El Software...

…vuela los aviones

…acelera y frena nuestros coches

…transfiere nuestro dinero

…cuida nuestra salud

…impacta todos los aspectos de nuestras vidas!!!

Sector de Comunicaciones

(1980) Arpanet

- Mutación de bits en memoria
- Código de detección de error usado en transmisión pero no en almacenamiento
- Algoritmo garbage-collection no borra mensajes duplicados
- Caída TODA la red 4 horas

(1990) AT&T

- Una mejora del software de 114 switches trajo el siguiente efecto:
  - Se cae un switch, avisa a sus vecinos y se autoresetea
  - Rearranca y avisa a sus vecinos
  - Debido a un error en el software, los vecinos entran en un ciclo similar
  - El fallo se reproduce una y otra vez…
  - 5 Millones de llamadas bloqueadas. 9 horas de colapso

Industria Aeroespacial

(1981) Columbia

- Para aumentar la fiabilidad del sistema, se utilizaron 5 ordenadores (dos para el control normal, otros dos por si hay desacuerdo entre los dos primeros, un quinto por si hay fallos en los otros cuatro)
- El lanzamiento se retrasa dos días debido a una desincronización del quinto ordenador

(1996) Ariane 5

- Se produjo una excepción no manejada por ADA al no poder convertir un valor de 64 bits a 16 bits
- La subrutina había sido reutilizada
- En su lanzamiento, tras alcanzar 3700m. se desvió, se rompió y estalló

Defensa

(Guerra del Golfo) Misiles Patriot

- Su precisión debía ser del 95%. Se quedó en el 13%.
- Funcionando 100 horas se acumuló un error en el reloj (en pruebas sólo se había tenido 14 horas)
- Dos versiones del número 0.1 (una de 48 bits y otra de 24 bits). No son idénticas.
- No hizo blanco a un scud que mató a 28 soldados americanos

(1988) Misil Aegis Vincennes

- Un F14 se encontraba en el punto de mira. La pantalla no indicaba la altura de los objetos.
- Un objeto que subía se interpretó que bajaba.
- Se disparó el misil a un Airbus iraní (290 muertos)

Aviación

(1988) Airbus A320

- Fallos de altímetro, repentinos cambios en aceleración, problemas de conducción en pista, lentitud del motor en repuesta a mandatos de pilotos, etc.
- En una exhibición murieron 3 personas y hubo 133 heridos

Medicina

THERAC-25

- La interfaz gráfica permitía proporcionar dosis de radiaciones mortales a los pacientes
- 6 accidentes (tres muertes)

(1993) London Ambulance Service

- El nuevo sistema fue incapaz de asignar con rapidez ambulancias a conductores y a emergencias.
- Retrasos de 11 horas. 20 muertes por el fallo del sistema

Errores en el Software

1 error cada 10 LOC

Un programa de 1.000.000 LOC tiene 100.000 errores potenciales:

- Los programadores encuentran y corrigen el 95% de los errores
- El programa final tiene 5.000 errores
- Cualquier error puede causar problemas

Definición Informal

Como hacer **buen** software **bien**

Algunas preguntas que intenta responder

- ¿Cómo pensar a la hora de enfrentarte a un nuevo problema?
- ¿En qué lenguaje puedo expresar mis ideas de forma que las pueda discutir con otras personas?
- ¿Cómo diseñar sistemas que tienen que responder en milésimas de segundo o con una precisión de nanómetros?
- ¿Cómo puedo construir un brazo médico que haga operaciones quirúrgicas de forma precisa y tenga la seguridad de que no se va a cargar al paciente?

Fuente: http://slnc.me/algunas-cosas-que-aprendes-estudiando-ingenieria-informatica/

Implementación

- Implementación en el PUD
- Arquitectura de tres capas
- Capa de presentación
- Capa de gestión de datos
- SI OO distribuidos

Iteración en PUD

- Planificación de la Iteración
  - Captura de requisitos:
    - Modelo de casos de uso, Modelo de Dominio, ...
  - Análisis:
    - Diagrama de Secuencia del Sistema, Contratos, Modelo Conceptual...
  - Diseño:
    - Diagramas de interacción, Diagrama de Clases
  - Implementación:
    - Codificación (Clases y métodos)
  - Pruebas:
    - Verificación de la implementación
- Evaluación de la iteración

Fases y entregas del Proceso Unificado de Desarrollo

- Captura de requisitos: qué SI debemos construir?
  - Modelo de casos de uso, Modelo de Dominio, ...
- Análisis: qué debe hacer el SI?
  - Diagramas de secuencia del sistema, Contratos, ...

- Diseño: cómo lo debe hacer el SI?
  - Diagramas de interacción, Diagrama de Clases, ...
- Codificación:
  - Código Fuente (clases y métodos)
- Pruebas:
  - Especificación de las pruebas de funcionamiento
- Mantenimiento:
  - Documentación y revisión de todo lo anterior

Dependiente de la tecnología

Construcción incremental e iterativa del SI

- Modelo dinámico del sistema (comportamiento):
  - Captura de requerimientos: Modelo de Casos de Uso
  - Análisis: Diagramas de secuencia del sistema, contratos
  - Diseño: Diagramas de interacción
- Modelo estático del sistema (propiedades):
  - Captura de requerimientos: Modelo de Dominio
  - Análisis: Modelo Conceptual
  - Diseño: Diagrama de clases
- Implementación: codificación (clases y métodos)

Implementación

- Codificar el diseño del SI.

- Situación de partida (diseño del SI)

  - Resultado del diseño (cómo lo debe hacer el SI?)
    - Diseño de los datos (Modelo de datos)
    - Diseño de procesos (Modelo de comportamiento)
    - Diseño de la interacción con el usuario (Modelo de la interfaz)

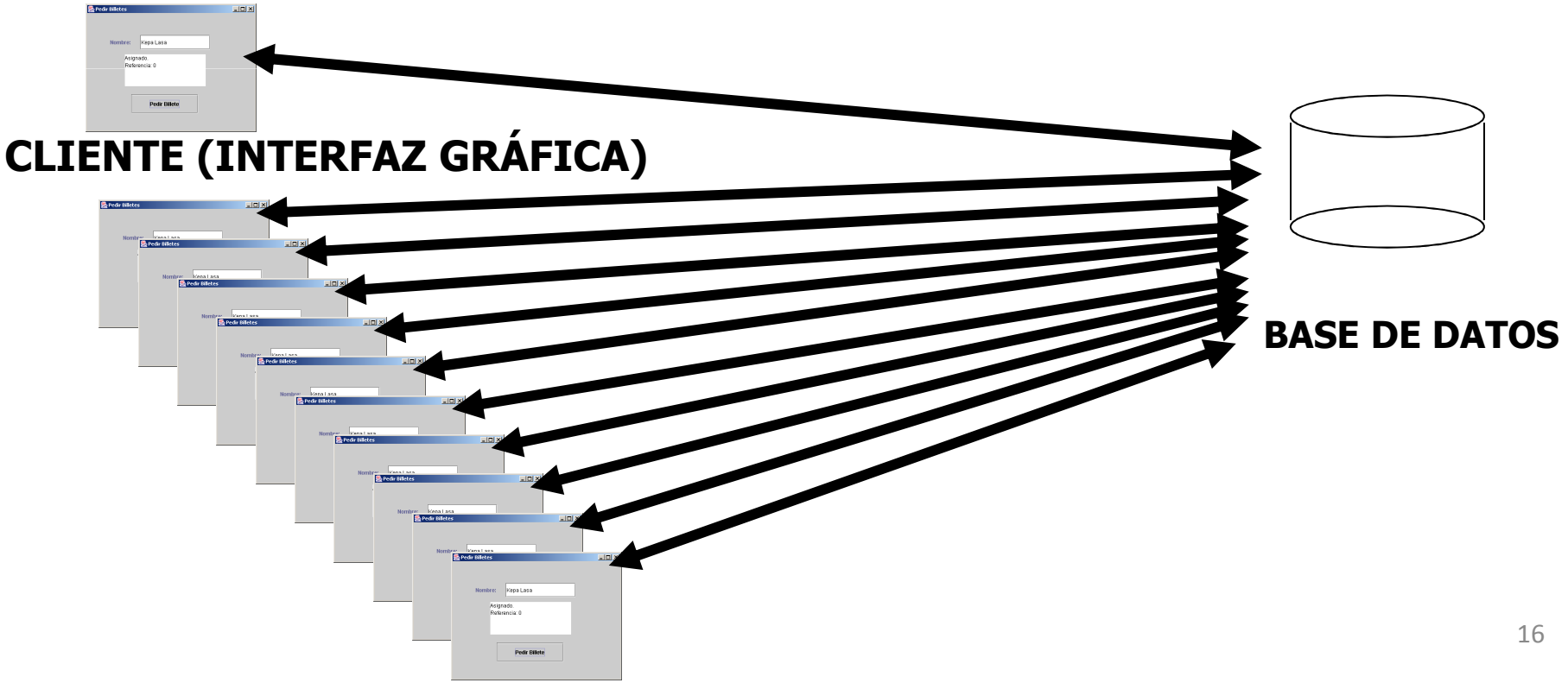
  - Recursos tecnológicos (hardware y software disponible)

