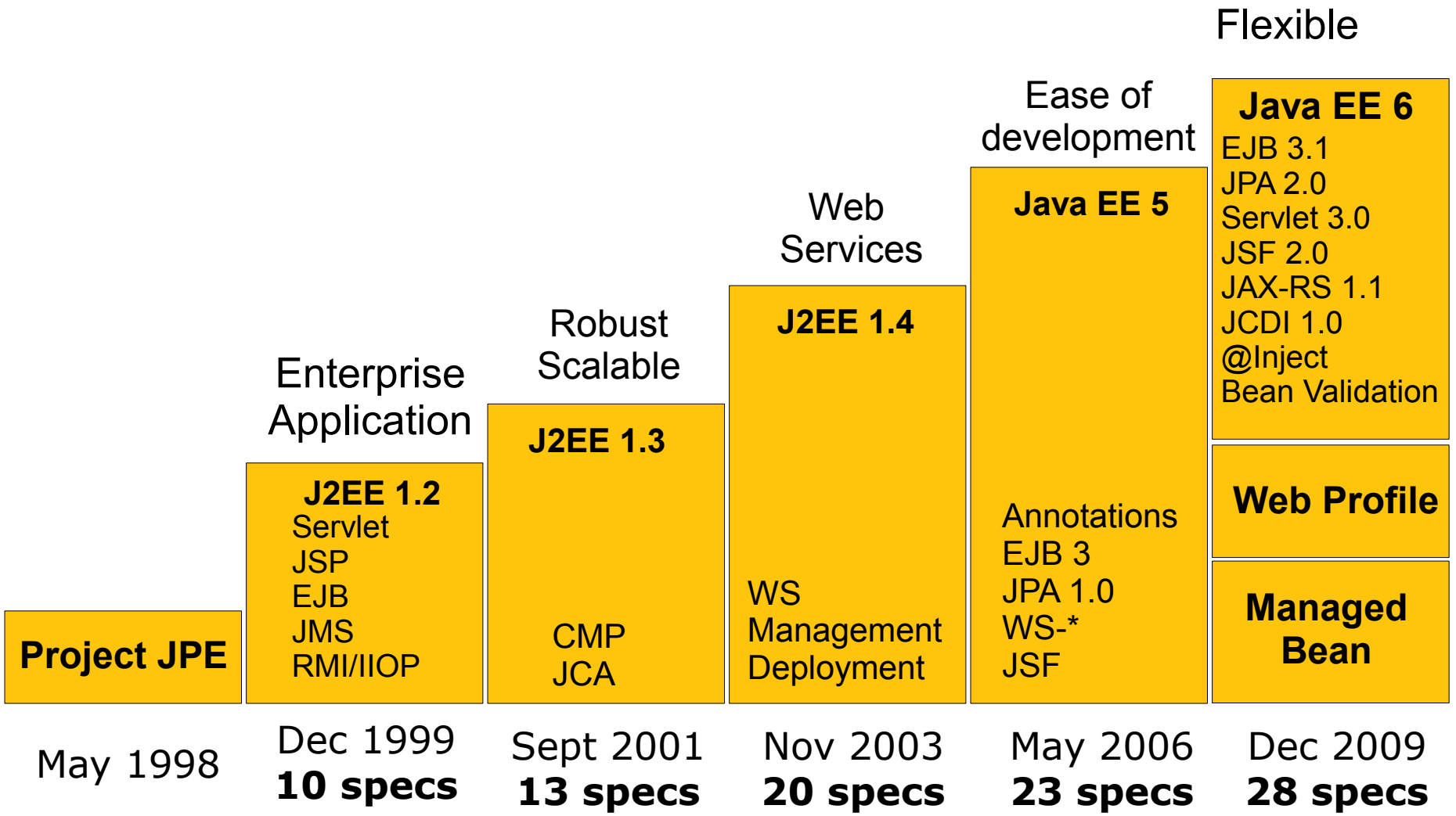
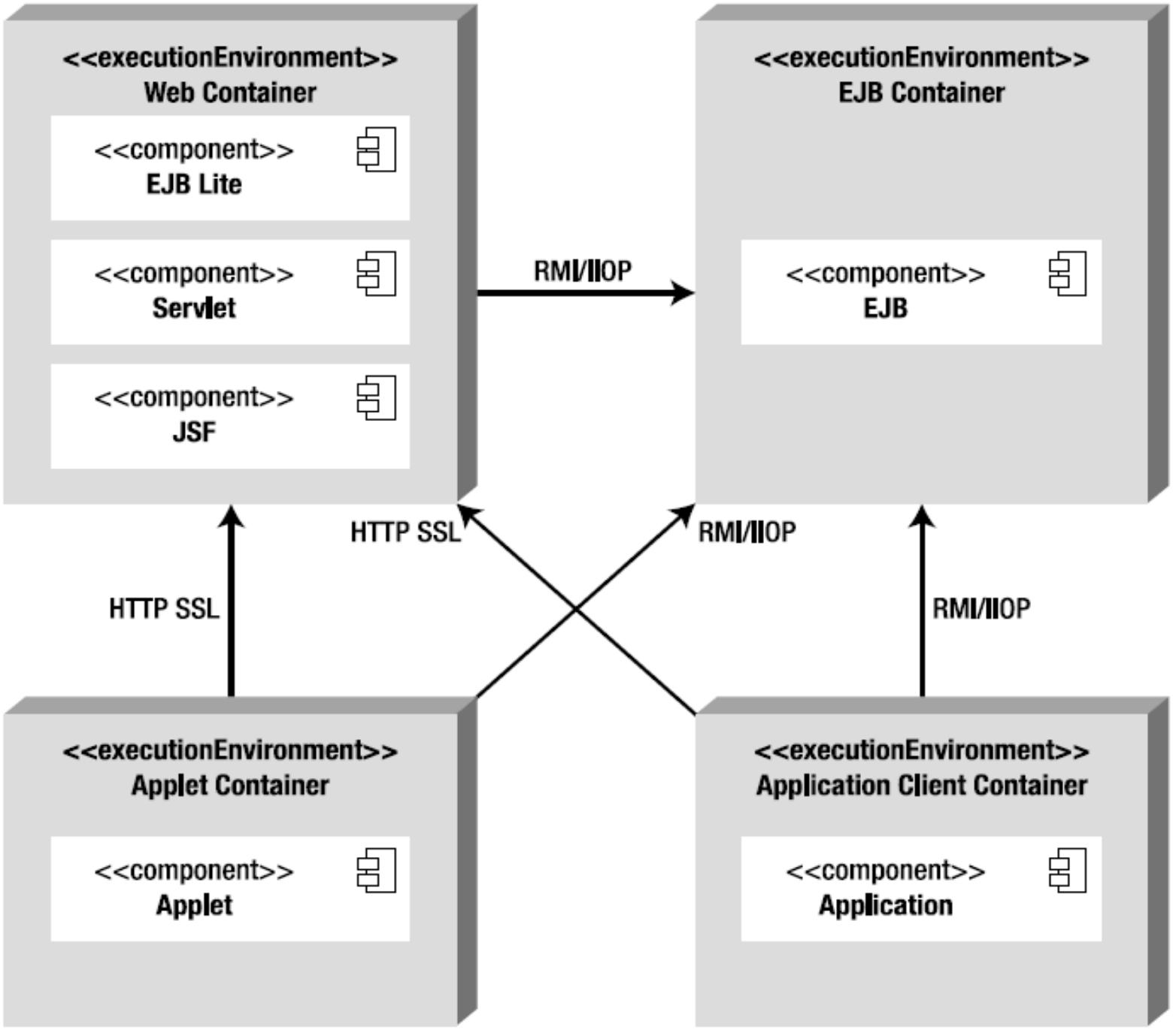
# Servlet

# A brief history of Java EE



# JavaEE Architecture and Standard Containers



# Containers

- The Java EE infrastructure is partitioned into logical domains called containers
- Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before it can be executed, a web, enterprise bean, or application client component must be assembled into a Java EE module and deployed into its container.
- Each container has a specific role, supports a set of APIs, and offers services to components (security, database access, transaction handling, naming directory, resource injection)
- Components deployed in containers can be invoked through different protocols

# Applet container

- Applet containers are provided by most web browsers to execute applet components.
- The applet container uses a sandbox security model where code executed in the “sandbox” is not allowed to “play outside the sandbox.” This means that the container prevents any code downloaded to your local computer from accessing local system resources, such as processes or files.

# Application client container

- The application client container (ACC) includes a set of Java classes, libraries, and other files required to bring injection, security management, and naming service to Java SE applications (Swing, batch processing, or just a class with a main() method).
- ACC communicates with the EJB container using RMI-IIOP and the web container with HTTP (e.g., for web services).

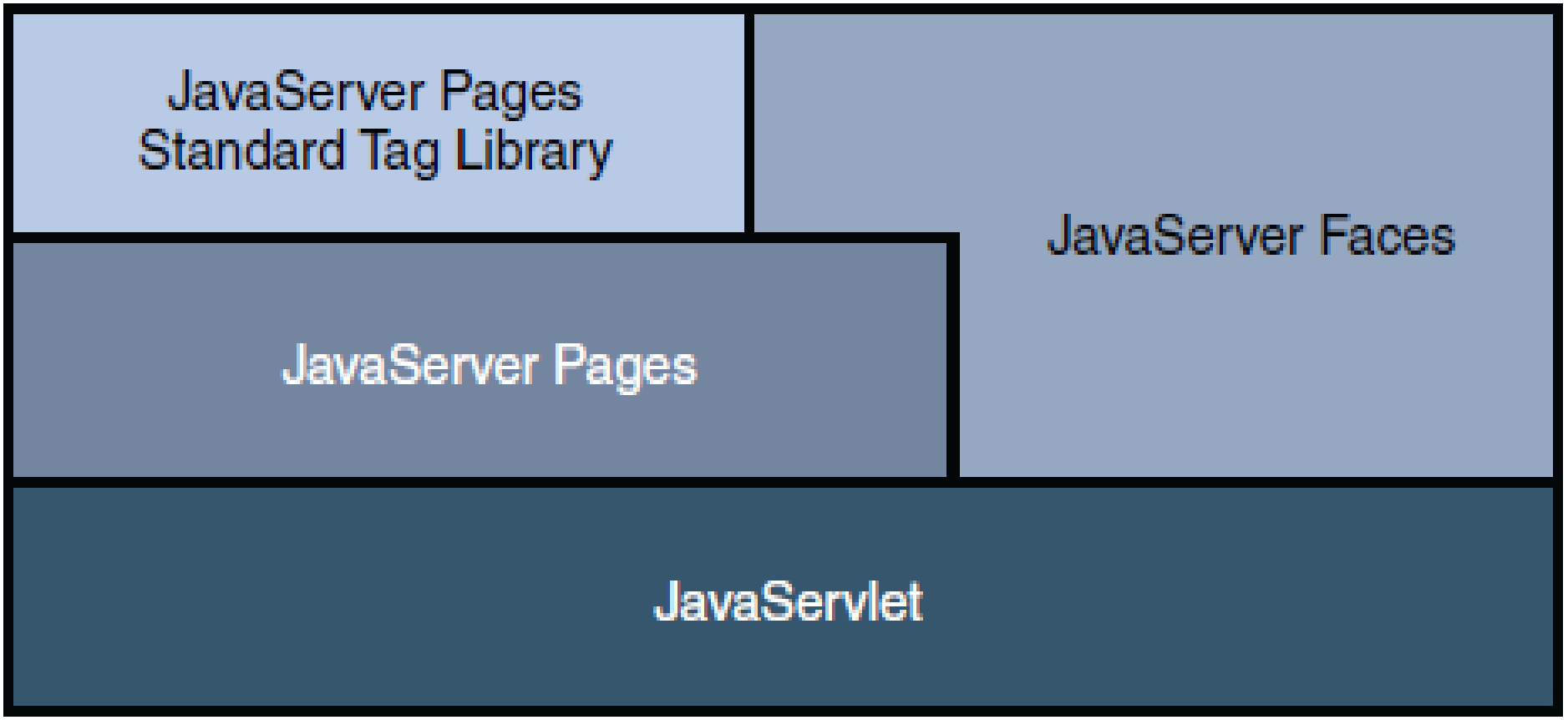
# Web Container

- The web container provides the underlying services for managing and executing web components (servlets, EJBs Lite, JSPs, filters, listeners, JSF pages, and web services).
- It is responsible for instantiating, initializing, and invoking servlets and supporting the HTTP and HTTPS protocols.
- The container used to feed web pages to client browsers

