Transparencias

de

J2EE

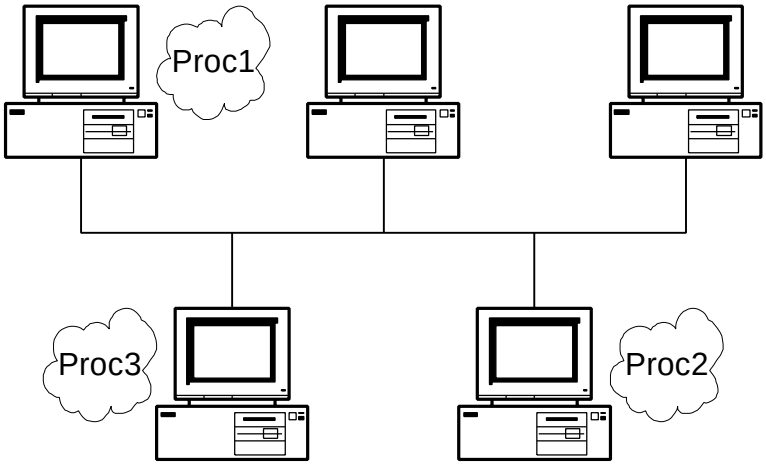
Tema 1: Introducción

w TEMA 1: COMPUTACIÓN DISTRIBUIDA

◆ ¿QUÉ ES LA COMPUTACIÓN DISTRIBUIDA?

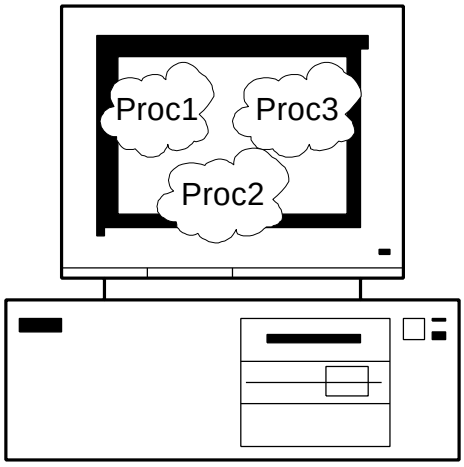
✶ La computación distribuida implica el reparto y la ejecución de una serie de procesos a lo largo de varios nodos de una red.

Aplicación Distribuida



App = Proc1 + Proc2 + Proc3

Aplicación Centralizada

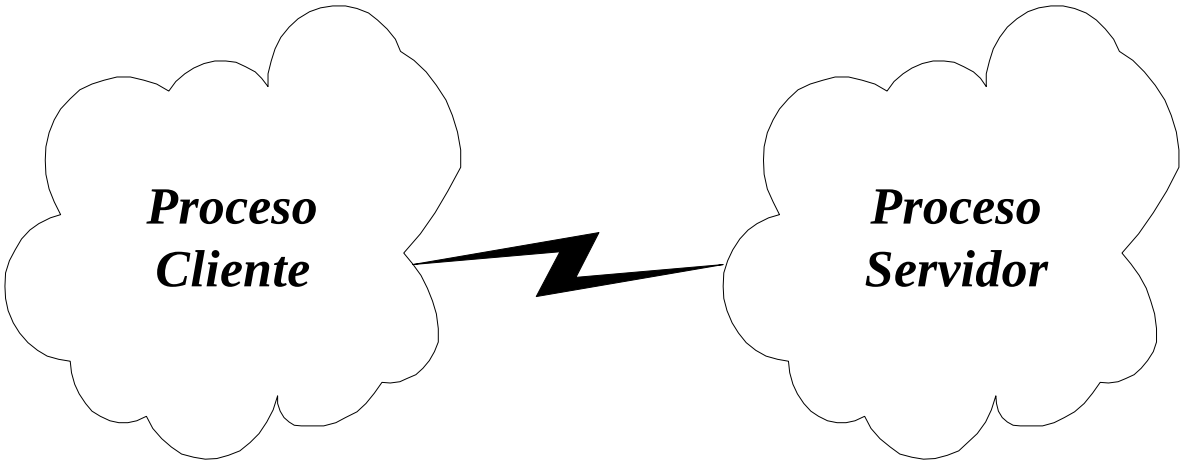


✶ Tipo concreto de aplicación distribuida ➔ Aplicación Web

◆ ARQUITECTURA CLIENTE-SERVIDOR

✶ Termino Cliente/Servidor

- ❑ Define un tipo de relación entre procesos proveedores de servicios y procesos consumidores



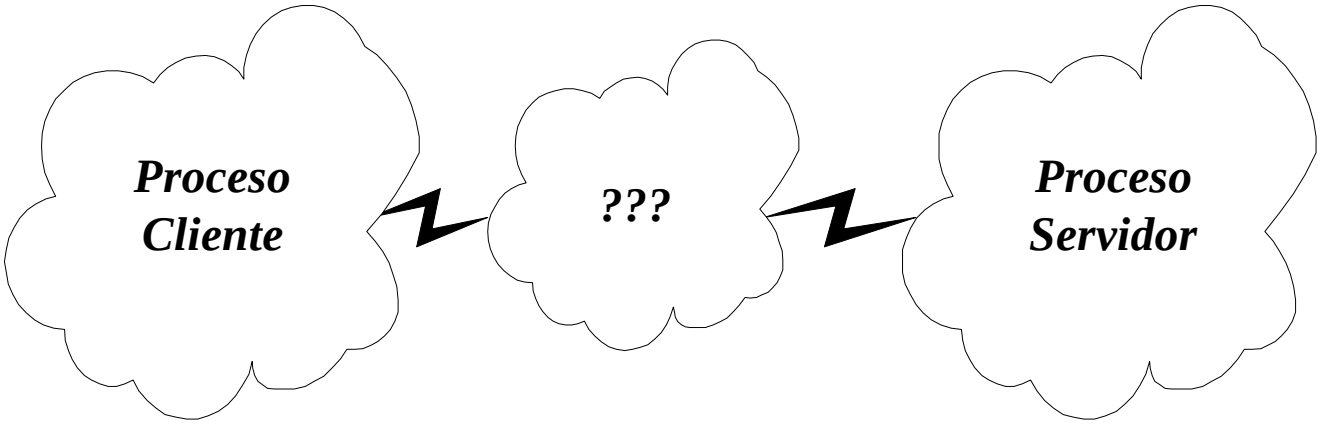
Solicita Servicios

Proporciona Servicios

- ❑ ¡Ojo! Un tipo de relación entre procesos, no entre máquinas

✶ Arquitectura Cliente/Servidor

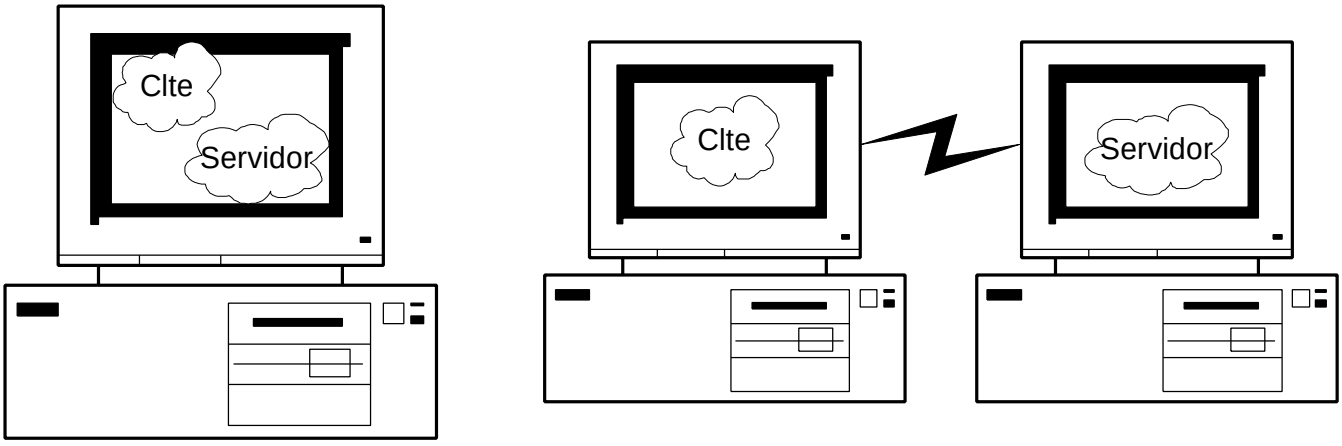
- ❑ Arquitectura base de las aplicaciones distribuidas
- ❑ Divide la funcionalidad de las aplicaciones en dos partes.



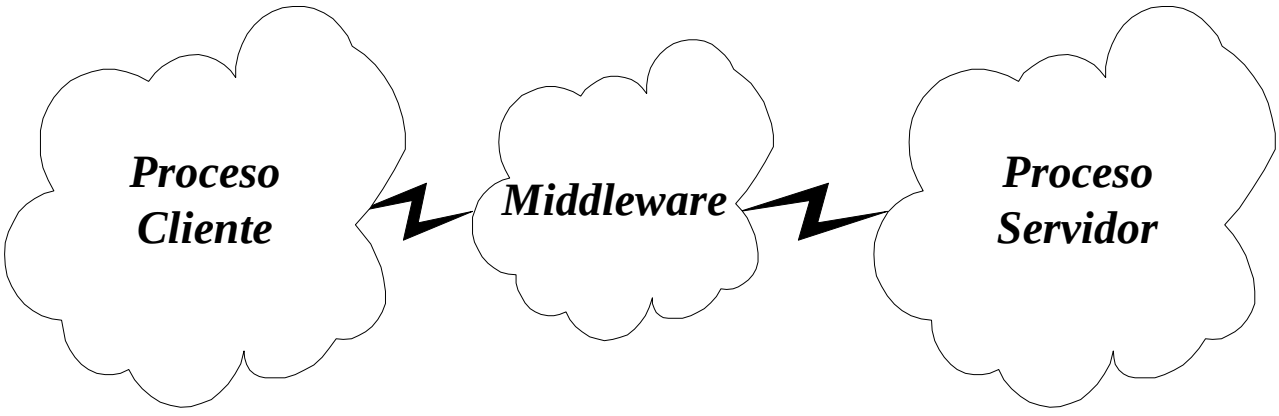
Interacción con el usuario
Solicitar servicios al servidor
Presentación de los resultados al usuario

Proporcionan servicios a los clientes
Generalmente sin GUI

- Cliente ➔ Visualización de la información al usuario
  - Se clasifican según su interfaz de usuario: sin GUI, con GUI, GUIOO
- Servidor ➔ Prestación de servicios
  - Se clasifican según la naturaleza de los servicios que proporcionan:
    - ✓ Servidores BD, Servidores web,...
- Distinción entre procesos clientes y servidores; y máquinas donde se ejecutan esos procesos (conlleva ambigüedad)
- Puede haber separación física entre los procesos o no.

