Contenido

- Network Address Translation
- Nivel de Aplicación
  - E-Mail
  - DNS
  - La Web
    - HTTP
    - Procesado en el cliente y en el servidor
  - Seguridad

Contenido

- **Network Address Translation**
- Nivel de Aplicación
  - E-Mail
  - DNS
  - La Web
    - HTTP
    - Procesado en el cliente y en el servidor
  - Seguridad

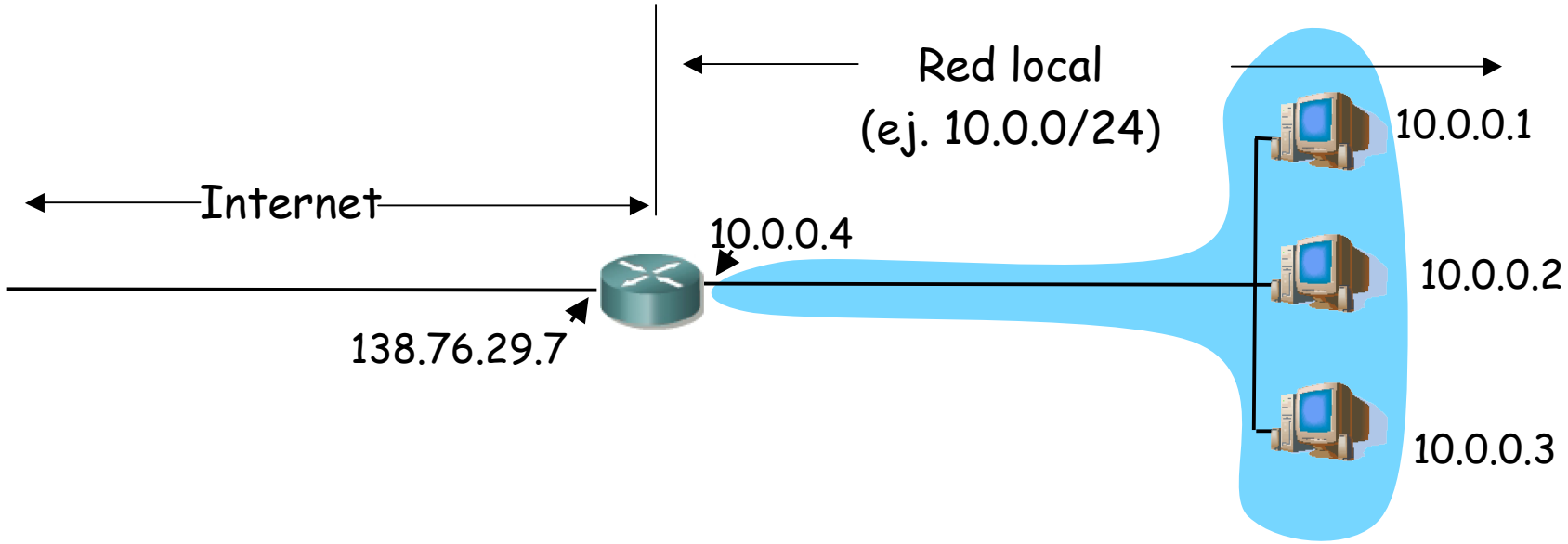
Referencias

- RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations
- RFC 2993: Architectural Implications of NAT
- [Kurose05] págs. 339-342

NAT

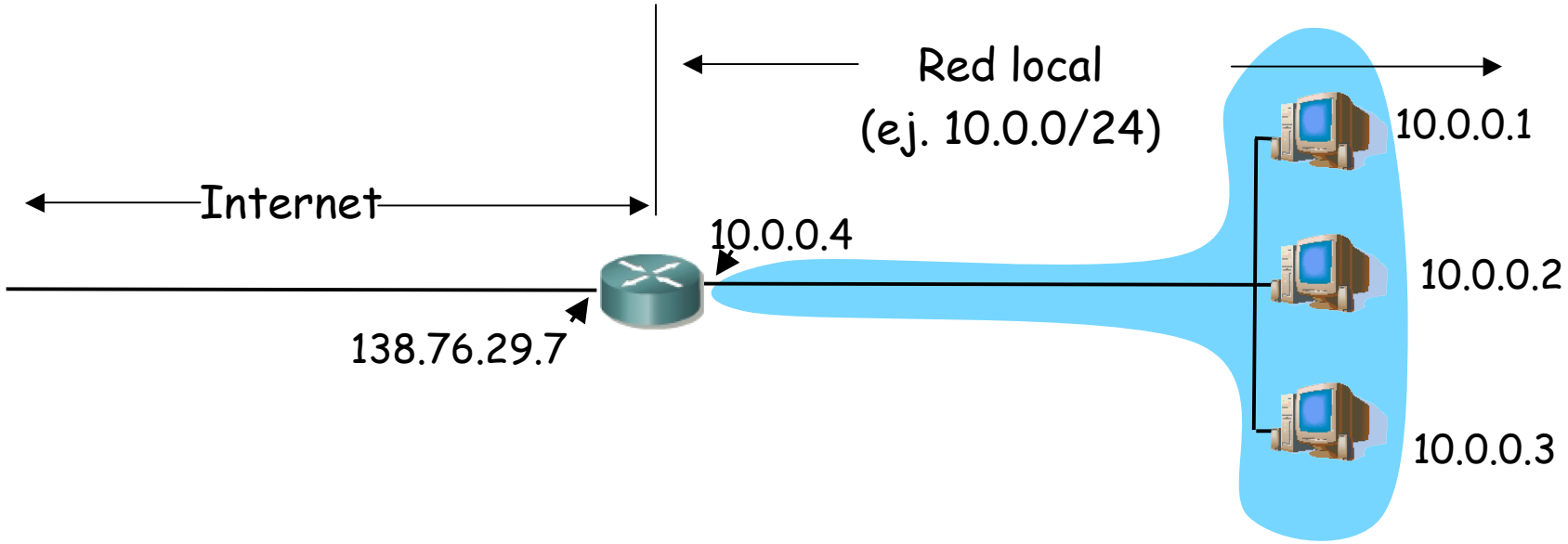
- Network Address Translation
- Una propuesta de solución al problema del agotamiento del espacio de direcciones
- Permite que una red que emplee **direccionamiento privado** se conecte a Internet

- El router que conecta la red a Internet:
  - Cambia la dirección IP privada por una dirección pública al reenviar un paquete hacia el exterior
  - Cambia la dirección IP pública por la correspondiente privada al reenviar un paquete hacia el interior



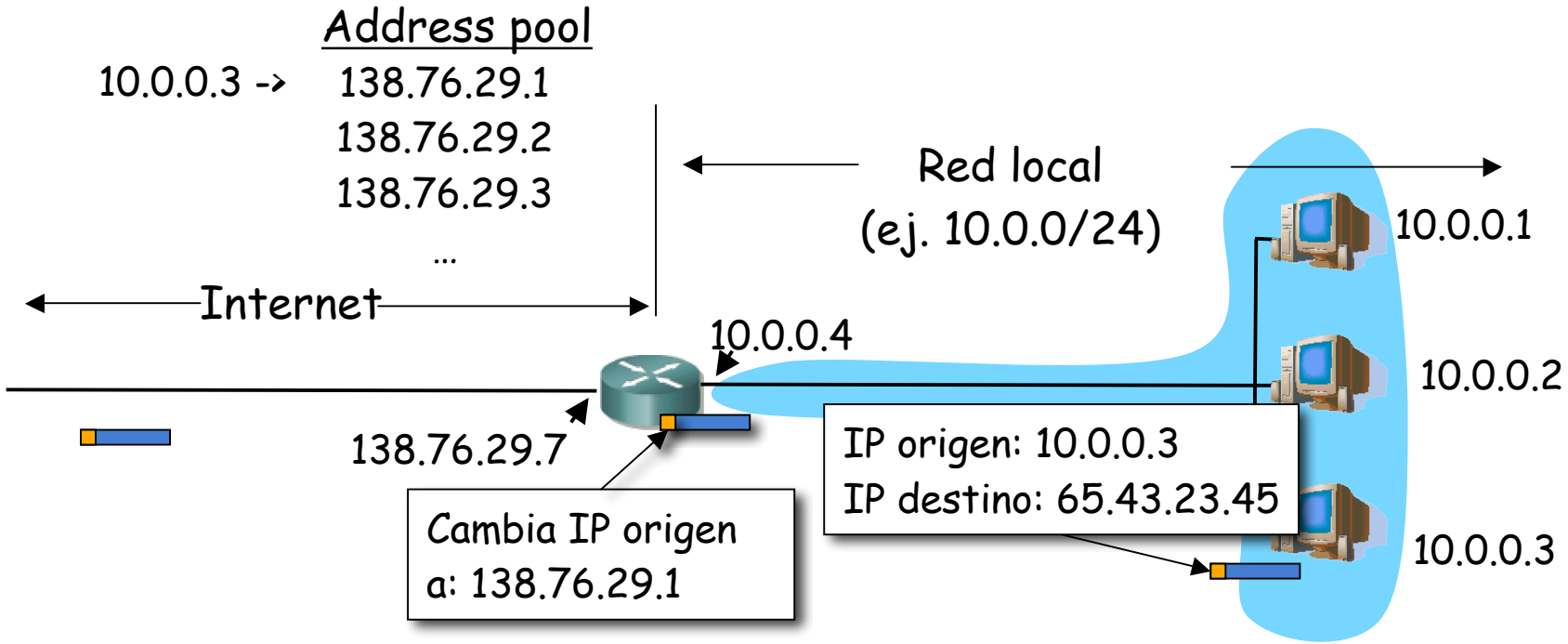
NAT

- El cambio puede ser:
  - **Estático**: una IP interna siempre se cambia por la misma IP pública
  - **Dinámico**: existe un pool de IPs públicas y se establece una relación entre las IPs internas y las de ese pool
- No se necesita reconfigurar los hosts de la red
- Si no todos los hosts de la red desean cursar tráfico con Internet "simultáneamente" no hacen falta tantas direcciones como hosts.



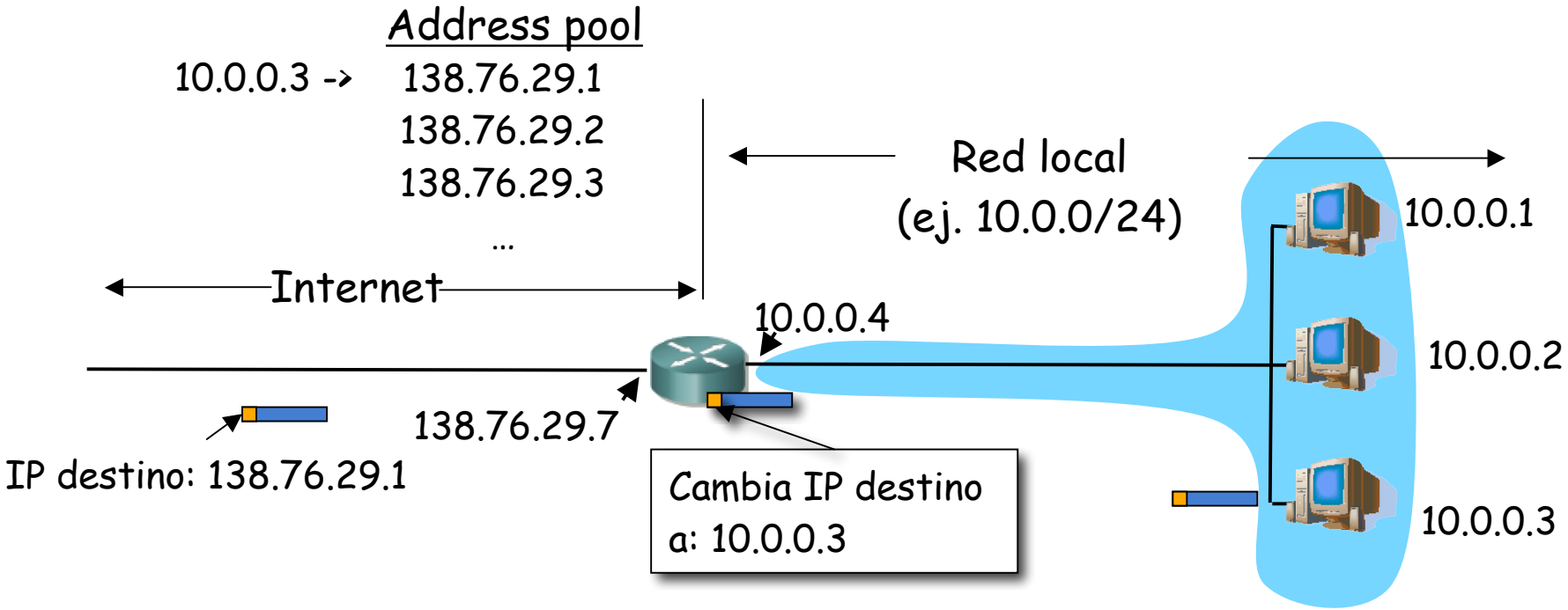
Ejemplo 1

- La red interna tiene direccionamiento privado
- El interfaz del router tiene una dirección pública
- Además tiene un **pool de direcciones** publicas disponibles
- Cuando un host quiere enviar un paquete IP a un destino en Internet el router NAT cambia la dirección IP origen antes de reenviarlo (…)
- El router NAT apunta la dirección por la que la ha cambiado (…)



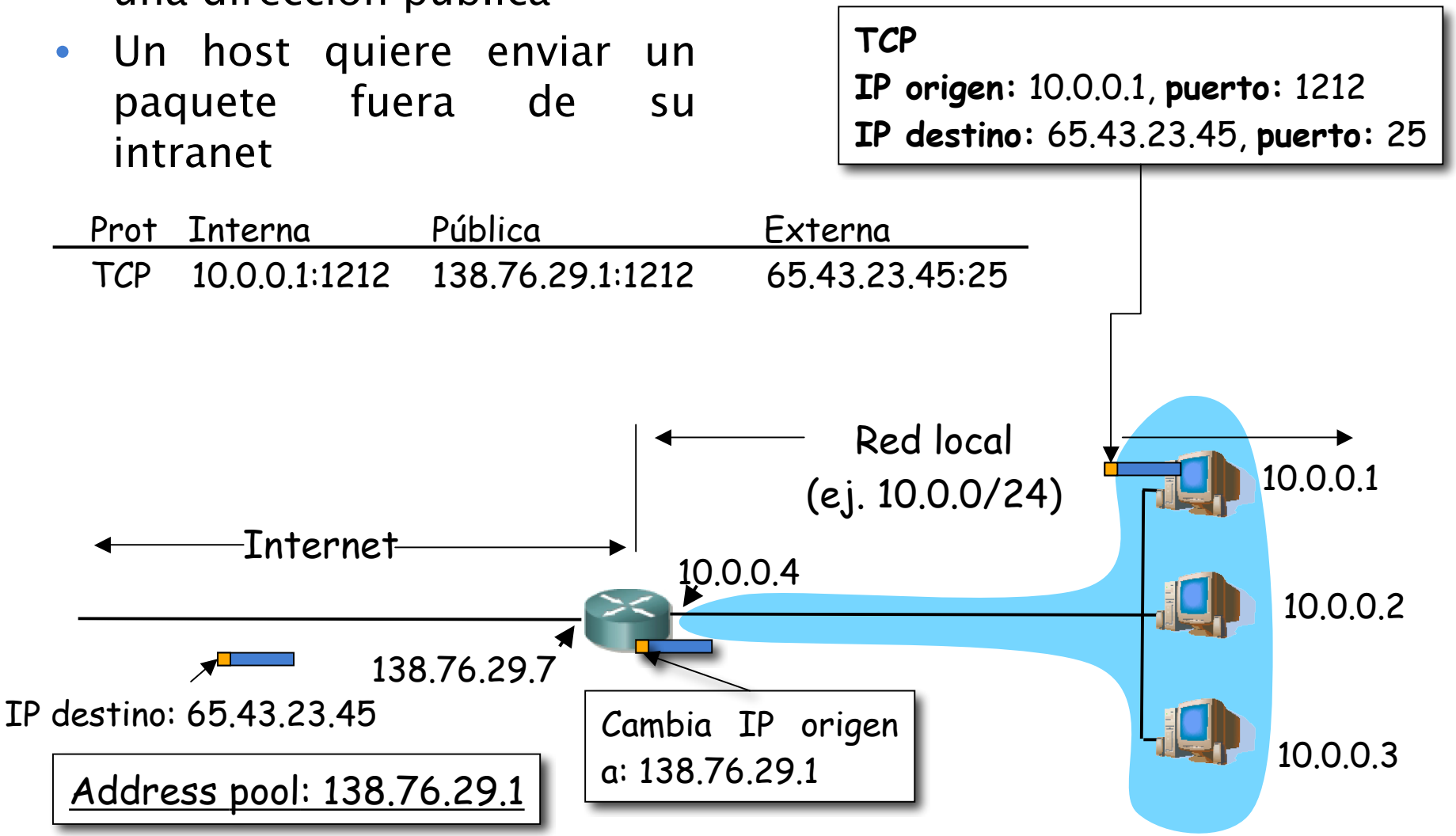
Ejemplo 1

- Cuando venga un paquete de esa IP destino vendrá dirigido a la IP que colocó el router NAT
- El router NAT ve en su tabla la dirección IP interna a la que corresponde y la cambia (… …)



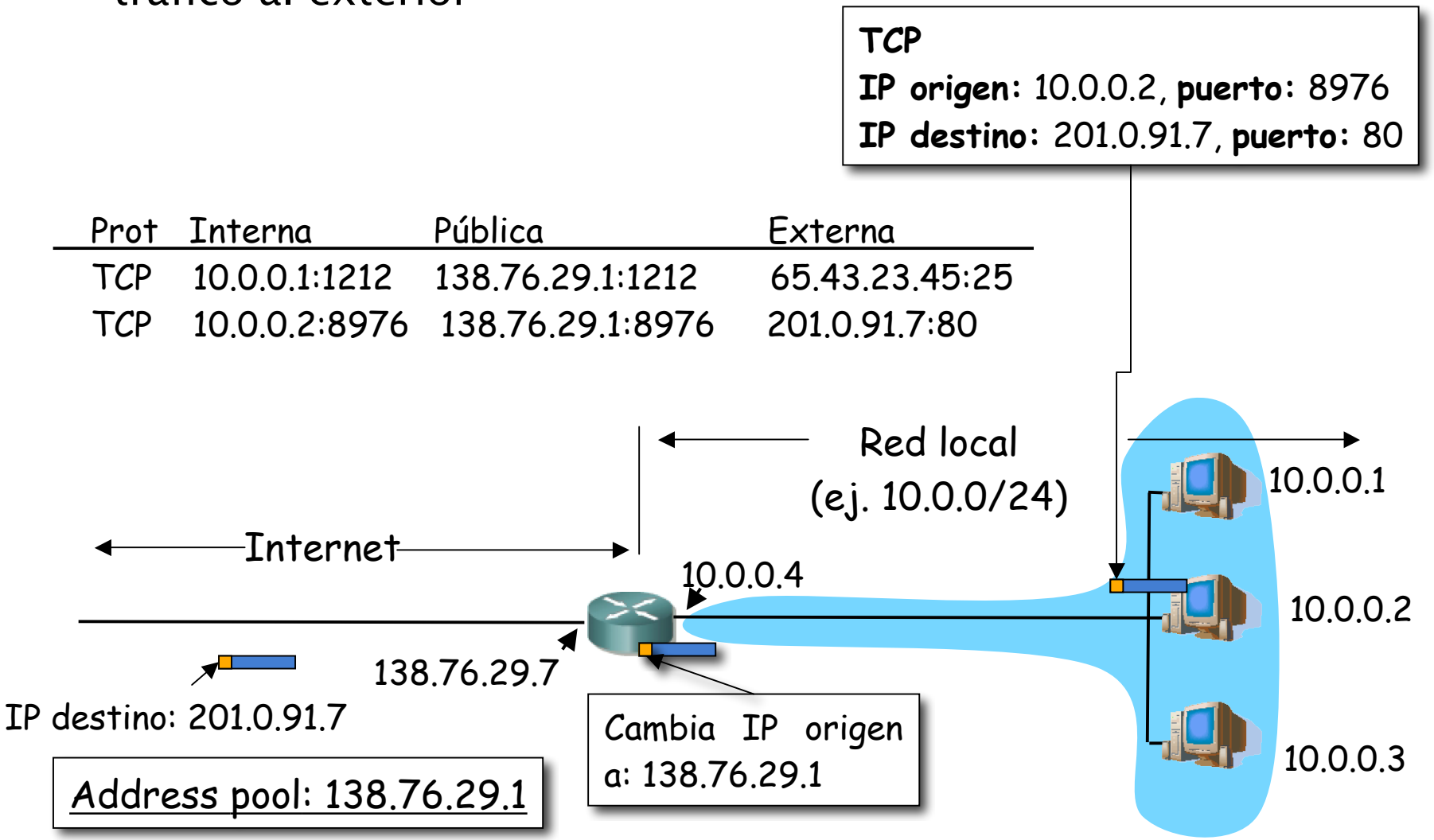
Ejemplo 2: Sobrecarga

- Supongamos que solo hay una dirección pública
- Un host quiere enviar un paquete fuera de su intranet