Implementación

- Situación final (solución)
  - Resultado de la implementación (cómo lo hace el SI?)
    - Estructura interna del SI (Arquitectura del SI)
    - Implementación de los datos: clases, BDs
    - Implementación de los programas: métodos
    - Implementación de la Interfaz: modelo de casos de uso reales
- Proceso de implementación
  - Codificación
  - Pruebas

Arquitectura de un SI en tres capas

- Hay aplicaciones que deben ejecutar operaciones de manera CONCURRENTE, SEGURA y EFICIENTE



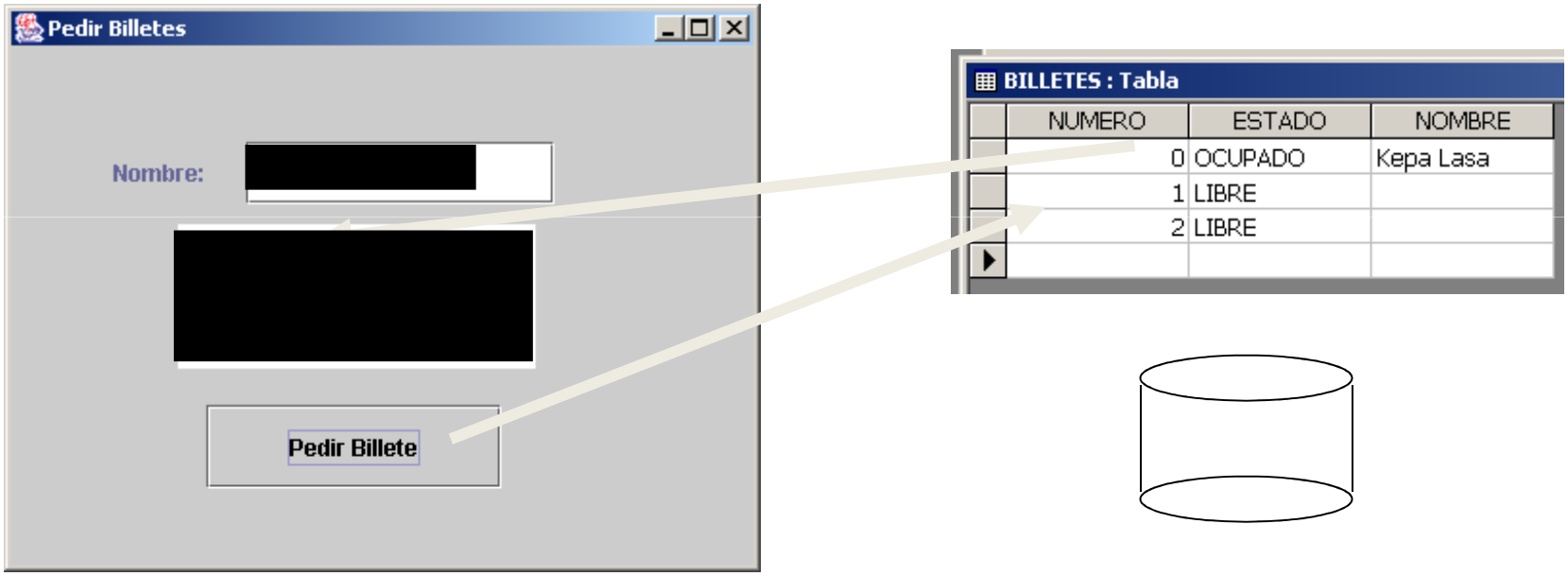
Arquitectura de un SI en tres capas

Ejemplos:

- Terminales donde se pueden comprar entradas para espectáculos
- Banca electrónica

- SOLUCIÓN: usar una
  - **Arquitectura con despliegue de componentes software en el lado del servidor**
  - Componente: código que implementa un conjunto conocido de interfaces

Ejemplo: comprar billetes para espectáculos



BASE DE DATOS

Se puede implementar en una sola clase (véase PedirBilleteNO3NIVELES) en nodos cliente, dejando la base de datos en un nodo servidor

```
public class PedirBilleteNO3NIVELES extends JFrame {
// Nota: NO ESTÁ COMPLETA !!
  JLabel jLabel1 = new JLabel(“Nombre:”);
  JButton jButton1 = new JButton(“Pedir Billete”);
public PedirBilleteNO3NIVELES() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conexion=DriverManager.getConnection("jdbc:odbc:Billetes"); }
void jButton1_actionPerformed(ActionEvent e) {
ResultSet rs = sentencia.executeQuery("SELECT NUMERO FROM"+
                    " BILLETES WHERE ESTADO=\'LIBRE\'");
    if (rs.next()) {
    int act = sentencia.executeUpdate("UPDATE BILLETES"+
     " SET ESTADO='OCUPADO', NOMBRE = "+jTextField1.getText()+
     " WHERE NUMERO="+rs.getString("NUMERO")+
     " AND ESTADO='LIBRE'");
    if (act>0) jTextArea1.append("Asignado. \nReferencia: "+n+"\n");
    else jTextArea1.append("Error al asignar billete"); }}

public static void main (String []arg) {
   PedirBilleteNO3NIVELES b = new PedirBilleteNO3NIVELES();
   b.setVisible(true);}}
```

```
public class PedirBilleteNO3NIVELES extends JFrame {
// Nota: NO ESTÁ COMPLETA !!
  JLabel jLabel1 = new JLabel("Nombre:");
  JButton jButton1 = new JButton("Pedir Billete");
public PedirBilleteNO3NIVELES() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conexion=DriverManager.getConnection("jdbc:odbc:Billetes"); }
void jButton1_actionPerformed(ActionEvent e) {
ResultSet rs = sentencia.executeQuery("SELECT NUMERO FROM"+
                    " BILLETES WHERE ESTADO=\'LIBRE\'");
    if (rs.next()) {
    int act = sentencia.executeUpdate("UPDATE BILLETES"+
     " SET ESTADO='OCUPADO', NOMBRE = "+jTextField1.getText()+
     " WHERE NUMERO="+rs.getString("NUMERO")+
     " AND ESTADO='LIBRE'");
    if (act>0) jTextArea1.append("Asignado. \nReferencia: "+n+"\n");
    else jTextArea1.append("Error al asignar billete"); }}

public static void main (String []arg) {
   PedirBilleteNO3NIVELES b = new PedirBilleteNO3NIVELES();
   b.setVisible(true);}}
```