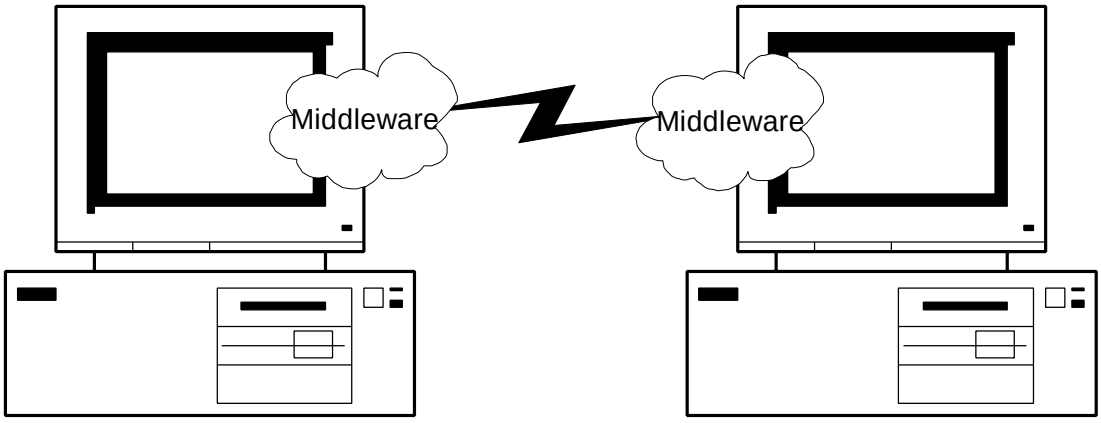


- Tercer elemento de la arquitectura C/S ➔ MIDDLEWARE
  - Infraestructura (capa software) que se ubica entre el cliente y el servidor.



  - Un conjunto de servicios independientes del negocio o dominio de la aplicación.
  - App Estadísticas – App Facturación ➔ Middleware no cambia
  - Proporciona todas las facilidades necesarias para la comunicación entre el proceso cliente y servidor.

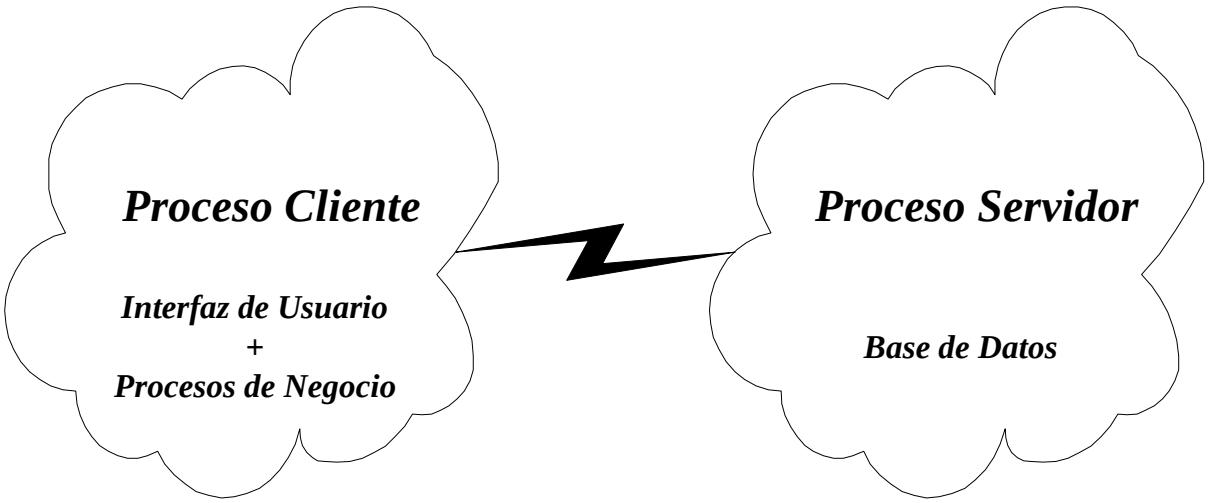
- Gestiona todos los aspectos necesarios para la comunicación entre el proceso cliente y el servidor a través de la red, liberando al programador de todos los aspectos relacionados con los protocolos de red y dedicándose únicamente de programar los procesos de negocio de la aplicación.
- En realidad una parte se ubicará en la máquina cliente y otra en la máquina servidora.



Máquina Cliente Máquina Servidora

◆ EL PESO DE LOS CLIENTES Y LOS SERVIDORES

- ✶ Cantidad de código invertida en cada uno de los extremos de una arquitectura C/S
- ✶ Cliente Pesado – Servidor Ligero



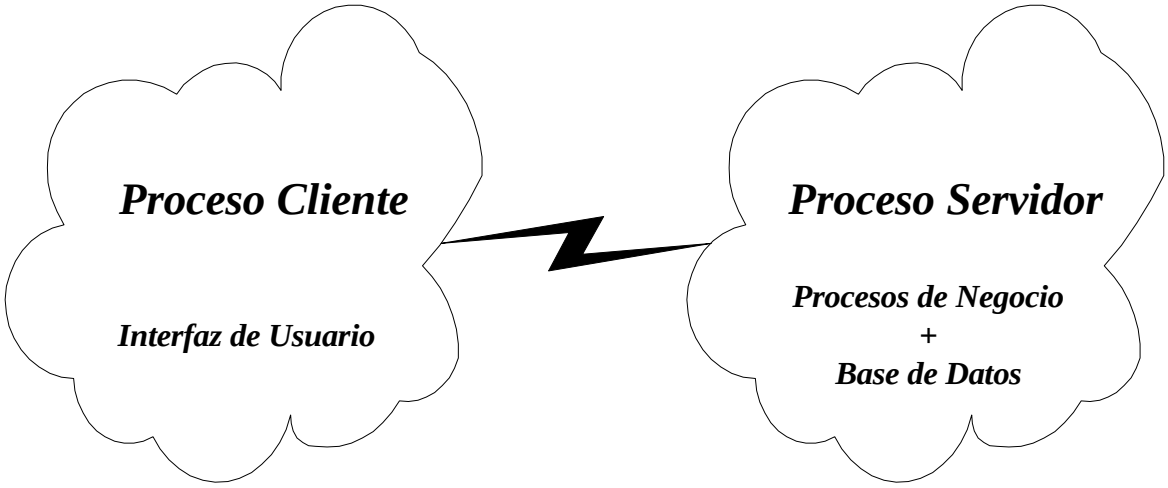
Interacctua con el usuario
Procesa los datos devueltos por el servidor
Busquedas, estadísticas,...
Presenta los resultados al usuario

Almacena la información
Responde consultas SQL

- Ejemplo: Aplicaciones desarrolladas con JBuilder
- Difícil mantenimiento ➔ un cambio en los procesos de negocio de la aplicación implica un cambio de todos los clientes ➔ afecta a miles de máquinas
  - Ejemplo: Impuestos 16% ➔ 15%

    Pesetas ➔ Euros

✶ Cliente Ligero – Servidor Pesado



Interactua con el usuario
Recibe los datos refinados del servidor (resultados)
Los presenta al usuario

Almacena los datos
Realiza los accesos a la BD (SQL)
Busqueda y tratamiento de los datos, estadísticas,...

