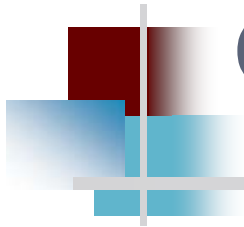
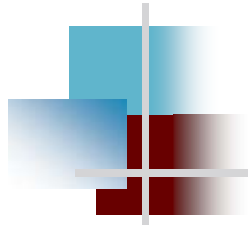


2.

## Modelos básicos de componentes



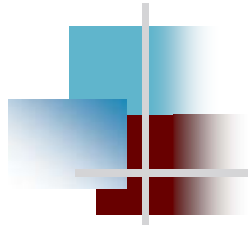
Miguel A. Laguna



# Objetivos del tema

---

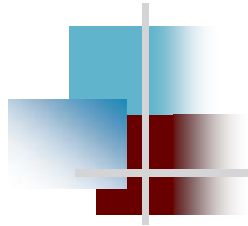
- Presentar los modelos básicos de componentes
  - Introducir las características técnicas de COM, JavaBeans..
- Plantear el problema de la heterogeneidad de lenguajes y sistemas
  - [Repasar los conceptos de sistemas distribuidos y middleware]
  - Introducir el concepto de computación en la nube



# Desarrollo del tema

---

- Modelos básicos de componentes:
  - COM y .NET
  - JavaBeans
  - Android
- El problema de la heterogeneidad
  - Sistemas distribuidos y middleware
  - Cloud Computing y SaaS

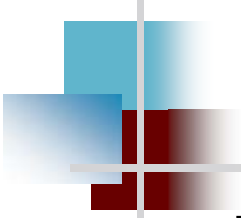


# Bibliografía

---

## Ejemplos de modelos de componentes

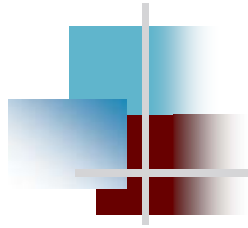
- 📖 Andy Ju An Wang, Kai Qian. Component-oriented programming. John Wiley & Sons, 2005
- 📖 Microsoft MSDN
- 📖 Oracle/Sun Tutoriales en línea
  
- 📖 Sommerville, I. "Ingeniería del software" Pearson, 2005 (7ª ed.)



# ¿Como se puede reutilizar código binario?

---

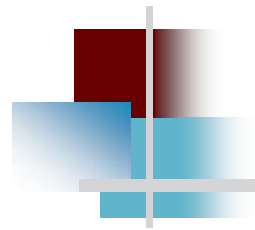
- En OO, dependiendo del lenguaje (C++, Java)
  - Encapsulamiento y herencia facilita la reutilización
  - El problema es que los objetos NO pueden ser reutilizados en otros lenguajes
  - Incluso existen incompatibilidades entre objetos creados por el mismo lenguaje (clases escritas en el mismo lenguaje)
    - Incompatibilidad Binaria (y a veces de código fuente!)
- Solución a los problemas de reutilización binaria:
  - Modelos de Componentes



# Modelos de Componentes

---

- Un Modelo de Componentes es una especificación en la que se define entre otros:
  - Estructura del componente
  - Comunicación entre componentes
  - Manipulación por los entornos de desarrollo
- Un Modelo de Componentes es una **especificación** que define componentes binarios.
  - NO es una biblioteca como tal

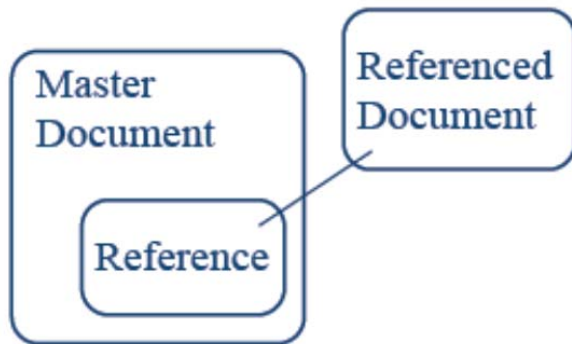


## 2.1. COM/COM+

---

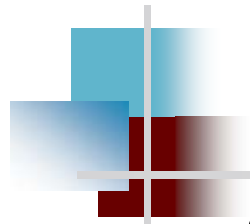
# Origen: Object\* linking and embedding

- OLE fue el precursor de COM
  - OLE (1991) para creación de documentos compuestos
  - Con OLE 2.0 (1993) aparecen controles OLE u OCX, paletas de componentes en formularios de Windows



\*NOTA: Microsoft llamaba Objetos a los Componentes





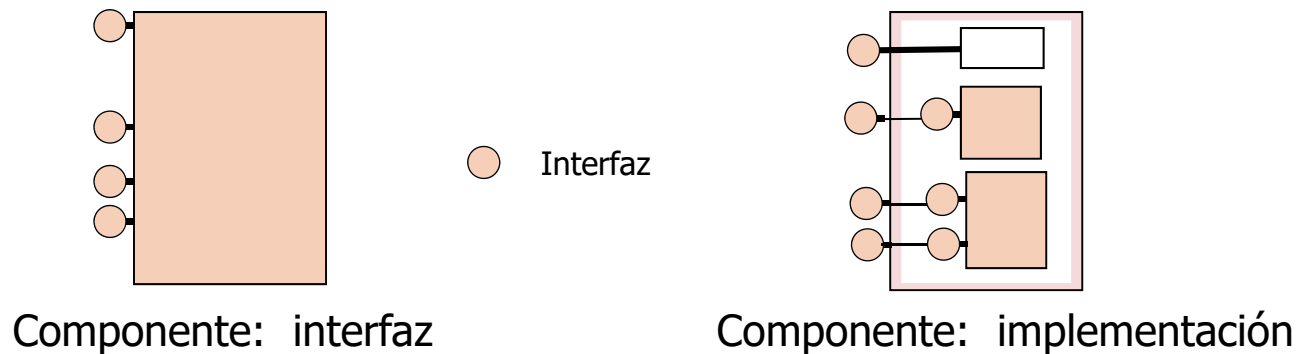
# Origen:

---

- OLE 2.0 dio origen a COM y ActiveX
  - No solo añadir controles (visuales o no) a un formulario sino también a una página Web (ActiveX)
- Después DCOM y COM+
- **Active Template Library:** biblioteca para la creación de componentes COM con C++ que facilita su desarrollo y registro.
  - Luego se pueden utilizar por ejemplo con VB
  - (Como alternativa, se pueden crear componentes sencillos directamente desde VB)

# Estructura de COM

- Concepto Fundamental de COM:
  - Separación entre la Interfaz Pública de un Componente y su Implementación

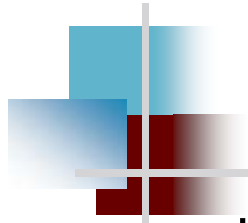




# Estructura de COM

---

- Interfaz Pública
  - Interfaces COM, tablas de métodos o clases
- Un componente COM implementa una o más interfaces.
  - Tablas de punteros a métodos
- Terminología:
  - **Componente (objeto)** COM: Análogo a una clase/objeto C++
  - **Interfaz** COM: conjunto de servicios expuestos por un componente COM
  - **Servidor**: los componentes COM se alojan en una DLL o EXE
  - **Cliente**: aplicación que hace uso de los servicios de un componente COM



# ¿Cómo funciona COM?

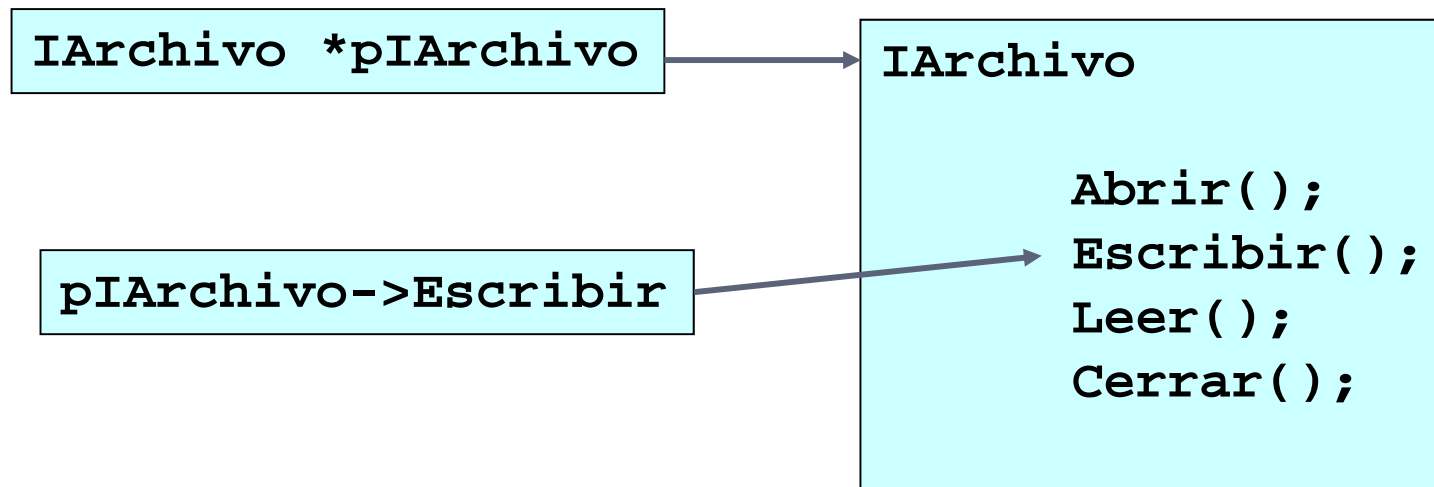
---

- Una INTERFAZ (internamente) es una tabla de PUNTEROS a métodos implementados por un componente
- Al crear un componente COM hay que decidir qué interfaces implementará
  - Implementar una Interfaz significa codificar TODOS los métodos indicados en ella
- Para utilizar un cierto componente, solo es preciso recuperar un puntero a alguna de sus interfaces



# Estructura de una Interfaz

- Una interfaz es una tabla con punteros a funciones.
  - Equivalente a una clase abstracta





# Estructura de una Interfaz

---

```
Class IArchivo
```

```
{
```

```
public:
```

```
    virtual __stdcall HRESULT Abrir(...) = 0;
```

```
    virtual __stdcall HRESULT Escribir(...) = 0;
```

```
    virtual __stdcall HRESULT Leer(...) = 0;
```

```
    virtual __stdcall HRESULT Cerrar()= 0;
```

```
};
```

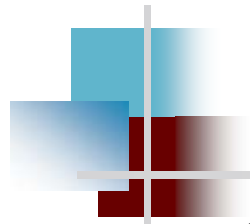
- La clase IArchivo es abstracta, no se puede instanciar, no hay constructor o destructor



# La Interfaz IUnknown

---

- Un objeto COM tiene que implementar siempre la interfaz `IUnknown`
- Objetivos:
  - Ciclo de vida
  - Resolución de interfaces
- Ciclo de vida
  - `AddRef( )`
  - `Release( )`
- Resolución de interfaces
  - `QueryInterface( )`



# Componente COM

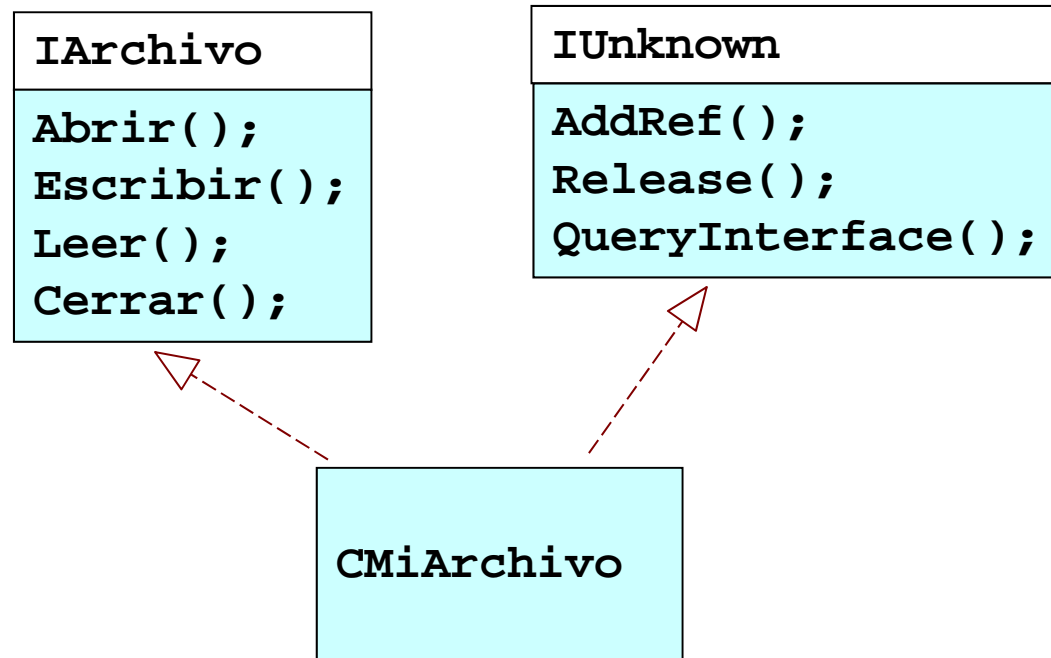
---

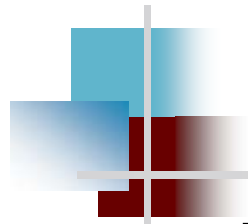
- Se utiliza el mecanismo de herencia para obtener la especificación de la interfaz
- Se crea una nueva clase que descienda de todas las interfaces COM
- Resolución de interfaces:
  - Se utiliza la interfaz **IUnknown**
  - Un objeto COM implementará todos los métodos de todas las interfaces definidas + los métodos de la interfaz **IUnknown**
  - Un cliente NUNCA tiene acceso directo al componente
  - El acceso es a través de un puntero a una de sus interfaces o a través de **IUnknown**





# Estructura de un componente COM

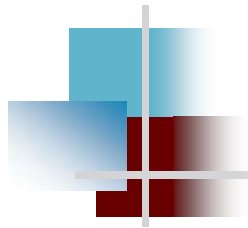




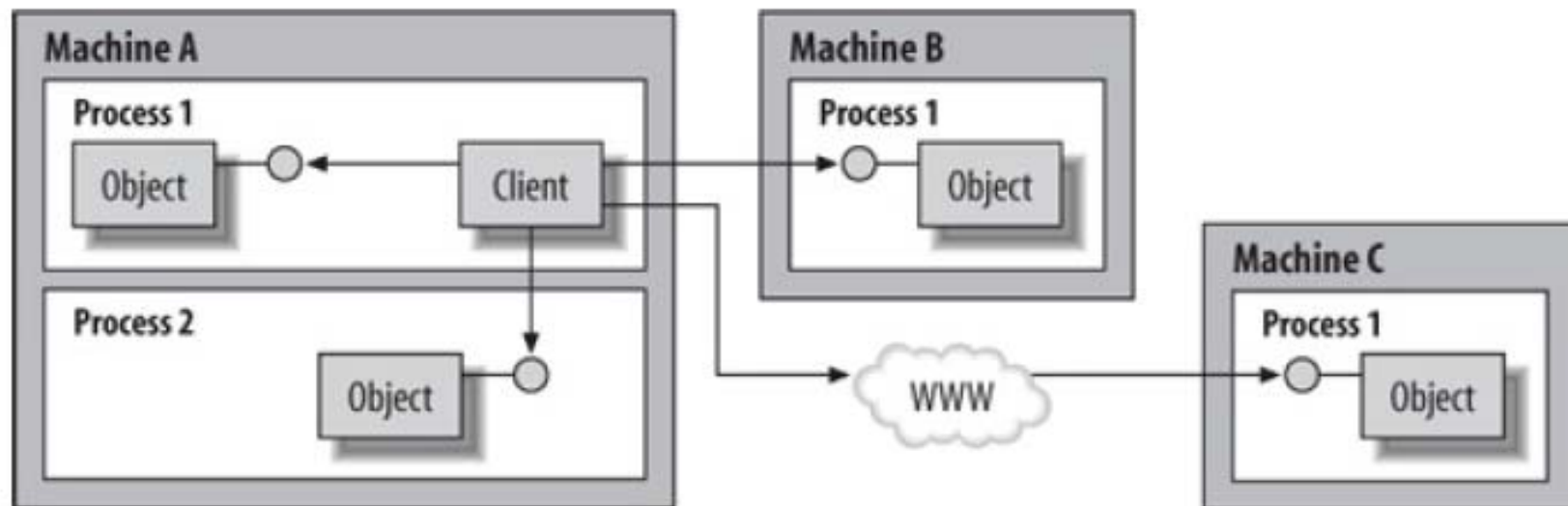
# Servidor COM

---

- Un componente COM no existe de forma aislada
- Se integra en **Servidores** (o Contenedores) COM
- Hay dos clases de servidores:
  - DLL: Se ejecuta en el mismo espacio de direcciones del cliente
  - EXE (Ejecutables): Se ejecuta otro programa con el cual se debe comunicar el cliente, ya sea mediante IPC (local) o RPC (remoto)
- Los servidores además de alojar y ejecutar los componentes realizan otras funciones como **Registrar el componente**



# Servidores COM



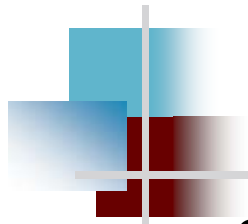


# Soporte: MIDL

---

- LENGUAJE IDL – Interface Definition Language
  - El IDL es el lenguaje descriptivo que hace a COM independiente del lenguaje
  - Se utiliza MIDL (Microsoft IDL), integrado con Visual C++

```
Interface IArchivo: IUnknown {  
    HRESULT Abrir([in] LPOLESTR nombre);  
    HRESULT Escribir([in] LPOLESTR datos,  
                    [in] int longitud);  
    HRESULT Leer([in] LPOLESTR datos, [in] int longitud,  
               [out, retval] int* bytesleidos);  
    HRESULT Cerrar();  
};
```

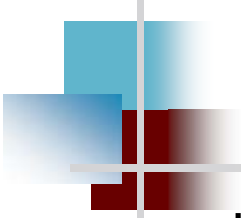


# Soporte: MIDL

---

- Solo se soporta la herencia simple
- Todo método retorna HRESULT,
  - indica condición o no de error
- Que pasa si un método retorna algún valor?
  - Atributo retval
  - [out, retval] tipo\* ...
- El pre-compilador MIDL toma un archivo \*.idl y genera una **biblioteca de Tipos**, Archivos de cabecera .h para C/C++, modulo de código que contendrá identificadores de interfaces y clases
- La biblioteca de tipos es la que se utiliza en otros lenguajes diferentes a C/C++ (Visual Basic)

# Identificador de componentes e interfaces

- 
- Los nombres de clases e interfaces son elementos con ámbito reducido
    - Válidos solo en una cierta aplicación o entorno de desarrollo.
  - Se utilizan identificadores conocidos como GUID
    - Global Unique Identifier
    - Teóricamente son irrepetibles en el tiempo y espacio.
    - 128 bits de longitud
  - CLSID (Class Identifier), GUID que identifican componentes
  - IID (Interface Identifier), GUID que identifican interfaces



# Registro del componente

---

- Puesta en marcha para un componente localizado en una DLL:  
C:\Windows\system32>regsvr32 C:\comp\pDLL.dll
- Funciones básicas:
  - CoInitialize()
  - CoCreateInstance()
- Localización del servidor:
  - Se invoca CoInitialize()
  - Y luego CoCreateInstance() facilitando el CLSID
  - La asociación del CLSID y el servidor está en el [Registro de Windows](#)
  - Si el servidor es una DLL utiliza la clave: [InprocServer32](#)
  - Otros casos son: LocalServer32 o RemoteServer32

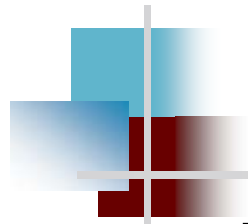


# COM y “DLL hell”

---

- La ubicación de los componentes se almacena en el registro de Windows
  - Solo puede haber una versión instalada de cada componente
- El infierno de las DLLs: dos o más aplicaciones requieren diferentes versiones de un mismo componente
- Windows XP introdujo un nuevo modo de registro de componentes COM
  - las aplicaciones que necesiten componentes COM almacenan toda la información de registro en un directorio propio, en lugar de en el registro global,
  - no se podía usar para EXE, COM o servidores como MDAC, MSXML, DirectX o Internet Explorer





# Mejoras en COM+

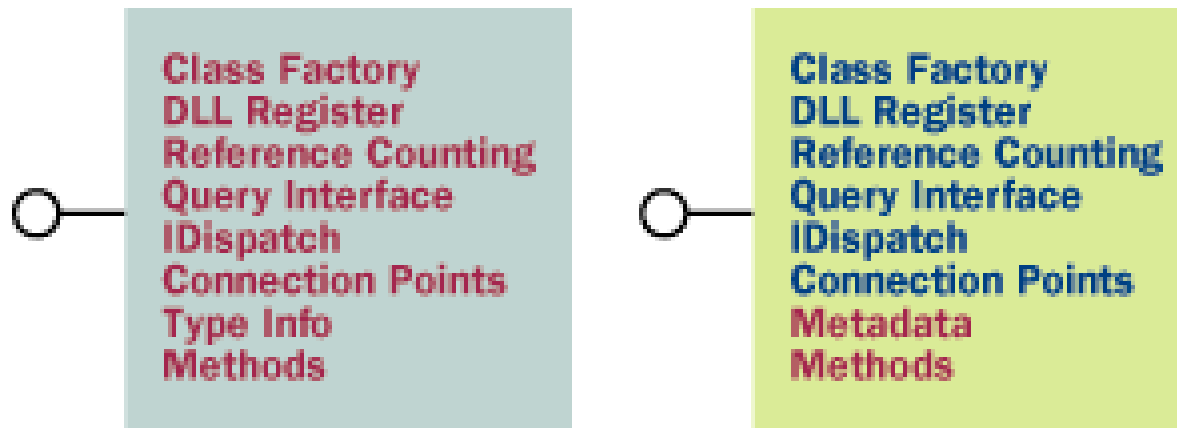
---

- Un entorno de ejecución (RunTime) y un conjunto de servicios
  - COM: Limitada a funciones que ofrece cada entorno
  - COM+: el sistema operativo proporciona el entorno de ejecución
- Diseñado para simplificar la creación y uso de componentes
- Permite una mejor interoperabilidad entre los componentes



# Mejoras en COM+

- El desarrollador COM debe preocuparse por cuestiones ajenas a la funcionalidad
  - El cliente debe utilizar el contador de referencias
  - El cliente debe utilizar el código para crear el objeto e inicializar el componente
  - Gran parte de este código es idéntico en todos los componentes. Implementación por defecto.





## 2.2. NET.

---



# Elementos de .NET

VB

C++

C#

J#

...

Common Language Specification

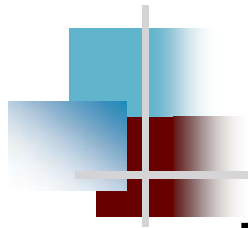
ASP.NET: Servicios Web  
y Web Forms

Windows  
Forms

ADO.NET: Datos y XML

Biblioteca de Clases Base

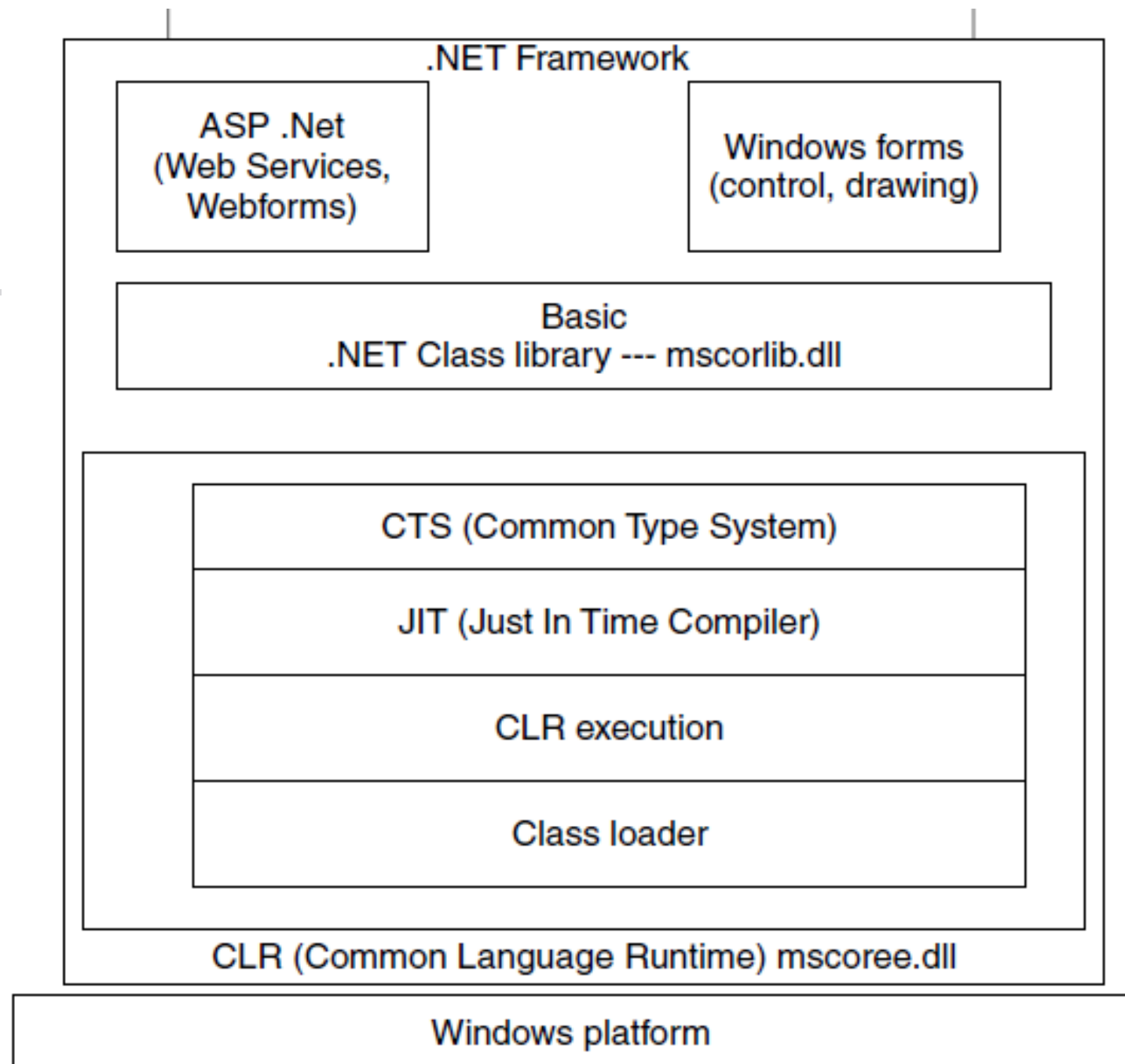
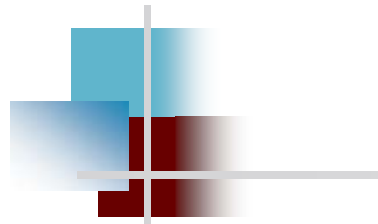
Common Language Runtime

