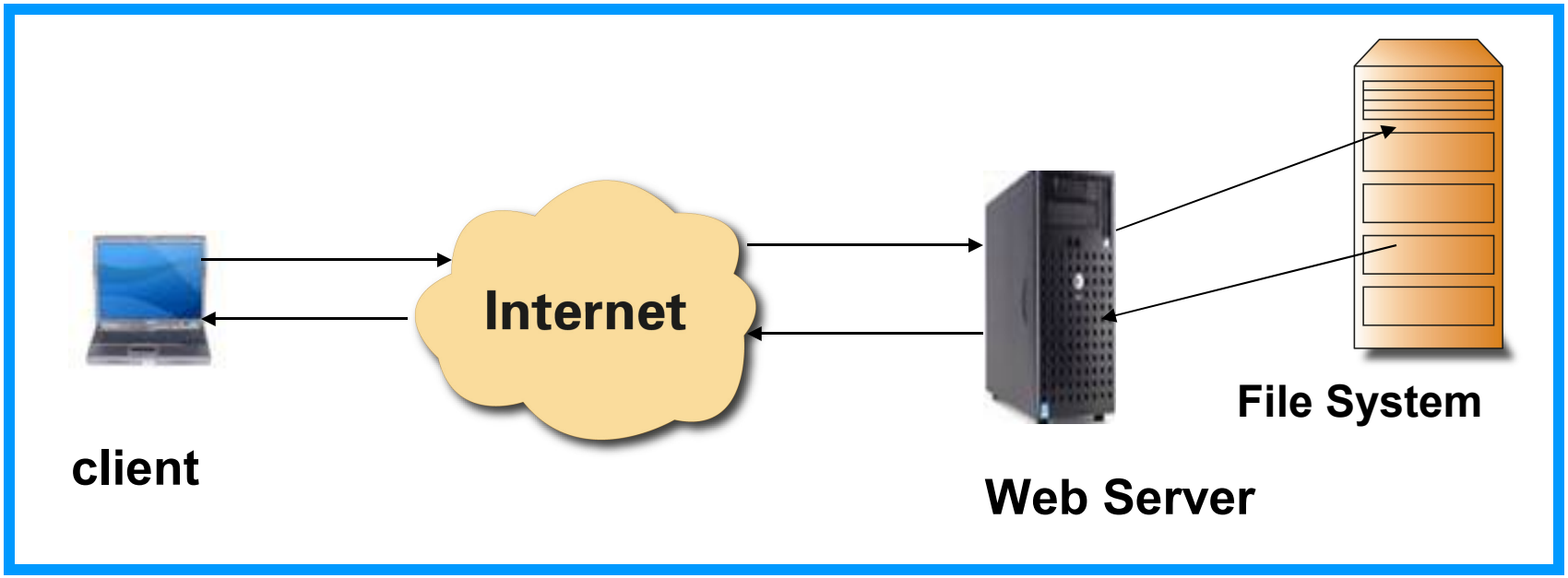
Basic Servlet

How Internet works?

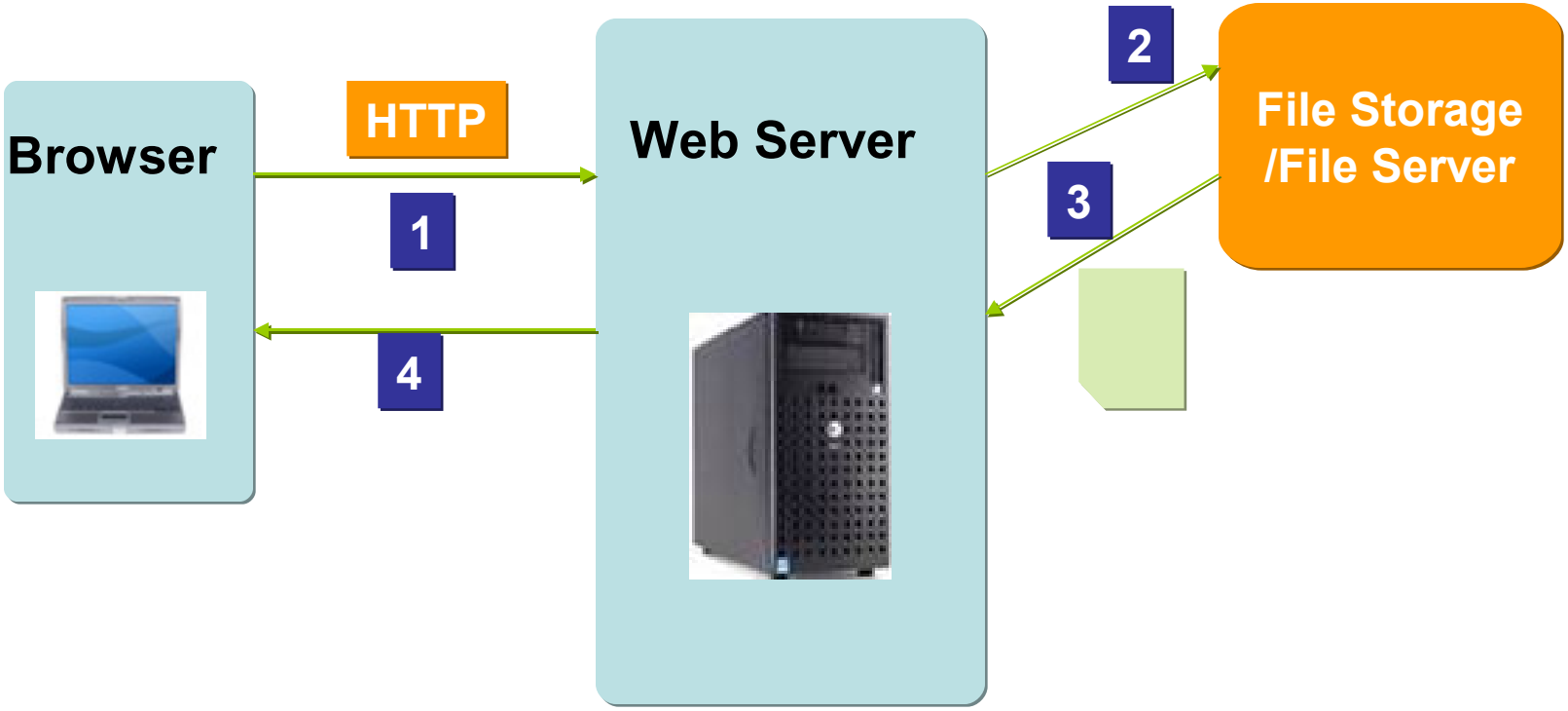
- A client makes an HTTP request through the web browser to a remote host (web site)
- A web server receives the request and sends the content (static content) through the response
- The client receives the response in its browser



Static Pages

- Static webpages
  - Plain files stored in the filesystem
  - Webserver accepts pathnames
  - Returns contents of corresponding file
- Boring!
  - Can’t generate customized content—always serve the same static pages…

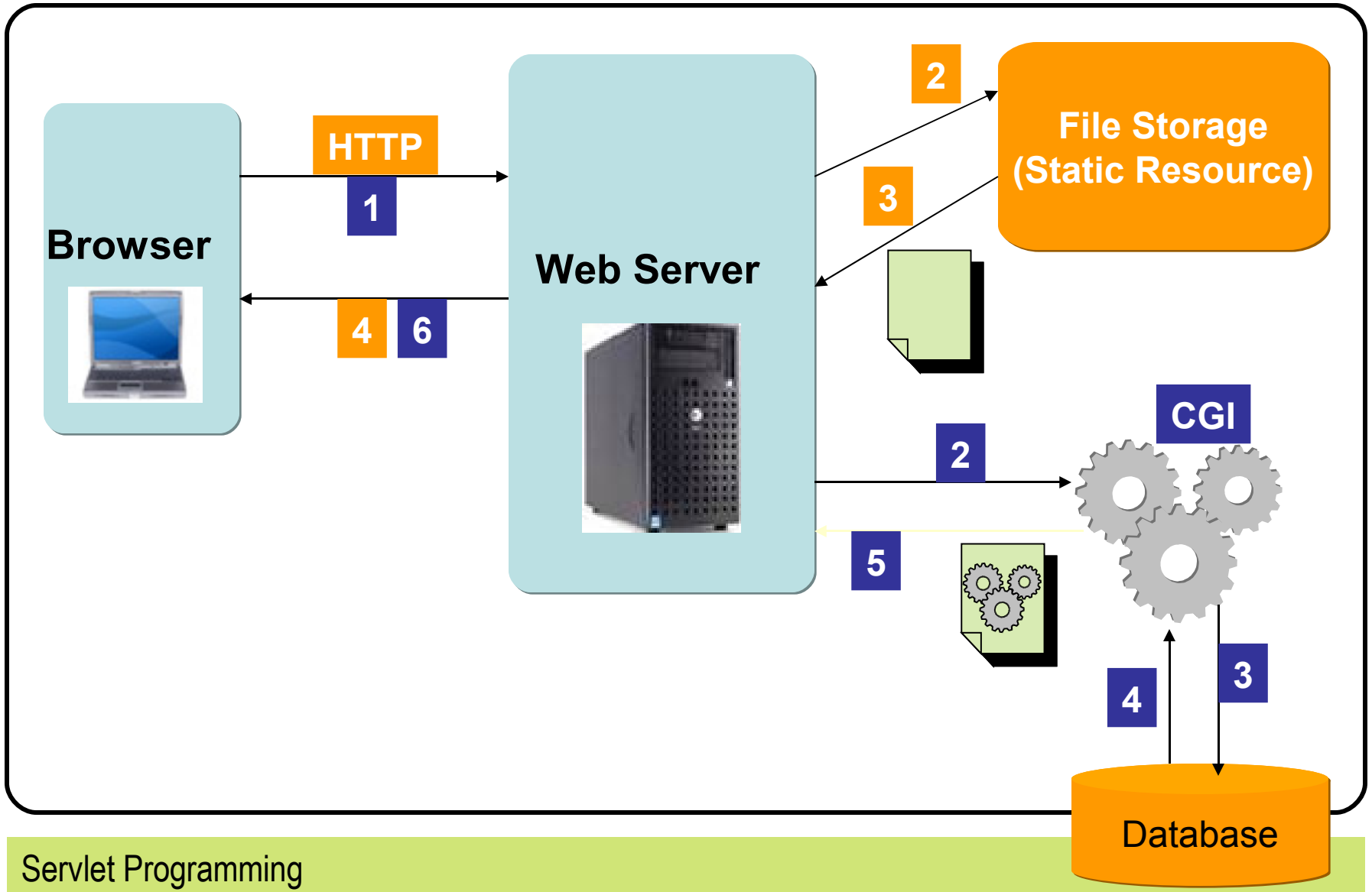
Static Pages



Dynamic Content

- CGI: Easy “fix”
  - Common Gateway Interface
  - Oldest standard
  - But at least a **standard**!
  - Inefficient
  - No persistent state
- Forward requests to external programs
  - Spawn one process for each new request (ouch!)
  - Communicate via
    - Standard input/output
    - Environment variables
  - Process terminates after request is handled

