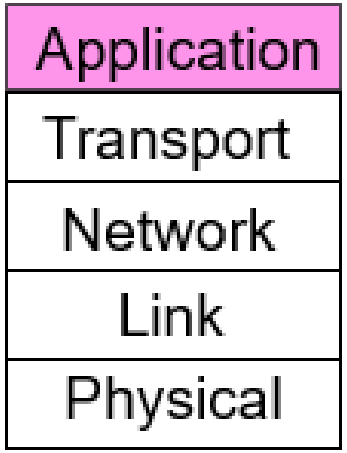
Capítulo 2

La Web – Parte 1



Páginas Web

- **Páginas web:**
  - Cada página web puede contener **vínculos** a otras páginas web ubicadas en cualquier lugar del mundo.
  - Una página web suele contener texto
  - Una página web suele referenciar varios **objetos**
    - Ejemplos de objetos son: archivo HTML, imagen JPEG, archivo de audio, etc.

Páginas Web

- Las **páginas web estáticas** son documentos en algún formato.
  - P.ej. HTML, pdf, etc.
  - Suelen escribirse en **HTML**
    - ahora se usa HTML 5

Páginas Web

- **Problema:** trabajar con páginas estáticas se torna muy ineficiente cuando la información cambia frecuentemente o cuando la información de la página varía de acuerdo a diferentes parámetros.
- Queremos evitar tener que modificar a mano las páginas estáticas a cada rato o tener que producir demasiadas páginas estáticas.
- **Solución:** usar **páginas dinámicas**:
  - Páginas HTML son generadas por medio de programas que ejecutan del lado del servidor..
  - Esos programas toman parámetros de entrada.
  - Esos parámetros de entrada suelen ser ingresados como valores de campos de formulario HTML.

Páginas Web

- **Que el servidor web tenga que construir páginas dinámicas puede ser ineficiente por los siguientes motivos:**
  1. La página nueva a generar dinámicamente en el servidor puede tener una parte importante en común con la página que ya tiene el browser;
     - Y esa parte que se repite se genera de nuevo y tiene que enviarse de nuevo por la red.
     - Esa parte repetida va a tener que ser interpretada de nuevo por el browser.

Páginas Web

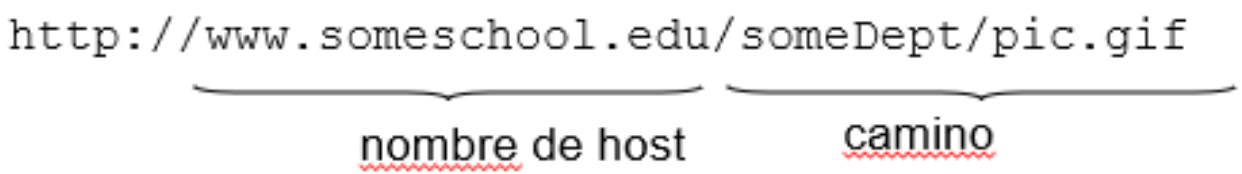
2. El cliente se queda bloqueado esperando luego de hacer un pedido HTTP al servidor web y recién puede continuar ejecutándose cuando recibe una página (estática o generada dinámicamente).
   - Estos son los llamados **pedidos sincrónicos**.
   - Si el procesamiento de un pedido del lado del servidor toma mucho tiempo, el no poder usar la aplicación web mientras tanto para otra cosa puede ser bastante desagradable para el usuario.

Páginas Web

- **Solución**: Usar **página única**.
  - Cuando se entra en la aplicación web el servidor web manda una **página única** al browser.
    - Esta contiene una **interfaz con el usuario completa** que tiene una apariencia similar a las interfaces de usuario de aplicaciones de escritorio.
  - Desde la página única se pueden hacer **pedidos de datos** al servidor web
    - El servidor web solo **obtiene los datos** (puede ser por medio de scripts), no computa páginas.
    - Luego de hacer pedido de datos la aplicación puede seguir haciendo otras tareas mientras se procesa ese pedido. A esto se lo llama **pedido asincrónico**.
  - Cuando llegan los datos se **actualiza** la interfaz del usuario de la página única en el browser.
    - Solo se cambia la parte de la UI que se necesite; lo que no cambia, no se toca; y todo esto se hace dentro del browser.

URLs

- **Problema**: Un enlace incrustado en una página web necesita una manera de nombrar una página en la web; similarmente un objeto referenciado por una página web necesita una manera de ser nombrado.
- **Solución**: Las páginas/objetos se nombran usando **URLs (Localizadores Uniformes de Recursos)**
- **Partes de un URL:**
  - ❑ Nombre del protocolo (HTTP).
  - ❑ Nombre de dominio de host que contiene la página
  - ❑ El nombre del archivo que contiene la página (camino al archivo).



URLs

- **Problema**: Con un URL como los de antes no basta para especificar la página dinámica deseada o el programa que obtiene los datos deseados.
  - Es necesario tener parámetros para la creación de páginas dinámicas o para el programa que obtiene datos (para apps de página única).
  - Además hace falta poder ingresar los parámetros en el pedido HTTP.
- **Solución 1**: el URL contiene nombre de programa y parámetros. Los parámetros son ingresados por medio de formulario HTML.
  - P. ej : `demo_get2.asp?fname=Henry&name=Ford`
  - **Este ejemplo nos dice:**
    - Los parámetros son pares: `nombre=valor`.
    - Se separa nombre de programa con '?'
    - Se separan parámetros con '&'.

