

Introduction to Java Servlets

James Baldo Jr.

SWE 432

Design and Implementation of Software for the Web

Web Applications

- A web application uses enabling technologies to
 1. make web site contents dynamic
 2. allow users of the system to implement **business logic** on the server
- Web applications allow users to affect state on the server
- Search engines, though complicated, do not really affect the server's state

An enabling technology is a mechanism that makes web pages interactive and responsive to user input

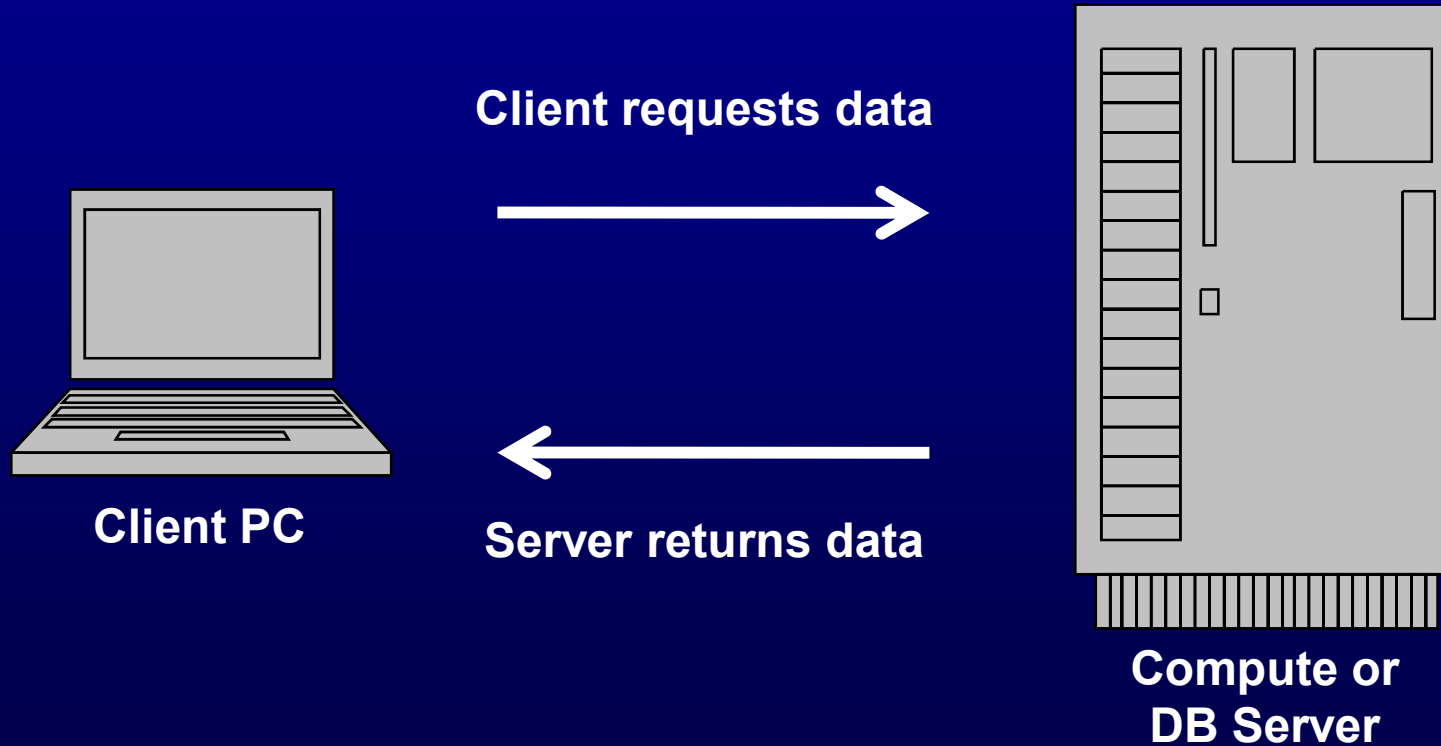
Traditional Computing Use

A user is working with software on her computer



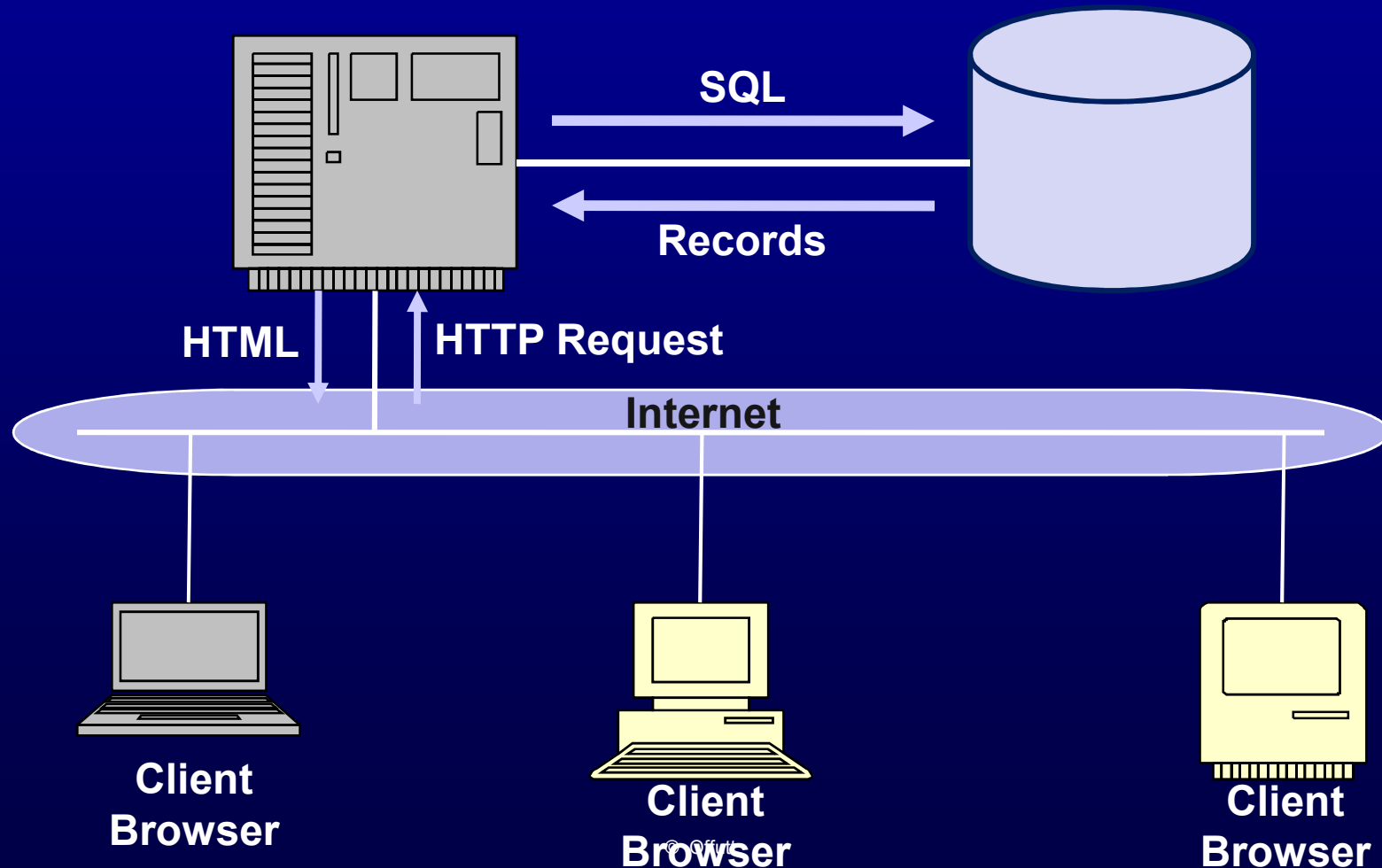
Client – Server Computing

A user is working with software or data on a separate computer called a **server**



Web Applications

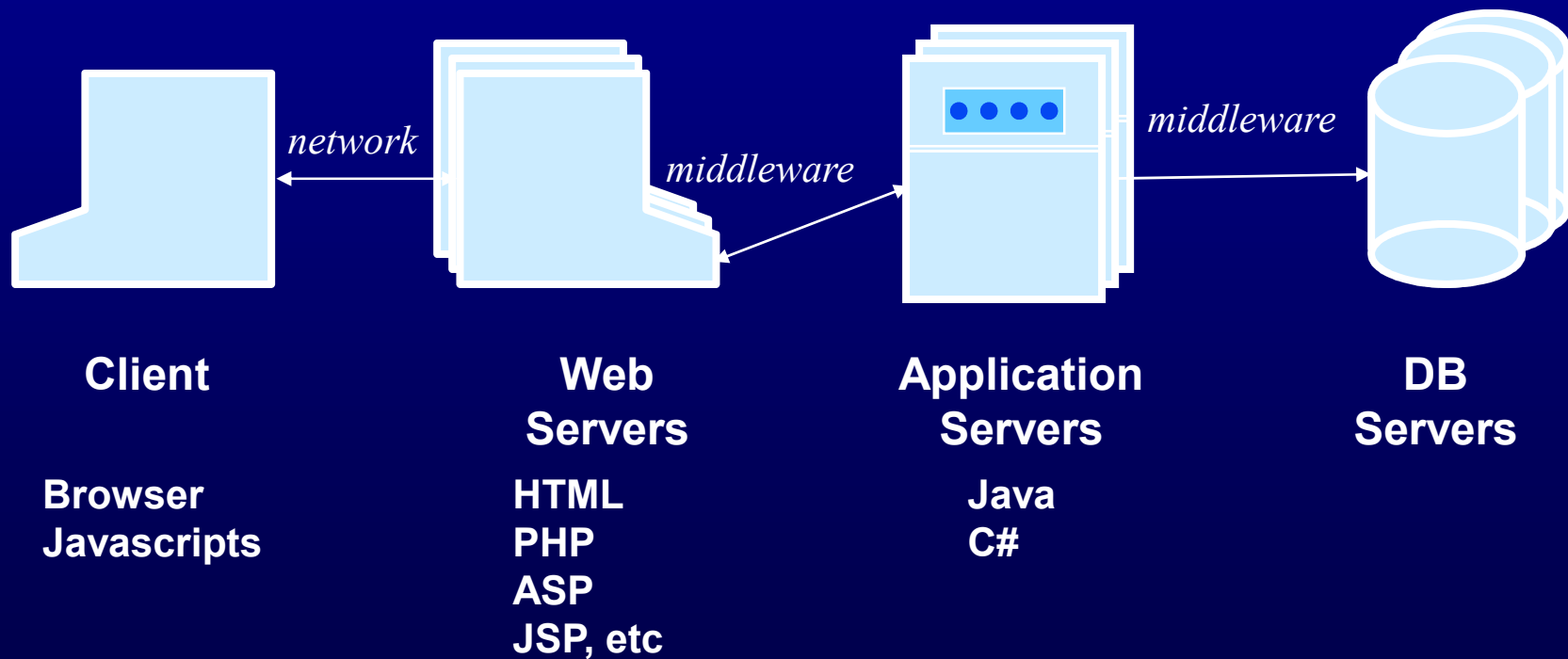
Many users work with servers and databases that can be reached through the Internet with the HyperText Transfer Protocol



