Extensions and Portlets

Modifying the OpenMRS interface

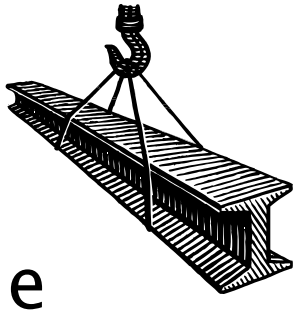
EHSDI

eBuzima

Modifying the interface

- We can use new JSPs and OpenMRS tags in our modules to add new screens to OpenMRS
- But what about modifying the existing screens? Adding items to existing menus?
- OpenMRS provides two main mechanisms for modifying the interface…
  - Extensions
  - Portlets

Extension points



- **Extensions** are classes which a module can define to output something to the view
- **Extension points** are "hooks" in the standard OpenMRS JSPs that extensions can be attached to

- So a module can define an extension, attach it to an existing extension point and output something at that location

Example: help extension

- There is an extension point on the help page */WEB-INF/view/help.jsp*

```
...
<spring:message code="help.text
<br/>

<openmrs:extensionPoint pointId="org.openmrs.help" />

<%@ include file="/WEB-INF/template/footer.jsp" %>
```

This a unique identifier for this point

Example: help extension

- We can use this to put HTML under the standard help page messages, i.e.

Example: help extension

- We start by creating the extension class, which must extend the abstract class `org.openmrs.module.Extension`

```
public class HelpExt extends Extension {

  public MEDIA_TYPE getMediaType() {
    return MEDIA_TYPE.html;
  }

}
```

Every extension must implement this method, and return `MEDIA_TYPE.html` ... because one day extensions might have different media types...

Example: help extension

- The extension outputs HTML by overriding `getOverrideContent`

```
public class HelpExt extends Extension {
  public MEDIA_TYPE getMediaType() {
    return MEDIA_TYPE.html;
  }

  public String getOverrideContent(String bodyContent) {
    return "<b>Help yourself!</b>";
  }
}
```

This content will appear at the extension point in the JSP

Example: help extension

- The extension is hooked to the extension point in the module's *config.xml* file...

```
<module>
  ...

  <extension>
    <point>org.openmrs.help</point>
    <class>@MODULE_PACKAGE@.HelpExt</class>
  </extension>

  ...
</module>
```

The extension point ID from the JSP

The extension class in our module

Example: help extension

- If `getOverrideContent` in the extension class returns non-null, the returned string is simply outputted...

OpenMRS Not logged in | Log in | Help

Home | Find Patient

Openmrs Help

If in doubt, log out...and try again.

Search openmrs.org.

Help yourself!

Extension HTML has been output here

English (United Kingdom) | English (United States) Last Build: May 07 2009 05:35 PM Version: 1.5.0 dev Build

Extension classes

- So the most basic extensions, simply return HTML, and this is displayed at the extension point
- However, in order to ensure consistent styling of HTML, some extension points call methods on attached extensions and expect data, rather than actual HTML
- OpenMRS defines several extension classes which we can extend to provide extensions with the required methods

Extension classes

- For example, the extension point `org.openmrs.headerFullIncludeExt` expects extensions which extend `HeaderIncludeExt`, and thus provide a method to return a list of file paths

/WEB-INF/template/headerFull.jsp

```
...
<openmrs:extensionPoint pointId="org.openmrs.headerFullIncludeExt"
  requiredClass="org.openmrs.module.web.extension.HeaderIncludeExt">

  <c:forEach var="file" items="${extension.headerFiles}">
    <openmrs:htmlInclude file="${file}" />
  </c:forEach>

</openmrs:extensionPoint>
```

Will call `getHeaderFiles` on the extension object

Extension classes

- Thus to create a 'header include' extension, don't override `getOverrideContent`, but implement `getHeaderFiles` instead, e.g.

```
public class MyHeaderIncludeExt extends HeaderIncludeExt {

  public List<String> getHeaderFiles() {
    ArrayList<String> files = new ArrayList<String>();
    files.add("/moduleResources/mymodule/myscript.js");
    return files;
  }
}
```

Extension classes

- OpenMRS defines the following extension classes…
  - `AdministrationSectionExt` – adds links to the administration page
  - `BoxExt` – adds a box to the patient overview
  - `LinkExt` – returns a link with label, URL and required privilege
  - `LinkProviderExt` – returns a list of links
  - `PatientDashboardTabExt` – adds a new tab to the patient dashboard
  - `PortletExt` – returns a portlet URL
  - `TableRowExt` – returns a table row