URLs

- **Solución 2**: los parámetros se ingresan separados por & en un campo especial del pedido HTTP (llamado **cuerpo de la entidad**).
  - Los URL tienen un tamaño máximo.
  - Cuando los parámetros exceden ese tamaño máximo no se puede usar la solución 1;
  - Es ahí que se torna útil la solución 2.

La Web

- Un **sitio web** es un conjunto de paginas web relacionadas localizadas bajo un único nombre de dominio, y publicadas por al menos un servidor web, típicamente producida por una organización o persona.
  - Los sitios web se centran en entregar contenido, usualmente mediante páginas estáticas.
  - **Ejemplo**: un sitio que permite ver información de una empresa y sus productos, pero no va más allá.
- Una **página de inicio** (home page) de un sitio web es una página de entrada al sitio web que sirve de guía hacia las páginas que contienen la información necesaria.
  - Es la página que se carga por default cuando el navegador busca por el sitio.

La Web

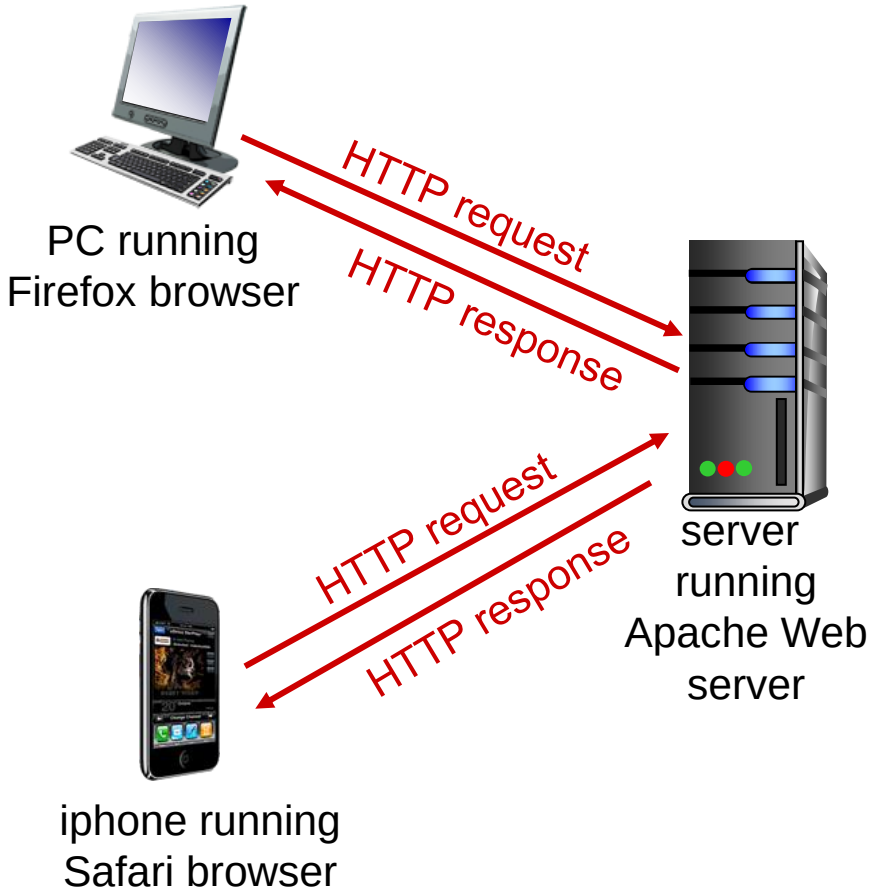
- Las **aplicaciones web** retornan y almacenan información usando scripts del lado del servidor (usando lenguajes de scripting como PHP, Perl, ASP, etc.) y del lado del browser ejecutan también scripts para diversas tareas (usando lenguajes como JavaScript).
  - Las mismas suelen soportar tareas, funciones o procesos (p.ej. para una organización.)
  - El objetivo principal de una aplicación web es que el usuario realice una o más tareas.
  - **Ejemplo**: Google docs permite hacer la tarea de editar documentos, Gmail permite la tarea de crear emails, etc.

La Web: Metas

- **Vamos a estudiar**:
  - Páginas Web
  - Navegadores (browsers)
  - Cómo hacen navegadores web para visualizar documentos.
  - Servidores Web
  - Cookies
  - Protocolo de comunicación HTTP

La Web

- Para ver las páginas web se usa un **navegador (browser)**.
- **¿Cómo funciona un navegador?**
  - El navegador permite pedir (usando protocolo HTTP) una página/objetos a un **servidor web**.
  - Una página pedida puede ser estática o dinámica o página única.
  - El servidor web retorna (usando HTTP) la página/objetos en respuesta al pedido del navegador.
  - Si el servidor retornó una página, el navegador interpreta el texto y los comandos de formateo que contiene la página y despliega la página adecuadamente formateada en pantalla



La Web

- **Problema: Para poder desplegar una página el navegador tiene que entender su formato**.
- **Solución**: Para que los navegadores entiendan las páginas web, estas se escriben en un lenguaje llamado **HTML**.

Resumen de comunicación entre browser y servidor web

Orden seguido:

1. El cliente inicia una conexión TCP (crea socket) con el servidor web, usando el puerto 80
2. El servidor web acepta la conexión TCP del cliente.
3. Mensajes HTTP (mensajes del protocolo de capa de aplicación) intercambiados entre el browser (cliente HTTP) y el servidor Web (servidor HTTP)
4. La conexión TCP se cierra.

Con HTTP:

- El servidor web no mantiene información acerca de pedidos del pasado del cliente

A su vez

protocolos que mantienen "estado" son complejos!

