

# Capítulo 2

## Capa de Aplicación de Internet

Application
Transport
Network
Link
Physical

# Capa de aplicación

- Veremos **dos enfoques para desarrollar aplicaciones de red para la internet:**
  1. El programador para especificar la comunicación usa **una interfaz para programas de aplicación (API)**.
    - Una API es conjunto básico de funciones.
    - La **socket API** se usa para el software que se comunica sobre la internet.
    - Esto lo van a ver en el taller.
    - **El saber sobre el sistema operativo de red subyacente tiene su utilidad:**
      - permite al programador escribir mejor código y desarrollar aplicaciones más eficientes.

# Capa de aplicación

**2. La Web:** El programador se apoya en **la tecnología de la web** para construir una aplicación de red..

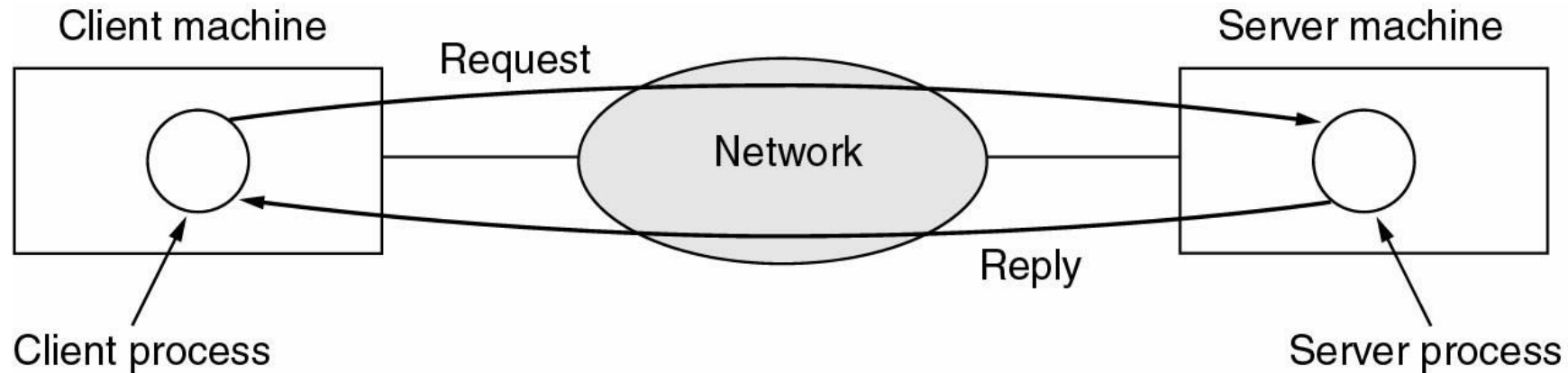
- La Web provee servicios al software de la aplicación que
  - hacen más fácil a los desarrolladores implementar la comunicación y la entrada/salida de modo que
  - se pueden enfocar en el propósito específico de la aplicación.
- Se estudiará en el teórico-práctico de la materia.

# Arquitecturas de aplicaciones

Las aplicaciones red de internet suelen usar uno de los siguientes **estilos de arquitectura**:

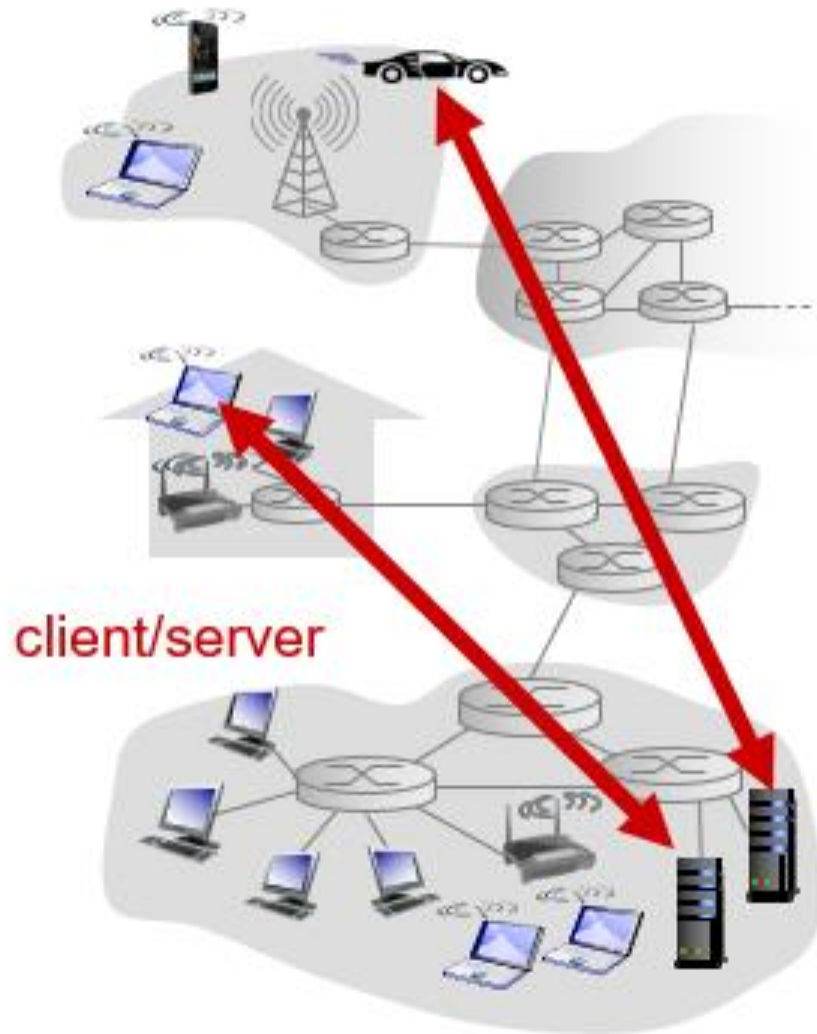
- Cliente-servidor
- Peer-to-peer (P2P)
- Arquitectura orientada a servicios
- Microservicios

# Arquitecturas Cliente-Servidor



- En el **modelo cliente-servidor** hay dos procesos que se comunican: uno en la máquina cliente y otro en la máquina servidor.
- **Forma de la comunicación:**
  1. El proceso cliente manda **solicitud** al proceso servidor,
  2. el proceso cliente espera un **mensaje de respuesta**;
  3. luego el proceso servidor recibe y procesa la solicitud;
  4. el proceso servidor manda mensaje de **respuesta** al proceso cliente.

# Arquitecturas Cliente-Servidor



## Características de los servidores:

- Siempre están en un host;
- con dirección IP permanente;
- se pueden usar centros de datos para escalabilidad.

## Características de los clientes:

- Pueden estar conectados **intermitentemente**;
- usando direcciones IP dinámicas;
- Los clientes no se comunican directamente entre sí.

# Aplicaciones Cliente Servidor en internet usando UDP

## Pasos de una aplicación cliente-servidor usando UDP.

1. Cliente crea datagrama con IP y puerto del servidor y envía datagrama
    - Datagrama puede perderse.
  2. Si llega, servidor lee datagrama
  3. Servidor envía respuesta especificando dirección y puerto de cliente
    - Datagrama puede perderse.
  4. Si llega, cliente lee datagrama.
  5. Cliente finaliza
- **Evaluación:**
    - No se dice qué se hace si respuesta no llega al cliente.
    - Es responsabilidad de la aplicación red manejar esto.

# Aplicaciones Cliente Servidor en internet usando TCP

- **Pasos de una aplicación cliente-servidor usando TCP:**

1. Se ejecuta proceso del servidor
2. Servidor espera por pedido de conexión entrante.
3. El cliente requiere pedido de conexión al servidor
4. El servidor acepta la conexión con el cliente
5. El cliente envía pedido al servidor
6. El servidor lee el pedido
7. El servidor envía la respuesta
8. El cliente lee la respuesta
  - TCP provee transferencia de stream de bytes ordenada
9. Si hay más pedidos al servidor: Goto 5
10. El cliente cierra la conexión
11. El servidor cierra la conexión