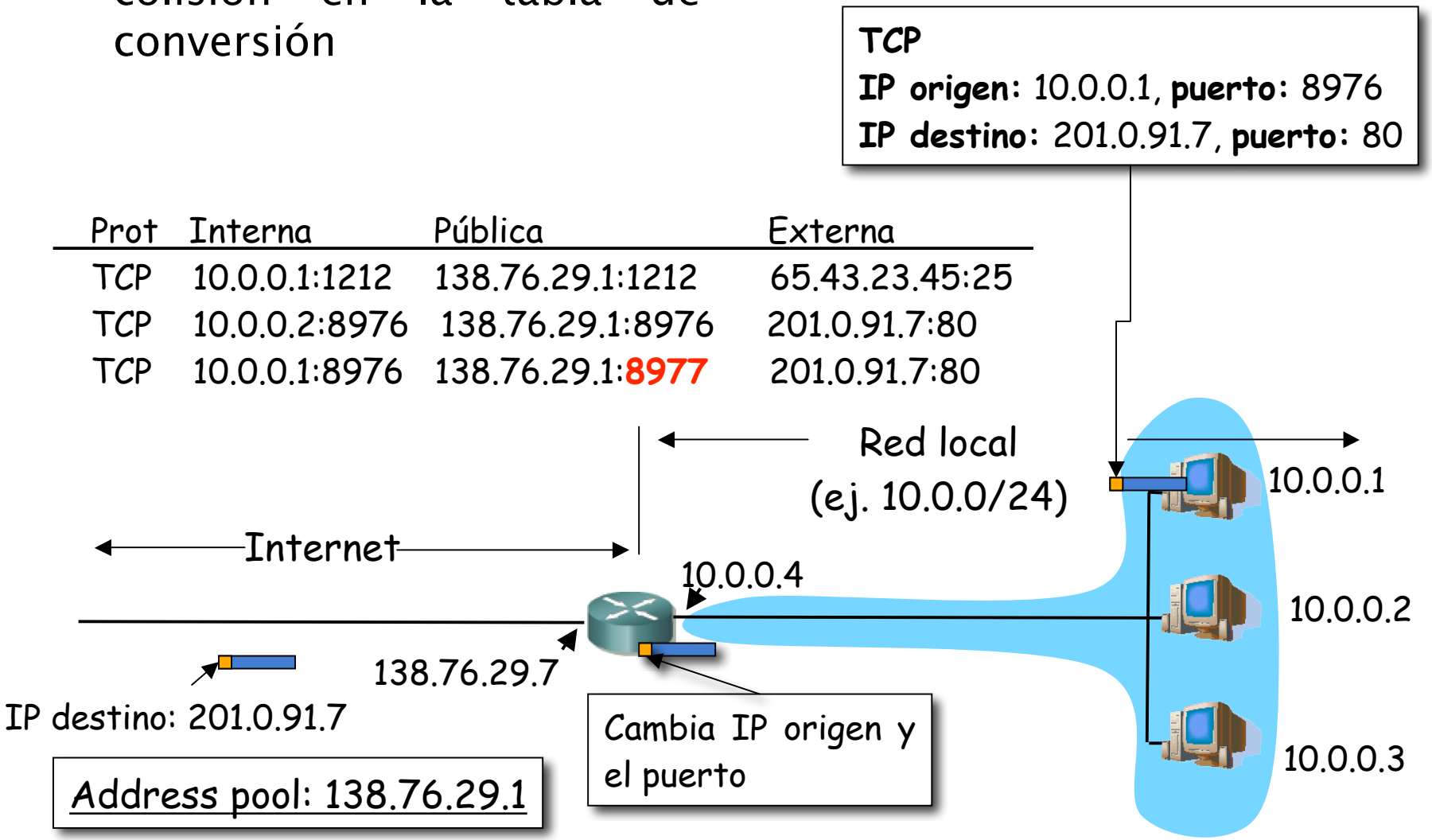
Ejemplo 2: Sobrecarga

- Otro host también envía tráfico al exterior



Ejemplo 2: Sobrecarga

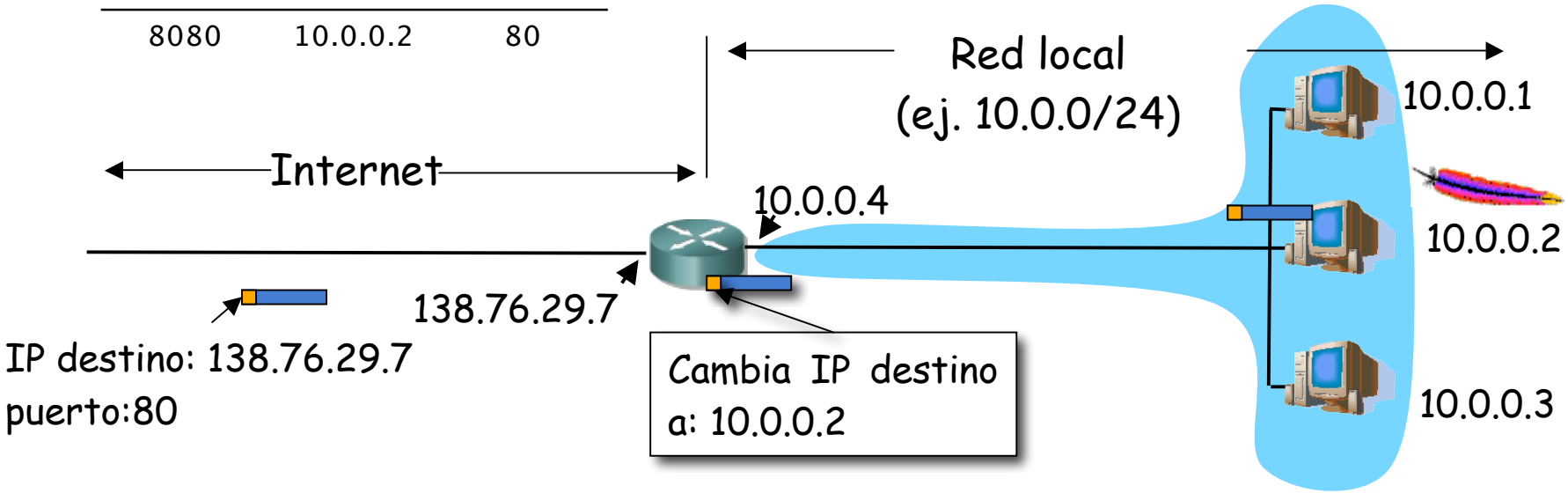
- Se puede producir una colisión en la tabla de conversión



Conexiones entrantes

- "Redirigir" los establecimientos de conexión a un puerto hacia una IP interna (…)
- Cambiando o no el puerto

Puerto externo	IP interna	Puerto interno
80	10.0.0.2	80
25	10.0.0.3	25
8080	10.0.0.2	80



NAT

Ventajas

- Se puede cambiar el rango de direcciones sin notificar
- Puede cambiar de ISP sin cambiar las direcciones
- Máquinas no accesibles desde el exterior (seguridad)
- ¿Una sola IP en el pool? La del router

Inconvenientes

- El puerto es de 16bits:
  - 64K conexiones con una sola dirección
- Consume memoria
- Controvertido:
  - Los routers solo hasta el nivel de red
  - Servidores no accesibles desde el exterior
  - Rompe el esquema extremo a extremo
  - Los diseñadores de aplicaciones deberán tener en cuenta la posibilidad de existencia de NATs entre cliente y servidor

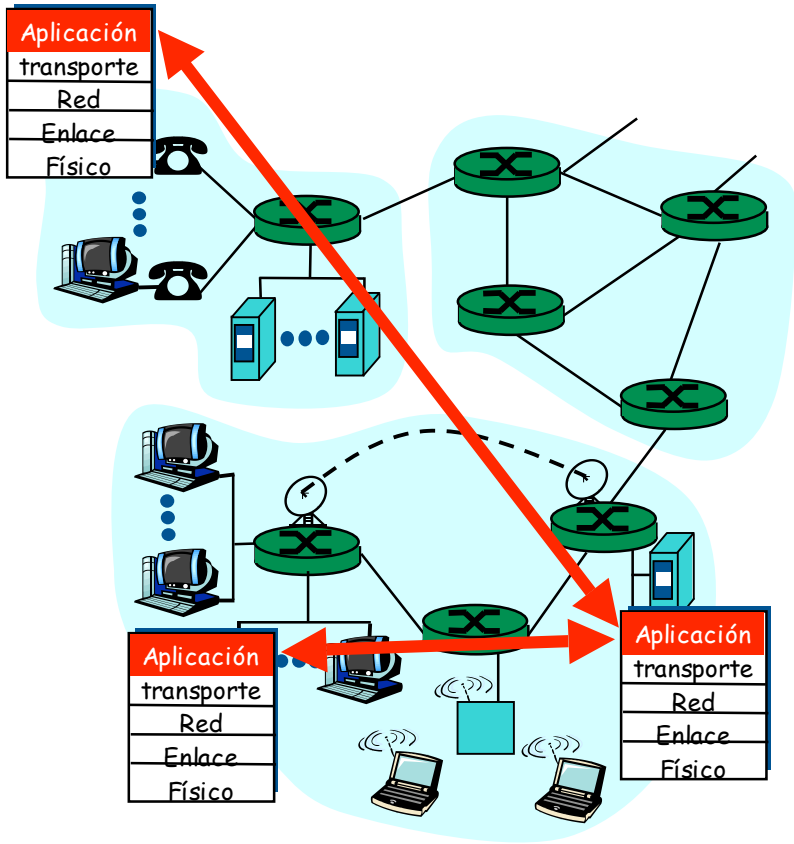
Contenido

- Network Address Translation
- **Nivel de Aplicación**
  - E-Mail
  - DNS
  - La Web
    - HTTP
    - Procesado en el cliente y en el servidor
  - Seguridad

Aplicaciones en red

Las aplicaciones

- Son software
- Diferentes máquinas y Sistemas Operativos
- Quienes se comunican son procesos
- IPC: Inter Process Communication
- Nos interesan procesos ejecutándose en diferentes máquinas
- Se comunican a través de una red
- Intercambian mensajes
- Emplean Protocolos de nivel de aplicación...



Aplicaciones y Protocolos

Los Protocolos de aplicación son una parte de las aplicaciones de red... ...

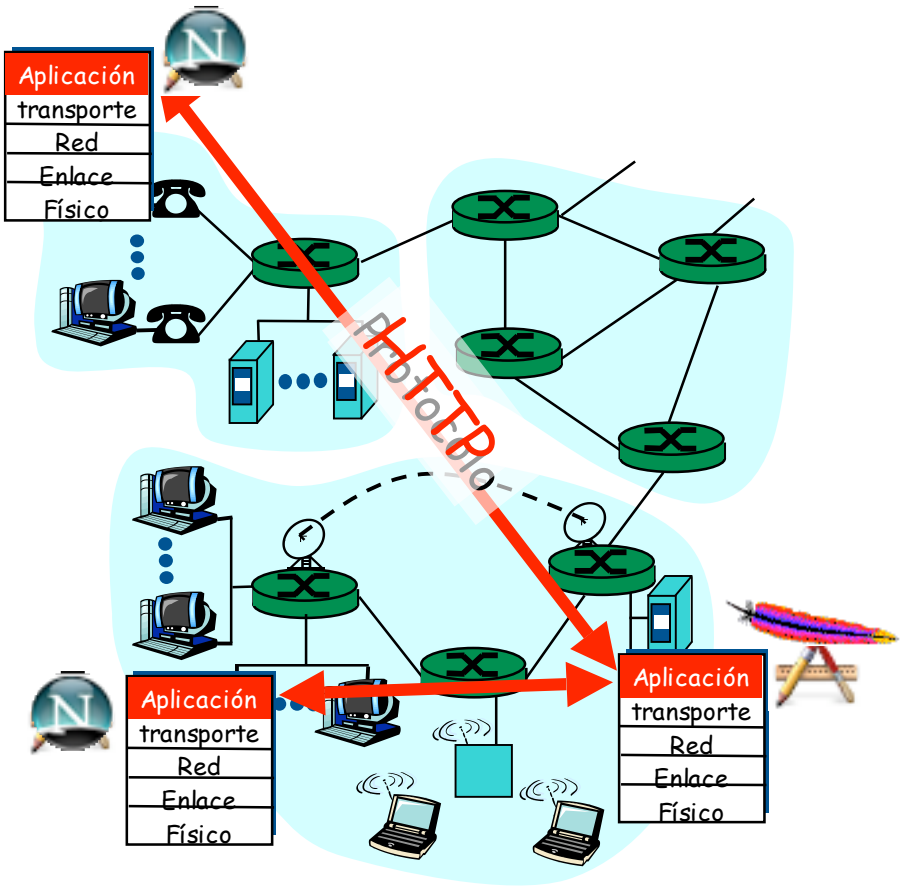
Definen:

- Tipos de mensajes
- Sintaxis/formato de mensajes
- Significado del contenido
- Reglas de funcionamiento

Ejemplo: La Web

- Navegador, Servidor Web...
- HTTP ...

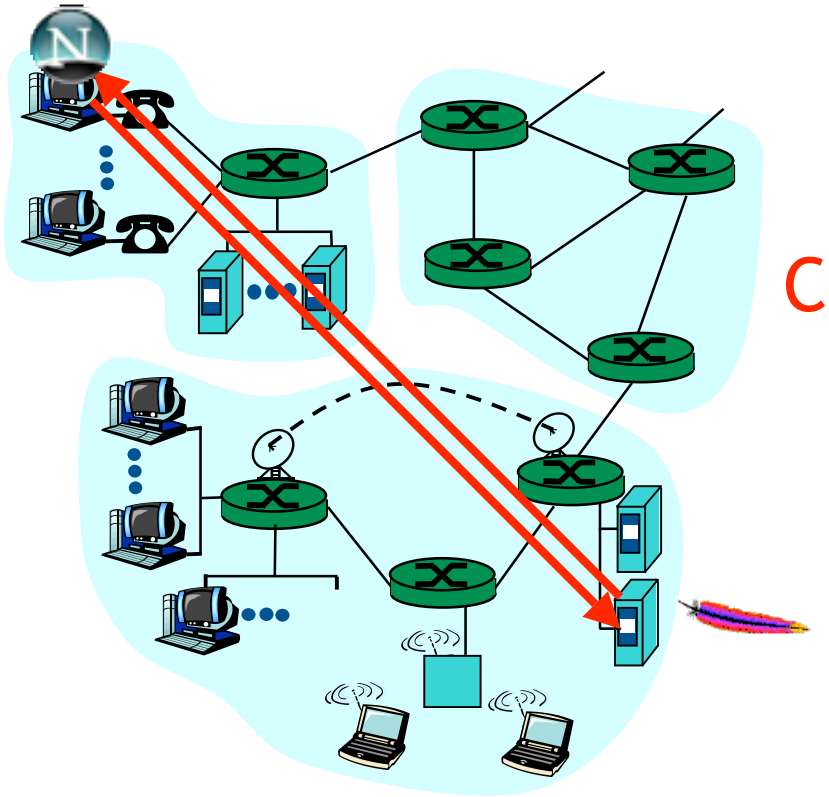
Muchos protocolos son estándares abiertos (en RFCs)



Paradigmas

- Cliente-servidor
- Peer-to-peer (P2P)
- Híbrido de cliente-servidor y P2P

Arquitectura cliente-servidor



Servidor:

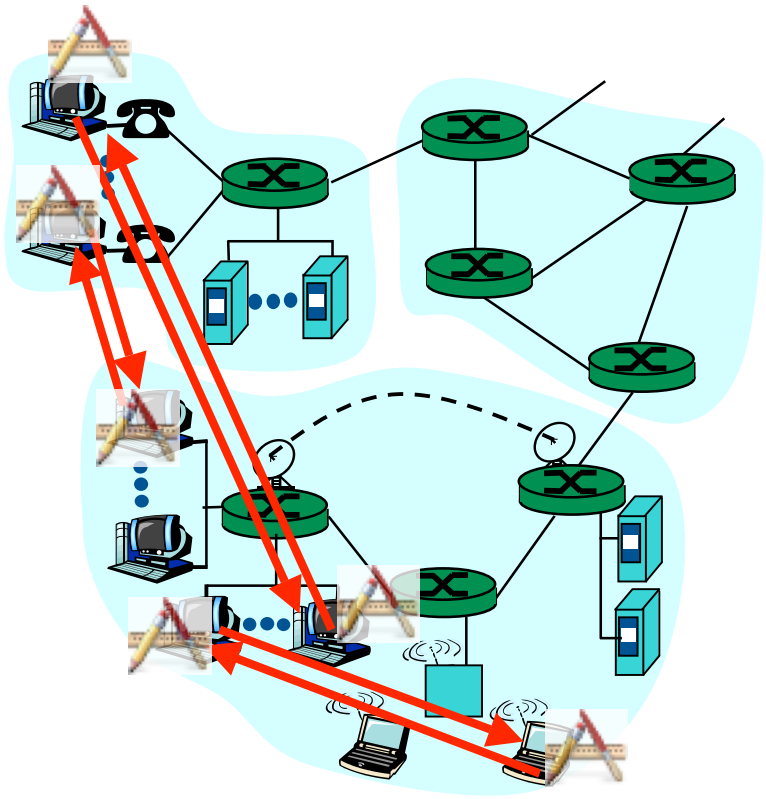
- Comienza a ejecutarse primero...
- Espera a ser contactado
- Host siempre disponible
- Dirección permanente

Cliente:

- Lanzado más tarde por el usuario...
- Inicia la comunicación con un servidor...
- No con clientes
- Termina cuando el usuario deja de usarlo
- Puede no tener siempre la misma dirección

Arquitectura Peer-to-Peer

- No hay un servidor siempre disponible
- Hosts extremos cualesquiera se comunican (peers)...
- Pueden no estar siempre conectados...
- Los peers pueden cambiar de dirección
- El mismo proceso puede ser cliente o servidor
- Ejemplo: Gnutella



Escalable

Difícil de controlar

Híbrido de cliente-servidor y P2P

Napster

- Transferencia de ficheros P2P
- Búsqueda de ficheros centralizada:
  - Peers registran el contenido ofrecido en un servidor central
  - Peers preguntan al mismo servidor para buscar ficheros

Mensajería Instantánea (Instant messaging=IM)

- Conversación entre dos usuarios suele ser P2P
- Detección de presencia y localización centralizada:
  - Los usuarios registran su dirección en un servidor central cuando se conectan a la red
  - Contactan con el servidor central para encontrar la dirección actual de sus contactos

Servicios que necesitan las apps

Pérdidas

- Algunas apps soportan pérdidas (ej. audio)
- Otras requieren 100% de fiabilidad (ej. transferencia de ficheros)

Retardo

- Algunas apps requieren bajo retardo (ej. juegos en red)

Ancho de banda

- Algunas apps requieren un mínimo de ancho de banda (ej. audioconf)
- Otras (elásticas) funcionan con cualquier cantidad pero pueden sacar provecho a todo el disponible

Requisitos de aplicaciones comunes

Aplicación	Pérdidas	Ancho de banda	Retardo
Transf. ficheros	ninguna	elastico	no
e-mail	ninguna	elastico	no
Web	ninguna	elastico	no
audio/vídeo en RT	soporta	audio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sí, 100’s mseg
audio/vídeo diferido	soporta	idem	sí, unos segs
juegos interactivos	soporta	desde unos kbps	sí, 100’s mseg
IM	ninguna	elastico	sí y no

Contenido

- Network Address Translation
- Nivel de Aplicación
  - **E-Mail**
  - DNS
  - La Web
    - HTTP
    - Procesado en el cliente y en el servidor
  - Seguridad

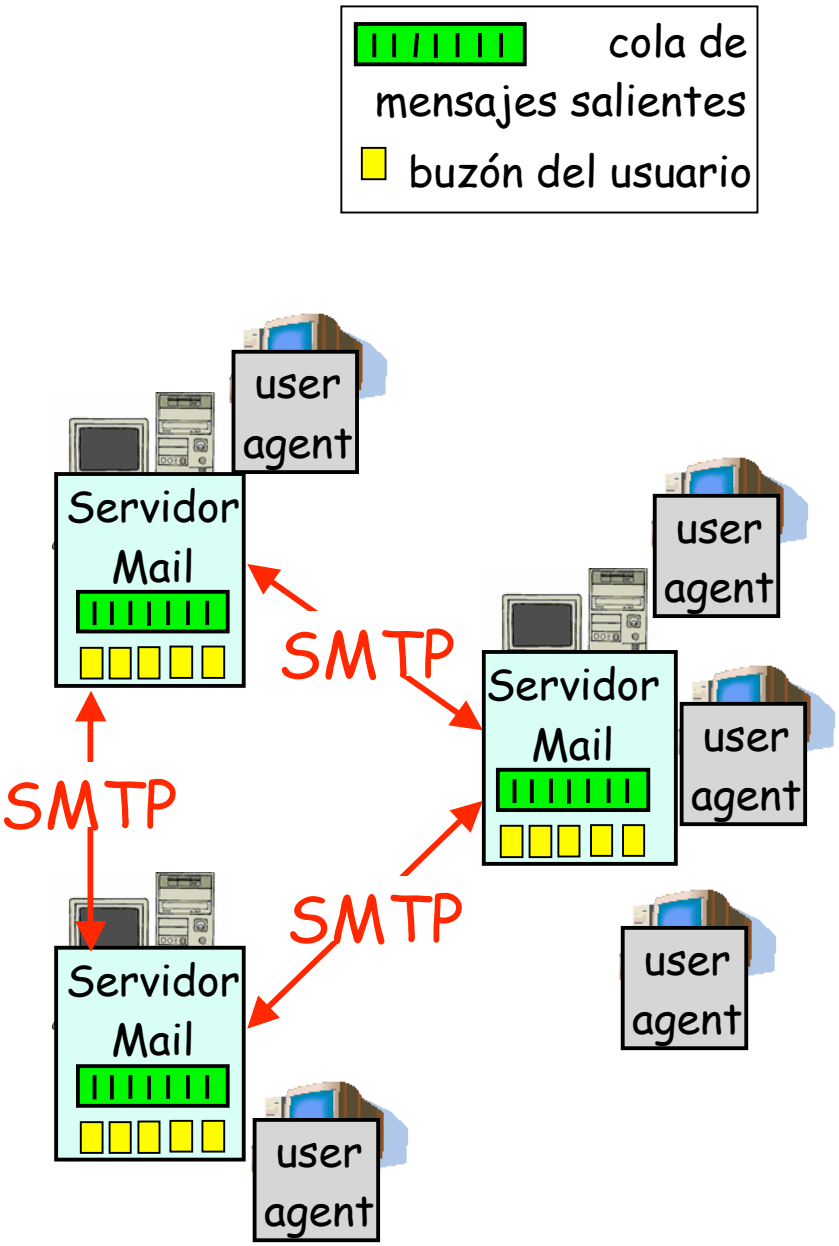
Electronic Mail

Tres elementos principales:

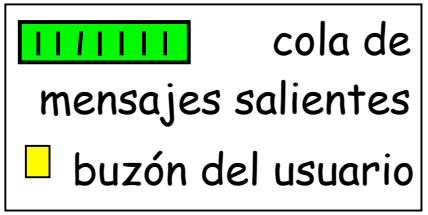
- Agentes de usuario (*user agents*)
- *Mail servers*
- Simple Mail Transfer Protocol: **SMTP**

User Agent

- alias “programa de correo”
- Componer, editar, leer mensajes de correo
- ej., Eudora, Outlook, elm, Netscape Messenger
- Mensajes salientes y entrantes en el servidor

