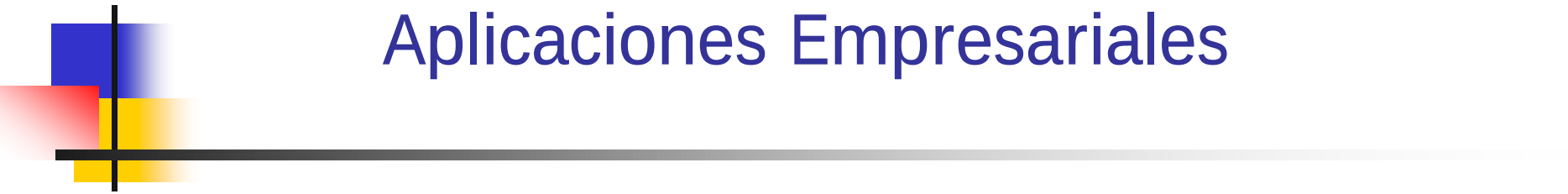


Tema 1: Introducción al Desarrollo de Aplicaciones Empresariales

Índice

- Características de las aplicaciones empresariales
- Diseño por Capas
- Arquitectura basada en Capas Típica
- Distribución de las capas
- Tecnologías Estándar JAVA
- Tecnologías JAVA por Capa
- Entorno de Desarrollo
- Alternativas a las Tecnologías JAVA

Introducción (1)

- Por aplicación empresarial nos referimos a las que abordan necesidades críticas, no forzosamente en empresas
- Deben satisfacer una serie de requisitos
  - Escalabilidad
    - Capacidad de soportar más usuarios o más carga al aumentar los recursos disponibles (e.g. hardware)
  - Alta disponibilidad
    - Tolerancia a fallos mediante replicación de recursos
  - Transaccionalidad
    - Propiedades ACID
    - Ejemplo: Transferencia Bancaria
  - Seguridad
    - No todos los usuarios pueden acceder a la misma funcionalidad

Introducción (y 2)

- Además, las aplicaciones modernas no funcionan de forma aislada
  - Dependen de otras aplicaciones (accesibles a través de la red) para proporcionar sus servicios a los usuarios
  - A menudo, ofrecen a su vez servicios a otras aplicaciones
  - Poco control sobre las otras aplicaciones
    - Pueden ser incluso de otras empresas (e.g. B2B, cloud,...)
    - No es posible modificarlas para que se ajusten a las necesidades de tu aplicación
- Esto plantea una serie de problemas
  - Acceso a través de la red
  - Interacción con aplicaciones desarrolladas con otras tecnologías
  - Evolución independiente de las distintas aplicaciones
    - Minimizar impacto de cambios

Diseño por capas (1)

- El diseño por capas es una de las técnicas de diseño más usadas en Ciencias de la Computación (e.g. Arquitecturas de Protocolos de Redes)
- Una capa “inferior” proporciona un servicio a otra capa “superior”
- El servicio ofrecido se define mediante un “contrato de servicio” (similar a interfaz JAVA)
- Permite independizar el software de ambas capas
  - A la capa superior no le importa cómo se implementa el servicio
  - Cambios en la manera de implementar una capa no afectan a la otra
- Será clave para facilitar mantenimiento, escalabilidad y tolerancia a fallos

Diseño por capas (y 2)

- Ventajas
  - Los desarrolladores de una capa no necesitan conocer las tecnologías usadas en otras capas
  - Cada capa puede ser desarrollada en paralelo con el resto de capas
  - Se facilita el mantenimiento del software
    - Cambios en la implementación de una capa (e.g. uso de nuevas tecnologías) no conllevan cambios en el resto de capas
  - Facilita la escalabilidad y la tolerancia a fallos: pueden aumentarse los recursos sólo de las capas que lo necesitan
- Riesgos
  - El software es más complejo
  - En ciertos casos una capa podría hacer su función de forma más óptima si “supiese” como funciona otra capa

Arquitectura Típica (1)

- Separación clara entre las capas “modelo” e “interfaz”

- Capa Modelo
  - Conjunto de clases que implementan la **lógica** de negocio de los **casos de uso** de la aplicación
  - Suele dividirse en dos subcapas
    - Capa Acceso a Datos
      - Bases de datos
      - Otras aplicaciones
    - Capa Lógica de Negocio
      - Implementa la lógica de negocio, usando la capa de acceso a datos para leer / escribir los datos que necesita
  - Independiente de la interfaz

Arquitectura Típica (2)

- Distinguimos dos casos
  - Usuario “humano”
    - Capa Interfaz Gráfica: interfaz gráfica que permite a los usuarios utilizar la funcionalidad de la capa modelo
  - Otras aplicaciones como usuarias
    - Capa Servicios: interfaz programática orientada a que otras aplicaciones remotas utilicen la funcionalidad de la capa modelo
      - Las aplicaciones remotas invocarán la capa Servicios a través de una capa a la que llamaremos capa Acceso a Servicios

Arquitectura Típica (3)