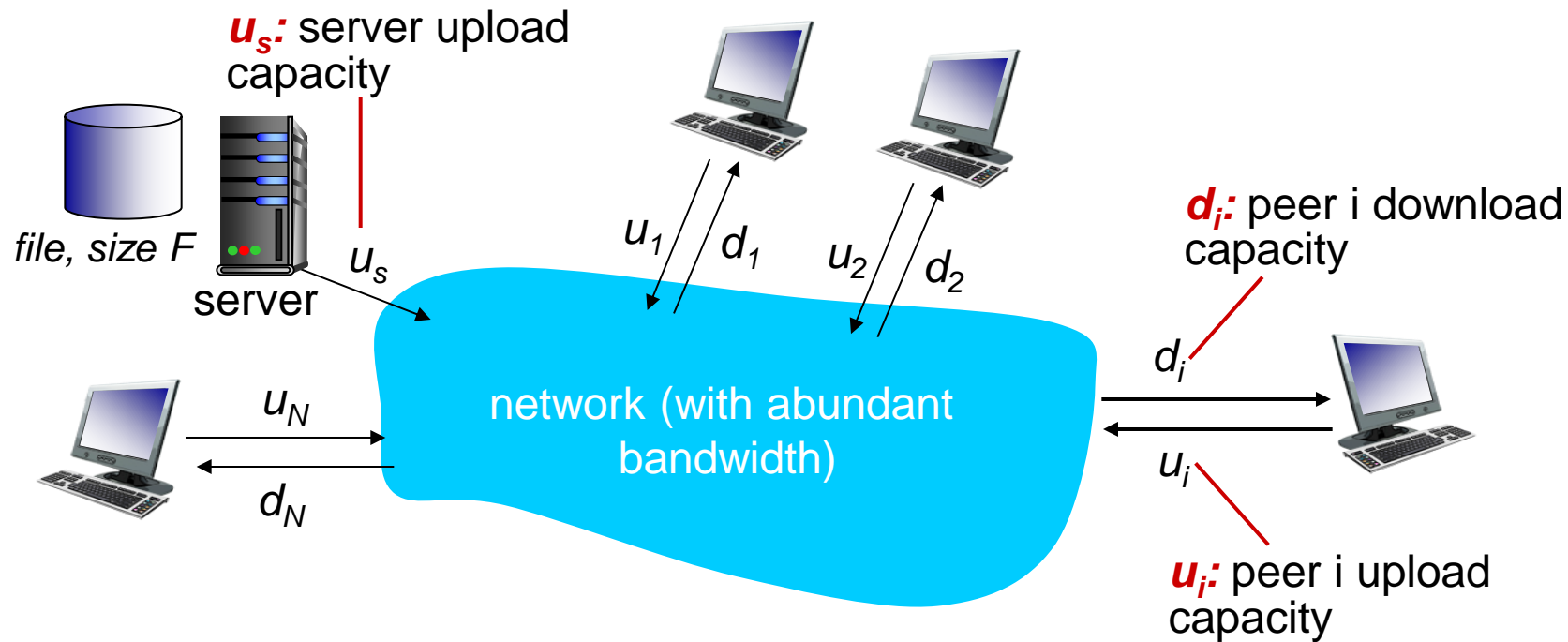
# Arquitectura P2P

- **Ejemplos de aplicaciones P2P:**
  - **Distribución de archivos:** la aplicación distribuye un archivo de una única fuente a un gran número de compañeros.
    - Un ejemplo es BitTorrent.
  - **Bases de datos distribuidas** sobre una gran comunidad de compañeros.
  - **Streaming:** video on demand - (P.ej: KanKan)
  - **VoIP:** voz sobre IP (P.ej: Skype)

# Distribución de archivos: P2P vs cliente-servidor

**Problema: ¿Cuánto tiempo se requiere para distribuir un archivo (de tamaño  $F$ ) de un servidor a  $N$  compañeros?**

- La capacidad de subida y de bajada de compañeros es un recurso limitado.
- Responderemos la pregunta para cliente-servidor y para P2P.

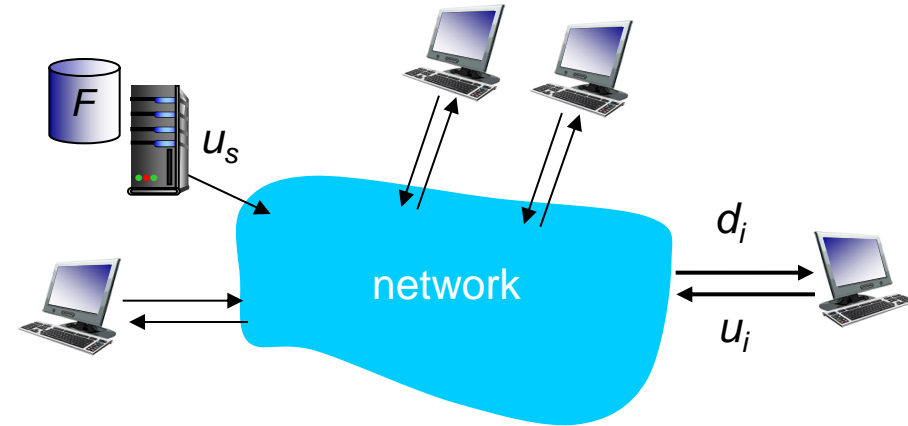


# Distribución de Archivos: Cliente-Servidor

- ¿Qué Parámetros hay que considerar?
  - Tasa de subida del enlace de acceso al compañero  $i$ :  $u_i$
  - Tasa de subida del enlace de acceso al servidor:  $u_s$
  - Tasa de descarga del enlace de acceso al compañero  $i$ :  $d_i$
  - Tamaño del archivo a ser distribuido:  $F$
  - Número de compañeros que quieren adquirir una copia del archivo:  $N$
- El **tiempo de distribución** es el tiempo que toma obtener una copia del archivo por los  $N$  compañeros.
  - Asumimos que la internet **tiene abundante ancho de banda** y todos los cuellos de botella suceden en ISP de acceso.
  - Asumimos que los servidores y clientes **no participan** de otras aplicaciones de red.

# Distribución de archivos: cliente-servidor

- *Transmisión del servidor:* debe enviar secuencialmente (subida)  $N$  copias de archivo a cada peer (manda  $NF$  bits).
  - Tiempo para enviar 1 copia:  $F/u_s$
  - Tiempo para enviar  $N$  copias:  $NF/u_s$
  - Demasiado trabajo del servidor



- ❖ *Descarga del Cliente:* cada cliente debe descargar una copia de archivo.
  - $d_{\min} = \min\{d_1, d_p, \dots, d_N\}$ .
  - Tiempo de descarga del cliente con  $d_{\min}$ :  $F/d_{\min}$  segs
  - Este es el tiempo de descarga peor.

Ningún compañero ayuda a distribuir el archivo.

Tiempo para distribuir  $F$   
a  $N$  clientes usando  
enfoque cliente-servidor

$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

aumenta linealmente en  $N$

# Distribución de Archivos: P2P

- Al comienzo de la distribución
  - solo el servidor tiene el archivo.
- Para que la comunidad de compañeros reciba este archivo,
  - el servidor debe enviar cada bit del archivo al menos una vez en su enlace de acceso.
- En P2P cada compañero puede redistribuir cualquier porción del archivo que ha recibido a cualesquiera otros compañeros.
  - Así los compañeros asisten al servidor en el proceso de distribución.
  - Cuando un compañero recibe algo de datos de un archivo, puede usar su capacidad de subida para redistribuir los datos a los otros compañeros.

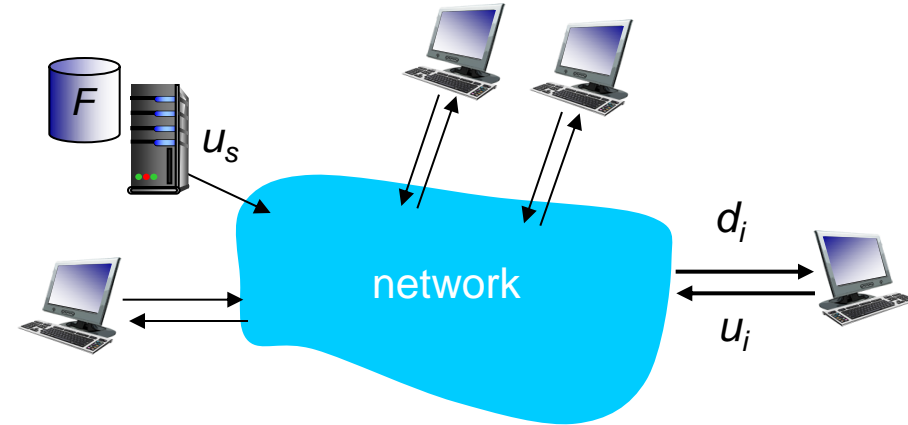
- La capacidad total de subida del sistema es:

$$u_{\text{total}} = u_s + \sum u_i$$

- Por lo tanto el **tiempo mínimo de distribución** es:  $NF/u_{\text{total}}$

# Tiempo de Distribución de Archivos: P2P

- *Transmisión de servidor*: debe subir al menos una copia.
  - Tiempo para enviar una copia :  $F/u_s$
- ❖ *cliente*: cada cliente debe descargar la copia de un archivo
  - Tiempo mínimo de descarga de cliente:  $F/d_{\min}$
- ❖ *clientes*: como agregado deben subir  $NF$  bits
  - Tasa de subida máxima (tasa máxima limitante de descarga) es  $u_s + \sum u_i$



tiempo para distribuir  $F$   
a  $N$  clientes usando  
enfoque P2P

$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

incrementa linealmente en  $N$  ...

... pero lo hace mientras cada compañero brinda servicio

# Distribución de Archivos

*Tiempo para distribuir  $F$   
a  $N$  clientes usando  
enfoque cliente-servidor*

$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

increases linearly in  $N$

versus

*tiempo para distribuir  $F$   
a  $N$  clientes usando  
enfoque P2P*

$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

increases linearly in  $N$  ...

