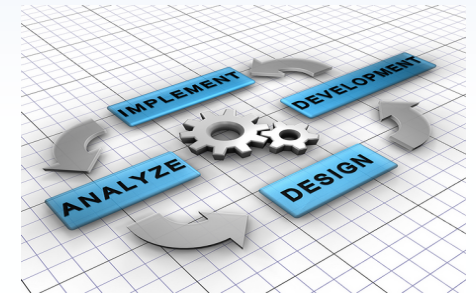
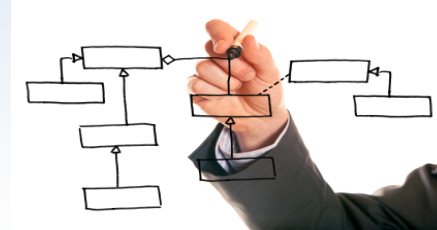


# **Developing Reusable Software Components using MVP, Observer and Mediator Patterns**

# Who is Kenan Sevindik?

- Over **15 years** enterprise software development experience
- Involved in **developing architectures** for various projects in nationwide
- Has **extensive knowledge and experience** about several enterprise Java technologies, such as Spring, Spring Security, Hibernate, Vaadin



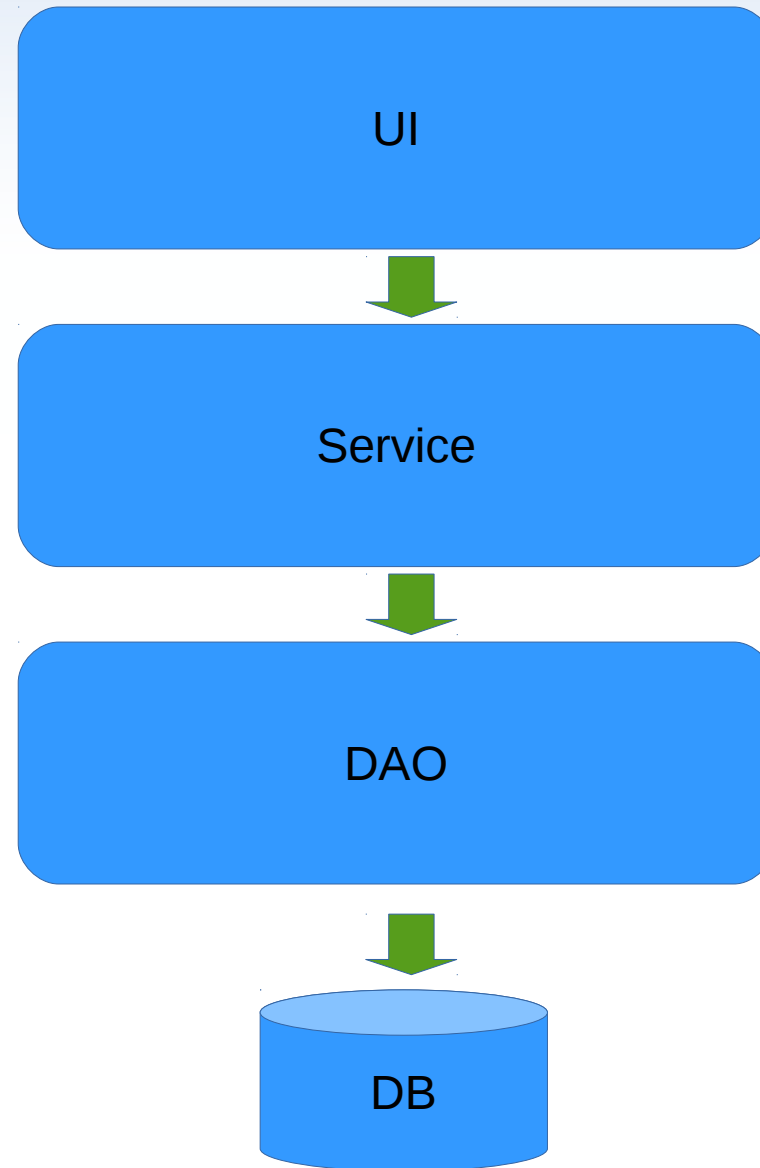


# Who is Kenan Sevindik?

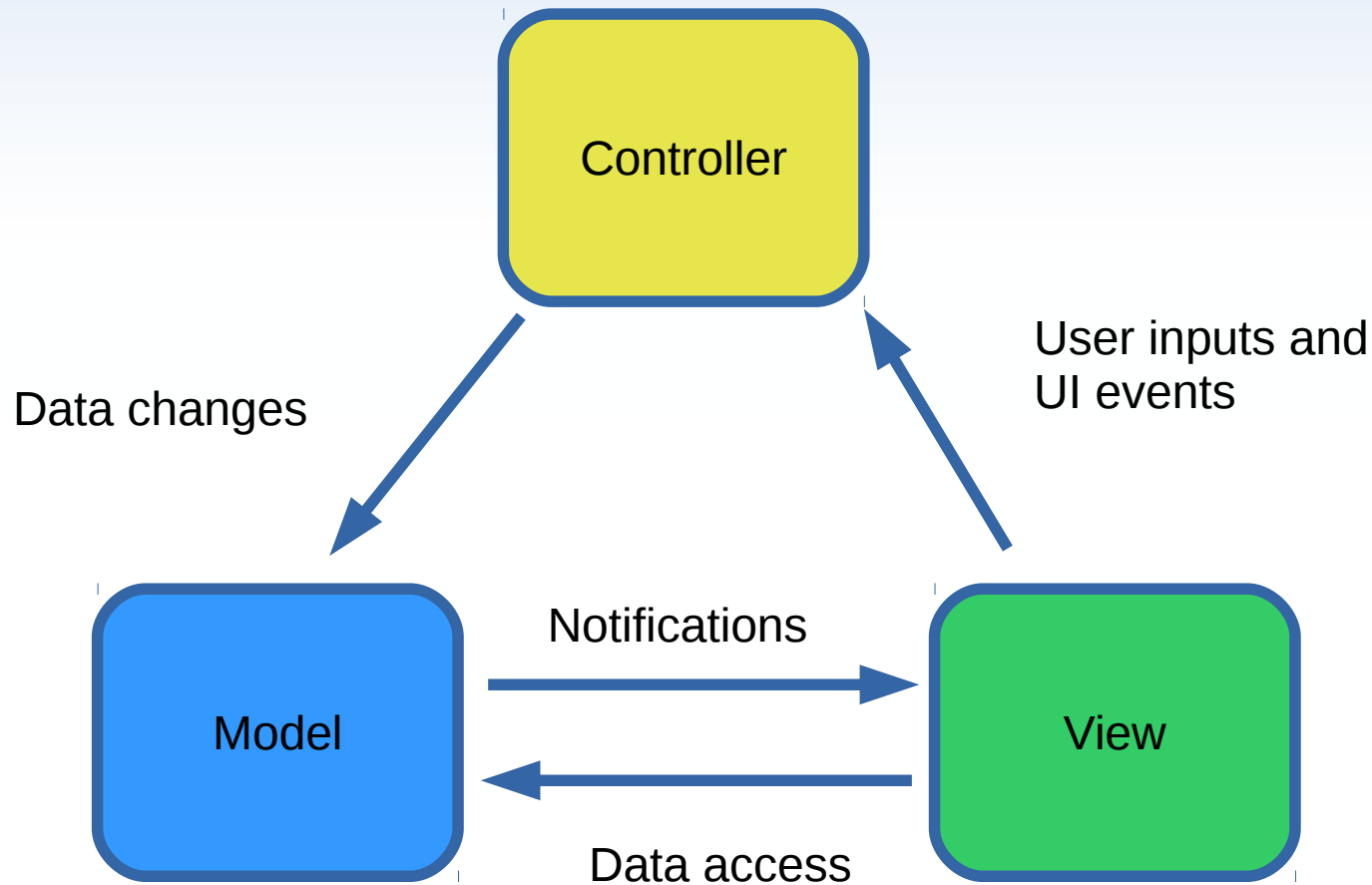
- Author of **Beginning Spring Book** published by Wiley in 2015
- Founded **Harezmi IT Solutions** in 2011
- Focused on **enterprise software development**
- Offering **software development consultancy and mentoring services**
- Organizing trainings under the brand name of “**Enterprise Java Trainings**”