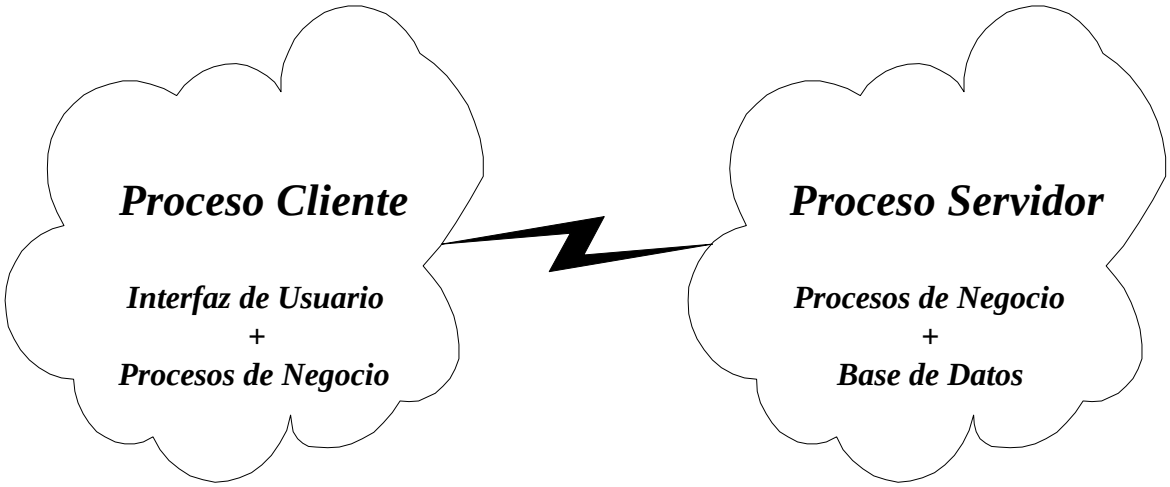
- Mejor mantenimiento ➔ un cambio en los procesos de negocio de la aplicación sólo afecta al servidor ➔ afecta a pocas máquinas.
  - Ejemplo: Impuestos 16% ➔ 15%

    Pesetas ➔ Euros

✶ Ejemplo: Elaboración de Estadísticas

- Cliente pesado y servidor ligero
  - El servidor devuelve los datos al cliente ➔ Todos los Datos
  - El cliente trata los datos y elabora las estadísticas
  - El cliente presenta los resultados al usuario

- Cliente ligero y servidor pesado
  - El servidor trata los datos y elabora la estadística
  - El cliente presenta los resultados al usuario
- Procesos de Negocio = Elaboración de estadística
  - En función de donde "caigan" los procesos de negocio estamos ante una configuración de cliente ligero y servidor pesado o cliente pesado y servidor ligero.
- También existen CONFIGURACIONES MIXTAS



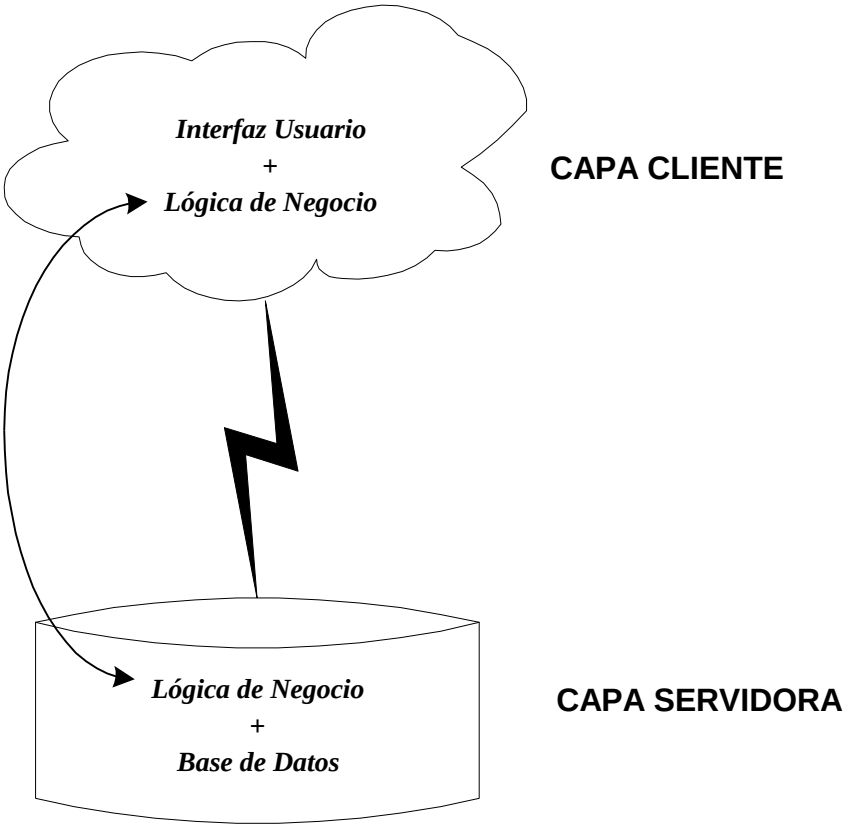
◆ ARQUITECTURA C/S DE 3 CAPAS

✶ Arquitectura de Aplicación: estructura de las partes en que se divide una aplicación y las relaciones entre ellas.

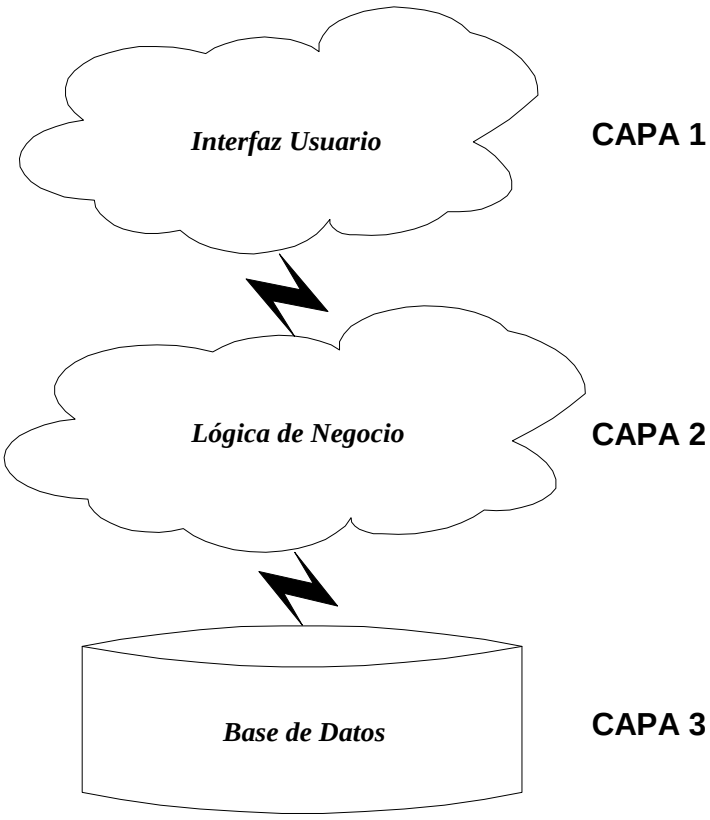
✶ División de una aplicación compleja en capas o unidades funcionales que nos permita abordar su desarrollo más fácilmente ➔ Divide y Venceras

✶ Arquitectura C/S básica ➔ Arquitectura C/S de 2 capas

- Capa cliente
- Capa servidora
- Configuraciones ➔ Cliente pesado y Servidor ligero
  ➔ Cliente ligero y Servidor pesado
  ➔ Mixta



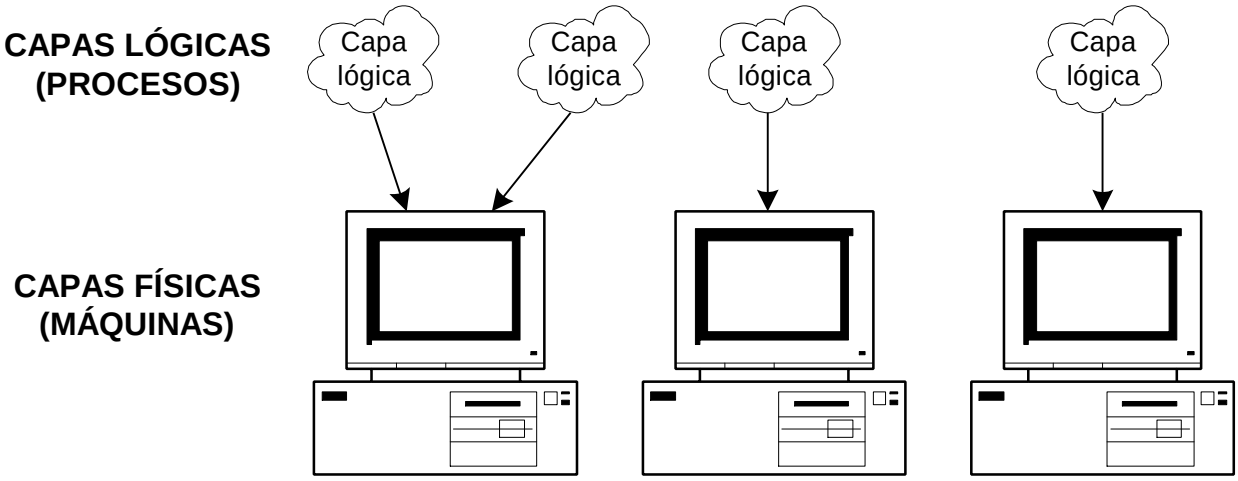
- Actualmente ➔ Arquitectura C/S de 3 capas
  - Capa de Interfaz de Usuario
  - Capa de Lógica de Negocio
  - Capa de Base de Datos



- Los procesos de negocio en lugar de encontrarse bailando entre la capa cliente y la servidora, se ubican en una capa independiente entre la Interfaz y la Base de Datos.
- Relaciones C/S entre cada una de las capas
- Ventajas ➔ Capas Independientes
  - Más robustas y fáciles de mantener
  - Mejor comportamiento frente a cambios
  - Más fácilmente distribuible

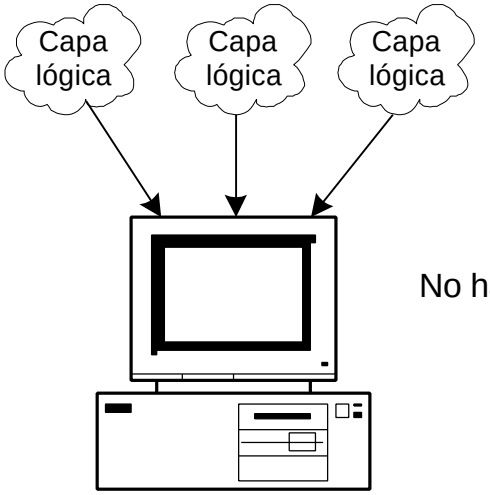
✶ También ARQUITECTURAS DE 4 ó N CAPAS LÓGICAS

✶ Asignación de capas lógicas a distintas configuraciones físicas ➔ Arquitecturas de 1, 2, 3 ó N capas físicas



