Fundamentos de Aplicaciones y Servicios Telemáticos

2º Grado en Ingeniería de Tecnologías de Telecomunicación

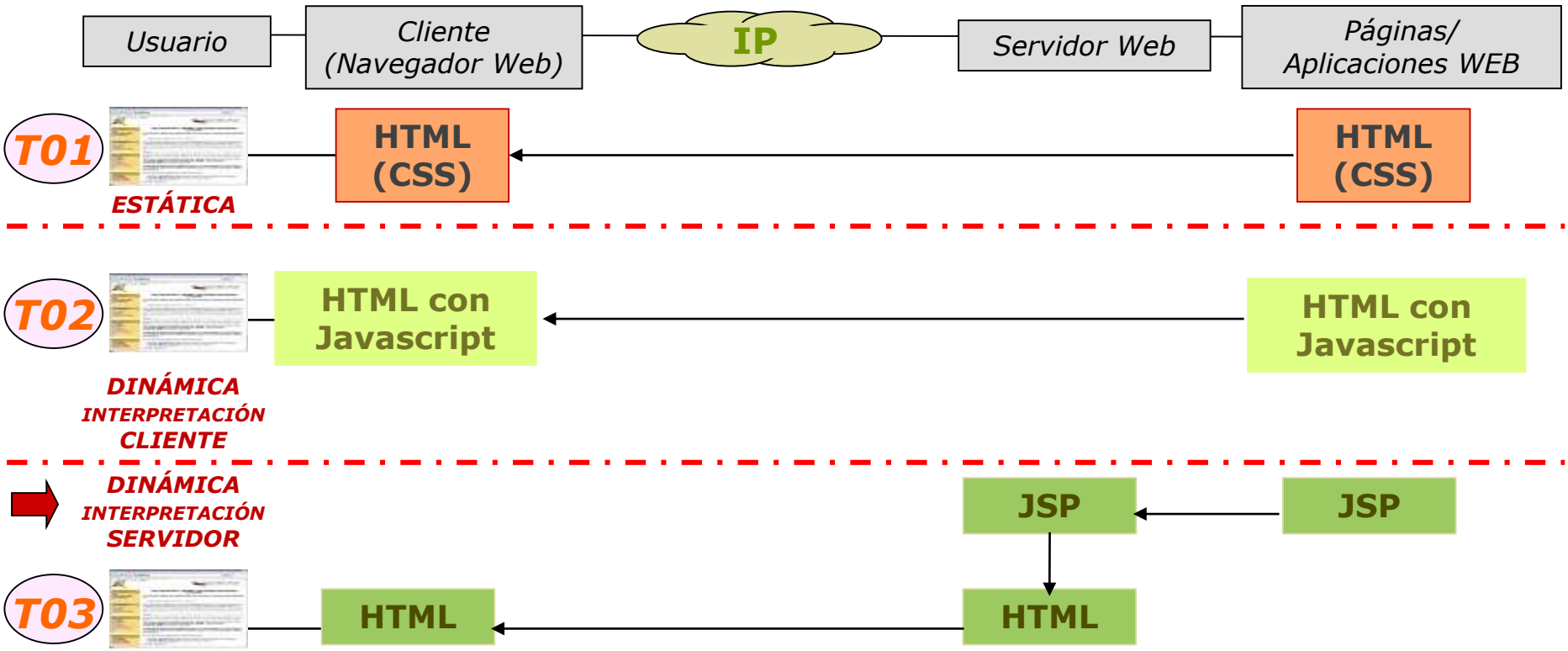
Departamento de Ingeniería Telemática

Tema/Práctica 03

Programación Web Dinámica con interpretación en el Servidor

Objetivo

- Aprender a **Diseñar Aplicaciones Web "dinámicas en *Servidor*" con:**
  - **Java (Servlets, JSP, Javabeans y EL).**



Contenido del Tema (Teoría + Práctica)

1. Introducción: Programación web dinámica **en Servidor**
   - Motivación
   - Clasificación: entorno ejecución DENTRO/FUERA servidor

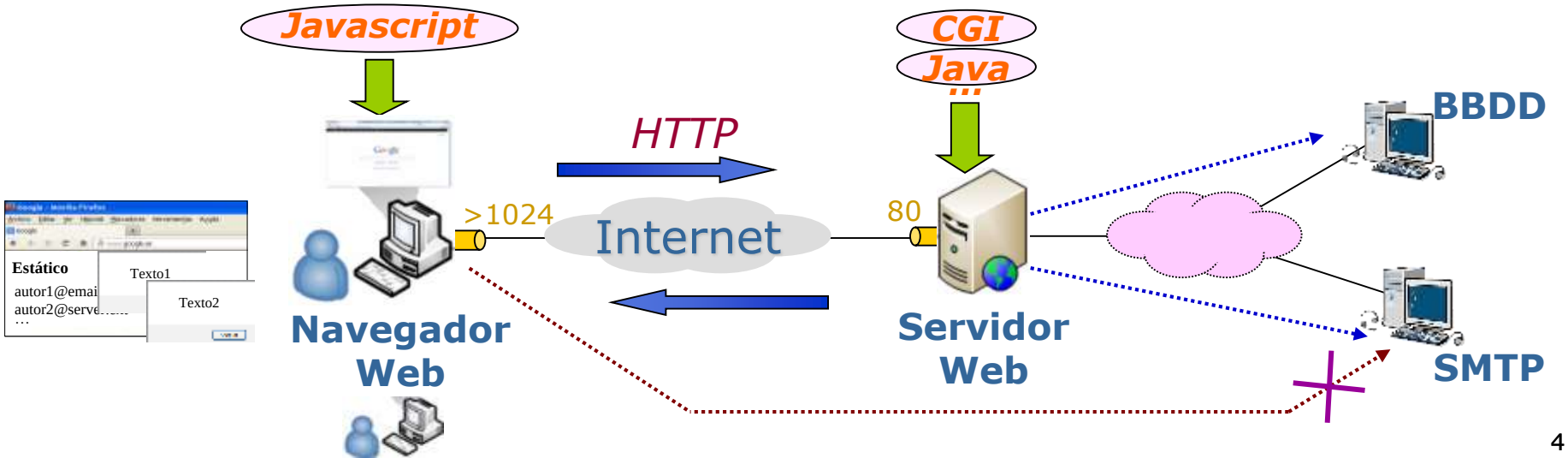
2. Conceptos comunes (a cualquier Lenguaje Web en Servidor)
   - Nº Lenguajes/fichero web
   - Uso protocolo HTTP
   - Tipos de información en servidor
   - Técnicas para Sesiones

3. Aplicaciones Web **FUERA** del servidor: Interfaz CGI

4. Aplicaciones Web **DENTRO** del servidor: Java
   - Servlets
   - JSP: Etiquetas de Scripts, Javabeans
   - EL (Expression Language)

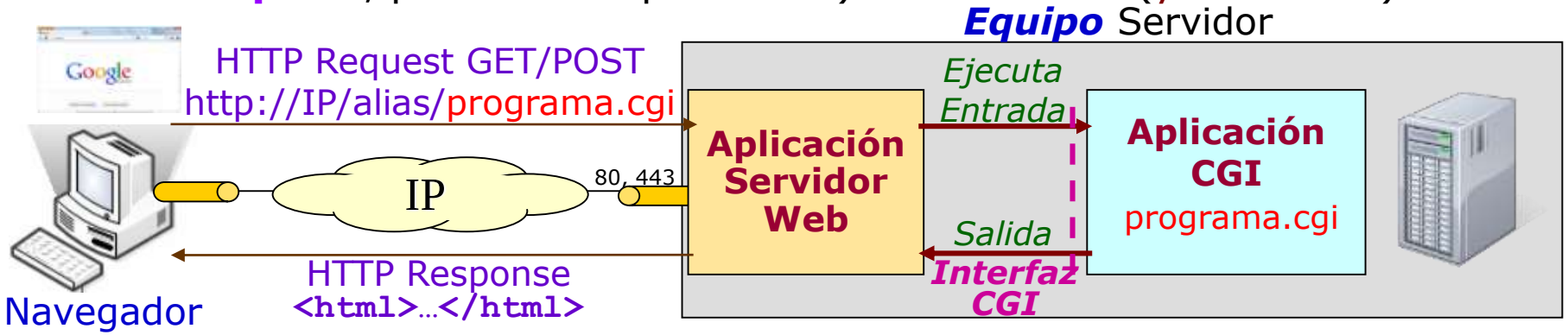
Introducción: Motivación

- Programación web dinámica "en servidor" vs "en cliente":
  - **En Cliente** (Javascript, …): útil para código que interesa se ejecute en el navegador (eventos, comprobar campos formularios, recargas páginas, …)
  - **En Servidor** (CGI - Common Gateway Interface, Java, PHP, …): útil para código que interesa se ejecute en el servidor (chats entre usuarios, acceso a servidores BBDD, de e-mail, …)
    - No depende del navegador; pero requiere permisos del servidor (problemas de ***sobrecarga y seguridad***).

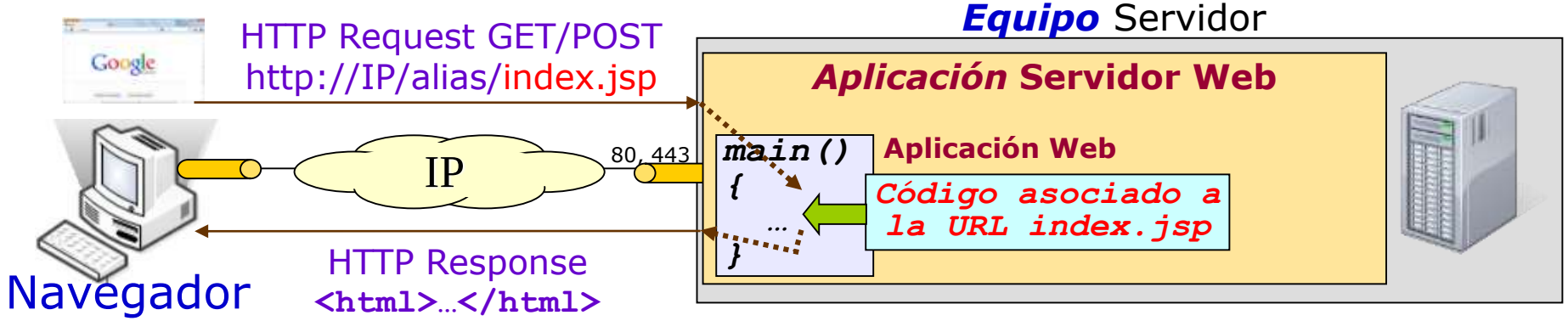


Introducción (2): Clasificación por entorno ejecución

- Lenguajes Web en servidor: 2 grupos según Aplicación Web ejecutada
  - ***FUERA*** del Programa Servidor Web (Aplicación CGI es un **programa completo**, proceso independiente): interfaz CGI (y derivadas).



  - ***DENTRO*** del Programa Servidor Web (**fragmento de código** integrado en el del servidor): demás (Java, PHP, ASP, …).



Introducción (3): Clasificación por entorno ejecución

Tipo	Lenguajes Web de Programación dinámica en Servidor
Ejecutados FUERA del servidor	❑ **CGI** (Common Gateway Interface): Interfaz de comunicación ***entre*** Servidor Web y ***programas externos en cualquier lenguaje***: ➢ Interpretado: Shell-script, Java, … ➢ Compilado: **C**, C++, …
Ejecutados DENTRO del servidor	❑ **Java** (Servlets, JSP, …): Lenguaje Java, abierto (Sun Microsystems) ❑ **PHP** (PHP Hypertext Preprocessor): Abierto (Fundación Apache). Sintaxis similar a "C". ❑ **ASP** (Active Server Pages): Microsoft ❑ **Clásico**: Multilenguaje JScript, VBScript, Perl (abierto). ❑ **ASP.NET**: Multilenguaje C#, VB.NET, … ❑ **SSJS** (Server-Side JavaScript=ECMAScript):node.js ❑ Otros: **Groovy, ColdFusion, Ruby, Python, …**

Introducción (4): uso

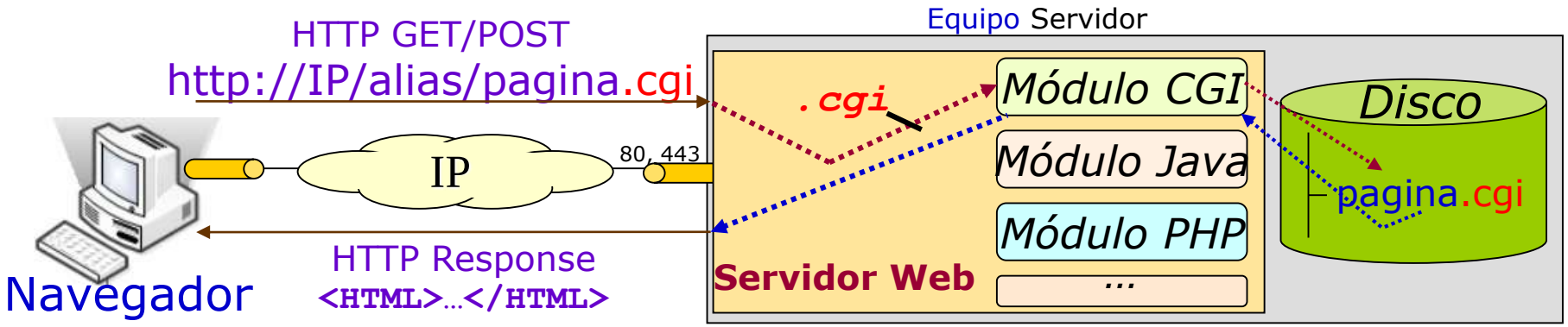
- CGI (Shell-script, C,…): reutilizar código, conocimiento lenguaje
- Java: empleado principalmente en Webs:
  - De complejidad media/alta: reutilización clases Java.
  - De gran tamaño (gran volumen tráfico, separando Presentación-Lógica): TagLibs, JSF.
- PHP: usado principalmente en Webs simples (numerosas).