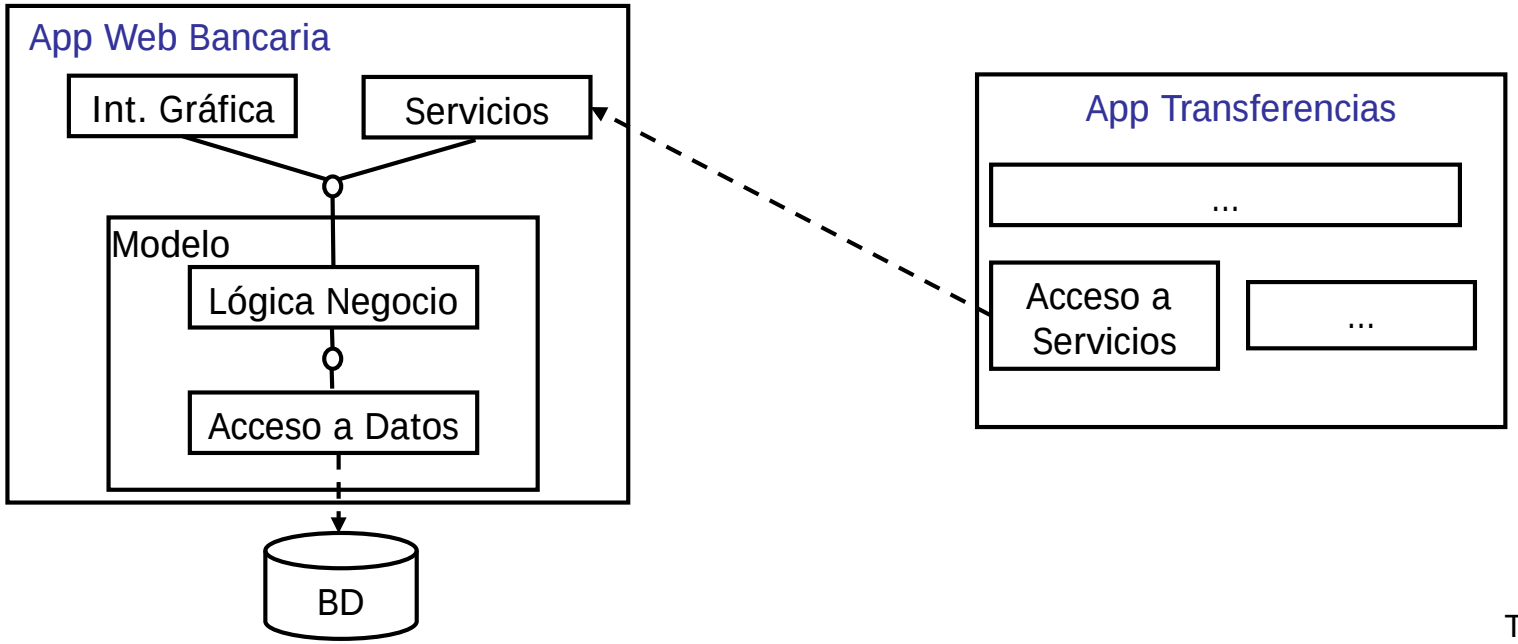
- Ejemplo => aplicación web bancaria
  - Capa Modelo: conjunto de clases que implementan los casos de uso (e.g. crear cuentas, destruirlas, encontrarlas por distintos criterios, ordenar transferencias, etc.)
    - Capa Acceso a Datos: utiliza una BD para guardar la información de cuentas y clientes
    - Capa Lógica de Negocio: implementa la lógica de los casos de uso sin preocuparse de cómo se almacenan los datos
    - Ejemplo: caso de uso transferencia

1. Leer saldoOrigen //llamada a capa acceso a datos
2. Si saldoOrigen > importe {
   1. nuevoSaldoOrigen = saldoOrigen – importe
      //llamada a capa acceso a datos
   2. Escribir nuevoSaldoOrigen
      //llamada a capa acceso a datos
   3. Escribir nuevoSaldoDestino a su valor actual + importe

   } sino lanzar error saldo insuficiente

Arquitectura Típica (4)

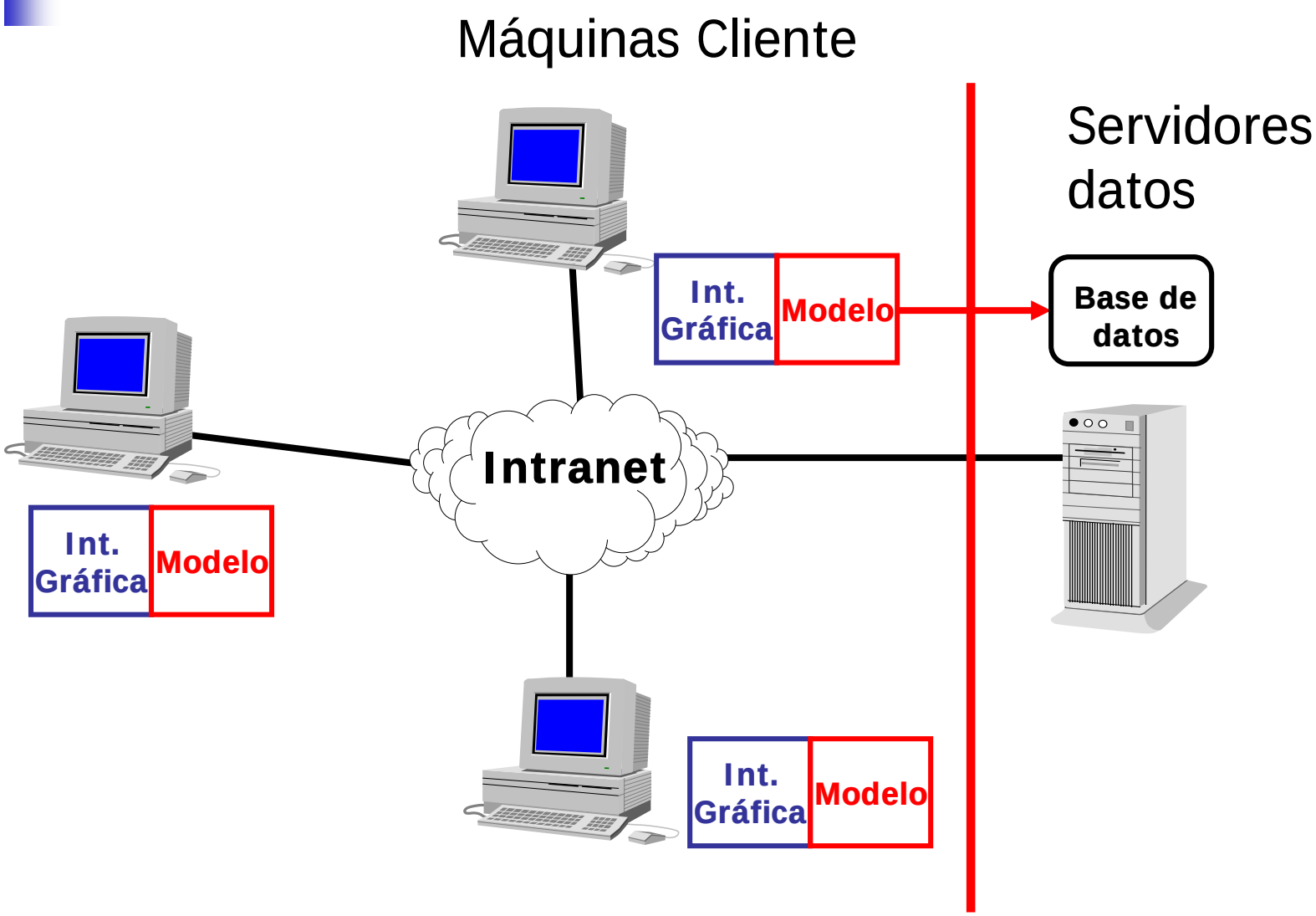
- Ejemplo => aplicación web bancaria
  - Capa Interfaz gráfica: interfaz web que utiliza la capa modelo para permitir a los usuarios crear cuentas, buscarlas, hacer transferencias, etc.
  - Capa Servicios: ofrece una API que puede ser invocada remotamente por otras aplicaciones (e.g. una aplicación que necesita automatizar la emisión de un conjunto de transferencias periódicas)



