Arquitectura de Software

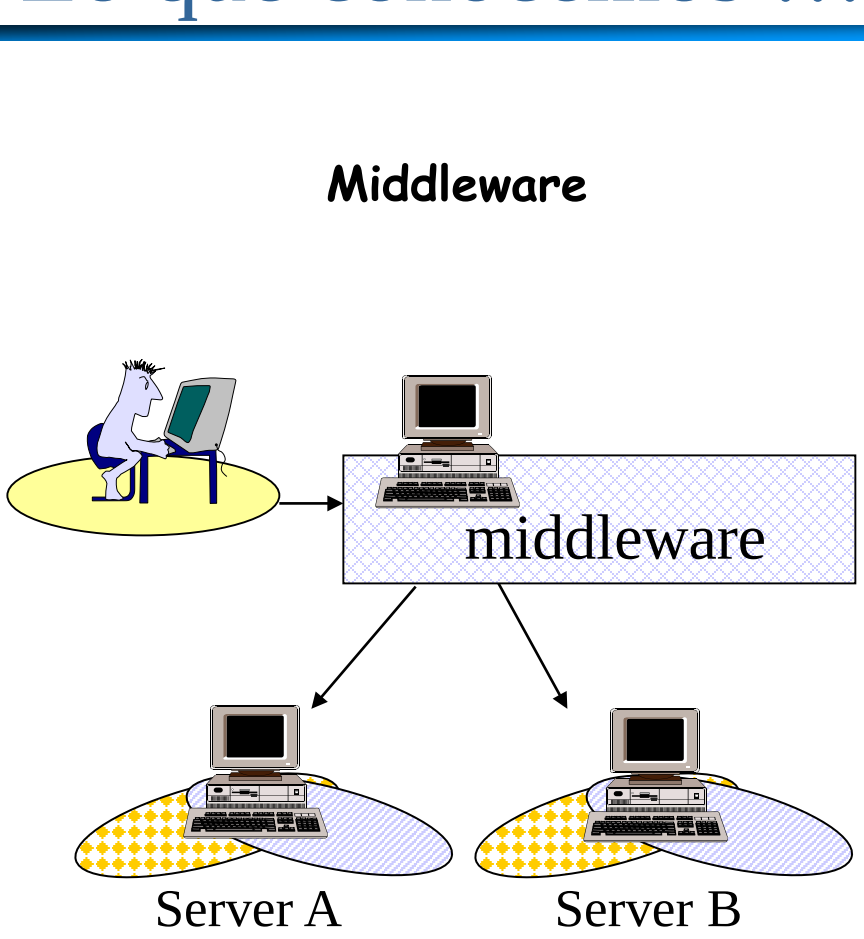
Cliente/Servidor

Créditos: Las imágenes que aparecen en este documento han sido realizadas por José Angel Bañares

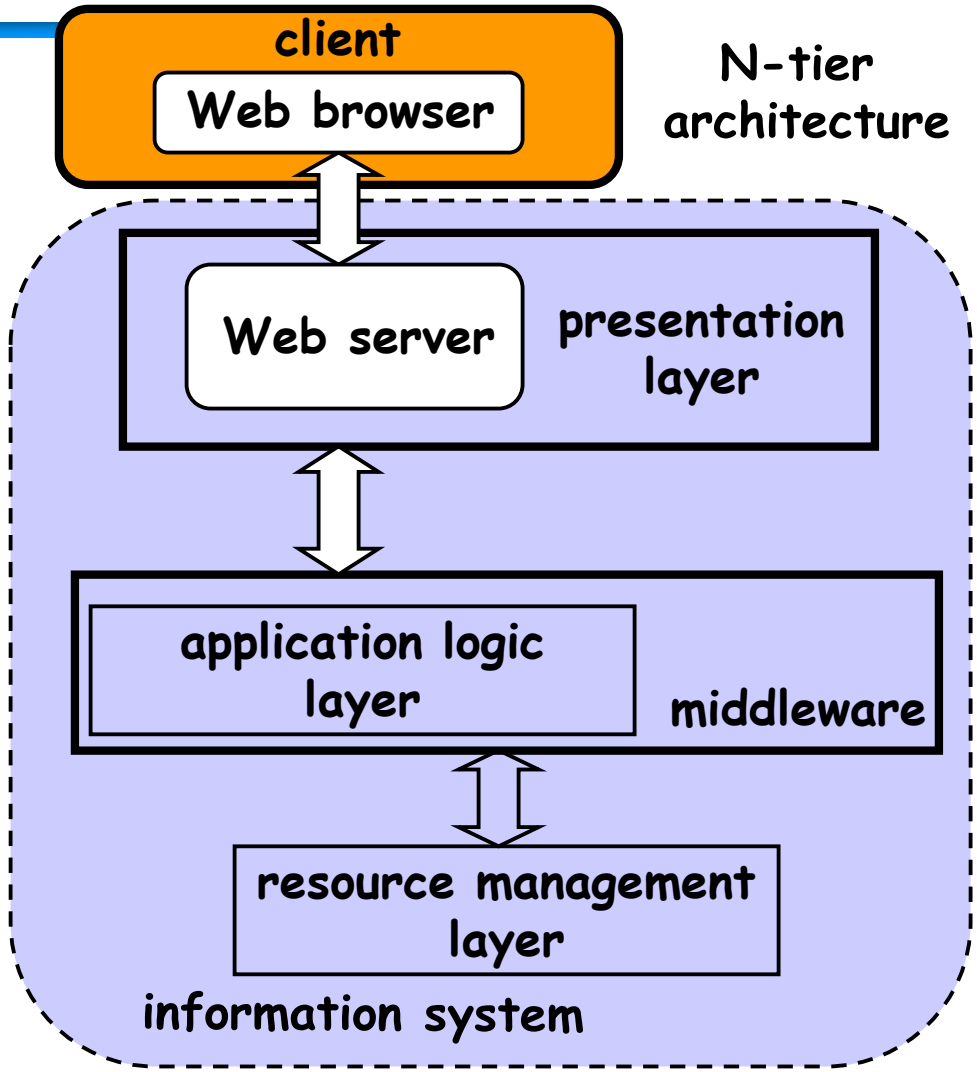
Contenidos

- Design of an information system
- Architecture of an information system
- Communication in an information system
- Tecnologías Web para dar soporte a sistemas de información
  - B2C
  - Applets
  - Programas CGI
  - Servlets
  - Servidores de aplicaciones

