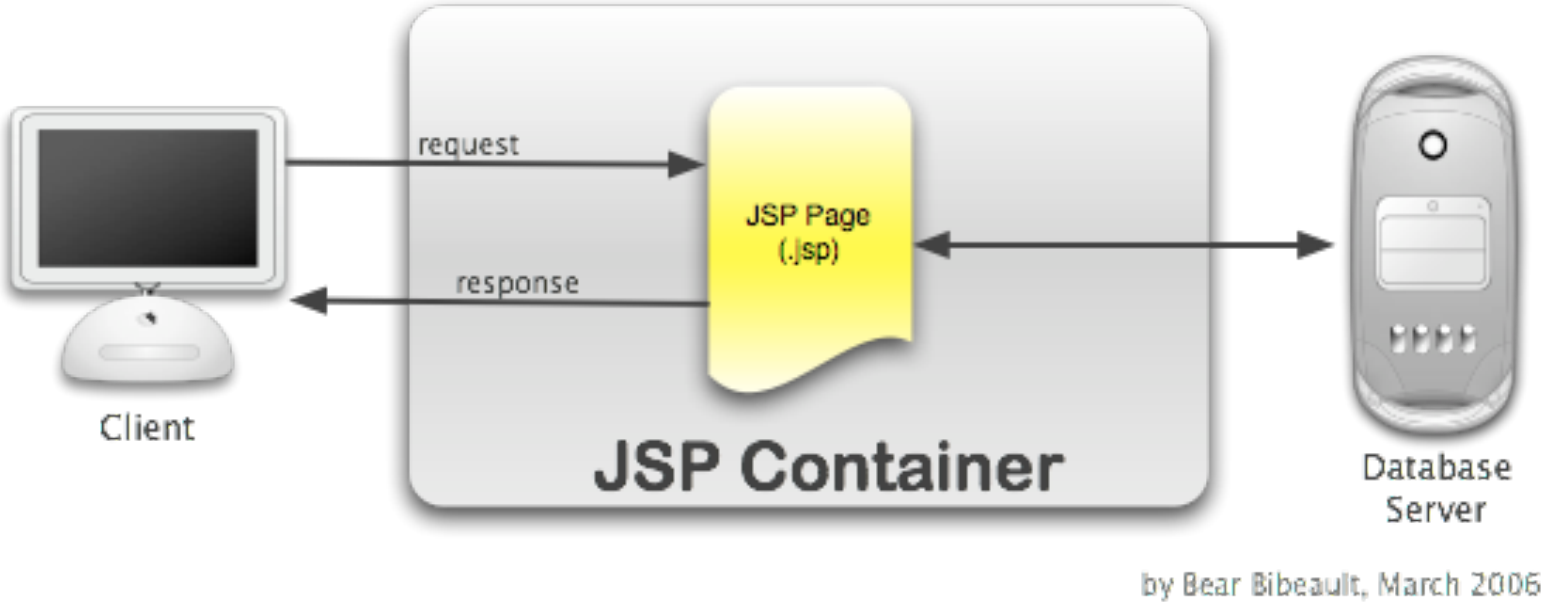
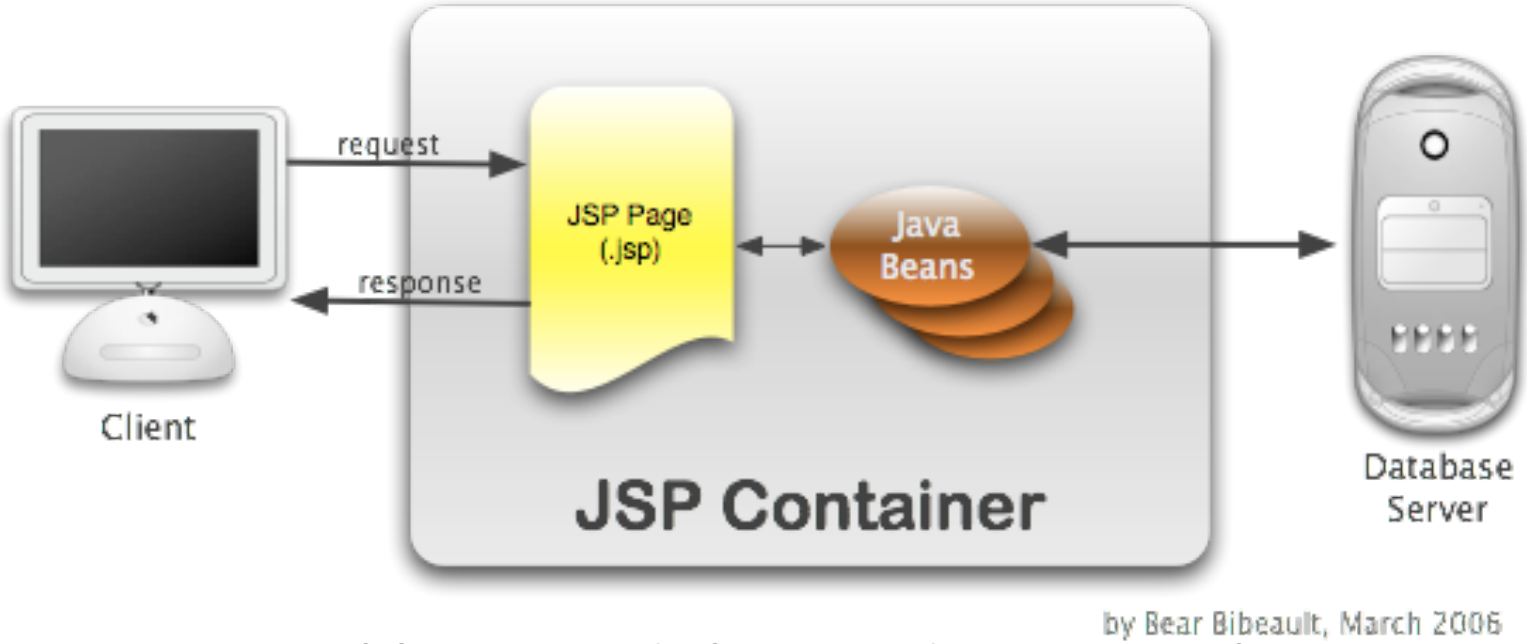
JSP usage:
MVC pattern