Arquitectura Típica (y 5)

- Ventajas potenciales de la separación
  - Cada capa puede ser desarrollada por personas con un perfil distinto
  - Es posible reusar la capa modelo en distintas interfaces gráficas y por distintas aplicaciones
  - Cambios en la manera de implementar una capa no afectan a las otras capas
    - Ejemplo: si tras una fusión bancaria, ahora hay dos bases de datos, necesitamos cambiar la implementación de la capa modelo, pero no necesitamos cambiar las interfaces gráficas ni las aplicaciones consumidoras
  - Normalmente, cada capa puede ejecutarse en máquinas diferentes
    - Aumenta escalabilidad
  - Es posible replicar la capa modelo sin necesidad de replicar a las aplicaciones consumidoras
    - Facilita escalabilidad y tolerancia a fallos

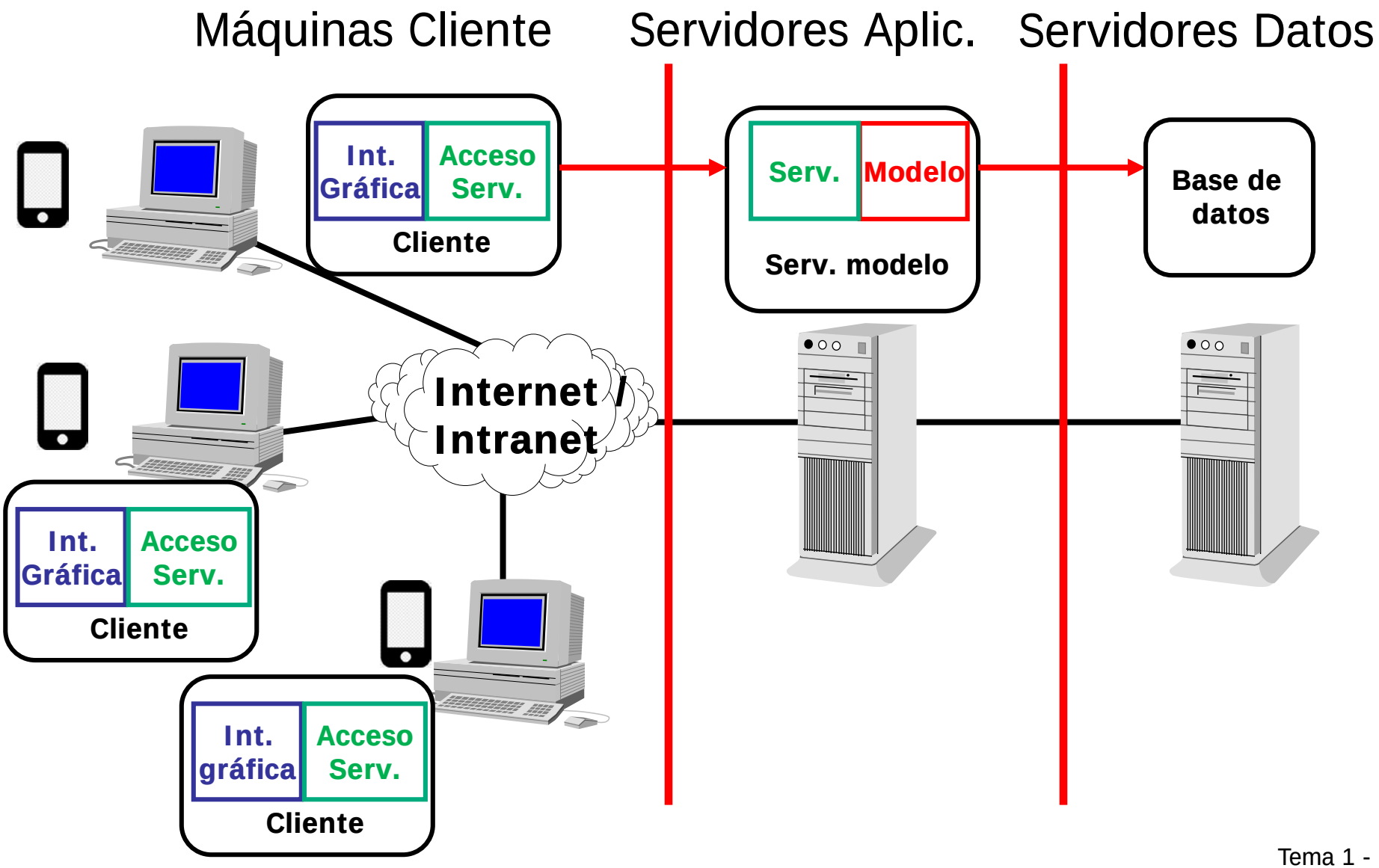
Distribución de las Capas: Opción 1 (1)



Distribución de las Capas: Opción 1 (y 2)

- Problemas
  - Cambios en la implementación de la capa Modelo => recompilación de toda la aplicación y reinstalación en máquinas cliente
    - Cambios en acceso a fuentes de datos (típicamente BD, aunque no necesariamente)
    - Cambios en la lógica del modelo
  - Seguridad: Código y accesos a BD desde el cliente
- Solución
  - Modelo en servidor intermedio
    - Cambios en implementación del modelo sólo afectan al servidor
  - Clientes de escritorio
    - Sólo disponen de la interfaz gráfica
    - Acceden al servidor que implementa el modelo