Lo que conocemos …



intra-enterprise application integration



inter-enterprise application integration

Internet

- En 1969 se crea la primera red, ARPANET. Conecta computadoras autónomas de diferentes Universidades americanas
- ARPANET desarrolla los protocolos necesarios
- TCP (*Transmission Control Protocol*)
  - Gestiona la transformación entre mensajes y paquetes de datos
- IP (*Internet Protocol*)
  - Gestiona el direccionamiento de los paquetes a través de las Redes
- TCP + IP define la tecnología para **Internet**
  - Envío de paquetes a través de múltiples redes usando múltiples estándares, *e.g.*, Ethernet, FDDI, X.25
- Internet es un sistema global de redes de computadoras

Antes de la Web

- Estándares para intercambiar información a través de Internet
  - Telnet
  - SMTP (*Simple Mail Transfer Protocol*)
  - Dos formas diferentes de conectar "**cuentas**", independientemente del sistema operativo y las computadoras
- SMTP fue extendido con MIME (*Multi-purpose Internet Mail Extensions*)
  - Soporta audio, video, imágenes, etc.
- FTP (*File Transfer Protocol*)
  - Publicar ficheros en un servidor FTP
  - Transferencia de ficheros entre diferentes computadoras
  - Permite el modo "**anónimo**", aunque con autenticación. Elimina la necesidad de cuentas en todos los sistemas donde quiero conectarme
  - Permitió el desarrollo de sistemas "Web-like": Archie y Gopher

Web (World Wide Web)

- Evolución de las tecnologías anteriores (Internet)
  - HTTP, HTML, Servidores Web, Browsers
- Objetivo inicial de la **Web**
  - Compartir información utilizando Internet
  - Compartir y enlazar documentos
  - Protocolo HTTP, documentos HTML
- La **Web** rápidamente evolucionó para convertirse en un medio para:
  - Conectar clientes remotos con aplicaciones usando Internet
  - Integrar aplicaciones a través de Internet

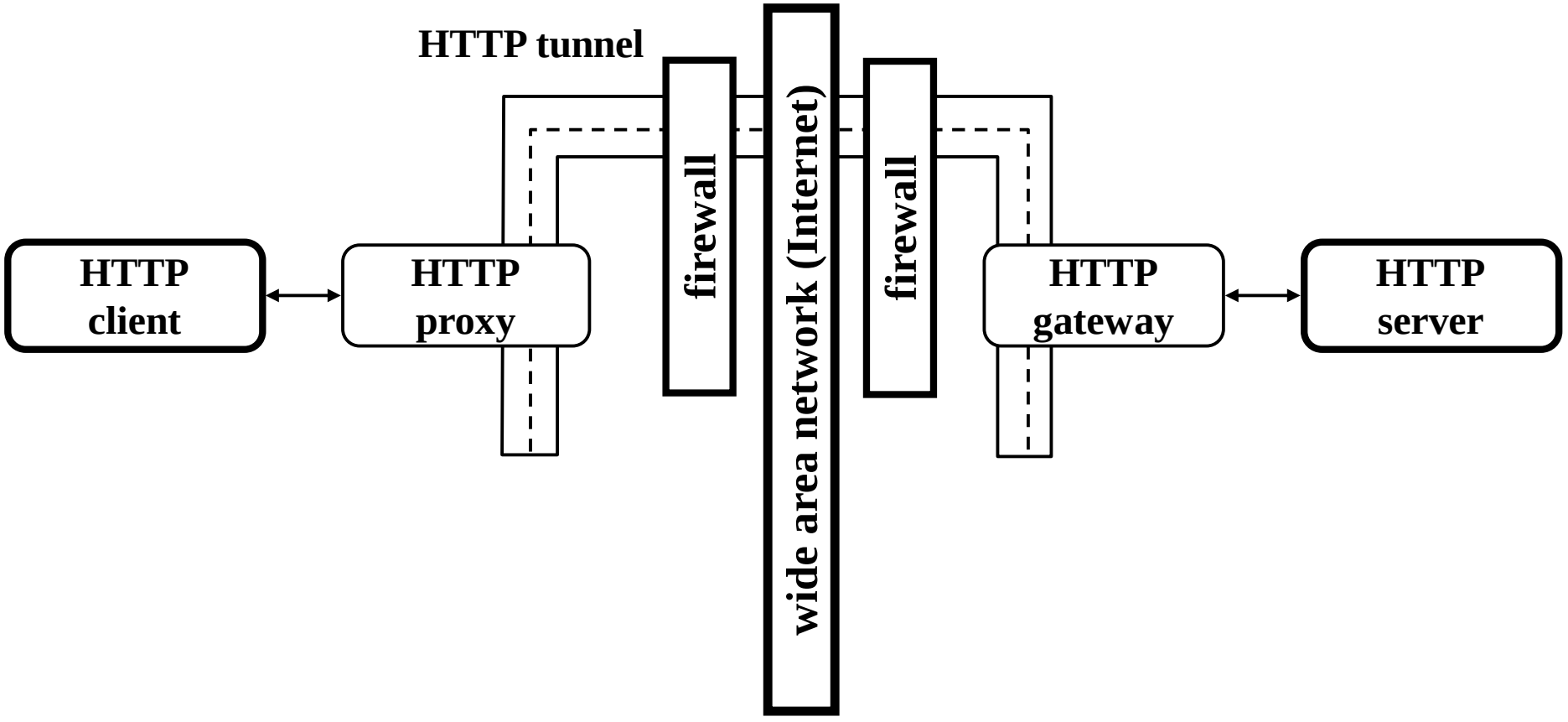
HTTP

- HTTP (HyperText Transfer Protocol)
  - Dice cómo transferir ficheros a través de la Red
  - Protocolo genérico porque da soporte a otros protocolos, e.g., FTP y SMTP
  - Soporta hipertexto → HTML (HyperText Markup Language)
    - Interconectar documentos insertando *links* entre ellos como parte del contenido del documento
    - Define un conjunto de indicadores (marcas) que indican cómo el browser debe mostrar el texto y las imágenes en una página web
  - La información se intercambia con documentos: estáticos o dinámicos (generados en tiempo de acceso). Identificados con URIs
  - Cada *recurso* tiene una URL que identifica su localización (*fichero* si es un documento, *programa* que producirá el documento)
  - Está basado en C/S. Usa sockets TCP/IP