PRESENTACIÓN

```
public class PedirBilleteNO3NIVELES extends JFrame {
// Nota: NO ESTÁ COMPLETA !!
  JLabel jLabel1 = new JLabel("Nombre:");
  JButton jButton1 = new JButton("Pedir Billete");
public PedirBilleteNO3NIVELES() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conexion=DriverManager.getConnection("jdbc:odbc:Billetes"); }
void jButton1_actionPerformed(ActionEvent e) {
ResultSet rs = sentencia.executeQuery("SELECT NUMERO FROM"+
               " BILLETES WHERE ESTADO=\'LIBRE\'");
   if (rs.next()) {
   int act = sentencia.executeUpdate("UPDATE BILLETES"+
    " SET ESTADO='OCUPADO', NOMBRE = "+jTextField1.getText()+
    " WHERE NUMERO="+rs.getString("NUMERO")+
    " AND ESTADO='LIBRE'");
    if (act>0) jTextArea1.append("Asignado. \nReferencia: "+n+"\n");
    else jTextArea1.append("Error al asignar billete"); }}

public static void main (String []arg) {
   PedirBilleteNO3NIVELES b = new PedirBilleteNO3NIVELES();
   b.setVisible(true);}}
```

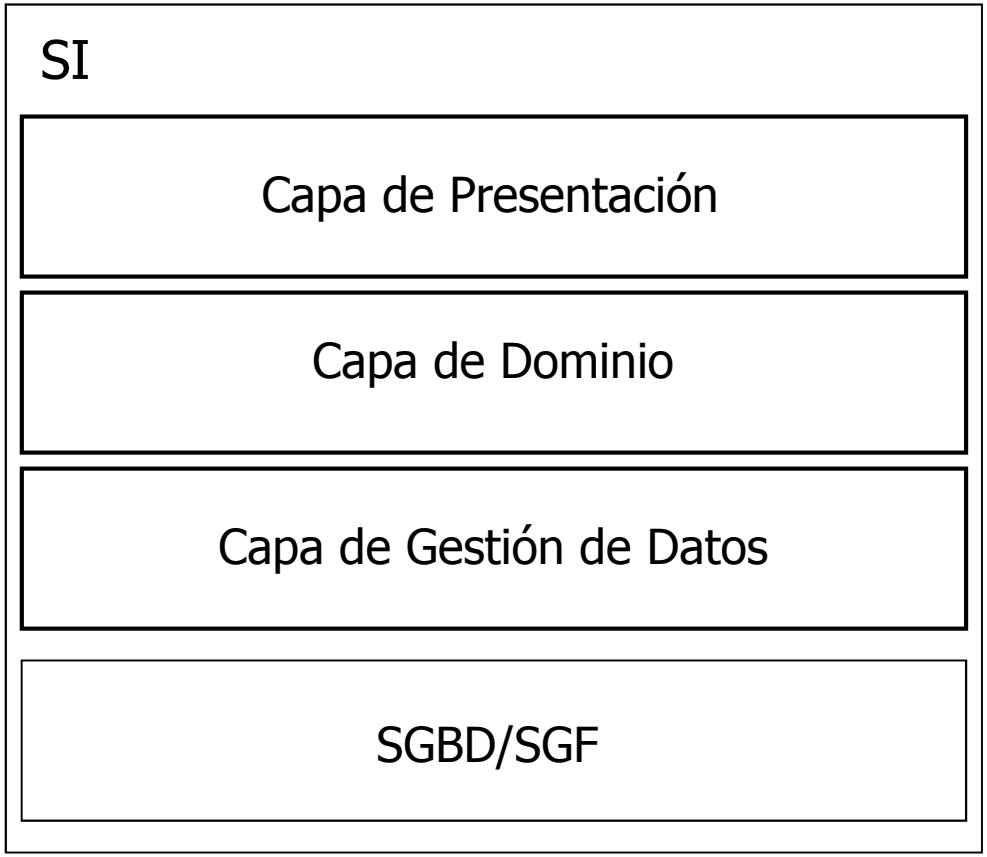
ACCESO A DATOS

```
public class PedirBilleteNO3NIVELES extends JFrame {
// Nota: NO ESTÁ COMPLETA !!
  JLabel jLabel1 = new JLabel("Nombre:");
  JButton jButton1 = new JButton("Pedir Billete");
public PedirBilleteNO3NIVELES() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conexion=DriverManager.getConnection("jdbc:odbc:Billetes"); }
void jButton1_actionPerformed(ActionEvent e) {
ResultSet rs = sentencia.executeQuery("SELECT NUMERO FROM"+
                    " BILLETES WHERE ESTADO=\'LIBRE\'");
    if (rs.next()) {
    int act = sentencia.executeUpdate("UPDATE BILLETES"+
     " SET ESTADO='OCUPADO', NOMBRE = "+jTextField1.getText()+
     " WHERE NUMERO="+rs.getString("NUMERO")+
     " AND ESTADO='LIBRE'");
    if (act>0) jTextArea1.append("Asignado. \nReferencia: "+n+"\n");
    else jTextArea1.append("Error al asignar billete"); }}

public static void main (String []arg) {
   PedirBilleteNO3NIVELES b = new PedirBilleteNO3NIVELES();
   b.setVisible(true);}}
```

LÓGICA DEL NEGOCIO

Arquitectura de un SI en tres capas



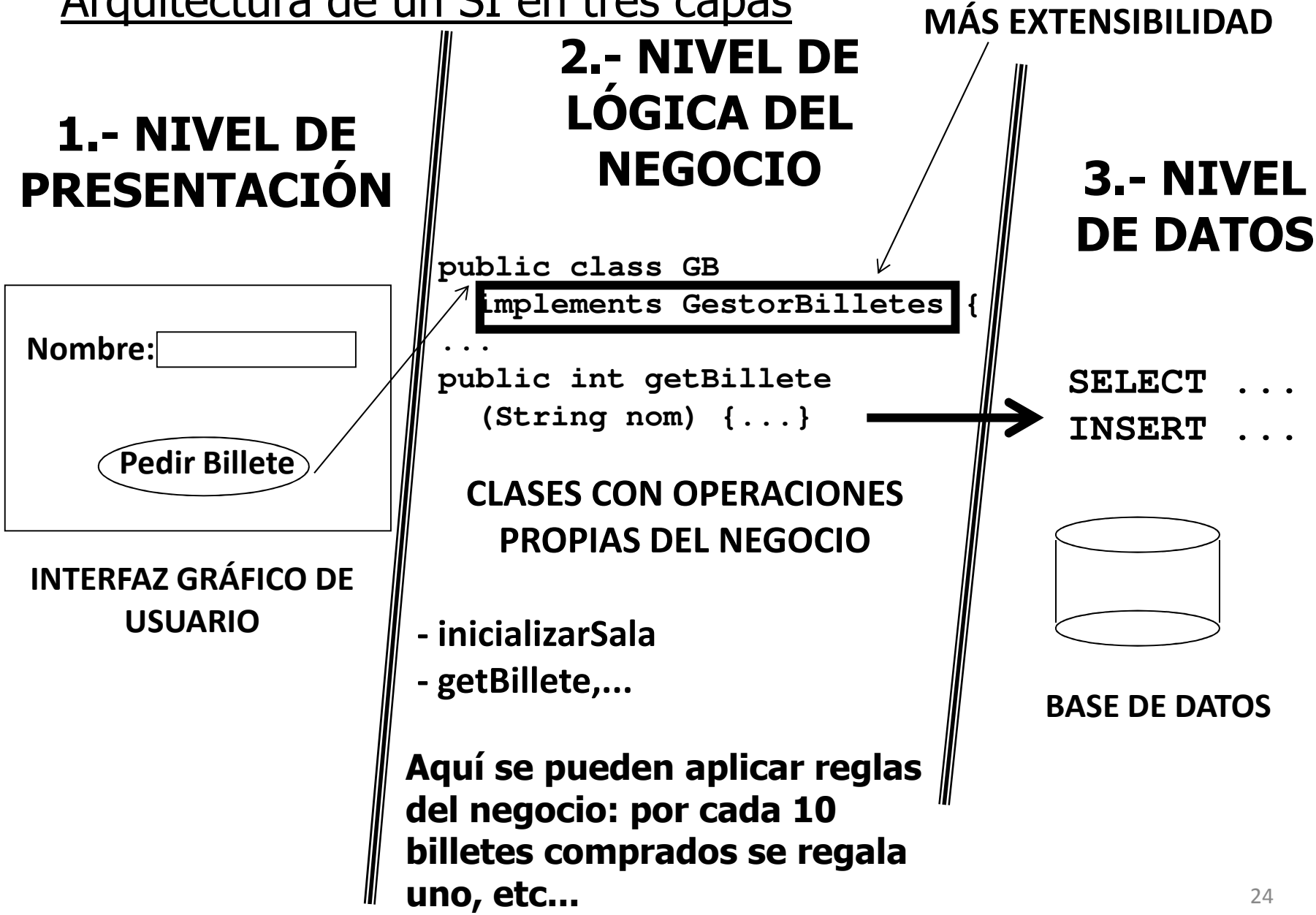
Responsable de la interacción con el usuario

