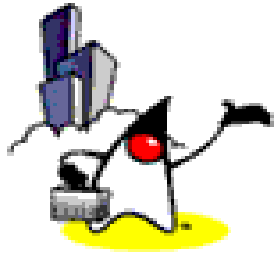
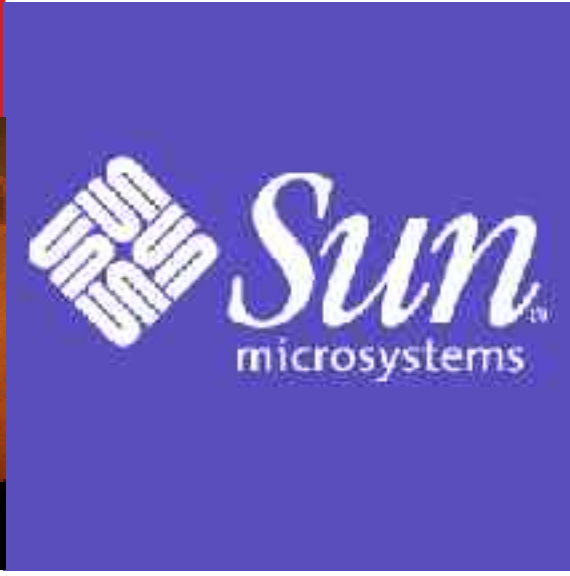
MVC Pattern (MVC Framework)





Sang Shin

sang.shin@sun.com
www.javapassion.com
Java™ Technology Evangelist
Sun Microsystems, Inc.

Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employees of Sun Microsystems, the contents here are created as their own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgements
  - Many slides are borrowed from “Sevlet/JSP” codecamp material authored by Doris Chen of Sun Microsystems
  - Some slides are from JavaOne 2003 “Blueprint for Web services” presentation authored by Inderjeet Singh and Sean Brydon

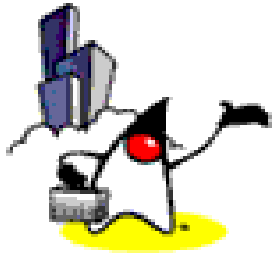
Revision History

- 11/01/2003: version 1: created by Sang Shin
- Things to do
  - speaker notes need to be polished

Agenda

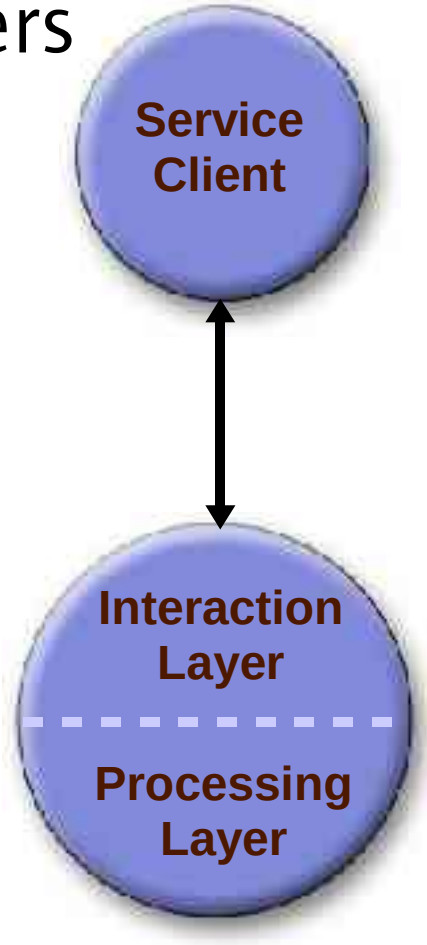
- Layered (or tiered) application design
- Introduction of MVC pattern
- Evolution of Web Application design architecture
  - Model 1
  - Model 2
  - Application frameworks

Layered Application Design



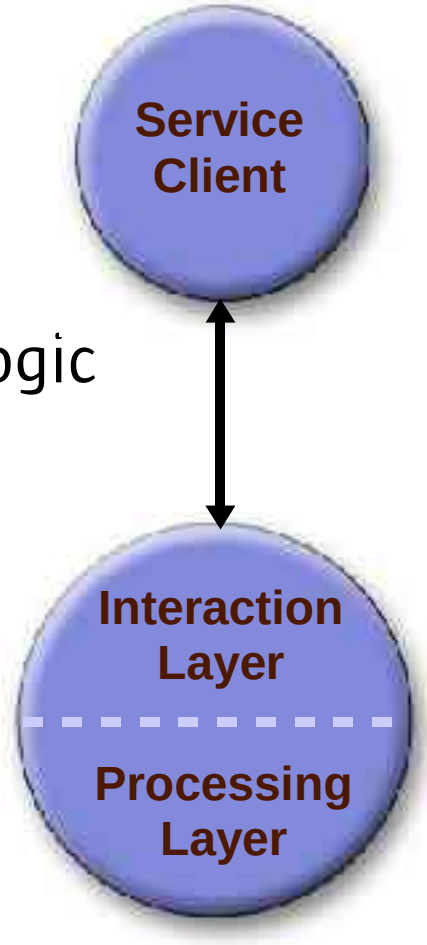
Layered Application Design

- Structure application in two layers
  - Interaction Layer and Processing Layer
- Interaction layer
  - Interface to clients
  - Receive requests and perform required translations and transformations
  - Delegate request to processing layer for processing
  - Respond to clients



Layered Application Design

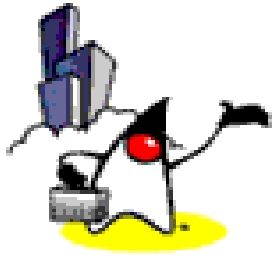
- Processing layer
  - Process request by performing business logic
  - Access database
  - Integrate with EIS