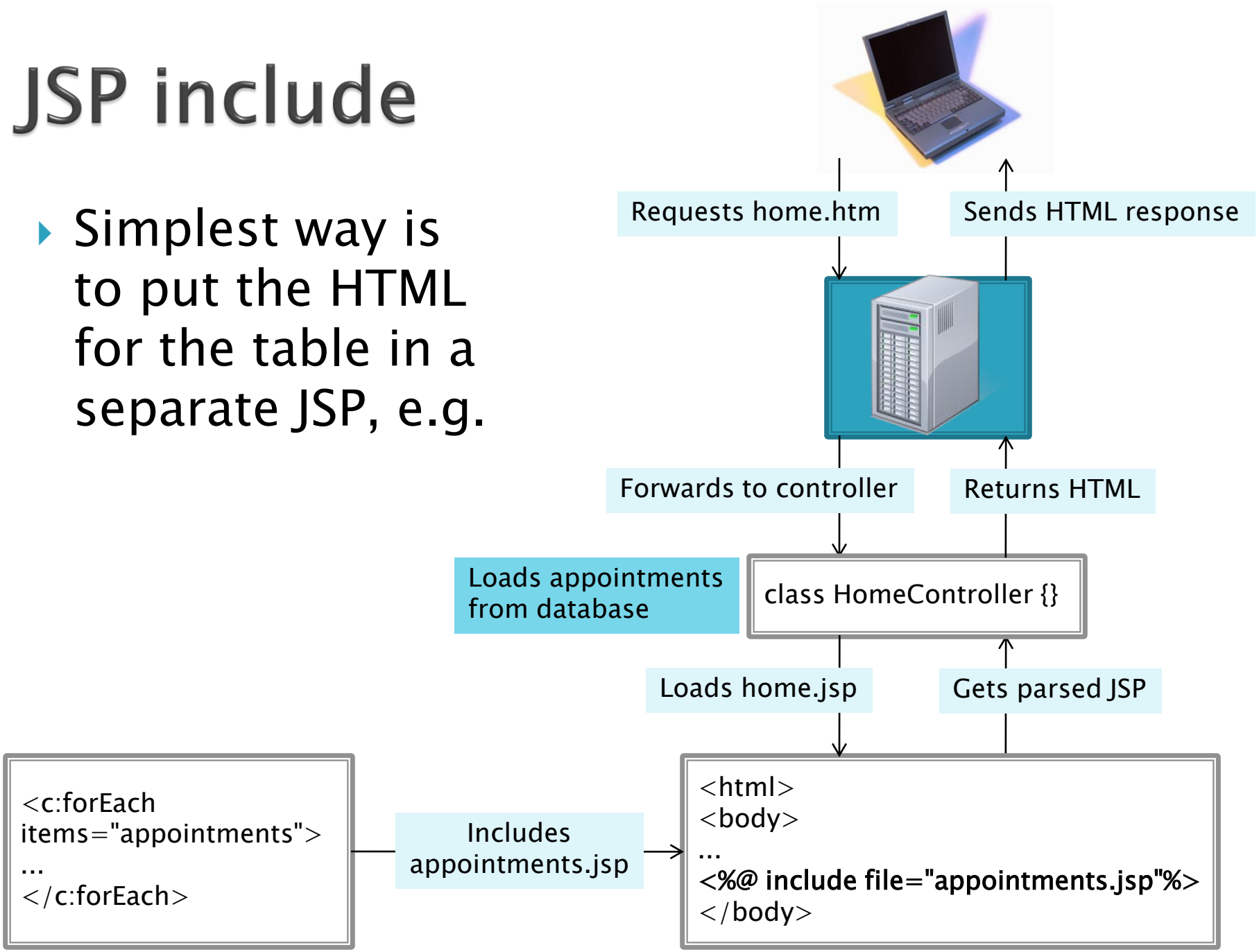
Reusable components

- We've already learnt 3 ways to create reusable components to use in OpenMRS
  - Simple .jsp include
  - Tag file
  - Tag handler class

- For example, we want to create a reusable component which is a table of patient appointments.....

