Swing Data Binding

Karsten Lentzsch
www.JGoodies.com

Presentation Goals

Understand MVC and Swing models.

Learn how to bind domain objects
to Swing UI components.

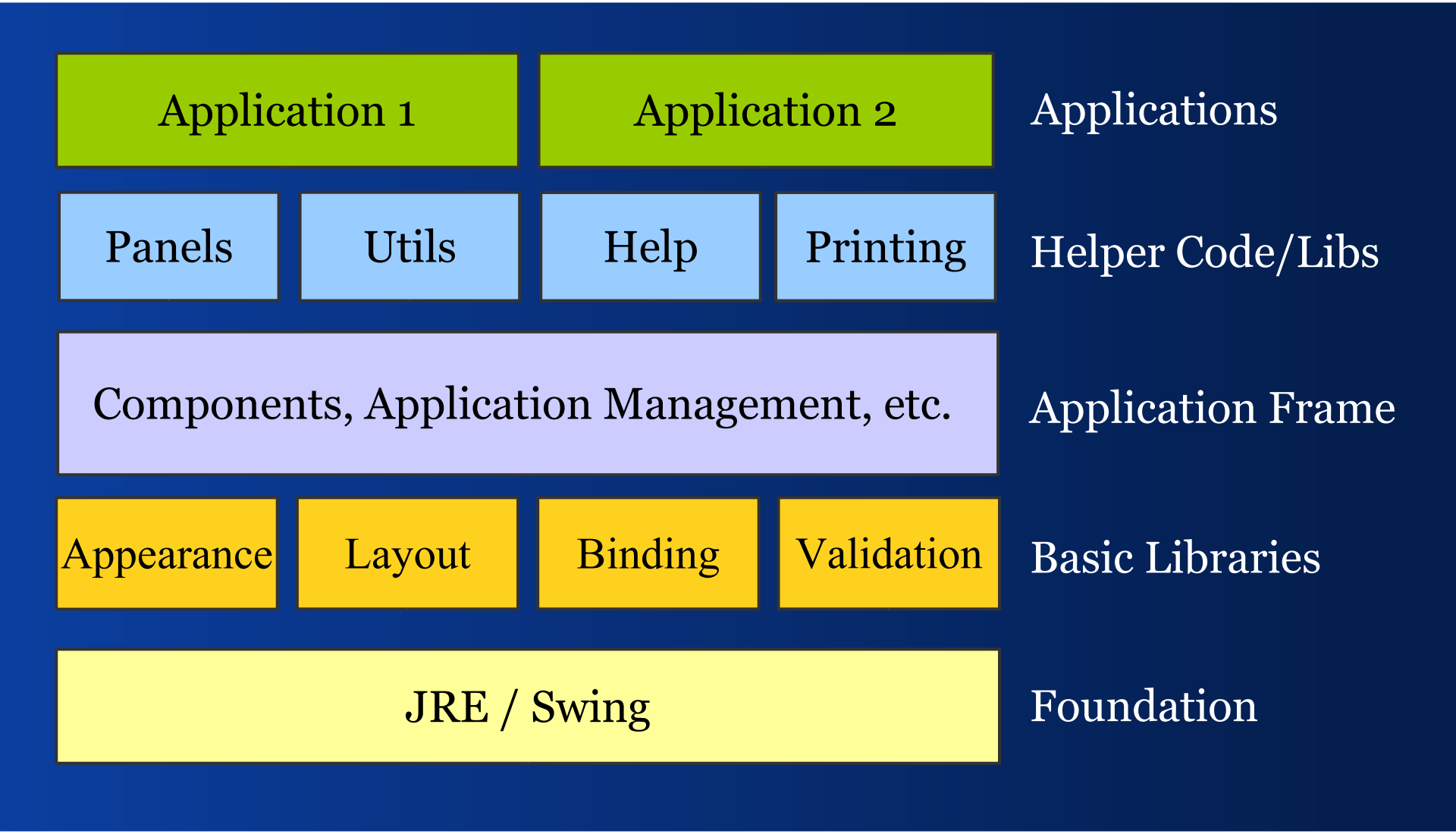
Speaker Qualifications

- Karsten builds elegant Swing apps
- works with Objects since 1990
- helps others with UI and architectures
- provides libraries that complement Swing
- provides examples for Swing architectures
- writes about Java desktop issues

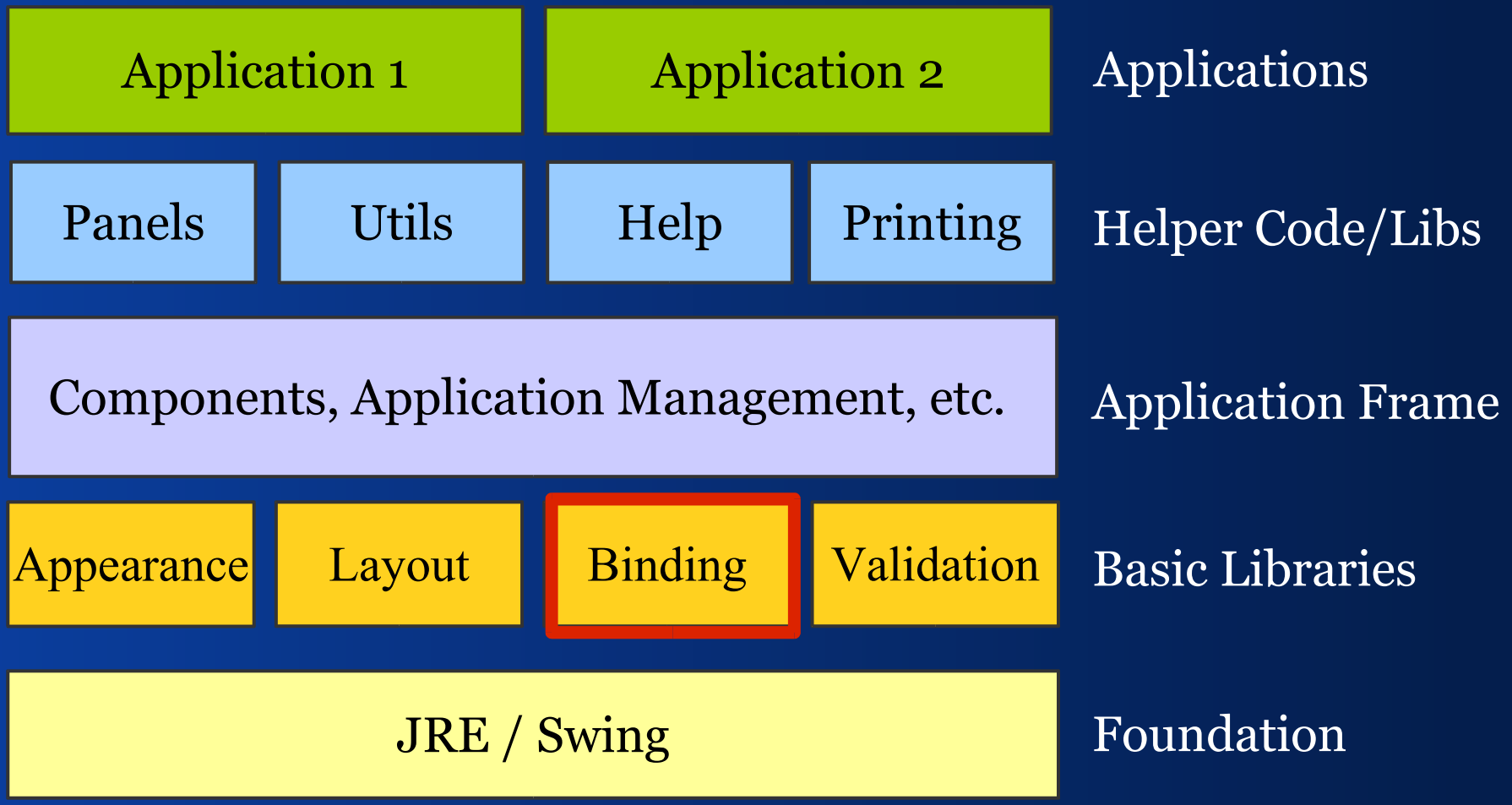
Agenda

- Introduction
- MVC and Swing
- How to bind single values?
- How to bind lists
- A 3-tier Swing architecture
- How binding works in projects

Swing Building Blocks



Swing Building Blocks



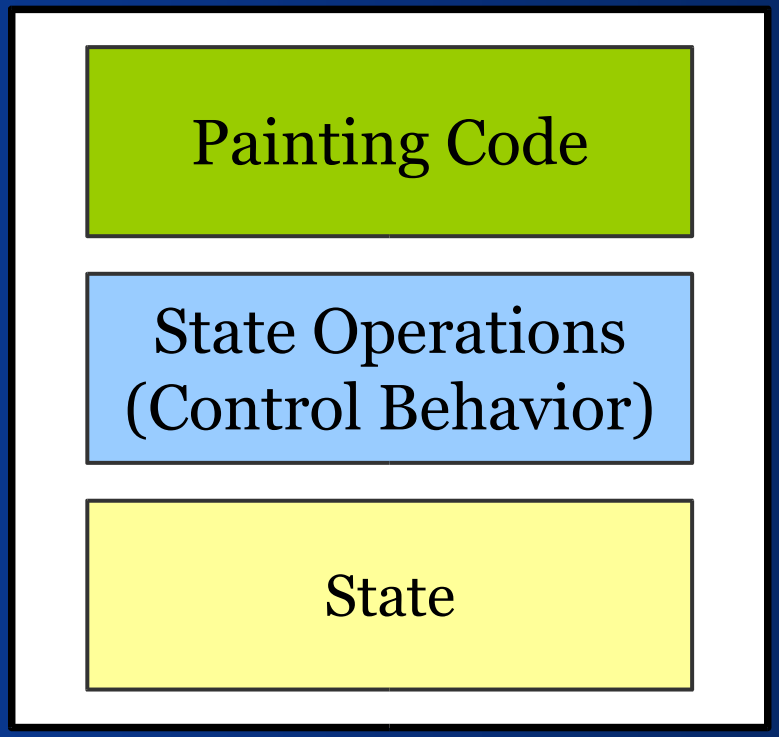
Questions

- Where do I find MVC in Swing?
- How to structure a Swing application?
- What is part of the model?
- How do I choose models?
- How to build a view?
- What does a controller do?
- Do I need controllers?

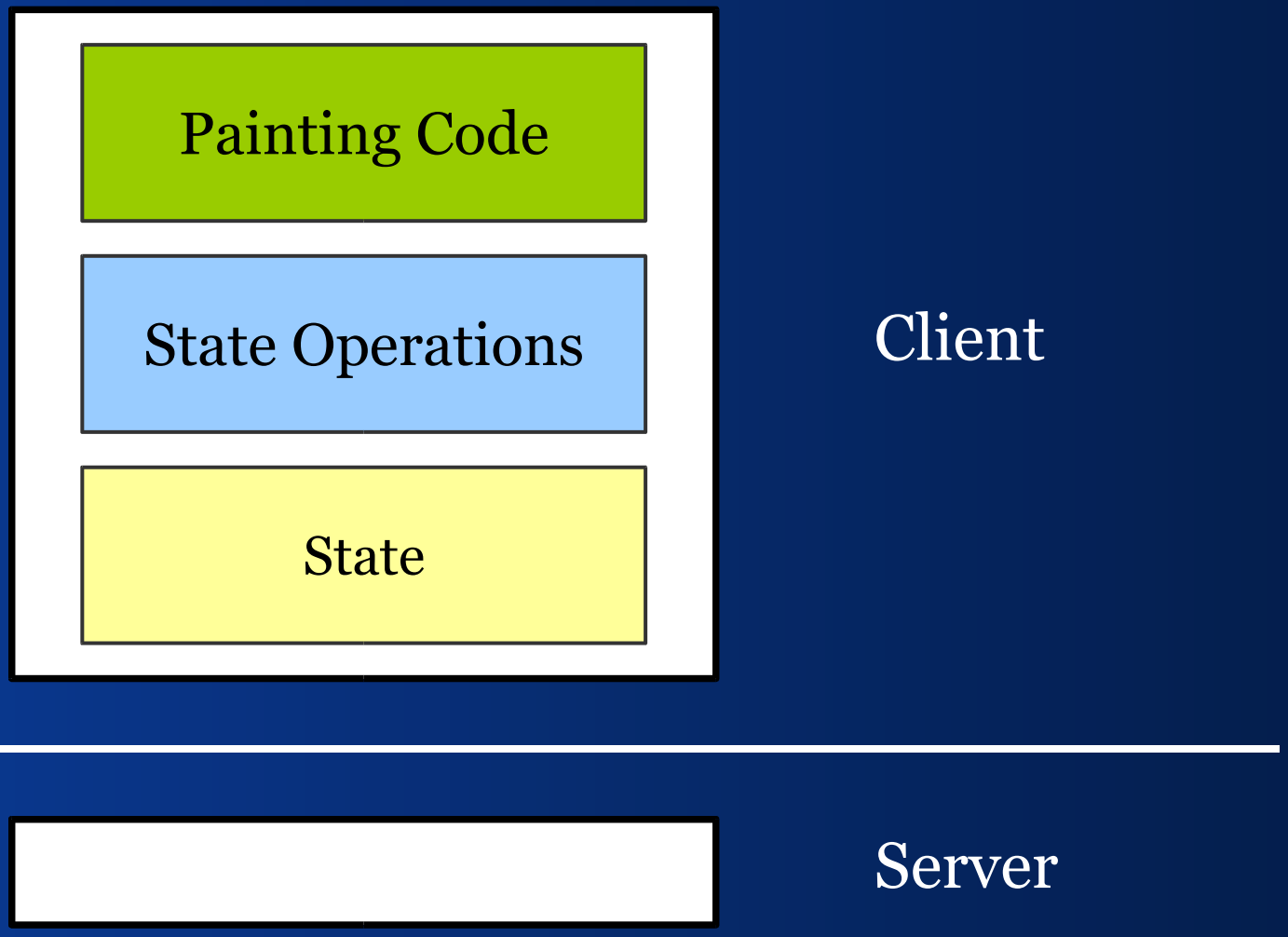
I - Basics

MVC and Swing

Before MVC



Before MVC: 2 Layers

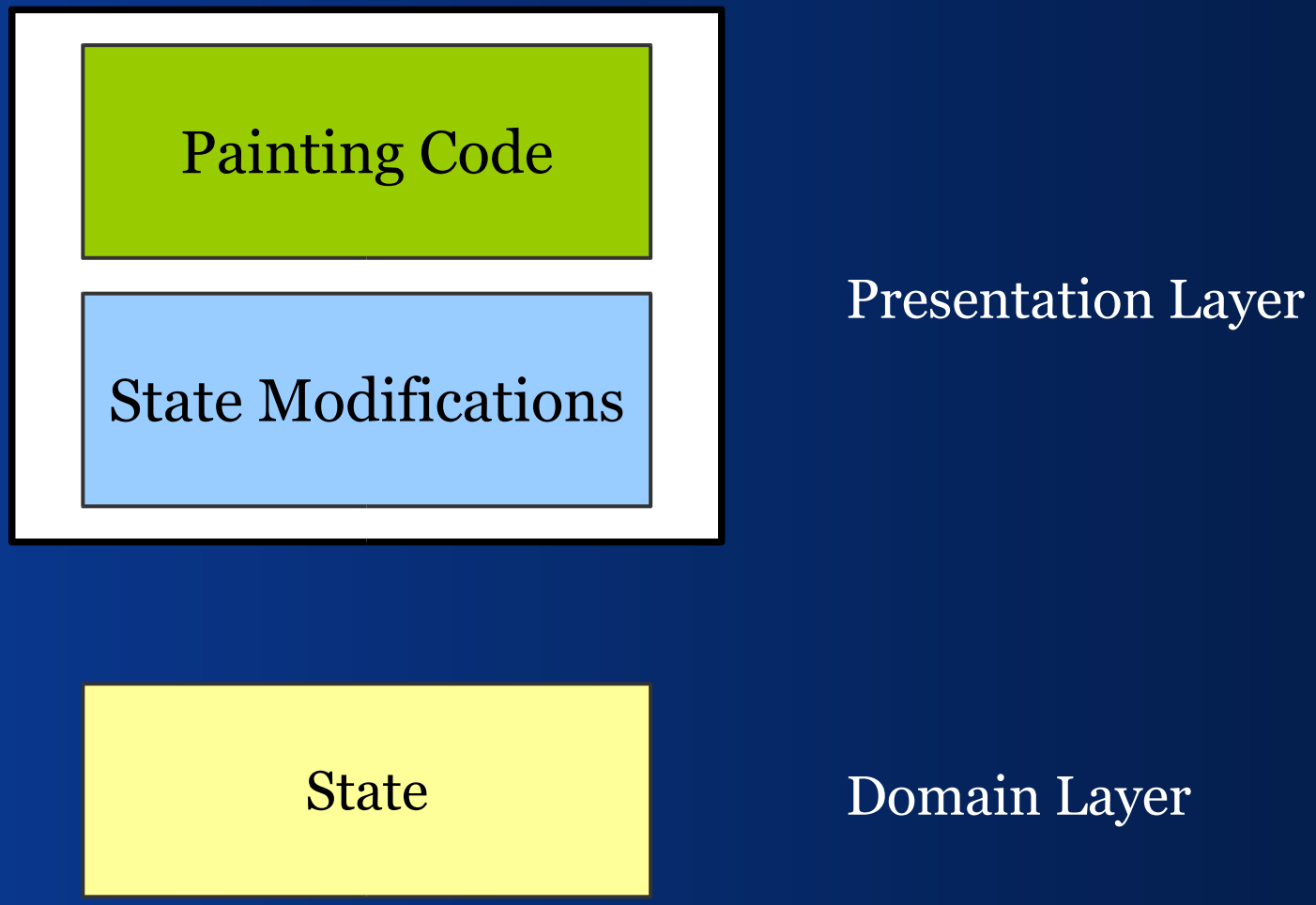


Separate Domain from Views

- Domain logic contains no GUI code
- Presentation handles all UI issues

- Advantages:
  - Each part is easier to understand
  - Each part is easier to change