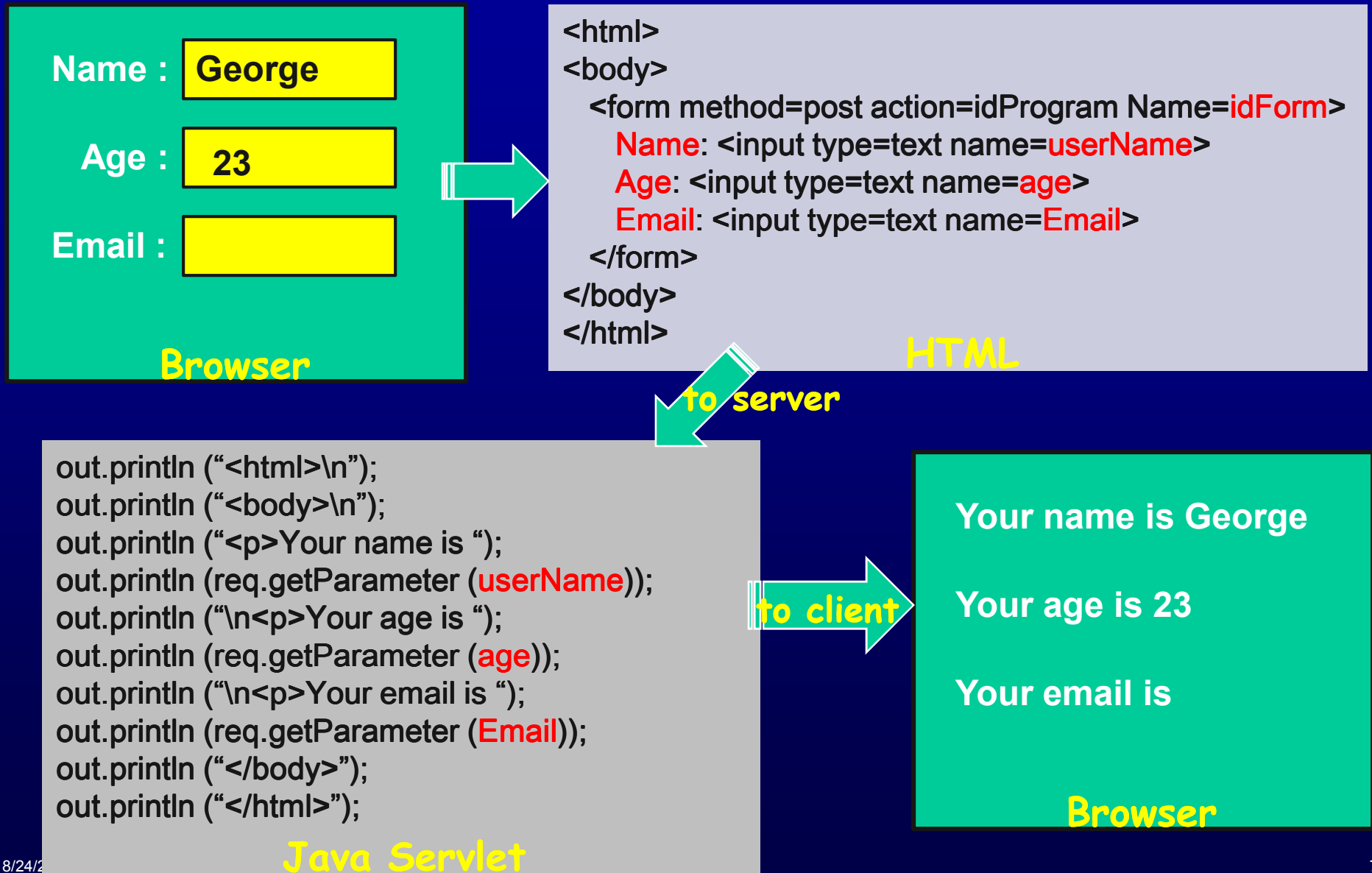
N-Tier Web Architecture

Large web applications run on many computers that have to coordinate with each other.