Responsable de las funcionalidades del SI

Responsable de la interacción con el SGBD

- Arquitectura cambiable, reusable, portable

Arquitectura de un SI en tres capas



Arquitectura de un SI en tres capas

- **Ventaja:** Se puede cambiar cada uno de los niveles minimizando los cambios en los otros niveles

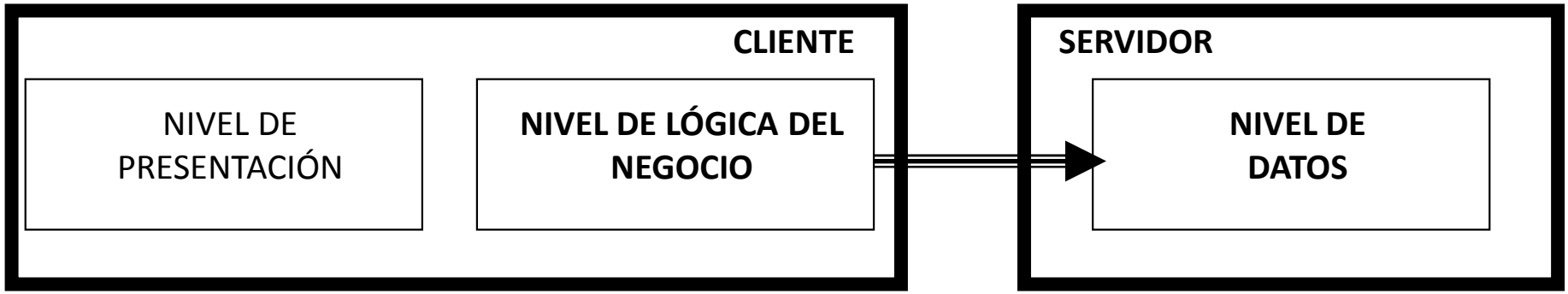
UNA ARQUITECTURA LÓGICA DEL SOFTWARE EN VARIOS NIVELES FAVORECE LA EXTENSIBILIDAD Y REUTILIZACIÓN DEL SOFTWARE

Arquitectura **física** de un SI en tres capas

- La separación FÍSICA de los niveles admite distintas posibilidades:
- Arquitectura en 2 niveles
  - Nivel de presentación en NODO CLIENTE
  - Nivel de datos en NODO SERVIDOR (de BD)
  - ¿Y el nivel de lógica del negocio?
    - En el CLIENTE: junto con el nivel de presentación
    - PARTE podría juntarse con el nivel de datos
- Arquitectura en 3 niveles (o más)
  - Cada nivel, al menos, en un nodo distinto

Arquitectura **física** en dos niveles: cliente gordo/servidor flaco

- El nivel de presentación y el de la lógica del negocio se unen en un nodo. En el otro queda el nivel de datos.



- **Comunicación entre Cliente y Servidor en SQL**
- **Se necesitan APIs como por ejemplo JDBC y/o ODBC**
- **Deben instalarse DRIVERS de la BD en todos los clientes**

CLIENTE

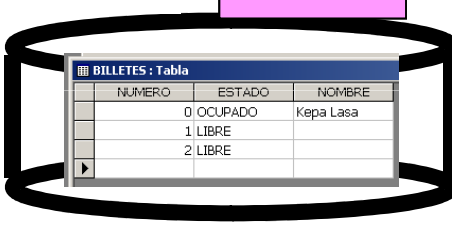
Presentación

```
public class PedirBillete2NivCliGordo extends JFrame {
GestorBilletes2NivCliGordo gestorBilletes;
void jButton1_actionPerformed(ActionEvent e) {
   int res = gestorBilletes.getBillete(jTextField1.getText()).getNum();
    if (res<0) jTextArea1.append("Error al asignar billete");
    else jTextArea1.append("Asignado. \nReferencia: "+res+"\n");} }
```

Lógica del Negocio

```
public class GestorBilletesBD
      implements GestorBilletes2NivCliGordo
{ public GestorBilletesBD() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conexion=DriverManager.getConnection("jdbc:odbc:Billetes");}
   public Billete getBillete(String nom)
    {ResultSet rs = sent.executeQuery("SELECT NUMERO…");
      int act = sent.executeUpdate("UPDATE BILLETES …");
      if (act>0) return new Billete(n,nom); // Núm. billete asignado
     else return new Billete(-1,"");; } // No había ninguno libre}}}
```