HTTP

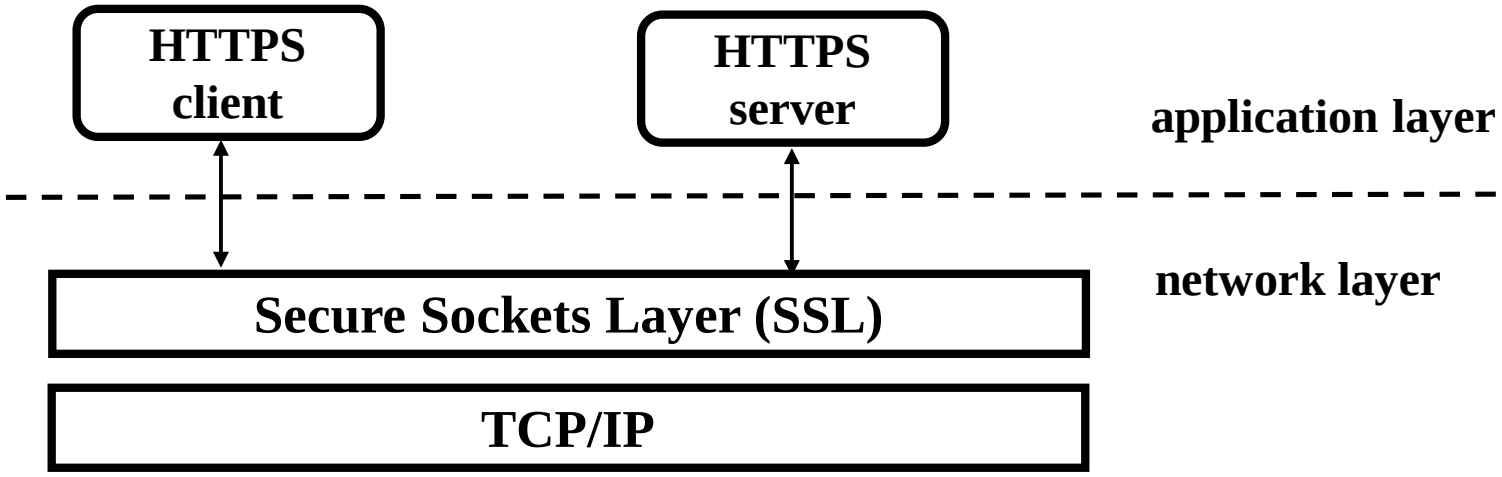
- HTTP (HyperText Transfer Protocol)
  - Funcionamiento
    - El cliente HTTP (Browser) abre la conexión con el servidor HTTP (web Server)
    - Envía la petición: request method, URI, versión del protocolo, mensaje MIME-like
    - El servidor envía la respuesta: estado, mensaje MIME-like
  - Request method
    - OPTIONS, GET, POST, PUT, DELETE

HTTP


HTTP tunnel
HTTP client
HTTP proxy
firewall
wide area network (Internet)
firewall
HTTP gateway
HTTP server

HTTP

- HTTP no encripta la información
  - Con un *sniffer* pueden leer la información que enviemos
  - Solución: SSL, protocolo que utiliza encriptación sobre TCP/IP
  - El cliente y el servidor Web establecen una conexión encriptada



HTTP

- HTTP no tiene estado
  - No proporciona soporte para *almacenar información* a través de diferentes transacciones HTTP
  - No hay relación entre dos peticiones aunque vayan dirigidas al mismo cliente por el mismo servidor
  - El programador debe almacenar y gestionar la información de estado
    - Cookies: mantener *la información de estado* que sea necesaria entre diferentes llamadas