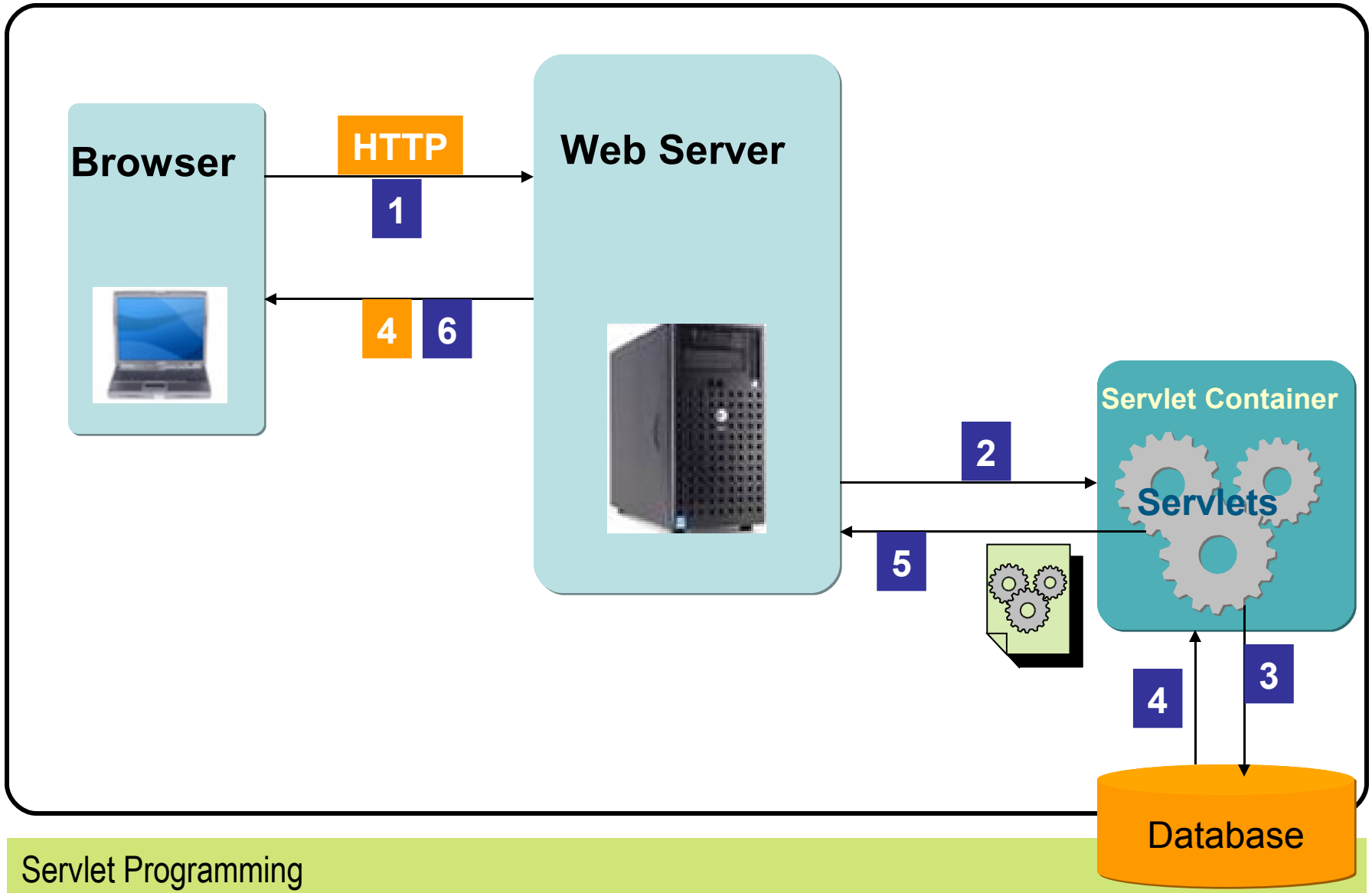
CGI


Browser
HTTP
1
4
6
Web Server
2
3
File Storage
(Static Resource)
CGI
2
5
4
3
Database

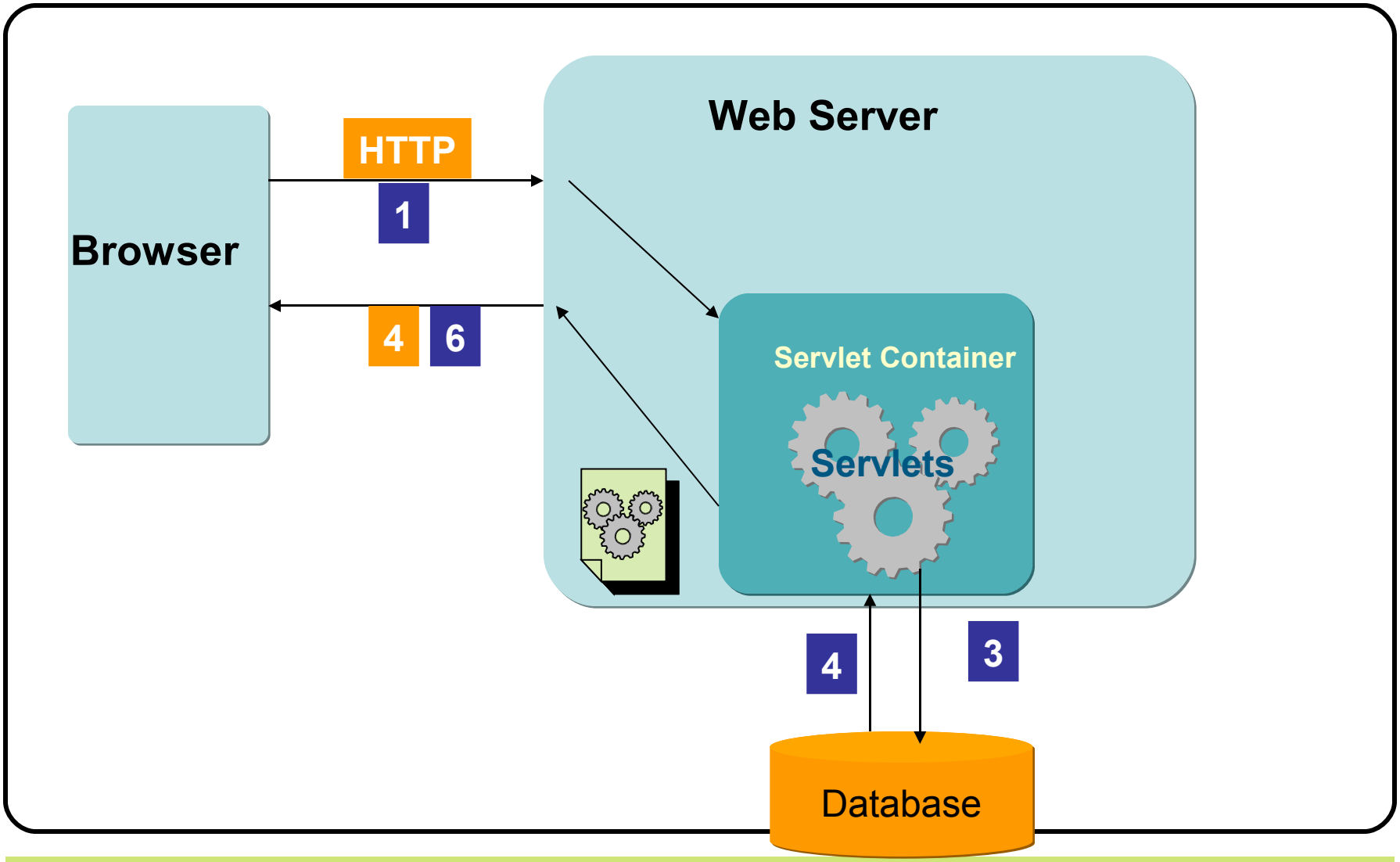
Servlets to the rescue…

- Little Java programs…
  - Contain application-specific code
  - Web server does generic part of request handling
  - Servlets run “in” the web server and do some of the handling
- Highlights
  - Standard!
  - Efficiency (much better than CGI)
  - Security (Java!)
  - Persistence (handle multiple requests)

Servlets


Browser
HTTP
1
Web Server
4
6
2
Servlet Container
Servlets
5
4
3
Database

Servlets


Web Server
HTTP
1
Browser
4
6
Servlet Container
Servlets
4
3
Database

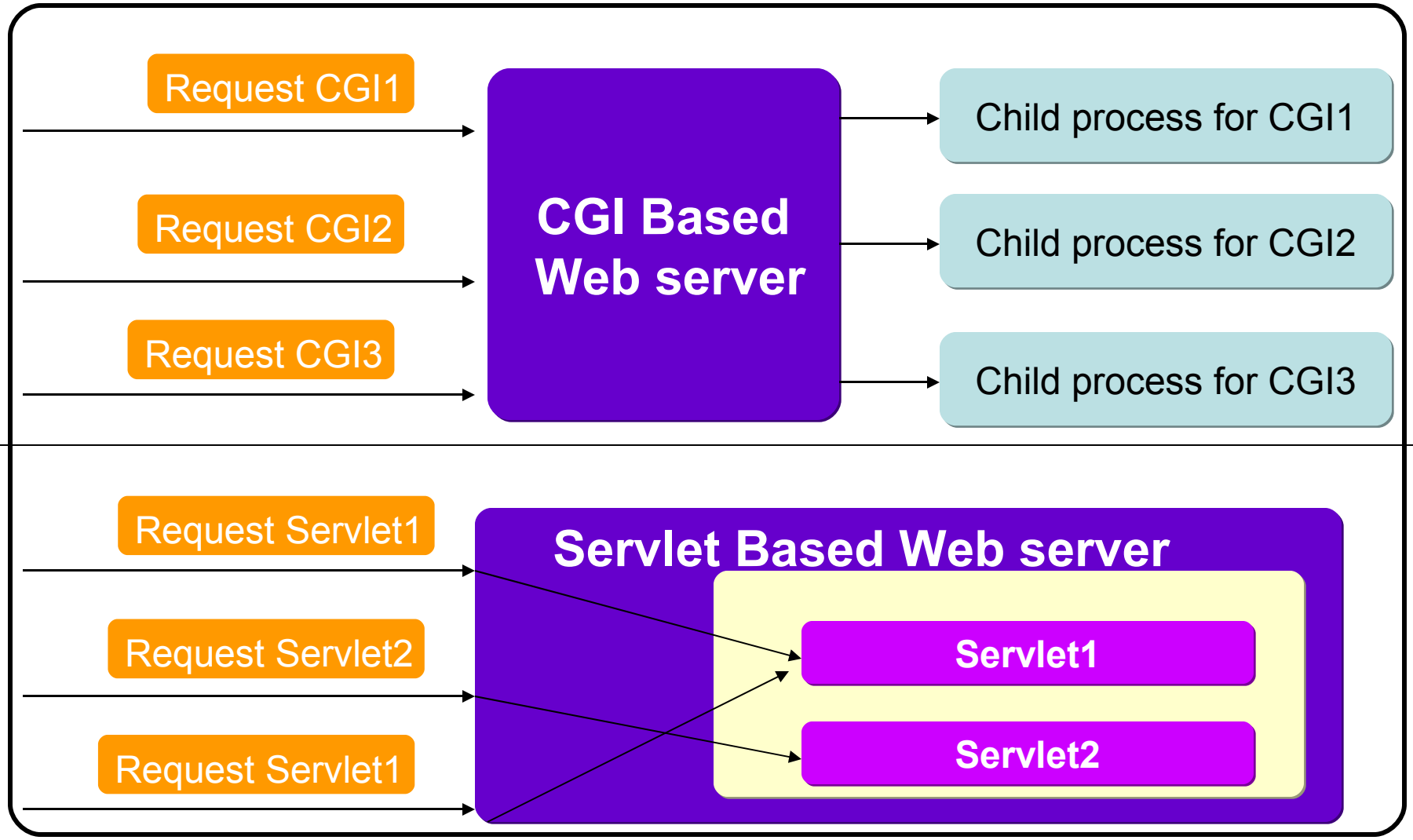
Comparison with CGI

- Servlets are much more efficient than CGI scripts.
- Once a servlet is loaded it can efficiently respond to many requests. Each is a separate thread (lightweight process)
- In CGI each request causes a separate external process to be started in the operating system.

Comparison with CGI

- Servlets have built-in object-oriented functionality for dealing with the HTTP request and the associated list of name-value pairs resulting from a form or url submission
- They have support for cookies
- They have support for session management
- They are essentially part of the Web server rather than separate processes.