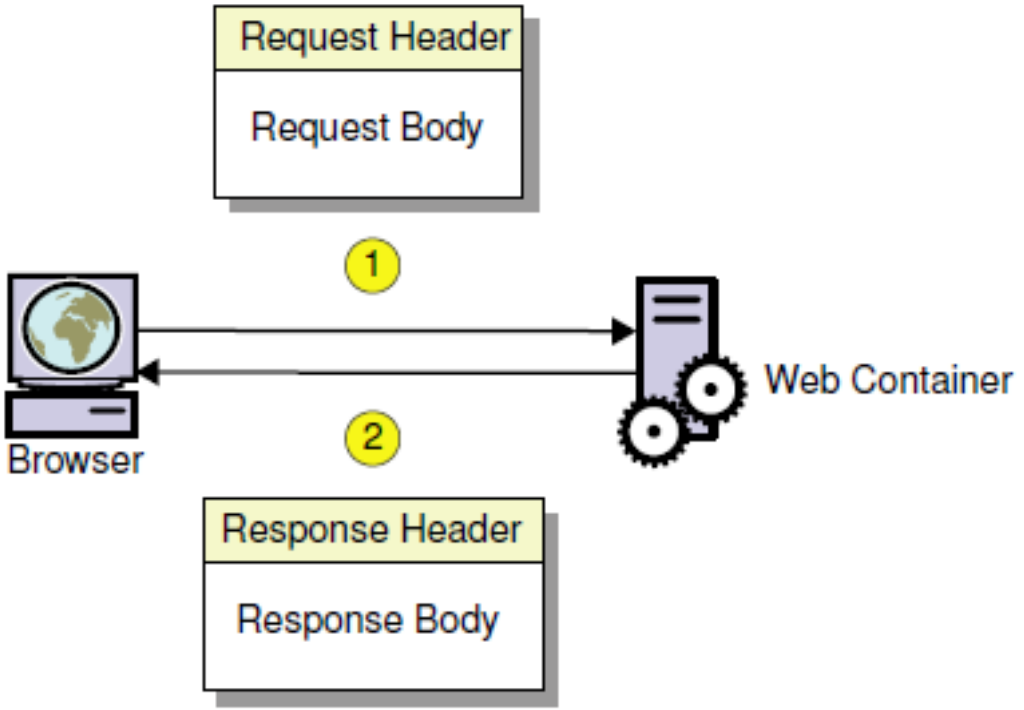
# EJB container

- The EJB container is responsible for managing the execution of the enterprise beans containing the business logic tier of your Java EE application.
- It creates new instances of EJBs, manages their life cycle, and provides services such as transaction, security, concurrency, distribution, naming service, or the possibility to be invoked asynchronously.

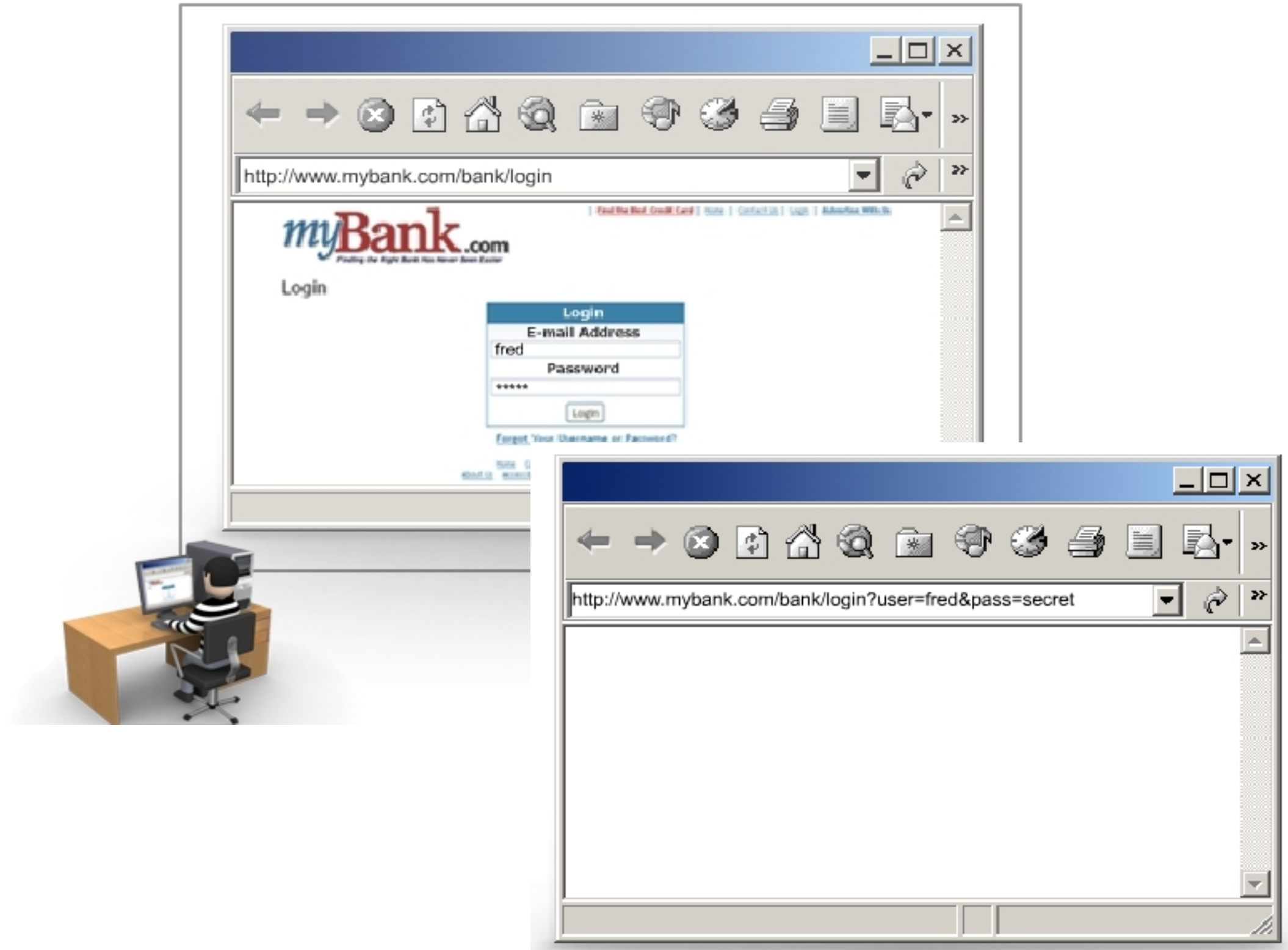
# JavaWebApplication Technologies



# HTTP Request-Response Model
## The GET and POST Requests




| | GET **Request** | POST **Request** |
|---|---|---|
| **Type of Use** | Default | Form submission |
| **Method of Sending Form Data** | • Sent with the URI<br>• Size is limited | • Sent in the request body<br>• Size is unlimited |
| **Display of Form Data** | Browser displays in the URI area | Browser does not normally display with the URI |


http://www.mybank.com/bank/login
myBank.com
Login
Login
E-mail Address
fred
Password
*****
Login
http://www.mybank.com/bank/login?user=fred&pass=secret

# Form Data

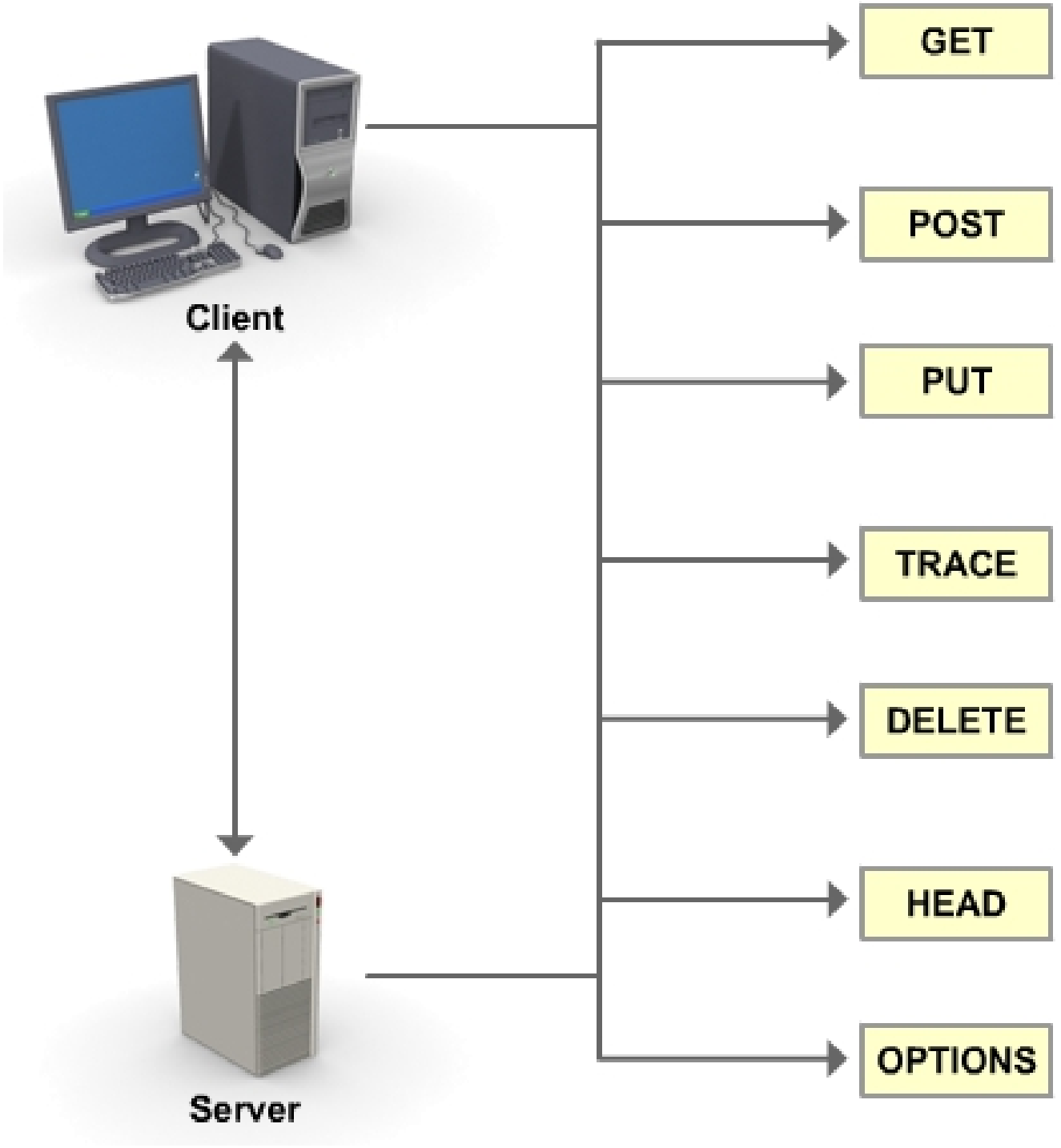
## HTML snippet:

```
<FORM ACTION='form_test' METHOD='POST'>
<INPUT NAME='input1' SIZE='20'/>
<INPUT TYPE='SUBMIT' VALUE='OK'/>
</FORM>
```