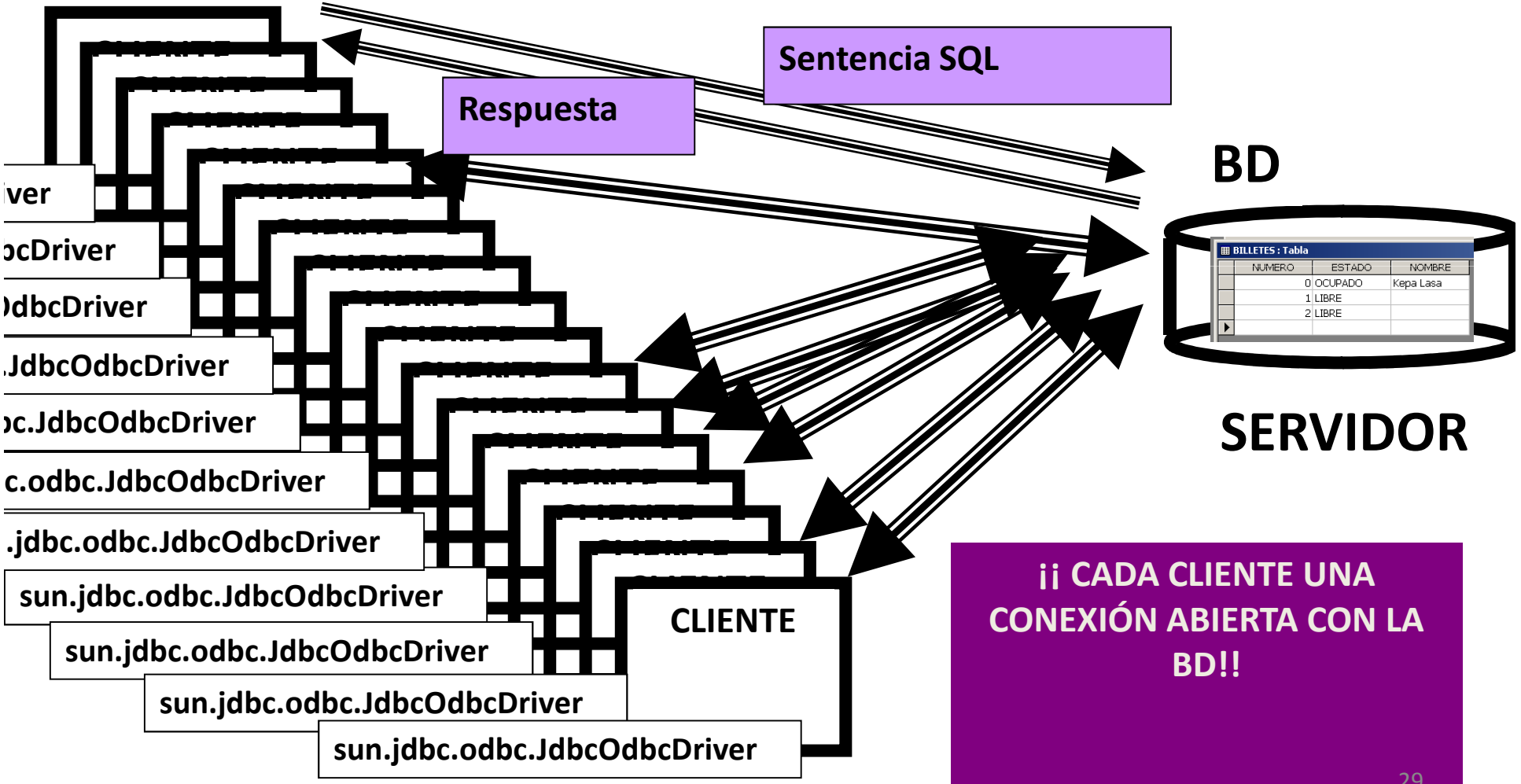
BD **Datos**



SERVIDOR

DEFINIR FUENTE DATOS ODBC “Billetes”

E INSTALAR LA CLASE sun.jdbc.odbc.JdbcOdbcDriver



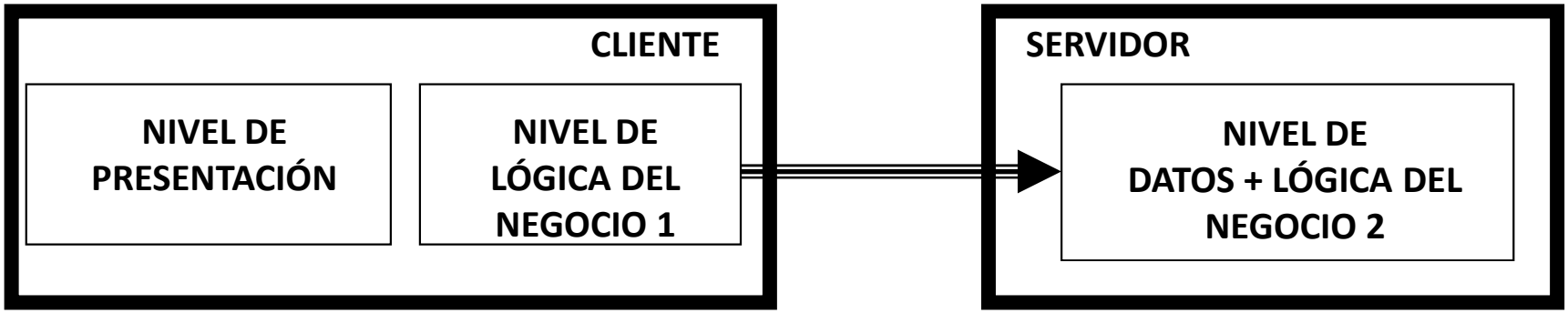
¡¡ CADA CLIENTE UNA CONEXIÓN ABIERTA CON LA BD!!

Arquitectura **física** en dos niveles: cliente gordo/servidor flaco

- El despliegue de la aplicación es alto: instalar drivers y configurar todos los clientes
- Cambiar de SGBD requiere reinstalar todos los clientes
- Cambiar el esquema de la BD puede afectar a los clientes
- Cambiar la lógica del negocio implica recompilar y desplegar en todos los clientes
- Costos de conexión con la BD son altos. Cada cliente una conexión.
- La red se puede sobrecargar. Cada sentencia SQL usa la red.

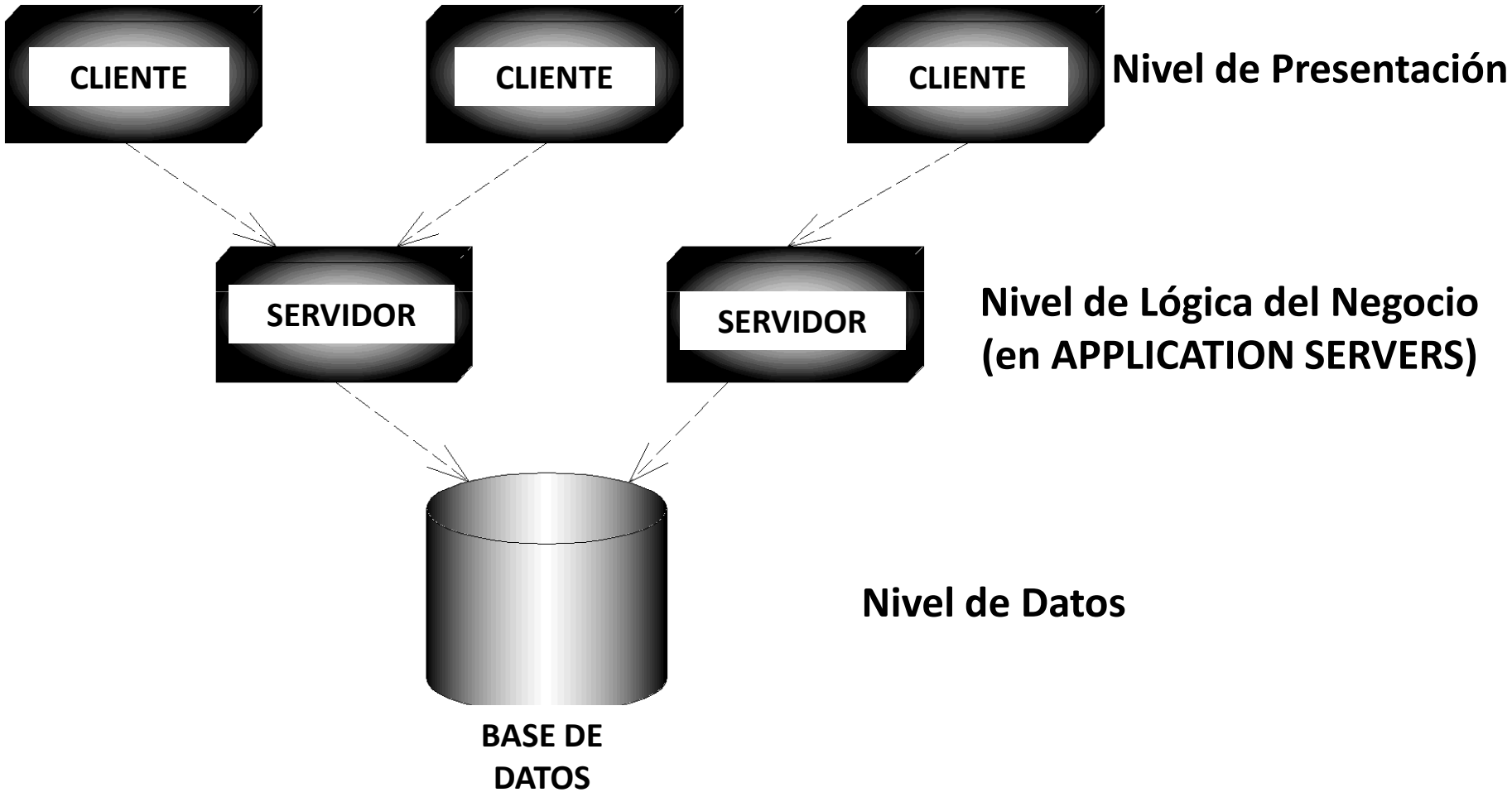
Arquitectura **física** en dos niveles: cliente flaco/servidor gordo