Aspectos comunes: Nº lenguajes/fichero

- Cada fichero Web sólo "1" lenguaje en servidor (CGI, o JSP, …):

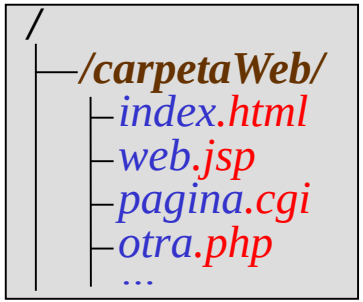
	Estático	Dinámico cliente	Dinámico servidor
Nº lenguajes	1	0, 1 ó varios	0 ó 1

  - ***Motivo***: el servidor web usa la URL (la analiza) para determinar el módulo (PHP, JSP, ...), según su configuración.



URL <≠> fichero/programa

- Aunque poco usual, una misma Aplicación Web sí podría usar varios Lenguajes (en distintos ficheros):



Aspectos comunes (2): Uso protocolo HTTP

- Funcionamiento Aplicaciones Web en servidor muy ligado a HTTP:
  - Distinto comportamiento según ***método*** HTTP Request:
    - GET (por omisión), POST (formularios HTML), HEAD, TRACE, OPTIONS, PUT, DELETE

```
<a href="http://www.google.es">Google</a>
```
⟸ *HTTP Request* ***GET***

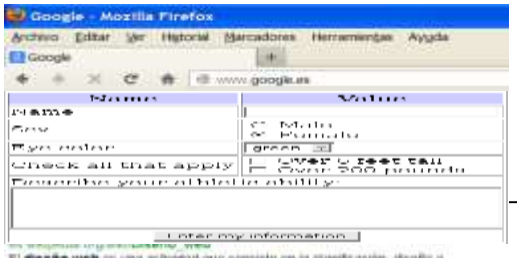
```
<form action="pagina.cgi" method="post">
```
⟸ *HTTP Request* ***POST***

```
XMLHttpRequest.open("método_HTTP","p.cgi")
```
⟸ *HTTP Request mét* ***xxx***

  - Distinta respuesta según ***cabeceras*** HTTP Request (e.g. User-Agent, …).
  - Generación de parte de HTTP Response (estado, cabeceras, cuerpo).
  - Formularios: datos usuario en línea petición (GET) o cuerpo (POST).

Aspectos comunes (3): Ejemplo POST

URL:
http://IP/alias/pagina.cgi



Navegador

```
POST /alias/pagina.cgi  http/1.1
Host: IP
User-agent: Chrome/4.0
Accept-language: es
Content-Length: 232
Content-Type: application/x-www-form-urlencoded
.....
par1=var1&…
```

Obligatoria con cuerpo

Línea en blanco

Cuerpo (datos_formulario)

HTTP Request ***(con datos formulario)***

IP

pagina.cgi

Servidor Web

HTTP Response ***(con cuerpo)***

```
HTTP/1.1 200 OK
Connection: close
Data: Thu, 03 Jul 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 5 May 2006 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
.....
<!DOCTYPE…
```

Obligatoria con cuerpo

Línea en blanco

Cuerpo (pagina_HTML)

Aspectos comunes (4): GET vs POST

GET — http://www.servidor.ext/alias/pag.cgi?name1=val1&name2=val2&...

```
GET /alias/pag.cgi?name1=val1&name2=val2&… HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.07 [en] (X11; I; Linux 2.2.15 i586; Nav) ...
Host: www.servidor.ext
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, i ...
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8

```

ruta(+ Datos del usuario)

IP/nombre extraído de la URL

Línea en blanco

POST - http://www.servidor.ext/alias/pag.cgi

Para enviar ficheros, multipart, …
usar POST (límite URL…)

```
POST /alias/pag.cgi HTTP/1.0
User-Agent: Mozilla/4.07 [en] (X11; I; Linux 2.2.15 i586; Nav) ...
Host: www.servidor.ext
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

name1=val1&name2=val2&…
```

ruta

IP/nombre extraído de la URL

MIME del contenido

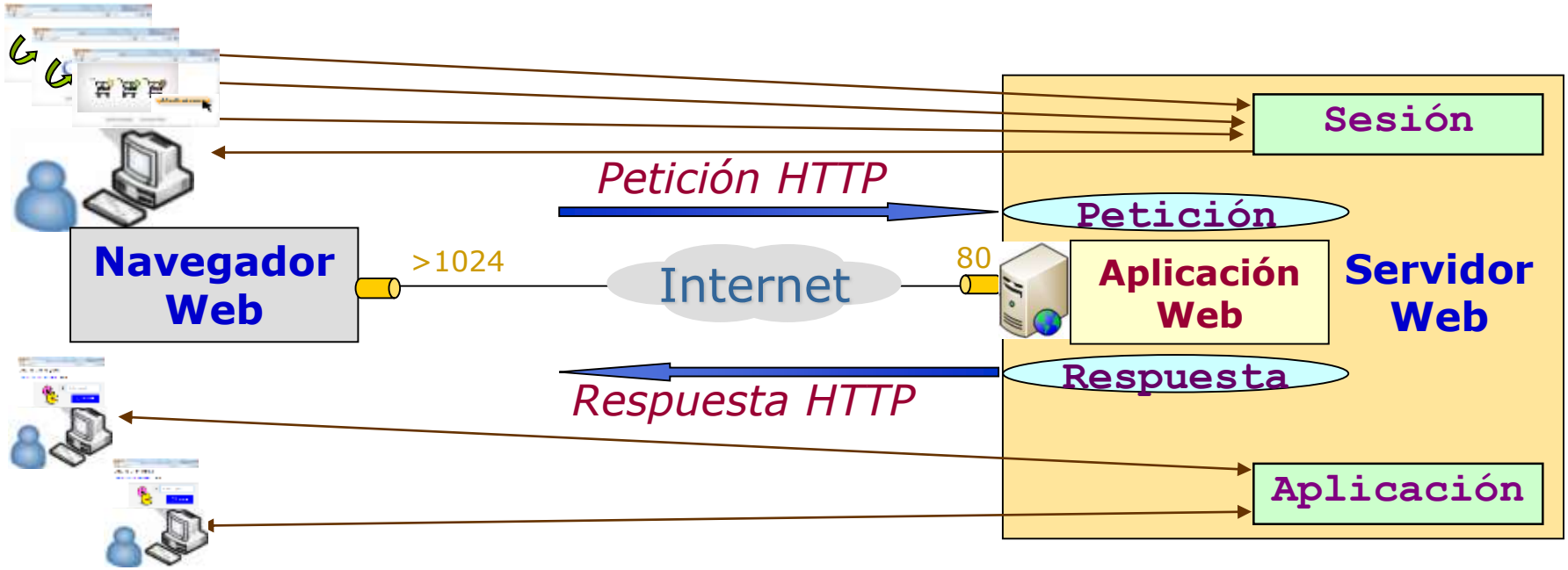
Línea en blanco

Datos del usuario

Aspectos comunes (5): Tipos de información

- Tipos de información usados por Aplicación Web en Servidor:

Ámbito/ Alcance	*Información*
Petición	Recibida en la petición HTTP (cabeceras HTTP, datos, …)
Respuesta	Para construir la respuesta HTTP (cabeceras HTTP, datos, …)
Sesión	Que persista cuando el usuario pasa de una página a otra de la misma web (Ej: cesta compra)
Aplicación	A compartir entre usuarios que acceden a la misma Web (Ej: chat)

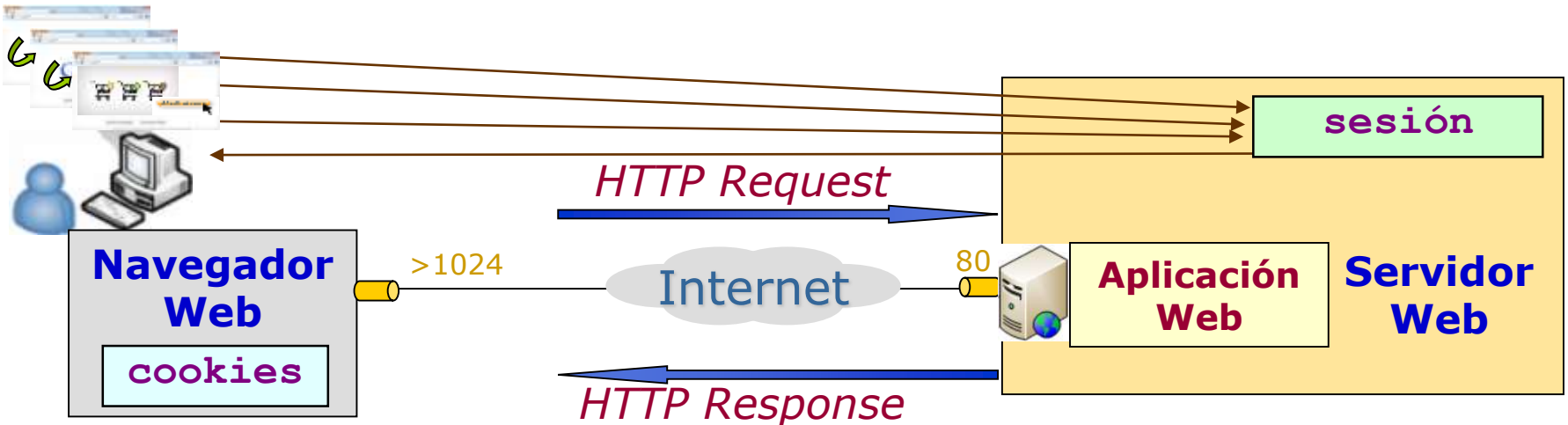


Aspectos comunes (6): Técnicas para Sesiones

- **Sesión**: información que persiste cuando el usuario pasa de una página a otra de la misma web (**1 sesión por Navegador y Web**)
  - Múltiples ***usos***: carrito compra, control de autenticación, ...
  - **HTTP** es **sin estado** (no guarda información de las solicitudes HTTP anteriores) => Debe implementarse en la página web (***sesión***).
  - Información de sesión: conjunto de "parámetros" y sus valores.

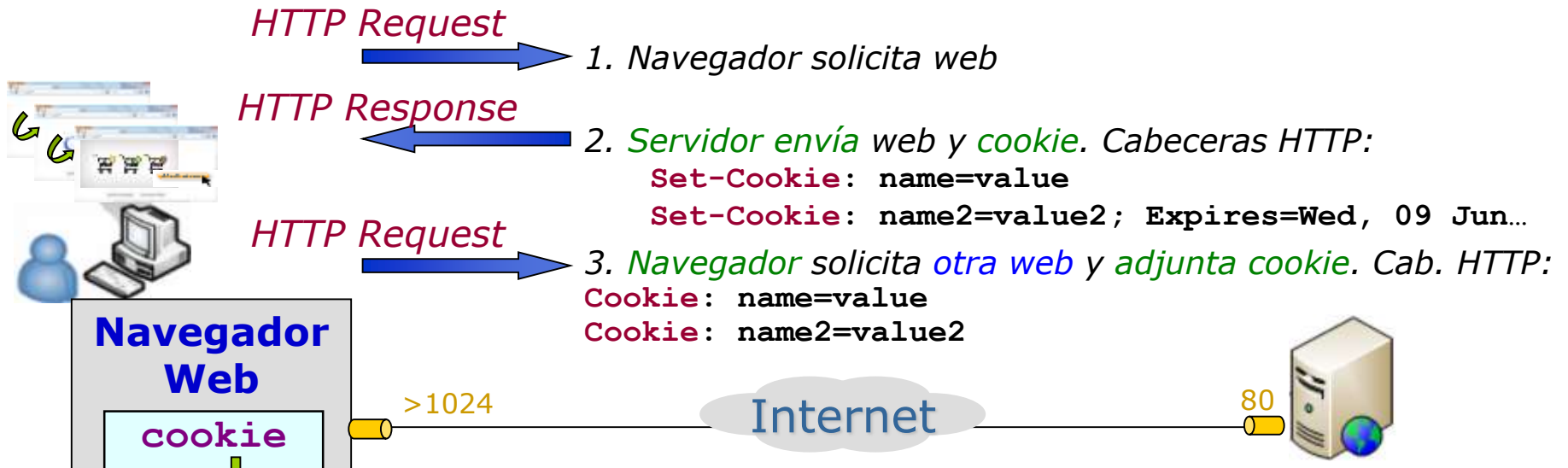
```
par1=value1
par2=value2; Expires=Wed, 09 Jun…
…
```

- Técnicas para sesiones: (2 formas)
  - Sesión almacenada en el navegador: Cookies
  - Sesión almacenada en el servidor.



