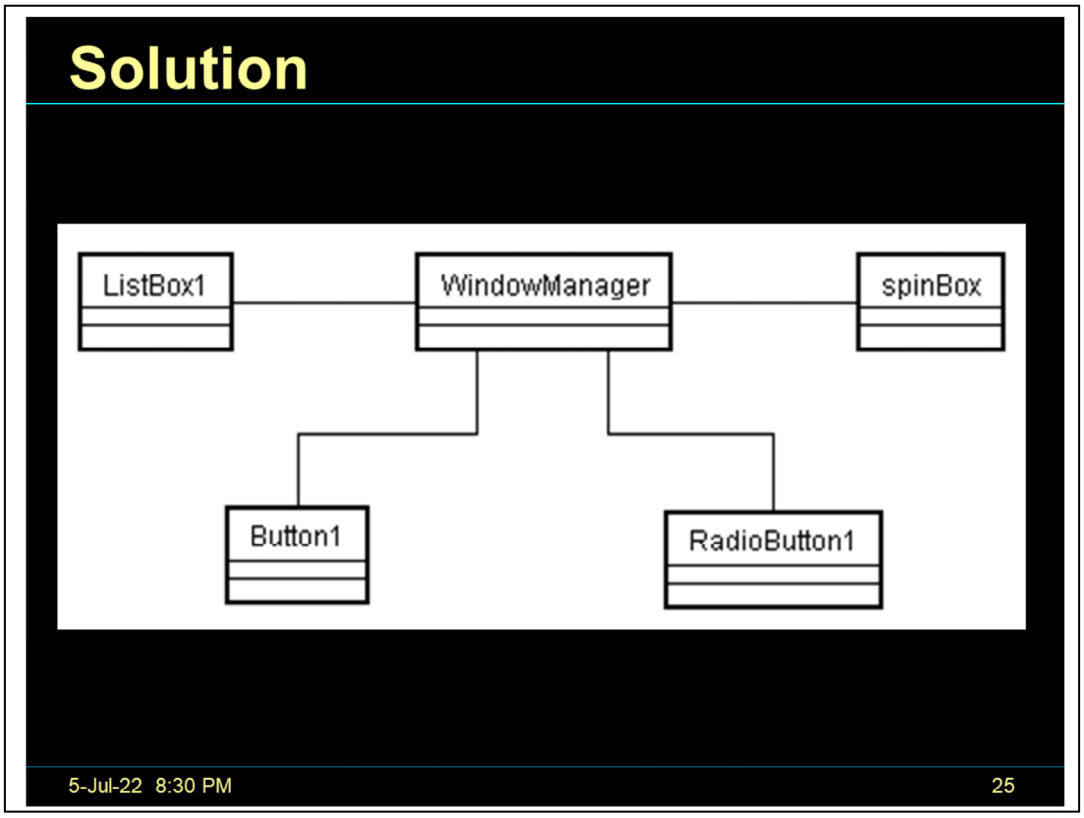
Mediator DP

Problem

- We have a set of graphic components in a window: list box, button, radio buttons, spin box, etc.
  - A change in the value of one graphic components causes others to change.
  - How do we implement the design?



Do not ask every graphic object to communicate with others.

Each graphic object communicates with a window manager. The window manager communicates with other GUI components.

It replaces many-to-many interactions with multiple one-to-many interactions.

One-to-many interactions are easier to understand, maintain and extend.

It centralizes control that otherwise would be distributed in many objects.

It is commonly used to coordinate related GUI components.

==========

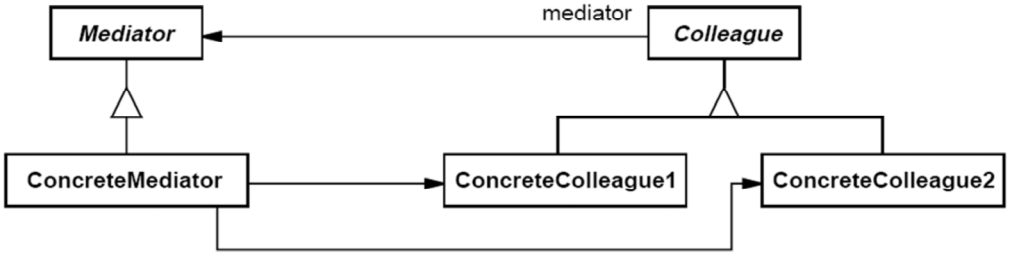
Defines an object that encapsulates how a set of objects interact.