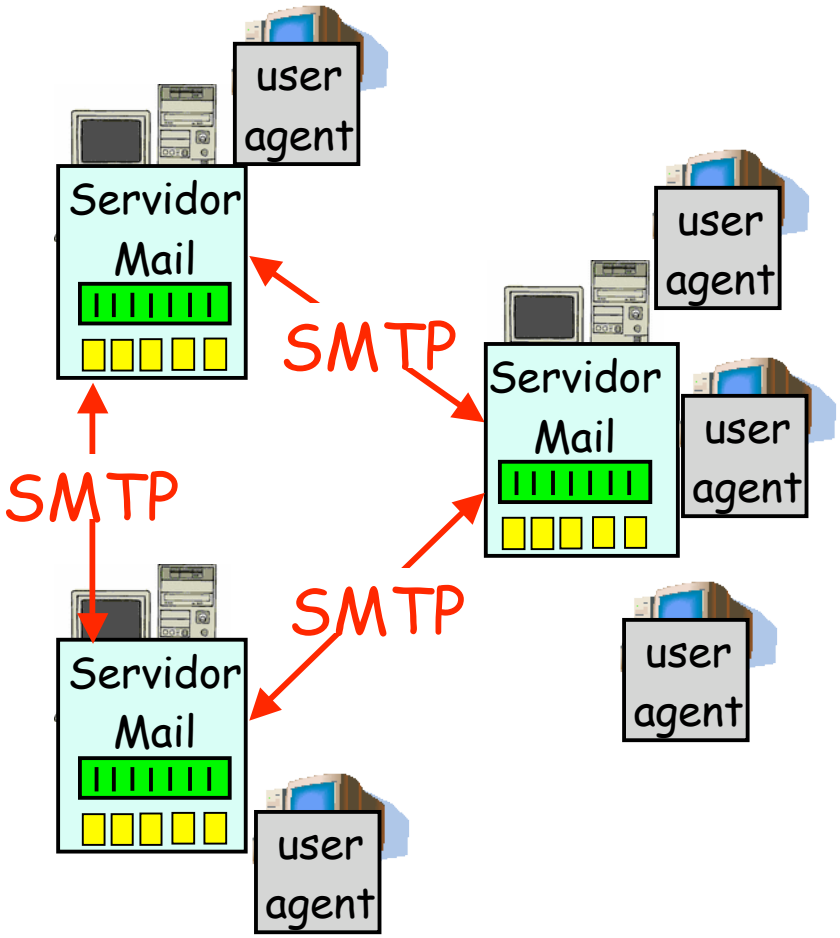


E-Mail: Servidores



Servidores de Mail:

- ***Mailbox*** contiene los mensajes entrantes para el usuario
- ***Cola de mensajes*** salientes (a enviar)
- **Protocolo *SMTP*** entre servidores de correo para enviar mensajes
  - cliente: el servidor de correo que envía
  - "servidor": el servidor de correo que recibe

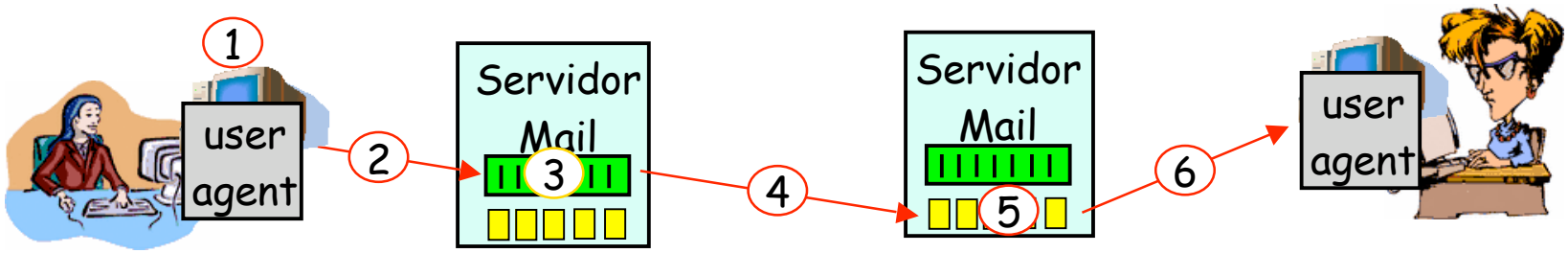


E-Mail: SMTP [RFC 2821]

- Emplea **TCP** para entregar de forma fiable los mensajes entre el cliente y el servidor
- Puerto **25**
- Transferencia directa: del servidor del emisor al servidor del receptor
- **Tres fases** en al transferencia
  - handshaking (el saludo)
  - transferencia de mensajes
  - cierre
- Interacción mediante comandos y respuestas
  - comandos: texto ASCII
  - respuestas: código de estado y frase de estado
- Los mensajes deben estar en **ASCII** de 7 bits

Ejemplo: Usuario 1 envía mensaje a Usuario 2

1) Usuario 1 emplea un UA para crear el mensaje para usuario2@micasa.com
2) El programa envía el mensaje a su servidor de correo y lo coloca en una cola de mensajes
3) El Servidor de Mail, como cliente, abre una conexión TCP con el Servidor de Usuario 2
4) Envía el mensaje de Usuario 1 empleando SMTP sobre esa conexión TCP
5) El servidor de mail de Usuario 2 coloca el mensaje en su buzón
6) Usuario 2 lanza su UA para leer el mensaje (volveremos a esta parte)



Ejemplo de SMTP

- 220 unavarra.es ESMTP Sendmail 8.9.3/8.9.1 (IRIS 3.0); Fri, 29 Apr 2005 14:00:19 +0200 (MET DST)
- **HELO daniel.tlm.unavarra.es**
- 250 unavarra.es Hello s169m159.unavarra.es [130.206.169.159], pleased to meet you
- **MAIL FROM: <daniel.moraro@unavarra.es>**
- 250 <daniel.moraro@unavarra.es>... Sender ok
- **RCPT TO: danielmorato@yahoo.com**
- 250 danielmorato@yahoo.com... Recipient ok
- **DATA**
- 354 Enter mail, end with "." on a line by itself
- **Hola**
- **Aqui, saludandome a mi mismo**
- **.**
- 250 OAA24057 Message accepted for delivery
- **QUIT**
- 221 unavarra.es closing connection

Probando SMTP

- **`telnet servername 25`**
- Pruebe los comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT
- Con esos comandos puede enviar un email sin emplear un programa de email

Algo más sobre SMTP

- Conexiones persistentes
- Requiere que el mensaje (cabecera y contenido) esté en ASCII de 7 bits
- El servidor de SMTP emplea **`CRLF.CRLF`** para reconocer el final del mensaje

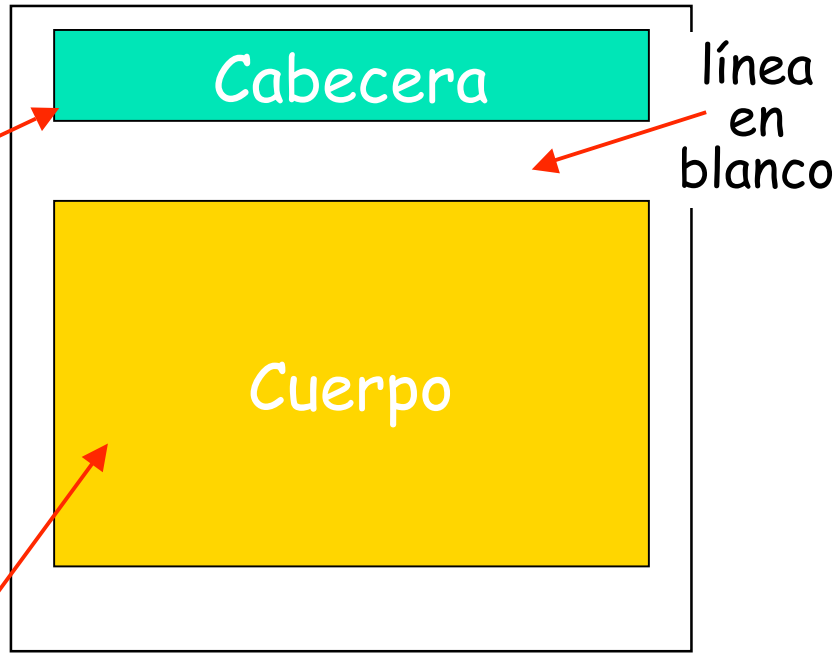
Comparación con HTTP:

- HTTP: pull
- SMTP: push
- Ambos emplean comandos y respuestas en ASCII

Formato del mensaje de email

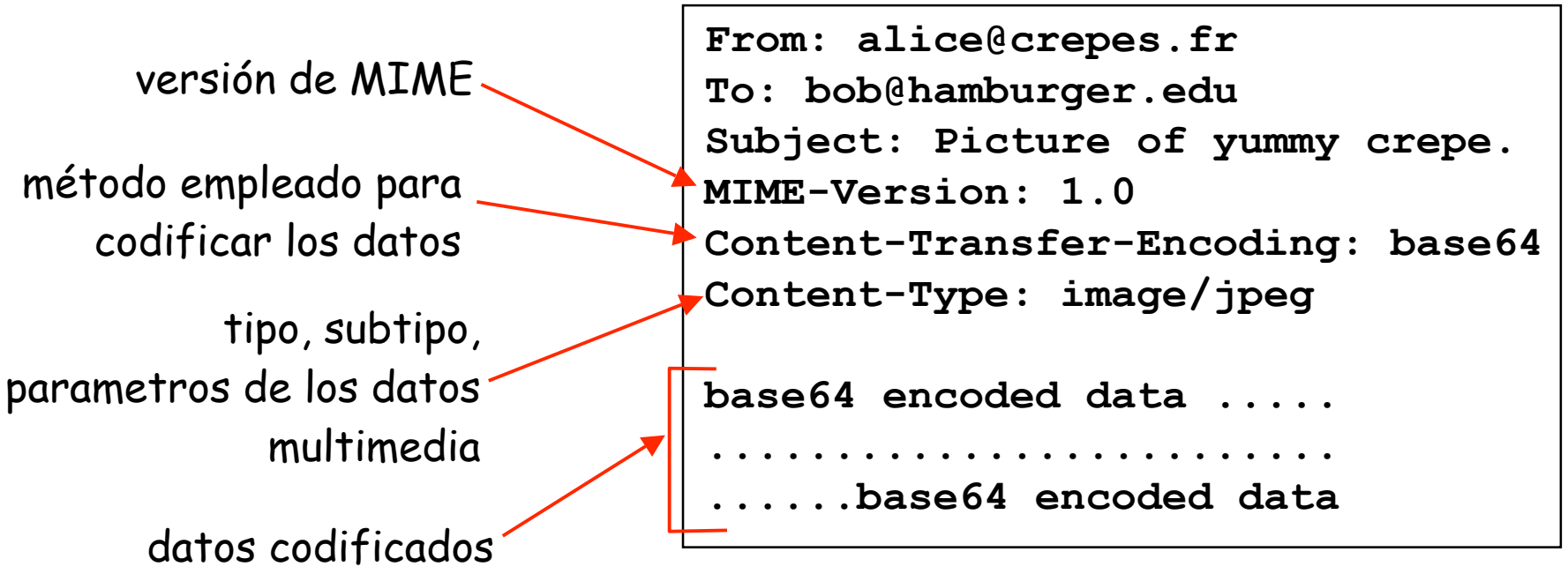
- SMTP: protocolo para intercambiar mensajes de email (RFC 2821)
- RFC 822: estándar para el formato del mensaje:
- Líneas de cabecera, ej.,
  - `To:`
  - `From:`
  - `Subject:`

  Diferentes de los comandos de SMTP
- Cuerpo
  - el “mensaje”, solo caracteres ASCII

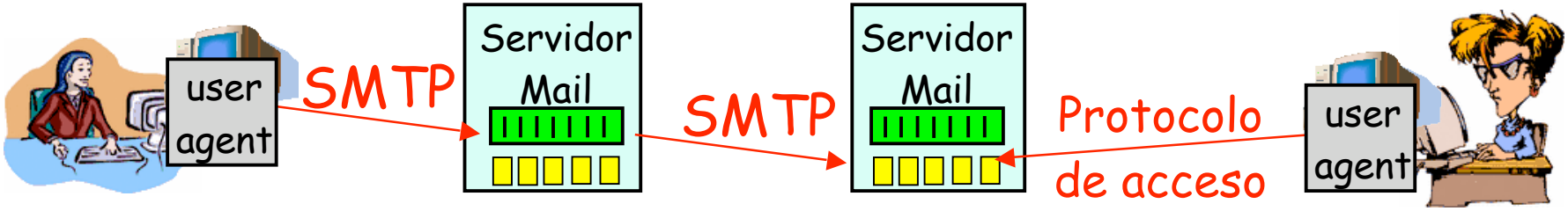


Formato del mensaje: Multimedia Extensions

- MIME: MultImedia Mail Extension, RFC 2045, 2056
- Permite mandar contenido que no sea texto ASCII
- Líneas adicionales en la cabecera del mensaje para declarar el tipo del contenido



Protocolos de acceso al Mail



- SMTP: entrega/almacena en el servidor del receptor
- Protocolo de acceso al Mail: obtención de mensajes del servidor
  - POP: Post Office Protocol [RFC 2821]
    - Autorización (agente ↔ servidor) y descarga
  - IMAP: Internet Message Access Protocol [RFC 3501]
    - Más funcionalidades (más complejo)
    - Manipulación de mensajes almacenados en el servidor
  - HTTP: Hotmail , Yahoo! Mail, etc.

Protocolo POP3

Autorización

- Comandos del cliente:
  - **`user`** : declara el nombre de usuario
  - **`pass`** : clave
- Respuestas del servidor:
  - **`+OK`**
  - **`-ERR`**

Fase de transacción, cliente:

- **`list`** : lista números de mensajes
- **`retr`** : descarga mensaje por número
- **`dele`** : borrar
- **`quit`**

+OK Qpopper (version 4.0.5) at si starting.
user daniel.morato
+OK
pass hungry
+OK daniel.morato has 412 visible messages (0 hidden) in 35020509 octets.

list
1 498
2 912
.
retr 1
<contenido mensaje 1>
.
dele 1
retr 2
<contenido mensaje 2>
.
dele 2
quit
+OK POP3 at si signing off

Más sobre POP3 e IMAP

Más sobre POP3

- El ejemplo anterior era “descargar y borrar”
- Bob no puede volver a leer los mensajes si cambia de cliente
- “Descargar y mantener”: copia el mensaje pero no lo borra. Permite descargarlos en otro cliente
- POP3 es sin estado entre sesiones

IMAP

- Mantiene todos los mensajes en un lugar: el servidor
- Permite al usuario organizar los mensajes en carpetas
- IMAP mantiene el estado entre sesiones:
  - Nombres de carpetas y relación entre ID de mensaje y carpeta en la que está

Contenido

- Network Address Translation
- Nivel de Aplicación
  - E-Mail
  - **DNS**
  - La Web
    - HTTP
    - Procesado en el cliente y en el servidor
  - Seguridad

Referencias

- [Forouzan03] 18.1-18.5

El problema de los nombres

- Las direcciones IP, que identifican a los interfaces de los hosts, son números de 32 bits
- Sencillas de manejar para las máquinas, complicado para los humanos
- Más sencillo memorizar nombres textuales
- Hace falta “traducir” el nombre textual en la dirección numérica para que se pueda realizar la comunicación. Esto se llama “resolver el nombre”
- La traducción se realiza mediante el Sistema de Nombres de Dominio o DNS (Domain Name System)

Domain Name System

- Es una **base de datos distribuida**
- Servidores de nombres organizados **jerárquicamente**
- Es un **protocolo de aplicación**
- Permite a los hosts traducir entre nombres y direcciones
  - Funcionalidad vital
  - Implementada como protocolo a nivel de aplicación
  - Complejidad en los extremos de la red

¿Por qué no centralizado?

- Punto de fallo
- Volumen de tráfico
- Base de datos centralizada lejana
- Mantenimiento

¡ No escala !

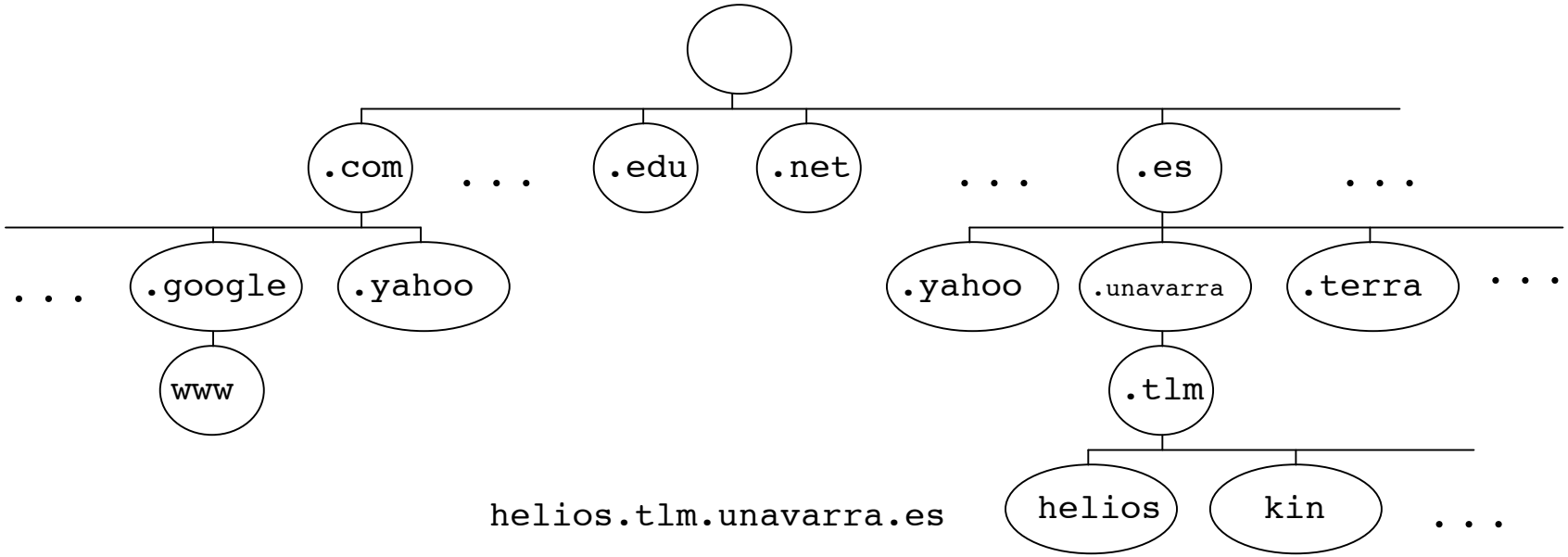
Jerarquía de nombres

- Los nombres están formados por segmentos alfanuméricos separados por puntos (no distingue mayúsculas)

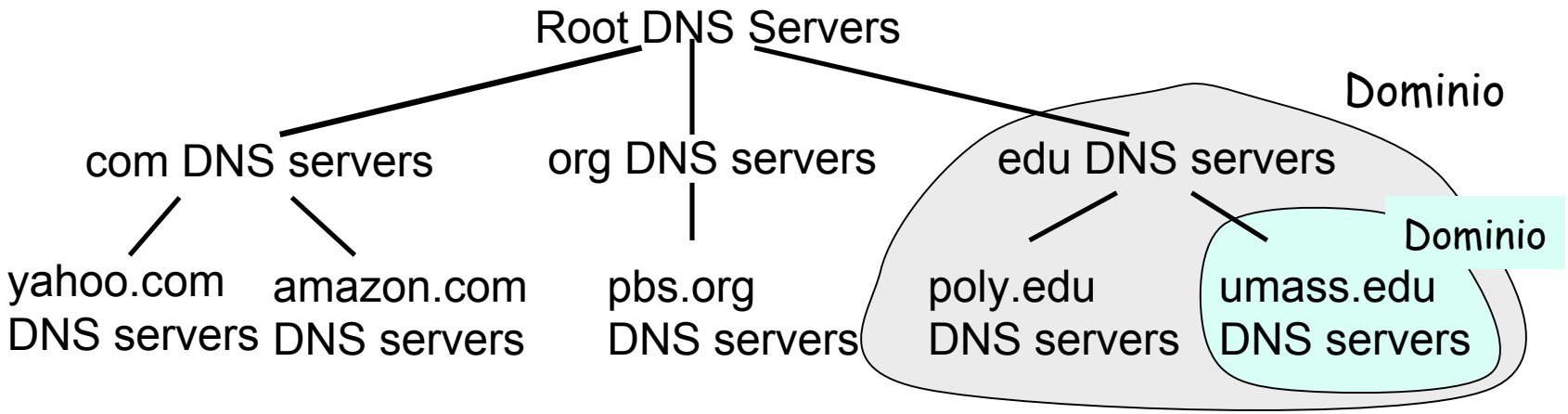
  `helios.tlm.unavarra.es`

  `www.google.com`

- Estructura jerárquica (. . .)



B.D. jerárquica distribuida



El cliente busca la IP de www.amazon.com:

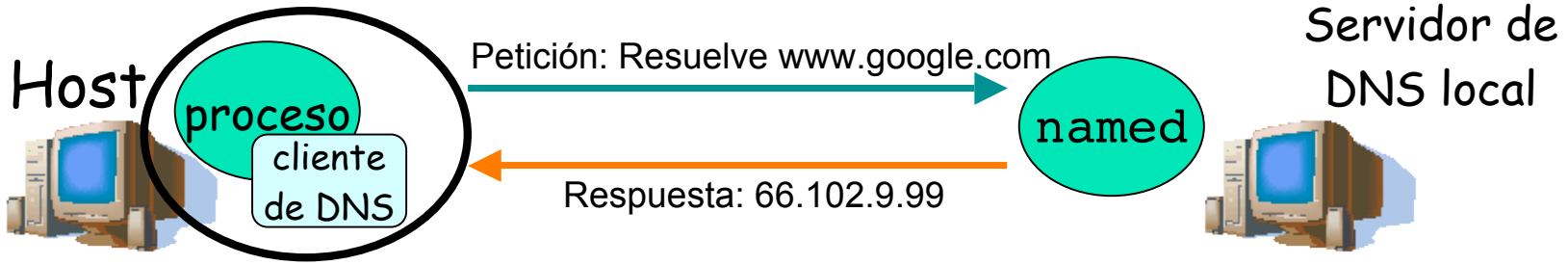
- El cliente pregunta a un ***servidor Root*** para encontrar el servidor de DNS del dominio *com*
- El cliente pregunta al *servidor del dominio* **com** para obtener el servidor del dominio **amazon.com**
- El cliente pregunta al servidor DNS del dominio **amazon.com** para obtener la IP de **www.amazon.com.**

Implementación

- El servidor es un programa específico pero el cliente es generalmente solo unas funciones en una librería (*resolver*) (…)
- La aplicación cliente de DNS es la propia aplicación del usuario (…)
- El software típico que lo implementa es BIND (Berkeley Internet Name Domain) (el programa servidor se llama `named`) (…)
- Emplea UDP (puerto servidor 53) o TCP si el mensaje de respuesta es de más de 512 Bytes.