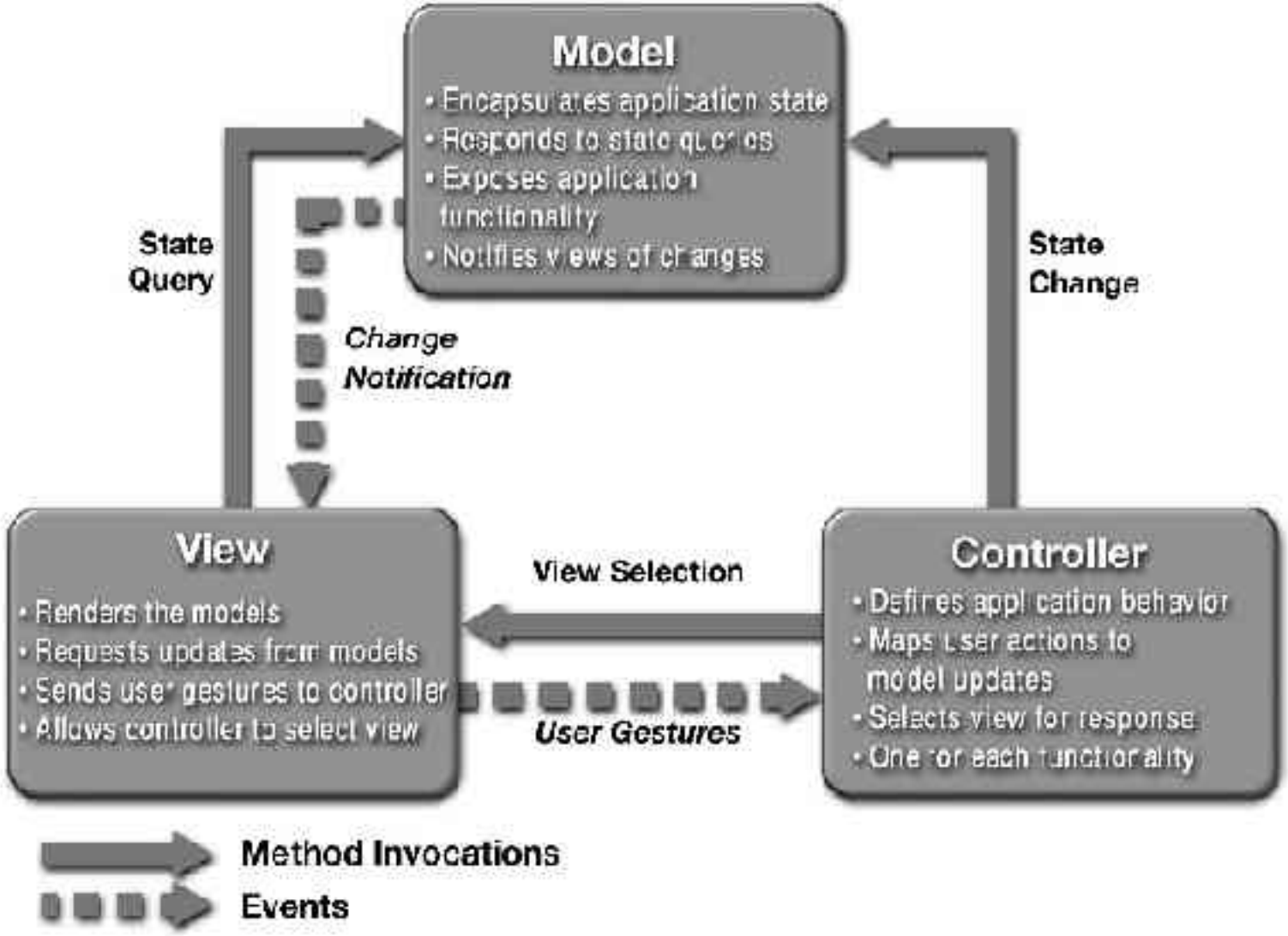
Why Layered Application Design?

- Clearly divide responsibilities
  - De-couple business logic from presentation
  - Change in business logic layer does not affect the presentation layer and vice-versa
- Provide a common “place” for pre-processing and post-processing of requests and responses
  - logging, translations, transformations, etc.

Introduction to MVC Pattern



MVC Pattern


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Three Logical Layers in a Web Application: Model

- Model (Business process layer)
  - Models the data and behavior behind the business process
  - Responsible for actually doing
    - Performing DB queries
    - Calculating the business process
    - Processing orders
  - Encapsulate of data and behavior which are independent of presentation

Three Logical Layers in a Web Application: View

- View (Presentation layer)
  - Display information according to client types
  - Display result of business logic (Model)
  - Not concerned with how the information was obtained, or from where (since that is the responsibility of Model)

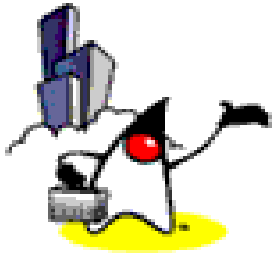
Three Logical Layers in a Web Application: Controller

- Controller (Control layer)
  - Serves as the logical connection between the user's interaction and the business services on the back
  - Responsible for making decisions among multiple presentations
    - e.g. User's language, locale or access level dictates a different presentation.
  - A request enters the application through the control layer, it will decide how the request should be handled and what information should be returned

Web Applications

- It is often advantageous to treat each layer as an independent portion of your application
- Do not confuse logical separation of responsibilities with actual separation of components
- Some or of the layers can be combined into single components to reduce application complexity