CGI Versus Servlet



Advantages of Servlet

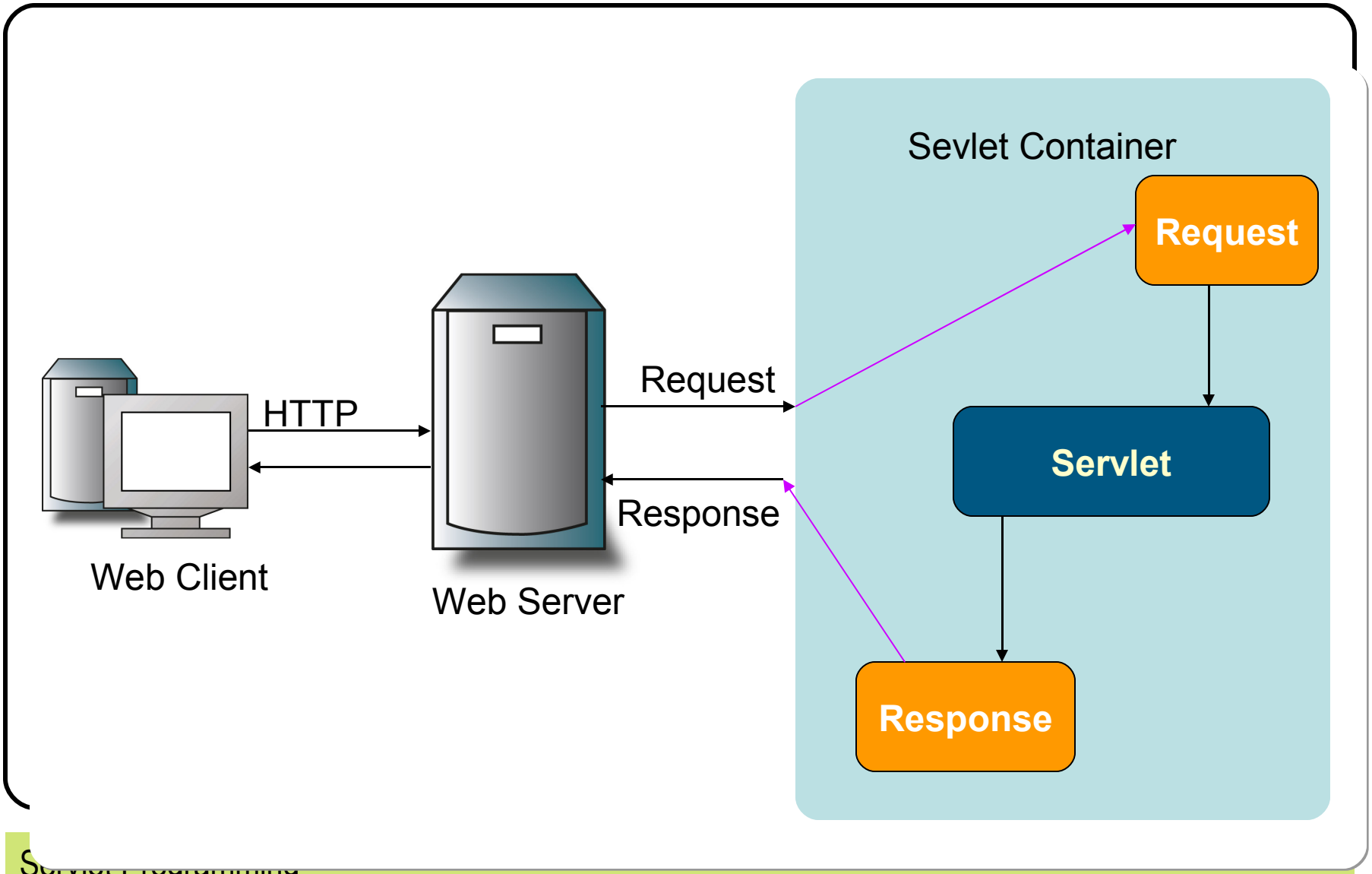
- No CGI limitations
- Abundant third-party tools and Web servers supporting Servlet
- Access to entire family of Java APIs
- Reliable, better performance and scalability
- Platform and server independent
- Secure
- Most servers allow automatic reloading of Servlet's by administrative action

What is Servlet?

- Java™ objects which are based on servlet framework and APIs and extend the functionality of a HTTP server.
- Mapped to URLs and managed by container with a simple architecture
- Available and running on all major web servers and app servers
- Platform and server independent

Servlet Request & Response Model

Servlet Request and Response Model



What does Servlet Do?

- Receives client request (mostly in the form of HTTP request)
- Extract some information from the request
- Do content generation or business logic process (possibly by accessing database, invoking EJBs, etc)
- Create and send response to client (mostly in the form of HTTP response) or forward the request to another servlet or JSP page

Requests and Responses

- What is a request?
  - Information that is sent from client to a server
    - Which client made the request
    - user data sent to the servlet/server
    - Http Headers sent to the server
- What is a response?
  - Information that is sent to client from a server
    - Text(html, plain) or binary(image) data
    - HTTP headers, cookies, etc

HTTP Request

- HTTP request contains
  - header
  - a method
    - Get: Input form data is passed as part of URL
    - Post: Input form data is passed within message body
    - Put
    - Header
  - request data

HTTP GET and POST

- The most common client requests
  - HTTP GET & HTTP POST
- GET requests:
  - User entered information is appended to the URL in a query string
  - Can only send limited amount of data
    - ../servlet/ViewCourse?FirstName=Shan&LastName=Ban
- POST requests:
  - User entered information is sent as data (not appended to URL)
  - Can send any amount of data

Sample HTTP Request

To retrieve the file at the URL

http://www.somehost.com/path/file.html

```
GET /path/file.html HTTP/1.0
From: someuser@somehost.com
User-Agent: HTTPTool/1.0

[blank line here]
```

Sample HTTP Response

- The server should respond with something like the following, sent back

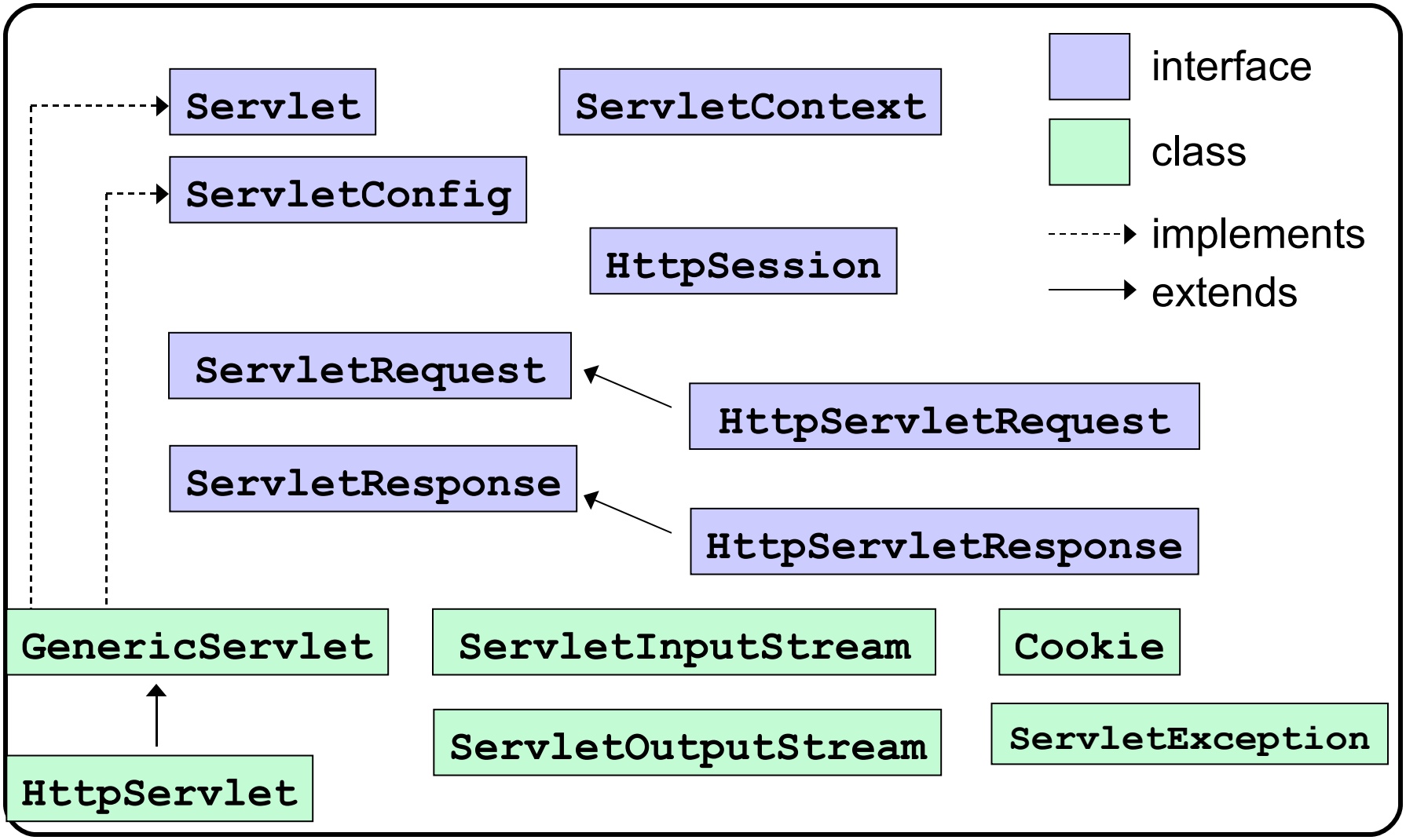
```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354
<html>
<body><h1>Happy New Millennium!</h1>
(more file contents) . . .
</body>
</html>
```

HTTP response Status Codes

1. 100-199 Informational
2. 200-299 Successful
   - 200: OK
   - 202: Accepted
   - 204: No Content
1. 300-399 Redirection
2. 400-499 Incomplete
   - 400: Bad Request
   - 401: Unauthorized
   - 403: Forbidden
   - 404: Not Found
1. 500-599 Server Error
   - 500: Internal Server Error
   - 501: Not Implemented

Interfaces & Classes of Servlet

Servlet Interfaces and Classes



Creating a servlet

- All the servlets must implement `javax.servlet.Servlet` interface
  - This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server.
  - These are known as life-cycle methods
- we can write a generic servlet that extends `javax.servlet.GenericServlet`
  - GenericServlet class implements Servlet interface
- Or an HTTP servlet that extends `javax.servlet.http.HttpServlet`
  - HttpServlet class extends GenericServlet class

Generic Servlet

- GenericServlet implements the `Servlet` and `ServletConfig` interfaces.
- `GenericServlet` may be directly extended by a servlet
- It provides simple versions of the lifecycle methods `init` and `destroy` and of the methods in the `ServletConfig` interface
- To write a generic servlet, we need only override the `abstract service` method

GenericServlet : an Example

```
package example;
import java.io.*;
import javax.servlet.*;

public class HelloWorld extends GenericServlet {
 public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException {
                    PrintWriter out = response.getWriter();
                   out.println("Hello World");
            }
}
```

HttpServlet

1. To create an HTTP servlet suitable for a Web site ,the servlet must extend HttpServlet class.
2. A subclass of `HttpServlet` must override at least one method, usually one of these:
    1. doGet, if the servlet supports HTTP GET requests
    2. doPost, for HTTP POST requests
    3. doPut, for HTTP PUT requests
    4. doDelete, for HTTP DELETE requests
    5. init and destroy, to manage resources that are held for the life of the servlet
    6. getServletInfo, which the servlet uses to provide

1. `service()` handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the do*XXX*)
1. There's almost no reason to override the service method

HttpServlet : an Example

```
package example;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {


                        PrintWriter out = response.getWriter();
     out.println("Hello World");
            }
 }
```

A Servlet that Generates HTML

```
package hall;
 import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;

 public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("<HTML>\n" + "<HEAD><TITLE>Hello                WWW</TITLE></
      HEAD>\n" + "<BODY>\n“
                 + "<H1>Hello World</H1>\n" + "</BODY></HTML>");
                }
}
```