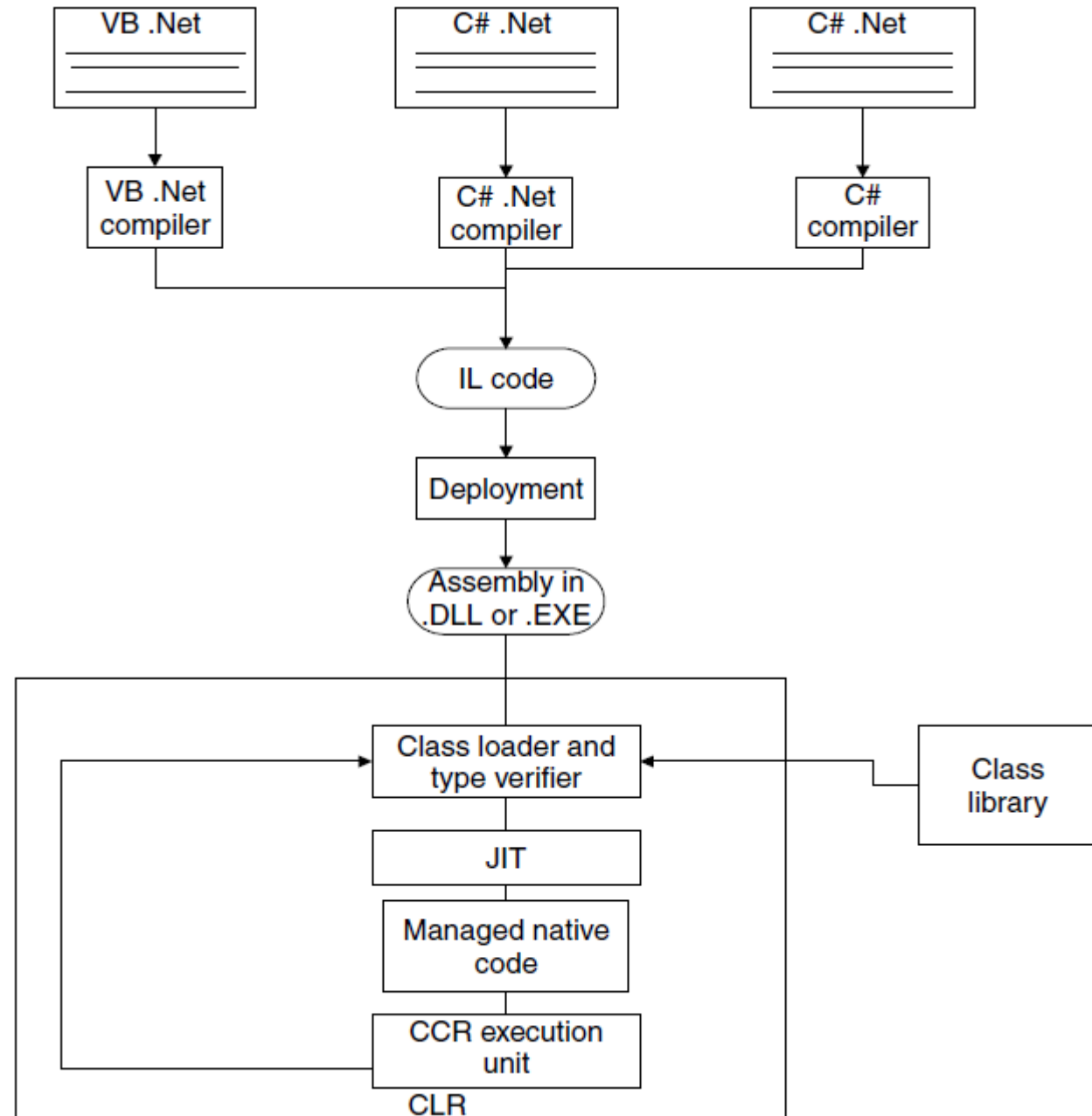
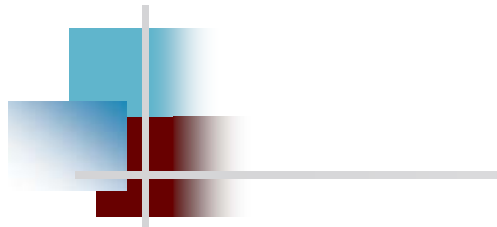


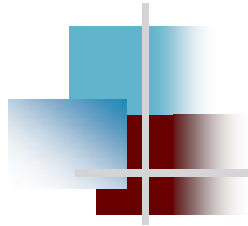
# Framework .NET

---

- Es un conjunto de:
  - Una máquina virtual (CLR) multilenguaje
  - Herramienta de desarrollo rápido (RAD) de aplicaciones de escritorio y Web
  - Conjunto de bibliotecas y frameworks para acceso a BD/XML, interfaces de usuario, etc.
  - Plataforma de soporte componentes y servicios Web
    - Soporta el despliegue de múltiples versiones del mismo nombre componente sin conflicto alguno
    - Simplifica la creación y despliegue de componentes,
    - Servicios fiables y escalables para componentes.
    - .NET y COM pueden coexistir y colaborar





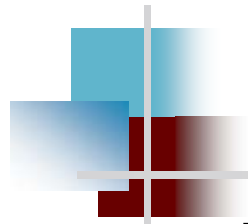


# Modelo de componentes .NET

---

- .NET en sí mismo está construido con componentes
  - Cada espacio de nombres como `System.XML` está incorporado en un componente (`System.XML.dll`)
  - Las .DLL son **ensamblados**, las unidades de despliegue de .NET
  - Bloques de creación de las aplicaciones .NET Framework
  - La interoperabilidad entre componentes es más fácil que con COM





# .NET frente a COM

---

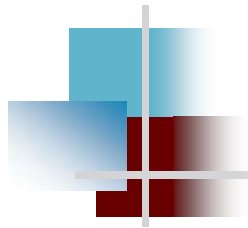
- No existe ninguna interfaz base Iunknown
  - todos los componentes se derivan de la clase System.Object.
- No hay necesidad de IDL
  - definiciones en el código fuente.
  - El compilador incrusta las definiciones de tipos en los metadatos del ensamblado
- Las dependencias se guardan durante la compilación en el manifiesto del ensamblado
- No hay recuento de referencias
  - .NET incorpora un mecanismo de recolección de basura



# .NET frente a COM

---

- La identificación no se basa en los identificadores únicos globales (GUID).
  - El tipo (clase o interface) es identificado por el espacio de nombres y el nombre del ensamblado.
- Cuando a un ensamblado se comparte, debe contener un nombre seguro ("strong name")
  - Es una firma digital generada mediante el uso de una clave de cifrado.
  - Garantiza la autenticidad de los componentes (.NET se niega a ejecutarlos si no hay coincidencia)



# Versiones e identificación

**Assembly Information** [?] [X]

Title: MyAssembly

Description:

Company: IDesign Inc.

Product: MyProduct

Copyright: Copyright © IDesign Inc. 2005

Trademark:

Asssembly Version: 1 2 3 4

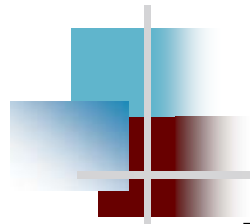
File Version: 1 0 0 0

GUID: bbe29006-fb90-4258-9852-225ccd5048cf

Neutral Language: (None) ▼

☐ Make assembly COM-Visible

OK Cancel

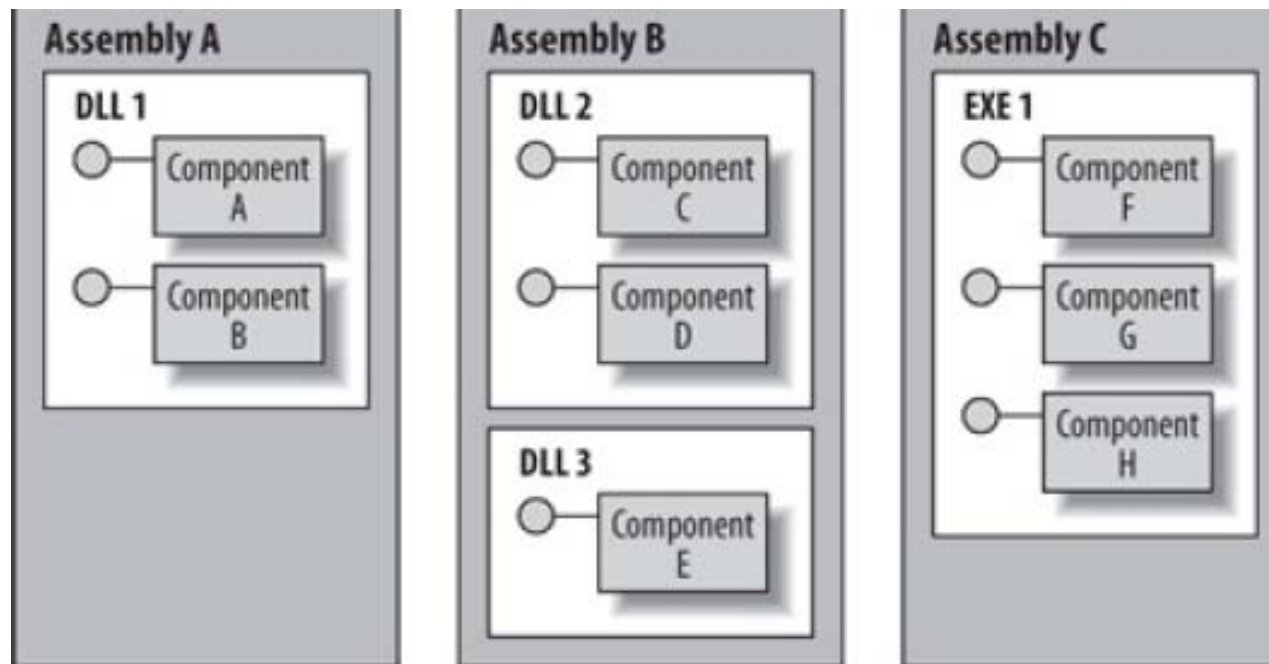


# Ensamblados (Assembly)

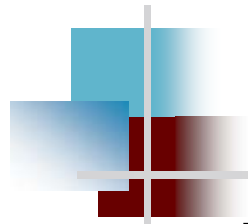
---

- Es una colección de tipos y recursos creados para funcionar en conjunto y formar una unidad lógica de funcionalidad
- Unidad fundamental de implementación, control de versiones, reutilización, ámbitos de activación y permisos de seguridad.
- Los ensamblados proporcionan al CLR la información necesaria para conocer las implementaciones de cada tipo
- En ejecución, un tipo no existe fuera del contexto de un ensamblado

# Modulos y ensamblados



- Un módulo o varios módulos
- DLL o EXE
- Si hace falta un componente, se añade una referencia



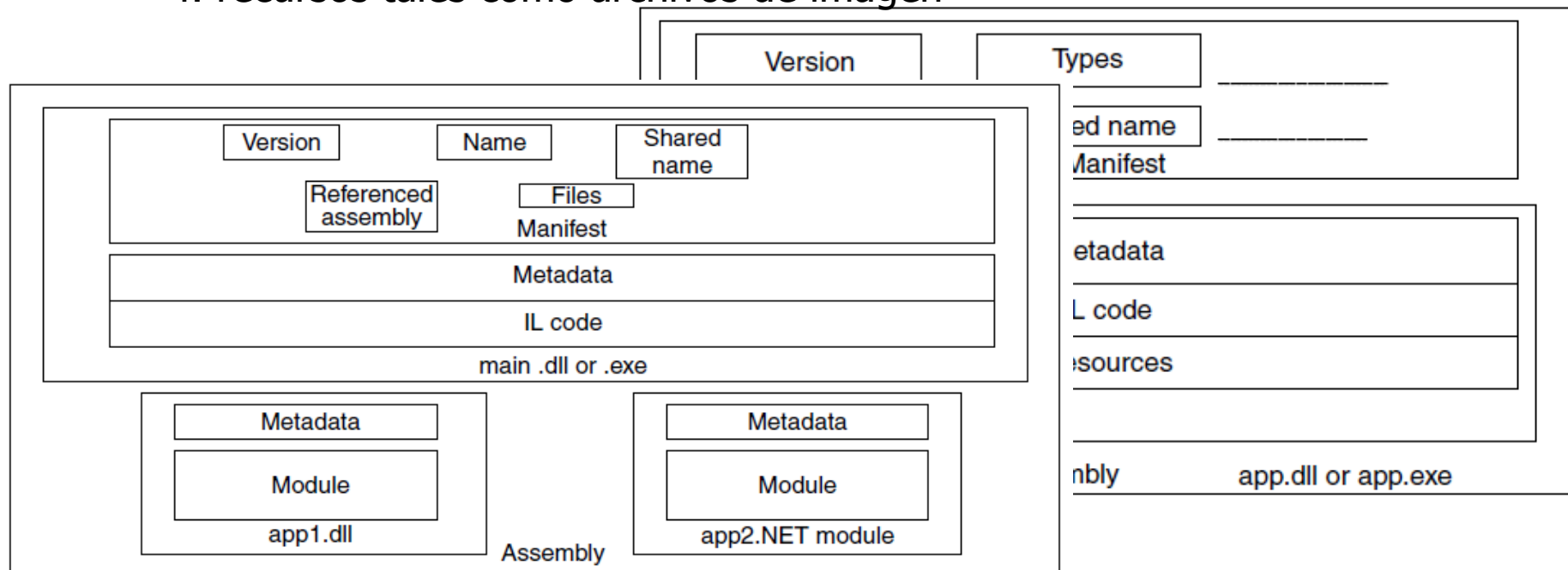
# Ensamblados y metadatos

---

- Resuelven problemas de versiones e implementación que surgen con los sistemas basados en componentes COM
- Había que mantener la coherencia de todas las entradas del Registro necesarias para activar una clase COM.
- Los ensamblados son componentes auto-descriptivos que no dependen de las entradas del Registro
- El compilador añade información adicional en todos los ensamblados (metadatos)

# Ensamblado: partes

- 1. Manifiesto (tabla de registros de información):
  - nombre del ensamblado, versión, nombre seguro, cultura, archivos que componen el ensamblado, dependencias de ensamblados...
- 2. metadatos de los módulos
- 3. código IL de los módulos
- 4. recursos tales como archivos de imagen





# Cualquier clase es un Componente

---

```
namespace TempConv
public class TempConvComp {
    public double cToF (double c)
        { return (int) ((c*9)/5.0+32);}
    public double fToC (double f)
        { return (int) ((f-32)*5/9.0);}
}
```

```
>csc /t:module TempConvComp.cs TempConvComp.netmodule
```

```
>csc /t:library /addmodule: TempConvComp.netmodule
    anotherComp.dll
```

**O directamente:**

```
>csc /t:library TempConvComp.cs TempConvComp.dll
```





# Para usar el componente DLL

---

- En C#

```
using TempConv;
```

```
...
```

```
TempConvComp myTempConvComp;
```

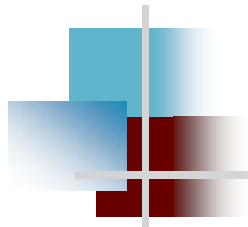
```
myTempConvComp = new TempConvComp();
```

- En C++

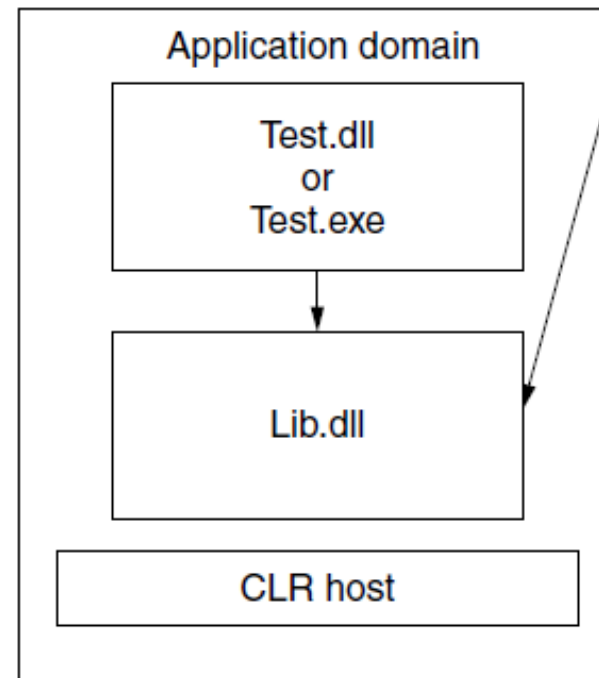
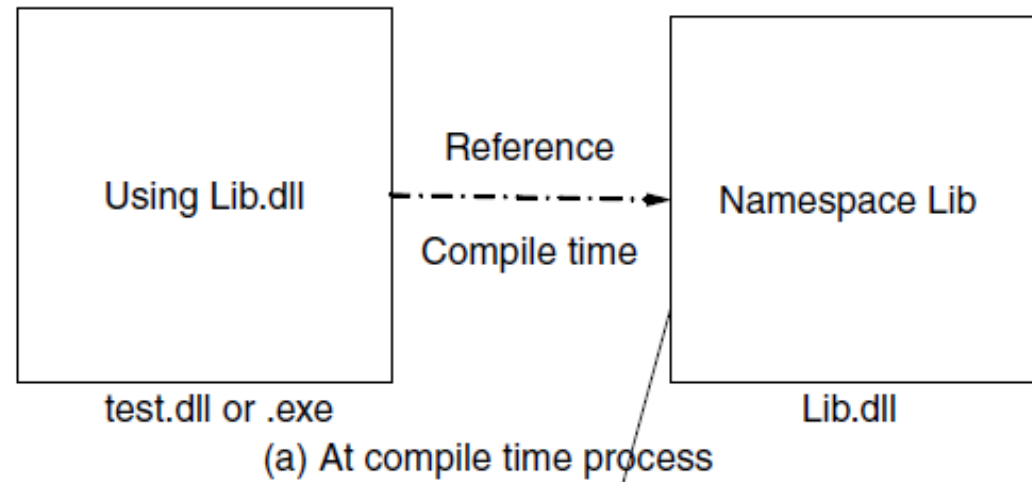
```
#using "TempConv.dll"
```

```
...
```

```
TempConv::TempConvComp *myCSConvComp =  
new TempConv::TempConvComp ();
```

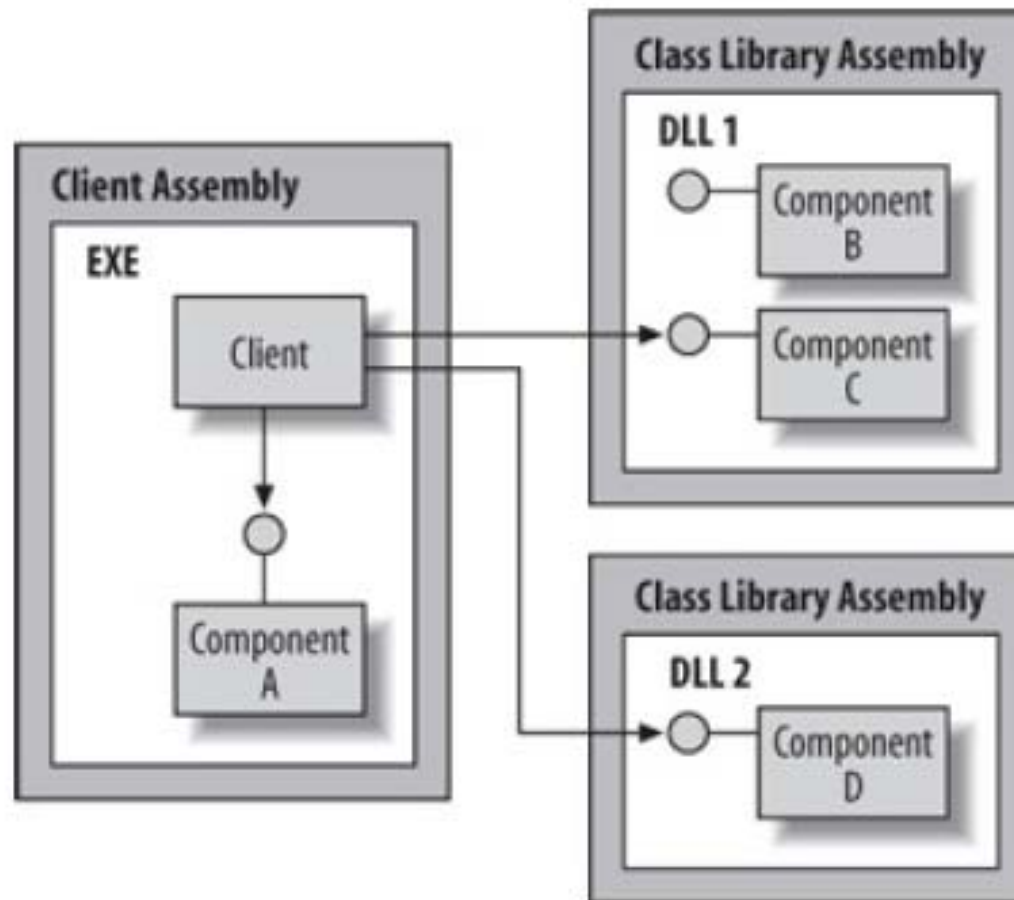


# Enlace:



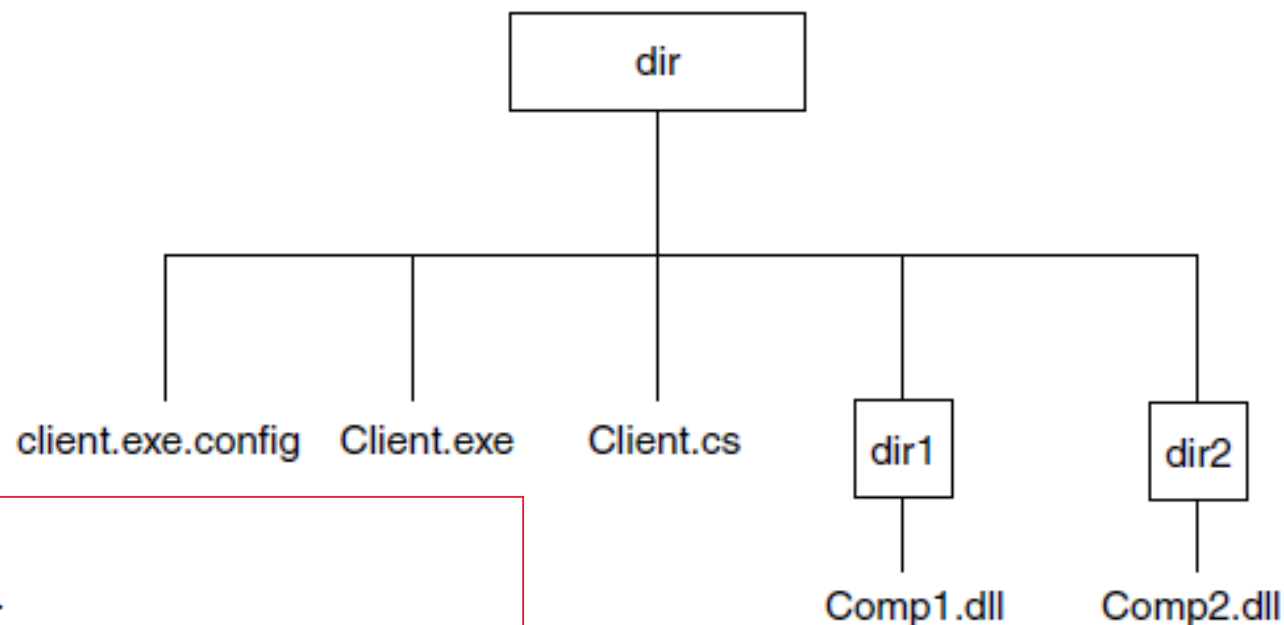
# Acceso a componentes

- Uso directo y uso mediante referencias...



