

**Universität Stuttgart**

Institute of Parallel and  
Distributed Systems (IPVS)

Universitätsstraße 38  
D-70569 Stuttgart

# **Net-based Applications: Web Technologies**

Adnan Tariq

(Based on the slides of Dr. Frank Dürr)

# World Wide Web

---

- History of the World Wide Web (WWW)
- Web Architectures
- Basic Technologies
  - Addressing: Uniform Resource Locators (URL)
  - Transfer: HyperText Transfer Protocol (HTTP)
  - Content: HyperText Markup Language (HTML)
- Extended Technologies
  - Common Gateway Interface (CGI)
  - Java Applets
  - Java Servlets / Java Server Pages (JSP)



# WWW History (1): In the Beginning

---

- **Developed at CERN** with intent to...
  - ...create a Hypertext system to enable “information sharing” among geographically distributed research groups.
- **March 1989:** **Tim Berners-Lee** submits proposal at CERN for a Hypertext system
- **Christmas 1990:** First running prototype
  - Web browser & editor: WorldWideWeb
  - Web server: nxoc01.cern.ch
- **May 17<sup>th</sup> 1991:** General release of WWW on central CERN machines
- **Aug. 1991:** Source files available on the Internet
- **Dec. 1991:** Demonstration at Hypertext '91



# WWW History (2): The Second Phase

---

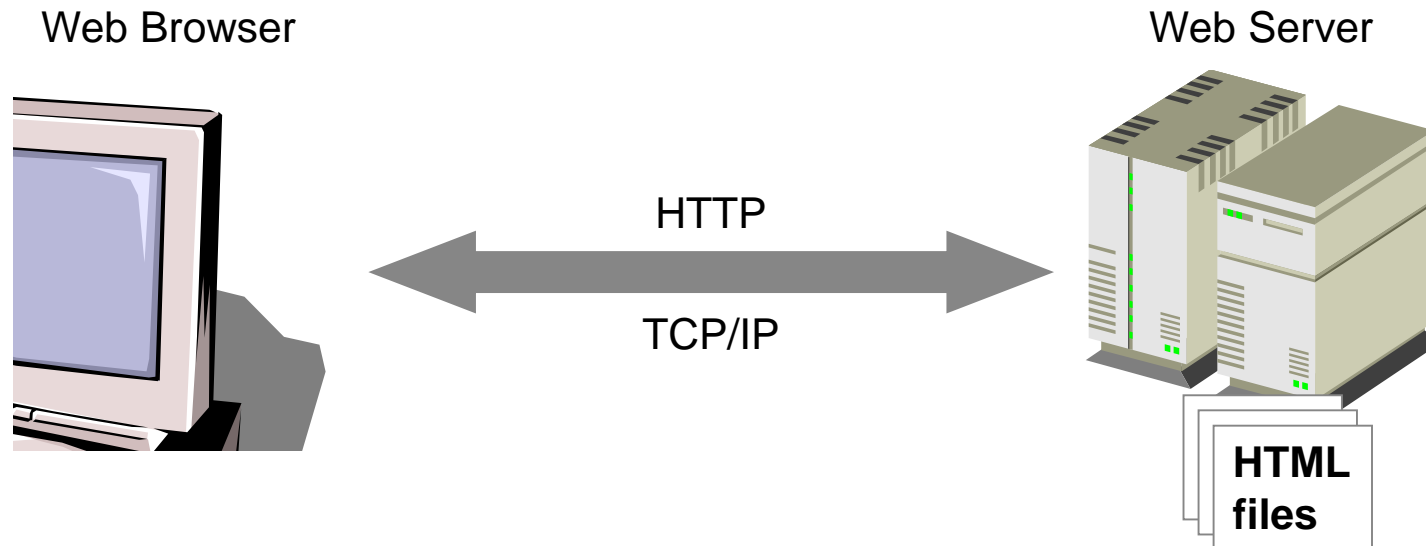
- **Feb. 1993:** First (wide spread) *graphical* interface (Mosaic)
  - Developed at NSCA (National Center for Supercomputing Applications)
  - Introduces embedded images, fill-out forms, ...
- **Mar. 1994:** Mosaic developer *Marc Andreessen* and colleagues leave NCSA to found Mosaic Communications Corp. (later **Netscape**)
- **Oct. 1994:** CERN and the MIT found the WWW Consortium
- **Aug. 1995:** Microsoft Internet Explorer 1.0 released (NCSA Mosaic licensed from Spyglass, Inc.)
- **Mar. 1998:** Netscape Communicator released as open source



# Basic WWW Architecture

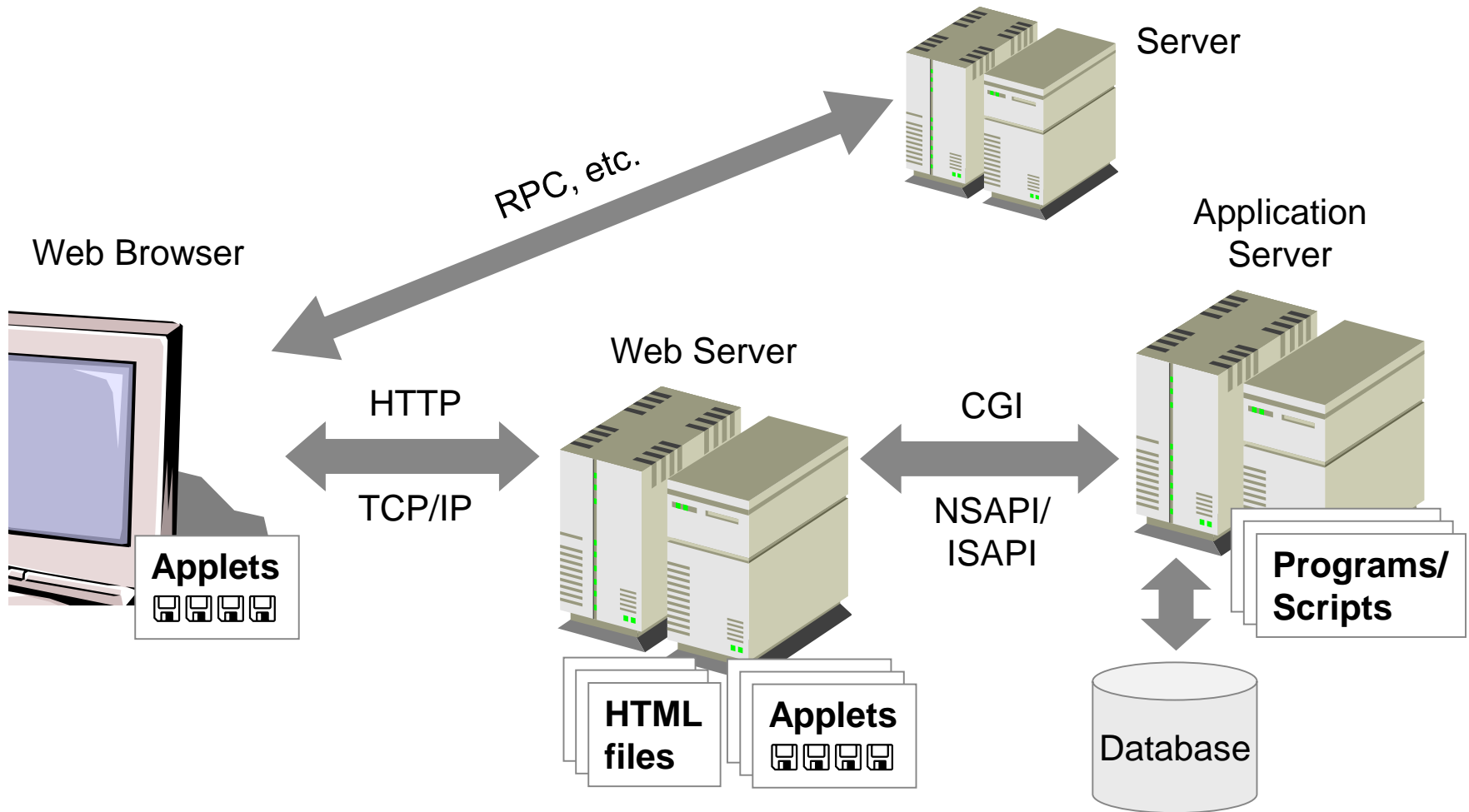
---

- User specifies resource (e.g. HTML page) to view (via URL)
- Web browser loads resource from web server using HTTP