- El estado de la historia pasada debe ser mantenido.
- Si el servidor/cliente se cae, sus visiones del "estado" pueden ser inconsistentes, y deben ser reconciliadas.

Navegadores

- **Problema:** No todas las páginas contienen solamente HTML.
  - **Ejemplo**: una página puede tener un ícono de formato GIF, una foto de formato JPEG, y un video de formato MPEG. Las páginas HTML pueden vincular cualquiera de estos.
  - Cuando el cliente recibe una página, no necesariamente es HTML.
  - **¿Cómo sabe el navegador de que tipo de página se trata?**
- **Solución: (adoptada en la práctica)**
  - cuando un servidor regresa una página también regresa alguna información adicional acerca de ella.
    - **Tipo MIME de la página**.
    - Las páginas de tipo text/html se despliegan de manera directa.
    - **Si el tipo MIME no es de los integrados:**
      - El navegador consulta una **tabla de tipos MIME** que asocia un tipo MIME con un **visor**.

La Web: Metas

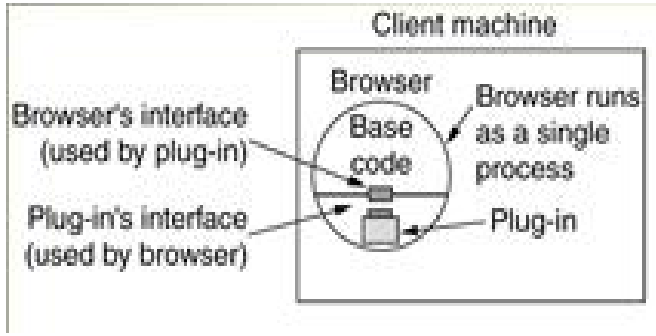
- **Vamos a estudiar**:
  - Páginas Web
  - Navegadores (browsers)
  - Cómo hacen navegadores web para visualizar documentos.
  - Servidores Web
  - Cookies
  - Protocolo de comunicación HTTP

Plug-ins y aplicaciones de ayuda

- Hay dos posibilidades para desarrollar **visores**: plug-ins y aplicaciones auxiliares.

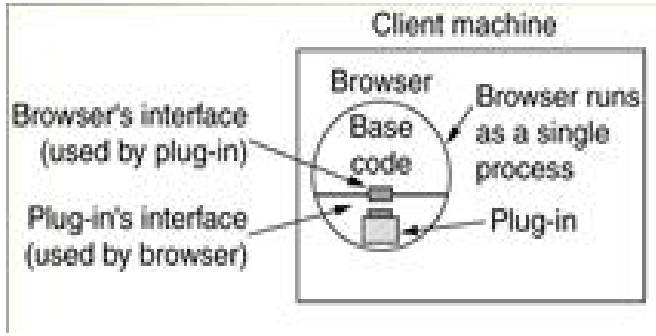
Plug-ins y aplicaciones de ayuda

- **Plug-in (conector):** es un módulo de código.
  - **Ejemplos**: Flash Player, Quick Time player
  - **El navegador obtiene los plug-in**
    - De un directorio especial de disco
  - Un navegador instala un plug-in como una **extensión de si mismo**.
  - Los plug-in se ejecutan **dentro** del proceso del navegador.
  - **Consecuencias**: los plug-in tienen acceso a la página actual y pueden modificar su apariencia..



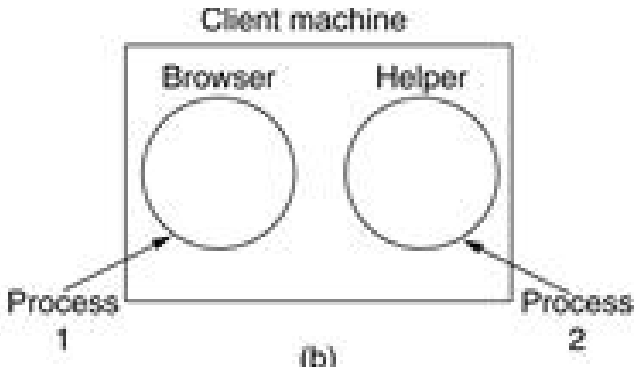
Plug-ins y aplicaciones de ayuda

- **Plug-in (cont):**
  - La **interfaz del plug-in**: conjunto de procedimientos que todos los plug-in tienen que implementar a fin de que el navegador pueda llamarlos
- La **interfaz del navegador**: conjunto de procedimientos del navegador que el plug-in puede llamar.
  - **Ejemplo**: procedimientos para asignar y liberar memoria, desplegar un mensaje en la línea de estado del navegador y consultar al navegador sobre los parámetros.
- **Una vez que el plug-in ha completado su trabajo**
  - se lo elimina de la memoria del navegador



Plug-ins y aplicaciones de ayuda

- **Aplicaciones auxiliares**.
  - ❑ Se ejecutan en un proceso separado del browser.
  - ❑ No ofrecen interfaz al navegador ni usan servicios de este.
  - ❑ Suelen aceptar el nombre de un archivo y lo abren y despliegan.
  - ❑ **Ejemplo**: application/pdf para archivos pdf.