# Despliegue privado

```
>csc /t:library /out:dir1\comp1.dll comp1.cs  
>csc /t:library /out:dir2\comp2.dll comp2.cs  
>csc /r:dir1\comp1.dll, dir2\comp2.dll /out:client.exe  
client.cs
```



```
<configuration>  
  <runtime>  
    <assemblyBinding>  
      <probing privatePath = "dir1; dir2"/>  
    </assemblyBinding>  
  </runtime>  
</configuration>
```



# Despliegue público

- Es necesario crear un par de claves (pública/privada)
- En el código referenciamos el archivo de claves:

```
using System.Reflection:  
[assembly:AssemblyKeyFile ("mykey.snk")]  
[assembly:AssemblyDelaySign (false)]  
[assembly:AssemblyVersion ("1,0,0,1")]
```

- Automáticamente, al compilarlo, código **firmado!**

```
.assembly extern MyServerAssembly  
{.publickeytoken = (22 90 49 07 87 53 81 9B )  
.ver 1:2:3:4}
```

- Registro en el Global Assembly Cache (GAC)

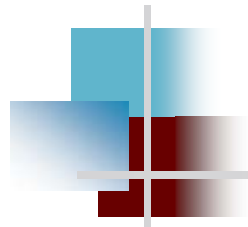
```
>gacutil /i mycomponent.dll
```

# Despliegue público

- Finalmente para usar el componente:

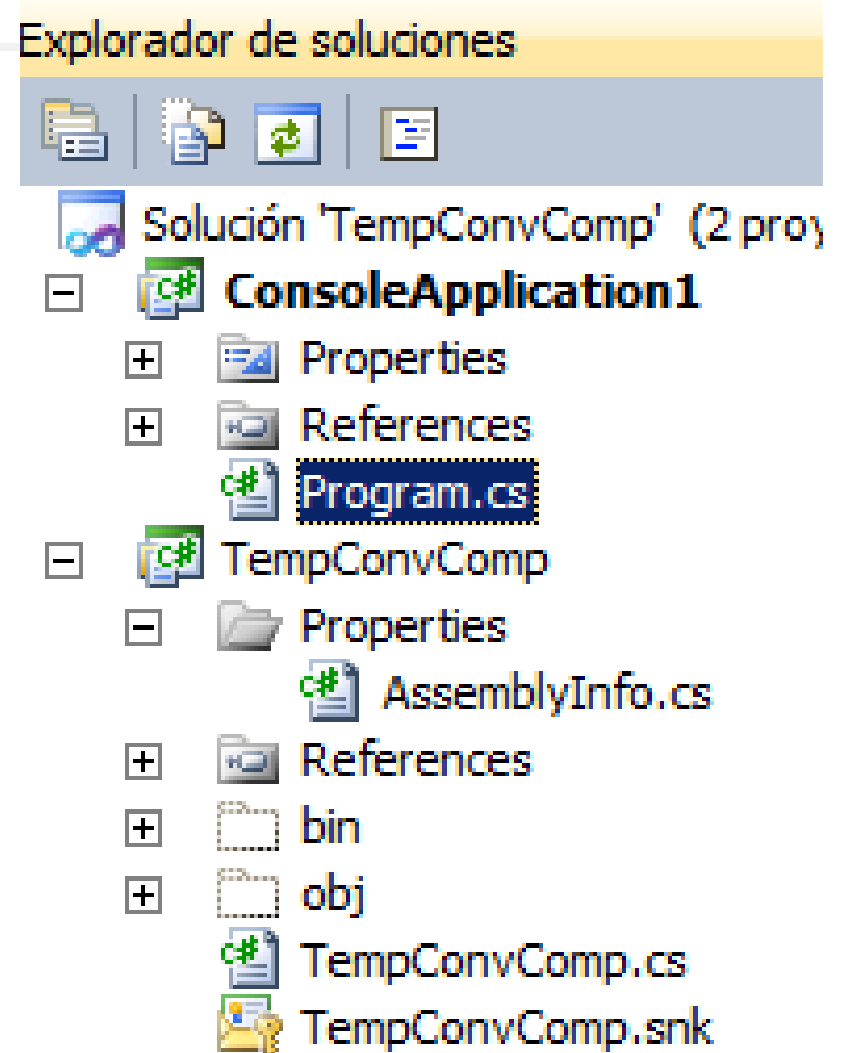
```
>csc /t:exe /r:mycomponent.dll /out:myapp.exe  
myapp.cs
```



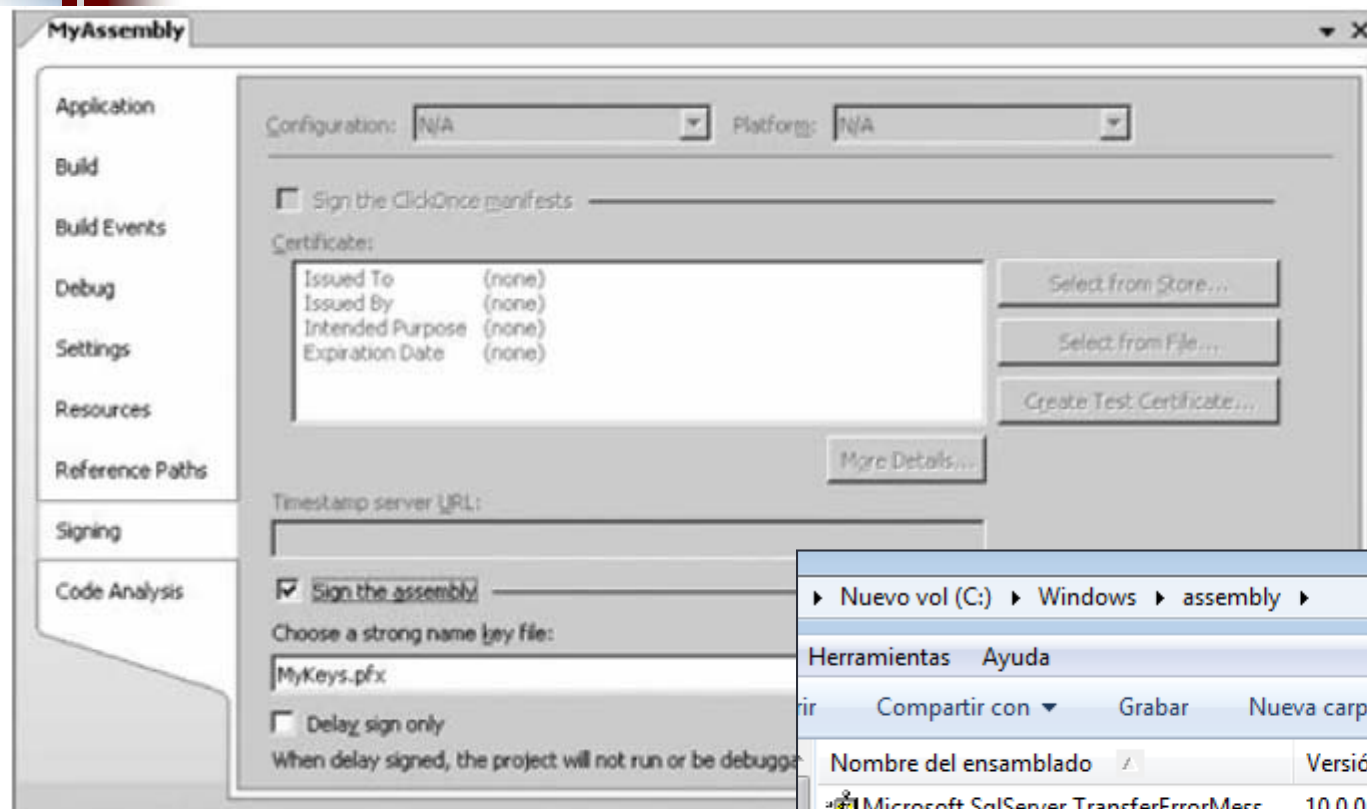


# En la práctica... Visual Studio

- Ref al ensamblado (DLL) ----->
- Aplicación de consola ----->
  
- Ensamblado (DLL) ----->
  
- Componente (fuente) ----->
- Archivo de claves generado --->



# Firmar, instalar en GAC



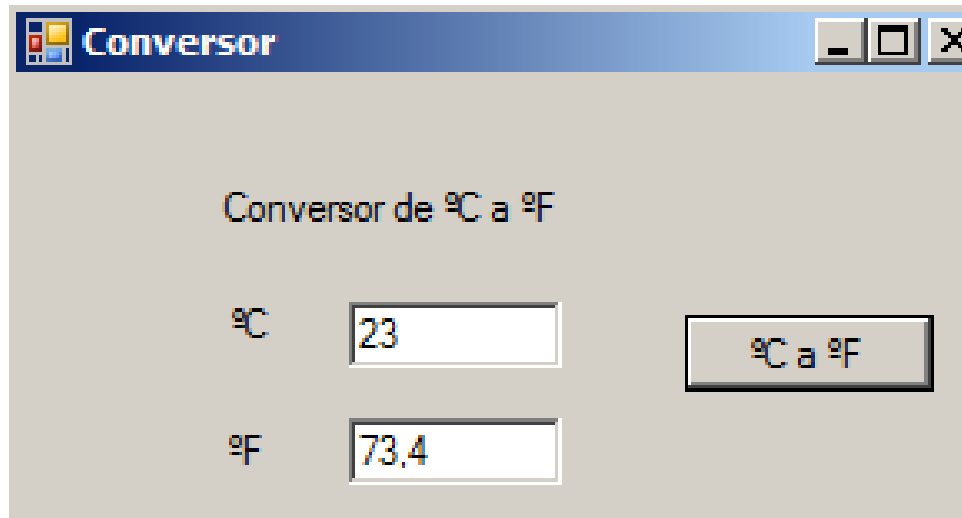
Nuevo vol (C:) > Windows > assembly >				
Herramientas Ayuda				
Ver Compartir con Grabar Nueva carpeta				
Nombre del ensamblado	Versión	Ref...	Símbolo de clave pública	
Microsoft.SqlServer.TransferErrorMess...	10.0.0.0	es	89845dcd8080cc91	
Microsoft.SqlServer.TransferJobsTask	10.0.0.0		89845dcd8080cc91	
Microsoft.SqlServer.TransferJobsTask.r...	10.0.0.0	es	89845dcd8080cc91	
Microsoft.SqlServer.TransferLoginsTask	10.0.0.0		89845dcd8080cc91	
Microsoft.SqlServer.TransferLoginsTas...	10.0.0.0	es	89845dcd8080cc91	
Microsoft.SqlServer.TransferObjectsTask	10.0.0.0		89845dcd8080cc91	



# En otra aplicación cliente...

- Solo hay que añadir una referencia al ensamblado DLL

```
private void C2F_Click(object sender, EventArgs e){  
    TempConvComp myTempConvComp = new TempConvComp();  
    double input;  
    input = Double.Parse(gC.Text);  
    gF.Text = myTempConvComp.cToF(input).ToString();  
}
```



# Componentes y controles

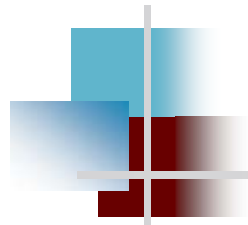
Examinador de objetos X Form1.Designer.cs Form1.cs Form1.cs [Diseño]

Examinar: Mi solución ...

<Búsqueda>

- [-] {} TempConv
  - [-] TempConvComp
    - [-] Tipos base
      - Object
- [-] WindowsFormsApplication 1
  - [-] {} WindowsFormsApplication 1
    - [-] Form1
      - [-] Tipos base
        - [-] Form
          - [-] ContainerControl
            - [-] IContainerControl
          - [-] ScrollableControl
            - [-] Control
              - [-] Component

~Component()  
Component()  
Dispose(bool)  
Dispose()  
GetService(System.Type)  
ToString()  
CanRaiseEvents  
Container  
DesignMode  
Events  
Site  
Disposed

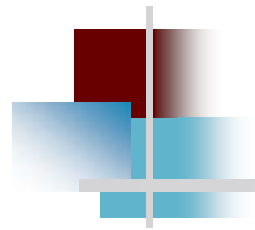


# Componentes .NET

---

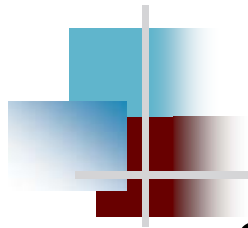
- .NET Framework proporciona la interfaz **IComponent** y la clase base **Component** para facilitar la creación de componentes (visuales)
- Las clases descendientes **UserControl** y **Control** facilitan la creación de estos componentes visuales)
- Cualquier clase que implemente la interfaz **IComponent** es un componente. En C#:

```
using System.ComponentModel  
public class MiComponente : Component { }
```



## 2.3. JavaBeans

---



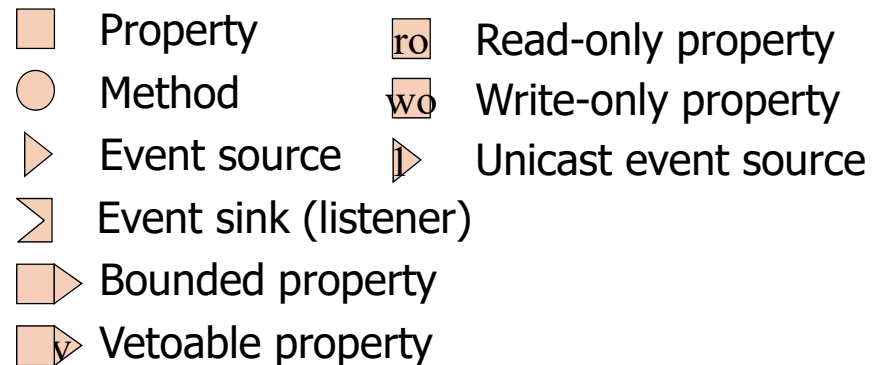
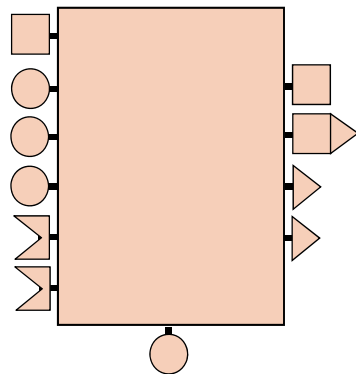
# JavaBeans

---

- Son componentes software Java reutilizables que pueden ser manipulados por herramientas de desarrollo.
- Java necesitaba simplificar la creación de las interfaces de usuario.
  - Su objetivo era la facilidad de uso del entorno VB y orientarse a tecnología de componentes.
- Diseñados para interaccionar en dos contextos:
  - En el momento de la composición, dentro de la herramienta de creación.
  - En tiempo de ejecución, en el entorno de ejecución

# Interfaz de un JavaBean

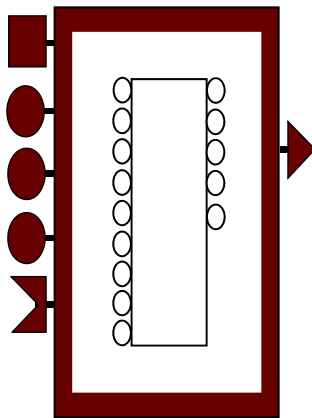
- Cuatro tipos de puertos:
  - Métodos
  - Propiedades
  - Eventos (lanzados)
  - Eventos (recibidos: listeners)



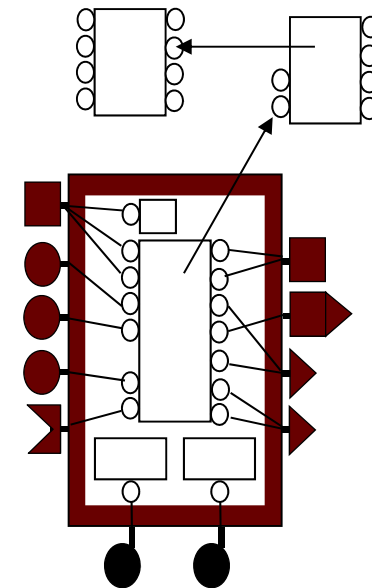
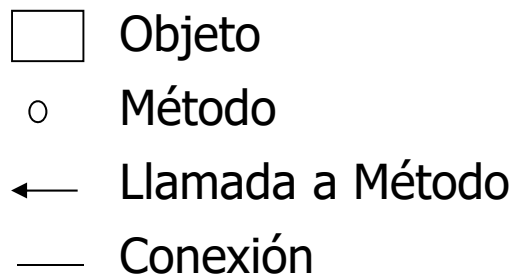
Puertos

# Implementación de un JavaBean

- La mayoría de los componentes JavaBean son implementados con una clase Java
- pero son posibles implementaciones más sofisticadas.
  - Envolver un sistema legado.
  - Varias clases

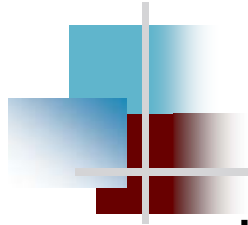


Implementación simple



Implementación compleja

# Características de un JavaBean



- Los JavaBeans son clases que:
  - Soportan introspección
  - Personalización: Tienen propiedades (se puede cambiar la apariencia y comportamiento del JavaBean, incluso desde un programa)
  - Pueden comunicarse mediante eventos
  - Tienen persistencia (se puede guardar y recuperar el estado del JavaBean)
- Y en general, promueven la Reusabilidad y la Portabilidad

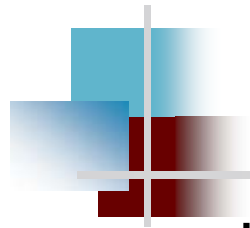




# Un JavaBean muy simple

---

```
public class MyBean implements java.io.Serializable{
    protected int theValue;
    public MyBean () { }
    public void setMyValue(int newValue ){
        theValue = newValue;
    }
    public int getMyValue(){
        return theValue;
    }
}
```



# Introspección

---

- Las herramientas de desarrollo descubren las características de un JavaBean y pueden modificar sus propiedades mediante introspección
  - La clase `Introspector` proporciona un conjunto de métodos para obtener información de métodos y eventos usando `java.lang.reflect`
- Convenciones específicas de nombres
  - Los modificadores deben de empezar por set.  

```
public void setColor(Color _color){}
```
  - Las funciones de consulta deben de empezar por get  

```
public Color getColor(){}
```



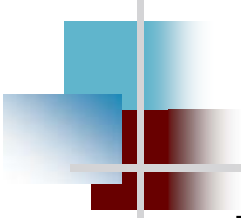
# Personalización

---

- Las características de apariencia y comportamiento de un JavaBean pueden ser modificadas durante el diseño.
  - Utilizando editores de propiedades
  - Opcionalmente, una clase auxiliar **BeanInfo** puede añadir información, un icono, etc.

```
public class MyBeanBeanInfo extends SimpleBeanInfo {  
    public Image getIcon(int iconKind) {  
        Image img = loadImage("wja.gif");  
        return img;  
    }  
}
```

# Comunicación mediante eventos

- 
- Los eventos son un mecanismo de notificaciones entre un componente fuente (source) y componentes receptores (observadores o listeners).
  - Las herramientas de desarrollo pueden examinar un JavaBean para determinar qué eventos puede disparar (enviar) y cuáles puede manejar (recibir).



# Notificación de cambios

---

- Las propiedades limitadas requieren que se notifique a ciertos objetos (observador o listener) cuando experimentan un cambio.
- La notificación del cambio se realiza a través de un evento `PropertyChangeEvent`.
- Los objetos que deseen ser notificados del cambio en una propiedad limitada deberán registrarse como observadores



# Registro de observadores

---

- Así, el JavaBean suministrará métodos:

```
public void  
    addPropertyChangeListener(PropertyChangeListener p1)
```

```
public void  
    addPropertyNameListener(PropertyChangeListener p1)
```

- Y el observador que implementa la interfaz `PropertyChangeListener`, implementará el método `PropertyChange()`



# Persistencia

---

- Permite a los JavaBeans guardar su estado, y restaurarlo posteriormente.
- Utilizan la Serialización de Objetos Java,
  - una forma sencilla y automática de almacenar el estado interno de un JavaBean

```
MyBean bean1;
```

```
...
```

```
ObjectOutputStream salida=new  
ObjectOutputStream(new  
    FileOutputStream("bean1.ser"));  
salida.writeObject(bean1);
```



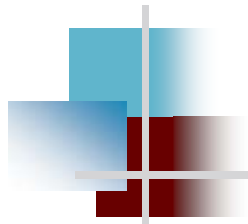
# Proceso de creación

---

- Una clase “normal”

```
public class Fecha implements Serializable {  
    private Date fecha;  
    public Fecha() {  
        fecha = new Date();  
    }  
    public Date getFecha() {  
        return fecha;  
    }  
    ...  
}
```





# Proceso de creación

---

- Crear un archivo manifiesto

```
Manifest-Version; 1.0  
Name: FechaBean.class  
Java-Bean: True
```

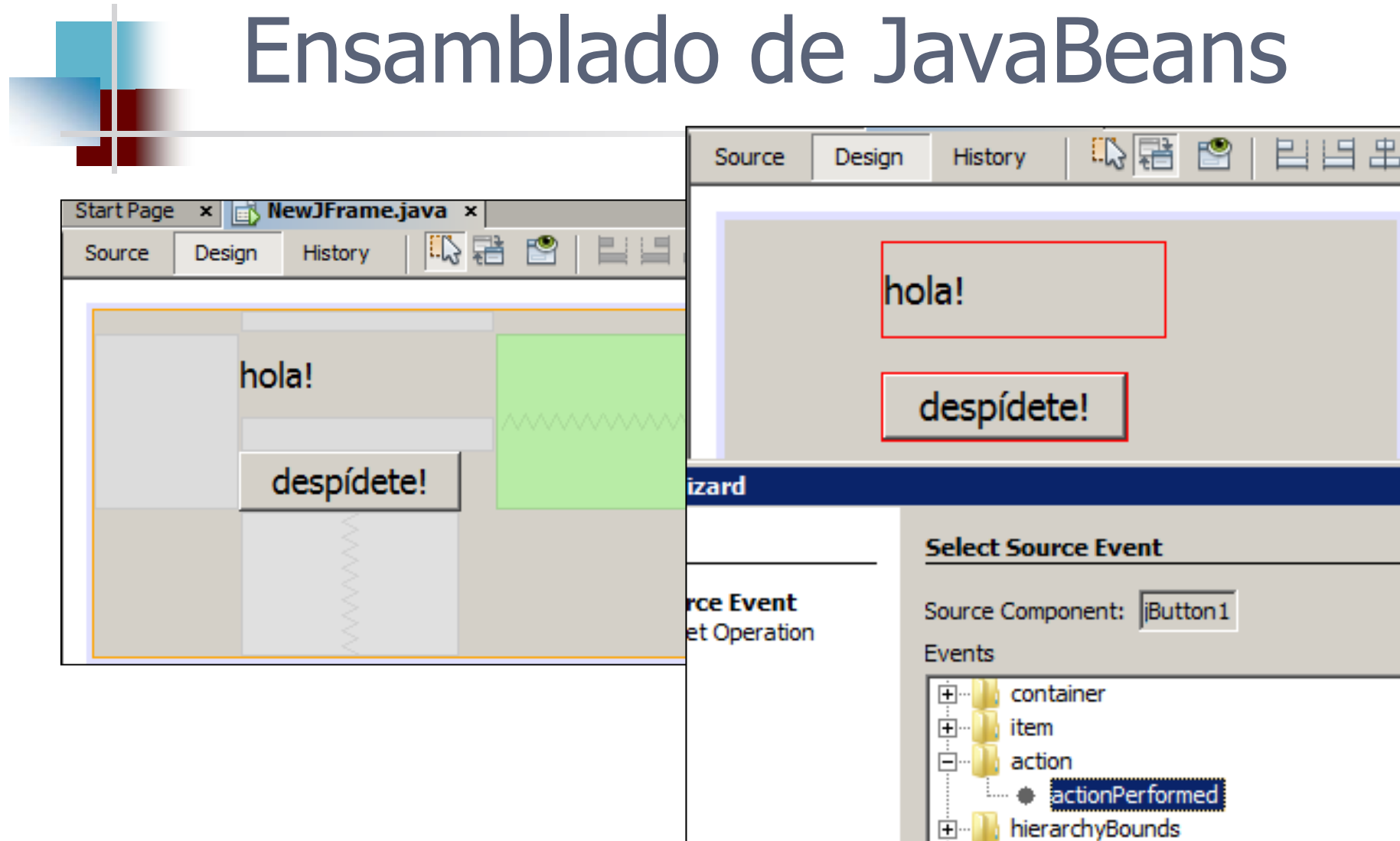
- Crear un archivo JAR (incluye todas las clases y el archivo de manifiesto)

```
jar cfm FechaBean.jar manifest.tmp *.class
```

- Importar el archivo JAR a la paleta de Beans

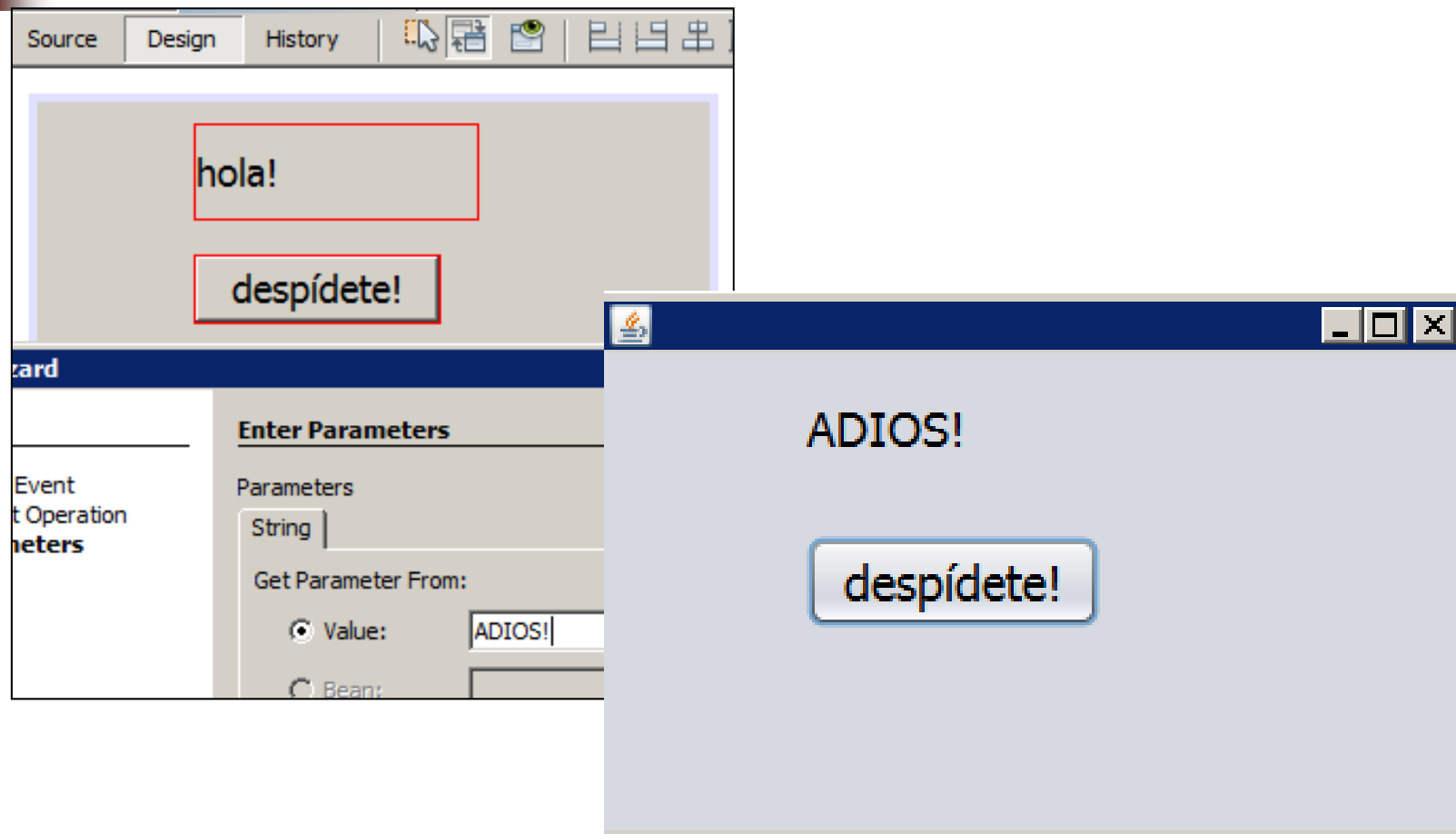
# Net Builder:

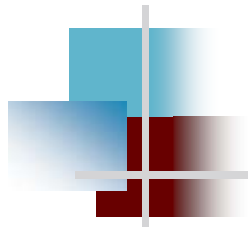
## Ensamblado de JavaBeans



<http://docs.oracle.com/javase/tutorial/javabeans/index.html>

# Net Builder: conexión





# Código generado

---

```
jButton1.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        jButton1ActionPerformed(evt);  
    }  
})
```

```
private void  
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    jLabel1.setText("ADIOS!");  
}
```