- User sees static, pre-defined data

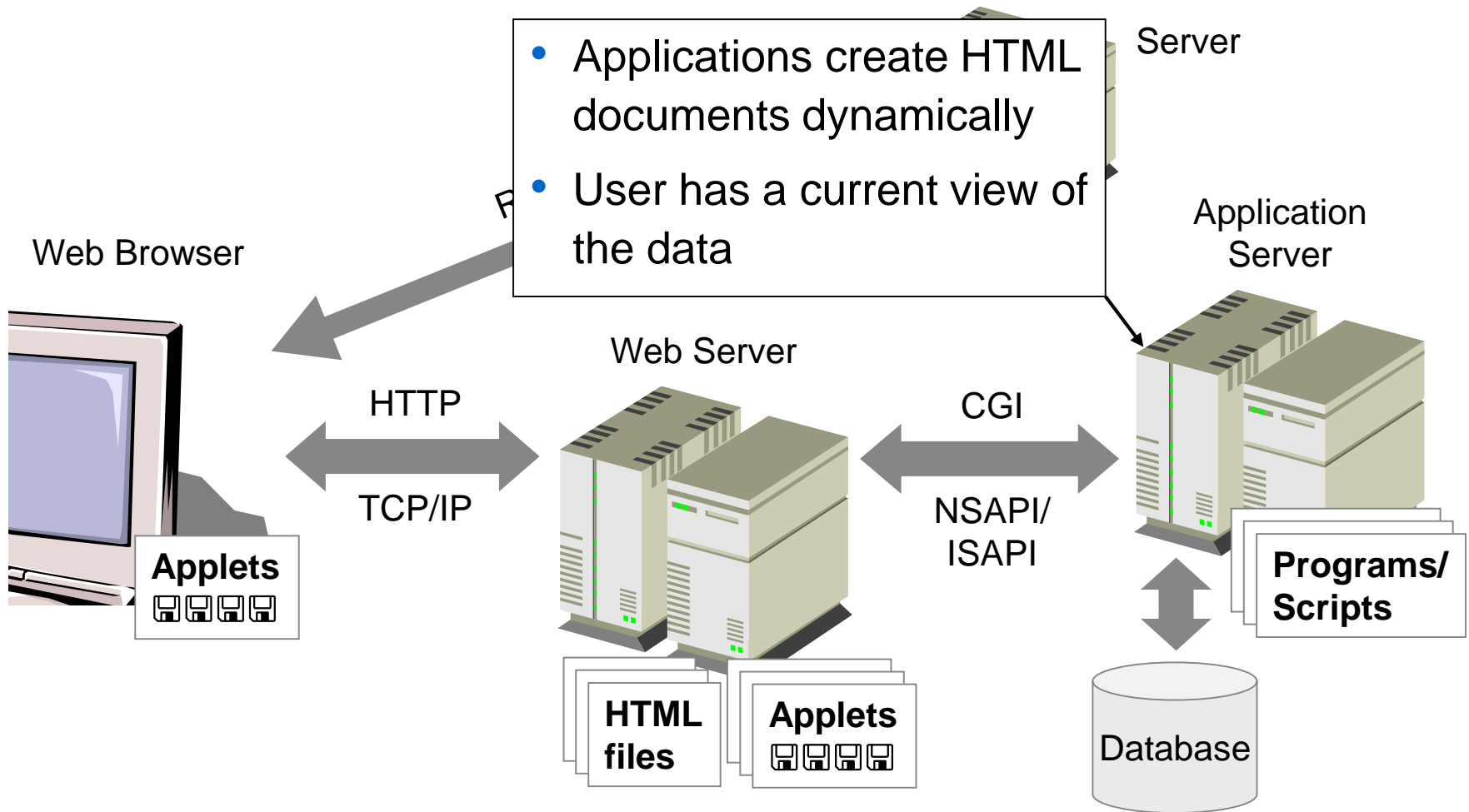
# Extended WWW Architecture



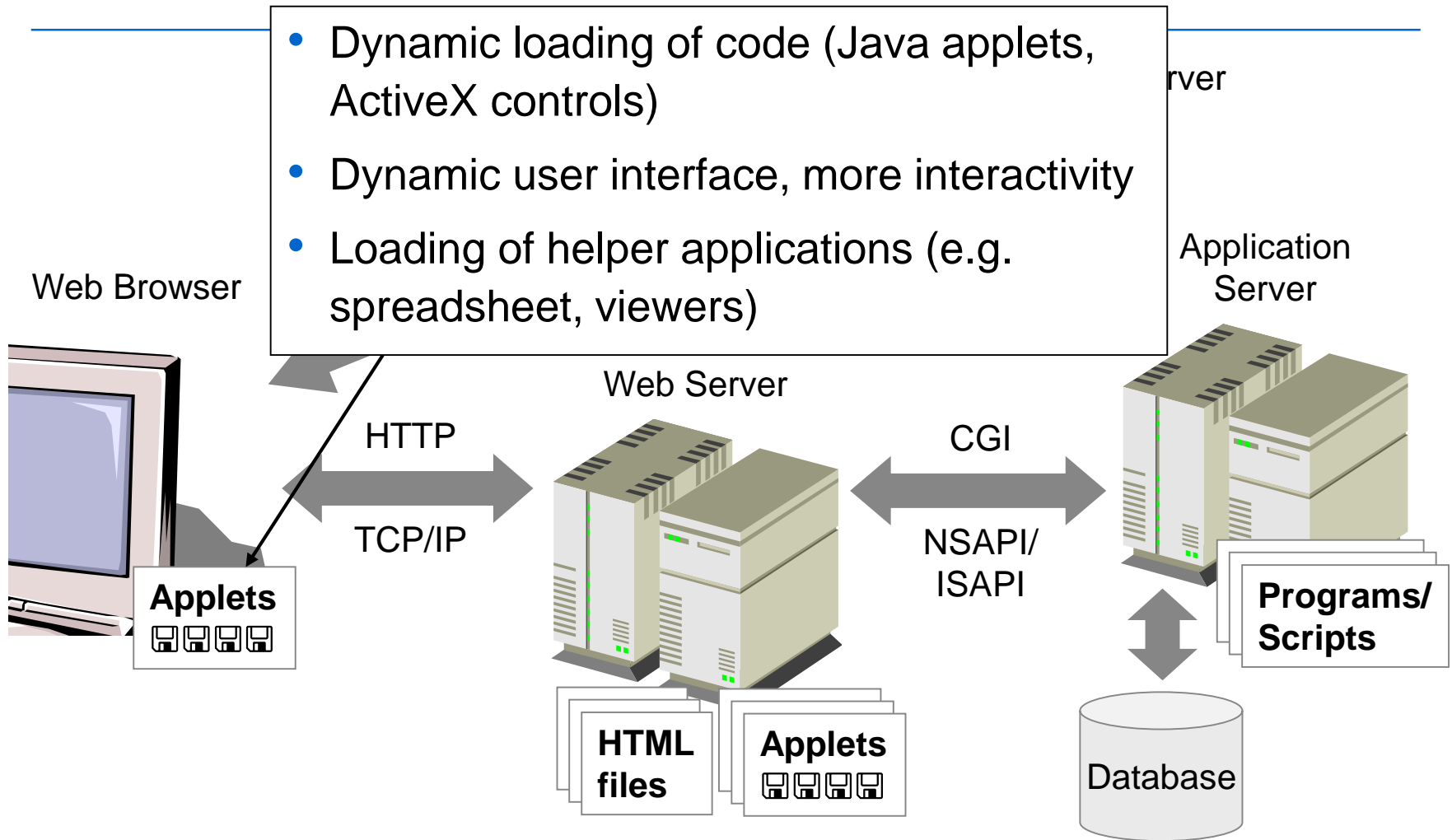
**IPVS**

Research Group  
Distributed Systems

# Extended WWW Architecture

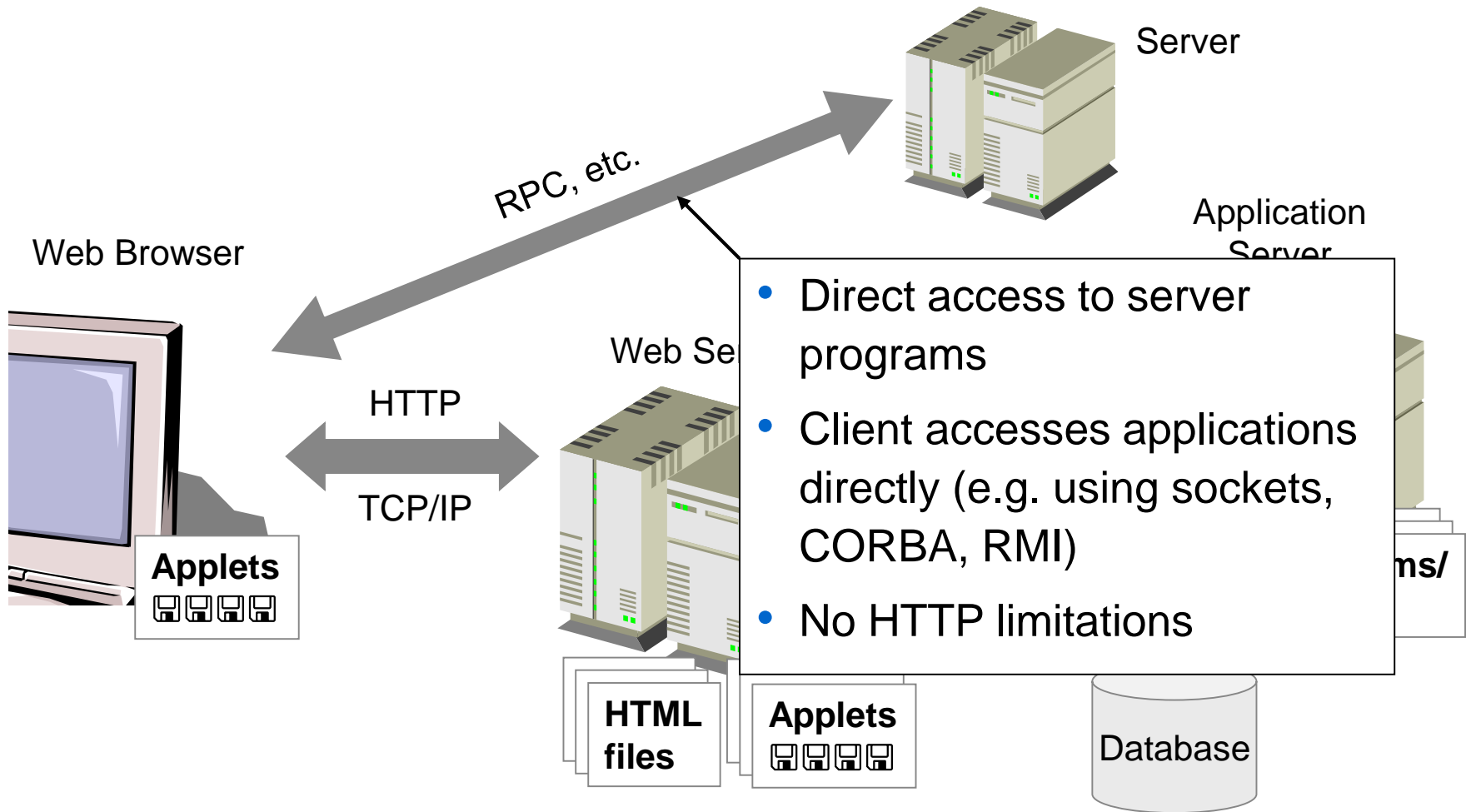


# Extended WWW Architecture





# Extended Web Architecture



**IPVS**

Research Group  
Distributed Systems

# Uniform Resource Identifiers (URIs)

---

- **Uniform Resource Identifier (URI):**

“...a compact string of characters for identifying an abstract or physical resource” (RFC 2396)
- **General form:** **<scheme>:<scheme-specific-part>**
  - **Resources:** e.g. HTML pages, web services, email accounts, ...
  - **Schemes:** e.g. http, ftp, gopher, news, mailto, tip, ...
- **Sub-Classification:**
  - **Uniform Resource Locator (URL):**
    - Identify resources by their primary access mechanism and network location
    - Non-persistent
  - **Uniform Resource Name (URN):**
    - Globally unique and persistent even when resource becomes unavailable
    - Special URN scheme for standardized URN namespace (see RFC 2141)



# Uniform Resource Locators (URLs)

---

## Example: HTTP addressing scheme



- **Hierarchical scheme** → defined root (//) and separators (/)
- **Port**: 80 is the default port for HTTP (can be omitted)
- **Resource details**: parameters, search strings, ...

Other examples:

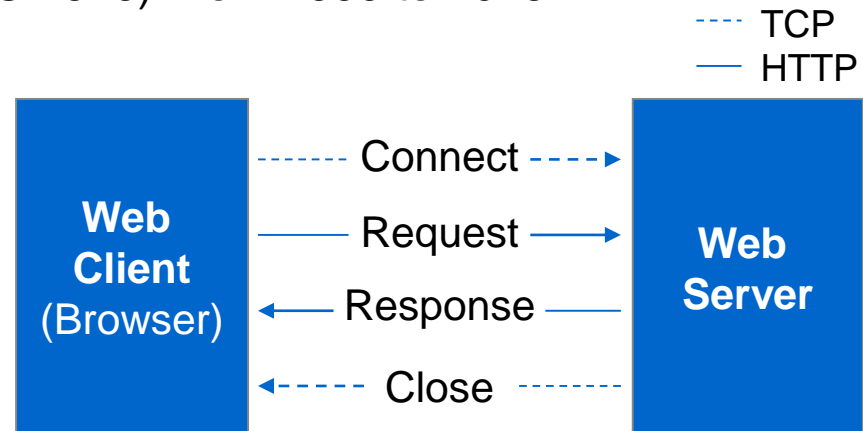
- `http://www.ipvs.uni-stuttgart.de/abteilungen/vs/start`