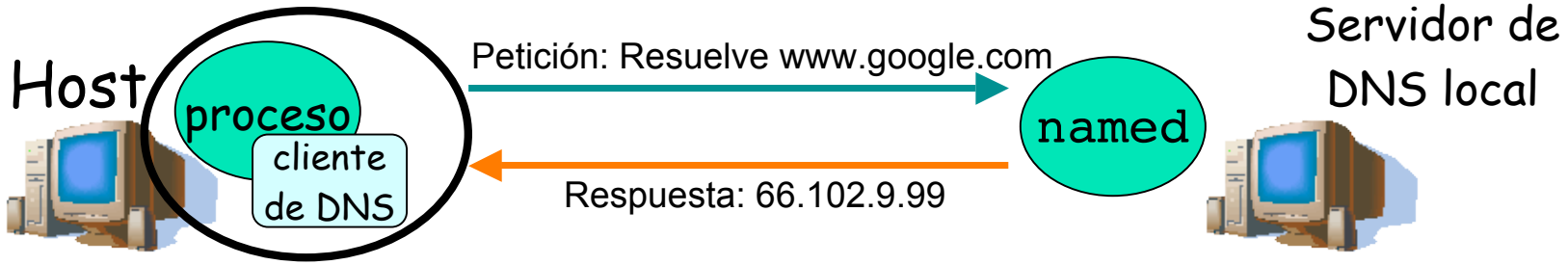
Funcionamiento

- Cada ISP posee un servidor de nombres local (…)
- Los hosts tienen configurado a su servidor local
- Cuando un host desea resolver un nombre hace la petición a su servidor local el cual le devuelve la respuesta (… …)



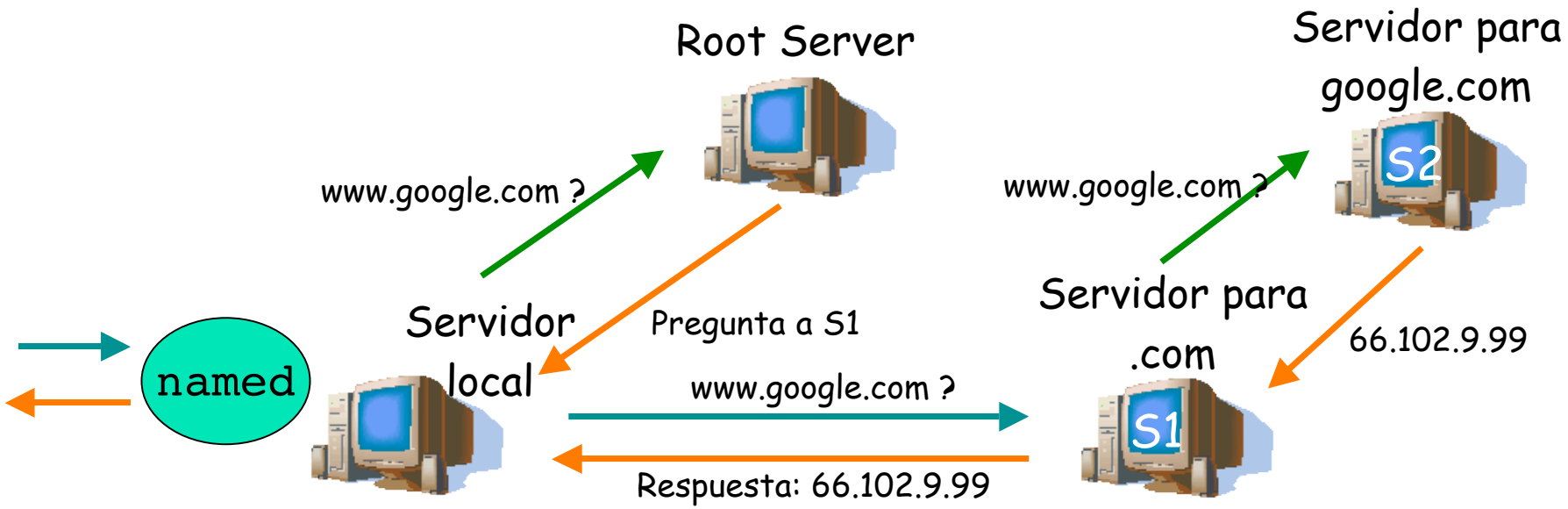
Funcionamiento

- ¿Cómo conoce la respuesta el servidor local?
  - Si es el servidor autoritario (***authoritative server***) para el dominio en el que está esa máquina él tiene la porción de la base de datos distribuida en la que está el mapeo (***zone file***)
  - Si no lo es preguntará a un ***Root Server***



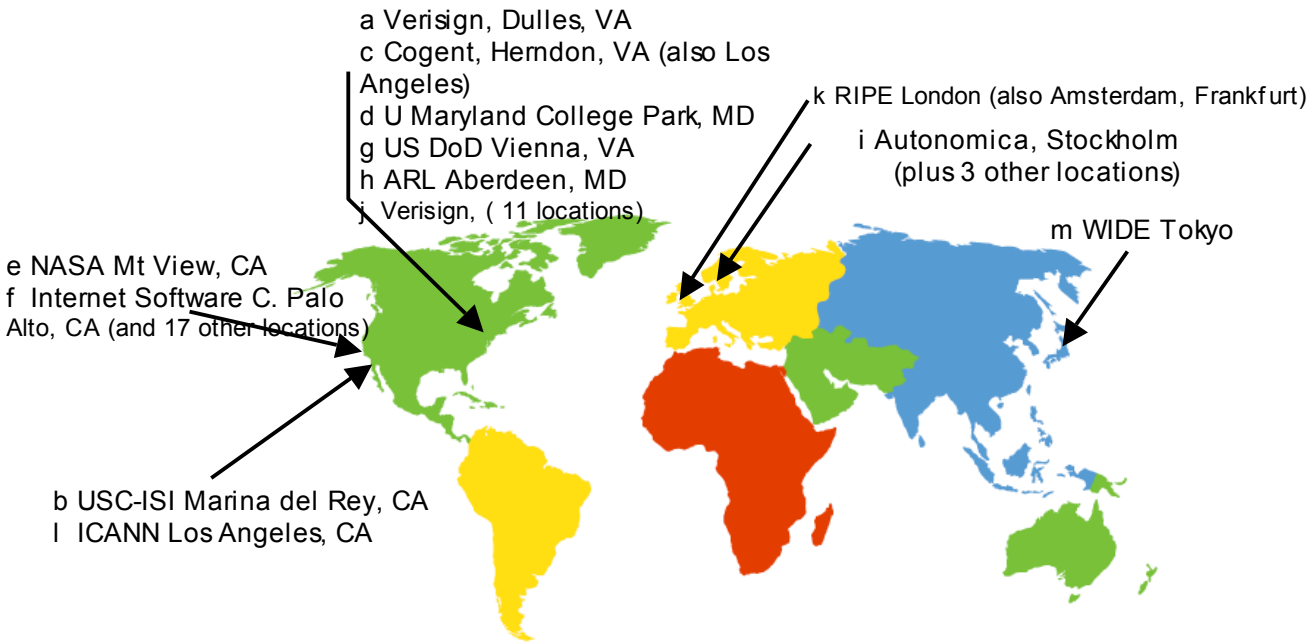
Funcionamiento

- El servidor local pregunta a un **Root Server** (…)
- Éste le devuelve la dirección de un servidor intermedio (petición **iterativa**) (…)
- El Servidor local hace una petición recursiva a ese servidor (…)
- Continuará haciendo la petición (**recursiva**) hasta que llegue un servidor autoritario (. . .)
- Todas las peticiones son recursivas menos la petición al Root Server para reducir la carga sobre los Root



DNS: Root name servers

- 13 en el mundo
- En el fichero de configuración de cada servidor de DNS



TLDs, Authoritative Servers, cache

Servidores *de Top-level domains* (TLD):

- Responsables de *com, org, net, edu*, (etc) y de los dominios raíz de países (*es, uk, fr, ca, jp, etc*)
- ESNIC para el TLD *.es* (http://www.nic.es)

Authoritative DNS servers:

- Servidores DNS de organizaciones
- Mantienen el mapeo autorizado para los nombres dentro del dominio de la organización

Fully Qualified Domain Name (FQDN)

- En realidad la raíz del árbol tiene también "nombre" pero es nulo
- Un FQDN incluye el nombre hasta la raíz, o sea, termina en un "."

  www.tlm.unavarra.es.

Una vez que un servidor de DNS aprende un mapeo lo cachea

- Las entradas en la cache caducan tras un tiempo
- Normalmente los servidores de los TLD van a estar cacheados en los servidores locales
  - Así que los Root no se suelen visitar

DNS records

DNS: Base de datos distribuida que almacena *resource records* (RR)

RR format: **(name, value, type, ttl)**

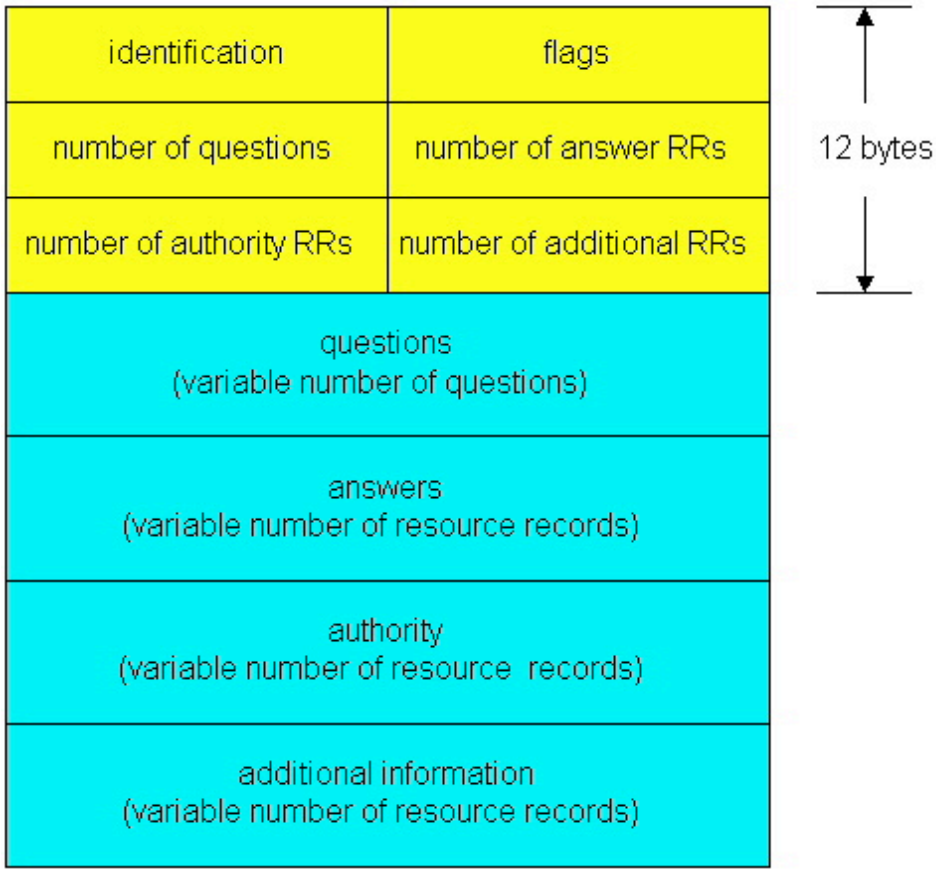
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "cannonical" (the real) name
    `www.ibm.com` is really
    `servereast.backup2.ibm.com`
  - **value** is cannonical name
- Type=MX
  - **value** is name of mailserver associated with **name**

Protocolo DNS, mensajes

Protocolo DNS: mensajes *query* y *reply*, mismo *formato de mensaje*

Cabecera

- Identificación: *reply* para un *query* usa el mismo valor
- flags:
  - query o reply
  - recursion desired
  - recursion available
  - reply is authoritative



Contenido

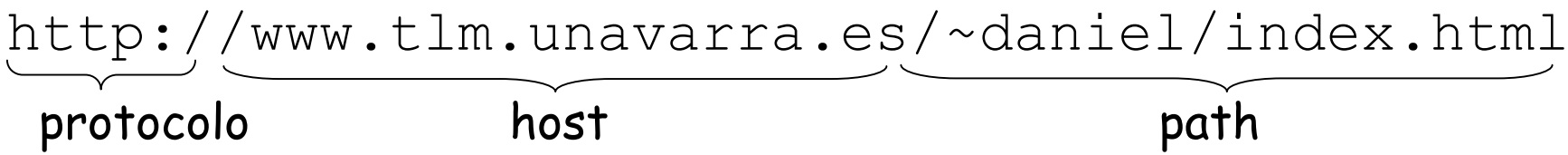
- Network Address Translation
- Nivel de Aplicación
  - E-Mail
  - DNS
  - **La Web**
    - **HTTP**
    - Procesado en el cliente y en el servidor
  - Seguridad

Referencias

- RFC 1945: Hypertext Transfer Protocol -- HTTP/1.0
- RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1
- RFC 2965: HTTP State Management Mechanism
- RFC 3986: Uniform Resource Identifiers (URI): Generic Syntax
- RFC 2246: The TLS Protocol Version 1.0

Web y HTTP: Terminología

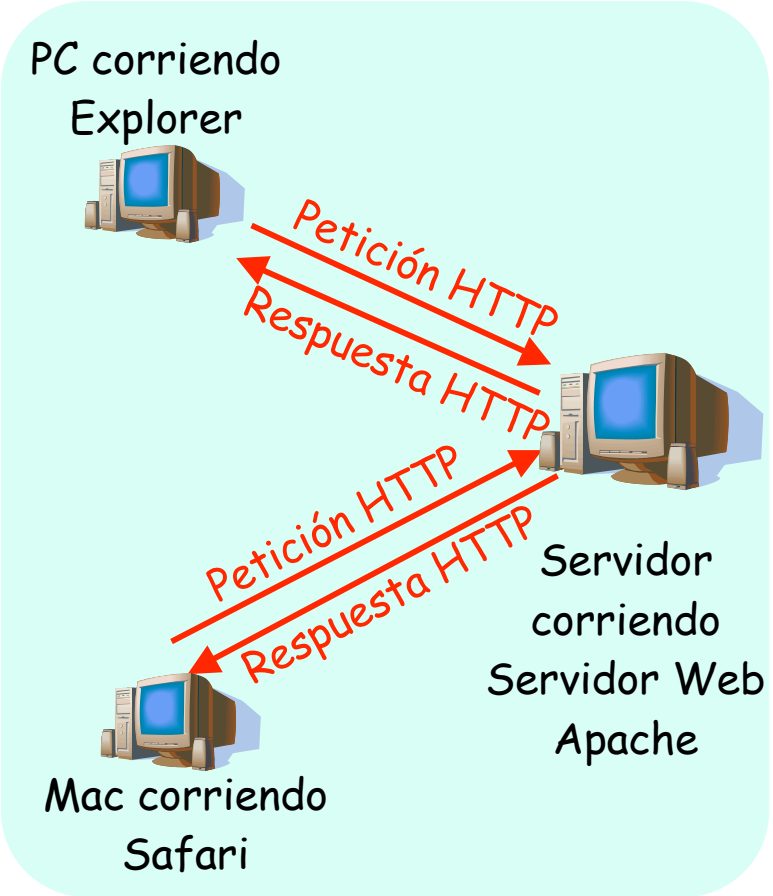
- Una **Página Web** está compuesta por **objetos**
- Un objeto puede ser un fichero HTML, una imagen JPEG, un applet JAVA, un fichero de sonido, etc
- La página Web está compuesta por un **fichero HTML base** que hace referencia a otros objetos
- Se hace referencia a cada objeto mediante un URI (Universal Resource Identifier)
- URLs subconjunto de URIs



HTTP

HTTP: HyperText Transfer Protocol

- Protocolo de nivel de aplicación de la Web
- Modelo cliente/servidor
  - cliente: browser (nevegador) que solicita, recibe y muestra objetos de la Web
  - servidor: el servidor Web envía objetos en respuesta a peticiones
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP

- Emplea TCP
- *Well known port*: 80
- Acciones típicas:
  - Cliente conecta con servidor
  - Solicita un objeto mediante su URI
  - Servidor envía el objeto y cierra la conexión
- HTTP es **sin estado**
- El servidor no mantiene ninguna información de peticiones anteriores del cliente
- Los protocolos sin estado son más simples

HTTP no persistente

- En cada conexión TCP se envía como máximo un objeto
- HTTP/1.0

HTTP no persistente

Supongamos que el usuario solicita el URL:

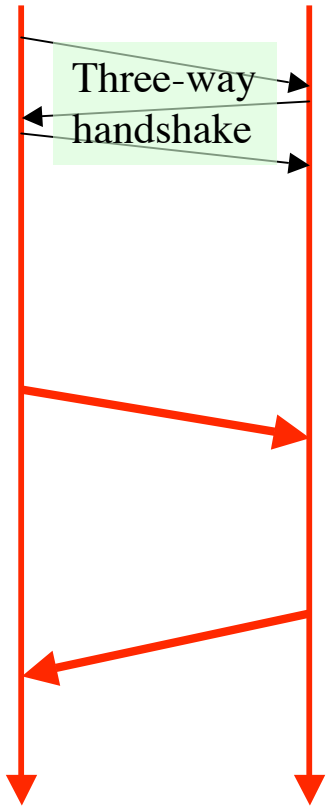
www.tlm.unavara.es/~daniel/index.html

(contiene texto y 1 referencia a una imagen JPEG)

1a: El cliente HTTP inicia la conexión TCP con el servidor en www.tlm.unavarra.es puerto 80

2: El cliente HTTP envía un mensaje de petición

El mensaje indica que el cliente quiere el objeto /~daniel/index.html



1b: El servidor acepta la conexión, notificando al cliente

3: El servidor HTTP recibe el mensaje de petición

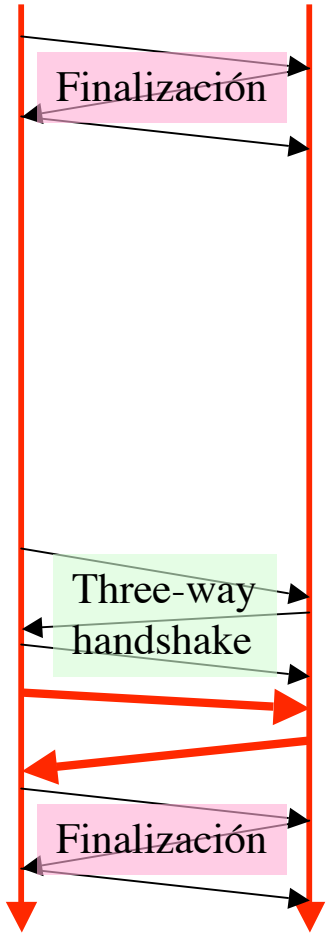
Forma un mensaje de respuesta que contiene el objeto solicitado y lo envía a través de su socket

HTTP no persistente

5: El cliente HTTP recibe el mensaje de respuesta que contiene el fichero HTML

Lo muestra y al interpretarlo encuentra la referencia a un objeto JPEG

6: Los pasos 1-5 se repiten para el objeto JPEG



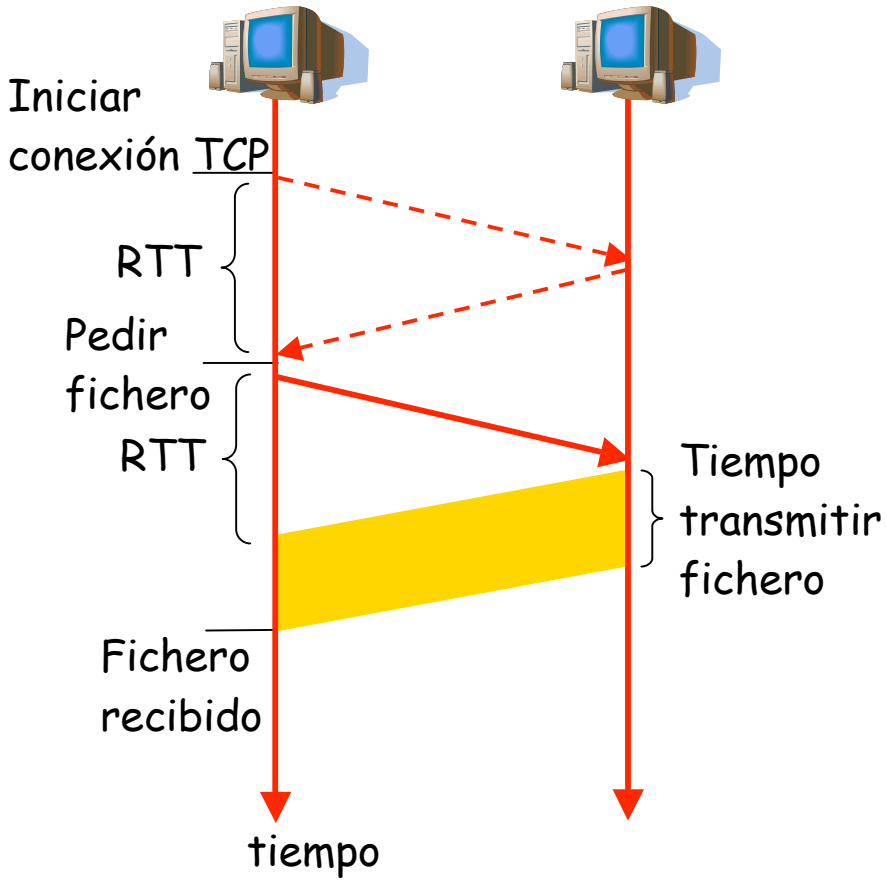
4: El servidor HTTP cierra la conexión TCP

Modelo del tiempo de respuesta

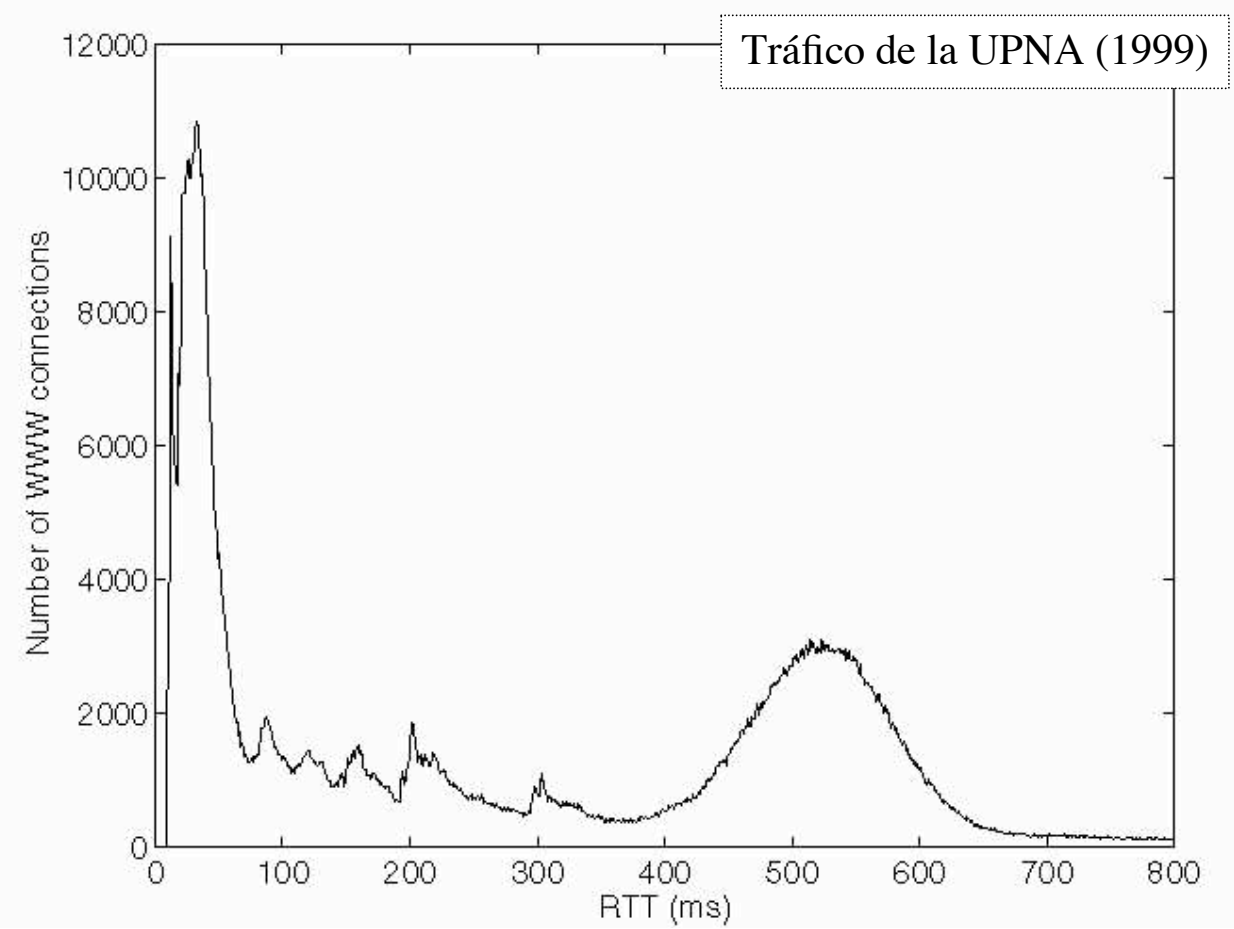
Tiempo de respuesta:

- Un RTT para iniciar la conexión
- Un RTT para la petición HTTP y el comienzo de la respuesta
- Tiempo de transmisión del fichero
- Mejor caso, ignorando mecanismos de TCP

total = 2xRTT + tiempo_transmisión



¿Cómo es el RTT?