- Parte de la lógica del negocio se combina con el nivel de datos



• Se usan procedimientos almacenados (stored procedures) en la BD. Un procedimiento almacenado puede servir para ejecutar una serie de sentencias SQL.
• Comunicación Cliente/Servidor en SQL + Proc. almacenados
• Se necesitan APIs como por ejemplo JDBC y/o ODBC
• Deben instalarse DRIVERS de la BD en todos los clientes

Arquitectura **física** en tres niveles

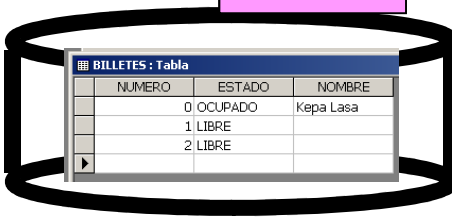


CLIENTE

Presentación

```
public class PedirBillete extends JFrame {
GestorBilletes gestorBilletes;
void jButton1_actionPerformed(ActionEvent e) {
   int res = gestorBilletes.getBillete(jTextField1.getText()).getNum();
    if (res<0) jTextArea1.append("Error al asignar billete");
    else jTextArea1.append("Asignado. \nReferencia: "+res+"\n");} }
```

BD — Datos



SERVIDOR DATOS

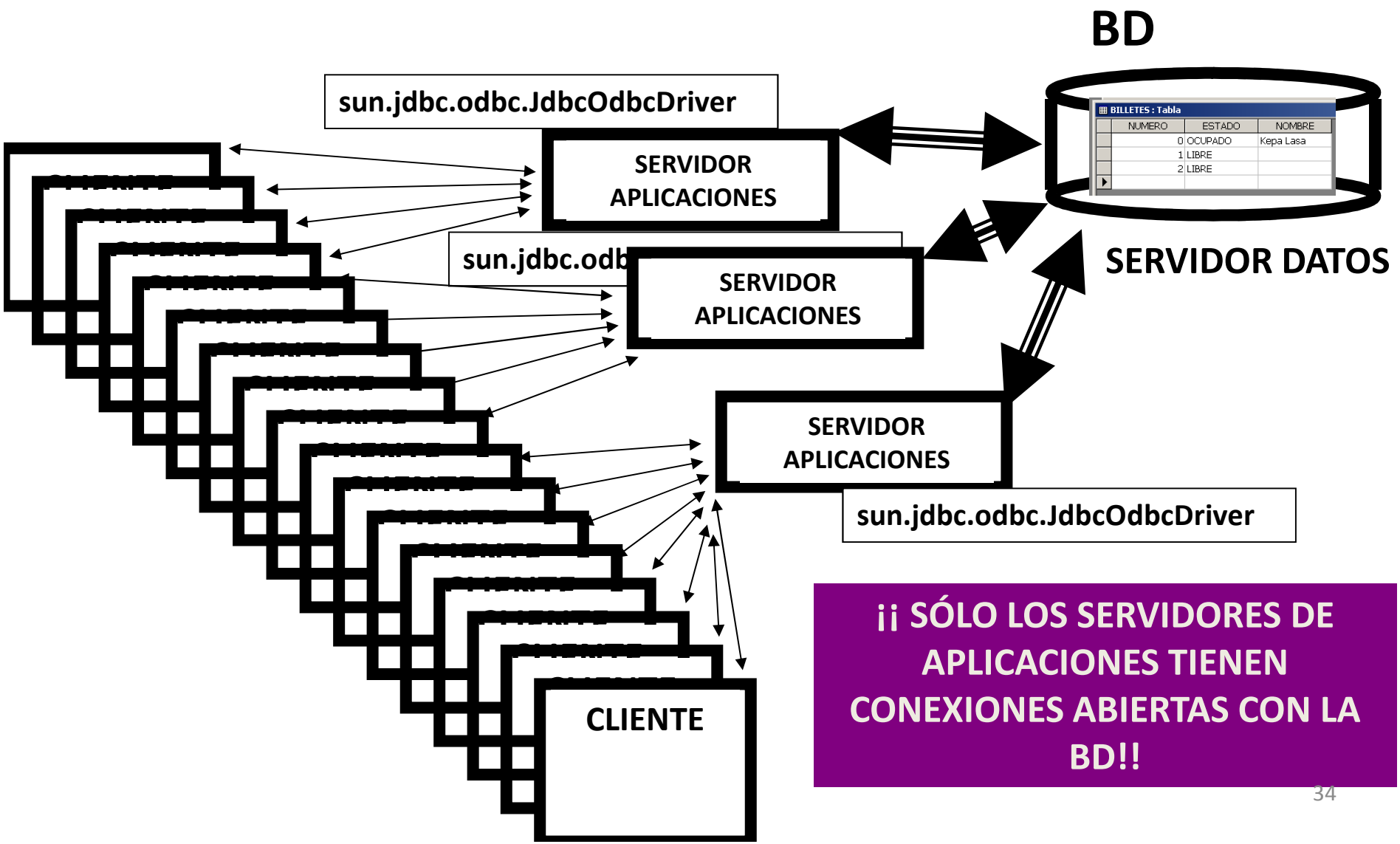
SERVIDOR APLICACIONES

Lógica del Negocio

```
public class ServidorGestorBilletesBD
      implements GestorBilletes
{ public ServidorGestorBilletesBD() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conexion=DriverManager.getConnection("jdbc:odbc:Billetes");}
   public Billete getBillete(String nom)
    {ResultSet rs = sent.executeQuery("SELECT NUMERO…";);
      int act = sent.executeUpdate("UPDATE BILLETES …“);
      if (act>0) return new Billete(n,nom); // Núm. billete asignado
     else return new Billete(-1,"");; } // No había ninguno libre}}}
```

INSTALAR LA CLASE sun.jdbc.odbc.JdbcOdbcDriver

Y DEFINIR FUENTE DATOS ODBC “Billetes”