... but so does this, as each peer brings service capacity

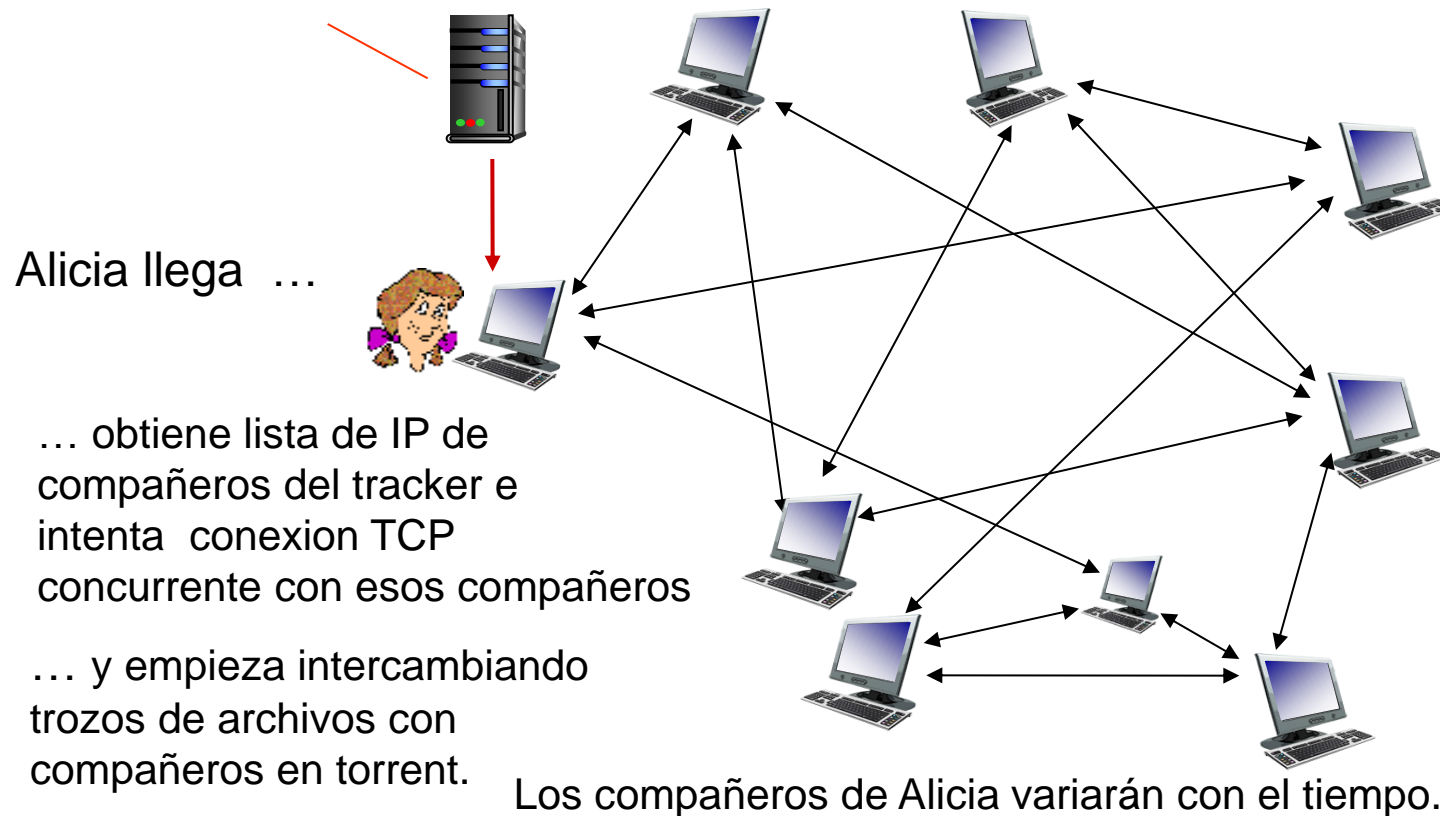
¿Qué diferencia observan?

# Distribución de archivos P2P: BitTorrent

- ❖ El archivo se divide en trozos de 256Kb.
- ❖ Los compañeros en torrent envían/reciben trozos.

**tracker:** lleva la pista de compañeros Participando en Torrent

**torrent:** grupo de compañeros intercambiando trozos de un archivo

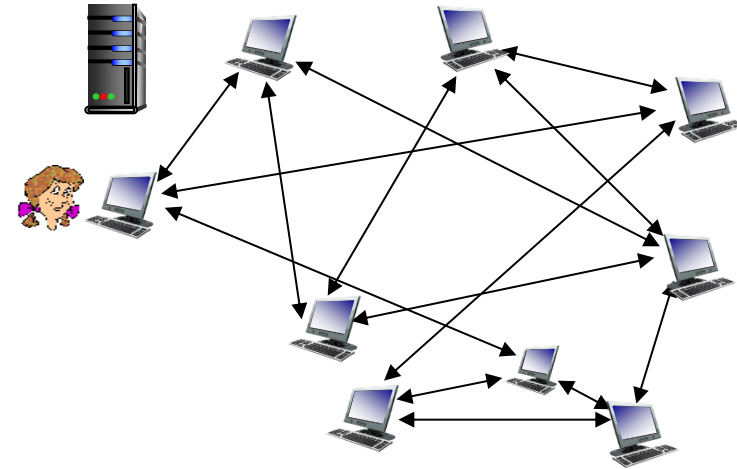




# Distribución de archivos P2P: BitTorrent

- **Cuando compañero se une a Torrent:**

- No tiene trozos, pero va a acumularlos a lo largo del tiempo de otros compañeros
- Se registra con tracker para obtener lista de compañeros.
- Se conecta con un subconjunto de compañeros (“vecinos.”)
- Un compañero avisa periódicamente a tracker que está en BitTorrent.



- ❖ Mientras descarga, compañero sube trozos a otros compañeros.
- ❖ Un compañero puede cambiar de compañeros con los cuales intercambia trozos.
- ❖ Los compañeros pueden ir y venir.
- ❖ **Una vez que un compañero tiene un archivo completo:**
  - Puede (egoístamente) irse o (altruísticamente) permanecer en Torrent subiendo trozos.

# BitTorrent: pedir y enviar trozos de archivos

## *Pedir trozos:*

- En un momento dado, diferentes compañeros tienen diferentes subconjuntos de trozos de archivos
- Periódicamente, Alicia pide a cada compañero por una lista de trozos que ellos tienen.
- **Alicia puede hacer esto porque**
  - sabe qué trozos tienen sus compañeros.
- **Conviene pedir primero los trozos**
  - menos comunes (i.e. con menos copias en compañeros).

## *Enviar trozos: tit-for-tat*

- Alicia envía trozos a aquellos 4 compañeros actualmente enviándole a Alicia trozos a la velocidad mayor.
  - re-evalua los 4 mejores cada 10 seg
- Cada 30 seg: elegir aleatoriamente otro compañero, y comenzar enviándole trozos.
  - Un compañero nuevo elegido puede unirse a los 4 de más arriba.
  - Pero para esto tiene que ser uno de los 4 mejores subidores para Alicia.

# Arquitectura orientada a servicios

- **Requisitos funcionales:**
  - **Provisión de servicios:** los servicios deben ser capaces de proporcionar funcionalidades específicas.
  - **Consumo de servicios:** los clientes (i.e. aplicaciones o usuarios) deben poder solicitar y recibir servicios.
- **Requisitos no funcionales:**
  - **Interoperabilidad:** entre servicios. O sea comunicación entre servicios de manera efectiva independientemente de la plataforma o lenguaje de programación usado.
  - **Reusabilidad:** los servicios deben diseñarse para reutilizarse en diferentes contextos.
  - **Escalabilidad:** los servicios deben escalar según sea necesario.
  - **Flexibilidad:** los servicios deben ser capaces de adaptarse a cambios en las necesidades del negocio.
  - **Seguridad:** Garantizar la seguridad de los datos y las transacciones entre servicios.

# Arquitectura orientada a servicios

- **Solución:** Organiza las aplicaciones en **servicios reutilizables** que se comunican entre sí a través de un bus de servicios.
  - Cada servicio realiza una función específica y puede ser usado por diferentes aplicaciones.
  - En arquitectura SOA los servicios se comunican entre si usando patrones como: solicitud-respuesta, publicar-suscribir, o enviar-olvidar.
  - Los servicios son modulares y pueden actuar tanto como clientes como servidores dependiendo del contexto.

# Arquitectura orientada a servicios

- **Solución:**
  - **Nodos** (roles):
    - **Servicios independientes:** cada servicio tiene un rol definido.
    - **Bus de servicios empresarial** (ESB): infraestructura de software que facilita la integración y comunicación entre los servicios.
    - **Clientes:** consumen los servicios ofrecidos
  - **Mensajes de comunicación:**
    - **Clientes a ESB:** solicitudes de servicios, datos a procesar
    - **ESB a servicios:** enrutamiento de solicitudes a los servicios adecuados.
    - **Servicios a ESB:** respuestas a las solicitudes, datos procesados
    - **Servicios entre sí:** comunicación para coordinar acciones y compartir datos.

# Arquitectura de microservicios

- **Requisitos funcionales:**

- **Provisión de servicios** especializados: proporcionar funciones únicas y bien definidas.
- **Comunicación entre servicios**: para cumplir con tareas más complejas.
- **Consumo de servicios**: los clientes o aplicaciones deben poder solicitar y recibir servicios de manera eficiente.

- **Requisitos no funcionales:**

- **Escalabilidad**: escalamiento independiente de cada servicio según demanda.
- **Flexibilidad**: en el desarrollo y despliegue, los servicios deben adaptarse a los cambios en las necesidades del negocio.
- **Mantenibilidad**: fácil implementar nuevas funcionalidades.
- **Seguridad**: se pueden implementar políticas de seguridad más centralizadas y consistentes a través de los servicios. Garantizar la seguridad de los datos y las transacciones entre servicios.
- **Independencia y autonomía**: cada servicio debe ser capaz de desarrollarse, implementarse y escalarse de manera independiente.
- **Resiliencia**: los servicios deben diseñarse para tolerar fallos y mantener la operación continua, incluso si uno de ellos falla.