## Browser form:


## Browser request:

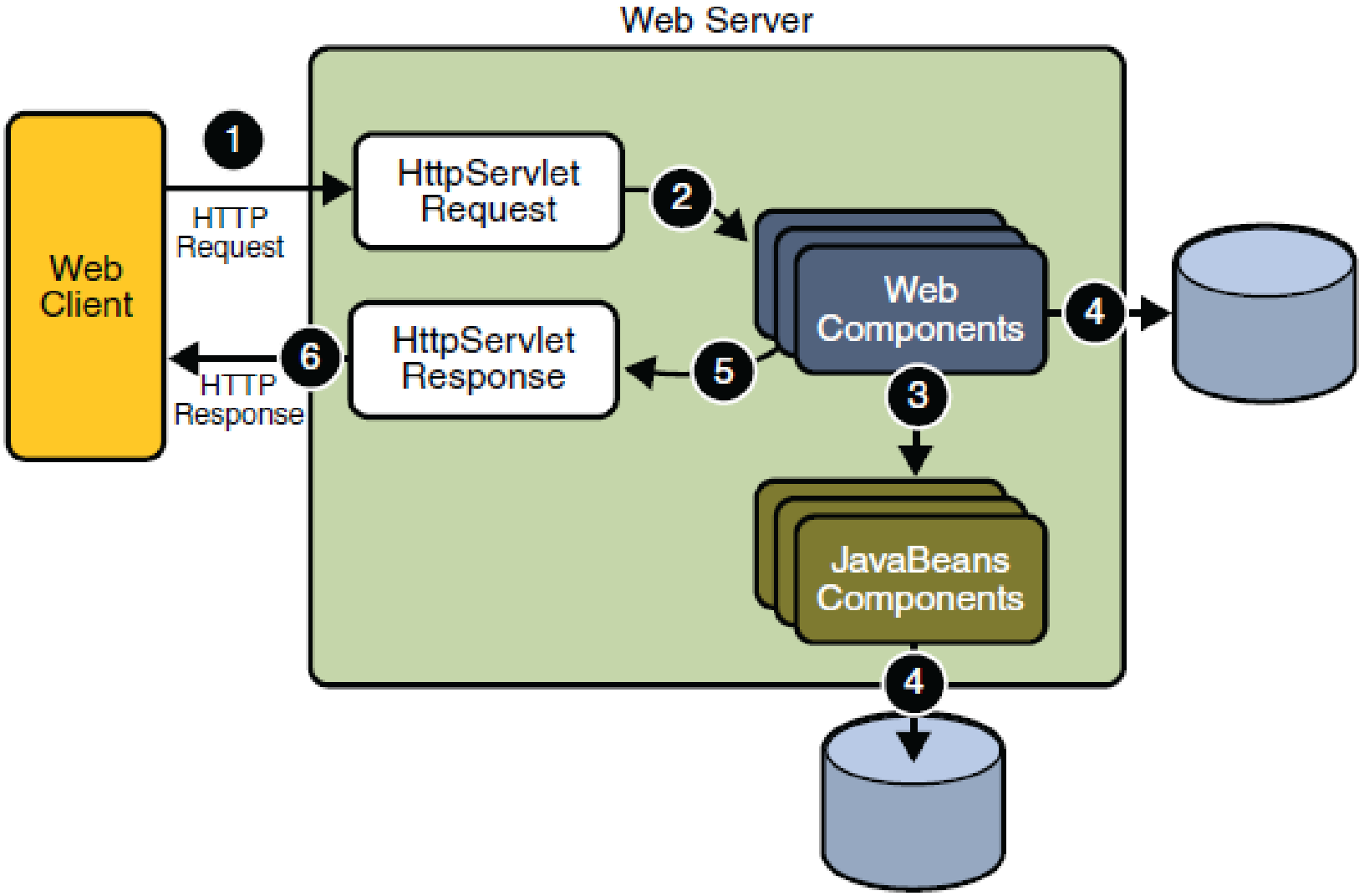
```
POST /bank/form_test HTTP/1.1
... request headers...

input1=this+is+a+test
```


Client
Server
GET
POST
PUT
TRACE
DELETE
HEAD
OPTIONS

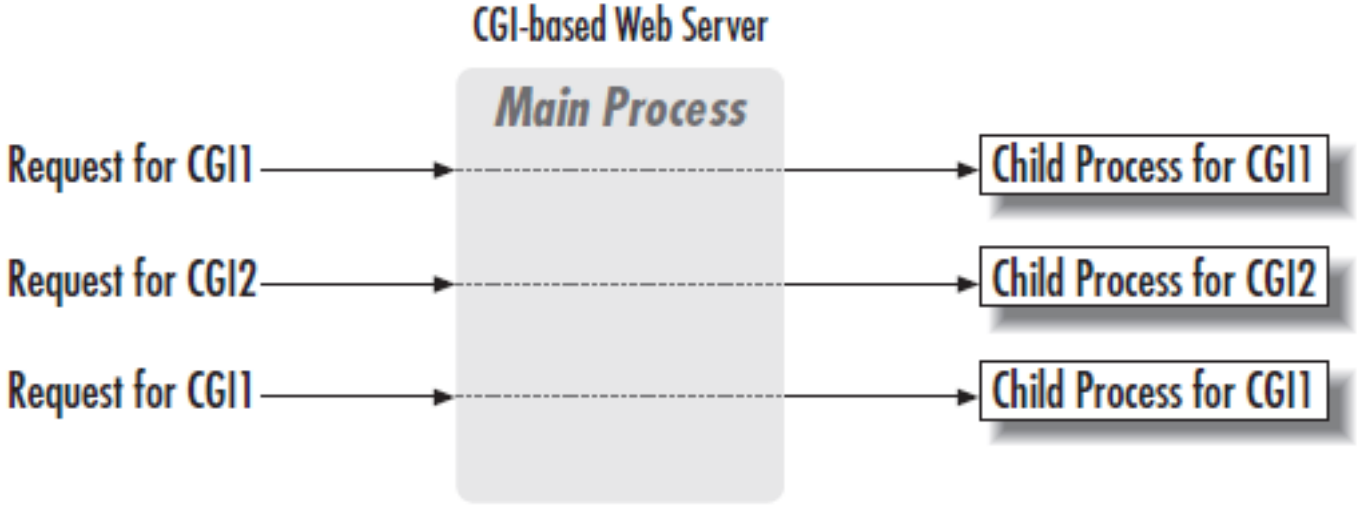
# Content Type and the Response Header

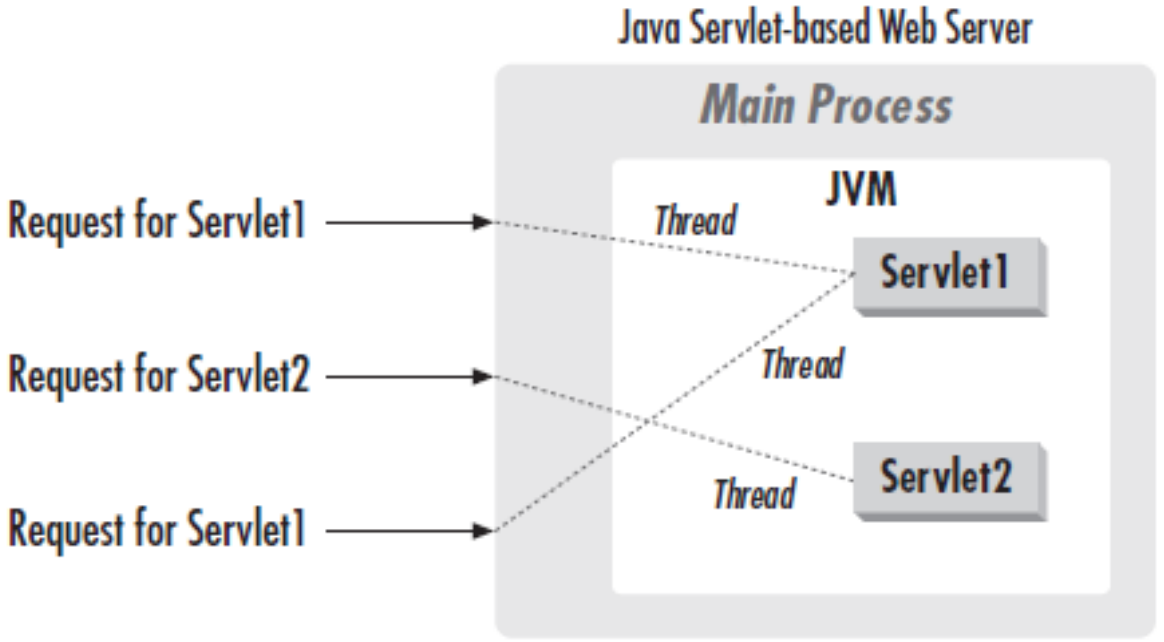
- The server response includes a Content-Type header that contains MIME type values including but not limited to:
  - text/html
  - text/xml
  - image/jpeg
- Examples of additional response headers include:
  - Content-Encoding
  - Content-Length
  - Cache-Control