A mediator is responsible for controlling and coordinating the interactions of a group of objects.

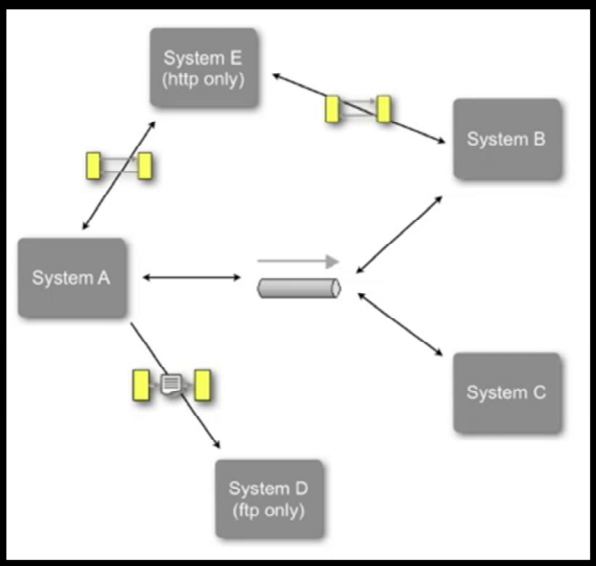
Mediator promotes loose coupling by preventing objects from referring to each other explicitly.

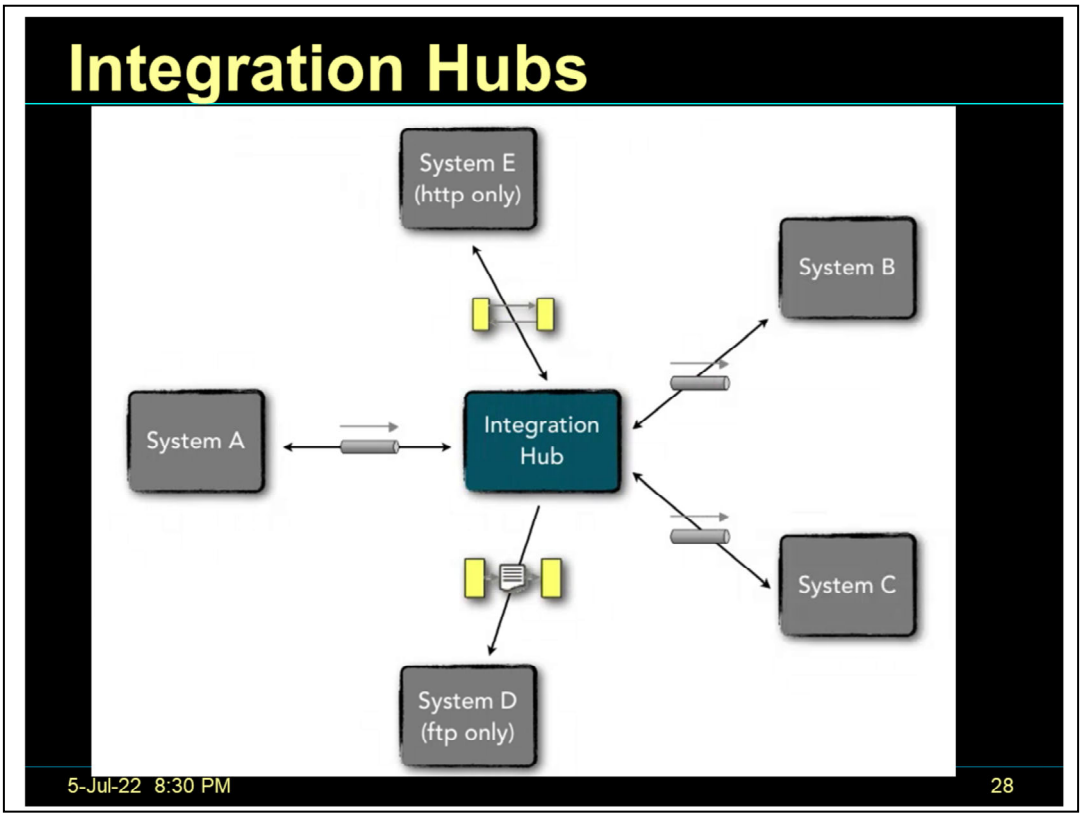
It lets us vary the interaction between objects independently.

