

# TEMA 4: APLICACIONES DE INTERNET

---

# WWW (World Wide Web o Web)

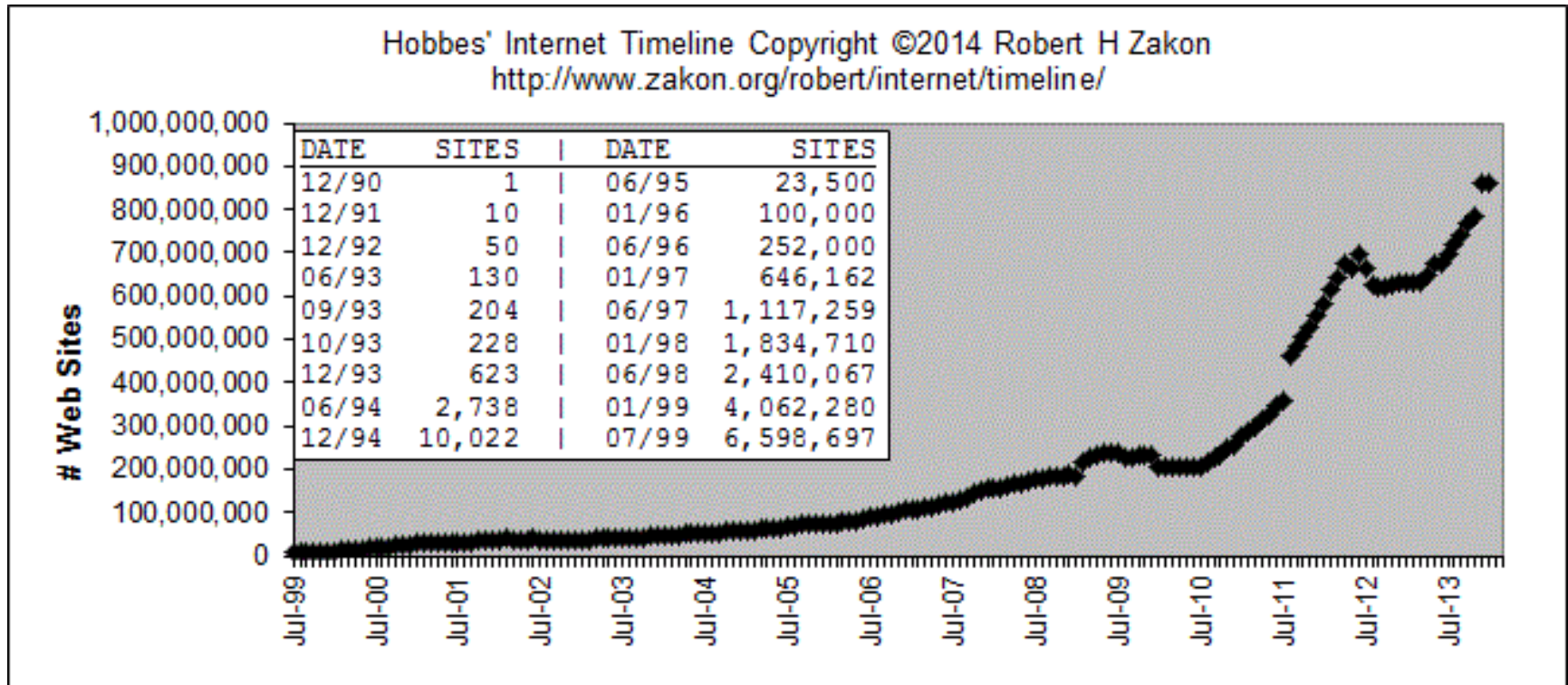
- Técnicamente la Web es un sistema distribuido de servidores y clientes HTTP, más conocidos como servidores y navegadores web.

La World Wide Web (más concretamente, el sistema de hipertexto universal) se inició con [Tim Berners-Lee](#) a finales de 1990 en el CERN, the European Particle Physics Laboratory en Ginebra, Suiza.



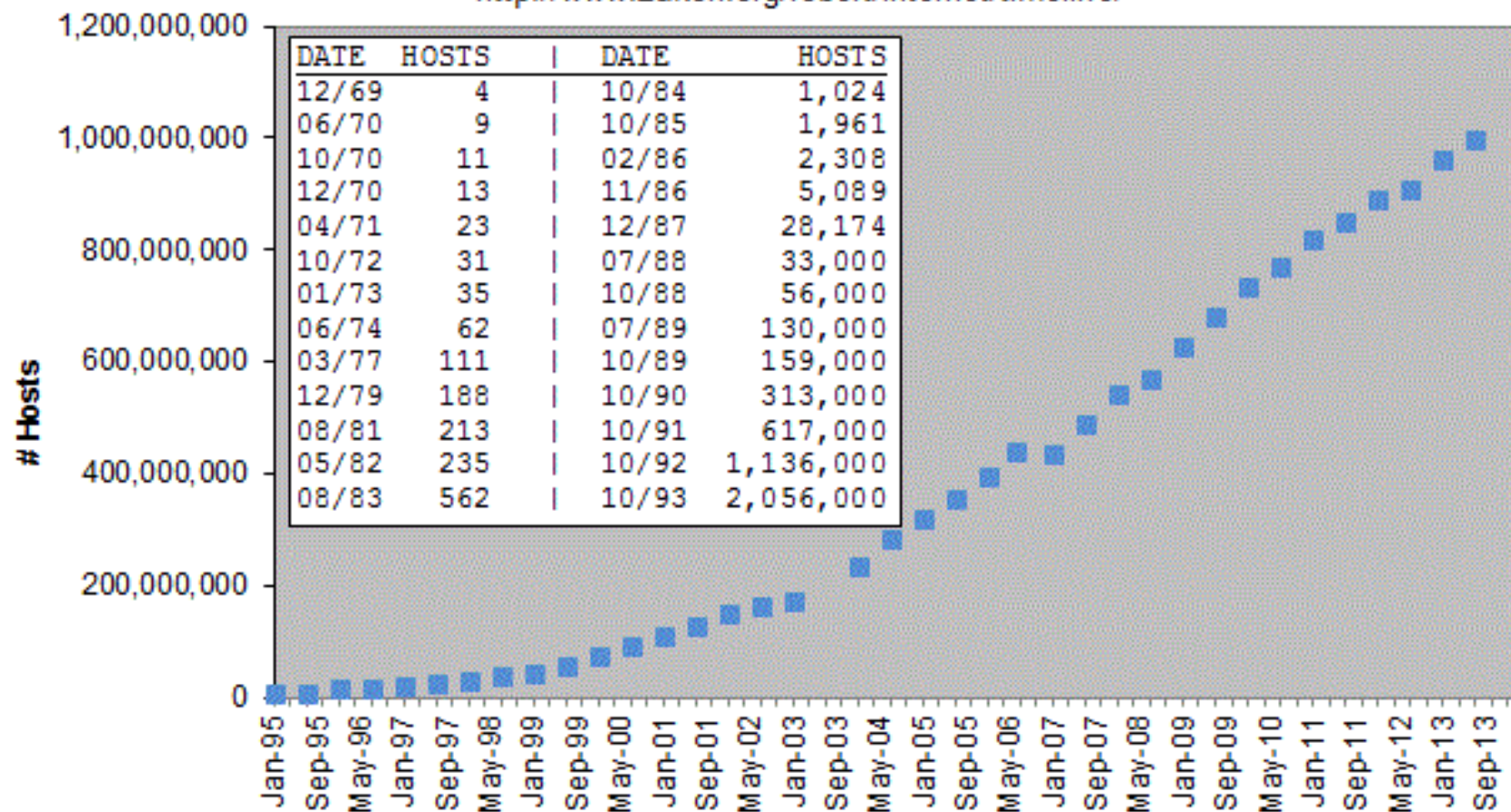
# WWW (World Wide Web)

Gráfica reciente de su evolución.



Sites = Number of web servers (one host may have multiple sites by using different domains or port numbers)

Hobbes' Internet Timeline Copyright ©2014 Robert H Zakon  
<http://www.zakon.org/robert/internet/timeline/>



# WWW (World Wide Web) o Web

En la World Wide Web se combinan tres tecnologías bien establecidas:

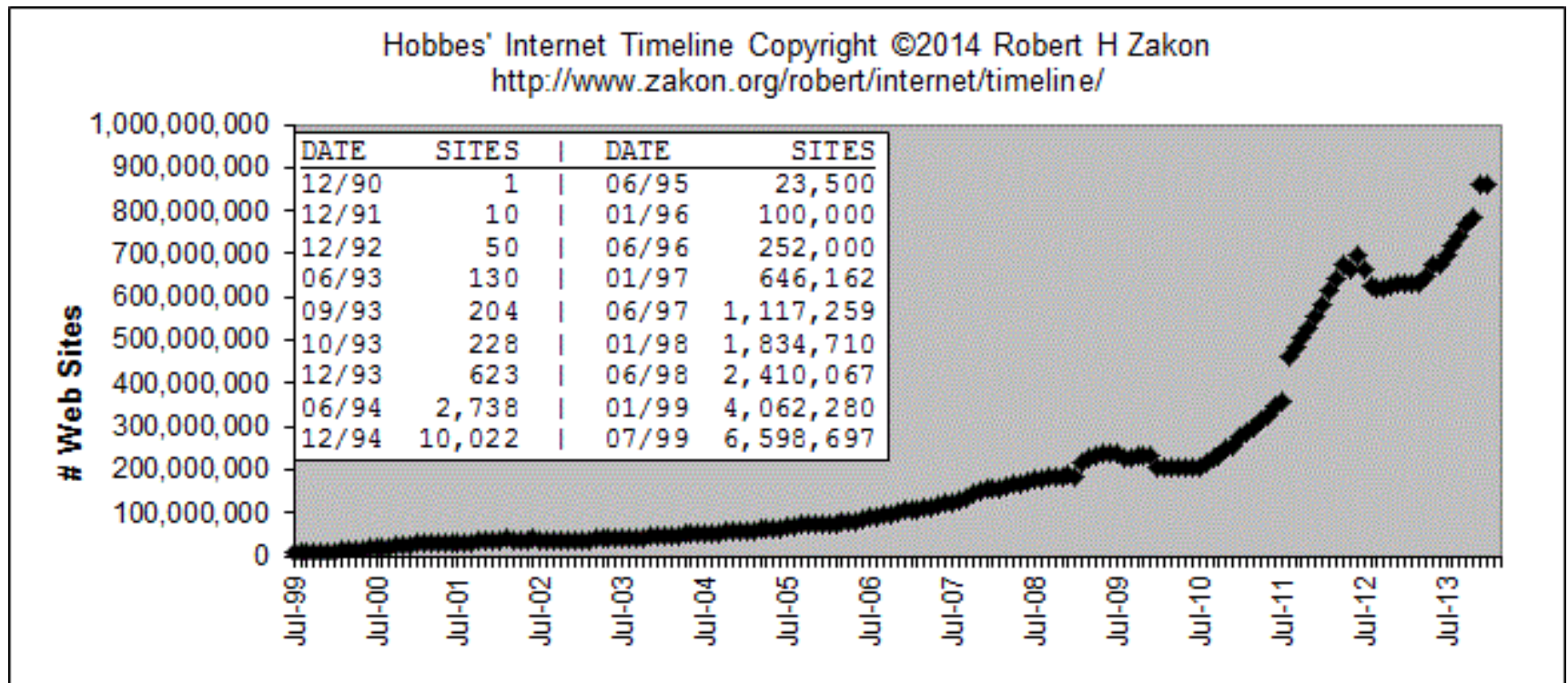
- *Documentos de Hipertexto.*
- *Recuperación de la información basada en la red.*
- *El lenguaje Standard Generalized Markup Language (SGML),*

Se basa en un protocolo denominado *HyperText Transfer Protocol (HTTP)*

Un servidor web es un servidor orientado a conexión que implementa HTTP, y que, por defecto, se ejecuta en el puerto 80.

