

# Capítulo 2: Capa Aplicación - II

## ELO322: Redes de Computadores Agustín J. González

Este material está basado en:

- Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet 3rd* edition. Jim Kurose, Keith Ross Addison-Wesley, 2004.

# Capítulo 2: Capa Aplicación

- 2.1 Principios de la aplicaciones de red
- **2.2 Web y HTTP**
- 2.3 FTP
- 2.4 Correo Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Compartición de archivos
- 2.7 Programación de Socket con TCP
- 2.8 Programación de socket con UDP
- 2.9 Construcción de un servidor WEB

# Web y HTTP

- Una página Web está compuesta de objetos
- En este contexto objetos pueden ser archivos HTML, imágenes (JPEG, GIF,...), Java applets, archivos de audio, archivos de vídeo,...
- Páginas Web consisten generalmente de un archivo HTML base el cual incluye referencias a objetos.
- Cada objeto es direccionable por un Universal Resource Locator (URL)
- Ejemplo URL:

`http://www.e1o.utfsm.cl/imgmenu/header.jpg`

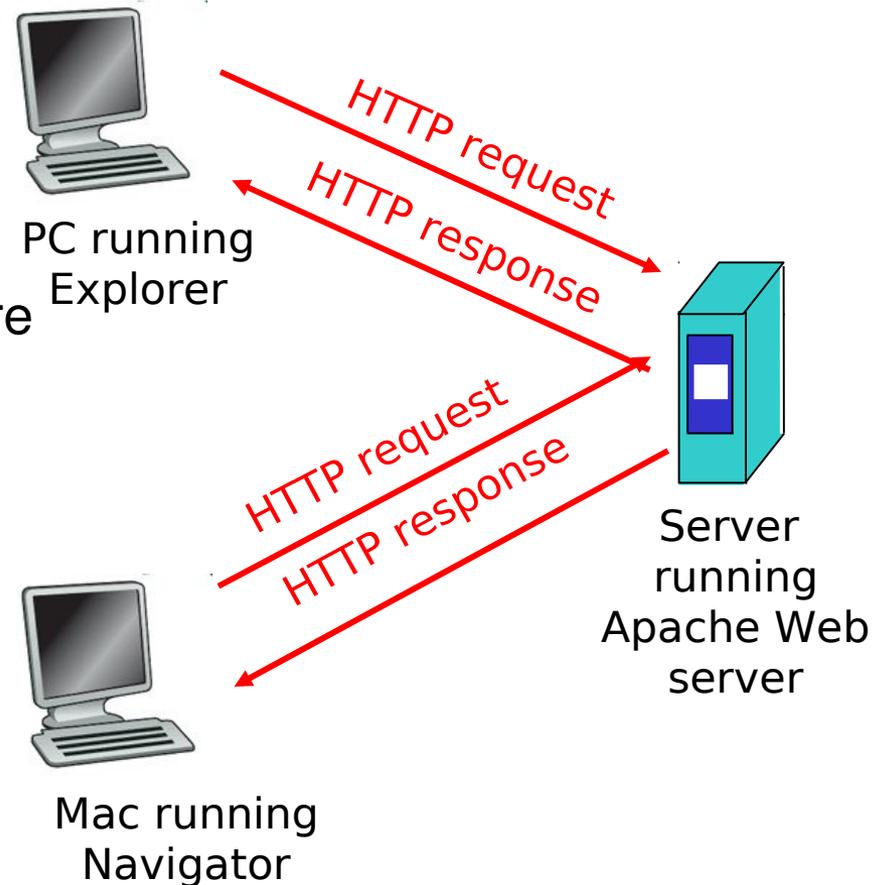
Nombre de la máquina y puerto

Nombre de camino (path name)

# HTTP Generalidades

## HTTP: hypertext transfer protocol

- Protocolo de la capa aplicación usado por la Web
- Modelo cliente/servidor
  - *cliente*: browser primero requiere y luego recibe y “despliega” objetos Web
  - *servidor*: Servidor Web envía objetos en respuesta a requerimientos
- HTTP 1.0: RFC 1945 (1996)
- HTTP 1.1: RFC 2068 (1997)
- HTTP 1.1 Mejorado RFC 2616 (1999)



# HTTP generalidades (cont.)

## Usa TCP:

- 1) Cliente inicia conexión TCP (crea socket) al servidor, puerto 80 (puede ser otro!)
- 2) Servidor acepta conexión TCP del cliente
- 3) Mensajes HTTP (mensajes del protocolo de capa aplicación) son intercambiados entre browser (cliente HTTP) y servidor Web (servidor HTTP)
- 4) Se cierra la conexión TCP

## HTTP no guarda “estado”

- El servidor no mantiene información sobre los requerimientos del clientes

### Protocolos que mantienen “estado” son complejos!

- Historia pasada (estado) debe ser mantenida
- Si servidor o cliente se cae, las vistas del estado pueden ser inconsistentes, y deben ser sincronizadas



Qué significa que una aplicación siga el modelo cliente servidor? Mencione una aplicación que siga este modelo.



Significa que la aplicación posee dos tipos de procesos. Un proceso servidor que debe ser ejecutado en un puerto específico y queda a la espera de peticiones, y procesos clientes que deben saber dónde está el servidor y el puerto donde corre para solicitar servicios. Los clientes pueden correr desde cualquier parte y no requieren estar funcionando permanentemente. Ejemplos son: la web, ssh, ftp, el correo electrónico.

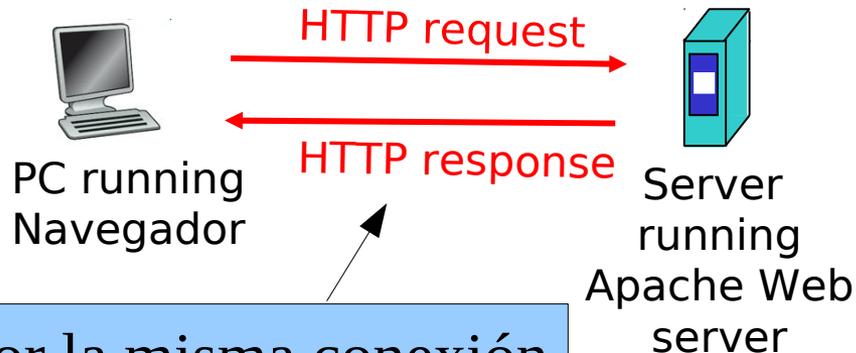
# Conexiones HTTP

## HTTP No-persistente

- A lo más un objeto es enviado por una conexión TCP.  
Es como hacer una llamada por objeto.
- HTTP/1.0 usa HTTP no-persistente

## HTTP Persistente

- Múltiples objetos pueden ser enviados por una única conexión TCP entre el cliente y servidor.
- HTTP/1.1 usa conexiones persistentes en su modo por defecto



Ambos por la misma conexión

# HTTP no-persistente

Supongamos que el usuario ingresa URL

`www.someSchool.edu/someDepartment/home/index`

(contiene texto, y referencias a 10 imágenes jpeg )

1a. Cliente HTTP inicia una conexión TCP al servidor HTTP (proceso) en `www.someSchool.edu` en puerto 80