Servlet Life Cycle

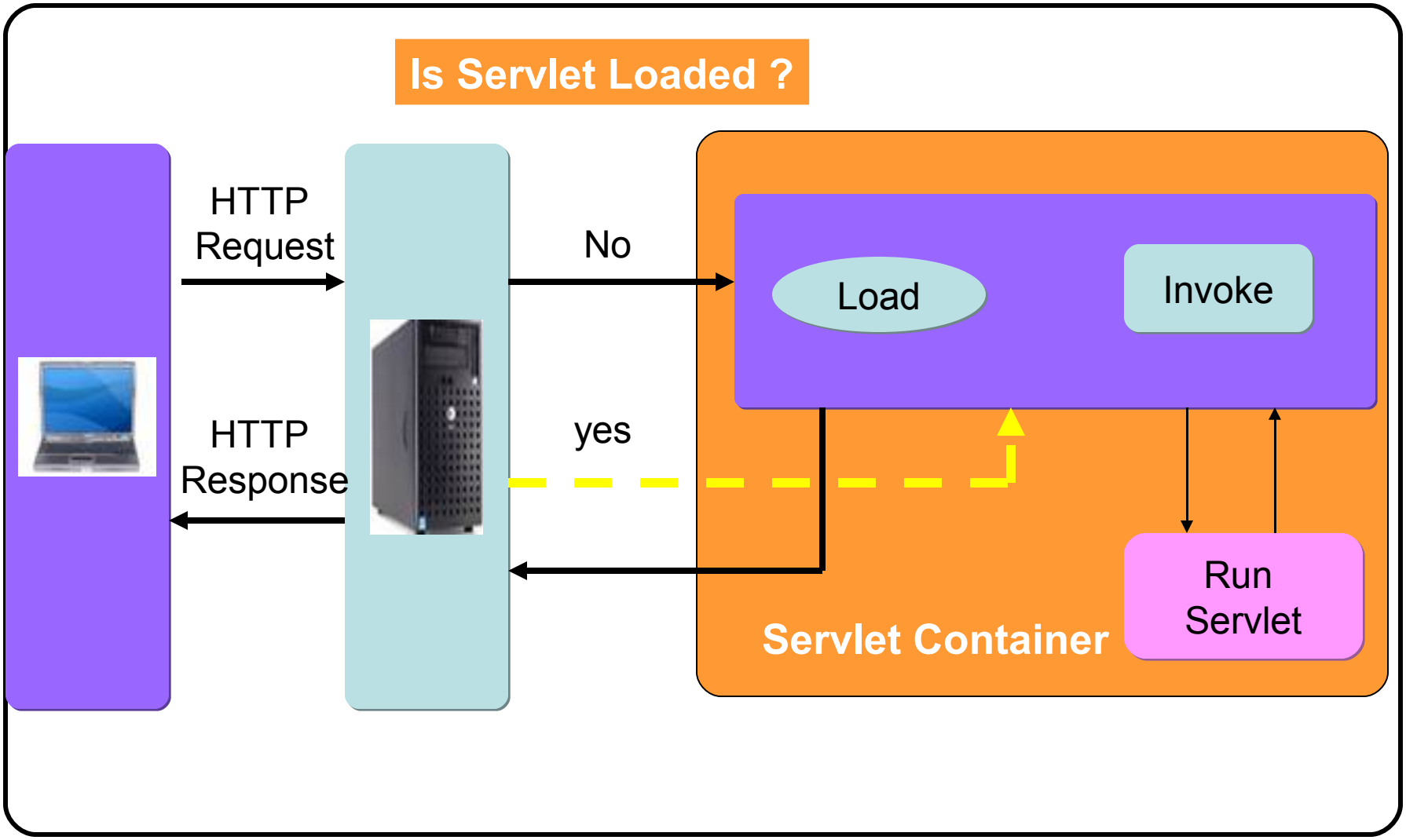
Servlet Life Cycle Methods

- Invoked by container
  - Container controls life cycle of a servlet
- Defined in
  - javax.servlet.GenericServlet class or
    - init()
    - destroy()
    - service() - this is an abstract method
  - javax.servlet.http.HttpServlet class
    - doGet(), doPost(), doXxx()
    - service() - implementation

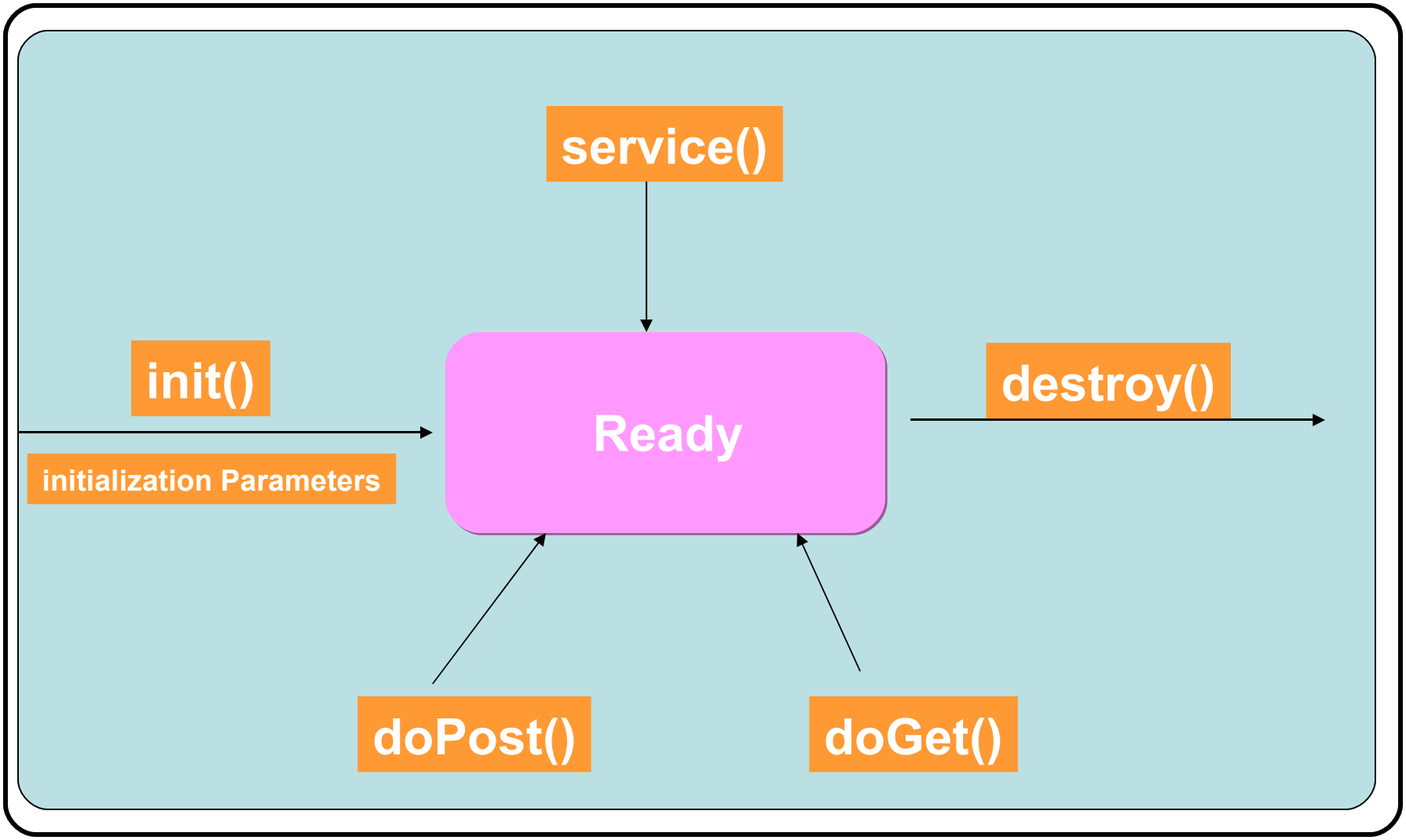
Servlet Life Cycle Methods

- init()
  - Invoked once when the servlet is first instantiated
  - Perform any set-up in this method
    - Setting up a database connection
- destroy()
  - Invoked before servlet instance is removed
  - Perform any clean-up
    - Closing a previously created database connection

Servlet Life-Cycle



Servlet Life-Cycle methods



Example: init() reading (1)

```
package com.example;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ConfigDemoServlet extends HttpServlet {

        ServletConfig config=null;

        public void init(ServletConfig config) throws
          ServletException {
        super.init();
        this.config=config;
    }
```

Example: init() reading (2)

```
public void doGet(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException, IOException {

                    resp.setContentType( "text/html");
                    PrintWriter out=resp.getWriter();
                    String driver = config.getInitParameter("driver");
                    String url  = config.getInitParameter("url");

                    out.println("<h2>parameter Name : "+"driver</h2>");
                    out.println("<h2>parameter Value : "+param+"</h2>");
                    out.println("<h2>parameter Name : "+"URL</h2>");
                    out.println("<h2>parameter Value : "+url+"</h2>");

    }
}
```

Setting Init Parameters in web.xml

```
<web-app>
    <servlet>
        <servlet-name>config</servlet-name>
        <servlet-class>com.example. ConfigDemoServlet </servlet-class>
            <init-param>
                <param-name>driver</param-name>
                <param-value>
                    oracle.jdbc.driver.OracleDriver
                </param-value>
            </init-param>
            <init-param>
                <param-name>url</param-name>
                <param-value>
                    jdbc:oracle:thin:@localhost:1521:orcl
                </param-value>
            </init-param>
    </servlet>
</web-app>
```

Example: destory()

```
public class CatalogServlet extends HttpServlet {
    private BookDB bookDB;
        public void init() throws ServletException {
        bookDB = (BookDB)getServletContext().
        getAttribute("bookDB");
        if (bookDB == null) throw new
                UnavailableException("Couldn't get database.");
        }

        public void destroy() {
        bookDB = null;
        }
        ...
}
```

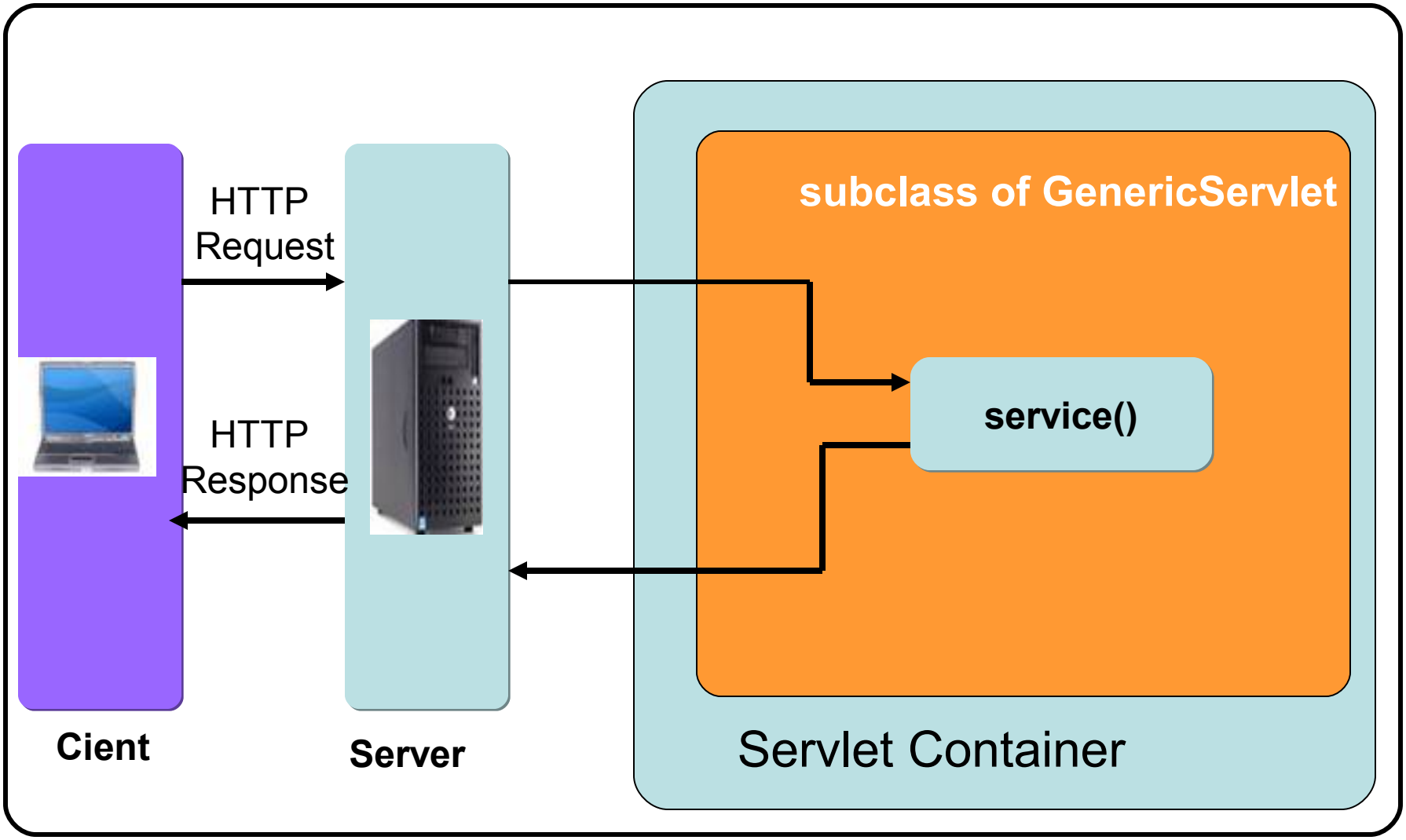
Servlet Life Cycle Methods

- service() javax.servlet.GenericServlet class
  - Abstract method
- service() in javax.servlet.http.HttpServlet class
  - Concrete method (implementation)
  - Dispatches to doGet(), doPost(), etc
  - Do not override this method!
- doGet(), doPost(), doXxx() is in javax.servlet.http.HttpServlet
  - Handles HTTP GET, POST, etc. requests
  - Override these methods in your servlet to provide desired behavior