# Arquitectura de microservicios

- **Solución:** **Arquitectura de microservicios** que es una evolución de SOA.
  - La aplicación se divide en pequeños **servicios independientes** que se comunican entre sí a través de APIs que no dependen de un lenguaje específico.
    - Cada microservicio se especializa en una sola tarea.
    - Cada microservicio se encarga de una funcionalidad específica.
    - Un microservicio puede actuar tanto como cliente como servidor dependiendo del contexto y la tarea que se está realizando.
  - Uso de APIs REST, o gRPC para la comunicación.
  - **Nodos** (roles):
    - **Servicios independientes:** Componentes funcionales de aplicación que interactúan entre sí.
    - **API Gateway:** intermediario que gestiona las solicitudes entre clientes y microservicios.
    - **Clientes:** aplicaciones que consumen los servicios.

# Arquitectura de microservicios

- **Solución – cont:**

- **Mensajes de comunicación:**

- **Cientes a API Gateway:** solicitudes de datos, comandos
    - **API Gateway a Microservicios:** Ruteo de solicitudes a los microservicios correspondientes.
    - **Microservicios a API Gateway:** respuesta a las solicitudes, resultados de las operaciones.
    - **Microservicios entre sí:** comunicación inter-servicios para operaciones complejas.

- **Protocolos:** se suele usar SOAP o REST para la comunicación entre servicios.



# Protocolos de capa de aplicación

## Cosas a definir en un protocolo de capa de aplicación:

- Tipos de mensajes intercambiados
  - P.ej: de pedido, de respuesta
- Sintaxis del mensaje:
  - Qué campos hay en un mensaje y Cómo los campos son delineados
- Semántica del mensaje
  - Significado de la información en los campos

**Reglas** de cuándo y cómo los procesos envían y responden a mensajes.

**Estado** de la aplicación. En qué consiste y cómo se lo mantiene.

## Tipos de protocolos

### Protocolos abiertos:

- Son definidos en RFCs
- Permiten interoperabilidad
- P.ej: HTTP, SMTP, FTP

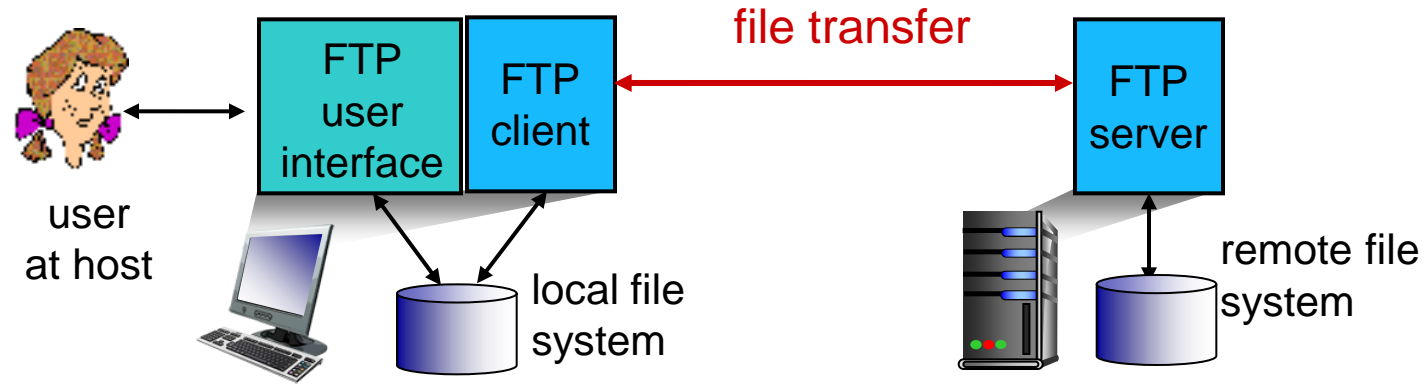
### Protocolos propietarios:

- P.ej: Skype

# Protocolos de capa de aplicación

- Para hacer un diseño detallado de una aplicación de red sobre internet conviene definir un **protocolo de capa de aplicación**.
- Este protocolo se puede apoyar o no en un protocolo de base.
  - P.ej. Una aplicación web se apoya en HTTP.
  - P.ej. FTP no se apoya en protocolo de base de capa de aplicación.
  - Estudiaremos FTP y HTTP.

# FTP: Protocolo de Transferencia de Archivos



## ❖ Algunas características de FTP:

- Usado para transferir archivo hacia/desde host remoto
- Cada archivo tiene restricciones de acceso y posesión.
- FTP permite inspeccionar carpetas.
- FTP permite mensajes de control textuales.

## ❖ Modelo cliente/servidor

- *cliente*: lugar que inicia la transferencia (hacia o desde el host remoto)
- *servidor*: host remoto.

## ❖ Servidor ftp: puerto 21

# FTP: Protocolo de Transferencia de Archivos

- **3 tipos de mensajes son intercambiados:**
  - Uso de **comandos** enviados al servidor ftp
    - Enviados como texto ASCII sobre un canal de control
  - P. ej: comando de transferencia de archivo.
  - **Mensajes de respuesta** a comandos del servidor ftp
  - **Mensajes con datos enviados.**
    - Listado de folder, archivo, etc.

# FTP: Protocolo de Transferencia de Archivos

## *Sintaxis de comandos*

- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR *filename*** retrieves (gets) file
- **STOR *filename*** stores (puts) file onto remote host

## *Sintaxis de Mensajes de respuesta*

- ❖ Código de estatus y frase

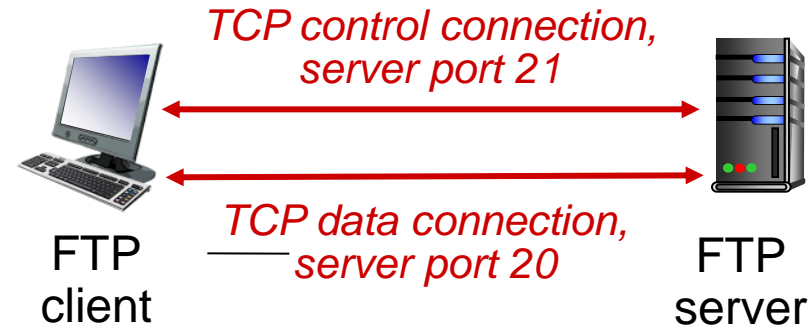
## *Ejemplos de Mensajes de respuesta*

- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

# FTP: Protocolo de Transferencia de Archivos

## Reglas:

1. Cliente FTP contacta servidor FTP en puerto 21, usando TCP.
2. El cliente es autorizado en la conexión de control.
3. El cliente inspecciona directorio remoto, envía comandos sobre la conexión de control.
  - Se comienza con identificación de usuario y password.
4. Cuando el servidor recibe un comando de transferencia de archivo, el *servidor* abre una 2<sup>da</sup> *conexión de datos* TCP (para el archivo) *con el cliente*.
5. Luego de transferir un archivo, el servidor cierra la conexión de datos.