Domain and Presentation

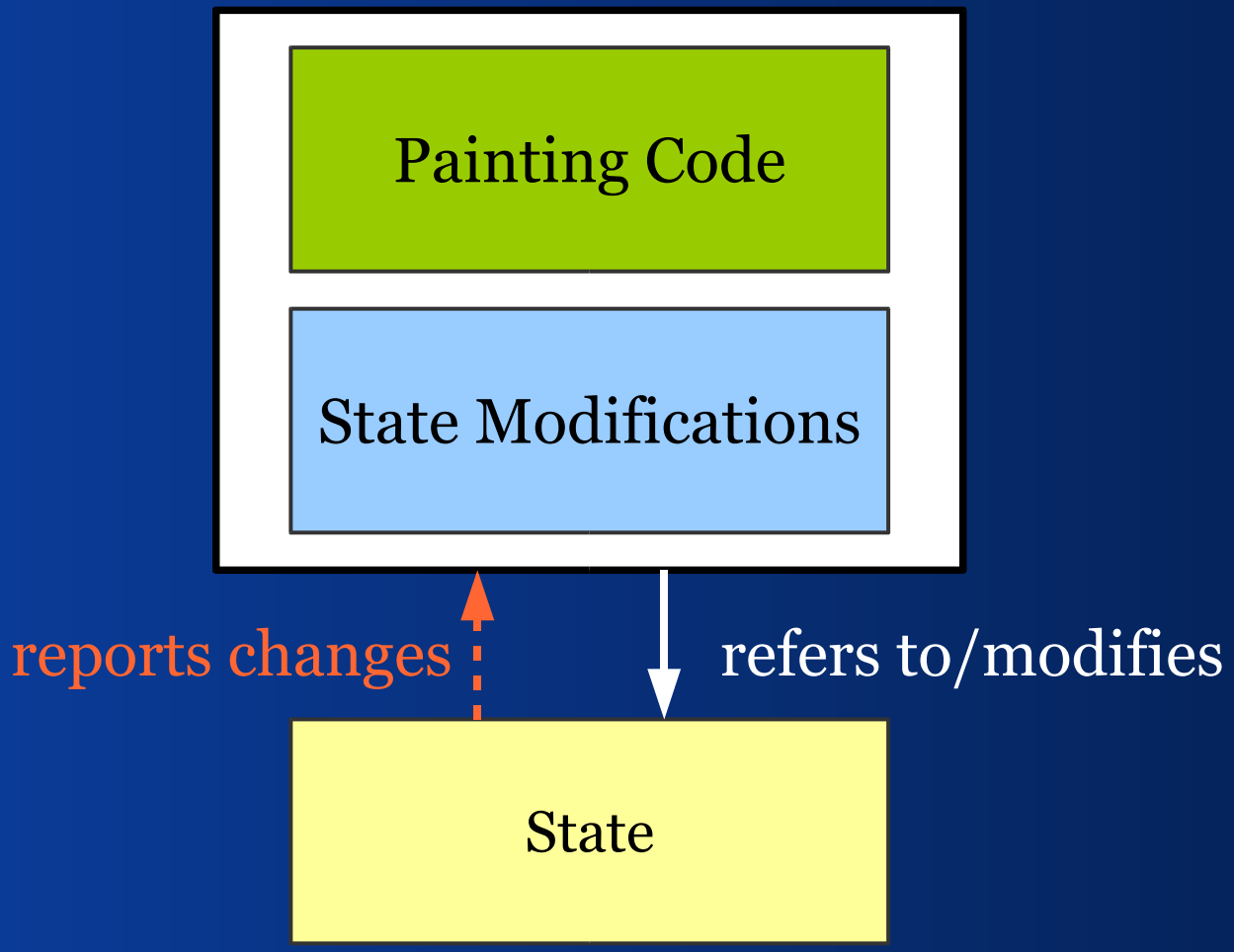


Loose Coupling

- The domain shall not reference the GUI
- Presentation refers to domain and can modify it

- Advantages:
  - Reduces complexity
  - Allows to build multiple presentations of a single domain object

Loose Coupling

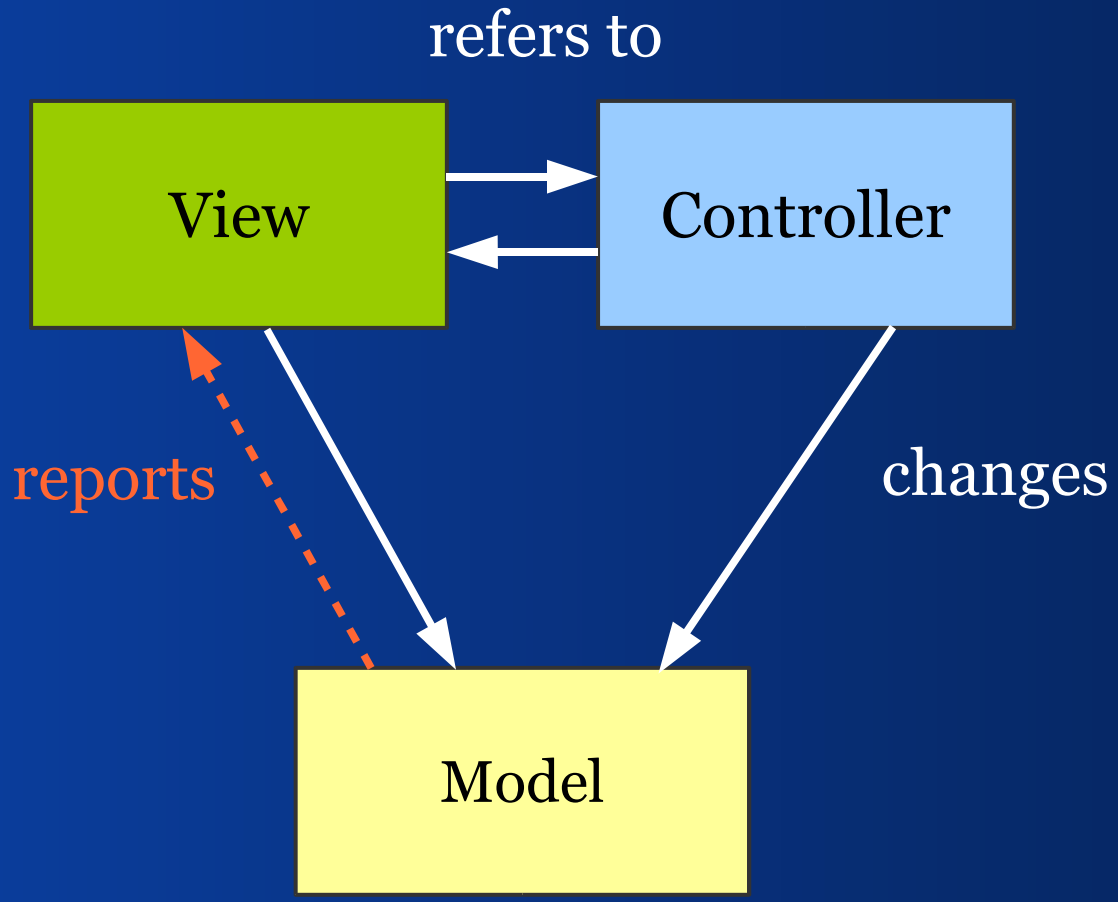


Separate View and Controller

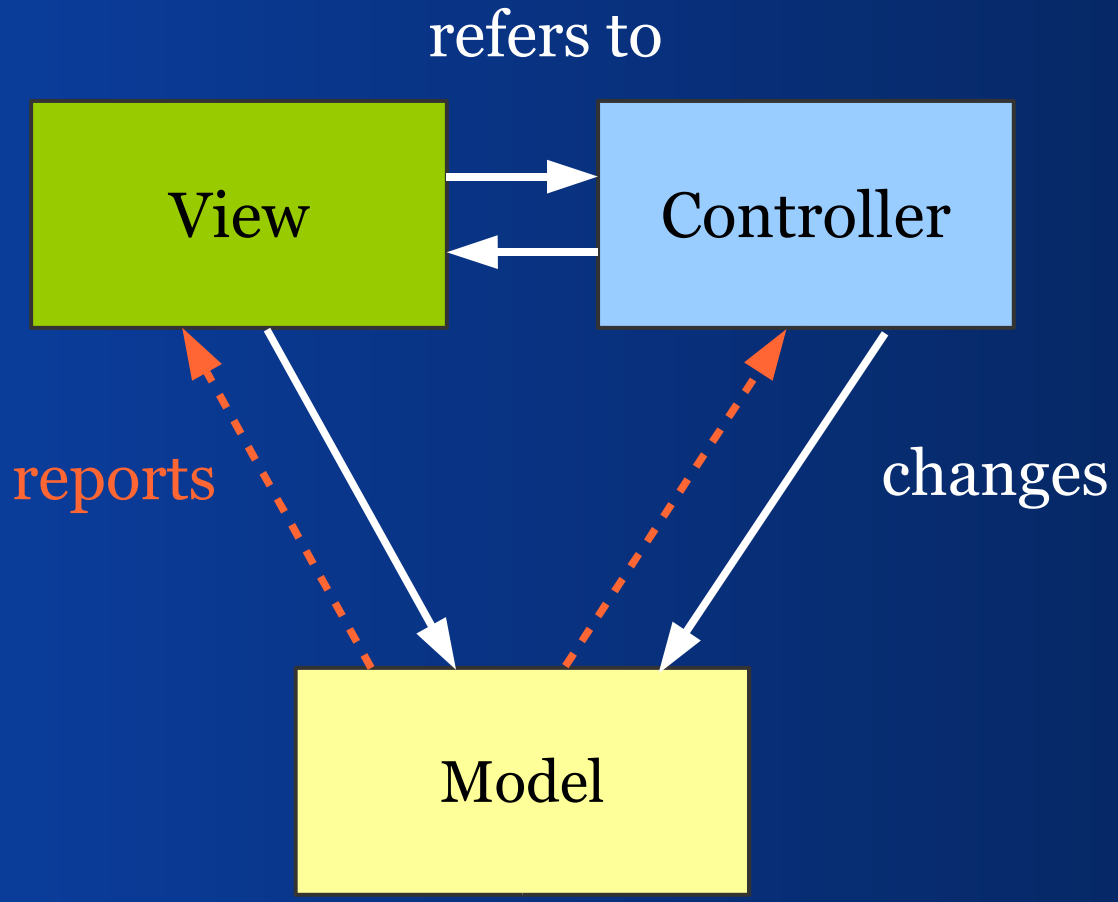
If you separate the painting code (View) from the state modifications (Controller), it's easier to:

- combine views and controllers
- reuse views and controllers

MVC



MVC



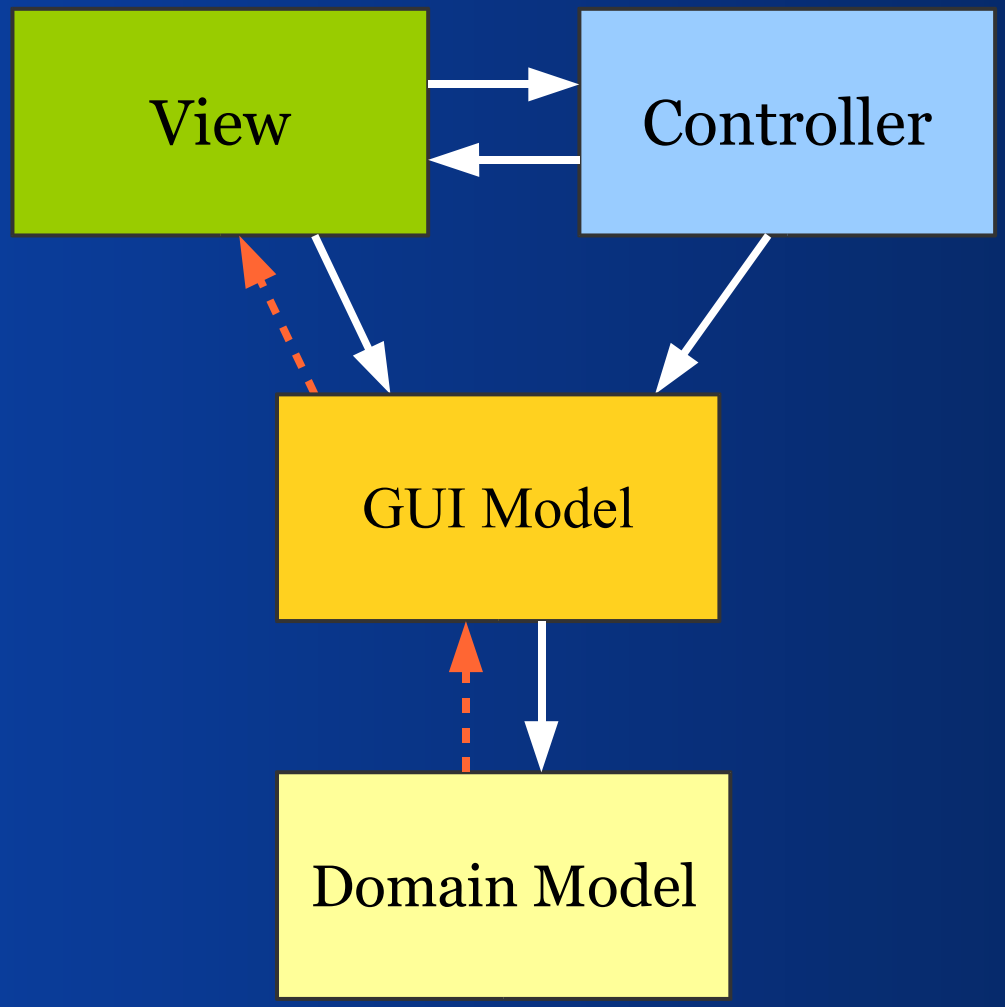
UI State vs. Data State

We can categorize models into:

- domain related
- GUI related

GUI state can make up its own layer.

MVC plus Model Layer



Presentation Layer

(GUI) Model Layer

Domain Layer

Candidates for a Model Layer

- TreeModel: converts a domain object tree into a form useable for JTree

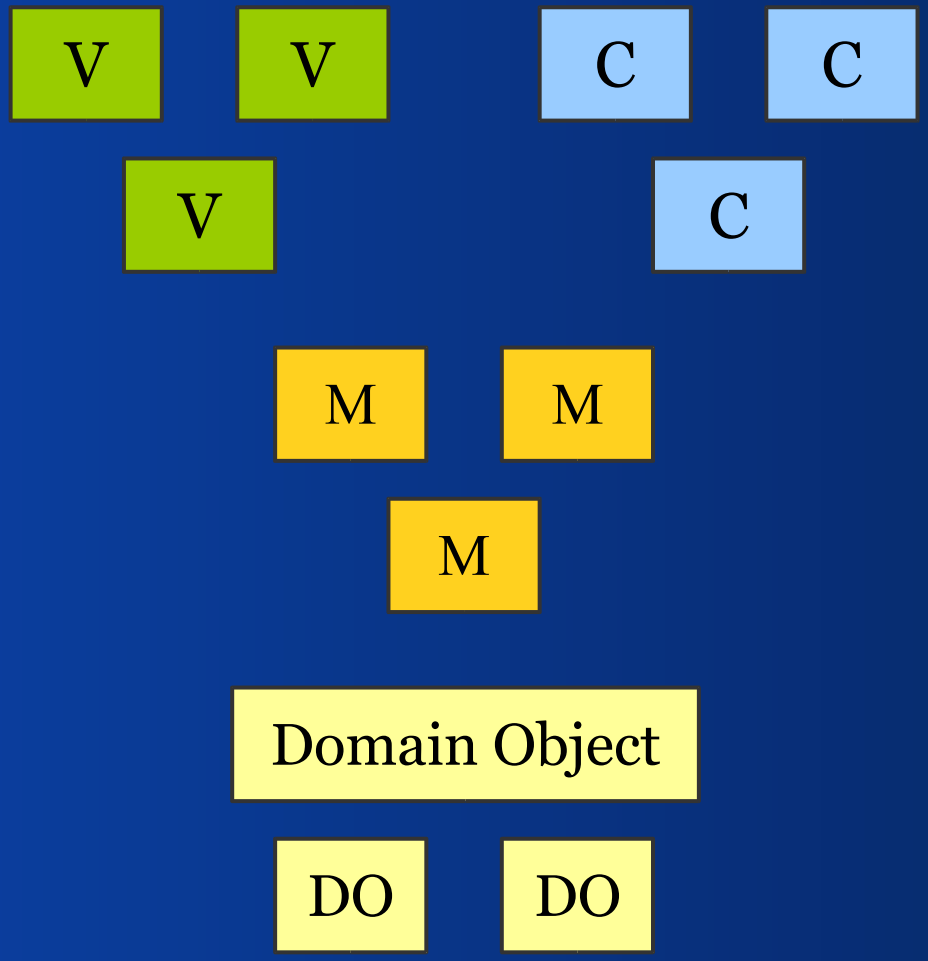
- Models that do not belong to the domain:
  - GUI state, e. g. *mouse pressed, mouse over*
  - Password in a login dialog
  - Search values

Conbining MVC Triads

A typical MVC UI combines MVC triads.

- Defines models as a graph of domain objects
- Composes larger views from small views
- Composes controllers from subcontrollers

MVC Triads with Model Layer



Presentation Layer

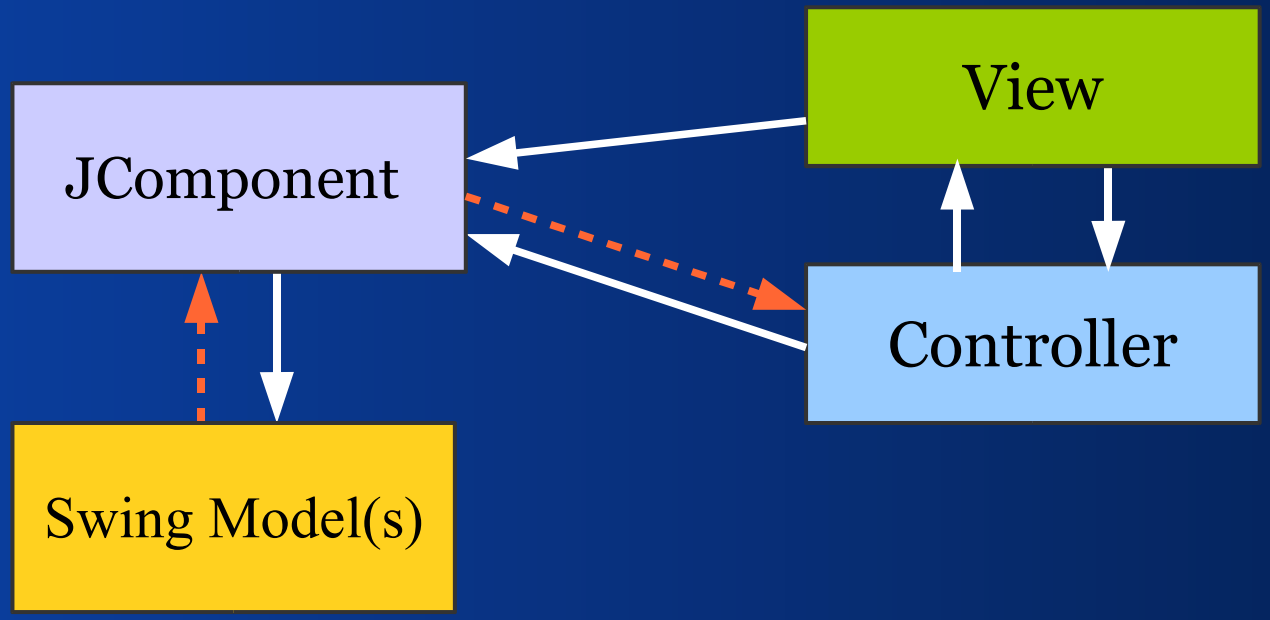
Model Layer

Domain Layer

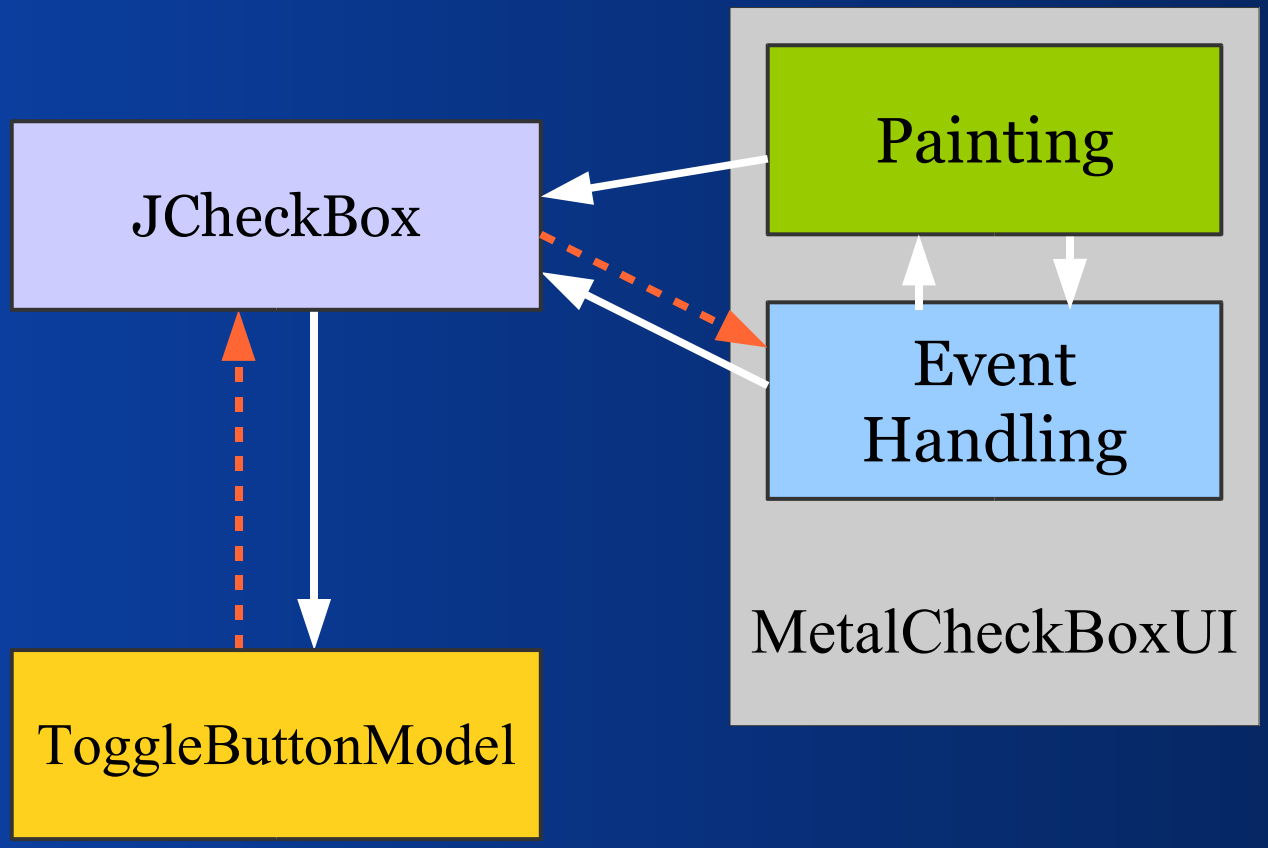
Factoring out the Look&Feel

Swing can change its appearance and behavior or in other words: look and feel.