Mediator Structure



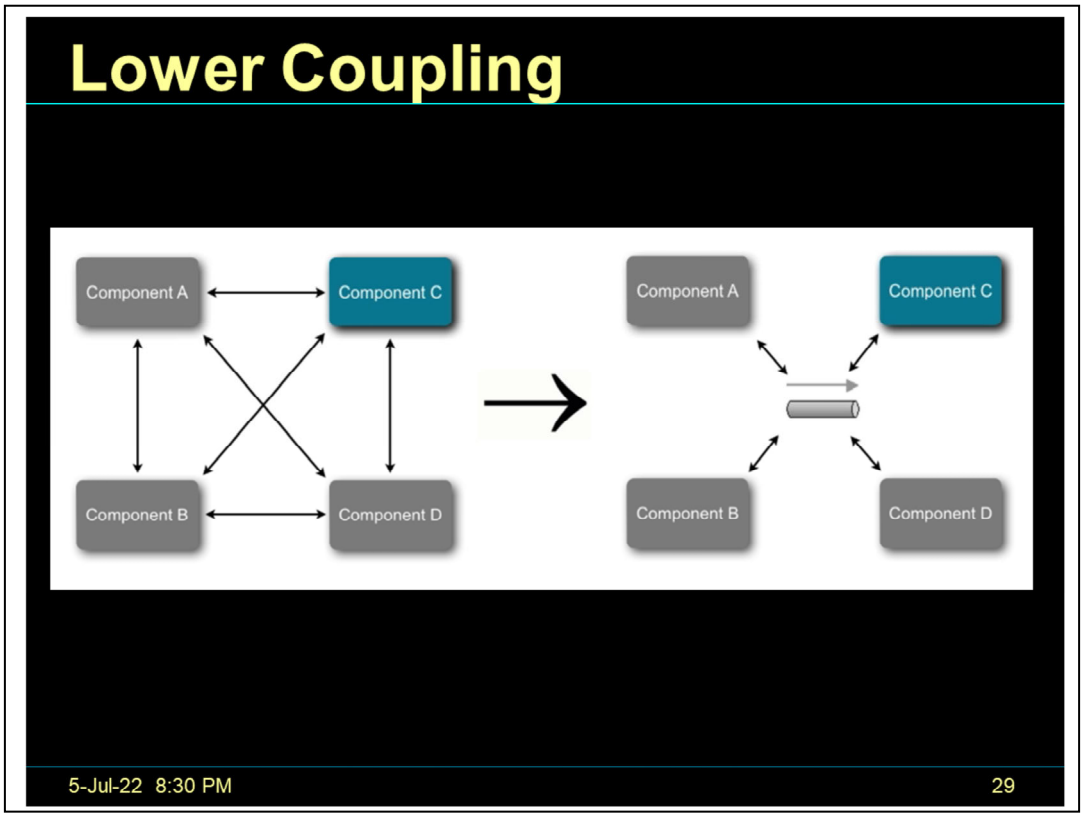
A complex, highly dependent spaghetti topology