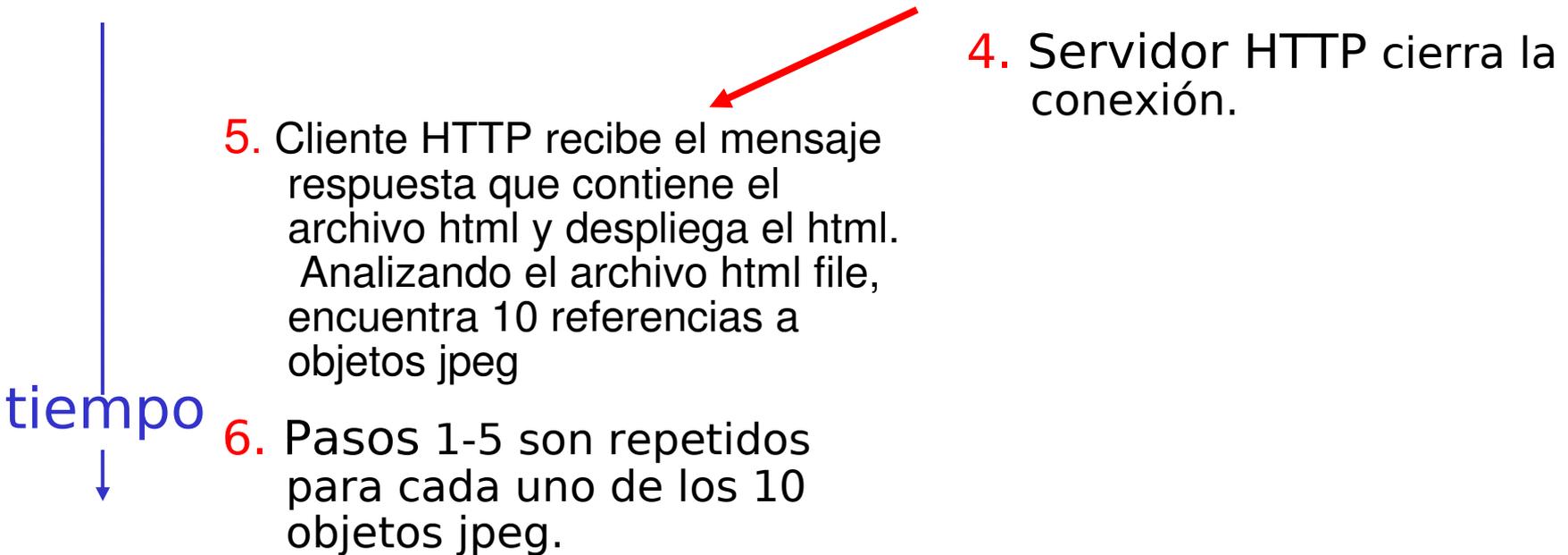
1b. Servidor HTTP en host `www.someSchool.edu` esperando por conexiones TCP en puerto 80 “acepta” conexión, notifica al cliente

2. Cliente HTTP envía *mensaje de requerimiento* (conteniendo el URL) por el socket de la conexión TCP. El mensaje indica que el cliente quiere el objeto `someDepartment/home/index`

3. El servidor HTTP recibe el mensaje de requerimiento, forma el *mensaje de respuesta* que contiene el objeto requerido y envía el mensaje por su socket.

tiempo  
↓

# HTTP no-persistente (cont.)



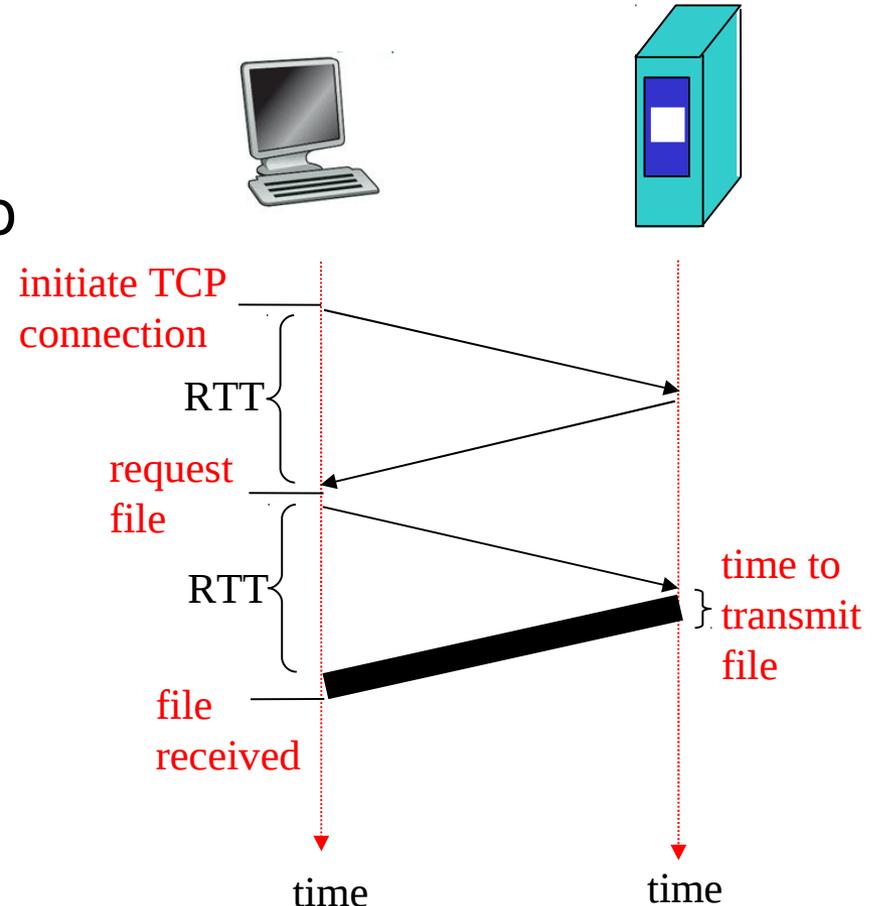
# Modelo para tiempo de Respuesta

**Definición de RTT(round-trip time):** tiempo ocupado en enviar un paquete pequeño desde el cliente al servidor y su regreso.

## Tiempo de respuesta:

- Un RTT para iniciar la conexión
- Un RTT por requerimiento HTTP y primeros bytes de la respuesta
- Tiempo de transmisión del archivo

**total =  $2RTT$  + tiempo de transmisión**



# HTTP Persistente

## Problemas de HTTP no-persistente:

- requiere 2 RTTs por objeto
- el navegador abre conexiones paralelas generalmente para traer objetos referenciados. => OS debe trabajar y dedicar recursos para cada conexión TCP

## HTTP Persistente

- servidor deja las conexiones abiertas después de enviar la respuesta
- mensajes HTTP siguientes entre los mismos cliente/servidor son enviados por la conexión