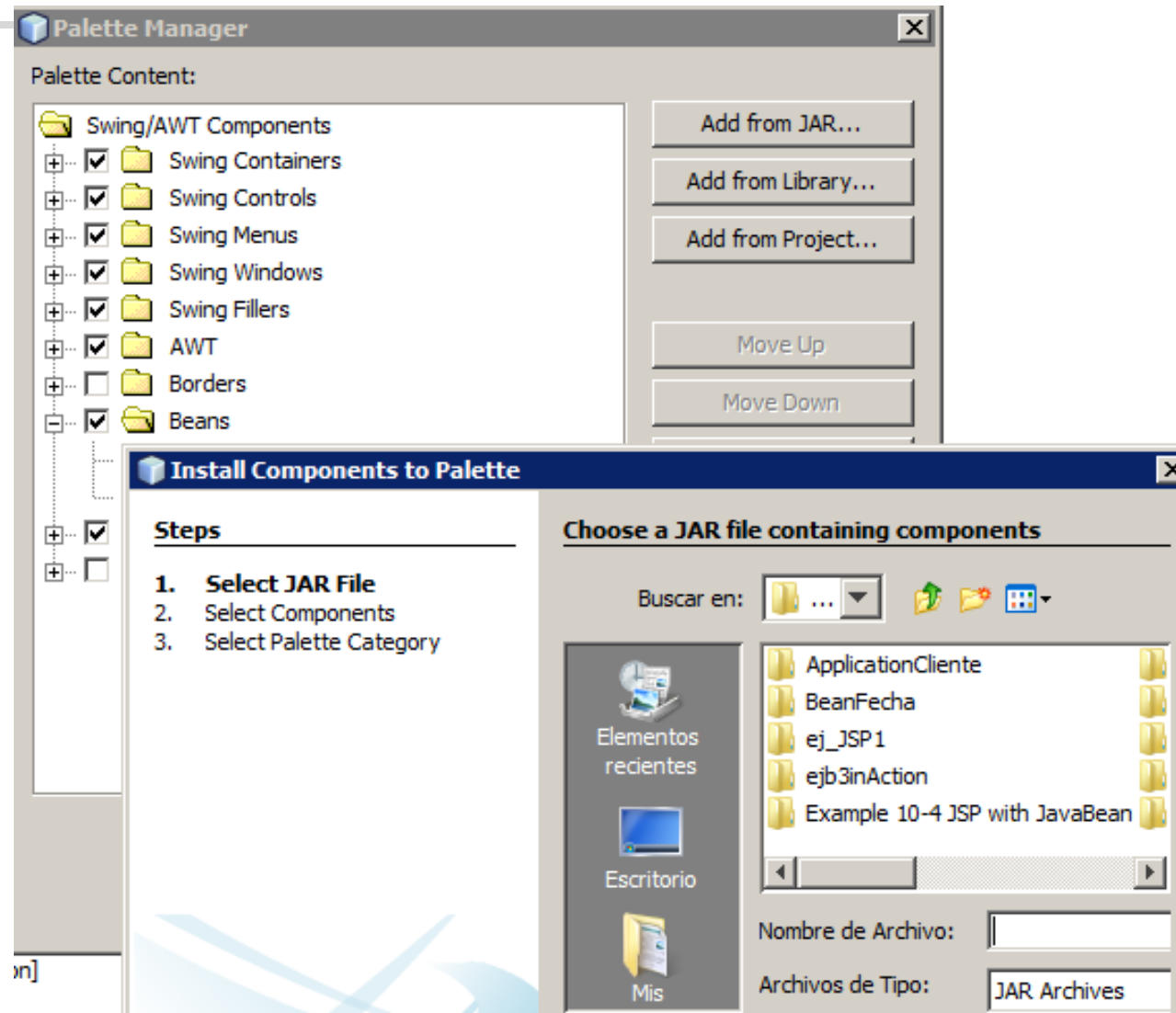


# Un JavaBean no visual... compilado como .jar

---

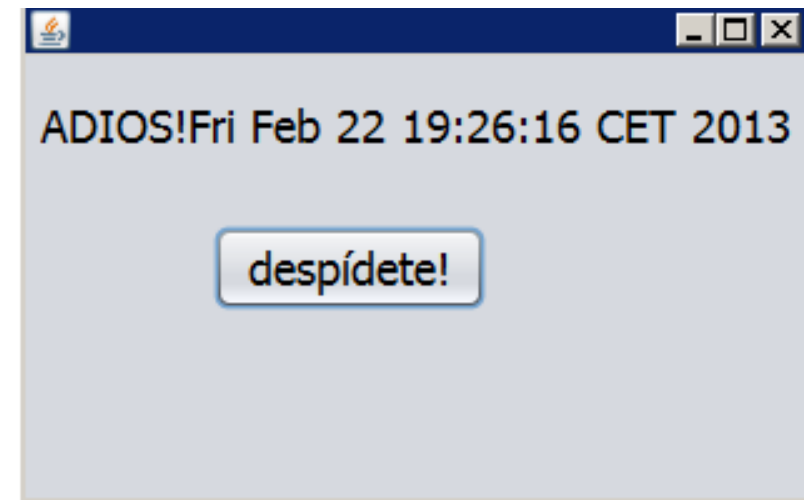
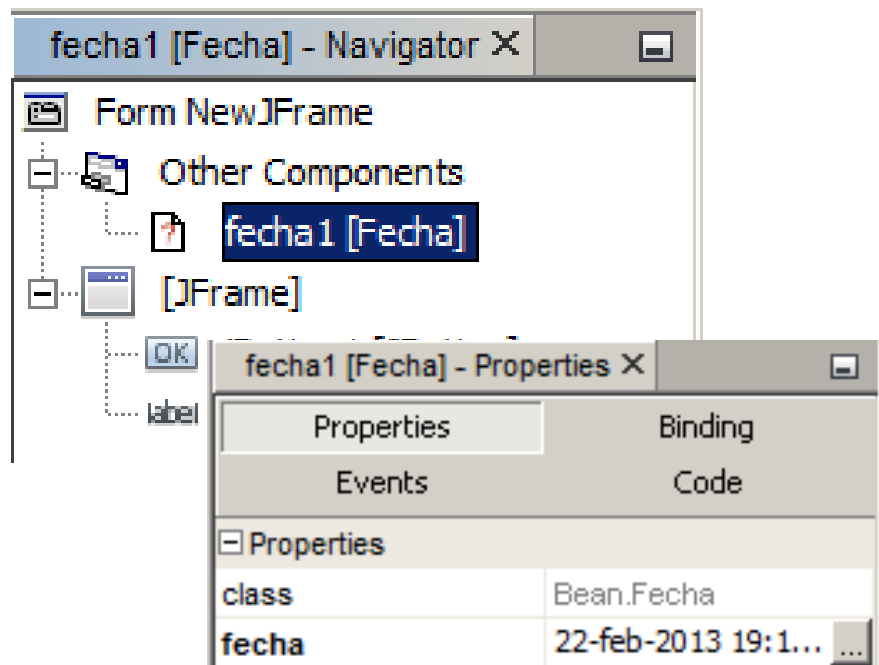
```
public class Fecha implements Serializable {  
  
    private Date fecha;  
  
    public Fecha() {  
        fecha = new Date();  
    }  
    public Date  getFecha() {  
        fecha= new Date();  
        return fecha;  
    }  
    ...  
  
}
```

# Net Builder: añadir Beans a la paleta

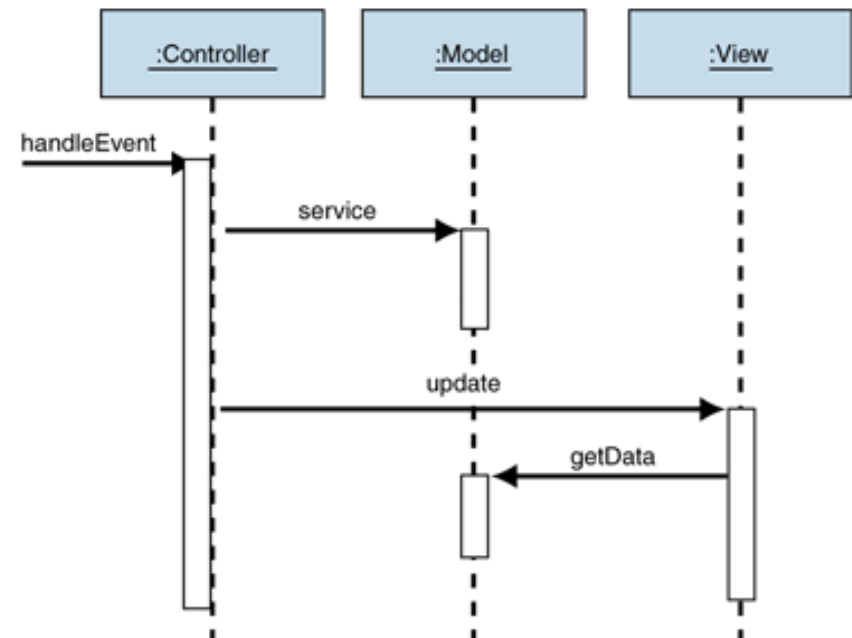
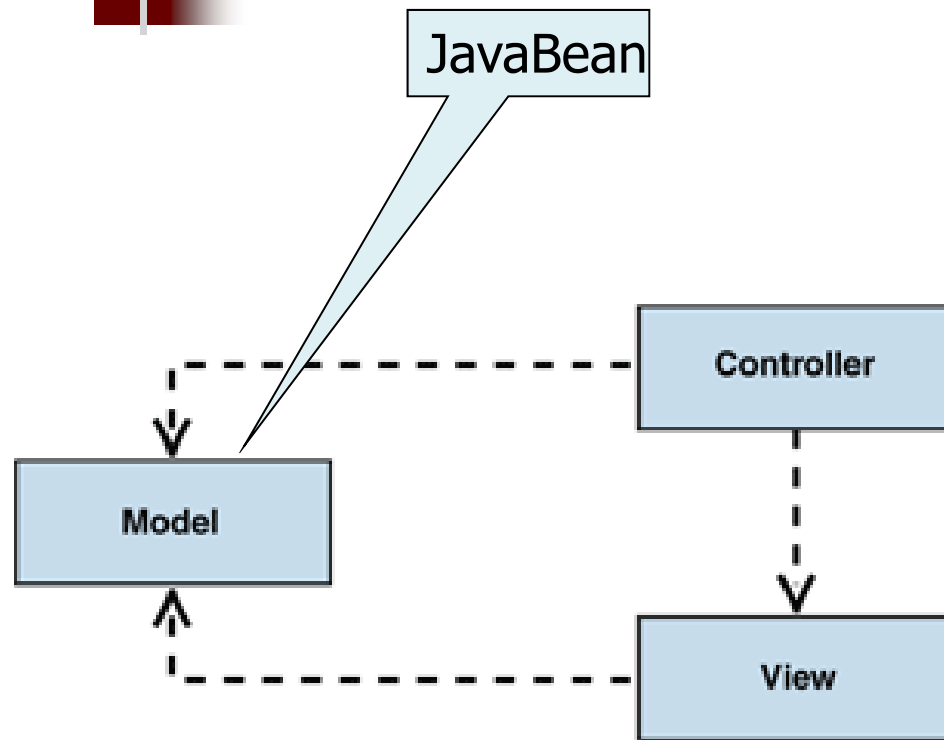


# Utilizar el JavaBean Fecha

```
private void  
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    jLabel1.setText("ADIOS! " + fecha1.getFecha().toString());  
}
```



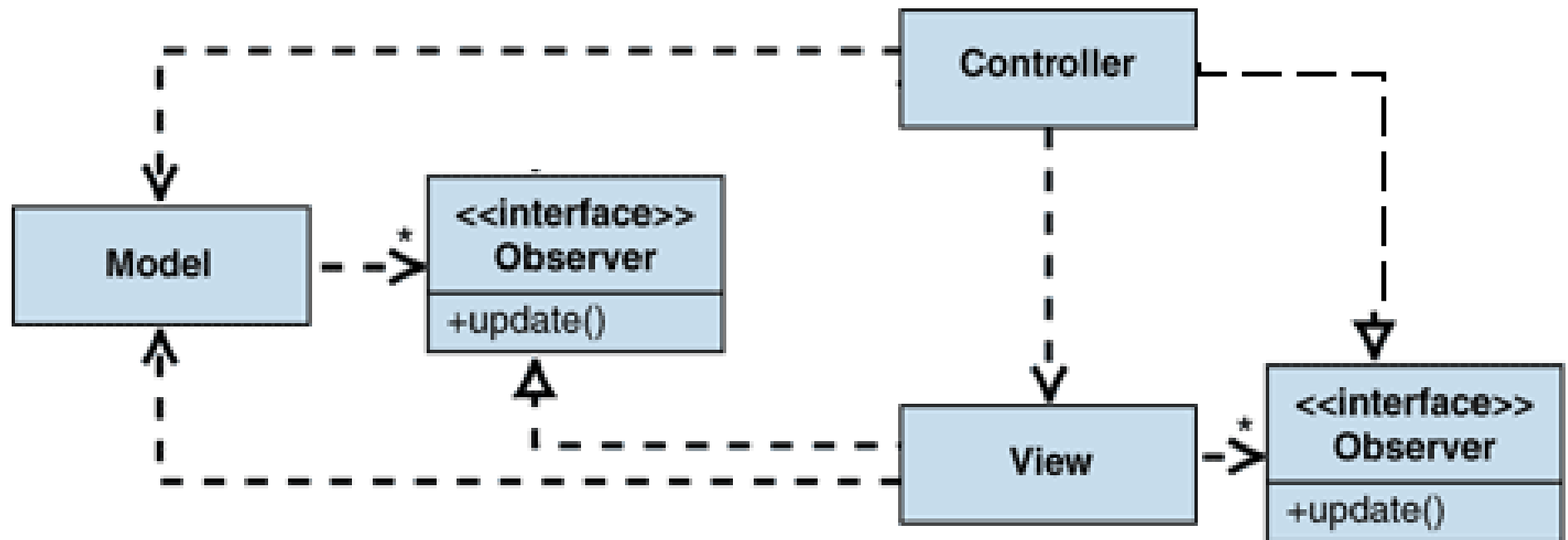
# JavaBeans y Modelo Vista Controlador





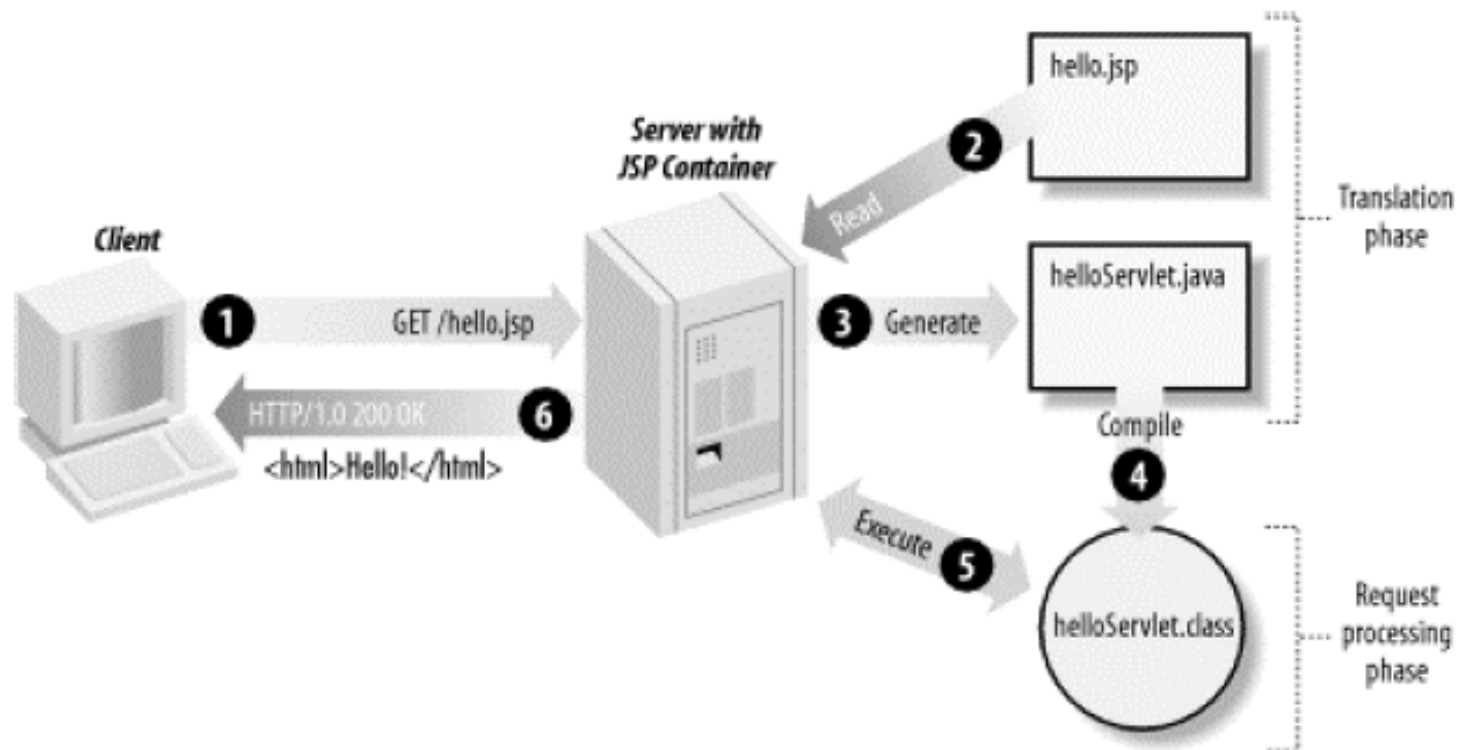
# MVC activo

## JavaBeans lanza eventos

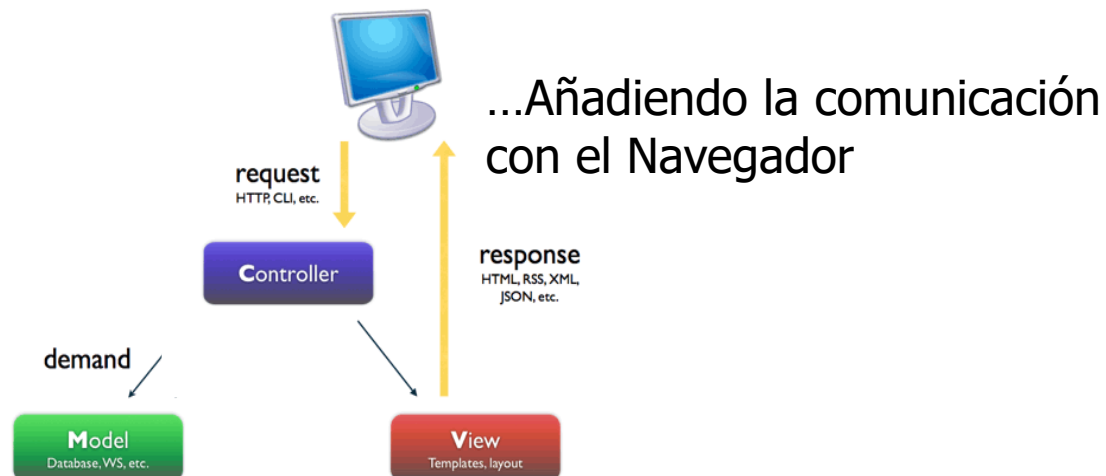
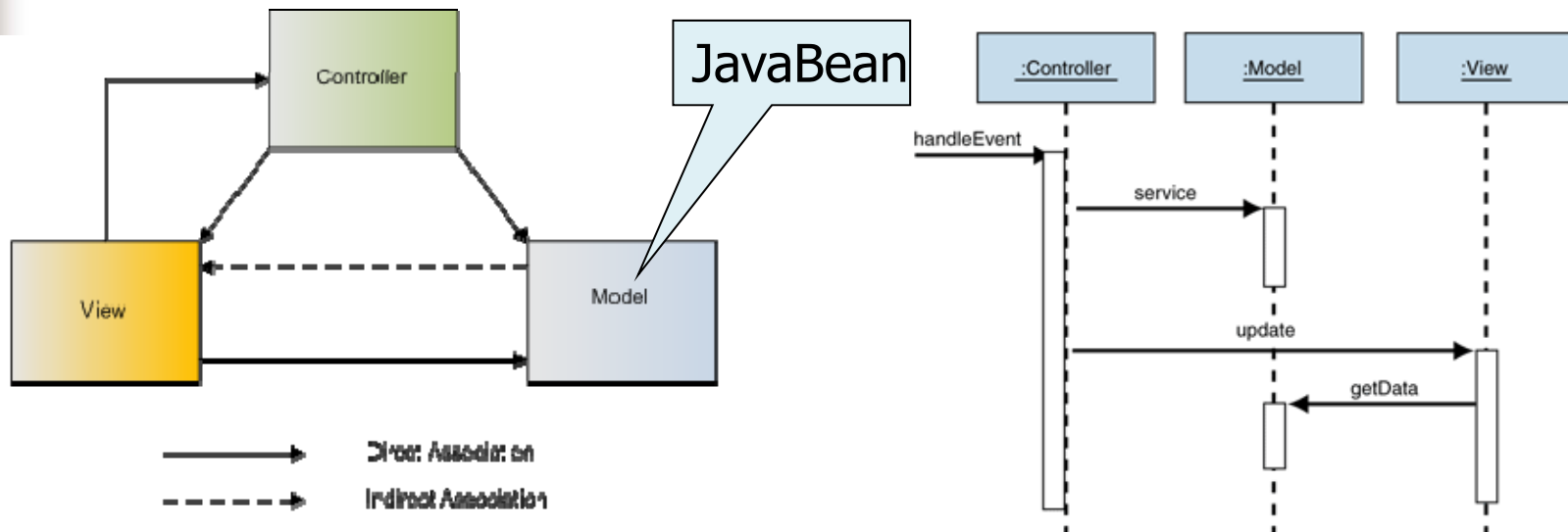


# JSP/JavaBean

- Diseño orientado a componentes con JSP/JavaBean
- Esquema de funcionamiento de JSP

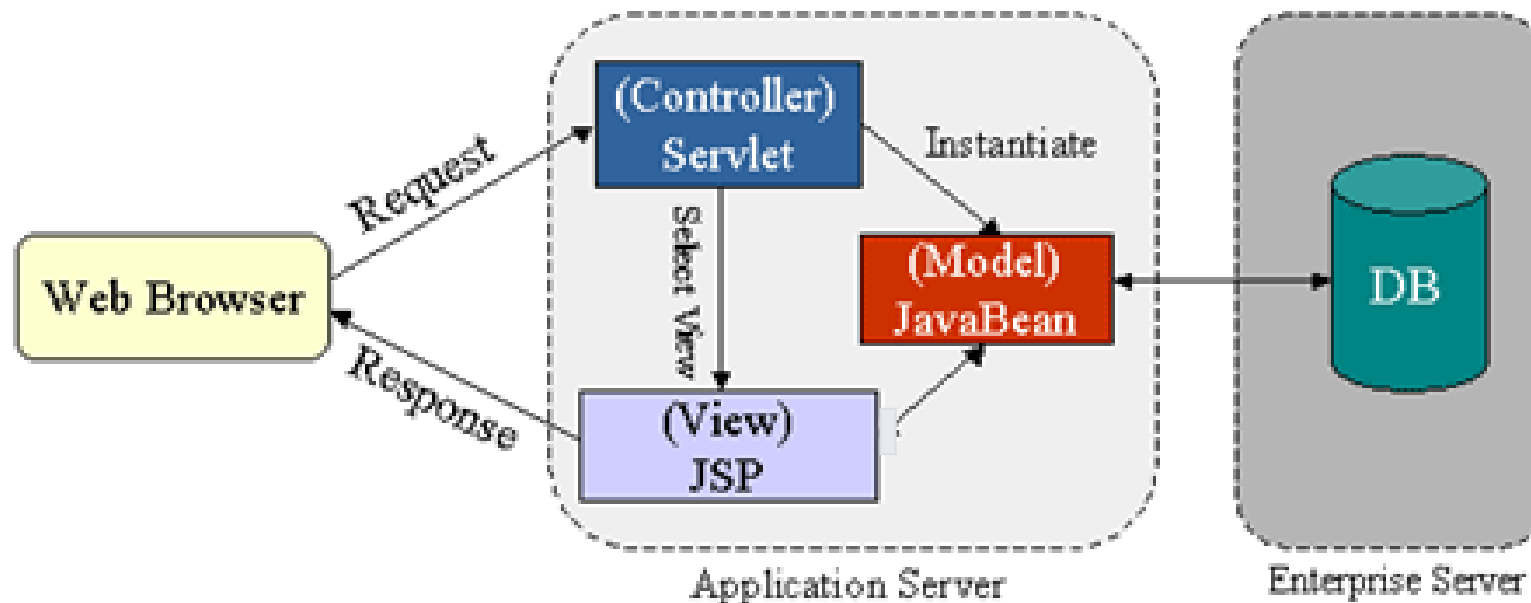


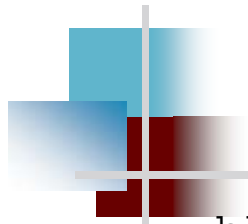
# Modelo Vista Controlador



# El modelo 2 de JSP (MVC pasivo)

- MVC y JavaBean como modelo
  - El contenedor (Tomcat) recibe la petición (vista 1, html/jsp) y la redirige al controlador (Servlet)
  - El controlador crea o accede al modelo (JavaBean) y se lo pasa a la vista 2 (JSP, que fabrica la respuesta)





# JavaBean Persona

---

```
public class Persona implements java.io.Serializable {  
    private String nombre = null;  
    private boolean empleado= false;  
    public Persona() {}  
    public String getNombre() {  
        return nombre;  
    }  
    public void setNombre(final String value) {  
        nombre = value;  
    }  
    public boolean isEmpleado() {  
        return empleado;  
    }  
    public void setEmpleado(final boolean value) {  
        empleado= value;  
    }  
}
```

# Integración JSP y JavaBean

```
<html> <body>
```

```
    <form name="test" method="POST" action="Confirm.jsp">
```

```
        Nombre : <input type="text" name="nombre" size="30"><br/>
```

```
        Estado :
```

```
        <select name="empleado">
```

```
            <option value="false">En paro</option>
```

```
            <option value="true">Empleado</option>
```

```
        </select>
```

```
        <input type="submit" value="Test">
```

```
    </form>
```

```
...
```

```
<jsp:useBean id="persona" class="modelo.Persona" scope="page">
```

```
    <jsp:setProperty name="persona" property="*" />
```

```
</jsp:useBean>
```

```
<html><body>
```

```
    Nombre      : <jsp:getProperty name="persona" property="nombre" /><br/>
```

```
    Empleado?   : <jsp:getProperty name="persona" property="empleado" /><br/>
```

```
<br/>
```

Confirm.jsp

Nombres de atributos (Persona) y  
parámetros (GET/POST) idénticos



# MVC: servlet Controlador

```
<html><body>
```

```
<form name="test" method="POST" action="controlador">
```

Controlador.java

```
@Override
```

```
protected void doPost(HttpServletRequest request,  
    HttpServletResponse response)
```

```
    throws ServletException, IOException {
```

```
    //con los parámetros recibidos-> buscar/crear un javaBean
```

```
    RequestDispatcher rd =
```