Aspectos comunes (7): Técnicas Sesiones, Cookies

- Sesión en el navegador: "cookies" (RFC 6265)
  - Cookie: archivos de texto en el navegador con datos sesión.
  - Poco seguro y depende de que estén habilitadas en el navegador.
  - Funcionamiento:
    0º Navegador [en Solicitud] solicita web.
    1º Servidor [en Respuesta] envía cookie (datos sesión).
    2º Navegador [en Solicitud] ***reenvía*** cookie (datos sesión) al servidor.



Aspectos comunes (8): Técnicas Sesiones, En servidor

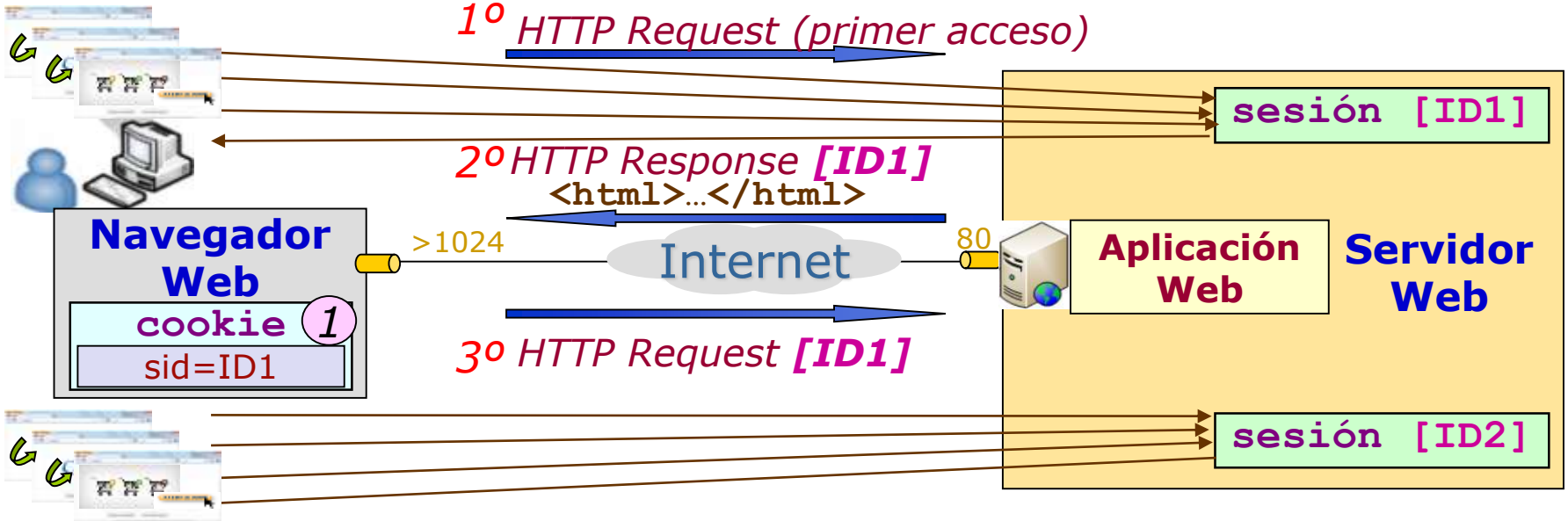
- Sesión en el servidor: *sesión* identificada por un ID.
- Sin cookies: métodos para indicar el ID de **sesión** de una petición:

1. *Reescritura URL* (GET)

```
<a href="http://IP/pag.cgi?sid=123">W</a>
<form action="p.cgi?sid=123" method="post">
```

2. *Formularios HTML ocultos (o campos) en la página* (GET o POST):

```
<form action="pag2.cgi" method="post">
   <input type="hidden" name="sid" value="123">
   <input type="submit" value="Comprar"></form>
```



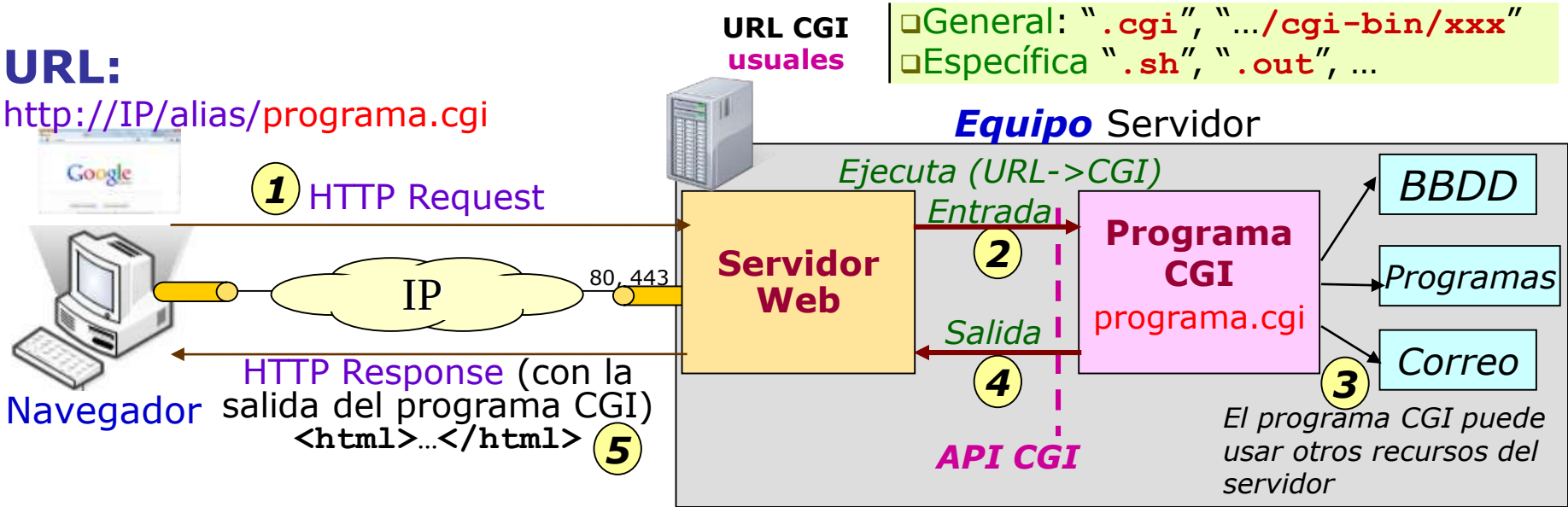
Entorno de ejecución *FUERA* del servidor web: CGI

CGI: Índice

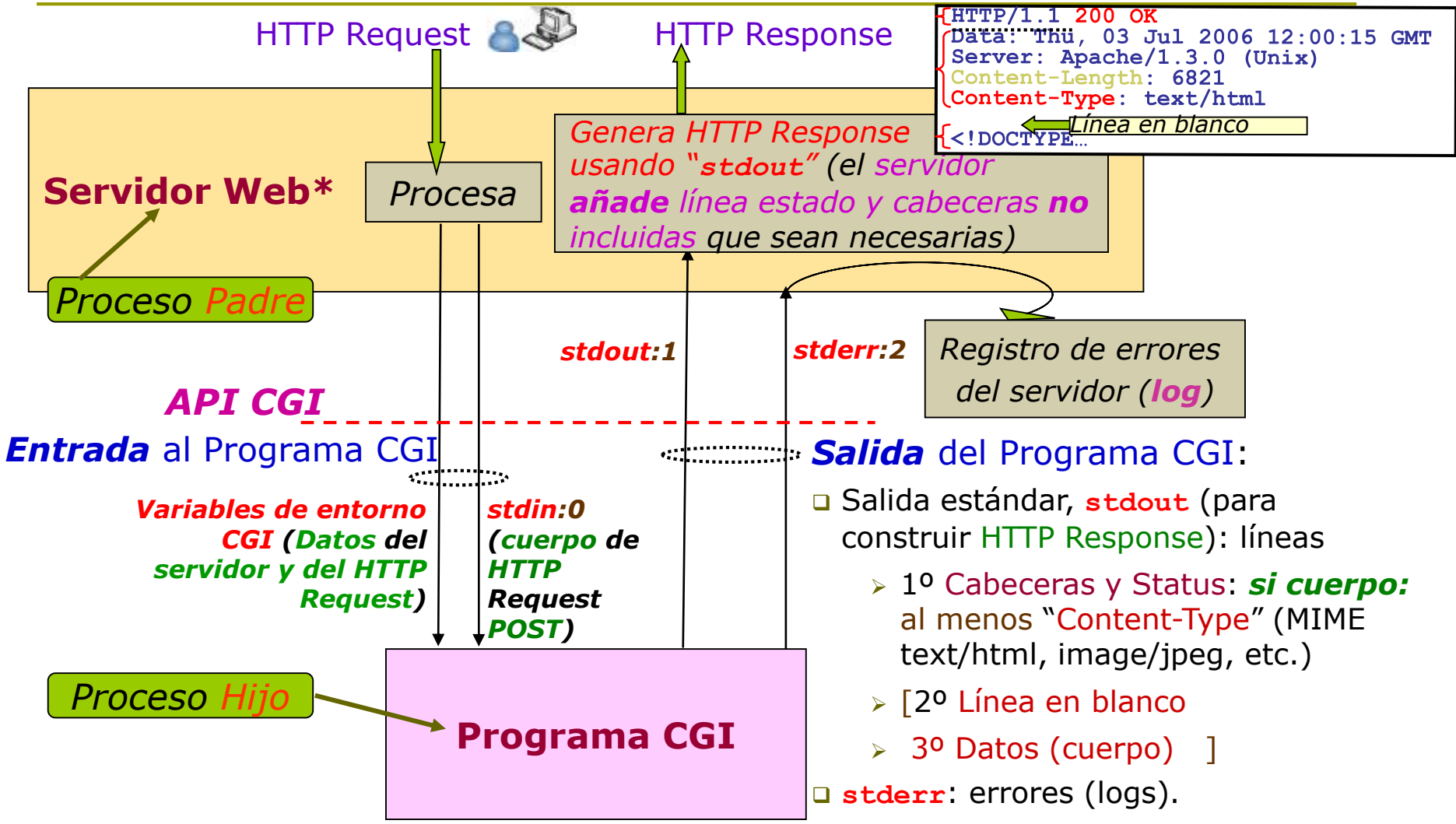
1. Introducción
2. Especificación CGI
   - API CGI
   - Ejemplo CGI
3. Inconvenientes de CGI / Derivados

CGI: Introducción

- **Programa/API CGI** (Common Gateway Interface 1.1 – RFC 3875):
  - En cualquier lenguaje (Shell-script, C, Java, Python, etc.).
    - Existen librerías adicionales para simplificar el desarrollo.
  - ***Completo/autónomo*** (en C tendría "`main()`"), pudiendo ser ejecutada sin que hubiese servidor web (proceso independiente).
  - Encargada de (analizar los datos recibidos del navegador y de) imprimir en su salida estándar la página de respuesta al navegador.
  - ***Ejecución: FUERA*** del Servidor Web



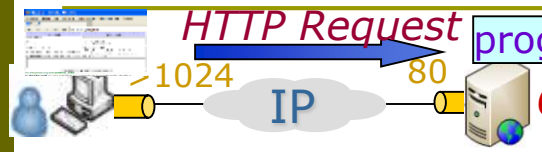
CGI: API CGI



- Salida estándar, `stdout` (para construir HTTP Response): líneas
  - 1º Cabeceras y Status: ***si cuerpo:*** al menos "Content-Type" (MIME text/html, image/jpeg, etc.)
  - [2º Línea en blanco
  - 3º Datos (cuerpo)]
- `stderr`: errores (logs).