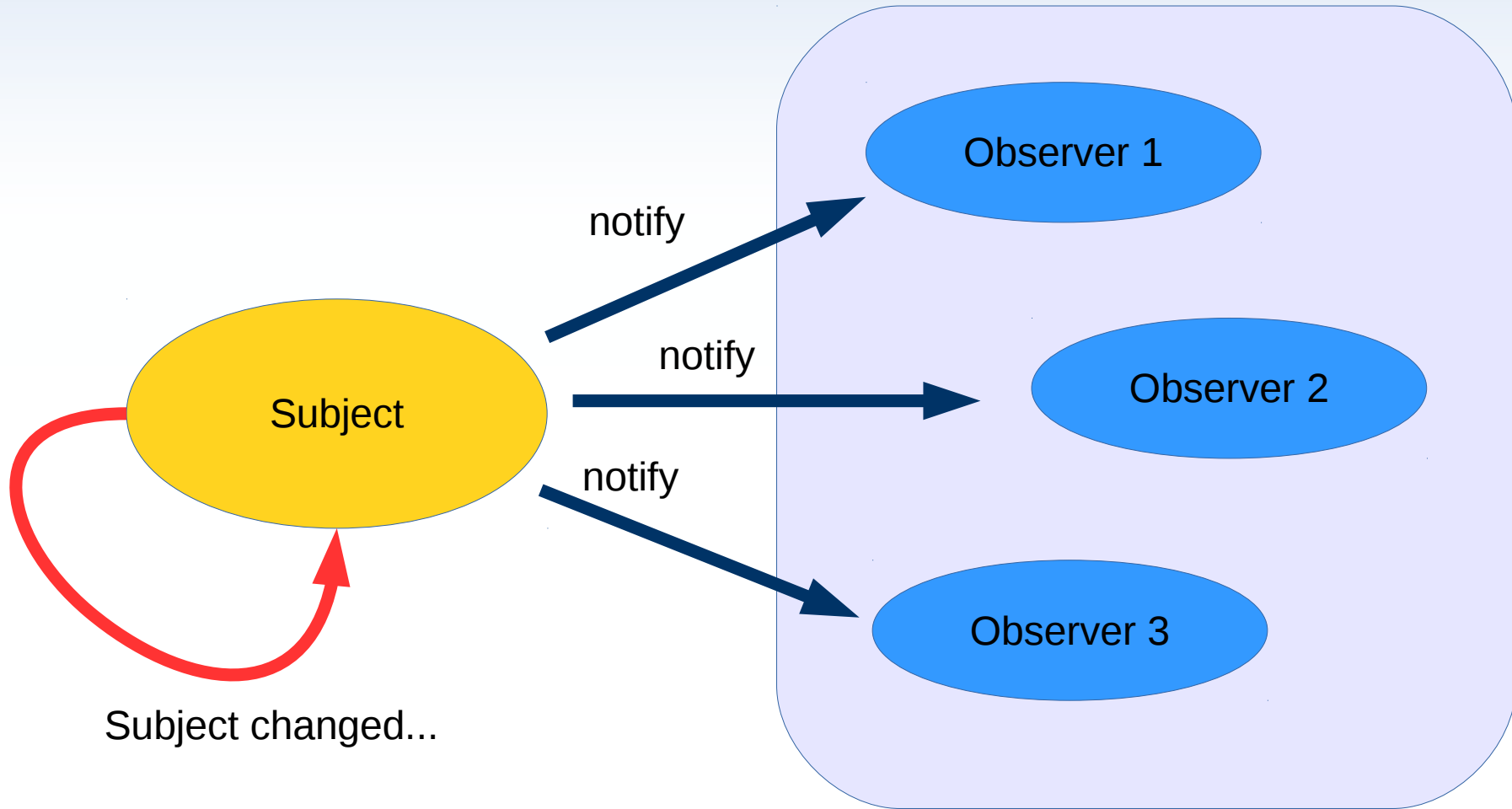
# Layered Architecture



# An Architectural Pattern: MVC



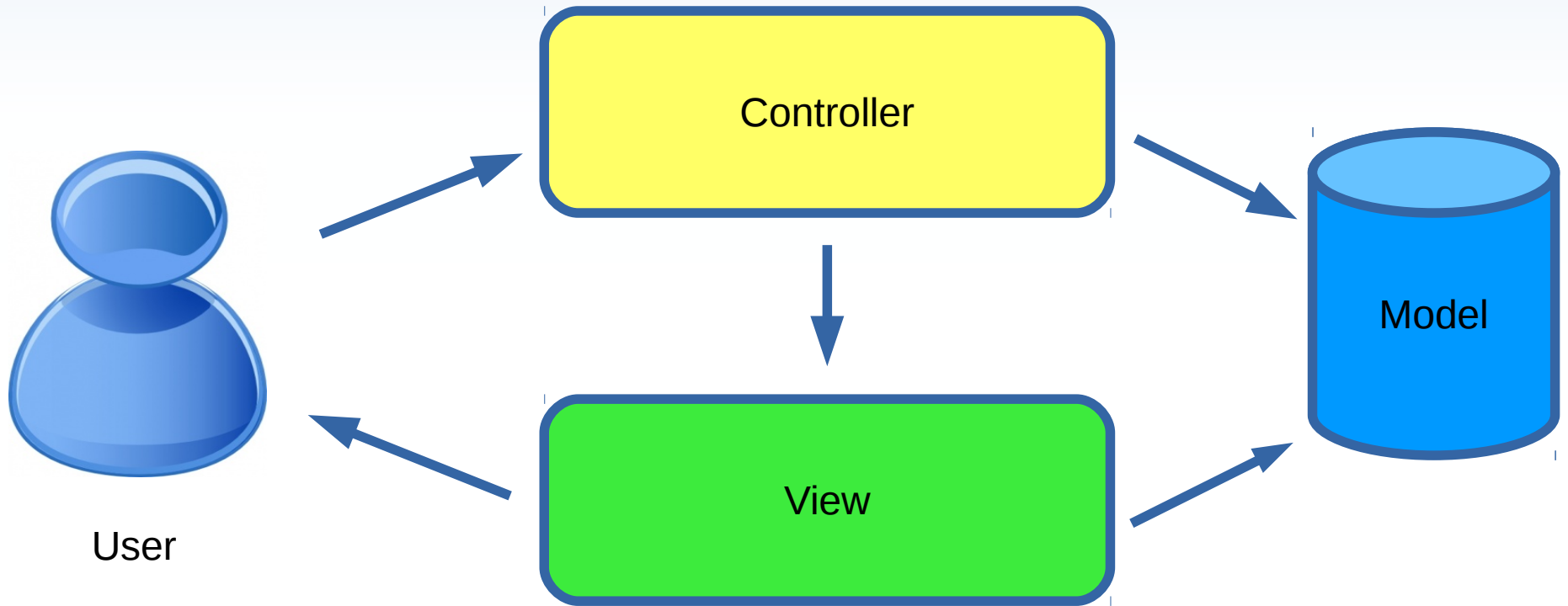
# MVC & Observer



# Fundamental Role of MVC

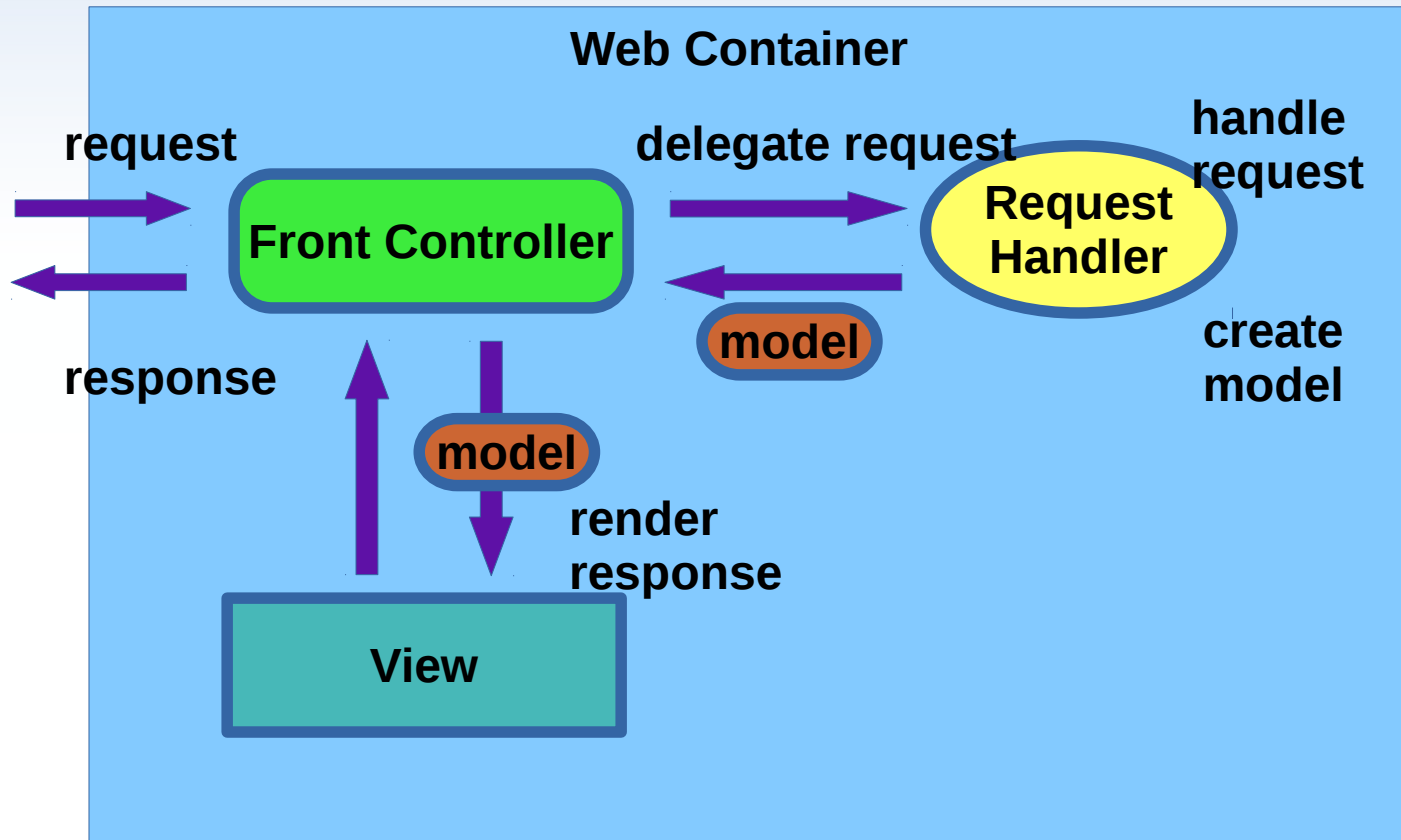
**“Seperation of Concern”**

# Current MVC Interpretation

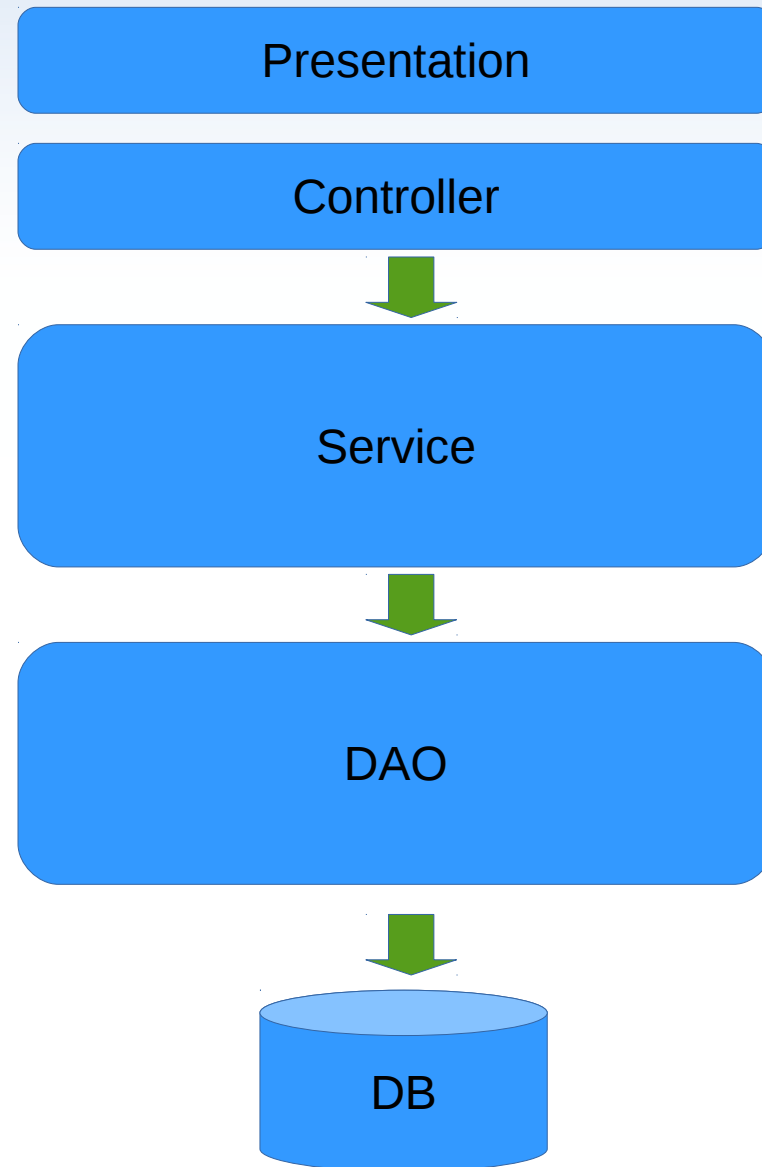




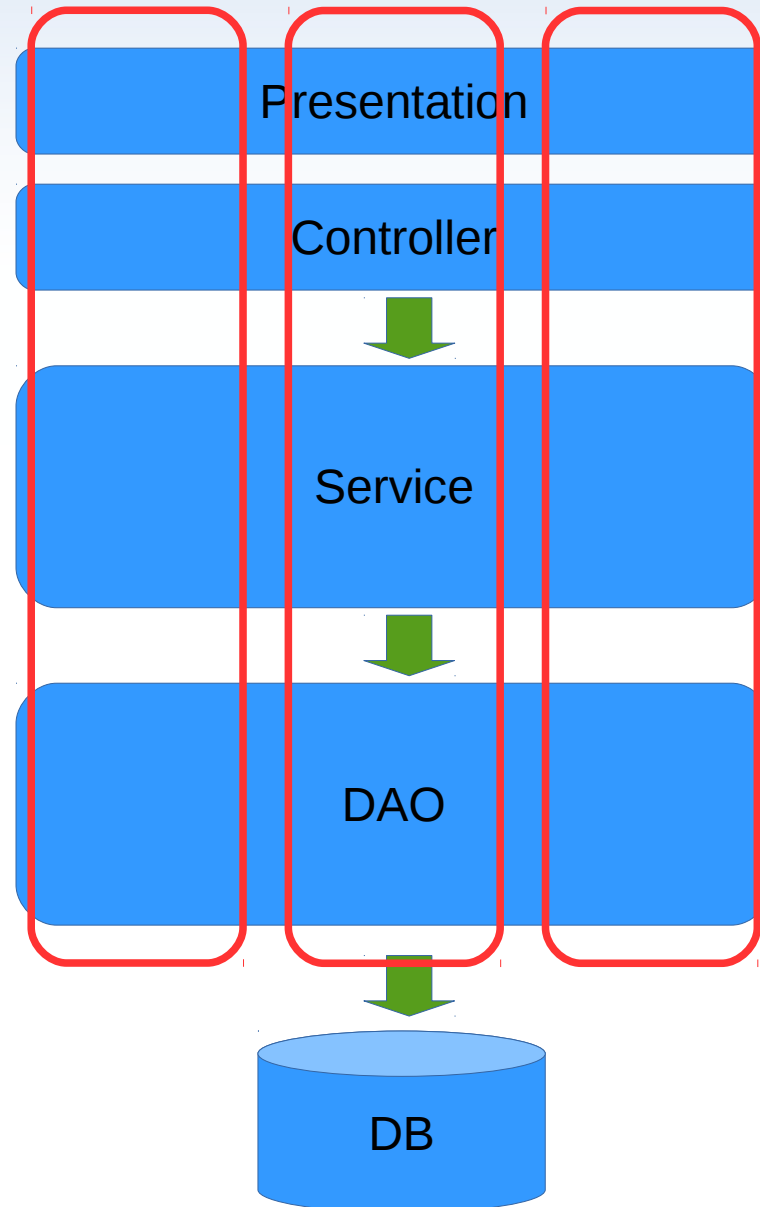
# Front Controller



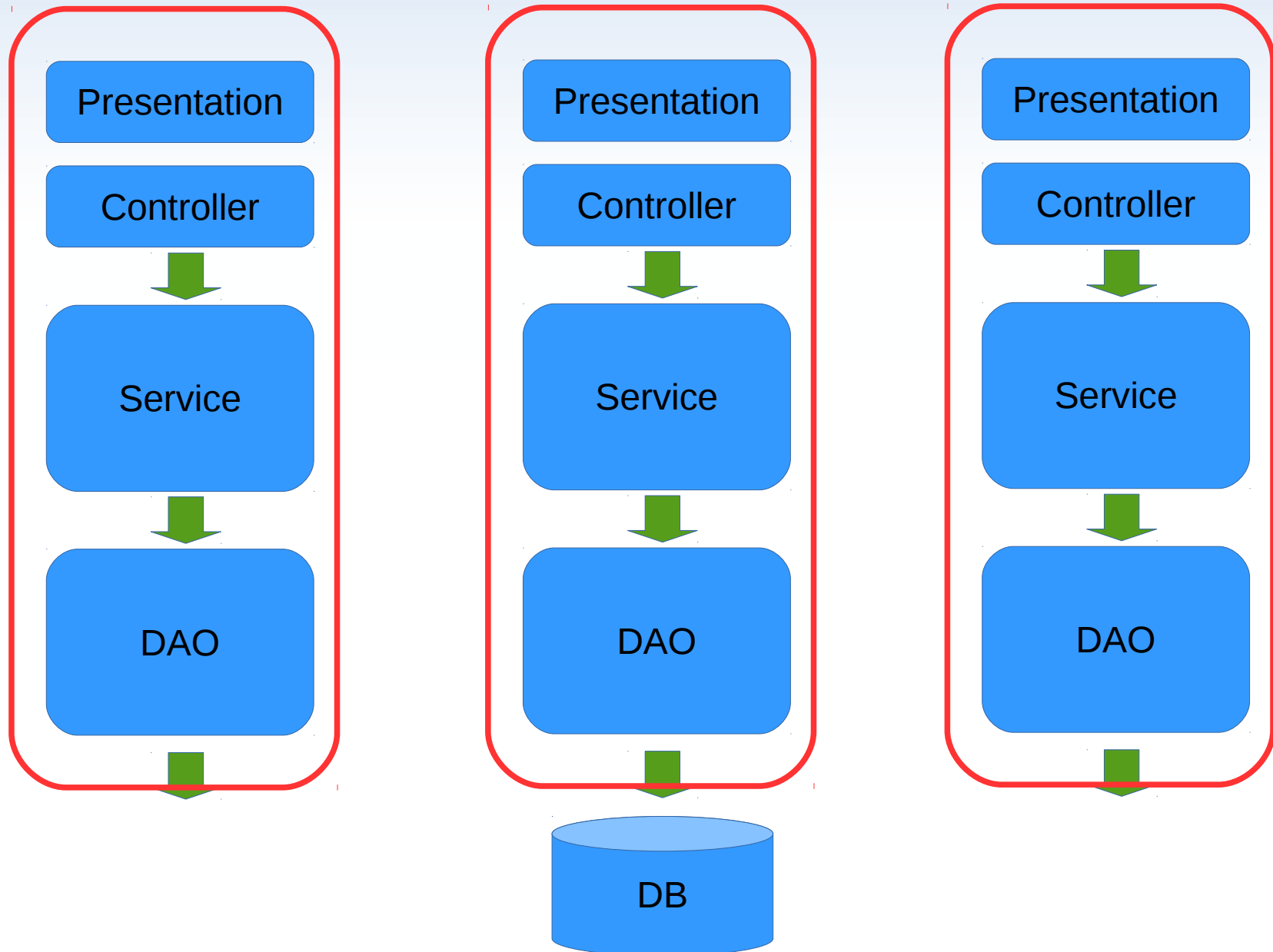
# Layered Architecture



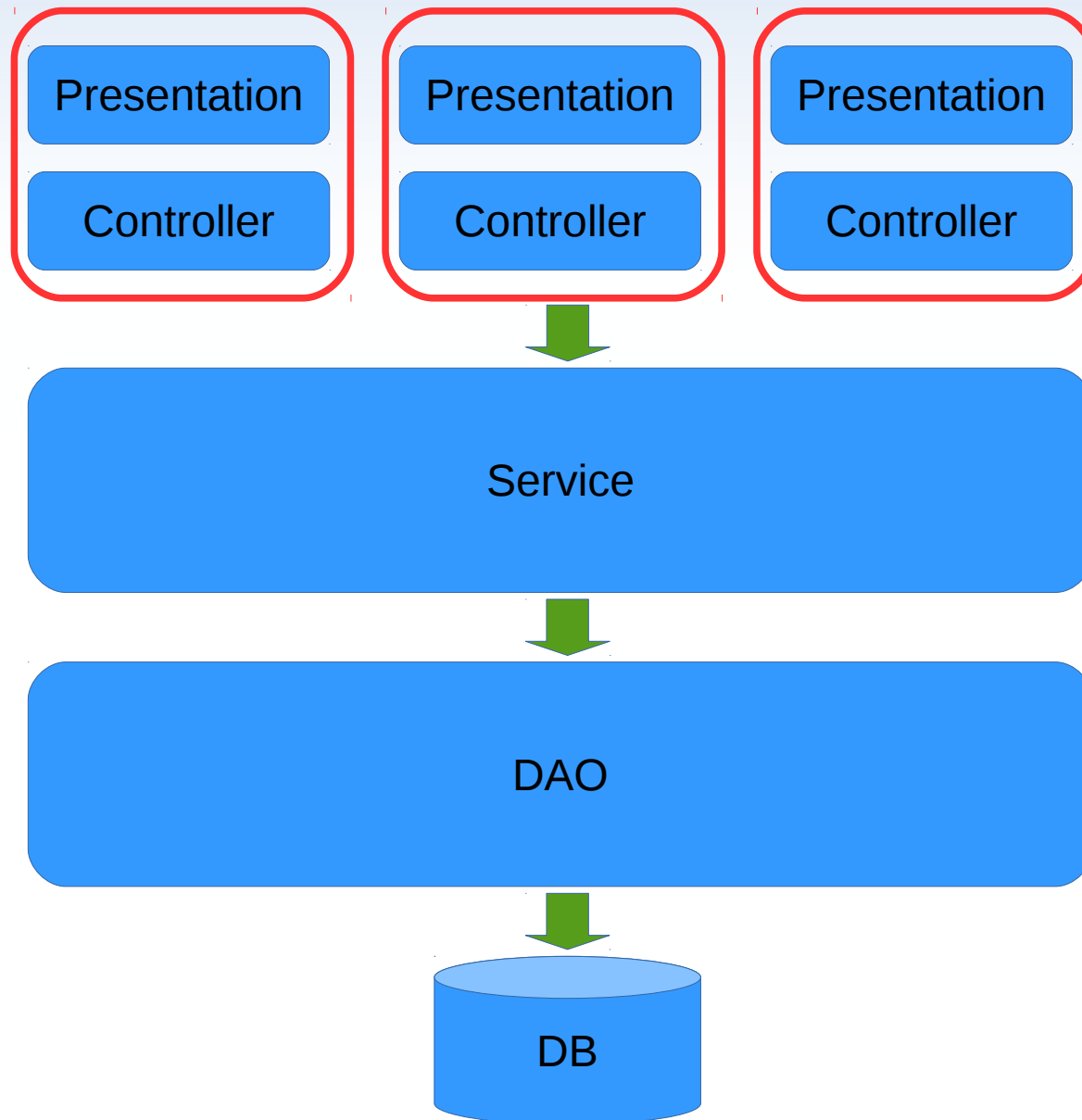
# Layered Architecture and Modularity



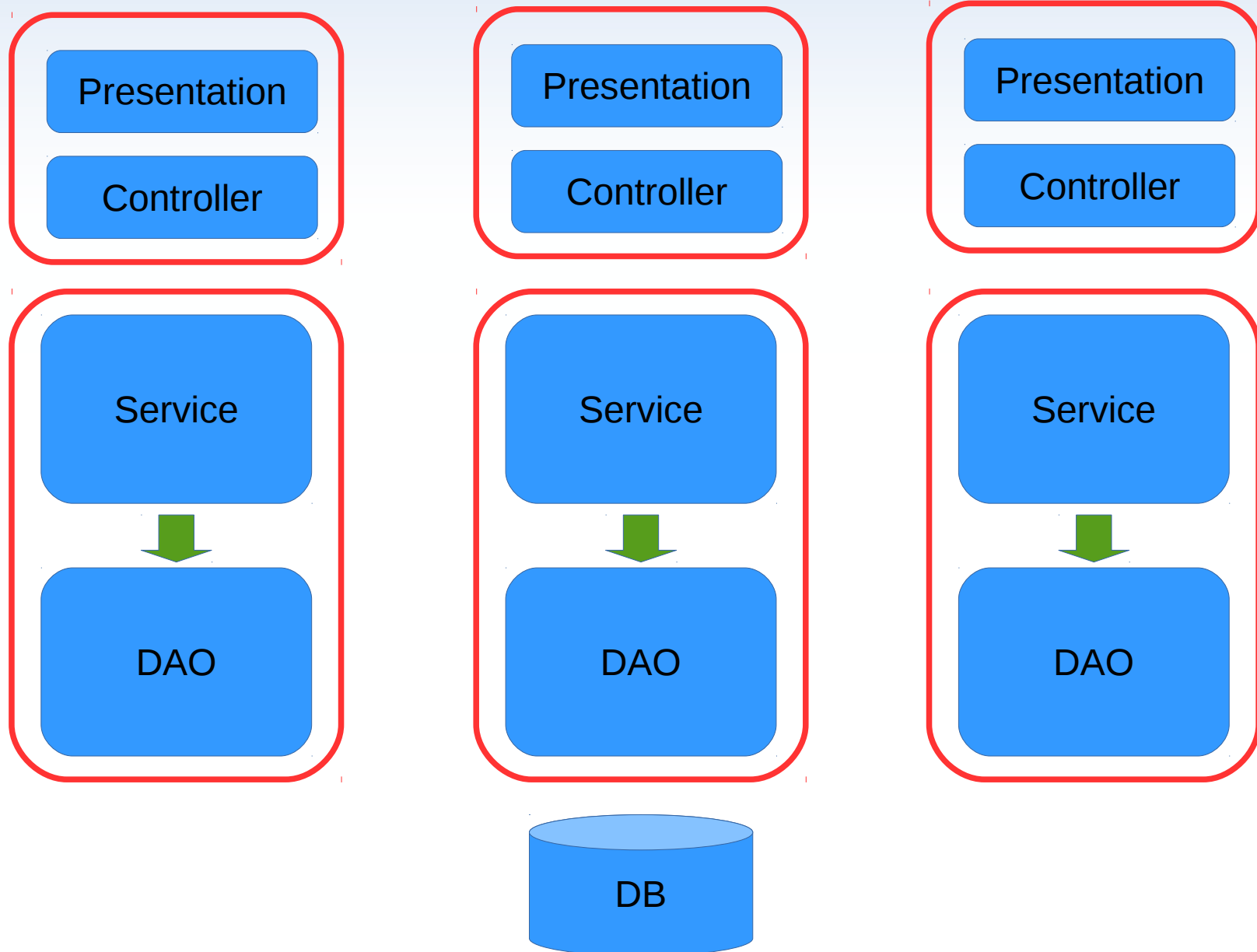