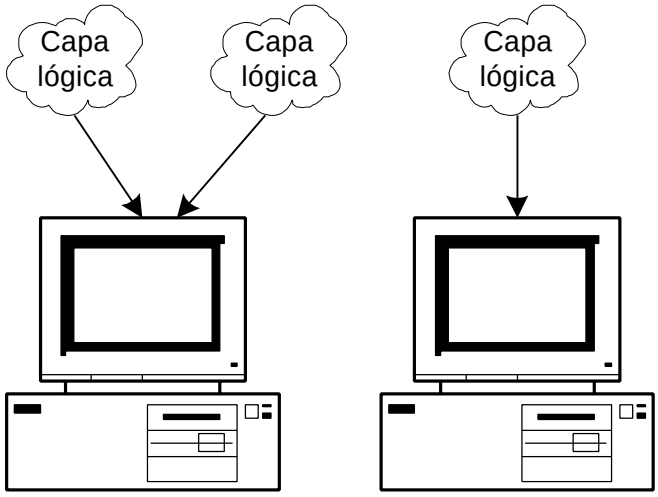
Ejemplo: Arquitectura de 4 capas lógicas y 3 capas físicas

✶ 1 Capa Física

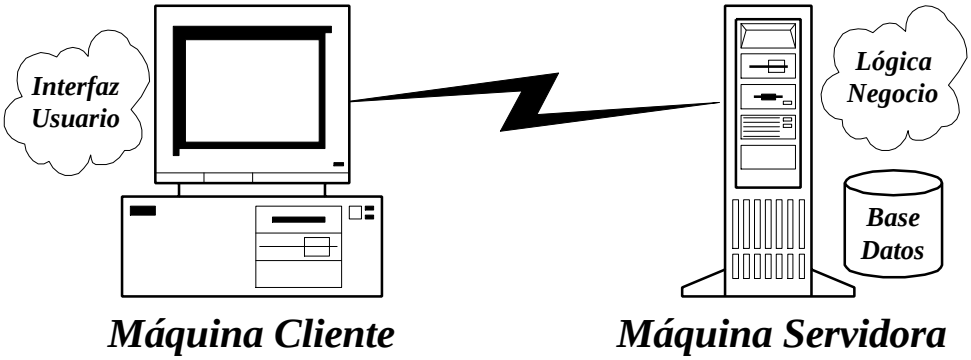


No hay distribución

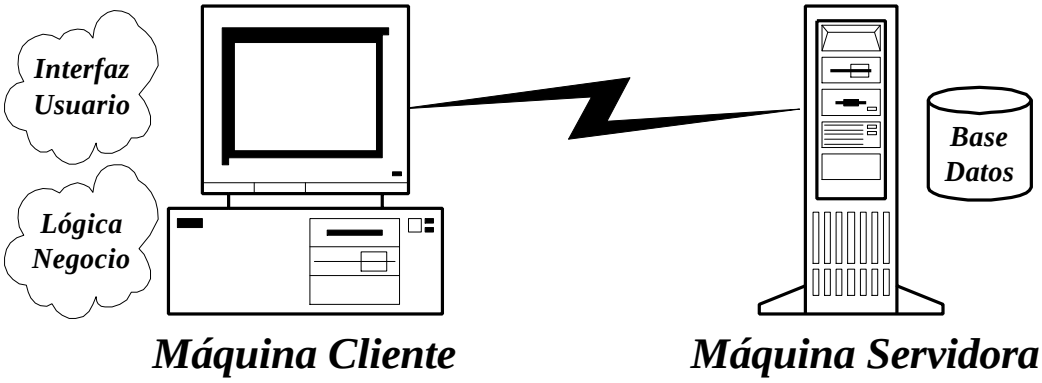
✶ 2 Capas Físicas



- En función de cómo se distribuya cada capa lógica...
  - Configuración de cliente ligero y servidor pesado

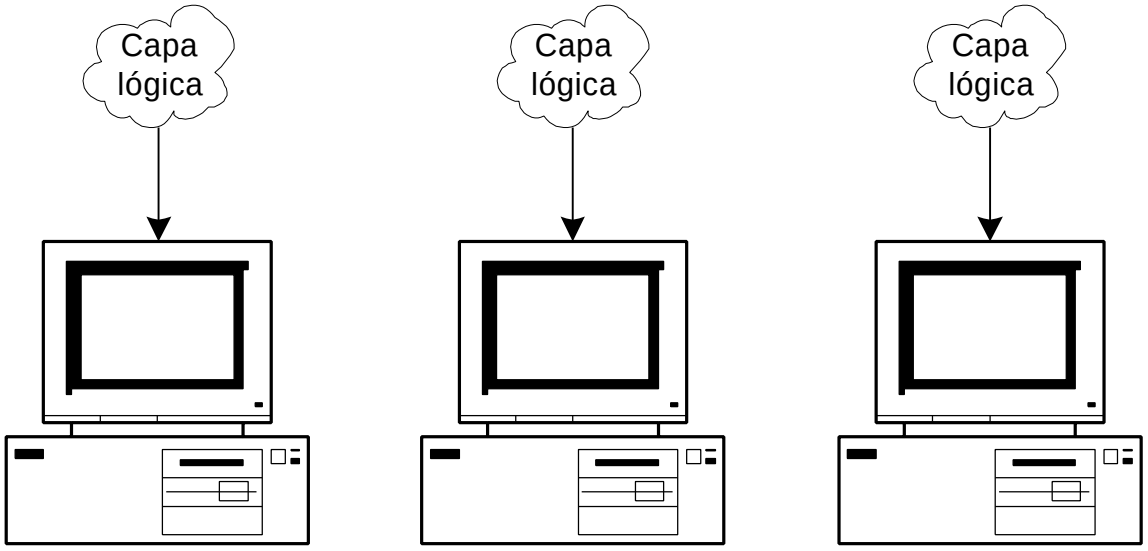


  - Configuración de cliente pesado y servidor ligero

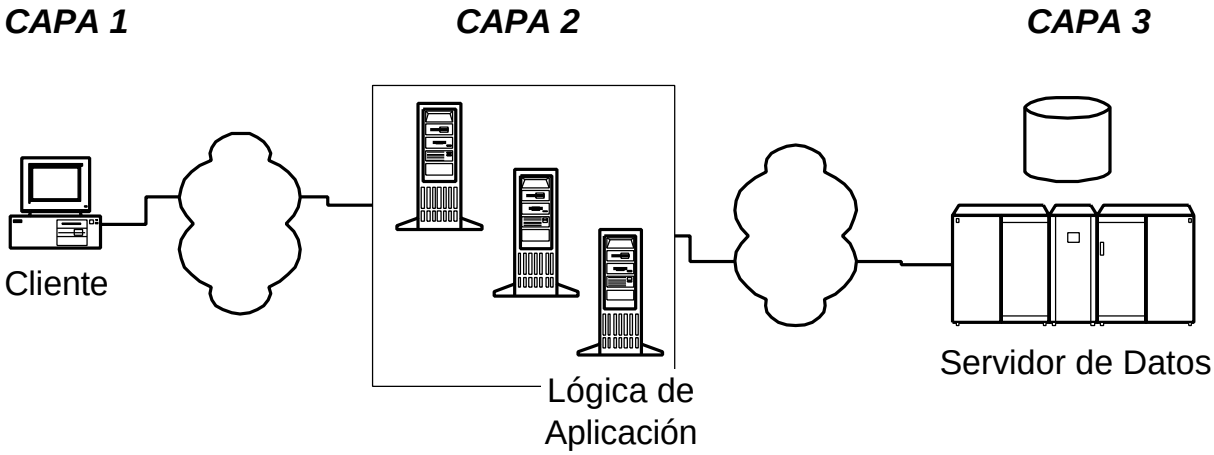


- ... desde el punto de vista de las máquinas

- 3 Capas Físicas



- Posibilidad de subdividir la capa de lógica de negocio en más capas logicas y obtener mayores niveles de distribución ➔ ARQUITECTURAS DE N CAPAS FÍSICAS