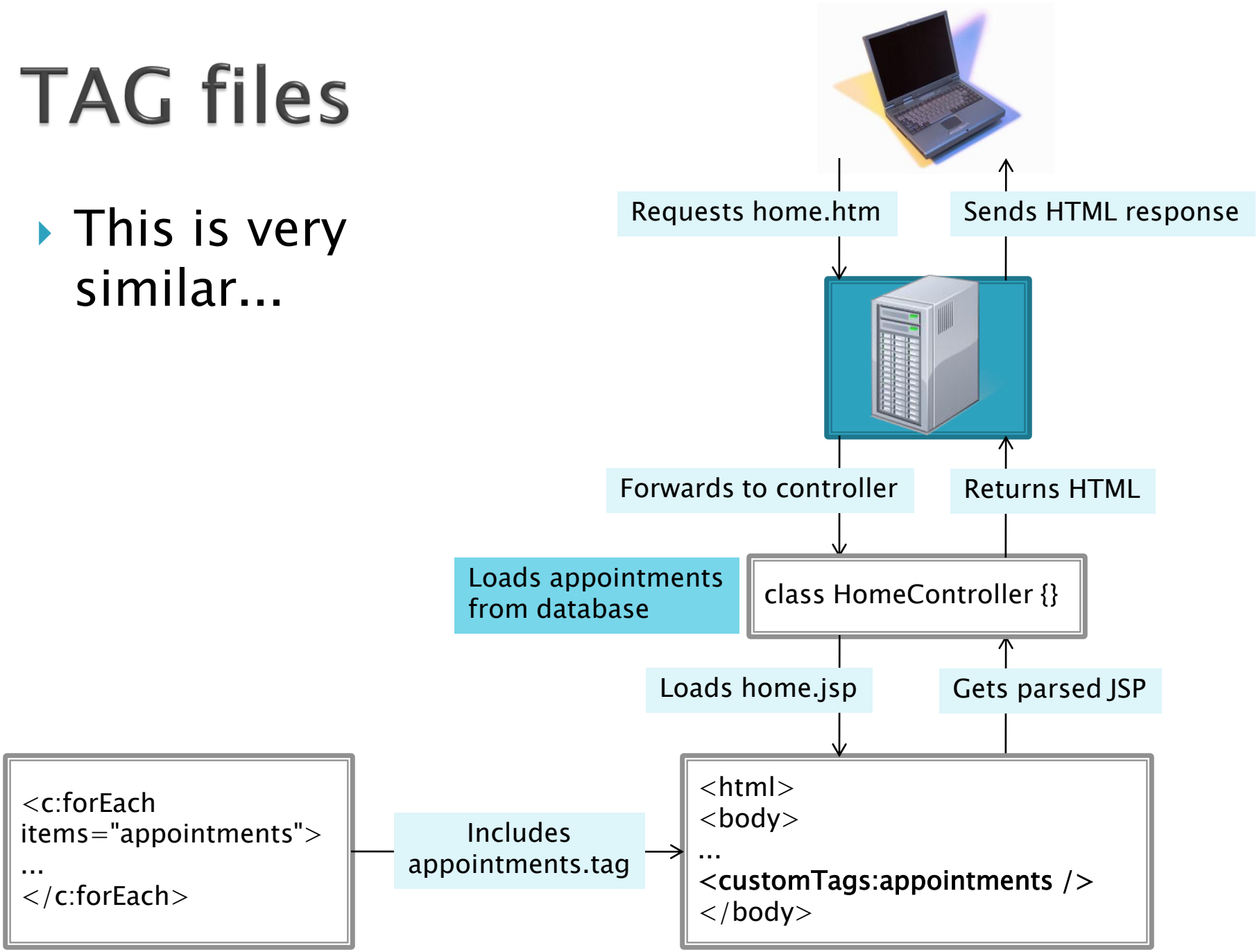
JSP include

- Simplest way is to put the HTML for the table in a separate JSP, e.g.