# Layered Architecture and Modularity



# Layered Architecture and Modularity

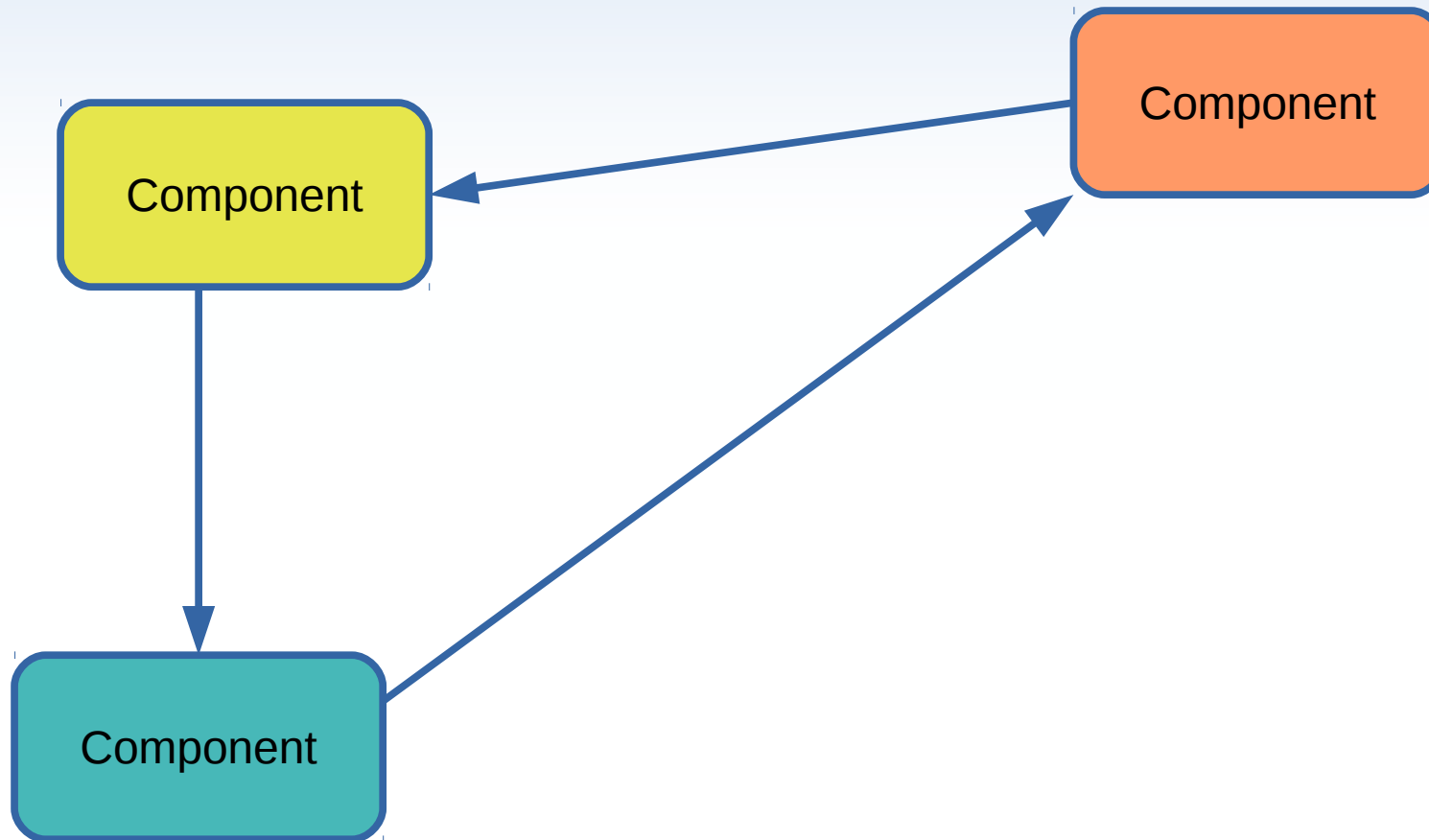


# Layered Architecture and Modularity

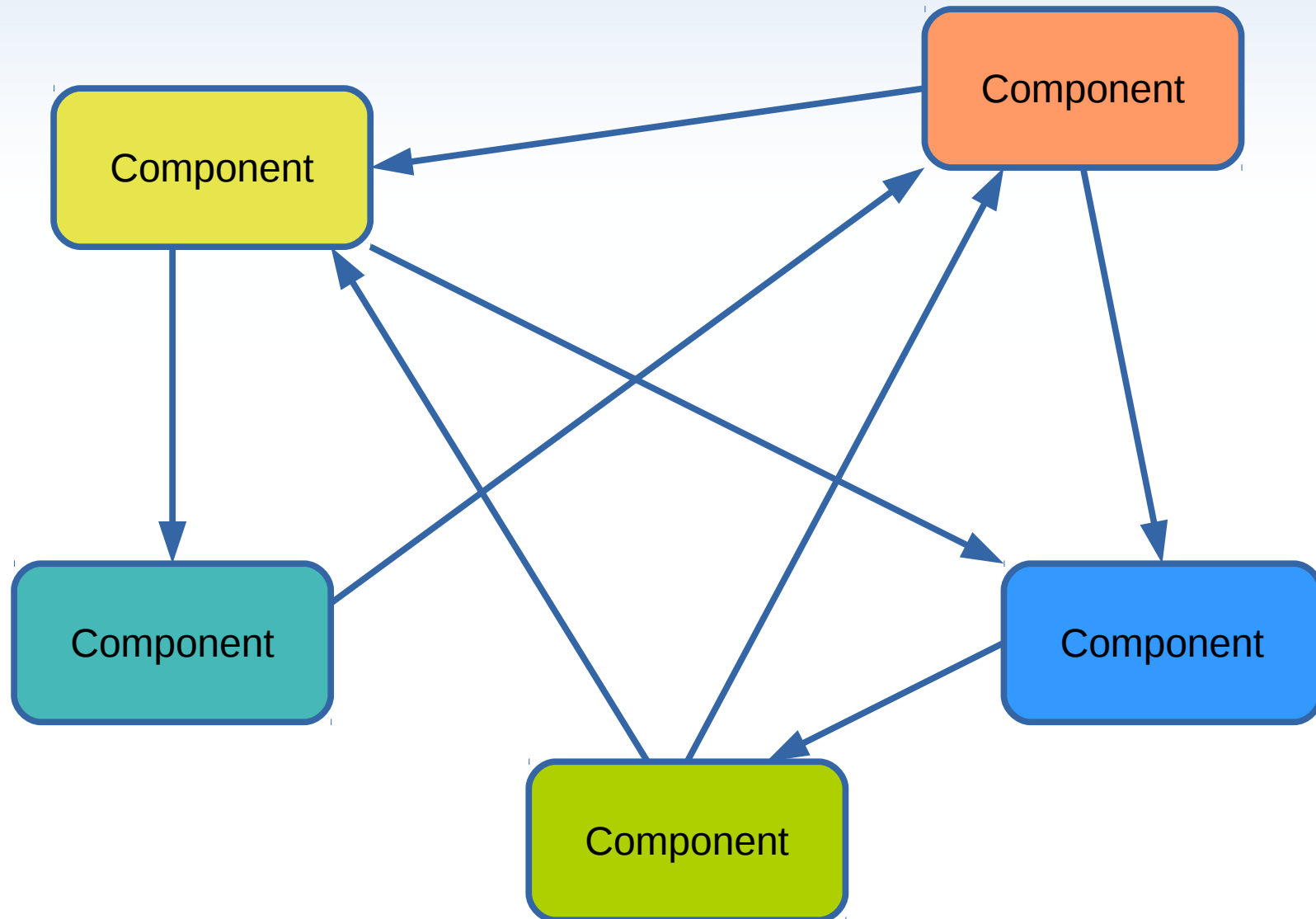




# Interactions Among Software Components



# Interactions Among Software Components



# Problems with MVC

- MVC helps **modularization of a system** in terms of business functionality
- However, it **doesn't lay a clear path** about how to transform user interactions into functional behaviour
- It **cannot organize interactions** among components and cannot remove dependencies among components either

# Solution: MVP + Mediator

- A variation of **MVP** helps us to handle UI rendering and functional behaviour separately
- **Mediator**, on the other hand, orchestrates interactions among components, and removes dependencies among themselves

# Model View Presenter



View

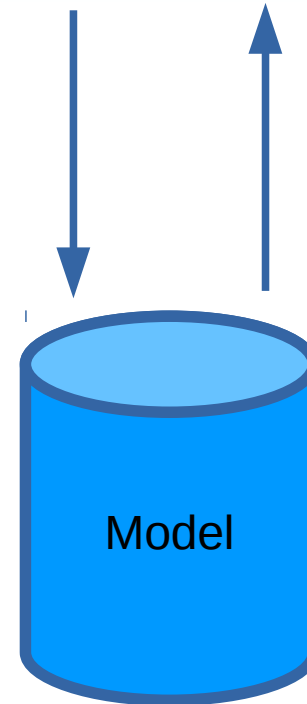
UI events are transformed into business events specific to application



Presenter reflect changes to UI by calling necessary functions and passing necessary data into it