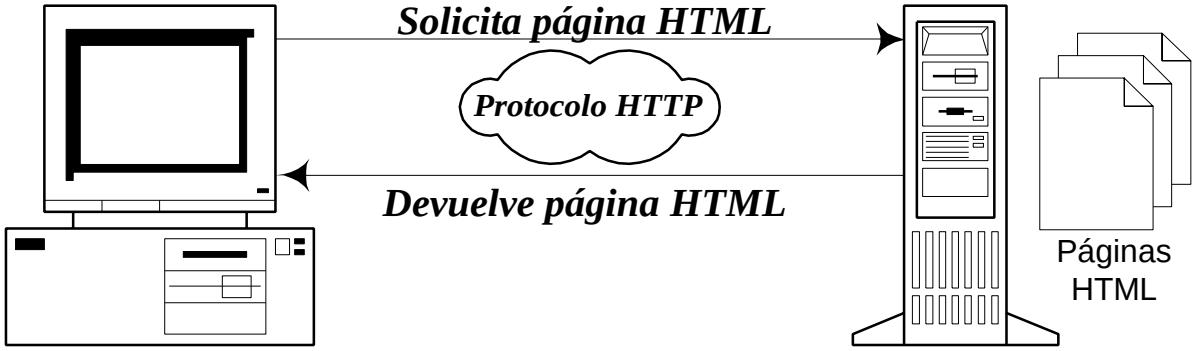
◆ ARQUITECTURA DE LAS APLICACIONES WEB

- ✶ Aplicación Web ➔ Tipo particular de aplicación distribuida

  ➔ Objetivo de nuestra asignatura

- ✶ Las aplicaciones web se basan en la arquitectura C/S

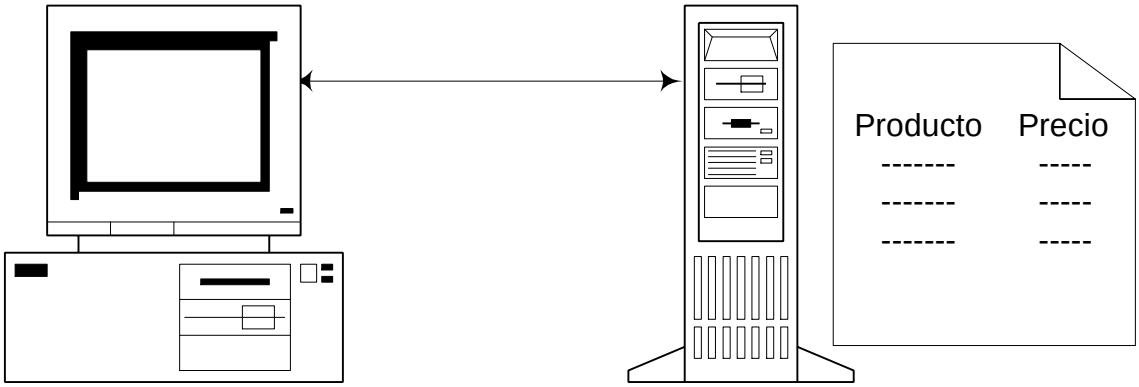
◆ FUNCIONAMIENTO TRADICIONAL DE LA WEB



- ✶ Funcionamiento básico Web ➔ Arquitectura C/S
  - ❑ Navegador ó Cliente Web
    - ■ Solicita una página HTML al servidor web mediante http
    - ■ Visualiza la página HTML al usuario
    - ■ Ejemplo claro de cliente ligero ➔ Sólo se encarga de la visualización de las páginas HTML al usuario.
  - ❑ Servidor Web
    - ■ Tradicionalmente ➔ Servir páginas HTML (un simple servidor de fich.)
    - ■ Actualmente ➔ Permite la ejecución de procesos

◆ TENDENCIAS ACTUALES PARA LAS APP. WEB

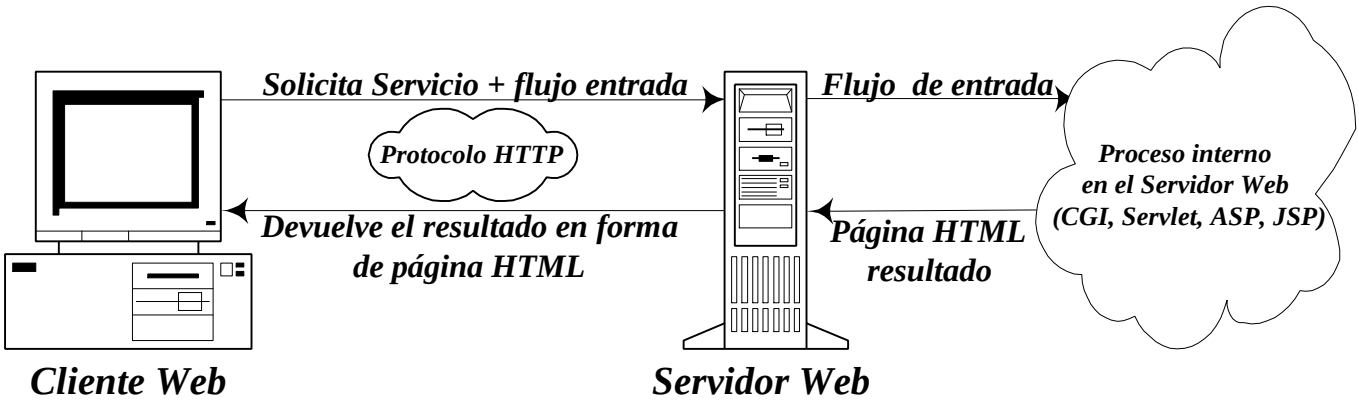
- ✶ Problema del funcionamiento tradicional de la web:
  - ❑ Su naturaleza estática
    - ■ Simple repositorio estático de páginas web
    - ■ No existe interacción con el usuario ➔ el usuario no puede enviar datos y recibir una respuesta personalizada
  - ❑ Ejemplo: Tienda de Deportes



Navegador ***Servidor Web***

  - ❑ Cada vez que cambia el precio debemos modificar la página web ➔ Mantenimiento complicado
- ✶ Solución:
  - ❑ Añadir más inteligencia al cliente (1ª solución)
  - ❑ Añadir más inteligencia al servidor(2ª solución)
- ✶ 1ª Solución
  - ❑ JavaScript o Applets incrustados en páginas HTML
    - ■ Procesos que se ejecutan en el cliente y que permiten dar un mayor dinamismo a la web
- ✶ 2ª Solución
  - ❑ CGI / Servlets ➔ Tecnologías de “bajo nivel”

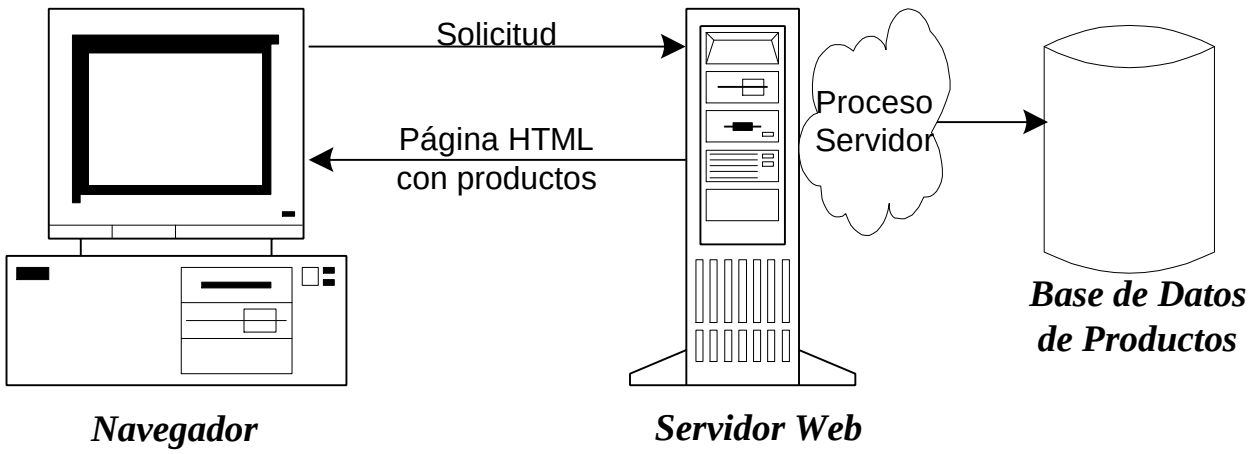
- ASP / JSP ➔ Tecnologías de “alto nivel”
- Característica común
  - Son procesos que residen en el servidor y que se ejecutan en él
  - Reciben una petición, la procesan y generan como resultado una página HTML
  - Resumiendo ➔ Procesos que generan dinámicamente páginas HTML



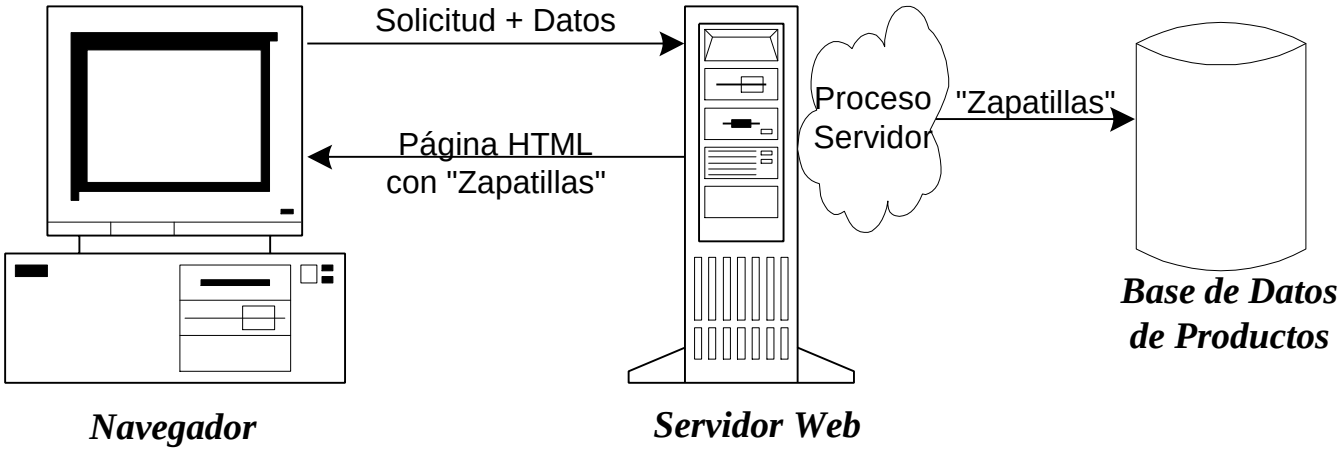
MAYOR INTERACCIÓN
CON EL USUARIO

- Ejemplo: Tienda de Deportes

  En lugar de tener una página HTML con los artículos, tenemos un proceso en el servidor que genera la página HTML dinámicamente.