- `ftp://spinett.informatik.uni-stuttgart.de/path/report.txt`
- `mailto:Kurt.Rothermel@informatik.uni-stuttgart.de`

# HyperText Transfer Protocol (HTTP) (1)

- **Application-layer** protocol to request and transfer resources (e.g. HTML pages) in the Internet
- Based on **TCP/IP**
- **Stateless** & fully **text-based**
- **MIME** used to transfer different resource types (e.g. text/html)
- Current version: **HTTP/2** (based on Google's SPDY protocol)
  - HTTP/1.1 was standard (RFC 2616) from 1999 to 2015

## Protocol (HTTP/1.0):

- Client opens TCP connection to server
- Client transmits request
- Server replies to request
- Server closes connection



# HyperText Transfer Protocol (HTTP) (2)

## Protocol (HTTP/1.1):

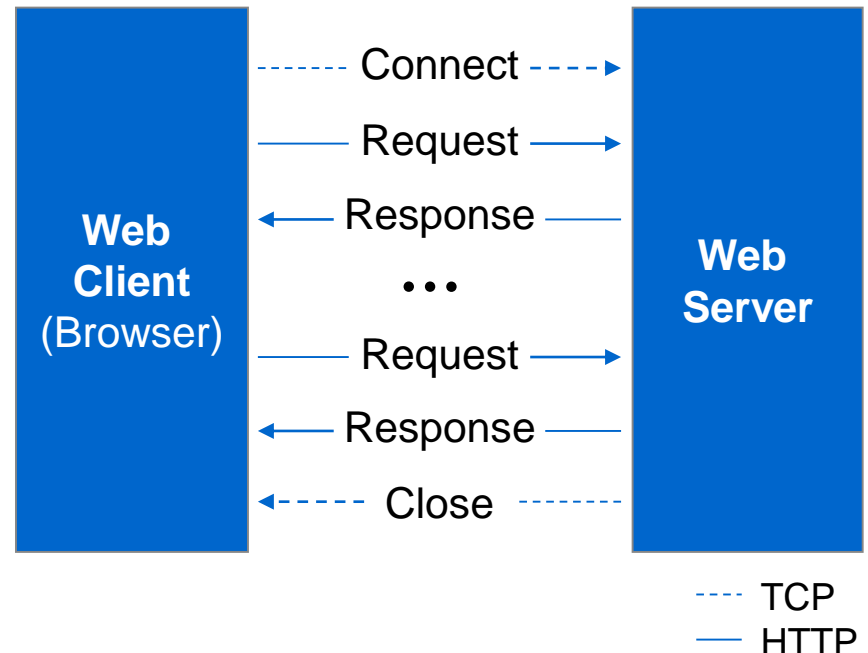
- TCP connection may be left open for further exchanges
  - Timeout
  - *Connection:close* request header field

## Caching to reduce communication

- At client and/or web proxy

## Mapping Server <-> Host:

- In HTTP/1.0: TCP connection determines host
- In HTTP/1.1 multiple (virtual) hosts per server (→ specify virtual host in HTTP requests)



# HTTP/1.1 Requests: Overview

---

- **General request structure:**

request = request-line  
(header-field CRLF)\*  
CRLF  
[message-body]

request-line = method SP request-uri SP HTTP-version CRLF

- **Header fields:**

- Provide additional information, parameterize request behavior
- Examples: *Host*, *If-Modified-Since*, *User-Agent*, *Content-Length*