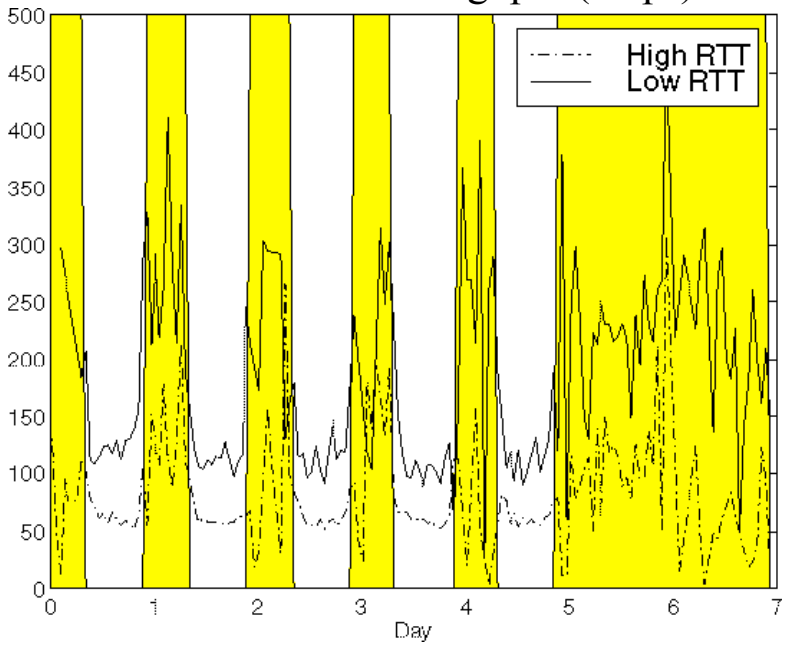


Throughput

Low RTT ρ ≈ 100Kbps

High RTT ρ ≈ 50Kbps

Data transfer throughput (Kbps)

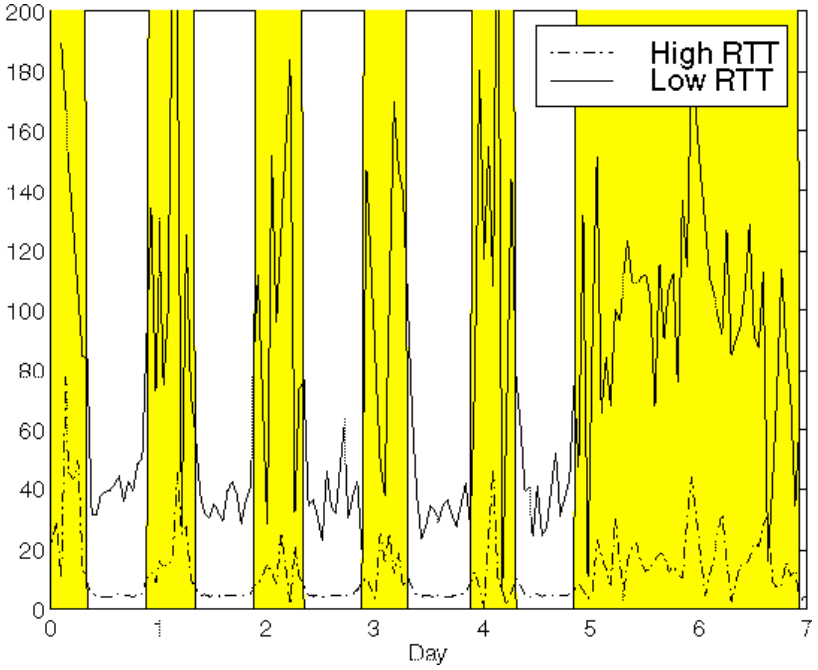


$$\frac{\text{Bytes transferidos}}{\text{Duración de la fase de transferencia}}$$

- Fuerte dependencia con RTT
- Fuerte dependencia con el tiempo de establecimiento de conexión TCP

$$\frac{\text{Bytes transferidos}}{\text{Duración total de la conexión}}$$

Total throughput (percieved by users) (Kbps)



Tráfico de la UPNA (1999)

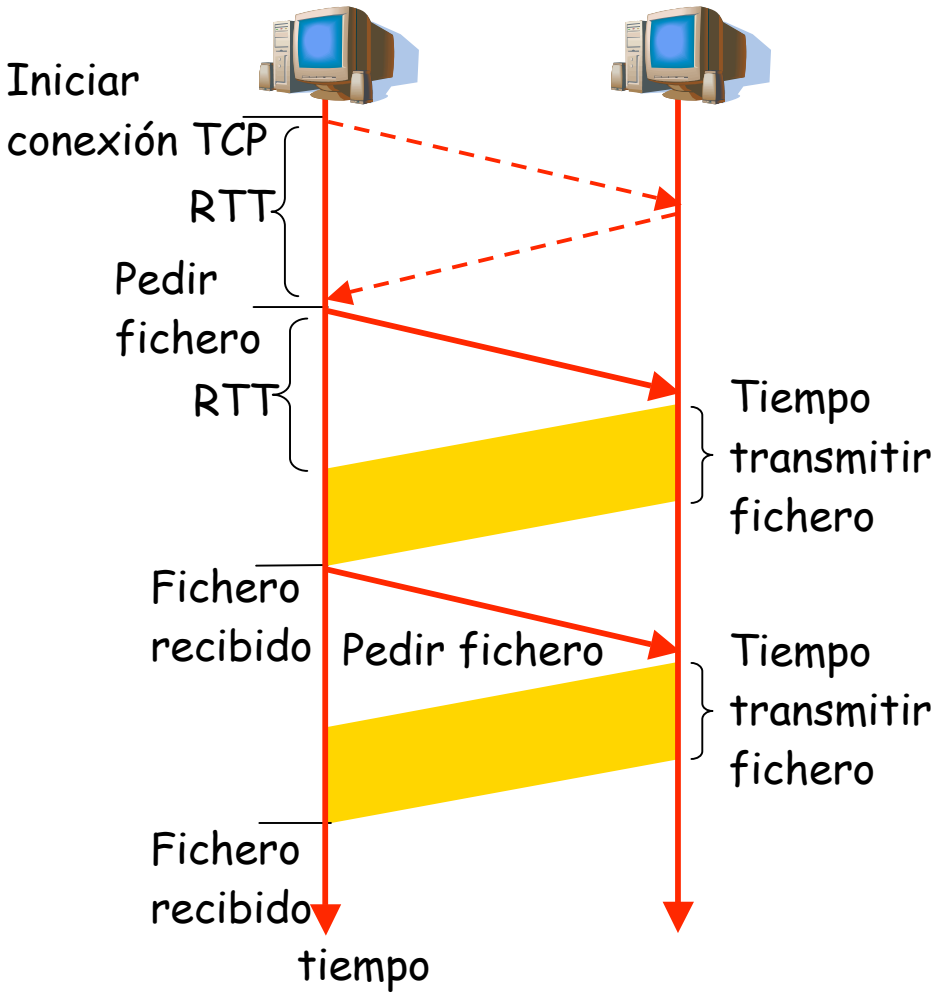
HTTP persistente

HTTP no persistente:

- Requiere 2 RTTs por objecto
- OS debe reservar recursos para cada conexión TCP
- Pero el navegador suele abrir varias conexiones TCP en paralelo

HTTP persistente:

- El servidor deja la conexión abierta tras enviar la respuesta
- Los siguientes mensajes HTTP entre cliente y servidor van por la misma conexión



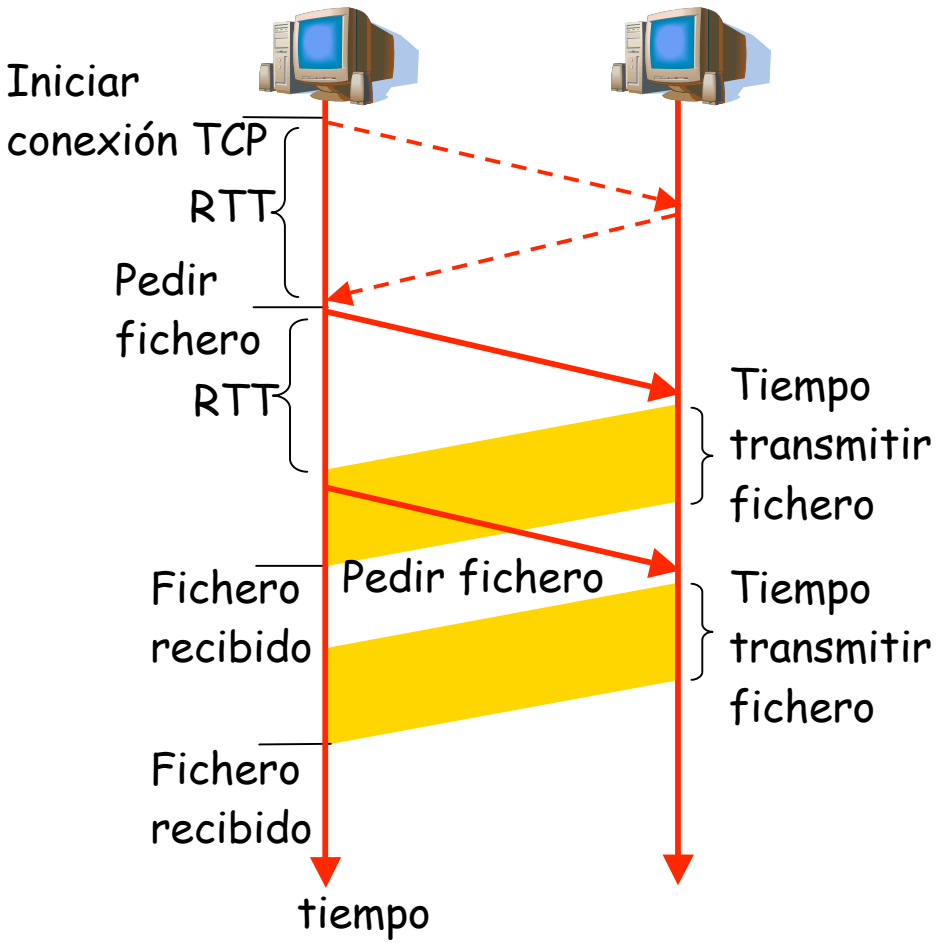
HTTP persistente

Persistente sin pipelining:

- El cliente manda la nueva petición cuando ha terminado de recibir la respuesta anterior
- Al menos un RTT por cada objeto

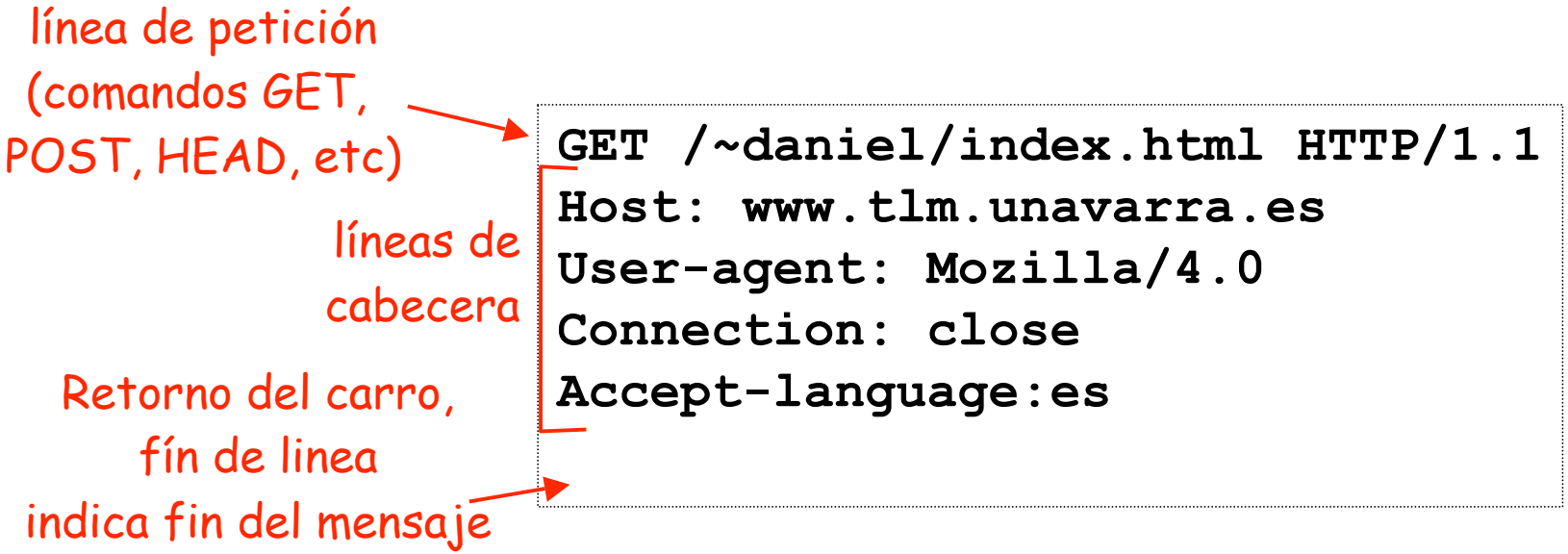
Persistente con *pipelining*:

- *default* en HTTP/1.1
- El cliente envía petición tan pronto como encuentra una referencia a objeto
- Solo un RTT para todos los objetos referenciados en la página base



HTTP request message

- Dos tipos de mensajes messages: request, response
- Mensaje HTTP request :
  - ASCII (formato legible por humanos)



Method types

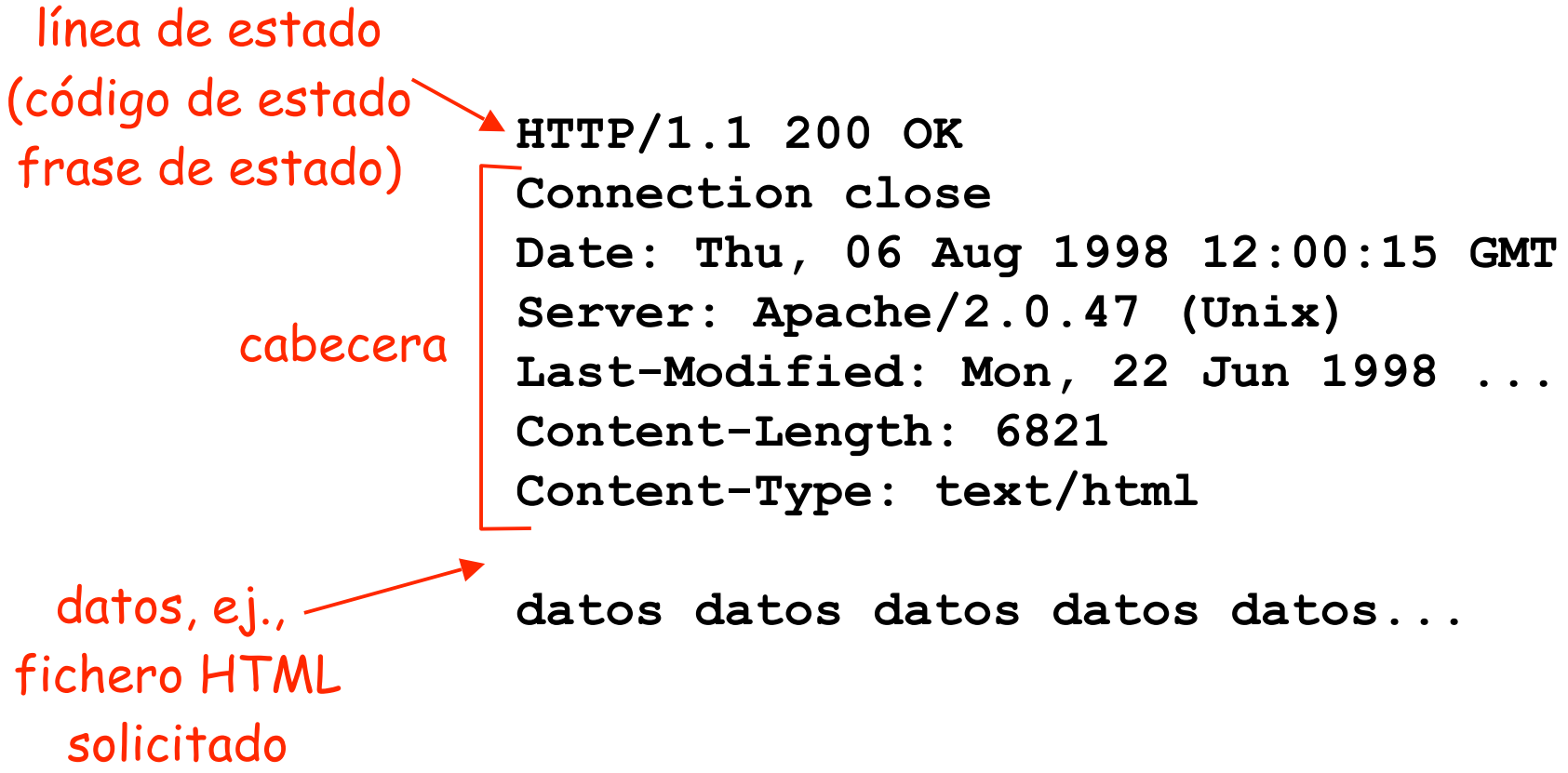
HTTP/1.0

- GET
  - Obtener la información identificada por el URI
  - *Conditional GET*
  - *Partial GET*
- HEAD
  - Como GET pero pide que se mande solo la cabecera HTTP y no el documento
- POST
  - Incluye un documento

HTTP/1.1

- GET, HEAD (obligatorios)
- POST
- PUT
  - Upload documento a un path especificado en el URI
- DELETE
  - Borra fichero especificado en el URI
- OPTIONS
  - Pide información sobre las opciones soportadas por el servidor para ese URI
- TRACE
  - Devuelve la petición hecha
- CONNECT
  - Proxy

HTTP response message



HTTP response status codes

1xx: Información

2xx: Éxito

3xx: Redirección

4xx: Client Error (bad syntax o no se puede servir)

5xx: Server Error

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Probando HTTP desde el cliente

1. Telnet a su servidor Web favorito:

```
telnet www.tlm.unavarra.es 80
```

Abre una conexión TCP al puerto 80 (puerto por defecto del servidor HTTP) de www.tlm.unavarra.es
Lo que se escriba se envía por la conexión TCP

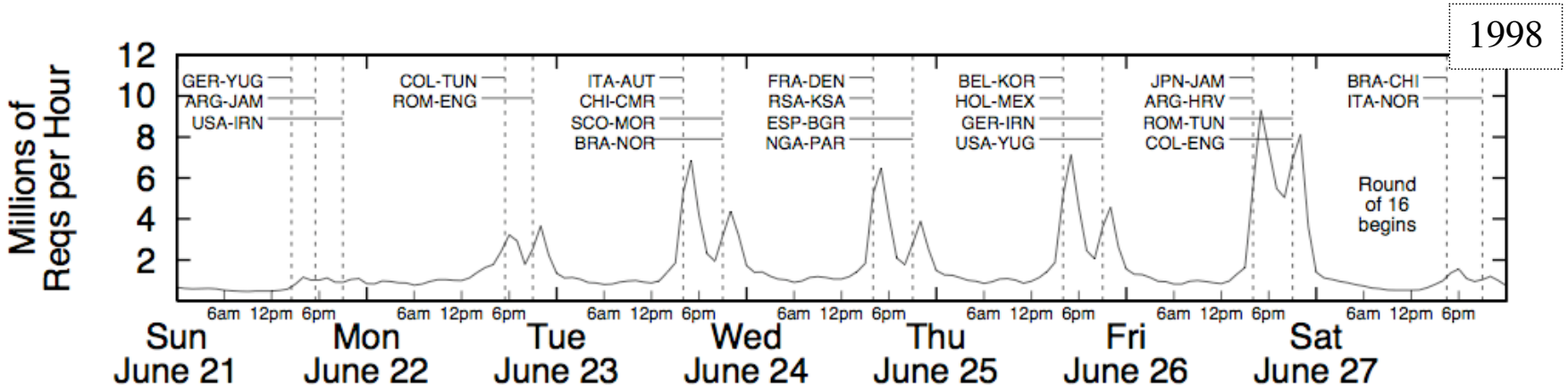
2. Escribir una petición GET de HTTP:

```
GET /~daniel/ HTTP/1.1
Host: www.tlm.unavarra.es
```