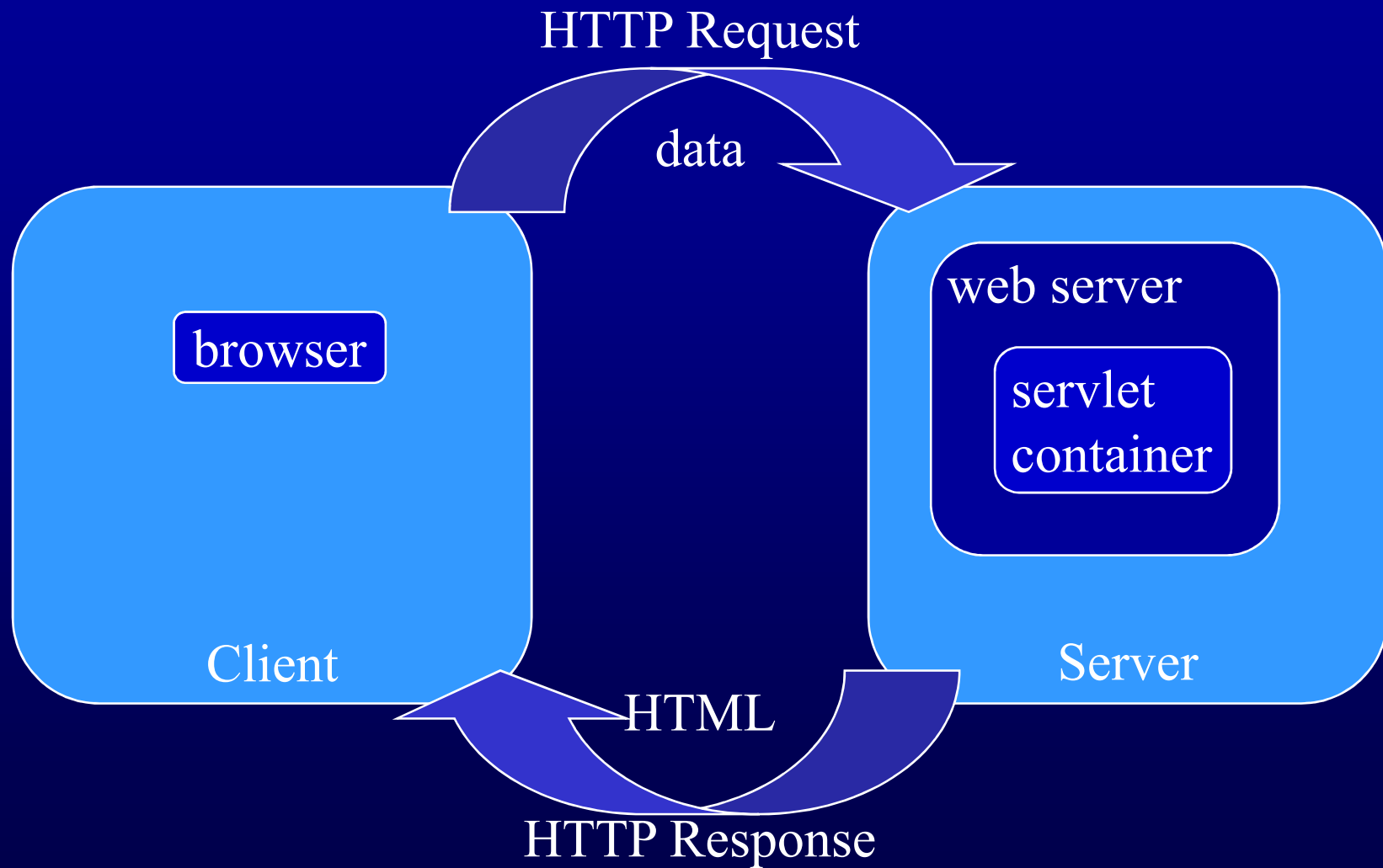
Amazon and **Netflix** have *thousands* of servers.



How the Software Works



Server Side Processing



Session Management

- HTTP client server communication is connectionless
 - as soon as the request is made and fulfilled, the connection is terminated
 - communication is more simple and resistant to network problems
- But how can a server keep track of state of **different** clients?
 1. Session: A single coherent use of the system by the same user
 - Example: shopping carts
 2. Cookies: A string of characters that a web server places on a browser's client to keep track of a session
 - usually used as an index into a table (dictionary) on the server
 - most dictionaries expire after a reasonable time (15 to 30 minutes)

We'll come back to this later ...