# Java WebApplication RequestHandling



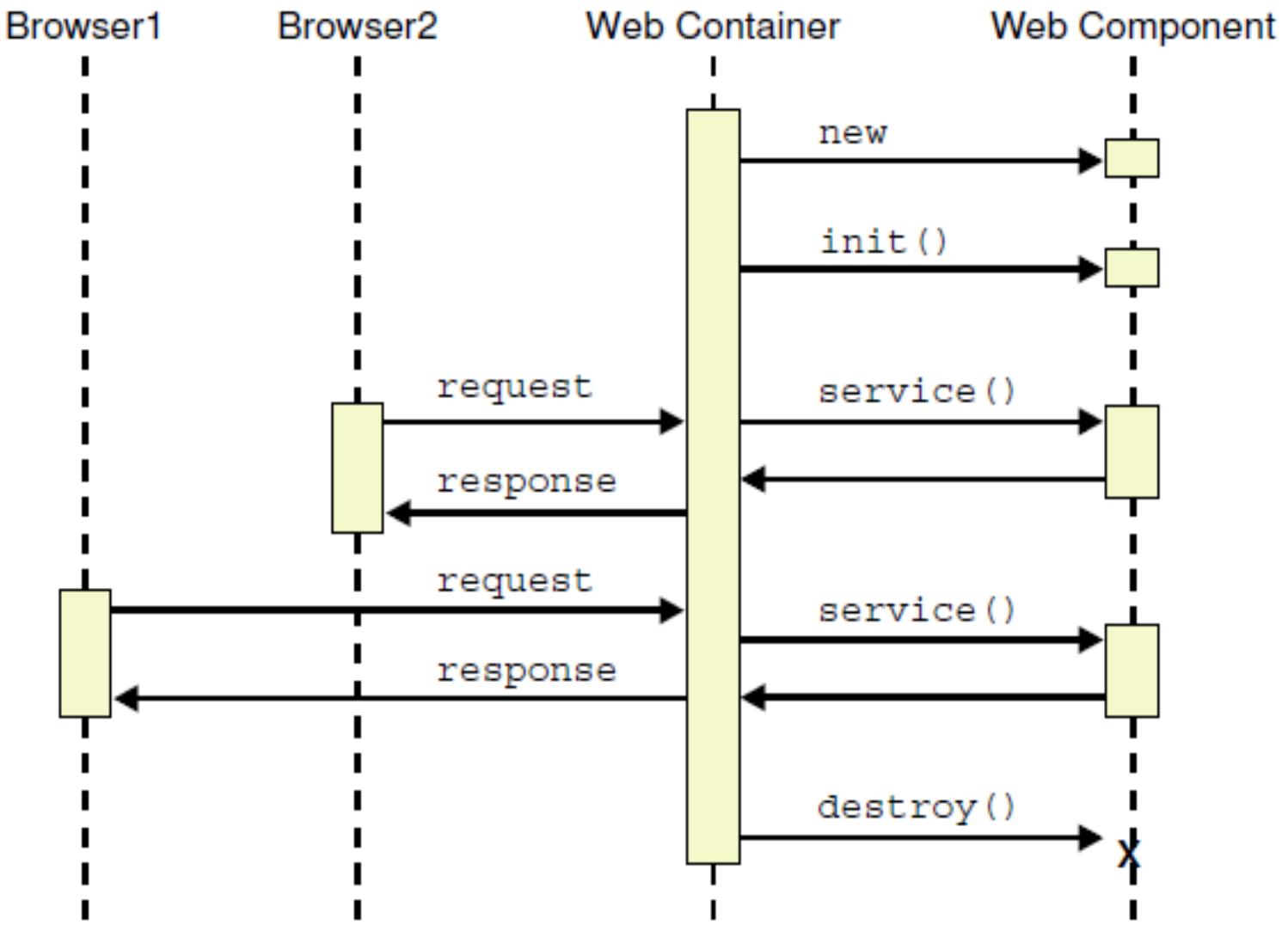

http://en.wikipedia.org/wiki/List_of_HTTP_header_fields



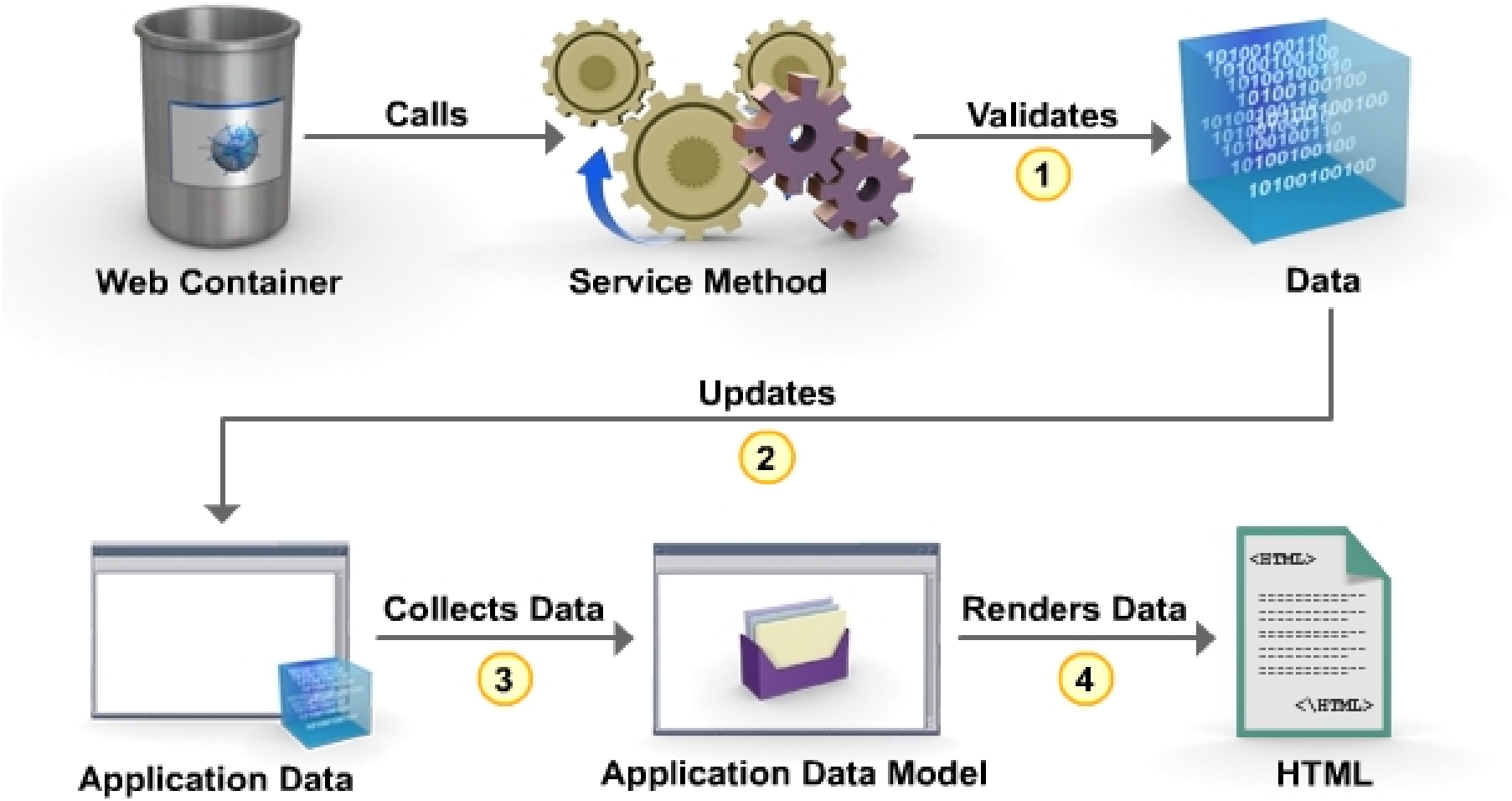

**Servlet is a Java class that can be loaded dynamically to expand the functionality of a server**

# Life Cycle of a Web Component



# The `service` Method

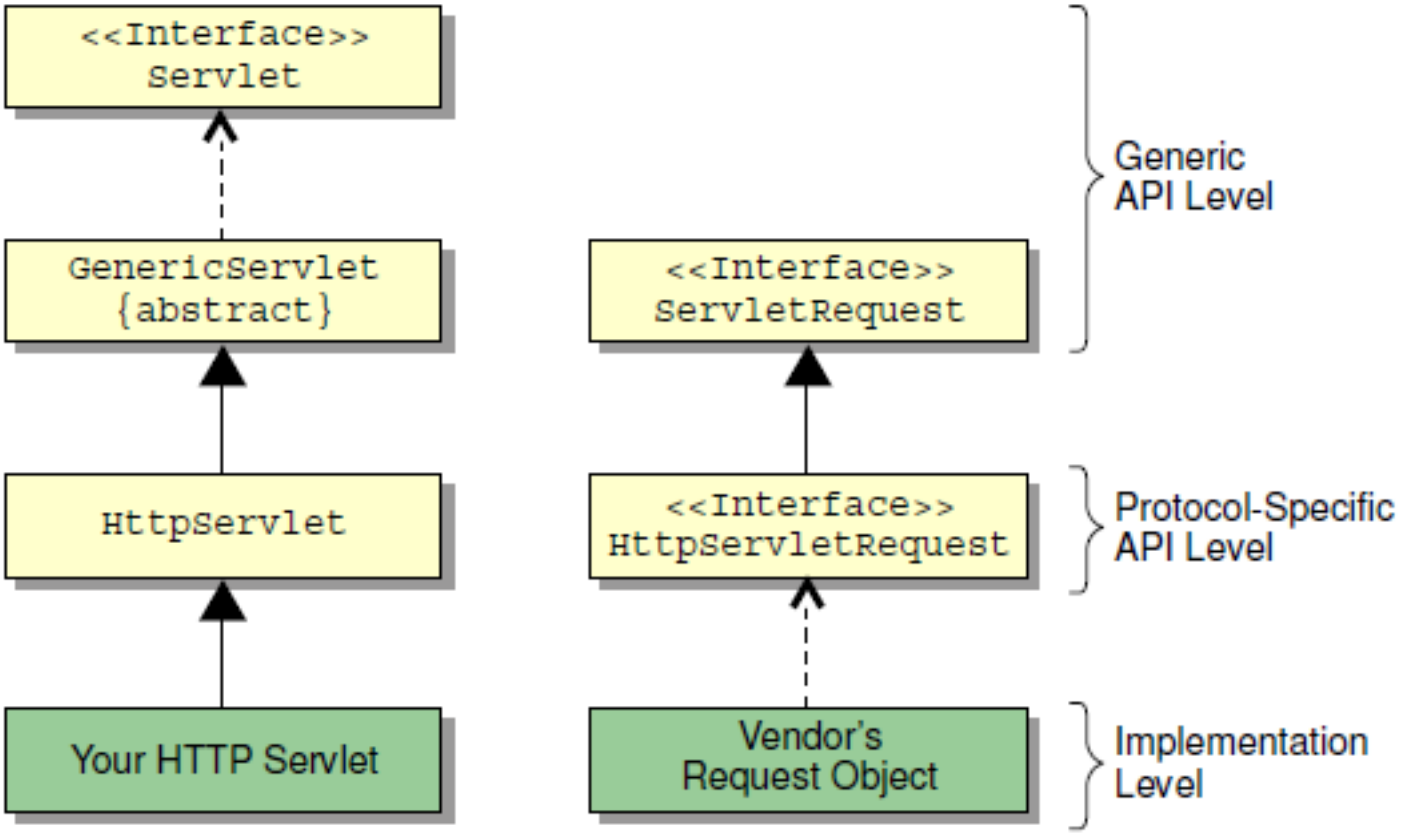
- The web container calls the service method once for each incoming request. The service method then typically completes the following operations:
  - Validates any form data
  - Updates the application's data model
  - Collects data from the model to be rendered
  - Renders the data in HTML or passes the request and the data to another component responsible for rendering them


Calls
Validates
1
Web Container
Service Method
Data
Updates
2
Collects Data
3
Renders Data
4
<HTML>
<\HTML>
Application Data
Application Data Model
HTML

# Basics of the Servlet API

- The servlet API provides the following facilities to servlets:
  - Callback methods for initialization and request processing
  - Methods by which the servlet can get configuration and environment information
  - Access to protocol-specific resources