  O bien mediante un formulario seleccionamos que queremos ver los artículos de tipo “Zapatillas”

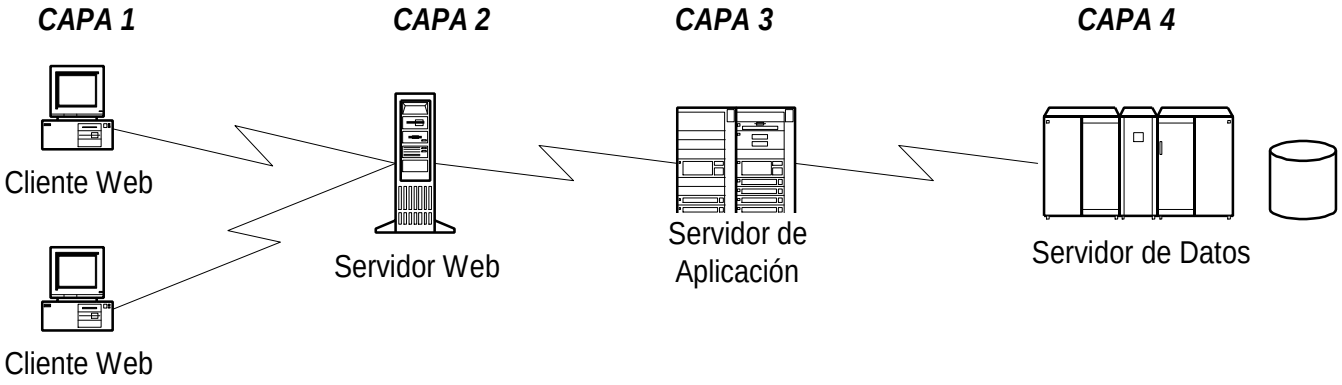


- ✶ Evolución de las aplicaciones web:

  Repositorio de Páginas HTML ➔ Procesos que reciben peticiones y generan páginas HTML dinámicamente

◆ ARQUITECTURA WEB ACTUAL

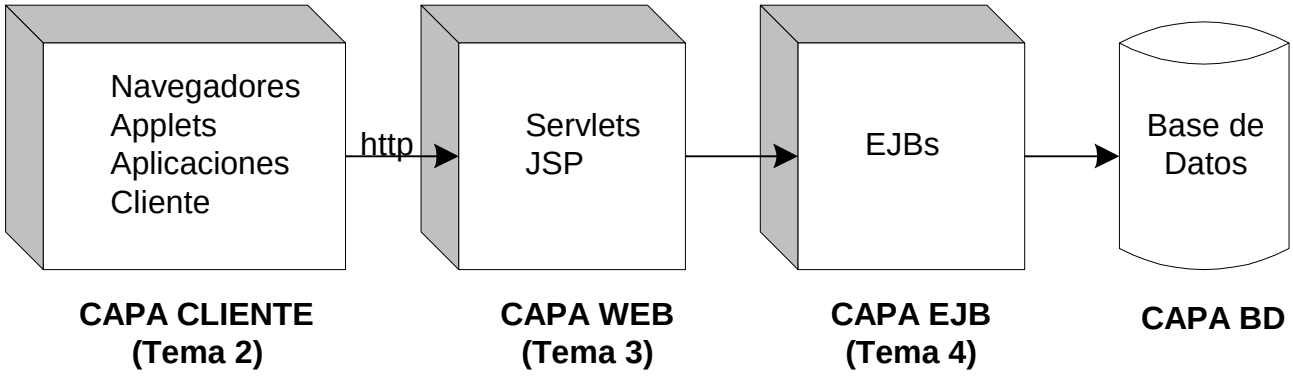
- ✶ Arquitectura C/S de 2 capas ➔ Arquitectura C/S 'N' capas



◆ ARQUITECTURA DE LA PLATAFORMA J2EE

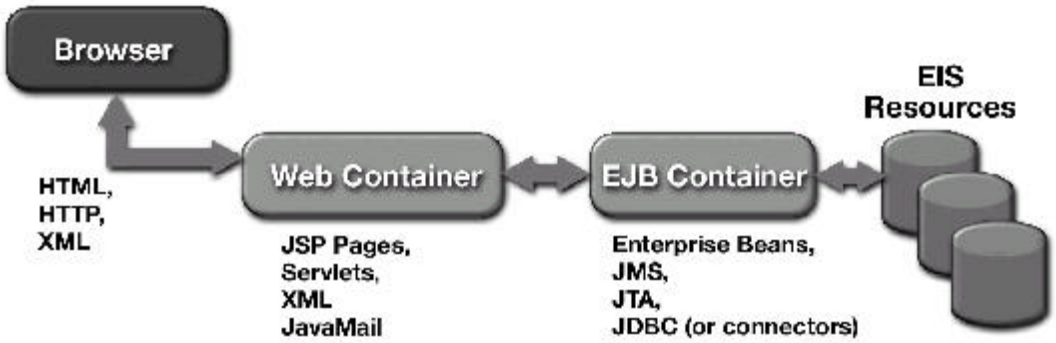
- ✶ Plataforma J2EE (Java 2 SDK Enterprise Edition)
  - ❑ Es un estándar para el desarrollo de aplicaciones distribuidas multicapa basado en la plataforma Java

★ Propone una arquitectura C/S de 4 capas

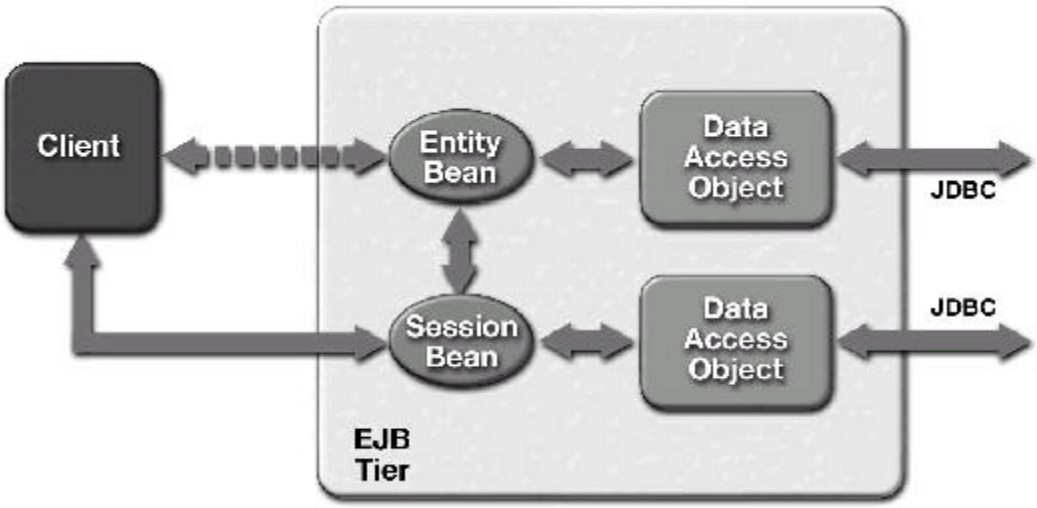


- Capa Cliente ➔ Presenta al usuario las páginas HTML generadas por la capa web (GUI)
- Capa Web ➔ Genera dinámicamente páginas HTML
  ➔ Soporta la lógica de presentación
- Capa EJB ➔ Contiene componentes EJB que soportan la lógica de negocio de la aplicación
- Capa BD ➔ Almacena la información

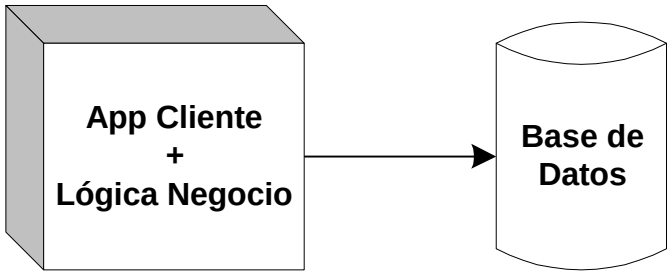
★ Partiendo de esta arquitectura base ➔ Distintas opciones:

- Arq. C/S de 4 capas (completa) ➔ versión web ➔ proyecto
- 

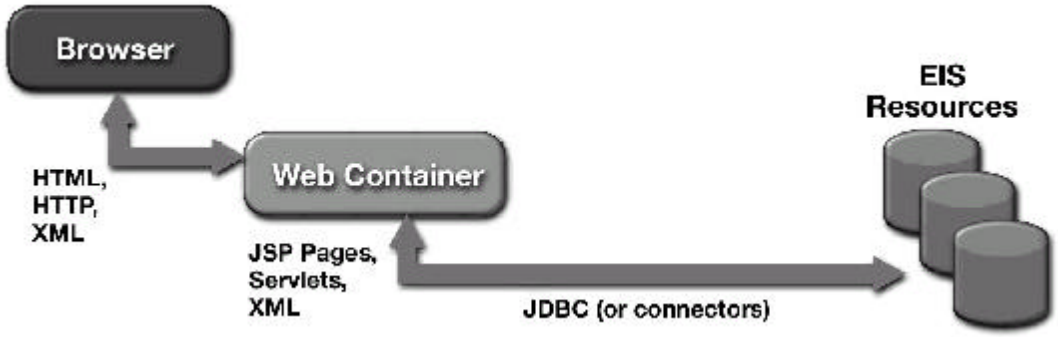
Arq. C/S de 3 capas ➔ versión no web ➔ prácticas EJB



- Arq. C/S de 2 capas ➔ versión no web



- Arq. C/S de 3 capas ➔ versión web ➔ practicas JSP



  - Ejemplo de la tienda de deportes

◆ TECNOLOGÍAS WEB

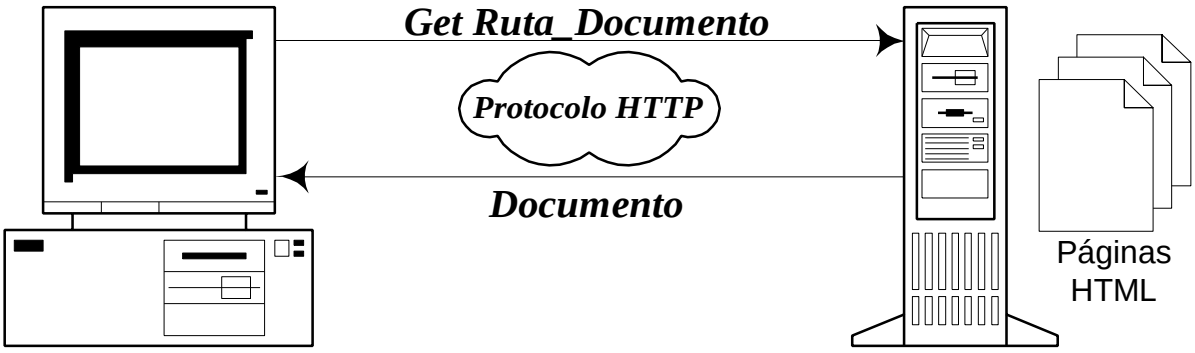
✶ Posibilidades tecnológicas para el desarrollo de aplicaciones web.