Enabling Technologies - CGI

- CGI: Common Gateway Interface allows clients to execute applications on the server
- CGI applications usually reside in a special “safe” directory
- Can be written in any language; PERL is most common
- CGI apps typically:
 1. process data
 2. modify server state
 3. return information (usually an HTML page)

Enabling Technologies

Problems with CGI

- CGI does not automatically provide session management services
- Each and every execution of a CGI module requires a new process on the web server

Solution: Plug-ins on the Web server

Enabling Technologies

Web Server Plug-ins

- A plug-in is an **extension** to a web server that allows a different program to handle certain types of requests
 - **images**, sound, video
 - **compiled module** applications
 - **scripted page** applications
- Plug-ins typically keep an **active process** as long as the web server is active

Enabling Technologies - Plug-ins

Compiled Modules

- Compiled modules are executable programs that the server uses
- Common compiled module application plug-ins:
 - Microsoft's Internet Server API (**ISAPI**) ASP
 - ~~Netscape Server API (**NSAPI**)~~
 - Java servlets
- Compiled modules are efficient and very effective
- They provide clear separation of the front-end from the back-end, which aids design but complicates implementation

Enabling Technologies - Plug-ins

Scripted Pages

- Scripted pages look like **HTML pages** that happen to process business logic
- Execution is server-side, not client (unlike JavaScripts)
- They are HTML pages that access software on the server to get and process data
- JSPs are compiled and run **as servlets** (very clean and efficient)
- **PHP** scripts are **interpreted** within the server

Enabling Technologies - Plug-ins

Scripted Pages (2)

- Common scripted pages:
 - Allaire's **Cold Fusion**
 - Microsoft's Active Server Pages (**ASP**)
 - **PHP**
 - *Java Server Pages* (JSP)
- Scripted pages are generally easy to develop, and deploy
- They mix logic with HTML, so can be difficult to read and maintain
- Not effective for **heavy-duty** engineering

Servlets

- Servlets are **small Java classes** that perform a service
- Servlet container or engine
 - connects to network
 - catches requests
 - produces responses
 - requests are handled by objects
- Programmers use a servlet API

Servlet Container (or Engine)

- Servlet container is a **plug-in** for handling Java servlets
- A servlet container has five jobs:
 1. Creates servlet **instance**
 2. Calls init()
 3. Calls service() whenever a request is made
 4. Calls destroy() before killing servlet
 5. **Destroys** instance
- Really a mini operating system

Servlet Container (2)

When a request comes to a servlet, the servlet container does one of two things:

1. If there is no active object for the servlet, the container instantiates a new object of that class, and the object handles the request
2. If there is an active object, the container creates a Java thread to handle the request

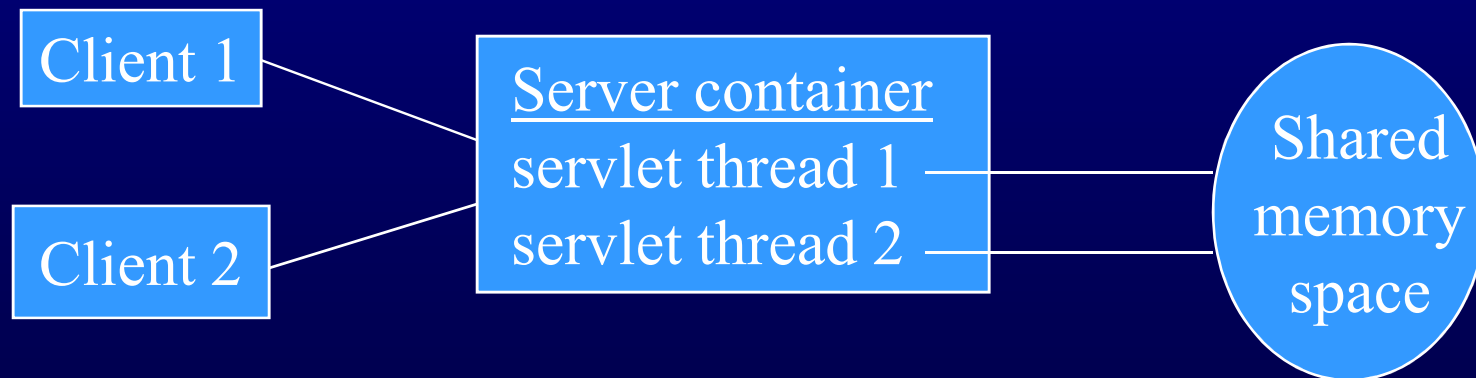
Servlet Container (3)

A **servlet instance** runs until the container decides to destroy it:

- When it gets destroyed is **not specified** by the servlet rules
- Most servlet containers destroy the object N minutes **after the last request**
- N is usually **15 or 30**, and can be set by the system administrator

Servlet Container (4)

- What if the same servlet gets multiple requests?
- More than one execution thread may be running at the same time, using the same memory



Risky ...

