

Aplicaciones Web

Diego C. Martínez

DIT
Facultad de Ingeniería - UNPSJB

Yo :)

Diego C. Martínez

dcm@cs.uns.edu.ar

<http://cs.uns.edu.ar/~dcm>

Departamento de Ciencias e Ingeniería de la Computación.
Universidad Nacional del Sur.
(0291)-4595101 interno 2604

Tópicos de interés

- Inteligencia Artificial – representación de conocimiento y razonamiento.
Formalismos de argumentación rebatible
- Inteligencia Artificial aplicada al Entretenimiento Digital Interactivo.
Estrategia en tiempo real, juegos con escenarios de inmersión
- Programación Web.
Java, PHP, JavaScript, Frameworks...

Ingeniería de Aplicaciones Web

Objetivo del curso



Examinar tecnologías, técnicas y buenas prácticas en el desarrollo de aplicaciones web, tanto desde el lado del servidor como desde el lado del cliente.

Régimen de cursado y aprobación

Habrá varios proyectos de programación.

Horarios

Teoría, una vez por mes, viernes y sábados a la mañana.

Práctica, en horario a convenir.

Agenda general del curso



Introducción

*Escenario cliente-servidor
World Wide Web, Servidores, etc*

Protocolos y lenguajes básicos

*Protocolo HTTP, URL,
estándares (X)HTML, DOM, etc*

Programación del lado cliente

*Navegadores Web, lenguaje
Javascript, presentación visual,
páginas dinámicas, etc*

Programación del lado servidor

*Historia, Server Side Includes, Scripts CGI,
PHP, operaciones tradicionales: procesamiento de
formularios, sesiones, uploads, etc*

Interacciones

*Interacción cliente-servidor.
AJAX, AJAX-push*

Frameworks para la Web

*Conceptos, MVC, Frameworks: CakePHP, Joomla!,
Drupal, Sinfonía, CodeIgniter, Jakarta Struts, etc*

Servicios Web

*XML-RPC, REST, SOAP, WSDL, UDDI
Arquitecturas Microservicios, Serverless*

Seguridad en la Web

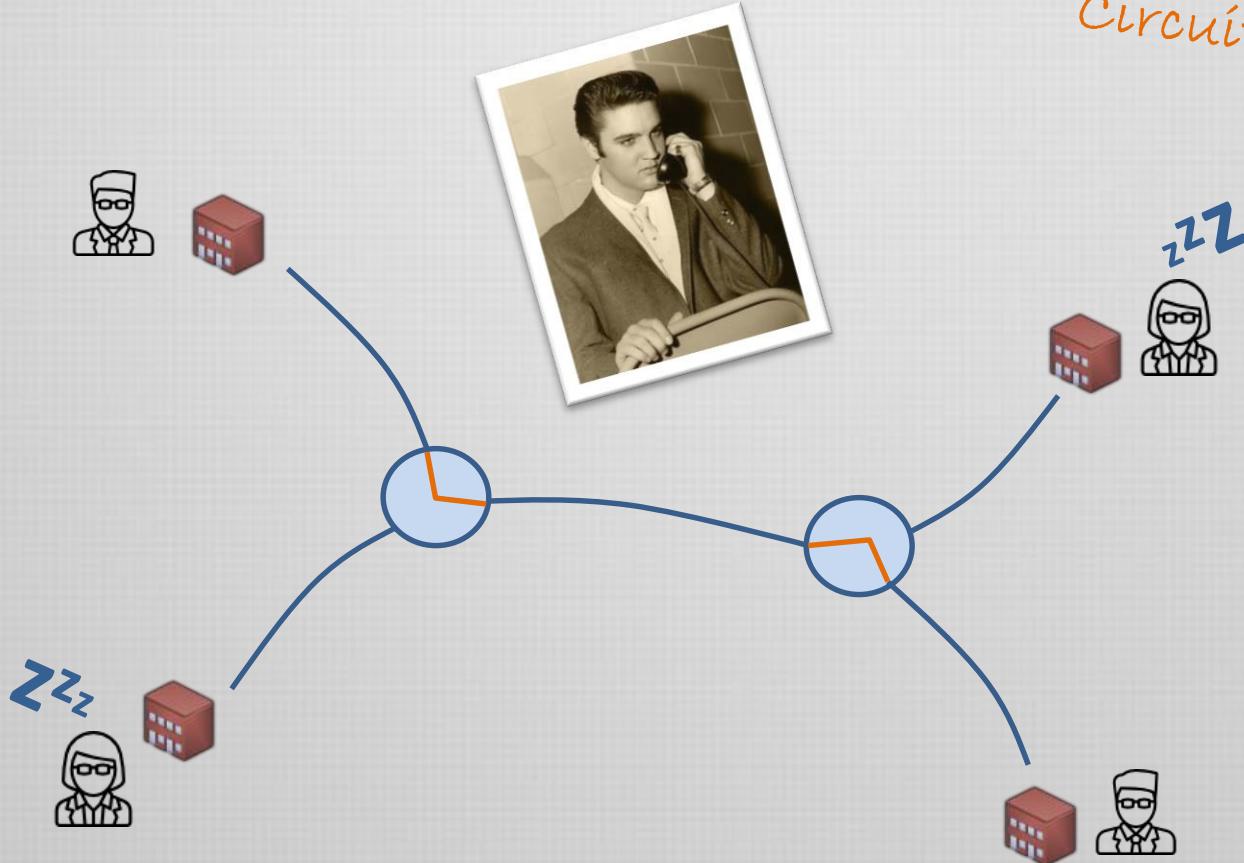
Amenazas comunes, debilidades, defensas, etc

Aplicaciones

*Web semántica, búsqueda en la web, web multimedial,
e-commerce, e-government, etc*

Historia de Internet

50's
Circuit switching



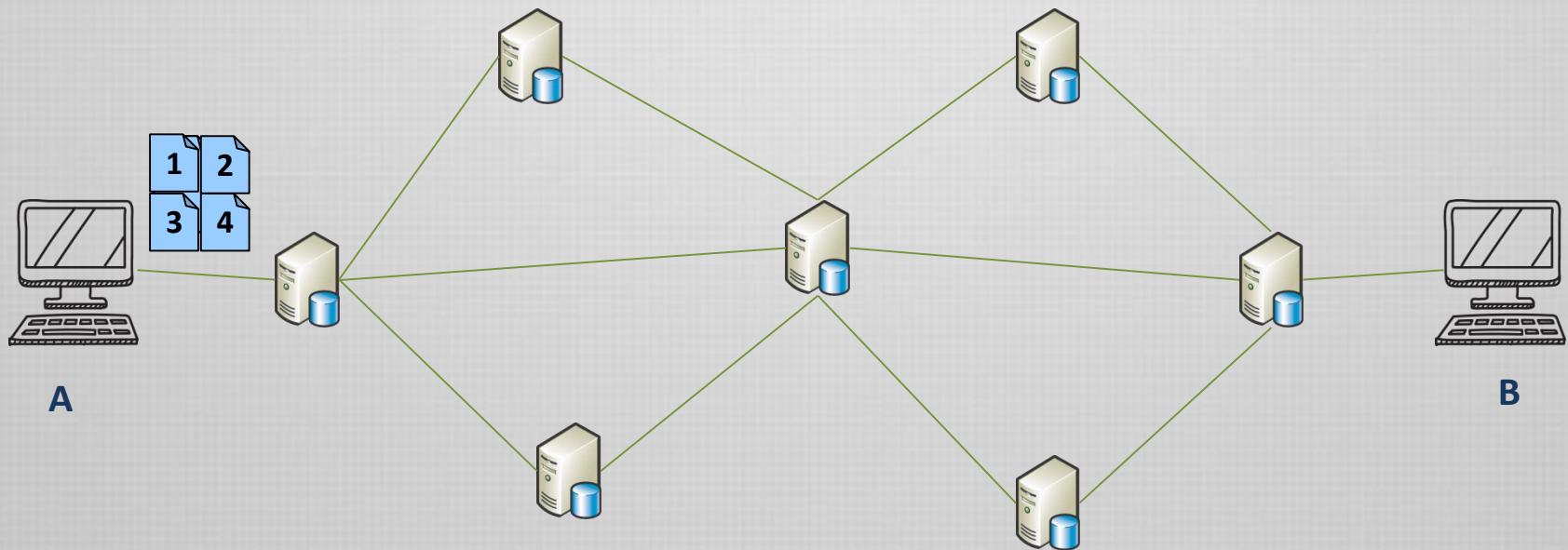
Historia de Internet

60's
Packet Switching



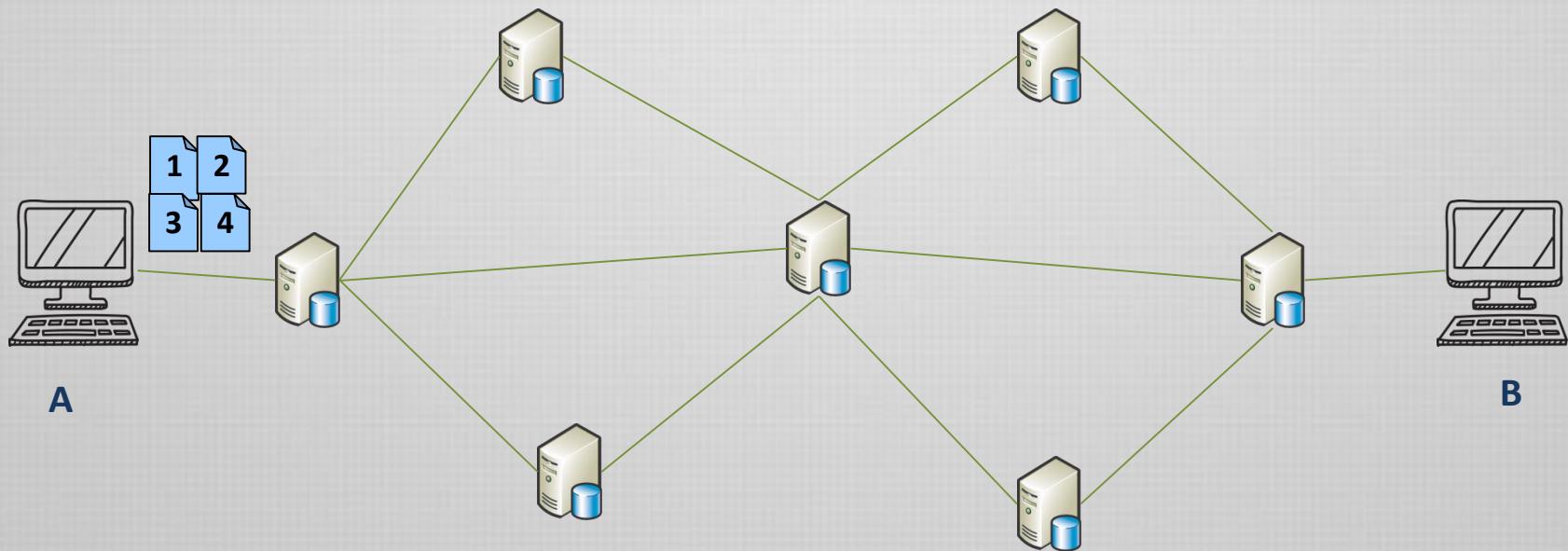
Historia de Internet

60's
Packet Switching



Historia de Internet

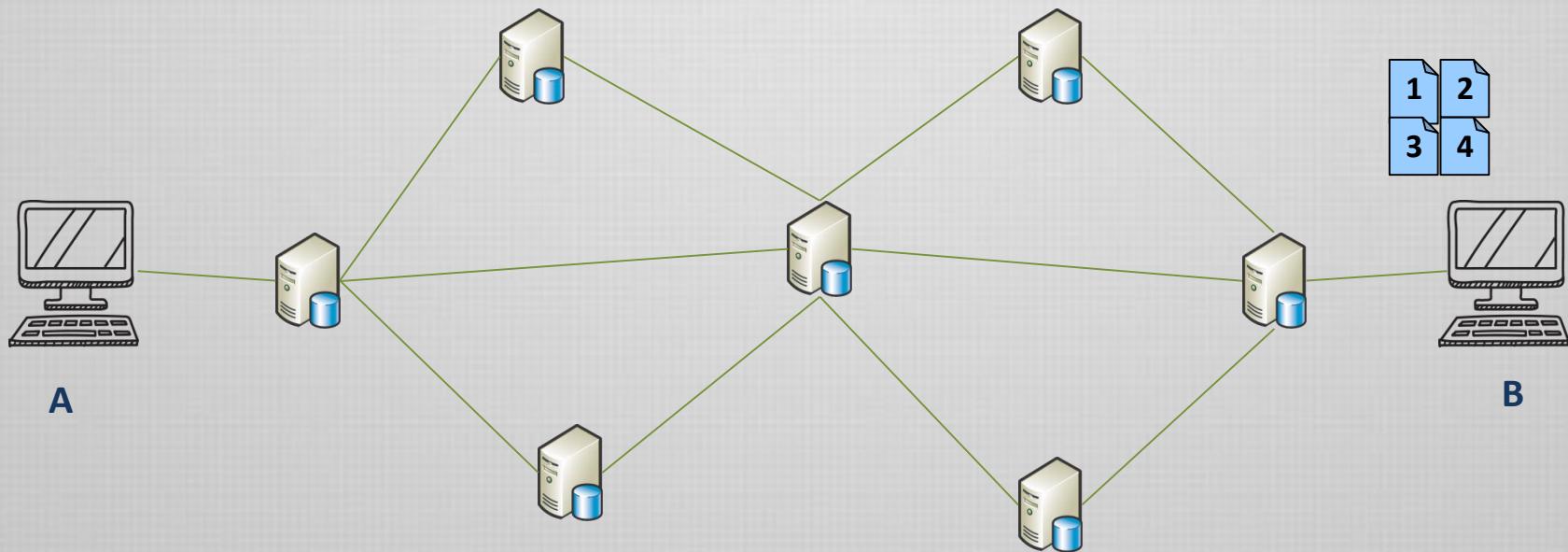
60's
Packet Switching



Historia de Internet

60's

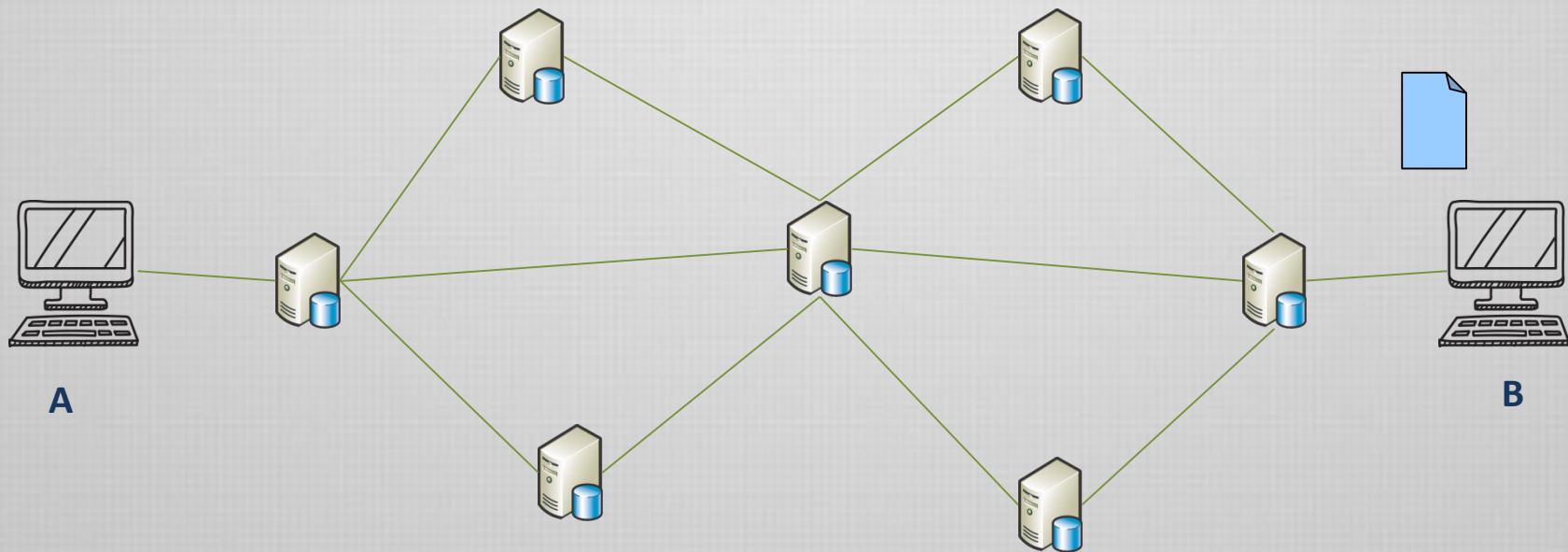
Packet Switching



Historia de Internet

60's

Packet Switching



Historia de Internet

Los avances en este aspecto comenzaron en
DARPA
(*Defense Advanced Research Project Agency*)
a fines del 50 y durante los 60,
en plena guerra fría.



Trabajaban allí Ivan Sutherland (computer graphics), Robert Taylor (networking), y Lawrence G. Roberts (Arpanet) del MIT, entre otros.

Paul Baran, Leonard Kleinrock publican los primeros trabajos sobre *packet switching* a comienzos de los 60.

La gente de DARPA comenzó a trabajar sobre esta idea, conformando lo que luego sería **ARPANET**, una de las redes centrales de Internet.

