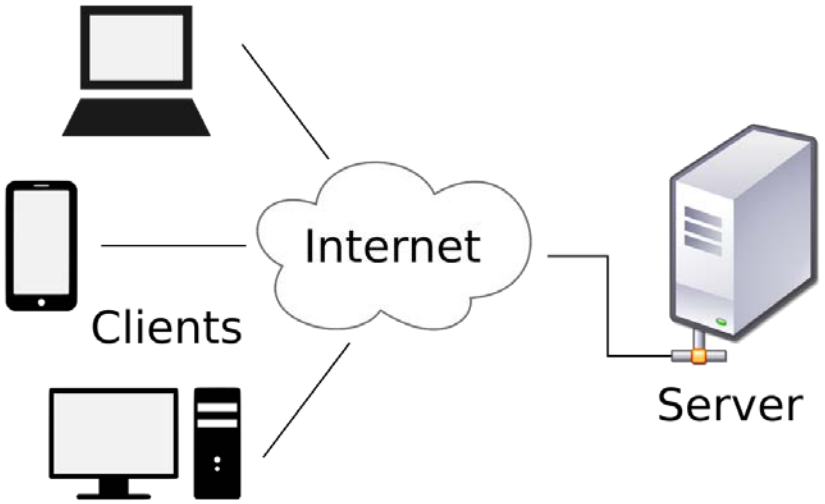
escuelaartegranada

TEMA 1: IMPLANTACIÓN DE ARQUITECTURAS WEB
PARTE I: ARQUITECTURAS WEB

Despliegue de Aplicaciones Web
2º Desarrollo de Aplicaciones Web
Curso 2018/2019 – Profesor: Borja Molina Zea

Índice

Modelo Cliente-Servidor



- El **modelo cliente-servidor** es una arquitectura que involucra uno o más clientes solicitando servicios/datos a un servidor o varios.
- Clientes y servidor intercambian mensajes, este intercambio se ha de realizar en lenguajes que ambos conozcan y siguiendo unas reglas → **protocolo de comunicación**

Características Cliente-Servidor

Cliente

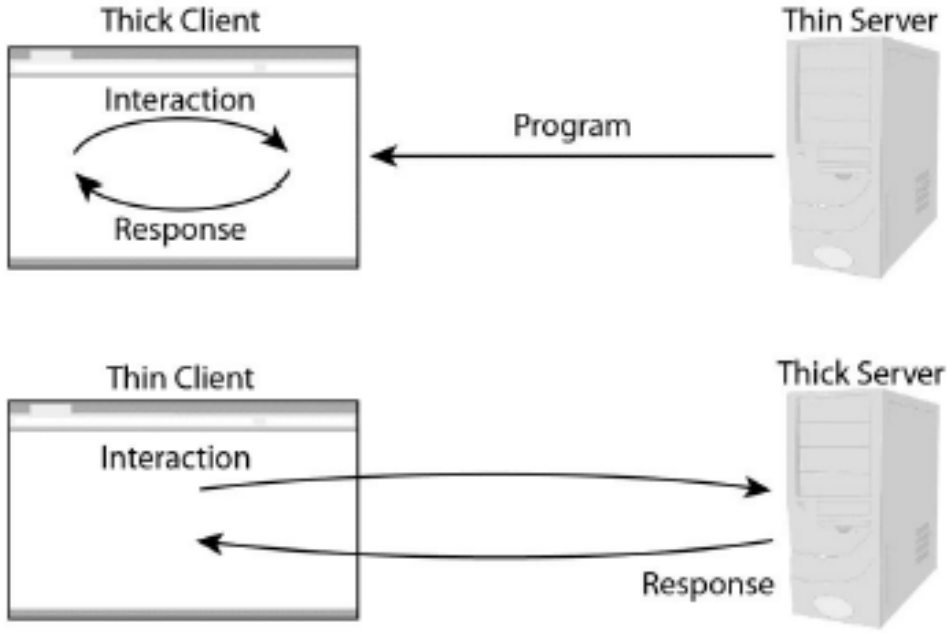
- Inicia solicitudes o peticiones. Tiene un papel activo en la comunicación.
- Espera y recibe respuestas del servidor.
- Puede conectarse a varios servidores a la vez.
- Normalmente interactúa con los usuarios mediante interfaz gráfica

Servidor

- Espera a que le lleguen las solicitudes de los clientes. Papel pasivo.
- Tras escribir una solicitud, la procesan y envían la respuesta al cliente.
- Por lo general, aceptan conexiones desde un gran número clientes.
- Los usuarios finales no interactúan directamente

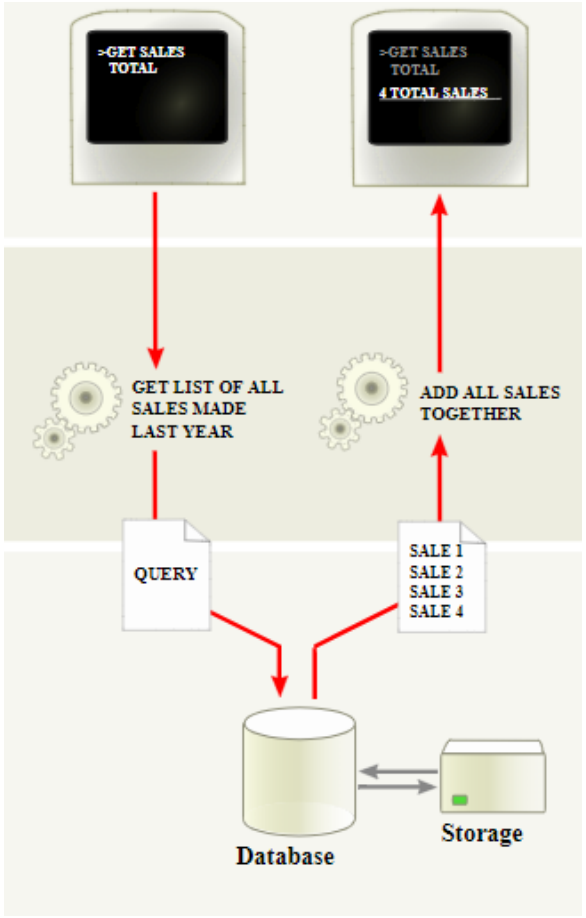
Clasificaciones de Cliente-Servidor

- Según el tamaños de los componentes: **Fat Client** (thin Server) o **Fat Server** (thin client)
- Según la naturaleza del **servicio ofrecido**: en función de las capacidades ofrecidas por el servidor. Ficheros, base de datos, transacciones o servidores web.
- **Reparto de funciones**: cómo de distribuyen las distintas capas lógicas