◆ PROTOCOLO HTTP

✶ Protocolo que permite la recuperación de documentos de hipertexto.

- ❑ Documento de hipertexto ➔ un fichero como cualquier otro

✶ Por extensión ➔ Protocolo para la transferencia de ficheros orientado a la web



- ❑ GET ➔ Solicita recurso
- ❑ POST ➔ Envía datos

✶ URL ➔ Indica la localización de cualquier recurso en la red (Universal Resource Locator)

- ❑ Ejemplo:

http://www.eside.deusto.es/asignaturas/LSD/pagina.html

- ❑ http ➔ protocolo a usar para la recuperación del documento
- ❑ www.eside.deusto.es ➔ máquina donde se ubica el documento

- :port ➔ puerto del proceso que sirve el documento
- /asignatura/LSD ➔ ruta del documento dentro de la máquina
- pagina.html ➔ documento a recuperar

✶ El navegador transforma la URL que escribe el usuario en una petición mediante el protocolo http

URL ➔ Petición HTTP

◆ LENGUAJE HTML

✶ Lenguaje para la creación de páginas web

✶ Conjunto de marcas (o etiquetas) que indican el formato de la información de la página web.

✶ Describe como va a ser la página web.

- Ejemplo:

<I>Hola</I> ➔ *Hola*

- Otro ejemplo:

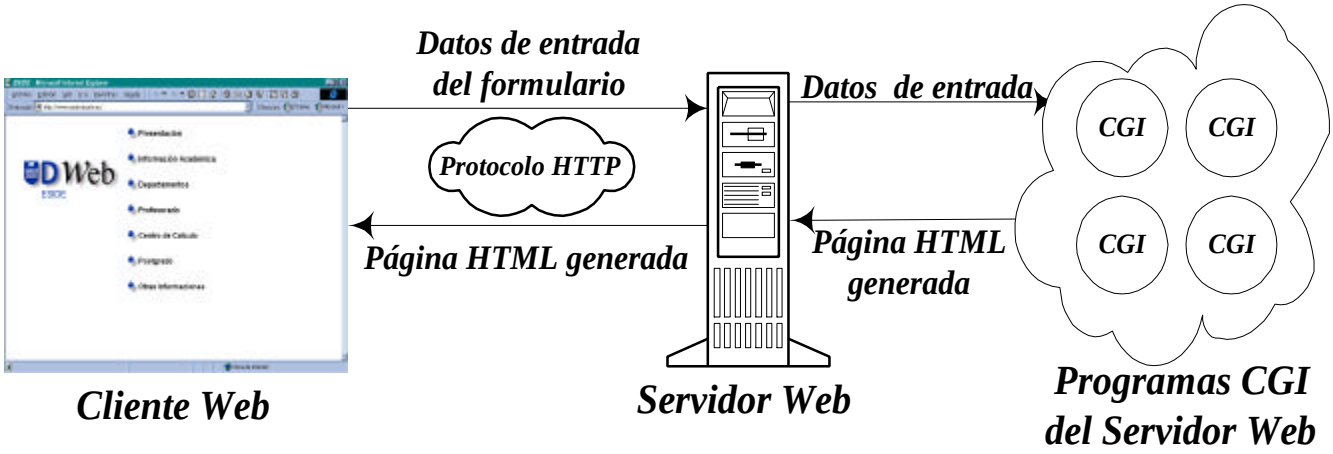
```
<HTML>
<HEAD>
<TITLE>Página de ejemplo</TITLE>
</HEAD>
<BODY>
<P>Este
<A HREF= "http://www.eside.deusto.es/asignaturas/LSD"> enlace </A>
lleva muy lejos. También hay una tabla con una imagen.
</P>
<TABLE WIDTH="50%" BORDER="1">
   <TR>
      <TD>La imagen <B>está aqui</B>:</TD>
      <TD><IMG SRC="imagenes/ud.gif"></TD>
   </TR>
</TABLE>
</BODY>
</HTML>
```

✶ Lenguaje interpretado por el navegador.

✶ Se verá en el tema 2

◆ CGI (COMMON GATEWAY INTERFACE)

- ✶ Programas escritos en cualquier lenguaje que reciben una serie de parámetros y generan como respuesta una página HTML dinámicamente.
- ✶ Se ejecutan en el servidor web



- ✶ HTML 2.0 ➔ Incorpora formularios que permiten enviar datos escritos por el usuario al servidor.
  - ❑ El usuario rellena los datos del formulario y pulsa el botón
  - ❑ Los datos se envían al servidor web
  - ❑ El CGI recibe los datos y comienza a ejecutarse.
  - ❑ A medida que se va ejecutando va generando dinámicamente una página HTML que se va devolviendo al cliente para que éste la visualice.
- ✶ Problema ➔ Se crea un proceso CGI para cada petición
  - ❑ Gran sobrecarga ➔ muchos procesos e inicialización para cada uno de ellos.
  - ❑ No se mantiene la información de estado

◆ SERVLETS

- ✶ Alternativa Java a los CGIs

- Funcionamiento similar a los CGIs
  - Generan dinámicamente una página HTML
- Ventajas:
  - Utilizan toda la potencia de la plataforma Java (librerías)
  - No crean un proceso para cada petición sino que se comparte un mismo proceso entre todos los clientes (multithreading)
    - No hay sobrecarga ➔ Sólo 1 proceso y 1 inicialización
    - Mantienen el estado
- Ejemplo:

```
public class ServletBienvenida extends HttpServlet
{

   // Procesar la solicitud POST de HTTP
   public void doPost(HttpServletRequest request, HttpServletResponse
                        response) throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = new PrintWriter (response.getOutputStream());
      out.println("<html>");
      out.println("<head><title>ServletBienvenida</title></head>");
      out.println("<body>");

      // Recoger el valor del campo nombre
      String nombre =  request.getParameter("Nombre");

      // Mostrar el mensaje de bienvenida
      out.println("Bienvenido a nuestra web, “ + nombre + “<BR>”);
      out.println("</body></html>");
      out.close();
   }
}
```

  - Aspecto es el de una clase Java normal
  - Sentencias print( )
- Se verá en el tema 3

◆ ASP

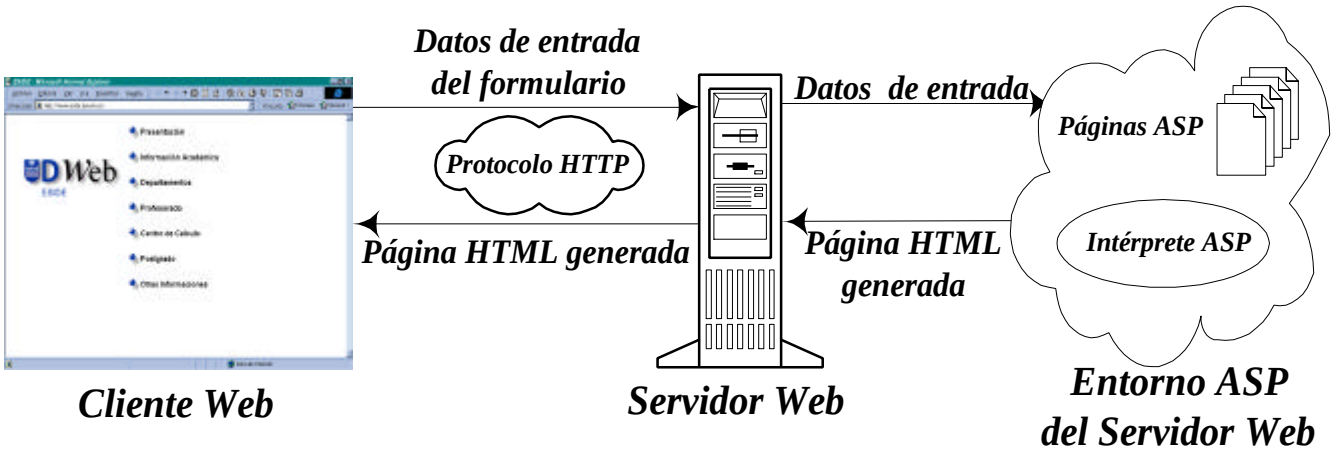
- ✶ Tecnología de Microsoft
- ✶ Filosofía ➔ La misma
  - ❑ Generan dinámicamente páginas HTML
- ✶ Característica Fundamental
  - ❑ El código HTML estático y el código ASP se encuentran juntos en la misma página ASP

Página ASP ➔ Código HTML Estático (Parte estática de la página HTML) + Código ASP (Genera el contenido dinámico de la página HTML)