- **Message body:** contains resource data (MIME), may be omitted

- **HTTP version:** *HTTP/1.1*



# HTTP/1.1 Requests: Method Types

---

Method	Description
<b>OPTIONS</b>	Asks for the available communication options
<b>GET</b>	Request for a web page
<b>HEAD</b>	Request for a web page header
<b>POST</b>	Request to add data to a resource
<b>PUT</b>	Request to store a web page on the server
<b>DELETE</b>	Delete a web page
<b>TRACE</b>	Asks the server to echo the request it received
<b>CONNECT</b>	Reserved for use with proxy that can dynamically switch to being a tunnel

**Note:** additional types defined in intermediate drafts:

- PATCH, LINK, UNLINK, COPY, MOVE, WRAPPED
- Only supported by some servers



# HTTP/1.1 Responses: Overview

---

- **General response structure:**

response = status-line

(header-field CRLF)\*

CRLF

[message-body]

status-line = HTTP-version SP status-code SP reason-phrase CRLF

- **Status code:** 3-digit integer result code specifying response class
- **Reason phrase:** Intended for human users
- **Header fields:** e.g. *Accept-Ranges*, *ETag*



# HTTP/1.1 Responses: Status Codes

---

Status code	Description
1xx	Informational – Request received, continuing process
2xx	Success – The action was successfully received, understood and accepted
3xx	Redirection – Further action must be taken in order to complete the request
4xx	Client Error – The request contains bad syntax or cannot be fulfilled
5xx	Server Error – The server failed to fulfill an apparently valid request

## Examples:

- 100 Continue
- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 500 Internal Server Error



# HTTP/1.1: Caching

---

- **Cache entry** is *fresh* until expiration time is reached
  - *Expires* or *max-age* header fields
  - Both client and server can overwrite this rule (e.g. *no-cache*)
- **Revalidate stale entries** with server (or reload)
  - Based on **modification date** or **entity tag** (*ETag*)
  - **Conditional Requests**:  
header field to specify that request should only be executed if condition true
    - *If-Modified-Since*: <*Time*>
    - *If-None-Match*: <*Etag*>



# HTTP: Example

oboe:~> **telnet** [www.informatik.uni-stuttgart.de](http://www.informatik.uni-stuttgart.de) 80

Trying 129.69.211.2...

Connected to inf.informatik.uni-stuttgart.de.

Escape character is '^['.

**GET** /informatik.html HTTP/1.1

Host: [www.informatik.uni-stuttgart.de](http://www.informatik.uni-stuttgart.de)

HTTP/1.1 200 OK

Date: Mon, 04 Feb 2002 14:09:26 GMT

Server: Apache/1.3.20 (Unix) mod\_perl/1.25

Last-Modified: Tue, 20 Nov 2001 13:58:09 GMT

ETag: "1f4055-3962-3bfa6171"

Accept-Ranges: bytes


Content-Length: 14690


Content-Type: text/html

*[HTML code of the document follows next]*

Connection closed by foreign host.

## Legend

 user input

 server output



**IPVS**

**Research Group**  
**Distributed Systems**

# HyperText Markup Language (HTML)

---

Language for the specification of documents

- Logical organization of documents
- Hyperlinks
- User interaction

Foundation: SGML (Standard Generalized Markup Language)

Uses explicit tags for formatting (“markup”)

`<H1> This is the heading </H1>`

`<P> This starts a new paragraph`

HTML documents consist of

- Header: document information (title, author, etc.)
- Body: actual contents



# Common HTML Tags (1)

---

<code>&lt;HTML&gt; ... &lt;/HTML&gt;</code>	Declares an HTML page.
<code>&lt;HEAD&gt; ... &lt;/HEAD&gt;</code>	Contains the header of an HTML page.
<code>&lt;TITLE&gt; ... &lt;/TITLE&gt;</code>	Defines the title of the page.
<code>&lt;BODY&gt; ... &lt;/BODY&gt;</code>	Declares body of an HTML page.
<code>&lt;Hn&gt; ... &lt;/Hn&gt;</code>	Defines a heading of level <b>n</b> .
<code>&lt;B&gt; ... &lt;/B&gt;</code>	Places ... in <b>bold face</b> .
<code>&lt;I&gt; ... &lt;/I&gt;</code>	Places ... in <i>italic</i> .
<code>&lt;UL&gt; ... &lt;/UL&gt;</code>	Defines the boundaries of an unordered list.
<code>&lt;OL&gt; ... &lt;/OL&gt;</code>	Defines the boundaries of an ordered list.
<code>&lt;LI&gt; ... &lt;/LI&gt;</code>	Defines the boundaries of a list item ( <code>&lt;/LI&gt;</code> is optional)



# Common HTML Tags (2)

---

<code>&lt;BR&gt;</code>	Forces a line break.
<code>&lt;P&gt; ... &lt;/P&gt;</code>	Defines a paragraph (</P> is optional)
<code>&lt;HR&gt;</code>	Inserts a horizontal line.
<code>&lt;PRE&gt; ... &lt;/PRE&gt;</code>	Preformatted text that will not be formatted.
<code>&lt;IMG SRC = "URL"&gt;</code>	Inserts an image from URL.
<code>&lt;A HREF="URL"&gt;...&lt;/A&gt;</code>	Inserts a hyperlink to URL with description
...	
<code>&lt;!-- ... --&gt;</code>	A comment.



# HTML: Hyperlinks

---

## “Anchor” Element

`<A HREF = “http://www.cs.uoregon.edu/staff.html”>Computer Science</A>`

URL of the link target

link text

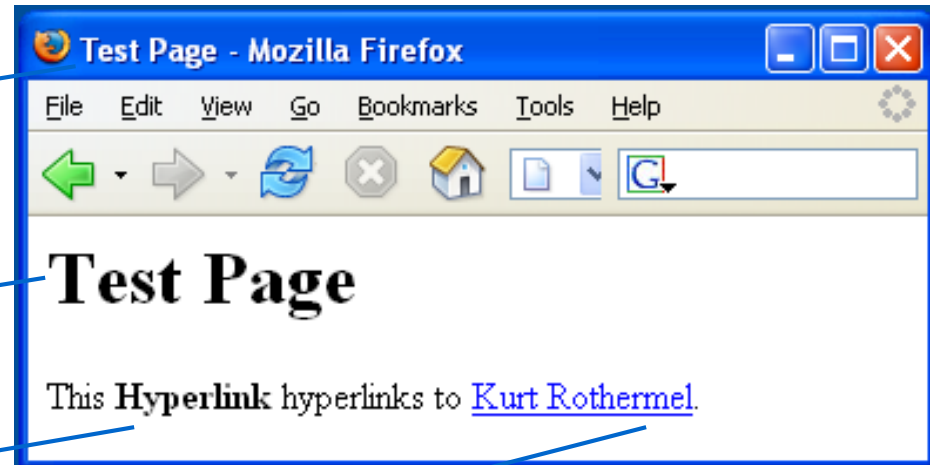
- Link text will be highlighted by a browser
- Instead of text, **images** can be used:

`<A HREF = “http://www.cs.uoregon.edu/staff.html”>  
    <IMG SRC = “cs-logo.jpg” ALIGN = “middle”> </A>`



# HTML: Hyperlink Example

```
<html>
<head>
<title>Test Page</title>
</head>
<body>
<h1>Test Page</h1>
```



This **Hyperlink** hyperlinks to `<a href="http://www.informatik.uni-stuttgart.de/ipvs/vs/de/people/rotherme/"> Kurt Rothermel`