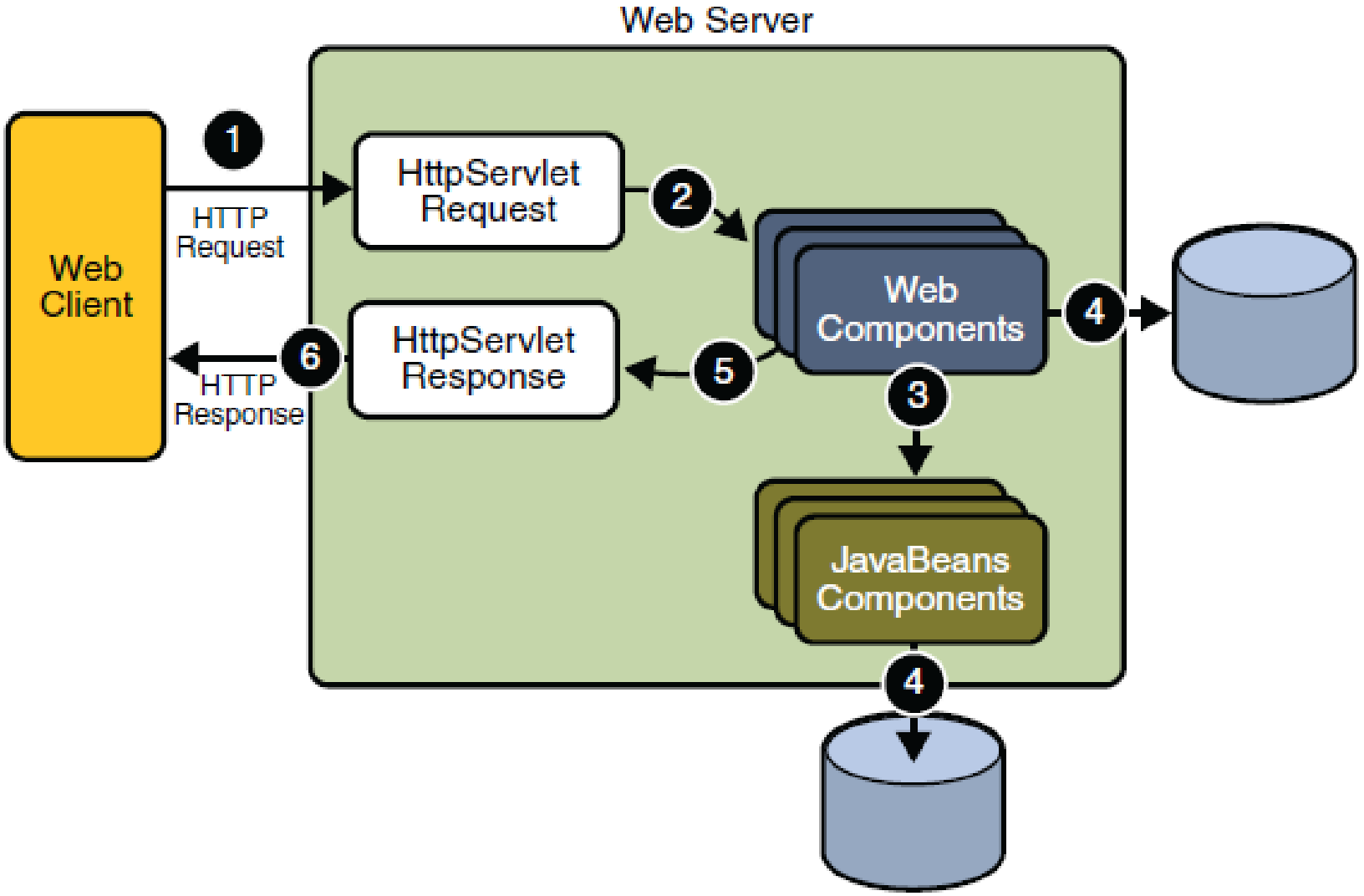
# Structure of the Servlet API



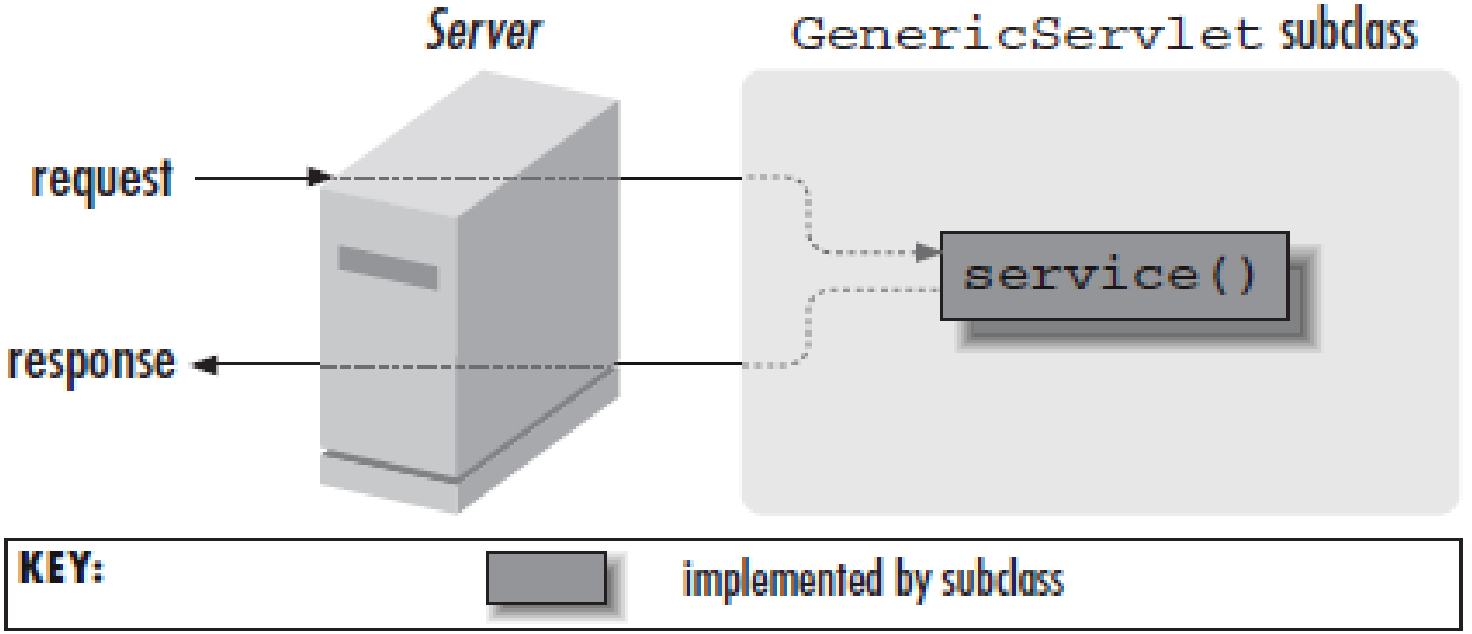
# Benefits of the HttpServlet Class

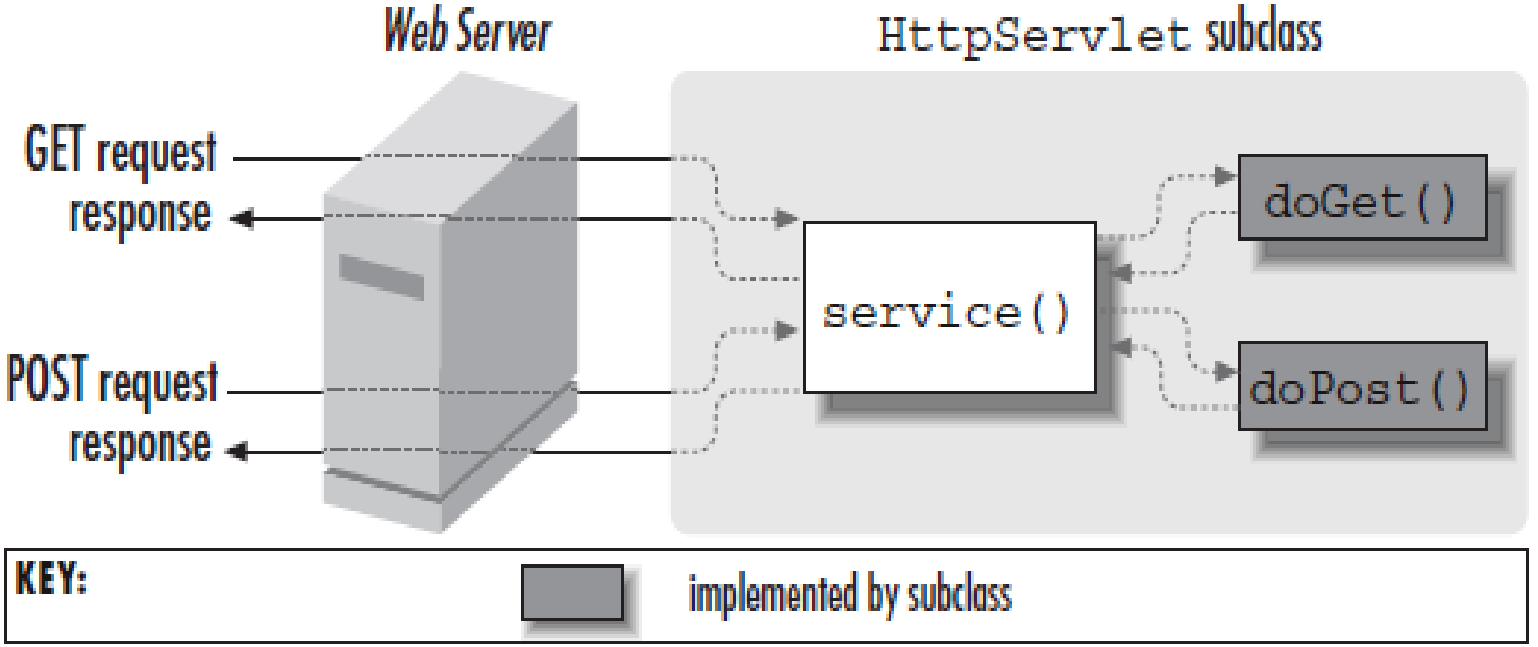
- Benefits of extending the HttpServlet base class include:
- A simplified, no-argument init method, which can be overridden to do initialization without the need to initialize the base class
- Standard handling of HTTP request types that are not of interest to the servlet
- Request handler arguments that are defined in terms of HTTP-specific **request and response objects**

# Java WebApplication RequestHandling

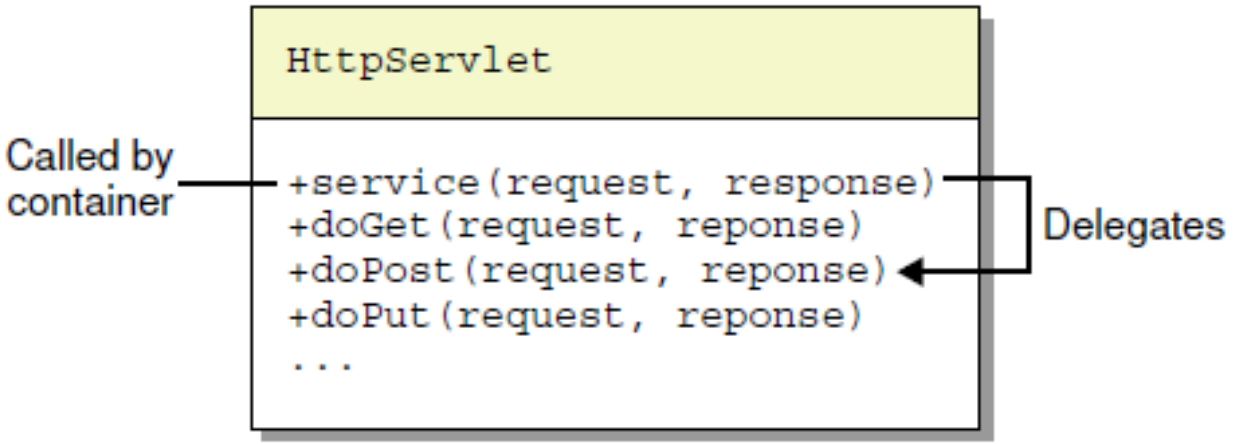



http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

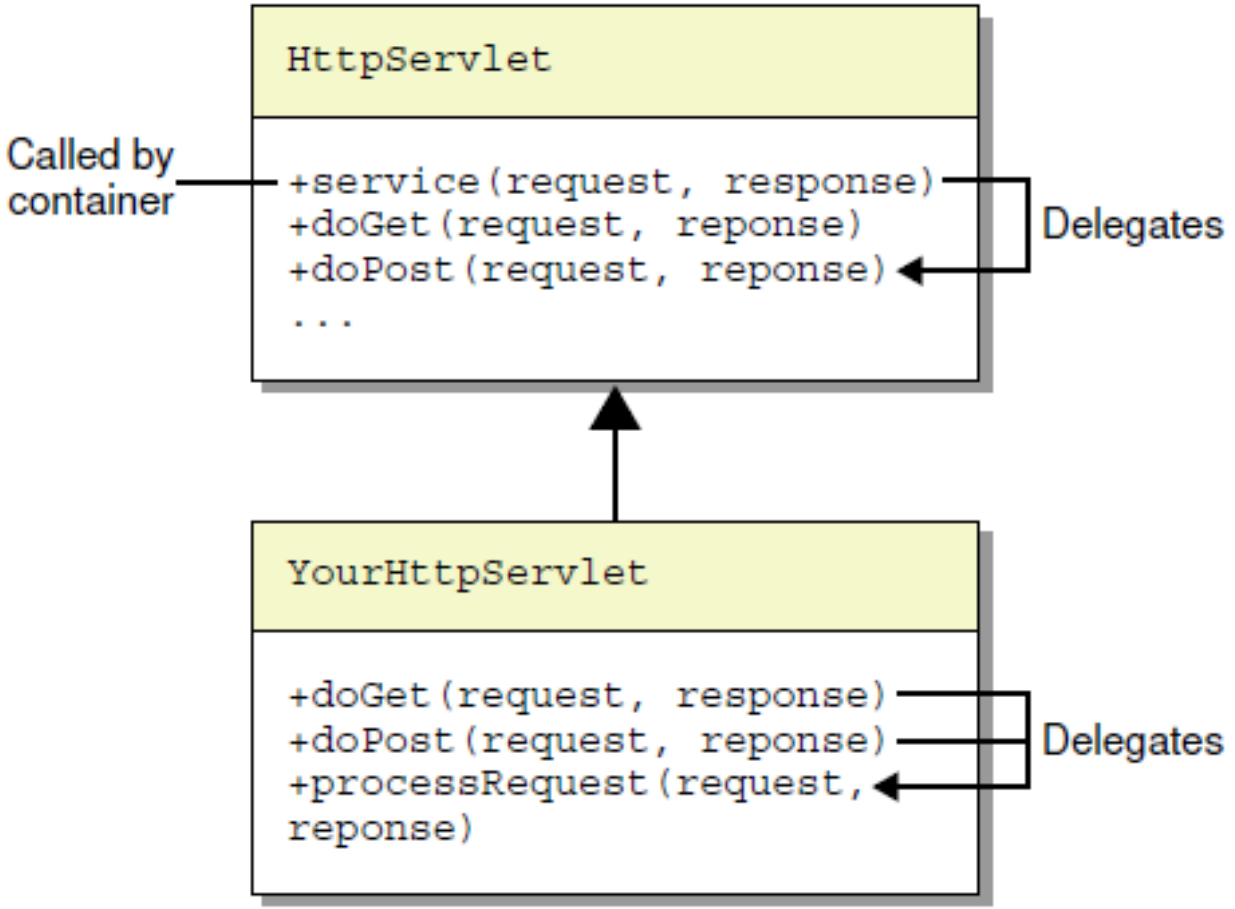

Server
GenericServlet subclass
request
response
service()
KEY:
implemented by subclass