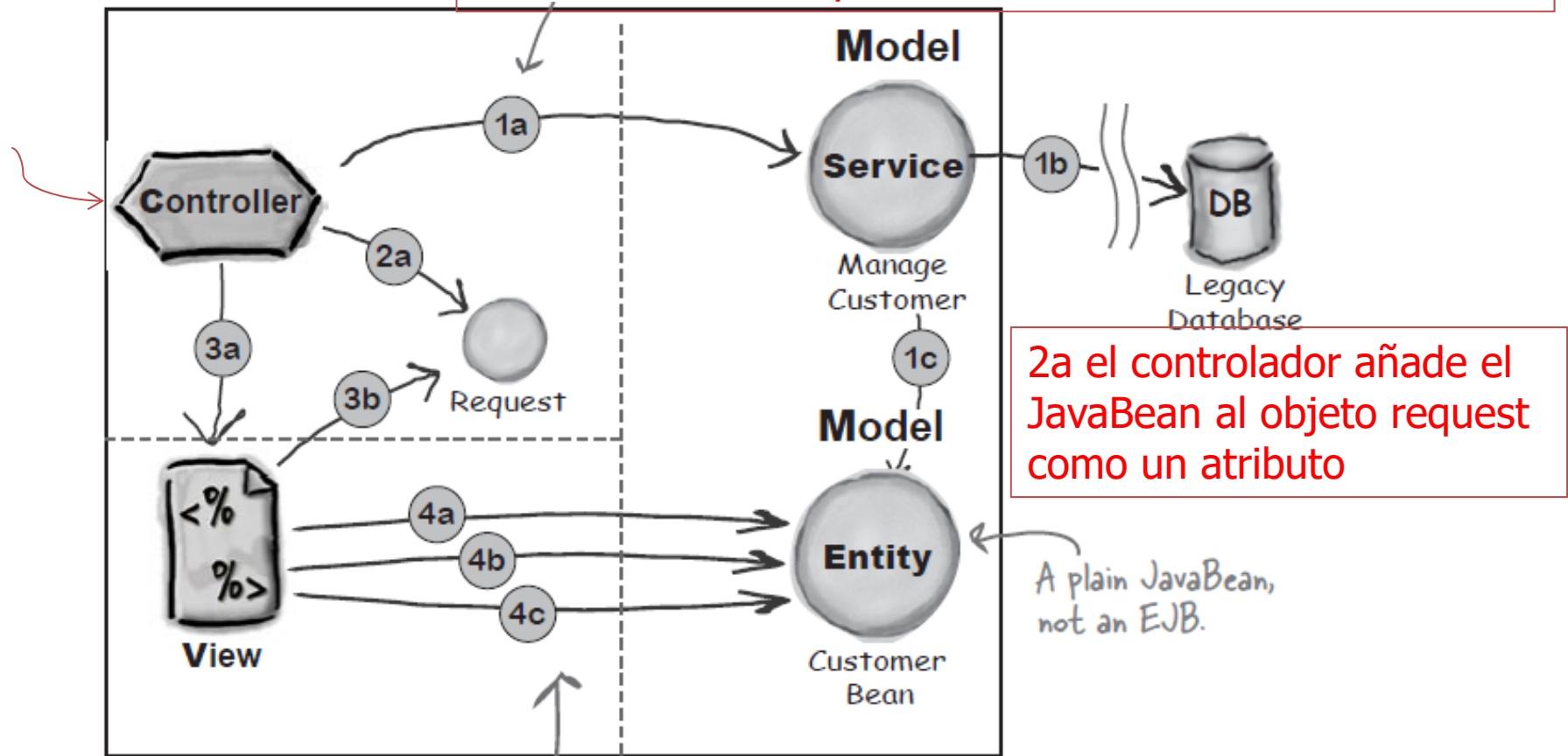
```
    request.getRequestDispatcher("Confirm.jsp");
```

```
    rd.forward(request, response);
```

```
}
```

# MVC

1a.b.c el controlador invoca un servicio que recupera un JavaBean de la BD y se lo devuelve al controlador

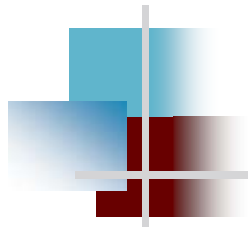


2a el controlador añade el JavaBean al objeto request como un atributo

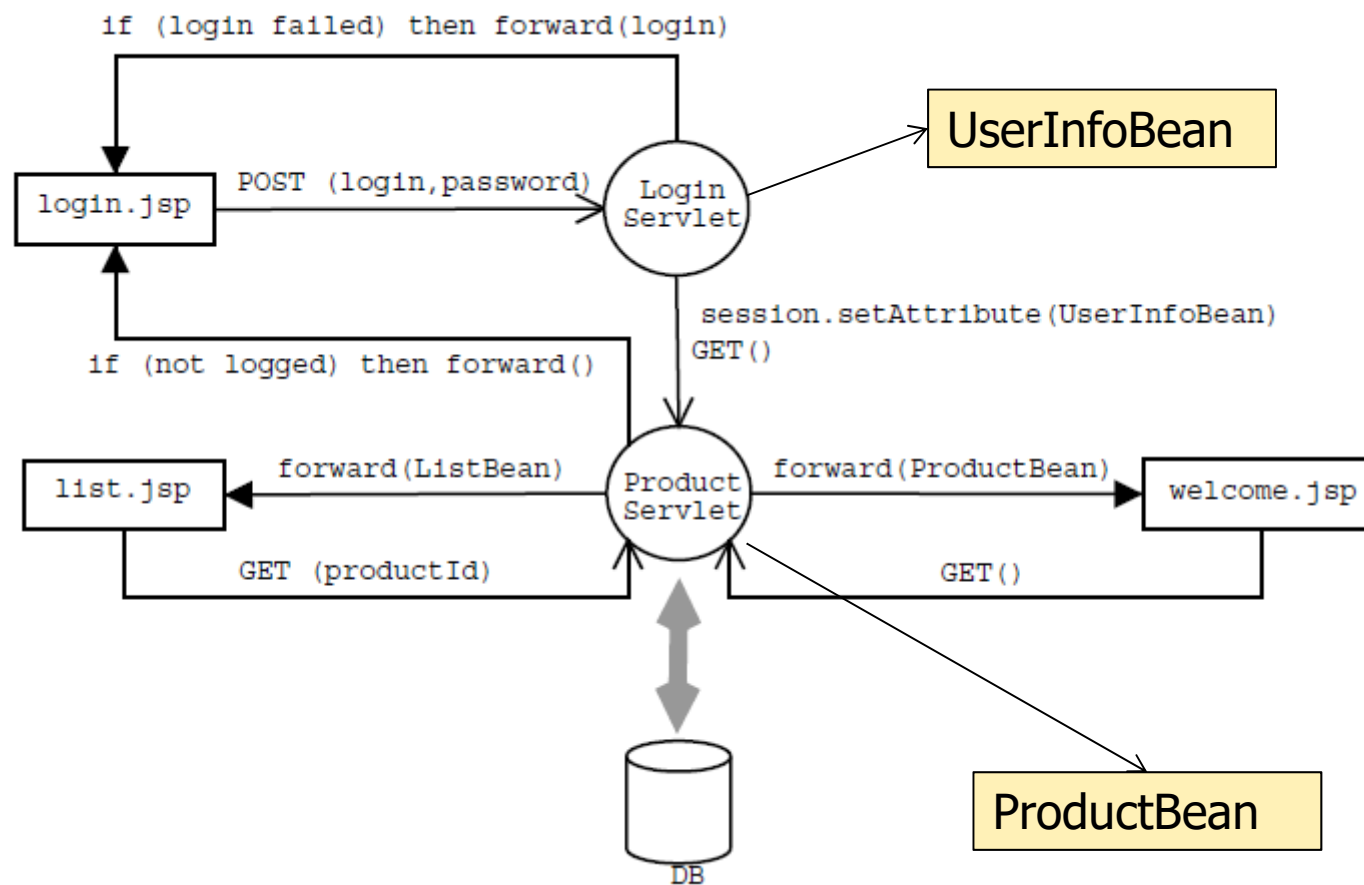
3a.b la vista recibe el Bean

4.a.b.c la vista accede a las propiedades del JavaBean





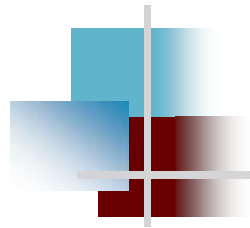
# JavaBeans: conexiones





## 2.4. Componentes en Android

---



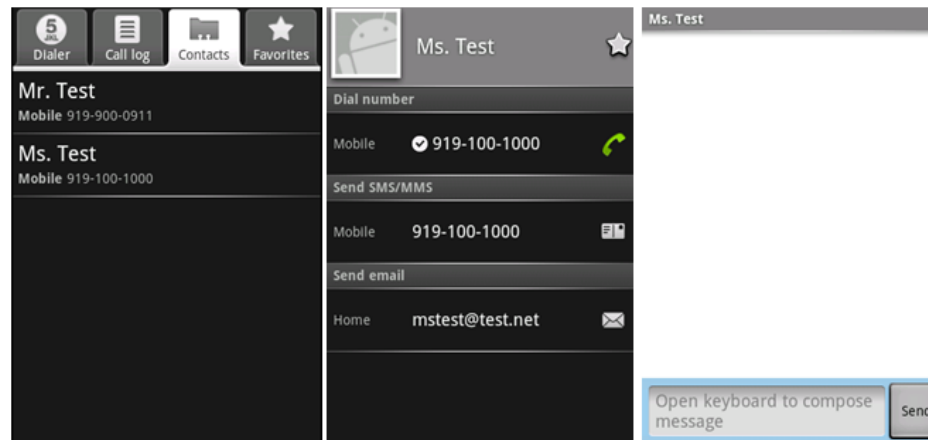
# Componentes en Android

- Android está basado en componentes
- Cuatro tipos básicos de componentes (y las **intents** como forma de comunicación):

Components	Description
Activity	UI component typically corresponding to one screen
Service	Background process without UI
Broadcast Receiver	Component that responds to broadcast Intents
Content Provider	Component that enables applications to share data

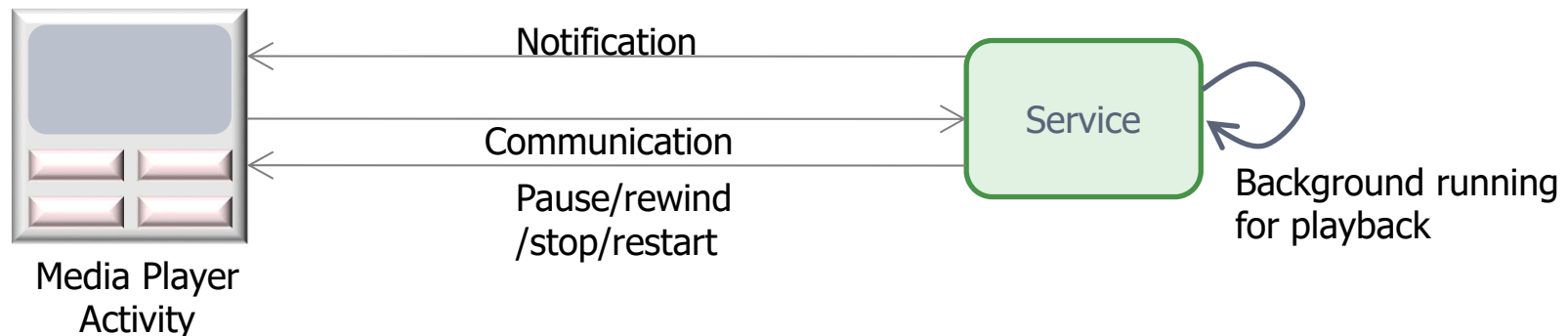
# Componentes: Activity

- Una actividad es por lo general una pantalla:
  - Controles de interfaz de usuario (vistas).
  - Reacciona a eventos del usuario.
- Una aplicación típica consiste en varias actividades
  - Pasar a la siguiente pantalla significa comenzar una nueva actividad.



# Componentes: Service

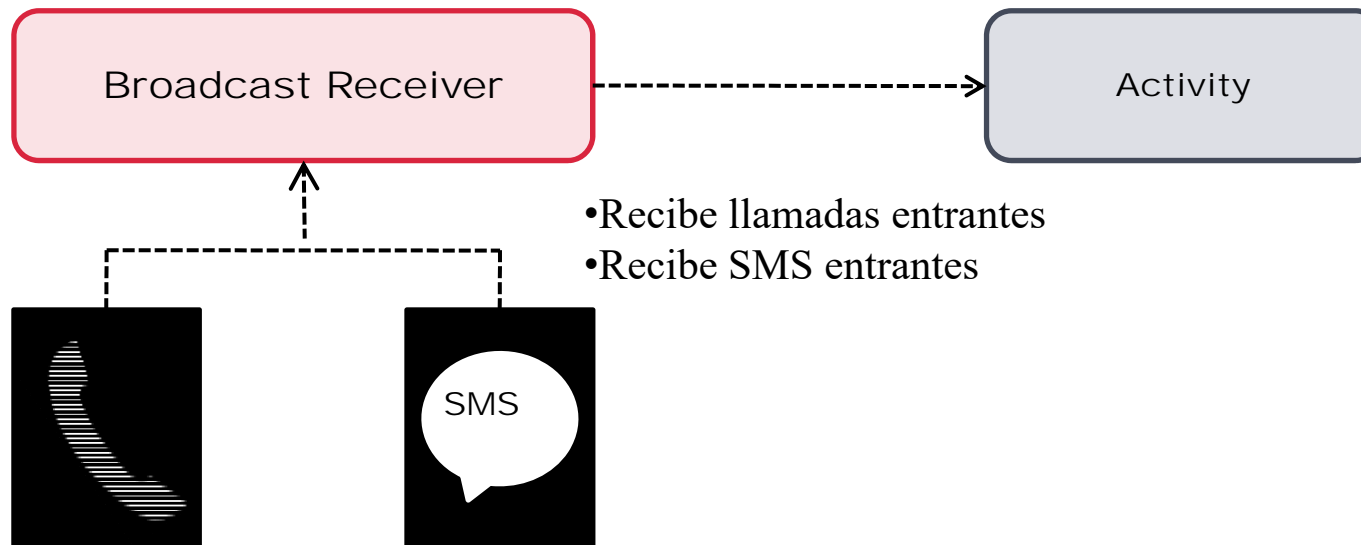
- Sin interfaz de usuario
  - Se ejecuta en segundo plano un período de tiempo indefinido.
- Ejemplo: reproductor de música, descarga de red, etc



# Componentes:

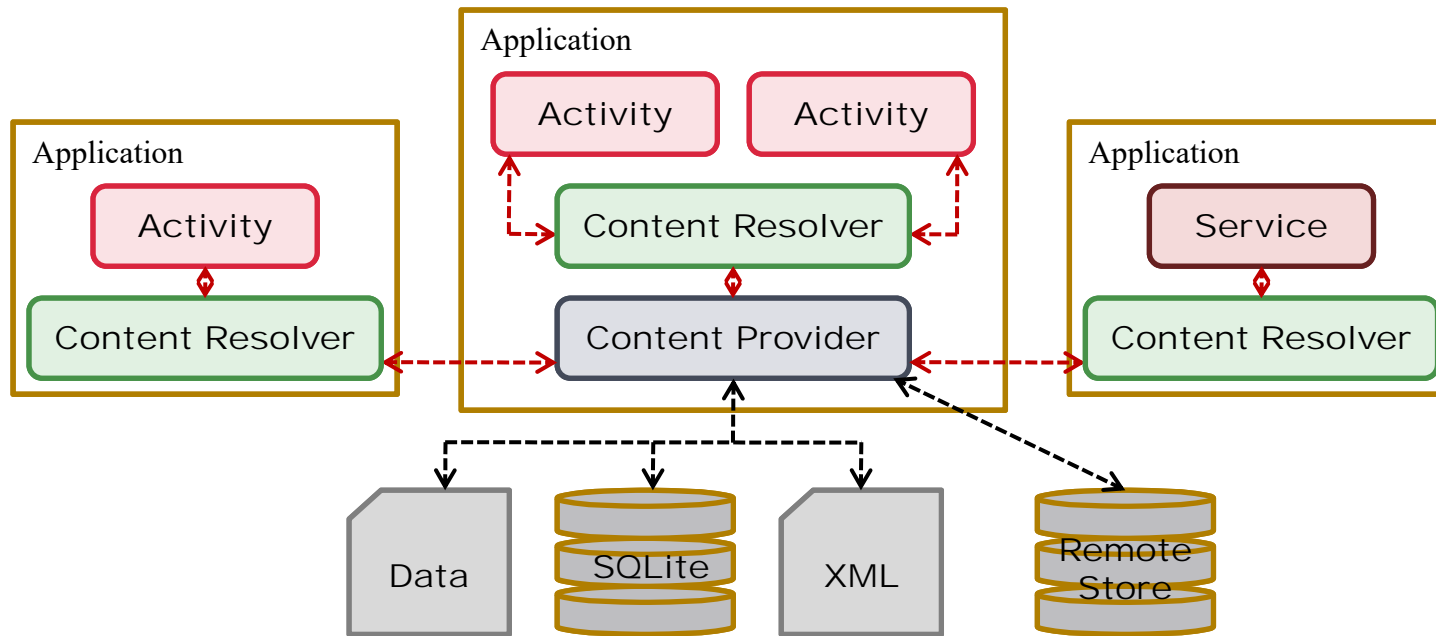
## Broadcast Receivers

- Un receptor es un componente que recibe y reacciona a la llegada de emisiones (intents)
- Muchas emisiones se originan en el código del sistema:
  - Por ejemplo “la batería está baja”



# Componentes: Content Providers

- Un **proveedor de contenido** hace que los datos estén disponibles para las aplicaciones.
  - Datos almacenados en el sistema de archivos, en SQLite, archivos XML...



# Intents

---

- Un **Intent** es un objeto (un mensaje simple) que representa una "intención de hacer algo"
- `android.content.Intent`
  - `VIEW_ACTION`
  - `EDIT_ACTION`
  - `PICK_ACTION`
  - `WEB_SEARCH_ACTION`
  - `SYNC_ACTION`





# Un Intent consiste en:

---

- La Acción
  - (MAIN/VIEW/EDIT/PICK/DELETE/DIAL/etc.)
- La información necesaria para operar sobre ella (URI)

```
startActivity(new Intent(Intent.VIEW_ACTION,  
    Uri.parse("http://www.fhnw.ch"));
```

```
startActivity(new Intent(Intent.VIEW_ACTION,  
    Uri.parse("geo:47.480843,8.211293"));
```

```
startActivity(new Intent(Intent.EDIT_ACTION,  
    Uri.parse("content://contacts/people/1"));
```



# Filtros Intents

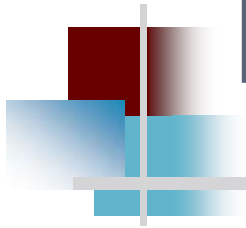
- Un componente puede tener un conjunto cualquiera de filtros intent, declarados en el archivo [AndroidManifest.xml](#)
  - cada uno de ellos declara un conjunto diferente de capacidades → equivalente a la interfaz que implementa (geo y NFC)

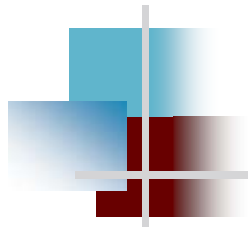
```
<intent-filter>
    <action android:name= "android.intent.action.VIEW" />
    <category android:name= "android.intent.category.DEFAULT" />

    <data android:scheme= "geo" />
</intent-filter>
```

```
<intent-filter>
    <action android:name= "android.nfc.action.NDEF_DISCOVERED" />
    <category android:name= "android.intent.category.DEFAULT" />
    <data
        android:host= "ext"
        android:pathPrefix= "/nfclab.com:shopping"
        android:scheme= "vnd.android.nfc" />
</intent-filter>
```

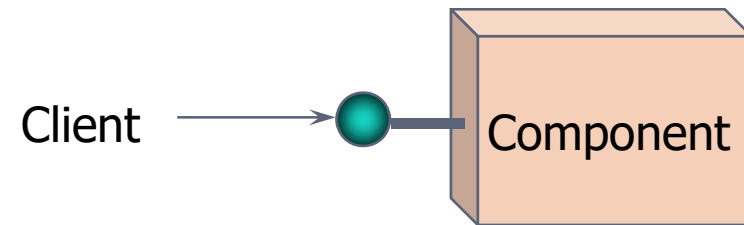
## 2.5. El problema de la heterogeneidad



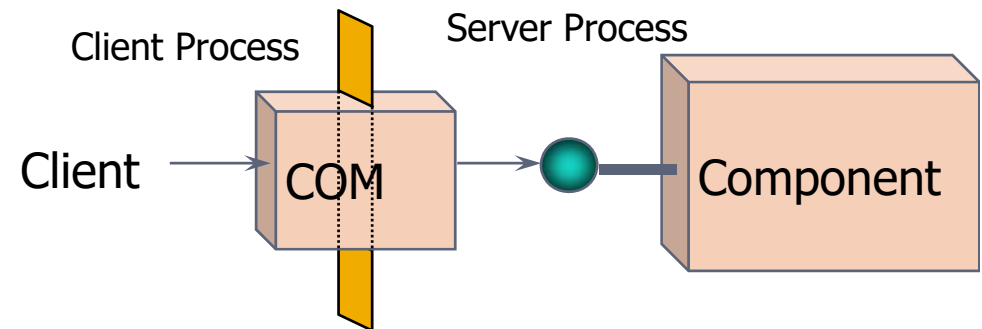


# COM: DLL y EXE

En el mismo proceso  
(DLL)  
Muy rápido

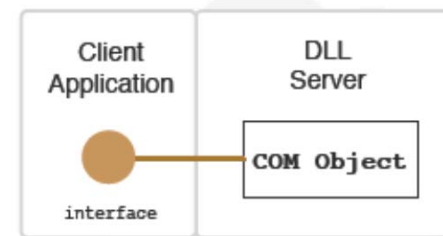


En la misma máquina  
(EXE)  
Menos rápido

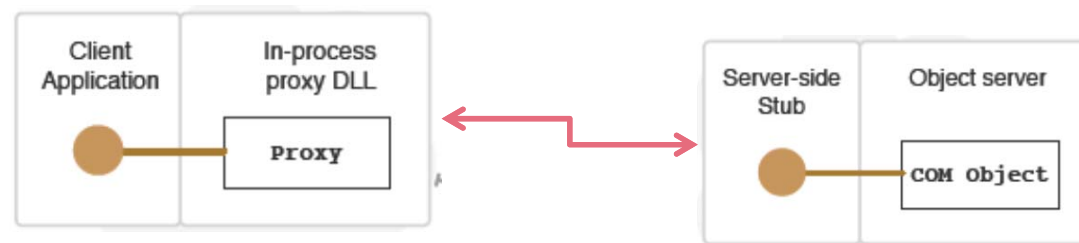


# COM: DLL y EXE

- Componentes ActiveX (.DLL): Residen en el mismo proceso (In-process)

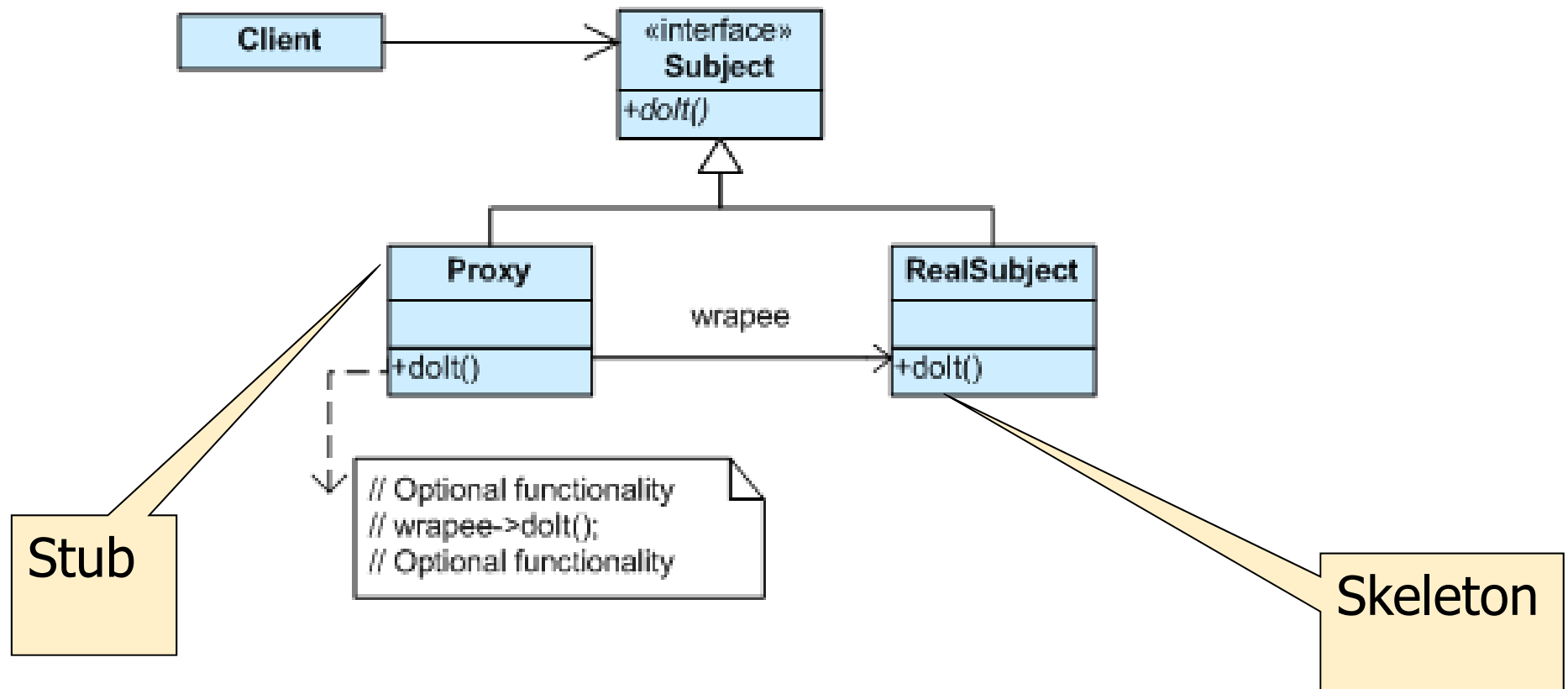


- Componentes ActiveX (.EXE): Residen en otro proceso separado (Out-process)
- Se comunican a través de Windows
- Se pueden crear a partir del mismo código



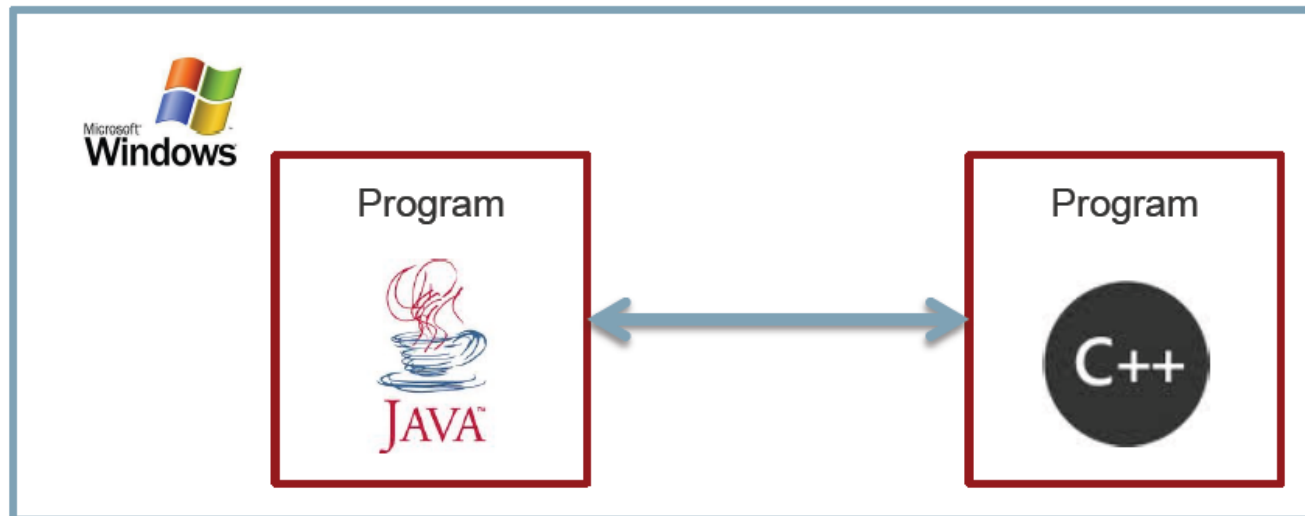
# Stubs y Skeletons

- Stub: patrón "proxy" del lado del cliente:



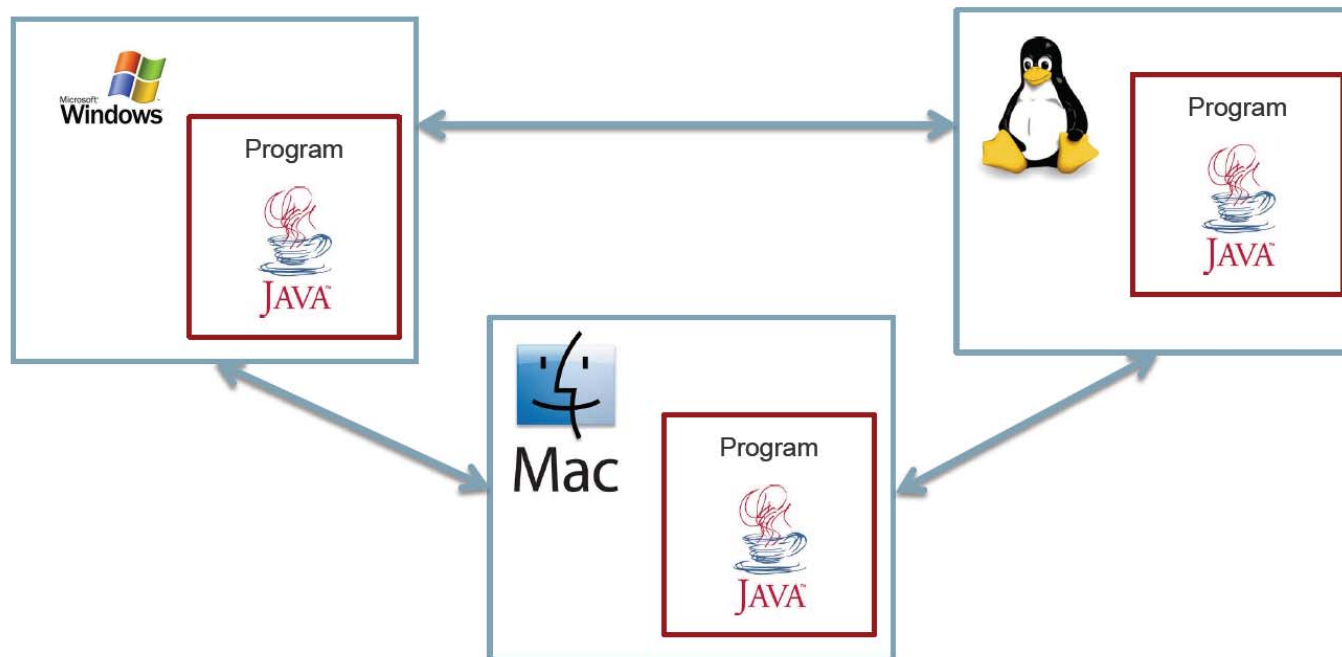
# Heterogeneidad

- Transparencia de Lenguajes: interoperabilidad de los programas en la misma plataforma, utilizando lenguajes de programación diferentes



# Heterogeneidad

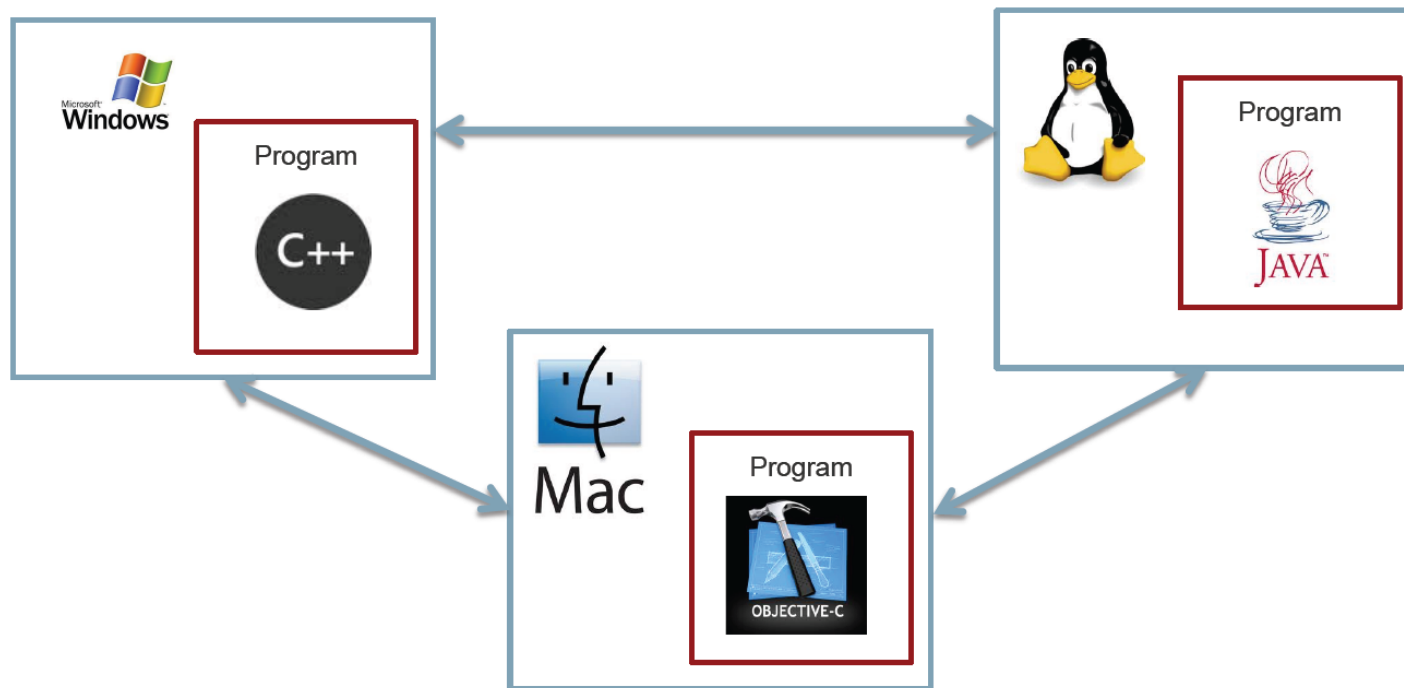
- Transparencia de ubicación: interoperabilidad de los programas en distintas máquinas, utilizando el mismo lenguaje de programación

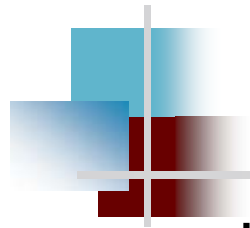




# Heterogeneidad general

- Cualquier ubicación y cualquier lenguaje: **Sistemas distribuidos**

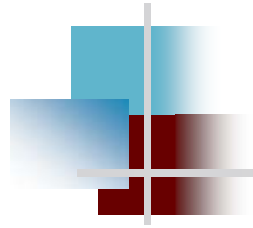




# A tener en cuenta...

---

- Llamadas:
  - Procedimientos, mensajes, ...
- Paso de parámetros:
  - por valor, por referencia, ...
- Tipos y representación de datos:
  - valores, referencias a objetos, matrices, enumeraciones, clases,
- Entorno en tiempo de ejecución:
  - Gestión de memoria, recolección de basura



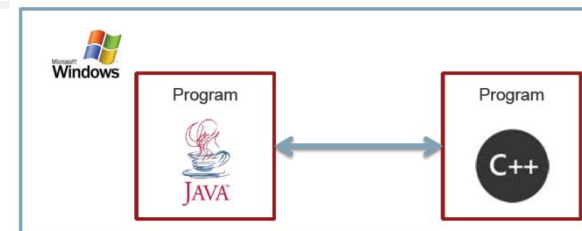
# Transparencia de Lenguajes

---

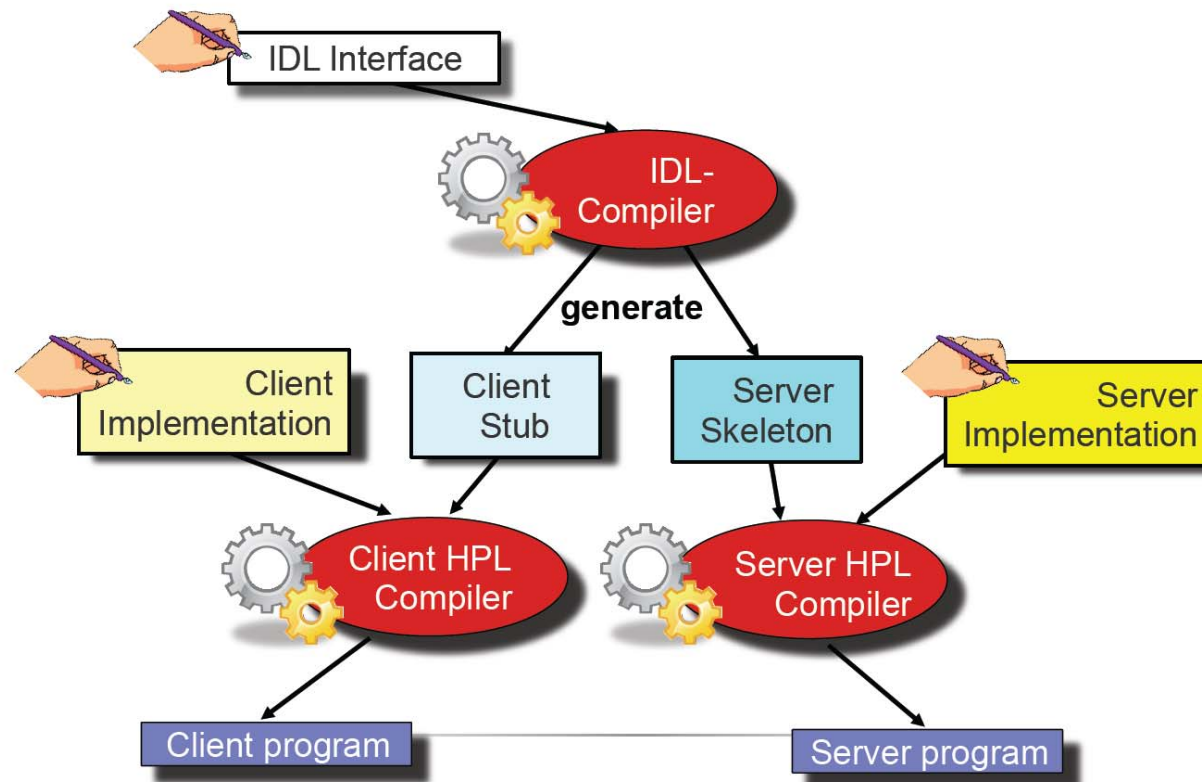
- Solución Microsoft: .NET y CLR
- Estándar CORBA

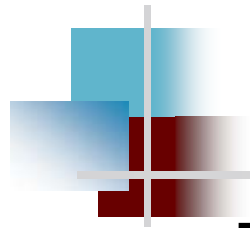
# Transparencia de Lenguajes

- Llamadas:
  - Llamadas estándar (RPC)
- Paso de parámetros:
  - Lenguaje estándar (.NET) o implementación de CORBA para cada lenguaje
- Tipos y representación de datos:
  - Sistemas de tipos (.NET) o traducción de CORBA para cada lenguaje (IDL mapping)
  - Representación en lenguajes estándar o formatos de intercambio (XML)
- Entorno en tiempo de ejecución:
  - Proporcionado por la implementación de CORBA, .NET...



# Generación automática





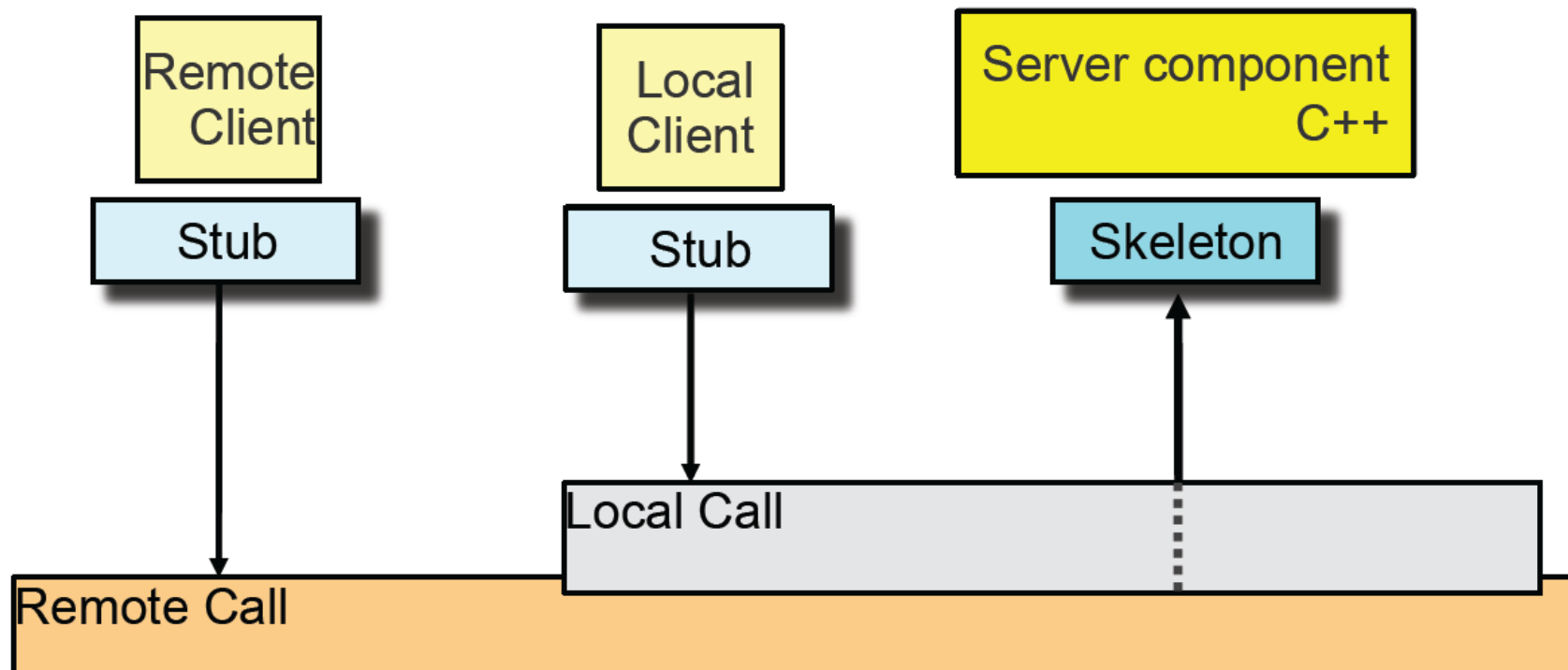
# Transparencia de ubicación

---

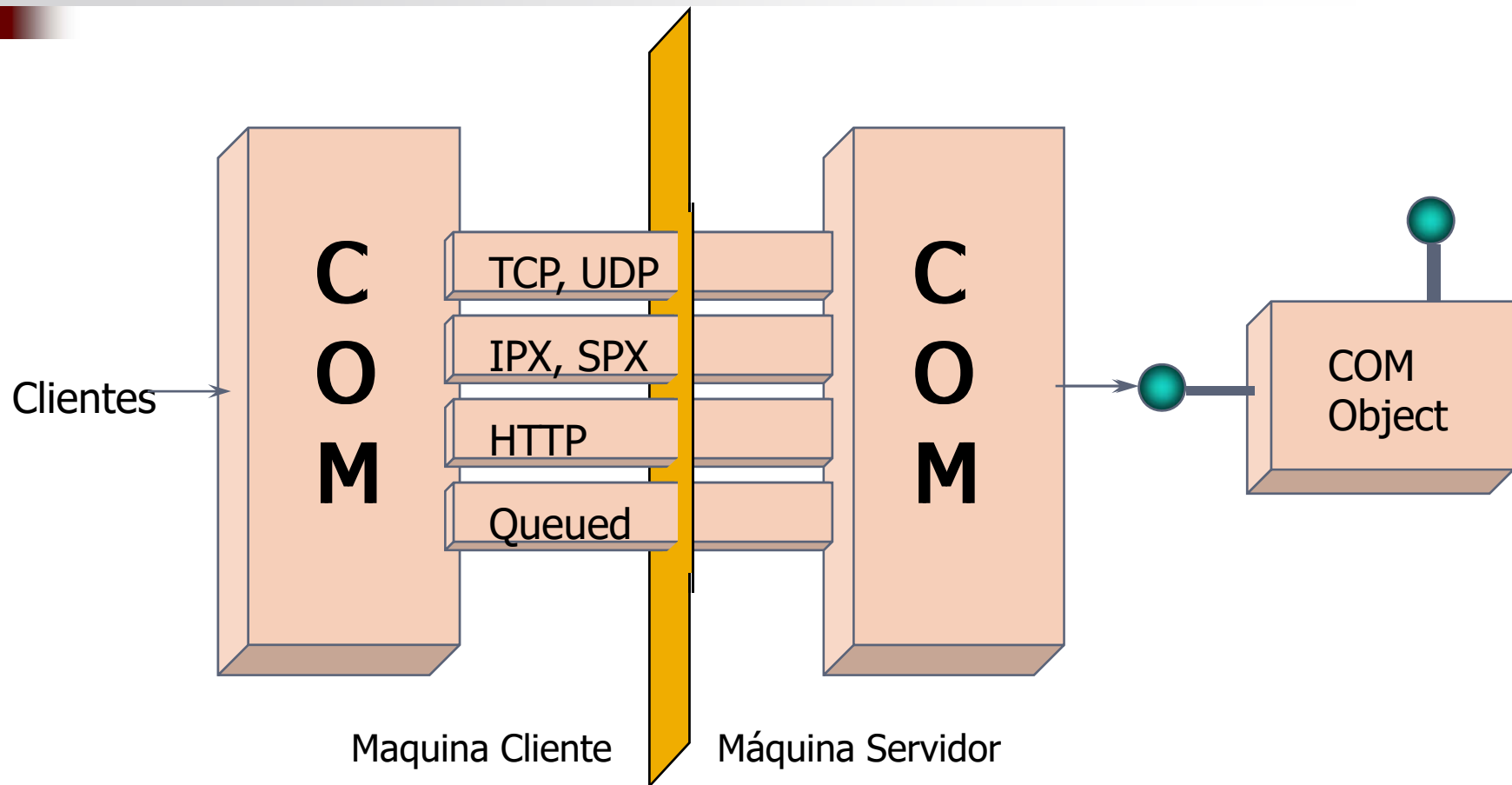
- Transparencia de ubicación
  - Interoperabilidad de los programas independientemente de su ubicación en tiempo de ejecución
- Problema a resolver: Comunicación Transparente
  - Forma transparente de iniciar una llamada local o remota
  - Transporte local o remoto a través de una red de forma transparente
  - ¿Cómo manejar las referencias de forma transparente?

# Transparencia de ubicación

- Variante del patrón Proxy (usando llamadas remotas RPC entre el Stub y el Skeleton)



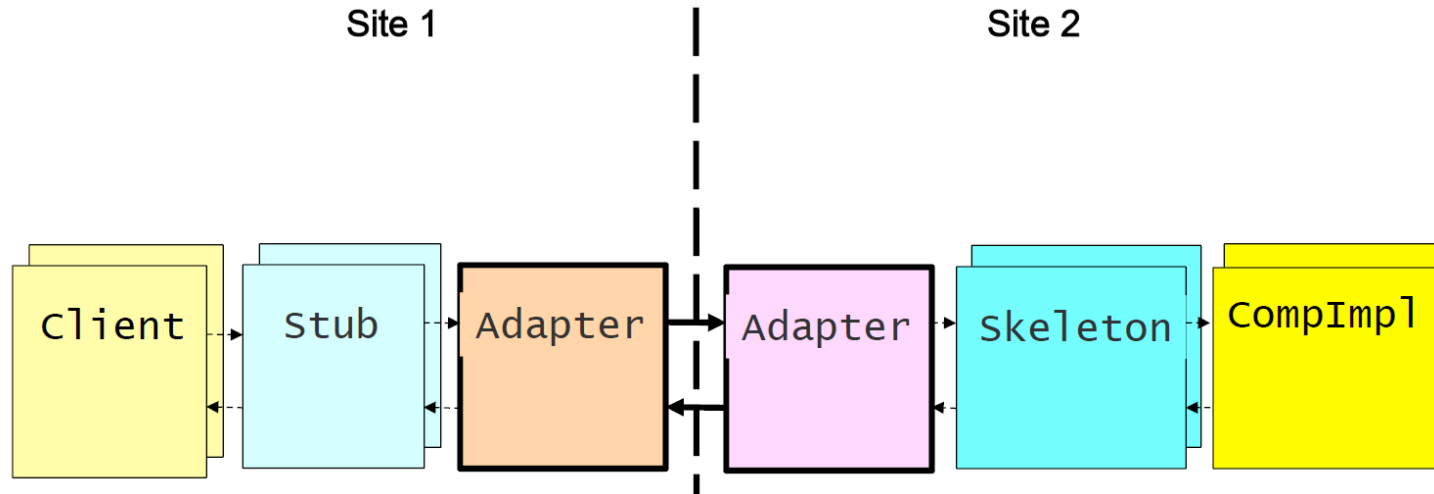
# DCOM (Distributed COM) y COM+





# Sistemas Distribuidos: Marshalling

- Los sistemas distribuidos son totalmente heterogéneos
- Marshalling: Traducir los datos a un formato de intercambio antes y después de la llamada



2.6.

## Sistemas distribuidos y middleware

---





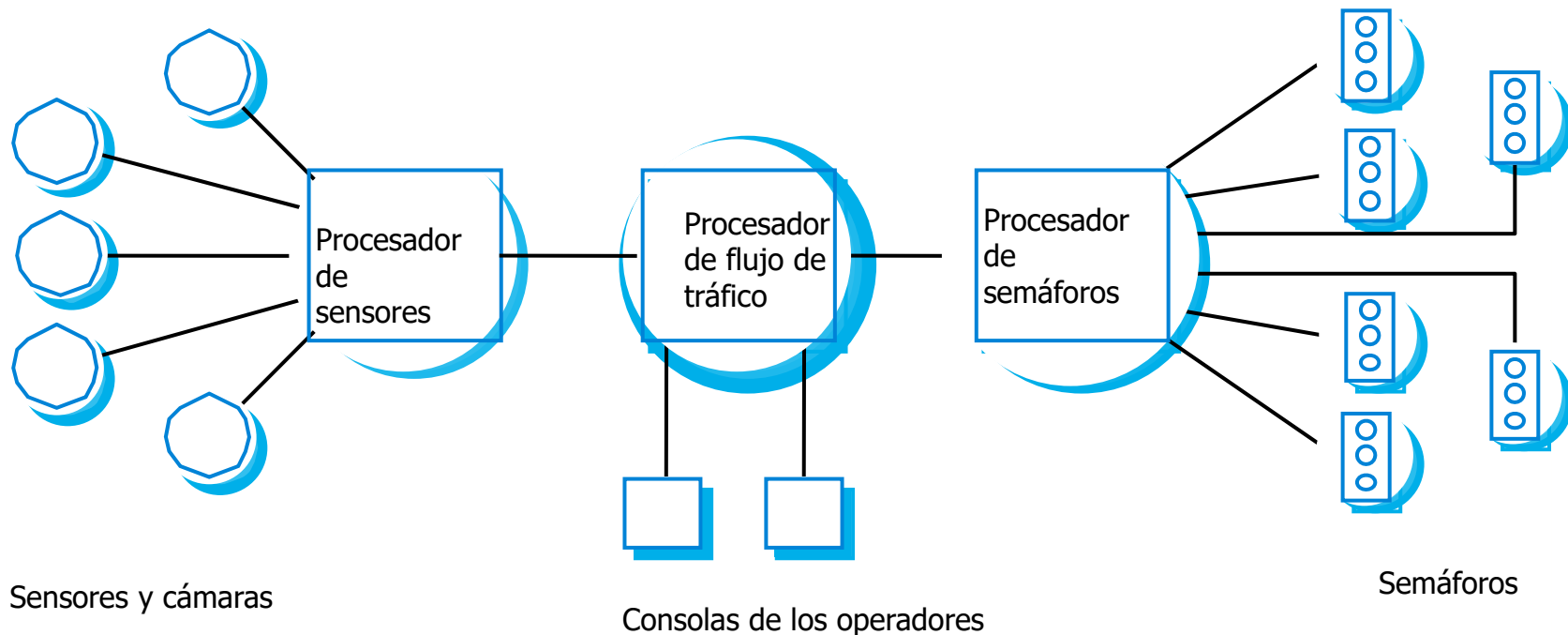
# Arquitectura de un sistema distribuido

---

- Arquitectura maestro-esclavo
  - En sistemas de control y tiempo real
- Arquitectura entre iguales Peer-to-Peer
  - Cuando los clientes y servidores intercambian sus papeles
- Arquitectura cliente-servidor de dos o más capas
  - Un servidor centralizado
- Arquitectura basada en componentes distribuidos
  - Se utiliza cuando los recursos de diferentes sistemas y bases de datos necesitan ser combinadas,
- Arquitectura basada en servicios
  - Utiliza la Web para conectar distintos sistemas (incluso de distintas empresas)

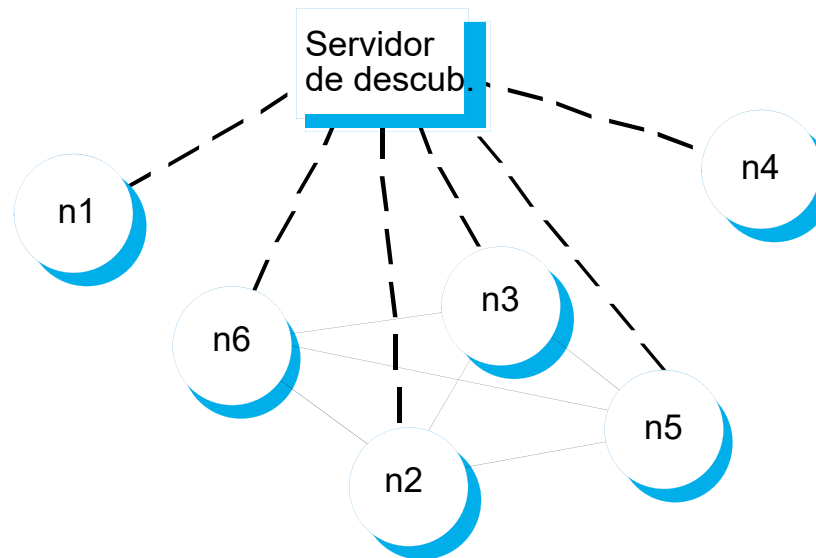
# Maestro/esclavo

- El proceso "maestro" es responsable de la la coordinación y la comunicación y controla los procesos de "esclavos"



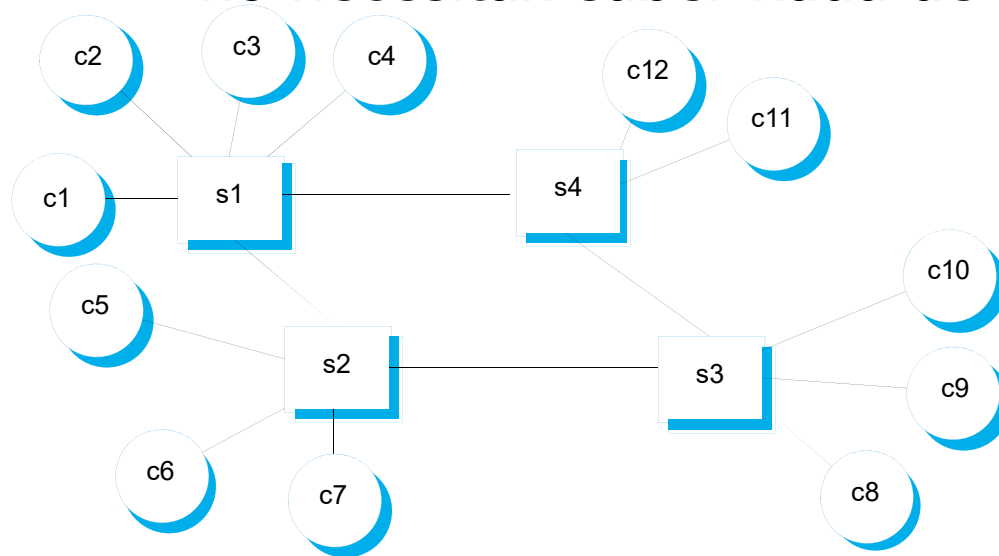
# Peer-to-Peer (P2P)

- Sistemas descentralizados en los que la computación puede ser realizada por cualquier nodo de la red.
- Gran número de ordenadores conectados en red.
- Uso creciente de esta tecnología en sistemas comerciales



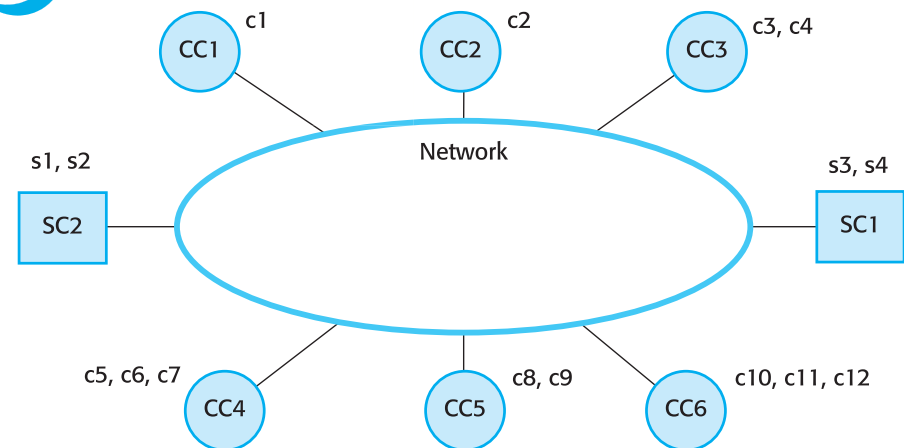
# Arquitectura Cliente Servidor

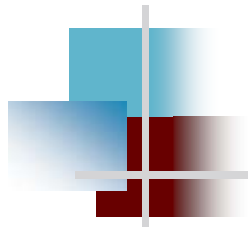
- Los clientes conocen los servidores, pero los servidores no necesitan saber nada de los clientes.