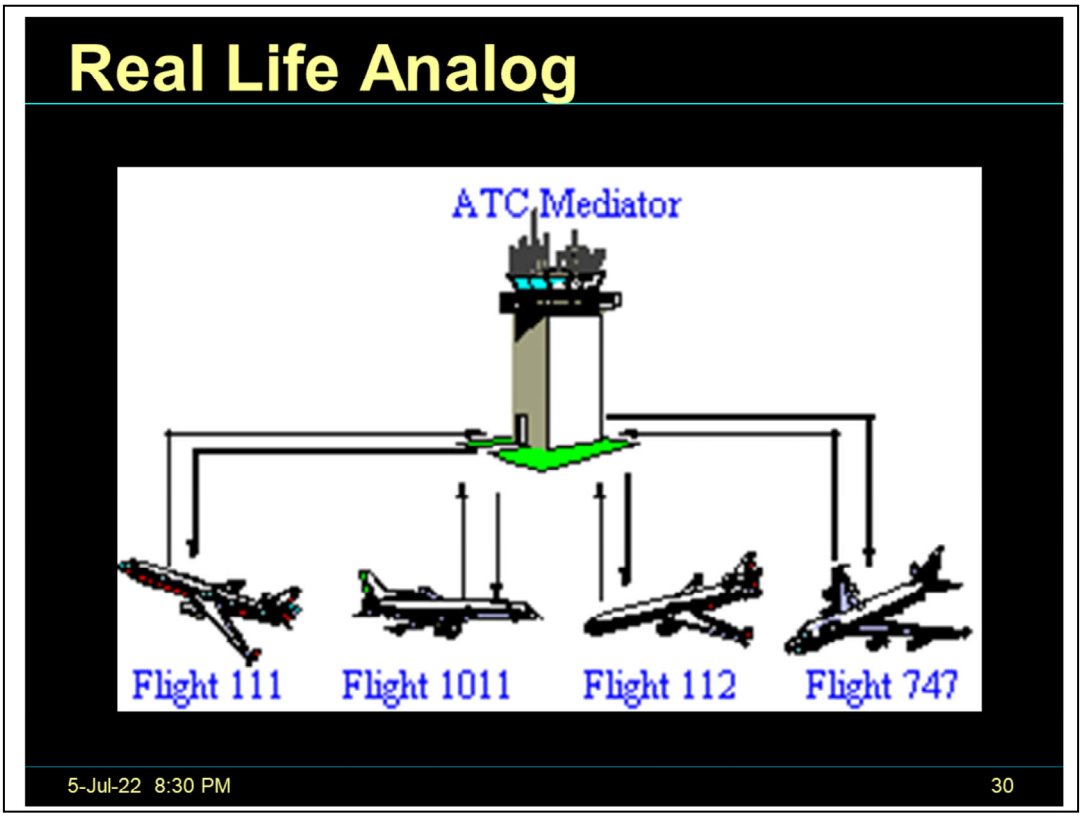


Open source Integration Hubs: Mule, Camel

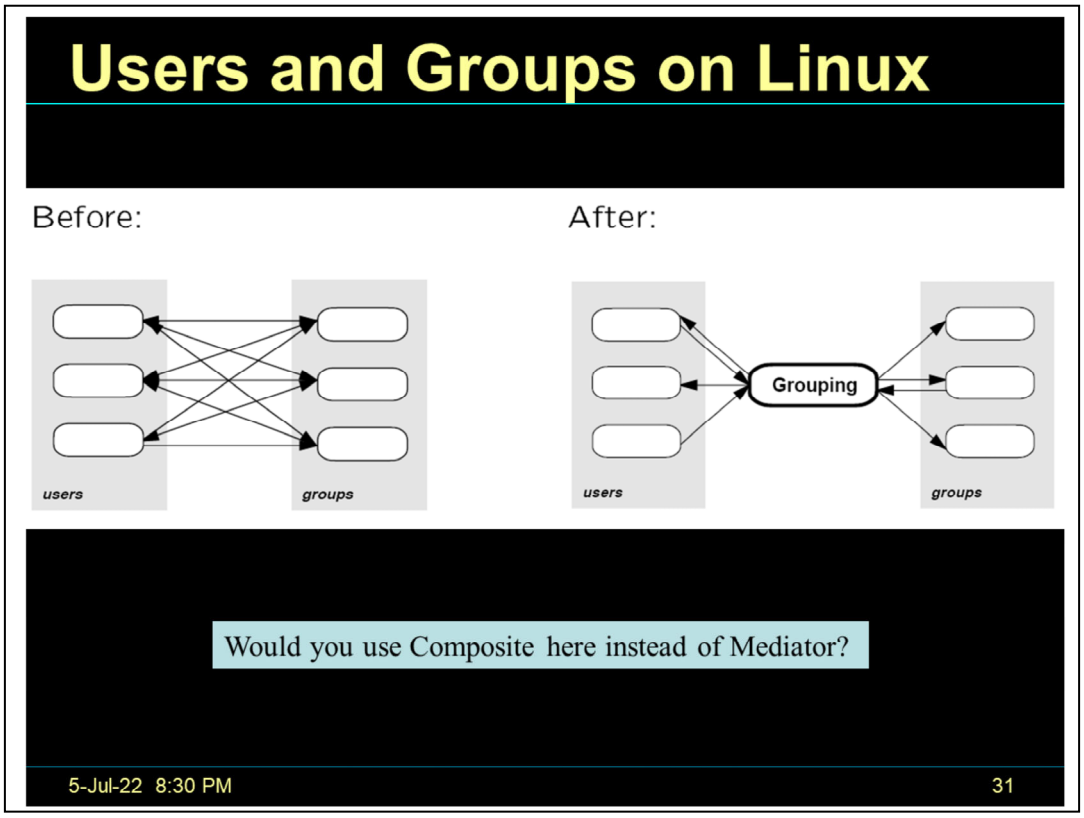