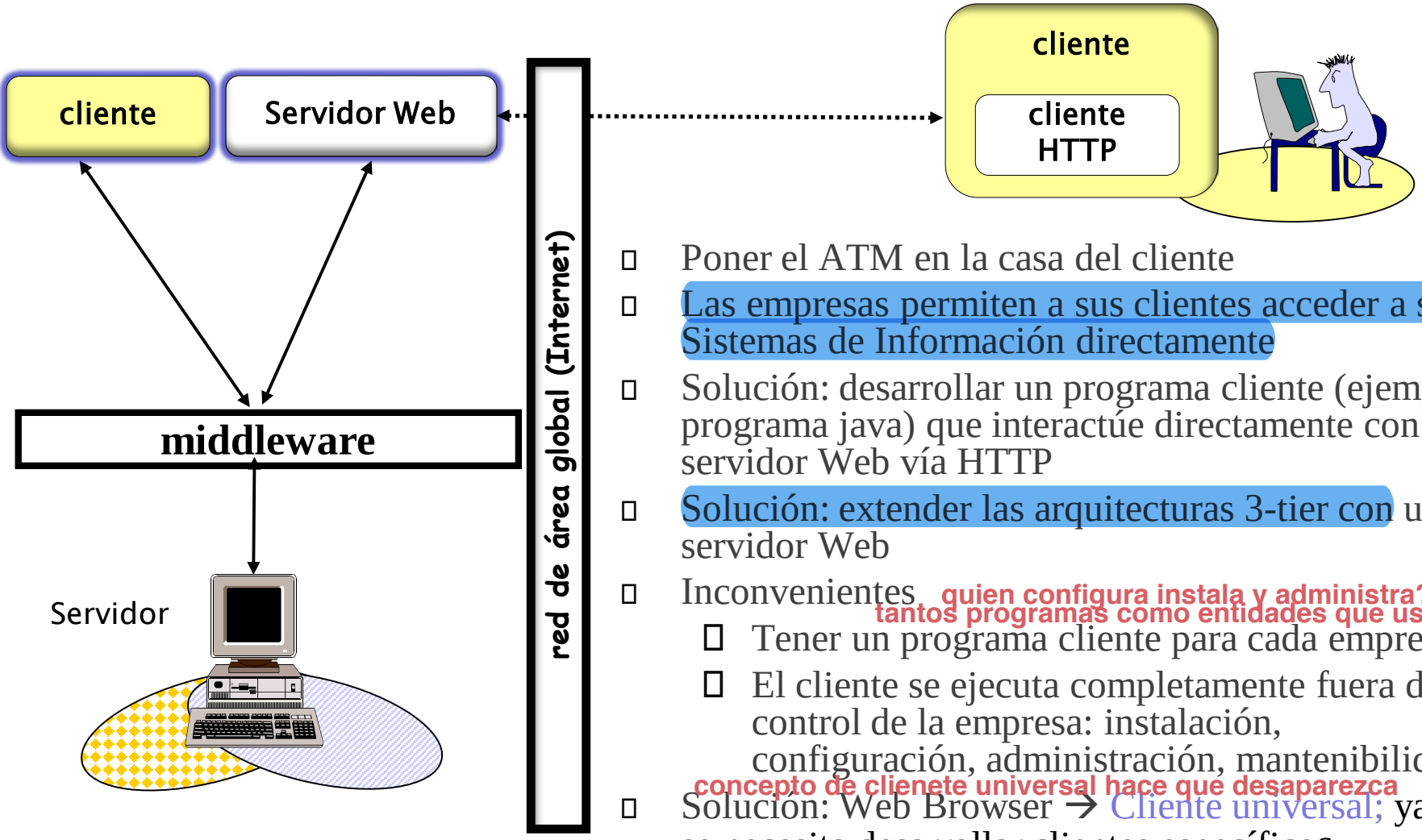
Visión

- Las arquitecturas 3-tier y el middleware “convencional” estaban diseñados fundamentalmente para:
  - Operar dentro de la empresa
  - Integrar SI dentro de la misma empresa (**intra-enterprise application integration**)
  - Problema:
    - El middleware era propietario y caro
- ¿Por qué no abrir los Sistemas de Información?
  - Escenario 1 (abrirlos a otras empresas):
    - **Inter-entreprise application integration**
  - Escenario 2 (abrirlos a otros usuarios, a los clientes)
    - Ejemplo Entidad bancaria

Solución → Usar las tecnologías Web

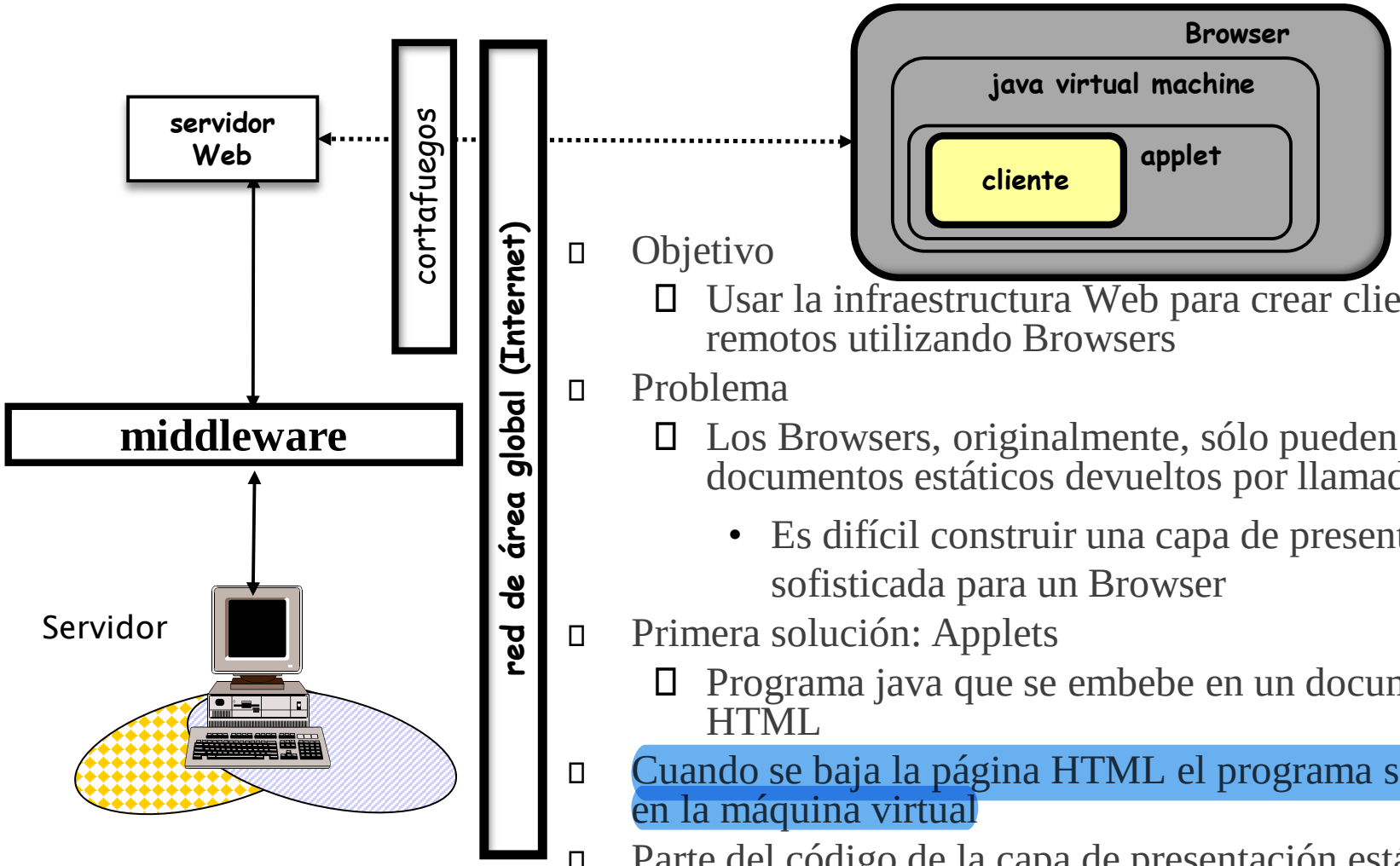
- Si encapsulamos ("*wrapping*") los SI locales y exponemos su capa de presentación con documentos HTML podríamos aprovechar las tecnologías Web para tener clientes distribuidos por Internet
- Objetivo inmediato fue
  - Crear clientes remotos → Mover la capa de presentación a cualquier parte del mundo
- Otra consecuencia
  - Integración de SI de diferentes empresas utilizando Internet → tecnologías Web (Futuro: utilizar la tecnología de los "Servicios Web")