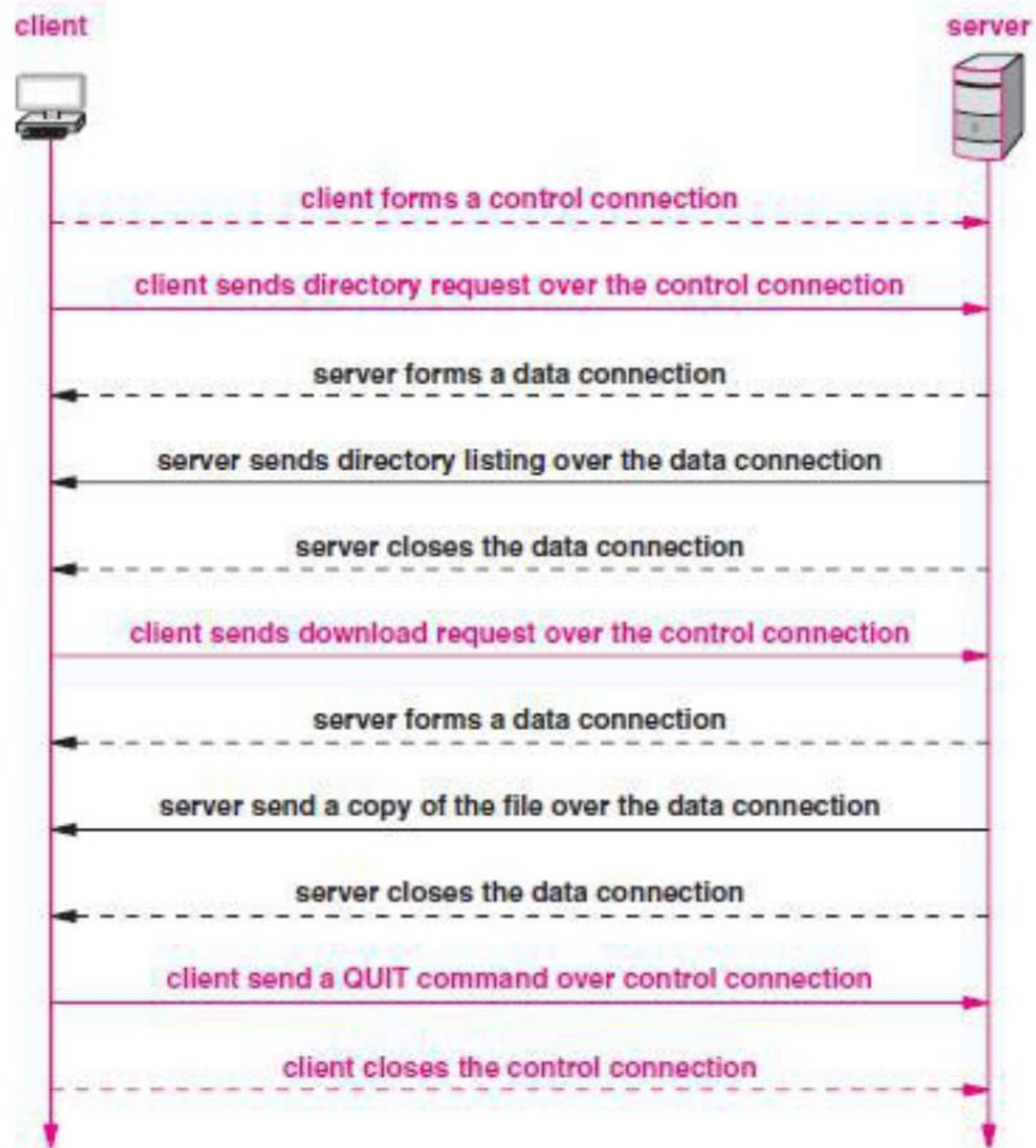


- El servidor abre otra conexión TCP de datos para transferir otro archivo.

## Estado

- El servidor FTP mantiene el “estado”: **directorio corriente, autenticación previa.**

# FTP



# La Web: Datos e información

- Para las **páginas web** suelen ser importantes los **datos** y la **información**.
- En el mundo hay **entidades** y **relaciones** entre entidades.
  - Por ejemplo: entidades persona y auto, relación dueño de auto.
- Los datos se refieren a los datos de esas entidades y relaciones.
  - Por ejemplo: para persona: DNI, nombre, edad, dirección, celular. Para auto: número de patente, kilometraje, marca, modelo, etc. Para dueño: DNI y número de patente.
- Los datos suelen estar en **bases de datos**.
- Los datos a los que accede una aplicación web pueden sacarse de varias fuentes de datos (p.ej. bases de datos).



# La Web: Datos e información

- Los datos se procesan de determinada manera y obtiene lo que se llama **información**.
- **Las páginas web muchas veces contienen información**; esa información puede obtenerse de distintas fuentes de datos, los datos extraídos son juntados y organizados de determinada manera dando lugar a información.
- También la información puede contener estadísticas sobre los datos.
- **Por ejemplo**, una página muestra la siguiente información: para una persona sus datos personales y los autos que posee indicando marca y cantidad de autos de esa marca que tiene.
- La información también se presenta mediante diagramas, figuras, tablas, infográficos. No necesariamente tiene que presentarse mediante texto.
- Una **página web** puede contener solo datos; una página web puede ser de información.

# La Web: Consultas y navegación

- Los datos pueden ser **consultados**. Una **consulta** suele describir usando un lenguaje los datos deseados.
- Por ejemplo: para para persona con DNI, nombre, edad, dirección, y celular: obtener los nombres de las personas de menos de 30 años que viven en barrio Alberdi.
- Existen **lenguajes de consulta** que sirven para expresar consultas. Las consultas son procesadas por **motores de bases de datos** para retornar los datos deseados.
- Se pueden consultar los datos para generar información.

# La Web: Consultas y navegación

- Para ver los datos deseados otra alternativa a escribir consultas en un lenguaje de consultas es **navegar**. Al navegar uno va viajando por una serie de pantallas que contienen los datos que desea inspeccionar.
- Pero también se puede navegar pasando por pantallas de información.
- Llamamos **hipertexto** a un conjunto de textos, cada uno de los cuales contiene enlaces a otros textos. Al seleccionar un enlace se muestra el texto deseado enlazado. Recorrer varios enlaces de hipertexto es navegar.
  - Por lo tanto,
  - hipertexto es un modelo para navegación.
- Nos referimos con **medias** a cosas como fotos, videos, audios, gráficos, animaciones.
- Podemos generalizar hipertexto a **hipermedia**; en hipermedia tenemos un conjunto de nodos, donde:
  - Un nodo puede contener texto y medias.
  - Cada nodo puede tener enlaces a otros nodos.

# La web

- La web original (1990) = hipertexto + transferencia de archivos
  - Los archivos residen en servidores web y son mostrados en un browser.
- Para soportar la web original se definió: HTTP, browser, servidor web y HTML.
- La web en 1993 = hipermedia + transferencia de archivos
  - Las páginas web contenían medias y para traer una página primero se traía el texto escrito en HTML y luego se traían las distintas medias.
- Luego la web = hipermedia + páginas generadas por servidor + transferencia de archivos
  - En 1993 surge CGI que permite usar lenguajes de programación para generar páginas web.
  - Ejemplos de lenguajes para generar páginas del lado del servidor: PHP (1994), Active server pages (1996), Java Server Pages.

# 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.
    - La página única suele estar escrita usando HTML y JavaScript.
    - Se inserta código en JavaScript a ejecutar dentro de una página HTML.
  - 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 asíncronico**.
  - 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).

`http://www.someschool.edu/someDept/pic.gif`

nombre de host

camino

# 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 páginas 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.

# Aplicaciones 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.
- En 1995 aparecieron aplicaciones de comercio electrónico como Amazon y Ebay.

# Aplicaciones Web

- En una **aplicación web social**:
  - Los usuarios pueden crear y compartir contenido
  - Los usuarios pueden comunicarse entre sí mediante posts, comentarios y mensajes.
- **Web 2.0** son sitios y aplicaciones que hacen uso de contenido generado por usuarios finales. La Web 2.0 se caracteriza por una mayor interacción del usuario, una mayor colaboración de los usuarios y canales de comunicación mejorados.
  - No solo contiene aplicaciones de web social sino también cosas como Wikipedia y blogs.
- En los 2000 aparecieron **plataformas para redes sociales** como Facebook (2004), YouTube (2005), Twitter (2006), LinkedIn (2002).
- En 2010 aparecieron aplicaciones de streaming como Netflix.

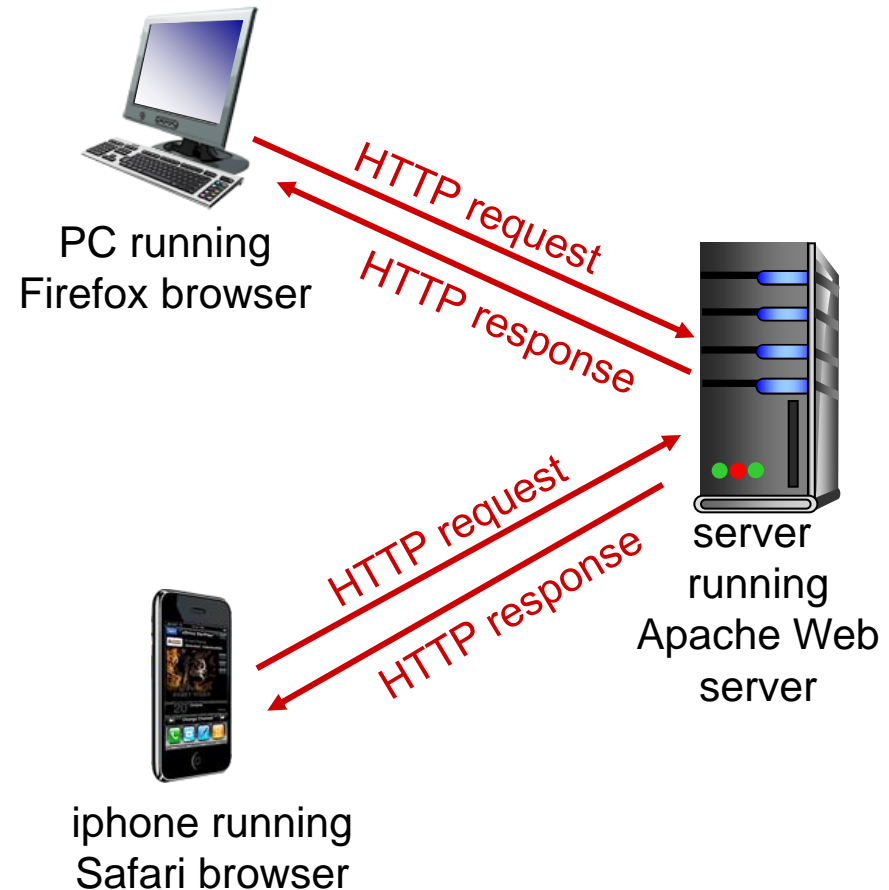


# Aplicaciones Web

- Actualmente se está desarrollando la **web 3.0** caracterizada por:
  - **Descentralización**: los usuarios están en control de sus datos; los usuarios son dueños del valor que generan; conexiones compañero a compañero.
  - **Blockchain**: permite transacciones más seguras, privacidad de datos y propiedad.
  - **Web semántica**: los datos tienen significado y son legibles por computadoras. Se consideran:
    - **Datos enlazados abiertos**: se identifican objetos con URIs, al ver un URI se accede a información útil, Dentro de la información a un URI puede haber enlaces a otros URIs.
    - **Metadatos semánticos**: etiquetas semánticas agregadas a las páginas web para describir mejor su significado.
  - **Uso de inteligencia artificial**: procesamiento de lenguaje natural y aprendizaje automático.