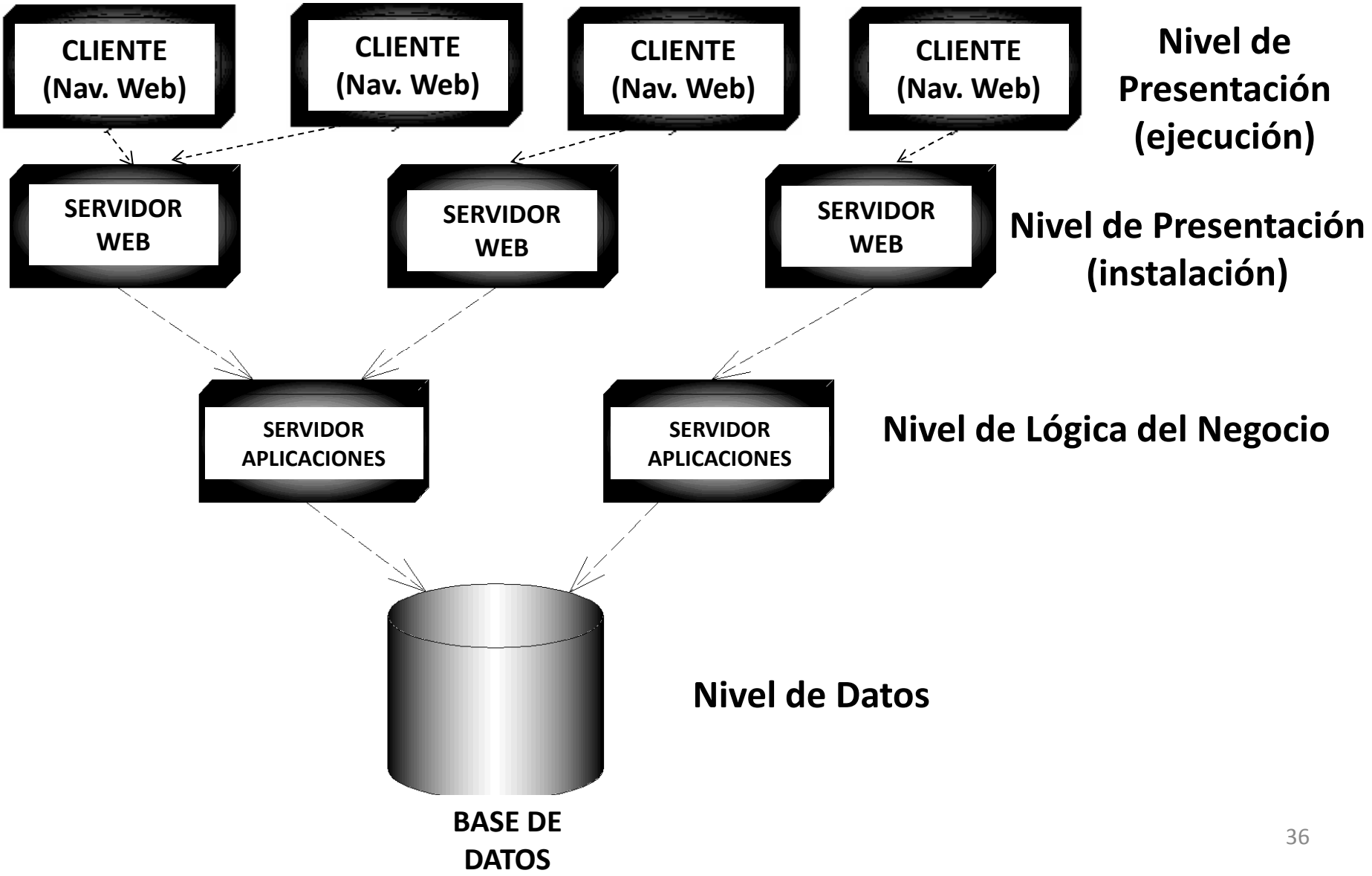

BD
BILLETES : Tabla
NUMERO
ESTADO
NOMBRE
0 OCUPADO Kepa Lasa
1 LIBRE
2 LIBRE
SERVIDOR DATOS
sun.jdbc.odbc.JdbcOdbcDriver
SERVIDOR APLICACIONES
sun.jdbc.odb
SERVIDOR APLICACIONES
SERVIDOR APLICACIONES
sun.jdbc.odbc.JdbcOdbcDriver
CLIENTE
¡¡ SÓLO LOS SERVIDORES DE APLICACIONES TIENEN CONEXIONES ABIERTAS CON LA BD!!

Arquitectura **física** en tres niveles

- Sólo hay que instalar los drivers de la BD en los nodos donde se encuentre la lógica del negocio (nodos servidores)
- Cambiar de SGBD/esquema de la BD **NO** requiere reinstalar todos los clientes. Sólo los de la lógica del negocio.
- Cambiar la lógica del negocio **NO** implica recompilar y desplegar en todos los clientes.
- Costos de conexión con la BD **NO** son tan altos. Los clientes no realizan conexiones con la BD. Sólo los servidores con la lógica del negocio lo hacen.

En general, se MEJORA en EFICIENCIA, MANTENIMIENTO y EXTENSIBILIDAD

Las aplicaciones Web permiten más niveles



CLIENTE

Navegador Web

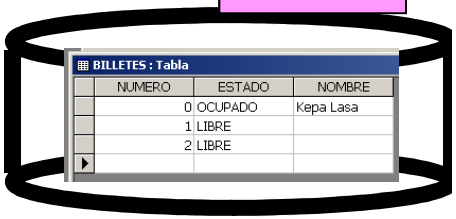
Ejecutar Presentación

SERVIDOR WEB

HTML + JSP + Bean

Instalar Presentación

BD Datos



SERVIDOR DATOS

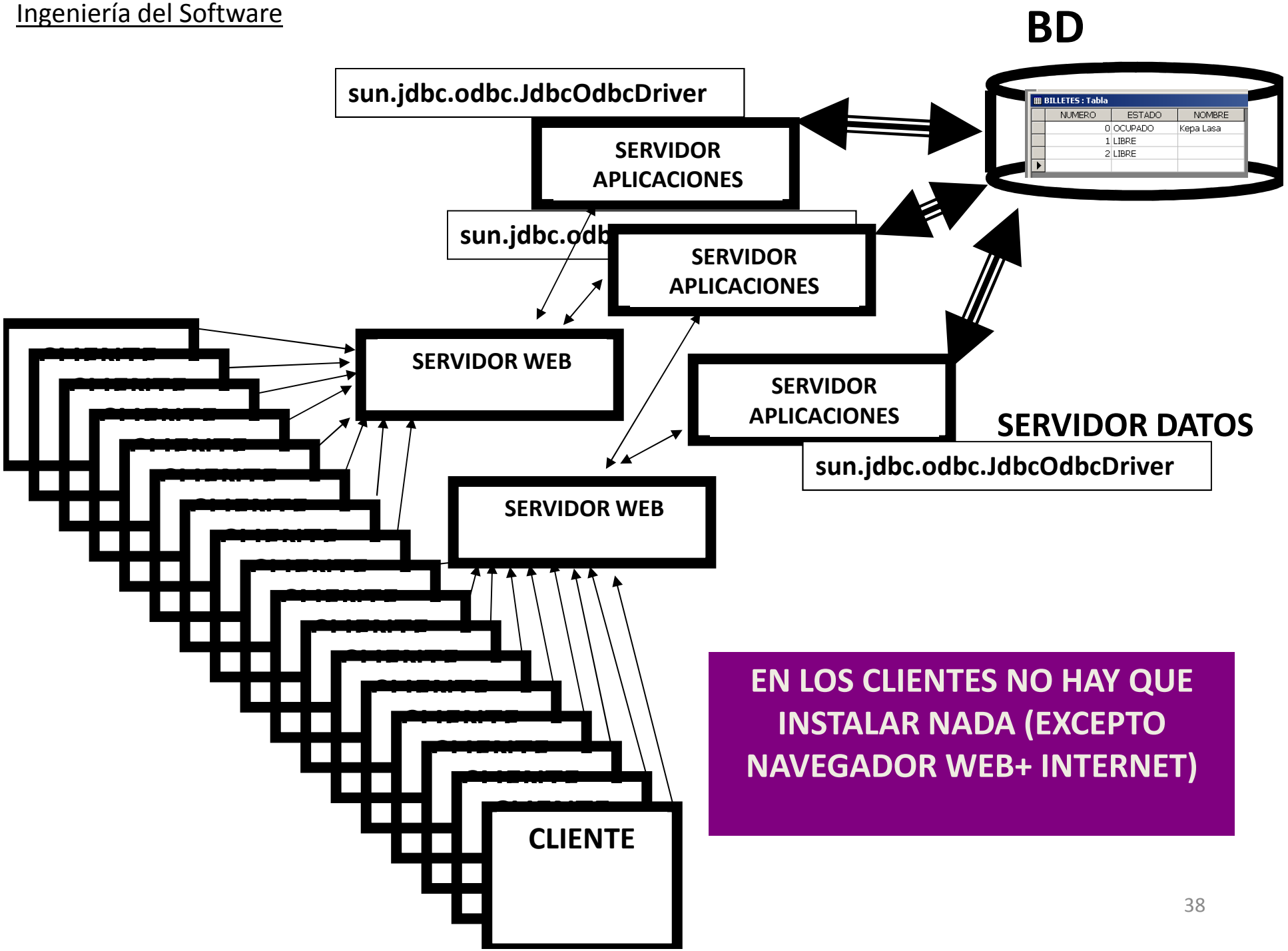
SERVIDOR APLICACIONES

INSTALAR LA CLASE sun.jdbc.odbc.JdbcOdbcDriver

Y DEFINIR FUENTE DATOS ODBC "Billetes"