The simple (but wrong) approach

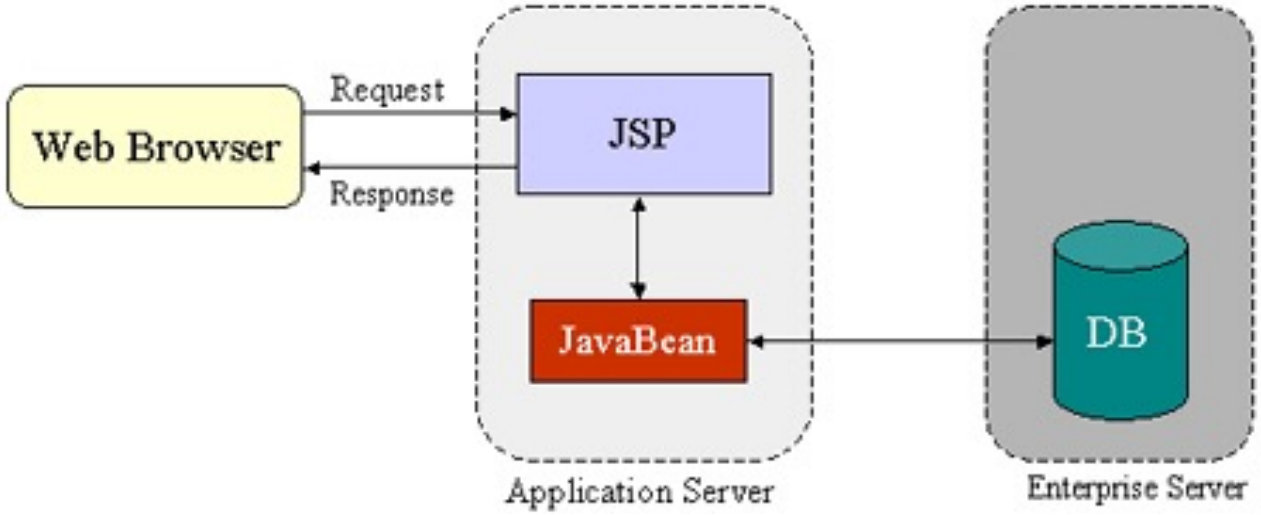


control, data management and presentation
in the same page!