# 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 formato 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

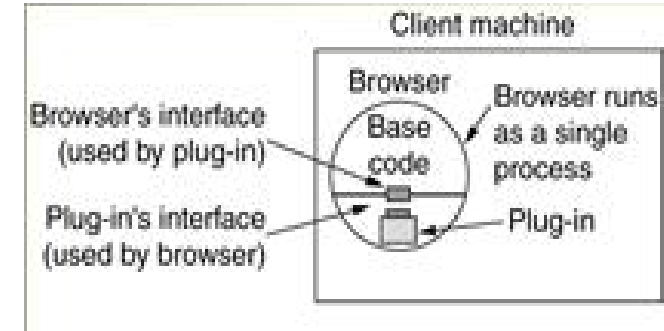
- **Problema:** No todas las páginas contienen solamente HTML.
  - ❑ **Ejemplo:** una página puede tener un ícono de formato GIF, una foto de formato JPEG, y un video de formato MPEG. Las páginas HTML pueden vincular cualquiera de estos.
  - ❑ Cuando el cliente recibe una página, no necesariamente es 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**.

# 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):**

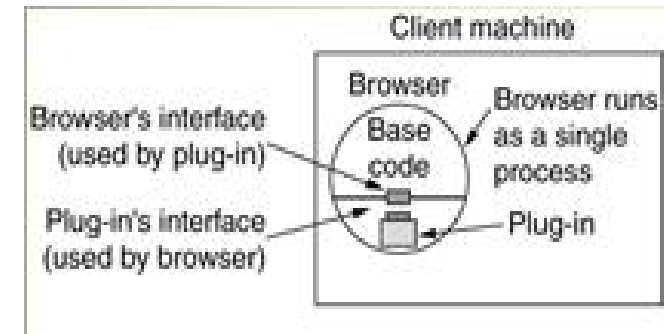
- ❑ La **interfaz del plug-in**: conjunto de procedimientos que todos los plug-in tienen que implementar a fin de que el navegador pueda llamarlos

- 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

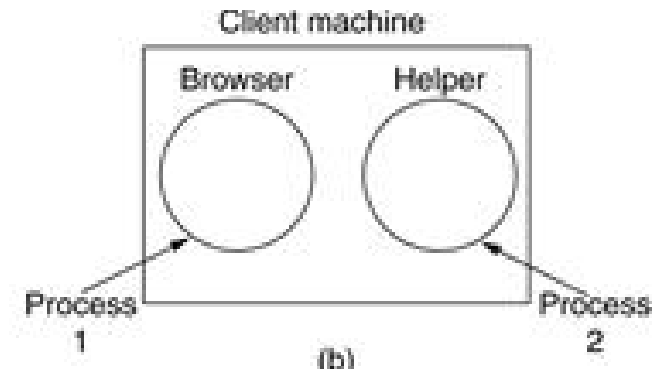




# Plug-ins y aplicaciones de ayuda

- **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.



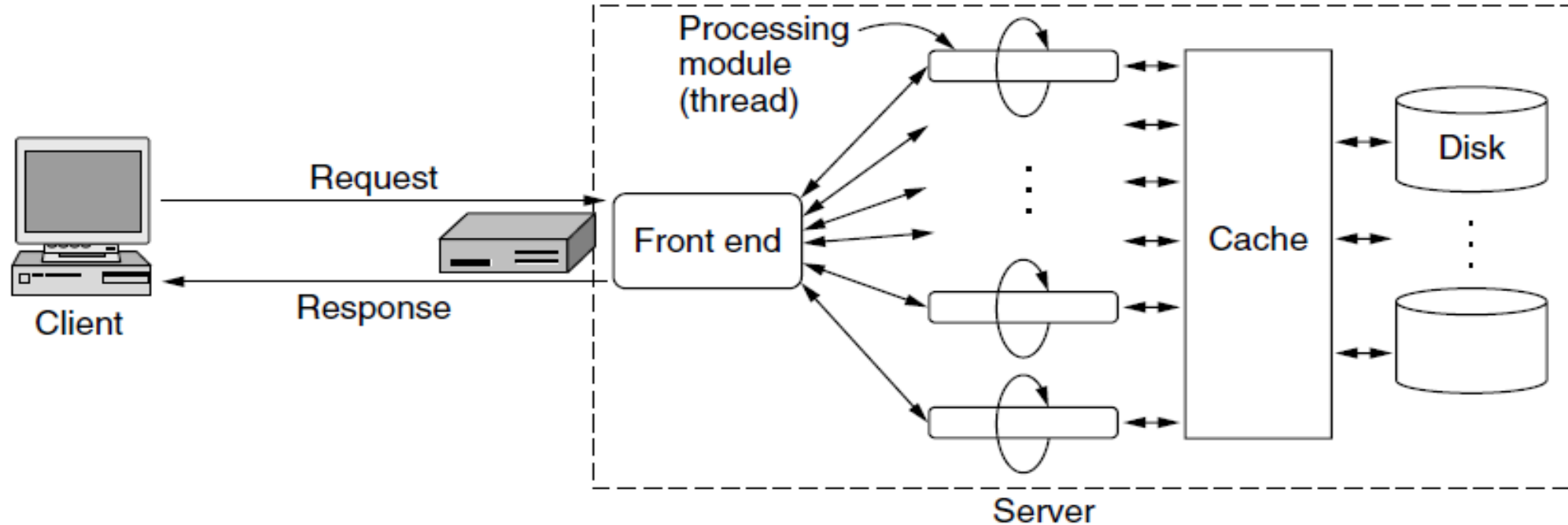
# Servidores Web

- 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.

# 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.

# 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**

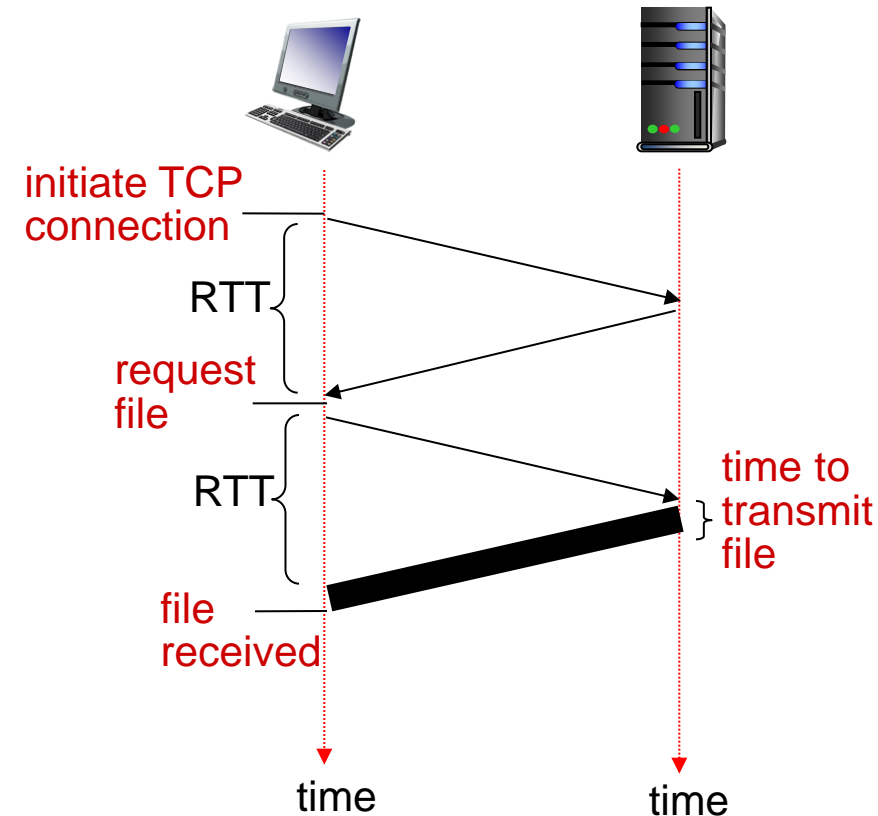


# Conexiones HTTP

**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 =  
 $2\text{RTT} + \text{tiempo de transmisión del archivo.}$



# 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.

		Method	Description
Fetch a page	→	GET	Read a Web page
		HEAD	Read a Web page's header
Used to send input data to a server program	→	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.