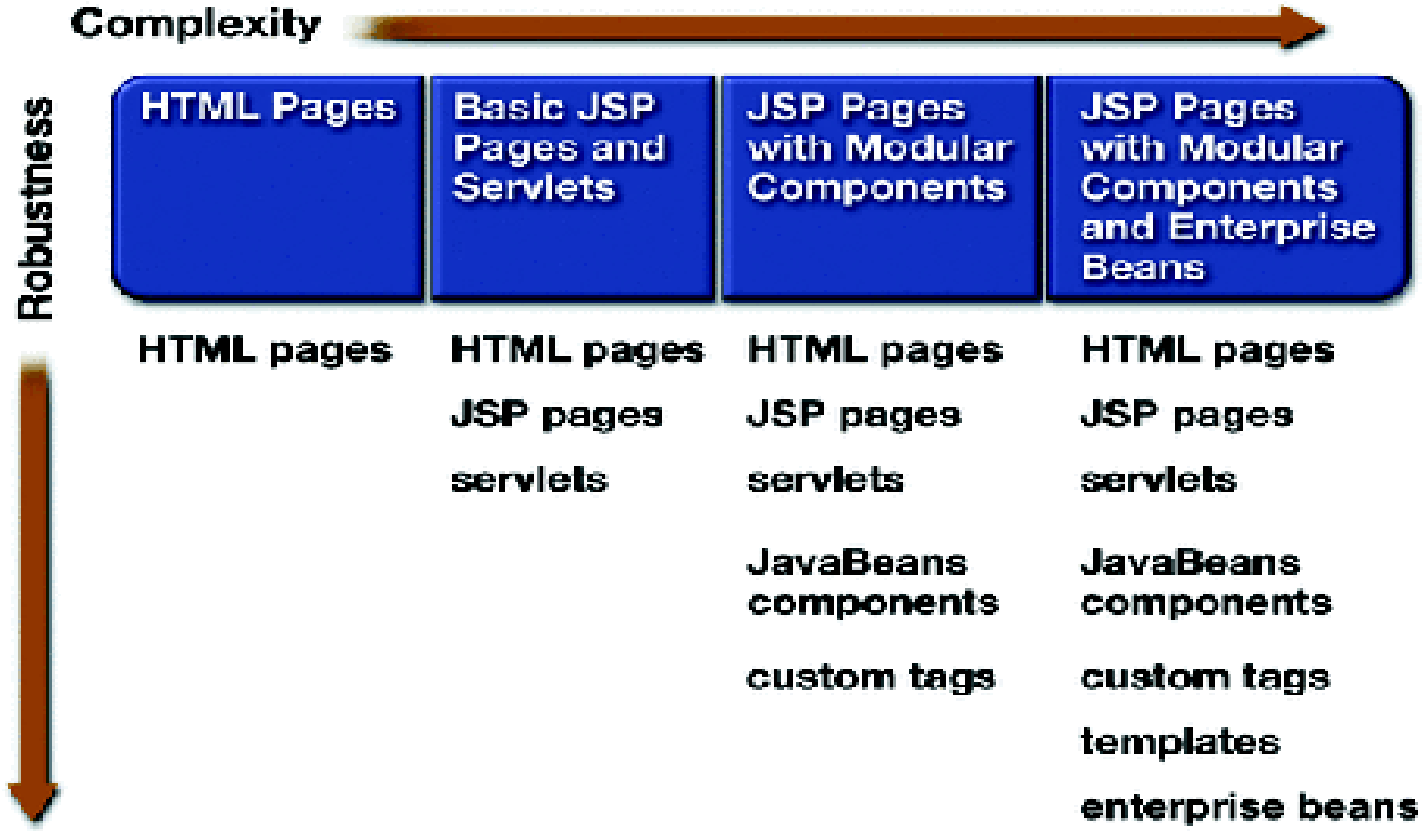
Evolution of Web Application Design Architecture

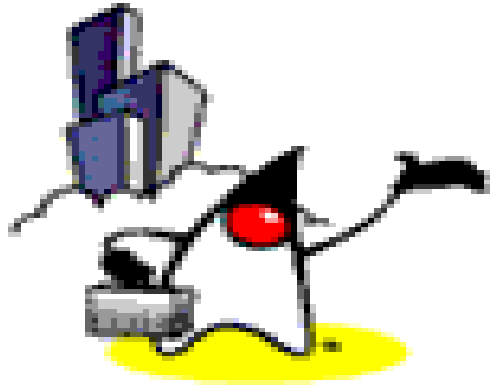


Evolution of MVC Architecture

1. No MVC
2. MVC Model 1 (Page-centric)
3. MVC Model 2 (Servlet-centric)
4. Web application frameworks
   - Struts
5. Standard-based Web application framework
   - JavaServer Faces (JSR-127)

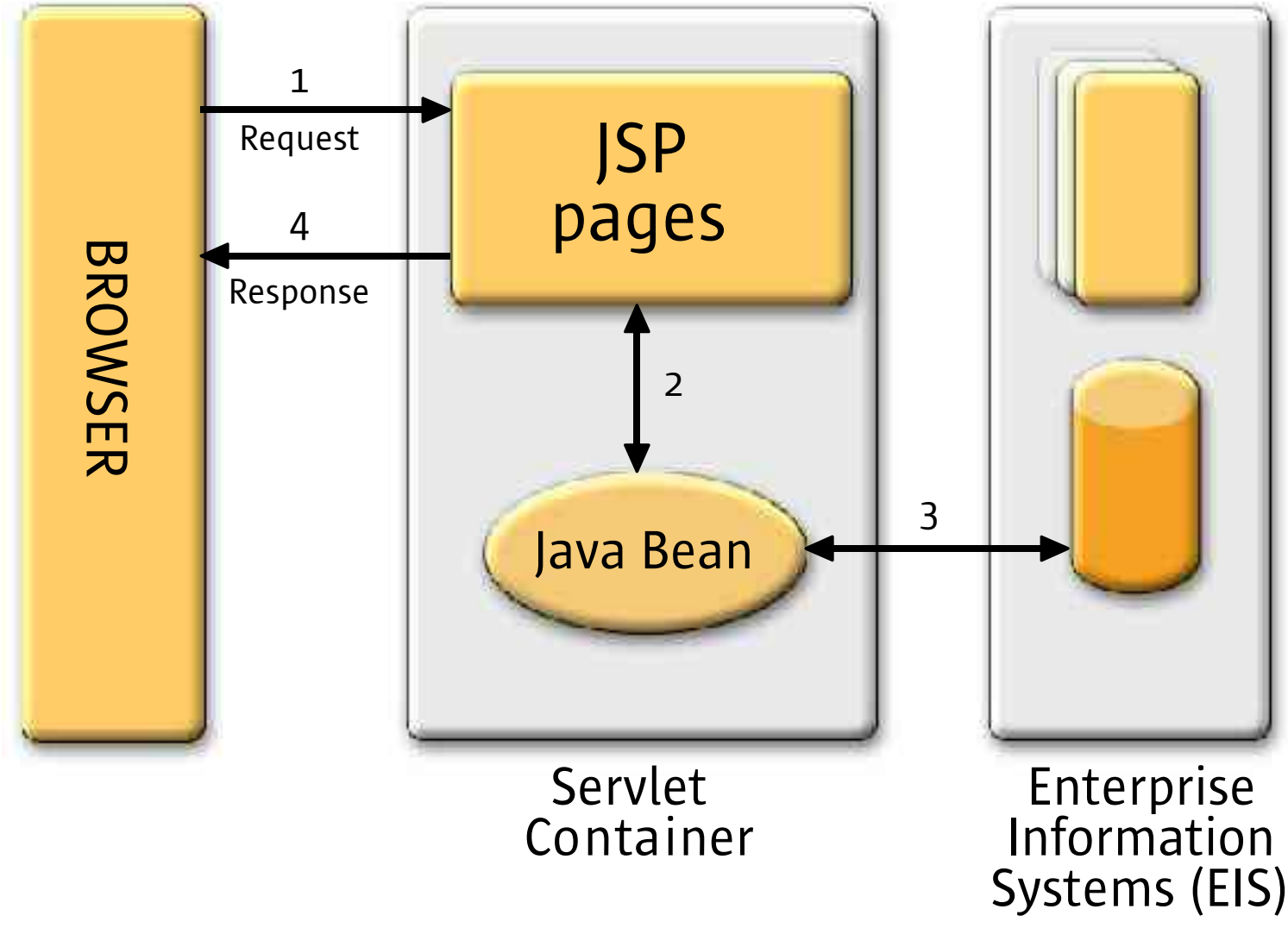
Evolution of Web Application Design until Model 1 Architecture