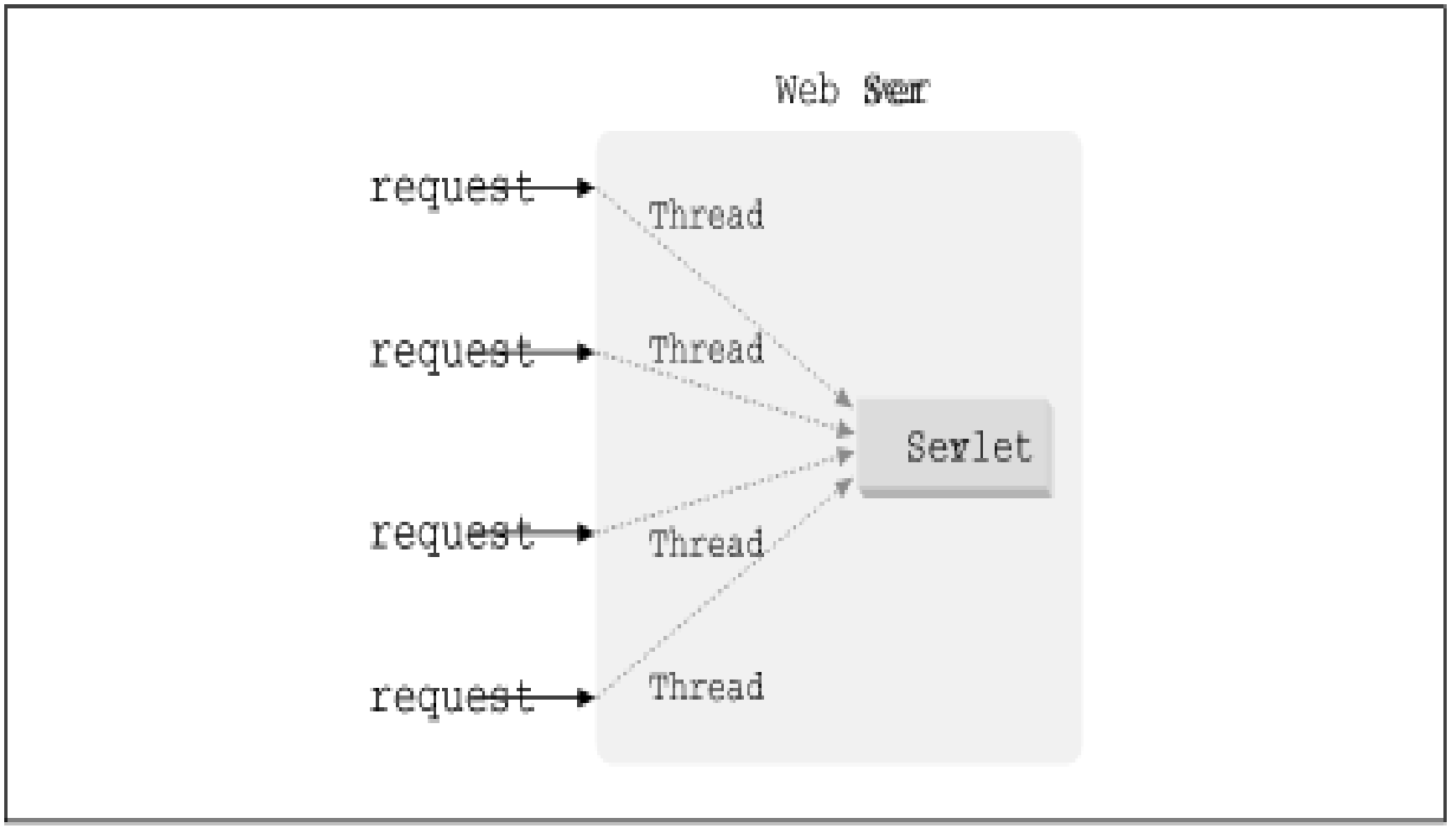

Web Server
HttpServlet subclass
GET request
response
POST request
response
service()
doGet()
doPost()
KEY:
implemented by subclass

# The `service` Method



# Request Handling Methods






If your servlets only read from the request, write to the response, and save information in local variables (that is, variables declared within a method), you needn't worry about the interaction among these threads. Once any information is saved in nonlocal variables (that is, variables declared within a class but outside any specific method), however, you must be aware that each of these client threads has the ability to manipulate a servlet's nonlocal variables. Without precautions, this may result in data corruption and inconsistencies.

# ServletConfig

```
public class SimpleInitServlet extends HttpServlet {

   protected String mydriver;
   protected String myurl;
   protected String myuserID;
   protected String mypassword;

public void init(ServletConfig config) throws ServletException {
   super.init(config);
   mydriver = config.getInitParameter("driver");
   myurl = config.getInitParameter("URL");
   myuserID = config.getInitParameter("userID");
   mypassword = config.getInitParameter("password");
}
```

Each servlet has an object associated with it called the ServletConfig. This object is created by the container and implements the javax.servlet.ServletConfig interface. It is the ServletConfig that contains the initialization parameters. A reference to this object can be retrieved by calling the getServletConfig() method.

# ServletContext

- The javax.servlet.ServletContext interface represents a Servlet's view of the Web Application it belongs to.
- Through the ServletContext  interface, a Servlet can access
  - raw input streams to Web Application resources,
  - virtual directory translation,
  - common mechanism for logging information,
  - application scope for binding objects.

# ServletContext

- public Object getAttribute(String name)
- public java.util.Enumeration getAttributeNames()
- public void setAttribute(String name, Object object)
- public void removeAttribute(String name)

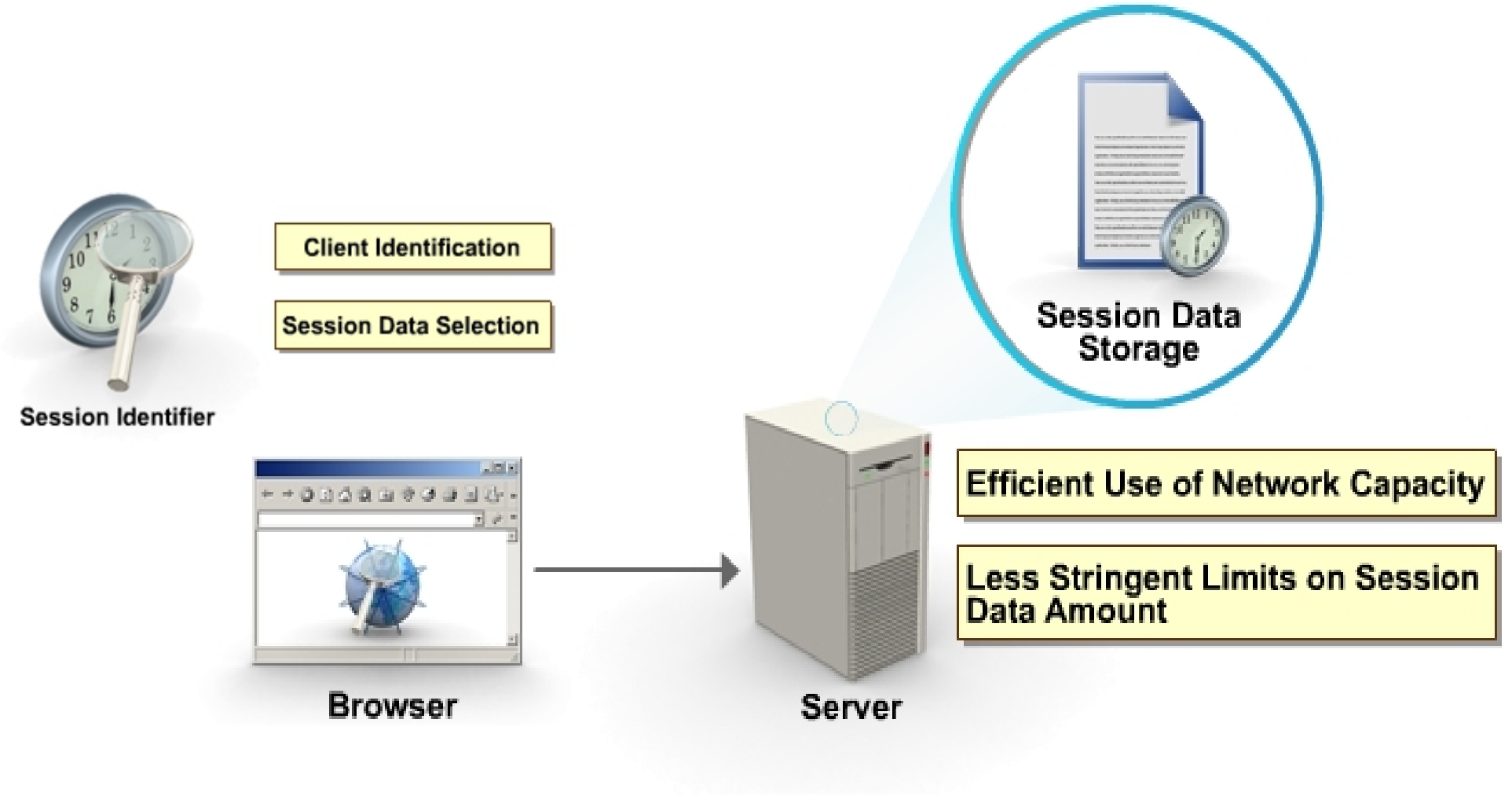
# Servlet Context and Application Scope

- A ServletContext instance allows for server-side objects to be placed in an application-wide scope. This type of scope is ideal for placing resources that need to be used by many different parts of a Web Application during any given time.
- An application scope should be used sparingly. Objects bound to a ServletContext object will not be garbage collected until the ServletContext is removed from use, usually when the Web Application is turned off or restarted.