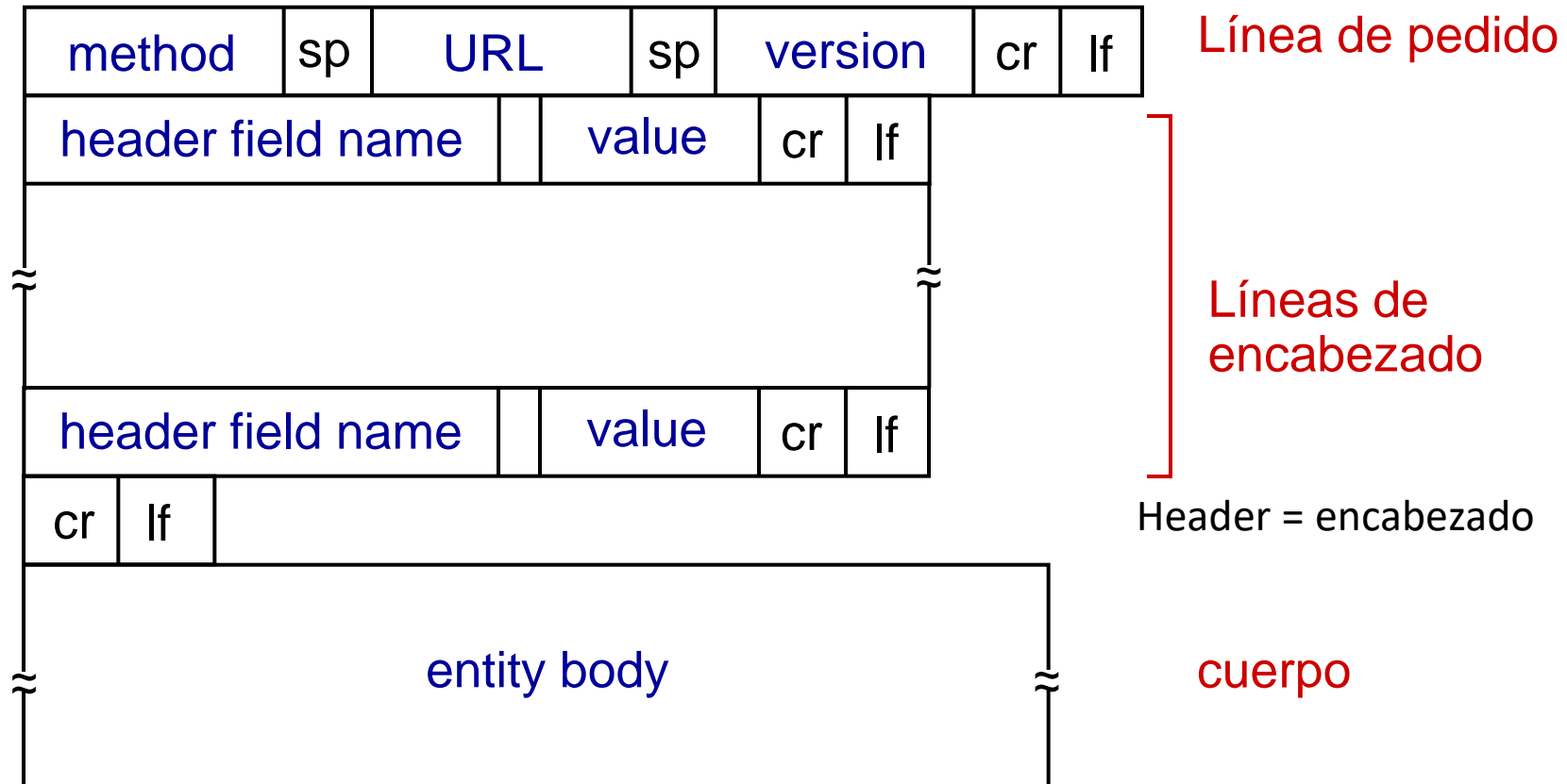
# Pedidos HTTP

- 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

request line  
(GET, POST,  
HEAD commands)

header  
lines

carriage return,  
line feed at start  
of line indicates  
end of header lines

carriage return character  
line-feed character

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

# 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](http://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

# HTML

- **HTML (lenguaje de marcado de hipertexto).**
  - ❑ HTML es el lenguaje estándar para crear páginas web
  - ❑ HTML describe la estructura de una página web
  - ❑ HTML indica al navegador como mostrar el contenido de la página
  - ❑ La **sintaxis** de HTML es muy *parecida* a la de ***XML***.
- **Cosas que se pueden representar con HTML:**
  - ❑ HTML permite producir páginas que incluyen texto, gráficos, hipervínculos, etc.
  - ❑ HTML también permite representar **tablas** y **formularios**

# HTML

Un documento HTML es una serie de **elementos**:

- Un **elemento** es contenido encerrado entre **etiquetas**.
- Una etiqueta tiene un **nombre**.
- Una etiqueta está demarcada entre '`<`' y '`>`'.
- P.ej. `<html>`, `<head>`, `<body>`, etc.
- Etiquetas pueden tener o no **atributos**.
- Un atributo tiene un **nombre** y un **valor** (que es un **string**) separados por '='.
- **P.ej:** atributo de nombre *href* tiene como valor un hiperenlace que es un URL.

## Algunos de los comandos de marcado más usados

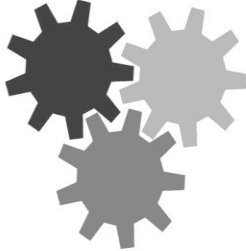
Tag	Description
<code>&lt;html&gt; ... &lt;/html&gt;</code>	Declares the Web page to be written in HTML
<code>&lt;head&gt; ... &lt;/head&gt;</code>	Delimits the page's head
<code>&lt;title&gt; ... &lt;/title&gt;</code>	Defines the title (not displayed on the page)
<code>&lt;body&gt; ... &lt;/body&gt;</code>	Delimits the page's body
<code>&lt;h n&gt; ... &lt;/h n&gt;</code>	Delimits a level <i>n</i> heading
<code>&lt;b&gt; ... &lt;/b&gt;</code>	Set ... in boldface
<code>&lt;i&gt; ... &lt;/i&gt;</code>	Set ... in italics
<code>&lt;center&gt; ... &lt;/center&gt;</code>	Center ... on the page horizontally
<code>&lt;ul&gt; ... &lt;/ul&gt;</code>	Brackets an unordered (bulleted) list
<code>&lt;ol&gt; ... &lt;/ol&gt;</code>	Brackets a numbered list
<code>&lt;li&gt;</code>	Starts a list item (there is no <code>&lt;/li&gt;</code> )
<code>&lt;br&gt;</code>	Forces a line break here
<code>&lt;p&gt;</code>	Starts a paragraph
<code>&lt;hr&gt;</code>	Inserts a Horizontal rule
<code>&lt;img src="..."&gt;</code>	Displays an image here
<code>&lt;a href="..."&gt; ... &lt;/a&gt;</code>	Defines a hyperlink