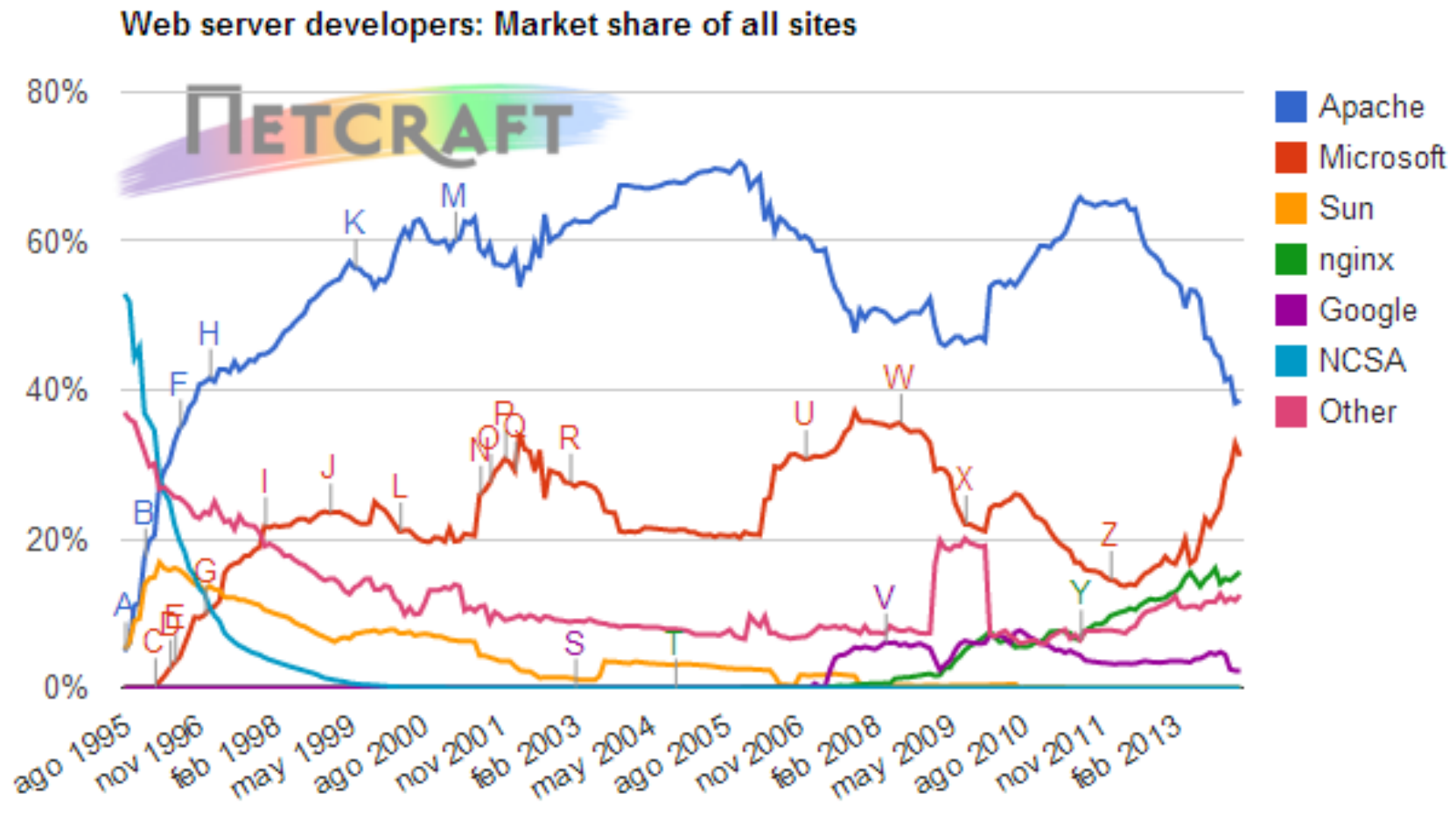


# Server Web



# Servidor Web:

Gran variedad: Apache, IIS, Sun, Google

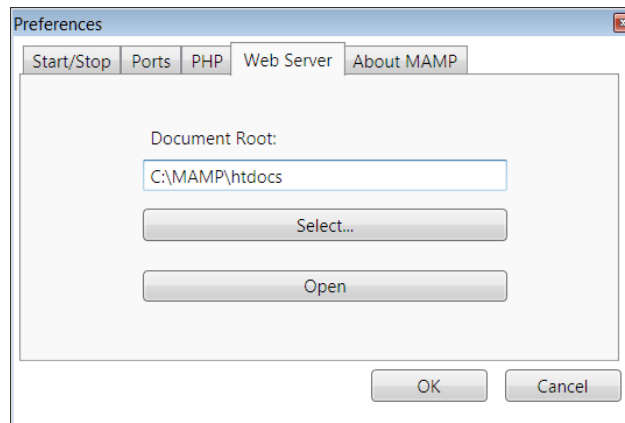
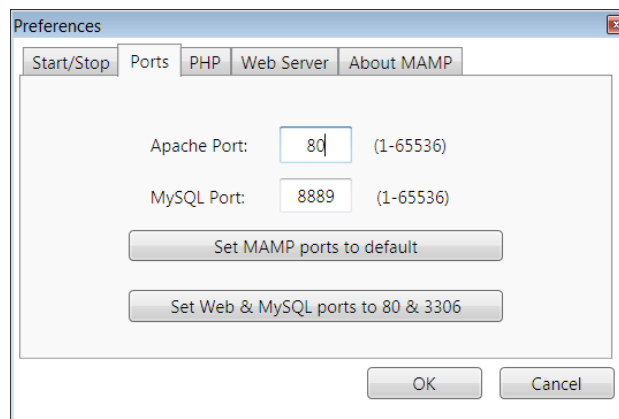
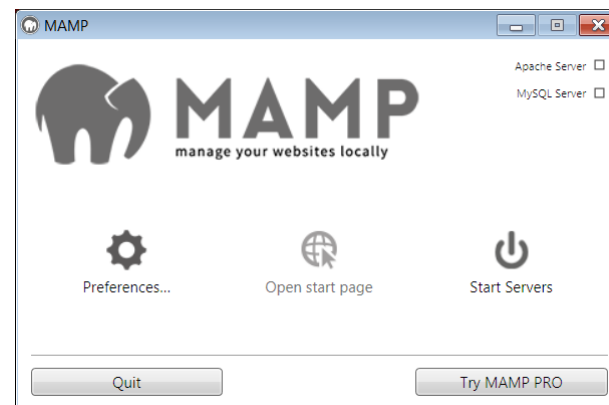


# Servidor Web:

Se muestra MAMP, una colección de programas y librerías para gestionar un sitio web basado en Apache.

Apache 2.2.27  
 MySQL 5.6.17  
 PHP 5.4.1, ..., 5.6.0  
 OpenSSL 1.0.1g  
 APC 3.1.13  
 APCu 4.0.4 & 4.0.6  
 eAccelerator 1.0  
 XCache 3.0.4 & 3.1.0  
 OPCache 7.0.4  
 phpMyAdmin 4.2.7  
 Python 2.7.6  
 mod\_wsgi 3.4.0  
 ImageMagick 6.8.9-1  
 Imagick 3.1.2

Perl 5.16.1  
 mod\_perl 2.0.8  
 SQLiteManager 1.2.4  
 phpLiteAdmin 1.9.4.1  
 Freetype 2.4.10  
 curl 7.36.0  
 libpng 1.5.18  
 gd 2.1.0  
 zlib 1.2.7.3  
 libxml2 2.9.1  
 gettext 0.18.1.1  
 iconv 1.14  
 mcrypt 2.5.8





# HTML (Hyper Text Markup Language)

```
<HTML>
<HEAD>
<TITLE>A Sample Web Page</TITLE>
</HEAD>
<HR>
<BODY>
<center>
<H1>My Home Page</H1>
<IMG SRC="/images/myPhoto.gif">
<b>Welcome to Kelly's page!</b>
<p>
A list of hyperlinks follows.
<a href="/doc/myResume.html"> My resume</a>.
<p>
<a href="http://www.someUniversity.edu/">My university<a>
</center>
<HR>
</BODY>
</HTML>
```

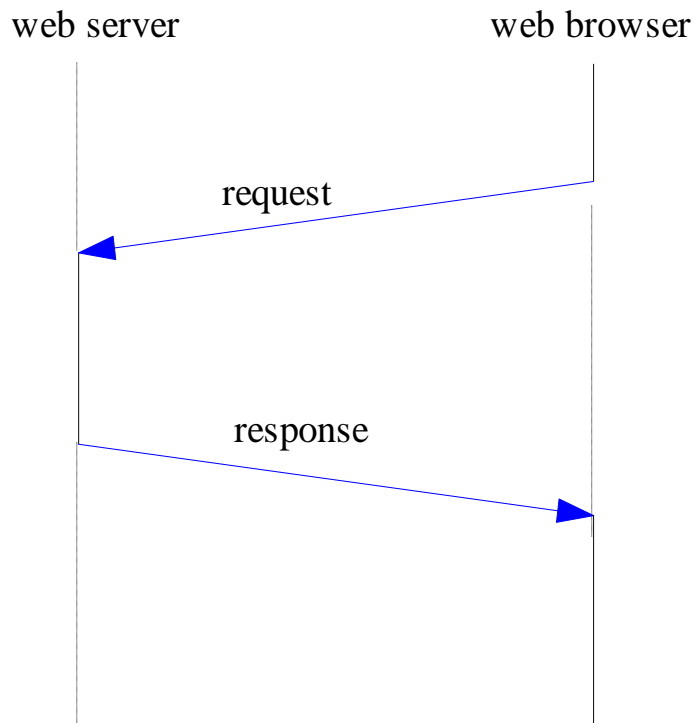
# XML (Extensible Markup Language)

HTML se ocupa de aspectos de representación de la información, XML de cómo identificar semánticamente la información.

```
<message>  
  <to>you@yourAddress.com</to>  
  <from>me@myAddress.com</from>  
  <subject>This is a message</subject>  
  <text>  
    Hello world!  
  </text>  
</message>
```

# HTTP (HyperText Transfer Protocol)

- La versión de HTTP más utilizada es HTTP/1.0, propuesta por Tim Berners Lee.
- Actualmente también se dispone de una versión mejorada conocida como HTTP/1.1.
- **HTTP** es un protocolo basado en **texto**, **sin estado**, de **petición-respuesta** y **orientado a conexión**.



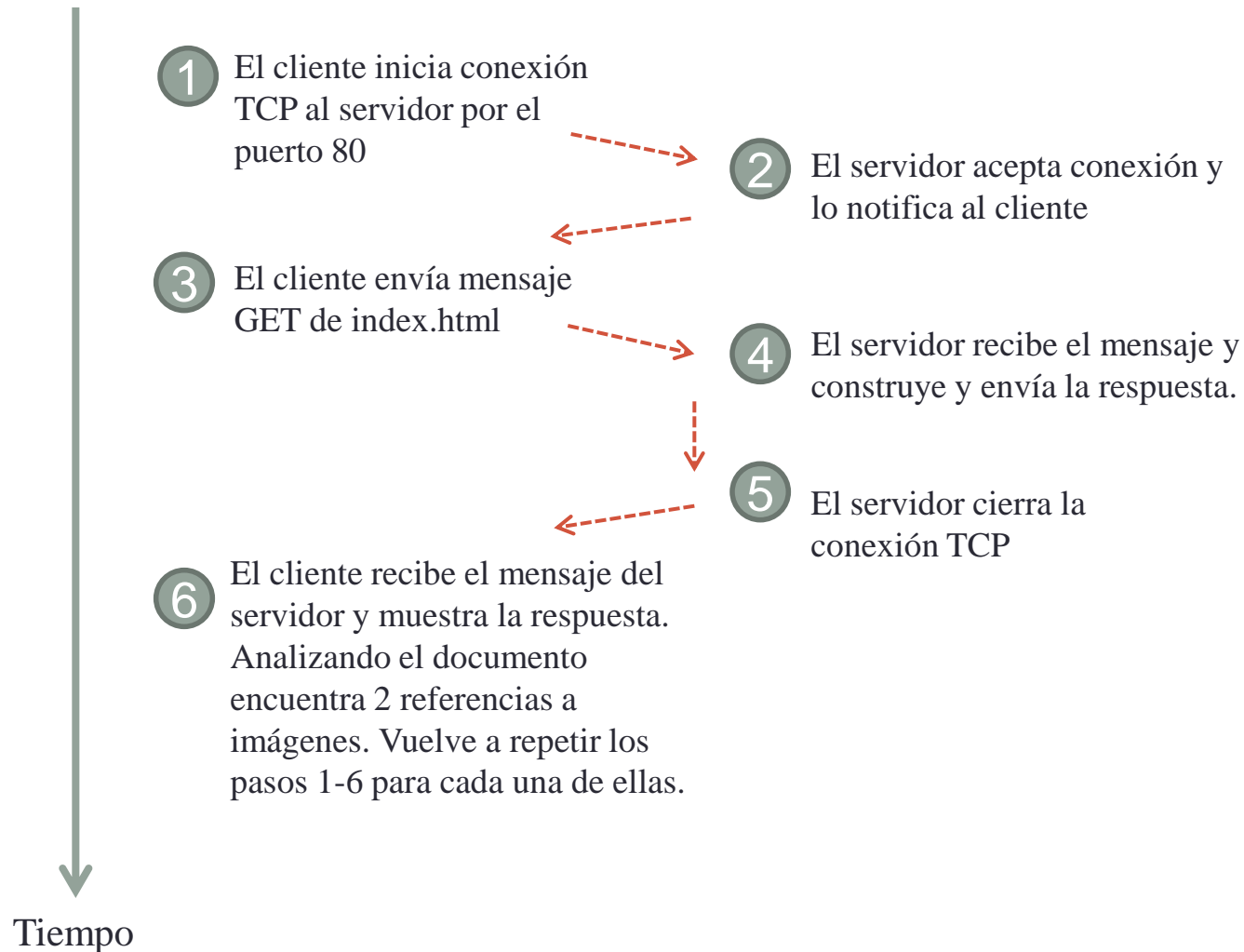
request is a message in 3 parts:

- <command> <document address> <HTTP version>
- an optional header
- optional data for CGI data using post method

response is a message consisting of 3 parts:

- a status line of the format <protocol><status code><description>
- header information, which may span several lines;
- the document itself.

# HTTP (bajo bambalinas)



# Una sesión ejemplo de HTTP

GET / HTTP/1.0 ← *HTTP Request*

HTTP/1.1 200 OK ← *HTTP response status line*  
Date: Wed, 29 Oct 2014 17:15:18 GMT ← *HTTP response header*  
Server: Apache/2.2.29 (Unix) mod\_fastcgi/2.4.6 mod\_wsgi/3.4 Python/2.7.8 PHP/5.6.2 mod\_ssl/2.2.29 OpenSSL/0.9.8za DAV/2 mod\_perl/2.0.8 Perl/v5.20.0  
Last-Modified: Wed, 29 Oct 2014 17:13:13 GMT  
ETag: "1dd1e-e27-39e3492a"  
Accept-Ranges: bytes  
Content-Length: 989  
Connection: Keep-Alive  
Content-Type: text/html

<HTML> ← *document content*  
<HEAD>  
<TITLE> Especialista en cervezas  
</TITLE>  
</HEAD>  
<BODY bgcolor=#ffffff>  
...