* *El servidor web invoca al CGI redirigiendo la entrada estándar y salidas estándar y de error.*

CGI: Entrada CGI: Datos desde formularios



QUERY_STRING

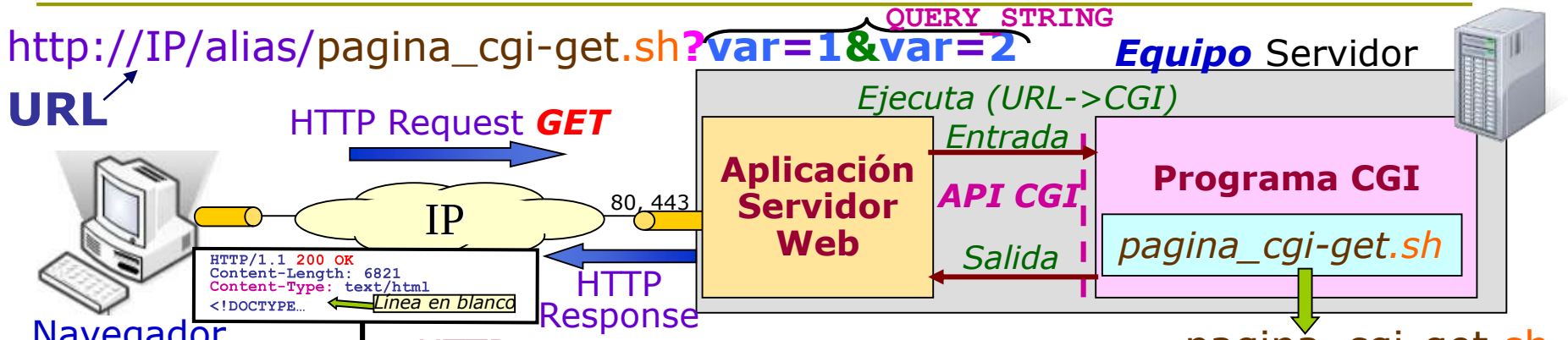
GET http://IP/alias/prog.cgi?name1=valor1&name2=valor2&

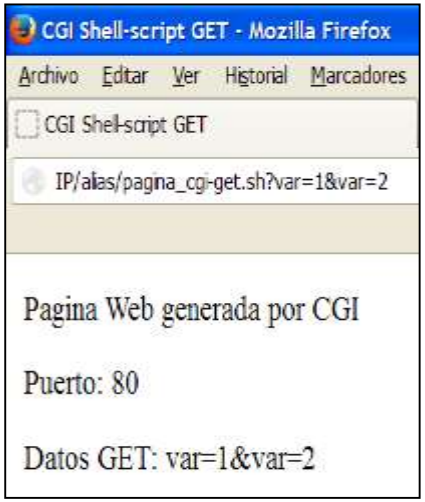
Entrada al programa CGI	Contenido de la entrada ante **HTTP Request**	
	GET *(enlaces y formul.)*	**POST** *(sólo formul.)*
QUERY_STRING	Datos del formulario (cadena "nameX=valorX&…" tras "?")	Vacía
stdin	Vacía	**Cuerpo HTTP**: Datos formulario: cadena "nameX=valorX&…", ficheros
CONTENT_TYPE	Sin definir	Tipo MIME de los Datos
CONTENT_LENGTH		Longitud del cuerpo (de los Datos, en bytes)

POST http://www.servidor.ext/alias/prog.cgi [Datos en ***cuerpo***: par1=valor1&…]

- Para extraer campos, programa CGI debe analizar cadena: `par1=val1&par2=val2&…`
- Es posible GET+POST: `<form action="p.cgi?p1=v1&p2=v2…" method="post">`

CGI: Ejemplo CGI Shell-script con GET





HTTP Response Cab.

Cuerpo

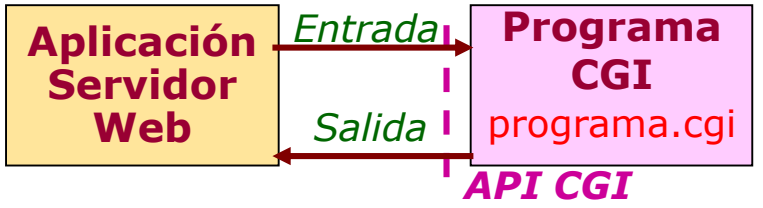
```
#!/bin/sh
echo "Content-type: text/html"
echo                          ← Línea en blanco
echo "<!DOCTYPE html>"
echo "<html><head><title>"
echo "CGI Shell-script GET</title></head><body>"
echo "<p>Pagina Web generada por CGI</p>"
echo "<p>Puerto:     $SERVER_PORT</p>"
echo "<p>Datos GET:  $QUERY_STRING</p>"
echo "</body></html>"
```

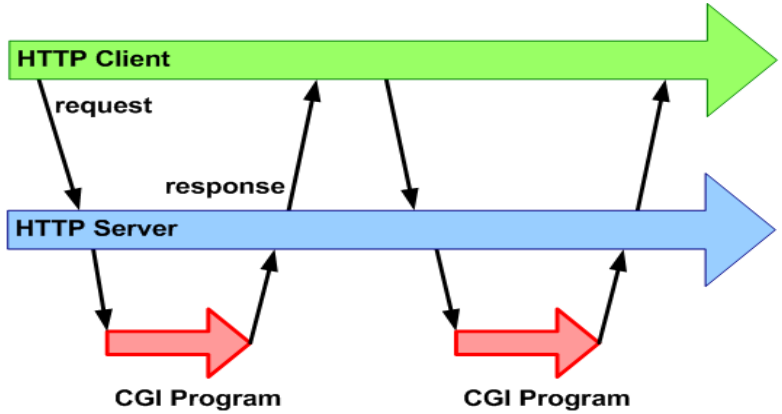
Código HTML que genera

```
<!DOCTYPE html>
<html><head><title>
CGI Shell-script GET</title></head><body>
<p>Pagina Web generada por CGI</p>
<p>Puerto:     80</p>
<p>Datos GET:  var=1&var=2</p>
</body></html>
```

CGI: Inconvenientes / Derivados

- Inconvenientes API CGI:
  - Comunicación limitada entre servidor y programa CGI.
  - Poco eficiente y respuesta "lenta": por cada HTTP Request a un programa CGI, se crea una instancia nueva del programa en memoria.



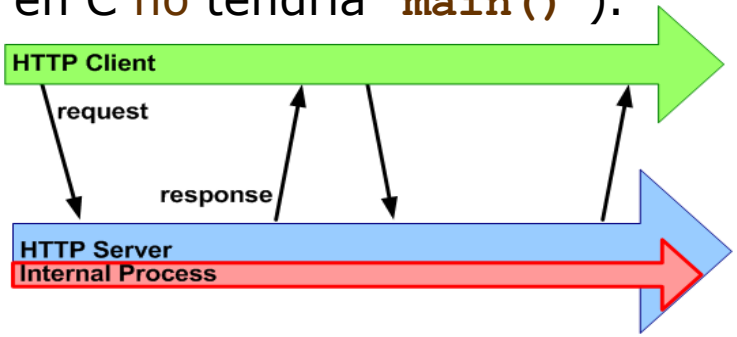


- Derivados CGI (mejoran comunicación y admiten varias peticiones/instancia CGI):
  - FastCGI, SCGI (Simple CGI), WSGI (Web Server Gateway Interface)

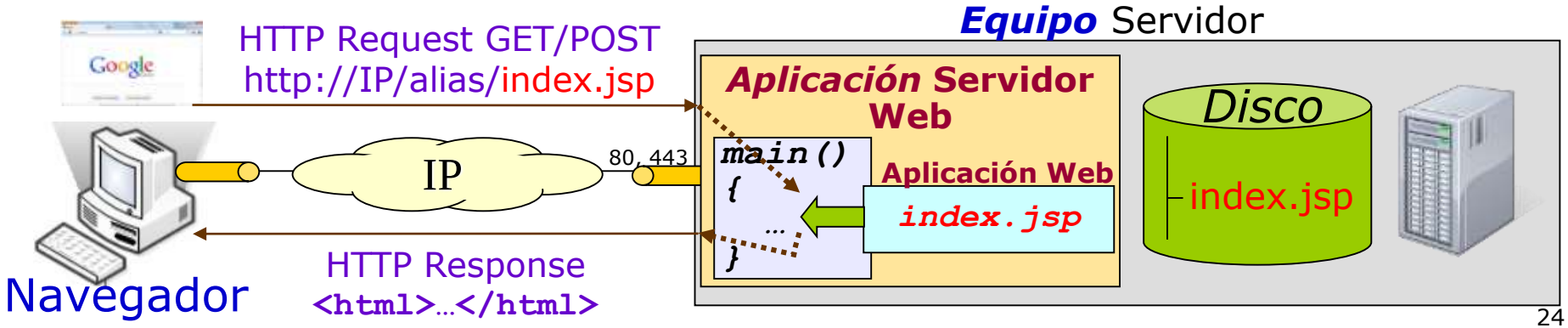
Entorno de ejecución *DENTRO* del Servidor Web: Aplicaciones Web Java

Entorno de ejecución DENTRO del Servidor Web

- Aplicación web ejecutada ***por el propio servidor*** web: Java, PHP, ASP,…
  - La aplicación Web es un **fragmento de código** integrado en el del servidor (NO es un programa completo, en C no tendría "`main()`").
  - **Sólo 1 proceso**: el del servidor web (que es quien interpreta la aplicación web).

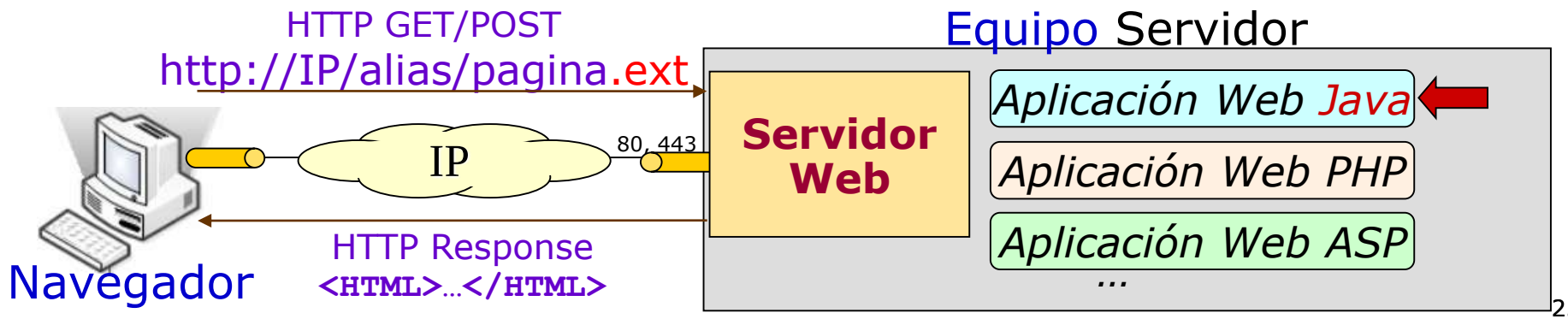


  - La aplicación servidor web crea un "entorno" (objetos implícitos, …) para la interpretación del código de la página web.



Java (Web en Servidor): Ventajas

- Aplicaciones Web ***Java*** (***vs*** PHP, ASP, …): Ventajas derivadas de Java:
  - Lenguaje conocido por elevado número de programadores.
    - => Estudiaremos funcionamiento, no lenguaje
  - Reutilización de software: Librerías Java
    - Ej.: JDBC, acceso a ficheros, ejecución de comandos, e-mail, …
  - Independencia de la plataforma (Máquina Virtual Java).
  - Múltiples tecnologías (Servlets, JSP, JSF) y potentes (separación Presentación/Lógica no ofrecida por otros lenguajes).