Model 1 (Page-Centric Architecture)

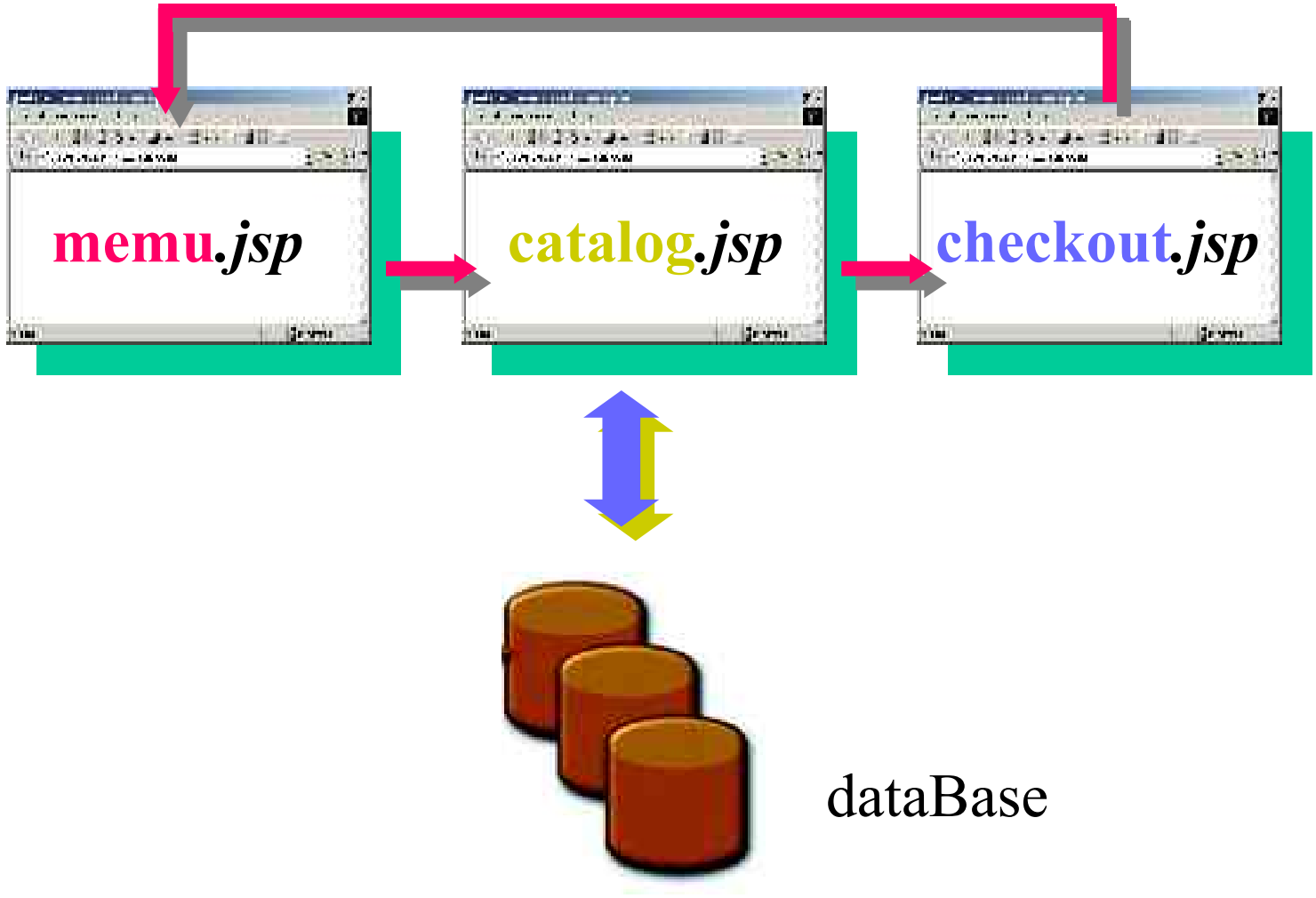
Model 1 Architecture (Page-centric)



Page-centric Architecture

- Composed of a series of interrelated JSP pages
  - JSP pages handle all aspects of the application - presentation, control, and business process
- Business process logic and control decisions are hard coded inside JSP pages
  - in the form of JavaBeans, scriptlets, expression
- Next page selection is determined by
  - A user clicking on a hyper link, e.g. <A HERF="find.jsp>
  - Through the action of submitting a form, e.g. <FORM ACTION="search.jsp">