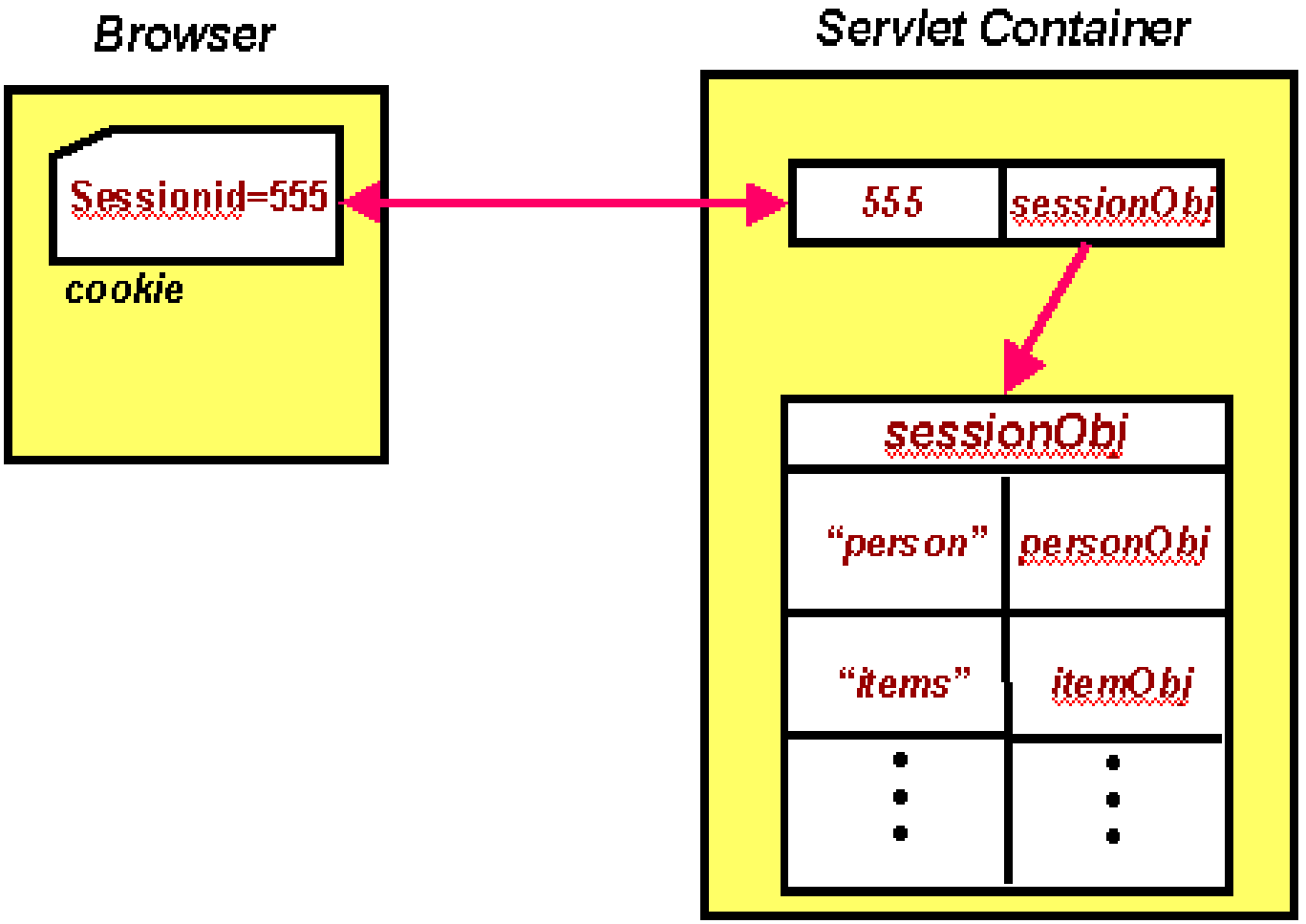
# Request Delegation and Request Scope

- A single client's request can pass through many Servlets and/or to any other resource in the Web Application.
- Request delegation is available through the javax.servlet.RequestDispatcher object
- HttpServletRequest object methods can be used to bind, access, and remove objects to and from the request scope that is shared by all Servlets to which a request is delegated

# Using Session Storage



# Session management

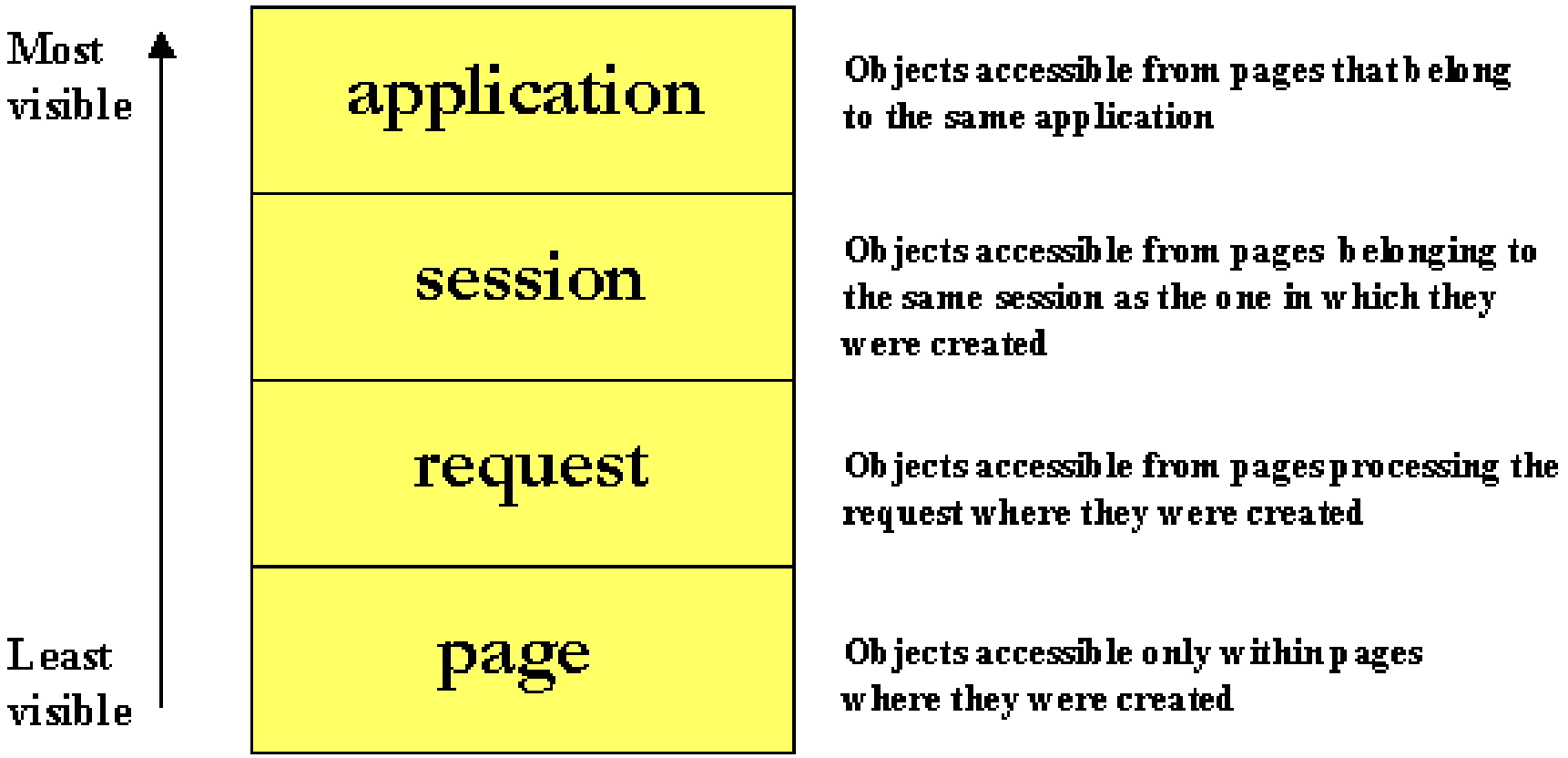



http://192.9.162.55/developer/onlineTraining/JSPIntro/contents.html

# Access scope

| Scope Object | Class | Accessible from |
|---|---|---|
| Web context | javax.servlet.ServletContext | Web components within a web context. |
| Session | javax.servlet.http.HttpSession | Web components handling a request that belongs to the session. |
| Request | Subtype of javax.servlet.ServletRequest | Web components handling the request. |
| Page | javax.servlet.jsp.JspContext | The JSP page that creates the object. |