A better solution: JSP model 1



control logic is delegated to Java classes



What is a Java bean?

A bean is a Java class that:

- Provides a public no-argument constructor
- Implements java.io.Serializable
- Follows JavaBeans design patterns
  - Has Set/get methods for properties
  - (Has Add/remove methods for events)
- Is thread safe/security conscious
  - Can run in an applet, application, servlet, ...

Example:

```
public class SimpleBean implements Serializable {
        private int counter;
        SimpleBean() {counter=0;}
        int getCounter() {return counter;}
        void setCounter(int c) {counter=c;}
}
```

Standard actions involving beans

<jsp:useBean id="name" class="fully_qualified_pathname"
scope="{request|session|application}" />

<jsp:setProperty name="nome" property="value" />
<jsp:getProperty name="nome" property="value" />

See: https://www.tutorialspoint.com/jsp/jsp_java_beans.htm

Example – BeanOne.java

```
package beans;
import java.io.Serializable;
public class BeanOne implements Serializable {
    String name;
    String surname;
    public BeanOne() {}
    public String getName() {  return name; }
    public String getSurname() {return surname;}
    public void setName(String name) { this.name = name;}
    public void setSurname(String surname) {this.surname = surname;}
    @Override
    public String toString() {
        return "BeanOne{" + "name=" + name + ", surname=" + surname + '}';
    }
}
```

Example – jspOne.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
      <h1>Setting the property...</h1>
        <jsp:useBean id="myBean1" class="beans.BeanOne" scope="application"/>
        <jsp:setProperty name="myBean1" property="surname" value="de pippis"/>
        <jsp:setProperty name="myBean1" property="name" value="pippo"/>
         <p><%=myBean1.toString()%></p>
</html>
```

Example – jspTwo.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Getting the property...</h1>
        <jsp:useBean id="myBean1" class="beans.BeanOne" scope="session"/>
        <jsp:getProperty name="myBean1" property="surname" />
        <jsp:getProperty name="myBean1" property="name" />
        <p><%=myBean1.toString()%></p>

</html>
```

Example – BeanOne.java

```
import beans.BeanOne;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
@WebServlet(urlPatterns = {"/BeanAccess"})
public class BeanAccess extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        HttpSession session=request.getSession();
        if (session.getAttribute("myBean1")==null) {
            session.setAttribute("myBean1", new BeanOne());
        }
        BeanOne aBean=(BeanOne)(session.getAttribute("myBean1"));
        aBean.setName(aBean.getName()+"z");
        request.getRequestDispatcher("jspTwo.jsp").forward(request, response);
    }
...
}
```

Pay attention to the scope!

JSP

```
<jsp:useBean id="myBean1"
class="beans.BeanOne"
scope="session"/>
```

Servlet

```
Session session=request.getSession();
BeanOne x= (BeanOne )session.getAttribute("myBean1");
```

JSP

```
<jsp:useBean id="myBean1"
class="beans.BeanOne"
scope="application"/>
```

Servlet

```
ServletContext context=request.getServletContext();
BeanOne x= (BeanOne )context.getAttribute("myBean1");
```