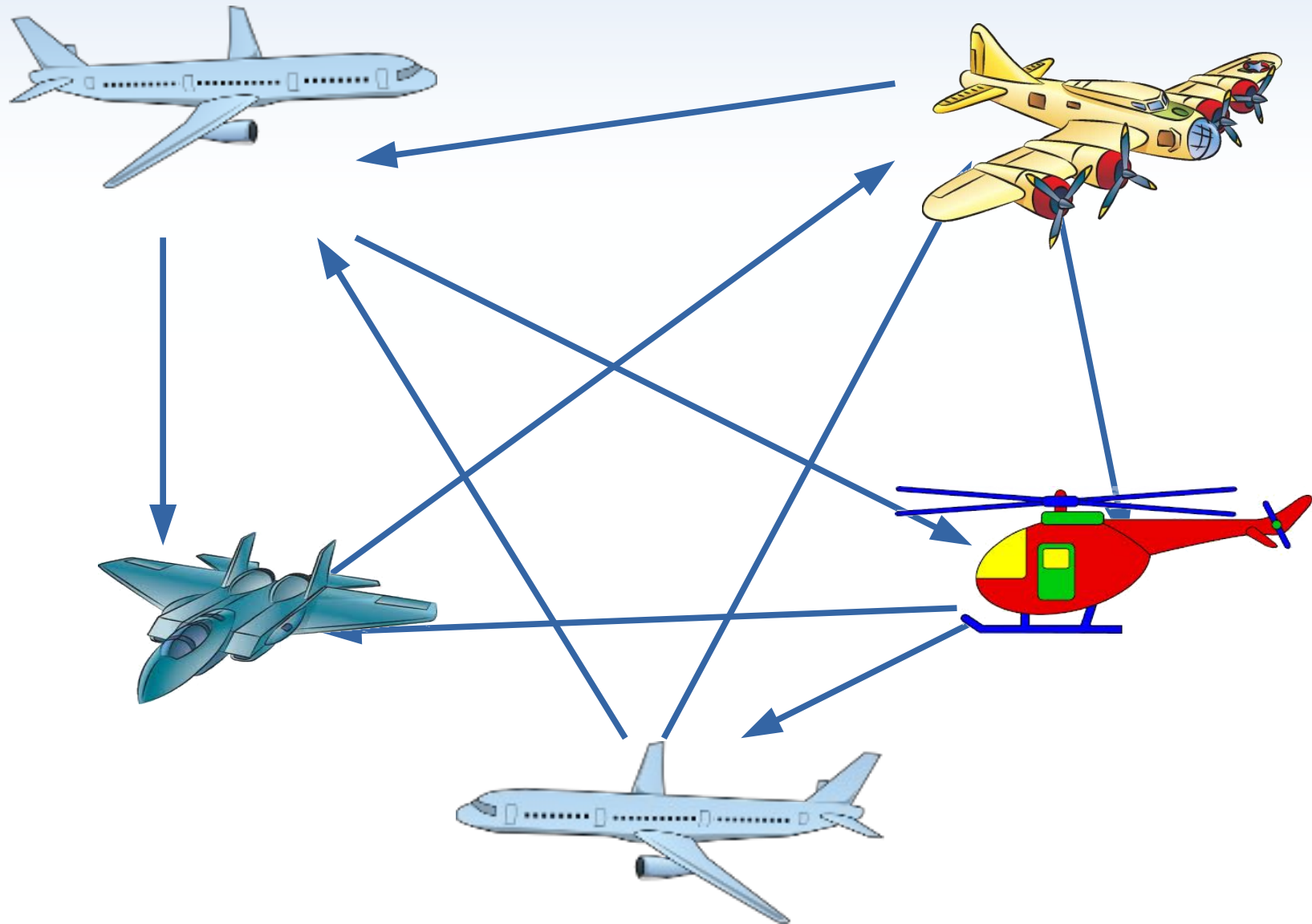
Presenter can update Model

Presenter can read Model



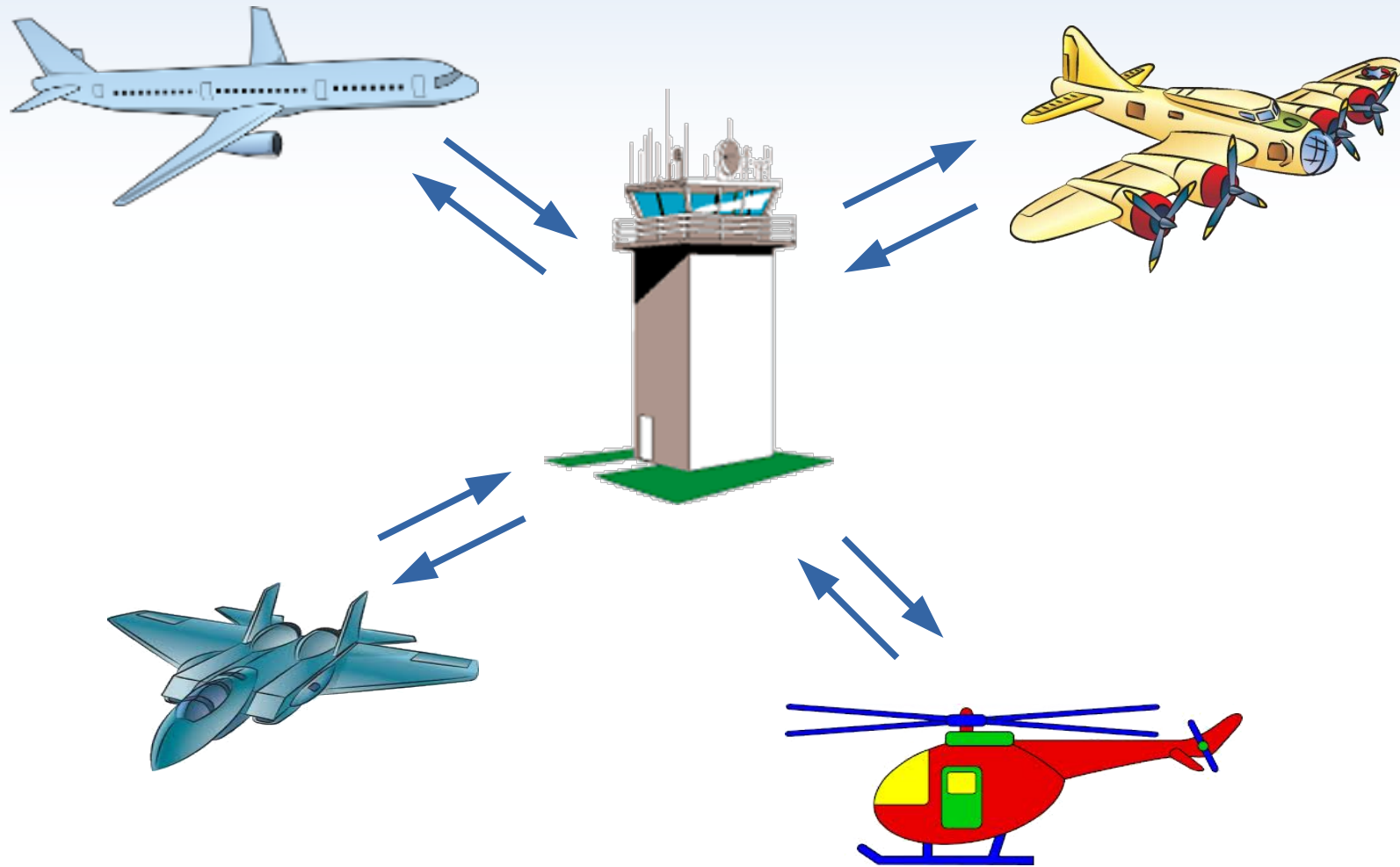
Changes on model are notified via events to Presenter