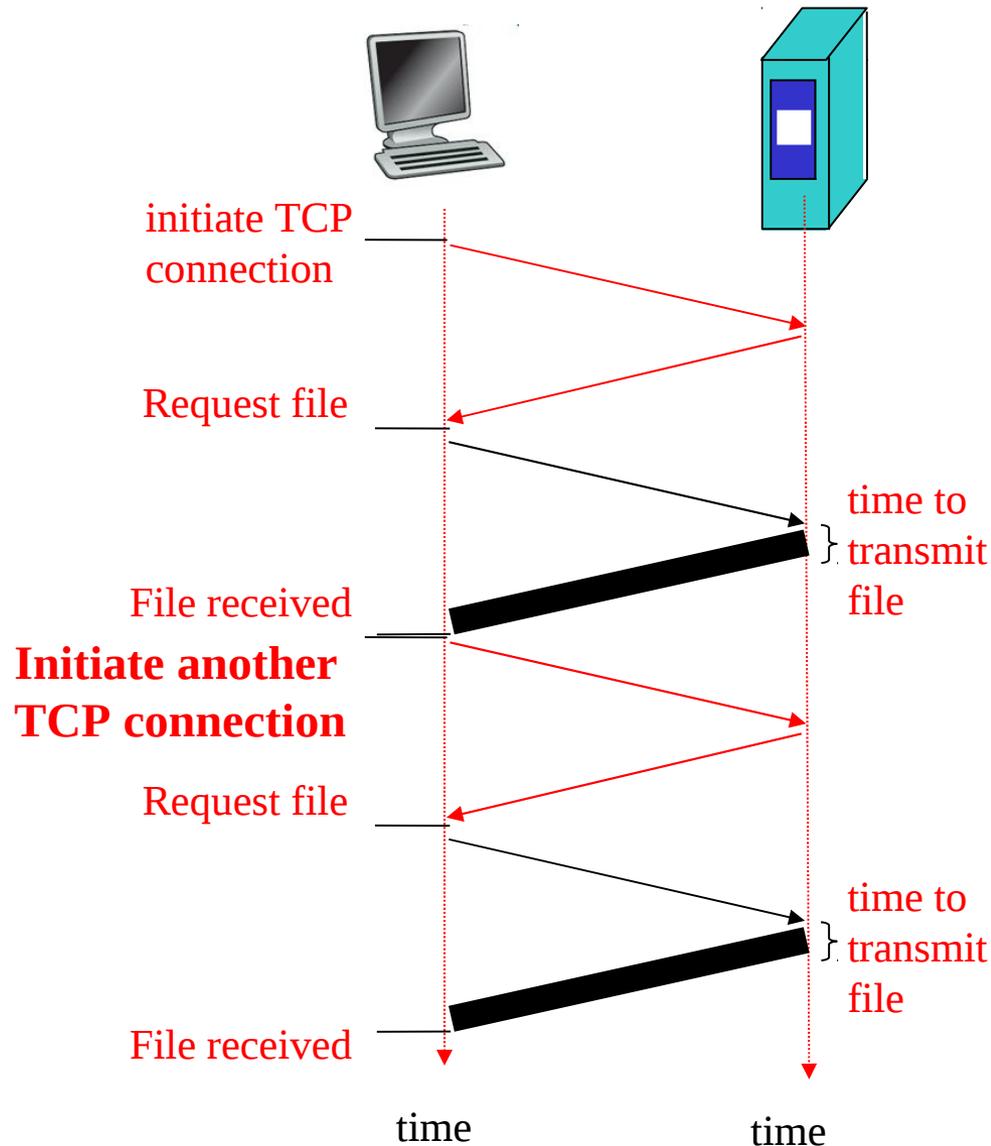
## Persistencia sin pipelining:

- cliente envía nuevo requerimiento sólo cuando el previo ha sido recibido
- un RTT por cada objeto referenciado

## Persistencia con pipelining:

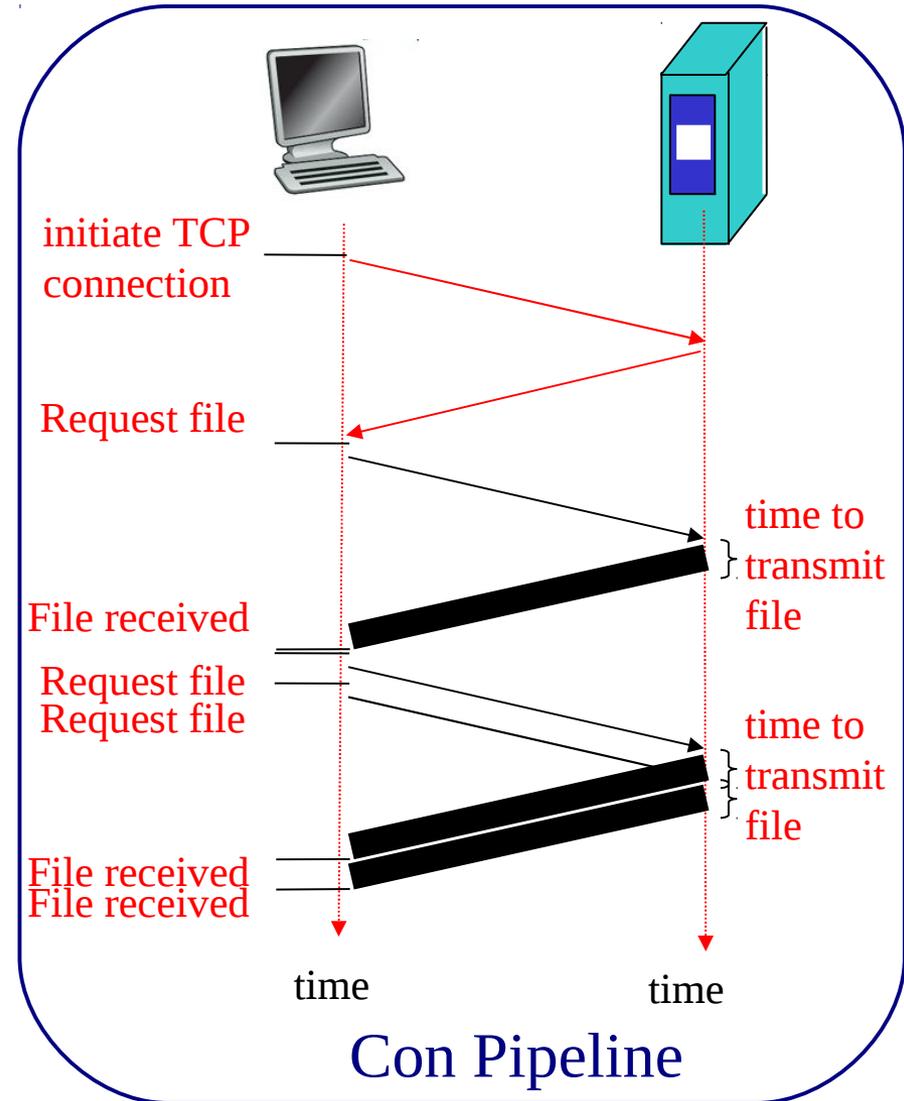
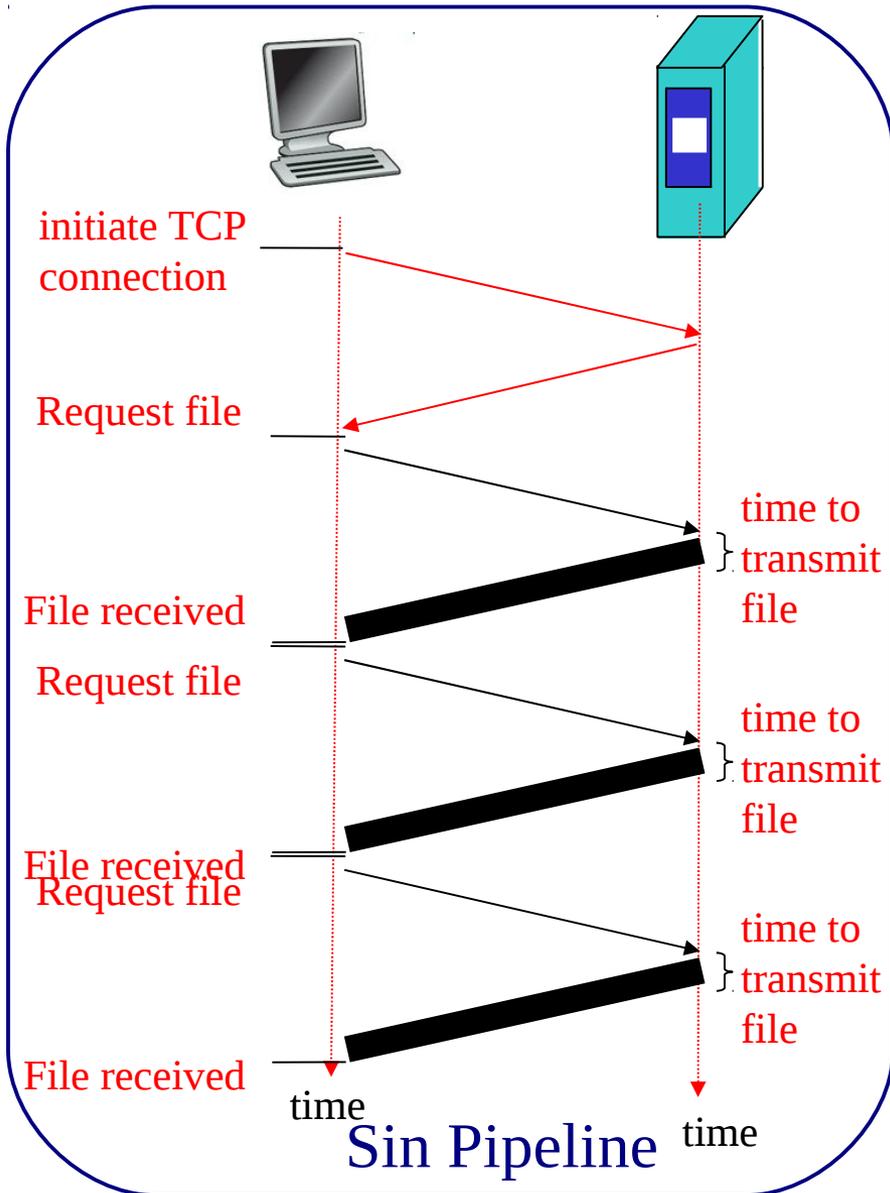
- default en HTTP/1.1
- cliente envía requerimientos tan pronto éste encuentra un objeto referenciado
- tan poco como un RTT para todas las referencias