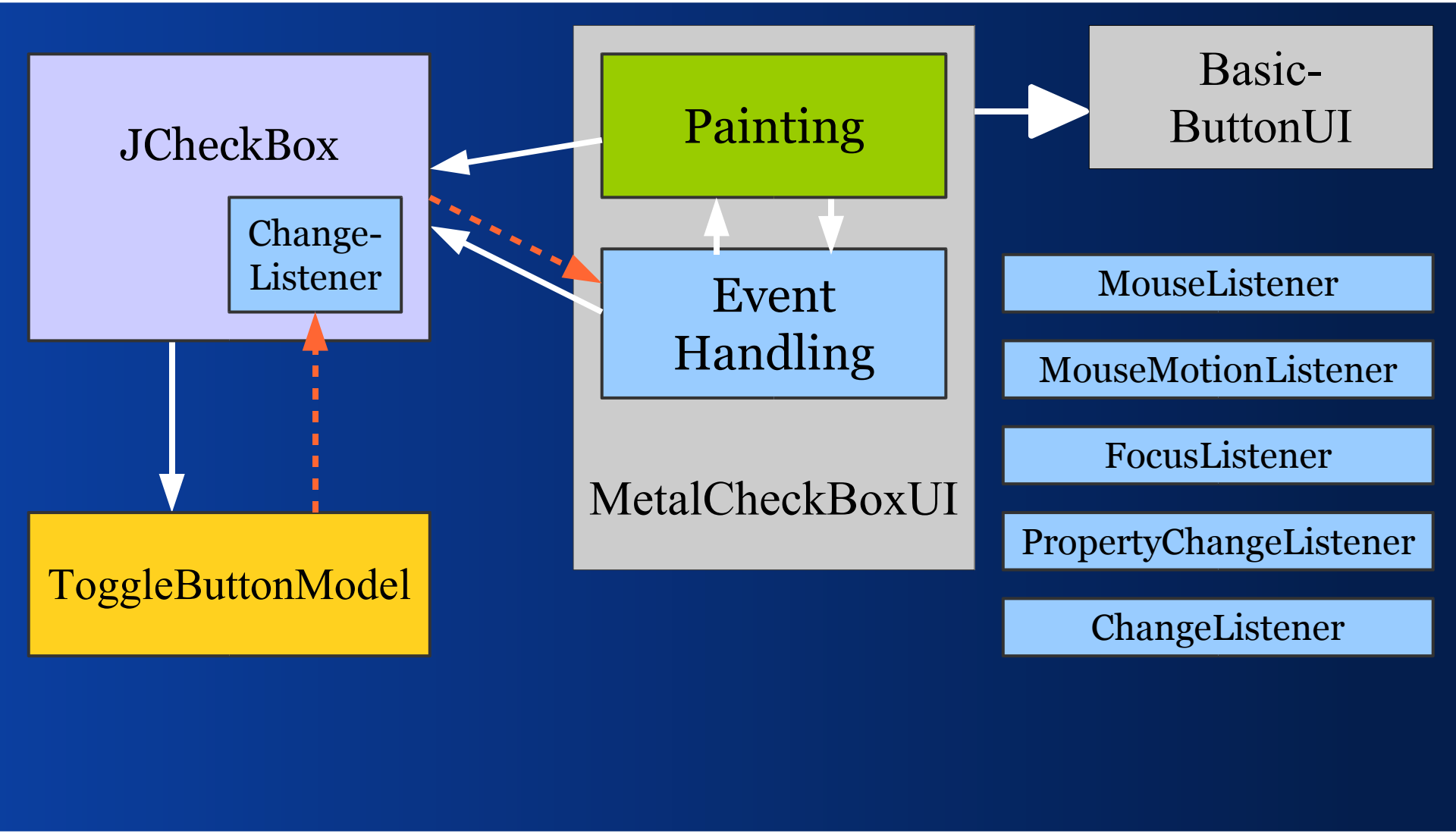
M-JComponent-VC



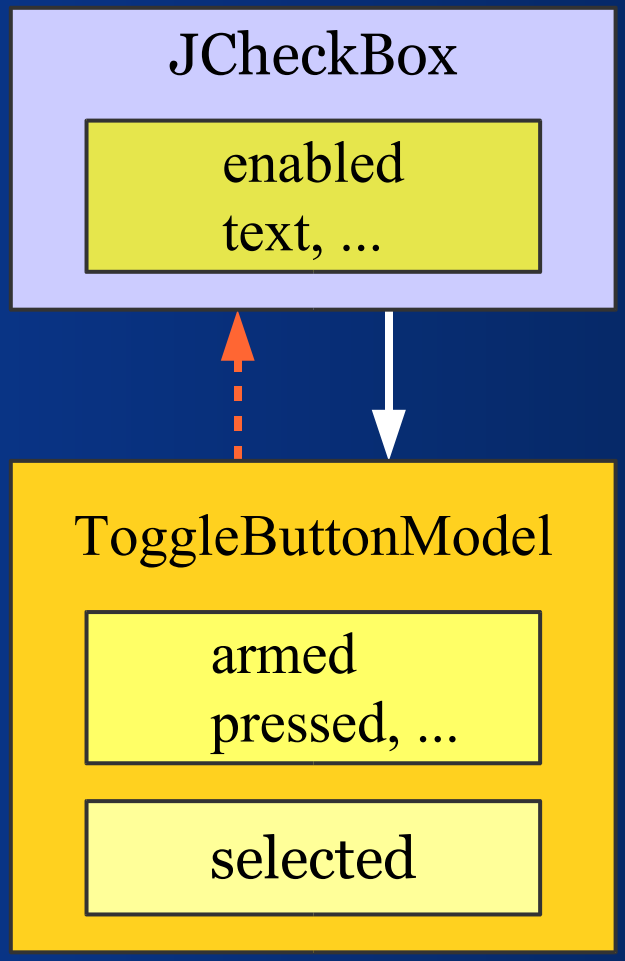
Example: JCheckBox



JCheckBox: Some Details



JCheckBox: Types of State

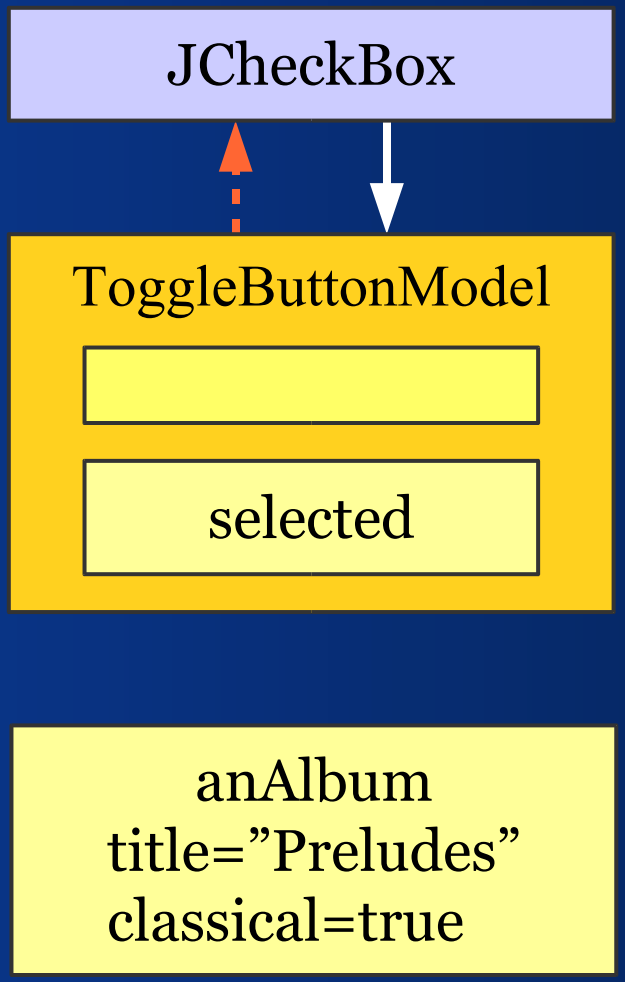


GUI Properties

GUI State

Data State

JCheckBox: Binding Task

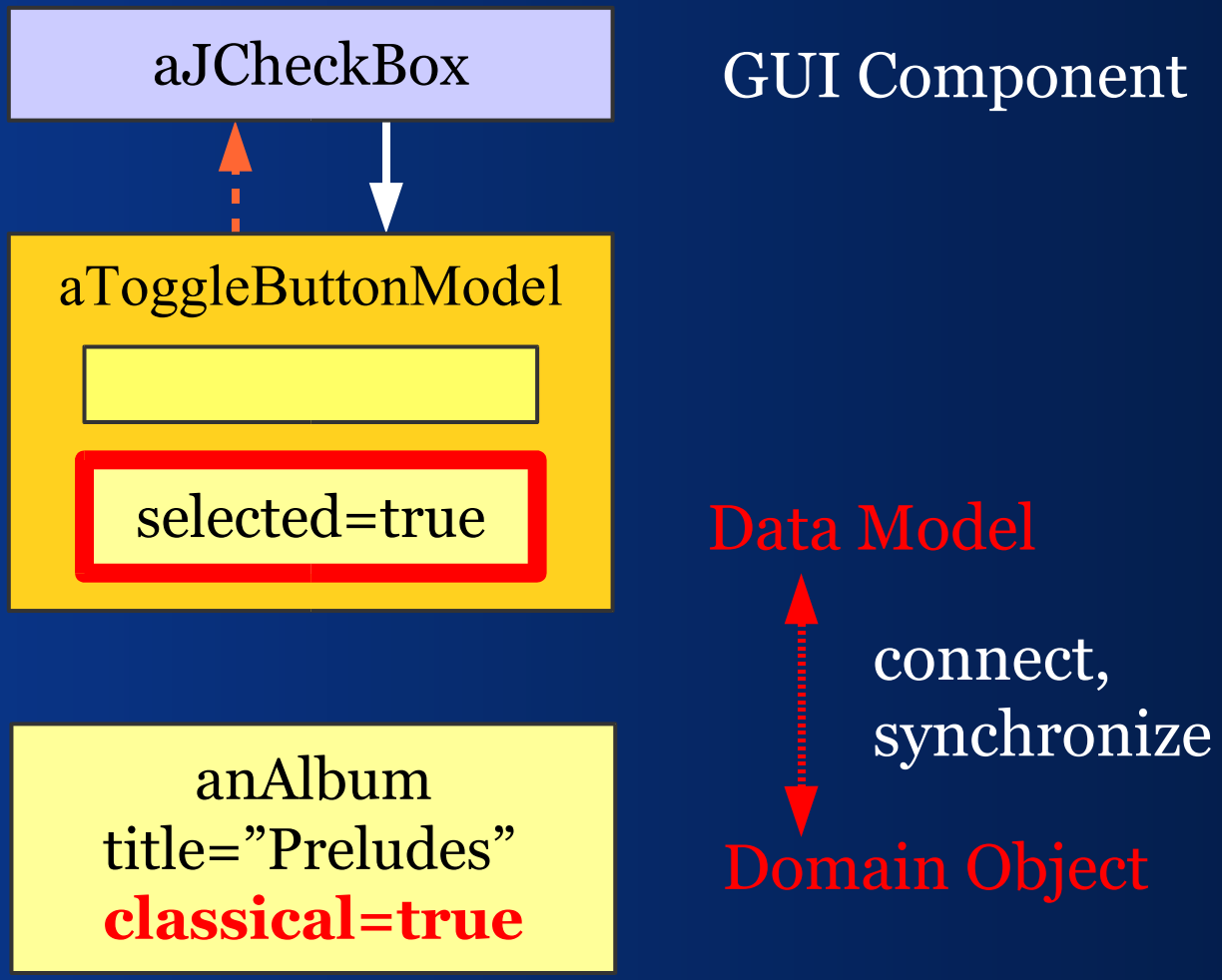


GUI Component

Data Model

Domain object

JCheckBox: Binding Task



Summary

- Swing doesn't use the original MVC
- Swing uses an extended form of MVC
- Swing shares the motivation behind MVC
- Swing adds features to the original MVC

Therefore, we will search and compare binding solutions for Swing, not MVC.

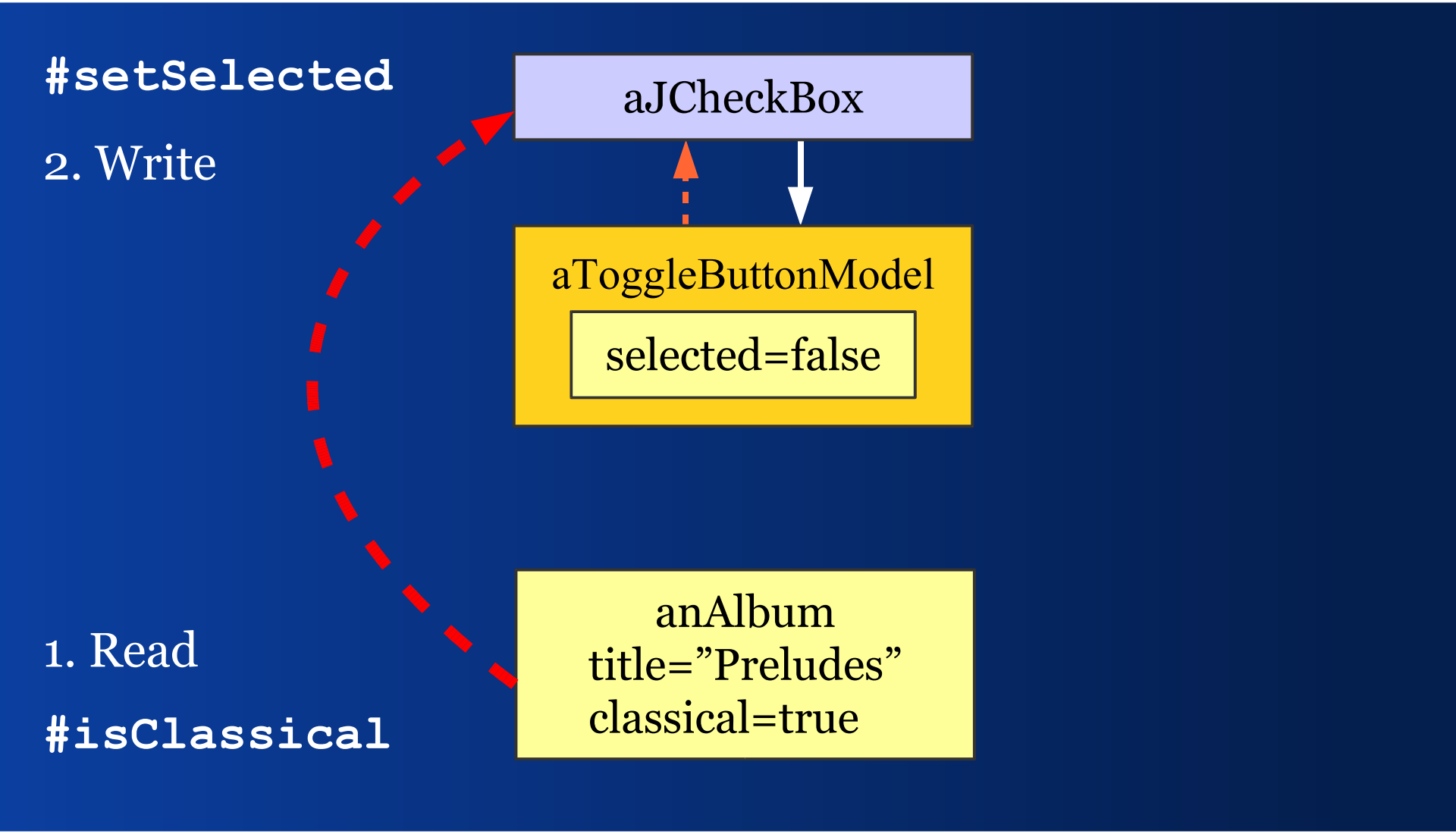
II - Binding Values

How to connect
domain objects with UI components?