Page-centric Architecture

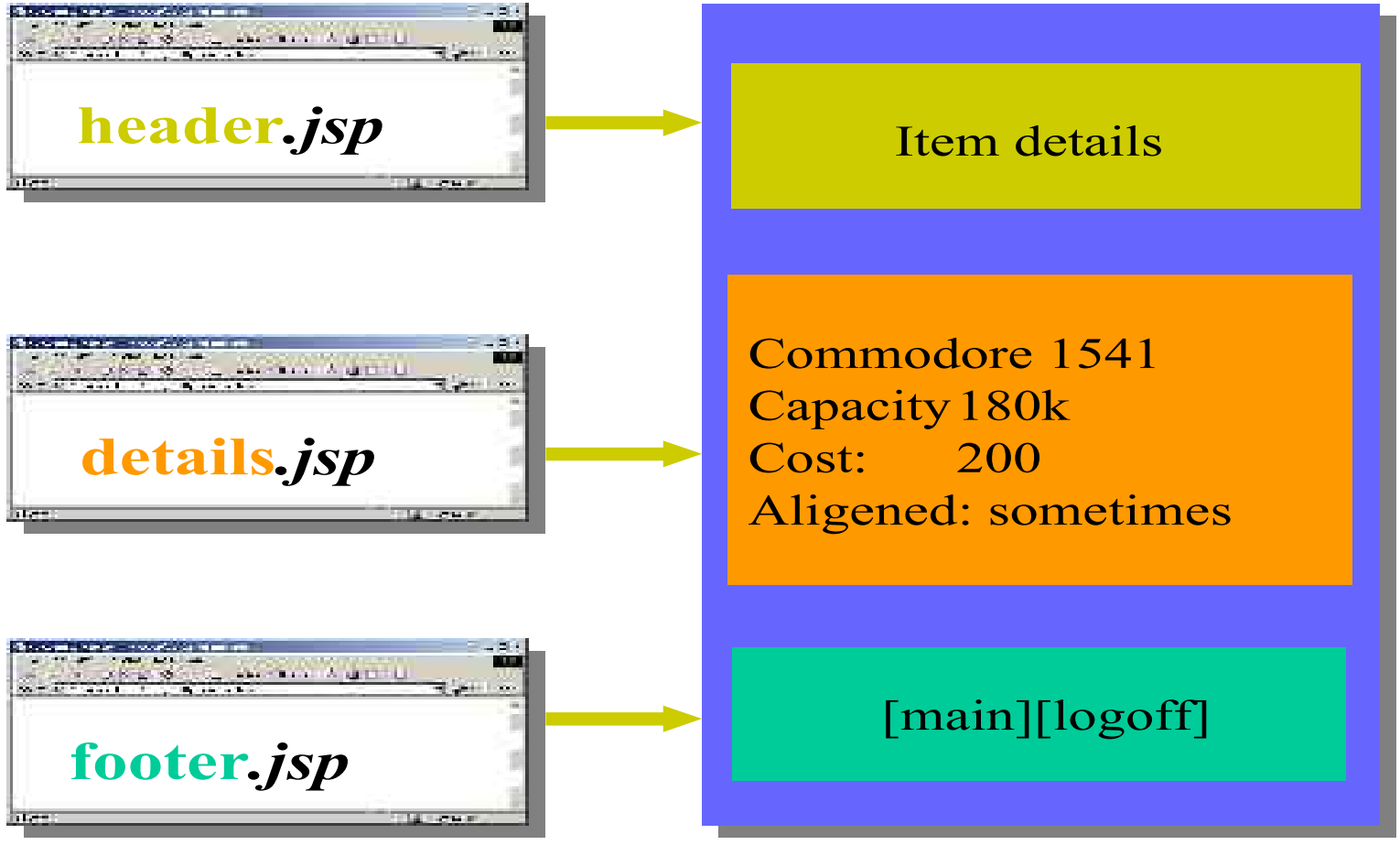


page–centric catalog application

Page-centric: Simple Application

- One page might display a menu of options, another might provide a form for selecting items from the catalog,and another would be to complete shopping process
  - This doesn't mean we lose separation of presentation and content
  - Still use the dynamic nature of JSP and its support for JavaBeans component to factor out business logic from presentation
  - The pages are tightly coupled:
    - Need to sync up request parameters
    - Be aware of each other's URLs

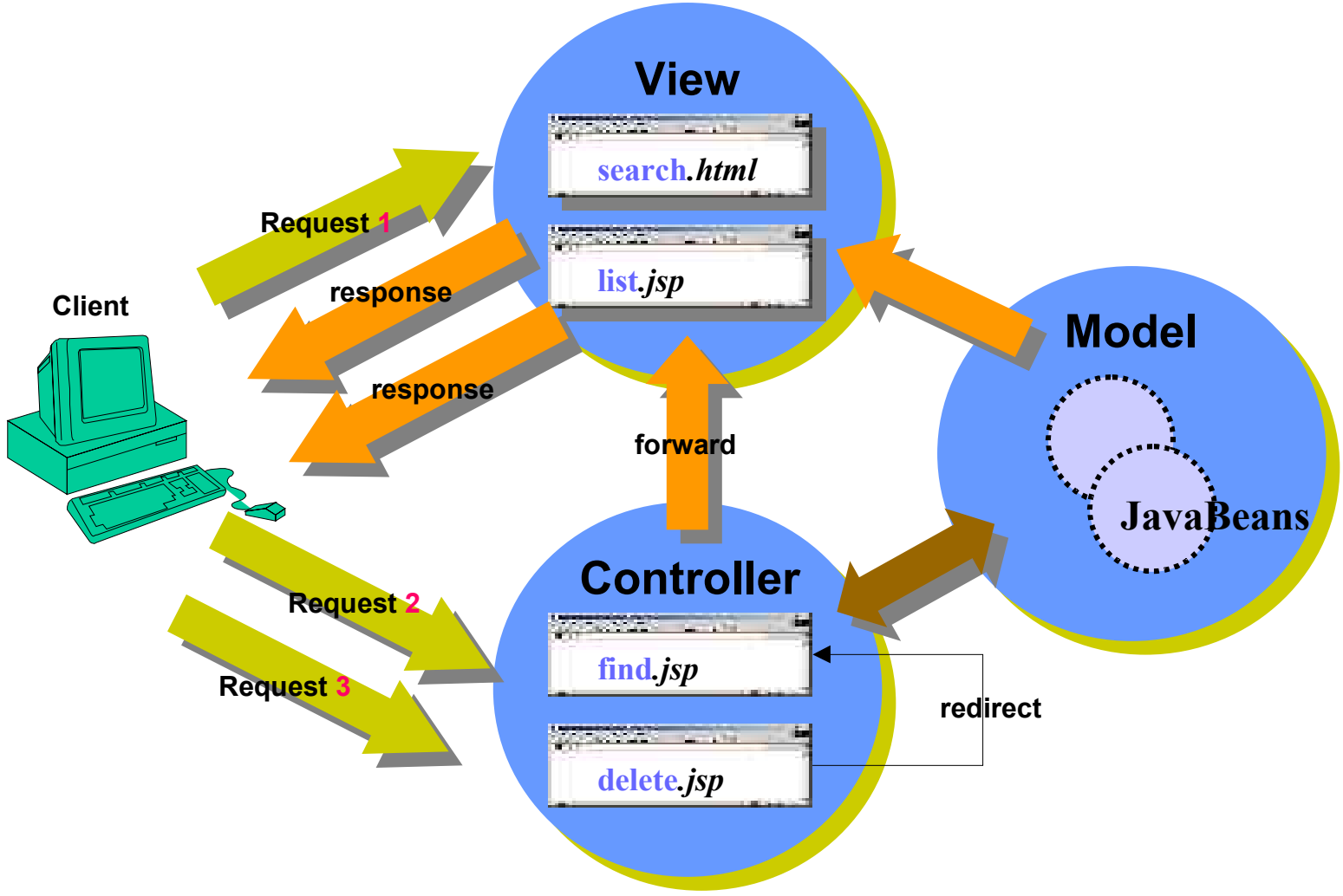
Page-centric: Component Page Diagram

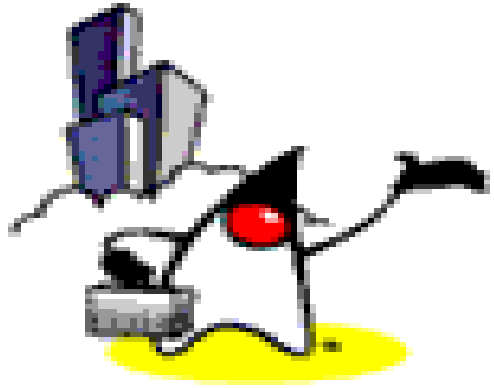


Page-centric: Component Page

- Create headers, footers and navigation bars in JSP pages
  - Provides better flexibility and reusability.
  - Easy to maintain.
- <%@ include file = "header.jsp" %>
  - Use it when the file (included) changes rarely.
  - Faster than jsp:include.
- <jsp:include page="header.jsp" flush="true">
  - Use it for content that changes often
  - if which page to include can not be decided until the main page is requested.