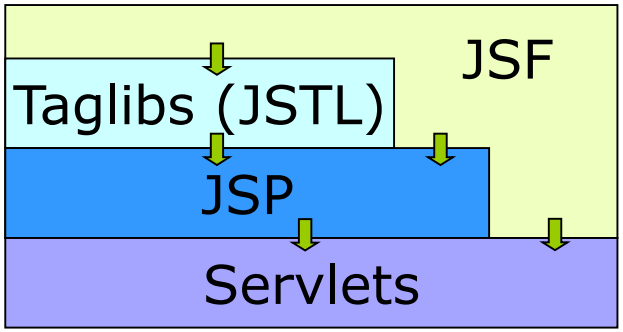
Java (Web en Servidor): Tecnologías

- ***Tecnologías*** Java para Programación Web (dentro servidor):

Técnica		Funcionalidad
➡ Servlets		Imprimir HTML desde Java
JSP (Java Server Pages)	➡ Etiquetas de Scripts	*Insertar* Java dentro de HTML
	➡ Javabeans	*Invocación simplificada* clases Java desde HTML
	Taglibs (y librería estándar JSTL "JSP Tag Library")	*Definir nuevas etiquetas* "personalizadas" e.g.: `<enviar_email direcc="u@s.ext">`
JSF (Java Server Faces), Facelets		Librerías de etiquetas para Simplificar el desarrollo
➡ EL (Expresssion Language, JSP y JSF)		Sintaxis simplificada para insertar expresiones Java en HTML

Mayor **separación** presentación (html) y lógica (java)

- Adicionalmente:
  - Frameworks: Struts, Spring, …
  - Otras API: JDBC (DB SQL), JAXP (XML Processing), JavaMail, …



Versiones actuales
(Java EE 7 - JSR342):
- *Servlets 3.1 (JSR340)*
- *JSP 2.1 (JSR245)*
- *JSTL 1.2 (JSR52)*
- *JSF 2.2 (JSR344)*
- *EL 3.0 (JSR341)*

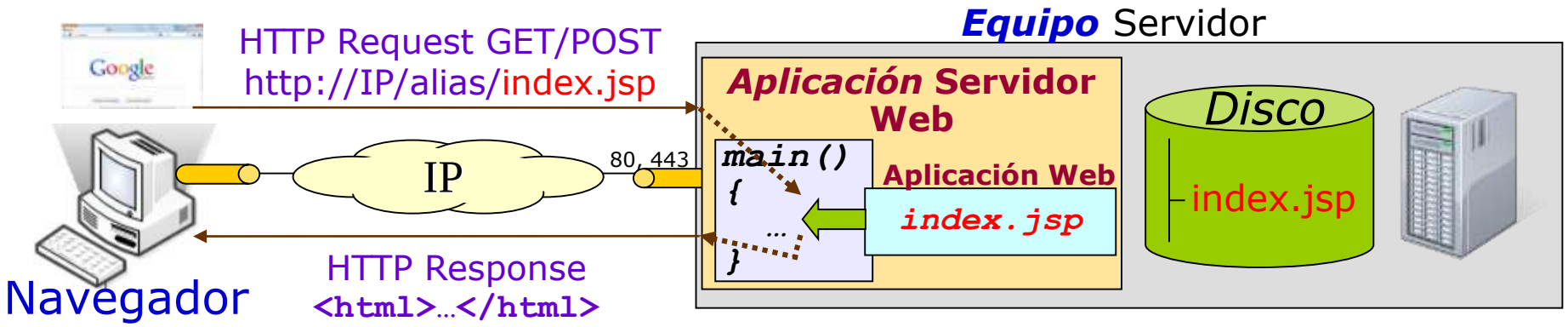
https://jcp.org/en/jsr/all

Java (Web en Servidor): Índice

1. API
   - Estructura de directorios de la Aplicación Web
   - Objetos implícitos del servidor
2. Servlets
   - Funcionamiento. Ciclo de vida
3. JSP Etiquetas de Scripts
   - Funcionamiento. Ciclo de vida
4. JSP Javabeans
   - Especificación
   - Invocación desde HTML
   - Funcionamiento
5. EL (Expresssion Language)
   - Objetivo y Sintaxis

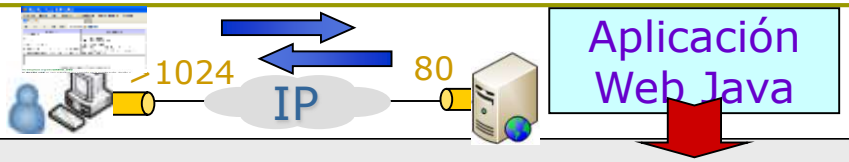
Java (Web en Servidor): API

- API J2EE (servlets, JSP, JSF): define
  - Estructura de directorios de la Aplicación Web.
  - Objetos implícitos del servidor: request, response, session, …
  - Etiquetas JSP posibles en HTML (por defecto) y cómo se definen otras (taglibs).
  - Clases Java (y métodos) que podemos definir y que el servidor invocará (ejecución DENTRO del servidor).



Java: API, Estructura Directorios

- Aplicación Web Java:




```
/carpeta_web/
    ├ index.html
    ├ pagina.jsp
    ├ /imágenes
    │   ├ graf.jpg
    ├ ...
    ├ /WEB-INF
    │   ├ web.xml
    │   ├ /classes/
    │   ├ /lib/
    │   ├ /tlds/
```

Parte ***Pública*** (visibilidad directa desde URL):

`http://IP/AppWeb/fichero.ext`

Parte ***Privada*** Java (acceso indirecto)

`web.xml` ← Descriptor de despliegue de la Aplicación (contiene características de la aplicación)

`/classes/` ← Clases Java ".class" propias (servlets, JB, ...)

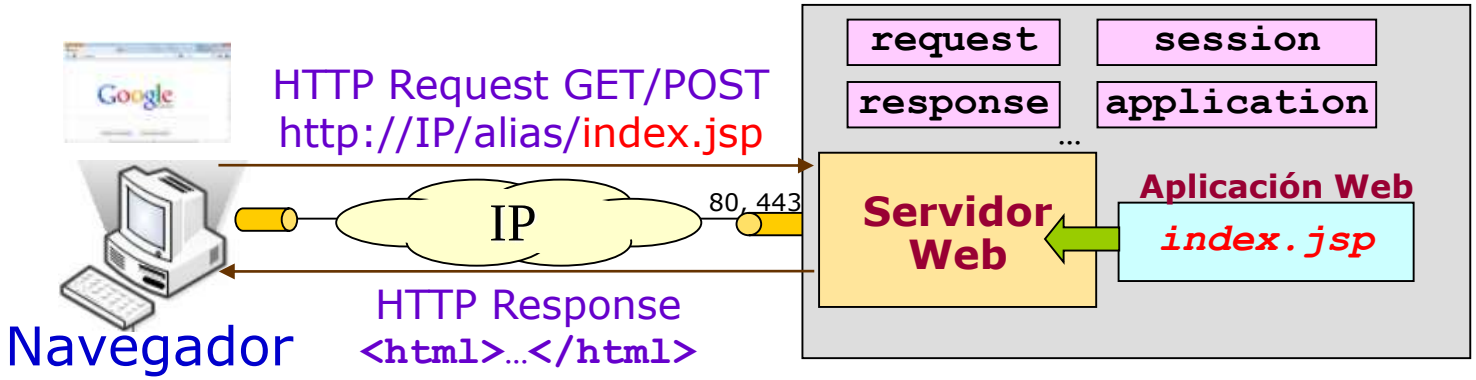
`/lib/` ← Librerías Java ".jar"

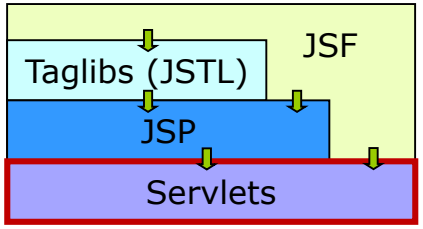
`/tlds/` ← Descriptores de "taglibs"

Java: APIs, Objetos Implícitos

- "**9**" Objetos implícitos: atender solicitudes a cada Aplicación Web
  - Servidor Web: ***crea*** instancias (por solicitud, sesión, …) y las ***pasa*** a Web
  - Aplicación Web: ***accede*** a las instancias (funciones, leer/escribir, …).

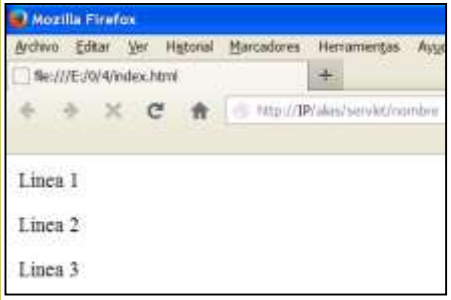
Objeto implícito	Interfaz	Contenido
`request`	HttpServletRequest	HTTP Request: tipo, cabeceras, cuerpo y otros
`response`	HttpServletResponse	HTTP Response: estado, cabeceras, cuerpo,otros
`out`	PrintWriter JspWriter	**Buffer** del objeto response (para escribir directamente en el cuerpo HTTP Response).
`session`	HttpSession	Información de sesión
`application`	ServletContext	Único por Aplicación Web, común para todos sus clientes.
config, pageContext, exception, page		





Servlets: Introducción

- **Servlet** (en Programación Web):
  - Clase Java de la Aplicación Web (invocada por el servidor web).
  - que analiza los datos recibidos del navegador e imprime la respuesta en el objeto "**`response`**" (*HTTP Response*).



http://IP/alias/servlet/nombre

HTTP Request

>1024

Internet

80

Navegador Web

HTTP Response

Servidor Web

clase_servlet.java

```
…
public class clase_servlet …
{…
 out.println("<html><head><title>Servlet" );
 out.println("</title></head><body>" );
 for( int i=1; i < 4; i++ )
    out.println("<p>Linea " + i + "</p>");
 out.println("</body></html>");
}
```

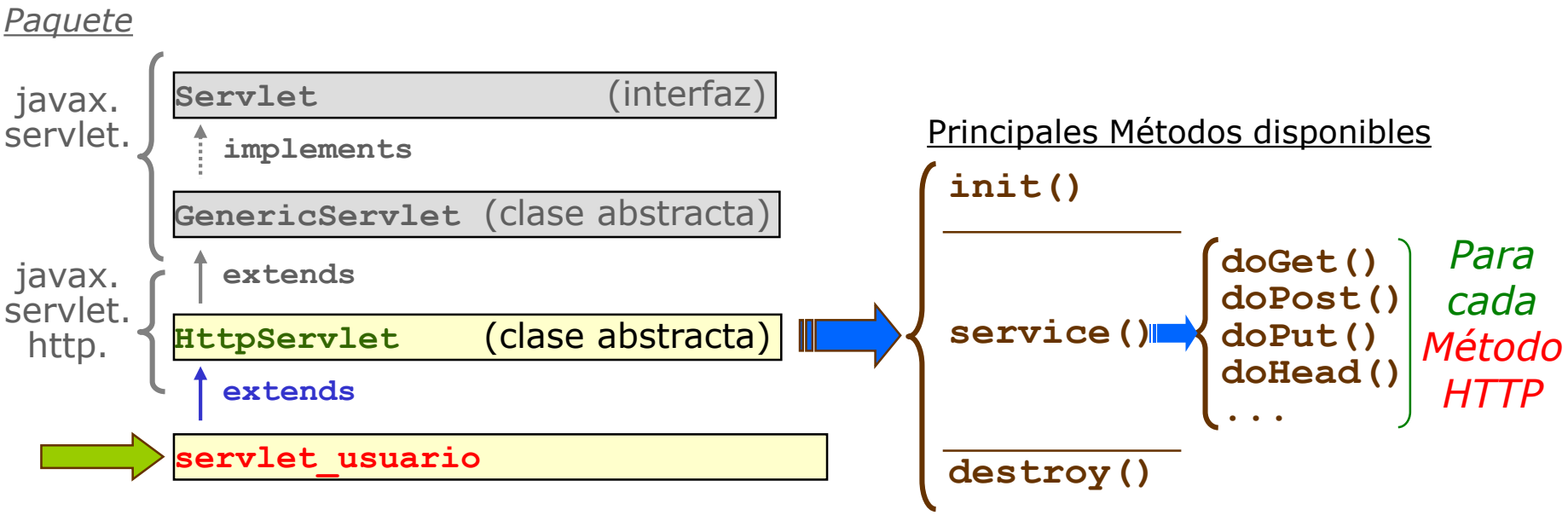
`response`

```
<html><head><title>Servlet
</title></head><body>
<p>Linea 1<p>
<p>Linea 2<p>
<p>Linea 3<p>
</body></html>
```