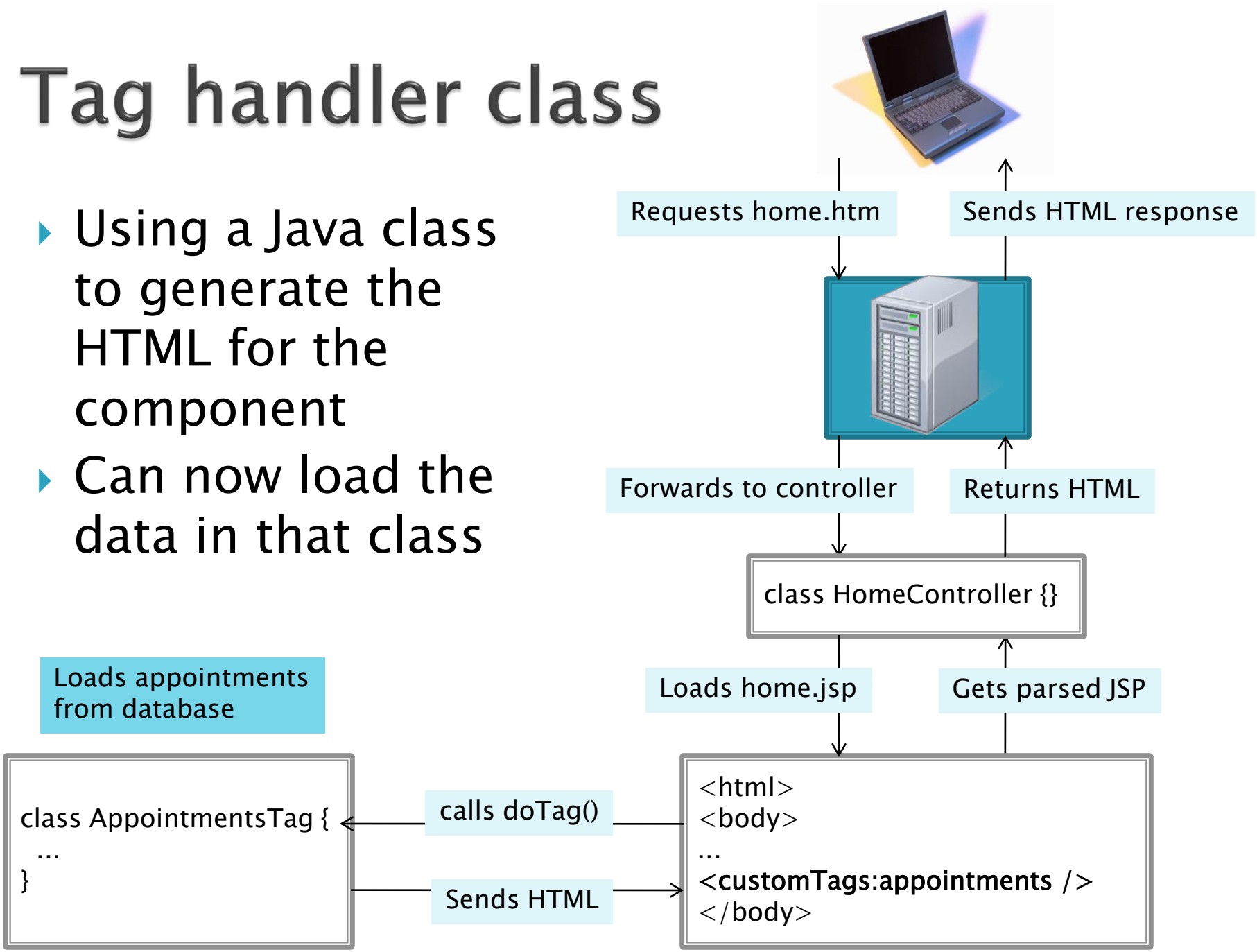
TAG files

- This is very similar...



Tag handler class

- Using a Java class to generate the HTML for the component
- Can now load the data in that class