`</a>.`

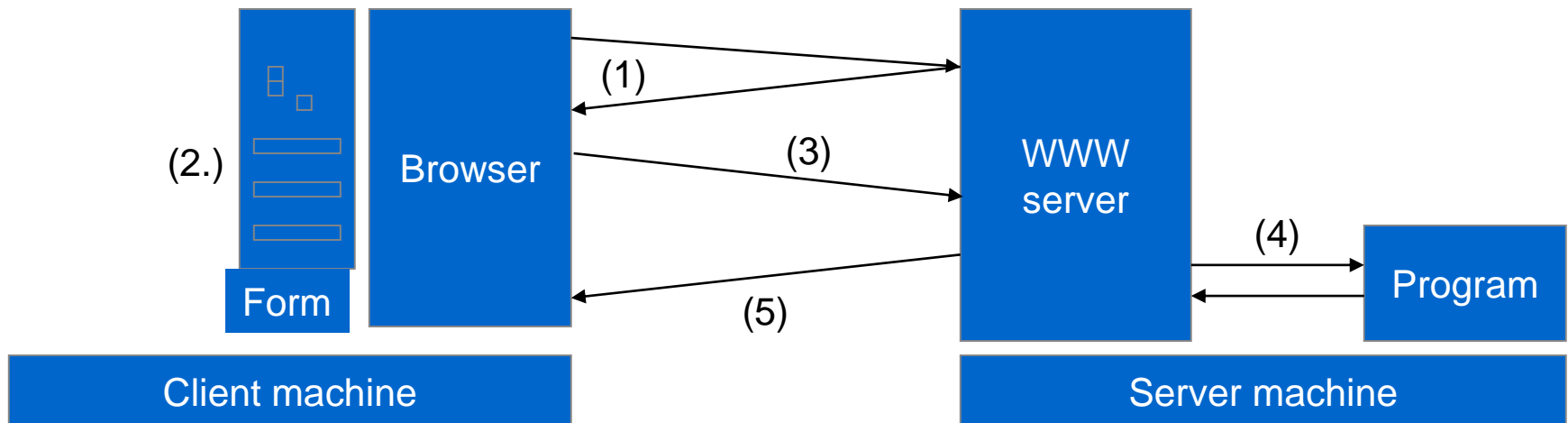
```
</body>
</html>
```





# User Interaction in HTML: Forms

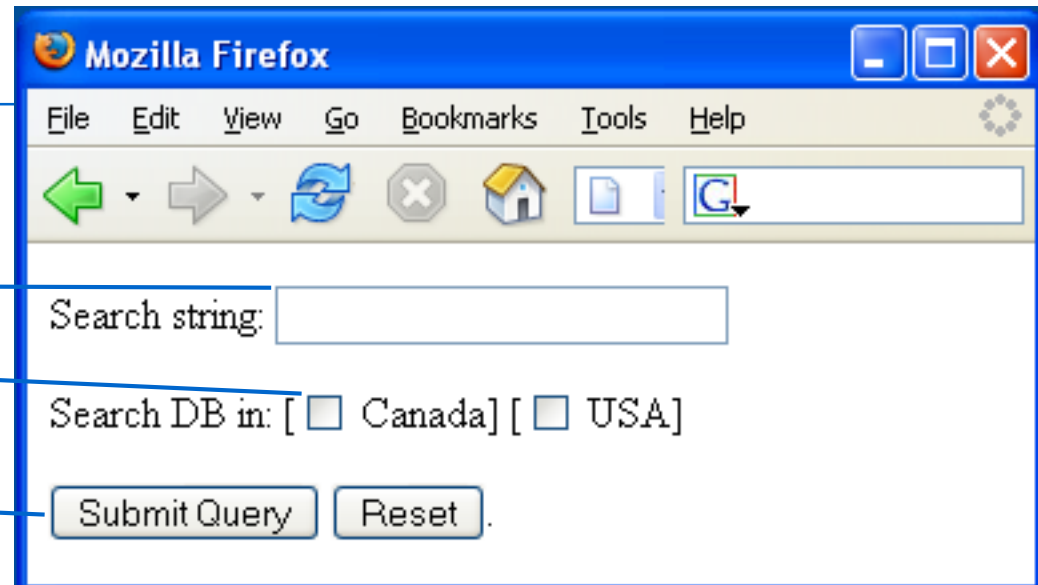
1. Browser gets form from server (HTTP GET request & response)
2. User fills out the form and clicks “submit” button.
3. Browser sends query to server (HTTP GET or POST request)
4. Program generates dynamic content (CGI script, servlet, PHP, etc.)
5. Web server delivers content to browser (HTTP response)



# HTML Forms (1)

Possible interaction elements:

- Text boxes
- Checkboxes
- Buttons  
(submit, reset)
- Menus



Input data is transferred as name/value pairs:

- Example: `search_string=Frank+D%FCrr&servers=USA`
- Encoding:
  - Unchanged: letters, numbers, ".", "-", "~", "\_"
  - Space: "+"
  - Everything else: %FF ("FF" hexadecimal code)



# HTML Forms (2)

```
<FORM ACTION= "http://side.edu/cgi-bin/script" method= "get"
  accept-charset= "ISO-8859-1">
```

<!-- The part in blue above is the URL of the CGI program-->

<!-- The method specifies the HTTP method (GET or POST) for transferring parameters between browser and server and the character set. -->

```
<p>Search string: <INPUT TYPE = "text" NAME = "search-string" SIZE=24>
```

```
<p> Search DB in:
```

```
[<INPUT TYPE= "checkbox" NAME= "servers" VALUE="CANADA"> Canada]
```

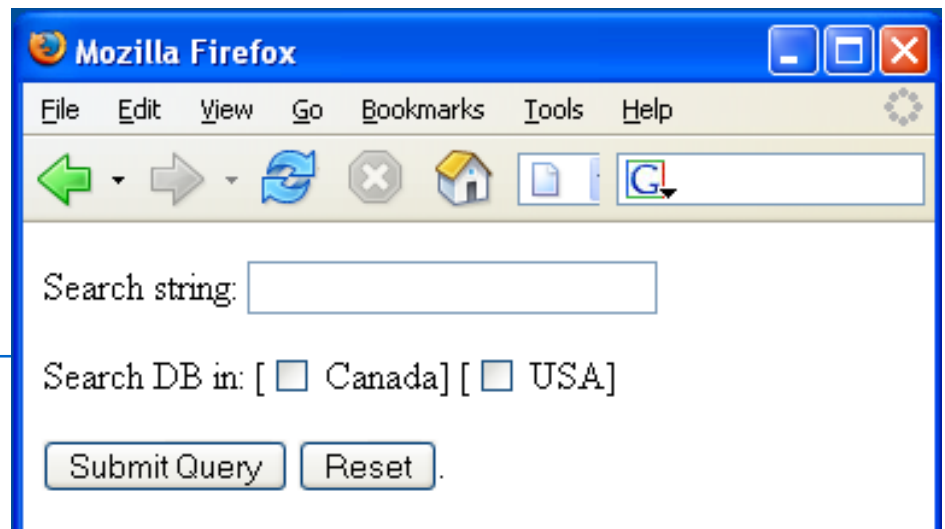
```
[<INPUT TYPE= "checkbox" NAME= "servers" VALUE="USA"> USA]
```

```
<p>
```

```
<INPUT TYPE="submit">
```

```
<INPUT TYPE=reset>.
```

```
</FORM>
```



**IPVS**

**Research Group**  
**Distributed Systems**

# HTML Forms: Example (1)

---

```
<HTML><HEAD>
```

```
  <TITLE> Example of an HTML FORM </TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<H1> Example of an HTML FORM</H1>
```

```
<FORM ACTION="http://side.edu/cgi-bin/submit\_abstract" method="post">
```

```
<p> <STRONG> 1) Send this note to: </STRONG>
```

```
<SELECT NAME="mailto_name">
```

```
  <OPTION SELECTED> Martin Grant
```

```
  <OPTION> Jack Smith
```

```
</SELECT>
```

```
<p> 2) <STRONG> Give your e-mail address: </STRONG>This indicates who sent the letter
```

```
<p> <INPUT TYPE="text" NAME="signature" VALUE="name@internet.address" SIZE=60>
```



# HTML Forms: Example (2)

---

<p> <STRONG> 3) Message Body: </STRONG>

<p>

<TEXTAREA COLS=70 ROWS=8 NAME="message\_body">

Delete this message and type your message into this textbox.