Algunos de los métodos HTTP más utilizados son:

**GET, HEAD, POST, PUT, DELETE.**

Tres tipos de **GET**:

**GET parcial**: Range: bytes= ... *Baja partes del recurso*

**GET condicional**: if-Modified\_Since, If-Match

*Baja el recurso si se cumplen ciertas condiciones*

**GET**

Algunas de las parejas más habituales clave-valor que pueden aparecer en una cabecera son:

**Accept**: tipos de contenido aceptables

**User-Agent**: tipo de buscador (cliente)

**Connection**: “Keep-Alive” mantener la sesión.

**Host**: nombre del host del servidor.

# Ejemplos de una petición HTTP

Ejemplo 1:

```
GET / HTTP/1.1  
<blank line>
```

Ejemplo 2:

```
HEAD / HTTP/1.1  
Accept: */*  
Connection: Keep-Alive  
Host: somehost.com  
User-Agent: Generic  
<blank line>
```

Ejemplo 3:

```
Post /servlet/myServer.servlet http/1.0  
Accept: */*  
Connection: Keep-Alive  
Host: somehost.com  
User-Agent: Generic  
<blank line>  
Name=donald&enail=donald@someU.edu
```

# La respuesta del servidor HTTP

La línea de estado es de la forma:

```
<protocol><sp><status-code><sp><description>\r\n
```

Los códigos de estado son:

100-199	Informational
200-299	Client request successful
300-399	Client request redirected
400-499	Client request incomplete
500-599	Server errors

**Ejemplo 1:**

```
HTTP/1.0 200 OK
```

**Ejemplo 2:**

```
HTTP/1.1 404 NOT FOUND
```

# Códigos de estado

**1xx:** Mensaje de información

**2xx:** Éxito en la transacción:

200 OK

201 Created

202 Accepted

204 No Content

**3xx:** Redirección de la  
petición del cliente

300 Multiple Choice

301 Moved Permanently

302 Found

304 Not Modified

**4xx:** Error del cliente

400 Bad Request

401 Unauthorized

403 Forbidden

404 Not Found

**5xx:** Error del servidor

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

## Hay dos tipos de líneas de cabecera:

1. **Respuesta:** trata de aspectos de la comunicación, como por ejemplo, `Age: seconds`, `Location: URI`, `Retry-After: date|seconds`, `Server: string`, `WWW-Authenticate: scheme realm`.
2. **Entidad:** trata de aspectos del contenido de la respuesta, como por ejemplo, `Content-Encoding`, `Content-Length`, `Content-Type: type/subtype (MIME)`, `Expires: date`, `Last-Modified: date`.



El cuerpo de la respuesta sigue a la cabecera y a una línea en blanco.

HTTP/1.1 200 OK

Date: Sat, 15 Sep 2001 06:55:30 GMT

Server: Apache/1.3.9 (Unix) ApacheJServ/1.0

Last-Modified: Mon, 30 Apr 2001 23:02:36 GMT

ETag: "5b381-ec-3aedef0c"

Accept-Ranges: bytes

Content-Length: 236

Connection: close

Content-Type: text/html

<html>

<head>

<title>My web page </title>

</head>

<body>

Hello world!

</BODY></HTML>

# Content Type – El protocolo MIME (Multipurpose Internet Mail Extension)

Una pequeña muestra de los tipo/subtipo permitidos en MIME:

Type	Subtype
text	plain, rich text, html, tab-separated-values, xml
message	Email, news
Application	Octet-stream (can be used for transferring Java .class files, for example), Adobe-postscript, Mac-binhex40, xml
Image	jpeg, gif
audio	basic,midi,mp3
video	mpeg, quicktime

# Implementación sencilla de un cliente HTTP

```
InetAddress host = InetAddress.getByName(args[0]);
int port = Integer.parseInt(args[1]);
String fileName = args[2].trim();
String request = "GET " + fileName + " HTTP/1.0\n\n";
MyStreamSocket mySocket = new MyStreamSocket(host, port);
mySocket.sendMessage(request);
// recibir el mensaje de respuesta
String response = mySocket.receiveMessage();
// escribe línea a línea
while (response != null) {
    System.out.println(response);
    response = mySocket.receiveMessage();
}
```

## La clase URL de java

Hay una clase URL específica para recuperar datos de un objeto web identificado por una URI.

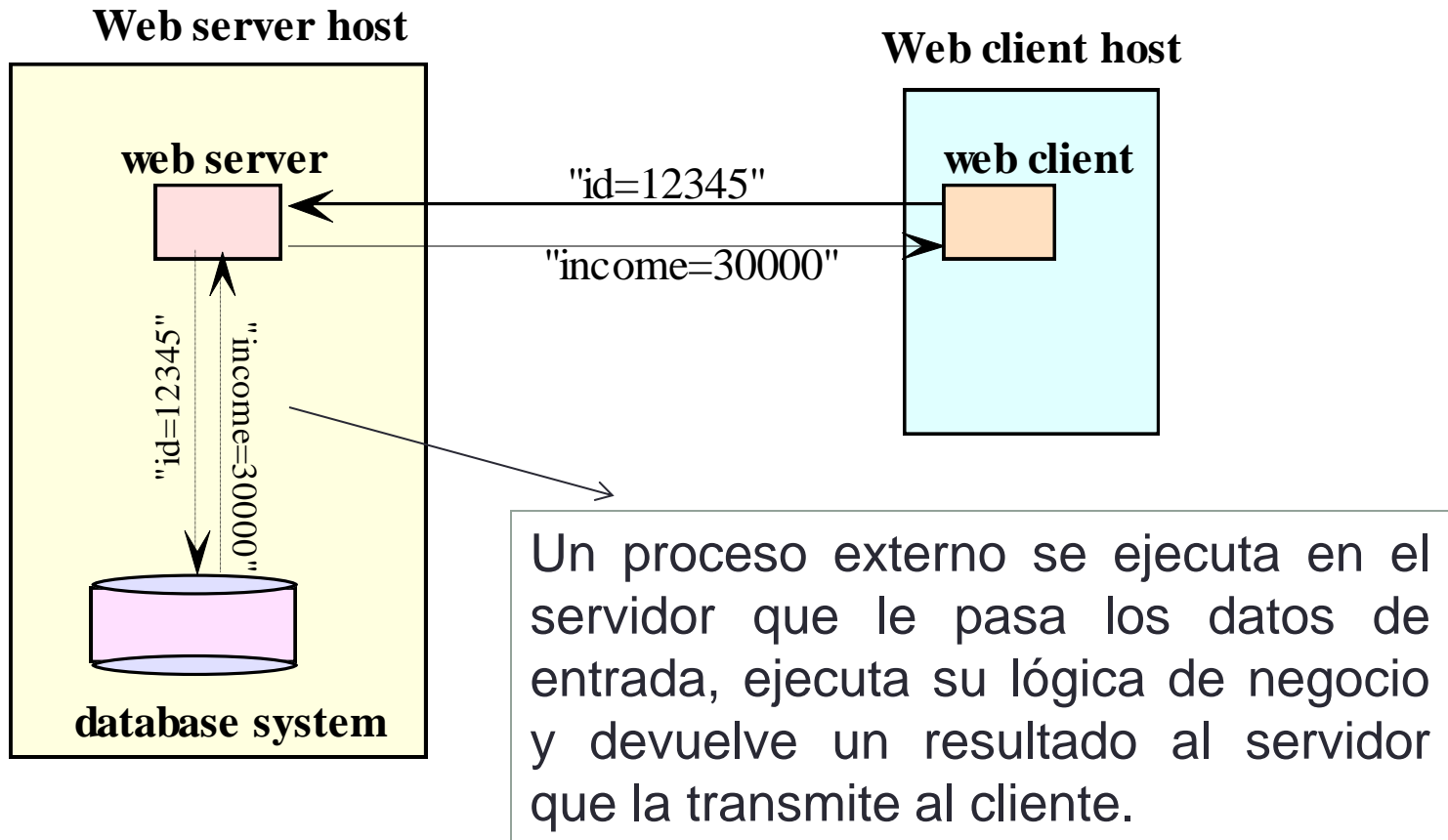
Method/Constructor	Description
<b>URL</b> ( <b>String</b> spec)	Creates a URL object from the URL name contained in a <code>String</code>
<b>URL</b> ( <b>String</b> protocol, <b>String</b> host, <b>int</b> port, <b>String</b> file)	Creates a URL object from the specified protocol, host, port number, and file.
<b>InputStream</b> <b>openStream</b> ( )	Opens a connection to this URL and returns an <code>InputStream</code> for reading from that connection.

# Un navegador URL (Cliente Web)

```
String host = args[0];
String port = args[1].trim();
String fileName = args[2].trim();
String HTTPString = "http://" + host + ":" + port + "/" + fileName;
URL theURL = new URL(HTTPString);
InputStream inStream = theURL.openStream( );
BufferedReader input = new BufferedReader(new InputStreamReader(inStream));
String response = input.readLine();
// lee y muestra una línea de cada vez
while (response != null) {
    System.out.println(response);
    response = input.readLine();
} //end while
```



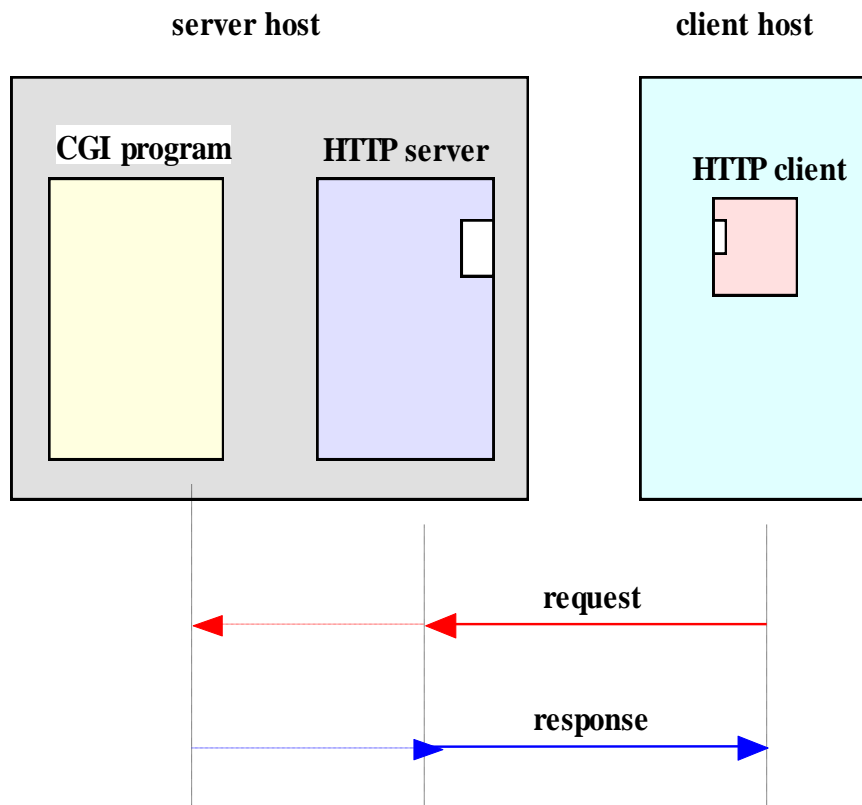
# Contenido Web generado dinámicamente



# El protocolo CGI (Common Gateway Interface)

Un cliente web especifica en la petición HTTP (`<FORM ACTION = "Hello.cgi">`) el programa externo, **CGI script**, que quiere utilizar.

El servidor lo activa como proceso y le suministra los datos de la petición. Cuando el proceso finaliza transmite la salida al servidor que la envía al cliente.



## Página web ejemplo (hola.html) que invoca a un script CGI

```
<HTML>
<HEAD>
<TITLE>A web page which invokes a web script</TITLE>
</HEAD>
<BODY>
<H1>This web page illustrates the use of a web script</H1>
<P>
<BR>
The script or program is either a run-script written in a
script language such as Perl, or an executable generated
  from a source program written in a language such as C/C++.
</P>
<HR>
<FORM METHOD="post" ACTION="hello.cgi">
<HR>
Press <input type="submit" value="here"> to submit your query.
</FORM>
<HR>
</BODY>
</HTML>
```

# Un script web de ejemplo: hola.c

```
/**
 * This C program is for a CGI script which generates
 * the output for a web page.  When displayed by a
 * browser, the message "Hello there!" will be shown
 * in blue.
 */
#include <stdio.h>

main(int argc, char *argv[]) {
    printf("Content-type: text/html%c%c",10,10);
    printf("<font color = blue>");
    printf("<H1>Hello there!</H1>");
    printf("</font>");
}
```

- En la práctica, un script CGI se invoca desde un formulario web.
- El contenido de los campos del formulario se codifica como una cadena de texto (formada por pares nombre=valor y separados por &) denominada “query string” o cadena de petición.
- La query string se pasa al servidor en la petición HTTP de una forma dependiente del método HTTP utilizado:

**GET /cgi/getForm.cgi?name=John%20Doe&quest=peace HTTP/1.0**

**POST /cgi/postForm.cgi HTTP/1.0**

**Accept: \*/\***

**Name=John%20Doe&quest=peace%20on%20earth&color=azure**



# Codificando y decodificando query strings

- Un programa CGI debe decodificar la query string ya sea de la variable de entorno `QUERY_STRING` (método get) o de la entrada estandar (método post).
- Otras variables de entorno utilizadas con CGI son:  
**REQUEST\_METHOD**, **CONTENT\_TYPE** (el tipo de contenido de los datos, que debería ser “application/x-www-form-urlencoded” para una query string) ,**CONTENT\_LENGTH** (la longitud del cuerpo del mensaje, ¡ojo! Con post no se detecta el final del cuerpo del mensaje con EOF, sino atendiendo al contenido de esta variable)

## Otra página web ejemplo (index.html) que invoca a un script CGI en PHP

```

<!DOCTYPE html> <html>
<head> <meta charset="UTF-8"> <title>Un cuestionario sencillo sobre cervezas</title> </head>
<body>
<form action="parametros.php" method="get">
<center> <br> <br> <br>
  Nombre: <input type="text" name="nombre"> <br>
  Tipo: <input type="radio" name="tipo" value="standard" checked> estandard
        <input type="radio" name="tipo" value="artesanal">artesanal <br>
<select name="beer">
<option value="negra">Negra</option> <option value="rubia">Rubia</option>
<option value="tostada">Tostada</option> <option value="Doble Malta">Doble Malta</option>
</select> <br>
<textarea name="texto" rows=5 cols=40></textarea> <br>
 
  <br>
<input type="submit" name="sBoton">
<input type="reset" name="rBoton">
</center>
</form> </body>
</html>