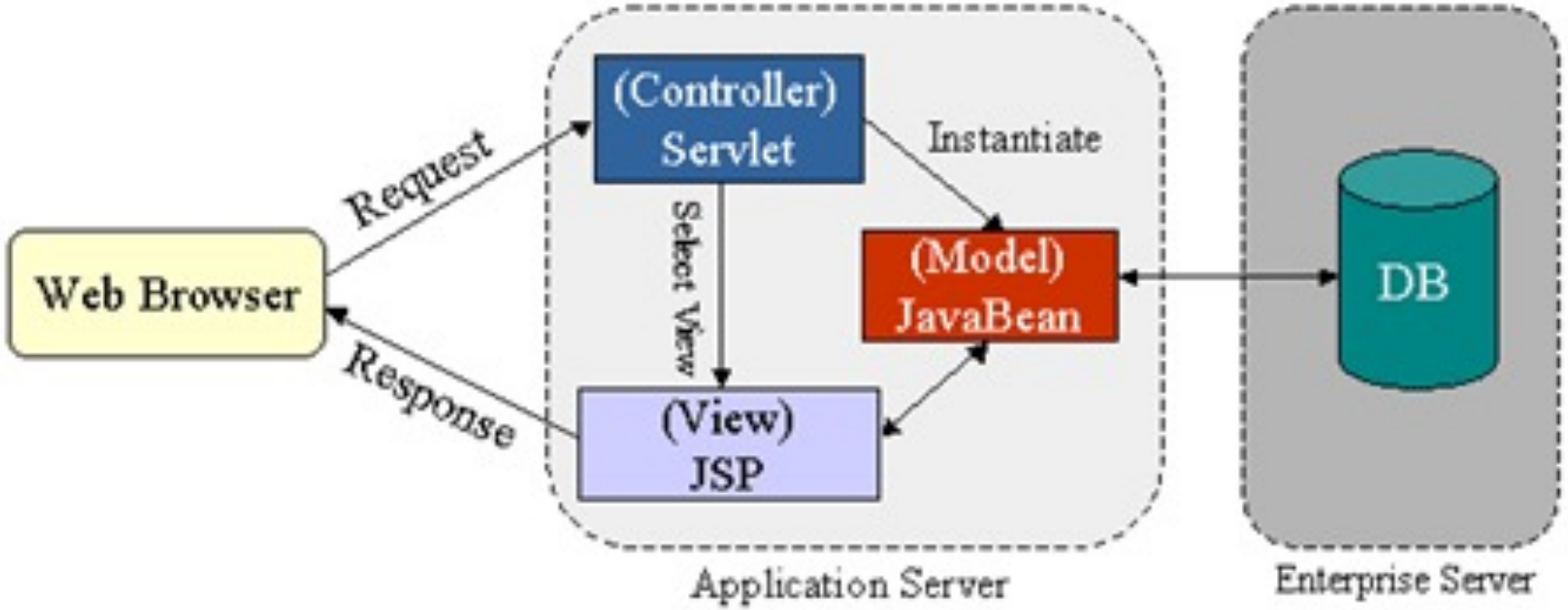
JSP

```
<jsp:useBean id="myBean1"
class="beans.BeanOne"
scope="request"/>
```