Binding Tasks

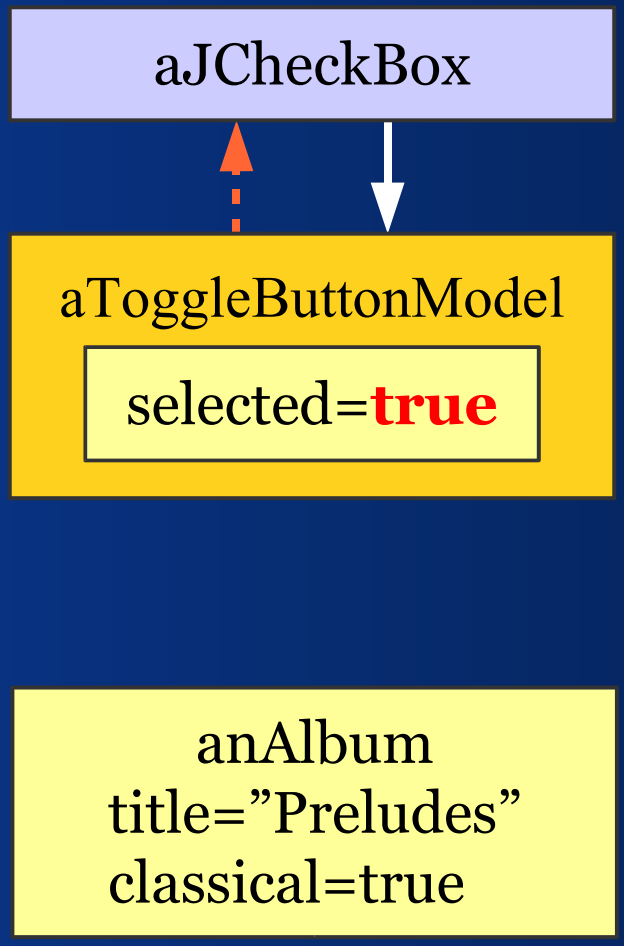
- Read and write domain object properties
- Get and set GUI model state
- Report and handle changes in the domain
- Buffer values – delay until OK pressed
- Change management – commit required?
- Indirection as in an *Master-Detail* view
- Convert types, e. g. Date to String

Copying Values to Views

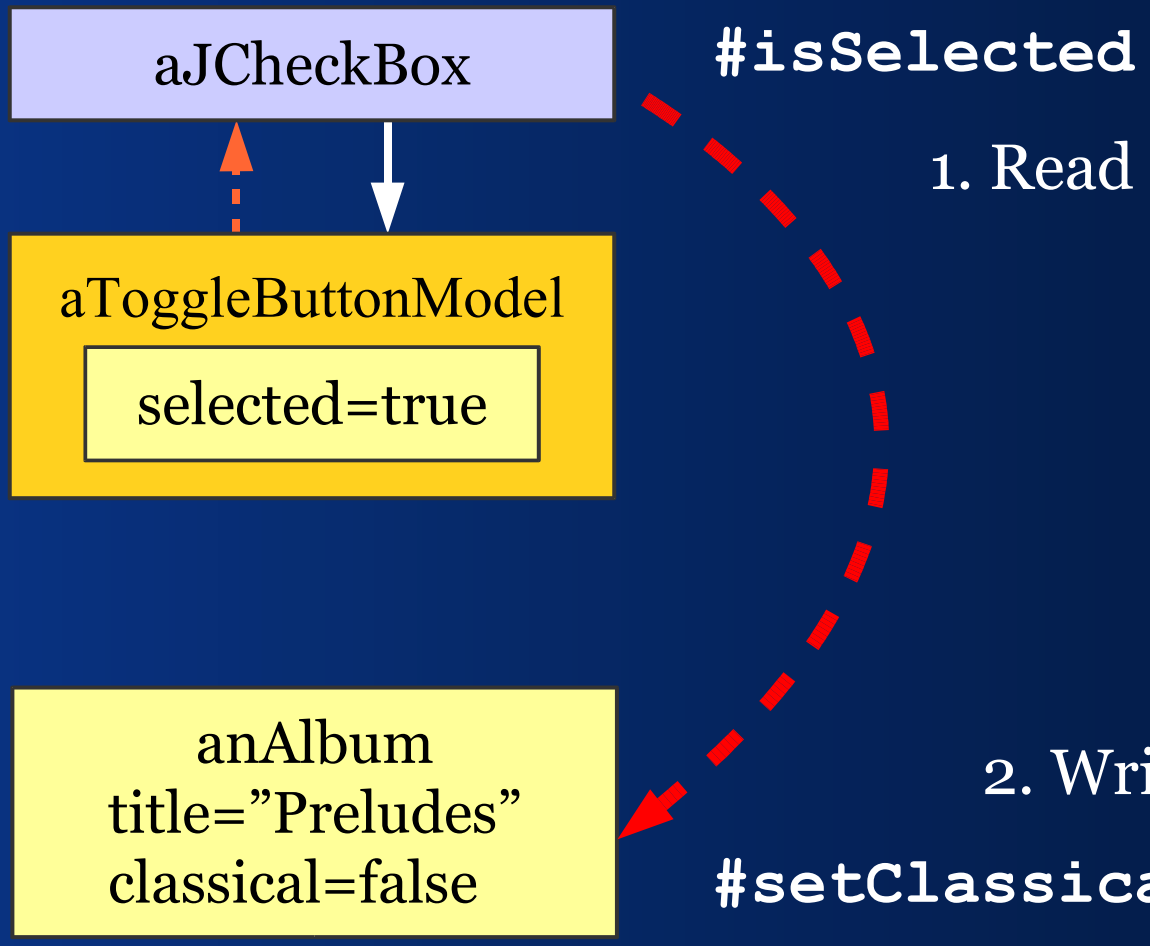


Copying Values to Views

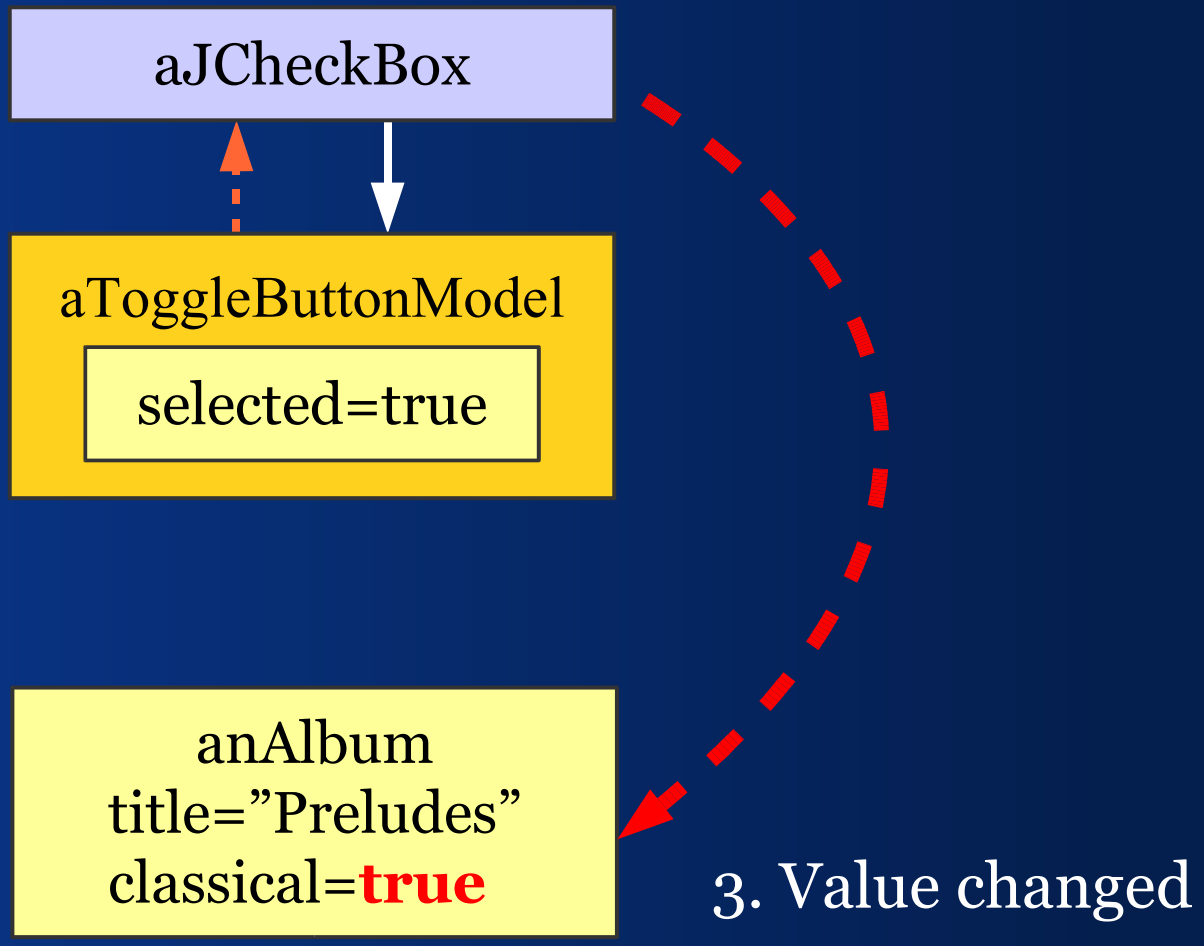
3. Value changed



Copying Values to the Domain



Copying Values to the Domain



Code Example: Copy to View

```
public void modelToView() {

    Album anAlbum = getEditedAlbum();

    classicalBox.setSelected(
        anAlbum.isClassical());

    titleField.setText(
        anAlbum.getTitle());
    ...
}
```

Code Example: Copy to Domain

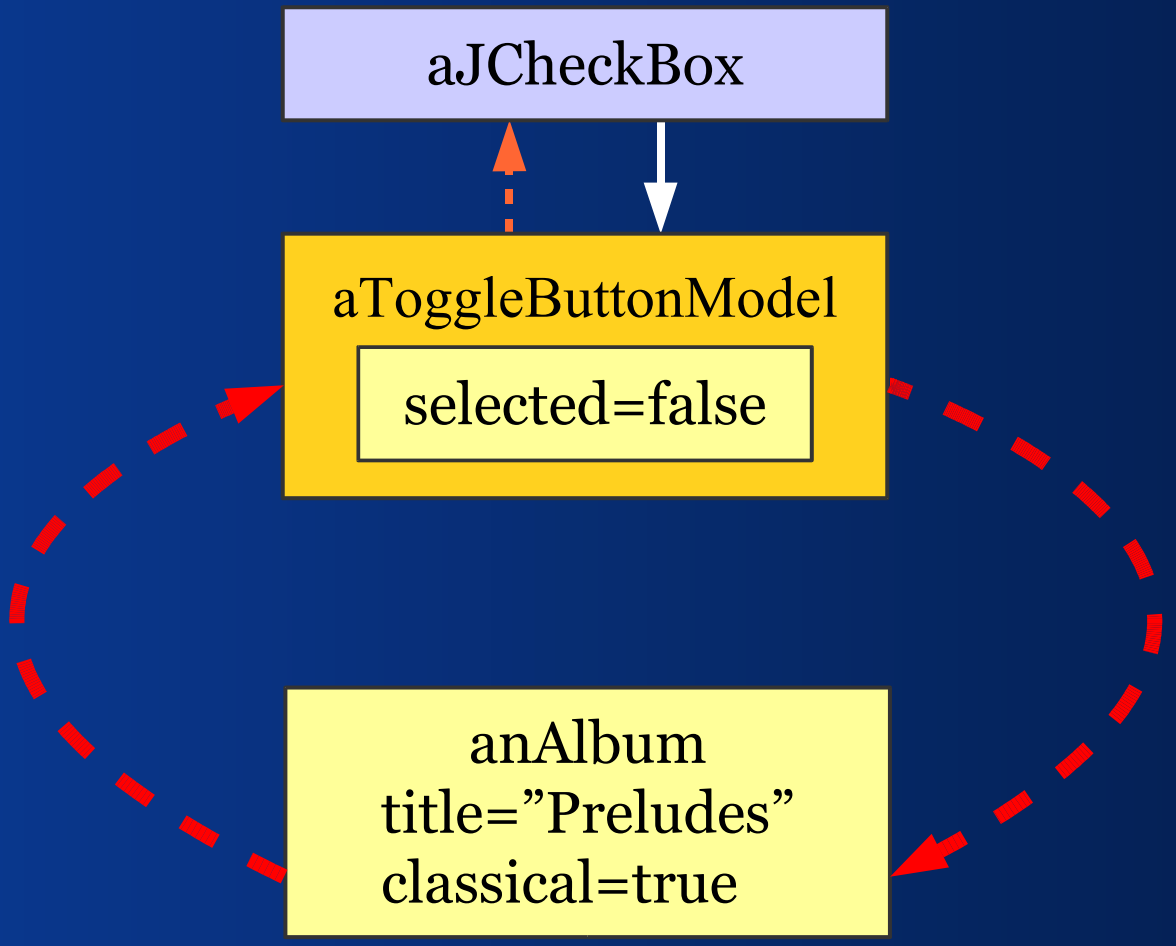
```
public void viewToModel() {

    Album anAlbum = getEditedAlbum();

    anAlbum.setClassical(
        classicalBox.isSelected();

    anAlbum.setTitle(
        titleField.getText();
    ...
}
```

Copying: Pros and Cons

- Easy to understand, easy to explain
- Works in almost all situations
- Easy to debug – explicit data operations

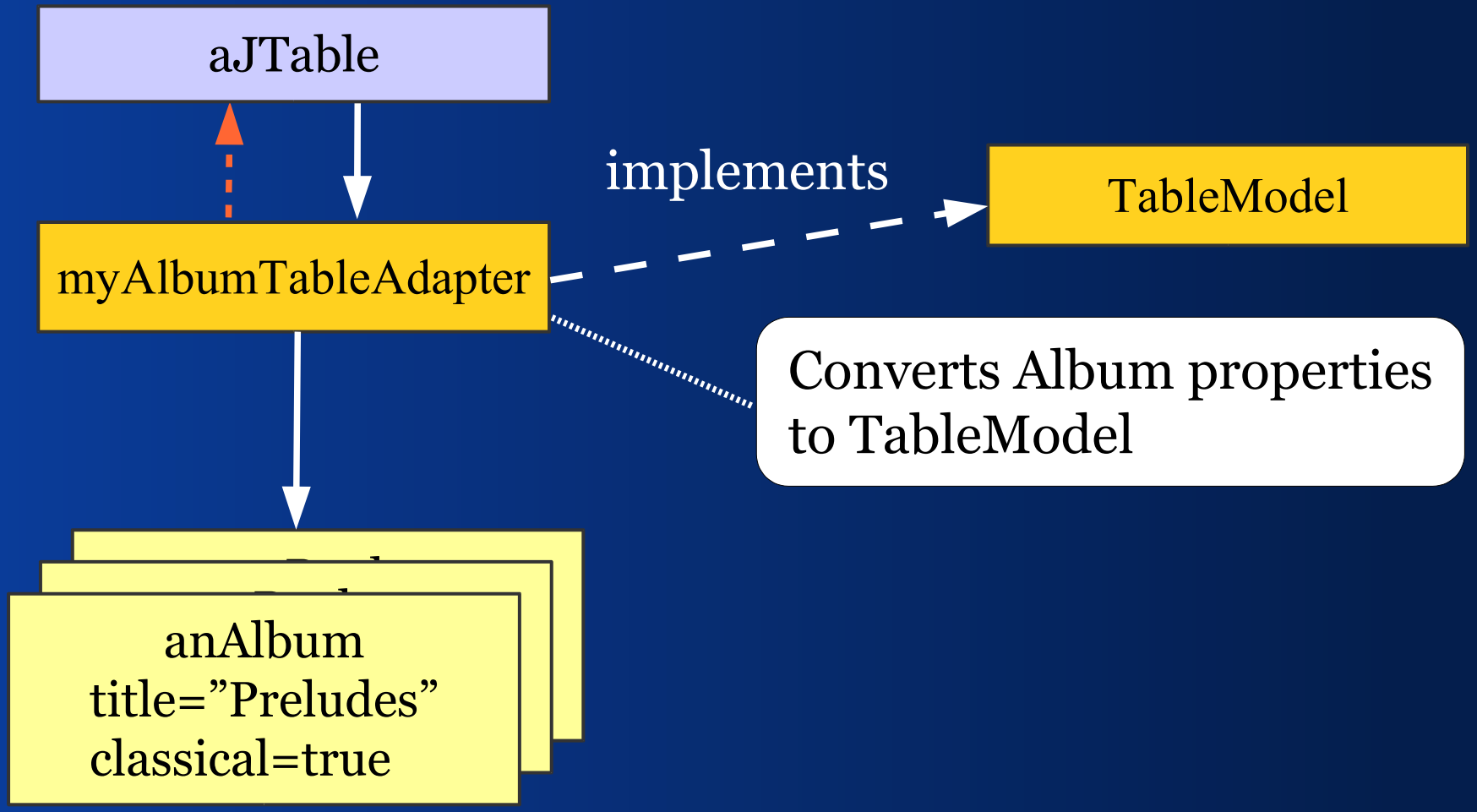
- Blows up the view code
- It's difficult to synchronize views
- Handles domain changes poorly

Alternative

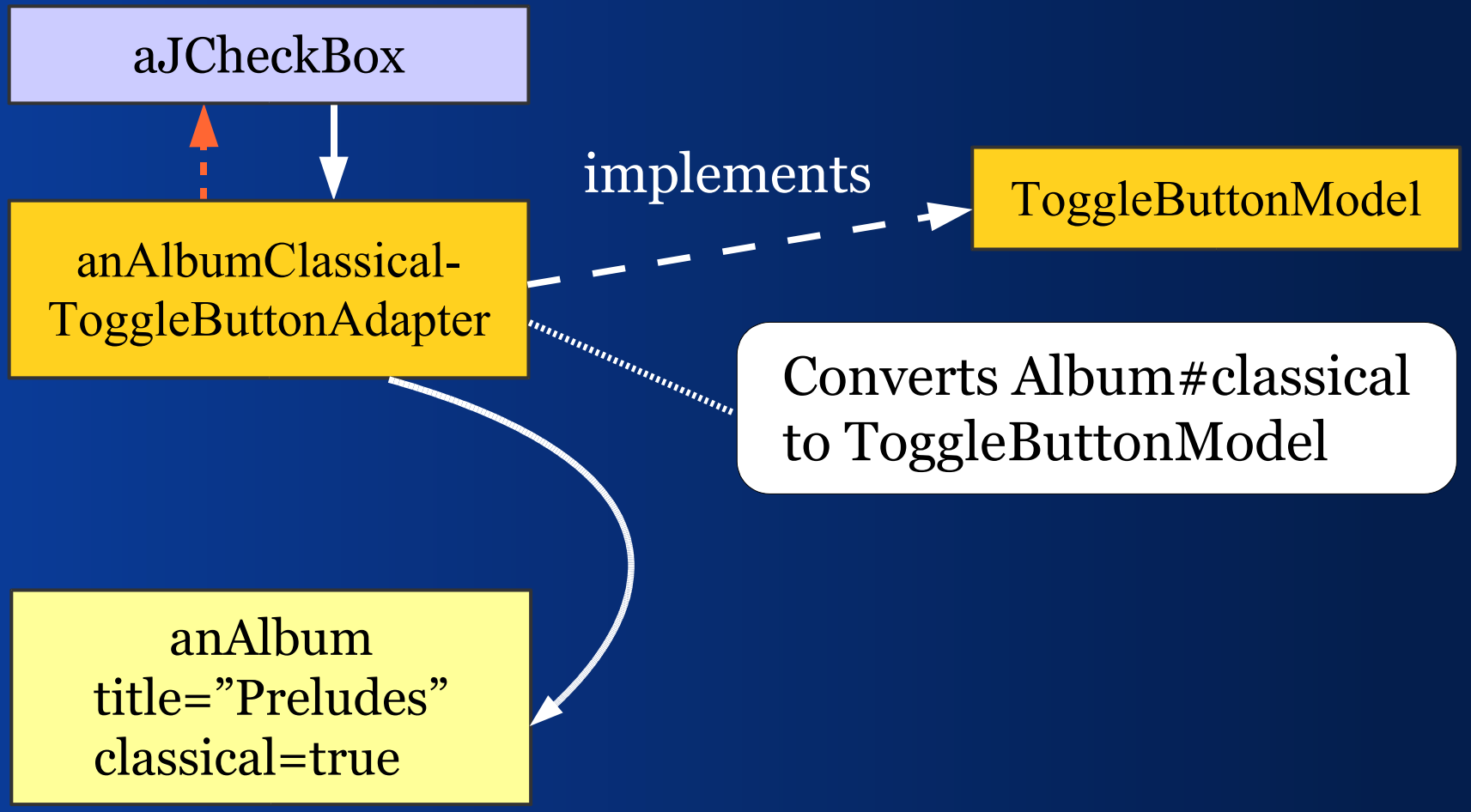


Note: you can't share the model

Direct Adapter: TableModel



Direct Adapter: JCheckBox



Problem with Direct Adapters

Requires an individual adapter for each domain object property.

Similar to incompatible electric connectors.