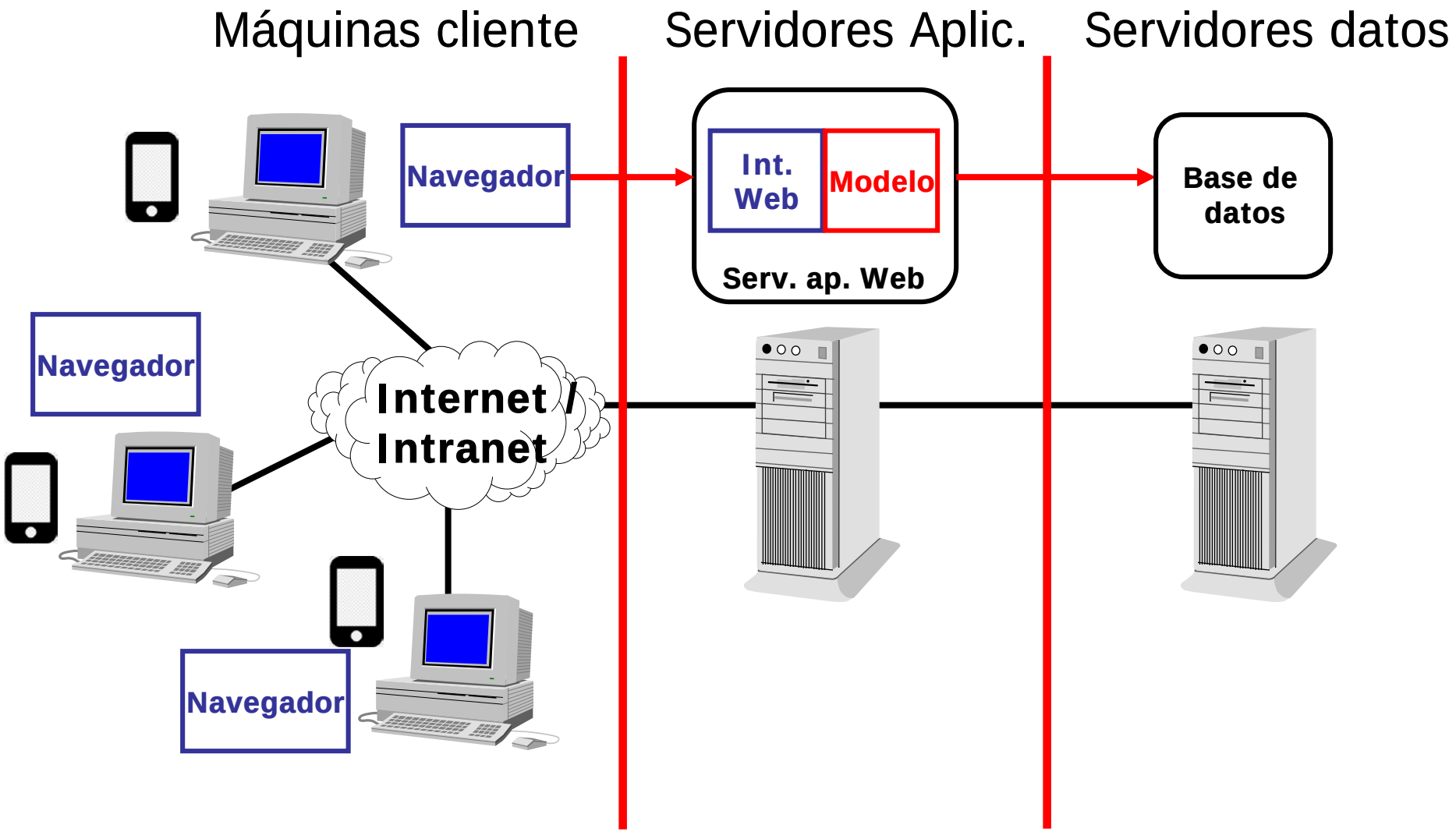
Distribución de las Capas: Opción 2 (1)



Distribución de las Capas: Opción 2 (y 2)

- Capa Servicios y Capa Acceso a Servicios necesarias porque hay invocación remota de la capa Modelo
- Problema
  - Cambios en la implementación de la interfaz gráfica => recompilación de la aplicación cliente y reinstalación en máquinas cliente
  - Usar Interfaz Web lo soluciona
- A esta distribución a veces se le llama “arquitectura en 3 capas”
  - “capa” tiene un significado diferente: conjunto de máquinas que cumple una función diferenciada (cliente, servidores de aplicaciones y servidores de datos)
- ¿Ventaja interfaz standalone frente a Interfaz Web?
  - Puede proporcionar cierta funcionalidad, incluso con conexión de red intermitente o lenta