Servlet Container (5)

- By default, there is only one instance of a servlet class per servlet definition in the servlet container
- Distributable : If the application is distributable, there is one instance of a servlet class per virtual machine (typically, each VM is on a different computer in the cluster)

Allowing Concurrency (SingleThreadModel)

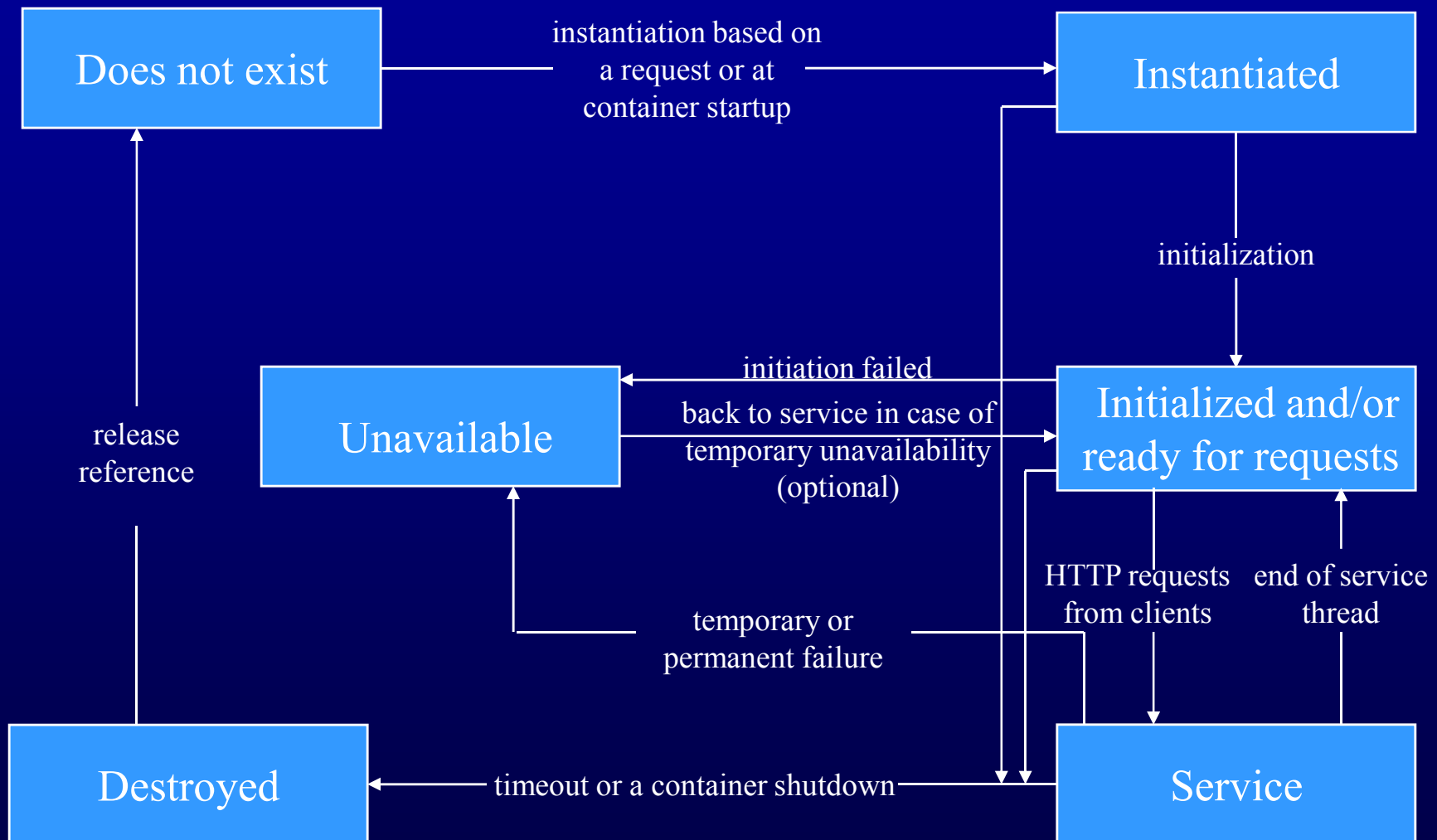
- Container may send multiple service requests to a single instance, using Java threads
 - Simply put, threads are Java's **concurrency** mechanism
- Thus, your service methods (doGet, doPost, etc.) should be thread-safe
 - Loosely speaking, “thread-safe” means that if two or more requests are operating at the same time, they will not interfere with each other.
- If the service methods are not thread-safe, use the SingleThreadModel

SingleThreadModel (2)

- The SingleThreadModel ensures that **only one thread** may execute the `service()` method at a time
- Containers may **implement** this in two ways:
 1. Instance Pooling : A “pool” of several servlet instances are available that do not share memory
 2. Request Serialization : Only one request is handled at a time
 3. Combination : A pool is used, and if there more requests than servlets in the pool, they are serialized
- This is **resource intensive** and can be slow
- Better to **synchronize** only the statements that might interfere with each other