# HTTP No persistente



**En todos estos diagramas suponemos que los objetos caben en un segmento (=paquete) TCP.**

# HTTP persistente





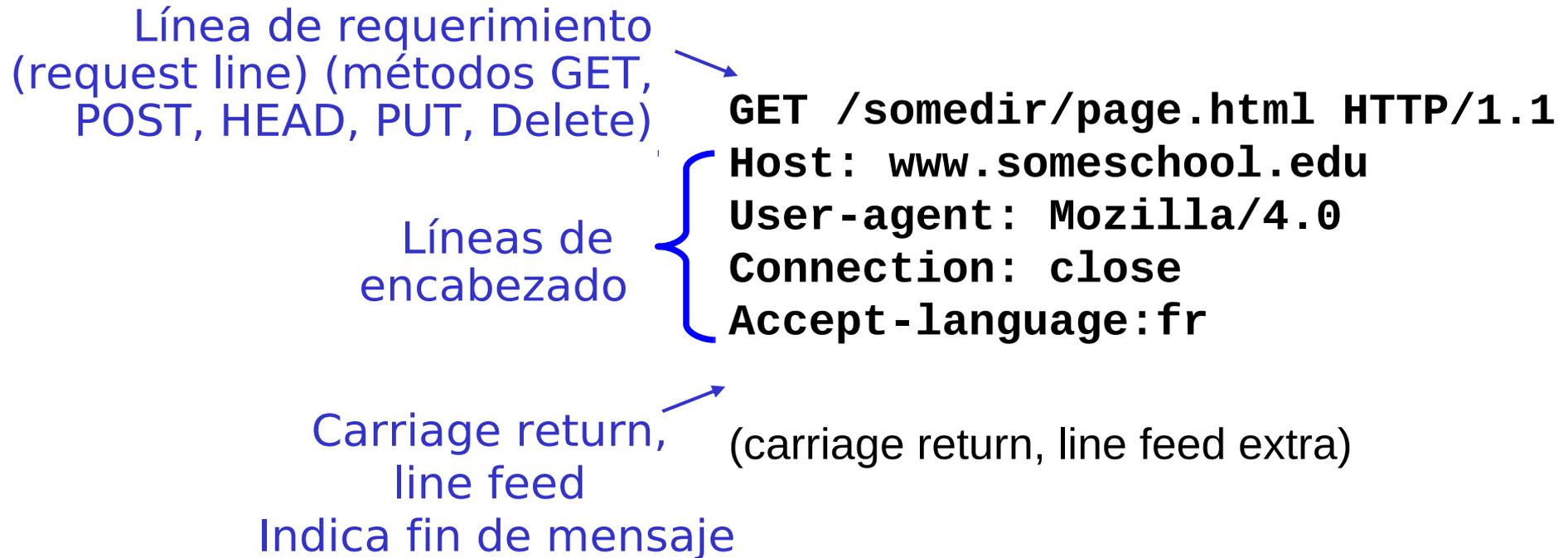
¿Cuál es la diferencia entre HTTP persistente con pipeline y sin pipeline que explica el menor tiempo para bajar una página WEB?



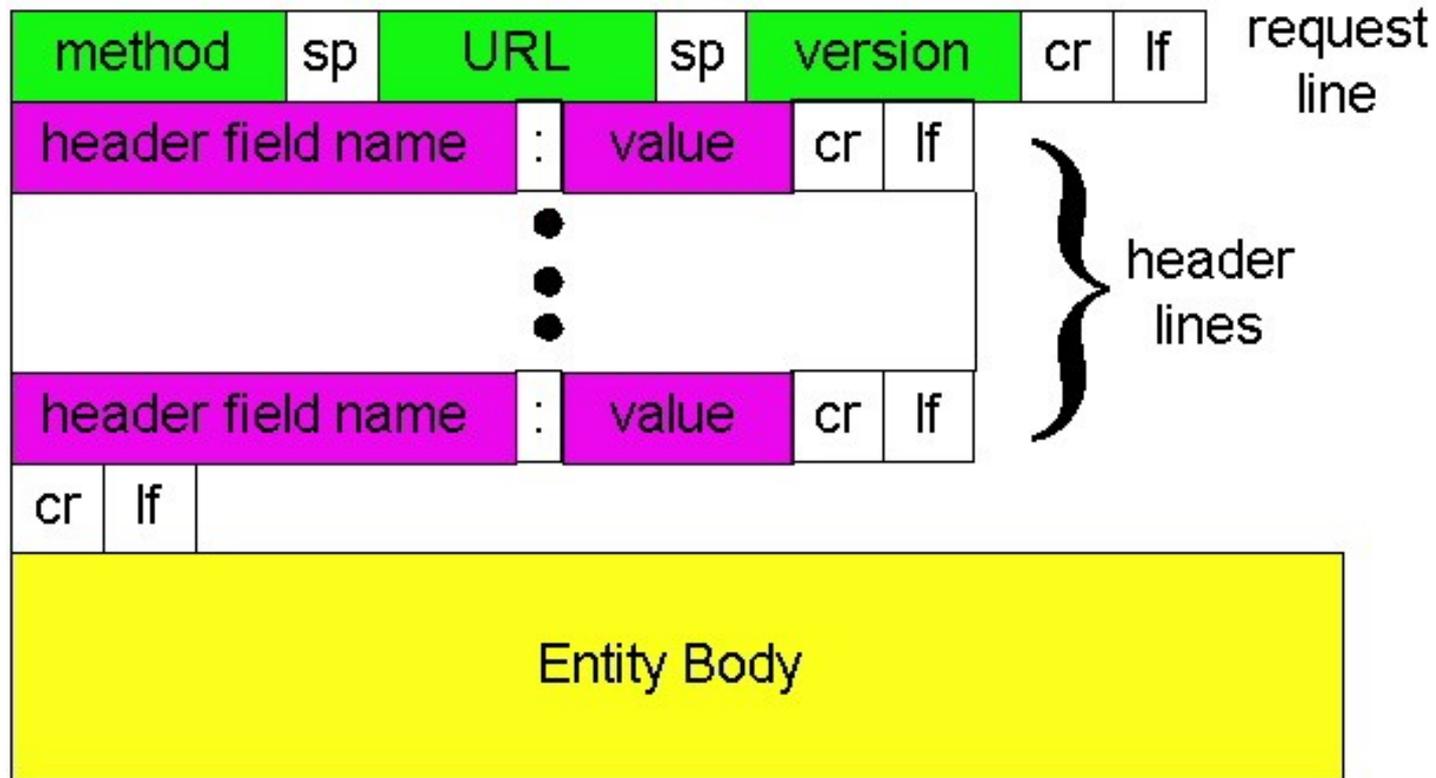
El menor tiempo lo presenta HTTP persistente con pipeline debido a que éste envía todos los requerimientos de objetos tan pronto los identifica. Así con pipeline se logra simultaneidad o paralelismo entre requerimientos que viajan al servidor y repuestas que van hacia el browser. Sin pipeline no hay paralelismo entre requerimientos y respuestas.