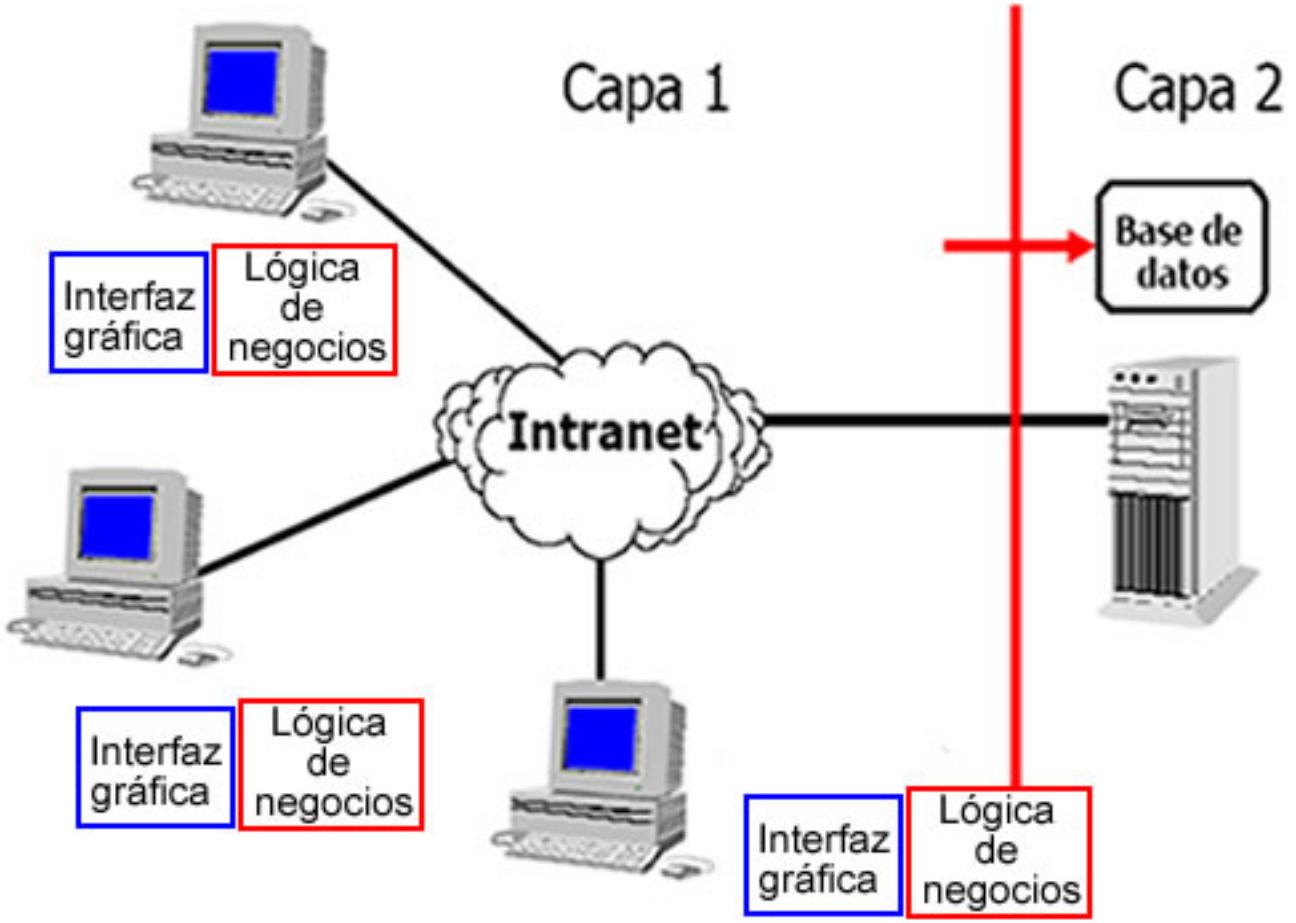


Arquitecturas multicapa

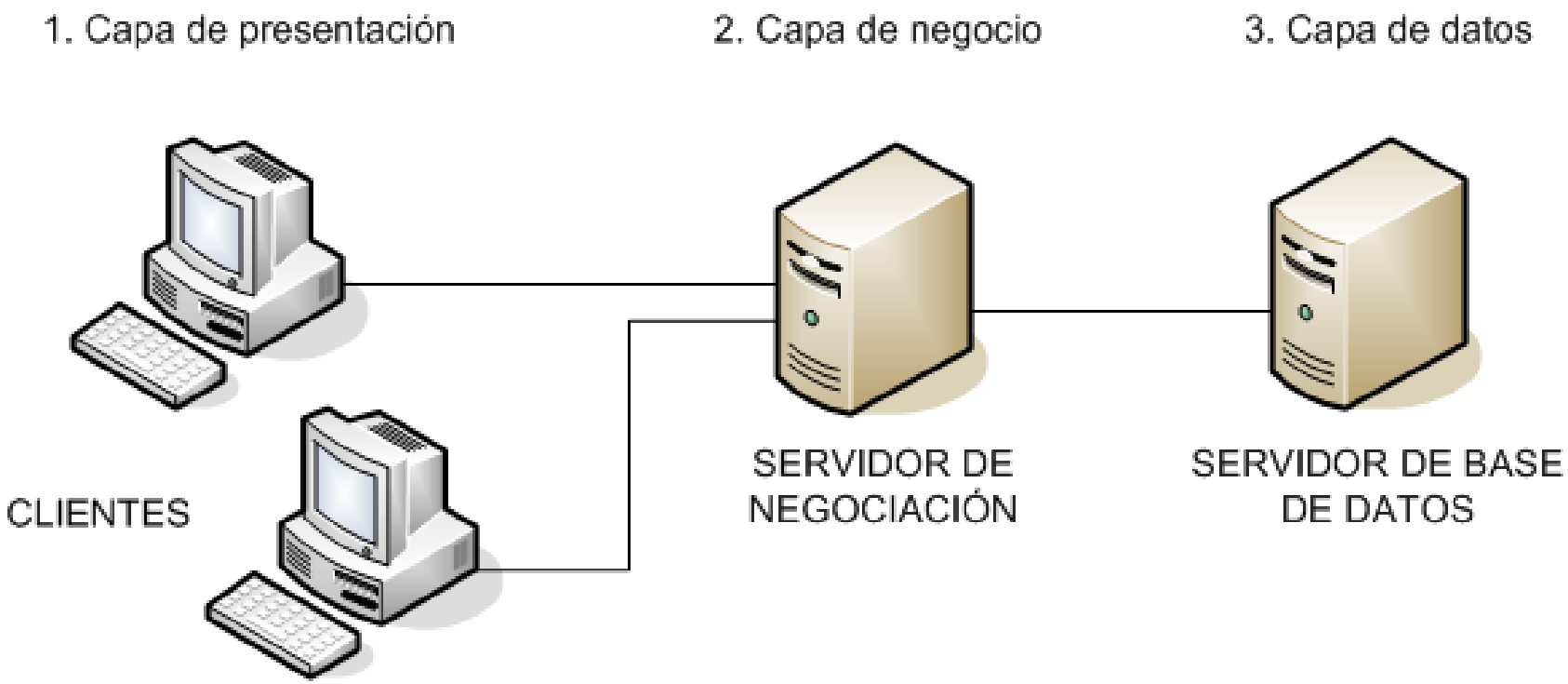
- **Capa de datos**: guarda, actualiza y envío los datos de la aplicación.
- **Capa de negocio**: procesa las peticiones del usuario, realiza decisiones y cálculos. Recoge datos de la capa de presentación y los envía a la capa de datos y viceversa.
- **Capa de presentación**: Es manejada por el usuario final, suele contar con interfaz gráfica. Muestra al usuario los datos y le permite trabajar con ellos. Validación de datos.