Press the "Send Message" button to send it off. You can press the "Reset" button to reset the form to the original values

</TEXTAREA>

<P>

<INPUT TYPE="submit" VALUE="Send Message"> <INPUT TYPE="reset">

(reset form)

</FORM>

</BODY>

</HTML>



# HTML Forms: The Result



The screenshot shows a Mozilla Firefox browser window with the title "Example of an HTML FORM - Mozilla Firefox". The address bar displays "file:///C:/Dokumente%20und%20Einstellungen". The main content area contains the following HTML form:

## Example of an HTML FORM

1) Send this note to:

2) Give your e-mail address: This indicates who sent the letter

3) Message Body:

Delete this message and type your message into this textbox.  
Press the "Send Message" button to send it off. You can press  
the "Reset" button to reset the form to the original values

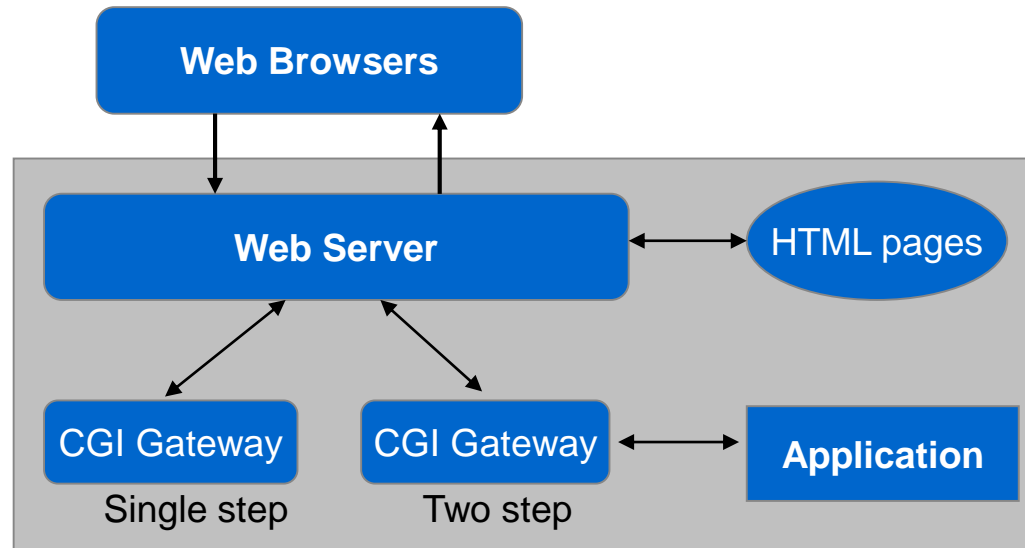
(reset form)

Done



# Common Gateway Interface (CGI)

- Standard for the communication between HTTP server and CGI programs



- Single-step CGI gateway:** Application executed as CGI program (e.g., Calculator)
- Two-step CGI gateway:** CGI program receives name/value pairs and calls external application (e.g. reservation system)



# CGI: Client ↔ Web Server



Call of a CGI program: Link or form (ACTION)

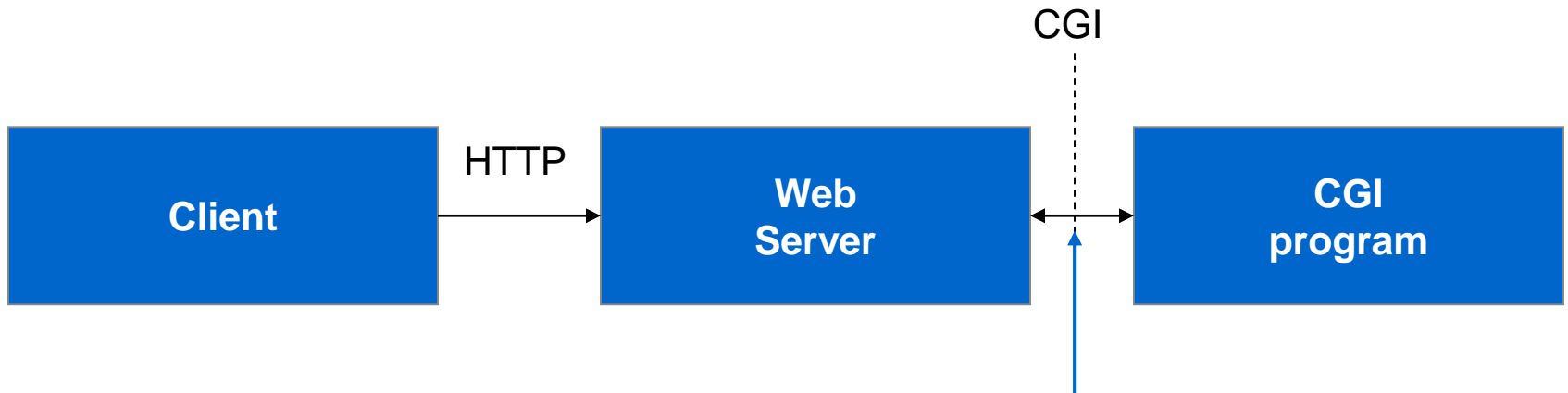
2 ways of passing parameters:

- GET: Part of the URL of the CGI program (URI part of GET request)  
`http://some.site.edu/cgi-bin/search?name=rothermel&dep=CS`
- POST: As part of the data sent with the POST request (in HTTP body)  
`name=rothermel&dep=CS`





# CGI: Web Server ↔ Application



Parameter transfer from web server to gateway

1. Standard input for POST requests
2. Environment variable (QUERY\_STRING) for GET requests

Transfer of the result from gateway to web server:

- Gateway (CGI script) generates HTML document, Image, ...
- Writes content to standard output after specifying CONTENT\_TYPE



# CGI: Example – STDIN Arguments (GET Request)

---

```
#!/usr/bin/perl
# Read body of request from environment variable
$body = $ENV{'QUERY_STRING'};
# Extract parameters from body: name1=value1&name2=value2&...
@parameters = split(/&/, $body);
foreach $parameter (@parameters) {
    ($paramname, $paramvalue) = split(/=/, $parameter); # extract name and value
    $paramvalue =~ tr/+/ /; # replace + signs by space
    # decode encoded special characters (hexadecimal notation)
    $paramvalue =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
    if ($paramname eq "author") {
        # Do something useful (usually you would also use the
        # other parameters for searching, processing, etc.)
        $result = searchindb($paramvalue);
        print "Content-type: text/plain\n\n"; # you may also return HTML here
        print $result;
    }
}
```



# CGI: Example – STDIN Arguments (POST Request)

---

```
#!/usr/bin/perl
# Read body of request from STDIN
read(STDIN, $body, $ENV{'CONTENT_LENGTH'});
# Extract parameters from body: name1=value1&name2=value2&...
@parameters = split(/&/, $body);
foreach $parameter (@parameters) {
    ($paramname, $paramvalue) = split(/=/, $parameter); # extract name and value
    $paramvalue =~ tr/+ / /; # replace + signs by space
    # decode encoded special characters (hexadecimal notation)
    $paramvalue =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
    if ($paramname eq "author") {
        # Do something useful (usually you would also use the
        # other parameters for searching, processing, etc.)
        $result = searchindb($paramvalue);
        print "Content-type: text/plain\n\n"; # you may also return HTML here
        print $result;
    }
}
```



# Java

---

Developed by Sun Microsystems (now Oracle)

## Consists of **platform**

- **Java Virtual Machine** defines virtual CPU
- **Sandbox** principle

## and **language**

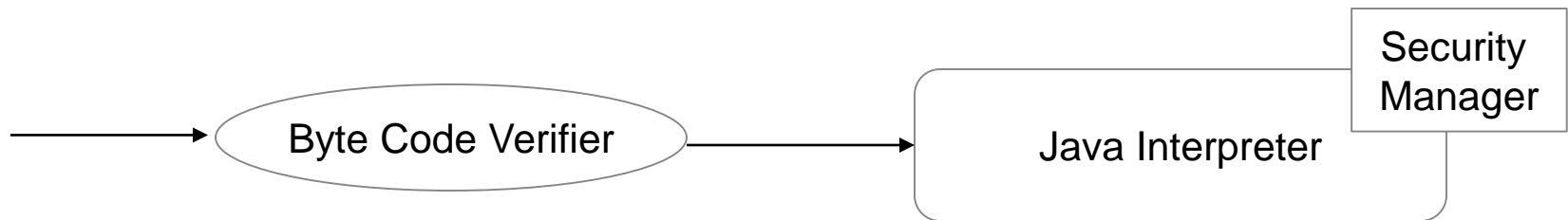
- Object oriented
- Dynamic loading and binding of classes
- Automatic Garbage Collection
- Multithreading on the language level (with monitor concept)



# Java Security Model (1)

---

**Sandbox principle** implemented by the components  
Byte Code Verifier and Security Manager.



**Byte Code Verifier** ensures that an applet conforms to the language specification:

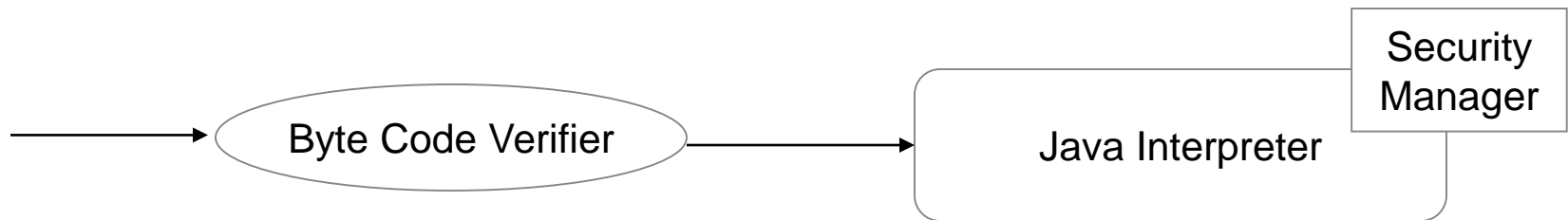
- No operand stack overflows or underflows
- Operand types of all byte code instructions are correct
- No illegal data conversions (integers to pointers)
- All object accesses are legal (*private*, *protected*, *public*)