```

## El script CGI en PHP para tratar la respuesta

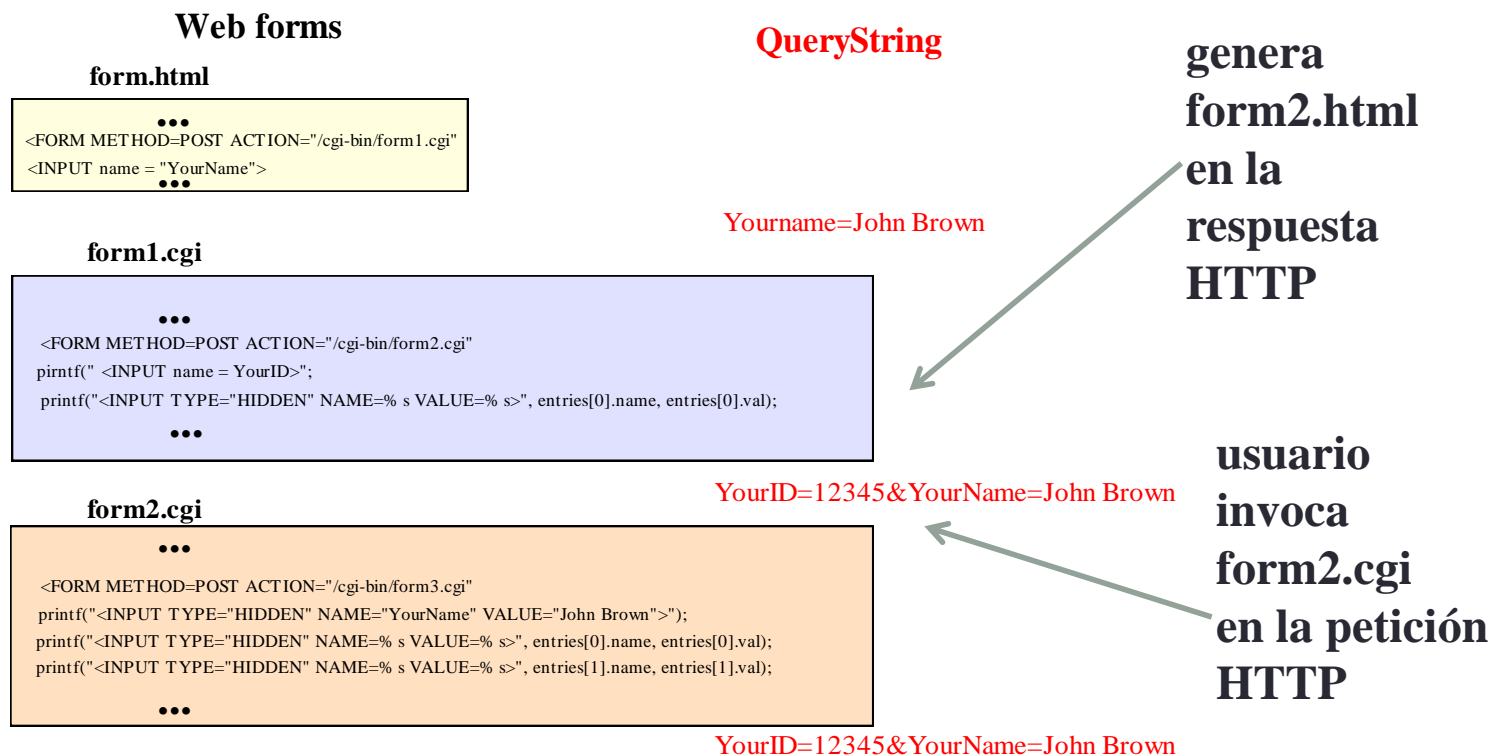
```
<html><head></head><body>
  <h1> Estos son los parametros pasados al CGI PHP:</h1>
  <?php
    foreach($_GET as $key => $value) // $_POST
    {
      echo "<h4>".$key." con valor ".$value."</h4>";
    }
  ?>
  <h2> Esto es to, esto es to, esto es todo AMIGOS!
    
  </h2>
</body></html>
```

# Mantenimiento de información de estado de clientes

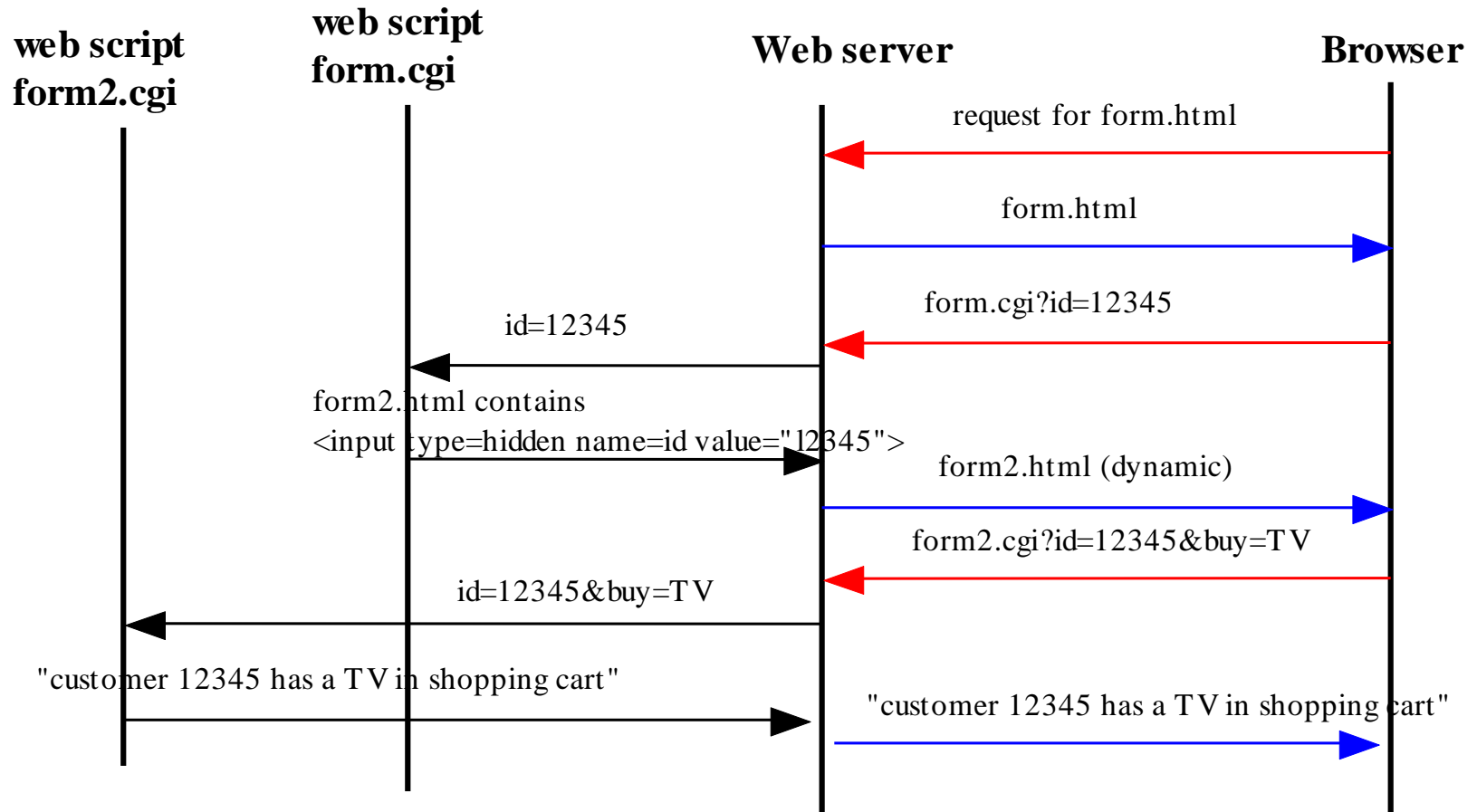
- HTTP es un protocolo de petición-respuesta sin estado.
- Hay que ingeniárselas para poder mantener la información de estado en cada sesión. Dos estrategias:
  - **Del lado del Cliente:** El cliente almacena esa información de estado y se la comunica al servidor en cada petición en que se precise. Existen principalmente dos formas de actuar para un cliente en este contexto:
    - uso de campos ocultos de información en los formularios.
    - uso de objetos denominados Cookies (junto con Servidor).
  - **Del lado del Servidor:** El servidor almacena esa información de estado para cada cliente actuando como un repositorio de información privada.

# Datos de estado mediante campos ocultos de formulario

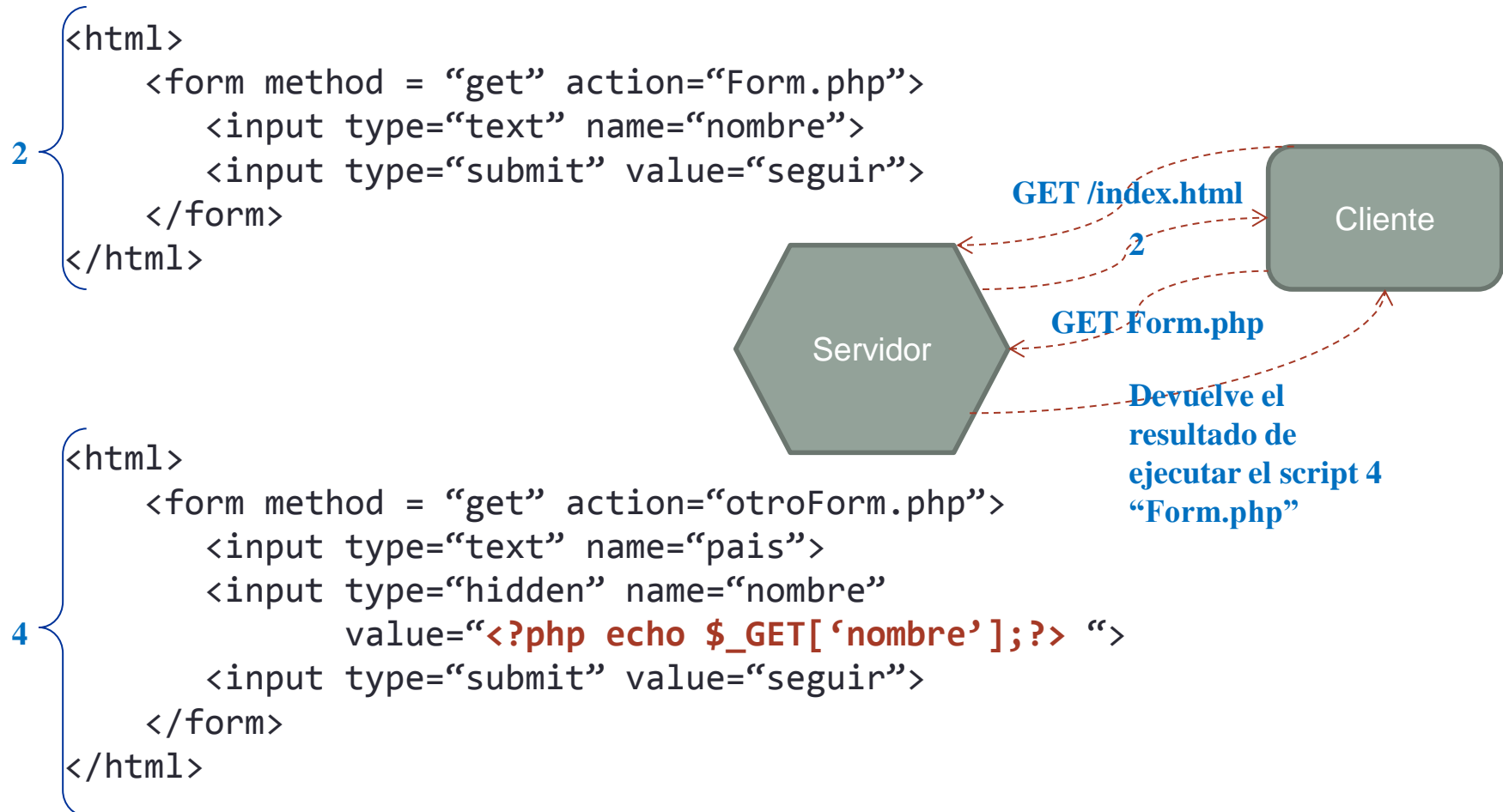
- Un campo oculto es un elemento INPUT de un formulario en el que se especifica TYPE=HIDDEN
- Un campo HIDDEN no se muestra en pantalla. Su valor es el valor del atributo VALUE, su nombre el del atributo NAME