Proprietary Integration Hubs: Websphere ESB, etc.



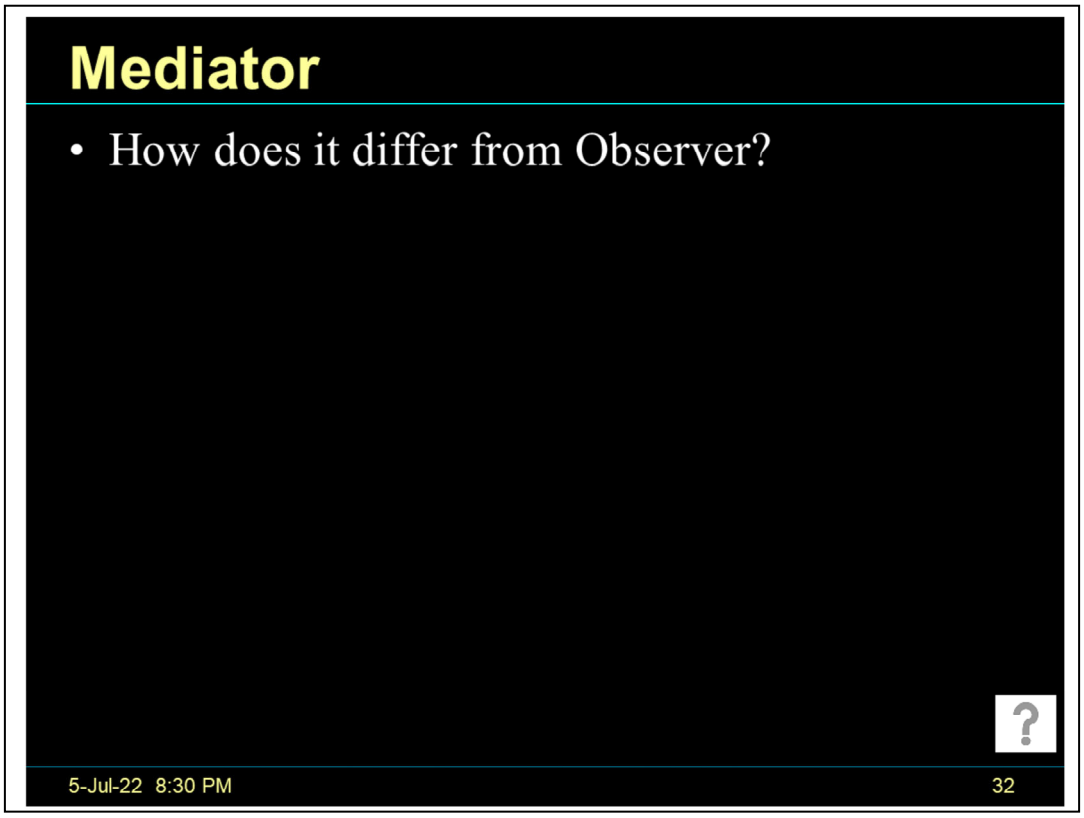
Component can be a class, package, Jar file or some application.