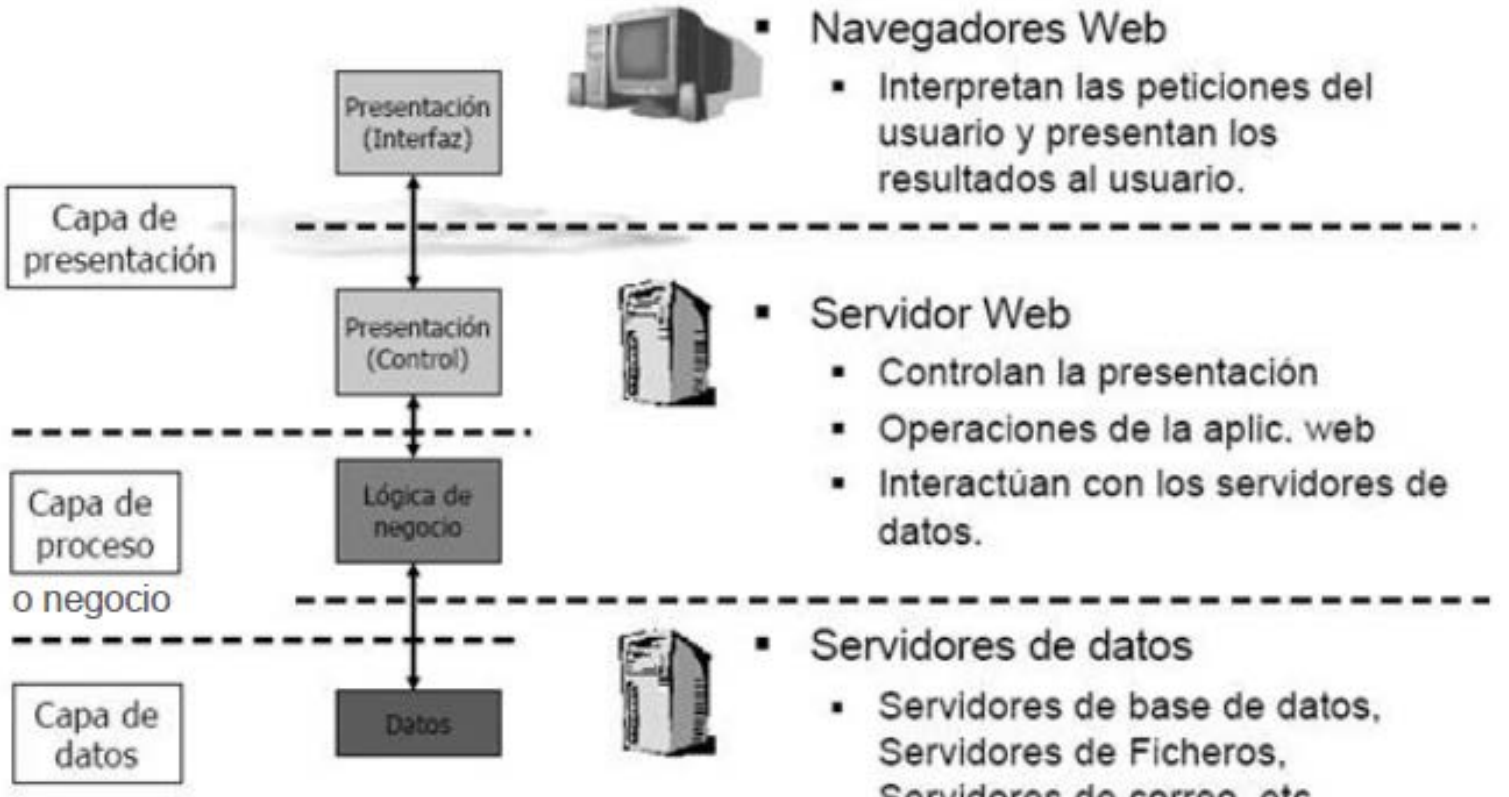
Modelo de 2 capas



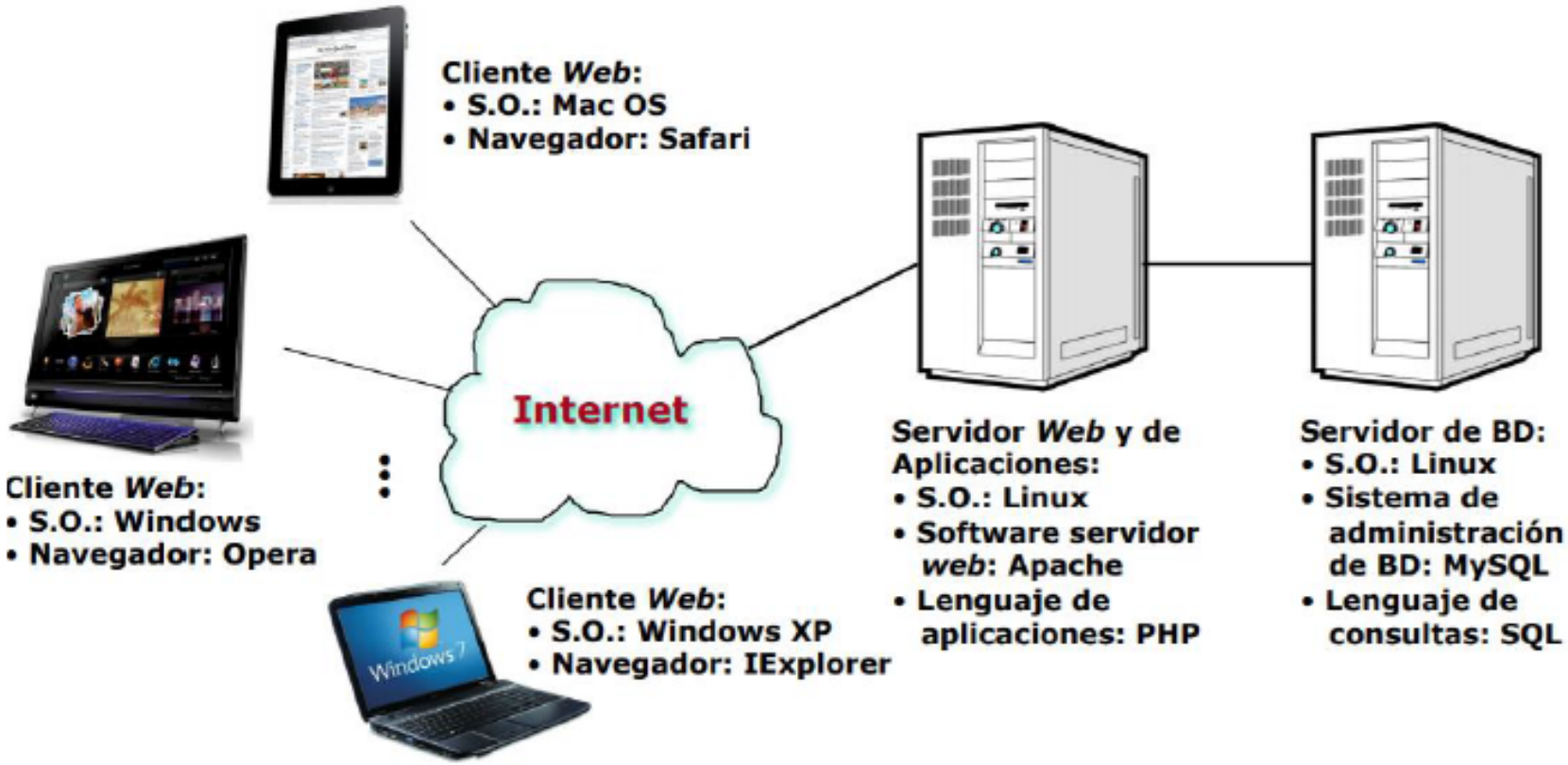
Modelo de tres capas



Modelos de 3 capas



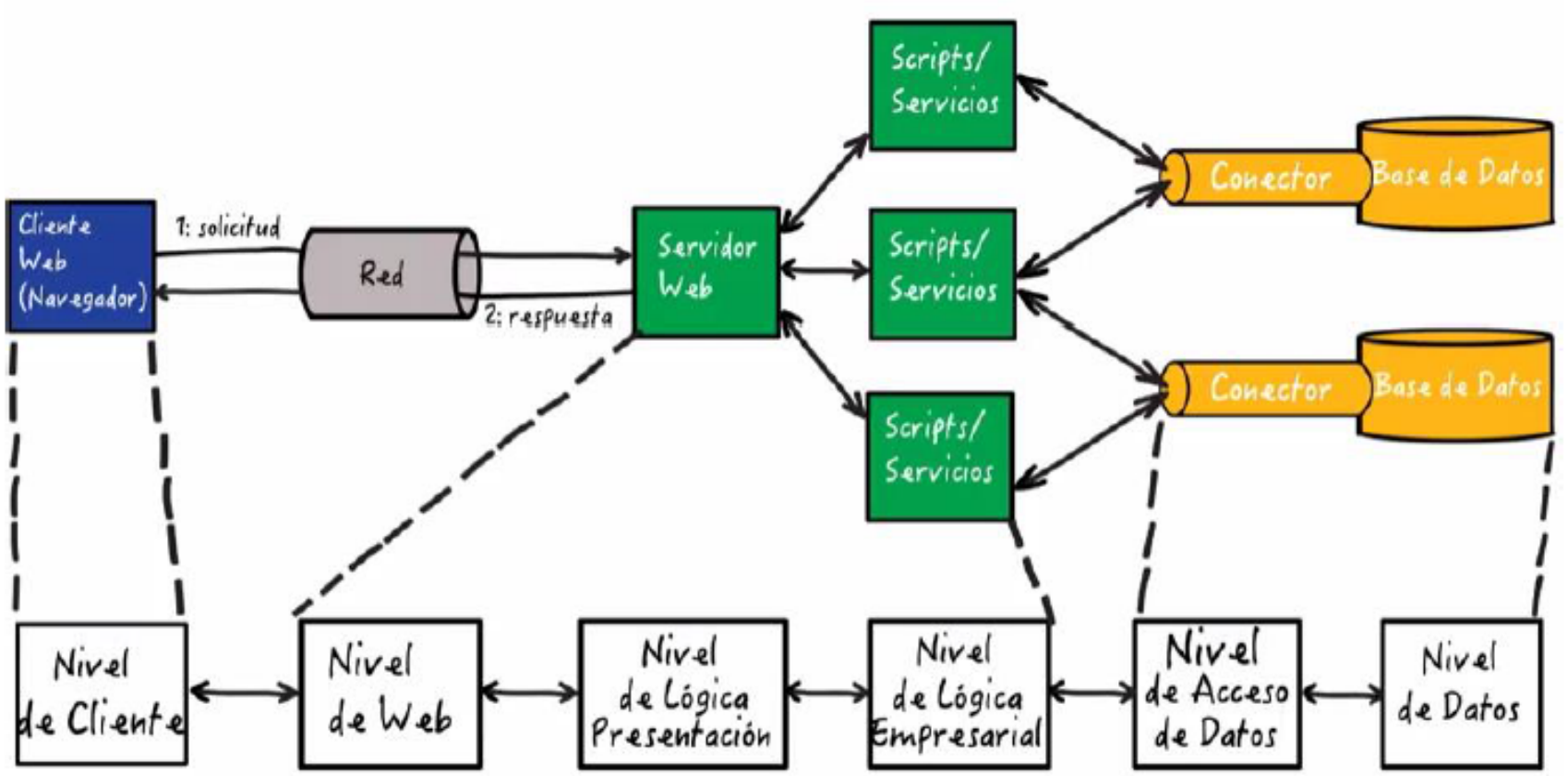
Ejemplo de modelo de 3 capas



Ventajas el modelo de 3 capas

- Un mayor grados de **flexibilidad**: priorización y gestión de peticiones.
- Mayor **seguridad**, ya que la seguridad puede ser definida independientemente para cada servicio y en cada nivel
- Mejor **rendimiento**, las tareas y datos se comparten entre servidores
- Su estructura modular le permite **recuperarse de fallos y errores** de manera más simple
- En definitiva incide positivamente en la implementación, mantenimiento, reutilización e interoperabilidad de la web
- No es obligatorio seguir dicha separación en capas aunque es bastante recomendable sobre todo si la aplicación es lo suficientemente grande