# Mensaje HTTP de requerimiento

- Dos tipos de mensajes HTTP: *requerimiento, respuesta*
- **Mensaje de requerimiento HTTP:**
  - ASCII (es decir, formato legible)



# Mensaje HTTP de requerimiento: formato general



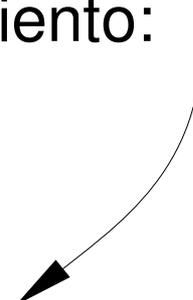
# Subiendo input de formulario

## Vía Método Post:

- Páginas Webs usualmente incluyen entradas de formularios
- Los datos son subidos al servidor en el cuerpo del mensaje

## Vía Método GET:

- Entrada es subida en campos URL de la línea de requerimiento:



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

# Tipos de Métodos

## HTTP/1.0

- GET
- POST
- HEAD
  - Pide al servidor que deje el objeto requerido afuera de la respuesta. Respuesta incluye sólo el encabezado.

## HTTP/1.1

- GET, POST, HEAD
- PUT
  - Sube archivos en cuerpo del requerimiento en localización indicada por el campo URL
- DELETE
  - Borra archivo especificado en el campo URL

# Mensajes HTTP de respuesta

Línea de estatus  
(código de estatus  
del protocolo  
Frase de estatus)

Líneas de  
encabezado

data, e.g.,  
archivo  
HTML solicitado

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

# Códigos HTTP de respuesta

En primera línea de respuesta del servidor-> cliente.  
Algunos códigos de muestra:

## **200 OK**

- ▣ request exitoso, objeto requerido es incluido luego en mensaje

## **301 Moved Permanently**

- ▣ Se movió el objeto requerido, nueva ubicación es especificada luego en el mensaje (Location:)

## **400 Bad Request**

- ▣ Requerimiento no entendido por el servidor

## **404 Not Found**

- ▣ Documento no encontrado en servidor

## **505 HTTP Version Not Supported**

# Probando HTTP (lado cliente)

## 1. Telnet a tu servidor favorito:

**telnet profesores.elo.utfsm.cl 80**

Telnet abre una conexión TCP al puerto 80 (puerto servidor HTTP por omisión) en mateo.elo.utfsm.cl. Cualquier cosa ingresada es enviada a puerto 80 de mateo

## 2. Escribir un requerimiento GET HTTP:

**GET /~agv/elo322/1s09/prueba.html HTTP/1.1**  
**Host: profesores.elo.utfsm.cl**

**NOTA: Campo Host es obligatorio en encabezado, requerido por proxy.**

Tipeando esto (doble carriage return), enviamos un GET request mínimo (pero completo) al servidor HTTP

## 3. Observar el mensaje de respuesta enviado por el servidor HTTP!

Hacer algo similar con navegador y wireshark

 Para acceder a un servidor web nuestro browser o navegador siempre se conecta a un mismo puerto de la máquina indicada en el URL. Si usted está de acuerdo, indicar el número de puerto. Si usted no está de acuerdo, justifique. 

No estoy de acuerdo. En general el puerto puede cambiar. Si se omite el puerto en el URL, siempre se conectará al puerto 80 (por omisión). Si al ingresar el URL se especifica un puerto específico, el browser se conectará a ése para buscar el servicio web.



Alguien dice: “Para acceder a un servidor web nuestro browser o navegador siempre se conecta desde un mismo puerto de la máquina local.” ¿Está usted de acuerdo? Justifique.



No. El servidor al cual accedemos debe usar el mismo puerto para todos los clientes (mismo número), pero los clientes pueden conectarse desde puertos diferentes (distintos números).

# Cómo conocer estado usuario-servidor: cookies

Muchos sitios Web importantes usan cookies

- Las cookies fueron implementadas para permitir personalizar la información Web.
- Cookies es información generada por un Web server y almacenada en el computador del usuario para **acceso** futuro.
- Las cookies son transportadas entre el computador del usuario y el servidor.
- Por ejemplo, cookies son usadas para almacenar ítems en un carro de compra mientras recorres un mall virtual.