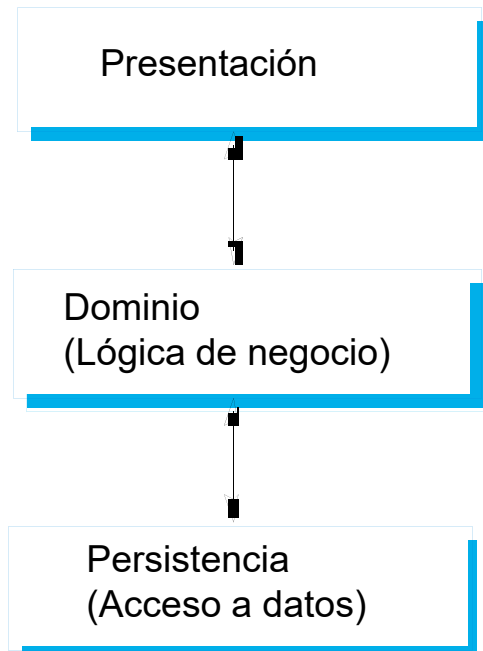
  
Servidores

  
Clientes



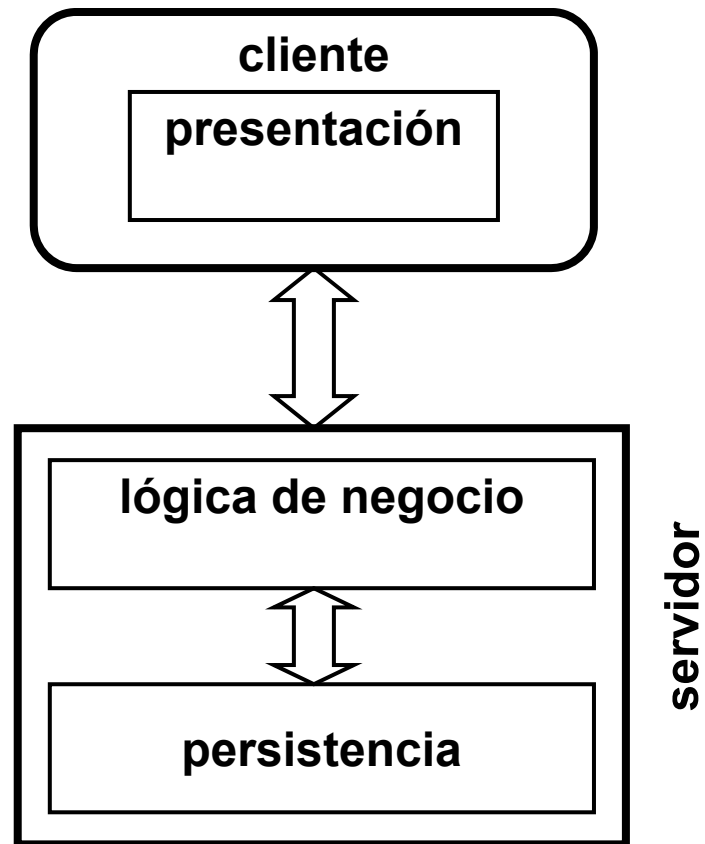
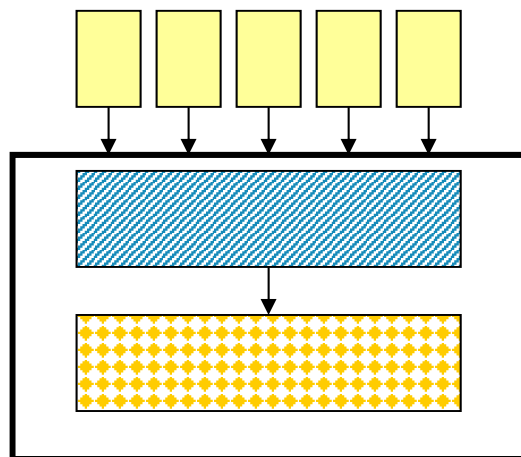
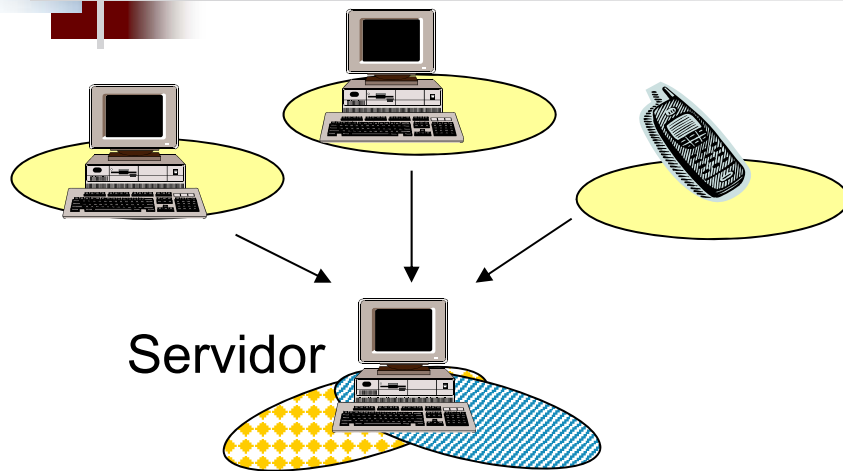


# Arquitectura Cliente Servidor



- Tres capas básicas de software (lógicas, layers):
  - Presentación: entrada/salida de los usuarios
  - Dominio (o Lógica de negocio): responsable del procesamiento de la información que tiene lugar en la aplicación.
  - Persistencia (o Acceso a datos): contiene los datos de la aplicación.
- Se pueden distribuir en 1,2,3...capas físicas o niveles (tiers)

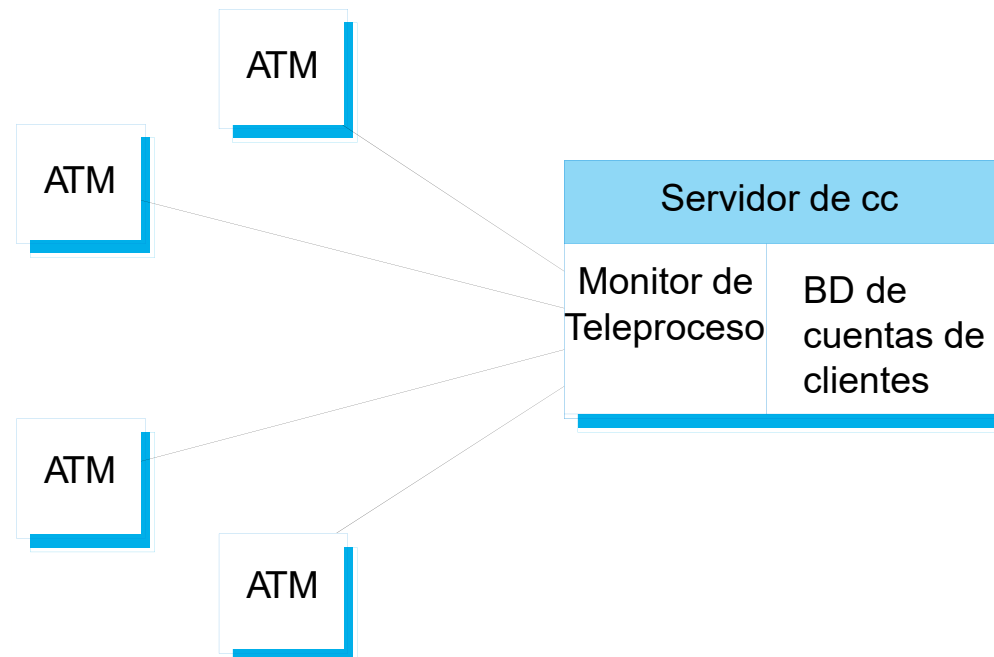
# Dos niveles



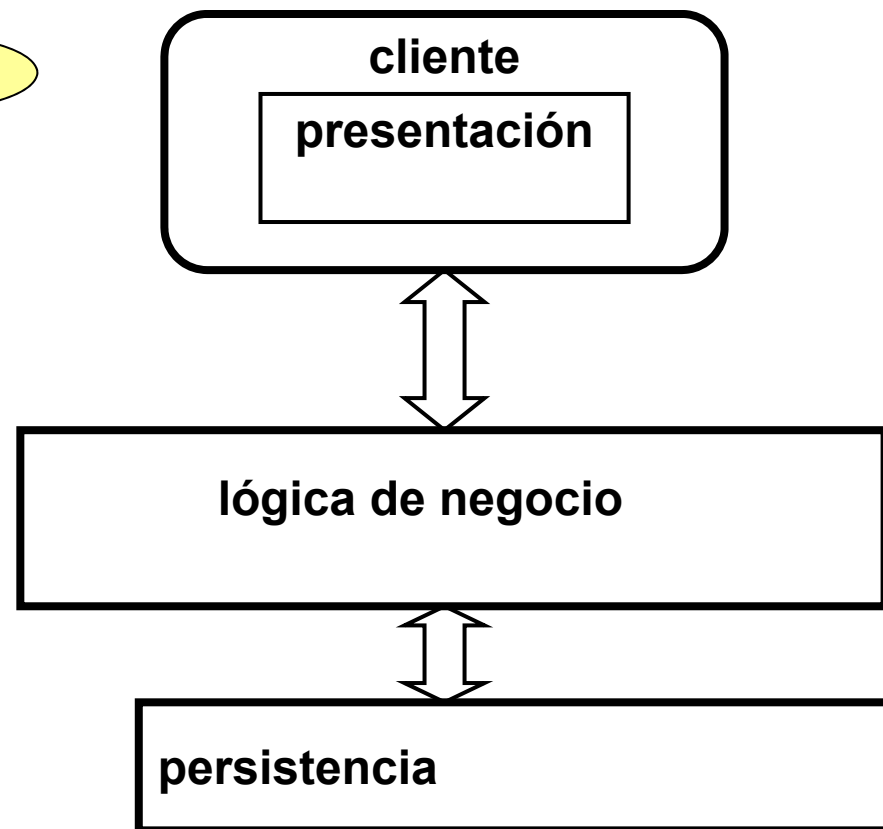
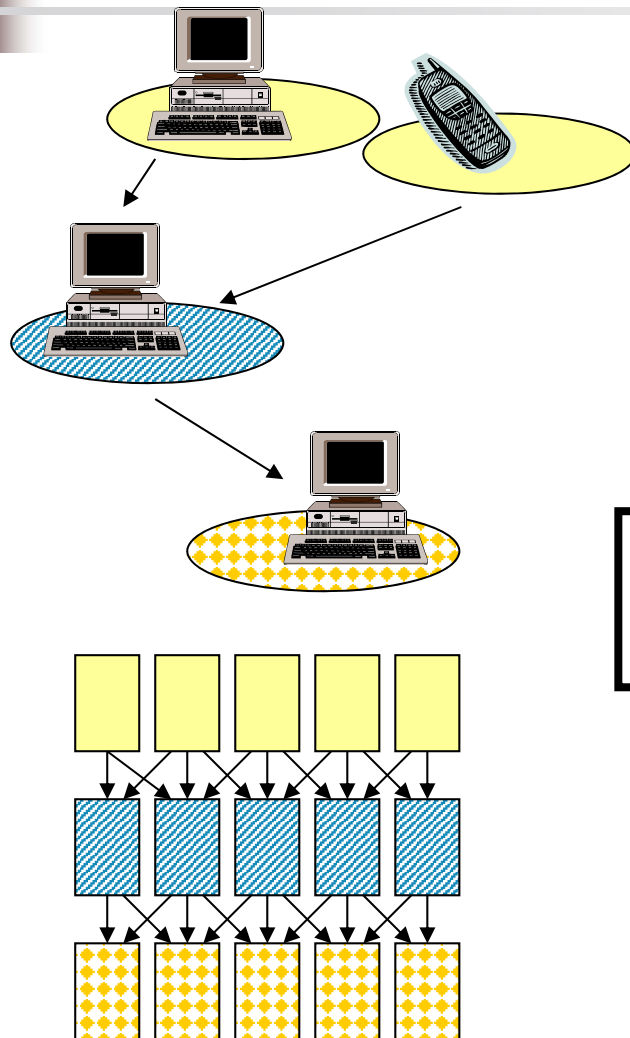


# Arquitectura de dos niveles: utilidad

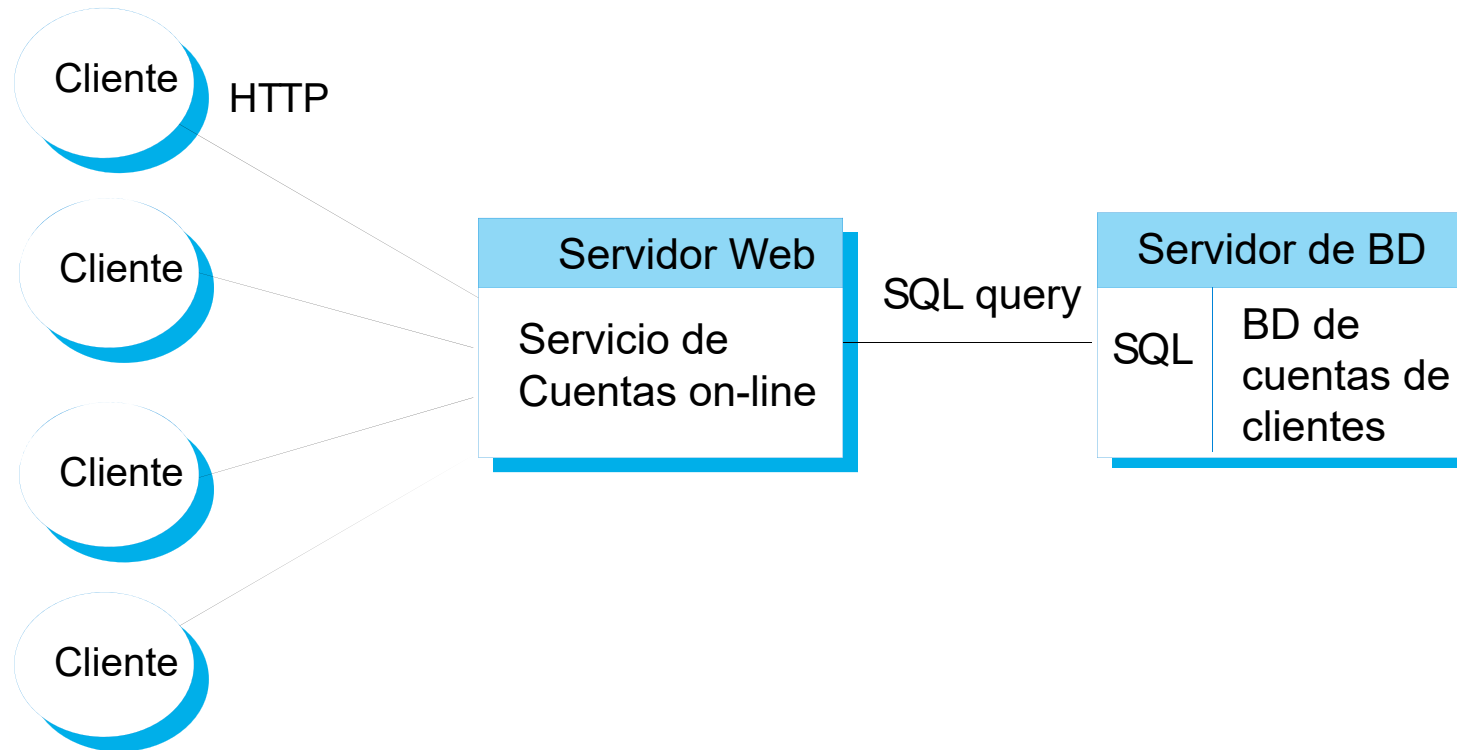
- Normalmente se utiliza:
  - Cuando se requiera poco procesamiento de datos en la organización.
  - Cuando se tiene una base de datos centralizada en un solo servidor.
  - Cuando la base de datos es relativamente estática.
  - Cuando se requiere un mantenimiento mínimo.

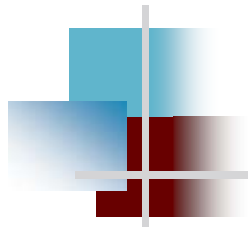


# Tres niveles

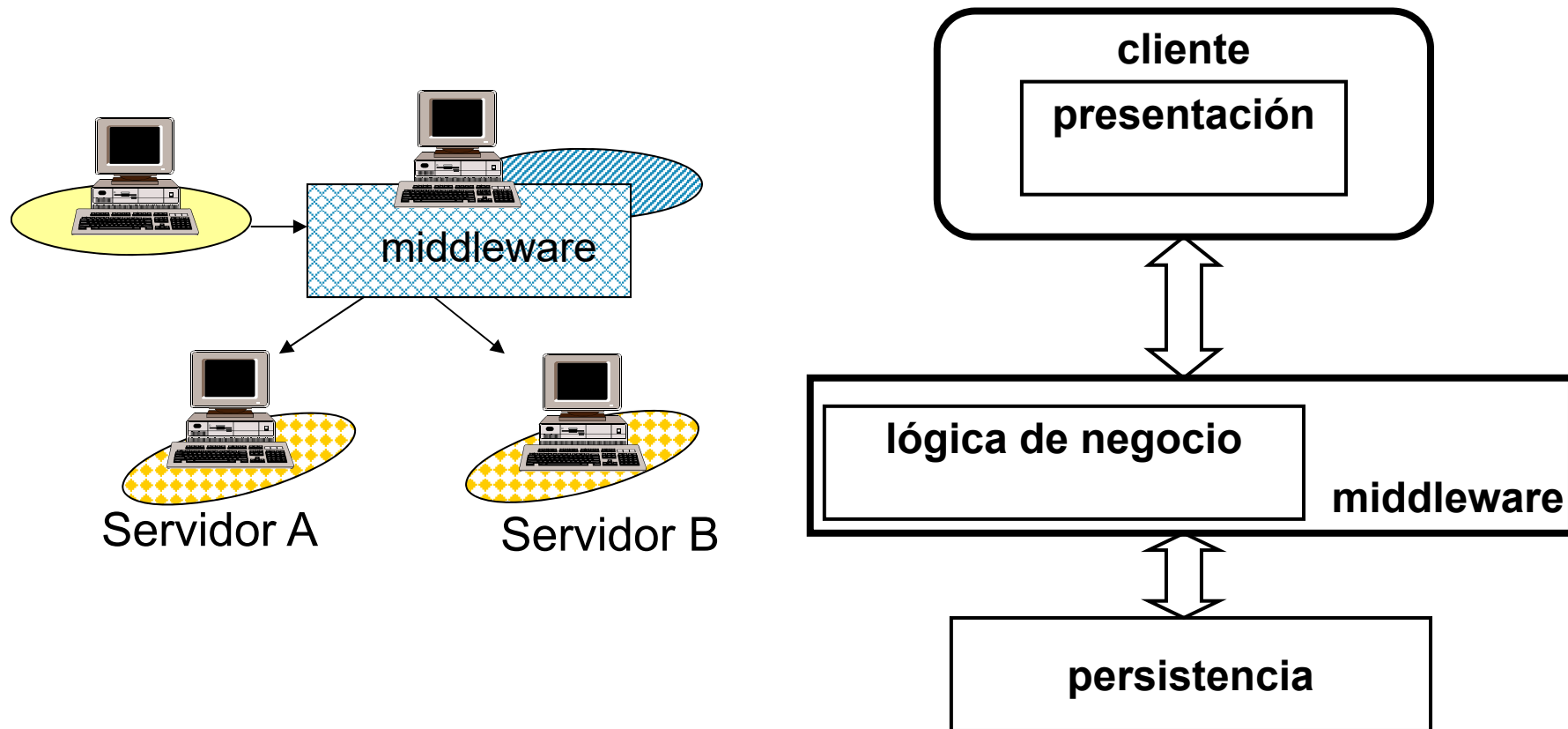


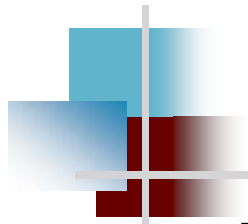
# Arquitectura de Tres Niveles





# Tres niveles: middleware!





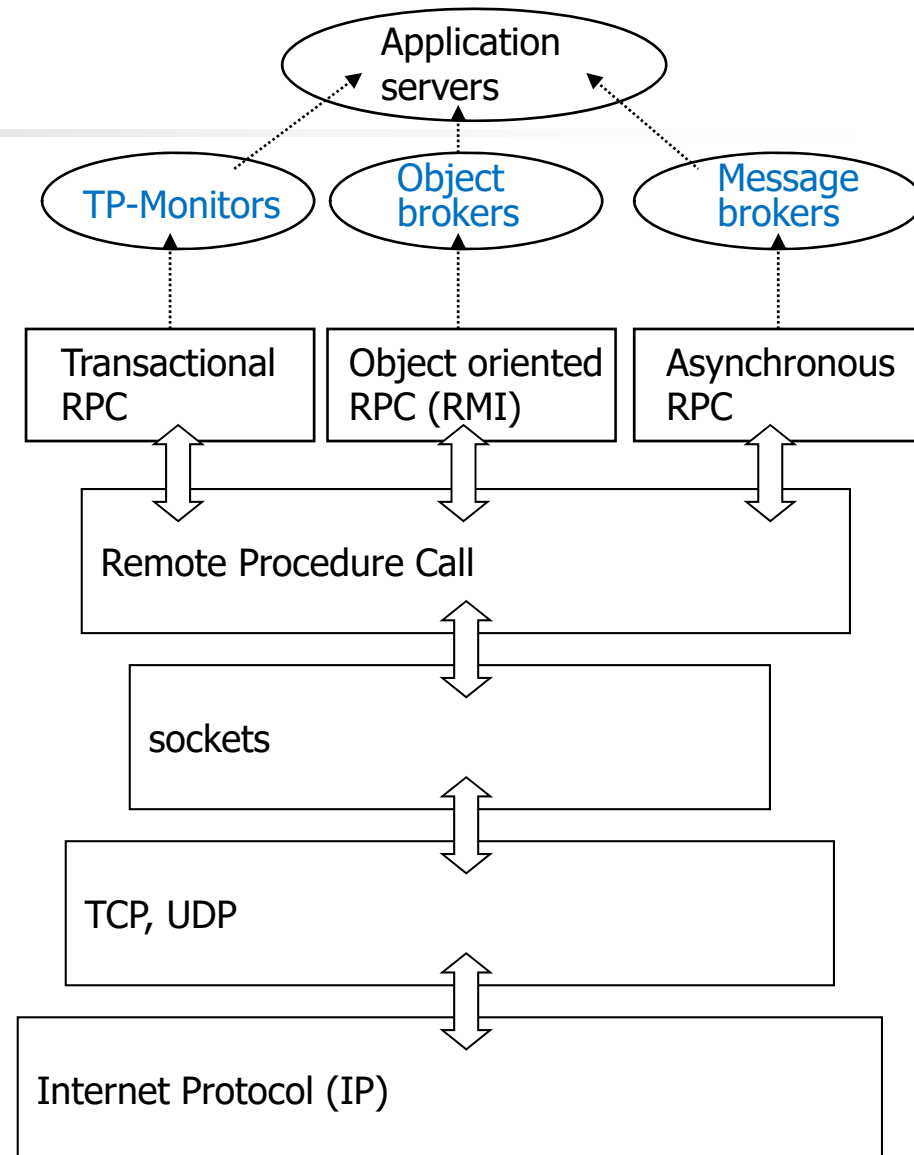
# Middleware

---

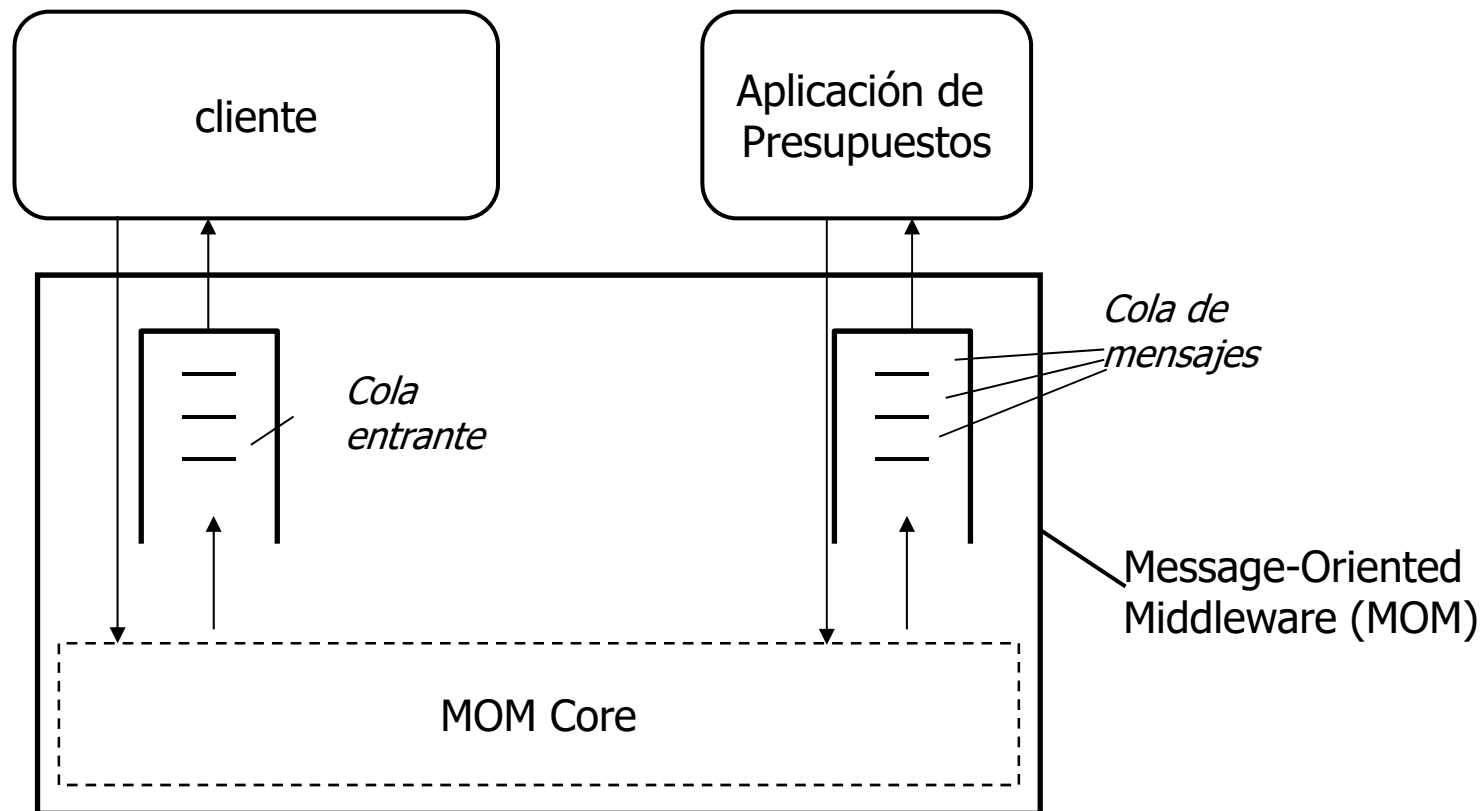
- El software que gestiona y apoya los diferentes componentes de un sistema distribuido.
- Se encuentra en el centro del sistema.
- Software de propósito general que normalmente se compra
- Ejemplo:
  - Petición de una página web desde un navegador en el cliente, el **middleware determina la ubicación** y envía una petición para dicha página.
  - El servidor Web, interpreta la petición y envía la página al middleware, quien **la dirige al navegador** de la máquina cliente que la solicitó.