# Diagrama de eventos campos ocultos



# Ejemplo Campos ocultos con PHP



# Resumen Campos ocultos

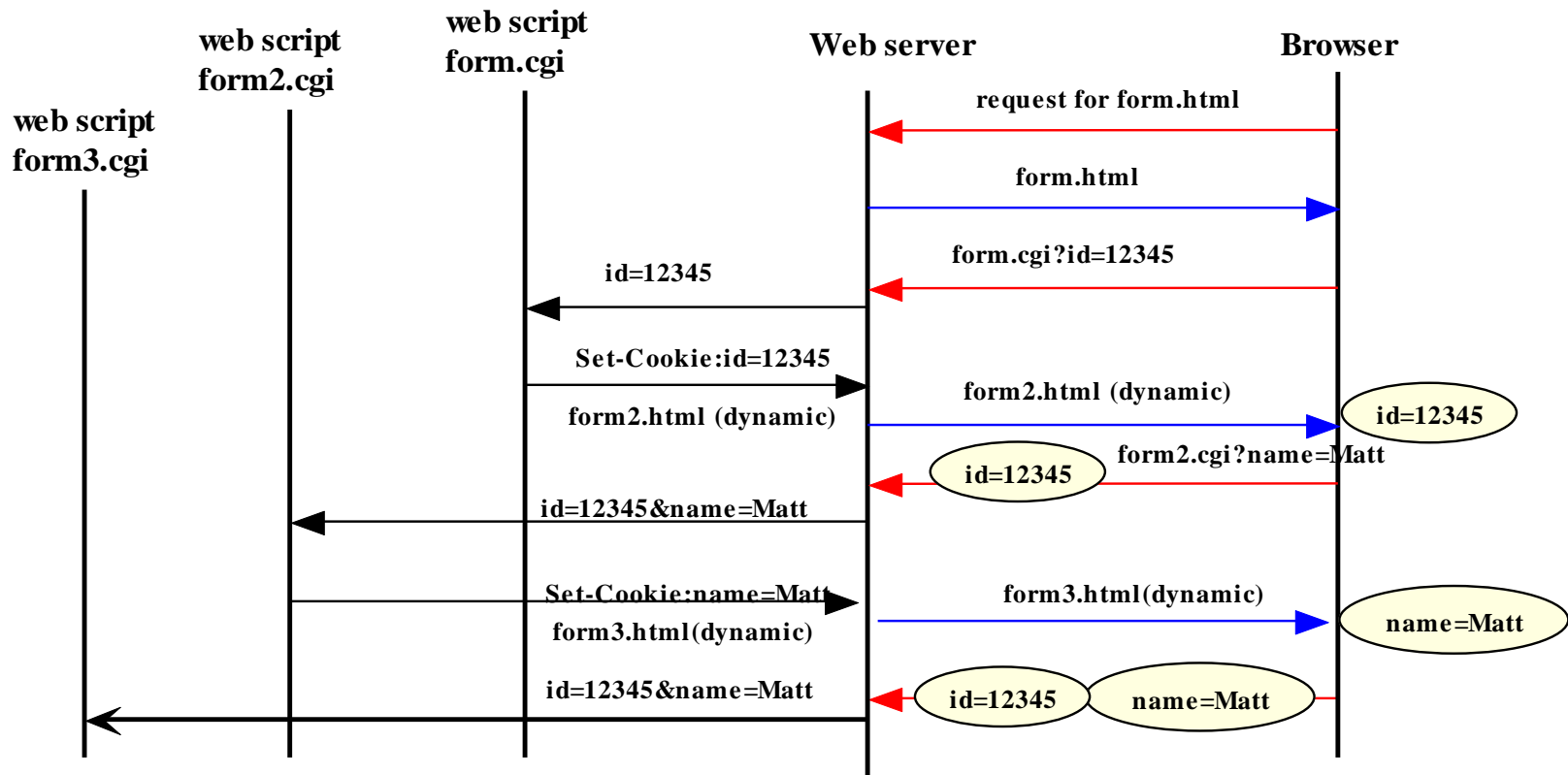
- Es un método sencillo para mantener datos de estado. No requiere recursos adicionales ni en el cliente ni en el servidor para gestionarlos.
- El cliente se encarga de mantener la información de estado y se transmite usando las query strings.
- No es seguro. Aunque no se visualiza se puede leer el código fuente de la página y extraer esos datos de estado.



# Campos de estado mediante Cookies

- El cliente debe reconocerlas y asumir su gestión y almacenamiento.
- La información de estado que se almacena en una Cookie incluye el rango de URL's para el que esta Cookie es válida, así como su plazo de expiración.
- El protocolo HTTP permite transferir Cookies mediante el uso de líneas de cabecera específicas (**Set-Cookie** para respuestas y **Cookie** para peticiones)
  - **Set-Cookie:** *NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN\_NAME; secure*
  - **Cookie:** *(NAME=VALUE)+*

# Diagrama de eventos con cookies



# Ejercicio sobre cookies

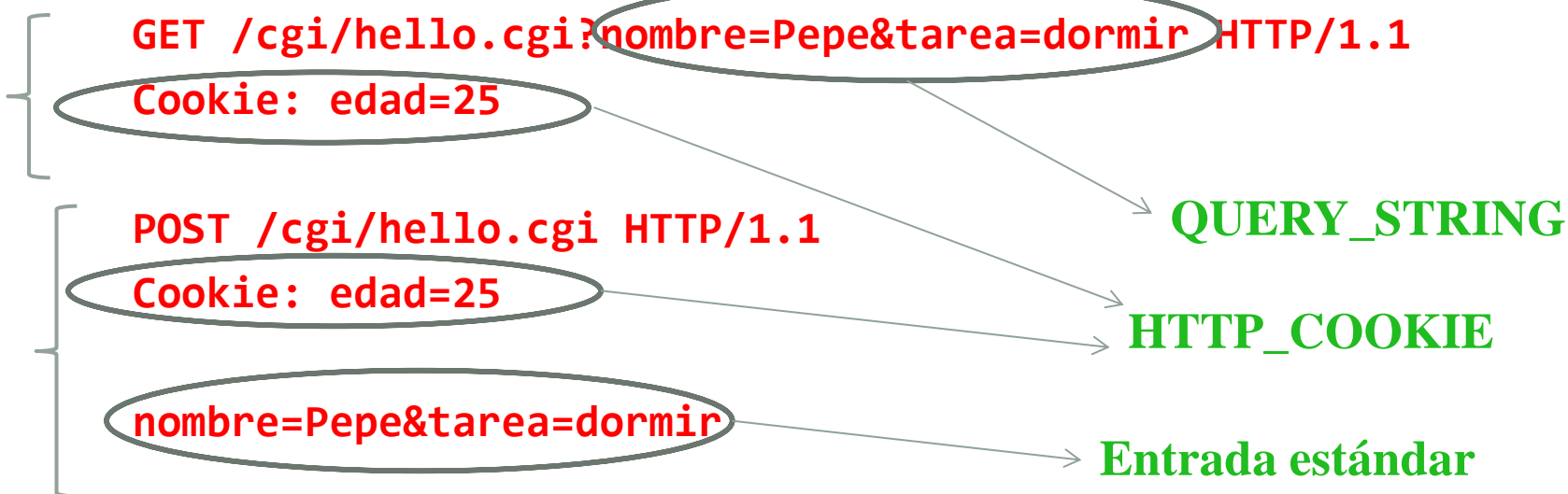
domain	path	¿URLs válidos?
<a href="http://www.uji.es">www.uji.es</a> uji.es es <a href="http://www.icc.uji.es">www.icc.uji.es</a> <a href="http://www.uji.es">www.uji.es</a> <a href="http://icc.uji.es">icc.uji.es</a> <a href="http://uji.es">uji.es</a> <a href="http://www.icc.uji.es">www.icc.uji.es</a> uji.es uji.es	/ / /reservado /cat /res / /reservado/res	<a href="http://www.uji.es/reservado">www.uji.es/reservado</a> <a href="http://www.icc.uji.es/dom">www.icc.uji.es/dom</a> <a href="http://www.elpais.es">www.elpais.es</a> <a href="http://www.icc.uji.es">www.icc.uji.es</a> <a href="http://www.biblioteca.uji.es/catala/noticies">www.biblioteca.uji.es/catala/noticies</a> <a href="http://www.uji.es">www.uji.es</a> doc.icc.uji.es/esp <a href="http://www.icc.uji.es/reservado/reservado">www.icc.uji.es/reservado/reservado</a> aia.act.uji.es/eng/res www.sg.uji.es/es

# Envío de cookies del navegador al servidor

- Cuando un servidor encuentra la línea de cabecera:

**Cookie: NAME<sub>1</sub>=VALUE<sub>1</sub>; NAME<sub>2</sub>=VALUE<sub>2</sub>; ...; NAME<sub>n</sub>=VALUE<sub>n</sub>;**

el servidor extrae las parejas nombre-valor y las ubica en una variable de entorno denominada **HTTP\_COOKIE**. El script CGI debe recuperar esa información por medio de esa variable. Ejemplo:



# Cookies de terceros

- 1) Un usuario visita <http://www.example1.com/home.html>
- 2) La página enviada al cliente contiene una etiqueta **IMG** con una referencia a <http://ad.example.com/adv1.gif> (una publicidad)
- 3) El cliente solicita automáticamente esa imagen. La respuesta incluye una cookie de [ad.example.com](http://ad.example.com) **(transacción no verificada)**
- 4) El mismo usuario visita <http://www.exampleB.com/home.html>
- 5) La página enviada al cliente contiene una etiqueta **IMG** con una referencia a <http://ad.example.com/adv2.gif> (otra publicidad)
- 6) El cliente solicita esa imagen enviando la cookie previamente recibida de este servidor automáticamente. La respuesta incluye otra cookie de [ad.example.com](http://ad.example.com) **(transacción no verificada)**

[ad.example.com](http://ad.example.com) **¡Puede calcular un perfil de ese usuario sin que él se entere!**

# Ejemplos cookies con PHP

## Crear una cookie

```
<?php
    $value = 'something from somewhere';
    setcookie("TestCookie", $value);
    setcookie("TestCookie", $value, time()+3600); /* expira en 1 hora */
    setcookie("TestCookie", $value, time()+3600, "/~rasmus/",
              "example.com", 1);
?>
```

## Leer una cookie

```
<?php $cookie_name = 'TestCookie';
    if(!isset($_COOKIE[$cookie_name])) {
        print 'Cookie de nombre "' . $cookie_name . '" no existe.';
    } else {
        print 'Cookie de nombre "' . $cookie_name . '" valor es: ' .
            $_COOKIE[$cookie_name];
    }
?>
```

# Ejemplos cookies con PHP

¿Qué hace el siguiente script PHP?

```
<?php
    if(isset($_COOKIE['cont'])) { // Caduca en un año
        setcookie('cont', $_COOKIE['cont']+1, time()+365*24*60*60);
        $mensaje = 'Número de visitas: ' . $_COOKIE['cont'];
    } else { // Caduca en un año
        setcookie('cont', 1, time()+365*24*60*60);
        $mensaje = 'Bienvenido a nuestra página web';
    }
?>
<html>
    <?php echo $mensaje; ?>
</html>
```

# Datos de estado del lado del servidor

Una **session** es un mecanismo, implementado normalmente con un objeto, para conservar datos de un cliente en el servidor.

Es habitual que las sesiones utilicen una cookie de forma implícita que identifique unívocamente a cada cliente.

El servidor gestiona al objeto sesión almacenando todos los datos de la sesión.

<?php

```
session_start();  
echo 'session_id(): ' . session_id();  
echo "<br />\n";  
echo 'session_name(): ' . session_name();  
echo "<br />\n";  
print_r(session_get_cookie_params());
```

?>



# session: ejemplos en PHP

```
<?php
    session_start();
    if(isset($_SESSION['cont'])) {
        $_SESSION['cont'] = $_SESSION['cont']+1;
        $mensaje = 'Número de visitas: ' . $_SESSION['cont'];
    } else {
        $_SESSION['cont'] = 1;
        $mensaje = 'Bienvenido a nuestra página web';
    }
?>
<html>
    <p>
        <?php echo $mensaje; ?>
    </p>
</html>
```

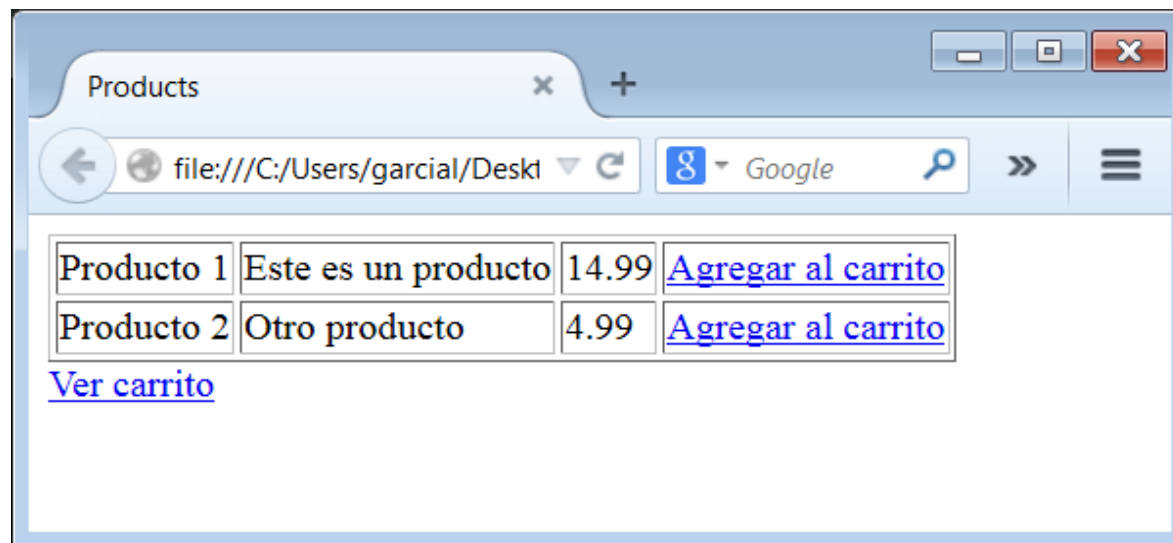
# session: más ejemplos en PHP

## Como eliminar una sesión:

```
<?php
// Borra todas las variables de sesión
$_SESSION = array();
// Borra la cookie que almacena la sesión
if(isset($_COOKIE[session_name()])) {
    setcookie(session_name(), '', time()-42000, '/');
}
// Finalmente, destruye la sesión
session_destroy();
?>
```