# Interactions Among Software Components

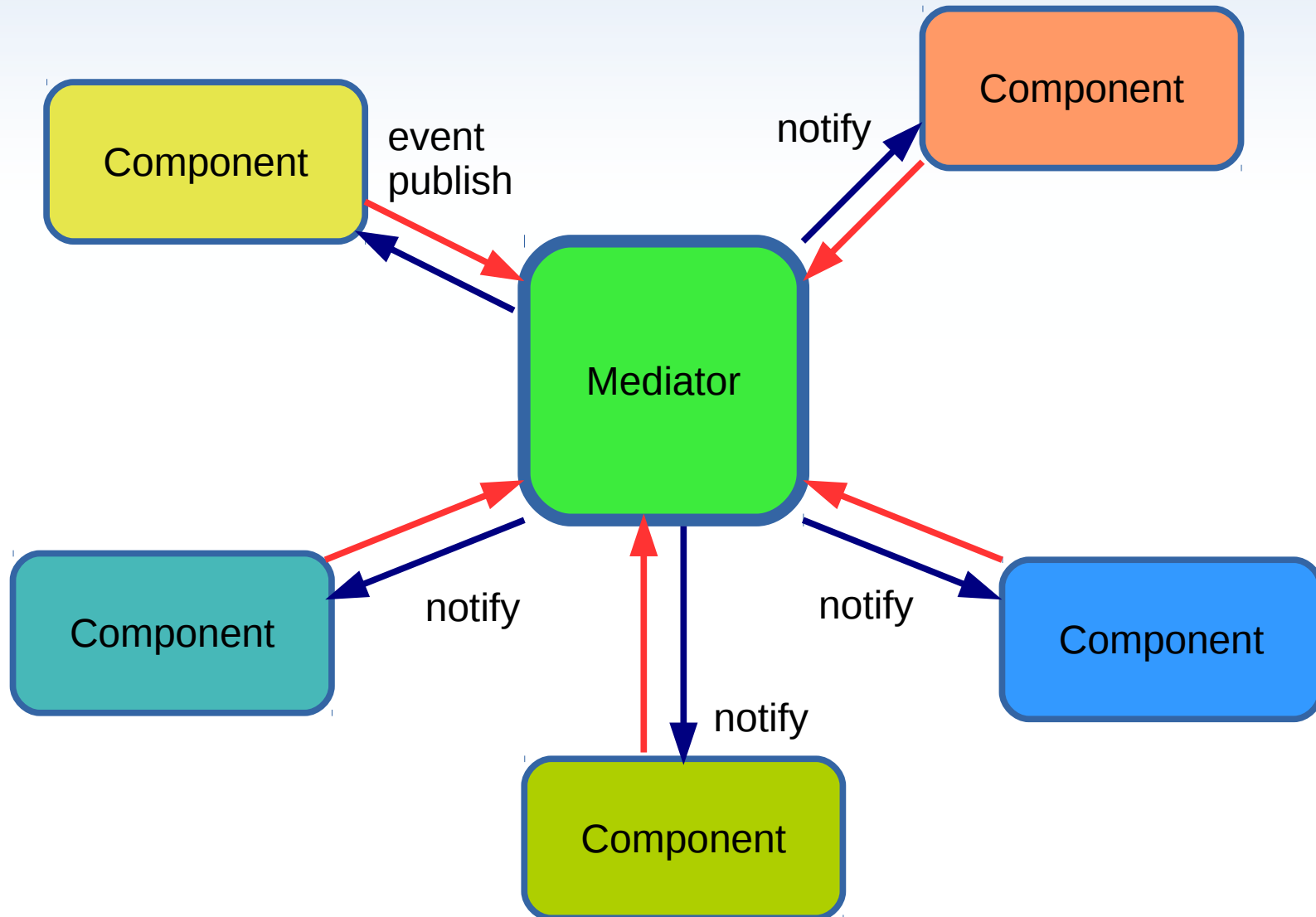




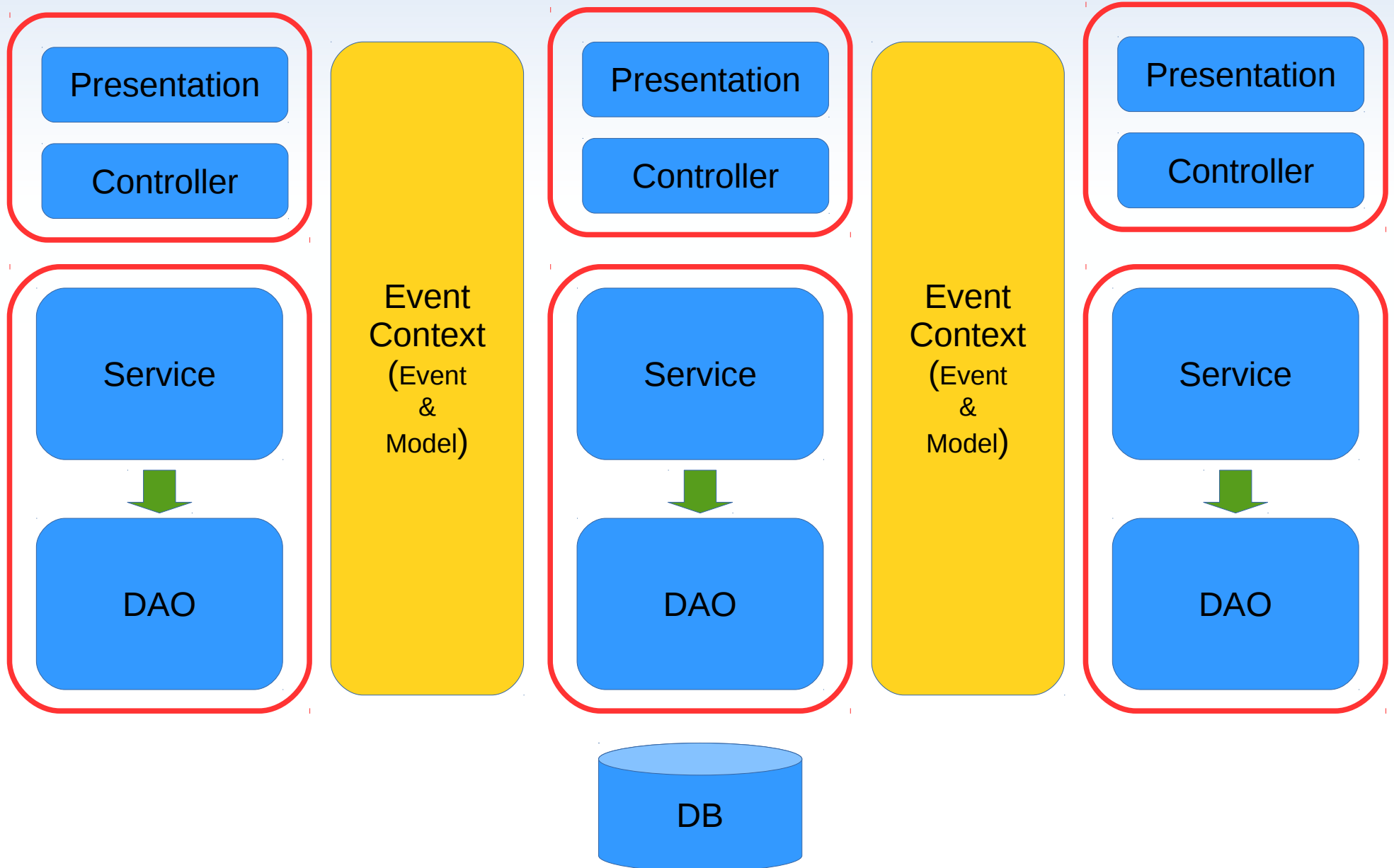
# Mediator



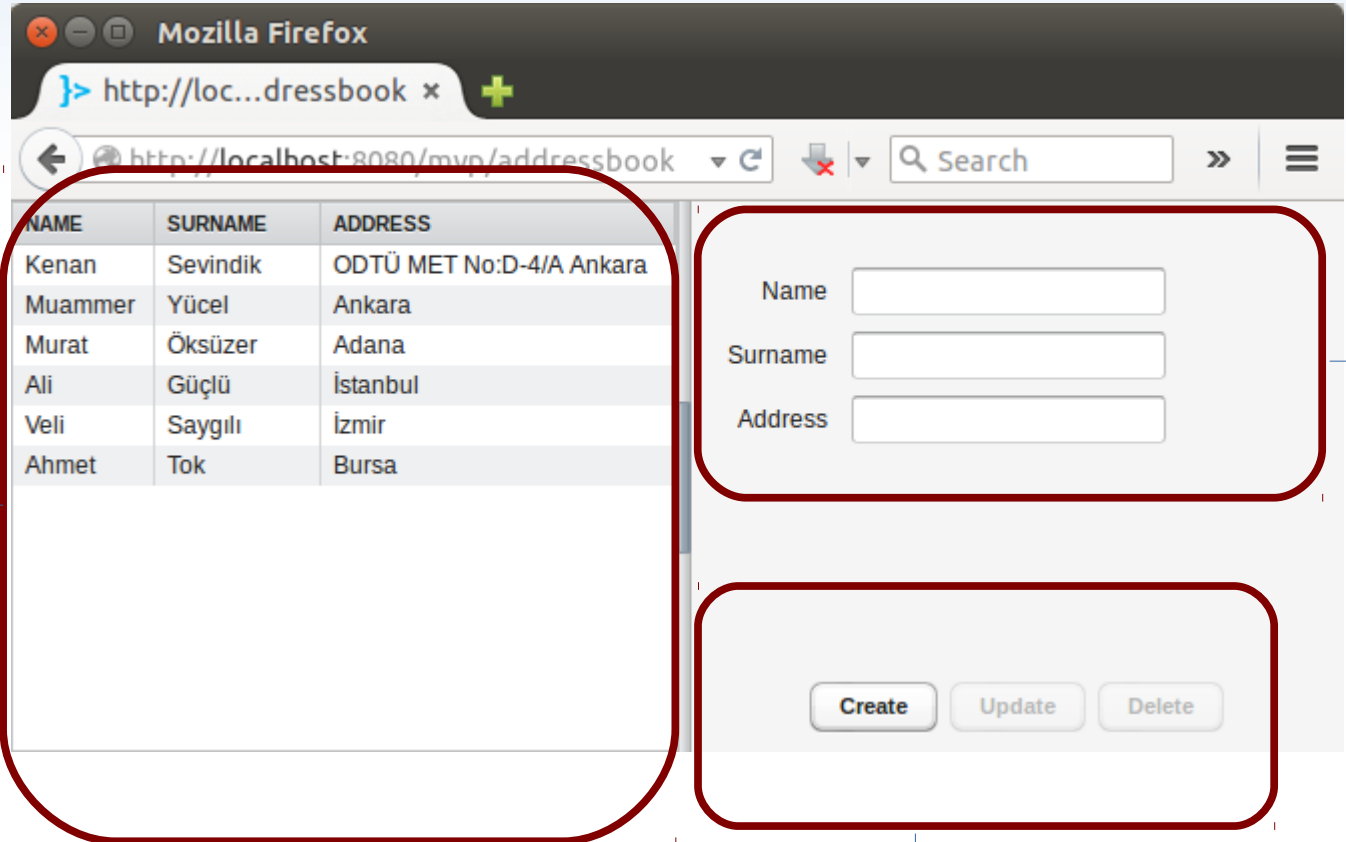
# Interactions Among Software Components After Mediator



# Interactions Among Software Components After Mediator



# Example: Address Info Management UI



The screenshot shows a Mozilla Firefox browser window displaying a web application at `http://localhost:8080/mvp/addressbook`. The application interface is annotated with three views:

- Address List View:** A table listing addresses with columns for NAME, SURNAME, and ADDRESS.
- Address Detail View:** A form with input fields for Name, Surname, and Address.
- Address ToolBar View:** A set of buttons for Create, Update, and Delete.

NAME	SURNAME	ADDRESS
Kenan	Sevindik	ODTÜ MET No:D-4/A Ankara
Muammer	Yücel	Ankara
Murat	Öksüzer	Adana
Ali	Güçlü	İstanbul
Veli	Saygılı	İzmir
Ahmet	Tok	Bursa

Form fields:

Name

Surname

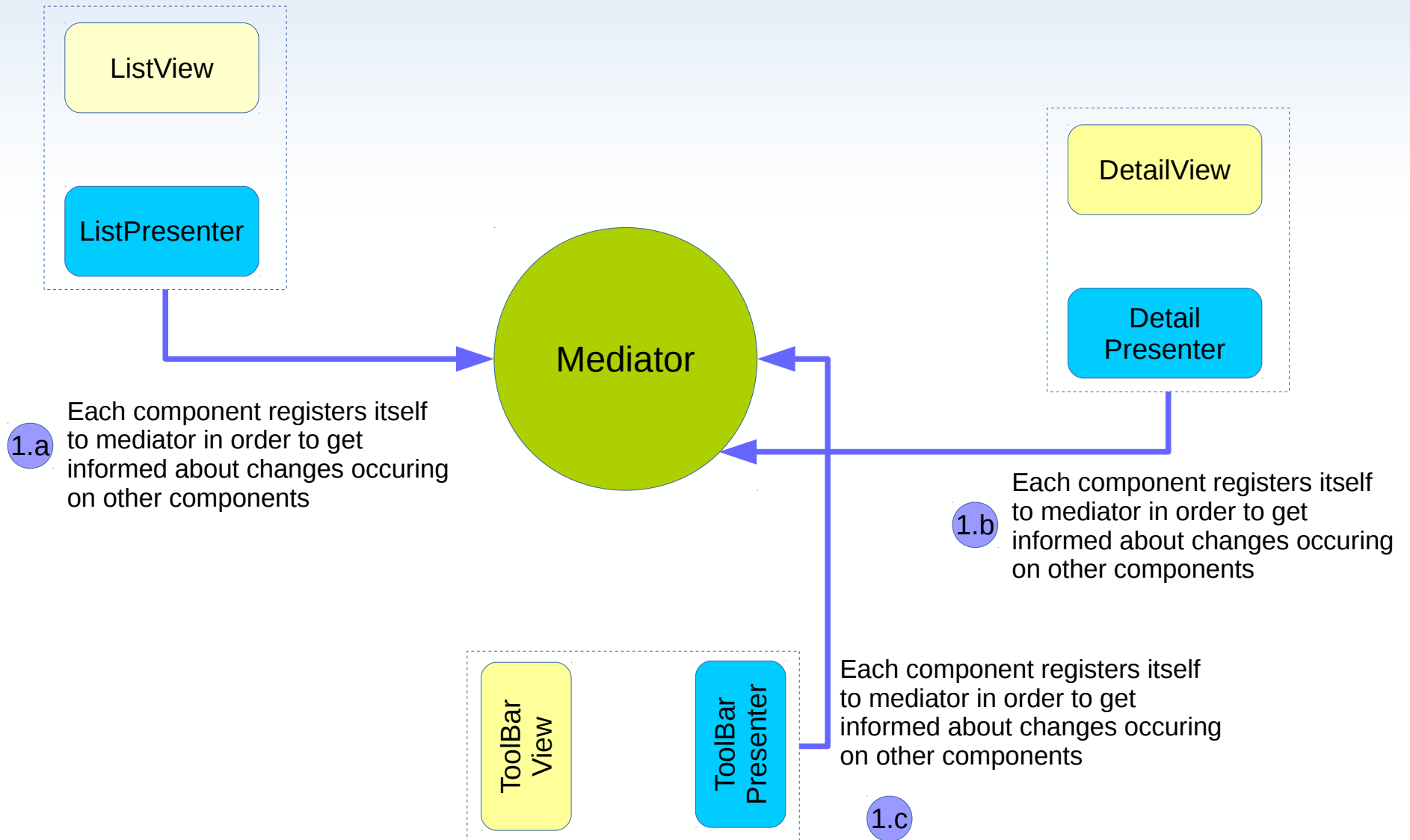
Address

Buttons: Create Update Delete

# Mediator

```
public class Mediator {  
  
    private Collection<Presenter> listeners = new  
        ArrayList<Presenter>();  
  
    public void addListener(Presenter listener) {  
        listeners.add(listener);  
    }  
  
    public void removeListener(Presenter listener) {  
        listeners.remove(listener);  
    }  
  
    public void publish(BusinessEvent event) {  
        for(Presenter listener:listeners) {  
            listener.handle(event);  
        }  
    }  
}
```

# Step 1: Mediator Registration





# Address List Presenter

```
public class AddressListPresenter implements Presenter {  
  
    private AddressListView view;  
  
    public AddressListPresenter(AddressListView view,  
        Mediator mediator) {  
  
        this.view = view;  
        mediator.addListener(this);  
    }  
  
    @Override  
    public void handle(BusinessEvent event) {  
        ...  
    }  
}
```

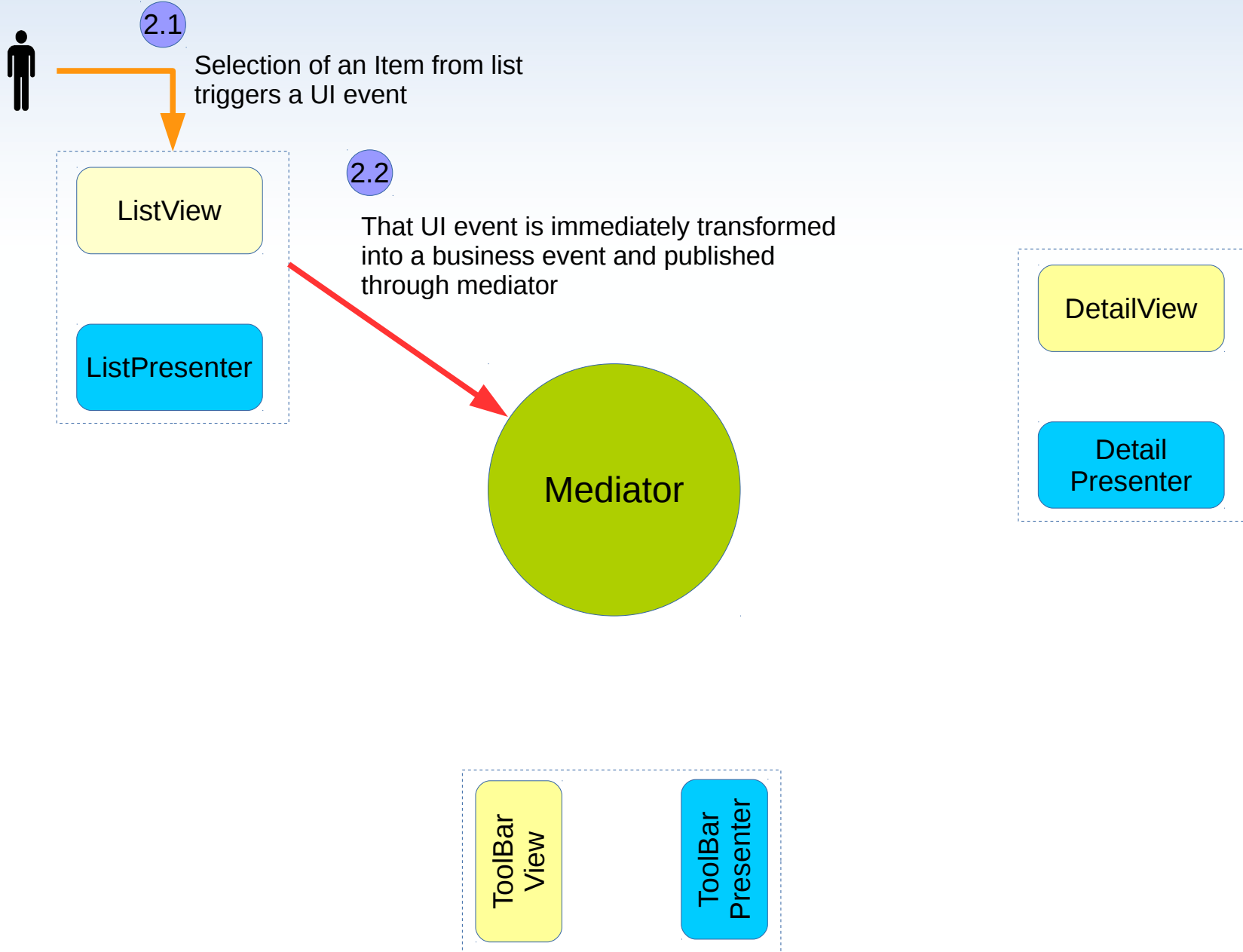
# Address Detail Presenter

```
public class AddressDetailPresenter implements Presenter {  
  
    private AddressDetailView view;  
  
    public AddressDetailPresenter(AddressDetailView view,  
        Mediator mediator) {  
  
        this.view = view;  
        mediator.addListener(this);  
    }  
  
    @Override  
    public void handle(BusinessEvent event) {  
        ...  
    }  
}
```