La Web: Metas

- **Vamos a estudiar**:
  - Páginas Web
  - Navegadores (browsers)
  - Cómo hacen navegadores web para visualizar documentos.
  - Servidores Web
  - Cookies
  - Protocolo de comunicación HTTP

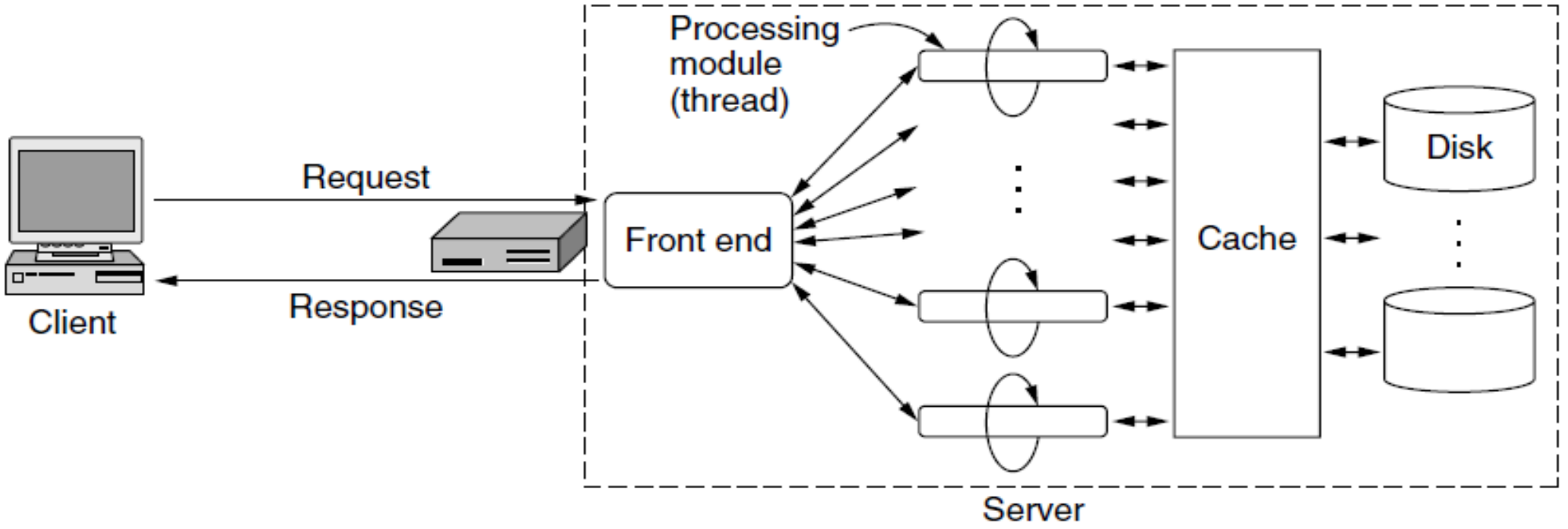
Servidores Web

- A un **servidor Web** se le proporcionará el nombre de un archivo correspondiente a una página a buscar y regresar. También se le puede proporcionar el nombre de un programa con parámetros a ejecutar.
- Un servidor web se lo puede ver como una computadora que almacena **software del servidor web** y **archivos** como documentos HTML, imágenes, archivos JavaScript, etc.
- Hoy los servidores web más populares son Apache HTTP y Nginx.

Servidores Web

- **Problema:** En el diseño anterior cada solicitud de página estática requiere un acceso al disco para obtener el archivo.
  - ❑ Esto es ineficiente porque la misma página estática puede ser pedida innumerables veces
- **Solución:** usar una **caché** de páginas estáticas en la memoria.
  - ❑ Acceder a la página en la caché es mucho más rápido que accederla en disco.
- **Problema**: hasta ahora un servidor web es un proceso con un solo hilo de ejecución.
  - ❑ **¿Cómo se podría hacer al servidor web más rápido?**

Servidores Web



- **Solución:** Arquitectura con un **módulo front end** y k **módulos de procesamiento – MP-** (hilos).
  - Todos los MP tienen acceso al caché y a uno o más discos.

Servidores Web

- **Pasos de un Servidor Web con múltiples hilos para manejar pedidos de páginas estáticas.**
  1. Cuando llega una solicitud el front end la acepta y construye un registro corto que la describe.
  2. Después entrega el registro a uno de los MP.
  3. Si se trata de pedido de página estática el MP primero verifica el caché para ver si el archivo está allí.
  4. Si el archivo está en caché actualiza el registro para incluir un apuntador al archivo.
  5. Si el archivo no está en caché el MP inicia una operación de disco.
     - Cuando el archivo llega del disco se coloca en la caché y se regresa al cliente.
  6. Mientras uno o más MP están bloqueados esperando a que termine una operación del disco, otros MP pueden estar trabajando en otras solicitudes.
  7. Conviene tener además múltiples discos, para que más de un disco pueda estar ocupado al mismo tiempo.

Servidores Web

- **Arquitecturas actuales** muestran una división entre front end y back end.
- El **front end server** se lo llama **proxy reverso**, porque retorna contenido de otros **servidores back-end** y sirve estos objetos al cliente.
- Hay soluciones que ponen en caché también páginas creadas dinámicamente.