Issues

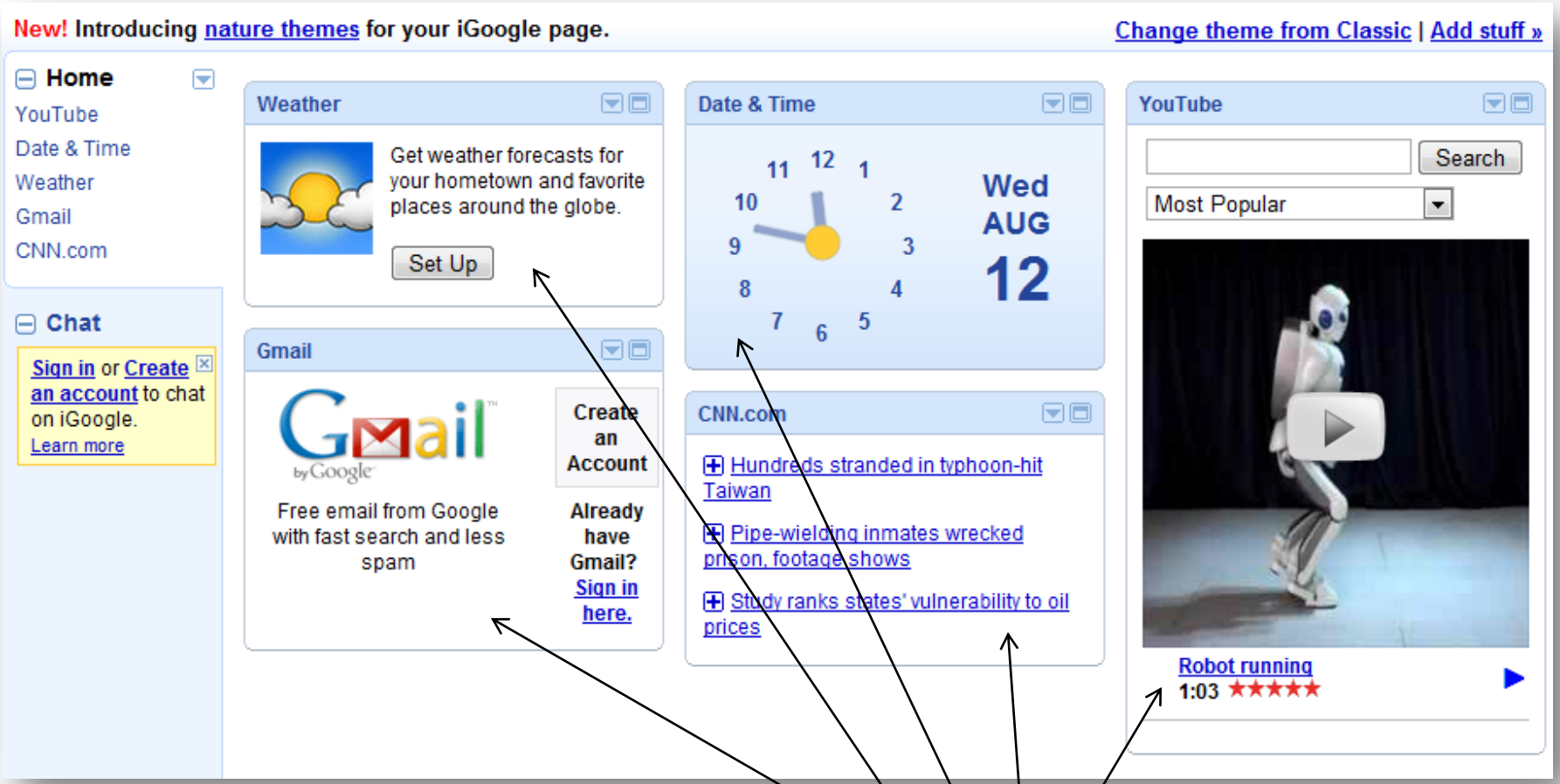
- The obvious disadvantage of the JSP include and TAG file is that the data has to be loaded in the page's controller... not so reusable
- The tag handler class allows us to move the data loading, but this won't be using MVC architecture
- What if we need an easy way of overriding components on existing pages?

- Introducing portlets...