Escribiendo esto (y retorno del carro dos veces) se envía un petición HTTP 1.1 mínima pero completa al servidor

3. Vea el mensaje de respuesta del servidor

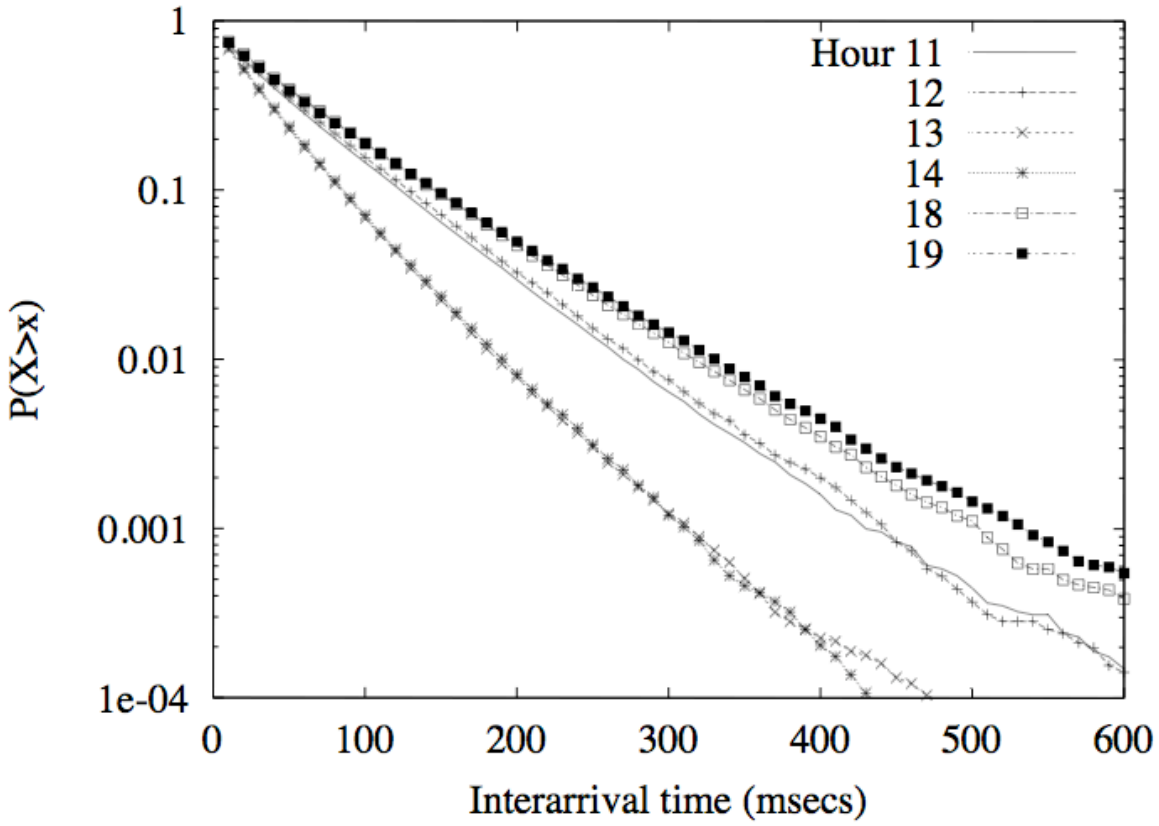
Dimensionamiento



- ¿Capacidad de proceso?
- ¿Velocidad del disco?
- ¿Carga en el enlace?

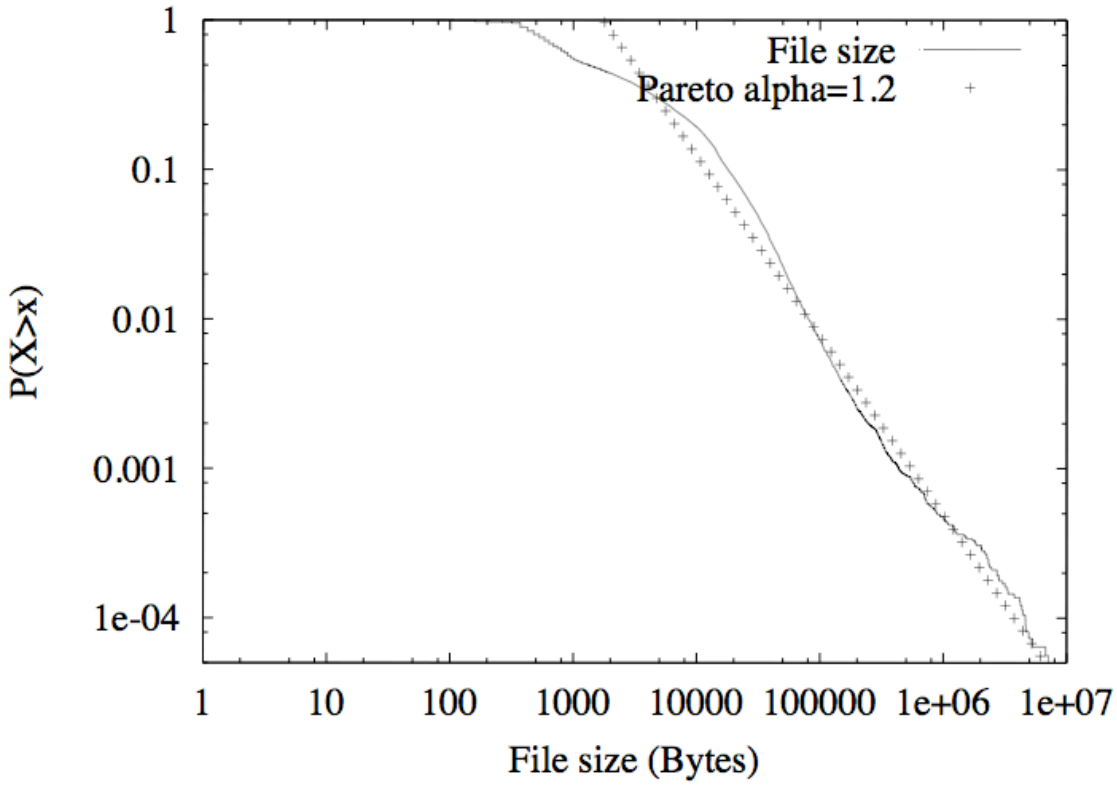
¿Modelos?

- ¿Llegadas de Poisson?



¿Modelos?

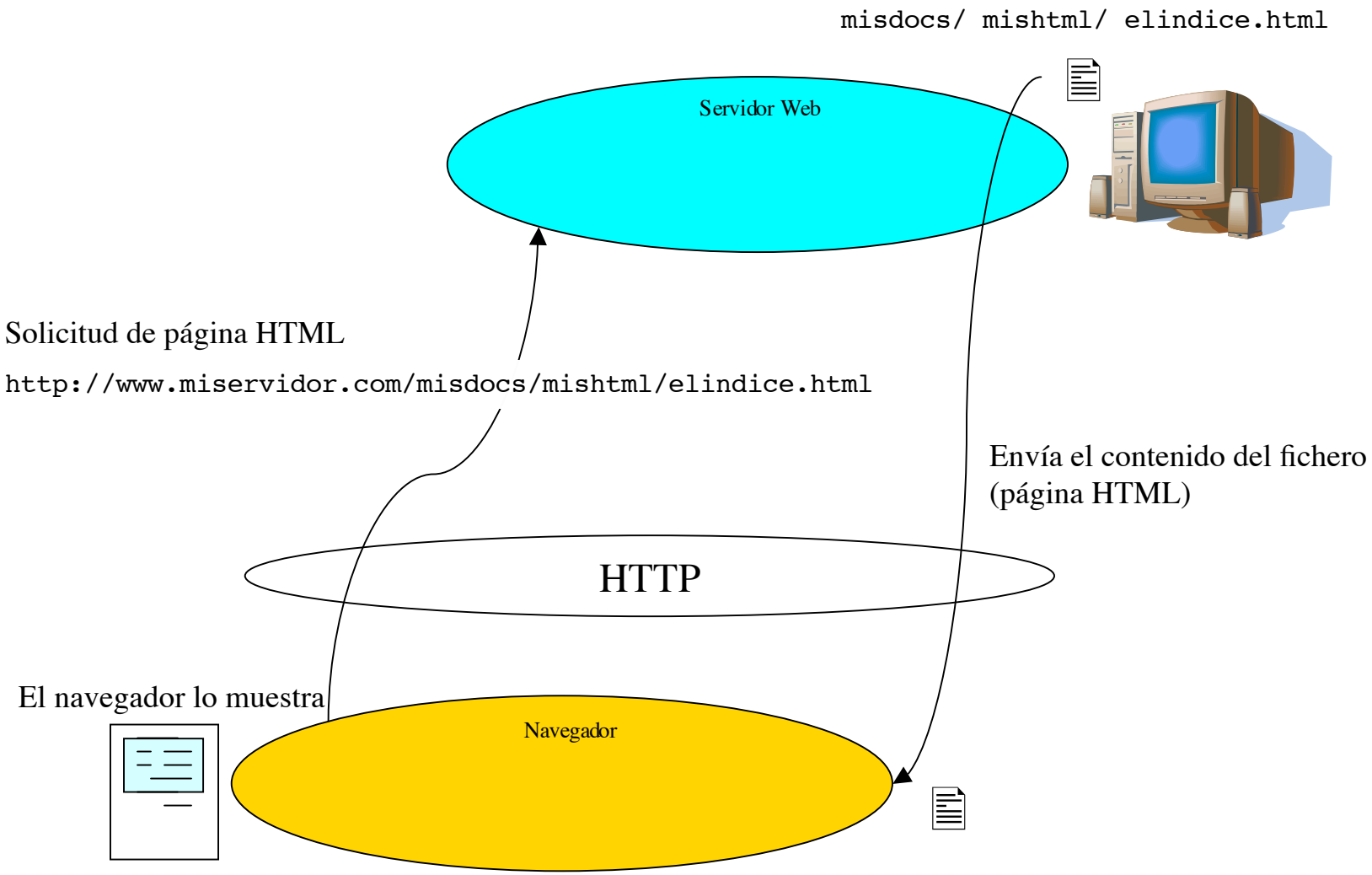
- ¿Tamaños de ficheros?



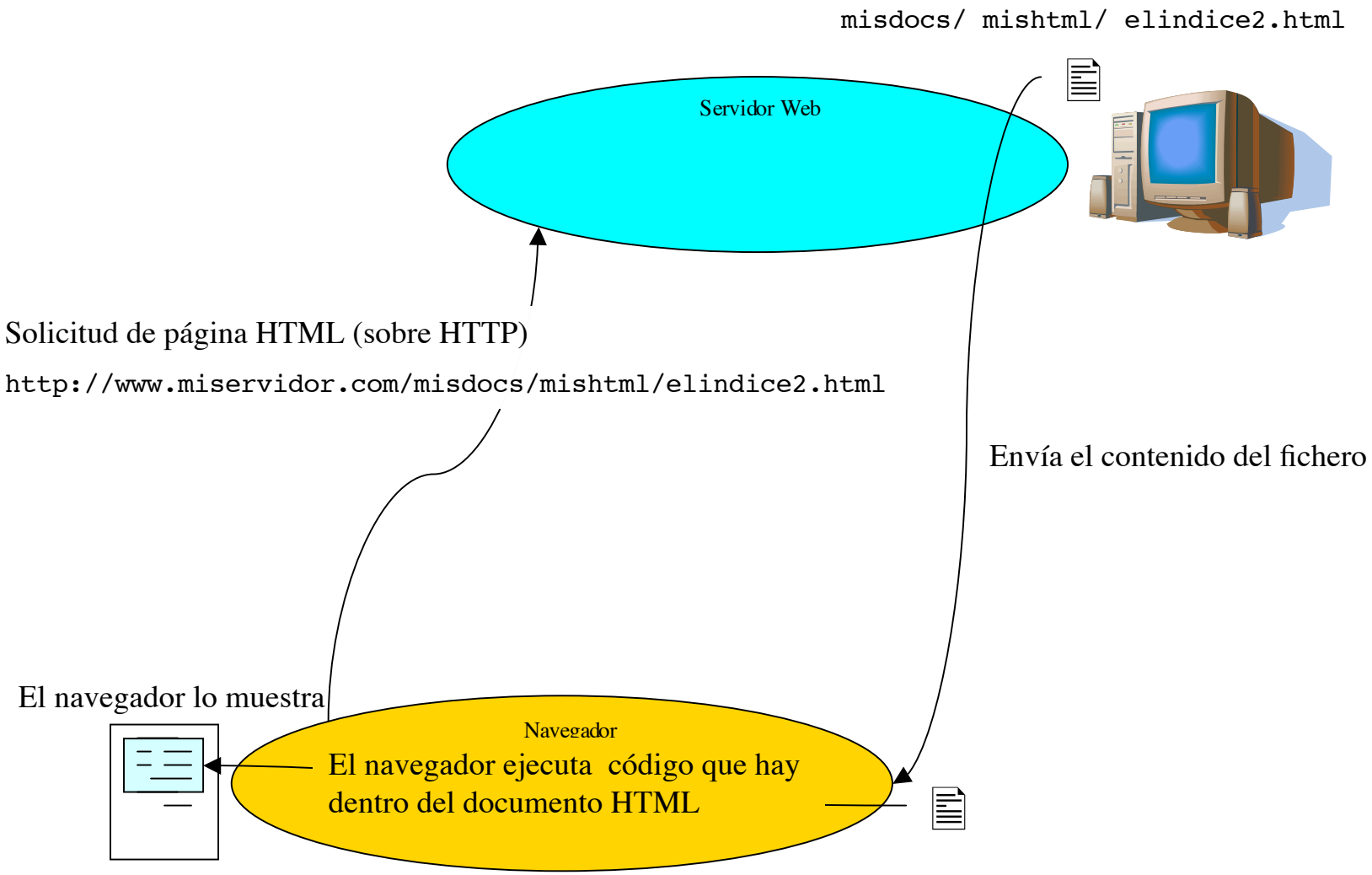
Contenido

- Network Address Translation
- Nivel de Aplicación
  - E-Mail
  - DNS
  - La Web
    - HTTP
    - **Procesado en el cliente y en el servidor**
  - Seguridad

HTML “estático”

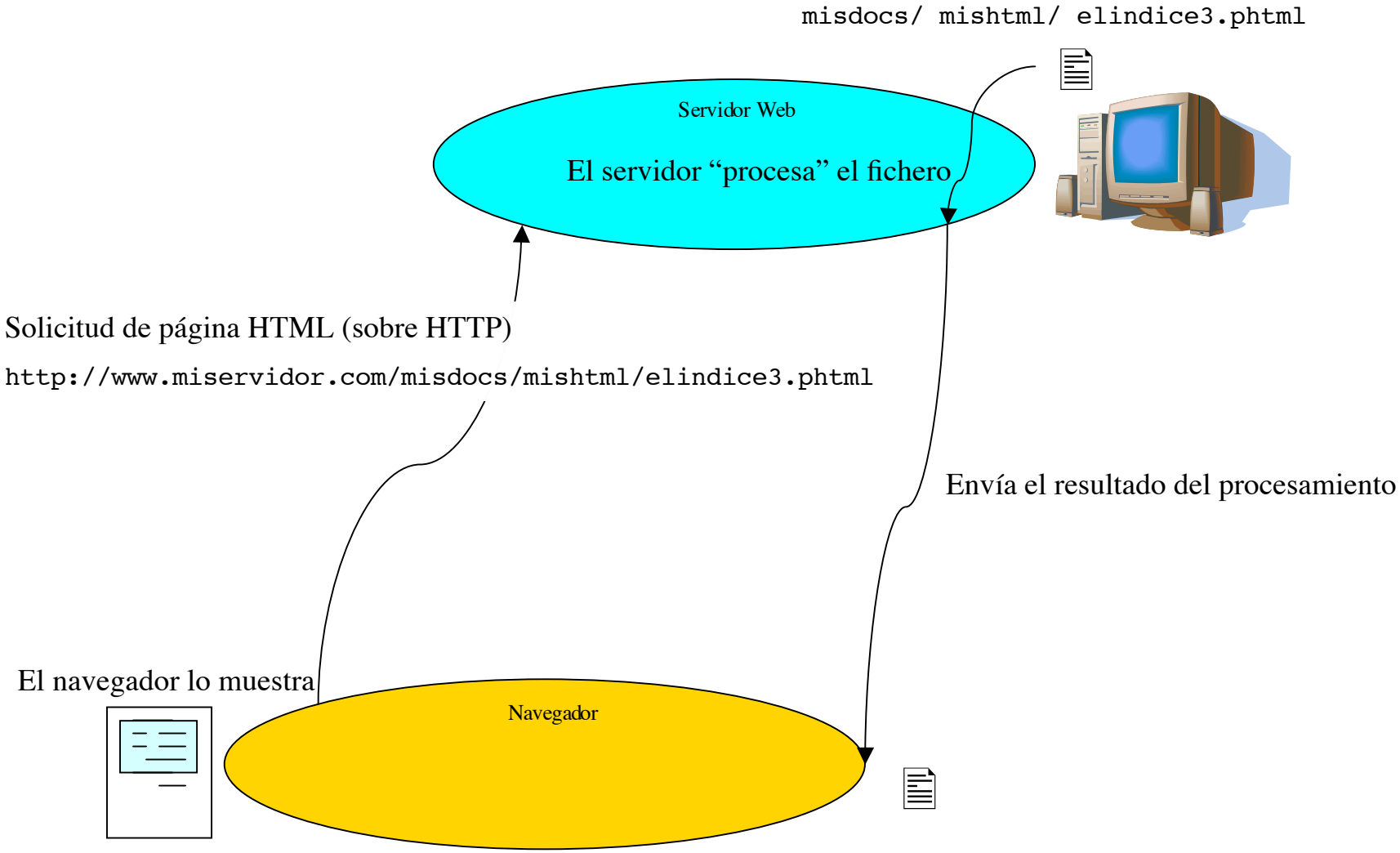


Procesado en el cliente



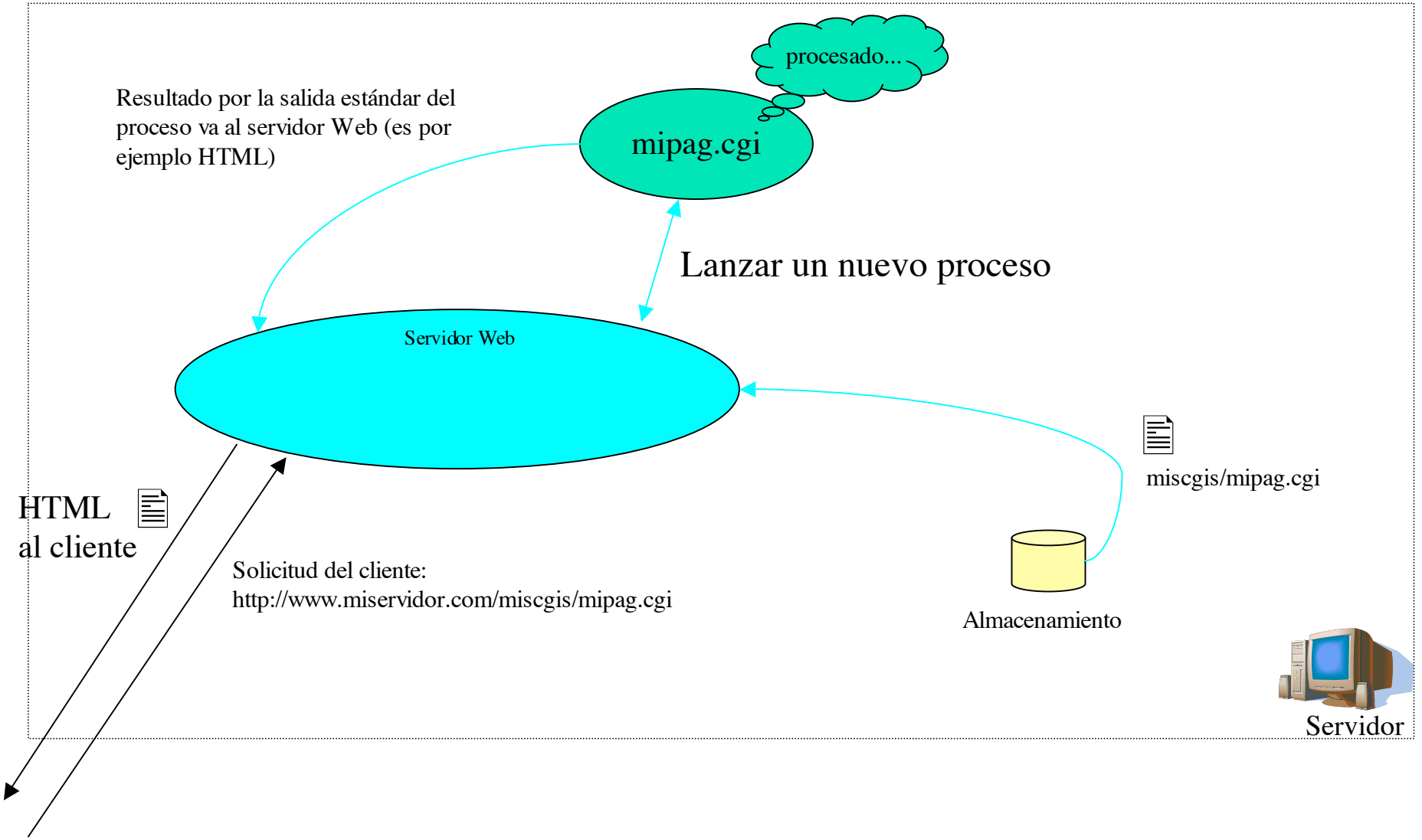
Ejemplo: JavaScript, VBScript

Procesado en el servidor



Ejemplo: PHP, CGIs, ASP

Ejemplo: procesado en servidor (CGI)



¿Qué significa “CGI”?