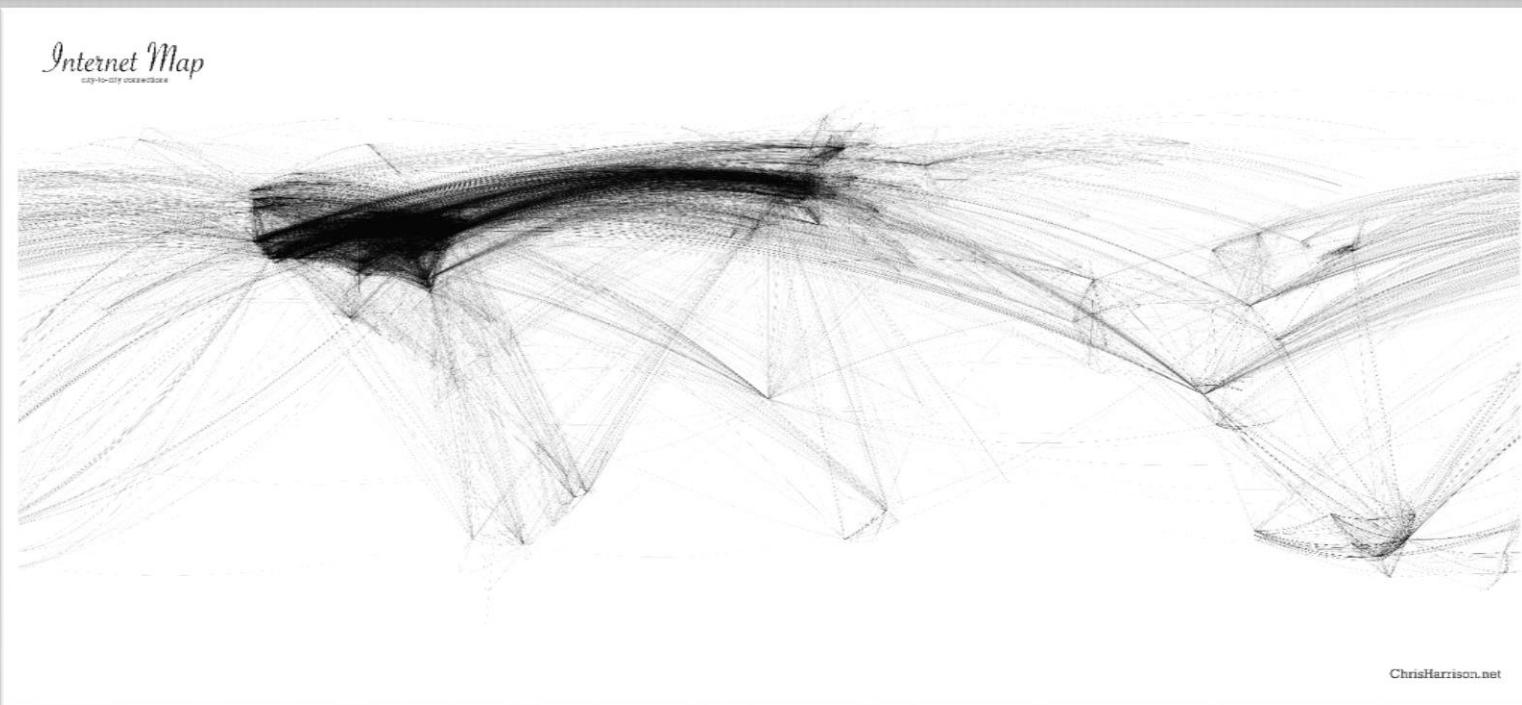
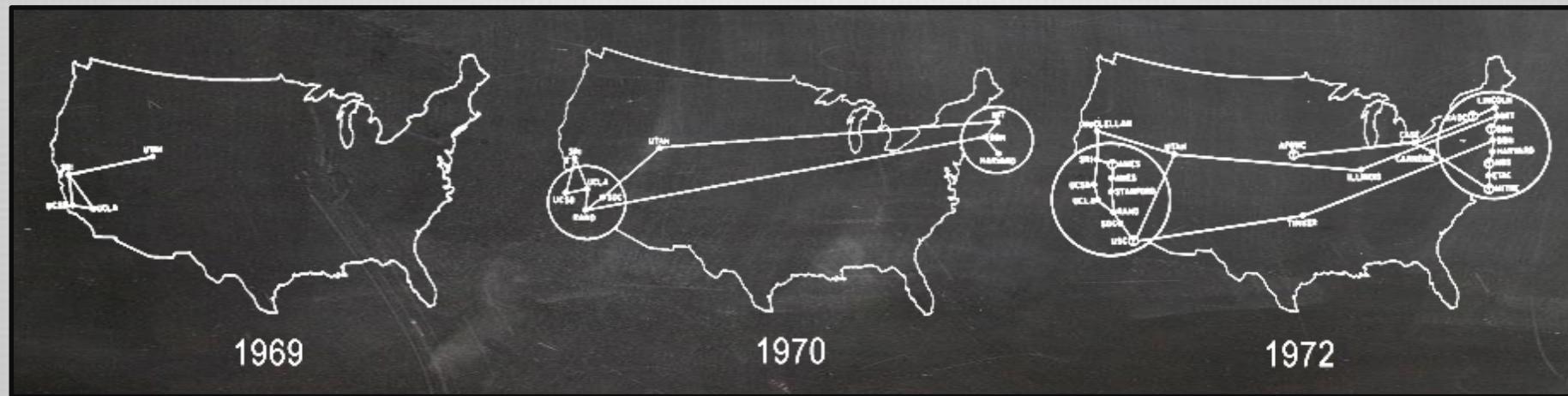


Leonard Kleinrock
(packet switching)



Paul Baran
(Packet switching)

Evolución de ARPANET

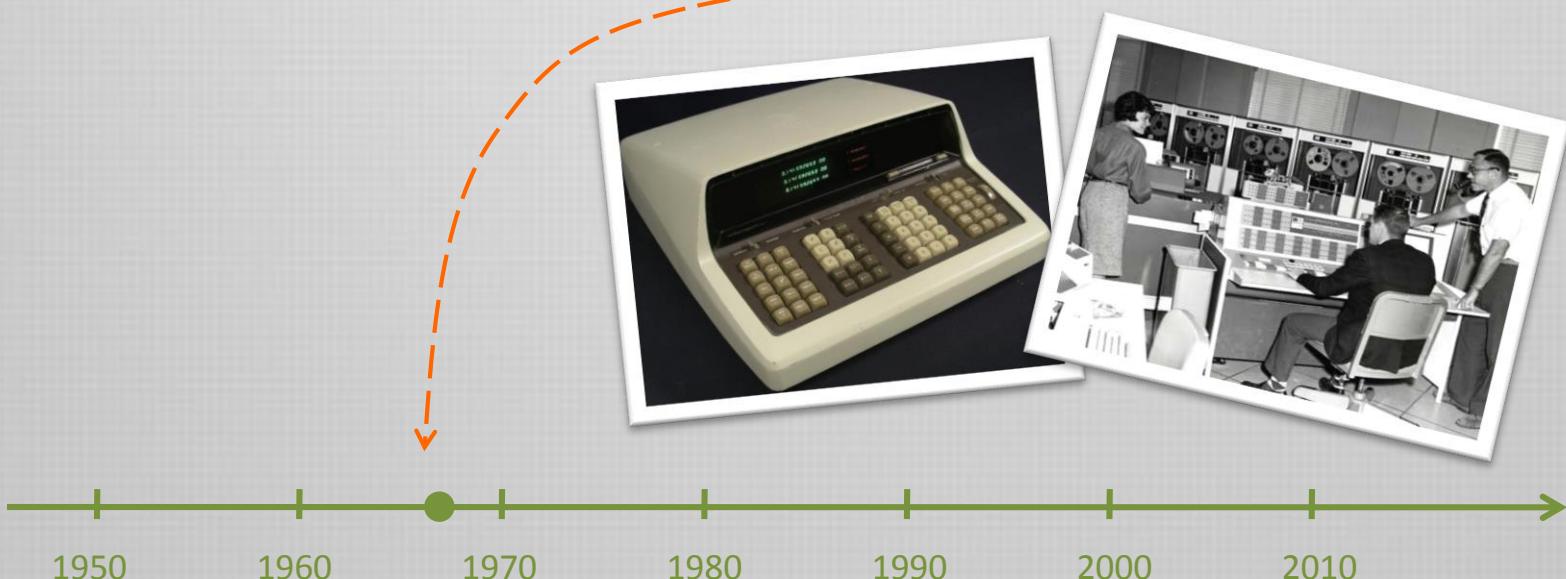


Comunicaciones

In a few years, men will be able to communicate more effectively through a machine than face to face.

The Computer as a Communication Device
JCR Licklider , Robert Taylor

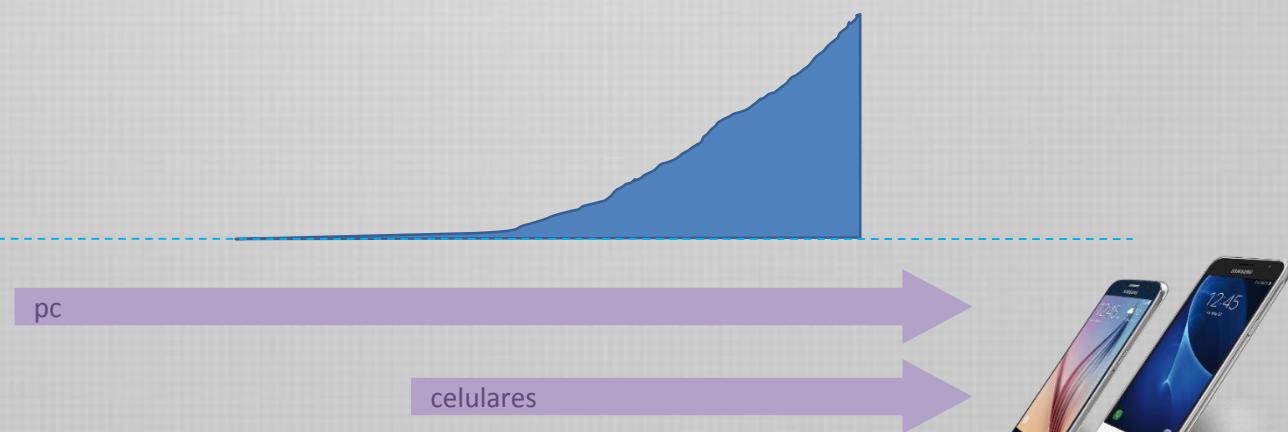
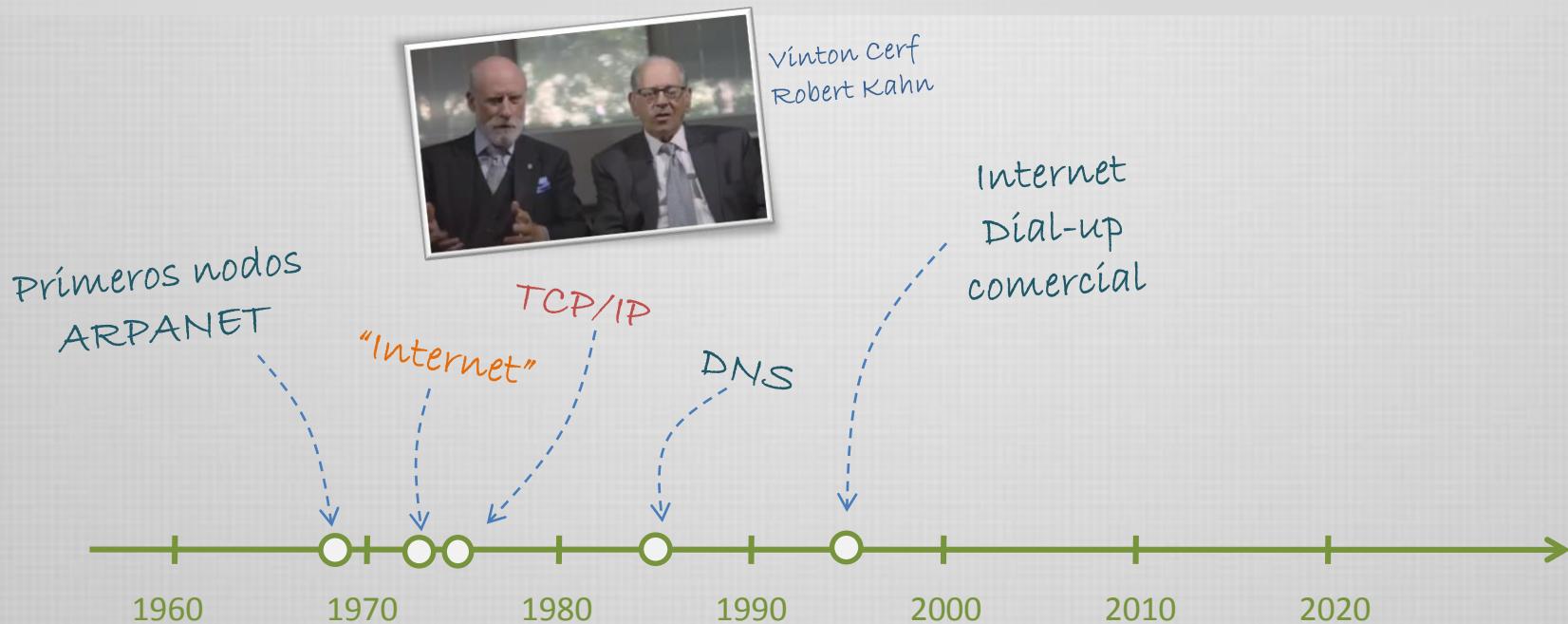
1968



Comunicaciones



Comunicaciones



Internet



Capa de Aplicación

Aplicaciones
clientes de mail, navegadores, ftp

Capa de Transporte

Comunicación de aplicación a aplicación
p2p, regulación de flujo de datos, confiabilidad

Capa de Internet

Comunicación máquina a máquina
paquetes IP, algoritmos de ruteo, ICMP

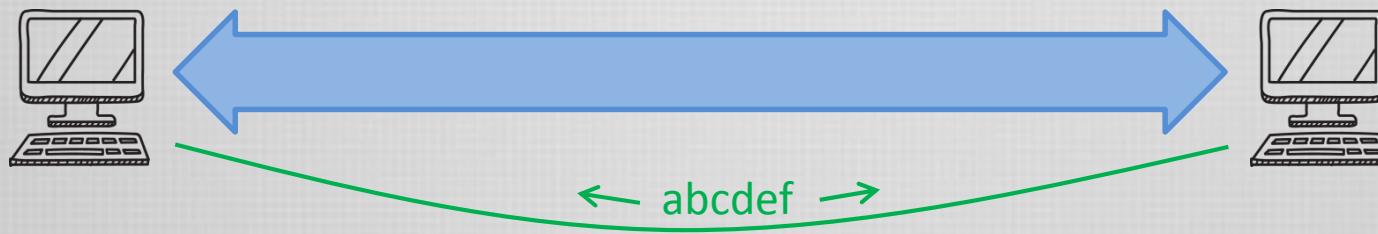
Capa de Interfaz de Red

Aspectos físicos del transporte de
información

Hardware

Internet

Internet define una forma de **conexión** de redes heterogéneas.



Los **usos** de esta conexión son variados y cada uno determina a su vez, diferentes **protocolos** de comunicación.

Entre ellos:
envío y recepción de mensajes (SMTP)
envío y recepción de archivos (FTP)

Documentos globales

Una de las motivaciones iniciales de este tipo de redes de comunicación fue el de la *difusión y exploración de documentos de información general*.

Al igual que con los otros protocolos, debía definirse:



cómo estructurar la información, para facilitar la exploración



cómo intercambiar esa información



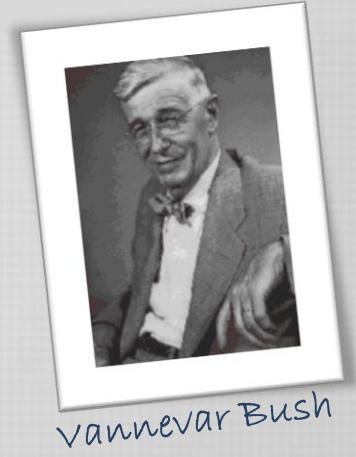
cómo visualizar esa información

Esta es la motivación inicial de la [World Wide Web](#) y el protocolo [HTTP](#).

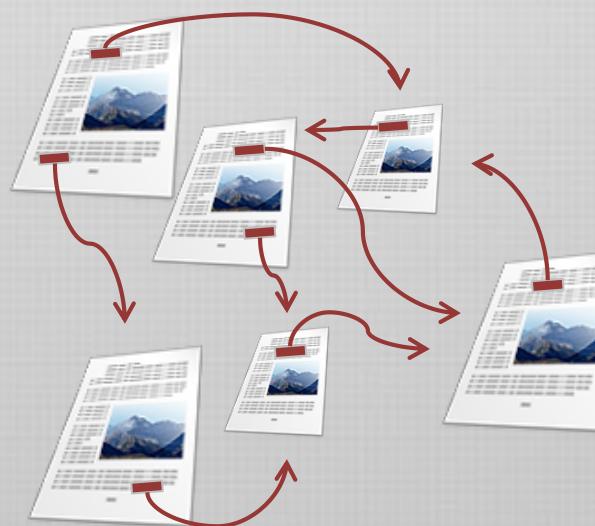
Hipertexto

El Dr. Vannevar Bush propone en 1945 la idea de una máquina denominada Memex, en el artículo “As we may think”.

Esta máquina tendría la capacidad de almacenar información gráfica y de texto, de una forma tal que cualquier pieza de información puede ser vinculada con otra(s) pieza(s).



En 1965 Ted Nelson introdujo el término *hipertexto* e *hipermedia* en un paper para la ACM.



Hipermedia
Medios de información no-lineal

Web

1989



Tim Berners-Lee



World Wide Web

*los documentos
están escritos
en hipertexto
(HTML)*



*El protocolo de
comunicación es HTTP*



Navegadores

En 1994 Berners-Lee funda el *World Wide Web Consortium (W3C)* en el MIT, con apoyo de DARPA.

La idea central era asegurar la compatibilidad por medio de la definición de estándares, denominados W3C Recommendations.

Web

Tim Berners-Lee - London 2012



Web

A screenshot of a vintage Netscape browser window. The title bar reads "Netscape". The menu bar includes "File", "Edit", "View", "Go", "Bookmarks", "Options", "Directory", "Window", and "Help". Below the menu is a toolbar with icons for Back, Home, Reload, Open, Print, and Find. The location bar shows the URL "http://www.boeing.com". A navigation bar below it contains links for "What's New!", "What's Cool", "Handbook", "Net Search", and "Net Directory". To the right of the navigation bar is a small "N" logo. The main content area is blank, indicating the page is still loading. At the bottom of the browser window, there is a status bar with icons for Stop, Refresh, Document, Done, and Help.