La Web: Metas

- **Vamos a estudiar**:
  - Páginas Web
  - Navegadores (browsers)
  - Cómo hacen navegadores web para visualizar documentos.
  - Servidores Web
  - Cookies
  - Protocolo de comunicación HTTP

Cookies

- **Propósito**: Comprender cómo se organiza y comunica **información de estado de sesión** de una aplicación web.
- **Situación:** Luego de retornar una página web el servidor olvida que ha visto alguna vez a ese cliente particular.
  - **Ejemplo**: si un usuario se autenticó para usar un sitio web
    - **¿cómo recuerda el servidor web que es un usuario autenticado?**
  - **Ejemplo**: En comercio electrónico, si un usuario navega y coloca artículos en un carrito de compras de vez en cuando;
    - **¿de qué manera el servidor mantiene un registro del contenido del carrito?**

Cookies

- **Problema: ¿Cómo hacer para que el servidor y el cliente sepan del estado de la sesión?**
- **Solución**: en los pedidos y respuestas HTTP se envía información de estado de sesión.
  - Se usan para esto las llamadas **cookies**
  - **¿Qué es un cookie y qué tamaño tiene?**
    - Una **cookie** es un pequeño archivo o cadena (de a lo sumo 4 KB).
    - El **contenido de una cookie** toma la forma nombre = valor.
    - **Ejemplo:** Carrito = 1-5011; 1-7013; 2-1372

Cookies

- **Estructura de una cookie:** además de su contenido hay otros campos en una cookie.
- **Campos** de una cookie.
  1. **dominio** (nombre del dominio de destino del cookie)
     - Cada dominio puede almacenar hasta 20 cookies por cliente.
  2. **ruta** en la estructura del directorio del servidor.
     - Identifica qué partes del árbol de archivos del servidor podrían usar el cookie.
  3. El **campo contenido** toma la forma nombre = valor.
  4. El campo **expira**.
     - **Si este campo está ausente** El navegador descarta el cookie cuando sale.
     - **Si se proporciona una hora y una fecha:** Se mantiene la cookie hasta que expira ese horario.
  5. El campo **seguro**.
     - Se usa para indicar que el navegador solo puede retornar la cookie a un servidor usando un transporte seguro (p.ej: SSL, TLS).
     - Esto se usa para aplicaciones seguras (p.ej. comercio electrónico, actividades bancarias, etc.)

Cookies

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	Yes
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	No
aportal.com	/	Prefs=Stk:SUNW+ORCL;Spt:Jets	31-12-10 23:59	No
sneaky.com	/	UserID=3627239101	31-12-12 23:59	No

Algunos ejemplos de cookies

- **Eliminación de una cookie del disco duro del cliente.**
- El servidor simplemente la envía nuevamente, pero con una fecha caducada.

Cookies

- **Problema**: Con solo recibir cookies no alcanza, es necesario almacenarlas.
- **Solución:** usar **directorio de cookies:**
  - para que el navegador pueda almacenar cookies en el disco duro de la máquina del cliente.
- **Problema:** no se dice cómo se hace del lado del servidor con las cookies recibidas.
- **Solución:** Se puede guardar la información en una base de datos.

Cookies

- **Comunicación de las cookies al cliente:**
- Cuando un cliente solicita una página web, el servidor puede proporcionar una cookie junto con la página solicitada.

Cookies

- **Comunicación de las cookies al servidor web:**
  - Antes que un navegador solicite una página a un sitio web, verifica su directorio de cookies para ver si el dominio al que está solicitando la página ya colocó alguna cookie.
  - De ser así, todas las cookies para ese dominio se incluyen en el mensaje de la solicitud.
  - Cuando el servidor web las obtiene, puede interpretarlas de la forma que desee.

La Web: Metas

- **Vamos a estudiar**:
  - Páginas Web
  - Navegadores (browsers)
  - Cómo hacen navegadores web para visualizar documentos.
  - Servidores Web
  - Cookies
  - Protocolo de comunicación HTTP

Necesidades para un Protocolo para la Web

- **Cosas que se necesita que soporte un protocolo para la web:**
  - Pedido de páginas, de objetos, o de ejecución de programas que generan páginas.
  - Manejo del estado de sesión
  - Poder mantener el sistema de archivos del servidor web
  - Recepción de páginas por un browser
  - Seguridad (encriptación de mensajes)
  - Feedback adecuado cuando no se puede responder los pedidos.
  - Comunicación confiable

HTTP

- **HTTP** = *Hyper Text Transfer Protocol*
  - HTTP transfiere páginas de servidores web a navegadores y manda pedidos de navegadores a servidores web.
  - **Tipos de mensajes soportados por HTTP**
    - HTTP-Request (de Navegador a servidor web)
    - HTTP-Response (de servidor web a Navegador Web)

Conexiones HTTP

HTTP no persistente

- A lo más un objeto se manda por conexión TCP.
  - Luego se cierra la conexión.
- Descargar múltiples objetos requiere de muchas conexiones.
- En **HTTP 1.0** se establece una conexión TCP por cada solicitud y se la libera al recibir la respuesta.
- **Problema**: para descargar una página estática puede ser necesario establecer varias conexiones (una por cada objeto referenciado por la página).
  - Esto es ineficiente.

Solución: *HTTP persistente*

- Múltiples objetos pueden ser enviados a través de una única conexión TCP entre el cliente y el servidor.
- Bajo una misma conexión TCP los pedidos son procesados en orden y los resultados se mandan en orden.
- Soportado por **HTTP 1.1**