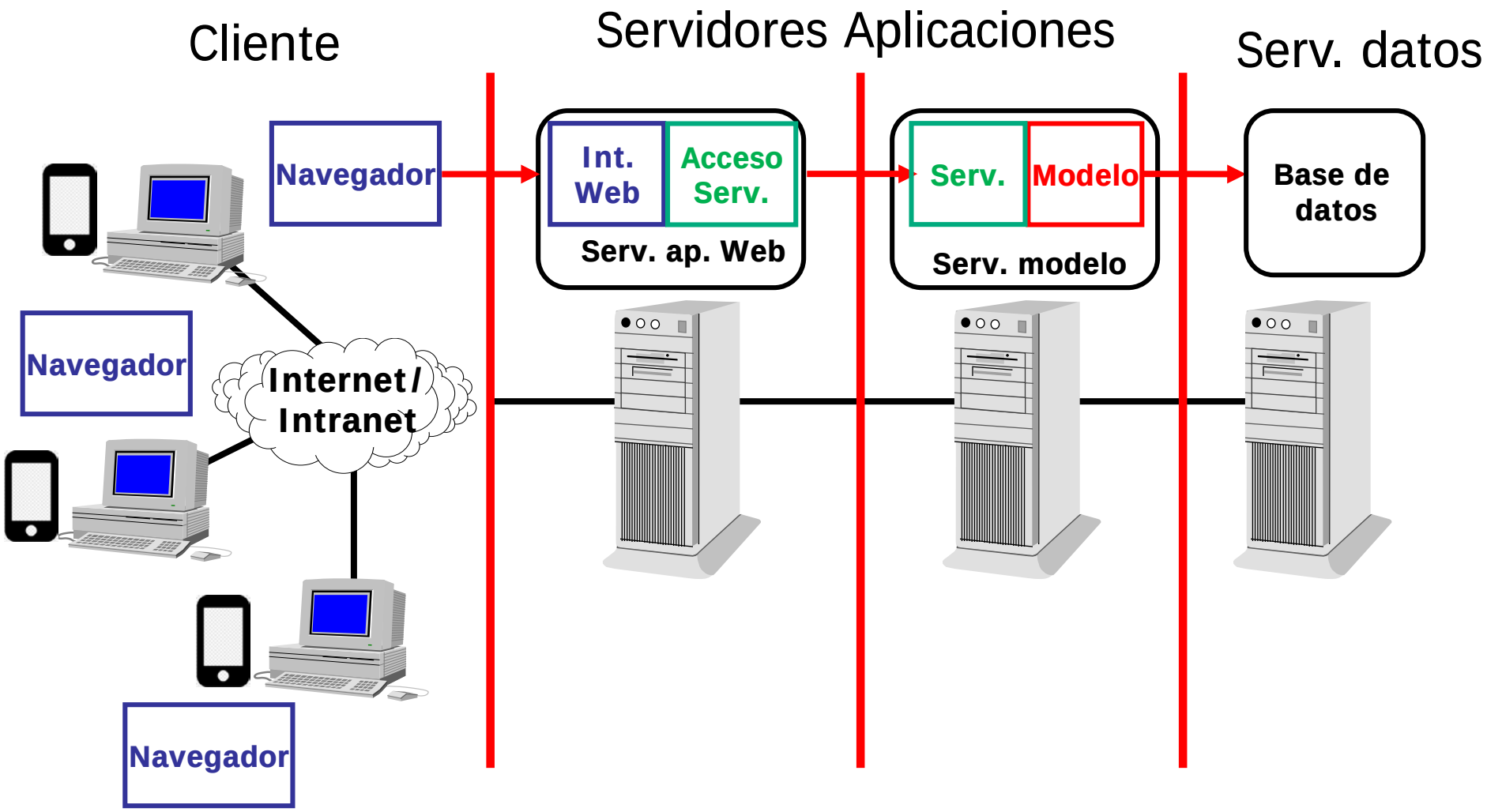
Distribución de las Capas: Opción 3 (1)



Distribución de las Capas: Opción 3 (y 2)

- Las aplicaciones Web del lado servidor se instalan en **servidores de aplicaciones**
- Cambios en la capa Interfaz Gráfica o en la capa Modelo, sólo requieren reinstalar la aplicación en el servidor de aplicaciones
- Los servidores de aplicaciones suelen tener soporte para escalabilidad y disponibilidad
  - Se pueden replicar en varias máquinas (“cluster de máquinas”)
  - Se emplea un balanceador de carga
    - Recibe todas las peticiones HTTP
    - Envía cada petición HTTP a uno de los servidores de aplicaciones
  - Escalabilidad
    - Para atender más peticiones => hay que añadir más máquinas al cluster
  - Disponibilidad
    - Si una máquina se cae, todavía quedan otras instancias del servidor de aplicaciones en el cluster

Distribución de las Capas: Opción 4 (1)



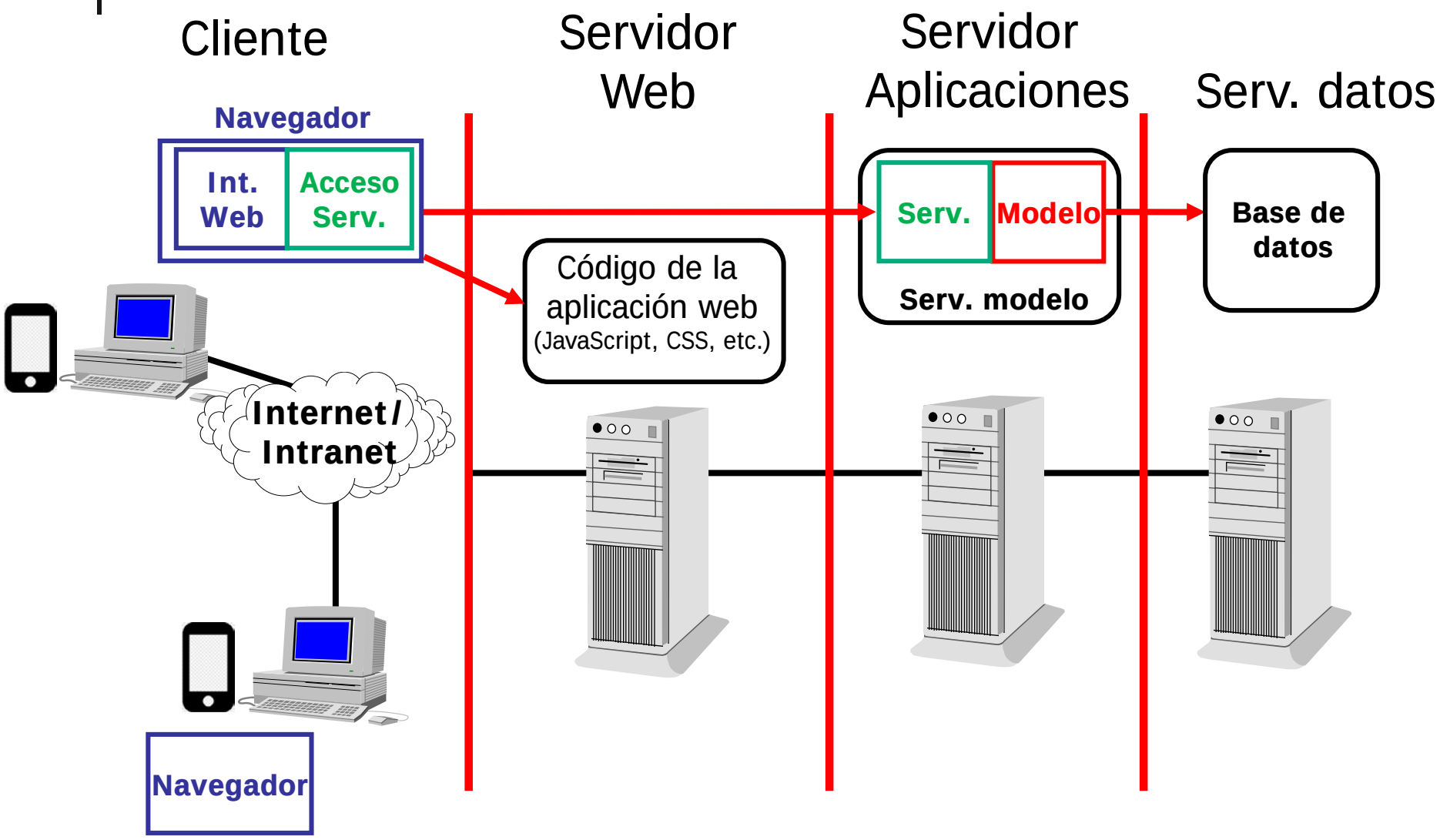
Distribución de las Capas: Opción 4 (2)