- CGI = “Common Gateway Interface”
- Define una forma de comunicarse con un servidor Web:
  - La forma de obtener del servidor Web información sobre las peticiones del cliente Web (ej: contenido de formularios)
  - La forma de darle al servidor Web un nuevo documento que enviar al cliente (ej: los bytes de una imagen o el código HTML de una nueva página Web)
- Los comúnmente llamados “CGIs” son simples programas preparados para ejecutarse en un sistema operativo en concreto.
- ¿En qué lenguaje hay que escribirlos? En casi cualquiera: C, C++, Pascal, Fortran, Perl, script de una shell, Python, Tcl/Tk, AppleScript, Visual Basic...
- ¿Qué se debe poder hacer para comunicarse con el servidor Web?:
  - Para recibir información del servidor Web debemos: poder leer de la entrada estándar y leer el contenido de las variables de entorno
  - Para mandar información al servidor Web debemos: poder escribir por la salida estándar

¿Qué lenguaje emplear?

- Tenemos dos tipos de lenguajes para elegir:
  - Lenguajes compilados
    - C, C++, Pascal...
    - Hay un código fuente que es compilado en un fichero ejecutable en el lenguaje de la máquina en que vaya a ejecutarse
    - Se puede distribuir solo el ejecutable con lo que se mantiene control sobre el código
    - Pero hay que compilar el código para la combinación procesador+S.O. en concreto en que se vaya a ejecutar
    - Generalmente son programas más pequeños y rápidos
  - Lenguajes interpretados
    - Perl, Tcl/Tk, Python, bash...
    - El código del programa es interpretado por otro programa. No se compila sino que un programa compilado (el intérprete) ejecuta las instrucciones que indica el "script"
    - Siempre que sobre esa plataforma exista el intérprete se puede ejecutar el script sin cambios
    - Generalmente son más lentos porque requieren ejecutar primero el programa intérprete y luego éste es más lento ejecutando las instrucciones que si fuera código compilado
    - Suelen ser lenguajes más sencillos de programar y depurar

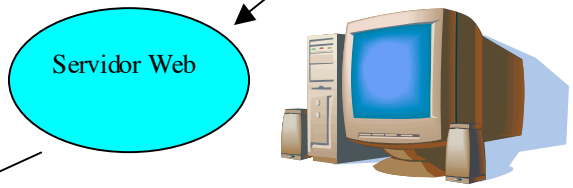
Ejemplo simple

micgi1.c

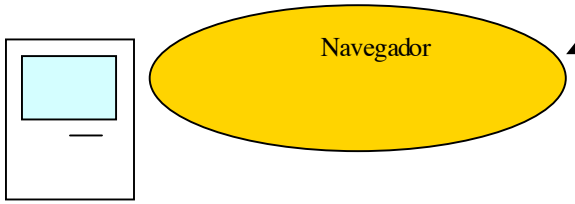
```
#include <stdio.h>

main()
{
 printf("Content-Type: text/html\r\n\r\n");
 printf("<html><head><title>Primer
CGI</title></head>\n");
 printf("<body><h1>Mi primera pagina web
resultado de un CGI en C</h1>\n");
 printf("<p>Casi nada!\n</body></html>\n");
}
```

gcc micgi1.c -o micgi1.cgi

micgi1.cgi



http://www.miservidor.com/miscgis/micgi1.cgi




```
HTTP/1.1 200 OK
Date: Mon, 06 Oct 2003 19:36:42 GMT
Server: Apache/1.3.27 (Darwin)
PHP/4.3.0
Connection: close
Content-Type: text/html

<html><head><title>Primer
CGI</title></head>
<body><h1>Mi primera pagina web
resultado de un CGI en C</h1>
<p>Casi nada!
</body></html>
```

Ejemplo simple

micgi1.c

```
#include

main()
{
 printf
 printf
CGI</ti
 printf
resulta
 printf
}
```

http://www.mi iscgis/micgi1.cgi

Navegador

```
HTTP/1.1 200 OK
Date: Mon, 06 Oct 2003 19:36:42 GMT
Server: Apache/1.3.27 (Darwin)
PHP/4.3.0
Connection: close
Content-Type: text/html

<html><head><title>Primer
CGI</title></head>
<body><h1>Mi primera pagina web
resultado de un CGI en C</h1>
<p>Casi nada!
</body></html>
```

Análisis del ejemplo

• Lo que el programa (CGI) envía a la salida estándar (`stdout`) llega al navegador a través del servidor Web

• Lo que envía el CGI tiene una primera parte que no muestra el navegador:

```
HTTP/1.1 200 OK
Date: Mon, 06 Oct 2003 19:36:42 GMT
Server: Apache/1.3.27 (Darwin) PHP/4.3.0
Connection: close
Content-Type: text/html
```

- Es la cabecera HTTP (el protocolo empleado en la conexión TCP). Va antes del documento enviado y separado por una línea en blanco.
- Esta cabecera tiene muchas opciones. `Content-Type` es lo mínimo de la cabecera HTTP que debe construir el CGI, el resto lo construirá el servidor Web (si el script construye toda la cabecera HTTP se llama script NPH=Non-Parsed Header)
- El `Content-Type` especifica el tipo de documento que se está enviando, en este caso una pagina HTML. El formato es:

  "Content-Type: Tipo/Subtipo" , donde Tipo/Subtipo hacen referencia a un tipo MIME (Multipurpose Internet Mail Extensions, RFC 1521).

  Algunos tipos son: `image/gif, video/mpeg, application/pdf, image/jpeg,` etc.
- Las líneas en la cabecera HTTP y la línea en blanco deben estar terminadas por un retorno del carro (`\r` en C) y un fin de línea (`\n` en C) aunque con muchos servidores Web funcione el emplear solo el fin de línea.
- Tras la cabecera HTTP va el contenido del documento, del tipo que se ha especificado. En el ejemplo es un texto (un documento en HTML) pero podría ser por ejemplo una imagen en cuyo caso ahora se escribirían los datos binarios que forman la imagen.

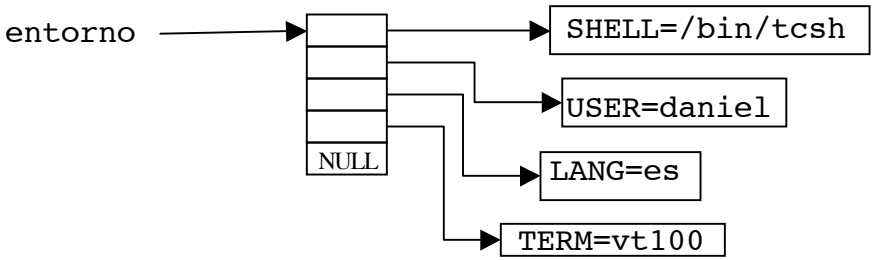
• El CGI tiene un nombre terminado en `.cgi` lo cual no es imprescindible. Al servidor Web se le configura qué ficheros, cuando se soliciten, son ejecutables CGI que debe lanzar en otro proceso. En este caso el servidor se ha configurado para reconocer estos ficheros por la extensión `.cgi`

Datos de entrada para un CGI

(Variables de entorno)

• El CGI puede acceder a datos generales sobre el servidor Web, el navegador y la petición. En versiones UNIX se hace mediante variables de entorno:

- Las llamadas " variables de entorno" son un conjunto de cadenas accesibles por el programa
- En las Shells se puede dar valor y consultar dichas variables. Se heredan al crear nuevos procesos con `fork()` (logicamente) y se pueden especificar para un nuevo proceso al ejecutar un programa con `execve()`
- En C estas cadenas están en un array de cadenas:



- En C se puede conseguir este array de dos formas:
  Está en el tercer argumento de la función `main()` : `main(int numero_args, char *args_en_linea[], char *entorno[]);`
  Hay una variable global con ese valor con nombre `environ` que se puede declarar como: `extern char **environ;`

- La forma de modificarlo dependerá del lenguaje en que se escriba el CGI: En C existen unas funciones muy útiles (`getenv()`, `setenv()`...)

- Algunas variables de entorno que crea el servidor Web:

`REMOTE_ADDR`	Dirección IP del cliente (el navegador)
`HTTP_ACCEPT`	Lista de tipos MIME que acepta el navegador
`HTTP_USER_AGENT`	Descripción del navegador (nombre, versión, sistema operativo...)
`SERVER_PORT`	Puerto por el que aceptó la conexión el servidor Web
`SERVER_SOFTWARE`	Nombre y versión del servidor web

Datos de entrada para un CGI

(Información de formularios)

• El CGI puede acceder a información introducida por el usuario en un formulario. La información puede venir de dos formas diferentes:

- Método `GET`:

  Se envía como parte del URL. Ejemplo: `http://myserver.org/cgi-bin/procesa.cgi?nombre=John+Smith&edad=54`

  El servidor Web se lo entega al CGI dentro de la variable `QUERY_STRING`

  Problemas:

  • El tamaño máximo suele estar limitado

  • Se ve el contenido del formulario en el URL

- Método `POST`:

  Se envía en la cabecera HTTP

  El CGI tiene acceso a él a través de la entrada estándar (se lo envía el servidor Web)

  En la variable `CONTENT_LENGTH` se especifica la longitud en bytes de lo que se puede leer por `stdin`

• Se especifica que se esté empleando un método u otro en la variable de entorno `REQUEST_METHOD` que valdrá `GET` o `POST` y proviene de que en el formulario se especifique el valor del atributo `method` del tag `FORM` como uno u otro

• El navegador codifica el contenido del formulario antes de enviarlo. Se llama “URL encoding” (RFC 1738):

- Los diferentes campos del formulario se separan con un ampersand (&)

- Se coloca nombre y valor de cada campo donde el nombre es el valor del atributo name y el valor depende del tipo de elemento

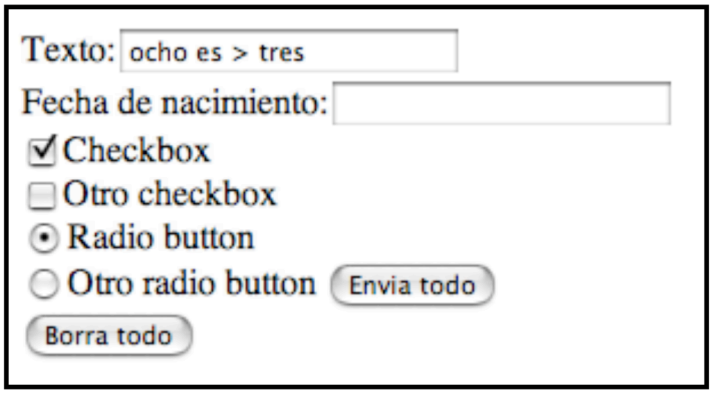
- Los espacios se cambian por el signo +

- Los caracteres “ extraños” (generalmente que no están en el US-ACSII o que sea un carácter reservado) aparecen como un signo de porcentaje seguido de un código hexadecimal

Datos de entrada para el CGI

(URL encoding: Ejemplo)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html><head><title>Controles</title></head><body>
<form action="http://localhost/~daniel/miscgis/cgi2.cgi"
method="get">
Texto:<input type="text" name="cadena" value="Texto por
defecto"><br>
Fecha de nacimiento:<input type="text" name="fecha"><br>
<input type="checkbox" checked name="check1">Checkbox<br>
<input type="checkbox" checked name="check2">Otro checkbox<br>
<input type="radio" name="miradio" value="1" checked>Radio
button<br>
<input type="radio" name="miradio" value="2">Otro radio button
<input type="submit" value="Envia todo"><br>
<input type="reset" value="Borra todo">
</form></body></html>
```




```
QUERY_STRING=cadena=ocho+es+%3E+tres&fecha=&check1=on&miradio=1
```

Redirección

- La cabecera HTTP de respuesta del CGI en vez de incluir un "`Content-Type`" puede lleva un "`Location`"
- Le indica al servidor que el CGI devuelve una referencia a un documento en vez del documento
- Se indica el camino al documento:
  - Si se indica como un camino relativo al disco local el servidor enviará ese fichero al cliente sin que éste note diferencia con que lo hubiera pedido directamente
  - Si se indica un URI a otro servidor le llegará al cliente una indicación de que debe dirigirse a ese otro documento en ese otro servidor

```
printf("Location: /~daniel/index.html\n\n");
```

```
HTTP/1.1 200 OK
Date: Tue, 07 Oct 2003 19:40:38 GMT
Server: Apache/1.3.27 (Darwin) PHP/4.3.0
Last-Modified: Tue, 17 Jun 2003 15:06:29 GMT
ETag: "2a-1fad-3eef2e75"
Accept-Ranges: bytes
Content-Length: 8109
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.or<html>99<head>ht<meta http-
equiv="content-type"
content="text/html;charset=iso-8
```

```
printf("Location:
http://www.tlm.unavarra.es/~daniel/docencia/l
pr/lpr03_04\n\n");
```

```
HTTP/1.1 301 Moved Permanently
Date: Tue, 07 Oct 2003 19:37:28 GMT
Server: Apache/1.3.22 (Unix)  (Red-Hat/Linux) PHP/3.0.15
mod_perl/1.21
Location: http://www.tlm.unavarra.es/asignaturas/lpr/
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD><TITLE>301 Moved Permanently</TITLE></HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A
HREF="http://www.tlm.unavarra.es/asignaturas/lpr/">here</A>.<P><
HR>
<ADDRESS>Apache/1.3.22 Server at www.tlm.unavarra.es Port
80</ADDRESS>
</BODY></HTML>
```

Server-Side Includes (SSI)

- Podemos indicarle al servidor Web una parte de un documento HTML que debe ser substituido por el resultado de algún comando o CGI.
- El servidor web busca en la página web esos tags especiales, ejecuta lo que indican y coloca el resultado en el documento que envía al navegador.
- En la página que recibe el navegador no queda rastro de ese CGI incluido en la página.
- Apache emplea la etiqueta de comentario de HTML para marcar estos SSIs, de forma que si están desactivados no confundan al cliente. Ejemplo:

Documento `.shtml`

```
<html>
<head><title>SSI 1</title></head>
<body>
<h1>SSI 1</h1>
<p>
Bla bla bla
<!--#exec cmd="uncgi.cgi" -->
</body>
</html>
```

`uncgi.cgi`

```
#!/bin/sh
echo "<hr><p>Soy el resultado de un SSI"
echo -n "<p>Y hoy es "
date
echo "<br><hr>"
```

HTML recibido por el navegador

```
<html>
<head><title>SSI 1</title></head>
<body>
<h1>SSI 1</h1>
<p>
Bla bla bla
<hr><p>Soy el resultado de un SSI
<p>Y hoy es Thu Oct  9 13:48:00 CEST 2003
<br><hr>

</body>
</html>
```

PHP

- PHP = “PHP Hypertext Preprocessor”
- Lenguaje de scripts para el servidor Web (server-side processing)
- Open Source
- Puede ir en el mismo documento que el código HTML
- Simple para el principiante
- Con muchas características avanzadas
- Soportado en gran número de sistemas operativos: variantes de UNIX (Linux, HP-UX, Solaris, OpenBSD), Microsoft Windows, Mac OS X
- Soporta la mayoría de servidores web: Apache, Microsoft IIS, Personal Web Server, iPlanet, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, etc
- Permite no solo generar HTML sino también imágenes, PDFs, Flash, XML
- Soporta un gran número de bases de datos: Orable, mSQL, MySQL, ODBC, Sybase, etc
- Soporta un gran número de servicios y protocolos: LDAP, IMAP, SNMP, NNTP, POP3, HTTP, etc

¿Puede ir en el documento HTML?

• Dentro del documento HTML podemos emplear tags especiales que indican que lo que va entre ellos es código PHP. Ejemplo:

ejemplo.php

```
<html>
  <head><title>Script de ejemplo</title></head>
  <body>
    <h1>Pagina simple</h1>
    <p>Aqui el codigo HTML
    <?php
      echo "<p>Y esto sale del codigo PHP";
    ?>
    <p>Has visto el parrafo anterior?
  </body>
</html>
```

• El servidor reconoce que el fichero puede contener código PHP generalmente por la extensión del fichero

• El servidor busca en el documento los tags que marcan el código PHP. Lo ejecuta y si el script quiere escribir texto (`print()`) ese texto aparece donde estaba el código PHP al enviarse el documento (no se cambia el fichero)

¿Puede ir en el documento HTML?

• Dentro del documento HTML podemos emplear tags especiales que indican que lo que va entre ellos es código PHP. Ejemplo:

ejemplo.php

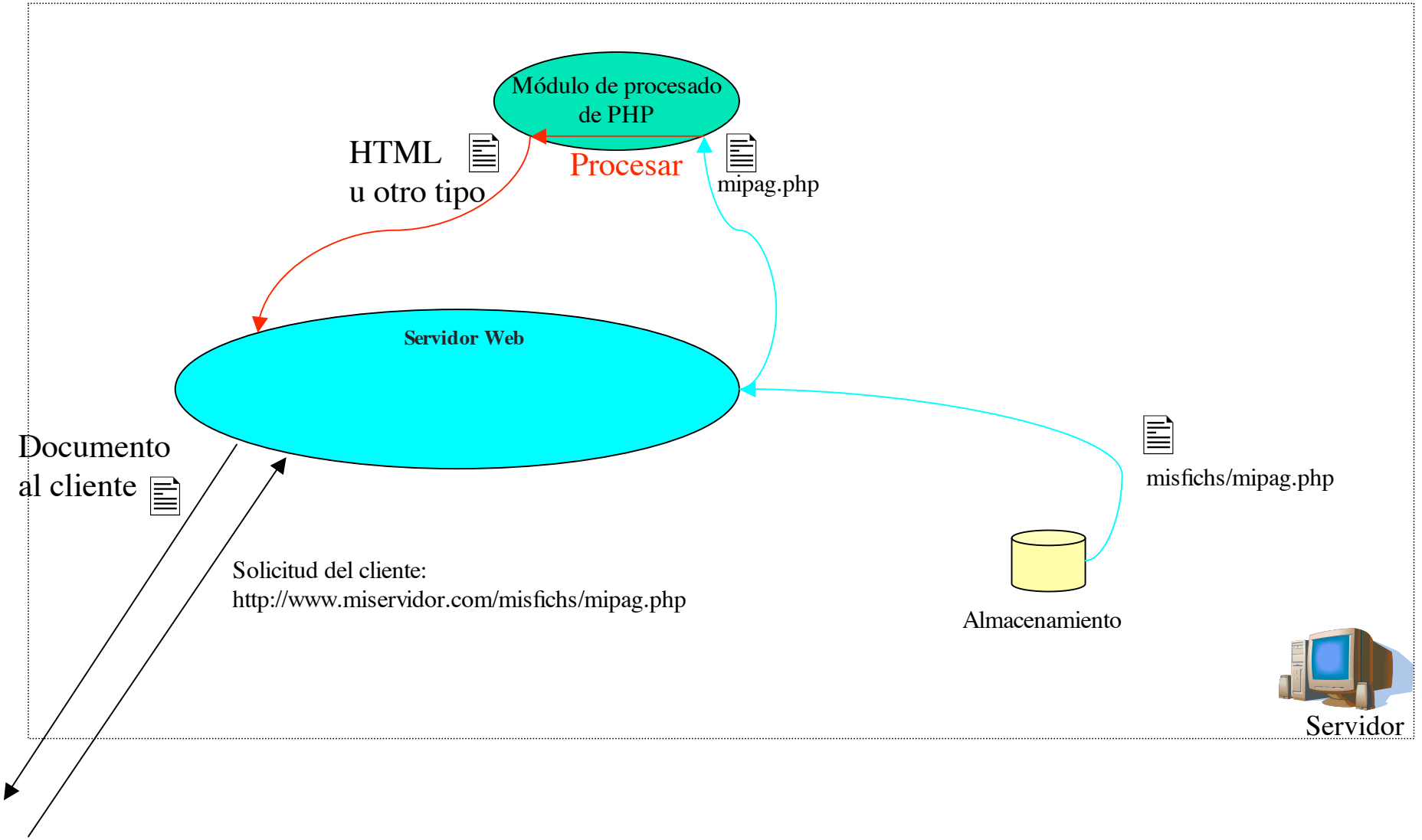
```
<html>
  <head><title>Script de ejemplo</title></head>
  <body>
    <h1>Pagina simple</h1>
    <p>Aqui el codigo HTML