Portlets

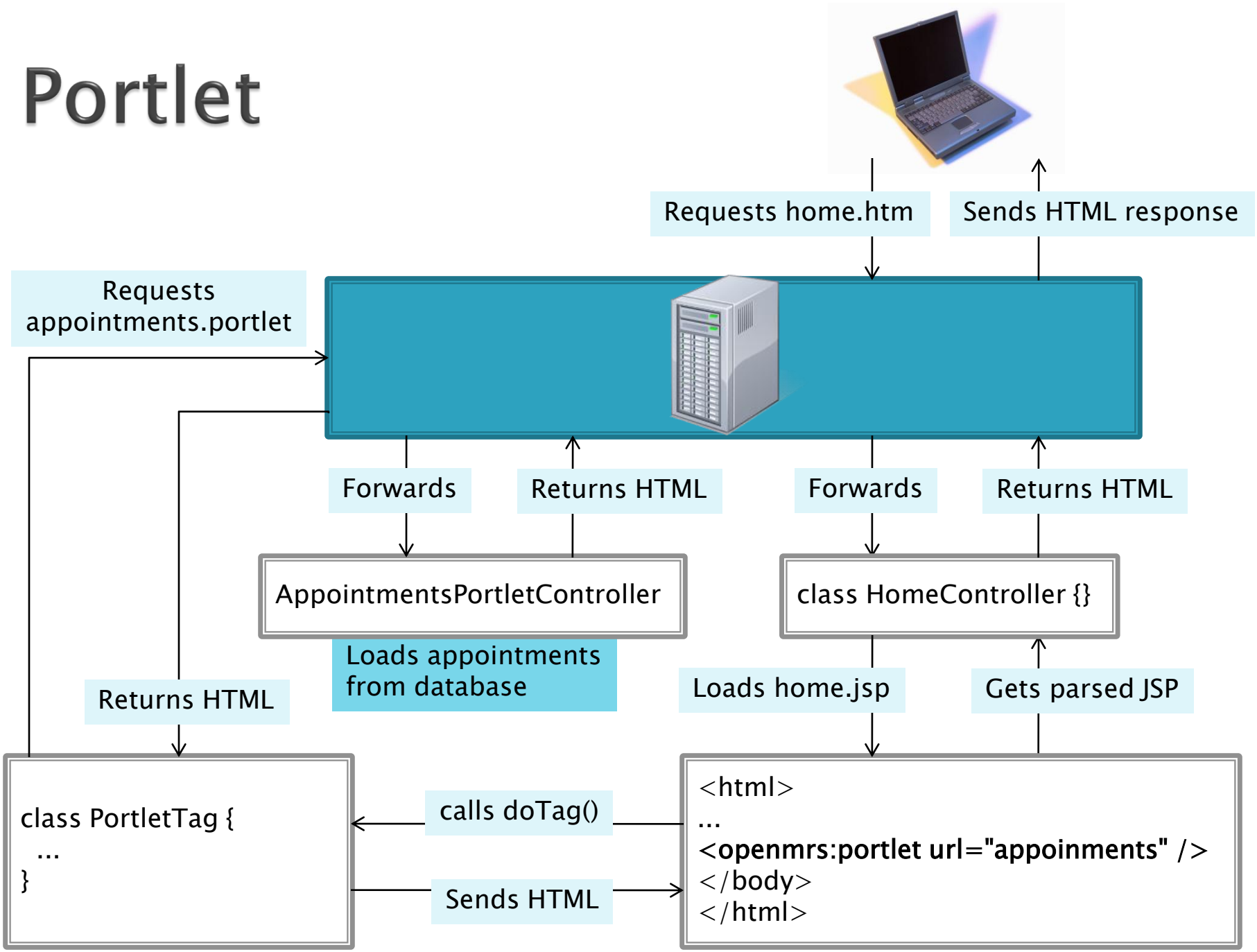
- Portlets are pluggable user interface components
- They are like independent pages, within a page
  - Have a URL
  - Have their own controllers
  - Use their own models

Example: iGoogle



Portlets

Portlet


Requests home.htm
Sends HTML response
Requests appointments.portlet
Forwards
Returns HTML
Forwards
Returns HTML
AppointmentsPortletController
class HomeController {}
Loads appointments from database
Returns HTML
Loads home.jsp
Gets parsed JSP
class PortletTag {
...
}
calls doTag()
Sends HTML
<html>
...
<openmrs:portlet url="appoinments" />
</body>
</html>

Portlets

- Disadvantages
  - Complicated!!!!
- Advantages
  - Have their own controller and model – keeps their data and logic separate from the page's
  - Are fetched by a URL using a server request – therefore can be overridden just by overriding the URL

