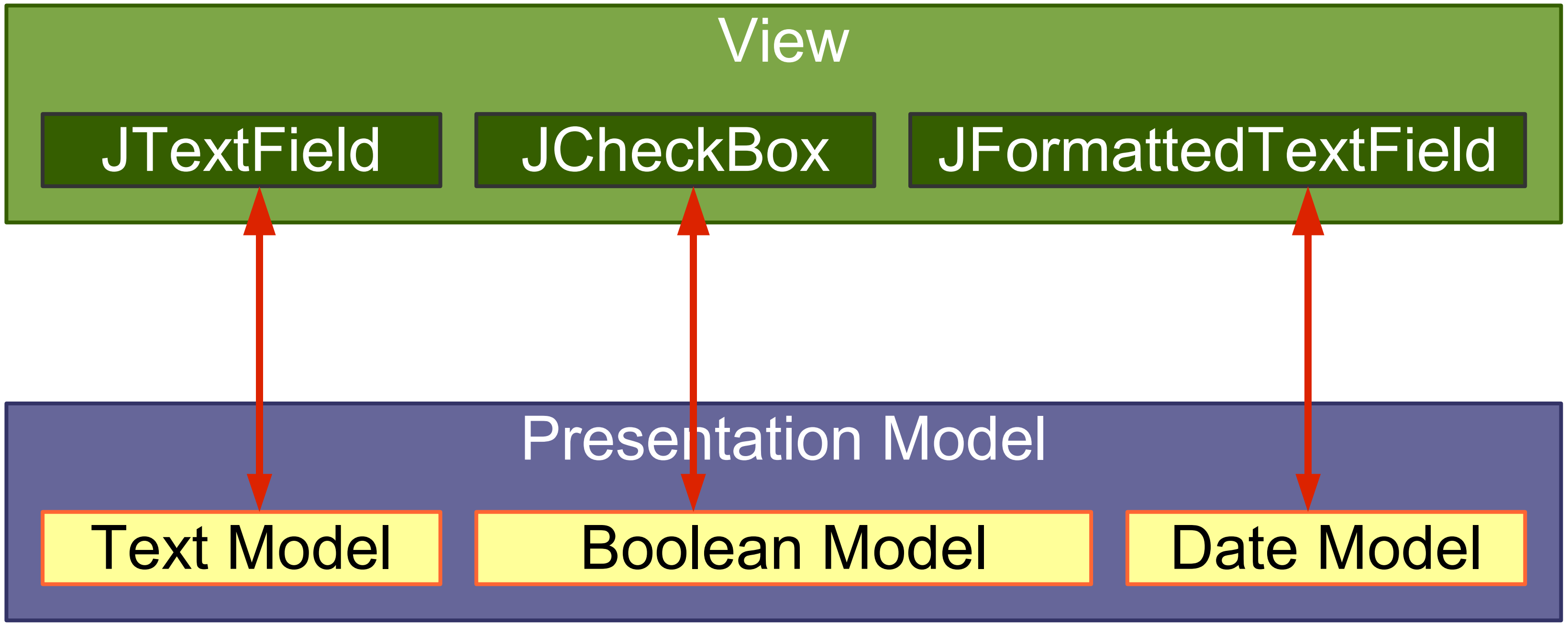
Useful Swing Bindings



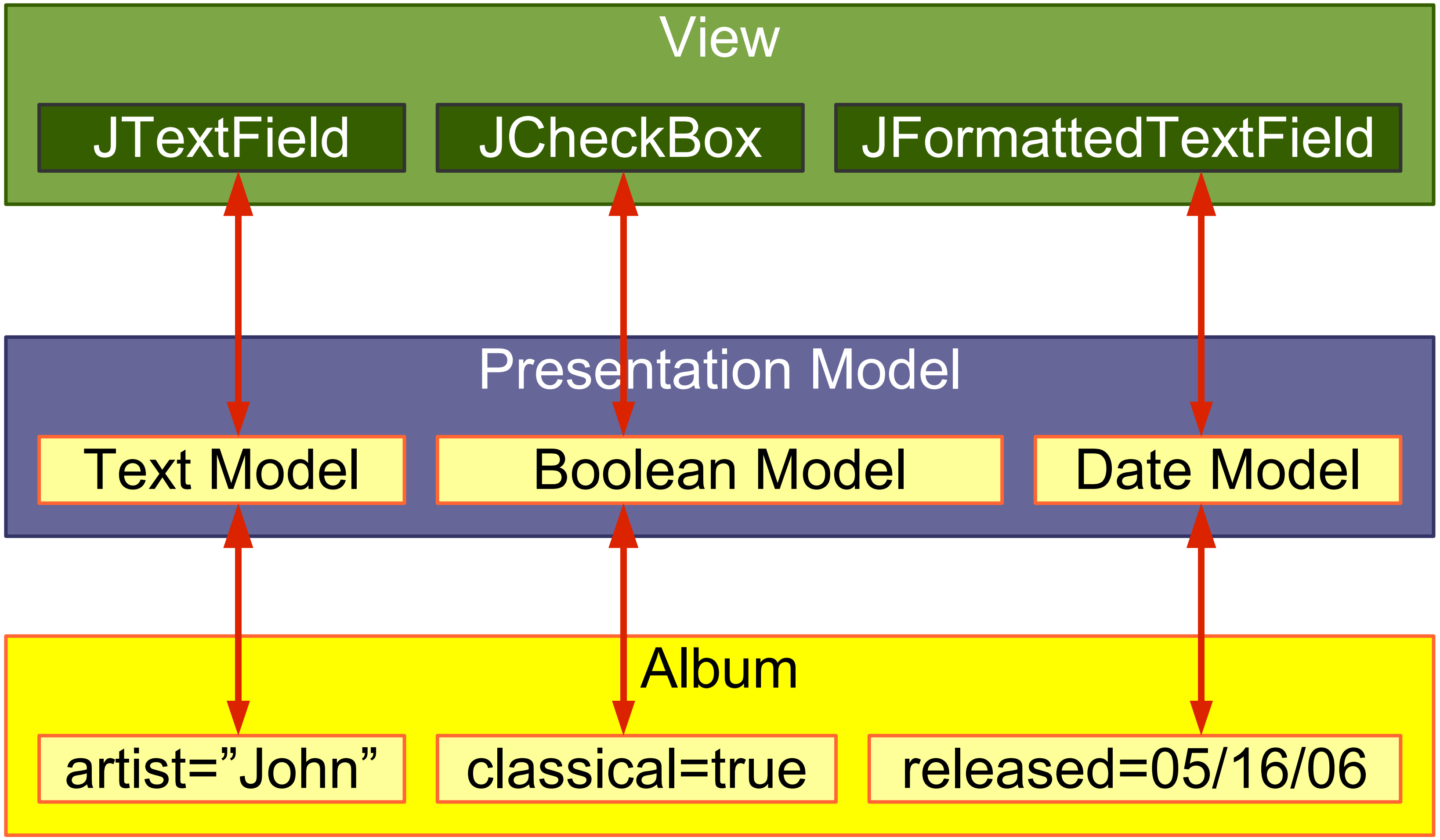
Swing Binding to Low-Level Models



Wanted: Higher-Level Binding



Wanted: Full Binding Path



JGoodies Binding

- Uses Swing bindings:
  - JList, JTable, JComboBox, JTree, JButton
- Fills the gap where Swing uses low-level models:
  - JTextField, JCheckBox,…
- Converts Bean properties to a uniform model (ValueModel)
- Makes the hard stuff possible
- Makes simple things a bit easier

AlbumView: Init and Bind Components

```
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() {

   // Synchronize field enablement
   // with the PresentationModel state.
   PropertyConnector.connect(
      presentationModel,
      “composerEnabled”,
      composerField,
      “enabled”);

}
```

Copying…

- Easy to understand
- Works in almost all situations
- Easy to debug; all data operations are explicit
- Difficult to synchronize views
- Needs discipline in a team
- Coarse-grained updates
- ~~Leads to a lot of boilerplate code~~

…vs. Automatic Binding

- Fine-grained updates
- Simplifies synchronization
- Harder to understand and debug
- Extra work for method renaming and obfuscators

Costs for Automatic Binding

- Increases **learning costs**
- Decreases **production costs** a little
- Can significantly reduce the **change costs**