<p>Y esto sale del codigo PHP

    <p>Has visto el parrafo anterior?
  </body>
</html>
```

Resultado del PHP

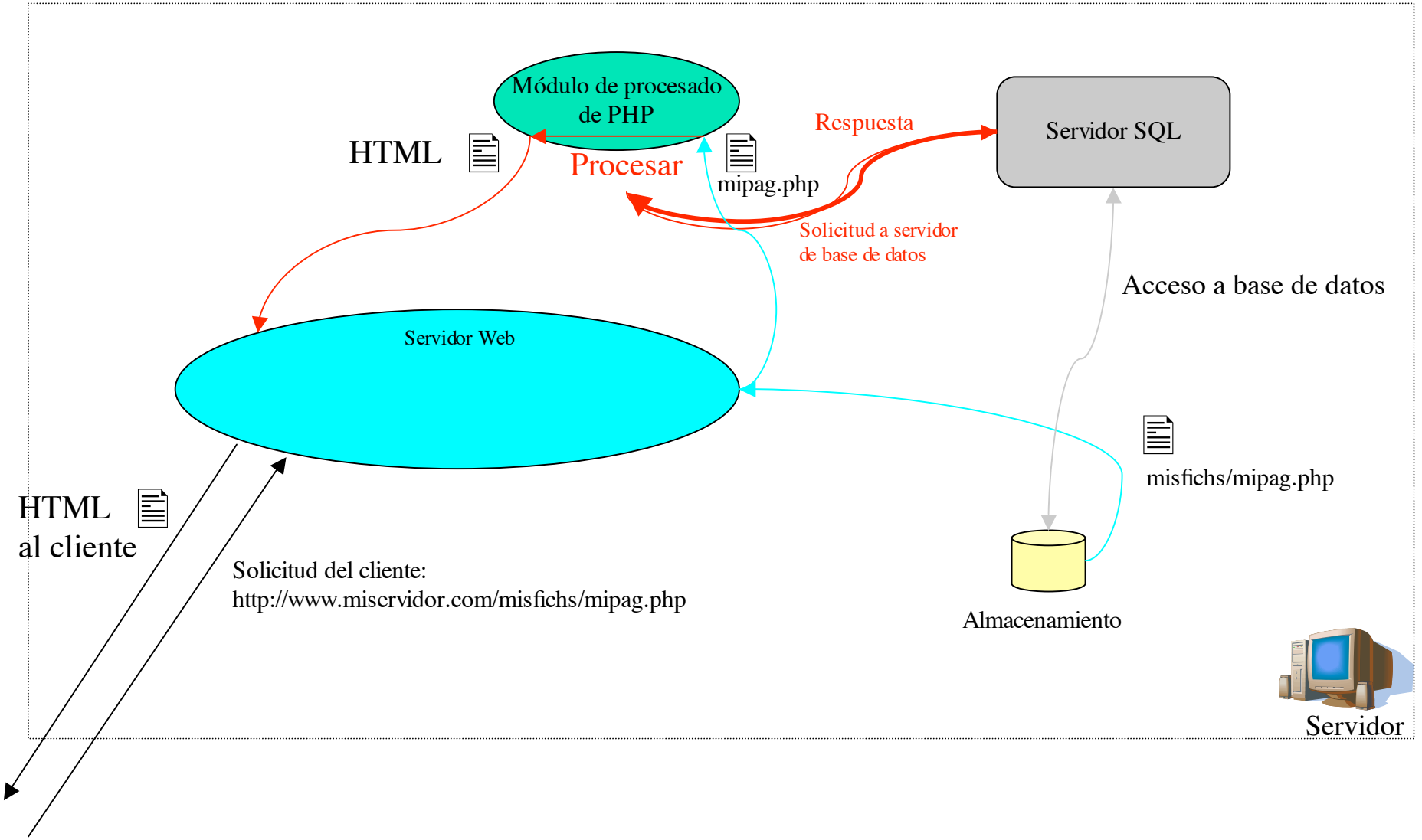
• El servidor reconoce que el fichero puede contener código PHP generalmente por la extensión del fichero

• El servidor busca en el documento los tags que marcan el código PHP. Lo ejecuta y si el script quiere escribir texto (`print()`) ese texto aparece donde estaba el código PHP al enviarse el documento (no se cambia el fichero)

Procesado en servidor



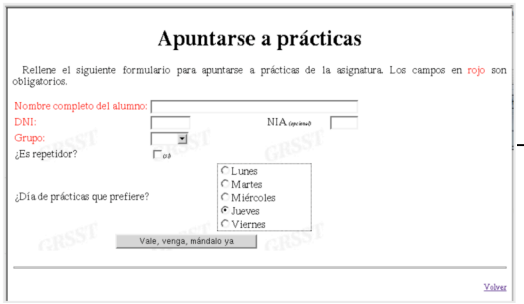
Ejemplo: procesado en servidor
(PHP+base de datos)



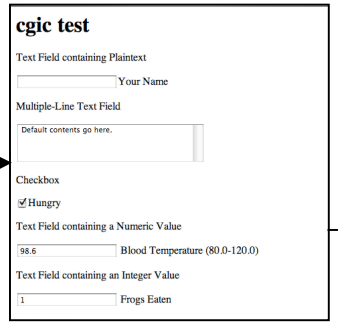
Persistent Client State HTTP Cookies

- Una de las limitaciones de la Web a la hora del desarrollo de aplicaciones/interfaces es que su funcionamiento es sin estado
- Cada petición de un URI es independiente de los anteriores y hay poca o ninguna información de lo que ha hecho el usuario anteriormente
- Eso quiere decir que formularios que ocupen varias páginas HTML son difíciles de implementar
- Un técnica clásica ha sido, al generar un CGI una página como resultado de un formulario esconder en esa página, en el siguiente formulario, la información que se quiere conservar (. . .)

Formulario



El CGI que procesa ese formulario crea el siguiente y esconde en él (atributo hidden) valores del anterior



Cuando se envía el contenido de este formulario se está enviando también información del anterior que se escondió en él

Persistent Client State HTTP Cookies

- Las Cookies son un mecanismo más cómodo para almacenar información de estado en el cliente
- Al enviar una página Web el servidor puede indicar al cliente que almacene cierta información
- Ese cliente, cuando solicite otras páginas de ese servidor enviará en la solicitud esa información que se le pidió almacenar (la cookie)
- Cuando el cliente solicita un URL envía las cookies que pertenezcan a ese dominio y dentro del camino (path) especificado

Persistent Client State HTTP Cookies

- Se introduce una cookie en el cliente mediante `Set-Cookie` en la cabecera HTTP:

  `Set-Cookie2: NAME=VALUE ; Comment=VALUE ; CommentURL="http_URL" ; Discard ; Max-Age=VALUE ; Path=PATH ; Port["portlist"] ; domain=DOMAIN_NAME ; secure ; Version=VALUE`

  - El contenido de la cookie sigue el formato `NAME=VALUE` (ni punto-y-coma ni coma ni espacios). "NAME" y "Version" son los únicos atributos obligatorios
  - Se le puede indicar una tiempo máximo de validez a la cookie
  - Si se ha indicado el atributo `domain` el cliente, cuando haga una petición, enviará la cookie solo si en el URL al que se le solicita el nombre de dominio de la máquina está dentro de ese dominio
  - Con el atributo `path` se puede restringir el subconjunto de URLs del dominio a los que se les enviará la cookie al solicitar una página
  - Si se indica el atributo `secure` esta cookie solo se enviará si la conexión es segura (sobre SSL)
- Las cookies se envían al servidor como parte de la cabecera HTTP

  `Cookie: $Version=VALUE; NAME1=VALUE1 [; path] [; domain] [; port] ; NAME2=VALUE2 ...`

Persistent Client State HTTP Cookies

Ejemplo (1):

- User Agent → Server

```
POST /acme/login HTTP/1.1
[form data]
```

El usuario se idenfica con un formulario

- Server → User Agent

```
HTTP/1.1 200 OK
Set-Cookie2: Customer="WILE_E_COYOTE"; Version="1"; Path="/acme"
```

La Cookie refleja la identidad del usuario

- User Agent → Server

```
POST /acme/pickitem HTTP/1.1
Cookie: $Version="1"; Customer="WILE_E_COYOTE"; $Path="/acme"
[form data]
```

El usuario selecciona un artículo para su carro de la compra

Persistent Client State HTTP Cookies

Ejemplo (2):

- Server → User Agent

```
HTTP/1.1 200 OK
Set-Cookie2: Part_Number="Rocket_Launcher_0001"; Version="1"; Path="/acme"
```

El carro de la compra contiene un artículo

- User Agent → Server

```
POST /acme/shipping HTTP/1.1
Cookie: $Version="1"; Customer="WILE_E_COYOTE"; $Path="/acme";
    Part_Number="Rocket_Launcher_0001"; $Path="/acme"
[form data]
```

El usuario selecciona la forma de envío

- Server → User Agent

```
HTTP/1.1 200 OK
Set-Cookie2: Shipping="FedEx"; Version="1"; Path="/acme"
```

La nueva Cookie contiene la forma de envío

Persistent Client State HTTP Cookies

Ejemplo (y 3):

- User Agent → Server

```
POST /acme/process HTTP/1.1
Cookie: $Version="1"; Customer="WILE_E_COYOTE"; $Path="/acme";
    Part_Number="Rocket_Launcher_0001"; $Path="/acme"; Shipping="FedEx";
    $Path="/acme"
[form data]
```

El usuario decide procesar el pedido

- Server → User Agent

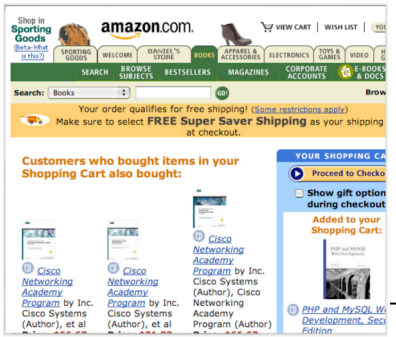
```
HTTP/1.1 200 OK
```

Transacción completada

Sesiones en PHP

- Sucede lo mismo con los scripts PHP que con CGIs: no se guarda estado
- PHP nos permite guardar el contenido de unas variables asociándolas a una sesión
- En realidad lo que hará será guardar esas variables localmente y mandar al usuario un identificador de sesión asociado a ese conjunto de variables en forma de una cookie
- Cuando el cliente solicita otro script PHP envía su cookie con el identificador de sesión
- Se accede al fichero correspondiente recuperando esas variables de forma que parece que conservar el contenido asignado por el anterior script (. . .)

Un script guarda la información obtenida del usuario en la pagina anterior en variables de sesión. Al mandar la pagina siguiente manda una cookie con el ID de la sesión



Al llamar a un nuevo script se le pasa el ID de la sesión con lo que este puede recuperar las variables guardadas

Contenido

- Network Address Translation
- Nivel de Aplicación
  - E-Mail
  - DNS
  - La Web
    - HTTP
    - Procesado en el cliente y en el servidor
  - **Seguridad**

Referencias

- [Kurose05] 8-8.5

Seguridad

Confidencialidad

- Solo emisor y receptor son capaces de entender el contenido del mensaje
- Encriptación

Autentificación

- Ambos son capaces de confirmar la identidad del otro

Integridad y no repudio

- Verificar que el mensaje no se ha alterado
- Probar que viene de quien dice

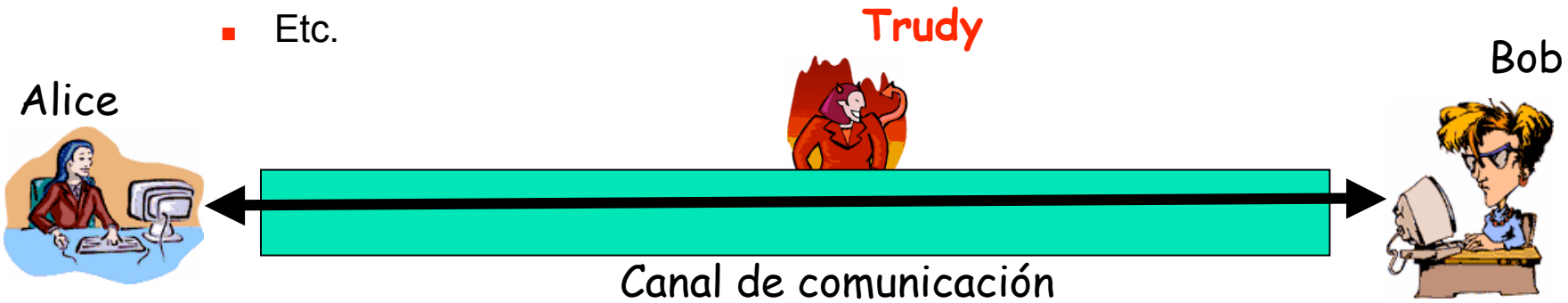
Disponibilidad y control de acceso

- Que la comunicación pueda tener lugar
- Evitar accesos no autorizados y DoS

Amigos y enemigos

- Usuarios A (Alice) y B (Bob)
- Desean comunicarse de forma segura
- Pueden ser:
  - Usuarios reales
  - Navegador y servidor Web
  - Cliente/Servidor de compras online
  - Routers intercambiando *updates*
  - Servidores de DNS
  - Etc.

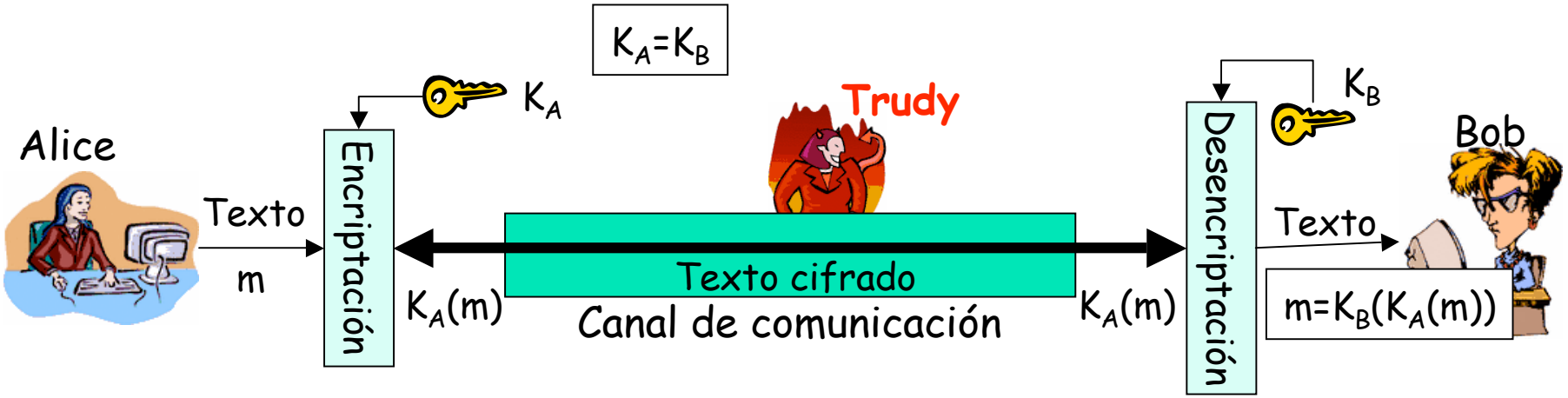
- Intruso (Trudy)
- ¿Qué puede hacer?
  - Interceptar mensajes
  - Insertar mensajes
  - Hacerse pasar por alguien
  - Evitar el uso del servicio



Criptografía

Clave simétrica

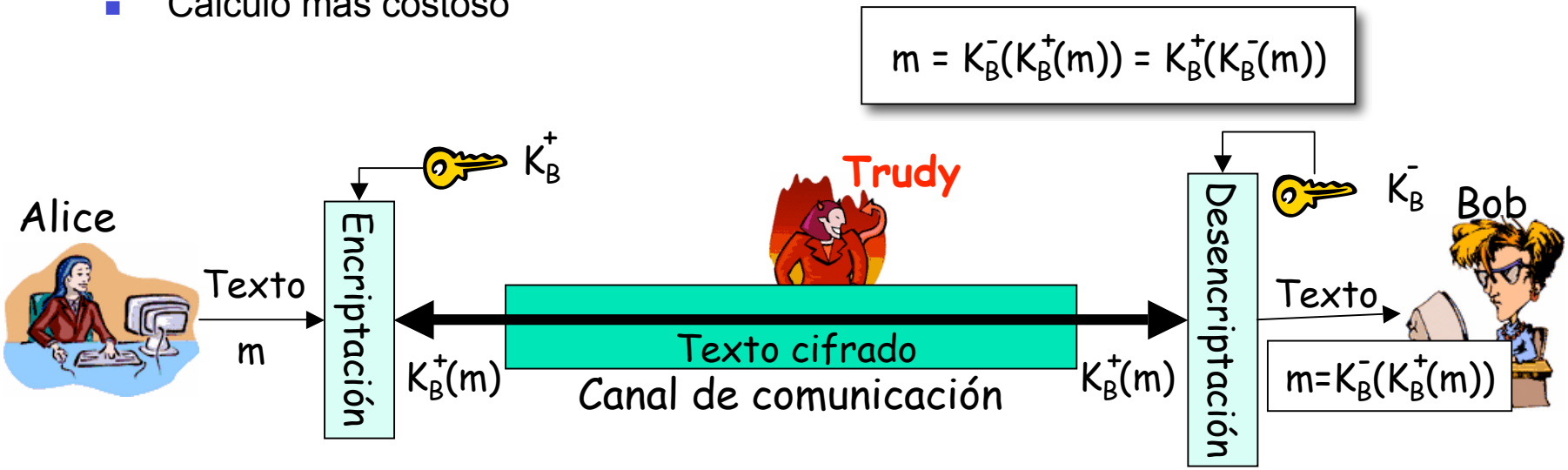
- Comparten un secreto $K_A=K_B$
- ¿Cómo se ponen de acuerdo en esa clave?
- Ejemplos: DES (Data Encryption Standard), AES (Advanced Encryption Standard)



Criptografía

Clave pública

- Cada uno tiene una **clave privada**
- También una **clave pública conocida por todos**
- Mensaje encriptado con una se desencripta con la otra
- Ejemplo: Diffie-Hellman, RSA
- Cálculo más costoso



Message Digest, Firma Digital

Hash:

- Dado un mensaje m obtener otro más corto H(m)
- De tamaño fijo
- Que sea imposible volver a *m*
- Muchos mensajes darán el mismo *hash*
- Ejemplo: MD5, SHA-1

Empleando criptografía de clave pública

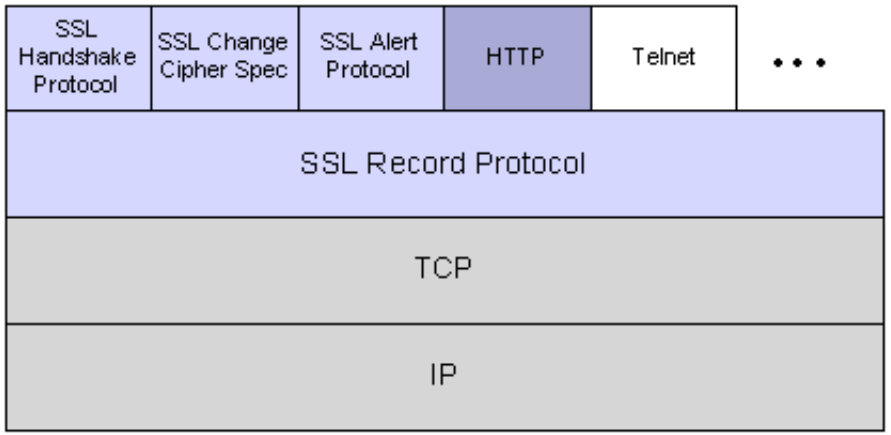
- Alice envía el mensaje y la firma
- La firma puede ser el propio mensaje encriptado con su clave privada
- Bob desencripta la firma con la clave pública de Alice
- Es igual al mensaje
- Solo ella conoce la clave privada luego solo ella pudo firmar

Certificados

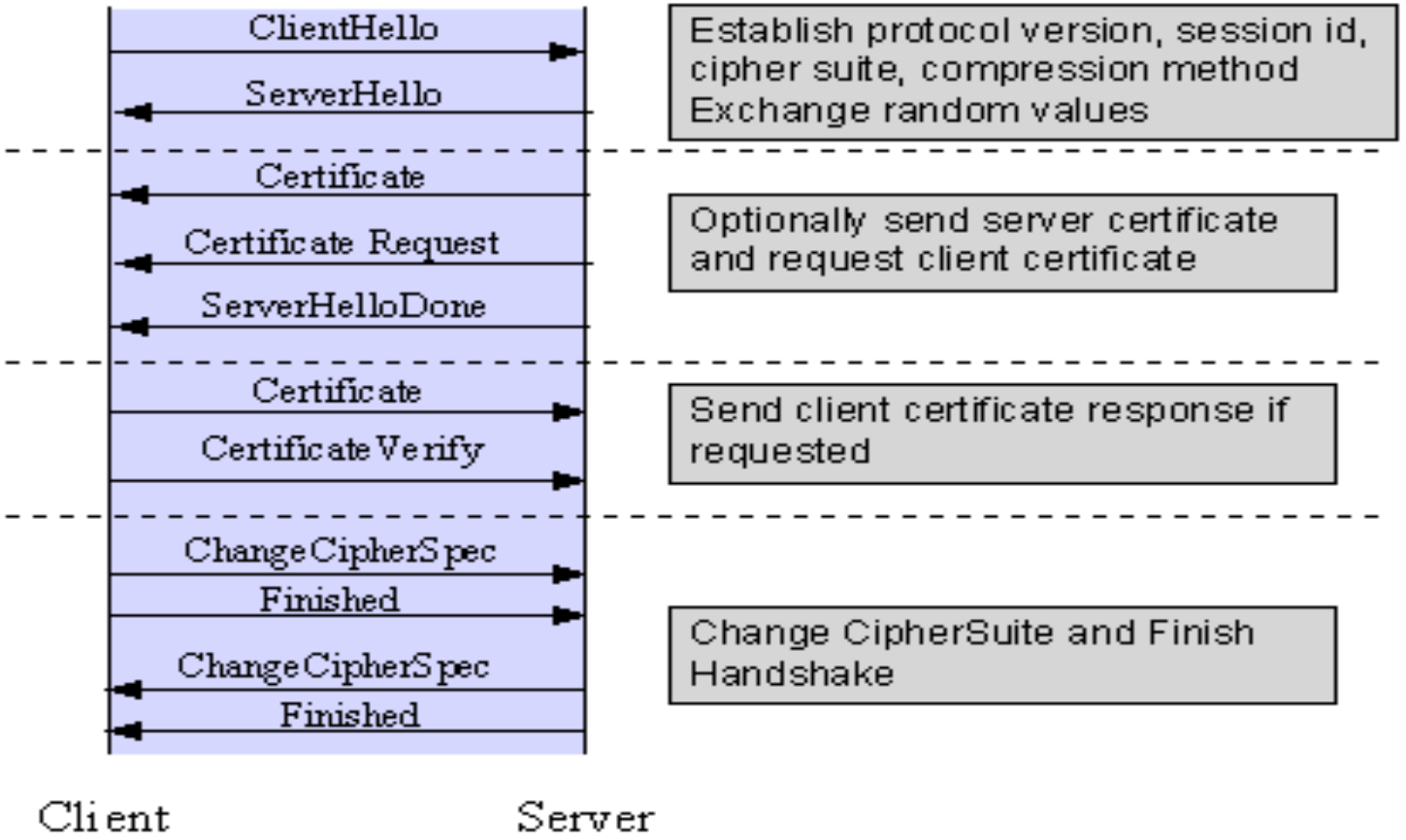
- ¿Cómo distribuir las claves públicas?
- ¿Cómo fiarse de que la clave pública es de quien dice ser?
- Cada usuario tenga un *certificado*:
  - Su clave pública y su identificación firmados
  - Por una *Autoridad de Certificación (CA)*
  - Suelen tener una caducidad
- Cada uno distribuye su clave pública junto con el certificado
- Una CA puede emitir un certificado para otra CA
- Para comprobar un certificado puede que haya que pasar por varias CAs
- Hasta llegar a una *Top-Level CA* que se firma a si misma el certificado
- Los navegadores Web tienen preconfiguradas algunas CAs bien conocidas
- Las CAs también crean *Certificate Revocation Lists*

SSL/TLS

- Secure Sockets Layer / Transport Layer Security
- SSL 3.0 → TLS 1.0
- Sesión:
  - Entre nivel de aplicación y un nivel de transporte fiable orientado a conexión (TCP)
  - Requiere un *handshake* entre cliente y servidor
  - Puede ser reutilizada mediante un session ID que se cachea
  - El cliente puede reutilizar el session ID para reducir el handshake
- Clave simétrica para encriptar los mensajes
- Acuerdo sobre clave empleando criptografía de clave asimétrica
- Una vez establecida una sesión con encriptación los mensajes de control SSL también se encriptan



Handshake


ClientHello
ServerHello
Establish protocol version, session id, cipher suite, compression method Exchange random values
Certificate
Certificate Request
ServerHelloDone
Optionally send server certificate and request client certificate
Certificate
CertificateVerify
Send client certificate response if requested
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
Change CipherSuite and Finish Handshake
Client
Server