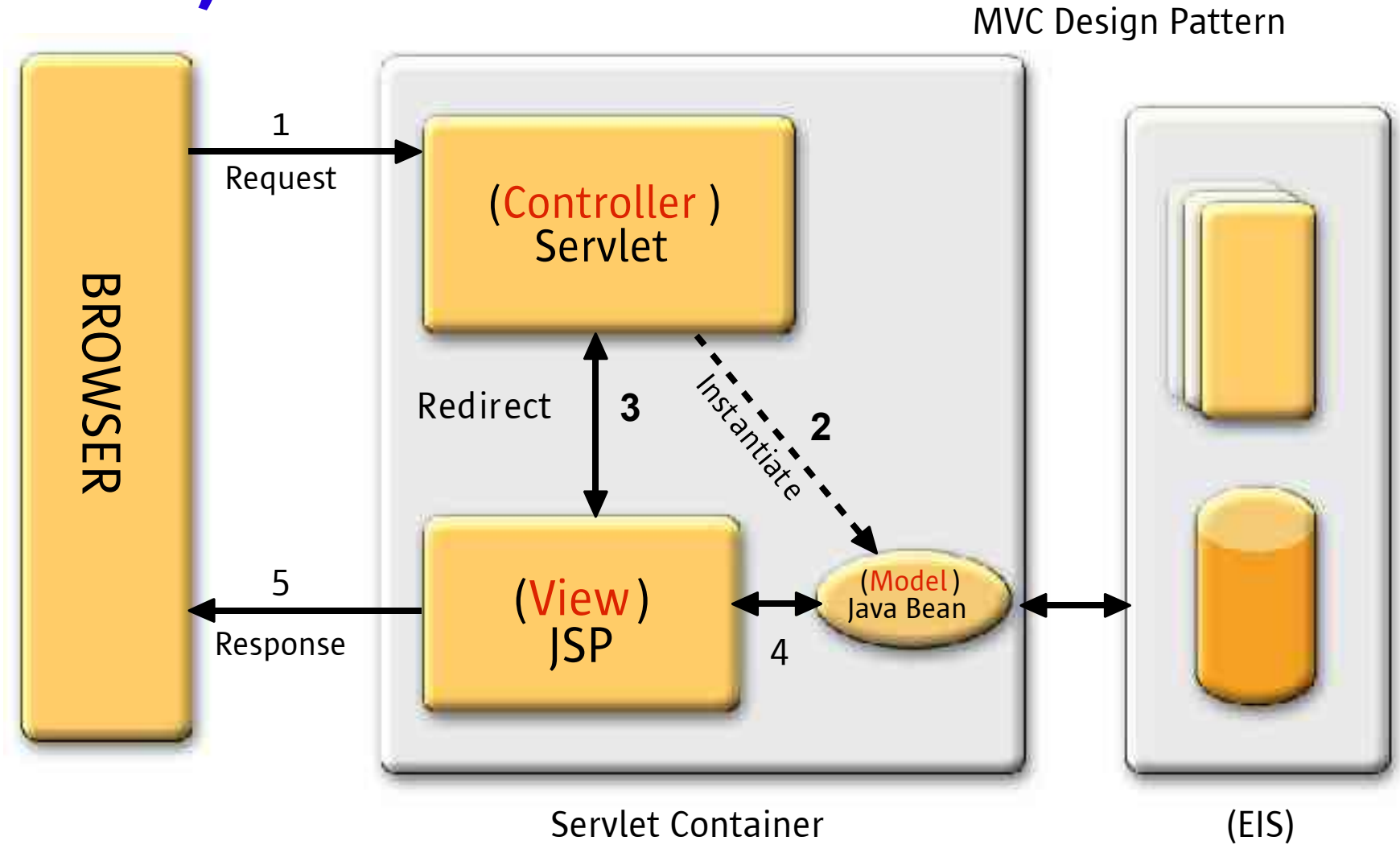
Page-centric Scenario





Model 2 (Servlet-Centric Architecture)

Model 2 Architecture (Servlet-centric)



Why Model 2 Architecture?

- What if you want to present different JSP pages depending on the data you receive?
  - JSP technology alone even with JavaBeans and custom tags (Model 1) cannot handle it well
- Solution
  - Use Servlet and JSP together (Model 2)
  - Servlet handles initial request, partially process the data, set up beans, then forward the results to one of a number of different JSP pages