# Estado usuario-servidor: cookies

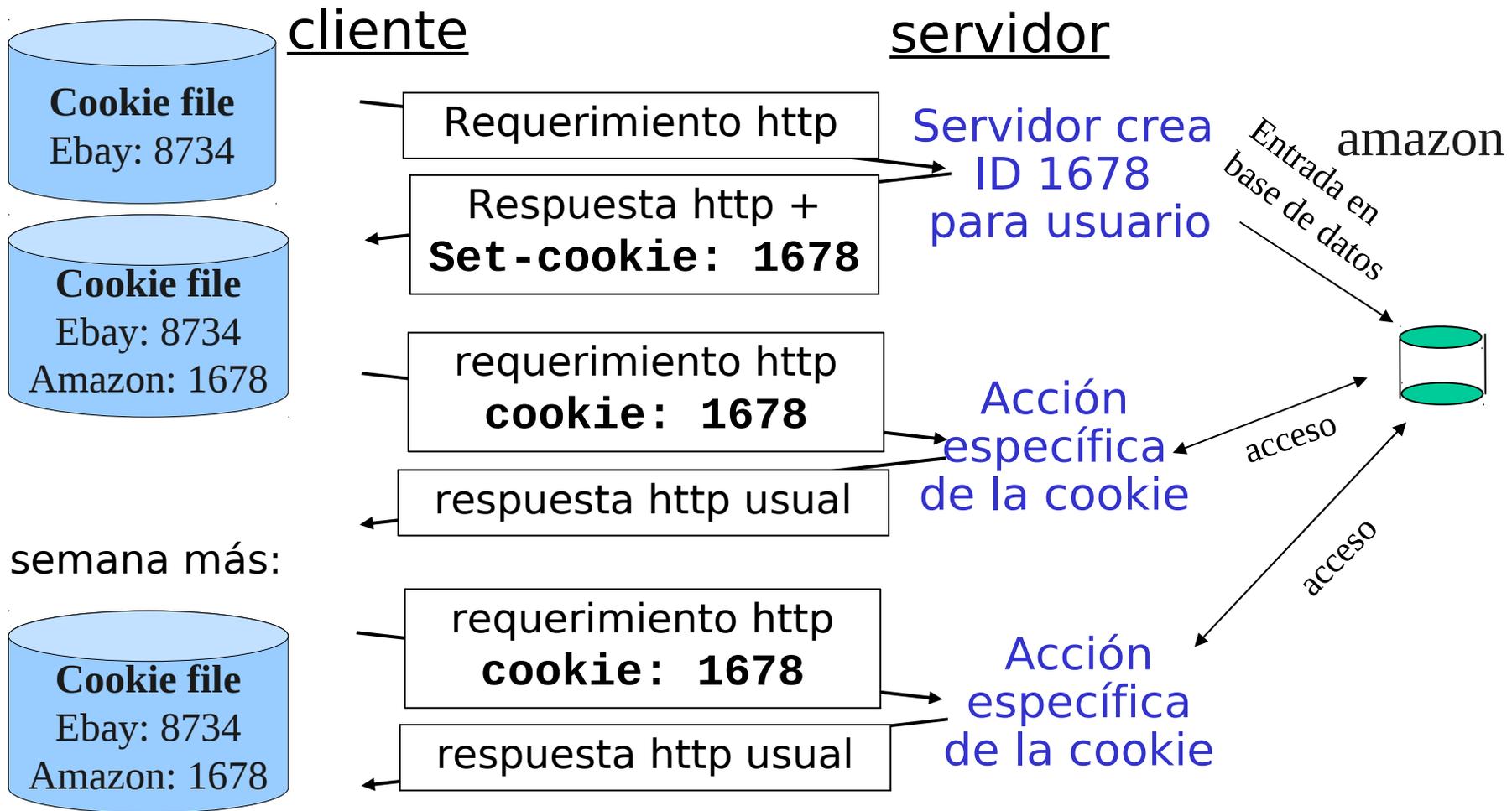
## Cuatro Componentes:

- 1) Línea encabezado cookie en el mensaje respuesta HTTP
- 2) Línea de encabezado cookie en requerimiento HTTP
- 3) Archivo cookie es almacenado en la máquina del usuario y administrada por su navegador.
- 4) Base de datos en sitio Web

## Ejemplo:

- ▣ Susan accede Internet siempre desde el mismo PC
- ▣ Ella visita un sitio e-commerce específico por primera vez.
- ▣ Cuando el requerimiento HTTP inicial llega al sitio, éste crea un ID único y crea una entrada en la base de datos para ese ID.
- ▣ En mensaje respuesta va información del sitio e ID (cookie)
- ▣ El navegador de Susan almacena la cookie en disco.
- ▣ En nuevo acceso al sitio, el navegador incluye ID.

# Cookies: conservando el "estado" (cont.)



# Cookies (cont.)

## Qué pueden transportar las cookies:

- Autorización
- Shopping carts
- Sugerencias
- Estado de la sesión del usuario (Web e-mail)

## Cookies y privacidad:

- Cookies permiten que el sitio aprenda mucho sobre uno.
- Podríamos proveer nombre y correo al sitio.
- Motores de búsqueda usan redirecciones y cookies para aprender aún más
- Compañías de avisos obtienen información de los sitios WEB



El protocolo HTTP no guarda el estado de la conexión; sin embargo, hoy los sitios de ventas por Internet sí pueden hacer seguimiento de un mismo usuario entre accesos distantes en el tiempo. ¿Cómo se logra?



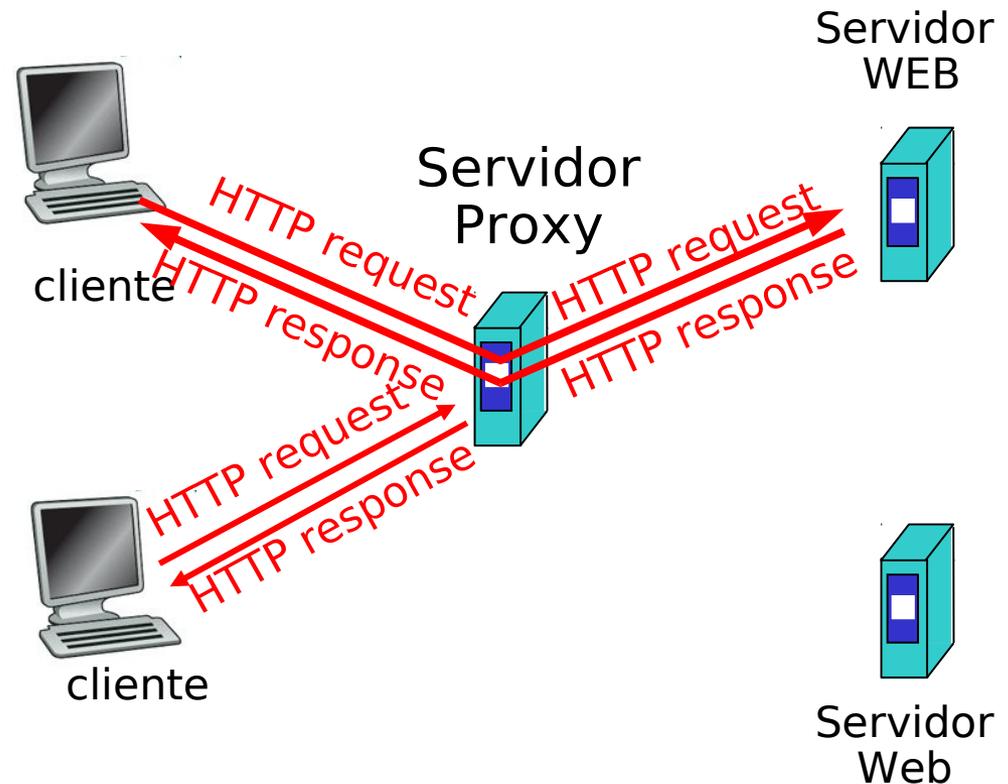
Se logra con el uso de cookies.

Éstas se almacenan en el computador del cliente y cuando éste vuelve a acceder al mismo sitio web, el browser envía la cookie al servidor y así éste reconocer el acceso de un usuario previo.

# Web caches (también servidores proxy)

**Objetivo:** satisfacer el requerimiento del cliente sin involucrar al servidor destino.

- Usuario configura el browser: Acceso Web vía cache
- Browser envía todos los requerimientos HTTP al cache
  - Si objeto está en cache: cache retorna objeto
  - Si no, cache requiere los objetos desde el servidor Web, los almacena y retorna el objeto al cliente.



# Caches v/s proxy

- La idea del **cache** es almacenar “localmente” datos ya solicitados y así poder acceder a los mismos datos más rápidamente en el futuro.
  - Un problema que debe atender el cache es la obsolescencia que puede tener los datos locales.
  - El cache puede usar tiempos de expiración, o consultar a la fuente por vigencia del dato local.
- Un **proxy** es un servicio que consiste en realizar una solicitud a pedido de otro.
- ¿Les ha pasado que para algunas cosas ustedes desean enviar a otro a hacer el trabajo por ustedes?
- Por ejemplo podemos usar proxy para acceder a servicios externos de una intranet, para que desde fuera no se sepa qué computadores hay dentro. El origen es siempre el mismo.