Servlet Lifecycle

UML State Diagram

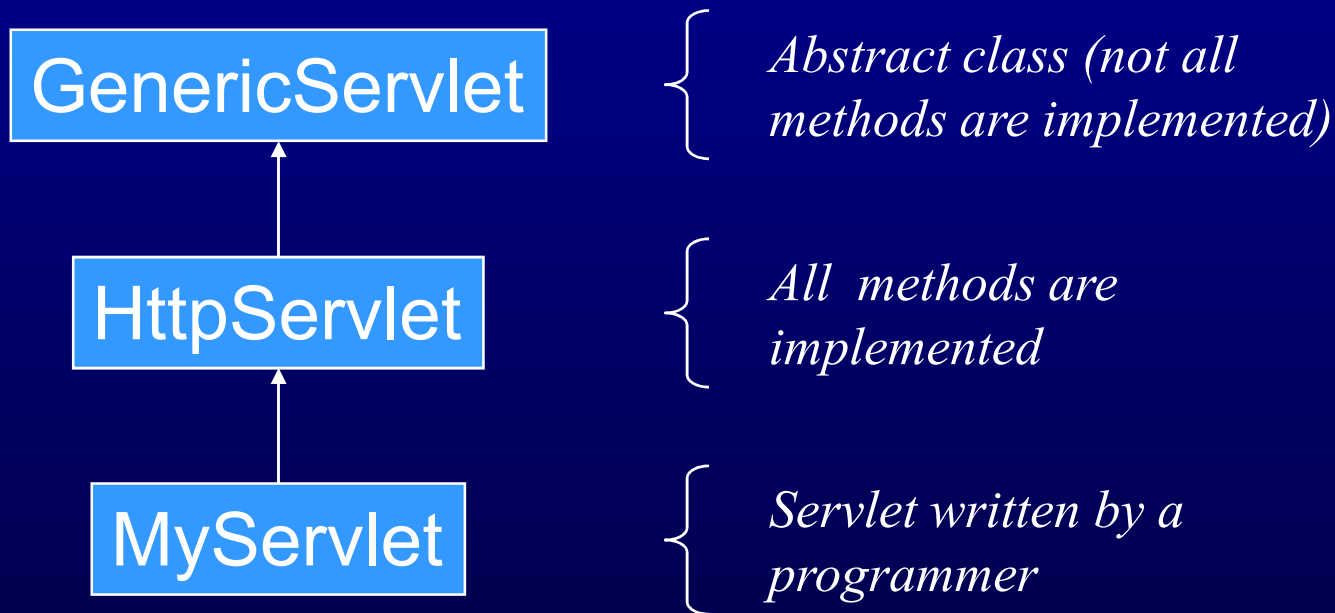


Common Servlet Containers

- Tomcat (installed on Hermes apps cluster)
- BEA's WebLogic
- Jrun
- IBM Websphere (uses Apache / Tomcat)

Servlet API

- javax.servlet – primarily containers
- javax.servlet.http – methods to service requests



Generic Servlet & HTTP Servlet

Servlets have **five** principal methods:

1. **init()** – called when servlet starts
2. **service()** – called to process requests
3. **destroy()** – called before servlet process ends
4. **getServletConfig()** – servlet can access information about servlet container
5. **getServletInfo()** – servlet container can access info about servlet

Generic Servlet & HTTP Servlet (2)

- These methods are defined by the library classes **GenericServlet** and **HttpServlet**
- We write servlets by extending (inheriting from) them
- **GenericServlet** does not implement `service()` (it is abstract)
- **HttpServlet** extends **GenericServlet** with:
*service (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException*

1. init ()

- Read **configuration** data
- Read **initialization** parameters (javax.servlet.ServletConfig)
- **Initialize** services:
 - Database driver
 - Connection pool
 - Logging service
- **Seldom used** in simple applications

2. service ()

- The **entry point** for the servlet – this is the method that is called from the servlet container
- Called **after** the initialization (init ())
- Primary purpose is to **decide** what type of request is coming in and then **call** the appropriate method
 - doGet ()
 - doPost ()

Types of HTTP Requests

- GET
- POST
- HEAD
- OPTIONS
- DELETE
- PUT
- TRACE

doGet ()
doPost ()
doHead ()
doOptions ()
doDelete ()
doPut ()
doTrace()

same signatures
as service()

Types of HTTP Requests (2)

- **HttpServlet** implements these methods as “stubs” that print error messages
doGet () ...
{ print ("Error, doGet() not implemented"); }
- Programmers implement services by overriding these methods
(most commonly **doGet()** and **doPost()**)

3) destroy ()

- Called by container when the servlet instance is killed
- The threads from the service() method are given time to terminate before destroy() is called
- Can be used for clean up tasks:
 - Un-registering a database driver
 - Closing a connection pool
 - Informing another application the servlet is stopping
 - Saving state from the servlet

4) `getServletConfig ()`

- Returns a `ServletConfig` object, which stores information about the servlet's configuration
- The `ServletConfig` object was passed into `init()`

5) `getServletInfo ()`

- Returns a `String` object that stores information about the servlet:
 - Author
 - Creation date
 - Description
 - Usage
 - ...
- This string should be formatted for human readability

Simple Servlet Example

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class JOHello extends HttpServlet
{
    public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
                      throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter out = res.getWriter ();
        out.println ("<HTML>");
    }
}
```

Simple Servlet (2)