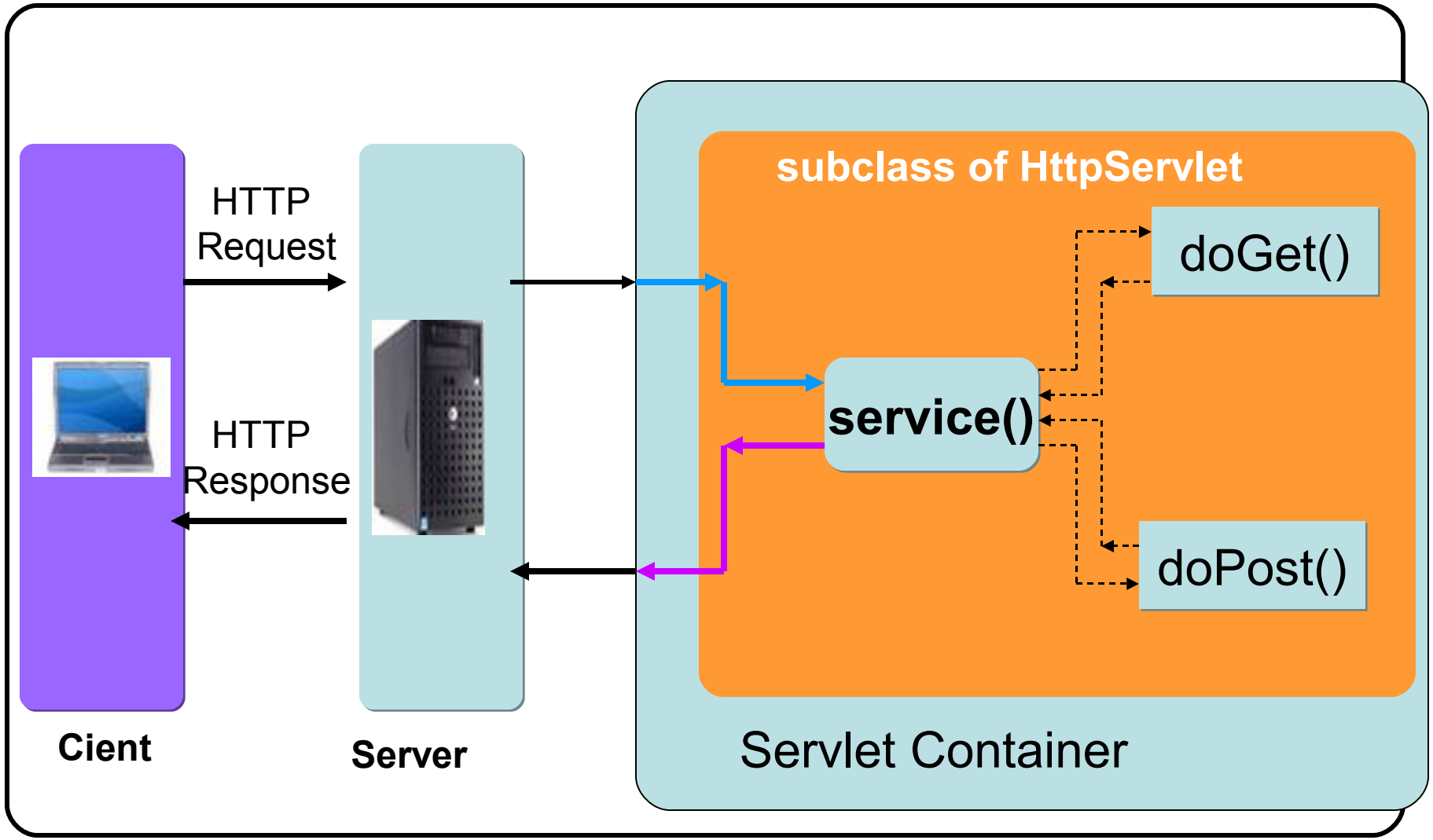
service() & doGet() / doPost()

- service() methods take generic requests and responses:
  - service(ServletRequest request, ServletResponse response)
- doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponseresponse)

service() method



doGet() and doPost() method



Things You Do in doGet() & doPost()

- Extract client-sent information (HTTP parameter) from HTTP request
- Set (Save) and get (read) attributes to/from Scope objects
- Perform some business logic or access database
- Optionally forward the request to other Web components (Servlet or JSP)
- Populate HTTP response message and send it to client

Steps of Populating HTTP Response

- Fill Response headers
- Set some properties of the response
  - Buffer size
- Get an output stream object from the response
- Write body content to the output stream

Request Object (ServletRequest and HttpServletRequest)

What is Servlet Request?

- Contains data passed from client to servlet
- All servlet requests implement ServletRequest interface which defines methods for accessing
  - Client sent parameters
  - Object-valued attributes
  - Locales
  - Client and server
  - Input stream
  - Protocol information
  - Content type
  - If request is made over secure channel (HTTPS)

Getting Client Parameters(1)

- A request can come with any number of parameters
- Parameters are sent from HTML forms:
  - GET: as a query string, appended to a URL
  - POST: as encoded POST data, not appeared in the URL
- getParameter(paraName)
  - Returns the value of paraName
  - Returns null if no such parameter is present
  - Works identically for GET and POST requests

Getting client parameters (2)

- getParameter can be used if name is known and there's only one value for the parameter
- Example: suppose that request is an **`HttpServletRequest`** object and that "name" and "age" are parameters:

```
String name = request.getParameter("name");
String age = request.getParameter("age");
```

- Note: values are always returned as strings

Getting client parameters (3)

- If a parameter has no value or there is no parameter with the specified name then null is returned so we can check using

```
String name = request.getParameter("name");
if (name == null)
{
   // didn't get parameter or name
}
else
{
   // parameter exists and has a value
}
```

Getting client parameters (4)

- Getting list of single-valued parameters

```
Enumeration names = request.getParameterNames();
if (names.hasMoreElements())
{   ...
    while (names.hasMoreElements())
    {
        String name = (String) names.nextElement();
        String value = request.getParameter(name);
        // do something with name and value here
    }
else
{
    // there were no parameters in request
}
```

Getting client parameters (5)

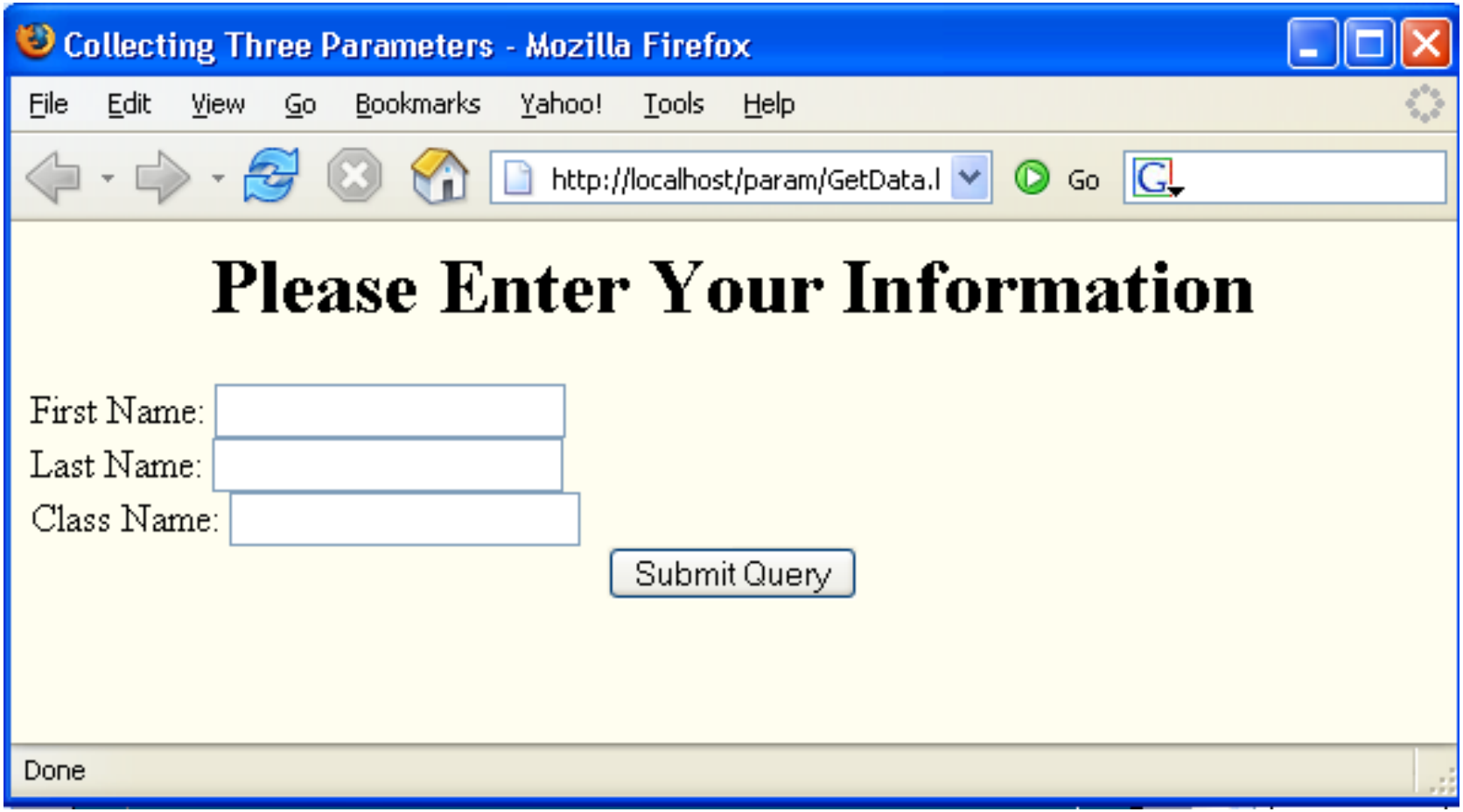
- Getting list of multiple-valued parameters.
  Use the following while loop

```
while (names.hasMoreElements())
{
   String name = (String) names.nextElement();
   String[] values =
      request.getParameterValues(name);
   for (int i = 0; i < values.length; i++)
   {
      // do something with the ith parameter with
      // this name
   }
}
```