- A esta distribución, a veces se le llama “arquitectura en cuatro capas”
- La capa Servicios se puede implementar utilizando protocolos y tecnologías usados para el desarrollo de aplicaciones web ➔ Es necesario ejecutarla dentro de un servidor de aplicaciones
- Ventajas
  - Múltiples aplicaciones (con interfaz gráfica o no) pueden invocar al modelo, sin necesidad de replicar el código del modelo en cada aplicación
  - Permite que la capa Interfaz Gráfica y la capa Modelo estén construidas con tecnologías diferentes

Distribución de las Capas: Opción 4 (y 3)

- Ejemplo
  - Existe una capa Modelo desarrollada en COBOL
  - La capa Modelo es grande, funciona correctamente, lleva muchos años funcionando y manteniéndose
  - Se desea construir una aplicación Web para poder invocar casos de uso de la capa Modelo desde un navegador
  - Se elige una tecnología moderna para implementar la interfaz Web fácilmente (e.g. Java)
  - No es viable volver a construir la capa Modelo en la misma tecnología que la usada en la interfaz Web (tiempo, coste, razones tecnológicas, etc.)
  - Solución: habilitar la capa Modelo para acceso remoto (e.g. Servicios Web)
  - Permite también que otras aplicaciones remotas desarrolladas con tecnologías modernas (aunque no sean interfaces gráficas) utilicen esa funcionalidad

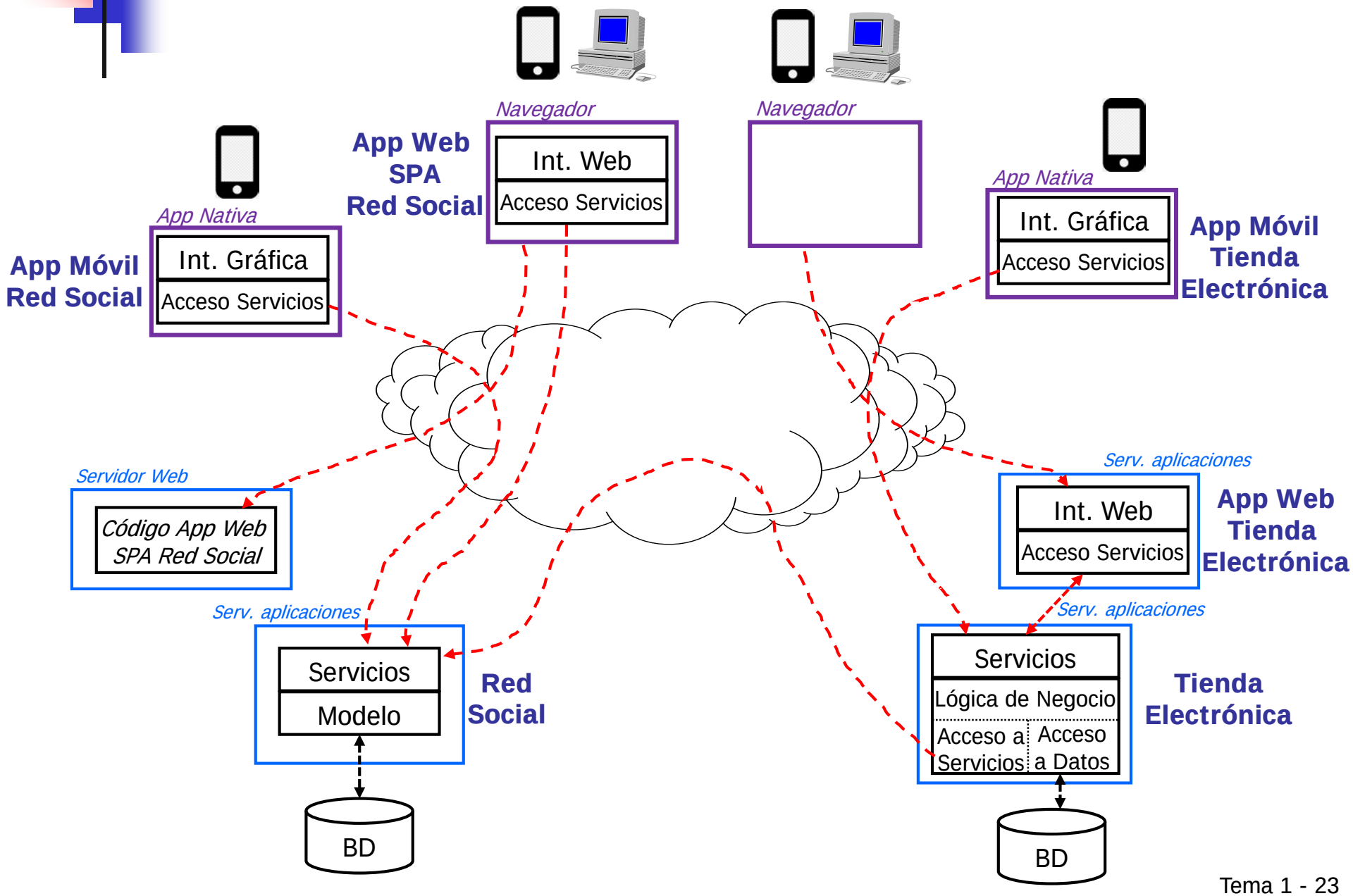
Distribución de las Capas: Opción 5 (1)



Distribución de las Capas: Opción 5 (y 2)

- Esta es la distribución que siguen las aplicaciones web SPA (Single Page Application)
  - El navegador se conecta a un servidor web para obtener todo el código de la aplicación web
  - La aplicación web se ejecuta dentro del navegador de la máquina cliente ➔ Es necesario utilizar capas Servicios y Acceso al Servicio porque hay invocaciones remotas
- Cambios en la capa Interfaz Gráfica solo requieren reinstalar la aplicación en el servidor web
- Cambios en la capa Modelo solo requieren reinstalar la aplicación en el servidor de aplicaciones