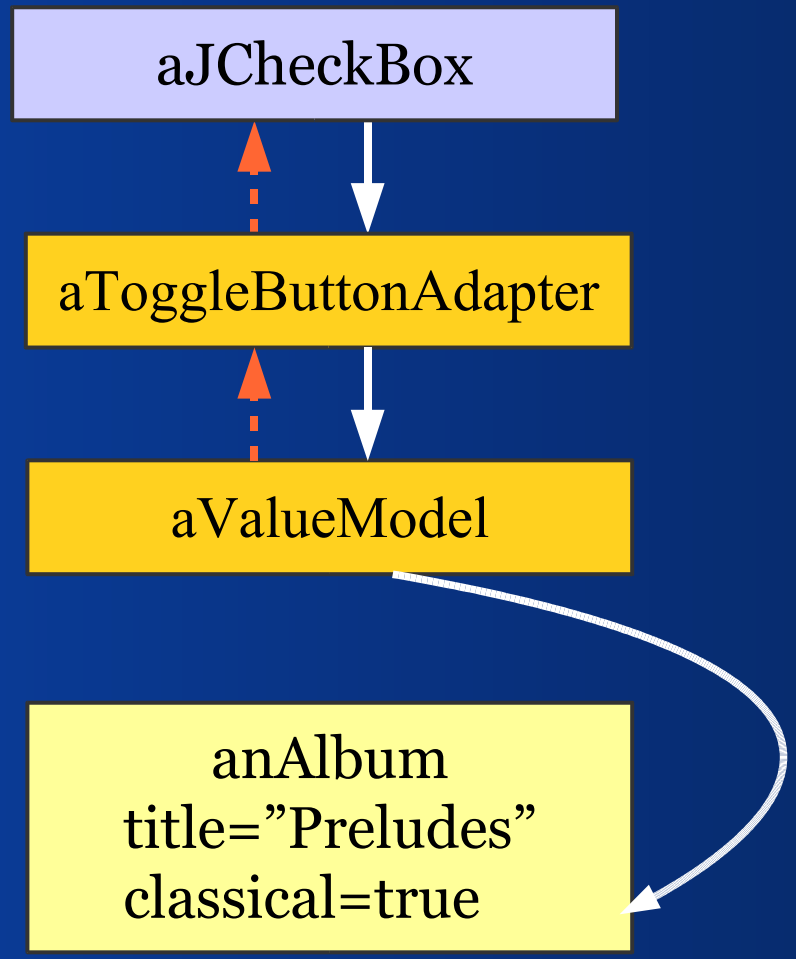
Code is all the same except for the methods that read and write domain properties.

Concept

- Use a universal model (ValueModel)
- Convert domain properties to ValueModel
- Build converters from ValueModel
  to Swing models: ToggleButtonModel, etc.

- We end up with about 15 classes.

ValueModel and Adapter



ValueModel: Requirements

- We want to get its value
- We want to set its value
- We want to observe changes

The ValueModel Interface

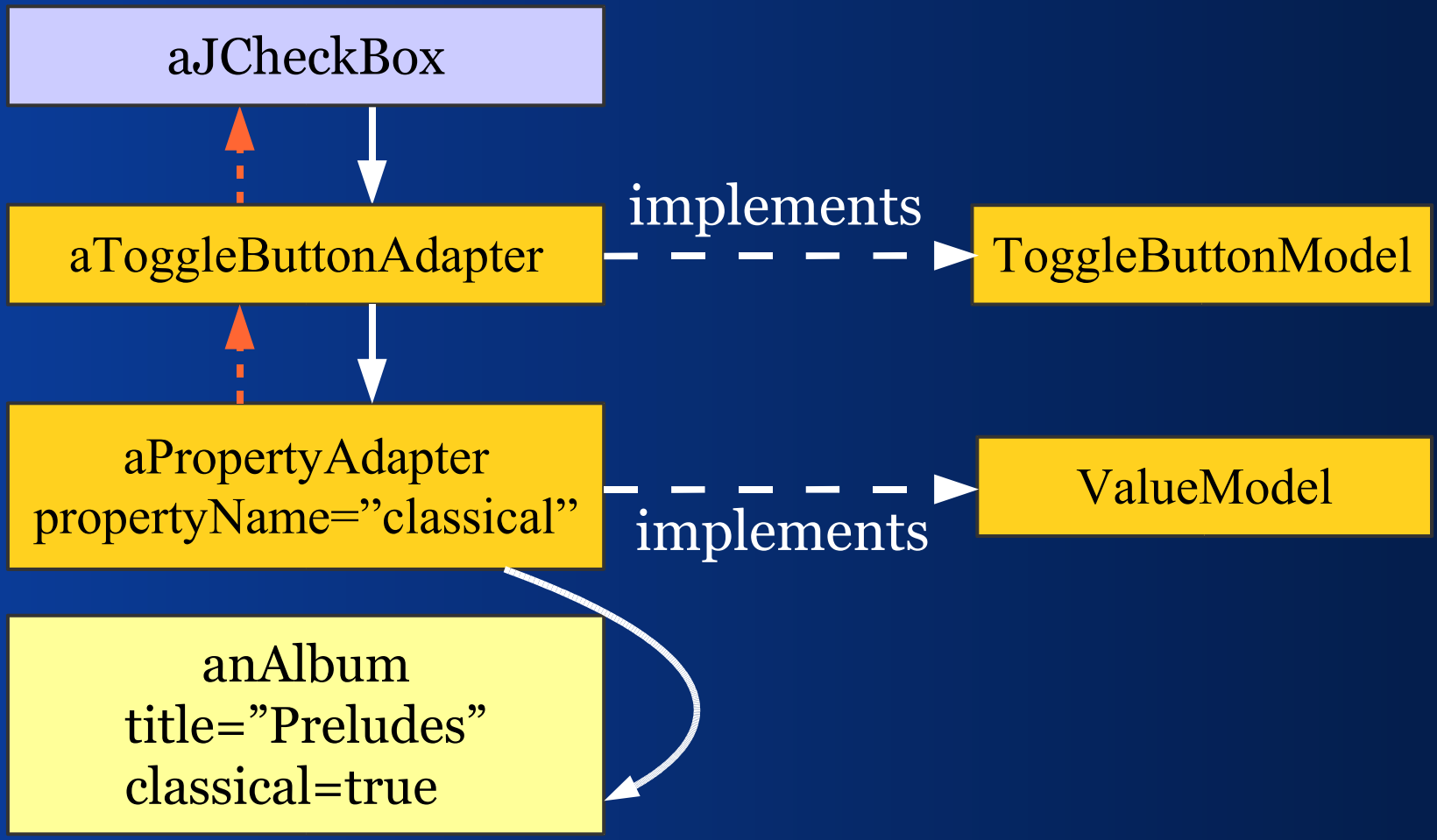
```
public interface ValueModel {

  Object getValue();

  void setValue(Object newValue);

  void addChangeListener(ChangeListener l);

  void removeChangeListener(ChangeListener l);
}
```

Which Event Type?

- ChangeEvent reports no new value;
  must be read from the model – if necessary

- PropertyChangeEvent
  provides the old and new value;
  both can be `null`

ValueModel & PropertyAdapter



Domain Object Requirements

- We want to get and set values
- We want to do so in a uniform way
- Changes shall be observable

That's what Java Beans provide.

(Bound) Bean Properties

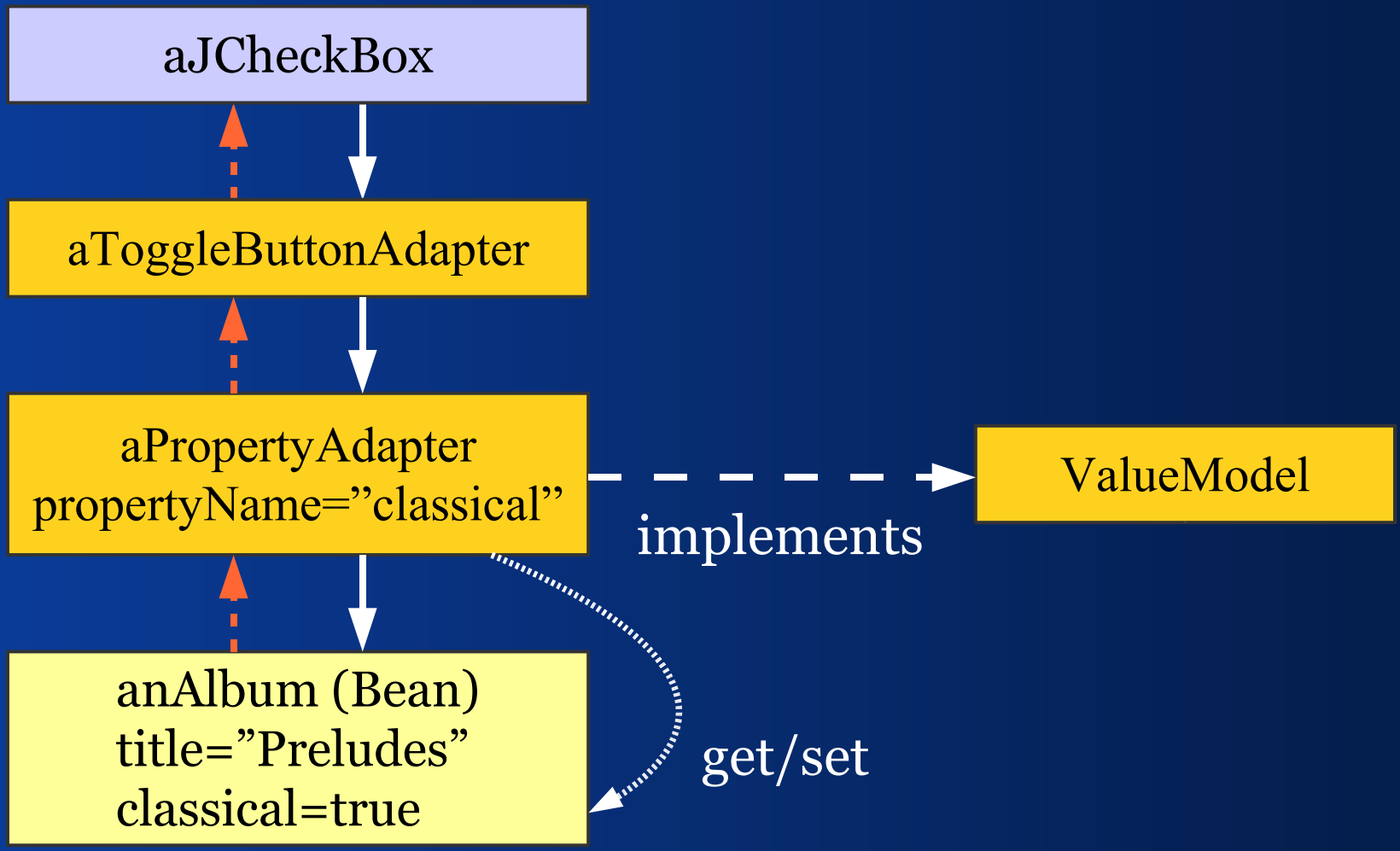
- Java Beans have properties,
  that we can get and set in a uniform way.

- Bean properties are bound,
  if we can observe property changes
  by means of PropertyChangeListeners.

PropertyAdapter

- `BeanAdapter` and `PropertyAdapter` convert Bean properties to ValueModel
- Observe bound properties
- Use Bean Introspection that in turn uses Reflection to get and set bean properties

ValueModel & PropertyAdapter



Build a Chain of Adapters

```
private void initComponents() {

   Album album = getEditedAlbum();

   ValueModel aValueModel =
        new PropertyAdapter(album, "classical");

   JCheckBox classicalBox = new JCheckBox();
   classicalBox.setModel(
        new ToggleButtonAdapter(aValueModel));
}
```

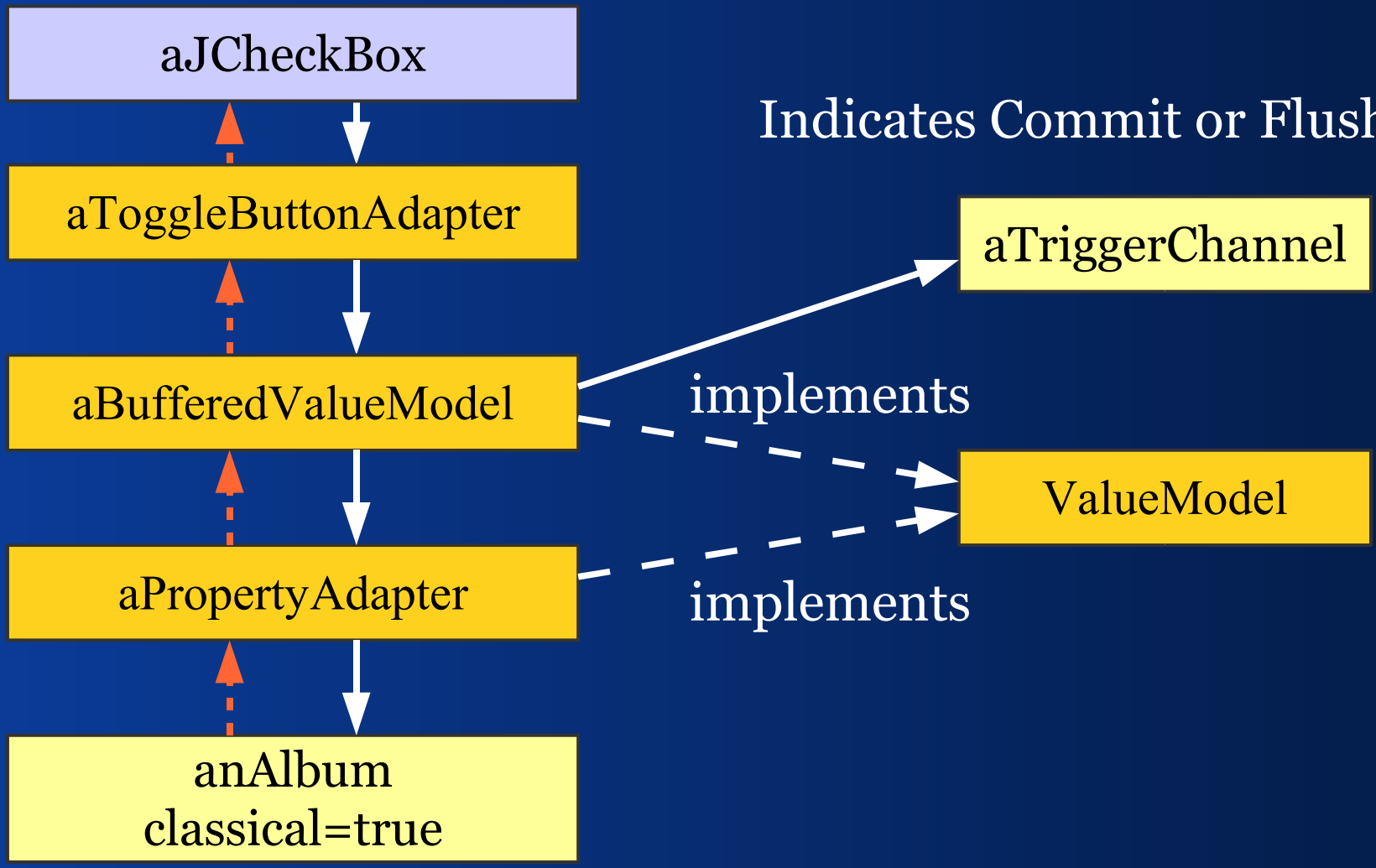
ComponentFactory

```
private void initComponents() {

   Album album = getEditedAlbum();

   JCheckBox classicalBox =
       ComponentFactory.createCheckBox(
           album,
           Album.PROPERTYNAME_CLASSICAL);
}
```

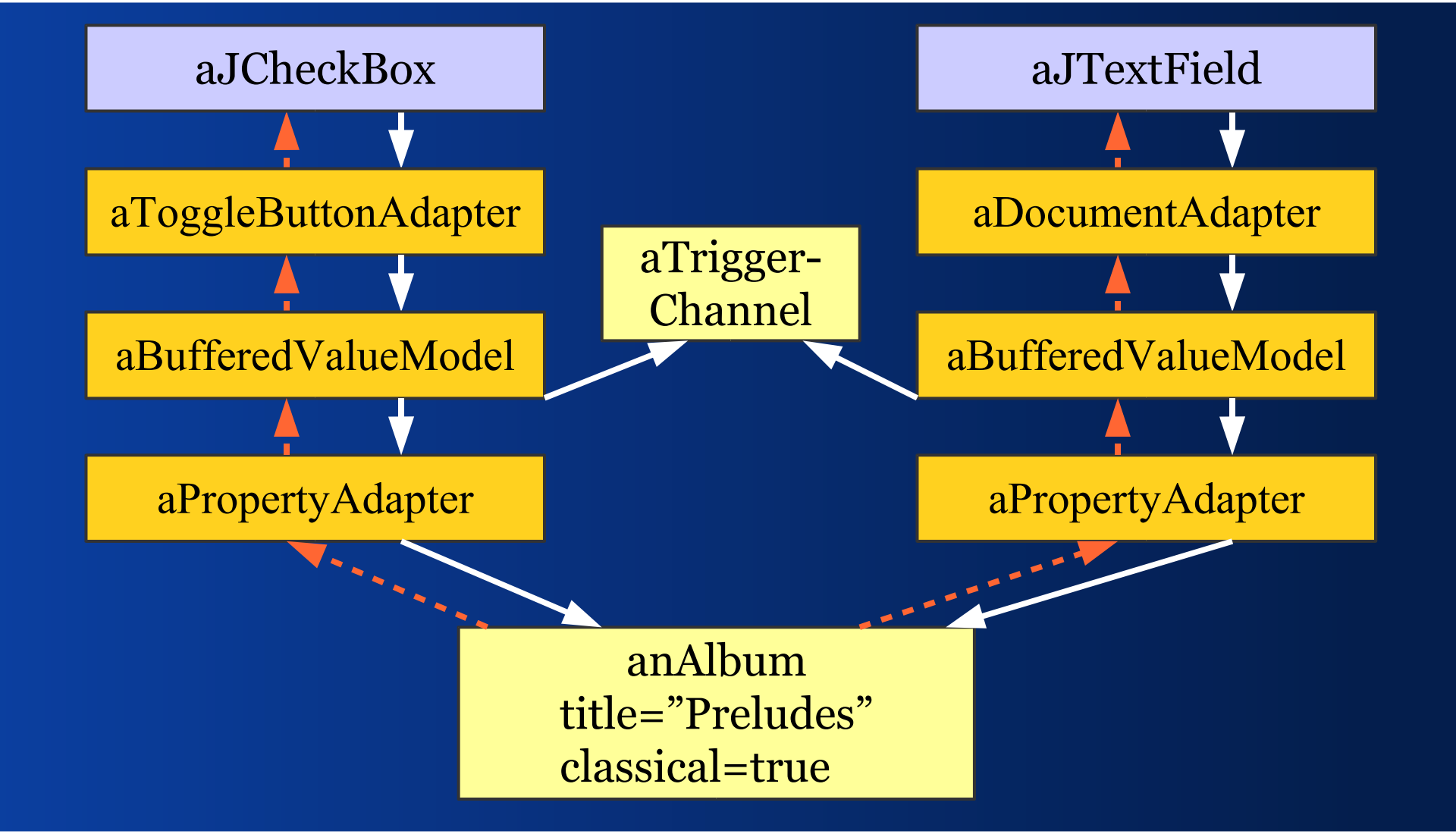
Buffering: Delaying Commits

- Selecting the JCheckBox changes the bound domain property immediately.
- Often we want to delay value commits until the user presses OK or Accept.
- We can buffer in the adapter chain or in the domain layer.