Location: http://www.boeing.com

What's New! What's Cool Handbook Net Search Net Directory

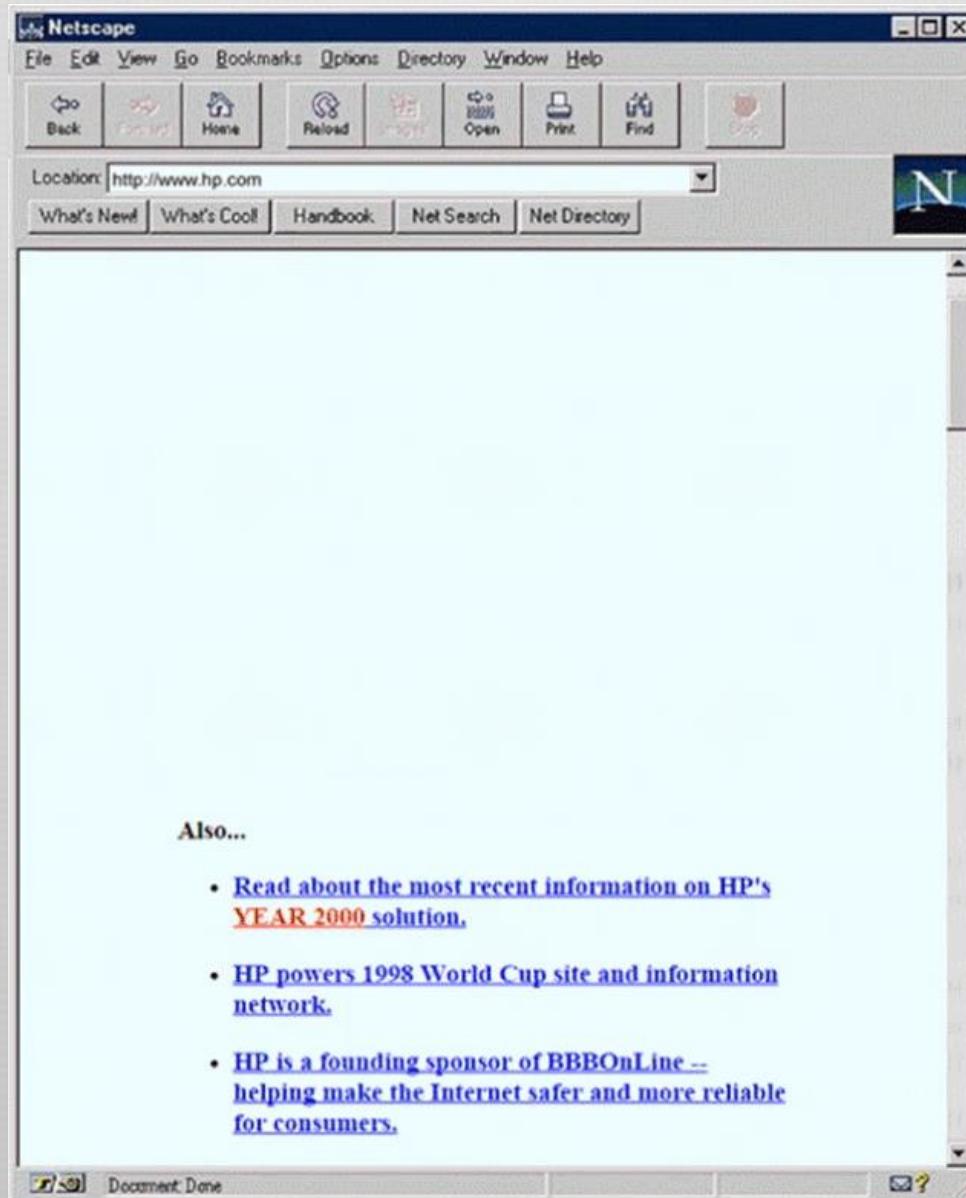
New | Inside Boeing | Images | Site Map |

The Boeing Company

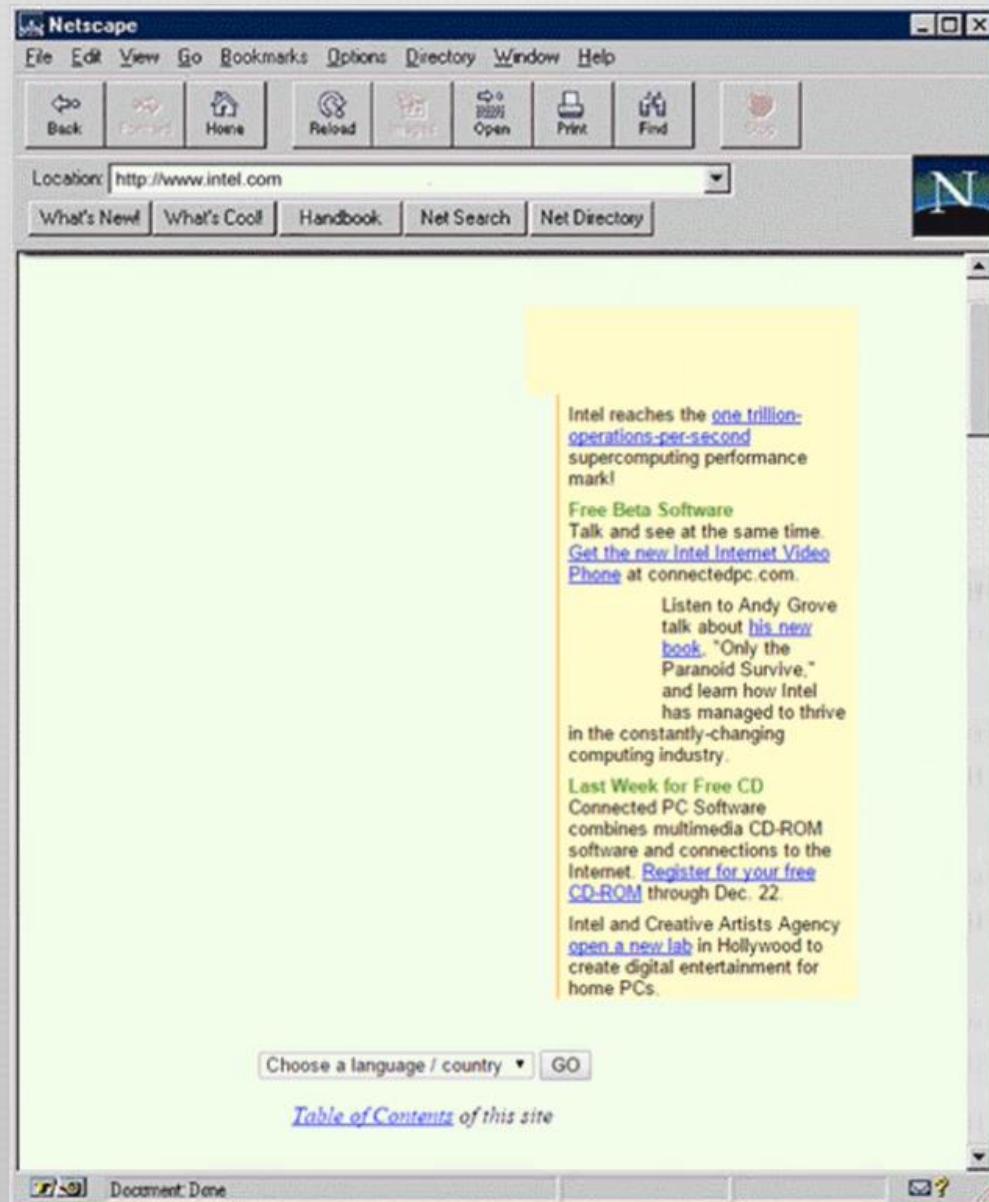
The Boeing Company, with headquarters in Seattle, Washington, U.S., is the world's leading manufacturer of [commercial airplanes](#) and one of the nation's largest exporters. The company is a major U.S. Government contractor, with capabilities in [space systems](#), [helicopters](#), [military airplanes](#), [missile systems](#), [information and electronic systems](#) and [software products](#).

Boeing Employment Opportunities

Web



Web



Web



HTTP

HTTP es el protocolo de red para la Web.

→ “*Hypertext Transfer Protocol*” →

protocolo de comunicación
define

- ✉ tipo y estructura de los mensajes
- 💬 las reglas del diálogo

protocolo de aplicación para la entrega de **documentos variados**

“recursos”
ubicables por medio del **URL**.



HTTP es un protocolo sin estado, o *stateless protocol*:
no mantiene información sobre la conexión entre transacciones

URLs - sintaxis

El nombre estandarizado de un recurso en Internet, que lo identifica únicamente se denomina **URL - Uniform Resource Locator**.

*Es un subconjunto de una clase más general de identificadores
(URI - Uniform Resource Identifier)*

URI identifica
URL localiza

“Epic Games” URI

“Epic Games, Inc. 620 Crossroads Blvd Cary, NC 27518”

URL
URI

Una URL describe el recurso por su *locación* (dónde está) y el *protocolo* que se entiende para obtenerlo (HTTP, HTTPS, FTP, SMTP, etc).

URLs - sintaxis

<schema>://<user>:<password>@<host>:<port>/<path>?<query>#<frag>

<schema> Protocolo para acceder al recurso

<user>:<password> Usuario y password requerido para acceder al recurso

<host>:<port> Nombre del host o dirección IP del servidor que tiene el recurso + port

<path> Nombre local del recurso

<query> Parámetros de entrada para el recurso

<frag> Referencia a una porción del recurso (de uso para el cliente)

Ejemplos

http://cs.uns.edu.ar

http://www.uns.edu.ar:80

http://www.fis-ski.com/uk/disciplines/alpineskiing

https://www.bancogalicia.com.ar

ftp://ftp.prep.ai.mit.edu/pub-gnu

ftp://anonymous:my_passwd@ftp.prep.ai.mit.edu/pub-gnu

https://www.youtube.com/watch?v=ej9vLgraWXk

→ port por defecto

URLs - sintaxis

Una URL puede ser
absoluta o relativa

las URL **absolutas** contienen
toda la información necesaria
para obtener el recurso

*Las anteriores son
todas absolutas*

las URL **relativas** son interpretadas
en relación con otra URL

`Go!`

La URL relativa se completa con

*una URL base
declarada en el recurso*

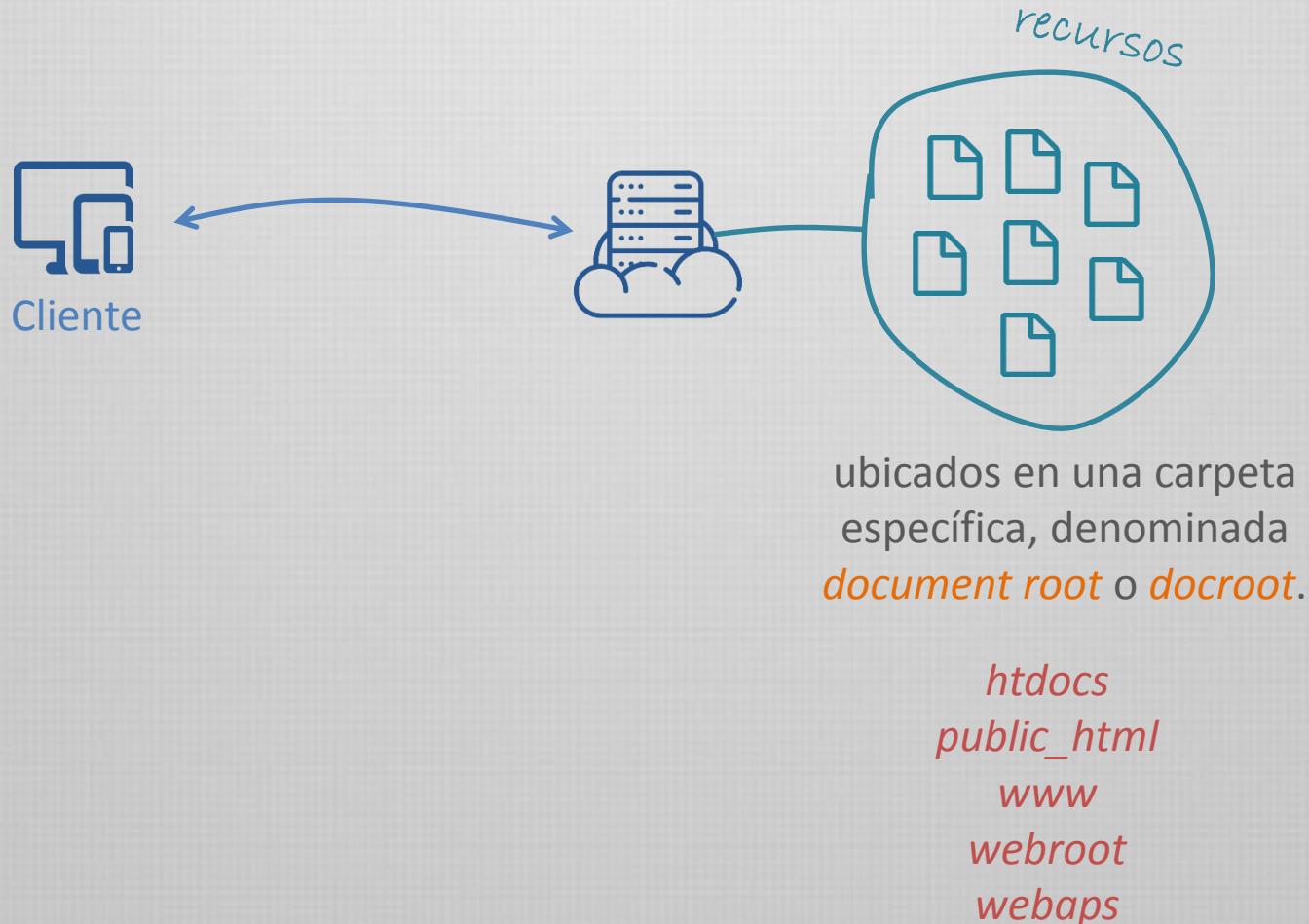
`<base href="http://..." >`

*la URL base del
recurso que contiene
la URL relativa*

(VER RFC 2396 de la W3C)

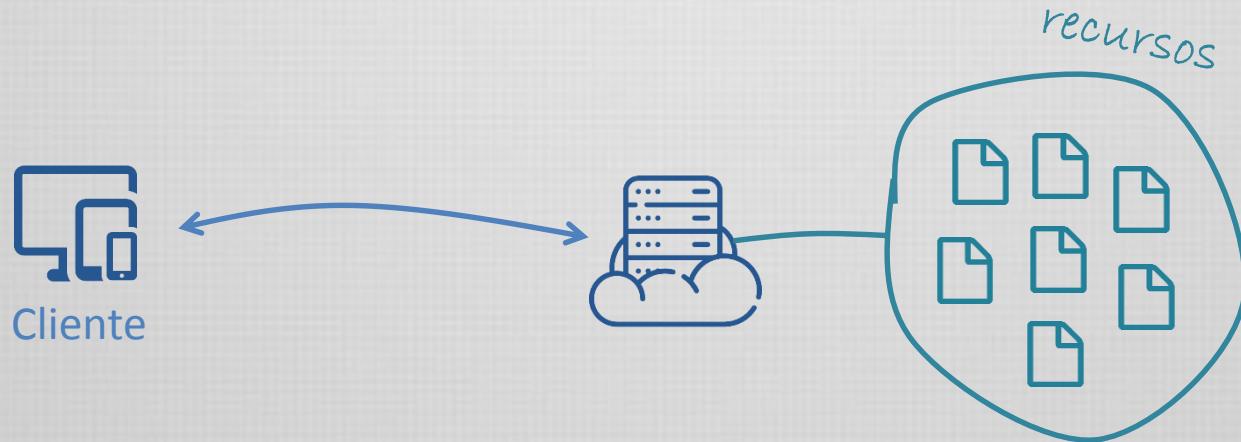
Servidores web

El servidor web actúa como una **interfaz** entre un conjunto de **recursos** y los **clientes**.



Servidores web

El servidor web actúa como una **interfaz** entre un conjunto de **recursos** y los **clientes**.



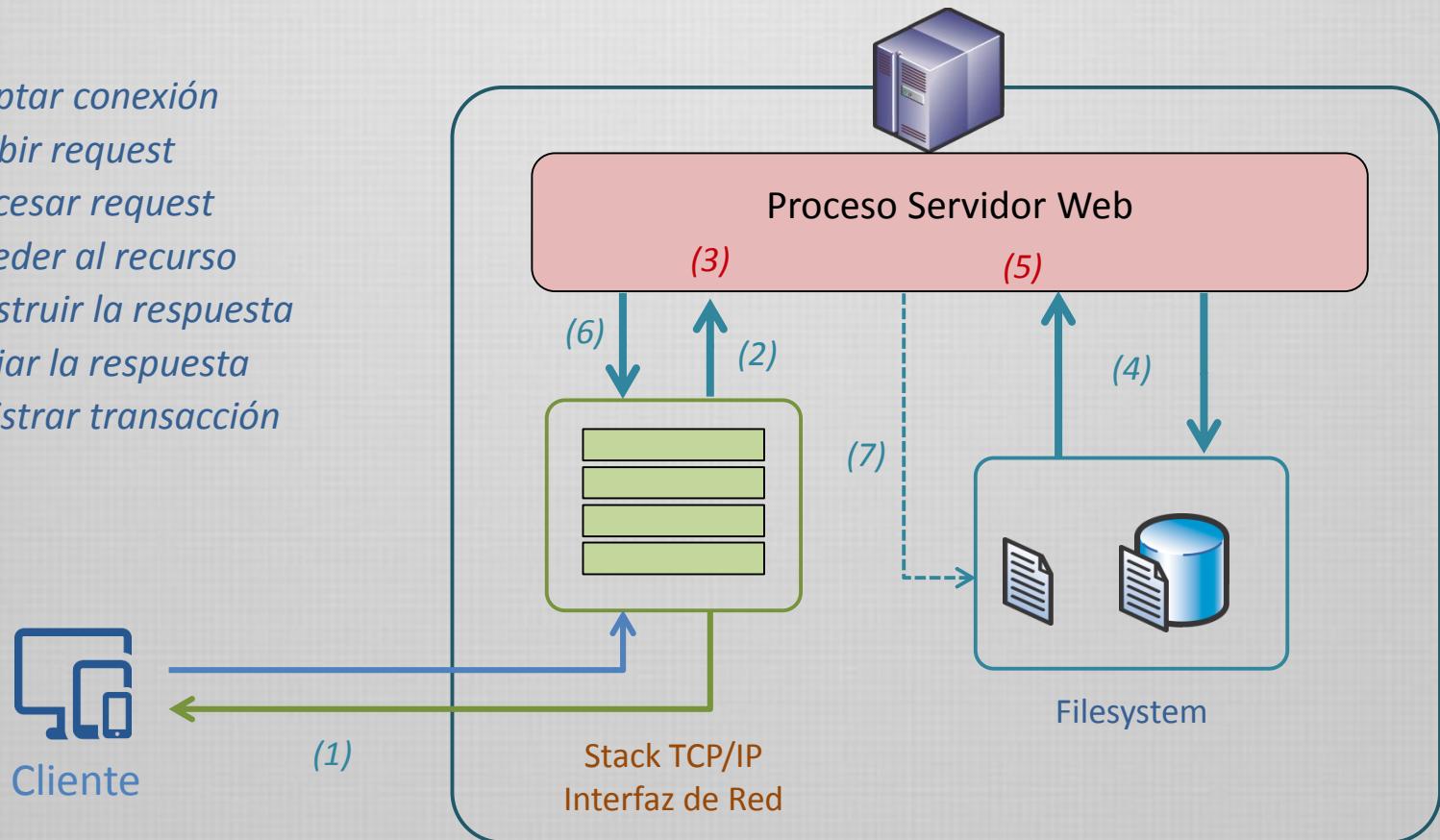
*Para solicitar un recurso del servidor,
es necesario mencionar
el **host** y el **camino hacia el recurso***

*No necesariamente el camino físico:
<http://unhost.com/~john/index.html>
puede ser
</home/users/john/web/index.html>
en el servidor unhost.com*

Servidores web

El esquema de trabajo de un servidor web es el siguiente:

1. aceptar conexión
2. recibir request
3. procesar request
4. acceder al recurso
5. construir la respuesta
6. enviar la respuesta
7. registrar transacción



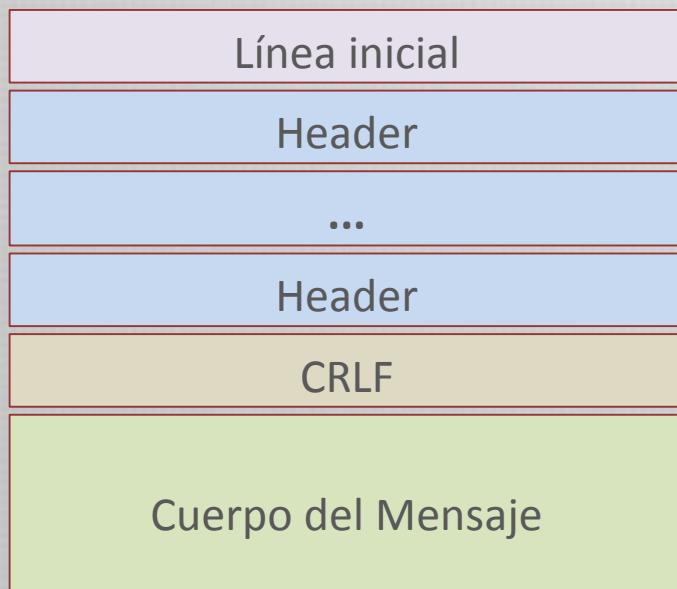
Mensajes HTTP



Como **protocolo de comunicación**, HTTP define el **tipo y estructura** de los mensajes que se envían y las reglas del diálogo entre los dos participantes.