A Sample FORM using GET

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Please Enter Your Information</H1>
<FORM ACTION="Params">
First Name: <INPUT TYPE="TEXT" NAME="param1"><BR>
Last Name: <INPUT TYPE="TEXT" NAME="param2"><BR>
Class Name: <INPUT TYPE="TEXT" NAME="param3"><BR>
<CENTER>
<INPUT TYPE="SUBMIT">
</CENTER>
</FORM>
</BODY>
</HTML>
```

A Sample Form Using GET

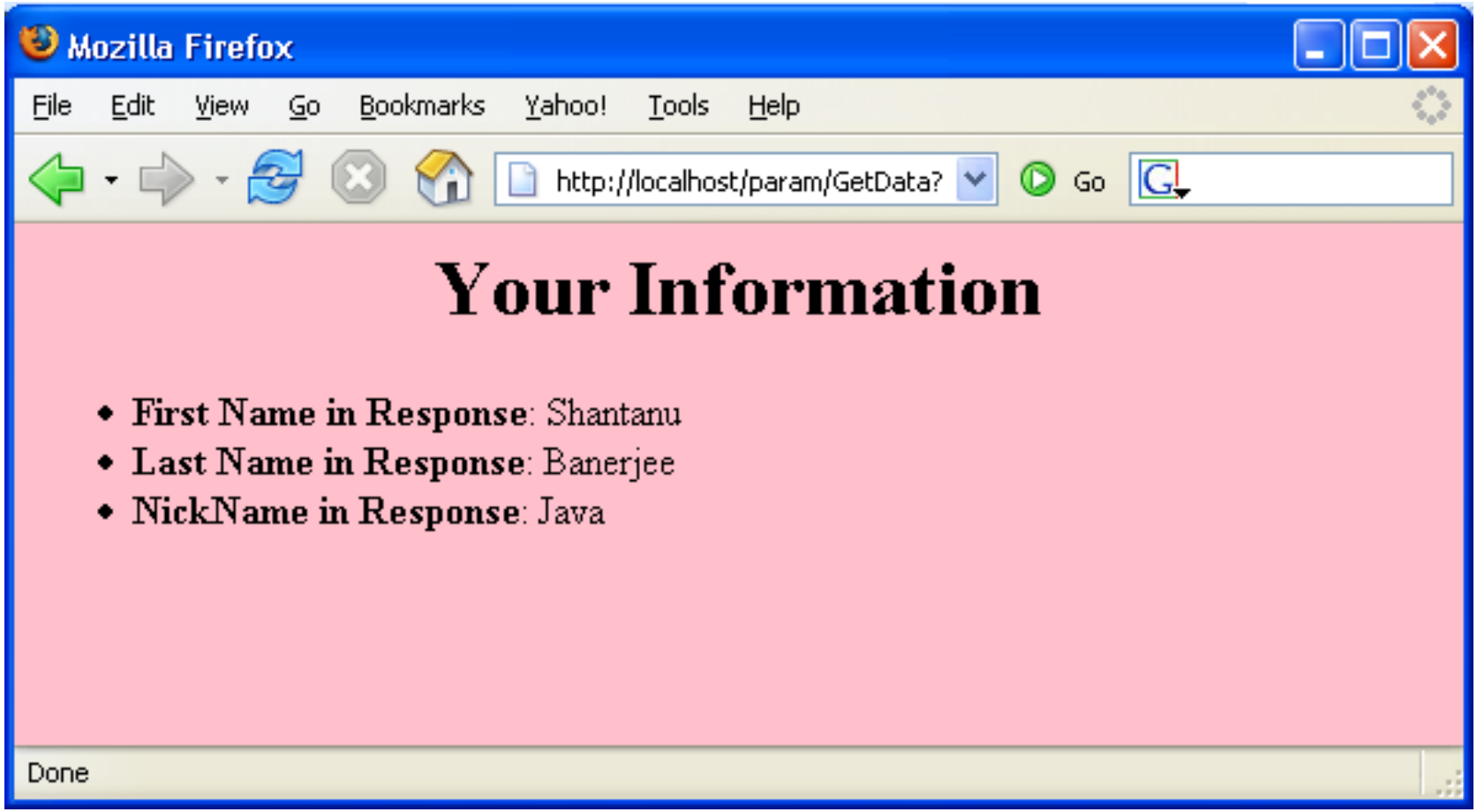


A FORM Based Servlet: Get

```
package com.example;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class GetData extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException,
        IOException {
        response.setContentType("text/html");
                PrintWriter out = response.getWriter();
            String title = "Your Information";
out.println("<HTML>" +"<BODY BGCOLOR=\"pink\">\n"
+"<H1 ALIGN=CENTER>" + title + "</H1>\n" +
"<UL>\n" +" <LI><B>First Name in Response</B>: "
+ request.getParameter("param1") + "\n" +
" <LI><B>Last Name in Response</B>: "
+request.getParameter("param2") + "\n" +
" <LI><B>NickName in Response</B>: "
+request.getParameter("param3") + "\n" +
"</UL>\n" +"</BODY></HTML>");
        }
}
```

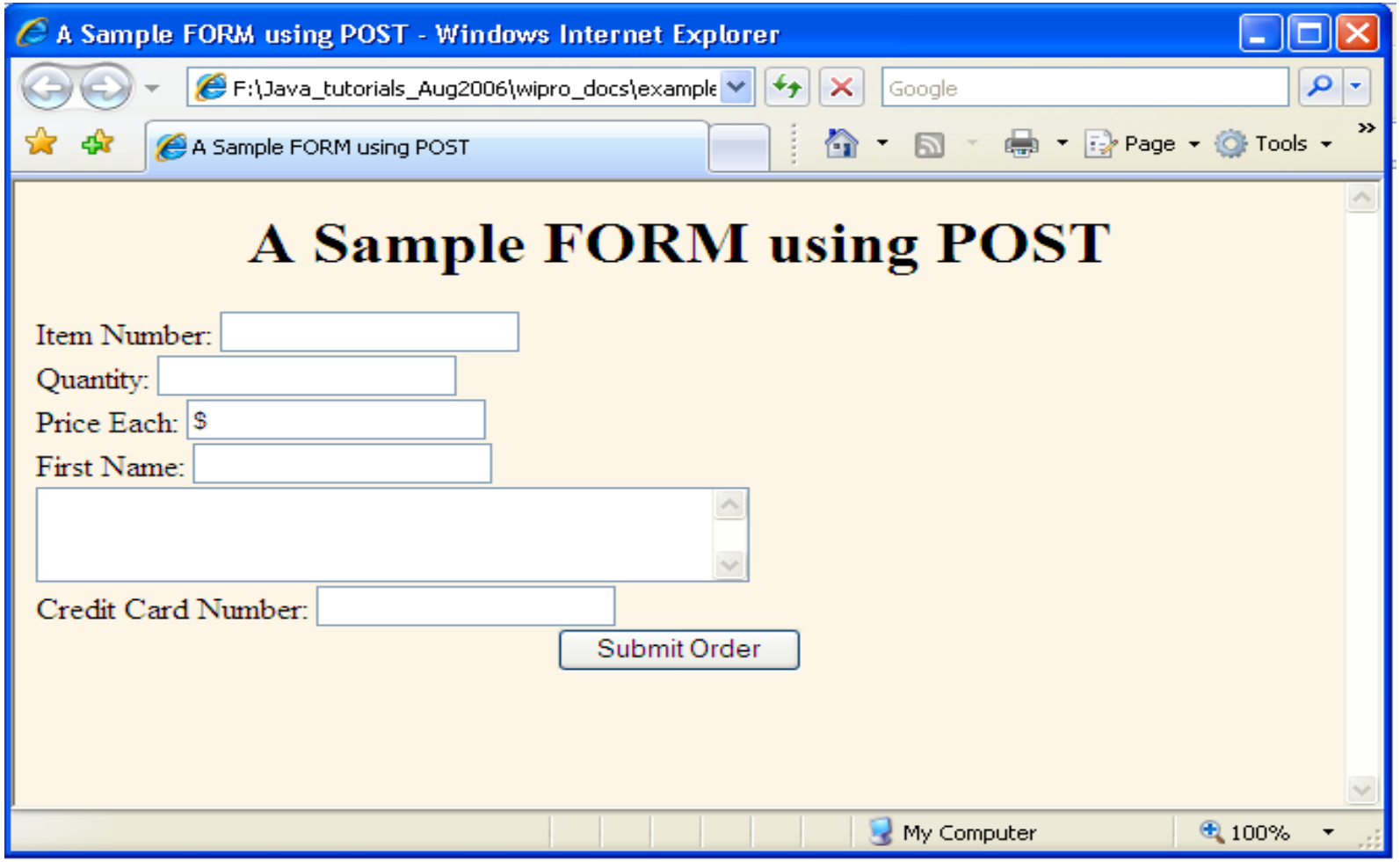
Response from the Servlet



A Sample FORM using POST

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>
<FORM ACTION="ShowParameters" METHOD="POST">
Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
Quantity: <INPUT TYPE="TEXT" NAME="quantity"><BR>
Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
<TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
Credit Card Number:
<INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
<CENTER>
<INPUT TYPE="SUBMIT" VALUE="Submit Order">
</CENTER>
</FORM>
</BODY>
</HTML>
```

A Sample form using POST



A Form Based Servlet: POST

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShowParameters extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
...
}
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
doGet(request, response);
}
}
```

Accessing Database using Servlet

- when we write JDBC code in a servlet we divide the code into the three servlet methods
  1. init()
     1. contains the initialization code like loading the jdbc driver and creating the connection. Executed only once
  2. destroy()
     1. contains the part of jdbc code related to closing the Connection, Statement and ResultSet Objects. Executed only once
  3. service() (or doGet()/doPost())
     1. contains the code related to executing the statement and processing the Result
     2. generating the user Response
     3. this part of the code is executed whenever a client requests for the data

Accessing Database using Servlet

```
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmpInfo extends HttpServlet
{
   Connection con;
   Statement st;
   ResultSet rs;
        int id;
public void init(ServletConfig config) throws ServletException
  {
   super.init(config);
   try{
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
           con=DriverManager.getConnection( "jdbc:odbc:datasource","scott","tiger");
           st=con.createStatement();
             }catch(Exception se) {
                    System.out.println("problem with connection establishment");}
  }
```

Accessing Database using Servlet

```
public void doGet(HttpServletRequest req, HttpServletResponse res)  throws ServletException, IOException
 {
            System.out.println("\n in the doGet ");
         res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        id = Integer.parseInt(req.getParameter("no"));
  try
  {
       ResultSet rs = st.executeQuery("Select name,sal from empsal1 where id="+id);
 while( rs.next())
          {
         out.println("Employee No :"+id);
        out.println("Name :"+rs.getString(2));
        out.println("Salary : "+rs.getInt(3));
          }
        out.close();
       } catch(SQLException e) {
        System.out.println("error while retrieving the data "+e);
    }
}
```

Accessing Database using Servlet

```
public void destroy()
    {
    System.out.println("\n in the destroy");
            try
            {
                    rs.close();
                    st.close();
                    con.close();
            }
            catch (SQLException e )
            {
            System.out.println("Sorry connections could not be closed");
            }
    }
};
```

Form to Access Database

```
<html>
<body>
<form method="get" action="EmpInfo">
Enter Employee Number : <input type="text" name = "no">
 <input  type="submit" value="Find Out">
</form>
</body>
</html>
```

Who Set Object/value Attributes

- Request attributes can be set in two ways
  - Servlet container itself might set attributes to make available custom information about a request
    - example: javax.servlet.request.X509Certificate attribute for HTTPS
  - Servlet set application-specific attribute
    - void setAttribute(java.lang.String name, java.lang.Object o)
- Embedded into a request before a `RequestDispatcher` call

Getting Client Information

- Servlet can get client information from the request
  - String request.getRemoteAddr()
    - Get client's IP address
  - String request.getRemoteHost()
    - Get client's host name

Getting Server Information

- Servlet can get server's information:
  - String request.getServerName()
    - e.g. www.sun.com
  - int request.getServerPort()
    - e.g. Port number 8080

Getting Misc. Information

- Input stream
  - ServletInputStream getInputStream()
  - java.io.BufferedReader getReader()
- Protocol
  - java.lang.String getProtocol()
- Content type
  - java.lang.String getContentType()
- Is secure or not (if it is HTTPS or not)
  - boolean isSecure()

Additional methods in HttpServletRequest

HttpServletRequest Methods

- HttpServletRequest is an extension of ServletRequest and provides additional methods for accessing
  - HTTP request URL
  - Context, servlet, path, query information
  - Misc. HTTP Request header information
  - Authentication type & User security information
  - Cookies
  - Session

HTTP Request URL

- Contains the following parts
  - http://[host]:[port]/[request path]?[query string]

HTTP Request URL: [request path]

- http://[host]:[port]/[request path]?[query string] [request path] is made of

  Context: /<context of web app>

  Servlet name: /<component alias>

  Path information: the rest of it

- Examples
  - http://localhost:8080/hello1/greeting
  - http://localhost:8080/hello1/greeting.jsp
  - http://daydreamer/catalog/lawn/index.html

Context,Path,Query, Parameter methods

- String getContextPath()
- String getQueryString()
- String getPathInfo()
- String getPathTranslated()

HTTP Request Headers

- HTTP requests include headers which provide extra information about the request
- Example of HTTP 1.1 Request:

```
GET /search? keywords= servlets+ jsp HTTP/ 1.1
Accept: image/ gif, image/ jpg, */*
Accept-Encoding: gzip
Connection: Keep- Alive
Cookie: userID= id456578
Host: www.sun.com
Referer: http:/www.sun.com/codecamp.html
User-Agent: Mozilla/ 4.7 [en] (Win98; U)
```

HTTP Request Headers

- Accept
  - Indicates MIME types browser can handle.
- Accept-Encoding
  - Indicates encoding (e. g., gzip or compress) browser can handle
- Authorization
  - User identification for password- protected pages
  - Instead of HTTP authorization, use HTML forms to send username/password and store info in session object

HTTP Request Headers

- Connection
  - In HTTP 1.1, persistent connection is default
  - Servlets should set Content-Length with setContentLength (use ByteArrayOutputStream to determine length of output) to support persistent connections.
- Cookie
  - Gives cookies sent to client by server sometime earlier. Use getCookies, not getHeader
- Host
  - Indicates host given in original URL.
  - This is required in HTTP 1.1.

HTTP Request Headers

- If-Modified-Since
  - Indicates client wants page only if it has been changed after specified date.
  - Don’t handle this situation directly; implement getLastModified instead.
- Referer
  - URL of referring Web page.
  - Useful for tracking traffic; logged by many servers.
- User-Agent
  - String identifying the browser making the request.
  - Use with extreme caution!

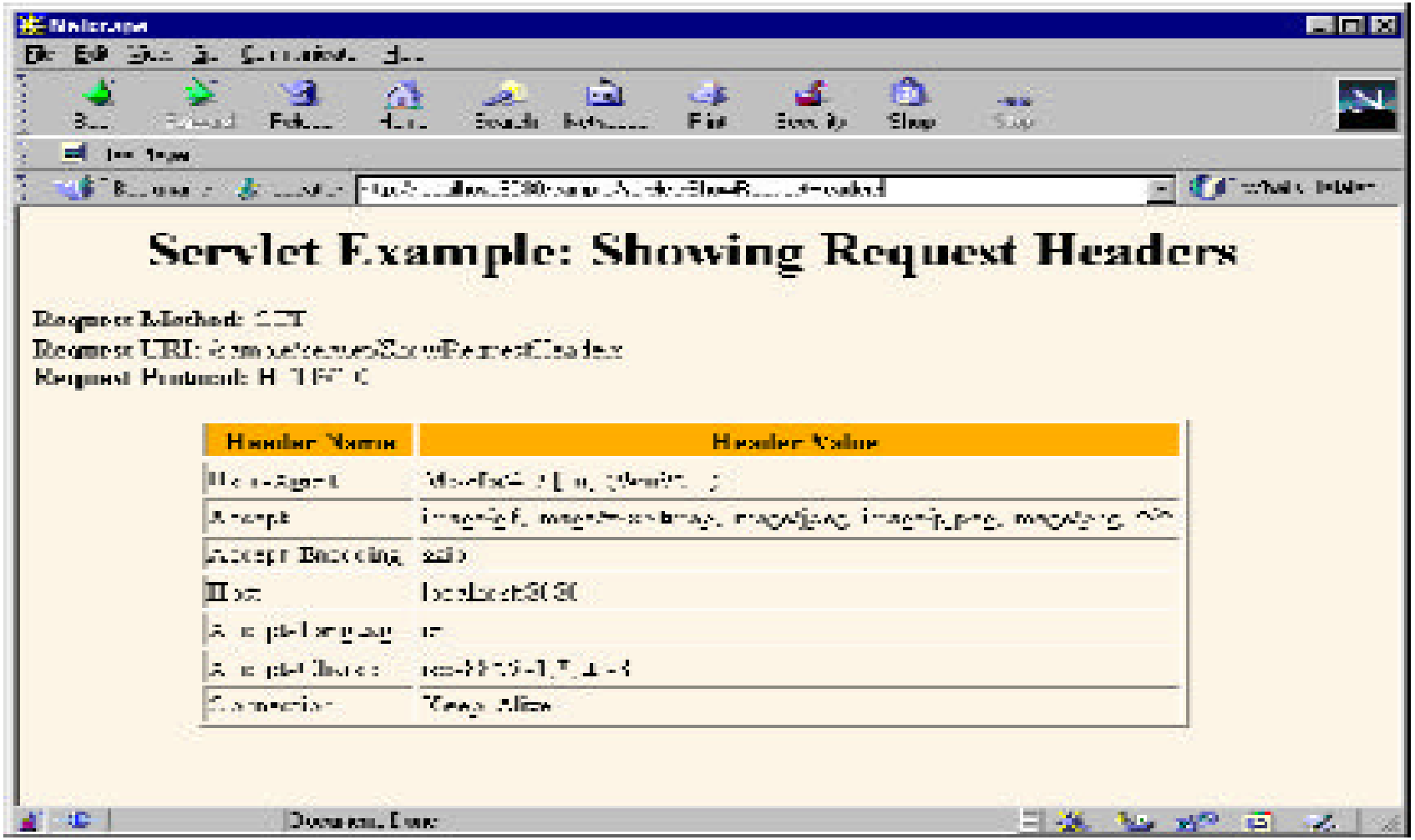
HTTP Header Methods

- String getHeader(java.lang.String name)
  - value of the specified request header as String
- java.util.EnumerationgetHeaders(java.lang.String name)
  - values of the specified request header
- java.util.Enumeration getHeaderNames()
  - names of request headers
- int getIntHeader(java.lang.String name)
  - value of the specified request header as an int

Showing Request Headers

```
//Shows all the request headers sent on this particular request.
public class ShowRequestHeaders extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Servlet Example: Showing Request Headers";
out.println("<HTML>" + ...
"<B>Request Method: </B>" +
request.getMethod() + "<BR>\n" +
"<B>Request URI: </B>" +
request.getRequestURI() + "<BR>\n" +
"<B>Request Protocol: </B>" +
request.getProtocol() + "<BR><BR>\n" +
...
"<TH>Header Name<TH>Header Value");
Enumeration headerNames = request.getHeaderNames();
while(headerNames.hasMoreElements()) {
String headerName = (String)headerNames.nextElement();
out.println("<TR><TD>" + headerName);
out.println(" <TD>" + request.getHeader(headerName) );
}
...
}
}
```

Request Headers Sample



Authentication & User Security Information

- String getRemoteUser()
  - name for the client user if the servlet has been password protected, null otherwise
- String getAuthType()
  - name of the authentication scheme used to protect the servlet
- boolean isUserInRole(java.lang.String role)
  - Is user included in the specified logical role ?
- String getRemoteUser()
  - login of the user making this request, if the user has been authenticated, null otherwise

Cookie Method (in HTTPServletRequest)

- Cookie[] getCookies()
  - an array containing all of the Cookie objects the client sent with this request

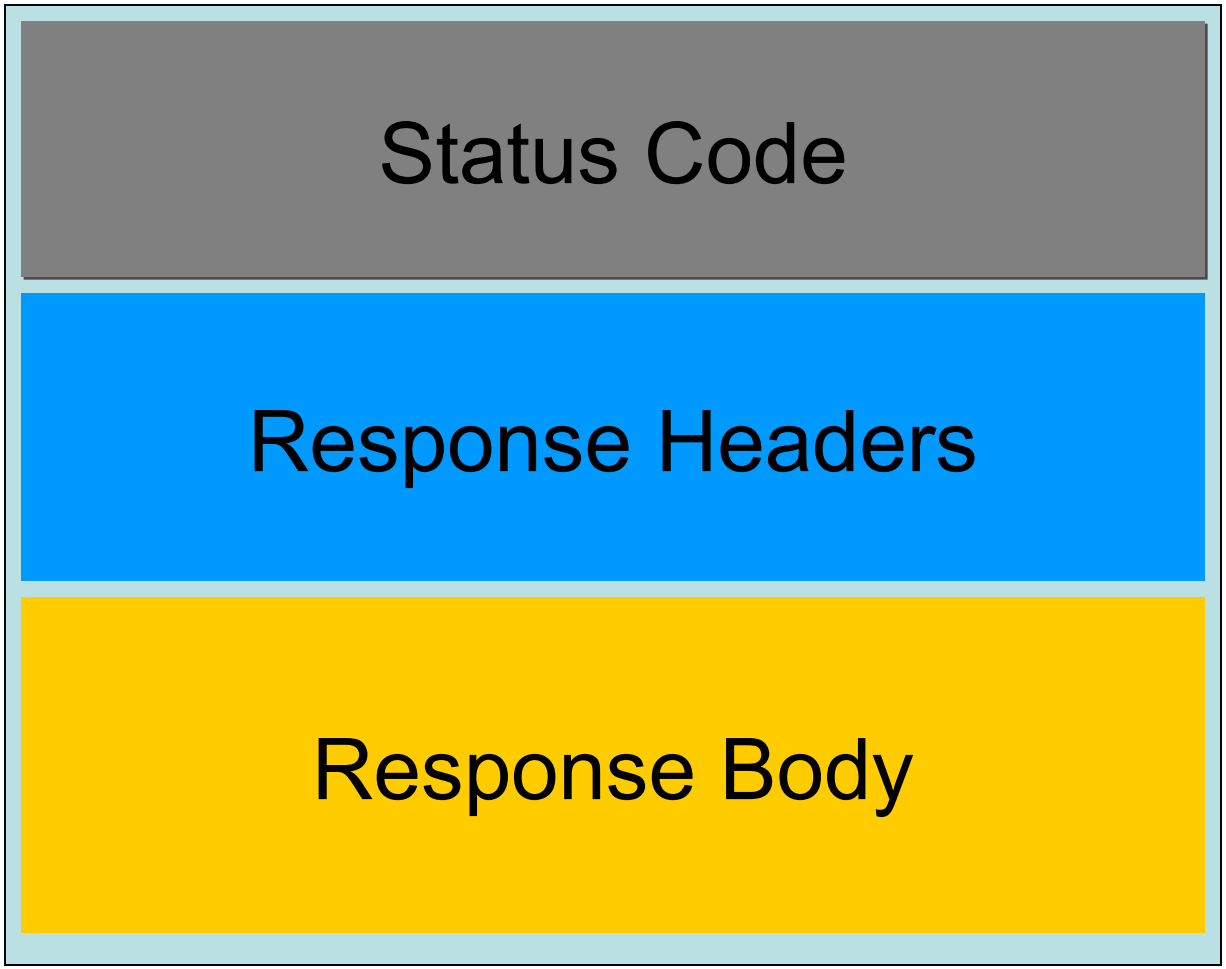
Servlet Response (HttpServletResponse)

What is Servlet Response?

- Contains data passed from servlet to client
- All servlet responses implement ServletResponse interface
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information

- HttpServletResponse extends ServletResponse
  - HTTP response status code
  - Cookies

Response Structure



Status Code in Http Response

HTTP Response Status Codes

- Why do we need HTTP response status code?
  - Forward client to another page
  - Indicates resource is missing
  - Instruct browser to use cached copy

Methods for Setting HTTP Response Status Codes

- public void setStatus(int statusCode)
  - Status codes are defined in HttpServletResponse
  - Status codes are numeric fall into five general categories:
    - 100-199 Informational
    - 200-299 Successful
    - 300-399 Redirection
    - 400-499 Incomplete
    - 500-599 Server Error
  - Default status code is 200 (OK)

Example of HTTP Response Status