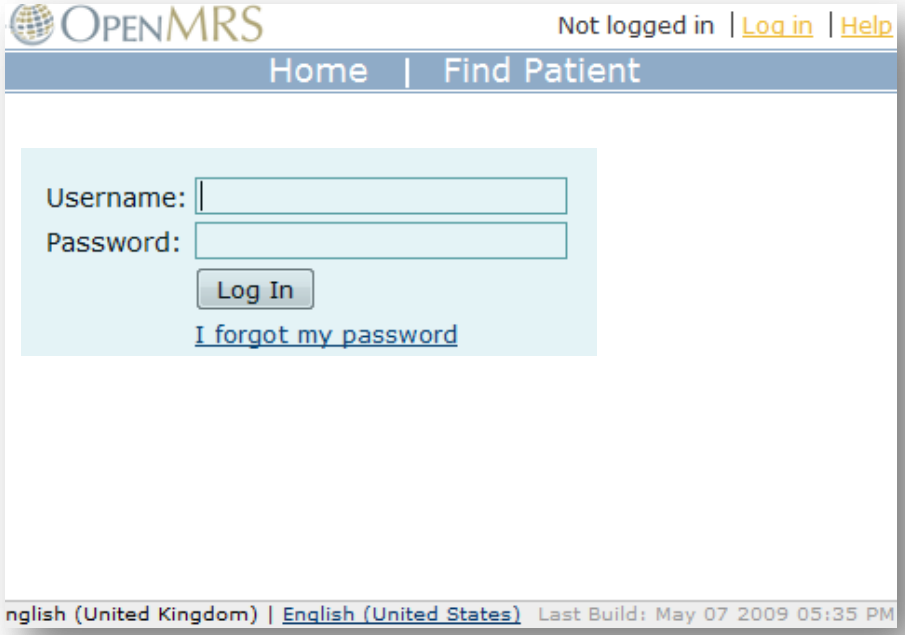
Example: login portlet

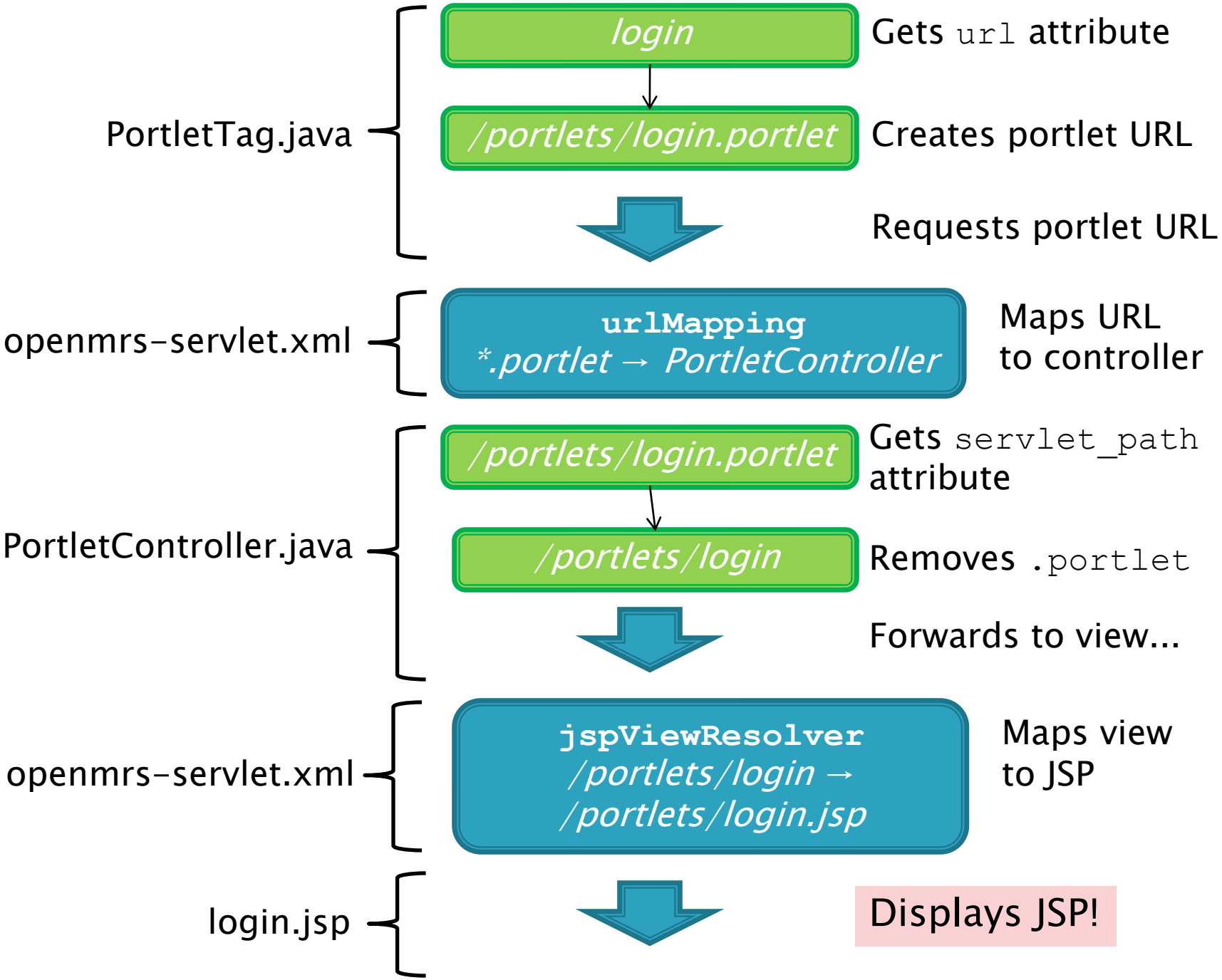
login.jsp

```
...
<openmrs:portlet url="login"/>
...
```