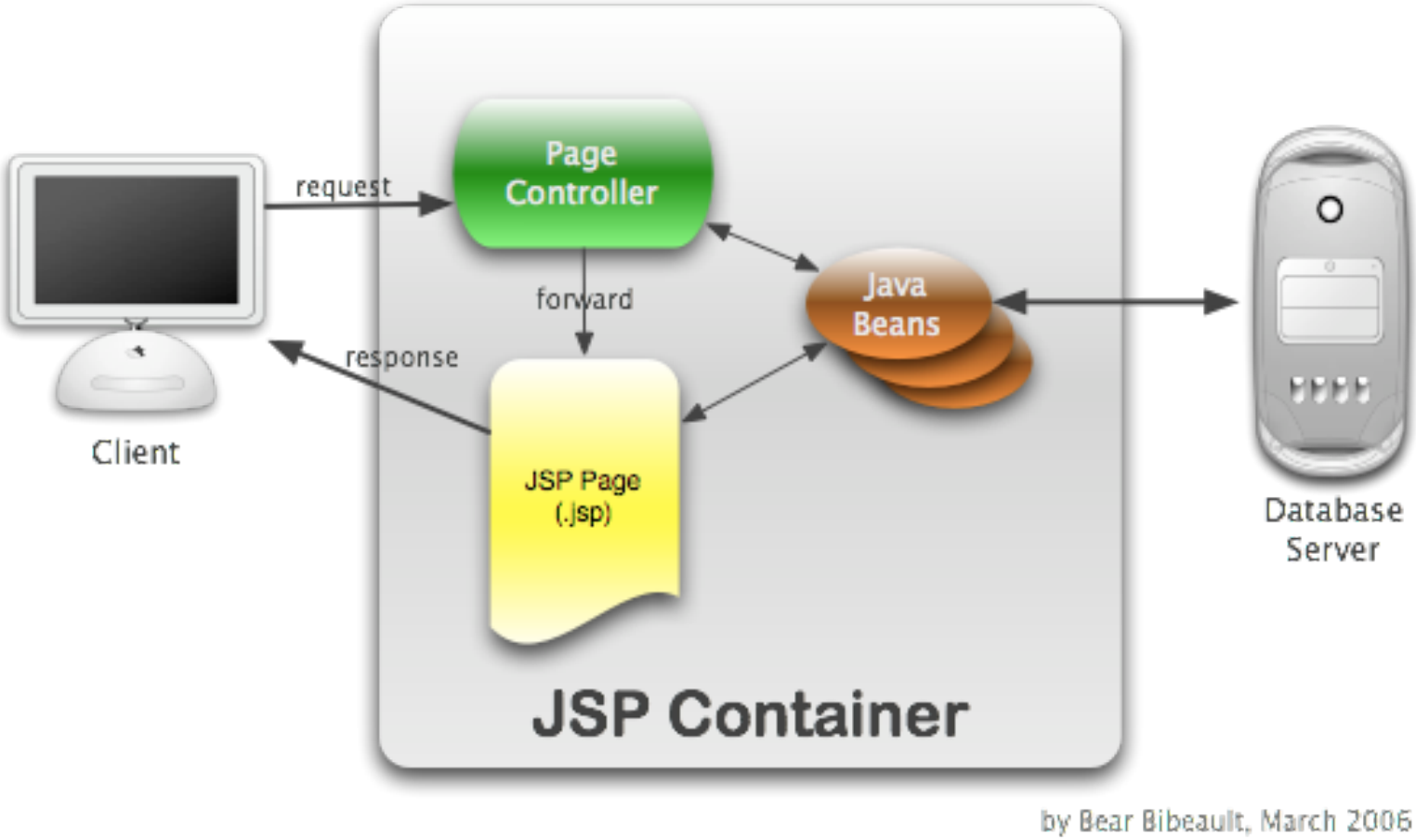
Servlet

```
BeanOne x= (BeanOne )request.getAttribute("myBean1");
```