Código HTML que genera

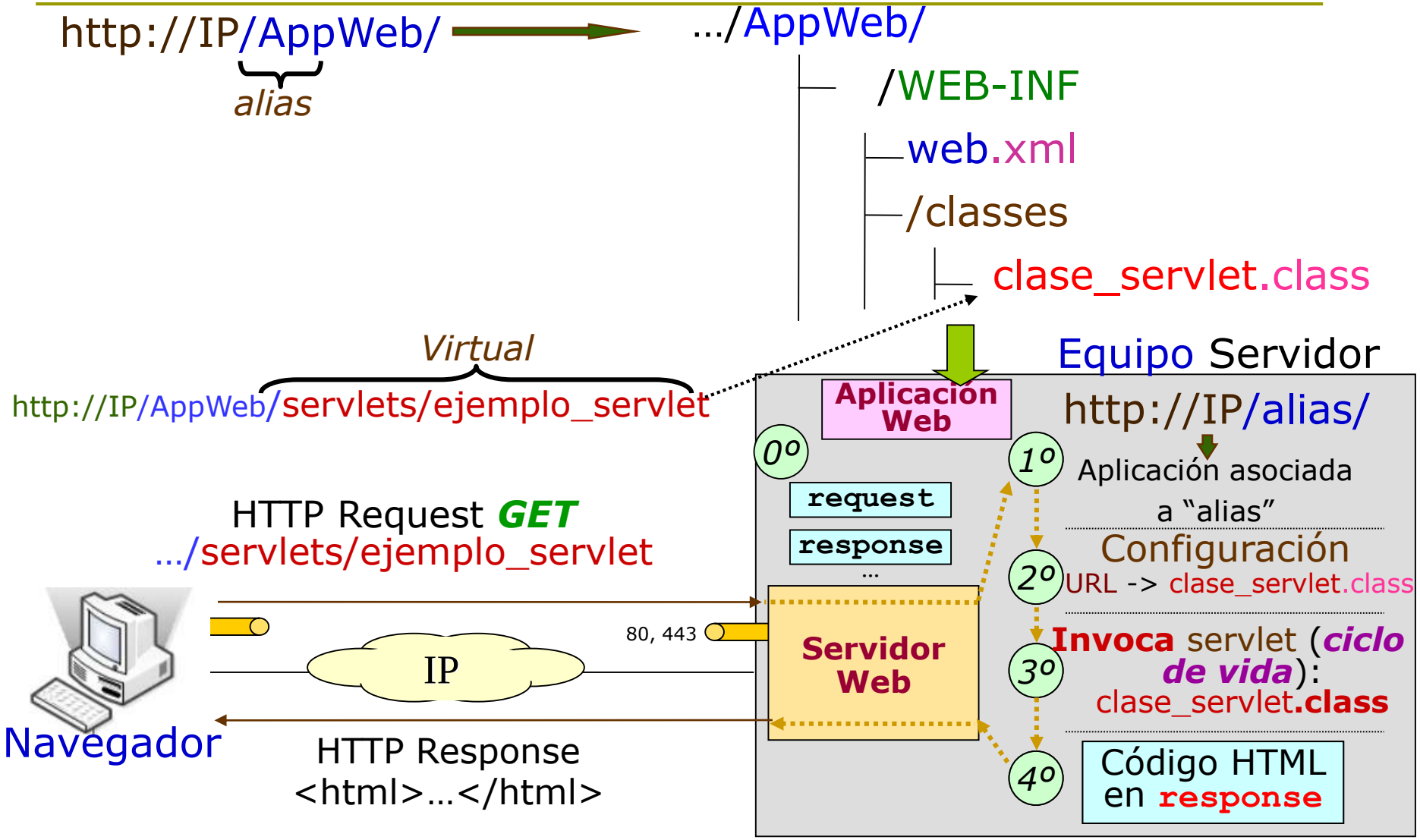
Servlets: Definición formal

- Servlet (en Programación Web): clase que deriva de la clase abstracta **`HttpServlet`** (y por tanto implementa la interfaz **`Servlet`**).



https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html

Servlets (de usuario): Funcionamiento

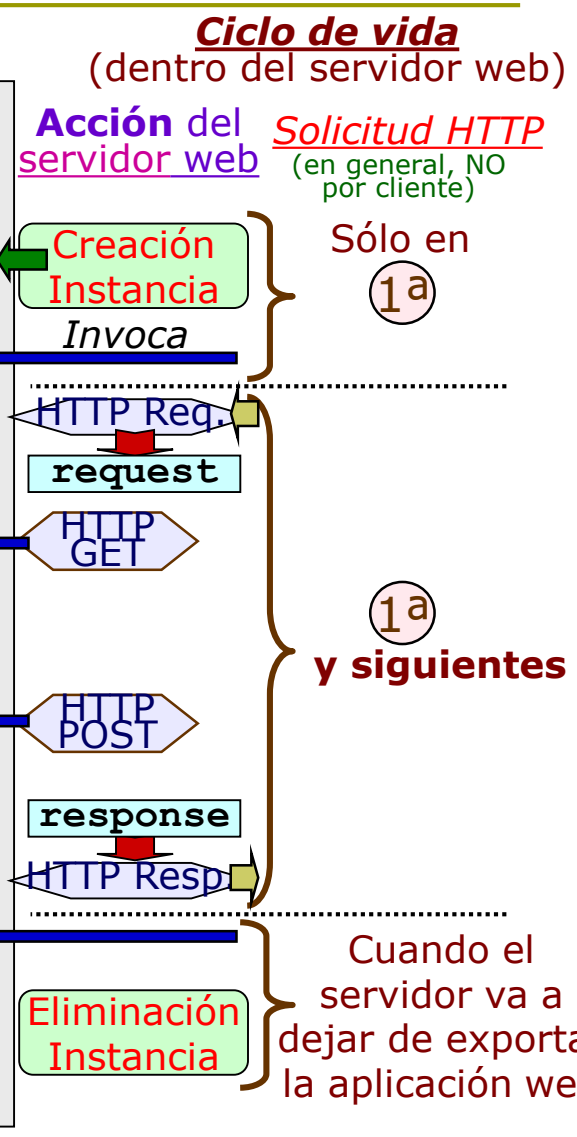


Servlets: Func. (2): Ciclo vida Servlet usuario

/WEB-INF/classes/clase_servlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;

public class clase_servlet extends HttpServlet
{
  public void init( ServletConfig conf )
  { // Ejemplo: Lectura de fichero
    ... }

  public void doGet (HttpServletRequest req,
                     HttpServletResponse res)
  { ... }

  public void doPost(HttpServletRequest req,
                     HttpServletResponse res)
  { ... }

  ...
  public void destroy()
  { // Ejemplo: Guardar en fichero ... }
 //…otros métodos y variables
}
```

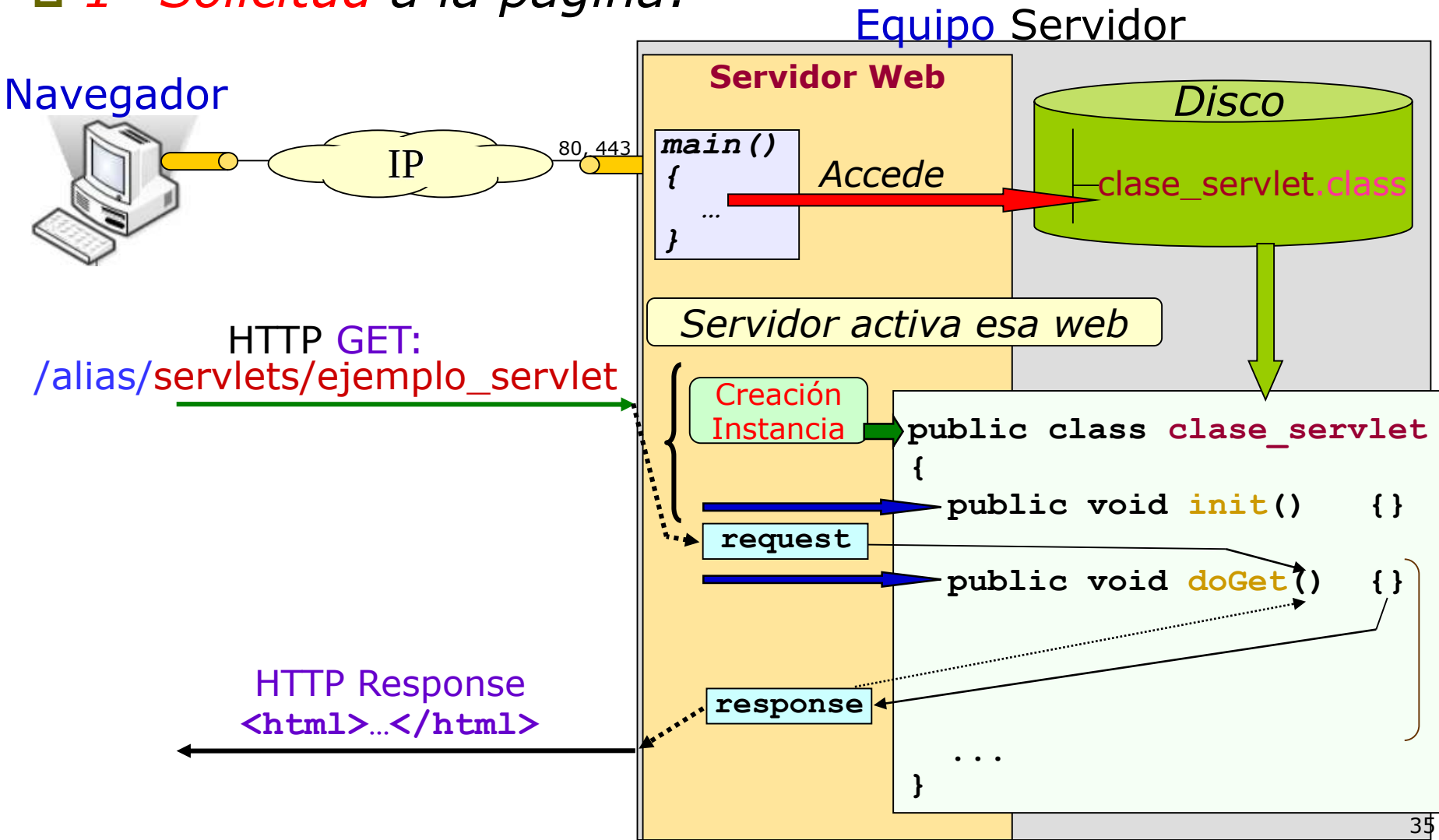
Referencias a Objetos: request, response



...

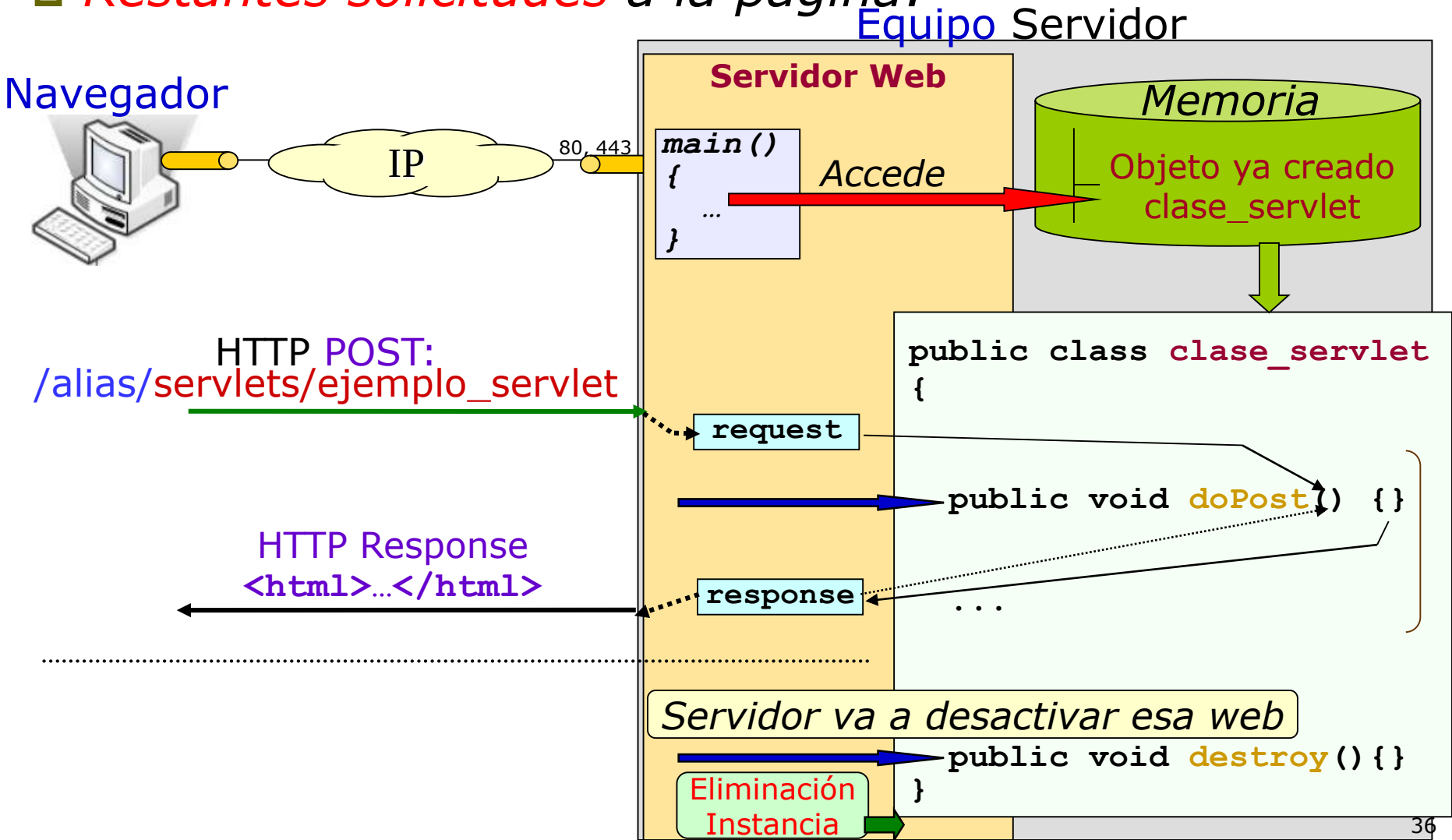
Servlets: Funcionamiento (3): Por solicitud

- *1ª Solicitud a la página:*



Servlets: Funcionamiento (4): Por solicitud

- *Restantes solicitudes a la página:*



Servlets: Configuración en web.xml

/WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app …>

  <servlet-mapping>
    <servlet-name>nombre_servlet</servlet-name>
    <url-pattern>/servlets/ejemplo_servlet</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>nombre_servlet</servlet-name>
    <servlet-class>paq.clase_servlet</servlet-class>
  </servlet>
</web-app>
```

alias

URL: http://IP/AppWeb/servlets/ejemplo_servlet

Virtual

/home/dit/tomcat/webapps/AppWeb/WEB-INF/classes/paq/clase_servlet.class

http://IP/AppWeb/

Servlets: Configuración con anotaciones

- Anotaciones (@annotation) metadatos que se añaden en el código. Están disponibles en tiempo de ejecución. Se pueden buscar.