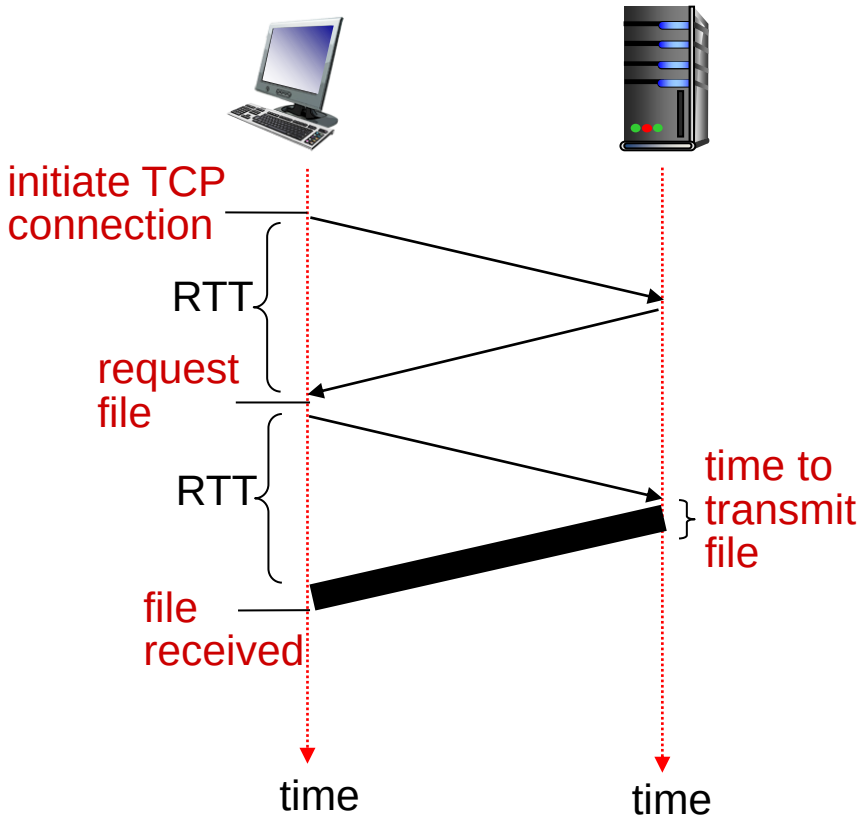
Conexiones HTTP

RTT (definición): tiempo necesario para que un paquete pequeño viaje del cliente al servidor y de regreso.

Tiempo de respuesta de HTTP no persistente para recibir un archivo:

- Un RTT para iniciar la conexión TCP
- Un RTT para el pedido HTTP y el regreso de los primeros bytes de la respuesta HTTP
- Tiempo de transmisión del archivo.
- Tiempo de respuesta de HTTP no persistente =

  2RTT+ tiempo de transmisión del archivo.



Conexiones HTTP

- Con HTTP 1.1 se pueden hacer varios pedidos bajo una misma conexión, pero van a ser procesados en orden y los resultados van a ser enviados en orden.
- **Problema**: con HTTP 1.1 no se envían documentos al browser cuando se sabe que se los va a necesitar (se espera primero que se los pida); además no permite recibir varios pedidos, priorizarlos y enviar las respuestas en el orden más conveniente.
  - Tener en cuenta estas cosas aumentaría el desempeño.

Conexiones HTTP

- **Por ejemplo**: si un cliente pide una página web y el servidor ve que usa una hoja de estilo y un archivo JavaScript, el servidor podría enviar la hoja de estilo y el archivo JavaScript antes que sean requeridos. Esto elimina algunas demoras.
- **Por ejemplo**: se piden dos imágenes primero una grande y luego una chica se puede responder primero con la chica y luego con la grande así el browser muestra en pantalla primero la imagen chica.

Conexiones HTTP

- **Solución**: **protocolo HTTP 2.0**
  - Por medio de mecanismo server push empuja archivos que sabe que van a necesitarse pero que el cliente puede no saber inicialmente.
  - Las respuestas a los pedidos pueden volver en cualquier orden.
  - HTTP 2.0 comprime los encabezados y los envía en binario para reducir uso de ancho de banda.
  - Cada respuesta lleva un identificador de su pedido.
- Hoy en día tanto Apache HTTP como Nginx soportan HTTP 2.0.

Pedidos HTTP

Informaciones que debería tener un mensaje de pedido:

- En caso que se quiera recibir una página:
  - El **URL** de un documento.
  - La **especificación de programa que genera página web**
- El **tipo de acción** que se quiere hacer en el sistema de archivos del servidor web (meter páginas, borrar páginas, etc.)
- Mandar **información sobre la máquina/software del cliente** para que servidor web pueda retornar páginas adecuadas al cliente.
- Mandar **información de estado de sesión** para que el servidor se entere.
- **Restricciones sobre el tipo de páginas** que el cliente puede aceptar.

Pedidos HTTP

- **Problema: ¿Cómo indicar el tipo de acción que se quiere hacer?**
  - Pedido de página o
  - acción en el sistema de archivos del servidor web
- **Solución**: Usar un campo con el tipo de la acción requerida.
  - En la primera palabra de la primera línea se pone el nombre del **método** solicitado.

HTTP: métodos

HTTP tiene varios métodos de pedido.

Fetch a page →

Used to send input data to a server program →

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a page

Pedidos HTTP

Para subir el input de un formulario hay dos opciones:

Método Post:

- La página web a menudo contiene input de formulario.
- El input es subido al servidor en un campo llamado el **cuerpo de la entidad**.

Método que usa URL:

- Usa el **Método Get.**
- El input es subido en el campo URL de la línea del pedido:

```
www.somesite.com/animalsearch?monkeys&banana
```

Pedidos HTTP

- **Métodos para actualizar las páginas del servidor web**
- **Método PUT**: Sube un archivo en el campo **cuerpo de la entidad** en el camino especificado por el campo URL.
- **Método DELETE**: Borra el archivo especificado en el campo URL.