Caso Práctico (1)



Caso Práctico (2)

- La aplicación Tienda Electrónica ofrece los servicios típicos de una tienda de comercio electrónico, por ejemplo, buscar productos, añadir al carrito, comprar, ver pedidos, etc.
  - Dado que se han desarrollado tanto una aplicación Web como una aplicación móvil nativa para interactuar con la tienda, se ha decidido ofrecer la funcionalidad de la capa Modelo a través de una capa Servicios y que ambas aplicaciones accedan al modelo a través de ella
- La aplicación Red Social es una red social desarrollada por otra compañía
  - También ha desarrollado dos aplicaciones para poder utilizarla: una aplicación móvil nativa y una aplicación web SPA. Al igual que en el caso anterior, ambas aplicaciones acceden a la funcionalidad de la capa Modelo a través de una capa Servicios
  - La capa Servicios también se ha diseñado para que otras aplicaciones puedan interactuar con ella (en la práctica, podría ser otra capa Servicios diferente)

Caso Práctico (y 3)

- La tienda electrónica tiene una cuenta en la red social y desea que al ejecutar ciertas operaciones de la capa Modelo, esta interactúe con la Red Social
  - Algunos ejemplos podrían ser:
    - Al añadir un nuevo producto, añadir una publicación en la red social
    - Al obtener los detalles de un producto, obtener también el número de “likes” que tiene la publicación asociada a ese producto en la red social
    - Añadir un comentario cuando queden pocas unidades de un producto, o cuando se reciban nuevas unidades de un producto agotado
    - Etc.
  - El diseño de la capa Modelo incluye
    - Una capa Acceso a Datos para acceder a los datos que guarda en su Base de Datos local (información sobre los productos, usuarios, compras, etc.)
    - Una capa Acceso a Servicios para interactuar con la aplicación Red Social (añadir una publicación, obtener likes de una publicación, añadir un comentario, etc.)
    - Una capa Lógica de Negocio que hace uso de las operaciones ofrecidas por las dos capas anteriores para implementar la lógica de la aplicación

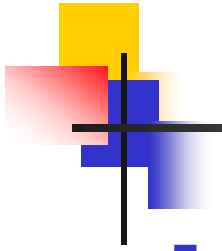
Tecnologías estándar Java

- Oracle, con la ayuda de otros fabricantes (e.g. IBM, Google, etc.), **estandariza** un conjunto de APIs para el desarrollo de aplicaciones Java
  - Existen **múltiples implementaciones** de distintos fabricantes, incluso muchas Open Source
  - Una aplicación construida con estas APIs no depende de una implementación particular
- **Java SE (Java Platform, Standard Edition)**
  - API básica + herramientas básicas (máquina virtual, compilador, etc.)
  - http://www.oracle.com/technetwork/java/javase/overview/index.html
- **Java ME (Java Platform, Micro Edition)**
  - API análoga a Java SE orientada a dispositivos con recursos limitados (e.g. televisiones, reproductores Blu-ray, impresoras, etc.)
  - http://www.oracle.com/technetwork/java/javame/index.html
- **Java EE (Java Platform, Enterprise Edition) / Jakarta EE**
  - Se apoya en Java SE y dispone de APIs para la construcción de aplicaciones empresariales (inclusive aplicaciones Web)
    - http://www.oracle.com/technetwork/java/javaee/overview/index.html
  - Desde 2018 es gestionada por la Eclipse Foundation y pasa a denominarse **Jakarta EE**
    - https://jakarta.ee/

Implementaciones de Java/Jakarta EE

- Las aplicaciones desarrolladas usando Java/Jakarta EE necesitan ser ejecutadas dentro de un servidor de aplicaciones
- Existen varios fabricantes que venden **servidores de aplicaciones certificados Java/Jakarta EE**
  - Lista completa en
    - https://www.oracle.com/java/technologies/compatibility-jsp.html
    - https://jakarta.ee/compatibility/
  - Algunos ejemplos
    - IBM WebSphere Liberty Application Server
    - Red Hat JBoss Enterprise Application Platform
    - Jetty: http://www.eclipse.org/jetty
    - Tomcat: http://tomcat.apache.org
  - Si una aplicación usa sólo las APIs estándar ➔ es posible instalarla en cualquier servidor de aplicaciones Java/Jakarta EE
  - ¡No se depende de un fabricante!

Tecnologías Capa Acceso a Datos



- API: JDBC (Java DataBase Connectivity)
- Posibilita el acceso a bases de datos relacionales, abstrayendo al programador de las peculiaridades del API de cada base de datos
- El programador puede lanzar consultas (lectura, actualización, inserción y borrado), agrupar consultas en transacciones, etc.
- Estudiaremos sus principios básicos en el Tema 2 (clases de prácticas)
- Existen soluciones de más alto nivel (e.g. JPA/Hibernate)

Tecnologías Capa Lógica de Negocio

- No usaremos ninguna tecnología de apoyo
- Existen tecnologías que ayudan a implementar funcionalidades útiles en muchas aplicaciones
  - Políticas de transacciones y seguridad
  - Utilidades que simplifican programación (e.g. factorías)
- Ejemplo: Spring Framework
  - https://spring.io/projects/spring-framework

Tecnologías Capa Interfaz Web

- API básica: Servlets
  - La veremos en el Tema 6
- Existen muchas otras opciones de más alto nivel (e.g. JSP)
- **No es el foco de la asignatura**
  - En el Tema 8 se presenta un resumen teórico de las opciones disponibles

Tecnologías Capa Servicios (1)

- Servicios web REST
  - Están basados en tecnologías surgidas alrededor de la Web
  - El cliente invoca al servidor usando HTTP
  - Para enviar datos se usan formatos de representación como XML y JSON (los veremos en el Tema 5)
  - Dan soporte al estilo arquitectónico “RESTful”
  - Los veremos en detalle en el Tema 6