Summary

- Starting point: **Separated Presentation**
- Common and workable: **Autonomous View**
- **MVP** works with view GUI state
- **PM** copies state and requires synchronization
- Swing has some **Presentation Model** support

Advice

- Use **Separated Presentation** whenever possible
- Split up **Autonomous Views** if appropriate
- Read Fowler’s **Organizing Presentation Logic**
- Use an automatic binding only if
  - It is reliable and flexible
  - **At least one expert** in the team masters it

For More Information

Sessions and BOFs

- TS-3399: A Simple Framework for Desktop Applications
- BOF-0381: Hop on the Swinging Event Bus!
- BOF-0461: The Spring Rich Client Project: Effective Desktop Application Architecture

For More Information

Web Resources

- Fowler's Further P of EAA–martinfowler.com/eaaDev
- SwingLabs data binding–databinding.dev.java.net
- Eclipse 3.2 data binding–www.eclipse.org
- Oracle ADF–otn.oracle.com, search JClient
- JGoodies Binding–binding.dev.java.net

  Binding tutorial contains Presentation Model examples

For More Information

Book

- *Scott Delap: Desktop Java Live*

Presentations-www.JGoodies.com/articles

- Desktop Patterns & Data Binding
- Swing Data Binding

Q&A

Desktop Patterns and Data Binding

Karsten Lentzsch

Founder
JGoodies
www.JGoodies.com

TS-1074