```
HTTP/ 1.1 200 OK
Content-Type: text/ html
<! DOCTYPE ...>
<HTML
...
</ HTML>
```

Common Status Codes

- 200 (SC_OK)
  - Success and document follows
  - Default for servlets
- 204 (SC_No_CONTENT)
  - Success but no response body
  - Browser should keep displaying previous document
- 301 (SC_MOVED_PERMANENTLY)
  - The document moved permanently (indicated in Location header)
  - Browsers go to new location automatically

Common Status Codes

- 302 (SC_MOVED_TEMPORARILY)
  - Note the message is Found
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use sendRedirect, not setStatus,when setting this header
- 401 (SC_UNAUTHORIZED)
  - Browser tried to access password- protected page without proper Authorization header
- 404 (SC_NOT_FOUND)
  - No such page

Methods for Sending Error

- Error status codes (400-599) can be used in sendError methods.
- public void sendError(int sc)
  - The server may give the error special treatment
- public void sendError(int code, String
- message)
  - Wraps message inside small HTML document

setStatus() & sendError()

```
try {
        returnAFile(fileName, out)
    }
    catch (FileNotFoundException e){
    response.setStatus(response.SC_NOT_FOUND);
        out.println("Response body");
    }
        //has same effect as
try {
        returnAFile(fileName, out)
    }
    catch (FileNotFoundException e)
    { response.sendError(response.SC_NOT_FOUND);
    }
```

Header in Http Response

Why HTTP Response Headers?

- Give forwarding location
- Specify cookies
- Supply the page modification date
- Instruct the browser to reload the page after a designated interval
- Give the file size so that persistent HTTP connections can be used
- Designate the type of document being generated
- Etc.

Methods for Setting Arbitrary Response Headers

- public void setHeader(String headerName, String headerValue)
  - Sets an arbitrary header.
- public void setDateHeader(String name, long millisecs)
  - Converts milliseconds since 1970 to a date string in GMT format
- public void setIntHeader(String name, int headerValue)
  - Prevents need to convert int to String before calling setHeader
- addHeader, addDateHeader, addIntHeader
  - Adds new occurrence of header instead of replacing.

Methods for setting Common Response Headers

- setContentType
  - Sets the Content- Type header. Servlets almost always use this.
- setContentLength
  - Sets the Content- Length header. Used for persistent HTTP connections.
- addCookie
  - Adds a value to the Set- Cookie header.
- sendRedirect
  - Sets the Location header and changes status code.

Common HTTP 1.1 Response Headers

- Location
  - Specifies a document's new location.
  - Use sendRedirect instead of setting this directly.
- Refresh
  - Specifies a delay before the browser automatically reloads a page.
- Set-Cookie
  - The cookies that browser should remember. Don’t set this header directly.
  - use addCookie instead.

Common HTTP 1.1 Response Headers (cont.)

- Cache-Control (1.1) and Pragma (1.0)
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- Content- Encoding
  - The way document is encoded. Browser reverses this encoding before handling document.
- Content- Length
  - The number of bytes in the response. Used for persistent HTTP connections.

Common HTTP 1.1 Response Headers (cont.)

- Content- Type
  - The MIME type of the document being returned.
  - Use setContentType to set this header.
- Last- Modified
  - The time document was last changed
  - Don’t set this header explicitly.
  - provide a getLastModified method instead.

Refresh Sample Code

```
public class DateRefresh extends HttpServlet
  {
  public void doGet(HttpServletRequest req,
          HttpServletResponse res)
     throws ServletException, IOException {

     res.setContentType("text/plain");
     PrintWriter out = res.getWriter();
     res.setHeader("Refresh", "5");
     out.println(new Date().toString());
  }
}
```

Body in Http Response

Writing a Response Body

- A servlet almost always returns a response body
- Response body could either be a PrintWriter or a ServletOutputStream
- PrintWriter
  - Using response.getWriter()
  - For character-based output
- ServletOutputStream
  - Using response.getOutputStream()
  - For binary (image) data

Scope Objects

Scope Objects

- Enables sharing information among collaborating web components via attributes maintained in Scope objects
  - Attributes are name/object pairs
- Attributes maintained in the Scope objects are accessed with
  - getAttribute() & setAttribute()
- 4 Scope objects are defined
  - Web context, session, request, page

Four Scope Objects: Accessibility

- Web context (ServletConext)
  - Accessible from Web components within a Web context
- Session
  - Accessible from Web components handling a request that belongs to the session
- Request
  - Accessible from Web components handling the request
- Page
  - Accessible from JSP page that creates the object

Four Scope Objects: Class

- Web context
  - javax.servlet.ServletContext
- Session
  - javax.servlet.http.HttpSession
- Request
  - subtype of javax.servlet.ServletRequest: javax.servlet.http.HttpServletRequest
- Page
  - javax.servlet.jsp.PageContext

Web Context (ServletContext)

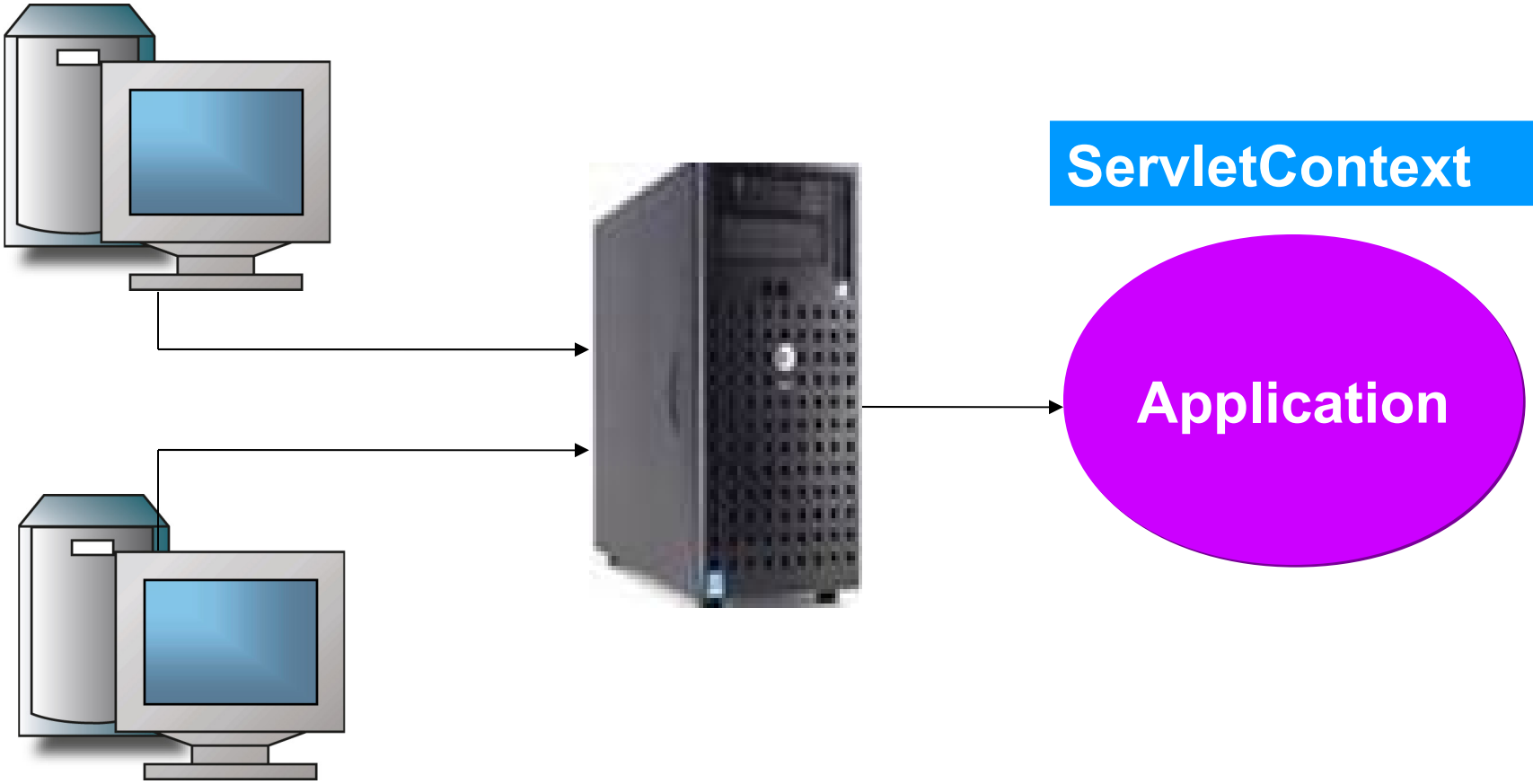
What is ServletContext For?

- Used by servets to
  - Set and get context-wide (application-wide) object-valued attributes
  - Get request dispatcher
    - To forward to or include web component
  - Access Web context-wide initialization parameters set in the web.xml file
  - Access Web resources associated with the Web context
  - Log
  - Access other misc. information

Scope of ServletContext

- Context-wide scope
  - Shared by all servlets and JSP pages within a "web " application
    - Why it is called “web application scope”
  - " A web application” is a collection of servlets and content installed under a specific subset of the server's URL namespace and possibly installed via a *.war file
    - All servlets of a web application share same ServletContext object
  - There is one " ServletContext object per web " application per Java Virtual Machine

Servlet Context : Web Application Scope



many clients-> One Web Application -> One ServletContext

How to Access ServletContext Object?

- Within your servlet code, call `getServletContext()`
- Within your servlet filter code, call `getServletContext()`
- The ServletContext is contained in ServletConfig object, which the Web server provides to a servlet when the servlet is initialized
  - init (ServletConfig servletConfig) in Servlet interface

Example: Getting Attribute Value from ServletContext

```
public class CatalogServlet extends HttpServlet {
private BookDB bookDB;
public void init() throws ServletException {
// Get context-wide attribute value from
// ServletContext object
bookDB = (BookDB)getServletContext().
getAttribute("bookDB");
if (bookDB == null) throw new
UnavailableException("Couldn't get database.");
}
}
```

This is an servlet example code in which the value of a context-wide attribute called bookDB is retrieved in init() method of the servlet class.

03/17/2006

Example: Getting and Using

RequestDispatcher Object

```
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
HttpSession session = request.getSession(true);
ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");
// set headers and buffer size before accessing the Writer
response.setContentType("text/html");
response.setBufferSize(8192);
PrintWriter out = response.getWriter();
// then write the response
out.println("<html>" +
"<head><title>" + messages.getString("TitleBookDescription") +
"</title></head>");
// Get the dispatcher; it gets the banner to the user
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null)
dispatcher.include(request, response);
...
```

Example: Logging

```
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
...
getServletContext().log("Life is good!");
...
getServletContext().log("Life is bad!", someException);
```

Session
(HttpSession)

Why HttpSession?

- Need a mechanism to maintain client state across a series of requests from a same user (or originating from the same browser) over some period of time
  - Example: Online shopping cart
- Yet, HTTP is stateless
- HttpSession maintains client state
  - Used by Servlets to set and get the values of session scope attributes

How to Get HttpSession?

- via getSession() method of a Request object (HttpRequest)

Example: HttpSession

```
public class CashierServlet extends HttpServlet {
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Get the user's session and shopping cart
HttpSession session = request.getSession();
ShoppingCart cart =
(ShoppingCart)session.getAttribute("cart");
...
// Determine the total price of the user's books
double total = cart.getTotal();
```

Handling Errors

Handling Errors

- Web container generates default error page
- You can specify custom default page to be displayed instead
- Steps to handle errors
  - Create appropriate error html pages for error conditions
  - Modify the web.xml accordingly

Example: Setting Error Pages in web.xml

```
<error-page>
    <exception-type>
        exception.BookNotFoundException
    </exception-type>
        <location>/errorpage1.html</location>
    </error-page>
<error-page>
    <exception-type>
        exception.BooksNotFoundException
    </exception-type>
    <location>/errorpage2.html</location>
</error-page>
<error-page>
        <exception-type>exception.OrderException</exception-type>
        <location>/errorpage3.html</location>
</error-page>
```