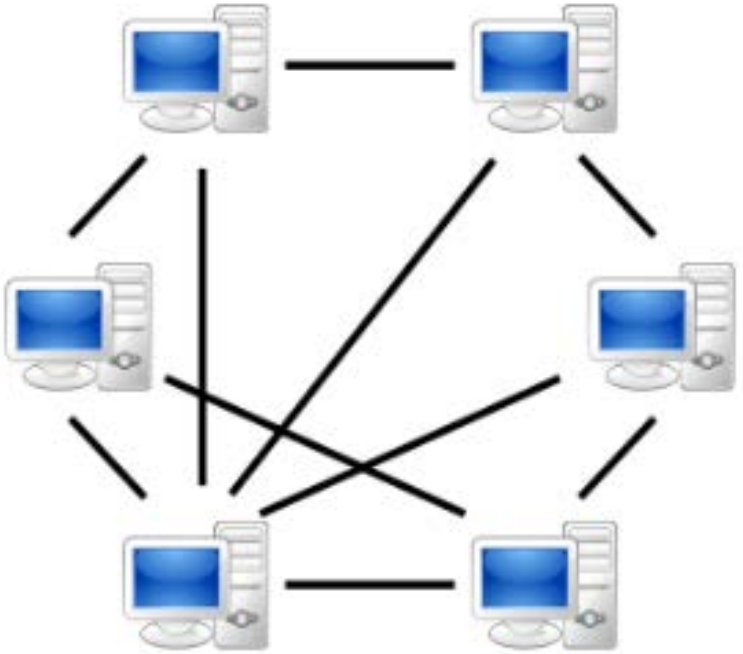
Arquitectura n-capas

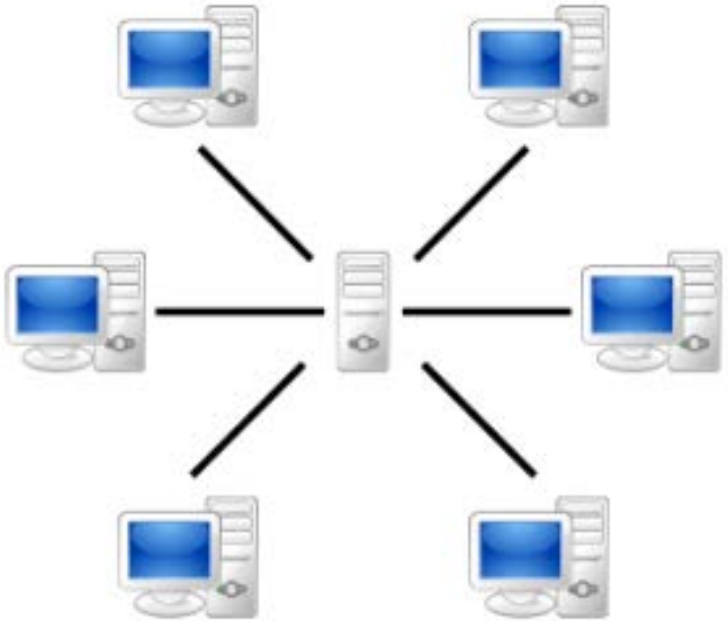


Modelos P2P

- Arquitectura **distribuida**, todos los nodos tienen los mismo privilegios y actúan como cliente y como servidor.
- El proceso se distribuye entre todos los nodos de la red.
- La primera red P2P famosa fue **Napster**
- **BitTorrent** y **criptomonedas** dos ejemplos de redes P2P actuales

C/S vs P2P





C/S vs P2P

Cliente-Servidor	P2P
Necesita un cliente y un servidor	Los equipos toman ambos roles el de cliente y el de servidor
El proceso depende de un servidor	El proceso corre sobre toda la red de ordenadores
El cliente se conecta a un servidor	El nodo se conecta a una red
Centralizado en un host	Redes descentralizadas con muchos nodos

C/S vs P2P

	P2P	Cliente-Servidor
Ventajas(+)	Facilidad para operar e instalar. Permite el intercambio directo de información en cualquier formato.	Centralización del control en el servidor. Mayor seguridad. Fácil mantenimiento.
Desventajas(-)	Menor seguridad. Sistema no centralizado.	Congestión de tráfico. Caída del servidor. Coste del servidor.

Modelos de Apps web

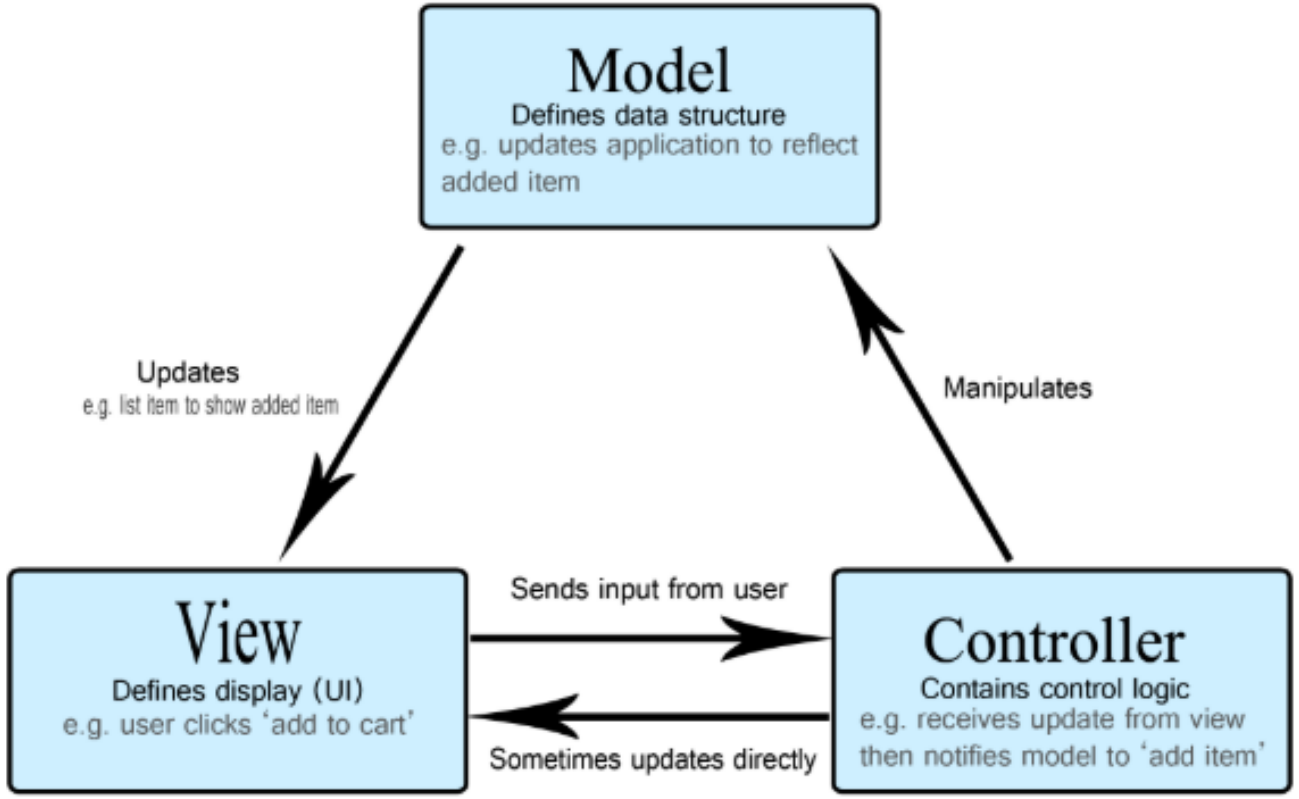
Estáticas

- El contenido esta **predefinido**. Cada vez que entras es el mismo hasta que webmaster no lo actualice.
- Utiliza solo **HTML** y **CSS**.
- Ejemplos: https://www.ugr.es/~borjamolina/