The *Mediator* defines an object that controls how a set of objects interact. Loose coupling between colleague objects is achieved by having colleagues communicate with the *Mediator*, rather than with each other. The control tower at a controlled airport demonstrates this pattern very well. The pilots of the planes approaching or departing the terminal area communicate with the tower, rather than explicitly communicating with one another. The constraints on who can take off or land are enforced by the tower. It is important to note that the tower does not control the whole flight. It exists only to enforce constraints in the terminal area.



User and Groups is not the same on Linux. So Composite cannot be used.
Grouping may be a Singleton



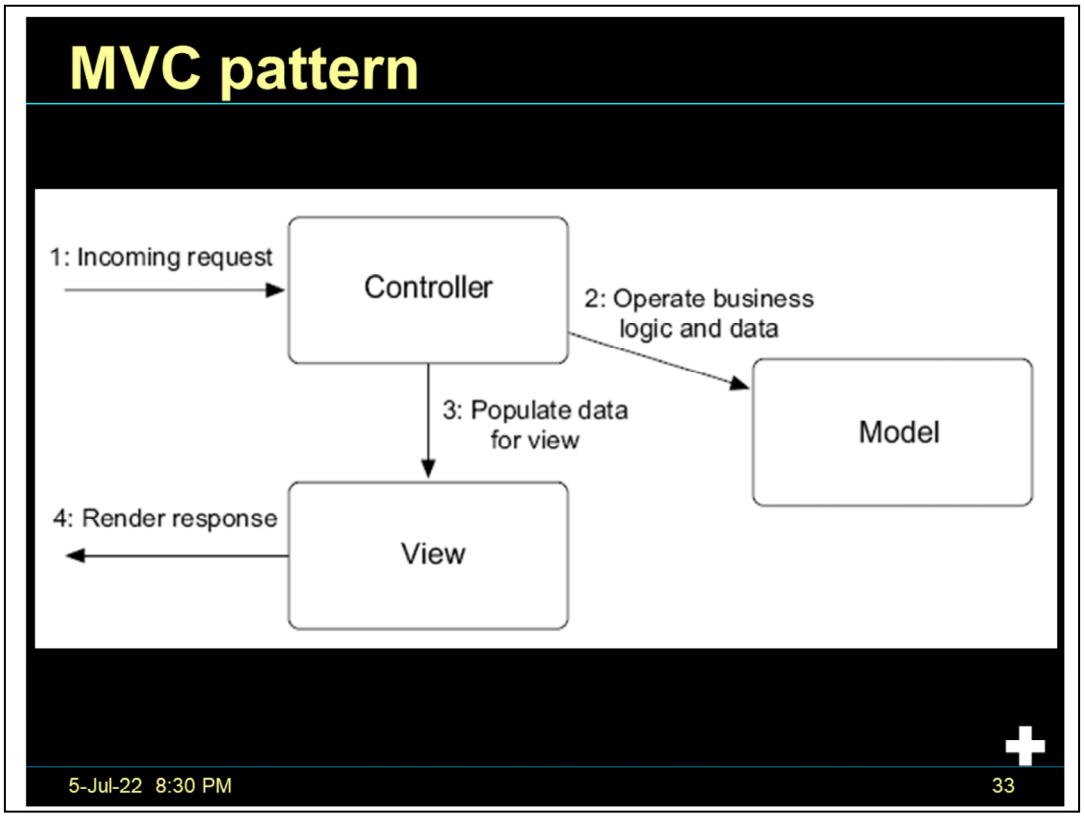
We use the mediator when

A set of objects communicate in well-defined but complex ways.