# Más sobre Web caching

- Cache actúa como clientes y servidores
- Típicamente el cache es instalado por ISP (universidad, compañía, ISP residencial)

## Por qué Web caching?

- Reduce tiempo de respuesta a las peticiones del cliente.
- Reduce tráfico en el enlace de acceso al ISP.
- Internet densa con caches permite a proveedores de contenido “chicos” (no \$\$) entregar contenido en forma efectiva.

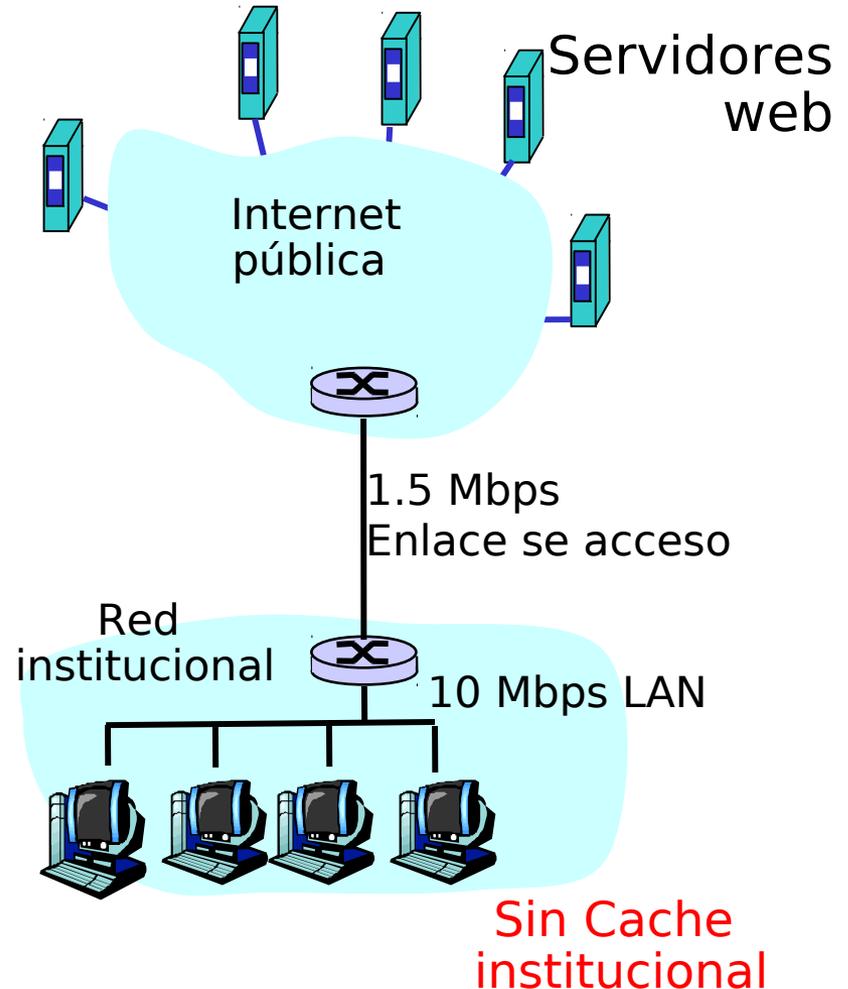
# Ejemplo de Cache

## Suposiciones

- Tamaño promedio de objetos = 100.000 bits
- Tasa de requerimientos promedio desde browsers de la institución al servidor WEB = 15/sec
- Retardo desde el router institucional a cualquier servidor web y su retorno = 2 sec

## Consecuencias

- utilización de la LAN = 15%
- utilización del enlace de acceso = 100%
- Retardo total = retardo Internet + retardo de acceso + retardo LAN = 2 sec + segundos + milisegundos



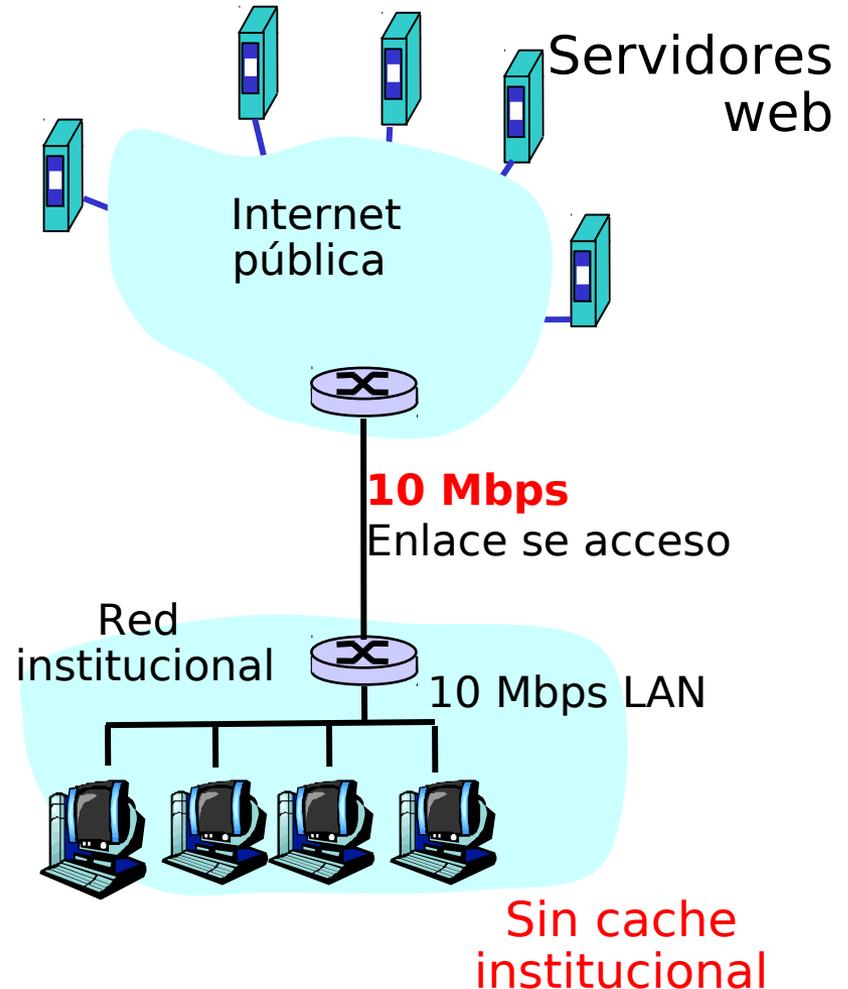
# Ejemplo de Cache (cont)

## Posible solución

- Aumentar ancho de banda del enlace a, por ejemplo, 10 Mbps

## Consecuencias

- Utilización de la LAN = 15%
- Utilización del enlace de acceso = 15%
- Retardo Total = Retardo Internet + retardo de acceso + retardo LAN  
= 2 sec + msec + msec
- A menudo un upgrade caro.