- mensajes hacia el servidor → *inbound messages/ requests*
- mensajes desde el servidor → *outbound messages / responses*

La **estructura** de los mensajes es la misma para todos



Diferente para los requerimientos y respuestas.

Cero o más encabezamientos (header lines)

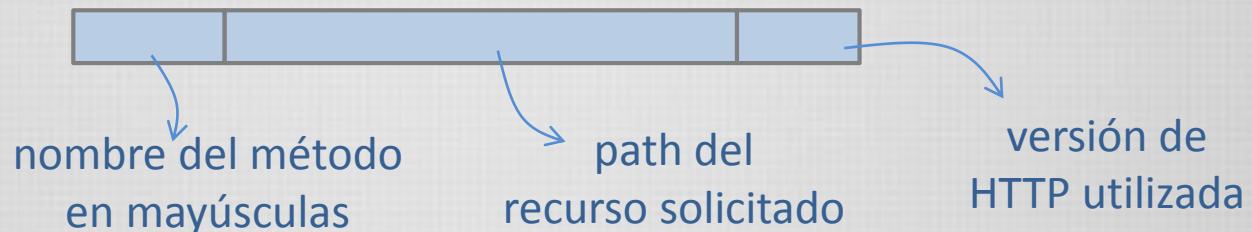
Línea en blanco

Contenido del mensaje: archivos, consultas a bases de datos, datos binarios, etc.



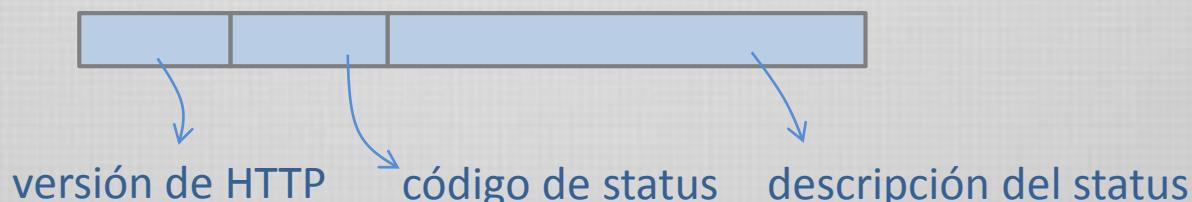
Mensajes HTTP

Una línea inicial de *request* tiene tres partes



**GET /path/to/file/index.html HTTP/1.0
POST subscribir.php HTTP/1.0**

Una línea inicial de *response* tiene tres partes



**HTTP/1.0 200 OK
HTTP/1.0 404 Not Found**

El primer dígito del *código de status* identifica la categoría:

- **1xx** indica un mensaje de información únicamente
- **2xx** indica éxito en general
- **3xx** redirecciona el cliente a otro URL
- **4xx** indica un error en la parte del cliente
- **5xx** indicates un error en la parte del servidor



► **200 OK**

El pedido tuvo éxito, y el recurso es retornado en el cuerpo del mensaje.

► **404 Not Found**

El recurso solicitado no existe

► **301 Moved Permanently**

► **302 Moved Temporarily**

► **303 See Other (HTTP 1.1 only)**

El recurso se movió a otro URL y debe ser pedido automáticamente por el cliente.

► **500 Server Error**

Error inesperado. Típicamente un error en algún script alojado en el servidor que impide que se ejecute correctamente.

El detalle completo de los códigos puede encontrarse en las *especificaciones de la W3C*:

[HTTP 1.0 : RFC 1945](#)

[HTTP 1.1 : RFC 2616](#)

El protocolo HTTP – header lines

Las **líneas de encabezamiento** proveen información sobre el pedido (request) o la respuesta (response) o el objeto que se está enviando en el cuerpo del mensaje.

Header-Name : value

HTTP 1.0 define 46 headers, pero ninguno es requerido.

HTTP 1.1 define 46 headers y requiere Host en los pedidos realizados.

Algunos headers del *cliente*:

- **From:** es la dirección de mail de quien realiza un pedido.
- **User-agent:** es el programa que realiza el pedido, de la forma “*Nombre/x.xx*”

Ejemplo: **User-agent: Mozilla/3.0Gold**

Algunos headers importantes del *servidor*:

- **Server:** identifica el software del servidor, en el mismo formato “*Nombre/x.xx*”

Ejemplo: **Server: Apache/1.2b3-dev**

- **Last-modified:** es la fecha de modificación del recurso otorgado

Ejemplo: **Last-Modified: Fri, 31 Dec 1999 23:59:59 GMT**

Métodos HTTP

GET → *Obtiene un documento del servidor.
El mensaje no tiene cuerpo.*

POST → *Envía datos al servidor para el procesamiento.
Los datos se especifican en el cuerpo del mensaje.*



HEAD → *Solicita solo los encabezados de un documento.
El mensaje no tiene cuerpo.*

PUT → *Almacena el cuerpo del request en el servidor, bajo el nombre indicado como URL.*



TRACE → *Hace un rastreo del mensaje al servidor. El mensaje no tiene cuerpo*

OPTIONS → *Determina qué métodos pueden operar en un servidor.
El mensaje no tiene cuerpo*

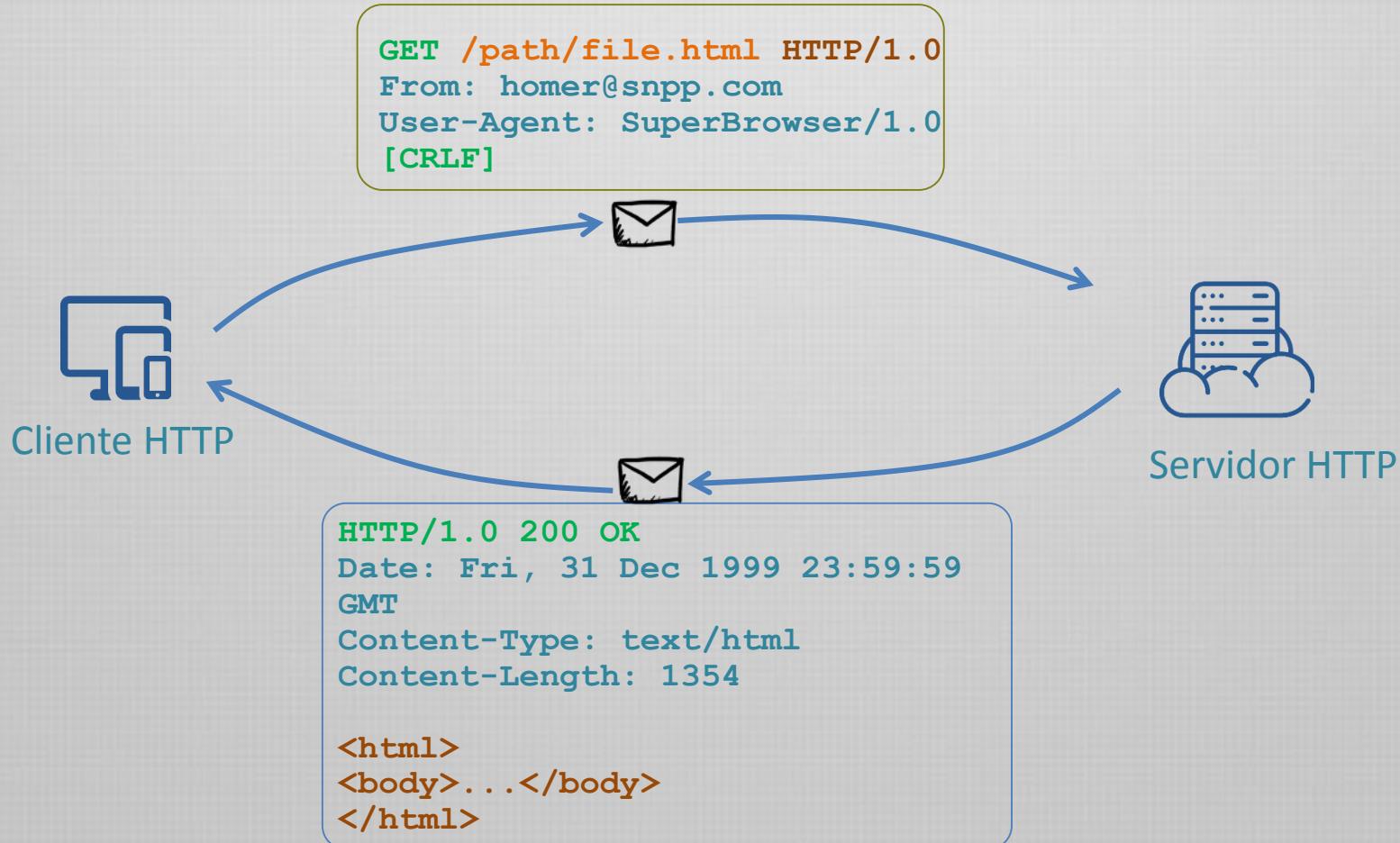
DELETE → *Remueve un documento del servidor. El mensaje no tiene cuerpo*

Requests - responses

Si el mensaje incluye un cuerpo, entonces habrá dos *headers* que describen ese cuerpo:

Content-Type: MIME-type del cuerpo, como *text/html* or *image/gif*.

Content-Length: número de bytes en el cuerpo.



El protocolo HTTP - métodos

Además de **GET**, los métodos más usados son **HEAD** y **POST**.

El método **HEAD** solicita al servidor únicamente los headers de la respuesta, sin cuerpo.

Permite ver si existe algún recurso, ver si ha sido modificado, etc.

El método **POST** es esencial para la interacción web.

Este método envía información al servidor para ser procesada por quien corresponda, como un script CGI o PHP.

Procesará los datos enviados

```
POST /empleados/despedir.cgi HTTP/1.0
From: burns@snppl.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 23
```

nombre=Homero§or=7G

Cuerpo del mensaje

El protocolo HTTP – POST vs GET

```
POST /empleados/despedir.cgi HTTP/1.0
From: burns@snppl.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 23
```

nombre=Homero§or=7G

```
GET /empleados/despedir.cgi?nombre=Homero&sector=7G HTTP/1.0
```

→ Síndrome de mensaje

GET debe usarse preferentemente cuando los datos son pocos.

En otro caso debe usarse el método **POST**.

En ambos casos, el URL será codificado (*URL-encoded*) para transmitir los caracteres especiales (espacio, &, %, etc).

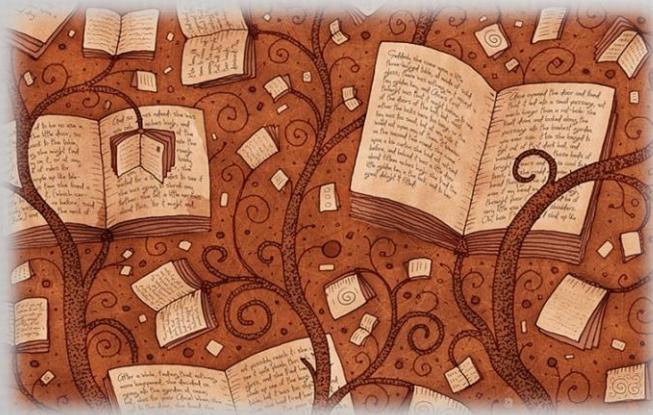
Documentos

Recordemos:

la Web surge como un servicio de Internet para estudiar y explorar documentos multimediales.



*documentos de texto, con imágenes y/o sonido.
facilidades de exploración de información relacionada.*



"The frontiers of a book are never clear-cut, because it is caught up in a system of references to other books, other texts, other sentences: it is a node within a network of references"

*The archeology of knowledge
Michel Foucault, 1969*

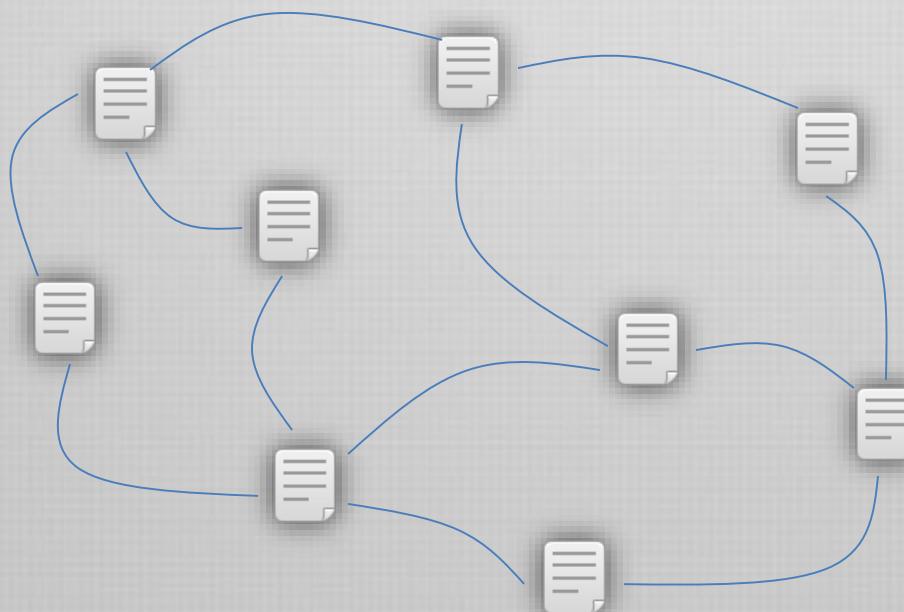
Esto requiere una forma especial de estructurar la información, para interrelacionarla y visualizarla adecuadamente.

Hipertexto

Sabemos que la idea de *hipertexto* es anterior al surgimiento de la web, e incluso de las redes globales...

"Hypertext is a non-sequential writing, a text that branches and allows choices to the reader, best read at an interactive screen. As popularly conceived, this is a series of text chunks connected by links which offer the reader different pathways"

Theodor Nelson (1960)



HTML

Propuesta de Berners-Lee:

Enriquecer un documento con marcas especiales para estructurar, vincular y componer la información, de manera tal de “convertirlo” en hipertexto.



Así surge HTML : *HyperText Markup Language*.

El aporte de Berners-Lee no se limita al HTML.

De hecho, ya existían lenguajes de marcado (SGML- Standard generalized markup language). Su propuesta es una **red global de documentos**, unidos por

“global hypertext links”

Berners-Lee implementa el primer browser de documentos HTML y el primer servidor HTML de la historia, en una computadora NeXT.

Estructura de HTML

Las marcas (*tags*) se encierran entre < y >

La versión original (de Berners-Lee) era simple, pero con posterioridad diferentes navegadores comenzaron a agregar su propias modificaciones (parte de la famosa “browsers-war”).

Actualmente la definición de HTML de la W3C es estricta, pero los navegadores siguen procesando documentos sin la estructura apropiada.