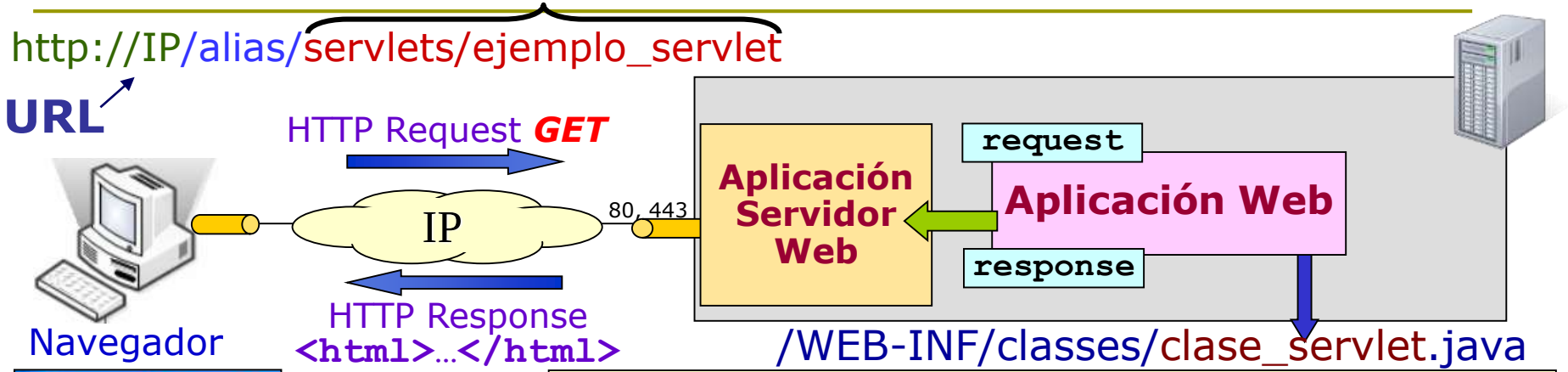
TestServlet.java

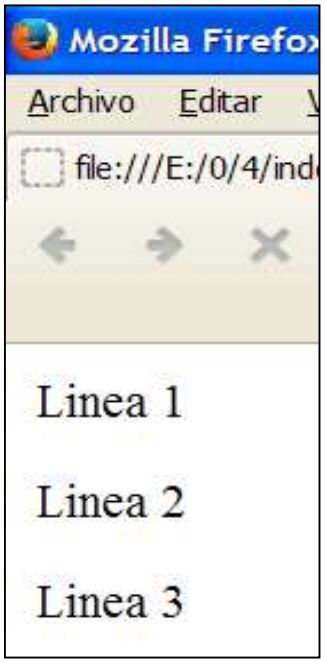
```
package fast.anotaciones;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

@WebServlet(urlPatterns={"/test", "*.test"})
public class TestServlet extends HttpServlet {
          @Override
          protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
                    …
          }
}
//http://docs.oracle.com/javaee/6/api/javax/servlet/annotation/WebServlet.html
```

Servlets: Ejemplo

http://IP/alias/servlets/ejemplo_servlet

URL






```
import javax.servlet.http.*;
public class clase_servlet extends HttpServlet
{ public void doGet (HttpServletRequest  req,
                     HttpServletResponse res)
  { ...   PrintWriter out = res.getWriter();
    out.println("<html><head><title>Servlet" );
    out.println("</title></head><body>" );
    for( int i=1; i < 4; i++ )
        out.println("<p>Linea " + i + "</p>");
    out.println("</body></html>"); }}
```

```
<html><head><title>Servlet
</title></head><body>
<p>Linea 1<p>
<p>Linea 2<p>
<p>Linea 3<p>
</body></html>
```

Código HTML que genera

Clase auxiliar: ServletContextListener

- Servlet Context ~= Aplicación Web en Java. Existen más tipos.
- Recibe notificaciones cuando la aplicación web arranca o para.

TestServletContextListener.java

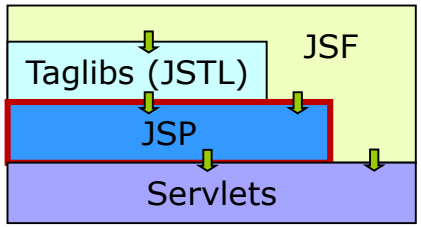
```
package fast.anotaciones;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class TestServletContextListener implements ServletContextListener {
  @Override
  public void contextDestroyed(ServletContextEvent contextEvent)      {
                    …
  }
  @Override
  public void contextInitialized(ServletContextEvent contextEvent) {
                    …
  }
}
//http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContextListener.html
```

Clase auxiliar: Filter

- Intercepta peticiones antes y después de ser procesadas. Ejemplos de filtrado:
  - Seguridad, bloquear el acceso a determinadas páginas.
  - Registro y auditoría de páginas consultadas.
  - Conversión de imágenes (escalado automático). Compresión de datos.
  - Servir distintas páginas según el idioma. Modificar las peticiones y respuestas.

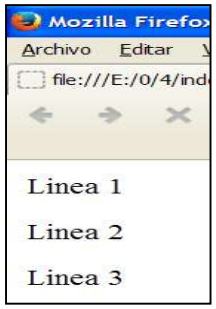
TestFilter.java

```
package fast.anotaciones;
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;

@WebFilter(urlPatterns = { "*.app", "/filtro" })
public class TestFilter implements Filter{
  @Override
  public void doFilter(ServletRequest r, ServletResponse rs, FilterChain chain)
                                throws IOException, ServletException    {
                        …
  }
}
//http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html
```

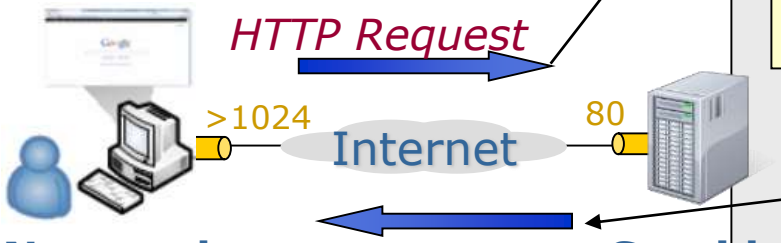


JSP-Etiquetas de Scripts

- Servlets: Codificación lenta si hay mucho código HTML
- JSP Etiquetas de Scripts: Insertar Java dentro de HTML
  - Etiquetas de Scripts: `<%, <%=, <%!, <%--, <%@, <jsp:`
  - Extensión por defecto: `.jsp, .jspx`
  - **Técnica similar a**: PHP, ASP, SSI, …



http://IP/alias/index.jsp



index.jsp

```
<html><head><title>Scriptlet
</title></head><body>
 <% for( int i=1; i < 4; i++ )
     out.println("<p>Linea "+ i +"</p>");
 %>
</body>
</html>
```

Equivale a:

```
<%= "<p>Linea " + i + "</p>" %>
```

response

```
<html><head><title>Scriptlet
</title></head><body>
<p>Linea 1<p>
<p>Linea 2<p>
<p>Linea 3<p>
</body>
</html>
```

Código HTML que genera

JSP-Etiquetas de Scripts: Funcionamiento

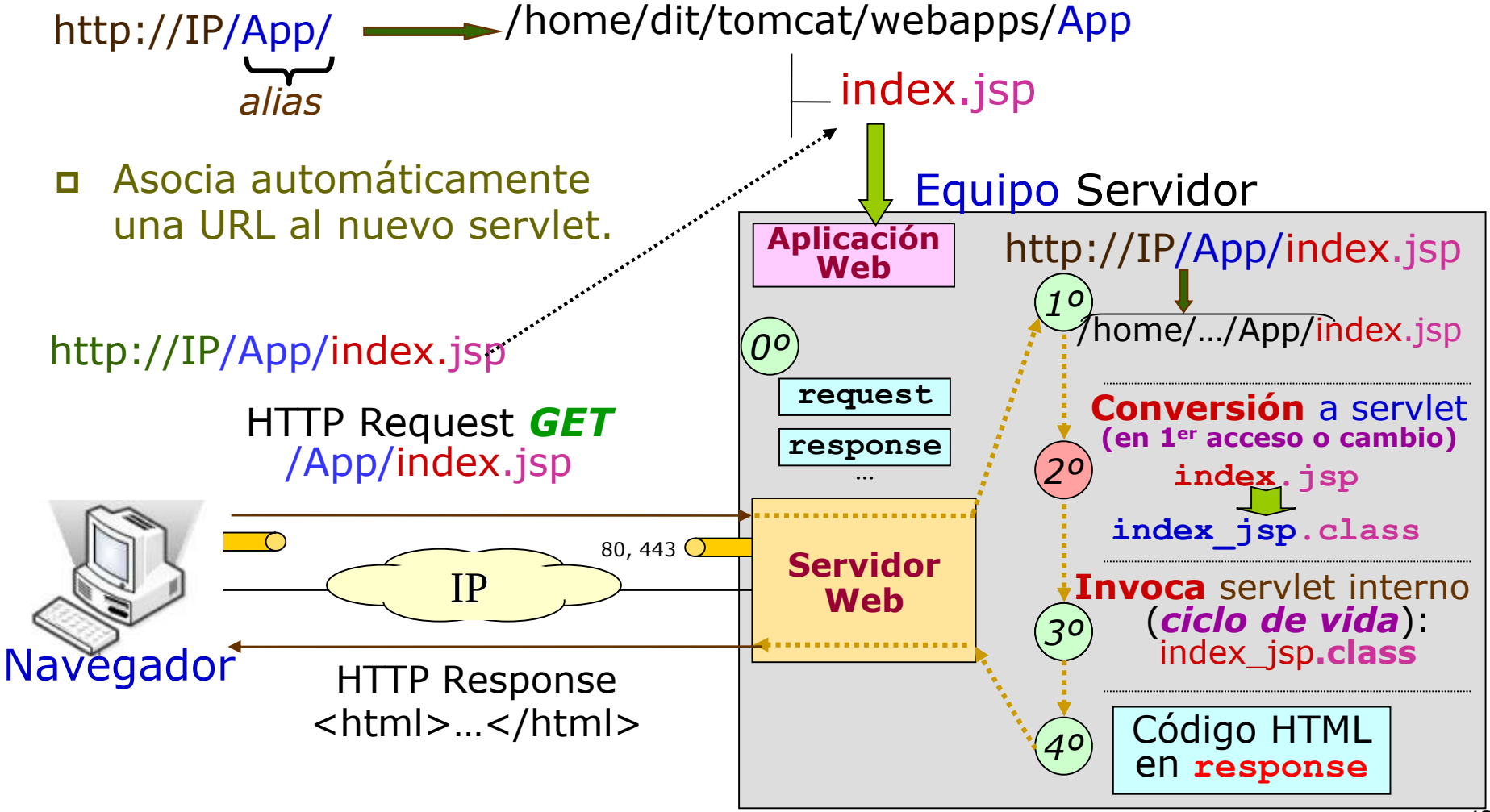
- Servidor Web convierte el fichero JSP en un servlet (interno).

http://IP/App/ → /home/dit/tomcat/webapps/App

alias

index.jsp

- Asocia automáticamente una URL al nuevo servlet.