## Como programar un carrito de compras:

## La página web de compras “*products.php*”



```
<html>
  <head>
    <title>
      Products
    </title>
  </head>
  <body>
    <table border="1">
      <tr> <td> Producto 1</td> <td> Este es un producto </td>
        <td> 14.99</td>
        <td> <a href="cart.php?action=add&id=1">Agregar al carrito</a></td></tr>
      <tr> <td> Producto 2</td>
        <td> Otro producto </td><td>4.99</td>
        <td> <a href="cart.php?action=add&id=2">Agregar al carrito</a></td></tr>
    </table>
    <a href="cart.php">Ver carrito</a>
  </body>
</html>
```

```

<?php session_start(); ?>
<html> <head> <title> Carrito </title>
    <?php
        //conexión a la base de datos
    ?>
</head>
<body>

<?php
    $product_id = $_GET[id]; // el id del producto de la URL
    $action = $_GET[action]; // la acción a realizar de la URL
    if($product_id && !productExists($product_id)) {
        die("Error. El producto no existe");
    }
    switch($action) {          //decidir qué hacer
        case "add":            $_SESSION['cart'][$product_id]++; break;
        case "remove":         $_SESSION['cart'][$product_id]--;
                                if($_SESSION['cart'][$product_id] == 0)
                                    unset($_SESSION['cart'][$product_id]);
                                break;
        case "empty":          unset($_SESSION['cart']); break;
    }
    function productExists($product_id) {
        $sql = sprintf("SELECT * FROM php_shop_products WHERE id = %d;", $product_id);
        return mysql_num_rows(mysql_query($sql)) > 0;
    }
?>

```

## El script “*cart.php*”

```

<?php
    if($_SESSION['cart']) { // si el carrito no está vacío, muéstralo
        echo "<table border=\"1\" padding=\"3\" width=\"40%\">"; // tabla de resultados
        foreach($_SESSION['cart'] as $product_id => $quantity) {
            $sql = sprintf("SELECT name, description, price FROM php_shop_products
                           WHERE id = %d;", $product_id);
            $result = mysql_query($sql);
            list($name, $description, $price) = mysql_fetch_row($result);
            $line_cost = $price * $quantity; // calcula la línea de coste
            $total = $total + $line_cost;    // súmalo al coste total
            echo "<tr><td align=\"center\">$name</td>"; // muéstralo en las celdas
            echo "<td align=\"center\">$quantity<a href=\"$_SERVER[PHP_SELF]\" .
                \"?action=remove&id=$product_id\">X</a></td>";
            echo "<td align=\"center\">$line_cost</td></tr>";
        }
        echo "<tr><td colspan=\"2\" align=\"right\">Total</td>";
        echo "<td align=\"right\">$total</td></tr><tr>";
        echo "<td colspan=\"3\" align=\"right\"><a href=\"$_SERVER[PHP_SELF]\" .
            \"?action=empty\" onclick=\"return confirm('¿Estás seguro?');\">
            \"Vaciar carrito</a></td></tr></table>";
    } else { // el carrito está vacío
        echo "No tienes productos en el carrito.";
    }
?>

```

```
<a href="products.php">
  Continua comprando</a>
```

```
<?php
/*
```

```
products table:
```

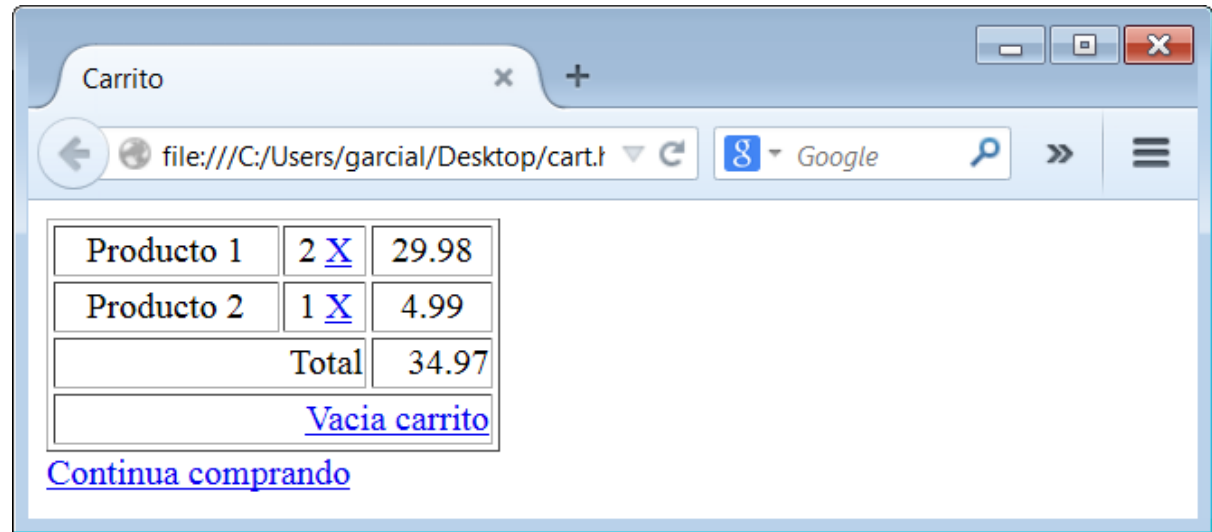
```
CREATE TABLE `products` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR( 255 ) NOT NULL ,
  `description` TEXT,
  `price` DOUBLE DEFAULT '0.00' NOT NULL ,
  PRIMARY KEY ( `id` )
```

```
);
```

```
*/
```

```
?>
```

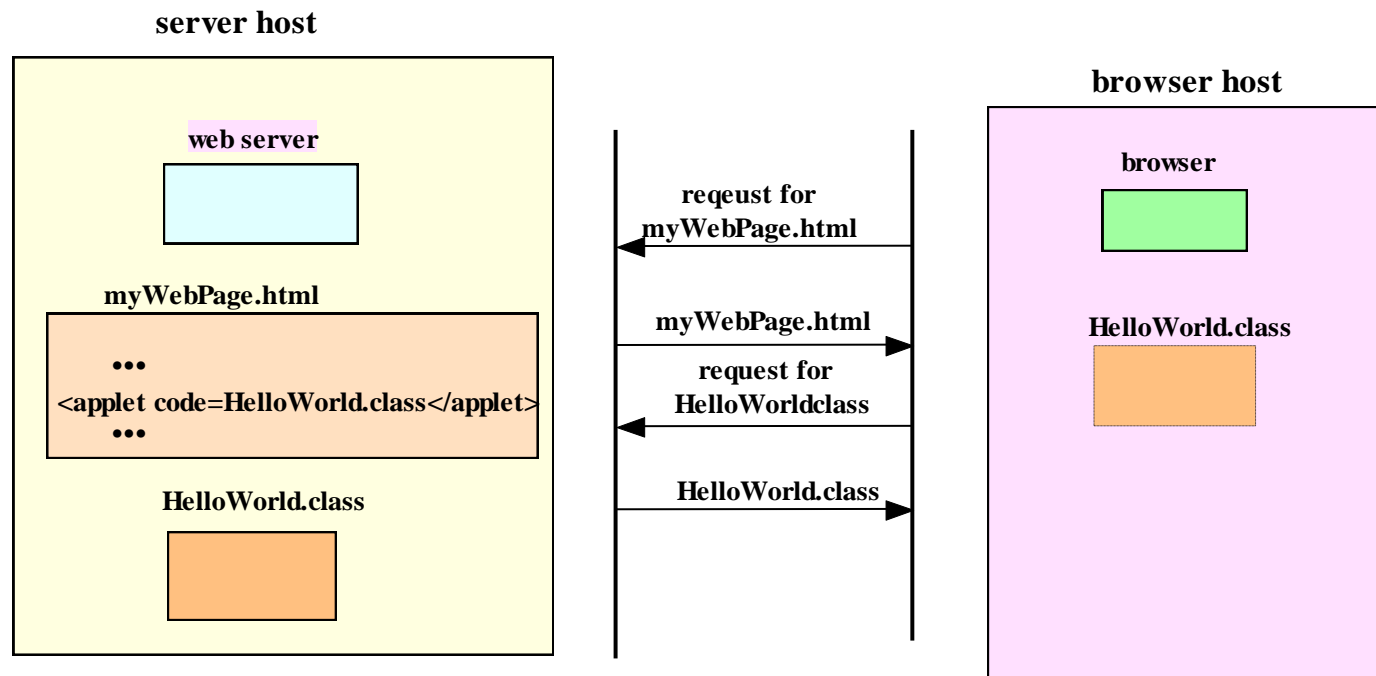
```
</body>
</html>
```



**Fin del script “*cart.php*”**

# Applets

- Un Applet se almacena en un servidor web de forma similar una página web.
- Cuando se referencia en una página web, el navegador lo solicita al servidor que lo envía para ejecución en el cliente.



# Ejecución de un Applet

- Es un objeto gráfico que se ejecuta en el contexto de un hilo que se ejecuta en paralelo al proceso del navegador.
- Cuando el cliente lo recibe ejecuta su método `init()`. El applet lanza el método `start()` y el `paint()` (ambos deben ser programados) y para finalizar su ejecución debe invocar a `stop()` y, posteriormente a `destroy()` para liberar recurso
- Aspectos de seguridad a considerar:
  - 1) No se le permite leer o escribir ficheros en el cliente;
  - 2) No puede realizar conexiones de red excepto con el host servidor.



# Especificación de un Applet en HTML

<APPLET *Opciones*>

*Opciones* :: CODE = "demoxxx.class"

CODEBASE = "demos/"

NAME = "sonrie"

WIDTH = "100"

HEIGHT = "50"

ALIGN = "Top" ...

(<PARAM NAME = "nombre" VALUE = "valor">)\*

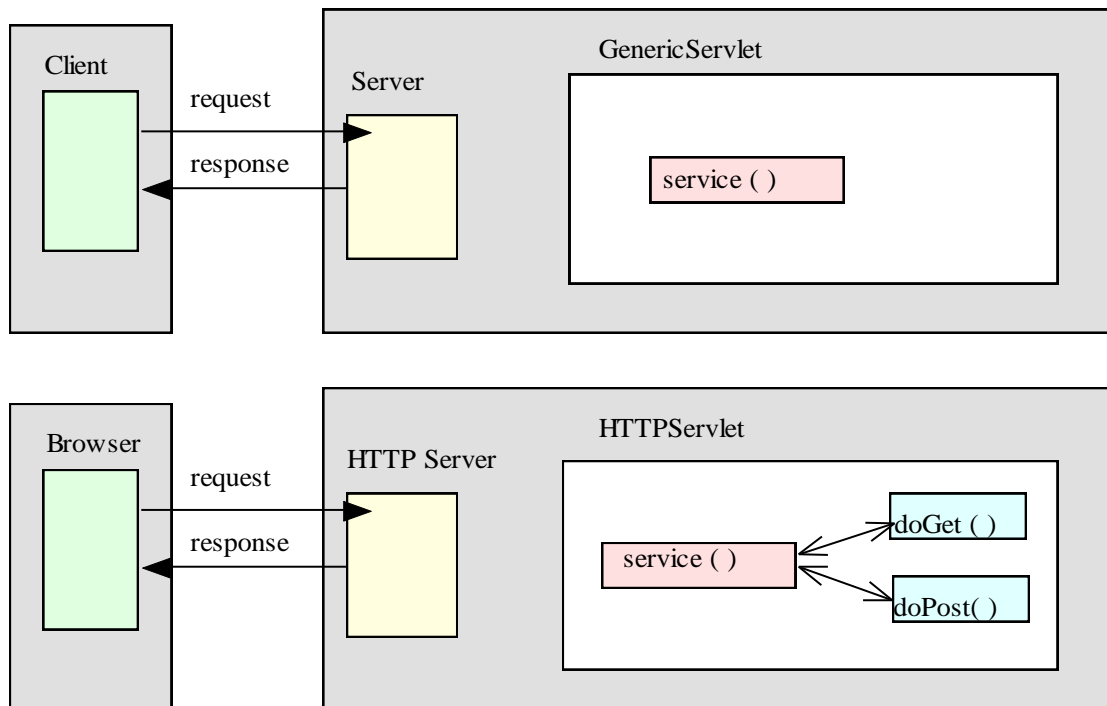
</APPLET>

# Java Servlets

- Cumplen la función de los tradicionales CGIs de una forma *más eficiente* para el lenguaje Java.
- Se ejecutan en la JVM del servidor.
- Un servlet es un objeto que se carga y ejecuta en un “contenedor de servlets” o “motor de servlets”
- Un servlet puede manejar múltiples peticiones concurrentemente, puede sincronizar peticiones y redirigirlas a otros servidores o servlets.
- Dos tipos de servlets que implementan `javax.servlet.Servlet`:
  - **Genéricos**: heredan de `javax.servlet.GenericServlet`
  - **Http**: heredan de `javax.servlet.http.HttpServlet`

# Java Servlets

- Un servlet genérico sobrescribe el método `service()`
- Un servlet `Http` sobrescribe los métodos `doGet()` `doPost()`



# Ejemplo de servlet sencillo

```
public class HolaMundoServlet extends HttpServlet {  
    /** * gestiona un método HTTP GET construyendo una página Web. */  
    public void doGet (HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        PrintWriter out;  
        String title = "Simple Servlet Output";  
        // Fija el contenido de la página web  
        response.setContentType("text/html");  
        // Escribe la página web de respuesta  
        out = res.getWriter();  
        out.println("<HTML><HEAD><TITLE>");  
        out.println(title);  
        out.println("</TITLE></HEAD><BODY>");  
        out.println("<H1>" + title + " </H1>");  
        out.println("<P> ¡Saludos a todos!.");  
        out.println("</BODY></HTML>");  
        out.close();  
    }  
}
```

# HTTP Servlet y formularios web

- Los métodos de la clase `HttpServlet` que gestionan peticiones de clientes tienen dos argumentos:

```
public void doGet (HttpServletRequest req, HttpServletResponse res)  
public void doPost(HttpServletRequest req, HttpServletResponse res)
```

- Un objeto `HttpServletRequest` **req** que recoge los datos del cliente.
  - `public String getParameter(String name)`
  - `public Enumeration getParameterNames()`
  - `public String[] getParameterValues(String name) // checkbox p.e.`
  - `public String getQueryString()`
- Un objeto `HttpServletResponse` **res** que encapsula la respuesta al cliente.
  - `public PrintWriter getWriter() vs public ServletOutputStream getOutputStream()`
  - `public void setContentType(String type)`
  - `public void SendRedirect(String location)`

# Concurrencia Http Servlets

- Los Servlets Http son capaces de servir múltiples clientes de forma concurrente.
- Si los métodos del servlet acceden a un recurso compartido entonces:
  - Se ha de sincronizar el acceso a ese recurso.
  - Alternativamente se modifica el servlet para que maneje sólo una petición de cliente de cada vez.