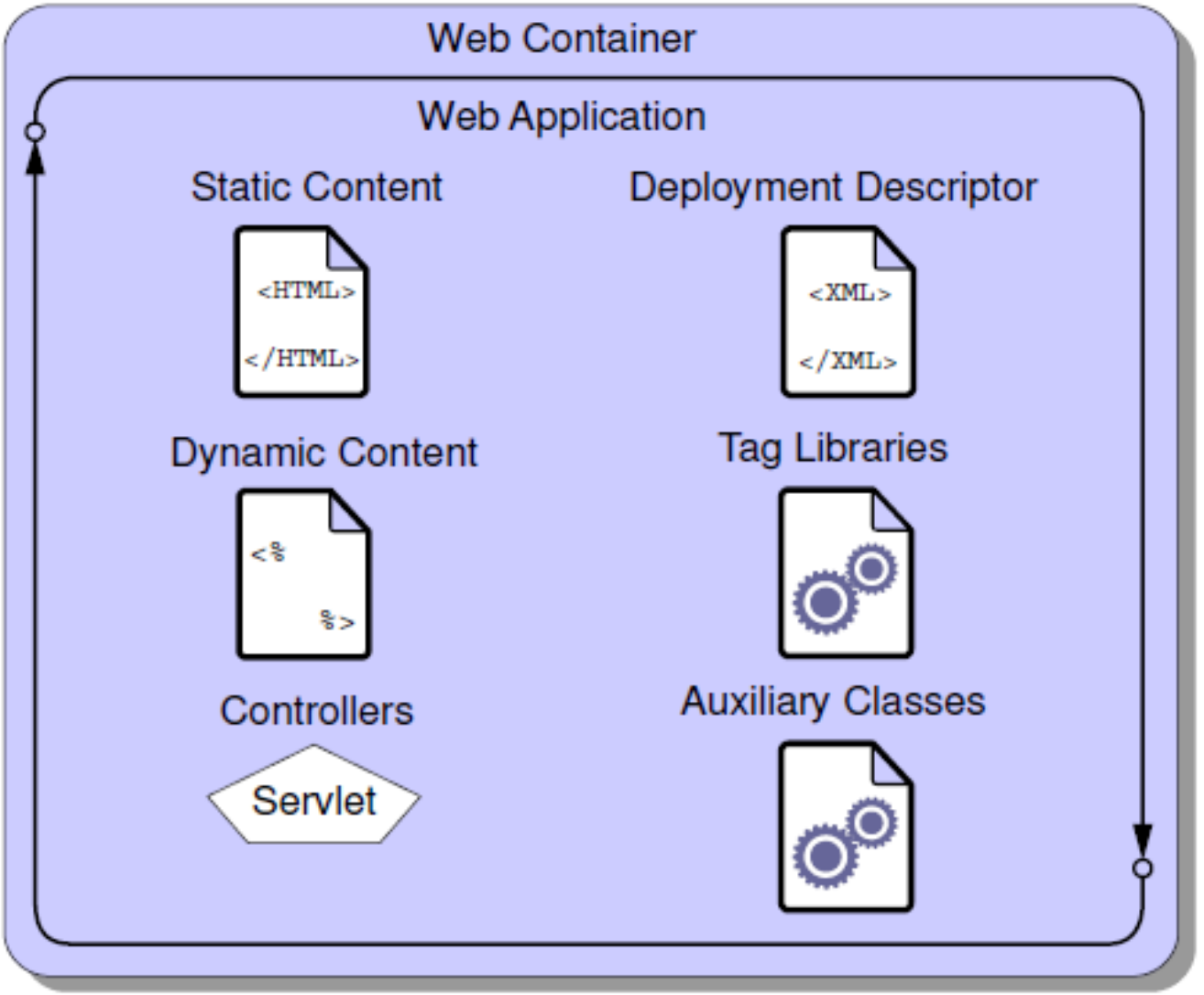
# Object Scopes



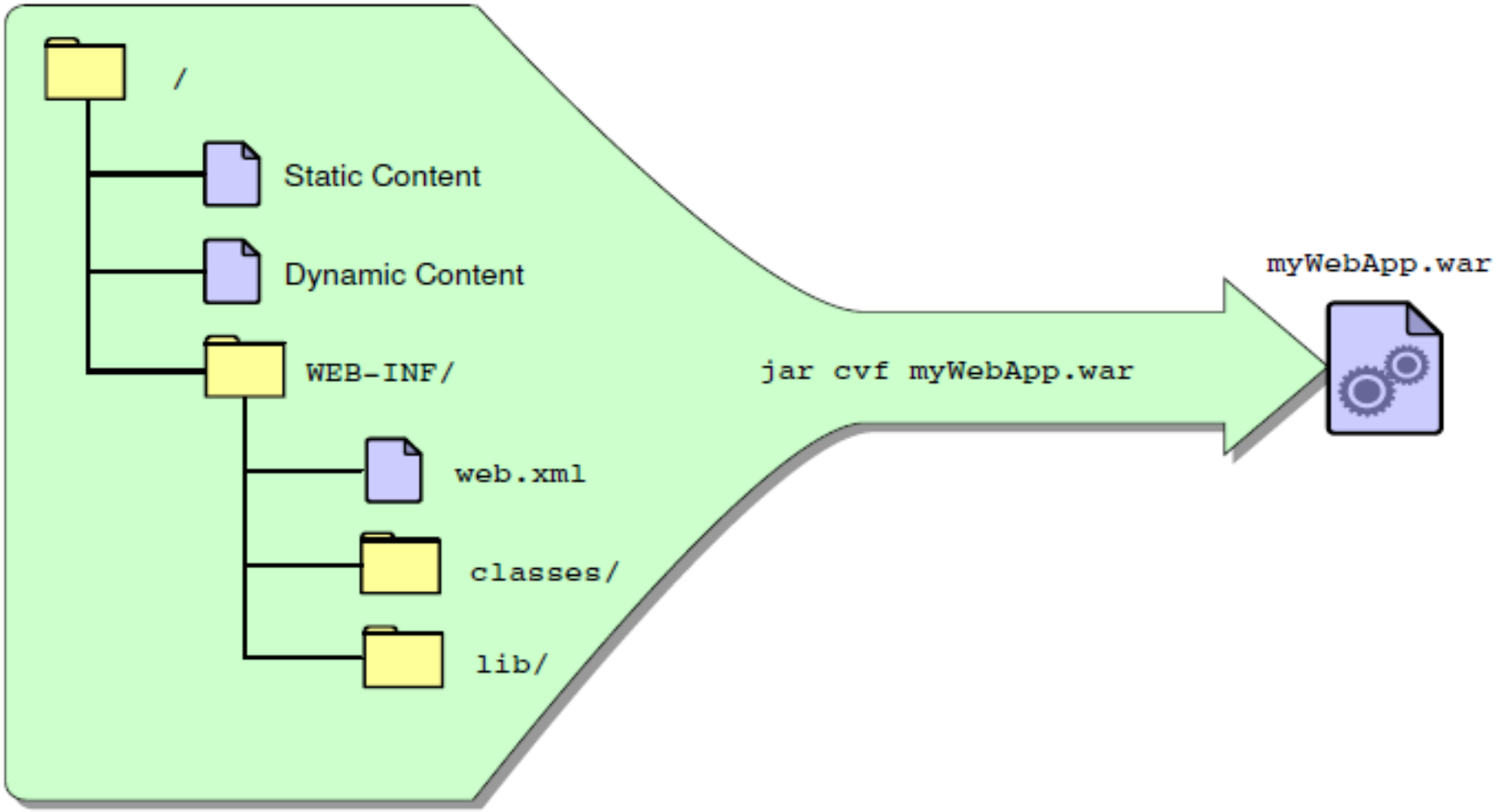
# Retrieving a Session Object

```
1   // Get a session object for the current client, creating
2   // a new session if necessary
3   HttpSession session = request.getSession();
4
5   // If this is a new session, initialize it
6   if (session.isNew()) {
7     // Initialize the session attributes
8     // to their start-of-session values
9     session.setAttribute ("account", new Account());
10    // ... other initialization
11  }
12
13  // Get this client's 'account' object
14  Account account = (Account) session.getAttribute("account");
```

# Web Application Elements



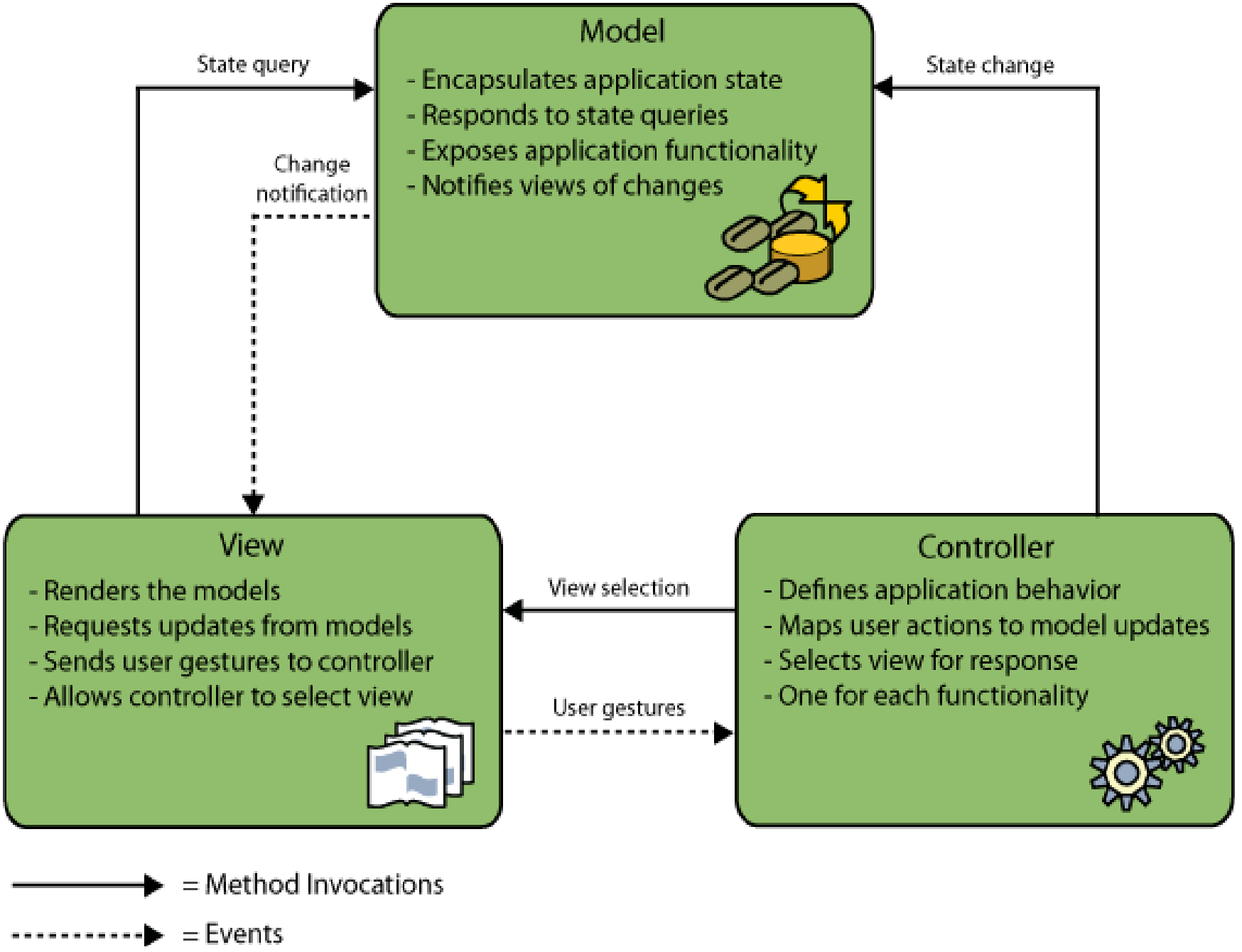
# Web Archive File Creation



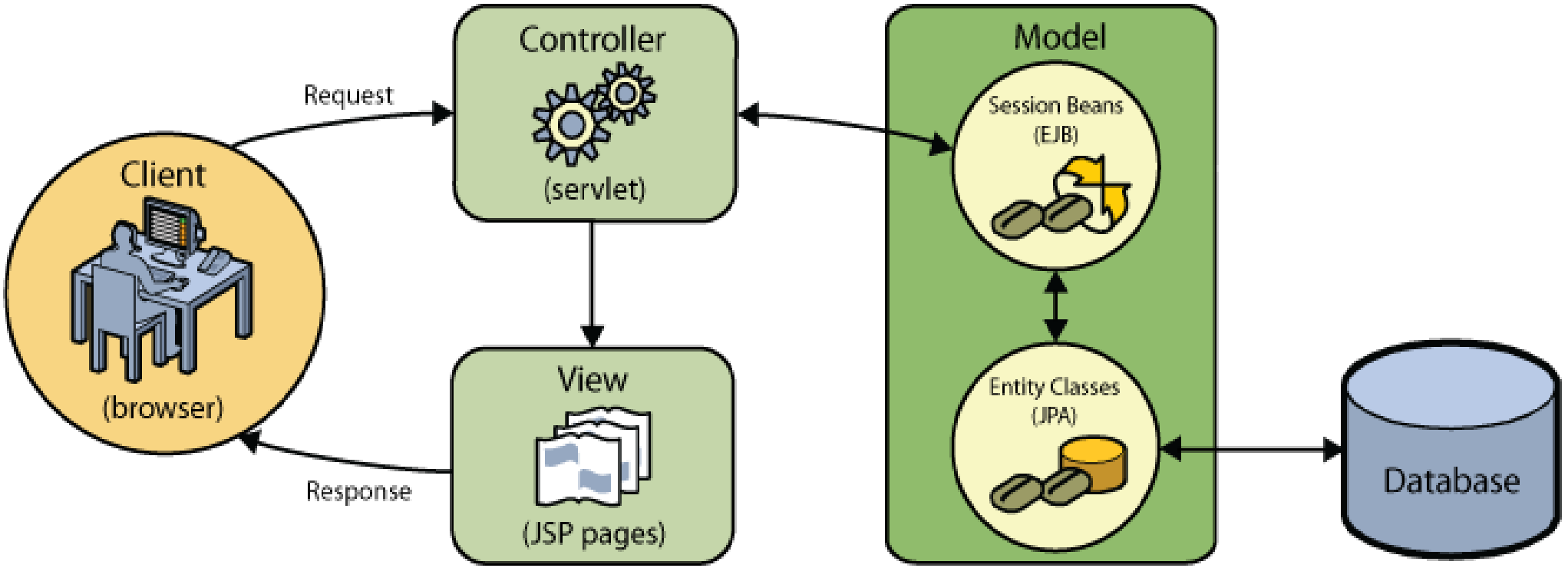
# Deployment Descriptors

- Are XML-formatted files
- Provide a declarative way to describe the interactions between components and between a component and its container
- Have their format, naming convention, and other attributes defined in the relevant component specification
- Are not always required. In-code annotations can be used by developers.

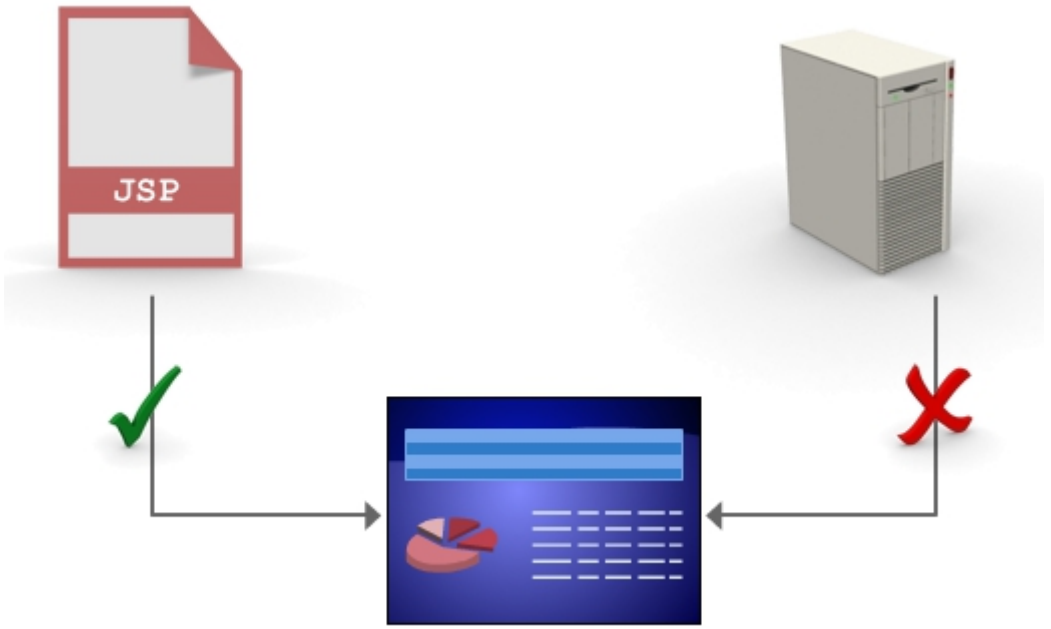
# MVC


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

# MVC in JavaEE



# JSP & Servlets




- JSP components are good for presentation.
- JSP components lose their benefits when they contain embedded Java programming language statement.
- Servlets are not ideal for generating presentation.