Cada tag tiene un nombre y posiblemente varios atributos (**attr=valor**)



Primera versión de HTML

Tags de la primera versión de HTML:

<TITLE> ... **</TITLE>** *Título del documento*

**** ... **** *Hipervínculo.*

<ISINDEX> *Indicador de una página índice*

<PLAINTEXT> *Texto a ser tomado en forma literal*

<LISTING> ... **</LISTING>** *Texto de font de tamaño fijo, tomado literalmente*

<P>, **<H1>**, **<H2>**, **<H3>**, **<H4>**, **<H5>**, **<H6>** *Párrafo y encabezados*

<ADDRESS> **text** ... **</ADDRESS>** *Direcciones, contacto, footers*

<HP1>...</HP1> <HP2>... </HP2> *Texto resaltado. Ya no se usa.*

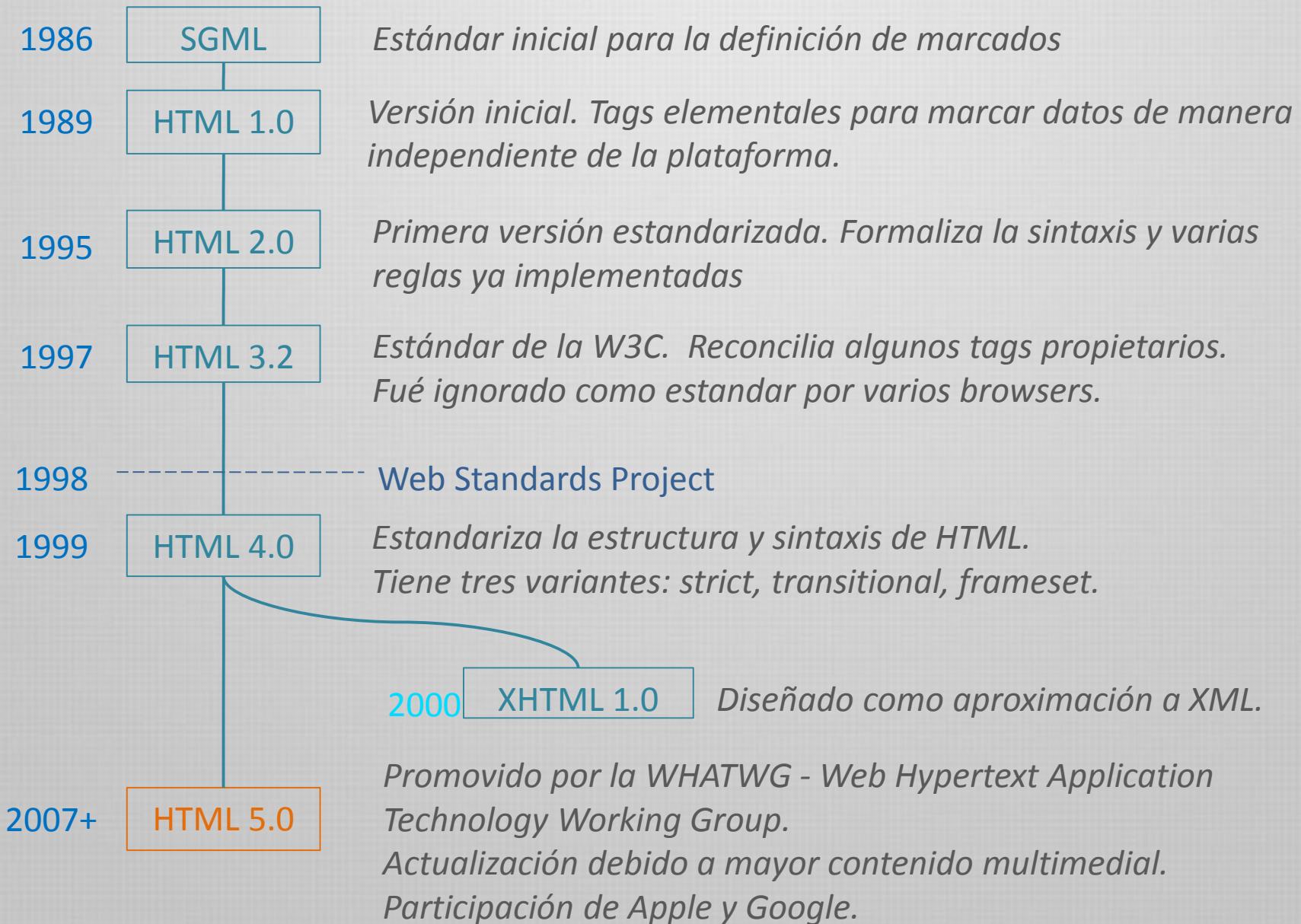
Listas descriptivas

```
<DL>
<DT>Term<DD>definition paragraph
<DT>Term2<DD>Definition of term2
</DL>
```

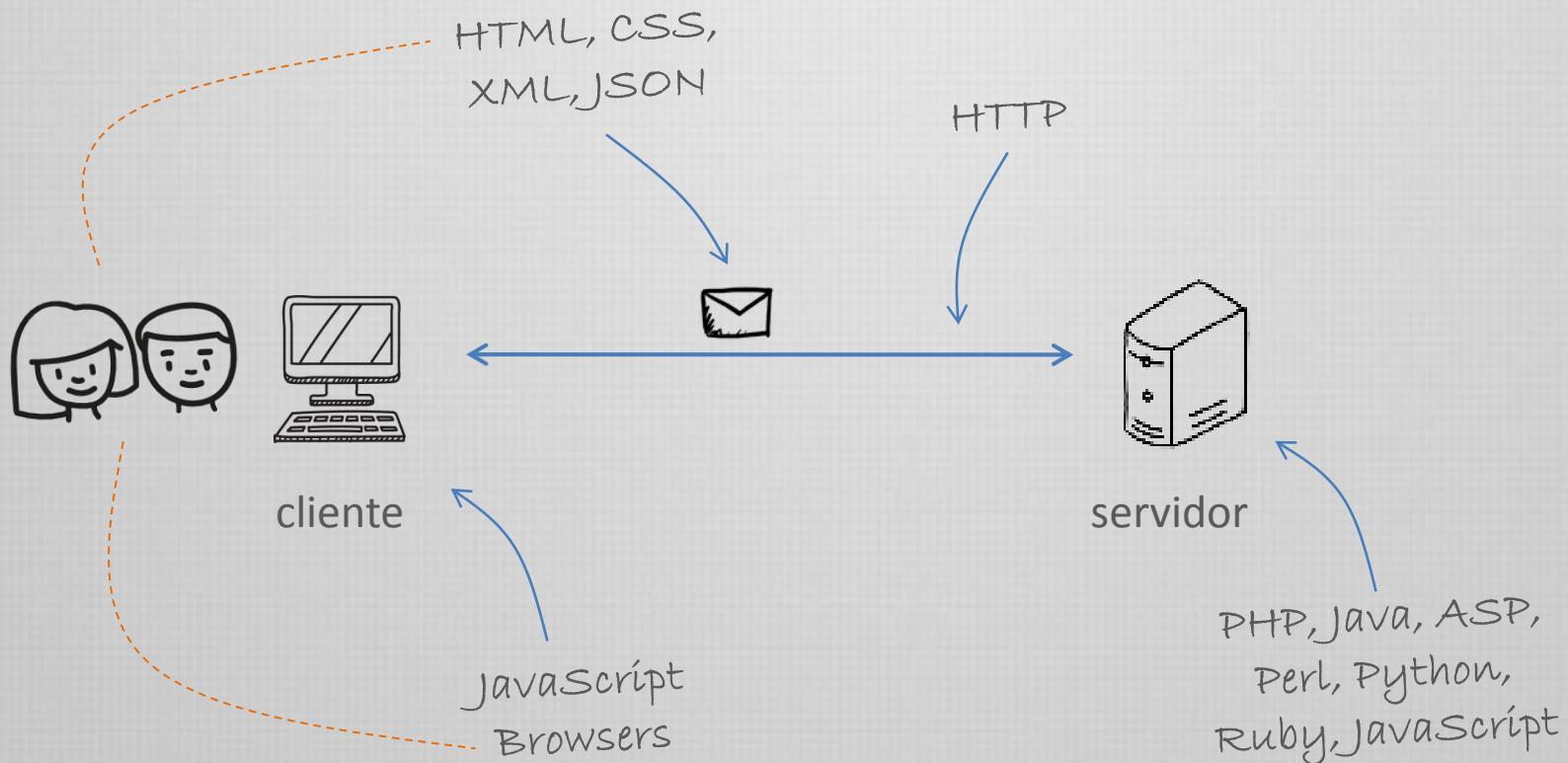
Listas no numeradas

```
<UL>
  <LI> list element
  <LI> another list element
</UL>
```

Evolución de HTML



Tecnologías web



XML

XML es el **lenguaje de marcado extensible**. (*eXtensible Markup Language*)

Es extensible porque los tags son definibles por el usuario

- Es la base de la *interoperabilidad* de muchos sistemas.
- Es el estándar para muchas tecnologías web (Ajax, Servicios web, etc)
- Es una recomendación de la W3C desde 1998.

La idea central detrás de XML es proveer una forma de

- estructurar información general,
- con independencia de la plataforma.

Facilita intercambio, procesamiento, claridad de datos.

```
<materia>
    <curricular/>
    <nombre>Historia</nombre>
    <profesor>
        Horacio Montesano
    </profesor>
    <horarios>
        <horario dia="lunes" desde="8" hasta="10"/>
        <horario dia="martes" desde="8" hasta="10"/>
    </horarios>
</materia>
```

XML - historia

XML no es el primer lenguaje de marcado.

GML

IBM - 1973



SGML

ANSI - 1986



HTML

W3C - 1990

Generalized Markup Language

Goldfarb, Mosher, Lorie

Standard Generalized Markup Language

Un meta-lenguaje para definir lenguajes de marcas.

Características distintivas:

- *Enfasis en marcado descriptivo, no procedural*
- *Concepto de tipo de documento*
- *Independencia del sistema*

XML - historia

XML nace entre la complejidad de SGML y las limitaciones de HTML

Los objetivos principales son:

- 1. XML debe usarse sin complicaciones sobre Internet.*
- 2. XML debe soportar una gran variedad de aplicaciones.*
- 3. XML debe ser compatible con SGML.*
- 4. Debe ser fácil escribir programas que procesan documentos XML.*
- 5. El número de características opcionales en XML debe mantenerse al mínimo.*
- 6. Los documentos XML deben ser legibles por el humano y razonablemente claros.*
- 7. El diseño XML debería ser preparado rápidamente.*
- 8. El diseño de XML debe ser formal y conciso.*
- 9. Los documentos XML deben ser fáciles de crear.*
- 10. El laconismo es de importancia mínima.*

XML - caracteres

XML utiliza caracteres Unicode (conjunto de caracteres de 32 bits <http://www.unicode.org>).
Principalmente UTF-8.

Algunos caracteres no pueden usarse literalmente.

Character reference ----> codificación del carácter literal por su equivalente numérico

A = A © = © é = é

Caracteres obligatoriamente no literales:

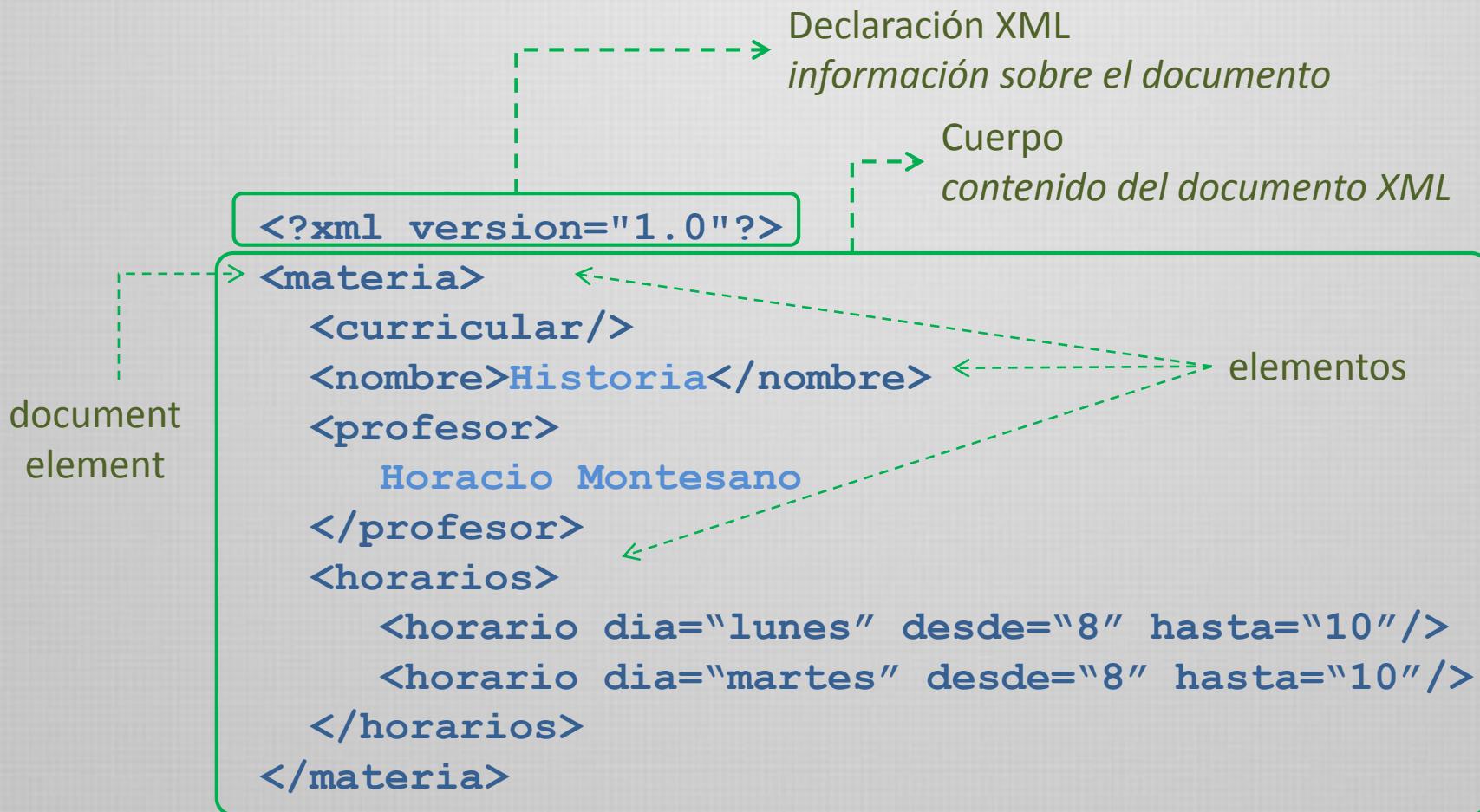
< < <
& & &
> > >
" " "
' ' '

Algunos caracteres no visibles tienen una importancia especial en XML...

Whitespaces ----> space
tab 	
carriage return 
line feed

XML es *case-sensitive*.

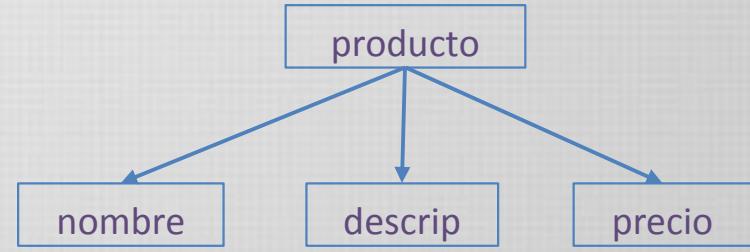
XML - anatomía



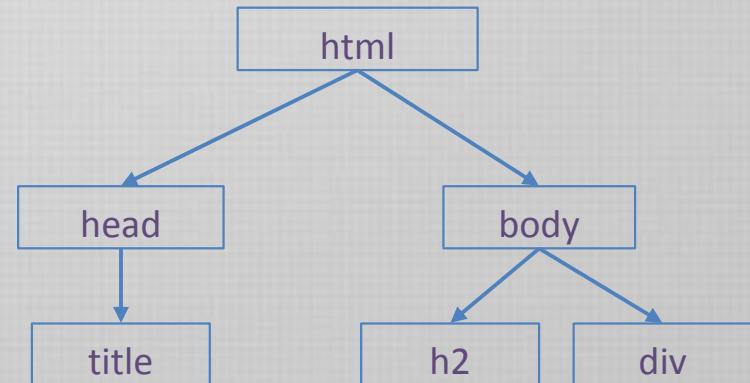
Estructurando documentos

HTML, XML y XHTML son estándares orientados principalmente a la representación y estructuración de información con algún fin particular

```
<producto>
  <nombre>PS3</nombre>
  <descrip>Buenisima</descrip>
  <precio>3000</precio>
</producto>
```



```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h2>Blog</h2>
    <div>Bla bla bla</div>
  </body>
</html>
```



HTML incluye algunos tags con semántica de presentación incorporada

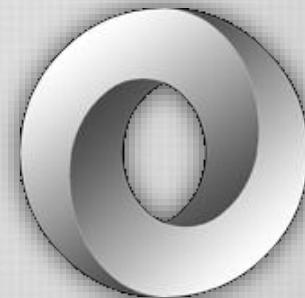
JSON = JavaScript Object Notation

Es un formato de intercambio de datos.

Minimal y textual.

Es una popular alternativa a XML.

*Es un **formato**, no un lenguaje*



Basado en un subconjunto de JavaScript, Standard ECMA-262

A veces denominados inicializadores en JavaScript

Usa convenciones de la familia de lenguajes C.

C, C++, Java, C#, JavaScript, Perl, PHP, Python, ActionScript.

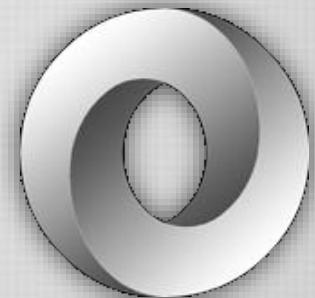
No es un formato de serialización de datos.

No admite estructuras cíclicas ni la inclusión de funciones

json.org

JSON es un buen formato de intercambio de datos

- Es fácilmente **interpretable** por el humano y por las computadoras
- **Poca decoración** a los datos, lo que facilita su transporte e interpretación algorítmica.
- Soporte *Unicode*, lo que lo hace **universal** en su contenido
- Representación de **estructuras de datos elementales**.
- Valores simples universales
- Facilidad de mapeo de estructuras nativas a notación JSON.



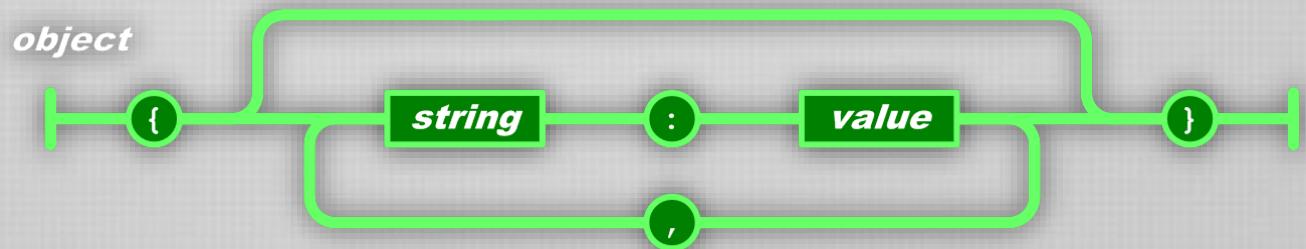
JSON

Consiste básicamente de dos estructuras de datos *universales*:

- Colección de pares nombre-valor
En muchos lenguajes: objetos, records, structs, etc
- Lista ordenada de valores
En muchos lenguajes: listas, arreglos, vectores, etc

En JSON, estas estructuras son:

- Objetos



- Arrays



JSON

Un valor JSON puede ser

- *string*
- *number*
- *object*
- *array*
- *true*
- *false*
- *null*

- Secuencia de cero o más caracteres Unicode entre comillas dobles
- Caracteres especiales escapados con \

- enteros, reales
- notación científica

- secuencia ordenada
- JSON no habla de indexado.