# HTML

```
<html>
<head><title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
 <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information </h2>
<ul>
  <li> <a href="http://widget.com/products/big"> Big widgets </a>
  <li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
  <li> By telephone: 1-800-WIDGETS
  <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(a)

## Welcome to AWI's Home Page



We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope *you* will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

---

### Product Information

- Big widgets
- Little widgets

### Telephone numbers

- 1-800-WIDGETS
- 1-415-765-4321

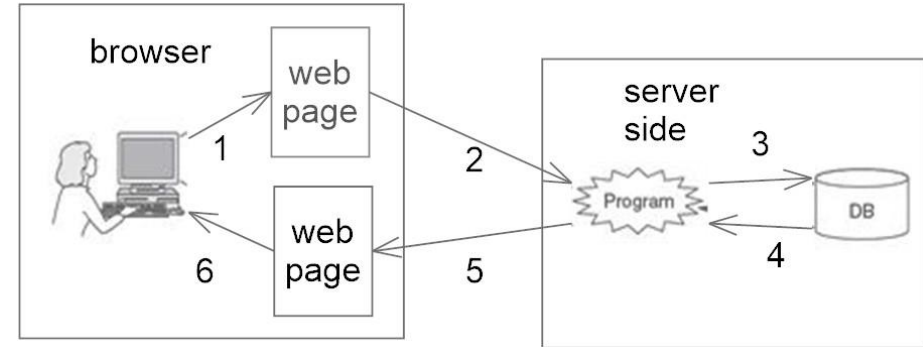
(b)

(a) HTML de una página de ejemplo. (b) La página formateada.

# Páginas dinámicas

- **Pasos para generar páginas dinámicas del lado del servidor:**

1. Un usuario llena un formulario y hace click en el botón de envío.
2. Se envía un mensaje al servidor web con el contenido del formulario.
  - Se proporciona el mensaje a un programa o una secuencia de comandos.
  - El programa procesa el mensaje.
3. El programa solicita información a un servidor de bases de datos.



4. El servidor de bases de datos responde con la información requerida.
5. El programa genera una página HTML personalizada y la envía al cliente.
6. El browser muestra la página recibida al usuario.



# HTML: Formularios

Etiquetas para definir **formularios HTML**

Tag	Description
<code>&lt;form action="", method =""&gt; ... &lt;/form&gt;</code>	Declara un formulario. Action es URL de la página ejecutable que procesa formulario. Method especifica cómo los datos se mandan al servidor (p.ej. GET, POST)
<code>&lt;select&gt; ... &lt;/select&gt;</code>	Para especificar una lista de la que usuario elige un elemento.
<code>&lt;option value = ""&gt; ... &lt;/option&gt;</code>	Para indicar opción de <select>
<code>&lt;textarea rows = "" cols=""&gt; ... &lt;/textarea&gt;</code>	Control de ingreso de texto de varias líneas
<code>&lt;input name="" type="" value=""&gt;</code>	Permite definir campo de input donde type puede ser: button, radio, password, text, submit, checkbox, hidden, etc.

# Formularios HTML

- (a) Formulario HTML para una orden de compra.
- (b) La página formateada.

Una respuesta posible desde el navegador al servidor con la información llenada por el usuario.

Los parámetros ingresados por el usuario se envían del navegador al servidor de la siguiente forma:

customer=John+Doe&address=100+Main+St.&city=White+Plains&  
state=NY&country=USA&cardno=1234567890&expires=6/98&cc=mastercard&  
product=cheap&express=on

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/widgetorder" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street Address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```

(a)

## Widget Order Form

Name	<input type="text"/>		
Street address	<input type="text"/>		
City	<input type="text"/>	State	<input type="text"/>
		Country	<input type="text"/>
Credit card #	<input type="text"/>	Expires	<input type="text"/>
		M/C	<input type="radio"/>
		Visa	<input type="radio"/>
Widget size	Big	<input type="radio"/>	Little
		<input type="radio"/>	Ship by express courier
			<input type="radio"/>
<input type="button" value="Submit order"/>			

(b)

Thank you for ordering an AWI widget, the best widget money can buy!

# Páginas dinámicas

- **Páginas dinámicas:** Son páginas web generadas por programas que se ejecutan en el servidor (posiblemente con una base de datos).
- **Tareas que suelen hacer las páginas dinámicas:**
  - Procesar parámetros de formularios
  - Procesar encabezados de pedido HTTP
  - Pedir datos a fuentes de datos (e.g. bases de datos)
  - Generar página web con los datos recibidos.
  - Generar encabezados de respuesta HTTP
- **Tecnologías para producir páginas dinámicas**
  - PHP, Java Server Pages , etc.
- Estudiaremos a continuación un poco de PHP.

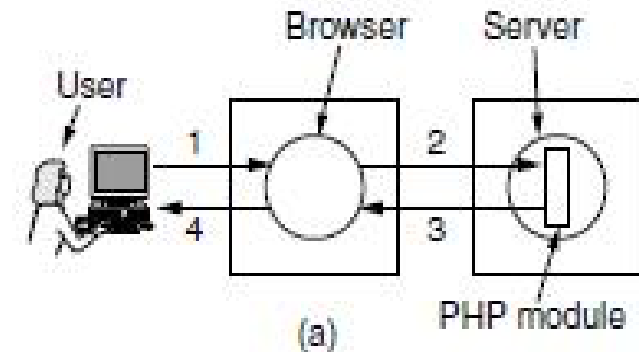
# PHP

- **Enfoque PHP (Preprocesador de Hipertexto).**

- ☐ Se definen páginas dinámicas mediante la **inserción de comandos especiales dentro de páginas HTML**

- ☐ Para utilizar PHP el servidor web tiene que entender PHP.

- El código PHP es interpretado por un servidor web.
- PHP se diseñó para trabajar con el servidor web Apache.
- PHP puede ser usado en la mayoría de los servidores web.



# PHP

- **Algunas cosas que puede hacer PHP:**
  - PHP puede generar contenido de página dinámica
  - PHP puede operar con archivos en el servidor.
  - PHP puede recolectar datos de formulario
  - PHP puede enviar y recibir cookies
  - PHP puede acceder a encabezados de pedido HTTP
  - PHP permite definir encabezados de respuesta HTTP
  - PHP permite acceder a base de datos
- PHP es gratuito y fácil de aprender y se ejecuta eficientemente.

# PHP

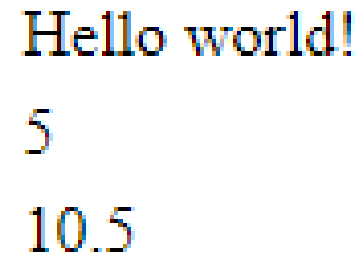
Un poquito de sintaxis fundamental de PHP

Construcción	Description
<?php ... ?>	PHP script Puede ir en cualquier lugar del documento
'\$' NAME	Declaración de variable Case sensitive
echo EXPR	Para mostrar datos en pantalla
//	comentarios
Define(name, value)	Definición de constantes
VARIABLE = EXPR	Igual que en C (+=, -=, *=, /=)
Include 'filename'	Toma el texto/código/markup en un archivo y lo copia en el archivo que usa la sentencia <i>include</i>

# PHP

```
<!DOCTYPE html>
<html>
  <body>
    <?php
      $txt = "Hello world!";
      $x = 5;
      $y = 10.5;
      echo $txt;
      echo "<br>";
      echo $x;
      echo "<br>";
      echo $y;
    ?>
  </body>
</html>
```

Resultado mostrado en el browser



Hello world!

5

10.5

# PHP

- **Acceso a campos de formularios:**

- `$_POST`: usado para recolectar datos de formulario luego de someter un formulario con método POST.
- P.ej: `$_POST['fname']` // recolecta valor del campo de nombre 'fname'
- `$_GET`: usado para recolectar datos de formulario luego de someter un formulario con método GET. Se usa como en el ítem anterior.



# PHP

Página web que  
obtiene input de  
formulario y llama un  
programa de  
servidor

```
<html><body>
  <form action="welcome.php" method="post">
    Name: <input type="text" name="name"><br>
    E-mail: <input type="text" name="email"><br>
    <input type="submit">
  </form>
</body> </html>
```

Programa PHP del  
servidor que crea  
una página web  
personalizada

```
<html><body>
  Welcome <php echo $_POST["name"]; ?><br>
  Your email address is: <?php echo $_POST["email"]; ?>
</body></html>
```

Página web resultante  
(para inputs "John" y  
"john.doe@example.com")

```
<html><body>
  Welcome John
  Your email address is john.doe@example.com
</body></html>
```

# PHP

- **Tipos de datos de PHP**

- String: secuencia de caracteres entre comillas.
- Integer: número entero entre - 2,147,483,648 y 2,147,483,647
- Float: número con punto decimal o número en forma exponencial.
- Boolean: valores booleanos TRUE y FALSE
- Array: un arreglo almacena varios valores en una variable.
- Object: PHP permite definir clases y objetos

- **Operadores**

- de comparación (como en C), de asignación (como en C)
- Para los distintos tipos de datos

# PHP

- **Acceso a información de encabezados HTTP:**

- ❑ `$_SERVER`: contiene información de encabezados, caminos y localización de scripts.
- ❑ Para acceder a encabezados poner como argumento alguna de las siguientes:
  - `HTTP_USER_AGENT`, `SERVER_ADDR`, `SERVER_NAME`, `SERVER_SOFTWARE`, `SERVER_PROTOCOL`, `REQUEST_METHOD`, `REQUEST_TIME`, `QUERY_STRING`, `HTTP_ACCEPT`, `HTTP_ACCEPT_CHARSET`, `HTTP_HOST`, etc.
- ❑ P.ej.: Para acceder al encabezado User-Agent: `$_SERVER['HTTP_USER_AGENT']`
- ❑ p.ej.: nombre del archivo del script ejecutándose: `$_SERVER['PHP_SELF']`
- ❑ P.ej.: URL completo de la página corriente: `$_SERVER['HTTP_REFERER']`
- ❑ P.ej.: el camino del script corriente: `$_SERVER['SCRIPT_NAME']`
- ❑ P.ej.: el nombre de dominio del host servidor: `$_SERVER['SCRIPT_NAME']`
- ❑ P.ej.: el encabezado Host del pedido actual: `$_SERVER['HTTP_HOST']`

# PHP

- ```
<!DOCTYPE html>
<html>
<body>
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
</body>
</html>
```

Resultado mostrado en el browser

```
/demo/demo_global_server.php
35.194.26.41
35.194.26.41
https://tryphp.w3schools.com/showphp.php?
filename=demo_global_server
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.90 Safari/537.36
/demo/demo_global_server.php
```

# PHP

- **Definición de encabezados de respuesta HTTP:**

- Hay que usar la función header()
- Se deben fijar encabezados antes de la etiqueta <html> aparezca.

- P.ej:

```
<?php header('Content-Type: text/plain');  
        header('Expires: Fri, 18 Jan 2002 05:30:00 GMT'); ?>
```

```
<html>
```

```
<body>
```

```
...
```

# PHP

- **Definición de cookies:**

- Setcookie() define cookie para ser enviada junto con el resto de los encabezados HTTP.
- Esta función debe usarse antes de generar cualquier salida, o sea antes que la etiqueta <html>
- Un cookie se crea con la función:  
    setcookie(name, value, expire, path, domain, secure, httponly)
- ```
<?php
    $cookie_name = "user";
    $cookie_value = "Alex Porter";
    setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
```

# PHP

- **Acceso al valor de una cookie:**

- ☐ \$\_COOKIE se usa para retornar el valor de una cookie.

- ☐ P.ej:

```
<?php
    echo '¡Hola ' . htmlspecialchars($_COOKIE["nombre"]). '!';
?>
```

- ☐ Asumiendo que la cookie "nombre" ha sido definida anteriormente, si el valor de esa cookie es “Juan” el resultado del ejemplo será :

- ☐ ¡Hola Juan!

- ☐ htmlspecialchars — Convierte caracteres especiales en entidades HTML

# PHP: uso de include

- En “footer.php” tenemos:
- ```
<?php  
echo "<p>Copyright &copy; 1999-" .  
date("Y") . " W3Schools.com</p>";  
?>
```
- Tenemos el siguiente script:
- ```
<!DOCTYPE html>  
<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
<?php include 'footer.php';?>  
  
</body>  
</html>
```

Resultado mostrado en el browser

**Welcome to my home page!**

Some text.

Some more text.

Copyright © 1999-2021 W3Schools.com