```
out.println("<HEAD>");
out.println("<TITLE>Servlet example</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("<P>My first servlet.");
out.println("</BODY>");
out.println("</HTML>");
out.close ();
} // end doGet()
} // end JHello
```

<http://apps-swe432.ite.gmu.edu:8080/swe432/servlet/offutt.Hello>

Servlet Parameters — requests

Parameters are conveniently stored in objects

- **String req.getParameter (String KEY)**

Returns value of field with the name = KEY

- **String[] req.getParameterValues (String KEY)**

Returns all values of KEY (eg, checkboxes)

- **Enumeration req.getParameterNames ()**

Returns an Enumeration object with a list of all parameter names

Servlet Output – responses

Standard output is sent directly back to the client browser

- `res.setContentType` (String type)

“text/html” is an HTML page

- `PrintWriter res.getWriter()`

Use `print()` and `println()` to write HTML to browser

Servlet Performance

- Some servlets will run a lot
- Servlets run as lightweight threads, so are fast
- The **network speeds** usually dominate, but:
 - avoid concatenation (“+”)
 - **out.flush()** – Sends current output to user’s screen while servlet continues processing

GET and POST Requests

- A GET request is generated when the URL is entered directly
 - **doGet ()** is called
- An HTML form can generate either a GET or a POST request
 - “... **Method=POST**” or “... **Method=GET**”
- GET requests put form data on the URL as parameters
 - **http://www ... /RunForm?NAME=Jeff&TITLE=prof**
- GET parameters are limited to 1024 bytes
- POST requests put form data in body of request

GET and POST Requests (2)

- Textbooks say:
 - Use GET to retrieve data
 - Use POST to change “state” on server (update file or DB)
 - Use POST when there are a lot of data items
- My usual strategy:
 - Use POST when sending data to server
 - Use GET when no data is sent

GET and POST Requests (3)

- If my servlet is primarily based on processing data and I use POST, implement a simple doGet() method as a filler:

```
...  
<BODY>  
  <CENTER>A Title ...</CENTER>  
  <HR>  
  
  <P>  
    You should run this from  
    <A Href="http://... .html"> http://... .html</A>  
  </BODY>
```

Sending Mail Messages from Servlets

Common job is to gather data from form and send through email

- Import mail utilities:
 - `import sun.net.smtp.SmtpClient;`
- Setup mail header:
 - `send = new SmtpClient ("gmu.edu");`
 - `send.from ("offutt@gmu.edu");`
 - `send.to ("offutt@gmu.edu");`
- Send message:
 - `out = send.startMessage ();`
 - `out.println ("... message header and body ...");`
 - `out.flush ();`
 - `out.close ();`
 - `out.closeServer ();`

Sending Mail Messages (2)

- This is the simplest mechanism for sending email, but is not very powerful
- JavaMail is a collection of abstract library classes for handling mail with a number of different protocols

Redirecting to Another URL from Servlets

Servlets usually generate an HTML file as a response, but sometimes you may want to send the client to a **different** URL.

- `res.sendRedirect (“http://www.ise.gmu.edu/”);`
- Do **not** need to set content type (`setContentType()`)
- The client will be “**sent**” to the specified URL
- Precisely:
 - Server tells the client to generate another request to the new URL
 - Handled by browser, but invisible to the user

Writing to files from Servlets

Common job is to save data into a file

- File must be in a **publically writeable** directory:
 - **cd** // File under your home directory
 - **mkdir Data** // Subdirectory named "Data/"
 - **chmod 777 Data** // Write permission for everyone
- Open a file, write to it, and close it:
 - **FileWriter outfile = new FileWriter ("/home/offutt/Data/info-file");**
 - **outfile.write (... the data to save ...);**
 - **outfile.close ();**
- Open a file in append mode:
 - **FileWriter outfile = new FileWriter ("/home/offutt/Data/info-file", true);**
- Remember **Unix / Windows** path differences!!
 - "info-file" does **NOT** equal "INFO-FILE" !!!

Deployment Testing

- Development and deployment computers often differ
- Web apps must be tested on final deployment platform
 - Must test just as real users use it
- Issues to check for:
 - Different platforms (DOS / Unix / Linux / Mac ...)
 - File names and path names (local/nonlocal, DOS/Unix)
 - Upper case dependencies
 - Incomplete deployment
 - Compiler and runtime system version
 - Permissions (data and DB)

Examples

<http://www.ise.gmu.edu/~offutt/classes/432/Examples/Servlets/>

1. Hello: Prints lots of hellos
2. Name: Accepts and prints a name from a form
3. FormHandler: Processes data from any form
4. ChoosAbs: Processes form data and sends through email
5. LoanCalclater: Compute time to pay off a loan
6. Convert: Convert values
7. Convert2: Better value conversion
8. FileLoad: Uploading files from client
9. StudInfoSys432: Processes data from students
10. Coupling Demo: Demos a tool
11. SessionLifeCycle: Demonstrates session data
12. Attribute Servlet: Prints values in session data
13. Shop: Simple shopping cart