- dato nulo

```
var unObjeto = {"id": 1234, "nombre": "Juan  
Fulano", "alumnoRegular": true, "ultimosCursos": [7145, 7854, 7321  
, "origen": {"ciudad": "Bahia Blanca", "provincia": "Buenos Aires"} ]}
```

JSON - arrays

```
var arreglo = [ "Argentina",
    "Brasil",
    "Chile",
    "Uruguay",
    "Ecuador",
    "Colombia"
]
```

```
var horarios = [
    ["martes", "jueves"],
    [14, 14],
    [18, 18]
]
```

JSON - Objetos

```
var unAlumno = {  
    "id": 1234,  
    "nombre": "Juan Fulano",  
    "alumnoRegular": true,  
    "ultimosCursos": [  
        7145,  
        7854,  
        7321  
    ],  
    "origen": {  
        "ciudad": "Bahia Blanca",  
        "provincia": "Buenos Aires"  
    }  
}
```

JSON - Objetos

```
var unAlumno = {  
    "id": 1234,  
    "nombre": "Juan Fulano",  
    "alumnoRegular": true,  
    "ultimosCursos": [  
        7145,  
        7854,  
        7321  
    ],  
    "origen": {  
        "ciudad": "Bahia Blanca",  
        "provincia": "Buenos Aires"  
    }  
}
```

Los campos de los
objetos van entre
comillas dobles

JSON - Objetos

```
var unAlumno = {  
    "id": 1234,  
    "nombre": "Juan Fulano",  
    "alumnoRegular": true,  
    "ultimosCursos": [  
        7145,  
        7854,  
        7321  
    ],  
    "origen": {  
        "ciudad": "Bahia Blanca",  
        "provincia": "Buenos Aires"  
    }  
}
```

Los valores son marcados de acuerdo a su tipo

JSON - Objetos

```
var unAlumno = {  
    "id": 1234,  
    "nombre": "Juan Fulano",  
    "alumnoRegular": true,  
    "ultimosCursos": [ 7145,  
                      7854,  
                      7321  
                    ],  
    "origen": {  
        "ciudad": "Bahia Blanca",  
        "provincia": "Buenos Aires"  
    }  
}
```

Un arreglo de enteros

JSON - Objetos

```
var unAlumno = {  
    "id": 1234,  
    "nombre": "Juan Fulano",  
    "alumnoRegular": true,  
    "ultimosCursos": [  
        7145,  
        7854,  
        7321  
    ],  
    "origen": {  
        "ciudad": "Bahia Blanca",  
        "provincia": "Buenos Aires"  
    }  
}
```

Un objeto con dos campos

JSON - Objetos

```
var unAlumno = {  
    "id": 1234,  
    "nombre": "Juan Fulano",  
    "alumnoRegular": true,  
    "ultimosCursos": [  
        7145,  
        7854,  
        7321  
    ],  
    "origen": {  
        "ciudad": "Bahia Blanca",  
        "provincia": "Buenos Aires"  
    }  
}
```

unAlumno.id
unAlumno.nombre
unAlumno.ultimosCursos[0]
unAlumno.origen.ciudad

MIME Media Type

Como JSON se utiliza como **formato de transmisión de datos** serializados, es necesario indicar su **tipo** cuando es transmitido (HTTP messages)

Content-Type: -----> **application/json**

A veces llamado **MIME Type**

-----> *Multipurpose Internet Mail Extensions*

Inicialmente referido al protocolo
SMTP y la transmisión de datos no-
ASCII agregados al mail.
Hoy propagado a otros protocolos.

Interpretando JSON

El lenguaje utilizado debe ser capaz de comprender y generar estructuras JSON.

Los *decoders* aceptan descripciones JSON bien formadas y las traducen a estructuras de datos nativas del lenguaje.

Los *encoders* traducen estructuras nativas a expresiones JSON.

JavaScript es un *decoder* natural de JSON :)
No se emplea tiempo de parsing ni transformación

Otros lenguajes ofrecen librerías para operar con datos JSON.

JSON

JSON no posee **validadores**, a diferencia de XML.

JSON permite una flexibilidad mayor que XML en algunos aspectos

Permite agregar campos a una estructura sin perturbar el código existente

JSON ha demostrado ser eficiente en *el intercambio de datos entre servidores y clientes, con tecnologías heterogéneas*

Estructura vs. presentación

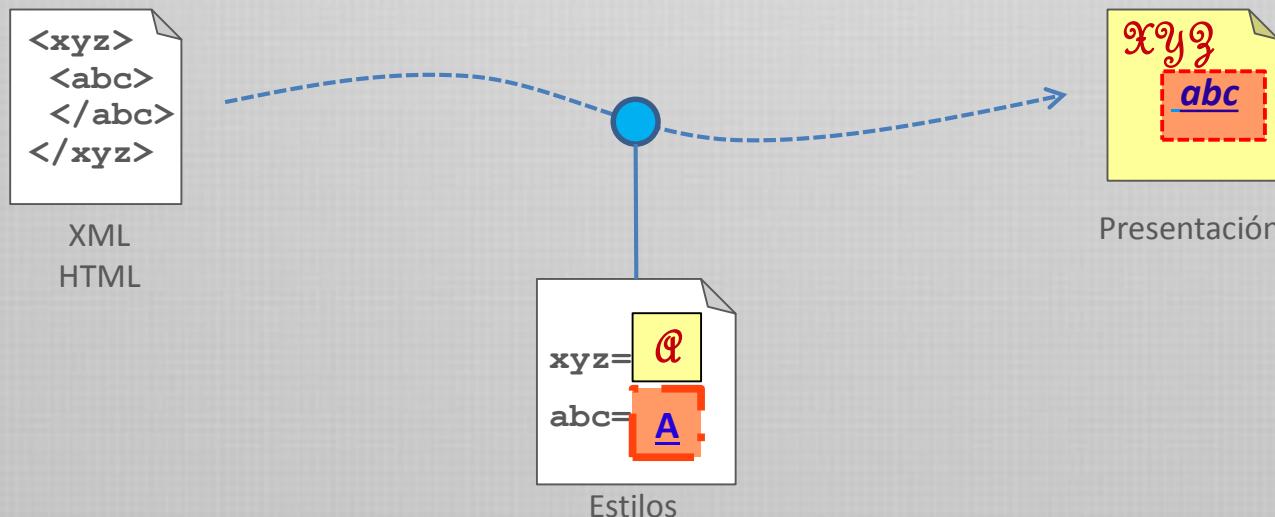
La W3C presenta *algunas* guías para la visualización de documentos HTML, basada en la interpretación de algunos tags.

- un elemento `strong` se verá en font **bold**
- un elemento `h1` se verá de mayor tamaño que uno de `h2`.

El estándar XML no incluye nada de aspectos de presentación , pues es un lenguaje de marcado de propósito general.

Tal vez no sea información destinada a la visualización.

Sin embargo, cuando estructuramos información destinada (de alguna manera u otra) a una interfaz visual, es deseable poder estilizarla adecuadamente.



Estilos

Existen dos mecanismos para agregar estilos a documentos de tags.

- **CSS – Cascading Style Sheets**

- *Documento de texto con sintaxis simple.*
- *Es aplicable a HTML y a XML.*
- *Permite aplicar formato a elementos (tags) – por tipo, nombre o identificador*
- *Permite heredar formatos de elementos contenedores.*

```
p {  
    margin-top: 8pt;  
    margin-bottom: 8pt;  
    font-size: 75%  
}
```

- **XSL – Extensible Stylesheet Language**

- *Documento de texto con sintaxis XML.*
- *Más que un mecanismo de estilos: transforma documentos XMLs.*
- *Por ser un mecanismo de transformación, se centra más en el uso de templates que en formatos de elementos individuales.*

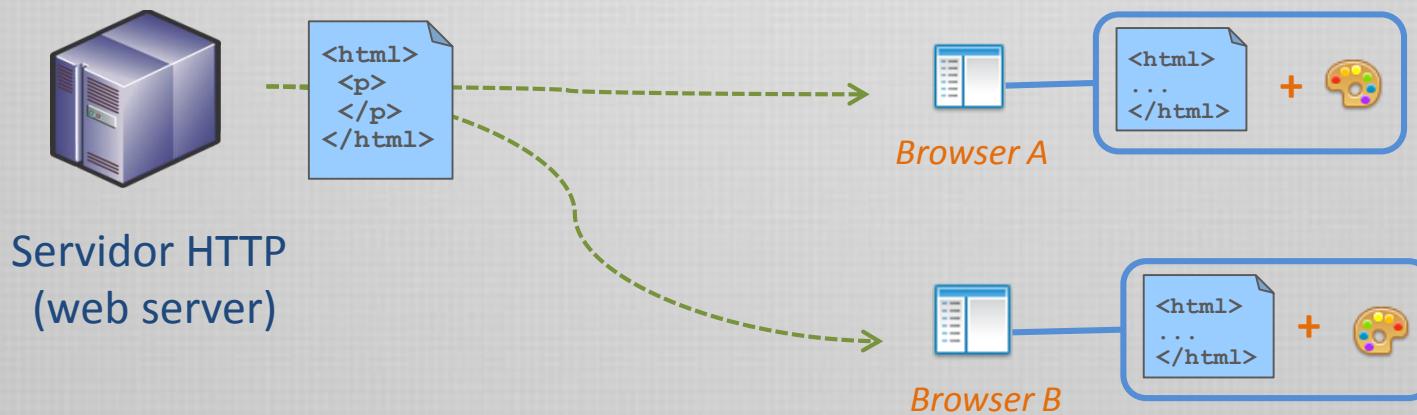
```
<xsl:template match="p">  
    <fo:block  
        space-before="8pt"  
        space-after="8pt"  
        font-size="75%">  
        <xsl:apply-templates/>  
    </fo:block>  
</xsl:template>
```

CSS – Cascading Style Sheets

Los estilos CSS nacen varios años después que HTML, a mediados de los 90.

Sin embargo, la separación *HTML-estilos de presentación* ya era considerada por Berners-Lee desde un comienzo.

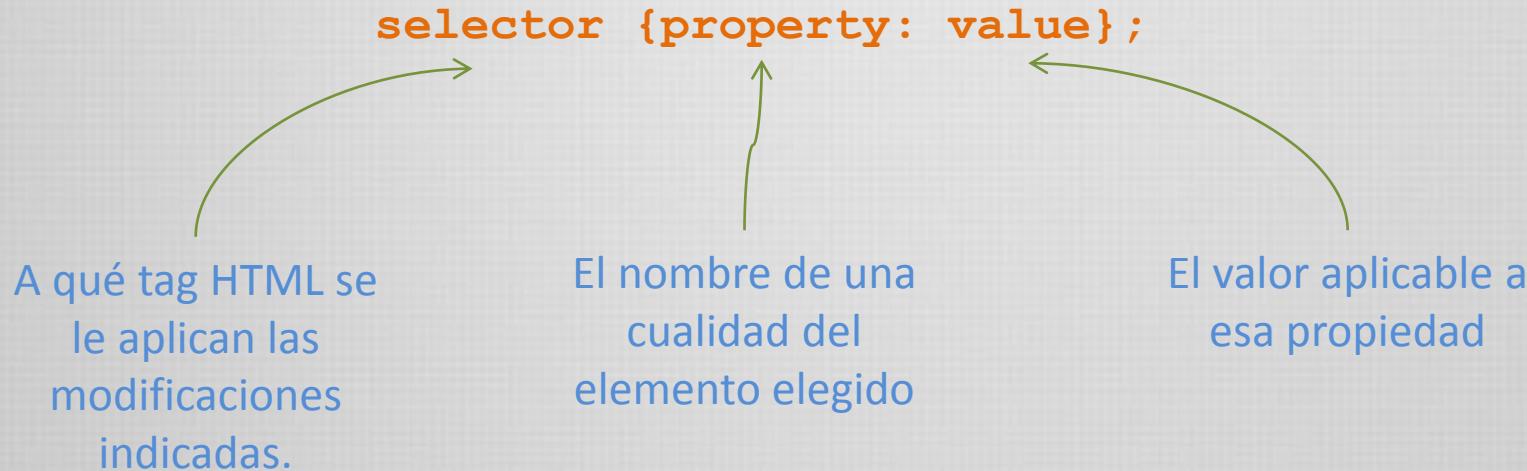
*El primer browser implementaba una lista de estilos universales. Berners-Lee suponía que todos los futuros browsers harían lo mismo.
NCSA Mosaic permitía ciertas personalizaciones de estilo*



En 1994 sale el primer borrador de CSS, hojas de estilo aplicables a HTML.

Cascading Style Sheets

Un documento CSS es simplemente una colección de reglas de la forma:



Por ejemplo,

```
p {  
color: blue;  
text-align: right;  
font-family:  
courier;  
}
```

```
hr {  
color : #91B2C5;  
height : 1px;  
width : 100%;  
}
```

```
td, tr, div {  
font-family : Arial;  
font-size : x-small;  
color : #000;  
}
```

Son características *heredables*.

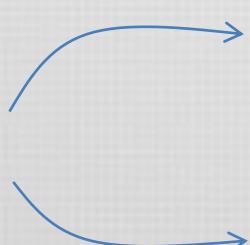
*Se puede indicar un font para un párrafo, y aparte un color para los links.
Luego, cada link usa sus propiedades y las del párrafo.*

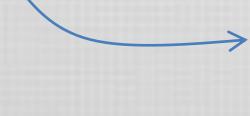
Cascading Style Sheets

Pueden categorizarse diferentes usos de un mismo **elemento**:

```
p.bodytext {color: black;}  
p.alert {color: red;}
```

Luego se indica, para ese **elemento**, de qué categoría es:

 <p class="bodytext">
 Esto va de color negro
</p>

 <p class="alert">
 Esto va de color rojo.
</p>

También pueden definirse elementos aplicables a cualquier tag. En este caso no se nombra el tag específico sino la *clase* de formato aplicada:

```
.highlight {color: blue;}
```

Esto se puede aplicar a cualquier tag!

```
<p class="highlight">Esto es texto azul</p>  
<h2 class="highlight">Esto es un encabezado azul</h2>
```

Cascading Style Sheets

CSS se puede aplicar de formas diferentes:

- Directamente en el elemento que se quiere formatear.
- Incluído al comienzo en el documento HTML (con el tag `<style>`), o combinado con los tags `<div>` y ``.
- En archivos separados del documento HTML.

● **Inline style**

Se utiliza el atributo `style`, aplicable a cualquier tag HTML:

```
<b style="color: blue;">Esto es bold y azul</b> y esto no.  
<span style="color:green;">Esto es verde</span> y esto no.
```

● **Por tipo de elemento**

Se utiliza el tag `style`:

```
<style type="text/css">  
b { color: blue }  
</style>
```

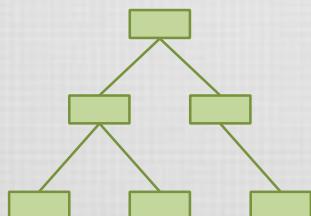
Todas las ocurrencias del tag bold `` también cambiará el texto a azul.

Procesando XML/HTML - modelo web - scripting

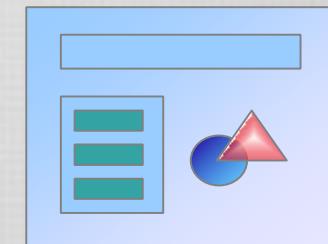
Documento

```
<html>  
  <p>  
  </p>  
</html>
```

Estructura del documento



Vista del documento



```
if()  
then  
else  
while
```



simple, dinámico

JavaScript

El más popular de los lenguajes de script para la programación del lado cliente.



Creado en sus comienzos por *Netscape* y luego en conjunto con *Sun*. *Microsoft implementó también el lenguaje, pero como es habitual lo hizo a su manera (Browsers Wars!).*

Se llamaba inicialmente *Mocha*, luego *Livescript*.

JavaScript es en realidad una implementación de ECMAScript, la creación original de Brendan Eich.

Brendan Eich



- Sintaxis simple, sin tipos de datos.
- Puede modificar dinámicamente el documento en el que se encuentra, por medio de una interfaz especial denominada DOM.
- Puede reaccionar ante eventos ocurriendo en la página en la que se encuentra (*onMouseOver, onClick, onLoad, etc.*).
- Puede ser utilizado para la validación de datos.
- Puede detectar el browser del visitante y ayudar a optimizar la visualización
- Puede realizar pedidos adicionales a un servidor, en *background*, en forma asincrónica (AJAX).

JavaScript

```
<html>
  <head>
    <title>Pagina</title>
  </head>
  <body>
    <h2>Bienvenido</h2>
    <div> etc </div>
    <script>
      //sentencias
    </script>
  </body>
</html>
```

sentencias de
alto nivel

El código será **interpretado por el navegador.**

El elemento tiene el código o una referencia a un recurso con código.