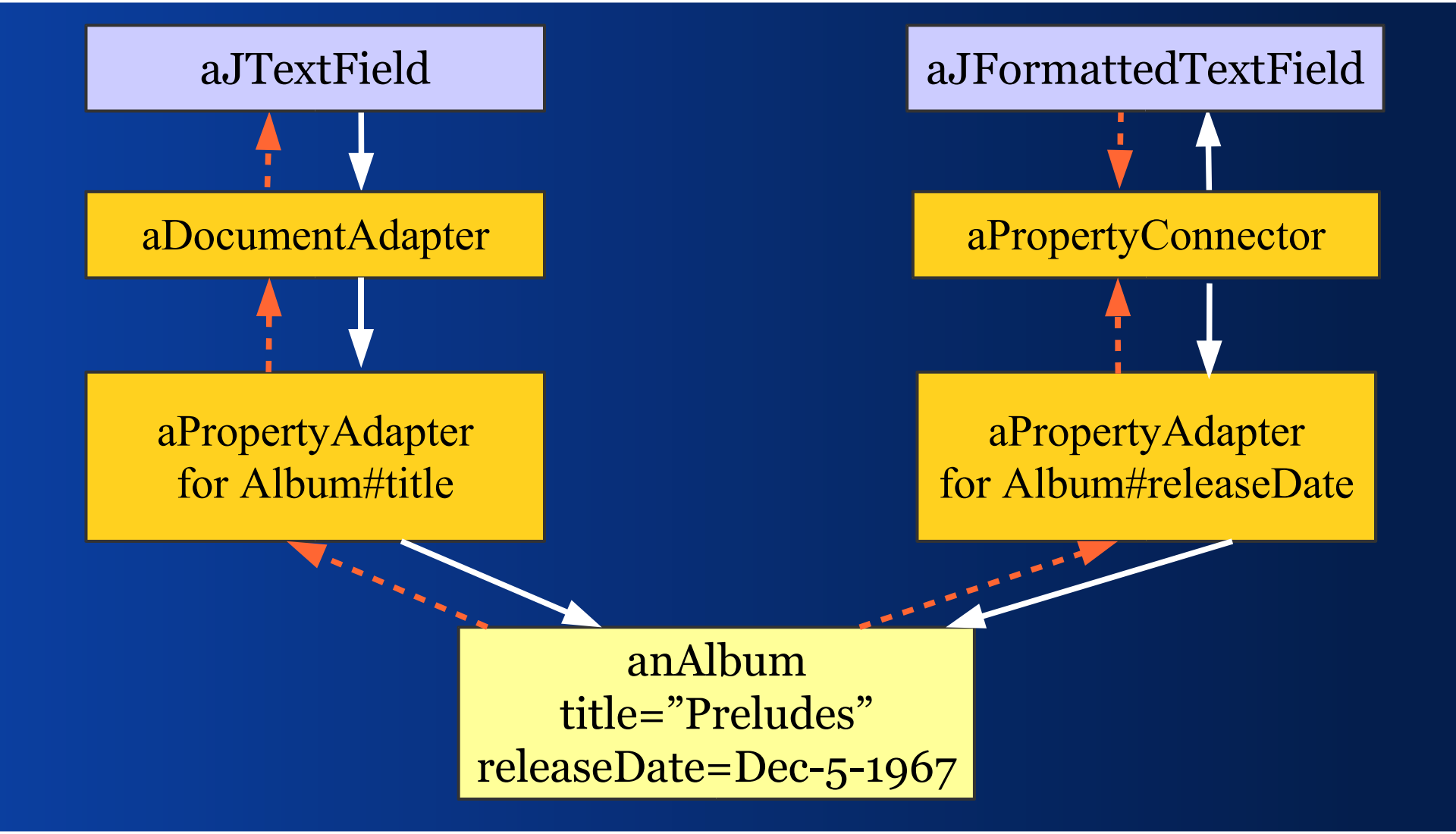
BufferedValueModel



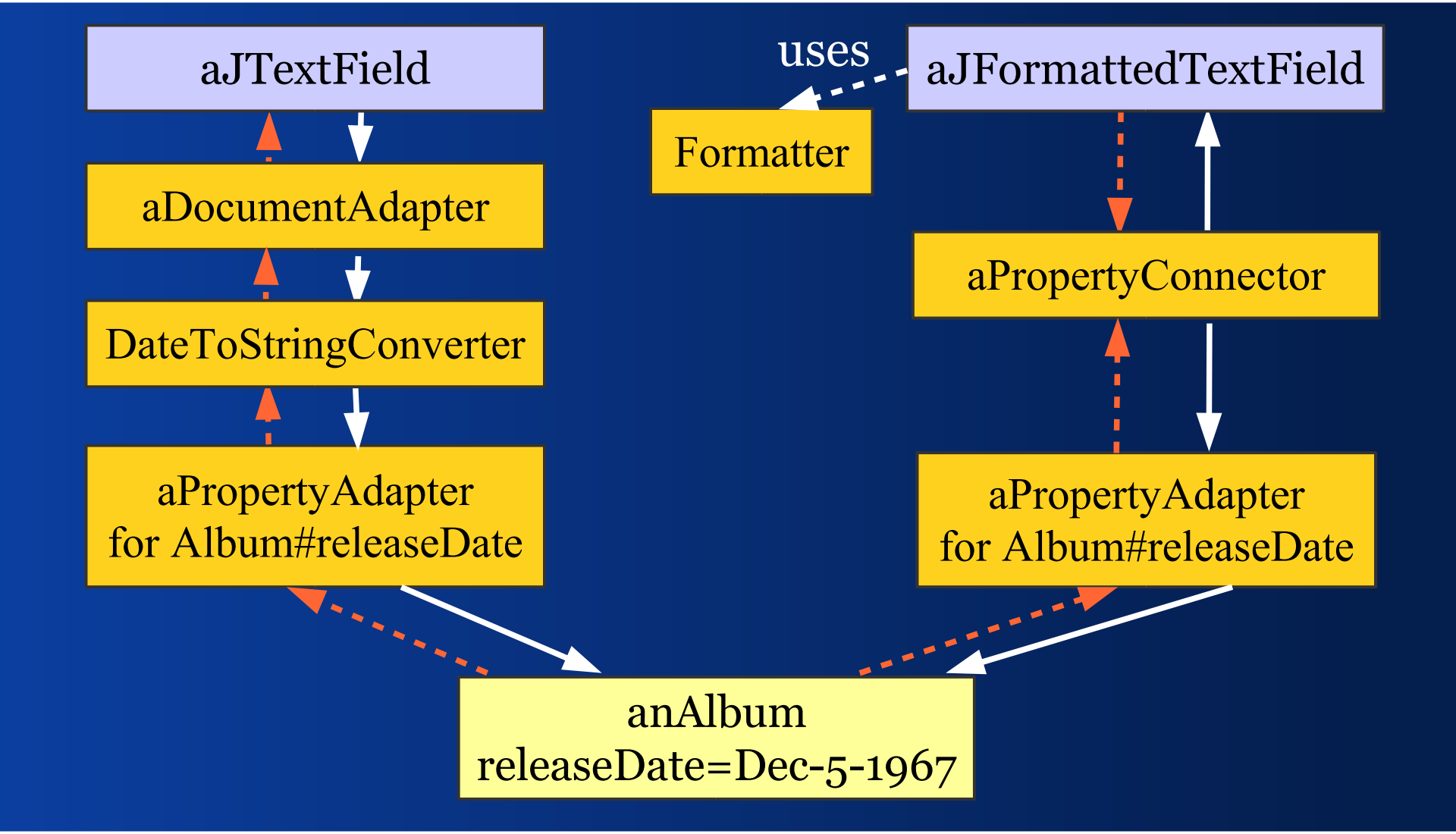
Sharing a Buffer Trigger



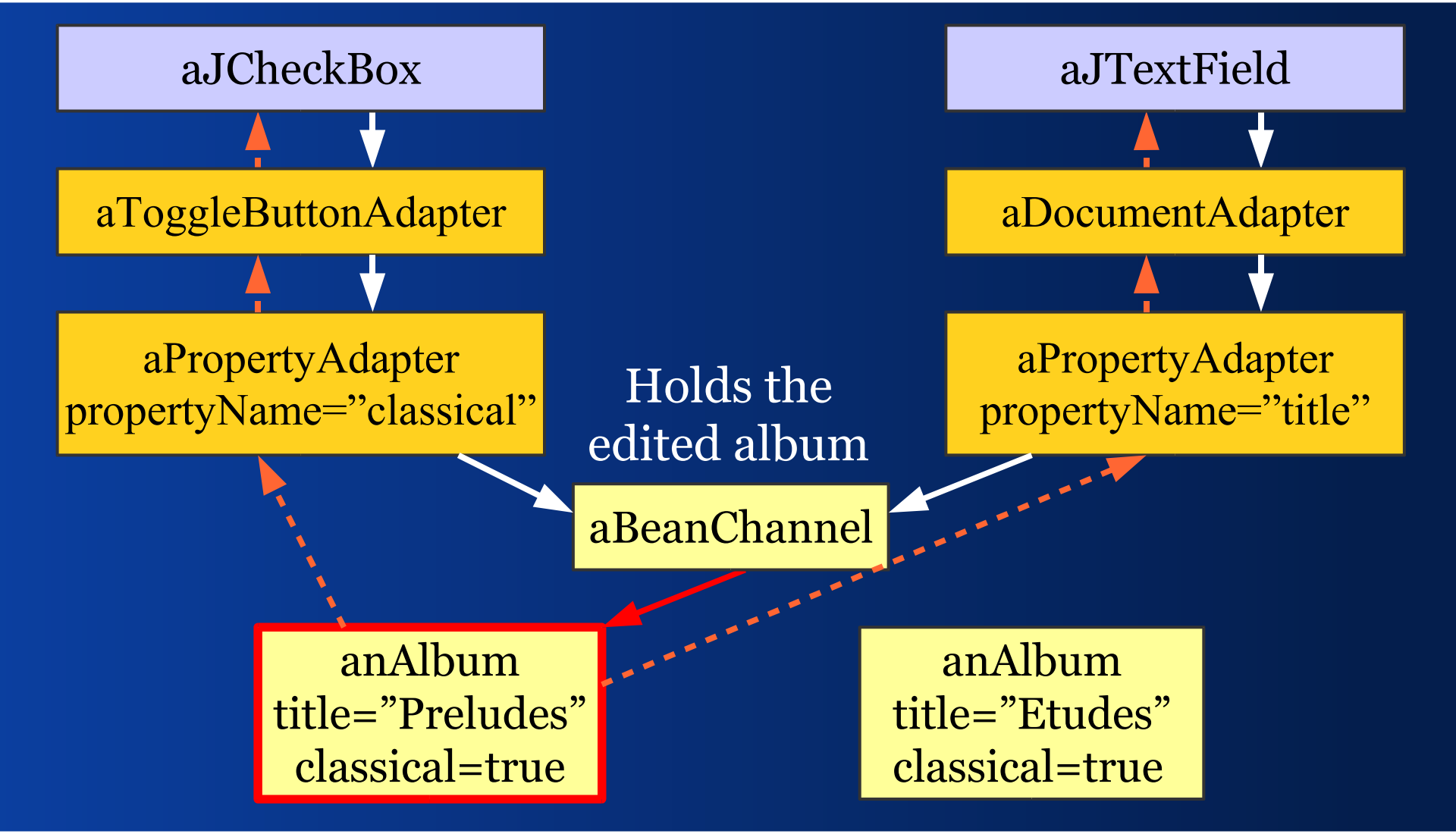
Adapter vs. Connector



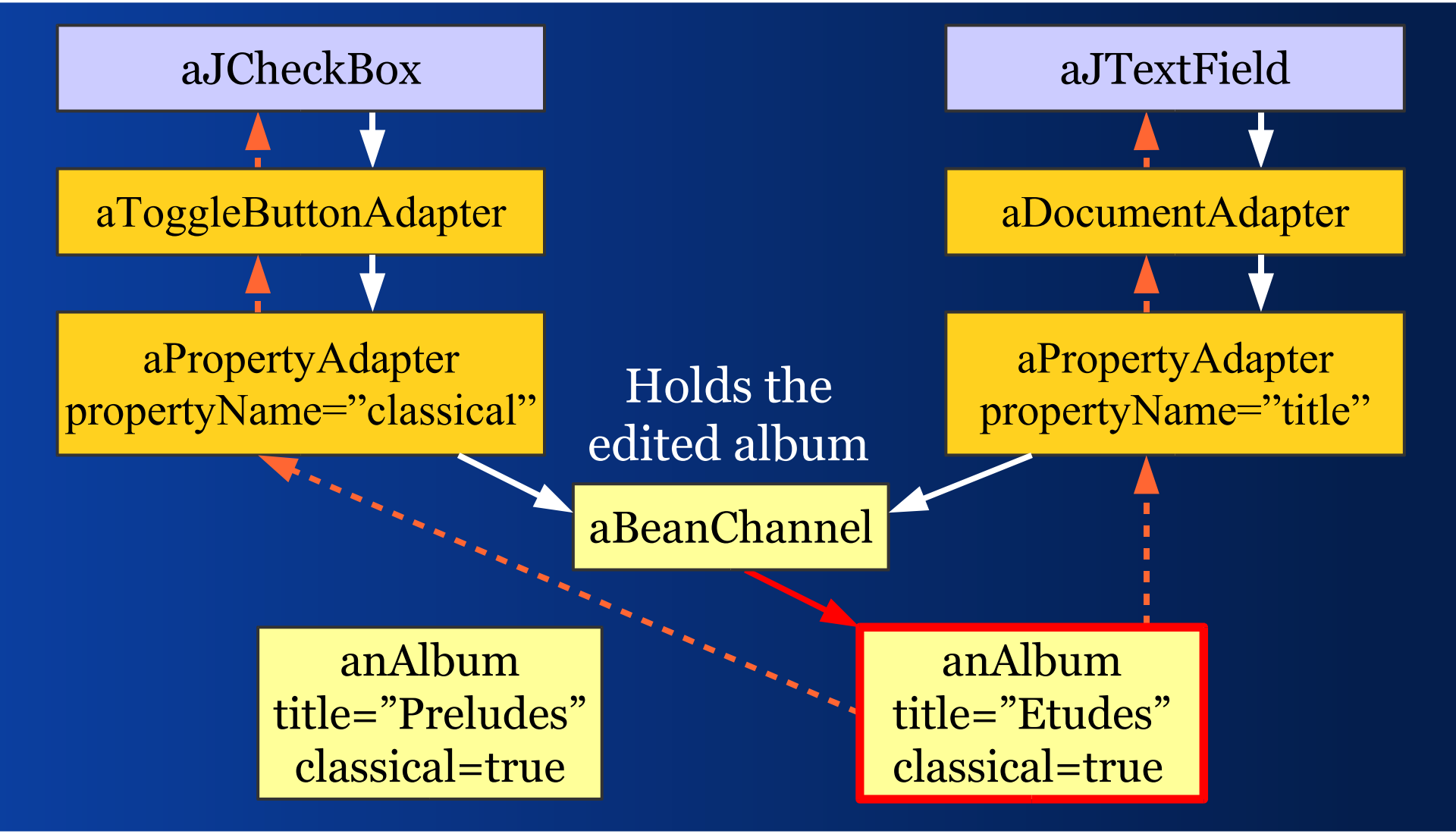
Converter vs. Formatter



Indirection



Indirection



III - Binding Lists

How to connect Lists of domain values with Swing components?

List Binding Problems

List views require fine grained change events.
We want to observe list content changes.

Otherwise list views poorly handle
the selection and scroll state.

Requirements for a List Model

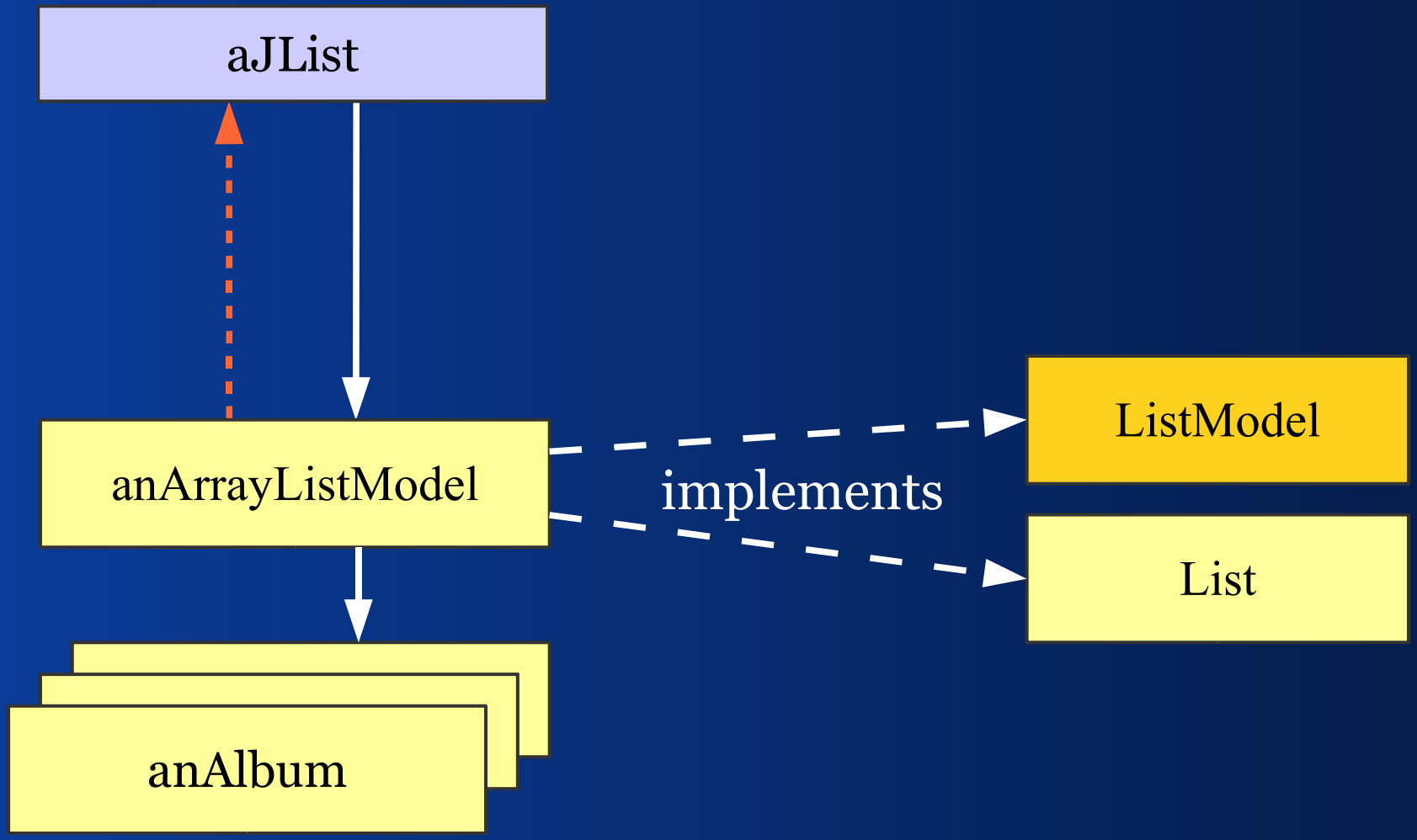
- Get list elements
- Provide the list size
- Report changes:
  - if elements change
  - if elements have been added
  - if elements have been removed

The Swing class `ListModel` provides this.