Primera solución: Business-to-Consumer (B2C)



- Poner el ATM en la casa del cliente
- Las empresas permiten a sus clientes acceder a sus Sistemas de Información directamente
- Solución: desarrollar un programa cliente (ejemplo, programa java) que interactúe directamente con el servidor Web vía HTTP
- Solución: extender las arquitecturas 3-tier con un servidor Web
- Inconvenientes quien configura instala y administra? tantos programas como entidades que uses
  - Tener un programa cliente para cada empresa
  - El cliente se ejecuta completamente fuera del control de la empresa: instalación, configuración, administración, mantenibilidad, ..
- concepto de clienete universal hace que desaparezca
- Solución: Web Browser → Cliente universal; ya no se necesita desarrollar clientes específicos

Applets



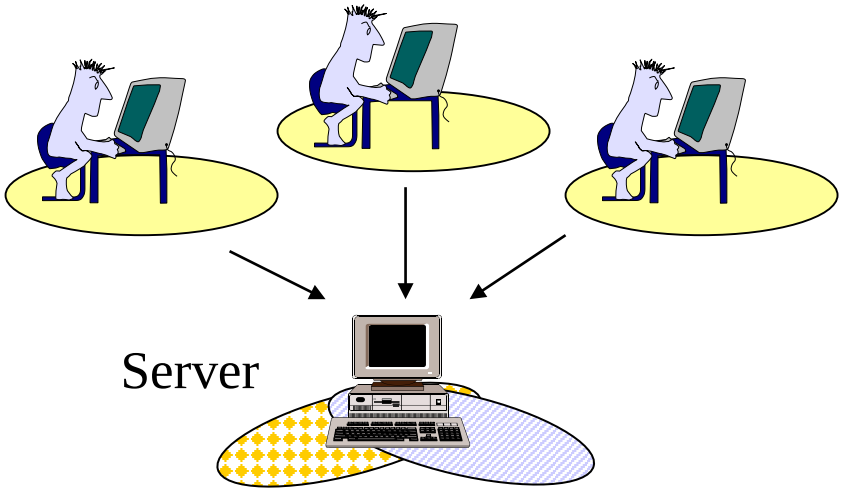
- Objetivo
  - Usar la infraestructura Web para crear clientes remotos utilizando Browsers
- Problema
  - Los Browsers, originalmente, sólo pueden mostrar documentos estáticos devueltos por llamadas HTTP
    - Es difícil construir una capa de presentación sofisticada para un Browser
- Primera solución: Applets
  - Programa java que se embebe en un documento HTML
- Cuando se baja la página HTML el programa se ejecuta en la máquina virtual
- Parte del código de la capa de presentación está en el applet

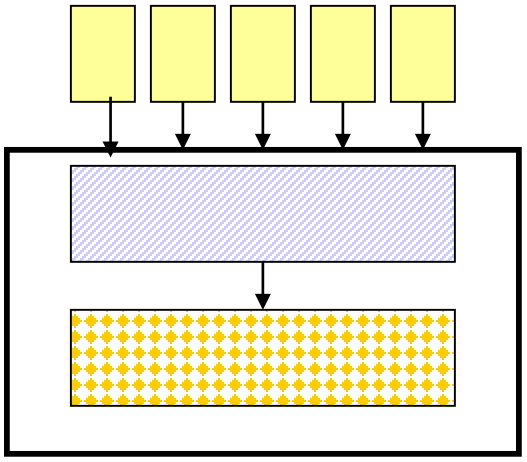
Applets - Discusión

- Ventaja
  - Convierten a un Browser en un cliente específico de un sistema de información *sin necesidad* de un procedimiento de instalación y configuración
- Inconvenientes
  - Existen de forma transitoria, durante la ejecución del Browser
    - Cada vez que el cliente se usa hay que bajarse el código del applet
- Consecuencia
  - No son adecuados para dar soporte a:
    - Clientes complejos → Documentos estáticos ni clientes "pesados"
    - Interacciones frecuentes
- Alternativa
  - Desarrollar un cliente (e.g., un programa java) que no se ejecute en un Browser pero que interactúe con el servidor Web vía HTTP → B2C
  - No tengo la limitación del Applet pero debo desarrollar el cliente
  - Estos sistemas son idénticos a las arquitecturas 2-tier …

Applets – Discusión …

- Estos sistemas son idénticos a las arquitecturas 2-tier … excepto que ahora la infraestructura de comunicación no la proporciona un middleware específico sino la Web: cambiamos llamadas RPC por llamadas HTTP