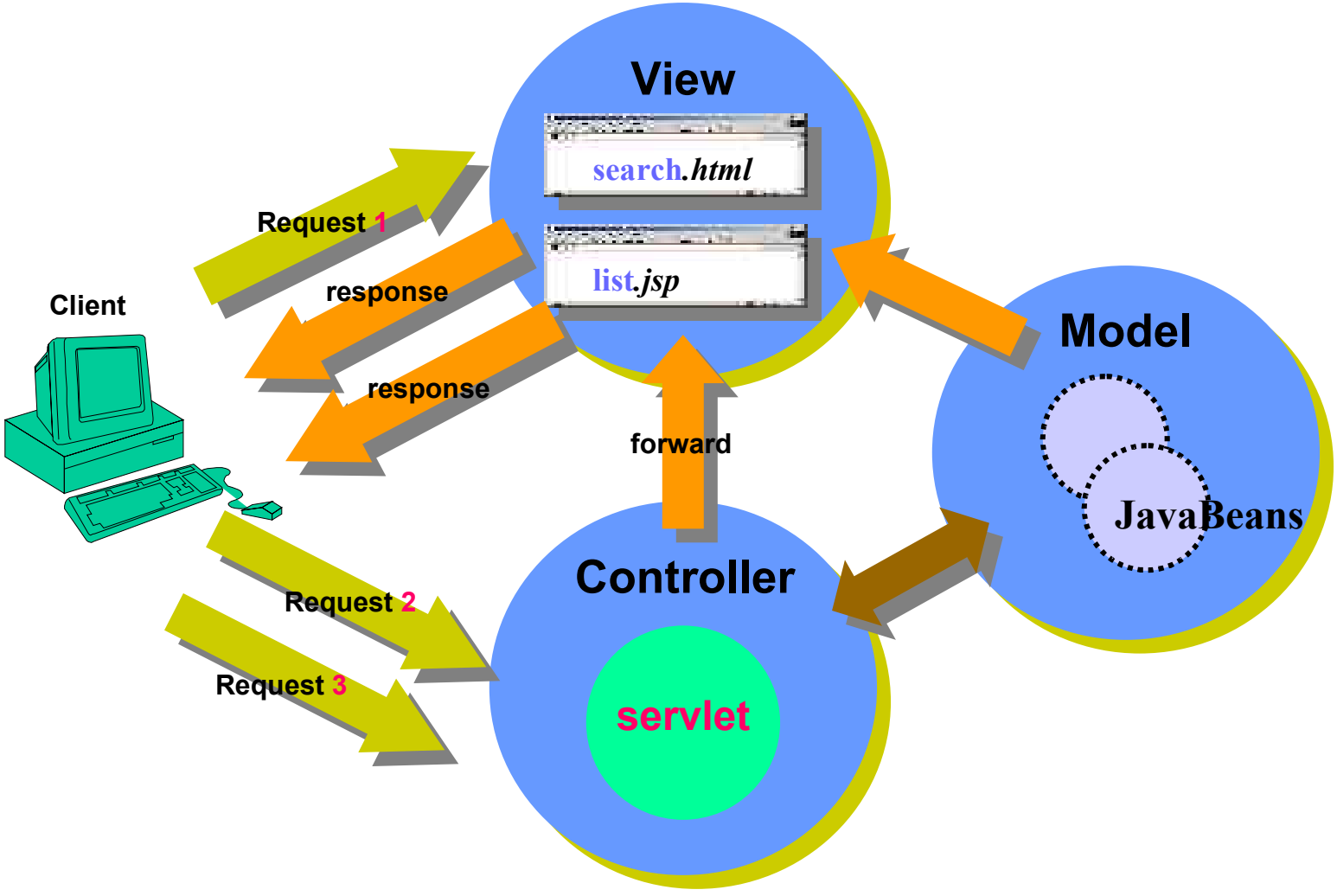
Servlet-centric Architecture

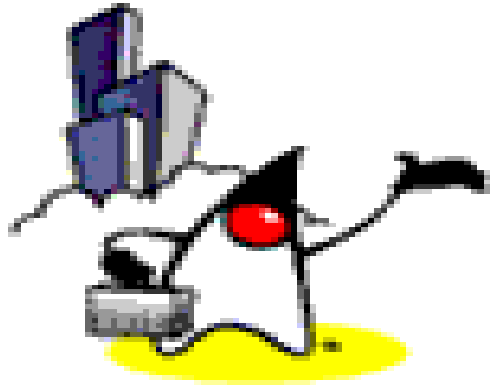
- JSP pages are used only for presentation
  - Control and application logic handled by a servlet (or set of servlets)
- Servlet serves as a gatekeeper
  - Provides common services, such as authentication, authorization, login, error handling, and etc
- Servlet serves as a central controller
  - Act as a state machine or an event dispatcher to decide upon the appropriate logic to handle the request
  - Performs redirecting

How many Servlets in Servlet-centric Approach?

- It depends on the granularity of your application
  - One master Servlet
  - One servlet per use case or business function
  - Combination of the two
    - master servlet handles common function (i.e. common login) for all business functions
    - master servlet then delegates to child servlets for further gatekeeping tasks

Servlet-centric Scenario





Dispatching and Saving Data in both Model 1 and Model 2 Architectures

Flow Control: Dispatching Requests

- How to control the flow?

```
String url="relativeURL";
RequestDispatcher dispatch =
  request.getRequestDispatcher(url);
//OR
String url="abosluteURL"
RequestDispatcher dispatch = getServletContext().
  getRequestDispatcher(url);
```

- Use forward or include methods:
  - Call forward to completely transfer control to destination page
  - Call include to insert output of destination page and then continue on

Storing Data in a JavaBean in Request

- Store data that servlet looked up and that JSP will use in this request
- Sevlet: store data

```
BeanClass value = new BeanClass(..)
request.setAtrribute("bean", value)
```

- JSP: retrieve data

```
<jsp: useBean id=”bean” class=”BeanClass”
        scope=”request”/>
```

Storing Data in Session

- Store data that servlet looked up and that JSP will use in this request and in later requests from the same client
- Sevlet: store data

```
BeanClass value = new BeanClass(..);
HttpSession session=request.getSession(true);
session.setAttribute("bean", value);
```

- JSP: retrieve data

```
<jsp:useBean id = "bean" class="BeanClass"
   scope="session" />
```

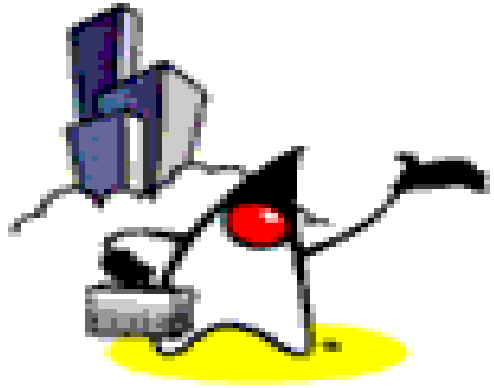
Storing Data in Servlet Context

- Store data that servlet looked up and that JSP will use in this request and in later requests from the any client
- Sevlet: store data

```
BeanClass value = mew BeanClass(..);
getServletContext().setAttribute("bean, value);
```

- JSP: retrieve data

```
<jsp:useBean id = "bean" class="BeanClass"
   scope="application" />
```



When to Use Model 1 or Model 2?