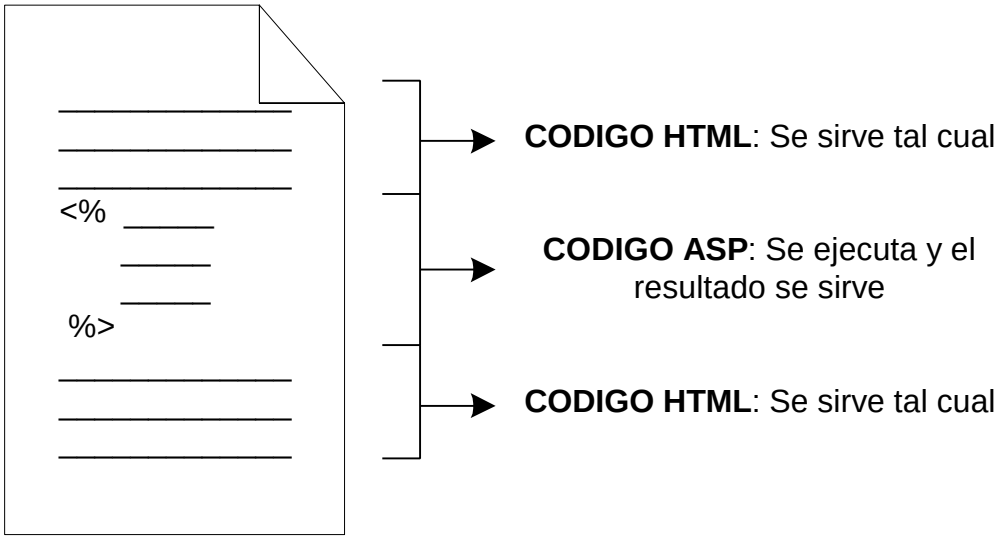
- ✶ Apariencia
  - ❑ Una página HTML con unas etiquetas especiales <%...%> entre las que se inserta el código ASP

```
<HTML>
<HEAD>
   <TITLE>Este es un ejemplo</TITLE>
</HEAD>
<BODY>
   Estos son los números del 1 al 5:
   <% For n = 1 To 5 %>
      Número <% Response.Write(n) %>
      <BR>
   <% Next n%>
</BODY>
</HTML>
```

✶ Funcionamiento



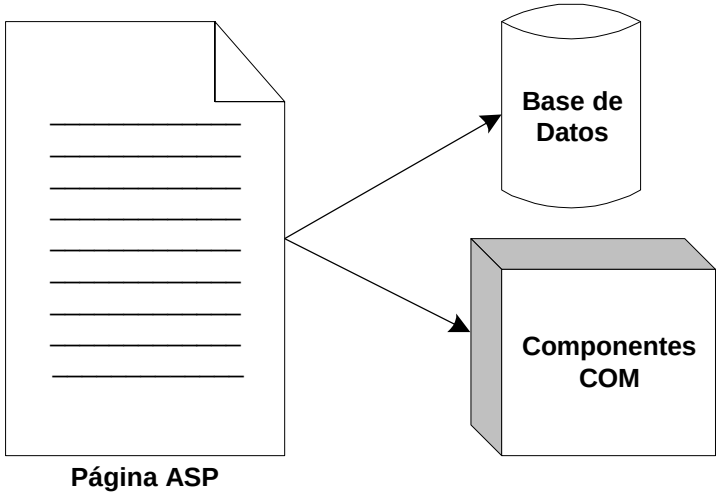
- El Interprete de ASPs del Servidor Web ejecuta el código ASP
  - Soportado por los servidores web de Microsoft



- El cliente siempre recibe código HTML y no tiene forma de saber qué parte se generó dinámicamente y cuál no.

```
<HTML>
<HEAD>
   <TITLE>Este es un ejemplo </TITLE>
</HEAD>
<BODY>
   Estos son los números del 1 al 5:
   Número 1<BR>
   Número 2<BR>
   Número 3<BR>
   Número 4<BR>
   Número 5<BR>
</BODY>
</HTML>
```

- Posibilidades



◆ JSP (JAVA SERVER PAGES)

- Versión Java de los JSP
- Funcionamiento idéntico a las ASP pero basado en Java
- <% Lenguaje Java %>
- Se integra perfectamente con las tecnologías Java: EJB, JDBC,...
- Se verá en el tema 3

◆ TECNOLOGÍAS DE CLIENTE: APPLETS Y JAVASCRIPT

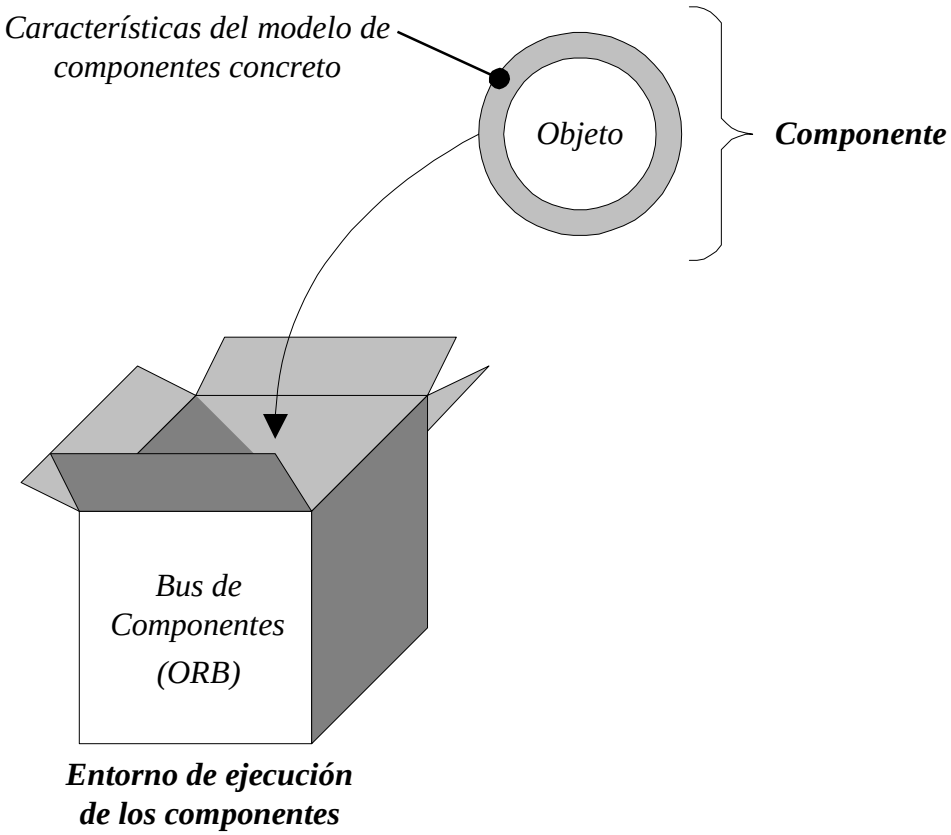
- Applets
  - Pequeños programas Java que se incrustan en páginas HTML
  - Se descargan (junto con su página HTML) y se ejecutan en el cliente dentro del propio navegador
- JavaScript
  - Lenguaje de Script que se inserta entre las etiquetas de una página HTML ➔ Como los ASPs y JSPs
  - Diferencia ➔ Se descarga íntegramente en el cliente (junto con su página HTML) y se ejecuta en él

◆ MODELOS DE COMPONENTES DISTRIBUIDOS

- ✶ No son tecnologías web a soportar la lógica de presentación (generar dinámicamente página HTML).
- ✶ Son tecnologías encaminadas a soportar los procesos de negocio de la aplicación.

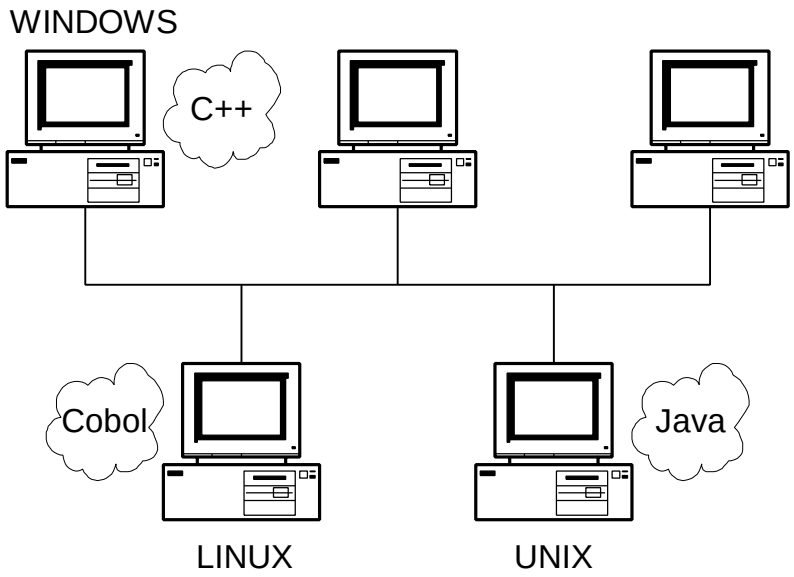
◆ ¿QUÉ SON LOS COMPONENTES?

- ✶ Objetos “normales” que cumplen una serie de características propias del modelo al que pertenecen



◆ CORBA

- ✶ Independencia del lenguaje de programación
- ✶ Independencia de la plataforma
- ✶ Independencia de la localización



- ✶ Modelo de Componente Abierto ➔ Consorcio de Empresas (OMG)

◆ COM

- ✶ Modelo de componente de Microsoft
- ✶ Sólo sobre la plataforma Microsoft (Windows)
  - ❑ No es multiplataforma
- ✶ Modelo de componentes cerrado
- ✶ DCOM ➔ versión distribuida de COM
  - ❑ Comunicación entre componente COM remotos

◆ ENTERPRISE JAVABEANS

- ✶ Modelo de componentes de la plataforma Java
- ✶ Características
  - ❑ Lenguaje Java
  - ❑ Multiplataforma
  - ❑ Modelo distribuido
  - ❑ Amplia gama de servicios ➔ Facilitan el desarrollo de app.
- ✶ Se verán en el tema 4

◆ ARQUITECTURA DEL PROYECTO