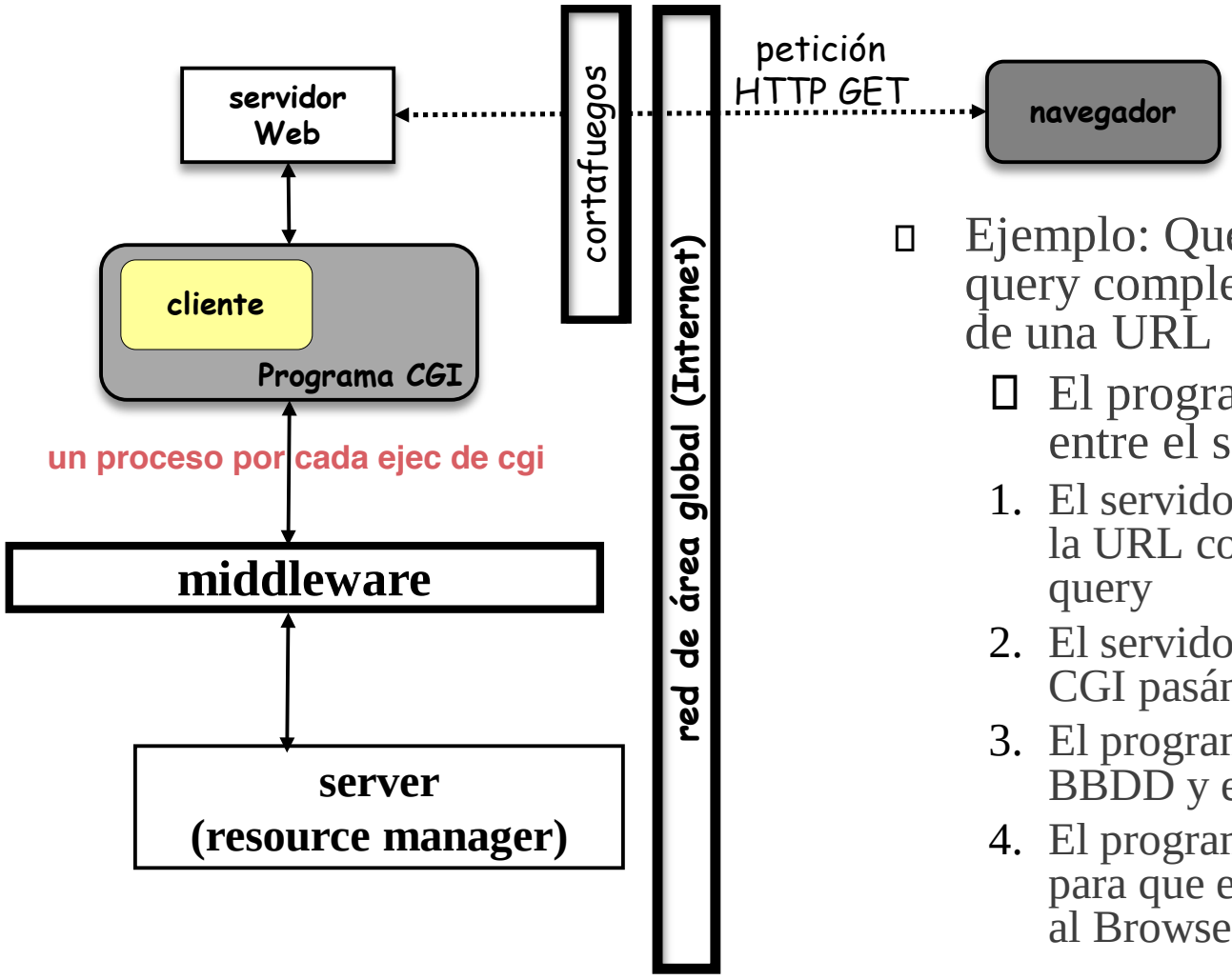


2-tier architecture

Programas CGI

- Problema: El servidor Web sólo retornaba contenido estático quizá con applets
- ¿Queremos que los servidores Web sirvan como interfaces con los Sistemas de Información?
  - Entonces deben servir contenidos de fuentes dinámicas (e.g., una BBDD)
  - ¿Cómo podría el servidor Web responder a una petición (lanzada a través de una URL) invocando a una aplicación que automáticamente genera un documento?
    - Programas CGI y Servlets.
- Common Gateway Interface (CGI), programas que interactúan con el servidor Web
  - Actúan como "gateways" hacia el SI local
  - Se sitúan en un directorio especial para que el servidor Web los identifique como programas y no como contenido estático
  - Escritos en cualquier lenguaje de programación
- ¿Cómo funciona CGI?
  - Asigna programas a URLs
  - Cuando se invoca la URL el servidor Web ejecuta el programa
  - Los argumentos del programa son parte de la URL. El servidor Web los pasa al programa
  - Se ejecuta el programa CGI como un proceso separado
  - El programa CGI puede interactuar con la lógica de negocio

CGI: Funcionamiento



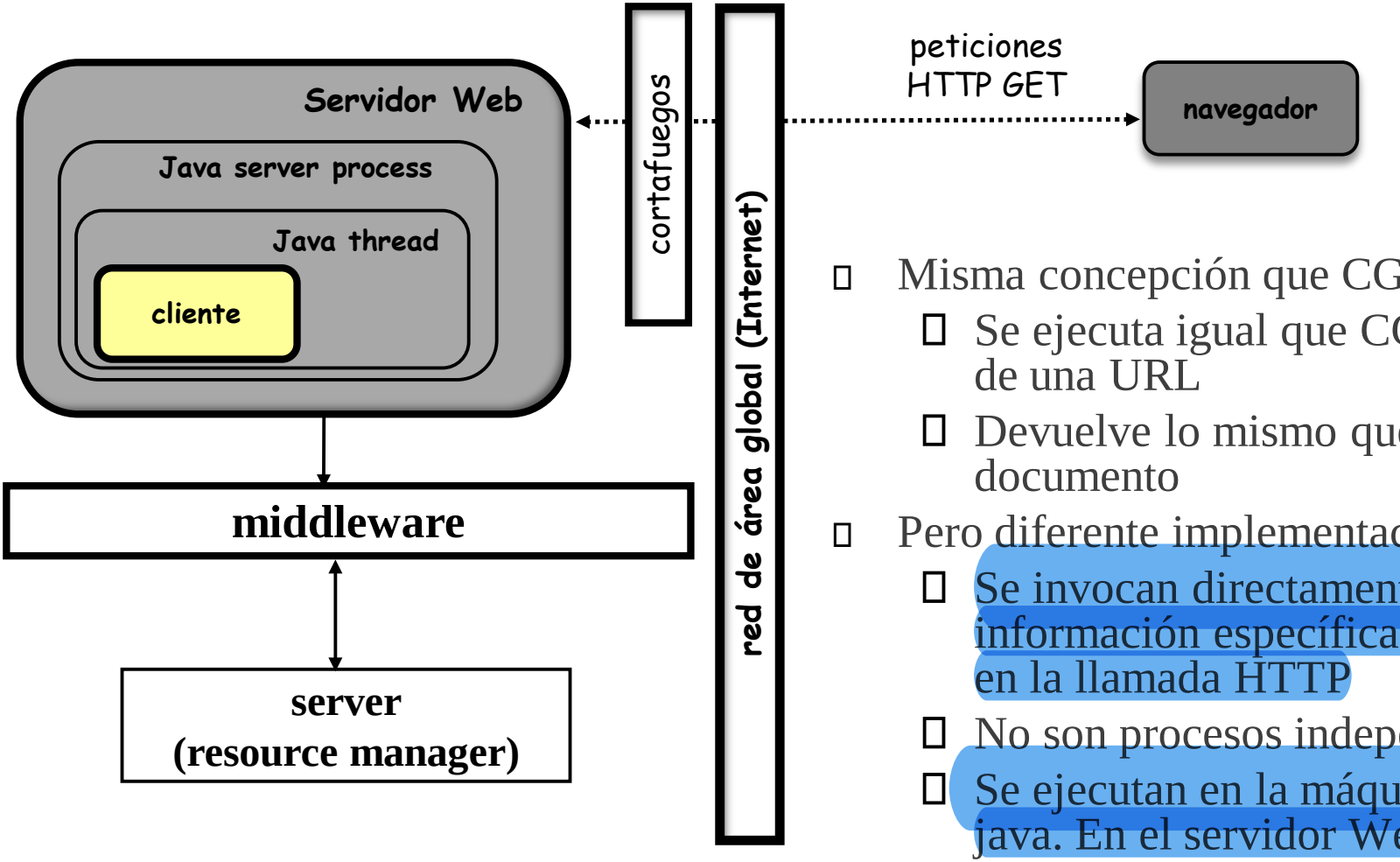
- Ejemplo: Queremos someter una query compleja a una BBDD a través de una URL
  - El programa CGI es la interfaz entre el servidor Web y la BBDD