# Ejemplo de gestión de petición GET

```
public class LibroServlet extends HttpServlet {  
    public void doGet (HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        ...  
        // Especifica tipo de contenido antes de obtener el PrintWriter  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        // ahora escribe la respuesta  
        out.println("<html>" +  
            "<head><title>Servlet educado</title></head>" + ... );  
        // Lee el id que el usuario introdujo en el campo de texto  
        String nombre = req.getParameter("id");  
        if (nombre != null) {  
            ... } // captura los datos de ese libro e imprímelos  
        out.println("</body></html>");  
        out.close();  
    }  
    ...}
```

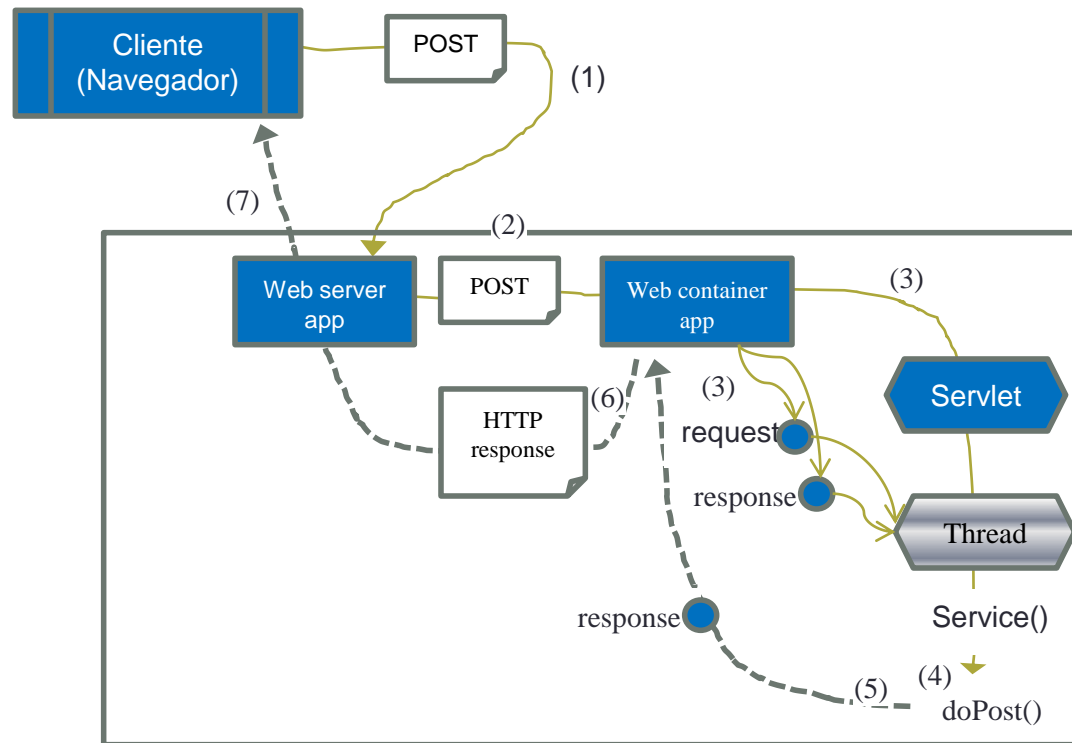
# Ejemplo de gestión de petición POST

```
public class ReciboServlet extends HttpServlet {  
    public void doPost(HttpServletRequest req,  
        HttpServletResponse res) throws ServletException, IOException {  
        ...  
        // Especifica tipo de contenido antes de obtener el PrintWriter  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter( );  
        // ahora escribe la respuesta  
        out.println("<html>" + "<head><title> Receipt </title>" + ...);  
        out.println("Gracias por comprar aquí!" +  
            req.getParameter("cardname") + ...);  
        out.close();  
    }  
    ...  
}
```



# Ciclo de vida de un servlet

- El contenedor de servlets invoca el método `init()`
- El servlet gestiona respuestas de clientes.
- El contenedor de servlets destruye el servlet (¿Opciones?).



# Relación entre URI y servlet

- Se especifica en el Deployment Descriptor (DD) web.xml

```
<web-app ... >
  <servlet>
    <servlet-name> Nombre interno 1 </servlet-name>
    <servlet-class> Servlet1 </servlet-class>
  </servlet>
  <servlet>
    <servlet-name> Nombre interno 2 </servlet-name>
    <servlet-class> Servlet2 </servlet-class>
  </servlet>
  ...
  <servlet-mapping>
    <servlet-name> Nombre interno 1 </servlet-name>
    <url-pattern> /publico1 </url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name> Nombre interno 2 </servlet-name>
    <url-pattern> /publico2 </url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

Información de sintaxis XML

Clase java del servlet (Servlet1.class)

URL para acceder al servlet

# Información de estado en servlets

- Los mecanismos de almacenamiento del estado de información estándar con CGIs pueden seguir utilizándose con servlets.
- Adicionalmente, los servlets proporcionan otro mecanismo nuevo denominado objeto *HttpSession*.

- **Campos ocultos de formulario:**

```
response.setContentType("text/plain");  
PrintWriter salida = response.getWriter();  
salida.println("<INPUT TYPE=\"HIDDEN\" NAME=ID VALUE=\" +algo);
```

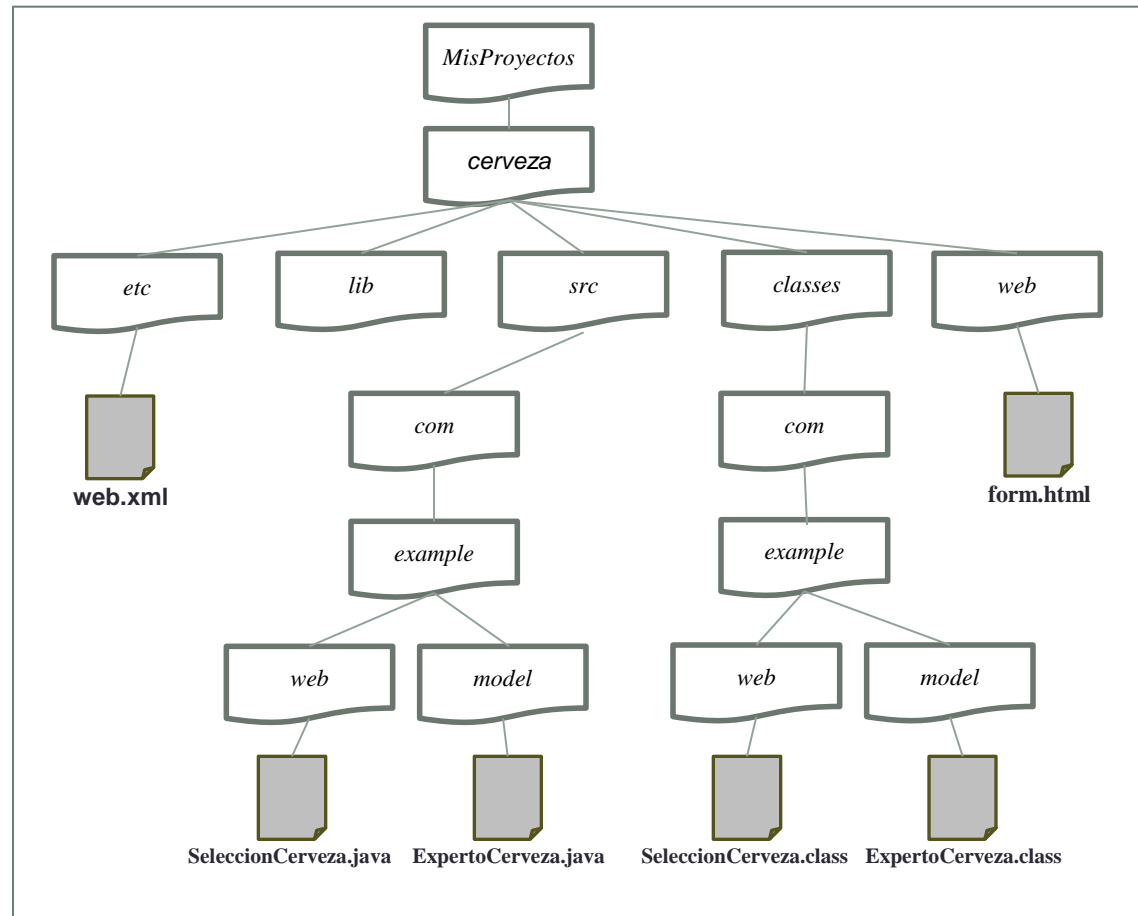
- **Variables:**

- Cuidado con el acceso a elementos compartidos.
  - métodos *synchronized* en su acceso y modificación.

# Estructura directorios aplicaciones Servlet

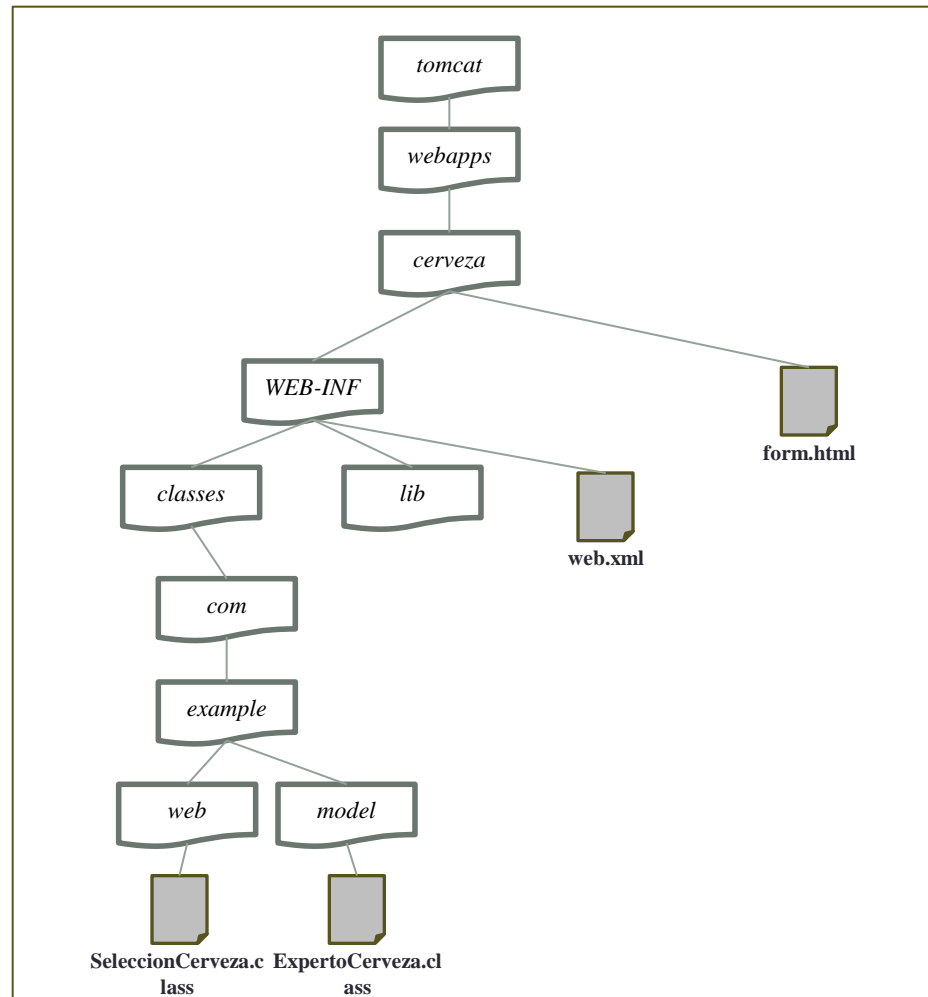
- Dos estructuras distintas: una para el entorno de desarrollo y otra para el entorno de desplegado o explotación.

Entorno de  
desarrollo



# Estructura directorios aplicaciones Servlet

Entorno de  
desplegado



# Un sencillo ejemplo MVC: modelo

```
// Un avanzado sistema inteligente para aconsejar cervezas
```

```
public class ExpertoCerveza {  
    public List dameMarcas(String color) {  
        List marcas = new ArrayList();  
        if (color.equals("negra")) {  
            marcas.add("Guinness");  
        } else {  
            marcas.add("Mahou");  
            marcas.add("Estrella Galicia");  
            marcas.add("San Miguel");  
        }  
        return(marcas);  
    }  
}
```

# Un sencillo ejemplo MVC: controlador

```
public class SeleccionCerveza extends HttpServlet {  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws IOException, ServletException {  
  
        String c = request.getParameter("color");  
        ExpertoCerveza ec = new ExpertoCerveza();  
        List resultado = ec.dameMarcas(c);  
  
        request.setAttribute("estilos", resultado);  
        RequestDispatcher vista =  
            request.getRequestDispatcher("resultado.jsp");  
  
        vista.forward(request, response);  
    }  
}
```

# Un sencillo ejemplo MVC: vista con JSP