Model 1 (Page-centric)

- May encourage spaghetti JSP pages
  - Business logic may get lost in the display pages
    - Use JavaBeans or custom tags that captures business logic (instead of scriptlets)
  - Page selection is done by each page
- JSPs are harder to debug than straight Java code:
  - Result in a failed compilation and a long list of useless compiler errors referring to the auto-generated code

Model 2 (Servlet-centric)

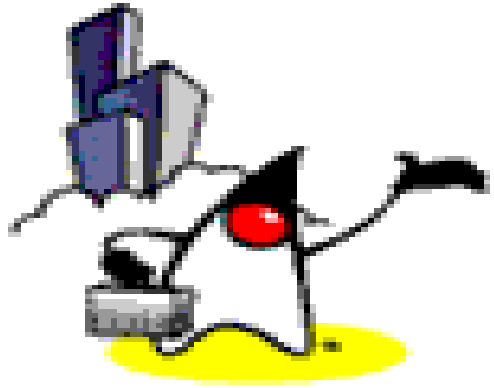
- Loosens the coupling between the pages and improves the abstraction between presentation and application logic
  - Use JSPs for pure data display and input collection activities
  - Most of the business logic can be debugged through the servlet before passed to JavaBeans and JSP

Best Practice Guideline

- Factor out the business logic into business objects and complex display logic into view objects
  - Improves reusability, maintainability, unit testing and regression testing.

How Do I Decide?

- Use page-centric
  - If the application is simple enough that links from page to page.
- Use servlet-centric
  - Each link or button click requires a great deal of processing and decision-making about what should be displayed next.
- "How mapping between requests and responses are done" can help you to decide
  - Each request maps to one and only one response
    - No need for controller.
  - Each request spawns a great deal of logic and a variety of different views can result
    - A servlet is ideal



Web Application Frameworks

Web Application Frameworks

- Based on MVC Model 2 architecture
- Web-tier applications share common set of functionality
  - Dispatching HTTP requests
  - Invoking model methods
  - Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

Why Web Application Framework?

- De-coupling of presentation tier and business logic into separate components
- Provides a central point of control
- Provides rich set of features
- Facilitates unit-testing and maintenance
- Availability of compatible tools
- Provides stability
- Enjoys community-supports
- Simplifies internationalization
- Simplifies input validation

Why Web Application Framework?

- Frameworks have evolved with Java Server technology
- JSP/Servlets are still hard to use
- Frameworks define re-usable components to make this job easier.
- A good framework defines how components work to create a usable application.

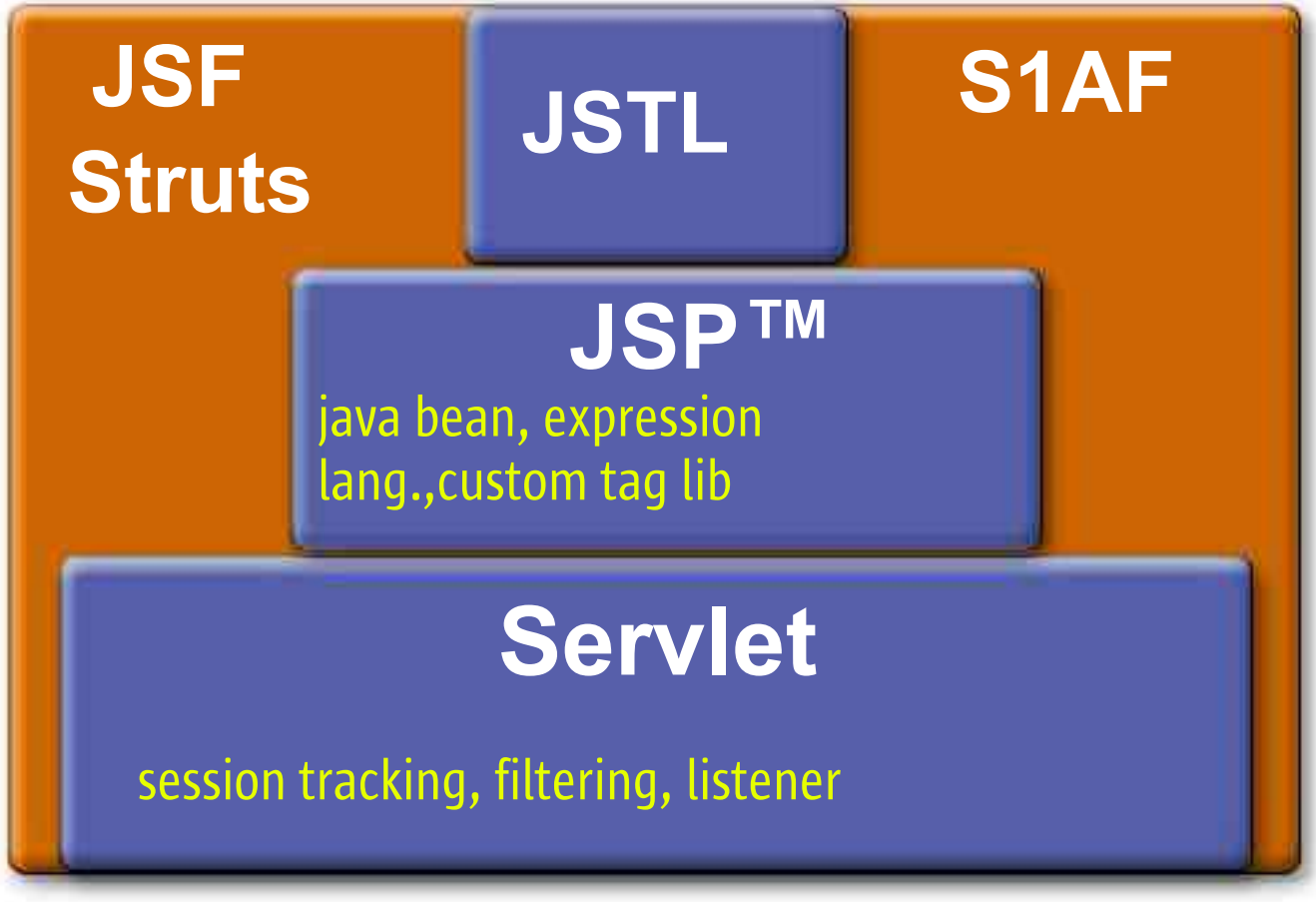
Web Application Frameworks

- Apache Struts
- JavaServer Faces (JSR-127)
  - A server side user interface component framework for Java™ technology-based web applications
- Echo
- Tapestry

Struts and JSF

- Struts is a very popular web application framework
  - Includes tag library for HTML presentation (overlapping area with JSF)
- Struts-Faces Integration Library
  - Use JSF component model for new UI components
  - A few Struts-specific components / renderers
  - Replace use of Struts HTML tags
  - Maintain investment in Struts business logic

How It Fits Together



Passion!