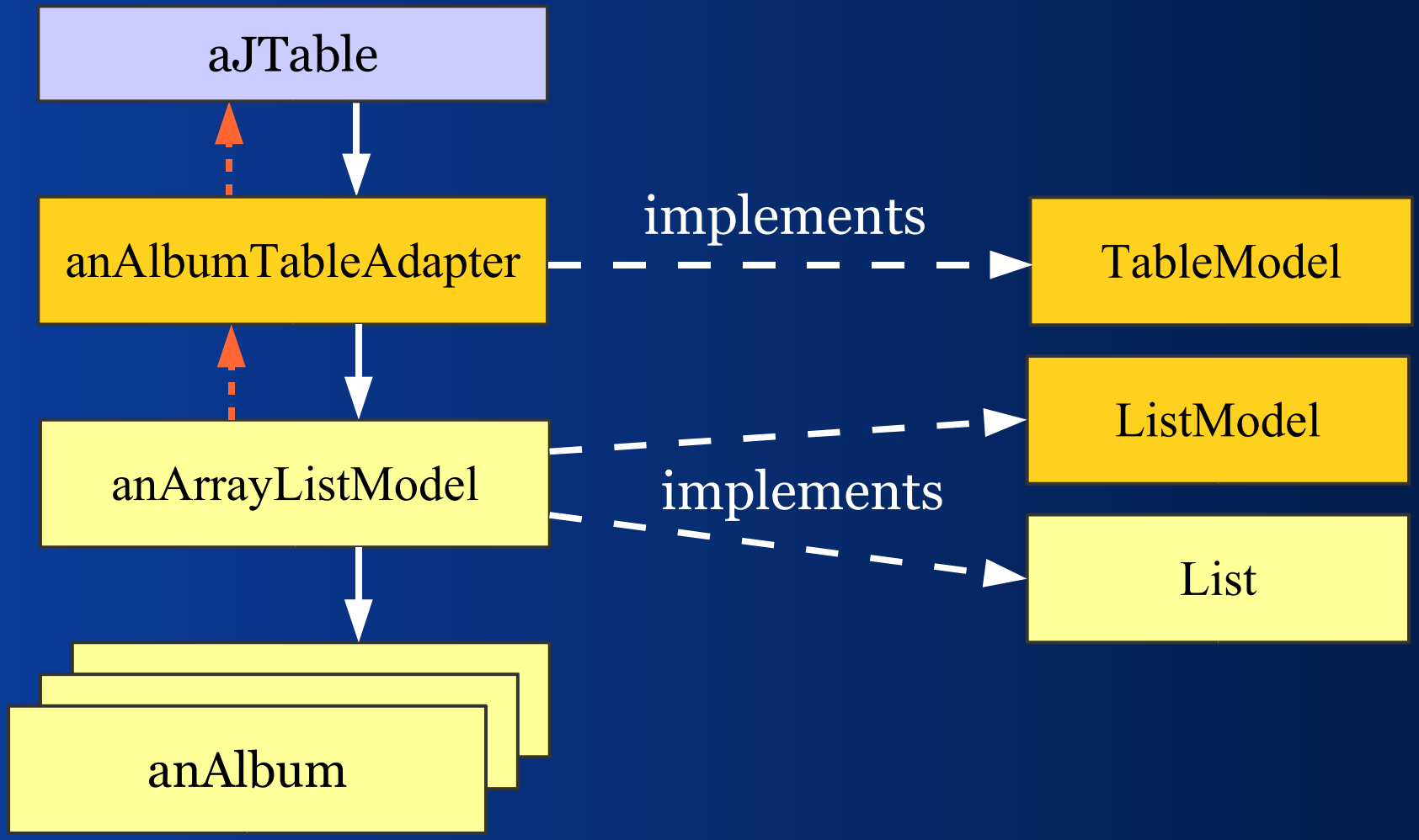
ListModel Implementations

- ArrayListModel extends ArrayList, implements ListModel
- LinkedListModel extends LinkedList, implements ListModel
- We can operate on List and can observe ListModel changes.

Binding Lists to JList



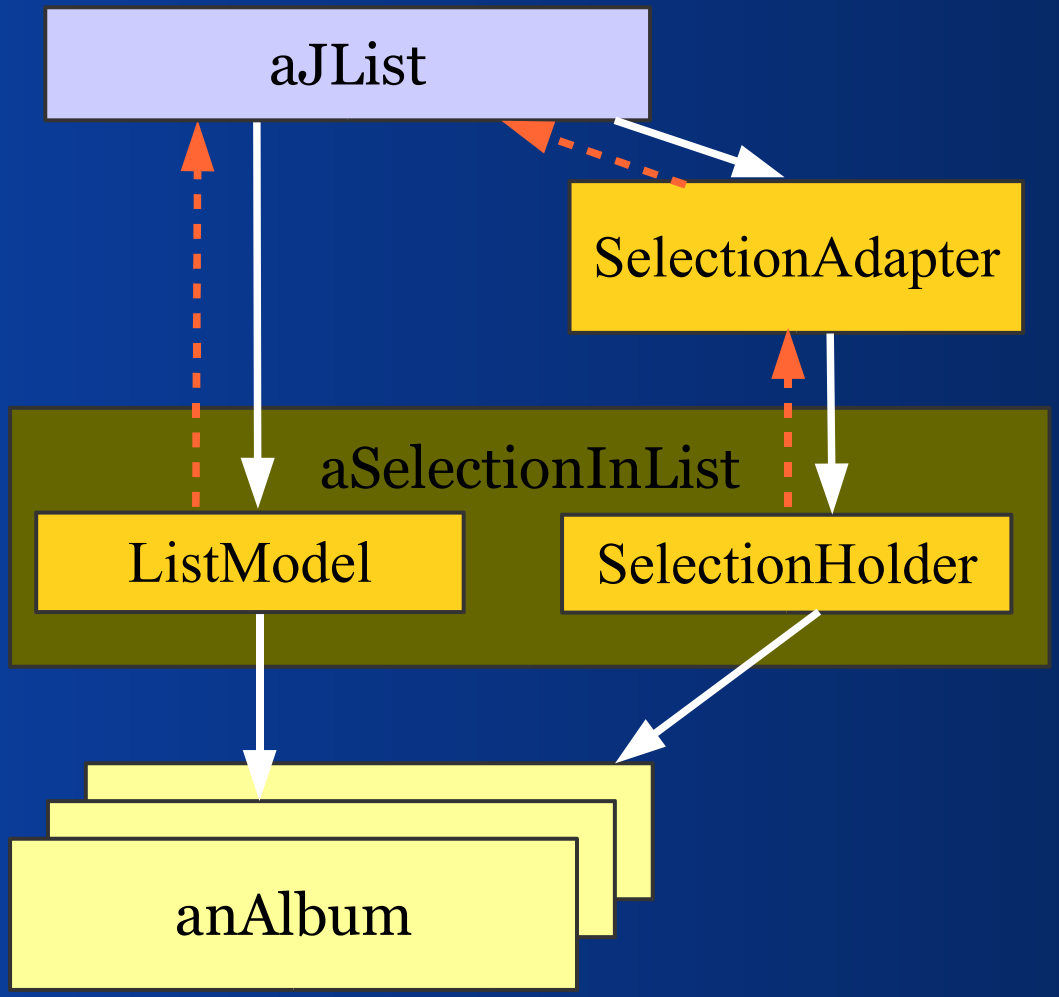
Binding Lists to JTable



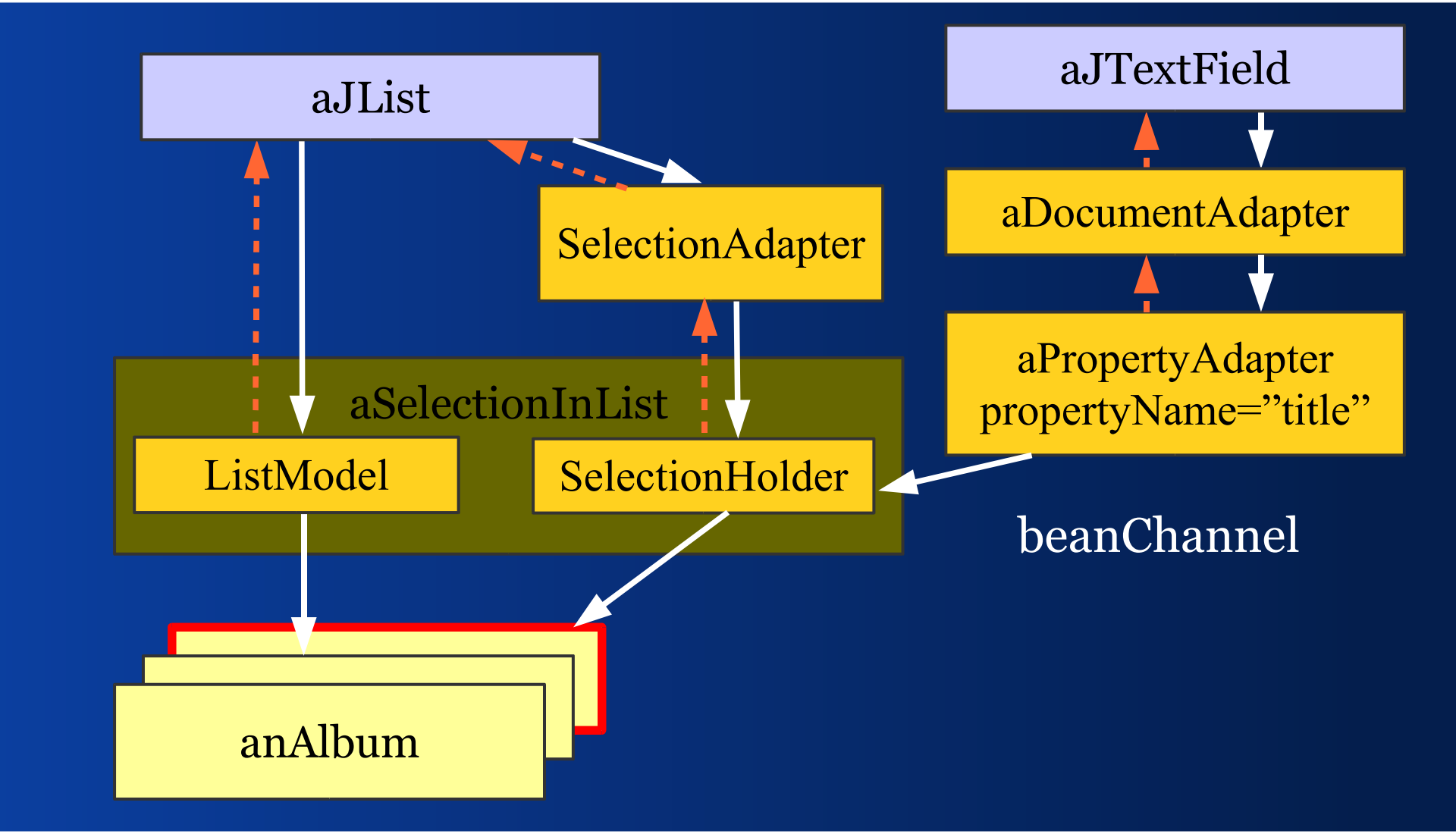
Often: List with Single Selection

- We build a compound model that holds the ListModel and a selection in the list.
- This model reports changes of:
  - the selection
  - the selection index
  - the list contents
  - the list