1. El servidor Web recibe la petición de la URL con los parámetros de la query
2. El servidor Web ejecuta el programa CGI pasándole los parámetros
3. El programa CGI hace de cliente de la BBDD y ejecuta la query
4. El programa empaqueta los resultados para que el servidor Web los devuelva al Browser en una página HTML

Programas CGI - Inconvenientes

- Tiempo de respuesta
  - Se crea un proceso por instancia (tiempo de crearlo + cambio de contexto en el sistema operativo para pasar el control al programa CGI)
- Escalabilidad
  - Múltiples peticiones → Múltiples procesos (compitiendo por memoria y por BBDD)
  - Múltiples peticiones al mismo CGI → Se carga en memoria una vez por petición

Servlets



- Misma concepción que CGI
  - Se ejecuta igual que CGI, a través de una URL
  - Devuelve lo mismo que CGI, un documento
- Pero diferente implementación
  - Se invocan directamente: se pasa información específica del Servlet en la llamada HTTP
  - No son procesos independientes
  - Se ejecutan en la máquina virtual java. En el servidor Web

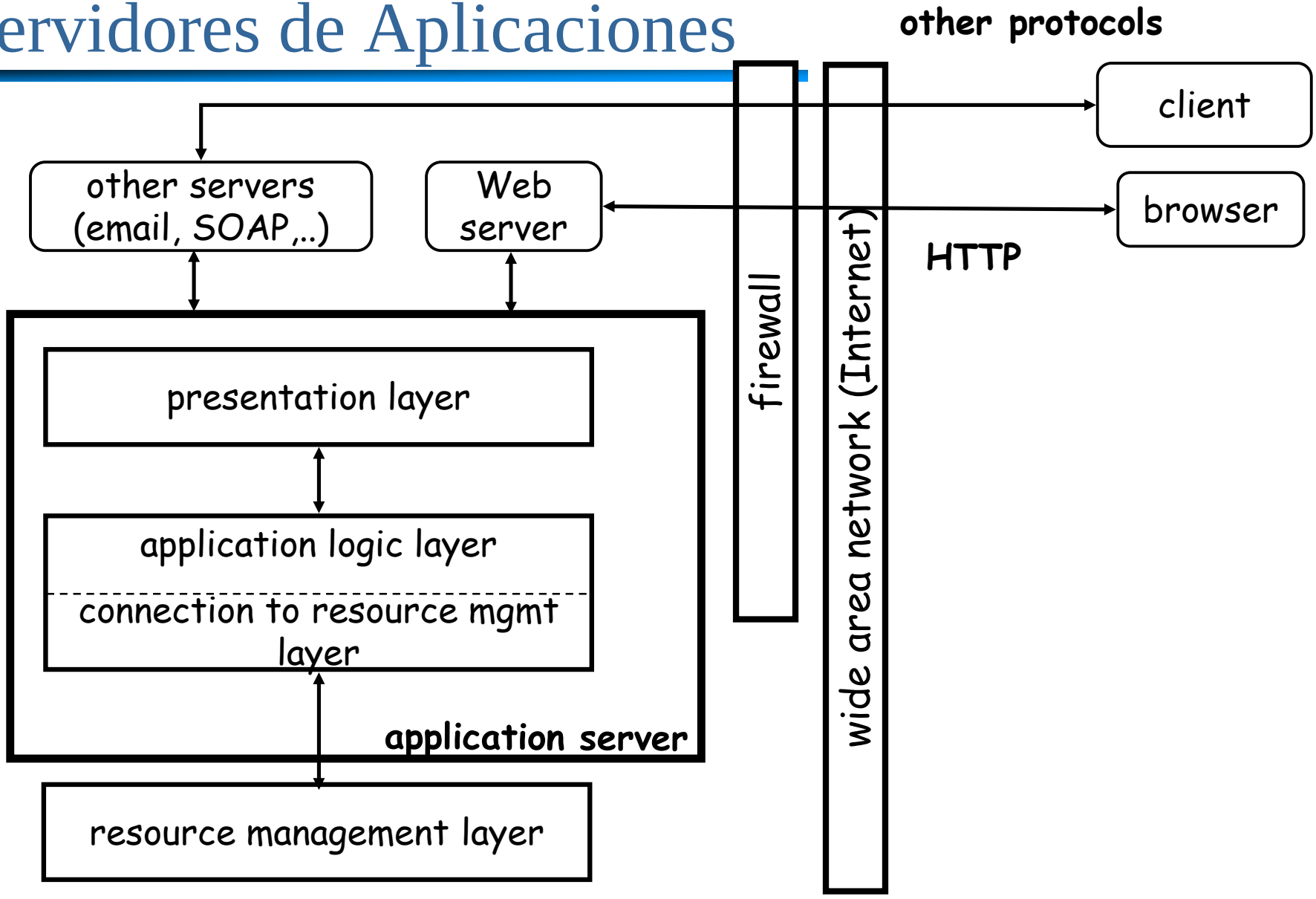
Servlets – Ventajas sobre CGI

- Tiempo de respuesta
  - No hay que crear un proceso por cada invocación
  - No hay cambio de contexto por parte del sistema operativo
- Escalabilidad
  - Múltiples peticiones al mismo Servlet → Una única imagen de programa
- Optimizaciones (contexto persistente de ejecución → Java server process)
  - Almacenar el resultado de una petición → Si otros clientes hacen la misma petición no hay que volver a ejecutarla
  - Compartir conexiones a la BBDD
  - Tracking de la sesión