# Tipos

- Controladores de comunicación
- Convertidores de datos
- Monitores de transacciones
- Gestores de mensajes
- Gestores de Objetos distribuidos

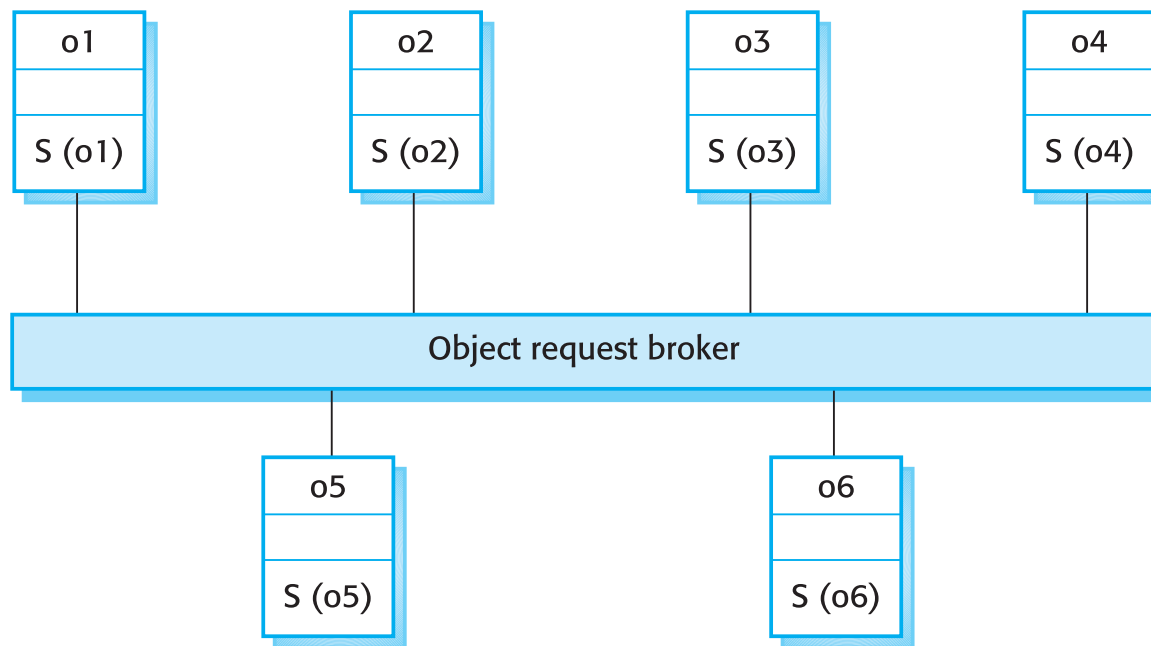


# Middleware orientado a mensajes



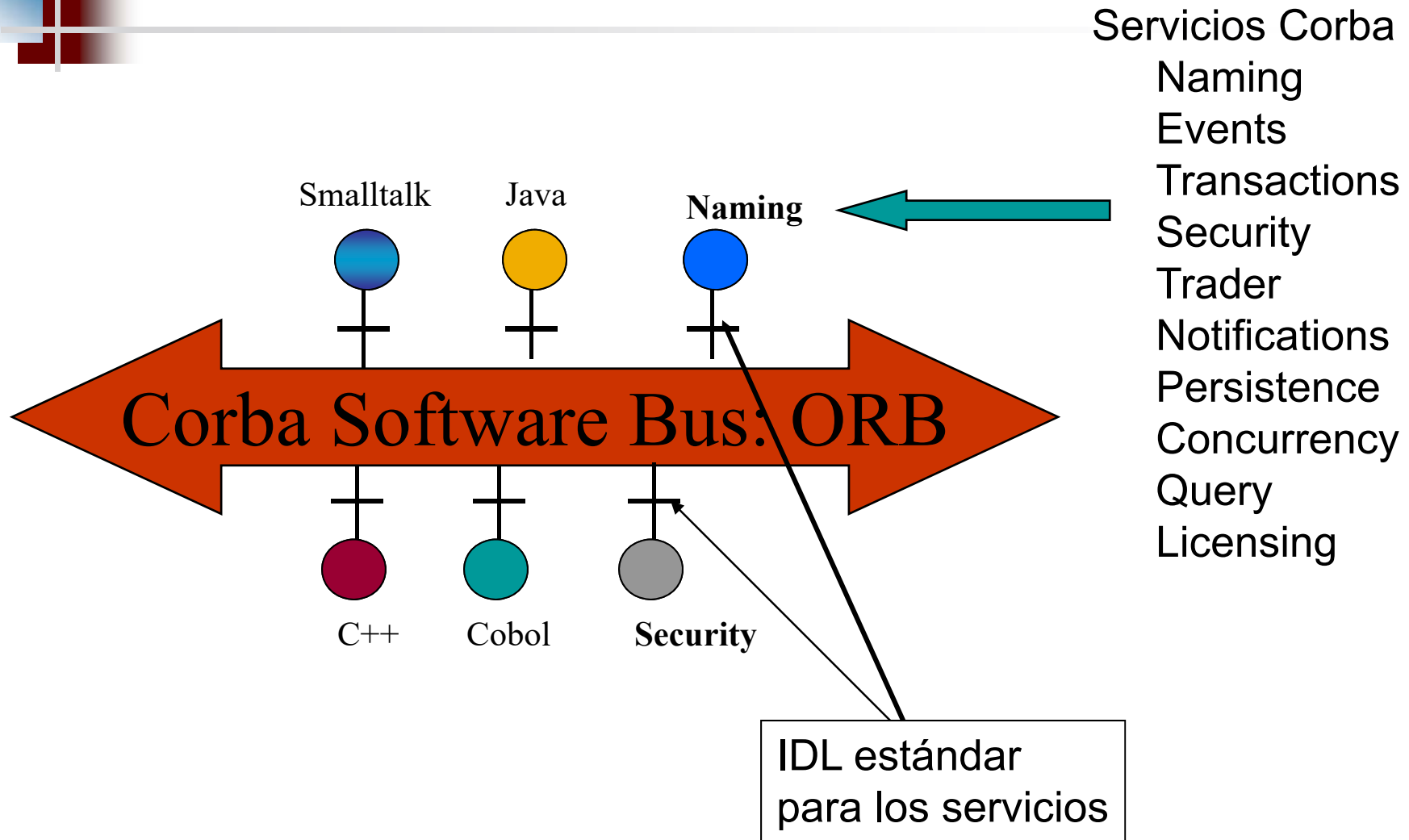
# Arquitectura de objetos distribuidos: CORBA

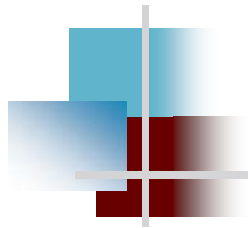
- No hay una distinción entre clientes y servidores.



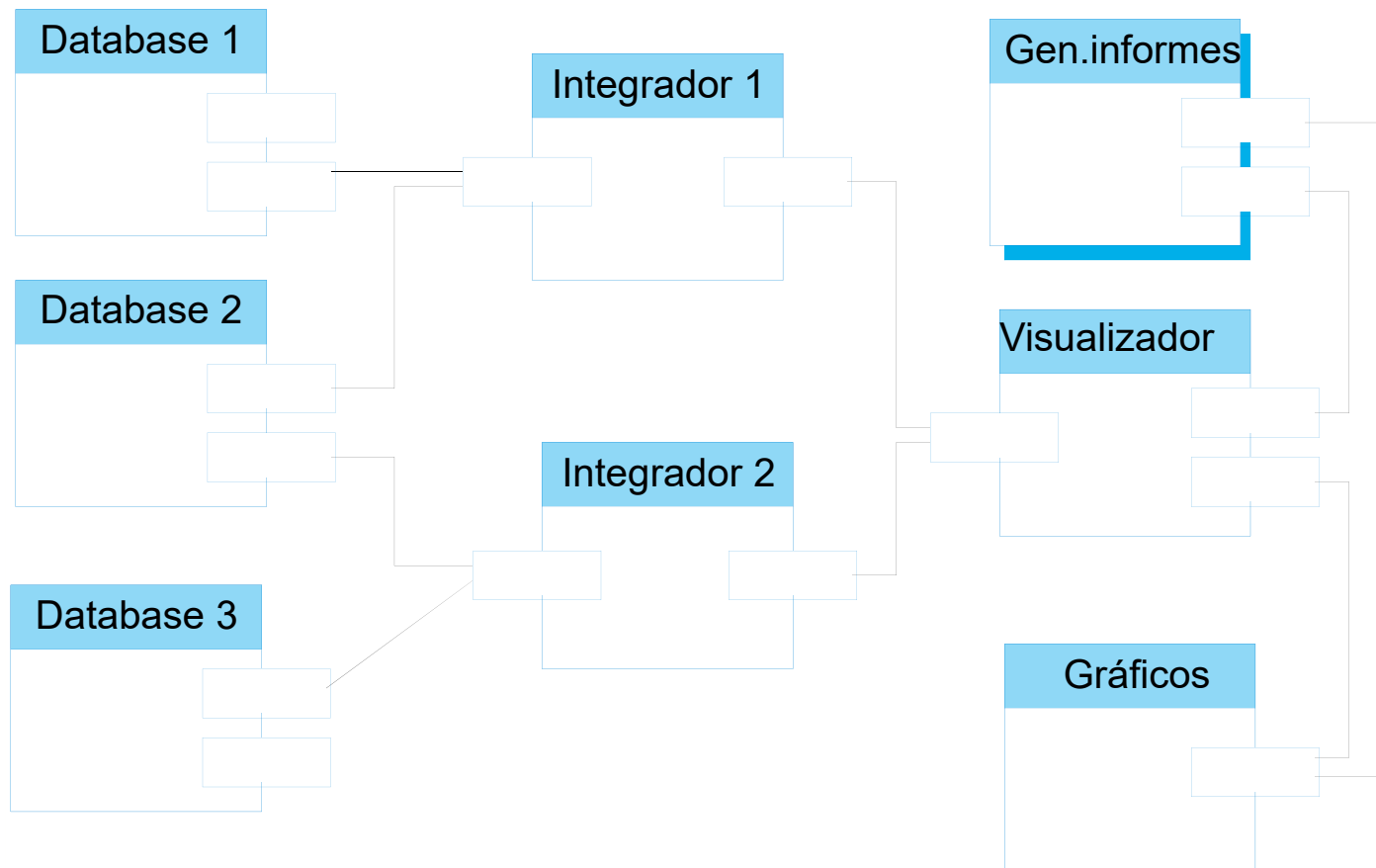


# CORBA: estructura

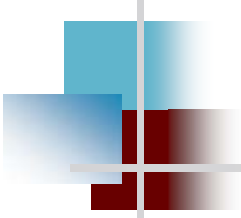




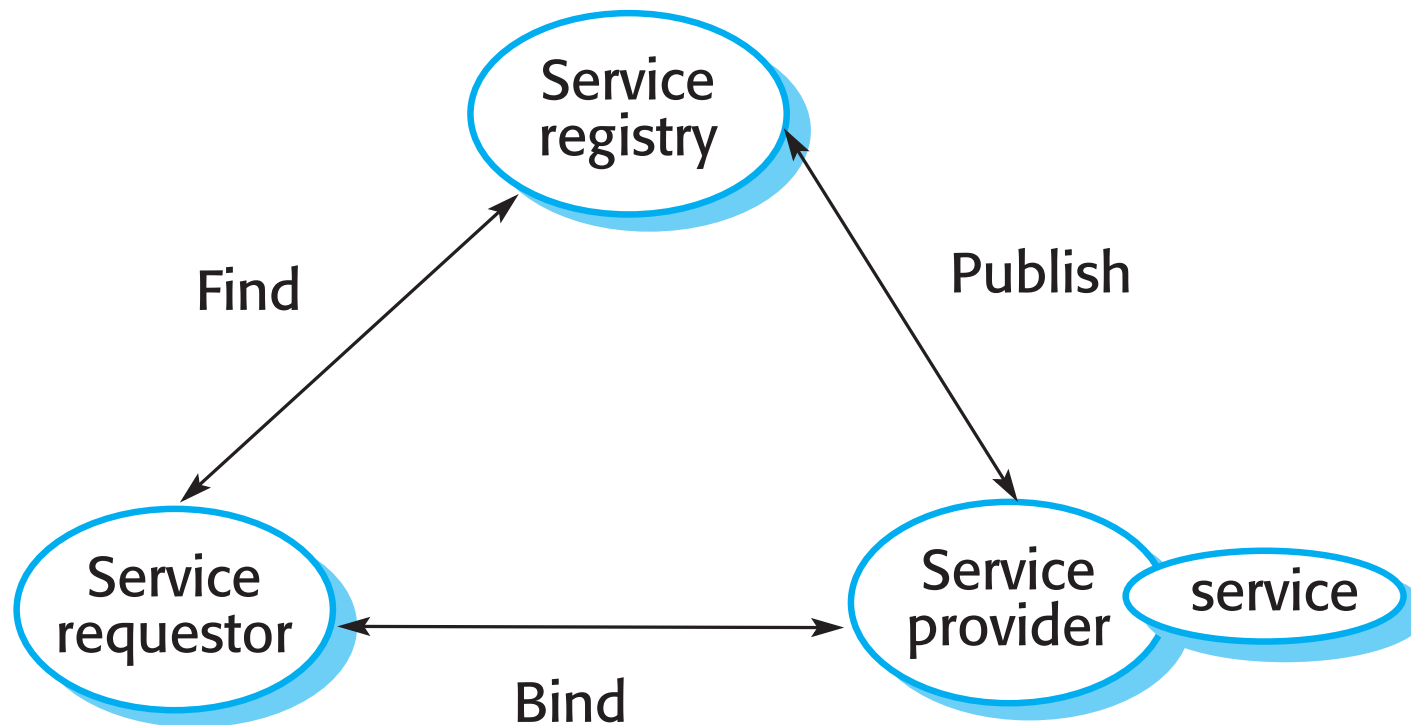
# Sistema de minería de datos

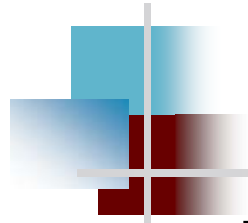


# Arquitecturas orientadas a servicios Web

- 
- Giran en torno al concepto de los servicios prestados externamente (servicios Web)
  - Intervienen distintas organizaciones
  - Un servicio Web es un modo de hacer un componente disponible reutilizable y accesible a través de la Web
  - Enlaza de forma natural con el concepto de “computación en la nube” (Cloud Computing)

# Arquitecturas orientadas a servicios (SOA)



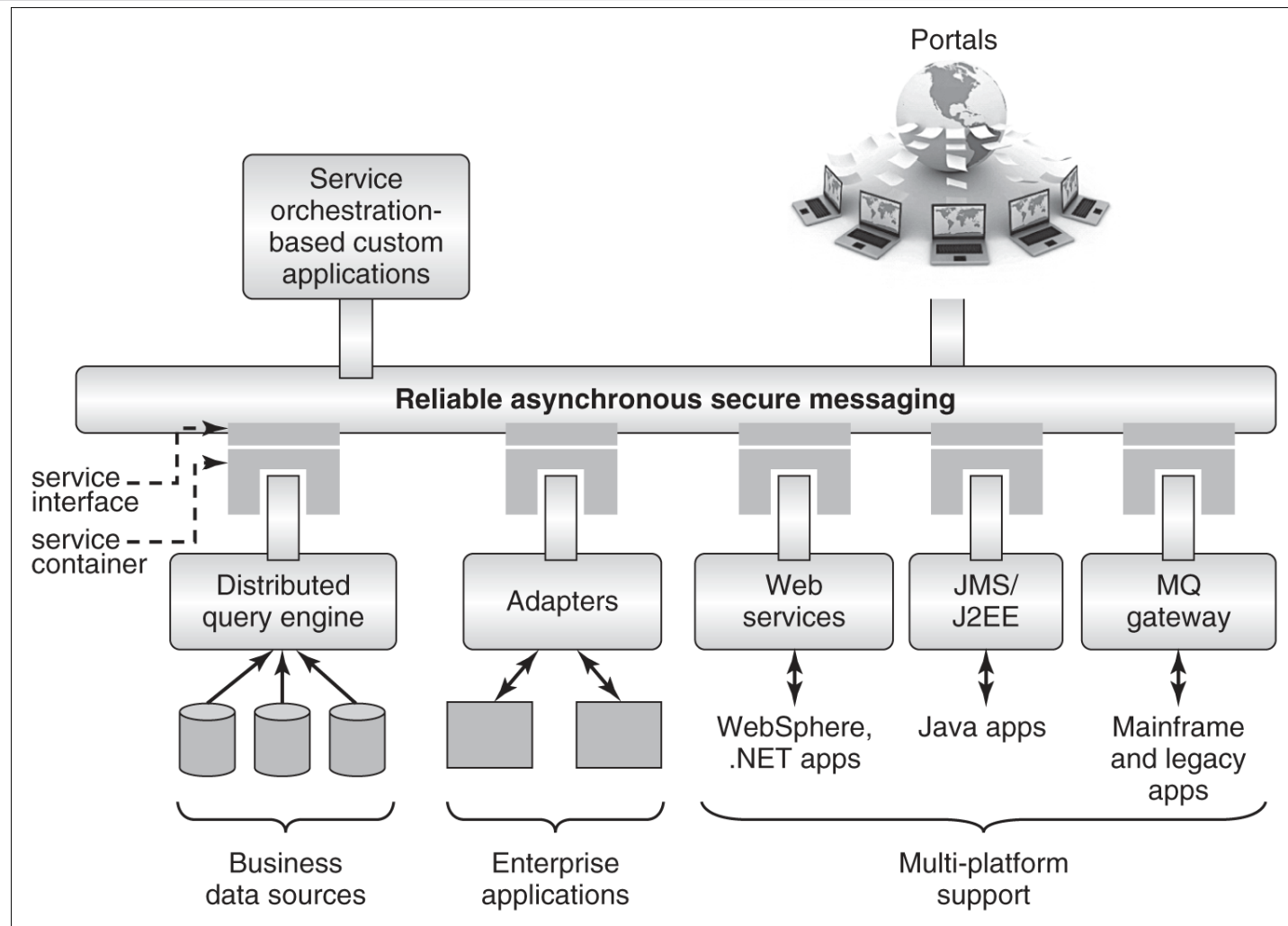


# Bus de servicios

---

- Enterprise service bus (ESB)
- Juega el papel de middleware para asegurar la entrega de mensajes siguiendo el protocolo SOAP
- Productos comerciales de IBM, Oracle, etc.
- Soporte para procesos de negocio

# Interconexión vía ESB





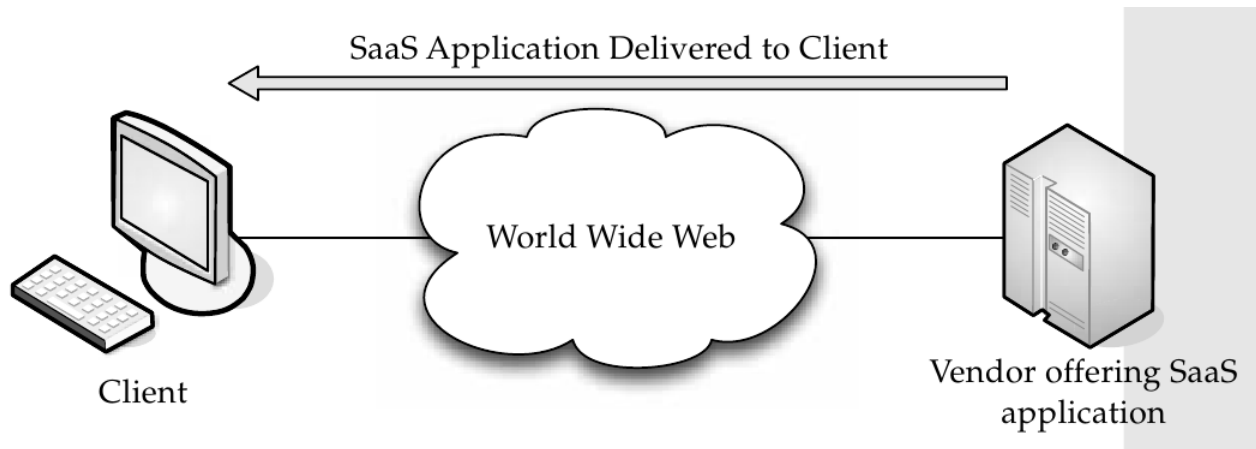
## 2.7. Cloud Computing y SaaS

---

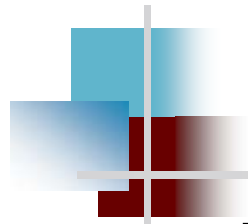
SaaS: Software as a Service

# Cloud Computing y SaaS

- Cloud Computing consiste en usar un software y/o hardware remoto que un proveedor (incluyendo la misma empresa internamente) administra y ofrece como servicio
- SaaS es una de las posibilidades de Cloud Computing







# Cloud Computing

---

- Un paso natural para la industria de TI:
  - del sector secundario de la economía (fabricación)
  - al sector terciario de la economía (servicios)
- Explotación de las economías de escala
- Modelos:
  - Nube pública o privada
    - economía de escala vs aspectos legales
  - Nivel de servicios:
    - IaaS : Infrastructure as a Service (Hardware)
    - PaaS: Platform as a Service (Hardware + Software básico)
    - SaaS: Software as a Service (Aplicaciones completas)

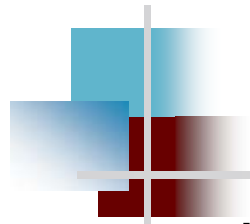


# Economías de escala

---

Clusters: ordenadores simples conectados en red

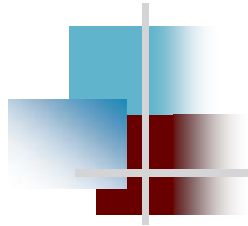
- Más escalable que los servidores convencionales
- Mucho más barato que los servidores convencionales
  - 20 veces más barato que los servidores grandes equivalentes
  - Y normalmente se utiliza un 10% de la capacidad
  - $1000 \text{ ordenadores} * 1 \text{ hora} = 1 \text{ ordenador} * 1000 \text{ horas}$
- Fiabilidad a través de redundancia
- Pocos operadores para miles de servidores
  - Selección cuidadosa de HW/SW idénticos
  - Monitores de máquinas virtuales (simplifican la operación)



# Infrastructure as a Service

---

- Hardware virtualizado formando una red
  - Máquinas, memoria, CPU, discos
- Networking, Balanceadores de carga, acceso a Internet
- Ejemplo IaaS: Amazon EC2 (o ETSII/INFOR)
  
- Se controla el SW y se paga por el HW que se necesita cuando se necesita (ventas de navidad)
  
- Ventajas para el medio ambiente
  - Reduce el consumo de energía, tanto para alimentación como para refrigeración.



# Platform as a Service

---

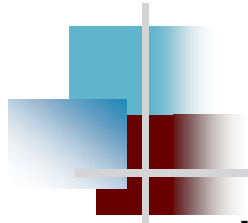
- Añade a IaaS:
  - Gestores Bases de datos o plataformas de desarrollo
  - Ya están instalados
  - Desventaja: menos flexibilidad
  
- Ejemplos: Google App Engine, Microsoft Azure, Heroku
  - Compiladores/Interpretes, Frameworks de desarrollo y Gestores de bases de datos preinstalados



# Software as a Service

---

- Software alojado en un host remoto y accedido a través de Internet (navegador)
  - El precio es pago por uso
  - La infraestructura local es mínima.
  - Siempre se tiene el software actualizado.
- Para empresas
  - Software de gestión que se paga mensualmente: Salesforce, MS Office
- Para el cliente particular
  - Juegos on-line, Facebook, Google Apps, etc. se paga con publicidad.



# Software as a Service

---

- La virtualización ha facilitado el crecimiento del SaaS.
  - Las pequeñas empresas de desarrollo lo tienen más fácil gracias a sistemas como Amazon EC2, Google (Android), Adobe (Flash) o Microsoft (Azure .NET, SQL)
- La modularidad del SaaS hace que sea también un modelo de servicios.
  - SOA (Service-Oriented Architecture) es la arquitectura en la que se apoya SaaS
- Ejemplos
  - Software horizontal de comunicación: e-mail.
  - Software vertical con acceso web o móvil: ventas