Lógica del Negocio

```
public class ServidorGestorBilletesBD
      implements GestorBilletes
{ public ServidorGestorBilletesBD() {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  conexion=DriverManager.getConnection("jdbc:odbc:Billetes");}
  public Billete getBillete(String nom)
   {ResultSet rs = sent.executeQuery("SELECT NUMERO…";);
     int act = sent.executeUpdate("UPDATE BILLETES …");
     if (act>0) return new Billete(n,nom); // Núm. billete asignado
    else return new Billete(-1,"");; } // No había ninguno libre}}}
```



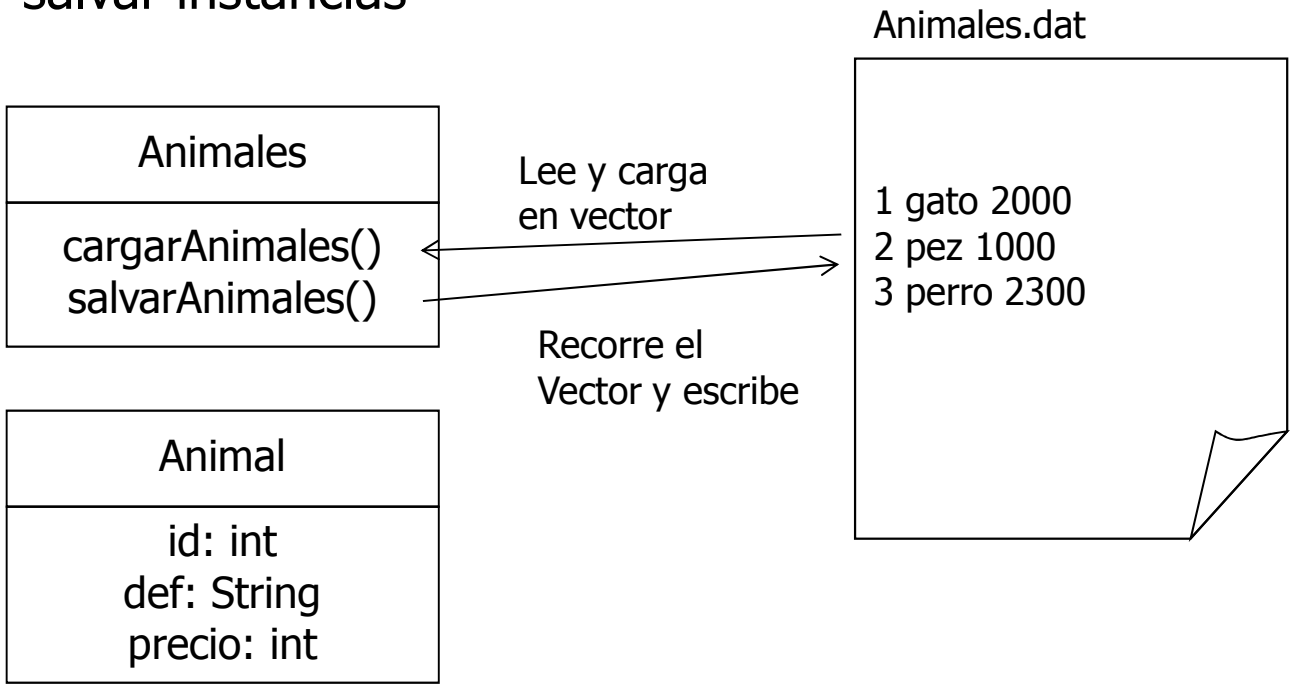
EN LOS CLIENTES NO HAY QUE INSTALAR NADA (EXCEPTO NAVEGADOR WEB+ INTERNET)

Arquitectura de un SI en tres capas (2)

- La capa de presentación
  - AWT y SWING
  - SI basados en la web (CGI, Java applets, servlets, JSPs, AJAX...)
- La capa de dominio
  - Java
- La capa de gestión de los datos
  - JDBC
  - Serialización
- SI OO distribuidos
  - RMI
  - CORBA, SOAP
  - OLE, DCOM/COM+ y la plataforma .NET

Persistencia en ficheros

- Se guardan todas las instancias de una clase en un fichero. Hay que añadir los métodos para:
  - cargar instancias
  - salvar instancias



Persistencia en ficheros

```
package tienda;

class animal {

    private int id;
    private String def;
    private int precio;

    public getId() {return id;}
    public getDef() {return def;}
    public getPrecio() {return precio;}
    public Animal (int id, String def, int precio) {
      this.id = id;
      this.def = def;
      this.precio = def;
    }
    public String toString() {return id+" "+def+" "+precio;}
}
```

Persistencia en ficheros

```
Import java.util.*; import java.io.*;

public class animales {
     private static Vector losAnimales = new Vector();
     public static final String EOF=null;
     public static void cargarAnimales() throws FileNotFoundException,IOException {
      BufferedReader entrada = new BufferedReader(new FileReader("animales.dat"));
      String linea;
      while ((linea=entrada.readLine())!=EOF) {
            StringTokenizer st = new StringTokenizer(linea, " ");
            Animal a = new Animal(Integer.parseInt(st.nextToken()),
                        st.nextToken(),
                        Integer.parseInt(st.nextToken());
            losAnimales.addElement(Animal a);
      }
      entrada.close();
}
```

Persistencia en ficheros

```
Import java.util.*; import java.io.*;

public class animales {
    ...
    public static void guardarAnimales() throws FileNotFoundException,IOException {
     PrintWriter salida = new PrintWriter(new FileWriter("animales.dat"));
     for(int i=0;i<losAnimales.size();i++) {
           salida.println(losAnimales.elementAt(i).toString());
     }
     salida.close();
}
```

Persistencia en ficheros

```
public static void main(String[] argv) {
    try {
      Animales.cargarAnimales();
      Animal a = new Anima(123, "siamés", 2400);
      Animales.guardarAnimales();
    }
    catch (Exception e) {
      System.out.println("Error: "+e.toString());
    }
}
```

Serialización

- Conversión de objetos Java en series de bytes

- Son útiles para:
  - Proporcionar persistencia de objetos:
  - Convertir objetos en bytes y guardar en streams de bytes (ficheros)
  - Enviar mensajes entre objetos de distintas máquinas
    - Usando sockets, RMI

- Para serializar objetos de una clase, en la definición de dicha clase hay que indicar que implementa la interfaz java.io.Serializable

- Entonces se pueden leer/escribir objetos en ObjectInputStream/ObjectOutputStream con los métodos readObject/writeObject

Serialización

- Los métodos writeObject y readObject funcionan bien con los tipos básicos string, arrays, vector, etc.
- writeObject serializa todos los objetos contenidos en él
- Para no serializar un atributo debe declararse trasient
- No se serializan los atributos static

```
ObjectOutputStream s = new ObjectOutputStream(new FileOutputStream("animales.dat"));
s.writeObject(losAnimales);

ObjectInputStream e = new ObjectInputStream(new FileInputStream("animales.dat"));
losAnimales = (Vector) e.readObject();
```

Persistencia en ficheros

```
package tienda;

class Animal implements serializable {

    private int id;
    private String def;
    private int precio;

    public getId() {return id;}
    public getDef() {return def;}
    public getPrecio() {return precio;}
    public Animal (int id, String def, int precio) {
      this.id = id;
      this.def = def;
      this.precio = def;
    }
    public String toString() {return id+" "+def+" "+precio;}
}
```

Persistencia en ficheros

```
Import java.util.*; import java.io.*;

public class animales {
     private static Vector losAnimales;
     private static final String nombreFichero = "animales.dat";
     public static final String EOF=null;

      synchronized public static void cargarAnimales() throws
       ClassNotFoundException, IOException {
       ObjectInputStream e =
             new ObjectInputStream(new FileInputStream(nombreFichero));
       losAnimales = (Vector) e.readObject();
     }
     synchronized public static void guardarAnimales() throws IOException {
       ObjectOutputStream s =
             new ObjectOutputStream(new FileOutputStream(nombreFichero));
       s.writeObject(losAnimales)
     }
}
```