JSP model 2

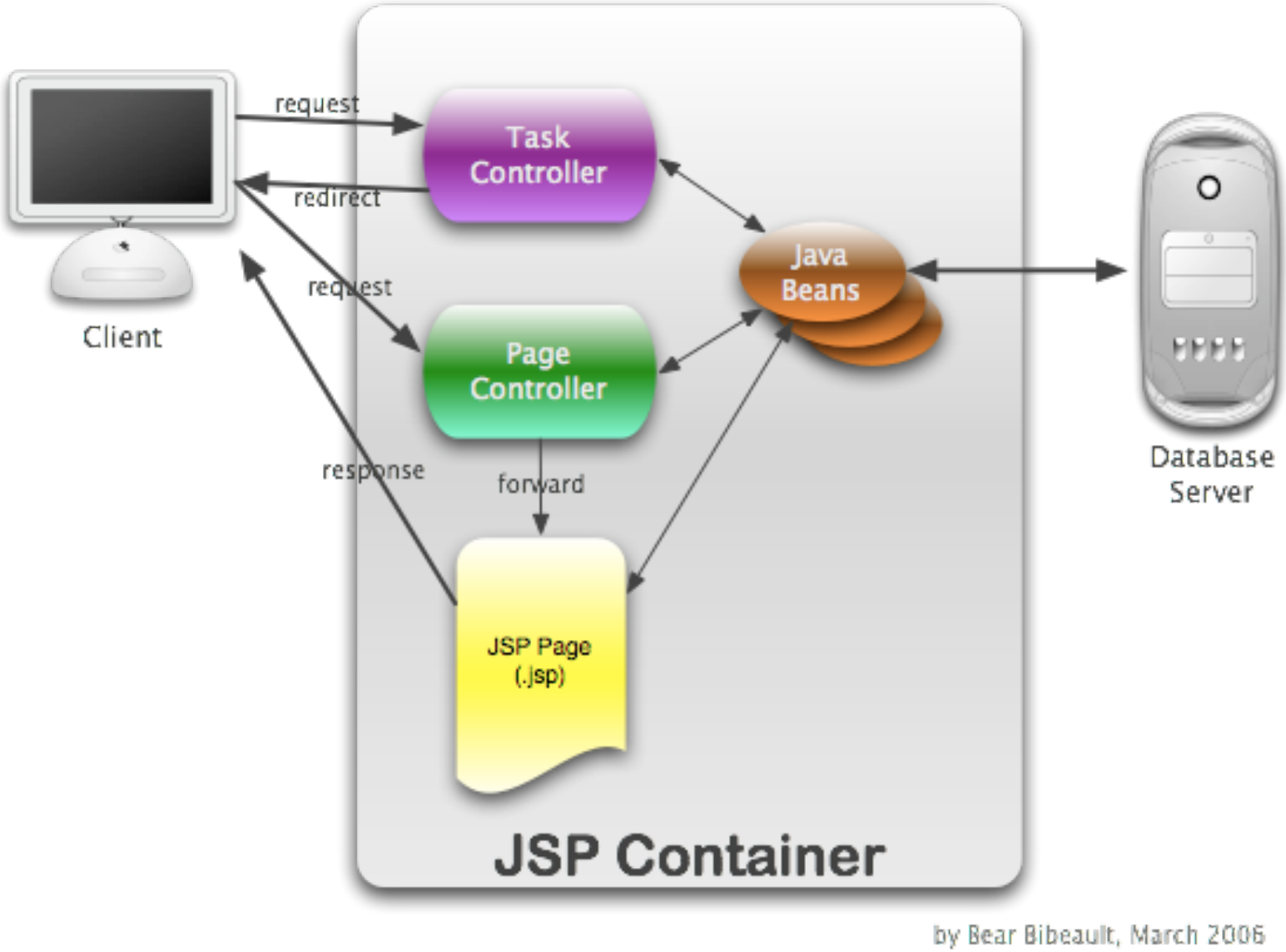

(Controller)
Servlet
Instantiate
Request
Select View
Web Browser
(Model)
JavaBean
DB
Response
(View)
JSP
Application Server
Enterprise Server