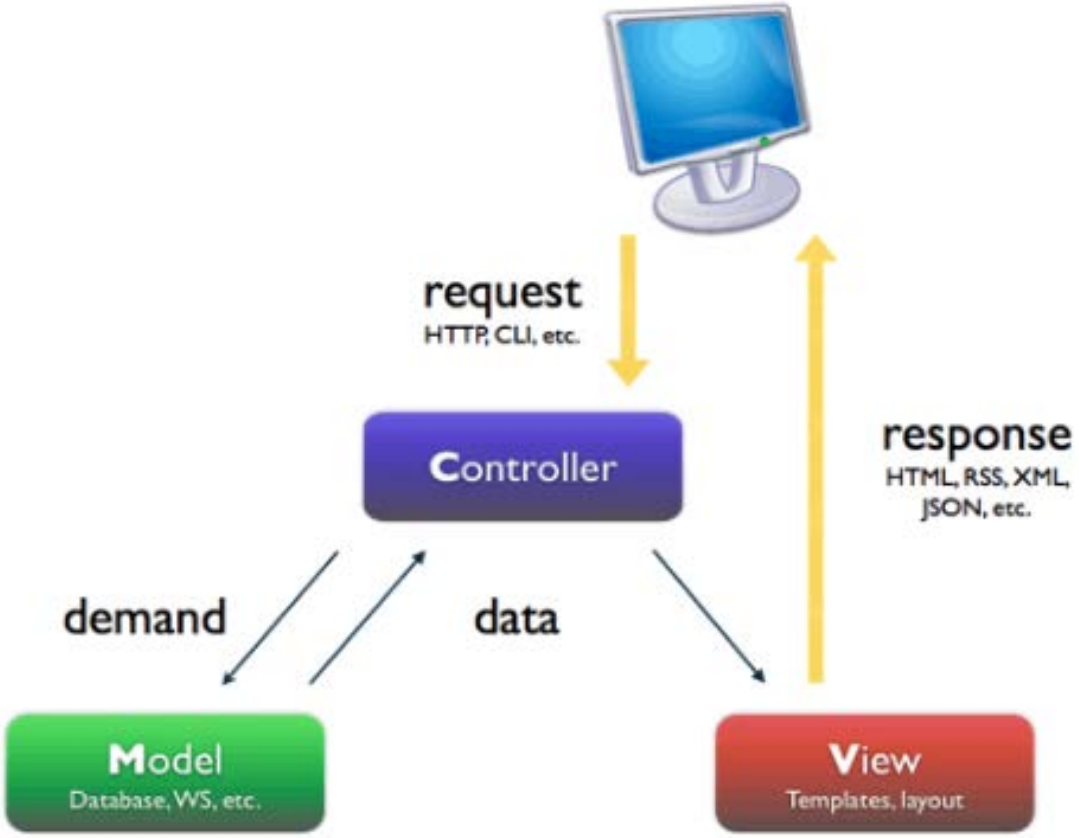
Dinámicas

- El contenido visualizado por el usuario puede cambiar de forma **dinámica** sin necesidad de que nadie lo actualice en el servidor.
- El servidor ha de contar con un lenguaje de programación como **PHP**, **ASP** o **JSP**.

Modelos de Apps web - MVC



Modelos de Apps web - MVC

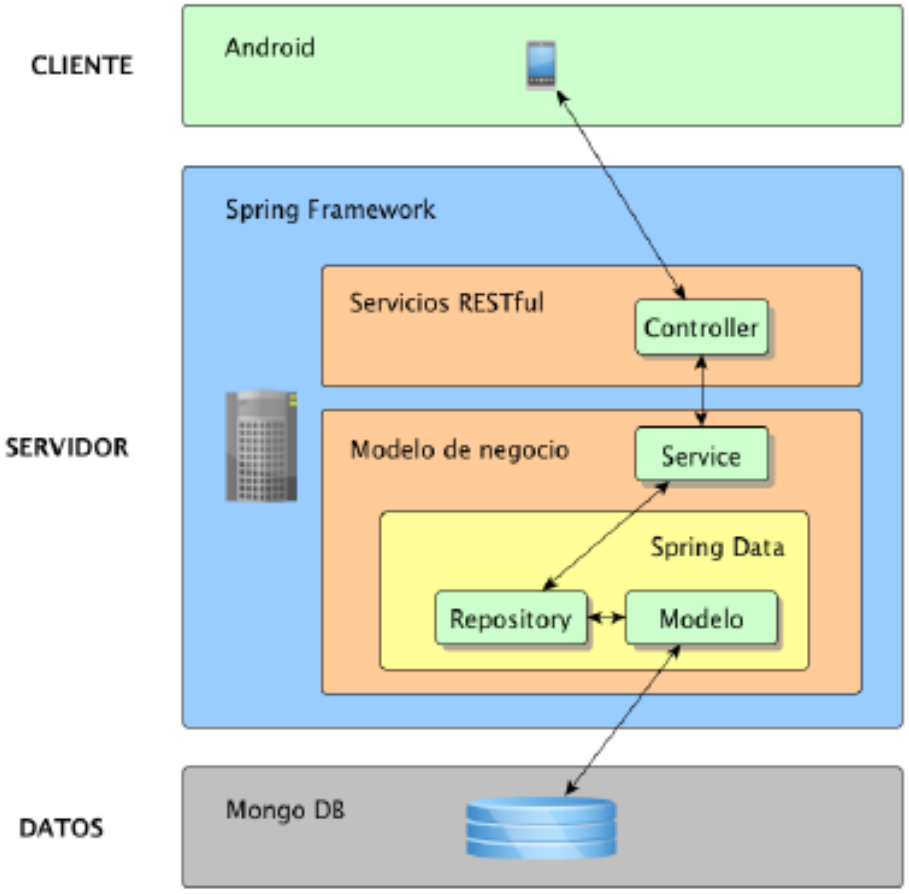


Actividad

- ¿Qué es un framework?
- Encuentra frameworks para el desarrollo web que sigan el patrón MVC

Modelo basado en Cloud Computing

- Es una evolución de **MVC** donde el acceso a datos está descentralizado
- Los datos son obtenidos en lenguajes de marcas y de listas también llamados NoSQL: **JSON** o **MongoDB**
- Las aplicaciones en la **nube** suelen usar este modelo



Actividad

- ¿Qué es el cloud computing?
- Investiga las bases de datos no relacionales. Cita un ejemplo. ¿Cuáles son las principales diferencias con las relacionales?
- ¿Qué BD se utilizan en cloud computing?

Plataformas web

- Entorno de desarrollo de software empleado para diseñar y ejecutar un sitio o aplicación web. Cuatro componentes básicos:
  - 1. El **sistema operativo**. En ocasiones limita la elección de otros componentes.
  - 2. El **servidor web**. Software que maneja las peticiones desde equipos remotos a través de internet.
  - 3. **Gestor de bases de datos**. Almacena y gestiona sistemáticamente datos que recopilamos y utilizaremos en nuestra App web
  - 4. Uno o varios **lenguajes de programación**. Controlarán el acceso a los datos y las acciones que el usuario pueda realizar.

Plataformas web

- En función de cómo se organice el software y las capas existen 3 tipos de plataformas web:
  - **Monolíticas**: Java-Sun
  - Software en **estructura apilada**
  - **Servicios web**

Plataformas monolíticas: Java-Sun

- Proporciona baja el estándar **Java** todo los componentes para desarrollar la aplicación
- Complejo intercambiar piezas del software
- Pensado **para grandes aplicaciones** por su complejidad y diferenciación de capas
- **Grandes requisitito** de las App Web
- **Poca interoperabilidad**