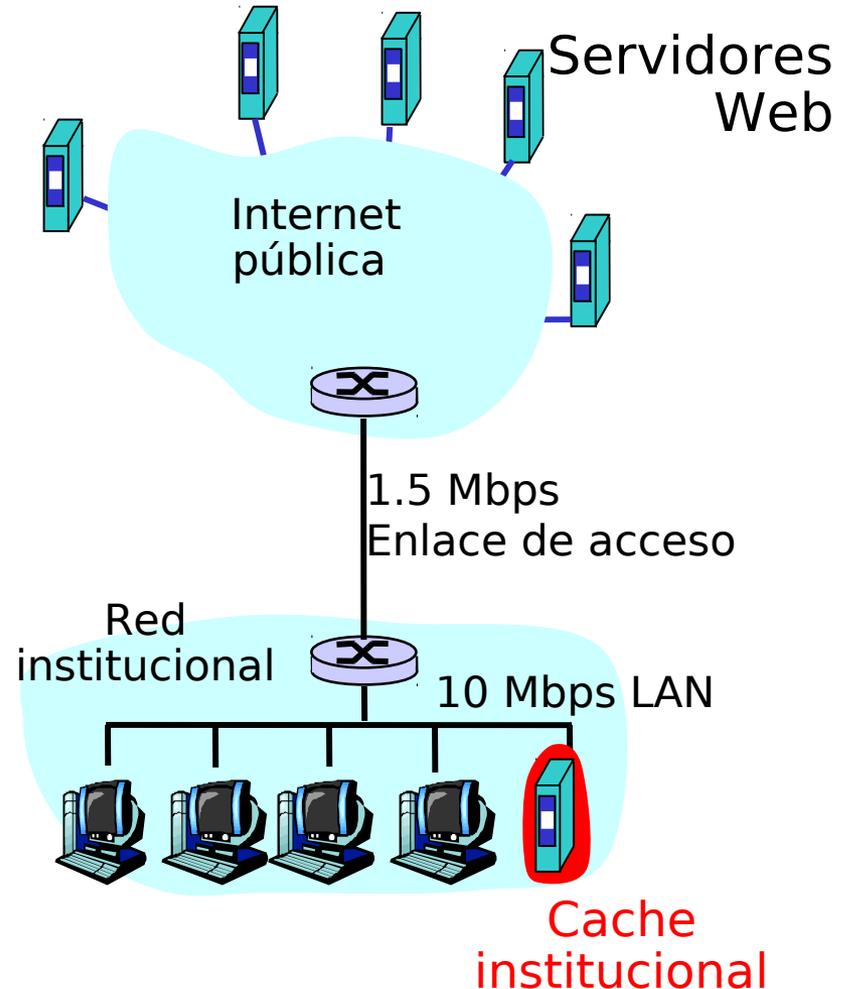
# Ejemplo de cache (cont)

## Instalar un web Cache

- Supongamos tasa de éxito<sup>1</sup> (acierto) de 0.4

## Consecuencias

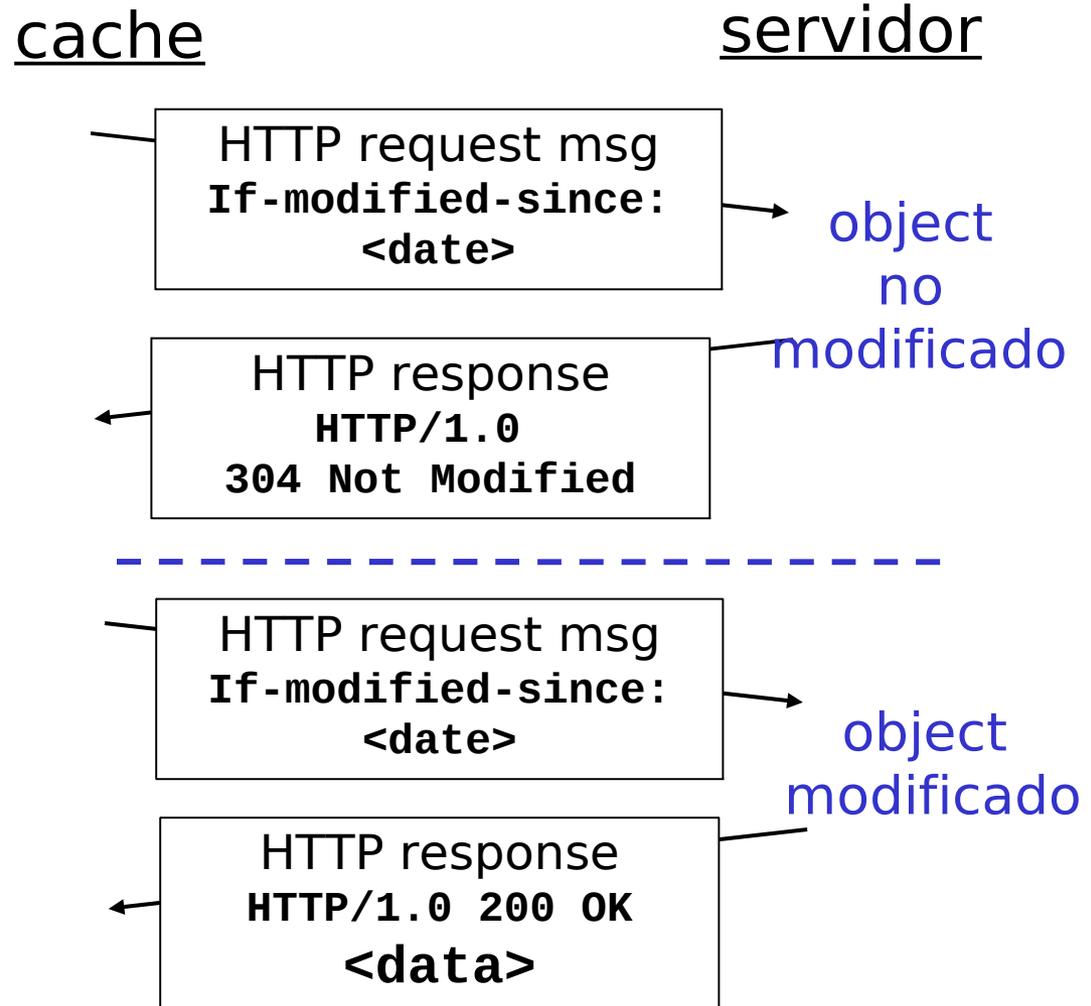
- 40% de los requerimientos serán satisfechos en forma casi inmediata (~10 msec)
- 60% de los requerimientos satisfechos por el servidor WEB
- Utilización del enlace de acceso es reducido al 60%, resultando en retardo despreciable (digamos 10 msec)
- Retardo total = Retardo Internet + retardo acceso + retardo LAN =  $0.6 \cdot (2.01) \text{ sec} + 0.4 \cdot 0.01 < 1.3 \text{ sec}$



<sup>1</sup>Tasa de éxito: Fracción de los requerimientos satisfechos por el cache.

# Get Condicional

- **Objetivo:** verificar que el cache tiene la versión actualizada de un objeto
- Cache: especifica la fecha de la copia en el requerimiento HTTP  
**If-modified-since:**  
**<date>**
- Servidor: responde sin el objeto si la copia de la cache es la última. :  
**HTTP/1.0 304 Not Modified**





## Explique cuándo y cómo se usa el GET condicional.



GET condicional es usado cuando un cache desea saber si un contenido que él tiene ha sido actualizado en el servidor.

Cuando un navegador pide un mismo contenido al cache, éste envía un GET condicional indicando además la fecha del objeto que él posee. Si no hay una nueva versión en el servidor, éste lo informa en el encabezado y no envía los datos. Si hay actualización, la envía y el cache actualiza su objeto.

# Capítulo 2: Capa Aplicación

- 2.1 Principios de la aplicaciones de red
- 2.2 Web y HTTP
- **2.3 FTP**
- 2.4 Correo Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Compartición de archivos
- 2.7 Programación de Socket con TCP
- 2.8 Programación de socket con UDP
- 2.9 Construcción de un servidor WEB