# Java Security Model (2)

---

**Sandbox principle** implemented by the components  
Byte Code Verifier and Security Manager.

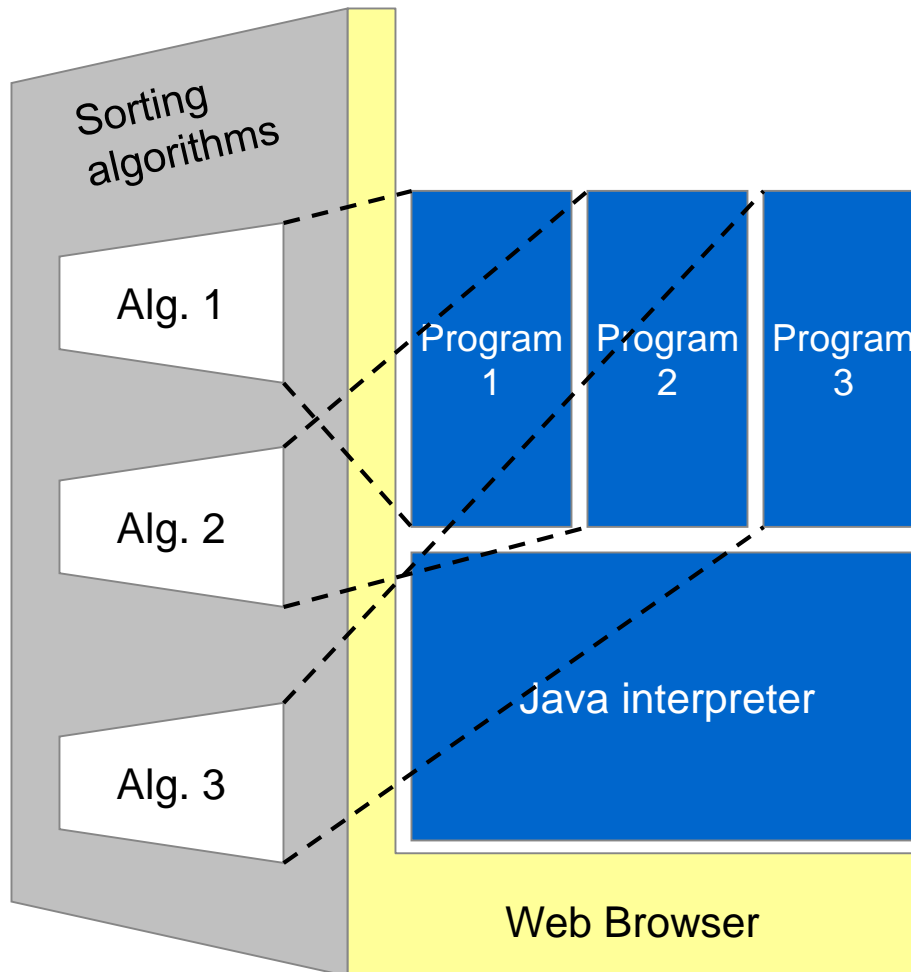


## Security Manager:

- Specifies the boundaries of the “sandbox”
- Called when an applet wants to invoke critical operations
- Checks the operation and denies it if necessary



# Programs on Web Pages: Java Applets



- Special (Java) programs
- Embedded into HTML page
- Fetched from WWW server (just like WWW documents)
- Started on web browsers (applet context)
- One web page can hold several applets
- Applets can communicate with each other
- Restricted access to local resources (e.g. file system) and networking



# Java Applets: Implementation

---

- **Derived from:** Java class “Applet” (AWT) or “JApplet” (Swing)
  - E.g.: public class EchoApp extends java.applet.Applet
- **Applet life cycle** (controlled by browser):
  - **void init()** : Initialize applet  
(each time it's loaded or reloaded)
  - **void start()** : Start applet execution (applet was loaded or user revisits page containing it)
  - **void stop()** : Stop applet execution  
(user leaves page or quits browser)
  - **void destroy()** : Perform final cleanup in preparation for unloading (user quits browser),  
Invoked only once





# Java Applets: Embedding Applets into HTML Pages

---

- **Typical approach:** `<APPLET>`
- **Since HTML 4.0:**  
`<APPLET>` deprecated (but still recommended by Sun/Oracle),  
use `<OBJECT>` instead
- **Examples:**
  - `<APPLET code="ExAppl.class" width="150" height="80" codebase=".">`  
`<PARAM name="snd" value="Hello.au">`  
Your browser is unable to execute Java  
`</APPLET>`
  - `<APPLET code="ExAppl" width="150" height="80" archive="ExAppl.jar">`  
`<PARAM name="snd" value="Hello.au">`  
Your browser is unable to execute Java  
`</APPLET>`
  - `<OBJECT classid="java:ExAppl.class" codetype="application/java-vm"`  
`width="150" height="80" codebase=".">`  
`<PARAM name="snd" value="Hello.au">`  
Your browser is unable to execute Java  
`</OBJECT>`



# Java Applets: Summary

---

## Just-in-time deployment of the client code

- ⇒ Services immediately available
- ⇒ Always the most current version
- ⇒ Platform independent

Executed in a Sandbox → Security

Minimal requirements with respect to the client platform:  
Java-enabled Browser (native or by Java plugin)!

Examples:

- ⇒ Dynamic user interface
- ⇒ Loading of helper applications, e.g. spreadsheet, viewer