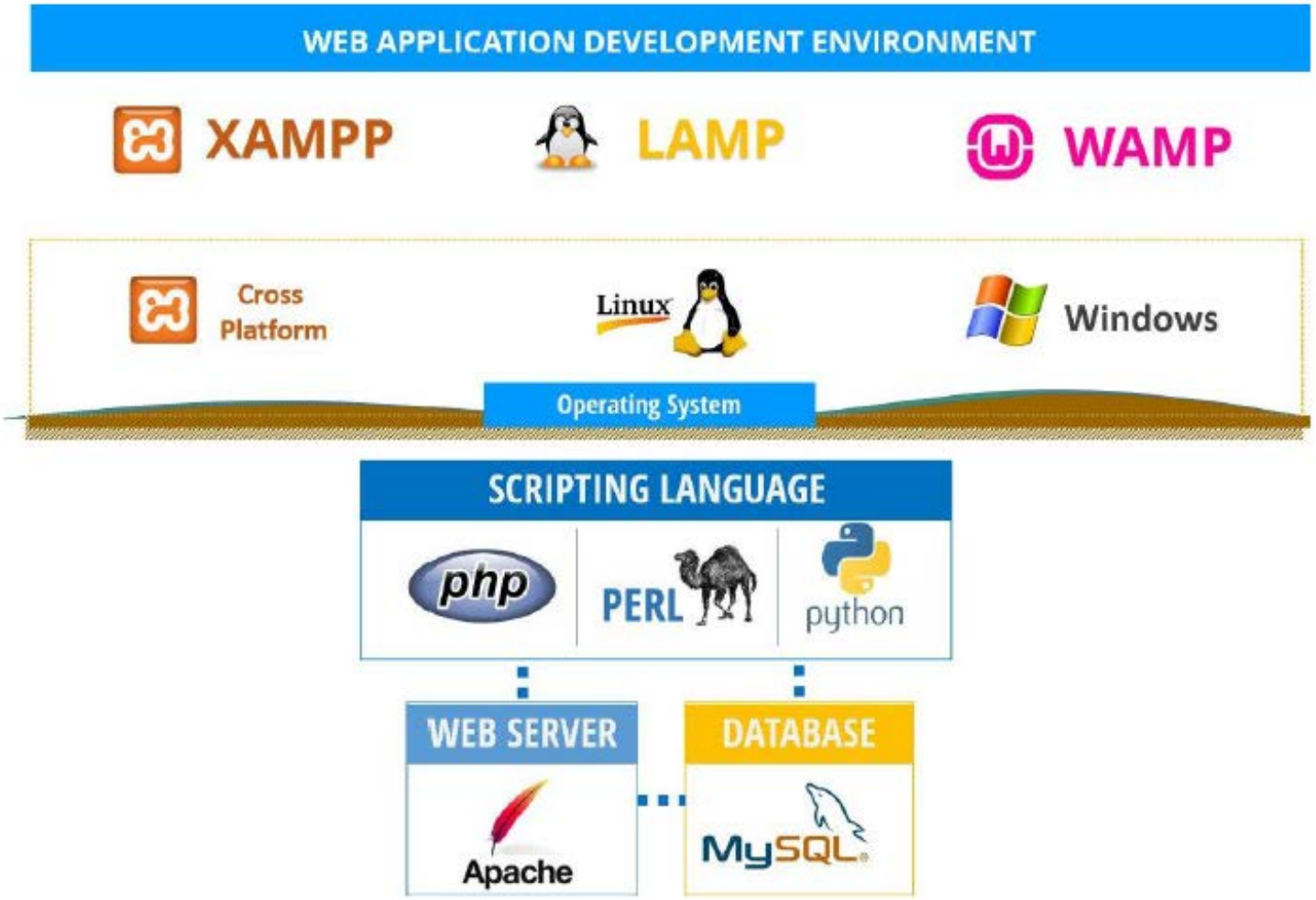
Plataformas monolíticas: Java-Sun

- Servidores web: Glassfish, **Tomcat**, WebLogic, JBoss
- La unidad básica de programación son los **Servlets**
- Facilita el desarrollo multiplataforma
- Móvil, escritorio, nube y web todo en una sola arquitectura

Plataformas de estructura apilada

- Son las más comunes hoy día.
- Requieren menos requisitos
- El código controla la aplicación como el despliegue de la misma.
- Los componentes se puede combinar dando lugar a **distintas alternativas**.
- **LAMP, WAMP, MAMP**.
- Sobre el **lenguaje interpretado** se suelen establecer **framework** para ayudar en el desarrollo:
  - Ruby on Rails
  - Django
  - Symfony

Plataformas de estructura apilada



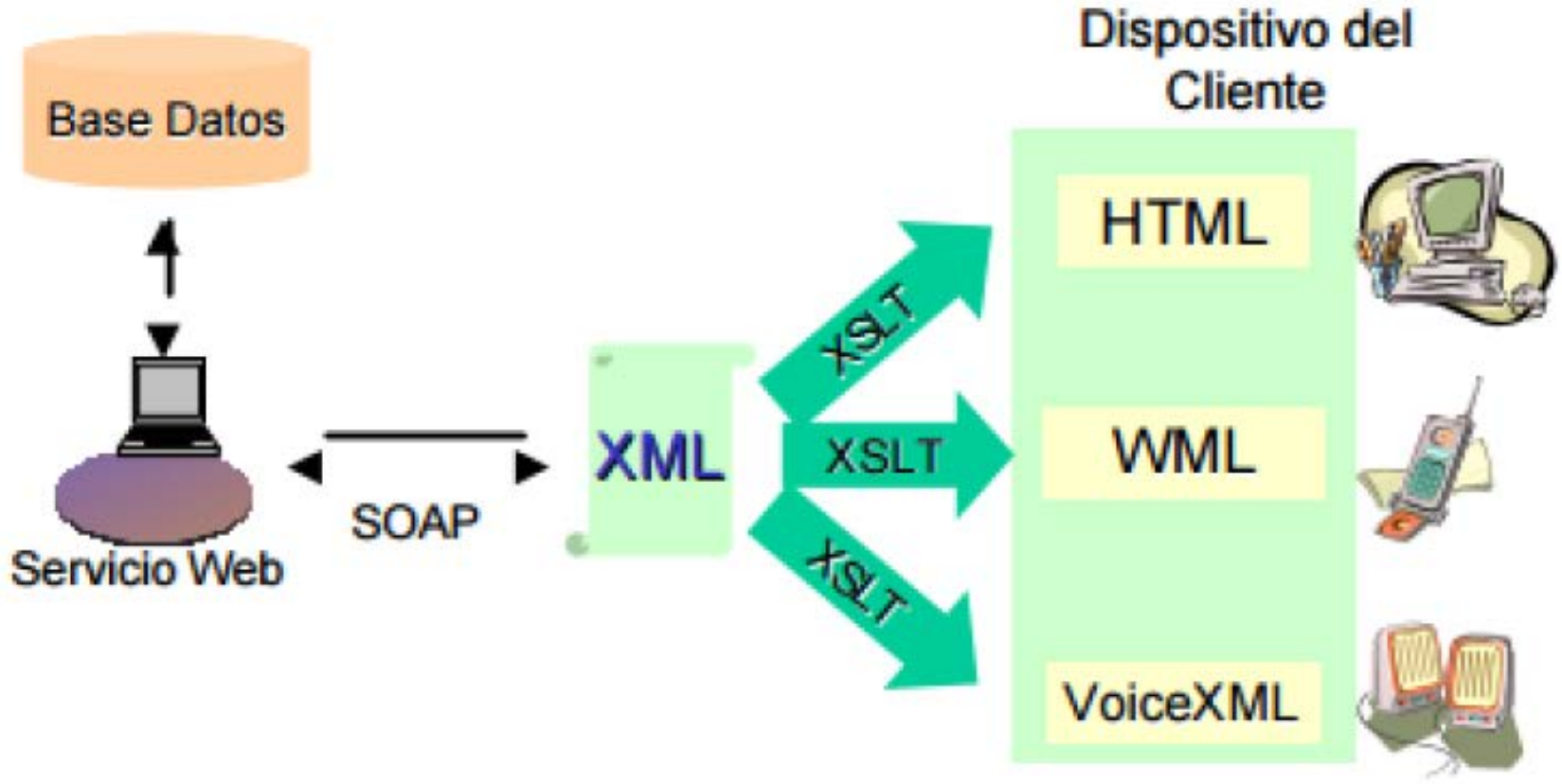
Servicios web

- Aplicaciones auto-contenidas, auto-descritas que poder ser publicadas, localizadas e invocadas a través de la web.
- Un servicio debe contener una descripción de sí mismo.
- Una vez desarrollados, otras aplicaciones y servicios Web pueden invocar su servicio dado.
- Dos tipo principales:
  - SOAP
  - RESTful

SOAP

- *Simple Access Object Protocol*
- Protocolo estándar de cómo dos objetos en diferente máquinas se comunican por intercambio de mensajes XML
- La arquitectura y formato del mensaje XML bien definido por el sublenguaje de XML llamado WSDL (Web Service Description Language)
- Interoperabilidad y seguridad HTTP
- Muy asociado a plataformas Java
- https://www.wsdl-analyzer.com/

SOAP


Base Datos
Servicio Web
SOAP
XML
XSLT
XSLT
XSLT
Dispositivo del Cliente
HTML
WML
VoiceXML