JavaScript - ejemplo

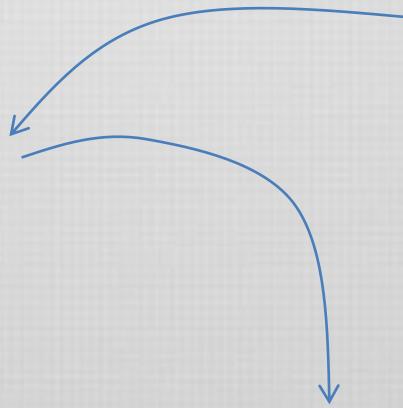
```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JS</TITLE>
  </HEAD>

  <BODY>
    <SCRIPT>
      var mostrar=true
    </SCRIPT>

    <p>Bienvenidos!</p>

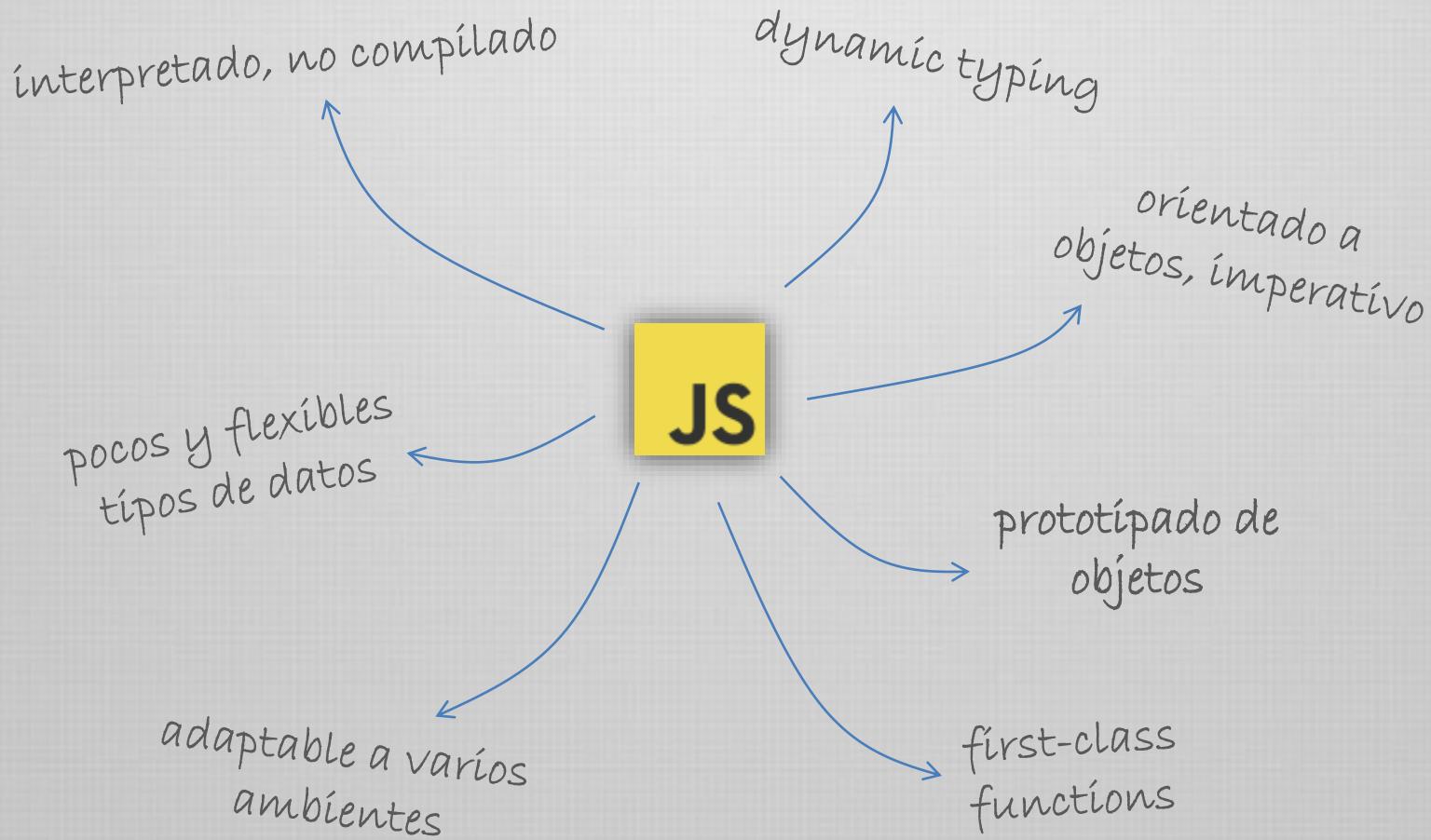
    <SCRIPT LANGUAGE="JavaScript">
      if (mostrar==true) { document.write("Hola! ") ; }
    </SCRIPT>

    <p>Fin.</p>
  </BODY>
</HTML>
```



continuidad

JavaScript



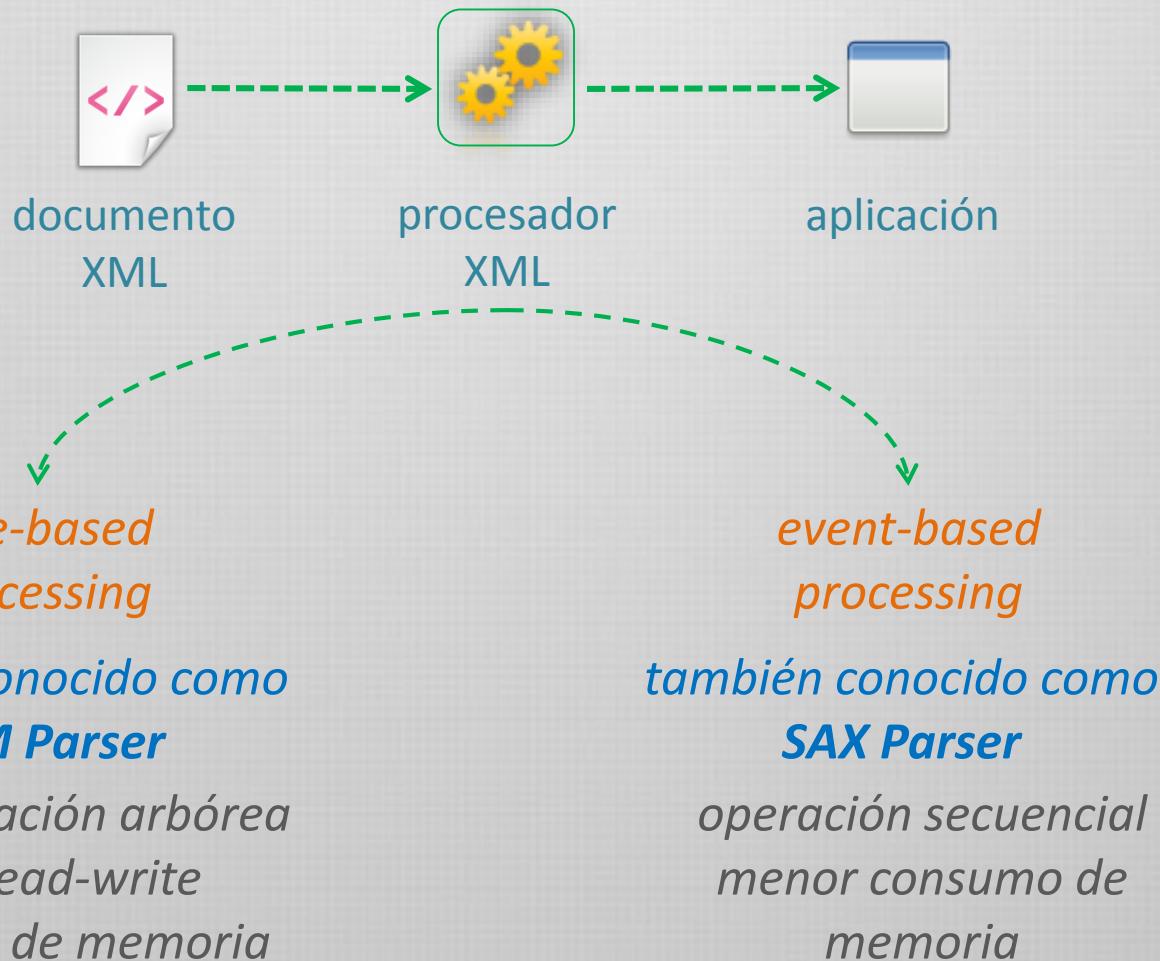
JavaScript

Ventajas principales del uso de JavaScript

- Menos interacción con el servidor
Pueden validarse datos antes de enviar el request al servidor.
- Respuesta inmediata al usuario
Parte del procesamiento puede “adelantarse” al lado cliente
- Reparación automática y de superficie de errores menores
Principalmente formato de datos, como las fechas
- Mejor experiencia de uso
Menues desplegables o colapsables, ayudas contextuales, etc
- Mayor interacción con el usuario
Le otorga mayor reacción al uso del mouse.
- Interfaces enriquecidas
Drag-and-drop, sliders, etc.
- Optimización de la interacción cliente-servidor
Parte de la información puede obtenerse en background, actualizando parcialmente la página en vista.

Procesando XML

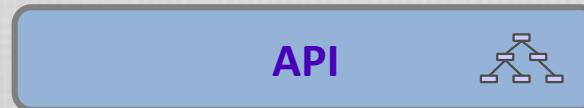
La especificación XML asume que el documento será procesado de manera especial



Procesando XML/HTML

*Documento
XML-HTML*

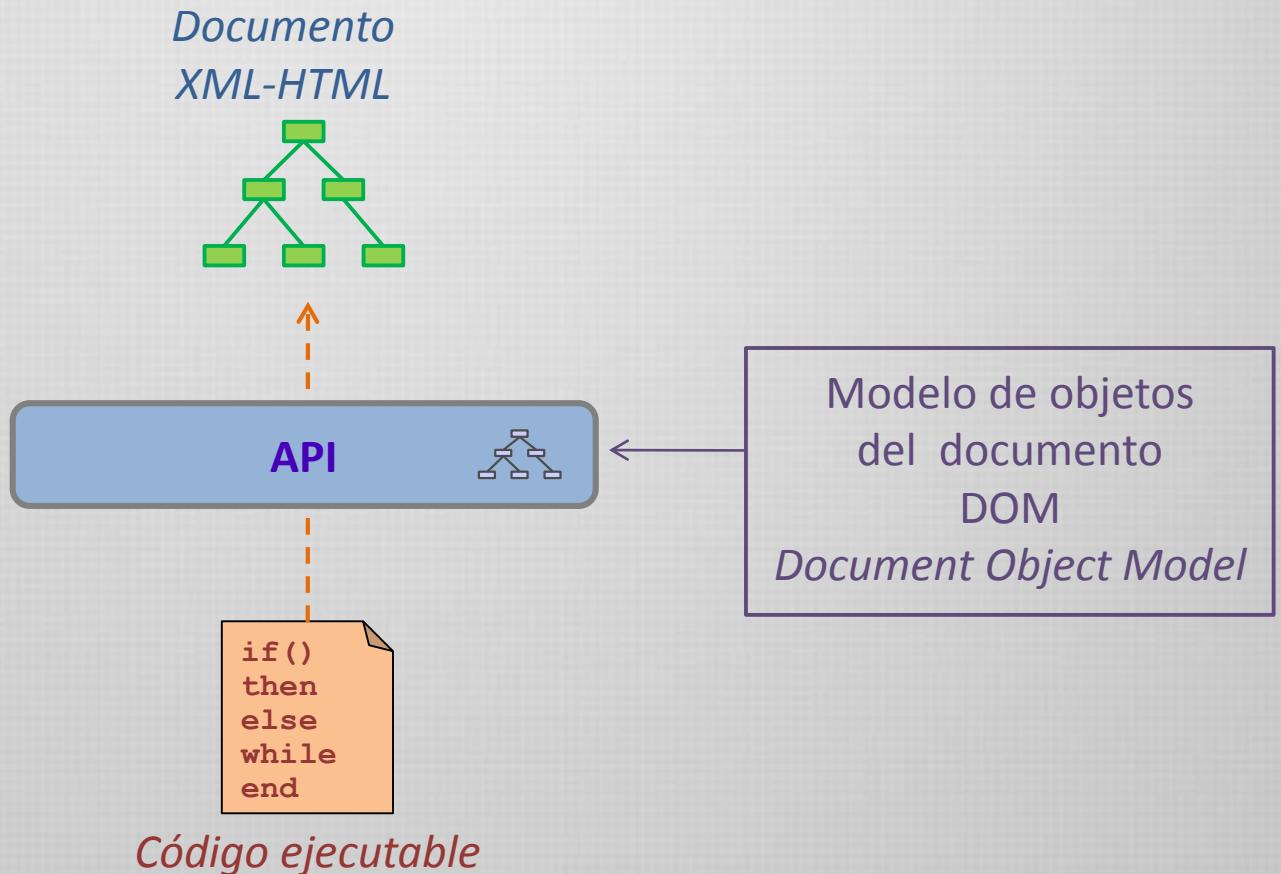
```
<html>
  <p>
  </p>
</html>
```



```
if()
then
else
while
end
```

Código ejecutable

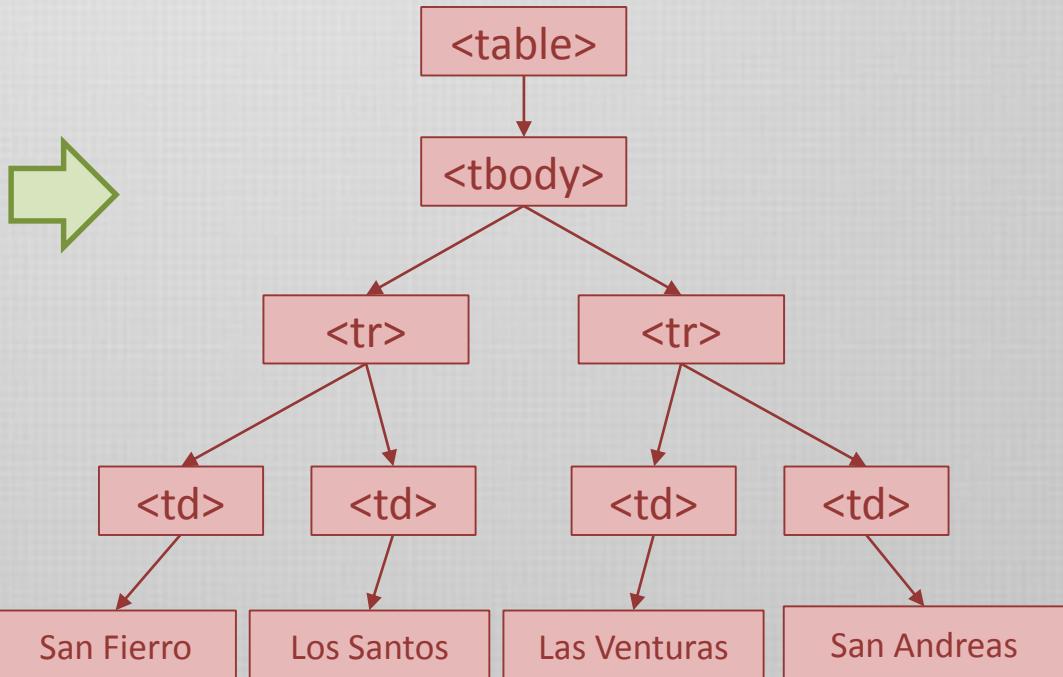
Procesando XML/HTML



Document Object Model - DOM

DOM es una interfaz de acceso y manipulación de documentos XML y HTML bien formados.
Es independiente de la plataforma, del navegador y del lenguaje que lo utiliza.
Es un estándar de la W3C desde 1998.

```
<table>
  <tbody>
    <tr>
      <td>San Fierro</td>
      <td>Los Santos</td>
    </tr>
    <tr>
      <td>Las Venturas</td>
      <td>San Andreas</td>
    </tr>
  </tbody>
</table>
```



Document Object Model - DOM

El estándar DOM identifica:

- Las interfaces y objetos usados para representar y manipular un documento.
- La semántica de estas interfaces y objetos (comportamiento y atributos)
- La relación y las colaboraciones entre estas interfaces y objetos



El estándar DOM no es una especificación binaria de implementación.

No es una forma de persistir objetos a XML.

No define la semántica interna de los XML.

El estándar DOM no es una estructura de datos

Document Object Model - ejemplo

```
<html>
<head>
<title></title>
</head>
<body><p>Este es un párrafo</p></body>
</html>
```

La siguiente expresión es equivalente al nombre "P"

document.documentElement.lastChild.firstChild.tagName

Tag HTML de la página

Tag BODY

1er. Elemento
del tag BODY

Nombre del 1er.
elemento del tag
BODY

Un acceso más simple puede conseguirse ante la posibilidad de nombrar los elementos de un documento HTML...

Document Object Model - ejemplo

```
...  
<p id="miParrafo">Este es un párrafo.</p>  
...
```

En este caso la expresión para acceder al nombre del tag es:

```
document.getElementById("miParrafo").tagName
```

También pueden accederse a todos los tags de un mismo tipo....

```
var nodeList = document.getElementsByTagName("A");  
for (var i = 0; i < nodeList.length; i++)  
    nodeList[i].style.color = "#ff0000";
```

DOM – Propiedades de nodos

| Propiedad | Descripción |
|---|--|
| nodeName | <i>Nombre de nodo. Usualmente el nombre del tag</i> |
| nodeValue | <i>Valor del nodo. Dependiente del tipo del nodo</i> |
| nodeType | <i>Tipo del nodo, indicado como un entero.</i> |
| childNodes | <i>Un arreglo de los nodos hijos de un nodo</i> |
| parentNode | <i>El ancestro inmediato de un nodo</i> |
| firstChild, lastChild | <i>El primer hijo de un nodo</i> <i>El último hijo de un nodo</i> |
| nextSibling, previousSibling | <i>Hermanos de un nodo</i> |
| attributes | <i>Un arreglo de atributos del nodo, si es un elemento</i> <i>Si no, vacío.</i> |

DOM – Propiedades de nodos

| Propiedad | Descripción |
|-----------------------------------|---|
| <code>documentElement</code> | <i>Referencia al nodo raíz del documento.</i> |
| <code>createElement()</code> | <i>Crea un nuevo elemento que puede insertarse en el documento. Solo creación.</i> |
| <code>createTextNode()</code> | <i>Crea un nuevo elemento de tipo texto que puede insertarse en el documento.</i> |
| <code>createCDATASection()</code> | <i>Crea una nueva sección CDATA que puede insertarse en el documento</i> |
| <code>getElementsByName()</code> | <i>Devuelve un arreglo de elementos que coinciden con el nombre indicado.</i> |
| <code>getElementById()</code> | <i>Devuelve una referencia a un nodo elemento en el documento, identificado por un Id particular.</i> |
| <code>innerHTML</code> | <i>El contenido HTML dentro de un dado elemento</i> |

Procesando XML - SAX

SAX significa *Simple Access for XML*.
Es un parser para XML basado en eventos.