Elementos de las páginas JSP

- Se pueden clasificar en 3 tipos:
  - Directivas: Afectan a la traducción
    - page <%@ page
    - include <%@ include
    - taglib <%@ taglib
  - Elementos de Script: Asociado al código
    - declaraciones <%!
    - expresiones <%=
    - scriptlets <%
    - comentarios <%--
  - Etiquetas de acción (Action Tags): controlan el funcionamiento
    - Obtener JavaBean <jsp:useBean
    - Obtener propiedad <jsp:getProperty
    - Fijar propiedad <jsp:setProperty
    - Incluir recurso <jsp:include
    - Reenviar petición <jsp:forward

JSP-Etiquetas de Scripts

Elemento	Etiquetas JSP de scripts	Funcionalidad

Directivas JSP	`<%@ page import="clase" %>`
	`<%@ include file="fichero"%>`
	…
		Importar clases de Java
		Insertar fichero (sin interpretar).
		…

Declaraciones	`<%! Declaraciones_Java; %>`	Declarar métodos o variables miembro en el servlet. Se pueden usar después en expresiones y scriptlets. Las variables miembro conservan su valor entre peticiones.
Expresiones	`<%= Expresión_Java %>`	Valor insertado dentro del código HTML (misma posición relativa al resto HTML). *Equivale habitualmente a:* `<% out.print(Expresión_Java) %>`
Scriptlets	`<% Código_Java_(posibles declaraciones_variables);%>`	Código Java (puede declarar variables locales, pero ***no*** funciones). Se ejecuta **en todos los accesos.**
Comentarios	`<%-- Comentario --%>`	Desaparece tras la interpretación JSP
Acciones JSP	`<jsp:include page="URL"/>` …	Insertar resultado de ejecutar otro servlet (puede ser la URL de otro JSP), reutilizando el objeto request. …

JSP: Scriptlets, Func. (2); Conversión JSP->Servlet interno

- "<%@ page import=" => import
- "<%!" => Variables/funciones miembro
- Def. funciones => Func. miembro
- **Lo demás** (incluyendo HTML) => en "_jspService()" (mismo orden)

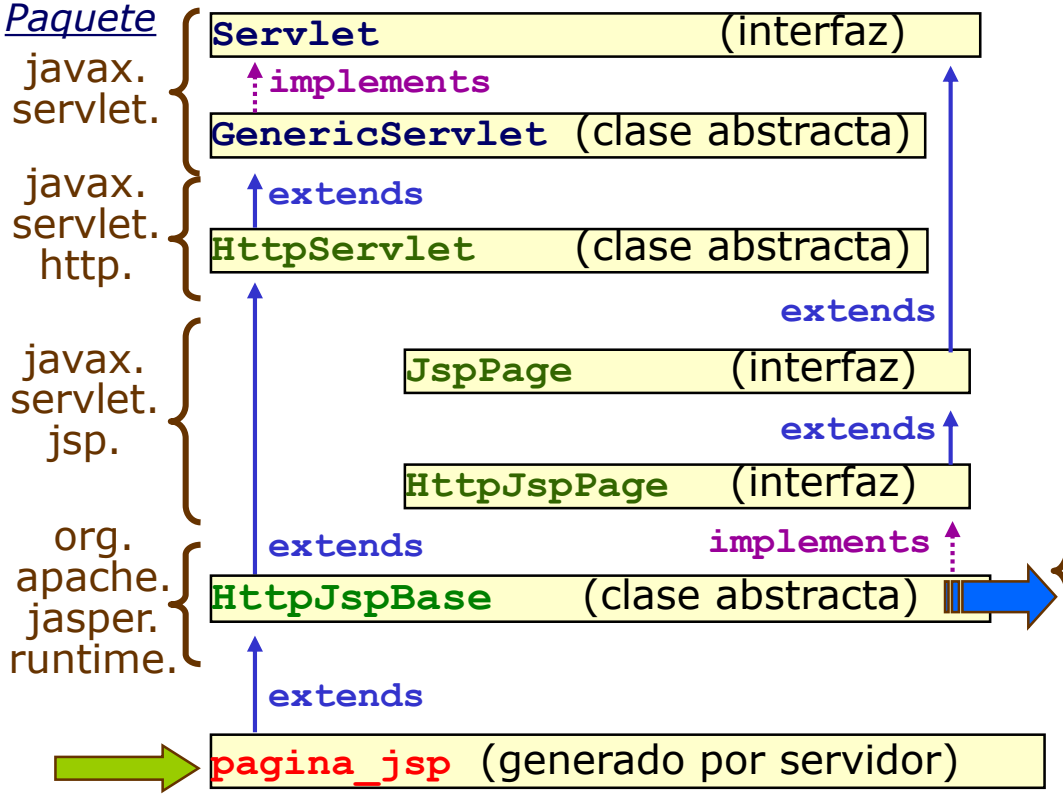
/carpeta web/pagina.jsp — *Conversión a servlet* → /DIR servidor/pagina_jsp.java

```
<html><body>
 <%-- Comentario JSP                --%>
 <!-- Comen HTML -->
 <p>Texto HTML</p>
 <%! String cadena=" ";               %>
 <%  int var=2;
     for( int i=1; i < 10; i++ ){
      cadena = cadena +i;           }%>
 <%= var                              %>
 <%@ page import="java.io.*"          %>
 <%! String cad=" ";
     public void jspInit() {...}
     public void jspDestroy(){...}
     public void funcion() {...}%>
</body></html>
```

```
...; import javax.io.*;
public final class pagina_jsp extends
org.apache.jasper.runtime.HttpJspBase
{ String cadena=" ";
  String cad=" ";
  public void funcion() {...}

  public void jspInit() {...}
  public void jspDestroy() {...}
  public void _jspService ()
  { ...
    out.write("<html><body>\n");
    out.write("<!--Comen HTML-->\n");
    out.write("<p>Texto HTML</p>\n");
    int var=2;
    for( int i=1; i < 10; i++ ) {
         cadena = cadena +i;     }
    out.print( var ); ← Variables con "print"
    out.write("\n"); ← Cambio línea en HTML
    out.write("</body></html>");
} }
```

JSP: Scriptlets, Func. (3); Clase Servlet interno

- El servlet que genera el servidor web hereda de la clase abstracta "**HttpJspBase**" (que a su vez hereda "**HttpServlet**"): define métodos para páginas JSP



* La redefinición de las funciones "`jspInit()`" y "`jspDestroy()`" en nuestra "pagina.jsp" es opcional

Principales Métodos disponibles

- `jspInit()` ← Inicialización*
- `_jspService()` ← Contenido de la página JSP
- `jspDestroy()` ← Antes del cierre*

JSP – Objetos implícitos

- 9 objetos implícitos que se pueden usar directamente. Cada uno de ellos tiene métodos útiles.
  - request: petición y objetos asociados (cookies y headers).
    - http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html
  - response: respuesta y objetos asociados (cookies y headers).
    - http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html
  - out: para generar contenido.
    - https://docs.oracle.com/javaee/7/api/javax/servlet/jsp/JspWriter.html
  - session: crear y borrar sesiones, almacenar objetos.
    - http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html
  - application: almacenamiento y acceso a objetos de la aplicación.
    - http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html
  - config: parámetros de configuración del servlet.
    - http://docs.oracle.com/javaee/7/api/javax/servlet/ServletConfig.html
  - pageContext: acceso a todos los demás y métodos útiles.
    - http://docs.oracle.com/javaee/7/api/javax/servlet/jsp/PageContext.html
  - page = this
  - exception = Excepción lanzada por una página anterior.

JSP – Atributos y ámbitos (scopes)

- Atributos: objetos asociados a un nombre, en tabla asociativa.
  - La información que necesita una aplicación web puede almacenarse en distintos ámbitos (scopes).
  - Se pueden crear, borrar, recuperar, listar y buscar.

Ámbito (id)	Objetos con acceso	Accesible desde
APPLICATION_SCOPE	`application, pageContext`	Todo el código de la aplicación.
SESSION_SCOPE	`session, pageContext`	Todo el código que procese las peticiones de una misma sesión.
REQUEST_SCOPE	`request, pageContext`	Todo el código que procese una petición. Por ejemplo podrían estar involucrados varios servlets por inclusión o redirección y filtros.
PAGE_SCOPE	`pageContext`	El servlet y petición actual.

- Ejemplos:
  - `pageContext.setAttribute( “nombreAttributo”, 4, SESSION_SCOPE);`
  - `pageContext.getAttribute( “nombreAttributo”, SESSION_SCOPE);`

JSP-Javabeans: Introducción

- Problemas Etiquetas de Scripts JSP (similares PHP, ASP, …):
  - Código confuso: muchos lenguajes mezclados.
  - Depuración compleja del código Java embebido.
  - Presentación (html) y Lógica (java) mezcladas.
- Javabeans: invocan (***no insertan***) clases Java desde HTML.
  - Separan Presentación/Lógica; Java en ficheros ".class" (depura.)

http://IP/alias/index.jsp

HTTP Request

>1024

Internet

80

Navegador Web

HTTP Response

Servidor Web

index.jsp

```
<html><body>
 <jsp:useBean id="instancia" scope="session"
      class="paquete.claseJB"/>
<p><jsp:getProperty name="instancia"
                property="var" /></p>
</body></html>
```

Código HTML que genera

response

```
<html><body>
<p>Hola<p>
</body></html>
```

claseJB.java

```
public class claseJB {
  private String var =
                    "Hola";
  public String getVar()
    { return (var); }
}
```

JSP-Javabeans: Especificación

- Bean o Javabean: ***clase*** que cumple especificación Javabeans

Requisitos especificación Javabeans (*entre otros*)

Constructor { Sólo uno y Sin argumentos*

Variables miembro {
```
private tipo variable;
public  tipo getVariable() {…}
public  void setVariable(tipo par) {…}
```

* **Java:**
- **Constructor por defecto**: sin argumentos.
- Si se definen constructores: el constructor por defecto no se genera.

- Objetivo: interfaz de acceso, mediante funciones, conocida
- Ejemplo:

/paquete/clasesJB.java

```
package  paquete;
public class claseJB {
  private String var = "Hola";
  public  String getVar ()            { return (var); }
  public  void   setVar (String arg)  { var = arg;    }
}
```

Para usarla desde JSP debe estar dentro de un paquete

JSP: Javabeans, Invocación desde HTML

- Etiquetas (acciones) JSP para "Javabeans": `<jsp:useBean` `<jsp:setProperty` `<jsp:getProperty`
- Ejemplo:

index.jsp

```
<html><body>
<%paquete.claseJB x =
    (paquete.claseJB) pageContext.getAttribute("x", SESSION_SCOPE);
  if (x == null) {
    x = new paquete.claseJB();
    pageContext.setAttribute("x", x, SESSION_SCOPE); } %>
<jsp:useBean id="x"  class="paquete.claseJB" scope="sesion"/>
<jsp:setProperty name="x" property="var" value="Adios" />
<p>
<% x.setVar("Adios"); %>
<jsp:getProperty name="x" property="var" />
</p>
<%= x.getVar(); %>
</body></html>
```

Código HTML que genera

```
<html><body>
<p>
Adios
</p>
</body></html>
```

EL: Objetivo y Sintaxis

- Objetivo: Sintaxis simplificada para expresiones Java en HTML
  - **Sintaxis** EL (Expression Language):
    - Sintaxis de acceso a propiedades similar a la de ECMAScript (`${obj.prop} == ${obj['prop']`)

```
${ expresion_EL }
```
NO Java

  - Funcionalidad ***similar*** a expresiones JSP:

```
<%= expresion_java %>
```

  - ***Pero*** **con <u>otros</u> objetos implícitos**:
    - initParam, pageContext (mismo),
    - pageScope, requestScope,
    - sessionScope, applicationScope,
    - param, paramValues,
    - header, headerValues,
    - cookie

index.jsp

EL

```
<p>
${cookie['parCookie'].value}
${header["User-Agent"]}
${param.num}
</p>…
```

=

Sin EL

```
…<p>
<%
 Cookie cookie = null;
 Cookie[] cookies = null;
 cookies = request.getCookies();
 if( cookies != null ){
  for (int i = 0; i < cookies.length; i++){
   cookie = cookies[i];
   if (cookie.getName().equals("parCookie"))
    out.print(cookie.getValue());
 }}  %>
<%= request.getHeader("User-Agent") %>
<%= request.getParameter("num") %>
</p>…
```

Referencias

- CGI:
  - *Estándar CGI 1.1:* RFC 3875 (http://www.ietf.org/rfc/rfc3875)
  - *Uso práctico:*
    - *Recursos (Ejemplos):* http://cgi.resourceindex.com/
      http://oreilly.com/openbook/cgi/

- Java:
  - *Especific. (JSRs):* http://jcp.org/
    http://www.oracle.com/technetwork/java/javaee/tech/
    - *API (JavaEE):* http://docs.oracle.com/javaee/
    - *Document.:* http://www.oracle.com/technetwork/java/javaee/documentation/
  - *Uso práctico:*
    - *Tutorial:* https://www.tutorialspoint.com/jsp/index.htm
    - *Recursos (Ejemplos):*
      http://www.java2s.com/Code/Java/Servlets/CatalogServlets.htm
      http://www.java2s.com/Code/Java/JSP/CatalogJSP.htm
      http://books.coreservlets.com/