SOAP

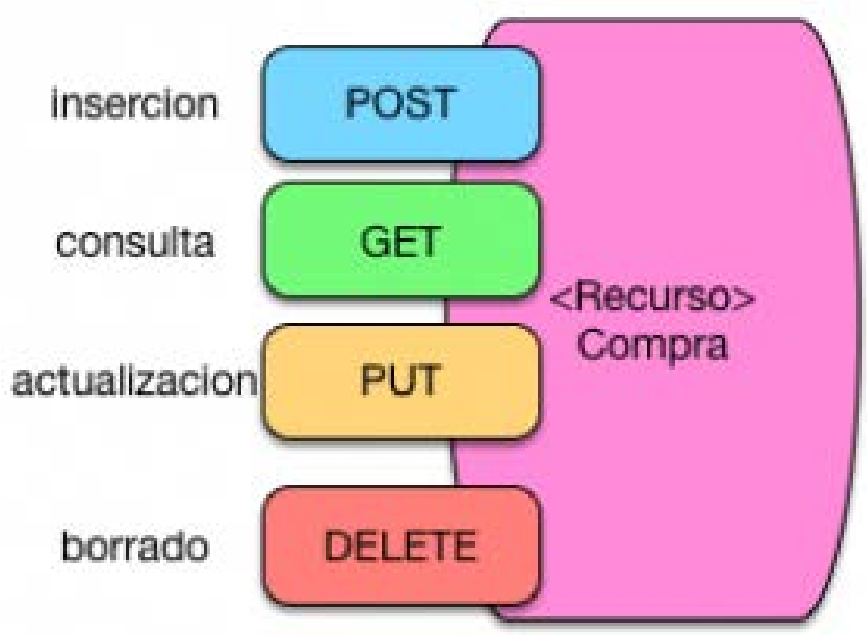
```
POST /Quotation HTTP/1.0
Host: www.xyz.org
Content-Type: text/xml; charset = utf-8
Content-Length: nnn

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding"

   <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotations">
      <m:GetQuotation>
         <m:QuotationsName>MiscroSoft</m:QuotationsName>
      </m:GetQuotation>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Servicios Web RESTful

- Siguen la arquitectura REST (Representational State Transfer) https://www.arquitecturajava.com/que-es-rest/
- Más simple que SOAP
- Permite realizar distintas acciones sobre el mismo recurso: POST, GET, DELETE, PUT
- Los servicios RESTful se acceden a través de una API RESTful bien definida en la que se indican las operaciones a realizar para acceder a recurso para su consulta o manipulación



Mensajes RESTful

- Los mensajes RESTful han de contener:
- **URI** del recursos que se quiere consultar o manipular. https://jsonplaceholder.typicode.com/users/1
- El **tipo** de la representación de dicho recurso. En la cabecera se indica el tipo de la respuesta. Los más comunes son JSON y XML → "Content-type": "application/json; charset=UTF-8“
- La acción a realizar: **GET, PUT, POST, DELETE**
- **Hipervínculos**: nuestra respuesta puede incluir hipervínculos hacia otras acciones que podamos realizar sobre los recursos.

Ejemplo de servicio RESTful

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo

Servicios RESTful

- Hoy en día muchísimas webs (Twitter, Facebook, meteomatics…) ofrecen servicios RESTful
- Sin embargo la mayoría requiere una API Key para poder utilizarlas, dicha Key puede ser obtenida (dependiendo de la API) registrándose, pidiendo formalmente acceso a ella o pagando.
- La web https://jsonplaceholder.typicode.com/ dispone de una API 'de juguete' con la que podemos aprender sin necesidad de pagar o de esperar a que nos den una Key.

Actividad

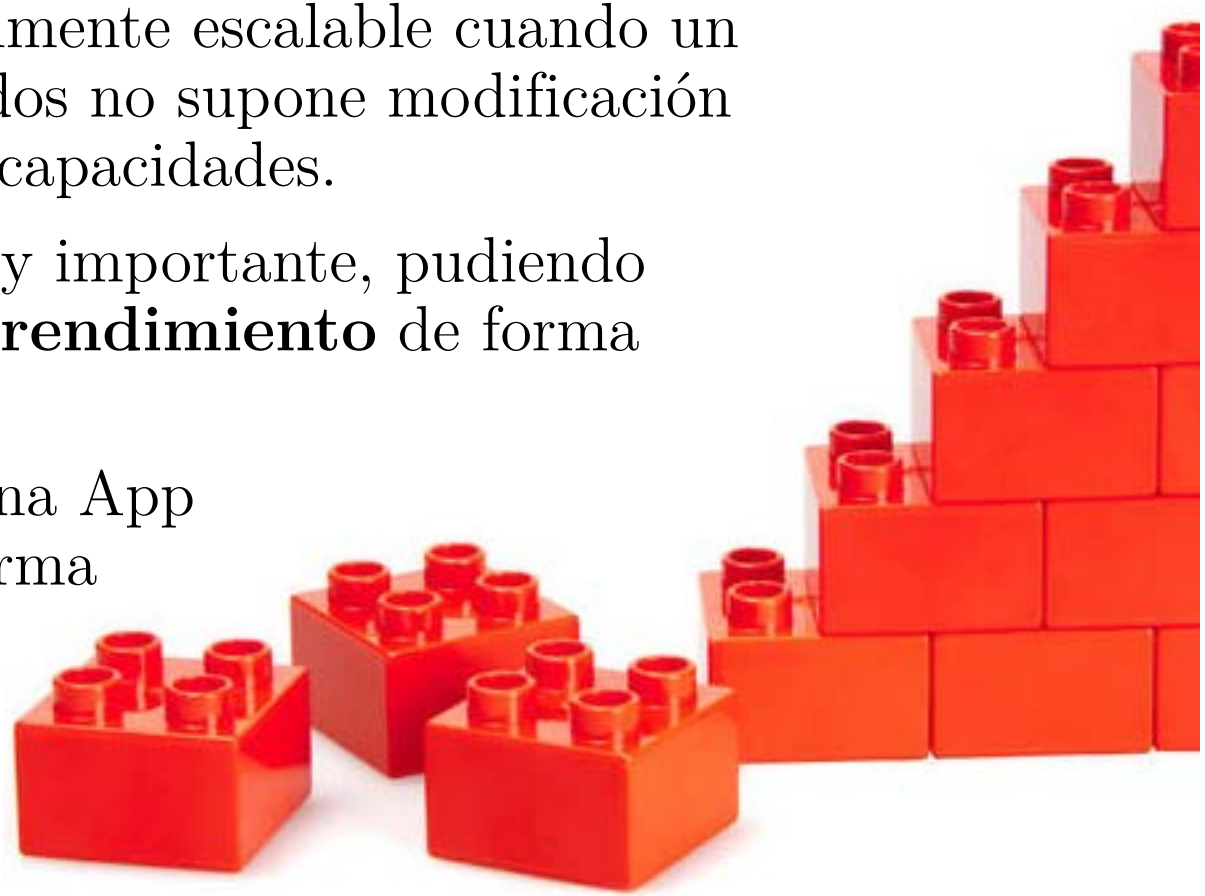
- Lee https://github.com/typicode/jsonplaceholder#how-to
- Con lo aprendido:
  - Realiza una consulta al recurso /users de https://jsonplaceholder.typicode.com/ ¿Cuál es el nombre del usuario número 10?
  - ¿Qué sucede si intentamos acceder al usuario número 11? ¿Por qué?
  - Borra el usuario 1 ¿Puedes acceder después de borrarlo? ¿Por qué?
- Infórmate acerca de la API de Twitter