# Programs on Web Servers: Java Servlets

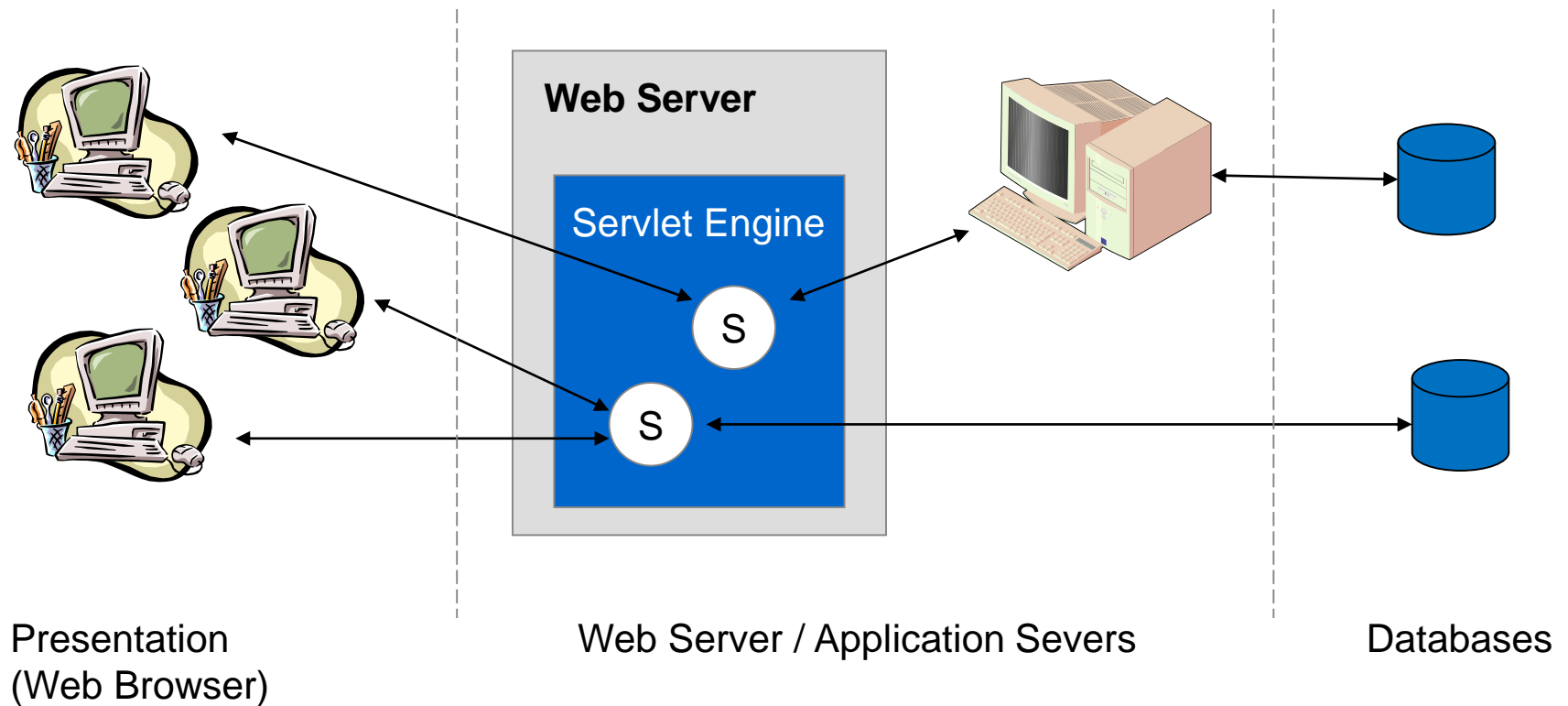
---

- Web-based technology for distributed applications
- Based on Java
- Provides an interface between web server and (legacy) applications
- Roots: Common Gateway Interface and Applets
- **Servlet API:**
  - Defines communication between web server and servlet
  - Allows access to client requests and environment variables
  - Allows to return responses (e.g. HTML)
  - Does *not* cover communication between servlet and application!
- **Servlet Engine:**
  - Executes servlets
  - Often integrated into WWW server



# Architecture of Distributed Applications with Servlets

## 3-Tier Architecture with Servlets



# Servlet API

---

Most important method:

```
void service (ServletRequest request,  
              ServletResponse response);
```

- Implemented by servlet
- Issued by web-server when servlet is to be activated
- `request` object: Allows servlet to access client request
  - Parameters (e.g., from web form)
  - Body of HTTP request
- `response` object: Allows servlet to return response to client
  - Response content: HTML page, images, etc.



# Servlets: A Typical Class Definition

---

```
class NewServlet extends javax.servlet.GenericServlet {  
    // service() must be implemented  
    public void service(ServletRequest req, ServletResponse res) {  
        // read req  
        // execute request  
        // write res  
    }  
}
```



# ServletRequest

---

Most important methods provided by ServletRequest class:

`Enumeration getParameterNames();`

delivers list of parameter names contained in the request

`String[] getParameterValues(String name);`

delivers value(s) for given parameter

`ServletInputStream getInputStream();`

returns input stream for reading body

`getContentType()` and `getContentLength()` return type and length of data



# ServletRequest: Example

---

```
Enumeration e = req.getParameterNames();

while (e.hasMoreElements()) {
    String name = (String) e.nextElement();
    String values[] = req.getParameterValues (name);
    System.out.print ("Name: " +name + " Values: ");
    if (values != null)
        for (int i=0; i<values.length; i++)
            System.out.print (values[i] + " ");
    else
        System.out.print ("(none)");
    System.out.println();
}
... } // End of service()
```





# ServletResponse

---

One possibility: ServletOutputStream

ServletResponse object offers `getOutputStream()`

```
public void service (ServletRequest req, ServletResponse res) {  
    ...  
    res.setContentType(type);  
    ServletOutputStream sout=res.getOutputStream();  
    ...  
    sout.println ("...");  
    ...  
    sout.close();  
    ... }  
}
```



# HTTP Servlets

---

- Class `HttpServlet` is a subclass of `GenericServlet`
- Most important HTTP requests supported: GET and POST
- Instead of `service()` method: `doGet()` and `doPost()` method
- Methods to access HTTP header fields through `HttpServletRequest` object:
  - `getHeader(String field)`:  
returns the value of the specified header field
  - Enumeration `getHeaderNames()`:  
returns all header names in request



# HTTP Servlets: Example

---

```
public class Date extends HttpServlet {  
    public void doGet (HttpServletRequest req,  
                      HttpServletResponse res) {  
        res.setContentType("text/html");  
        PrintWriter out=res.getWriter();  
  
        out.println("<html><body>");  
        out.println("It is: ");  
        out.println((new java.util.Date()).toString());  
        out.println("</body></html>");  
    }  
}
```



# Servlets: Context Information

---

- To maintain **context information** (client/servlet-relationship):  
cookies and sessions
  - **cookies** stored at client
    - created, sent, received by servlets
  - **session data** stored at server
    - created, destroyed by servlet
    - session identified by session id in URL (URL re-writing)



# HTTP Session Class

---

`HttpSession` object manages info about session as key/value pairs

- Implementation transparent to the user
  - Might use cookies or server-side session data (plus URL re-writing)

- **Get/create session object from `HttpServletRequest` object:**

```
HttpSession session = req.getSession(true)
```

- **Set the value of a key of a session:**

```
session.setAttribute("keyname",  
    new String("value"));
```

If session does not exist,  
create new one.

- **Read the value of a key of a session:**

```
String value =  
    (String) session.getAttribute("keyname");
```

- **End session:** `session.invalidate();`



# CGI vs Servlets

---

- Similar communication mechanism
- Servlets based on Java
- CGI language independent
- Servlets are executed in engine
- With CGI: one process per request



# Java Server Pages (JSP)

---

- Based on servlet technology
- HTML documents with embedded java code
- Document is automatically compiled into servlet when accessed by browser



# JSP: Example (1)

---

## Intended result:

It is: 2005-11-01, 17:03:02

## HTML file:

```
<html><body>  
  It is:  
  <%out.println((new java.util.Date()).toString());%>  
</body></html>
```

When accessed: all elements that are not between `<%` and `%>`  
become parameters of `out.println()` method calls of Servlet



## JSP: Example (2): Created Servlet

Can also be accessed from JSP page.

```
public class Date extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response) {  
        res.setContentType("text/html");  
        PrintWriter out=res.getWriter();  
  
        out.println("<html><body>");  
        out.println("It is: ");  
        out.println((new java.util.Date()).toString());  
        out.println("</body></html>");  
    }  
}
```



# Servlets vs JSP

---

Formatting predominant part → JSP

Computing predominant part → Servlets



# Client-side Scripting

---

Client-side scripting (JavaScript, VBScript, ...)

- Script embedded in HTML page, interpreted by script engine of browser
- Event handlers attached to HTML tags (load, mouse over, mouse click, ...)
- Library functions to perform common tasks (change page, open window, ...)
- Document object model to manipulate page contents dynamically (DHTML)
- Limited access to client resources (security)
- Access to browser plug-ins (e.g., XML parser)

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!-- function square() {
    var result = document.aform.ainput.value
        * document.aform.ainput.value;
    alert("Square is" + result);
} //-->
</script>
</head>
<body>
<form name="aform" action="">
<input type="text" name="ainput" size="3">
<input type="button" value="Compute square"
    onclick="square()">
</form>
</body>
</html>
```



# Server-Side Scripting

---

- Server-side scripting
  - **PHP**: Widely used server-side scripting language
    - Session management
    - Database access
    - Similar to JSP but interpreted
  - **Active Server Pages**
    - Microsoft's Server Pages scripted with VBScript, Perl, or Jscript
    - ASP.NET for use with .NET Framework
    - Embedded in web server

```
<html>
<head>
  <title>PHP-Test</title>
</head>
<body>
  <?php echo '<p>Hello World</p>'; ?>
</body>
</html>
```

```
<html>
<head>
  <title>ASP-Test</title>
</head>
<body>
  <% Dim strText
    strText = "Hello World"
    Response.Write(strText) %>
</body>
</html>
```



# Web 2.0

---

- Buzzword for new **web usage patterns**
  - Increasing number of **users become authors** (Wiki, Blogs, ...)
  - **Collaboration** between many users **via www** (communities)
- Often used to describe a **combination of technologies**
  - HTTP to deliver a page containing JavaScript
  - **AJAX (Asynchronous JavaScript and XML)**
    - JavaScript generates XML documents as requests
    - XML requests sent to web server via browser plug-in over HTTP
    - Client receives XML response asynchronously from web server
  - JavaScript uses XML document contents to manipulate HTML page dynamically (DHTML)



# Summary

---

- **URI** (URL/URN): identifying resources in the WWW
- **HTTP**: Purely text-based transfer protocol
- **HTML**: Pages and forms
- **CGI** as an interface to call programs on the web server
- **Java Applets**
  - Java programs that run on the client web browser
  - platform independence, “sandbox” security model
- **Java Servlets and JSP**
  - API for execution of Java programs on web server
- **Scripting**: scripts interpreted by browser or server