# Address ToolBar Presenter

```
public class AddressToolBarPresenter implements Presenter {  
  
    private AddressToolBarView view;  
  
    public AddressToolBarPresenter(AddressToolBarView view,  
        Mediator mediator) {  
  
        this.view = view;  
        mediator.addListener(this);  
    }  
  
    @Override  
    public void handle(BusinessEvent event) {  
        ...  
    }  
}
```

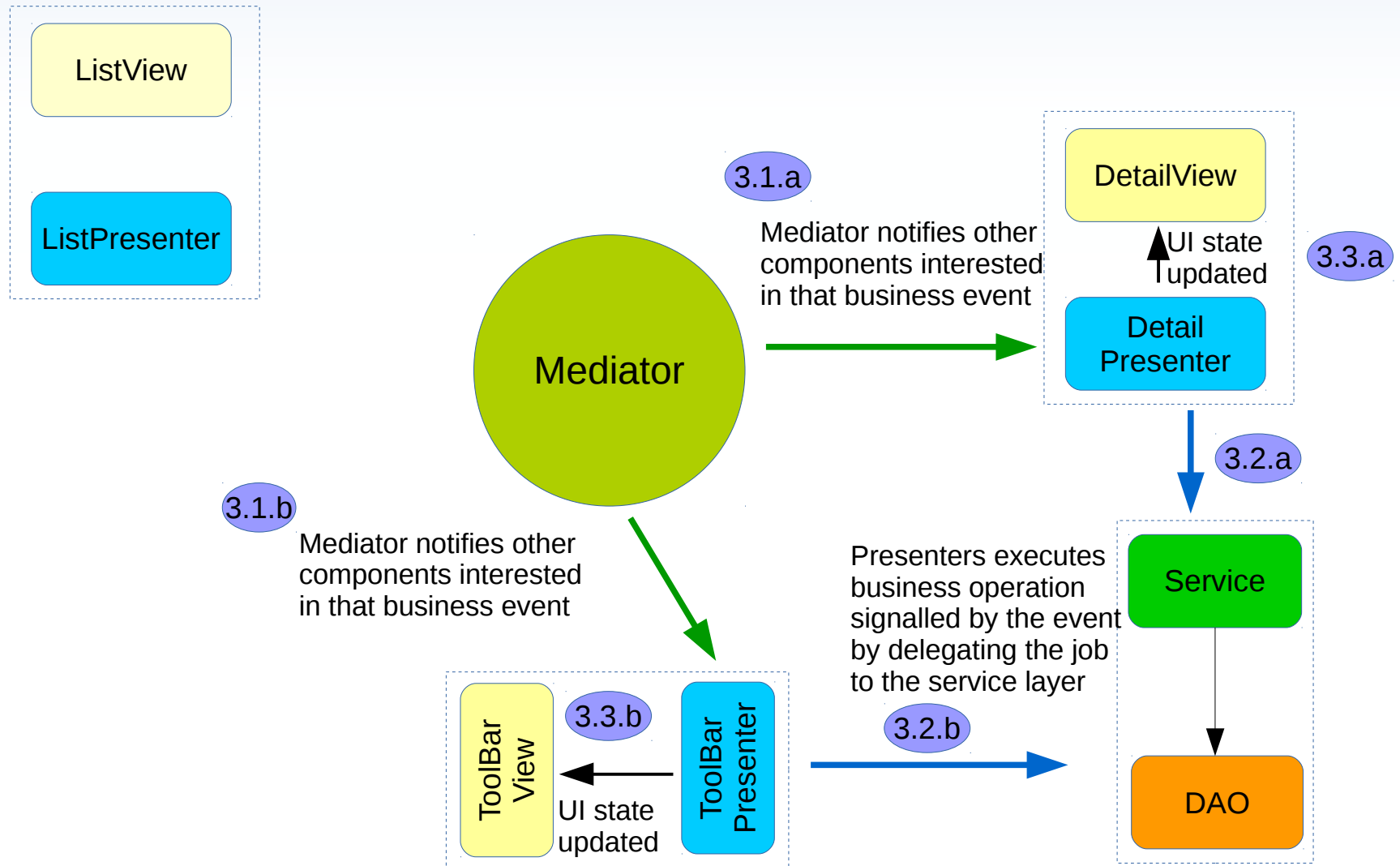
# Step 2: UI Interaction (Item Select)



# Address List View

```
public class AddressListView {  
  
    public AddressListView(Mediator mediator) {  
        this.mediator = mediator;  
    }  
  
    @Override  
    public void valueChange(ValueChangeEvent event) {  
        Address address = (Address) table.getValue();  
  
        AddressSelectedEvent selectedEvent = new  
            AddressSelectedEvent(address);  
        mediator.publish(selectedEvent);  
    }  
  
    public void loadAddresses(Collection<Address> addresses) {  
        ...  
    }  
}
```

# Step 3: Event Notification (Address Selected)





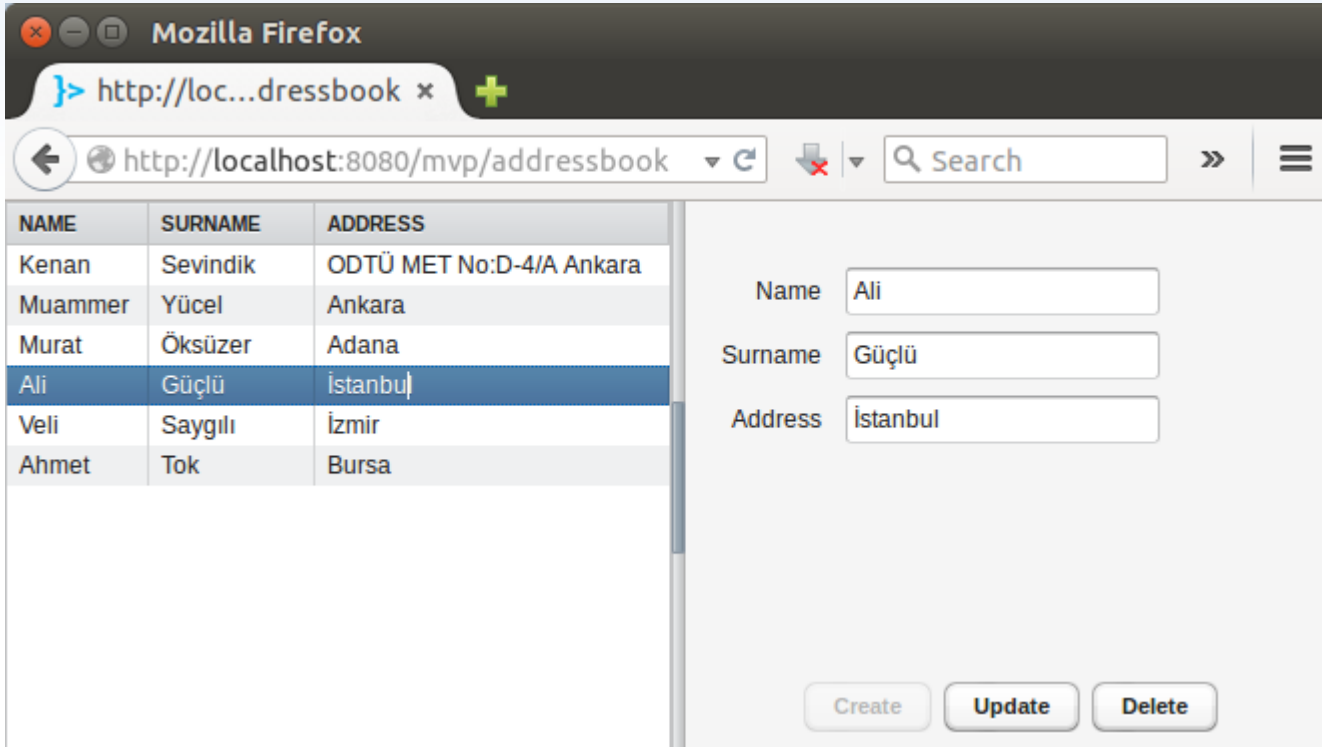
# Address Detail Presenter

```
public class AddressDetailPresenter implements Presenter {  
  
    ...  
  
    @Override  
    public void handle(BusinessEvent event) {  
        if(event instanceof AddressSelectedEvent) {  
            AddressSelectedEvent selectedEvent =  
                (AddressSelectedEvent)event;  
            Address address =  
                selectedEvent.getSelectedAddress();  
            view.displayAddress(address);  
        }  
    }  
}
```

# Address ToolBar Presenter

```
public class AddressToolBarPresenter implements Presenter {  
  
    ...  
  
    @Override  
    public void handle(BusinessEvent event) {  
        if(event instanceof AddressSelectedEvent) {  
            AddressSelectedEvent selectedEvent =  
                (AddressSelectedEvent)event;  
            Address address =  
                selectedEvent.getSelectedAddress();  
  
            view.switchToUpdateMode();  
            view.setAddress(address);  
  
        } else if(event instanceof AddressUpdateEvent) {  
            view.switchToSelectionMode();  
        }  
    }  
}
```

# Address Selected



Mozilla Firefox

http://loc...dressbook x

http://localhost:8080/mvp/addressbook

NAME	SURNAME	ADDRESS
Kenan	Sevindik	ODTÜ MET No:D-4/A Ankara
Muammer	Yücel	Ankara
Murat	Öksüzer	Adana
Ali	Güçlü	İstanbul
Veli	Saygılı	İzmir
Ahmet	Tok	Bursa