portlet tag is handled by `PortletTag`

login.htm




PortletTag.java
login
Gets url attribute
/portlets/login.portlet
Creates portlet URL
Requests portlet URL
openmrs-servlet.xml
urlMapping
*.portlet → PortletController
Maps URL
to controller
PortletController.java
/portlets/login.portlet
Gets servlet_path
attribute
/portlets/login
Removes .portlet
Forwards to view...
openmrs-servlet.xml
jspViewResolver
/portlets/login →
/portlets/login.jsp
Maps view
to JSP
login.jsp
Displays JSP!

Adding a module portlet...

1. Create a controller which extends `PortletController`
2. Override `populateModel` to create the model for the portlet
3. Add a bean instance of the controller to the module's application context, e.g.

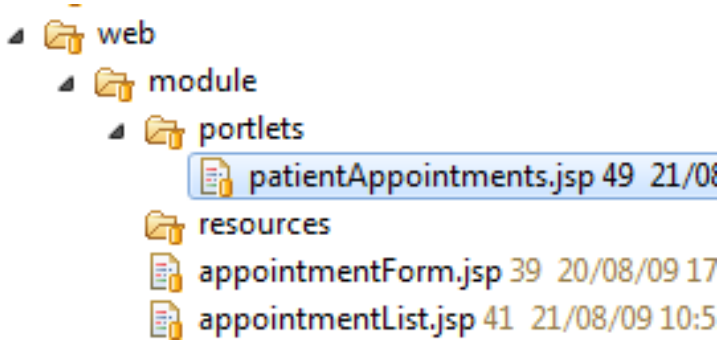
```
<bean
  id="patientAppointmentsPortletController"
  class="org.....web.controller.PatientAppointmentsPortletController"
/>
```

Adding a module portlet...

4. Add an entry to the URL mapping bean to map the portlet's URL to the controller, e.g.

```
<property name="mappings">
  <props>
    <prop key="module/@MODULE_ID@/appointment.form">appointmentFormController</prop>
    <prop key="module/@MODULE_ID@/appointment.list">appointmentListController</prop>
    <prop key="**/patientAppointments.portlet">patientAppointmentsPortletController</prop>
  </props>
</property>
```

5. Place the JSP for the portlet in */web/module/portlets*



Using a module portlet...

- To use a portlet from a module you need to specify in the portlet tag which module it is defined in
- This allows the portlet controller to find the portlet's JSP
- For example:

```
<openmrs:portlet
  url="patientAppointments"
  moduleId="appointments"
/>
```

Overriding portlets

- URLs for controllers in OpenMRS can be overridden by modules
- Put a mapping in the url mapping bean for that module, e.g.

```
<bean class="org.spring...SimpleUrlHandlerMapping">
  <property name="order">
    <value>50</value>
  </property>
  <property name="mappings">
    <props>
      <prop key="/login.htm">customLoginPage</prop>
    </props>
  </property>
</bean>
```

Tells spring that these mappings take priority over OpenMRS's

Overrides the login URL, mapping it to our controller

Overriding portlets

- Portlets are imported by `PortletTag` via a *URL*
- So we can override them by overriding their URL, redirecting it to a controller in our module, e.g.

```
<bean class="org.spring...SimpleUrlHandlerMapping">
  <property name="order">
    <value>50</value>
  </property>
  <property name="mappings">
    <props>
      <prop key="/portlets/login.portlet">customLoginPortlet</prop>
    </props>
  </property>
</bean>
```

Overrides the login portlet, mapping it to our portlet controller

References

- Websites
  - http://openmrs.org/wiki/Module_Extension_Points
  - http://openmrs.org/wiki/Module_Portlets