```
<%@ page import="java.util.*" %>
<html>
<body>
<h1 align="center"> Recomendaciones de Cervezas (via JSP's) </h1>
<p>
<%
    List estilos = (List) request.getAttribute("estilos");
    Iterator it = estilos.iterator();

    while (it.hasNext()) {
        out.print("<br>Prueba con:      " + it.next());
    }
%>
</body>
</html>
```

→ Código Java

→ Código Java

→ Código Java

Código HTML



# Información de estado en servlets

```
public class Counter extends HttpServlet {

    public int counter = 0;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter output = response.getWriter();
        increment(output);
    }

    private synchronized void increment(PrintWriter output) {
        output.println("Este servlet ha sido accedido " + counter + " veces.");
        counter++;
    } //end increment
}
```

# Cookies con servlets

- Una **cookie** tiene un nombre, valor y atributos opcionales tales como path, dominio, periodo de validez, etc.
- Un servlet envía cookies a un navegador mediante el empleo del método `HttpServletResponse.addCookie(javax.servlet.http.Cookie Cookie)`, que añade campos a la cabecera de una respuesta HTTP para enviar una a una las cookies al navegador.
- Un navegador devuelve cookies al servlet añadiendo campos a la cabecera de la petición HTTP. Estas cookies pueden ser recuperadas del objeto request mediante el método `HttpServletRequest.getCookies()`.
- Recordad que el nombre de varias cookies puede coincidir, pero en este caso deben diferenciarse mediante el atributo path.

# Cookies con servlets

- Una cookie es un objeto de la clase `javax.servlet.http.cookie`
- Algunos métodos útiles:

`public Cookie(String name, String value)`

`public String getName()`

`public String getValue()`

`public setValue(String val)`

`public void setMaxAge(int expiry)`

`public void setPath(String uri)`

`public String getPath()`

`public String getDomain()`

`public void setDomain(String domain)`

# Cookies (doGet) con servlets

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

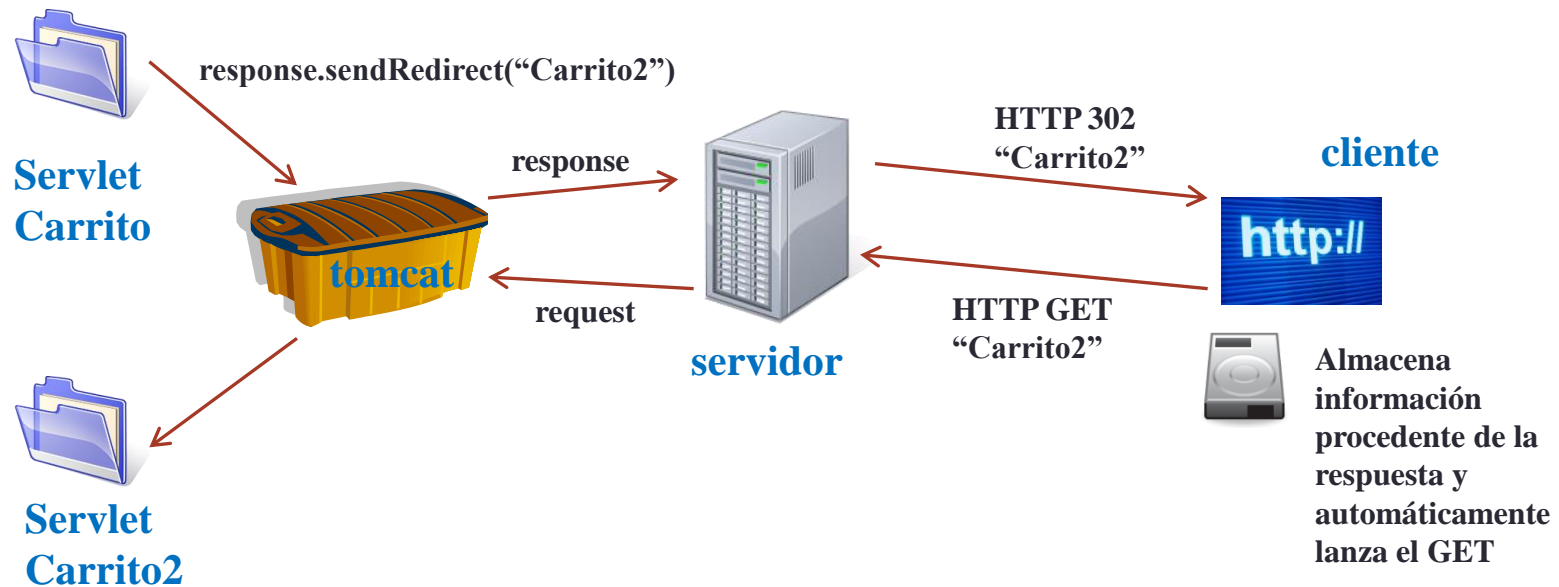
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<H1>Contenido del carrito de compra:</H1>");

    Cookie cookies[];
    cookies = request.getCookies();
    if (cookies != null)
        for ( Cookie c : cookies )
            if (c.getName().startsWith("Item"))
                out.println( c.getName() + ": " + c.getValue());

    out.close();
}
```

# Cookies (sendRedirect)

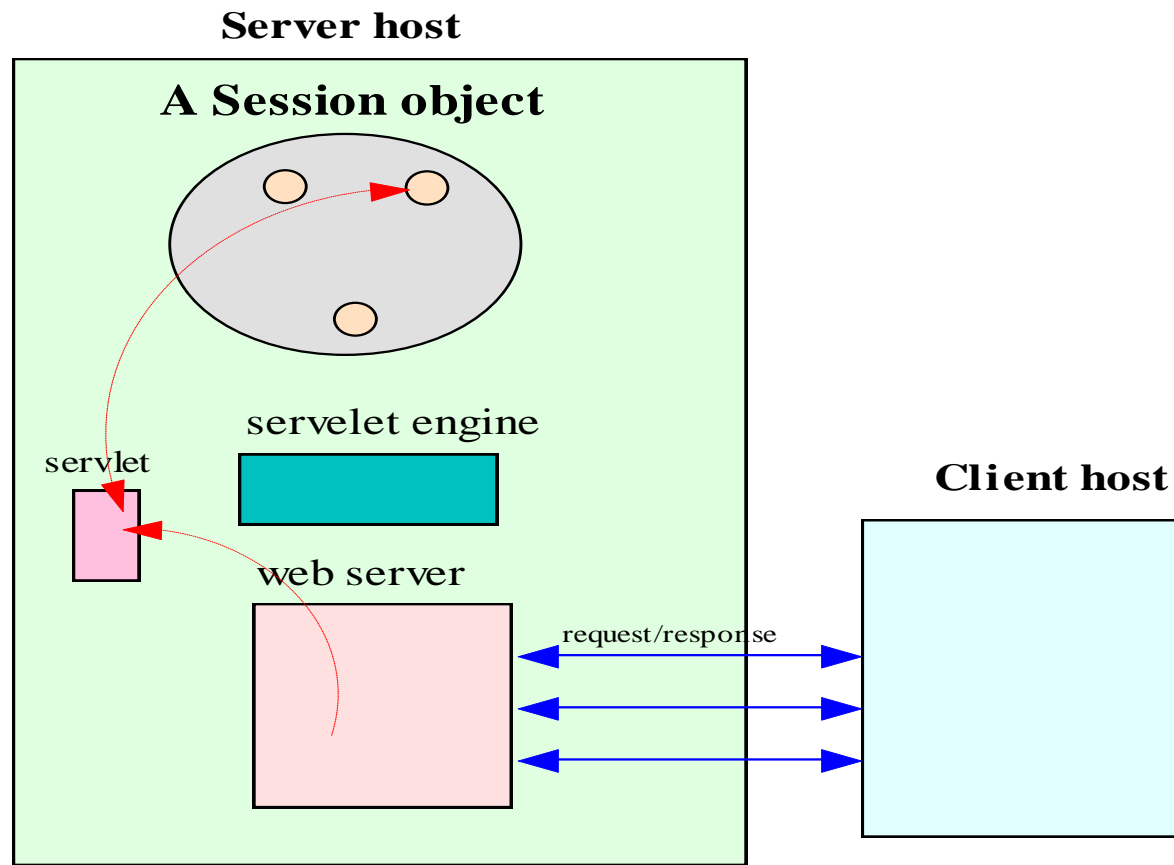
¿Qué involucra ejecutar `response.sendRedirect("Carrito2")` ?



# Objetos *HTTP Session*

- El paquete `javax.servlet.http` proporciona el interfaz público `HttpSession` que proporciona un método para identificar a un usuario en peticiones o visitas a un sitio web y para almacenar información sobre un usuario.
- El contenedor de servlet utiliza este interfaz para crear una sesión entre un cliente HTTP y el servidor HTTP. La sesión persiste para un periodo de tiempo determinado. Una sesión corresponde frecuentemente con un usuario que puede visitar el sitio web muchas veces.
- Este interfaz permite a los servlets:
  - Ver y manipular información de sesión tal como identificador de la sesión, instante de creación y último acceso.
  - Ligar objetos a sesiones de forma que la información persista entre múltiples conexiones del usuario.
- La información de sesión sólo alcanza a la aplicación web en ejecución (Contexto), por lo que la información de un contexto no será visible directamente en otro contexto.

# Objetos *HTTP Session*



# Objetos *HTTP Session*

- Algunos métodos:

```
public HttpSession getSession(boolean create)
```

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

```
    ...
    // get session object
    HttpSession session = req.getSession(true)
    if (session != null) {
```

```
        ...
    } ...
```

```
public String getID()
public Object getAttribute(String name)
public Enumeration getAttributeNames()
public void removeAttribute(String name)
```



# Ejercicio Servlet

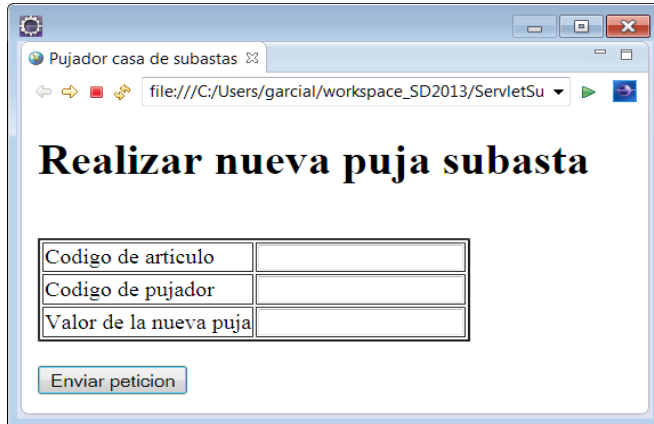
Implementar parcialmente un servicio de subastas de artículos. Por simplicidad, de cada artículo guardamos su código (`int`), una descripción del artículo (`String`), el código del pujador que lleva la máxima puja (`int`) y el valor de esa puja máxima hasta el momento (`int`). Se supone implementada la siguiente clase para representar a un artículo en la casa de subastas:

```
public class Articulo implements Serializable {
    private int codArticulo, codPujador, valorPuja;
    private String descripcion;
    Articulo(int codArticulo, int codPujador, int valorPuja, String descripcion){...} // constructor
    // getters, setters y toString()
}
```

Los artículos disponibles se guardan en un almacén implementado mediante la siguiente clase:

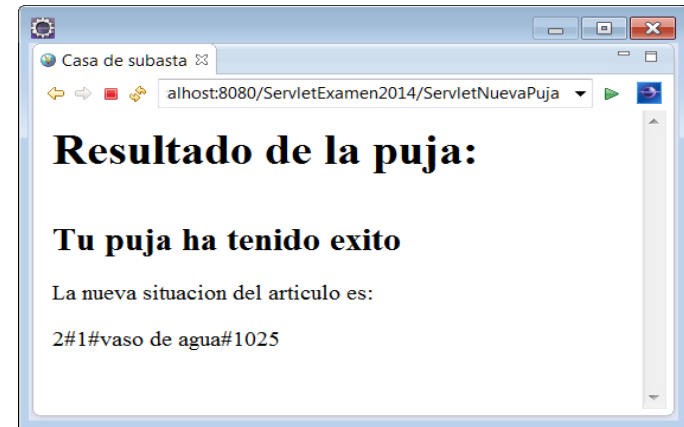
```
public class AlmacenArticulos {
    // No importa con qué estructura de datos se guardan los artículos
    // Devuelve un vector con los datos de todos los articulos
    public String[] listaArticulos() {...}
    // Devuelve los datos del articulo, "-1" si no lo encuentra
    public String dameArticulo(int codArticulo) {...}
    // Si la puja tiene exito devuelve los nuevos datos del articulo
    // Si no encuentra el articulo, devuelve "-1"
    // Si lo encuentra pero el valor no supera la puja actual, devuelve "-2"
    public String nuevaPuja(int codArticulo, int codPujador, int valor) {...}
}
```

La página web cliente tiene que mostrar y recoger la información solicitada en el siguiente formulario web



The screenshot shows a web browser window with the title 'Pujador casa de subastas'. The address bar shows a file path: 'file:///C:/Users/garcial/workspace\_SD2013/ServletSu'. The main content area has the heading 'Realizar nueva puja subasta'. Below the heading is a form with three input fields: 'Codigo de articulo', 'Codigo de pujador', and 'Valor de la nueva puja'. At the bottom of the form is a button labeled 'Enviar peticion'.

La página web resultado que mostrará el cliente tiene que tener el siguiente formato:



The screenshot shows a web browser window with the title 'Casa de subasta'. The address bar shows 'alhost:8080/ServletExamen2014/ServletNuevaPuja'. The main content area has the heading 'Resultado de la puja:'. Below the heading is the text 'Tu puja ha tenido exito'. Underneath that is the text 'La nueva situacion del articulo es:' followed by '2#1#vaso de agua#1025'.

**Escribe el código de la página del cliente y de la operación doPost de un ServletNuevaPuja sencillo en el servidor**