JSP used only for presentation

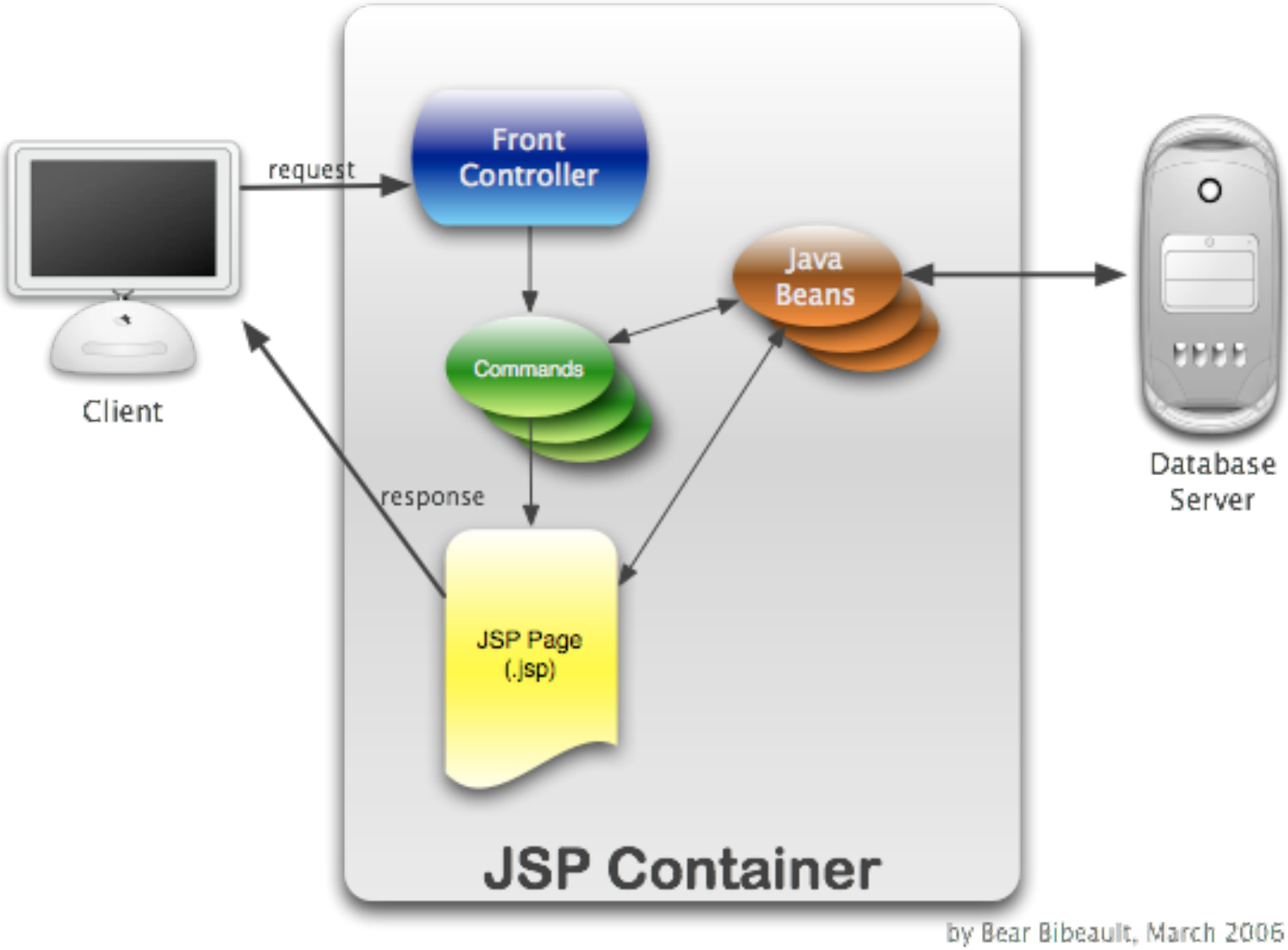


JSP used only for presentation

Let's separate post and get



Front controller pattern



Examples

from:
https://javaranch.com/journal/200603/frontman.html

The Front Controller employed by the Struts package typically uses a servlet mapping of ***.do** where whatever appears as the prefix of the ".do" is used to lookup the actual class path of the Command (called "actions" in Struts) in an internal configuration map.

Another example, the pattern that I usually use, is to employ a servlet mapping such as **/command/***
where the prefix "command" triggers the front controller,
and the rest of the path info is used to lookup the Command class I n an internal map.
It would be typical to see URLs along the lines of:

http://some.server.com/webapp/command/deleteItem
http://some.server.com/webapp/command/insertItem
http://some.server.com/webapp/command/doSomethingWonderful

Best practices

- **Don't overuse Java code in HTML pages**
- **Choose the right include mechanism**:
  Static data such as headers, footers, and navigation bar content is best kept in separate files and not regenerated dynamically.
  Once such content is in separate files, they can be included in all pages using one of the following include mechanisms:
  Include directive: <%@ include file="filename" %>
  Include action: <jsp:include page="page.jsp" />
- **Don't mix business logic with presentation**
  JSP code should be limited to front-end presentation.
- **Use filters if necessary** (See next lecture)
- **Use a database for persistent information** (See next lecture)
  Use connection pooling

Exercise

Transform the servlets-based web app
of Exercise 2
into a JSP-based web app