SelectionInList



Overview / Detail



IV - Architecture

A 3-tier Swing client architecture

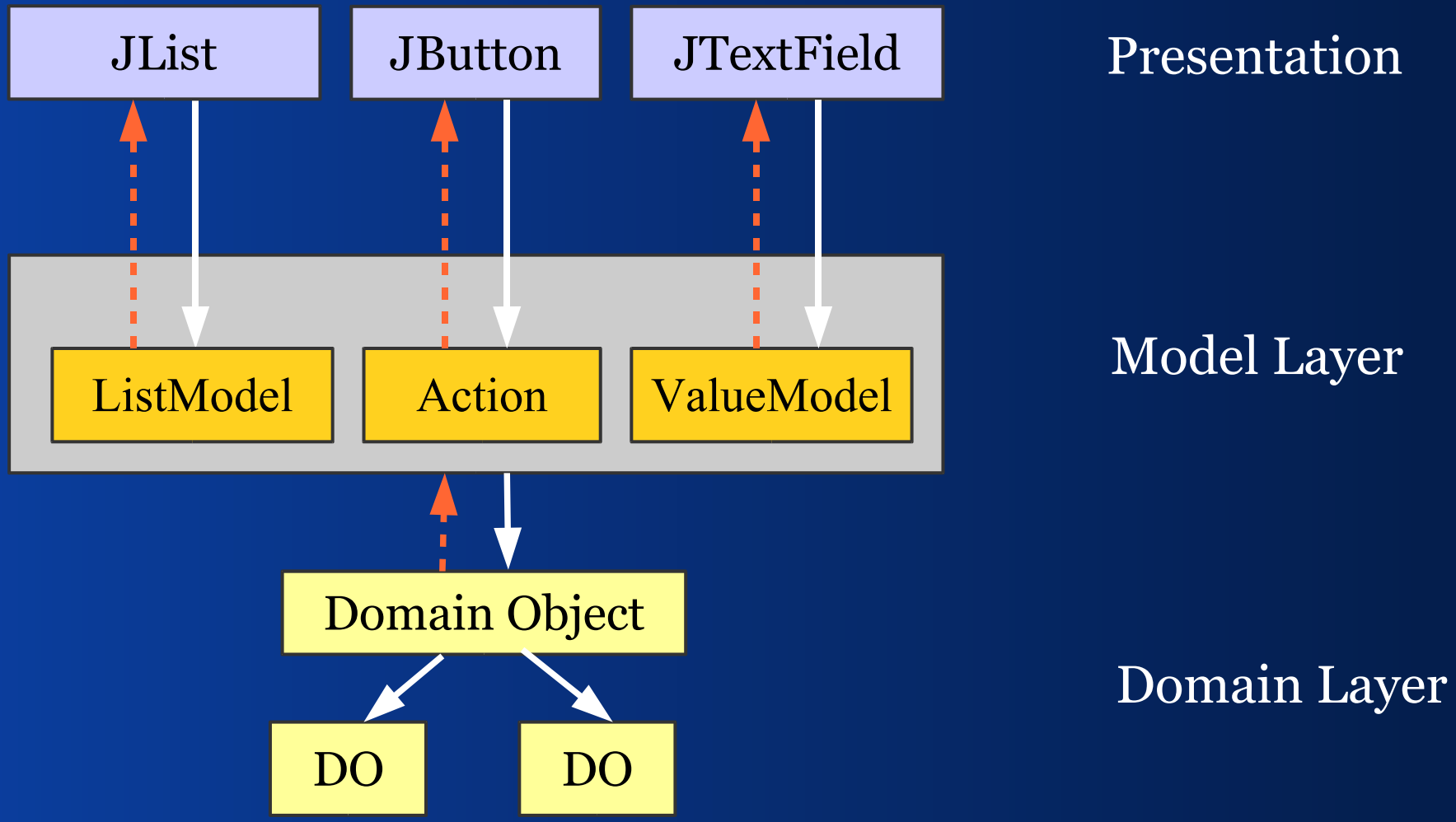
Design Goals

- Works with standard Swing components
- Works with custom Swing components

- Requires no special components
- Requires no special panels

- Integrates well with validation
- Works with different validation styles

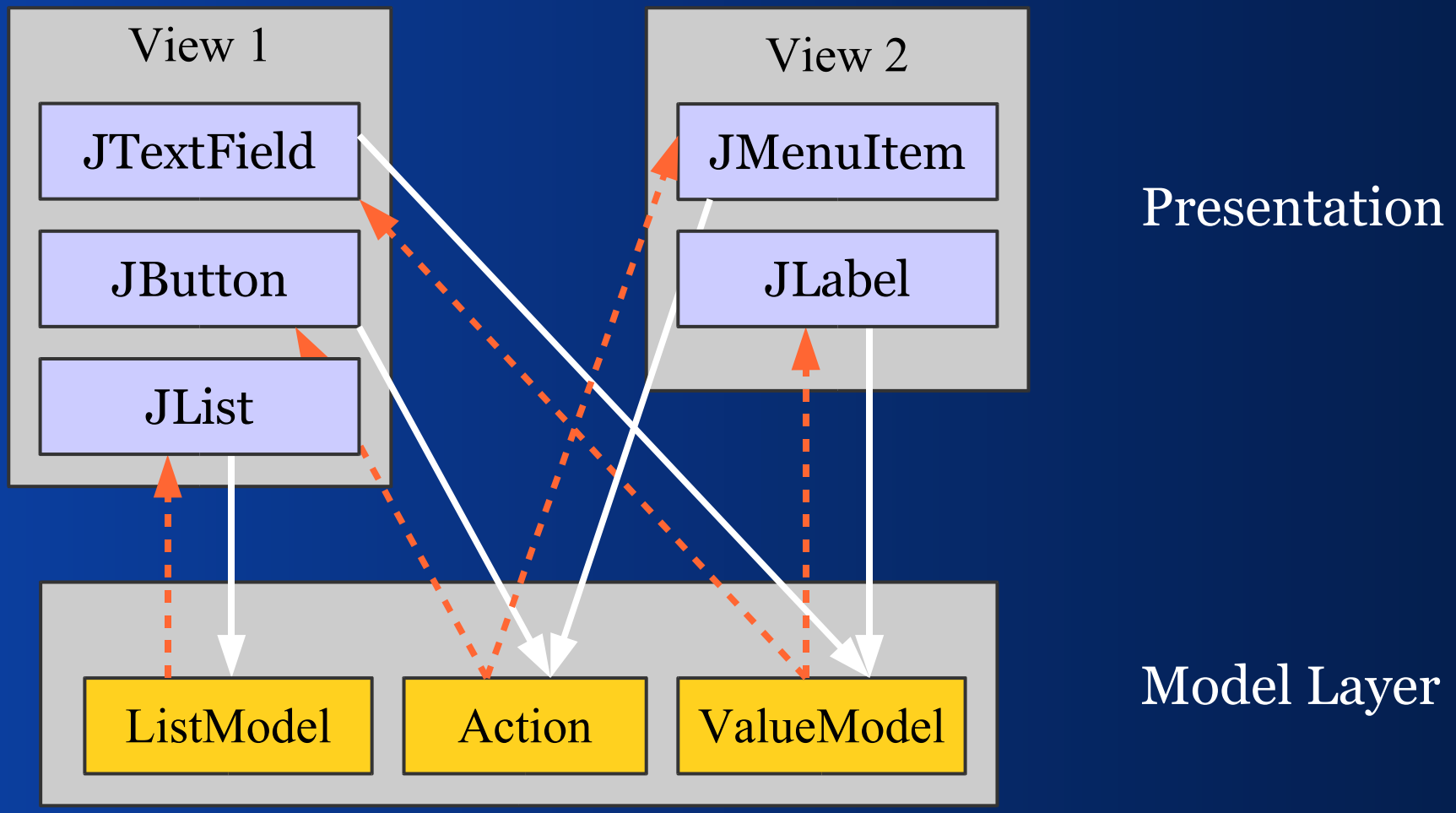
3-Tier Client Architecture



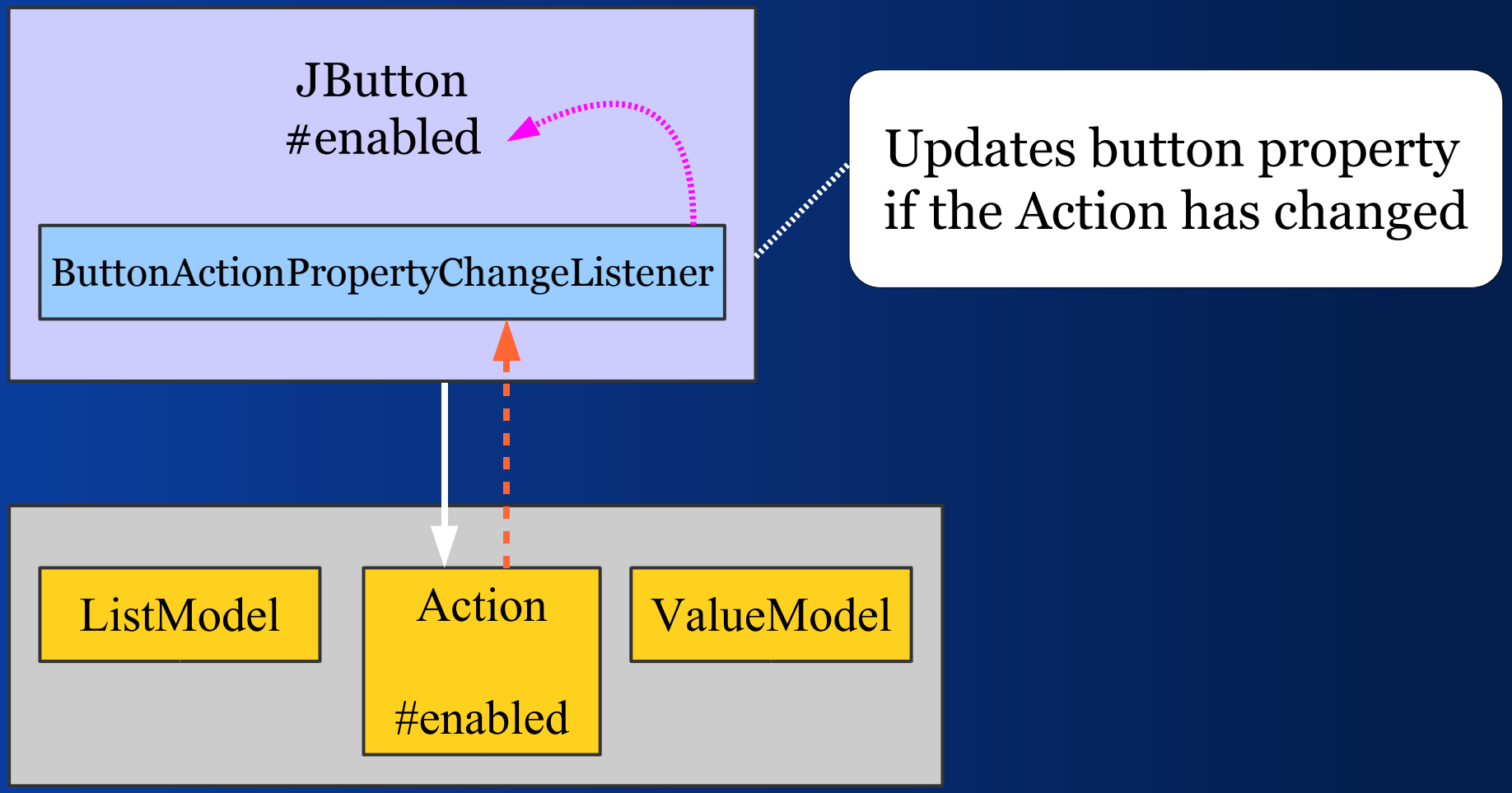
Benefit of 3 Layers

- Views are easy to build
- Views are decoupled
- Domain layer is separated
- Developers know where to put what code
- Synchronization is easy
- Decreased complexity
- Model operations located in a single layer
- Poor code limited to the model layer

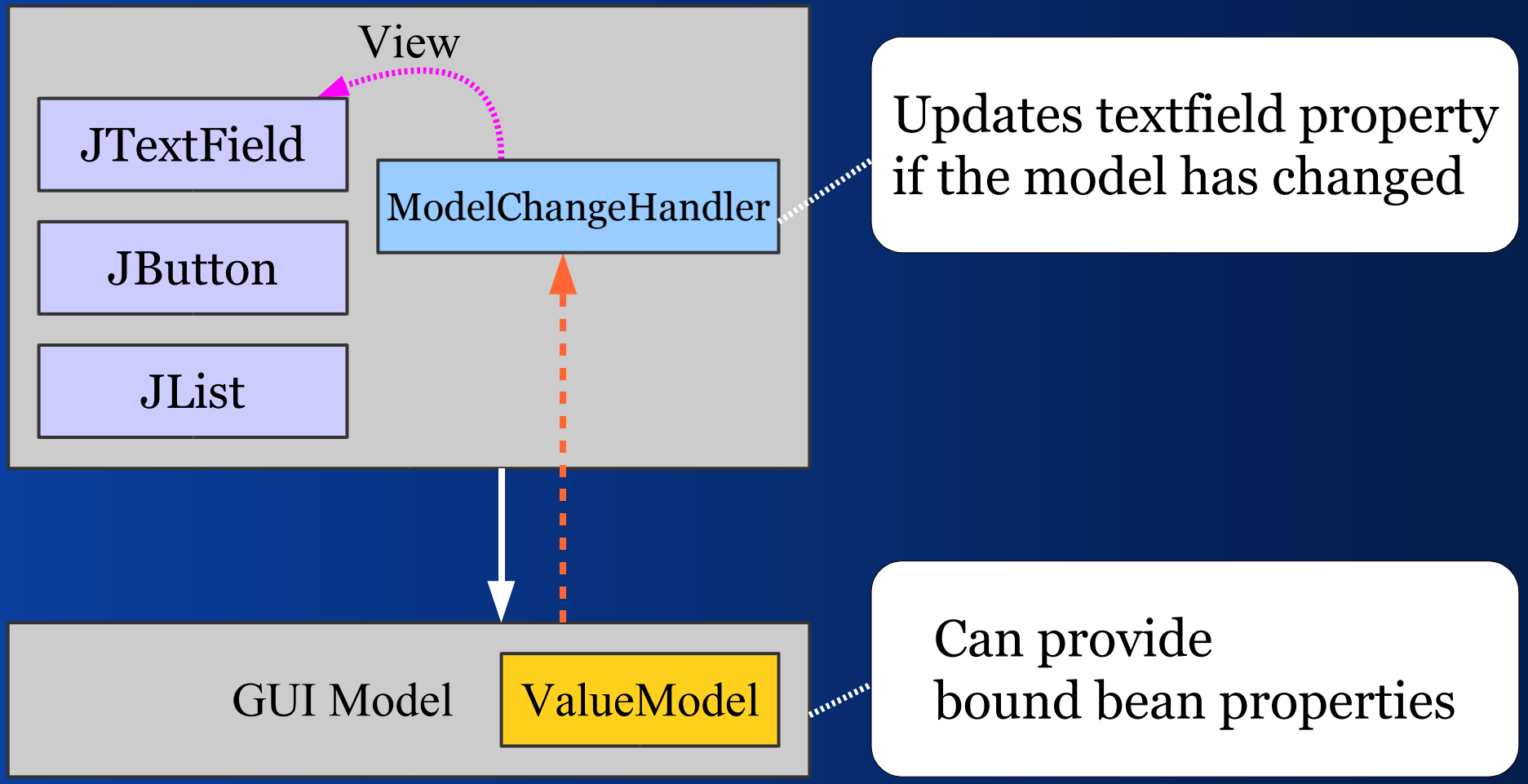
Multiple Views



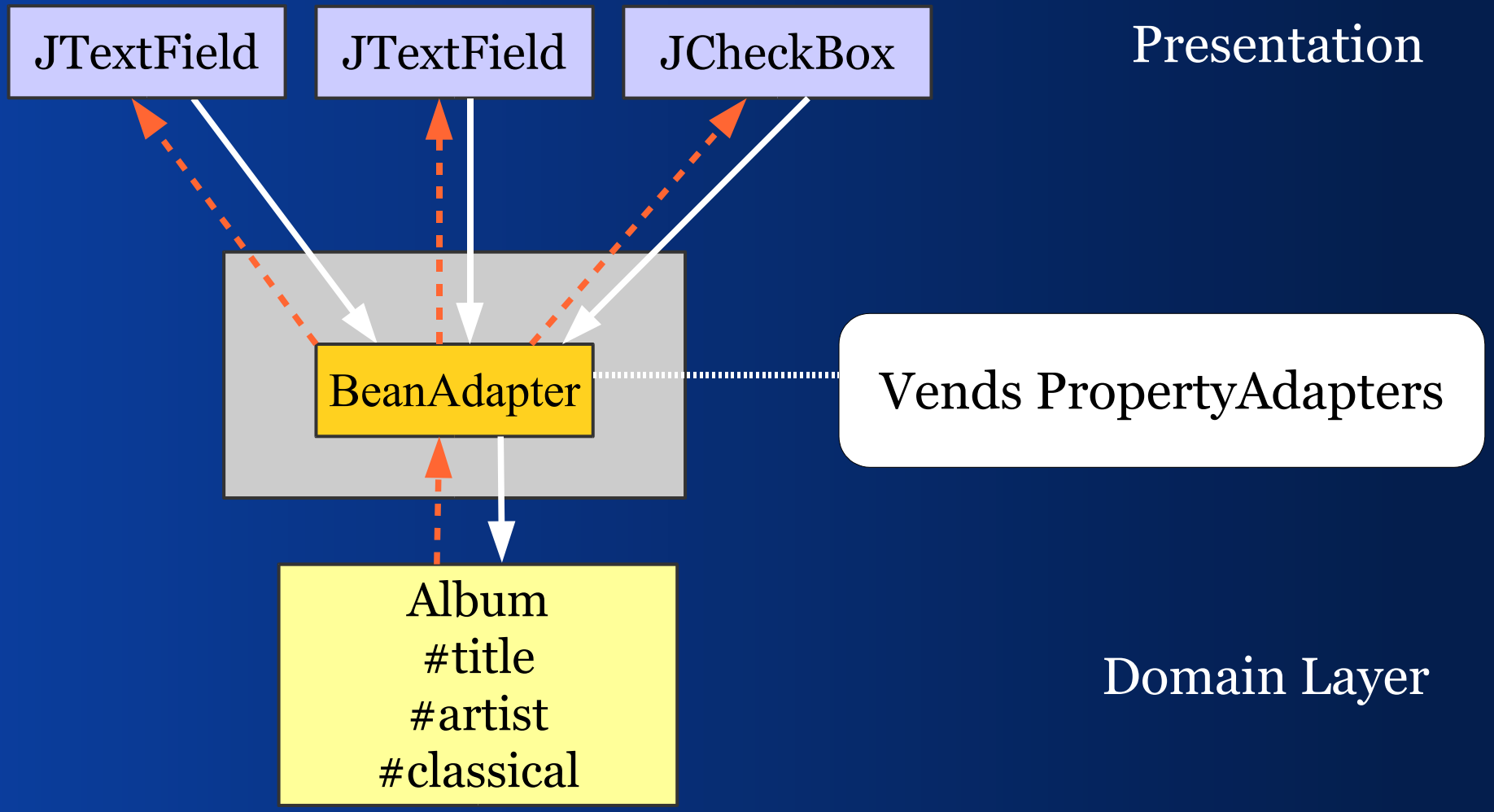
Setting UI Properties: Actions



Setting UI Properties



Adapting Multiple Properties



Example View Source Code

1) Variables for UI components

2) Constructors

3) Create, bind, configure UI components

4) Register GUI state handlers with the model

5) Build and return panel

6) Handlers that update GUI state

Example View 1/7

```
public final class AlbumView {

    // Refers to the model provider
    private AlbumPresentationModel model;

    // UI components
    private JTextField titleField;
    private JCheckBox  classicalBox;
    private JButton    buyNowButton;
    private JList      referencesList;
    ...
```

Example View 2/7

```
public AlbumView(AlbumPresentationModel m) {

    // Store a ref to the presentation model
    this.model = m;

    // Do some custom setup.
    ...
}
```

Example View 3/7

```
private void initComponents() {
    titleField = ComponentFactory.createField(
        model.getTitleModel());
    titleField.setEditable(false);

    buyNowButton = new JButton(
        model.getBuyNowAction());

    referenceList = new JList(
        model.getReferenceListModel());
    referenceList.setSelectionModel(
        model.getReferenceSelectionModel());
```

Example View 4/7

```
private initEventHandling(){
     // Observe the model to update GUI state
     model.addPropertyChangeListener(
         "composerEnabled",
         new ComposerEnablementHandler());
}
```

Example View 5/7

```
public JPanel buildPanel() {
    // Create, bind and configure components
    initComponents();

    // Register handlers that change UI state
    initEventHandling();

    FormLayout layout = new FormLayout(
        "right:pref, 3dlu, pref", // 3 columns
        "p, 3dlu, p");            // 3 rows

    ...
```

Example View 6/7

```
PanelBuilder builder =
    new PanelBuilder(layout);
CellConstraints cc = new CellConstraints();


builder.addLabel("Title",  cc.xy(1, 1));
builder.add(titleField,    cc.xy(3, 1));
builder.add(availableBox,  cc.xy(3, 3));
builder.add(buyNowButton,  cc.xy(3, 5));
builder.add(referenceList, cc.xy(3, 7));

return builder.getPanel();
}
```

Example View 7/7

```
/* Listens to #composerEnabled,
   changes #enabled of the composerField.   */
private class ComposerEnablementHandler
    implements PropertyChangeListener {

    public void propertyChange(
        PropertyChangeEvent evt) {

        composerField.setEnabled(
            model.isComposerEnabled());
    }
}
```

Simpler Event Handling

```
private initEventHandling(){
     // Synchronize model with GUI state
     PropertyConnector.connect(
        model,          "composerEnabled",
        composerField, "enabled");
}
```

V - Field Report

How does Adapter Binding work?

Costs

- Adapter Binding:
  - increases learning costs
  - decreases production costs a little
  - can significantly reduce change costs

Use a ComponentFactory!

- Encapsulate the creation of adapters from ValueModel to Swing components.
- Some components have no appropriate model, e. g. JFormattedTextField
- Vends components for ValueModels

Buffering

- Use BufferedValueModel judicously
  - prevents validation on domain models
  - makes it harder to use domain logic
- The client domain layer can buffer if:
  - domain objects are copies
  - domain objects temporarily accept invalid data

Performance

- Adapter chains fire many change events
- That seems to be no performance problem

- ListModel can improve the performance compared to copying list contents

Debugging

- Copying approach is easy to debug;
  you can see when where what happens.
- Adapter chains “move“ values implicitly;
  it's harder to understand updates.
- Reflection and Introspection hide
  who reads and writes values.
- Favor named over anonymous listeners.

Renaming Methods

- Reflection and Introspection make it more difficult to rename bean properties and their getter and setters.
- Use constants for bean property names!
- Obfuscators fail to detect the call graph.

When is Binding Useful?

- I guess that adapter binding can be applied to about 80% of all Swing projects.
- However, you need at least one expert who masters the binding classes.

Benefits of Adapter Binding

- Adapter binding can save a lot of code.
- Code is easier to read.
- Helps you separate code into layers.
- Can significantly reduce the complexity.

Where does Binding stand?

- Approach is 10 years old and stable.
- Architecture of the Java port is stable.
- Tests cover 90% of the classes.
- Little documentation.
- Tutorial is quite small.

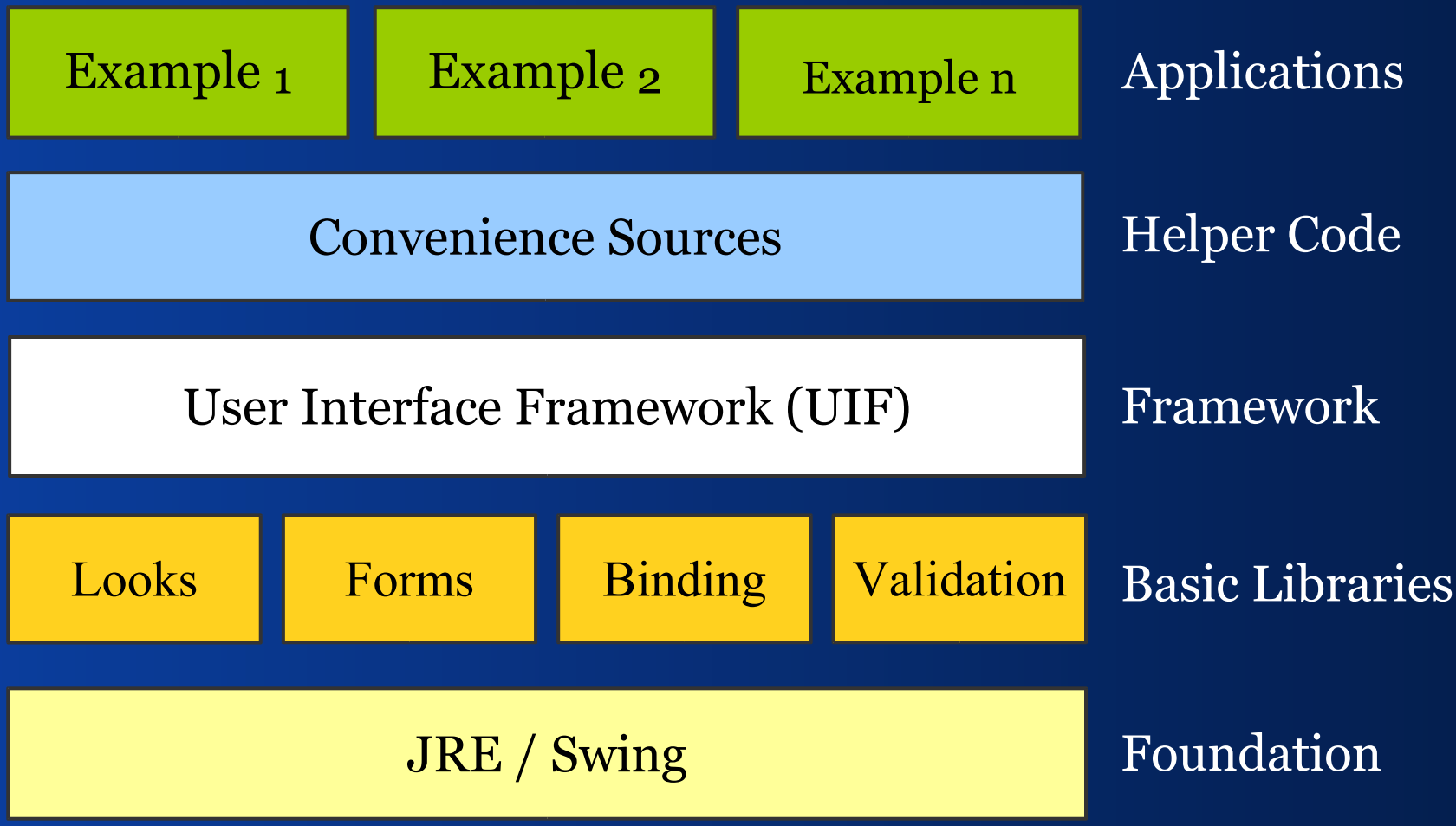
End

Summary and References

Summary

- We've learned about MVC and Swing
- We've identified Binding tasks
- We've motivated the ValueModel interface
- We've learned how to bind single values
- We've learned how to bind lists
- We've seen a 3-tier architecture

JGoodies Swing Suite



References I

- Fowler's Enterprise Patterns
  martinfowler.com/eaaDev/
- JGoodies Binding
  binding.dev.java.net
- JGoodies Articles
  www.JGoodies.com/articles/
- JGoodies Demos
  www.JGoodies.com/freeware/

References II

- Sun's JDNC
  jdnc.dev.java.net
- Understanding and Using ValueModels
  c2.com/ppr/vmodels.html
- Oracle's JClient and ADF
  otn.oracle.com/, search for 'JClient'
- Spring Rich Client Project
  www.springframework.org/spring-rcp.html

Demo/Tutorial:

JGoodies Binding Tutorial

Binding Problems and Solutions

(in progress)

Ships with the JGoodies Binding library.

Questions and Answers

End

Good Luck!