  - ¿Qué es necesario para usarla?
  - ¿Qué acciones nos permite?

Escalabilidad

- La **escalabilidad** es la capacidad del software para **adaptarse** a las necesidades de **rendimiento** a medida que el número de usuario crece, las transacciones aumentan y la base de datos recibe más peticiones.
- Es un **propiedad deseable** de casi cualquier sistema, red o proceso
- Podemos resumirlo como la capacidad de un sistema para ser usado por **más usuarios** sin que su rendimiento se vea perjudicado.

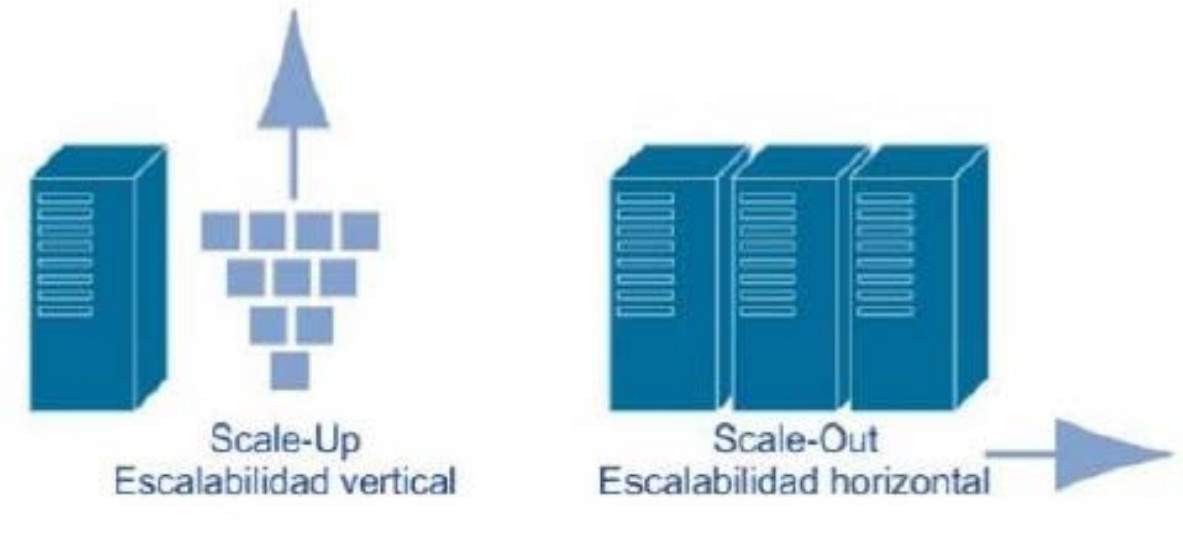
Escalabilidad

- Una **arquitectura web** es totalmente escalable cuando un aumento de los recursos solicitados no supone modificación alguna en su comportamiento o capacidades.
- La escalabilidad es un factor muy importante, pudiendo llegar a ser crítica. Influye en el **rendimiento** de forma significativa.
- La **cantidad de usuarios** de una App Web se puede incrementar de forma exponencial



Tipos de escalabilidad

- Escalabilidad **vertical**:
  - Mejorar o **añadir** hardware
  - **Optimizar** el software para ser más eficiente
- Escalabilidad **horizontal**:
  - **Añadir nodos** y que trabajen en **paralelo**
  - Añade el problema de balancear la carga y la comunicación entre los distintos nodos



Cómo escalamos

Scaling up - vertical

- Más CPU
- Más GPU
- Más memoria
- Sencillo
- limitado

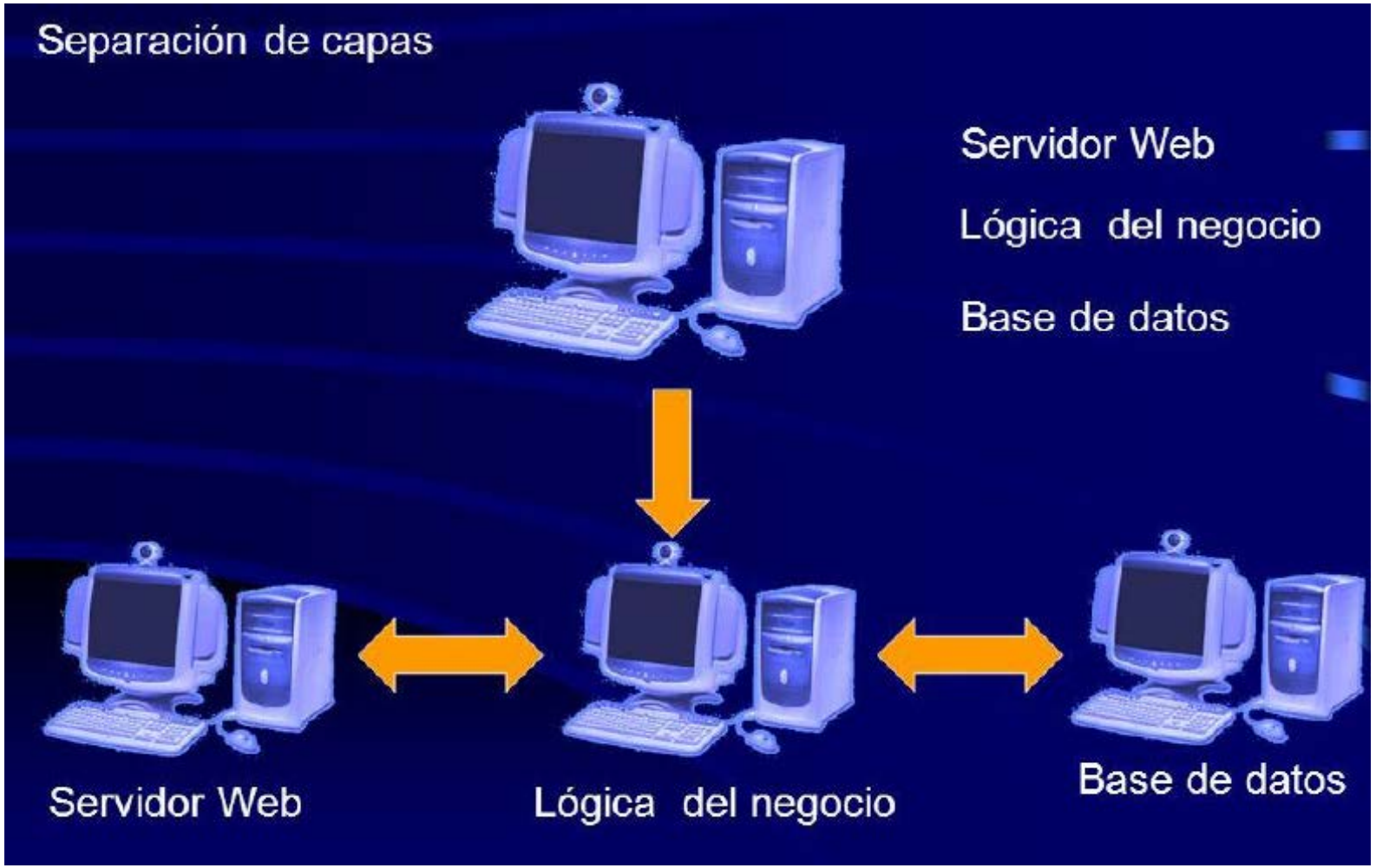
Scaling out - horizontal

- Aumentar el número de nodos, es decir más ordenadores
- Descentralizado
- Distribuir la carga de trabajo
- Complejo

Escalabilidad vertical

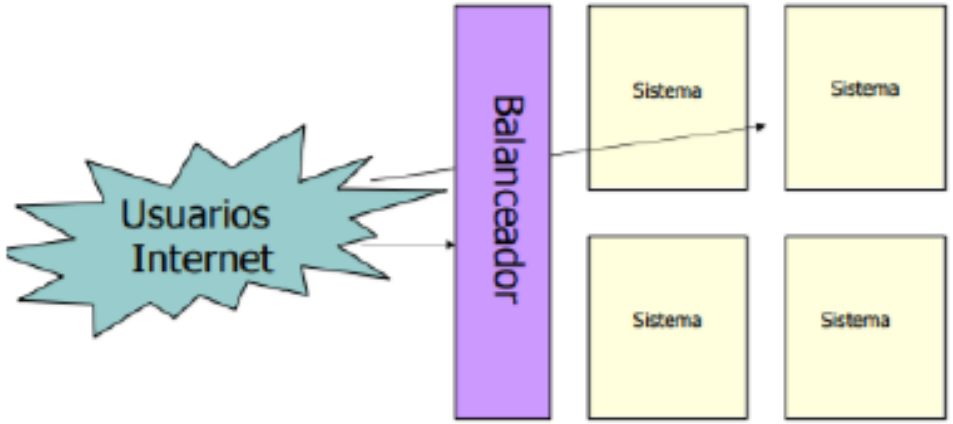
- La **separación lógica** entre capas se implementará de forma que permita la **separación física** de las mismas.
- Es necesario un **Middleware** entre capas que permite la **comunicación** remota.
- Está más **limitada** y poder ser **muy cara** según las prestaciones del hardware que necesitemos adquirir.
- Es mas **sencilla** de realizar que la escalabilidad horizontal

Escalabilidad vertical



Escalabilidad horizontal

- **Clonamos el sistema** (con sus capas)
- Los nodos **trabajan en paralelo** y **balanceamos** la carga.
- **Más flexible** que la vertical: adecuada para aquellas situaciones donde es difícil predecir los cabios en las necesidades.
- **Puedes ser más barato**: las maquinas usadas pueden ser de menor costo.



Escalabilidad horizontal