Reusing an object is difficult because it refers to and communicates with many other objects.

A behavior that's distributed between several classes should be customizable without a lot of sub-classing.

Mediator has senders and receivers reference each other indirectly. Observer defines a very decoupled interface that allows for multiple receivers to be configured at run-time.



Several problems can arise when applications contain a mixture of data access code, business domain logic code, and presentation code.

> Such applications are difficult to maintain, because interdependencies between all of the components cause strong ripple effects whenever a change is made anywhere.

The Model-View-Controller design pattern solves these problems by decoupling business domain logic and data presentation.

> MVC is an example of Mediator design pattern.

==========

Struts overview

Client browser
An HTTP request from the client browser creates an event. The Web container will respond with an HTTP response.

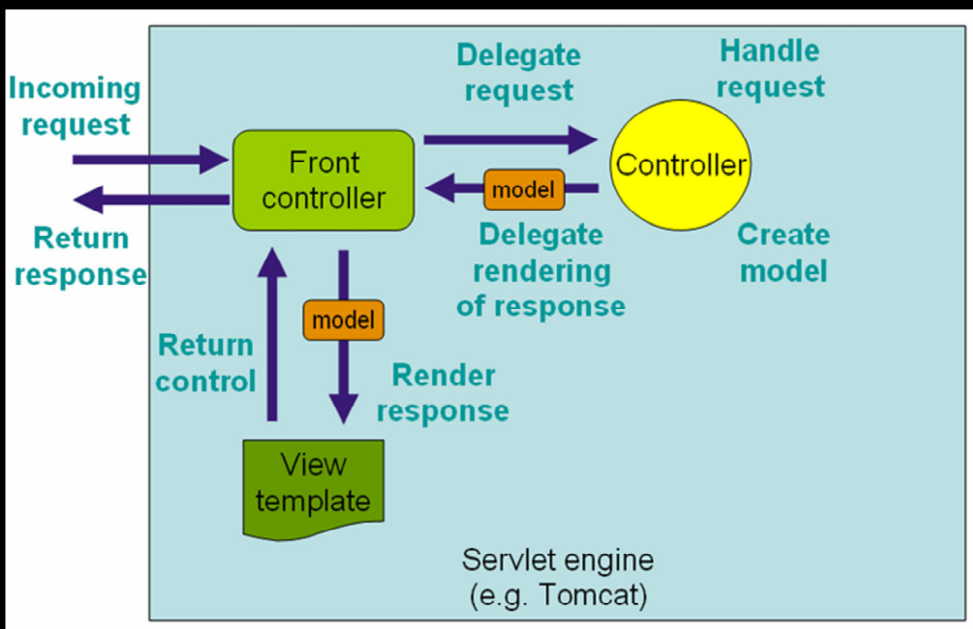
Controller
The Controller receives the request from the browser, and makes the decision where to send the request. With Struts, the Controller is a command design pattern implemented as a servlet. The struts-config.xml file configures the Controller.

Business logic
The business logic updates the state of the model and helps control the flow of the application. With Struts this is done with an Action class as a thin wrapper to the actual business logic.

Model state
The model represents the state of the application. The business objects update the application state. ActionForm bean represents the Model state at a session or request level, and not at a persistent level. The JSP file reads information from the ActionForm bean using JSP tags.

View
The view is simply a JSP file. There is no flow logic, no business logic, and no model information -- just tags. Tags are one of the things that make Struts unique compared to other frameworks like Velocity.

Spring MVC


Incoming request
Return response
Front controller
Delegate request
Handle request
Controller
model
Delegate rendering of response
Create model
model
Return control
Render response
View template
Servlet engine (e.g. Tomcat)