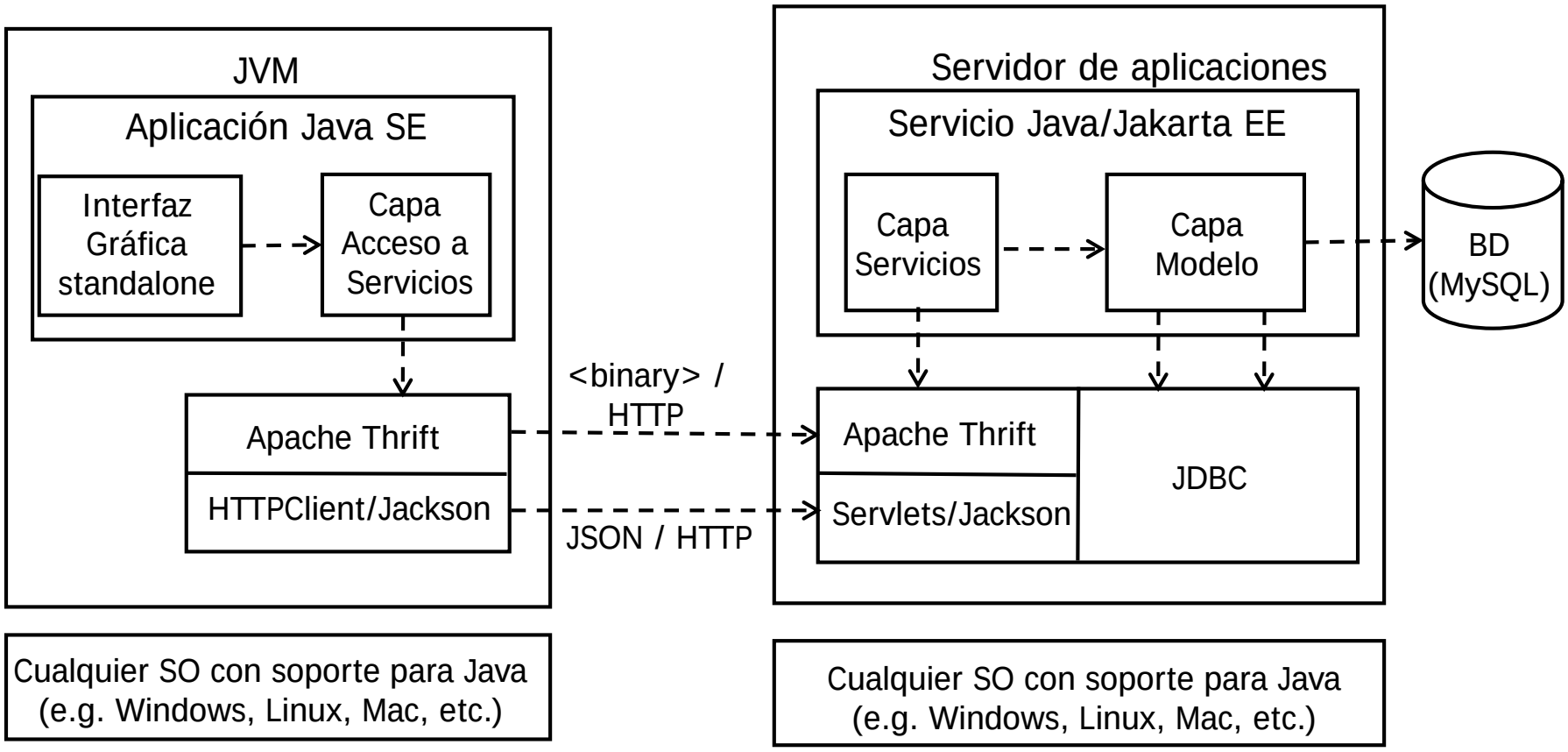
Tecnologías Capa Servicios (y 2)

- El Modelo RPC (Remote Procedure Call)
  - Un proceso expone su funcionalidad como una serie de operaciones (procedimientos) que pueden ser invocados desde cualquier programa en cualquier punto de la red
  - El objetivo que persigue el modelo RPC es que para el programador sea igual de sencillo invocar una operación de una aplicación remota que invocar un procedimiento de una librería local
  - Se basa en generar automáticamente código que se encarga de abstraer al programador de los detalles de crear y parsear mensajes, y de enviarlos/recibirlos por la red
  - Existen numerosos frameworks que siguen este modelo
    - En el Tema 7 veremos uno de estos frameworks: Apache Thrift
      - Disponible en Java y en muchos otros lenguajes

Tecnologías de la asignatura

- En esta asignatura, se prefiere el uso de las tecnologías de más bajo nivel para entender los principios básicos
- Entendiendo los principios básicos es relativamente sencillo entender como funcionan los *frameworks* de más alto nivel

Nuestro entorno de desarrollo (1)



Nuestro entorno de desarrollo (2)

- Nuestro servicio necesitará ser instalado en un servidor de aplicaciones JAVA para funcionar
- Como servidor de aplicaciones utilizaremos
  - Jetty para desarrollo
  - Tomcat para “producción”
  - Es posible utilizar cualquier otro servidor de aplicaciones Java
- Como BD utilizaremos
  - MySQL (http://mysql.com)
  - Es posible utilizar cualquier otra BD con driver JDBC

Nuestro entorno de desarrollo (y 3)

- Además, usaremos
  - Maven
    - http://maven.apache.org
    - Para automatizar la construcción del software (inclusive la ejecución de pruebas de integración de la capa modelo)
  - Git
    - https://git-scm.com/
    - Repositorio de control de versiones
  - IntelliJ IDEA
    - https://www.jetbrains.com/idea/
    - Es posible utilizar otro IDE

Alternativas a las tecnologías Java (1)

- .NET
  - https://www.microsoft.com/net
  - Define un Common Language Runtime (CLR) y un IL (Intermediate Language) al que todos los lenguajes conformes a .NET compilan
    - Idea similar a la máquina virtual de Java
  - Lenguajes
    - Visual C# .NET, Visual Basic .NET, etc.
  - Tecnologías
    - ADO.NET: para capa modelo
    - ASP.NET: para interfaz Web

Alternativas a las tecnologías Java (y 2)

- LAMP
  - Linux + Apache + MySQL + Perl/PHP/Python
- Ruby on Rails
  - https://rubyonrails.org
  - Framework Web para el lenguaje Ruby (http://www.ruby-lang.org)
- JavaScript
  - Existen multitud de librerías del lado cliente y servidor
  - Node.js: Entorno de ejecución JavaScript
- ...