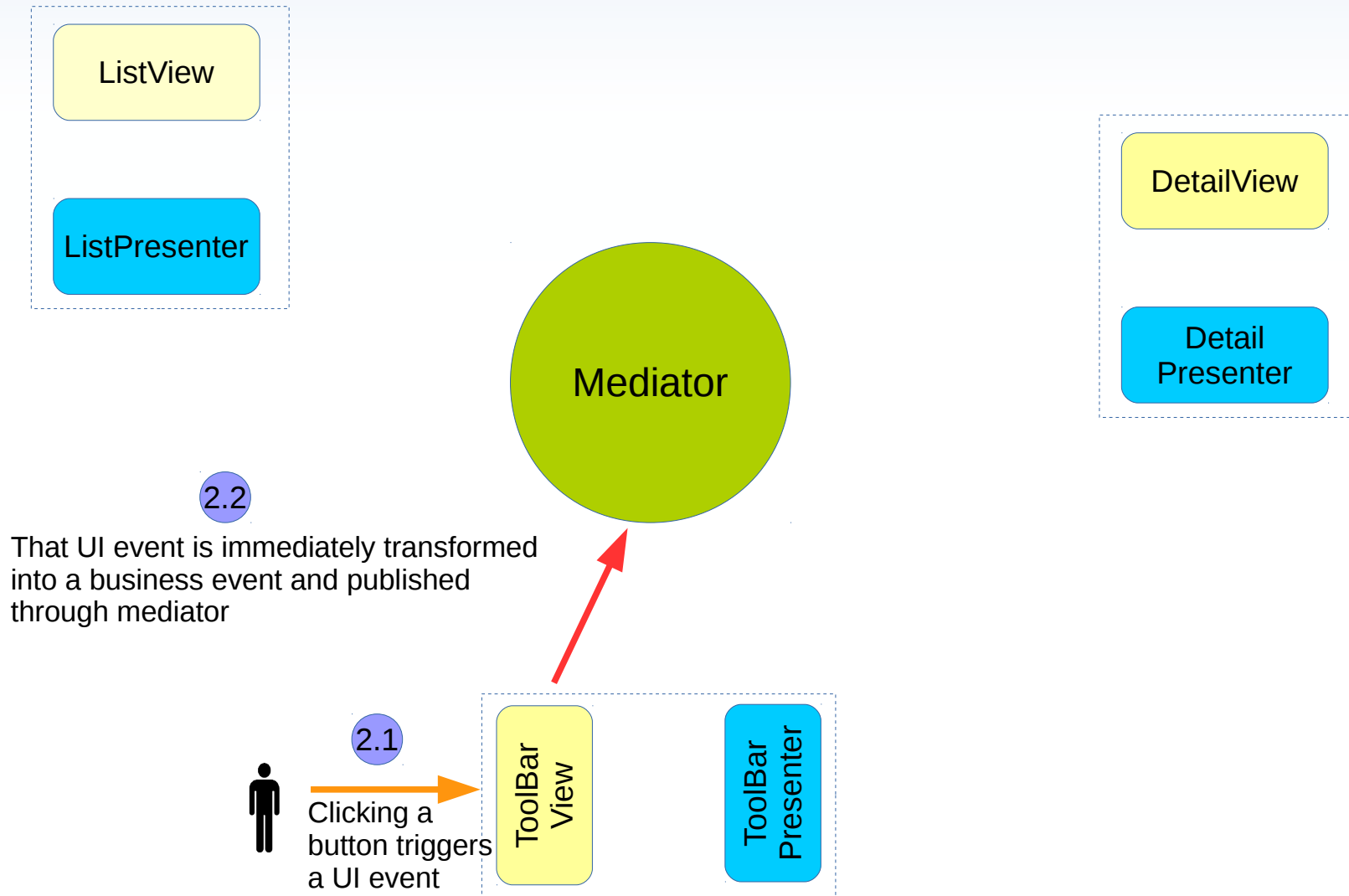
Name

Surname

Address

Create Update Delete

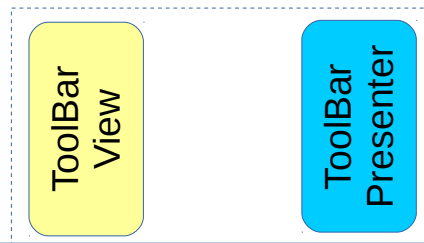
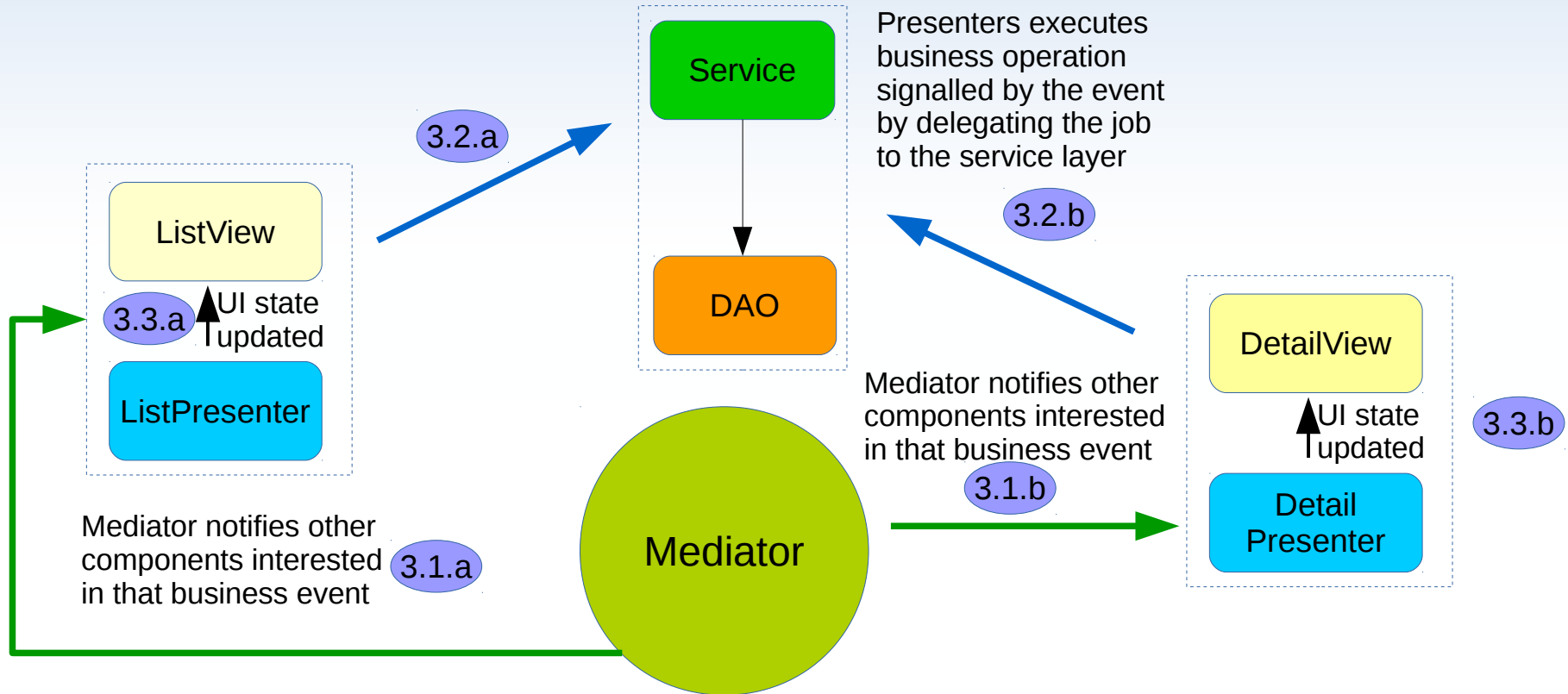
# Adım 2: UI Interaction (Button Click)



# AddressToolBar View

```
public class AddressToolBarView {  
  
    public AddressToolBarView(Mediator mediator) {  
        this.mediator = mediator;  
    }  
  
    @Override  
    public void buttonClick(ClickEvent event) {  
        if(event.getButton() == updateButton) {  
            AddressUpdateEvent updateEvent =  
                new AddressUpdateEvent(address);  
            mediator.publish(updateEvent);  
        }  
    }  
  
    public void setAddress(Address address) {  
        this.address = address;  
    }  
}
```

# Adım 3: Event Notification (Address Update)



# Address List Presenter

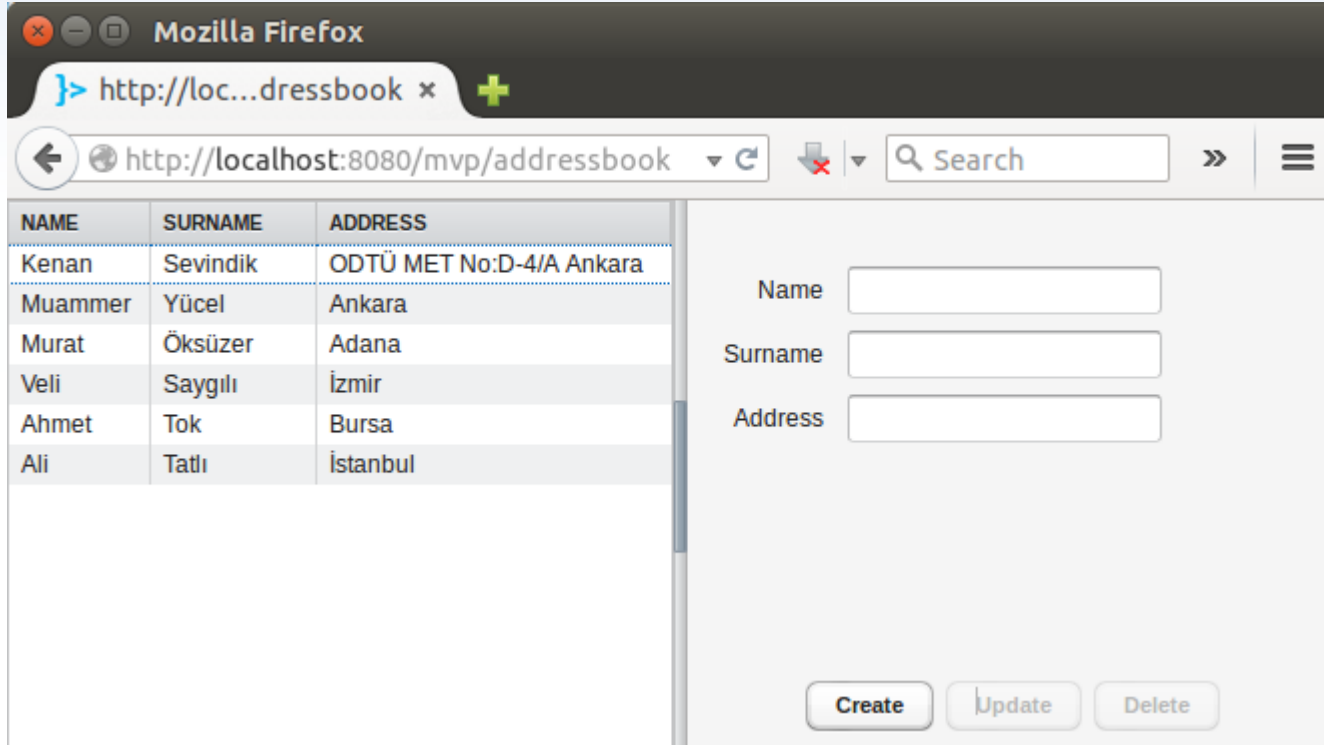
```
public class AddressListPresenter implements Presenter {  
  
    ...  
  
    @Override  
    public void handle(BusinessEvent event) {  
  
        if(event instanceof AddressUpdateEvent) {  
            AddressUpdateEvent updateEvent =  
                (AddressUpdateEvent)event;  
  
            Address address = updateEvent.getAddress();  
  
            view.reloadAddress(address);  
  
        }  
  
    }  
  
}
```

# Address ToolBar Presenter

```
public class AddressToolBarPresenter implements Presenter {  
  
    ...  
  
    @Override  
    public void handle(BusinessEvent event) {  
        if(event instanceof AddressSelectedEvent) {  
            AddressSelectedEvent selectedEvent =  
                (AddressSelectedEvent)event;  
            Address address =  
                selectedEvent.getSelectedAddress();  
  
            view.switchToUpdateMode();  
            view.setAddress(address);  
  
        } else if(event instanceof AddressUpdateEvent) {  
            view.switchToSelectionMode();  
        }  
    }  
}
```



# Address Update



Mozilla Firefox

http://loc...dressbook x

http://localhost:8080/mvp/addressbook

NAME	SURNAME	ADDRESS
Kenan	Sevindik	ODTÜ MET No:D-4/A Ankara
Muammer	Yücel	Ankara
Murat	Öksüzer	Adana
Veli	Saygılı	İzmir
Ahmet	Tok	Bursa
Ali	Tatlı	İstanbul

Name

Surname

Address

Create Update Delete

# Q & A

# Contact

- **Harezmi** IT Solutions
- <http://www.harezmi.com.tr>
- [info@harezmi.com.tr](mailto:info@harezmi.com.tr)