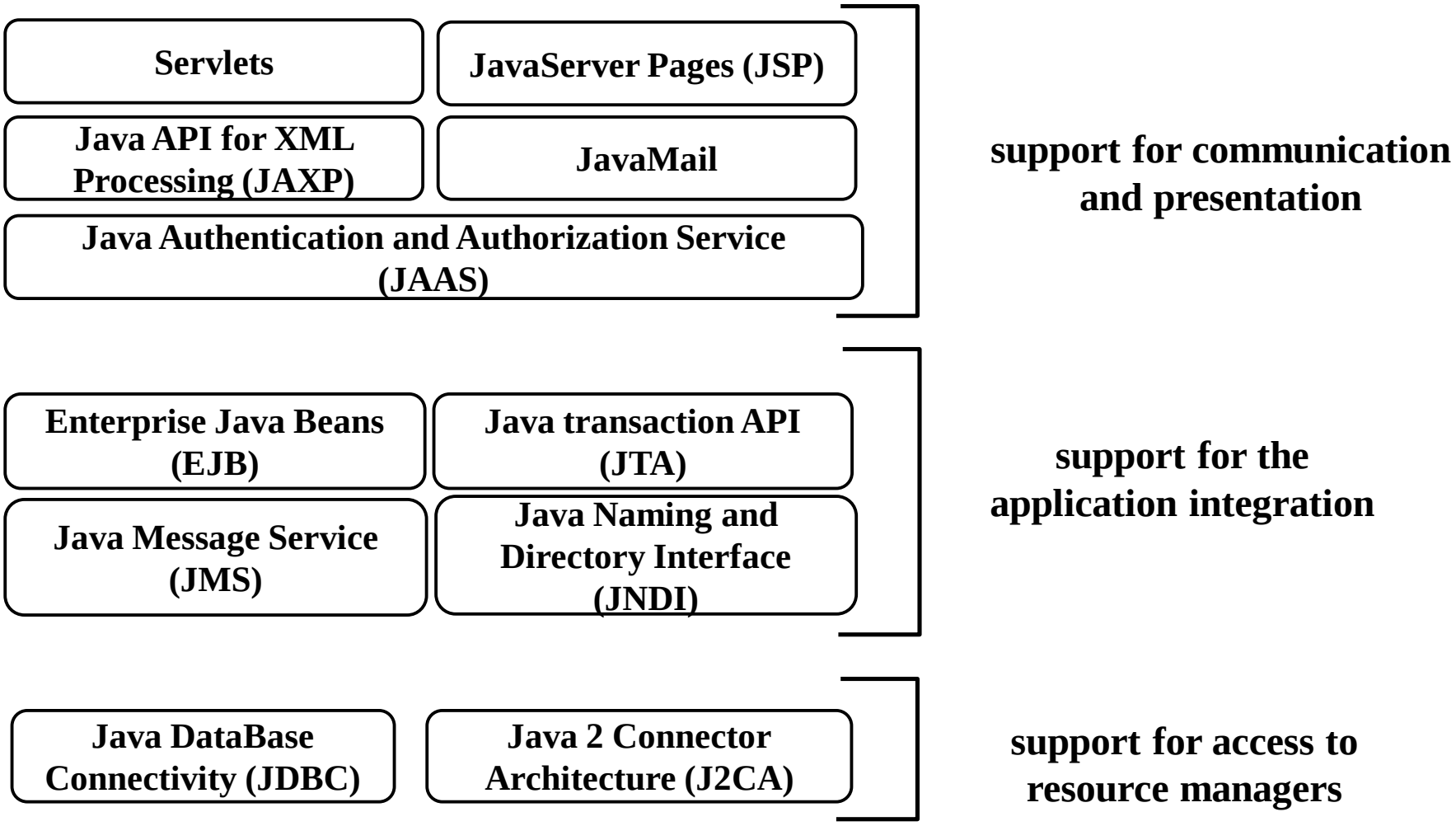
Servidores de Aplicaciones

- El uso de la Web como medio de acceso a los Sistemas de Información fuerza a las **plataformas Middleware** a dar soporte para acceder al Sistema de Información a través de la Web
- Por tanto los servidores de aplicaciones son las antiguas plataformas Middleware actualizadas con un canal de acceso a la Web
- Ejemplos: .NET, J2EE
- La consecuencia es que la capa de Presentación adquiere mayor relevancia
  - Lo que "extiende" el SA es la preparación, la generación dinámica y la gestión de los documentos
  - Se hace "juntando" la capa de Presentación que proporciona la Web con la capa de la Lógica de la Aplicación que proporciona la plataforma Middleware
  - Se simplifica la gestión de la aplicación Web y se hace eficiente la entrega de contenido a través de la Web
- La conexión con la capa de Gestión de Recursos se sigue haciendo con APIs estándar (ODBC, JDBC)

Servidores de Aplicaciones



J2EE


Servlets
JavaServer Pages (JSP)
Java API for XML Processing (JAXP)
JavaMail
Java Authentication and Authorization Service (JAAS)
support for communication and presentation
Enterprise Java Beans (EJB)
Java transaction API (JTA)
Java Message Service (JMS)
Java Naming and Directory Interface (JNDI)
support for the application integration
Java DataBase Connectivity (JDBC)
Java 2 Connector Architecture (J2CA)
support for access to resource managers