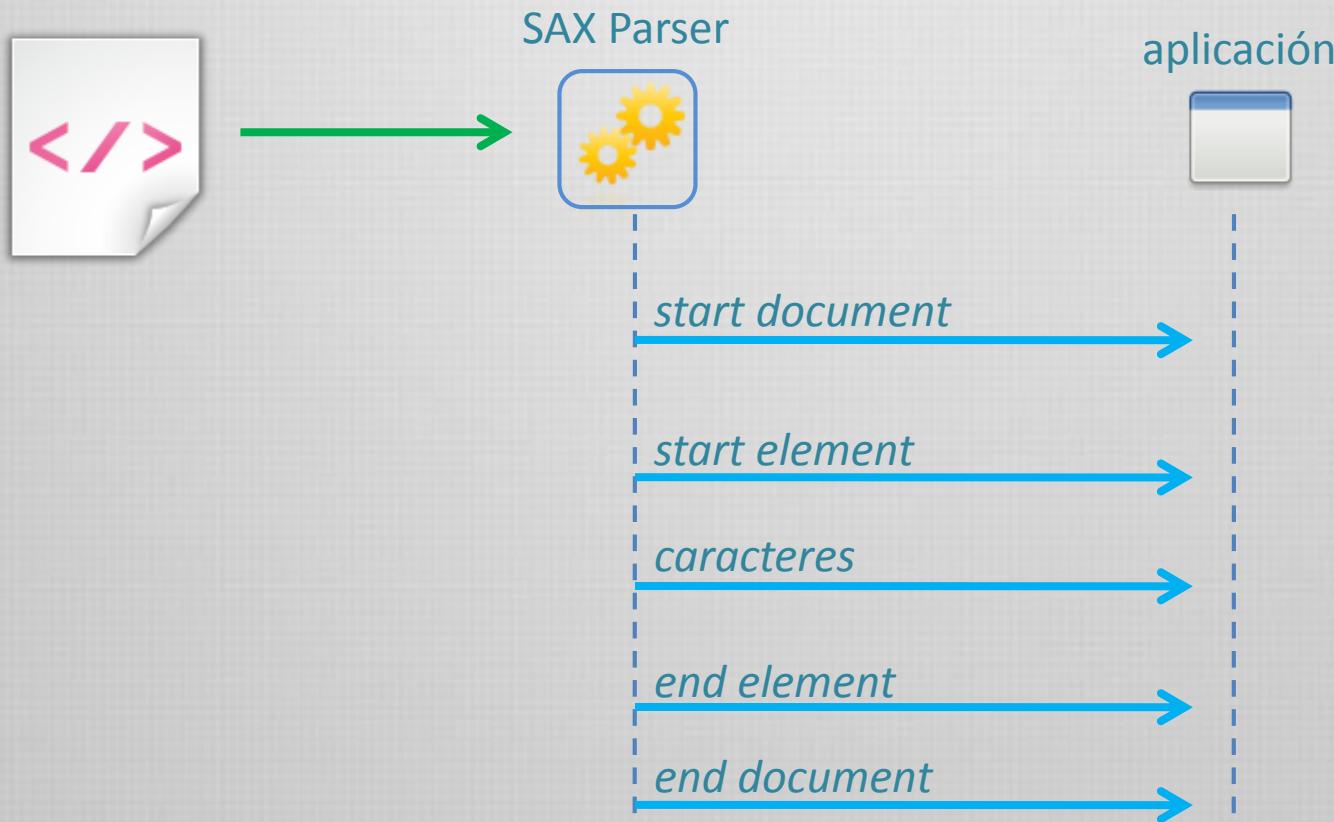
No existe una especificación formal, pero la implementación Java es referencial.

SAX opera sobre el documento en forma **secuencial**.

Notifica a la aplicación de **eventos** durante el procesado del XML.

- Es más natural para aplicaciones no muy centradas en XML
 - Es más eficiente en recursos que la aproximación DOM
 - La aplicación puede comenzar a procesar el XML mientras el parser lo lee
-
- No puede navegarse por el documento con la misma libertad que DOM
 - La aplicación debe registrar los eventos en los que está interesada.

Procesando XML - SAX



Procesando XML - SAX

```
<?xml version="1.0"?>
<saludo>
    <texto>
        Hola Mundo!
    </texto>
</saludo>
```

- 
1. start document
 2. start element: *saludo*
 3. start element: *texto*
 4. characters: *Hola Mundo!*
 5. end element: *texto*
 6. end element: *saludo*
 7. end document

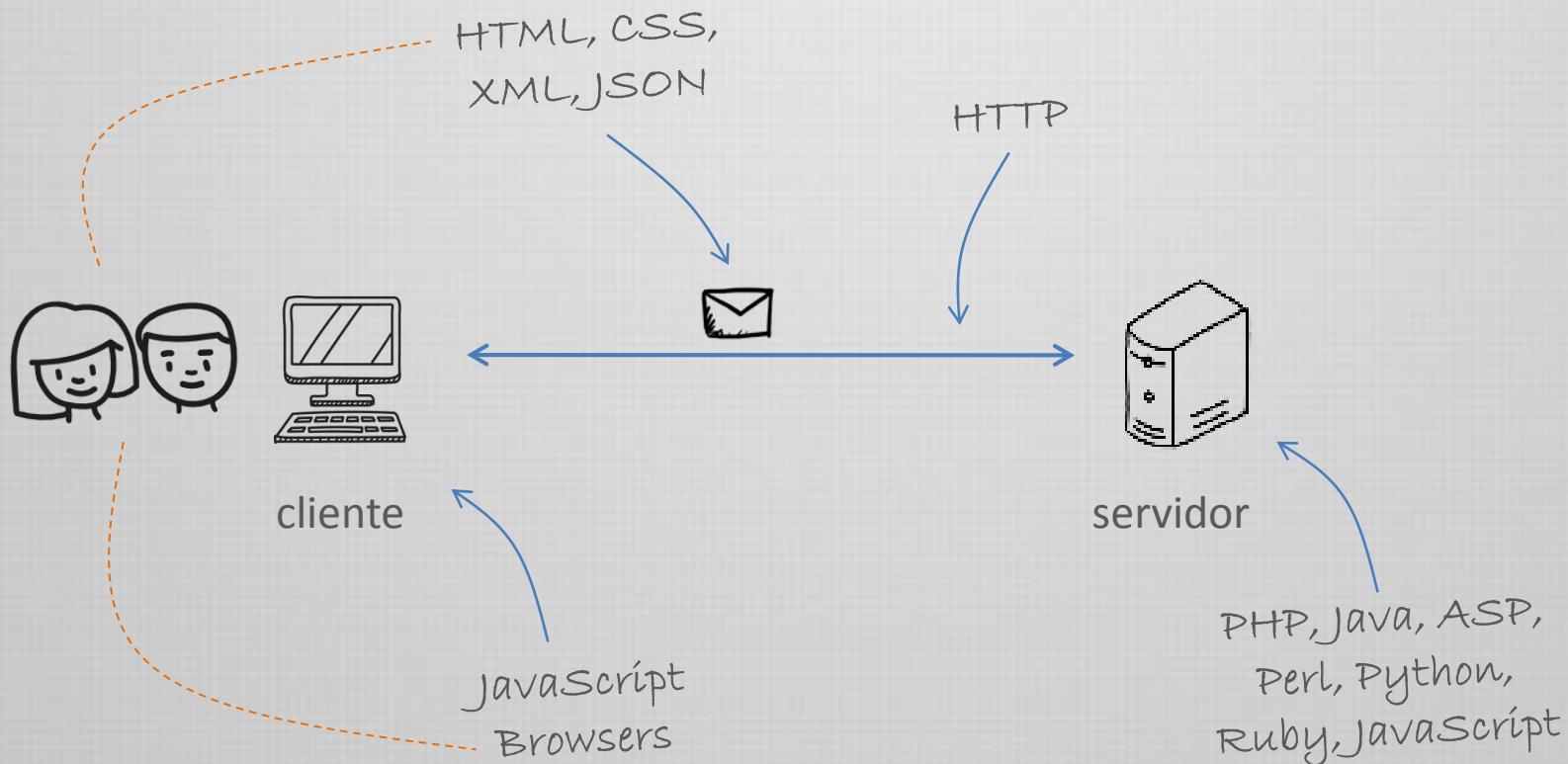
La aplicación debe registrar funciones *call-back* para atender estos eventos.

```
public void startElement(String uri, String localName, String qName,
                         Attributes attributes)

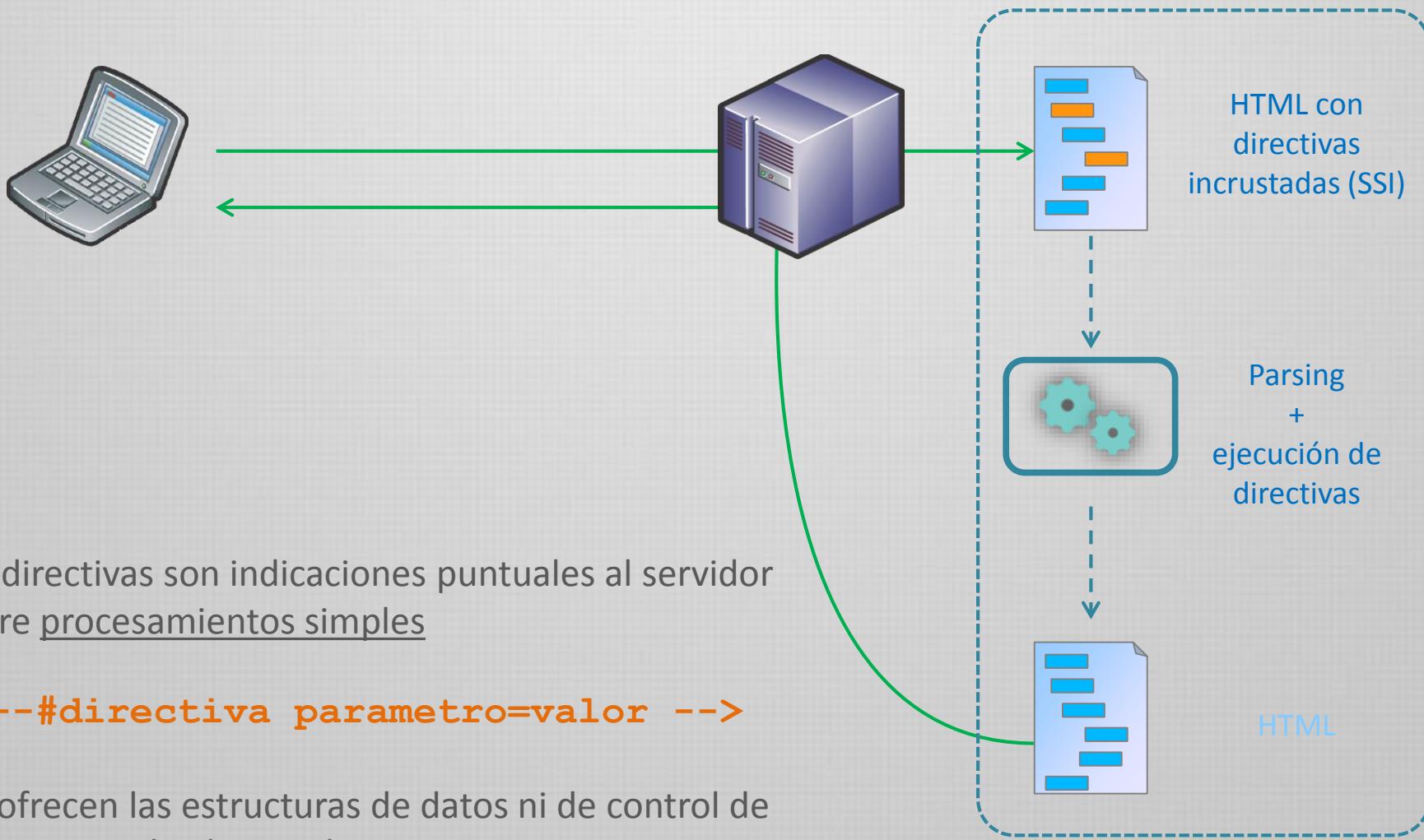
public void endElement(String uri, String localName, String qName)

public void characters(char[] ch, int start, int length)
```

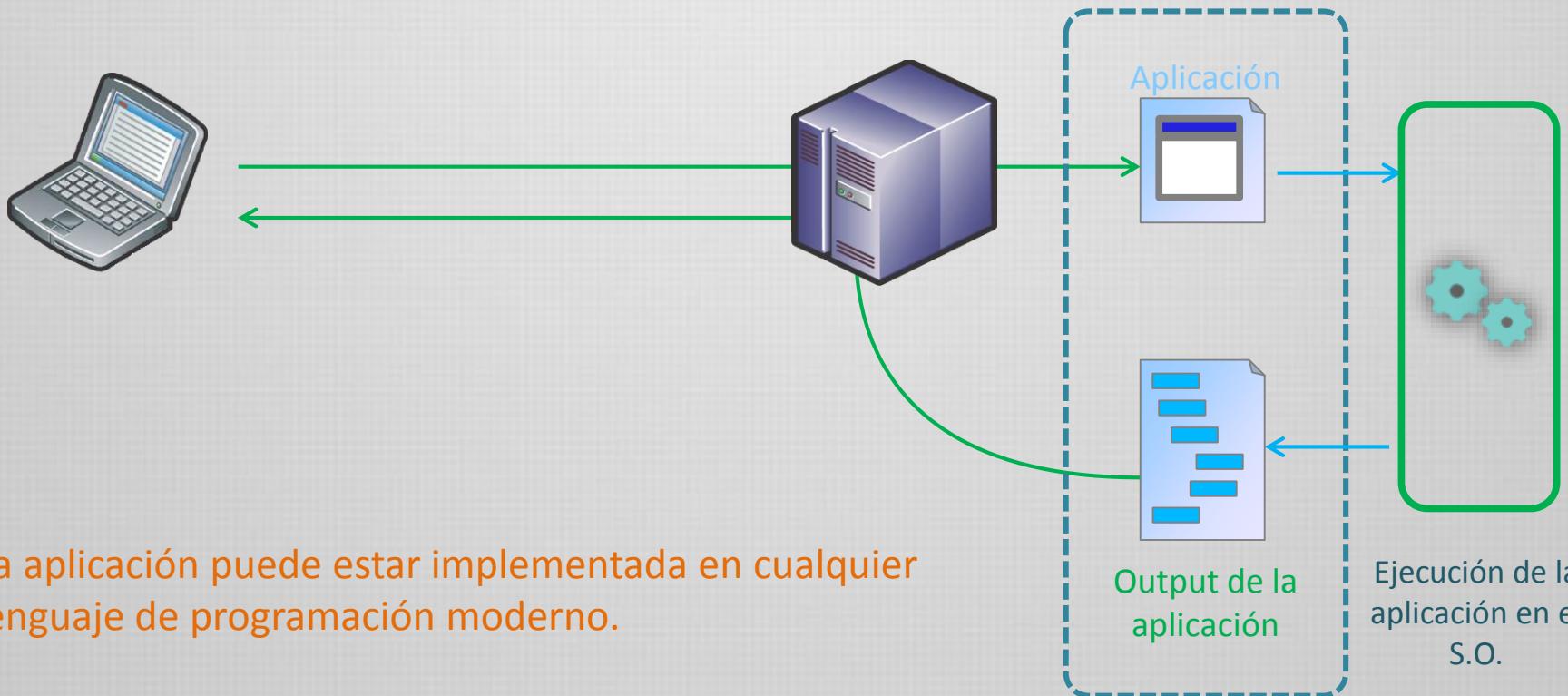
Tecnologías web



Procesando una respuesta – Server Side Includes



Procesando una respuesta – Aplicaciones CGI



La aplicación puede estar implementada en cualquier lenguaje de programación moderno.

La integración con el servidor es simple pero rústica, por medio de la entrada-salida estándar :(

La aplicación no necesariamente está basada en tecnologías para la web o Internet.

Procesando una respuesta – Scripting



El lenguaje PHP surge en 1995 bajo el nombre PHP/FI, creado por Rasmus Lerdorf.

Originalmente Rasmus creó el lenguaje como reemplazo de unos scripts Perl que había hecho para su página personal.

PHP significaba “*Personal Home Page*” y FI “*Form Interpreter*”.

Luego de liberar PHP, junto con **Zeev Suraski** y **Andi Gutmans** reescriben el intérprete y nace PHP3 en 1997, siendo testeado publicamente en 1998.

En 1999 reescriben el núcleo de PHP para obtener **Zend Engine**.

En 2004 sale PHP5, con varias modificaciones, entre ellas un mejor modelo de objetos, deficiente en la versión anterior.

Hoy es usado por muchos sitios gracias al desarrollo de varios productos, la mayoría GNU: **phpBB**, **WordPress**, **MediaWiki**, **Joomla**, **Mambo**, etc.



Rasmus Lerdorf

PHP

La sintaxis y las características orientadas a objetos del lenguaje PHP(5) son fáciles de aprender, pues guardan similitud con lenguajes vistos anteriormente.

| | | | |
|---|---|---|--|
| <code>if (cond) { ... } else { ... }</code> | <code>while (cond) { ... }</code> | <code>do{ ... } while (cond)</code> | <code>for (\$i=1; \$i<=5; \$i++) { ... }</code> |
|---|---|---|--|

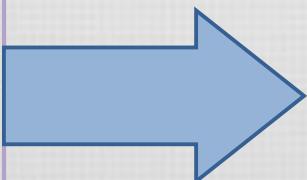
PHP es *inscrutable* en HTML, similar a JavaScript
Todo código restante HTML se envía tal cual al cliente.

| Tag inicial | Tag final |
|-------------------------|-----------|
| <?php | ?> |
| <? | ?> |
| <script language="php"> | ?> |
| <% | %> |

PHP

```
<html>
<head>
    <title>Ejemplo</title>
</head>
<body>
    <?php
        echo "Hola Mundo!";
    ?>
</body>
</html>
```

hola.php



```
<html>
<head>
    <title>Ejemplo</title>
</head>
<body>
    Hola Mundo!
</body>
</html>
```

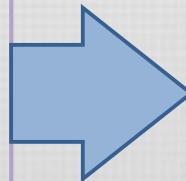
Resultado recibido por el cliente

PHP

El script .php no necesariamente debe incluir código HTML...

```
<?php  
echo "<HTML>";  
echo "<TITLE>Ejemplo</TITLE>";  
echo "<BODY>";  
echo "Hola Mundo!";  
if date('w')=5  
    { echo "Hoy es sabado"; }  
else  
    { echo "Hoy no es sabado"; }  
echo "</BODY></HTML>";  
  
?>
```

todophp.php



```
<HTML>  
<TITLE>Ejemplo</TITLE>  
<BODY>  
Hola Mundo! Hoy no es sabado  
</BODY>  
</HTML>
```

Resultado
recibido por el
cliente

PHP

Pueden implementarse funciones...

```
<?php
```

Las variables
comienzan con \$

```
function cuadrado($num) {  
    return $num*$num;  
}
```

```
$cartell1 = "El cuadrado de ";  
$cartel2 = " es ";  
$numero=4;
```

```
echo $cartell1;  
echo $numero;  
echo $cartel2;  
echo cuadrado($numero);
```

```
echo "<BR>";
```

```
$numero=10;  
echo $cartell1.$numero.$cartel2.cuadrado($numero);
```

```
?>
```

PHP - Modelo de objetos

```
class Persona
{
    public $nombre;
    public $apellido;

    public function __construct($n, $a = '') {
        $this->nombre = $n;
        $this->apellido = $a;
    }

    public function saludo() {
        return 'Hola, soy ' . $this->nombre .
            (($this->lastName != '') ? (' ' . $this->lastName) : '') . '.';
    }

    public static function saludoPersona($n, $a) {
        return 'Hola, soy ' . $n . ' ' . $a . '.';
    }
}

$h      = new Person('Homero', 'Simpson');
echo $h->saludo();
echo '<br />';

echo Person::saludoPersona('Carl', 'Carlson');
```