Pedidos HTTP

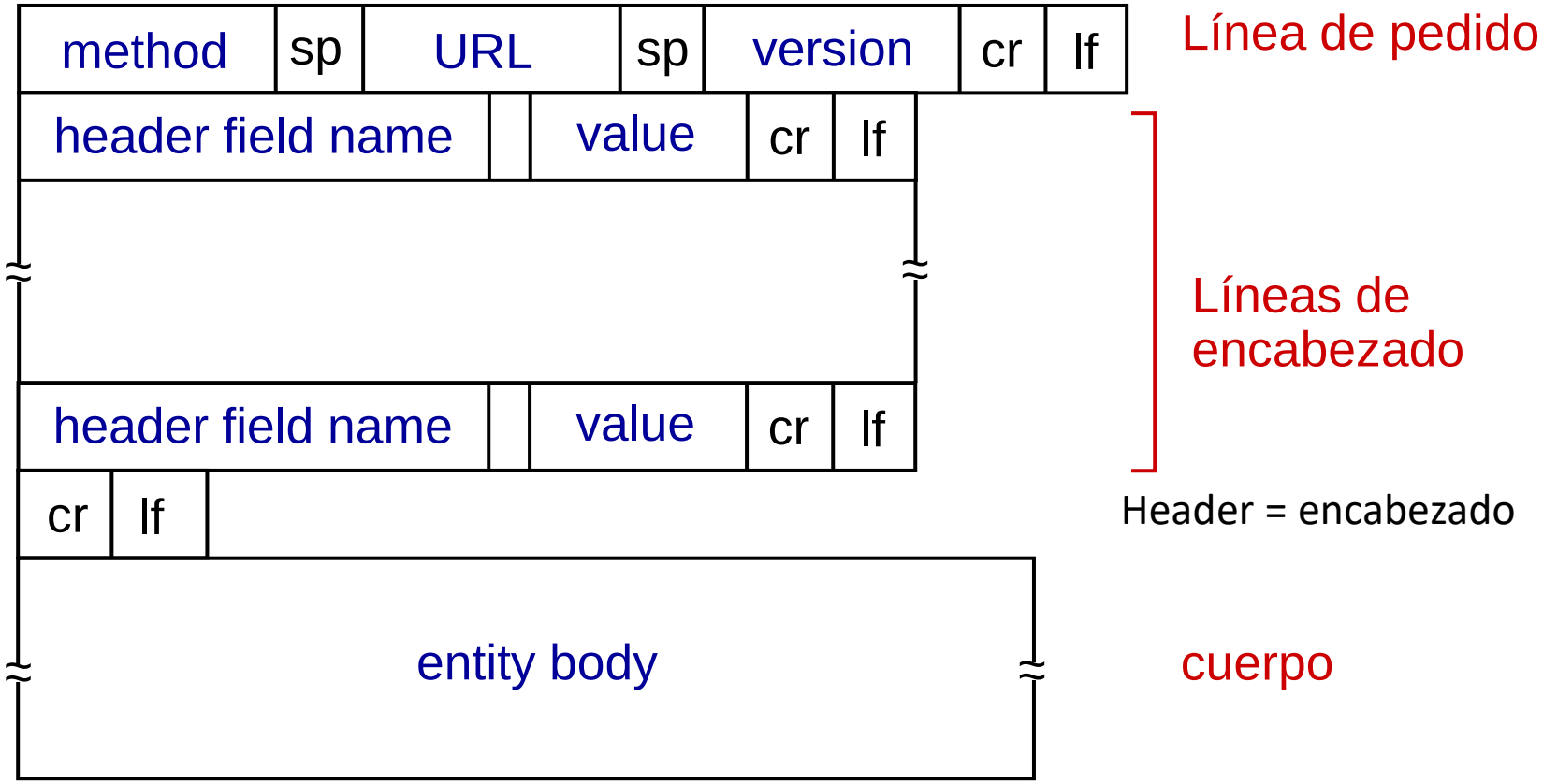
- El **método HEAD** simplemente solicita el encabezado de la respuesta del servidor web, sin la página o datos de la respuesta – o sea, feedback sobre el resultado del pedido, tipo de contenido, etc.
- El **método OPTIONS:** permite que el cliente consulte al servidor por una página y obtenga los métodos y encabezados http que pueden ser usados con esa página.
  - También puede usarse para saber los métodos http soportados por el servidor web en general.

Pedidos HTTP

- **Problema: Se necesita enviar información diversa:**
  - P.ej. sobre la máquina/software del cliente, sobre estado de sesión, restricciones de tipo de páginas a recibir, etc.
- **Solución:** indicar el tipo de información de que se trata y luego la información en sí.
  - Se usan **encabezados de pedido** que son pares:

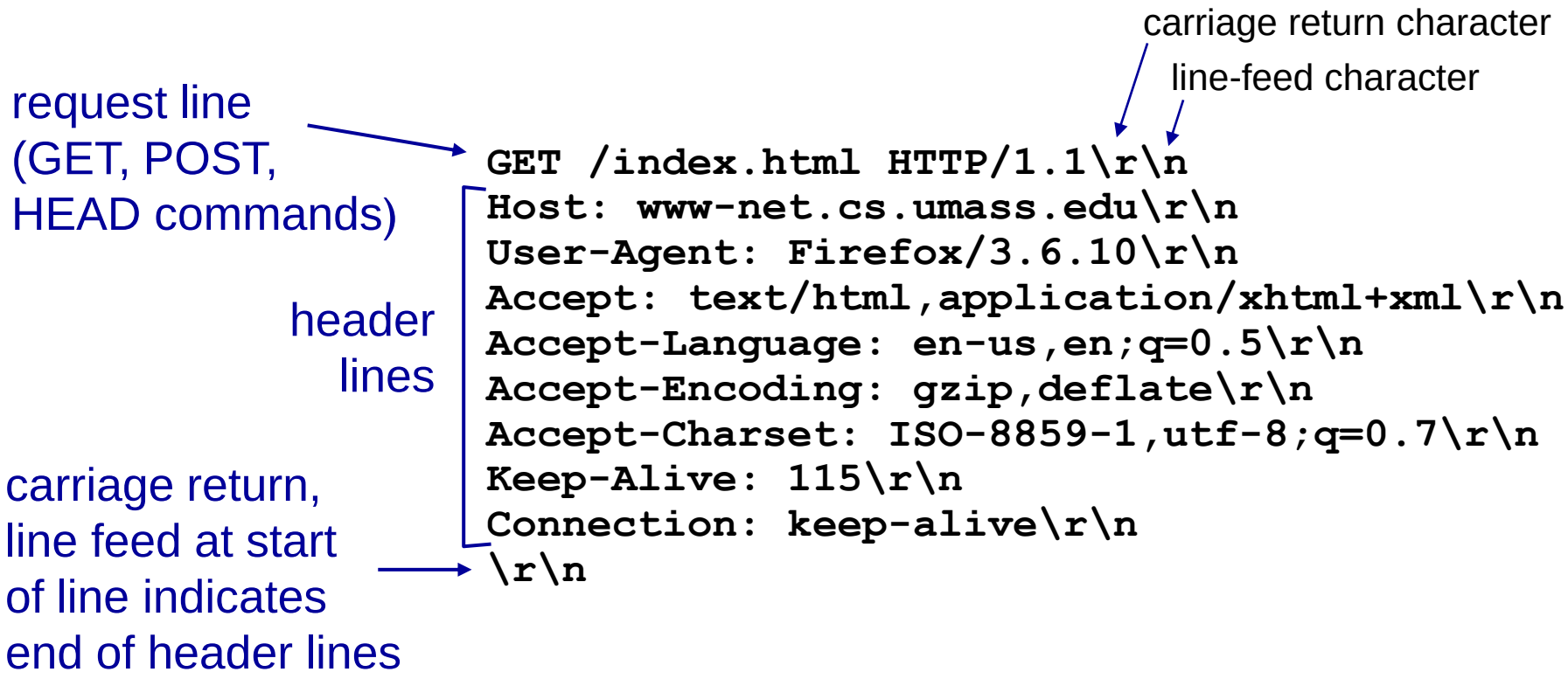
    nombre de encabezado y valor

Pedidos HTTP (Formato)



- A la línea de solicitud le pueden seguir líneas adicionales llamadas **encabezados de solicitud**.

Pedidos HTTP



Respuestas HTTP

Informaciones que debería tener un mensaje de respuesta:

- Feedback adecuado sobre el pedido realizado
  - P. ej: cuando no se puede cumplir con el pedido.
- Página o documento solicitado.
- En ese caso información sobre el tipo de documento enviado.
  - P.ej. el tipo MIME del documento, cuando fue modificado por última vez el documento, etc.
- Información de estado de sesión para mantener actualizado al cliente.

Respuestas HTTP

- **Problema: ¿Cómo especificar en la respuesta HTTP el feedback sobre el pedido recibido?**
- **Solución**: usar un código y un mensaje.
  - **Línea de estado**: contiene un **código de estado** de 3 dígitos que indica si la solicitud fue atendida y sino por qué.

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

Respuestas HTTP

- **Problema: ¿Cómo mandar además de la página solicitada/datos solicitados información adicional en una respuesta http?**
  - P.ej. sobre el tipo del documento enviado, sobre estado de sesión, etc.
- **Solución:** indicar el tipo de información de que se trata y luego la información en sí.
  - Se usan **encabezados de respuesta** que son pares: nombre de encabezado y valor

Respuestas HTTP

- **Partes de una respuesta HTTP**
  1. **Línea de estado**.
  2. (opcional) **encabezados de respuesta**.
  3. Luego vienen el cuerpo de la respuesta:
     - p.ej. página estática usando archivo en formato HTML.
     - P.ej. datos pedidos usando documento en formato XML.

Respuestas HTTP

Ejemplo de respuesta HTTP

The start of the output of *www.ietf.org/rfc.html*.

```
Trying 4.17.168.6...
Connected to www.ietf.org.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: avoid browser bug

<html>
<head>
<title>IETF RFC Page</title>

<script language="javascript">
function url() {
 var x = document.form1.number.value
 if (x.length == 1) {x = "000" + x }
 if (x.length == 2) {x = "00" + x }
 if (x.length == 3) {x = "0" + x }
 document.form1.action = "/rfc/rfc" + x + ".txt"
 document.form1.submit
}
</script>

</head>
```

Encabezados HTTP

- Los mensajes HTTP suelen tener **encabezados**.
- Los encabezados pueden ser de pedido, de respuesta, o de ambos tipos.
- Los encabezados se usan para **proveer información** a ser procesada por el receptor del mensaje.
  - Información sobre la máquina del cliente, sobre la máquina del servidor, etc.
  - Información sobre la página retornada por el servidor.
- También se usan para **fijar restricciones** que deben cumplir mensajes futuros.
  - Tipos de páginas que pueden manejar los clientes, conjuntos de caracteres aceptables, lenguajes naturales aceptables, etc.
- También se usan para proveer información sobre **eventos** importantes:
  - Cuando fue modificada la página por última vez, cuando fue enviado el mensaje, cuando deja de ser válida la página, etc.
- Además se usan para proveer datos de **estado de la sesión**.

Encabezados HTTP

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Algunos Encabezados HTTP